《数据科学与工程算法》项目报告

报告题目：基于主成分分析的图像压缩技术实现

姓　　名：温兆和

学　　号：10205501432

完成日期：2023.05.17

摘要 [中文]：

主成分分析可用于对图片进行压缩。在本次实验中，我们将一百张图片转化为矩阵，并选取不同的主成分个数，用主成分分析技术压缩图片，并计算压缩时间、压缩率和重构误差等性能指标。

Abstract [English]

PCA can be used to compress images. In this experiment, we convert 100 images into matrix, calculate its covariance matrix, compress the images with PCA by the principle component number we choose and illustrate its performance by compression time, compression rate and reconstruction error.

一、 项目概述（阐明项目的科学价值与相关研究工作，描述项目主要内容）

1.1 科学价值

图片是生活中常见的高维数据，在保证图片清晰度不受太大影响的前提下对图片进行压缩可以大大地节省内存空间。PCA 技术通过计算矩阵的协方差矩阵的特征值来选取原矩阵的主成分，可以在降低矩阵维度的情况下最大程度地保留原矩阵的信息，而图片又可以表示为矩阵，所以 PCA 技术显然可以实现在保证清晰度的前提下压缩图片这一目标。

1.2 相关研究工作

目前，在传统 PCA 的基础上，还发展出了鲁棒 PCA、稀疏 PCA、核 PCA 等 PCA 变体，它们或者减少了算法的运行时间，或者降低了 PCA 的重构误差。

在 PCA 技术中，还涉及到求矩阵的协方差矩阵的特征值和特征向量。在计算机中，我们常合用幂法和平移法或者降价技术来求解矩阵的特征值和特征向量。将这个算法用代码实现出来往往费时费力。目前，Peter J. Olver 提出了用 QR 分解来计算矩阵特征值和特征向量的方法[1]，这在代码实现上比幂法要方便不少。

1.3 项目主要内容

在本次实验中，我们先逐一读取 100 张图片，将三维的图片表示为二维的矩阵，计算矩阵的协方差矩阵，求出协方差矩阵的特征值和特征向量并根据主成分个数k保留前k大的特征值对应的特征向量，并与原来的矩阵相乘。

随后，我们用 PCA 算法得到的矩阵对图片进行重构，并计算压缩时间、压缩率和重构误差等性能指标，对所选主成分个数的好坏进行评判。

二、 问题定义（提供问题定义的语言描述与数学形式）

2.1 语言描述

对于某一张图片，我们要在保证清晰度的前提下得到其占用空间较小的存储方式。

2.2 数学形式

对于某个矩阵，我们要算出它的协方差矩阵，计算协方差矩阵的特征向量和特征值，并保留最大的若干个特征值对应的特征向量。

三、 方法（问题解决步骤和实现细节）

3.1 数据集的预处理

在本次实验中，数据集包含 100 张大小为 256*256*3 的图片。为了节省内存空间，我们只保存每张图片在计算机中的存储路径，在压缩某一张图片的时候才将根据其路径将其读取出来。

```
route = "D:/数据科学与工程算法/10205501432_温兆和_数据科学算法实验
2/Images/beach/beach"
ROUTE = []
for i in range (100):
    if (i<10):
```

[1] Peter J. Olver. Orthogonal Bases and the QR Algorithm

```
        ROUTE.append(route+str(0)+str(i)+".tif")
    else:
        ROUTE.append(route+str(i)+".tif")
```

图片是三维数据，我们需要将其转化为二维矩阵才能进行主成分分析。在本次实验中，我们分别将图片的 R、G、B 三个维度存进三个 256*256 的矩阵中，再将这三个矩阵合并成$(R\ G\ B)$的形式，形成一个大小为 256*768 的矩阵。

```
#把三维的图片转化为二维的矩阵
Red = np.zeros((img0.shape[0],img0.shape[1]))
Green = np.zeros((img0.shape[0],img0.shape[1]))
Blue = np.zeros((img0.shape[0],img0.shape[1]))
for i in range (img0.shape[0]):
    for j in range (img0.shape[1]):
        Red[i][j]=img0[i][j][0]
        Green[i][j]=img0[i][j][1]
        Blue[i][j]=img0[i][j][2]
img1 = np.hstack((Red,Green,Blue))
```

