University of Alcala

Computer Vision: Lab 2

By: Peeter Tarvas, Gjergj Kukaj

Alcalá de Henares, Spain

1. Experiment more filters with different masks defined in fspecial function and draw conclusions about their operation.

Fspecial function creates a two dimensional filter for a specific type that you can use to filter an image.

Gaussian: returns a rotationally symmetric Gaussian lowpass filter that puts the coloring of the picture in a more discrete format by adding noise to the figure.

Prewitt: returns a 3x3 filter that calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point, and therefore how likely it is that part of the image represents an *edge*, as well as how that edge is likely to be oriented.

Laplcaian: returns a 3-by-3 filter approximating the shape of the two-dimensional Laplacian operator, basically distorts the image so that it's mostly gray edges and contures and otherwise everything is black.

Log: Laplacian of Gauss filter, basically creates a rotationally symmetric Gaussian lowerpass filter and then filters that with the Laplacian 3-by-3 distorting the image so that the contures will go a bit fuzzy.

Average: Returns a filter x-by-x size that averages the picture so to say. Default size is 3x3

Unsharp: 3-by-3 contrast enchantment that fazes the picture a bit by disturbing the contrast of it. It kind of adds Gaussian noise to the picture.

2. Perform the following operations related to image enhancement:

2.1. Load and display image 'pout.tif'

```
I = imread( 'pout.tif' );
subplot(1, 1, 1)
imshow(I)
```

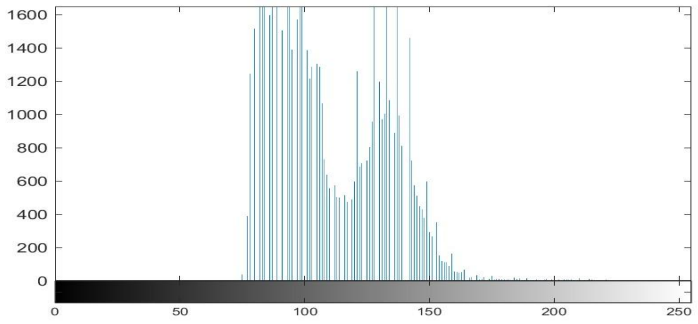
2.2. Check the image in memory (within the Matlab's work space)

Image is a matrix in memory

2.3. Obtain its histogram

An **image histogram** is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance. In Matlab we use imhist() to see the histogram of the image

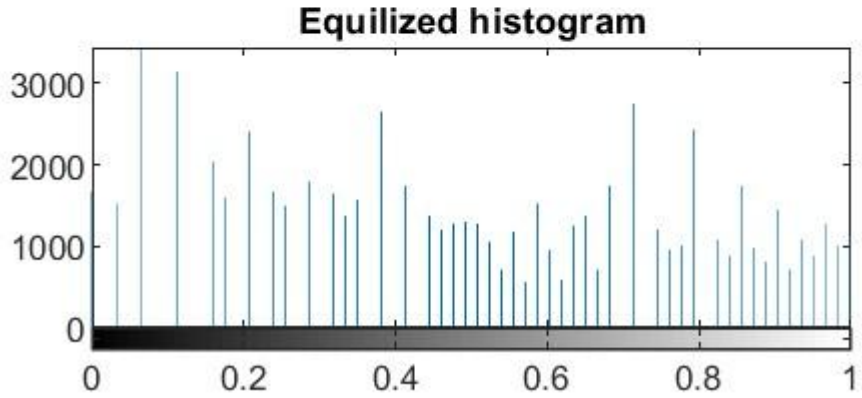
Histogram of pout.tif



2.4. Equalize its histogram and display the equalized image

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values. Through this adjustment, the intensities can be better distributed on the histogram utilizing the full range of intensities evenly. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the highly populated intensity values which are used to degrade image contrast. In MatLab we use histeq()

Equalized image of pout.tif



2.5. Write the image to a jpeg file

To write image to jpeg file we need to use **imwrite()** command.

```
imwrite(I, "pout.jpeg");
imwrite(histeq(I), "equ_pout.jpeg");
```

2.6. Retrieve the original image

To retrive the original image we use **imread()** command and then grayscale the image to type double with **im2double()**

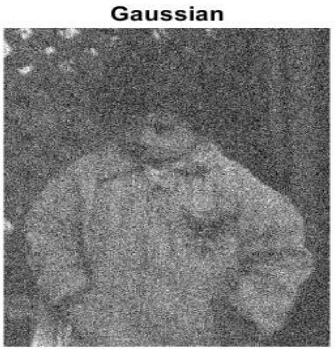
```
I = imread("equ_pout.jpeg");
I=im2double(I);
```

