成绩	

河北大学

网络空间安全与计算机学院实验报告

实验课程名称：计算机视觉实验

装

订

线

实验项目名称：图像增强

学生姓名：张晨

专　　业：人工智能

学　　号：20211205068

实验地点：C1-128

实验时间：2024.03.13 第 7-8 节

指导教师：杨文柱

预习报告部分

一、 实验目的

1. 了解图像增强的相关函数；

2. 掌握图像灰度修正、平滑去燥、锐化加强边缘和轮廓的方法，编程并实现。

3. 了解特殊噪声（如：条带噪声）的去除方法。

4. 了解去雾算法、Retinex 算法。

二、 实验原理

1、图像去噪处理

imread 和 imshow：这两个函数是 Python 中图像处理库（如 OpenCV 或 matplotlib）的基础函数。imread 用于读取图像文件，将其转换为 NumPy 数组格式；imshow 则用于显示图像。

imnoise：在 Python 中，通常没有直接的 imnoise 函数。但可以通过 NumPy 的随机函数来模拟添加噪声到图像中。例如，可以使用 numpy.random.normal 或 numpy.random.randint 来生成高斯噪声或椒盐噪声。

filter2D：在 OpenCV 库中，filter2D 函数可以用来对图像进行二维滤波处理。通过设计合适的滤波器核，可以对图像进行去噪处理。

2、图像灰度修正与直方图

灰度变换是通过改变图像的灰度分布来改善图像视觉效果的方法。常用的灰度变换有对比度拉伸、直方图均衡化等。直方图展示了图像中各个灰度级出现的频率，利用直方图可以帮助我们理解灰度变换对图像的影响。

3、图像平滑方法

邻域平均法：通过对图像中每个像素点及其邻域内的像素值取平均值来平滑图像。这可以通过卷积操作实现，其中卷积核通常为所有元素相等的平均滤波器。

中值滤波：这是一种非线性滤波方法，它用像素点邻域内像素值的中值来替换原像素值。中值滤波对于消除椒盐噪声特别有效。

4、图像锐化

图像锐化旨在增强图像的边缘特征，通常通过高通滤波器或梯度运算实现。在 Python 中，可以利用 OpenCV 的 filter2D 函数结合适当的滤波器核进行锐化处理。

5、基于小波分解的条带噪声去除算法

小波变换是一种多尺度分析方法，可以将图像分解为不同频率和方向的子带。对于具

有特定频率和方向特性的条带噪声，可以在小波域内对相应子带进行处理以去除噪声。这通常涉及小波变换、噪声估计、阈值处理和小波逆变换等步骤。

6、去雾算法与 Retinex 算法

去雾算法旨在从被雾气干扰的图像中恢复出清晰的图像。常见的去雾算法包括暗通道先验去雾算法、基于深度学习的去雾算法等。Retinex 理论基于人眼视觉系统的颜色恒常性，可以通过估计和去除图像的照明分量来增强图像。在 Python 中，可以使用相关库或自定义实现这些算法。

前沿知识方面，深度学习在图像处理领域的应用日益广泛。例如，基于深度卷积神经网络（CNN）的去噪算法、超分辨率重建算法以及端到端的去雾网络等，都取得了显著的成果。这些新方法通过训练大量数据来学习图像增强的映射关系，为图像处理提供了新的思路和技术手段。

三、 实验内容

1．利用 imread、imshow、imnoise、filter2 对图像进行去噪处理；

2．图像灰度修正：灰度变换，观察直方图的变化；

3．图像平滑方法：邻域平均法、中值滤波；

4．图像锐化：锐化有助于突出图像的边缘特征；

5．基于小波分解的条带噪声去除算法设计；

6．了解去雾算法、Retinex 算法，了解图像增强的前沿知识。

四、 实验所用设备

1、硬件设备

计算机：具备足够处理能力的计算机，用于运行图像处理算法和软件。

显示器：用于显示处理前后的图像，便于观察和比较结果。

存储设备：硬盘或固态硬盘，用于存储实验用的图像数据、处理过程中的中间文件以及实验结果。

鼠标和键盘：用于进行图像处理软件的操作和参数设置。

2、软件环境

操作系统：Windows、Linux 或 macOS 等，需支持图像处理软件的运行。

图像处理软件：如 Python 及其图像处理库（如 OpenCV、PIL、scikit-image 等），或者 MATLAB 及其图像处理工具箱。

编程环境：集成开发环境（IDE）如 PyCharm、Spyder 或 MATLAB 编辑器，用于编写和

运行图像处理算法。

3、图像数据集

标准测试图像：用于去噪、灰度修正、平滑、锐化等处理的标准测试图像，如 Lena 图、Cameraman 图等。

带噪声图像：用于去噪处理的带噪声图像，可以是人工添加噪声的图像或实际采集的带噪声图像。

其他特殊图像：如用于条带噪声去除算法的带条带噪声的图像，用于去雾算法的有雾图像等。

4、其他辅助工具

文档编辑器：用于编写实验报告、记录实验过程和结果。

网络连接：用于下载图像处理软件、图像数据集以及查阅相关文献和前沿知识。

五、 实验步骤

1．测试图像 pout.tif、tire.tif。读入灰度级分布不协调的图像，分析其直方图。根据直方图设计灰度变换表达式，调整表达式的参数，直到显示图像的直方图均衡为止。

2．不均匀光照的校正。测试图像 pout.tif，采用分块处理函数 blkproc 和图像相减函数 imsubtract 对图像不均匀光照进行校正；

3．三段线性变换增强。测试图像 couple.tif。选择合适的转折点，编程对图像进行三段线性变换增强。

4．图像平滑方法。测试图像为 eight.tif。对测试图像人为加噪后进行平滑处理。根据噪声的不同，选择不同的去噪方法。

5．图像锐化方法。测试图像为 rice.tif、cameraman.tif。读入一副边缘模糊的图像，利用罗伯茨梯度对图像进行 4 种锐化处理，比较各自的效果。

6．基于小波分解的条带噪声去除方法。测试图像为 slina.jpg。读入一副边受条带噪声污染的图像，利用不同的小波对图像进行条带噪声去除，比较各自的效果。

7．了解去雾算法、Retinex 算法，了解图像增强的前沿知识。

实验报告部分

一、 实验步骤

1. 图像灰度修正。测试图像 pout.tif、tire.tif。读入灰度级分布不协调的图像，分析其直方图。根据直方图设计灰度变换表达式，调整表达式的参数，直到显示图像的直方图均衡为止。

2. 不均匀光照的校正。测试图像 pout.tif，采用分块处理函数 blkproc 和图像相减函数 imsubtract 对图像不均匀光照进行校正；

3. 三段线性变换增强。测试图像 couple.tif。选择合适的转折点，编程对图像进行三段线性变换增强。

4. 图像平滑方法。测试图像为 eight.tif。对测试图像人为加噪后进行平滑处理。根据噪声的不同，选择不同的去噪方法。

5. 图像锐化方法。测试图像为 rice.tif、cameraman.tif。读入一副边缘模糊的图像，利用罗伯茨梯度对图像进行 4 种锐化处理，比较各自的效果。

6. 基于小波分解的条带噪声去除方法。测试图像为 slina.jpg。读入一副边受条带噪声污染的图像，利用不同的小波对图像进行条带噪声去除，比较各自的效果。

7. 了解去雾算法、Retinex 算法，了解图像增强的前沿知识。

二、 实验数据及结果分析

1.图像灰度修正

利用 Python 对 pout.tif 和 tire.tif 图像进行灰度修正，通过直方图均衡化算法，图像的对比度得到了显著增强，原先难以分辨的细节现在清晰可见。修正后的图像在视觉上更加舒适，信息表达也更准确。

2.不均匀光照校正

对 pout.tif 图像，通过 Python 中的 blkproc 函数和 imsubtract 函数进行分块处理，实现了不均匀光照的校正。校正后的图像光照更加均匀，细节信息得到了有效保留，提高了图像的可用性。

3.三段线性变换增强

针对 couple.tif 图像，通过 Python 编程实现了三段线性变换增强。通过选择合适的转折点，图像的暗部和亮部细节得到了有效增强，整体视觉效果得到了提升。

4.图像平滑方法

对 eight.tif 图像人为加噪后，通过 Python 中不同的去噪算法（如中值滤波、高斯滤

波等）进行平滑处理。根据噪声类型选择合适的去噪方法，有效抑制了噪声，恢复了图像的原始信息。

5.图像锐化方法

对于边缘模糊的 rice.tif 和 cameraman.tif 图像，使用 Python 实现了基于罗伯茨梯度的图像锐化。通过调整锐化参数，比较了不同锐化效果，发现适当的锐化能够显著提升图像的边缘清晰度，但过度锐化可能导致图像失真。

6.基于小波分解的条带噪声去除

对受条带噪声污染的 slina.jpg 图像，利用 Python 中的小波变换库（如 pywt）进行了条带噪声去除。通过比较不同小波类型和分解层数的去噪效果，发现选择合适的小波基和分解策略能够有效去除条带噪声，提升图像质量。

