Computing Assignment – 3B
Saqib Azim - 150070031

main.sce

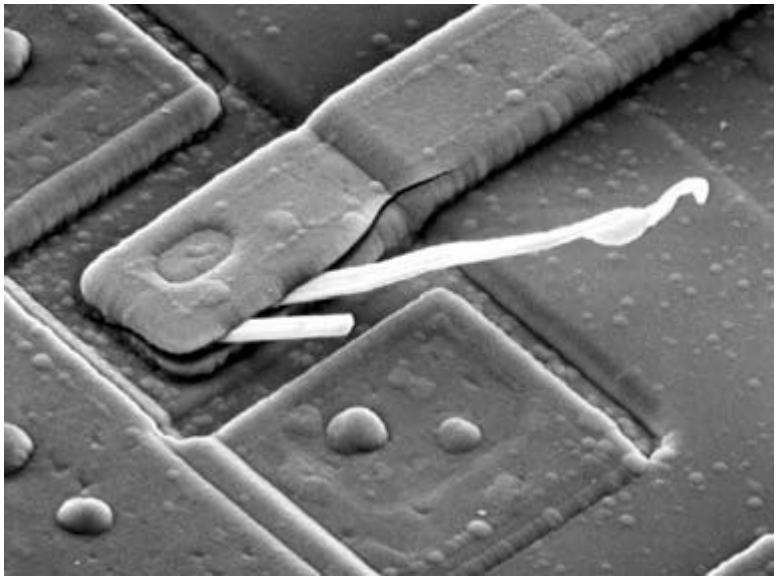
```
exec('/home/saqib1707/Acads/4thsem/EE210_Signals&Systems/3_Assignment/Image_Assignment/conv_2D.sci');


// You can use the function 'conv_2D.sci' to to do the convolution between two 2D-signals
// The inputs to the function are, the input image 'X' and the kernel 'H'.
// The output will be the resultant signal 'Y' after the convolution.
// You can call this function multiple times depending on how many times you are performing the convolution.

// Below an example code is given to read an image and to find fft of the image. Also the magnitude of the fft (in logarithmic scale) is plotted
// you can use this code for finding the 2D fft and ploting its spectrum

// get the input image
//X = imread('/home/saqib1707/Acads/4thsem/EE210_Signals&Systems/3_Assignment/Image_Assignment/Images/mypic.jpg');
X = imread('/home/saqib1707/Acads/4thsem/EE210_Signals&Systems/3_Assignment/Image_Assignment/Images/sem_ic.jpg');
X = rgb2gray(X);

//display the image
//imshow(X)

// convert to double format for performing calculations
X = im2double(X);
```

```
//imwrite(X,'/home/saqib1707/Acads/4thsem/EE210_Signals&Systems/3_Assignment/Image_Assignment/Images/sem_ic_orig.png');
//finding fft of the image
Fin = fft(X);
Fin = fftshift(Fin);


// Now you can define the kernel H and call the convolution function here. An example is shown below
//define a kernel
H1= [1/9 1/9 1/9;1/9 1/9 1/9;1/9 1/9 1/9];   //  will produce a blur image
H2= [0 -1 0; -1 5 -1 ; 0 -1 0];              // will sharpen the image
H3= [1/16 2/16 1/16; 2/16 4/16 2/16;1/16 2/16 1/16];              // Gaussian blur
H4= [0 -1 0; -1 4 -1 ; 0 -1 0];

// Now call the fuction
Y = conv_2D(X,H4);
Y = conv_2D(Y,H4);
Y = conv_2D(Y,H4);
Y = conv_2D(Y,H4);
Y = conv_2D(Y,H4);
imshow(Y);
//imwrite(Y,'/home/saqib1707/Acads/4thsem/EE210_Signals&Systems/3_Assignment/Image_Assignment/Images/Badass_H2_5.png');
imwrite(Y,'/home/saqib1707/Acads/4thsem/EE210_Signals&Systems/3_Assignment/Image_Assignment/Images/sem_ic_H4_5.png');


// You can call this function as many times you needed
Y = im2double(Y);
output = fft(Y);
output = fftshift(output);
//plot of magnitude spectrum
Fin3 = log(abs(output));
set(gcf(),"color_map",graycolormap(128));
```

```
clf;
Fin3=flipdim(Fin3,1);
//grayplot(-229:230,-229:230,Fin3');
grayplot(-192:192,-143:142,Fin3');
// title and axes properties
b = get("current_axes");
b.title.text="Magnitude Spectrum of output image ";
b.axes_visible="on";
b.tight_limits="on";
b.x_location = "bottom";
b.y_location = "left";
b.x_label.text="v";
x_label.auto_position="on";
b.y_label.text="u";
// Now you plot the output image, find its fft , and plot its magnitude spectrum here. Compare input and output in both the spatial domain and frequency domain
// While using the function 'imshow' to display the image check the data type of the image and do proper scaling to get a better display of the image. For more details enter 'help imshow'
```

