LAB2 REPORT

Image Processing and Application - 9804

JEWEL MAHMUD NIMUL SHAMIM - 201890846

Procedure:

1. Image downloaded (img1.png and img2.png) from Brightspace under Lab2 and save them in C:\Users\Wohhie\Dropbox\University\Semester 6\Labs\lab2\ working directory.

2. function that will perform the spatial filtering operations below.
   a. Kernel 1 = (1/9)*ones(3)
   b. Kernel 2 = (1/49)*ones(7)
   c. Kernel 3 = fspecial(`average',[7,7]);
   d. Kernel 4 = fspecial(`gaussian',[3,3],0.5);
   e. Kernel 5 = fspecial(`gaussian',[7,7],1.2);

MATLAB Code

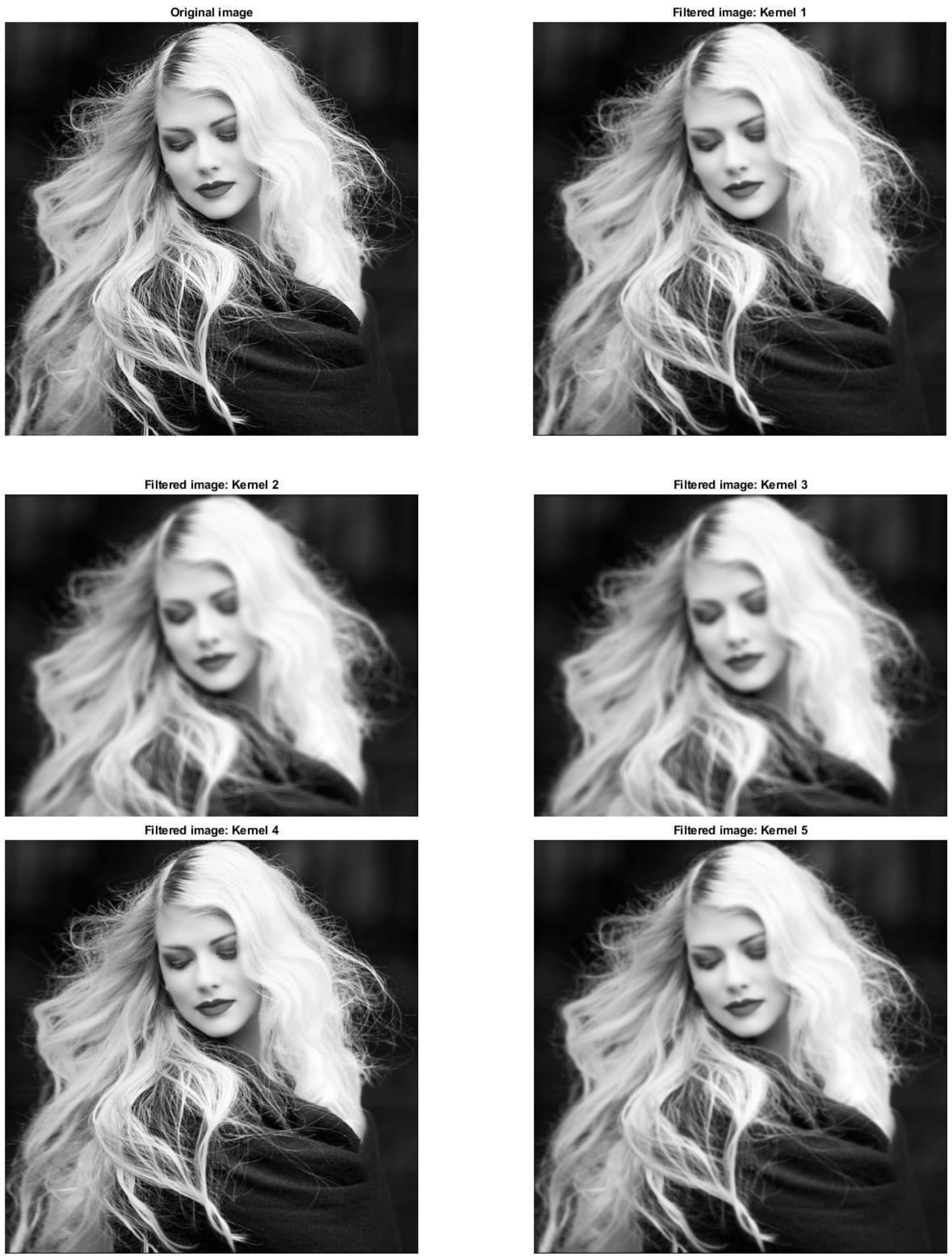
% It takes Kernel and image as input and produce the filter output

```
function filter_image = filter_manual(KernelNo, image)
    im = imread(image);
    % RGB to gray
    grayscale_image = rgb2gray(im);
    grayscale_image = double(grayscale_image);
    subplot(1,2,1);imshow(grayscale_image,[]);
    title('Original image:')

    % Read Kernels
    n = KernelNo;
    switch n
        case 1
        Kernel = (1/9)*ones(3);
        case 2
        Kernel = (1/49)*ones(7);
        case 3
        Kernel = fspecial('average',[7,7]);
        case 4
        Kernel = fspecial('gaussian',[3,3],0.5);
        case 5
        Kernel = fspecial('gaussian',[7,7],1.2);
    end

    filter_image = convolution(grayscale_image, Kernel);
    subplot(1,2,2);
    imshow(filter_image,[]);
    title('Filtered image:')

end
```

