人工智能 2204 王子旭 U202215150

任务一

$$B=\begin{bmatrix}1&2&1&4&3\\1&10&2&3&4\\5&2&6&8&8\\5&5&7&0&8\\5&6&7&8&9\end{bmatrix}$$

$$g(m,n)=\frac{1}{9}\sum_{i\in Z}\sum_{j\in Z}f(m+i,n+j)\quad Z=\{-1,0,1\}$$

$g(1,1)=(1+2+1+1+10+2+5+2+6)/9=\frac{30}{9}\approx 3$

$g(1,2)=(2+1+4+10+2+3+2+6+8)/9=\frac{38}{9}\approx 4$

$g(1,3)=(1+4+3+2+3+4+6+8+8)/9=\frac{39}{9}\approx 4$

$g(2,1)=(1+10+2+5+2+6+5+5+7)/9=\frac{43}{9}\approx 5$

$g(2,2)=(10+2+3+2+6+8+5+7+0)/9=\frac{43}{9}\approx 5$

$g(2,3)=(2+3+4+6+8+8+7+0+8)/9=\frac{46}{9}\approx 5$

$g(3,1)=(5+2+6+5+5+7+5+6+7)/9=\frac{48}{9}\approx 5$

$g(3,2)=(2+6+8+5+7+0+6+7+8)/9=\frac{49}{9}\approx 5$

$g(3,3)=(6+8+8+7+0+8+7+8+9)/9=\frac{61}{9}\approx 7$

$$G=\begin{bmatrix}1&2&1&4&3\\1&3&4&4&4\\5&5&5&5&8\\5&5&5&7&8\\5&6&7&8&9\end{bmatrix}$$

$g(1,1)$: $\begin{bmatrix}1&2&1\\1&10&2\\5&2&6\end{bmatrix}$ 中值为 2　　$g(1,3)$ $\begin{bmatrix}1&4&3\\2&3&4\\6&8&8\end{bmatrix}$ 中值为 4

$g(1,2)$: $\begin{bmatrix}2&1&4\\10&2&3\\2&6&8\end{bmatrix}$ 中值为 3　　$g(2,1)$ $\begin{bmatrix}1&10&2\\5&2&6\\5&5&7\end{bmatrix}$ 中值为 5

$g(2,2)$: $\begin{bmatrix}10&2&3\\2&6&8\\5&7&0\end{bmatrix}$ 中值为 5　　$g(2,3)$ $\begin{bmatrix}2&3&4\\6&8&8\\7&0&8\end{bmatrix}$ 中值为 6

$g(3,1)$ $\begin{bmatrix}5&2&6\\5&5&7\\5&6&7\end{bmatrix}$ 中值为 5　　$g(3,2)$ $\begin{bmatrix}2&6&8\\5&7&0\\6&7&8\end{bmatrix}$ 中值为 6

$g(3,3)$ $\begin{bmatrix}6&8&8\\7&0&8\\7&8&9\end{bmatrix}$ 中值为 8

$$G=\begin{bmatrix}1&2&1&4&3\\1&2&3&4&4\\5&5&5&6&8\\5&5&6&8&8\\5&6&7&8&9\end{bmatrix}$$

任务二

2.1 计算并显示灰度直方图

2.1.1 实现步骤：

- 调用 MATLAB 库函数 rgb2gray 将图像灰度化
- 调用自己实现的 my_imhist 统计每种灰度值出现的次数，并返回概率质量函数
- 调用 MATLAB 的库函数 bar 进行绘制

