AAiT

ADDIS ABABA INSTITUTE OF TECHNOLOGY

አዲስ አበባ ቴክኖሎጂ ኢንስቲትዩት

ADDIS ABABA UNIVERSITY

አዲስ አበባ ዩኒቨርሲቲ

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Digital Signal Processing

Course Project

Meambers:

1) Olantu Ebisa.....ATR/8121/11

2) Akililu Wendie.....ATR/1547/11

3) Bereket Abate ATR/0146/09

4) Zelalem Hailu.....ATR/2456/10

5) Naol Abera ETR/6595/11

6) Muluken Tigabie.....ATR/6867/11

Submitted to Yiwab E.

Sept 2021

Overview

1. Introduction

1.1 Understanding Category and Classification on MATLAB

Digital Image Processing refers to the manipulation of image data which is motivated by conversion between Special and Frequency domains. Image classification, or object recognition, is the process of identifying a specific object or class of objects in an image or video. Computer Vision System Toolbox™ offers a variety of algorithms, tools, and techniques to create image classification and object recognition systems. This project shows how to use a "bag of features" approach for image category classification. This technique is also often referred to as bag of words. Visual image categorization is a process of assigning a category label to an image under test. Categories may contain images representing just about anything, for example, dogs, cats, trains, or boats. Here we have tried to show Image Blurring, edge detection, line detection, sharpening an image, image histogram, compressing an image, fast-Fourier transform on images, low-Pass filter on images, high pass filter on images. This are some application areas of DSP on image category but we tried to relate the concepts that we have learned in class to real time applications. Through this project imread() and imshow() functions are used to load and display an images respectively.

2. Image Blurring

When we blur an image, we make the color transition from one side of an edge in the image to another smooth rather than sudden. The blur, or smoothing, of an image removes “outlier” pixels that may be noise in the image. Blurring is an example of applying a low-pass filter to an image.

Here to blur images we have used a Gaussian filter a filter whose impulse response is a Gaussian function (or an approximation to it, since a true Gaussian response would have infinite impulse response). It is considered the ideal time domain filter, just as the sinc is the ideal frequency domain filter. The Gaussian Filter Compressed higher frequencies of an image perfectly and we have got a blurred output image.

```
pom = imread('Zeritu.png');
gaussianfilter = fspecial ('gaussian',[7,7],5);
gaussianpom = imfilter(pom, gaussianfilter, 'symmetric', 'conv');
```

```
subplot(1, 2, 1), image(pom),title('Original Image')
subplot(1, 2, 2), image(gaussianpom), title('Blured image, blurmatrix7');
```

