EE417

POST-LAB #2 REPORT

1. Linear Filtering

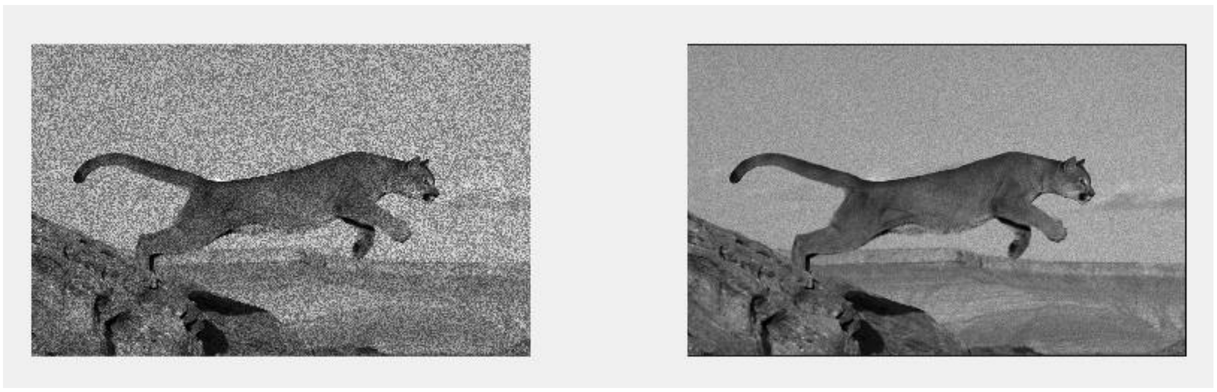
In this section we used a gaussian filtering to smooth an image. That filter is implemented by a kernel function which is obtained from gaussian formula

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

/273

By convolving this kernel gaussian smoothing will be achieved.



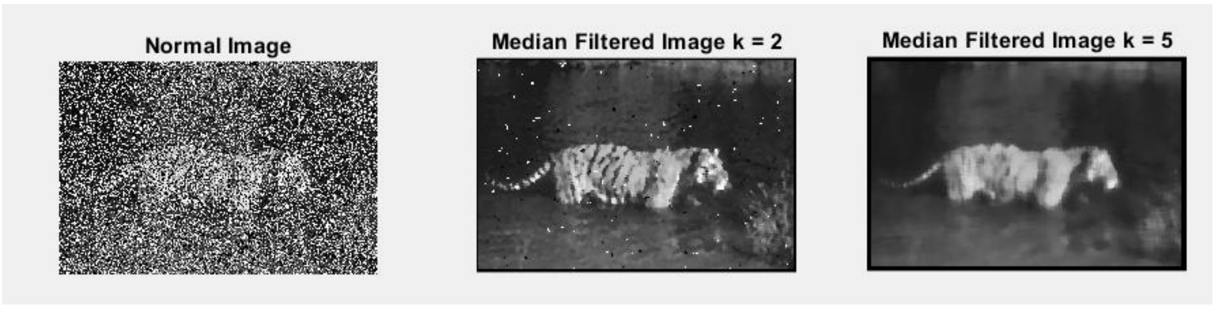
Gaussian Filtered jump.png

After applying of gaussian smoothing algorithm on jump.png we can clearly see that it reduced the noise in the image. It is also known as gaussian blur, as its name implies by reducing noise it also introduces some blur in the image.

```
function G = lab2gaussfilt(img)

    [row,col,ch] = size(img);
    A = zeros(size(img));
    k = 2;

    if(ch==3)
        img = rgb2gray(img);
    end

    Dimg = double(img);

    gaussWindow = 1/273*[1  4  7  4  1;
                         4 16 26 16 4;
                         7 26 41 26 7;
                         4 16 26 16 4;
                         1  4  7  4  1];

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subIm = Dimg(i-k:i+k,j-k:j+k);
            value = sum(sum(subIm.*gaussWindow));
            A(i,j) = value;
        end
    end

    G = uint8(A);

End
```

2. Nonlinear Filtering

In this section we applied a median filter that actually operating on a window, sorting the pixel values and taking the median pixel and assign that median value for the whole window.



