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

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

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）。

实验环境：

硬件环境：x64CPU

软件环境：pycharm

库版本号：anaconda python==3.6, numpy==1.17.0

三、设计思想（本程序中的用到的主要算法及数据结构）

主成分分析是一种被广泛使用的技术，应用领域包括维度降低、有损数据压缩、特征抽取、数据可视化。PCA 可以被定义为数据在低维线性空间上的正交投影，这个线性空间被称为主子空间，使得投影数据的方差被最大化。等价地，它也可以被定义为使得平均投影代价最小的线性投影。平均投影代价是指数据点和它们投影之间的平均平方距离。

3.1 最大方差形式

考虑观测数据集 $\{\mathbf{x}_n\}_{n=1}^N$，我们的目标是将数据投影到维度 $M < D$ 的空间中，同时最大化投影数据的方差。首先考虑在一维空间投影即 $M = 1$ 的情形，使用 D 维向量 $\mathbf{u}_1$ 定义这个方向（假定选择单位向量 $\mathbf{u}_1{}^T\mathbf{u}_1 = 1$）。从而每个数据点 $\mathbf{x}_n$ 被投影到标量值 $\mathbf{u}_1{}^T\mathbf{x}_n$ 上，投影数据的均值是 $\mathbf{u}_1{}^T\overline{\mathbf{x}}$，其中 $\overline{\mathbf{x}}$ 是样本的均值，定义为 $\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^N \mathbf{x}_n$。则投影数据的方差为 $\frac{1}{N}\sum_{n=1}^N \{\mathbf{u}_1{}^T\mathbf{x}_n - \mathbf{u}_1{}^T\overline{\mathbf{x}}\}^2 = \mathbf{u}_1{}^T S\mathbf{u}_1$。其中 S 是样本数据的协方差矩阵，定义为 $S = \frac{1}{N}\sum_{n=1}^N (\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T$。

我们现在关于 $\mathbf{u}_1$ 最大化投影方差 $\mathbf{u}_1{}^T S\mathbf{u}_1$，限制条件是 $\mathbf{u}_1{}^T\mathbf{u}_1 = 1$。利用拉格朗日乘数法最大化的结果是：

$$S\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

等式两边同时左乘 $\mathbf{u}_1^T$ 得到：

$$\mathbf{u}_1^T S\mathbf{u}_1 = \lambda_1$$

因此，如果我们将 $\mathbf{u}_1$ 设置为与具有最大的特征值 λ_1 的特征向量相等时，方差会达到最大值。这个特征向量被称为第一主成分。

如果我们考虑 M 维的投影空间的一般情形，那么最大化投影数据方差的最优线性投影由数据协方差矩阵 S 的 M 个特征向量 $\{\mathbf{u}_i\}_{i=1}^M$ 来定义，对应 M 个最大特征值。

3.2 最小误差形式

PCA 的另一种形式为误差最小化的投影。引入 D 维基向量的一个完整的单位正交集合 $\{\mathbf{u}_i\}_{i=1}^D$，则每个数据点都可以表示为基向量的线性组合：

$$\mathbf{x}_n = \sum_{i=1}^{D} \alpha_{ni}\mathbf{u}_i$$

其中系数 α_{ni} 对于不同数据点来说是不同的。将等式两边同时乘 $\mathbf{u}_i$ 可得：

$$\alpha_{ni} = \mathbf{x}_n^T\mathbf{u}_i$$

代入 $\mathbf{x}_n$ 表达式则有：

$$\mathbf{x}_n = \sum_{i=1}^{D} (\mathbf{x}_n^T\mathbf{u}_i)\mathbf{u}_i$$

若要将数据 $\mathbf{x}_n$ 投影到 M 维空间，则投影后 $\mathbf{x}_n$ 可用前 M 个基向量表示。因此每个数据点可以近似为：

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D} b_i\mathbf{u}_i$$

其中 z_{ni} 与特定数据点有关，而 b_i 与数据点无关。接下来我们用平方距离度量原数据 $\mathbf{x}_n$ 与近似点 $\tilde{\mathbf{x}}_n$ 之间的距离：

$$J = \frac{1}{N}\sum_{n=1}^{N} ||\,\mathbf{x}_n - \tilde{\mathbf{x}}_n\,||^2$$

接下来最小化 z_{ni} 与 b_i 得：

$$z_{ni} = \mathbf{x}_n^T\mathbf{u}_i$$
$$b_i = \bar{\mathbf{x}}^T u_i$$

代入 J 表达式可得：

$$
\begin{aligned}
J &= \frac{1}{N}\sum_{n=1}^{N}||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}(\sum_{i=M+1}^{D}\{(\mathbf{x}_n - \overline{\mathbf{x}})^T\mathbf{u}_i\}\mathbf{u}_i)^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}(\mathbf{x}_n{}^T\mathbf{u}_i - \overline{\mathbf{x}}^T\mathbf{u}_i)^2 \\
&= \sum_{i=M+1}^{D}\mathbf{u}_i^T S\mathbf{u}
\end{aligned}
$$

