广东工业大学

计算机学院

《计算机视觉》

实验报告



|  |  |
| --- | --- |
| 课程名称： | 计算机视觉 |
| 实验名称： | 图像处理：模糊、锐化、去噪、直方图、色彩空间转换 |
| 指导教师： | 赵靖亮 |
| 学生姓名： | 欧炜标 |
| 班级学号： | 3121005358 |
| 实验日期： | 2024/5/21 |
| 实验成绩： |  |

实验报告撰写要求

实验操作是教学过程中理论联系实际的重要环节，而实验报告的撰写又是知识系统化的吸收和升华过程，因此，实验报告应该体现完整性、规范性、正确性、有效性。现将实验报告撰写的有关内容说明如下：

1、 实验报告模板为电子版。

2、 下载统一的实验报告模板，学生自行完成撰写和打印。报告的首页包含本次实验的一般信息：

*  组 号：例如：2-5 表示第二班第5组。
*  实验日期：例如：05-10-06 表示本次实验日期。(年-月-日)……
*  实验编号：例如：No.1 表示第一个实验。
*  实验时间：例如：2学时 表示本次实验所用的时间。

实验报告正文部分，从五个方面（目的、内容、步骤等）反映本次实验的要点、要求以及完成过程等情况。模板已为实验报告正文设定统一格式，学生只需在相应项内填充即可。（第一部分已经填好）。

3、 实验报告正文部分具体要求如下：

一、实验目的

本次实验所涉及并要求掌握的知识点。

二、实验环境

实验所使用的设备名称及规格，网络管理工具简介、版本等。

三、实验内容与实验要求

实验内容、原理分析及具体实验要求。

四、实验过程与分析

根据具体实验，记录、整理相应命令、运行结果等，包括截图和文字说明。

详细记录在实验过程中发生的故障和问题，并进行故障分析，说明故障排除的过程及方法。

五、实验结果总结

对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的改进意见。

注：本部分仅为展示报告内容，具体报告内容见下一页。

一、实验目的

基于python环境和OpenCV等软件包，实现图像模糊、锐化和去噪等图像处理操作，掌握图像直方图的生成、直方图均衡化的实现，掌握色彩空间转换的实现。

二、实验环境

设备名称 LAPTOP-BHF8HOH1

处理器 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

机带 RAM 16.0 GB (15.8 GB 可用)

设备 ID 8A9CD625-D5C1-40B0-AFDA-7C0219502457

产品 ID 00342-36199-79021-AAOEM

系统类型 64 位操作系统, 基于 x64 的处理器

实验环境：pycharm+opencv

三、实验内容与实验要求

实验内容：图像滤波（模糊、锐化、去噪）、图像直方图的生成与均衡化、色彩空间转换

实验要求：

2. 指导书的代码仅供参考，具体以压缩包里的 \*.py 文件为准。学生需要对代码进行改动，从而完成基础实验要求。

3. 思考题是拓展实验。需要在报告中列出解题思路、关键代码、执行结果等。

**空间域滤波器**

对代码的补充：

1. 补充滤波公式：

pixel = np.sum(img[i:i + 3, j:j + 3] \* filter)

1. 补全统计数据公式

pixel = np.median(img[i:i + 3, j:j + 3])

1. 补全main函数调用：

# 2. 将平滑后的图像行锐化高通滤波 查看结果

# 更换Filtering2D(img, filter) 的filter，实现锐化

# 需要锐化的图片是 test/3.jpg img = cv2.imread("test/3.jpg")

img = getGrayImg(img)

cv2.imshow('original image', img)

img\_padding = paddingWithNeighbor(img)

filtered\_img = Filtering2D(img\_padding, laplacianKernel)

cv2.imshow('sharpened image', filtered\_img)

cv2.imwrite("3\_sharpened.jpg", filtered\_img)

