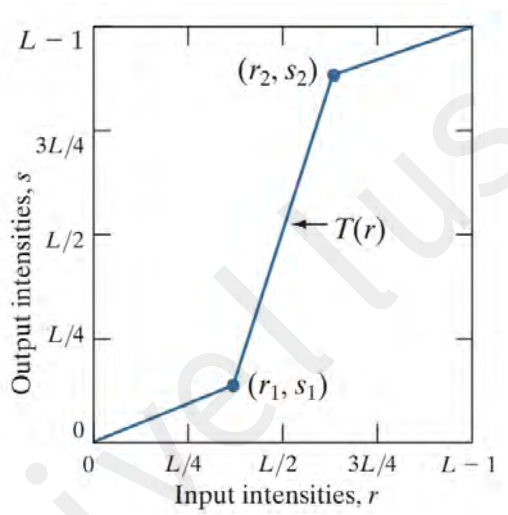
图像处理与可视化 Homework 01

姓名: 雍崔扬
学号: 21307140051

Note: Please provide a report file (in PPT/word/pdf) of your homework consisting of everything including description of data, code and result. Also please submit a zip file to elearning including the report, data/image if you use, and source code (with comments)

Problem 1

Implement a piecewise linear transformation function for image contrast stretching.
The code should read in an image; for intensity of all pixels, use the function to compute new intensity values;
and finally output/ save the image with new intensity values.



Solution:
分段线性函数的表达式为:

$$s = T(r) := \begin{cases} \frac{s_1}{r_1} r & 0 \le r \le r_1 \\ s_1 + \frac{s_2 - s_1}{r_2 - r_1}(r - r_1) & r_1 < r \le r_2 \\ s_2 + \frac{L-1-s_2}{L-1-r_2}(r - r_2) & r_2 < r \le L-1 \end{cases}$$

其中 $[0, L-1]$ 是输入输出图像的灰度值区间，r 和 s 分别代表输入图像和输出图像的灰度值.
而 r_1, r_2, s_1, s_2 是给定的常数.
本例中我们取 $\begin{cases} L = 256 \\ [r_1, r_2] = [\frac{3}{8}L, \frac{5}{8}L] = [96, 160] \\ [s_1, s_2] = [\frac{1}{8}L, \frac{7}{8}L] = [32, 324] \end{cases}$

① **分段线性变换函数** `piecewise_linear_transformation`:

```python
# 定义 Piecewise_Linear_Transformation 函数
def piecewise_linear_transformation(r, r1, r2, s1, s2, L=256):
    """
    对给定的灰度值 r 应用分段线性对比度拉伸变换.

    :param r: 输入的灰度值，要求是 [0, L-1] 范围内的整数
    :param r1: 第一个输入灰度分段阈值，要求满足 0 <= r1 < r2 < L-1
    :param r2: 第二个输入灰度分段阈值
    :param s1: 第一个输出灰度值，映射到 r1
    :param s2: 第二个输出灰度值，映射到 r2
    :param L: 灰度级别，默认值为 256，表示输入图像的灰度范围是 [0, 255]

    :return: 对应的输出灰度值 s，若输入灰度值 r 无效，则返回 None
    """

    # 检查输入灰度值 r 是否在 [0, L-1] 范围内
    if not (0 <= r <= L - 1):
        print(f"Invalid input intensity value r = {r}, should be an integer
within [0,{L-1}]")
        return None

    # 第一段: 对应 0 <= r <= r1 的灰度值线性映射
    if 0 <= r <= r1:
        return (s1 / r1) * r

    # 第二段: 对应 r1 < r <= r2 的灰度值线性映射
    elif r1 < r <= r2:
        return s1 + ((s2 - s1) / (r2 - r1)) * (r - r1)

    # 第三段: 对应 r2 < r <= L-1 的灰度值线性映射
    elif r2 < r <= L - 1:
        return s2 + ((L - 1 - s2) / (L - 1 - r2)) * (r - r2)

    # 默认情况下不会进入此分支，但为了安全性也处理一下无效输入
    else:
        print(f"Unexpected input value r = {r}")
        return None
```

② **对比度拉伸函数** `apply_contrast_stretching`：

```
# 应用分段线性变换到整个图像
def apply_contrast_stretching(image, r1, r2, s1, s2, L=256):
    """
    将分段线性变换应用到输入的灰度图像.
    :param image: 输入灰度图像的二维numpy数组
    :param r1, r2, s1, s2: 变换中的阈值和输出值
    :param L: 灰度值的级数，默认为 256
    :return: 经过对比度拉伸的图像
    """
    # 获取图像的尺寸
    height, width = image.shape
    print(f"height = {height}, width = {width}")

    # 创建新的图像数组用于存储输出图像
    stretched_image = np.zeros_like(image, dtype=np.uint8)

    # 对每个像素应用分段线性变换
    for i in range(height):
        for j in range(width):
            stretched_image[i, j] = piecewise_linear_transformation(image[i, j],
r1, r2, s1, s2, L)

    return stretched_image
```

