22336216_陶宇卓

实验一

要求

编写代码计算图像的直方图并实现直方图均衡化，包括原始图像、其直方图、均衡转换函数、增强后的图像及其直方图的可视化展示。

原理

直方图均衡化的基本思路是将图像的灰度分布进行“拉平”或“均衡化”，使得不同灰度级的像素数趋于均匀分布。这增加了图像的对比度，尤其是对于灰度集中在特定范围的图像效果更为显著。

原始图像的灰度级在区间 [0,L−1] 之间变化（L=256），我们定义：

- r 为原图像的灰度值（取值范围为 [0,L−1]）。
- T(r) 为变换函数，通过该函数，将原灰度 r 映射到均衡化后的灰度值 s。

直方图均衡化的变换函数可以通过累计分布函数（CDF）来实现：

$$s = T(r) = (L-1)\int_0^r p_r(w)\,dw$$

其中， $p_r(w)$ 是灰度级 w 的概率密度函数，也就是灰度值为 w 的像素数占总像素数的比例。

在离散的图像处理中，累计分布函数（CDF）可以表示为：

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

其中， $p_r(r_j) = \frac{n_j}{MN}$ ， n_j 是灰度级为 r_j 的像素数量，M 和 N 分别为图像的宽度和高度。

因此，均衡化步骤如下：

1. 计算原始图像的直方图，即每个灰度级的像素数量。
2. 计算原始图像的累计分布函数（CDF）。
3. 归一化 CDF，将其映射到新的灰度值范围。

4. 根据归一化的 CDF，将原图像的每个像素灰度值替换为新的灰度值。

具体实现

Python

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

font = {'family': 'MicroSoft YaHei',
        'weight': 'bold',
        'size': 'larger'}
(plt.rc("font", family='MicroSoft YaHei', weight="bold"))


def compute_histogram(image):
    # 计算灰度直方图
    hist = np.zeros(256)
    for pixel in image.ravel():
        hist[pixel] += 1
    hist = hist / hist.sum()
    return hist


def histogram_equalization(image):
    # 直方图均衡化
    hist = compute_histogram(image)
    cdf = hist.cumsum()  # 累积分布函数
    cdf_normalized = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
    cdf_normalized = cdf_normalized.astype('uint8')

    equalized_image = cdf_normalized[image]
    return equalized_image, cdf_normalized


# 加载图像a
image_a = Image.open("a.jpg").convert("L")
image_a = np.array(image_a)

# 直方图均衡化
equalized_image, equalized_cdf = histogram_equalization(image_a)

# 绘制结果
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
axs[0, 0].imshow(image_a, cmap='gray')
axs[0, 0].set_title('原始图像')

original_hist = compute_histogram(image_a)
axs[0, 1].bar(range(256), original_hist)
axs[0, 1].set_title('图像直方图')
```

