Computer Vision

(Assignment 1 Report)

Team Members:

Abdelaziz Mohamed Khalil	7166
Abdelaziz Mohamed Rahwan	6834
Habiba Khaled ElMazahy	6796

Course Instructor:

Dr. Marwan Torki

Date: Nov 3rd, 2023

Part I: Image Cartoonifying

Code Description:

Our code in the first part processes the input image to create a cartoon-like effect by applying a series of filters and transformations, resulting in edge detection, smoothing, and color enhancement. The final output image combines the filtered image with the detected edges to create a cartoon-like appearance. Let's go through the code step by step:

1. The input image is read using cv2.imread() function, which reads the image in the BGR format and assigns it to the variable img_BGR.

2. The image is then converted from BGR to RGB format using cv2.cvtColor() function and stored in the variable img_RGB.

3. The RGB image is converted to grayscale using cv2.cvtColor() function with the argument cv2.COLOR_BGR2GRAY, and the result is stored in the variable img_Grayscale.

4. A function median_filter() is defined to apply a median filter to the grayscale image. This function takes the grayscale image and the size of the filter as inputs. It applies the median filter by iterating over each pixel and calculating the median value of the neighboring pixels within the filter window. The resulting filtered image is returned.

5. The median_filter() function is called with the img_Gray (grayscale image) and a filter size of 7.

6. Another function conv2D() is defined to perform a 2D convolution operation on an image using a given filter. This function takes the input image and the filter as inputs. It pads the image with zeros, creates a zeroed array for the filtered image, and then iterates over each pixel of the image to apply the filter. The convolution operation involves taking the sum of the element-wise multiplication of the filter and the corresponding image region. The resulting filtered image is returned.

7. A Laplacian filter is defined as a 2D numpy array and stored in the variable laplacian_filter. It is a 7x7 filter with a specific pattern of values representing the Laplacian operator.

8. The conv2D() function is called with the img_Median (output of median filtering) and the laplacian_filter to apply the Laplacian filter on the median-filtered image. The absolute values of the filtered image are stored in img_Laplacian.

9. The Laplacian filtered image is then thresholded to create a binary image. The thresholding operation sets the pixel values above a threshold (128) to 0 (black) and the values below the threshold to 255 (white). The resulting binary image is stored in img_Threshold.

10. A function bilateral_filter() is defined to apply a bilateral filter to the RGB image. This function takes positional strength, color strength, filter size, and the RGB image as inputs. It iterates over each pixel of the image, calculates the Euclidean distances and intensity differences between the central pixel and its neighbors, and applies the bilateral filter operation. The resulting filtered image is returned.

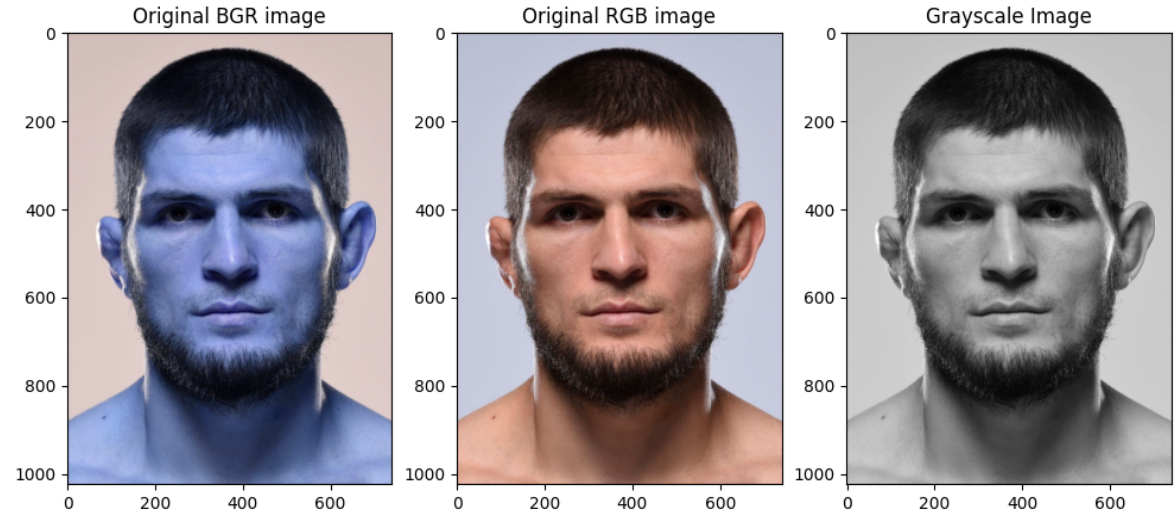
11. Several parameters are set for the bilateral filter: repetition_count (number of times to apply the bilateral filter), filter_size (size of the bilateral filter), color_strength (color strength of the filter), and positional_strength (positional strength of the filter).

