Project 4: Discrete Fourier Transform and Filters

Daniel Sawyer U3363-7705
CAP5400 Image Processing Fall 2019
University of South Florida, Tampa

I. 1. Introduction and overall description

In this assignment we use Discrete Fourier Transform in order to get frequency domain for an image which we can use to setup filters based on the frequencies. Once it is in the frequency domain, you can easily run low-pass filter, high-pass filter, notch filter, band-pass filter, and combinations of the filters.

II. Description of algorithms

In this section the basic algorithms used in this assignment are described. Since we used DFT from OpenCV, this will not focus as much on DFT as the filters we had to create.

A. 2.1 OpenCV DFT Implementation

OpenCV contains a Discrete Fourier Transform function as well as an Inverse Discrete Fourier Transform that we use. Here is a link to a tutorial on how it works and how to use it https://docs.opencv.org/2.4/doc/tutorials/core/discrete_fourier_transform/discrete_fourier_transform.html. After converting, we have two channels the real and complex. We then swap the quadrants to concentrate the high frequencies in the center and the low frequencies towards the borders. This creates a circular filter window inside the region of interest.

B. 2.2 Low Pass Filter

Using euclidean distance by pixel location X and Y, along with the high frequencies being concentrated in the center and is circular, from the center of the region of interest we can filter it remove low frequencies causing a blurring effect to remove noise. We create a mask where: If Distance ¡= D0, mask(i, j) = 1, else mask(i, j) = 0. After creating the mask we apply it to the real and complex planes and run IDFT to convert it back to an image.

C. 2.3 High Pass Filter

Using euclidean distance by pixel location X and Y, along with the high frequencies being concentrated in the center and is circular, from the center of the region of interest we can filter it remove high frequencies causing an edge detecting effect. We create a mask where: If Distance ¿= D0, mask(i, j) = 1, else mask(i, j) = 0. After creating the mask we apply it to the real and complex planes and run IDFT to convert it back to an image.

D. 2.4 Notch Filter

Using euclidean distance by pixel location X and Y, along with the high frequencies being concentrated in the center and is circular, from the center of the region of interest we can filter it removing a ”Notch” of frequencies within a range D1 to D2 which removes spectral noise. We create a mask where: If Distance ¿= D1 and Distance ¡= D2, mask(i, j) = 0, else mask(i, j) = 1. After creating the mask we apply it to the real and complex planes and run IDFT to convert it back to an image.

E. 2.5 Band Pass Filter

Using euclidean distance by pixel location X and Y, along with the high frequencies being concentrated in the center and is circular, from the center of the region of interest we can filter it by keeping a ”Band” of frequencies within a range D1 to D2 which enhances edges. We create a mask where: If Distance ¿= D1 and Distance ¡= D2, mask(i, j) = 1, else mask(i, j) = 0. After creating the mask we apply it to the real and complex planes and run IDFT to convert it back to an image.

F. 2.6 Combination of Filters

We can also combine filters to help enhance one another. Good filters to combine are low-pass and notch, as well as high-pass and band-pass. Low pass and notch both remove noise and high-pass and band-pass both work on edges. Using these in combination help to enhance one another depending on your goals with the image.

III. 3.Description of implementation

The entire code is developed in C++ language on Ubuntu 18.04.3 and USF's FSPrime linux server. The code for reading and writing the ppm and pgm image files is provided as part of the image class file. The utilities class file contains all 4 process that were described in this report above. The main is in main.cpp and reads the parameters file, which contains all the info on the input image, output image, and process to run on the image with their ROIs. It then saves the resulting image(s).

IV. 4.Description and Analysis of Results

Below are images created from the filters on the frequency domain of the images.

