****

**数字图像处理课内实习报告**

班级序号： 111211

学生姓名： 李安南

任课教师： 周林

**中国地质大学计算机学院**

**2024年4月**

## 题目一：

设计方案对图像“shixi1.jpg”进行处理，以提升图像质量。



**原理：**

**去噪：**

使用BM3D算法进行去噪，这是一种基于非局部均值的先进去噪技术。该算法首先将图像分割成重叠的块，并在图像中搜索与当前块相似的其他块。这些相似的块被组织成一个三维数据数组，随后在变换域中进行协同过滤，有效去除噪声的同时保留图像的结构细节。

**对比度增强：**

CLAHE是一种改良的直方图均衡化方法，专门用于改善图像的局部对比度而不过度放大噪声。与传统的直方图均衡化不同，CLAHE对每个小区域独立应用均衡化，并通过限制对比度增强来避免在均衡化过程中产生噪声放大。

**锐化：**

Unsharp Masking通过增强图像的边缘来提高其视觉锐度。该技术涉及三个主要步骤：首先，对原始图像进行高斯模糊处理；然后，将模糊图像从原图中减去，得到边缘强调的差分图像；最后，将这个差分图像加回到原图中，通过这种方式强调细节和边缘，使图像看起来更清晰。

**色彩增强：**

在HSV颜色空间中进行色彩增强，通过调整饱和度分量来增强图像的色彩表现。HSV颜色空间将图像颜色分解为色调（H）、饱和度（S）和亮度（V），其中饱和度调整能够直接影响颜色的纯度和强度。通过提高饱和度值，可以使颜色看起来更加鲜明和生动，这种方法在广告和媒体产业中尤为常用。

**形态学处理：**

使用形态学操作中的膨胀和腐蚀来计算图像的形态学梯度。形态学膨胀可以扩大图像中的亮区域，而腐蚀则缩小亮区域。通过这两个操作的组合，可以得到图像边缘的粗略表示，称为形态学梯度。