3. 利用均值、中值、最大值、最小值对椒盐、椒、盐噪声图像进行去噪 并 查看结果

# 需要调整denoisewithOrderStatisticsFilter(img)，来实现不同的功能

# 每种功能执行一次，输出一个结果，保存

# 均值、中值滤波处理的图像：test/2.jpg

# 最大值滤波：421.jpeg, 最小值滤波：419.jpeg

img = cv2.imread("test/2.jpg", cv2.IMREAD\_GRAYSCALE)

cv2.imshow('original image', img)

img\_padding = paddingWithZero(img)

filtered\_img = denoisewithOrderStatisticsFilter(img\_padding)

cv2.imshow('denoised image with median filter', filtered\_img)

cv2.imwrite("2\_denoised\_median.jpg", filtered\_img)

img = cv2.imread("test/419.jpeg", cv2.IMREAD\_GRAYSCALE)

cv2.imshow('original image', img)

img\_padding = paddingWithZero(img)

filtered\_img = denoisewithOrderStatisticsFilter(img\_padding)

cv2.imshow('denoised image with minimum filter', filtered\_img)

cv2.imwrite("419\_denoised\_min.jpg", filtered\_img)

img = cv2.imread("test/421.jpeg", cv2.IMREAD\_GRAYSCALE)

cv2.imshow('original image', img)

img\_padding = paddingWithZero(img)

filtered\_img = denoisewithOrderStatisticsFilter(img\_padding)

cv2.imshow('denoised image with maximum filter', filtered\_img)

cv2.imwrite("421\_denoised\_max.jpg", filtered\_img)

将denoisewithOrderStatisticsFilter修改如下：

def denoisewithOrderStatisticsFilter(img, method='median'):

filtered\_img = np.zeros((img.shape[0] - 2, img.shape[1] - 2), np.uint8)

for i in range(0, filtered\_img.shape[0]):

for j in range(0, filtered\_img.shape[1]):

region = img[i:i+3, j:j+3]

**if method == 'median':**

**pixel = np.median(region)**

**elif method == 'mean':**

**pixel = np.min(region)**

**elif method == 'min':**

**pixel = np.min(region)**

**elif method == 'max':**

**pixel = np.max(region)**

**filtered\_img[i][j] = pixel**

return filtered\_img

1. 运行结果：

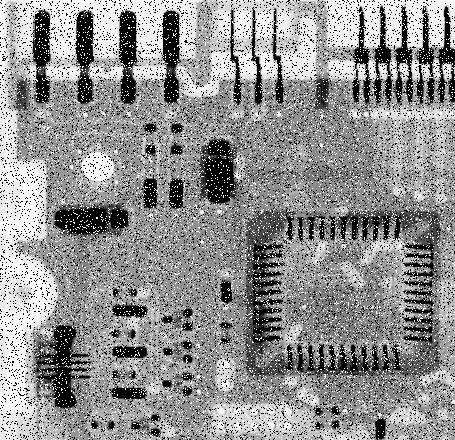
1.jpg 1\_smooth.jpg



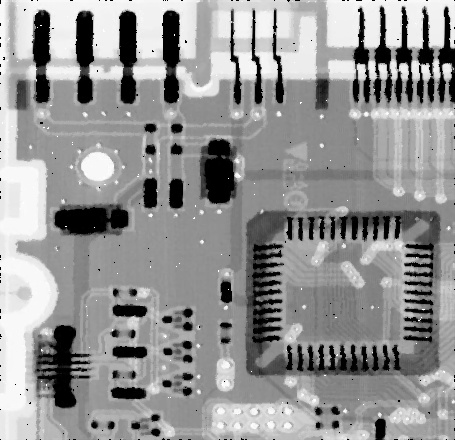
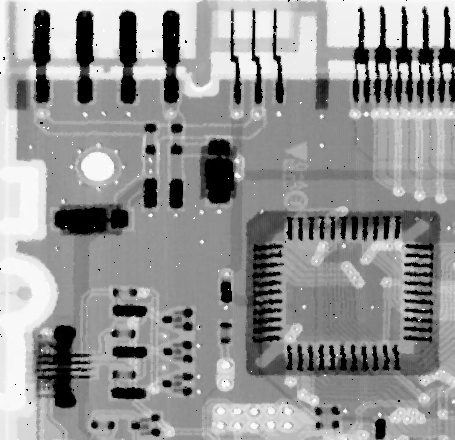
1\_enhenced.jpg

