Signal and image processing: Assignment 2

Mariuta Nicolae 10-09-2014

Exercise 1.1.

- CDF(cumulative distribution function) is the integral of the PDF(probability density function)
- This means that thee PDF if the first order derivative of CDF
- CDF is always ascending because the values of PDF are always positive and in discrete domain, the integral is a sum of values from the PDF(which are always positive).

Exercise 1.2.

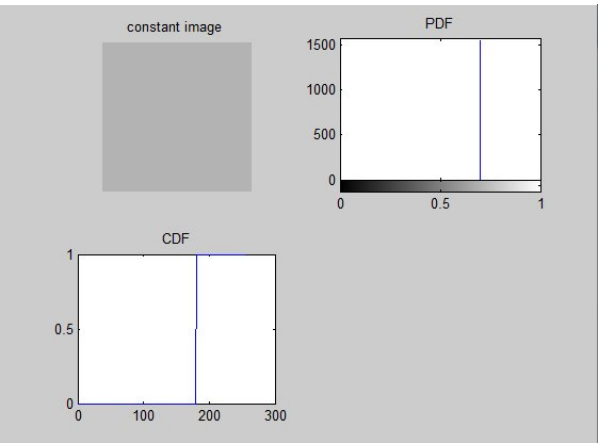


Figure 1 Example of constant image and the results for calculating the PDF and CDF of the image

- For a constant image, the PDF shows a single Dirac at the value *x* where the values of the pixels are located
- Because the PDF is like a Dirac and CDF is the integral of PDF, the CDF looks like a stair, values of *f(x)* are 0 until the value of the pixel2 and 1 after the value of the pixels

Exercise 1.3.

```
function C = imChist(I)
H = imhist(I);
C = cumsum(H);
count = size(I,1)*size(I,2);

for i = 1 : size(C)
    C(i) = C(i)/count;
end

end
```

Figure 2 Function for calculating the cumulative histogram of an image

- I have implemented the function for calculating the cumulative histogram of an image by using the Matlab function *imhist()* to calculate the histogram of the image.
- In the next step I used the function *cumsum()* of Matlab function to obtain the cumulative histogram
- In the final stage I calculated the number of pixels by multiplying the sizes of the image and used the result (*count*) to divide all the values obtained from using the *cumsum()* function in order to normalize the result(have values between 0 and 1)

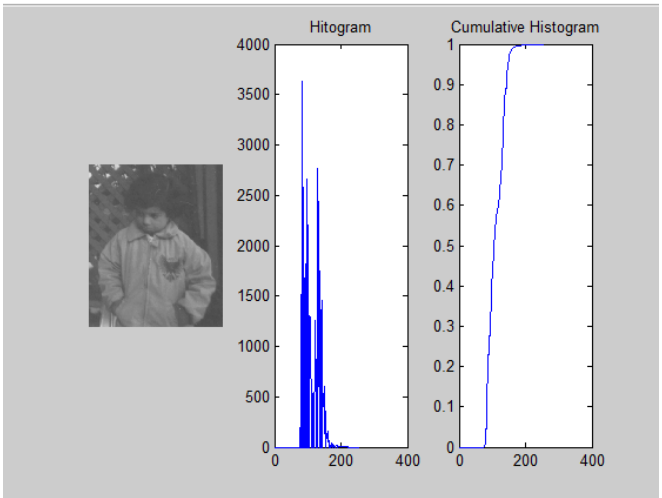


Figure 3 The *pout.tif* image, the histogram and the cumulative histogram of the image

- By using the own function for cumulative histogram on the *pout.tif* image, I get a function that is growing very fast on a short interval from 0 to 1.
- regions of fast growing correspond to regions where there are the largest values in the histogram (most of the pixels have values between 50 and 180)
- the flat regions correspond to pixel values that are not present in the picture, to regions where the values in the histogram are 0
- this is also an effect of that the CDF is the first integral of the PDF(sum of values from PDF). In the regions where there are summed high values of PDF, the value is increasing very fast and in the regions where we add 0 or very small values the CDF is constant
- in the picture itself I notice that there are no areas with very white pixels of values of black pixel. There are mostly grey values centered in the middle of the interval 0-255

