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： 考查课

实验题目： PCA模型实验

学号：120L021329

姓名： 蒋世鑫

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）

1. 实验要求及实验环境
2. 实验要求
3. 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。
4. 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）。
5. 实验环境

PycharmCommunity+Anconda3(Python3.8)+Windows

* 1. 设计思想（本程序中的用到的主要算法及数据结构）

1.PCA的基本思想

主成分分析算法（PCA）是最常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中，并期望在所投影的维度上数据的信息量最大（方差最大），以此使用较少的数据维度，同时保留住较多的原数据点的特性。

PCA降维的目的，就是为了在尽量保证“信息量不丢失”的情况下，对原始特征进行降维，也就是尽可能将原始特征往具有最大投影信息量的维度上进行投影。将原特征投影到这些维度上，使降维后信息量损失最小。

2.PCA的步骤

* 去除平均值（中心化）

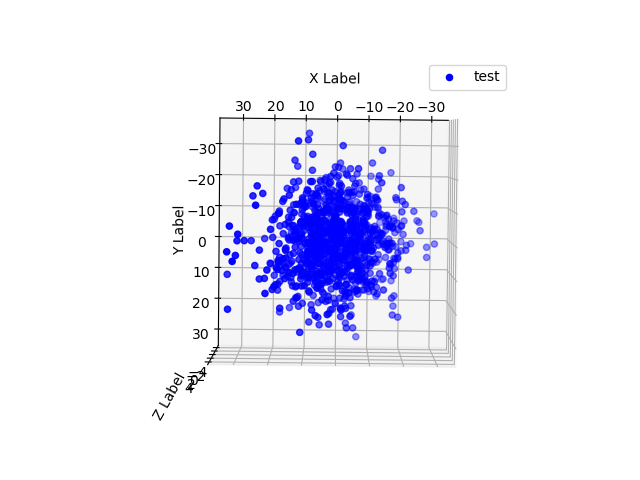
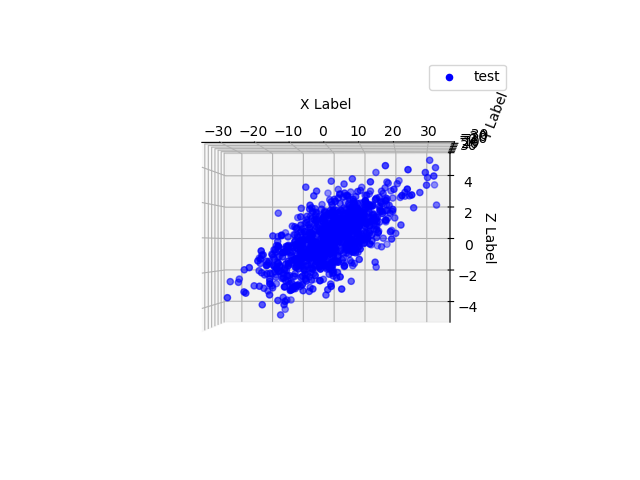
其中表示第k个特征。

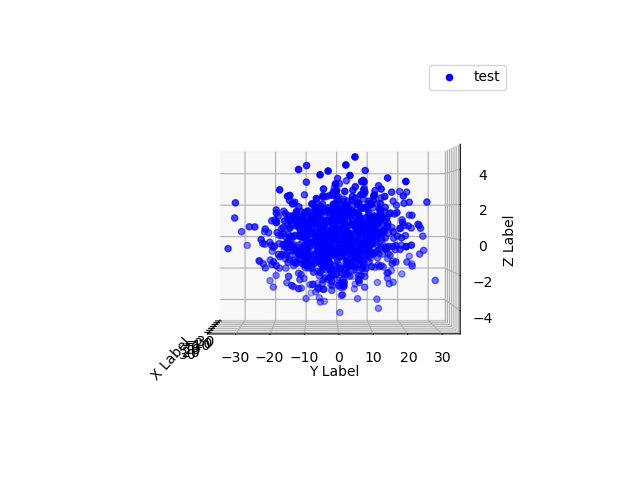
* 计算协方差矩阵
* 计算协方差矩阵的特征值和特征向量
* 将特征值排序
* 保留前N个最大的特征值对应的特征向量
* 将原始特征转换到上面得到的N个特征向量构建的新空间中（最后两步，实现了特征压缩）。

