

# Project2,PCA

---

## 摘要 [中文]:

---

图片压缩是图片处理中的一个重要问题，好的图片压缩需要在保留图片的主要特征的前提下进行压缩。本实验中我们通过使用主成分分析技术选取图像的主成分对图片数据进行降维，达到在保留图片主要特征的前提下进行图片压缩的目的。最终得到了较好的压缩后图片品质；把图片维度从196608维降到40维，降维效果显著；平均压缩时间为37ms，压缩时间较快；平均重构误差3965，压缩品质较高。

## Abstract [English]:

---

Image compression is an important issue in image processing. Good image compression needs to be compressed under the premise of retaining the main features of the image. In this experiment, we use principal component analysis technology to select the principal components of the image to reduce the dimensionality of the image data, so as to achieve the purpose of image compression while retaining the main features of the image. In the end, a good image quality after compression was obtained; the image dimension was reduced from 196608 dimensions to 40 dimensions, so the dimensionality reduction effect was remarkable; the average compression time was 37ms, so the compression time was fast; the average reconstruction error was 3965, so the compression quality was high.

## 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）

---

PCA作为一个非监督学习的降维方法，它只需要特征值分解，就可以对数据进行压缩，去噪。因此在实际场景应用很广泛。为了克服PCA的一些缺点，出现了很多PCA的变种，比如为解决非线性降维的KPCA，还有解决内存限制的增量PCA方法Incremental PCA，以及解决稀疏数据降维的PCA方法Sparse PCA等。

## 二、问题定义（提供问题定义的语言描述与数学形式）

---

数据集：三类图片,每类100张,每张 $256 \times 256$ 像素

实验内容：实现PCA算法：

输入：n维样本集 $X=(x_1, x_2, \dots, x_m)$ ，要降维到的维数n'.

输出：降维后的样本集Y

1、对所有的样本进行中心化  $x_i = x_i - \frac{1}{m} \sum_{j=1}^m x_j$

2、计算样本的协方差矩阵  $C = \frac{1}{m} X X^T$

3、求出协方差矩阵的特征值及对应的特征向量

4、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

5、 $Y=PX$  即为降维到k维后的数据

综合考虑图片恢复程度与空间节省程度，选择合适的主成份个数，对图片进行压缩。

并且计算其重构误差、压缩时间、压缩率来衡量压缩效果。

## 三、方法（问题解决步骤和实现细节）

### 将图片表示为矩阵

用python的PIL库中的Image来打开图片文件，然后用numpy的array函数将图片转换为矩阵即可

```
from PIL import Image
#图片预处理
m=100 #100张图片
n=256*256*3 #一张图片的维度为256*256*3(pixel数量为256*256, 一个pixel由3通道rgb来表示)
picturename=[] #存放图片的路径
mg = Image.open("./Images/agricultural/agricultural00.tif")
mat=np.array(mg)
print(mat)
```

得到的第一张图片agricultural00.tif的矩阵为：

```
[[[ 44  41  45]
 [ 53  50  56]
 [ 47  44  50]
 ...
 [125 126 122]
 [137 137 130]
 [119 119 111]]

 [[ 41  38  42]
 [ 46  43  46]
 [ 44  41  45]
 ...
 [ 83  83  81]
 [124 124 118]
 [135 131 126]]

 [[ 42  39  43]
 [ 44  41  44]
 [ 43  40  44]
 ...
 [ 90  95  91]
 [ 99 103  98]
 [101  99  97]]

 ...
 ...
 ...
 [101 101  99]
 [ 99  95  94]]
```

```
[135 129 127]]]
```

上面数据的每一行是一个像素，每一个像素由3个元素组成，分别是其rgb的值。

要同时压缩100张图片，我们需要把一张图片拉成一个行向量，然后100张图片对应100个行向量，把它们叠起来得到原始数据矩阵：(用flatten函数来把图片拉成一个行向量，然后用np.row\_stack函数来把这些行向量叠起来)

```
pictures = np.array(mg).flatten()
for i in range(1,10):
    picturename.append('./Images/agricultural/agricultural0'+str(i)+'.tif')
for i in range(10,100):
    picturename.append('./Images/agricultural/agricultural'+str(i)+'.tif')
for pic in picturename:
    img=Image.open(pic)
    picture=np.array(img).flatten()
    pictures=np.row_stack((pictures,picture)) #把向量叠在一起，叠成矩阵
```

