《数据科学与工程算法》项目报告

报告题目：图像压缩

姓　　名：郭夏辉

学　　号：10211900416

完成日期：2024.06.20

摘要 [中文]：

主成分分析（Principal Component Analysis, PCA）是一种用于数据约简和降维的技术。它常用于减少数据集的维度，同时保留对方差贡献最大的特征。通过保留重要的主成分并忽略不太重要的部分，PCA 能够实现数据维度的约减，揭示高维数据中的主要特征。主成分分析主要应用于高维数据的降维，为后续的机器学习任务和数据可视化做好准备。图像作为典型的高维数据，因此我们可以尝试使用 PCA 来压缩图像。

在本实验中，我应用主成分分析对 100 张同类型的图像进行了压缩实验。实验的整体逻辑是，使用 Python 编程语言，提取每张图像的 RGB 矩阵，并分别对这三个矩阵进行 PCA 压缩，最后将压缩后的三个矩阵合并回原始结构，得到压缩后的图像。实验结果显示，PCA 压缩图像的效果显著，我们可以清楚地看出压缩后的图像结构，而且压缩一张图像所需的时间是可以接受的。

最后，我分析了使用 PCA 压缩图像的重构误差、压缩时间和压缩率、空间节省、实际视觉效果，并尝试使用 Kernel PCA 进行进一步优化，并基于此进行了未来展望。

Abstract [English]

Principal Component Analysis (PCA) is a technique used for data reduction and dimensionality reduction. It is commonly employed to reduce the dimensionality of datasets while retaining the features that contribute the most to variance. By retaining important principal components and ignoring less significant ones, PCA can achieve dimensionality reduction and reveal the main features in high-dimensional data. PCA is primarily used for dimensionality reduction in high-dimensional data, preparing it for subsequent machine learning tasks and data visualization. Images, as typical high-dimensional data, can be compressed using PCA.

In this experiment, I applied PCA to compress 100 images of the same type. The overall logic of the experiment is to use the Python programming language to extract the RGB matrices of each image, and then perform PCA compression on these three matrices separately. Finally, the three compressed matrices are combined back into the original structure to obtain the compressed images. The experimental results show that PCA performs well in compressing images. We can clearly see the structure of the compressed images, and the time required to compress a single image is acceptable.

Finally, I analyzed the reconstruction error, compression time, compression rate, space saving, and actual visual effects of compressing images using PCA. I also attempted further optimization using Kernel PCA and conducted future prospects based on this.

一、 项目概述（阐明项目的科学价值与相关研究工作，描述项目主要内容）

主成分分析（Principal Component Analysis，PCA）是一种用于数据降维和约减的技术。它是一个线性变换，通过将数据投影到一个新的坐标系中，使得高维空间中的数据点分别在最大方差的方向上进行投影，这些方向依次被称为第一主成分、第二主成分等。PCA 常用于减少数据集的维度，同时保留对方差贡献最大的特征。通过保留重要的主成分，忽略次要的部分，PCA 实现了数据维度的约减，揭示了高维数据中的主要特征。

PCA 主要应用于高维数据的降维，降维后的数据可以更好地服务于后续的机器学习任务和数据可视化。图片作为一种典型的高维数据，可以尝试使用 PCA 进行压缩，从而有效减少其维度。

二、 问题定义（提供问题定义的语言描述与数学形式）

在本次实验中，会有三类图片，分别是飞机类图片（Airplane），农业类图片（Agricultural）和沙滩类图片（Beach）。每一类图片都有 100 张 256×256 的图片。我们的问题是选择一类图片，对其实现一个 PCA 算法，综合考虑图片恢复程度与空间节省程度，选择合适的主成分个数，对图片进行压缩。

三、 方法（问题解决步骤和实现细节）

3.1 协方差矩阵

主成分分析需要我们理解并使用协方差矩阵的概念。因此，我们需要先知道协方差矩阵的定义与内涵。协方差是一种衡量两维数据间关联程度的常用指标。对于均含有 n 个样本的数据集合 X 和 Y，它们的协方差为：

$$\operatorname{Cov}(X,Y)=\frac{\sum_{i=1}^{n}\left(X_i-\bar{X}\right)\left(Y_i-\bar{Y}\right)}{n-1}$$