3.2 用 PCA 选取矩阵的主成分

在 PCA 算法中，首先要求原矩阵的协方差矩阵。首先，求出原矩阵每一列的均值、方差，对原矩阵的每一列进行中心化、标准化：

```
#中心化、标准化
means = np.mean(img1, axis=0).reshape(1, -1)
stvr = np.std(img1, axis=0).reshape(1, -1)
img2 = (img1 - means) / stvr
```

然后，用中心化、标准化之后的矩阵求出协方差矩阵，并求协方差矩阵的特征值和特征向量：

```
#求出协方差矩阵并对其进行主成分分析
S = img2.T@img2
W, Q = eig_by_hand(S)
```

这里，我们用 QR 算法来求解矩阵的特征值和特征向量。具体的做法是，对矩阵 A 进行 QR 分解得到 Q 和 R，再用 R 左乘 Q 得到一个新的矩阵，再对这个新的矩阵进行 QR 分解。当得到的矩阵不再发生较大的变化，就认为其对角元为矩阵 A 的特征值。将每一步得到的 Q 矩阵按照$Q_1Q_2\ldots\ldots Q_k$的顺序乘在一起，就能得到原矩阵 A 的特征向量。

```
def eig_by_hand(M,max_iteration = 100,epsilon = 1e-3):
    A = M
    S = np.eye(len(M))
    for I in range (max_iteration):
        Q,R = np.linalg.qr(A)
        S = S@Q
        A_1 = R@Q
        if (np.linalg.norm(A_1-A,ord = np.inf)<=epsilon):
            A = A_1
            break
        A = A_1
```

```
    ev = []
    for i in range(len(M)):
        ev.append(A[i][i])
    eigenvalue = np.array(ev)
    return eigenvalue,S
```

最后，对特征值和特征向量进行排序，并根据事先确定好的主成分个数保留前 k 大的主成分对应的特征向量。最后，将原矩阵与保留下来的特征向量矩阵相乘，得到原图片矩阵的 k 个主成分。

```
EV_sorted = np.flip(np.argsort(W))
Q = Q[:, EV_sorted]
W = W[EV_sorted]
#根据主成分个数 k 保存前 k 个特征向量，并与原矩阵作乘积
C = img2@Q
C_to_save = C[:, :k]
Q_to_save = Q.T[:k, :]
```

3.3 对图片进行重构，并计算重构性能

首先，对图片进行重构。重构的方式是计算我们在 3.2 中得到的前 k 个主成分和前 k 个特征向量的乘积。需要注意的是，由于我们在 3.1 中进行过中心化、标准化，这里我们还要将 3.1 中得到的标准差乘回去、把得到的均值加回去。

```
img3 = np.matmul(C_to_save, Q_to_save)
img3 = img3 * stvr + means
```

然后，把 256*768 维的矩阵重新表示成 256*256*3 维的图片：

```
#把的矩阵转化为三维的图片
img4=np.zeros((img0.shape[0],img0.shape[1],3),dtype = int)
for i in range (img0.shape[0]):
    for j in range (img0.shape[1]):
        img4[i][j][0]=math.floor(img3[i][j])
        img4[i][j][1]=math.floor(img3[i][j+img0.shape[1]])
        img4[i][j][2]=math.floor(img3[i][j+img0.shape[1]*2])
```

最后，我们还要计算算法的性能。由于我们实际上秩序存储前 k 个主成分和前 k 个特征向量，压缩率的计算方式是用前 k 个主成分和前 k 个特征向量大小之和比上原图片的大小（256*256*3）：

```
compact_rate =
(C_to_save.shape[0]*C_to_save.shape[1]+Q_to_save.shape[0]*Q_to_save.s
hape[1])/(img0.shape[0]*img0.shape[1]*img0.shape[2])
```

