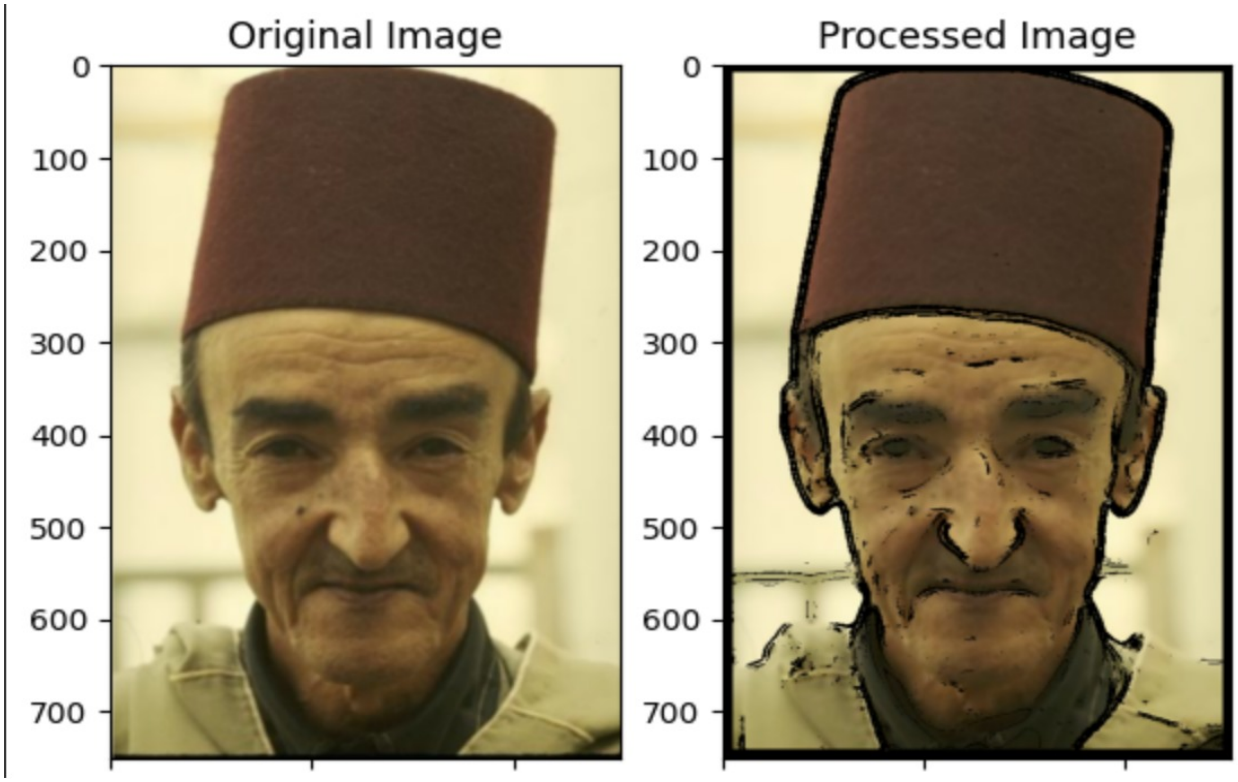
Omar Salah Abdelkader **6809**
Youssef Yousry El-Beltagy **6953**
Aly Mohamed Nasr **6835**

