《数据科学与工程算法》项目报告

报告题目：利用 PCA 进行图片压缩

姓　　名：杨茜雅

学　　号：10215501435

完成日期：2023.5

摘要 [中文]：

随着数字图像的广泛应用，处理和管理大规模图像数据的需求日益迫切。图片降维作为一种重要的数据预处理技术，可以有效地减少数据的维度，并提高后续数据处理和分析的效率。主成分分析（PCA）可以用来减少矩阵（图像）的维度，并将这些新的维度投射到图像上，使其保留质量，但 K 值更小。本文从给定的数据集的三类图片中选择了 100 张同类的图像，利用 PCA 方法对图像进行降维，并评估其图像压缩的性能，如重建误差、节省空间、压缩率和案例分析等。

首先，介绍了 PCA 的基本原理和数学模型。PCA 通过线性变换将原始图像数据映射到一组正交的主成分上，其中每个主成分表示图像数据中的最大方差。通过选择保留的主成分数量，可以实现图像数据的降维。

其次，详细描述了利用 PCA 实现图片降维的步骤和方法。包括图像数据的预处理，例如灰度化、归一化等；计算图像数据的协方差矩阵；通过特征值分解求取主成分和对应的特征向量；选择适当数量的主成分进行图像降维。

最后，对利用 PCA 实现图片降维的研究进行了评估。本文使用了两种方法，分别是对所有 100 张图像的每一张单独压缩和对所有 100 张图像一起压缩。本研究的成果表明，PCA 在图像降维中具有良好的效果和潜力，但仍存在一些挑战和改进空间。未来的研究可以探索优化的 PCA 算法、非线性和深度学习方法在图像降维中的应用，以进一步提升图片降维技术的效果和应用范围。

关键词：主成分分析，PCA，图片降维，图像压缩，图像处理，数字经济

Abstract [English]

With the widespread use of digital images, the need to process and manage large-scale image data is becoming increasingly urgent. Image dimensionality reduction, an important data pre-processing technique, can effectively reduce the dimensionality of data and improve the efficiency of subsequent data processing and analysis. Principal Component Analysis (PCA) can be used to reduce the dimensionality of a matrix (image) and project these new dimensions onto the image in such a way that it retains quality but with smaller k-values. In this paper, 100 images of the same type are selected from three types of images from a given dataset, and the PCA method is used to reduce the dimensionality of the images and evaluate their performance in terms of image compression, such as reconstruction error, space saving, compression rate and case study.

Firstly, the basic principle and mathematical model of PCA are introduced. PCA maps the original image data onto a set of orthogonal principal components through a linear transformation, where each principal component represents the maximum variance in the image data. By selecting the number of principal components to be retained, dimensionality reduction of the image data can be achieved.

Next, the steps and methods for implementing image dimensionality reduction using PCA are described in detail. This includes pre-processing of the image data, such as greyscaling and normalisation; calculating the covariance matrix of the image data; finding the principal components and the corresponding eigenvectors by eigenvalue decomposition; and selecting the appropriate number of principal components for image dimensionality reduction.

Finally, the research on image dimensionality reduction using PCA is evaluated. Two methods are used in this paper, compression of each of all 100 images individually and compression of all 100

images together. The results of this study show that PCA has good results and potential in image dimensionality reduction, but there are still some challenges and room for improvement. Future research can explore the application of optimised PCA algorithms, non-linear and deep learning methods in image dimensionality reduction to further enhance the effectiveness and application of image dimensionality reduction techniques.

Keywords: principal component analysis, PCA, image dimensionality reduction, image compression, image processing, digital economy

一、 项目概述

科学价值：

利用主成分分析（Principal Component Analysis，PCA）进行图片压缩是一项具有科学价值的工作，它在数字经济领域中有以下几个方面的科学价值：

1、 数据降维：数字图像通常由大量像素组成，每个像素包含多个颜色通道。这导致图像数据集的维度非常高。利用 PCA 进行图片压缩可以将高维的图像数据降低到较低的维度，从而减少存储空间和传输带宽的需求。这对于处理大规模图像数据的数字经济应用非常重要。
2、 资源优化：图像数据的存储和传输需要大量的计算和通信资源。通过使用 PCA 进行压缩，可以减小数据的规模，从而优化资源的利用。这对于数字经济中的云计算、大数据分析和图像传输等任务具有显著的经济效益。
3、 信息保留：尽管 PCA 将原始图像数据降维，但它能够保留图像中的重要信息。PCA 通过选择主成分来表示图像，这些主成分包含了原始图像中最显著的变化和特征。因此，通过合理选择保留的主成分数量，可以在压缩过程中最大限度地保留图像的关键特征，从而实现高效的压缩和恢复。
4、 算法研究与改进：PCA 作为一种经典的降维方法，在图片压缩领域已有广泛的研究和应用。研究人员不断探索改进和优化 PCA 算法，以提高压缩的效率和质量。这包括优化计算方法、选择合适的主成分数量和开发适应不同图像特点的压缩策略。这些研究推动了数字经济领域图片压缩技术的进步。

总的来说，利用 PCA 进行图片压缩在数字经济领域具有科学价值。它通过数据降维、资源优化、信息保留和算法改进，为大规模图像数据处理、存储和传输提供了有效的解决方案。这项工作的科学价值在于推动数字经济的发展，促进图像处理和通信技术的创新。

相关研究工作：

利用主成分分析（Principal Component Analysis，PCA）进行图片压缩的相关研究工作包括以下方面：

1、 压缩效率和质量的平衡：研究人员关注如何在保持压缩效率的同时提高压缩图像的质量。他们尝试通过调整主成分的数量或采用自适应方法来平衡图像的压缩率和重建质量。
2、 压缩算法改进：研究人员致力于改进 PCA 压缩算法，以提高压缩的效率和质量。他们探索使用变种的 PCA 算法或结合其他技术（如小波变换）来改善图像的压缩性能。
3、 压缩参数的优化：研究人员研究如何通过优化压缩参数来提高 PCA 压缩的效果。这包括选择合适的主成分数量、调整压缩比例以及考虑不同图像区域的压缩需求。
4、 非线性 PCA 压缩：除了传统的线性 PCA 压缩方法，研究人员还探索了非线性 PCA 压缩方法的应用。这些方法通过引入非线性变换来更好地捕捉图像数据

