1 Python correction

```python
# display images
import matplotlib.pyplot as plt
# ndimage defines a few filters
from skimage.filters.rank import mean
import skimage.filters  # prewitt, gaussian
# numeric calculation
import numpy as np
# measure time
import time
# read and save images
from skimage.io import imread, imsave
# convolution method
from scipy.signal import convolve2d
# data
import skimage.data as data
```

1.1 First manipulations

1.1.1 Open, write images

The following file loads the camera image and display it. The print function is optional.

```python
# load file camera
camera = data.camera()
# load file cerveau.jpg
brain = imread('cerveau.jpg')
print(type(brain))
print(brain.shape, brain.dtype)
# save file
imsave('test.png', brain)
```

1.1.2 Display images

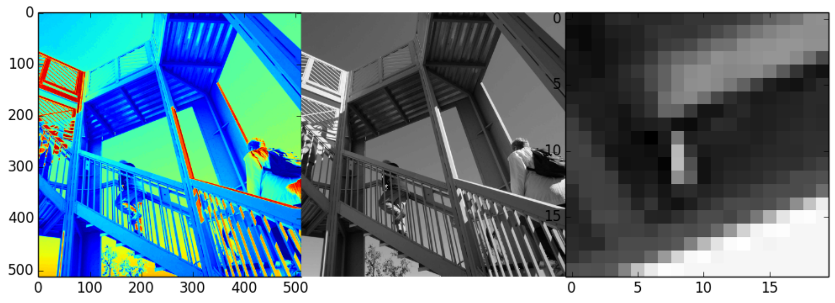
You can modify to previous example to add the following lines:

```python
plt.imshow(camera)
plt.show()
```

Notice that you have to close the image window to write commands again. Also, the colormap is not the good one by default (see Fig. 1).

```python
plt.figure(figsize=(10, 3.6))
# first subplot
plt.subplot(131)
plt.imshow(camera)
# second subplot
plt.subplot(132)
plt.imshow(camera, cmap=plt.cm.gray)
plt.axis('off')
# third subplot (zoom)
plt.subplot(133)
plt.imshow(camera[200:220, 200:220], cmap=plt.cm.gray, interpolation='
    ↪ nearest')
plt.subplots_adjust(wspace=0, hspace=0.,
                    top=0.99, bottom=0.01,
                    left=0.05, right=0.99)
plt.show()
```

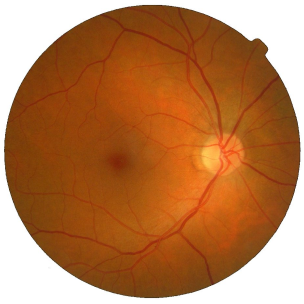
Figure 1: Displaying images with an adapted colormap.

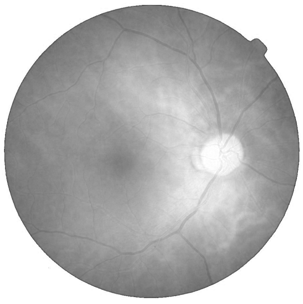


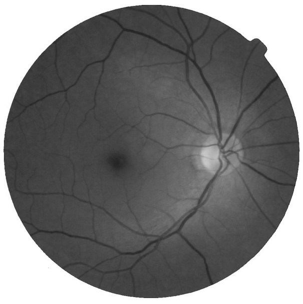
1.1.3 Color channels

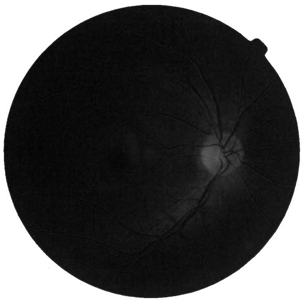
A color image is constituted of (generally) three channels. This representation follows the human visual perception principles: in the human retina, the sensitive cells (the cones) react to specific wavelength that correspond to red, green and blue. The sensors technology adopted the *same* characteristics and a so-called Bayer filter has 2 green filters for 1 red and 1 blue. Consequently, the green channel presents a better resolution than the other channels.

Figure 2: The green channel presents the best contrasts in the case of retina images.


(a) Color image.


(b) Red channel.

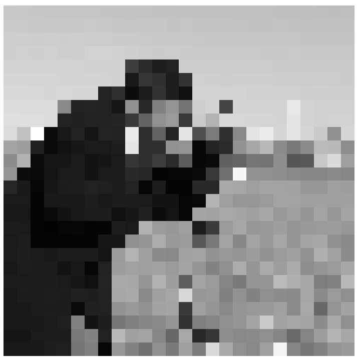

(c) Green channel

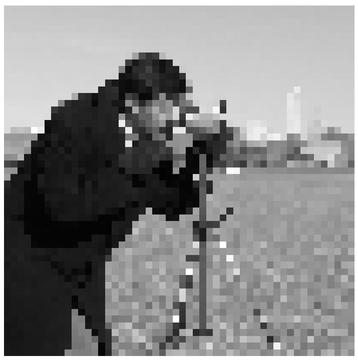

(d) Blue channel.

1.1.4 Image Resizing

The number of pixels is reduced by subsampling the image. Notice that there is no anti-aliasing filter applied to the image before reducing its size. The result is presented in Fig.3 with images of the same size, and in Fig.4 with images at the same resolution.

Figure 3: Reduction of the scale of the image on each axis, showing a so-called *pixellisation* effect. Represented at the same size, the density of pixels is thus reduced.


(a) Scale divided by 20.


(b) Scale divided by 10.

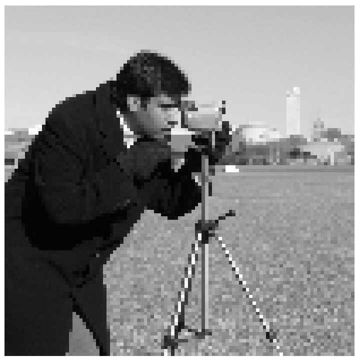
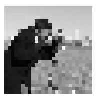

(c) Scale divided by 5.

Figure 4: At a constant resolution, images with different definitions are represented with different sizes.


(a)

(b)

(c)

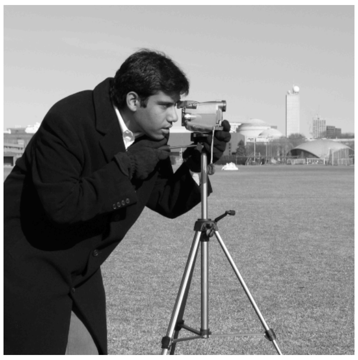
1.1.5 Color quantization

The following code uses the properties of integer operations to round values to the nearest integer. Illustration is presented Fig. 5.

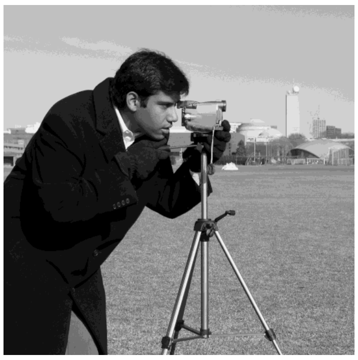
```python
# Integer division operator //
q4 = camera // 4*4;
q16= camera //16*16;
q32= camera //32*32;
```

Figure 5: Reduction of the number of gray levels (quantization).

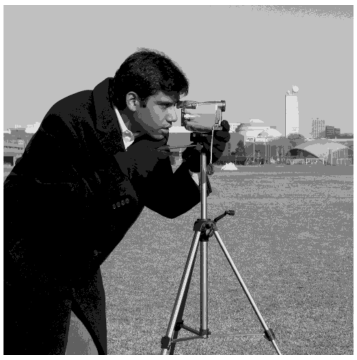
(a) q4.



(b) q16.



(c) q32.

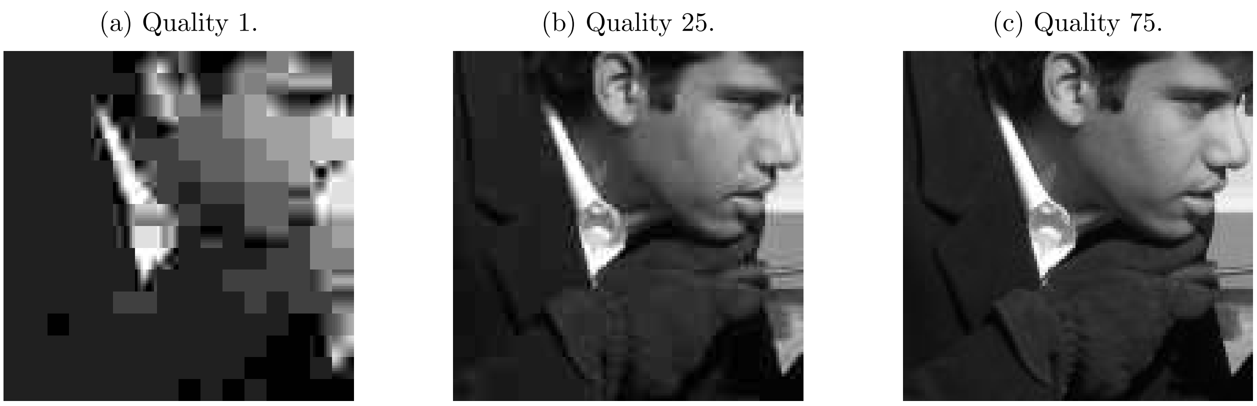


1.1.6 JPEG file format

In order to test the effect of jpeg compression, one can use the parameter quality. For the lowest quality, the loss of informations is really important (see Fig.6).

```python
# test jpeg quality
imsave("a_25.python.jpeg", camera, quality=25);
imsave("a_100.python.jpeg", camera, quality=100);
imsave("a_50.python.jpeg", camera, quality=50);
imsave("a_75.python.jpeg", camera, quality=75);
imsave("a_1.python.jpeg", camera, quality=1);
```

Figure 6: Different quality used to compress jpeg files.

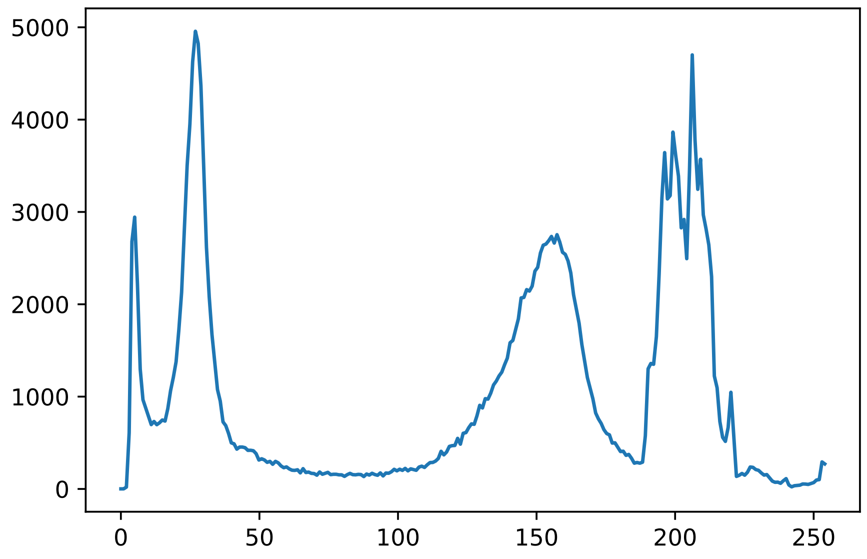


1.2 Histogram

To compute the histogram of an image, we recommand the numpy function histogram (see result in Fig. 7).

```python
h, edges = np.histogram(camera, bins=256)
plt.plot(edges[:-1], h)
```

Figure 7: Histogram of camera image.



You can also write your own code. The execution time is lower for the numpy method, which is vectorized and optimized.

```python
# Histogram function with 2D image
def compute_histogram(image):
    tab = np.zeros((256, ), dtype='I')
    X, Y = image.shape
    for i in range(X):
        for j in range(Y):
            tab[image[i,j]]+=1

    return tab
```

```python
# Histogram function with flatten image (vector)
def compute_histogram2(image):
    im = image.flatten()
    tab = np.zeros((256, ), dtype='I')
    for i in im:
        tab[i]+=1
    return tab
```

The following code presents a comparison of the different method for histogram computation.

```python
# load camera image and compute histograms
camera = data.camera()
t0 = time.time()
h = compute_histogram(camera)
t1 = time.time()
h2 = compute_histogram2(camera)
t2 = time.time()

# .... plots
print(f"execution time 2D: {t1-t0:.2} s")
plt.subplot(131)
plt.plot(h)
plt.title('2D function')

print(f"execution time 1D: {t2-t1:.2} s")
plt.subplot(132)
plt.plot(h2)
plt.title('1D function')

# last plot: with numpy function
plt.subplot(133)
t3 = time.time()
h, edges = np.histogram(camera, bins=256)
plt.plot(edges[:-1], h)
t4 = time.time()
print(f"execution time numpy: {t4-t3:.2} s")

# display
plt.show()
```

The console outputs the following computation durations:

```
execution time 2D: 0.44 s
execution time 1D: 0.4 s
execution time numpy: 0.0025 s
```

1.3 Linear mapping of the image intensities

The linear mapping is a simple method that stretches linearly the histogram. If displayed with matplotlib, the images is linearly stretched, thus the modification cannot be observed.

```python
def image_stretch(image):
    # returns image with new maximum and minimum at 255 and 0
    I = I - np.min(I)
    I = 255 * I / np.max(I)
    return I.astype('int')
```

1.4 Low-pass filtering

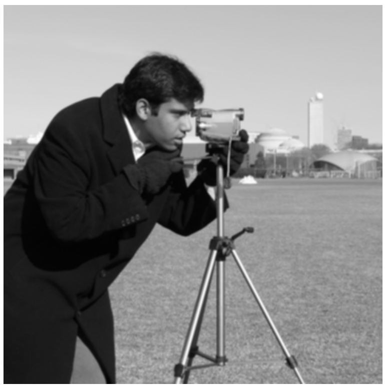
The module scipy.ndimage.filters contains the usual filter functions.

The mean filter is illustrated in Fig. 8.

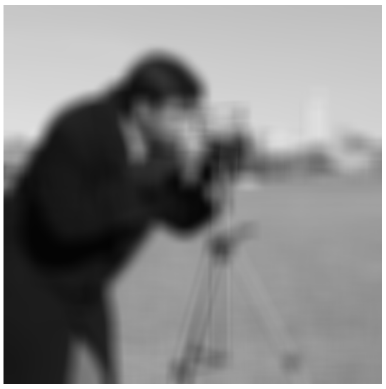
```python
# mean on a 3x3 neighborhood
m3 = mean(camera, np.ones((3, 3))
m25= mean(camera, np.ones((25, 25))

plt.subplot(121)
plt.imshow(m3, cmap=plt.cm.gray)
plt.axis('off')
plt.title('3x3 mean filter')

plt.subplot(122)
plt.imshow(m25, cmap=plt.cm.gray)
plt.axis('off')
plt.title('25x25 mean filter')

plt.show()
```

Figure 8: Mean filters.

(a) Neighborhood of size 3x3.



(b) Neighborhood of size 25x25.

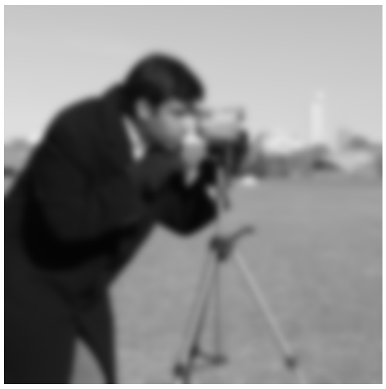


1.4.1 Gaussian filter

The gaussian filter is presented in Fig. 9.

```python
# camera image
camera = data.camera()

# Gaussian filter
gaussian = skimage.filters.gaussian(camera, 5)
```

Figure 9: Gaussian filter of size 5.



1.5 High-pass filter

The computation of the high-pass filter is simply the subtraction of a low-pass filter from the original image. For example:

```python
H = I−m25
```

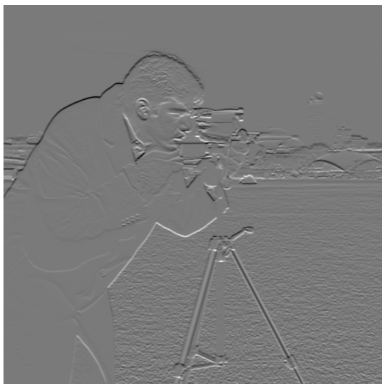
1.6 Derivative filters

Derivative filters (Prewitt, Sobel...) use a finite derivation approximation. They are very sensitive to noise (as every system using a derivation). The gradient is defined as a vector, and a norm should be used to display a resulting image. Notice that with these filters, the connexity of the contours is not preserved. Illustration is proposed in Fig. 10.

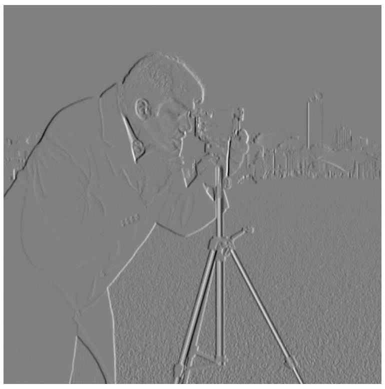
```python
# camera image
camera = data.camera()
camera.astype('int32');

# Prewitt filter
prewitt0 = skimage.filters.prewitt(camera, axis=0)
prewitt1 = skimage.filters.prewitt(camera, axis=1)

# Sobel filter
dy = skimage.filters.sobel(camera, axis=0) # vertical
dx = skimage.filters.sobel(camera, axis=1) # horizontal
mag = np.hypot(dx, dy)  # magnitude
sobel = mag * 255.0 / mag.max()  # normalize (Q&D)

# display results
plt.subplot(131)
plt.imshow(prewitt0, cmap=plt.cm.gray)
plt.axis('off')
plt.title('Prewitt filter axis 0')

plt.subplot(132)
plt.imshow(prewitt1, cmap=plt.cm.gray)
plt.axis('off')
plt.title('Prewitt filter axis 1')

plt.subplot(133)
plt.imshow(sobel, cmap=plt.cm.gray)
plt.axis('off')
plt.title('Sobel filter')

plt.show()
```

Figure 10: Prewitt filter.

(a) Prewitt filter for axis x.



(b) Prewitt filter for axis y.



The previous code uses filters proposed by the skimage module. If you want to write down the convolution matrix, this is the solution:

```python
h = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1] ])
grad1 = convolve2d(camera, h);
plt.imshow(grad1, "gray")
plt.show()

grad0 = convolve2d(camera, h.transpose());
plt.imshow(grad0, "gray")
plt.show()

plt.imshow(np.sqrt(grad0**2 + grad1**2), "gray")
plt.show()
```

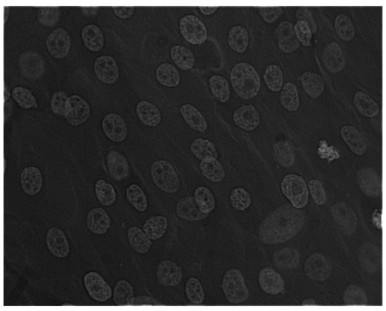
1.7 Enhancement filter

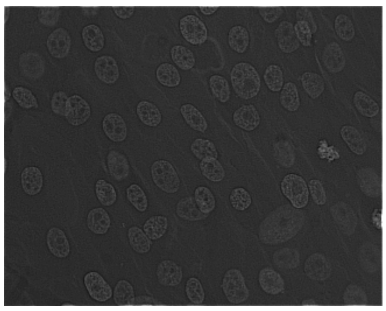
This functions adds the Laplacian filter to the original image.

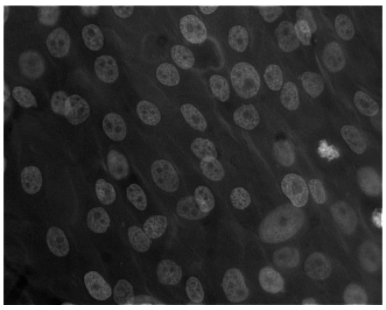
```python
def sharpen(I, alpha):

    h = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1] ])
    L = convolve2d(I, h, mode='same')
    np.max(L)
    E = alpha * I + L
    E = skimage.exposure.rescale_intensity(E, out_range=(0,255))
    E = E.astype(np.uint8)

    return E
```

This constitutes a really simple and efficient edge sharpening method. To test it, you may write this example code:

```python
I = imread('osteoblaste.png').astype(np.float)
I = I/255
A = [1, 0.5, 5]

fig = plt.figure(figsize=(12,8))
plt.subplot(221)
plt.imshow(I, cmap=plt.cm.gray)
for i,alpha in enumerate(A):
    plt.subplot(222+i)
    E = sharpen(I, alpha)
    E = skimage.exposure.rescale_intensity(E, out_range=(0,255))
    plt.imshow(E, cmap=plt.cm.gray)
    plt.title('alpha='+str(alpha))

    imsave('osteoblaste_rehauss_'+str(alpha)+'.python.png', E)

plt.show()
```


(c) $\alpha = 1$.


(d) $\alpha = 0.5$.


(e) $\alpha = 5$.

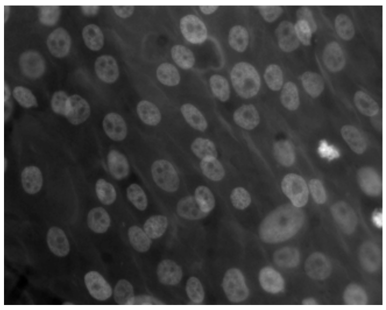

(f) Original image.

Figure 11: Image enhancement: $I = \alpha \cdot I + HP(I)$, where HP is a high-pass filter (the Laplacian filter in these illustrations).

1.8 Aliasing effect

```python
# aliasing effect (Moire)
def circle(fs, f):
    # Generates an image with aliasing effect
    # fs: sample frequency
    # f : signal frequency
    t = np.arange(0,1,1./fs);
    ti,tj = np.meshgrid(t,t);
    C = np.sin(2*np.pi*f*np.sqrt(ti**2+tj**2));
    return C
```

The image of Fig. 12 is generated with the following code.

```python
C = circle(300,50);
plt.imshow(C, cmap=plt.cm.gray);
plt.show()
imsave('moire.png', C);
```

Figure 12: Moiré effect, generated with $f_s = 300$ and $f = 50$.