**图像融合：**

将形态学处理后的图像与色彩增强后的图像通过加权平均的方式进行融合，整合形态学梯度带来的边缘信息和色彩增强带来的色彩信息，以此显著提升图像的视觉效果。

**伪代码：**

读取原始图像并将图像颜色空间从 BGR 转换到 RGB

将图像数据类型转换为 float32 并归一化至 0-1 范围

应用 BM3D 算法去噪

创建并应用 CLAHE 对象调整图像对比度

将图像从 RGB 转换到 LAB 颜色空间

分离 L, A, B 通道

对 L 通道应用 CLAHE

合并调整后的 L 通道与原始 A, B 通道

将图像从 LAB 转换回 RGB

应用高斯模糊进行 Unsharp Masking 锐化处理

使用高斯模糊对图像进行模糊

将锐化图像和模糊图像进行加权融合以增强细节

提高图像饱和度以增强色彩

将图像从 RGB 转换到 HSV

增加 S 通道的值以提高饱和度

将图像从 HSV 转换回 RGB

使用形态学处理突出图像边缘

将图像转换为灰度图

应用形态学膨胀和腐蚀操作

计算形态学梯度

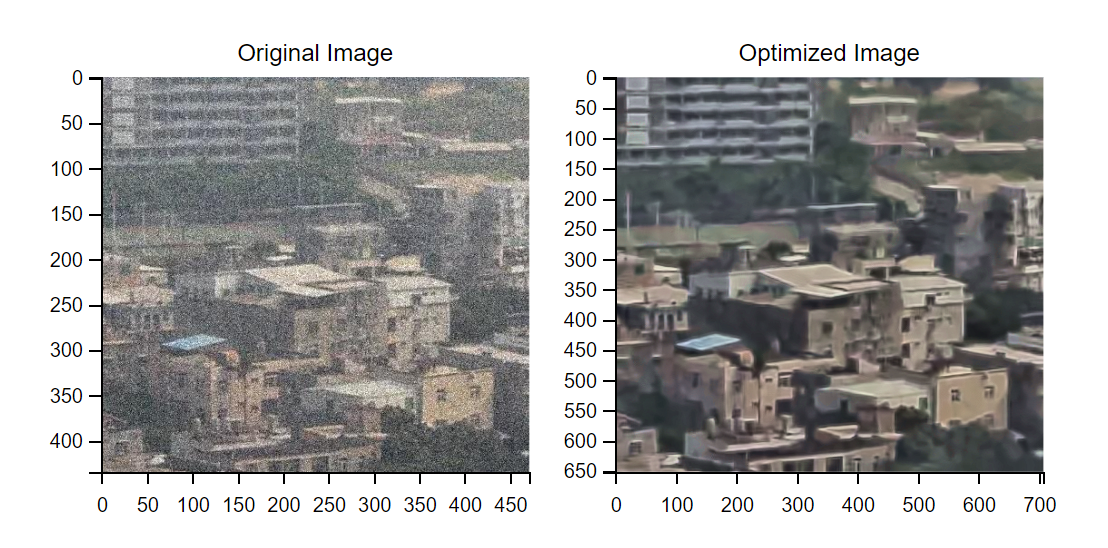
将梯度图与原图进行加权融合

使用双三次插值方法对图像进行超分辨率重建

**源代码：**  
import cv2  
from skimage import morphology  
from skimage.util import img\_as\_ubyte  
import numpy as np  
import matplotlib.pyplot as plt  
from PIL import Image  
import bm3d  
  
# 读取图像  
original\_img\_path = 'first.jpg'  
img = cv2.imread(original\_img\_path)  
original\_img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
  
# 将图像数据类型转换为 float32，以便使用 BM3D  
img\_float = original\_img\_rgb.astype(np.float32) / 255  
  
# 使用 BM3D 去噪算法  
bm3d\_denoised\_img = bm3d.bm3d(img\_float, sigma\_psd=30/255)  
  
# 将去噪后的图像转换回 uint8 类型  
bm3d\_denoised\_img\_uint8 = img\_as\_ubyte(bm3d\_denoised\_img)  
  
# 对比度调整 - 使用CLAHE  
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))  
lab = cv2.cvtColor(bm3d\_denoised\_img\_uint8, cv2.COLOR\_RGB2LAB)  
l, a, b = cv2.split(lab)  
l\_clahe = clahe.apply(l)  
enhanced\_lab = cv2.merge((l\_clahe, a, b))  
contrast\_enhanced\_img = cv2.cvtColor(enhanced\_lab, cv2.COLOR\_LAB2RGB)  
  
# 锐化 - 使用高斯模糊和加权方法进行Unsharp Masking  
gaussian\_blurred = cv2.GaussianBlur(contrast\_enhanced\_img, (3, 3), 0)  
unsharp\_image = cv2.addWeighted(contrast\_enhanced\_img, 1.5, gaussian\_blurred, -0.5, 0)  
  
# 色彩增强 - 提高饱和度  
hsv = cv2.cvtColor(unsharp\_image, cv2.COLOR\_RGB2HSV)  
h, s, v = cv2.split(hsv)  
s = cv2.add(s, 15)  
enhanced\_hsv = cv2.merge([h, s, v])  
color\_enhanced\_img = cv2.cvtColor(enhanced\_hsv, cv2.COLOR\_HSV2RGB)  
  
# 形态学处理 - 使用形态学梯度突出边缘  
gray = cv2.cvtColor(color\_enhanced\_img, cv2.COLOR\_RGB2GRAY)  
selem = morphology.disk(3)  
dilated = morphology.dilation(gray, selem)  
eroded = morphology.erosion(gray, selem)  
gradient = dilated - eroded  
gradient = np.stack((gradient,)\*3, axis=-1)  
  
# 形态学处理结果与色彩增强图像融合  
morphology\_enhanced\_img = cv2.addWeighted(color\_enhanced\_img, 0.8, gradient, 0.2, 0)  
  
# 超分辨率重建 - 双三次插值上采样  
upscaled\_img = cv2.resize(morphology\_enhanced\_img, None, fx=2, fy=2, interpolation=cv2.INTER\_CUBIC)  
  
# 显示原始图像和优化后的图像  
plt.figure(figsize=(10, 20))  
plt.subplot(1, 2, 1)  
plt.imshow(original\_img\_rgb)  
plt.title('Original Image')  
plt.axis('off')  
  
plt.subplot(1, 2, 2)  
plt.imshow(upscaled\_img)  
plt.title('Optimized Image')  
plt.axis('off')  
  
plt.show()  
  