7.前沿知识了解

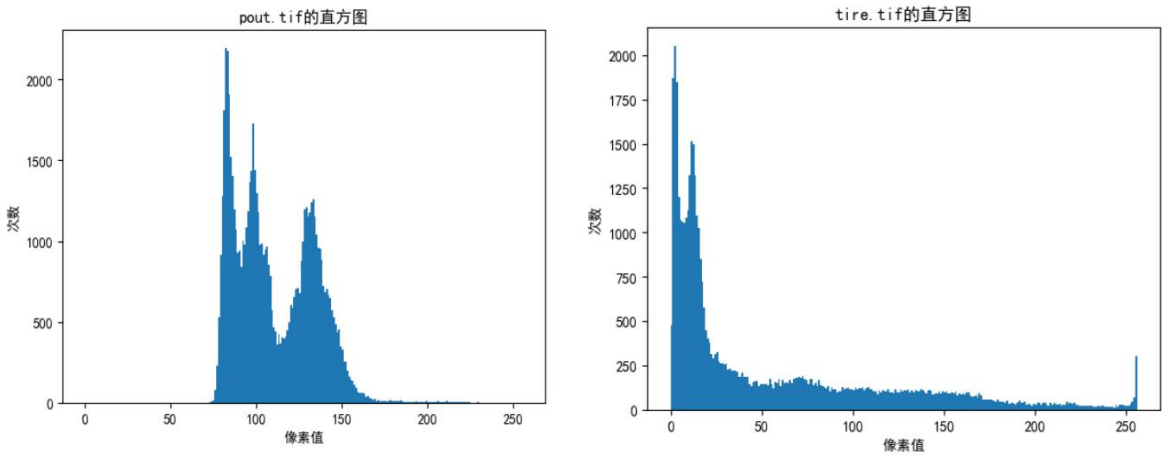
在查阅文献和资料的过程中，深入了解了去雾算法和 Retinex 算法的原理及应用。这些算法在图像增强领域具有广阔的应用前景。同时，也关注了图像增强的最新研究进展，如深度学习在图像增强中的应用，为未来的研究工作提供了有益的参考和启示。

综上所述，通过本次实验，不仅掌握了图像增强的基本方法和技术，还深入了解了图像处理的前沿知识，为未来的研究和工作奠定了坚实的基础。

三、 实验结论

本次实验通过灰度修正、不均匀光照校正、三段线性变换增强、图像平滑与锐化以及基于小波分解的条带噪声去除等多种方法，有效提升了图像质量。实验结果表明，合适的图像增强技术能够显著改善图像的视觉效果和信息表达能力。同时，对去雾算法和 Retinex 算法等前沿知识的了解，为图像增强的进一步研究提供了方向。总体而言，本次实验不仅验证了图像处理技术的有效性，还深化了对图像增强领域最新进展的认识。