在 $\mathbf{u}_i{}^T\mathbf{u}_i = 1$ 的条件下，利用拉格朗日乘数法最小化 J 得：

$$S\mathbf{u}_i = \lambda_i\mathbf{u}_i$$

两边左乘 $\mathbf{u}_i^T$ 并代入 J 的表达式得：

$$J = \sum_{i=M+1}^{D}\lambda_i$$

因此将这些特征向量选择为 $D-M$ 个最小特征值对应的特征向量即可得到 J 的最小值，因此定义主子空间的特征向量是对应于 M 个最大特征值的特征向量。

四、实验结果与分析

4.1 生成数据

根据给定均值与方差的高斯分布生成数据，其中维度即为输入均值和方差矩阵的维度。代码如下：

```python
def generate_data(mu, sigma, num_sample):
    """
    generate data
    :param mu: mean
    :param sigma: variance
    :param num_sample: number of samples in each class
    :return: generated data
    """
    x_dim = len(mu)
    data = []
    for i in range(num_sample):
        x = generate_gauss_data(mu, sigma, x_dim)
        data.append(x)
    return np.array(data)
```

图 1 生成数据部分代码

以三维数据的生成为例。生成数据的三维视图以及 xOy, xOz, yOz 方向的三视图如下所示：

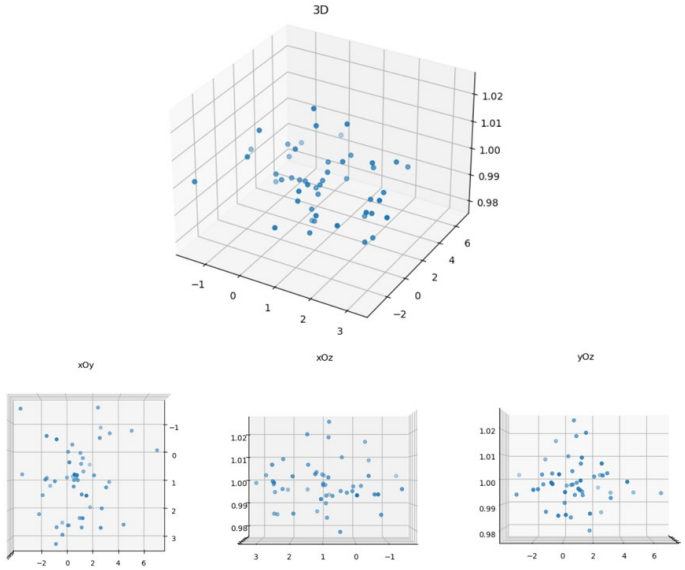

图 2 生成三维数据的三维图及三视图-1

这组数据在 x 方向和 y 方向的方差分别为 1 和 2，在 z 方向方差为 0.01。根据坐标我们也可以看出，在 x, y 方向波动较大，信息含量多，而在 z 方向波动较小。

同理，生成方差差距较大的二维数据如下所示：

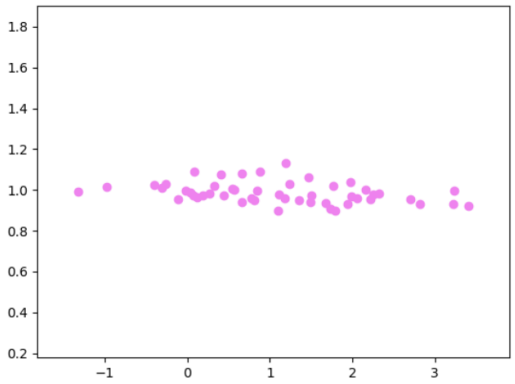

图 3 生成二维数据-1

4.2 使用 PCA 对数据进行降维

首先以二维情况为例。生成共 100 个数据点，这些样本在 x 方向均值为 0，方差为 1；在 y 方向均值为 0，方差为 0.05，如下图左所示；同时将其旋转如下图右所示：

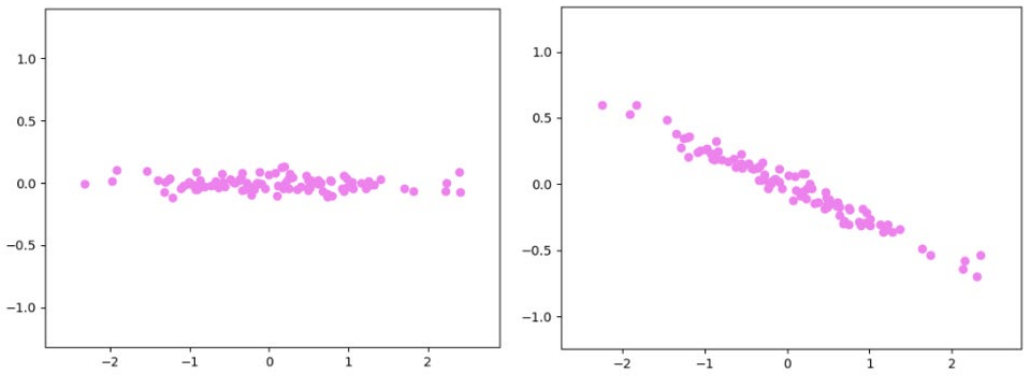

图 4 生成二维数据-2

在旋转后的数据上应用 PCA。注意为方便计算，在应用 PCA 前需对数据进行中心化处

理，之后再将二维数据压缩为一维，并根据公式 $\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D} b_i\mathbf{u}_i$ 对数据进行重建（由于中心化后均值为 0，可以将后一项省略）。将原数据和重建后的数据可视化，如下图所示：