# 将处理后的图像转换为PIL图像格式  
final\_optimized\_image\_pil = Image.fromarray(upscaled\_img)

**处理结果：**

BM3D算法去除了图像中的随机噪声，同时保留了图像细节，可能观察到的颗粒感在优化图像中减少了；通过CLAHE方法，优化后的图像展示了更丰富的细节，特别是在暗部和高光区域，图像的整体对比度得到提升，使建筑物的轮廓和特征更为清晰；Unsharp Masking的应用使得图像边缘更加锐利，图像中建筑物的边缘和纹理更加突出，增强了视觉上的清晰度，通过调整HSV颜色空间中的饱和度，优化图像的颜色看起来更鲜艳和丰满；图像的形态学处理强化了边缘信息，通过膨胀和腐蚀操作增强了边缘对比，使得建筑的轮廓更为突出；将形态学处理的结果与色彩增强图像融合后，再通过双三次插值上采样，最终得到的图像不仅在细节上有所改善，而且在视觉尺寸上也有所增加。

****

**题目二：**

请根据图像的特点，选择或设计压缩算法，完成图像压缩，给出算法描述、实现过程、压缩结果以及压缩参数计算结果。



**原理：**

**降采样：**降采样是减少图像分辨率的过程，通常是为了减小文件大小或为后续处理准备。在downsample\_image函数中，每个像素块（由因子factor指定大小）被合并成一个像素，计算块内所有像素的平均值来代表这个新像素。

**灰度化：**灰度化将RGB颜色空间的图像转换为灰度图像，减少颜色信息，只保留亮度信息。rgb\_to\_grayscale函数使用加权和方法计算灰度值，其中R、G、B三个通道的权重分别为0.2989、0.5870和0.1140。

**量化：**量化是将像素值映射到有限数量的值的过程，进一步减少图像的数据量。在quantize函数中，灰度图像的每个像素值被映射到最近的量化级别。例如，如果量化级别为16，则像素值将被映射到16个可能的值中的一个。

**伪代码：**

函数downsample\_image

新高度 <- 图像数据高度除以因子

新宽度 <- 图像数据宽度除以因子

初始化新图像数据为零矩阵，大小为 新高度 x 新宽度 x 3

对于新图像数据的每个 y, x 坐标：

块 <- 从图像数据中取出当前坐标对应的子矩阵

新图像数据在(y, x) <-计算块的平均值

返回新图像数据

函数rgb\_to\_grayscale

将每个像素的 RGB 值与固定权重相乘并求和得到灰度值

返回灰度图像数据

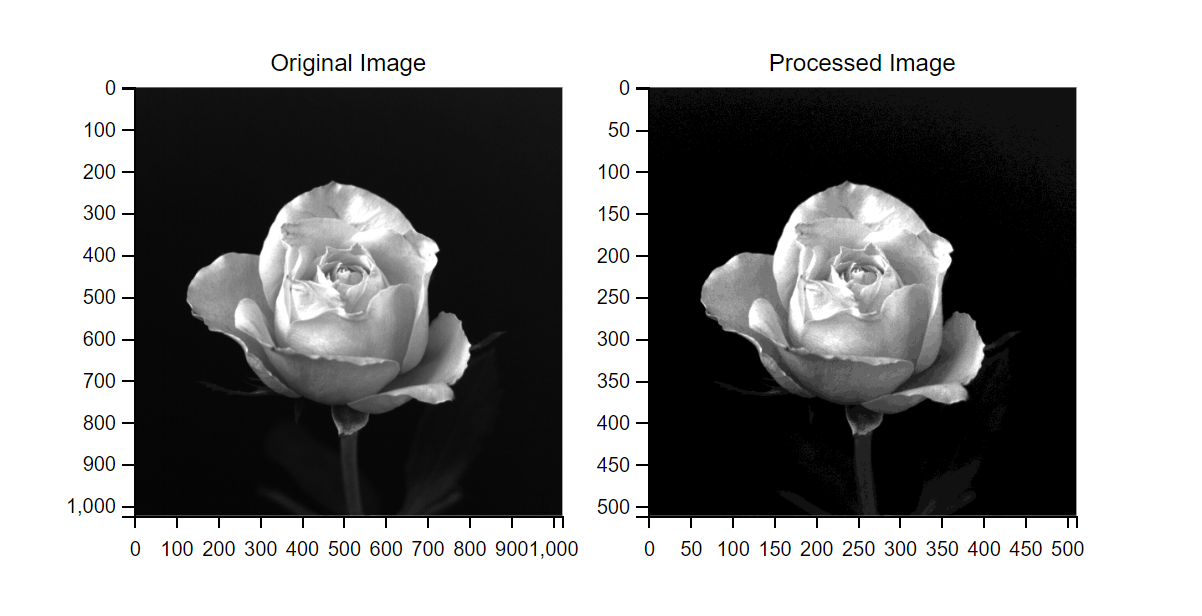
函数quantize

量化图像数据的每个像素值到指定的等级

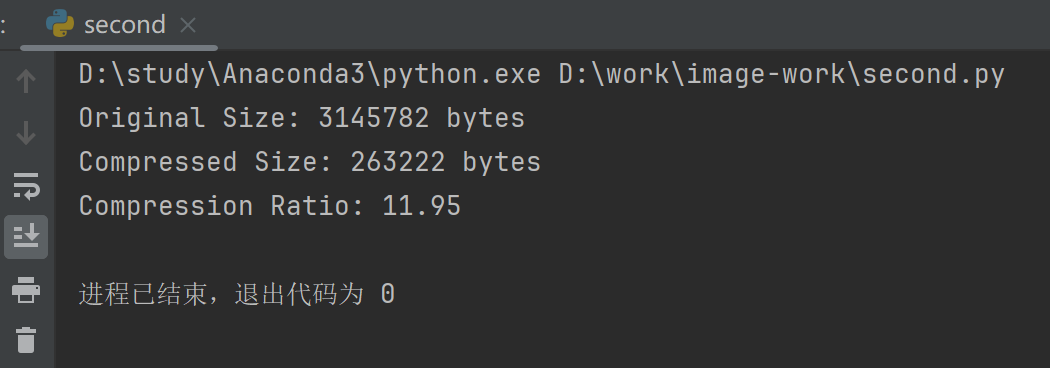
返回量化后的图像数据

**源代码：**  
from PIL import Image  
import numpy as np  
import os # 用于获取文件大小  
  
def downsample\_image(image\_data, factor=2):  
 """  
 降低图像分辨率。  
 """  
 new\_height = image\_data.shape[0] // factor  