1 . Read and display an image (If the image is a color image, convert it in to a grayscale image since here we are not interested in color dependent properties).

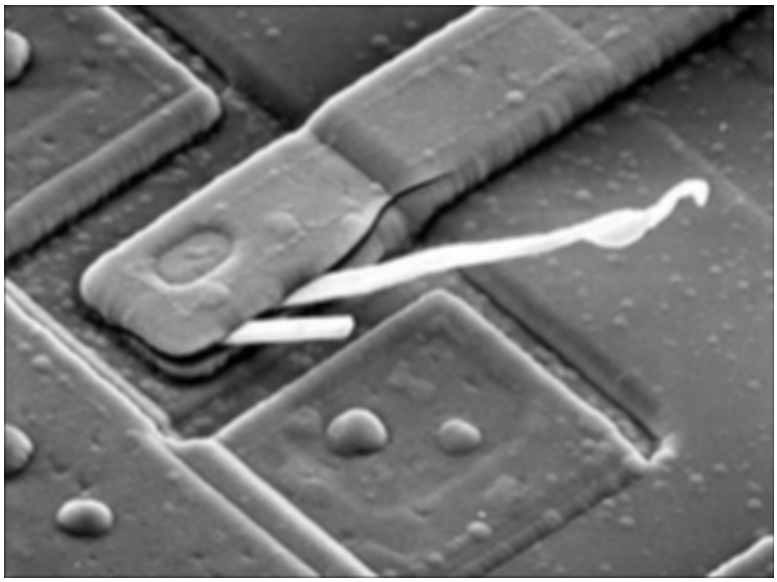
Original Image-



With this kernel -

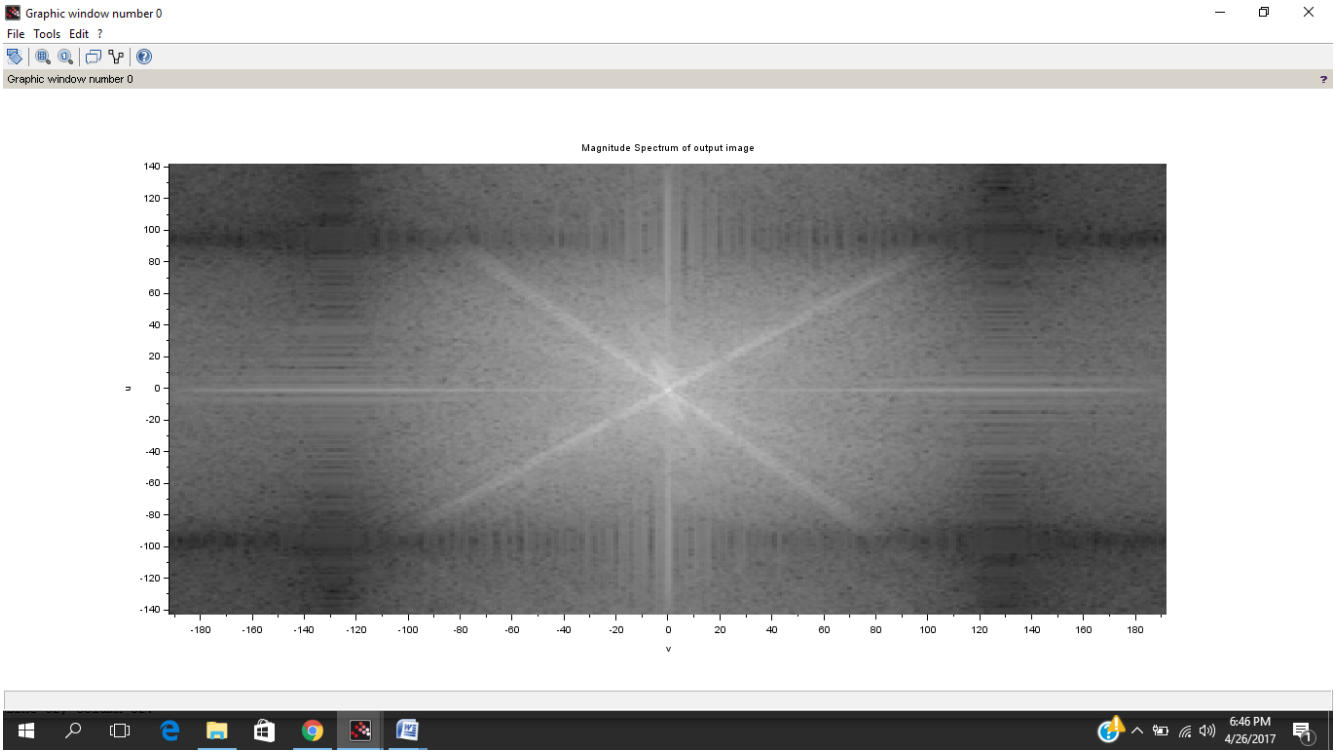
$$H1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The image after convolving one time which basically functions as blurring the image.

