Medical Physics: Image processing

Marc Parcerisa i Conesa

May 2025

1. BASES OF IMAGE ENCODINGS AND COLORS

The first part of this script focuses on loading an image, manipulating its histogram, and showing it on screen. We first load images into ram by using `plt.imread` and get familiar with pyplot's functions for displaying images on screen (`plt.imshow`). We also experiment with colored images and gray-scale images, leading to learning about parameters `cmap` and `vmin` and `vmax`, corresponding to changes on the color representation of gray-scale images, and the minimum and maximum values of the images (usually 0 to 255).

Colored images can be split into their three channels: Red, Green and Blue, as well as be converted into gray-scale images by using different algorithms. The most trivial calculation to transform a colored image into gray-scale is simply computing the average value over the three channels on every pixel:

$$GS_{ij} = \frac{1}{3}\left(R_{ij} + G_{ij} + B_{ij}\right)$$

Another option is to compute the luminosity of each pixel, weighting each channel with a different value, usually determined by previous studies. The most common formula reads:

$$L_{ij} = \frac{1}{3}\left(0.2126\, R_{ij} + 0.7152\, G_{ij} + 0.0722\, B_{ij}\right)$$

2. HISTOGRAMS

Histograms are one of the most useful tools for image processing. It consists of running over all the image and recording the times each intensity appears (this is usually done only for gray-scale images, or for a single channel of a colored one). `scipy.ndimage` offers a wide range of functionalities, such as histogram computation, which we've used all over the lab.

When observing an image histogram, one can determine whether the image is correctly equalized or not. For instance, if there is an abundance of darker tones and little to no lighter ones, it could mean that the image is too dim, and details on the image could be harder to see with the naked eye, and the opposite could mean that the image is burned and over-saturated. To fix this, one can simply change the value of the pixels to "move" the histogram up and down, rescale it, or even use more complex calculations. This is usually done by using look-up tables (LUT). For instance, the following histogram corresponds to a plain image with an abundance of lighter tones, and was moved left and rescaled to make the image clearer.

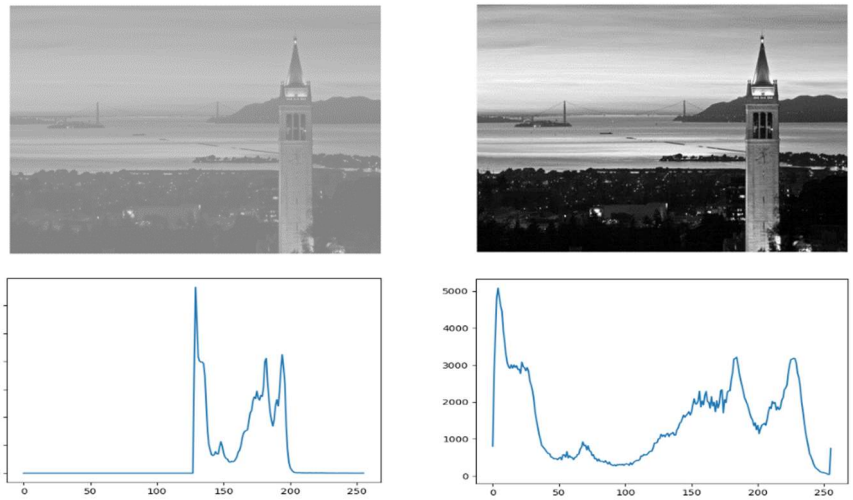
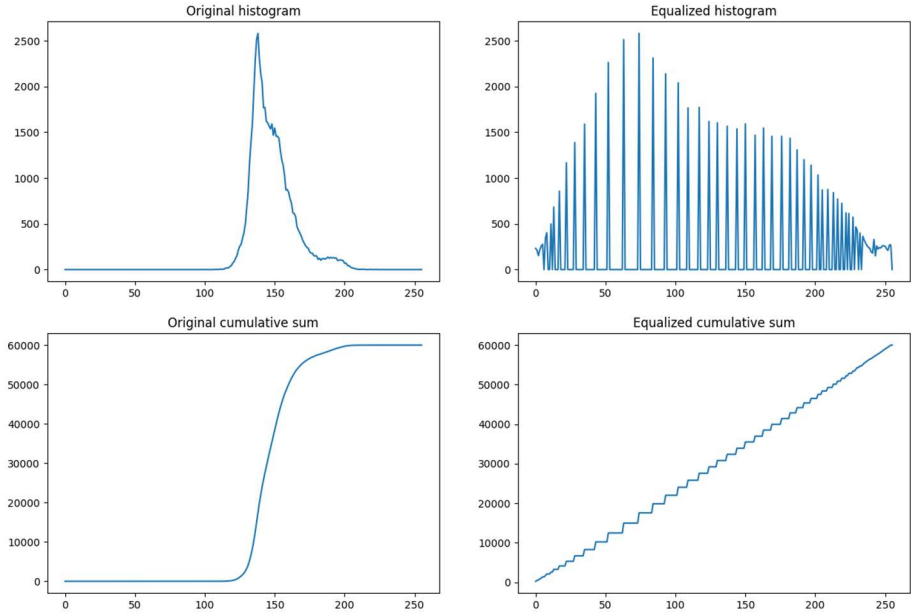