中的复杂特征，从而提高压缩效果。
5、 基于深度学习的压缩方法：随着深度学习的发展，研究人员开始探索基于深度学习的图像压缩方法。这些方法利用卷积神经网络等技术来提取图像特征，并通过学习图像的表示来实现高效的压缩。

综上所述，利用 PCA 进行图片压缩的研究工作涉及压缩效率和质量的平衡、算法改进、参数优化，以及探索非线性和深度学习等新方法。这些工作的目标是提高图像压缩的效果，使其更适用于数字经济中大规模图像数据的处理、存储和传输需求。

二、 问题定义

本文给定了一个数据集，该数据集中有三类图片，分别为 agricultural、airplane、beach。每类 100 张，每张 256x256。实验需要实现 PCA 算法，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。
在最终报告中, 展示任意两张图片压缩前后的图像,及其重构误差、压缩时间、压缩率。统计 100 张图片的平均重构误差、平均压缩时间、压缩率。

实验步骤

- 任选一类图片,作为本次实验的实验对象。
- 将图片表示为矩阵。
- 实现 PCA 算法(手动实现,包括计算特征值/特征向量)。
- 选择合适的主成份个数,压缩所有图片。
- 重构所有图片,使之与原图形状一致。
- 评估 PCA 的性能，统计 100 张图片的平均重构误差、平均压缩时间、压缩率，并结合两张图片详细分析(压缩前后图像、重构误差、时间、压缩率等)。

实验注意事项

- 不允许调用直接计算特征值/特征向量、PCA、SVD 的已封装函数
- 压缩彩色图片而不是其灰度图

三、 方法

PCA 的基本原理和数学模型：

主成分分析（Principal Component Analysis，PCA）是一种常用的数据降维技术，用于将高维数据映射到低维空间中，同时保留数据中的主要信息。其基本原理和数学模型如下：

基本原理：
主成分分析的目标是通过线性变换将原始数据映射到一组正交的主成分上，使得投影后的数据具有最大的方差。
第一个主成分解释了原始数据中的最大方差，第二个主成分解释了剩余方差中的最大部分，依此类推。

通过选择保留的主成分数量，可以实现数据降维，同时最大程度地保留原始数据中的变异性。

数学模型：

假设有一个包含 m 个样本和 n 个特征的数据矩阵 X，其中每一行表示一个样本，每一列表示一个特征。

1、首先，对数据进行去中心化处理，即将每个特征减去该特征在整个数据集上的平均值，使得数据的均值为零。

2、接下来，计算数据的协方差矩阵 C，其元素 c_ij 表示第 i 个特征和第 j 个特征之间的协方差。

3、对协方差矩阵 C 进行特征值分解，得到特征值和对应的特征向量。

4、特征值表示数据在对应特征向量方向上的方差，特征向量表示数据在新的主成分方向上的投影。

5、根据特征值的大小进行排序，选择保留的主成分数量 k。

6、选取前 k 个特征值对应的特征向量作为主成分，构成变换矩阵 W。

7、最后，将原始数据矩阵 X 与变换矩阵 W 相乘，得到降维后的数据矩阵 Y，其中每一行表示一个样本在 k 个主成分上的投影。

通过上述数学模型，PCA 能够通过线性变换将原始数据映射到主成分空间中，实现数据降维的目标，并且保留了数据中的主要信息。这使得 PCA 成为一种常用的数据分析和降维技术，在多个领域中得到广泛应用。

问题解决步骤：

一、 通过使用`imageio`库读取指定 beach 文件夹中的图片数据，然后将各通道的像素值打包成一个矩阵，并将每一通道的矩阵以列向量的形式叠起来。

代码与执行流程如下：

```
a=imageio.imread("/Users/86138/Documents/数据科学与工程算法/project2/Images/beach/"+d)
a_np=np.array(a)
#print(a_np.shape)
a_r=a_np[:,:,0]
a_g=a_np[:,:,1]
a_b=a_np[:,:,2]
```

利用`imageio.imread()`函数读取图片数据，并利用`numpy`库将读取的图片数据转换为矩阵数据，然后将矩阵数据按通道分离，分别存储矩阵的 Red、Green、Blue 通道。

```
a_rs=np.reshape(a_r,(-1,1))
a_gs=np.reshape(a_g,(-1,1))
a_bs=np.reshape(a_b,(-1,1))
#print(a_rs.shape)
n=a_rs.shape[0]
m=a_gs.shape[0]
k=a_bs.shape[0]
#print(n)
#if a_rs.shape!=65536:
if count==0:
    array_nr=a_rs
    array_ng=a_gs
    array_nb=a_bs
    count=count+1
else:
    if n!=65536:
        a_rs=np.pad(a_rs,(0,65536-n))
    if m!=65536:
        a_gs=np.pad(a_gs,(0,65536-m))
    if k!=65536:
        a_bs=np.pad(a_bs,(0,65536-k))
    array_nr=np.column_stack((array_nr,a_rs))#列向量叠起来
    array_ng=np.column_stack((array_ng,a_gs))
    array_nb=np.column_stack((array_nb,a_bs))
```

利用`numpy`库的`reshape`函数将 Red、Green、Blue 通道的矩阵数据按列向量形式叠起来，并分别存储在`array_nr`、`array_ng`、`array_nb`三个矩阵中。如果当前处理的矩阵的行数不足 65536，那么在末尾进行 0 填充。

其中，if 语句中的`if count==0`作用是用于记录是否为第一次遍历图片，如果是，则直接将三个通道矩阵的数值赋值给`array_nr`、`array_ng`、`array_nb`三个矩阵；否则，将当前遍历的通道矩阵叠加在`array_nr`、`array_ng`、`array_nb`三个矩阵相应数字的列索引位置上，得到最终的`array_nr`、`array_ng`、`array_nb`三个矩阵。

