Coşkun Orkun Vural
518181042

BBL588 HOMEWORK 1

Information About Code Repository

Code repository can be found via following the link below:

https://github.com/OrkunVural/BBL588_HW1

All the code pieces and methods are available on the report and the github repository.

Questions & Answers

1. **Download the "Sunny Lake" image.**

"Sunny Lake" image is downloaded from source and inserted into MATLAB using imread() function.

2. **Obtain the gray scale image, I, by taking the average values of R, G, B channels.**

MATLAB's default rgb2gray() command takes the following coefficients:

```
0.2989 * R + 0.5870 * G + 0.1140 * B
```

In order to manually implement that command, following function is created:

```
function [outputImage] = toGrayscale(inputImage,rCo,gCo,bCo)
      red = inputImage(:, :, 1);
      green = inputImage(:, :, 2);
      blue = inputImage(:, :, 3);

      % color components are multiplied with related coefficients
      outputImage = rCo*double(red) + gCo*double(green) + bCo*double(blue);
      outputImage = uint8(outputImage); %for display
end
```

By this way, user can decide on the coefficients of the color channels. Two example values are take in the main script and the results are shown below:

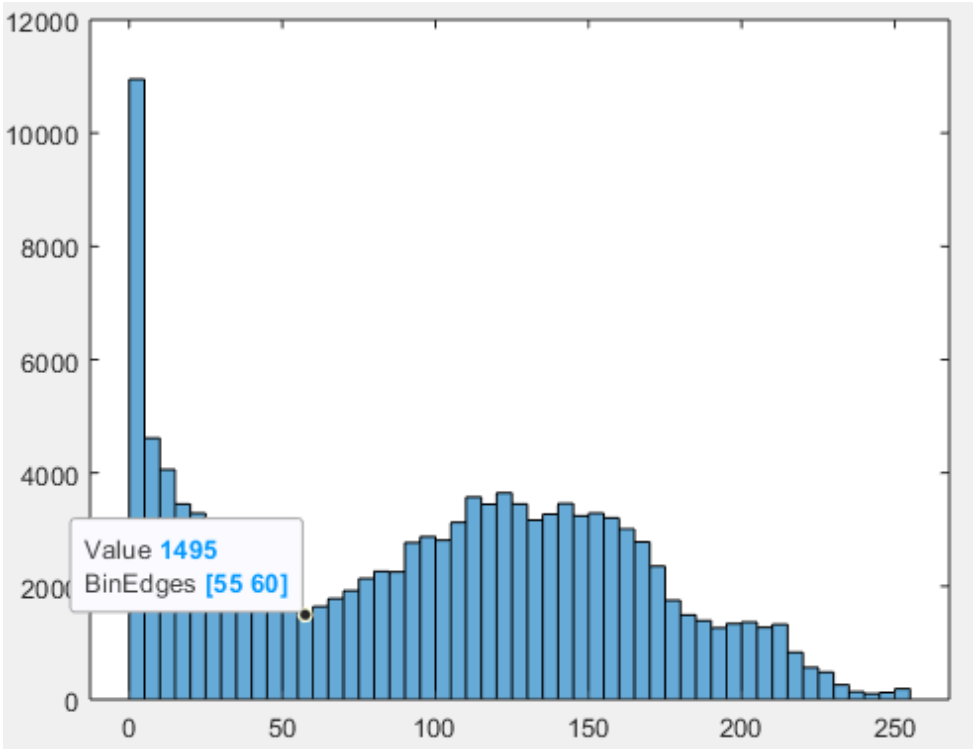
```
I_gray=toGrayscale(I,0.3,0.6,0.1);
figure
imshow(I_gray);
I_gray2=toGrayscale(I,0.7,0.6,0.1);
figure
imshow(I_gray2);
```

Coşkun Orkun Vural
518181042

As it can be seen from the figures above the varying coefficients caused a significant change on the grayscale image. The image on the right hand side is brighter as the red coefficient is higher than the image on the left hand side, which is caused by the sun image.