To test the program we can run the program below: (for the 1st kernel):

```
clc;
clear all;
close all;
x = filter_manual(1, 'img1.png');
```

Output:



3. function that will perform median filters to image img1.png.
   a. Median Filter with kernel size 3 x 3.
   b. Median filter with kernel size 5 x 5

MATLAB Code

% It takes Kernel size and image as input and produce the filter output

```
function [filtered_image] = medianfilter(image,size_of_filter)
    green  = image(:,:,2);
    [rows,cols] = size(green);
    pad = size_of_filter-1;
    nimage = zeros(rows + pad,cols + pad);
    nimage(pad/2+1:rows+pad/2, pad/2+1:cols+pad/2) = green;
    filtered_image = zeros(rows,cols);
    for x = pad/2 + 1 : cols + pad/2
        for y = pad/2 + 1 : rows + pad/2
            if nimage(y,x) == 0 || nimage(y,x) == 255
                win = nimage(y-pad/2:y+pad/2, x-pad/2:x+pad/2);
        % find median
                filtered_image(y-pad/2,x-pad/2) = median(win(:));
            else
                filtered_image(y-pad/2,x-pad/2) = nimage(y,x);
            end
              % get mxm window
            win = nimage(y-pad/2:y+pad/2, x-pad/2:x+pad/2);
          % find median
            filtered_image(y-pad/2,x-pad/2) = median(win(:));
        end
    end

    filtered_image=uint8(filtered_image);
    imshow(filtered_image);

end
```

To test the program we can run the program below:

```
img = imread('img1.png');
subplot(1,2,1);
x = medianfilter(img,3);
title('3x3 Median Filter Output');
subplot(1,2,2);
y = medianfilter(img,5);
title('5x5 Median Filter Output');
```

Output:

4. The following kernels implement linear sharpening filters which are used to enhance the edges in images. Use the function developed in question 2 and apply the filters to image img2.png.

$$a)\ Kernel\_6 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$b)\ Kernel\_7 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

