3. Frequency Filtering

Abstract
Working in the frequency domain can be less intuitive than doing it in the spatial domain, but it offers some benefits in some situations. On the one hand, some operations such as convolutions, can be performed more efficiently in some cases, thanks to the convolution theorem. On the other hand, some filter types can be designed or applied more naturally in the frequency domain. In this lab we will learn to apply, manipulate and visualize the Fourier Transform on images; we will check experimentally the convolution theorem and will become familiar with it; and then we will design and apply simple filters in the frequency domain.

Keywords
Fourier Transform • Convolution Theorem • Low-pass, band pass and high-pass filters

Contents

1. Fourier Transform

There is no difficulty in applying the Fourier Transform (FT) using some adequate library, beyond identifying the functions and little more. There are several Python libraries supporting the FT; we will use NumPy's module fft.

It is important, however, to bear in mind that the Fourier Transform works on complex numbers. Although computations should use the complex numbers, visualising the Fourier transform of an image usually relies on using some (simplified) representation of such numbers; the *magnitude* of the complex numbers and, perhaps, their phase, are commonly used.

In the case of the magnitude, it is its logarithm that is generally visualised to deal with the wide range of values of the magnitudes at different frequencies. On the other hand, the order of frequencies in the FT corresponds to the indices of the spatial coordinates. Therefore, the transform for frequency $(u,v) = (0,0)$ will be located in the first row and first column of theresult (i.e. the upper-left corner in the usual graphical representations of images and matrices). However, for the visualisation of the FT, it is common practice to locate the low frequencies in the center of the image. To achieve this, shifts are performed, e.g. with fftshift(). Do not forget that when we apply the inverse FT (IFT), the origin of frequencies is assumed to be at $(0,0)$. This implies that we need to "undo the shift" if required, with ifftshift() before the actual IFT.

▷ E_1

Now, make sure you understand the provided code; then, display the direct and inverse FT of several images, and try to change some aspect of this process. For instance, what happens if you apply `fftshift()` but not `ifftshift()`, or vice versa; how is the magnitude displayed if the logarithm is not applied?

2. Convolution Theorem

The convolution theorem states the equivalence between applying a filter through a convolution in the spatial domain and an element-wise product of their Fourier transforms. Concretely, if I is the image we want to filter, and M is the filter mask, then:

$$I * M = \mathscr{F}^{-1}(\mathscr{F}(I) \odot \mathscr{F}(M)),$$

where $*$ represents the convolution operation, $\odot$ is the element-wise product, and $\mathscr{F}$ y $\mathscr{F}^{-1}$ are the direct and inverse Fourier Transforms, respectively.

Run the code where this theorem is illustrated, and study the implementation. As usual, make sure you understand it, and would know how to implement a similar process.

▷ E_2, E_3

3. Frequency filters

In the previous section, to check the convolution theorem, we have applied the Fourier Transform to the filter defined in the spatial domain. However, in some cases, it can be more natural do define or design filters directly in the frequency domain.

For instance, when we smooth an image, we know that the information corresponding to small details is lost. Intuitively, this corresponds to "removing high frequencies" and "keeping the small ones". Seen in this way, it turns out quite straightforwardly how a filter in the frequency domain should behave. Such filter is known as low-pass filter. Since low frequencies in the frequency domain are located in the center of the array of the FT transform, we can produce an array (matrix) with *ones* in a central area, and *zeros* outside this area. With such filter (in the frequency domain) and through the convolution theorem, we know how to produce the filtered image, *without*

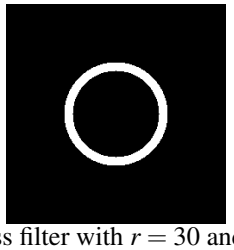

Figure 1. Band-pass filter with $r = 30$ and $R = 80$ for a 255×255 image

knowing what the corresponding filter looks like in the spatial domain.

With the code you are provided with, try the band-pass filter for different values of r, and interpret the results. As usual, review and study the implementation. Try also the high-pass filter and the band-pass filter (Fig. 1), and experiment with different values for r and R.

▷ E_7

4. Exercises

1. Given the Fourier Transform of an image, compute (a) the lowest and highest magnitude; (b) display a *boxplot* with the magnitude values; and (c) plot an histogram of the phase values. To display the *boxplot* and the histogram, you can use Matplotlib.

2. We have seen the convolution theorem applied to the mean filter. Now, apply it to the Gaussian filter and check the result is also as expected. In addition the (filtered) output image, display also the Fourier transform of the filter, and the element-wise product of the Fourier transforms of the image and the filter. Try to figure out the relationship between the size of the mean filter and its Fourier transform. Repeat this analysis on the standard deviation.

3. A practical implication of the convolution theorem is the computational saving it may offer by performing computations in the frequency domain instead of in the spatial domain. Carry out a study of the running times for differnt sizes of the mask filter, and for different image sizes. Display a plot of the measured times (mean and deviation) to appreciate the influence of the image size for a given mask size, and another plot where the image size is fixed and the mask size is varied. Observe the results carefully, and comment on them.

4. Write `my_mask(n)` without explicit loops that returns a 2D array of size $n \times n$ following this pattern:

$$\underset{n=3}{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}}, \quad \underset{n=4}{\begin{bmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 \end{bmatrix}}.$$

   Then, write `my_filter(im,n)` to use the output of `my_mask()` as a mask to filter image `im` in the frequency domain.

5. Get the Fourier transform of stp1.gif and stp2.gif. Let F_1 and F_2 be these transforms. Compute and observe the inverse transform of their combination $\lambda \cdot F_1 + (1 - \lambda) \cdot F_2, \lambda \in [0,1]$. What property of the FT does this result illustrate?

6. (Optional) Study whether the FT is distributive with respect to the product, namely, whether $\mathscr{F}(I_1 \odot I_2) = \mathscr{F}(I_1) \odot \mathscr{F}(I_2)$. You can use the images of the previous exercise to perform an empirical analysis.

7. (Optional) The frequency filters we have seen set to all (1) or nothing (0) so that the desired frequency ranges (either high, low or band) are selected or removed, with abrupt changes. Write smoothed versions of the filters so that transition between 0 to 1 is somehow *smooth*. For the same values of r and R, compare the abrupt and smooth filters and comment on the possible influence on the result.

8. (Optional) Besides its application to filtering, the Fourier Transform is useful in other contexts. This exercises illustrates one of them: *watermarking*. We can add water marks to one image without the mark being noticeable to the naked eye. We did it on image `lena255.pgm` and your task is to find out whether image `a.pgm` or image `b.pgm` (provided files) is the one with the watermark; the other image has just been added a litte bit of Guassian noise so that it is not exactly the same image as the original. To solve your task, you should consider, besides the candidate images `a.pgm` and `b.pgm`:

   - the process followed to generate the watermark (section Digital Watermarking whose first paragrah begins with "Now we turn to the main application…", in fourier-analysis);
   - the provided file `positions.p` with the points used; (variable `indices` in that page); and
   - the original image (`lena255.pgm`).

   Explain the process followed for your analysis and to get to your decision.

   By the way, strictly speaking, we would not require neither the original image nor the positions, to find out whether a given image contains a watermark. However, the procedure would be somehow more complicated, and it is not worthwhile in the context of this lab. This is why we have simplified the problem without, hopefully, making it less interesting.

   Note: you can load the points in `position.p` with the pickle module:

```
import pickle
locationsFile = "./imgs-P3/positions.p"
file = open(locationsFile, "rb")
indices = pickle.load(file)
```