Assignment 1

Under supervision of DR Marwan Torki

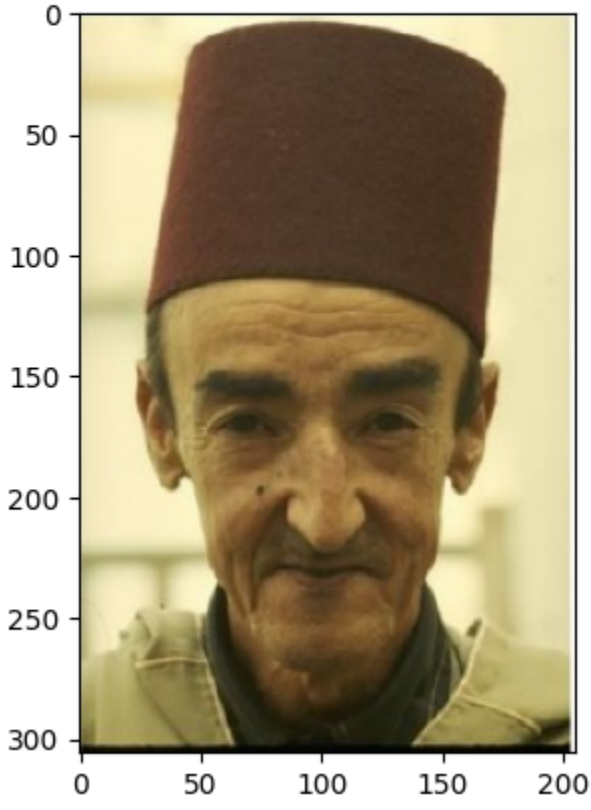


Importing the Libraries:

```
import sys
import cv2
import PIL
import math
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
```

The image is read as BGR then converted to RGB to be suitable for processing with the bilateral filter, the image is also read as grayscale to be suitable for processing with Laplacian filter.

```
img = cv2.imread("original.jpg")
img_grayscale = cv2.imread("original.jpg", cv2.IMREAD_GRAYSCALE)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img);
```



Part I: Applying Image Processing Filters For Image Cartoonifying

Laplacian filter is used to detect the sharp edges in the image, after being smoothed by a median filter, and bilateral filter is used smooth the image and give it a blurring effect, overlaying the two filtered images results in a cartoonified image

Trying Different Filter combinations:

```
laplacian_filter1 = np.array([[0, 0, -1, 0, 0],
                              [0, -1, -2, -1, 0],
                              [-1, -2, 17, -2, -1],
                              [0, -1, -2, -1, 0],
                              [0, 0, -1, 0, 0]])

laplacian_filter2 = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])

laplacian_filter3 = np.array([[-1, -1, -1, -1, -1],
                              [-1, -1, -1, -1, -1],
                              [-1, -1, 24, -1, -1],
                              [-1, -1, -1, -1, -1],
                              [-1, -1, -1, -1, -1]])

laplacian_filter = np.array([[1, 1, 1, 1, 1, 1, 1],
                             [1, 1, 1, 1, 1, 1, 1],
                             [1, 1, 1, 1, 1, 1, 1],
                             [1, 1, 1, -48, 1, 1, 1],
                             [1, 1, 1, 1, 1, 1, 1],
                             [1, 1, 1, 1, 1, 1, 1],
                             [1, 1, 1, 1, 1, 1, 1],])
```

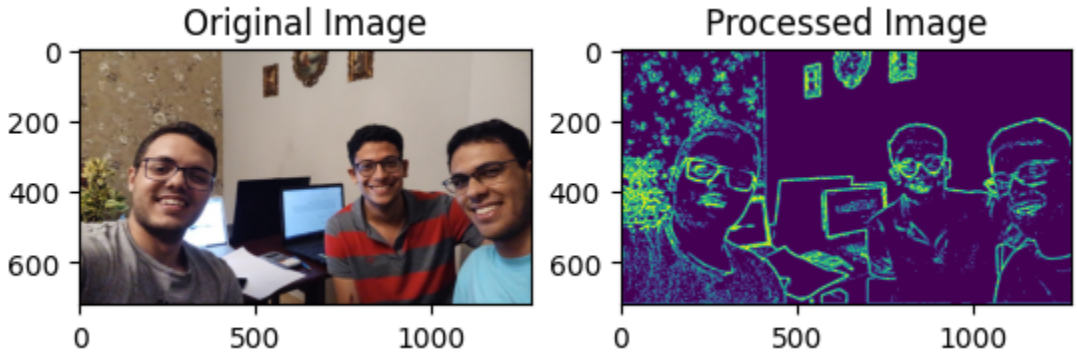
Step By Step

First of all we need to convolve the image with a bilateral filter to get the blurred image, then convolve the image with a median filter of filter size 3, then applying the laplacian filter and overlaying the blurred and the edge detected image gives the following results:

```
blurred = convolveBillateral(img, filter_size=9)
```

```
subplot(img, blurred)
```

