第二次作业

人工智能 2204 班 王誉诺 U202214969

1. 理论作业

实现代码：

```
% 定义原始矩阵
B = [1 2 1 4 3;
     1 10 2 3 4;
     5 2 6 8 8;
     5 5 7 0 8;
     5 6 7 8 9];

% 使用自定义均值及中值滤波器，使用 3x3 滤波器
ave_filtered_B = avefilt(B, 3);
med_filtered_B = medfilter(B, 3);

% 显示滤波后的矩阵
disp('均值滤波后的图像（边框保持不变）：');
disp(ave_filtered_B);

disp('中值滤波后的图像（边框保持不变）：');
disp(med_filtered_B);
```

avefilt.m:

```
function avgf = avefilt(I, n)
% I 为原图，n 为滤波器的大小
a = ones(n);  % 生成一个全为 1 的滤波器矩阵
[r, c] = size(I);  % 获取图像的行和列大小
dI = double(I);  % 将图像转换为 double 类型，便于计算
dt = (n - 1) / 2;  % 滤波器半径

% 初始化输出图像，保留原图边缘像素
avgf = dI;

% 对图像内部进行均值滤波处理，不处理边缘
for i = 1+dt:r-dt
    for j = 1+dt:c-dt
        % 提取当前模板覆盖的区域并与模板相乘
```

```
            w = dl((i-dt):(i+dt), (j-dt):(j+dt)) .* a;
            % 求模板中所有元素的和
            sw = sum(sum(w));
            % 将均值赋值给模板的中心位置
            avgf(i, j) = sw / (n * n);
        end
end

avgf = round(avgf,4);   % 保留 4 位小数

end
```

medfilter.m:

```
function medf = medfilter(I, n)
% I 为原图，n 为滤波器的大小
[r, c] = size(I);   % 获取图像的行和列大小
dI = double(I);   % 将图像转换为 double 类型，便于计算
dt = (n - 1) / 2;   % 滤波器半径

% 初始化输出图像，保留原图边缘像素
medf = dI;

% 对图像内部进行中值滤波处理，不处理边缘
for i = 1+dt:r-dt
    for j = 1+dt:c-dt
        % 提取当前模板覆盖的区域
        window = dI((i-dt):(i+dt), (j-dt):(j+dt));
        % 将窗口中的元素展平为一维向量，然后取中值
        medf(i, j) = median(window(:));
    end
end

medf = round(medf, 4);   % 保留 4 位小数

end
```

处理步骤：

（1）均值滤波器

对图像进行灰度处理，并将其转换为‘double’类型，生成一个 3×3 的滤波器矩阵；保证边缘像素保持不变，只对内部区域进行处理，将内部区域每个像素的邻域像素值求均值，替换中心像素值；处理完成后，将结果四舍五入保留 4

位小数。

（2）**中值滤波器**

同样对图像进行灰度处理，并转换为‘double’ 类型,；保证边缘像素保持不变，只对内部区域进行处理，对内部区域每个像素的 3×3 邻域进行排序，取中间的值替换中心像素值。

结果图像：

```
>> lbq_1
均值滤波后的图像（边框保持不变）：
    1.0000    2.0000    1.0000    4.0000    3.0000
    1.0000    3.3333    4.2222    4.3333    4.0000
    5.0000    4.7778    4.7778    5.1111    8.0000
    5.0000    5.3333    5.4444    6.7778    8.0000
    5.0000    6.0000    7.0000    8.0000    9.0000

中值滤波后的图像（边框保持不变）：
     1     2     1     4     3
     1     2     3     4     4
     5     5     5     6     8
     5     5     6     8     8
     5     6     7     8     9

>>
```

