作业一

15331191 廖颖泓

1. 实现river.jpg图像的直方图均衡，不能直接使用Matlab的histeq( )函数。将有关均衡图像和调用histeq( )函数的结果作比较。

a.算法描述：

(1) 用Matlab的imread()函数扫描river.jpg整个图像，获取图像矩阵I；

(2) 用Matlab的size()函数获取图像矩阵I的行数m和列数n；

(3) 用循环的方法遍历图像矩阵I，计算每个灰度级对应的像素个数；

(4) 计算每个灰度级像素的频率作为出现的概率，绘制相应的直方图；

(5) 计算每个灰度级像素的概率累计分布函数cdf，将cdf乘上图像对应的灰度值范围的最大值，一般为255，取结果最接近的整数，记为该灰度级像素映射为均衡化后的像素灰度值，即h(v)

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$;

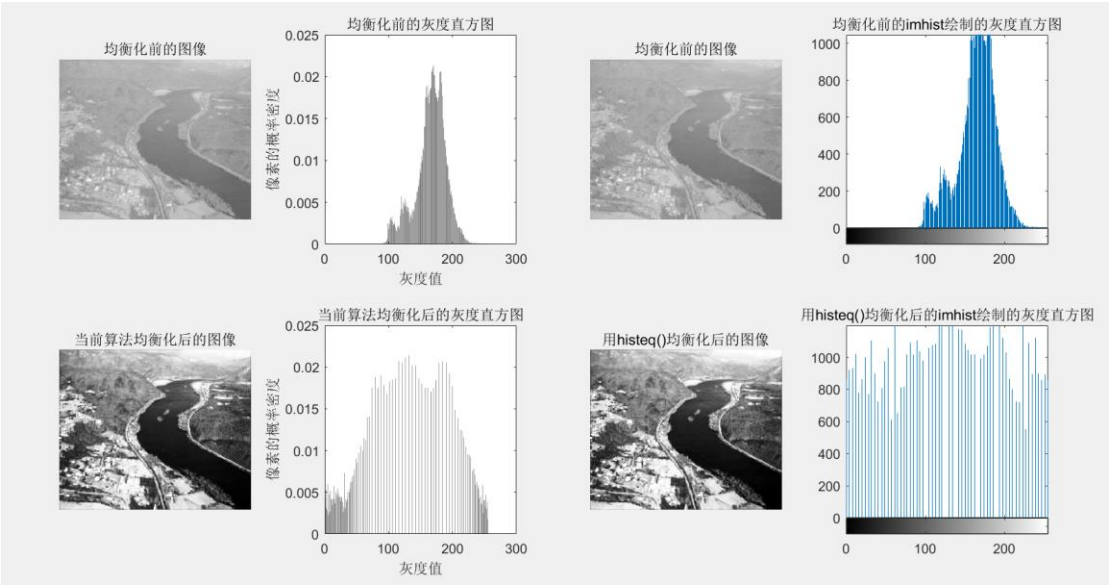
(6) 用循环的方法遍历图像矩阵I，将里面的像素灰度值i换为其均衡化映射对应的灰度值h(v_i);

(7) 计算新图像每个灰度级像素的频率作为出现的概率，绘制相应的直方图；

b.程序效果示意图

输入以下命令得到效果图1.png

```
>>    histogram_equalization('river.JPG')
```



1.png

c.与调用histeq( )函数结果的比较

调用histeq( )得到的图像与上述算法实现的均衡化图像效果近似，但是从直方图的效果来看，histeq( )均衡化的出来的直方图比上述算法得到的更加均衡，每个灰度值上的像素数量相差没有那么大，说明histeq( )在均衡化算法上进行了一定的优化，得到的效果更好。

