COSC 6380 Digital Image Processing

DFT and Frequency Filtering

1. DFT

In this part, I have implemented functions to calculate forward fourier transform, inverse fourier transform, discrete cosine transform and magnitude of the fourier transform. The input to the functions is a 2D matrix of size 15X15.

For computing discrete fourier transform, I have written a code in forward_transform() function which is called in main() function. The formula I used is below. To initialize the empty matrix for computation, I used dtype as complex.

$$\tilde{I}(u,v) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} I(i,j)\, e^{-\sqrt{-1}\frac{2\pi}{N}(ui+vj)}$$

Figure (1). Discrete Fourier Transform

For computing inverse discrete fourier transform, I have written the formula mentioned below in inverse_transform() function. It gives complex values. To initialize the empty matrix for computation, I used dtype as complex.

$$I(i,j) = \sum_{u=0}^{N-1}\sum_{u=0}^{N-1} \tilde{I}(u,v)\, e^{\sqrt{-1}\frac{2\pi}{N}(ui+vj)}$$

Figure (2). Inverse Discrete Fourier Transform

To display the complex values of inverse fourier transformed image, I have written magnitude() function which returns the absolute value of complex number. For calculating cosine transform of the given 2D matrix, I used the formula mentioned in figure (3) in discrete_cosine_tranform() function. To initialize the empty matrix for computation, I used dtype as float.

$$\tilde{I}_{real}(u, v) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} I(i, j) \cos\left[\frac{2\pi}{N}(ui + vj)\right]$$

Figure (3). Discrete Cosine Transform

Additionally, I have implemented the function for computing sine values of the 2D matrix (imaginary part of the transform). I used multiple 'for loops' to compute value at particular position for discrete fourier transform, inverse discrete transform, discrete cosine transform. Although I have used np.abs() function from numpy library to calculate the magnitude once, I have implemented 'for loops' to compute absolute value as well.

Problem faced:
I did not face any problem in this part of the assignment. For confirming the solution of dft and inverse dft, I used in-built functions as well.

Command to run the functions mentioned above,
python dip_hw3_dft.py

2. Frequency filtering

Ideal lowpass filter:

For computing the ideal lowpass filter mask, I have used the formula below,

Ideal Lowpass Filters (ILPF)

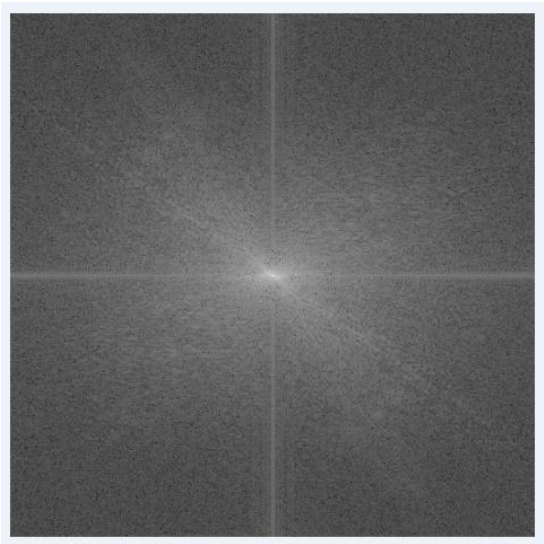
$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

where, D0 is the cut-off frequency and D is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle. Distance formula is as below,

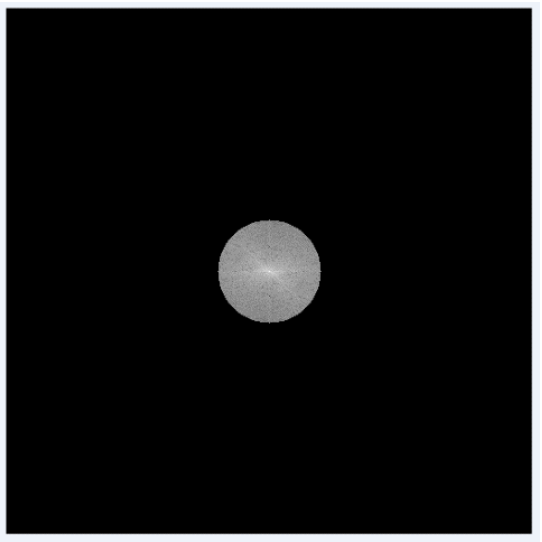
$$D(u,v) = \left[(u - P/2)^2 + (v - Q/2)^2 \right]^{1/2}$$

Convoluting lowpass filter with input image returns smoothness or low frequency values from the image.

