哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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

1.1 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

2 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$\left(x^{(1)}, x^{(2)}, \ldots, x^{(m)}\right)$都已经进行了中心化，即$\sum_{i=1}^{m} x^{(i)}=0$。经过投影变换后得到的新坐标系为$\{w_1, w_2, \ldots, w_n\}$，其中w是标准正交基，即$\| w \|_2=1$，$w_i^T w_j=0$
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

$\{w_1, w_2, \ldots, w_{n'}\}$，样本点$x^{(i)}$在n'维坐标系中的投影为：$z^{(i)}=\left(z_1^{(i)}, z_2^{(i)}, \ldots, z_{n'}^{(i)}\right)^T$。

其中，$z_j^{(i)}=w_j^T x^{(i)}$是$x^{(i)}$在低维坐标系里第j维的坐标。

对于任意一个样本$x^{(i)}$，在新的坐标系中的投影为$W^T x^{(i)}$，在新坐标系中的投影方差为$x^{(i)T} W W^T x^{(i)}$，要使所有的样本的投影方差和最大，也就是最大化$\sum_{i=1}^{m} W^T x^{(i)} x^{(i)T} W$的迹，即：

$$\underbrace{\arg max}_{W} \operatorname{tr}(W^T X X^T W) \text{ s.t. } W^T W=I$$

利用拉格朗日函数可以得到

$$J(W)=\operatorname{tr}\left(W^T X X^T W+\lambda(W^T W-I)\right)$$

对W求导有$X X^T W+\lambda W=0$，整理下即为：

$$X X^T W=(-\lambda) W$$

W为XX^T的n'个特征向量组成的矩阵，而−λ为XX^T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$z^{(i)}=W^T x^{(i)}$，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$x^{(i)}$的 n'维的主成分其实就是求样本集的协方差矩阵XX^T的前 n'个特征值对应特征向量矩阵 W，然后对于每个样本$x^{(i)}$,做如下变换$z^{(i)}=W^T x^{(i)}$，即达到降维的 PCA 目的。
具体的算法流程为：
输入：给定样本集$\left(x^{(1)}, x^{(2)}, \ldots, x^{(m)}\right)$，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：

$$\mu=\frac{1}{m} \sum_{j=1}^{m} \mathbf{x}_j$$

（1.2）所有样本减去均值得到中心化后的数据集：

$$x^{(i)} = x^{(i)} - \frac{1}{m}\sum_{j=1}^{m} x^{(j)}$$

（2）计算样本的协方差矩阵XX^T

（3）对矩阵XX^T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$(w_1, w_2, \dots, w_{n'})$，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。

```
def create_data_by_two_or_three_dimension(data_dimension, num):
```

```
    """
    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
    """
    if data_dimension == 2:  # 对二维数据定义均值和方差
        mean = [-2, 2]
        cov = [[1, 0], [0, 0.01]]
    elif data_dimension == 3:  # 对三维数据定义均值和方差
        mean = [1, 2, 3]
        cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
    else:
        assert False
    data_set = []  # 定义数据集
    for index in range(num):  # 生成 num 个数据，加入数据集
        data_set.append(np.random.multivariate_normal(mean, cov).tolist())
    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。

```
def PCA(data_set, k):
    """
    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
```

```python
    :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)
    """
    rows, cols = data_set.shape  # 得到数据集的行和列
    data_mean = np.sum(data_set, 0) / rows  # 计算降维前样本均值
    center_data = data_set - data_mean  # 进行数据集的中心化操作
    covariance_matrix = np.dot(center_data.T, center_data)  # 计算协方差矩阵 X.T · X
    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
    eigenvalue_sorted = np.argsort(eigenvalue)  # 将所有特征值排序
    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
    return center_data, eigenvector_matrix, data_mean
```

2.2.3 原始图片压缩的实现

作用：

```python
"""
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。

```python
def read_from_file(file_path):
    """
    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
    """
    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
    i = 1
    file_list = os.listdir(file_path)  # 读取该路径下所有图像的列表，放入 file_list
    data_set = []  # 定义数据集
    plt.figure(figsize=size)
    for file in file_list:  # 对于 file_list 中所有图像
        path = os.path.join(file_path, file)  # 连接文件路径，得到每个图像的路径
```

```
        plt.subplot(3, 4, i)
        with open(path) as f:
            image = cv2.imread(path)  # 读取这张图像
            image = cv2.resize(image, size)  # 将图像压缩至 size 大小
            image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # 将图像通过三通道转换为灰度图
            plt.imshow(image_gray)  # 预览该图像
            h, w = image_gray.shape  # 得到该图像的维度
            image_col = image_gray.reshape(h * w)  # 对(h,w)的图像数据拉平
            data_set.append(image_col)  # 加入该图像数据给数据集中
        i += 1
    plt.show()
    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。

```
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
"""
```

[redacted]

具体实现：调用绘制图像的函数即可。