得到的原始数据矩阵pictures为：

```
[[ 44  41  45 ... 135 129 127]
 [ 94  97  95 ... 108  87  86]
 [ 88  85  85 ... 112 118 108]
 ...
 [ 82  85  96 ... 122 113 115]
 [125 157 127 ... 159 183 150]
 [ 93 128  98 ...  54  97  77]]
```

形状是(100, 196608)，说明有100张图片，每个图片有 $256 \times 256 \times 3 = 196608$ 个维度

## 实现PCA算法(手动实现,包括计算特征值/特征向量)

### 计算协方差矩阵

将原始数据矩阵pictures去中心化后得到Y,然后 $Y^* Y^T$ 得到协方差矩阵C

```
#求协方差矩阵
#Y为去中心化后的pictures
x_mean=np.zeros(n)
for i in range(n):
    x_mean[i]=pictures[:,i].mean()

Y=np.zeros((m,n)) #cov=Y*Y^T
for i in range(m):
    y=pictures[i,:]-x_mean
    Y[i]=y
C=np.dot(Y,Y.T)/(m-1) #C为协方差矩阵
print(C.shape)
```

## 对协方差矩阵C进行特征分解

对C进行特征分解， $C = U\Sigma U^T$ ，为了进行特征分解，需要求得C的所有特征向量与特征值。

不用幂法配合降解技术的原因：用降阶技术和幂法可以算出所有特征值，但不一定可以算出所有特征向量，因为用幂法的话如果遇上一个特征值对应多个特征向量的话，只能求出其特征向量的一个线性组合而QR分解是常用有效的计算所有特征值与特征向量的方法

所以用QR分解法来求C所有特征值与特征向量：

```
#QR分解
def qrSplit(A):
    n=A.shape[0] #A的维度
    Q=[]
    R=A
    for i in range(0,n-1):
        B=R
        if i!=0:
            #删除一行一列，得n-1阶子阵
            B=B[i:,i:]
            #取第一列向量
            x=B[:,0]
            #向量摸长
            m=np.linalg.norm(x)
            #生成一个模长为m，其余项为0的向量y
            y=[0 for j in range(0,n-i)]
            y[0]=m
            #计算householder反射矩阵
            #w = (x-y)/||x-y||
            w=x-y
            w=w/np.linalg.norm(w)
            #H=E-2*WT*w
            H=np.eye(n-i)-2*np.dot(w.reshape(n-i,1),w.reshape(1,n-i))#H是个正交矩阵
            #第一次计算不需对正交正H升维
            if i==0:
                #第一次迭代
                Q=H
                R=np.dot(H,R)
            else:
                #因为降了维度，所以要拼接单位矩阵
                D=np.c_[np.eye(i),np.zeros((i,n-i))]
                H=np.c_[np.zeros((n-i,i)),H]
                H=np.r_[D,H]
                #迭代计算正交矩阵Q和上三角R
                Q=np.dot(H,Q)
                R=np.dot(H,R)
            Q=Q.T
    return [Q,R]
```

```
#QR迭代求特征值特征向量
def qrEigs(A):
    # QR迭代(尽量让它多迭代几次，以至于AK收敛为上三角)
    qr = []
    n = A.shape[0] # A的维度
```

```

Q = np.eye(n)
for i in range(0, 100):
    # A=QR
    qr = qrSplit(A)
    # 将Q右边边累成
    Q = np.dot(Q, qr[0])
    # A1=RQ
    A = np.dot(qr[1], qr[0])

AK = np.dot(qr[0], qr[1])
#把e取出来
e=[]
for i in range(0,n):
    e.append(AK[i][i])
#对特征值按降序排序(冒泡排序), 特征向量与其对应
for i in range(0,n-1):
    max=e[i]
    for j in range(i+1,n):
        if e[j]>max:
            max=e[j]
            #交换特征值
            tmp=e[i]
            e[i]=e[j]
            e[j]=tmp
            #交换特征向量
            r=np.copy(Q[:,i])
            Q[:,i]=Q[:,j]
            Q[:,j]=r;
return [e,Q] #e存储特征值, Q存储特征向量