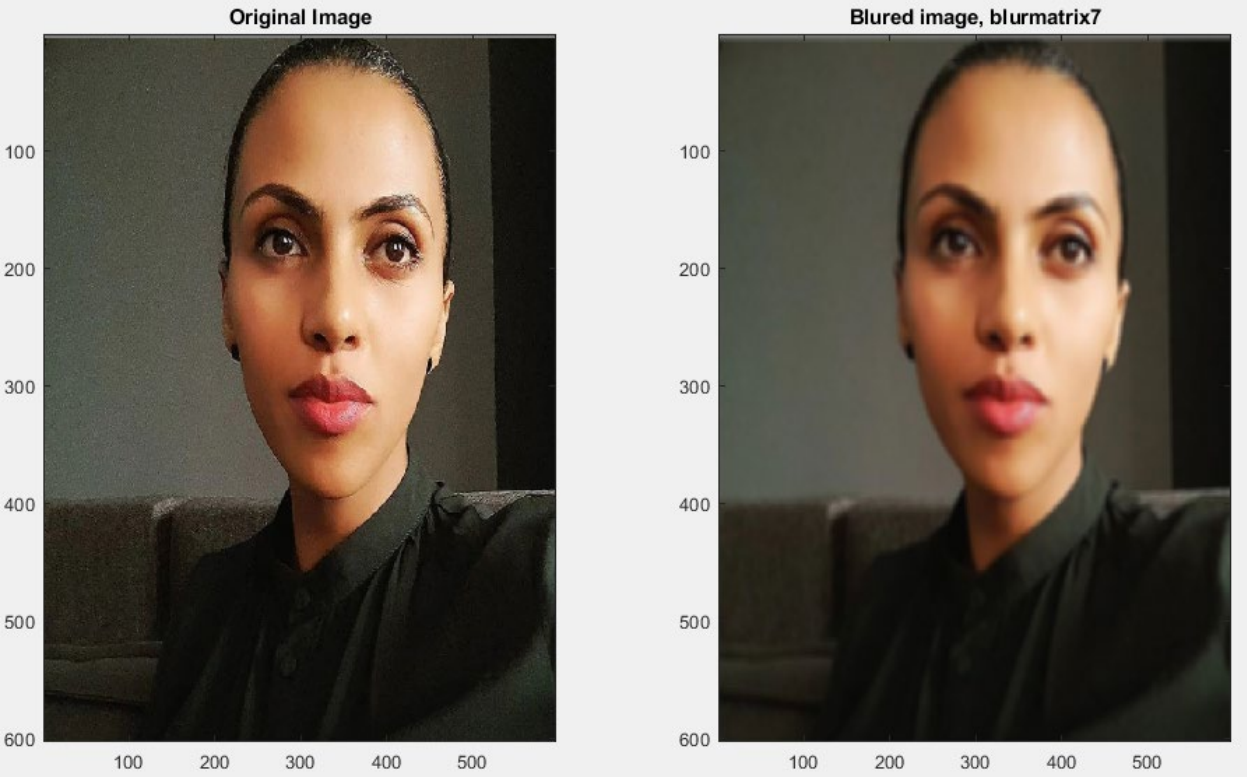

Figure 1. Image blurring

3. Edge Detection

Edge detection is a technique of image category is used to identify points in a digital image with discontinuities, simply to say, sharp changes in the image brightness. These points where the image brightness varies sharply are called the edges (or boundaries) of the image.

Here we have used Sobel filter for detecting edges in an image. It works by calculating the gradient of image intensity at each pixel within the image. It finds the direction of the largest increase from light to dark and the rate of change in that direction. We can think of edge detection as a high pass filter operation.

Mathematical Formulation of the Sobel Operator

1. G_x = x-direction kernel * (3×3 portion of image A with (x,y) as the center cell)
2. G_y = y-direction kernel * (3×3 portion of image A with (x,y) as the center cell)
3. magnitude(G) = square_root($G_x^2 + G_y^2$)
4. $\Theta = \tan^{-1}(G_y / G_x)$

```
clear all;
clc;

pic = imread('image.jpg');
edges = [];

% applying sobel filter on individual layers
edges(:,:,1) = edge(pic(:,:,1),'sobel');
edges(:,:,2) = edge(pic(:,:,2),'sobel');
edges(:,:,3) = edge(pic(:,:,3),'sobel');

% plotting the results channel wise
subplot(221),imshow(pic),title('Original image');
subplot(222),imshow(edges(:,:,1)),title('Red channel');
```

Sobel filters applied independently on red, green and blue layers of the input image. The results were shown combined, as well as distinct.

