

**2020年春季学期  
计算学部《机器学习》课程**

**Lab 4实验报告**

|  |  |
| --- | --- |
| 姓名 | 余涛 |
| 学号 | 1180300829 |
| 班号 | 1803202 |
| 电子邮件 | 1063695334@qq.com |
| 手机号码 | 15586430583 |

**目录**

[1 实验内容 3](#_Toc56030656)

[1.1 实验目的 3](#_Toc56030657)

[1.2实验要求 3](#_Toc56030658)

[1.3实验环境 3](#_Toc56030659)

[2 实验设计思想 3](#_Toc56030660)

[2.1 算法原理 3](#_Toc56030661)

[2.1.1 PCA的推导 3](#_Toc56030662)

[2.1.2 PCA算法流程 4](#_Toc56030663)

[2.1.3 原始图片的压缩 5](#_Toc56030664)

[2.1.4 特征向量矩阵的虚部处理 5](#_Toc56030665)

[2.2 算法实现 5](#_Toc56030666)

[2.2.1创建二维或者三维数据集 5](#_Toc56030667)

[2.2.2 PCA算法实现 6](#_Toc56030668)

[2.2.3 原始图片压缩的实现 7](#_Toc56030669)

[2.2.4特征向量矩阵的虚部处理的实现 8](#_Toc56030670)

[2.2.5 绘制图像 8](#_Toc56030671)

[2.2.6 计算信噪比 10](#_Toc56030672)

[3 实验结果分析 11](#_Toc56030673)

[3.1 生成数据集进行测试 11](#_Toc56030674)

[3.2使用人脸数据集进行测试 14](#_Toc56030675)

[4 结论 19](#_Toc56030676)

[5 源代码 19](#_Toc56030677)

# 实验内容

## 实验目的

实现一个PCA模型，能够对给定数据进行降维（即找到其中的主成分）

## 1.2实验要求

测试：

（1）首先人工生成一些数据（如三维数据），让它们主要分布在低维空间中，如首先让某个维度的方差远小于其它唯独，然后对这些数据旋转。生成这些数据后，用你的PCA方法进行主成分提取。

（2）找一个人脸数据（小点样本量），用你实现PCA方法对该数据降维，找出一些主成分，然后用这些主成分对每一副人脸图像进行重建，比较一些它们与原图像有多大差别（用信噪比衡量）。

## 1.3实验环境

Windows 10 专业版；python 3.8.6；PyCharm Community Edition 2020.2.2 x64

# 实验设计思想

## 2.1 算法原理

### 2.1.1 PCA的推导

PCA，全称为Principal Component Analysis，即主成分分析，是一种常用的降维方法。作用是从一堆高维数据中提取一部分特征，然后根据这些特征向低维进行变换，能够应用于数据压缩和高维数据的可视化。

PCA一共有两种形式，基于最小投影距离和基于最大投影方差两种形式。但两种形式的本质是一样的，以本次实验中用到的最大方差为例：

最大方差，顾名思义，就是为了得到最大的方差，这个最大方差是指将高维数据向高维空间中的某一个平面投影，以求得投影得到的数据点的方差最大，当反差最大时，能够尽可能多的保留原数据的特征。可以用一个例子来说明：

假设有一个三维空间内的椭球体，当其向某个平面进行投影时，投影为椭圆能够更多的保留椭球体的特征，此时的反差最大，而投影为圆的方差却更小。

对于最大方差的推导如下：

假设m个n维数据都已经进行了中心化，即。经过投影变换后得到的新坐标系为，其中w是标准正交基，即，

如果我们将数据从n维降到n'维，即丢弃新坐标系中的部分坐标，则新的坐标系为，样本点在n'维坐标系中的投影为：。

其中，是在低维坐标系里第j维的坐标。

对于任意一个样本，在新的坐标系中的投影为，在新坐标系中的投影方差为，要使所有的样本的投影方差和最大，也就是最大化的迹,即：

利用拉格朗日函数可以得到

对W求导有, 整理下即为：

W为的n'个特征向量组成的矩阵，而−λ为的若干特征值组成的矩阵，特征值在主对角线上，其余位置为0。当我们将数据集从n维降到n'维时，需要找到最大的n'个特征值对应的特征向量。这n'个特征向量组成的矩阵W即为需要的矩阵。对于原始数据集，我们只需要用，就可以把原始数据集降维到最小投影距离的n'维数据集。

### 2.1.2 PCA算法流程

求样本的n'维的主成分其实就是求样本集的协方差矩阵的前n'个特征值对应特征向量矩阵W，然后对于每个样本,做如下变换，即达到降维的PCA目的。

具体的算法流程为：

输入：给定样本集，要降维的维数为n'

输出：中心化后的数据集，特征向量矩阵，降维前特征向量均值

（1）对样本集进行去中心化操作

（1.1）计算样本均值：

（1.2）所有样本减去均值得到中心化后的数据集：

（2）计算样本的协方差矩阵

（3）对矩阵进行特征值分解

（4）取出最大的n’个特征值对应的特征向量，将所有的特征向量标准化后，组成特征向量矩阵W

（5）输出中心化后的数据集，特征向量矩阵W，降维前特征向量均值μ

### 2.1.3 原始图片的压缩

由于较大的数据在求解特征值和特征向量时很慢，所以需要将原图像进行压缩。先依次读取文件中的每一个图像，然后将图像进行压缩，将图像通过三通道转化为灰度图，然后得到该图像的维度，然后将图像数据拉平即可。

### 2.1.4 特征向量矩阵的虚部处理

当通过PCA算法得到了中心化后的数据集，特征向量矩阵W，降维前特征向量均值μ后，若对原数据集进行降维，由于特征向量矩阵W可能存在虚部，所以需要先对特征向量矩阵W进行虚部处理（保留实部即可），然后就可以得到投影数据集了。

## 2.2 算法实现

### 2.2.1创建二维或者三维数据集

作用：

*"""  
生成三维或二维数据集****:param*** *data\_dimension: 需要生成的维度****:param*** *num: 需要生成的数据集的数据量****:return****: 生成的数据集  
"""*

具体实现：

对于二维和三维数据分别定义均值和方差，然后生成num个数据加入数据集即可

具体以注释形式给出。

1. **def** create\_data\_by\_two\_or\_three\_dimension(data\_dimension, num):
2. """
3. 生成三维或二维数据集
4. :param data\_dimension: 需要生成的维度
5. :param num: 需要生成的数据集的数据量
6. :return: 生成的数据集
7. """
8. **if** data\_dimension == 2:  # 对二维数据定义均值和方差
9. mean = [-2, 2]
10. cov = [[1, 0], [0, 0.01]]
11. **elif** data\_dimension == 3:  # 对三维数据定义均值和方差
12. mean = [1, 2, 3]
13. cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
14. **else**:
15. **assert** False
16. data\_set = []  # 定义数据集
17. **for** index **in** range(num):  # 生成num个数据，加入数据集
18. data\_set.append(np.random.multivariate\_normal(mean, cov).tolist())
19. **return** np.array(data\_set)

### 2.2.2 PCA算法实现

作用：

*"""  
将数据集data\_set用PCA从D维降至k维，data\_set.shape = (N, D)****:param*** *data\_set:原始数据集****:param*** *k:PCA后的维度****:return****:center\_data，中心化后的数据，shape=(N, D)。eigenvector\_matrix，特征向量矩阵，shape=(D, k)。data\_mean，降维前样本均值，shape=(1, D)  
"""*