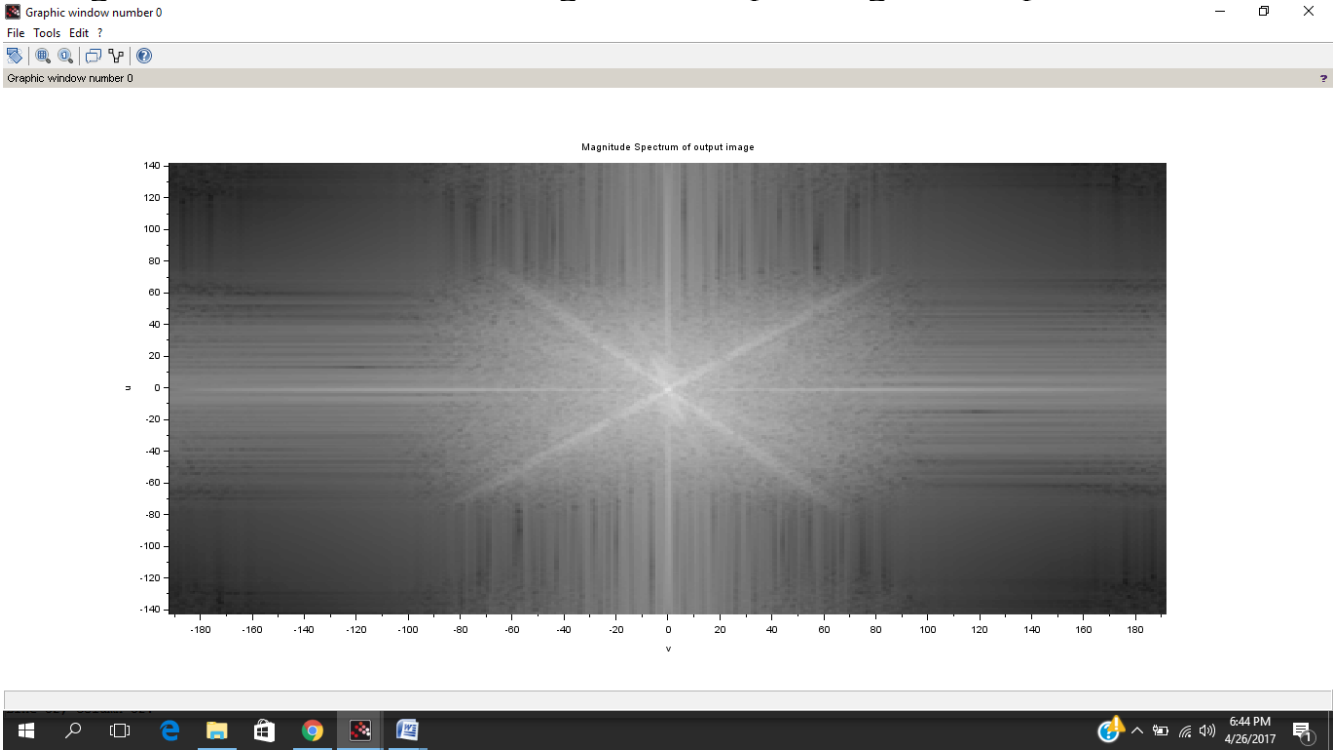


The differences between the output and the original image is -
>>> The output image is a little blurred as compared to the original image. Since the kernel basically filters the low frequencies near the edges resulting in making the image blurred.

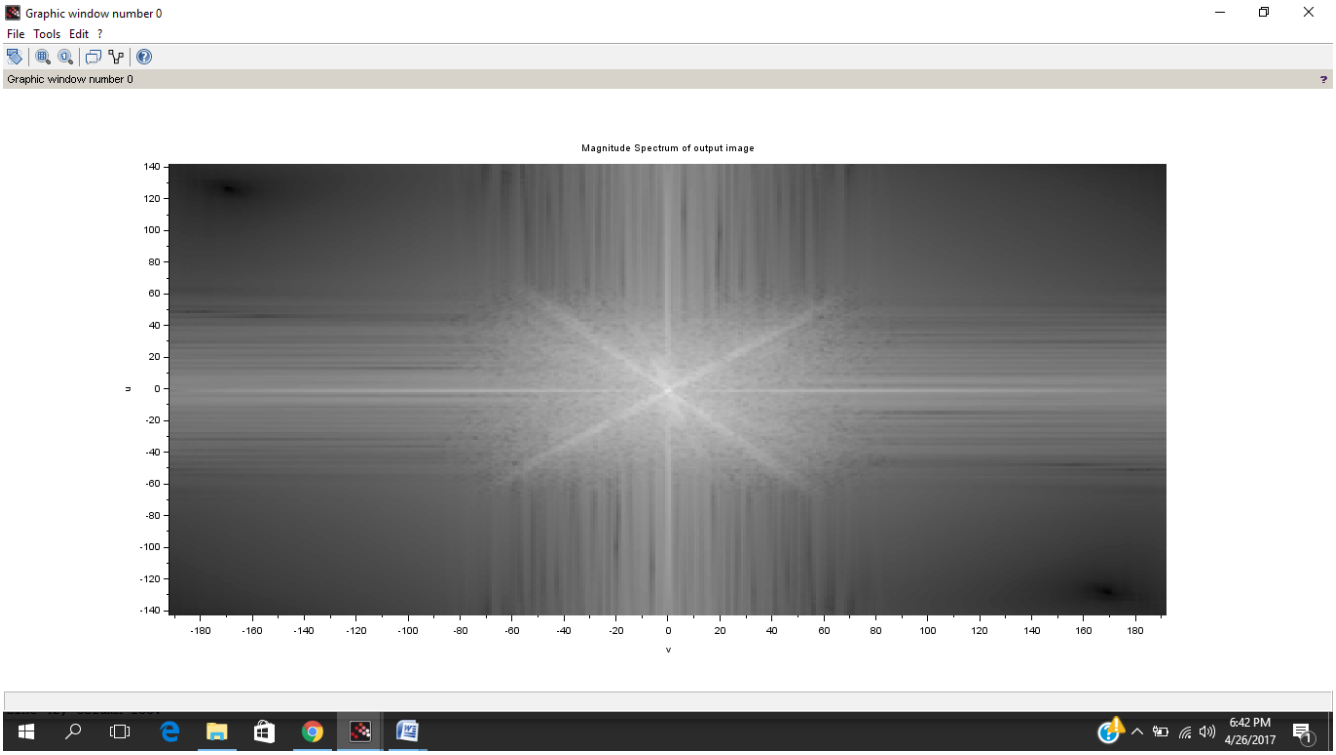
Doing the convolution just one time, I got this output magnitude spectrum-



Doing the convolution 3 times I got this output magnitude spectrum-



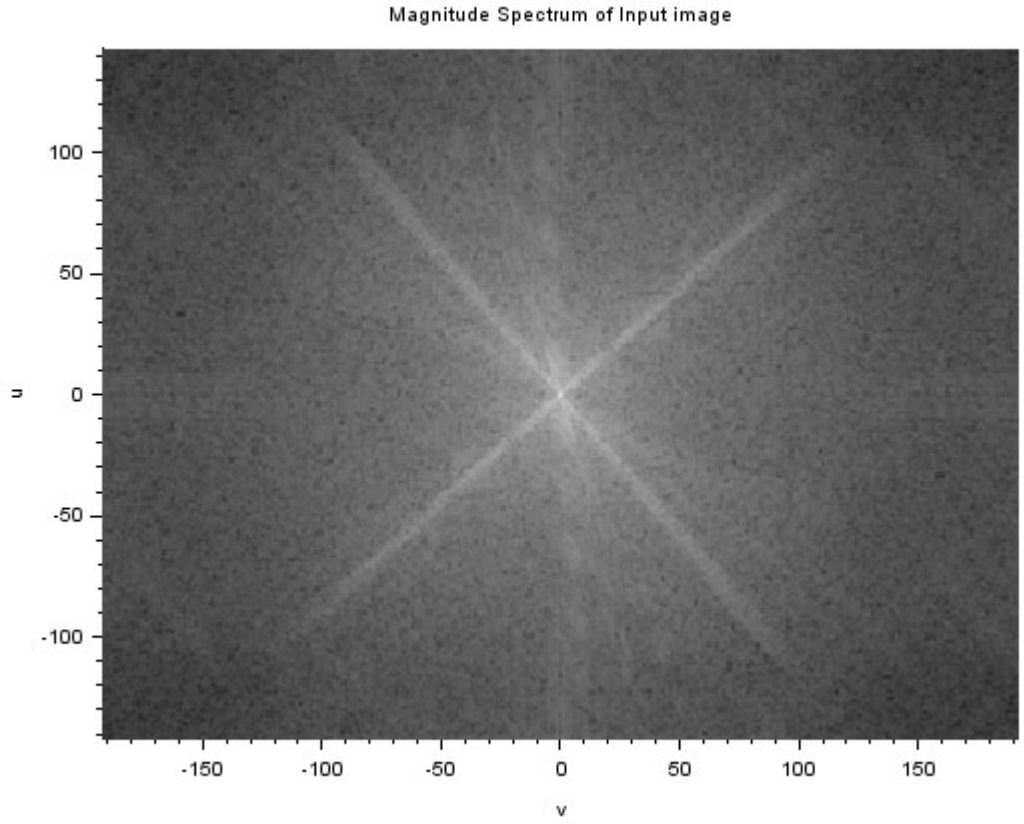
Doing the convolution 5 times I got this output magnitude spectrum-



From the above frequency spectrum it can be clearly seen as the number of convolutions is increased , the higher frequencies get vanished since the kernel acts as a low pass filter. The kernel matrix(mask) takes the average of the itself with the surrounding 8 values and puts it in the middle (2,2) position. Thus any irregularity or sudden changes in pixel values are somewhat reduced.

4 . Next find DFT of the output image and compare the spectra of input and output images to interpret the filtering effect. Comment on the output image features w.r.t. the input image.-

>>>The magnitude spectra of the original image is -

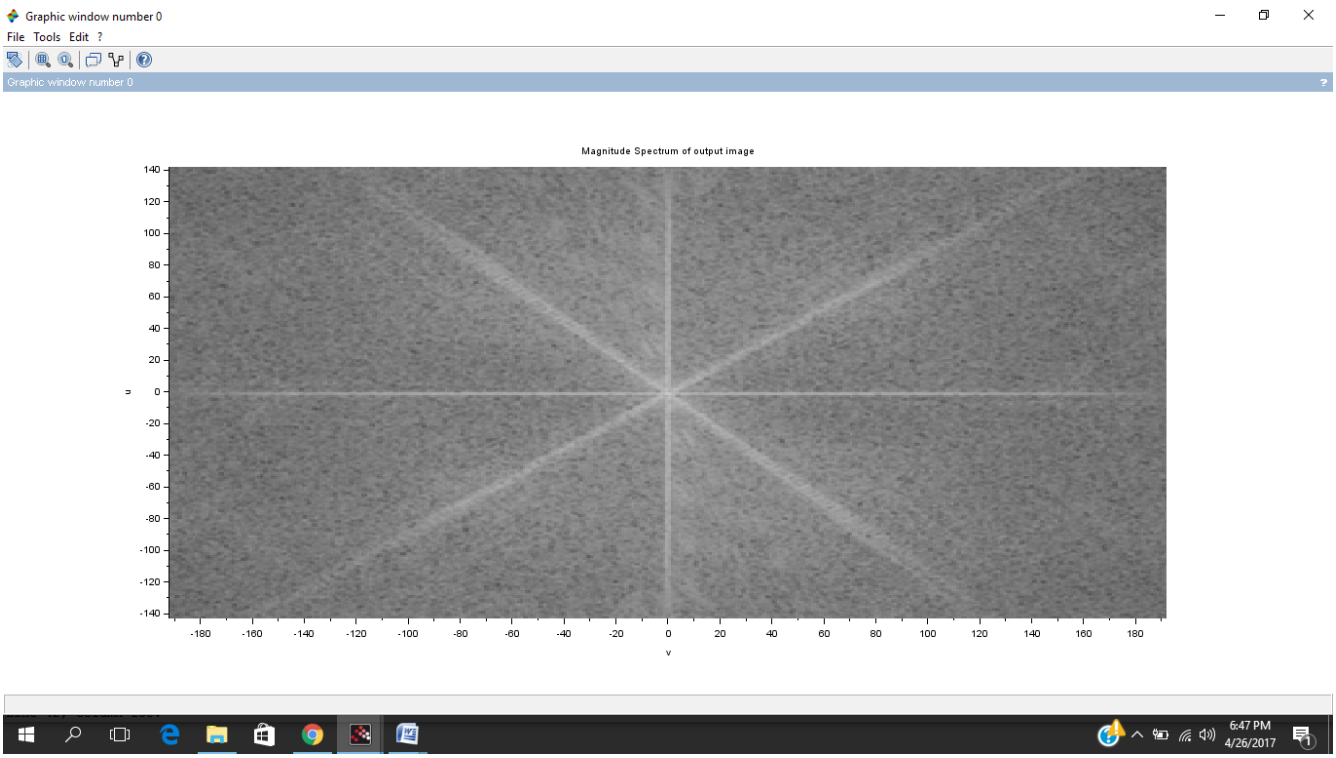


5 . Repeat the above steps with a new kernel , H2, as given below--

With this kernel -