假如有一个多维数据集，维度间两两可以分别计算协方差，这样就得到了多个协方差。对于一个 n 维的数据集，可以计算 n×n 个协方差。若采用一个 n×n 的矩阵来表示它们，该矩阵则称为协方差矩阵。定义协方差矩阵为 Σ，则协方差矩阵中的第 i 行和第 j 列的元素定义为：

$$\Sigma_{ij}=\operatorname{Cov}\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)$$

3.2PCA 算法的实现

我们输入一个数据矩阵 X，它由 m 个 n 维的样本组成。那么我们的主要步骤如下所示：

1. 样本点去中心化。我们令 $\bar{x}=\frac{1}{m}\sum_{j=1}^{m}x_j$ 为样本中心点，那么我们可以构造 $y_i=x_i-\bar{x}$.

2. 样本协方差矩阵的奇异值分解。实际上，由于协方差矩阵是对称矩阵，因此我们进行特征分解即可，即 $\frac{YY^T}{m-1}=U\Sigma U^T$.

矩阵 U 的列向量可以作为正交基向量。

3. 主成分选取。我们将奇异值按照从大到小顺序排列，即 $\lambda_1\geq\lambda_2\geq\ldots\geq\lambda_m$.

然后我们根据以下规则确定主成分的个数 k：

$$k=\arg\min_{l}\left\{\frac{\sum_{i=1}^{l}\lambda_i}{\frac{\mathrm{rank}(A)}{\sum_{i=1}\lambda_i}}\geq\alpha\right\}.$$

由前 k 个奇异值所对应的特征向量组成矩阵 $\boldsymbol{W}=(w_1,w_2,\cdots,w_k)$.

这里的 α 是一个阈值，限定主成分对原始数据的解释能力，通常 α 设定在 90%左右。

4. 数据点向新空间中投影。将每个样本点 x_i 投影到新空间中，得到新样本点 $z_i=\boldsymbol{W}^T x_i$，其中 z_i 和 x_i 分别是 k 维和 n 维的，由于 $k<n$，从而实现了数据降维。

5. 降维输出。降维后的样本集为 $D'=(z_1,z_2,\cdots,z_m)$.

3.3 实验环境

本次实验使用了 PyCharm 2024.1 作为集成开发环境，Python 3.10 作为编程语言实现主成分分析压缩图片。

3.4 引入必要模块与常数

我们将选择农业类（Agricultural）图片来进行主成分分析的图片压缩。先引入必要的模块，定义必要的常数。

```
from PIL import Image
import numpy as np
import time
DIMENSION = 256
PIC_NUM = 100
```

PIL 模块是一个 Python 图像处理工具包，它可以为我们提供图像处理的方法。numpy 模块提供了一种更高效地处理列表，向量，矩阵的方式。

time 模块可以让我们获取当前的时间。基于这个功能，我们可以计算一段代码的运行时间，用以判断我们的代码的运行时间。

DIMENSION 是一个常数，它代表了我们的图片的维度。由于我们的图片规模为 256×256,故 DIMENSION 的数值为 256.

PIC_NUM 是一个常数，它代表了我们图片的张数。我们有 100 张同类型图片，故 PIC_NUM 的数值为 100.

3.5 获取图片的 RGB 矩阵

本质来说，计算机存储彩色图片的方式实际上是存储了三个同规模的矩阵。

计算机中的一张图片由三个矩阵构成，每一个矩阵都代表了红色、绿色、蓝色中的其中一种，矩阵中的每一个元素都代表了图片的一个像素点，某个色彩对应的矩阵中元素的数值代表了该色彩的深浅程度。矩阵中的每个元素的数值大小范围均在 0 到 255 之间。我们称呼一个图片的三个矩阵分别为 RGB 矩阵。

我们对图片进行压缩，正是对图片的 RGB 矩阵进行压缩。我使用如下代码，获取图片的 RGB 矩阵。最终，RED_init,GREEN_init,BLUE_init 会分别存储每张图片的 RGB 矩阵：

```
start_time = time.time()
RED_init = []
GREEN_init = []
BLUE_init = []
for i in range(PIC_NUM):
    if i < 10:
        i = str(i)
        img_fn = "Images/agricultural/agricultural0" + i + ".tif"
    else:
        i = str(i)
        img_fn = "Images/agricultural/agricultural" + i + ".tif"

    img = Image.open(img_fn)
    rgb_img = img.convert("RGB")
    im = np.array(rgb_img)
    Red = im[:,:,0]
    Green = im[:,:,1]
    Blue = im[:,:,2]
    RED_init.append(Red)
    GREEN_init.append(Green)
    BLUE_init.append(Blue)

# 转化为 numpy 列表
RED_init = np.array(RED_init) # 存储每张图片的 R 矩阵
GREEN_init = np.array(GREEN_init) # 存储每张图片的 G 矩阵
BLUE_init = np.array(BLUE_init) # 存储每张图片的 B 矩阵
```

