SMOOTHING, SHARPENING AND FIRST DERIVATIVE FILTERS

OCTOBER 26, 2021

TA's: Muhammed Zemzemoğlu and Mehmet Emin Mumcuoğlu

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417)

TABLE OF CONTENTS

INTRODUCTION

In this laboratory, smoothing, sharpening and first derivative filters are explored. These are all geared towards ensuring that an image is properly preprocessed before it’s ready for image analysis.

Image smoothing are local operators that are employed to get rid of “outliers” in image intensities. These outliers usually constitute noise because they are values (either too high or too low) in a set of intensities that shouldn’t belong to that set.

Note: These outliers could be as a result of errors in the camera used to capture the image (e.g., due to malfunctioning sensors). In some cases, the outliers are very visible white or black dots in images. In this case, the noise is identified as speckle/impulsive/salt and pepper noise.

Smoothing methods are usually low-pass filters that are used to “filter out” high frequency components of images. While some smoothing methods reduce image contrast significantly, some others have little effect on contrast. Some methods require higher computational cost, especially when sorting or while using processes beyond the basic convolution are involved.

Sharpening, on the other hand, is primarily applied to increase the contrast of a given image by masking unsharp areas of that image with a scaled version of the high frequency residual.

Derivatives are important methods applied in edge detection. From math, we know that the derivative encodes information about the change of a function. This same principle is applicable to images. If we take the first-order derivative of the intensity values of an image, wherever there’s a peak (maxima or minima), we have an edge (i.e., an area of dramatic variation in intensity).

EXPLANATION OF METHODS

Smoothing

Gauss Filter

This is used for gaussian smoothing. The idea behind the gauss filter is that the filter is not just one with weights of 1, as in the box filter. Here, a more logical concept is applied to the filter kernel. The three-sigma rule is applied to give meaningful weights to elements of the window, such that weights decrease as we move away from the central/reference pixel. The gauss filter is obtained using the 2D centered gauss function (with the window reference centered at 0,0).

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

where σ = the standard deviation or the radius of the function

To obtain a gaussian filtered image, the gauss filter is used to scan the image, and, at every instance, the convolution of the image window with the gaussian filter is taken.

$$L(x,y,\sigma) = [I * G_\sigma](x,y)$$

where σ = the standard deviation or the radius of the function, I = original image; the LHS represents the gaussian filtered image.

In MATLAB, the above is performed using the following function that takes an image as the input and returns the gaussian smoothed version of the image:

```
function [output] = lab2gaussfilt(img)

% The row, column, and channels of the image are obtained
%along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
```

```
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operations
I = double(img);

%create the gaussian filter
Gauss_filt = [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;

% The window size for the gaussian filtering operation is
%specified.
k = 2;

% To ensure that we don't end up with an image of a
%different size relative to the original image, we
%initialize our final image with the same number of rows
% and columns as the original one.
Inew = zeros(r, c);

% Use nested for-loops to create a window for scanning the
%image.
for i=(k+1):r-k
    for j=(k+1):c-k
        %the window will go from -k to +k
        wp = I(i-k:i+k, j-k:j+k);
        % Perform convolution using the previously created
%gaussian filter.
        Inew(i,j) = sum(wp(:).*Gauss_filt(:));
    end
end

% Convert the resulting image to unsigned 8-bit image and
%return the result.
output = uint8(Inew);

end
```

In the main script, the following code calls the gauss filter function and applies it on the input image, before displaying the results.

```
%% Gauss Filter
a = imread('jump.png');
% We call the gauss filter function.
res = lab2gaussfilt(a);
%The original image and the gaussian filtered image are
%displayed.
figure
subplot(1, 2, 1)
imshow(a)
title('Original Image');

subplot(1, 2, 2)
imshow(res)
title('Gaussian Filtered Image');
```

Median Filter

This is a nonlinear smoothing operation that first sorts out the intensity values in a window, obtains the median value, and replaces the intensity at window's reference with the median value.

- The median is obtained using: $\frac{2n+1}{2}$.
- This is because we make use of windows with odd number rows and columns. So, the (n+1)th intensity in the sorted row is the median.

