Assignment #1 - Basic image operations

Faculty of Engineering Science
Dept. of Electrical and Computer Engineering
Introduction to Digital Image Processing, 361-1-4751

Maor Assayag 318550746

Refahel Shetrit 204654891

1 Histogram Manipulation

In this section you will play with image histograms and quantization's.

1.1 Reading the Image

Read the image named picasso.jpg and transform it into a gray scale image of type double using double(*rgb2gray()*) function. Normalize the image between [0; 1].

Figure 1.1

Explanation: We coded a simple function for normalizing a grey-scale image to the range of [0, 1]. The function "im_normlized" simply finds the range in values in the original grey scale and map the image to the new range.

1.2 Histogram Construction

1. Write your own function named dip_histogram(img, nbins) that will return the histogram of the image 'img' using 'nbins' bins.
2. Display the generated histogram using 256 bins.

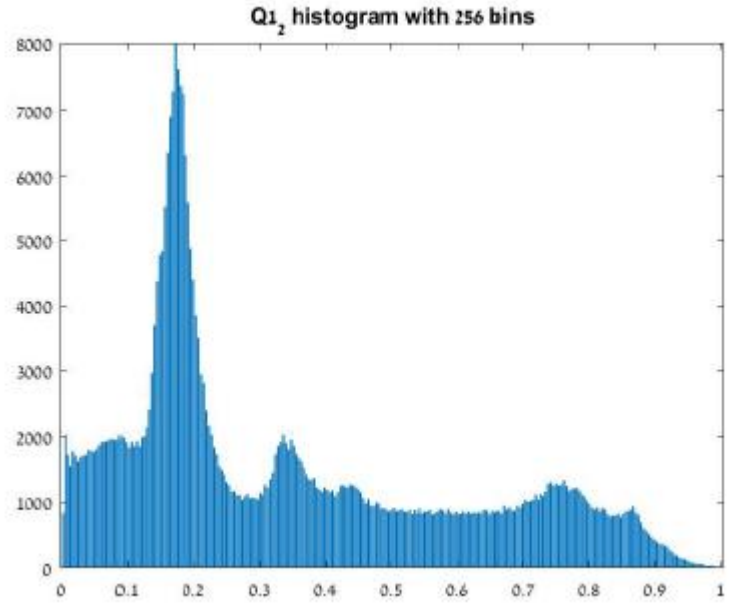

Figure 1.2.1

Explanation: We coded a function to returns a histogram of a normalized grey-scale image. The function gets also in parameter the desired number of bins, then count how much values fit each bin value-range. The function returns a counting array. Then we can simply display the array with the function 'bar'.

3. We can say that the histogram roughly represents a mixture of Gaussians (Normal Distribution). To which image region (hair, face, background etc.) does each Gaussian in the histogram approximately correspond?

Explanation: We can see that there are mainly 3 spikes in accumulate values at ~0.17 (darker color), 0.35 (a little brighter), 0.43 and 0.76. We can recognize in the picture the dominants 4 colors and attached them to the characteristic areas – the Beard, the background, the clothes and then the skin (from the darker color to the brighter) etc.'.

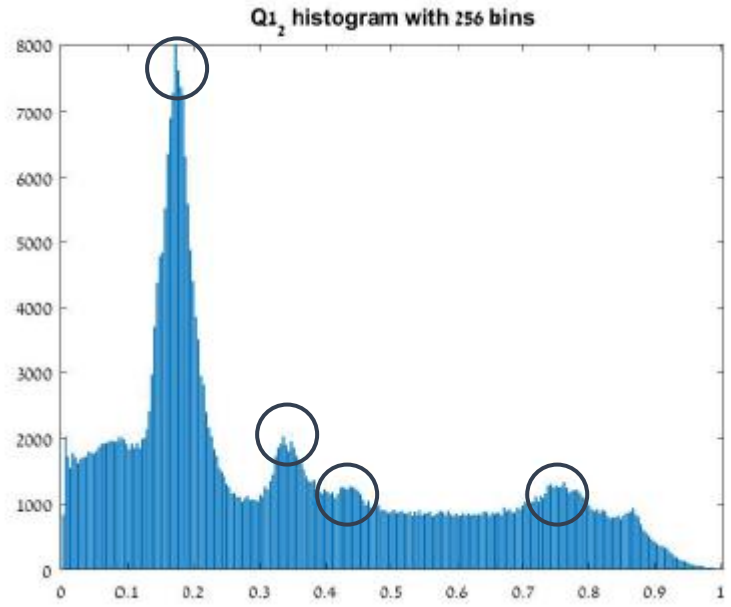

Figure 1.2.2

4. Display the histograms using 128,32,4 bins. Note: Here, you can use the *imhist()* function only for checking your answer.

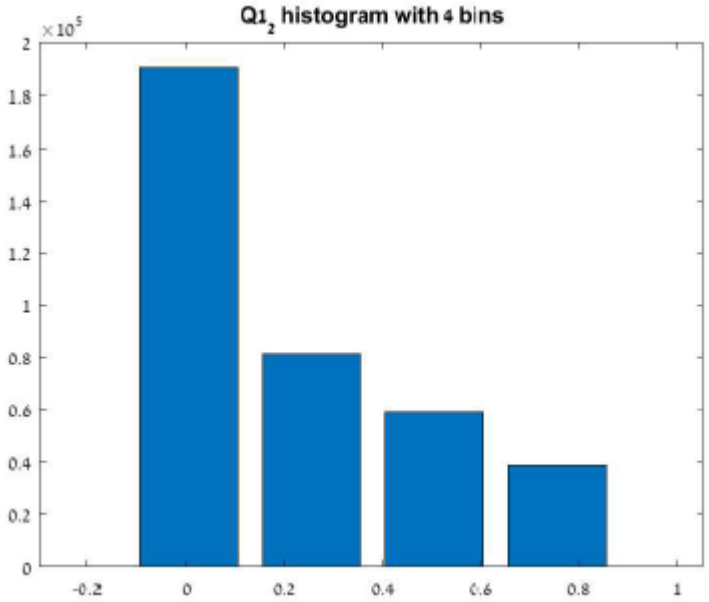

Figure 1.2.3

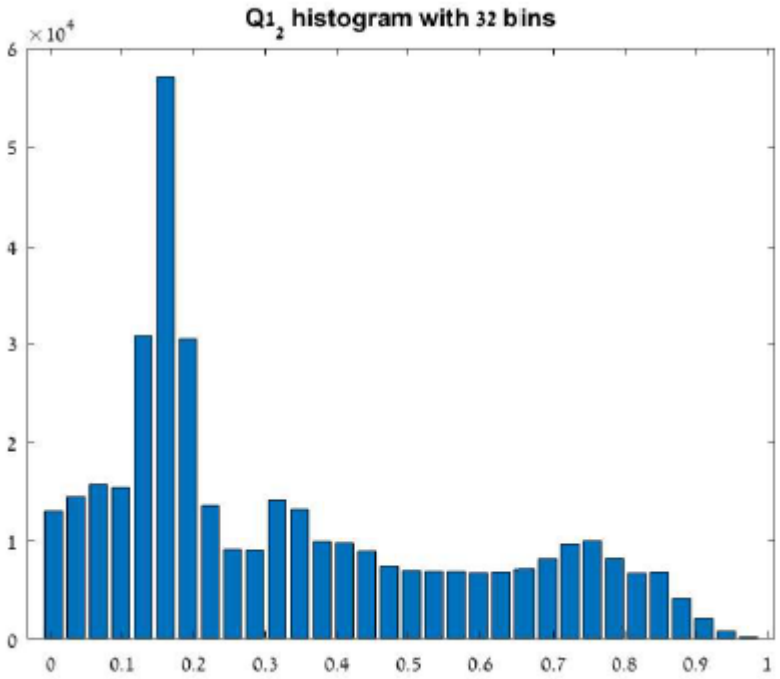

Figure 1.2.4

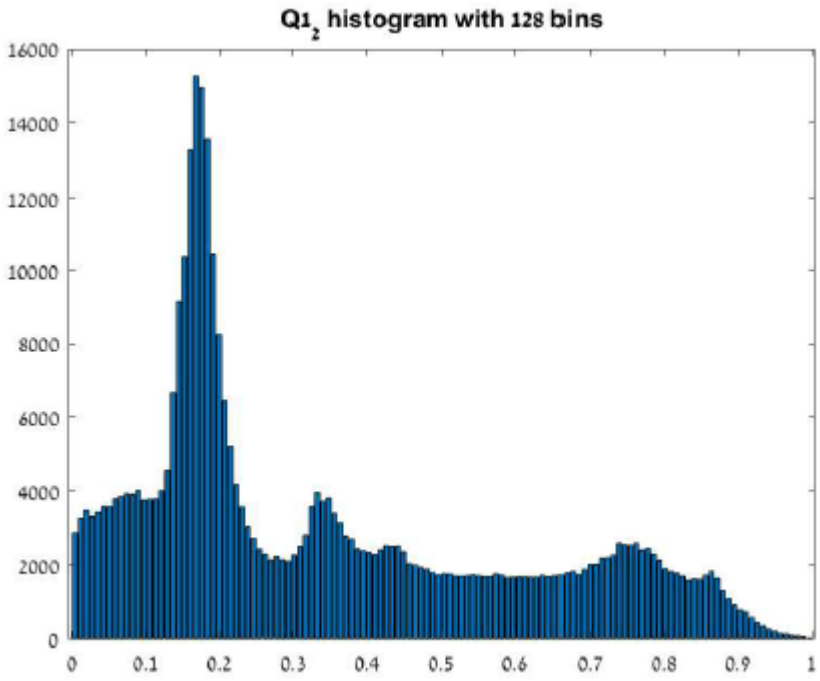

Figure 1.2.5

Explanation: We can see that each time that we increase the number of bins less and less pixels are corresponding to a specific bin, hence the y-axis is decreasing.

1.3 Brightness

1. Write your own function named *adjust_brightness(img, action, parameter)* in which 'action' could get either 'mul' for multiplication or 'add' for addition. Adjust the brightness of 'img' using the 'parameter'. The output of the function will be the modified image.

2. Display the original gray scale image together with the adjusted images of increased and decreased brightness.

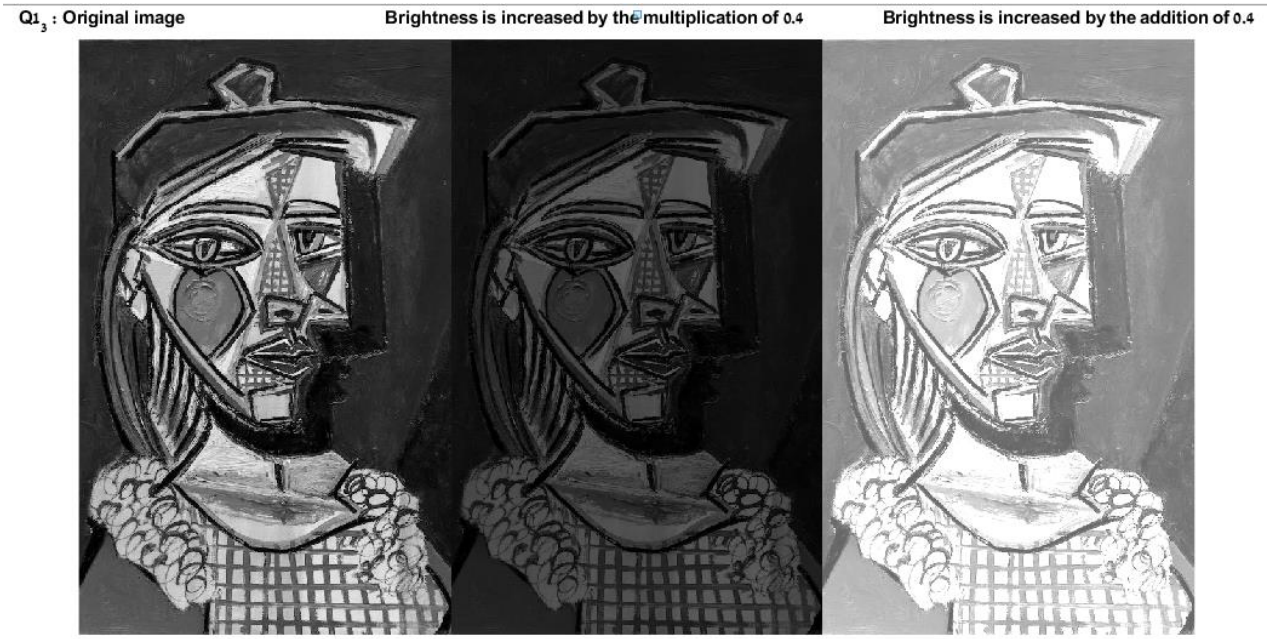

Figure 1.3

Explanation: By adding/multiplying to the values in the image we can change it brightness level as shown above. An important part is to not forget to limit the max value to be 1 and the min value to be 0.