具体实现：

根据算法原理中介绍的PCA算法过程即可，先得到原数据集的均值，然后对原数据集进行中心化，然后生成中心化后的数据集的协方差矩阵，然后求解协方差矩阵的特征值和特征向量，接着对特征值排序，取前k个最大的特征值，然后选取特征值对应的特征向量组成特征向量矩阵，然后返回即可。

具体以注释形式给出。

1. **def** PCA(data\_set, k):
2. """
3. 将数据集data\_set用PCA从D维降至k维，data\_set.shape = (N, D)
4. :param data\_set:原始数据集
5. :param k:PCA后的维度
6. :return:center\_data，中心化后的数据，shape=(N, D)。eigenvector\_matrix，特征向量矩阵，shape=(D, k)。data\_mean，降维前样本均值，shape=(1, D)
7. """
8. rows, cols = data\_set.shape  # 得到数据集的行和列
9. data\_mean = np.sum(data\_set, 0) / rows  # 计算降维前样本均值
10. center\_data = data\_set - data\_mean  # 进行数据集的中心化操作
11. covariance\_matrix = np.dot(center\_data.T, center\_data)  # 计算协方差矩阵X.T · X
12. eigenvalue, feature\_vectors = np.linalg.eig(covariance\_matrix)  # 对协方差矩阵(D,D)进行特征值分解，分别求得特征值和特征向量
13. eigenvalue\_sorted = np.argsort(eigenvalue)  # 将所有特征值排序
14. eigenvector\_matrix = feature\_vectors[:, eigenvalue\_sorted[:-(k + 1):-1]]  # 取出前k个最大的特征值对应的特征向量组成特征向量矩阵
15. **return** center\_data, eigenvector\_matrix, data\_mean

### 2.2.3 原始图片压缩的实现

作用：

*"""  
从文件中中读取面部图像数据并压缩****:param*** *file\_path: 文件路径****:return****: 返回解析面部图像得到的数据集  
"""*

具体实现：按照算法原理中给出的原理进行实施即可，要注意首先需要读取文件路径中所有的图像放入一个集合中。

具体以注释形式给出。

1. **def** read\_from\_file(file\_path):
2. """
3. 从文件中中读取面部图像数据并压缩
4. :param file\_path: 文件路径
5. :return: 返回解析面部图像得到的数据集
6. """
7. size = (50, 50)  # 由于较大的数据在求解特征值和特征向量时很慢，故统一压缩图像为size大小
8. i = 1
9. file\_list = os.listdir(file\_path)  # 读取该路径下所有图像的列表，放入file\_list
10. data\_set = []  # 定义数据集
11. plt.figure(figsize=size)
12. **for** file **in** file\_list:  # 对于file\_list中所有图像
13. path = os.path.join(file\_path, file)  # 连接文件路径，得到每个图像的路径
14. plt.subplot(3, 4, i)
15. with open(path) as f:
16. image = cv2.imread(path)  # 读取这张图像
17. image = cv2.resize(image, size)  # 将图像压缩至size大小
18. image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # 将图像通过三通道转换为灰度图
19. plt.imshow(image\_gray)  # 预览该图像
20. h, w = image\_gray.shape  # 得到该图像的维度
21. image\_col = image\_gray.reshape(h \* w)  # 对(h,w)的图像数据拉平
22. data\_set.append(image\_col)  # 加入该图像数据给数据集中
23. i += 1
24. plt.show()
25. **return** np.array(data\_set)

### 2.2.4特征向量矩阵的虚部处理的实现

作用：

当降维后的维度超过某个值，特征向量矩阵将出现复向量，对其保留实部

具体实现：除去PCA算法得到的特征向量矩阵W可能存在的虚部，只保留实部即可。

具体以注释形式给出。

1. w = np.real(w)  # 当降维后的维度超过某个值，特征向量矩阵将出现复向量，对其保留实部

### 2.2.5 绘制图像

作用：

*"""  
对执行PCA前后的数据集，在图像上显示****:param*** *dimension: 维度****:param*** *data\_before\_PCA: 原始数据集****:param*** *x\_after\_PCA: 执行PCA之后的数据集****:return****:  
"""*

*"""  
输出PCA后的图像并打印信噪比****:param*** *data\_set: PCA前的数据集****:param*** *w: 特征向量矩阵****:param*** *center\_data: 中心化后的数据****:param*** *mu\_x: 降维前样本均值****:param*** *x\_num: 数据个数  
"""*

具体实现：调用绘制图像的函数即可。

1. **def** draw\_picture\_by\_create\_PCA(dimension, data\_before\_PCA, x\_after\_PCA):
2. """
3. 对执行PCA前后的数据集，在图像上显示
4. :param dimension: 维度
5. :param data\_before\_PCA: 原始数据集
6. :param x\_after\_PCA: 执行PCA之后的数据集
7. :return:
8. """
9. **if** dimension == 2:  # 对二维数据画图
10. plt.scatter(data\_before\_PCA[:, 0], data\_before\_PCA[:, 1], facecolor="none", edgecolor="b",
11. label="data\_before\_PCA")
12. plt.scatter(x\_after\_PCA[:, 0], x\_after\_PCA[:, 1], facecolor='r', label='x\_after\_PCA')
13. plt.xlabel('x')
14. plt.ylabel('y')
15. **elif** dimension == 3:  # 对三维数据画图
16. fig = plt.figure()
17. ax = fig.gca(projection='3d')
18. ax.scatter(data\_before\_PCA[:, 0], data\_before\_PCA[:, 1], data\_before\_PCA[:, 2], edgecolor="b",
19. label='data\_before\_PCA')
20. ax.scatter(x\_after\_PCA[:, 0], x\_after\_PCA[:, 1], x\_after\_PCA[:, 2], facecolor='r', label='x\_after\_PCA')
21. ax.set\_xlabel('x')
22. ax.set\_ylabel('y')
23. ax.set\_zlabel('z')
24. **else**:
25. **assert** False
26. plt.legend()
27. **def** draw\_picture\_by\_image(data\_set, w, center\_data, mu\_x, x\_num):
28. """
29. 输出PCA后的图像并打印信噪比
30. :param data\_set: PCA前的数据集
31. :param w: 特征向量矩阵
32. :param center\_data: 中心化后的数据
33. :param mu\_x: 降维前样本均值
34. :param x\_num: 数据个数
35. """
36. size = (50, 50)  # 由于较大的数据在求解特征值和特征向量时很慢，故统一压缩图像为size大小
37. w = np.real(w)  # 当降维后的维度超过某个值，特征向量矩阵将出现复向量，对其保留实部
38. x\_after\_PCA = np.dot(center\_data, w)  # 计算降维后的数据
39. refactoring\_data = np.dot(x\_after\_PCA, w.T) + mu\_x  # 重构降维后的数据
40. plt.figure(figsize=size)
41. **for** i **in** range(x\_num):
42. plt.subplot(3, 4, i + 1)
43. plt.imshow(refactoring\_data[i].reshape(size))  # 预览该图像
44. plt.show()
45. **print**("PCA后的信噪比如下所示：")
46. **for** i **in** range(x\_num):  # 打印信噪比
47. psnr = PSNR(data\_set[i], refactoring\_data[i])
48. **print**('图像', i + 1, '的信噪比: ', psnr)
49. plt.show()

### 2.2.6 计算信噪比

作用：

*"""  
计算两章图像的峰值信噪比PSNR****:param*** *image1: 第一张图像****:param*** *image2: 第二张图像****:return****: 信噪比PSNR  
"""*