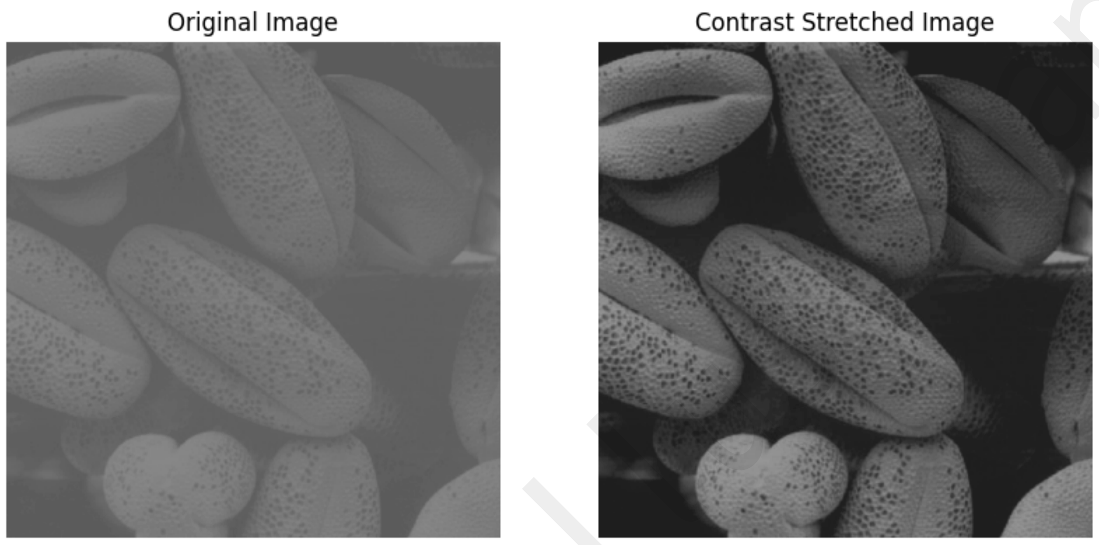
③ 读取/输出图像的函数 `in_and_out`:

```
# 读取图像，应用对比度拉伸，并显示结果
def in_and_out(image_name, a1=3/8, a2=5/8, b1=1/8, b2=7/8):
    """
    读取图像，应用分段线性对比度拉伸，并显示和保存结果。
    :param image_name: 输入图像的文件名
    :param a1, a2: 输入灰度值比例（默认为图像灰度范围的 3/8 到 5/8 之间）
    :param b1, b2: 输出灰度值比例（默认为图像灰度范围的 1/8 到 7/8 之间）
    """

    # 读取灰度图像
    input_image = Image.open(image_name).convert("L")
    image_array = np.array(input_image)

    # 获取灰度值的级数 L
    L = 256  # 假设输入图像是8位灰度图像，灰度级别为256

    # 定义对比度拉伸参数
    r1 = max(a1 * L, 0)  # 确保 r1 不小于 0
    r2 = min(a2 * L, L - 1)  # 确保 r2 不超过 L-1
    s1 = max(b1 * L, 0)  # 确保 s1 不小于 0
    s2 = min(b2 * L, L - 1)  # 确保 s2 不超过 L-1
    print(f"[r1, r2] = [{r1}, {r2}], [s1, s2] = [{s1}, {s2}]")

    # 应用分段线性对比度拉伸
    stretched_image = apply_contrast_stretching(image_array, r1, r2, s1, s2, L)

    # 转换为PIL图像
    output_image = Image.fromarray(stretched_image)

    # 显示原始图像和拉伸后的图像
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.title("Original Image")
    plt.imshow(input_image, cmap="gray", vmin=0, vmax=255)
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.title("Contrast Stretched Image")
    plt.imshow(output_image, cmap="gray", vmin=0, vmax=255)
    plt.axis("off")

    plt.show()

    # 保存拉伸后的图像
    output_image.save('Contrast_Streched_'+ str(image_name))
```

④ 函数调用:

```
# 主函数
def main():
    """
    主函数，用于运行对比度拉伸算法。
    """
    # 调用 in_and_out 函数处理图像
    in_and_out("DIP 3.10 (pollen).tif")

if __name__ == "__main__":
    main()
```

运行结果: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.10)



Problem 2

Part (1)

Implement n-dimensional joint histogram and test the code on two-dimensional data; plot the results.
(所谓 n 维联合灰度直方图，指的是图像的每个像素都有 n 维灰度向量，而 $2\mathrm{D}$ 数据可由两幅 $1\mathrm{D}$ 灰度图像堆叠而成)

Solution:

① 计算图像联合直方图的函数 `compute_joint_histogram`:

```
def compute_joint_histogram(image_array, num_bins=256):
    """
    计算 n 维联合直方图。

    :param image_array: n 维灰度图像的 numpy 数组
    :param num_bins: 直方图的 bins 数量
    :return: n 维联合直方图和每个维度的 bins 边缘
    """
    # 检查图像是否为 n 维
    if image_array.ndim < 2:
        raise ValueError("The gray-level dimension of input picture must be
higher than 2")
```

```
    # 将 n 维图像展平为二维数组，其中每一行是一个像素的 n 维向量
    reshaped_array = image_array.reshape(-1, image_array.shape[-1])

    # 计算 n 维联合直方图
    joint_hist, edges = np.histogramdd(
        reshaped_array,
        bins=[num_bins] * image_array.shape[2],
        range=[[0, num_bins]] * image_array.shape[2]
    )

    return joint_hist, edges
```

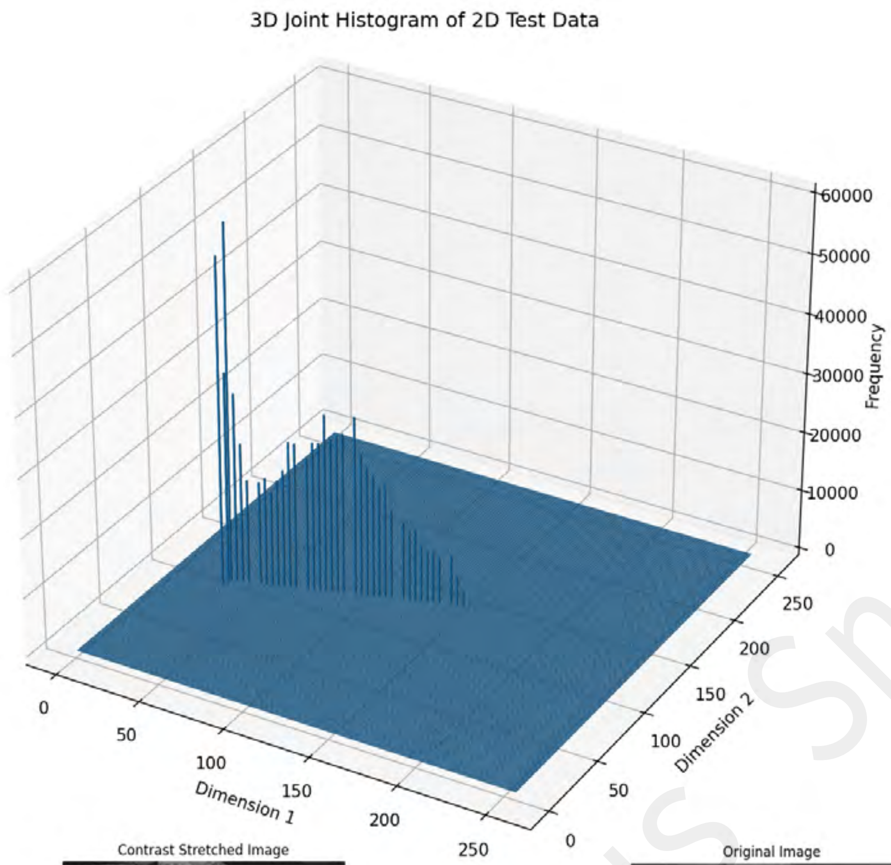
② **绘制图像直方图的函数** `plot_joint_histogram`：

