|  |  |
| --- | --- |
| 成绩 |  |

装

订

线

网络空间安全与计算机学院实验报告

实验课程名称**：数字图像处理实验**

实验项目名称： 图像的增强

学生姓名： 刘新媛

专 业： 人工智能

学 号： 20221205037

实验地点： C1-128

实验时间： 2024.12.11 第3-4节

指导教师： 杨文柱

实验报告部分

1. **实验内容与要求**

**（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。

**（二）实验内容和实验原理**

1．利用opencv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。

**（三）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，采用分块处理函数和图像相减函数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.png。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.png、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。

**（四）实验报告要求**

1．描述图像增强的基本原理；

2．列出上述图像处理的程序；

3．记录实验结果的图像；

4．撰写心得体会。

1. **实验数据与实验步骤**
2. **实验数据**

pout.tif rice.png

**pout **

cameraman.tif couple.png

**Cameraman **

eight.tif tire.tif

**eight tire**

1. **实验步骤**

**图像增强的原理：**图像增强是指通过各种处理方法改善图像质量，使图像的视觉效果或后续处理更为有效。目的是突出图像中的某些特征，例如边缘、细节或对比度，或通过去噪、平滑等手段消除不必要的信息。

* 增强图像的亮度和对比度：通过调整图像的亮度和对比度来增强视觉效果。亮度调整是对整个图像像素值进行加权或平移，而对比度增强则是通过扩大像素值的差异来使图像更具层次感。
* 锐化：锐化处理通过增强图像的边缘或细节，使图像看起来更加清晰。通常通过高通滤波器去除低频成分（如平滑区域），留下高频成分（如边缘和细节）。
* 去噪：图像在采集、传输或处理过程中往往会受到噪声干扰。去噪的目的是消除或减少图像中的噪声，恢复图像的真实内容。常见的噪声类型包括高斯噪声、椒盐噪声等。常见方法有均值滤波、中值滤波、高斯滤波、双边滤波。
* 边缘增强：边缘增强方法通过检测图像中的边缘并增强其对比度，使边缘更加明显，从而突显物体轮廓。边缘增强通常基于梯度或拉普拉斯算子来提取边缘信息。

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。大致步骤如下所示。

|  |
| --- |
| **图像灰度修正** |
| * 图像读取: 使用 cv2.imread 函数读取灰度图像。 |
| * 直方图绘制: 使用 matplotlib 的 hist 函数分析图像的灰度分布。 |
| * 灰度变换: 调整图像的亮度和对比度，参数 alpha 控制对比度，beta 控制亮度。灰度变换表达式如公式1所示。   **（1）** |
| * 直方图均衡化: 使用 cv2.equalizeHist 实现图像直方图均衡化。 |

通过 apply\_gray\_transform 函数实现了灰度修正，可以调整对比度 (alpha) 和亮度 (beta) 的参数。对于测试图像 pout.tif 和 tire.tif，可以分别调整参数，使图像的灰度分布更为合理。alpha 增大，图像对比度增强。beta 增大，图像整体亮度提升。

使用 plot\_histogram 绘制图像直方图，展示像素强度的分布情况。直方图中均匀的像素分布代表灰度级分布协调；分布过于集中则需要灰度变换进行优化。

|  |
| --- |
| **图像灰度修正代码：** |
| '''  @Function:图像灰度修正。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。  @Author: 刘新媛  @Date: 2024/12/11  '''  **import** cv2  **import** numpy as np  **import** matplotlib.pyplot as plt    **def** plot\_histogram(image, title, ax):      """      绘制图像的直方图。      :param image: 输入图像（灰度图）      :param title: 直方图标题      :param ax: Matplotlib的轴对象，用于绘制直方图      """      # 使用像素值范围 [0, 256] 绘制直方图，ravel 将图像拉平成一维数组      ax.hist(image.ravel(), bins**=**256, range**=**(0, 256), color**=**'blue', alpha**=**0.7)      ax.set\_title(title)  # 标题      ax.set\_xlabel("Pixel Intensity")  # 横轴标签：像素强度      ax.set\_ylabel("Frequency")  # 纵轴标签：频率    **def** apply\_gray\_transform(image, alpha**=**1.0, beta**=**0):      """      应用线性灰度变换，用于调整图像对比度和亮度。      :param image: 输入图像（灰度图）      :param alpha: 对比度调整系数（默认值为1.0）      :param beta: 亮度调整系数（默认值为0）      :return: 调整后的图像      """      # convertScaleAbs函数进行线性变换：output = alpha \* input + beta      adjusted **=** cv2.convertScaleAbs(image, alpha**=**alpha, beta**=**beta)  **return** adjusted    **def** histogram\_equalization(image):      """      对图像执行直方图均衡化。      :param image: 输入图像（灰度图）      :return: 均衡化后的图像      """      # equalizeHist 实现直方图均衡化  **return** cv2.equalizeHist(image)    images **=** ["pout.tif", "tire.tif"]    # 遍历每个测试图像  **for** img\_path **in** images:      # 读取灰度图像      image **=** cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)      # 显示原始图像及其直方图      fig, axs **=** plt.subplots(1, 3, figsize**=**(15, 5))  # 创建包含三个子图的图形      axs[0].imshow(image, cmap**=**'gray')  # 在第一个子图中显示原始图像      axs[0].set\_title("Original Image")  # 设置标题      axs[0].axis('off')  # 隐藏坐标轴      plot\_histogram(image, "Original Histogram", axs[1])  # 在第二个子图中绘制原始图像直方图      # 应用灰度变换调整对比度和亮度      adjusted\_image **=** apply\_gray\_transform(image, alpha**=**1.2, beta**=**15)  # 设置alpha和beta的值      plot\_histogram(adjusted\_image, "Adjusted Histogram", axs[2])  # 在第三个子图中绘制调整后的直方图      # 显示第一组结果      plt.show()        # 对原始图像进行直方图均衡化      equalized\_image **=** histogram\_equalization(image)      # 显示原始图像、灰度调整后的图像和均衡化后的图像      fig, axs **=** plt.subplots(1, 3, figsize**=**(15, 5))  # 创建包含三个子图的图形      axs[0].imshow(image, cmap**=**'gray')  # 在第一个子图中显示原始图像      axs[0].set\_title("Original Image")  # 设置标题      axs[0].axis('off')  # 隐藏坐标轴      axs[1].imshow(adjusted\_image, cmap**=**'gray')  # 在第二个子图中显示调整后的图像      axs[1].set\_title("Adjusted Image")  # 设置标题      axs[1].axis('off')  # 隐藏坐标轴      axs[2].imshow(equalized\_image, cmap**=**'gray')  # 在第三个子图中显示均衡化后的图像      axs[2].set\_title("Equalized Image")  # 设置标题      axs[2].axis('off')  # 隐藏坐标轴      # 显示第二组结果      plt.show()        # 绘制均衡化后的直方图及对应的图像      fig, axs **=** plt.subplots(1, 2, figsize**=**(10, 5))  # 创建包含两个子图的图形      plot\_histogram(equalized\_image, "Equalized Histogram", axs[0])  # 在第一个子图中绘制均衡化后的直方图      axs[1].imshow(equalized\_image, cmap**=**'gray')  # 在第二个子图中显示均衡化后的图像      axs[1].set\_title("Equalized Image")  # 设置标题      axs[1].axis('off')  # 隐藏坐标轴      # 显示第三组结果      plt.show() |