重构误差是重构后得到的图片矩阵和原有图片矩阵之差的 F 范数与原图片矩阵的 F 范数的比值：

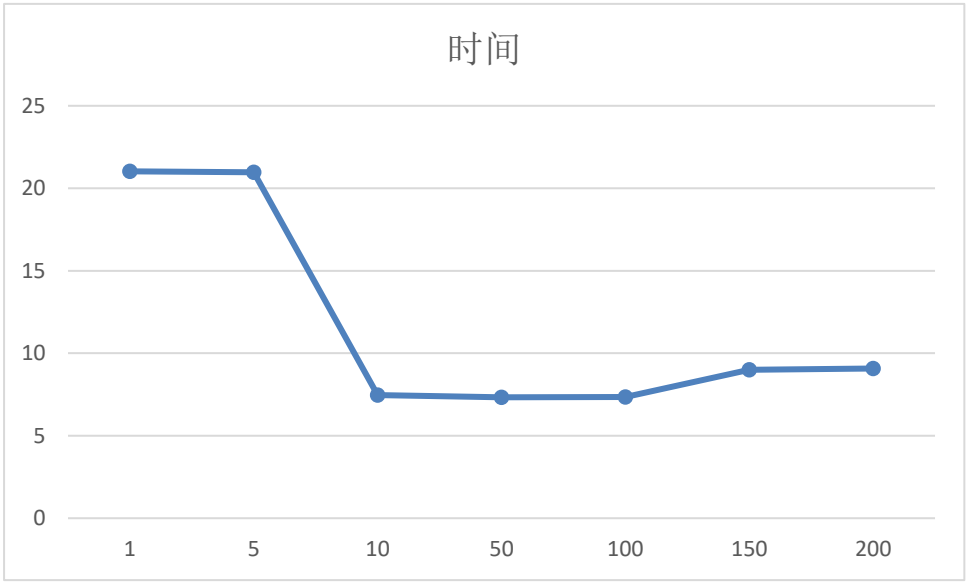
```
reconstruction_error = np.linalg.norm(img3-img1)/np.linalg.norm(img1)
```

压缩时间则是压缩开始前测得的时间点和算法结束后测得的时间点之差。

四、 实验结果（验证提出方法的有效性和高效性）

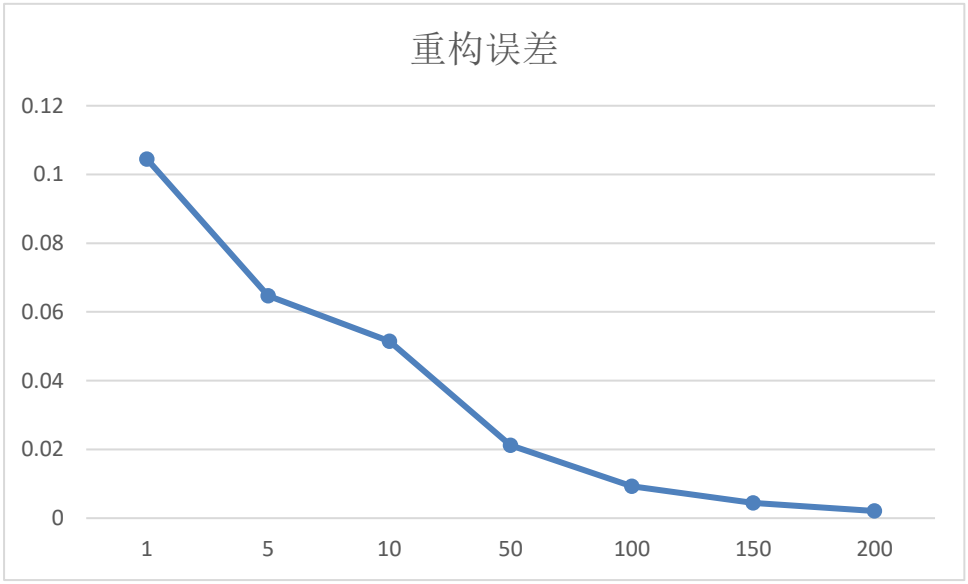
在本次实验中，我们分别选取了 1、5、10、50、100、150、200 这几个主成分个数。接下来，分别展示各主成分个数下的性能指标。

