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	包宇昊
学号	1180300605
班号	1803105
电子邮件	2568235796@qq.com
手机号码	18267116288

目录

1 .实验目标

- 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）

2 .实验要求及环境

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）。

实验环境：

pycharm 2020.2.2

python 3.8

win10

X86-64

3 .数学思想

PCA：主成分分析，n 维特征映射到 k 维上（k<n），这 k 维是全新的正交特征，这 k 维特征称为主成分。它是一种矩阵的压缩算法，在减少矩阵维数的同时尽可能的保留原矩阵的信息，消除不重要的信息和噪声，从而节省空间、压缩数据量。

本质是对角化协方差矩阵，目的是让维度之间的相关性最小（降噪），保留下来的维度的能量最大（去冗余）。

推导有两种方式：最大投影方差和最小投影距离。

- 最大投影方差：样本点在这个超平面上的投影尽可能分开
- 最小投影距离：样本点到这个超平面的距离都足够近

主要介绍最小投影距离这种推导.

3.1 中心化

在开始 PCA 之前需要对数据进行预处理，即对数据中心化。

设数据集 $X=\{x_1,x_2,...,x_n\}$，其中 $x_i=\{x_{i1},x_{i2},...,x_{id}\}$，

则此数据集的中心向量（均值向量）为：

$$\mu=\frac{1}{n}\sum_{i=1}^{n}x_i$$

对数据集每个样本均进行操作$xi=xi-\mu$，得到中心化后的数据.中心化后的线性变化就变成了旋转.

协方差为

$$S=\frac{1}{n}\sum_{i=1}^{n}x_ix_i^T=\frac{1}{n}X^TX$$

设定一组标准正交基:

$U_{k\times d}=\{u_1,u_2,...,u_k\}$，$k<d,u_i=\{u_{i1},u_{i2},...,u_{id}\}$.

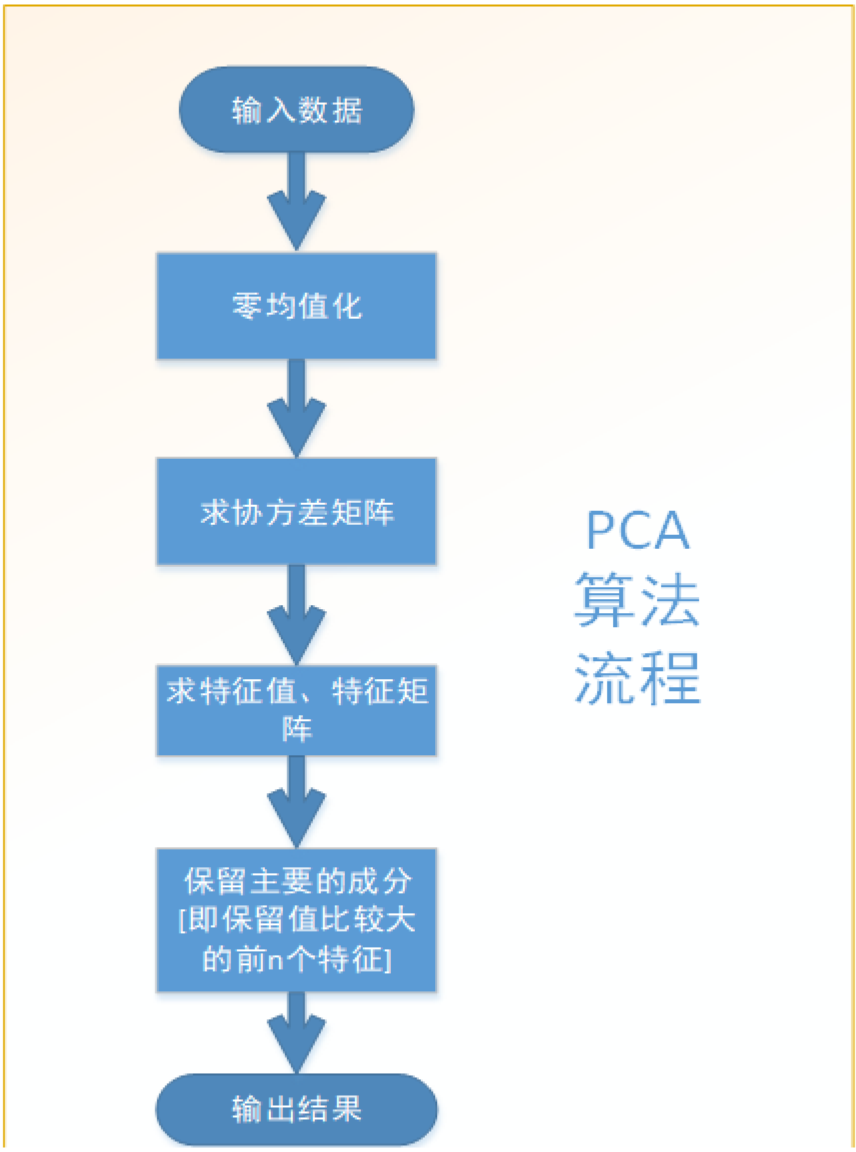
降维压缩后的矩阵: $Y_{n\times k}=XU^T$ ，降维压缩后的矩阵为:$Y=XU^T$