2．不均匀光照的校正。测试图像pout.tif，采用分块处理函数和图像相减函数对图像不均匀光照进行校正，大致步骤如下所示：

|  |
| --- |
| **不均匀光照的校正** |
| * **计算局部均值图：**将图像划分为固定大小的块，计算每个块的平均值，形成局部均值图。局部均值图模拟了图像中光照的不均匀分布。 * **校正不均匀光照：**将原始图像减去局部均值图，从而去除低频的光照分量。原始图像减局部均值图消除光照的影响。将减法结果通过加权平衡，避免图像过暗。使用 cv2.addWeighted 实现加权操作。 * **调参优化**：块大小（block\_size）控制局部均值的计算范围，亮度偏移量来调整图像整体亮度。 |

|  |
| --- |
| **不均匀光照的校正代码：** |
| '''  @Function:不均匀光照的校正。采用分块处理函数和图像相减函数对图像不均匀光照进行校正  @Author: 刘新媛  @Date: 2024/12/11  '''  **import** cv2  **import** numpy as np  **import** matplotlib.pyplot as plt    **def** correct\_uneven\_illumination(image, block\_size**=**32):      """      使用分块处理和图像相减函数对图像进行不均匀光照校正。      :param image: 输入图像（灰度图）      :param block\_size: 分块大小（默认32）      :return: 校正后的图像      """      # 创建局部均值图（通过分块处理计算图像的局部平均值）      kernel **=** (block\_size, block\_size)      local\_mean **=** cv2.blur(image, kernel)        # 校正图像：原始图像减去局部均值图，再通过线性变换调整亮度范围      corrected **=** cv2.addWeighted(image, 1.0, local\_mean, **-**1.0, 128)  **return** corrected      # 测试图像路径  img\_path **=** "pout.tif"    # 读取灰度图像  image **=** cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)  **if** image **is** None:      print(f"Cannot read image {img\_path}")  **else**:      # 校正图像      corrected\_image **=** correct\_uneven\_illumination(image, block\_size**=**32)        # 显示原始图像和校正后的图像      fig, axs **=** plt.subplots(1, 3, figsize**=**(15, 5))        axs[0].imshow(image, cmap**=**'gray')      axs[0].set\_title("Original Image")      axs[0].axis('off')        axs[1].imshow(corrected\_image, cmap**=**'gray')      axs[1].set\_title("Corrected Image")      axs[1].axis('off')        # 绘制校正后图像的直方图      axs[2].hist(corrected\_image.ravel(), bins**=**256, range**=**(0, 256), color**=**'blue', alpha**=**0.7)      axs[2].set\_title("Histogram of Corrected Image")      axs[2].set\_xlabel("Pixel Intensity")      axs[2].set\_ylabel("Frequency")        plt.tight\_layout()      plt.show() |

3.三段线性变换增强。测试图像couple.png。选择合适的转折点，编程对图像进行三段线性变换增强。大致步骤如下：

|  |
| --- |
| **三段线性变换增强** |
| * **设置转折点**：确定两个转折点 p1和 p2，将灰度值范围划分为三个区间：   [0,p1]对低灰度值进行拉伸或压缩、[p1,p2]对中间灰度值线性变换、[p2,255]对高灰度值进行拉伸或压缩。   * **实现三段线性变换**：对每个灰度区间按照公式2进行变换。   **（2）** |