```python
def draw_picture_by_create_PCA(dimension, data_before_PCA, x_after_PCA):
    """
    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:
    """
    if dimension == 2:  # 对二维数据画图
        plt.scatter(data_before_PCA[:, 0], data_before_PCA[:, 1], facecolor="none", edgecolor="b",
                    label="data_before_PCA")
        plt.scatter(x_after_PCA[:, 0], x_after_PCA[:, 1], facecolor='r', label='x_after_PCA')
        plt.xlabel('x')
        plt.ylabel('y')
    elif dimension == 3:  # 对三维数据画图
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.scatter(data_before_PCA[:, 0], data_before_PCA[:, 1], data_before_PCA[:, 2], edgecolor="b",
                   label='data_before_PCA')
        ax.scatter(x_after_PCA[:, 0], x_after_PCA[:, 1], x_after_PCA[:, 2], facecolor='r', label='x_after_PCA')
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')
    else:
        assert False
    plt.legend()
```

```python
def draw_picture_by_image(data_set, w, center_data, mu_x, x_num):
    """
    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
    """
```

```
    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
    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
    x_after_PCA = np.dot(center_data, w)  # 计算降维后的数据
    refactoring_data = np.dot(x_after_PCA, w.T) + mu_x  # 重构降维后的数据
    plt.figure(figsize=size)
    for i in range(x_num):
        plt.subplot(3, 4, i + 1)
        plt.imshow(refactoring_data[i].reshape(size))  # 预览该图像
    plt.show()
    print("PCA 后的信噪比如下所示：")
    for i in range(x_num):  # 打印信噪比
        psnr = PSNR(data_set[i], refactoring_data[i])
        print('图像', i + 1, '的信噪比：', psnr)

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。

```
def PSNR(image1, image2):
    """
    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
    """
    mse = np.mean((image1 / 255. - image2 / 255.) ** 2)
    if mse < 1.0e-10:
        return 100
    max_pixel = 1  # 将最大像素设置为 1
    return 20 * math.log10(max_pixel / math.sqrt(mse))  # 计算信噪比即可
```

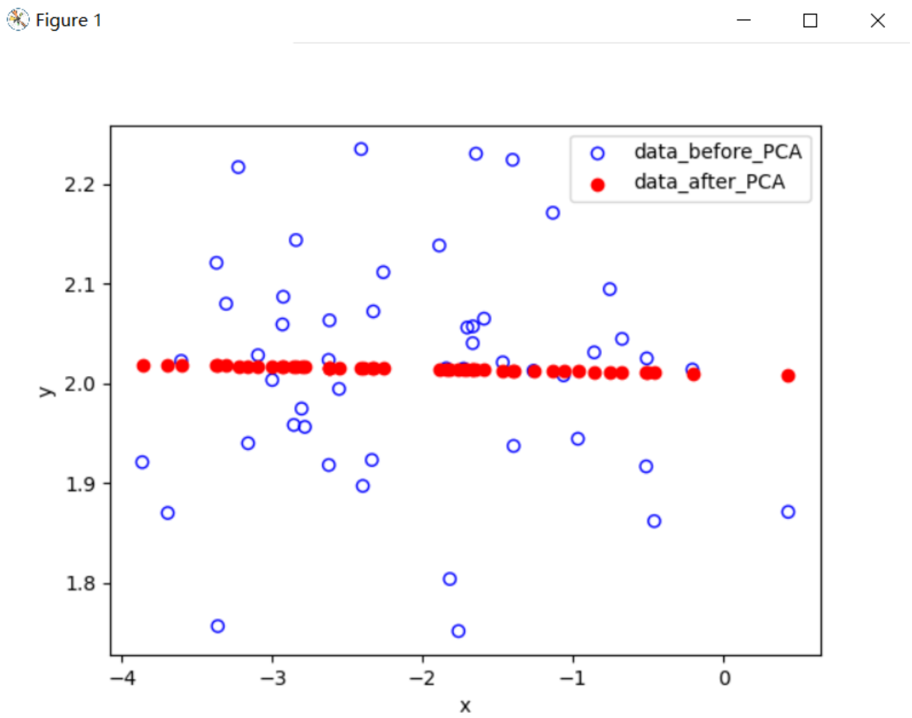
3 实验结果分析

3.1 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 $\begin{bmatrix} 0.01 & 0 \\ 0 & 1 \end{bmatrix}$

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：

```
中心化后的数据集为为：
 [[-7.27556683e-01  5.51241395e-02]
 [ 1.29631048e+00 -5.39390592e-02]
 [-7.82608302e-01  4.08518740e-02]
 [-2.84366874e-01 -6.61254808e-05]
 [-1.51856505e+00 -4.71153231e-02]
 [ 3.22294201e-02 -9.79843457e-02]
 [ 1.62720184e-01 -6.65224197e-02]
```

