Correction

Exercice 1

```python
import numpy as np
from PIL import Image

def correlation(A, B):
    return np.sum(A * B)

def convolution(A, B):
    return np.sum(A * np.rot90(np.rot90(B)))

def convolution_over_2dimage(image, filter):
    result = np.empty((image.shape[0] - filter.shape[0] + 1, image.shape[1] - filter.shape[1] + 1),
                      dtype=np.int32)

    for x in range(result.shape[0]):
        for y in range(result.shape[1]):
            result[x, y] = convolution(image[x:x+filter.shape[0], y:y+filter.shape[1]], filter)
    return result

def convolution_over_3dimage(image, filter):
    assert len(image.shape) == 3 and image.shape[2] == 3
    result = np.empty((image.shape[0] - filter.shape[0] + 1, image.shape[1] - filter.shape[1] + 1, image.shape[2]),
                      dtype=np.int32)

    for rgb in range(3):
        result[:,:,rgb] = convolution_over_2dimage(image[:,:,rgb], filter)

    return result

A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

box_blur = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])/ 9
sharpen = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])

color_pil = Image.open('in/starry.jpg')
grayscale_pil = color_pil.convert("L")
color_np = np.array(color_pil).astype(np.int32)
gray_np = np.array(grayscale_pil).astype(np.int32)

print(correlation(A, B))
print(convolution(A, B))

gray_np = gray_np.astype(np.uint8)
gray_blur = convolution_over_2dimage(gray_np, box_blur).astype(np.uint8)
gray_sharpen = convolution_over_2dimage(gray_np, sharpen).clip(0,255).astype(np.uint8)
```

```python

color_np = color_np.astype(np.uint8)
color_blur = convolution_over_3dimage(color_np, box_blur).astype(np.uint8)
color_sharpen = convolution_over_3dimage(color_np, sharpen)
color_sharpen = color_sharpen.clip(0,255).astype(np.uint8)

first_row =np.vstack( [
    np.repeat(gray_np[1:-1, 1:-1, np.newaxis], 3, axis=2),
    np.repeat(gray_blur[:, :, np.newaxis], 3, axis=2),
    np.repeat(gray_sharpen[:, :, np.newaxis], 3, axis=2)
])
second_row =np.vstack([
    color_np[1:-1,1:-1],
    color_blur,
    color_sharpen
])
Image.fromarray(np.hstack([first_row, second_row])).save('out/exo01.jpg')
```

Mini-projet : Détection de contours

```python
import numpy as np
from PIL import Image
from scipy.signal import convolve2d


def grayscale(numpy_image):
    # Formule NTSC
    return (numpy_image[:, :, 0] * 0.299 + numpy_image[:, :, 1] * 0.587 + numpy_image[:,
     :, 2] * 0.114).astype(np.uint8)


def apply_blur(numpy_image):
    box_blur = np.array([
        [1, 2, 1],
        [2, 4, 2],
        [1, 2, 1]]
    ) / 16 # Filtre gaussien 3x3
    return convolve2d(numpy_image, box_blur).astype(np.uint8)

def sobel_analysis(numpy_image):
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    sobel_x_np = convolve2d(numpy_image, sobel_x, mode='valid')
    sobel_y_np = convolve2d(numpy_image, sobel_y, mode='valid')

    # Formules données dans l'énoncé
    magnitude = np.sqrt(sobel_x_np**2 + sobel_y_np**2)
    gradient = np.arctan2(sobel_y_np, sobel_x_np)

    return sobel_x_np, sobel_y_np, magnitude, gradient

def non_maximum_suppression(magnitude, grad):
    res = np.zeros(magnitude.shape, dtype=np.int32)

    # Pour chaque pixel qui n'est pas sur les bords
