主成分分析 – 图片压缩

- PCA可以用少量维度来表示高维数据，从而用更少的数据量表示丰富的信息
- 是不是可以用PCA来压缩图片呢？

主成分分析 – 图片压缩

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

img = np.load('lena.npy')
plt.imshow(img, cmap='Greys')
plt.axis('off')
plt.savefig('lena.png')
plt.clf()
```

主成分分析 – 图片压缩

- 思路
  - 直接把512x512的图片当成一个有512行，每行512个特征值的数据集

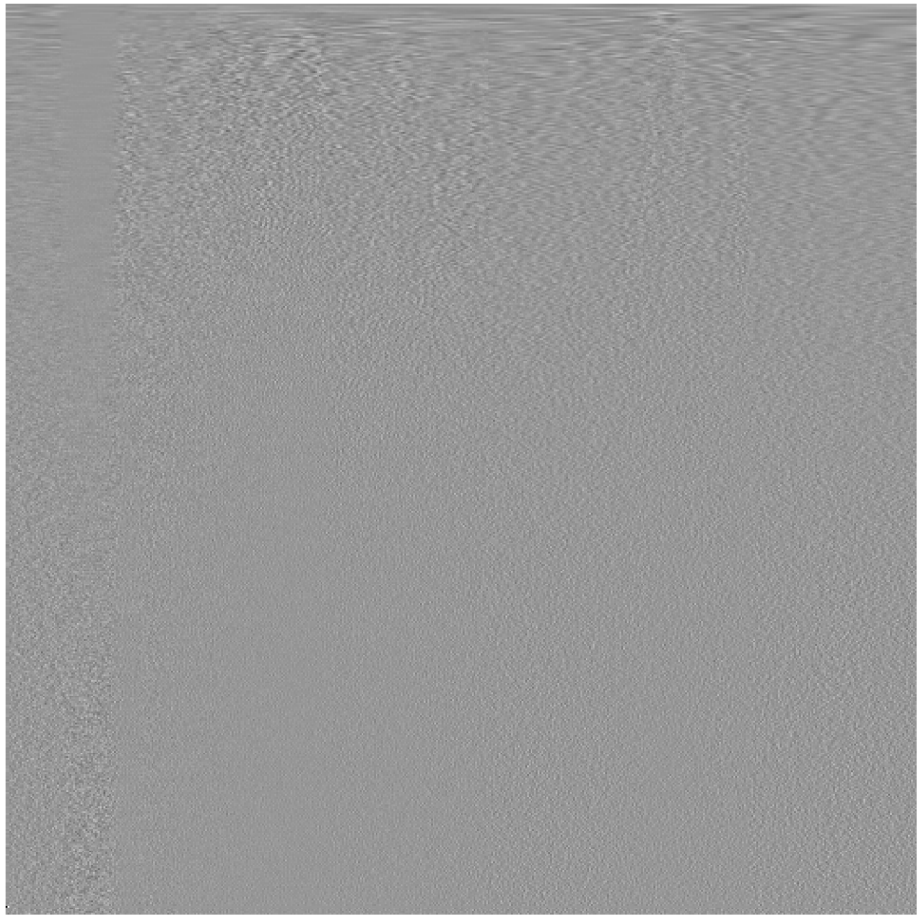
```
from sklearn.decomposition import PCA
for n_components in range(1, 26):
    pca = PCA(n_components=n_components)
    compressed_img = pca.fit_transform(img)
    decomp_img = pca.inverse_transform(compressed_img)
    plt.subplot(5, 5, n_components)
    plt.imshow(decomp_img, cmap='Greys')
    plt.title('n_comp = %d\ncompress ratio = %f%%'%(n_components,\
        100 - n_components * 513 / (512 * 512) * 100))
    plt.axis('off')
plt.show()
plt.savefig('lena_compress.png')
```

主成分分析 – 图片压缩


n_comp = 1
compress ratio = 99.804306%
n_comp = 2
compress ratio = 99.608612%
n_comp = 3
compress ratio = 99.412918%
n_comp = 4
compress ratio = 99.217224%
n_comp = 5
compress ratio = 99.021530%
n_comp = 6
compress ratio = 98.825836%
n_comp = 7
compress ratio = 98.630142%
n_comp = 8
compress ratio = 98.434448%
n_comp = 9
compress ratio = 98.238754%
n_comp = 10
compress ratio = 98.043060%
n_comp = 11
compress ratio = 97.847366%
n_comp = 12
compress ratio = 97.651672%
n_comp = 13
compress ratio = 97.455978%
n_comp = 14
compress ratio = 97.260284%
n_comp = 15
compress ratio = 97.064590%
n_comp = 16
compress ratio = 96.868896%
n_comp = 17
compress ratio = 96.673203%
n_comp = 18
compress ratio = 96.477509%
n_comp = 19
compress ratio = 96.281815%
n_comp = 20
compress ratio = 96.086121%
n_comp = 21
compress ratio = 95.890427%
n_comp = 22
compress ratio = 95.694733%
n_comp = 23
compress ratio = 95.499039%
n_comp = 24
compress ratio = 95.303345%
n_comp = 25
compress ratio = 95.107651%

主成分分析 – 图片压缩

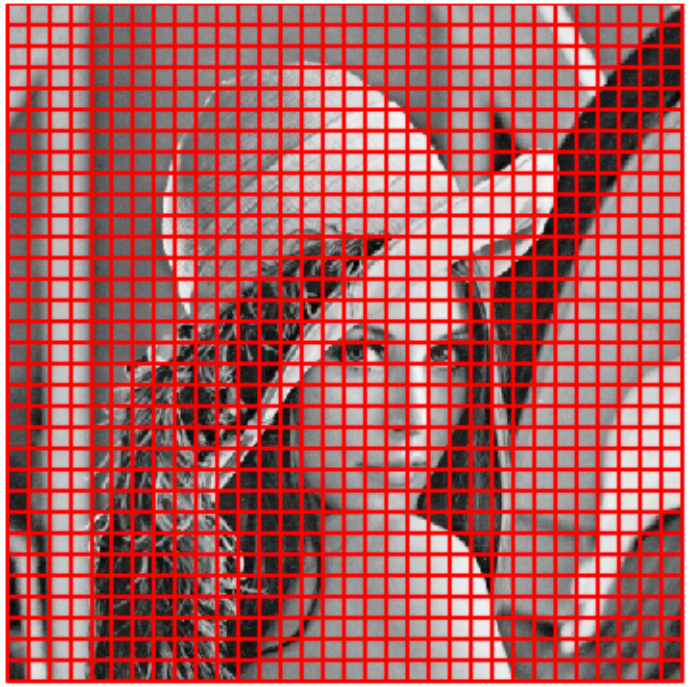
- 右图为全部512个主成分
- 每行为一个主成分
- 越高的行所对应的主成分越大
- 我们可以清楚地看到越高的主成分中的数据越有结构
- 越低的主成分越接近噪音
- 这样的主成分虽然压缩效果很好但分解出来的主成分实际意义不大



主成分分析 – 图片压缩

- 换一个思路
  - 把图片切成32x32格，每格16x16个像素
  - 将每格当做一个长度为16x16=256的一行数据，共32x32=1024行数据
  - 对1024行数据进行PCA压缩

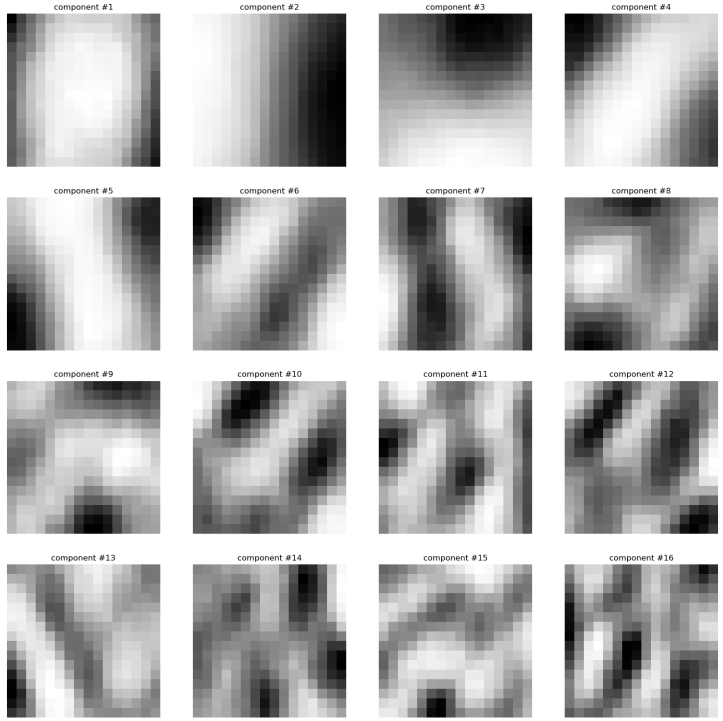
```
size = 16
width = int(512 / size)