Exercise 1.4.

```
function I2 = fpImage(I)
I2 = zeros(size(I));
C = imChist(I);

for W = 1: size(I,1)
    for L = 1: size(I,2)
        pixel = I(W,L);
        I2(W,L) = C(pixel);
    end
end
end
```

Figure 4 My own function that takes an image and computes the floating-point image C(I) for given I image.

- I created function *fpImage* that at first calculates the cumulative histogram of the image and then computes all values in the Image such that the intensity at each pixel *x* is *C(I(x))*.
- I used a for in for loop to read each pixel in the image and then look in the cumulative histogram and replace the pixel value with the value *y* from the cumulative histogram

- Using for in for loop is not a fast way to make the computation as it has $O(?^2)$ time order but I did like this to have better understanding of the transformation. Matlab has methods to make the computation a lot faster

- By applying the function that I created over the *pout.tif* image I obtain an image with a lot better contrast. That is happening because the cumulative histogram of the original image is a very fast growing slope and for the value of a pixel there is higher chance to have values of 0(black)or 1(which is pure white).

Figure 5 Result for applying the fpImage over the *pout.tif* image

Exercise 1.5.

- The CDF is invertible only of it has no flat regions. Because if there are more than 1 value *x* for which *y = f(x)*, we will not know which is the value x for which we have the value y. For pictures it happens very often that to be flat region (regions in the PDF that have 0 values) so in most of the cases the CDF is not invertible

```
function Cinv = invChist(C)
Cinv = (1000);

for i = 1: 1000
    l = (i-1)/999;
    for j = 1:255
        if(C(j)>=l)
            Cinv(i) = j;
            break
        end
    end
 end

end
```

Figure 6 The implementation of the pseudo-inverse function of CDF

- The function that I created is using the cumulative histogram of the target image, I split the values between 0 and 1 of the cumulative histogram which is normalized. For each of those 1000 values I look in the cumulative histogram to see which is the min pixel value that is higher or equal with the corresponding pixel value.
- Once again, I could use *min* or *find* functions of Matlab to avoid the for loops and have shorter computation times, but I have chosen this implementation to have better understanding of the function

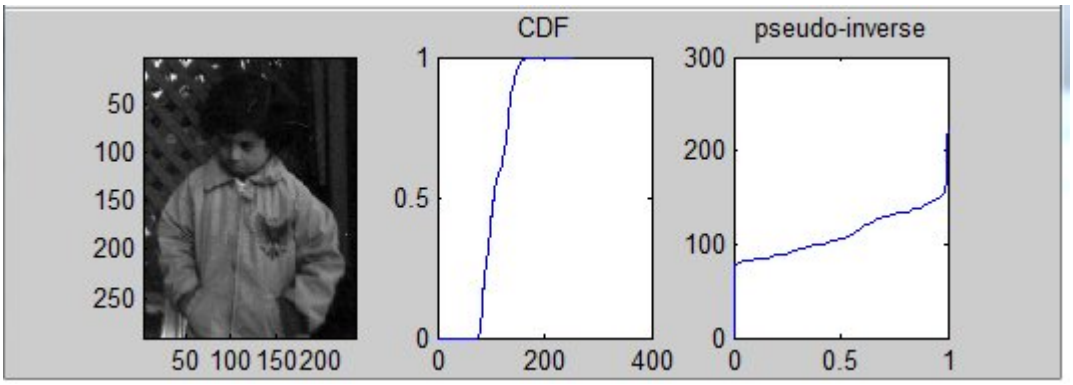


Figure 7 Pseudo-inverse of the CDF obtained for pout.tif image

By testing the function that I created over the *pout.tif* image I can see that my implementation is correct. In the CDF plot, the function is growing starting from aprox. 80 pixel value and in the inverse function I have values of 80 right after *x=0.*