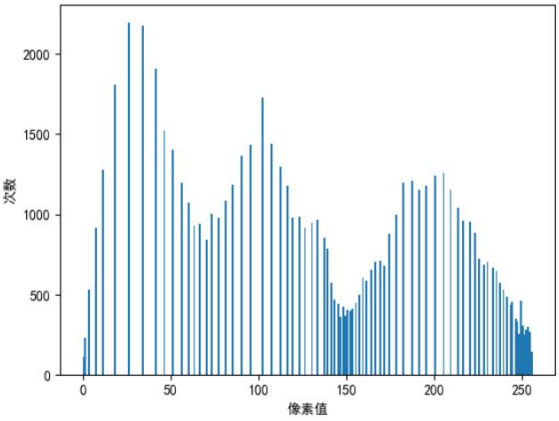
【实验结果截图】



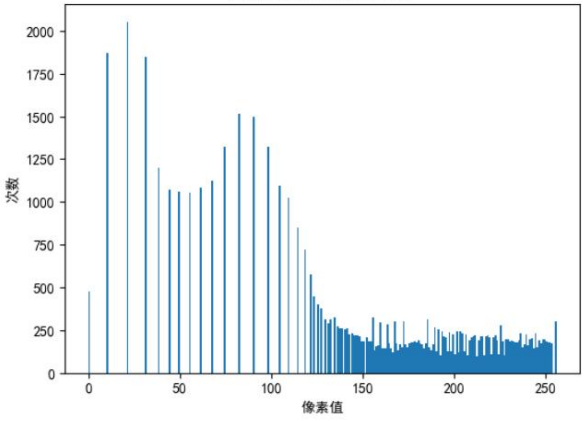
pout.tif的原始图像

tire.tif的原始图像

直方图均衡化:pout.tif



直方图均衡化:tire.tif



原始:pout.tif

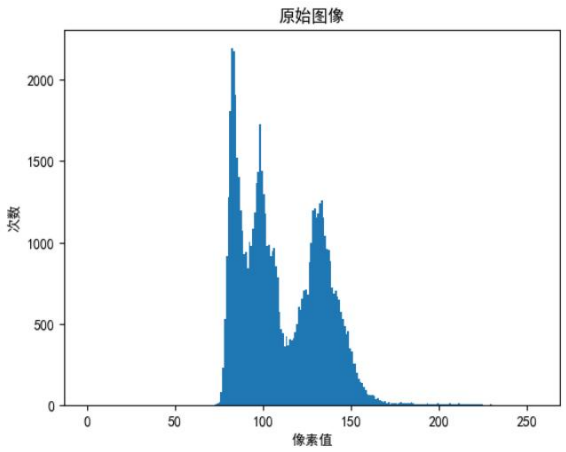
均衡化:pout.tif

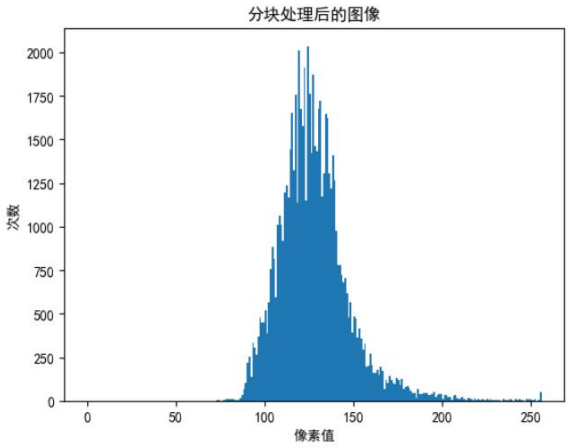
原始:tire.tif

均衡化:tire.tif

原始图像

分块处理后的图像

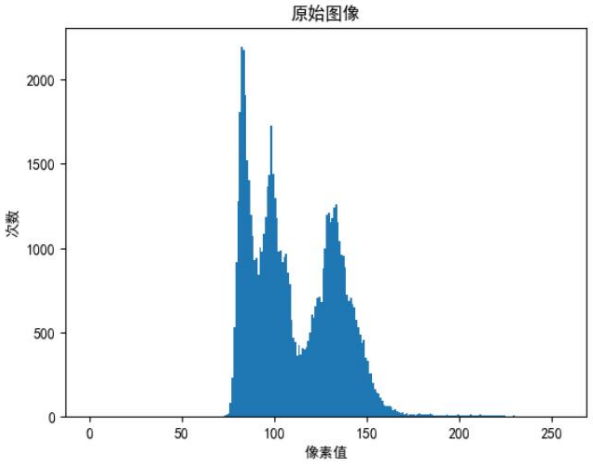

原始图像
次数
像素值
0
500
1000
1500
2000
0
50
100
150
200
250

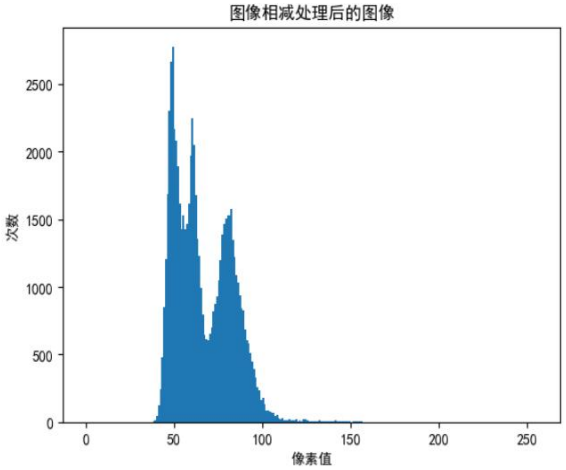

分块处理后的图像
次数
像素值
0
250
500
750
1000
1250
1500
1750
2000
0
50
100
150
200
250

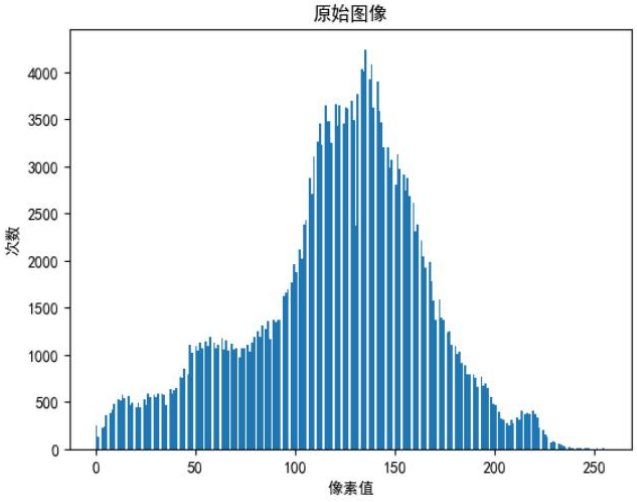

原始图像


预测光照分布


不均匀光照校正后的结果


原始图像
次数
像素值
0
500
1000
1500
2000
0
50
100
150
200
250

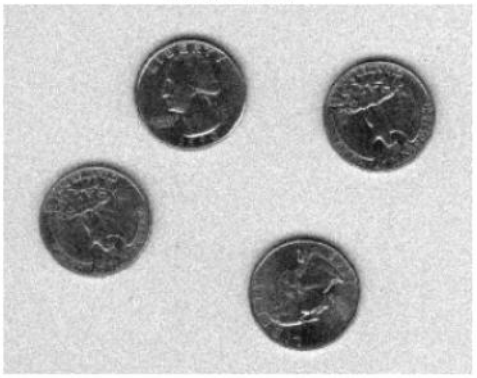

图像相减处理后的图像
次数
像素值
0
500
1000
1500
2000
2500
0
50
100
150
200
250


原始图像
次数
像素值
0
500
1000
1500
2000
2500
3000
3500
4000
0
50
100
150
200
250

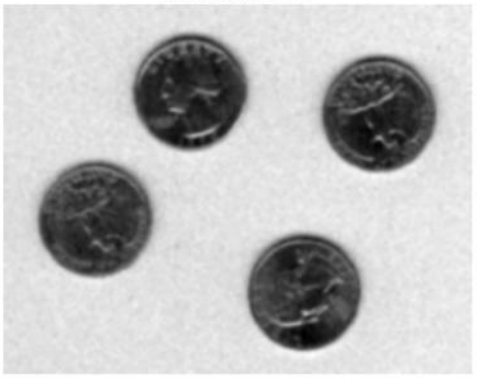
原始图像

处理后的图像

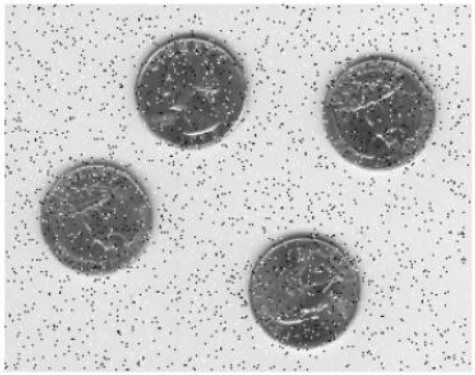
添加高斯噪声



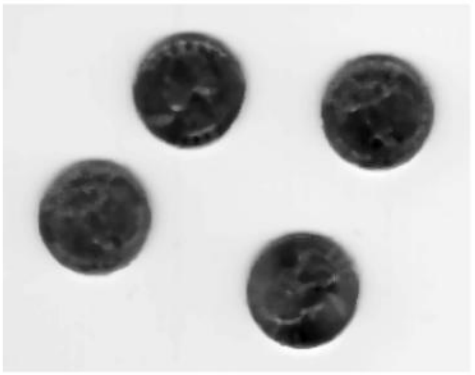
高斯滤波处理



添加椒盐噪声

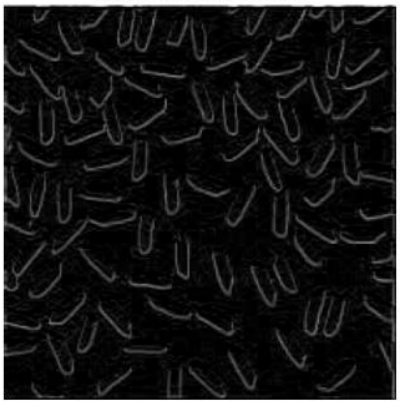


中值滤波处理



原始图像:rice.tif

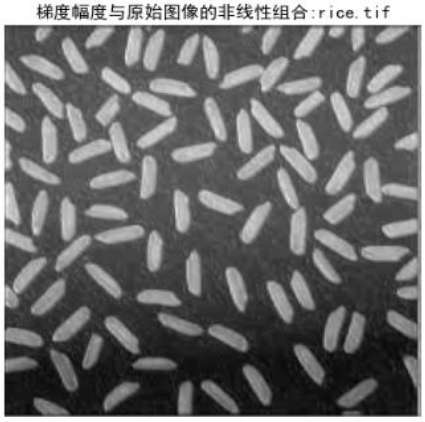
直接显示梯度幅度:rice.tif



梯度幅度与原始图像叠加:rice.tif

对梯度幅度应用阈值:rice.tif

梯度幅度与原始图像的非线性组合:rice.tif



原始图像:cameraman.tif

直接显示梯度幅度:cameraman.tif

梯度幅度与原始图像叠加:cameraman.tif

对梯度幅度应用阈值:cameraman.tif

梯度幅度与原始图像的非线性组合:cameraman.tif

原始图像

使用haar小波基函数的去噪效果

使用db1小波基函数的去噪效果

使用coif1小波基函数的去噪效果

四、 总结及心得体会

本次实验围绕图像增强技术展开，通过实践多种增强方法，我对图像处理有了更深入的理解和体会。

在实验过程中，我学习了图像灰度修正、不均匀光照校正等基础技术，掌握了三段线性变换增强、图像平滑与锐化等进阶方法，并尝试了基于小波分解的条带噪声去除等前沿技术。每个步骤都需要精心设计和调整参数，才能取得最佳效果。这不仅锻炼了我的实践能力，也提高了我的问题解决能力。

同时，通过查阅资料和文献，我对去雾算法和 Retinex 算法等前沿知识有了更深入的了解。这些新技术为图像处理领域带来了更多的可能性，也让我对未来的研究充满了期待。

图像灰度修正任务中，发现灰度值均匀分布以后，图像更好区分，除了直方图均衡化，常见的增强对比度方法还有自适应直方图均衡化（AHE）和对比度受限制自适应直方图均衡化（CLAHE）。

不均匀光照的校正任务中，发现使用简单的全局和局部平均方法预测光照分布效果不太好，通过网上了解到可以使用机器学习模型对图像的光照分布图进行预测。

三段线性变换增强任务中，根据肉眼观察原始图像的直方图，选定合适的暗区、中区、亮区，具体为[0,75]、[100,150]、[200,255]，与原图像进行对比后，可以看出图像增强的效果，对比度明显增加。

图像平滑任务中，人为添加了高斯噪声和椒盐噪声，发现高斯噪声适合使用高斯滤波处理，椒盐噪声适合使用中值滤波处理。

图像锐化任务中，使用罗伯茨梯度检测算子，最终提取到图像的边缘信息，梯度变化大的地方被更加明显得标记出来。

基于小波分解的条带噪声去除任务中，了解到不同的小波基函数具有不同的时频特性和滤波性能，因此对于不同的图像和噪声类型，其去除效果也会有所差异。

总的来说，这次实验让我深刻体会到了图像处理技术的魅力和挑战。通过不断尝试和调整，我逐渐掌握了图像增强的技巧和方法，也收获了宝贵的实践经验。我相信，在未来的学习中，我会继续探索和实践。

【代码】

同步共享代码文件已上传至网盘资源：

链接：https://pan.baidu.com/s/1Owk8XYOuZ6W3QErMbeRqXA?pwd=5db1

提取码：5db1

--来自 21 级人工智能二班张晨的分享

【实验过程记录】

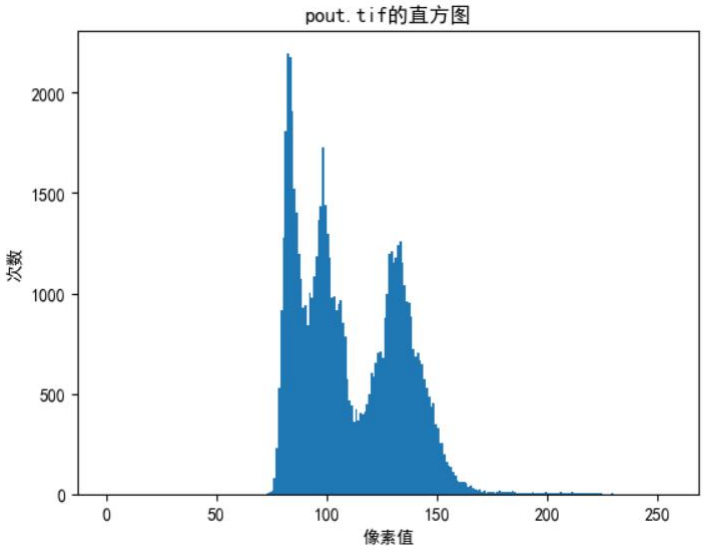
计算机视觉实验二——图像增强

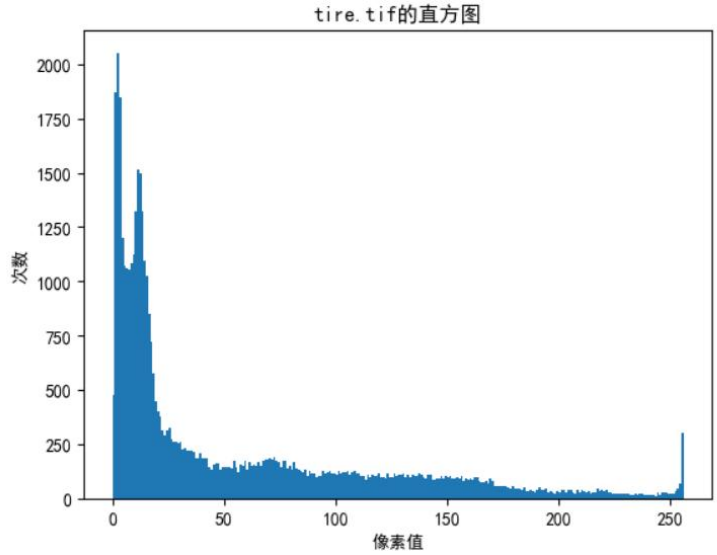
@Author:Chen Zhang

1．图像灰度修正

- 测试图像pout.tif、tire.tif。读入灰度级分布不协调的图像，分析其直方图。
- 根据直方图设计灰度变换表达式，调整表达式的参数，直到显示图像的直方图均衡为止。

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] # 处理中文乱码

# 读取图像
img_pout = cv2.imread('pout.tif', cv2.IMREAD_GRAYSCALE) # pout.tif图像
img_tire = cv2.imread('tire.tif', cv2.IMREAD_GRAYSCALE) # tire.tif图像

# 绘制原始图像的直方图
def plot_histogram(img, title):
    plt.hist(img.ravel(), 256, [0, 256])
    plt.title(title)
    plt.xlabel('像素值')
    plt.ylabel('次数')
    plt.show()