2.1.2 结果：

图 1.图像灰度化结果

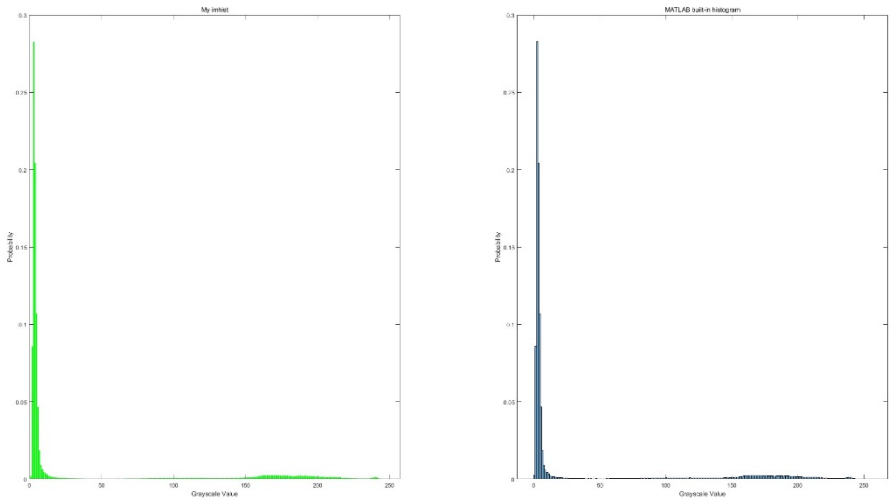


图 2. 灰度直方图(左)与 MATLAB 库函数 histogram(右)对比

对比发现自行实现的函数和 matlab 库函数区别不大。观察灰度直方图分布也能发现，大部分像素都集中在较低灰度级，也证明该图像低照度的情况

2.1.3 代码（分别为 rgbToGray 函数、my_imhist 函数和 main 函数）

```
function grayImage = rgbToGray(colorImage)
    % 获取图像的尺寸
    [rows, cols, ~] = size(colorImage);

    % 初始化灰度图像矩阵
    grayImage = zeros(rows, cols);

    % 使用加权平均计算每个像素的灰度值
    for row = 1:rows
        for col = 1:cols
            % 获取像素的 R、G、B 通道值
            pixel = colorImage(row, col, :);
            R = double(pixel(1));
            G = double(pixel(2));
            B = double(pixel(3));

            % 计算灰度值
            grayValue = 0.2989 * R + 0.5870 * G + 0.1140 * B;

            %imshow 能显示两大类型：当图像为 unit8 类型时，支持范围 0~255，图像为 double
类型时，支持范围 0~1
            grayValue = grayValue./255;
            % 将灰度值写入灰度图像矩阵
            grayImage(row, col) = grayValue;
        end
    end
    imwrite(grayImage,'E:\华中科技大学\大三\hw2\Output\gray1.jpg');
end

function pmf = my_imhist(I)
    % 检查图像是否为彩色图像，如果是则转换为灰度图像
    if ndims(I) == 3
        I = rgbToGray(I);
    end
    % 获取图像的尺寸
    [m, n] = size(I);
    % 初始化概率密度函数数组，长度为 256，对应 0 到 255 的灰度级
    pmf = zeros(1,256);
    % 计算每个灰度级像素值的频率
    for k = 0:255
        % 计算灰度级为 k 的像素数量，除以总像素数，得到归一化的频率
        pmf(k+1) = length(find(I==k))/(m*n);
    end
```

```
end

% 添加函数路径
addpath('E:\华中科技大学\大三\hw2\Functions\')

% 读取彩色图像
colorImage = imread('E:\华中科技大学\大三\hw2\Input\低照度图像.jpg');

%使用自行编写函数灰度化
grayImage = rgb2gray(colorImage);

%绘制出 matlab 库函数和自编函数的结果图
figure(1);
subplot(1, 2, 1), bar(my_imhist(grayImage), 'g')
title('My imhist')
xlabel('Grayscale Value')
ylabel('Probability')

subplot(1, 2, 2), histogram(grayImage, 256, 'Normalization', 'probability')
title('MATLAB built-in histogram')
xlabel('Grayscale Value')
ylabel('Probability')
```

