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： 必修

实验题目： PCA 模型实验

学号：

姓名：

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

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 2020.3.2 x64
Python 3.8　numpy1.20.3　matplotlib 3.4.2

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 基变换的矩阵表示

假设p_i为行向量，表示第i个基；a_j是一个列向量，表示第j个原始数据记录。那么可以得到通用的表示形式：

$$\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{pmatrix} (a_1 \quad a_2 \quad \cdots \quad a_M) = \begin{pmatrix} p_1 a_1 & \cdots & p_1 a_M \\ \vdots & \ddots & \vdots \\ p_R a_1 & \cdots & p_R a_M \end{pmatrix} \tag{1}$$

此时即可为矩阵中每一列向量a_i变换到左边矩阵中以每一行行向量为基所表示的空间中去。

2. 最大方差形式

在1中讨论了不同的基可以给出对同一组数据的不同表示，如果基的数量少于向量本身的维度，则可以达到降维的效果。

考虑一组观测数据集$\{x_n\}$，其中$\mathrm{n} = 1, \ldots, \mathrm{n}$，因此$\boldsymbol{x}_n$是一个D维欧几里得空间中的变量。我们的目标是将数据投影到维度$\mathrm{M} < \mathrm{D}$的空间中，同时最大化投影数据的方差，假定M的值是给定的。

当$\mathrm{M} = 1$时，我们使用D维向量$\boldsymbol{u}_1$定义这个空间的方向，并使$\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$，这样每个数据点$\boldsymbol{x}_n$被投影到一个标量值$\boldsymbol{u}_1^T x_n$上，投影数据的均值是$\boldsymbol{u}_1^T \bar{\mathrm{x}}$，其中$\bar{\mathrm{x}}$是样本集合的均值。

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \tag{2}$$

因此有投影数据的方差

$$\frac{1}{N}\sum_{n=1}^{N} \{\boldsymbol{u}_1^T \boldsymbol{x}_n - \boldsymbol{u}_1^T \bar{\boldsymbol{x}}\}^2 = \boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 \tag{3}$$

其中S是数据的协方差矩阵矩阵

$$\mathrm{S} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T \tag{4}$$

协方差矩阵可以表示两个变量的相关性，为了让每个变量尽可能表示更多的原始信息，我们希望他们之间尽可能不存在线性相关性。现在我们关于u_1最大化投影方差$\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$。由于最大化的过程需要满足$\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$归一化条件，故实际为有约束优化，因此采用拉格朗日乘数法，并将乘数记作λ_1，因此我们要对下式进行最大化：

$$\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1) \tag{5}$$

对其求关于$\boldsymbol{u}_1$的导数，可得

$$\frac{\partial}{\partial \boldsymbol{u}_1}\left(\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1)\right) = 2\boldsymbol{S}\boldsymbol{u}_1 - 2\lambda_1 \boldsymbol{u}_1 \tag{6}$$

令公式(6)为0，可得

$$\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1 \tag{7}$$

故$\boldsymbol{u}_1$一定是$\boldsymbol{S}$的一个特征向量，对等式左乘$\boldsymbol{u}_1^T$，可得

$$\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \tag{8}$$

因此，我们将$\boldsymbol{u}_1$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。

因此对于更一般的形式，我们有

$$\begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_M \end{pmatrix} (\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \dots \quad \boldsymbol{x}_N) = (\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad \dots \quad \boldsymbol{y}_M) \tag{9}$$

$$= \begin{pmatrix} \boldsymbol{u}_1 \boldsymbol{x}_1 & \dots & \boldsymbol{u}_1 \boldsymbol{x}_N \\ \vdots & \ddots & \vdots \\ \boldsymbol{u}_M \boldsymbol{x}_1 & \dots & \boldsymbol{u}_M \boldsymbol{x}_N \end{pmatrix} \tag{10}$$

设$\boldsymbol{x}$的均值为$\boldsymbol{\mu}$

$$\boldsymbol{\mu} = (\boldsymbol{\mu}_1 \quad \dots \quad \boldsymbol{\mu}_N) \tag{11}$$

由随机变量性质可得，

$$\mathrm{E}(\boldsymbol{y}_i) = \boldsymbol{u}_i^T \boldsymbol{\mu} \tag{12}$$

$$\mathrm{var}(\boldsymbol{y}_i) = \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_i \tag{13}$$

$$\mathrm{cov}(\boldsymbol{y}_i, \boldsymbol{y}_j) = \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_j \tag{14}$$

由公式(8)的推广，我们有$\boldsymbol{x}$的第k主成分是$\boldsymbol{u}_k^T \boldsymbol{x}$，其方差为协方差矩阵的第k大特征值$\lambda_k$。

$$\mathrm{var}(\boldsymbol{u}_k^T \boldsymbol{x}) = \boldsymbol{u}_k^T \boldsymbol{S} \boldsymbol{u}_k = \lambda_k \tag{15}$$

3. 求解步骤

降维：

假定数据集$\{\boldsymbol{x}_n\}$，其中$\mathrm{n}=1,\ldots,\mathrm{N}$，因此$\boldsymbol{x}_n$是一个D维欧几里得空间中的变量，并将其组织为一个矩阵

$$\mathbf{X}=\{\boldsymbol{x}_1 \quad \ldots \quad \boldsymbol{x}_N\} \tag{16}$$

我们得到的$\mathbf{X}$的矩阵为D行N列数据。将$\mathbf{X}$零值化，即减去每一行的均值。

设

$$\boldsymbol{\mu}_i=\left(\sum_{j=1}^{N}\boldsymbol{x}_{ij} \quad \ldots \quad \sum_{j=1}^{N}\boldsymbol{x}_{ij}\right) \tag{17}$$

$$\boldsymbol{\mu}=\begin{pmatrix}\sum_{j=1}^{N}\boldsymbol{x}_{1j} & \cdots & \sum_{j=1}^{N}\boldsymbol{x}_{1j}\\ \vdots & \ddots & \vdots\\ \sum_{j=1}^{N}\boldsymbol{x}_{Dj} & \cdots & \sum_{j=1}^{N}\boldsymbol{x}_{Dj}\end{pmatrix}$$