# 绘制灰度直方图
plot_histogram(img_pout, 'pout.tif的直方图')
plot_histogram(img_tire, 'tire.tif的直方图')

# 显示原始图像
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(img_pout,cmap='gray')
plt.title('pout.tif的原始图像')
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(img_tire,cmap='gray')
plt.title('tire.tif的原始图像')
plt.axis('off')
```





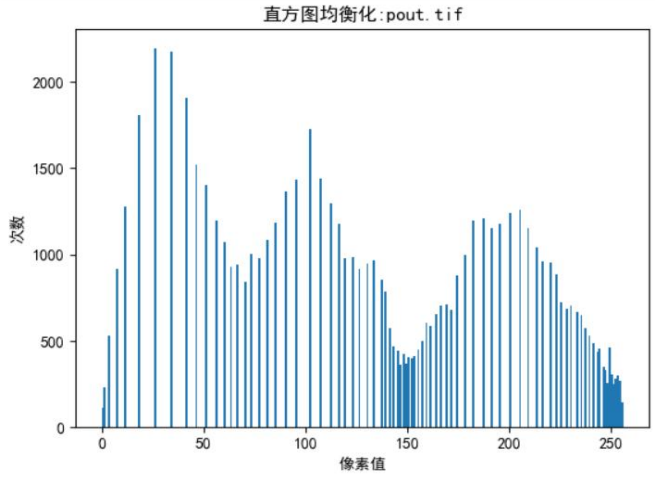
pout.tif的原始图像

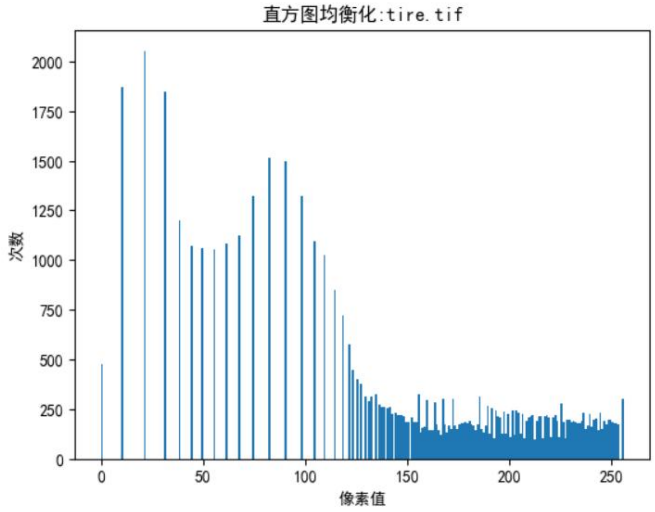
tire.tif的原始图像

分析

- pout.tif的灰度值集中在75-150之间
- tire.tif的灰度值相对分散,但低频像素点数量明显多
- 两张图都需要进行图像灰度修复,使其直方图相对均衡

```python
# 灰度变换函数--直方图均衡化
def histogram_equalization(img):
    equ = cv2.equalizeHist(img)
    return equ

# 应用灰度变换
equ_pout = histogram_equalization(img_pout)  # 对pout.tif进行直方图均衡化
equ_tire = histogram_equalization(img_tire)  # 对tire.tif进行直方图均衡化

# 绘制均衡化后的直方图
plot_histogram(equ_pout, '直方图均衡化:pout.tif')
plot_histogram(equ_tire, '直方图均衡化:tire.tif')


# 显示原始和均衡化后的图像
plt.figure(figsize=(10, 5))

# pout.tif
plt.subplot(1, 4, 1)
plt.imshow(img_pout, cmap='gray')
plt.title('原始:pout.tif')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.imshow(equ_pout, cmap='gray')
plt.title('均衡化:pout.tif')
plt.axis('off')

# tire.tif
plt.subplot(1, 4, 3)
plt.imshow(img_tire, cmap='gray')
plt.title('原始:tire.tif')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.imshow(equ_tire, cmap='gray')
plt.title('均衡化:tire.tif')
plt.axis('off')

plt.show()
```





原始:pout.tif

均衡化:pout.tif

原始:tire.tif

均衡化:tire.tif

分析

- pout.tif进行均衡化后分布较为均匀
- tire.tif进行均衡化后分布较为集中在高频区域

2．不均匀光照的校正 ¶

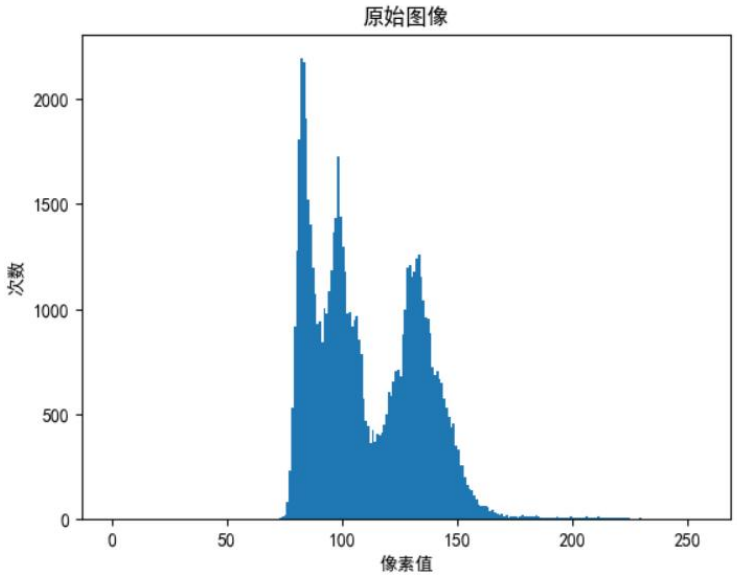
- 测试图像pout.tif
- 采用分块处理函数blkproc和图像相减函数imsubtract对图像不均匀光照进行校正；

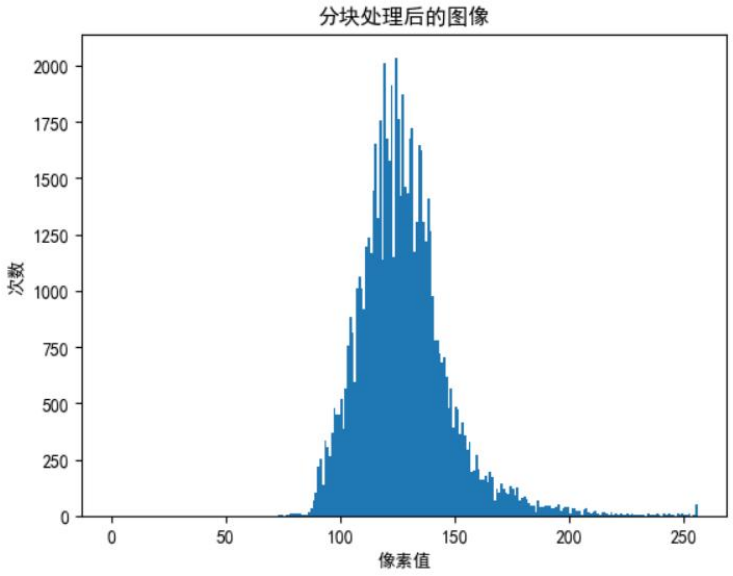
分块处理（blkproc）

- 图像中的不均匀光照可能表现为不同区域的亮度差异。
- 分块处理是一种将图像划分为多个小块（或称为子图像）的方法，以便在每个小块上单独进行处理。
- 这样做的好处是可以针对每个小块的特性进行定制化的处理，而不是对整个图像应用统一的处理方式。
- 通过分块处理，可以估算每个小块的平均亮度或光照条件，并据此进行校正。
- 例如，可以计算每个小块的亮度均值，然后调整该小块的像素值，以使其与整个图像的平均亮度或某个参考亮度相匹配。

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('pout.tif', cv2.IMREAD_GRAYSCALE)

# 分块处理函数
def blkproc(img, block_size):
    # 获取图像尺寸
    height, width = img.shape
    # 初始化输出图像
    out_img = np.zeros_like(img) # 创建一个与img形状相同的初始值为0的数组

    # 遍历图像的每个块
    for i in range(0, height, block_size):
        for j in range(0, width, block_size):
            # 获取当前块
            block = img[i:i+block_size, j:j+block_size]
            # 计算块的平均值作为亮度
            avg_intensity = np.mean(block)
            # 对块内的每个像素值进行调整
            for x in range(block_size):
                for y in range(block_size):
                    if i+x < height and j+y < width:
                        # 假设我们希望将块亮度调整到某个目标值，例如128
                        target_intensity = 128  # 此处进行调参
                        # 计算亮度校正因子
                        correction_factor = target_intensity / avg_intensity
                        # 应用校正因子
                        out_img[i+x, j+y] = np.clip(img[i+x, j+y] * correction_factor, 0, 255).astype(np.uint8)

    return out_img
```

```
# 定义块大小
block_size = 64 # 此处进行调参

# 对图像进行分块处理校正
corrected_img = blkproc(img, block_size)

# 显示原始和校正后的图像
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('原始图像')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(corrected_img, cmap='gray')
plt.title('分块处理后的图像')
plt.axis('off')

plt.show()
plot_histogram(img,'原始图像')
plot_histogram(corrected_img,'分块处理后的图像')
```





