University of Lincoln

School of Computer Science

CMP3108M - Image Processing

Week-4: Image Enhancement & Intensity Transformation

1. MATLAB

In each section, a few new MATLAB built-in functions are introduced. To know more about these functions, you can Google the function name together with the keyword MATLAB to find the relevant MathWorks documentation, and read the instructions. For example: '*rgb2gray* MATLAB'.

2. Histogram

An image histogram is a type of histogram that acts as a graphical representation of the intensity distribution in a digital image. It plots the number of pixels for each intensity value. By looking at the histogram for a specific image a viewer will be able to judge the entire intensity distribution at a glance. Have a look at the lecture notes for week-3.

2.1 Task

Download the "SampleImages" zip file, un-compress the file, and save the files in your working folder. Then write a MATLAB script which loads the input grey-scale image 'small.png'. If you want to display the image, you can use the function *imagesc* instead of *imshow*. This is because *imshow* assumes a range of [0,255] for the intensity values, and displays the image at 100% magnification (one screen pixel for each image pixel). This makes it difficult for visual inspection of the images with low intensity values or small sizes. First display the loaded image using *imshow* as:

```
imshow(I);
colorbar;
colormap gray;
```

Then try using the function *imagesc* and see the difference:

```
imagesc(I);
colorbar;
colormap gray;
```

Find the range of grey-scale intensity values in the image. You can obtain this by finding the minimum and maximum pixel values:

- `mi = min(min(I));`
- `ma = max(max(I));`

The number of discrete grey levels will, therefore, be L = ma – mi + 1;

Now you can compute the histogram of the image using a 'for loop' such as:

```
for i = 1:L
    pixel_value(i) = i - 1;
    frequency = find( I == pixel_value(i) );
    Nk(i) = length( frequency );
end
```

You can then plot the computed histogram using the function *bar*:

```
bar(pixel_value,Nk,0.1);
```

Here, 0.1 is a scaling value which defines the width of the bars. Save the script you have written to compute the histogram for latter stages.

→ new MATLAB functions used in this section: *imagesc*, *colorbar*, *colormap*, *min*, *max*, *find*, *length*, *bar*

3. Contrast Stretching

Enhancements are used to make it easier for visual interpretation and understanding of image. By manipulating the range of digital values in an image, graphically represented by its histogram, we can apply various enhancements to the data. There are many different techniques and methods of enhancing contrast and detail in an image; we will cover only a few common ones here.

The simplest type of enhancement is a **linear contrast stretch**. This involves identifying lower and upper bounds from the histogram (usually the minimum and maximum brightness values in the image) and applying a transformation to stretch this range to fill the full range.

In the example shown below, the minimum value in the histogram is 84 and the maximum value is 153. These 70 levels occupy less than one-third of the full 256 levels available. A linear stretch uniformly expands this small range to cover the full range of values from 0 to 255. This enhances the contrast in the image with light toned areas appearing lighter and dark areas appearing darker, making visual interpretation much easier.