2. 编程作业

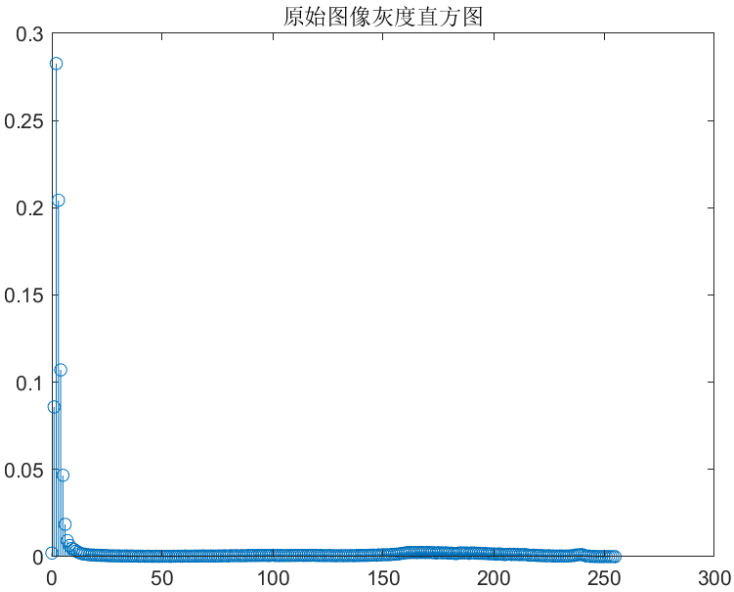
（1）灰度直方图显示以及直方图均衡化操作：

实现代码：

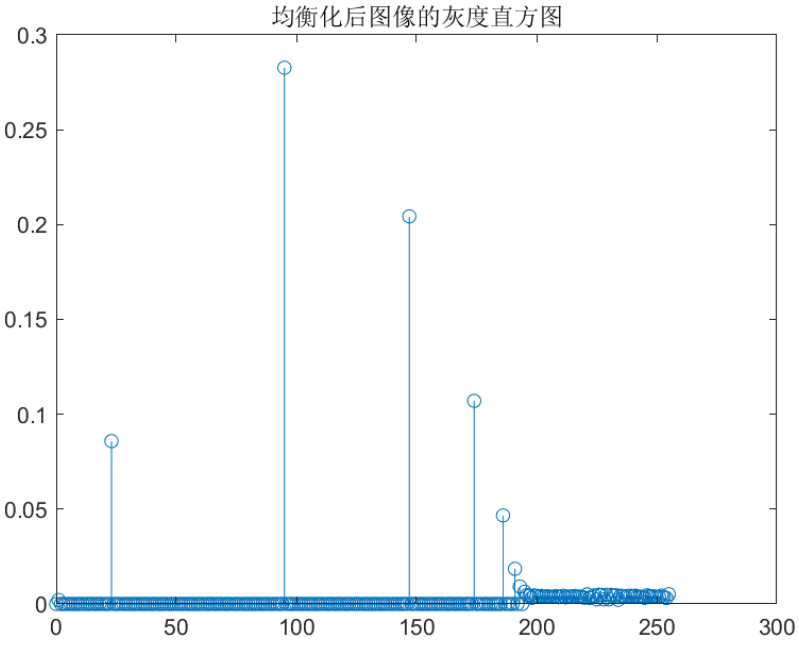
```
Img = imread('D:\文档\数字图像处理\low_light_image.jpg'); % 读取低照度图像
Img = rgb2gray(Img); % 转化为灰度图

[height, width] = size(Img);% 获取图像的尺寸信息

figure;
imshow(Img); title('原始灰度图像');% 绘制原始图像及其灰度直方图

% 计算并显示直方图，归一化到[0, 1]
[counts1, x] = imhist(Img, 256);
counts2 = counts1 / (height * width); % 归一化直方图
figure;
stem(x, counts2); title('原始图像灰度直方图');
```