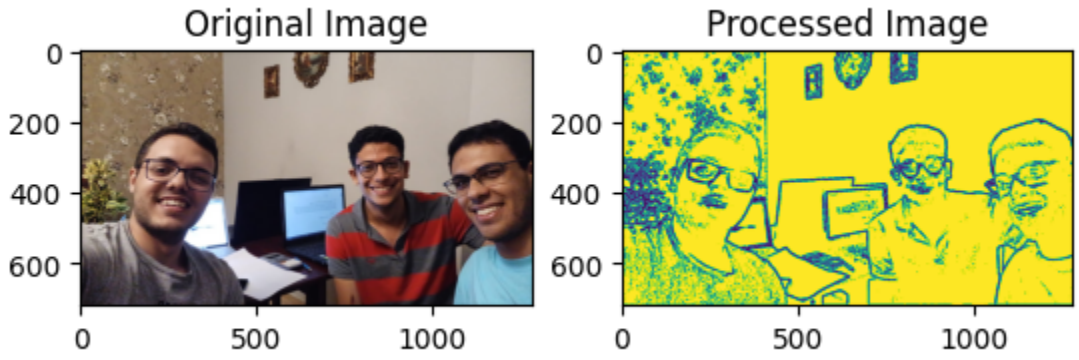
```
edge = convolveMedian(img_grayscale, filter_size=3)
edge = convolveLaplacian(edge, laplacian_filter)
subplot(img, edge)
```



However to overlay the image filtered by bilateral and the image filtered by laplacian to get effective results we need to invert the image, convert the white edges to be black and the black background to be white:

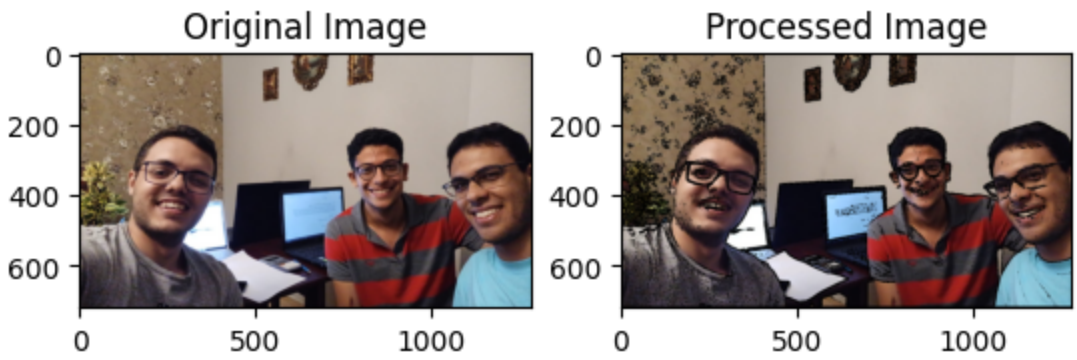
Inverting the image:

```
edge = invert_img(edge)
subplot(img, edge)
```



Overlaying the image:

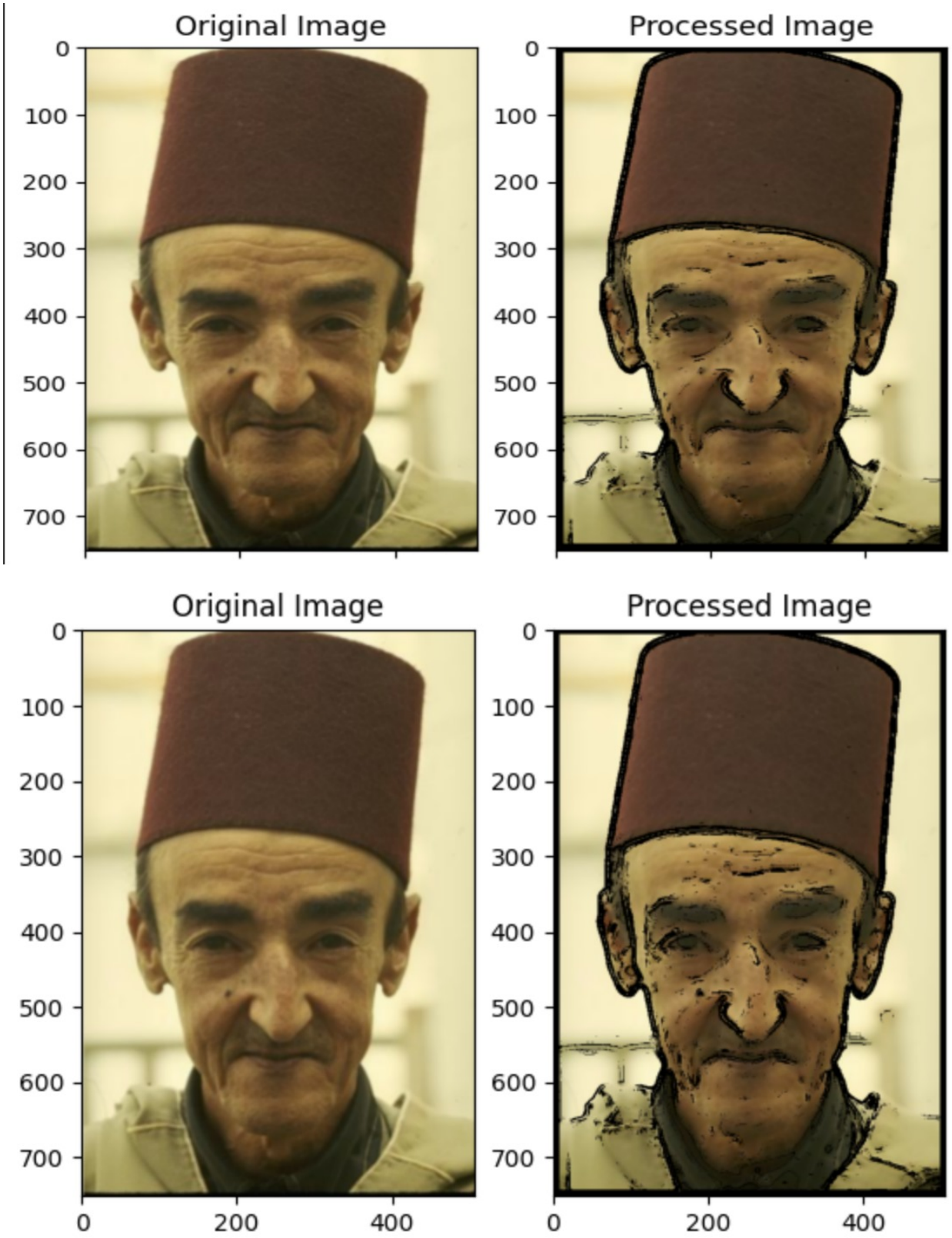
```
cartoonized = overlay(blurred, edge)
subplot(img, cartoonized)
```



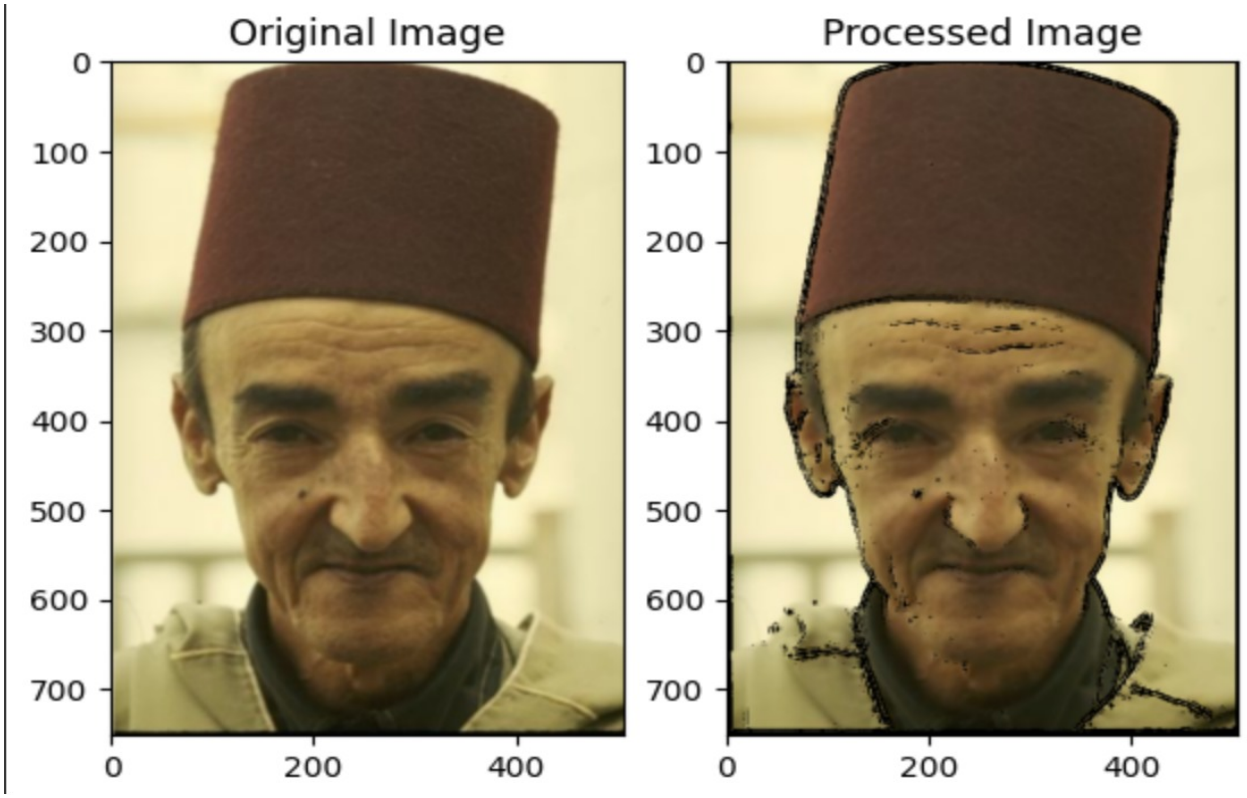
Getting all of these steps in a single function:

```
def cartoonize(img_path, filter, filter_size):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_grayscale = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    blurred = convolveBillateral(img, filter_size)
    edge = convolveMedian(img_grayscale, filter_size=3)
    edge = convolveLaplacian(edge, filter)
    edge = invert_img(edge)
    return img, overlay(blurred, edge)
```