原始图片test3 锐化后



原始图片2.jpg

均值滤波去噪 中值滤波去噪

原始图片 最小滤波去噪

原始图片 最大滤波去噪

**直方图均衡化**

实验原始代码运行结果：

RuntimeWarning: overflow encountered in ubyte\_scalars 警告表明，在使用 np.uint8 类型进行像素值相减时发生了溢出，导致计算结果超出了无符号字节（0-255）的有效范围。这个问题可以通过在计算中使用不同的数据类型来解决

对getPSNR函数修改如下：

def getPSNR(ori\_img, en\_img):

MAX = 255.0

total = 0.0

for i in range(ori\_img.shape[0]):

for j in range(ori\_img.shape[1]):

total = total + (int(ori\_img[i][j]) - int(en\_img[i][j]))\*\*2

MSE = total / (ori\_img.shape[0] \* ori\_img.shape[1])

PSNR = 10 \* math.log10(MAX \* MAX / MSE)

return PSNR

对实验代码进行补充：

def image\_equalization(img, S):

img\_eq = np.zeros((img.shape[0], img.shape[1]), np.uint8)

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

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

# ###### 实现像素值的重映射 ##########

img\_eq[i][j] = S[img[i][j]]

# ############## 结束编程 #############

pass

return img\_eq

def get\_histogram(gray\_img):

# 利用hash table实现统计直方图

# 注意统计范围较大，使用int32类型numpy array

Pr = np.zeros(256, np.int32) # Hash Table

for i in range(gray\_img.shape[0]):

for j in range(gray\_img.shape[1]):

# ###### 1. 这里编程实现直方图统计 ##########

Pr[gray\_img[i][j]] += 1

# ############## 结束编程 #############

pass

# ###### 2. 归一化直方图，获得概率分布计 ##########

Pr = Pr / (gray\_img.shape[0] \* gray\_img.shape[1])

# ############## 结束编程 #############

pass

S = np.zeros(256, np.float32)

pre\_sum = 0 # 提示1

for i in range(256):

# ###### 3. 获得累积概率分布 ##########

pre\_sum += Pr[i]

S[i] = pre\_sum

# ############## 结束编程 #############

pass

S = S \* 255

S = np.round(S).astype(np.uint8)

Ps = np.zeros(256, np.float32)

for i in range(256):

# 提示线索

Ps[S[i]] = Ps[S[i]] + Pr[S[i]]

return S, Ps

运行结果如下：

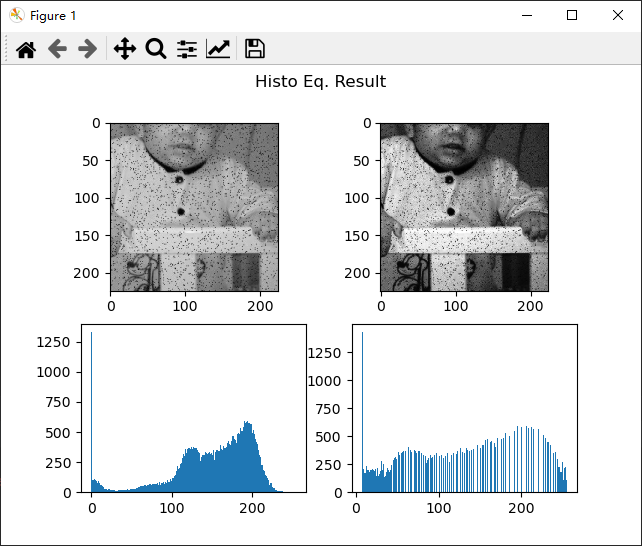
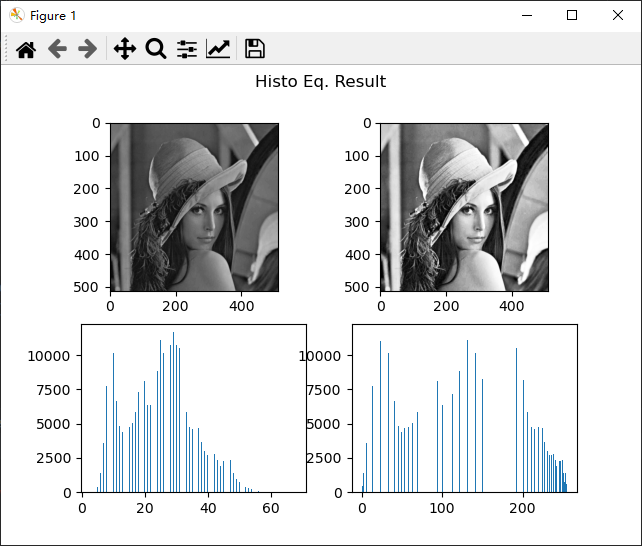