Output of the ideal lowpass filter:



DFT of the input image



DFT of the filtered image

Filtered image

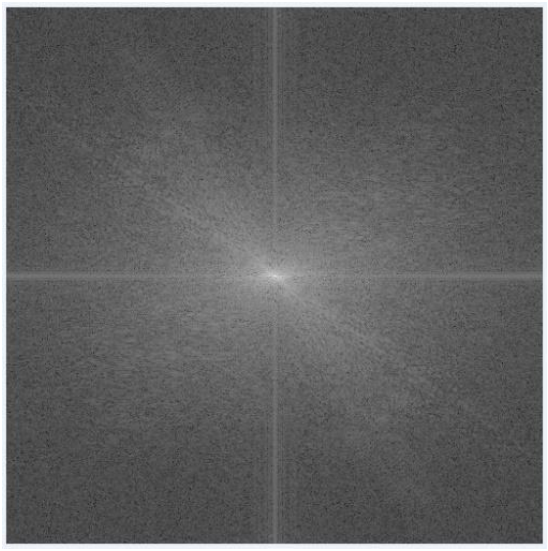
Ideal highpass filter:

For computing the ideal highpass filter mask, I have used the formula below,

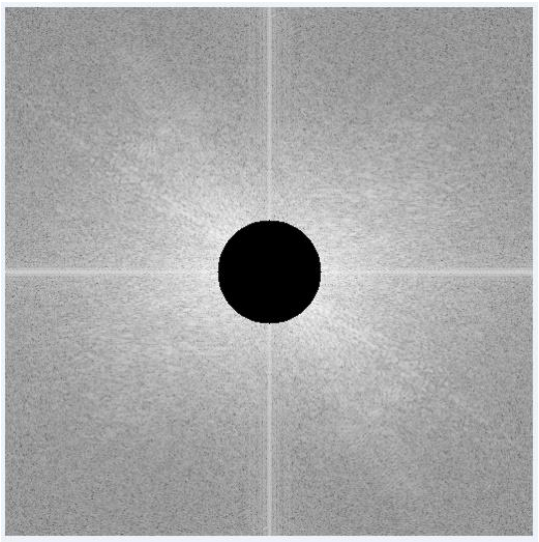
$$H_{HP}(u,v) = 1 - H_{LP}(u,v)$$

Convoluting highpass filter with input image returns edges or high frequency values from the image. I have recalled the function created for lowpass mask filter and subtracted 1 from it.

Output of the ideal highpass filter:



DFT of the input image



DFT of the filtered image

Filtered image

Butterworth lowpass filter:

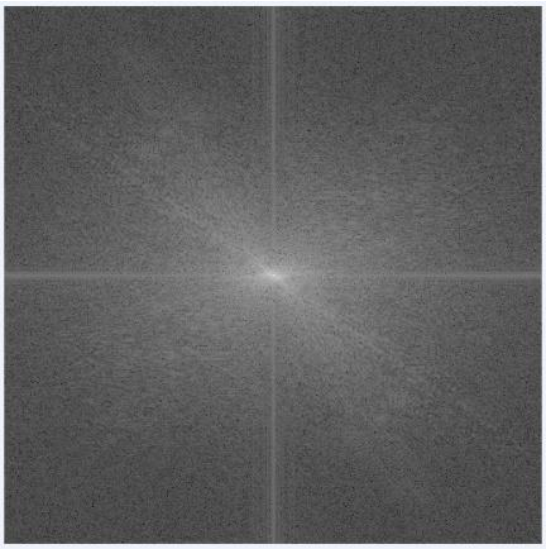
For computing the butterworth lowpass filter mask, I have used the formula below,

Butterworth Lowpass Filters (BLPF) of order n and with cutoff frequency D_0

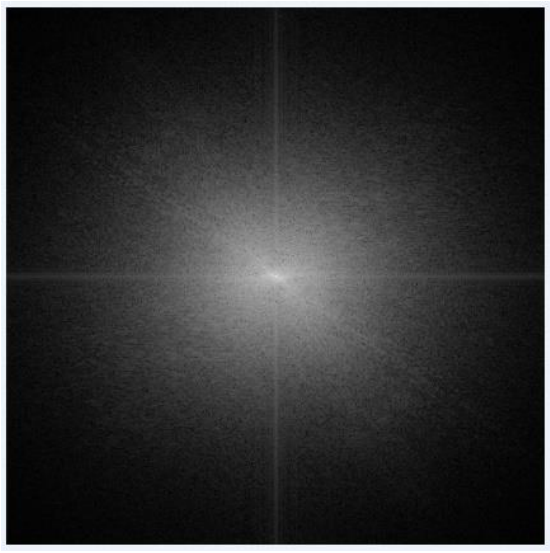
$$H(u,v) = \frac{1}{1+\left[D(u,v)/D_0\right]^{2n}}$$

Where, n is the order of the filter and D0 is the cut-off frequency. As the order of butterworth filter increases, we can see a ringing effect in the filtered image.