d.程序代码

```
function histogram_equalization(input_img)
    % 读取输入图像获取图像矩阵
    I = imread(input_img);
    % 获取图像矩阵
    m = size(I,1);
    n = size(I,2);
    % 如果是彩色图像将其转换成灰度图像
    if ndims(I) == 3
        I = rgb2gray(I);
    end
    % 计算每个灰度级像素的个数
    num_pixels = zeros(1, 256);
    for i = 1:m
        for j = 1:n
            index = I(i, j) + 1;
            num_pixels(index) = num_pixels(index) + 1;
        end
    end
    % 计算灰度级像素在图像中出现的频率
    prob_pixels = num_pixels./(m * n);
    % 绘制图像和直方图
    figure;
    subplot(2,4,1),imshow(I),title('均衡化前的图像');
    subplot(2,4,2);
    bar(prob_pixels, 0.4);
    title('均衡化前的灰度直方图');
    xlabel('灰度值');
    ylabel('像素的概率密度');
    subplot(2,4,3);
    imshow(I);
    title('均衡化前的图像');
    subplot(2,4,4);
    imhist(I);
    title('均衡化前的imhist绘制的灰度直方图');
    % 计算灰度值的累计分布函数然后形成均衡化公式
    new_prob_indexes = zeros(1, 256);
    N = I;
    for i = 1:256
        for j = 1:i
            new_prob_indexes(i) = new_prob_indexes(i) + prob_pixels(j);
        end
        new_prob_indexes(i) =  new_prob_indexes(i) * 255;
    end
```

```
new_prob_indexes = round(new_prob_indexes);
% 对图像中每一个像素点进行均衡化映射
for i = 1:m
    for j = 1:n
        temp = N(i, j) + 1;
        N(i, j) = new_prob_indexes(temp) - 1;
    end
end
% 计算每个灰度级像素的个数
new_num_pixels = zeros(1, 256);
for i = 1:m
    for j = 1:n
        index = N(i, j) + 1;
        new_num_pixels(index) = new_num_pixels(index) + 1;
    end
end
%计算灰度级像素在图像中出现的频率
new_prob_pixels = new_num_pixels./(m * n);
% 绘制图像和直方图
subplot(2,4,5),imshow(N),title('当前算法均衡化后的图像');
subplot(2,4,6);
bar(new_prob_pixels, 0.4);
title('当前算法均衡化后的灰度直方图');
xlabel('灰度值');
ylabel('像素的概率密度');
subplot(2,4,7);
V = histeq(I);
imshow(V);
title('用histeq()均衡化后的图像');
subplot(2,4,8);
imhist(V);
title('用histeq()均衡化后的imhist()绘制的灰度直方图');
```

2. 将图像EightAM.png的直方图匹配为图像LENA.png的直方图，显示EightAM.png在直方图匹配前后的图像，并绘制LENA.png的直方图、直方图匹配前后EightAM.png的直方图，检查直方图匹配的效果。

   a.算法描述

   (1) 用上述均衡化的方法计算图像一每个灰度值对应的的$h_1(v)$;

   (2) 用上述均衡化的方法计算图像二每个灰度值对应的的$h_2(v)$;

   (3) 对图像一的每个灰度值v_i，在图像二中找到满足$h_1(v_i) = h_2(v_j)$所有灰度值v_j中的最小值v_{jmin}，构建匹配映射$mapping(v_i) = v_{jmin}$;

   (4) 用循环的方法遍历图像一矩阵I_1，将里面的像素灰度值换为其均衡化映射对应的灰度值$mapping(v_i) = v_{jmin}$;

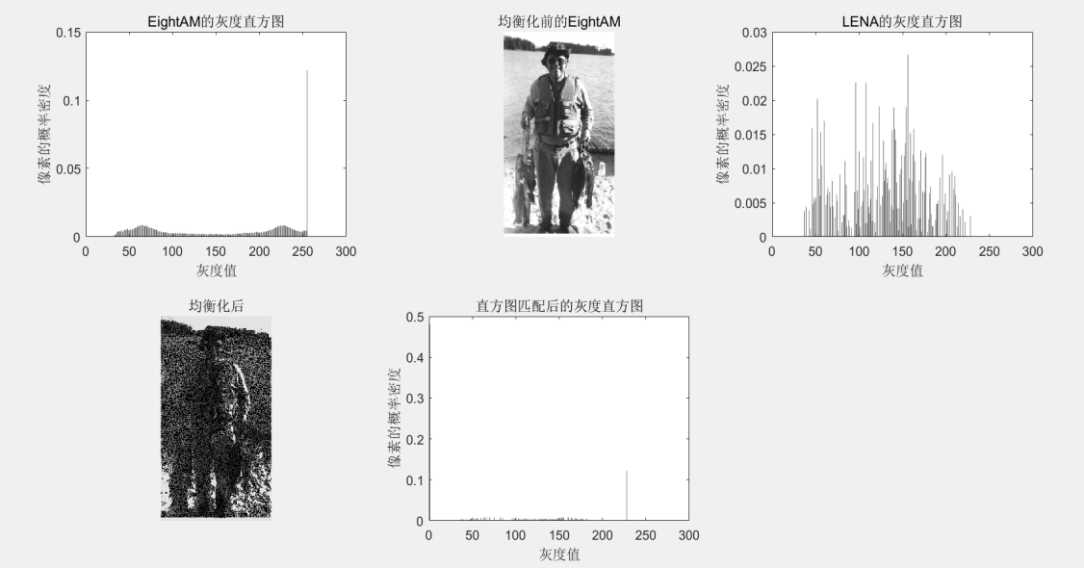
   (5) 计算新图像每个灰度级像素的频率作为出现的概率，绘制相应的直方

图；

b.程序效果示意图

输入以下命令得到效果图2.png

>> histogram_matching('EightAM.png','LENA.png')