图1 Lena图片的直方图均衡化展示 图2 婴儿图片的直方图均衡化展示

由图1图2可以看到测试图片的直方图均衡化结果。

**色彩空间变换**

对实验代码进行补充，完整代码如下：

import numpy as np

import cv2

import math

import os

import matplotlib.pyplot as plt

def RGB2YUV\_enhance(img, lightness\_en=3.5):

temp\_YUV=np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

res\_RGB=np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

timg=img.astype(np.float32)

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

# 打印一下进度信息，知道处理到第几行了

if i % 100 == 0:

print("Processing line " + str(i))

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

##############################################################

# Note that, should be careful about the RGB or BGR order

# Hint: check the transformation matrix to convert RGB to YUV

##############################################################

## write your code here

Y=0.299 \* timg[i][j][2] + 0.587 \* timg[i][j][1] + 0.114 \* timg[i][j][0]

U=-0.169 \* timg[i][j][2] - 0.331 \* timg[i][j][1] + 0.5 \* timg[i][j][0] + 128

V=0.5 \* timg[i][j][2] - 0.419 \* timg[i][j][1] - 0.081 \* timg[i][j][0] + 128

## 1. save temp\_YUV for visualization

temp\_YUV[i][j][0]=Y

temp\_YUV[i][j][1]=U

temp\_YUV[i][j][2]=V

## 2. enhance Y and convert YUV back to the RGB

Y\_enhanced=np.clip(Y \* lightness\_en, 0, 255)

R=Y\_enhanced + 1.402 \* (V - 128)

G=Y\_enhanced - 0.344 \* (U - 128) - 0.714 \* (V - 128)

B=Y\_enhanced + 1.772 \* (U - 128)

## 3. store the enhanced RGB

res\_RGB[i][j][0]=np.clip(B, 0, 255)

res\_RGB[i][j][1]=np.clip(G, 0, 255)

res\_RGB[i][j][2]=np.clip(R, 0, 255)

#############################################################

# end of your code

#############################################################

# pass

#############################################################

# (Optional) consider more efficent way to implement such a conversion

#############################################################

return temp\_YUV, res\_RGB

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

img=cv2.imread("test/Lena.jpg")

imgyuv, res\_rgb=RGB2YUV\_enhance(img)

plt.figure()

plt.suptitle('RGB to YUV Result')

plt.subplot(221)

plt.imshow(imgyuv[:, :, 0], cmap='gray')

plt.subplot(222)

plt.imshow(imgyuv[:, :, 1], cmap='gray')

plt.subplot(223)

