ECE253 HW2

Name: Zhouhang Shao
PID: A99086018

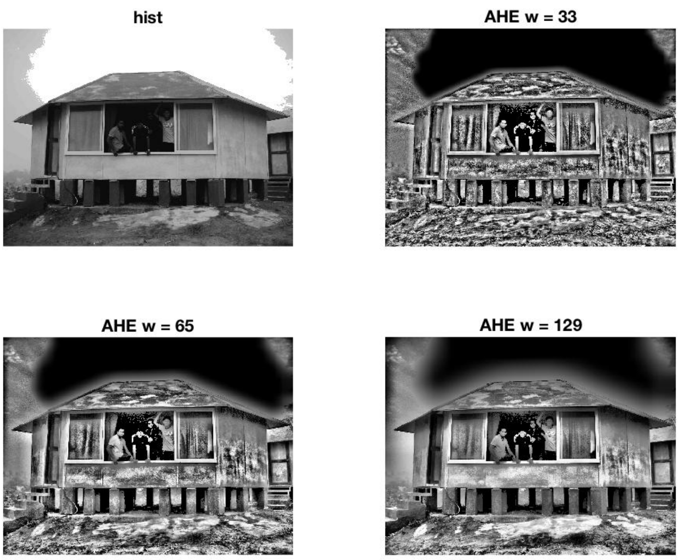
Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind. By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

Problem 1: Adaptive histogram equalization

The resulting image with different window is size is attached below

- The original image

- Image after global histogram equlization and AHE



- **How does the original image qualitatively compare to the image after AHE and HE respectively.**

  HE and AHE both enhance the contrast of the original image. We can observe more details in the enhanced images. For all the image after enhancement, they look brighter compared to the original one.However, all the images after histogram equalization seems to be unreal compared to the original one

- **Which strategy (AHE or HE) works best for beach.png and why? Is this true for any image in general?**

  AHE works better on the beach.png. I think the reason probably would be that the image contains regions that are significantly lighter or darker than than most of the image. Thus, the normal histogram equalization does not work very well.

  This conclusion will not always hold in general. Since AHE only operates on a small region, when the pixels in the neighborhood are very similar, the mapping function will map the narrow range of pixel values to the entire intensity range in the resulting image. In this case, AHE tends to enhance subtle area but are very sensitive to the noise. Thus, it will not work very well for the noisy image.

Source Code

- **AHE.m(the adaptive histogram equalization function)**

```
% This function is designed to perform the adaptive histgram equalization
% Before the operation, it will first pad the image based on the window
% size
% for each pixel, it will perform histogram equalization around the certain
% regions. The region is defined to be a square , it's size always equals
% to win_size
function [output] = AHE(image, win_size)
    %pad the image based on the window size
    pad_size = floor(win_size/2);
    paddedImage = padarray(image, [pad_size,pad_size], 'symmetric');
    [height,width] = size(paddedImage);
    output= uint8(zeros(size(image,1), size(image, 2)));
    %perform Adaptive histogram equalization
    for x =  1 + pad_size : height - pad_size
        for y = 1 + pad_size  : width - pad_size
            rank = 0;
            %iterate through the window around certer pixel
            for i = x - pad_size :  x + pad_size
                for j = y - pad_size : y + pad_size
                    if paddedImage(x,y) > paddedImage(i,j)
                        rank = rank + 1;
                    end
                end
            end
            intensity = 255 * (rank/(win_size * win_size));
            output(x- pad_size, y - pad_size) = intensity;
        end
    end
end
```