2.png(注意纵坐标的大小)

c.检查直方图匹配的效果

进行直方图匹配后的图像与原图像相比，布满了更多的黑点，说明低灰度值像素点的数量偏多，匹配出的图像质量较差。从直方图的效果上看，LENA的灰度直方图在灰度50~225的像素点比较多，匹配出的EightAM图像在50~225的像素点也比较多。虽然显示的直方图感觉像素数量很少，但是根据纵坐标换算，得到的概率密度的大小是近似的，所以在一定程度上做到了直方图匹配。但是得出来的图像效果让人失望，需要更深入的知识对直方图匹配算法进行优化。

d.程序代码

```
function histogram_matching(input_img1, input_img2)
    % 读取输入图像获取图像矩阵
    I1 = imread(input_img1);
    I2 = imread(input_img2);
    % 获取图像矩阵
    m1 = size(I1,1);
    n1 = size(I1,2);
    m2 = size(I2,1);
    n2 = size(I2,2);
    % 如果是彩色图像将其转换成灰度图像
    if ndims(I1) == 3
        I1 = rgb2gray(I1);
    end
    if ndims(I2) == 3
        I2 = rgb2gray(I2);
    end
```

```
    % 计算每个灰度级像素的个数
    num_pixels_img1 = zeros(1, 256);
    num_pixels_img2 = zeros(1, 256);
    for i = 1:m1
        for j = 1:n1
            index = I1(i, j) + 1;
            num_pixels_img1(index) = num_pixels_img1(index) + 1;
        end
    end
    for i = 1:m2
        for j = 1:n2
            index = I2(i, j) + 1;
            num_pixels_img2(index) = num_pixels_img2(index) + 1;
        end
    end
    % 计算灰度级像素在图像中出现的频率
    prob_pixels_img1 = num_pixels_img1./(m1 * n1);
    prob_pixels_img2 = num_pixels_img2./(m2 * n2);
    figure;
    subplot(2,3,1);
    bar(prob_pixels_img1, 0.4);
    title('EightAM的灰度直方图');
    xlabel('灰度值');
    ylabel('像素的概率密度');
    subplot(2,3,2);
    imshow(I1);
    title('均衡化前的EightAM');
    subplot(2,3,3);
    imhist(I2);
    bar(prob_pixels_img2, 0.4);
    title('LENA的灰度直方图');
    xlabel('灰度值');
    ylabel('像素的概率密度');
    % 均衡化
    new_prob_indexes_img1 = zeros(1, 256);
    for i = 1:256
        for j = 1:i
            new_prob_indexes_img1(i) = new_prob_indexes_img1(i) +
prob_pixels_img1(j);
        end
        new_prob_indexes_img1(i) =  new_prob_indexes_img1(i) * 255;
    end
    new_prob_indexes_img1 = round(new_prob_indexes_img1);
    new_prob_indexes_img2 = zeros(1, 256);
```

```
    for i = 1:256
        for j = 1:i
            new_prob_indexes_img2(i) = new_prob_indexes_img2(i) +
prob_pixels_img2(j);
        end
        new_prob_indexes_img2(i) =  new_prob_indexes_img2(i) * 255;
    end
    new_prob_indexes_img2 = round(new_prob_indexes_img2);
    % 找到映射关系
    mapping = ones(1, 256);
    for i = 1:256
        for j = 1:256
            if new_prob_indexes_img2(j) == new_prob_indexes_img1(i)
                mapping(i) = j;
                break;
            end
        end
    end
    % 进行直方图匹配
    N = I1;
    for i = 1:m1
        for j = 1:n1
            temp = N(i, j) + 1;
            N(i, j) = mapping(temp) - 1;
        end
    end
    % 计算每个灰度级像素的个数
    new_num_pixels = zeros(1, 256);
    for i = 1:m1
        for j = 1:n1
            index = N(i, j) + 1;
            new_num_pixels(index) = new_num_pixels(index) + 1;
        end
    end
    % 计算灰度级像素在图像中出现的频率
    new_prob_pixels = new_num_pixels./(m1 * n1);
    % 打印原图和直方图
    subplot(2,3,4),imshow(N),title('均衡化后');
    subplot(2,3,5);
    bar(new_prob_pixels, 0.4);
    title('直方图匹配后的灰度直方图');
    xlabel('灰度值');
    ylabel('像素的概率密度');
```