UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTE OF COMPUTING

INTRODUCTION TO DIGITAL IMAGE PROCESSING
MC920 / MO443

HOMEWORK 1

STUDENT: Gian Franco Joel Condori Luna
RA: 234826

1.- Problem Specification

The objective of this work is to implement some operations in images, both monochrome and colorful, in the spatial domain. The masks and matrices represented must be explicitly used in the codes, that is, functions available in libraries must not be used in the implementation.

2.- Data Entry

The input images are in PNG (Portable Network Graphics) format. Some examples of monochrome images will be available in the directory: "http://www.ic.unicamp.br/~helio/imagens_png/" as soon as the examples of colorful images are in the directory: "https://www.ic.unicamp.br/~helio/imagens_coloridas/ "

3.- Data Output

Output images must be in PNG format.

4.- Problem Solving

The work was developed in Python language, for that, jupyter notebooks were used in the Google Colaboratory environment (Google Colab).

4.1 Colorful pictures

4.1.1 Problem 1: This problem asks us to alter an RGB colorful image with the following formula:

$$R' = 0.393R + 0.769G + 0.189B$$
$$G' = 0.349R + 0.686G + 0.168B$$
$$B' = 0.272R + 0.534G + 0.131B$$

Then if the transformation, in the case that R', G' or B' have some value greater than 255, this has to be limited to 255.

a) Codification:

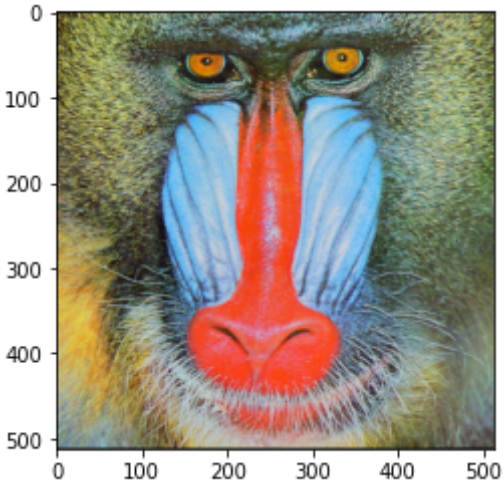
To solve this problem we first import the libraries that we are going to use:

```
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. [1]
- 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. [2]
- Matplotlib is a plotting library used for 2D graphics in Python programming language, it is very flexible and has many built-in default values. [3]

Then we read the image from the directory that was provided, We read the image using "cv2.imread ()" this function reads us the image in the BGR form for which with the function "cv2.split ()" I separate the image in bands and then with the function "cv2.merge ()" I put it in the position that I want RGB.

```
img_colorida =
cv2.imread("/content/drive/MyDrive/imagenes/imagens_coloridas/baboon.png
")
b,g,r = cv2.split(img_colorida)
rgb_img = cv2.merge([r,g,b])
plt.imshow(rgb_img)
```



Function limit255: This function was created to be able to limit each pixel to 255. It receives any number as a parameter, then we ask if the entered number is negative then 0 is returned, otherwise it asks if the number is greater than 255, if it is greater, it is modulo 256 is applied (the modulo operation returns the remainder of a division) otherwise we return the same value.

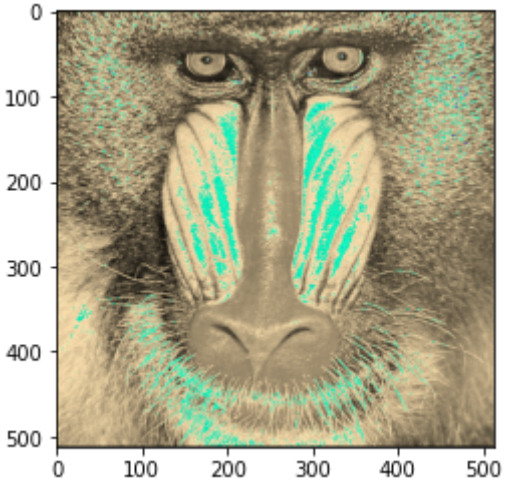
```
def limit255( px ):
 n_px = np.where(px<0, 0, np.where(px>255, px%256, px))
 return n_px
```

In order to perform mathematical equations in the image we have to convert it to a floating value for which we use the function "np.asarray ()" sending it as the second parameter "dtype = np.float32"

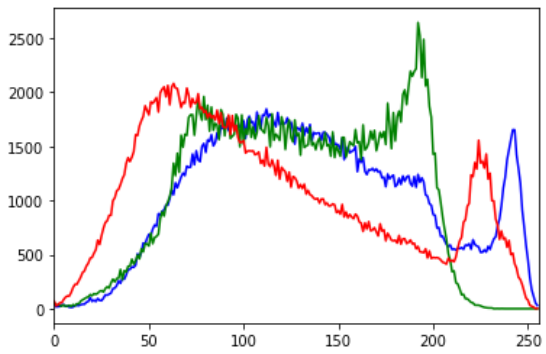
```
rgb_img = np.asarray(rgb_img,dtype=np.float32)
```

Finally we apply the requested operations in a vector way, I decided to do it in parts, to have better control and to be able to apply the "limit255 ()" function. At the end of the operations, I put together the three responses in a single array with the function "cv2.merge ()" and I convert the result to the format "uint8"

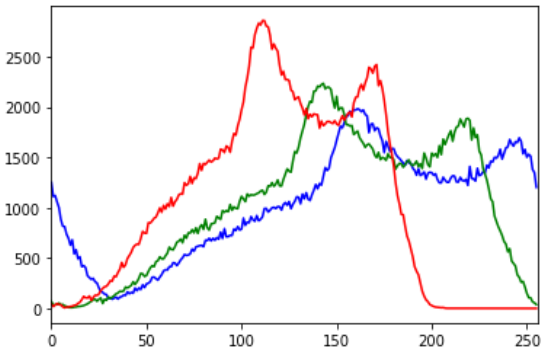
```
#R'=  0.393R+ 0.769G+ 0.189B
Rp = limit255(0.393*rgb_img[:,:,0] + 0.769*rgb_img[:,:,1]+
0.189*rgb_img[:,:,2])
#G'=  0.349R+ 0.686G+ 0.168B
Gp = limit255(0.349*rgb_img[:,:,0] + 0.686*rgb_img[:,:,1]+
0.168*rgb_img[:,:,2])
#B' = 0.272R+ 0.534G+ 0.131B
Bp = limit255(0.272*rgb_img[:,:,0] + 0.534*rgb_img[:,:,1]+
0.131*rgb_img[:,:,2])
#Together
new_img = cv2.merge([Rp,Gp,Bp])
#Convert a format uint8
new_img = new_img.astype(np.uint8)
plt.imshow(new_img)
```



b) Conclusions: You can see an image with greenish tones in certain places and the others in sepia tones. Looking at the histograms we can see how the red band has more pronounced peaks than the original image and the green band has decayed a bit.



hist-img-origin



hist-img-result

4.1.2 Problem 2:

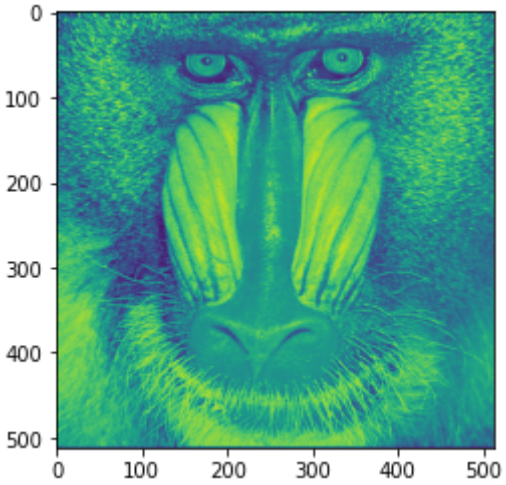
Given the image in RGB format, alter the image so that it contains only one color band, whose values are calculated by the weighted average:

$$I= 0.2989R+ 0.5870G+ 0.1140B$$

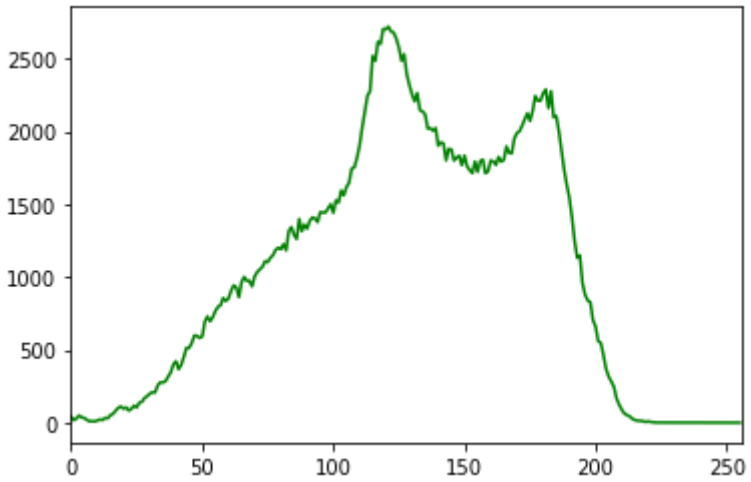
a) Codification:

We elaborate, in code, the equation that was given to us and to each result of this we apply the function "limit255 ()" to limit the pixels, to the final result we convert it to UINT8 to be able to visualize it.

```
#I= 0.2989R+ 0.5870G+ 0.1140B
I = limit255(0.2989*rgb_img[:,:,0] + 0.5870*rgb_img[:,:,1]+
0.1140*rgb_img[:,:,2])

I = I.astype(np.uint8)
plt.imshow(I)
```



b) Conclusions: Looking at the image I can tell that the image is in grayscale and looking at the hiistogram we can conclude that it has predominant colors in the range of 100 to 200.



4.2 Monochrome Images:

Filtering applied to a digital image is a local operation that alters the intensity values of the pixels in the image taking into account both the value of the pixel in question and the values of the neighboring pixels. In the filtering process, a convolution operation of a mask by the image is used. This process is equivalent to traversing the entire image, changing its values according to the weights of the mask and the intensities of the image.

To solve these problems, the "conv_vectorial ()" function was performed, which receives the image and the filter as parameters. This function was performed based on the general convolution formula:

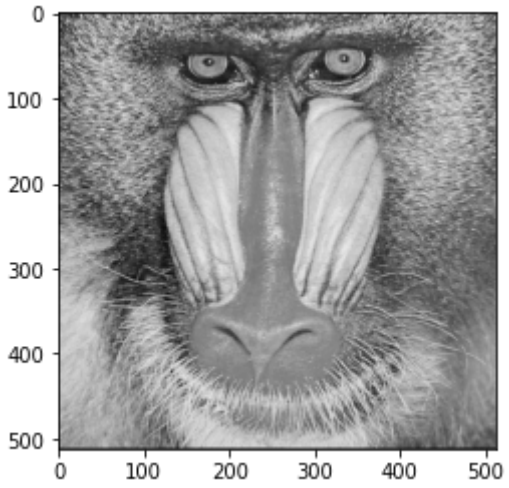
$$g(x,y) = h(x,y) * f(x,y) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} f(i,j)h(x-i, y-j)$$

In order to carry out the general formula, 4 loops are frequently used, but in this work we did it vectorized, for which only one loop was necessary, since we are seeing the matrix as if it were a very long vector where the number of elements that is going to have are multiplying its height by its width. In order to obtain the result of each new pixel, we only have to multiply the filter with the corresponding image portion and add this result.

```
#VECTORIAL
def conv_vectorial(image, kernel):
  height, width = image.shape
  h, w = kernel.shape

  new_h = height - h + 1
  new_w = width - w + 1

  iter_h = int(h /2)
```

```
  iter_w = int(w / 2)

  # New matriz
  new_image = np.zeros((height, width), dtype=np.float)
  new_image[:,:] = image[:,:]

  for i in range(new_w*new_h):
      r = int(np.floor(i/new_w))
      c = int(i%new_w)

      new_image[r+iter_h, c+iter_w] = np.sum(image[r:r+h, c:c+w] *
kernel)

  return new_image
```

Our base image to solve our problems with filters will be this:

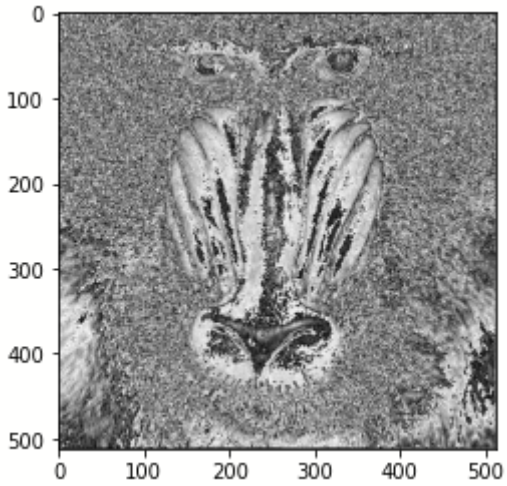


4.2.1 Problem 3:

Apply filter h1:

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

Result of applying the filter



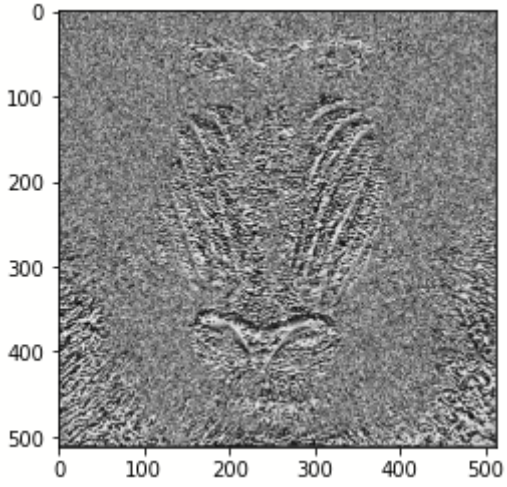
The conclusions that we can draw from the image is that apparently a filter was added that caused noise of black and white colored dots. It can be a horizontal filter

4.2.2 Problem 4:

Apply filter h2:

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

Result of applying the filter



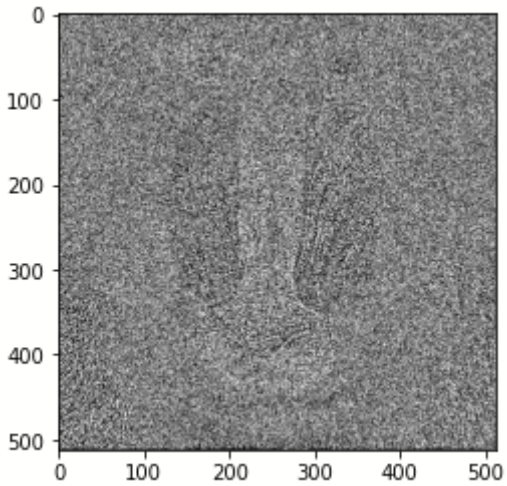
The conclusions that we can draw from the image seem to be highlighting the edges. It can be a vertical filter

4.2.3 Problem 5:

Apply filter h3:

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

Result of applying the filter



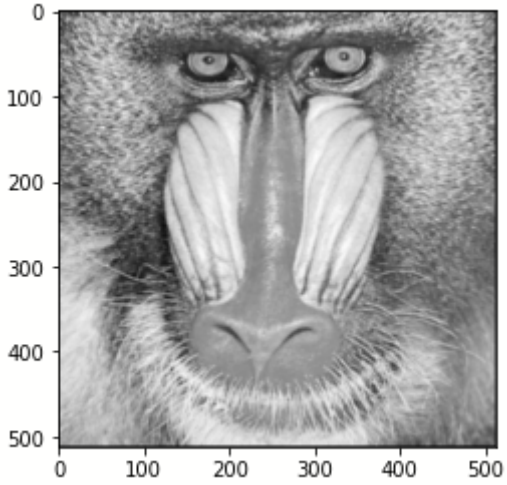
The conclusions that we can draw from the image seem to be highlighting the edges. It can be a vertical filter

4.2.4 Problem 6:

Apply filter h4:

1/9 *	1	1	1
	1	1	1
	1	1	1

Result of applying the filter



The conclusions that we can draw from the image is that the values decrease.

4.2.5 Problem 7:

Apply filter h5:

-1	-1	2
-1	2	-1
2	-1	-1

Result of applying the filter

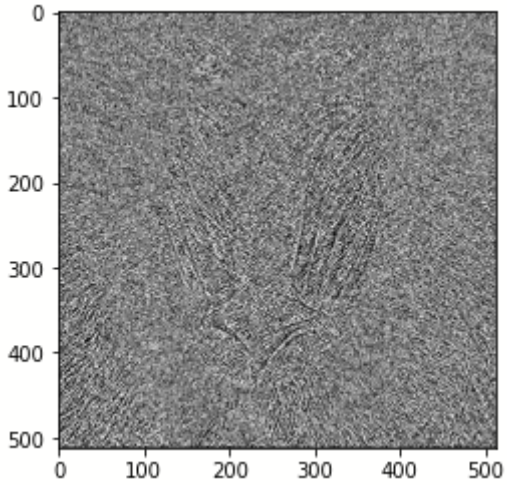


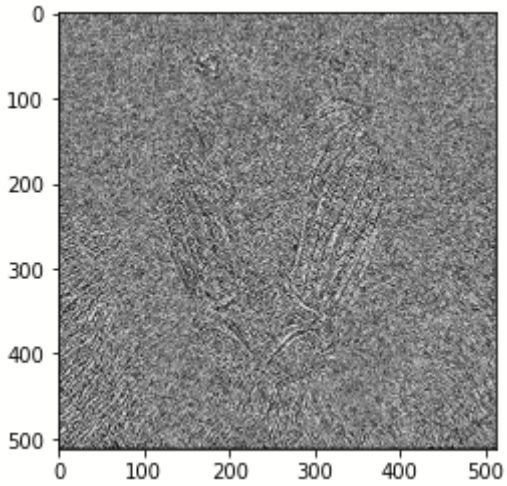
Image similar to h3.

4.2.6 Problem 8:

Apply filter h6:

2	-1	1
-1	2	-1
-1	-1	2

Result of applying the filter



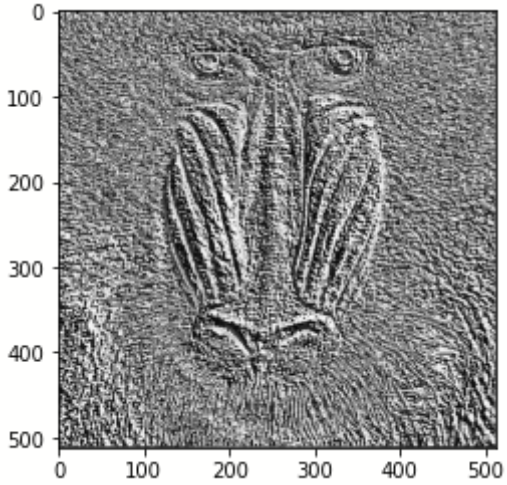
We can conclude from seeing the image that the filter produces horizontal edges

4.2.7 Problem 9:

Apply filter h7:

0	0	1
0	0	0
-1	0	0

Result of applying the filter



We can conclude from seeing the image that the filter produces vertical edges

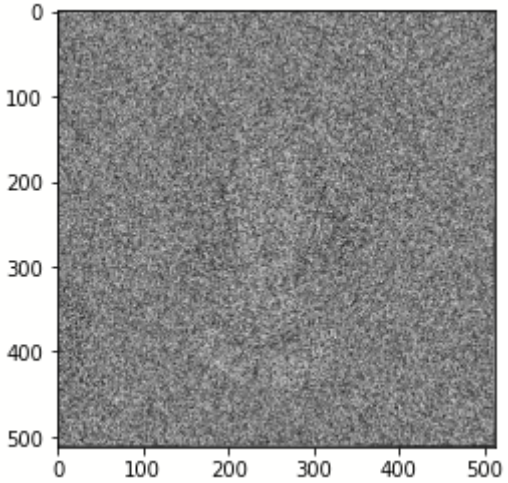
4.2.8 Problem 10:

Apply filter h8:

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

-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Result of applying the filter

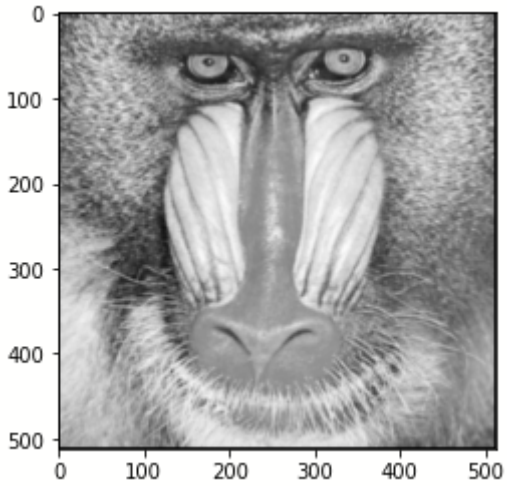


4.2.9 Problem 11:

Apply filter h9:

1/256 *	1	4	6	4	1
	4	16	24	16	4
	6	24	36	24	6
	4	16	24	16	4
	1	4	6	4	1

Result of applying the filter

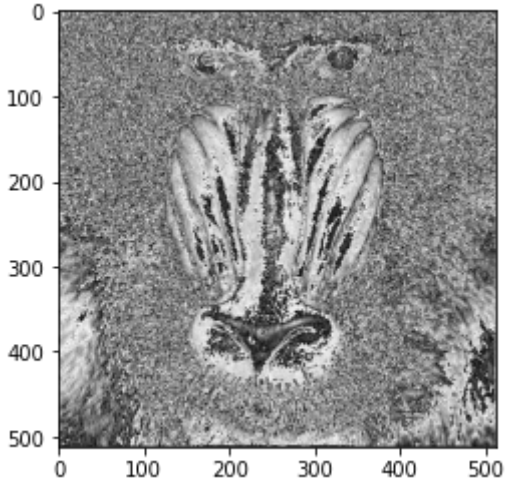


4.2.10 Problema 10:

Combined filter h1 and h2 in operation:

$$\sqrt[2]{(h1)^2 + (h2)^2}$$

Result of applying the filter



The conclusions that I can draw from this image is that it seems that it is applying the vertical filter and horizontal filter.

Referencias

1.- https://pypi.org/project/opencv-python/
2.- https://numpy.org/
3.- https://matplotlib.org/