2.7. Maximize its contrast

Contrast of an image is the difference of luminance in a picture. To increase it we have to use imadjust() which adjust the contrast of the image so that 1% of the data is saturated at low and high intensities. In matlab we use
 imadjust(image, [min(min(I)) max(max(I))], [0 1])
Here is a maximised pout.tif image

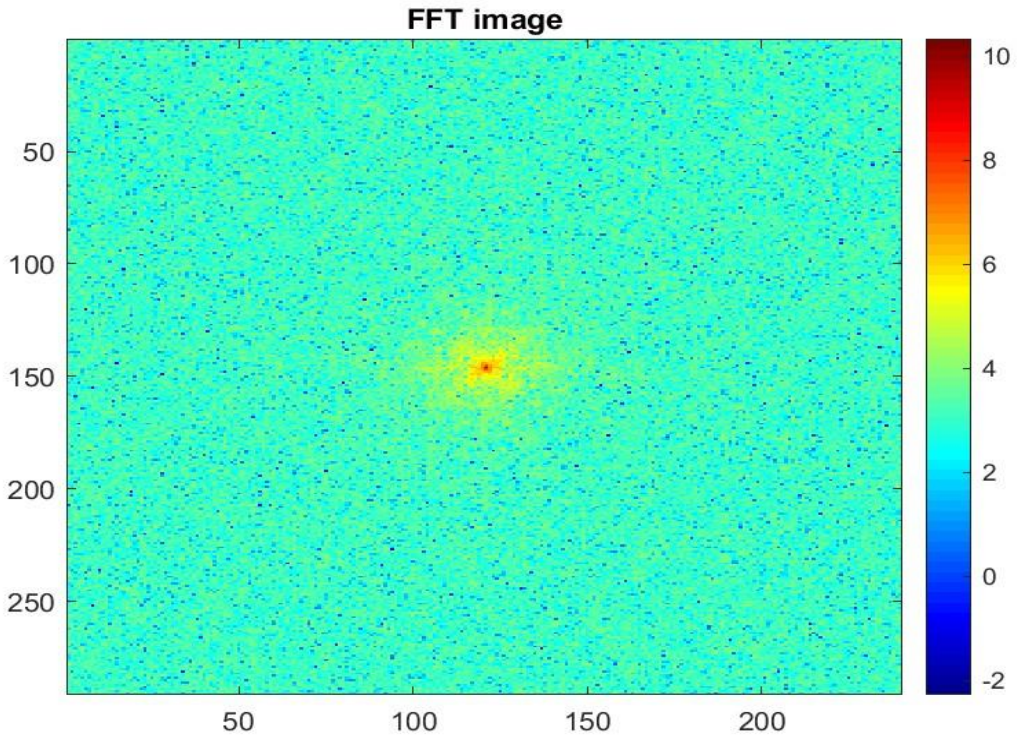
2.8. Add some Gaussian noise

Gaussian noise, named after Carl Friedrich Gauss, is a term from signal processing theory denoting a kind of signal noise that has a probability density function (pdf) equal to that of the normal distribution (which is also known as the Gaussian distribution). In other words, the values that the noise can take are Gaussian-distributed. In Matlab we use imnoise(image, “gaussian”) to filter image with Gaussian noise.
Here is an image of pout.tif that has gaussian noise added to it



2.9. Obtain the FFT of the noisy image

A **fast Fourier transform** (**FFT**) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence.



2.10. Filter the image optimally (using the frequency transformation method. Whereas the optimal filter in the spatial domain corresponds to Wiener filtering)

Frequency transformation is used in vison system design and analysis to compute the transfer function of a high pass, bandpass, or bandstop filter directly from the transfer function for a low pass filter.

Frequency transformation filter

Homework:

1.- Read image therese.gif and indicate its format (RGB or indexed color map).

Using the imread() function to store the image inside I and the colormap inside cmap.

2.- Display the previous image.

Showing the picture by using imshow().

3.- Convert the previous image to a grayscale image (containing only luminance information) and store it in an array called i. Indicate the range of values in which the luminance (blackwhite) of the resultant image is quantified.