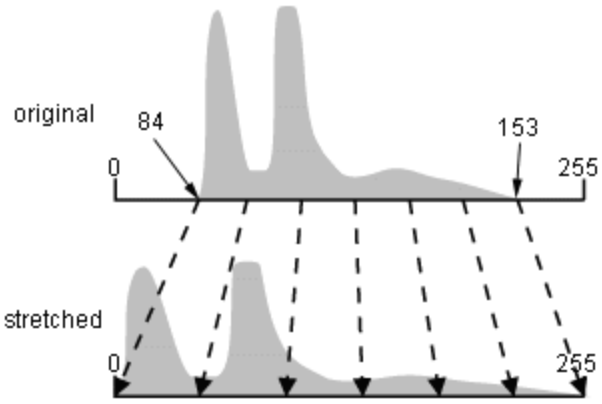
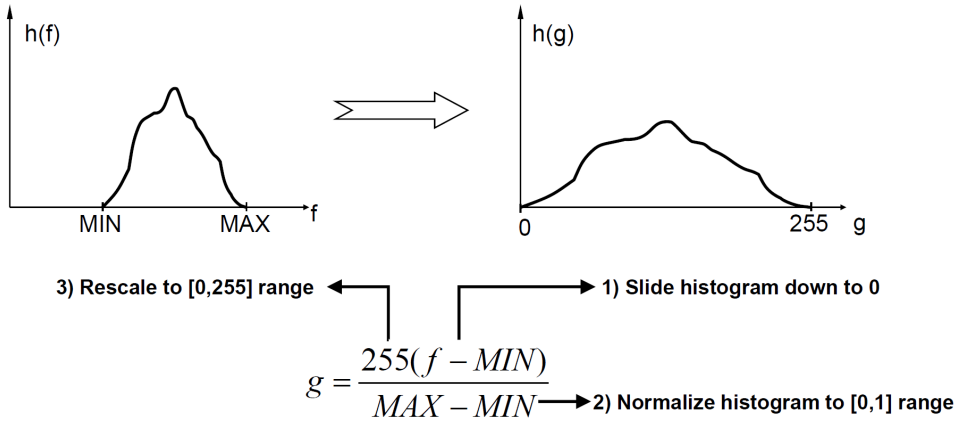
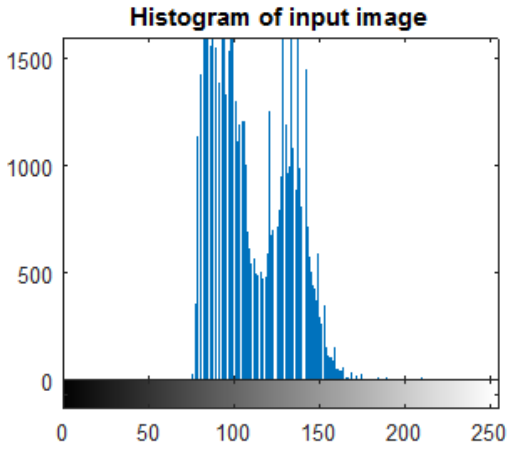


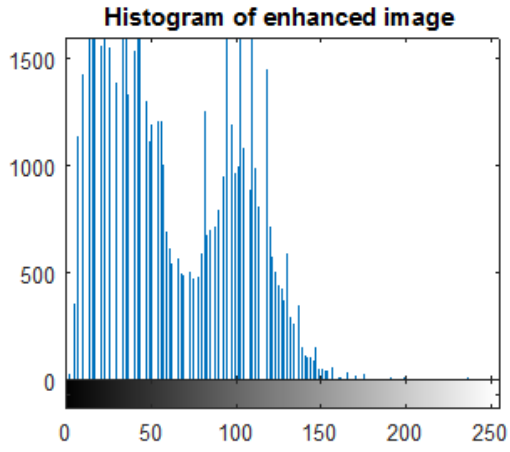
Figure 1. Linear contrast stretch.

3.1 Task

Write a MATLAB script for linear contrast stretching which loads the input image. Then performs a linear stretch by uniformly expanding the range of the pixel values in the input image to cover the full range of values from 0 to 255. The formula to carry out such operation is shown below:







Tip: you will need to follow the steps provided in the following pipeline:

- convert the uint8 input image to double precision
- find the maximum and minimum pixel values in the input image

- transform the image
- convert the transformed image back to uint8

```
I = rgb2gray(I);
J = 255*im2double(I); % converts the intensity image I to double precision
mi = min(min(J)); % find the minimum pixel intensity
ma = max(max(J)); % find the maximum pixel intensity

for row = 1 :size(J,1)
    for col = 1:size(J,2)
        J1(row,col) = ???? you will need to complete this section
    end
end

J1 = im2uint8(J1/255); % converts the grayscale image I to uint8
```

Now use the MATLAB built-in function *imadjust* to perform the same task:

```
J2 = imadjust(I,[low_in; high_in],[low_out; high_out]);
```

Two parameters 'low_out' and 'high_out' are the contrast limits for the output image with normalised values between 0 and 1. Two parameters 'low_in' and 'high_in' are the contrast limits for the input image which are also normalised by 255. You can use the previously calculated minimum and maximum values here. For example:

```
J2 = imadjust(I,[mi/255; ma/255],[0; 1]);
```

Check whether you get similar results by subtracting the two enhanced images and plotting the difference image:

```
Z = imsubtract(J1,J2);
imshow(Z);
```

The main MATLAB function for dealing with image histograms is *imhist*. Now that you have learnt how to compute the histogram for an image in section 2.1, you can start using this function as:

```
imhist(I,n)
```

Here, *n* specifies the number of bins used in the histogram. Use subplot to display the results in one output figure which should include the original image, a plot of its histogram, the enhanced image, and a plot of its histogram.