|  |
| --- |
| **三段线性变换增强代码：** |
| '''  @Function:选择合适的转折点，对图像进行三段线性变换增强。  @Author: 刘新媛  @Date: 2024/12/11  '''  **import** cv2  **import** numpy as np  **import** matplotlib.pyplot as plt    **def** plot\_histogram(image, title):      """绘制图像直方图"""      plt.hist(image.ravel(), bins**=**256, range**=**(0, 256), color**=**'blue', alpha**=**0.7)      plt.title(title)      plt.xlabel("Pixel Intensity")      plt.ylabel("Frequency")      **def** linear\_transform(image, p1, p2):      """三段线性变换增强"""      output **=** np.zeros\_like(image, dtype**=**np.uint8)        # 低灰度区间      alpha1 **=** 255 **/** p1      beta1 **=** 0      mask1 **=** (image <**=** p1)      output[mask1] **=** (alpha1 **\*** image[mask1] **+** beta1).astype(np.uint8)        # 中灰度区间      alpha2 **=** 255 **/** (p2 **-** p1)      beta2 **=** **-**alpha2 **\*** p1      mask2 **=** (image > p1) & (image <**=** p2)      output[mask2] **=** (alpha2 **\*** image[mask2] **+** beta2).astype(np.uint8)        # 高灰度区间      alpha3 **=** 255 **/** (255 **-** p2)      beta3 **=** **-**alpha3 **\*** p2 **+** 255      mask3 **=** (image > p2)      output[mask3] **=** (alpha3 **\*** image[mask3] **+** beta3).astype(np.uint8)    **return** output      # 读取图像  image\_path **=** "couple.png"  image **=** cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    **if** image **is** None:      print(f"Cannot read image {image\_path}")  **else**:      # 显示原始图像及其直方图      plt.figure(figsize**=**(10, 5))      plt.subplot(1, 2, 1)      plt.imshow(image, cmap**=**'gray')      plt.title("Original Image")      plt.axis('off')      plt.subplot(1, 2, 2)      plot\_histogram(image, "Original Histogram")      plt.show()        # 设置转折点      p1, p2 **=** 85, 170        # 进行三段线性变换      enhanced\_image **=** linear\_transform(image, p1, p2)        # 显示增强后的图像及其直方图      plt.figure(figsize**=**(10, 5))      plt.subplot(1, 2, 1)      plt.imshow(enhanced\_image, cmap**=**'gray')      plt.title("Enhanced Image")      plt.axis('off')      plt.subplot(1, 2, 2)      plot\_histogram(enhanced\_image, "Enhanced Histogram")      plt.show() |

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。大致步骤如下：

|  |
| --- |
| **图像平滑：** |
| * **添加噪声：**add\_noise函数，通过不同类型的噪声（高斯噪声或椒盐噪声）向原始图像添加噪声。高斯噪声生成正态分布的随机值并添加到原图像中。椒盐噪声随机选择像素点，并将其值设置为最大值（255，盐噪声）或最小值（0，胡椒噪声），根据噪声类型分别生成加噪后的图像。 * **选择去噪方法：**smooth\_image函数，应用不同的平滑滤波方法来去除噪声。 * 均值滤波（mean）：使用cv2.blur()函数进行均值滤波。 * 中值滤波（median）：使用cv2.medianBlur()函数进行中值滤波，适用于去除椒盐噪声。 * 高斯滤波（gaussian）：使用cv2.GaussianBlur()函数进行高斯滤波，适用于去除高斯噪声。 * 双边滤波（bilateral）：使用cv2.bilateralFilter()函数进行双边滤波，能够在去噪的同时保留边缘信息。 * **应用去噪方法：**对加噪图像应用相应的平滑滤波方法。 |

|  |
| --- |
| **图像平滑代码:** |
| '''  @Function: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。  @Author: 刘新媛  @Date: 2024/12/11  '''  **import** cv2  **import** numpy as np  **import** matplotlib.pyplot as plt    # 加载原始图像  image **=** cv2.imread('eight.tif', cv2.IMREAD\_GRAYSCALE)    # 添加噪声  **def** add\_noise(image, noise\_type**=**"gaussian"):      noisy\_image **=** image.copy()  **if** noise\_type **==** "gaussian":          mean **=** 0          sigma **=** 25          gaussian\_noise **=** np.random.normal(mean, sigma, image.shape)          noisy\_image **=** np.clip(image **+** gaussian\_noise, 0, 255)  **elif** noise\_type **==** "salt\_pepper":          s\_vs\_p **=** 0.5          amount **=** 0.02          noisy\_image **=** image.copy()          # Salt noise          num\_salt **=** int(np.ceil(amount **\*** image.size **\*** s\_vs\_p))          salt\_coords **=** [np.random.randint(0, i**-**1, num\_salt) **for** i **in** image.shape]          noisy\_image[salt\_coords[0], salt\_coords[1]] **=** 255          # Pepper noise          num\_pepper **=** int(np.ceil(amount **\*** image.size **\*** (1.0 **-** s\_vs\_p)))          pepper\_coords **=** [np.random.randint(0, i**-**1, num\_pepper) **for** i **in** image.shape]          noisy\_image[pepper\_coords[0], pepper\_coords[1]] **=** 0  **return** noisy\_image    # 生成加噪图像  noisy\_image\_gaussian **=** add\_noise(image, noise\_type**=**"gaussian")  noisy\_image\_sp **=** add\_noise(image, noise\_type**=**"salt\_pepper")    # 应用平滑滤波处理  **def** smooth\_image(image, method**=**"gaussian"):  **if** method **==** "mean":  **return** cv2.blur(image, (5, 5))  # 均值滤波  **elif** method **==** "median":  **return** cv2.medianBlur(image, 5)  # 中值滤波  **elif** method **==** "gaussian":  **return** cv2.GaussianBlur(image, (5, 5), 0)  # 高斯滤波  **elif** method **==** "bilateral":  **return** cv2.bilateralFilter(image, 9, 75, 75)  # 双边滤波    # 针对不同噪声选择合适的去噪方法  smoothed\_gaussian **=** smooth\_image(noisy\_image\_gaussian, method**=**"gaussian")  smoothed\_sp **=** smooth\_image(noisy\_image\_sp, method**=**"median")    # 显示结果  plt.figure(figsize**=**(12, 8))    # 显示加噪后的图像  plt.subplot(2, 3, 1)  plt.imshow(noisy\_image\_gaussian, cmap**=**'gray')  plt.title('Gaussian Noise')  plt.axis('off')    plt.subplot(2, 3, 2)  plt.imshow(noisy\_image\_sp, cmap**=**'gray')  plt.title('Salt & Pepper Noise')  plt.axis('off')    # 显示平滑后的图像  plt.subplot(2, 3, 4)  plt.imshow(smoothed\_gaussian, cmap**=**'gray')  plt.title('Smoothed with Gaussian Filter')  plt.axis('off')    plt.subplot(2, 3, 5)  plt.imshow(smoothed\_sp, cmap**=**'gray')  plt.title('Smoothed with Median Filter')  plt.axis('off')    plt.tight\_layout()  plt.show() |