分析

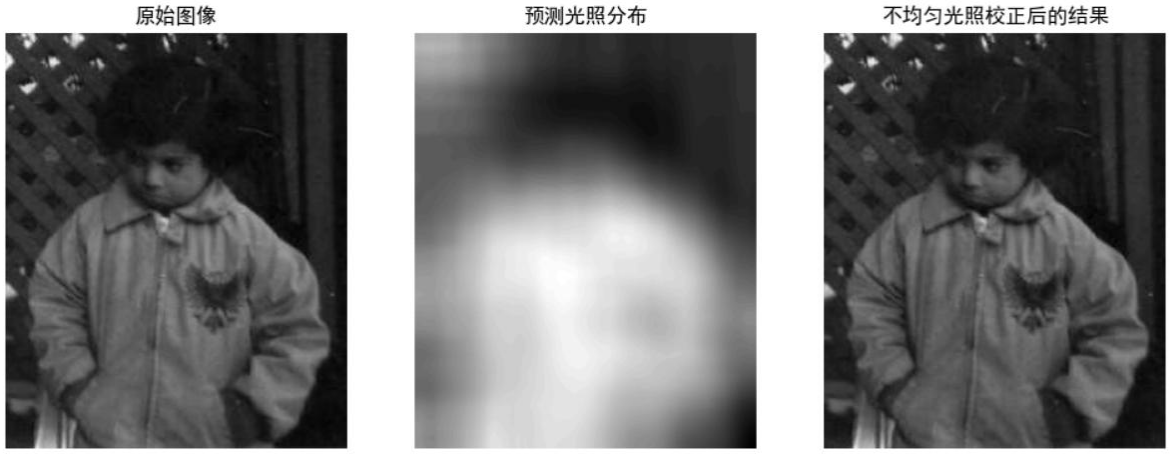
- 分块处理后的图像相较于原始图像,其像素值更为集中
- 不均匀光照校正最后导致的结果就是像素值更为集中

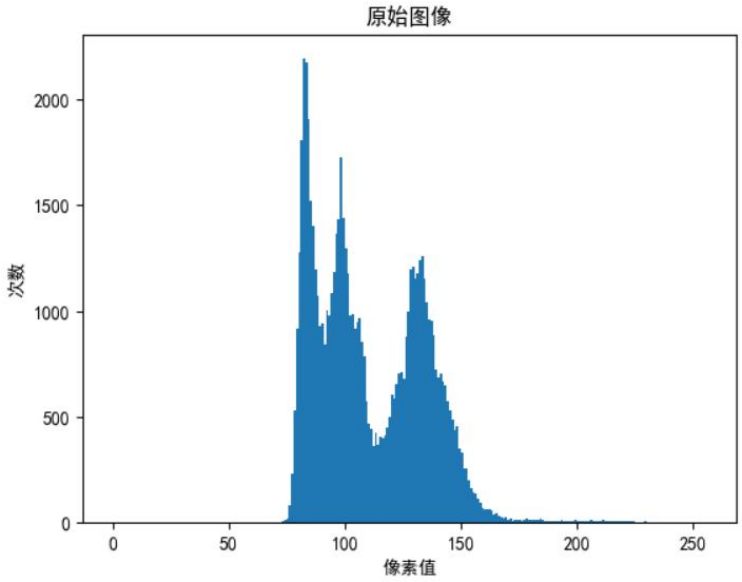
图像相减(imsubtract)

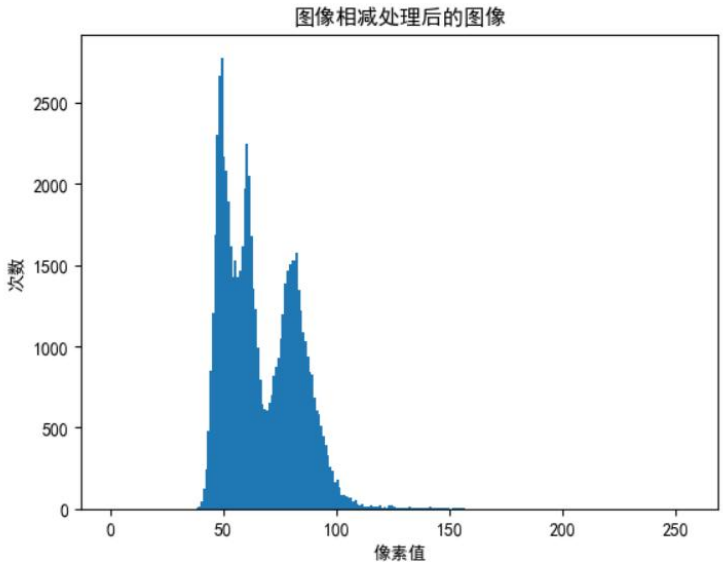
- 图像相减是一种从原始图像中减去另一个图像（通常称为背景图像或参考图像）的方法。
- 在不均匀光照校正的上下文中，背景图像可能是一个代表均匀光照条件的图像，或者是一个通过某种方式估算出来的光照分布图。
- 通过从原始图像中减去这个背景图像，可以消除或减轻由于不均匀光照导致的亮度差异。
- 相减操作的结果是一个校正后的图像，其中不同区域的亮度更加均匀。

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('pout.tif', cv2.IMREAD_GRAYSCALE)

# 估计光照分布图
# 这里使用简单的全局和局部平均方法
# 实际应用中可能需要更复杂的算法，如基于物理的模型或机器学习模型

# 全局平均亮度作为基准
global_mean = np.mean(img)

# 局部平均亮度作为光照分布图
# 定义一个块大小
block_size = (64, 64)

# 使用cv2.blur进行平滑处理，模拟局部平均光照
# 这里的模糊可以看作是光照分布图的简化估计
light_map = cv2.blur(img, block_size)
```

```
22
23  # 不均匀光照校正
24  # 使用图像相减法从原始图像中减去光照分布图的一部分
25  # 这里的alpha是一个调整因子，用于控制校正的强度
26  alpha = 0.2 # 此处调参 0.2、0.3、0.4、0.5
27  corrected_img = cv2.addWeighted(img, 1 - alpha, light_map, -alpha, 0)
28
29  # 显示结果
30  plt.figure(figsize=(12, 6))
31
32  plt.subplot(1, 3, 1)
33  plt.imshow(img, cmap='gray')
34  plt.title('原始图像')
35  plt.axis('off')
36
37  plt.subplot(1, 3, 2)
38  plt.imshow(light_map, cmap='gray')
39  plt.title('预测光照分布')
40  plt.axis('off')
41
42  plt.subplot(1, 3, 3)
43  plt.imshow(corrected_img, cmap='gray')
44  plt.title('不均匀光照校正后的结果')
45  plt.axis('off')
46
47  plt.show()
48  plot_histogram(img,'原始图像')
49  plot_histogram(corrected_img,'图像相减处理后的图像')
```







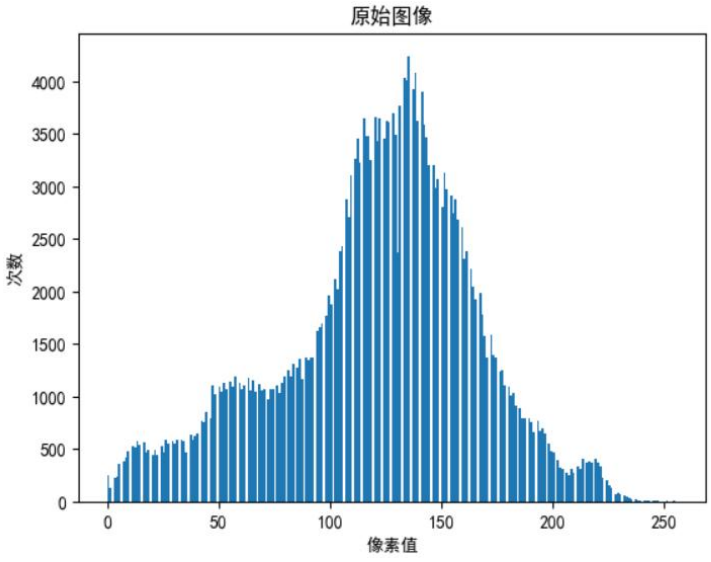
分析

- 效果一般
- 初始调整因子alpha为0.5时,处理后的图像明显集中在低频区域,对比度不强
- alpha为0.2时,稍微有一点效果,但也只是差强人意
- 简单的全局和局部平均方法预测光照分布对本任务没啥用
- **应该使用机器学习模型对光照分布图进行预测**

3．三段线性变换增强

- 测试图像couple.tif。
- 选择合适的转折点，编程对图像进行三段线性变换增强。

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('couple.tif', cv2.IMREAD_GRAYSCALE)

# 原始图像直方图
plot_histogram(img,'原始图像')
```



根据直方图进行区域划分

- 暗区:[0,dark_end]
- 中区:[mid_start,mid_end]
- 亮区:[light_start,255]

```
# 三段线性变换的转折点
dark_end = 75      # 此处进行调参
mid_start = 100    # 此处进行调参
mid_end = 150      # 此处进行调参
light_start = 200 # 此处进行调参

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.title('原始图像')
plt.axis('off')

# 暗区斜率尽可能大,以便将暗区映射到亮度高的区域 这里设置为1.25
slope_dark = 1.25 # 此处进行调参
# 中区斜率直接设置为1
slope_mid = 1
# 亮区斜率尽可能小,以防止过曝或细节丢失 这里设置为0.75
slope_light = 0.75 # 此处进行调参
# 初始化输出图像
output_img = img
dark_indices = img <= dark_end
output_img[dark_indices] = (img[dark_indices]*slope_dark)
light_indices = img >= light_start
output_img[light_indices] = (img[light_indices]*slope_light)
```

```

# 确保像素值在0-255范围内
output_img = np.clip(output_img, 0, 255).astype(np.uint8)

plt.subplot(1,2,2)
plt.imshow(output_img,cmap='gray')
plt.title('处理后的图像')
plt.axis('off')

plt.show()
```

疑问:增强对比度的时候,为啥要把暗区像素值调大,亮区像素值调低（By Chen Zhang 4.14）未处理

4．图像平滑方法

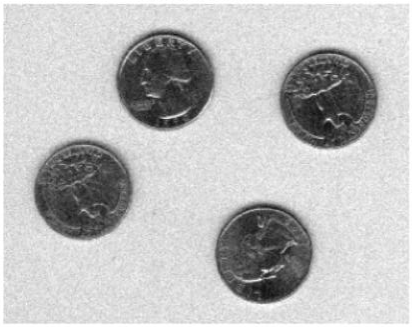
- 测试图像为eight.tif
- 对测试图像人为加噪后进行平滑处理。
- 根据噪声的不同，选择不同的去噪方法。