```
46  axs[0, 1].set_ylim(0, 0.6)
47  axs[0, 1].set_xticks([0, 63, 127, 191, 255])
48
49  axs[0, 2].plot(equalized_cdf, color='red')
50  axs[0, 2].set_title('直方图均衡变换')
51
52  equalized_hist = compute_histogram(equalized_image)
53  axs[1, 0].imshow(equalized_image, cmap='gray')
54  axs[1, 0].set_title('直方图均衡化后的图像')
55
56  axs[1, 1].bar(range(256), equalized_hist)
57  axs[1, 1].set_title('均衡化后的图像直方图')
58  axs[1, 1].set_xticks([0, 63, 127, 191, 255])
59  axs[1, 1].set_yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
60
61  axs[1, 2].axis('off')
62
63  plt.show()
64
```

结果分析

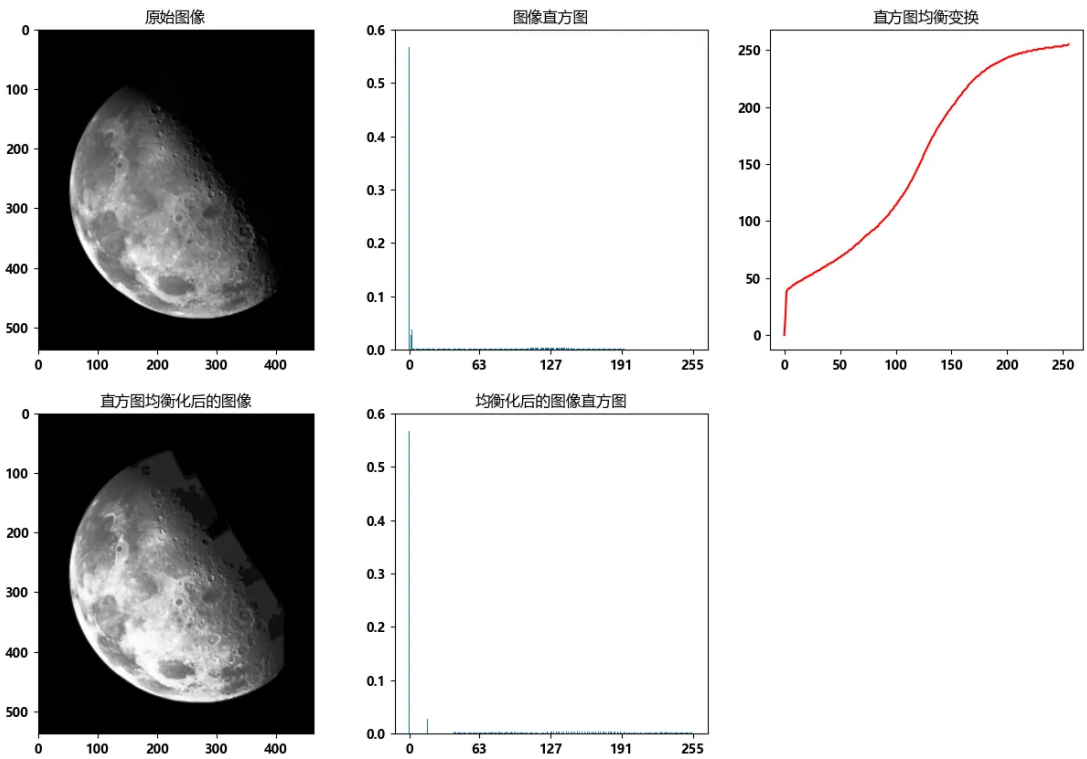

Figure_1

直方图分析

- **低灰度值的高峰**：灰度值为 0 的位置有一个非常高的峰值。这是因为超过一半的像素都是黑色，所以对应灰度值 0 的频率会远高于其他灰度值。
- **灰度值较低的区域密集分布**：除了灰度值为 0 的高峰外，其他较低灰度值（1 到 50 之间）也可能有一定的像素分布。如果图像中包含灰色的部分，这些灰色像素会分布在较低灰度区间。
- **高灰度值区域频率较低或接近 0**：由于图像整体偏暗，高灰度值的像素会非常少，甚至可能没有。因此，直方图右侧几乎没有像素分布或频率非常低。

均衡变换曲线分析

- **快速上升的初始部分**：由于黑色和灰色部分像素频率很高，低灰度值的累计频率增长很快。因此，变换曲线在低灰度区域会快速上升。
- **缓慢上升的中高灰度部分**：中高灰度值区域的像素频率较低，因此变换曲线在中高灰度区域的上升速度明显减慢。
- **趋于平缓的末端部分**：在较高灰度值区域，变换曲线会逐渐趋于平缓。因为在原始图像中，高灰度值几乎没有像素，累计频率变化小。

均衡化后直方图分析

- **填补中高灰度区间**：由于原图像中大多数像素集中在低灰度区域，均衡化将把这些像素重新分布到更高的灰度值，填补原本较为空缺的中高灰度区域。
- **不完全均匀**：尽管直方图均衡化的目的是让灰度分布更均匀，但由于原始图像中黑色占比非常大，均衡化后中低灰度区域的像素密度仍高于高灰度区域，即无法达到理想的完全均匀分布。

均衡化后的图像分析

- **提升对比度**：均衡化将使黑色和灰色部分分布在更大的灰度范围内，从而提升图像的整体对比度。
- **图像变亮**：由于均衡化将低灰度区域的像素拉伸到中高灰度范围，图像整体亮度会有所增加。

实验二

要求

使用指定的拉普拉斯核 [[−1,−1,−1],[−1,8,−1],[−1,−1,−1]] 对图像进行锐化，基于公式 $g(x,y) = f(x,y) + c \times \nabla^2 f(x,y)$ 进行操作。

原理

图像锐化的目标是增强图像中的边缘和细节，使得图像更加清晰。拉普拉斯锐化通过计算图像的拉普拉斯变换（即图像的二阶导数）来实现这一目标。二阶导数可以很好地检测图像的灰度变化区域，因为在边缘区域，灰度值的变化速度最大。