In MATLAB, the following function was created for median filtering operation. The function accepts an image and a constant k (for the window size) and returns the median filtered image. The for-loop doesn't introduce convolution; rather, it introduces a sorting operation because median involves sorting not convolution.

```
function [output] = lab2medfilt(img, k)

% The row, column, and channels of the image are obtained
%along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;
```

```
% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operations
I = double(img);

% To ensure that we don't end up with an image of a
%different size relative to the original image, we
%initialize our final image with the same number of rows
% and columns as the original one.
Inew = zeros(r, c);

% Use nested for-loops to create a window for scanning the
%image.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        %the window will go from -k to +k
        wp = I(i-k:i+k, j-k:j+k);
        % Obtain the parameters (number of rows and
%columns of the window and its cardinality).
        [ro, co] = size(wp);
        w_card = ro*co;
        % We'll need to sort the intensities in ascending
%or descending order, so it's imperative that we create a
%row vector to make the sorting process straightforward.
        d = reshape(wp, [1, w_card]);
        sorted = sort(d);
        % The following is used to obtain the index of the
%median intensity.
        ind = (w_card + 1)/2;
        med = sorted(ind);
        % Replace the intensity at the window's reference
%point with the above median.
        Inew(i,j) = med;
    end
end
```

```
% Convert the resulting image to unsigned 8-bit image and
%return the result.
output = uint8(Inew);
end
```

The following code calls the median filter function and applies it on the input image, before displaying the results.

```
%% Median Filter

p = imread('tiger.png');
r = 5;
%The following syntax calls the median filter and gaussian
%filter functions.
re = lab2medfilt(p, r);
rem = lab2gaussfilt(p);
%The original image, the gaussian filtered image, and the
% median filtered image are displayed.
figure
subplot(1, 3, 1)
imshow(p)
title('Original Image')

subplot(1, 3, 2)
imshow(rem)
title('Gaussian Filtered Image');

subplot(1, 3, 3)
imshow(re)
title('Median Filtered Image')
```

Sharpening

The method of sharpening, in succinct terms, entails 'adding' a residual image to an original one to increase the contrast along the edges while keeping the noise in the immediate regions in check. The residual image is obtained by 'subtracting' a low pass filtered version of an image from the original one. The low pass filter (LPF) could be any smoothing filter. A sharpened image is obtained by the following:

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

Where $\lambda > 0$ is a scaling factor which determines the influence of the correction signal; S(p) is the smoothed image; I(p) is the original image.

In MATLAB, the following function was written for the sharpening process. The function accepts an image, a constant λ, and an integer M (to decide on which smoothing operator to use) and returns a sharpened image.

```
function [output] = lab2sharpen(img, li, m)

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operations
I = double(img);

% The window size for the smoothing operation is
specified.
k = 2;

% The following conditional statements are used to specify
the smoothing
% method to apply.
if m == 1
    % This applies the Box Filter as the smoothing method.
    im = lab1locbox(img, k);
elseif m == 2
        % This applies the Gaussian Filter as the
smoothing method.
        im = lab2gaussfilt(img);
```

```
elseif m == 3
        % This applies the Median Filter as the smoothing
method.
        im = lab2medfilt(img, k);
end

% Convert the resulting image from the above conditional
statements to a
% double for mathematical operations.
ima = double(im);

% Sharpening doesn't involve any convolution, hence, no
for-loop is used.
Inew = I + li*(I - ima);

% Convert the resulting image to unsigned 8-bit image and
return the
% result.
output = uint8(Inew);
end
```

The following code calls the local mean filter function and applies it on the input image, before displaying the results.

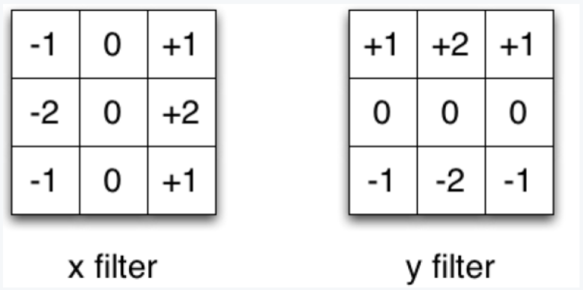
```
%% Sharpening
% Read the image to be preprocessed
d = imread('mother.png');
% This is lambda which controls the influence of the
correction signal.
l = 10;
% This integer is used to select the smoothing operation
to be applied
m = 1;

% This syntax calls the Sharpening function
r = lab2sharpen(d, l, m);

%The original image and the sharpened image are displayed.
figure
subplot(1, 2, 1)
imshow(d)
```

```
title('Original Image')

subplot(1, 2, 2)
imshow(r)
title('Sharpened Image')
```

First Derivative

Sobel Filter

The Sobel filter is a first derivative operator that uses special filter kernels there perform simultaneous derivative and smoothing on an image. It does this by the process of convolution. The kernel filters are:



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

x filter

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

y filter

In MATLAB, the following function was written. The function accepts an image returns a Sobel filtered image.

```
function [output1, output2] = lab2sobelfilt(img)

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operation
I = double(img);
```

