Université Libre de Bruxelles

INFO-H500 - Image acquisition and processing

Project – Watermark

ELGHAZOUANI Nada

13 novembre 2020

Sommaire

1 Introduction

In order to practice our knowledge and competences in the class *Image acquisition and processing*, we have to realize a project. The project consists of manipulating watermark image and adding it to a photograph.

2 Main requirements

The minimum requirement are to add the white pixels from the watermark in an image. Then, the result should be saved as an image file.

In order to achieve this, we used the following Libraries : Python Imaging Library (PIL), the matplotlib library and sckit image library. Firstly, we open the main image and the watermark image using Image.open() function. Secondly, we paste the watermark image on the other one using paste() function. The conversion of the watermark to 'L' mode is mandatory to remove the background of the image. 'L' mode inform that the image is black and white and stored as 8 bit pixels. Thirdly, we save the image using save() function. Finally, we plot the figure in the notebook using function imread() and imshow(). We note that the default position is x=0 and y=0 in paste() function.

```
from PIL import Image
from matplotlib import pyplot as plt
from skimage.io import imread, imshow
%matplotlib notebook

im= Image.open('paysage.jpg')
watermark = Image.open('watermark.png')
im.paste(watermark,mask=watermark.convert('L'))
im.save('main.jpg')
im = imread('main.jpg')
plt.figure()
imshow(im)
plt.show()
```

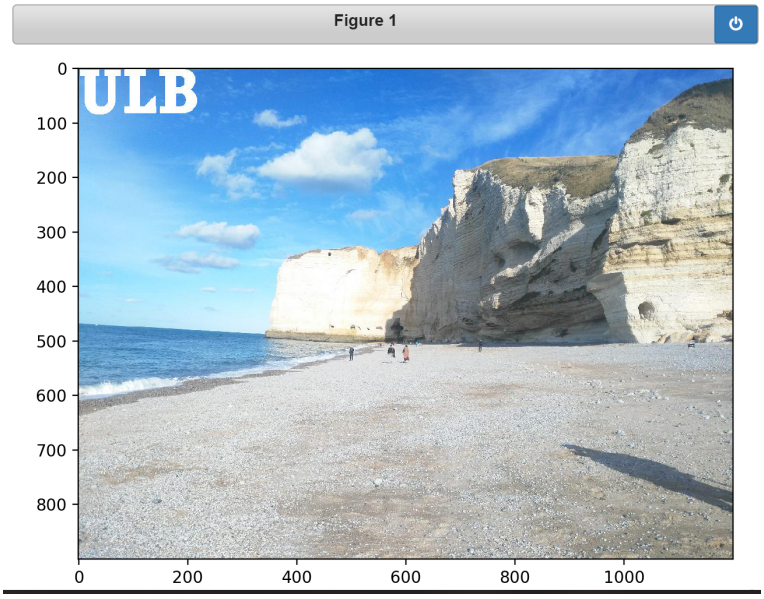

FIGURE 1 – the code and first image displayed in Notebook Jupyter

3 Additional requirements

The additional requirements are some possible improvements to the first method .

3.1 Choosing the watermark location

This option gives the freedom to the user to position the watermark on the image. The function paste(image,position) in PIL library has an optional parameter called "box" that enables to put the watermark in the desired position.

```
from PIL import Image
from matplotlib import pyplot as plt
from skimage.io import imread, imshow
%matplotlib notebook

im= Image.open('paysage.jpg')
watermark = Image.open('watermark.png')
im.paste(watermark,box=(0,800),mask=watermark.convert('L'))
im.save('main_position.jpg')
im = imread('main_position.jpg')
plt.figure()
imshow(im)
plt.show()
```

FIGURE 2 – the code and second image displayed in Notebook Jupyter

3.2 Centring the watermark

The main image chosen by the user could have different size than the one used. So, it could be useful to center the watermark image in the main image . To do so, we get the width and the length of the image using the "shape" function. Then, we divide the values by 2 to get the center. It's important to mind that the fist value given by the shape function is the width but the first value needed for the position is horizontal axis.

```
#adding the watermark in the center of the image
from PIL import Image
from matplotlib import pyplot as plt
from skimage.io import imread, imshow
%matplotlib notebook

im= Image.open('paysage.jpg')
im_array = imread('paysage.jpg')
watermark = Image.open('watermark.png')
#compute the center
centery, centerx = [int (im_array.shape[0] /2) , int (im_array.shape[1] /2)]
im.paste(watermark,box=(centerx,centery),mask=watermark.convert('L'))
im.save('main_centring.jpg')
im = imread('main_centring.jpg')
plt.figure()
imshow(im)
plt.show()
```

FIGURE 3 – the code and third image displayed in Notebook Jupyter

3.3 Transparency effect on the watermark

The transparency effect is enabled by means of putalpha() function. This function adds or replaces an alpha layer to the image. But, it's mandatory to convert the image to "LA" or "RGBA" mode. The A added in the modes is referring to the alpha channel. The argument to use is a constant between O and 255, 0 means 100% transparency and 255 means no transparency.

```
from PIL import Image
from matplotlib import pyplot as plt
from skimage.io import imread, imshow
%matplotlib notebook

im= Image.open('paysage.jpg')
watermark = Image.open('watermark.png').convert('RGBA')
watermark.putalpha(70)
im.paste(watermark,box=(0,800),mask=watermark)
im.save('main_transparent.jpg')
im = imread('main_transparent.jpg')
plt.figure()
imshow(im)
plt.show()
```

FIGURE 4 – the code and fourth image displayed in Notebook Jupyter

We note that the background of the watermark of the image also appear. To solve this problem, we should distinguish the transparency effect from the image. The mask contains a black background and semi-transparent white having the same size as the watermark image. To do so, we create a first new image which contains only black. Then a second new image which contains the value of the white pixels chosen. We know that we handle white and black pixels by dint of the 'L' mode. The arguments in Image.new() function are image mode, image size and the color of the image. When the mask is constituted to reuse the paste() function.

```
from skimage.io import imshow
import matplotlib . pyplot as plt
from PIL import Image

im= Image.open('paysage.jpg')
watermark = Image.open('watermark.png')
im2 = Image.new ('L',watermark.size)
mask = Image.new ('L',watermark.size, 70)
im2.paste(watermark,(0,0),mask)
im.paste(watermark, (800,400),im2)
im.save('main_transparent2.jpg')
im = imread('main_transparent2.jpg')
plt.figure()
imshow(im)
plt.show()
```

FIGURE 5 – the code and fifth image displayed in Notebook Jupyter

3.4 Choosing between dark or light watermark depending on the luminosity of the image

In order to determine the luminosity of an image, we compute the average pixel brightness. To do so, we convert first the image to grey scale then we compute the min and max for each band in the image using Stat() function and *ImageStat()* library. In our case, we have only one band representing the grey scale. We reuse the return values from stat to compute the mean of the image. If the mean is greater than 120, we consider that the image is bright. If the image is bright, we use the inverted watermark using *ImageOps* library and invert() function. The threshold 120 is founded after testing the mean value of several different images.

```python
#we consider a image with mean superior to 120 is a bright image
from skimage.io import imshow
import matplotlib . pyplot as plt
from PIL import Image, ImageStat,ImageOps

im= Image.open('paysage.jpg')
im_grey = Image.open('paysage.jpg').convert('L')
stat = ImageStat.Stat(im_grey)

watermark = Image.open('watermark.png').convert('L')
im2 = Image.new ('L',watermark.size)
mask = Image.new ('L',watermark.size, 90)
im2.paste(watermark,(0,0),mask)

if stat.mean[0] > 120: #we have only one value because we already convert the image to grey
    watermark_invert= ImageOps.invert(watermark)
    im.paste(watermark_invert,box=(200,150),mask=im2)
else :
    im.paste(watermark,box=(200,150),mask=im2)
im.save('main_darkorlight.jpg')
im = imread('main_darkorlight.jpg')
plt.figure()
imshow(im)
plt.show()
```

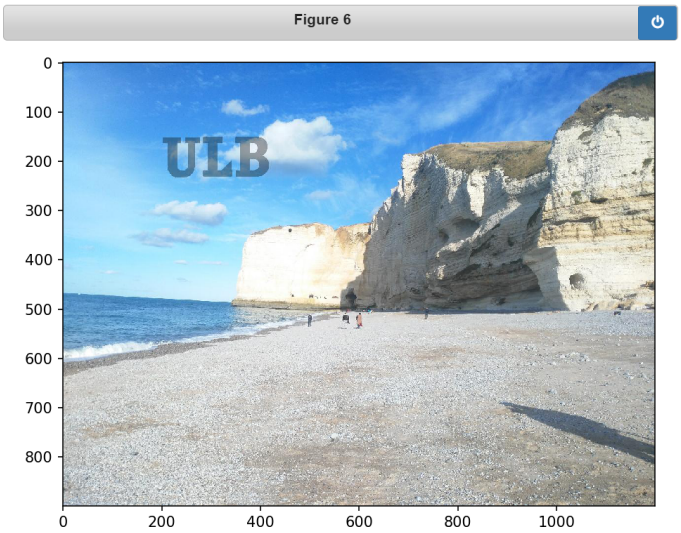

FIGURE 6 – the code and sixth image 6 displayed in Notebook Jupyter

3.5 showing only contour of the watermark

The Sobel filter can be used to show only the contour of the watermark.

```
from PIL import Image
from skimage import filters
import numpy as np
from matplotlib import pyplot as plt
from skimage.io import imread, imshow,imsave

watermark = imread('watermark.png')

fsobel = filters.sobel(watermark)
norm = 255*fsobel/np.max(fsobel)
norm = np.reshape(norm, (85, 219))

sobelwatermark=Image.fromarray(norm)
sobelwatermark.dtype = 'uint8'
imsave('sobelwatermark.png',sobelwatermark)
im= Image.open('paysage.jpg')
im.paste(sobelwatermark,mask=sobelwatermark.convert('L'))
im.save('main_sobel.png')
im = imread('main_sobel.png')
plt.figure()
imshow(im)
plt.show()
```

```
Lossy conversion from float32 to uint8. Range [0.0, 255.0]. Convert image to uint8 prior to saving to suppress this warning.
```

FIGURE 7 – the code and seventh image displayed in Notebook Jupyter