```

调用以上函数对C进行特征分解：

```
egis =qrEgis(C)
```

这里保险起见检验一下用上述qeEgis与用python自带的linalg.eigh求出的结果是不是差不多

自己写的QR分解

```
[[ 5.72400873e-02  2.41859675e-02  2.26759571e-02 ...  5.47280619e-03
   2.51636167e-03  1.00000000e-01]
 [ 4.04573758e-02  2.50731722e-02  1.68046158e-02 ...  1.12568658e-02
   3.36178013e-03  1.00000000e-01]
 [ 5.15745117e-02  5.11466201e-02  2.56612343e-02 ...  1.96645179e-03
   -2.99159392e-03  1.00000000e-01]
 ...
 [ 9.09675831e-02  2.18395272e-02  1.54045301e-02 ...  1.07513451e-02
   -1.74693907e-05  1.00000000e-01]
 [-6.06792235e-02 -6.58465071e-02  3.64357839e-02 ... -2.59777498e-03
   -1.95350307e-03  1.00000000e-01]
 [-3.35574399e-02 -2.96451096e-01 -1.61312704e-02 ... -2.57141000e-03
   -4.15249672e-03  1.00000000e-01]]
```

numpy自带的分解器

```
[[-1.00000000e-01  2.51636167e-03 -5.45422111e-03 ...  2.26759595e-02
   -2.41859674e-02  5.72400873e-02]
 [-1.00000000e-01  3.36178013e-03 -1.12897590e-02 ...  1.68046183e-02
   -2.50731722e-02  4.04573758e-02]
 [-1.00000000e-01 -2.99159392e-03 -1.90755234e-03 ...  2.56612365e-02
   -5.11466200e-02  5.15745118e-02]
 ...
 [-1.00000000e-01 -1.74693911e-05 -1.07189353e-02 ...  1.54045342e-02
   -2.18395271e-02  9.09675831e-02]
 [-1.00000000e-01 -1.95350307e-03  2.59288320e-03 ...  3.64357769e-02
   6.58465070e-02 -6.06792236e-02]
 [-1.00000000e-01 -4.15249672e-03  2.52895541e-03 ... -1.61312774e-02
   2.96451096e-01 -3.35574404e-02]]
```

可以看出确实得到的特征向量与python自带的算出的特征向量值是接近的（顺序上的差别是因为自己实现的将特征值从大到小排列，而python自带的将特征值从小到大排列）

分解后将结果保存到 $C = U\Sigma U^T$ 中的U和 $\Sigma$ （代码中用B表示）

```
#利用QR分解来特征分解
length=len(egis[0])
B=np.eye(len(egis[0])) #B为由特征值构成的矩阵
for i in range(length):
    B[i][i]=egis[0][i]
U=egis[1]
```

## 选择合适的主成份个数,压缩所有图片

参考教材，解释率设置为0.9

```
#PCA
x=0.9 #x为解释率
eigvalue_sum=sum(egis[0])
count=0
i=0
while(count<x*eigvalue_sum):
    count+=egis[0][i]
    i+=1
```

```
print("选取了%d个主成分"%(i))
print("对数据降维, 从%d维降到%d维"%(n,i))
W=U[:,0:i] #W^T为对数据进行降维变换的矩阵
pictures_new=np.dot(W,W.T)
pictures_new=np.dot(pictures_new,Y)
for i in range(m):
    pictures_new[i]=pictures_new[i,:]+x_mean
```

将解释率设置为0.9运行后结果:

… 选取了40个主成分  
对数据降维, 从196608维降到40维

然后根据选取主成分的个数构造进行降维变化的矩阵W, 用W\* W^T\* Y+偏置量, 即可得到所有图片压缩后组成的矩阵pictures\_new。

## 将压缩后的图片的矩阵转为图片

```
from numpy import uint8

compactImage=[]
for i in range(10):

    compactImage.append('./Images/compactImages/com_agricultural0'+str(i)+'.tif')
for i in range(10,100):
    compactImage.append('./Images/compactImages/com_agricultural'+str(i)+'.tif')

# for i in range(pictures_new.shape[0]):
#     pic=pictures_new[i,:]
#     pic=pic.reshape(256,256,3)
#     mg=Image.fromarray(uint8(pic))
#     mg.save(compactImage[i])
pic=pictures_new[4,:]
pic=pic.reshape(256,256,3)
mg=Image.fromarray(uint8(pic))

pic=pictures[4,:]
pic=pic.reshape(256,256,3)
mg_old=Image.fromarray(uint8(pic))
```

## 四、 实验结果 (验证提出方法的有效性和高效性)

### 平均重构误差

计算重构误差的算法(采用欧式距离)

```
#重构误差
def recon_error(i):    #recon_error(i)返回第i张图片的重构误差
    return np.linalg.norm(pictures_new[i,:]-pictures[i,:])
```

结果：

```
1 # for i in range(10):
2 #     print(recon_error(i))
3 count=0
4 for i in range(100):
5     count+=recon_error(i)
6 print("average reconstruct error: %f"%(count/100))
[✓] 0.1s
```