时间：



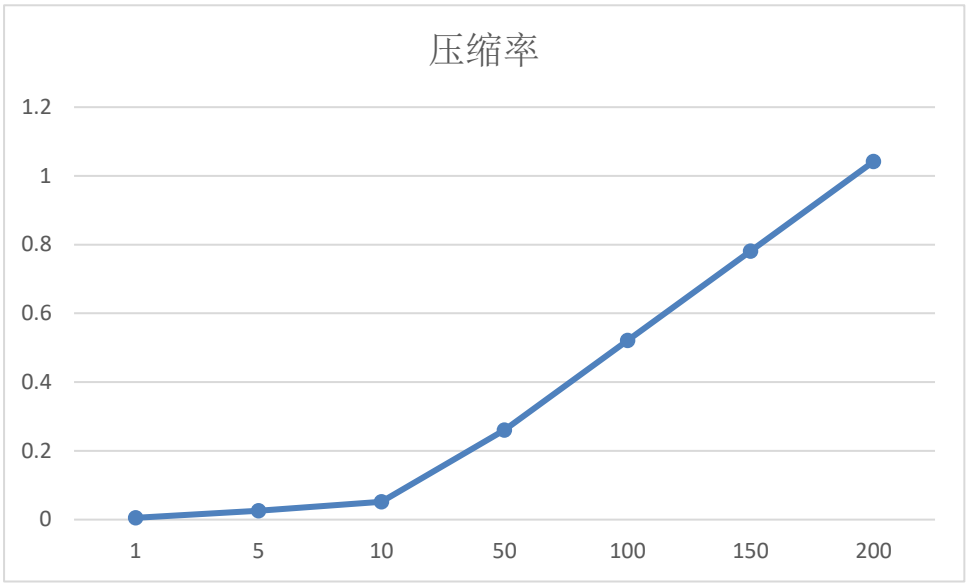
我们发现，时间随着主成分个数的变化并没有体现出规律性的变化，与我们的预期也相去甚远，主成分个数较少的时候反而时间更多。究其原因，可能是我们在不同的时候进行了不同主成分个数的实验，运行时间可能还受到电脑电量的影响。不过，10、50、100 都是在一起做的，时间都在 7 秒左右；1、5 都是在一起做的，时间都在 21 秒左右；150、200 都是在一起做的，时间都在 9 秒左右。所以，时间岁主成分个数的变化应该不大。

重构误差：



我们可以看到，随着主成分个数变多，重构误差逐渐变小，这是因为主成分变多了，更多原有图片的信息被保留了。这种变化在 50-100 之间最陡峭，在主成分个数大于 100 后趋缓。

压缩率：

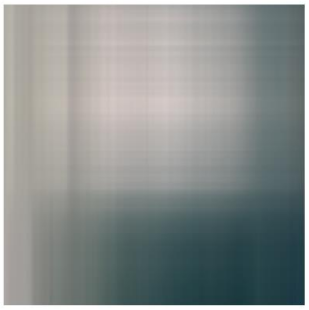


我们可以看到，随着主成分个数变多，压缩率逐渐变大，这是因为主成分变多了，需要保留的矩阵变得更大了。压缩率与主成分呈正比关系。值得一提的是，当主成分个数增加到 200，压缩率超过 1，这说明压缩后的矩阵占用的空间比原图片还大。此时就没必要再增加主成分个数了。

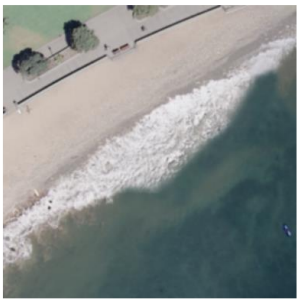
接着，我们再看看不同主成分个数下的重构效果。

- k=1

```
Image.fromarray(np.uint8(IMG_1[sample_1]))
```



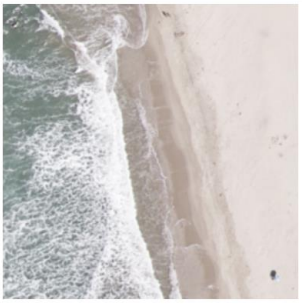
```
img_sample_1 = tf.imread(ROUTE[sample_1])
Image.fromarray(np.uint8(img_sample_1))
```