……

```
 [ 9.08831407e-01 -9.76824054e-02]
 [-9.57007817e-01 -7.24260327e-03]
 [-1.82547283e-01 -1.37090478e-02]
 [-1.45004478e+00  4.87258215e-02]
 [ 2.15703854e+00 -1.31673693e-01]
 [-7.26790935e-01 -2.31070672e-02]
 [ 3.56054345e-01 -3.79342467e-04]
 [ 1.76826964e-01  6.78643685e-02]]
特征向量矩阵为：
 [[0.99999821]
 [0.00189053]]
降维前样本均值为：
 [-1.80106805  2.02506468]
```

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 $\begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

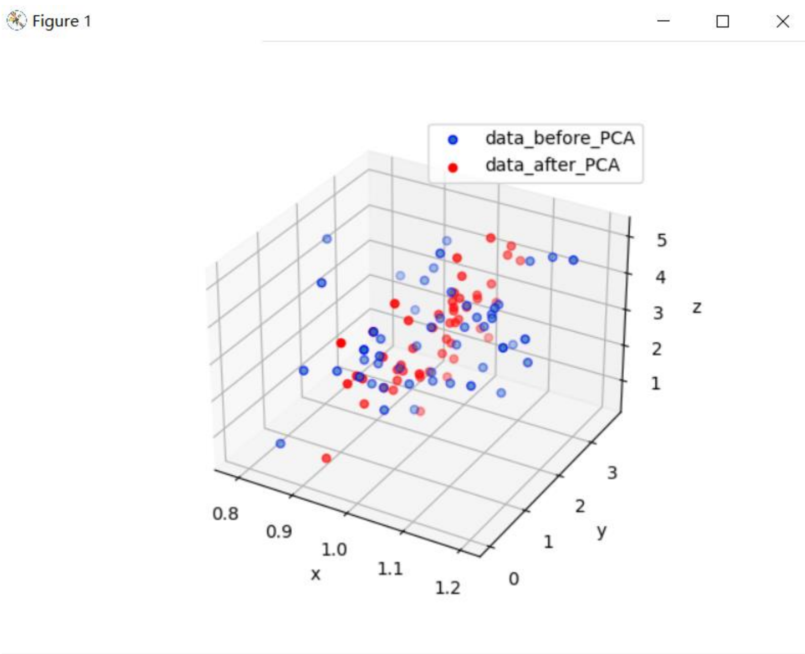
在这里的数据中，第一维的反差远小于另外两维的方差，所以第一维包含了的信息很少，直接 PCA 即可

输出为

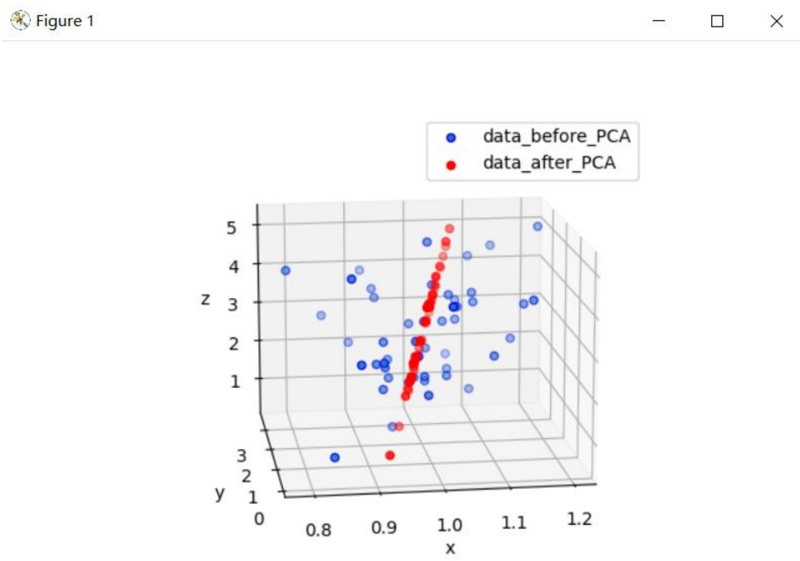
```
中心化后的数据集为为：
 [[-2.11028025e-02  1.60884136e+00 -4.25056336e-01]
 [-3.82880990e-03  8.01128774e-01 -5.05100826e-01]
 [ 6.03520649e-02  1.77394731e+00  1.16796882e-01]
 [ 3.07341072e-01  9.56511254e-01 -1.27105121e+00]
 [ 6.72688316e-02 -1.04947158e+00 -1.53342116e+00]
```

……

```
 [-1.69798291e-01 -2.89985684e+00  1.93667917e+00]
 [-5.44843884e-03 -3.42083217e-01  1.29347861e+00]]
特征向量矩阵为：
 [[ 2.85848151e-02  6.09699751e-04]
 [ 7.13891546e-01  6.99946057e-01]
 [-6.99672616e-01  7.14195453e-01]]
降维前样本均值为：
 [1.00035303 1.77230723 2.89782046]
```

