哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： PCA 模型实验

学号：1190201421
姓名： 张瑞

一、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）。

二、实验要求及实验环境

（一）实验要求

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维度，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），用实现的 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，比较它们与原图像有多大差别（用信噪比衡量）。

（二）实验环境

Windows 10; PyCharm Community Edition 2021.2;Python 3.6

三、设计思想

PCA（主成分分析）从高维数据中提取一部分特征，根据这些特征进行从高维向低维的转换，常常用于数据压缩和高维数据可视化。

PCA 有两种解释形式，即最大方差形式和最小误差形式，实际上两者的结果是相同的。

（一）最大方差形式：

设有一组数据$X = \{x_1, x_2, \dots, x_n\}$，其中$x_i = \{x_{i1}, x_{i2}, \dots, x_{id}\}$，即 X 是$N \times D$的矩阵。若$u_1$是一个投影方向（单位基底，D 维向量，且满足$u_1^T u_1 = 1$），那么投影距离就是$x^T u_1$。

样本的均值为：

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

方差为：

$$S = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T = \frac{1}{N}X^T X$$

投影后的方差为：

$$\frac{1}{N}\sum_{n=1}^{N}\{u_1^T x_n - u_1^T \bar{x}\}^2 = u_1^T S u_1$$

于是需要最大化下式：

$$u_1^T S u_1$$

再加上u_1是单位基底的限制，根据拉格朗日乘子法有：

$$u_1^T S u_1 + \lambda(1 - u_1^T u_1)$$

对u_1求导并且令导数值为 0，得：

$$Su_1 = \lambda_1 u_1$$

于是u_1为 S 的特征向量，λ_1为 S 的特征值，得到最大化的值为：

$$u_1^T S u_1 = \lambda_1$$

于是，若想取得最大化的方差即要使得特征值最大。要将 D 维数据降到 K 维，只需要计算 S 的前 K 个最大特征值，将其对应的特征向量组成的矩阵$U_{D\times K}$构成新的基底，然后用原数据矩阵$X_{N\times D}$乘$U_{D\times K}$，即可得到压缩后的数据$Y_{N\times K}$，这样就把$X_{N\times D}$的原数据压缩为了$U_{D\times K}$的基底数据和$Y_{N\times K}$的压缩后数据，以更小的代价进行存储和传输。当恢复的时候，只需要将$Y_{N\times K}$乘上$U_{K\times D}^T$即可将数据规模恢复回来。

（二）最小误差形式：

考虑所有数据，希望所有样本在重构后与投影前的误差之和最小，即：

$$U = arg\min_U \sum_{i=1}^{n} \left|\left|x_i UU^T - x_i\right|\right|_2^2$$

$$= arg\min_U \sum_{i=1}^{n} ((x_i UU^T)(x_i UU^T)^T - 2(x_i UU^T)x_i^T + x_i x_i^T)$$