具体实现：对PCA前后两张图片求信噪比即可。

1. **def** PSNR(image1, image2):
2. """
3. 计算两章图像的峰值信噪比PSNR
4. :param image1: 第一张图像
5. :param image2: 第二张图像
6. :return: 信噪比PSNR
7. """
8. mse = np.mean((image1 / 255. - image2 / 255.) \*\* 2)
9. **if** mse < 1.0e-10:
10. **return** 100
11. max\_pixel = 1  # 将最大像素设置为1
12. **return** 20 \* math.log10(max\_pixel / math.sqrt(mse))  # 计算信噪比即可

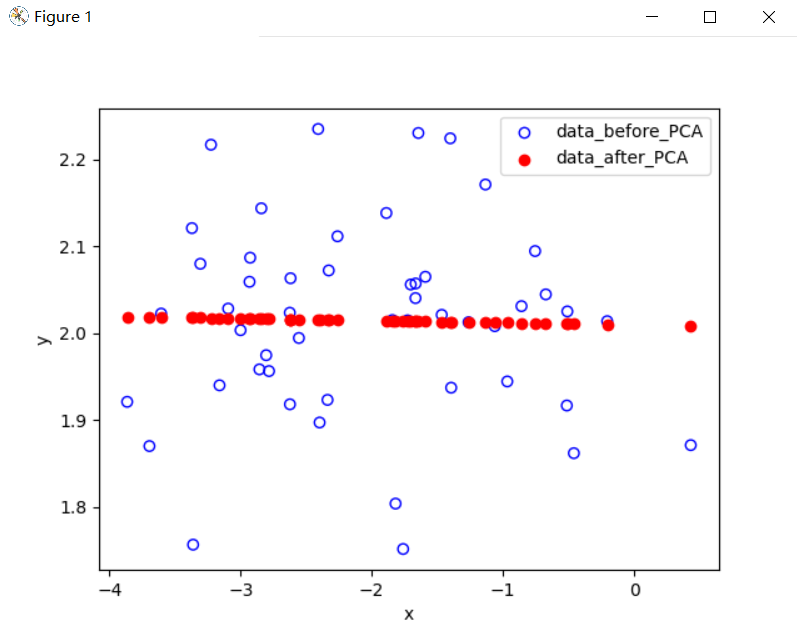
# 实验结果分析

## 生成数据集进行测试

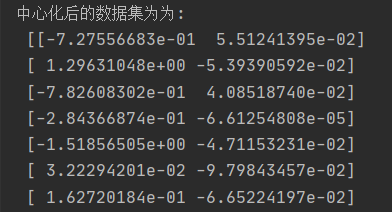
（1）生成二维数据集进行测试：

均值为[-2, 2]，反差为

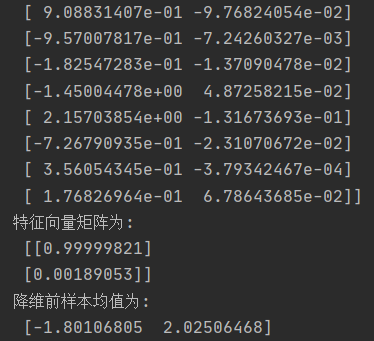
在这里的数据中，第一维的方差远小于第二维的方差，所以第二维包含了更多的信息，直接PCA即可：



输出为：



……



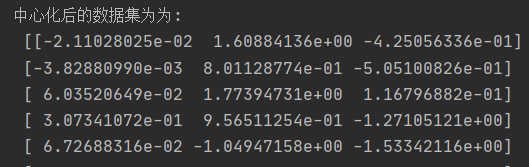
分析可以得到执行PCA之后，数据分布在了一维的直线上，并且在横轴上的方差更大，在纵轴上的方差更小，说明在PCA会后得到的直线与横轴更近。

（2）生成三维数据集进行测试：

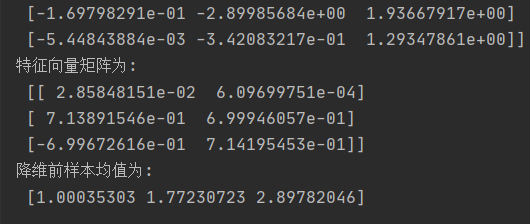
均值为[1, 2, 3]，方差为

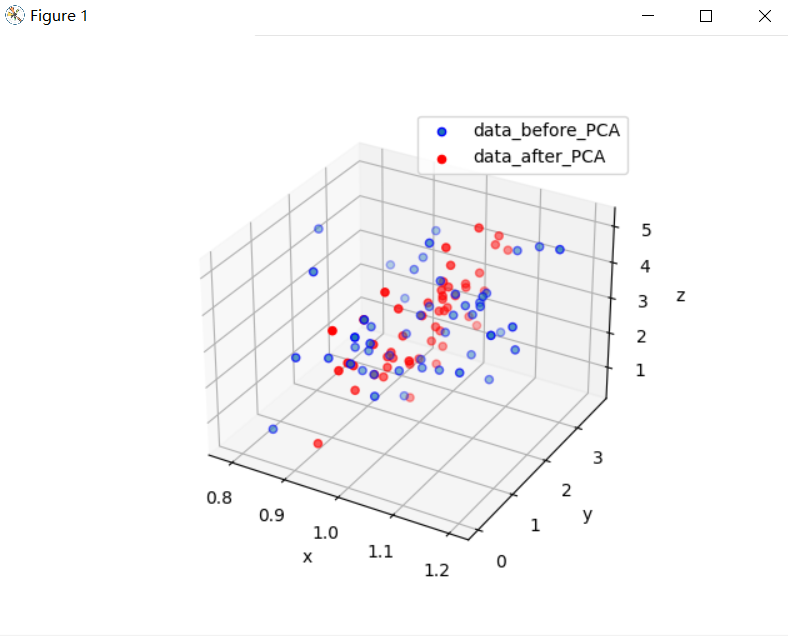
在这里的数据中，第一维的反差远小于另外两维的方差，所以第一维包含了的信息很少，直接PCA即可