二、 定义函数 comp_2d（），用于执行 PCA 压缩。实现了对输入矩阵的去中心化、计算协方差矩阵、计算特征值和特征向量、对特征值进行排序、选择前 n 个主成分，将数据投影到新的空间等步骤。
代码与执行流程如下：

```python
import imageio
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import os
#定义函数
# 打印计算过程RGB的值
# 挑选的特征向量的个数
# 参数选择0.9 0.95比较重建效果 alfa参数的影响
#还原去中心话的数据
#加回原来的中心
#计算压缩率中需考虑（保证重构正常 需要存储每张图片的中心向量）
def comp_2d(image_2d,alfa):
    cov_mat=image_2d-np.mean(image_2d,axis=0)
    center=np.mean(image_2d,axis=0)
    eig_val,eig_vec=np.linalg.eig(np.cov(cov_mat))
    #特征值排序
    index=np.argsort(eig_val)
    index=index[::-1]
    eig_vec=eig_vec[:,index]
    eig_val=eig_val[index]
    #特征值选取
    eig_val_sum=np.sum(eig_val)
    p=np.linalg.matrix_rank(image_2d)#矩阵的秩的个数
    part_eig_val=0
    for i in range(p):
        part_eig_val=part_eig_val+eig_val[i]
        if part_eig_val>(0.01*alfa*eig_val_sum):
            break
```

1.计算矩阵均值并进行去中心化

利用如下公式求解样本中心点。

$$\overline{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

2.样本协方差矩阵分解

利用 python 的 np.cov 计算协方差矩阵。利用 np.linalg.eig 对得到的协方差矩阵计算特征值和特征向量。

3.根据 alfa 参数选取前 n 个主成分

对求出协方差矩阵的特征值即奇异值进行升序排序。然后从大大小依次选择奇异值进行累加。

利用如下公式选取主成分个数，取满足精度要求的最小特征值个数。

$$k = \arg\min_{l} \left\{ \frac{\sum_{i=1}^{l} \lambda_i}{\sum_{i=1}^{rank(\boldsymbol{A})} \lambda_i} \geq \alpha \right\}$$

在代码中，我们通过循环累加，满足条件即 break 来得到主成分个数。

此处α为压缩的精度，用以限定主成分对原始数据的解释能力。在实验中我们通过对比不同α取值下的压缩率，重构误差等值分析 PCA 降维中精度对最后压缩结果的影响。

```
numpc=i#主特征的个数
eig_vec=eig_vec[:,range(numpc)]
#print("主特征值个数:")
#print(i)
#投影到新的空间
score=np.dot(eig_vec.T,cov_mat)
#计算压缩率
#print("压缩率:")
x=(np.size(eig_vec)+np.size(score)+np.size(center))/np.size(image_2d)
x1=(np.size(eig_vec)+np.size(score))/np.size(image_2d)
#print(x)
#print("空间节省:")
y=np.size(image_2d)-(np.size(eig_vec)+np.size(score)+np.size(center))
y1=np.size(image_2d)-(np.size(eig_vec)+np.size(score))
#print(y)
#重建图像
#计算重构误差，案例研究
#采用二范数求重构误差
recon=np.dot(eig_vec,score)+np.mean(image_2d,axis=0).T#将去除的中心加回来 图像质量更好
recon1=np.dot(eig_vec,score)
recon_img_mat=np.uint8(np.absolute(recon))
# sum1=0
# sum2=0
# error=0
# errors=image_2d-recon_img_mat
# for i in range(image_2d.shape[0]):
#     sum1+=np.dot(image_2d[i],image_2d[i])
#     sum2+=np.dot(errors[i],errors[i])
#     error+=sum2**2
# error=error**0.5

#print("重构误差",error)
#print("信息丢失率:",sum2/sum1)
recon_img_mat1=np.uint8(np.absolute(recon1))

#reconstruction_error = np.linalg.norm(recon_img_mat1.astype(int) - image_2d.astype(int)) / np.linalg.norm(image_2d)
reconstruction_error = np.linalg.norm(recon_img_mat1.astype(int) - image_2d.astype(int))

# recon_error=-recon_img_mat+image_2d
# recon_error_val=np.linalg.norm(recon_error)
#print(recon_error_val**2)

return recon_img_mat1,reconstruction_error,x1,y1
#return recon_img_mat
```

4.将去中心化的数据投影到新的空间，然后进行重构，最后计算重构误差，并返回重构的图像和重构误差。
计算投影到新空间的空间节省和压缩率。空间节省通过调用 python 的 size 函数计算原始矩阵的大小和重构矩阵大小与特征向量大小以及样本中心大小的大小的差值计算。压缩率通过重构矩阵大小与特征向量大小以及样本中心大小的大小的和与原始矩阵大小的比值计算。
5.重建图像
重构矩阵时，通过先前计算的图片的中心值进行归一化操作来使图像的质量更优。

三、　　定义 image_result（）函数用于读取图像并调用 comp_2d() 对其进行压缩和重构。函数对彩色图像进行颜色分离，然后调用 comp_2d()函数分别对每个颜色通道进行压缩和重构，并将压缩后的三个颜色通道重构为彩色图像，并返回重构误差、压缩率以及空间节省等指标。

代码与执行流程如下：

```
def image_result(img_path,alfa):
    a=imageio.imread(img_path)
    a_np=np.array(a)
#     a_r=a_np[:,:,0]
#     a_g=a_np[:,:,1]
#     a_b=a_np[:,:,2]
    dir=os.listdir("/Users/86138/Documents/数据科学与工程算法/project2/Images/airplane/")
    count=0
    for d in dir:
        a=imageio.imread("/Users/86138/Documents/数据科学与工程算法/project2/Images/airplane/"+d)
        a_np=np.array(a)
    #print(a_np.shape)
        a_r=a_np[:,:,0]
        a_g=a_np[:,:,1]
        a_b=a_np[:,:,2]


    # a_r_recon=comp_2d(a_r)
    # a_g_recon=comp_2d(a_g)
    # a_b_recon=comp_2d(a_b)
    a_r_recon,a_r_recon_error,r_x,r_y=comp_2d(a_r,alfa)
    a_g_recon,a_g_recon_error,g_x,g_y=comp_2d(a_g,alfa)
    a_b_recon,a_b_recon_error,b_x,b_y=comp_2d(a_b,alfa)
    recon_color_img=np.dstack((a_r_recon,a_g_recon,a_b_recon))
    recon_color_img=Image.fromarray(recon_color_img)
    #recon_color_img.show()#显示图像

    return a_r_recon_error,a_g_recon_error,a_b_recon_error,r_x,g_x,b_x,r_y,g_y,b_y
```

