EE417 - Assignment 2 - Post-Lab Report

Baris Sevilmis
Lab Date: 16/10/2018
Due Date: 22/10/2018

Overview

Second assignment of EE417 focuses on new types of image smoothing and sharpening techniques as well as some of the previous techniques used in Assignment 1. Namely, this assignment concentrates on Gaussian Smoothing, Median Filtering, Image Sharpening and Sobel Filtering. In next sections, logic of these techniques with corresponding MatLab Codes and results will be explained in detail.

Part 1: Gaussian Smoothing

Linear filtering, namely gaussian filtering, is a linear operation, in which image is a local convolution with a gaussian distributed kernel is involved. This technique is mainly used for eliminating specific pixel values that are outliers. In other words, purpose of this filter is to eliminate noise within the image. Figure 1 depicts the 5x5 Gaussian Filter that is used for Gaussian Smoothing. Operation is as follows, as sub-images of main image that have same size with gaussian kernel are consecutively multiplied element-wise and their sum is divided by 273. Original pixel value is replaced with this result.

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

Figure 1: 5x5 Gaussian Filter

In Listing 1, MatLab Code for Gaussian smoothing is provided, as it is a MatLab function.

Listing 1: MatLab Code for Gaussian Smoothing

```
function J = lab2gaussfilt(img)
    [row, col, ch] = size(img);
    k = 2;
    gauss_filter = [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];

    if(ch == 3)
       img = rgb2gray(img);
    end
    img = double(img);
    J = zeros(size(img));

    for i = k + 1: 1: row - k -1
        for j = k + 1: 1 : col - k - 1
           subimg = img(i-k: i+k, j-k:j+k) ;
           J(i,j) = sum(sum(subimg .* gauss_filter)) / 273;
        end
    end
    J = uint8(J);
    subplot(1,2,1);imshow(uint8(img));
    title('Original Image');
    subplot(1,2,2);imshow(J);
    title('Gaussian Filtered Image');
end
```

Lastly, Figure 2 shows results of the given image both before and after Gaussian Smoothing, such that image is smoothed out.

Figure 2: Original and Gaussian Smoothed Image

Part 2: Median Filtering

Median Filtering is a distinct operation compared to the previous operations we achieved. Specifically, it is a non-linear operation that involves sorting and picking the median element. Therefore, given image pixel values need to be reshaped into a 1-D array for sorting and median element selection. Lastly, resulting median is replaced with the original pixel value. In Listing 2, required MatLab code for the operation is shown.

Listing 2: MatLab Code for Median Filtering

```
function J = lab2medfilt(img, k)
    [row, col, ch] = size(img);
    if(ch == 3)
       img = rgb2gray(img);
    end
    img = double(img);
    J = zeros(size(img));

    for i = k + 1: 1: row - k -1
         for j = k + 1: 1 : col - k - 1

             subimg = img(i-k: i+k, j-k:j+k) ;
             Card = ((2*k)+1) ^ 2;
             Data = reshape(subimg, [1 Card]);
             Data = sort(Data);
             if mod(Card, 2) == 0
                ind = Card / 2;
                med = (Data(ind) + Data(ind+1))/2;
             else
                ind = (Card + 1) / 2;
                med = Data(ind);
             end
             J(i,j) = med;
         end
    end
    J = uint8(J);
    subplot(1,3,1);imshow(uint8(img));
    title('Original Image');
    subplot(1,3,3);imshow(J);
    title('Median Filtered Image');

end
```

As we use (2k+1)x(2k+1) rule for filters, k parameter has to be given beforehand in terms of preventing hardcoding issues.Figure 3 depicts k=1, namely 3x3 median filtering, result for Median Filtering. For comparison purposes, Gaussian Filter is also used on the same image.

Figure 3: Median Filtering for k=1

It is clear that Median Filtering with k=1 eliminates noise much better than Gaussian Filter. However, Median Filter with higher kernel size starts blurring out the resulting image. Figure 4 and 5 focuses on higher kernel sizes, for this reason clearly shows the blurring effect. Additionally, corners and 4 distinct edges of the image start turning black because of same reason.

Figure 4: Median Filtering for k=3

Figure 5: Median Filtering for k=7

Part 3: Sharpening

Sharpening is an operation, that enhances images. In other words, it increases contrast of the given image along the existing edges. Important part for this operation is, unlike the previous operations, noise addition within homogeneous regions are minimized. This operation can be done through the given formula below:

$$J(P) = I(P) + \lambda * (I(P) - S(P))$$

$J(P)$ is the resulting image out of the sharpening operation, where as $I(P)$ stands for the original image. On the other hand, $S(P)$ refers to the smoothed image. It means that, formula is based on addition of original image and difference of original and smoothed image. However, that is not all for the formula. λ refers to the sharpening parameter and as it increases brightness and contrast of the image edges. Namely, it

controls the influence of the correction signal and therefore, $\lambda > 0$ must always hold. For the smoothing, any smoothing filter can be used. For the sake of experimentation, box filter, gaussian filter and median filter were used. Code for gaussian filter and median filter can be seen respectively in Listings 1 and 2. For the box filter, Listing 3 is used as demonstration.Lastly, MatLab code of sharpening itself is depicted in Listing 4.

Listing 3: MatLab Code for Box Filtering

```
function J = lab1locbox(img, k)
    [row, col, ch] = size(img);
    if(ch == 3)
       img = rgb2gray(img);
    end
    J = zeros(size(img));
    img = double(img);

    for i = k + 1: 1: row - k -1
        for j = k + 1: 1 : col - k - 1
            subimg = img(i-k: i+k, j-k:j+k) ;
            mu = mean(subimg(:));
            J(i, j) = mu;
        end
    end
    J = uint8(J);

    subplot(2,1,1);imshow(uint8(img));
    title('Original Image');
    subplot(2,1,2);imshow(J);
    title(['Box Filter Image, k = ', num2str(k)]);
end
```

Listing 4: MatLab Code for Box Filtering

```
function J = lab2sharpen(img, lambda, M, k)
    [row, col, ch] = size(img);
    if(ch == 3)
       img = rgb2gray(img);
    end
    img = double(img);
    J = zeros(size(img));

    %Box filter
    if M == 1
        smoothed_img = lab1locbox(img, k);
        smoothed_img = double(smoothed_img);
        J = img + (lambda*(img - smoothed_img));
    %Gaussian Filter
    elseif M == 2
        smoothed_img = lab2gaussfilt(img);
        smoothed_img = double(smoothed_img);
        J = img + (lambda*(img - smoothed_img));
    %Median Filter
    elseif M == 3
        smoothed_img = lab2medfilt(img, k);
        smoothed_img = double(smoothed_img);
        J = img + (lambda*(img - smoothed_img));
    end
    J = uint8(J);
end
```

In addition, in Figure 6, 7 and 8 respective results for sharpening with different lambda and different smoothing operations can be obtained. For testing purpose and because of gaussian filter size, filters of size 5 x 5 will be used as k = 2. Figure 6 focuses on box filter, with different lambda values, as Figure 7 focuses on gaussian filter with same lambda values of box filter. Lastly, Figure 8 depicts the median filtering results with same lambda values as used previously.

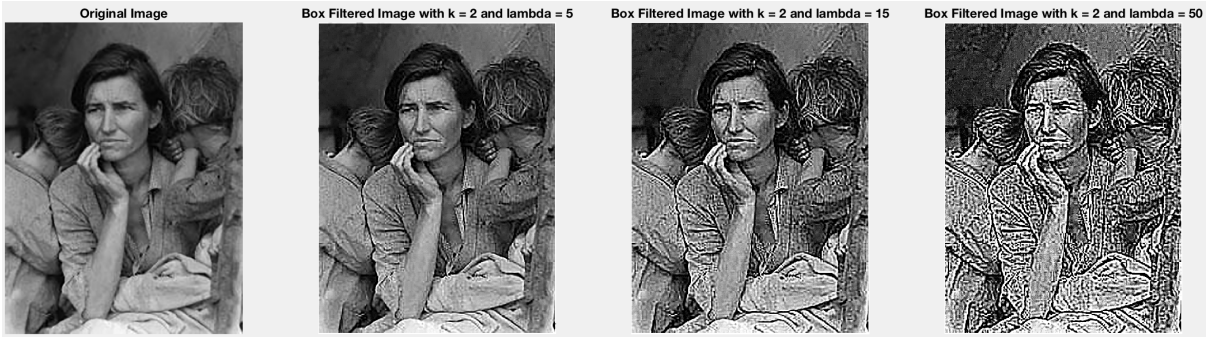

Figure 6: Sharpening with Box Filter

Figure 7: Sharpening with Gaussian Filter

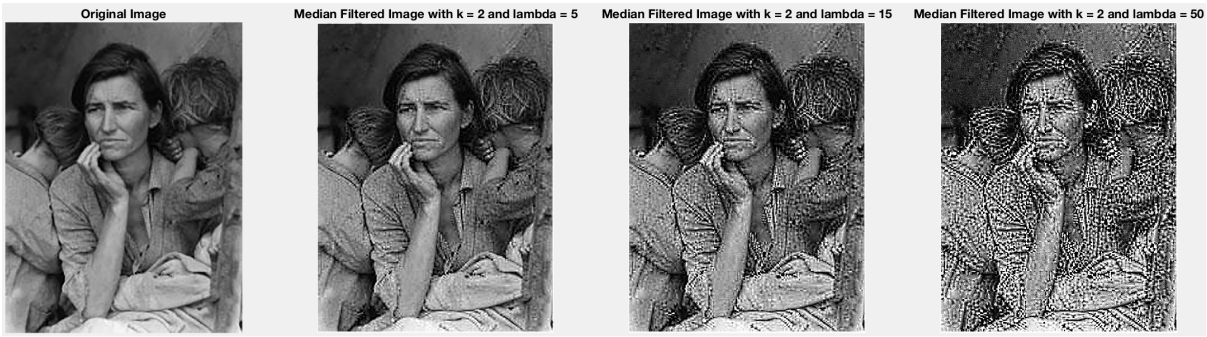

Figure 8: Sharpening with Median Filter

It is definite that contrast increases as the λ increases. In case of median filter, $Contrast/\lambda$ ratio increases slightly fastest, compared to box filter and gaussian filter. Although they have slightly different results, changes on the original image can be seen clearly.

Part 4: Sobel Filtering

Sobel Filtering is a 2D operation, in which through taking derivative of images edges are detected. There are two different filters/kernels used for edge detection in sobel filtering. Figure 9(a) depicts the filter that is used for vertical edge detection by taking horizontal derivation, on the other hand Figure 9(b) demonstrates horizontal edge detector by taking vertical derivation.

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

(a) Vertical Edge Detector

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

(b) Horizontal Edge Detector

Figure 9: Sobel Filters

Listing 5 demonstrates the MatLab code used for Sobel filtering.

Listing 5: MatLab Code for Box Filtering

```
function [J_hor, J_ver] = lab2sobelfilt(img)
    [row, col, ch] = size(img);
    k = 1;
    x_filter = [-1 0 1; -2 0 2; -1 0 1];
    y_filter = [1 2 1; 0 0 0; -1 -2 -1];

    if(ch == 3)
       img = rgb2gray(img);
    end
    img = double(img);
    J_hor = zeros(size(img));
    J_ver = zeros(size(img));

    for i = k + 1: 1: row - k -1
        for j = k + 1: 1 : col - k - 1
            subimg = img(i-k: i+k, j-k:j+k) ;
            J_ver(i,j) =  sum(sum(subimg .* x_filter));
            J_hor(i,j) =  sum(sum(subimg .* y_filter));
        end
    end
    J_hor = uint8(J_hor);
    J_ver = uint8(J_ver);
end
```

Since, Sobel filters have 3x3 fixed size and there are no other parameters to change, there will be only one result that is actually more than enough. Result of sobel filtering is depicted in Figure 10.

Figure 10: Sobel Edge Detection