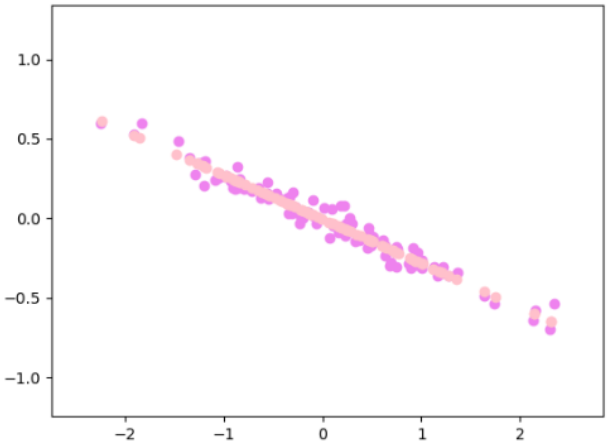

图 5 二维数据 PCA 结果可视化

可以看出，在方差较大的方向，信息基本保留；再方差小的方向，信息有所损失。

在三维的数据上进行实验。生成共 100 个数据点，设定 x, y, z 三个方向均值分别为 0, 0, 0；方差为 1, 2, 0.5，生成数据三维视图和三视图如下所示：

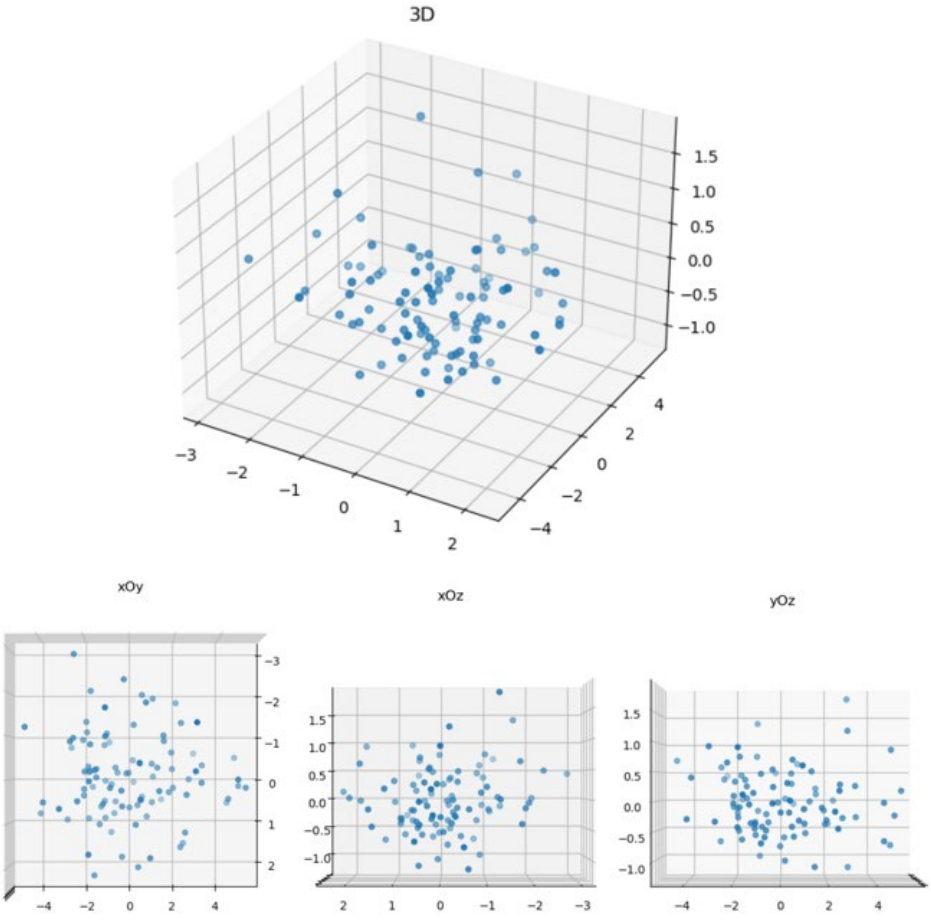

图 6 生成三维数据的三维图及三视图-2

同理，将三维数据点用 PCA 降维到二维并重建，得到的数据可视化如下所示：