- **P1.m(script for problem 1)**

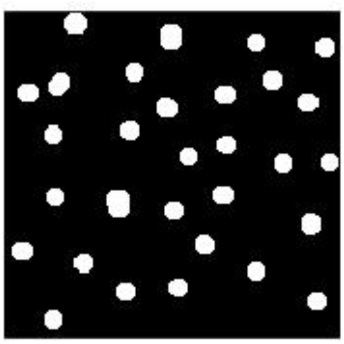
```
image = imread('beach.png');
imshow(image);
win_size = 129;
pad_size = floor(win_size/2);
disp(pad_size);
paddedImage = padarray(image,[pad_size,pad_size],'symmetric');
[height,width] = size(paddedImage);
output= uint8(zeros(size(image,1), size(image, 2)));

padImage33 = AHE(image,33);
padImage65 = AHE(image, 65);
padImage129 = AHE(image, 129);
histEq = histeq(image);
figure, subplot(2,2,1)
imshow(histEq);
title('hist')
subplot(2,2,2);
imshow(padImage33);
title('AHE w = 33')
subplot(2,2,3);
imshow(padImage65);
title('AHE w = 65')
subplot(2,2,4);
imshow(padImage129);
title('AHE w = 129')
```

Problem 2: Binary Morphology

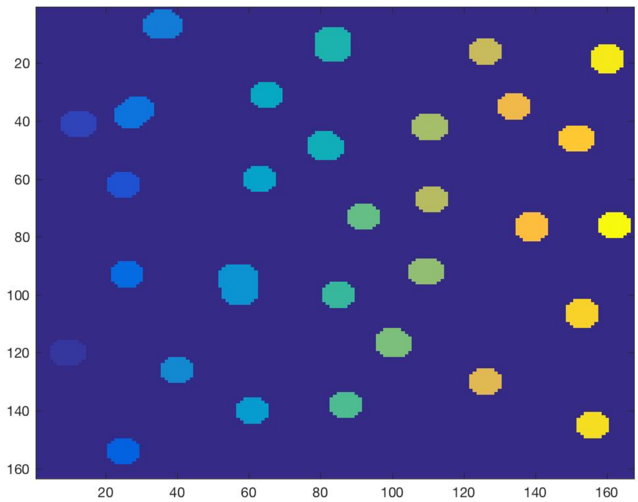
Saperate circles:

- **Structure element:** Disk with radius 5 pixel
- The original image

- The image after opening



* The image after connected componets labeling



* The following contains both the area and centroid data for each connected components

1	**2**	**3**	**4**	**5**	**6**
78	78	69	69	69	95
(120,10)	(41,13)	(62,25)	(154,25)	(93,26)	(37,28)
7	**8**	**9**	**10**	**11**	**12**
92	69	136	69	69	69
(7,36)	(126,40)	(96,57)	(140,61)	(60,63)	(31,65)
13	**14**	**15**	**16**	**17**	**18**
85	108	69	69	69	85
(49,81)	(14,84)	(100,85)	(138, 87)	(73,92)	(117,100)
19	**20**	**21**	**22**	**23**	**24**
78	78	69	69	69	69
(92,110)	(42,111)	(67,111)	(16,126)	(130,126)	(35,134)
25	**26**	**27**	**28**	**29**	**30**
78	78	78	69	78	69
(77,139)	(46,152)	(107,153)	(145,156)	(19,160)	(76,162)

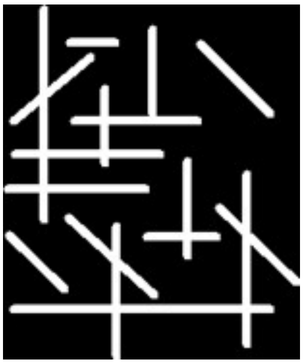
Source code

```
image = imread('circles_lines.jpg');
image = rgb2gray(image);        %do we need to convert to gray scale first?
image = imbinarize(image);
imshow(image);

SE = strel('disk',5);
circle = imopen(image,SE);
write = figure;
imshow(circle);
saveas(write,'circle_only.jpg')

%find the connected components
ccCircles = bwlabel(circle);
write = figure;
imagesc(ccCircles);
saveas(write,'connect_component_circles.jpg');

%calculate the area for each connected componenets
numElement = max(ccCircles(:));
circleArea = zeros(numElement,1);
circleCentroid = zeros(numElement,2);

%looping through the circle image and find the area for each connected
%componenets
```