因此我们得到零均值化后的矩阵

$$\mathbf{X}=\boldsymbol{X}-\boldsymbol{\mu} \tag{18}$$

求出协方差矩阵

$$\mathrm{S}=\frac{1}{\mathrm{N}-1}\sum_{i=1}^{N}(x_i)(x_i)^T \tag{19}$$

$(x_i)(x_i)^T$为一个 D 维方阵。

计算协方差矩阵$\mathbf{S}$的特征值和特征向量，并根据特征值对特征向量排序，取前K行组成的矩阵P。

假设

$$\mathrm{P}=\{\boldsymbol{u}_1 \quad \ldots \quad \boldsymbol{u}_K\} \tag{20}$$

且$\boldsymbol{u}_i$为一个$\mathrm{n}*1$的列向量，则P为一个$\mathrm{n}*\mathrm{K}$的矩阵。

因此我们有P^{T}为一个$\mathrm{K}*\mathrm{n}$的矩阵。并计算降维后的矩阵

$$\mathbf{Y}=\mathrm{P}^{\mathrm{T}}\mathbf{X} \tag{20}$$

即为降维后的数据。

重建：

使用P矩阵可以降维，那么我们也可以通过P矩阵还原回原来的维度。

$$\mathrm{X}'=\mathrm{P}\mathbf{Y} \tag{21}$$

即可完成重建。

4. 信噪比计算

我们通过 PSNR 峰值信噪比来衡量图像质量。

对于一个给定的大小为m * n的干净图像I和噪声图像K，均方误差MSE定义为

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{22}$$

此时PSNR定义为：

$$\mathrm{PSNR} = 10 * \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{23}$$