A. 4.1. Description of Results

Fig. 1. Baboon Magnitude Before

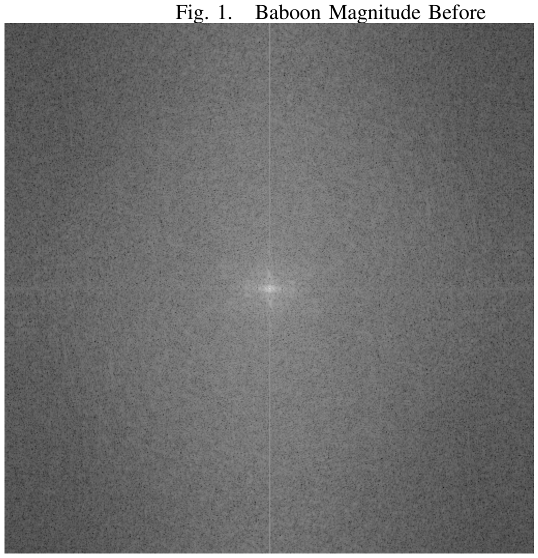


Fig. 2. Low Pass Filter Baboon

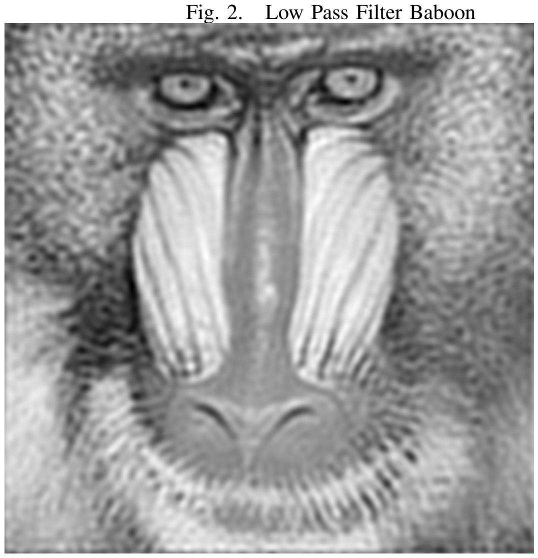


Fig. 3. Low Pass Magnitude After

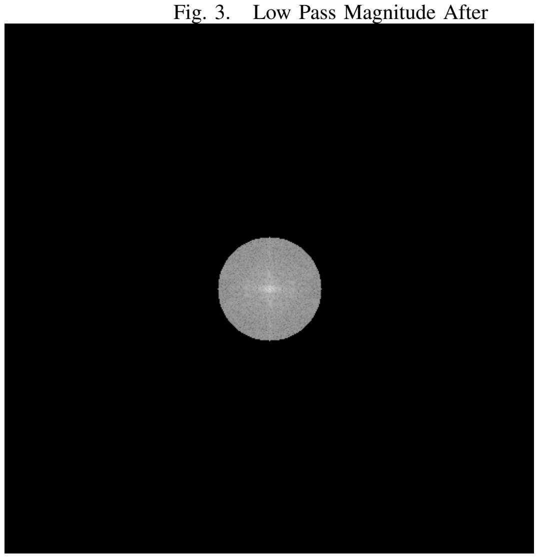


Fig. 4. Baboon Magnitude Before

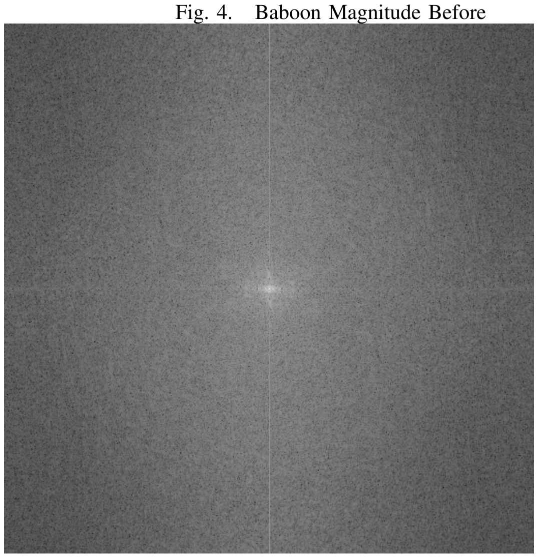


Fig. 5. High Pass Filter Baboon

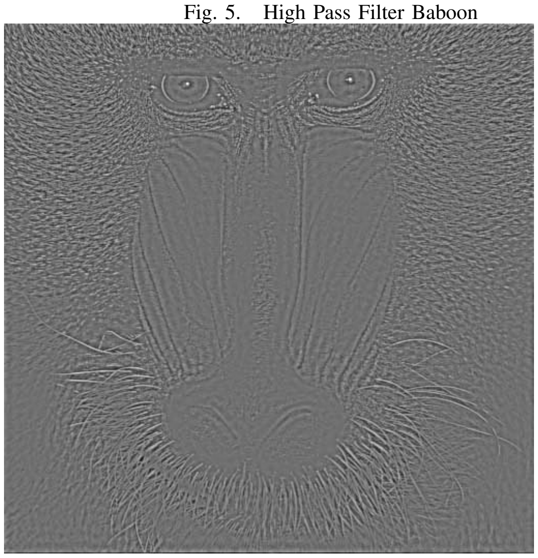


Fig. 6. High Pass Magnitude After

