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

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）。

三、 实验环境

- 硬件：Intel i5-8265U、512G SSD、8G RAM；
- 系统：Windows 11；
- IDE：Pycharm。

四、 设计思想

4.1 算法原理

有一个样本 $\boldsymbol{x_i}$，其维度为 m，其在基向量 $\boldsymbol{u_1}$ 上的投影为

$$z_{i1} = \boldsymbol{x_i'u_1},$$

并且有

$$\boldsymbol{u_1'u_1} = 1,$$

将之拓展到 m 个标准正交基向量 $\boldsymbol{u_1}$、$\boldsymbol{u_2}\cdots\boldsymbol{u_m}$，则有

$$z_{ij} = \boldsymbol{x_i'u_j}, j = 1, 2\cdots m$$

$$\boldsymbol{z_i'} = \begin{bmatrix} z_{i1} & z_{i2} & \cdots & z_{im} \end{bmatrix},$$

$\boldsymbol{z_i}$ 可以视为 $\boldsymbol{x_i}$ 在基向量 $\boldsymbol{u_1}$、$\boldsymbol{u_2}\cdots\boldsymbol{u_m}$ 构成的向量空间上的投影。

不妨设

$$\boldsymbol{y_i} = \sum_{j=i}^{m} z_{ij}\boldsymbol{u_j} = \sum_{j=i}^{m} (\boldsymbol{x_i'u_j})\boldsymbol{u_j},$$

我们可以将 $\boldsymbol{z_i}$ 视为 $\boldsymbol{x_i}$ 的压缩后的向量，$\boldsymbol{y_i}$ 视为解压后的向量，$\|\boldsymbol{x_i}-\boldsymbol{y_i}\|^2$ 视为样本压缩后的损失。

显然当基向量数目为 m 时，压缩损失为 0，但此时并未起到压缩效果。要起到压缩效果，需要让 $\boldsymbol{z_i}$ 维度降低，即减少基向量的数目。不妨设基向量的数目减少到 k，那么此时压缩后的样本 $\boldsymbol{z_i}$，还原后的样本 $\boldsymbol{y_i}$，压缩导致的误差 $\boldsymbol{\Delta_i}$ 为：

$$\boldsymbol{z_i'}=\begin{bmatrix}z_{i1} & z_{i2} & \cdots & z_{ik}\end{bmatrix}=\begin{bmatrix}\boldsymbol{x_i'u_1} & \boldsymbol{x_i'u_2} & \cdots & \boldsymbol{x_i'u_k}\end{bmatrix}, \tag{1}$$

$$\boldsymbol{y_i}=\sum_{j=i}^{k}z_{ij}\boldsymbol{u_j}=\sum_{j=i}^{k}(\boldsymbol{x_i'u_j})\boldsymbol{u_j}, \tag{2}$$

$$\boldsymbol{\Delta_i}=\sum_{j=k+1}^{m}z_{ij}\boldsymbol{u_j}=\sum_{j=k+1}^{m}(\boldsymbol{x_i'u_j})\boldsymbol{u_j}, \tag{3}$$

现在给定压缩的维度 k，要求的一组基向量，使压缩的总体误差尽可能小。不妨设数据集为 $\boldsymbol{X}$，样本数量为 n，每一个样本维度为 m，总误差为 S。

$$S=\sum_{i=1}^{n}\|\boldsymbol{\Delta_i}\|^2=\sum_{i=1}^{n}\sum_{j=k+1}^{m}(\boldsymbol{x_i'u_j})\boldsymbol{u_j},$$

为后续推导的方便，将基向量组织为

$$\boldsymbol{U_k}=\begin{bmatrix}\boldsymbol{u_1} & \boldsymbol{u_2} & \cdots & \boldsymbol{u_k}\end{bmatrix},$$

$$\boldsymbol{U_{m-k}}=\begin{bmatrix}\boldsymbol{u_{k+1}} & \boldsymbol{u_{k+2}} & \cdots & \boldsymbol{u_m}\end{bmatrix}.$$

