模式识别作业二: 基于PCA的图像压缩

段尧　　20303053

实验目的

1. 熟悉并掌握主成分分析的基本原理
2. 学会应用主成分分析实现数据降维，并应用到图像压缩

实验内容

1. 按照主成分分析的原理实现 PCA 函数接口
2. 利用实现的 PCA 函数对图像数据进行压缩和重建
3. 利用实现的 PCA 函数对高维数据进行低维可视化

实验过程

1　实现 PCA 函数接口

任务说明:

> 实现一个自己的 PCA 函数。 PCA 函数的主要流程是：先对计算数据的协方差矩阵，然后在对协方差矩阵进行 SVD 分解，得到对应的特征值和特征向量

1.1　PCA 基本原理

主成分分析（PCA）是一种统计方法，用于通过正交变换将一组可能相关的变量转换为一组线性不相关的变量，这些变量称为主成分。PCA可以用于降维，数据压缩，特征抽取，以及在机器学习中识别数据的内在结构。

PCA的原理可以通过以下数学公式来说明:

1. **数据集的协方差矩阵**：首先，我们有一个数据集 X，其中包含了多个特征。协方差矩阵 C 用于衡量这些特征之间的关系，计算公式为：$C = \frac{1}{N-1}X^TX$ 其中，X^TX 是数据集X 的转置，N是样本的数量。
2. **特征值分解**：对协方差矩阵进行特征值分解：$C = W\Lambda W^T$，其中W是特征向量矩阵，Λ是对角矩阵，对角线上的元素是特征值。
3. **构造投影矩阵**：选择前k 个最大的特征值对应的特征向量，构造一个投影矩阵W_k： $W_k = [v_1, v_2, \ldots, v_k]$ 其中，v_i是第i个特征向量。
4. **数据转换**：最后，使用投影矩阵W 将原始数据集X转换到新的特征空间，得到降维后的数据集 $Z = XW_k$，其中W_k是由前k个特征向量构成的矩阵。

通过这个过程，PCA 能够将原始数据集转换为一组线性不相关的变量，这些变量按照方差的大小排序，保留了数据集中最重要的变化信息。降维后的数据集Y 可以用更少的变量来近似原始数据集，同时减少计算复杂度和存储需求。

有的时候我们需要在计算协方差矩阵之前进行数据中心化处理，公式为 $X_c = X - \mu$ ，之后计算中，使用 X_c 代替 X 即可。

在真正使用PCA时，我们不会像上述算法中一样计算特征向量，我们直接使用 SVD 奇异值分解方法计算，在 SVD 中，对于任意矩阵A，SVD 为:

$$A = U \cdot S \cdot V^T \tag{2}$$

其中:

- U 是 A 的左奇异矩阵（左特征向量），它的列向量是 $A \cdot A^T$ 的特征向量，且是标准正交的。
- S 是一个对角矩阵，其对角线上的元素称为奇异值，表示了 A 的重要性或贡献程度。奇异值按照降序排列，即 $S_{11} \geq S_{22} \geq \cdots \geq S_{nn}$，$S$ 的零元素除了对角线元素之外都为零。
- V^T 是A 的右奇异矩阵（右特征向量），它的行向量是$A^T \cdot A$ 的特征向量，同样也是标准正交的。

1.2 代码实现

代码实现如下：

```
def pca(X, num_components):
    # Step 1: Mean centering the data
    X_meaned = X - np.mean(X, axis=0)
    mean = np.mean(X, axis=0)

    # Step 2: Compute the covariance matrix of the data
    covariance_matrix = np.cov(X_meaned, rowvar=False)

    # Step 3: Compute the SVD of the covariance matrix
    U, S, Vt = np.linalg.svd(covariance_matrix)

    # Step 4: Select the top 'num_components' principal components
    components = Vt[:num_components]

    # Step 5: Project the data onto the principal components
    X_reduced = np.dot(X_meaned, components.T)

    # Step 6: Compute explained variance
    explained_variance = S[:num_components] / np.sum(S)

    # Step 7: Restore the data from the reduced dimension
    X_restored = np.dot(X_reduced, components) + mean

    return X_reduced, components, explained_variance, X_restored
```