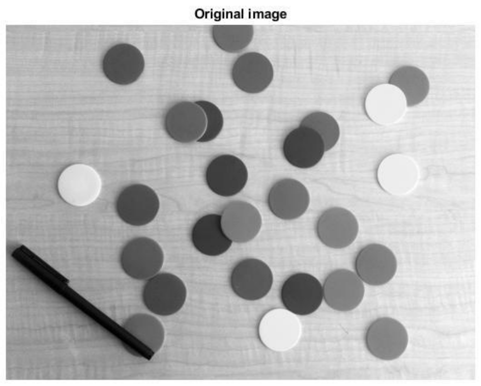
$$c)\ Kernel\_8 = fspecial('log', 3);$$

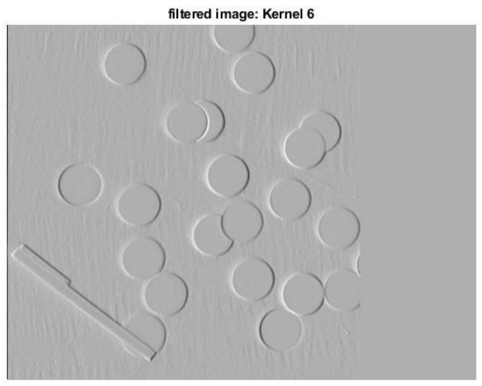
```
function filter_image = filter_manual1(KernelNo, image)
    im = imread(image);
    grayscale_image = rgb2gray(im);
    grayscale_image = double(grayscale_image);
    subplot(1,2,1);imshow(grayscale_image,[]);
    title('Original image')
    % Read Kernels
    n = KernelNo;
    switch n
        case 6
        Kernel = [-1 0 1;-2 0 2;-1 0 1];
        case 7
        Kernel = [-1 -2 -1;0 0 0;1 2 1];
        case 8
        Kernel = fspecial('log',3);
    end

        filter_image = convolution(grayscale_image,Kernel);
        subplot(1,2,2);
        imshow(filter_image,[]);
        title('filtered image: Kernel 7')
end
```

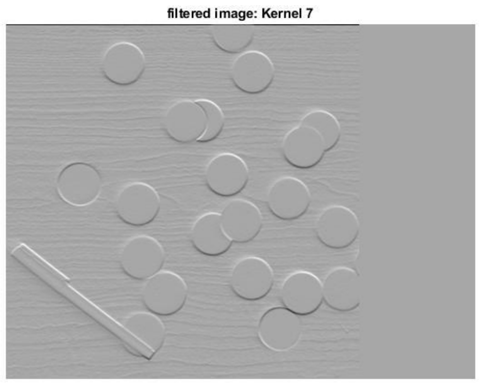
To test the program we can run the program below

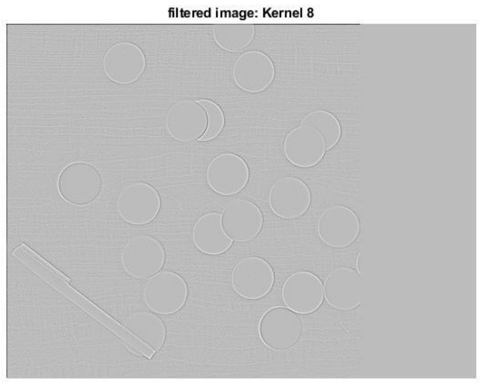
```
clc;
clear all;
close all;
x = filter_manual1(7, 'img2.png');
```

Output:









Thresholding:

1. Image downloaded (img3.png and img4.png) from Brightspace under Lab2 and save them in C:\Users\Wohhie\Dropbox\University\Semester 6\Labs\lab2\ working directory.
2. Implement equations 1 and 2 as a function to calculate the threshold value for any given image.

MATLAB Code

% It takes image as input and output the threshold value

```
function threshold = fthreshold(image)
    I = imread(image);
    img = rgb2gray(I);
    % Initialize histogram
    H = zeros(256);

    % Calculate histogram
    [l, w] = size(I);
    for r = 1:l
        for c = 1:w
            index = I(r, c);
            index = index + 1;
            H(index) = H(index) + 1;
        end
    end
    % Thresholding
    q = zeros(255);
    q(1) = H(1) / (l * w);
    n = zeros(255);
    u = zeros(255);
    P = zeros(255);
    max = 0;
    index = 0;

    sum = 0;
    for j = 1:256
        sum = sum + j * H(j);
    end
    sum = sum / (l * w);
    for i = 1:255
    %     find sigma b^2 for each
    %     pick the maximum value and use the corresponding i
    %     if (sigb^2 > th; th = sigb^2
        P = H(i + 1) / (l * w);
        q(i + 1) = P + q(i);
        n(i + 1) = ((i + 1) * P) / q(i + 1) + q(i) * n(i) / q(i + 1);
        u(i + 1) = (sum - q(i + 1) * n(i + 1)) / (1 - q(i + 1));
        s1 = q(i + 1) * (1 - q(i + 1)) * (n(i + 1) - u(i + 1));
        s = s1 ^ 2;
        if (s > max)
            max = s;
            index = i;
        end
    end
    threshold = index
end
```