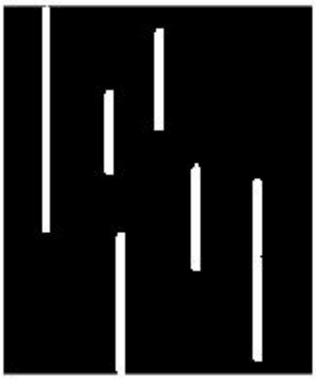
```
for x = 1 : size(ccCircles,1)
    for y = 1 : size(ccCircles, 2)
        if ccCircles(x,y) ~= 0
            circleArea(ccCircles(x,y),1) = circleArea(ccCircles(x,y),1) + 1;
            circleCentroid(ccCircles(x,y),1) =  circleCentroid(ccCircles(x,y),1) + x;
            circleCentroid(ccCircles(x,y),2) =  circleCentroid(ccCircles(x,y),2) + y;
        end
    end
end

% calculating the centroid
for i = 1 : numElement
   circleCentroid(i,1) =  circleCentroid(i,1)/circleArea(i,1);
   circleCentroid(i,2) =  circleCentroid(i,2)/circleArea(i,1);
end
```

Seperate lines:

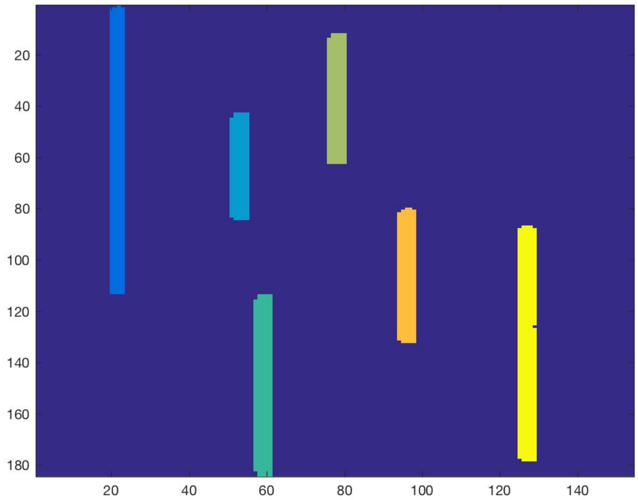
- **Structure element:** line with radius 20 pixels and the degree is 90 degree
- The original image



* The image after opening



- The image after connected componets labeling



- Centroid for each vertical line

1	2	3	4	5	6
(57,22)	(64,53)	(149,59)	(37,78)	(106,96)	(133,127)

- Length of each vertical line

1	2	3	4	5	6
112	41	70	50	52	91

Source code:

```
imageLine = imread('lines.jpg');
imageLine = imbinarize(rgb2gray(imageLine),0.4);
SE = strel('line',20,90);
line = imopen(imageLine,SE);
write = figure;
imshow(line);
saveas(write, 'line_only.jpg');

ccLines = bwlabel(line);
write = figure;
imagesc(ccLines);
saveas(write, 'connected_component_lines.jpg');

%calculate the area for each connected componenets
numElement = max(ccLines(:));
lineArea = zeros(numElement,1);
lineLength = zeros(numElement,3);
lineCentroid = zeros(numElement,2);

%looping through the line image and find the area for each connected
%componenets
for x = 1 : size(ccLines,1)
    for y = 1 : size(ccLines, 2)
        if ccLines(x,y) ~= 0
```

```
                lineArea(ccLines(x,y),1) = lineArea(ccLines(x,y),1) + 1;
                lineCentroid(ccLines(x,y),1) =  lineCentroid(ccLines(x,y),1) + x;
                lineCentroid(ccLines(x,y),2) =  lineCentroid(ccLines(x,y),2) + y;
                if lineLength(ccLines(x,y),1) == 0
                    lineLength(ccLines(x,y),1) = x;
                    lineLength(ccLines(x,y),2) = x;
                else
                    if(x < lineLength(ccLines(x,y),1))
                        lineLength(ccLines(x,y),1) = x;
                    else
                        lineLength(ccLines(x,y),2) = x;
                    end
                end
            end
        end
end
%
% calculating the length
for i = 1 : numElement
    lineLength(i,3) = lineLength(i,2) - lineLength(i,1); %calculate the length
    lineCentroid(i,1) =  lineCentroid(i,1)/lineArea(i,1); %calculate the centroid for x
    lineCentroid(i,2) =  lineCentroid(i,2)/lineArea(i,1); %calculate the centroid for y
end
```