- k=5

```
Image.fromarray(np.uint8(IMG_5[sample_5]))
```

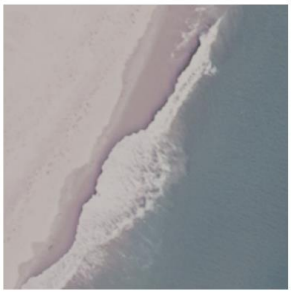
```
img_sample_5 = tf.imread(ROUTE[sample_5])
Image.fromarray(np.uint8(img_sample_5))
```



- k=10

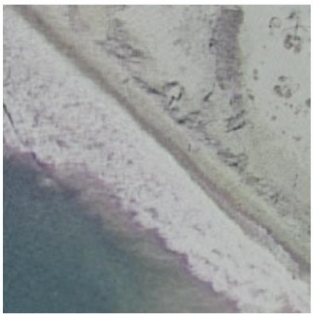
```
Image.fromarray(np.uint8(IMG_10[sample_10]
```

```
img_sample_10 = tf.imread(ROUTE[sample_10
Image.fromarray(np.uint8(img_sample_10))
```

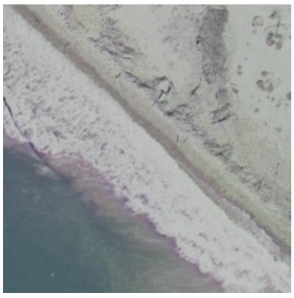


- k=50

```
Image.fromarray(np.uint8(IMG_50[sample_50
```



```
img_sample_50 = tf.imread(ROUTE[sample_50])
Image.fromarray(np.uint8(img_sample_50))
```



- k=100

```
Image.fromarray(np.uint8(IMG_100[sample_100
```

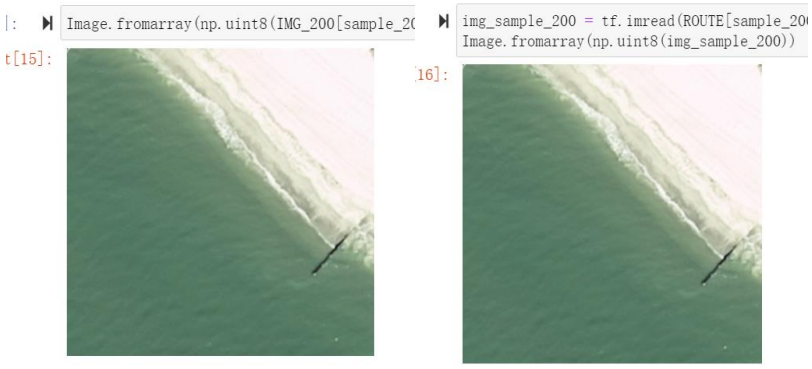
```
img_sample_100 = tf.imread(ROUTE[sample_1
Image.fromarray(np.uint8(img_sample_100))
```

- k=150

```
Image.fromarray(np.uint8(IMG_150[sample_15
```

```
img_sample_150 = tf.imread(ROUTE[sample_15
Image.fromarray(np.uint8(img_sample_150))
```

- k=200



可以看到，当主成分个数为 1 的时候，只能看到相应位置大致是什么颜色；当主成分个数为 5 的时候，图片稍微清晰一些，但是部分点颜色错误；当主成分个数为 10 的时候，没有颜色错误的点了，但是图片仍然比较模糊。当主成分个数为 50，图片的细节也依稀可见；当主成分个数大于 100，就跟没压缩前看起来没什么两样了。

五、 结论（对使用的方法可能存在的不足进行分析，以及未来可能的研究方向进行讨论）

5.1 结论

- PCA 算法能够在保证清晰度的前提下减少图片存储时的内存占用；
- 如果对清晰度要求不高，但是内存比较有限，可以取 10-50 个主成分；如果对清晰度要求较高但内存足够，可以去 100-150 个主成分；如果对清晰度和内存要求都比较高，取 50-100 个主成分。

5.2 不足

- 没有对 QR 算法求特征值和幂法求特征值的效率进行比较；
- 没有使用稀疏 PCA、鲁棒 PCA、核 PCA 等技术进一步提升 PCA 的性能。

5.3 对未来的展望

- 未来可以通过各种 PCA 变体，在相同内存大小的情况下进一步提升图片的清晰度。