sub_images = np.zeros([width*width, size*size])
for i in range(width):
    for j in range(width):
        sub_images[i*width+j] = img[i*size:(i+1)*size, j*size:(j+1)*size].reshape([256])
```



主成分分析 – 图片压缩

- 分解出的主成分更加有意义

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(sub_images)
# show learned kernels
for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.imshow(pca.components_[i].reshape([16, 16]), cmap='Greys')
    plt.title('component #%d'%(i+1))
    plt.axis('off')
plt.show()
```



主成分分析 – 图片压缩

- 分解出的主成分更加有意义
- 在更高压缩率的情况下保留了更多图片信息

```
# show recon with diff n_comp
for n_components in range(1, 26):
    pca = PCA(n_components=n_components)
    compressed_img = pca.fit_transform(sub_images)
    decomp_subimgs = pca.inverse_transform(compressed_img)

    rec_img = np.zeros([512, 512])
    for i in range(width):
        for j in range(width):
            rec_img[i*size:(i+1)*size, j*size:(j+1)*size] = decomp_subimgs[i*width+j].reshape([size,size])

    plt.subplot(5, 5, n_components)
    plt.imshow(rec_img, cmap='Greys')
    plt.title('n_comp = %d\ncompress ratio = %f%%'%(n_components,\
        100 - n_components * (size*size+1) / (512 * 512) * 100))
    plt.axis('off')
plt.show()
```

主成分分析 – 图片压缩

- 分解出的主成分更加有意义
- 在更高压缩率的情况下保留了更多图片信息
- 想一想，为什么会有这样的现象呢?