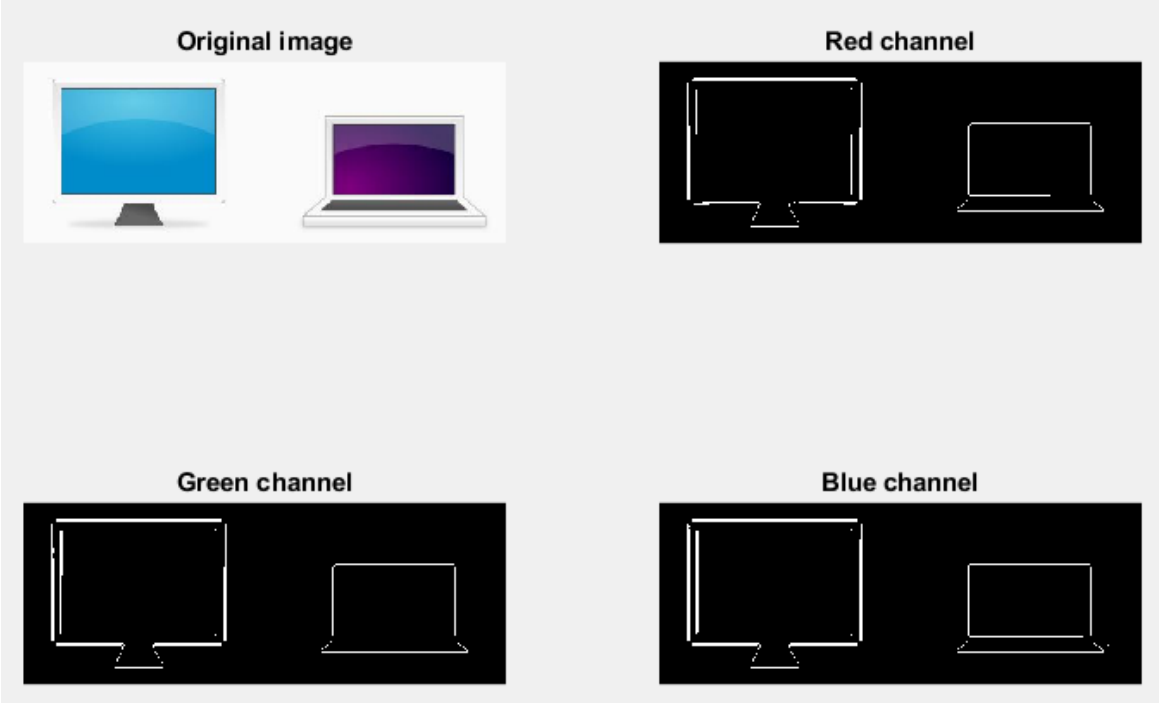

Figure 2. Edge detection Sobel filters on red, green and blue layers of the input image

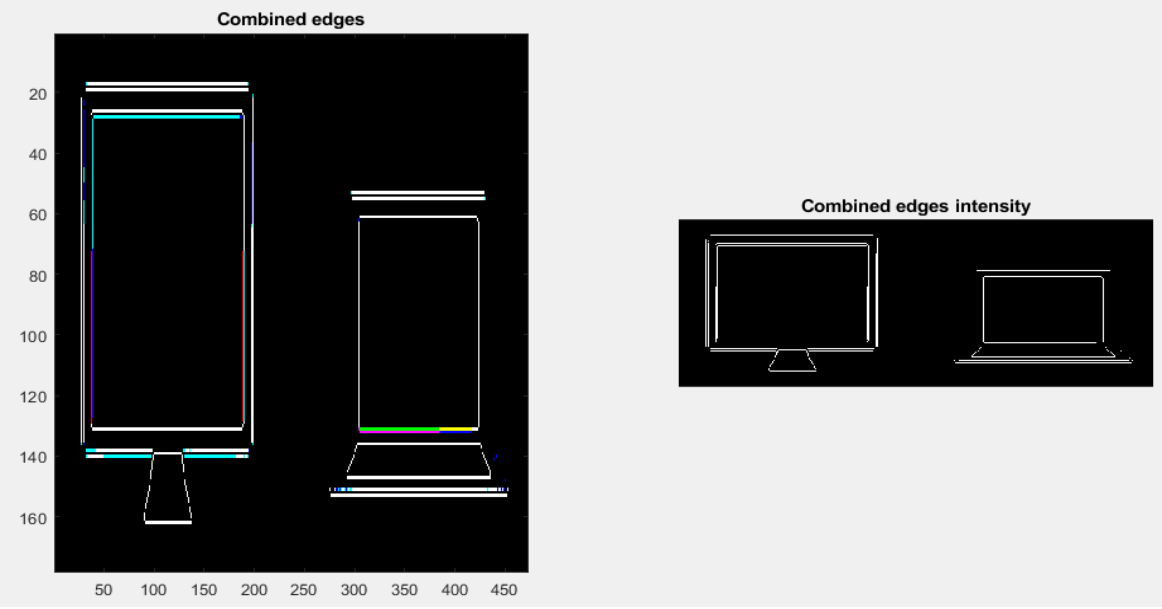

Figure 3. Edge detection combined images

We can also use kernel matrices with images for edge detection. In image category, many filter operations are applied to in image by performing a special operation called convolution with a matrix. this matrix is called a KERNEL. Kernels are typically 3x3 square matrices, although kernels of size 4x4, 5x5 are also used. The values stored in a kernel directly relates to the result after applying the filter, and filters are characterized solely by kernel matrix. For example, the following kernels are used for detecting the horizontal and vertical edges in an image.

```
% Define the Sobel kernels
kernel_horizontal = [1 2 1;0 0 0;-1 -2 -1];
kernel_vertical = [1 0 -1;-2 0 2;-1 0 1];
% Convolve the gray image with Sobel Kernels
M1 = conv2(double(gray_image), double(kernel_horizontal));
M2 = conv2(double(gray_image), double(kernel_vertical));
```

double is used because the loaded image was by defaults in Unit8 format, which needs to be converted in numbers before processing.

Figure 4. kernel matrices with images for edge detection

Displaying the horizontal and vertical edges separately

```
figure(4)
imshow(abs(M1),[]);
title('horizontal parts');
figure(5)
imshow(abs(M2),[]);
title('vertical parts');
```

Figure 5. displaying the horizontal and vertical edges separately

4. Line Detection

In Edge Detection, a pixel is attenuated, if there is a dramatic change in color in any direction. Line detection is a special kind of edge detection. For line detection, the direction in which a color changes are considered is restricted. Common filter kernels used to detect horizontal, vertical and diagonal edges in the input image successfully.

The common filter kernels are

```
edge = [-1 -1 -1;- 1 8 -1;-1 -1 -1];
horizontal = [-1 -1 -1;2 2 2;-1 -1 -1];
vertical = [-1 2 -1;-1 2 -1;-1 2 -1];
diagonal_1 = [-1 -1 2;-1 2 -1; 2 -1 -1];
diagonal_2 = [2 -1 -1;-1 2 -1;-1 -1 2];
```

```
building = imread('build.jpg');
imshow(building), title('ORIGINAL IMAGE')
```

Figure 6. displaying the original image of line detection

```
horizontal_building = imfilter(building, horizontal);
vertical_building = imfilter(building, vertical);
diagonal_1_building = imfilter(building, diagonal_1);
diagonal_2_building = imfilter(building, diagonal_2);
```

```
subplot(221),imshow(horizontal_building), title('Horizontal edges')
subplot(222),imshow(vertical_building), title('Vertical edges')
subplot(223),imshow(diagonal_1_building), title('Diagonal UP edges')
subplot(224),imshow(diagonal_2_building), title('Diagonal DOWN edges')
```