1. 实验结果与分析

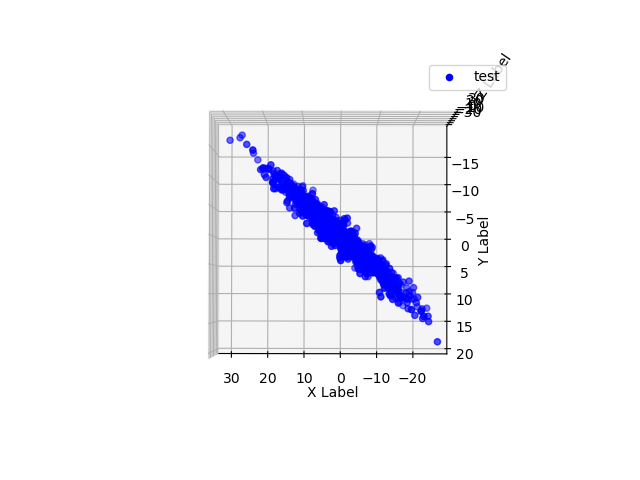
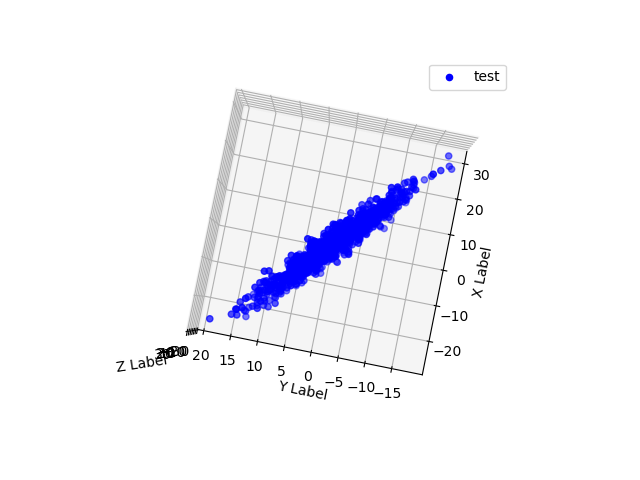
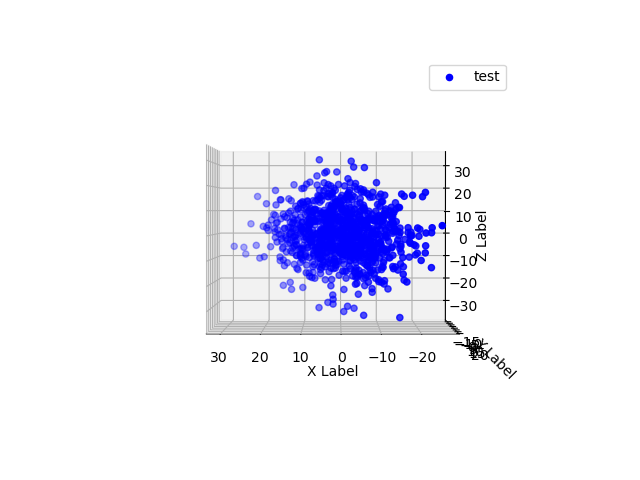
1.三维数据的压缩