将式（1）、式（2）、式（3）改写为向量内积、矩阵乘法的形式，

$$\boldsymbol{z_i}=\boldsymbol{U_k'x_i}, \tag{4}$$

$$\boldsymbol{y_i}=\sum_{j=i}^{k}z_{ij}\boldsymbol{u_j}=\boldsymbol{U_kz_i}=\boldsymbol{U_kU_k'x_i}, \tag{5}$$

$$\boldsymbol{\Delta_i}=\sum_{j=k+1}^{m}z_{ij}\boldsymbol{u_j}=\boldsymbol{U_{m-k}U_{m-k}'x_i}. \tag{6}$$

不妨设数据集 $\boldsymbol{X}$ 为

$$\boldsymbol{X}=\begin{bmatrix}\boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_n}\end{bmatrix}.$$

则式（4）、式（5）、式（6）可以进一步写作：

$$\boldsymbol{Z}=\boldsymbol{U_kX},$$

$$\boldsymbol{Y} = \boldsymbol{U_k U_k' X},$$

$$\boldsymbol{\Delta} = \boldsymbol{U_{m-k} U_{m-k}' X},$$

上式 $\boldsymbol{\Delta}$ 中每一列为一个样本的误差，所求 $\boldsymbol{U}$ 即为使总误差 S 最小的向量。

$$\begin{aligned} S &= tr(\boldsymbol{\Delta' \Delta}) \\ &= tr(\boldsymbol{X' U_{m-k} U_{m-k}' U_{m-k} U_{m-k}' X}) \\ &= tr(\boldsymbol{X' U_{m-k} U_{m-k}' X}) \\ &= tr(\boldsymbol{U_{m-k}' X X' U_{m-k}}) \end{aligned}$$

不妨设此时 X 是去中心化之后的数据集，则有

$$\boldsymbol{D} = \boldsymbol{X X'},$$

$$S = tr(\boldsymbol{U_{m-k}' D U_{m-k}}), \tag{7}$$

式中 $\boldsymbol{D}$ 为数据集 $\boldsymbol{X}$ 的协方差矩阵，是一个 $m \times m$ 的实对称阵，其有 m 个特征值与特征向量。不妨设其某一个特征向量为 $\boldsymbol{p_1}$，对应的特征值为 λ_1，则有

$$\boldsymbol{D p_1} = \lambda_1 \boldsymbol{p_1},$$

$$\begin{aligned} \boldsymbol{D}\begin{bmatrix} \boldsymbol{p_1} & \boldsymbol{p_2} & \cdots & \boldsymbol{p_{m-k}} \end{bmatrix} &= \begin{bmatrix} \lambda_1 \boldsymbol{p_1} & \lambda_2 \boldsymbol{p_2} & \cdots & \lambda_{m-k} \boldsymbol{p_{m-k}} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{p_1} & \boldsymbol{p_2} & \cdots & \boldsymbol{p_{m-k}} \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_{m-k} \end{bmatrix} \\ &= \boldsymbol{P_{m-k} \Lambda_{m-k}} \end{aligned}$$

由于实对称阵的特征向量是正交标准化的，故

$$\boldsymbol{P_{m-k}' P_{m-k}} = \boldsymbol{E_{(m-k)\times(m-k)}}$$

$$\boldsymbol{D P_{m-k}} = \boldsymbol{P_{m-k} \Lambda_{m-k}}$$

$$\boldsymbol{P_{m-k}' D P_{m-k}} = \boldsymbol{\Lambda_{m-k}} \tag{8}$$

综合来看式（7）、式（8）的形式，$\boldsymbol{U_{m-k}}$ 如果选取的是 $\boldsymbol{D}$ 的特征向量，那么为了使 $\boldsymbol{U_{m-k}' D U_{m-k}}$ 最小，应该选取 $m-k$ 个最小的特征值对应的特征向量。但如果 $\boldsymbol{U_{m-k}}$ 中某个基向量选取的不是 $\boldsymbol{D}$ 的特征向量，则该基向量可以视作由几个特征向量构成，最

后的迹仍可以看作为特征值的组合。$\boldsymbol{U_{m-k}}$ 选取了 $m-k$ 个最小的特征值对应的特征向量，则 $\boldsymbol{U_k}$ 选取的是其他 k 个特征向量。

最后推导得出基向量 U_k 为中心化后的 $\boldsymbol{X}$ 的协方差矩阵的前 k 个特征值对应的特征向量。压缩后的数据为 $\boldsymbol{U_k X}$。还原的数据为 $\boldsymbol{U_k U_k' X}$。

