Teaching Assistant: Sherlon Almeida **Email:** sherlon@usp.br

Teaching Assistant: Bruno Resende **Email:** messias@ifsc.usp.br

Try to code the assignment by yourself. Plagiarism is not tolerated

Assignment 3
Filtering in Spatial and Frequency Domain

Problem Statement

In this assignment, you have to implement functions in order to filter images in the frequency domain using the Discrete Fourier Transform 2D. Read the instructions for each step. Use python with the **numpy** and **imageio** libraries.

Your program must have the following steps:

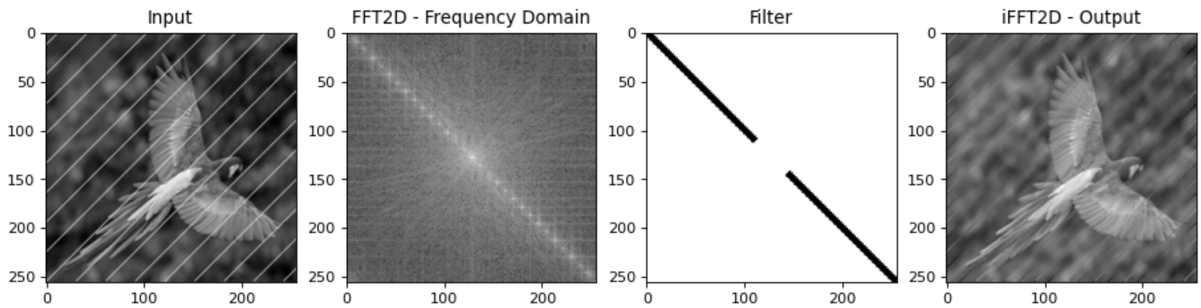
1. **Parameters input:**
   a. Filename for the input image ($\mathbf{I}$),
   b. Filename for the filter ($\mathbf{M}$),
   c. Filename for the reference image ($\mathbf{G}$).
2. **Generate the Fourier Spectrum** ($\mathbf{F(I)}$) for the input image $\mathbf{I}$.
3. **Filter** $\mathbf{F(I)}$ multiplying it by the input filter $\mathbf{M}$.
4. **Generate the filtered image** ($\hat{\mathbf{I}}$) **back** in the space domain.
5. **Compare the output image** ($\hat{\mathbf{I}}$) **with the reference image** ($\mathbf{G}$).

Discrete Fourier Transform 2D (DFT 2D)

Performing a transformation in the original domain of an image using DFT 2D changed the way of processing images. Processing images in the frequency domain is faster than its version in the spatial domain, and also it allows us to identify noisy frequencies that can be filtered easily.

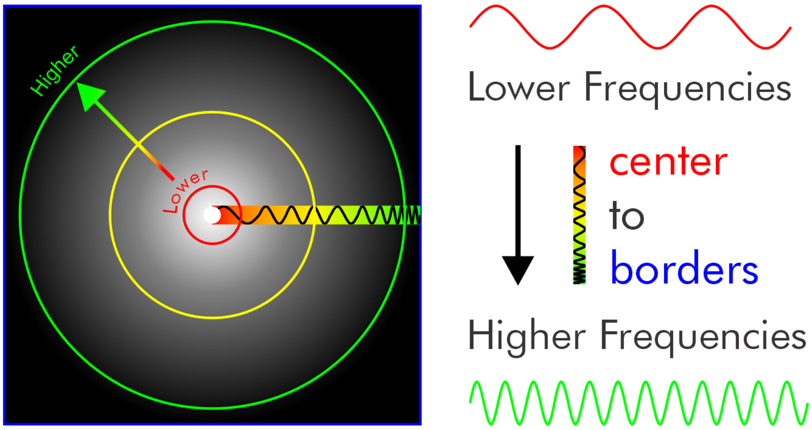
Figure 1 illustrates the approach you have to implement. Given an image as input, firstly you must generate its Fourier spectrum, which will be multiplied by a given binary filter to obtain a filtered image back in the spatial domain.

Figure 1. Step-by-Step to implement in this assignment.