在二维图像中，拉普拉斯算子 ∇^2 定义为：

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

其中，f(x,y) 表示图像在位置 (x,y) 处的灰度值。

本题拉普拉斯算子如下：

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

拉普拉斯锐化的基本公式为：

$$g(x,y) = f(x,y) + c \cdot \nabla^2 f(x,y)$$

其中：

- f(x,y) 是输入图像的灰度值，
- $\nabla^2 f(x,y)$ 是图像的拉普拉斯变换结果，代表边缘和细节的增强信息，
- c 是一个常数（1）。

因此，拉普拉斯锐化的步骤如下：

1. 使用拉普拉斯核对图像进行卷积，得到拉普拉斯变换结果。
2. 将拉普拉斯结果与原图像相加，得到锐化后的图像。

具体实现

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

font = {'family': 'MicroSoft YaHei',
        'weight': 'bold',
        'size': 'larger'}
(plt.rc("font", family='MicroSoft YaHei', weight="bold"))


def laplacian_enhancement(image, kernel, c=1):
    # 拉普拉斯锐化
    padded_image = np.pad(image, 1, mode='edge')
    output_image = np.zeros_like(image)

    for i in range(1, padded_image.shape[0] - 1):
        for j in range(1, padded_image.shape[1] - 1):
            region = padded_image[i - 1:i + 2, j - 1:j + 2]
            laplace_value = np.sum(region * kernel)
            output_image[i - 1, j - 1] = np.clip(image[i - 1, j - 1] + c * laplace_value, 0, 255)

    return output_image


# 拉普拉斯核
laplacian_kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])

# 加载图像b
image_b = Image.open("b.jpg").convert("L")
image_b = np.array(image_b)

# 应用拉普拉斯增强
enhanced_image = laplacian_enhancement(image_b, laplacian_kernel)

# 绘制结果
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image_b, cmap='gray')
axs[0].set_title('原始图像')

axs[1].imshow(enhanced_image, cmap='gray')
axs[1].set_title('拉普拉斯增强后的图像')

plt.show()
```