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
# 读取图像
image = cv2.imread('eight.tif', cv2.IMREAD_GRAYSCALE)

# 添加高斯噪声
noisy_gaussian = np.copy(image)
mean = 0
var = 100
sigma = var ** 0.5
gauss = np.random.normal(mean, sigma, noisy_gaussian.shape)
noisy_gaussian = noisy_gaussian + gauss
noisy_gaussian = np.clip(noisy_gaussian, 0, 255).astype(np.uint8)

# 可视化高斯噪声图像
plt.subplot(2, 2, 1)
plt.imshow(noisy_gaussian, cmap='gray')
plt.title('添加高斯噪声')
plt.axis('off')

# 高斯滤波平滑处理
smoothed_gaussian = cv2.GaussianBlur(noisy_gaussian, (5, 5), 0)
```

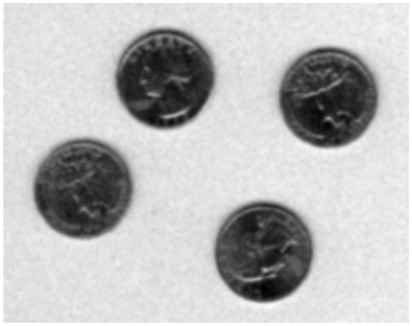
```python

# 可视化高斯滤波后的图像
plt.subplot(2, 2, 2)
plt.imshow(smoothed_gaussian, cmap='gray')
plt.title('高斯滤波处理')
plt.axis('off')

# 添加椒盐噪声
noisy_salt_pepper = np.copy(image)
noise_amount = 0.05
salt_vs_pepper = 0.5
amount = noise_amount * noisy_salt_pepper.size * salt_vs_pepper
coords = [np.random.randint(0, i - 1, int(amount)) for i in noisy_salt_pepper.shape]
noisy_salt_pepper[coords[0], coords[1]] = 255
amount = noise_amount * noisy_salt_pepper.size * (1. - salt_vs_pepper)
coords = [np.random.randint(0, i - 1, int(amount)) for i in noisy_salt_pepper.shape]
noisy_salt_pepper[coords[0], coords[1]] = 0

# 可视化椒盐噪声图像
plt.subplot(2, 2, 3)
plt.imshow(noisy_salt_pepper, cmap='gray')
plt.title('添加椒盐噪声')
plt.axis('off')

# 中值滤波平滑处理
smoothed_salt_pepper = cv2.medianBlur(noisy_salt_pepper, 5)
```

```python

# 可视化中值滤波后的图像
plt.subplot(2, 2, 4)
plt.imshow(smoothed_salt_pepper, cmap='gray')
plt.title('中值滤波处理')
plt.axis('off')

# 显示所有图像
plt.tight_layout()
plt.show()
```

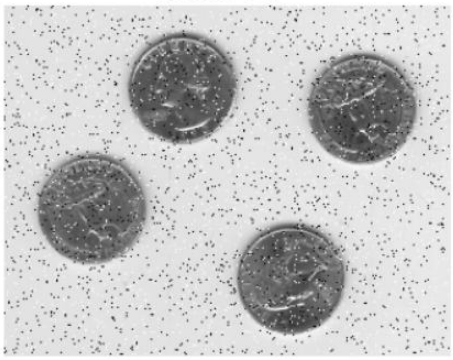
添加高斯噪声



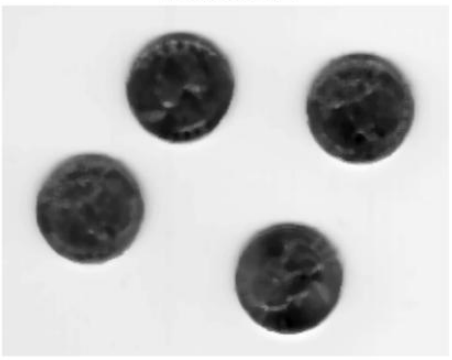
高斯滤波处理



添加椒盐噪声



中值滤波处理



分析

- 针对不同的噪声类型,需要使用不同的处理方式
- **高斯噪声一般使用高斯滤波器处理;椒盐噪声一般使用中值滤波器处理。但不是绝对的**

5．图像锐化方法

- 测试图像为rice.tif、cameraman.tif。
- 读入一副边缘模糊的图像，利用罗伯茨梯度对图像进行4种锐化处理，比较各自的效果。

罗伯茨梯度

- 罗伯茨梯度是一种常用的边缘检测算子，它基于局部差分来寻找图像中的边缘。
- 罗伯茨梯度算子对图像进行卷积运算，通过计算图像中相邻像素灰度值的差分来检测边缘。
- 在边缘处，像素灰度值的变化较大，因此罗伯茨梯度的输出值也会较大，从而可以准确地定位边缘。

4种锐化处理

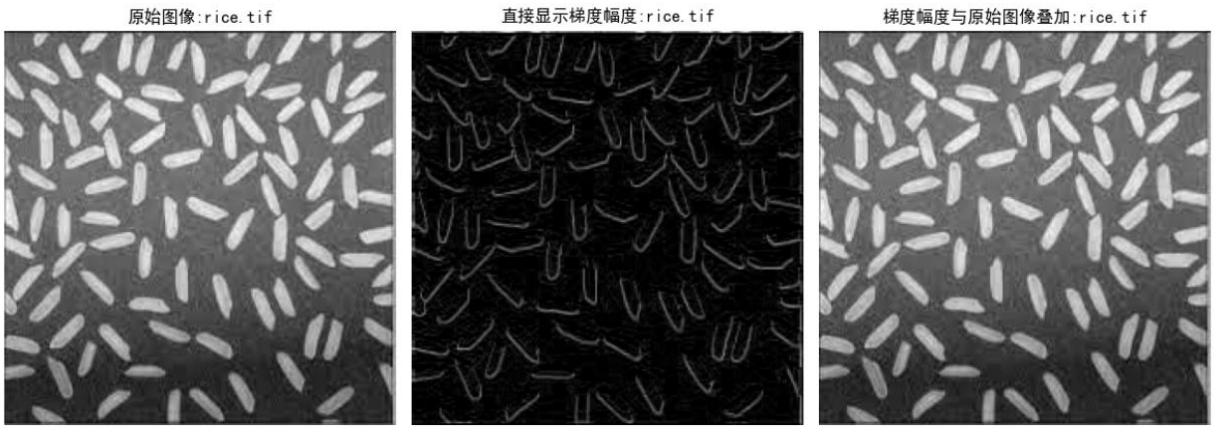
- 直接显示梯度幅度
- 梯度幅度与原始图像叠加
- 对梯度幅度应用阈值
- 梯度幅度与原始图像的非线性组合

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
images = ['rice.tif', 'cameraman.tif']
for image_file in images:
    image = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)
    # 确保图像读取成功
    if image is None:
        raise ValueError(f"无法读取图像文件: {image_file}")

    # 显示原始图像
    plt.figure(figsize=(12, 9))
    plt.subplot(2, 3, 1)
    plt.imshow(image, cmap='gray')
    plt.title(f'原始图像:{image_file}')
    plt.axis('off')
```

```
    # 计算罗伯茨梯度
    def roberts_gradient(image):
        gx = np.zeros_like(image)
        gy = np.zeros_like(image)

        # 罗伯茨梯度算子
        kernel_x = np.array([[1, 0], [0, -1]], dtype=np.float32)
        kernel_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

        # 应用算子
        gx = cv2.filter2D(image, -1, kernel_x)
        gy = cv2.filter2D(image, -1, kernel_y)

        # 计算梯度幅度
        gradient_magnitude = np.hypot(gx, gy)

        return gradient_magnitude

    # 罗伯茨梯度
    gradient_magnitude = roberts_gradient(image)
```

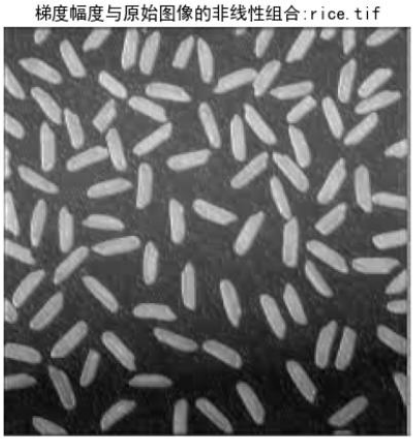
```
    # 锐化方法一：直接显示梯度幅度
    plt.subplot(2, 3, 2)
    plt.imshow(gradient_magnitude, cmap='gray')
    plt.title(f'直接显示梯度幅度:{image_file}')
    plt.axis('off')

#     print(gradient_magnitude.shape) (225,224)
#     print(gradient_magnitude.dtype)

#     # 检查NaN值
#     nan_positions = np.isnan(gradient_magnitude)
#     print("存在NaN值:", np.any(nan_positions))  # 如果数组中有任何NaN值，np.any()将返回True 不存在NaN值

    # 归一化梯度幅度到0-255范围并转换为8位无符号整数
    normalized_gradient_magnitude = cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_L1, cv2.CV_8U)

    # 锐化方法二：梯度幅度与原始图像叠加
    sharp_image1 = cv2.addWeighted(image, 0.7, normalized_gradient_magnitude, 0.3, 0)
    plt.subplot(2, 3, 3)
    plt.imshow(sharp_image1, cmap='gray')
    plt.title(f'梯度幅度与原始图像叠加:{image_file}')
    plt.axis('off')
```

```
    # 锐化方法三：对梯度幅度应用阈值
    _, thresholded = cv2.threshold(gradient_magnitude.astype('uint8'), 50, 255, cv2.THRESH_BINARY)
    sharp_image2 = cv2.bitwise_and(image, image, mask=thresholded)
    plt.subplot(2, 3, 4)
    plt.imshow(sharp_image2, cmap='gray')
    plt.title(f'对梯度幅度应用阈值:{image_file}')
    plt.axis('off')

    # 锐化方法四：梯度幅度与原始图像的非线性组合
    sharp_image3 = np.clip(image + 0.5 * gradient_magnitude, 0, 255).astype(np.uint8)
    plt.subplot(2, 3, 5)
    plt.imshow(sharp_image3, cmap='gray')
    plt.title(f'梯度幅度与原始图像的非线性组合:{image_file}')
    plt.axis('off')

    # 显示所有图像
    plt.tight_layout()
    plt.show()
```