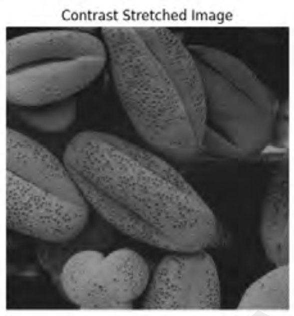
```
def plot_joint_histogram(joint_hist, edges):
    """
    绘制联合直方图。只绘制二维联合直方图。

    :param joint_hist: n 维联合直方图的 numpy 数组
    :param edges: 每个维度的 bins 边缘
    """
    if joint_hist.ndim == 2:
        xedges = np.array(edges[0])
        yedges = np.array(edges[1])

        # 绘制 3D 条形图
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')

        xpos, ypos = np.meshgrid(xedges[:-1] + (xedges[1] - xedges[0]) / 2,
                                 yedges[:-1] + (yedges[1] - yedges[0]) / 2,
indexing="ij")
        xpos = xpos.ravel()
        ypos = ypos.ravel()
        zpos = 0

        # Construct arrays with the dimensions for the bars.
        dx = dy = (xedges[1] - xedges[0]) * 0.8  # 增大柱形图的底面积
        dz = joint_hist.ravel()

        ax.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average')
        ax.set_title('3D Joint Histogram of 2D Test Data')
        ax.set_xlabel('Dimension 1')
        ax.set_ylabel('Dimension 2')
        ax.set_zlabel('Frequency')
        plt.tight_layout()
        plt.savefig('3d_joint_histogram.png', dpi=150)
        plt.show()

        # 绘制 2D 彩色联合直方图
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111)
        H = ax.hist2d(xedges[:-1], yedges[:-1], bins=[xedges, yedges],
weights=joint_hist.ravel(), cmap='viridis')
        fig.colorbar(H[3], ax=ax)
        ax.set_title('2D Joint Histogram of 2D Test Data')
        ax.set_xlabel('Dimension 1')
        ax.set_ylabel('Dimension 2')
        plt.tight_layout()
        plt.savefig('2d_joint_histogram.png', dpi=150)
        plt.show()
    else:
        print(f"Cannot plot {joint_hist.ndim}-dimensional joint histogram. Only 
2D histograms can be plotted.")
```

③ 函数调用:

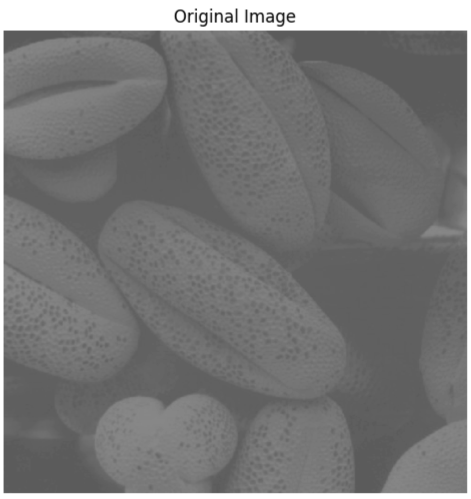
```
def main():
    # 加载两幅图像并将它们堆叠为一个 3D 数组
    image1 = Image.open('Contrast_Streched_DIP 3.10 (pollen).tif').convert('L')
    image2 = Image.open('DIP 3.10 (pollen).tif').convert('L')

    # 转换为 numpy 数组
    image1_array = np.array(image1)
    image2_array = np.array(image2)

    # 检查图像是否具有相同的尺寸
    if image1_array.shape != image2_array.shape:
        raise ValueError("The two pictures must have the same shape!")

    # 堆叠图像形成 3D 数组
    stacked_images = np.stack((image1_array, image2_array), axis=-1)

    # 打印图像的维度
    print(f"Stacked image dimensions: {stacked_images.shape}")

    # 计算联合直方图
    joint_hist, edges = compute_joint_histogram(stacked_images, num_bins=256)

    # 绘制联合直方图
    plot_joint_histogram(joint_hist, edges)

if __name__ == "__main__":
    main()
```

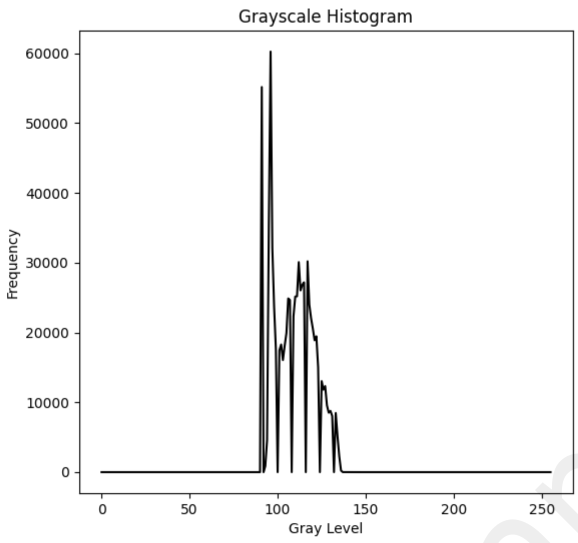
运行结果: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.10)

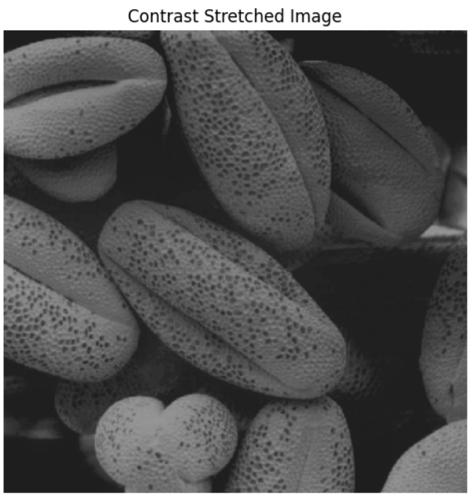

3D Joint Histogram of 2D Test Data
Frequency
Dimension 1
Dimension 2
0
50
100
150
200
250
0
10000
20000
30000
40000
50000
60000

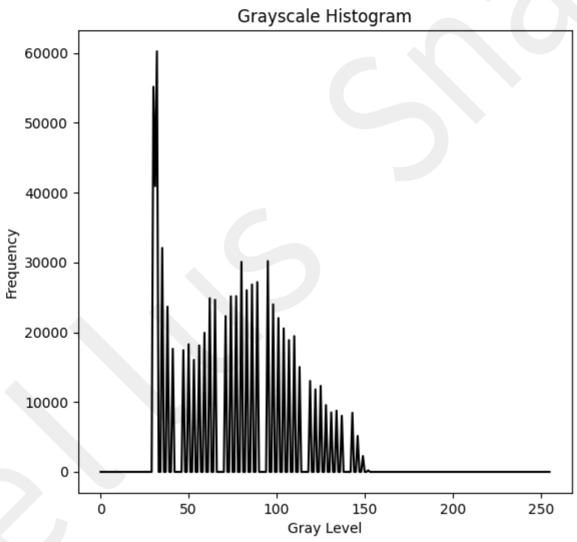

Contrast Stretched Image

Original Image









Part (2)

Implement computation of local histograms of an image using the efficient update of local histogram method introduced in local histogram processing. Note that because only one row or column of the neighborhood changes in a one-pixel translation of the neighborhood, updating the histogram obtained in the previous location with the new data introduced at each motion step is possible and efficient in computation.