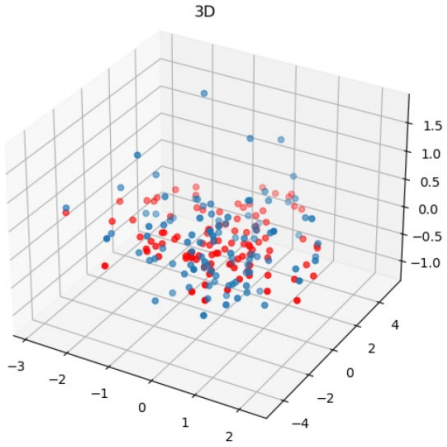

图 7 三维数据 PCA 结果可视化

从 xOy, xOz, yOz 方向分别可视化数据，如下图所示：

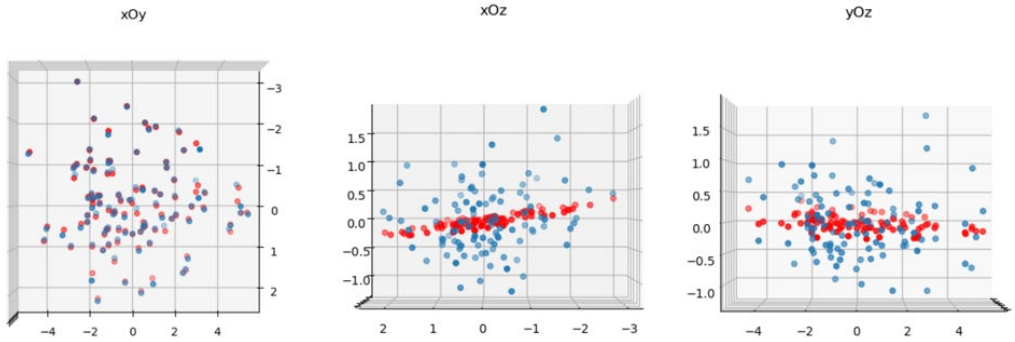

图 8 三维数据 PCA 结果三视图

从图中可以看出，在方差较大的 x 方向和 y 方向，数据信息损失很少，xOy 方向的三视图几乎中重建数据几乎与原数据重合。然而，在 z 方向，信息损失较多，从 xOz 方向和 yOz 方向的三视图也可以得出这一结论。这也正是 PCA 的特征，即在降维时，尽可能多地保留方差较大方向的信息，并会损失掉一些方差小方向的信息。