3. Obtain the histogram, h, of the gray scale image, I.

Histogram of the grayscale image is generated via histogram() command of MATLAB and the resulting histogram is shown below:



Coşkun Orkun Vural
518181042

4. **Inspect h and propose a threshold value, T, to segment the image into two parts and hence obtain a binary image, B: i. Part I: Pixels with intensity values above T. ii. Part II: Pixels with intensity values below T.**

As it can be seen from the figure in part 3 the data point shows the local minima on the gray level between 55-60. Thus 57 is taken as the threshold value for converting the grayscale image to black and white. The following function is generated for thresholding:

```
function [outputImage] = singleThreshold(inputImage,threshold)
sizeX=size(inputImage,1);
sizeY=size(inputImage,2);

outputImage = zeros(sizeX,sizeY);
for i=1:sizeX

    for j=1:sizeY

        if(inputImage(i,j)>threshold)
            outputImage(i,j) = 1; %% If the value is over the threshold it
is converted to 1
        else
            outputImage(i,j) = 0; %% If the value is less than the
threshold it is converted to 0
        end

    end

end

end
```

5. **Present the output image B.**

Output image B is shown below:

Coşkun Orkun Vural
518181042

6. **Add the following zero mean Gaussian noises, separately to red, green and blue channels of 256x256 colored "Sunny Lake" image, with standard deviations of 1, 5, 10, 20. Show resulting images. 7. Obtain gray scale images, I_1, I_5, I_10 and I_20 by taking the average values of R, G, B channels corresponding to different noise levels.**

For adding zero mean gaussian noise to separate channels the RGB image is separated to its channels and imgaussfilt() command of MATLAB is applied to all channels. Function implemented for parts 6 and 7 is shown below:

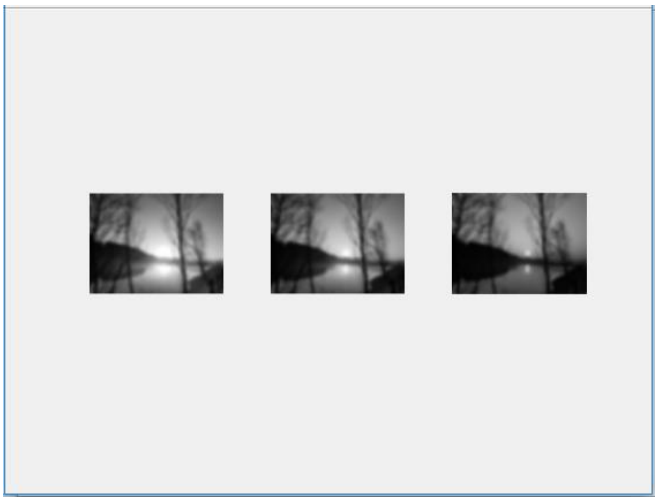
```
function [grayOut] = gaussianBlur(inputImage,sigma)

    %channels of rgb image are separated
    red = inputImage(:, :, 1);
    green = inputImage(:, :, 2);
    blue = inputImage(:, :, 3);

    redBlur = imgaussfilt(red,sigma);
    greenBlur = imgaussfilt(green,sigma);
    blueBlur = imgaussfilt(blue,sigma);

    grayOut = (0.3*redBlur + 0.6*greenBlur + 0.1*blueBlur);

    figure
    subplot(1,3,1)
    imshow(redBlur);
    subplot(1,3,2)
    imshow(greenBlur);
    subplot(1,3,3)
    imshow(blueBlur);


end
```

Output images are shown below(Red, Green, Blue from left to right):