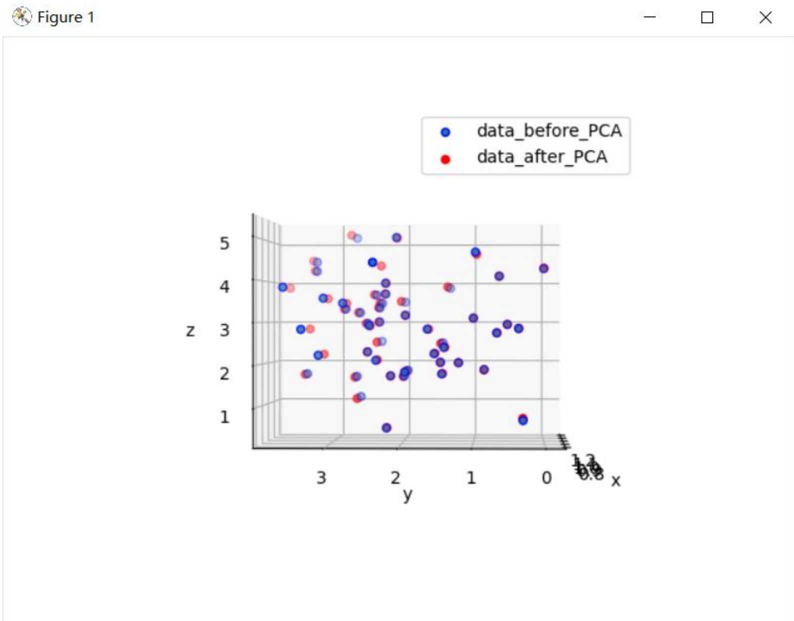


在图中可以看出在 x 轴的单位长度表示的长度更小，说明表示的原数据集上的第一维数据，然后对图像旋转得到如下：



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。

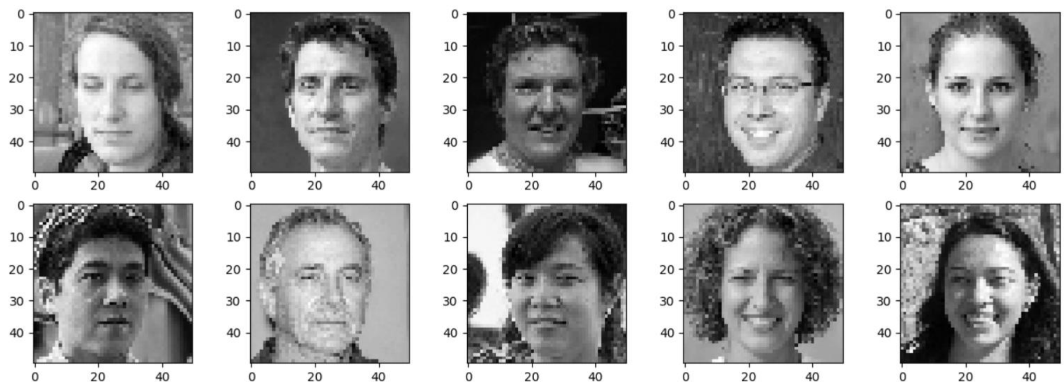
而对于其他方向，经过 PCA 后数据集都进行了投影（y 轴和 z 轴的平面），如下所示：