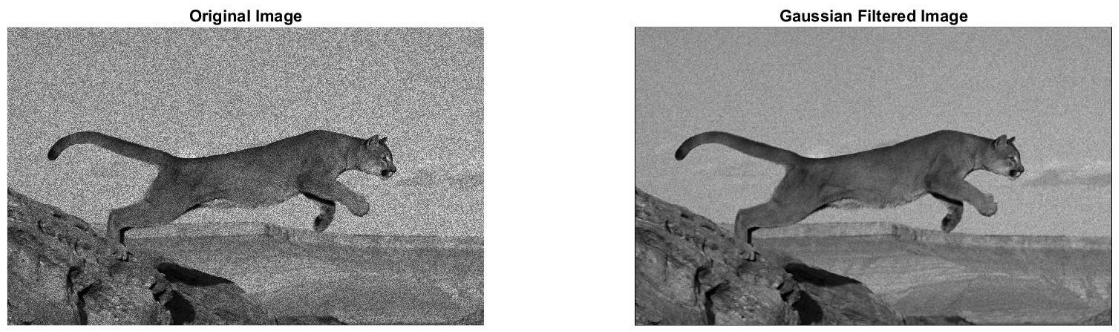
```
% Create the sobel filters; the x-filter will be used to
create the
% vertically filtered image, while the y-filter will be
used to create the
% horizontally filtered image.
x_filt = [-1 0 1; -2 0 2; -1 0 1];
y_filt = [1 2 1; 0 0 0; -1 -2 -1];

% The sobel filter has a size (2k+1)x(2k+1), where k = 1;
k = 1;

% To ensure that we don't end up with images of different
sizes relative
% to the original images, we initialize our final images
with the same number of rows
% and columns as the original ones.
Ihor = zeros(r, c);
Iver = zeros(r, c);

% Use nested for-loops to create a window for scanning the
image.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        %the window will go from -k to +k
        wp = I(i-k:i+k, j-k:j+k);
        % Perform convolution using the previously created
masks.
        Iver(i,j) = sum(wp(:).*x_filt(:));
        Ihor(i,j) = sum(wp(:).*y_filt(:));
    end
end

% Convert the resulting images to unsigned 8-bit images
and return the
% results.
output1 = uint8(Ihor);
output2 = uint8(Iver);

end
```

The following code calls the Sobel filter function and applies it on the input image, before displaying the results.

```
%% Sobel Filter
% Read the image to be preprocessed
f = imread('house.png');

This syntax calls the local Sobel filter function
[hor, ver] = lab2sobelfilt(f);

%The original image and the Sobel min filtered images are
%displayed.
figure
subplot(2, 2, [1, 2])
imshow(f)
title('Original Image')

subplot(2, 2, 3)
imshow(ver)
title('Sobel Vertical Filtered image')

subplot(2, 2, 4)
imshow(hor)
title('Sobel Horizontal Filtered Image')
```

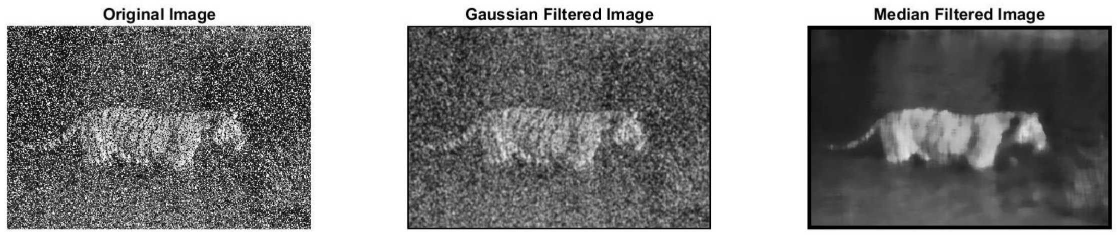
RESULTS

Gauss Filter



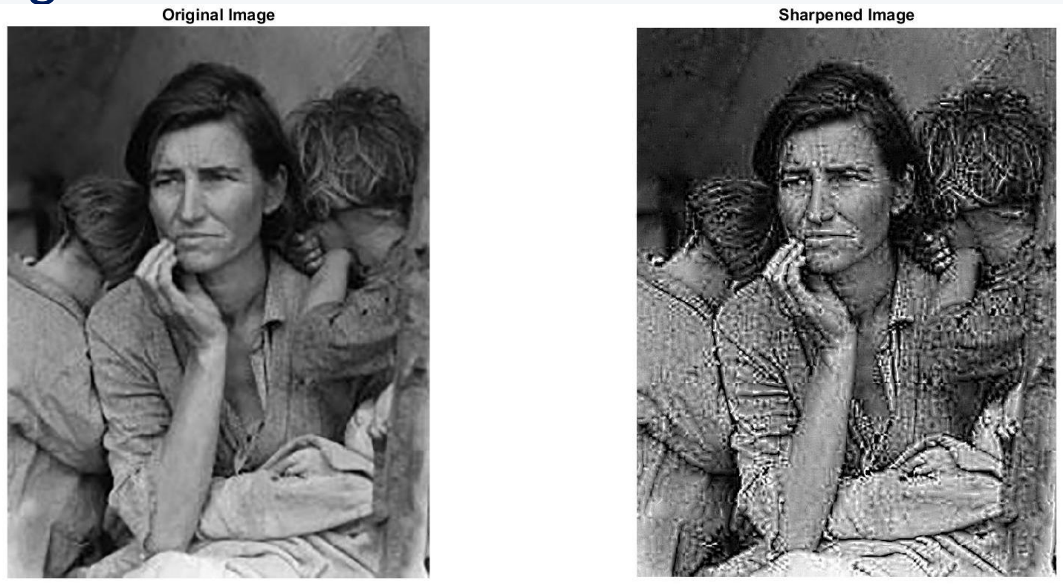
The gaussian filter does a good job of cleaning up the above image. Since it is based on convolution, it is suitable for real-time applications.

Median Filter



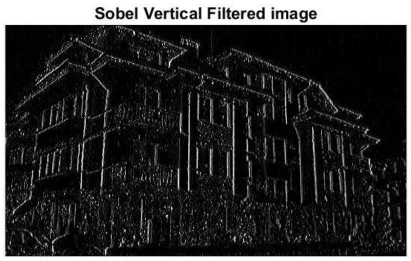
Notice: The median filter performs significantly better than the gaussian filter. It was able to eliminate the salt & pepper noise while smoothening the image. Therefore, we can conclude that gaussian filters are good for smoothing, but not necessarily for removing salt & pepper noise.

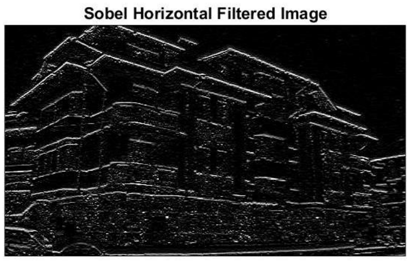
Sharpening



We immediately recognize that the presence of brighter pixels which is due to the masking of the unsharp parts with a scaled correction signal (or residual image).
Sharpening could be applied in cases where we want to ensure that the contrast in the image is easily noticeable.
Since sharpening entails that an image be smoothened first, we can use it to eliminate salt & pepper noise if our smoothing method is the median filtering.

Sobel Filter





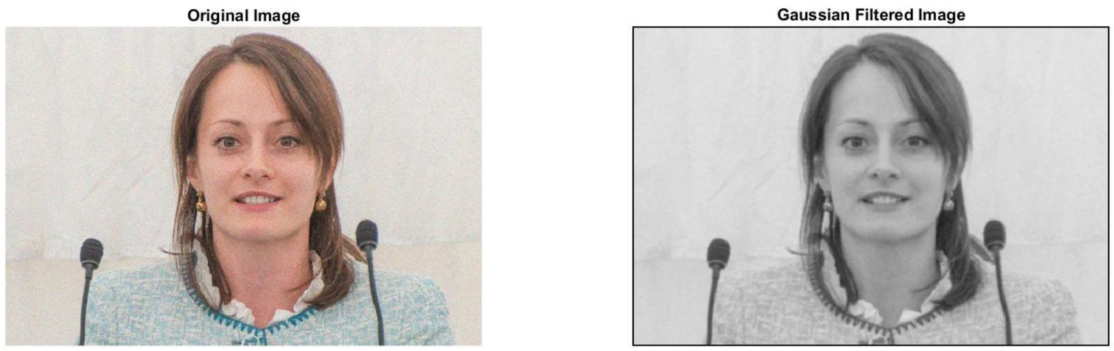
- The x-filter was used to obtain the Sobel vertical filtered image.
- The y-filter was used to obtain the Sobel horizontal filtered image.

Sobel filter is good for edge detection. It wouldn’t be a good method for smoothing since the scale applied in the filter are not uniform.