Solution:
① 计算窗口直方图的函数 `compute_histogram`**:**

```
def compute_grayscale_histogram(image, num_bins=256):
    """
    计算图像的灰度直方图。

    :param image: 灰度图像的 numpy 数组
    :param num_bins: 直方图的 bins 数量
    :return: 直方图和 bins 边缘
    """
    # 将图像展平并计算直方图
    histogram, bin_edges = np.histogram(image.ravel(), bins=num_bins, range=[0,
num_bins])

    return histogram, bin_edges
```

② 根据已有的窗口直方图得到邻近窗口直方图的函数 `update_histogram`:

```
def update_histogram(old_hist, new_col=None, remove_col=None, num_bins=256):
    """
    使用增量更新直方图.
    :param old_hist: 当前的直方图.
    :param new_col: 要加入的新的列像素 (可以为 None).
    :param remove_col: 要移除的列像素 (可以为 None).
    :param num_bins: 直方图的 bins 数量.
    :return: 更新后的直方图.
    """
    if new_col is None:
        return old_hist - np.bincount(remove_col, minlength=num_bins)
    elif remove_col is None:
        return old_hist + np.bincount(new_col, minlength=num_bins)
    else:
        return old_hist - np.bincount(remove_col, minlength=num_bins) +
np.bincount(new_col, minlength=num_bins)
```

③ 计算整个图像的所有窗口直方图的函数 `compute_local_histograms`:

```
def compute_local_histograms(image, window_size=(9, 9), num_bins=256):
    """
    计算图像的局部直方图.
    :param image: 输入的灰度图像.
    :param window_size: 邻域窗口的尺寸 (height, width).
    :param num_bins: 直方图的 bins 数量.
    :return: 所有局部直方图的列表.
    """
    h, w = image.shape
    win_h, win_w = window_size

    # 初始化局部直方图列表
    local_histograms = np.zeros((h - win_h + 1, w - win_w + 1, num_bins),
dtype=np.uint8)

    # 计算第一个窗口的直方图
    first_window = image[0:win_h, 0:win_w]
    local_histograms[0, 0, :], _ = compute_histogram(first_window,
num_bins=num_bins)

    # 移动窗口
    for i in range(h - win_h + 1):
        if i > 0:
            # 更新直方图 (垂直移动)
            new_row = image[i + win_h - 1, 0:win_w]
            remove_row = image[i - 1, 0:win_w]
            local_histograms[i, 0, :] = update_histogram(local_histograms[i-1,
0, :], new_row, remove_row, num_bins=num_bins)

        for j in range(1, w - win_w + 1):
            # 更新直方图 (水平移动)
            new_col = image[i:i + win_h, j + win_w - 1]
            remove_col = image[i:i + win_h, j - 1]
            local_histograms[i, j, :] = update_histogram(local_histograms[i, j-
1, :], new_col, remove_col, num_bins=num_bins)

    return local_histograms
```

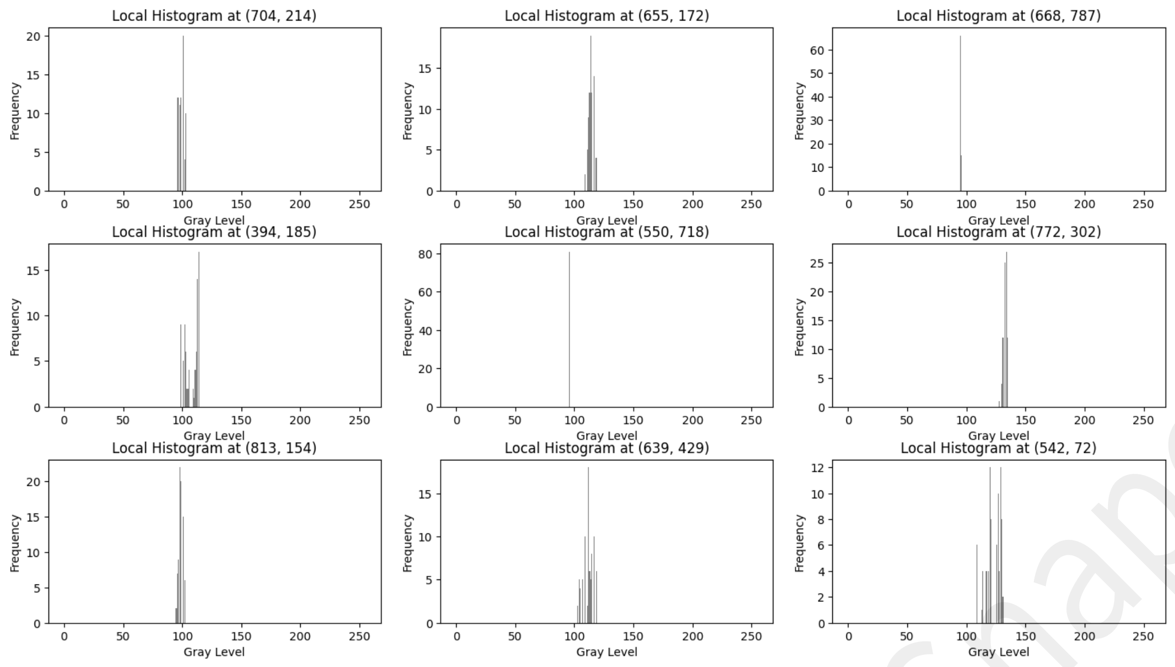
④ **随机绘制某些窗口直方图的函数** `plot_random_local_histograms`:

```
def compute_local_histograms(image, window_size=(9, 9), num_bins=256):
    """
    计算图像的局部直方图.
    :param image: 输入的灰度图像.
    :param window_size: 邻域窗口的尺寸 (height, width).
    :param num_bins: 直方图的 bins 数量.
    :return: 所有局部直方图的列表.
    """
    h, w = image.shape
    win_h, win_w = window_size
    half_win_h = win_h  // 2  # 使用整数除法，得到窗口半径
    half_win_w = win_w  // 2

    # 初始化局部直方图列表
    local_histograms = np.zeros((h, w, num_bins), dtype=np.uint8)

    # 移动完整窗口
    for i in range(h):
        if i == 0:
            # 计算第一个完整窗口的直方图
            local_histograms[0, 0, :], _ =
compute_histogram(image[0:half_win_h+1,0:half_win_w+1], num_bins=num_bins)
        else:
        # 更新直方图 (垂直移动)
            if i <= half_win_h: # 首
                new_row = image[i+half_win_h, 0:half_win_w+1]
                remove_row = None
            elif i >= h - half_win_h: # 尾
                new_row = None
                remove_row = image[i-half_win_h-1, 0:half_win_w+1]
            else: # 中间部分
                new_row = image[i+half_win_h, 0:half_win_w+1]
                remove_row = image[i-half_win_h-1, 0:half_win_w+1]
            # 更新直方图 (垂直移动)
            local_histograms[i, 0, :] = update_histogram(local_histograms[i-1,
0, :], new_row, remove_row, num_bins=num_bins)

        i_safe_lower = max(i-half_win_h,0)
        i_safe_upper = min(i+half_win_h+1,h)

        for j in range(1, w):
            # 更新直方图 (水平移动)
            if j <= half_win_w: # 首
                new_col = image[i_safe_lower:i_safe_upper, j+half_win_w]
                remove_col = None
            elif j >= w - half_win_w: # 尾
                new_col = None
                remove_col = image[i_safe_lower:i_safe_upper, j-half_win_w-1]
            else: # 中间部分
                new_col = image[i_safe_lower:i_safe_upper, j+half_win_w]
                remove_col = image[i_safe_lower:i_safe_upper, j-half_win_w-1]

            # 更新直方图 (水平移动)
            local_histograms[i, j, :] = update_histogram(local_histograms[i, j-
1, :], new_col, remove_col, num_bins=num_bins)

    return local_histograms
```