$$= arg\min_U \sum_{i=1}^{n} (x_i UU^T x_i^T - 2x_i UU^T x_i^T + x_i x_i^T)$$

$$= arg\min_U \sum_{i=1}^{n} (-x_i UU^T x_i^T + x_i x_i^T)$$

$$= arg\min_U -\sum_{i=1}^{n} x_i UU^T x_i^T$$

$$= arg\max_U \sum_{i=1}^{n} x_i UU^T x_i^T$$

$$= arg\max_U tr(U^T \left(\sum_{i=1}^{n} x_i^T x_i\right) U)$$

$$= arg\max_U tr(U^T X^T X U)$$

到此，可以发现最小误差形式得到的需要最大化的式子与最大方差形式是一样的，说明两种形式实际上是等价的，所求结果相同。

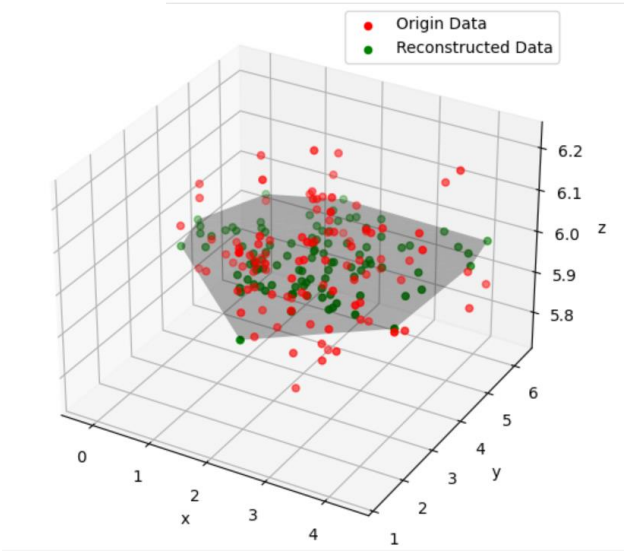
四、实验结果与分析

1.人工生成数据

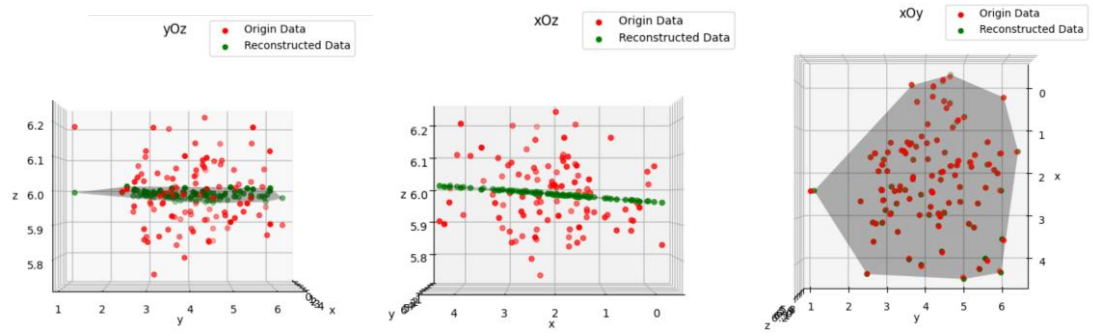
如下图参数所示生成三维数据，使得数据在第 3 个维度上的方差远远小于前 2 个维度，即数据在前 2 个维度上的特征比第 3 个维度更明显，属于更为主要的成分：

```
mus = [[2, 4, 6]]
sigmas = [[1, 1, 0.1]]
nums = [100]
```

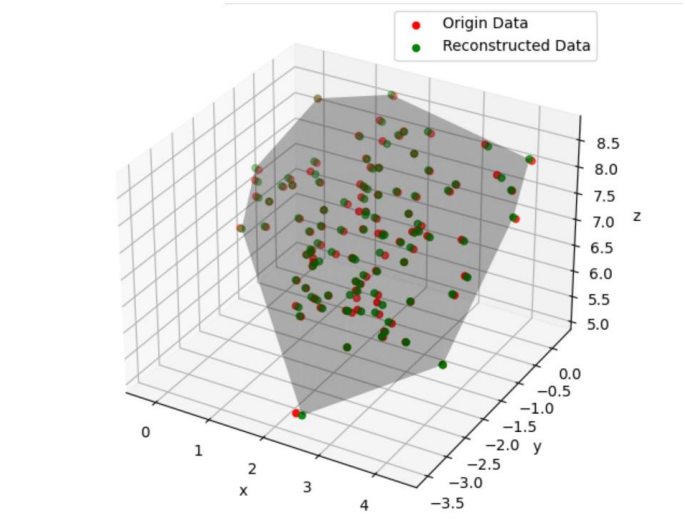
用 PCA 降维至 2 维后，再将数据重构，画图如下：



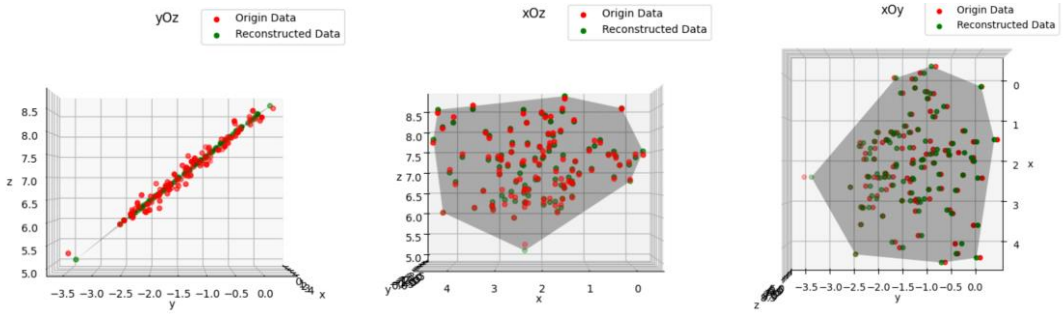
再观察三视图，发现重构数据主要保留的是前 2 维的特征，与生成数据的方差大小情况吻合：