这个函数中，有四个返回值，含义分别为 原始数据投影到主成分张成空间中的值（也就是降维数据），主成分，计算方差（这个量可以用于衡量降维效果的好坏），重建数据。

2 Eigen_faces

任务说明：

> 利用实现的 PCA 函数，对 Eigen Face 数据集中的灰度人脸数据进行压缩和重建。数据位于 data/faces.mat，数据如下图所示。利用 PCA 对这些人脸图像进行主成分分析，展示前 49 个的主成分，将结果保存为results/PCA/eigen_faces.jpg。然后采用 PCA 对这些人脸数据降维到不同维度 (10， 50, 100, 150)进行压缩，然后再重建，对比不同的压缩和重建效果，将结果保存为 results/PCA/recovered_faces_top_xxx.jpg。实验报告中要有压缩前，和不同压缩程度的结果结果对比。

2.1 代码说明

首先有两个辅助绘图函数：

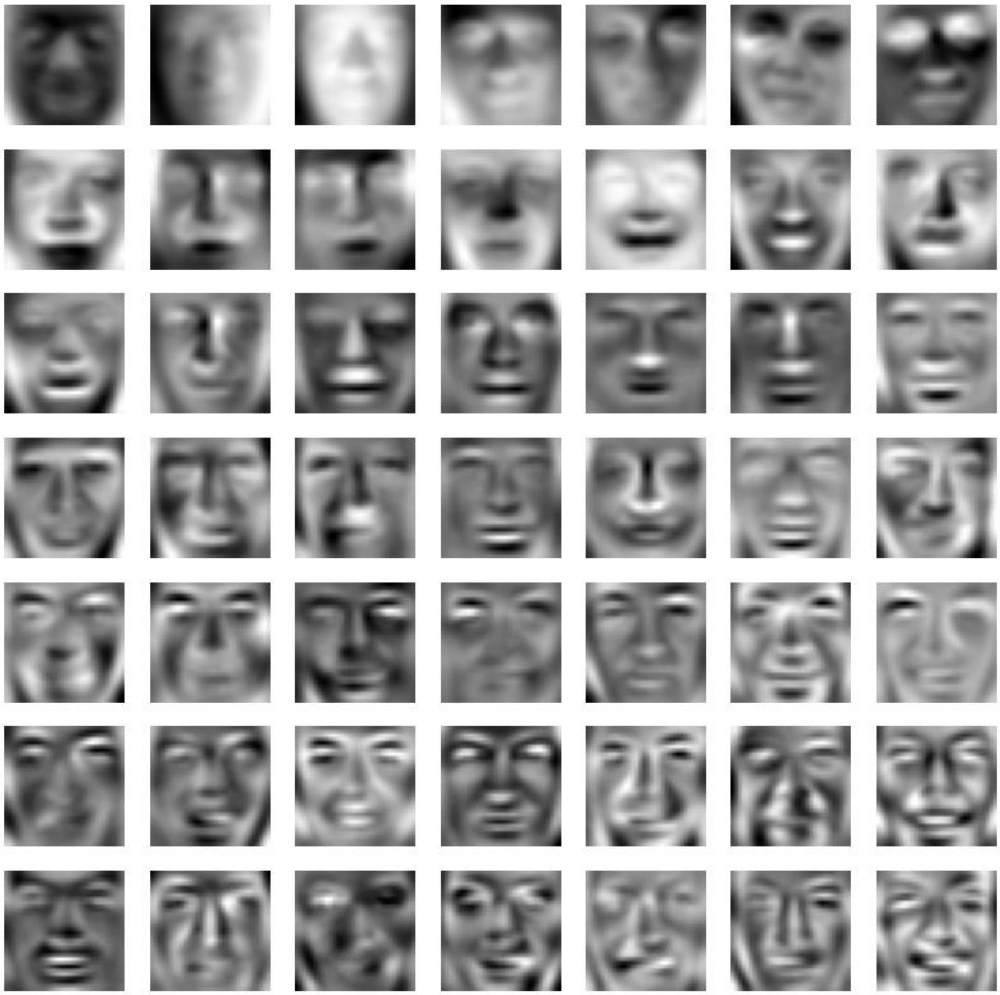
```
def plot_faces(images, num_rows, num_cols, output_path): # 绘制一批人脸图像
def plot_eigen_faces(components, output_path):  # 绘制特征脸，也就是保存前49个主成分的可视化结果
```