MULTIPLE RESULTS

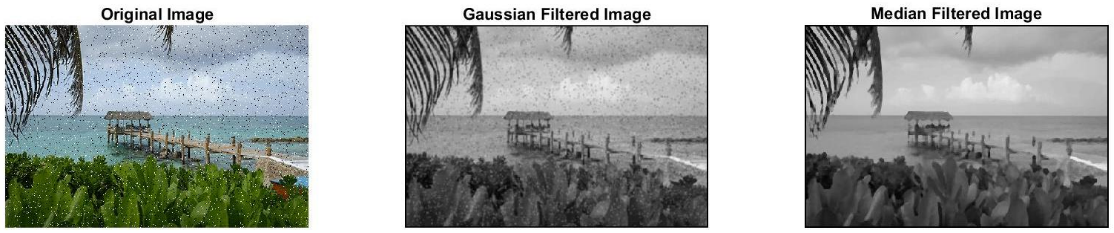
Gauss Filter

Using a color image, let’s see the outcome:



Median Filter

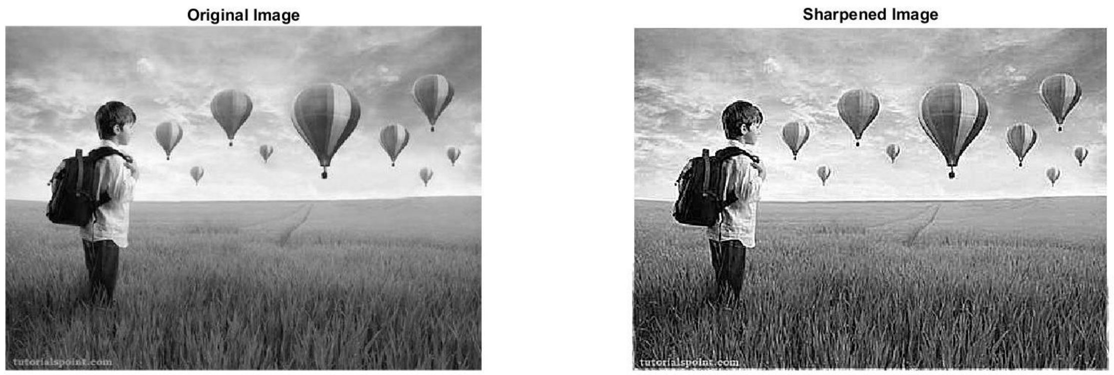
Performing median filtering and gaussian filtering on an image with impulsive noise and comparing their performance:



The gaussian filtered image is smooth, but it still contains speckle noise; the median filtered image, however, doesn’t have the salt & pepper noise. Also, the edges in the median filtered image are not so crisp.
Note: The same window size was used in both cases.

Sharpening

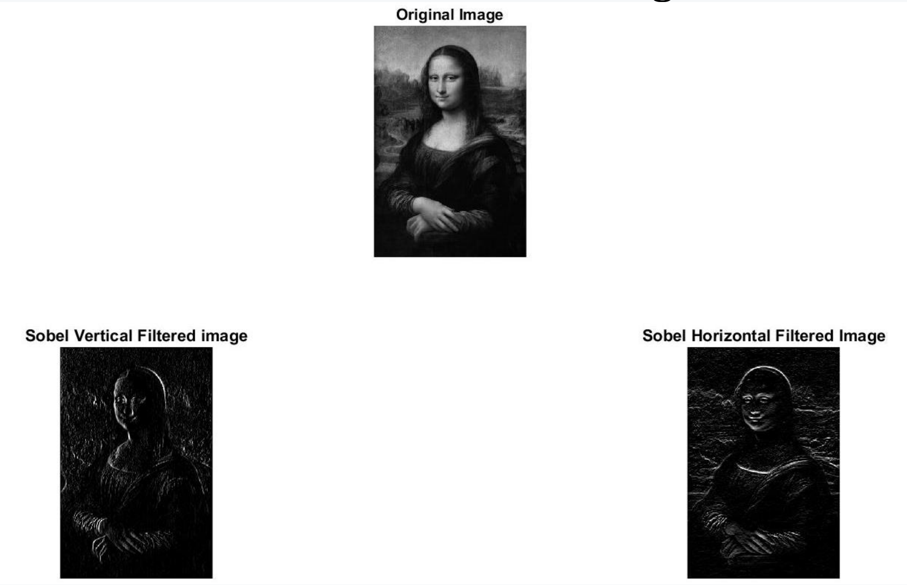
Sharpening a random gray image with lamda = 1.5, and using the box (low pass) filter, we obtain:



Notice how the grass is now more evident due to the increase in contrast.

Sobel Filter

Applying the Sobel filter on some random images:


Original Image
Sobel Vertical Filtered image
Sobel Horizontal Filtered Image

QUESTIONS

Sigma Filter

The Sigma Filter is a non-linear filter based on averaging; however, it does not involve classical averaging as in the case of the box filter. We understand a classical "straight" averaging filter like the box filter smoothens images at the expense of blurring edges and smearing subtle details.