四、　最后我们导入图片数据集，选择 RGB 图像，重建 RGB 分量，堆叠 RGB 并将数组转化为图片并显示图像。对比压缩后图片与原图片的效果。对于重构后的图像分 RGB 三个通道分别计算样本集 100 张图片的重构误差均值。
代码与执行流程如下：

```
#a=imageio.imread("/Users/86138/Documents/数据科学与工程算法/project2/Images/airplane/airplane00.tif")
errors_r=[]
errors_g=[]
errors_b=[]
ysl=[]
compacts=[]
dir=os.listdir("/Users/86138/Documents/数据科学与工程算法/project2/Images/airplane/")

for alfa in range(50,100,5):
    sum_r=0
    compact_s=0
    error_sum_r=0
    error_sum_g=0
    error_sum_b=0
    count=0
    for d in dir:
      #print(d)
        #count=count+1
    #image_result("/Users/86138/Documents/数据科学与工程算法/project2/Images/beach/"+d)
        r_error,g_error,b_error,r_x,g_x,b_x,r_y,g_y,b_y=image_result("/Users/86138/Documents/数据科学与工程算法/project2/Images/airplane/"+d,alfa)
        error_sum_b=error_sum_b+b_error
        error_sum_r=error_sum_r+r_error
        error_sum_g=error_sum_g+g_error
        sum_r=sum_r+r_x+g_x+b_x
        compact_s=compact_s+r_y+g_y+b_y
    print("********")
    print(count)
    print("压缩率：",alfa)
    print("数据集的重构误差：")
    errors_r.append(error_sum_r/100)
    errors_g.append(error_sum_g/100)
    errors_b.append(error_sum_b/100)
    print(error_sum_r/100)
    print(error_sum_g/100)
    print(error_sum_b/100)
    print("压缩率")
    print(sum_r/300)
    ysl.append(sum_r/300)
    print("空间节省")
    print(compact_s/300)
    compacts.append(compact_s/300)
```

主程序部分循环遍历一个目录中的所有图片，并对每个图像进行压缩和重构。循环中定义了三个列表 errors_r、errors_g 和 errors_b，用于分别存储压缩过程中的三个颜色通道的重构误差。同时，定义了两个列表 ysl 和 compacts，用于存储压缩率和图像压缩的空间节省。在每次循环中，调用 image_result() 对当前图像进行压缩和重构，并计算压缩率和重构误差，最后将它们加到对应的列表中。

本次项目采用两种压缩方式进行对比，两种方式的主要区别在于对图像的压缩。

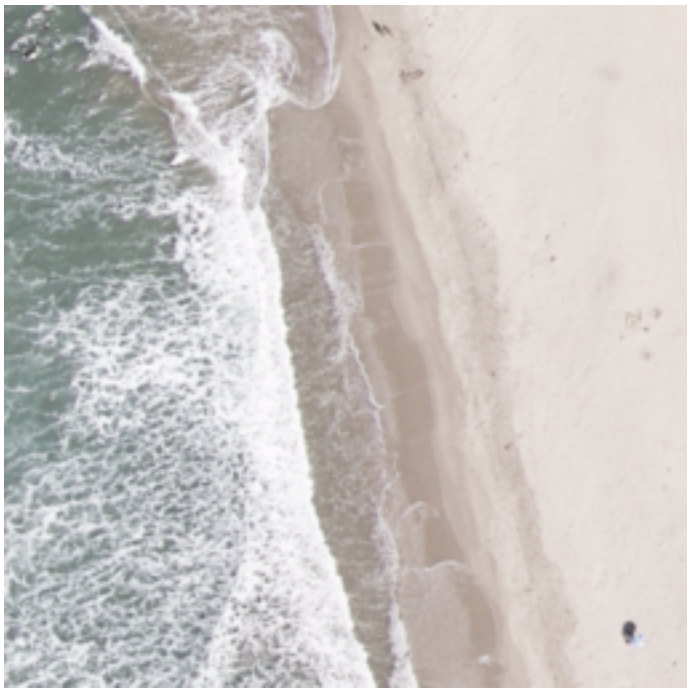
方法 1：对所有 100 张图像的每一张单独压缩，并汇总，最后计算得到重构误差、压缩率等的均值。

方法 2:对所有 100 张图像一起压缩，并计算重构误差，压缩率等。在方法 2 中，我们将一个文件夹下的所有图像的矩阵 reshape 成一维矩阵然后按行拼成一个大的矩阵，对大矩阵求中心，对去中心化后的图片数据求协方差矩阵的特征值和特征向量，按压缩精度选取主元，最后再将得到的重构矩阵按行分割，得到经过 PCA 降维后的图像。由于发现文件夹中部分图像尺寸不一致，我们统一降维为(65536,1)的图像，并对尺寸不符的图像利用 pad 函数进行 0 填充。