1.4 Contrast

1. Write your own function named *adjust_contrast(img, range_low, range_high)* that will change the contrast of the image 'img' and in which the range_low & range_high parameters will determine the new dynamic range of modified image. The output of the function will be the modified image.

2. Calculate the modified image for a new dynamic range of [0:3; 0:8], [0:45; 0:55] and [1; 0] and display the images and corresponding histograms. Explain the effect of each new range.

Explanation: First, let us check that our function work well by comparing the results to the built-in function *imadjust()* :

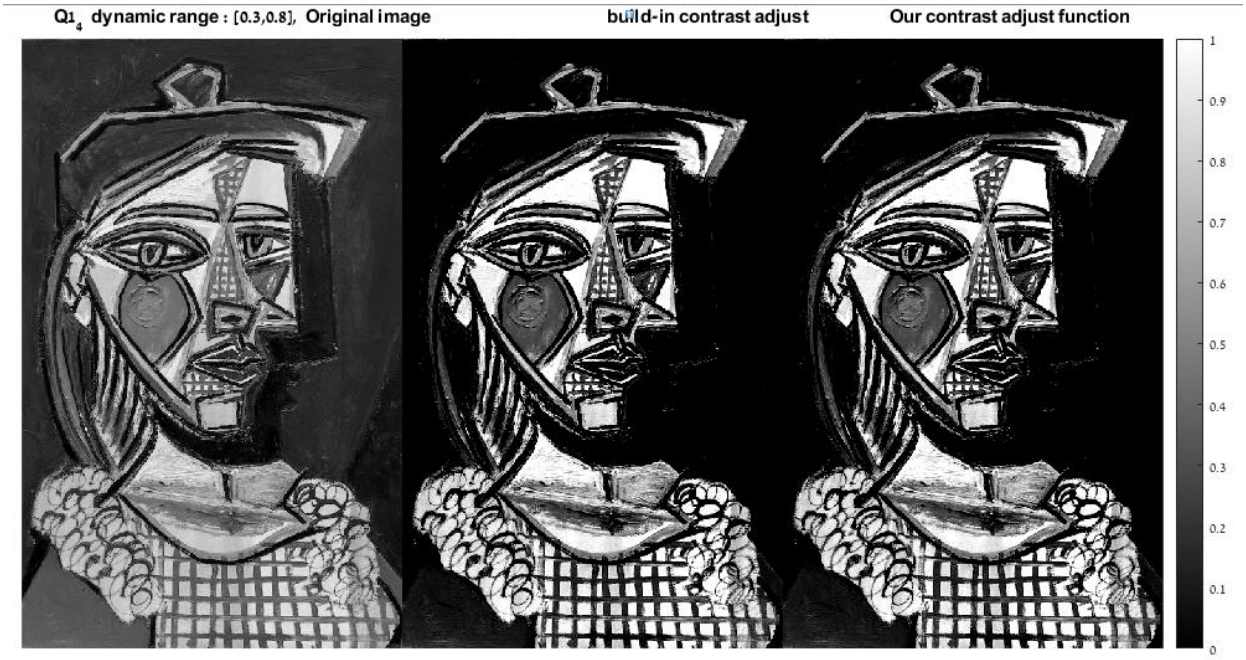

Figure 1.4.1

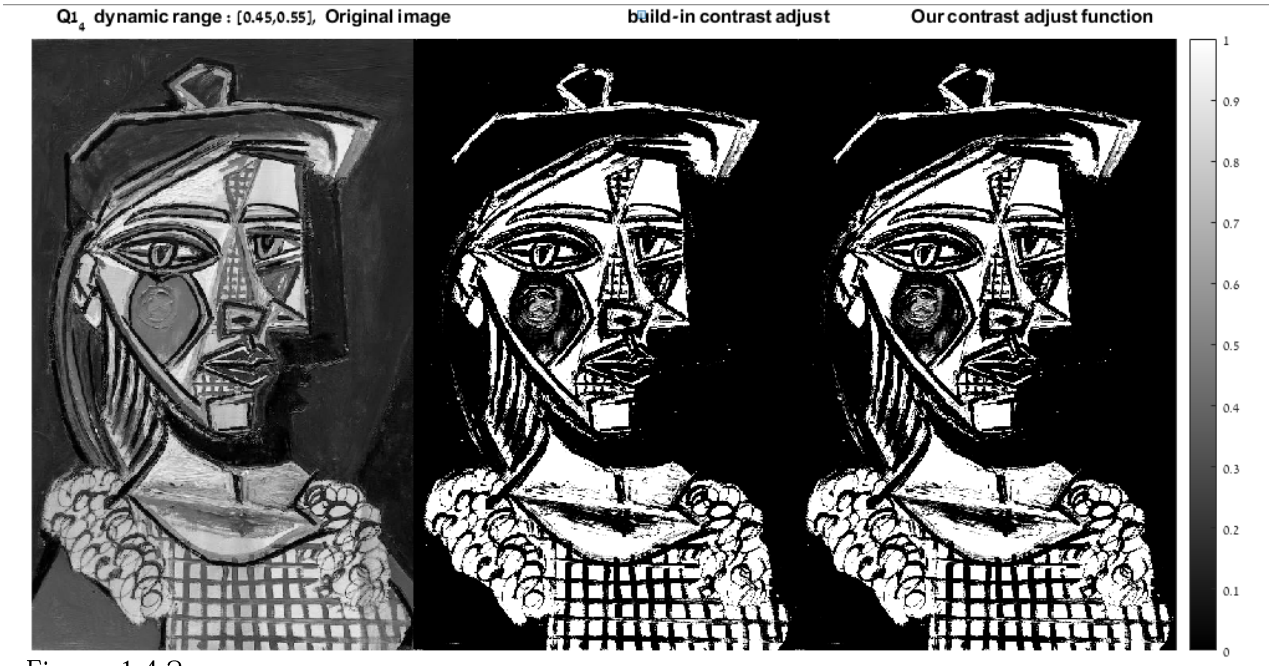

Figure 1.4.2

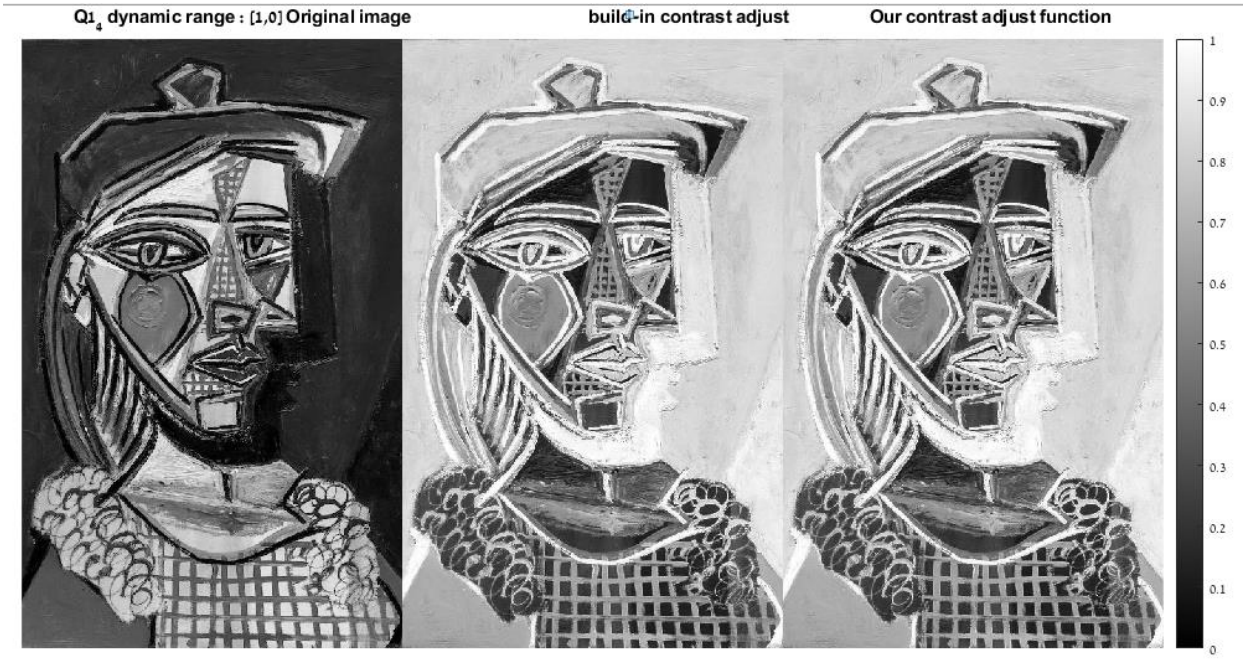

Figure 1.4.3

Now, we can show the images and correspond histograms (we choose 32 bins for convenience visualization):

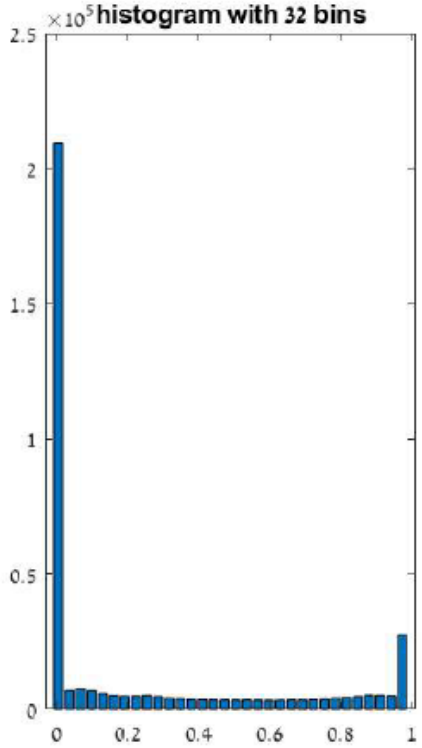

Figure 1.4.4

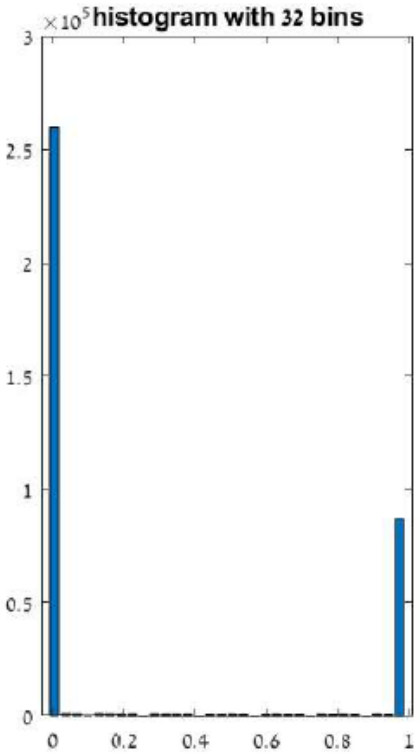

Figure 1.4.5

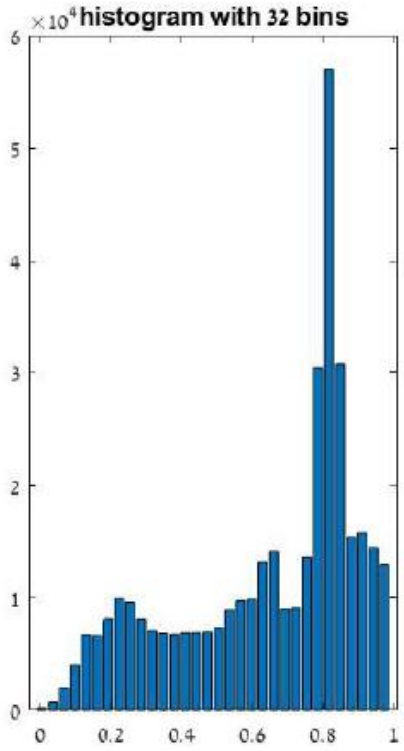

Figure 1.4.6

If the contrast set to high, resulting in the blacks being too black, and the whites being too white. Also, the pixels that are a mid-shade of gray fade into each other. In the last range [1, 0] we can see that the grey-scale image has been inverted color-wise.