To test the program, we can run the program below:

```
clc;
clear all;
close all;
x = fthreshold('img3.png');
```

Output of MATLAB code

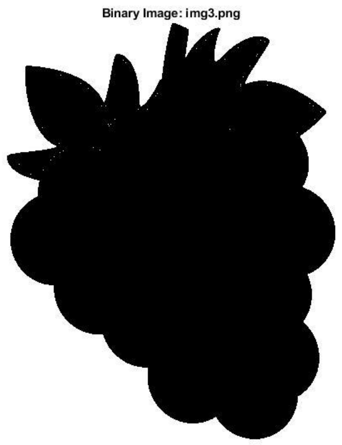
```
   threshold =
      157
```

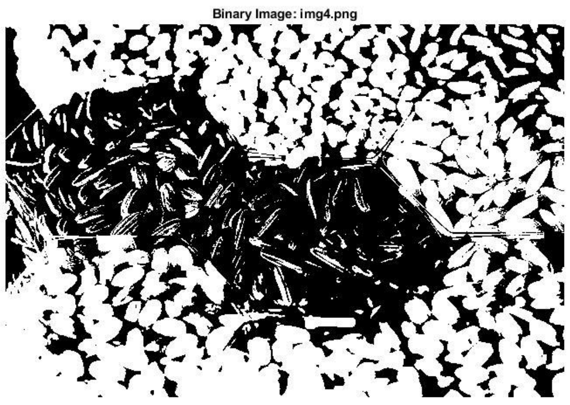
3. Threshold Value of img3.png = **157**
   img4.png = **128**

4. Develop code to convert a grayscale image to a binary image using a given threshold.
   MATLAB Code

```
I = imread('img3.png');
img=rgb2gray(I);
% Initialize histogram
H = zeros(256);

% Calculate histogram
[l, w] = size(I);
for r = 1:l
    for c = 1:w
        index = I(r, c);
        index = index + 1;
        H(index) = H(index) + 1;
    end
end

% Display histogram
figure;
hist = bar(0:255, H, 'histc');

% Thresholding
q = zeros(255);
q(1) = H(1) / (l * w);
n = zeros(255);
u = zeros(255);
P = zeros(255);
max = 0;
index = 0;

sum = 0;
for j = 1:256
    sum = sum + j * H(j);
end
sum = sum / (l * w);

for i = 1:255
%     find sigma b^2 for each
%     pick the maximum value and use the corresponding i
%     if (sigb^2 > th; th = sigb^2
    P = H(i + 1) / (l * w);
    q(i + 1) = P + q(i);
    n(i + 1) = ((i + 1) * P) / q(i + 1) + q(i) * n(i) / q(i + 1);
    u(i + 1) = (sum - q(i + 1) * n(i + 1)) / (1 - q(i + 1));
    s1 = q(i + 1) * (1 - q(i + 1)) * (n(i + 1) - u(i + 1));
    s = s1 ^ 2;
    if (s > max)
        max = s;
        index = i;
    end
end

th = index
I2 = zeros(size(img));
I2 (find(img>=th)) = 1;
I2 (find(img<th)) = 0;
%Z = im2bw( I ,map, th );
figure;
imshow(I2)
title('Binary Image: img3.png');
```

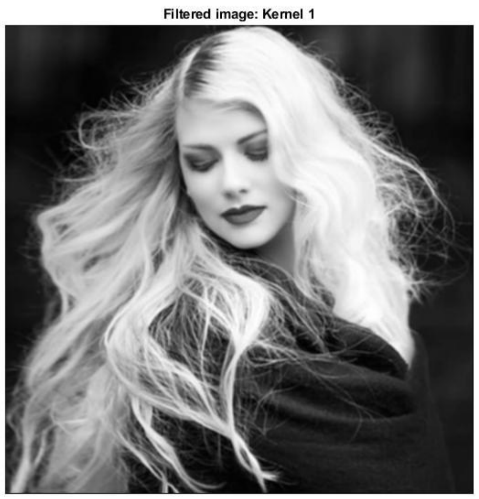
5. binary image for img3.png and img4.png.