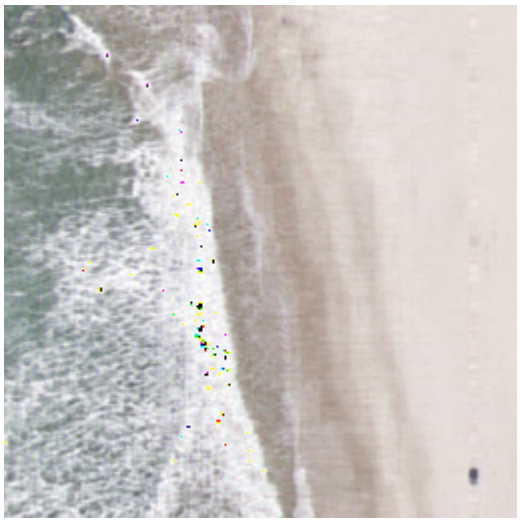
四、 实验结果

方法一

压缩前的图像：



压缩后的图像：



评价指标：

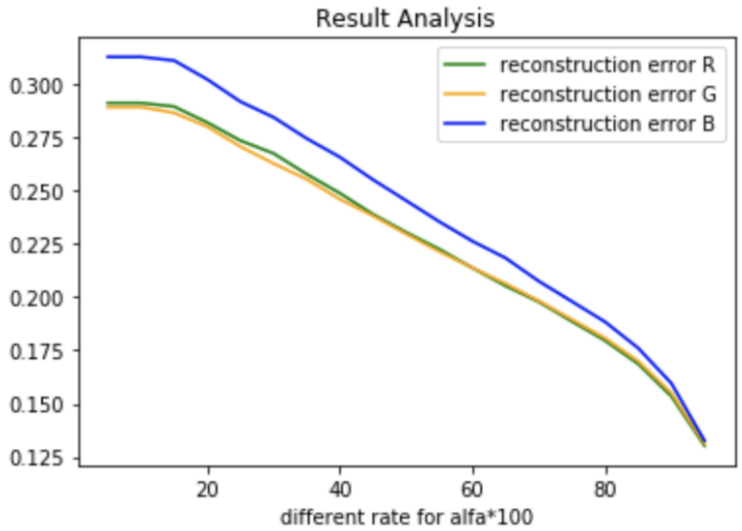
********	********	********	********	********
0	0	0	0	0
压缩率： 50	压缩率： 55	压缩率： 60	压缩率： 65	压缩率： 70
数据集的重构误差:	数据集的重构误差:	数据集的重构误差:	数据集的重构误差:	数据集的重构误差:
34154.74	34154.74	34154.74	34140.29	34140.29
39739.47	39739.47	39739.47	39202.42	38856.68
35897.03	35897.03	35741.51	35741.51	35237.19
压缩率	压缩率	压缩率	压缩率	压缩率
0.015625	0.015625	0.018229	0.023438	0.028646
空间节省	空间节省	空间节省	空间节省	空间节省
64512	64512	64341.33	64000	63658.67
********	********	********	********	********
********	********	********	********	********
0	0	0	0	0
压缩率： 75	压缩率： 80	压缩率： 85	压缩率： 90	压缩率： 95
数据集的重构误差:	数据集的重构误差:	数据集的重构误差:	数据集的重构误差:	数据集的重构误差:
33634.08	32919.6	32725.83	32188.3	31839.84
38749.81	38386.59	38165.43	37144.07	36912.66
34954.2	34501.81	33472.87	32390.77	32294.24
压缩率	压缩率	压缩率	压缩率	压缩率
0.039063	0.052083	0.067708	0.091146	0.145833
空间节省	空间节省	空间节省	空间节省	空间节省
62976	62122.67	61098.67	59562.67	55978.67
********	********	********	********	********

在方法 1 中，比较不同特征值精度要求得到的重构误差、压缩率、节省空间大小。其中重构误差通过原矩阵和重构后的矩阵的差的 F 范数与原矩阵 F 范数的比值来度量。压缩率为压缩后特征向量的大小，中心化向量的大小以及新矩阵的大小的和与原图像矩阵大小的比值。节省空间为原图像矩阵大小减去压缩后特征向量的大小，中心化向量的大小以及新矩阵的大小的和。