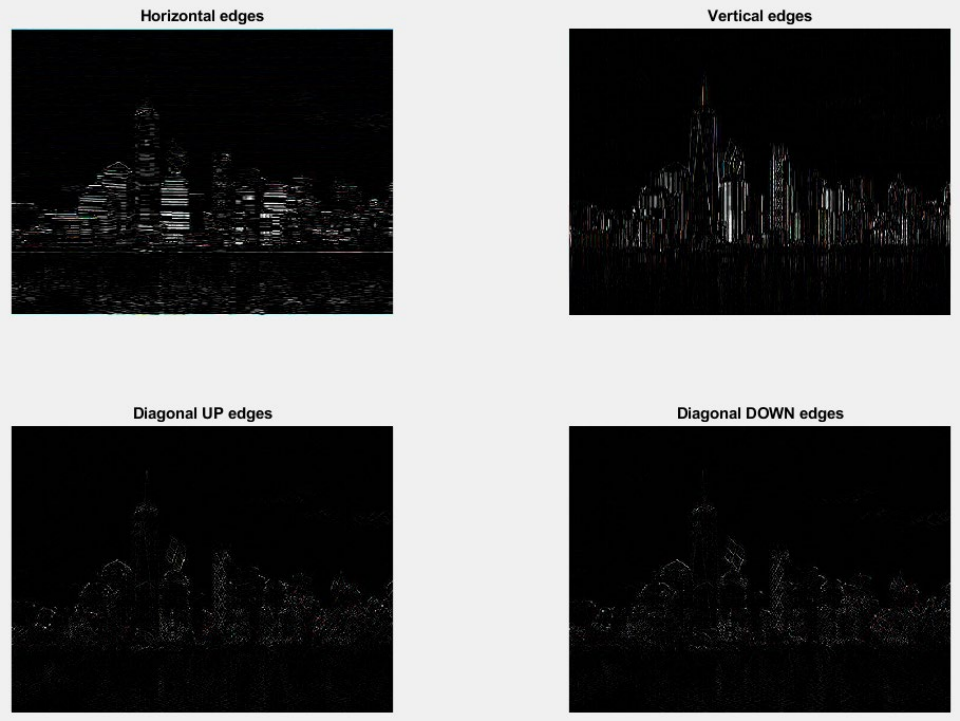

Figure 7. Horizontal, Vertical, Diagonal UP, Diagonal DOWN edges

5. Sharpening an Image

The sharpening process works by first creating a slightly blurred version of the original image, the unsharp mask. This is subtracted away from the original to detect the presence of edges. Contrast is then selectively increased along these edges using this mask leaving behind a sharper final image. We have used a function using fspecial() for sharpening. Input image sharped using fspecial filter. Excessive sharpening however produces negative results. h = fspecial('motion', len , theta) returns a filter to approximate, once convolved with an image, the linear motion of a camera. len specifies the length of the motion and theta specifies the angle of motion in degrees in a counter-clockwise direction. The filter becomes a vector for horizontal and vertical motions.

```
im = imread('image.jpg');
sharpfilter = fspecial('unsharp');
subplot(131),imshow(im),title('ORIGINAL');
sharp = imfilter(im,sharpfilter,'replicate');
subplot(132),imshow(sharp),title('Sharpened image');
sharpmore = imfilter(sharp,sharpfilter,'replicate');
subplot(133),imshow(sharpmore),title('Excessive sharped image');
```

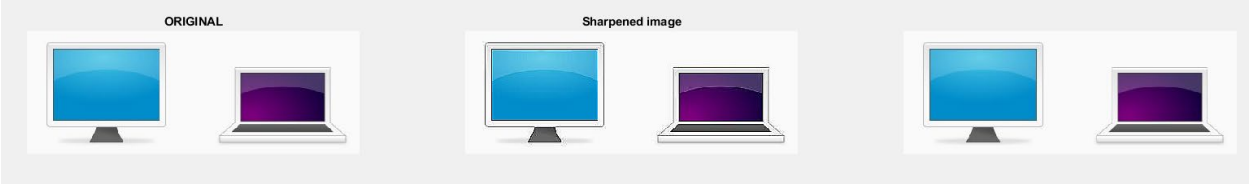

Figure 8. Image sharpening

6. Image Histogram

In an image category context, the histogram of an image normally refers to a histogram of the pixel intensity values. This histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. Imhist() function plotting the intensities of a gray scale image successfully. The imhist function creates a histogram plot by defining n equally spaced bins, each representing a range of data values, and then calculating the number of pixels within each range. We can use the information in a histogram to choose an appropriate enhancement operation.

```
I = imread('Zeritu.png');
I = rgb2gray(I);
imshow(I),title('Input Image');
h = imhist(I,256);
figure
subplot(121),imhist(I),title('Histrogram - imhist function'),ylim('auto');
%Plotting using bar function
h1 = h(1:10:256);
horz = 1:10:256;
subplot(122),bar(horz,h1),title('Histogram - bar function.');
```