 new\_width = image\_data.shape[1] // factor  
 downsampled\_image\_data = np.zeros((new\_height, new\_width, 3), dtype=np.uint8)  
  
 for y in range(new\_height):  
 for x in range(new\_width):  
 block = image\_data[y \* factor:(y + 1) \* factor, x \* factor:(x + 1) \* factor]  
 downsampled\_image\_data[y, x] = block.mean(axis=(0, 1))  
  
 return downsampled\_image\_data  
  
def rgb\_to\_grayscale(image\_data):  
 """  
 RGB到灰度的颜色空间转换。  
 """  
 return np.dot(image\_data[..., :3], [0.2989, 0.5870, 0.1140]).astype(np.uint8)  
  
def quantize(image\_data, levels=16):  
 """  
 将灰度值量化到指定的等级。  
 """  
 quantized = np.floor(image\_data / (256 / levels)) \* (256 / levels)  
 return quantized.astype(np.uint8)  
  
# 加载原始图像  
image\_path = '实习2.bmp'  
original\_image = Image.open(image\_path)  
original\_image\_data = np.array(original\_image)  
  
# 应用降采样  
downsampled\_image\_data = downsample\_image(original\_image\_data, factor=2)  
  
# 转换到灰度  
grayscale\_data = rgb\_to\_grayscale(downsampled\_image\_data)  
  
# 应用量化  
quantized\_data = quantize(grayscale\_data, 16)  
  
# 将处理后的数据转换回PIL图像以便保存  
processed\_image = Image.fromarray(quantized\_data)  
  
# 保存处理后的图像  
compressed\_image\_path = 'compressed\_image.bmp'  
processed\_image.save(compressed\_image\_path, 'BMP')  
  
# 获取原始和压缩后的图像文件大小  
original\_size = os.path.getsize(image\_path)  
compressed\_size = os.path.getsize(compressed\_image\_path)  
  