这两个函数并不是主要函数，不再赘述。任务书中任务a与任务b(1) 被集成在代码中的 `task_1_2()` 函数中，代码如下：

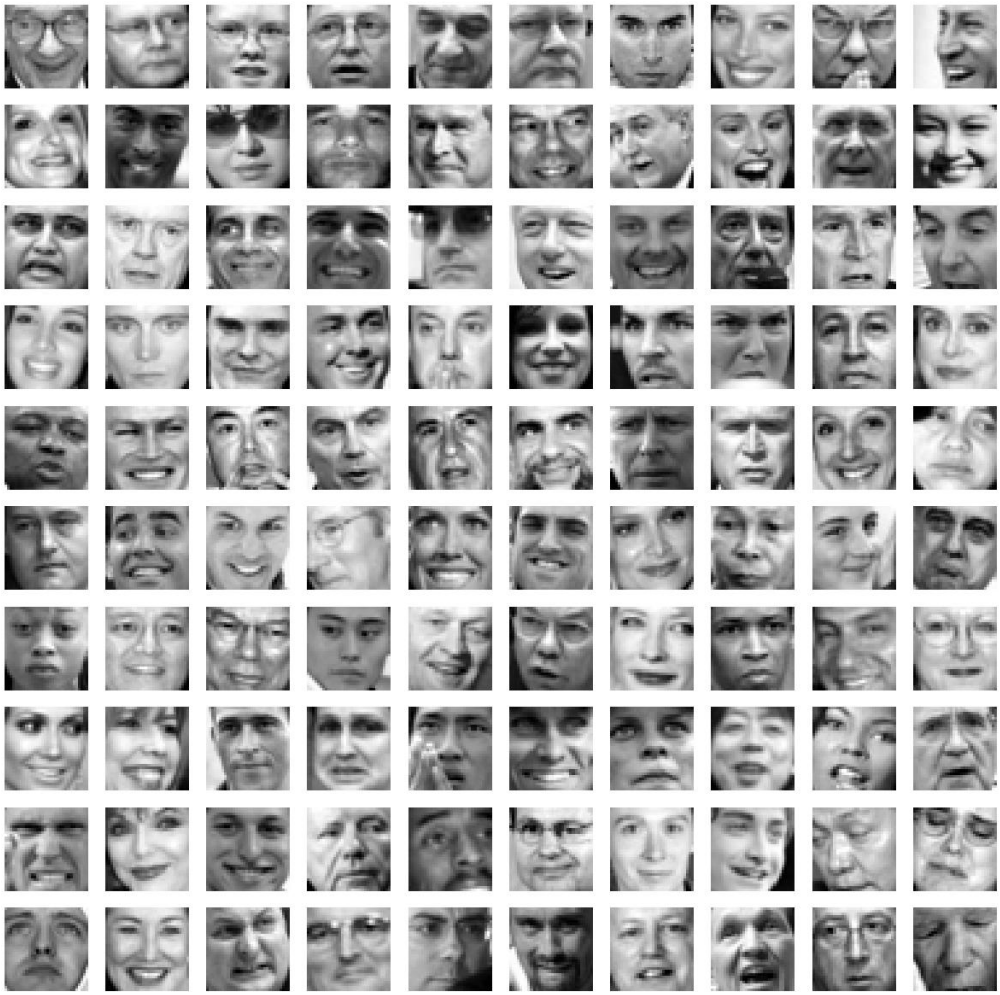
```
def task_1_2():
    # Ensure the directories exist
    if not os.path.exists('../results/PCA'):
        os.makedirs('../results/PCA')

    # Load data
    mat_data = sio.loadmat('../data/faces.mat')
    X = mat_data['X']

    # Only take the first 1024 columns for processing
    X = X[:, :1024]

    # Save the original 100 faces (reshape to 32x32 for visualization)
    original_faces = X[:100].reshape(-1, 32, 32)
    plot_faces(original_faces, num_rows=10, num_cols=10,
output_path='../results/PCA/original_faces.jpg')

    # Perform PCA on all 5000 faces
    dimensions = [10, 50, 100, 150]

    for dim in dimensions:
        _, _, _, X_restored = pca(X, num_components=dim)
        # Extract the restored first 100 faces and reshape for visualization
        X_restored_faces = X_restored[:100].reshape(-1, 32, 32)
        plot_faces(X_restored_faces, num_rows=10, num_cols=10,
                   output_path=f'../results/PCA/recovered_faces_top_{dim}.jpg')

    print("Processing complete. Check the results in the '../results/PCA' directory.")
```