结果分析

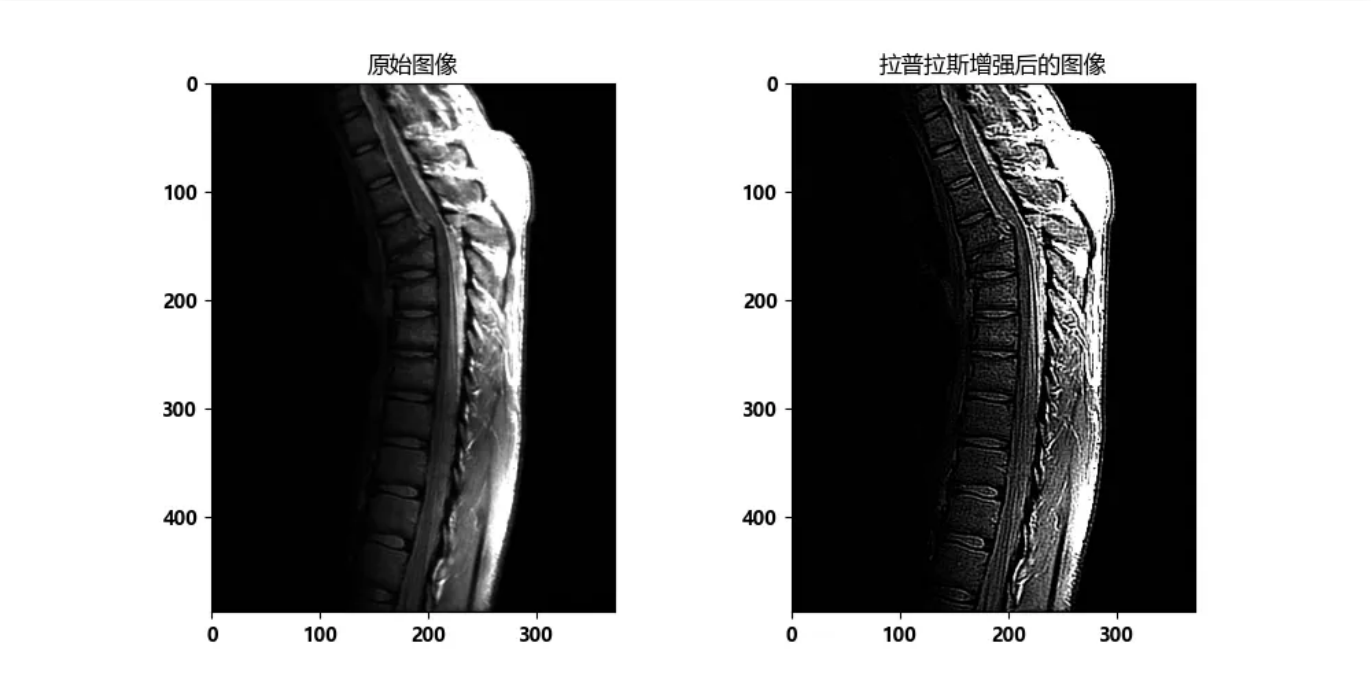

Figure_2

- **边缘增强**：增强后的图像中，物体的边缘更明显和清晰，轮廓线条显得更加锐利。
- **细节增强**：拉普拉斯增强会提升图像中较小的细节，使得这些细节在图像中更容易被观察到。
- **噪声放大**：拉普拉斯增强会放大图像中的高频成分（噪声也是），所以图像中的噪声在拉普拉斯增强后会变得更明显。

实验三

要求

编写代码提取并展示灰度图像的8个位平面，并修改最低位平面的值以观察修改后的差异。

原理

在灰度图像中，每个像素的灰度值可以用8位二进制表示，从最高位到最低位分别代表不同的灰度级权重。最低位平面（也就是第0位平面）控制的是细节变化，对整个图像的整体影响最小，而最高位平面对图像的影响最大。

如果一幅灰度图像的像素值用8位表示，那么可以将图像分解成8个位平面（位平面0到位平面7）。每个位平面包含对应二进制位的信息，例如，第0位平面表示每个像素二进制值的最低位（LSB），第7位平面表示最高位（MSB）。位平面的分布如下：

- **位平面0**：对图像整体影响最小，控制细节变化。
- **位平面7**：对图像的整体亮度和细节影响最大。

具体实现

Python

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

font = {'family': 'MicroSoft YaHei',
        'weight': 'bold',
        'size': 'larger'}
(plt.rc("font", family='MicroSoft YaHei', weight="bold"))


def bit_plane_extraction(image, bit_plane):
    # 提取位平面
    bit_mask = 1 << bit_plane
    return (image & bit_mask) >> bit_plane


def modify_bit_plane(image, bit_plane, new_bit_plane_values):
    bit_mask = 1 << bit_plane
    modified_image = image & ~bit_mask
    modified_image |= (new_bit_plane_values & 1) << bit_plane
    return modified_image