Output of the butterworth lowpass filter (order 2):



DFT of the input image



DFT of the filtered image

Filtered image (order 2)

Filtered image (order 70)

As we can see, right bottom image of lenna has ringing effect as order is 70.

Butterworth highpass filter:

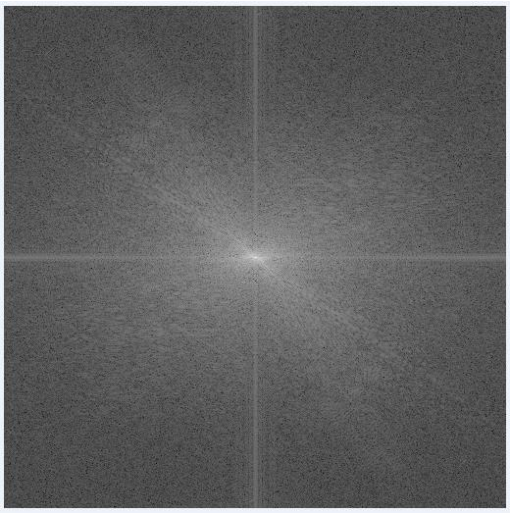
For computing the butterworth highpass filter mask, I have used the formula below,

A 2-D Butterworth highpass filter (BHPL) is defined as

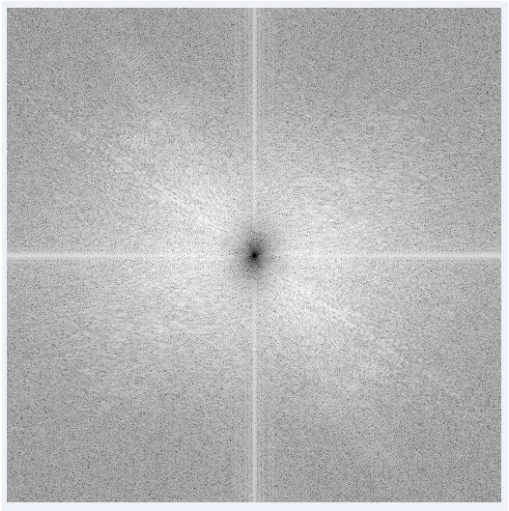
$$H(u,v) = \frac{1}{1+\left[D_0 / D(u,v)\right]^{2n}}$$

Where, n is the order of the filter and D0 is the cut-off frequency.

Output of the butterworth highpass filter:



DFT of the input image



DFT of the filtered image

Filtered image (order 2)

Filtered image (order 50)

Again, if we can see, right bottom image of lenna has ringing effect as order is 50.

Gaussian lowpass filter:

For computing the gaussian lowpass filter mask, I have used the formula below,

Gaussian Lowpass Filters (GLPF) in two dimensions is given

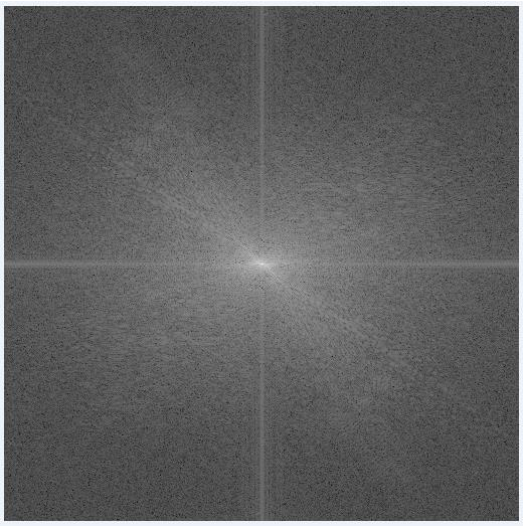
$$H(u,v) = e^{-D^2(u,v)/2\sigma^2}$$

By letting $\sigma = D_0$

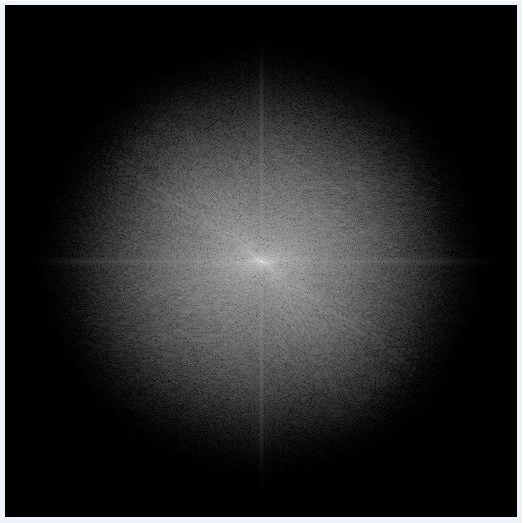
$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