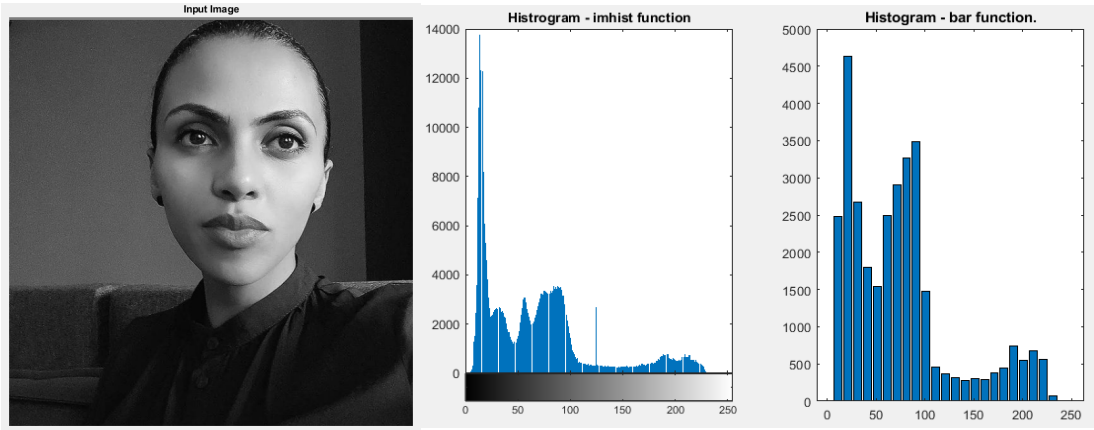

Figure 9. Input image and Image Histogram graphs

7. Compressing an image

Image compression is the process of encoding or converting an image file in such a way that it consumes less space than the original file. It is a type of compression technique that reduces the size of an image file without affecting or degrading its quality to a greater extent. Our eyes only see low frequencies, so we can remove higher frequencies without affecting the image mush to the end user. Image compression: 50%, 25% and 12.5% done successfully on image.

```
clear all;
close all;
clc;

pic = imread('image_2.jpg');

origWidth = size(pic, 2);

samplesHalf = floor(origWidth / 2);
samplesQuater = floor(origWidth / 4);
samplesEighth = floor(origWidth / 8);

picCompressed2 = [];
picCompressed4 = [];
picCompressed8 = [];

for k = 1:3
    for i = 1:size(pic, 1)
        rowDCT = dct(double(pic(i,:,k)));
        picCompressed2(i,:,k) = idct(rowDCT(1:samplesHalf),origWidth);
        picCompressed4(i,:,k) = idct(rowDCT(1:samplesQuater),origWidth);
        picCompressed8(i,:,k) = idct(rowDCT(1:samplesEighth),origWidth);
     end
 end

subplot(221),image(uint8(pic)),title('Original Image');
subplot(222),image(uint8(picCompressed2)),title('Compression Factor : 2');
subplot(223),image(uint8(picCompressed4)),title('Compression Factor : 4');
subplot(224),image(uint8(picCompressed8)),title('Compression Factor : 8');
```

