# 《数据科学与工程算法基础》实践报告

报告题目： 应用PCA、RPCA、2DPCA和KPCA进行图像压缩

姓 名： 周辛娜

学 号： 10195501442

完成日期： 2021.11.24

提供中英文摘要

摘要 [中文]：

本次实验采用四种方法实现了对图像的压缩，分别是PCA、RPCA、2DPCA和KPCA。对于KPCA，由于KPCA的kernel常用有linear，rbf这两种核函数，故在本次实验的数据集上，在不同的超参数取值下根据压缩率、空间节省和运行时间这三个评价指标来判断选用哪种kernel效果更好，实验结果表明linear表现更优。明确选用哪种kernel表现最佳后，再比较以上四种方法在本次实验的数据集上，在不同的超参数取值下根据压缩率、空间节省、重构误差和运行时间这四个评价指标比较这四种方法的优劣。

Abstract [英语]：

In this experiment, four methods are used to achieve image compression, namely PCA, RPCA, 2DPCA and KPCA. For KPCA, the kernel functions linear and RBF are commonly used in KPCA. Therefore, in the data set of this experiment, which kernel has better effect is judged according to the three evaluation indexes of compression rate, space saving and running time under different hyperparameter values, and the experimental results show that Linear has better performance. After determining which kernel has the best performance, the advantages and disadvantages of the above four methods are compared on the data set of this experiment under different hyperparameter values according to the four evaluation indexes of compression rate, space saving, reconstruction error and running time.

1. 项目概述（阐明该项目的科学和应用价值，以及相关工作进展并描述项目的主要内容）

PCA有很多的局限性，例如对噪声非常敏感等。本项目主要利用四种不同的实现方法对图像进行降维处理，除了PCA的其余三种方法均针对PCA的缺陷进行改进。

在实际的应用中，对图片进行降维处理，采用何种方法去实现是非常重要的。本次实验通过比较压缩率、空间节省、重构误差和运行时间这四个评价指标来判断哪种方法更优。

1. 问题描述（问题定义）

**1）RPCA的提出：**

PCA是使用最广泛的降维方法，然而实际应用中，当存在遮挡时，PCA效果较差，而RPCA能够从较大的且稀疏噪声污染的观测数据中恢复出本质上低秩的数据。RPCA正是为了处理PCA不能很好解决的遮挡等问题而提出来的。RPCA希望将一个数据矩阵拆解为一个低秩矩阵和一个稀疏矩阵之和，更通俗而言，这个稀疏矩阵便代表着遮挡，低秩矩阵又能够在PCA上有很好表现。PCA和RPCA区别在于对于误差的假设不同，PCA假设数据误差是服从高斯分布的，即数据噪声较小；RPCA假设数据噪声是稀疏的，并且可能是强的噪声[1]。在本次实验中，RPCA将采用Inexact ALM方法进行优化。

**2）2DPCA的提出：**

在PCA中，对于彩色图，三维矩阵要转化为二维矩阵，对于灰度图，二维矩阵要转化为一维向量，使得图像内在的结构信息丢失[2]，同时由于可能一维向量构造的协方差矩阵尺寸过大，计算复杂度增加，训练的样本如果数量还过少，还容易产生过拟合。而2DPCA直接用原始图像构造协方差矩阵，协方差矩阵更加准确。其基本思路和PCA相同[3]。

**3）KPCA的提出：**

PCA降维是一种线性变化，在高维空间的样本点如果线性不可分的话，到了低维空间依旧线性不可分，为了能够让样本在低维空间线性可分故引入KPCA，KPCA的基本思想是对源空间的点进行非线性高维映射，在映射后的空间再进行PCA降维[4]。由于常用的kernel有linear和rbf等，本次实验便比较了在此次图像集上哪种kernel表现更优。然后再采用更优的kernel去和另外3种实现方式比较。