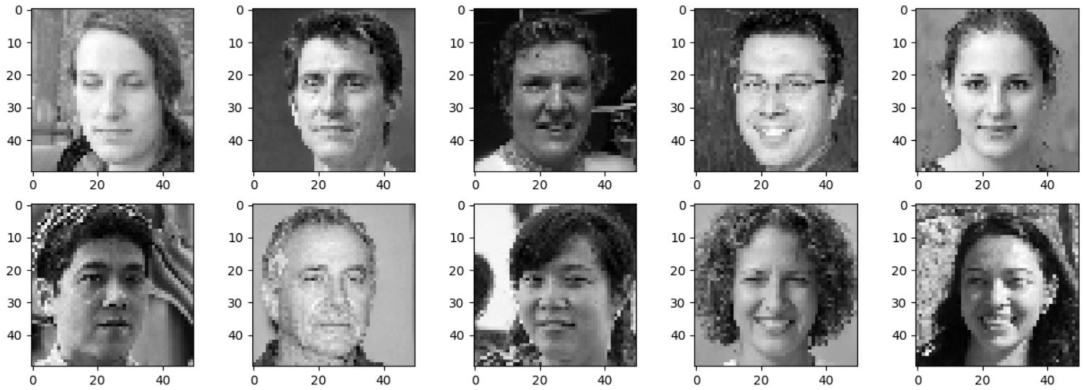
3.2 使用人脸数据集进行测试

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：

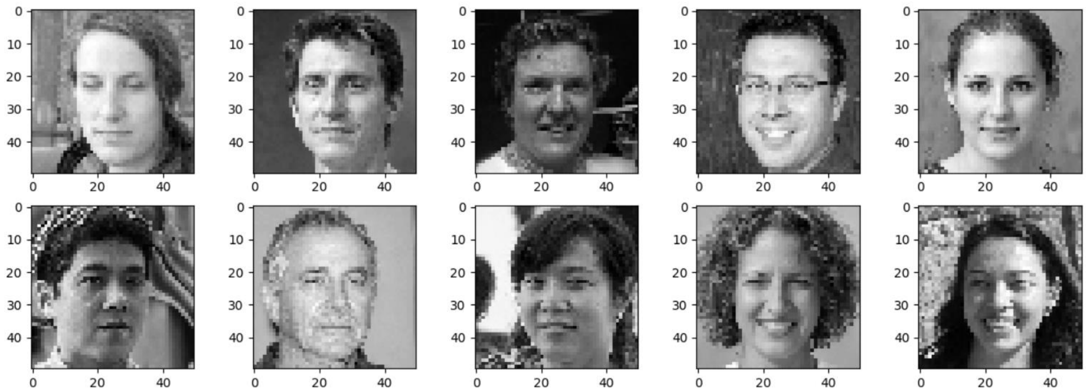


（2）降维至 30 维结果：



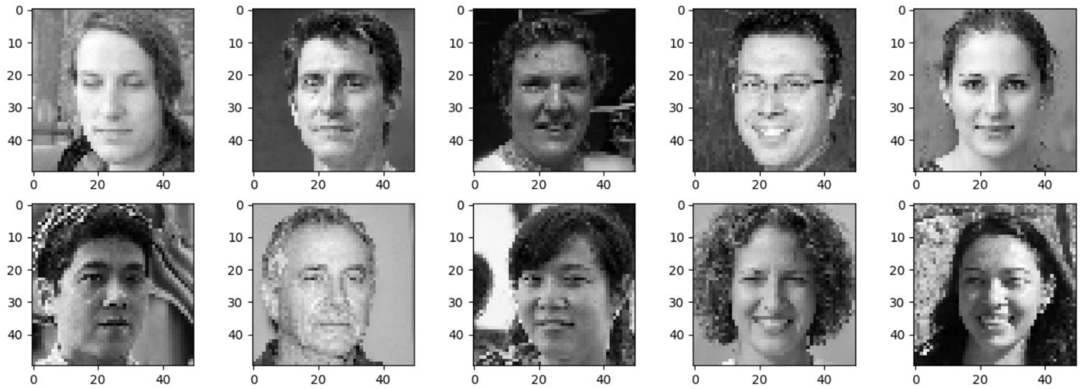
```
PCA后的信噪比如下所示：
图像 1 的信噪比:  100
图像 2 的信噪比:  100
图像 3 的信噪比:  100
图像 4 的信噪比:  100
图像 5 的信噪比:  100
图像 6 的信噪比:  100
图像 7 的信噪比:  100
图像 8 的信噪比:  100
图像 9 的信噪比:  100
图像 10 的信噪比:  100
```

（3）降维至 20 维结果：



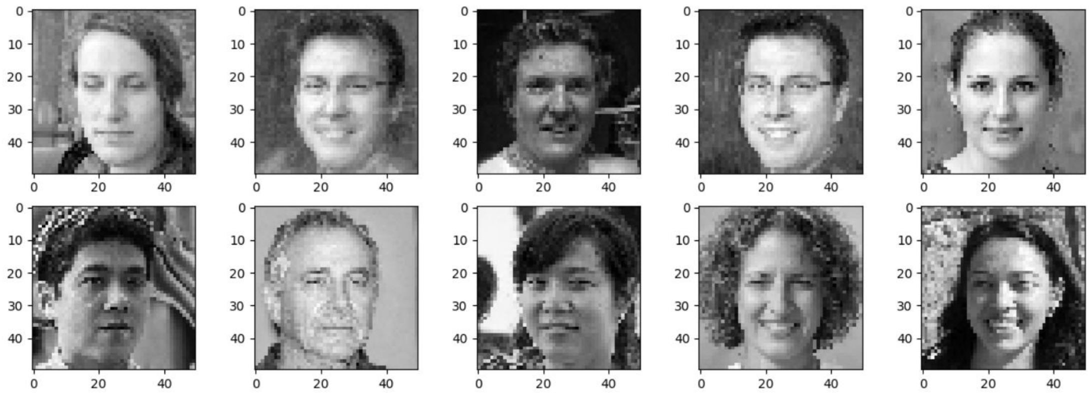
```
PCA后的信噪比如下所示：
图像 1 的信噪比:  100
图像 2 的信噪比:  100
图像 3 的信噪比:  100
图像 4 的信噪比:  100
图像 5 的信噪比:  100
图像 6 的信噪比:  100
图像 7 的信噪比:  100
图像 8 的信噪比:  100
图像 9 的信噪比:  100
图像 10 的信噪比:  100
```