# 计算压缩比  
compression\_ratio = original\_size / compressed\_size  
  
print(f"Original Size: {original\_size} bytes")  
print(f"Compressed Size: {compressed\_size} bytes")  
print(f"Compression Ratio: {compression\_ratio:.2f}")

**处理结果：**

原始图像的分辨率被减少到原来的一半，显著减少了图像的像素数量，但此过程可能会导致细节的丢失，特别是细微特征；压缩后的图像将不再包含颜色信息，只有灰度信息，进一步减少每个像素所需的数据量，此后每个像素只有一个灰度值，而不是三个颜色通道的值；量化进一步减少了表示每个像素所需的信息量，将连续的灰度值映射到较小的离散集，灰度值从256个等级减少到了16个等级，使得图像的对比度降低。





## 题目三：

以下图像包含了一些噪声点并且比较模糊，请对图像进行形态学处理，以消除噪声并使目标更加清晰。

实习3

**原理：**

**灰度化：**将具有多个颜色通道的图像转换为灰度图像。

**去噪：**使用中值滤波去除图像中的噪声。这是一种非线性滤波技术，能有效去除盐和胡椒噪声。

**锐化：**使用unsharp mask增强图像的细节和边缘。

**形态学处理：**应用形态学膨胀和腐蚀操作以计算图像的形态学梯度，突出边缘，并应用白帽子变换以提取小对象或细节，这些对象比开操作中的结构元素更亮。应用形态学开操作，去除小的对象或图像上的噪点。应用形态学闭操作，用于填充小空洞和连接邻近对象。

**边缘增强与混合：**将形态学梯度和白帽子变换结果混合以增强边缘，与锐化后的图像混合，根据混合比例调整以补全两种方式的不足。

**伪代码：**

如果图像有多个颜色通道:

转换图像到灰度

将图像强度标准化到[0,1]范围

对标准化后的图像应用中值滤波去噪

定义形态学结构元素disk(2)