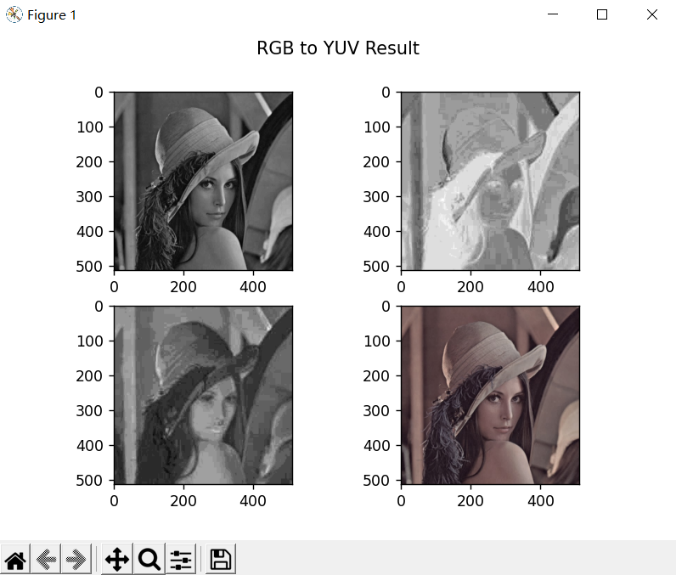
plt.imshow(imgyuv[:, :, 2], cmap='gray')

plt.subplot(224)

plt.imshow(cv2.cvtColor(res\_rgb, cv2.COLOR\_BGR2RGB))

plt.show()

运行结果：

原始图像 处理后的结果截图

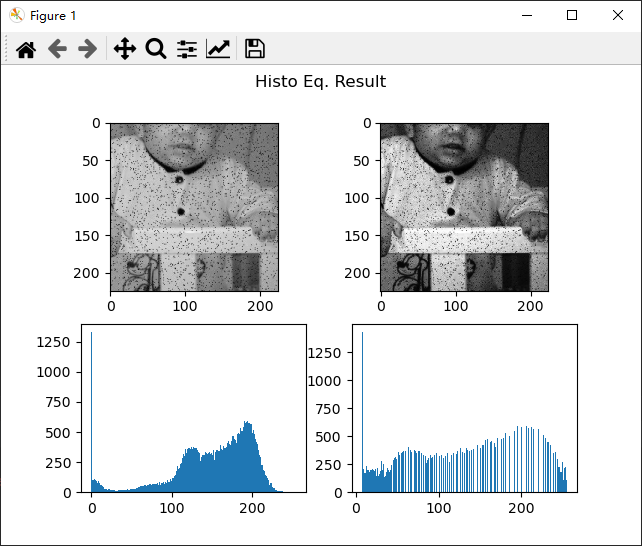
四、实验结果总结

*对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的****改进意见****。（可多页）*

1. 空间域滤波

实验通过应用平均核、高斯核和拉普拉斯核实现了图像的模糊、锐化和边缘增强。通过ce试图图片的展示可以看到，对提升图片质量有一定的效果。

1. 直方图均衡化



以婴儿为例，分析实验结果。

原始图像质量：原始图像展示了清晰的人脸特征和背景细节。

处理后的图像质量：处理后的图像在视觉效果上更加清晰，噪声减少，颜色分布更均匀。

直方图调整：直方图调整使得图像的亮度级别更加丰富，从而改善了图像的质量。

像素值分布：处理前的直方图显示像素值集中在较低的亮度级别，导致图像看起来较暗；处理后的直方图显示像素值分布在更广泛的范围内，使图像看起来更加明亮和清晰

1. 色彩空间变换

通过对lena测试图片的从RGB色彩空间到YUV和YIQ色彩空间的转换，调整亮度增强了图像的视觉效果。

1. 实验改进意见：

对于图像滤波，可以考虑实现更复杂的滤波器，如双边滤波，以更好地保持边缘信息的同时进行平滑。

在直方图均衡化部分，可以探索自适应直方图均衡化，以应对不同光照条件下的图像增强。

色彩空间转换可以进一步扩展到其他色彩空间，如HSV或HSL，以及它们在图像处理中的应用。

思考题

1. 测试其他3×3滤波器,比如检测边缘、一阶、二阶梯度算子。

**解题思路**：首先，从文件中读取灰度图像，然后定义常见的3x3滤波器矩阵，包括Sobel（一阶微分算子）、Roberts（边缘检测算子）和Laplacian（二阶微分算子）。这些滤波器用于检测图像中的边缘。接着使用OpenCV的filter2D函数对图像应用不同的滤波器，并生成相应的边缘检测结果图像。最后将原始图像和处理后的图像显示出来。