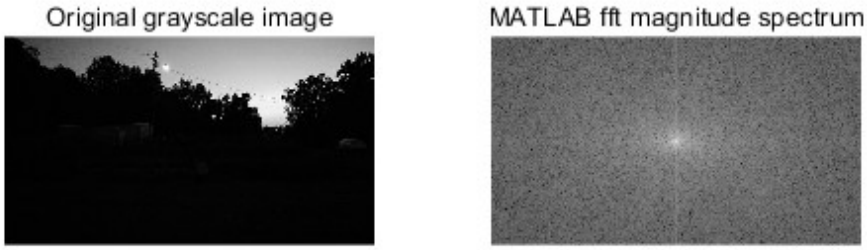
2.2 离散傅里叶频谱变换幅度图

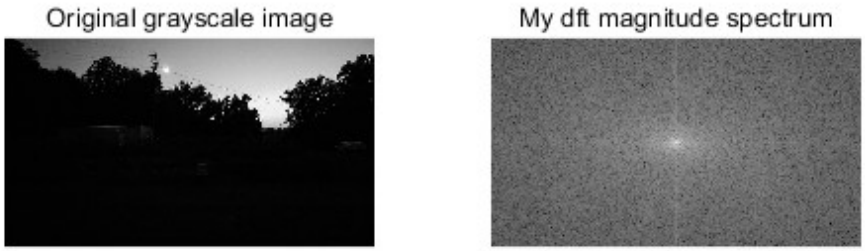
2.2.1 实现步骤：

- 将图像灰度化
- 调用自己实现的 my_dft 将计算过程分成两部分，分别对列和对行进行计算，并且采用向量化思想，不使用 for 循环，提高离散傅里叶变换的计算速度
- 调用 MATLAB 库函数 fftshift 将得到的离散傅里叶变换移动到中心，并取对数，计算幅度后调用 imshow 显示幅度谱

2.2.2 结果：

下图分别为灰度图像(左上、左下)、使用 matlab 库函数得到的频谱幅度图(右上)、自行编写函数得到的频谱幅度图(右下)。对比发现两个幅度图差别不大，证明我们实现的函数没有问题。观察频谱图也能发现，中心区即低频区域幅度较高，因为原图像灰度分布集中在低灰度级，整体为低照度。





2.2.3 代码：

```
function F_uv = my_dft(f_xy)
    % 检查图像是否为彩色图像，如果是则转换为灰度图像
    if ndims(f_xy) == 3
        f_xy = rgb2gray(f_xy);
    end

    % 将图像数据转换为 double 类型以便进行计算
    f_xy = double(f_xy);

    % 获取图像的尺寸
    [nrows, ncols] = size(f_xy);

    % 计算列方向的傅里叶变换
    vy = (0:nrows-1)' * (0:nrows-1); % 生成行索引和列索引矩阵
    M_vy = exp(-1i * 2 * pi * vy / nrows); % 计算列方向的傅里叶矩阵
    F_xv = M_vy * f_xy; % 进行列方向的傅里叶变换

    % 计算行方向的傅里叶变换
    ux = (0:ncols-1)' * (0:ncols-1); % 生成行索引和列索引矩阵
    M_ux = exp(-1i * 2 * pi * ux / ncols); % 计算行方向的傅里叶矩阵
    F_uv = F_xv * M_ux; % 进行行方向的傅里叶变换
end
% MATLAB 中的 fft 函数
```

```
F = fftshift(fft2(grayImage));
F_mag = abs(F);

% 自行实现的 dft 函数
img_dft2 = fftshift(my_dft(grayImage));
magnitude = abs(img_dft2);

% 显示结果图像
figure(2);
subplot(2, 2, 1),
    imshow(grayImage), title('Original grayscale image');
subplot(2, 2, 2),
    imshow(log(F_mag+1), []), title('MATLAB fft magnitude spectrum');
subplot(2, 2, 3),
    imshow(grayImage), title('Original grayscale image');
subplot(2, 2, 4),
    imshow(log(magnitude+1), []), title('My dft magnitude spectrum');
```

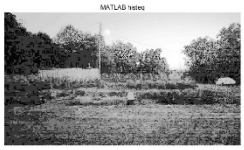
任务三

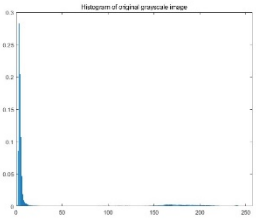
3.1 直方图均衡化

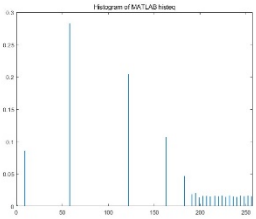
3.1.1 实现步骤：

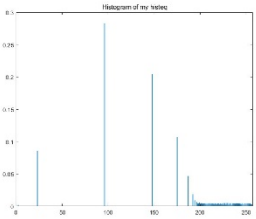
- 调用 MATLAB 库函数 rgb2gray 将图像灰度化
- 调用 my_imhist 计算输入灰度图的概率质量函数
- 调用 MATLAB 库函数 cumsum 计算累积分布函数并获得原图像灰度值与新图像灰度值映射关系
- 将获得的映射关系应用在图像上，并计算新图像的概率质量函数