3.6 样本去中心化并计算协方差矩阵

我将分别对 RGB 矩阵进行主成分分析的压缩。因此，我需要对每一个矩阵的 R、G、B 矩阵均进行去中心化与协方差矩阵的计算。

我使用如下的代码编写了一个函数，使得我可以通用地对样本进行去中心化操作，且可以计算协方差矩阵，它也将在之后所有图片压缩中被应用到：

```
def Cov_calculation(Red_cal, Green_cal, Blue_cal):
    average_red = np.mean(Red_cal, axis=0)
    average_green = np.mean(Green_cal, axis=0)
    average_blue = np.mean(Blue_cal, axis=0)
    Red_Sample = Red_cal - average_red
    Green_Sample = Green_cal - average_green
    Blue_Sample = Blue_cal - average_blue
    Cov_Red = np.matmul(Red_Sample, Red_Sample.T) / (DIMENSION - 1)
    Cov_Green = np.matmul(Green_Sample, Green_Sample.T) / (DIMENSION -
1)
```

```
    Cov_Blue = np.matmul(Blue_Sample, Blue_Sample.T) / (DIMENSION - 1)
    return average_red, average_green, average_blue, Red_Sample,
Green_Sample, Blue_Sample, Cov_Red, Cov_Green, Cov_Blue
```

3.7 协方差矩阵求特征值与特征向量

我们将对协方差矩阵求特征值与特征向量。虽然我在一开始考虑过使用课本中提到的幂法与反幂法，但由于我们需要求的是协方差矩阵的所有特征值而不是单独一个特征值，故幂法与反幂法无法达成我的目的。即使我们平移后使用反幂法，也必须对矩阵的特征值进行多次的估计，这样的做法并不效率。

这里我使用了 Peter J. Olver 在论文 Orthogonal Bases and the QR Algorithm 中提及的利用 QR 分解求特征值与特征向量的方法。如果一个对称矩阵可以被 QR 分解，那么我们可以对该矩阵进行 QR 分解，然后将分解得到的 Q 与 R 交换乘积的顺序，即计算 RQ 的乘积，之后我们再将由 RQ 的乘积得到的矩阵进行 QR 分解，重复上述步骤，如此循环直至达到循环的终止条件，并最终得到新的矩阵 A。在这个过程中，我们将每次分解得到的 Q 矩阵作乘法运算，在循环结束时得到一个 S 矩阵。利用该方法得到的矩阵 A 的对角线元素即为原矩阵的特征值，矩阵 S 的每一列即为每个特征值对应的特征向量。

```
def eigenvalue_func(cov):
    A_pre = np.zeros((DIMENSION, DIMENSION), dtype=float)
    A_post = cov
    S = np.eye(DIMENSION, dtype=float)
    while np.linalg.norm(np.diag(A_post) - np.diag(A_pre)) >= 0.00001:
        A_pre = A_post
        Q, R = np.linalg.qr(A_post)
        S = np.matmul(S, Q)
        A_post = np.matmul(R, Q)
    return np.diag(A_post), S
```

3.8 选取 k 个主成分

我按照课本中说明的，取 α 为 0.9

```
def num_com(eigen):
    k = 0 # 主成分的个数
    k_eigen = 0
    eigen_sum = np.sum(eigen)
    for i in eigen:
        k_eigen = k_eigen + i
        k = k + 1
        if k_eigen / eigen_sum >= 0.9:
            break
    return k
```