1.5 cation

The default MATLAB setting *imhist()* function uses 8bit quantization (256 bins). Use the histograms to display the original gray scale image using 6bit, 4bit, 2bit and 1bit quantizations.

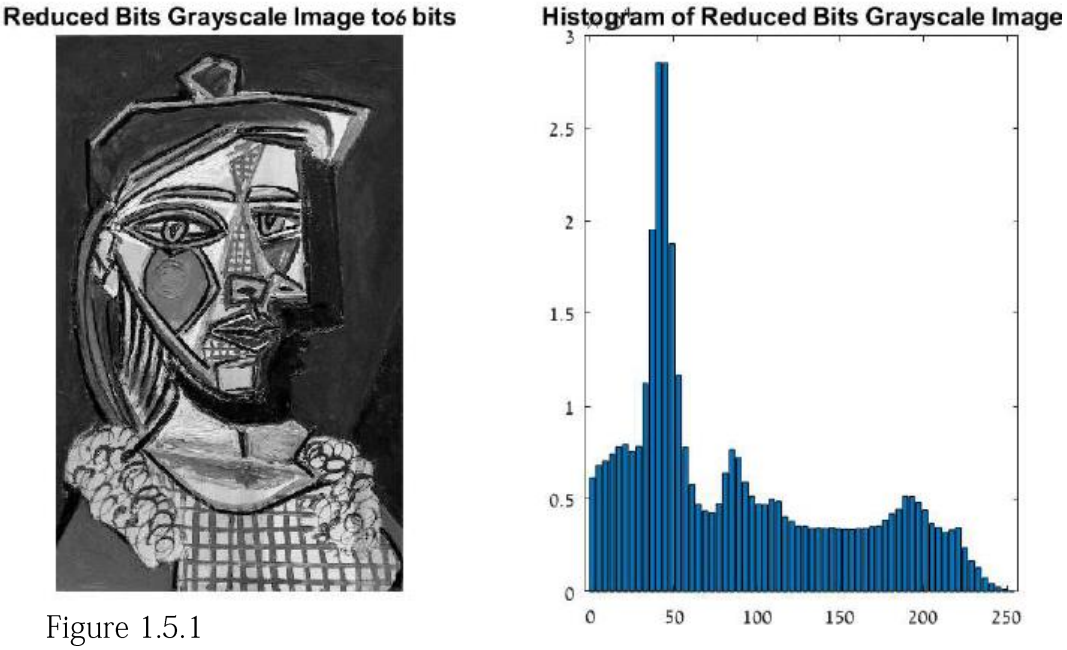

Figure 1.5.1

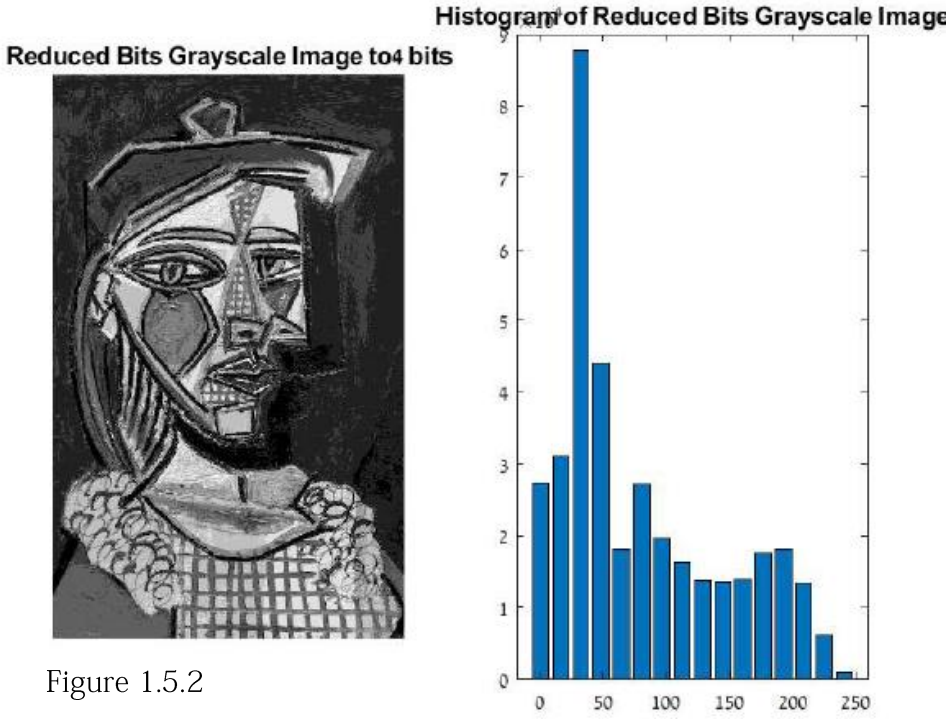

Figure 1.5.2

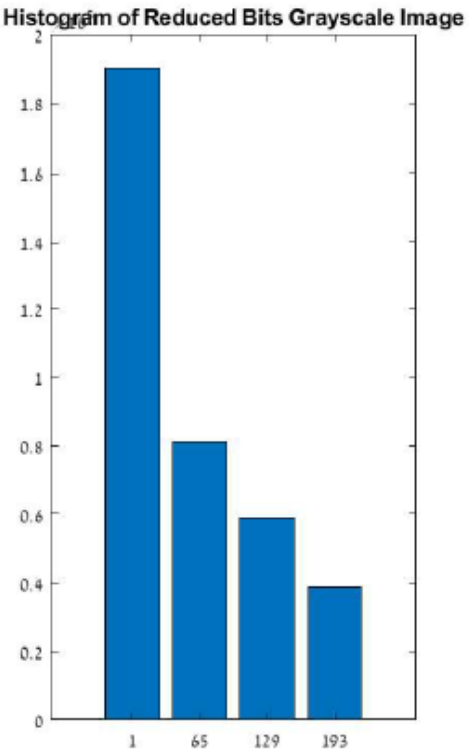

Figure 1.5.3

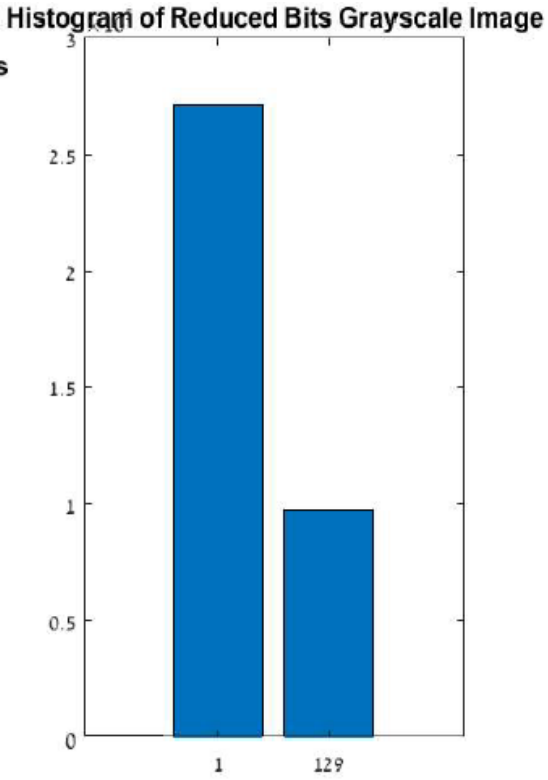

Figure 1.5.4

Explanation: We reduced the number of grey shades by divide the values of the image by 2^{8-n}, when n is the n-bit quantization's that we want. After its compressed to 2^n shades, we been able to decimate some of the original information to less bits.

We can notice that by decreasing the number of bits in which we quantization the grey-

scale level of the image, we get less and less colors, so m bits quantization enables us to use 2^m colors, as shown in the histograms and visually.

1.6 Histogram Matching

1. Take an image using your camera/phone/computer and transform it into a gray scale image of type double using *double(rgb2gray())* function. Normalize the image between [0; 1].
2. Read by yourself about one of the algorithms for histogram matching between two images and write a short summery about the algorithm you read.

   ➔ Histogram matching is a process where a time series, image, or higher dimension scalar data is modified such that its histogram matches that of another reference dataset. The algorithm is as follows. The cumulative histogram is computed for each dataset. For any value (x_i) in the data to be adjusted has a cumulative histogram value given by G(x_i). This in turn is the cumulative distribution value in the reference dataset, namely H(x_j). The input data value x_i is replaced by x_j.

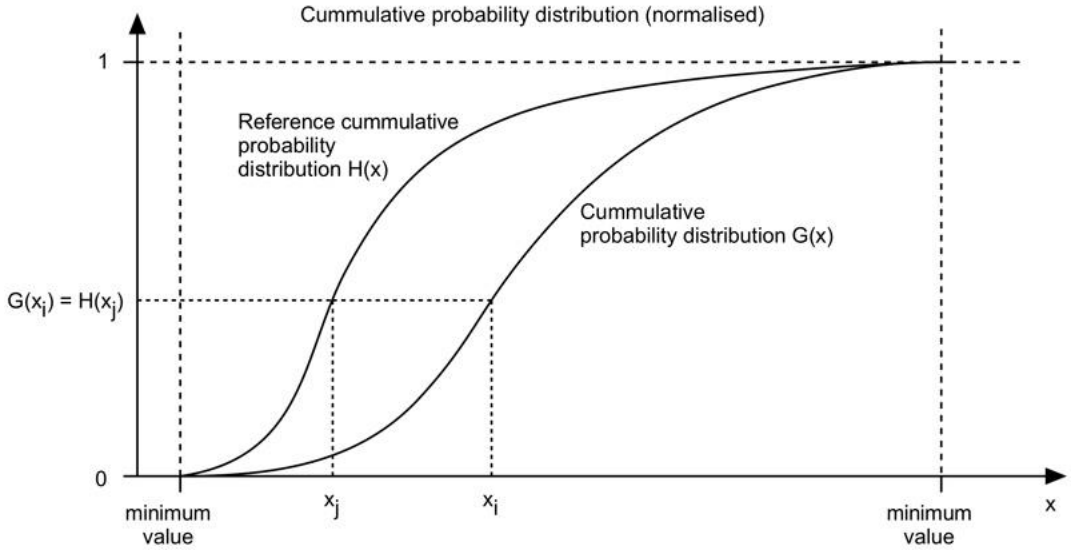

Figure 1.6.1

3. Use the MATLAB function imhistmatch() to match the histogram of your image to the histogram of your previously processed picasso.jpg image.
4. Display all three images and their corresponding histograms. Explain the results.