4.2 使用 PCA 对图片进行压缩

选取一组尺寸为 50*50 的图片共 6 张，原图片如下所示：

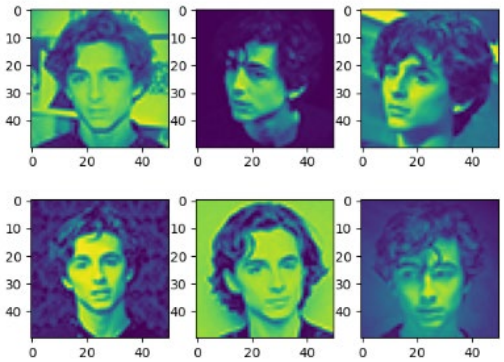

图 9 压缩前图片

取压缩后维度分别为 1, 2, 3, 4, 5，用 PCA 对原图片进行压缩，重建后计算信噪比并可视化，如下所示：

维度为 1 时：

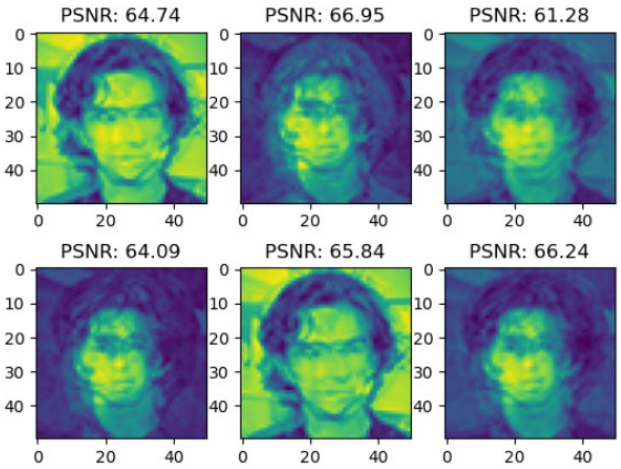

图 10 维度为 1 时压缩后图片

可以看出，此时维度较低，许多图片的信息都丢失了，信噪比也比较低，PCA 只能保留图片中差别比较大的一些特征。

维度为 2 时：

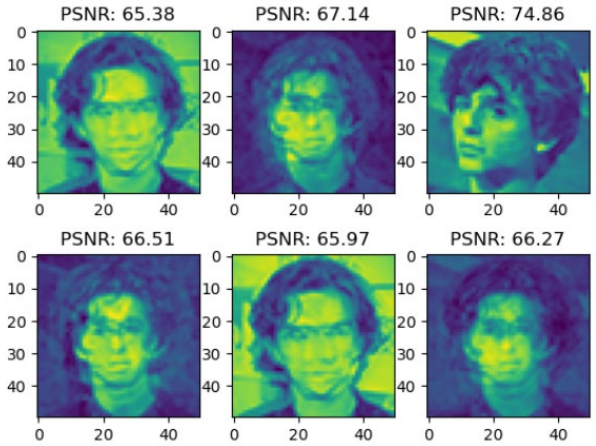

图 11 维度为 2 时压缩后图片

信噪比稍有提升，但此时维度仍然较低，仍然不能保留图片中足够的信息。

维度为 3 时：

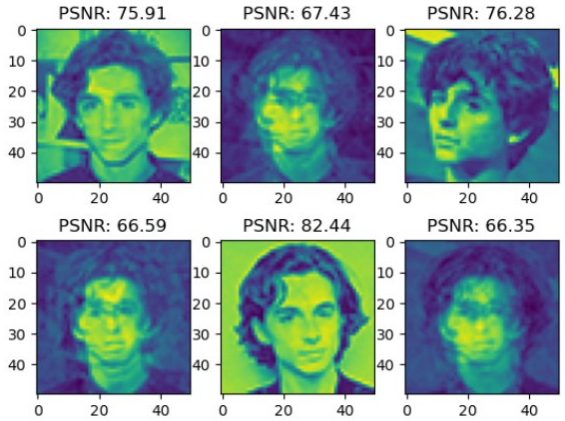

图 12 维度为 3 时压缩后图片

这时更多的图片已经变得清晰，信噪比进一步提升。

维度为 4 时：

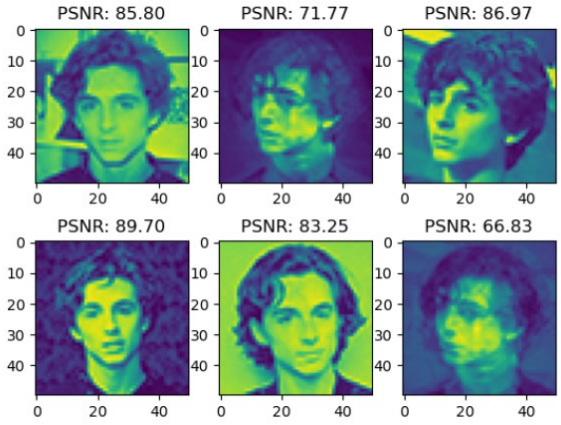

图 13 维度为 4 时压缩后图片

维度为 5 时：

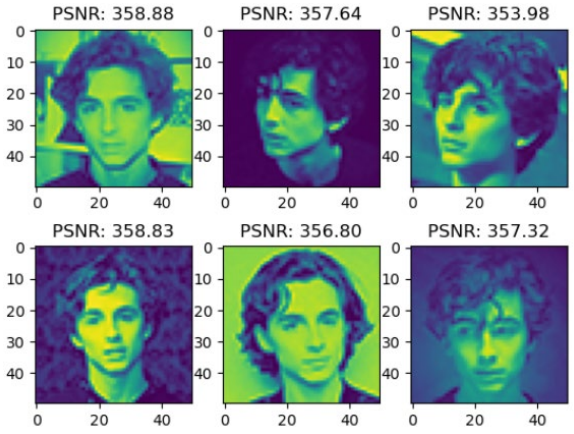

图 14 维度为 5 时压缩后图片

可以发现，维度为 5 时已经能够提取大部分特征，压缩后图片仍然比较清晰，信噪比较高。可见 PCA 能够去除图片中的方差较小的一些扰动，在保留最多信息的前提下尽可能压缩图片大小。