The Sigma Filter computes an average based a predefined intensity range enforced by a user input σ. The equation for the Sigma Filter is as follows:

In MATLAB, the following function was written for the Sigma Filter. The function accepts three inputs – 1 image and two numbers for σ and k (to determine the window size).

```
function [output] = lab2sigmafilt1(img, sig, k)

% The row, column, and channels of the image are obtained
%along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operation
I = double(img);

% Initialize the variables to be used for the sigma
%equation.
denom = 0;
kipp = 0;
mult = 0;
```

```
ad = 0;

% To ensure that we don't end up with an image of a
%different size relative to the original image, we
%initialize our final image with the same number of rows
% and columns as the original one.
Inew = zeros(r, c);

% Use nested for-loops to create a window for scanning the
%image.
for i = (k+1):1:(r-k)
    for j = (k+1):1:(c-k)
        %the window will go from -k to +k
        wp = I(i-k:i+k, j-k:j+k);
        % Obtain the intensity of the central pixel, and
%create the upper and lower limits for the sigma function.
        w = wp((1+k), (1+k));
        low = w - sig;
        high = w + sig;

        % Use a for-loop to scan the window for pixels
%values within the limits.
        for range = low :1: high
            % Use a find function to locate pixels of the
%same intensity.
            kip = find(wp==range);
            % Use the size function to count the pixels
found above
            % (this serves as a histogram function).
            kipp = size(kip);
            % Obtain the u.H(u)
            mult = kipp(1)*range;
            % Take the summation of the above
            ad = ad + mult;
            % This is for the denominator of the sigma
%equation, S.
            denom = denom + kipp(1);
        end
     Inew(i,j) = ad/denom;

    % It's important to return the sigma equation
%parameters to zero
```

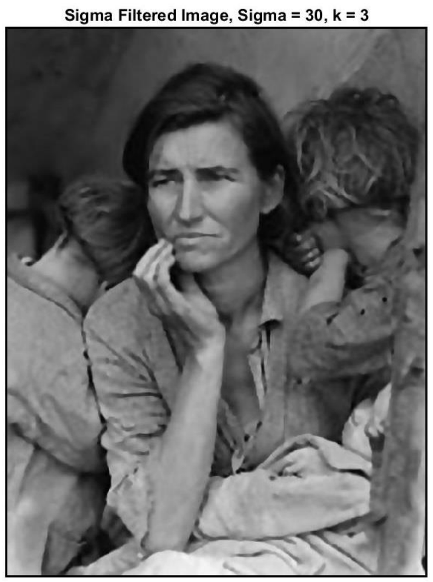
```
    % before the next iteration. This helps us avoid
%producing a really awkward image.
    denom = 0; kip = 0; kipp = 0; mult = 0; ad = 0;
    end
end

% Convert the resulting image to unsigned 8-bit image and
%return the result.
output = uint8(Inew);
end
```

The following code calls the Sigma filter function and applies it on the input image, before displaying the results.

```
% Read the image to be preprocessed,and initialize the
%sigma and k valuees.
rp = imread('mother.png');
k_ = 3;
sigm = 30;

% Call the Sigma Filtering function.
sigfil = lab2sigmafilt1(rp, sigm, k_);

%The original image and the sigma filtered image are
%displayed.
figure
subplot(1, 2, 1)
imshow(rp)
title('Original Image')

subplot(1, 2, 2)
imshow(sigfil)
title(['Sigma Filtered Image, Sigma = ', num2str(sigm), ',
k = ', num2str(k_)])
```