具体结果如下：

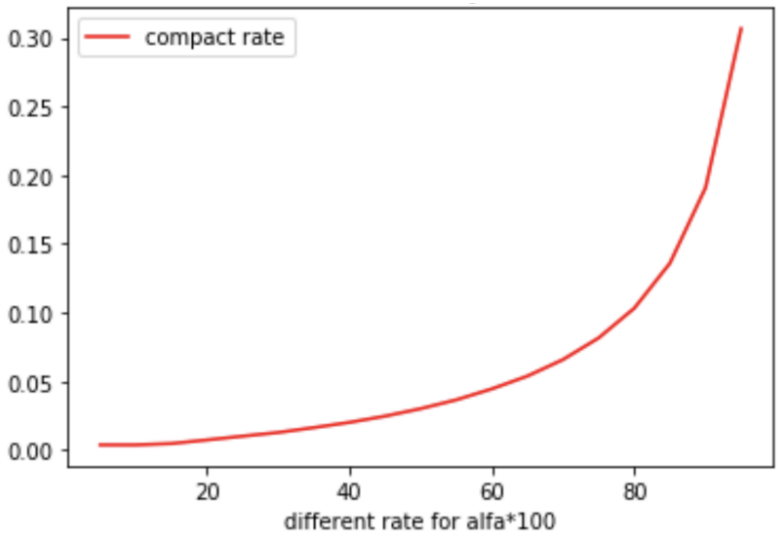
一、 **计算重构矩阵时加上原先的中心**

1）比较不同压缩精度下重构误差的变化情况和趋势



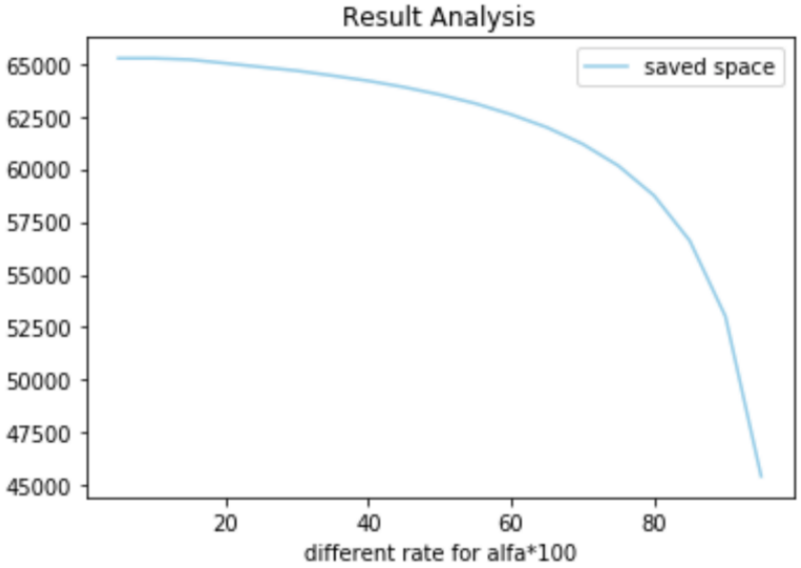
如上图可以看到，随着压缩精度从 0.05 变化到 0.95，重构误差逐渐减小，从 0.3 减小到 0.13。比较不同通道可以发现，B 通道的重构误差总体上略高于另两个通道，R 和 G 通道的变化情况较一致。

2）比较不同压缩精度下压缩率的变化情况和趋势



如上图可以看到，随着压缩精度的增加，压缩率逐渐增大。压缩精度从 0.05 变化到 0.95 的过程中，压缩率由 0.01 变化到 0.3。此外可以看到压缩精度在 0.05 到 0.45 的范围中，压缩率增幅较缓，随机压缩精度的增大，压缩率增速加快，在压缩精度处于 0.8 到 0.95 之间时，压缩率猛增，从 0.1 增至 0.3。

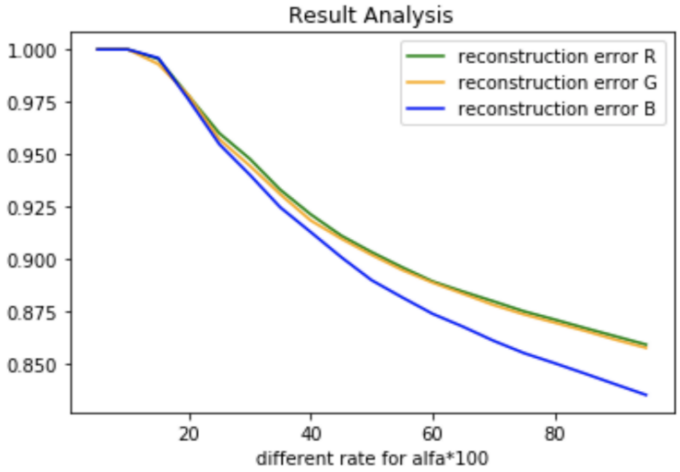
3）比较不同压缩精度下节省空间的变化情况和趋势



如上图所示，节省空间随着压缩精度的增加而减小。压缩精度从 0.05 变化到 0.95 的过程中，节省空间从 65257 减小到 45000。随着压缩精度的增加，节省空间减小的速度增大，在压缩精度在 0.8 到 0.95 之间时节省空间减小速度陡然增大。

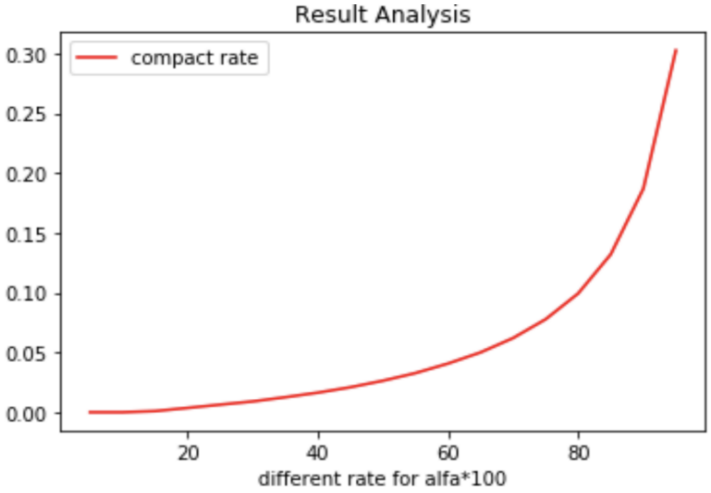
二、 **重构矩阵时不加原先的中心（存在系统误差）**

1）比较不同压缩精度下重构误差的变化情况和趋势



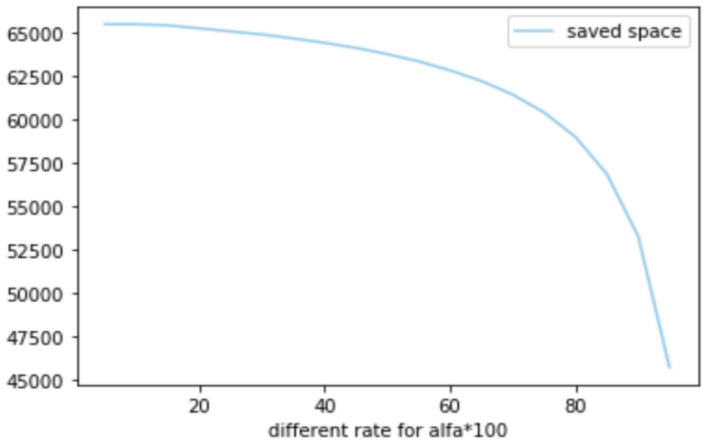
如上图所示，可以明显地发现重构矩阵计算的中没有加上原先的中心导致的重构误差的增大。随着压缩精度从 0.05 增加到 0.95，重构误差从 1 降低到 0.85。从不同通道的变化角度分析，R 和 G 通道的重构误差变化较一致，与重构矩阵计算加上原先中心的情况正好相反的是，B 通道的重构误差整体低于 R 和 G 通道的重构误差。