⑤ 函数调用:

```
def main():
    # 加载灰度图像
    image = Image.open('DIP 3.10 (pollen).tif').convert('L')
    image_array = np.array(image)

    # 定义窗口大小
    window_size = (9, 9)

    # 定义直方图的 bins 数量
    num_bins = 256

    # 计算局部直方图
    local_histograms = compute_local_histograms(image_array,
window_size=window_size, num_bins=num_bins)

    # 随机绘制某些窗口的直方图
    num_samples = 9
    plot_random_local_histograms(local_histograms, num_samples)

if __name__ == "__main__":
    main()
```

运行结果: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.10)
其中我们选用 9×9 尺寸的窗口.



Problem 3

Part (1)

Implement histogram equalization algorithm.

Solution:
(图像均衡化算法)
图像 f 中出现灰度级 r_k 的频率为:

$$p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN} \quad (k = 0, 1, \dots, L-1)$$

其中 $M \times N$ 为图像 f 的尺寸，n_k 表示灰度值为 r_k 的像素数.

此时变换 $s = T(r) = (L-1)\int_0^r p_r(w)dw$ 的离散形式为:

$$s_k = T(r_k) = (L-1)\sum_{i=0}^{k} p(r_i) \quad (k = 0, 1, \dots, L-1)$$

(注意最后还需将 s_k $(k = 0, \dots, L-1)$ 舍入到区间 $[0, L-1]$ 中的整数值)

---

① 计算窗口直方图的函数 `compute_histogram`: (定义见 Problem 2 Part (2) ①)
② 计算 (经验) 累积分布函数的函数 `compute_cdf`:

```python
def compute_cdf(histogram):
    """
    计算(经验)累积分布函数 (CDF)
    :param histogram: 图像的灰度直方图
    :return: 累积分布函数
    """
    cdf = np.cumsum(histogram)  # 累加直方图的每个值
    cdf_normalized = cdf / cdf[-1]  # 归一化
    return cdf_normalized
```

③ 应用均衡化变换的函数 `histogram_equalization`：

```python
def histogram_equalization(image, num_bins=256):
    """
    对图像进行均衡化
    :param image: 输入灰度图像的 numpy 数组
    :return: 均衡化后的图像
    """
    # 计算灰度直方图
    histogram, bin_edges = compute_histogram(image, num_bins=num_bins)

    # 计算累积分布函数 (CDF)
    cdf_normalized = compute_cdf(histogram)

    # 计算均衡化映射
    equalized_map = (num_bins - 1) * cdf_normalized
    equalized_map = np.round(equalized_map).astype(np.uint8)

    # 应用均衡化映射到图像
    equalized_image = equalized_map[image]
    return histogram, bin_edges, equalized_image
```