# 加载图像
#image_c = Image.open("lena-512-gray.png").convert("L")
image_c = Image.open("第四章编程作业图.jpg").convert("L")
image_c = np.array(image_c)
random_bit_plane_values = np.random.randint(0, 2, size=image_c.shape)
# 提取和展示8个位平面
fig, axs = plt.subplots(2, 4, figsize=(12, 6))
for i in range(8):
    bit_plane_image = bit_plane_extraction(image_c, i)
    axs[i//4, i%4].imshow(bit_plane_image, cmap='gray')
    axs[i//4, i%4].set_title(f'Bit Plane {i+1}')
plt.show()

# 修改最低位平面并展示
modified_image_array = modify_bit_plane(image_c, 0, random_bit_plane_value
s)
modified_image = Image.fromarray(modified_image_array)
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image_c, cmap='gray')
axs[0].set_title('Original Image')

axs[1].imshow(modified_image, cmap='gray')
axs[1].set_title('Modified Image (Lowest Bit Plane Changed)')
```

```
45
46
47
    plt.show()
```

结果分析

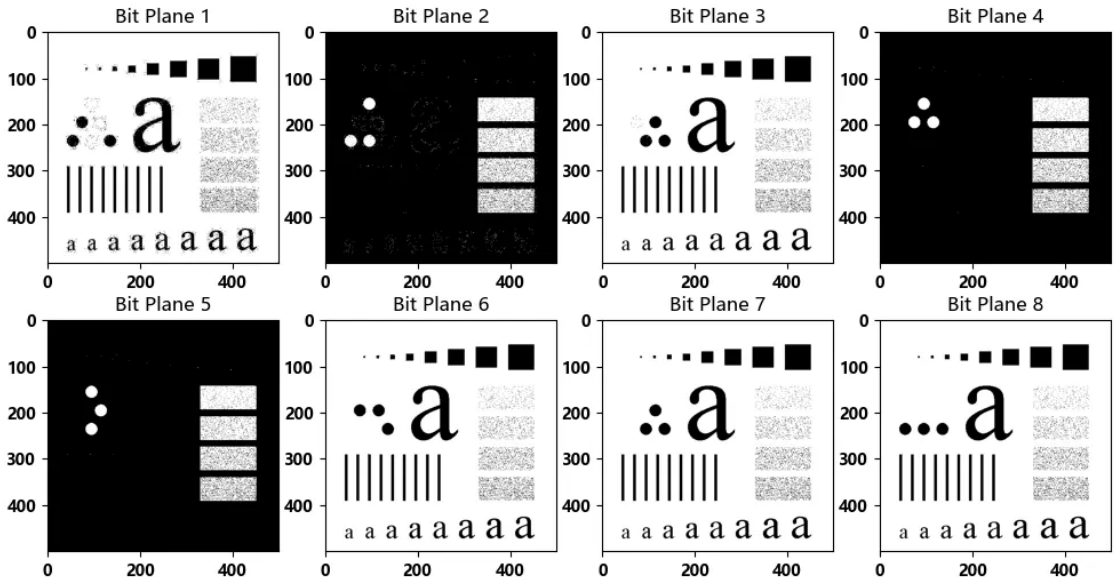

Figure_3

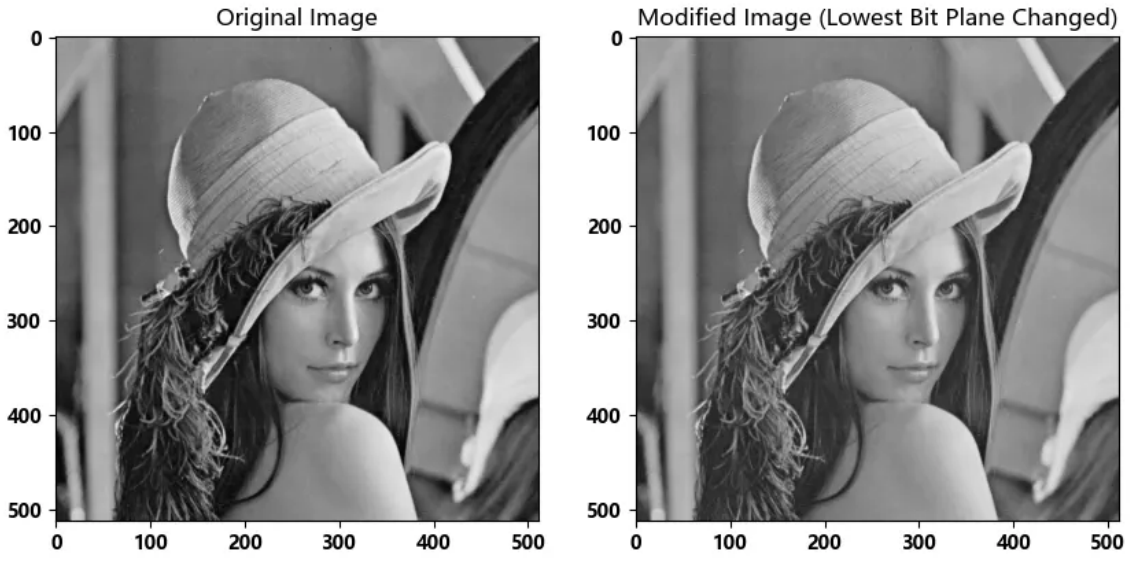

Figure_4

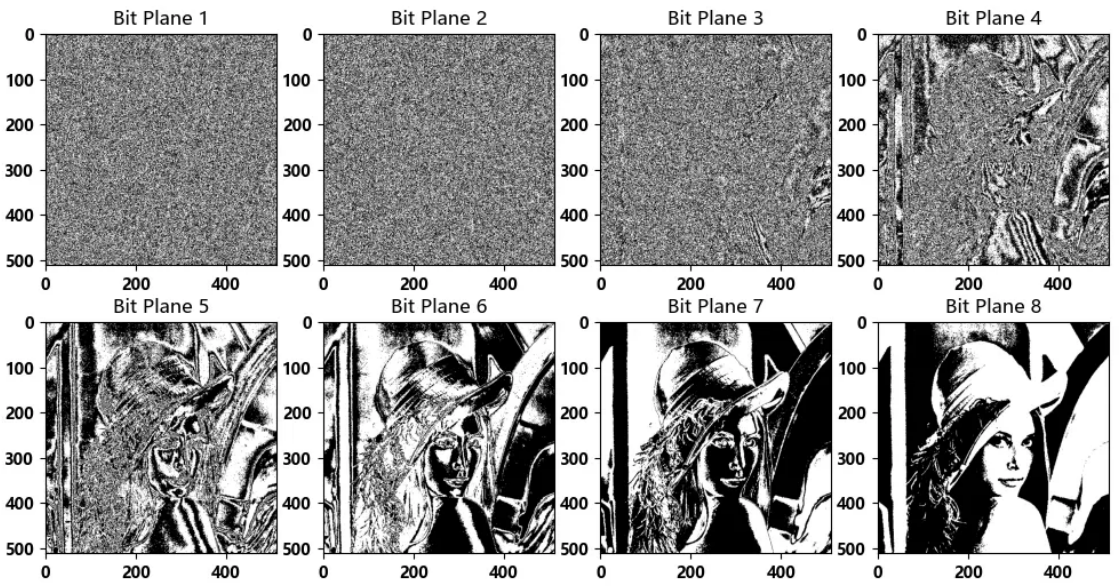

Figure_5

位平面提取效果

- **高位平面（第 8 位到第 5 位）**：这些位平面包含了图像的主要轮廓和结构信息。特别是最高位平面，保留了图像最主要的轮廓.
- **中位平面（第 4 位到第 2 位）**：这些位平面在高位平面的基础上增加了一些细节信息。它们对图像的清晰度有一定的贡献，但对整体结构的影响较小。
- **最低位平面（第 1 位）**：最低位平面包含了大量随机噪声，通常对图像的整体结构贡献较小。

最低位平面修改后的图像效果

- **整体视觉效果**：在更改最低位平面后重新组合得到的图像与原图像相比，图像变化会和课后习题3.5(a)的答案一样，整体变亮，色调降低。