2）比较不同压缩精度下压缩率的变化情况和趋势



如上图所示，压缩率随着压缩精度从 0.05 增大到 0.95 而从 0 增大到 0.3,除了压缩率的初始值为 0，与计算重构矩阵时加入原中心得到的结果很类似。

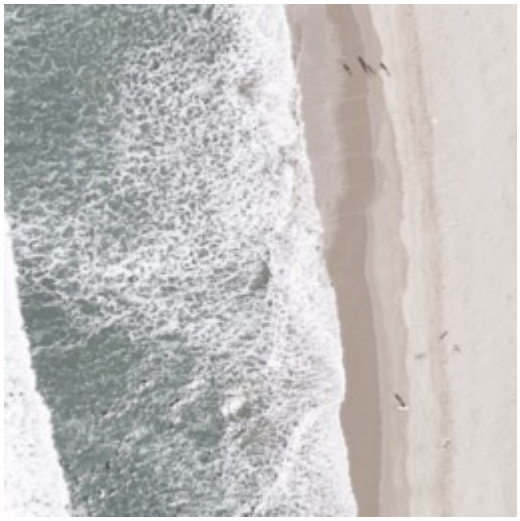
3）比较不同压缩精度下节省空间的变化情况和趋势



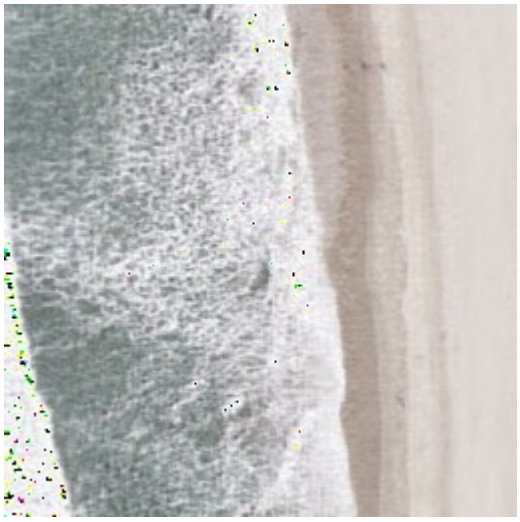
如上图所示，节省空间随着压缩精度从 0.05 增大到 0.95 而从 65512 减小到 45686。与计算重构矩阵时加入原中心得到的结果很类似。

方法二

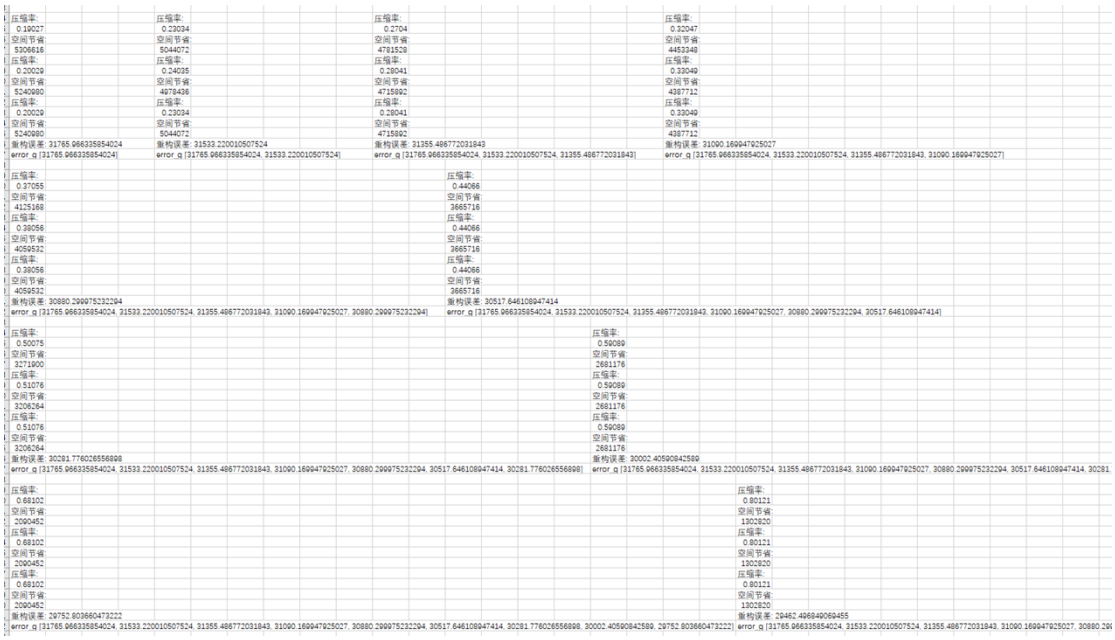
压缩前的图像：



压缩后的图像：



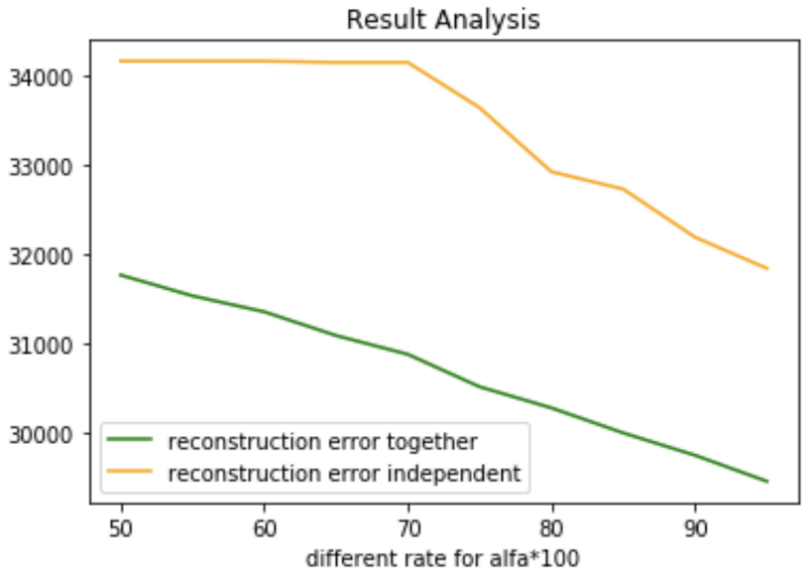
评价指标：



在方法二，我们对由大矩阵按行分解重建后的矩阵和原图像矩阵的 F 范数差来度量重构误差。压缩率为压缩后特征向量的大小，中心化向量的大小以及新矩阵的大小的和与原图像矩阵大小的比值。节省空间为原图像矩阵大小减去压缩后特征向量的大小，中心化向量的大小以及新矩阵的大小的和。此外，对得到的结果与独立压缩的结果进行比较。