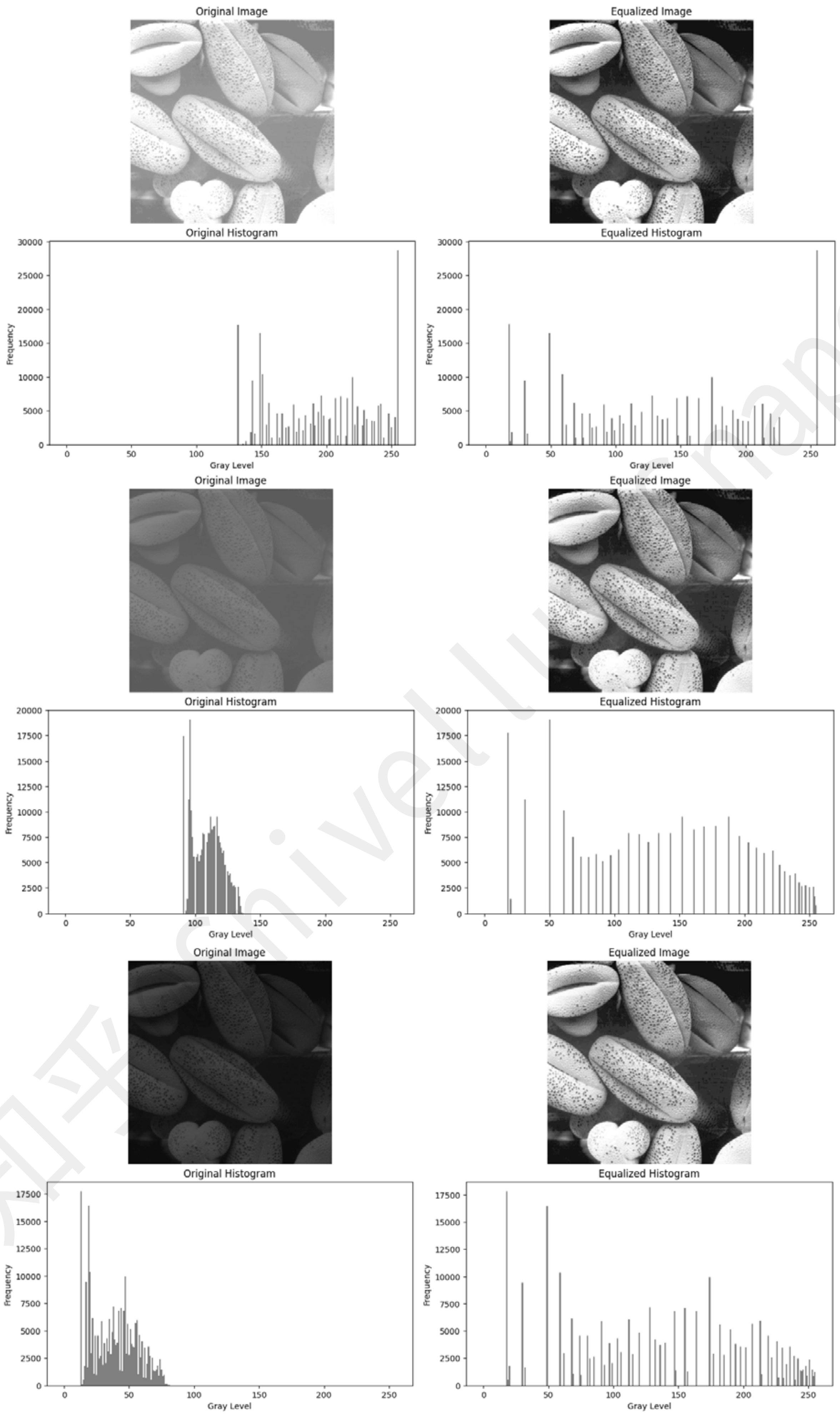
④ 绘制原图像和均衡化图像及其直方图的函数 `plot_image_and_histogram`：

```python
def plot_image_and_histogram(image, histogram, bin_edges, equalized_image,
num_bins=256):
    """
    绘制原始图像、均衡化后的图像以及它们的灰度直方图
    :param image: 原始灰度图像的 2D numpy 数组
    :param equalized_image: 均衡化后的图像的 2D numpy 数组
    """
    fig, axes = plt.subplots(2, 2, figsize=(14, 10))

    # 绘制原始图像
    axes[0, 0].imshow(image, cmap='gray', vmin=0, vmax=num_bins-1)
    axes[0, 0].set_title('Original Image')
    axes[0, 0].axis('off')

    # 绘制原始图像的直方图
    axes[1, 0].bar(bin_edges[:-1], histogram, width=1, color='gray')
    axes[1, 0].set_title('Original Histogram')
    axes[1, 0].set_xlabel('Gray Level')
    axes[1, 0].set_ylabel('Frequency')

    # 绘制均衡化后的图像
    axes[0, 1].imshow(equalized_image, cmap='gray', vmin=0, vmax=num_bins-1)
    axes[0, 1].set_title('Equalized Image')
    axes[0, 1].axis('off')

    # 绘制均衡化后的图像的直方图
    equalized_histogram, _ = compute_histogram(equalized_image)
    axes[1, 1].bar(bin_edges[:-1], equalized_histogram, width=1, color='gray')
    axes[1, 1].set_title('Equalized Histogram')
    axes[1, 1].set_xlabel('Gray Level')
    axes[1, 1].set_ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```

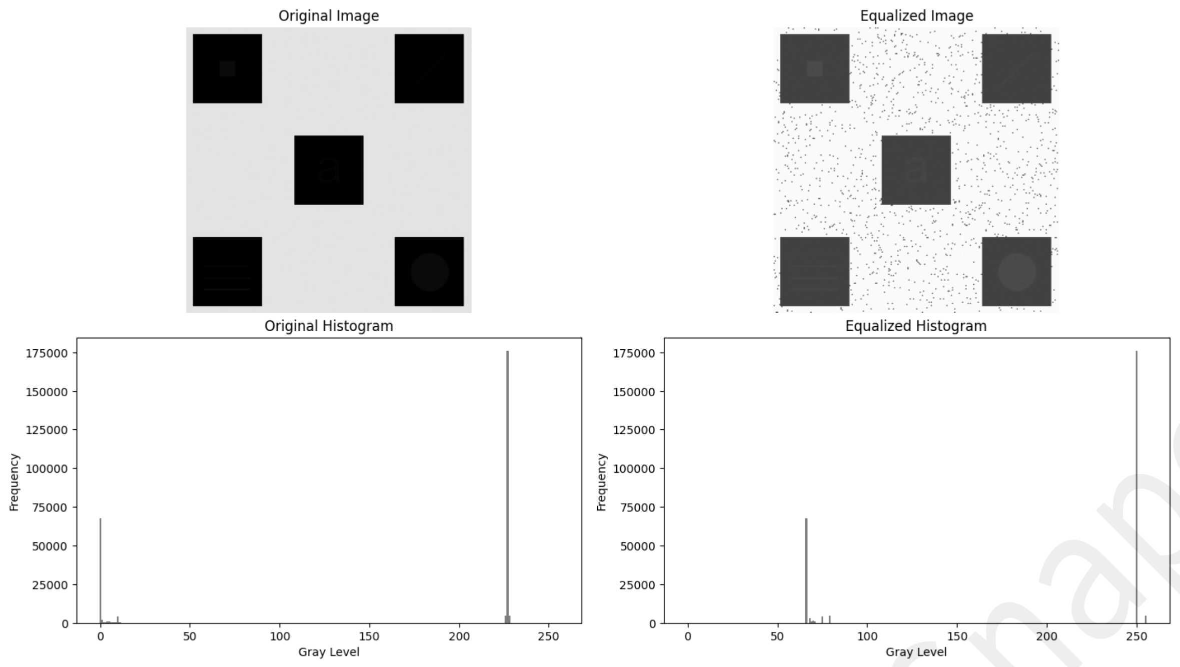
⑤ 函数调用:

```
def main():
    # 加载灰度图像
    image_name = 'DIP 3.20 (4).tif'
    image = Image.open(image_name).convert('L')
    image_array = np.array(image)

    # 进行图像均衡化
    num_bins = 256
    histogram, bin_edges, equalized_image = histogram_equalization(image_array,
num_bins=num_bins)