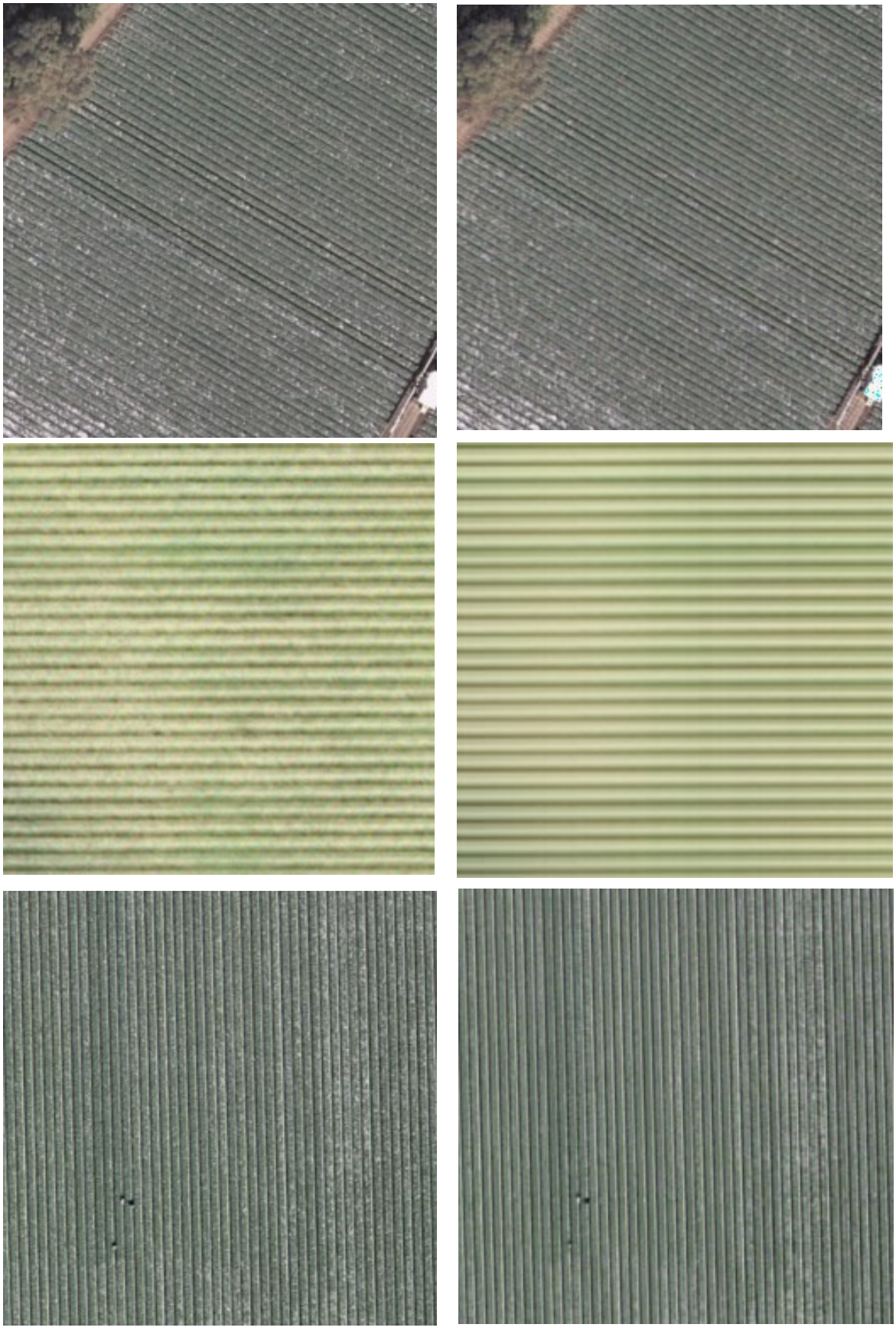
3.9 对图片进行压缩

在上述内容基础上，我可以直接对 100 张图片进行压缩操作。我的具体操作是，将一张图片的 R 矩阵，G 矩阵和 B 矩阵分别进行主成分分析的压缩，最后再合成回一张图片，从而达到一张图片的主成分分析压缩的目的。这样的操作只需要执行 100 次，即可将 100 张图片的压缩完成。