4.2 程序设计

4.2.1 数据生成

使用 numpy.random.multivariate_normal 函数生成一定数量的数据，为了压缩的效果，需要使某一个维度的方差相对较小，以使得压缩误差尽可能小。

4.2.2 图片读取

图片读取使用了 cv2.read 函数将图片转化为一个矩阵，使用了 cv2.cvtcolor 函数将矩阵变成灰度矩阵。

4.2.3 PCA 实现

首先对输入的矩阵进行去中心化的操作，然后求得其协方差矩阵，然后调用 numpy.linalg.eig 函数得到其特征值、特征向量，取前 k 大的特征值对应的特征向量作为压缩的基向量，然后依据上述公式对原数据进行压缩，压缩后基于之前的基向量还原，并且加上中心值，即完成了解压缩。

五、 实验结果与分析

5.1 生成数据

下图为二维、三维数据压缩一个维度的压缩效果。其中绿色点表示原始数据，红色点表示解压缩后的数据，图中的直线表示基向量。

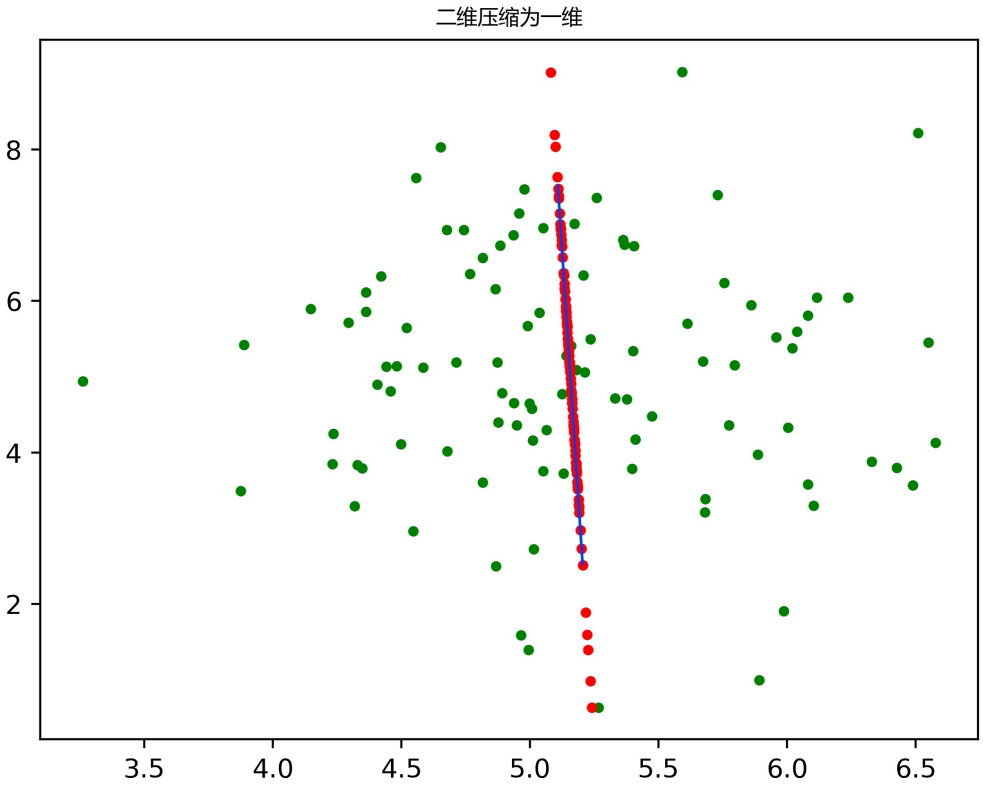

图 **1**　二维压缩效果

该图是从垂直于压缩平面的方向观察，可以看到绿色点和红色点几乎重合，压缩可以看作绿色点投影到平面上成为了红色点。

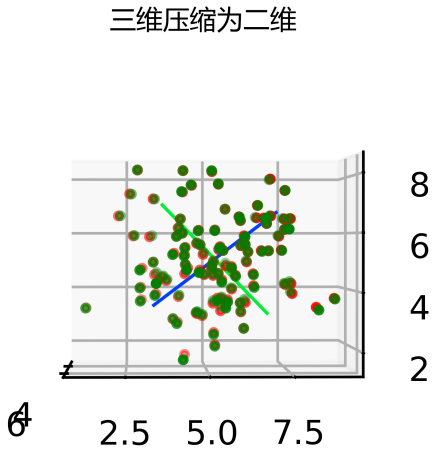

图 **2**　三维压缩效果一

该图为平行于压缩平面的方向观察，可以看到红色点在此视角下成为了一条直线。

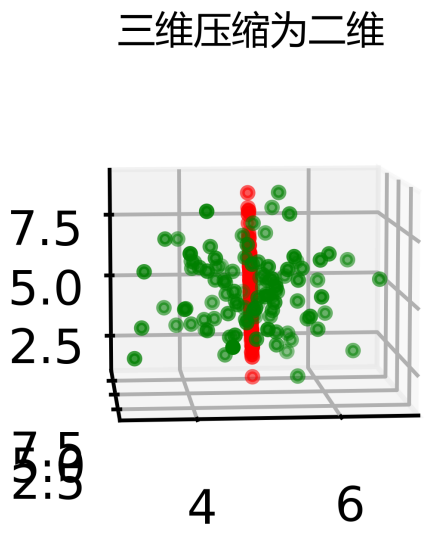

