

统计分析方法作业 3

16337203 屈博雅

一. 使用 PCA 进行图像压缩

输入一张灰度图片 Lena，使用 PCA 方法把原始图片分别按照 2:1、8:1、32:1 进行压缩，即压缩后的数据量为原始图片的 1/2、1/8、1/32。分析压缩后的数据所含信息量大小，并比较压缩数据再经过重建后与原始图片的视觉差异。

(1) 数据描述



属性	值
图像	
分辨率	512 x 512
宽度	512 像素
高度	512 像素
位深度	8
文件	
名称	lena_gray.bmp
项目类型	BMP 文件

(2) 算法介绍:

有两种算法，一种采用传统 PCA 算法，另一种是基于 SVD 分解的 PCA 算法。

(i) 传统 PCA 算法

主成分分析能够通过提取数据的主要成分，减少数据的特征，也可以说是选取包含信息量最多的方向对数据进行投影（投影方向可以从最大化方差或者最小化投影误差这两个方面考虑），以达到数据降维的目的。

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 ，即一列为一条数据；
- 2) 将 X 的每一列（代表一个属性变量）进行零均值化，即减去这

一列的均值；

3) 求出协方差矩阵 C；

4) 求出协方差矩阵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大到小（降序）按列排列成矩阵，取前 k 列组成矩阵 P，也就是说若特征值为[1, 2, 3]→[3, 2, 1],

则特征向量

	1	2	3		3	2	1
4	5	6	→	6	5	4;	
	7	8	9		9	8	7

6) $Y=P^T X$ 即为降维到 k 维后的数据。

下一步开始重建图像。

1) $image=PY$ 即将降维矩阵维度恢复为原图片维度；

2) 将 image 的每一列加上原图片这一列的均值，即将 PCA 过程中的 2) 这一步逆向，消除零均值化的影响，得到压缩图片；

(ii) 基于 SVD 分解的 PCA 算法

设原矩阵为 X，则 SVD 分解则是 $X=U\Sigma V^T$ ，其中向量 **U** 和 **V** 分别为 σ 的左奇异向量和右奇异向量。

推导不在此描述，总之结果如下：

左奇异矩阵可以用于行数的压缩。相对的，右奇异矩阵可以用于列数即特征维度的压缩， ΣV^T 就是主成分 (PC)，也就是我们的 PCA 降维。

这样一来，对 X 做 SVD 就可以直接得到 PCA 的结果 Y，不用求出协方差矩阵，计算方便。

但是图像处理中，我们得到 SVD 分解的值后，直接通过截尾的奇异值分解公式重构出原图像，不再特意求出 $Y=\Sigma V^T$ 这个中间值。

公式如下，k 为所需的特征值个数。

$$\hat{X} = \sum_{i=1}^k \sigma_i u_i v_i^T$$

(3) 实现结果:

(i) PCA

PCA 算法+分块为 32×32 个 (16, 16)

压缩为 1/2, 贡献率 0.993298

压缩为 1/8, 贡献率 0.940172

压缩为 1/8, 贡献率 0.784808



(ii) SVD

SVD 算法+分块为 32×32 个 (16, 16)

压缩为 1/2, 贡献率 0.929115168604

压缩为 1/8, 贡献率 0.76418746267

压缩为 1/8, 贡献率 0.612547863346



(4) 实现方法:

实验环境: python3+win10+vscode

算法如上介绍, 只是在实现算法时加入分块处理, 把图分成 N 个不重叠的 16×16 的小块, 变成 $N \times 256$ 的数据集, 然后进行 PCA 处理, 再重新恢复为 $N \times 256$ 的数据集, 再还原分块处理, 得到 512×512 的原图。

这是上面结果的得到方法。贡献率的求法为所取特征值的和与总特征值的和之比。

实际上程序已实现更为灵活的分块模式，可以自行选择，不过我没有写输入界面，可以在源代码处修改。下面简单介绍一下使用方法。

(i) PCA

运行代码为 `pca_2_improve.py`

函数：

```
def pca_image(size_block0,size_block1,num_val,row,col,im2)
```

输入：

```
row,col = im2.shape
size_block0=16
size_block1=16
out1=pca_image(size_block0,size_block1,int(size_block0*size_block1/2),row,col,im2)
```

`num_val` 是指所需特征值的个数。

要求分块大小是 512 的除数，如 (16, 16)，(16, 32)。

(ii) SVD

运行代码为 `pca_3_improve.py`

函数：

```
def pca_image(size_block0,size_block1,row,col,im2,r)
```

输入：

```
row,col=im2.shape
size_block0=16
size_block1=16
img_1=pca_image(size_block0,size_block1,row,col,im2,2)
```










`r` 为压缩倍数。

要求分块大小是 512 的除数，如 (16, 16)，(16, 32)。

(5) 实现过程：

一开始题目要求是将图片尺寸变为 256*256，再开始压缩，所以初版本很简单，情况如表 1，之后图片尺寸不需要改动，所以将算法改进了，最后总结修改后就是如 (4) 实现方法中描述的两个函数了，但是在实现过程中，还是将这些中间过程展示一下。

256*256 照片测评:

名称+说明	压缩倍数 (贡献率 score)			备注
	2	8	32	
pca_1.py: svd 算法, 对原图不做处理即可	compression ratio is 2  K= 128 score= 0.950766058559	compression ratio is 8  K= 32 score= 0.726416559465	compression ratio is 32  K= 8 score= 0.519785566398	
pca_3.py: svd 算法 对原图做 分块处理, 要求 原图为长度可开方的 方阵	 K= 128 score= 0.961905278315	 K= 32 score= 0.748146405275	 K= 8 score= 0.560827175686	待改进, 只能处理 256*256, 不能处理 512*512. 分块为 16*16 个 (16*16)
pca_2.py: pca 算法, 分成 256 块的效果	 K= 128 score= 0.997764	 K= 32 score=0.917937	 K= 8 score=0.720919	分块为 16*16 个 (16, 16)

说明:

代码还处在最初级阶段, 有很多不完善的地方。

SVD:

pca_1.py 可以直接处理图片, 无论尺寸, 但是明显压缩倍数越大, 效果越不好, 差距十分明显, 32 倍时信息丢失严重。







pca_3.py 就是 pca_1.py 加入分块处理, 信息丢失情况明显改善, 但是此时分块尺寸处理还不灵活, 要求为方阵且可整除 256。

PCA:

pca_2.py 加入了分块处理，但是要求分块为方阵。

512*512 照片测评

名称： 说明	压缩倍数（贡献率 score）			备注
	2	8	32	
pca_1.py: svd 算法， 对原图不 做处理即 可	 K= 256 score= 0.963888508407	 K= 64 score= 0.791910894269	 K= 16 score= 0.582582909219	
pca_3_im prove.py: svd 算法： 可以自定义分块大小，请注意整除问题	 K= 256 score= 0.958867072229	 K= 64 score= 0.796136546321	 K= 16 score= 0.617908719669	改进了， 目前为 16*32 个 (16, 32) 的分法
pca_3_im prove.py: svd 算法	 K= 128 score= 0.929115168604	 K= 32 score= 0.764187462671	 K= 8 score= 0.612547863346	目前为 32*32 个 (16, 16) 的分法
pca_2.py: pca 算法， 要求分块 为方阵， 推荐 8, 16 (4 与 32 开始会出现问题)	 K= 128 score= 0.993298	 K= 32 score=0.940172	 K= 8 score=0.784808	分块为 32*32 个 (16, 16) 像素块

pca_2_improve.py: pca 算法, 可以自定义分块大小, 请注意整除问题	 K=256 score=0.998591	 K=64 score=0.971813	 K=16 score=0.863429	目前 是 16*32 个 (16, 32) 的分法
pca_2_improve.py:	 K=256 score=0.999053	 K=64 score=0.978954	 K=16 score=0.911452	目前 是 128*4 个 (4, 128) 的分法

说明:

代码处在最终分析整理阶段, 有不清晰的地方。

SVD:

pca_1.py 未作改变, 展示了 512*512 照片的处理结果。

pca_3_improve.py 就是 pca_3.py 优化分块尺寸处理后的结果, 分块尺寸可整除 256 即可, 展示了 (16, 16) 与 (16, 32) 两种分块的结果。

PCA:

pca_2.py 未作改变, 展示了 (16, 16) 分块的结果。

pca_2_improve.py 为 pca_2.py 优化分块尺寸处理后的结果, 分块尺寸可整除 256 即可, 展示了 (16, 32) 与 (4, 128) 两种分块的结果。

补充说明:

pca_2_improve.py 中, 分块尺寸为 (4, 4), 压缩 32 倍时, 只能选取 0 个特征值, 贡献率为 0, 特征值没有超过 32 个, 只有 16 个, 但是解压图片效果还可

以，比较模糊；(32, 32) 时压缩 2 倍会出现选取 512 个特征值，贡献率为 1，特征值共有 1024 个，解压效果非常奇怪，目前没想到错误原因，如果进一步将分块变大，效果更明显。总之贡献率为 1 时会出现较大的解压误差，请不要随意将分块大小扩大；贡献率为 0 时压缩倍数受限，强行压缩效果一般。图片展示如下。

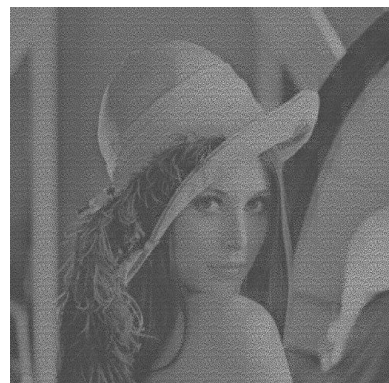


原图

总结：



pca_2_improve.py, 分块尺寸为 (4, 4), 贡献率为 0, 压缩 32 倍的解压图片



pca_2_improve.py, 分块尺寸为 (32, 32), 贡献率为 1, 压缩 2 倍的解压图片

整理完，最后就是 SVD——pca_3_improve.py, PCA——pca_2_improve.py 的对应关系。

(6) 文件说明：

1. 报告 16337203_屈博雅.pdf

2. 图片文件夹

压缩照片对比

pca2_im

4_132

16_16

16_32

pca2_im_bad

4_4

32_32

pca3_im

16_16

16_32

文件夹命名即为代码分属+分块大小；

图片命名则为代码简名_像素大小

(512)_img1\2\3(压缩倍数 2\8\32)_分块大小, 如

pca_2_im_512_img3_4_4。

3. 代码

pca_2_improve

pca_3_improve