使用形态学开操作处理去噪后的图像

使用形态学闭操作进一步处理图像

应用unsharp mask方法锐化处理后的图像

对锐化后的图像执行形态学膨胀和腐蚀

计算形态学梯度

应用白帽子变换到锐化后的图像

将形态学梯度和白帽子变换结果混合以增强边缘

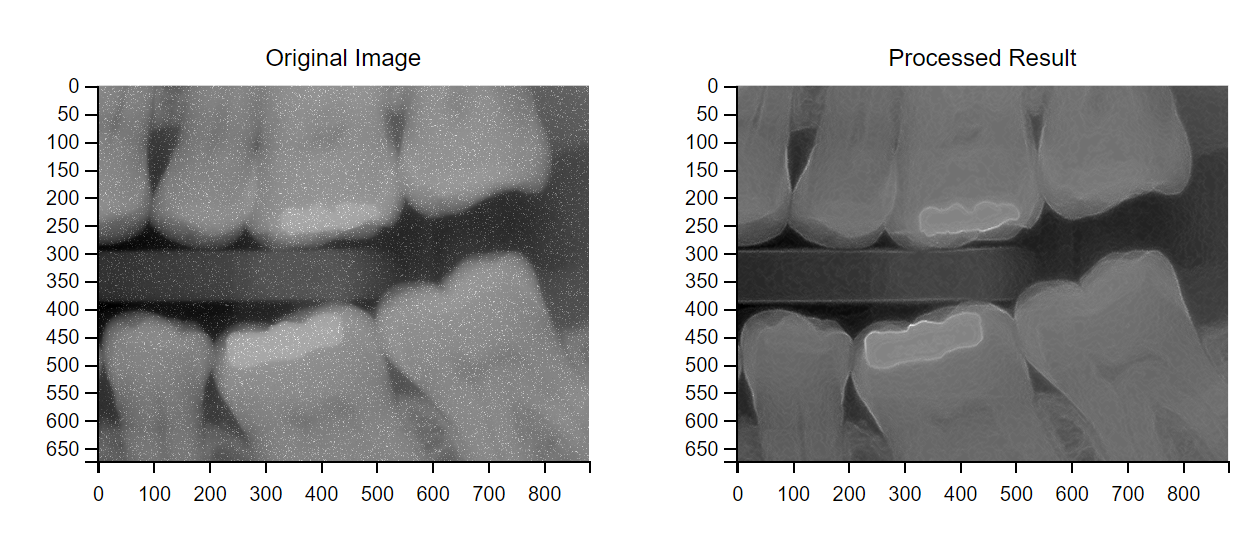
将混合的边缘与锐化后的图像混合

**源代码：**

import matplotlib.pyplot as plt  
  
image\_path = '实习3.tif'  
image = Image.open(image\_path)  
image\_array = io.imread(image\_path)  
if len(image\_array.shape) > 2:  
 image\_gray = color.rgb2gray(image\_array)  
else:  
 image\_gray = image\_array  
image\_normalized = exposure.rescale\_intensity(image\_gray, out\_range=(0, 1))  
  
denoised\_image = median(image\_normalized, morphology.disk(1))  
  
selem = disk(2)  
  
opened\_image = morphology.opening(denoised\_image, selem)  
closed\_image = morphology.closing(opened\_image, selem)  
  
sharpened\_full\_image = unsharp\_mask(closed\_image, radius=1, amount=1.5)  
  
dilated = morphology.dilation(sharpened\_full\_image, selem)  
eroded = morphology.erosion(sharpened\_full\_image, selem)  
morphological\_gradient = dilated - eroded  
top\_hat = morphology.white\_tophat(sharpened\_full\_image, selem)  
enhanced\_edges = morphological\_gradient + top\_hat  
  
mix\_ratio = 0.7  
mixed\_image = mix\_ratio \* enhanced\_edges + (1 - mix\_ratio) \* sharpened\_full\_image  
  
fig, ax = plt.subplots(1, 2, figsize=(12, 6), dpi=120)  
ax = ax.ravel()  
  
ax[0].imshow(image\_normalized, cmap='gray')  
ax[0].axis('off')  
ax[0].set\_title('Original Image')  
  
ax[1].imshow(mixed\_image, cmap='gray')  
ax[1].axis('off')  
ax[1].set\_title('Processed Result')  
  
plt.subplots\_adjust(wspace=0.3, hspace=0.4)  
plt.show()

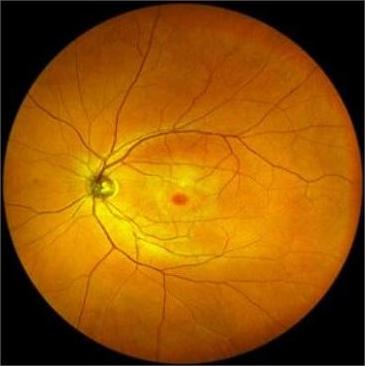
**处理结果：**

原始图像经过归一化处理后，使用中值滤波器去除噪声，中值滤波器对于去除图像中的椒盐噪声特别有效，因此处理后的图像应该看起来更加平滑，噪声点减少；通过形态学开闭运算，图像中的小物体和缝隙被处理，开运算断开连接较细的点并消除小的明亮区域，而闭运算则填充小的黑暗区域，有效清理了图像中的杂散点和小的结构缺陷。图像锐化步骤增强图像的边缘和细节，使得图像看起来更加清晰，不锐利掩模通过强调高频成分来实现这一点，使得边缘更加突出；形态学梯度通过膨胀和腐蚀操作之间的差异来突出边缘，而顶帽变换则专注于增强图像中的明亮特征；将增强的边缘图像与原始锐化图像混合，通过调整混合比例来控制两种效果的平衡。

****

## 题目四：

请设计方法从下面图像中分割出血管。



**原理：**

**直方图均衡化：**直方图均衡化是一种增强图像对比度的方法，它拉伸图像的直方图，使得像素强度分布更加均匀。自适应直方图均衡化则根据图像的局部区域来调整对比度，对于光照不均的图像特别有效。

**Sobel算子：**一种用于边缘检测的离散微分算子，结合了高斯平滑和微分求导，能够提取出图像中的边缘信息。通过计算图像中每个像素点处的梯度强度和方向，可以得到边缘的强度和方向信息。

**Multi-Otsu阈值分割：**Otsu阈值分割是一种基于类间方差最大化的自动阈值选择方法。Multi-Otsu是对Otsu方法的扩展，可以计算多个阈值，用于将图像分割成多个类别。这里使用Multi-Otsu方法来确定边缘图像的二值化阈值。

**形态学闭运算：**形态学闭运算是先膨胀后腐蚀的过程，用于填充图像中的小洞和狭窄的断裂。使用圆盘形结构元素进行闭运算，以平滑图像并连接相邻的边缘和避免提取到视网膜边缘。

**骨架化：**用于提取图像的拓扑骨架，即图像中物体的中心线。骨架化过程通过不断腐蚀图像并保留关键点来逐渐细化图像，最终得到只包含物体中心线的图像。

**伪代码：**

将图像转换为灰度图像对灰度图像应用自适应直方图均衡化

使用Sobel算子对均衡化后的图像进行边缘检测

应用Multi-Otsu阈值分割->获取多个阈值->根据中间阈值对边缘图像进行二值化

定义形态学元素为半径为3的圆盘，以防止提取到视网膜圆形边缘

对二值图像应用二值闭合操作，进行形态学闭运算填充二值图像中的小洞

使用圆盘形态学元素对闭合后的图像进行标记，识别连通区域

初始化一个空列表用来保存要删除的小区域标签

遍历所有识别的区域：

如果区域的面积小于100，在标记图像中将该区域的标签设置为0

创建清理后的图像，将标记图像中大于0的部分设置为True

对清理后的图像应用骨架化处理，提取中心线。

**源代码：**

from skimage import io, color, exposure, filters, morphology, measure, util  
import matplotlib.pyplot as plt  
  
# 图像文件路径  
image\_path = '实习4.jpg'  
# 加载图像  
image = io.imread(image\_path)  
# 转换为灰度图像  
gray\_image = color.rgb2gray(image)  
# 应用直方图均衡化增强对比度  
equalized\_image = exposure.equalize\_adapthist(gray\_image, clip\_limit=0.02)  
# 使用Sobel算子进行边缘检测  
edges\_sobel = filters.sobel(equalized\_image)  
# 使用Multi-Otsu算子进行阈值分割  
threshold\_value = filters.threshold\_multiotsu(edges\_sobel, classes=3)  
binary\_image = edges\_sobel > threshold\_value[1]  
# 形态学处理  
selem = morphology.disk(3)  
closed\_image = morphology.binary\_closing(binary\_image, selem)  
# 去除小的连通区域  
labeled\_image = measure.label(closed\_image)  
props = measure.regionprops(labeled\_image)  
for prop in props:  
 if prop.area < 100:  
 labeled\_image[labeled\_image == prop.label] = 0  
cleaned\_image = labeled\_image > 0  
# 骨架化提取中心线  
skeleton = morphology.skeletonize(cleaned\_image)  
# 保存处理后的图像  
processed\_image\_path = 'processed\_retinal\_image.png'  
io.imsave(processed\_image\_path, util.img\_as\_ubyte(skeleton))  
# 展示图像  
fig, axes = plt.subplots(1, 2, figsize=(8, 4)) # 更新为展示两个图像  
axes\_flat = axes.flatten()  
titles = ['Original Image', 'Skeleton Image']  
images = [image, skeleton]  
for idx, ax in enumerate(axes\_flat):  
 cmap = 'gray' if idx > 0 else None  
 ax.imshow(images[idx], cmap=cmap)  
 ax.axis('off')  
 ax.set\_title(titles[idx])  
plt.tight\_layout()  
plt.show()

**处理结果：**

通过灰度化处理、对比度增强来突出血管结构，使用Sobel算子进行边缘检测并使用Multi-Otsu算子进行阈值分割。将分割后的结果进行补全和清理，将最大的连通区域保留下来并骨架化处理，最终得到了较为完整的血管框架。但是由于对图片亮斑和其他视网膜结构的错误识别，有些不该出现的结构被二值化图像留存。而有一些细节较为模糊的毛细血管则没能被成功提取，但总体而言处理结果成功保留了大部分血管框架。