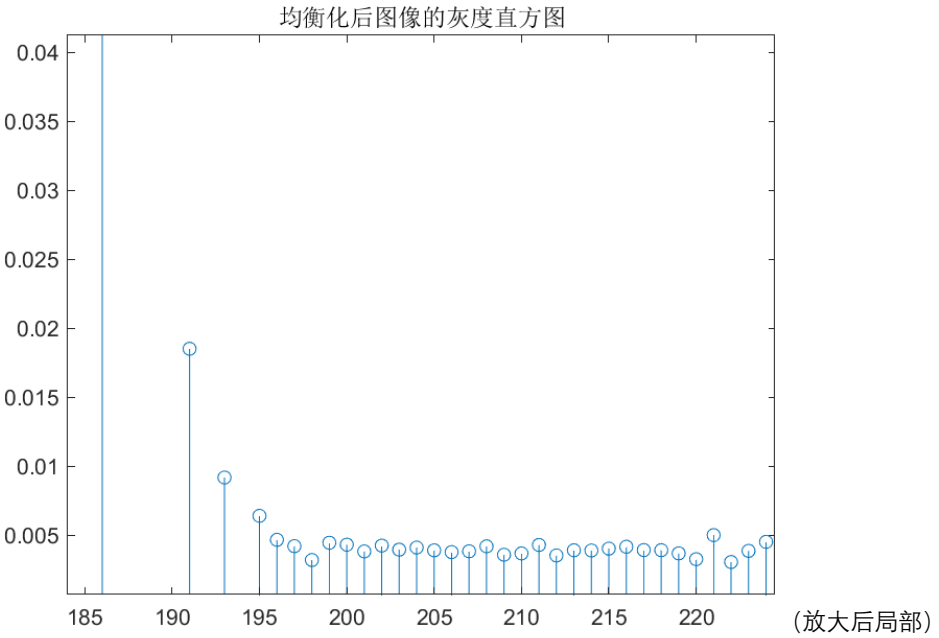
```
% 统计每个灰度级的像素数
NumPixel = zeros(1, 256); % 初始化各灰度值的计数数组
for i = 1:height
    for j = 1:width
        % 对应灰度值像素点数量+1 (灰度值从 0 开始，因此+1)
        NumPixel(Img(i,j) + 1) = NumPixel(Img(i,j) + 1) + 1;
    end
end

ProbPixel = NumPixel / (height * width); % 将频数转换为概率分布（频率）

% 计算累积分布函数(CDF)，并将其映射到[0, 255]
CumuPixel = cumsum(ProbPixel);  % 累积频率分布
CumuPixel = uint8(255 * CumuPixel + 0.5); % 映射到[0, 255]的整数值

% 直方图均衡化：将原始图像中的像素映射到均衡化后的灰度值
Img_eq = Img; % 创建均衡化后的图像
for i = 1:height
    for j = 1:width
        Img_eq(i,j) = CumuPixel(Img(i,j) + 1); % 利用 CDF 映射灰度值
    end
end

figure;
imshow(Img_eq); title('直方图均衡化后的图像');% 显示均衡化后的图像及其直方图

% 计算并显示均衡化后的直方图
[counts1_eq, x_eq] = imhist(Img_eq, 256);
counts2_eq = counts1_eq / (height * width); % 归一化直方图
figure;
stem(x_eq, counts2_eq); title('均衡化后图像的灰度直方图');
```

输出图像：


原始图像灰度直方图
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250
300


直方图均衡化后的图像


均衡化后图像的灰度直方图
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250
300



（2）离散傅里叶变换频谱幅度图显示以及同态滤波操作：

实现代码：

```
image = imread('D:\文档\数字图像处理\low_light_image.jpg'); % 读入低照度图像
image_gray = rgb2gray(image);  % 灰度化
[M, N] = size(image_gray);  % 获取图像大小

figure;
imshow(image_gray);
title('灰度图'); % 显示灰度图像

% 计算离散傅里叶变换并显示频谱幅度图
image_fft = fft2(double(image_gray));  % 进行傅里叶变换
image_fft_shift = fftshift(image_fft);  % 将低频移到中心
magnitude_spectrum = log(1 + abs(image_fft_shift));  % 取对数处理

figure;
imshow(magnitude_spectrum, []); % 显示频谱图
title('频谱幅度图');

% 同态滤波参数设置
rH = 0.2;        % 高频增益
rL = 0.2;    % 低频增益
c = 0.1;       % 影响频率范围的参数
D0 = 5;    % 截止频率

% 图像取对数处理，增强低亮度区域
img_log = log(double(image_gray) + 1);  % 加 1 防止对数为负
```