Exercise 1.6.

```
C1inv = invChist(C1);
C2inv = invChist(C2);
I4 = zeros(size(I1));

for W = 1: size(I4,1)
    for L = 1: size(I4,2)
        pixel = I1(W,L);
        inv = C1(pixel);
        out = C2inv(max(floor(inv*1000),1));
        I4(W,L) = out;
    end
end
```

Figure 8 Part of the code that is doing the histograms matching

- The original image has the cumulative histogram C1, and the target histogram is C2
- I create a matrix full of zeros with size of the image that has to be transformed
- I used my own functions to calculate the histograms and the inverse of the histograms
- for each pixel in the original image, I look for the result (*f(x)*) and for that value I look in the inverse histogram of the target picture to get the corresponding pixel value
- I could use the *find* function from Matlab to obtain a faster function but I have chosen the for loops to have better understanding of the procedure of histogram matching

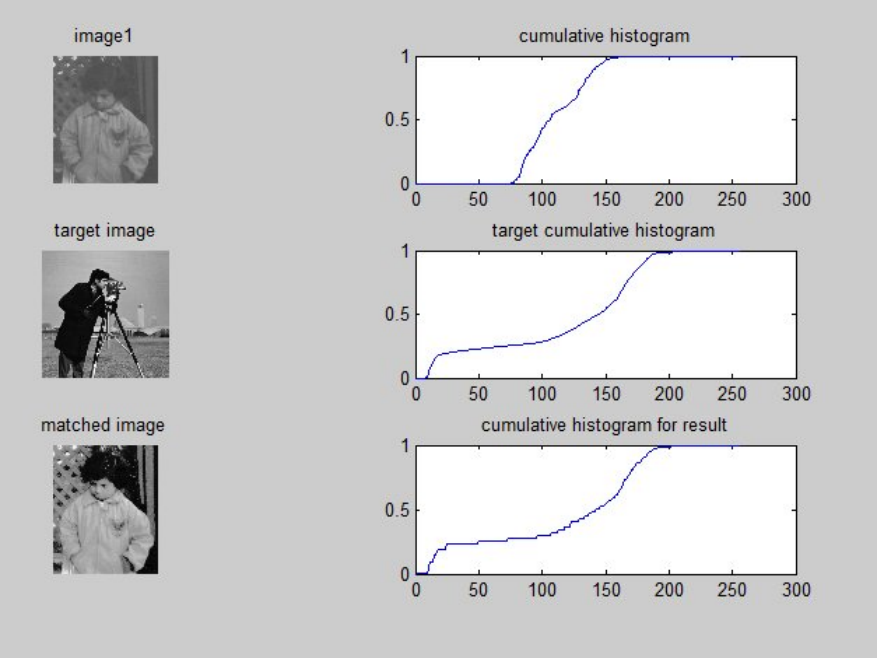


Figure 9. Results for applying the histogram matching having the pout.tif as image to transform and cameraman.tif histogram as the target one.

- By testing the procedure that I created, I notice that the cumulative histogram of the original image looks very much like the cumulative histogram of the target image.
- *pout.tif* image has a very fast growing slope and bad contrast while the *cameraman.tif* image has a better contrast and slower growing cumulative histogram.
- by making the histogram matching, I obtain an image with a better contrast than the original one and the cumulative histogram looking very much like the one of the target histogram, with slower growing slope

Exercise 1.7.