1. 方法（问题解决步骤和实现细节）

**1）PCA的算法实现：**

1. 样本点去中心化

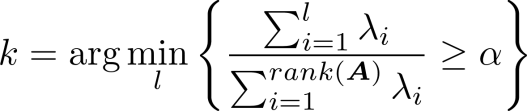


1. 样本协方差矩阵分解

计算样本协方差矩阵，对该协方差矩阵进行分解，得到特征值和特征向量

1. 主成分选取

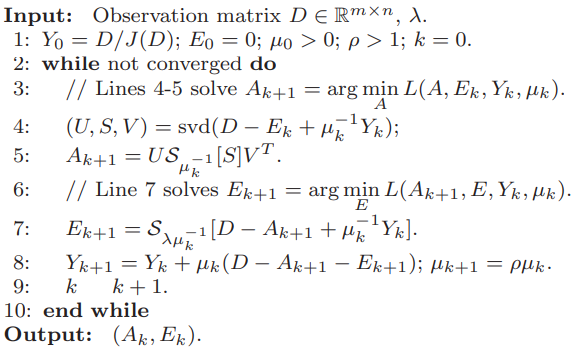
协方差矩阵的特征值按照降序排列，从中选取前k个主成分，使得：



即选取最少数量的主成分，使得这些主成分至少解释原数据α以上的信息。由前k个奇异值所对应的特征向量组成矩阵W

1. 投影到新的空间，进行数据降维，得到新样本点组成的矩阵Z
2. 输出
3. **RPCA的算法实现：**

由Inexact ALM方法优化的RPCA的算法伪代码如下：



最终为了解决这一凸优化问题：



1. **2DPCA的算法实现：**
2. 样本是三维图像矩阵，进行中心化
3. 计算协方差矩阵

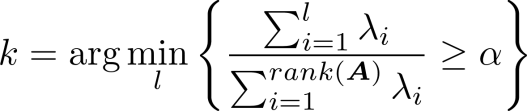


1. 样本协方差矩阵分解

计算协方差矩阵，对该协方差矩阵进行分解，得到特征值和特征向量

1. 主成分选取

协方差矩阵的特征值按照降序排列，从中选取前k个主成分，使得：



即选取最少数量的主成分，使得这些主成分至少解释原数据α以上的信息。由前k个奇异值所对应的特征向量组成矩阵W

1. 投影到新的空间，进行降维
2. 输出

其中步骤3-6与PCA的算法实现2-5是类似的。

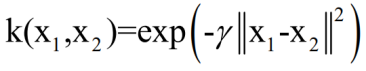
1. **KPCA的算法实现**

首先要明确核函数的选择：

多项式核：

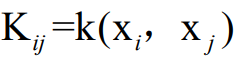


rbf核：

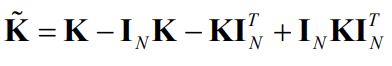


具体实现步骤：

1. 给定样本x，维度 [N, D]
2. 利用核函数计算样本两两之间的关系矩阵K

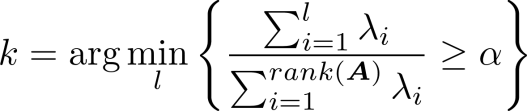


1. 对K进行中心化

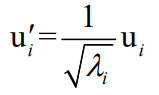


1. 对中心化后的K进行特征值分解
2. 主成分选取

矩阵K的特征值按照降序排列，从中选取前k个主成分，使得：



即选取最少数量的主成分，使得这些主成分至少解释原数据α以上的信息。前k个奇异值所对应的特征向量进行正则化：



进而组成矩阵：



1. 降维



以上是四种方法实现的步骤，具体代码见本文附录。

1. **性能指标**

为了比较压缩率、空间节省、重构误差和运行时间这四个评价指标，还需要在各自的算法实现代码中加入计算压缩率、空间节省等的代码。

1. 压缩率的计算公式为：
2. 空间节省的计算公式为：
3. 重构误差的计算公式为：

重构数据矩阵减去原始数据矩阵的差的二范数

在重构矩阵时，导入图片数据集，分RGB三个通道进行降维，降维得到的RGB的3个矩阵再重构回去，然后堆叠这三个RGB的重构矩阵，将其转化为图片显示，可以对比压缩后的图片与原图的效果。对于一个文件夹中100张图片，共有300个通道，最终计算的重构误差是将这300个通道对应的重构误差累加再取平均。

1. 运行时间是利用python中的datetime模块的datetime.now()函数得到当前和结束时间，两者作差便得到运行时间。
2. 实验结果（验证提出方法的有效性和高效性）
3. 在不同的超参数取值α下，根据压缩率、空间节省和运行时间这三个评价指标来判断选用哪种kernel效果更好
4. 在压缩率表现上：