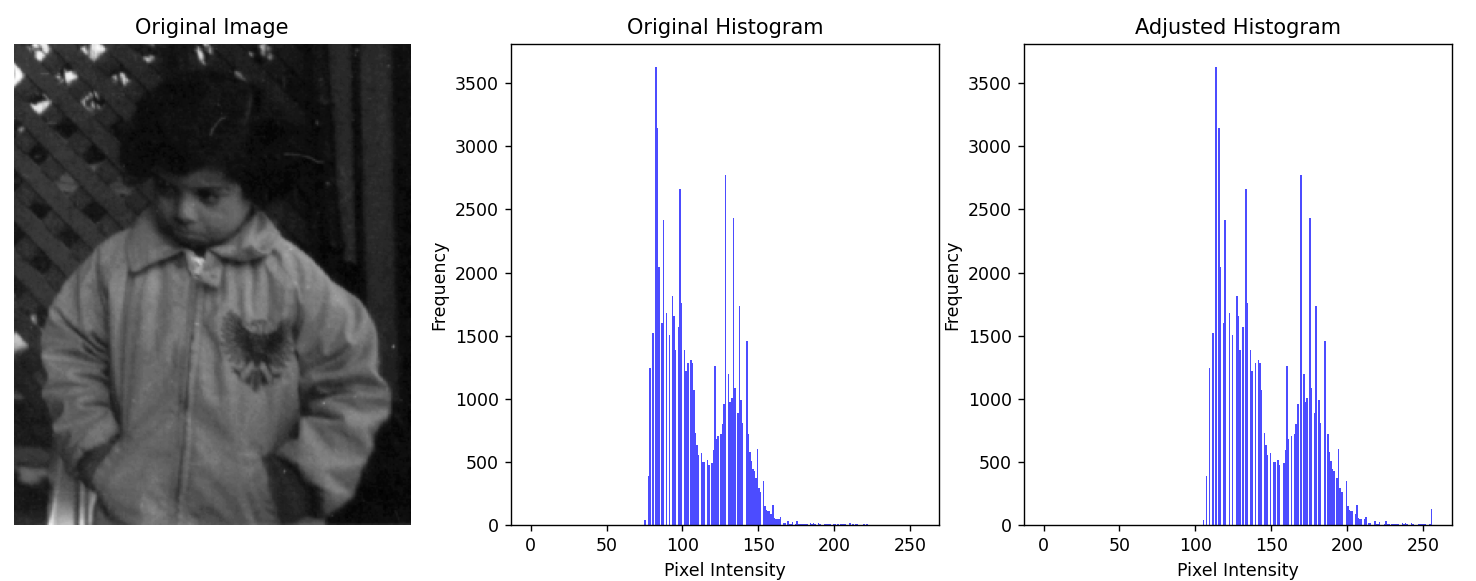
1. 图像锐化方法。测试图像为rice.png、cameraman.tif。读入一副边缘模糊的图像，利用罗伯茨梯度对图像进行4种锐化处理，比较各自的效果。大致步骤如下：

|  |
| --- |
| **图像锐化：** |
| * 罗伯茨梯度计算：建罗伯茨梯度算子（分别为水平和垂直方向），使用cv2.filter2D()对图像进行卷积操作，计算图像的梯度，得到边缘信息。使用cv2.magnitude()计算水平和垂直梯度的幅度来获得边缘图。 * **锐化方法**：经典锐化（直接将边缘图像与原图进行叠加，增强图像的边缘）；增强边缘对比（将梯度幅度增加，使边缘更加突出）；高通滤波锐化（通过从原图中减去低通滤波（高斯模糊）得到高频部分，然后与原图结合进行锐化）；自定义锐化（结合图像的原图和边缘信息来进行锐化，控制锐化的程度）。 |