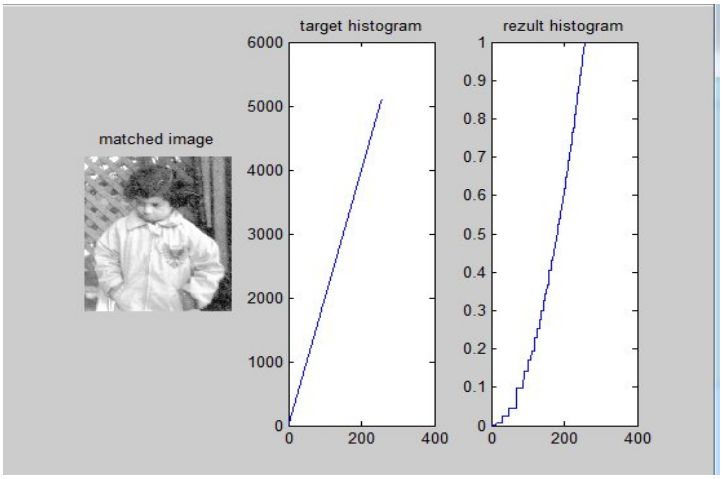
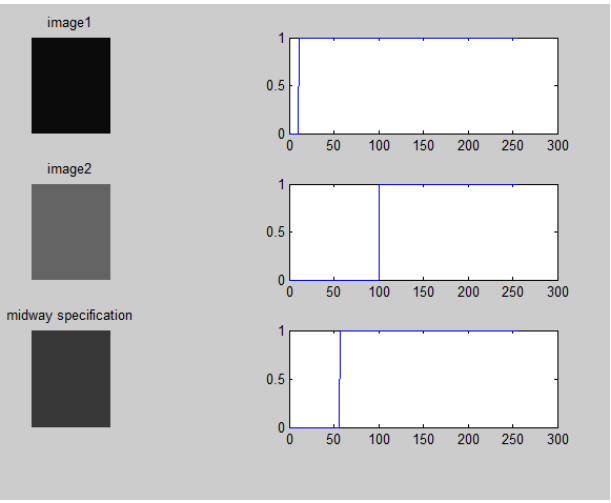


Figure 10 Restults for matching the histogram with an image having a constant histogram

- For an image with constant histogram(equal number of pixels for each grey value), the cumulative histogram is a constant ascending function
- By doing the matching, the resulting histogram looks almost the same like the target one
- By doing the matching with the cameraman in the image from exercise 1.6, I obtain an image with better contrast. But for an image with constant histogram, the matched image has more faded colors

Exercise 1.8.