（1）原始数据



（2）PCA压缩后的数据



（3）分析

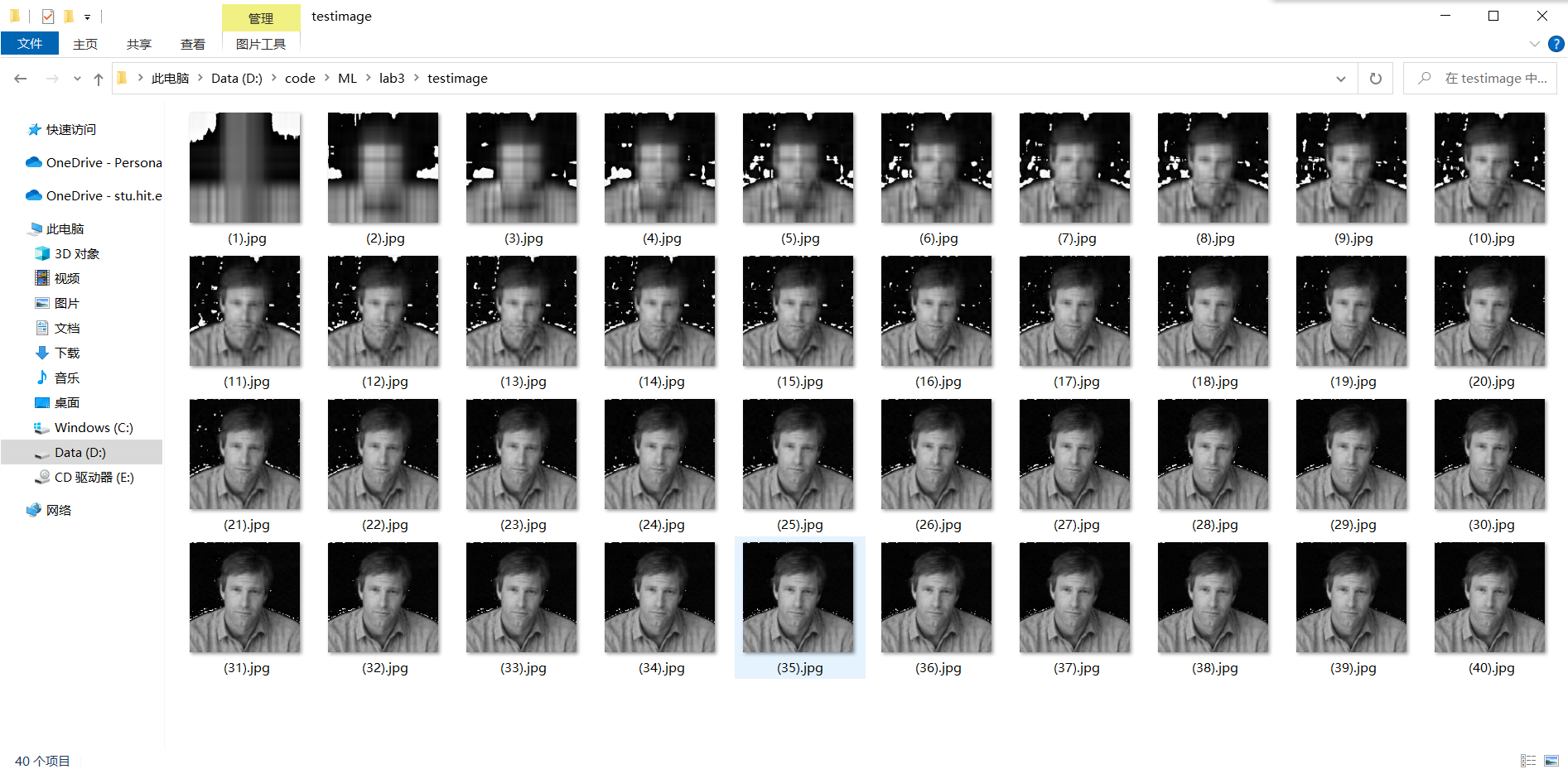
经过压缩之后的数据，可以较好地用二维投影面来表示原有的数据。实现了从3维到2维的压缩。

2.人脸图像的压缩

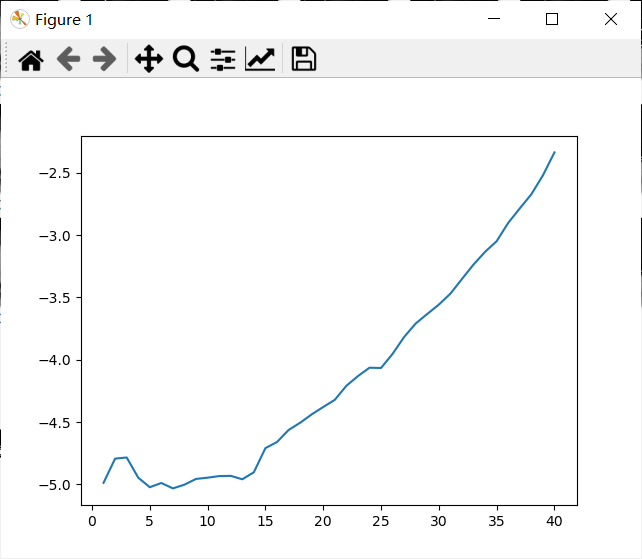
（1）原始图像（来自MTFL数据集）



（2）取前k个特征向量重构（k取1~40）



随着特征向量数量的增多，图像也越来越清晰。而信噪比也越来越大，说明重构的图像与原图像越接近。

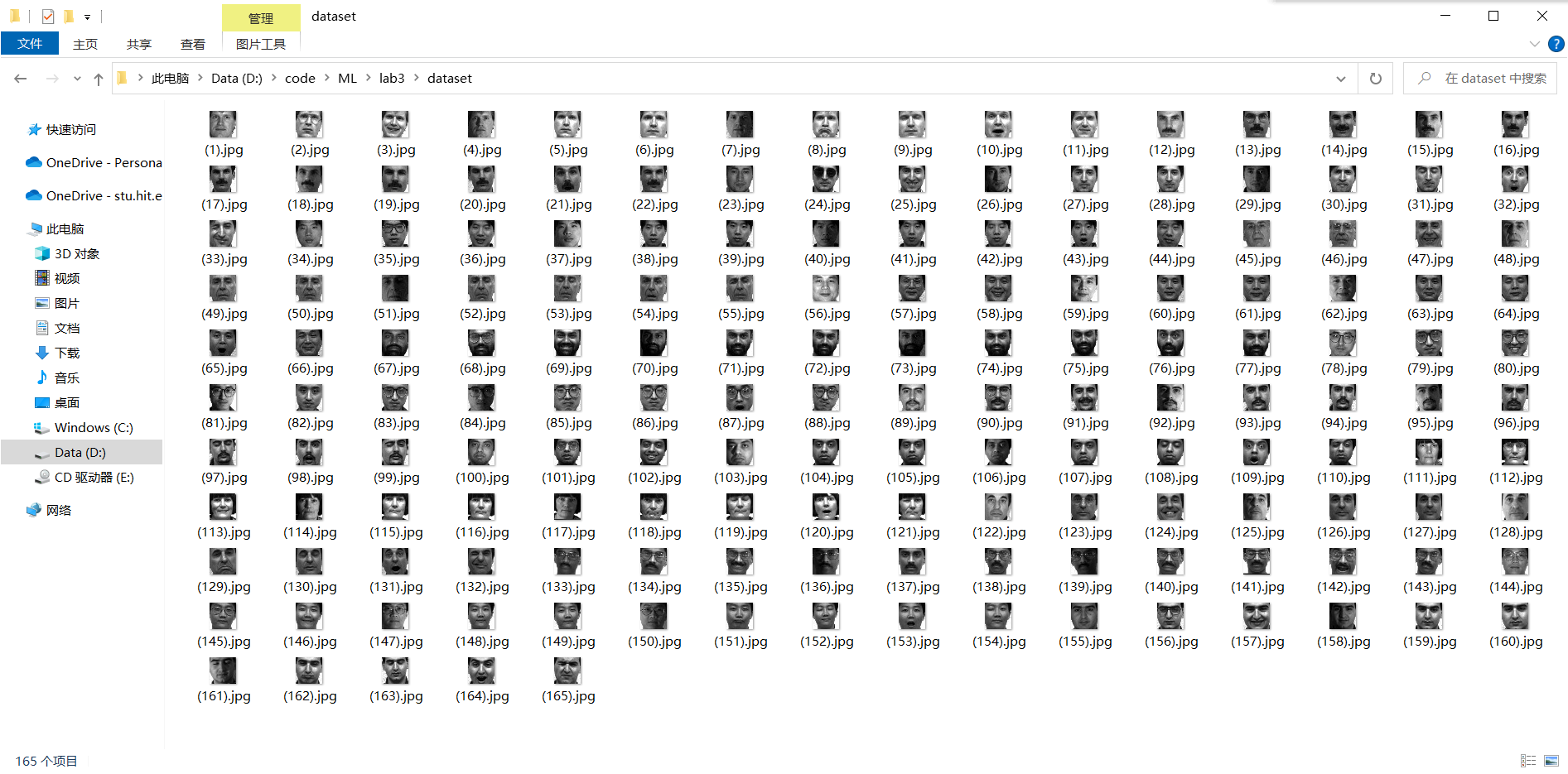


1. 对多张图像的特征提取

对多张图像的特征提取，更多的是得到一个平均脸。

1. 原始图像





1. 平均脸VS原始脸



1. 利用PCA重构（k取100~1）

随着特征数的增多，重构图像越来越接近原始图像。



1. 结论
2. PCA优点

* 使得数据集更易使用；
* 降低算法的计算开销；
* 去除噪声；
* 使得结果容易理解；
* 完全无参数限制。

1. PCA缺点

* 如果用户对观测对象有一定的先验知识，掌握了数据的一些特征，却无法通过参数化等方法对处理过程进行干预，可能会得不到预期的效果，效率也不高；
* 特征值分解有一些局限性，比如变换的矩阵必须是方阵；
* 在非高斯分布情况下，PCA方法得出的主元可能并不是最优的。

1. 参考文献

[机器学习--主成分分析(PCA)算法的原理及优缺点 - 泰初 - 博客园 (cnblogs.com)](https://www.cnblogs.com/lsm-boke/p/11760224.html)

[CodingLabs - PCA的数学原理](http://blog.codinglabs.org/articles/pca-tutorial.html)

[(95条消息) deep learning PCA(主成分分析)、主份重构、特征降维\_小明知道的博客-CSDN博客](https://blog.csdn.net/hlx371240/article/details/40459649)

七、附录：源代码（带注释）

PCA1：

import numpy as np

import sys

import cv2

from sklearn.decomposition import PCA

def max(data,maxset):

max=sys.float\_info.min

tempt=-1

for i in range(len(data)):

if data[i]>max and (i not in maxset):

tempt=i

max=data[i]

if tempt!=-1:

print(tempt)

maxset.append(tempt)

else:

print(max)

print("sda")

def centralize(data):

data\_m=np.array(np.ones(data.shape[0]\*data.shape[1])).reshape(data.shape[0],data.shape[1])

data\_mean=[]

result=[]

for i in range(data.T.shape[0]):

data\_mean.append(np.mean(data.T[i]))

result.append(data.T[i]-np.mean(data.T[i]))

print(data\_mean)

result=np.array(result,ndmin=2)

# print(result.shape)

data\_mean = np.array(data\_mean)

print(data\_mean)

# show(data\_mean, "mean")

print(result.T)

# print(type(result[0][0]))

# #将原数据矩阵转为int8类型

for i in range(data.shape[0]):

for j in range(data.shape[1]):

data\_m[i][j]=data[i][j]

for i in range(data\_m.T.shape[0]):

data\_m.T[i]=data\_m.T[i]-np.mean(data\_m.T[i])

# print(type(data\_m[0][0]))

# exit()

#翻转一下

return result.T,data\_mean

def show(w,count):

w=w.reshape(32,32)

cv2.imshow(str(count),w)

filpath="./features/"+"("+str(count)+")"+".jpg"

cv2.imwrite(filpath, w)

def pca(X,n):

# print(F\_X)

print(X)

print(X.shape)