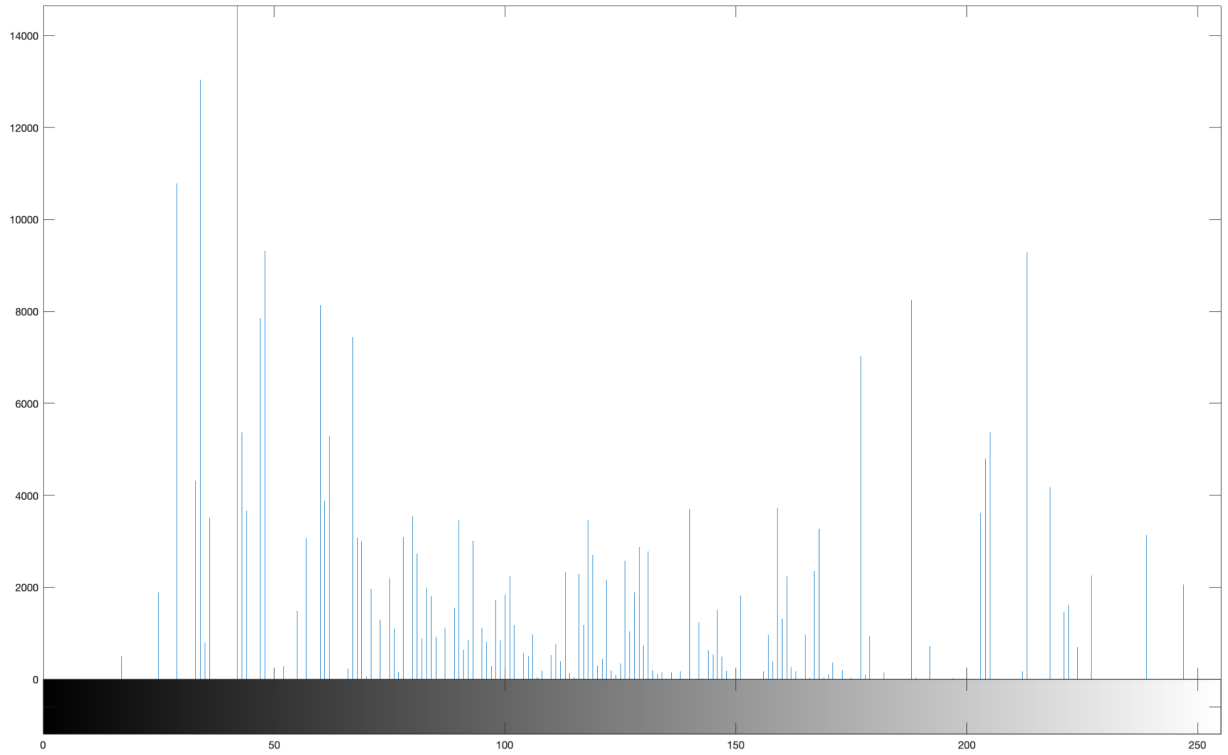
ind2gray() function is used to convert the previous image into a grayscale image that only contains luminance information.
To get the range of values the min() function is used on the image to get the lowest value of the picture and the max() function to get the highest. ‘all’ flag in min/max is used to get the lowest/highest value inside the whole matrix.
This works as the grayscale image only contains luminance info. The output shows that the image contains pure black (0) and pure white (255).

4.- Find out the dimensions of our image i (rows x columns).

To obtain the dimensions of the picture a simple size() function is used.
It returns the rows and columns as a row vector.
The size of the picture is 422x640.

5.- Display the histogram of the gray scale image.

Using imhist() to get the histogram of the image.



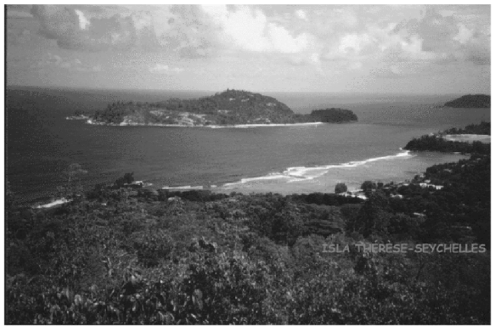
6.- Modify the brightness (adding a constant) and contrast (multiplied by a constant) of the grayscale image and display both results.

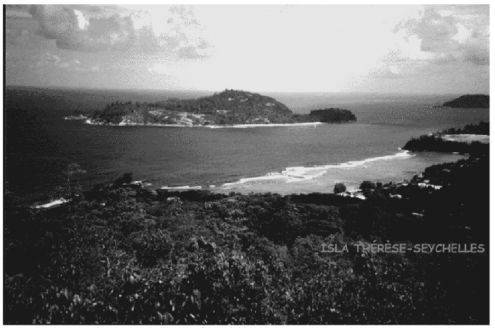
(1,1) Shows original picture.

