模式识别第二次作业：基于 PCA 的图像压缩

21307174 刘俊杰

May 2024

1 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

2 实验要求

1．提交实验报告，要求有适当步骤说明和结果分析
2．将代码和结果打包提交
3．不能直接调用现有的库函数提供的 PCA 接口

3 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

4 实验过程

4.1 实现 PCA 函数接口

实现一个你自己的 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。

PCA 的算法步骤：

设有 m 条 n 维数据.

1）将原始数据按列组成 n 行 m 列矩阵 X

2）将 X 的每一行（代表一个特征）进行零均值化，即减去这一行的均值

3）求出协方差矩阵 $C = \frac{1}{m}XX^{\mathsf{T}}$

4）求出协方差矩阵 C 的特征值及对应的特征向量

5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 P

6)Y = PX 即为降维到 k 维后的数据

实现的 PCA 函数接口：

```
def PCA(data, dim=10):
    # 数据中心化
    data_mean = np.mean(data, axis=0)
    centered_data = data - data_mean
    # 计算协方差矩阵
    cov_matrix = np.cov(centered_data, rowvar=False)
    # SVD 分解
    U, S, Vt = np.linalg.svd(cov_matrix)
    # 选择前 dim 个特征向量
    P = Vt[:dim]
    # 转换数据
    Y = np.dot(centered_data, P.T)
    #print(centered_data.shape,P.shape,Y.shape)
    return Y, P, Vt
```

4.2 PCA 的基本应用

4.2.1 PCA 灰度人脸数据进行压缩和重建

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。

4.2.1.1 实现思路

首先使用 scipy.io.loadmat 读取数据，使用 PCA 函数接口计算出前 49 张图像的主成分。再提取原始图像方便后续对比重建结果。使用 PCA 对原始数据进行压缩，再通过图像重建比较不同维度降维重建的效果。

4.2.1.2 实现代码

读取 mat 数据，并提取主成分并展示：

```
# 读取数据
data = scipy.io.loadmat("data/faces.mat")
faces = data['X']

# 展示前 49 个主成分
fig, axes = plt.subplots(7, 7, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
    data = faces[i].reshape(32,32).T
    _,_,Vt = PCA(data) # PCA获取主成分
    ax.imshow(Vt, cmap='gray')
    ax.axis('off')
plt.suptitle('First 49 Principal Components', fontsize=16)
plt.show()
plt.close()
```

```
fig.savefig("results/PCA/eigen_faces.jpg")
```

提取原图展示并保存：

```
# 展示前 100 张原图
fig, axes = plt.subplots(10, 10, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
    data = faces[i].reshape(32,32).T
    ax.imshow(data, cmap='gray')
    ax.axis('off')
plt.suptitle('First 100 Original faces', fontsize=16)
plt.show()
plt.close()
fig.savefig("results/PCA/first_100_original_faces.jpg")
```

对原图进行 PCA 压缩并重建图像：

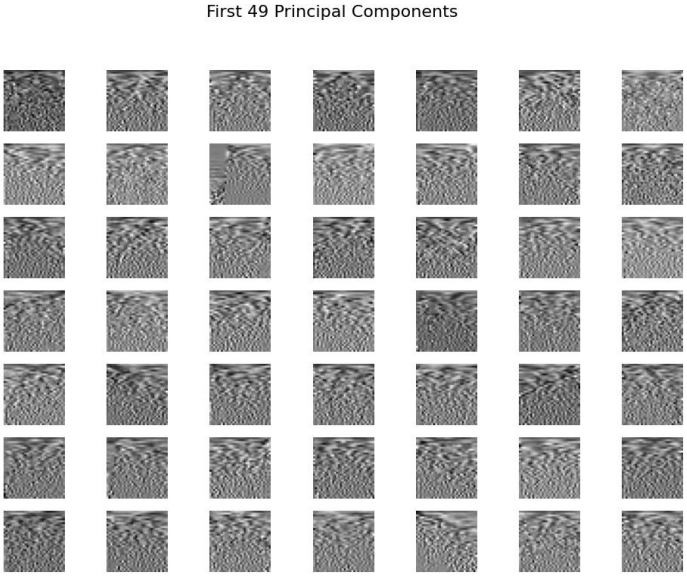
```
# 降维维度
compression_levels = [10, 50, 100, 150]

# 对原图进行PCA降维重建
for level in compression_levels:
    # 执行 PCA
    compressed_data, components,Vt = PCA(faces,level)
    # 重建图像
    reconstructed_data = np.dot(compressed_data, components)
    reconstructed_image_array = (reconstructed_data + np.mean(faces, axis=0))
    #reconstructed_image = Image.fromarray(reconstructed_image_array.astype(np.uint8))

    # 展示前 100 张原图PCA降维重建结果
    fig, axes = plt.subplots(10, 10, figsize=(12, 12))
    for i, ax in enumerate(axes.flat):
        data = reconstructed_image_array[i].reshape(32,32).T
        ax.imshow(data, cmap='gray')
        ax.axis('off')
    plt.suptitle(f'recovered_faces_top_{level}', fontsize=16)
```

```
20      plt.show()
21      plt.close()
22      fig.savefig(f"results/PCA/recovered_faces_top_{level}.jpg")
```

4.2.1.3 **实验结果**

前 49 张图像的主成分：

由于 5000 张图像难以一起展示对比，故选择前 100 张图像展示。



原图

前 100 张原始图像：

不同维度降维重建效果（具体可见 results 文件夹中）：

4.2.1.4 **实验结果分析**

可以看到随着维度的增多，人脸图像的细节复原得更多，能够较好地保留原始图像的特征和细节。

原图

4.2.2 PCA 对彩色 RGB 图进行压缩和重建

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。

4.2.2.1 实现思路

对彩色图像的 3 个通道分别进行 PCA 压缩和重建,3 个通道 PCA 压缩和重建完成后再将 3 个通道合并在一起。

4.2.2.2 实现代码

```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

降维维度为 10 的重建结果

降维维度为 50 的重建结果

降维维度为 100 的重建结果

降维维度为 150 的重建结果

```
from PIL import Image
import os
from PCA import PCA
# 加载图像
image_path = "data\scenery.jpg"
image = Image.open(image_path)
data = np.array(image)

# 降维到不同维度
compression_levels = [10, 50, 100, 150]

for level in compression_levels:
    # 不同通道重建结果
    reconstructed_channels = []
    # 对不同通道进行PCA重建
    for channel in range(data.shape[2]):
        channel_data = data[:,:, channel]
        # 执行 PCA
        compressed_data, components,Vt= PCA(channel_data, level)

        # 重建图像
        reconstructed_data = np.dot(compressed_data, components)
        reconstructed_image_array = (reconstructed_data + np.mean(channel_data, axis=0))
        reconstructed_channels.append(reconstructed_image_array)

    # 将三个通道的数据合并为重建的彩色图像
    reconstructed_image = np.stack(reconstructed_channels, axis=-1)
    reconstructed_image = Image.fromarray(reconstructed_image.astype(np.uint8))

    # 可视化原始图像和重建图像
    fig, axes = plt.subplots(1,2, figsize=(18, 6))
    axes[0].imshow(image)
    axes[0].set_title('Original Image')
```

```
36      axes [0]. axis ('off')
37      axes [1]. imshow(reconstructed_image)
38      axes [1]. set_title (f'Reconstructed Image (Top {level} components)')
39      axes [1]. axis ('off')
40      plt .show()
41      # 保存重建图像
42      reconstructed_image.save(f"results/PCA/recovered_scenery_top_{level}.jpg")
43      fig . savefig (f"results/PCA/recovered_scenery_top_{level}_comparison.jpg")
44      plt . close ( fig )
```

4.2.2.3 **实验结果**

原图

比较不同降维维度对图像重建的效果的影响：

降维维度为 10 的重建结果

降维维度为 50 的重建结果

降维维度为 100 的重建结果

降维维度为 150 的重建结果

4.2.2.4 实验结果分析

降维维度决定了保留的主要特征数量。在本实验中，选择了 10、50、100 和 150 作为降维维度，以观察不同数量的主成分对于图像重建的影响。

当降维维度较低时，例如 10 我们观察到重建的图像失去了一些细节和清晰度。这是因为较低数量的主成分无法捕获原始数据的全部变化，导致了信息的丢失。

随着降维维度的增加，重建的图像变得更加清晰，更接近原始图像。特别是在 50 或 100 或 150 的降维维度下，虽然有一些噪声，但重建的图像已经能够较好地保留原始图像的特征和细节。

通过这个实验结果对比，可以清楚地观察到降维维度对于图像重建的影响。选择适当的降维维度可以在减少数据维度的同时，尽可能地保留原始数据的特征，从而实现数据压缩和降维的目的。