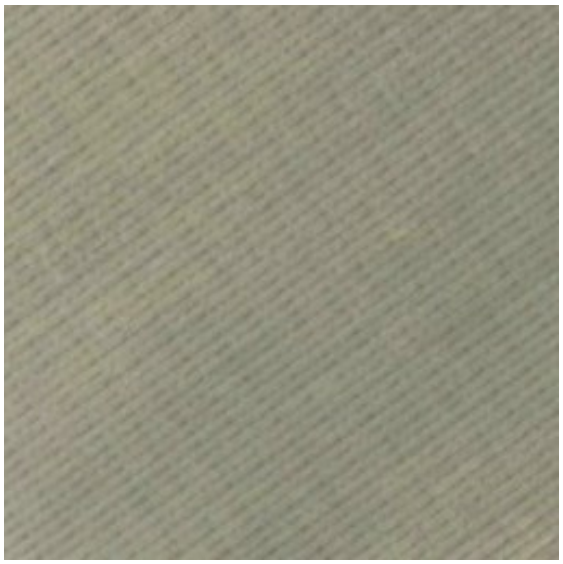
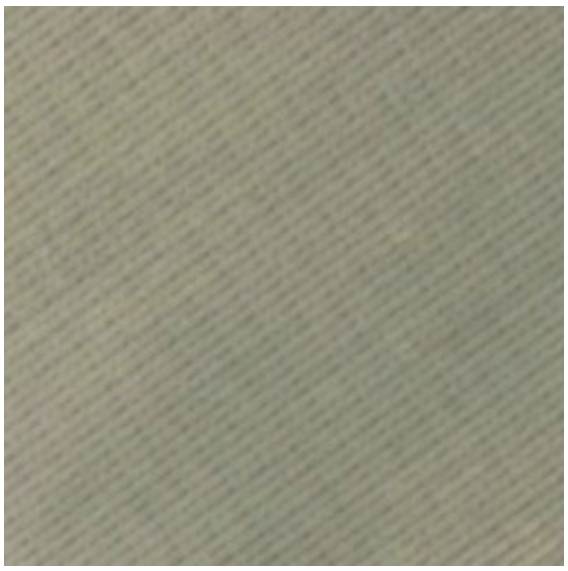
```
for i in range(PIC_NUM):
    Red_cal = RED_init[i]
    Green_cal = GREEN_init[i]
    Blue_cal = BLUE_init[i]
    average_red, average_green, average_blue, Red_Sample, Green_Sample,
Blue_Sample, Cov_Red, Cov_Green, Cov_Blue = Cov_calculation(
        Red_cal, Green_cal, Blue_cal)
    Red_eigenvalue, Red_eigenvector = eigenvalue_func(Cov_Red)
    Green_eigenvalue, Green_eigenvector = eigenvalue_func(Cov_Green)
    Blue_eigenvalue, Blue_eigenvector = eigenvalue_func(Cov_Blue)
    k_Red = num_com(Red_eigenvalue)
    k_Green = num_com(Green_eigenvalue)
    k_Blue = num_com(Blue_eigenvalue)
    Slice_Red = slice(0, k_Red, 1)
    Slice_Green = slice(0, k_Green, 1)
    Slice_Blue = slice(0, k_Blue, 1)
    S_Red = Red_eigenvector.T[Slice_Red]
    S_Green = Green_eigenvector.T[Slice_Green]
    S_Blue = Blue_eigenvector.T[Slice_Blue]
    W_Red = S_Red.T
    W_Green = S_Green.T
    W_Blue = S_Blue.T
    Red_result = np.matmul(W_Red.T, Red_Sample)
    Red_result = np.matmul(W_Red, Red_result)
    Red_result = Red_result + average_red
    Green_result = np.matmul(W_Green.T, Green_Sample)
    Green_result = np.matmul(W_Green, Green_result)
    Green_result = Green_result + average_green
    Blue_result = np.matmul(W_Blue.T, Blue_Sample)
    Blue_result = np.matmul(W_Blue, Blue_result)
    Blue_result = Blue_result + average_blue
    im1 = np.hstack((Red_result, Green_result, Blue_result))

    im3_channels = np.hsplit(im1, 3)
    im4 = np.zeros_like(im)
    for j in range(3):
        im4[:, :, j] = im3_channels[j]

    if i < 10:
        k = str(i)
        Image.fromarray(im4).save("Result/agricultural/agricultural0" +
k + ".tif")
    else:
        k = str(i)
```

```
        Image.fromarray(im4).save("Result/agricultural/agricultural" +
k + ".tif")
```

我以 agricultural00.tif 到 10,20,30,40,50 为例，展示 PCA 压缩图片的效果。（左边是原图，右边是压缩后的图片）可以观察到，压缩后的图片仍然与原图有着非常相似的整体结构，仅在细节上有所缺失。说明我的压缩是比较成功的。