采用更多的图片进行实验，结果如下：

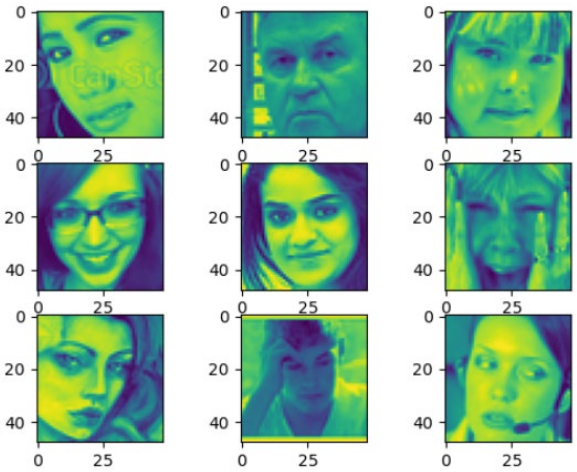

图 15 原图片

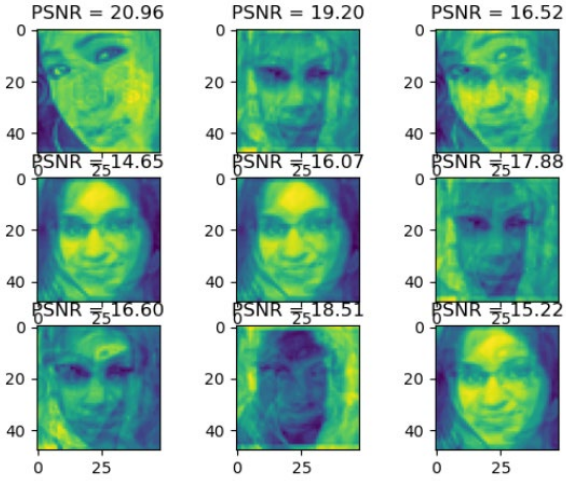

图 16 维度为 2 压缩结果

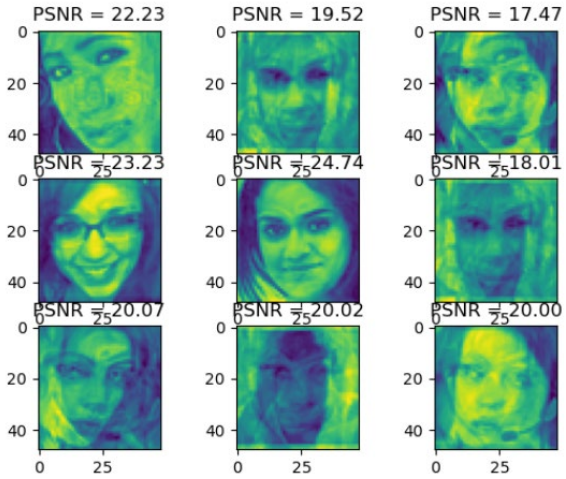

图 17 维度为 4 压缩结果

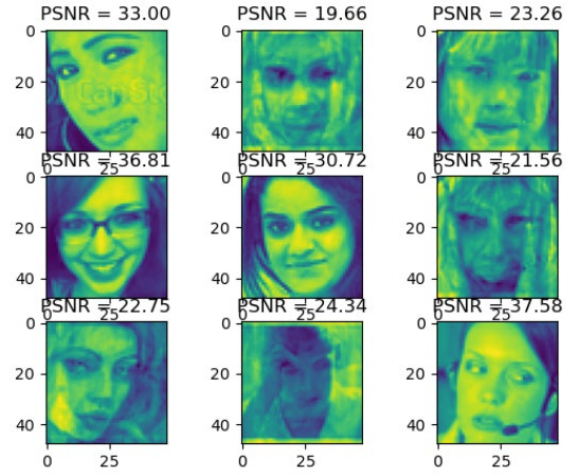

图 18 维度为 6 压缩结果

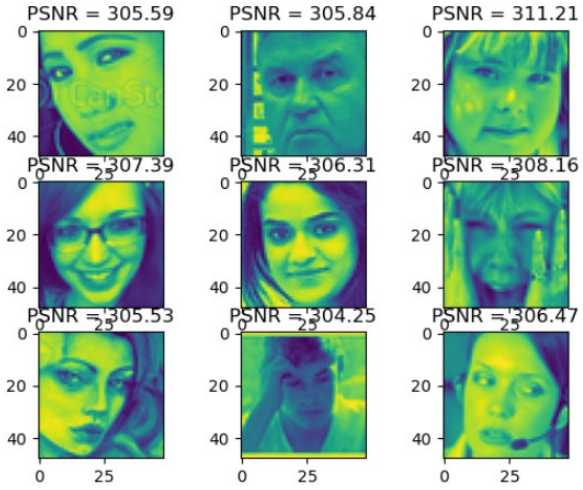

图 19 维度为 8 压缩结果

五、结论

PCA 是一种数据压缩、高维数据可视化的有效的手段。压缩后的数据在方差大的方向上保持良好，在方差小的方向上信息损失较多。

PCA 也可以对图片进行压缩，并且效果较好，可以将图片压缩到与原来相比较低维度的空间，达到较高的信噪比。

六、参考文献

[1] Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

[2] 周志华. 2016. 机器学习.