**关键代码分析：**

加载测试图像：（以1\_smooth.jpg作为测试用例，其他测试图像同理，只需要修改路径）：  
img = cv2.imread("test/1\_smooth.jpg", cv2.IMREAD\_GRAYSCALE)  
定义滤波器：

sobel\_x = np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]])

sobel\_y = np.array([[-1, -2, -1],

[0, 0, 0],

[1, 2, 1]])

roberts\_x = np.array([[1, 0],

[0, -1]])

roberts\_y = np.array([[0, 1],

[-1, 0]])

laplacian = np.array([[0, 1, 0],

[1, -4, 1],

[0, 1, 0]])

定义了Sobel、Roberts和Laplacian算子。这些算子用于计算图像中水平和垂直方向的梯度，以检测边缘。

完整代码如下：

import numpy as np

import cv2

# Sobel算子

sobel\_x = np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]])

sobel\_y = np.array([[-1, -2, -1],

[0, 0, 0],

[1, 2, 1]])

# Roberts算子

roberts\_x = np.array([[1, 0],

[0, -1]])

roberts\_y = np.array([[0, 1],

[-1, 0]])

# Laplacian算子

laplacian = np.array([[0, 1, 0],

[1, -4, 1],

[0, 1, 0]])

# 加载测试图片

img = cv2.imread("test/1\_smooth.jpg", cv2.IMREAD\_GRAYSCALE)

# 使用不同的滤波器进行边缘检测

sobel\_x\_edges = cv2.filter2D(img, -1, sobel\_x)

sobel\_y\_edges = cv2.filter2D(img, -1, sobel\_y)

roberts\_x\_edges = cv2.filter2D(img, -1, roberts\_x)

roberts\_y\_edges = cv2.filter2D(img, -1, roberts\_y)

laplacian\_edges = cv2.filter2D(img, -1, laplacian)

# 显示结果

cv2.imshow('Original Image', img)

cv2.imshow('Sobel X Edges', sobel\_x\_edges)

cv2.imshow('Sobel Y Edges', sobel\_y\_edges)

cv2.imshow('Roberts X Edges', roberts\_x\_edges)

cv2.imshow('Roberts Y Edges', roberts\_y\_edges)

cv2.imshow('Laplacian Edges', laplacian\_edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

**运行结果**

原图test1\_smooth.jpg：



边缘检测算子（Robert）

roberts\_x\_edges roberts\_y\_edges

一阶微分算子（Sobel）：

sobel\_x\_edges sobel\_y\_edges

二阶微分算子（laplacian）：



laplacian\_edges

1. 简述手工设计的滤波器与CNN的卷积核的关系？

不同点：

手工设计的滤波器用于传统机器学习，在运算过程中其中的数值不会发生改变，而卷积核是通过特定初始化方法如随机初始化得到。

在模型学习的过程中不断更新其中的值（参数），最后的值是不能人为确定的，而检测的作用随着网络越来越深，也越缺乏可解释性，而手工设计的滤波器在设计的时候就可以确定其用途（如边缘检测）

相同点：

滤波器和卷积核都是用于在图像处理中应用卷积操作的工具。

它们都是一个小的矩阵或核，用于在图像或特征图上滑动进行卷积运算。

两者都可以用于实现一些图像处理任务，如边缘检测、模糊、锐化等。

1. 测试其他色彩空间转换效果，如RGB -> YIQ or CMY？

以kena为测试图片进行测试，测试RGB->YIQ，RGB->CMY效果，完整代码如下：

import numpy as np

import cv2

import matplotlib.pyplot as plt

def RGB2YIQ\_enhance(img, lightness\_en=3.5):

temp\_YIQ = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

res\_RGB = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

timg = img.astype(np.float32)