    # 保存均衡后的图像
    Image.fromarray(equalized_image).save('equalized_'+ str(image_name))

    # 绘制原始图像和均衡化后的图像及其直方图
    plot_image_and_histogram(image_array, histogram, bin_edges, equalized_image)

if __name__ == "__main__":
    main()
```

运行结果 1: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.20)



运行结果 2: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.26)



Part (2)

Implement the local histogram equalization using efficient computation of local histogram.

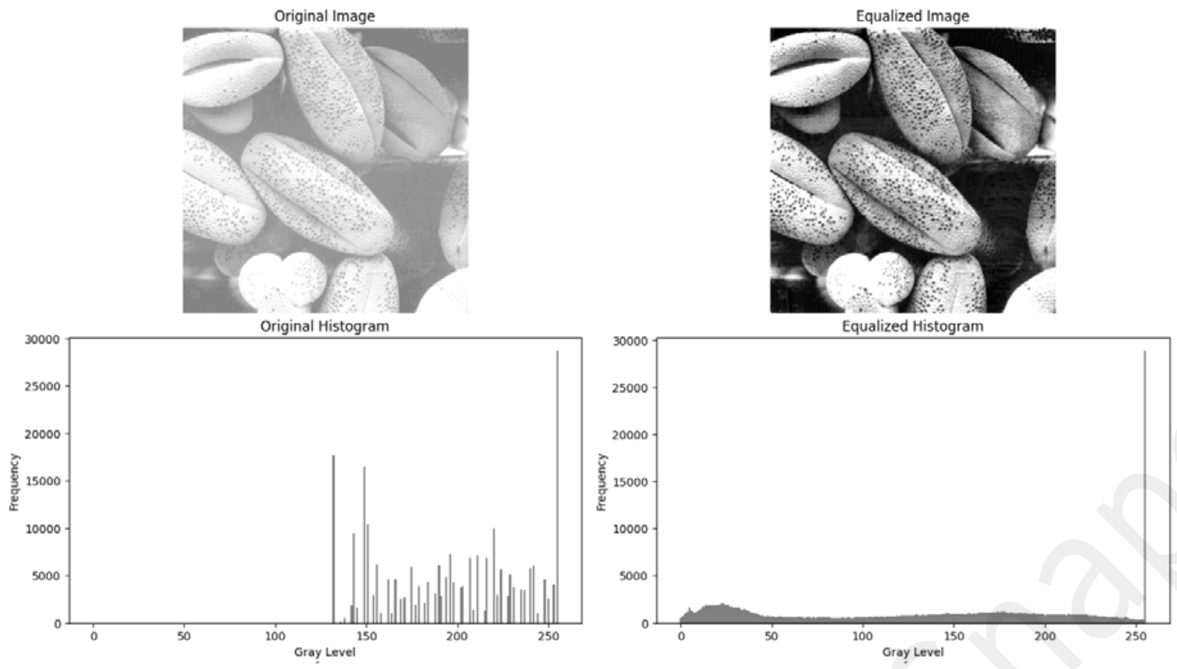
Solution:
① 计算窗口直方图的函数 `compute_histogram`: (定义见 Problem 2 Part (2) ①)
② 根据已有的窗口直方图得到邻近窗口直方图的函数 `update_histogram`: (定义见 Problem 2 Part (2) ②)
③ 计算整个图像的所有窗口直方图的函数 `compute_local_histograms`: (定义见 Problem 2 Part (2) ③)
④ 计算 (经验) 累积分布函数的函数 `compute_cdf`: (定义见 Problem 3 Part (1) ②)
⑤ 利用局部直方图进行均衡化变换的函数 `local_histogram_equalization`:

```
def local_histogram_equalization(image, local_histograms, num_bins=256):
    """
    对图像进行局部均衡化.
    :param image: 输入的灰度图像.
    :param local_histograms: 所有局部直方图的列表.
    :param window_size: 邻域窗口的尺寸 (height, width).
    :param num_bins: 直方图的 bins 数量.
    :return: 局部均衡化后的图像.
    """
    h, w = image.shape

    # 初始化均衡化后的图像
    equalized_image = np.zeros_like(image, dtype=np.uint8)

    # 遍历像素点，参照局部直方图应用均衡化映射
    for i in range(h):
        for j in range(w):
            # 使用局部直方图
            current_hist = local_histograms[i, j, :]

            # 计算累积分布函数 (CDF)
            current_cdf = compute_cdf(current_hist)
            equalized_map = (num_bins - 1) * current_cdf
            equalized_map = np.round(equalized_map).astype(np.uint8)

            # 应用均衡化映射到当前点
            equalized_image[i, j] = equalized_map[image[i, j]]

    return equalized_image
```

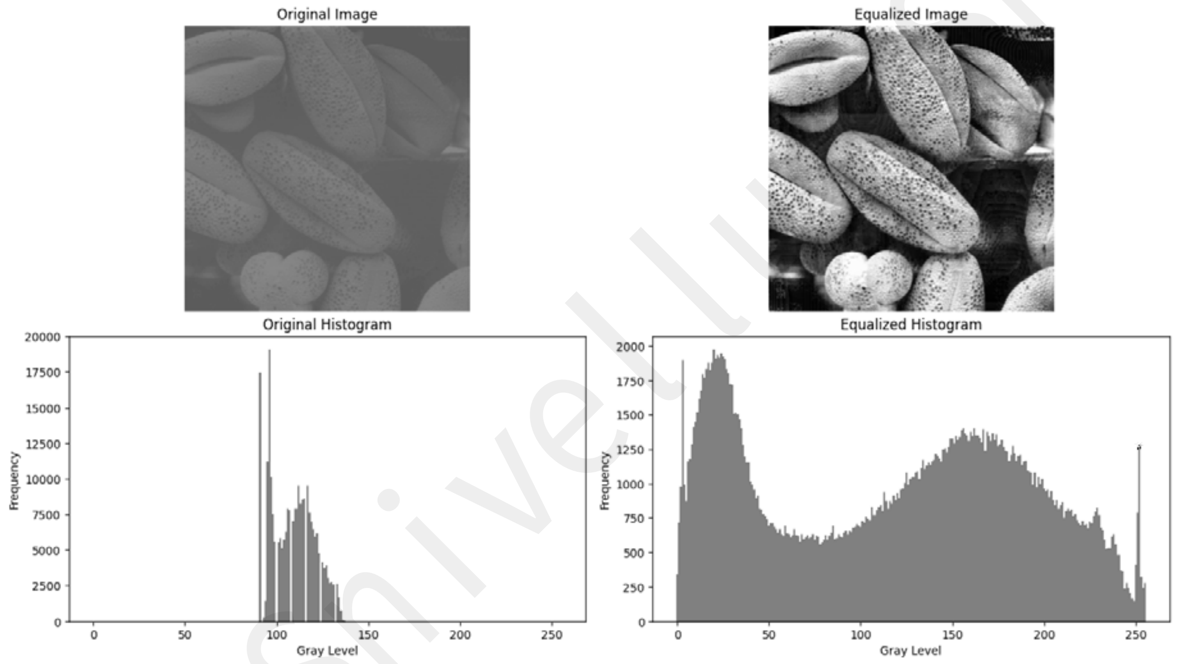
⑥ 绘制原图像和均衡化图像及其直方图的函数 `plot_image_and_histogram` **: (定义见 Problem 3 Part (1) ④)**
⑦ 函数调用:

```
def main():
    # 加载灰度图像
    image_name = 'DIP 3.20 (1).tif'
    image = Image.open(image_name).convert('L')
    image_array = np.array(image)

    # 定义窗口大小
    window_size = (121, 121)

    # 计算局部直方图
    num_bins = 256
    local_histograms = compute_local_histograms(image_array,
window_size=window_size, num_bins=num_bins)

    # 进行局部均衡化
    equalized_image = local_histogram_equalization(image_array,
local_histograms, num_bins=num_bins)

    # 保存均衡后的图像
    Image.fromarray(equalized_image).save('Local_equalized_'+ str(image_name))

    # 绘制原始图像和均衡化后的图像及其直方图
    plot_image_and_histogram(image_array, equalized_image, num_bins=num_bins)

if __name__ == "__main__":
    main()
```

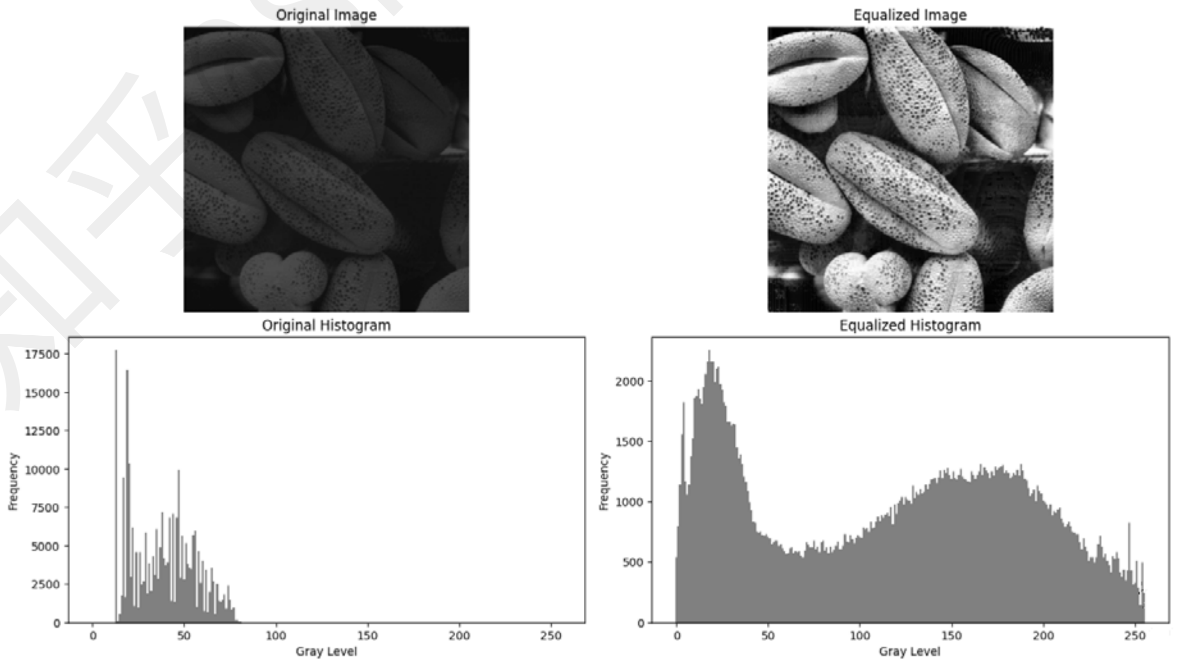
运行结果 1: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.20)
其中我们使用尺寸为 121×121 的窗口.



Original Image
Equalized Image
Original Histogram
Equalized Histogram
Frequency
Gray Level

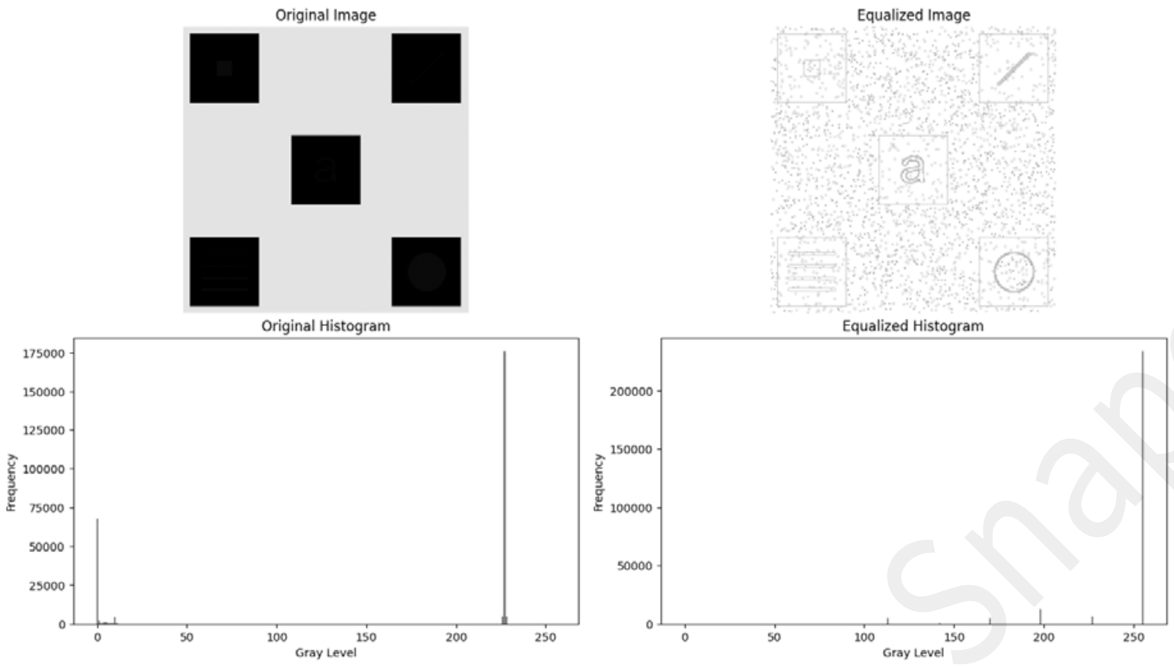


Original Image
Equalized Image
Original Histogram
Equalized Histogram
Frequency
Gray Level



Original Image
Equalized Image
Original Histogram
Equalized Histogram
Frequency
Gray Level

运行结果 2: (图片来源: Digital Image Process (3rd Edition, R. Gonzalez, R. Woods) Figure 3.26)
其中我们使用尺寸为 3×3 的窗口.


Original Image
Equalized Image
Original Histogram
Equalized Histogram
Frequency
Gray Level