《数据科学与工程算法基础》实践报告

报告题目：图片压缩——PCA、2DPCA、$(2D)^2$PCA 的对比分析

姓　　名：叶秋雨

学　　号：10184102103

完成日期：2021.11.25

摘要

利用主成分分析（PCA）对图像降维，需要将图片转换为一维向量，再计算特征的协方差矩阵，现实中特征的数量可能维数很高，计算机无法计算。2 维主成分分析（2DPCA）基于原始二维图像矩阵构建图像的协方差矩阵，而不需要将图像转化为一维行向量，减少了协方差矩阵的维度。$(2D)^2$PCA 方法，对行、列两个方向都进行 2DPCA 的处理，更好地实现图片降维。本实验通过 SVD 分解实现 PCA 算法，并利用农田、飞机和沙滩三类图片，以欧式距离和 RMSE 作为重构误差评价指标对比分析了 PCA、2DPCA、$(2D)^2$PCA 的降维效果。此外，基于 PCA 算法对比了将 RGB 通道分离后分别降维和一起降维两种方法。并且实现了求解正交矩阵特征值与特征向量的 Jacobi 算法，用于特征向量和特征值的求解。实验结果显示，1）随着变换矩阵选取特征向量数 k 增多，重构误差减小。2）在 RMSE 和欧式距离两种度量重构误差的指标下，均表现出 2DPCA 效果依次优于 PCA，优于$(2D)^2$PCA。3)空间压缩率从高到低依次为 PCA、2DPCA、$(2D)^2$PCA。4）基于 PCA 算法，将彩色图像 RGB 三个通道一起降维优于分离后降维，一起降维能更好减少三个通道间的冗余。

Abstract

Principal Component Analysis (PCA) is used to reduce image dimension. The image needs to be transformed into 1D vectors, and then compute the image covariance matrix. In reality, the dimension is always too high to be calculated by the computer. Two-dimensional Principal Component Analysis (2DPCA) is based on 2D image matrices rather than 1D vectors so the image matrix does not need to be transformed into a vector. Instead, an image covariance matrix is constructed directly using the original image matrices, reducing the dimension of covariance matrix. What's more, Two-directional two-dimensional PCA ((2D) ^{2}PCA) is applied to both directions of row and column to achieve better image dimension reduction. In this experiment, PCA was realized by SVD decomposition. The performance of 2DPCA, (2D) ^{2}PCA and PCA was tested by the reconstruction error which is evaluated by European distance and RMSE. And the experiment was performed on three types of image: agriculture, airplane and beach. In addition, PCA is used to compare the performance of reducing dimension after RGB channel being separated and reducing dimension with RGB channel together. And Jacobi algorithm is also solved to calculate matrix eigenvalue and eigenvector. The result shows that: 1) As the number of selected feature vectors k increases, the reconstruction error decreases; 2) Under RMSE and Euclidean distance, 2DPCA is superior to PCA and (2D) ^{2}PCA in sequence; 3) The spatial compression rate from high to low is PCA, 2DPCA , (2D) ^{2}PCA; 4) Based on PCA algorithm, dimension reduction of RGB three channels in color image is better than dimension reduction after separation, which can reduce redundant information among the three channels.

一、 项目概述

1.1 项目的科学和应用价值

主成分分析（PCA）是一种数据减约和降维技术。在“信息”损失较小的前提下，通过保留重要的主成分，忽略不太重要的主成分保留数据的重要特征，将高维的数据转换到低维。PCA 主要应用于高维数据的降维，数据降维后可以为后续机器学习任务和数据可视化做准备。在实际的生产和应用中，降维在一定的信息损失范围内，可以节省大量的时间和成本。图片是一类典型的高维数据，为避免维度灾难和节约存储空间，对高维图片进行降维是非常必要的。本实验利用农田、机场和沙滩三类地点的图片数据，从重构误差、图片压缩率等指标，对比分析 PCA、2DPCA、$(2D)^2$ PCA 三种降维方法对彩色图像的降维效果。同时，基于 PCA 方法对彩色图片 R、G、B 三个通道分别进行降维再合并与将图片拉成行向量对 RGB 通道一起进行降维的两种方法进行了对比。

1.2 相关工作进展

PCA 方法通过计算数据点协方差矩阵的特征值和特征向量，选择特征值最大（即方差最大）的 k 个特征所对应的特征向量组成的矩阵[1]。进而将数据转换到新的空间当中，实现数据特征的降维。利用传统的 PCA 方法对图片进行降维需要将图片拉伸为一维行向量，丢失了图像行列的相关信息。同时，彩色图像的协方差矩阵非常庞大，比如一副 256*256*3 的 uint8 图像，协方差矩阵的大小为$(256*256*256)^2$，计算机求解该矩阵非常困难。但可以用 SVD 分解对其进行优化。

Jian Yang 和 David Zhang（2004）[2]等人提出了 2DPCA 的方法，2DPCA 基于原始二维图像矩阵构建图像的协方差矩阵，而不需要将图像转化为一维行向量，图像的协方差矩阵可以通过原始图像矩阵直接构造出来，减小了协方差矩阵的维度。在此基础上，Daoqiang Zhang 和 Zhihua Zhou 等人（2005）[3]提出了(2D) 2PCA 方法，对行、列两个方向都进行 2DPCA 的处理，更好地实现降维。