（4）降维至 10 维结果：



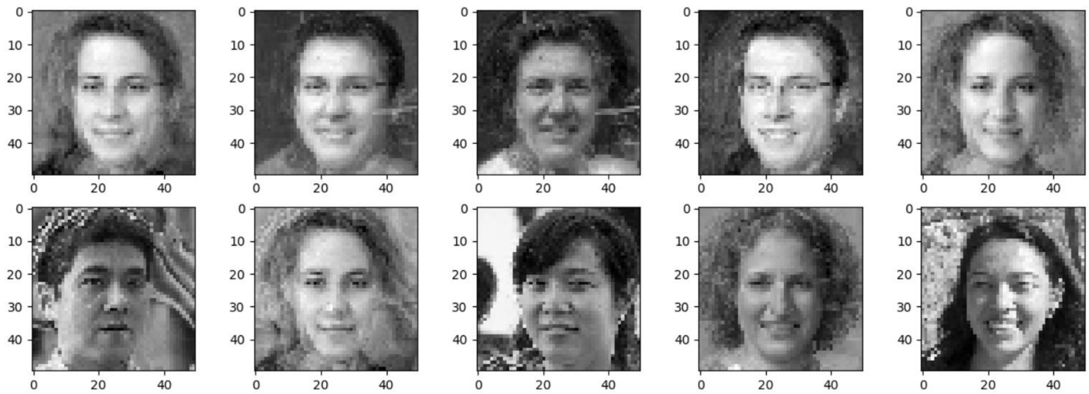
```
PCA后的信噪比如下所示：
图像 1 的信噪比： 100
图像 2 的信噪比： 100
图像 3 的信噪比： 100
图像 4 的信噪比： 100
图像 5 的信噪比： 100
图像 6 的信噪比： 100
图像 7 的信噪比： 100
图像 8 的信噪比： 100
图像 9 的信噪比： 100
图像 10 的信噪比： 100
```

（5）降维至 8 维结果：



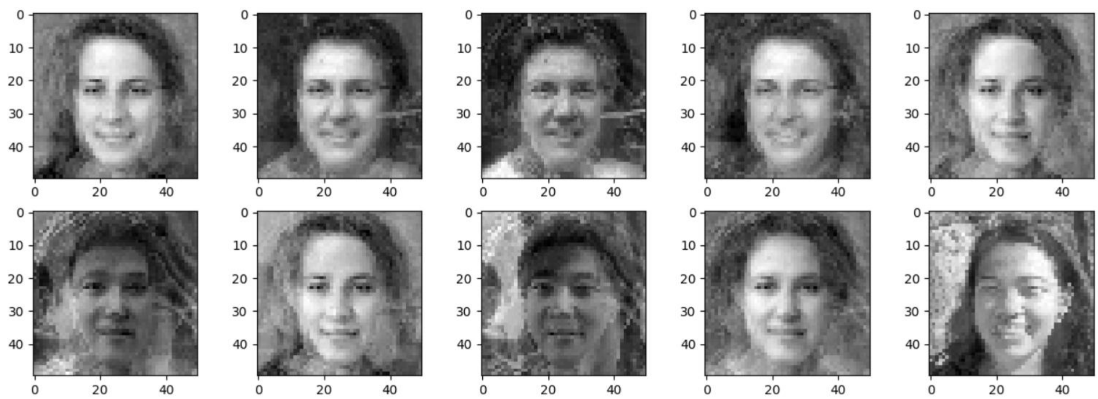
```
PCA后的信噪比如下所示：
图像 1 的信噪比： 39.52950661476858
图像 2 的信噪比： 22.124704562480005
图像 3 的信噪比： 34.482207771929055
图像 4 的信噪比： 27.384895021901208
图像 5 的信噪比： 38.18885501609668
图像 6 的信噪比： 45.084001131392
图像 7 的信噪比： 45.69767280622025
图像 8 的信噪比： 87.36701137918168
图像 9 的信噪比： 33.84235671401048
图像 10 的信噪比： 52.81968785187787
```