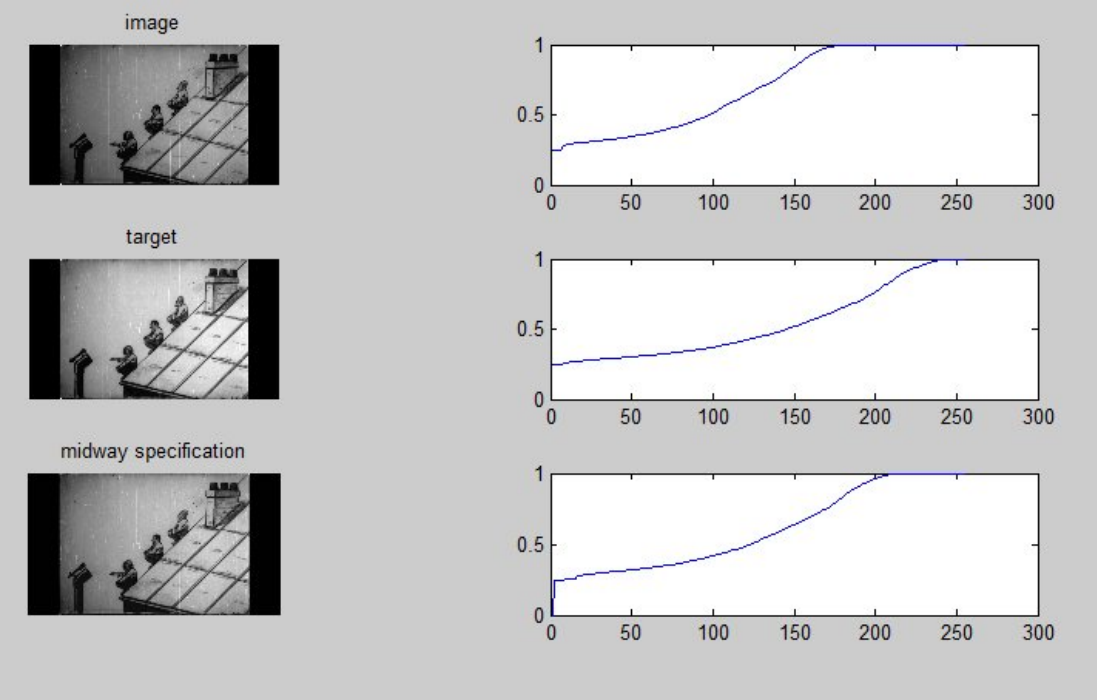


- By calculating the midway specification for 2 images with constant pixel values (one with values of 10 and the other one with values of 100) As expected, in the resulting image the pixels have values of 56, the average of pixel values in both images

Figure 11. Cumulative histogram for applying the midway specification on 2 constant images

- As demonstration, if both images are constant, then I1 will have all pixels with value *a* and I2 has all pixels with value *b*
- The cumulative histograms for these 2 images will both look like the histogram in Exercise 1.2, and the values of *x* where the stairs will be located are *a* and *b*
- The inverse of these 2 cumulative histograms will be 2 constant function $?_1^{-1} = ?$ and $?_2^{-1} = ?$
- If the equation from the exercise is used, then the result will be $\frac{?+?}{2}$ so the midway specification will be the average of the constant image

Exercise 1.9.



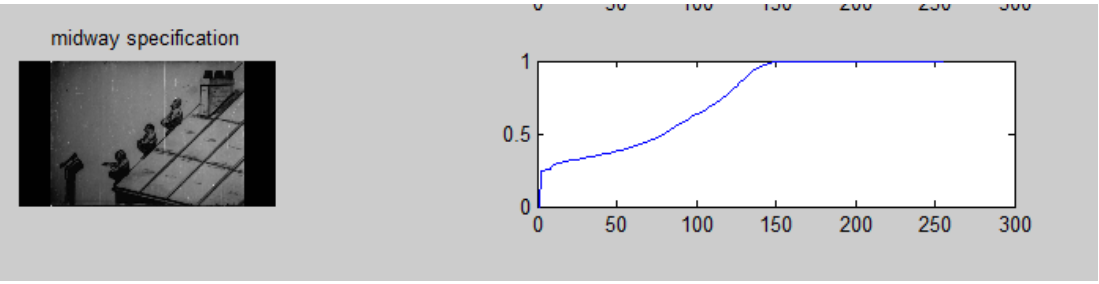


Figure 12 Results for computing the midway specification over 2 sequences of a movie. The first one is darker and the other one has lighter colors.

- The resulting image for matching the darker image with the midway specification makes it to become lighter
- The resulting image for matching the lighter image with the midway specification makes it to become darker
- The resulting cumulative histogram in both cases look much like the histograms for the target histogram
- For and arbitrary number of images, the sum of the cumulative histograms should be calculated and then divided by the number of images
- In case of having more sequences from grey-scale movies that are affected by noise, by applying the midway specification for all of them, we can increase the quality of old movies

Exercise 2.1.