Problem 3: Lloyd-Max Quantizer

i. UniformQuantizer function

```
function [compressedImage] = uniformQuantization(inputImage, s)
    inputImage = double(inputImage);
    interval = 2^8/2^s;
    [h,w] = size(inputImage);
    compressedImage = zeros(h,w);
    for x = 1 : size(inputImage,1)
        for y = 1 : size(inputImage,2)
            compressedImage(x,y) = floor(inputImage(x,y)/interval)*interval + 0.5 * interval;
        end
    end
    compressedImage = uint8(compressedImage);
end
```

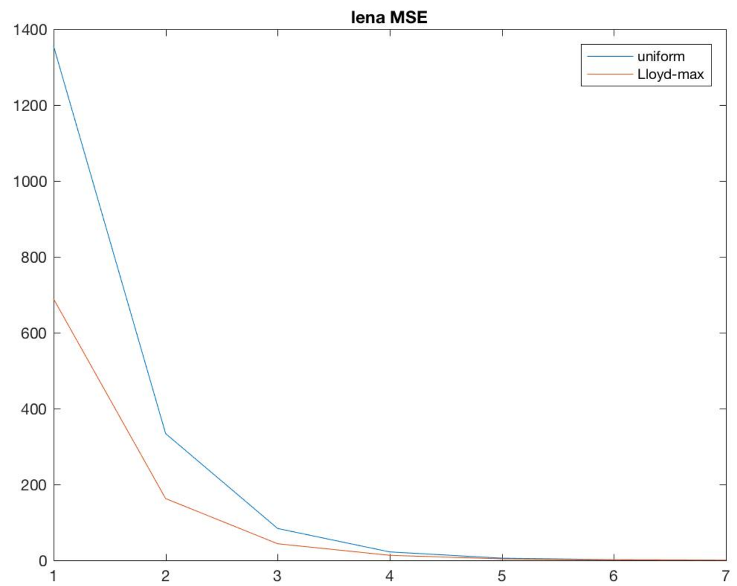
Plot: MSE verse num of bits

ii. UniformQuantization(blue) vs Lloyd-Max(yellow)

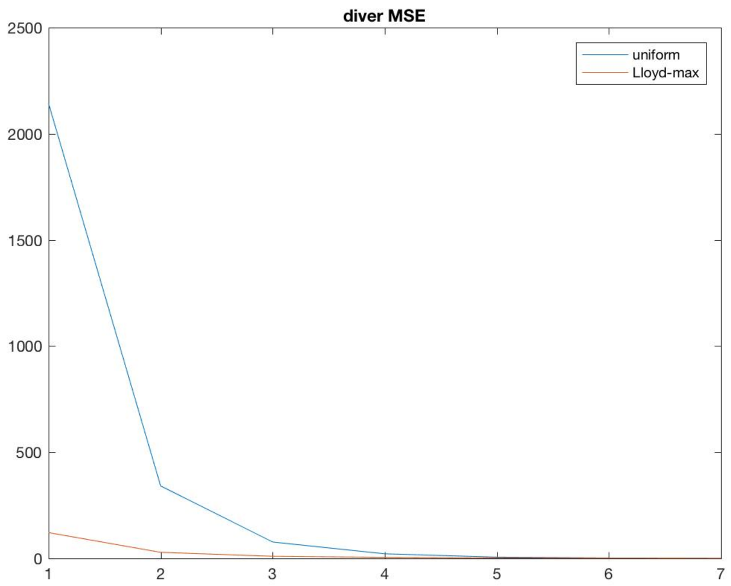
When using less than 4 bits, Lloyd's quantization method(with less MSE)performed much better compared to the uniformQuantization method. One reason for this performance difference might be Lloyd's method will dynamically determined the quantization intervals(bin size) based on the image's histogram while uniform histogram equalization uses fixed bin size.