→ new MATLAB functions used in this section: *im2double*, *im2uint8*, *imadjust*, *imsubtract*, *imhist*

4. Histogram Equalisation

A uniform distribution of the input range of values across the full range may not always be an appropriate enhancement, particularly if the input range is not uniformly distributed. In this case, a **histogram-equalised stretch** may be better. This stretch assigns more display values (range) to the

frequently occurring portions of the histogram. In this way, the detail in these areas will be better enhanced relative to those areas of the original histogram where values occur less frequently.

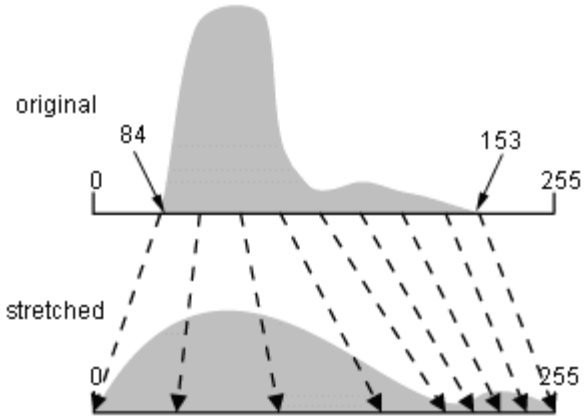


Figure 2. Histogram Equalisation.

4.1 Task

You previously computed the histogram for the input image 'small.png'. Now we are interested in producing the histogram equalised output image. First, complete the table provided below (lecture notes from week-3 may be helpful here).

Tip: To calculate the cumulative sum, you can either use a 'for loop' or use the built-in function *cumsum*. Also to round a value to its nearest integer, you can use the function *round*.

```
I = 255*im2double(I);
mi = min(min(I));
ma = max(max(I));
L = ma - mi + 1;
for i = 1:L
    pixel_value(i) = i - 1;
    frequency = find( I == pixel_value(i) );
    Nk(i) = length( frequency );
end
```

r_k	n_k	$p_r(r_k) = \frac{n_k}{MN}$	$\sum_{j=0}^{k} p_r(r_j)$	$T(r_k)$	S_k
0	135	0.054	0.054	0.378	0
1	473	0.189	0.243	1.701	2
2	626				
3					
4					
5					
6					
7					

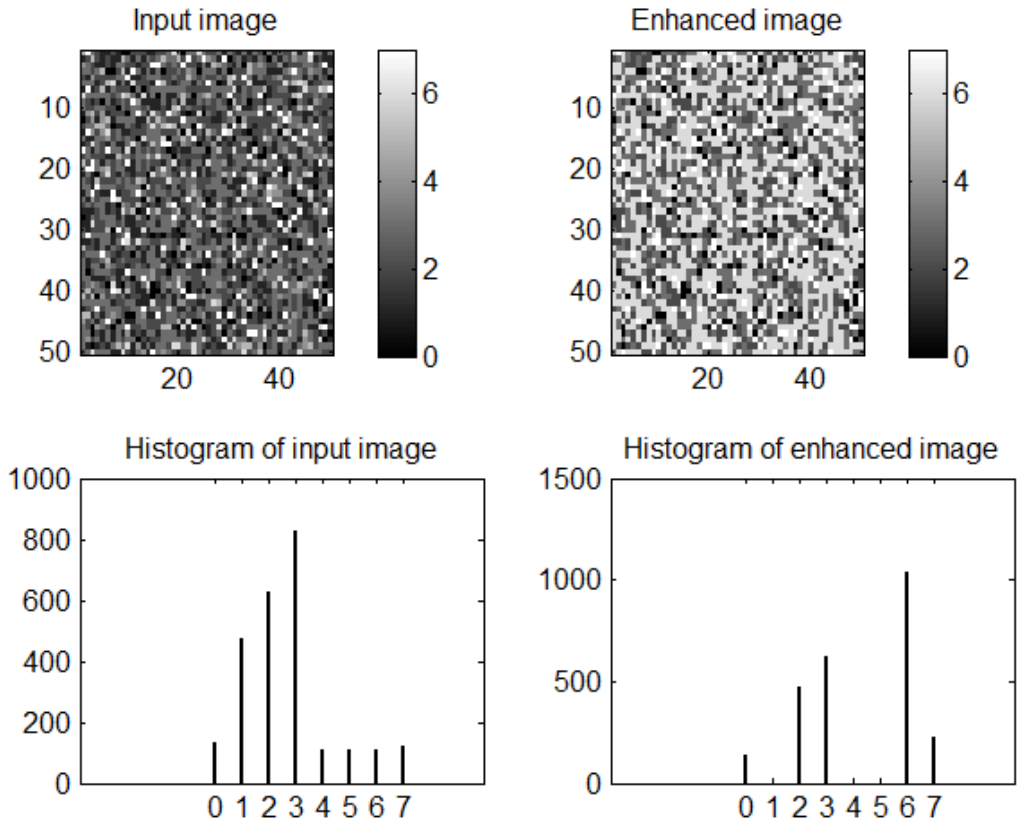
Now you can fill in the intensity values in the output image J by the corresponding S_k values using a nested 'for loop':

```
J = uint8(zeros(size(I)));
for row = 1 :size(I,1)
```

```
    for col = 1:size(I,2)
        J(row,col) = Sk(I(row,col)+1);
    end
end
```

Once you have obtained the histogram equalised output image, you can compute and plot its histogram as you did for the input image in section 2.1.

→ new MATLAB functions used in this section: *find*, *cumsum*, *round*



4.2 Task

Write a MATLAB script for histogram equalization which loads the input image and, if a colour image, converts it to grey-scale. Then returns a transformed image whose histogram is close to the uniform distribution.

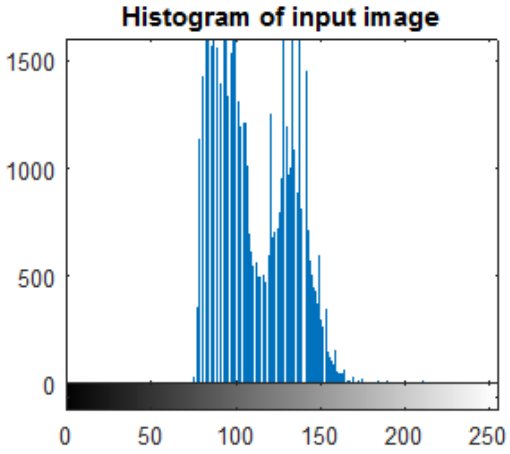
The main MATLAB function for dealing with histogram equalization is *histeq*. Now that you have learnt how to perform the histogram equalization for an image in section 4.1, you can start using this function as:

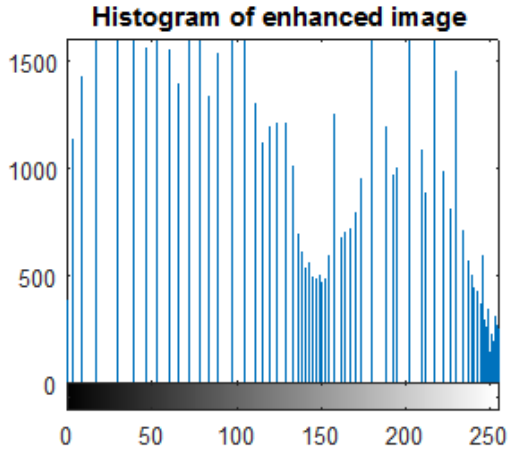
```
J = histeq(I,n)
```

The number of bins used in the histogram is set to be $n = 256$.

Use subplot to display the results in one output figure which should include the original image, a plot of its histogram, the enhanced image, and a plot of its histogram. Use this information to explain why the resulting image was enhanced as it was.

→ new MATLAB function used in this section: *histeq*





Additional Challenges

The focus of this task is to experiment with intensity transformations to enhance an image. Write a MATLAB script for histogram equalisation which loads the input image and, if a colour image, converts it to grey-scale. Then using the equations provided in the slides 22-23 of the lecture notes from week-3, perform:

a) The log transformation
b) The power-law transformation

In (a) the only free parameter is c, but in (b) there are two parameters, c and r for which values have to be selected. As in most enhancement tasks, experimentation is a must. The objective of this task is to obtain the best visual enhancement possible with the methods in (a) and (b). Once (according to your judgment) you have the best visual result for each transformation, explain the reasons for the major differences between them.