UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTE OF COMPUTING

INTRODUCTION TO DIGITAL IMAGE PROCESSING
MC920 / MO443

HOMEWORK 2

STUDENT: Gian Franco Joel Condori Luna
RA: 234826

1.- Fast Fourier Transform

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. [1]

1.1 Apply the FFT:

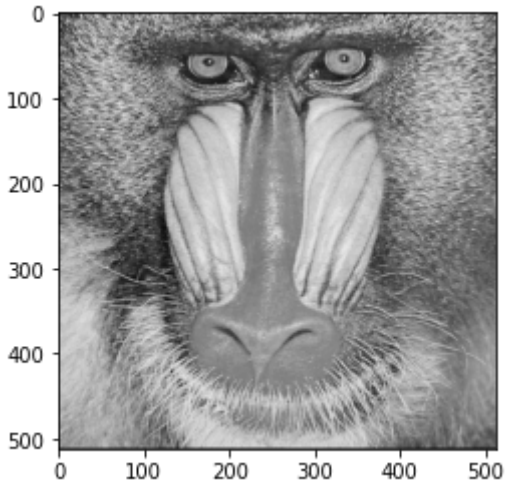
To be able to apply the FFT we first import the libraries that we are going to use:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

- Cv2 is an open source library for computer vision, image analysis and machine learning. For this, it has an infinite number of algorithms that allow, by just writing a few lines of code, to identify faces, recognize objects, classify them, detect hand movements. [2]
- Numpy is an open source Python library that introduces vectors and matrices in Python. It has numerous functions to work with these vectors and matrices. [3]
- Matplotlib is a plotting library used for 2D graphics in Python programming language, it is very flexible and has many built-in default values. [4]

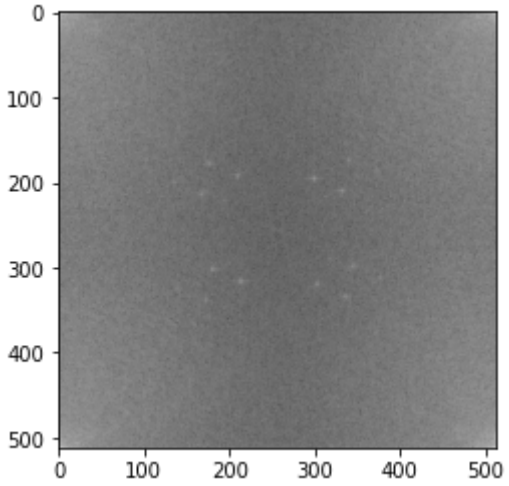
Then we read the image from the directory that was provided: “http://www.ic.unicamp.br/~helio/imagens_png/”, We read the image using "cv2.imread ()" this function reads the image.

```
img =
cv2.imread("/content/drive/MyDrive/imagenes/imagens_png/baboon.png", 0)
```



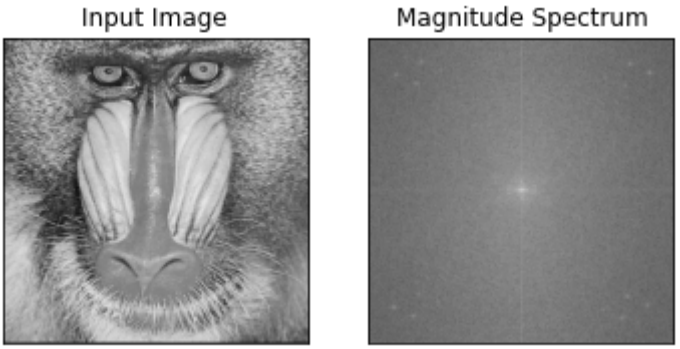
We use the function "np.fft.fft2 ()" from the numpy library, This function computes the *n*-dimensional discrete Fourier Transform over any axes in an *M*-dimensional array by means of the Fast Fourier Transform (FFT). By default, the transform is computed over the last two axes of the input array, i.e., a 2-dimensional FFT.[5]

```
f = np.fft.fft2(img)
```



As we can see, the result image does not have the spectrum in the center, so we use the "np.fft.fftshift ()" function to be able to locate the spectrum in the center of the image. This function swaps half-spaces for all axes listed (defaults to all). Note that y[0] is the Nyquist component only if len(x) is even.[6]

```
fshift = np.fft.fftshift(f)
```



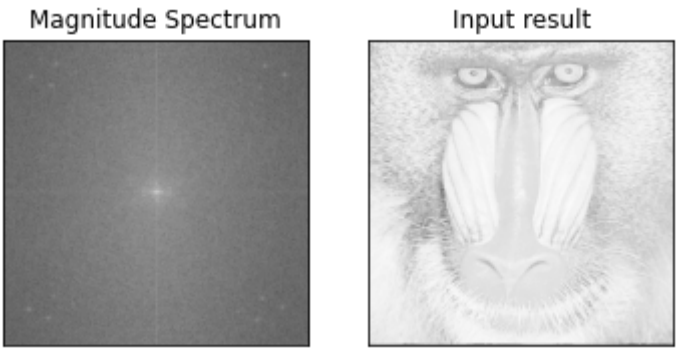
2.- Inverse Fast Fourier Transform (IFFT)

Is an algorithm to undo the process of DFT. It is also known as backward Fourier transform. It converts a space or time signal to signal of the frequency domain. The DFT signal is generated by the distribution of value sequences to different frequency component. Working directly to convert on Fourier transform is computationally too expensive. So, Fast Fourier transform is used as it rapidly computes by factorizing the DFT matrix as the product of sparse factors. As a result, it reduces the DFT computation complexity from $O(N^2)$ to O(N log N). And this is a huge difference when working on a large dataset. Also, FFT algorithms are very accurate as compared to the DFT definition directly, in the presence of round-off error.[7]
This transformation is a translation from the configuration space to frequency space and this is very important in terms of exploring both transformations of certain problems for more efficient computation and in exploring the power spectrum of a signal. This translation can be from xn to Xk. It is converting spatial or temporal data into the frequency domain data.[7]

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(x)e^{-ixt}\,dx$$

We use the function "`np.fft.ifft2` ()" from the numpy library, this function computes the inverse of the 2-dimensional discrete Fourier Transform over any number of axes in an M-dimensional array by means of the Fast Fourier Transform (FFT). In other words, ifft2(fft2(a)) == a to within numerical accuracy. By default, the inverse transform is computed over the last two axes of the input array.[14]

```
f_ishift = np.fft.ifftshift(fshift)
idft = np.fft.ifft2(f_ishift)
img_back = np.log(np.abs(idft))
img_back = np.where(img_back < 0, 0, img_back)
```



3.- Low Pass Filters (LPF)

A low-pass filter is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. The exact frequency response of the filter depends on the filter design. The filter is sometimes called a high-cut filter, or treble-cut filter in audio applications. A low-pass filter is the complement of a high-pass filter.[8]

3.1 Apply LPF: We create the function:

```
#Function Low Pass Filter
def LowPassFilter (magnitude_spectrum, radius):
 #Dimensions
 rows, cols = magnitude_spectrum.shape # (512, 512)
 #Point center the image
 crow, ccol = int(rows / 2), int(cols / 2) #(256, 256)
 #Mask of one
 mask_pb = np.ones((rows, cols), np.uint8) # (512, 512, 2)
 #Center the image
 center = [crow, ccol]
 x, y = np.ogrid[:rows, :cols] # 0 to 511
 #If they are outside
 the radius it is TRUE
 mask_area_pb = (x - center[0]) ** 2 + (y - center[1]) ** 2 > radius*radius
 #Zeroing everything TRUE
 mask_pb[mask_area_pb] = 0
 #Multiplying
 complex_pb = mask_pb*fshift
 result_pb = magnitude_spectrum*mask_pb

 return result_pb, mask_pb
```

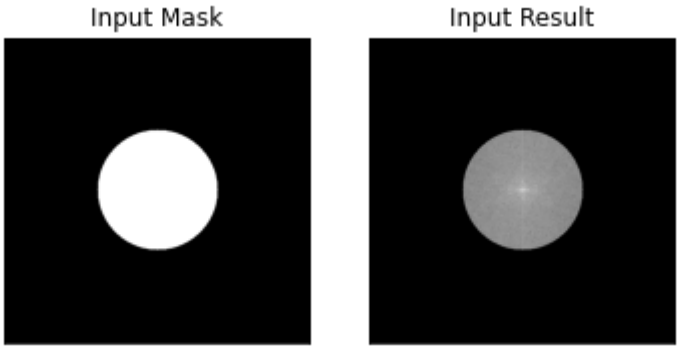


Image wuth radius 100:

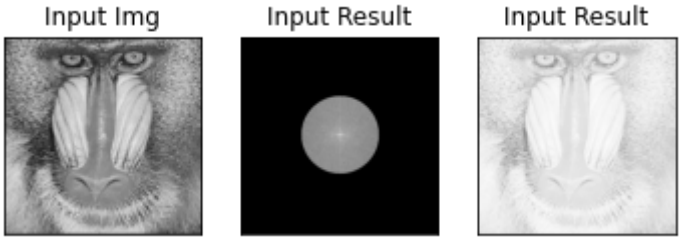


Image with radius 50

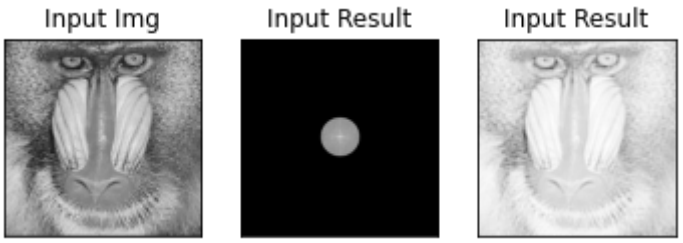
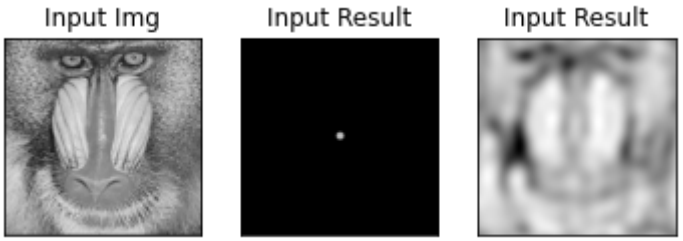


Imagen with radius 10



3.2 Conclusions:

As the radius is smaller the image begins to fade.

4.- High Pass Filters (HPF)

A high-pass filter (HPF) is an electronic filter that passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency. The amount of attenuation for each frequency depends on the filter design. A high-pass filter is usually modeled as a linear time-invariant system. It is sometimes called a low-cut filter or bass-cut filter in the context of audio engineering. High-pass filters have many uses, such as blocking DC from circuitry sensitive to non-zero average voltages or

radio frequency devices. They can also be used in conjunction with a low-pass filter to produce a bandpass filter. [9]

4.1 Apply HPF: We create the function:

```
#Function High Pass Filter
def HighPassFilter (magnitude_spectrum, radius):
 #Dimensions
 rows, cols = magnitude_spectrum.shape # (512, 512)
 #Center the image
 crow, ccol = int(rows / 2), int(cols / 2) #(256, 256)
 #Mask of one
 mask_pa = np.ones((rows, cols), np.uint8) # (512, 512, 2)
 #center
 center = [crow, ccol]
 x, y = np.ogrid[:rows, :cols] # 0 to 511
 #If they are within the radius it is TRUE
 mask_area_pa = (x - center[0]) ** 2 + (y - center[1]) ** 2 <=
radius*radius
 #Zeroing everything TRUE
 mask_pa[mask_area_pa] = 0
 #Multiplying
 complex_pa = mask_pa*fshift
 result_pa = magnitude_spectrum*mask_pa

 return result_pa, mask_pa
```

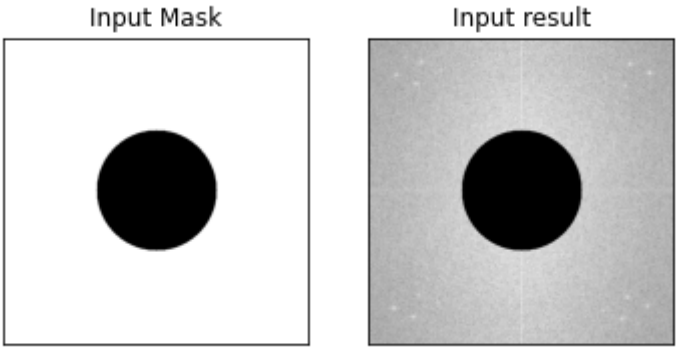


Image with radius 100

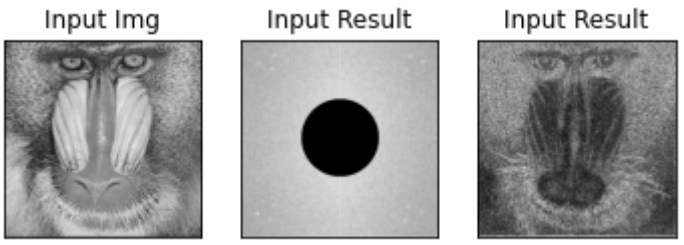


Image with radius 50

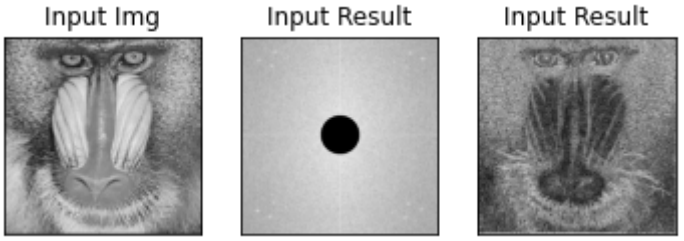
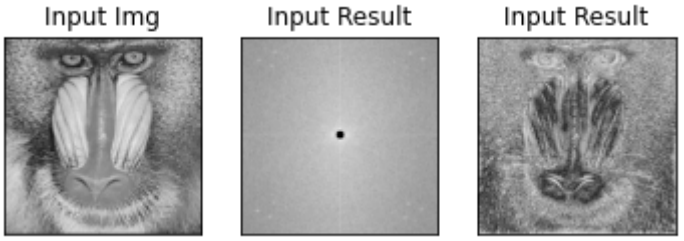


Imagen with radius 10



4.2 Conclusions:

As the radius is smaller, the edges of the image are more noticeable.

5.- Band Pass Filters (BPF)

A band-pass filter allows through components in a specified band of frequencies, called its *passband* but blocks components with frequencies above or below this band. This contrasts with a high-pass filter, which allows through components with frequencies above a specific frequency, and a low-pass filter, which allows through components with frequencies below a specific frequency. In digital signal processing, in which signals represented by digital numbers are processed by computer programs, a band-pass filter is a computer algorithm that performs the same function. The term band-pass filter is also used for optical filters, sheets of colored material which allow through a specific band of light frequencies, and acoustic filters which allow through sound waves of a specific band of frequencies. [10]

5.1 Apply BPF: We create the function:

```
#Function Band Pass Filter
def BandPassFilter (magnitude_spectrum, radius_minor,
radius_major):
 #Dimensions
```

```
 rows, cols = magnitude_spectrum.shape # (512, 512)
 #Center the image
 crow, ccol = int(rows / 2), int(cols / 2) #(256, 256)
 #Mask of one
 mask_pf = np.ones((rows, cols), np.uint8) # (512, 512, 2)
 #center
 center = [crow, ccol]
 x, y = np.ogrid[:rows, :cols] # 0 to 511
 #All those that are less than the minor radius or all those that
are greater than the major radius is TRUE
 mask_area_pf = ((x - center[0]) ** 2 + (y - center[1]) ** 2 <=
radius_minor*radius_minor)+((x - center[0]) ** 2 + (y - center[1])
** 2 > radius_major*radius_major)
 #Zeroing everything TRUE
 mask_pf[mask_area_pf] = 0
 #Multiplying
 complex_pf = mask_pf*fshift
 result_pf = magnitude_spectrum*mask_pf

 return result_pf, mask_pf
```

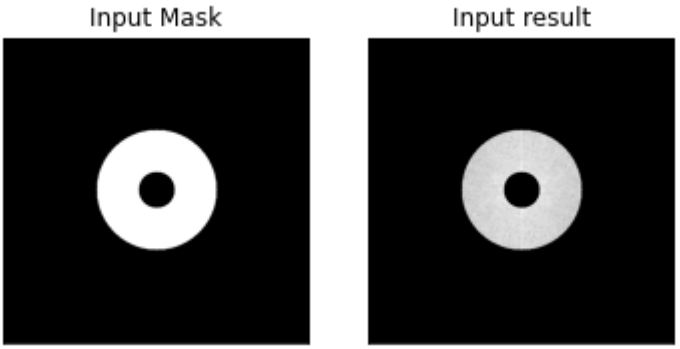


Image with radius minor 100 and radius major 110

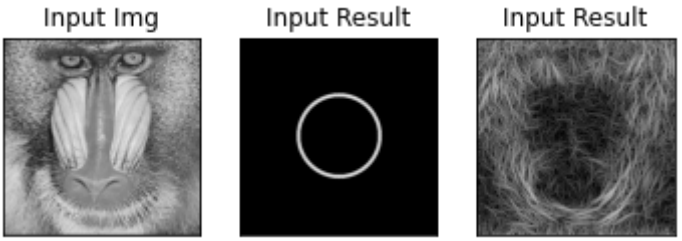


Image with radius minor 50 and radius major 110

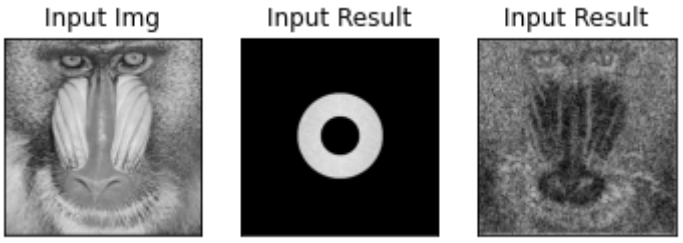


Image with radius minor 10 and radius major 110

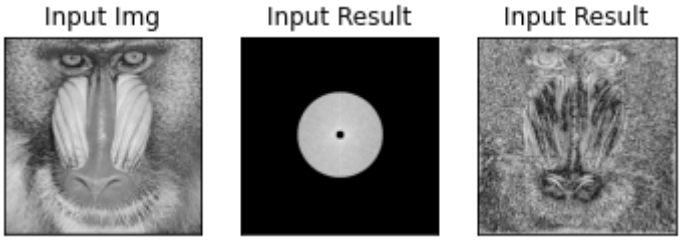


Image with radius minor 5 and radius major 100

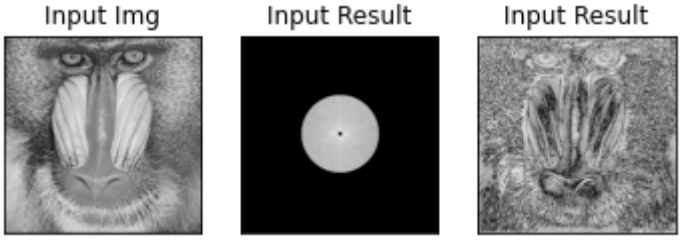


Image with radius minor 5 and radius major 50

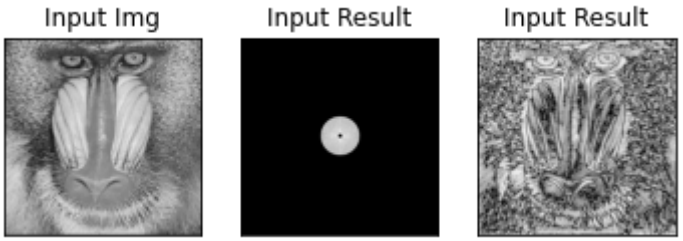
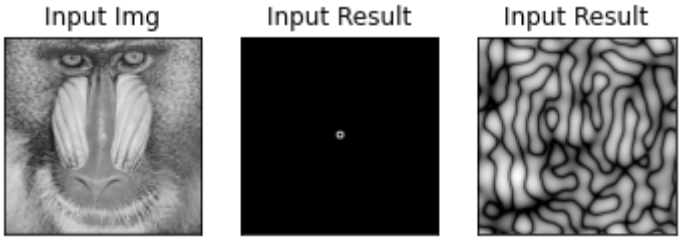


Image with radius minor 5 and radius major 10



5.2 Conclusions:

When the smaller radius is smaller, the edges of the image are more noticeable.
When the larger radius is smaller, the image is deformed.

6.- Compression

7.- Rotating an image

To rotate an image we are going to use the function “cv2.warpAffine()” Applies an affine transformation to an image. The function warpAffine transforms the source image using the specified matrix:[12]

$$\mathbf{dst}(x, y) = \mathbf{src}(\mathrm{M}_{11}x + \mathrm{M}_{12}y + \mathrm{M}_{13}, \mathrm{M}_{21}x + \mathrm{M}_{22}y + \mathrm{M}_{23})$$

Affine Transformation is a transformation that can be expressed in the form of a matrix multiplication (linear transformation) followed by a vector addition (translation).[13]

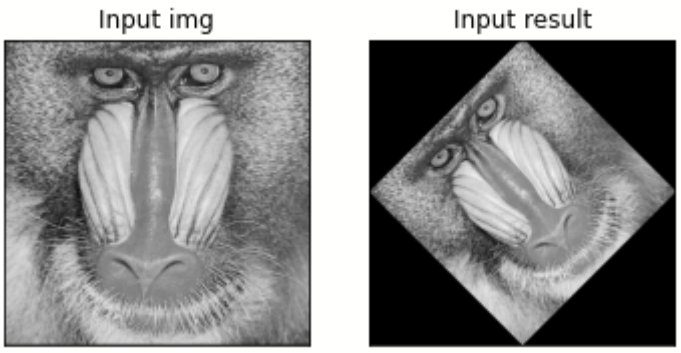
We create the function:

```
def RotatingImg (img):
 srcTri = np.array( [[0, 0], [img.shape[1] - 1, 0], [0, img.shape[0] - 1]]
).astype(np.float32)

 #Here we decide that the angle of rotation is 45
 dstTri = np.array( [[0, img.shape[1]*0.5], [img.shape[1]*0.5, 0],
[img.shape[1]*0.5, img.shape[0]*1]] ).astype(np.float32)

 warp_mat = cv2.getAffineTransform(srcTri, dstTri)
 warp_dst = cv2.warpAffine(img, warp_mat, (img.shape[1], img.shape[0]))

 return warp_dst
```



REFERECIAS:

1.- https://en.wikipedia.org/wiki/Fast_Fourier_transform
2.- https://pypi.org/project/opencv-python/
3.- https://numpy.org/
4.- https://matplotlib.org/
5.- https://numpy.org/doc/stable/reference/generated/numpy.fft.fft2.html
6.- https://numpy.org/doc/stable/reference/generated/numpy.fft.fftshift.html
7.- https://www.geeksforgeeks.org/python-inverse-fast-fourier-transformation/
8.- https://en.wikipedia.org/wiki/Low-pass_filter
9.- https://en.wikipedia.org/wiki/High-pass_filter
10.- https://en.wikipedia.org/wiki/Band-pass_filter
11.- https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html
12.- https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga0203d9ee5fcd28d40dbc4a1ea4451983
13.- https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html
14.- https://numpy.org/doc/stable/reference/generated/numpy.fft.ifft2.html