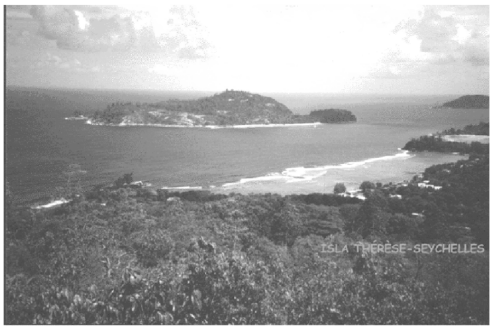
(1,2) Shows picture after using imadjust(i, [0.2 0.8], [0.1 0.9]).
All values between 20% and 80% luminance get mapped to 10% and 90% respectively. Some areas, especially on the right side of the picture, get too dark.

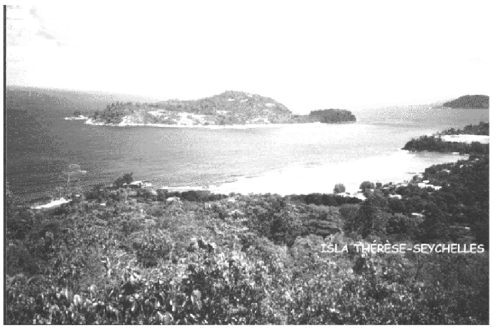
(2,1) Shows the image after adding a constant.
Increases the overall brightness of the image, it seems overexposed. This is only logical as every pixel is increased by the constant. It doesn't serve good to get higher contrast.

(2,2) Shows the image after multiplying by a constant.
The contrast is higher than by adding a constant. Dark pixels stay relatively dark as multiplying a low value by a constant will stay low. The brighter pixels on the other get easily exposed as multiplying them increases their value way more. This can clearly be seen on the water surface and on the trees. Some of the pixels that were brighter now are too bright. This serves better to get a higher contrast, but it changes bright pixels too much.









7.- Perform a binarization of image i with a threshold whose value is approximately half of its range of gray scale and display the result.

For this exercise two different thresholds were used to compare the results.

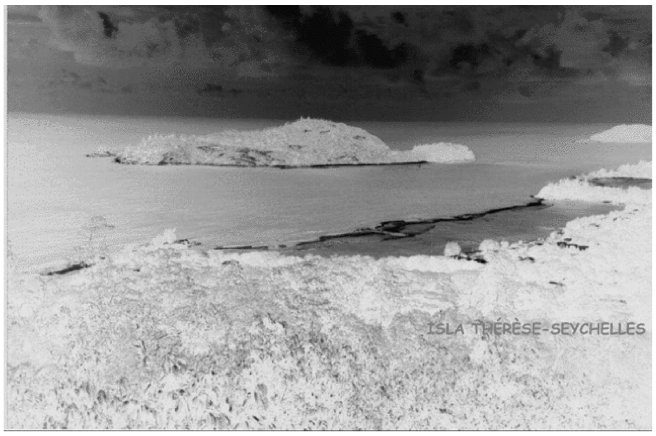
Using a simple filter by mapping all pixel <= 127 to 0 and all pixels > 127 to 255.

Using the average value as threshold so that pixel <= 106 to 0 and pixel > 106 to 255.

It's hard to tell which of these "filters" give better results. In general the mean value approach should give way better results as there may be pictures that are pretty dark or overexposed. With the mean value the threshold should be more fitting. But as the mean value of the grayscale picture is 106 it is already pretty close to the simple logic used in the first binary filter. In this case, as the mean value is below 127, many more pixels got assigned to pure white (255) as in the first image.

8.- Get the negative of the grayscale image obtained in point

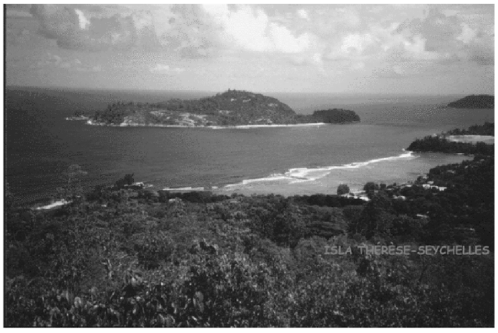
The negative grayscale is simply obtained by subtracting 255 minus the image's pixels.
i_negative = 255 - i;