1.3 项目的主要内容

本实验实现了用 SVD 特征分解来解 PCA 算法，复现了论文的 2DPCA 算法[文献]和$(2D)^2$PCA 算法[文献]，利用农田、机场和沙滩三类图像数据，对比分析三种方法对彩色图像的降维效果。同时，基于 PCA 算法，对比了对彩色图像三通道一起降维

和分别降维的效果。本实验还实现了求正定矩阵特征值和特征向量的 Jacobi 算法，用于 PCA 计算协方差矩阵的特征值和特征向量。

二、 问题描述

1、 实现求取正定矩阵特征值和特征向量的 Jacobi 算法

2、实现 PCA 算法，并对训练图片进行降维

3、复现 2DPCA 算法，并对训练图片数据进行降维

4、复现(2D) ^{2}PCA 算法，并对训练图片数据进行降维

5、 对比分析 PCA、2DPCA、(2D) ^{2}PCA 三种方法的降维效果

6、基于 PCA 方法对比彩色图片进行三通道分别降维和一起降维的效果。

三、 实验方法

3.1 PCA 算法（直接求解）

1、读入图片数据与预处理

设有 n 张图片，将原始数据按列组成 m 行 n 列的矩阵$X_{m\times n}$，将X的每一行（代表一个特征）进行中心化，即减去这一行的均值。

2、求解协方差矩阵的特征值与特征向量

$$\mathrm{Cov} = (\mathrm{X}X^T)_{m\times m}$$

求出协方差矩阵$(\mathrm{X}X^T)_{m\times m}$的特征值及特征向量，将特征向量按照对应特征值大小从上到下按行排列成矩阵 P，根据阈值（通常设置为 95%），选取前 k 行组成变换矩阵$W_{k\times m}$

3、图像降维

$$\mathrm{Z}_{k\times n} = \mathrm{W}_{k\times m}\mathrm{X}_{m\times n}$$

得到降维后的矩阵Z。

4、图像重构

$$\mathrm{X}_{m\times n} = W^T_{\mathrm{k\times m}}\mathrm{Z}_{k\times n}$$

由于特征向量相互正交，W为正交矩阵，$\mathrm{W}^{-1} = W^T$。

本实验的图片大小多数都为 256*256*3，若按此方法计算，则协方差矩阵的大小为(256*256*3)^2，矩阵过大计算机无法完成计算，因此我们在此基础上利用 SVD 分解对其进行了优化。

3.2 基于 SVD 求解 PCA

设有 n 条 m 维数据，将原始数据排列成矩阵$X_{m\times n}$

X经过 SVD 分解可以得到：

$$X_{m\times n}=D_{m\times m}\Sigma_{m\times n}V_{n\times n}^{T}$$

其中$D_{m\times m}$是$(\mathrm{X}X^{T})_{m\times m}$的特征向量按列组成的矩阵，$\mathrm{V}_{n\times n}$是$(X^{T}X)_{n\times n}$的特征向量按列组成的矩阵，$\Sigma_{m\times n}$的对角线上是奇异值σ，奇异值与特征值的关系$\sigma=\sqrt{\lambda}$。其中$\mathrm{D}^{T}$即为变换矩阵P，故可得

$$D_{m\times m}=X_{m\times n}(\Sigma_{m\times n}V_{n\times n}^{T})^{-1}$$

因此我们可以求$(X^{T}X)_{n\times n}$的特征值与特征向量，利用Σ、V，计算出矩阵P，并选取前 k 行组成变换矩阵$W_{k\times m}$。

由于测试图像为彩色图像，此方法既可以将图像拉伸为一个一维向量进行计算，也可以对 RGB 三个通道分别进行降维。实验中分别利用这两种方式基于 PCA 对彩色图片进行降维并进行比较。

3.3 2DPCA

1、定义一个线性变换方程

$$\mathrm{Y}=\mathrm{AX}$$

其中，A为原始图像，X是一组正交基，Y是图像投影到特征子空间后的数据。

2、构造图像的协方差矩阵

投影样本的分散度可以用来衡量矩阵X的投影效果，投影后的样本点越分散，说明该投影矩阵更好地保留了主要特征。投影样本的分散度可以通过投影特征向量的协方差矩阵的迹来表征。

$$S_{x}=E(Y-EY)(Y-EY)^{T}=E[AX-E(AX)][AX-E(AX)]^{T}$$

$$=E[(A-EA)X][(A-EA)X]^{T}$$

$$\mathrm{tr}(S_{x})=X^{T}[E(A-EA)^{T}(A-EA)]X$$

故，图像的协方差矩阵为：

$$G_{t}=E[(A-EA)^{T}(A-EA)]$$

定义矩阵G_{t}为图像的协方差矩阵。易证，G_{t}是一个$n\times n$正定矩阵。我们可以利用训练样本来估计G_{t}，假设共有 M 个训练图像，第 j 个训练图像$A_{j}(j=1,2,\ldots,M)$的矩阵大小为$m\times n$，所有图像每个像素点求平均得到$\bar{A}_{m\times n}$，则