其中MAX_I^2定义为图片可能的最大像素值。如果每个像素都由 8 位二进制来表示，那么就为 255。通常，如果像素值由B位二进制来表示，那么

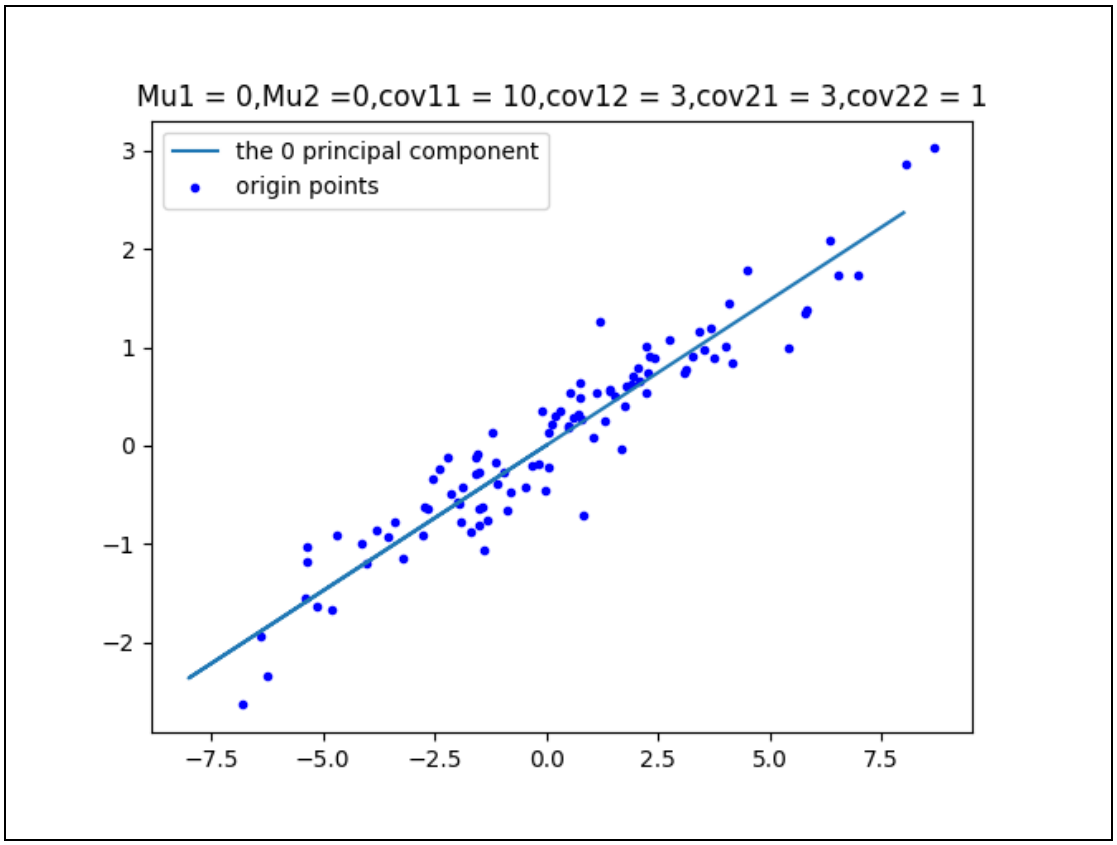
$$MAX_I = 2^B - 1 \tag{24}$$

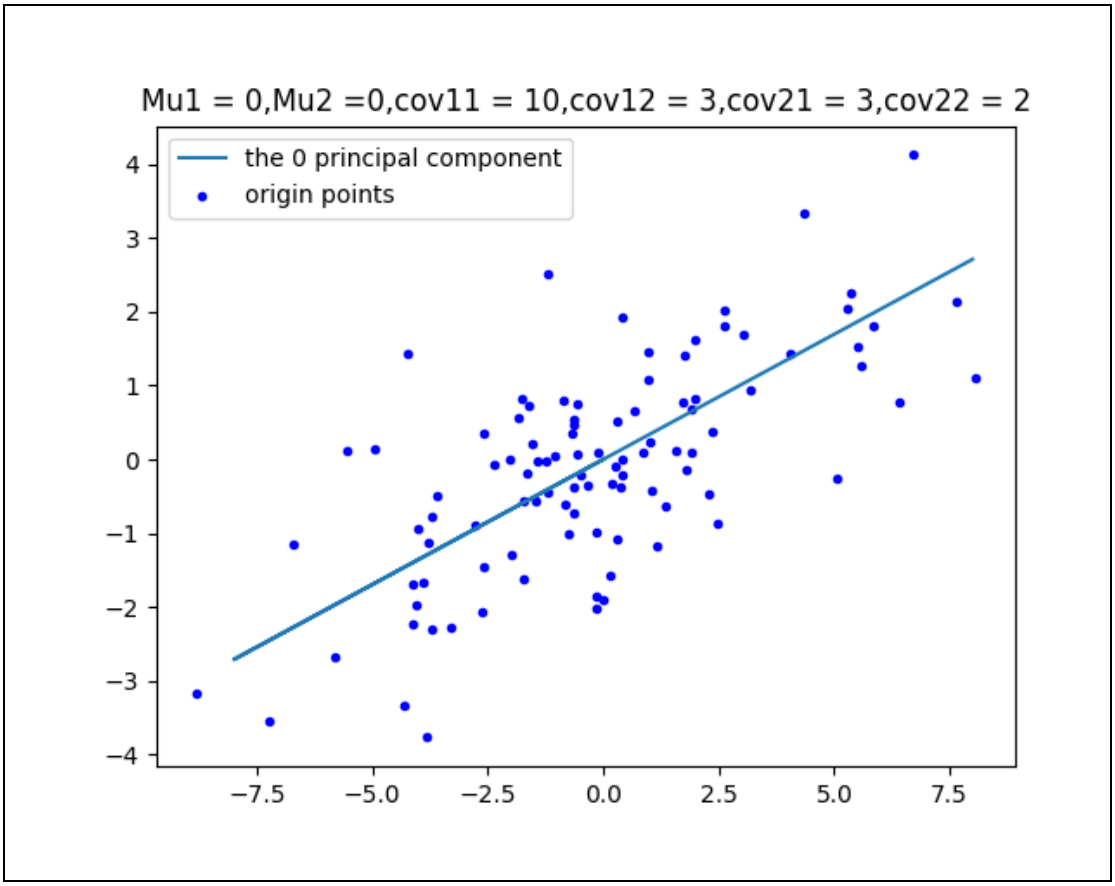
通常针对unit8数据，最大像素值为 255，针对浮点型数据，最大像素值为 1.

四、实验结果与分析

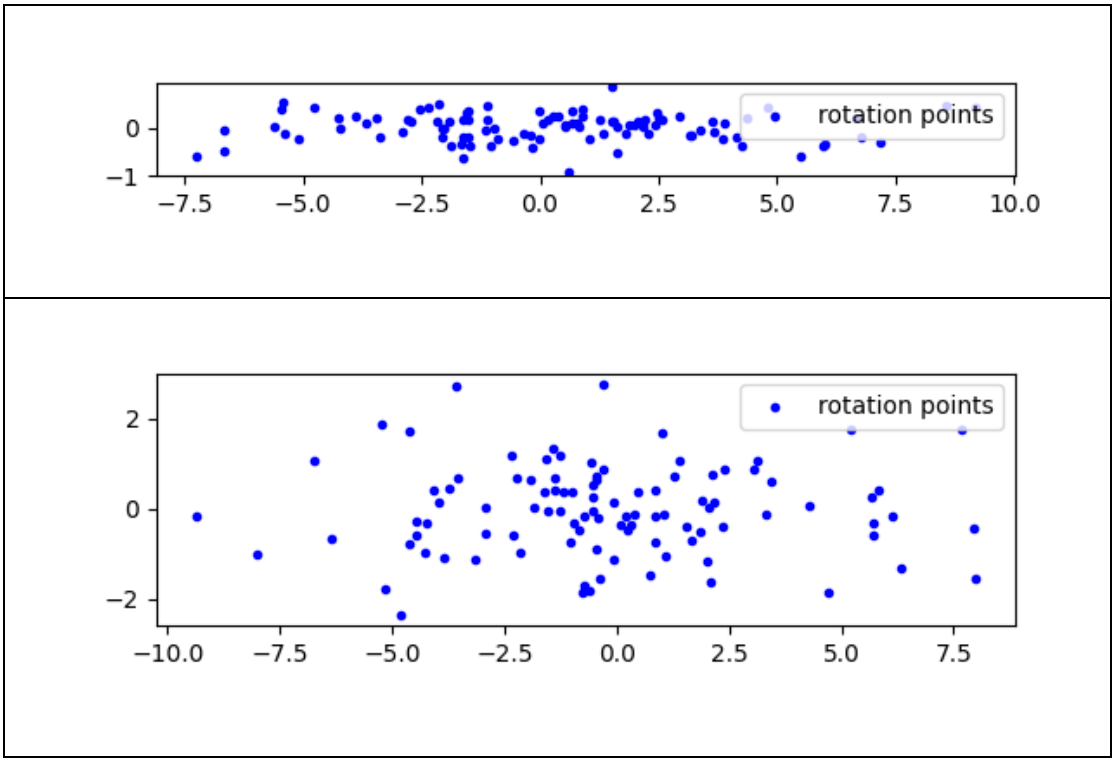
1. 对二维数据点测试

使用 np.random.multivariate_normal 生成二维数据，样本的个数为 100。
使用PCA降维，此时降至一维如图所示。





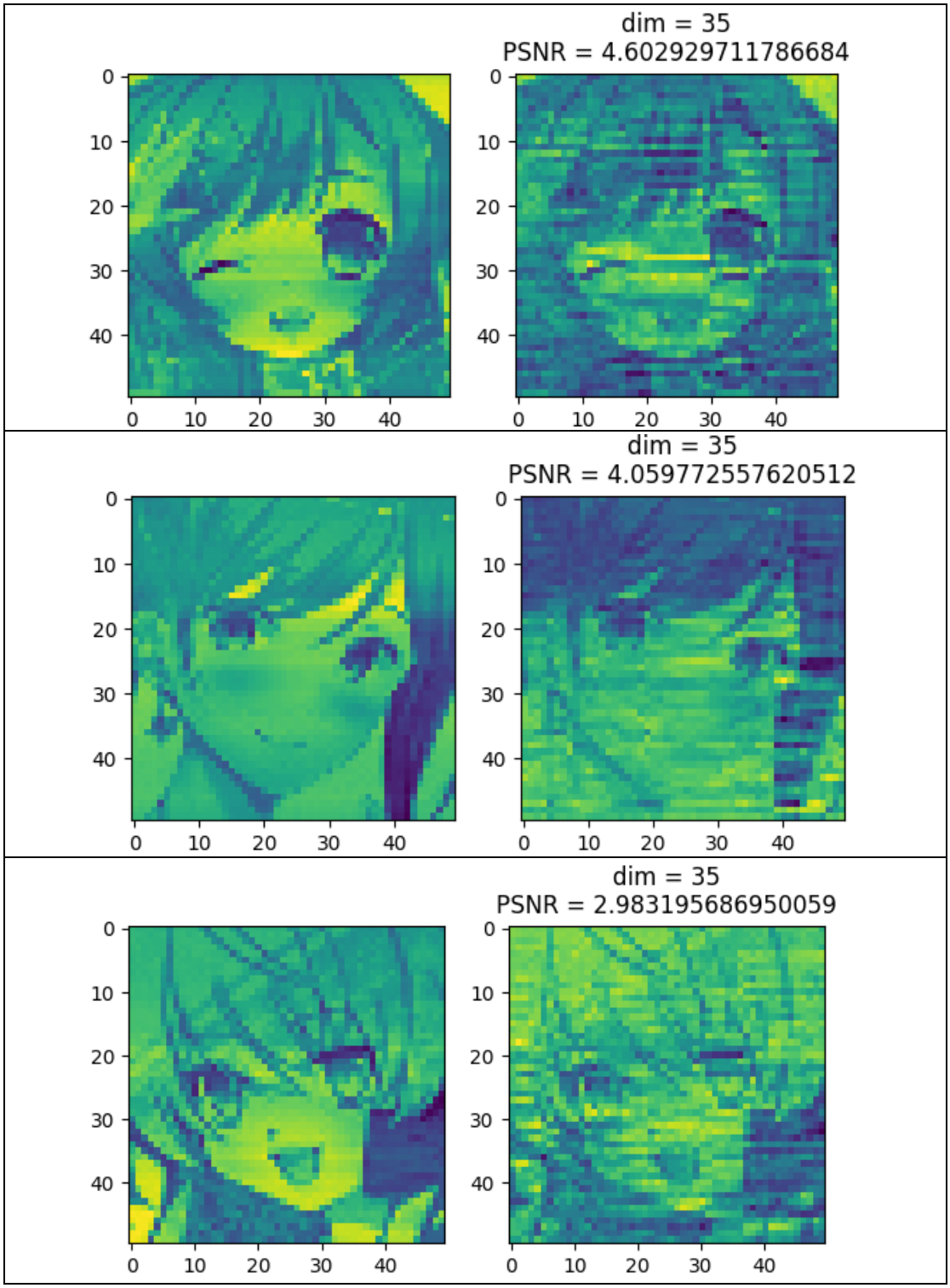
将所有点旋转至新的坐标轴如图所示：



2. 对图片测试

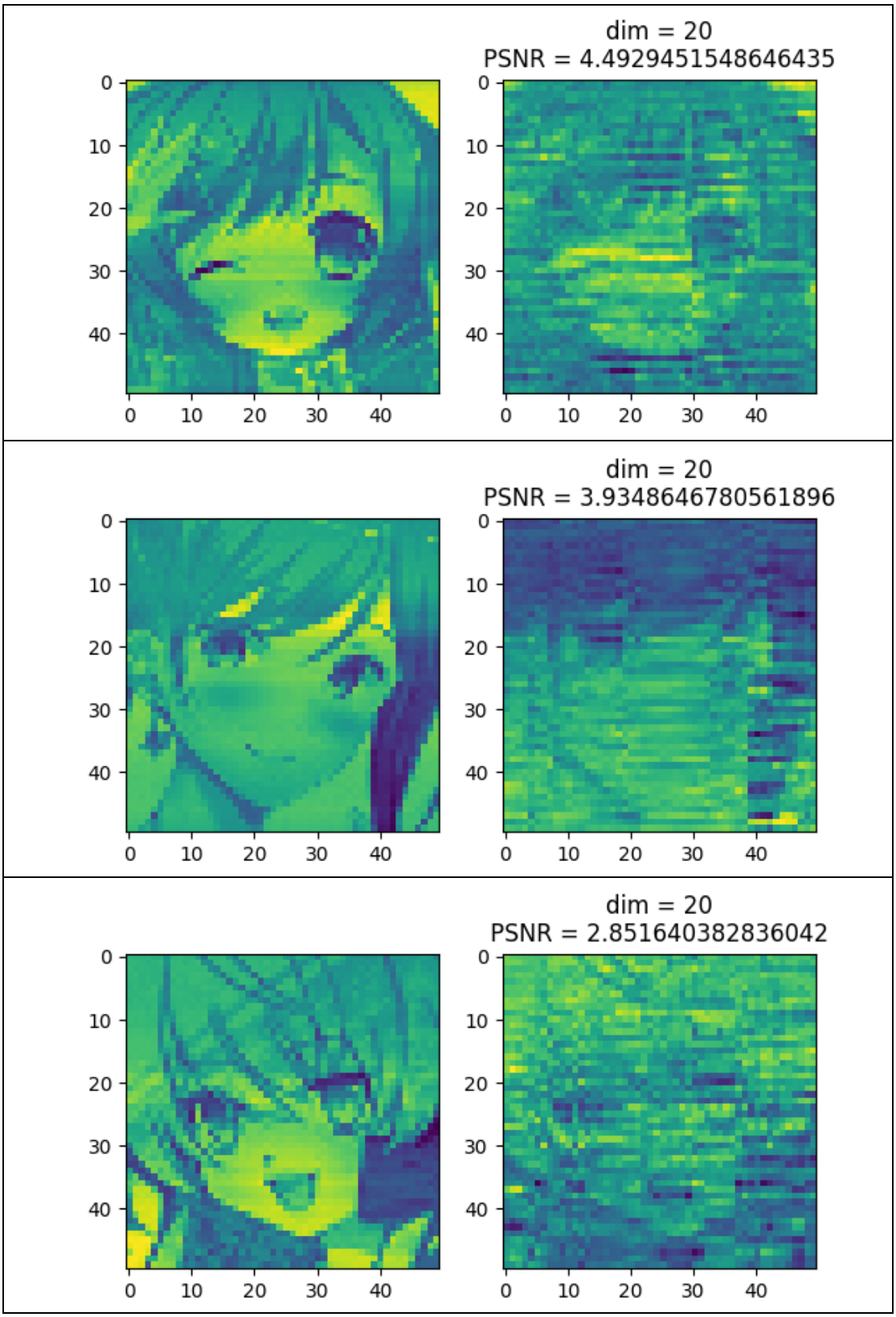
对人脸数据测试：

实验初期的做法为对每一张人脸数据50 * 50的数据看作是50列的数据。此时对该 50 列数据作降维处理，将其降为K维数据，再通过特征向量构成的矩阵对数据重建，并通过PSNR衡量图片质量。



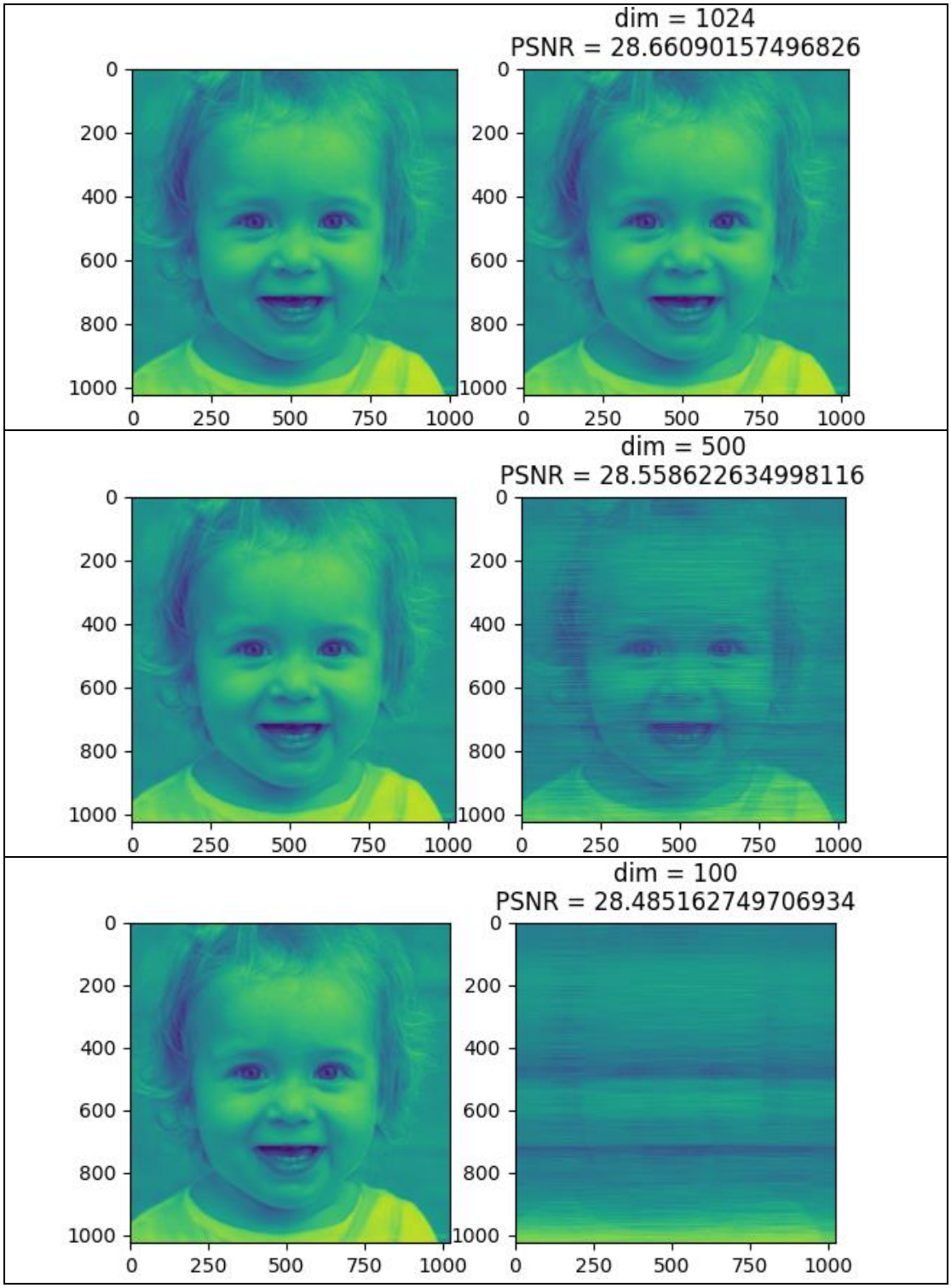
可以看到，在降维图片为 35 维时，经过重建还原有较好还原效果，虽然丢失部分细节，但整体还原效果较好。

当维度降至 20 维时，图片细节丢失更多，PSNR 进一步下降。



本部分实验还对其他人脸进行了测试：

如图，对$1024 * 1024$的图片进行降维并使用特征矩阵进行还原。

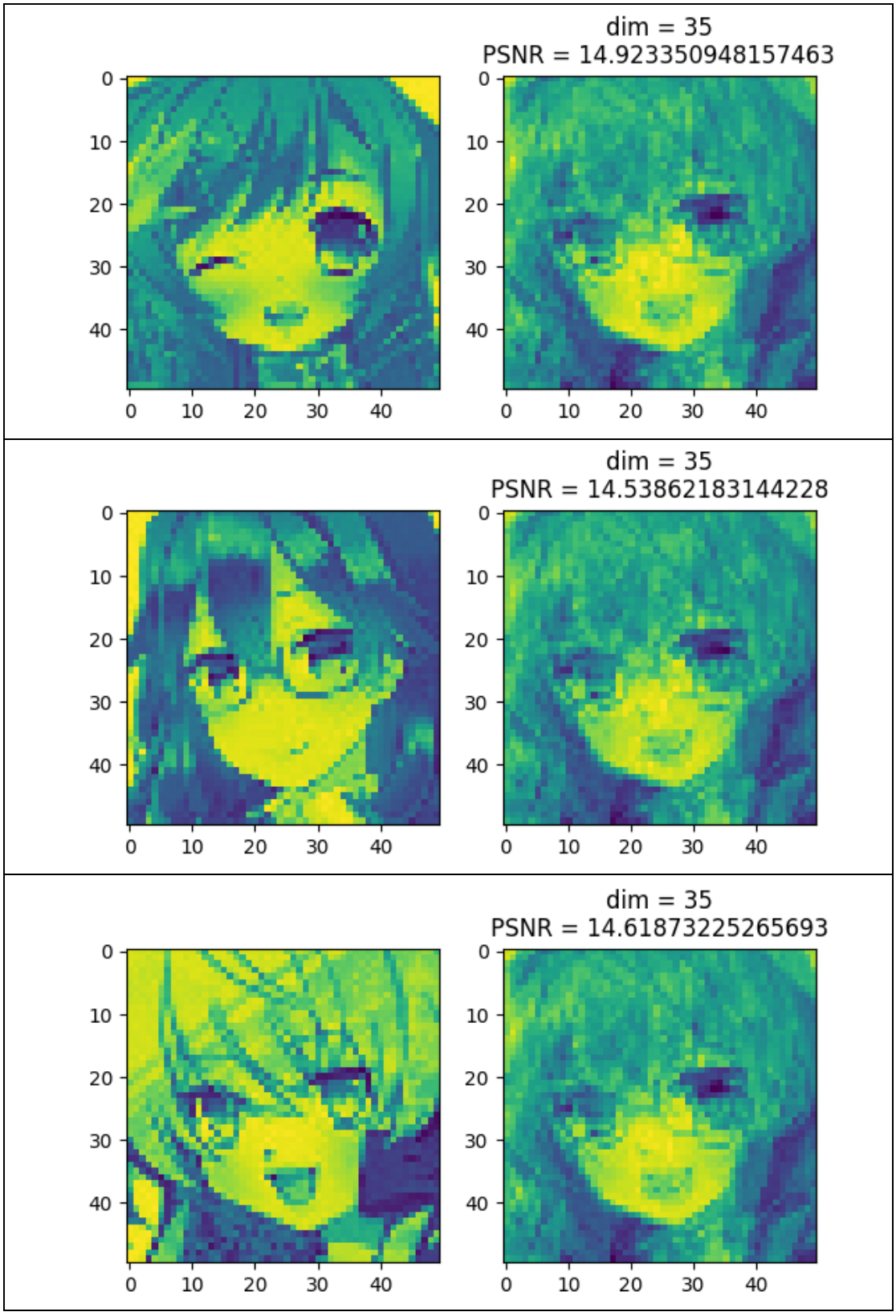


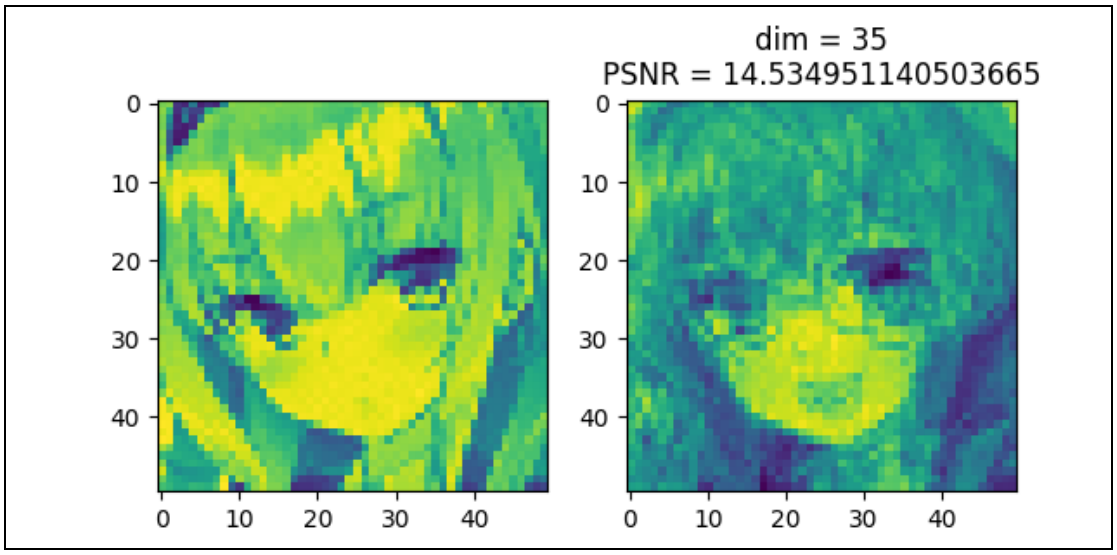
可以看到，在使用 PCA 降维后，在位原图像维度时，图像非常清晰，在 500 维时，图片仍有较好的效果，在 100 维时，仍然可以依稀看出人脸。当维数降低时，PSNR 进一步下降，符合预期效果。

3. 对图片做PCA降维的更正

经老师指导，实际上本实验需要将所有图片用一个矩阵表示，在此基础上，使用PCA降维，并重建。

实验仍然对$50*50$的图片进行测试。





本部分实验将四张图片表示为2500 * 4的矩阵，并对其降维成35 * 4的矩阵，再对其还原得到新的图片，此时重构出的人脸更趋向于平均脸。可以观察到四幅图重构出的人脸都比较相近，PSNR非常接近。

五、结论

在机器学习中经常会碰到一些高维的数据集，它们会占用计算机的内存和硬盘空间，而且在运算时会减缓速度。降维能够使得数据量被压缩，加快运算速度，减小储存空间，以及方便可视化的观察数据特点。

六、参考文献

[1] Pattern Recognition and Machine Learning
[2] 机器学习 周志华著 北京：清华大学出版社，2016 年 1 月.
[3] https://zhuanlan.zhihu.com/p/77151308
[4] https://www.jianshu.com/p/1977c98967f5

七、附录：源代码（带注释）

main.py

```
from PCAimage import *
from PCAdata import *


if __name__ == '__main__':
    # 原图片 1024 维
    # 降成 500 维重构
    # Face(1024)
    # Face(500)
    # Face(100)

    # 原图片 50 维
    # 降成 35 维重构
    # 非平均脸
    # FaceData(20)
    # 平均脸
    FaceDataImprove(50)


    # 生成二维数据并做旋转
    # DealData()
```

GenerateData.py

```
import numpy as np

def generate2DimensionalData(Mu1, Mu2, cov11, cov12, cov21, cov22, Num, noiseSigma1, noiseSigma2):
    """
    生成二维高斯分布数据
    :param Mu1: 维度 1 的平均值
    :param Mu2: 维度 2 的平均值
    :param cov11: 维度 1 的方差
    :param cov12: 维度 12 的协方差
    :param cov21: 维度 21 的协方差
    :param cov22: 维度 2 的方差
    :param Num: 生成样本点的个数
    :param noiseSigma1: 维度 1 噪声的方差
    :param noiseSigma2: 维度 2 噪声的方差
    :return: 分别返回两个维度的样本点
```

```
    """
    mean = np.array([Mu1, Mu2])
    cov = np.array([[cov11, cov12], [cov21, cov22]])
    Data = np.random.multivariate_normal(mean, cov, Num)
    X1 = Data[:, 0]
    X2 = Data[:, 1]
    # 添加高斯噪声
    GuassNoise1 = np.random.normal(0, scale=noiseSigma1, size=Num)
    GuassNoise2 = np.random.normal(0, scale=noiseSigma2, size=Num)
    X1 = X1 + GuassNoise1
    X2 = X2 + GuassNoise2
    return X1, X2
```

DealData.py

```
from PIL import Image
import numpy as np
import os


def readData(Path):
    """
    从文件中读取图像数据
    :return: 数据，数据的维度，数据的个数，文件目录
    """
    Data = []
    fileDirs = os.listdir(Path)
    for filename in fileDirs:
        image = Image.open(os.path.join(Path, filename)).convert("L")
        data = np.array(image, dtype=np.float32)
        Data.append(data)
    Data = np.array(Data)
    Dimension = Data.shape[2]
    Numbers = len(Data)
    return Data,Dimension,Numbers,fileDirs


def saveToFile(AfterProcessingData,FileDirs,DirPath):
    """
    保存到文件
    :param AfterProcessingData:处理后的数据
    :param PictureDimension: 图片的维度
    :param FileDirs: 文件的名字
    :return:
```

```
    """
    for i in range(AfterProcessingData.shape[0]):
        DataPiece = AfterProcessingData[i]
        im = Image.fromarray(DataPiece)
        im = im.convert('L')
        im.save(os.path.join(DirPath,"Processed_"+FileDirs[i]))
```

PCAdata.py

```
from GenerateData import *
import matplotlib.pyplot as plt
from PCAModel import *

def PlotStraightLine(KEigenvalueVector):
    for i in range(0,1):
        X1 = [0]
        X2 = [0]
        if i ==0:
            for x in range(-8, 9):
                X1.append(x)
                X2.append(KEigenvalueVector[i][0] / KEigenvalueVector[i][1] * x)
        else:
            for x in range(-1, 2):
                X1.append(x)
                X2.append(KEigenvalueVector[i][0] / KEigenvalueVector[i][1] * x)
        plt.plot(X1,X2,label="the {} principal component".format(i))

def DealData():
    """
    使数据旋转
    :return:
    """
    Mu1 = 2
    Mu2 = 0
    cov11 = 10
    cov12 = 1
    cov21 = 1
    cov22 = 1
    X1,X2 = generate2DimensionalData(Mu1,Mu2,cov11=cov11,cov12=cov12,cov21=cov21,cov22=cov22,Num=100,noiseSigma1=0,noiseSigma2=0)

```

```
    plt.title("Mu1 = {},Mu2 ={},cov11 = {},cov12 = {},cov21 = {},cov22 = {}"
.format(Mu1,Mu2,cov11,cov12,cov21,cov22))
    Data = []
    Data.append(X1)
    Data.append(X2)
    Data = np.array(Data)
    FacePCAModel = PCAModel()
    KEigenvalueVector = np.transpose(FacePCAModel.fit(Data,2))
    PlotStraightLine(KEigenvalueVector)
    X = np.vstack((X1,X2))
    plt.scatter(X[0], X[1], marker='.', c='b', label="origin points")
    # plt.axis("scaled")
    plt.legend()
    plt.show()
    # 旋转
    KEigenvalueVector  = KEigenvalueVector.T
    temp = np.zeros(KEigenvalueVector[1].shape)
    temp = temp+KEigenvalueVector[0]
    KEigenvalueVector[0] = np.zeros(KEigenvalueVector[1].shape)
    KEigenvalueVector[0] = KEigenvalueVector[0] +KEigenvalueVector[1]
    KEigenvalueVector[1] = np.zeros(KEigenvalueVector[1].shape)
    KEigenvalueVector[1] = KEigenvalueVector[1] +temp
    KEigenvalueVector = KEigenvalueVector.T
    X = np.dot(KEigenvalueVector,X)
    plt.scatter(X[0],X[1],marker='.',c = 'b',label="rotation points")
    plt.axis("scaled")
    plt.legend()
    plt.show()
```