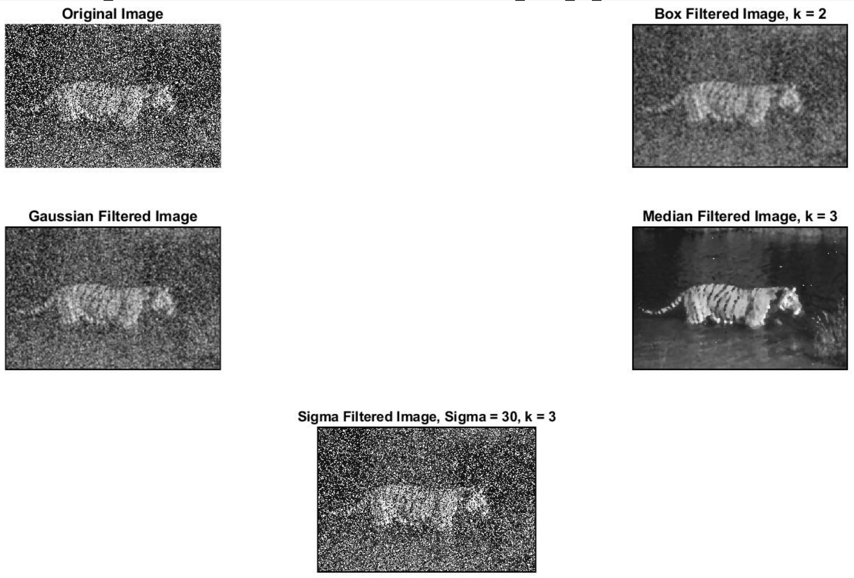
Results and Comparisons

The result is as follows:

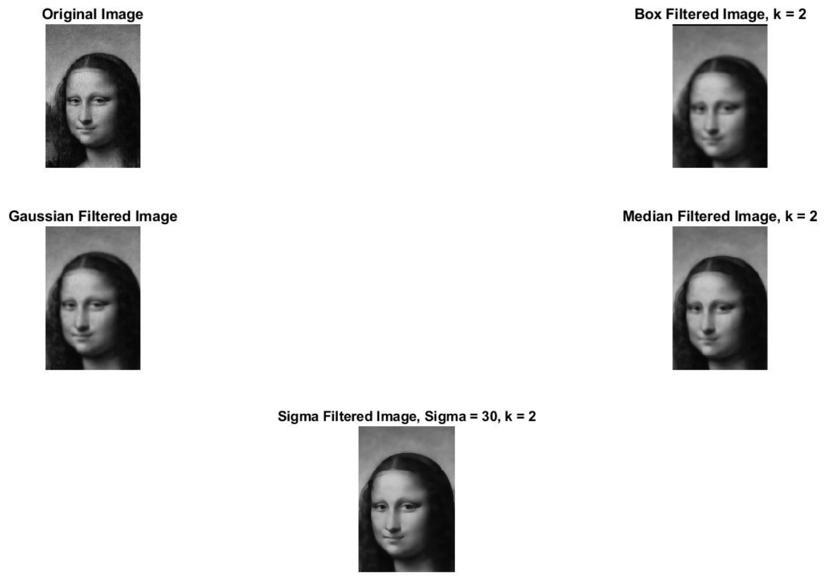


The Sigma filter does a relatively good job of smoothening the image.

How does the Filter perform with salt & pepper noise?


Original Image
Box Filtered Image, k = 2
Gaussian Filtered Image
Median Filtered Image, k = 3
Sigma Filtered Image, Sigma = 30, k = 3

With the above result, it’s safe to say that the Sigma Filter wouldn’t be a prime choice in eliminating speckle noise. In fact, the gaussian and box filters performing significantly better than the sigma filter here, while the median filter would be the ideal choice.


Original Image
Box Filtered Image, k = 2
Gaussian Filtered Image
Median Filtered Image, k = 2
Sigma Filtered Image, Sigma = 30, k = 2

In the above, we see that the box filter introduces the most noise, the gaussian filter seems to smoothen and sort of preserve the original contrast. The median filter performs well, but it's only slightly less blur than the box filtered image. The sigma filter is the least blur of the results.

P.S.

One key discovery while writing the code for the Sigma filter is that MATLAB's inbuilt find function is a straightforward way of writing a pseudo-histogram function

DISCUSSION

-

Gauss Filter

This makes uses of the three sigma-rule because, from statistics, 3σ covers 99.7% of the area under the normal distribution curve.

- Compared to the box filter, the gauss filter introduces lesser noise to the image while smoothening it. This is primarily because the fourier transform of a gaussian returns a gaussian (without any secondary lobes). The absence of secondary lobes means that there'll be lower chances of propagating noise to other places.
- Generally, the default filter is the gaussian filter because it is common not to know the underlying distribution of the noise in an image, so, like a random process, it makes sense to assign a gaussian distribution to the noise.

Median Filter

The median filter is very instrumental in eliminating speckle noise (a.k.a., impulsive noise or salt & pepper noise). It does this by ensuring that the outliers are sent to the extremes after sorting, so when the median is chosen, the outlier is completely taken away.

- The median filter is much better the box filter because it removes outliers with insignificant reduction in contrast.
- However, the above advantage comes with a catch. Instead of the computationally friendly convolution, **the median filter employs the *slower and computationally costlier* sorting procedure**.

Sharpening

An important observation is that sharpening is used to emphasize the high frequency components of an image. This is visible by the increased contrast along the edges in the image.