输出为

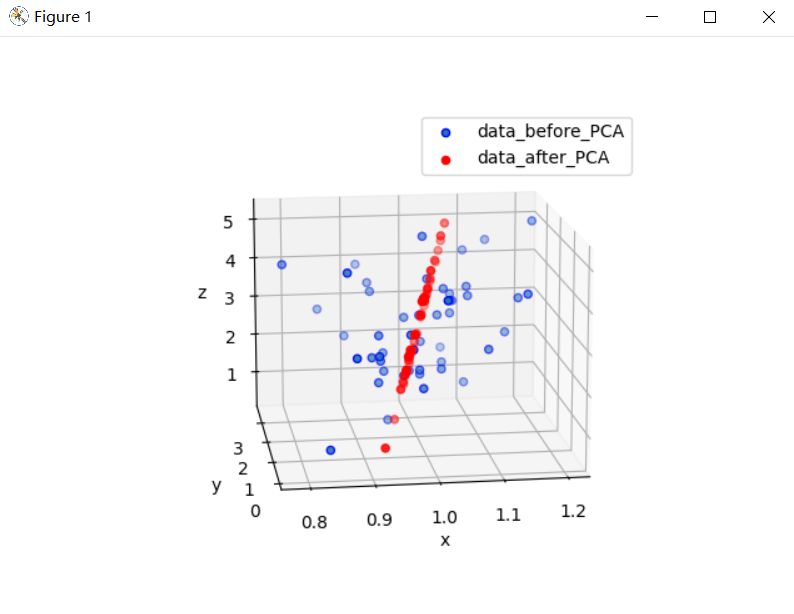


……



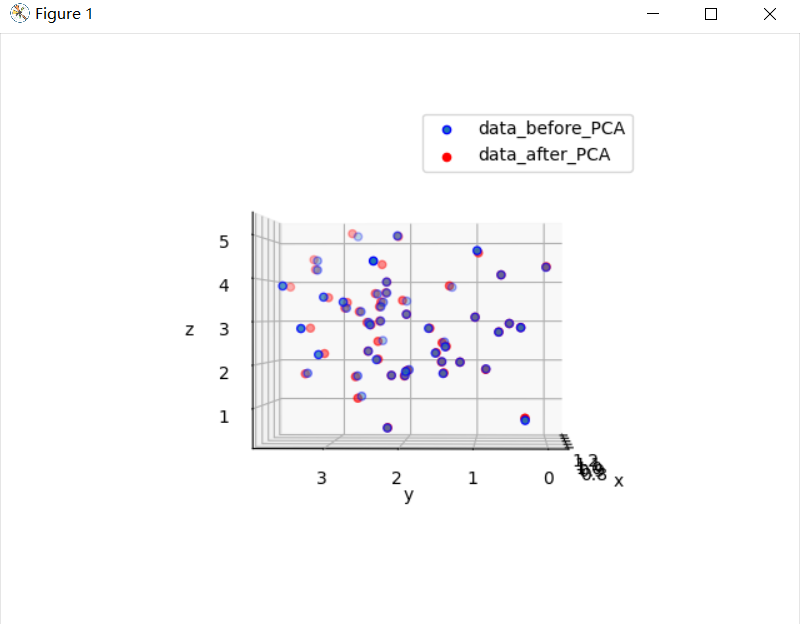


在图中可以看出在x轴的单位长度表示的长度更小，说明表示的原数据集上的第一维数据，然后对图像旋转得到如下：



此时将三维降低到了二维的平面上，并且和方差最小的一维（x轴）相垂直。

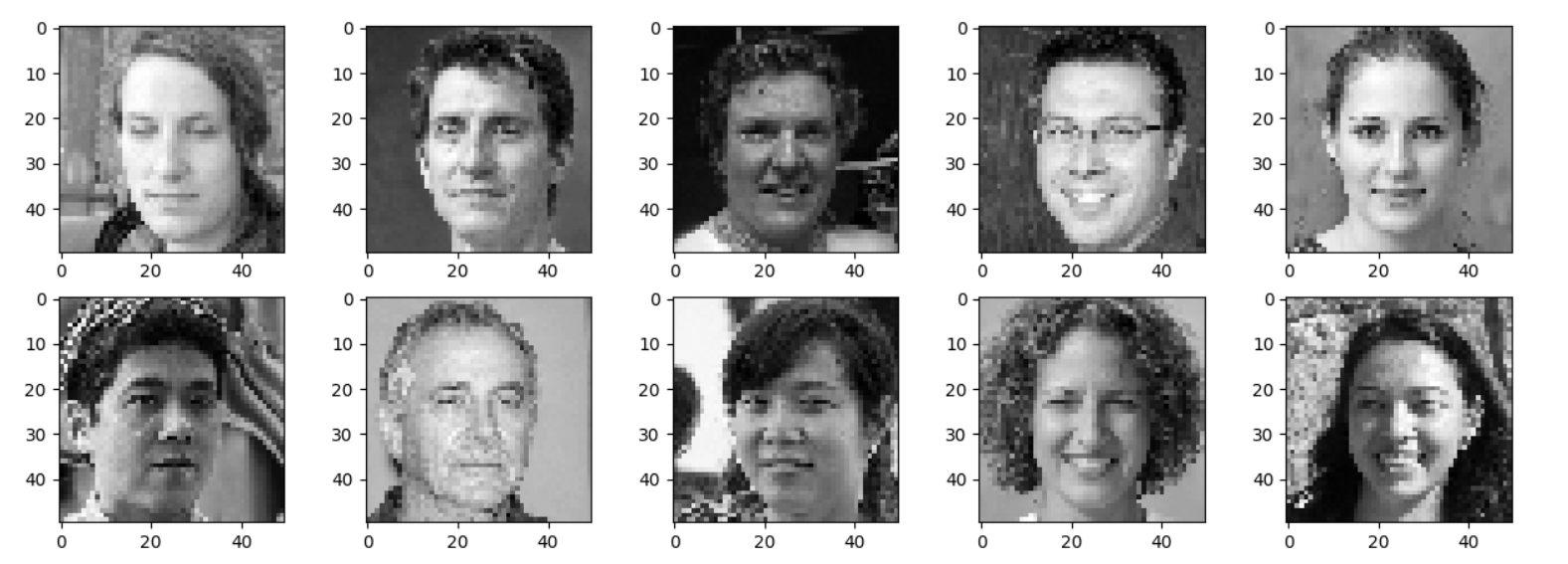
而对于其他方向，经过PCA后数据集都进行了投影（y轴和z轴的平面），如下所示：



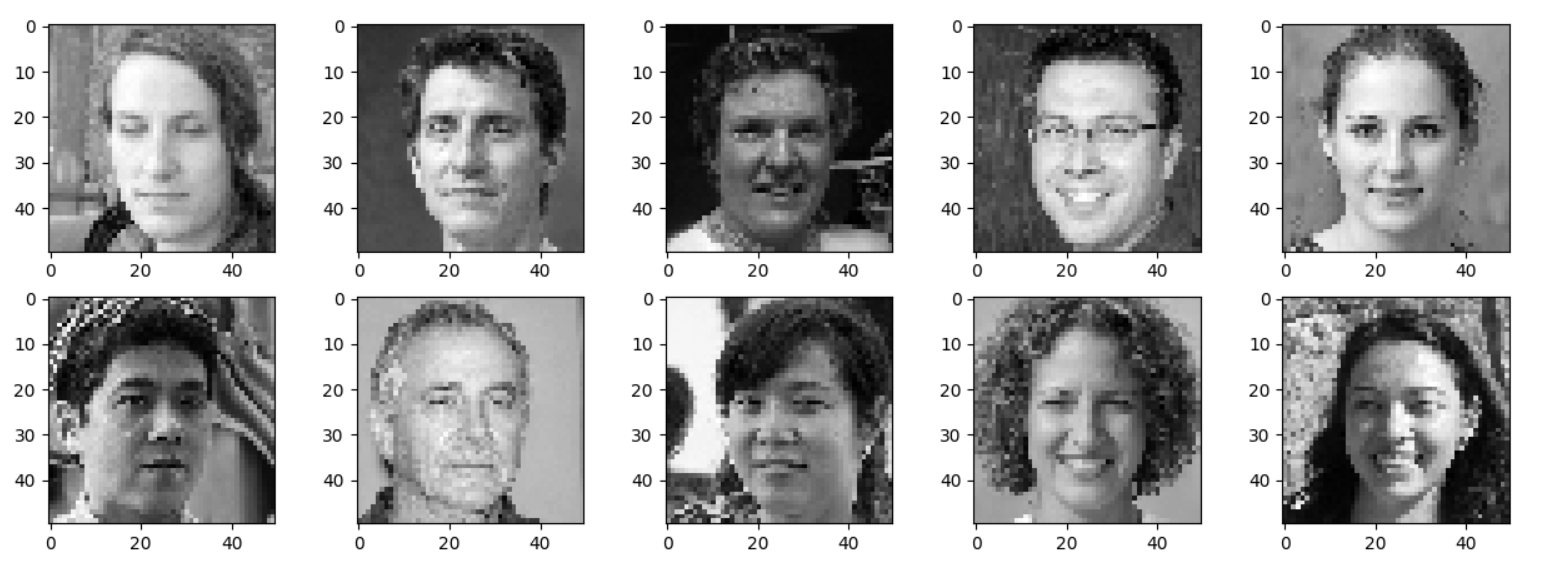
## 3.2使用人脸数据集进行测试

读取文件face\_collection中的所有图像进行测试，由于图片本身过大，需要将其压缩为size =（50,50）的大小，否则运行时间特别长。进行特征值提取，然后对于不同的降维维数进行测试。