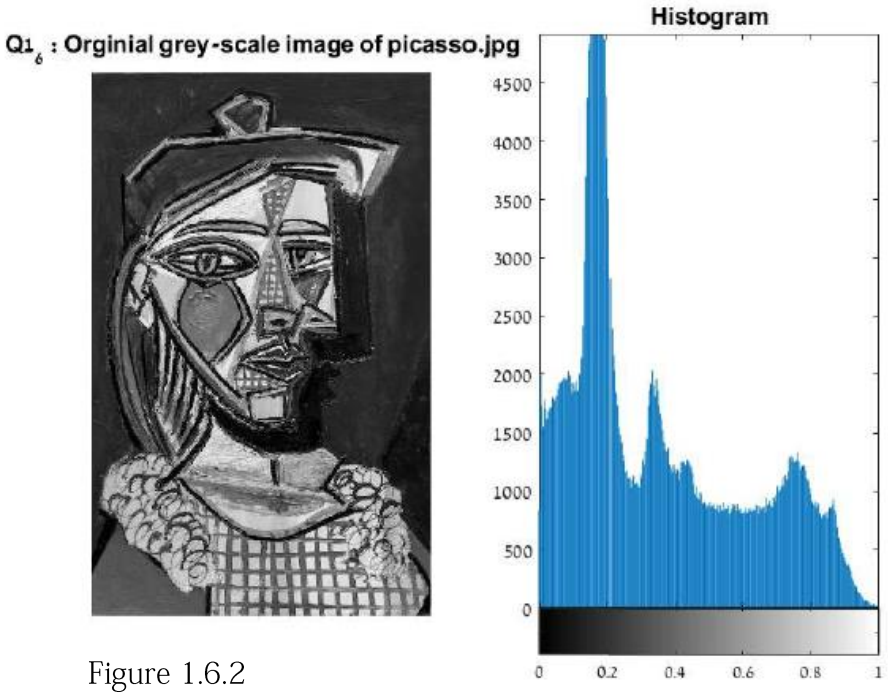

Figure 1.6.2

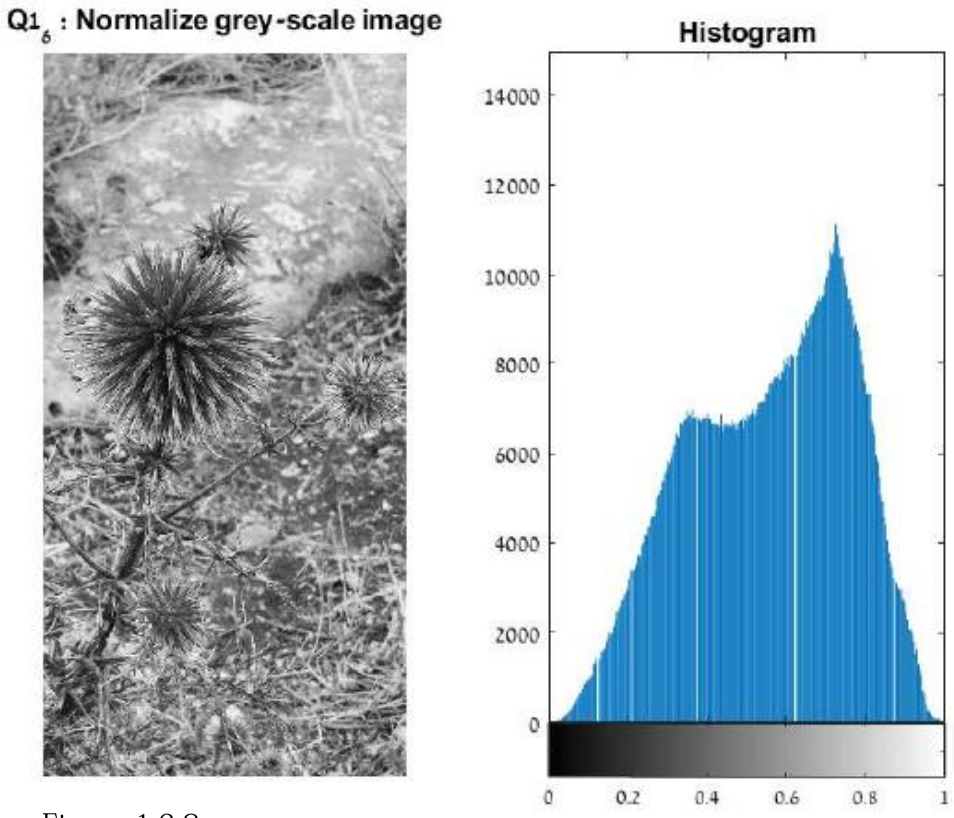

Figure 1.6.3

$Q1_6$: Normalize grey-scale matching image

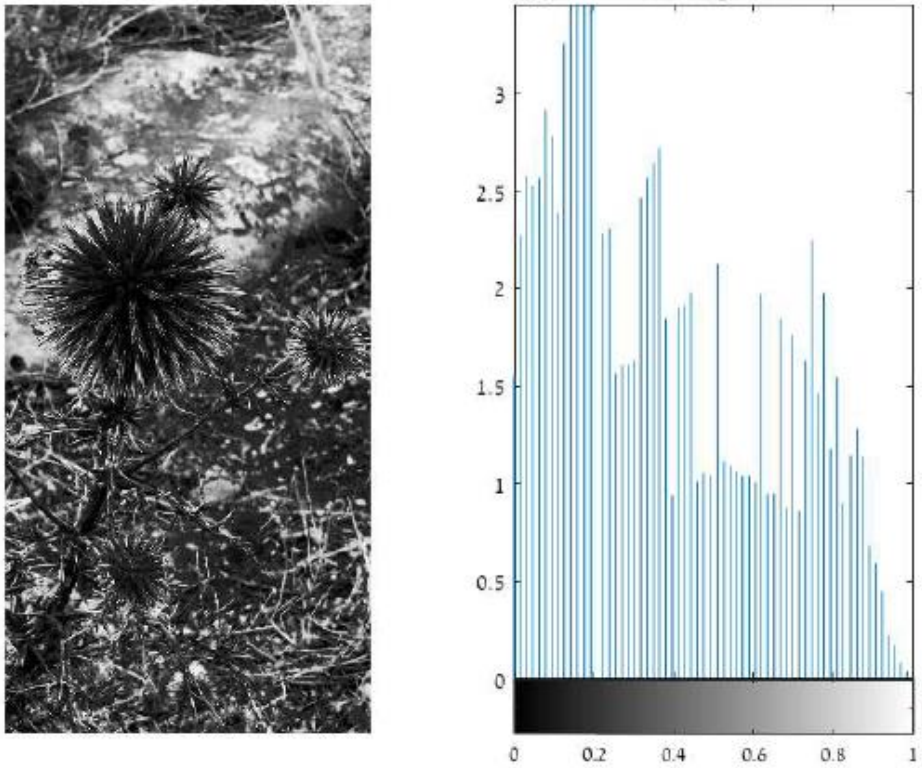

Figure 1.6.4

Explanation: We can notice that the new matching image have a histogram that resemble to the reference histogram dataset. In our case its look like the process increases the contrast and darkened the dark areas in the photo, making it clearer to recognize objects.

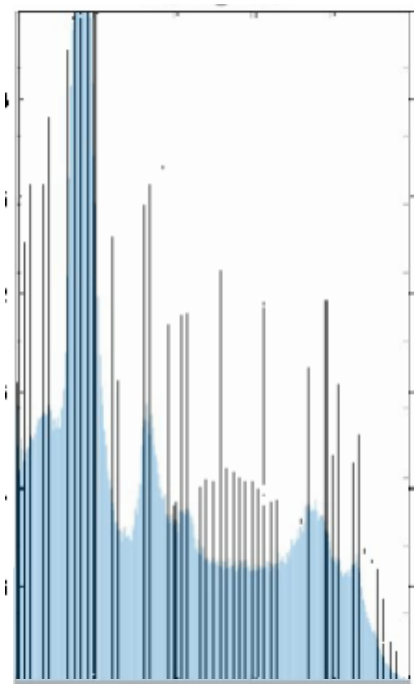

Figure 1.6.4 – overlay the 2 histograms

2 Spatial Filters and Noise

In this section you will examine the effect of different filters on a given image.

2.1 Read the Image

Read the image named dog.jpg.

Figure 2.1

2.2 Mean vs Median Filter

1. Write a function named mean_filter(img) that will apply a mean of size 3x3 on the image 'img'. Make sure that the size of the output image is the same as the input image. Find a method to deal with the boundaries.

Figure 2.2.1.1

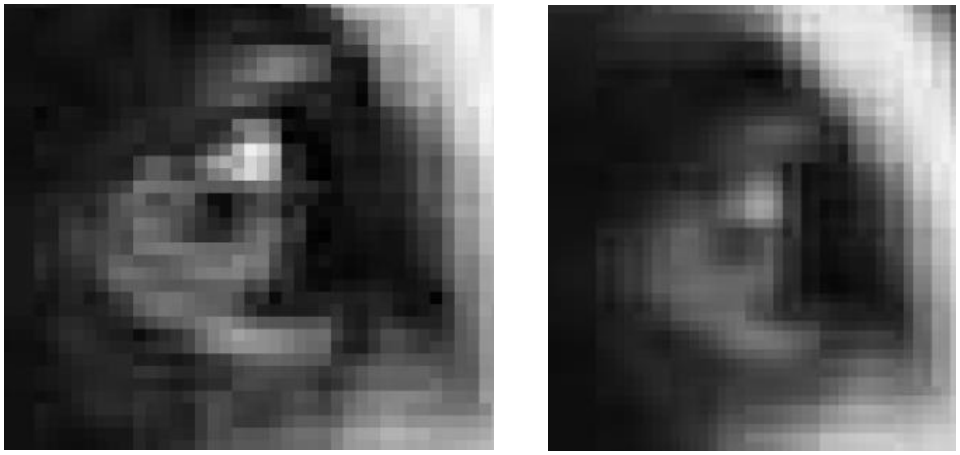

Figure 2.2.1.2 – Zoom in Original vs Mean filtered image

2. Write a function named median_filter(img) that will apply a median filter of size 3x3 on the image 'img'. Make sure that the size of the output image is the same as the input image. Find a method to deal with the boundaries.

Figure 2.2.2.1

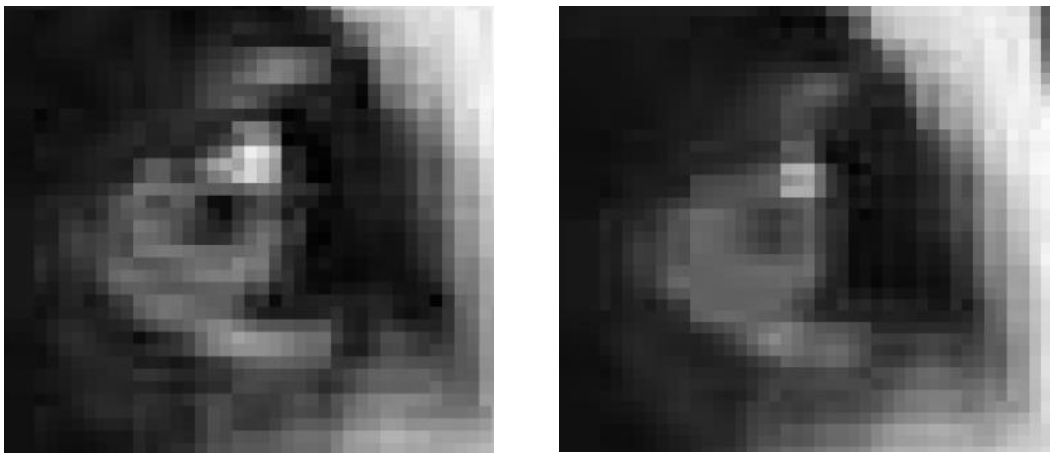

Figure 2.2.2.2 – Zoom in Original vs Median filtered image

Explanation : We deal with the boundaries with **zero-padding** – we enlarge the image by adding rectangular strips of zeros outside the rectangular edge of the image, so that you have a new larger rectangular image with a black frame around it. In this way we can calculate the values of the edges with the filter matrixes.

The **mean** filter is a simple sliding-window spatial filter that replaces the center value in the window with the average (mean) of all the pixel values in the window.

The **median** filter is also a sliding-window spatial filter, but it replaces the center value in the window with the median of all the pixel values in the window.

The median filter is also widely claimed to be 'edge-preserving' since it theoretically preserves step edges without blurring – which we can see that from the results – the

median filter is less blurring at objects edges (e.g. the dog eye), and we can see the eye in more details.

2.3 Gaussian Filter

1. Write a function named dip_gaussian_filter(img) that will apply a Gaussian filter of size 3x3 on the image 'img'. The smoothing kernel should be with covariance matrix of $\begin{matrix} 0.3 & 0 \\ 0 & 0.3 \end{matrix}$
2. Display the filtered images.