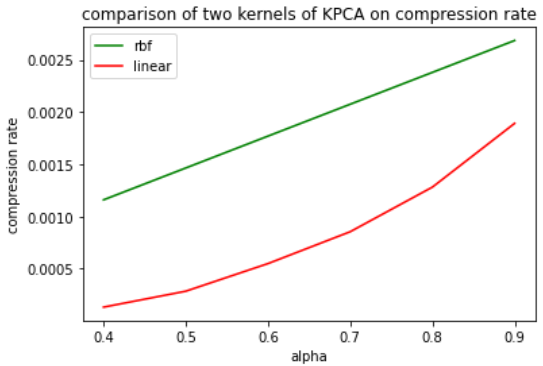


图1

超参数取值α从0.4变化到0.9的过程中，可以看到，两种核对应的压缩率都是逐渐上升的，并且rbf核对应的压缩率都是高于linear核对应的压缩率。

1. 在空间节省表现上：

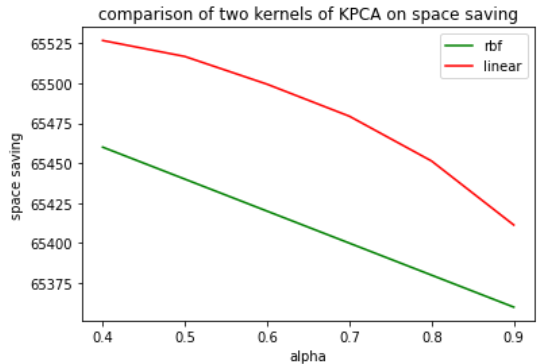


图2

超参数取值α从0.4变化到0.9的过程中，可以看到，两种核对应的空间节省都是逐渐下降的，并且rbf核对应的空间节省都是高于linear核对应的空间节省。

1. 在运行时间方面：

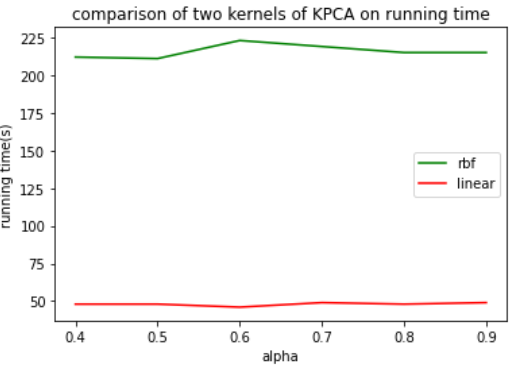


图3

超参数取值α从0.4变化到0.9的过程中，可以看到，两种核对应的运行时间基本是平稳的，并且rbf核对应的运行时间都是远高于linear核对应的运行时间。

综上3个指标，可以得出结论，linear核在该图片数据集上的表现要优于rbf核，故在接下来进行4种方法的比较时，KPCA的核使用linear核。

1. 在不同的超参数取值α下，根据空间节省、压缩率、重构误差和运行时间这四个评价指标来判断选用哪种方法效果更好
2. 在空间节省方面：

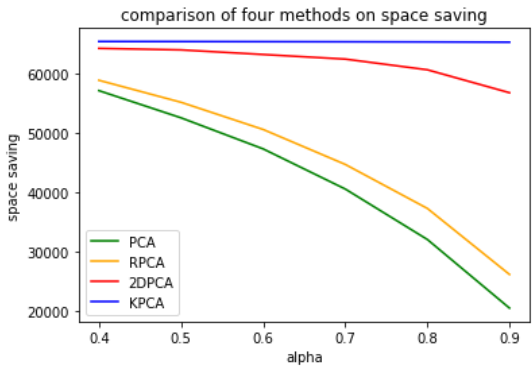


图4

超参数取值α从0.4变化到0.9的过程中，可以看到，RPCA、PCA的空间节省都是逐渐下降，数值以及变化趋势都比较相近，而2DPCA、PCA在空间节省上的变化很小，而且这两种方法在空间节省上要大于RPCA和PCA，在空间节省这一评价指标上，KPCA要优于2DPCA，2DPCA优于RPCA，RPCA优于PCA。