- If we consider the convention of eq. (4.1), the filters would be: $\begin{bmatrix} \frac{-1}{2} \\ 0 \\ \frac{1}{2} \end{bmatrix}$ for the first image derivative and $\begin{bmatrix} \frac{-1}{2} & 0 & \frac{1}{2} \end{bmatrix}$ for the second image derivative
- If we consider the convention of eq. (2.20), the filters would be: $\begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{-1}{2} \end{bmatrix}$ for the first image derivative and $\begin{bmatrix} \frac{1}{2} & 0 & \frac{-1}{2} \end{bmatrix}$ for the second image derivative
- If investigate the Matlab help for the *imfilter* function, we get the following:

```
- Correlation and convolution

    'corr'       imfilter performs multidimensional filtering using
                 correlation, which is the same way that FILTER2
                 performs filtering.  When no correlation or
                 convolution option is specified, imfilter uses
                 correlation.

    'conv'       imfilter performs multidimensional filtering using
                 convolution.
```

Figure 13. Answer from calling the Matlab help for the function imfilter which describes how does Matlab deal with these 2 conventions

- So Matlab is going to use the convention in (4.1) if the *imfilter* has the parameter *'corr'* and the convention in eq.(2.20) if we set the parameter *'conv'* for *imfilter*
- The results for the Gaussian or mean filter will be the same, no matter which convention is used because both filters are symmetric. In case of filters that are not symmetric, then there will be different results for using each of these two conventions

Exercise 2.2.

- In the previous question, there was used only a simple derivative difference approximation on one direction for each filter making them very sensitive to noise
- From the expression given in 4.5.2.1 we can notice that Sobel and Prewitt filters are doing a derivative difference approximation on one direction and they are also applying one averaging on the other direction causing them to be more robust to noise. Sobel filter is using the Gaussian averaging for the other direction making it even more robust to noise.

Exercise 2.3.

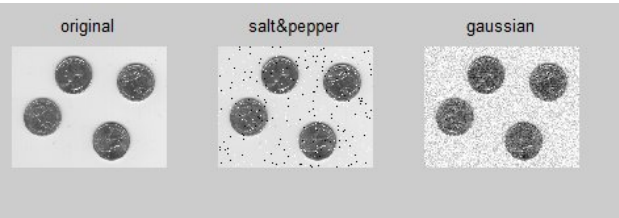


Figure 14 Restult of applying salt&pepper and the Gaussian filter on a image

Figure 15 Results for applying the mean filter with window size of 3*3

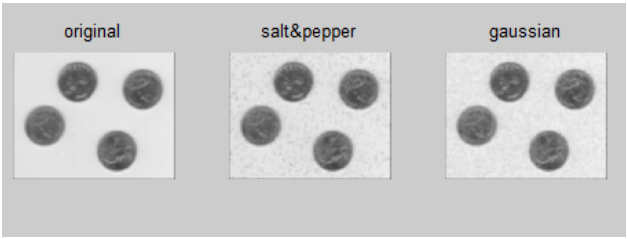


Figure 16 Results for applying the mean filter with window size of 6*6

Figure 17 Restult for applying the median filter over the noisy images with window size of 3*3

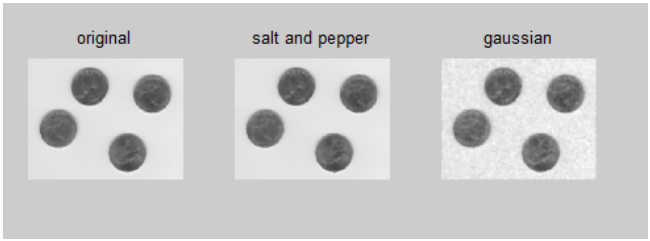


Figure 18 Restult for applying the median filter over the noisy images with window size of 6*6