（1）原图像如下：

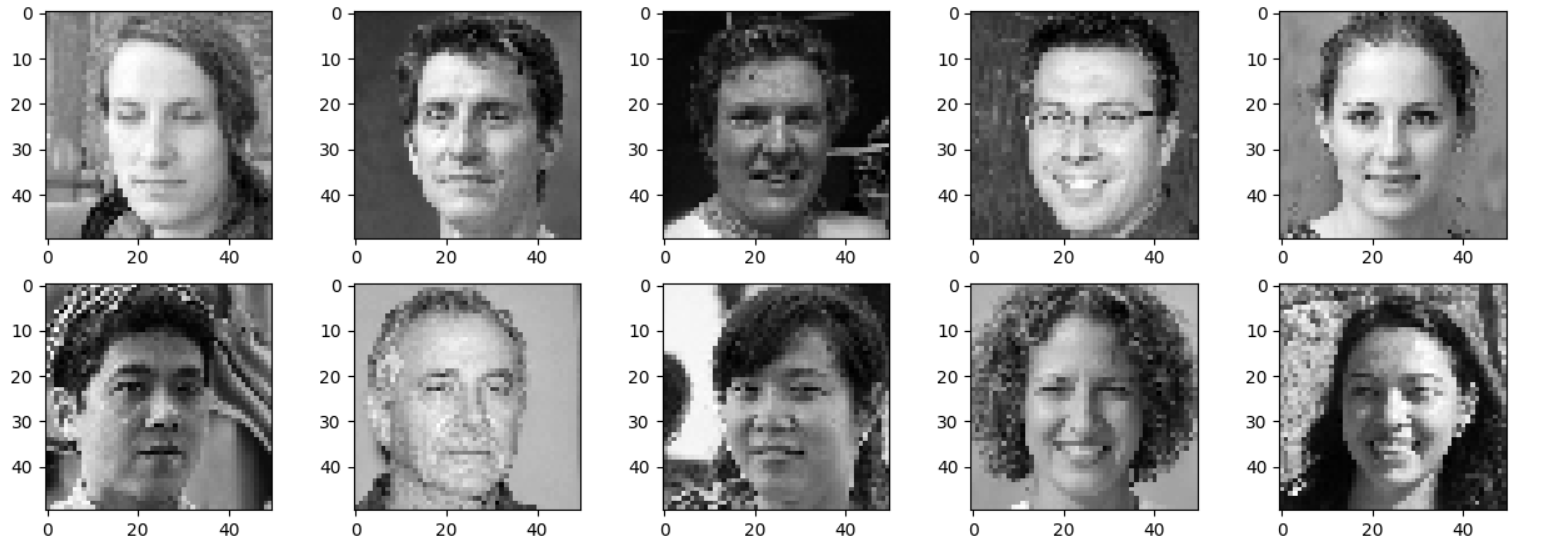


（2）降维至30维结果：



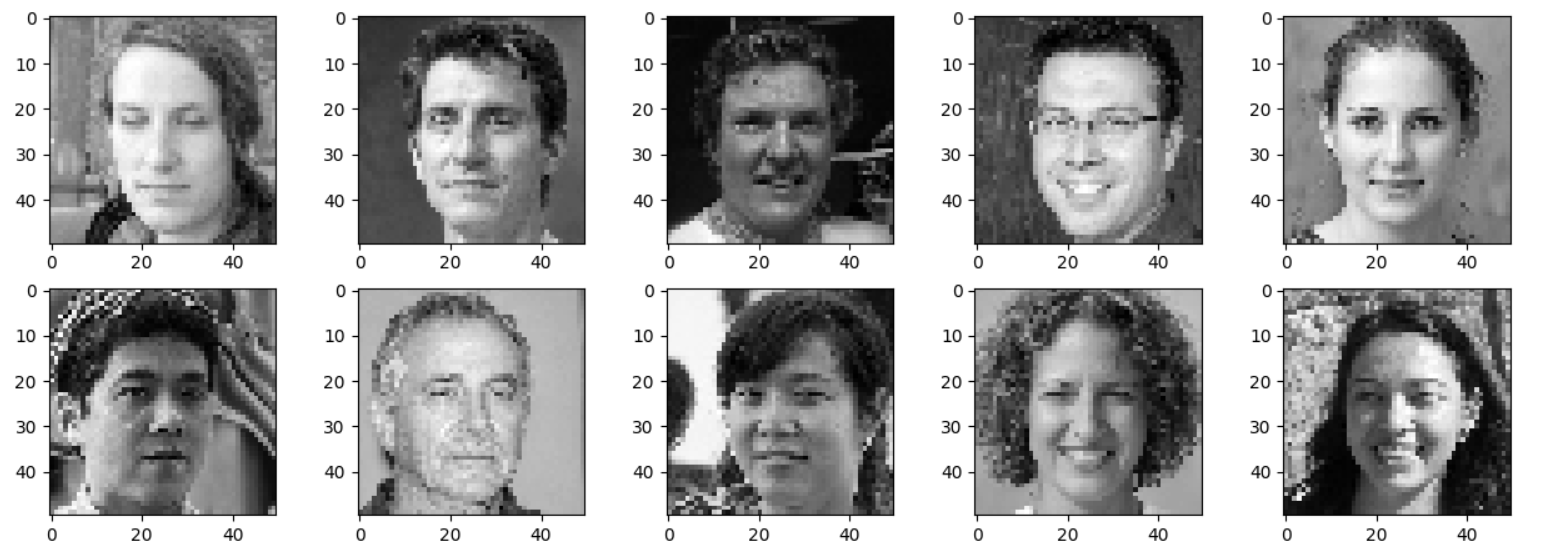


（3）降维至20维结果：



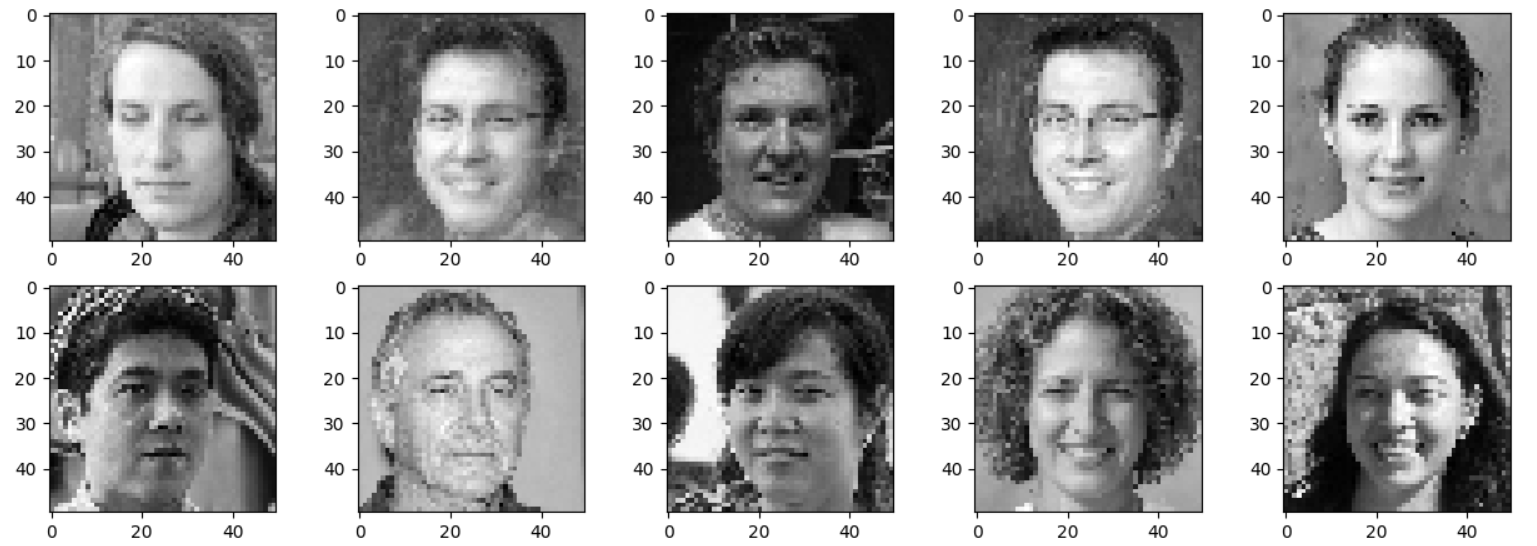


（4）降维至10维结果：



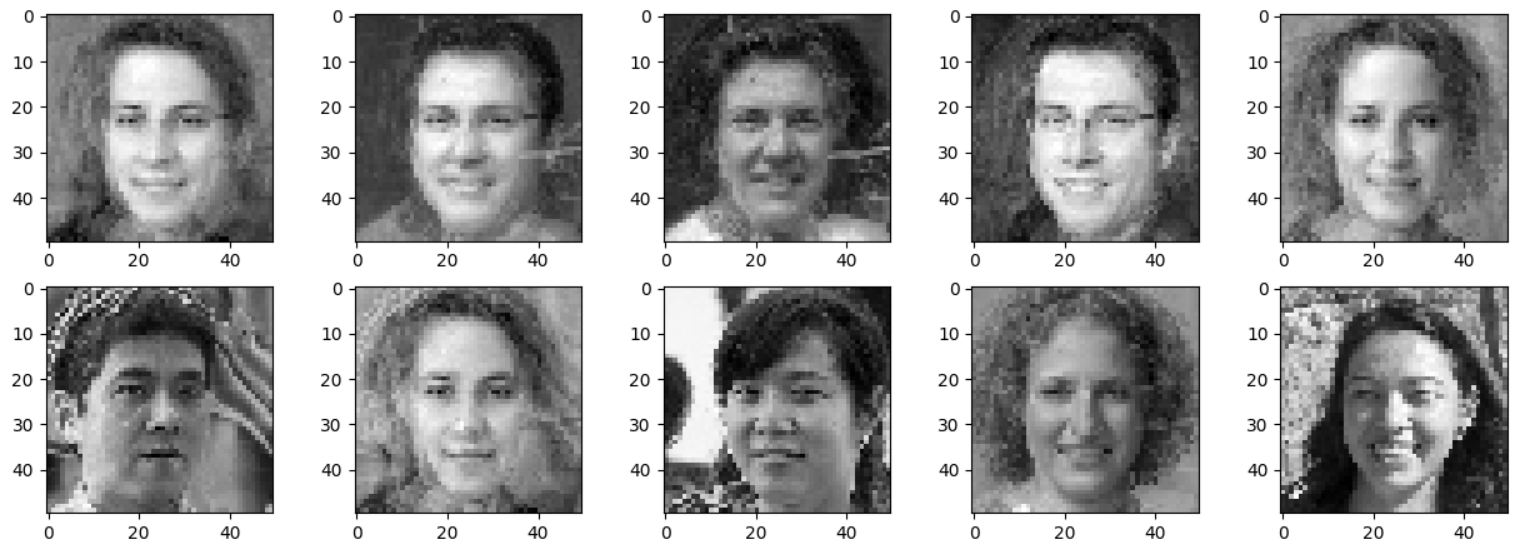


（5）降维至8维结果：



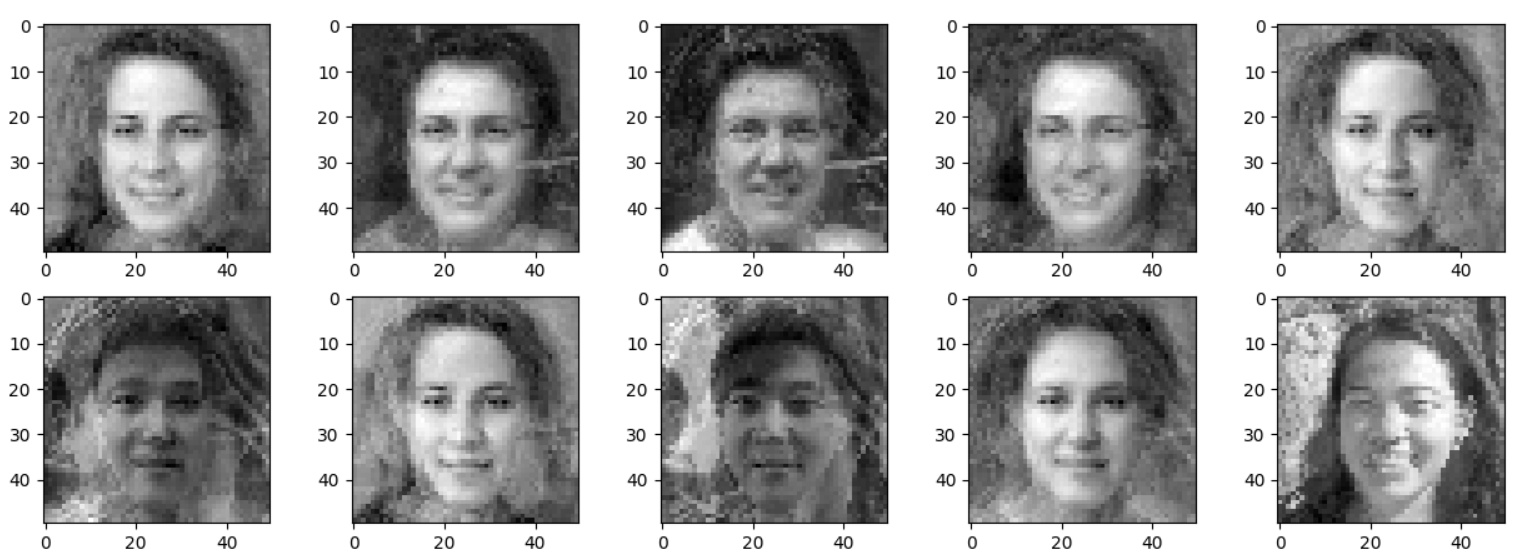


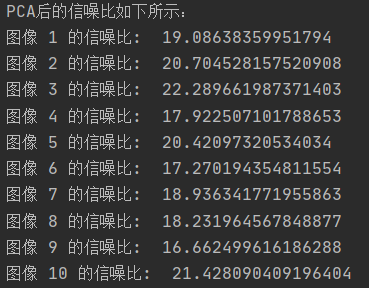
（6）降维至5维结果：



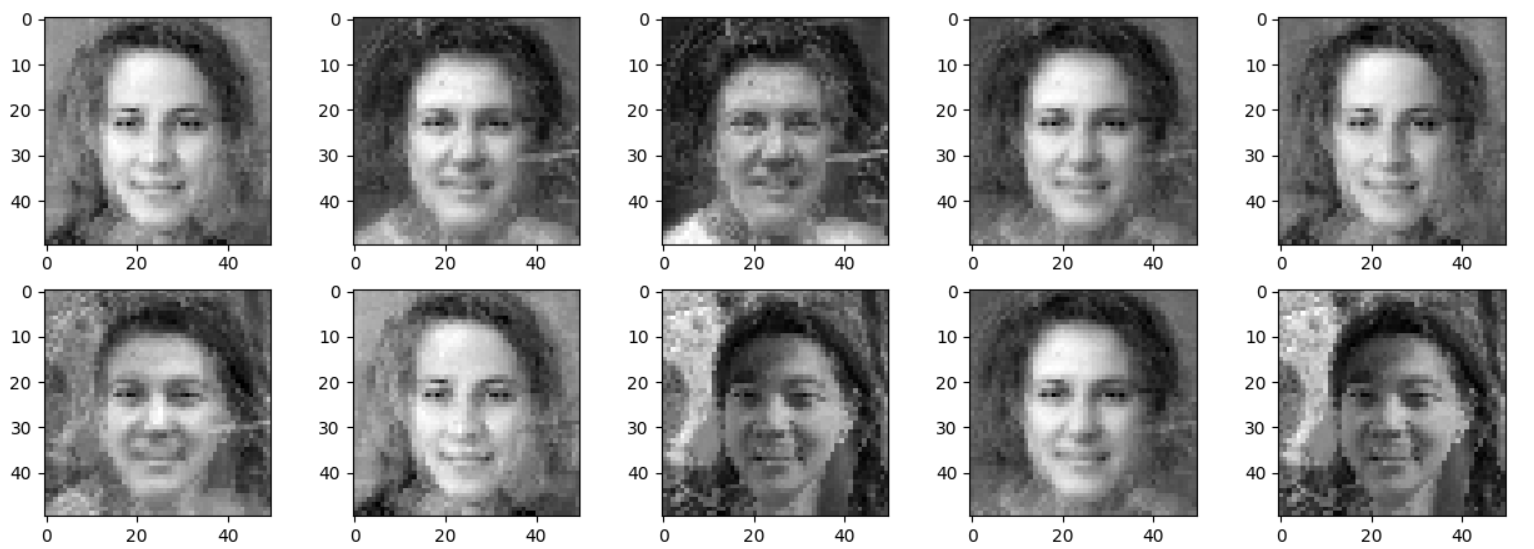


（7）降维至3维结果：



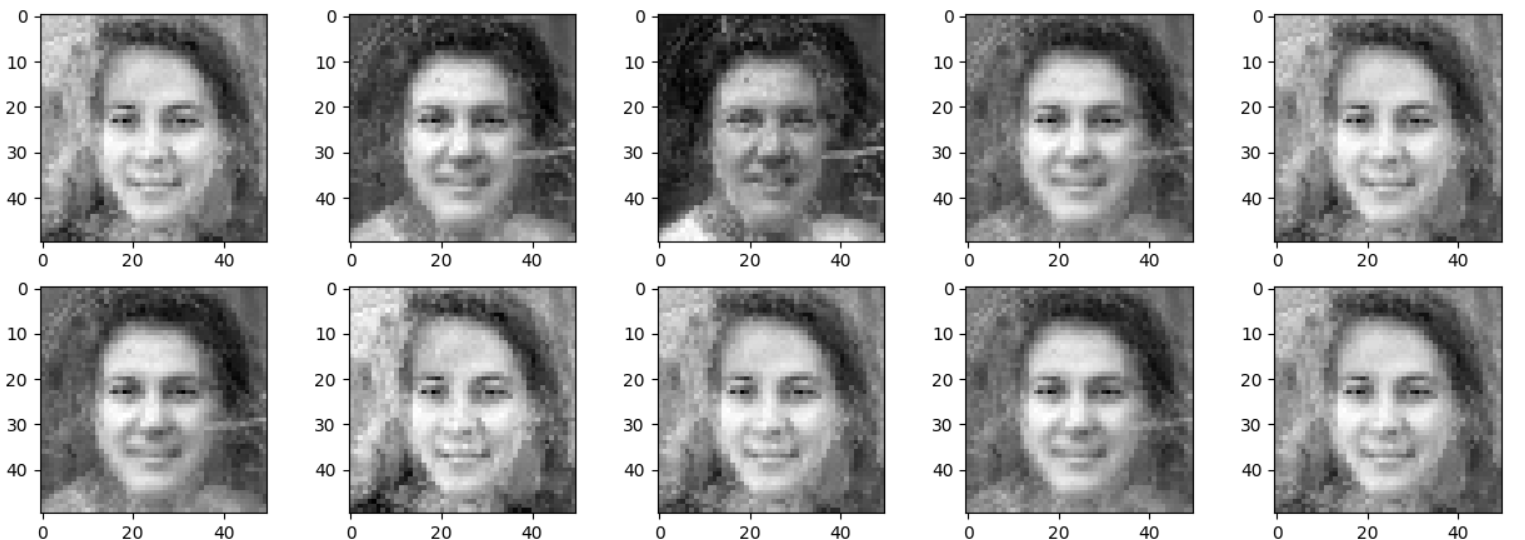


（8）降维至2维结果：





（9）降维至1维结果：





通过对比各个降低维度的变化可以看出：

随着维度的降低，重构出来的图片越来越失真，信噪比越来越低。在降低维数从30到10维之间图像都有极高还原度。而降低到8维就开始失真了，图二和图四混合在了一起。然后随着维度的降低，图像越来越奇怪，到1维的时候只剩下两种可以区分开的图像。