Image equalization is a histogram manipulation technique that tries to make it the most plain possible, meaning that all the intensities appear roughly the same amount of times. The equation for this looks as follows:

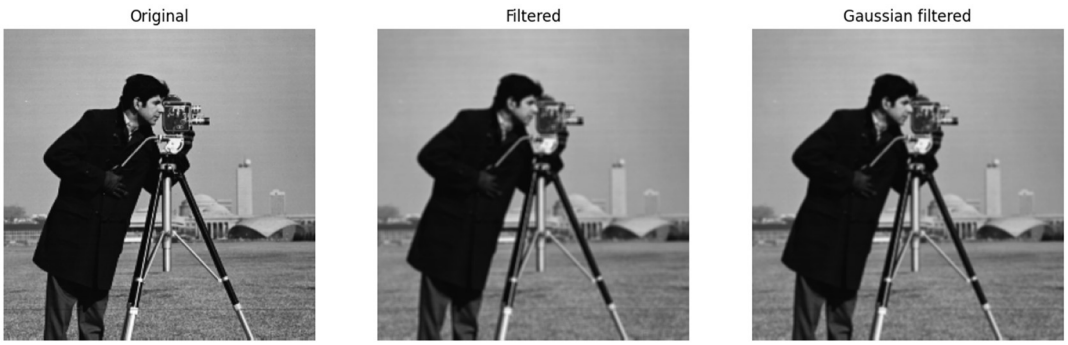
$$EQ_{ij} = \frac{1}{m \cdot n} CUMSUM(GS_{ij}) \quad \rightarrow \quad CUMSUM(x) = \sum_{j=0}^{x} HIST_j$$

This transforms the cumulative histogram of the image like the following:



3. FILTERS

Filters apply to the whole image and transform it in various ways, rather than simply changing the pixels intensities. Convolutional filters work by convolving a kernel over the entire image (for each pixel P, we align the kernel with that pixel on the image, multiply each of the pixels J on the kernel times their counterpart on the image P+J, add all the values that resulted from the multiplication, and set that pixel J to the result). These sort of filters are used for a tone of things, for instance, when convolving an image with a square kernel full of ones, one can blur out the original image, sort of like a low-pass image (removing the information about the edges and high frequency changes). This is most commonly achieved by using a gaussian-like kernel, or gaussian filter:



In contrast, high-pass filters can be applied to detect edges on the images. These can also be implemented via a convolution of a kernel with negative values. The following are directional high-pass kernels:

-1	0	1
-2	0	2
-1	0	1

Vertical edge detection

-1	-2	-1
0	0	0
1	2	1

Horizontal edge detection

-1	-1	-1
-1	8	-1
-1	-1	-1

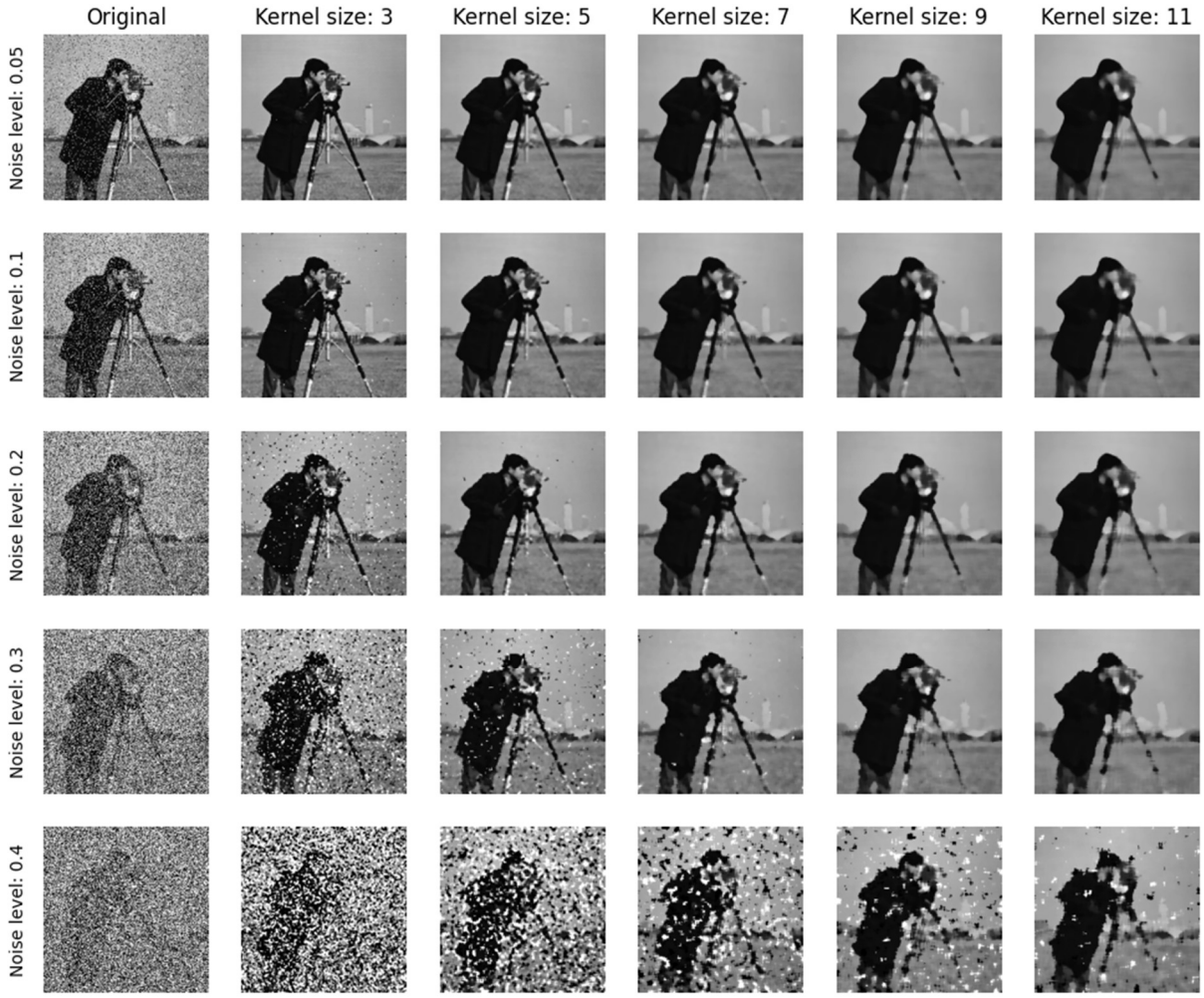
Omnidirectional edge detection

A. Median filter

This sort of filter cannot be implemented as a convolution, but is very useful for any device that captures data and is susceptible to impulse noise. Impulse noise, also known as salt and pepper noise, refers to sudden, random, and sporadic disruptions in a signal or image. It appears as isolated, randomly occurring pixels or points with significantly different intensity values compared to their neighboring pixels, resembling the appearance of grains of salt and pepper. To get rid of this type of noise, a commonly used algorithm consists of simply substituting each pixel value with the median value of its neighbors. Note how this method completely removes the salt and pepper noise.