3.1.2 实验结果：









与 MATLAB 库函数 histeq 对比，发现自己实现的算法与 histeq 有明显区别，直观上看

histeq 整体偏暗，并且灰度值 200 以上似乎进行了归并，没有像 my_histeq 那么密集。我推测可能是 MATLAB 对 histeq 进行了优化，也许是直方图规定化，也许是其他优化方法，这就使得 histeq 处理的结果噪声更小，尤其是在边缘上的细节更加清楚。

还可以看到，虽然直方图均衡化使得该低照度图像的内容能被人眼辨认，但是也同时放大了噪声，效果并不是很理想

直方图均衡化的一个优点是参数可以自适应，即不需要调节参数，算法会根据图像的不同而自动调节

3.1.3 代码：

```
function [I_eq, I_eq_pmf] = my_histeq(I)

    % 如果图像是彩色的，则转换为灰度图像
    if ndims(I) == 3
        I = rgb2gray(I);
    end
    [m, n] = size(I);  % 获取图像的尺寸

    % 调用自定义的 my_imhist 函数计算原图像的概率密度函数（PMF）
    pmf = my_imhist(I);

    % 计算累计分布函数（CDF）
    cdf = cumsum(pmf);

    % 将 CDF 映射到 0-255 之间的整数值
    s = round(cdf * 255);

    % 应用映射关系，生成均衡化后的图像
    I_eq = I;
    for i = 0:255
        I_eq(I == i) = s(i + 1);
    end

    % 计算均衡化后图像的概率密度函数（PMF）
    I_eq_pmf = zeros(1, 256);
    for k = 0:255
```

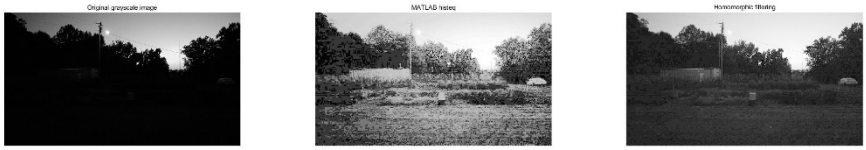
```
            I_eq_pmf(k+1) = length(find(I_eq == k)) / (m * n);
        end
    end

    [I_eq, I_eq_pdf] = my_histeq(grayImage);
    % MATLAB 中的直方图均衡化函数
    matlab_ans = histeq(grayImage);

    figure(3);
    subplot(2, 3, 1)
    imshow(grayImage)
    title('Original grayscale image')

    subplot(2, 3, 2)
    imshow(matlab_ans)
    title('MATLAB histeq')

    subplot(2, 3, 3)
    imshow(I_eq)
    title('My histeq')

    subplot(2, 3, 4)
    bar(my_imhist(grayImage))
    title('Histogram of original grayscale image')

    subplot(2, 3, 5)
    bar(my_imhist(matlab_ans))
    title('Histogram of MATLAB histeq')

    subplot(2, 3, 6)
    bar(my_imhist(I_eq))
    title('Histogram of my histeq')
```

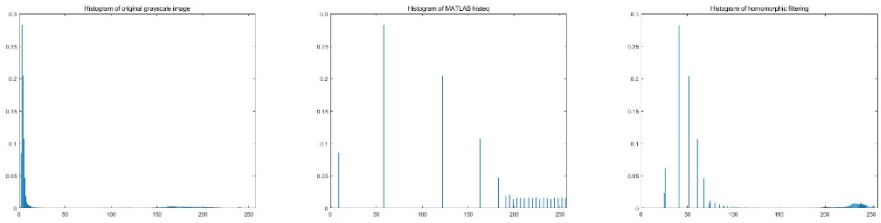
3.2 同态滤波

3.2.1 实验步骤：

- 调用 MATLAB 库函数 rgb2gray 将图像灰度化
- 取对数，并使用 my_dft 进行离散傅里叶变换
- 调用自己实现的高斯高通滤波 GHPF 并作用在经离散傅里叶变换的图像上
- 使用 my_idft 进行离散傅里叶逆变换，并取指数
- 重新调整成灰度图，使用 imshow 显示

3.2.2 实验结果：





观察发现，同态滤波的效果明显好过直方图均衡化，没有这么刺眼，也没有这么多噪声，图像中的细节也保留比较全面，但同态滤波的缺点是参数需要根据图像的不同来调节高通滤波器的参数，不能做到自适应