此外还可以发现，再一次降维中，压缩的10张图片有的信噪比高，有的信噪比低。

# 4 结论

1. PCA能够实现数据的压缩和高维数据的可视化。

2. PCA算法在降低维度的过程中会舍弃掉n-d个最小的特征值对应的特征向量，所以低维空间一定会与高维空间不同。但是由于n-d个特征值对原数据集的影响相对而言最小，因此这样方法有效地提高了样本的采样密度。

3.PCA算法虽然使得样本丢失了某些特征，但是却保留了主要的信息。虽然被丢弃的信息表面上没有用，但仅针对训练集上，在整体上可能会有很大作用。

4.PCA算法降维后的各维相互独立。

# 5 源代码

Lab4\_1180300829.py

1. **from** Lab4.all\_operation **import** create\_data\_by\_two\_or\_three\_dimension, read\_from\_file, PCA
2. **from** Lab4.draw **import** draw\_picture\_by\_image, draw\_picture\_by\_create\_PCA
4. # 用于生成数据的测试
5. dimension = 3
6. data\_num = 50
7. x = create\_data\_by\_two\_or\_three\_dimension(dimension, data\_num)
8. center\_data, w, mu\_x = PCA(x, dimension - 1)
9. x\_after\_PCA = (x - mu\_x).dot(w).dot(w.T) + mu\_x
10. **print**("中心化后的数据集为为:\n", center\_data)
11. **print**("特征向量矩阵为:\n", w)
12. **print**("降维前样本均值为:\n", mu\_x)
13. draw\_picture\_by\_create\_PCA(dimension, x, x\_after\_PCA)
15. # 用人脸图像进行测试
16. x = read\_from\_file('face\_collection')
17. x\_num, x\_dimension = x.shape  # 数据个数x\_num和维度x\_dimension
18. center\_data, w, mu\_x = PCA(x, 1)  # PCA降维
19. **print**("中心化后的数据集为为:\n", center\_data)
20. **print**("特征向量矩阵为:\n", w)
21. **print**("降维前样本均值为:\n", mu\_x)
22. draw\_picture\_by\_image(x, w, center\_data, mu\_x, x\_num)