- Using the mean filter over the noisy images I noticed that it is very effective for removing the Gaussian filter but with loss for the quality of image. It is not effective for removing the salt and pepper noise. By increasing the window size I noticed even more reduction in quality of the resulting images. The image affected by salt and pepper noise became very blurred
- Using the median filter, both types of noises were removed very well, but there is a loss of quality in case of the image affected by Gaussian noise. And there is a higher loss of quality in case of increasing the filter window for all the images (the original one, the SP and Gaussian)
- In case of analyzing the computational times needed for filtering using both filters and different sizes for the window, I noticed that the time needed by mean filter is almost the same (around 4.8seconds) while the time needed by the median filter is a lot higher (20seconds for 3*3 window and 12 seconds for 16*16 window). This happens because the median filter has to do more arithmetical operations(first sort the pixel values and then take the middle one). Sorting algorithms have higher time orders (usually nlogn) while the mean filter only needs to sum the pixels within the window and then divide them to the number of pixels.
- I can conclude that each type of filter is better in specific cases: mean filter is better for removing Gaussian noise because it has good results and fast computational times while the median filter is better for removing the salt and pepper noise because it is more effective despite the longer time needed for the computation

Exercise 2.4.

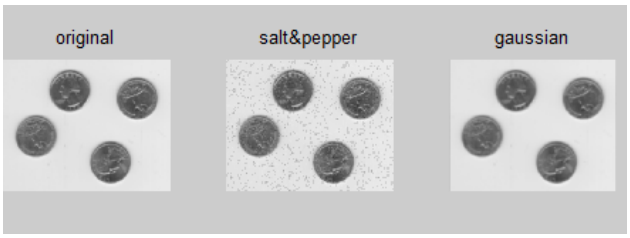


Figure 19. Resulting images for applying 3x3 filter

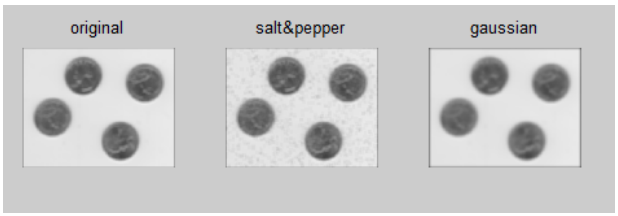


Figure 20 Resulting images for applying 7x7 filter

- Gaussian filter is very effective for removing the Gaussian noise and the results for 3x3 image are very good but the salt&pepper noise is still visible
- By increasing the size of Gaussian filter window the quality of image is decreasing but there is a better removal of salt&pepper noise
- This happens because salt&pepper noise changes the values of pixels in some regins to 0 and 255 while the pixels in the image have greyscale values close to the middle of 0:255 interval. By averaging pixels over a larger window size, the pixels with values of 0 and 255 get to be closer to the values of the surrounding pixels. But this also makes that details of the image to disappear
- Because the edges are not considered because of the window shape, the image is getting thicker and thicker black boundaries

Exercise 2.5.

- In case of changing the value of standard deviation of the Gaussian filter I notice that for low values the details of image are not lost for large window of filter. But the noise removal is no longer that efficient
- In case of using large values for standard deviation the noise removal is more effective but the image is getting blurred and the edges disappear

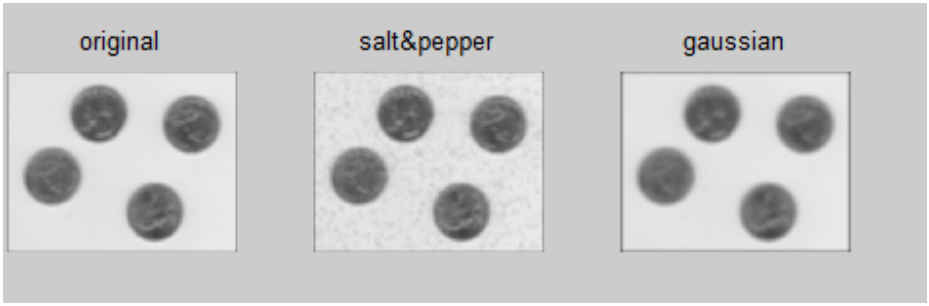


Figure 21 Results for using 7x7 Gaussian filter with standard deviation 10

- I can conclude that standard deviation should be high enough to have good noise removal but not too high because it will make the image edges to disappear