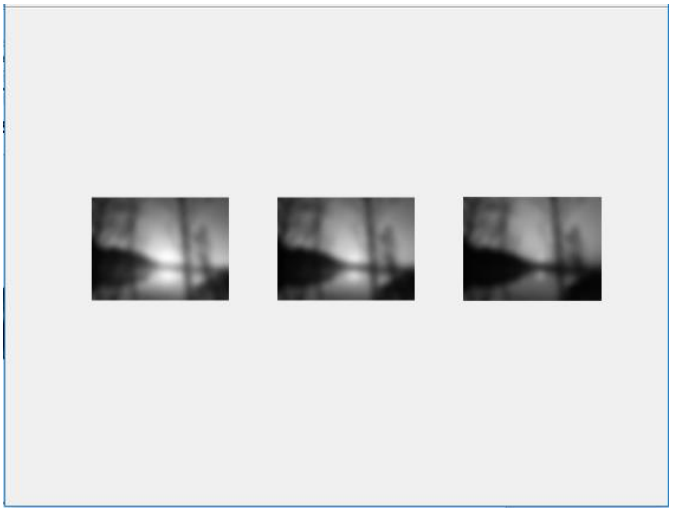
Sigma = 1

Coşkun Orkun Vural
518181042

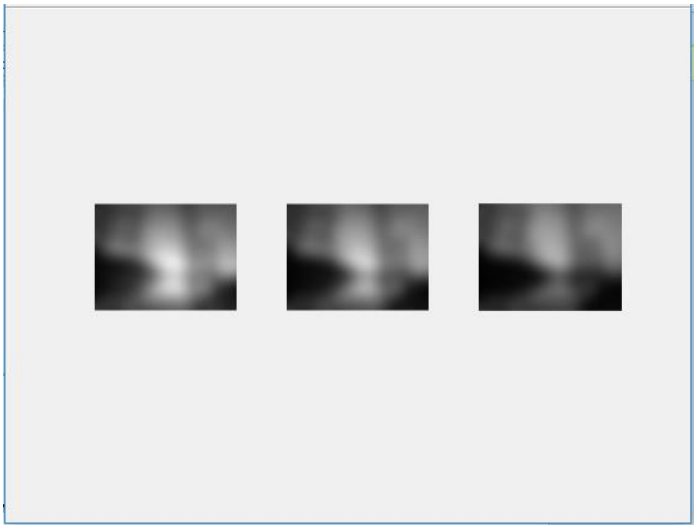
Sigma = 5



Sigma = 10



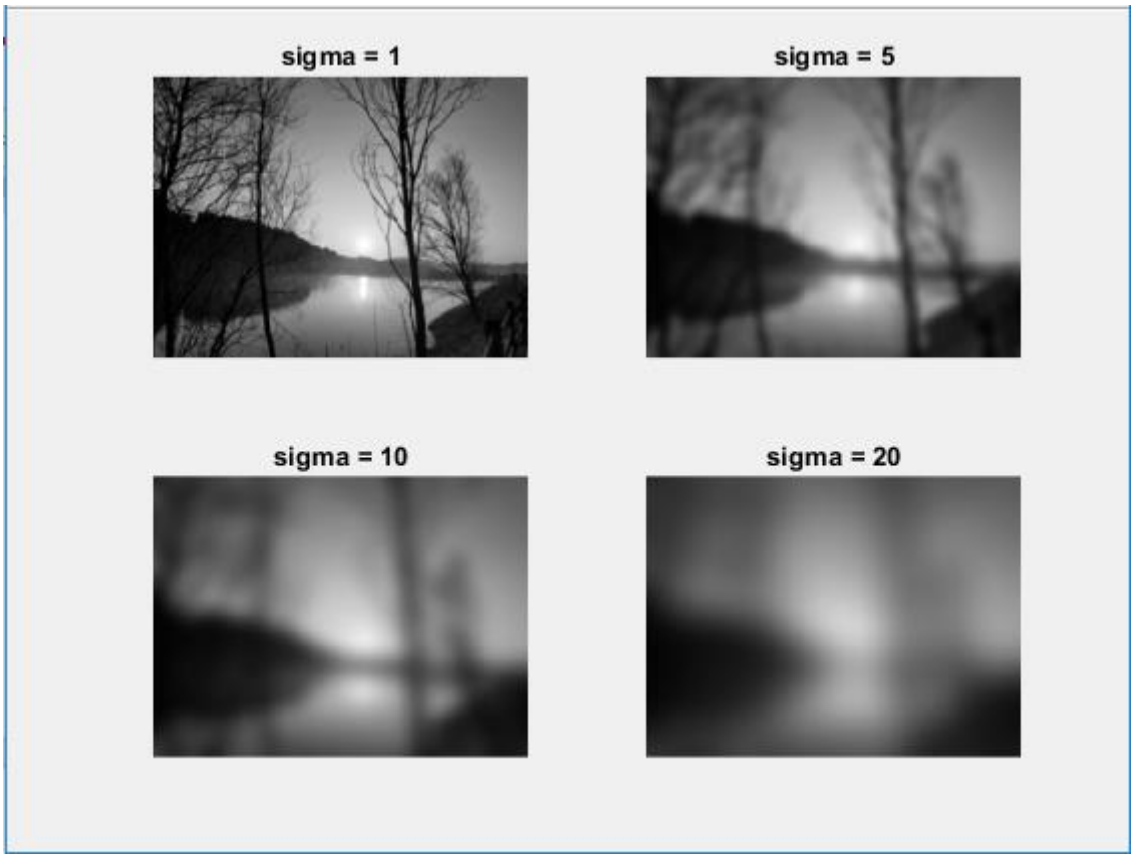
Sigma = 20



Coşkun Orkun Vural
518181042

7. **Obtain gray scale images, I_1, I_5, I_10 and I_20 by taking the average values of R, G, B channels corresponding to different noise levels.**

By using the function in part 6, output grayscale images are plotted and the resulting image is shown below:



By checking the results it can be commented that the amount of blur is increased with increasing sigma.

8. **Filter these images using low-pass filters with kernels presented on pages 9 and 12 of "filter.pdf" document. Comment on the results.**

Filtering images is implemented via 2D convolution of grayscale image with the kernels found on "filter.pdf" document. Function designed for this process is shown below:

```
function [] = lowPass(inputImage)

    LPF_kernel3x3=[
        0.111,  0.111,  0.111;
        0.111,  0.111,  0.111;
        0.111,  0.111,  0.111
    ]; %%Coefficients for 3x3 LPF Kernel
    LPF_kernel5x5=[
        0.04, 0.04, 0.04, 0.04, 0.04;
        0.04, 0.04, 0.04, 0.04, 0.04;
        0.04, 0.04, 0.04, 0.04, 0.04;
        0.04, 0.04, 0.04, 0.04, 0.04;
        0.04, 0.04, 0.04, 0.04, 0.04
```