1. 在压缩率方面：

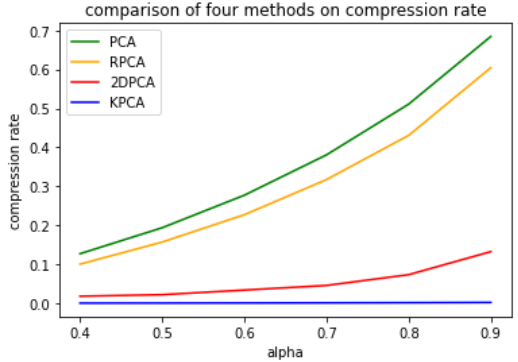


图5

超参数取值α从0.4变化到0.9的过程中，可以看到，RPCA、PCA的压缩率都是逐渐上升，数值以及变化趋势都比较相近，而2DPCA、PCA在压缩率上的变化很小，而且这两种方法在压缩率上要小于RPCA和PCA，在压缩率这一评价指标上，KPCA要优于2DPCA，2DPCA优于RPCA，RPCA优于PCA。

1. 在重构误差方面：

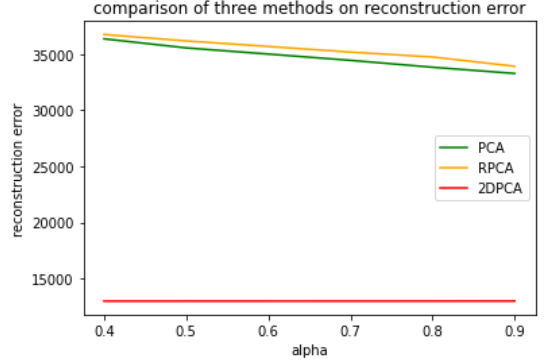


图6

超参数取值α从0.4变化到0.9的过程中，可以看到，RPCA、PCA的重构误差都是逐渐减小，数值以及变化趋势都比较相近，而2DPCA在重构误差上的变化很小，而且远远小于RPCA、PCA，在重构误差这一评价指标上，2DPCA优于PCA，PCA优于RPCA。

1. 在运行时间方面：

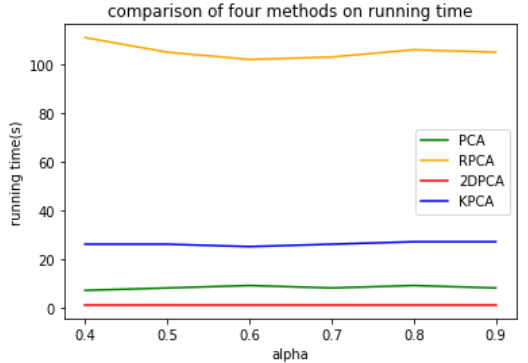


图7

超参数取值α从0.4变化到0.9的过程中，RPCA的运行时间最长，是KPCA的运行时间的三倍以上，KPCA的运行时间大于2DPCA和PCA，2DPCA的运行时间最短，在运行时间这一评价指标上，2DPCA优于PCA，PCA优于KPCA，KPCA优于RPCA。

综合考虑以上4种评价指标，2DPCA最适合本次实验的图像数据集，虽然在空间节省和压缩率上，KPCA略优于2DPCA，但KPCA的运行时间远大于2DPCA，故2DPCA是比较合适的选择。对于KPCA和PCA而言，在空间节省和压缩率上，KPCA远优于PCA，但运行时间远大于PCA，各有优劣。至于RPCA，虽然在空间节省和压缩率上略优于PCA，但在重构误差上略高于PCA，特别是考虑到它的运行时间最长，可以说远超PCA，故RPCA在本次实验的数据集上的表现是不太好的。

1. 结论（对使用的方法可能存在的不足进行分析，以及未来可能的研究方向进行讨论）

本次实验采用了四种方法实现对图像的压缩。对于KPCA，KPCA的kernel并不仅局限于linear，rbf这两种核函数，而且核函数的参数也需要调优，那么如何更好地去优化KPCA也值得进一步研究。对于RPCA，它还有很多优化算法，不仅局限于本次实验中用到的Inexact ALM方法，如何进一步去提高RPCA的表现也是可以去研究的方向。此外，不同方法之间的结合，也可能会进一步带来性能上的提升。

[参考文献]

[1] Z. Lin, M. Chen, L. Wu, and Y. Ma. The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices. UIUC Technical Report UILU-ENG-09-2215, November 2009.