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

# 打印一下进度信息，知道处理到第几行了

if i % 100 == 0:

print("Processing line " + str(i))

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

##############################################################

# Note that, should be careful about the RGB or BGR order

# Hint: check the transformation matrix to convert RGB to YIQ

##############################################################

Y = 0.299 \* timg[i][j][2] + 0.587 \* timg[i][j][1] + 0.114 \* timg[i][j][0]

I = 0.596 \* timg[i][j][2] - 0.275 \* timg[i][j][1] - 0.321 \* timg[i][j][0] + 128

Q = 0.212 \* timg[i][j][2] - 0.523 \* timg[i][j][1] + 0.311 \* timg[i][j][0] + 128

temp\_YIQ[i][j][0] = Y

temp\_YIQ[i][j][1] = I

temp\_YIQ[i][j][2] = Q

Y\_enhanced = np.clip(Y \* lightness\_en, 0, 255)

R = Y\_enhanced + 1.402 \* (Q - 128)

G = Y\_enhanced - 0.344 \* (I - 128) - 0.714 \* (Q - 128)

B = Y\_enhanced + 1.772 \* (I - 128)

res\_RGB[i][j][0] = np.clip(B, 0, 255)

res\_RGB[i][j][1] = np.clip(G, 0, 255)

res\_RGB[i][j][2] = np.clip(R, 0, 255)

return temp\_YIQ, res\_RGB

def RGB2CMY\_enhance(img, cyan\_en=1.5):

temp\_CMY = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

res\_RGB = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

timg = img.astype(np.float32)

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

if i % 100 == 0:

print("Processing line " + str(i))

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

# RGB to CMY

R = timg[i][j][2]

G = timg[i][j][1]

B = timg[i][j][0]

C = 255 - R

M = 255 - G

Y = 255 - B

# Save temp\_CMY for visualization

temp\_CMY[i][j][0] = C

temp\_CMY[i][j][1] = M

temp\_CMY[i][j][2] = Y

# Enhance the cyan channel

C\_enhanced = np.clip(C \* cyan\_en, 0, 255)

# CMY to RGB

R\_enhanced = 255 - C\_enhanced

G\_enhanced = 255 - M

B\_enhanced = 255 - Y

# Store the enhanced RGB

res\_RGB[i][j][0] = np.clip(B\_enhanced, 0, 255)

res\_RGB[i][j][1] = np.clip(G\_enhanced, 0, 255)

res\_RGB[i][j][2] = np.clip(R\_enhanced, 0, 255)

return temp\_CMY, res\_RGB

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

img = cv2.imread("test/Lena.jpg")

imgyiq, res\_rgb = RGB2YIQ\_enhance(img)

plt.figure()

plt.suptitle('RGB to YIQ Result')

plt.subplot(221)

plt.imshow(imgyiq[:, :, 0], cmap='gray')

plt.subplot(222)

plt.imshow(imgyiq[:, :, 1], cmap='gray')

plt.subplot(223)

plt.imshow(imgyiq[:, :, 2], cmap='gray')

plt.subplot(224)

plt.imshow(cv2.cvtColor(res\_rgb, cv2.COLOR\_BGR2RGB))

plt.show()

imgcmy, res\_rgb = RGB2CMY\_enhance(img)

plt.figure()

plt.suptitle('RGB to CMY Result')

plt.subplot(221)

plt.imshow(imgcmy[:, :, 0], cmap='gray')

plt.subplot(222)

plt.imshow(imgcmy[:, :, 1], cmap='gray')

plt.subplot(223)