Coşkun Orkun Vural
518181042

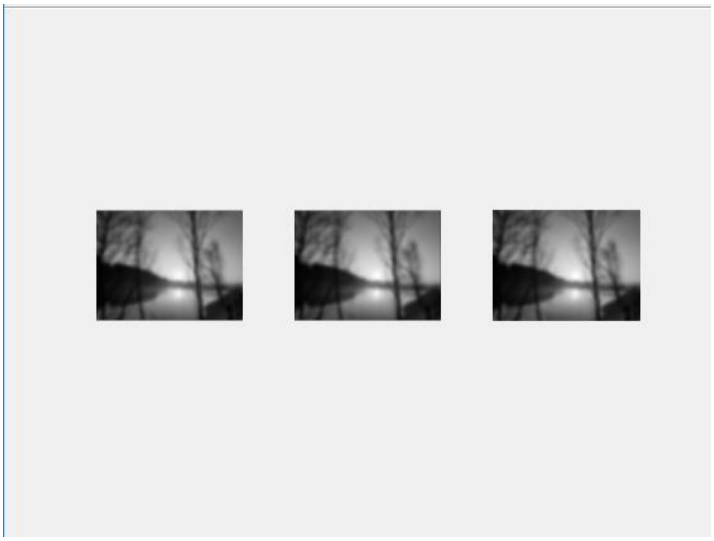
```
    ]; %%Coefficients for 5x5 LPF Kernel
    gaussian_kernel3x3=[
        1/16,   2/16,   1/16;
        2/16,   4/16,   2/16;
        1/16,   2/16,   1/16
     ]; %%Coefficients for 3x3 Gaussian Kernel

    blurred_matrix=conv2(inputImage, LPF_kernel3x3, 'same'); %%2-D
convolution is used for applying 3x3 LPF Kernel on original image
    output_3x3= mat2gray(blurred_matrix); %output matrix is converted to
grayscale image
    blurred_matrix=conv2(inputImage, LPF_kernel5x5, 'same'); %%2-D
convolution is used for applying 5x5 LPF Kernel on original image
    output_5x5= mat2gray(blurred_matrix); %output matrix is converted to
grayscale image
    blurred_matrix=conv2(inputImage, gaussian_kernel3x3, 'same'); %%2-D
convolution is used for applying Gaussian Kernel on original image
    output_gaussian= mat2gray(blurred_matrix); %output matrix is converted
to grayscale image

    figure
    subplot(1,3,1)
    imshow(output_3x3);
    subplot(1,3,2)
    imshow(output_5x5);
    subplot(1,3,3)
    imshow(output_gaussian);

end
```

Resulting images(3x3, 5x5 and gaussian from left to right) are shown below:

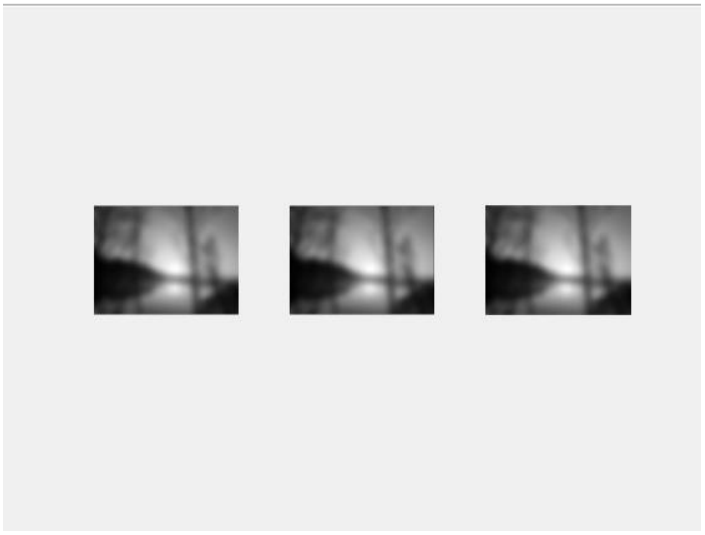
Sigma = 1

Coşkun Orkun Vural
518181042

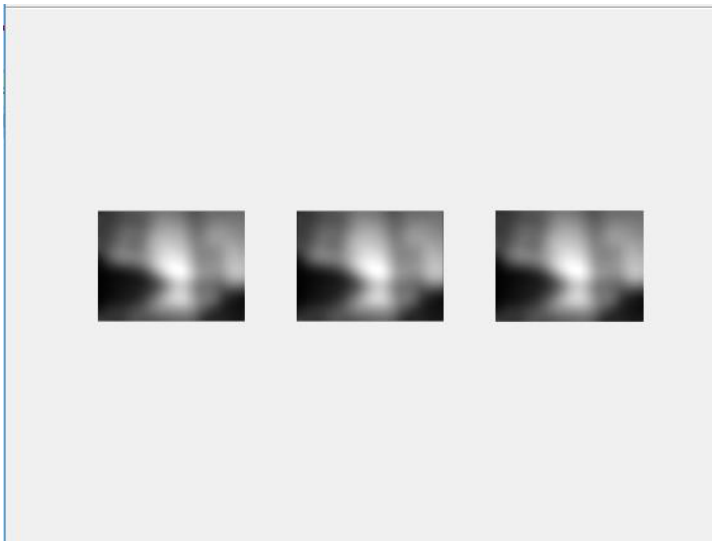
Sigma = 5



Sigma = 10



Sigma = 20



Coşkun Orkun Vural
518181042

Low pass filters above are used for eliminating high frequency components from the image but as the images on part 7 have already preprocessed to eliminate these components there is not a significant change in the images after using filters.