Figure 2.3.2.1

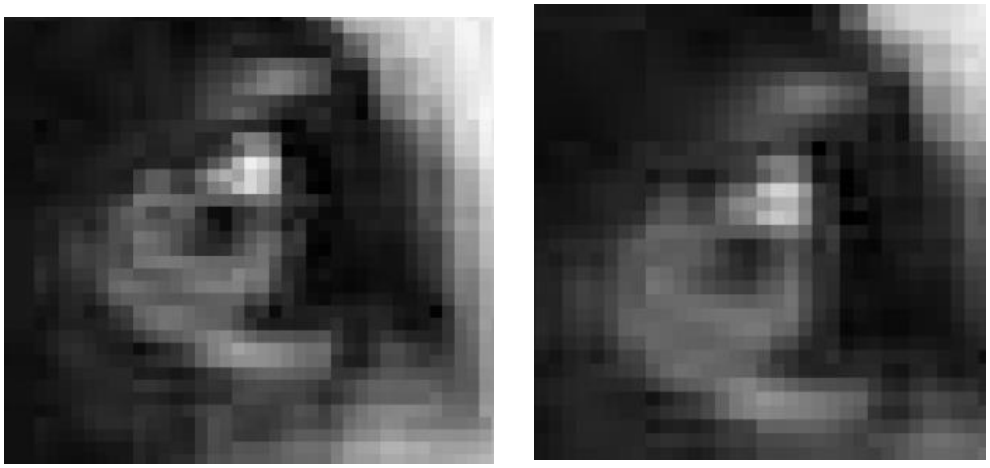

Figure 2.3.2.2 – Zoom in Original vs Gaussian filtered image

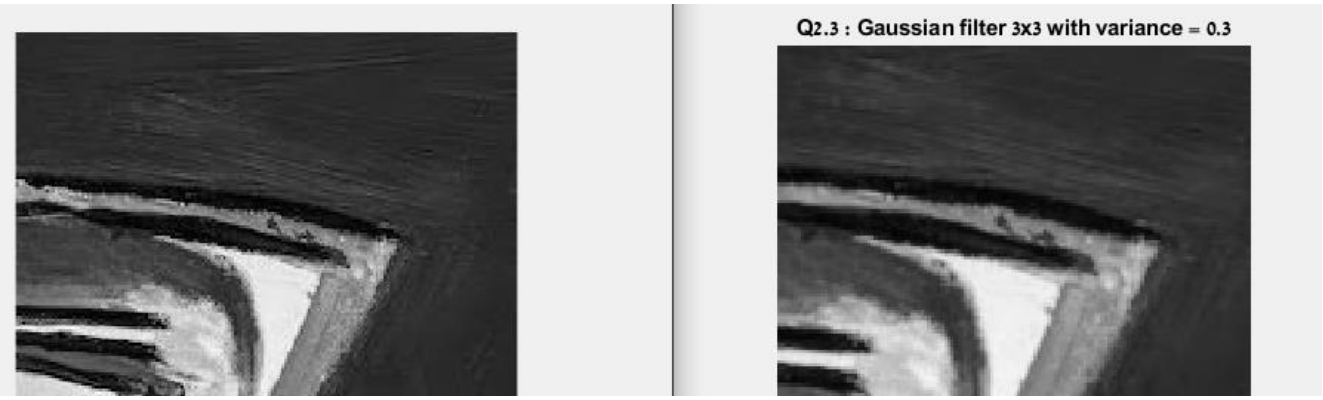

Figure 2.3.2.3 – Zoom in Original vs Gaussian filtered image

Explanation : The Gaussian smoothing operator is a 2-D convolution operator that is used to 'blur' images and remove detail and noise. In this sense it is like the mean filter, but it uses a different kernel that represents the shape of a Gaussian bell. The Gaussian outputs a 'weighted average' of each pixel's neighborhood, with the average weighted more towards the value of the central pixels. We use this kernel function :

$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ When $\sigma^2 = 0.3 = variance$ as required.

3. Subtract the original image from the filtered image. Display the new image using 'imshow(out_img, [])'. Explain what you see.

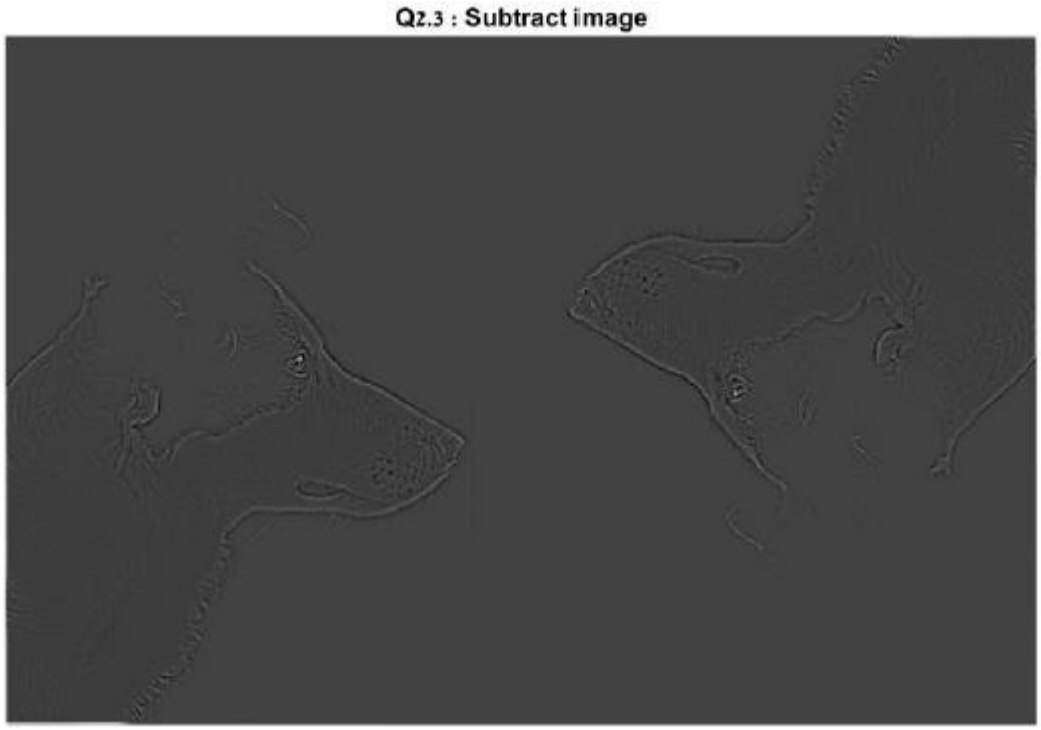

Figure 2.3.3

Explanation : We can notice that in the subtract image are all the edges of different color areas in the image, which equivalent to the image details.

2.4 Anisotropic Diffusion Filter

The attached anisodi_2D.m function performs conventional anisotropic diffusion (Perona & Malik) upon a gray scale image. The filter aiming at reducing image noise without removing significant parts of the image content.

1. Apply this filter over the given image.
2. Display the image and explain what you see.

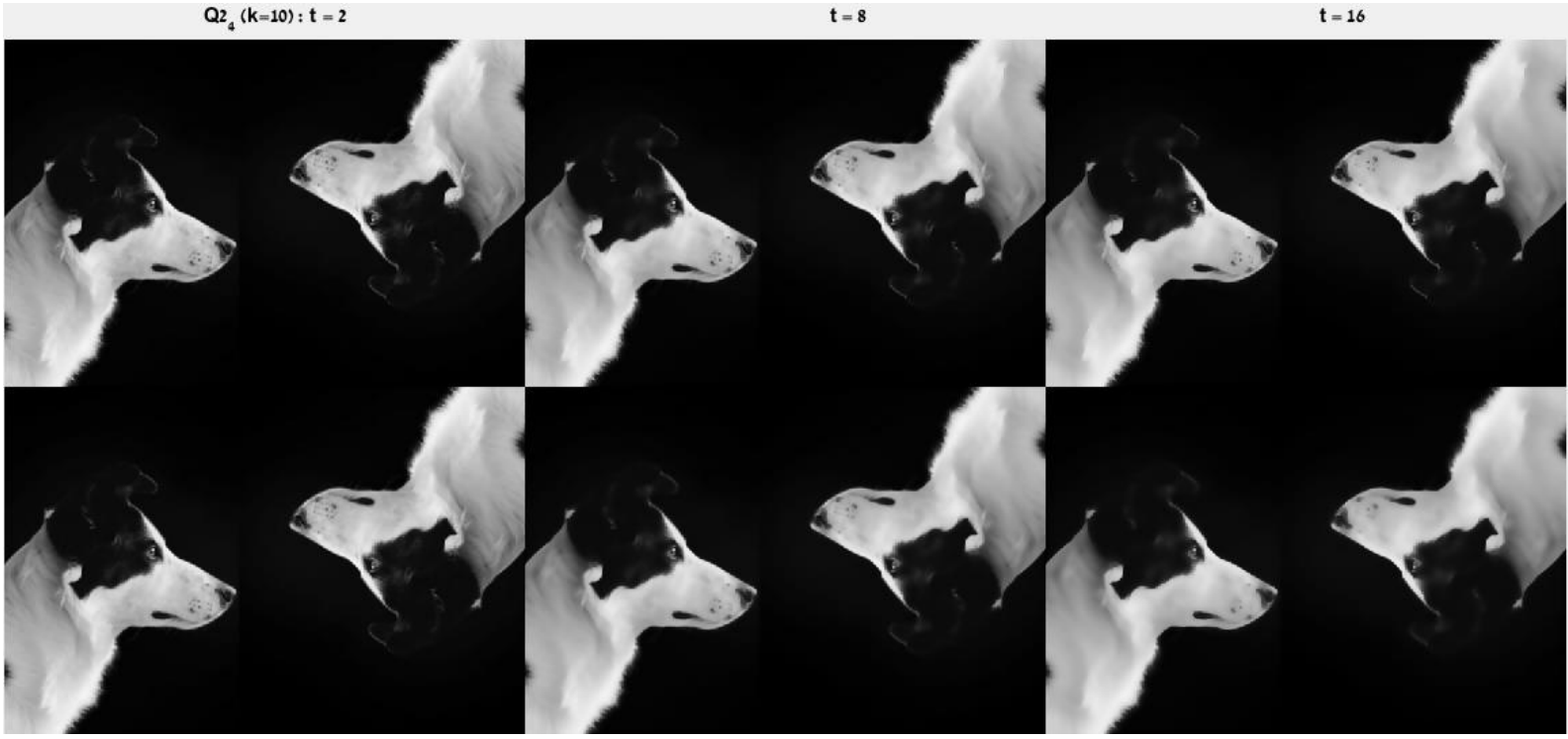

Figure 2.4.1 – k = 10

Note : The upper line is using the first function (exponent) and the second line is using the second function (diversion).

```
% 1 - c(x,y,t) = exp(-(nablaI/kappa).^2),
% privileges high-contrast edges over low-contrast ones.
% 2 - c(x,y,t) = 1./(1 + (nablaI/kappa).^2),
% privileges wide regions over smaller ones.
```