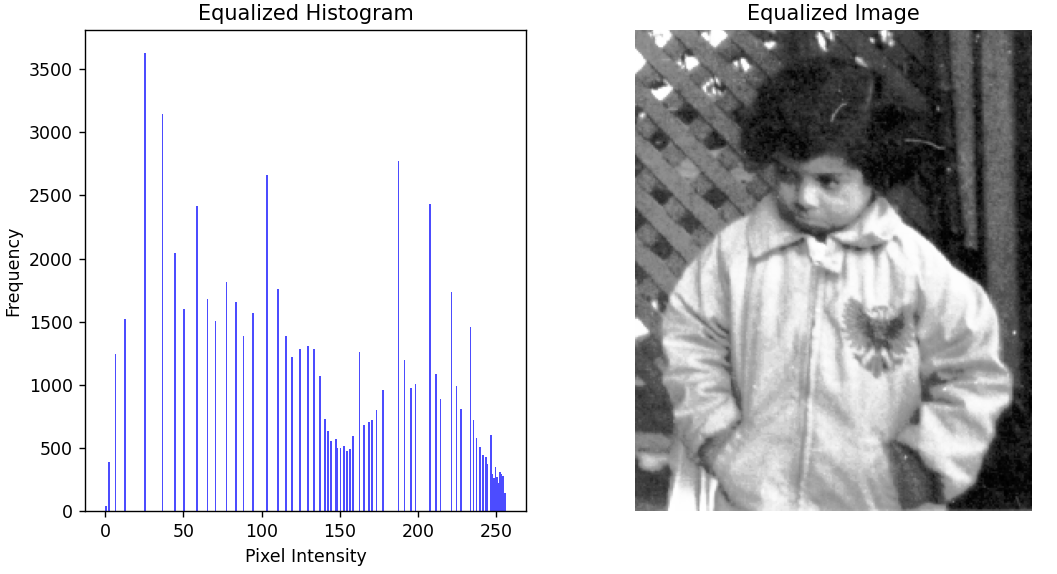
|  |
| --- |
| **图像锐化代码：** |
| **import** cv2  **import** numpy as np  **import** matplotlib.pyplot as plt    # 读取图像  image\_rice **=** cv2.imread('rice.png', cv2.IMREAD\_GRAYSCALE)  image\_cameraman **=** cv2.imread('cameraman.tif', cv2.IMREAD\_GRAYSCALE)      # 罗伯茨梯度算子  **def** roberts\_gradient(image):      # 创建罗伯茨梯度算子（水平和垂直）      kernel\_x **=** np.array([[1, 0], [0, **-**1]], dtype**=**np.float32)      kernel\_y **=** np.array([[0, 1], [**-**1, 0]], dtype**=**np.float32)        # 使用卷积计算梯度      grad\_x **=** cv2.filter2D(image.astype(np.float32), **-**1, kernel\_x)      grad\_y **=** cv2.filter2D(image.astype(np.float32), **-**1, kernel\_y)        # 计算梯度幅值      gradient **=** cv2.magnitude(grad\_x, grad\_y)  **return** gradient      # 锐化处理方法  **def** sharpen\_image(image, method**=**"classic"):  **if** method **==** "classic":          # 经典锐化：图像加上边缘检测结果          gradient **=** roberts\_gradient(image)          sharpened **=** cv2.add(image.astype(np.uint8), gradient.astype(np.uint8))  **return** sharpened    **elif** method **==** "enhance\_edges":          # 增强边缘对比：加大梯度幅度，突出边缘          gradient **=** roberts\_gradient(image)          gradient **=** np.clip(gradient **\*** 2, 0, 255)  # 增强幅度          sharpened **=** cv2.add(image.astype(np.uint8), gradient.astype(np.uint8))  **return** sharpened    **elif** method **==** "high\_pass\_filter":          # 高通滤波锐化：用原图减去低通滤波结果得到高频          blurred **=** cv2.GaussianBlur(image, (5, 5), 0)          high\_pass **=** image **-** blurred          sharpened **=** cv2.add(image.astype(np.uint8), high\_pass.astype(np.uint8))  **return** sharpened    **elif** method **==** "custom\_sharpen":          # 自定义锐化：结合边缘和原图来进行锐化          gradient **=** roberts\_gradient(image)          sharpened **=** image **+** 0.5 **\*** gradient.astype(np.uint8)          sharpened **=** np.clip(sharpened, 0, 255)  # 防止溢出  **return** sharpened      # 对两张图像进行锐化处理  methods **=** ["classic", "enhance\_edges", "high\_pass\_filter", "custom\_sharpen"]  results\_rice **=** [sharpen\_image(image\_rice, method) **for** method **in** methods]  results\_cameraman **=** [sharpen\_image(image\_cameraman, method) **for** method **in** methods]    # 显示结果  fig, axes **=** plt.subplots(2, 5, figsize**=**(15, 8))    # 显示原图像  axes[0, 0].imshow(image\_rice, cmap**=**'gray')  axes[0, 0].set\_title('Rice Original')  axes[0, 0].axis('off')    axes[1, 0].imshow(image\_cameraman, cmap**=**'gray')  axes[1, 0].set\_title('Cameraman Original')  axes[1, 0].axis('off')    # 显示四种锐化结果  **for** i, method **in** enumerate(methods):      axes[0, i **+** 1].imshow(results\_rice[i], cmap**=**'gray')      axes[0, i **+** 1].set\_title(f'Rice {method}')      axes[0, i **+** 1].axis('off')        axes[1, i **+** 1].imshow(results\_cameraman[i], cmap**=**'gray')      axes[1, i **+** 1].set\_title(f'Cameraman {method}')      axes[1, i **+** 1].axis('off')    plt.tight\_layout()  plt.show() |