9. **Filter images in 7) using high-pass filters with kernels presented on pages 17 and 19 of "filter.pdf" document. Comment on the results.**

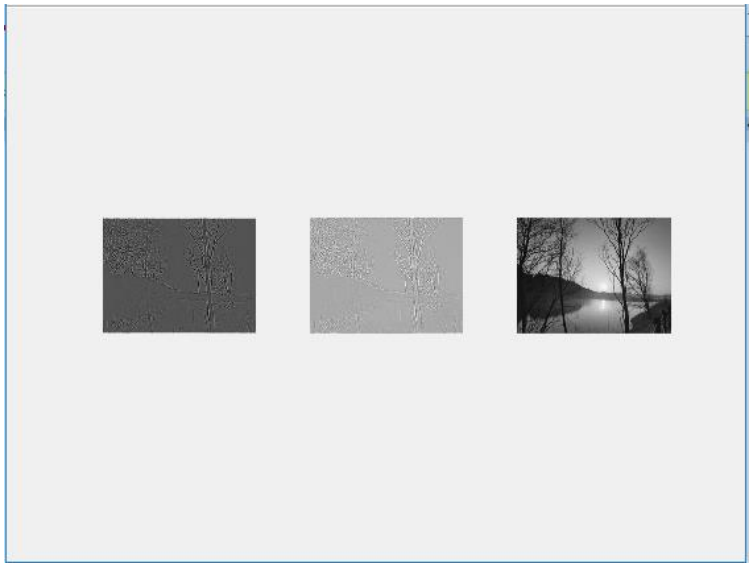
Filtering images is implemented via 2D convolution of grayscale image with the kernels found on "filter.pdf" document. Function designed for this process is shown below:

```
function [] = highPass(inputImage)

    HPF_kernel3x3=[
        -1, -1, -1;
        -1, 8,  -1;
        -1, -1, -1
    ]; %%Coefficients for 3x3 HPF Kernel
    HPF_kernel3x3_2=[
        0.17, 0.67, 0.17;
        0.67, -3.33, 0.67;
        0.17, 0.67, 0.17
    ]; %%Coefficients for 2nd 3x3 HPF Kernel
    boost_kernel3x3=[
        -1, -1, -1;
        -1, 12.5,   -1;
        -1, -1, -1
     ]; %%Coefficients for 3x3 High Boost Filter Kernel -- A=1.5

    blurred_matrix=conv2(inputImage, HPF_kernel3x3, 'same'); %%2-D
convolution is used for applying 3x3 HPF Kernel on original image
    output_3x3= mat2gray(blurred_matrix); %output matrix is converted to
grayscale image
    blurred_matrix=conv2(inputImage, HPF_kernel3x3_2, 'same'); %%2-D
convolution is used for applying 3x3 HPF Kernel on original image
    output_3x3_2= mat2gray(blurred_matrix); %output matrix is converted to
grayscale image
    blurred_matrix=conv2(inputImage, boost_kernel3x3, 'same'); %%2-D
convolution is used for applying High Boost Kernel on original image
    output_boost= mat2gray(blurred_matrix); %output matrix is converted to
grayscale image

    figure
    subplot(1,3,1)
    imshow(output_3x3);
    subplot(1,3,2)
    imshow(output_3x3_2);
    subplot(1,3,3)
    imshow(output_boost);

end
```