图 **3**　三维压缩效果二

5.2 图片数据

本次实验选取了两幅图片，先将其转化为灰度图后，然后进行压缩。解压缩图片以及 PSNR 如图所示。

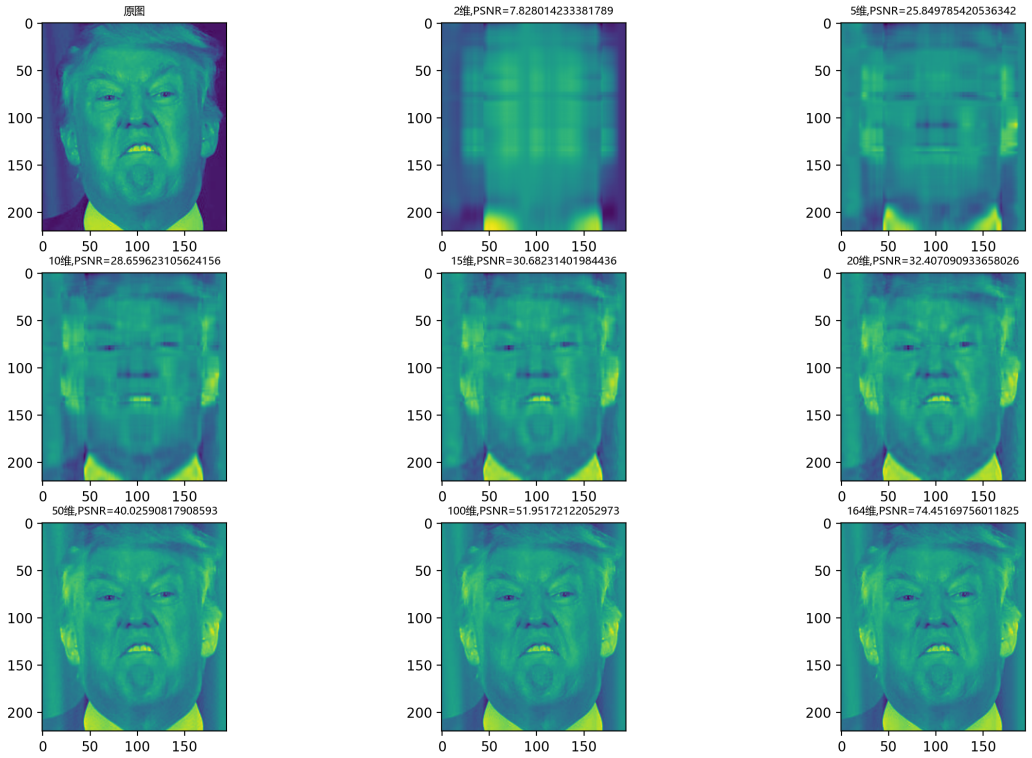

图 **4**　图片压缩例一

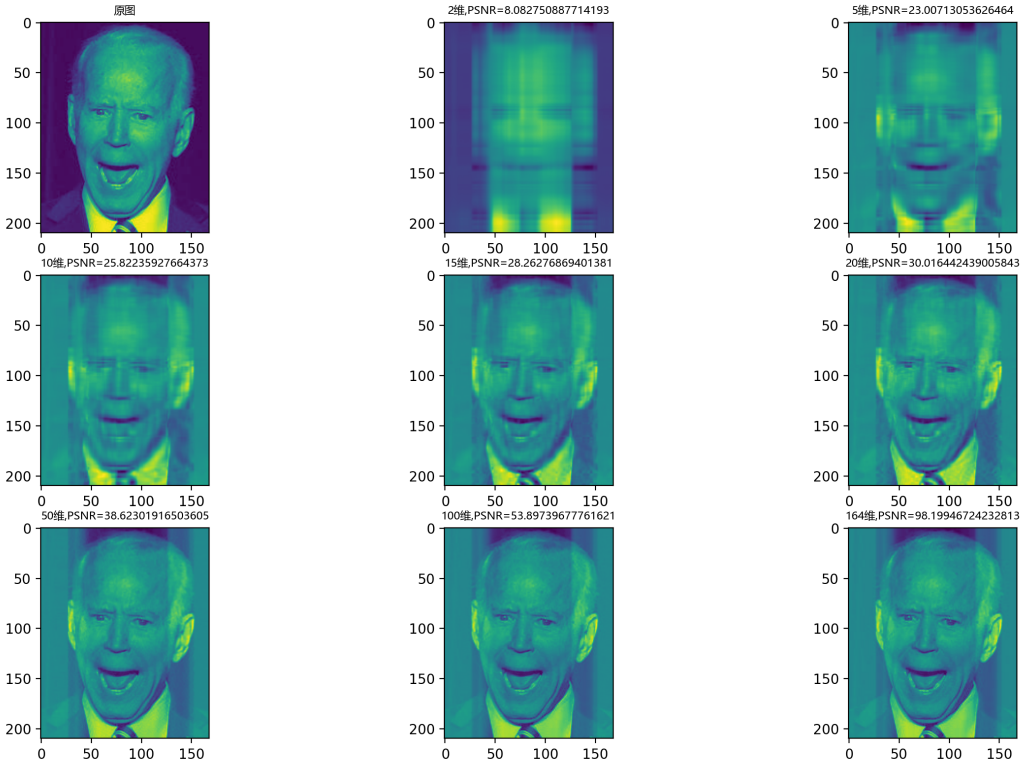

图 **5**　图片压缩例二

六、 结论

PCA 通过舍弃某些维度上变化较小的信息，降低了数据的维度，有效的实现了数据的压缩存储。但 PCA 也有一定的缺陷，不同维度的数据其重要程度是不同的，不能简单的依据方差或是特征值来衡量，数据集不同维度的重要性并不没有保存在数据集中。