[2] [Xuerong Zhang](https://s.wanfangdata.com.cn/paper?q=%E4%BD%9C%E8%80%85:"Xuerong Zhang"" \t "https://d.wanfangdata.com.cn/periodical/_blank), [Liwei Wang](https://s.wanfangdata.com.cn/paper?q=%E4%BD%9C%E8%80%85:"Liwei Wang"" \t "https://d.wanfangdata.com.cn/periodical/_blank), [Jufu Feng](https://s.wanfangdata.com.cn/paper?q=%E4%BD%9C%E8%80%85:"Jufu Feng"" \t "https://d.wanfangdata.com.cn/periodical/_blank), et al. [The equivalence of two-dimensional PCA to line-based PCA](https://d.wanfangdata.com.cn/periodical/a1ba40de8fde7e117fea726f31e1d56c" \t "https://d.wanfangdata.com.cn/periodical/_blank)[J]. Pattern Recognition Letters 26 (1), 2005.

[3] Jian Yang, David Zhang, Alejandro F. Frangi, and Jing-yu Yang. Two-Dimensional PCA: A New Approach to Appearance-Based Face Representation and Recognition. IEEE transactions on pattern analysis and machine intelligence, vol. 26, no. 1, Jan. 2004.

[4] B.Scholkopf, A.J.Smola, K.Muller. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10 (5), 1299–1399, 1998.

附录：

|  |
| --- |
| #初始工作，读取图片 , 构造输入矩阵  import numpy as np  import cv2 as cv  import imageio  import matplotlib.pyplot as plt  from PIL import Image  import os  import datetime  data\_path = 'D:/airplane/'  save\_path = 'D:/save/'  files=os.listdir(data\_path)  cnt=0  image = []  for file in files:  img=imageio.imread(data\_path+file)  i = cv.imread(data\_path + file)  i = cv.resize(i, (256, 256), interpolation=cv.INTER\_LINEAR)  image.append(i)  np.save(save\_path + 'data.npy', image)  imgs=np.array(img)  #注意imread返回的是BGR，但并不影响，后续还原图像的时候仍然可以按照此顺序  imgs\_r,imgs\_g,imgs\_b=imgs[:,:,0],imgs[:,:,1],imgs[:,:,2]  im\_r,im\_g,im\_b=np.reshape(imgs\_r,(-1,1)),np.reshape(imgs\_g,  (-1,1)),np.reshape(imgs\_b,(-1,1))  if cnt==0:  imageR,imageG,imageB=im\_r,im\_g,im\_b  cnt=cnt+1  #如果尺寸不对，需要填0  else:  if im\_r.shape[0]<65536:  A=np.zeros(65536-im\_r.shape[0]).reshape(65536-im\_r.shape[0],1) im\_r=np.vstack((im\_r,A))  B=np.zeros(65536-im\_g.shape[0]).reshape(65536-im\_g.shape[0],1) im\_g=np.vstack((im\_g,B))  C=np.zeros(65536-im\_b.shape[0]).reshape(65536-im\_b.shape[0],1)  im\_b=np.vstack((im\_b,C))  if cnt!=1:  #分别得到R、G、B通道的大矩阵  imageR,imageG,imageB=np.column\_stack((imageR,im\_r)),np.column\_stack((imageG,im\_ g)),np.column\_stack((imageB,im\_b))  cnt=cnt+1 |

|  |
| --- |
| def PCA(X,alfa):  #首先求解样本中心点，对应书上第一行算法  center=np.mean(X,axis=0).reshape(1, -1)  #对样本进行中心化，对应书上第二行算法  X=X-center  #求协方差矩阵，对应书上第三行算法  C=np.cov(X)  #求特征值，特征向量，对应书上第四行算法  sigma,U = np.linalg.eigh(C)  sigma = sigma[::-1]  U = U[::-1]  #特征值选取，对应书上第5行算法  rank=np.linalg.matrix\_rank(X)  for k in range(rank):  alpha = sum(sigma[:k])\*1.0/sum(sigma)  if alpha >= alfa:  break  num=i#主特征的个数  #得到W，对应书上第6行算法  W=U[:,range(num)]  #投影到新的空间，对应书上第7-8行算法  Z=np.dot(W.T,X)  #计算压缩率  print("compression rate:")  cr=(np.size(W)+np.size(Z)+np.size(center))/np.size(X)  print(cr)  #计算空间节省  print("space saving:")  ss=np.size(X)-(np.size(W)+np.size(Z)+np.size(center))  print(ss)  #重构图像，并且用二范数求重构误差  recon\_x=np.dot(W,Z)+center  recon\_img=np.uint8(np.absolute(recon\_x))  re\_error = np.linalg.norm(recon\_img.astype(int) - X.astype(int))  return recon\_img,re\_error,cr,ss |