3.10 使用 Kernel Principal Component Analysis 进行图片压缩

在由 Bernhard Schölkopf, Alexander Smola 与 Klaus-Robert Müller 编写的论文 Kernel Principal Component Analysis 中，他们详细介绍了核主成分分析的方法（Kernel PCA）。经过论文阅读、查阅资料与同学交流后，在这里我将试图复现这个算法。我采用高斯核函数作为核函数，然后进行 Kernel PCA 的图片压缩。

3.10.1 构造 100*196608 的矩阵，获得总体样本

```
pictures = np.hstack((RED_init[0].flatten(), GREEN_init[0].flatten(),
BLUE_init[0].flatten()))
for i in range(1, PIC_NUM):
    pic_data = np.hstack((RED_init[i].flatten(),
GREEN_init[i].flatten(), BLUE_init[i].flatten()))
    pictures = np.row_stack((pictures, pic_data))
picture_mean = np.mean(pictures, axis = 0)
```

3.10.2 构造高斯核函数

```
def kernel(x, y, sigma):
    norm = np.dot((x - y).T, (x - y))
    result = np.exp(-(norm / 2 / sigma ** 2)) # 高斯核函数
    return result
```

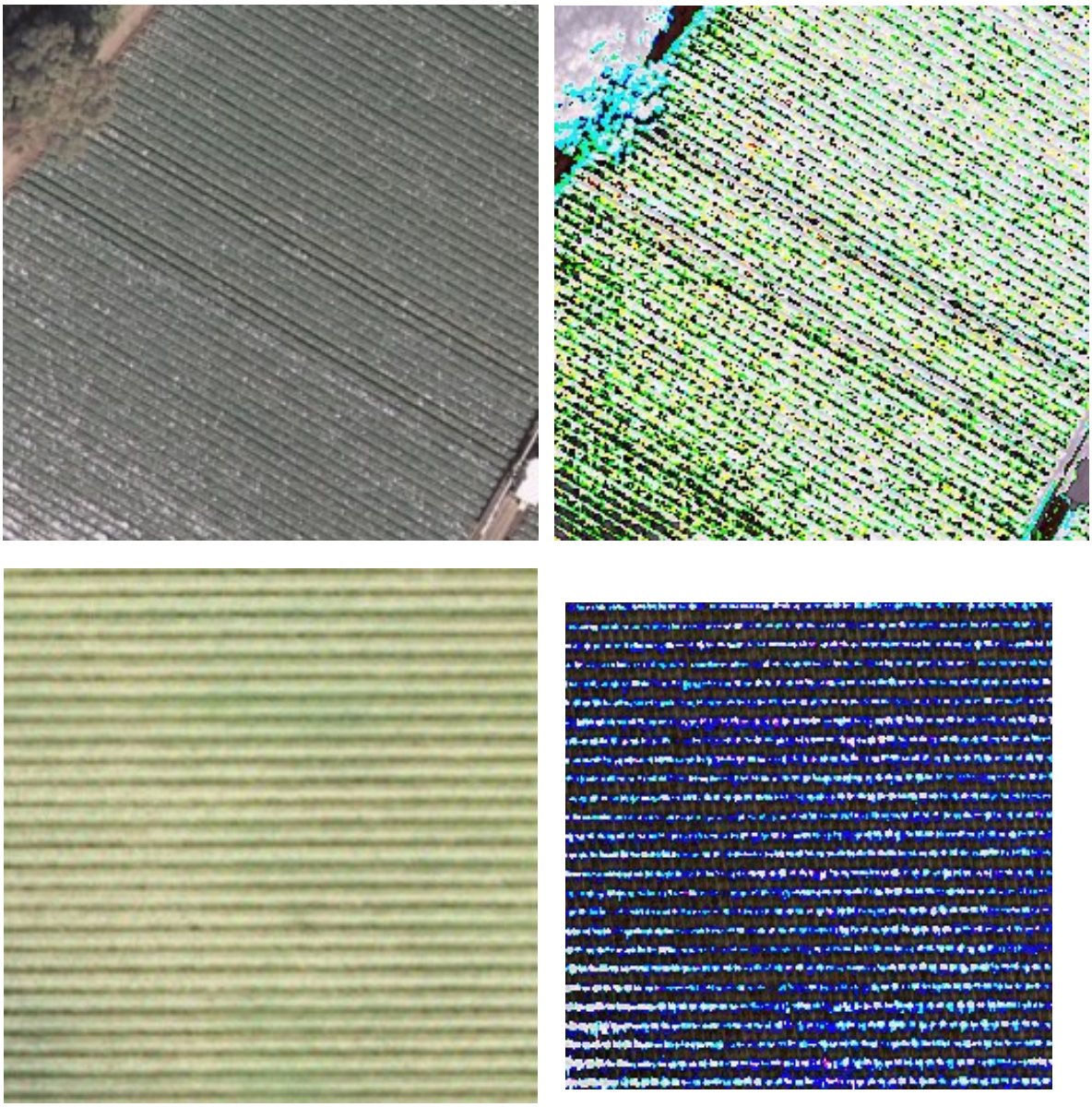
3.10.3 实际的压缩过程

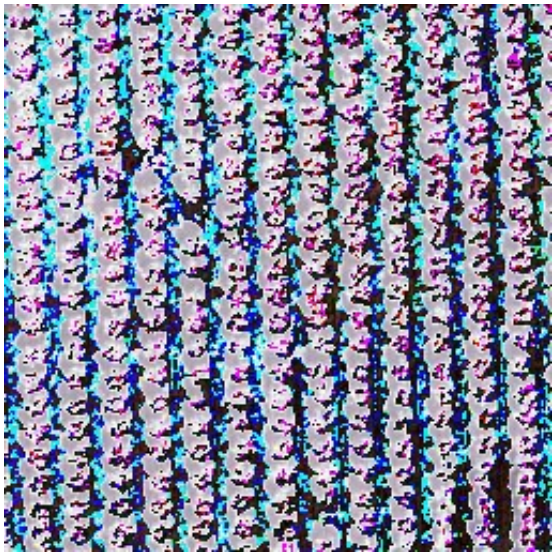
```
KernelMatrix = np.zeros((PIC_NUM, PIC_NUM))
for i in range(PIC_NUM):
    for j in range(PIC_NUM):
        KernelMatrix[i][j] = kernel(pictures[i], pictures[j], SIGMA)

I = np.ones((PIC_NUM, PIC_NUM)) / PIC_NUM
Normalized_K = KernelMatrix - I @ KernelMatrix - KernelMatrix @ I + I @
KernelMatrix @ I

eigenvalue, eigenvector = eigenvalue_func(Normalized_K)
Normalized_eigenvector = np.zeros((PIC_NUM, PIC_NUM))
for i in range(PIC_NUM):
    if eigenvalue[i] < 0:
        break

    norm = np.sqrt(np.dot(eigenvector[:, i].T, eigenvector[:, i]))
    Normalized_eigenvector[:, i] = eigenvector[:, i] / norm /
np.sqrt(eigenvalue[i])

k = 100
project_vector = Normalized_K @ Normalized_eigenvector[:, :k]
new_axis = np.matmul(project_vector.T, pictures)

# reconstruct the image data with k PCs
new_pictures = np.matmul(project_vector[:, :k], new_axis[:k, :])
```

```
im3 = np.matmul(project_vector[:, :k], new_axis[:k, :])[0]

im3 = im3.astype('uint8')

# reconstruct the three (R,G,B) channels
im3_channels = np.hsplit(im3, 3)
for i in range(3):
    im3_channels[i] = im3_channels[i].reshape((256, 256))
im4 = np.zeros((256, 256, 3))
for i in range(3):
    im4[:,:,i] = im3_channels[i]
new_image = Image.fromarray(np.uint8(im4*255*255))
new_image.save("./test1.tif")
end_time = time.time()
```

3.10.4 实际的效果

遗憾的是，使用该代码压缩得到的图像与原图像有很大的色彩差异。不过，我们仍然能看出图片的大致轮廓。该段代码可能在经过一定的优化后能够有更好的表现。以 agricultural00.tif 到 10,21 为例，实际效果如下所示：



四、 实验结果（验证提出方法的有效性和高效性）

4.1 单张图片 PCA 压缩性能分析

我们先以一张具体的图像 agricultural00.tif 为例进行分析，考量其重构误差、压缩时间与压缩率。我们可以在代码编写的过程中认为，将压缩后的矩阵与原先的矩阵作差，差值矩阵的 F 范数即为一张图片的重构误差。我们对所有的图片都添加如下代码，即可分析其重构误差。

```
Red_Error = Red_result - Red_cal
Green_Error = Green_result - Green_cal
Blue_Error = Blue_result - Blue_cal

Error = np.linalg.norm(Red_Error, ord="fro") +
np.linalg.norm(Green_Error, ord="fro") + np.linalg.norm(Blue_Error,
ord="fro")
print(Error)
```