- The lambda controls the effect of the correction signal [I(p) – S(p)] such that when lambda is too high, the contrast becomes too high, and we'd be left with lots of bright pixels.

- Since the image must be smoothened before it can be sharpened, we can comfortably say that **sharpening is computational costlier than smoothing.**

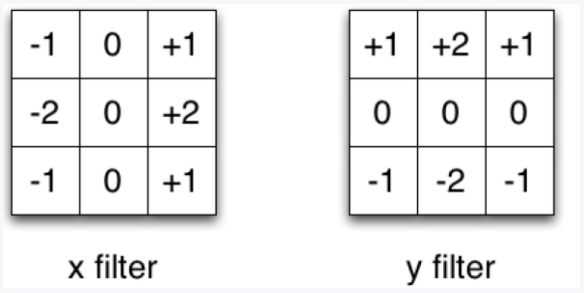
Sobel Filter

The Sobel filter is based on the central difference principle.

- Instead of a 1D filter, a 2D filter is used because 1D filters are very **noise sensitive**.
- Notice that the x-filter simply emphasizes the vertical lines in the image due to the change in that direction.
- Likewise, the y-filter emphasizes the horizontal lines due to the change in that in direction.

Key Observation:

- If we apply derivative directly on noisy images without first smoothing, we'll obtain very bizarre results.
- This is specifically why if we take a closer look at the Sobel filter, we notice that it performs smoothing and derivative simultaneously.



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

x filter

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

y filter

As we can see above, for the:

a. x-filter – derivative is performed horizontally while smoothing is performed vertically.
b. y-filter – derivative is performed vertically while smoothing is performed horizontally.

- From our implementation of the local mean, we clearly see that it's a relatively good method for getting rid of noise in images.

- The central difference method is preferred for taking derivative because it produces errors of the order ε^2 as opposed to $\varepsilon -$

order errors produced by forward and backward difference methods.

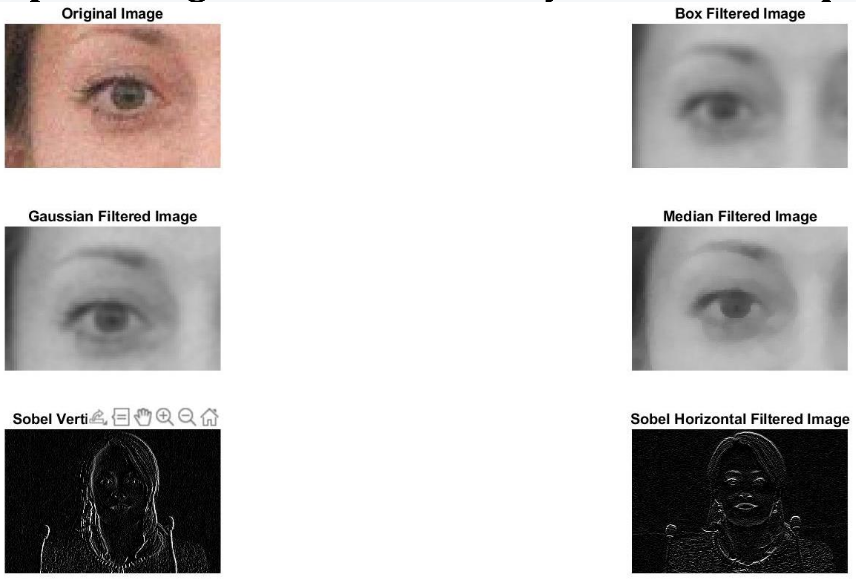
- To properly apply convolution in our code, we implement the element-by-element matrix multiplication as seen below:

```
Ihor(i,j) = sum(wp(:).*y_filt(:));
```

Comparison

Taking the following image:

We apply box filtering, gaussian filtering, median filtering, and Sobel filtering, and crop the region around the eye for close-up comparison:



We observe that the box filter smoothens the image, but it introduces lots of blur. The gaussian filter does a good job as it partially retains strands of the eye lashes. The median filter, which doesn't apply convolution, also properly smoothens the image, however, the eye lashes are completely blurred.
In this specific case, the gaussian filter seems to be the best.

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

https://www.mathworks.com/help/matlab/ref/xlabel.html#btpmg0w-7

https://blog.kyleingraham.com/2016/12/14/salt-pepper-noise-and-median-filters-part-i-the-theory/

https://photography.tutsplus.com/tutorials/how-to-use-imagenomic-noiseware-for-next-level-noise-reduction-on-photos--cms-24041

https://www.revolutiondatasystems.com/blog/grayscale-or-bitonal-which-is-a-better-method-for-scanning-my-records