若在生成数据之后，对数据进行旋转，例如使所有样本点绕 x 轴旋转$\frac{\pi}{4}$，其余参数同上，最终结果如下：



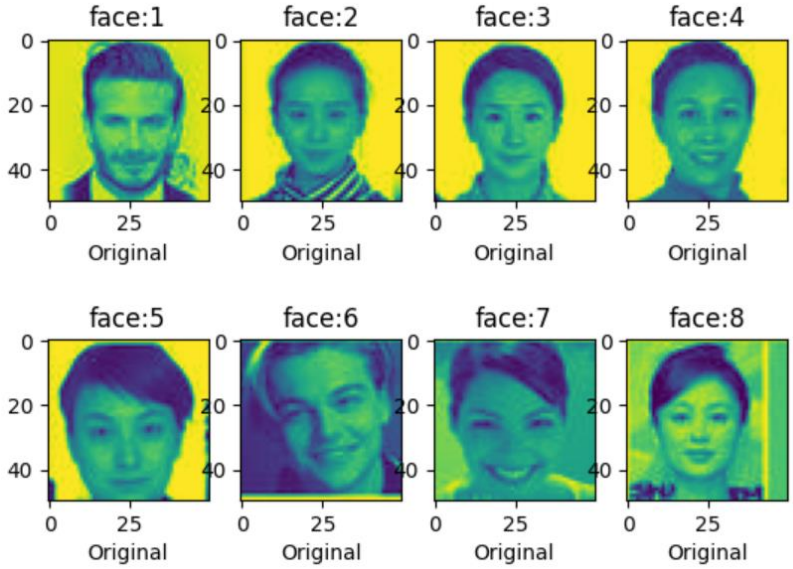
三视图如下：



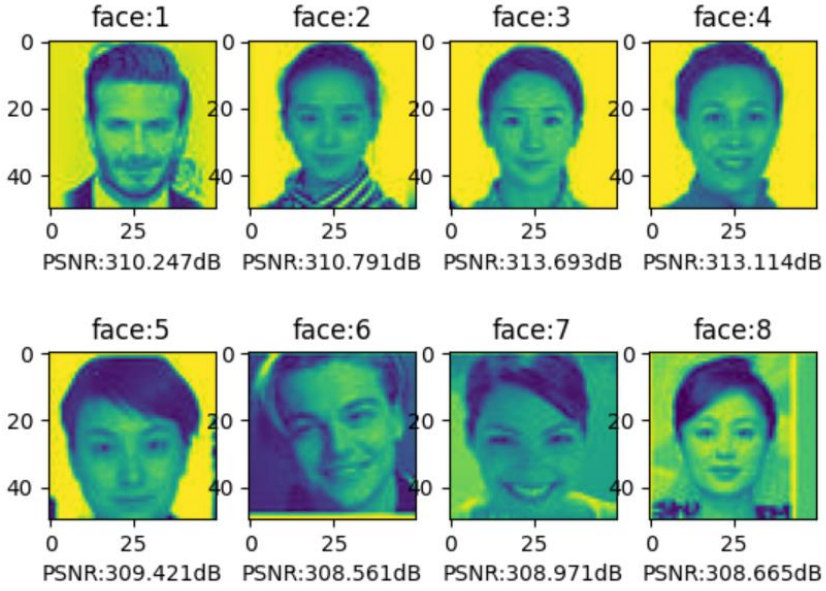
可以看到，由于旋转，导致第 3 个维度的特征变得更明显，最终保留的特征也更明显地包含了第 3 维度。