$$G_t = \frac{1}{M}\sum_{j=1}^{M}(A_j - \bar{A})^T(A_j - \bar{A})$$

4、计算图像协方差矩阵的特征值和特征向量

计算图像协方差矩阵G_t的特征值与特征向量，并对特征值进行排序，根据阈值选择前 k 个特征值的特征向量构成变换矩阵$W_{n\times k}$

5、特征提取（将图像映射到特征子空间）

$$Y_j = (A_j)_{m\times n}W_{n\times k}$$

图像的大小由$m \times n$降维为$m \times k$，压缩了图像矩阵列向量的位数，行向量位数不变。

6、图像重构

$$\hat{A}_j = Y_j W_{n\times k}^T$$

由于特征向量相互正交，$W_{n\times k}$为正交矩阵，$\mathrm{W}^{-1} = W^T$。

3.4 (2D)^{2}PCA

为了更好的降维，(2D)^{2}PCA 方法在行和列两个方向都进行 2DPCA 处理。

1、由 3.3 2DPCA 方法得到行方向的变换矩阵 X

2、定义一个列方向的线性变换方程

$$\mathrm{Y} = \mathrm{Z}^T\mathrm{A}$$

3、构造图像的协方差矩阵

$$S_x = E(Y - EY)(Y - EY)^T = E[\mathrm{Z}^T\mathrm{A} - E(\mathrm{Z}^T\mathrm{A})][\mathrm{Z}^T\mathrm{A} - E(\mathrm{Z}^T A)]^T$$

$$= E[\mathrm{Z}^T(A - EA)][\mathrm{Z}^T(A - EA)]^T$$

$$\mathrm{tr}(S_x) = Z^T[E(A - EA)(A - EA)^T]Z$$

因此，列变换对应的G_t矩阵

$$G_t = E[(A - EA)(A - EA)^T]$$

4、计算G_t的特征值与特征向量

对列进行 2DPCA，即对上述G_t矩阵求特征值与特征向量，并选择前k个特征值的特征向量，得到一组基Z。

5、图像降维

$$\mathrm{Y} = \mathrm{Z}^T\mathrm{AX}$$

其中，X为行方向上 2DPCA 得到的变化矩阵，Z为列方向上 2DPCA 得到的变换矩

阵，A为原数据。

6、图像重构

$$\hat{A} = ZYX^T$$

因为特征向量相互正交，Z、X均为正交矩阵，正交矩阵的逆矩阵与转置矩阵相等。

3.5 Jacobi 算法计算特征值与特征向量

Jacobi 算法通过迭代计算正定对称矩阵的特征值与特征向量，而测试数据的协方差矩阵正是一种实对称正定矩阵，因此利用 Jacobi 算法实现三种降维方法设计的特征值与特征矩阵的求解。

假设S为目标矩阵

基本步骤：

1、初始化特征向量为单位阵V，即主对角线元素为 1，其他元素为 0.