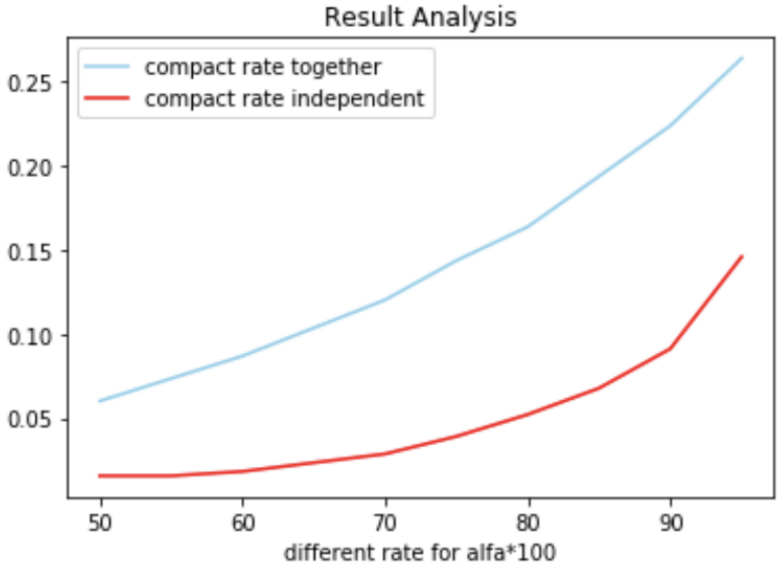
具体结果如下

1）比较不同精度下重构误差的变化情况：



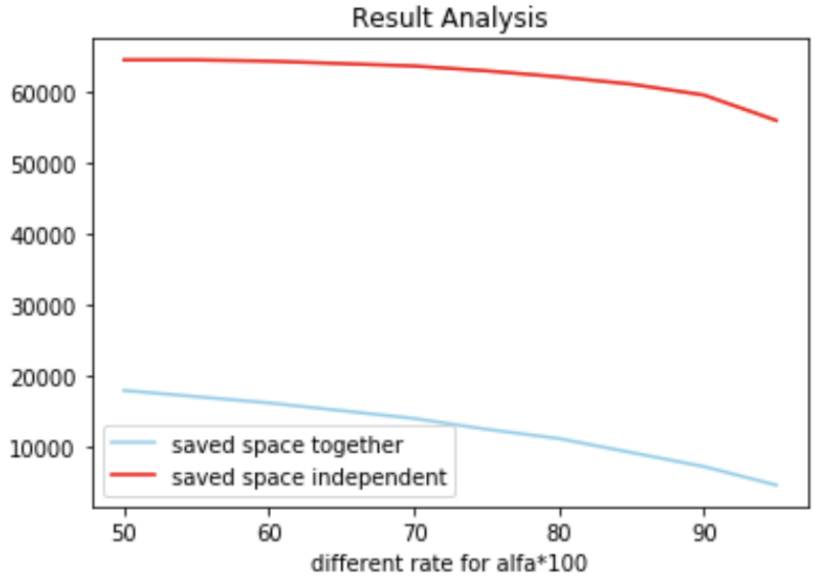
对比一起压缩和独立压缩的重构误差。图中一起压缩的重构误差变化由绿色线条显示，独立压缩的重构误差变化由黄色线条显示。可以看到拼合成大矩阵一起压缩的效果更好。

2）比较不同压缩精度下压缩率的变化情况：



对比一起压缩和独立压缩的压缩率。图中一起压缩的压缩率变化由蓝色线条显示，独立压缩的压缩率化由红色线条显示。可以看到拼合成大矩阵一起压缩的的压缩率整体比独立压缩的压缩率更高。

3）比较不同精度下空间节省的变化情况：



对比一起压缩和独立压缩的空间节省。图中一起压缩的空间节省变化由蓝色线条显示，独立压缩的空间节省化由红色线条显示。可以看到拼合成大矩阵一起压缩的的空间节省整体比独立压缩的空间节省更高。

五、 结论

可能的不足：

在本次实验中，我使用了两种方法，分别是对 100 张图像独立压缩和将 100 张图像一起压缩，分别分析比较重构误差，压缩率，空间节省的变化情况。

其中可能存在的不足有：

1、方法二将所有图片一起压缩时，对于缺失值的处理的更好解决方法值得继续研究。

2、本次实验未涉及对于不同文件夹的图像，放在一起压缩，并研究结果，可以比较不同文件夹图片一起压缩的效果与单个文件夹压缩效果。

未来可能的研究方向：

PCA 有许多应用，其中之一是在图片处理领域，特别是图像降噪、压缩、恢复和识别等方面，都可以利用 PCA 来进行降维和特征提取。未来在图片降维方向的研究，可从以下几个方面探索：

1、深度学习与 PCA 的结合：PCA 是一种传统的线性降维技术，而深度学习是一种非线性降维技术，将它们有机地结合起来可以更有效地提取图像特征。

2、大规模图像处理：PCA 在降维时需要遍历整个数据集，当数据集很大时，PCA 的计算成本会变得非常高。因此，未来的研究可以探索如何让 PCA 在处理大规模图像时更有效率。

3、结合卷积神经网络（CNN）：CNN 在图像处理方面取得了巨大的成功，在抽取图像特征方面优于 PCA。而 PCA 在图像重建、压缩方面表现更优。这两种方法可以相互结合，最大限度地提取图像特征，完成图像重建和压缩的工作。

4、处理大尺度图像：对于大尺度的高分辨率图像，PCA 在计算上会遇到很大的问题，而使用分块的 PCI 方法可以在保持降维效果的同时，降低计算量。

综上，未来在图片降维方向的研究可以结合深度学习、CNN、大规模图像处理以及分块 PCA 等技术，来发掘更多实际场景下的应用。