（6）降维至 5 维结果：



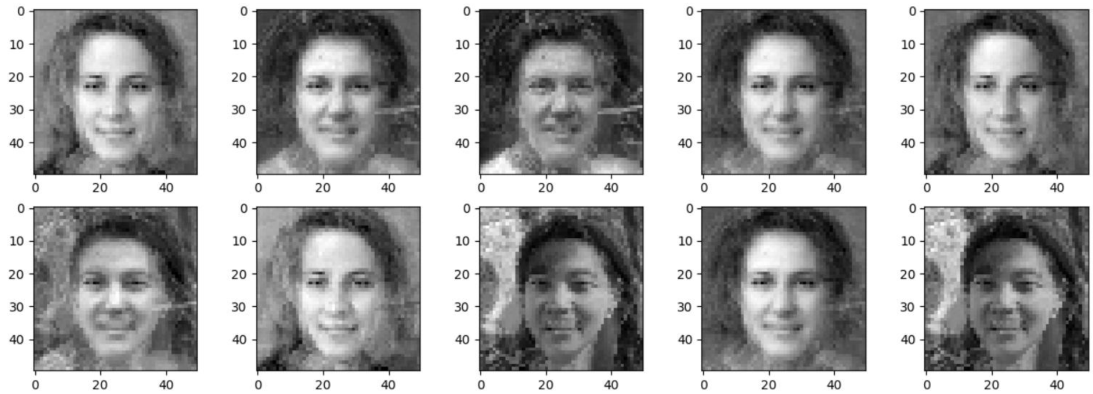
```
PCA后的信噪比如下所示:
图像 1 的信噪比:  19.232067725993268
图像 2 的信噪比:  21.467029628910876
图像 3 的信噪比:  23.205463036157298
图像 4 的信噪比:  23.69546670038911
图像 5 的信噪比:  20.832324552802476
图像 6 的信噪比:  29.071776530753773
图像 7 的信噪比:  19.38063749546661
图像 8 的信噪比:  60.967066494117226
图像 9 的信噪比:  22.750503490102506
图像 10 的信噪比:  35.67998551625918
```

（7）降维至 3 维结果：



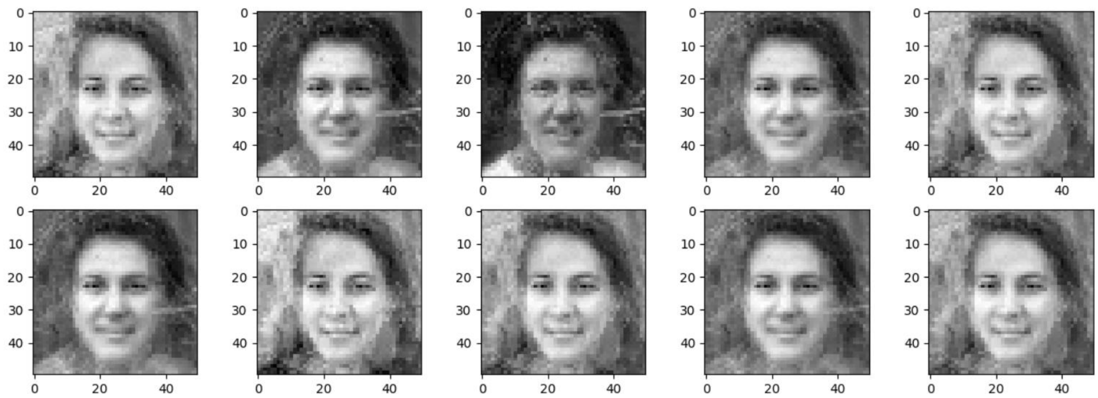
```
PCA后的信噪比如下所示:
图像 1 的信噪比:  19.08638359951794
图像 2 的信噪比:  20.704528157520908
图像 3 的信噪比:  22.289661987371403
图像 4 的信噪比:  17.922507101788653
图像 5 的信噪比:  20.42097320534034
图像 6 的信噪比:  17.270194354811554
图像 7 的信噪比:  18.936341771955863
图像 8 的信噪比:  18.231964567848877
图像 9 的信噪比:  16.662499616186288
图像 10 的信噪比:  21.428090409196404
```

（8）降维至 2 维结果：



```
PCA后的信噪比如下所示：
图像 1 的信噪比:  18.701023606686025
图像 2 的信噪比:  19.808556428572924
图像 3 的信噪比:  22.28701724819073
图像 4 的信噪比:  15.628246032483833
图像 5 的信噪比:  19.7498934182871
图像 6 的信噪比:  14.473644014182089
图像 7 的信噪比:  18.67770560517188
图像 8 的信噪比:  16.108750646597503
图像 9 的信噪比:  16.261333161055827
图像 10 的信噪比:  14.922813248221875
```

（9）降维至 1 维结果：



```
PCA后的信噪比如下所示：
图像 1 的信噪比:  17.488320935267232
图像 2 的信噪比:  19.303664554290098
图像 3 的信噪比:  21.90772738696125
图像 4 的信噪比:  14.942408465836118
图像 5 的信噪比:  16.52283043488063
图像 6 的信噪比:  14.104125834461335
图像 7 的信噪比:  17.624338351737507
图像 8 的信噪比:  12.999242677947136
图像 9 的信噪比:  15.634686117879959
图像 10 的信噪比:  12.375046308995945
```