    for i in range(1, magnitude.shape[0] - 1):
```

```python
        for j in range(1, magnitude.shape[1] - 1):

            premier_voisin = 255
            second_voisin = 255

            # Gradient horizontal
            if (-np.pi/8 <= grad[i, j] < np.pi/8) or (7*np.pi/8 <= grad[i, j] <= np.pi) or (-np.pi <= grad[i, j] < -7*np.pi/8):
                premier_voisin = magnitude[i, j - 1]
                second_voisin = magnitude[i, j + 1]

            # Gradient diagonal bas gauche / haut droit
            elif (np.pi/8 <= grad[i, j] < 3 * np.pi/8) or (-7*np.pi/8 <= grad[i, j] < -5*np.pi/8):
                premier_voisin = magnitude[i - 1, j + 1]
                second_voisin = magnitude[i + 1, j - 1]

            # Gradient vertical
            elif (3 * np.pi/8 <= grad[i, j] < 5 * np.pi/8) or (-5*np.pi/8 <= grad[i, j] < -3*np.pi/8):
                premier_voisin = magnitude[i - 1, j]
                second_voisin = magnitude[i + 1, j]

            # Gradient diagonal haut gauche / bas droit
            elif (5 * np.pi/8 <= grad[i, j] < 7 * np.pi/8) or (-3*np.pi/8 <= grad[i, j] < -np.pi/8):
                premier_voisin = magnitude[i + 1, j + 1]
                second_voisin = magnitude[i - 1, j - 1]

            # Si le pixel est plus grand que ses voisins dans le sens du gradient, on le garde, sinon on le met à 0
            if magnitude[i, j] >= premier_voisin and magnitude[i, j] >= second_voisin:
                res[i, j] = magnitude[i, j]
            else:
                res[i, j] = 0
    return res

def threshold(img, seuil):
    res = np.zeros(img.shape, dtype=np.int32)

    res[np.where(img >= seuil)] = 255
    res[np.where(img < seuil)] = 0

    return res

# Lecture de l'image et enchaînement des fonctions définies dans la donnée
image_np = np.array(Image.open('in/go.jpg'))
gray_np = grayscale(image_np)
blurred_np = apply_blur(gray_np)
sobel_x_np, sobel_y_np, magnitude_np, gradient_np = sobel_analysis(blurred_np)
nms_np = non_maximum_suppression(magnitude_np, gradient_np)
print(np.unique(nms_np))
thresholded = threshold(nms_np, 200)

# Visualisation
sobel_x_np = ((sobel_x_np - sobel_x_np.min()) / (sobel_x_np.max() - sobel_x_np.min()) * 255).astype(np.uint8)
sobel_y_np = ((sobel_y_np - sobel_y_np.min()) / (sobel_y_np.max() - sobel_y_np.min()) * 255).astype(np.uint8)
```

```python
magnitude_np = ((magnitude_np - magnitude_np.min()) / (magnitude_np.max() - magnitude_np.min()) * 255).astype(np.uint8)
gradient_np = ((gradient_np - gradient_np.min()) / (gradient_np.max() - gradient_np.min()) * 255).astype(np.uint8)
nms_np = ((nms_np - nms_np.min()) / (nms_np.max() - nms_np.min())*255).astype(np.uint8)
thresholded = ((thresholded - thresholded.min()) / (thresholded.max() - thresholded.min()) * 255).astype(np.uint8)

Image.fromarray(gray_np).save('out/exo02_gray.jpg')
Image.fromarray(blurred_np).save('out/exo02_blurred.jpg')
Image.fromarray(sobel_x_np).save('out/exo02_sobelx.jpg')
Image.fromarray(sobel_y_np).save('out/exo02_sobely.jpg')
Image.fromarray(magnitude_np).save('out/exo02_magnitude.jpg')
Image.fromarray(gradient_np).save('out/exo02_gradient.jpg')
Image.fromarray(nms_np).save('out/exo02_nms.jpg')
Image.fromarray(thresholded).save('out/exo02_thresholded.jpg')
```