PCAimage.py

```
from PCAModel import *
from DealData import *
import matplotlib.pyplot as plt

def cal_psnr(im1, im2):
    diff = im1 - im2
    mse = np.mean(np.square(diff))
    psnr = 10 * np.log10(255 * 255 / mse)
    return psnr

def Face(K):
    """
    处理 Face 文件夹
```

```
    :param K:PCA 降维后的维度
    :return:
    """
    # 读取人脸图片
    path = "./Face"
    Data,Dimension,Numbers,FileDirs = readData(path)
    ReconsitutionData = []
    # 训练 PCA 模型
    for i in range(Data.shape[0]):
        # 初始化 PCA 模型
        FacePCAModel = PCAModel()
        FacePCAModel.fit(Data[i],K)
        # 得到 PCA 处理后重构的数据
        ReconsitutionData.append(FacePCAModel.Reconsitution())
    ReconsitutionData = np.array(ReconsitutionData)
    # 保存到文件
    Dirpath = "./ProcessedFace"
    saveToFile(ReconsitutionData,FileDirs,Dirpath)
    for len in range(Data.shape[0]):
        path = "./Face/2.jpg"
        image = Image.open(path).convert("L")
        image1 = np.array(Data[len],dtype=np.uint8)
        image2 = np.array(ReconsitutionData[len],dtype=np.uint8)
        psnr = cal_psnr(image1,image2)
        plt.figure()
        plt.subplot(1,2,1)
        plt.imshow(image)
        plt.subplot(1,2,2)
        plt.imshow(ReconsitutionData[len])
        plt.title("dim = " + str(K) + "\nPSNR = " + str(psnr))
        plt.show()



def FaceData(K):
    """
    处理 FaceData 文件夹
    :param K:PCA 降维后的维度
    :return:
    """
    # 读取人脸图片
    path = "./FaceData"
    Data, Dimension, Numbers, FileDirs = readData(path)
```

```
    ReconsitutionData = []
    # 训练 PCA 模型
    for i in range(Data.shape[0]):
        # 初始化 PCA 模型
        FacePCAModel = PCAModel()
        FacePCAModel.fit(Data[i], K)
        # 得到 PCA 处理后重构的数据
        ReconsitutionData.append(FacePCAModel.Reconsitution())
    ReconsitutionData = np.array(ReconsitutionData)
    # 保存到文件
    Dirpath = "./ProcessedFaceData"
    saveToFile(ReconsitutionData, FileDirs, Dirpath)
    for i in range(10):
        plt.figure()
        psnr = cal_psnr(Data[i],ReconsitutionData[i])
        plt.subplot(1,2,1)
        plt.imshow(Data[i])
        plt.subplot(1,2,2)
        plt.imshow(ReconsitutionData[i])
        plt.title("dim = " + str(K) + "\nPSNR = " + str(psnr))
        plt.show()

def FaceDataImprove(K):
    """
    处理 FaceData 文件夹
    :param K:PCA 降维后的维度
    :return:
    """
    # 读取人脸图片
    path = "./FaceData"
    Data, Dimension, Numbers, FileDirs = readData(path)
    DataList = []
    for i in range(Data.shape[0]):
        DataVector = np.empty(0)
        # 转为行向量
        DataTranspose = np.transpose(Data[i])
        for j in range(Data[i].shape[1]):
            DataVector= np.hstack((DataVector,DataTranspose[j]))
        DataList.append(DataVector)
    DataArray = np.array(DataList)
    DataArray = np.transpose(DataArray)
    FacePCAModel = PCAModel()
     FacePCAModel.fit(DataArray, K)
     ReconsitutionData = FacePCAModel.Reconsitution()
```

```
    # 将每一列改为行
    ReconsitutionData = np.transpose(ReconsitutionData)
    ImagesList = []
    for i in range(ReconsitutionData.shape[0]):
        ImagesList.append(np.transpose(np.reshape(ReconsitutionData[i],(Dimension,-1))))
    ImagesArray = np.array(ImagesList,dtype=np.uint8)
    # ImagesArray = np.array(ImagesList)
    # 保存到文件
    Dirpath = "./ProcessedFaceDataImprove"
    # saveToFile(ImagesArray , FileDirs, Dirpath)
    for i in range(4):
        plt.figure()
        psnr = cal_psnr(Data[i],ImagesArray[i])
        plt.subplot(1,2,1)
        plt.imshow(Data[i])
        plt.subplot(1,2,2)
        plt.imshow(ImagesArray[i])
        plt.title("dim = " + str(K) + "\nPSNR = " + str(psnr))
        plt.show()
```

PCAModel.py

```
import numpy as np

class PCAModel:
    def __init__(self) -> None:
        """
        对其初始化
        """
        super().__init__()
        self.Data = np.empty(0)
        self.K = 0
        self.CovarianceMatrix = np.empty(0)
        self.KEigenvalueVector = np.empty(0)
        self.Average = np.empty(0)
        self.AfterProcessingData = np.empty(0)

    def zeroValue(self):
        """
        将给定的数据进行零值化
        :return:
        """
        averageList = []
```

```
        Data = self.Data
        for i in range(Data.shape[0]):
            sum = np.sum(Data[i])
            average = sum/Data.shape[1]
            averageList.append(average)
            for j in range(Data.shape[1]):
                Data[i][j] = Data[i][j]-average
        self.Data = Data
        self.Average = np.array(averageList)

    def generateCovarianceMatrix(self):
        """
        生成协方差矩阵
        :return:
        """
        N = self.Data.shape[1]
        CovarianceMatrix = np.zeros((self.Data.shape[0],self.Data.shape[0]))

        Data = np.transpose(self.Data)
        for i in range(self.Data.shape[1]):
            CovarianceMatrix = CovarianceMatrix+np.dot(np.reshape(Data[i],(-
1,1)),np.reshape(Data[i],(1,-1)))
        CovarianceMatrix = CovarianceMatrix/(N-1)
        self.CovarianceMatrix = CovarianceMatrix


    def takeKArray(self):
        """
        取前 K 行作为矩阵
        :return:
        """
        Eigenvalue,EigenvalueVector = np.linalg.eig(self.CovarianceMatrix)
        ArgSort = np.argsort(Eigenvalue)
        VectorList = []
        for i in range(self.K):
            for j in range(Eigenvalue.shape[0]):
                if ArgSort[j] == i:
                    VectorList.append(EigenvalueVector[j])
        self.KEigenvalueVector = np.transpose(np.array(VectorList))


    def AfterProcessing(self):
        """
        降维后的数据
```

```
        :return:返回降维后的数据
        """
        AfterProcessingData = np.dot(self.KEigenvalueVector.T,self.Data)
        # for i in range(returnData.shape[0]):
        #     for j in range(returnData.shape[1]):
        #         returnData[i][j] = returnData[i][j]+self.Average[i]
        self.AfterProcessingData = AfterProcessingData

    def fit(self,Data,K):
        """
        对给定的数值适配模型
        :param Data: 给定的数据
        :param K: 降维后的数据的维度
        :return:
        """
        self.Data = Data
        self.K = K
        assert self.Data.shape[0] >= self.K
        self.zeroValue()
        self.generateCovarianceMatrix()
        self.takeKArray()
        return self.KEigenvalueVector

    def Reconsitution(self):
        self.AfterProcessing()
        ReconsitutionData = np.dot(self.KEigenvalueVector,self.AfterProcessi
ngData)
        # ReconsitutionData = ReconsitutionData +self.Average
        for i in range(ReconsitutionData.shape[0]):
            for j in range(ReconsitutionData.shape[1]):
                ReconsitutionData[i][j] = ReconsitutionData[i][j]+self.Avera
ge[i]
        return ReconsitutionData
```