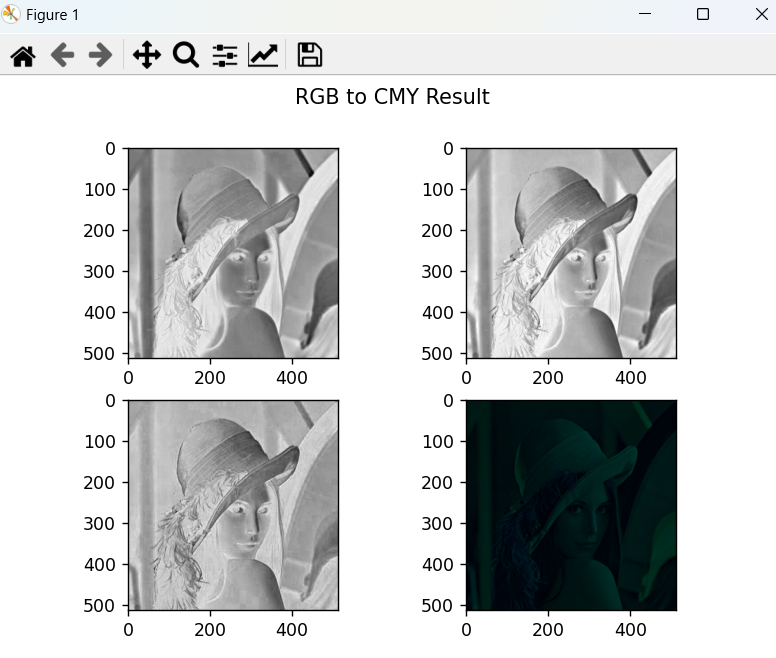
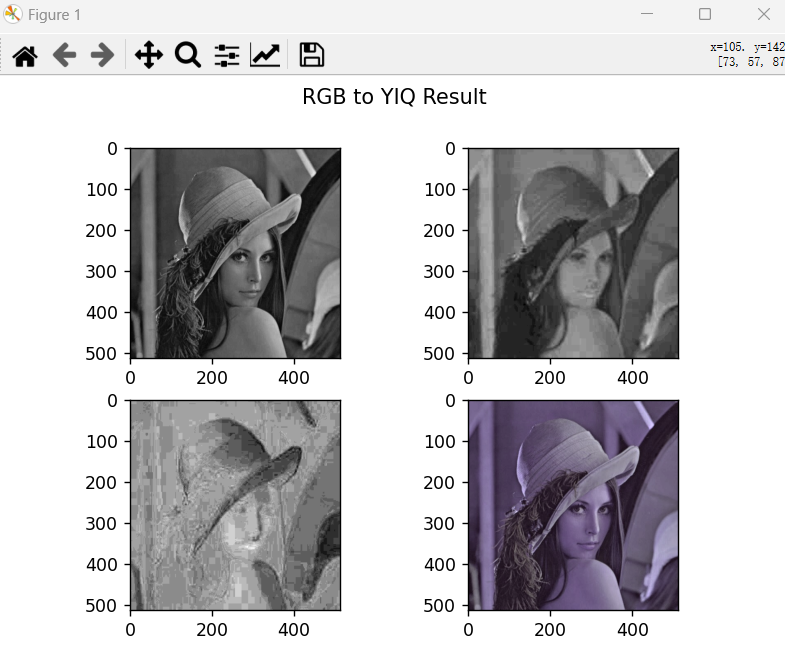
plt.imshow(imgcmy[:, :, 2], cmap='gray')

plt.subplot(224)

plt.imshow(cv2.cvtColor(res\_rgb, cv2.COLOR\_BGR2RGB))

plt.show()

测试结果：

RGB->CMY测试结果 RGB->YIQ测试结果

1. 彩色图像如何进行直方图均衡化？

思路：

颜色分离：将彩色图像分解为其RGB（红、绿、蓝）通道。

通道独立处理：分别对每个通道进行直方图统计和均衡化处理。

合并通道：将均衡化后的通道重新合并为彩色图像。

代码实现：  
def equalize\_rgb(image):

# 分离RGB通道

r, g, b = cv2.split(image)

# 对每个通道应用直方图均衡化

r\_eq = cv2.equalizeHist(r)

g\_eq = cv2.equalizeHist(g)

b\_eq = cv2.equalizeHist(b)

# 重新组合通道

image\_eq = cv2.merge([r\_eq, g\_eq, b\_eq])

return image\_eq