average reconstruct error: 3965.826040

平均重构误差为3965

## 平均压缩时间

```
1 end_time=time.time()
2 print("average time cost:%f ms"%((end_time-start_time)/100*1000))
[✓] 0.0s
```

average time cost:37.545915 ms

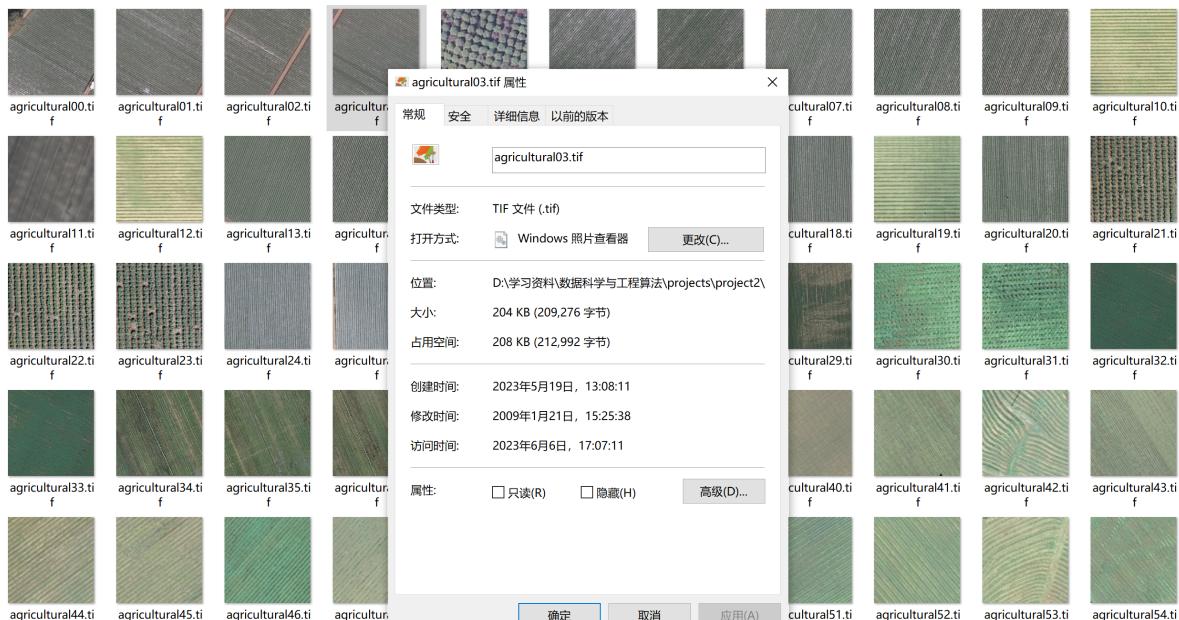
平均重构时间为37ms

可以看出压缩时间较为快速

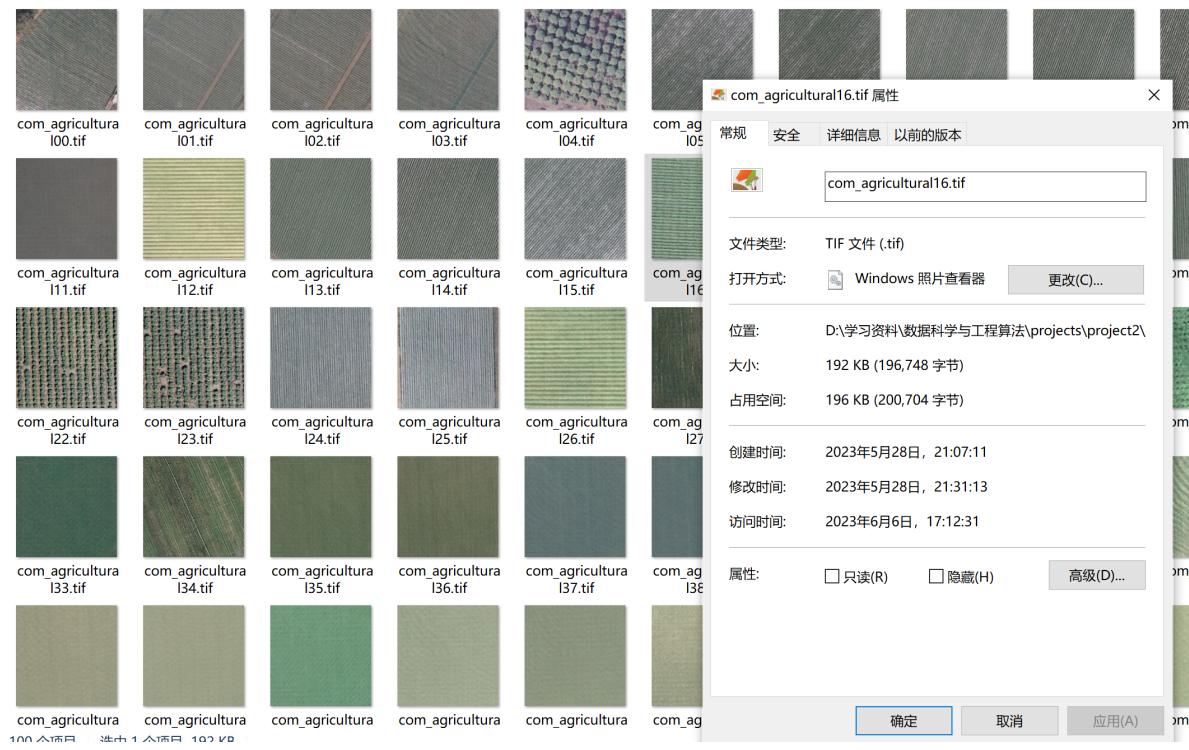
## 压缩率

$$\text{压缩率} = \frac{\text{压缩后图片大小}}{\text{原来图片大小}} = \frac{192}{204} = 0.94$$

原来图片大小大概为204 KB左右



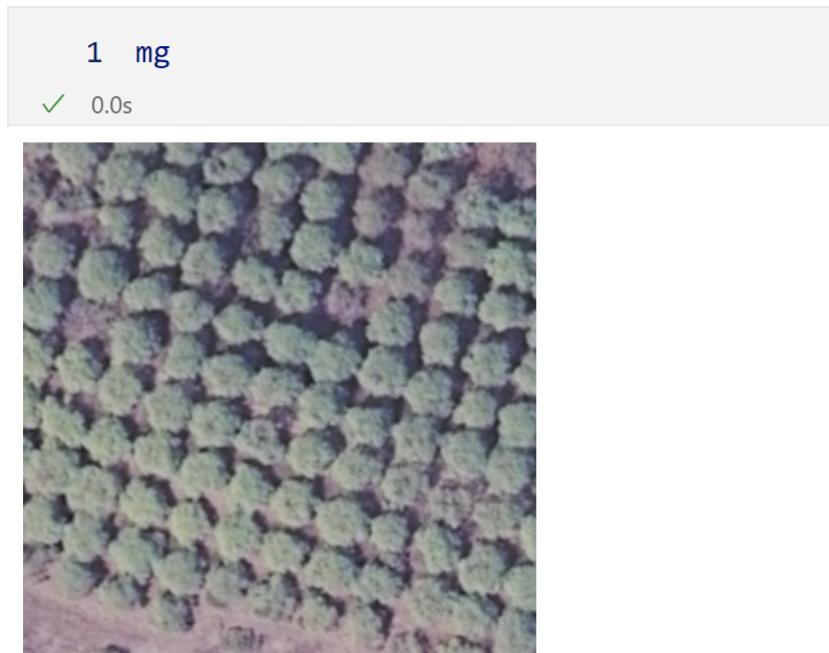
压缩后的图片大小大概为192 KB左右



## 结合两张图片分析

### 压缩前后图像

压缩后图片:



原始图片:

```
1 mg_old  
✓ 0.1s
```

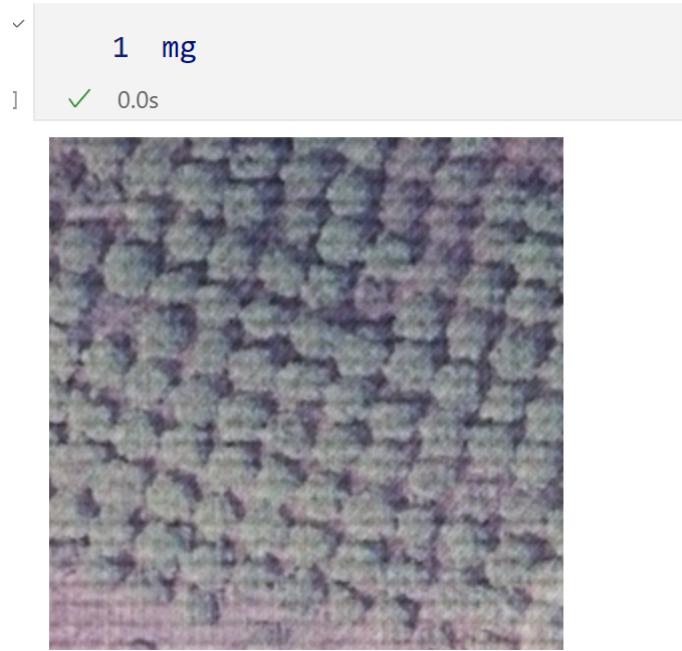


可以看到解释率为0.9时几乎看不出原图与压缩后图片的区别

故把解释率调为0.73：（但后面分析其压缩率，压缩时间，重构误差还是基于0.9的解释率

```
▷ ▾  
1 #PCA  
2 x=0.73 #x为解释率  
3 eigenvalue_sum=sum(eigis[0])  
4 count=0  
5 i=0  
6 while(count<x*eigenvalue_sum):  
7     count+=eigis[0][i]  
8     i+=1  
9 print("选取了%d个主成分"%(i))  
10 print("对数据降维，从%d维降到%d维"%(n,i))  
11 W=U[:,0:i] #W^T为对数据进行降维变换的矩阵  
12 pictures_new=np.dot(W,W.T)  
13 pictures_new=np.dot(pictures_new,Y)  
14 for i in range(m):  
15     pictures_new[i]=pictures_new[i,:]+x_mean  
[33] ✓ 0.1s
```

… 选取了14个主成分  
对数据降维，从196608维降到14维



此时可以明显看到图片被压缩的痕迹。

## 重构误差

```
1
2 print(recon_error(4))
3 # count=0
4 # for i in range(100):
5 #     count+=recon_error(i)
6 # print("average reconstruct error: %f"%(count/100))
2] ✓ 0.0s
```

573.5060480790611

这张图片的重构误差明显小于平均重构误差3965

前十张图片的重构误差如下

3757.7152934684295  
5754.787889386027  
5154.907335461174  
5769.785344271143  
573.5060480790611  
977.008853904264  
539.9148708862662  
4577.917033753968  
4368.123518649153  
582.9013760837299