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

```
from Lab4.all_operation import create_data_by_two_or_three_dimension, read_from_file, PCA
from Lab4.draw import draw_picture_by_image, draw_picture_by_create_PCA

# 用于生成数据的测试
dimension = 3
data_num = 50
x = create_data_by_two_or_three_dimension(dimension, data_num)
center_data, w, mu_x = PCA(x, dimension - 1)
x_after_PCA = (x - mu_x).dot(w).dot(w.T) + mu_x
print("中心化后的数据集为:\n", center_data)
print("特征向量矩阵为:\n", w)
print("降维前样本均值为:\n", mu_x)
draw_picture_by_create_PCA(dimension, x, x_after_PCA)

# 用人脸图像进行测试
x = read_from_file('face_collection')
x_num, x_dimension = x.shape  # 数据个数 x_num 和维度 x_dimension
center_data, w, mu_x = PCA(x, 1)  # PCA 降维
```

```
print("中心化后的数据集为为:\n", center_data)
print("特征向量矩阵为:\n", w)
print("降维前样本均值为:\n", mu_x)
draw_picture_by_image(x, w, center_data, mu_x, x_num)
```

draw.py

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


def draw_picture_by_create_PCA(dimension, data_before_PCA, x_after_PCA):
    """
    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:
    """
    if dimension == 2:  # 对二维数据画图
        plt.scatter(data_before_PCA[:, 0], data_before_PCA[:, 1], facecolor="none", edgecolor="b",
                    label="data_before_PCA")
        plt.scatter(x_after_PCA[:, 0], x_after_PCA[:, 1], facecolor='r', label='data_after_PCA')
        plt.xlabel('x')
        plt.ylabel('y')
    elif dimension == 3:  # 对三维数据画图
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.scatter(data_before_PCA[:, 0], data_before_PCA[:, 1], data_before_PCA[:, 2], edgecolor="b",
                   label='data_before_PCA')
        ax.scatter(x_after_PCA[:, 0], x_after_PCA[:, 1], x_after_PCA[:, 2], facecolor='r', label='data_after_PCA')
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')
    else:
        assert False
    plt.legend()
    plt.show()

```

```python

def PSNR(image1, image2):
    """
    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
    """
    mse = np.mean((image1 / 255. - image2 / 255.) ** 2)
    if mse < 1.0e-10:
        return 100
    max_pixel = 1  # 将最大像素设置为 1
    return 20 * math.log10(max_pixel / math.sqrt(mse))  # 计算信噪比即可


def draw_picture_by_image(data_set, w, center_data, mu_x, x_num):
    """
    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
    """
    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
    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
    x_after_PCA = np.dot(center_data, w)  # 计算降维后的数据
    refactoring_data = np.dot(x_after_PCA, w.T) + mu_x  # 重构降维后的数据
    plt.figure(figsize=size)
    for i in range(x_num):
        plt.subplot(3, 5, i + 1)
        plt.imshow(refactoring_data[i].reshape(size), cmap="gray")  # 预览该图像
    plt.show()
    print("PCA 后的信噪比如下所示：")
    for i in range(x_num):  # 打印信噪比
        psnr = PSNR(data_set[i], refactoring_data[i])
        print('图像', i + 1, '的信噪比：', psnr)
```

all_operation.py

```python
import numpy as np
```

```
import matplotlib.pyplot as plt
import os
import cv2


def create_data_by_two_or_three_dimension(data_dimension, num):
    """
    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
    """
    if data_dimension == 2:  # 对二维数据定义均值和方差
        mean = [-2, 2]
        cov = [[1, 0], [0, 0.01]]
    elif data_dimension == 3:  # 对三维数据定义均值和方差
        mean = [1, 2, 3]
        cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
    else:
        assert False
    data_set = []  # 定义数据集
    for index in range(num):  # 生成 num 个数据，加入数据集
        data_set.append(np.random.multivariate_normal(mean, cov).tolist())
    return np.array(data_set)


def PCA(data_set, k):
    """
    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)
    """
    rows, cols = data_set.shape  # 得到数据集的行和列
    data_mean = np.sum(data_set, 0) / rows  # 计算降维前样本均值
    center_data = data_set - data_mean  # 进行数据集的中心化操作
    covariance_matrix = np.dot(center_data.T, center_data)  # 计算协方差矩阵X.T · X
    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
    eigenvalue_sorted = np.argsort(eigenvalue)  # 将所有特征值排序
    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
```

```python
    return center_data, eigenvector_matrix, data_mean


def read_from_file(file_path):
    """
    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
    """
    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
    i = 1
    file_list = os.listdir(file_path)  # 读取该路径下所有图像的列表，放入 file_list
    data_set = []  # 定义数据集
    plt.figure(figsize=size)
    for file in file_list:  # 对于 file_list 中所有图像
        path = os.path.join(file_path, file)  # 连接文件路径，得到每个图像的路径
        plt.subplot(3, 5, i)
        with open(path) as f:
            image = cv2.imread(path)  # 读取这张图像
            image = cv2.resize(image, size)  # 将图像压缩至 size 大小
            image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # 将图像通过三通道转换为灰度图
            plt.imshow(image_gray, cmap="gray")  # 预览该图像
            h, w = image_gray.shape  # 得到该图像的维度
            image_col = image_gray.reshape(h * w)  # 对(h,w)的图像数据拉平
            data_set.append(image_col)  # 加入该图像数据给数据集中
        i += 1
    plt.show()
    return np.array(data_set)
```