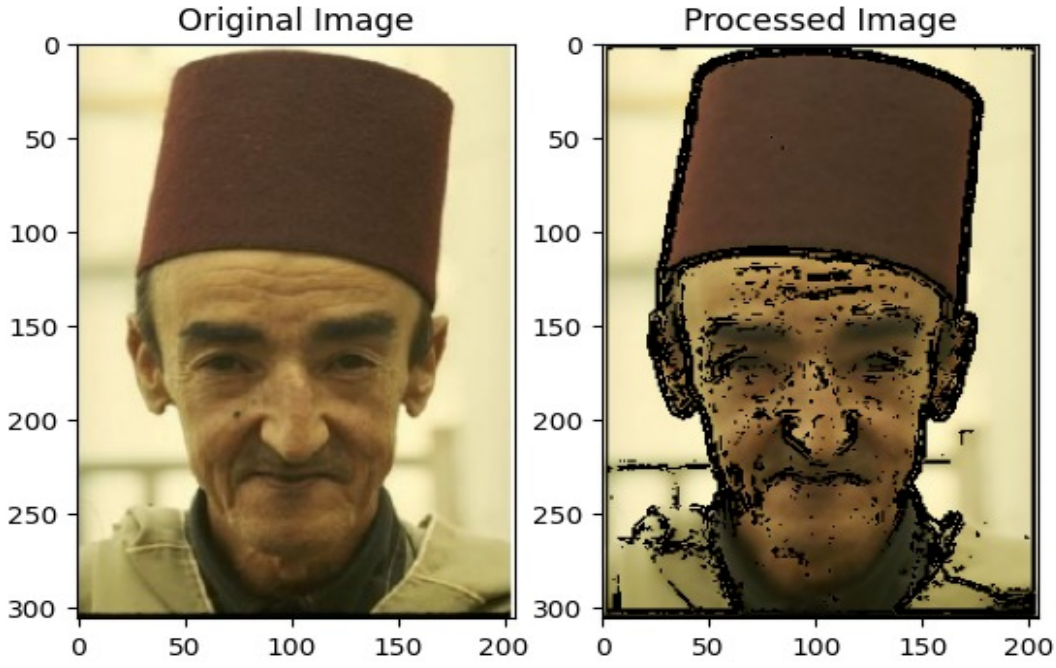
Now we can try different filters with different kernels to observe the output:
Best Results so far:



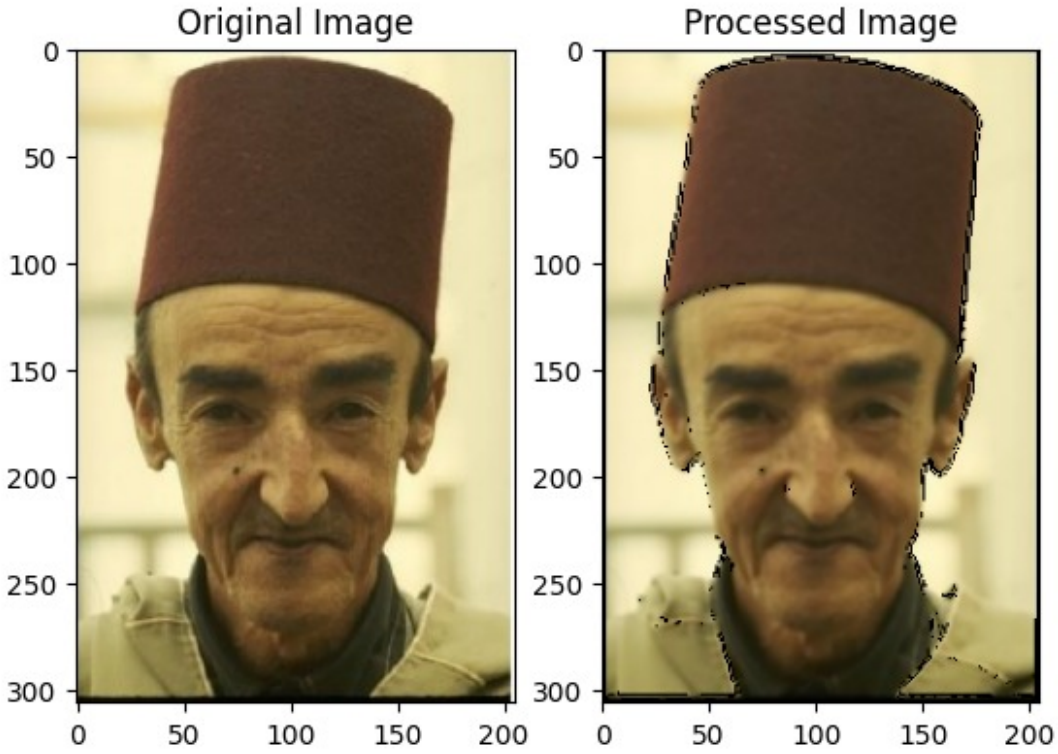
Other results:



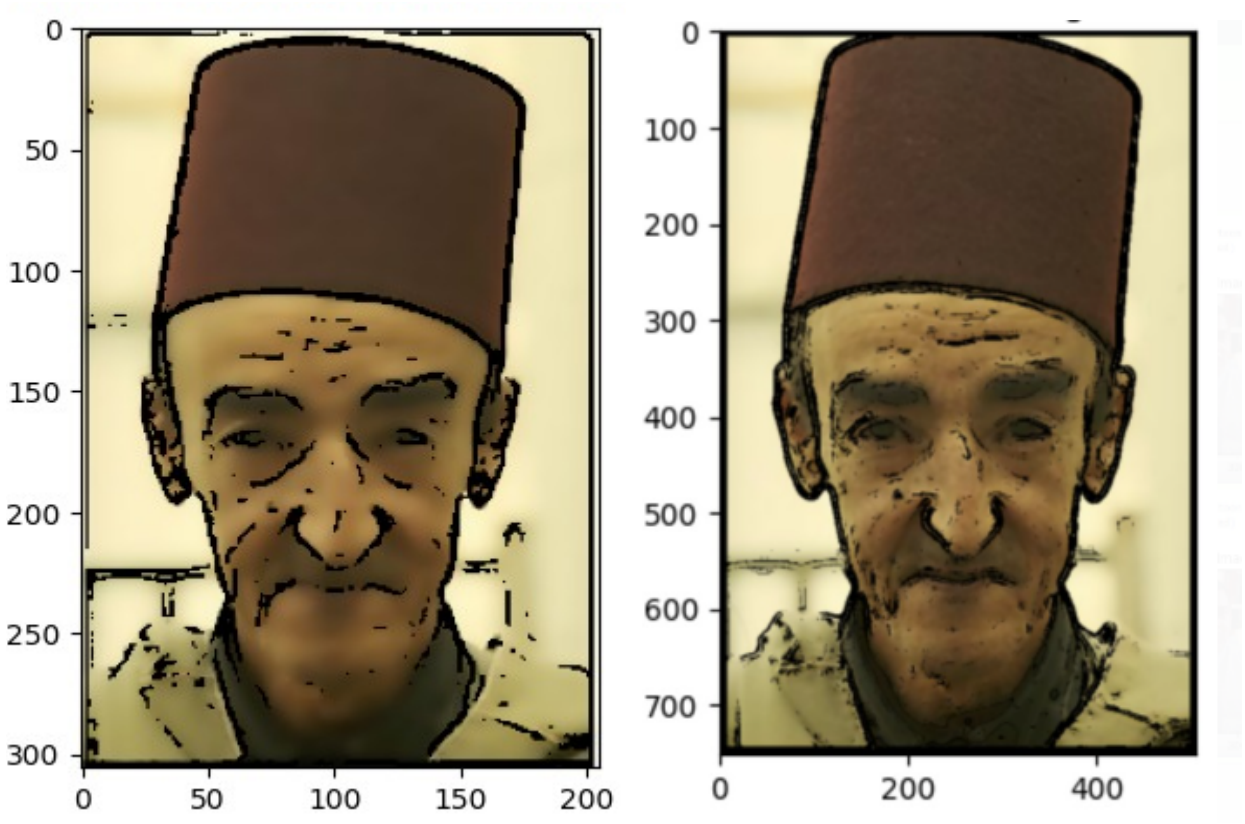

```
[61] img, cartoonized = cartoonize("/content/original.jpg", laplacian_filter3)
     subplot(img, cartoonized)
```



```
[60] img, cartoonized = cartoonize("/content/original.jpg", laplacian_filter2)
     subplot(img, cartoonized)
```



Comparing the output with the output from the library:



Part II: Road Lane Detection Using Hough Transform

First step was to apply Canny's algorithm to do that we applied a median filter to the image then convolved the image twice, once with the sobel filter and the second time with the transpose of the sobel filter to get the gradients in the x and y direction.

Then the gradient magnitude is calculated according to the following formula:

```
grad_mag = np.sqrt((new_img_x ** 2) + (new_img_y ** 2))
```

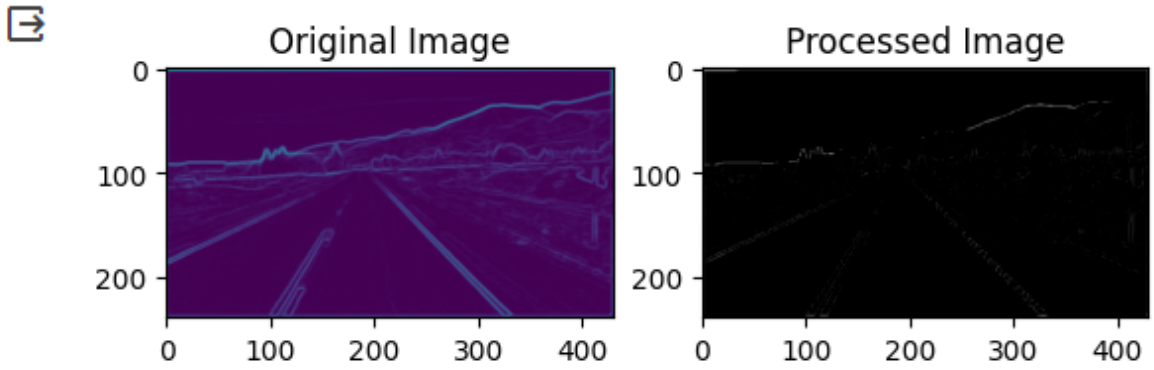
And the gradient phase was measured as follows

```
grad_angle = (new_img_y / (new_img_x + sys.float_info.epsilon))
```