As we used more and more bits, the MSE for the result images from both methods decrease dramatically. When we use 6 or 7 bits, the result images are almost indentical to the original image. In this case, both methods generate very good results with using more than 5 bits.

- Lena



- Diver



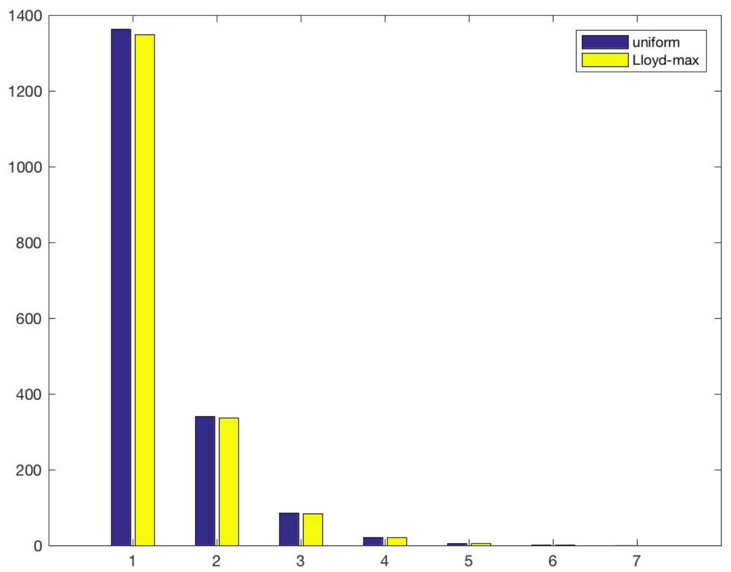
iii. UniformQuantization(blue) vs Lloyd-Max(yellow) after histogram equalization

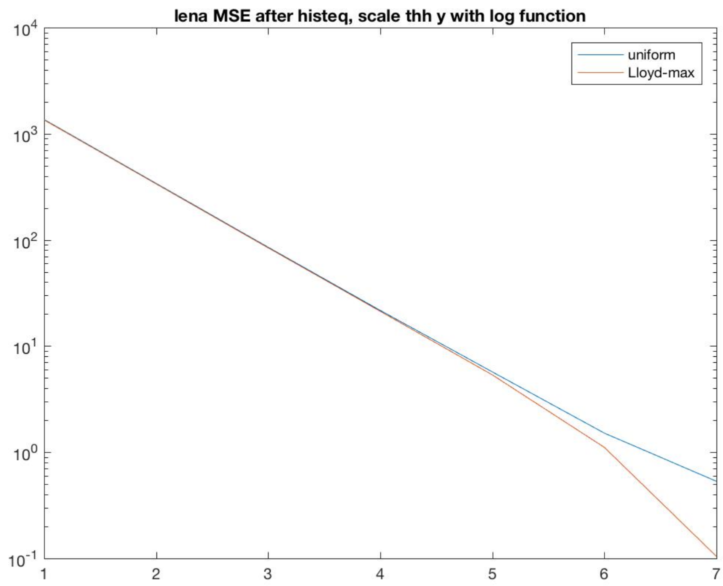
In general, when using the images after histogram equalization, both uniform's Quantization and Lloyds's Quantization peform better compared to the resulting when using non equalized image. This can be verified both visually and mathmatically. Th new images in general looks more brigher and with less MSE(when using less than 4 bits). However, when using the 6 or 7 bits, there is no big performance gap found. Both the equalized and non_equalized image will have very great image quality for both quantization methods.