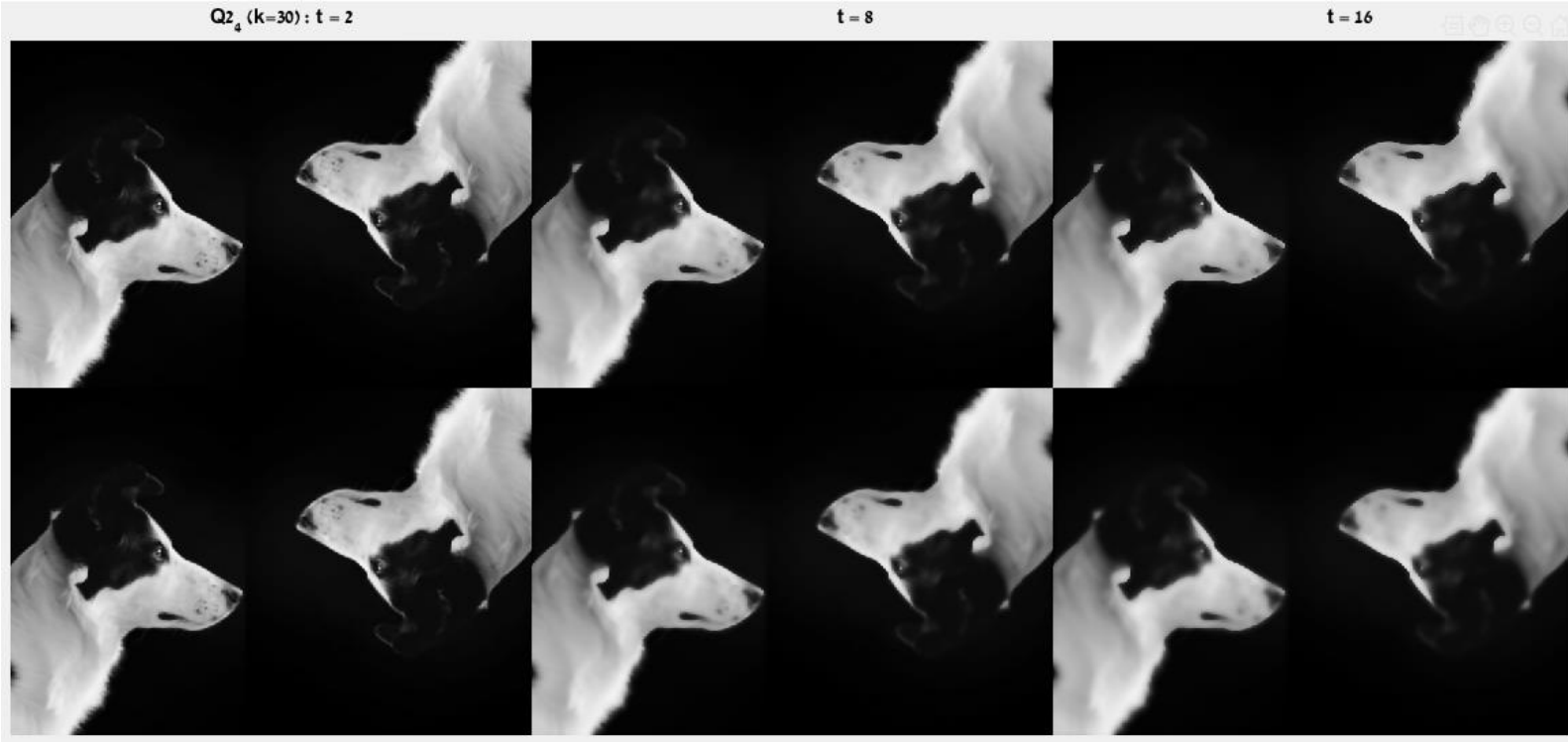

Figure 2.4.2 – k = 30

Note : The upper line is using the first function (exponent) and the second line is using the second function (diversion).

```
% 1 - c(x,y,t) = exp(-(nablaI/kappa).^2),
% privileges high-contrast edges over low-contrast ones.
% 2 - c(x,y,t) = 1./(1 + (nablaI/kappa).^2),
% privileges wide regions over smaller ones.
```

Explanation : Anisotropic diffusion can be used to remove noise from digital images without blurring edges. It's very easy to see that the object get blurred, but its edges stay relatively sharp. As shown at class, we can notice that while the diffusion process blurs the image considerably as the number of iterations increases, the edge information progressively degrades as well.

The flux functions (of this filter) offer a trade-off between edge-preservation and blurring (smoothing) homogeneous regions. Both the functions are governed by the free parameter κ which determines the edge-strength to consider as a valid region boundary. Intuitively, a large value of κ will lead back into an isotropic-like solution.

We been testing with k = 10 and k = 30 as shown above.

2.5 Noise Filtering

1. Create 3 new images by adding 3 different kinds of noises to the original image using imnoise() function. The noises are: 'salt & pepper', 'gaussian' and 'speckle'.
2. Apply the implemented filters (and the anisotropic diffusion filter) on each of the noisy images.
3. Display the noisy and the filtered.
4. What is the effect of each filter on each noise? Which is the best filter for any given noise?

- Speckle noise

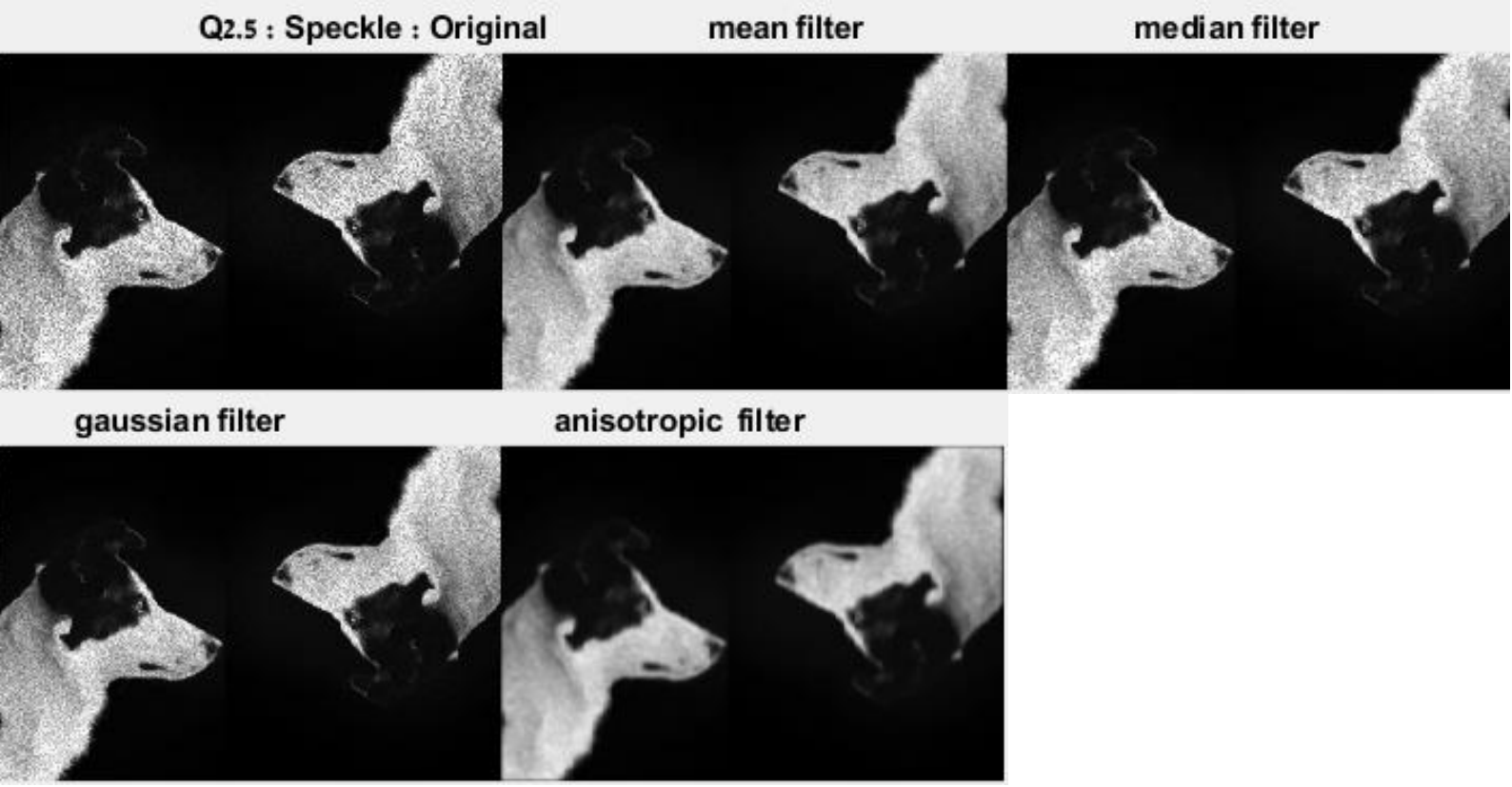

Figure 2.5

Explanation :
Mean filter : Closest to the source, since the averaging operation allows the filter to blur the noise - which have less significant impact on the white areas of the picture in the averaging process.
Median filter : Because the filter is looking for the median, when the noise is high, it cannot be clear, so we see noise in the filtered picture.
Gaussian filter : Because the filter giving more weight to the closest values, when the noise is high concentration the impact of the noise can be seen in the filtered picture.
Anisotropic filter : Managed to clear the noise but blur the image, probably because there is high contrast where there is noise.
Best filter for this noise : The Mean filter had mange to give the best results in terms of reducing the noise (but blur the image), and the Gaussian filter manage to reduce some of it + keep the sharpness of the image.

- Gaussian noise

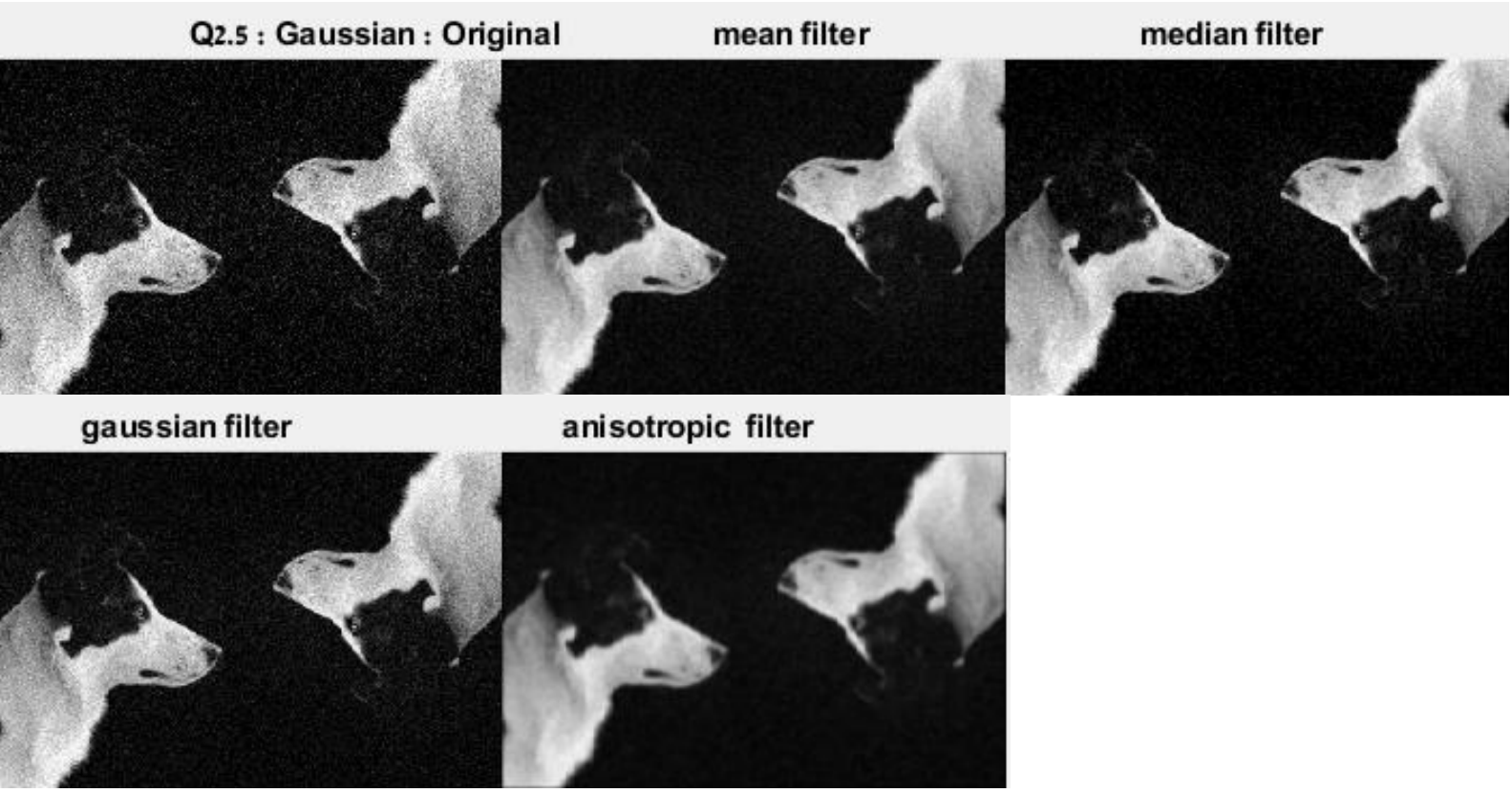

Figure 2.5.2

Explanation :
It is known that this noise is hard to filter because of the normal distribution attribute, so its still have a big impact on the operations of the difference filters.
Mean filter : Closest to the source, since the averaging operation allows the filter to blur the noise - which have less significant impact on the white areas of the picture in the averaging process.
Median filter : Because the filter is looking for the median, it can be clear a bit better on the bright areas (if it get lucky with the distribution of the original values etc'.).
Gaussian filter : Because the filter giving more weight to the closest values, when the noise is high concentration the impact of the noise can be seen in the filtered picture.
Anisotropic filter : Managed to clear the noise on the object but blur the image.
Best filter for this noise : The anisotropic filter in terms of reducing the noise (but blur the image), in terms of overall looks it' s a tie for us for the gaussian and the mean filter.

- Salt & Pepper noise

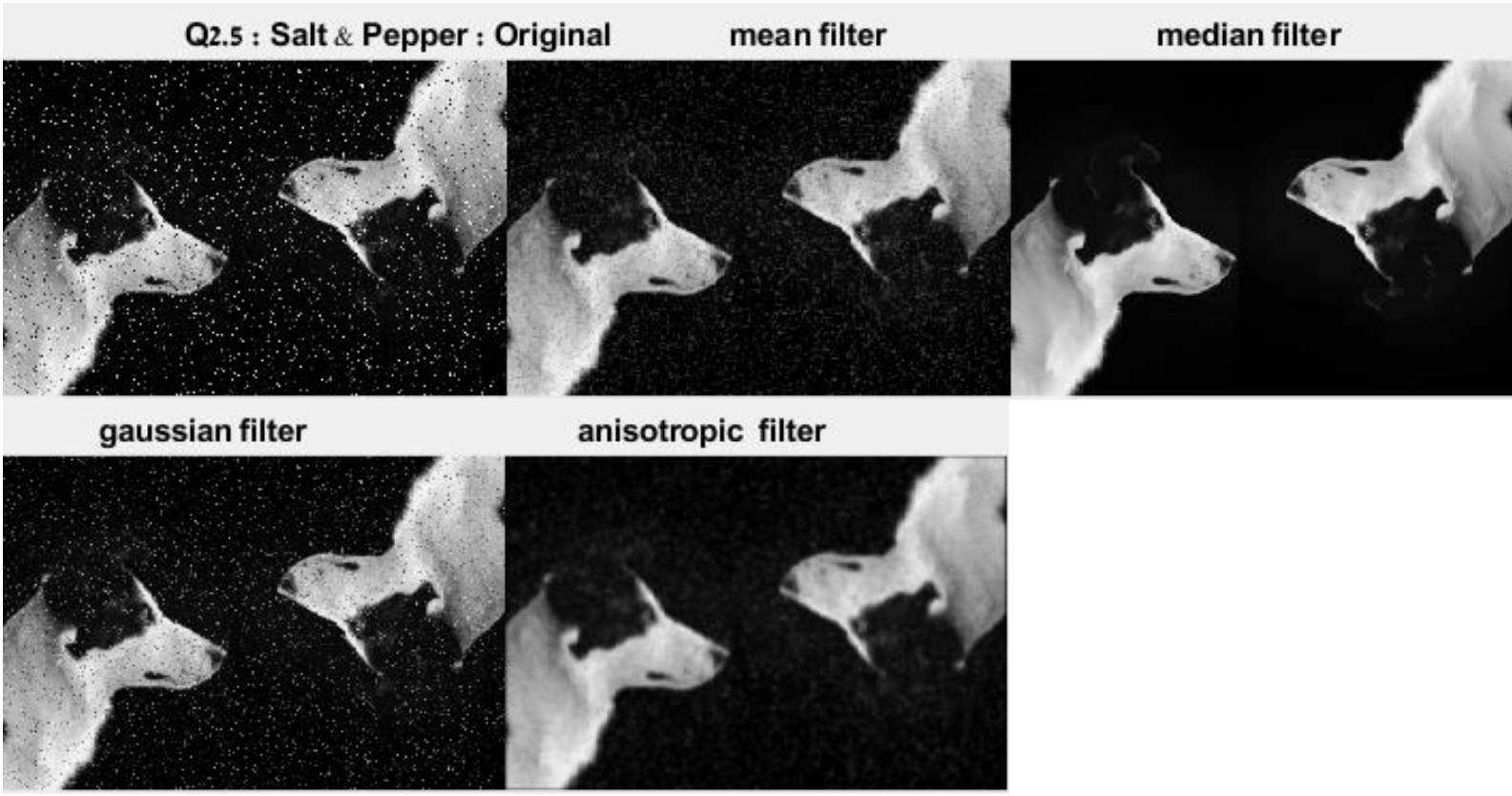

Figure 2.5.3

Explanation :

Mean filter : with The distribution of this noise the mean filter gives it a lot of weight, hence we can still see the noise in the filtered image.
Median filter : Because the filter is looking for the median, it can be clear better even if there is low concentration – it's hard to find the noise values as the median in small part of the environment (3x3 pixels).
Gaussian filter : Didn' t did a good job in the noise.
Anisotropic filter : Reduce the noise in the image but blurred it.
Best filter for this noise : Median filter as shown above – we can see that the noise is completely reduced (from eye sight).

5. What are the pros and cons of each filter?
   With our understanding base on those results :
   - Mean filter
     Pros - Work best when there is no high contrast of values in the sliding window (matrix).
     Cons - Often blur edges and not working good where is high contrast of values.

   - Median filter
     Pros – Work better with high contrast of values. The filtered image is smoother.
     Cons – Not working well if the noise has high concentration, blur edges.

   - Gaussian filter
     Pros - Semi-blur edges and reduce the noise if it has low contrast of values.
     Cons - Not working good where is high contrast of values.

   - Anisotropic filter
     Pros – Preserve object edges
     Cons - When there is high concentration of noise it will success to reduce the image but blur the image (Probably because it will not recognize the edges anymore with the derivative).

2.6 Larger Kernels

Repeat sections 2.2, 2.3 and 2.5. With filter size of 9x9. Explain the difference.

- 2.2

Mean filter

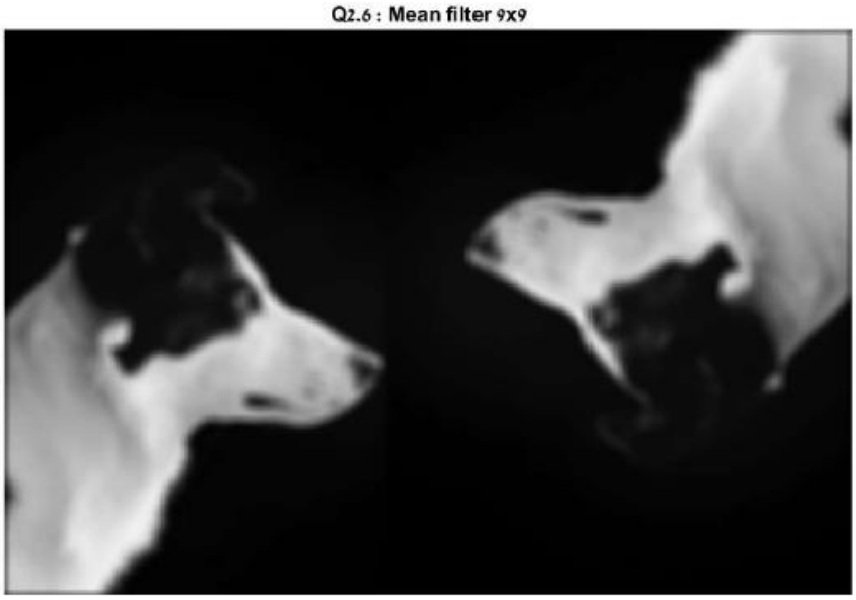

Figure 2.6.1

Difference : Here we get blurrier image, and the edges are not preserve.

Median filter

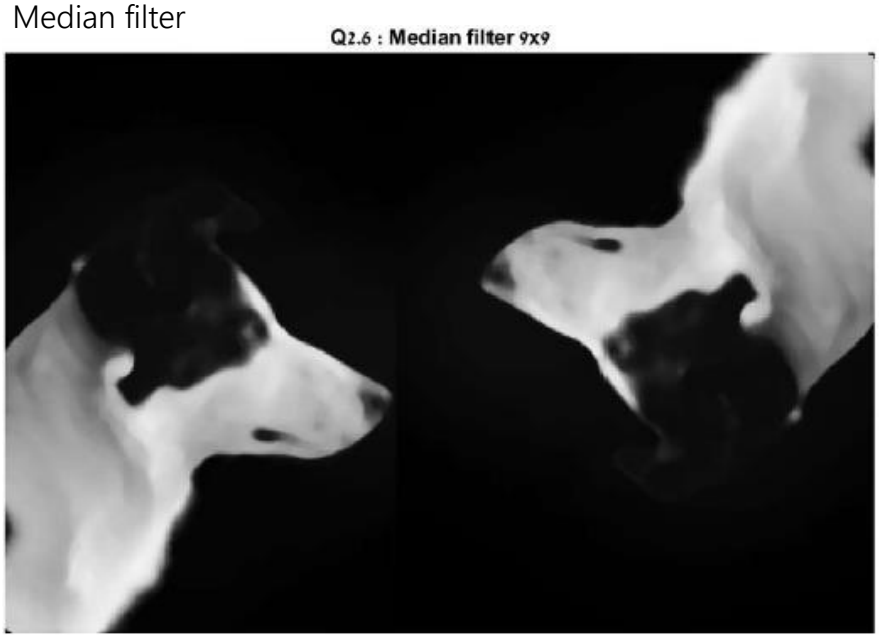

Figure 2.6.2

Difference : As we expected, the median filter preserved the edges well, but the object itself got a 'smudgy' texture to it like oil painting. Although it's a bit blurry, we can still recognize the object well.

❖ 2.3

Gaussian filter

Figure 2.6.3

Subtract image

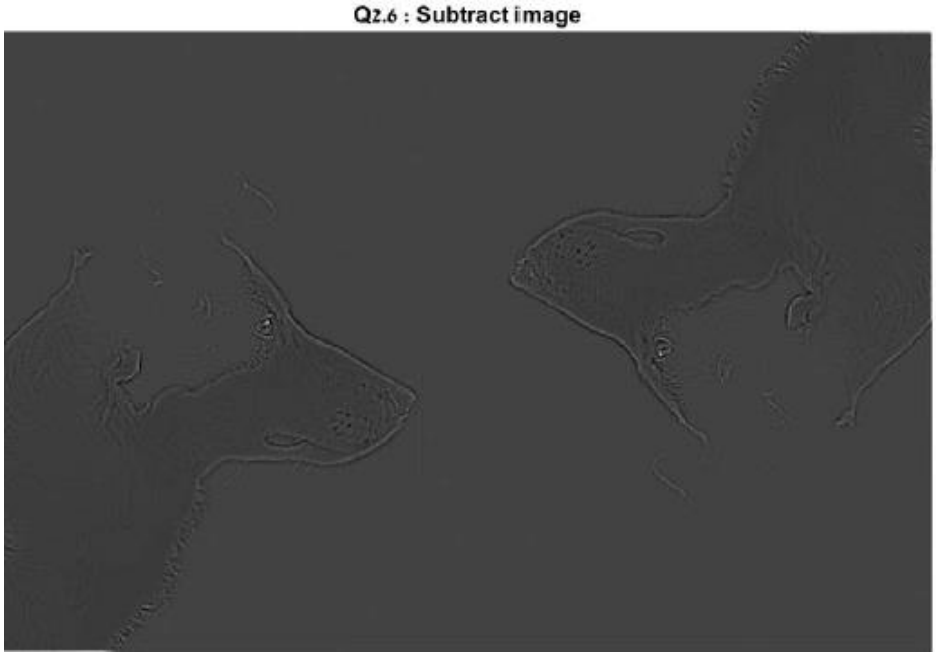

Figure 2.6.4

Difference : Now we have more details in the subtract image. The Gaussian has more pixels in the neighborhood (81 in 9x9 matrix oppose to 9 in 3x3 matrix) to give weight to, so if the noise has no high concentration it will get less weight hence it will have less impact on the results = reduce noise.