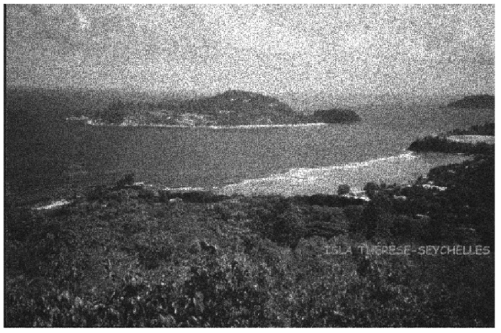


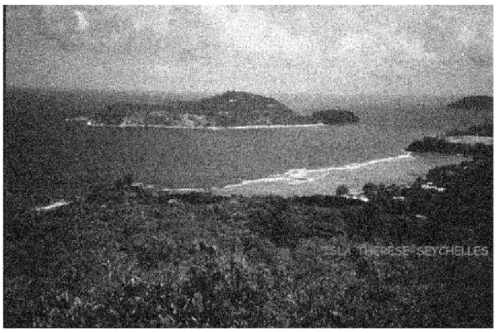
9.- Obtain three noisy versions of the image i: (a) with uniform distribution noise ‘speckle’, (b) with ‘salt and pepper’ noise with 5% noise density, (c) with Gaussian noise with zero mean and 0.01 variance.

For this exercise the imnoise() function will be used as it provides the different options for all the three filters. The result looks as follows:

(1,1) Shows original picture.
(1,2) Shows the speckle image.
(2,1) Shows the salt and pepper image.
(2,2) Shows the gaussian image.







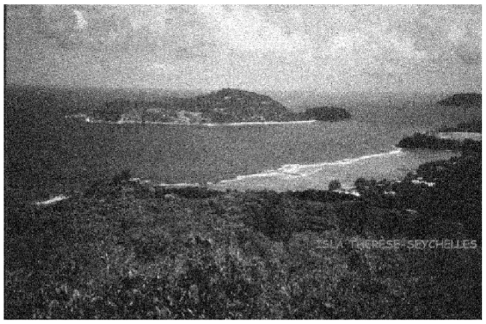
10.- Based on the above image contaminated with Gaussian noise, filter it using the neighborhood average, first with a 3x3 neighborhood and then with a 5x5, and display the results.

(1,1) Image contaminated by gaussian noise.

(2,1) Image after 3x3 neighborhood operation.
This image contains less noise and details are still visible.

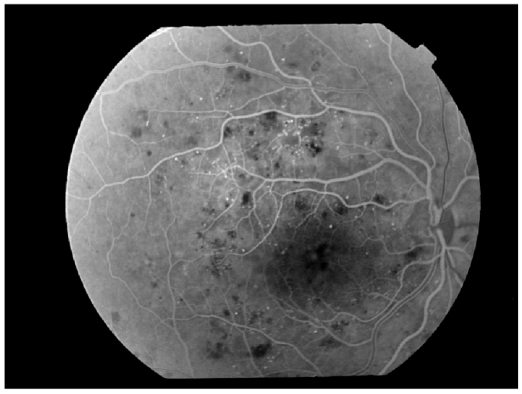
(2,2) Image after 5x5 neighborhood operation.
This image contains way less noise, but on the other side it is more blurry and details are less visible.

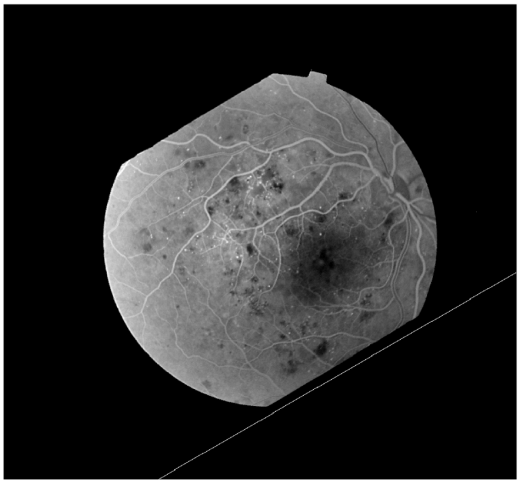
Concluding it is to say that the size of the average filter should be based on the objective of the filtering. If details should still be visible a smaller filter should be used. On the other side, if there is a lot of noise, a small-sized filter will not be able to remove the noise. Especially in line-detection it could be more important to get a less noisier image even though it would be a bit blurrier.



11.- Read image venas.tif and rotate it a 30-degree angle.

imrotate() is used to rotate the image. The result is the following:





12.- From the original image venas.tif, get 16 images in different trials with Gaussian noise of zero mean and variance 0.01. Save them in files with their respective names (noisy1.bmp, …, noisy16.bmp).

For this exercise a for loop is used. Inside the for loop the iteration variable is converted to a char and with the strcat() function a string is concatenated to save it in the right folder.

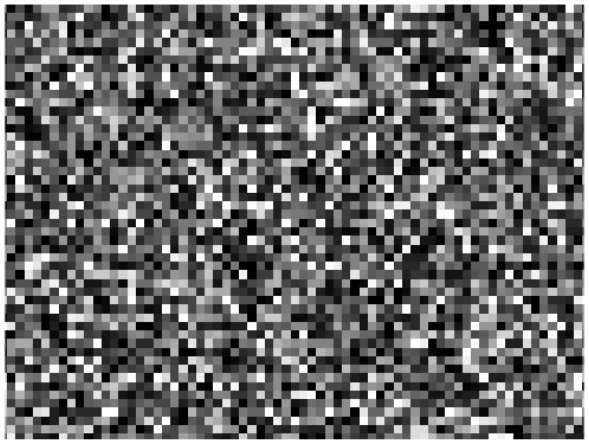
```
113     file_name = "output_pictures/noisy";
114     file_type = ".bmp";
115     for j = 1:16
116         temp_file = imnoise(I,'gaussian', 0, 0.01);
117         number = int2str(j);
118         temp_name = strcat(file_name, number, file_type);
119         imwrite(temp_file, temp_name);
120     end
```

13.- Perform some tests on noise filtering using the technique of image averaging on the images created in the previous point. To do this, try it first with 4 noisy images, then 8, and finally with 16. Compare results visually.

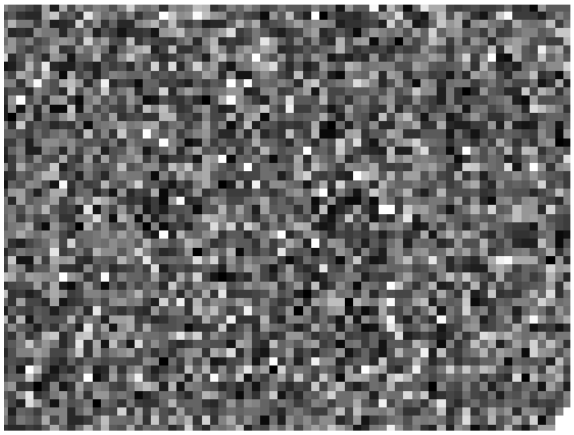
The approach for this exercise is similar to the previous exercise. A for-loop and the string concatenation is used to get the needed files.
It's important to note that before each for-loop an array of the same size as the image was initialized with zeros. The type of the array must be at least uint16 as the adding the pictures with each other will result in an array that has values higher than 255.
The noisy pictures must also be converted to the same data type because integers can only be combined with integers of the same class, or scalar doubles.

To better show the result and the difference between the three different images the images are modified:

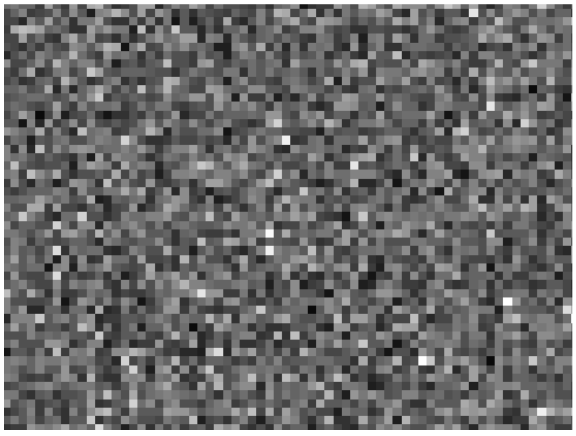
- 8x Zoom into the left upper corner
- Intensifying the noise by multiplying the image times 10.



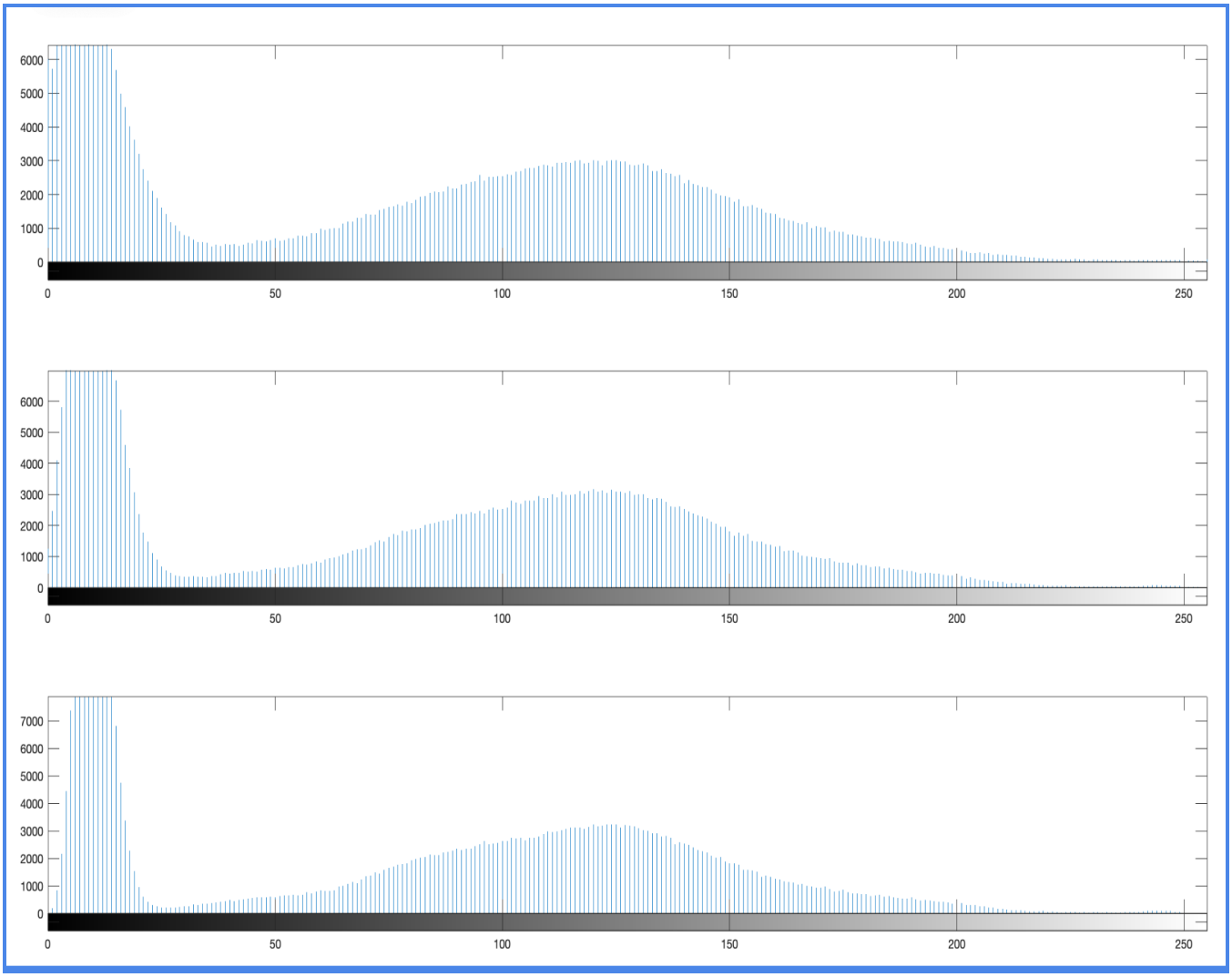
Using 4 images to average the picture the noise is still intense. The distribution of different pixel intensities (brightness) is higher.



Using 8 images to average the noise is less intense. The distribution of different pixel intensity is less.

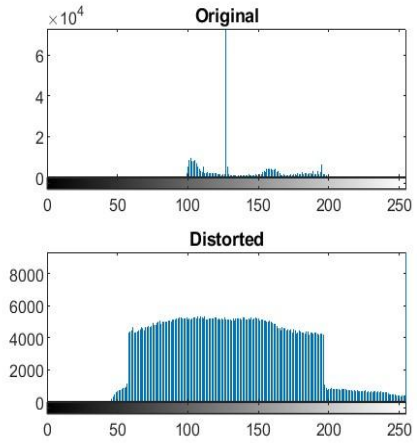


Using 16 images to average the noise is the least intense. The distribution of pixel intensity is the least of all images. The noise is more uniform.



The histogram proves the impression. On the left side it shows that the dark pixels are less distributed on the last picture. There are more pixels of the same color and therefore less noise. Important note: The histogram was taken of the image without zoom and x10 filter.

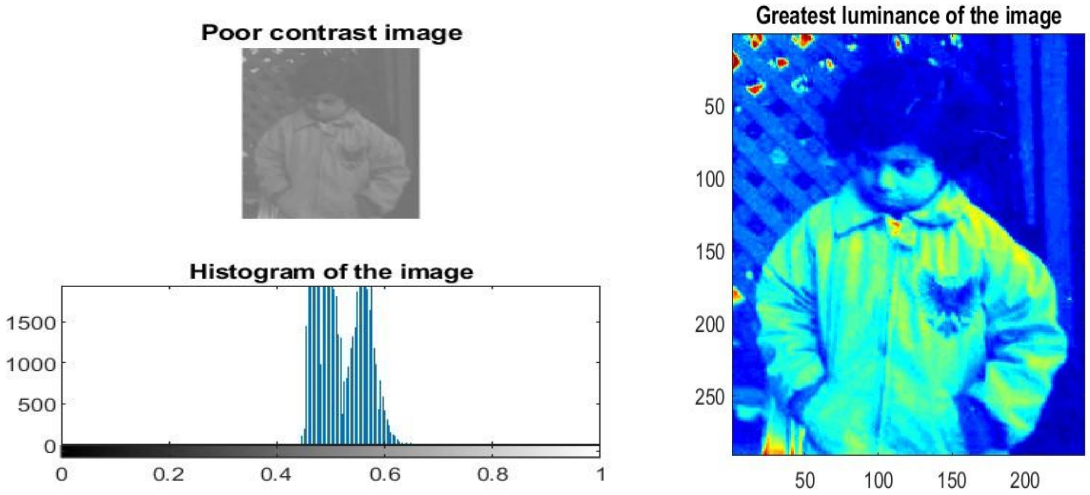
14.- Create an image with Paint, composed entirely of an intermediate greyish background luminance and save it to a file. Complete the following steps in Matlab: - read the image with Matlab, convert it to grayscale and display its histogram - contaminate the grayscale image with 'speckle' noise and store the result in another image file - observe the histogram of the noisy image and confirm that the noise has a uniform distribution - ask for help on the imnoise command to see how we can change the noise variance and contaminate the image with speckle noise, using other variances, and see what happens with its histogram.



Firstly I added some speckle noise with imnoise command, with intensity of 0.1 and then generated the images.

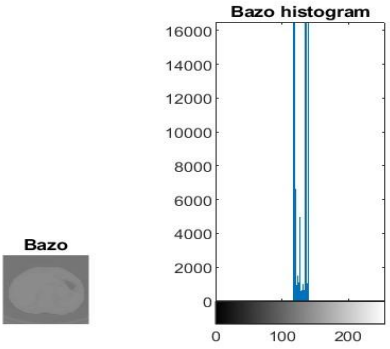
The histogram so to say starts to spread - flatten - apart x on the axis the more distorteded or speckled the image is.

15.- Create a dark, very poorly contrasted image, preferable from a photograph whose gray levels are multiplied by 0.1. Then display that image and its histogram. Finally, obtain the logarithm of the image, calculating previously the optimum value of K that maximizes its output gray level. Display the resulting image and observe the enhancement obtained.



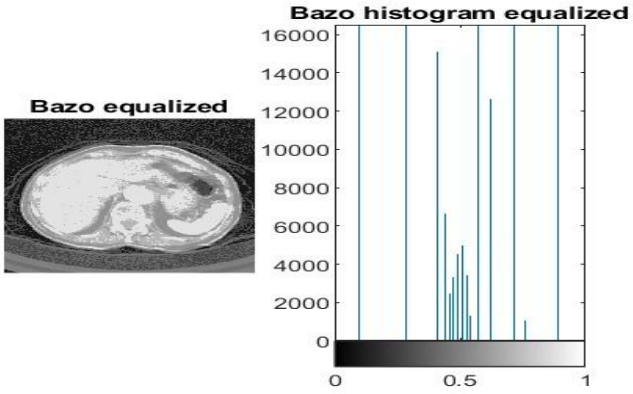
I used pout .tif as the picture. Firstly I made its contrast worse with imadjust command and then displayed its histogram. After that I obtained the logarithm of the image by the theorems in the logarith of and image file. After that I displayed it. It looks like it’s luminance is at maximum.

16.- Read image bazo.bmp and display its histogram. Check that it is an image poorly contrasted.



Bazo is a poorly contrasted image as can be seen form the histogram, because its gray levels are very unequally distributed.

17.- Convert the image above to grayscale and then equalize its histogram. Display the resulting image and its histogram, contrasting the differences with the image of the previous point



Since the histogram has been equalized, not perfectly but still better, the contrast of the image is better as well. This is due to there being more gray colors in the picture than before on more equal density.