Note also how, in comparison, a gaussian low-pass filter does not fix the image.

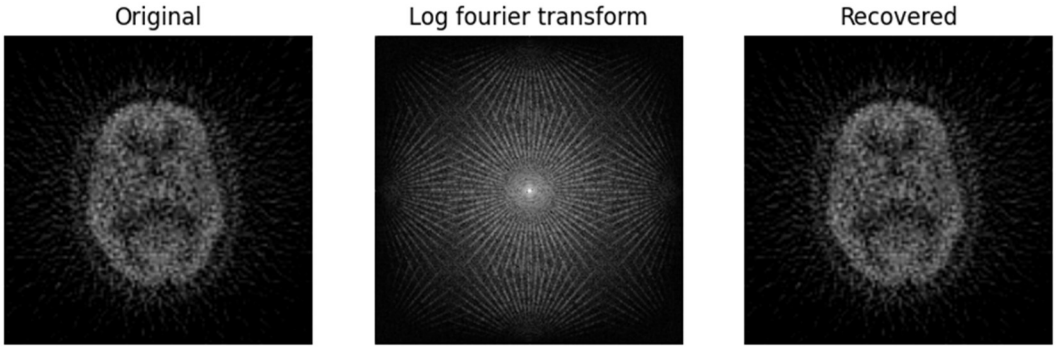
Here's a little study on how this filter responds to different noise amounts and kernel sizes:



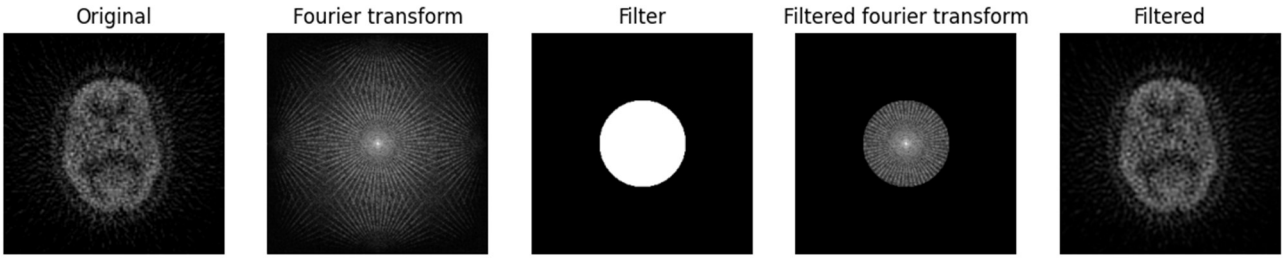
See the `.ipynb` file for an extensive study on how to apply these methods onto MRI images.