Initially, it is essential to understand the concepts behind the Fourier transformation and the spectrum analysis. Figure 2 illustrates the distribution of **higher** (far from the center) and **lower** (close to the center) frequencies.

Figure 2. Fourier Spectrum illustration.

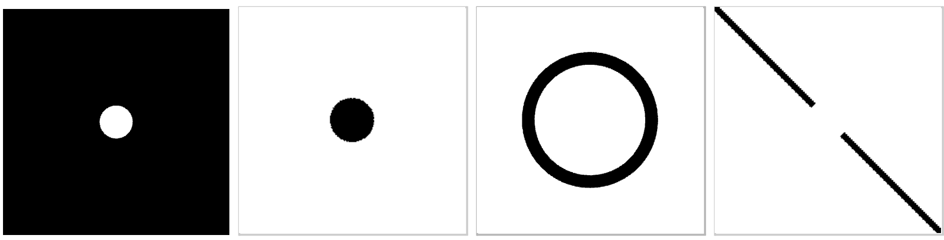


To generate the fourier spectrum it is highly recommended to implement your own functions for **DFT2D(...)**, **DFT_shift(...)** and **IDFT2D(...)**. However, you can use the numpy implementations, **np.fft.fft2(...)**, **np.fft.fftshift(...)** and **np.fft.ifft2(...)**.

Filters

Once you will implement an approach to process images in the **frequency domain** using the Discrete Fourier Transform 2D, you will need to use well-known filters, such as: **low pass** (Fig.3 (a)), **high pass** (Fig.3 (b)), and **band-stop** filters (Fig.3 (c) and (d)).

Figure 3. Example of some filters provided for this assignment.

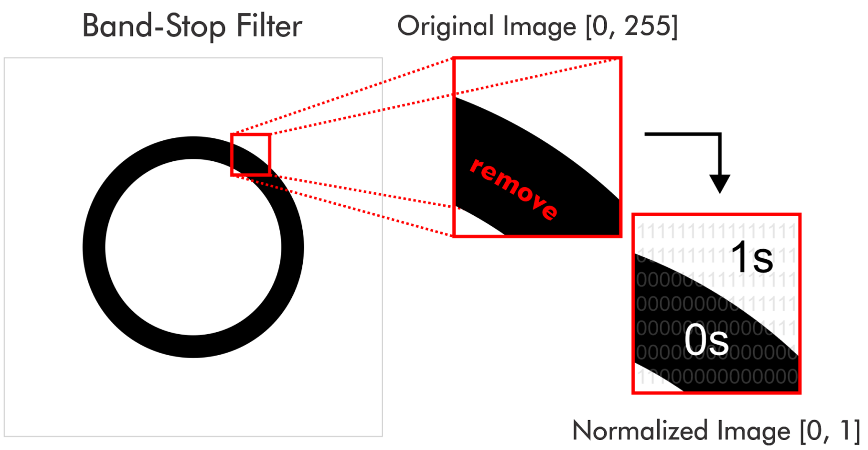


(a) (b) (c) (d)

It is important to remember that the same result of a convolution in the spatial domain can be obtained by a simple multiplication in the frequency domain. Thus, the provided filters have the same dimensions as the input images. Also, the filters must be represented by 0s and 1s in order to “remove” and “keep” the desired frequencies. For example, when a frequency is multiplied by zero, it will be removed, while the frequency multiplied by one is kept. An example of the format of the filter is shown in Figure 4.

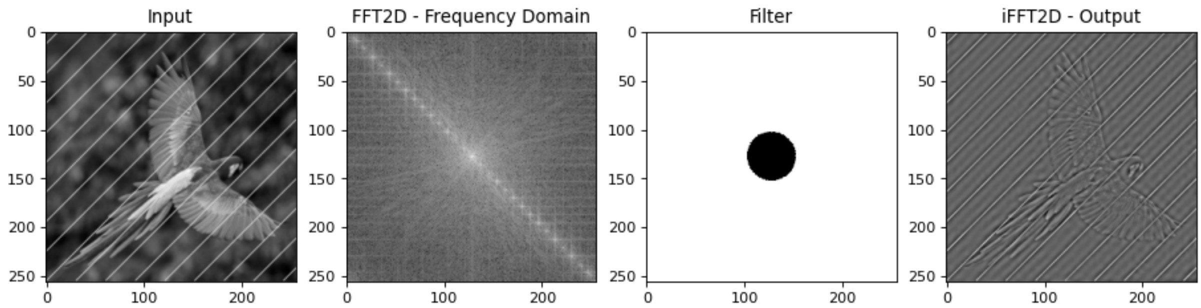
Thus, it is important to normalize the input filters from the [0, 255] interval to [0, 1] before performing the multiplication in the frequency domain. To perform the multiplication, you can use the **np.multiply(...)** function between F($\mathbf{I}$) and the filter $\mathbf{M}$.