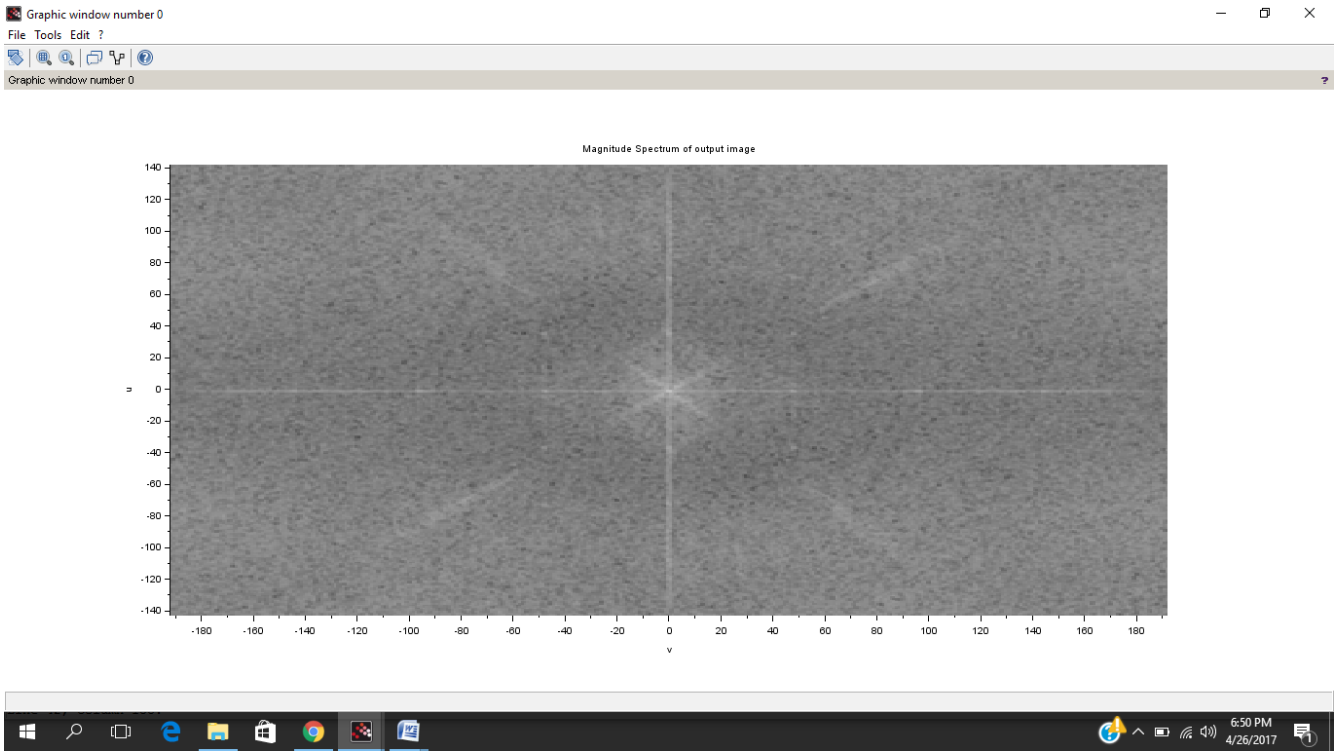
$$H2=\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Doing the convolution just 1 time we got this output magnitude spectrum-