2.2 结果展示

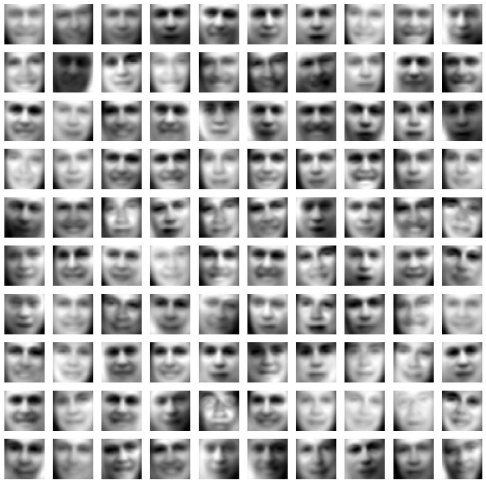
首先展示提取的49个主成分可视化：



原始前100张人脸如下所示：

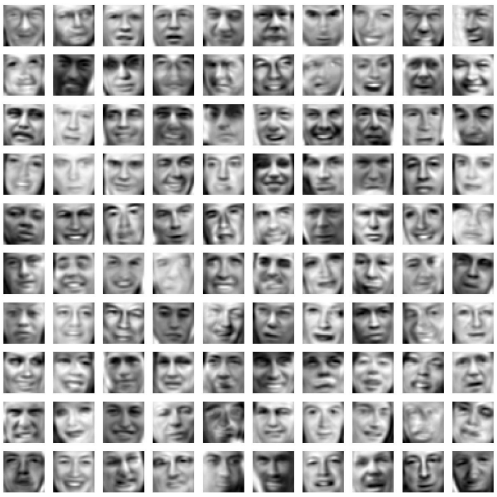


降维重建后：

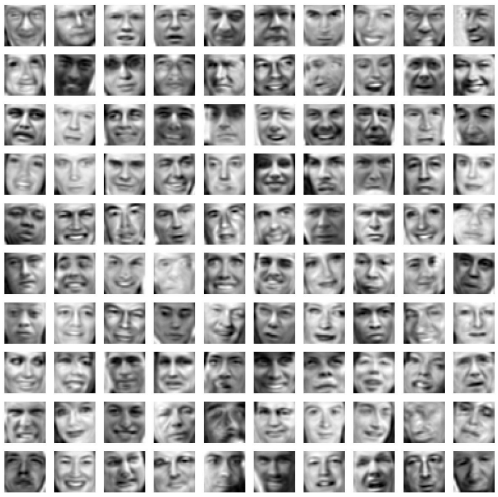


recovered_faces_top_10

recovered_faces_top_50



recovered_faces_top_100



recovered_faces_top_150

从上述结果中可以看出：

- 实际上仅仅用前10维数据也能够看出来是一张人脸，这说明在人脸检测等任务里面，完全可以进行大幅度的数据降维，而几乎不损失精度。
- 到50维重建时，已经几乎可以区分人了，这时候损失了一部分人的独特特征，可能对于人脸识别会有损失，但仅以人的身份观察，还是可以作为区分标准的。
- 100维重建结果除了损失了一小部分细节信息，也就是稍微糊一点，结果其实已经相当清晰。
- 到150维重建，肉眼已经很难分辨与原图的细节差距了。
- 也就是说，实际上，图像中的主要信息仍存储在最靠前的成分中，在不损失或者损失少量精度下，大幅度的数据降维，对于许多规模较大任务，可以预见，是极为重要的。

