Midterm 1
Assignment 6

Alessandro Ristori

Intelligent Systems for Pattern Recognition
Master Degree in Computer Science, AI Curriculum
A.Y. 2020/2021

Edge Detection

Kernels

```python
class RobertsKernel(Kernel):

    def __init__(self):
        k_x = np.array(([1, 0], [0, -1]))
        k_y = np.array(([0, -1], [1, 0]))
        super(RobertsKernel, self).__init__(2, k_x, k_y, "Roberts_Kernel")

    def kernel_pad(self, m, pad_zero: bool):
        if pad_zero:
            padded_m = np.pad(m, [(0, 1), (0, 1)], mode='constant')
        else:
            padded_m = np.concatenate((m, m[-1, :].reshape(1, m.shape[1])), 0)
            padded_m = np.concatenate((padded_m, padded_m[:, -1].reshape((padded_m.shape[0]), 1)), 1)

        return padded_m

class PrewittKernel(Kernel):

    def __init__(self):
        k_size = 3
        k_x = np.array(([1, 0, -1], [1, 0, -1], [1, 0, -1]))
        k_y = k_x.T
        super(PrewittKernel, self).__init__(k_size, k_x, k_y, "Prewitt_Kernel")

    def kernel_pad(self, m, pad_zero: bool):
        return super(PrewittKernel, self).kernel_pad(m, pad_zero)


class SobelKernel(Kernel):

    def __init__(self):
        k_size = 3
        k_x = np.array(([1, 0, -1], [1, 0, -1], [1, 0, -1]))
        k_x[1, :] *= 2
        k_y = k_x.T
        super(SobelKernel, self).__init__(k_size, k_x, k_y, "Sobel_Kernel")

    def kernel_pad(self, m, pad_zero: bool):
        return super(SobelKernel, self).kernel_pad(m, pad_zero)
```

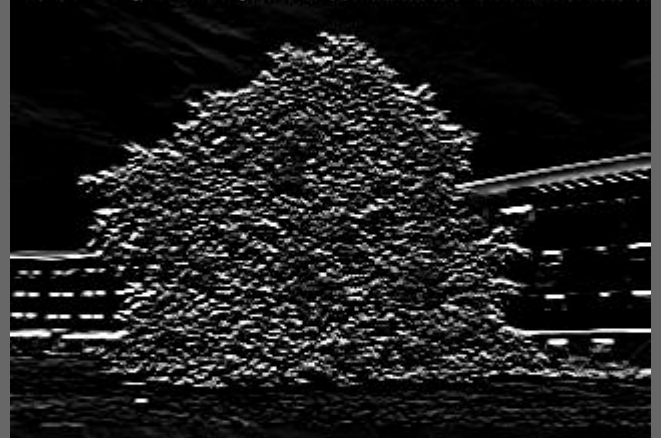
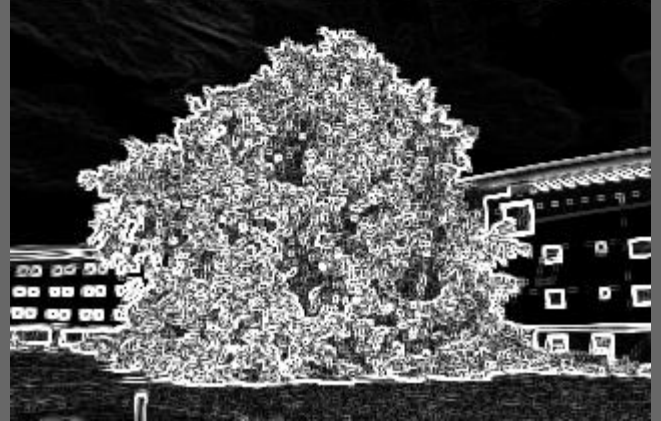
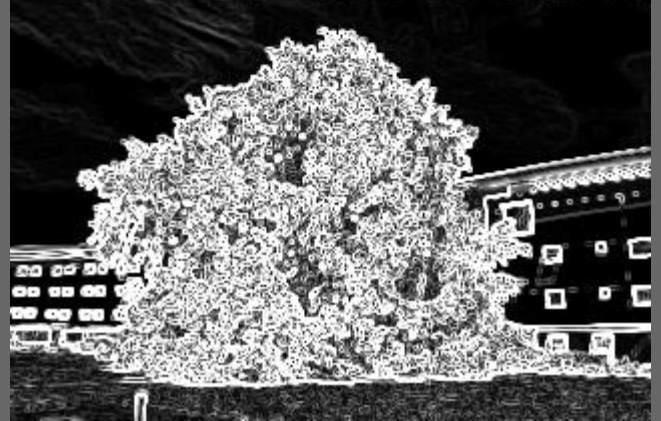
Convolution

```python
@staticmethod
def __convolution(image, kernel):
    k = kernel.k_size
    i_x = np.zeros((image.shape[0] - k + 1, image.shape[1] - k + 1))
    i_y = np.zeros((image.shape[0] - k + 1, image.shape[1] - k + 1))
    for i in range(len(i_x)):
        for j in range(len(i_x[0])):
            i_x[i, j] = np.sum(np.multiply(image[i:i + k, j:j + k], kernel.k_x))
            i_y[i, j] = np.sum(np.multiply(image[i:i + k, j:j + k], kernel.k_y))

    return i_x, i_y

def compute(self, image_path, kernel, pad_zero: bool, image_processor):
    image_gray = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2GRAY)
    if image_processor is not None:
        image_gray = image_processor.apply(image_gray)  # if we'd like to apply a blur effect

    image_gray = kernel.kernel_pad(image_gray, pad_zero)
    i_x, i_y = self.__convolution(image_gray, kernel)
    magnitude = np.sqrt(i_x**2+i_y**2)
    return i_x, i_y, magnitude
```