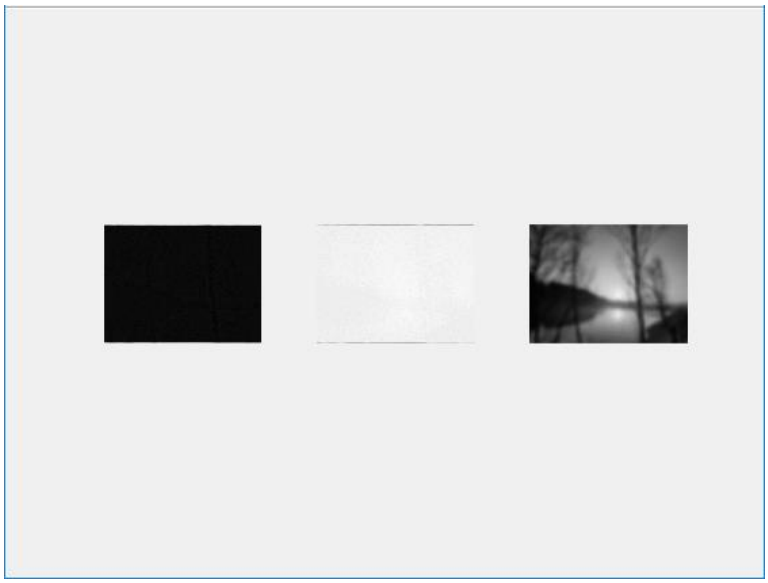
Coşkun Orkun Vural
518181042

Resulting images(laplacian, 3x3 and boost from left to right) are shown below:

Sigma = 1

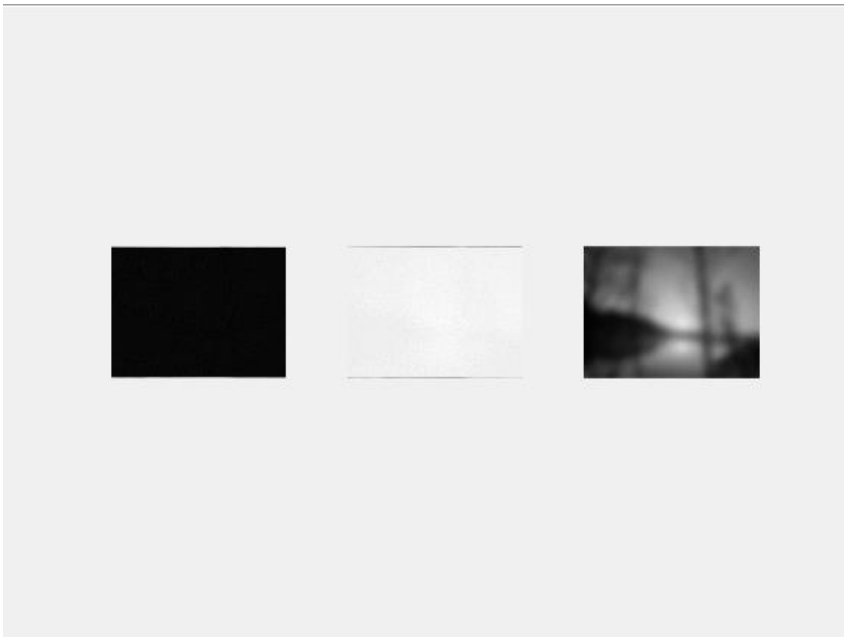


Sigma = 5

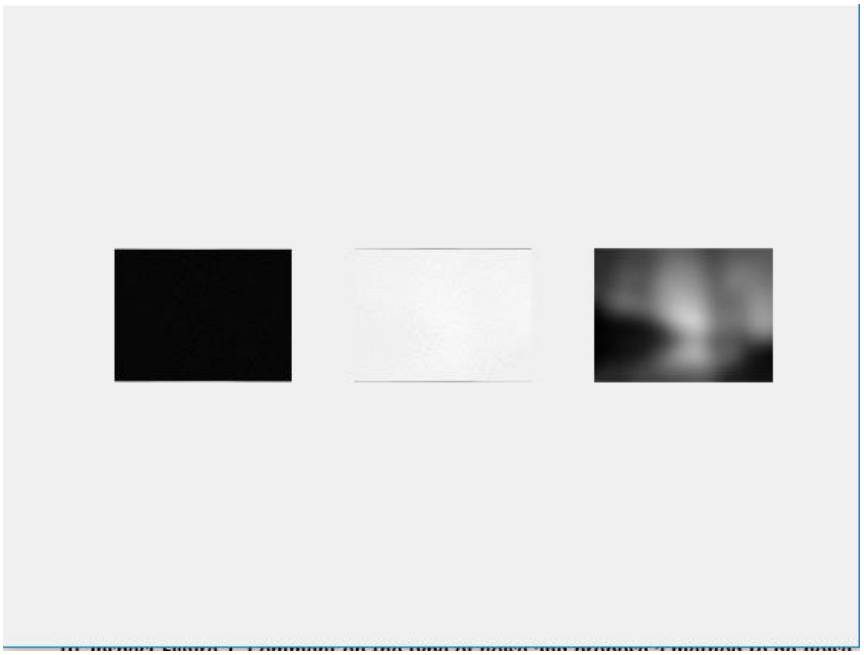


Coşkun Orkun Vural
518181042

Sigma = 10



Sigma = 20



As previously in part 7 gaussian filter was applied to the images high frequency components are lowered. Thus applying a HPF results in black and white images(depending on the value on center of the kernel) for the high sigma values as most of the high frequency components are eliminated. For the high boost case there is not a significant change in the resulting image.

10. **Inspect Figure-1. Comment on the type of noise and propose a method to de-noise the image. Implement your method and present the de-noised image.**

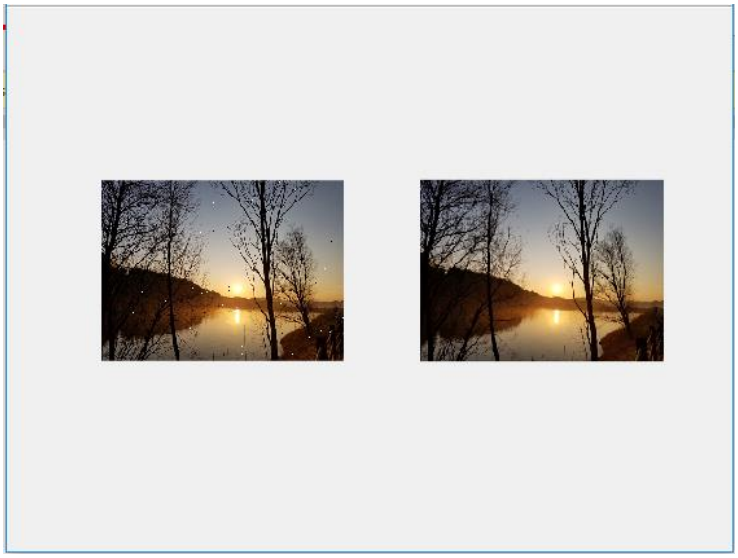
Coşkun Orkun Vural
518181042

The noise on “Figure-1” is salt and pepper noise and median filter is used to de-noise the image. As this noise causes a pixelwise error in the image resulting a pixel to get 1 or 0 value, checking a kernel and taking the middle value is useful for getting rid of salt and pepper noise. Two kernel sizes(3x3 and 5x5) are implemented for de-noising the figure and both gave successful results. Code piece used for median filter is shown below:

```
I_saltAndPepper=imread('Figure_1.png');
I_noiseFree=median(I_saltAndPepper,3);
I_noiseFree2=median(I_saltAndPepper,5);
```

Resulting images:

Original Image vs 3x3 Median Filter



Original Image vs 5x5 Median Filter