# print(np.dot(X.T,X).shape)

# Y=np.cov(X,rowvar=0)

# exit()

a,b=np.linalg.eig(np.dot(X.T,X))

# a=1

# b=1

for i in range(len(a)):

a=a.real

d=np.array(np.ones(b.shape[0]\*b.shape[1])).reshape(b.shape[0],b.shape[1])

for i in range(b.shape[0]):

for j in range(b.shape[1]):

d[i][j]=b[i][j]

print(a)

print(b)

# print(b[:,1])

w=[]

maxset=[]

for i in range(n):

print(i)

max(a,maxset)

print(maxset)

count=0

for i in maxset:

w.append(d[:,i])

# show(b[:,i],count)

count+=1

print(w)

print(a)

print(b)

return np.array(w),b

PCA\_Image:

import numpy as np

import cv2 as cv

import math

import matplotlib.pyplot as plt

from sklearn import decomposition

# 数据中心化

def centere\_data(dataMat):

rows, cols = dataMat.shape

meanVal = np.mean(dataMat, axis=0) # 按列求均值，即求各个特征的均值

meanVal = np.tile(meanVal, (rows, 1))#扩大矩阵

newdata = dataMat - meanVal

print(newdata)

return newdata, meanVal

# 最小化降维造成的损失，确定k

def Percentage2n(eigVals, percentage):

sortArray = np.sort(eigVals) # 升序

sortArray = sortArray[-1::-1] # 逆转，即降序

arraySum = sum(sortArray)

temp\_Sum = 0

num = 0

for i in sortArray:

temp\_Sum += i

num += 1

if temp\_Sum >= arraySum \* percentage:

return num

# 得到最大的k个特征值和特征向量

def EigDV(covMat, k):

D, V = np.linalg.eig(covMat) # 得到特征值和特征向量

# k = Percentage2n(D, p) # 确定k值

print("降维后的特征个数：" + str(k) + "\n")

eigenvalue = np.argsort(D)

K\_eigenValue = eigenvalue[-1:-(k + 1):-1]

K\_eigenVector = V[:, K\_eigenValue]

return K\_eigenValue, K\_eigenVector

def SNR(img\_raw,img):

sum1=0

sum2=0

print(img)

print(img\_raw)

# exit()

img=np.array(img)

img\_raw=np.array(img\_raw)

for i in range(img\_raw.shape[0]):

for j in range(img\_raw.shape[1]):

sum1+=img\_raw[i][j]\*\*2

print(sum1)

sum2+=(img\_raw[i][j]-img[i][j])\*\*2

return math.log(sum1/sum2,10)\*10

# PCA算法

def PCA(data, k):

dataMat = np.float32(np.mat(data))

# 数据中心化

dataMat, meanVal = centere\_data(dataMat)

# 计算协方差矩阵

print(dataMat.shape)

covMat = np.cov(dataMat, rowvar=0)

print(covMat.shape)

# 选取最大的k个特征值和特征向量

D, V = EigDV(covMat, k)

# 得到降维后的数据

lowDataMat = dataMat \* V

print(lowDataMat.shape)

# 重构数据

reconDataMat = lowDataMat \*V.T+ meanVal

return reconDataMat

if \_\_name\_\_ == '\_\_main\_\_':

img = cv.imread('./gray\_data/(1).jpg',0)

rows, cols = img.shape

print(img)

#pca = decomposition.PCA()

print("降维前的特征个数：" + str(cols) + "\n")

print(img)

print('----------------------------------------')

k=[]

loss=[]

for i in range(40):

k.append(i+1)

PCA\_img = PCA(img, i+1)

print(PCA\_img.shape)

PCA\_img = PCA\_img.astype(np.uint8)

print(PCA\_img)

loss.append(SNR(img,PCA\_img))

cv.imshow('test', PCA\_img)

cv.imwrite("./testimage/"+"("+str(i+1)+")"+".jpg",PCA\_img)

# cv.waitKey(0)

cv.destroyAllWindows()

plt.plot(k,loss)

plt.show()