Tree	Roberts	Prewitt	Sobel
Gx			
Gy			
Magn.			

Face	Roberts	Prewitt	Sobel
Gx			
Gy			
Magn.			

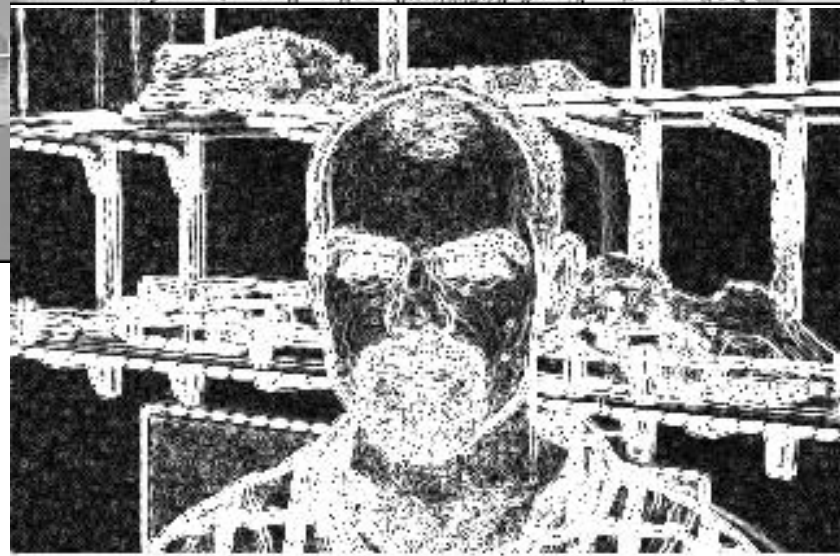
What if we blur or sharpen the initial image?

Gaussian Blur $\begin{matrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{matrix}$

Applying the blur or the sharperner before or after the convolution doesn't change the final result

Sharperner $\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$

It seems that we get better results by applying the blur, but we can't say the same for the sharpener… i'm not an expert by the way



Final considerations and improvements

- Padding is not necessary
- Convolution can be done more efficiently and quickly with parallellization
- Kernels of bigger sizes give a worse result than smaller ones, the image becomes more confused/hazy
- Image's borders can be ignored, they are not very meaningful in the grand scheme of things
- Using a bigger blur processor reduces the image's noise, with the consequence of detecting only the most important edges

Thanks for your attention