2、在S的非主对角线元素中，找到绝对值最大的元素S_{ij}。

3、计算$\tan 2\theta$, $\cos\theta$, $\sin\theta$及旋转矩阵G_{ij}。

$$\tan(2\theta) = \frac{2S_{ij}}{S_{jj} - S_{ii}}$$

4、更新S, V

$$S'_{ii} = c^2 S_{ii} - 2scS_{ij} + s^2 S_{jj}$$

$$S'_{jj} = s^2 S_{ii} + 2scS_{ij} + c^2 S_{jj}$$

$$S'_{ij} = S'_{ji} = (c^2 - s^2)S_{ij} + sc(S_{ii} - S_{jj})$$

$$S'_{ik} = S'_{ki} = cS_{ik} - sS_{jk}, \qquad k \neq i, j$$

$$S'_{jk} = S'_{kj} = sS_{ik} + cS_{jk}, \qquad k \neq i, j$$

$$S'_{lk} = S'_{kl}, \qquad k \neq i, j$$

其中$s = \sin\theta$, $c = \cos\theta$。

5、若迭代矩阵S的非主对角元素中最大值小于给定的阈值，或迭代次数等于最大迭代次数，则停止计算；否则，令$S = S'$，重复执行 2~5，直至达到终止条件。

6、求得特征值即为S的主对角线元素以及特征向量。

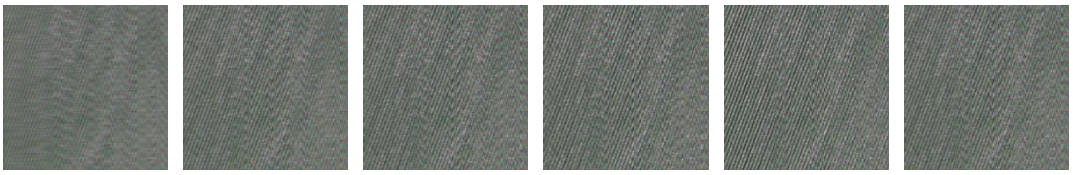
四、 实验结果

4. 1 图像重建结果

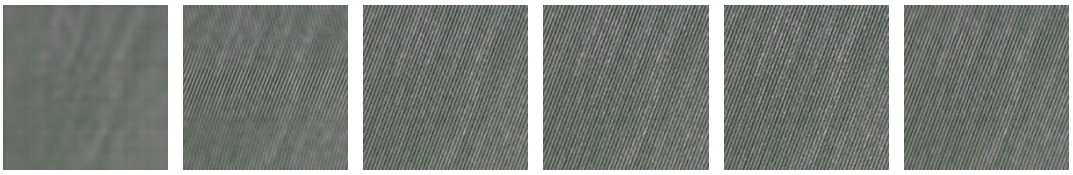
Agriculture

d=20 d=40 d=60 d=80 d=10 α=95%

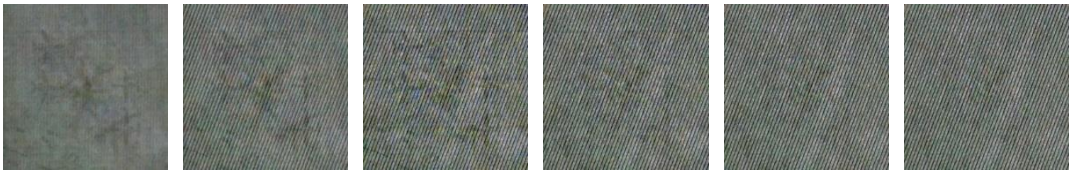
2DPCA



$(2D)^2PCA$

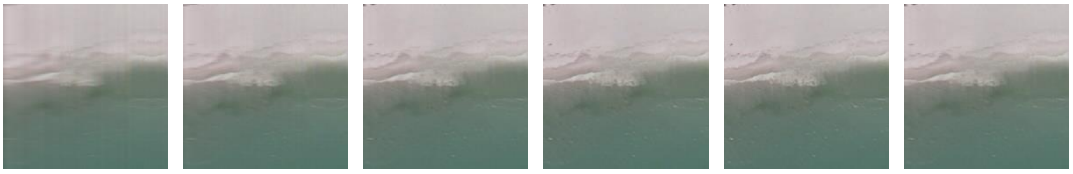


PCA



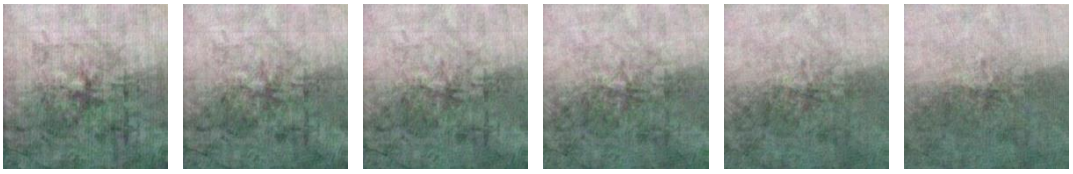
Beach

2DPCA



$(2D)^2PCA$

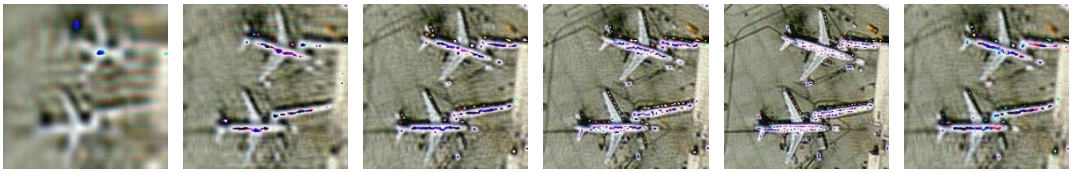
PCA



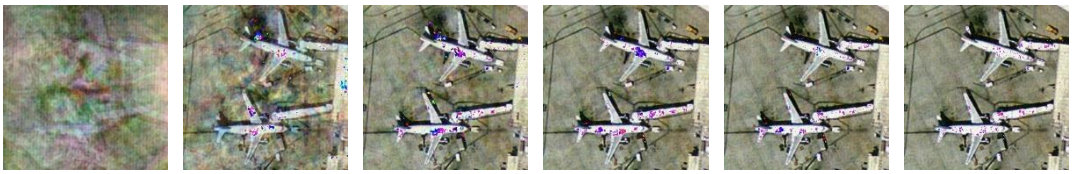
Airplane

2DPCA

$(2D)^2PCA$



PCA(分离 RGB)



PCA(未分离 RGB)

注：α=95%表示保留原始图像 95%得到的重构图像，d=20,40,60,80,100 表示根据特征值从大到小选择的特征向量数量。

由图可以看出，当 k=60 左右时，已经可以较好地重建出原图像了。但是对三类不同的图片同一种方法的降维效果也呈现出差异性。农田图片的主要特征是纹理，$(2D)^2PCA$ 和 PCA 算法表现出色，在 k=60 时，便能清晰地看出图片中的纹理信息。而对于沙滩这类物体边界不清晰的图片，2DPCA、$(2D)^2PCA$ 都有较好表现，PCA 算法很难提划分边界。而对于最后一类飞机的图片，2DPCA、$(2D)^2PCA$ 和 PCA 算法的重构图像中均出现了噪音点，这三种方法都是将图片三个通道分别进行降维的，最后一种方法为 PCA 方法将三个通道合在一起进行降维，有良好的效果，故猜想图中的噪点可能与 RGB 三通道分开降维有关。