2.人脸数据

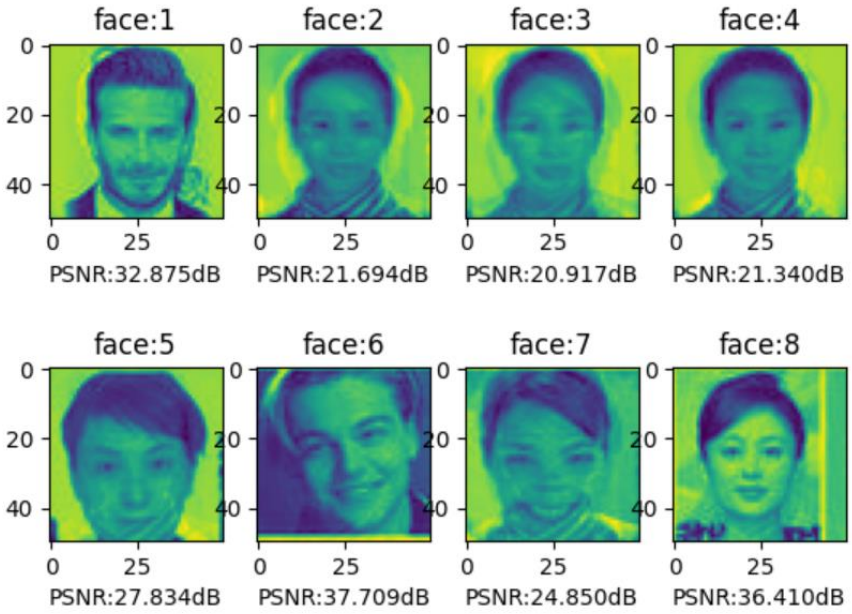
选取 8 张 50*50 的人脸图像进行降维，初始情况如下：



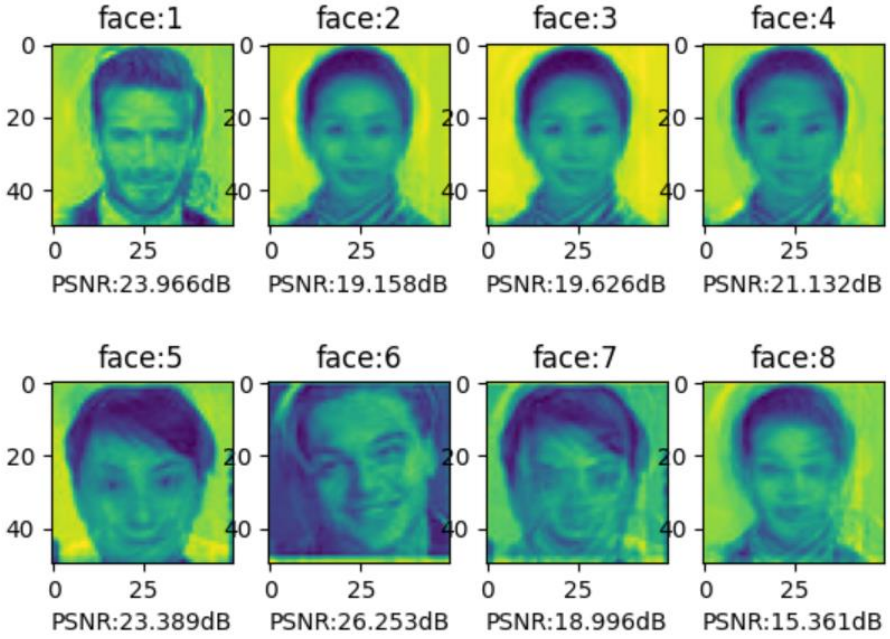
降到 10 维时：



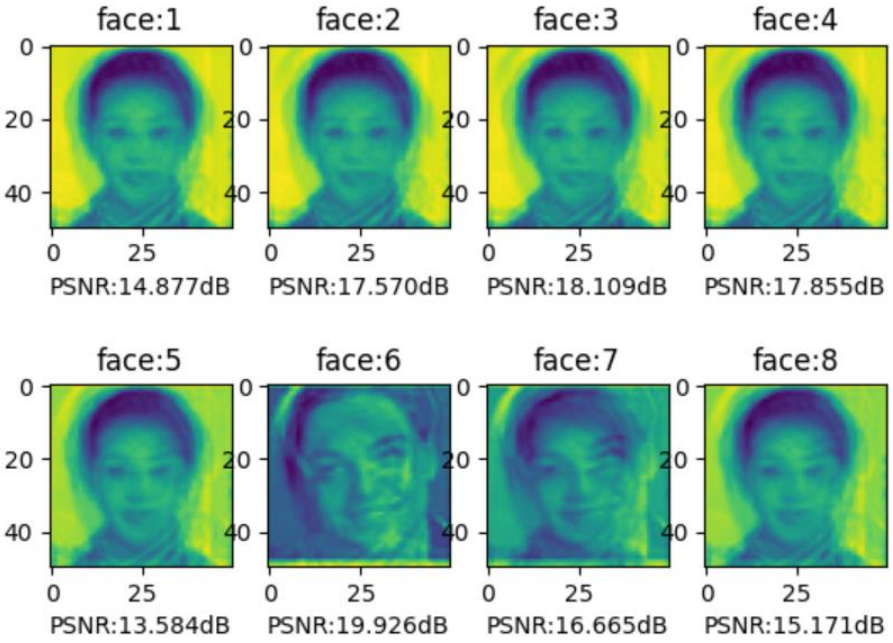
降到 5 维时：


face:1
face:2
face:3
face:4
PSNR:32.875dB
PSNR:21.694dB
PSNR:20.917dB
PSNR:21.340dB
face:5
face:6
face:7
face:8
PSNR:27.834dB
PSNR:37.709dB
PSNR:24.850dB
PSNR:36.410dB

降到 3 维时：


face:1
face:2
face:3
face:4
PSNR:23.966dB
PSNR:19.158dB
PSNR:19.626dB
PSNR:21.132dB
face:5
face:6
face:7
face:8
PSNR:23.389dB
PSNR:26.253dB
PSNR:18.996dB
PSNR:15.361dB

降到 1 维时：



可以看到，随着维度变小，信噪比逐渐变小，且不同人脸逐渐开始趋同（即特征丢失），越来越相像。

五、结论

PCA 是数据压缩和高维数据可视化的有效的手段。它能在降低数据维度的同时保留主要信息，使得数据在方差大的方向上保持良好，在方差小的方向上影响较大。而且 PCA 压缩数据时，是对较小特征值对应的特征向量进行舍弃，而较小特征值往往与噪声有关，这样一来 PCA 就能在一定程度上起到降噪的功能。但是，如果降的维度过多，可能会导致数据的特征丢失，样本趋同。同时，通过在人脸数据上的测试，我们能发现 PCA 对人脸图片的压缩效果是很好的，原本 50*50 的人脸数据在压缩到 10 之后依然很清晰可辨，这样一来图片便能很好地被存储，极大地减少内存压力。

六、参考文献

周志华.机器学习[M].北京：清华大学出版社，2016

七、附录：源代码（带注释）

1.main.py

```
import numpy as np

import mytool as mt
import pca


if __name__ == '__main__':
    np.random.seed(0)
```

```
    mus = [[2, 4, 6]]
    sigmas = [[1, 1, 0.1]]
    nums = [100]
    data = mt.generate_data(mus, sigmas, nums, dim=3)
    rotate_data = mt.rotate(data.T).T
    compressed_data, reconstructed_data = pca.pca(rotate_data, 2)
    mt.show_pca_result(rotate_data, reconstructed_data)

    data = mt.load_faces()
    for k in [10, 5, 3, 1]:
        compressed_data, reconstructed_data = pca.pca(data, k)
        mt.show_compressed_faces(data, reconstructed_data)
```