If we examine the tiger.png image it is actually very noisy, by applying median filter it can be said that it actually reduced the noise in the image and gets a good quality. By increasing k (window size) we introduces more blur in the image because it assign the same value to a bigger area that reduces details in the image.

```
function G = lab2medfilt(img)

    [row,col,ch] = size(img);
    A = zeros(size(img));
    k = 2;

    if(ch==3)
        img = rgb2gray(img);
    end

    Dimg = double(img);

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subIm = Dimg(i-k:i+k,j-k:j+k);
            value = median( subIm , 'all' );
            A(i,j) = value;
        end
    end

    G = uint8(A);

end
```

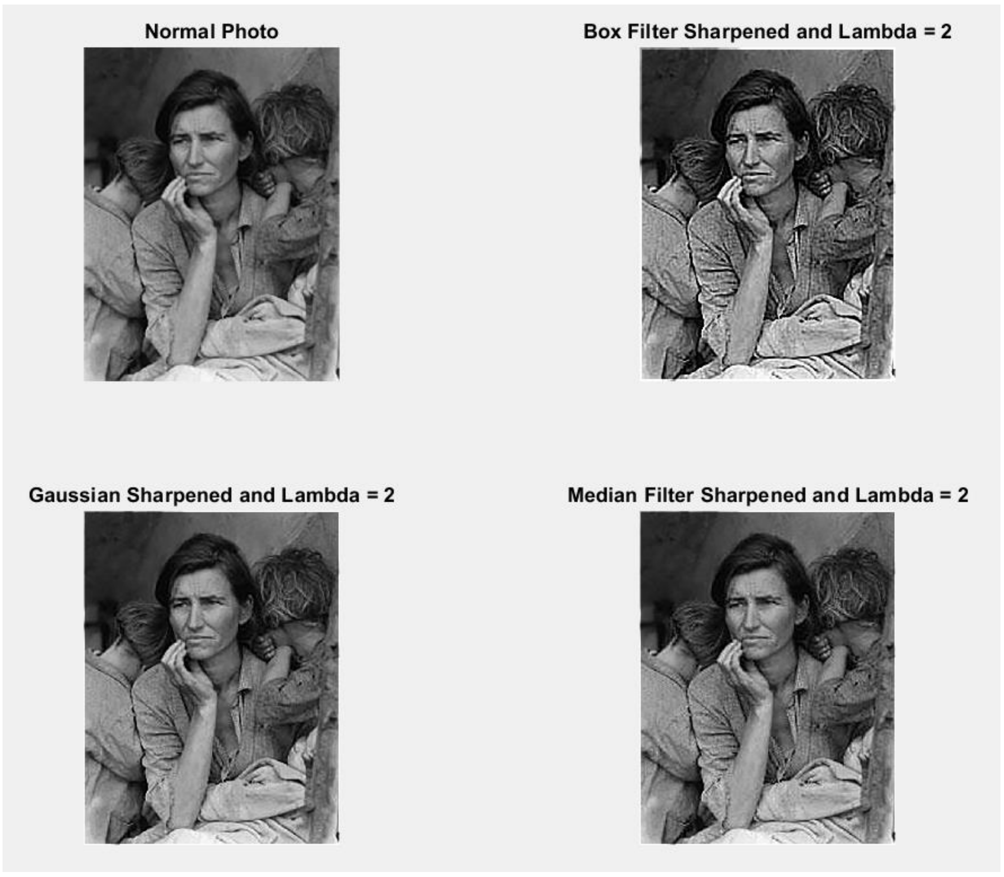
3. Sharpening

In this section we are implementing a sharpening operation on the image. This is done by the following function:

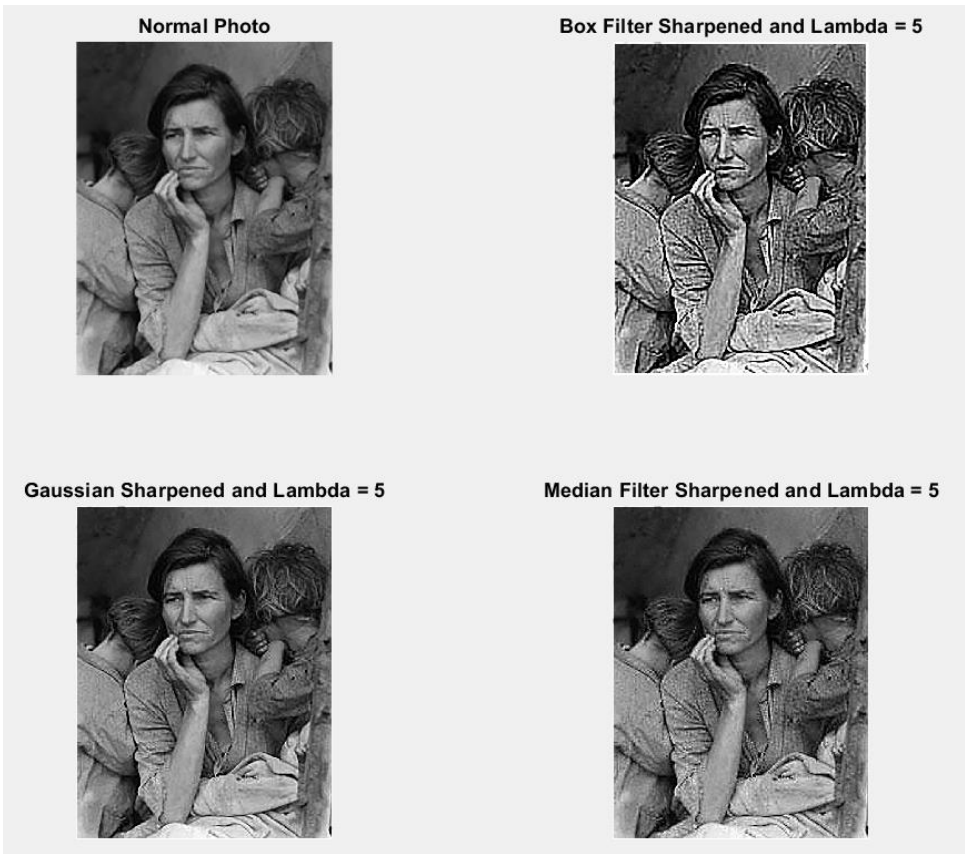
$$J(p) = I(p) + \lambda\Big[I(p) - S(p)\Big]$$

Where I(p) is our image, S(p) is the smoothed version of our image and λ is a scaling factor that controls the influence of the correction signal.

We used three different smoothing methods which are “box filter”, “gaussian filter” and “median filter” and we got the following results:


Normal Photo
Box Filter Sharpened and Lambda = 2
Gaussian Sharpened and Lambda = 2
Median Filter Sharpened and Lambda = 2

By applying sharpening we actually introduced some contrast on the edges on the image as can be seen above. In box filtered version it gives us a view like a sketch version of the image. On the other hand, we can say that gaussian and median filtered sharpened images does not show an influential sharpening effect on the image by λ = 2.

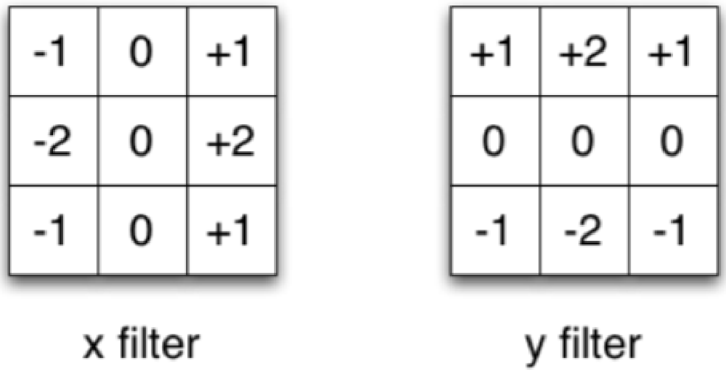

Normal Photo
Box Filter Sharpened and Lambda = 5
Gaussian Sharpened and Lambda = 5
Median Filter Sharpened and Lambda = 5

After changing λ as 5, we actually see the sharpening effect more clearly. In the box filter it can be seen so harshly. By increasing λ, we actually introduces some noise on the image as can be seen above.

```
function G = lab2sharpen(img,M,lambda)

    if (M==1)
        Simg = lab1locbox(img);
    end
    if (M==2)
        Simg = lab2gaussfilt(img);
    end
    if (M==3)
        Simg = lab2medfilt(img);
    end

    [row,col,ch] = size(img);
    A = zeros(size(img));
    k = 2;

    if(ch==3)
        img = rgb2gray(img);
    end

    Simg = double(Simg);
    Dimg = double(img);

    K = Dimg+lambda.*(Dimg-Simg);
    G = uint8(K);

end
```

4. First Derivative

In this section we implemented a 2D sobel filtering that gives us the edges on the image. It can be done by the following kernels:

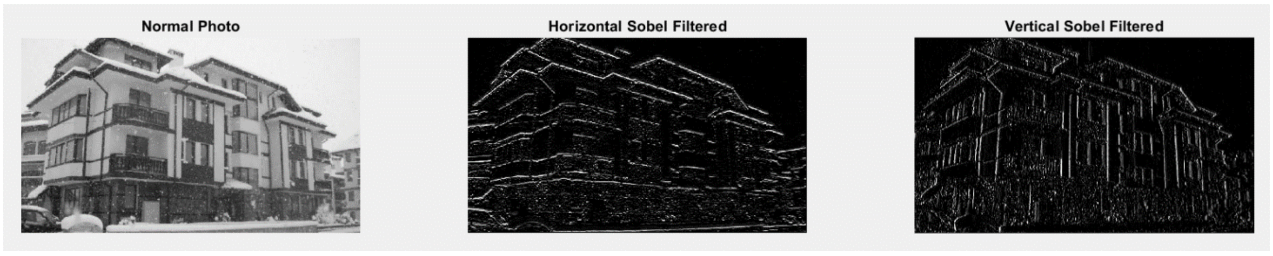


-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter



As can be seen on the above convolving the image with x kernel we get the horizontal edges of the image because it is taking the horizontal derivative of the image. When there is a sudden change in the image pixels of the window it interprets this change as an edge. Best way to detect that change is taking the derivative by nature of derivative (computes rate of change). Convolving the window with this kernel means taking derivative. Similarly, by convolving y kernel, it takes the vertical first derivative of the image that detects the vertical edges on the image as can be seen on the rightmost image.

```
function [XK,YK] = lab2sobelfilt(img)

    [row,col,ch] = size(img);
    X = zeros(size(img));
    Y = zeros(size(img));
    k = 1;

    if(ch==3)
        img = rgb2gray(img);
    end

    Dimg = double(img);

    xkernel = [-1 0 1;
               -2 0 2;
               -1 0 1];

    ykernel = [1 2 1;
               0 0 0;
               -1 -2 -1];
    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subIm = Dimg(i-k:i+k,j-k:j+k);
            value1 = sum(sum(subIm.*xkernel));
            value2 = sum(sum(subIm.*ykernel));
            X(i,j) = value1;
            Y(i,j) = value2;
        end
    end

    XK = uint8(X);
    YK = uint8(Y);

end
```