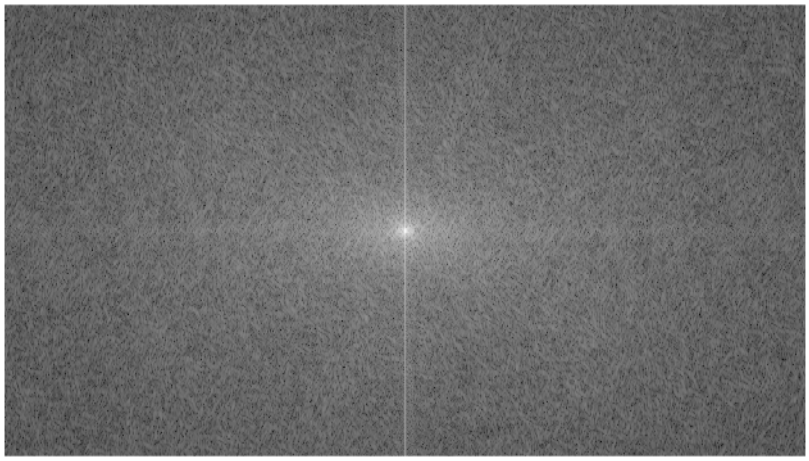
```
% 将图像平移到频域中心
img_shifted = zeros(M, N);
for i = 1:M
    for j = 1:N
        % 实现中心平移，交替正负 1
        img_shifted(i, j) = img_log(i, j) * (-1)^(i + j);
    end
end

img_fft = fft2(img_shifted); % 对平移后的图像进行傅里叶变换

% 构建同态滤波函数
img_filter = zeros(M, N);
deta_r = rH - rL;
D = D0^2;
m_mid = floor(M / 2);  % 图像中心点的坐标
n_mid = floor(N / 2);

for i = 1:M
    for j = 1:N
        dis = (i - m_mid)^2 + (j - n_mid)^2; % 计算距离中心点的欧氏距离
        img_filter(i, j) = deta_r * (1 - exp(-c * (dis / D))) + rL; % 构造同态滤波器的函数
    end
end

img_filtered_fft = img_fft .* img_filter;  % 应用同态滤波器
img_filtered = real(ifft2(img_filtered_fft)); % 对滤波后的图像进行傅里叶逆变换

% 将图像反平移回原位置
img_filtered_shifted = zeros(M, N);
for i = 1:M
    for j = 1:N
        img_filtered_shifted(i, j) = img_filtered(i, j) * (-1)^(i + j);
    end
end
img_exp = exp(img_filtered_shifted) - 1; % 取指数并恢复图像

% 图像归一化
max_val = max(img_exp(:));
min_val = min(img_exp(:));
img_normalized = uint8(255 * (img_exp - min_val) / (max_val - min_val));

figure;
```

```
imshow(img_normalized);
title('同态滤波后的图像'); % 显示同态滤波后的图像
```

输出图像：

灰度图像

频谱幅度图



同态滤波后的图像

（3）**最终效果对比：**

原图像：

处理后图像：

a. 比较：

从第一张图像来看，图像整体亮度有所提高，尤其是天空和树木的细节更加明显。然而，在低光区域如地面部分，噪声显著增加，这可能是由于过度拉伸某些灰度级引起的。从第二张图像可以看出，整体亮度变化不如直方图均衡化那么显著，但低光区域的细节被有效增强，且图像中噪声较少。尤其是地面部分，相比直方图均衡化后更加均匀和自然，光源附近的细节也处理得较好。

b. 分析：

同态滤波主要针对图像的频域处理，它同时增强了高频成分（细节部分）并压缩了低频成分（亮度）。其目的是改善图像对比度的同时，保留细节信息，特别是适合低照度场景的处理。

直方图均衡化通过调整图像的对比度，使得像素值更加均匀地分布在整个灰度范围内。其结果是图像的亮部和暗部细节得到了增强，亮度对比度有所提升，但有时会引入一些噪声和伪影，特别是在低照度场景中。

c. 总结：

综上，直方图均衡化增强了对比度，但在低光区域引入了明显噪声，适合用

于需要全局亮度提升的场景。同态滤波通过频域处理实现了局部增强，在抑制噪声的同时有效提升了图像的细节，特别适用于低照度场景的图像处理。