In terms of the performance differences between Uniform Quantization and Lloyd-Max quantization, unlike the big performance gaps when using with unequalized images(Lloyd's quantiztion perform significantly better), both methods generate very similar results, almost identical.

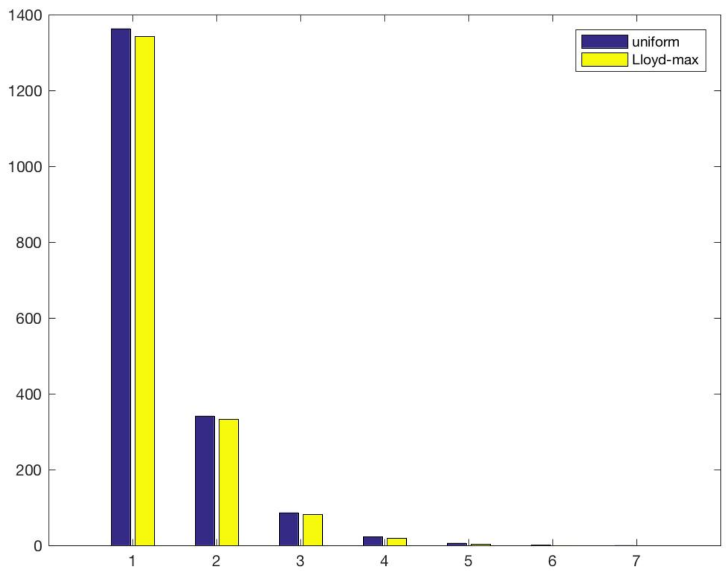
For this problem, since the two results are very close, I use the semilog y function in matlab to scale the y value. The result indicates that the Lloyd's method still perform slightly better compared to the uniform quantization.

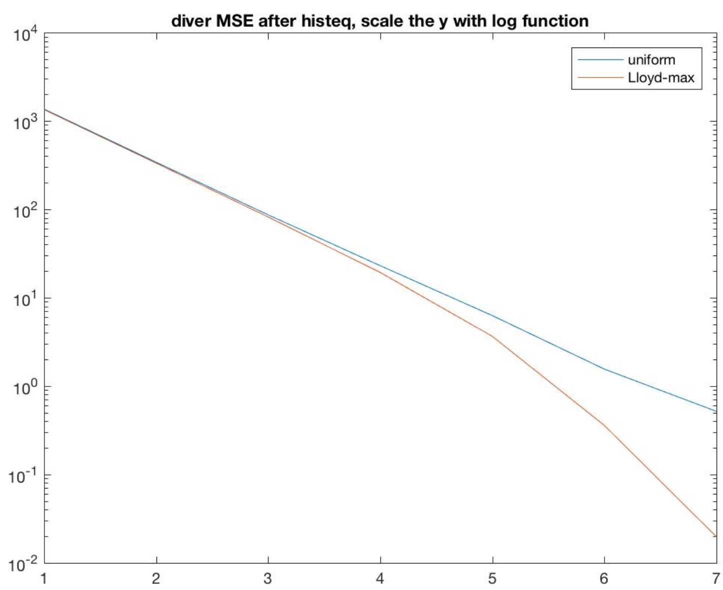
- Lena




lena MSE after histeq, scale thh y with log function
uniform
Lloyd-max
10^4
10^3
10^2
10^1
10^0
10^-1
1
2
3
4
5
6
7

- Diver


uniform
Lloyd-max
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
6
7


diver MSE after histeq, scale the y with log function
uniform
Lloyd-max
10^4
10^3
10^2
10^1
10^0
10^-1
10^-2
1
2
3
4
5
6
7

Explain the performance after histogram equalization

As for why the MSE is very small in general for Lloyd's method when using 7 bit, I think that is because, when using 7 bit, each inteval(bin size) is very small. When we estimating the pixel value during quantization, the new pixel value will be very close to the original one. This can also explains that when we use less bits, the estimated pixel values become inaccuarte as the intevals used in the quantization process become comparatively large.

As for why the histogram equalization make the results for both methods very similar. I think this can be explained by the fact that histogram equalization can be considered as a uniform transformation. In other word, it makes the histogram of the new image more like a uniform distribution. As we know, the Lloyd's method will determine of it's intevals based on the histogram of a image while uniform quantization divides the intervals evenly. That is to say, for the original image, the two methods can have very different intervals and bin sizes, thus different pixel estimations. However, after histogram euqualization, for a new uniform liked pixel distribution, Lloyd and uniform quantizations will divide the intervals and esitmiate pixel value in very similar ways, thus generate almost indentical results as shown in the above graphs.

Source code:

```
lena = imread('lena512.tif');
diver = imread('diver.tif');

figure, imshow(lena);
figure, imshow(diver);
figure,imshow(uniformQuantization(lena,4));
figure, imshow(uniformQuantization(diver,4));

x = 1 : 1 :7;
[lenaUniformMSE, lenaLloydMSE] = calculateMSE(lena);
bar_pic = figure;
plot(x, [lenaUniformMSE,lenaLloydMSE]);
title('lena MSE');
legend('uniform', 'Lloyd-max');
saveas(bar_pic ,'p3_lena_bar_MSE.jpg')
%
[diverUniformMSE, diverLloydMSE] = calculateMSE(diver);
bar_pic = figure;
plot(x, [diverUniformMSE, diverLloydMSE]);
title('diver MSE');
legend('uniform', 'Lloyd-max');
saveas(bar_pic ,'p3_diver_bar_MSE.jpg')

lenahistEq = histeq(lena,256);
diverhistEq = histeq(diver,256);
[histLenaUniformMSE, histLenaLloydMSE] = calculateMSE(lenahistEq);
bar_pic = figure;
semilogy(x, [histLenaUniformMSE, histLenaLloydMSE]);
title('lena MSE after histeq, scale thh y with log function');
legend('uniform', 'Lloyd-max');
saveas(bar_pic ,'p3_lena_bar_hist_MSE.jpg')
[histDiverUniformMSE, histDiverLloydMSE] = calculateMSE(diverhistEq);
bar_pic = figure;
semilogy(x, [histDiverUniformMSE,histDiverLloydMSE]);
title('diver MSE after histeq, scale the y with log function');
legend('uniform', 'Lloyd-max');
```

```
saveas(bar_pic ,'p3_diver_bar_hist_MSE.jpg')
```

```
function [uniformMSE, lloydMSE] = calculateMSE(inputImage)
    image = double(inputImage);
    [r1,c1] = size(image);

    uniformMSE = zeros(7,1);
    lloydMSE =zeros(7,1);
    trainSet = reshape(image, r1*c1, 1);
    for i = 1 : 7
        % perform the uniform Quantization for Lena
        uniImage = uniformQuantization(image,i);
        diff = (double(uniImage) - image).^2;
        uniformMSE(i,1) = sum(diff(:))/(r1*c1);

        % perform the Lloyds for Lena and calcuate
        [partition, codebook] = lloyds(trainSet, 2^i);
        Lloyds = image;
        newPartition = double(zeros(2^i+1,1));
        newPartition(2:2^i,1) = partition;
        newPartition(1,1) = 0;
        newPartition(2^i+1,1) = 255;
        for x = 1 : 2^i
           Lloyds(image >= newPartition(x,1) & image < newPartition(x+1,1)) = codebook(x);
        end


        diff = (Lloyds - image).^2;
        lloydMSE(i,1) = sum(diff(:)/(r1*c1));

    end
end
```