❖ 2.5 – Noise filtering

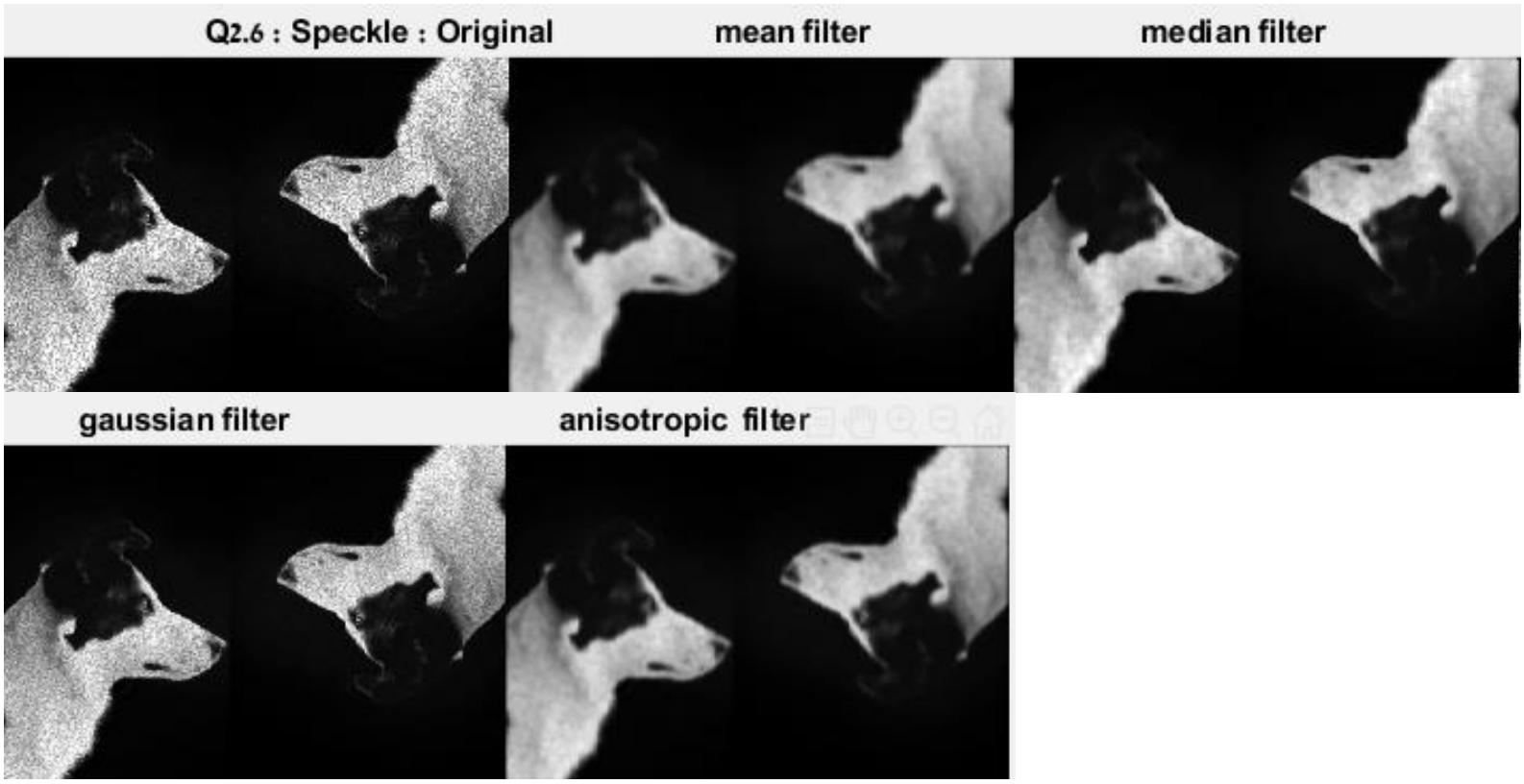

Figure 2.6.5 – Speckle Noise

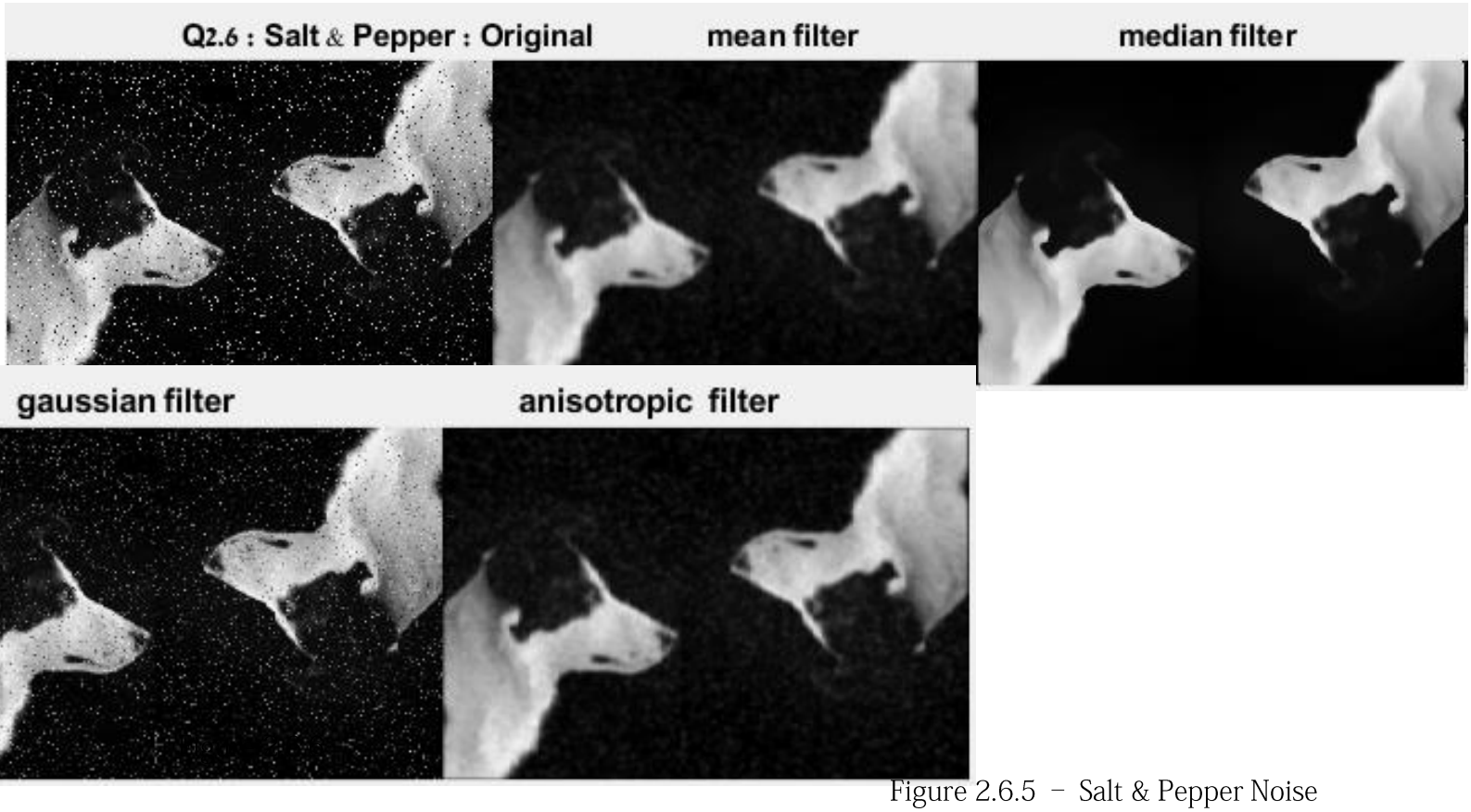

Figure 2.6.5 – Salt & Pepper Noise

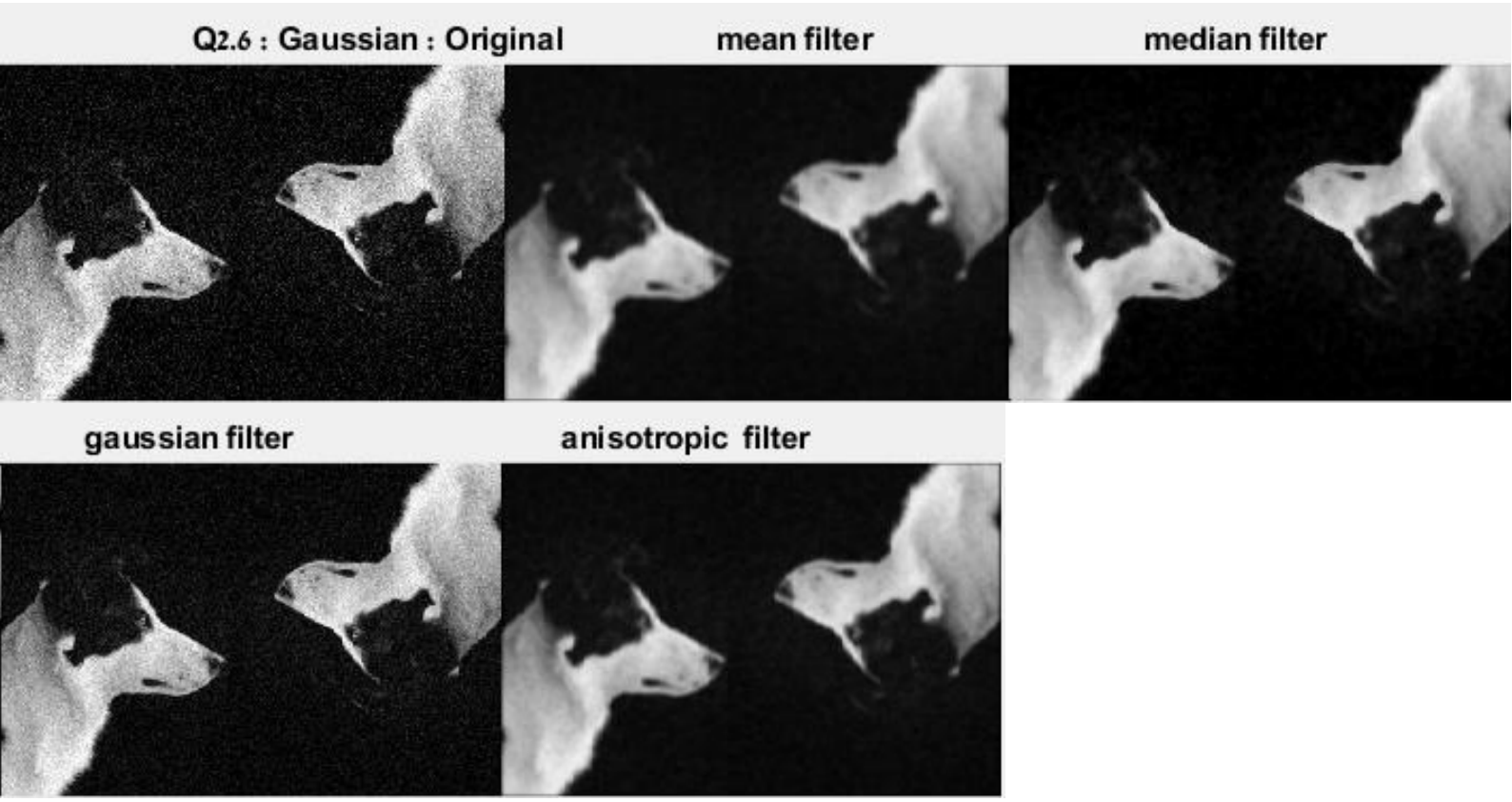

Figure 2.6.7 – Gaussian Nosie

Difference :

We can notice that filters based on averaging the neighbors or find the median in them are perform much better result, thats because now with 9x9 matrix we have much more samples – so now the noise (easy to spot with low concentration noise like speckle noise) will affect less on the final values in the filtered image, resulting reduce noise. But this come with a price, and the price is that we consider more values around the pixel that may or may not have the same shades in common, resulting blurrier images. The gaussian for example cannot blur the image that much comparing to the other filters but can reduce the noise with the high contrast (e.g. Salt & Pepper noise).

In Conclusion : In this assignment we learned basics of image manipulation using histograms and filters. We implemented our self some of the basic operations that the image processing tool-kit is providing us, which improve our understanding about the difficulties in those tasks, and common solution to how to deal with them.