2.mytool.py

```
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
from PIL import Image


def generate_data(mus, sigmas, nums, dim):
    """
    生成高斯分布的数据
    :param mus: 高斯分布的均值
    :param sigmas: 高斯分布的标准差
    :param nums: 高斯分布的个数
    :param dim: 数据的维度
    :return: 生成的数据集
    """
    datasets = []
    for mu, sigma, num in zip(mus, sigmas, nums):
        dataset = np.random.randn(num, dim)
        for i, v in enumerate(sigma):
            dataset[:, i] *= sigma[i]
        for i, m in enumerate(mu):
            dataset[:, i] += mu[i]
        datasets.append(dataset)
    datasets = np.concatenate(datasets)
    return np.array(datasets, dtype=float)


def rotate(data, theta=0, axis='x'):
    """
```

```
    将数据进行旋转
    :param data: 待旋转数据
    :param theta: 旋转角度
    :param axis: 旋转轴
    :return: 旋转后的数据集
    """
    if axis == 'x':
        r = [[1, 0, 0], [0, np.cos(theta), -np.sin(theta)], [0,
np.sin(theta), np.cos(theta)]]
    elif axis == 'y':
        r = [[np.cos(theta), 0, np.sin(theta)], [0, 1, 0], [-
np.sin(theta), 0, np.cos(theta)]]
    else:
        r = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta),
np.cos(theta), 0], [0, 0, 1]]
    return np.dot(r, data)


def show_pca_result(data, reconstructed_data):
    """
    以 3D 图展示 PCA 降维后的数据与原始数据
    :param data: 原始数据
    :param reconstructed_data: 降维后的数据
    :return: 数据的三视图
    """
    # 降维前后 3D 图
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ax.scatter(data[:, 0], data[:, 1], data[:, 2], c="r",
label='Origin Data')
    ax.scatter(reconstructed_data[:, 0], reconstructed_data[:, 1],
reconstructed_data[:, 2], c='g',
               label='Reconstructed Data')
    ax.plot_trisurf(reconstructed_data[:, 0], reconstructed_data[:,
1], reconstructed_data[:, 2], color='k', alpha=0.3)
    ax.legend(loc='best')
    plt.show()
    plt.style.use('default')
    # 三视图
    for elev, azim, title in zip([0, 0, 90], [0, 90, 0], ["yOz",
"xOz", "xOy"]):
```

```
        fig = plt.figure()
        fig.suptitle(title)
        ax = Axes3D(fig)
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')
        ax.view_init(elev=elev, azim=azim)
        ax.scatter(data[:, 0], data[:, 1], data[:, 2], c="r",
label='Origin Data')
        ax.scatter(reconstructed_data[:, 0], reconstructed_data[:, 1],
reconstructed_data[:, 2], c='g',
                   label='Reconstructed Data')
        ax.plot_trisurf(reconstructed_data[:, 0],
reconstructed_data[:, 1], reconstructed_data[:, 2], color='k',
                        alpha=0.3)
        ax.legend(loc='best')
        plt.show()
        plt.style.use('default')


def load_faces():
    """
    读入初始人脸数据，并展示初始图像
    :return: 人脸数据组成的矩阵
    """
    data = []
    for i in range(8):
        img = Image.open('./faces/' + str(i + 1) + '.jpg')
        img = np.array(img)
        plt.subplot(2, 4, i + 1)
        plt.title("face:" + str(i + 1))
        plt.xlabel("Original")
        plt.imshow(img)
        data.append(img.reshape(50 * 50))
    plt.show()
    return np.array(data)


def show_compressed_faces(data, reconstructed_data):
    """
    展示 PCA 处理后的人脸图像，并计算信噪比
    :param data: 处理前数据
    :param reconstructed_data: 降维后数据
    :return: 处理后人脸图像
```

```
    """
    for i in range(8):
        plt.subplot(2, 4, i + 1)
        plt.title("face:" + str(i + 1))
        plt.xlabel("PSNR:%.3fdB" % cal_psnr(data[i],
reconstructed_data[i]))
        plt.imshow(reconstructed_data[i].astype(np.float).reshape((50,
50)))
    plt.show()


def cal_psnr(origin, compress):
    """
    计算图像信噪比
    :param origin: 原始图像
    :param compress: 压缩后图像
    :return: 图像信噪比
    """
    mse = np.mean((compress - origin) ** 2)
    psnr = 20 * math.log10(255 / math.sqrt(mse))
    return psnr
```

3.pca.py

```
import numpy as np


def pca(data, k):
    """
    用 PCA 对数据降维
    :param data: 待降维数据集
    :param k: 需要降到的维度
    :return: 降维后的数据以及恢复后的数据
    """
    m, n = np.shape(data)
    average = np.mean(data, axis=0)
    average_matrix = np.tile(average, (m, 1))
    data_adjust = data - average_matrix
    cov_matrix = np.cov(data_adjust.T)
    values, vectors = np.linalg.eig(cov_matrix)
    index = np.argsort(-values)
    if k > n:
        print("k should be smaller than the featrue")
        return
    else:
```

```
        select_vectors = vectors[:, index[:k]]
        select_vectors = np.real(select_vectors)  # 对于带实数的特征向量矩
阵，保留实部即可
        compressed_data = np.dot(data_adjust, select_vectors)
        reconstructed_data = np.dot(compressed_data, select_vectors.T)
+ average
    return compressed_data, reconstructed_data
```