3.2.3 代码(包含 homo_fliter、GHPF、im_norm、my_idft)

```
function I_homo = homo_filter(I, rH, rL, c, D0)

    % 转化为灰度图像
    if ndims(I) == 3
        I = rgb2gray(I);
    end
    I = double(I);

    % 对数变换将乘法关系（图像中的光照和反射分量）转化为加法关系，从而简化后续的滤波过程。
    ln_I = log(I+1);
    % 使用自定义的 my_dft 函数对图像进行二维离散傅里叶变换（DFT），将图像转换到频率域。
    F = my_dft(ln_I);
    %调用自定义函数 GHPF 生成高斯高通滤波器，滤波器的作用是增强图像中的高频成分（细节），同时抑制低频成分（光照变化）。
    H = GHPF(I, rH, rL, c, D0);

    % 在频域中，将生成的滤波器 H 乘以傅里叶变换后的图像 F，从而过滤掉图像中的低频部分，保留高频部分。
    HF = H.*F;
    % 使用自定义的 my_idft 函数对滤波后的结果进行逆傅里叶变换，回到空间域。
    ln_hf = my_idft(HF);
    % 通过对数变换回到原始的尺度，使用指数函数进行恢复
```

```
    intermediate = exp(ln_hf)-1;
    % 对处理后的图像进行归一化，以确保像素值落在 [0, 255] 范围内。
    intermediate = im_norm(intermediate);
    % 将处理后的图像转换回 8 位无符号整数格式，并返回增强后的图像。
    I_homo = uint8(round(intermediate*255));
end

function H = GHPF(I, rH, rL, c, D0)
    % 转化为灰度图
    if ndims(I) == 3
        I = rgb2gray(I);
    end
    [M, N] = size(I);
    % 获取输入图像的尺寸 M 和 N，并计算图像中心位置 m 和 n
    m = floor(M/2);
    n = floor(N/2);
    % R 初始化一个与图像尺寸相同的滤波器矩阵 H，用于存储滤波器的值。
    H = zeros(M, N);
    % 使用双重 for 循环计算每个频率点 (u, v) 对应的距离 D(u, v) 和滤波器值 H(u, v)。
    for u = 1:M
        for v = 1:N
            D2_uv = (u-m)^2+(v-n)^2;
            H(u,v) = (rH-rL)*(1-exp(-c*(D2_uv/(D0^2))))+rL;
        end
    end
end

function I_norm = im_norm(I)
    % 确保图像数据是双精度浮点数，避免计算过程中出现整数截断
    I = double(I);
    % 使用 max 和 min 函数分别获取图像的最大像素值和最小像素值。
    I_max = max(I, [], 'all');
    I_min = min(I, [], 'all');
    % 将输入图像的像素值通过线性变换缩放到 [0, 1] 之间：
    I_norm = (I - I_min)/(I_max-I_min);
end

function f_xy = my_idft(F_uv)

    % 转化为灰度图
    if ndims(F_uv) == 3
        F_uv = rgb2gray(F_uv);
    end
    % 获取图像尺寸
```

```
    [nrows, ncols] = size(F_uv);
    % 进行行反傅里叶变换
    vy = (0:nrows-1)' * (0:nrows-1);
    M_vy = exp(1i*2*pi*vy/nrows);
    f_uy = M_vy * F_uv;
    % 进行列反傅里叶变换
    ux = (0:ncols-1)' * (0:ncols-1);
    M_ux = exp(1i*2*pi*ux/ncols);
    f_xy = real(f_uy * M_ux / (ncols*nrows));
end

% 设置同态滤波参数
rH = 2; rL = 0.1; c = 0.5; D0 = 10000;

% 同态滤波
I_homo = homo_filter(grayImage, rH, rL, c, D0);

%显示结果图
figure(5),
subplot(2, 3, 1)
imshow(grayImage)
title('Original grayscale image')

subplot(2, 3, 2)
imshow(matlab_ans)
title('MATLAB histeq')

subplot(2, 3, 3)
imshow(I_homo, [])
title('Homomorphic filtering')

subplot(2, 3, 4)
bar(my_imhist(grayImage))
title('Histogram of original grayscale image')

subplot(2, 3, 5)
bar(my_imhist(matlab_ans))
title('Histogram of MATLAB histeq')

subplot(2, 3, 6)
bar(my_imhist(I_homo))
title('Histogram of homomorphic filtering')
```