|  |
| --- |
| from numpy.linalg import norm, svd  def RPCA(D, alfa, λ , lim, maxiteration):  #对应算法伪代码第一行，进行初始化  J\_D = np.max([norm(D.ravel(), 2), norm(D.ravel(), np.inf)/λ])  Y = D /J\_D  E = np.zeros(Y.shape)  μ = 1.25 / norm(D.ravel(), 2)#μ>0  ro = 1.55#ro>1  A = np.zeros(Y.shape)  b = 12.  k = 0#表示迭代次数  Z = D-A-E  #对应算法伪代码第二行 while not converged do  while ((k < maxiteration) and (norm(Z, 'fro') / norm(D, 'fro')>=lim)): E\_0 = D-A+(1/μ)\*Y E\_k = np.minimum(E\_0+λ/μ ,0)+np.maximum(E\_0-λ/μ , [0](#_bookmark1))  #对应算法伪代码第四行  U,S,V = svd(D-E\_k+(1/μ)\*Y, full\_matrices=False)  a = (S>1/μ).shape[0]  if a >= b:  b = np.min([Y.shape[1],a+round(0.05\*Y.shape[1])])  else:  b = np.min([a+1,Y.shape[1]])  #对应伪代码第五行  A\_k = np.dot(np.dot(U[:,:a],np.diag(S[:a]-1/μ)),V[:a,:])  A,E = A\_k,E\_k  # print(k)  Z = D-A-E  #对应伪代码第八行  Y = Y+μ\*Z  μ = np.min([μ\*ro, μ\*1.1e7])  k += 1  return PCA(A.T,alfa) |

|  |
| --- |
| data =np.load(save\_path + 'data.npy')  print(data.shape)  data\_r=data[:,:,:,0]  data\_g=data[:,:,:,0]  data\_b=data[:,:,:,0] |

|  |
| --- |
| def twoD\_PCA(X,alfa):  #求样本均值  center = np.zeros((X.shape[1],X.shape[2]))  for i in range(X.shape[0]):  center += X[i,:,:]/(X.shape[0]\*1.0)  #样本中心化  X=X-center  #计算协方差矩阵  G\_t = np.zeros((X.shape[2],X.shape[2]))  for j in range(X.shape[0]):  img = X[j,:,:]  tmp = img-center  G\_t = G\_t + np.dot(tmp.T,tmp)/(X.shape[0]\*1.0)  sigma,u=np.linalg.eig(G\_t)  #对特征值进行排序  idx=np.argsort(sigma)  idx=idx[::-1]  u=u[:,idx]  sigma=sigma[idx]  #特征值选取  sigma\_sum=np.sum(sigma)  for i in range(X.shape[2]):  p\_sigma=p\_sigma+sigma[i]  if p\_sigma>(alfa\*sigma\_sum):  break  num=i#主特征的个数  #得到W  W=u[:,range(num)]  # 压缩后的图像  newX = np.dot(X, W)  #计算压缩率  print("compression rate:")  cr=(np.size(W)+np.size(newX)+np.size(center))/np.size(X)  print(cr)  #计算空间节省  print("space saving:")  ss=np.size(X)-(np.size(W)+np.size(newX)+np.size(center))  print(ss)  #重建图像，变回X，并且用二范数求重构误差  recon\_X=np.dot(newX,W.T)+center  recon\_img=np.uint8(np.absolute(recon\_X))  re\_error = np.linalg.norm(recon\_img.astype(int) - X.astype(int))  return recon\_img,re\_error,cr,ss |