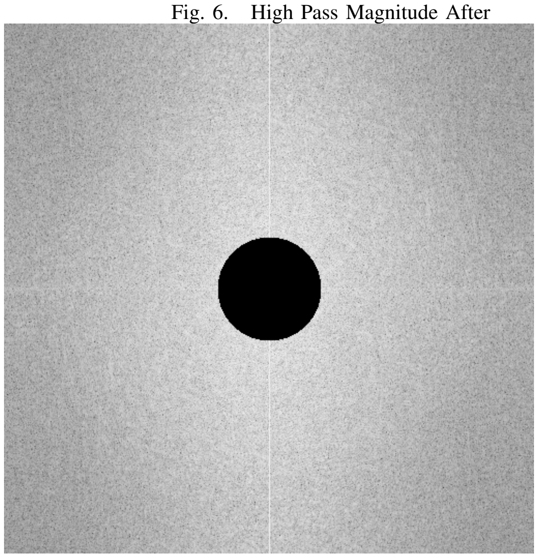


Fig. 7. Baboon Magnitude Before

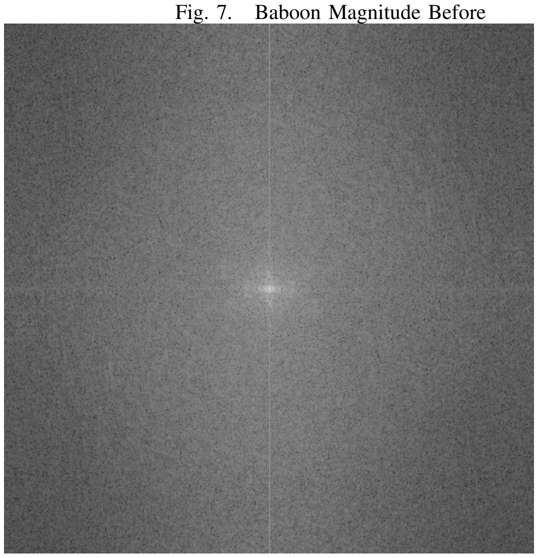


Fig. 8. Notch Filter Baboon

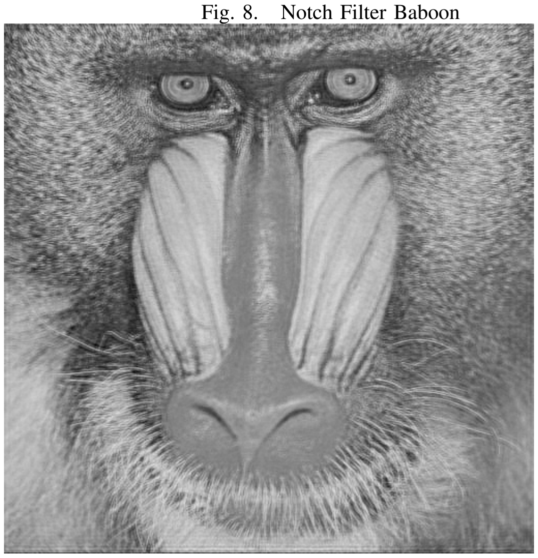


Fig. 9. Notch Magnitude After

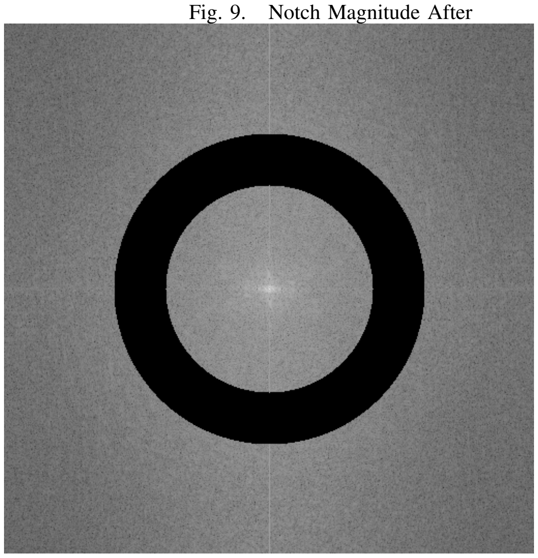


Fig. 10. Baboon Magnitude Before

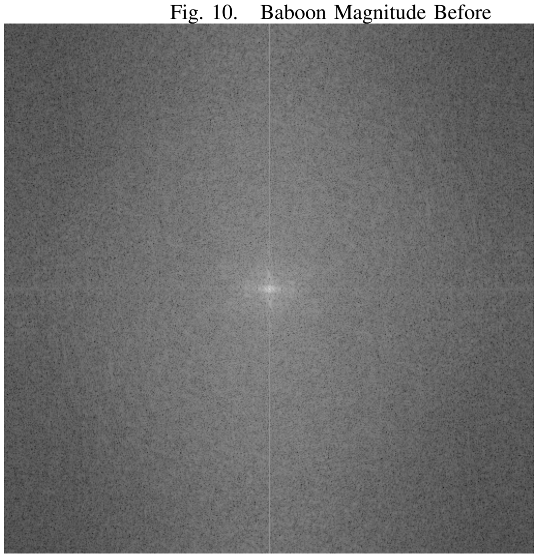


Fig. 11. Band Pass Filter Baboon

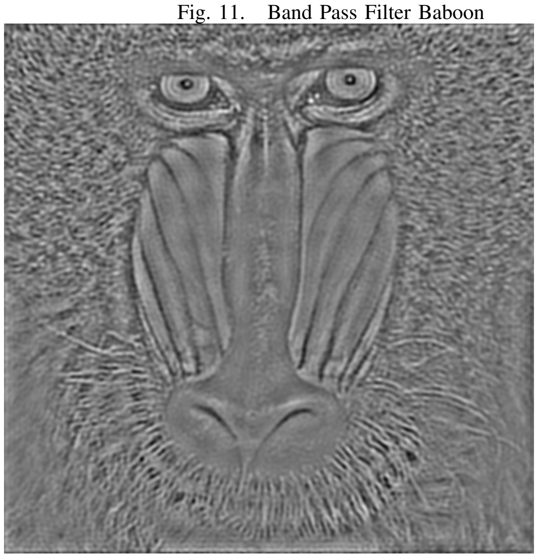


Fig. 12. Band Pass Magnitude After