A 主程序

```
import copy

import numpy as np
import cv2

from graph import init_graph, font_title, draw_graph
import matplotlib.pyplot as plt


def generate_data(size, raw_dimension_num):
    mean_list = [5.0] * raw_dimension_num
    cov_list = [[0.0] * i + [2.0] + [0.0] * (raw_dimension_num - i - 1) for i in
        range(raw_dimension_num)]
    cov_list[0][0] = 0.5

    data = np.random.multivariate_normal(mean_list, cov_list, size)
    return data


def PCA(data, compress_dimension_num):
    raw_dimension_num = len(data[0])
    mean_array = np.mean(data, axis=0)
    for i in range(len(data)):
        data[i] -= mean_array

    cov_matrix = (data.T @ data) / len(data)
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
    sort_index = np.argsort(eigenvalues)
    # 变换矩阵 or 未移位的基向量
    compress_matrix = np.array([[0.0] * compress_dimension_num] * raw_dimension_num)
    for i in range(compress_dimension_num):
        index = sort_index[-1 - i]
        compress_matrix[:, i] = eigenvectors[:, index]

    # 压缩后的矩阵
    compressed_data = data @ compress_matrix

    # 解压后的矩阵
    decompressed_data = compressed_data @ compress_matrix.T
    for i in range(len(data)):
        # 加上中心值
        decompressed_data[i] += mean_array

    return mean_array, compress_matrix, compressed_data, decompressed_data
```

```
def draw(data, decompressed_data, mean_array, compress_matrix):
    raw_dimension_num = len(data[0])

    if raw_dimension_num == 2:
        init_graph(plt, dpi=400)
        plt.title("二维压缩为一维", font=font_title)
        plt.scatter(data[:, 0], data[:, 1], c='g', marker='.')
        plt.scatter(decompressed_data[:, 0], decompressed_data[:, 1], c='r', marker='.')
        plt.plot([mean_array[0] - 2.5 * compress_matrix[0][0], mean_array[0] + 2.5 *
            compress_matrix[0][0]],
                [mean_array[1] - 2.5 * compress_matrix[1][0], mean_array[1] + 2.5 *
                    compress_matrix[1][0]], linewidth=1)
        draw_graph(plt)
    elif raw_dimension_num == 3:
        init_graph(plt, dpi=400)
        ax = plt.axes(projection='3d')
        plt.title("三维压缩为二维", font=font_title)
        ax.scatter3D(data[:, 0], data[:, 1], data[:, 2], c='g', marker='.')
        ax.scatter3D(decompressed_data[:, 0], decompressed_data[:, 1], decompressed_data[:, 2],
            c='r', marker='.')
        ax.plot3D([mean_array[0] - 2.5 * compress_matrix[0][0], mean_array[0] + 2.5 *
            compress_matrix[0][0]],
                [mean_array[1] - 2.5 * compress_matrix[1][0], mean_array[1] + 2.5 *
                    compress_matrix[1][0]],
                [mean_array[2] - 2.5 * compress_matrix[2][0], mean_array[2] + 2.5 *
                    compress_matrix[2][0]], linewidth=1)
        ax.plot3D([mean_array[0] - 2.5 * compress_matrix[0][1], mean_array[0] + 2.5 *
            compress_matrix[0][1]],
                [mean_array[1] - 2.5 * compress_matrix[1][1], mean_array[1] + 2.5 *
                    compress_matrix[1][1]],
                [mean_array[2] - 2.5 * compress_matrix[2][1], mean_array[2] + 2.5 *
                    compress_matrix[2][1]], linewidth=1)
        draw_graph(plt)
    else:
        print("Matplotlib can't draw if dimension num>3")


def PSNR(img1, img2):
    img0 = img1 - img2
    return 10 * np.log10(65025 * len(img0) * len(img0[0]) / np.sum(img0 ** 2))


if __name__ == '__main__':
    raw_dimension_num = 3
    data = generate_data(size=100, raw_dimension_num=raw_dimension_num)
    mean_array, compress_matrix, compressed_data, decompressed_data = PCA(copy.deepcopy(data),
```

```
        raw_dimension_num - 1)
    draw(data, decompressed_data, mean_array, compress_matrix)

    # img = cv2.imread("./figures/biden.jpg")
    # img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # plt.subplot(3, 3, 1)
    # plt.title("原图", font=font_title)
    # plt.imshow(img_gray)
    # img_gray_float = np.array(img_gray, dtype=np.float32, copy=True)
    # dimension_list = [2, 5, 10, 15, 20, 50, 100, 164] # biden
    # # dimension_list = [2, 5, 10, 15, 20, 50, 100, 193] # trump
    # for i in range(8):
    #     plt.subplot(3, 3, i + 2)
    #     mean_array, compress_matrix, compressed_data, decompressed_data = PCA(img_gray_float,
        dimension_list[i])
    #     decompressed_data_int = np.array(decompressed_data, dtype=np.int32, copy=True)
    #     plt.title(str(dimension_list[i]) + "维,PSNR=" + str(PSNR(img_gray_float,
        np.array(decompressed_data, dtype=np.float32, copy=True))),
    #               font=font_title)
    #     plt.imshow(decompressed_data_int)
    # plt.show()
```

B 可视化

```
# 设置字体，解决中文无法识别的问题
# 图表标题
font_title = {
    'family': 'Microsoft Yahei',
    'weight': 'regular',
    'size': 8
}

# 坐标轴标题
font_label = {
    'family': 'Microsoft Yahei',
    'weight': 'regular',
    'size': 10
}

# 图例
font_legend = {
    'family': 'Microsoft Yahei',
    'weight': 'regular',
    'size': 6
}
```

```python
def init_graph(plt, dpi=150, style="seaborn-bright"):
    # 设置清晰度
    plt.figure(dpi=dpi)

    # 设置样式
    plt.style.use(style)


def draw_graph(plt, save=True, filename="picture.jpg", show=True):
    # 是否保存
    if save:
        plt.savefig("./figures/" + filename)

    if show:
        plt.show()
```