对 agricultural00.tif 运行该代码，可得其重构误差为 5338.124808. 我们再在代码中添加 start_time = time.time()，end_time = time.time()，计算图片压缩所消耗的时间。对 agricultural00.tif 运行该代码，可得其压缩时间为 113.825 秒。由于我们在选定主成分个数时，要求 α 为 0.9，因为我们可以认为压缩率为 10%，压缩比例 90%，但这个是我们理论上认为的情况，具体情况还要以实际为准。实际来看，原始图片大小为 209,728 字节，压缩之后为 196,748 字节，空间节省 12980 字节，压缩率 6.18%，压缩比例 93.82%，是正常的。

4.2 所有图片 PCA 压缩性能分析

通过将 100 张图片全部进行压缩，我们可以得到所有图片的压缩性能。经过运行，我们发现压缩 100 张图片消耗了 6680.42 秒的时间。这意味着压缩一张图片平均需要 66.8 秒的时间，这个时间是可以接受的。

利用该方法压缩 100 张图片得到的重构误差为 447548.7956，平均每张图片的重构误差为 4475.48，也在可接受的范围内。

由于我们在选定主成分个数时，要求 α 为 0.9，因为我们可以认为压缩率

为 0.1，压缩比例 90%。实际来看，原始的 100 张图片总占大小为 20,828,780 字节，压缩之后为 19,674,800 字节，空间节省 1,153,980 字节，压缩率 5.54%，压缩比例 94.46%，是正常的。

4.3 Kernel PCA 压缩单张图片性能分析

通过对图片 agricultural00.tif 进行压缩，我们可以大致分析 Kernel PCA 的压缩性能。经过运行，我们发现压缩该图片消耗了 4.59027 秒的时间，这个时间是较快的。该图片的重构误差为 96528.653。我压缩得到的图片大小为 193,748 字节，原图大小为 209,728 字节，空间节省 15980 字节，因此我们可以认为其压缩率为 7.62%，压缩比例 92.38%.

我本次实验实现的 Kernel PCA 可能存在一些尚未优化的问题，因而其重构误差的数值是非常大的。Kernel PCA 所消耗的时间比传统 PCA 要短，但是我们透过三张图片来看压缩后的图像与原图像即使有着相似的轮廓，但有太大的色彩差异，Kernel PCA 的表现要差于传统 PCA 。

五、 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）

在本次实验中，我使用了主成分分析的方法，对 100 张农业类(Agricultural)图片进行了压缩。利用该方法，我们让图片所占据的计算机存储空间变小了，且仍然能保留图片最主要的特征，并且压缩的时间也是尚可接受的。然而，如果我们需要压缩多张图片，则该方法消耗的时间可能并不能接受，如在本课题中需要压缩 100 张图片，最终运行的时间超过了一个小时。因此，我们可能需要寻找一些优化主成分分析的方法。

其中，我在求矩阵特征值和特征向量时创新地采用了 QR 分解的方法。该方法可以避免我们在使用传统的幂法或反幂法时进行的大量估计，且运行效率较高。这是一个创新的尝试，且十分有效。

另外，我还对进阶的 Kernel PCA 进行了尝试，并且发现它的图片压缩时间非常短，如果使用得当将会是一个非常有效率的压缩方式。然而，由于我们对其本质的理解有所欠缺，最终代码的实现并没有特别成功，但我们仍然保留住了图片大致的特征。对 Kernel PCA 的更进一步的理解，可以作为这个课题的一个研究方向。

六、 参考文献

[1] 高明，胡卉芪. 数据科学与工程算法基础 [M]. 北京：高等教育出版社，2021.
[2] Olver, P. J. (2008). Orthogonal bases and the QR algorithm. University of Minnesota.
[3] Schölkopf, B., Smola, A., & Müller, K. R. (2005, June). Kernel principal component analysis. In Artificial Neural Networks—ICANN'97: 7th International Conference Lausanne, Switzerland, October 8–10, 1997 Proceeedings (pp. 583-588). Berlin, Heidelberg: Springer Berlin Heidelberg.