where, D0 is the cut-off frequency. After reading through some research papers, I got to know the PSNR and MSE values for each filter mentioned in this assignment. For gaussian filter, PSNR value is higher and MSE value is lower. Hence, gaussian is best filter out of ideal, butterworth and gaussian filter for filtering effects. Additionally, we can see here as well that gaussian lowpass filter gives smoothest image.

<u>Output of the gaussian lowpass filter:</u>



DFT of the input image



DFT of the filtered image

Filtered image

Gaussian highpass filter:

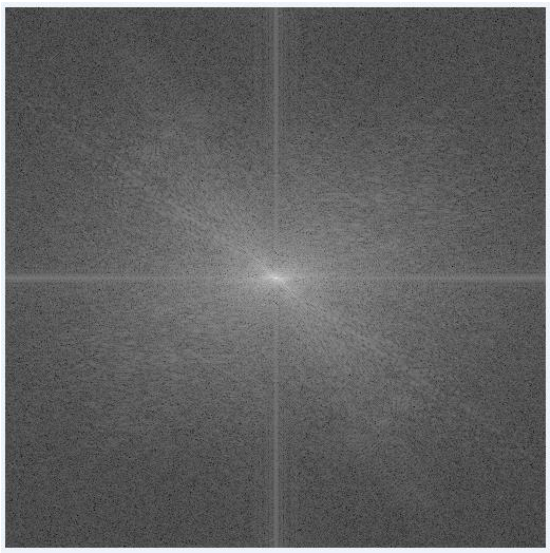
For computing the gaussian highpass filter mask, I have used the formula below,

A 2-D Gaussian highpass filter (GHPL) is defined as

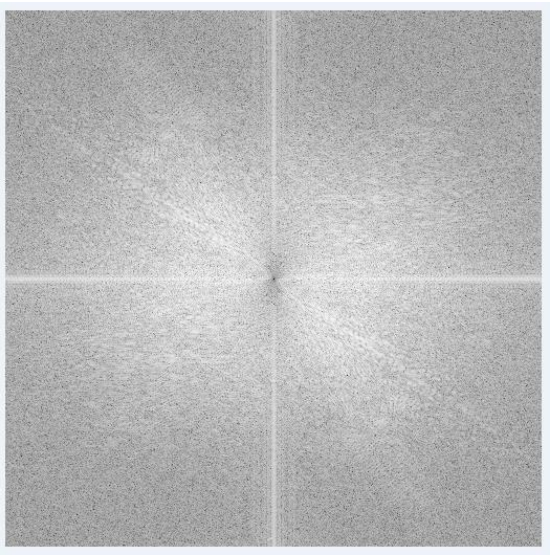
$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

where, D0 is the cut-off frequency.

Output of the gaussian highpass filter:



DFT of the input image



DFT of the filtered image

Filtered image

Problem faced:

1. In post_process_image() function, I was getting an error of division by zero when d and c was equal. So, I added the condition to handle the case and it solved the problem.
2. For butterworth and gaussian filters, I created an empty image with dtype as int. It was not reflecting the respective masks and filtered image was incorrect. Since, the values can be non-integer, I removed the dtype assignment which resulted into proper mask for both the filters.
3. To handle the situation for order of butterworth filter, I added the condition in filtering() function.
4. In butterworth filter function, the distance between a point and center position was becoming zero which was raising an error. I added a condition to solve the division by zero problem.
5. In butterworth filter function, for highpass filter, when distance was getting zero, I used much smaller value for computation. But, for higher order, above 20, function was giving an error. Hence, I changed the condition by making value zero. Because, according to the formula, for distance equal to zero, output is zero as well.

Command to run the functions mentioned above,
python dip_hw3_filter.py -i image -m ideal_l -c 50 -o 2

where, -i image = name of the image, Lenna0.jpg
-m filter_name = name of the filter, ['ideal_l', 'ideal_h', 'butterworth_l', 'butterworth_h', 'gaussian_l', 'gaussian_h']
-c cutoff = cut-off frequency
-o order = order of the filter (only valid for butterworth filter)