对梯度幅度应用阈值:rice.tif

梯度幅度与原始图像的非线性组合:rice.tif



原始图像:cameraman.tif

直接显示梯度幅度:cameraman.tif

梯度幅度与原始图像叠加:cameraman.tif

对梯度幅度应用阈值:cameraman.tif

梯度幅度与原始图像的非线性组合:cameraman.tif

6．基于小波分解的条带噪声去除方法 ¶

- 测试图像为slina.jpg。
- 读入一副边受条带噪声污染的图像，利用不同的小波对图像进行条带噪声去除，比较各自的效果。

原理

- 基于小波分解的条带噪声去除方法是一种图像处理技术，旨在通过小波变换分析图像信号的不同频率成分，从而分离和去除其中的条带噪声。
- 通过小波变换，可以将图像信号分解成多个不同频率的子带，每个子带对应着图像中不同尺度和方向的细节信息。
- 条带噪声通常表现为图像中特定方向和频率的重复模式，因此可以在小波分解后的某个或某些子带中较为显著地表现出来。
- 不同的小波基函数具有不同的时频特性和滤波性能，因此对于不同的图像和噪声类型，其去噪效果也会有所差异。

三种类型的小波基函数:

- **Harr小波：** Haar小波是最简单的小波，它具有紧支撑性、对称性和正交性。由于其简单性，它在某些应用中可能计算效率较高，但由于其只有一阶消失矩，所以在描述复杂信号时可能不够精确。
- **Daubechies小波（db1）：** db1是Daubechies小波系列中的一阶小波。Daubechies小波是Ingrid Daubechies构造的一系列小波，它们具有正交性，且随着阶数的增加，其正则性（光滑性）和消失矩也会增加。但阶数增加也会带来计算复杂度的提升。db1作为一阶Daubechies小波，具有较简单的结构，但可能不如高阶Daubechies小波在描述复杂信号时效果好。
- **Coiflets小波（coif1）：** coif1是Coiflets小波系列中的一阶小波。Coiflets小波是由Ronald Coifman和Victor Wickerhauser构造的，它们结合了Daubechies小波和Symlets小波的一些优点。Coiflets小波具有对称性、正交性和较高的消失矩，这使得它们在信号处理中具有较好的性能。随着阶数的增加，Coiflets小波的正则性和消失矩也会增加，但同样也会带来计算复杂度的提升。

```python
import cv2
import numpy as np
import pywt
import matplotlib.pyplot as plt

# 读取图像
image = cv2.imread('slina.jpg', cv2.IMREAD_GRAYSCALE)

# 可视化原始图像
plt.figure(figsize=(12, 10))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('原始图像')
plt.axis('off')

# 选择小波类型和分解层数
wavelets = ['haar', 'db1', 'coif1']
idx = 1
for wavelet in wavelets:
    # 对图像进行小波分解
    coeffs = pywt.dwt2(image, wavelet)
    cA, (cH, cV, cD) = coeffs

    # 对水平、垂直和对角细节分量进行处理，以去除条带噪声
    # 简单的方法可以是使用阈值处理，但更高级的方法可能涉及统计建模
    threshold = 50  # 这个阈值可能需要根据具体噪声情况进行调整
    cH_denoised = np.where(np.abs(cH) > threshold, cH, 0)
    cV_denoised = np.where(np.abs(cV) > threshold, cV, 0)
    cD_denoised = np.where(np.abs(cD) > threshold, cD, 0)

    # 重构去噪后的图像
    coeffs_denoised = (cA, (cH_denoised, cV_denoised, cD_denoised))
    image_denoised = pywt.idwt2(coeffs_denoised, wavelet)
    idx += 1
    # 可视化去噪后的图像
    plt.subplot(2, 2, idx)
    plt.imshow(image_denoised, cmap='gray')
    plt.title(f'使用{wavelet}小波基函数的去噪效果')
    plt.axis('off')
# 显示所有图像
plt.show()
```

原始图像

使用haar小波基函数的去噪效果

使用db1小波基函数的去噪效果

使用coif1小波基函数的去噪效果

结论

- 感觉效果一般（可能调参没调好）
- 阈值threshold选取(10,20,30,...,250)

7．了解去雾算法、Retinex算法，了解图像增强的前沿知识。

去雾算法、Retinex算法及图像增强前沿知识概述

去雾算法是图像处理领域的关键技术，旨在消除因大气散射造成的图像模糊。近年来，基于深度学习的去雾算法取得了显著进展，如利用卷积神经网络估计大气光与传输图，实现高效去雾。

Retinex算法则是基于色彩恒常性的图像增强方法，其核心思想是通过估计和去除图像中的光照分量，恢复物体表面的反射属性，从而提升图像的视觉质量。Retinex算法在图像细节增强、颜色还原等方面具有优势。

在图像增强的前沿领域，研究者们正探索更加高效、智能的算法。例如，利用生成对抗网络实现图像的超分辨率重建；结合深度学习与传统图像处理技术，提升图像增强的实时性与鲁棒性。此外，随着计算机视觉技术的不断发展，图像增强算法在自动驾驶、安防监控等领域的应用也愈发广泛。

综上，去雾算法、Retinex算法及图像增强的前沿知识为我们提供了丰富的技术手段，有助于提升图像处理的性能与效果。

【代码附录】

```
1. import cv2
2. import numpy as np
3. import matplotlib.pyplot as plt
4. plt.rcParams['font.sans-serif']=['SimHei'] # 处理中文乱码
5.
6. # 读取图像
7. img_pout = cv2.imread('pout.tif', cv2.IMREAD_GRAYSCALE) # pout.tif 图像
8. img_tire = cv2.imread('tire.tif', cv2.IMREAD_GRAYSCALE) # tire.tif 图像
9.
10.# 绘制原始图像的直方图
11.def plot_histogram(img, title):
12.    plt.hist(img.ravel(), 256, [0, 256])
13.    plt.title(title)
14.    plt.xlabel('像素值')
15.    plt.ylabel('次数')
16.    plt.show()
17.
18.# 绘制灰度直方图
```

```
plot_histogram(img_pout, 'pout.tif 的直方图')
plot_histogram(img_tire, 'tire.tif 的直方图')

# 显示原始图像
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(img_pout,cmap='gray')
plt.title('pout.tif 的原始图像')
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(img_tire,cmap='gray')
plt.title('tire.tif 的原始图像')
plt.axis('off')
# 灰度变换函数--直方图均衡化
def histogram_equalization(img):
    equ = cv2.equalizeHist(img)
    return equ

# 应用灰度变换
equ_pout = histogram_equalization(img_pout)  # 对pout.tif进行直方图均衡化
equ_tire = histogram_equalization(img_tire)  # 对tire.tif进行直方图均衡化

# 绘制均衡化后的直方图
plot_histogram(equ_pout, '直方图均衡化:pout.tif')
plot_histogram(equ_tire, '直方图均衡化:tire.tif')


# 显示原始和均衡化后的图像
plt.figure(figsize=(10, 5))

# pout.tif
plt.subplot(1, 4, 1)
plt.imshow(img_pout, cmap='gray')
plt.title('原始:pout.tif')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.imshow(equ_pout, cmap='gray')
plt.title('均衡化:pout.tif')
plt.axis('off')

# tire.tif
plt.subplot(1, 4, 3)
plt.imshow(img_tire, cmap='gray')
plt.title('原始:tire.tif')
```

```
plt.axis('off')

plt.subplot(1, 4, 4)
plt.imshow(equ_tire, cmap='gray')
plt.title('均衡化:tire.tif')
plt.axis('off')

plt.show()
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('pout.tif', cv2.IMREAD_GRAYSCALE)

# 分块处理函数
def blkproc(img, block_size):
    # 获取图像尺寸
    height, width = img.shape
    # 初始化输出图像
    out_img = np.zeros_like(img) # 创建一个与 img 形状相同的初始值为 0 的数组

    # 遍历图像的每个块
    for i in range(0, height, block_size):
        for j in range(0, width, block_size):
            # 获取当前块
            block = img[i:i+block_size, j:j+block_size]
            # 计算块的平均值作为亮度
            avg_intensity = np.mean(block)
            # 对块内的每个像素值进行调整
            for x in range(block_size):
                for y in range(block_size):
                    if i+x < height and j+y < width:
                        # 假设我们希望将块亮度调整到某个目标值，例如 128
                        target_intensity = 128  # 此处进行调参
                        # 计算亮度校正因子
                        correction_factor = target_intensity / avg_intens
ity
                        # 应用校正因子
                        out_img[i+x, j+y] = np.clip(img[i+x, j+y] * corre
ction_factor, 0, 255).astype(np.uint8)

    return out_img

# 定义块大小
block_size = 64 # 此处进行调参
```

```
109.
110.# 对图像进行分块处理校正
111.corrected_img = blkproc(img, block_size)
112.
113.# 显示原始和校正后的图像
114.plt.figure(figsize=(10, 5))
115.
116.plt.subplot(1, 2, 1)
117.plt.imshow(img, cmap='gray')
118.plt.title('原始图像')
119.plt.axis('off')
120.
121.plt.subplot(1, 2, 2)
122.plt.imshow(corrected_img, cmap='gray')
123.plt.title('分块处理后的图像')
124.plt.axis('off')
125.
126.plt.show()
127.plot_histogram(img,'原始图像')
128.plot_histogram(corrected_img,'分块处理后的图像')
129.import cv2
130.import numpy as np
131.import matplotlib.pyplot as plt
132.
133.# 读取图像
134.img = cv2.imread('pout.tif', cv2.IMREAD_GRAYSCALE)
135.
136.# 估计光照分布图
137.# 这里使用简单的全局和局部平均方法
138.# 实际应用中可能需要更复杂的算法，如基于物理的模型或机器学习模型
139.
140.# 全局平均亮度作为基准
141.global_mean = np.mean(img)
142.
143.# 局部平均亮度作为光照分布图
144.# 定义一个块大小
145.block_size = (64, 64)
146.
147.# 使用 cv2.blur 进行平滑处理，模拟局部平均光照
148.# 这里的模糊可以看作是光照分布图的简化估计
149.light_map = cv2.blur(img, block_size)
150.
151.# 不均匀光照校正
152.# 使用图像相减法从原始图像中减去光照分布图的一部分
153.# 这里的 alpha 是一个调整因子，用于控制校正的强度
154.alpha = 0.2 # 此处调参 0.2、0.3、0.4、0.5
```

```
corrected_img = cv2.addWeighted(img, 1 - alpha, light_map, -alpha, 0)

# 显示结果
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('原始图像')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(light_map, cmap='gray')
plt.title('预测光照分布')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(corrected_img, cmap='gray')
plt.title('不均匀光照校正后的结果')
plt.axis('off')

plt.show()
plot_histogram(img,'原始图像')
plot_histogram(corrected_img,'图像相减处理后的图像')
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
img = cv2.imread('couple.tif', cv2.IMREAD_GRAYSCALE)

# 原始图像直方图
plot_histogram(img,'原始图像')
# 三段线性变换的转折点
dark_end = 75     # 此处进行调参
mid_start = 100   # 此处进行调参
mid_end = 150     # 此处进行调参
light_start = 200 # 此处进行调参

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.title('原始图像')
plt.axis('off')

# 暗区斜率尽可能大,以便将暗区映射到亮度高的区域 这里设置为1.25
slope_dark = 1.25 # 此处进行调参
```

```
# 中区斜率直接设置为1
slope_mid = 1
# 亮区斜率尽可能小,以防止过曝或细节丢失 这里设置为0.75
slope_light = 0.75 # 此处进行调参
# 初始化输出图像
output_img = img
dark_indices = img <= dark_end
output_img[dark_indices] = (img[dark_indices]*slope_dark)
light_indices = img >= light_start
output_img[light_indices] = (img[light_indices]*slope_light)

# 确保像素值在0-255范围内
output_img = np.clip(output_img, 0, 255).astype(np.uint8)

plt.subplot(1,2,2)
plt.imshow(output_img,cmap='gray')
plt.title('处理后的图像')
plt.axis('off')

plt.show()
import cv2
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
# 读取图像
image = cv2.imread('eight.tif', cv2.IMREAD_GRAYSCALE)

# 添加高斯噪声
noisy_gaussian = np.copy(image)
mean = 0
var = 100
sigma = var ** 0.5
gauss = np.random.normal(mean, sigma, noisy_gaussian.shape)
noisy_gaussian = noisy_gaussian + gauss
noisy_gaussian = np.clip(noisy_gaussian, 0, 255).astype(np.uint8)

# 可视化高斯噪声图像
plt.subplot(2, 2, 1)
plt.imshow(noisy_gaussian, cmap='gray')
plt.title('添加高斯噪声')
plt.axis('off')

# 高斯滤波平滑处理
smoothed_gaussian = cv2.GaussianBlur(noisy_gaussian, (5, 5), 0)