1. **实验结果及分析**
2. **图像灰度修正：**

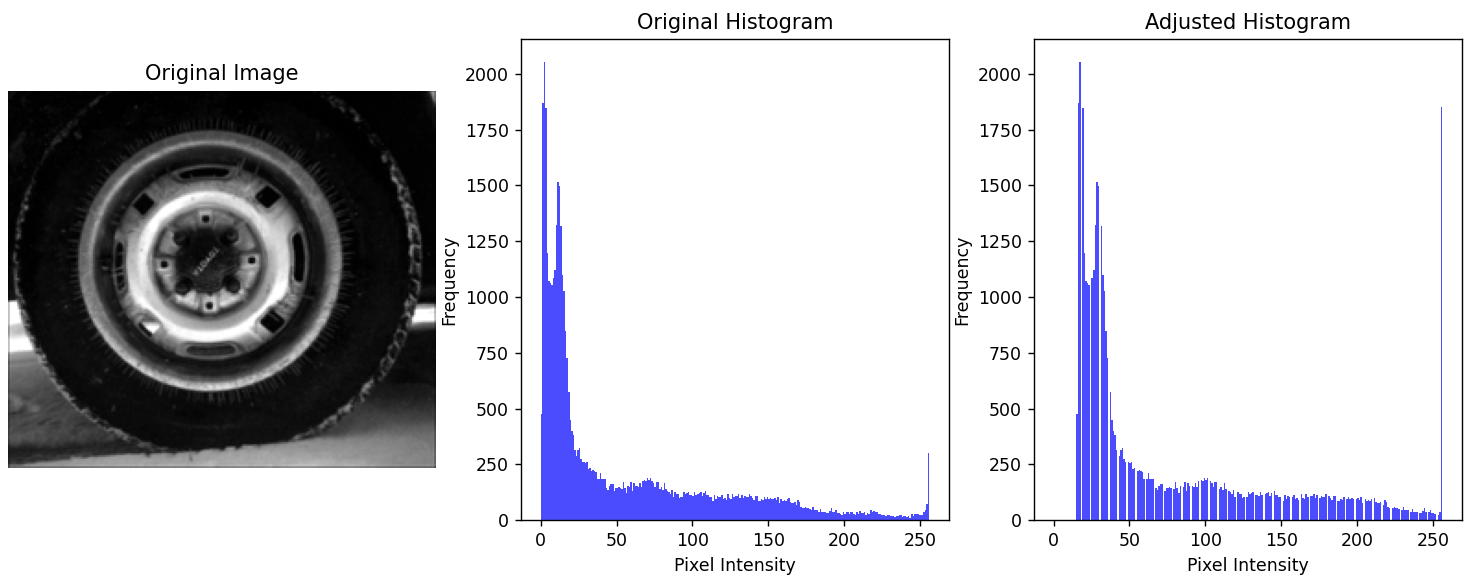
* **pout.tif:**

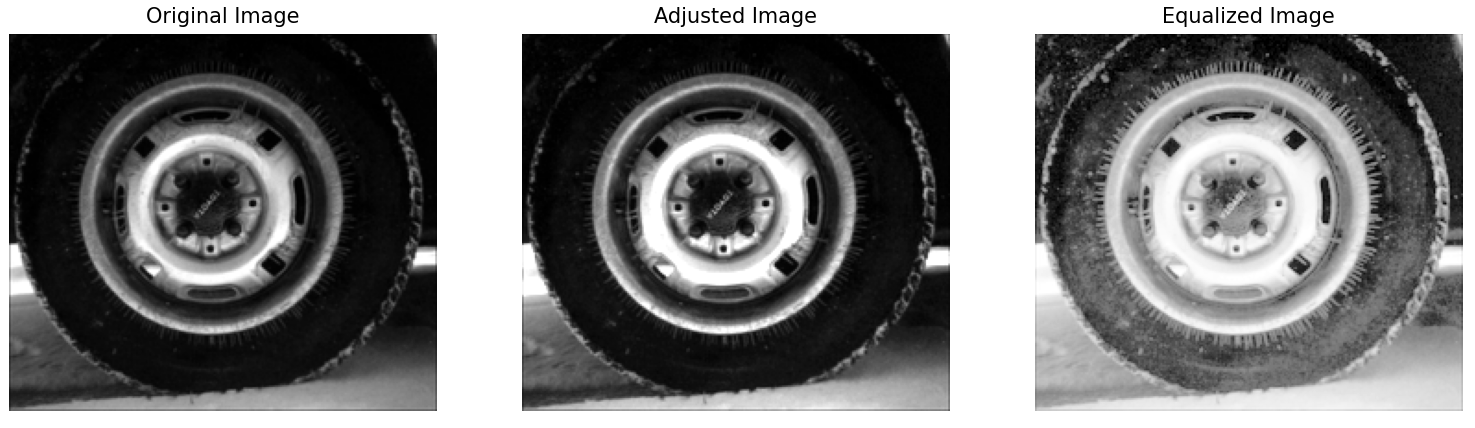


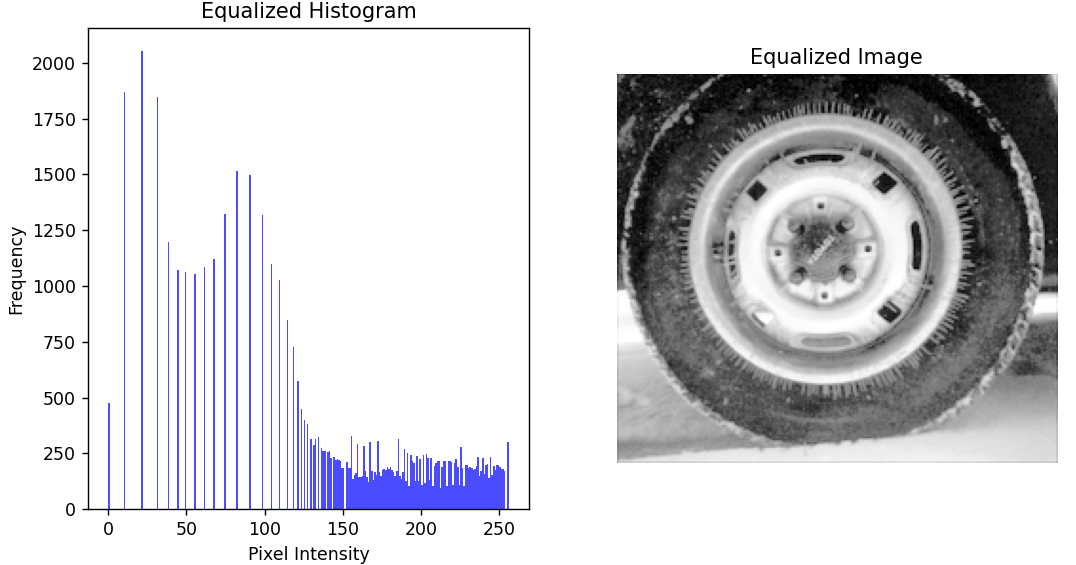




* **tire.tif:**