3 Scenery

任务说明：

利用实现的 PCA 函数，对 scenery.jpg 彩色 RGB 图进行压缩和重建。数据位于 data/scenery.jpg，对该图片分布降维到不同维度(10， 50, 100, 150)进行压缩，然后再重建，对比不同的压缩和重建效果。将结果保存为results/PCA/recovered_scenery_top_xxx.jpg。实验报告中要有压缩前，和不同压缩程度的结果结果对比。

3.1 代码说明

这部分代码我作为 `task_3()` 写成一个单独的函数方便调用，代码如下所示：

```
def task_3():
    from PIL import Image

    # Load the colored image
    image_path = '../data/scenery.jpg'
    image = Image.open(image_path)
    image_data = np.array(image)

    # Get image dimensions
    height, width, channels = image_data.shape

    # Perform PCA on each channel separately
    dimensions = [10, 50, 100, 150]
    for dim in dimensions:
        restored_channels = []

        for channel in range(channels):
            # Extract the channel data
            channel_data = image_data[:, :, channel]
            reshaped_data = channel_data.reshape(height, -1)

            # Perform PCA on the channel data
            X_reduced, components, _, X_restored = pca(reshaped_data, num_components=dim)

            # Reshape the restored channel data
            restored_channel = X_restored.reshape(height, width)
            restored_channels.append(restored_channel)

        # Stack the restored channels back into an image
        restored_image_data = np.stack(restored_channels, axis=2)
        restored_image_data = np.clip(restored_image_data, 0, 255).astype(np.uint8)  # Ensure
values are within valid range

        # Save the restored image
        output_path = f'../results/PCA/recovered_scenery_top_{dim}.jpg'
        restored_image = Image.fromarray(restored_image_data)
        restored_image.save(output_path)

    print("Scenery image compression and restoration complete. Check the results in the
'../results/PCA' directory.")
```

上述代码整体思路和之前处理灰度人脸是一致的，但是有一个很关键的点（在后面问题解决中第五点也有说明）是我这里**将彩色图像的每一个通道分别处理，之后组合，单通道PCA时，将每一行当作一个样本，将每一列当作一个特征**。另外，我认为将通道分开处理，有附加好处：可以保证色彩信息不丢失，否则处理出来的色彩应该会很奇怪。