draw.py

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **import** math

6. **def** draw\_picture\_by\_create\_PCA(dimension, data\_before\_PCA, x\_after\_PCA):
7. """
8. 对执行PCA前后的数据集，在图像上显示
9. :param dimension: 维度
10. :param data\_before\_PCA: 原始数据集
11. :param x\_after\_PCA: 执行PCA之后的数据集
12. :return:
13. """
14. **if** dimension == 2:  # 对二维数据画图
15. plt.scatter(data\_before\_PCA[:, 0], data\_before\_PCA[:, 1], facecolor="none", edgecolor="b",
16. label="data\_before\_PCA")
17. plt.scatter(x\_after\_PCA[:, 0], x\_after\_PCA[:, 1], facecolor='r', label='data\_after\_PCA')
18. plt.xlabel('x')
19. plt.ylabel('y')
20. **elif** dimension == 3:  # 对三维数据画图
21. fig = plt.figure()
22. ax = fig.gca(projection='3d')
23. ax.scatter(data\_before\_PCA[:, 0], data\_before\_PCA[:, 1], data\_before\_PCA[:, 2], edgecolor="b",
24. label='data\_before\_PCA')
25. ax.scatter(x\_after\_PCA[:, 0], x\_after\_PCA[:, 1], x\_after\_PCA[:, 2], facecolor='r', label='data\_after\_PCA')
26. ax.set\_xlabel('x')
27. ax.set\_ylabel('y')
28. ax.set\_zlabel('z')
29. **else**:
30. **assert** False
31. plt.legend()
32. plt.show()

35. **def** PSNR(image1, image2):
36. """
37. 计算两章图像的峰值信噪比PSNR
38. :param image1: 第一张图像
39. :param image2: 第二张图像
40. :return: 信噪比PSNR
41. """
42. mse = np.mean((image1 / 255. - image2 / 255.) \*\* 2)
43. **if** mse < 1.0e-10:
44. **return** 100
45. max\_pixel = 1  # 将最大像素设置为1
46. **return** 20 \* math.log10(max\_pixel / math.sqrt(mse))  # 计算信噪比即可

49. **def** draw\_picture\_by\_image(data\_set, w, center\_data, mu\_x, x\_num):
50. """
51. 输出PCA后的图像并打印信噪比
52. :param data\_set: PCA前的数据集
53. :param w: 特征向量矩阵
54. :param center\_data: 中心化后的数据
55. :param mu\_x: 降维前样本均值
56. :param x\_num: 数据个数
57. """
58. size = (50, 50)  # 由于较大的数据在求解特征值和特征向量时很慢，故统一压缩图像为size大小
59. w = np.real(w)  # 当降维后的维度超过某个值，特征向量矩阵将出现复向量，对其保留实部
60. x\_after\_PCA = np.dot(center\_data, w)  # 计算降维后的数据
61. refactoring\_data = np.dot(x\_after\_PCA, w.T) + mu\_x  # 重构降维后的数据
62. plt.figure(figsize=size)
63. **for** i **in** range(x\_num):
64. plt.subplot(3, 5, i + 1)
65. plt.imshow(refactoring\_data[i].reshape(size), cmap="gray")  # 预览该图像
66. plt.show()
67. **print**("PCA后的信噪比如下所示：")
68. **for** i **in** range(x\_num):  # 打印信噪比
69. psnr = PSNR(data\_set[i], refactoring\_data[i])
70. **print**('图像', i + 1, '的信噪比: ', psnr)

all\_operation.py

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **import** os
4. **import** cv2

7. **def** create\_data\_by\_two\_or\_three\_dimension(data\_dimension, num):
8. """
9. 生成三维或二维数据集
10. :param data\_dimension: 需要生成的维度
11. :param num: 需要生成的数据集的数据量
12. :return: 生成的数据集
13. """
14. **if** data\_dimension == 2:  # 对二维数据定义均值和方差
15. mean = [-2, 2]
16. cov = [[1, 0], [0, 0.01]]
17. **elif** data\_dimension == 3:  # 对三维数据定义均值和方差
18. mean = [1, 2, 3]
19. cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
20. **else**:
21. **assert** False
22. data\_set = []  # 定义数据集
23. **for** index **in** range(num):  # 生成num个数据，加入数据集
24. data\_set.append(np.random.multivariate\_normal(mean, cov).tolist())
25. **return** np.array(data\_set)

28. **def** PCA(data\_set, k):
29. """
30. 将数据集data\_set用PCA从D维降至k维，data\_set.shape = (N, D)
31. :param data\_set:原始数据集
32. :param k:PCA后的维度
33. :return:center\_data，中心化后的数据，shape=(N, D)。eigenvector\_matrix，特征向量矩阵，shape=(D, k)。data\_mean，降维前样本均值，shape=(1, D)
34. """
35. rows, cols = data\_set.shape  # 得到数据集的行和列
36. data\_mean = np.sum(data\_set, 0) / rows  # 计算降维前样本均值
37. center\_data = data\_set - data\_mean  # 进行数据集的中心化操作
38. covariance\_matrix = np.dot(center\_data.T, center\_data)  # 计算协方差矩阵X.T · X
39. eigenvalue, feature\_vectors = np.linalg.eig(covariance\_matrix)  # 对协方差矩阵(D,D)进行特征值分解，分别求得特征值和特征向量
40. eigenvalue\_sorted = np.argsort(eigenvalue)  # 将所有特征值排序
41. eigenvector\_matrix = feature\_vectors[:, eigenvalue\_sorted[:-(k + 1):-1]]  # 取出前k个最大的特征值对应的特征向量组成特征向量矩阵
42. **return** center\_data, eigenvector\_matrix, data\_mean

45. **def** read\_from\_file(file\_path):
46. """
47. 从文件中中读取面部图像数据并压缩
48. :param file\_path: 文件路径
49. :return: 返回解析面部图像得到的数据集
50. """
51. size = (50, 50)  # 由于较大的数据在求解特征值和特征向量时很慢，故统一压缩图像为size大小
52. i = 1
53. file\_list = os.listdir(file\_path)  # 读取该路径下所有图像的列表，放入file\_list
54. data\_set = []  # 定义数据集
55. plt.figure(figsize=size)
56. **for** file **in** file\_list:  # 对于file\_list中所有图像
57. path = os.path.join(file\_path, file)  # 连接文件路径，得到每个图像的路径
58. plt.subplot(3, 5, i)
59. with open(path) as f:
60. image = cv2.imread(path)  # 读取这张图像
61. image = cv2.resize(image, size)  # 将图像压缩至size大小
62. image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # 将图像通过三通道转换为灰度图
63. plt.imshow(image\_gray, cmap="gray")  # 预览该图像
64. h, w = image\_gray.shape  # 得到该图像的维度
65. image\_col = image\_gray.reshape(h \* w)  # 对(h,w)的图像数据拉平
66. data\_set.append(image\_col)  # 加入该图像数据给数据集中
67. i += 1
68. plt.show()
69. **return** np.array(data\_set)