Epsilon is used to avoid division by zero

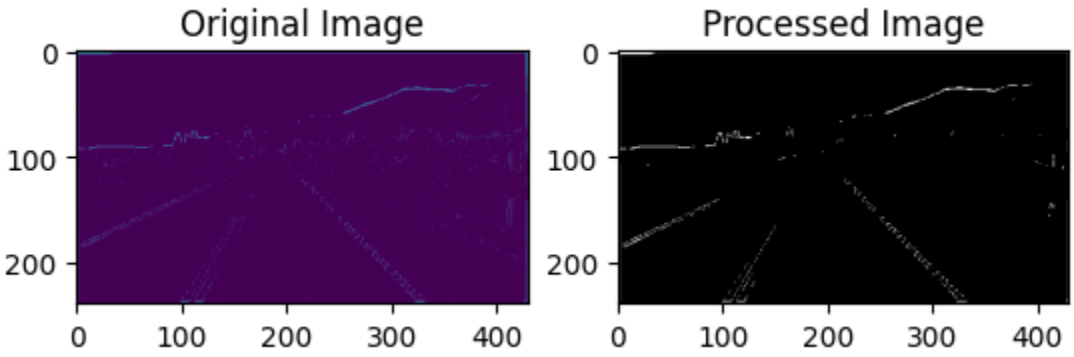
Then non-max suppression was used to obtain the following result

```
subplot(grad_mag, non_max_output)
```

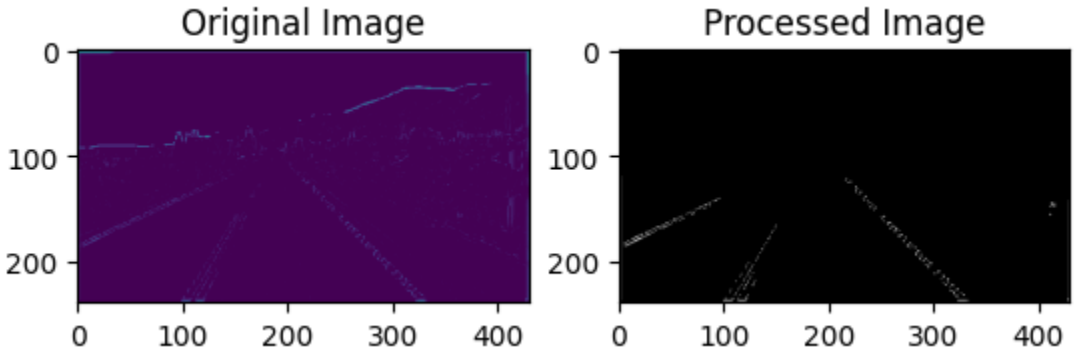


Then we applied thresholding and hysteresis to obtain the following result

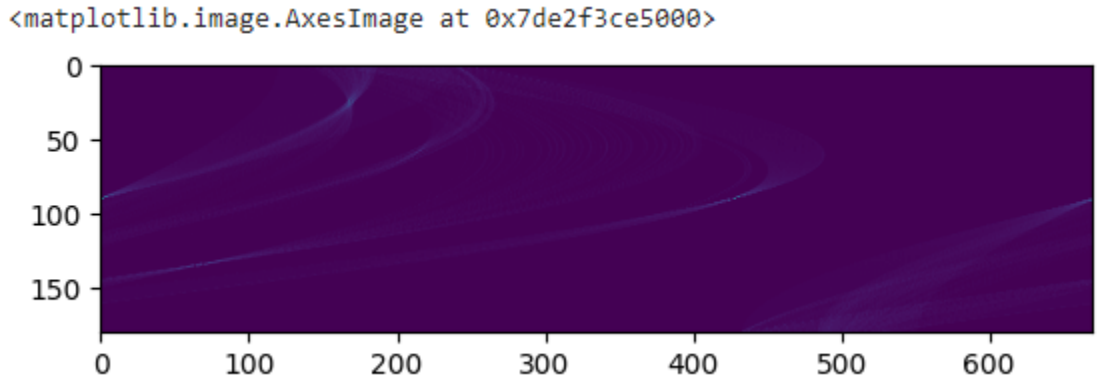
```
res, weak, strong = threshold(non_max_output, lowThresholdRatio=5,
highThresholdRatio=0.18)
hys_img = hysteresis(res, weak, strong)
subplot(non_max_output, hys_img)
```



The image was masked to retain only the region of interest, resulting in removing approximately the upper half of the image



Then hough transform was applied to the resulting image to get the following sine waves:



From the intersections we were able to retain the most significant straight lines to plot on top of the image resulting in the following shape