4. FOURIER TRANSFORMS

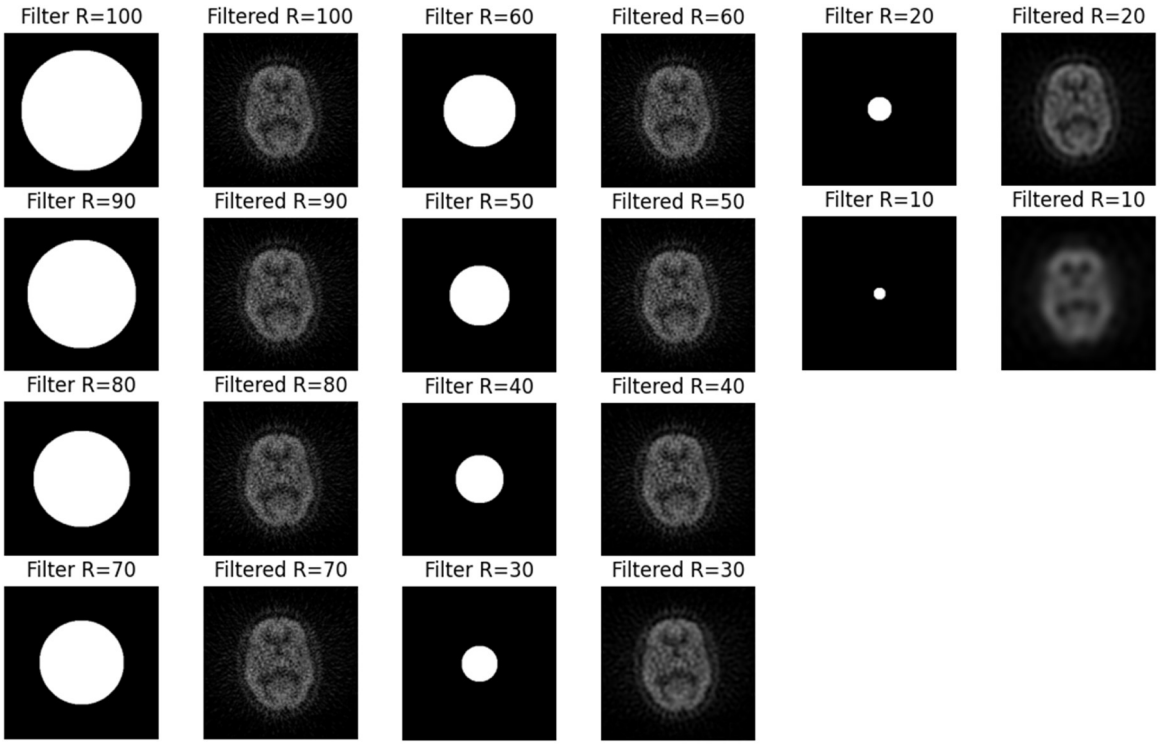
All of the convolutional filters can be applied also by analyzing the Fourier spectrum of the image. Numpy provides a *Fast Fourier Transform* library for obtaining and manipulating the image frequency spectrums. By using this library, one can, for example, extract the fft, and then recover the original image without any loss of information:



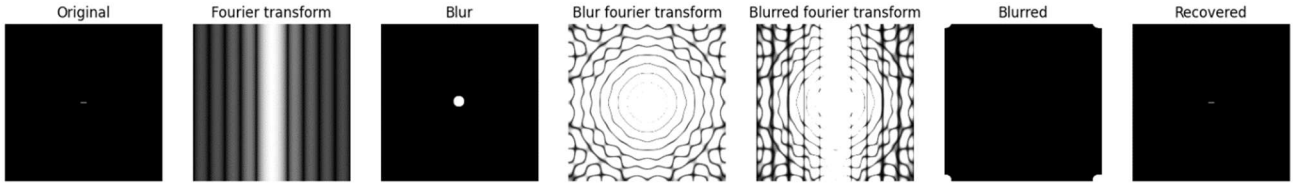
Note that low-pass filters on the Fourier space consist simply on maintaining the spectrum for frequencies smaller than a set value and cutting off the others, which is the same as multiplying the image by a circle of radius R filled with ones, and zeros everywhere else. This of course leaves only behind the lowest frequencies of the image, and recovering it back gives the same result as the convolutional low-pass filters:



In this method, decreasing the radius corresponds to an image with a higher amount of blur (or a bigger kernel size for the low-pass convolutional filters):



Fourier transform filters can also be used to reconstruct an image from a blurred one, given that we have a rough understanding of the noise and displacements that were present during the obtaining of it. For instance, in the following example, we simulated a blurred image by applying a filter on the fourier space. Then, we take the blurred image and apply another fourier transform, then dividing it by the model of blurring artifact we have. Then, reverting the fourier transform, one can recover the image.



I think the above example is incorrect, but it's a quite direct copy of the example provided in the reference code. Please refer the `.ipynb` file for this full example.