3.2 最小投影距离

$$\begin{aligned}
\arg\min_U \sum_{i=1}^n ||\hat{x}_i - x_i||_2^2 &= \arg\min_U \sum_{i=1}^n ||x_i U^T U - x_i||_2^2 \\
&= \arg\min_U \sum_{i=1}^n ((x_i U^T U)(x_i U^T U)^T - 2(x_i U^T U)x_i^T + x_i x_i^T) \\
&= \arg\min_U \sum_{i=1}^n (x_i U^T U U^T U x_i^T - 2x_i U^T U x_i^T + x_i x_i^T) \\
&= \arg\min_U \sum_{i=1}^n (-x_i U^T U x_i^T + x_i x_i^T) \\
&= \arg\min_U -\sum_{i=1}^n x_i U^T U x_i^T + \sum_{i=1}^n x_i x_i^T \\
&\Leftrightarrow \arg\min_U -\sum_{i=1}^n x_i U^T U x_i^T \\
&\Leftrightarrow \arg\max_U \sum_{i=1}^n x_i U^T U x_i^T \\
&= \arg\max_U \ tr(U(\sum_{i=1}^n x_i^T x_i)U^T) \\
&= \arg\max_U \ tr(U X^T X U^T) \qquad s.t.\ UU^T = 1
\end{aligned}$$

通过公式推导推出

4 .实验步骤



4.1 生成自己的数据

生成三维高斯随机分布数据，完成转置，使矩阵一行为一个特征，一列为一个数据样本。

```
mean = (5, 1, 10)
cov = [[5, 0, 0], [0, 0.01, 0], [0, 0, 6]]
size = 100
np.random.seed(0)
data = np.random.multivariate_normal(mean, cov, size)
data = data.T
```

4.2 PCA 具体实现

4.2.1 零均值化

先通过 np.mean 将数据按行取特征平均值，保存到一行，即 mean_data。

再取均值，即原 data 矩阵每行（每个特征）都减去特征平均值，得到

mean_removed。

```
mean_data = np.mean(data,axis=0)      #被压缩到1行
mean_removed = data - mean_data      #去均值
```

4.2.2 求协方差矩阵

通过 np.cov 直接对完成零均值化的矩阵求取协方差矩阵。

得到 cov_mean_removed。

```
cov_mean_removed = np.cov(mean_removed)
```

4.2.3 求特征值、特征向量

np.linalg.eig 直接求取特征值 eigvals 和特征向量 eigvects

```
eigvals,eigvects = np.linalg.eig(cov_mean_removed)    #特征值和特征向量
```

4.2.4 保留主要成分（选取最大的 k 个特征值）

调用 np.argsort 对特征值进行排序，返回数组，元素是从小到大排好序后特征值在原特征值数组中的下标

```
eigvals_Loc = np.argsort(eigvals)
```

从 eigvals_Loc 数组中从后往前选取 k 个数，即从数组返回 k 个最大特征值下标

```
eigvals_Loc_max_k = eigvals_Loc[:-(k+1):-1]    #返回k个最大特征值下标
```

根据下标在特征向量数组中选取 k 个最大特征值对应的特征向量

```
eigvects_max_k = eigvects[:,eigvals_Loc_max_k]    #返回k个最大特征值对应的特征向量
```

如此，我就完成了最大 k 个特征值的选取

4.2.5 映射到新的投影空间（实现降维）

将 k 个最大特征值的特征向量集的转置左乘零均值化矩阵，将原数据映射到降维后的投影空间

```
lower_data = np.dot(eigvects_max_k.T,mean_removed)    #降维后的数据集
```

4.2.6 在原空间重构降维数据集

将 k 个最大特征值的特征向量集左乘零均值化矩阵，相当于把被降维的数据集重新映射到原空间。

```
re_data = np.dot(eigvects_max_k,lower_data)+mean_data
```

4.3 绘图

4.3.1 二维绘图

```python
#二维绘图
def draw0(data):
    rows, cols = data.shape
    fig = plt.figure()
    ax = fig.add_subplot(111)
    for i in range(cols):
        ax.scatter(data[0,i],data[1,i],color='red')
    ax.set_title('2D')
    plt.show()
    return
```

4.3.2 三维绘图

```python
#三维绘图
def draw1(data):
    rows, cols = data.shape
    fig = plt.figure()
    ax = Axes3D(fig)
    for i in range(cols):
        ax.scatter(data[0,i],data[1,i],data[2,i],color='red')
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    ax.set_title('3D')
    plt.show()
    return
```

4.4 人脸数据处理

4.4.1 读入照片

Cv2.imread 读入文件，转换为 BGR 格式数据.我有三个测试数据，可以一一测试。

```
img = cv2.imread('1.JPG')
# img = cv2.imread('2.JPG')
# img = cv2.imread('3.JPG')
```

4.4.2 放缩图片

考虑到图片像素可能过大，可以先做预处理，减小数据量，将缩小后的图片作为原图，避免出现图片过大导致运行时间过长或溢出，此处由于我的图片较小，因此倍率 pra 置为 1，即不放缩。

```
img_resize = cv2.resize(img,(int(pra*cols),int(pra*rows)))
```

4.4.3 转换为单通道灰度图

由于三通道处理较为麻烦，我直接把转化为单通道灰度图后的图片作为原图。

这样灰度图矩阵就变成了二维矩阵，直接适用于之前写的 PCA 函数。

```
img_gray = cv2.cvtColor(img_resize,cv2.COLOR_BGR2GRAY)     #转换为单通道灰度图
```

```
plt.imshow(img_gray, cmap='gray')
plt.title('Original')
plt.show()
```

4.4.4 处理灰度矩阵降维后的数据

首先 PCA 处理，返回后的数据元素为复数，也有小数部分，需要进行处理。

1） 通过 np.real 取实数部分

2） 通过循环和 int（）强转，未完成预期目标，只是消除掉了小数部分，但数据类型仍然是 float64，这是 np 库某些函数的隐藏特性（把我恶心到了，debug 了一个下午）。

3） 通过 astype(int)才把灰度矩阵处理为正常的整型整数元素形式。

注意：直接改 dtype 的话会倍长数据

4） 随后显示重构后的灰度图

```
lower_data,re_data = PCA(data,k)
re_data = np.real(re_data)

#只能取整数，但仍为float64型
for i in range(Rows):
    for j in range(Cols):
        re_data[i][j] = int(re_data[i][j])
#特殊方法转换
re_data = re_data.astype(int)
print(re_data)

plt.imshow(re_data,cmap='gray')
plt.title('Restored')
plt.show()
```

4.4.5 计算最大峰值信噪比

PSNR 代表了两张图的差异，可以展示主要成分在图片中的作用占比

```
#计算峰值信噪比PSNR
def cal_PSNR(data1,data2):
    raws = data1.shape[0]
    cols = data1.shape[1]
    noise = data2 - data1
    sum = 0
    for i in range(raws):
        for j in range(cols):
            sum += np.abs(noise[i][j])
    MSE = sum/(raws*cols)
    PSNR = 20 * np.log10(255/np.sqrt(MSE))
    return PSNR
```

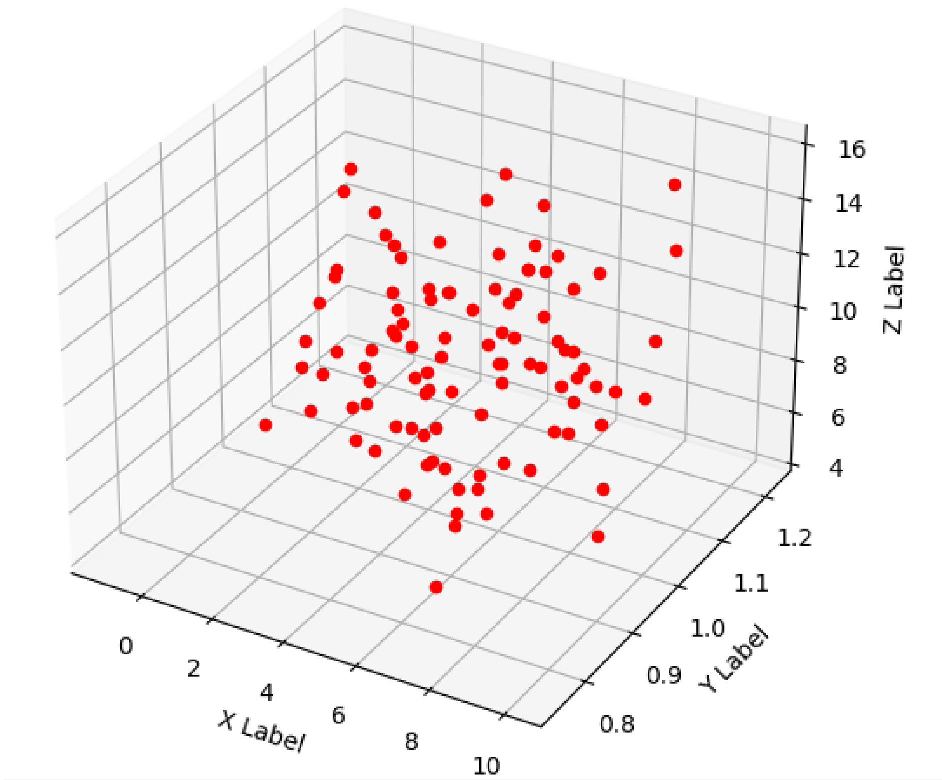
5 .测试效果

5.1 人工数据测试

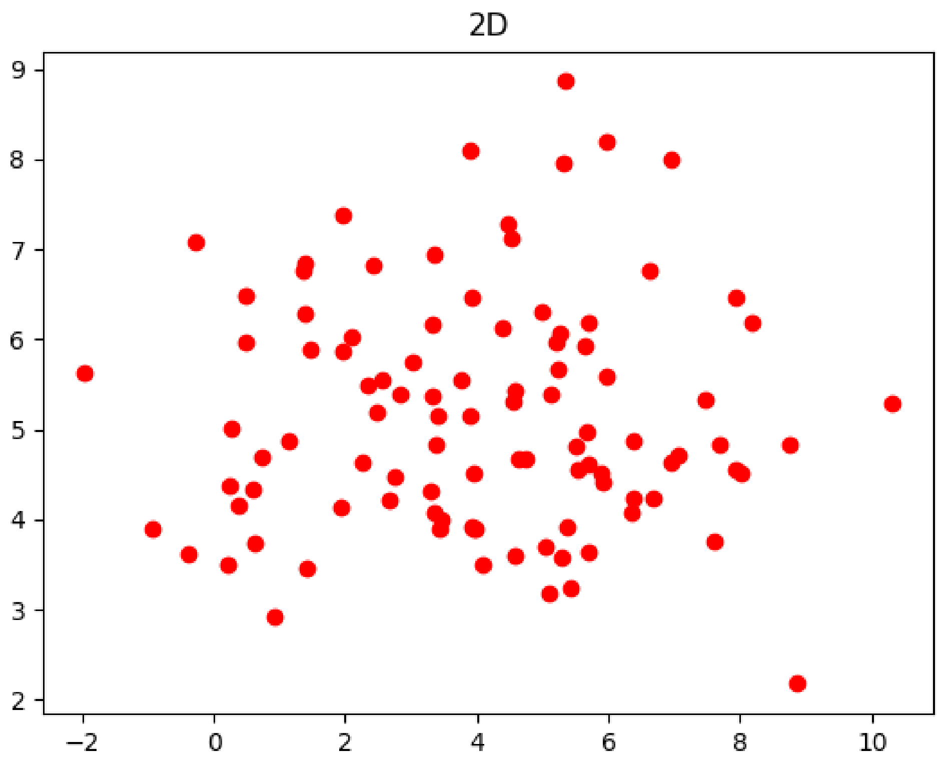
我做的是三维 → 二维

```
mean = (5, 1, 10)
cov = [[5, 0, 0], [0, 0.01, 0], [0, 0, 6]]
size = 100
np.random.seed(0)
data = np.random.multivariate_normal(mean, cov, size)
data = data.T
```