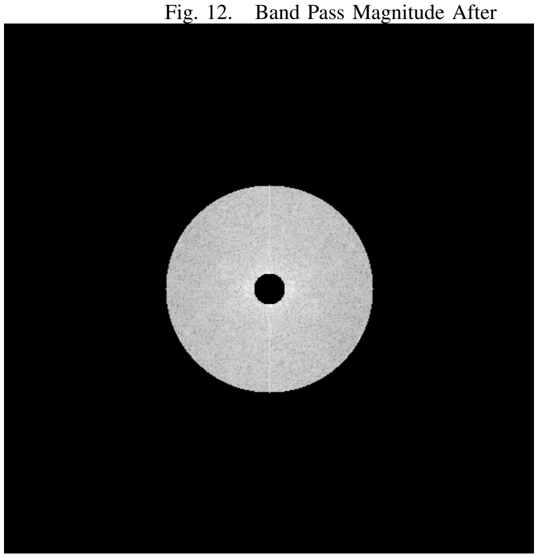


Fig. 13. Baboon Magnitude Before

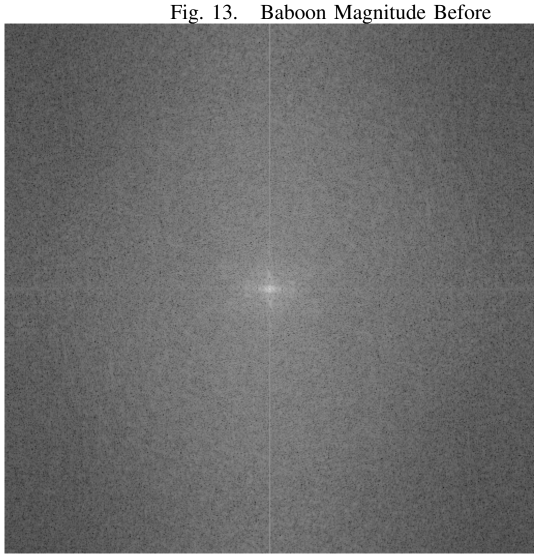


Fig. 14. Low Pass and Notch Filter Baboon

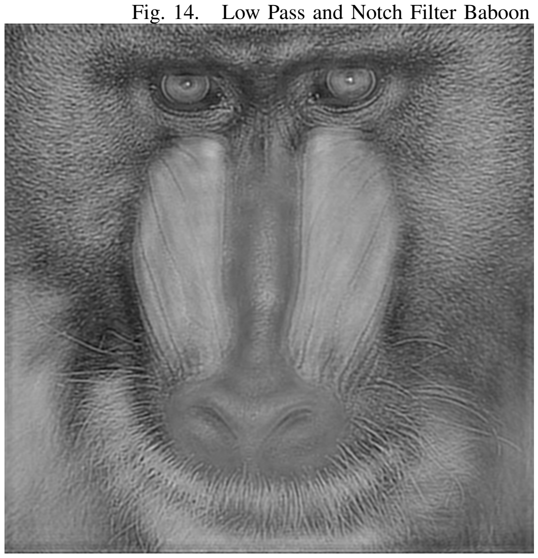


Fig. 15. Low Pass and Notch Magnitude After

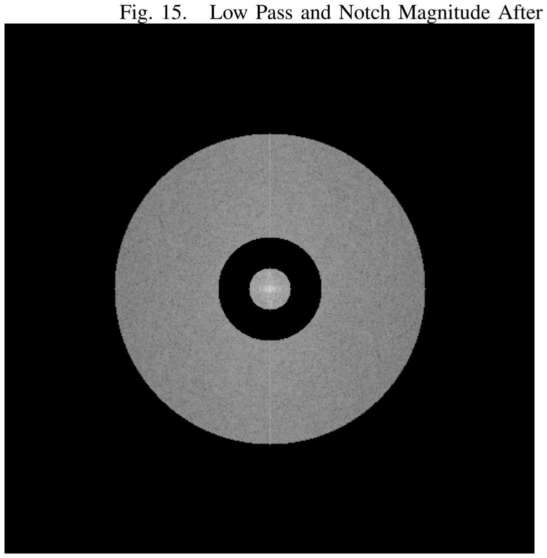


Fig. 16. Baboon Magnitude Before

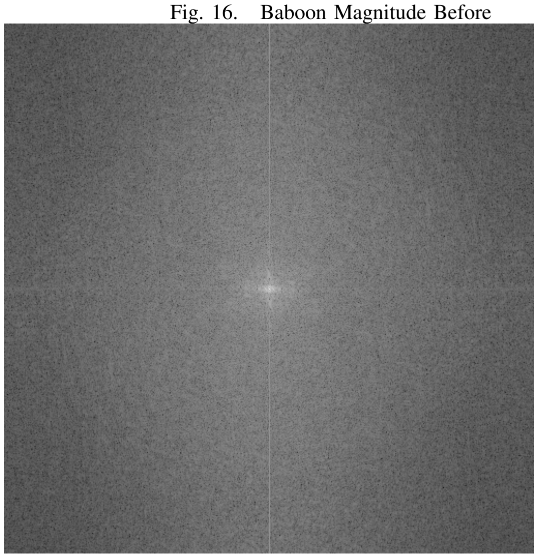


Fig. 17. High and Band Pass Filter Baboon

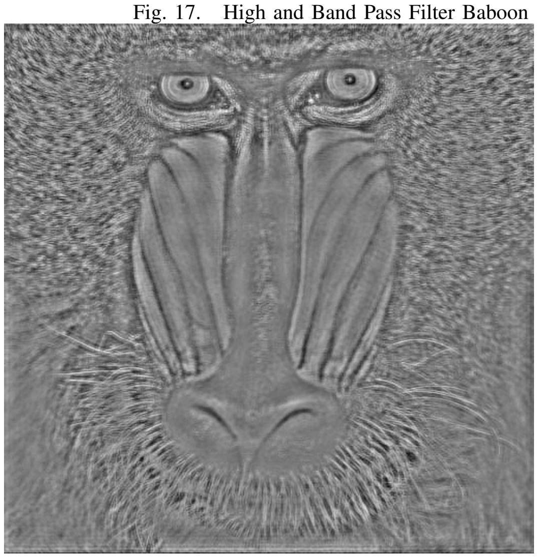
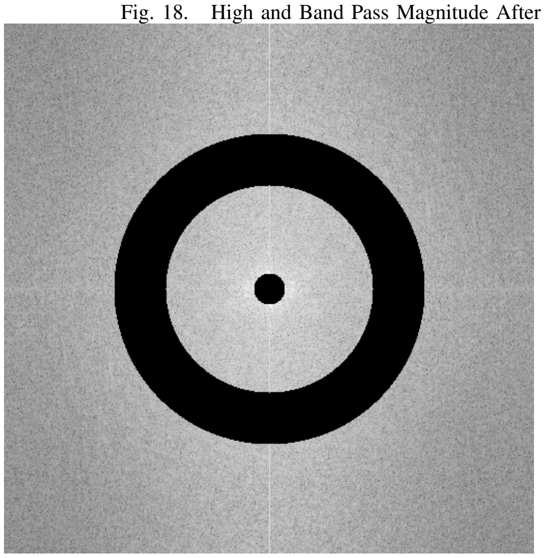


Fig. 18. High and Band Pass Magnitude After



B. 4.2. Performance Evaluation and Analysis of Results

DFT runs at O(N^2) as long as the image size is a power of 2. The filters take the number of pixels to run and IDFT also takes O(N^2) as long as the size of image is power of 2.

V. 5.Conclusion

This assignment shows how to do convolutions using frequency domain which does not need a sliding window and is reversible, where sliding window convolutions are not. It amazes me how you can do smoothing/blurring and edge detection by just eliminating certain frequencies and the fact that you can reverse the convolutions with a fair amount of ease.

References

[1] Example Project Code and Example Report from TA website.

[2] OpenCV Documentation and Tutorials https://docs.opencv.org/2.4/doc/tutorials/core/discrete_fourier_transform/discrete_fourier_transform.html