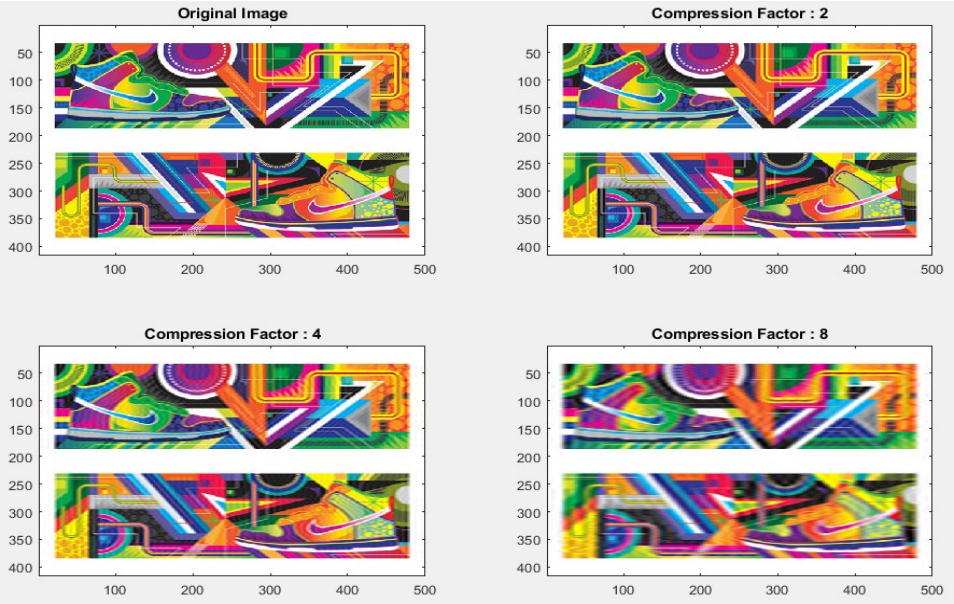

Figure 10. Original image and images with increasing compression factor

8. Fast-Fourier Transform on Images

The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. Fast-Fourier Transform of image produces the same results as Digital Fourier Transform, however it as faster in computations.

```
clc;
clear all;
close all;
I = imread('image.jpg');
I = im2double(I);
subplot(221),imshow(I),title('Original Image');
F = fftshift(fft2(I));
subplot(222),imshow(abs(F)),title('Magnitude spectrum');
G = mat2gray(log10(1+abs(F)));
subplot(223),imshow(G),title('Log Scaled magitude spectrum');
I_recons = ifft2(ifftshift(F));
subplot(224),imshow(I_recons),title('Reconstructed image');
```

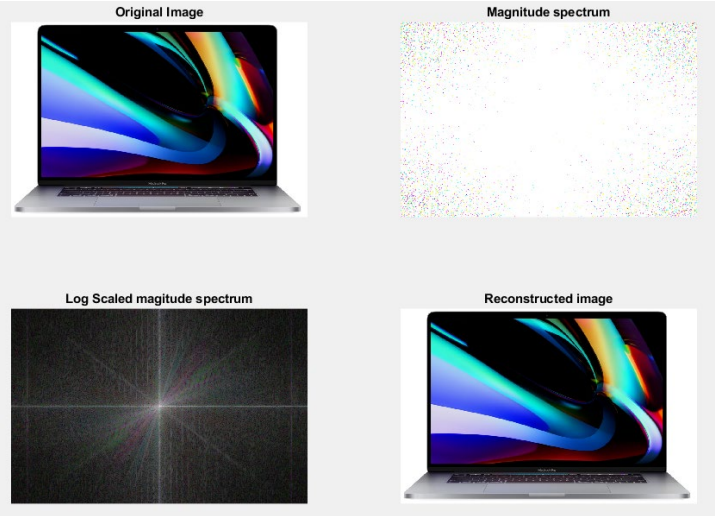

Figure 11. Original image magnitude spectrum and reconstructed image

9. Low-Pass Filter on Images

Low Pass Filter allows only low frequencies of the image, and bocks higher frequencies. As our eyes more sensitive to low frequencies, hence the output is not much different to the input.

```
clear all;
clc;
close all;
```

```
I = imread('Zeritu.png');
I = rgb2gray(I);

[M,N] = size(I);
```

Finding D(u,v)

```
D = zeros(size(I));for
u = 1:M
    for v = 1:N
        D(u,v) = ((u-(M/2))^2 + (v-(N/2))^2)^(1/2);
    end
end
```

Finding H(u,v)

```
H = zeros(size(I));for
u = 1:M
    for v = 1:N
        H(u,v) = 1/(1 + (D(u,v)/20)^4);
    end
end
```

Processing

```
F = fft2(I);
F = fftshift(F);Y
= F.*H;
Y1 = ifftshift(Y);y =
ifft2(Y1);
```

Displaying results

```
subplot(121),imshow(I),title('Original Image');
subplot(122),imshow(uint8(y)),title('Output image');
```