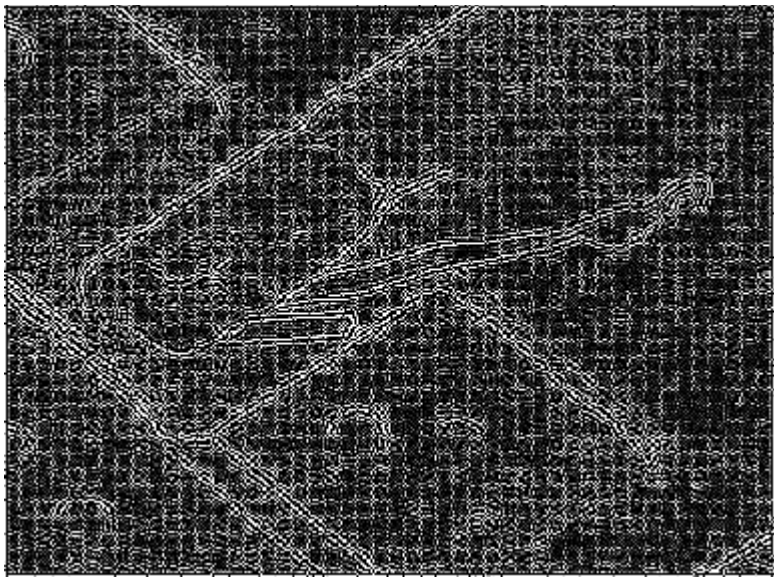


The image after one convolution looks like this-

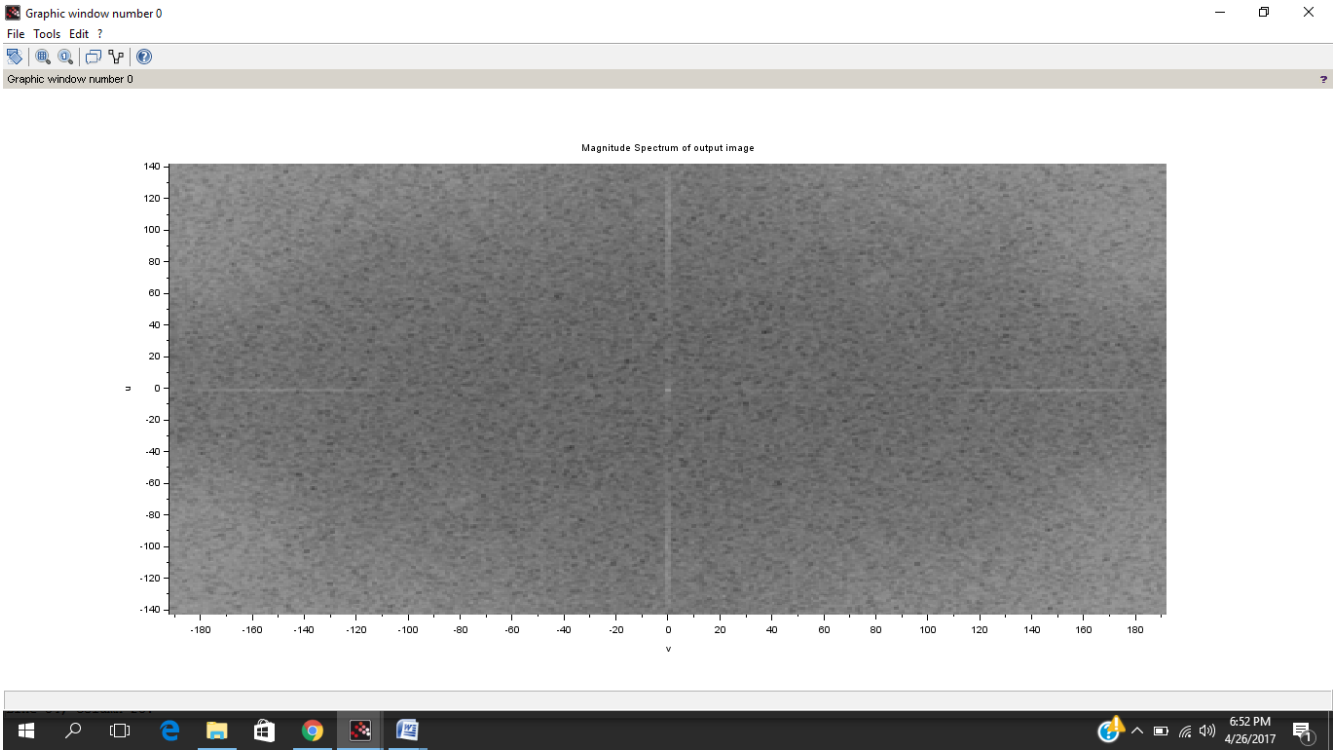
Doing the convolution 3 times we got this output magnitude spectrum-



The image after three convolutions looks something like this-



Doing the convolution 5 times we got this output magnitude spectrum-



The second kernel acts as a high pass filter .Since the matrix has -1 at the positions adjacent to the center so while masking it subtracts the values at the surrounding positions from four times the value at the center (2,2) kernel matrix position. In the similar areas(not edges) where grayscale values are almost similar this leads to a zero making that pixel black. While near the edges where the adjacent values are different with respect to the center , the corresponding center pixel is clearly filled with a high value making those pixels(edge region) detectable. Hence this kernel has a high use in detecting the edges of objects and hence identifying them.

6. Based on your observations, can you suggest useful image applications for each of the filter types?

>>>The first kernel can be used to blur the image since it acts as a low pass filter and hence the changes near the edges are blurred making the image a little blurred.

while the 2nd kernel can be used for the edge detection since the second kernel acts as a high pass filter and thus only changes near the edges are distinctly visible. This can be used for detecting objects of any specific shape by matching its values with its edge values.

T*he complete experiments done with diferent images can be seen here at the github link since it was not possible to include everything in the ppt*-

https://github.com/saqib1707/EE-210-Audio-Assignment-Project/tree/master/3_Assignment/Image_Assignment

Something experiments which I did with my image-