Figure 4. Filter representation.



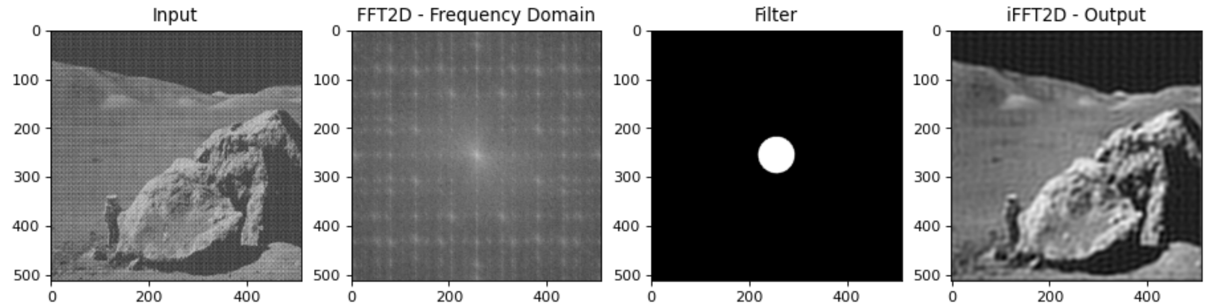
Each filter can capture different patterns from the images. Usually, higher frequencies represent the details of images, for example, the borders that highlight transitions of elements. Thus, applying a high-pass filter to the output image preserves these transitions. The Figure 5 output shows us basically the edges of the original image.

Figure 5. Applying a high-pass filter.



On the other hand, low frequencies represent the "heart" of the image, however with smooth changes between the neighbor pixels. Thus, applying a low-pass filter the output image in Figure 6 partially removes the noise, at the same time blurs the result.

Figure 6. Applying a low-pass filter.



Comparison with the reference image

After generating the output image $\hat{\mathrm{I}}$, compare it with the reference image **G** using RMSE. Since **G** has values between 0 and 255, you must normalize $\hat{\mathrm{I}}$ so that it also has values in the same range in the **uint8** format.

$$\text{RMSE} = \sqrt{\frac{1}{MN}\Sigma\Sigma(G(i,j) \; - \; \hat{\mathrm{I}}\,(i,j))^2}$$

Where M x N is the size of the image.

Input and Output

Example of input:

Input image ($\mathbf{I}$), input filter ($\mathbf{M}$) and reference image ($\mathbf{G}$).

Input	apollo17.png apollo17_filter1.png apollo17_filter1_ref.png

Example of output:

RMSE value in float format with 4 decimal places.

Output	0.7079

Submission

Submit your source code using the Run.Codes (only the .py file)

1. **Comment your code.** Use a header with name, USP number, course code, year/semester and the title of the assignment. A penalty on the grading will be applied if your code is missing the header and comments.
2. **Organize your code in programming functions.** Use one function per method.

Contact

If you have any questions, contact us by sending an email following the five steps below:

1st step: Include **BOTH** emails, sherlon@usp.br **and** messias@ifsc.usp.br.
2nd step: Include the subject **exactly** like this:
Subject: "**[Digital Image Processing 2022 | sem1] - Assignment 1**"
Do not change the initial part (**black**).
Replace the final part with the topic you are interested in (**red**).
3rd step: Add your personal information to help us find your submissions in Run.Codes and E-Disciplinas quickly.
4th step: Formulate your question in detail. Include your implementation and/or screenshots if necessary.
5th step: Send email and wait. We will respond as soon as possible.

Example of Email: