

Relatorio Trabalho 1
Introduction to Image Processing
IC UNICAMP

Luis Antonio Bernal Chahuayo
RA 239423

April 2021

1 Color Images

1.1 Sepia Effect

It's requested to alter the channels using the following equation:

$$\begin{aligned} R' &= 0.393R + 0.769G + 0.189B \\ G' &= 0.349R + 0.686G + 0.168B \\ B' &= 0.272R + 0.534G + 0.131B \end{aligned}$$

For this task, the three channels are separated in variables. It's important to remember that OpenCV, read the color images in the BGR order. After applying the formula, it's necessary to verify if the new values. Are in the real range [0, 255]. If we don't apply this verification weird behaviors could happened, like in Fig. 1.

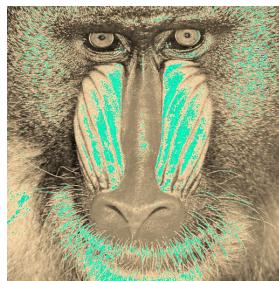


Figure 1: Wrong behavior without verification of values are in range

The following code returns the new image, casted as Unsigned Int 8 bits.

```

import numpy as np
def sephia(image):
    b = image[:, :, 0]
    g = image[:, :, 1]
    r = image[:, :, 2]

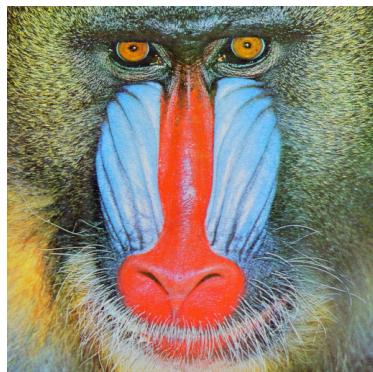
    rp = 0.393*r + 0.796*g + 0.189*b
    gp = 0.349*r + 0.686*g + 0.168*b
    bp = 0.272*r + 0.534*g + 0.131*b

    # Verify if new values aren't greater than 255, else write 255
    rp = np.where(rp>255,255,rp)
    gp = np.where(gp>255,255,gp)
    bp = np.where(bp>255,255,bp)

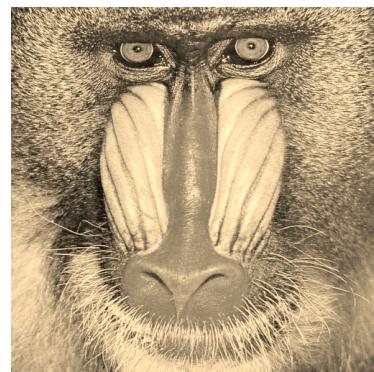
    new_image = np.stack((bp, gp, rp), axis=2)
    return new_image.astype(np.uint8)

```

The results are shown in the images below. The execution time is near to 0.1s per image.



(a) Original Picture



(b) After transformation



(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation

1.2 Gray scale Transform

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

In this case the following code will output a one channel image. It will produce a gray scale image, based on the previous formula. Also it's necessary to verify the values to avoid weird behaviors.

```
def to_gris(image):
    b = image[:, :, 0]
    g = image[:, :, 1]
    r = image[:, :, 2]

    i = 0.2989 * r + 0.5870 * g + 0.1140 * b
    i = np.where(i > 255, 255, i)

    return i.astype(np.uint8)
```

The images below shown the results:



(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation

2 Monochromatic Filters

The following code receives a 2D grayscale image as first argument, and a square kernel (3×3 , 5×5 , ...)

First adds a zero-padding to the image in order to keep the original dimensions of the image. This algorithm iterates over the image, and for each pixel is obtained the sum of the windowed image and the kernel. The addition and multiplication are using the vectorization functions of numpy. The average time of execution is near to 1.3s.

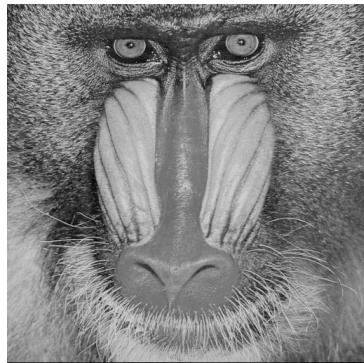
```
def filter(image2D, kernel):
    padding = int((kernel.shape[0]-1)/2)
    image2D_pad = np.pad( image2D, padding )
    new_image = np.zeros( image2D.shape )
    for x in range( image2D.shape[0] - padding ):
        for y in range( image2D.shape[1] - padding ):
            window = image2D_pad[ x : x + kernel.shape[0]
                                  , y : y + kernel.shape[0] ]
            new_image[x][y] = ( window * kernel ).sum()

    new_image = np.where( new_image > 255, 255, new_image )
    new_image = np.where( new_image < 0, 0, new_image )
    return new_image.astype(np.uint8)
```

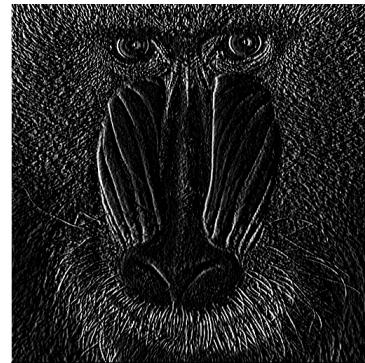
2.1 Filter h1

$$h_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

This filter, highlight the vertical borders and hide the horizontal borders.



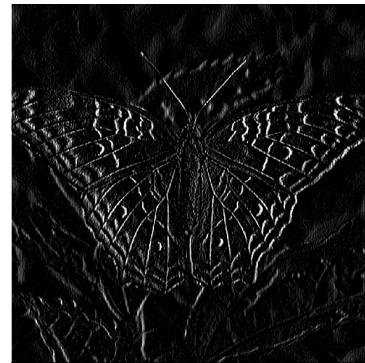
(a) Original Picture



(b) After transformation



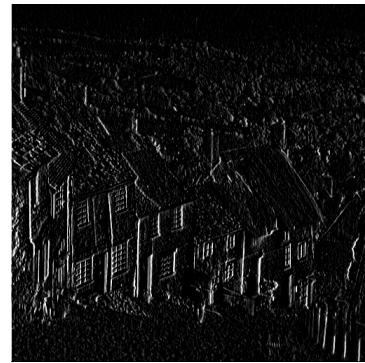
(c) Original Picture



(d) After transformation



(e) Original Picture

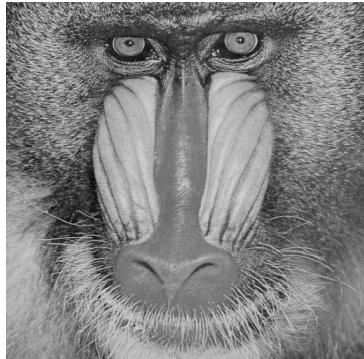


(f) After transformation

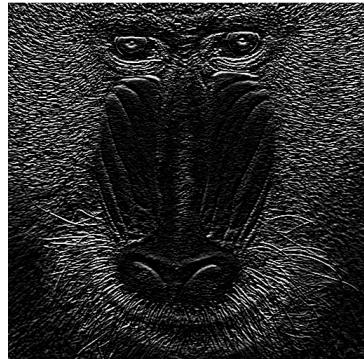
2.2 Filter h2

$$h_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

This filter, highlight the horizontal borders and hide the vertical borders.



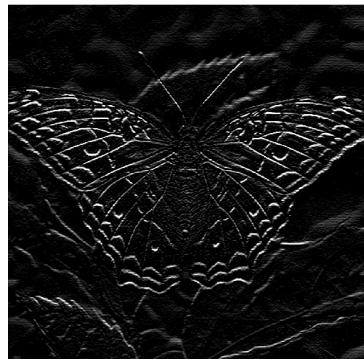
(a) Original Picture



(b) After transformation



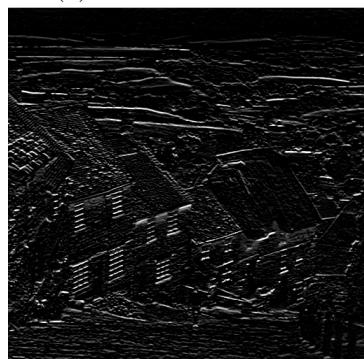
(c) Original Picture



(d) After transformation



(e) Original Picture

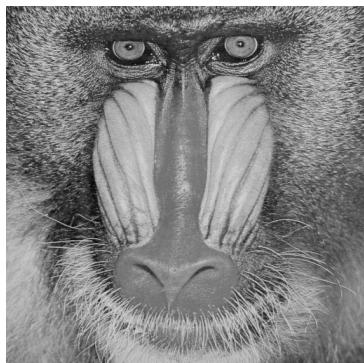


(f) After transformation

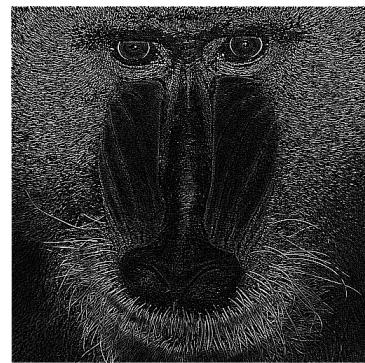
2.3 Filter h3

$$h_3 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

This filter highlight all borders. However it feels a pixelation effect. And the borders aren't continuous.



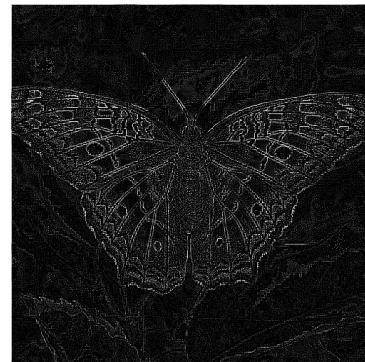
(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation



(e) Original Picture

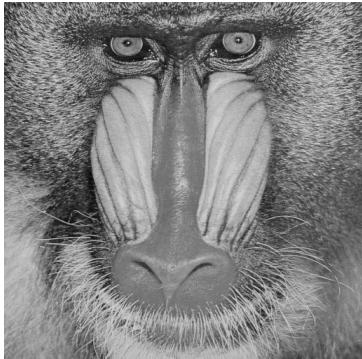


(f) After transformation

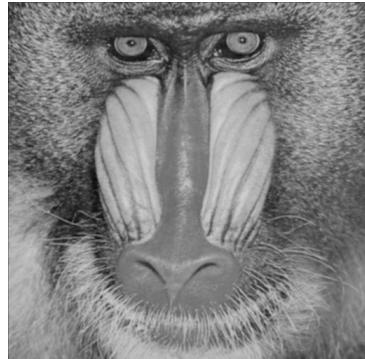
2.4 Filter h4

$$h_4 = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This filter applies a blur effect to the image.



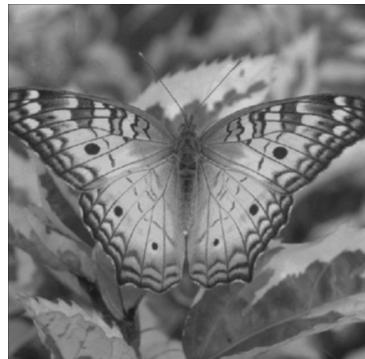
(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation



(e) Original Picture

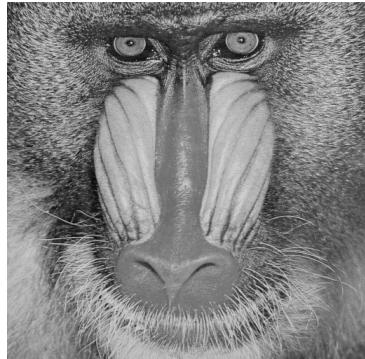


(f) After transformation

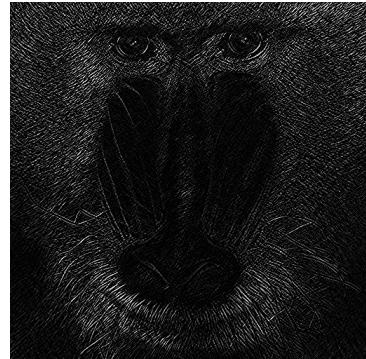
2.5 Filter h5

$$h_5 = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

This effects highlights the diagonal borders. Going from left to right.



(a) Original Picture



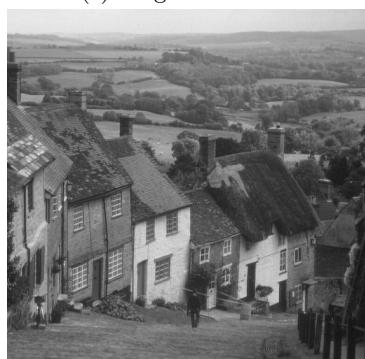
(b) After transformation



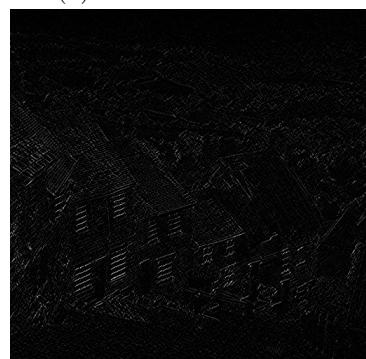
(c) Original Picture



(d) After transformation



(e) Original Picture

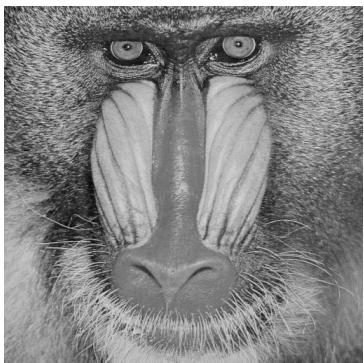


(f) After transformation

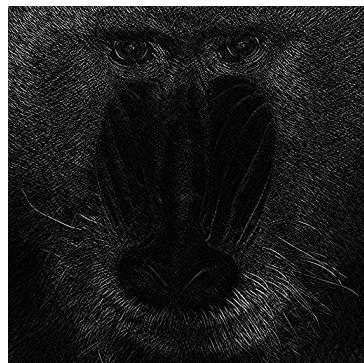
2.6 Filter h_6

$$h_6 = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

Similarly to the h_5 effect. Highlights the diagonal borders, but now in the inverse diagonal.



(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation



(e) Original Picture

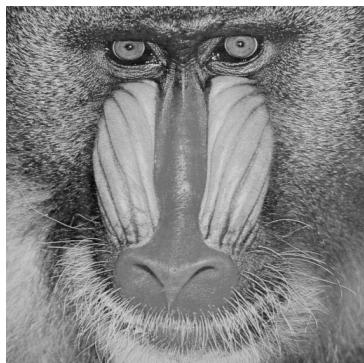


(f) After transformation

2.7 Filter h_7

$$h_7 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

This filter highlights the borders, but similarly to the h_3 filter produces a noise effect.



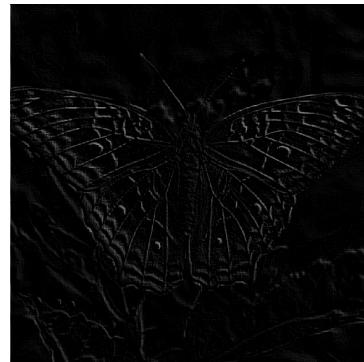
(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation



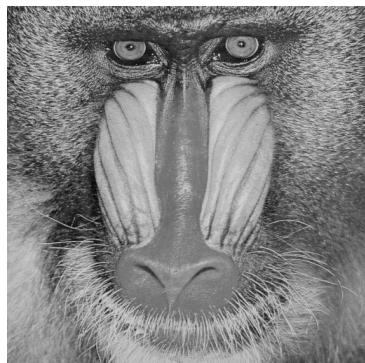
(e) Original Picture



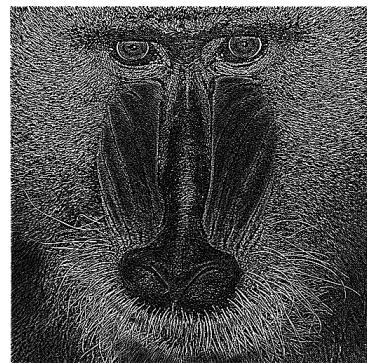
(f) After transformation

2.8 Filter h8

$$h_8 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$



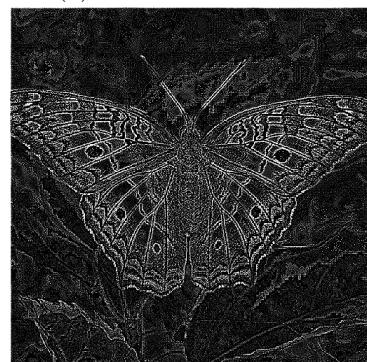
(a) Original Picture



(b) After transformation



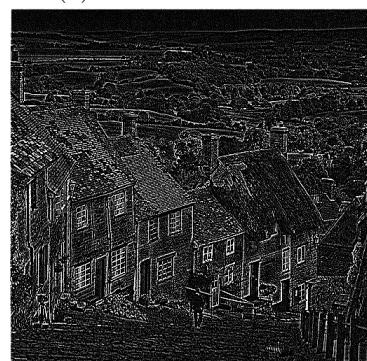
(c) Original Picture



(d) After transformation



(e) Original Picture

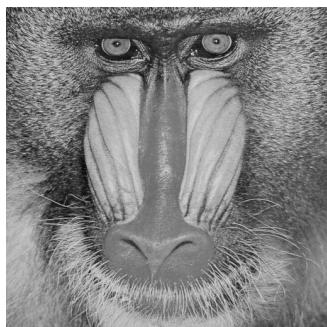


(f) After transformation

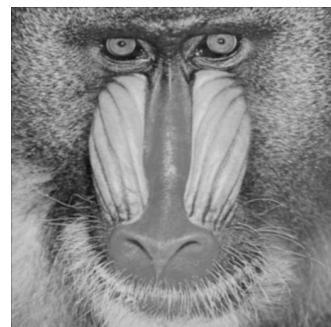
2.9 Filter h9

$$h_9 = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Similarly to filter h_4 applies a blur effect to the image.



(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation



(e) Original Picture

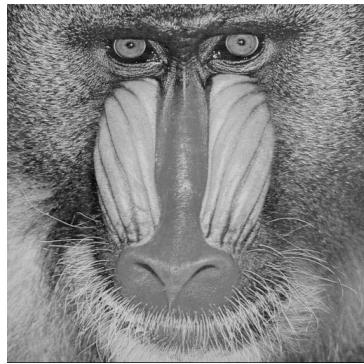


(f) After transformation

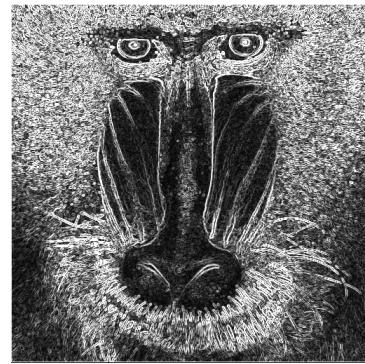
2.10 Filter Combination

$$h_{10} = \sqrt{(h_1)^2 + (h_2)^2}$$

This filter combines the h_1 and h_2 effects, strongly highlighting both borders: Horizontal and vertical. Additionally produces continuous borders.



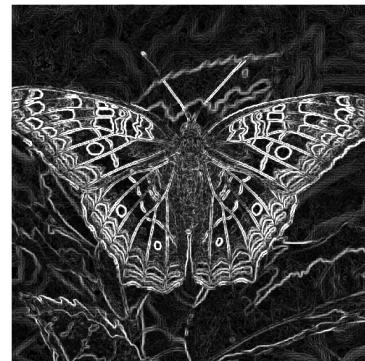
(a) Original Picture



(b) After transformation



(c) Original Picture



(d) After transformation



(e) Original Picture



(f) After transformation

3 Command Line Interface Programs

In order to easily execute and test algorithms, a command line interface was written for each program. Allows to save on an output file or show a window with the image output.

3.1 Color effects Program

This program allows to choose between "sepia" and "toGray" effects. **python3 color_change.py -help**

```
usage: color_change.py [-h] [-e {sepia,gray}] [-s SAVE]
                      file

Sepia 2D color Image Processing – UNICAMP

positional arguments:
  file                  input file image .png

optional arguments:
  -h, --help            show this help message and exit
  -e {sepia,gray}, --effect {sepia,gray}
                        Choose effect
  -s SAVE, --save SAVE  Save Image
```

3.2 Grayscale Filters Program

This program allows to choose between 10 filter kernels.

python3 filter.py -help

```
usage: filter.py [-h] [-k {1,2,3,4,5,6,7,8,9,10}] [-s
                 SAVE] file

Filter 2D Gray Image Processing – UNICAMP

positional arguments:
  file                  input file image .png

optional arguments:
  -h, --help            show this help message and exit
  -k {1,2,3,4,5,6,7,8,9,10}, --kernel
                        {1,2,3,4,5,6,7,8,9,10}
                        Choose number of kernel to apply
  -s SAVE, --save SAVE  Save Image
```

References

- [1] OpenCV Python Documentation (https://docs.opencv.org/master/d7/d16/tutorial_py_table_of_contents_core.html)
- [2] Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.