三维：

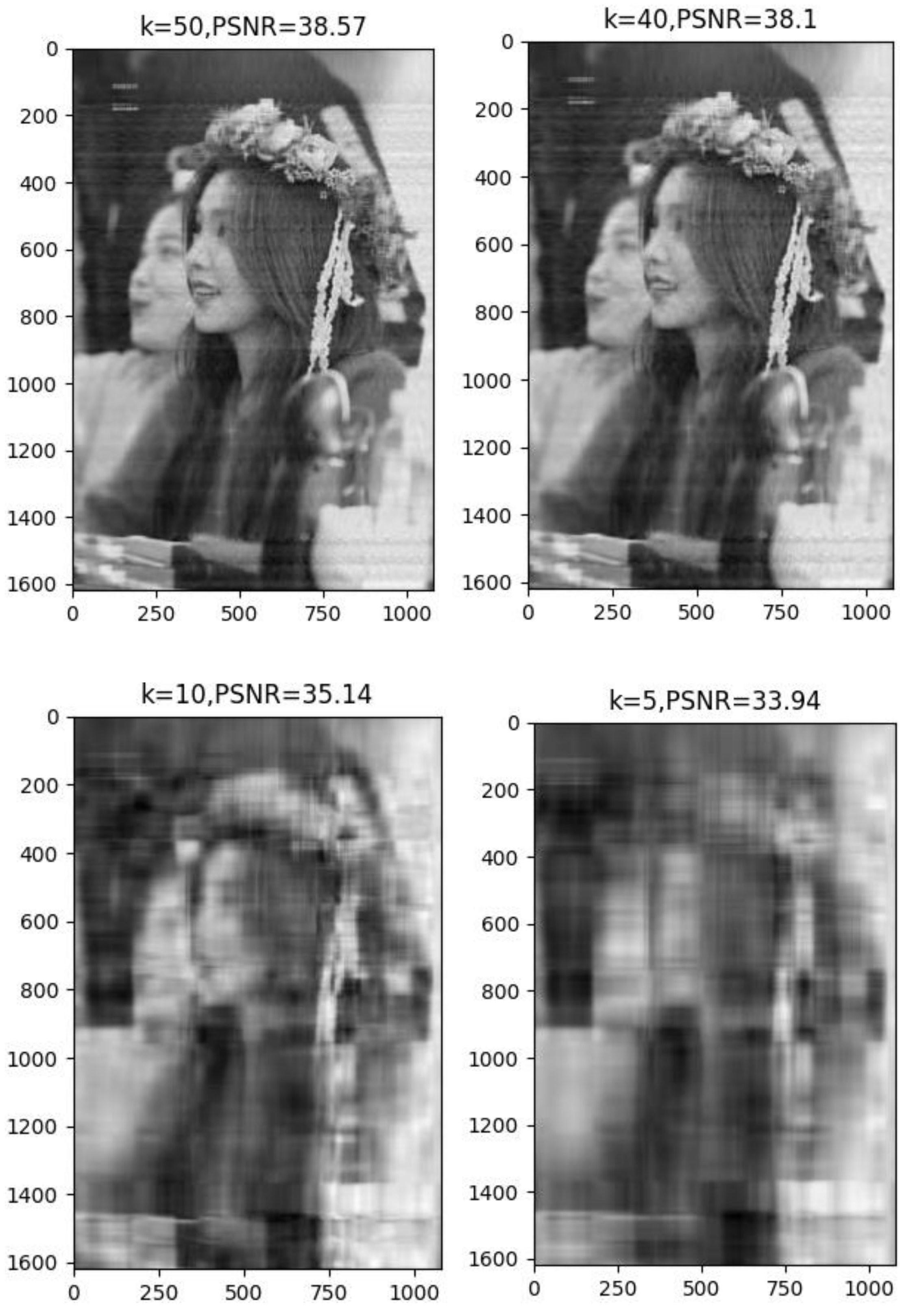

X Label
Y Label
Z Label
0
2
4
6
8
10
0.8
0.9
1.0
1.1
1.2
4
6
8
10
12
14
16

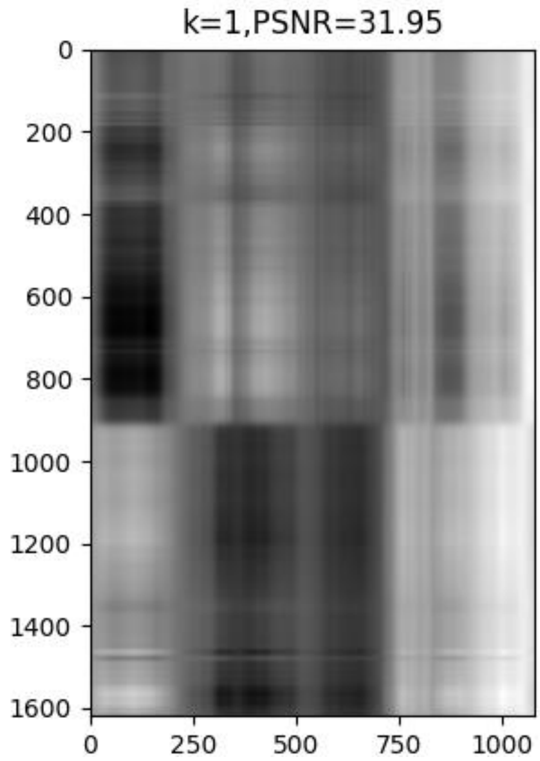
二维：


2D
−2
0
2
4
6
8
10
2
3
4
5
6
7
8
9

5.2 人脸数据测试

原图： 1680＊1080


k=50,PSNR=38.57
k=40,PSNR=38.1
k=10,PSNR=35.14
k=5,PSNR=33.94



易知: k 越大,图片和原图越接近,信噪比越大

6 .实验结论

1. k 越大,图片和原图越接近,信噪比越大
2. PCA 本质是保留主要信息,使保留下来的维度的能量最大（去冗余）,但是被舍去的信息未必不重要,是带有随机性的.
3. PCA 一定程度上起到了降噪的作用
4. PCA 对节省空间有很大作用,保持图片基本和原图视觉误差的条件下,数据维度减少了很多.