总体来看，三种降维方法适用于不同特点的图片降维，随着 k 值增大，保留原图像的特征越多，重构图片的还原度也越高。

4.2 重构误差分析

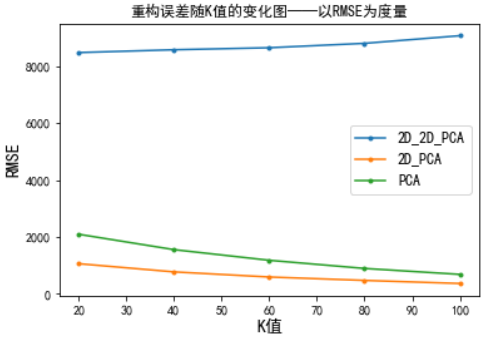


图 1

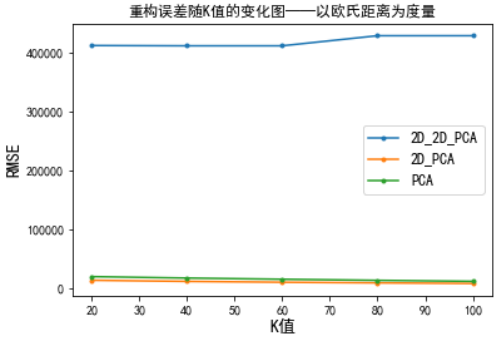


图 2

图 1 展示了在以 RMSE 为度量的重构误差下，PCA、2DPCA、$(2D)^2PCA$ 三种方法 RMSE 随 K 值的变化对比图。K 值越大代表保留原图像的特征越多。由图可见，随着 k 值增大，PCA、2DPCA 的 RMSE 值呈现下降趋势，原因在于 k 越大，保留原始图像的特征也越多，因此重构后的图像与原始图像差异较小。$(2D)^2PCA$ 方法 RMSE 远大于 PCA 和 2DPCA，其次为 PCA，2DPCA 方法 RMSE 最小。因此，在以 RMSE 度量重构误差的情况下，保留原图 95%的信息，三种降维方法效果由好到差以此为 2DPCA、PCA、$(2D)^2PCA$。

图 2 以欧式距离度量降维前后图片的相似性，相似性越高说明降维带来的信息损失越少。由图 2 分析可得，$(2D)^2PCA$ 降维图像信息损失最多，2DPCA 略优于 PCA。

在两种重构误差的度量方式下，三种降维方法均呈现出相似的规律，总体来看，2DPCA 降维最优、PCA 降维其次、$(2D)^2PCA$ 最差。

4.3 压缩率分析

表 1 不同 PCA 方法空间压缩率

方法	原始维度	降维后维度	压缩率
2DPCA	256*256*3	R: 256*55 G: 256*60 B: 256*62	77.1%
$(2D)^2PCA$	256*256	R: 55*62 B: 53*60 G: 47*55	95.33%
PCA(RGB 分离)	65536*296*3	R: 132*296 B: 144*296	99.89%

		G: 140*296	
PCA(RGB 未分离)	196608*296	58*296	99.78%

注：降维后维度指保留原始图像 95%信息的情况下，降维得到的维度，由于将 RGB 三个通道分离进行降维，每个通道降维后的维度可能不同，即每个通道求得的选取特征向量的数量 k 可能不同。

三种降维方法压缩率从高到低依次为 PCA、(2D)^{2}PCA、2DPCA。传统的 PCA 算法压缩后图像的存储空间仅为原始图像的 0.2%，节省了 99.8%
的空间；2DPCA 算法压缩后图像的存储空间为原始图像的 23%，节省了 77%的空间；而(2D)^{2}PCA 算法压缩后图像的存储空间为原始图像的 5%左右，节省了 95%的空间。故，虽然 2DPCA 压缩效果最好，尽可能多得保留了原始的信息，但其节省的空间也最少。而 PCA 尽管在降维效果上略逊色于 2DPCA 算法，但其大大节省了图像的存储空间。

4.4 基于 PCA 彩色图像 RGB 通道两种降维方式比较

基于 PCA 算法对彩色图像进行降维，既可以将图片的 RGB 三个通道直接拉伸为一个行向量，也可以分离出 RGB 三个通道，依次进行降维，最后再将重构后的图片进行合并。

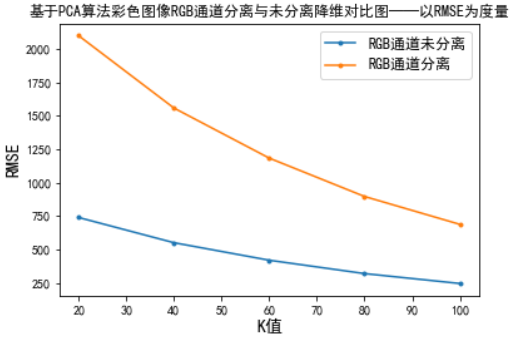


图 3

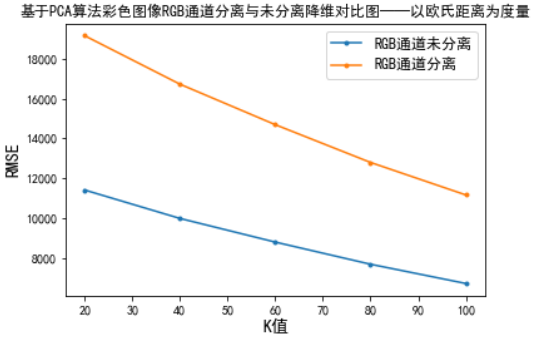


图 4

不论是以 RMSE 还是欧式距离来度量重构误差，均呈现出随 K 值增大而减小的趋势；其中 RGB 三个通道一起降维的重构误差始终小于将其分开进行降维的误差。从节省空间来看，两种方式均表现出色，一起降维后的图片节省了 99.9%的存储空间，分开降维的图片节省了 98.2%的存储空间。总体来看，将彩色图片三个通道一起进行降维似乎要优于将其分开降维。究其原因，可能在于三个通道之间存储的信息可能存在冗余，若将三个通道一起进行降维，能更好地减少冗余，选出一组正交基

能充分表示原始图像。而将 RGB 通道分别进行降维则无法避免三个通道之间的信息冗余。

五、 结论

5.1 实验总结

本次实验利用三种类型的彩色图片，基于欧氏距离和 RMSE 来度量重构误差，从重构误差和空间压缩率对比了 PCA、2DPCA、$(2D)^2$PCA 的降维效果。实验结论：1）基于欧氏距离和 RMSE 来衡量重构误差，实验结果均表现出 2DPCA 算法降维的效果依次优于 PCA、$(2D)^2$PCA。且基本表现出随着选取特征数量 k 的增加，重构误差减小的趋势。2）空间压缩率从高到低依次为 PCA、2DPCA、$(2D)^2$PCA。其中 PCA 算法空间压缩率高达 99.9%，压缩后的图片进展原始图像的 0.1%。4）基于 PCA 算法，将彩色图像 RGB 三个通道一起降维优于分离后降维，原因可能在于一起降维能减少三个通道间的冗余信息。

5.2 实验反思与问题

PCA 为一种线性降维方法，本次实现只探究对比了几种基于 PCA 的线性降维方法的效果，而并没有与非线性降维方法做对比，还可以将 PCA 降维与其他降维方法进行比较分析；其次，实验中的特征向量与特征值均利用自己实现的 Jacobi 算法计算，若矩阵过大，则 Jacobi 算法计算速率很慢，还可以对其进行相应优化

实验过程中的一些思考：

1、实验过程中发现，随着 k 值的增大，$(2D)^2$PCA 的重构误差呈现增长态势，并且$(2D)^2$PCA 的降维表现不如 2DPCA，由于$(2D)^2$PCA 对行列两个方向都进行了降维，那么降维的重构误差是两个方向的乘积，故高于 2DPCA，但与复现的论文结果不太相符，还值得进行进一步探索分析。

2、利用欧式距离和 RMSE 分别作为评价指标，三种方法的表现有所不同，故降维方法的降维效果可能会受到评价指标的影响。

3、本次实验中通过将 RGB 通道分离依次进行降维的三种方法，在对飞机类的图片进行重构时，图中均出现黑点；但使用将 RGB 一起进行降维的方法则无此现象。并没有想清楚这一问题的原因。

总体来说，这次实验虽然花费了很多时间和精力，但也学习到了很多东西。通过本次实验对最基本的 PCA 已经掌握，同时不断地查阅资料、阅读论文，也学习和了

解到许多在最基础的 PCA 方法上进行优化与改进的 PCA，加大了学习的深度。更重要的是，本次实验与以往的实验不同，需要学习更规范的论文排版以及查阅和复现一些参考文献，类似于在培养我们学习写作一篇规范的学术论文，让我收获很多。

参考文献：

[1] Andrzej Maćkiewicz, Waldemar Ratajczak, Principal components analysis (PCA), Computers & Geosciences, Volume 19, Issue 3, 1993, Pages 303-342, ISSN 0098-3004, https://doi.org/10.1016/0098-3004(93)90090-R.

[2] Jian Yang, D. Zhang, A. F. Frangi and Jing-yu Yang, "Two-dimensional PCA: a new approach to appearance-based face representation and recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 1, pp. 131-137, Jan. 2004, doi: 10.1109/TPAMI.2004.1261097.

[3] Daoqiang Zhang, Zhi-Hua Zhou, (2D)2PCA: Two-directional two-dimensional PCA for efficient face representation and recognition, Neurocomputing, Volume 69, Issues 1–3, 2005, Pages 224-231, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2005.06.004.

附件：

注：附件中为实验过程中涉及的核心代码，实验完整代码已附在文件压缩包中也可至 github 查看(git@github.com:RachelllYe/Algorithm_Foundation.git)

• PCA 核心代码：

```
# Jacobi 算法精度
eps = 1e-10
# Jacobi 算法迭代次数
T = 10000
def pca(pic_matrix):
    # m 表示图片的特征数
    m = pic_matrix.shape[0]
    # n 表示图片的数量，即样本数
    n = pic_matrix.shape[1]

    # -----------------对数据做中心化操作----------------------#
    # --------------矩阵的每一行特征减去其所在行的均值------------#
    x_avg = np.average(pic_matrix, axis=1).reshape(m, 1)
    pic_matrix_center = (pic_matrix - x_avg)

    # --------------求 P'·P 的特征值与特征向量------------------#
    # 不必求协方差矩阵的特征值与特征向量，因为图片的维度过高，难以存储下 Memory Over Flow
    p = pic_matrix_center

    # 可以不用求系数，常数不会影响
    matrix = np.dot(p.T, p)
    mat, vector, value = func.jacobi(matrix, n, eps, T)
    eig_value = abs(np.array(value))
    # -------求奇异值构成的对角阵-------#
    sin_value = np.diag(np.sqrt(eig_value))
    # 特征向量
    vector = np.array(vector)

    # ------将特征向量归一化-------------#
    # np.linalg.norm 求解矩阵二范数
    # 对称矩阵，其特征向量是正交的
    vector = vector / np.linalg.norm(vector, axis=0)
    w = np.dot(p, np.linalg.inv(np.dot(sin_value, vector.T)))
    # 得到变换矩阵 W
    w = w.T

```

```
    # -----------------将特征值和特征向量按照从大到小排列------------#
    w, eig_value = func.sort_eigen(w, eig_value)

    # -------------选取特征值中前 k 大的-------------#
    # 阈值 alpha 限定主成分对原数据的解释能力，通常设在 95%
    # 调用 choose_dim 确定 k 的大小，即图片降维后的维度
    alpha = 0.95
    k = func.choose_dim(eig_value, alpha)
    trans_matrix = w[0:k, :]
    return trans_matrix


path = "F:/PCA_Project/Images/all_pics"
# ----------------完成了图片的读入，并以矩阵的方式表示----------------#
# -------------矩阵的大小为 m*n，n 为图片的数量，m 为特征数------------#
pic_Matrix = func.read_pic_vector(path)
trans_Matrix = pca(pic_Matrix)
# 得到降维后的图片矩阵
pic_PCA = np.dot(trans_Matrix, pic_Matrix)
# 通过降维后的图片进行重构
pic_PCA_reconstruct = np.dot(trans_Matrix.T, pic_PCA)
```

注：实验中将 RGB 通道分离后降维和一起降维均使用的 PCA 算法，只是读入矩阵的形式不同。

• 2DPCA 核心代码

```
# Jacobi 算法精度
eps = 1e-10
# Jacobi 算法迭代次数
T = 10000
def pca_2d(pics):
    pics = np.array(pics)
    size = pics[0].shape
    # 取得归一化后的所有图片对应位置像素的均值
    mean = np.average(pics, axis=0)

    cov_row = np.zeros((size[1], size[1]))
    for i in range(pics.shape[0]):
        # 每张图片减去所有图片的均值
        diff = pics[i] - mean
        # 生成协方差矩阵
        cov_row = cov_row + np.dot(diff.T, diff)
```

```
    cov_row = cov_row/cov_row.shape[0]

    # 返回协方差矩阵特征值与特征向量
    m, row_evec, row_eval = func.jacobi(cov_row, 256, eps, T)
    # 将特征值从大到小排列
    row_evec, row_eval = func.sort_eigen(row_evec, row_eval)
    alpha = 0.95
    k = func.choose_dim(row_eval, alpha)
    print(k)
    x = row_evec[:, 0:k]
    return x


path = "F:/PCA_Project/Images/all_pics"
channels = func.read_pic_split(path)
number = len(channels[0])
pic_de = []

for c in range(3):
    # 得到每个通道的变换矩阵
    trans_X = pca_2d(channels[c])
    # 图片降维
    p = np.dot(channels[c][150], trans_X)
    # 图片重构
    res = np.dot(p, trans_X.T)
```

- $(2D)^2PCA$

```
# Jacobi 算法精度
eps = 1e-10
# Jacobi 算法迭代次数
T = 10000

def pca_2d_2d(pics):
    pics = np.array(pics)
    size = pics[0].shape

    # 取得归一化后的所有图片对应位置像素的均值
    mean = np.average(pics, axis=0)

    cov_row = np.zeros((size[1], size[1]))
    for i in range(pics.shape[0]):
        # 每张图片减去所有图片的均值
        diff = pics[i] - mean
```

```
        # 生成协方差矩阵
        cov_row = cov_row + np.dot(diff.T, diff)
    cov_row = cov_row/cov_row.shape[0]

    m, row_evec, row_eval = func.jacobi(cov_row, 256, eps, T)
    # 返回协方差矩阵特征值与特征向量
    # 直接调用现有库计算特征值与特征向量
    # 将特征值从小到大排列，并将索引赋值给 sorted_index
    row_evec, row_eval = func.sort_eigen(row_evec, row_eval)
    alpha = 0.95
    k = func.choose_dim(row_eval, alpha)
    # print(k)
    # k = 100
    x = row_evec[:, 0:k]

    # 将图片的宽为新生方阵的 m
    cov_col = np.zeros((size[0], size[0]))
    for i in range(pics.shape[0]):
        diff = pics[i] - mean
        cov_col += np.dot(diff, diff.T)
    cov_col = cov_col/cov_col.shape[0]
    # 直接调包计算特征值与特征向量
    m, row_evec, row_eval = func.jacobi(cov_row, 256, eps, T)
    alpha = 0.95
    row_evec, row_eval = func.sort_eigen(row_evec, row_eval)
    k = func.choose_dim(col_eval, alpha)
    # k = 20
    z = row_evec[:, 0:k]
    print(k)
    return x, z


path = "F:/PCA_Project/Images/all_pics"
channels = func.read_pic_split(path)
pic_de = []
number = len(channels[0])

# 对 RGB 三个通道分别降维
for c in range(3):
    # 得到变换行、列矩阵
    trans_X, Z = pca_2d_2d(channels[c])
    # 图片降维
    p = np.dot(Z.T, np.dot(channels[c][150], trans_X))
    # 图片重构
```

```
    res = np.dot(Z, np.dot(p, trans_X.T))
```

- Jacobi 算法计算特征值与特征向量

```
# 计算特征向量与特征值
def jacobi(matrix, dim, precision, iteration_max):
    # 将特征向量初始化为单位阵
    eigenvectors = np.eye(dim)
    iter_count = 0
    while 1:
        db_max = matrix[0][1]
        n_row = 0
        n_col = 1
        for i in range(dim):
            for j in range(dim):
                d = fabs(matrix[i][j])
                if i != j and d > db_max:
                    db_max = d
                    n_row = i
                    n_col = j
        if db_max < precision:
            break
        if iter_count > iteration_max:
            break
        iter_count = iter_count + 1
        s_ii = matrix[n_row][n_row]
        s_jj = matrix[n_col][n_col]
        s_ij = matrix[n_row][n_col]
        theta = 0.5*atan2((-2) * s_ij, (s_jj - s_ii))
        sin_theta = sin(theta)
        cos_theta = cos(theta)
        sin_2theta = sin(2*theta)
        cos_2theta = cos(2*theta)
        matrix[n_row][n_row] = s_ii*cos_theta*cos_theta + s_jj * sin_theta * sin_theta + 2*s_ij*cos_theta*sin_theta
        matrix[n_col][n_col] = s_ii*sin_theta*sin_theta + s_jj * cos_theta * cos_theta - 2*s_ij*cos_theta*sin_theta
        matrix[n_row][n_col] = 0.5*(s_jj - s_ii)*sin_2theta + s_ij*cos_2theta
        matrix[n_col][n_row] = matrix[n_row][n_col]
        for i in range(dim):
            if i != n_row and i != n_col:
                db_max = matrix[i][n_row]
```

```
                matrix[i][n_row] = matrix[i][n_col] * sin_theta + db_max*cos_theta
                matrix[i][n_col] = matrix[i][n_col] * cos_theta - db_max*sin_theta

        for j in range(dim):
            if j != n_col and j != n_row:
                db_max = matrix[n_row][j]
                matrix[n_row][j] = matrix[n_col][j] * sin_theta + db_max*cos_theta
                matrix[n_col][j] = matrix[n_col][j] * cos_theta - db_max*sin_theta

        # compute eigenvector
        for i in range(dim):
            db_max = eigenvectors[i][n_row]
            eigenvectors[i][n_row] = eigenvectors[i][n_col]*sin_theta + db_max*cos_theta
            eigenvectors[i][n_col] = eigenvectors[i][n_col]*cos_theta - db_max*sin_theta

    # 取 matrix 的对角元素，得到特征值
    eigenvalues = np.diagonal(matrix)
    for i in range(dim):
        if np.sum(eigenvectors[:, i]) < 0:
            eigenvectors[:, i] *= -1

    return matrix, eigenvectors, eigenvalues
```

- 根据阈值，计算 k 的大小

```
# 根据阈值的大小，计算 k 的大小
def choose_dim(eigenvalues, alpha):
    threshold = np.sum(eigenvalues)*alpha
    total = 0
    for i in range(len(eigenvalues)):
        total = total + eigenvalues[i]
        if total >= threshold:
            return i+1
    return len(eigenvalues)
```

- 将特征向量根据特征值由大到小进行重排

```
# 将特征值从大到小进行排序
```

```
# 根据特征值的由大到小将特征向量进行重排
# np.argsort 返回元素从小到大排列的索引值，添加负号，返回元素从大到小的排列
#
def sort_eigen(w, eig_value):
    sort_index = np.argsort(-eig_value)
    temp = eig_value.copy()
    for i in range(eig_value.shape[0]):
        eig_value[i] = temp[sort_index[i]]
    temp = w.copy()
    for i in range(w.shape[0]):
        w[i, :] = temp[sort_index[i], :]
    return w, eig_value
```