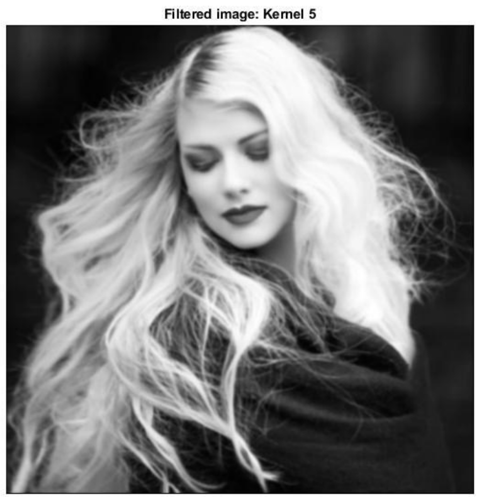




Discussion:

1. **Compare the smoothing effects obtained by Kernel 1 to Kernel 5 and the median filters:**





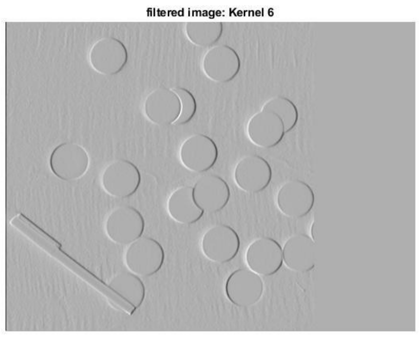
- Kernel 5 filter output is much smoother than Kernel 1 filter.
  Here, Kernel 1 to 3 is mean filtering which smooth the noise by average of pixels and its neighbors. Whereas Kernel 4 to 5 is gaussian filter which done by weighted average of each pixels neighborhood in image.

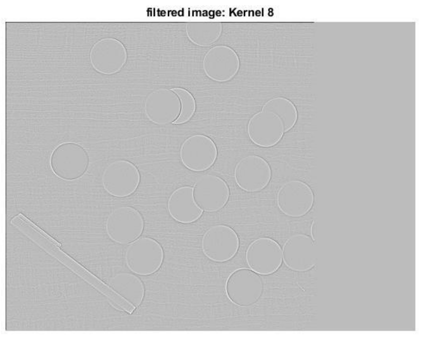
  The median filter: The median filter takes individual pixel of an image and match its neighborhood pixels whether it is matched with its neighborhood pixels or not. So the output produce a better details on edge.

2. **Discuss the effect the kernel size has on the output images in your experiments.**

- The information in the images reduces as the kernel size increases. We can see the image passes through the 5x5 Kernel is blurred, so edge information is diminished. But more smoothing will reduce its image details.

3. Compare the outputs obtained by Kernel 6 to Kernel 8.





- The output of Kernel 6 is having much more edge information as compared to the Kernel 8. Kernel 6 is the vertical mask of Sobel operator which finds rounded vertical edges. Whereas Kernel 7 is horizontal mask which finds flat edges in the input image. And Kernel 8 is a Log operator that calculates the spatial derivative of the image.

4. Discuss the effect noise has on edge detection.

- There may be a lot of issue can happen in detection of edge. As we know it can have a false edge on image and most of the noise in image is filtered by smoothing filter. Higher order of smoothing filter will blur the image. So, we are having a tradeoff.

5. Global thresholding can fail if the lighting conditions in an image are not uniform. Review adaptive thresholding ad explain how it can address uneven lighting conditions.

- In local adaptive thresholding, it selects the individual threshold for each pixel based on the range of intensity values of their local neighborhood, this will help to address lighting Local adaptive thresholding, if it is global thresholding then global intensity histogram doesn't contain distinctive peaks.