TEDS20_T0138

-

Assignment 1

Members:

- Adrien Guezennec

Table of contents

Table of contents 2

Introduction 3

Question 1 4

Question 2 8

Introduction

The main purpose of this assignment is AI for computer vision. The assignment is divided into 2 part, first one is for edge detection based on an image in input, and the second one is recognizing and counting people in an image.

For the first one, we only constraint that we had was that we had to implement the hysteresis threshold from scratch by our self. For the rest, like image processing, we could use any library wanted.

In the second part, the constraint was more open and we could either create our own algorithm or use a pre-trained model already existing.

Question 1

After some research, I found out that there were different famous algorithms for edge detection. The Canny, is one of them. I based my work on the Canny edge detection.
For the small history, the Canny edge detector is an invention from Mr. Canny around 1986 using a multi-stage algorithm to detect a wide range of edges in an image.

Based on the Wikipedia page for the Canny edge detector, I cut my Canny edge detector in 5 steps.

- Transform image in grayscale + denoising
- Apply a Gaussian filter to blur the image
- Apply the Sobel algorithm to get the main edges
- Non-maximum suppression to reduce and thin the edges
- Hysteresis threshold for filtering essential pixel and edges

In the normal Canny edge detector, there is no denoising of the image but I founded it interesting to add it. In fact, I didn't notice real changes (with and without) but this is due to my image which was already clean. Till the Sobel filter, I use the Opencv library which allows me to go faster and skip the processing step and focus on the hysteresis threshold + the non-maximum suppression algorithm.

To launch the Python script you need to give an image as parameter with the flag -i or **--image**.

Here is my NMS algorithm:

```
# Non max supression to reduce the thickness of the edges.
def nonMaximumSupression(image, angle):
    # Get image shape.
    y, x = image.shape
    # Create new image based on the shape of the input image.
    newImage = np.zeros((y, x))

    # Loop through Y abscisse. -1 otherwise i'm out of bounce.
    for indexY in range(0, y - 1):
        # Loop through X abscisse. -1 otherwise i'm out of bounce.
        for indexX in range(0, x - 1):
            point1 = None
            point2 = None

            # Check on horizontal direction.
            if (0 <= angle[indexY, indexX] < 22.5) or \
            (157.5 <= angle[indexY, indexX] <= 180) or \
```

```
            (-22.5 <= angle[indexY, indexX] < 0) or \
            (-180 <= angle[indexY, indexX] < -157.5):
                point1 = image[indexY, indexX + 1]
                point2 = image[indexY, indexX - 1]
            # CHeck on angle 45°.
            elif (22.5 <= angle[indexY, indexX] < 67.5) or (-157.5 <= angle[indexY,
indexX] < -112.5):
                point1 = image[indexY + 1, indexX + 1]
                point2 = image[indexY - 1, indexX - 1]
            # Check on angle 90°
            elif (67.5 <= angle[indexY, indexX] < 112.5) or (-112.5 <= angle[indexY,
indexX] < -67.5):
                point1 = image[indexY + 1, indexX]
                point2 = image[indexY - 1, indexX]
            # Check on angle 135°
            elif (112.5 <= angle[indexY, indexX] < 157.5) or (-67.5 <= angle[indexY,
indexX] < -22.5):
                point1 = image[indexY + 1, indexX - 1]
                point2 = image[indexY - 1, indexX + 1]

            # If image pixel has a higher intensity than point1 and point2, then we
keep the pixel as a edge.
            if (image[indexY, indexX] >= point1) and (image[indexY, indexX] >=
point2):
                newImage[indexY, indexX] = image[indexY, indexX]
            # In the other way we remove the pixel. Not really remove but set to
black.
            else:
                newImage[indexY, indexX] = 0
    return newImage
```

I didn't found an NMS on the Opencv library so I had to make my own. I struggle a lot to understand what to do and how it works, but I found these articles which helped me a lot:

- Canny Edge and Line Detection
- Opencv Canny doc

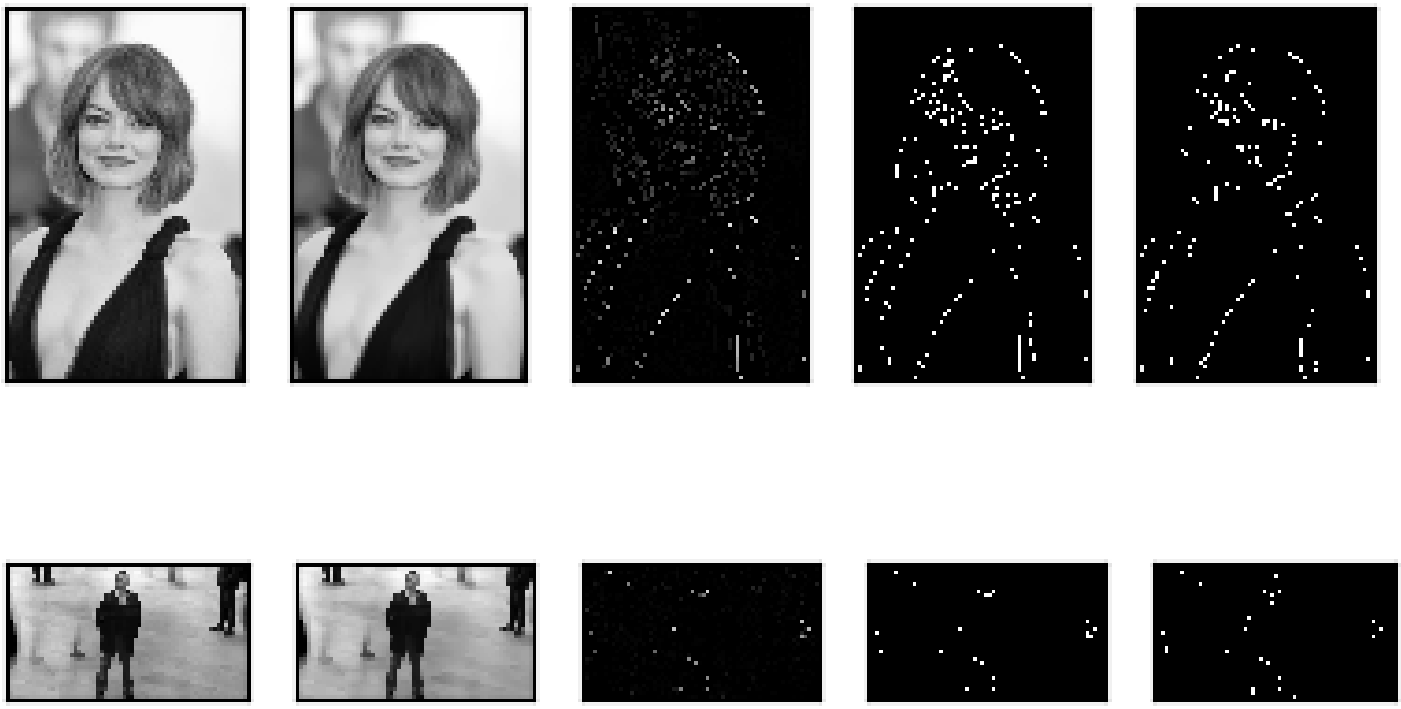
Now my hysteresis threshold:

```
def hysteresisThresholding(image, weak=10, strong=70):
    # Get image shape.
    y, x = image.shape
    # Create a new image and initial it at 0 everywhere based on the image input
shape.
    finalImage = np.zeros((y, x))

    # Thanks to numpy, get pixel lower than my weak threshold point.
    weakX, weakY = np.where(image < weak)
    # Same as above but other way around, get pixel higher than my strong threshold
point.
    strongPixelX, strongPixelY = np.where(image >= strong)
    # Get pixel in between, this is gonna be my weak pixels to improve.
    mixedX, mixedY = np.where((image <= strong) & (image >= weak))

    # Standardization of weak, strong, and my weak pixels to the same color.
    finalImage[weakX, weakY ] = 0
    finalImage[strongPixelX, strongPixelY] = 255
    finalImage[mixedX, mixedY] = 75

    # Loop through the Y abscisse.
    for indexY in range(0, y):
        # Loop through the X abscisse.
        for indexX in range(0, x):
            # If the point is a mixed point (in-between).
            if (finalImage[indexY, indexX] == 75):
                # Check all around the point if the pixel is a strong one (white),
then add a white pixel into my final image.
                if 255 in [finalImage[indexY + 1, indexX - 1],
                finalImage[indexY + 1, indexX],
                finalImage[indexY + 1, indexX + 1],
                finalImage[indexY, indexX - 1],
                finalImage[indexY, indexX + 1],
                finalImage[indexY - 1, indexX - 1],
                finalImage[indexY - 1, indexX],
                finalImage[indexY - 1, indexX + 1]]:
                    finalImage[indexY, indexX] = 255
                # Otherwise put the pixel in black.
                else:
```

```
                finalImage[indexY, indexX] = 0
    # Return the result after hysterisis thresholding.
    return finalImage
```

I took the profit of the Numpy library with the method where() to reduce the complexity of the algorithm. In any case, I struggle less than the NMS and the loop is a lot easier to implement.

To conclude here is the result of 2 different images:



Explanation of the images from left to right:

1. The image in a grayscale
2. The image after Gaussian blur
3. The image after NMS
4. The final result of my hysteresis threshold
5. The result of the Opencv Canny algorithm to compare

Sorry for the quality, I recommend you to test directly on your computer because the screenshot isn't that well.

In conclusion, I'm pretty happy with the result but it was hard to implement and understand how to manipulate an image. OpenCV uses the Numpy library so it makes it easier to manipulate the image since I'm used to the Numpy library.

Question 2

Here we had to identify and count people in an image. I wanted to push the project a little further and use the webcam in real-time for identifying and counting.
I use the Opencv library again for the streaming, create boxes around people, and draw the image with the boxes. The more I was searching how to do it, I discover a library called Tensornets. Tensornet provides a lot of pre-trained neural network written with Tensorflow that you could use. I choose to use the YOLOV3 neural network to identify people in the image. YOLOV3 is a perfect algorithm for this kind of task.
To launch and try the script you can easily install the library needed and run either the script without any argument (this will automatically use the webcam) or give an image in input with the flag **-i** or **--image.** With an image, this will display the image with the boxes around the people for 5 seconds and then close the program.

For this task, I struggle to use Tensorflow at the beginning because I had version issues so I decided to go with Tensornet and use the v1 method of Tensorflow. Using the camera with Opencv was completely new for me and thanks to Opencv this is really easy to implement and use.
On the contrary, creating the boxes around the people was a bit confusing but I manage to do it even if it is not perfect around. The webcam is a bit laggy, don't know if it is because my laptop is a bit old now but creating an image every loop round might be the issue.
I'm gonna focus on the streaming flow to have something fluent and maybe change to another library, I need to dig more into it.

The code is commented so please take a look at the code if you want the details.

Here you can find the reference that I use to help me for this task:

- YOLO: Real-Time Object Detection
- OpenCV: cv::VideoCapture Class Reference
- YOLOv3: An Incremental Improvement