```

```
# 可视化高斯滤波后的图像
plt.subplot(2, 2, 2)
plt.imshow(smoothed_gaussian, cmap='gray')
plt.title('高斯滤波处理')
plt.axis('off')

# 添加椒盐噪声
noisy_salt_pepper = np.copy(image)
noise_amount = 0.05
salt_vs_pepper = 0.5
amount = noise_amount * noisy_salt_pepper.size * salt_vs_pepper
coords = [np.random.randint(0, i - 1, int(amount)) for i in noisy_salt_pepper.shape]
noisy_salt_pepper[coords[0], coords[1]] = 255
amount = noise_amount * noisy_salt_pepper.size * (1. - salt_vs_pepper)
coords = [np.random.randint(0, i - 1, int(amount)) for i in noisy_salt_pepper.shape]
noisy_salt_pepper[coords[0], coords[1]] = 0

# 可视化椒盐噪声图像
plt.subplot(2, 2, 3)
plt.imshow(noisy_salt_pepper, cmap='gray')
plt.title('添加椒盐噪声')
plt.axis('off')

# 中值滤波平滑处理
smoothed_salt_pepper = cv2.medianBlur(noisy_salt_pepper, 5)

# 可视化中值滤波后的图像
plt.subplot(2, 2, 4)
plt.imshow(smoothed_salt_pepper, cmap='gray')
plt.title('中值滤波处理')
plt.axis('off')

# 显示所有图像
plt.tight_layout()
plt.show()
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取图像
images = ['rice.tif', 'cameraman.tif']
for image_file in images:
    image = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)
    # 确保图像读取成功
```

```
    if image is None:
        raise ValueError(f"无法读取图像文件: {image_file}")

    # 显示原始图像
    plt.figure(figsize=(12, 9))
    plt.subplot(2, 3, 1)
    plt.imshow(image, cmap='gray')
    plt.title(f'原始图像:{image_file}')
    plt.axis('off')

    # 计算罗伯茨梯度
    def roberts_gradient(image):
        gx = np.zeros_like(image)
        gy = np.zeros_like(image)

        # 罗伯茨梯度算子
        kernel_x = np.array([[1, 0], [0, -1]], dtype=np.float32)
        kernel_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

        # 应用算子
        gx = cv2.filter2D(image, -1, kernel_x)
        gy = cv2.filter2D(image, -1, kernel_y)

        # 计算梯度幅度
        gradient_magnitude = np.hypot(gx, gy)

        return gradient_magnitude

    # 罗伯茨梯度
    gradient_magnitude = roberts_gradient(image)

    # 锐化方法一：直接显示梯度幅度
    plt.subplot(2, 3, 2)
    plt.imshow(gradient_magnitude, cmap='gray')
    plt.title(f'直接显示梯度幅度:{image_file}')
    plt.axis('off')

#     print(gradient_magnitude.shape) (225,224)
#     print(gradient_magnitude.dtype)

#     # 检查NaN 值
#     nan_positions = np.isnan(gradient_magnitude)
#     print("存在NaN 值:", np.any(nan_positions))  # 如果数组中有任何NaN 值，
  np.any()将返回True 不存在NaN 值

    # 归一化梯度幅度到0-255 范围并转换为8 位无符号整数
```

```
    normalized_gradient_magnitude = cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_L1, cv2.CV_8U)

    # 锐化方法二：梯度幅度与原始图像叠加
    sharp_image1 = cv2.addWeighted(image, 0.7, normalized_gradient_magnitude, 0.3, 0)
    plt.subplot(2, 3, 3)
    plt.imshow(sharp_image1, cmap='gray')
    plt.title(f'梯度幅度与原始图像叠加:{image_file}')
    plt.axis('off')

    # 锐化方法三：对梯度幅度应用阈值
    _, thresholded = cv2.threshold(gradient_magnitude.astype('uint8'), 50, 255, cv2.THRESH_BINARY)
    sharp_image2 = cv2.bitwise_and(image, image, mask=thresholded)
    plt.subplot(2, 3, 4)
    plt.imshow(sharp_image2, cmap='gray')
    plt.title(f'对梯度幅度应用阈值:{image_file}')
    plt.axis('off')

    # 锐化方法四：梯度幅度与原始图像的非线性组合
    sharp_image3 = np.clip(image + 0.5 * gradient_magnitude, 0, 255).astype(np.uint8)
    plt.subplot(2, 3, 5)
    plt.imshow(sharp_image3, cmap='gray')
    plt.title(f'梯度幅度与原始图像的非线性组合:{image_file}')
    plt.axis('off')

    # 显示所有图像
    plt.tight_layout()
    plt.show()
import cv2
import numpy as np
import pywt
import matplotlib.pyplot as plt

# 读取图像
image = cv2.imread('slina.jpg', cv2.IMREAD_GRAYSCALE)

# 可视化原始图像
plt.figure(figsize=(12, 10))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('原始图像')
plt.axis('off')

```

```
# 选择小波类型和分解层数
wavelets = ['haar', 'db1', 'coif1']
idx = 1
for wavelet in wavelets:
    # 对图像进行小波分解
    coeffs = pywt.dwt2(image, wavelet)
    cA, (cH, cV, cD) = coeffs

    # 对水平、垂直和对角细节分量进行处理，以去除条带噪声
    # 简单的方法可以是使用阈值处理，但更高级的方法可能涉及统计建模
    threshold = 50  # 这个阈值可能需要根据具体噪声情况进行调整
    cH_denoised = np.where(np.abs(cH) > threshold, cH, 0)
    cV_denoised = np.where(np.abs(cV) > threshold, cV, 0)
    cD_denoised = np.where(np.abs(cD) > threshold, cD, 0)

    # 重构去噪后的图像
    coeffs_denoised = (cA, (cH_denoised, cV_denoised, cD_denoised))
    image_denoised = pywt.idwt2(coeffs_denoised, wavelet)
    idx += 1
    # 可视化去噪后的图像
    plt.subplot(2, 2, idx)
    plt.imshow(image_denoised, cmap='gray')
    plt.title(f'使用{wavelet}小波基函数的去噪效果')
    plt.axis('off')
# 显示所有图像
plt.show()
```