七、附录：源代码（带注释）

utils.py:

```
1. import numpy as np
2. import random
3. import matplotlib.pyplot as plt
4. import numpy.random
5. from scipy import linalg
6. from mpl_toolkits.mplot3d import Axes3D
7. import math
8.
9.
10. random.seed(0)
11. numpy.random.seed(0)
12.
13. def generate_gauss_data(mu, sigma, dim):
14.     """
15.     generate data that satisfy gauss distribution
```

```
    :param mu: mean
    :param sigma: variance
    :param dim: dimension of parameter x
    :return: generated data
    """
    x = []
    for j in range(dim):
        x.append(random.gauss(mu[j], sigma[j]))
    return x


def generate_data(mu, sigma, num_sample):
    """
    generate data
    :param mu: mean
    :param sigma: variance
    :param num_sample: number of samples in each class
    :return: generated data
    """
    x_dim = len(mu)
    data = []
    for i in range(num_sample):
        x = generate_gauss_data(mu, sigma, x_dim)
        data.append(x)
    return np.array(data)


def rotate_mat(axis, radian):
    """
    compute rotate matrix
    :param axis: axis
    :param radian: angle
    :return: rotate matrix
    """
    rot_matrix = linalg.expm(np.cross(np.eye(3), axis /
linalg.norm(axis) * radian))
    return rot_matrix


def rotate_2d_data(data, angle):
    rotate_matrix = np.asarray([[np.cos(angle), -np.sin(angle)],
[np.sin(angle), np.cos(angle)]])
    rotate_data = np.dot(data, rotate_matrix)
    return rotate_data
```

```
58.
59.
60.
61. def rotate_3d_data(data, axis):
62.     radian = math.pi / 3
63.     rotate_matrix = rotate_mat(axis, radian)
64.     rotate = np.dot(data, rotate_matrix)
65.     return rotate
66.
67.
68. def visualize_2d_data(data):
69.     assert data.shape[1] == 2
70.     plt.plot(data[:, 0], data[:, 1], 'ro', color='violet')
71.     plt.xlim([min(data[:, 0]) - 0.5, max(data[:, 0]) + 0.5])
72.     # plt.ylim([min(data[:, 0]) + 1.5, max(data[:, 0]) - 1.5])
73.     plt.ylim([min(data[:, 0]) + 1, max(data[:, 0]) - 1])
74.     plt.show()
75.
76.
77. def visualize_3d_data(data):
78.     assert data.shape[1] == 3
79.
80.     # projection in xOy, xOz, and yOz
81.     for elev, azim, title in zip([0, 0, 90], [0, 90, 0], ["yOz",
   "xOz", "xOy"]):
82.         fig = plt.figure()
83.         fig.suptitle(title)
84.         ax = Axes3D(fig)
85.         ax.view_init(elev=elev, azim=azim)
86.         ax.scatter(data[:, 0], data[:, 1], data[:, 2])
87.         plt.show()
88.
89.     # 3d image
90.     fig = plt.figure()
91.     fig.suptitle('3D')
92.     ax = Axes3D(fig)
93.     ax.scatter(data[:, 0], data[:, 1], data[:, 2])
94.     plt.show()
95.
96.
97. def visualize_difference_3d(data, recover):
98.     fig = plt.figure()
99.     fig.suptitle('3D')
100.        ax = Axes3D(fig)
```

```
101.         ax.scatter(recover[:, 0], recover[:, 1], recover[:, 2],
     c='r')
102.         ax.scatter(data[:, 0], data[:, 1], data[:, 2])
103.         plt.show()
104.
105.
106.     def visualize_difference(data, recover):
107.         for elev, azim, title in zip([0, 0, 90], [0, 90, 0], ["yOz",
     "xOz", "xOy"]):
108.             fig = plt.figure()
109.             fig.suptitle(title)
110.             ax = Axes3D(fig)
111.             ax.view_init(elev=elev, azim=azim)
112.             ax.scatter(recover[:, 0], recover[:, 1], recover[:, 2],
     c='r')
113.             ax.scatter(data[:, 0], data[:, 1], data[:, 2])
114.             plt.show()
115.
116.
117.     def visualize_difference_2d(data, recover):
118.         plt.plot(data[:, 0], data[:, 1], 'ro', color='violet')
119.         plt.plot(recover[:, 0], recover[:, 1], 'ro', color='pink')
120.         plt.xlim([min(data[:, 0]) - 0.5, max(data[:, 0]) + 0.5])
121.         plt.ylim([min(data[:, 0]) + 1, max(data[:, 0]) - 1])
122.         plt.show()
123.
124.
125.     def compute_psnr(origin, compress):
126.         assert origin.shape[0] == compress.shape[0]
127.         assert origin.shape[1] == compress.shape[1]
128.         r = origin.shape[0]
129.         c = origin.shape[1]
130.         mse = np.sum(np.square(origin - compress)) / (r * c)
131.         psnr = 20 * math.log10(255 / math.sqrt(mse))
132.         return psnr
133.
134.
135.     if __name__ == '__main__':
136.         # mu = [1, 1, 1]
137.         # sigma = [1, 2, 0.01]
138.         mu = [0, 0]
139.         sigma = [1, 0.05]
140.         num_sample = 100
141.         data = generate_data(mu, sigma, num_sample)
```

```
        # axis = np.array([1, 2])
        angle = np.pi / 12
        # data = rotate_2d_data(data, angle)
        # data = rotate_3d_data(data, axis)
        # visualize_3d_data(data)
        visualize_2d_data(data)
        # mean = np.mean(data, axis=0)
        # print(mean)
        # print(data - mean)
```

load.py:

```
import os
import struct
import numpy as np
import csv


def load_mnist():
    """Load MNIST data from `path`"""
    labels_path =
'./t10k-labels-idx1-ubyte/t10k-labels.idx1-ubyte'
    images_path =
'./t10k-images-idx3-ubyte/t10k-images.idx3-ubyte'
    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II',
                                 lbpath.read(8))
        labels = np.fromfile(lbpath,
                             dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack('>IIII',
                                               imgpath.read(16))
        images = np.fromfile(imgpath,
                             dtype=np.uint8).reshape(len(labels),
784)

    return images, labels


def load_fer():
    path = 'test.csv'
    with open(path) as f:
```

```
        reader = csv.reader(f)
        row_id = 0
        faces = []
        for row in reader:
            if row_id == 0:
                row_id += 1
                continue
            else:
                face = [float(s) for s in row[0].split(' ')]
                faces.append(face)
    return np.array(faces)


if __name__ == '__main__':
    load_fer()