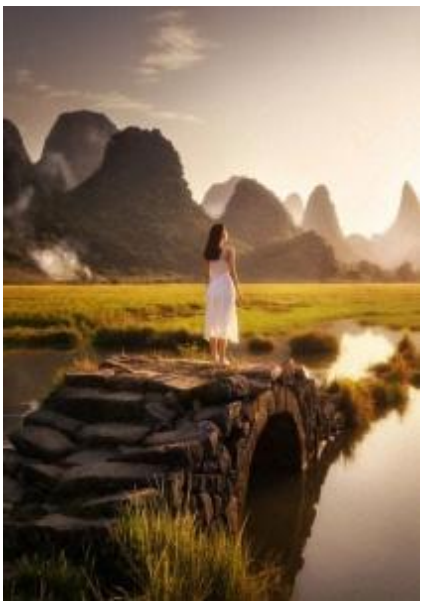
3.2 结果展示

对比结果展示如下：

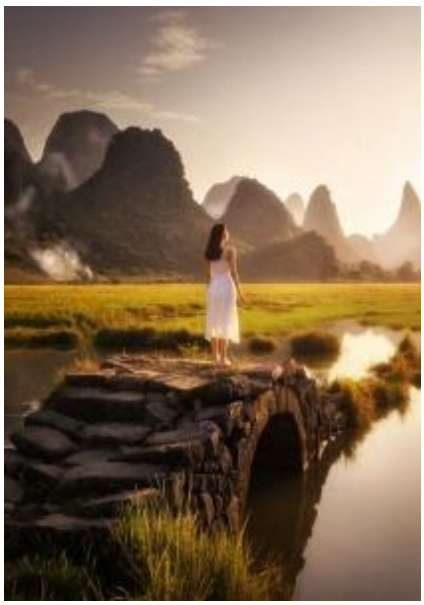
scenery

recovered_scenery_top_10

recovered_scenery_top_50



recovered_scenery_top_100



recovered_scenery_top_150

降维10维后重建图像还是比较模糊的，会出现一些鬼影现象，但是能基本分辨具体有什么内容，降维50及更高维度后重建，基本上已经看不出什么区别了。

实验总结与问题解决

1 实验总结

通过本次实验，我对PCA（Principal Component Analysis）的原理和应用有了更加深入的理解，并掌握了如何实现PCA函数及其在图像压缩和数据降维中的应用。实验内容包括实现PCA函数、应用PCA对人脸数据进行压缩和重建、以及对彩色图像进行压缩和重建，取得了预期效果。

一些个人**启示性总结**：

- 处理高维数据时，需要对数据进行中心化和协方差矩阵计算。通过对数据进行减均值操作，确保了数据的中心化。

- 在SVD分解中，确保数据的维度正确，避免了数据维度不匹配的问题。
- 在对彩色图像进行处理时，分别对每个通道进行PCA压缩和重建，确保了图像的色彩信息不丢失。
- PCA在降维和特征提取中非常有效，尤其是在高维数据中，可以显著减少数据维度，同时保留大部分重要信息。
- 对于不同的图像数据，应根据实际需求选择适当的压缩维度，以平衡压缩效果和重建质量。
- 对于大规模数据集，计算协方差矩阵和进行SVD分解的计算复杂度较高，需要考虑性能优化和高效算法的应用。

2 问题解决与注意点说明

1. `mat` **数据的处理**：
    - Matlab中的`.mat`文件读取后会成为一个字典，需要通过键来获取数据。确保在处理数据时，正确读取和使用键值对。一般我们需要使用`matfile.keys()`打印出来看一下具体是什么键。
2. **图像旋转问题**：
    - 在处理图像数据时，图像可能会出现旋转现象。通过调整数据的转置操作，确保图像在可视化时的正确显示。图像输出结果看起来像是逆时针旋转了九十度，这个是因为预处理数据时做了转置，所以处理结束进行可视化的时候需要将重建数据重新转置一次，就可以显示正常了，因为人脸数据看起来转置和逆时针转九十度很像，不好区分。
3. **特征提取与主成分选择**：
    - 在提取主成分时，应使用整个数据集进行特征值分解，以确保主成分的代表性和重建效果。需要注意，我们在展示的时候展示前100张人脸的重建结果，但是取主成分的时候，需要用整个数据集取，否则效果会非常糟糕。
4. **SVD分解错误处理**：
    - 调用`numpy.SVD`时，如果出现"0-dimension"错误，通常是由于数据被扁平化为一维数组。应确保数据的正确形状，避免错误发生。
5. **彩色图像处理策略**：
    - 在彩色图像处理中，可以选择将每个像素作为一个样本，将RGB三个通道值作为特征。这种方式可以保留更多色彩信息，但计算复杂度较高。
    - 最终选择将每个通道单独处理，将一列作为一个特征，将一行作为一个样本，通过逐通道PCA处理并重建，保证了计算的可行性和图像质量。
    - 在彩色图像处理中，出现了“0-dimension”问题，结合群内同学的讨论，我刚开始尝试“**将一个像素视为一个样本，将RGB三通道值作为特征**”，个人认为这样更加合理，但是这个数组形状就变成了630000*3，转置之后为3*630000计算非常困难，所以之后我又重新更改为**将每个通道单独处理，将一列作为一个特征，将一行作为一个样本**。

参考文献

[1] OpenCV：SIFT（尺度不变特征变换）简介

[2] OpenCV 特征检测

[3] Random sample consensus - Wikipedia