|  |
| --- |
| def linear(x1,x2,a=1,c=0,b=1):  x = np.dot(x1, x2)  x = np.power((a\*x+c),b)  return x  def rbf(x1,x2,gamma = 3):  x = np.dot((x1-x2),(x1-x2))  x = np.exp(-gamma\*x)  return x  def kpca(X, alfa, kernel):  N,D = np.shape(X)  K = np.zeros([N,N])  # 利用核函数计算K  for i in range(N):  for j in range(N):  K[i,j]=kernel(X[i],X[j])  # 对K进行中心化  one\_n = np.ones((N, N)) / N  K = K - one\_n.dot(K) - K.dot(one\_n) + one\_n.dot(K).dot(one\_n)  #求特征值，特征向量，对应书上第四行算法  sigma,U = np.linalg.eigh(C)  sigma = sigma[::-1]  U = U[::-1]  #特征值选取，对应书上第5行算法  rank=np.linalg.matrix\_rank(X)  for k in range(rank):  alpha = sum(sigma[:k])\*1.0/sum(sigma)  if alpha >= alfa:  break  num=i#主特征的个数  # 选取较大的特征值  val = U[num][0]  vector = U[:, range(num)]  # 进行正则  val = val\*\*(1/2)  u = vector/val  # 进行降维  new\_X = np.dot(K, u)  #计算压缩率  print("compression rate:")  cr=(np.size(u)+np.size(new\_X))/np.size(X)  print(cr)  #计算空间节省  print("space saving:")  ss=np.size(X)-(np.size(u)+np.size(new\_X))  print(ss)  return cr,ss |

PCA的重构误差和运行时间的计算代码如下所示，其余三种方法的对应代码均类似PCA，详见four PCA.ipynb文件。

|  |
| --- |
| #在不同超参数alpha下，看reconstruction error, space saving, compressing rate的变化  recon\_error=[]  space\_saving=[]  compression\_rate=[]  running\_time=[]  alpha\_list = [0.4,0.5,0.6,0.7,0.8,0.9]  for i in alpha\_list:  print('----alpha(' + str(i) + ')----')  starttime = datetime.datetime.now()  reconstruction\_error=0  #3个通道分别去做PCA  r\_recon\_matrix,r\_recon\_error,r\_cr,r\_ss=PCA(imageR.T,i)  g\_recon\_matrix,g\_recon\_error,g\_cr,g\_ss=PCA(imageG.T,i)  b\_recon\_matrix,b\_recon\_error,b\_cr,b\_ss=PCA(imageB.T,i)  #将R,G,B三通道的复原成原图  recon\_img=np.dstack((r\_recon\_matrix,g\_recon\_matrix,b\_recon\_matrix))  #recon\_img = Image.fromarray(recon\_img)  #recon\_img.save(str(file))  #100张图片R,G,B的总的重构误差  for i in range (100):  k\_r=r\_recon\_matrix[i].reshape(256,256)  t\_r\_error=np.linalg.norm(k\_r.astype(int)-imgs\_r.astype(int))  k\_g=g\_recon\_matrix[i].reshape(256,256)  t\_g\_error=np.linalg.norm(k\_g.astype(int)-imgs\_g.astype(int))  k\_b=b\_recon\_matrix[i].reshape(256,256)  t\_b\_error=np.linalg.norm(k\_b.astype(int)-imgs\_b.astype(int))  reconstruction\_error=reconstruction\_error+t\_r\_error+t\_g\_error+t\_b\_error  print("reconstruction error:",reconstruction\_error/300)  recon\_error.append(reconstruction\_error/300)  endtime = datetime.datetime.now()  duringtime = endtime - starttime  print('runnig time:', duringtime.seconds, '(s)')  running\_time.append(duringtime.seconds)  space\_saving.append((r\_ss+g\_ss+g\_ss)/300)  compression\_rate.append((r\_cr+g\_cr+b\_cr)/3) |

作图代码，其余图的绘制均类似该段代码，详见four PCA.ipynb文件。

|  |
| --- |
| x\_axix=[]  for i in alpha\_list:  x\_axix.append(i)  plt.title('comparison of two kernels of KPCA on space saving')  plt.plot(x\_axix, space\_saving3, color='green', label='rbf')  plt.plot(x\_axix, space\_saving5, color='red', label='rbf')  plt.legend()  plt.xlabel('alpha')  plt.ylabel('space saving')  plt.show() |