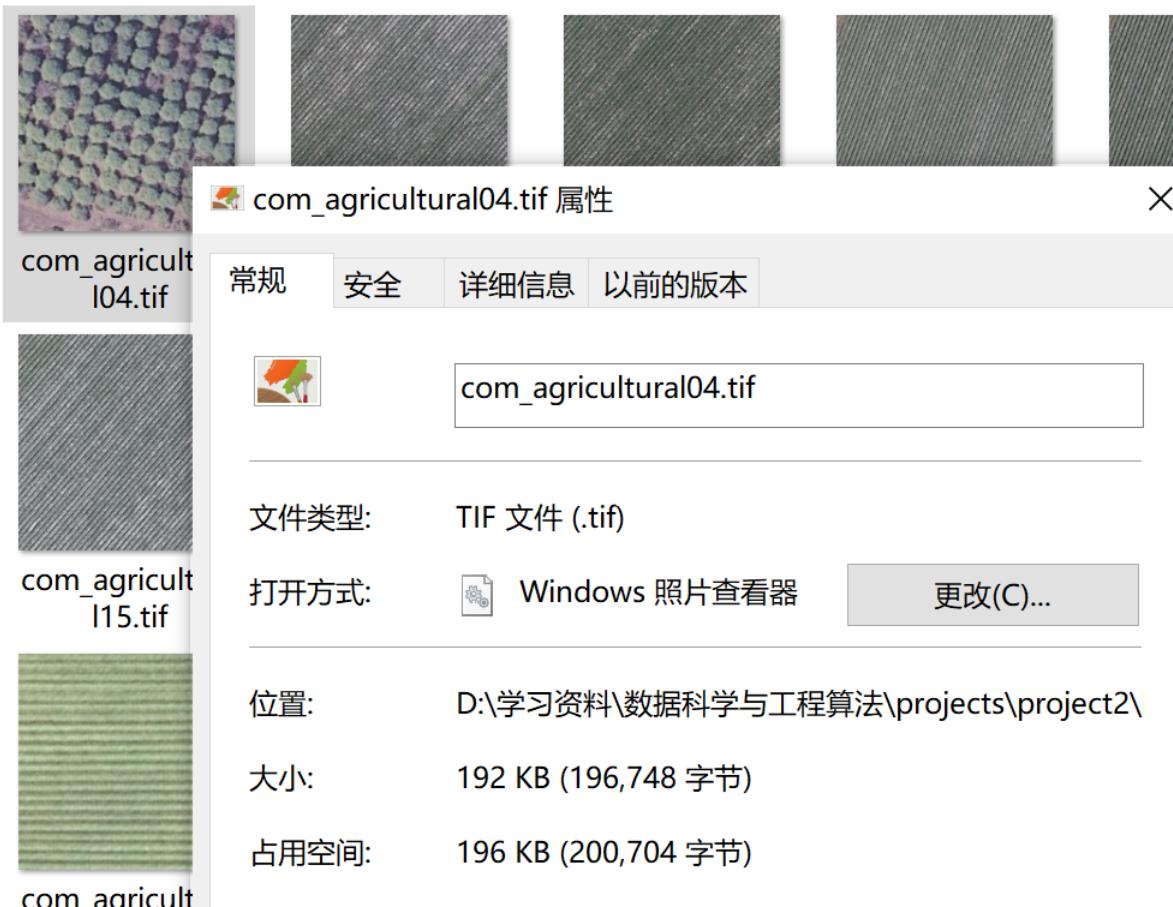
可以看出重构误差的波动很大

## 压缩时间

因为是100张图片一起压缩的，所以不知道一张图片的压缩时间，只知道平均压缩时间

## 压缩率

$$\text{压缩率} = \frac{\text{压缩后图片大小}}{\text{原来图片大小}} = \frac{192}{205} = 0.93$$



## **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）**

---

使用PCA可以快速，损失较少地对彩色图片进行压缩。本实验中采用解释率为0.9，压缩后的图片平均重构误差为3965，而平均压缩时间为37ms。

降维效果很好，在解释率达到0.9的情况下选取了40个主成分，将维度从196608维降到40维，降维效果显著，而且解释率高达0.9。说明前面的特征值比后面的特征值大得多。

不足：没有求平均压缩率，没有用代码来读取生成压缩文件的大小来求准确的平均压缩率。但是文件的大小都差不多，因此平均压缩率与单个图片的压缩率差别不大。