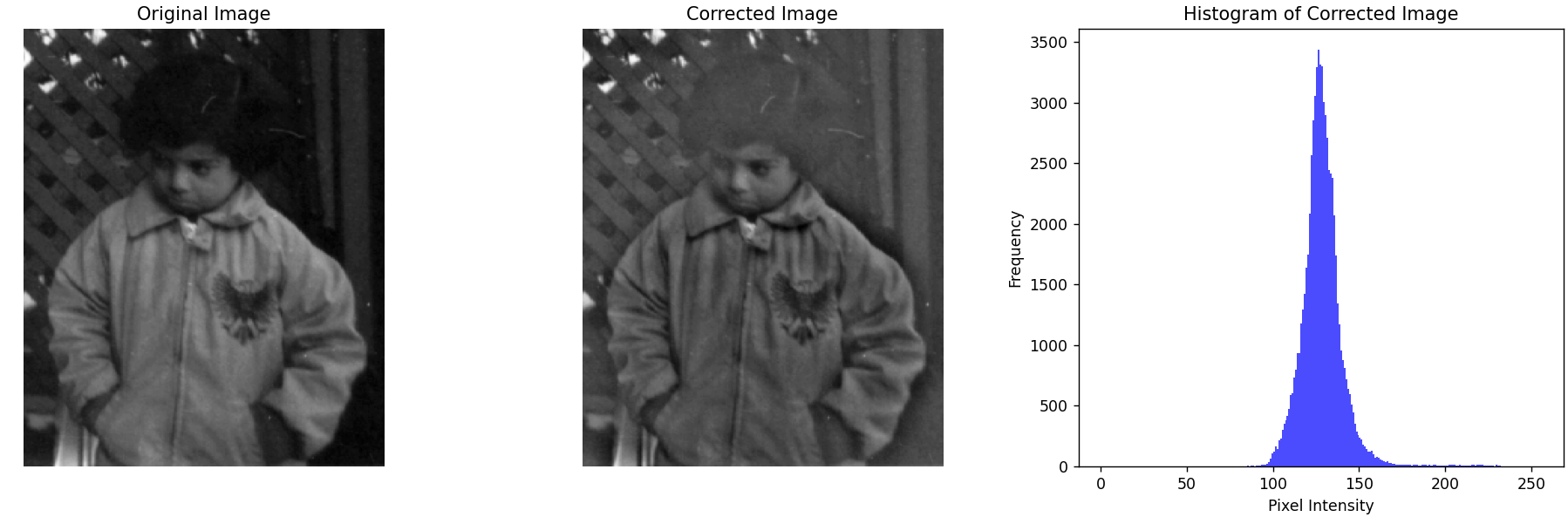






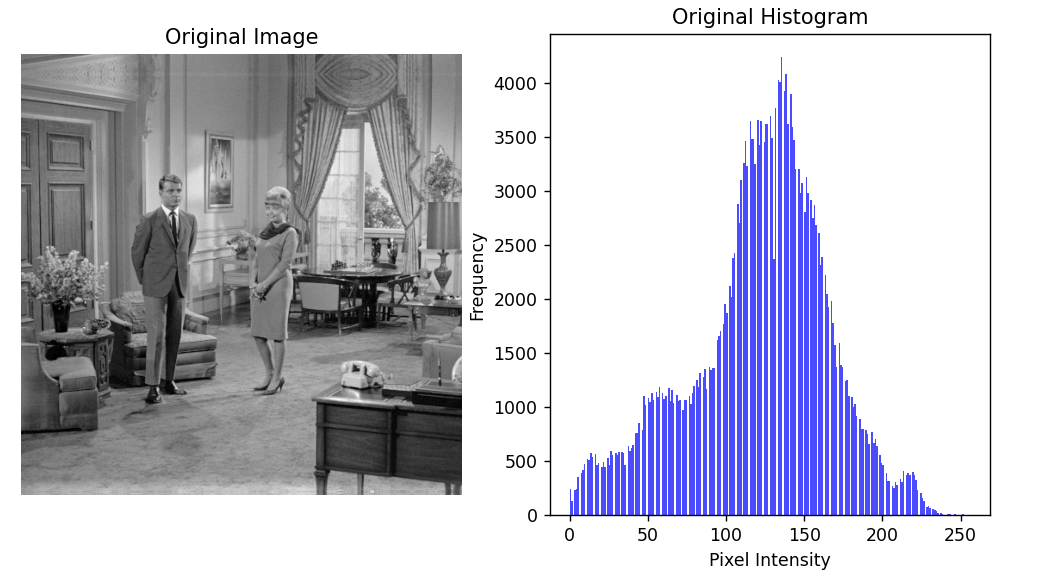
【分析】原始图像直方图偏集中不均衡，经过灰度修正，调整 alpha=1.2 和 beta=15，使原始图像的灰度级分布更加均匀。均衡化后的直方图更加均匀，显示图像的对比度显著改善。

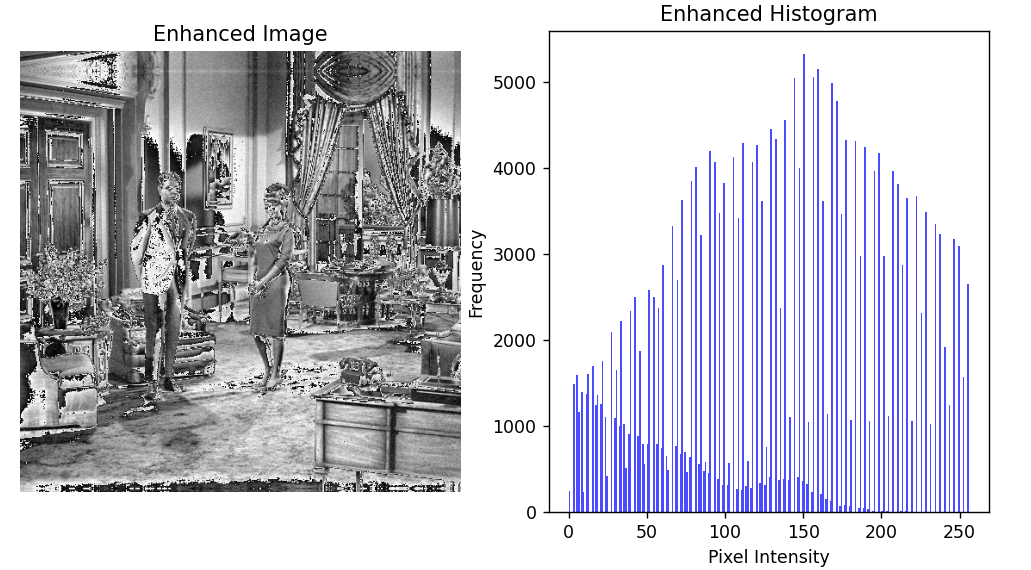
1. **不均匀光照校正：**



【分析】左图：原始图像有明显的光照不均现象，左上角较亮，其他地方相对较暗；校正后图像，不均匀光照被有效移除，整体光照均匀；校正后图像的直方图，像素分布也更均匀。

1. **三段线性变换增强**

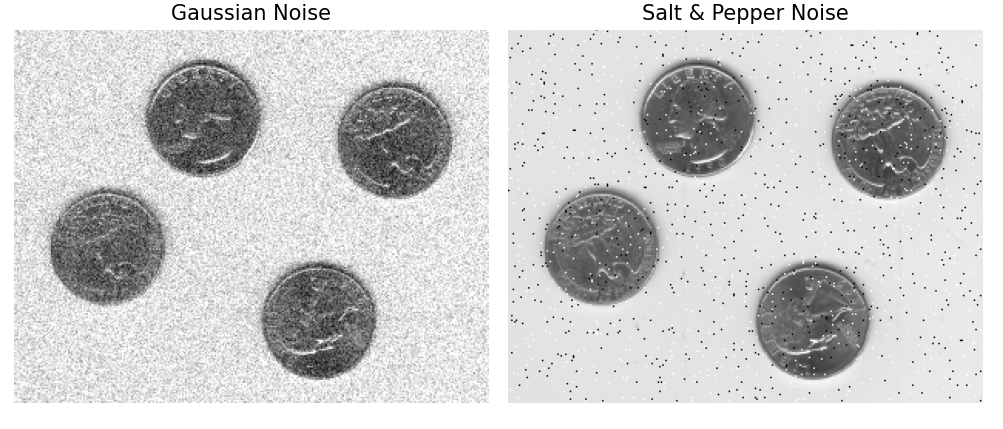


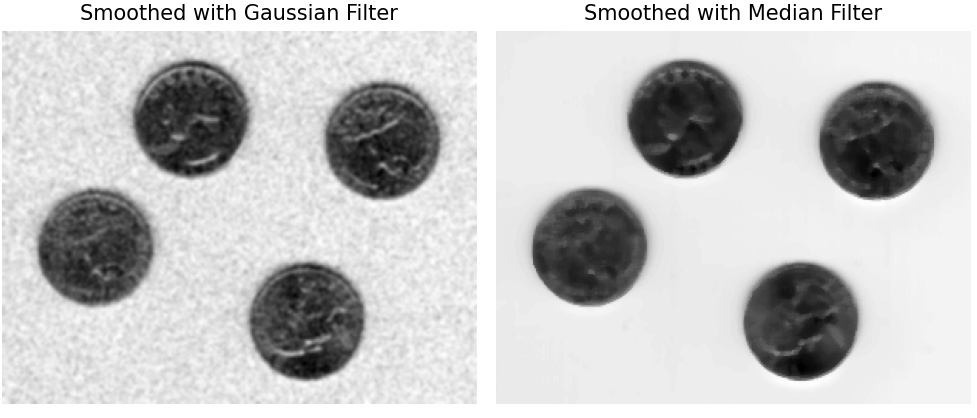


【分析】原始图像亮度和对比度较低，细节表现不佳。直方图分布较为集中，像素值未充分利用整个灰度范围，视觉效果偏灰暗。

增强图像的直方图显示像素值更加分散，灰度范围得到了充分利用，整个图像更清晰且对比度更高。通过三段线性变换增强处理了测试图像 couple.png，实现了对图像的细节增强。

1. **图像平滑**





【分析】对于高斯噪声，高斯滤波成功去除了噪声，但图像的整体锐度可能稍微降低。对于椒盐噪声，中值滤波表现出色，几乎能完全去除噪点，同时保留了边缘和细节，使得图像恢复了较高的质量。

1. **图像锐化**



【分析】 经典锐化和增强边缘方法效果最为显著，边缘更加突出，但有时会产生噪点或过度锐化的现象。高通滤波锐化方法能平滑图像的低频成分，使边缘更加清晰，但可能导致部分细节丢失。自定义锐化则是一种相对平衡的锐化方法，在增强细节的同时，避免过度锐化，可以保留较多的细节。

1. **总结及心得体会**

这次实验玩了一把图像增强的魔术，让那些看起来灰蒙蒙、不太清楚的图片变得清晰、生动。调整图片的明暗和对比度：就像是给图片调色一样，通过改变图片的灰度，让那些看起来太暗或者太亮的地方变得刚刚好。搞定光照不均匀的问题：有时候，图片的某些部分会因为光线的原因显得特别亮或者特别暗。用分块处理，让这些不均匀的光照变得均匀，图片看起来就自然多了。图片的三段线性变换：这个有点像是给图片的亮度和对比度做微调，还有平滑处理，去掉噪点、锐化处理。

这次实验还大大提升了我的编程能力和解决问题的本事。从运用函数到搭建逻辑框架，再到解决那些让人头疼的问题，每一步都不容易，但每一次突破都让我更强大，调试能力也越来越强。