```

pca.py:

```
import numpy as np


class PCA:
    def __init__(self, data, dim):
        self.raw_data = data
        self.mean_data = np.mean(data, axis=0)
        self.data = data - self.mean_data
        self.pre_dim = data.shape[1]
         self.target_dim = dim

    def compute_cov_matrix(self):
        cov = np.cov(self.data, rowvar=False)
        return cov

    def compute_direction(self):
        cov = self.compute_cov_matrix()
        cov.astype(np.float64)
        feature_value, feature_vector = np.linalg.eig(cov)
        # column feature_vector[:, i] is eigenvector corresponding
  to the eigenvalue feature_value[i].
        index = np.argsort(feature_value)[::-1]
        return feature_vector[:, index[:self.target_dim]]

    def reduce_dim(self):
```

```
        u = self.compute_direction()
        u = np.array(u, dtype=np.float64)
        target = np.dot(self.data, u)
        return target

    def recover_dim(self):
        u = self.compute_direction()
        u = np.array(u, dtype=np.float64)
        z = self.reduce_dim()
        recover = np.dot(z, u.T) + self.mean_data
        return recover

```

main.py:

```
import random

import matplotlib.pyplot as plt
import matplotlib.image as img
import numpy as np

from pca import PCA
import utils
from load import load_mnist
from load import load_fer

random.seed(0)
np.random.seed(0)


def testPCA_3d():
    mu = [0, 0, 0]
    sigma = [1, 2, 0.5]
    num_sample = 100
    data = utils.generate_data(mu, sigma, num_sample)
    # axis = np.array([1, 1, 2])
    # data = utils.rotate_3d_data(data, axis)
    utils.visualize_3d_data(data)
    expected_dim = 2
    model = PCA(data, expected_dim)
    recover = model.recover_dim()
    utils.visualize_difference_3d(data, recover)
    utils.visualize_difference(data, recover)

```

```
30.
31. def testPCA_2d():
32.     mu = [0, 0]
33.     sigma = [1, 0.05]
34.     num_sample = 100
35.     data = utils.generate_data(mu, sigma, num_sample)
36.     angle = np.pi / 12
37.     data = utils.rotate_2d_data(data, angle)
38.     utils.visualize_2d_data(data)
39.     expected_dim = 1
40.     model = PCA(data, expected_dim)
41.     recover = model.recover_dim()
42.     utils.visualize_difference_2d(data, recover)
43.
44.
45. def face_compression():
46.     faces = []
47.     for i in range(6):
48.         path = './chalemet/' + str(i + 1) + '.png'
49.         # path = './yalefaces/subject0' + str(i + 1) + '.happy'
50.         face = img.imread(path)[:, :, 0]
51.         # print(face.shape)
52.         plt.subplot(2, 3, i + 1)
53.         plt.imshow(face)
54.         face = np.array(face).reshape(50 * 50)
55.         face = face.astype(np.float64)
56.         faces.append(face)
57.     plt.show()
58.     faces = np.array(faces)
59.
60.     model = PCA(faces, 6)
61.     compress = model.recover_dim()
62.
63.     for i in range(6):
64.         image = compress[i].reshape((50, 50))
65.         origin_image = faces[i].reshape((50, 50))
66.         psnr = utils.compute_psnr(image, origin_image)
67.         plt.subplot(2, 3, i+1)
68.         plt.title('PSNR: %.2f' % psnr)
69.         plt.imshow(image)
70.     plt.show()
71.
72.
73. def compression_mnist():
```

```
    imgs, labels = load_mnist()
    imgs = imgs[:9]
    id = 1
    for img in imgs:
        plt.subplot(3, 3, id)
        id += 1
        image = img.reshape((28, 28))
        plt.imshow(image)
    plt.show()

    imgs = np.array(imgs)
    model = PCA(imgs, 2)
    compress = model.recover_dim()
    for i in range(9):
        image = compress[i].reshape((28, 28))
        origin_image = imgs[i].reshape((28, 28))
        psnr = utils.compute_psnr(image, origin_image)
        plt.subplot(3, 3, i + 1)
        plt.title('PSNR = %.2f' % psnr)
        plt.imshow(image)
    plt.show()


def compression_fer():
    faces = load_fer()
    faces = faces[1:10]
        id = 1
        for face in faces:
            plt.subplot(3, 3, id)
            id += 1
            image = face.reshape((48, 48))
            plt.imshow(image)
        plt.show()

        model = PCA(faces, 6)
        compress = model.recover_dim()
        for i in range(9):
            image = compress[i].reshape((48, 48))
            origin_image = faces[i].reshape((48, 48))
            psnr = utils.compute_psnr(image, origin_image)
            plt.subplot(3, 3, i + 1)
            plt.title('PSNR = %.2f' % psnr)
            plt.imshow(image)
        plt.show()
```

```
118.
119.
120.    if __name__ == '__main__':
121.        # testPCA_2d()
122.        # testPCA_3d()
123.        # face_compression()
124.        # compression_mnist()
125.        compression_fer()
126.
```