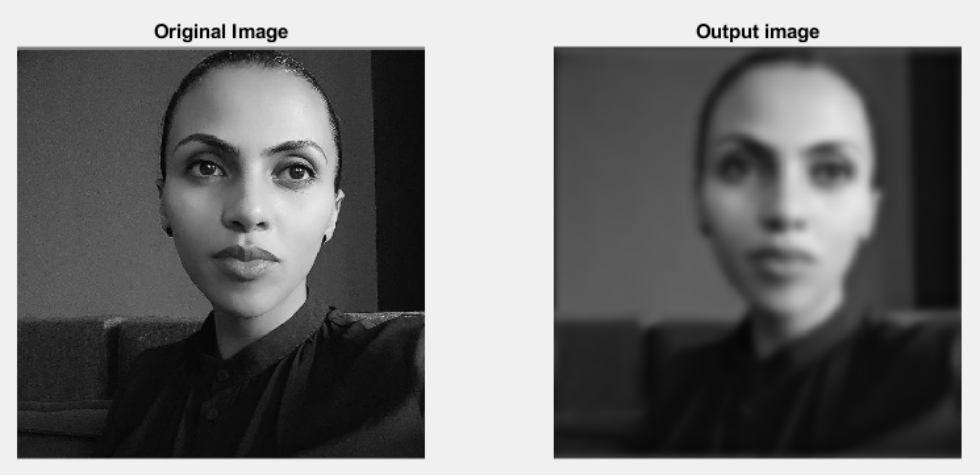

Figure 12. The original image and an image out of the low-pass filter

10. High-Pass Filter on Images

A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information. The kernel of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels. The High Pass Filter Blocks low frequencies on the image, hence the image loses its definitions.

```
clear all;
clc;
close all;
```

```
I = imread('Zeritu.png');
I = rgb2gray(I);
```

Finding D(u,v)

```
D = zeros(size(I));for
u = 1:M
    for v = 1:N
        D(u,v) = ((u-(M/2))^2 + (v-(N/2))^2)^(1/2);
    end
end
```

Finding H(u,v)

```
H = zeros(size(I));for
u = 1:M
    for v = 1:N
        H(u,v) = 1/(1 + (D(u,v)/20)^-4);
    end
end
```

Processing

```
F = fft2(I);
F = fftshift(F);Y
= F.*H;
Y1 = ifftshift(Y);y =
ifft2(Y1);
```

Displaying results

```
subplot(121),imshow(I),title('Original Image');
subplot(122),imshow(uint8(y)),title('Output image');
```

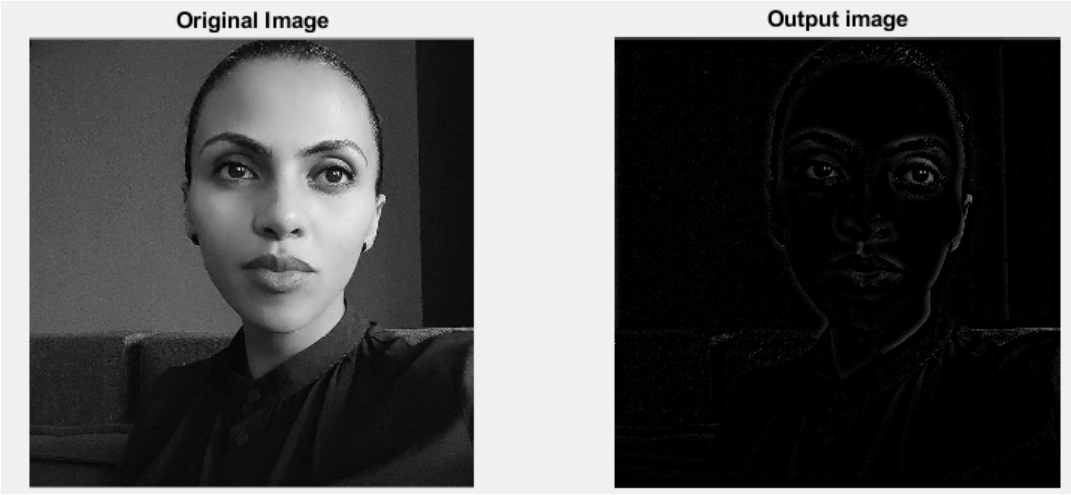

Figure 13. The original image and an image out of high-pass filter