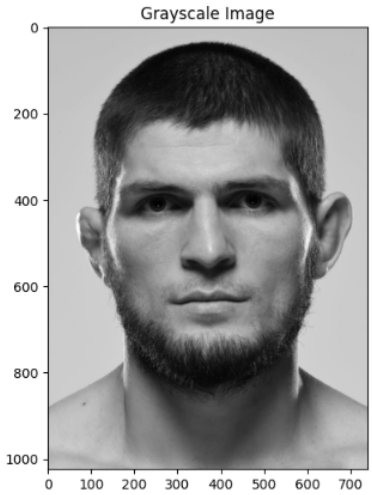
12. The bilateral filter is applied repetition_count number of times to the img_Bilateral using the bilateral_filter() function.

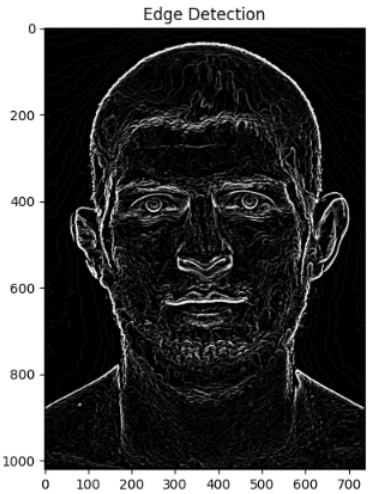
13. An edge mask is created by copying the pixel values from the img_Bilateral to img_Cartoon only for the pixels that are not black (edges). The mask is determined by the img_Threshold, where non-zero values indicate edges.

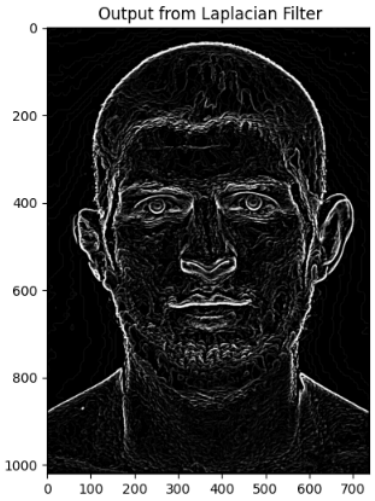
14. Finally, the resulting images at different stages of processing are displayed using matplotlib.pyplot library.

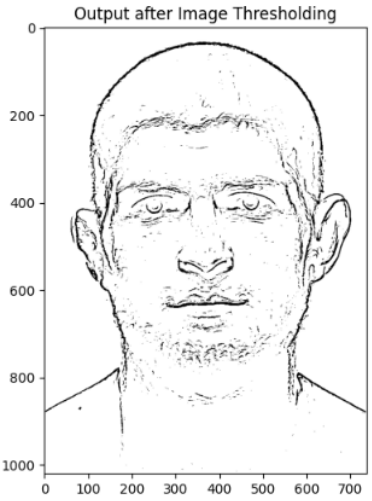
Representative Results:







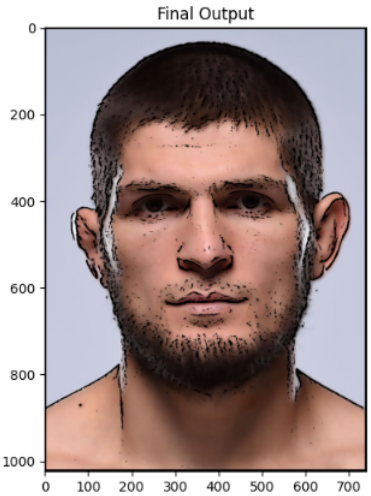

Output from Laplacian Filter
0
200
400
600
800
1000
0 100 200 300 400 500 600 700


Output after Image Thresholding
0
200
400
600
800
1000
0 100 200 300 400 500 600 700


Original RGB image
0
200
400
600
800
1000
0 100 200 300 400 500 600 700


Output from Bilateral Filter
0
200
400
600
800
1000
0 100 200 300 400 500 600 700


Output from Bilateral Filter
0
200
400
600
800
1000
0 100 200 300 400 500 600 700


Final Output
0
200
400
600
800
1000
0 100 200 300 400 500 600 700

Part II: Road Lane Detection

Code Description:

The objective of the code in this part of the assignment is to process the input road image and detect the lanes in it using Hough Transform. The following is the explanation for each function used:

1. median_filter: the function operates by applying a median filter by convolution of a unity kernel of dynamic size chosen as a parameter. During the convolution process, at each pixel sizexsize pixels are gathered, flattened then sorted to get their median which would be assigned to the target pixel.

2. Sobel_filter: the function generates gaussain kernel according to given size, and generates differential 1xsize or sizex1 kernels for both axis. The next step is to make them into separate filters, convolution of the separable gaussian with the differential kernels sequentially according to the separable filters operation.

3. non_max_suppression: the function takes horizontal and vertical gradients and an edge map, it generates gradients direction matrix then according to non max suppression operation pixels are taken within a 3x3 window and compare between the edges direction in the edge map with the resultant directions in the gradient direction neglecting pixels with non max values not on the given direxions.

4. Thresh: the function takes resultant suppressed version of the edge map classifying pixels into weak,strong or non-relevent pixels and neglecting the non-relevent. The remaining weak pixels are observed through a 3x3 window if and only if a weak pixel exist and there is a strong pixel around it, it turns into a strong pixel otherwise the weak pixel is neglected.

5. Hough_lines: takes edge map and generating a hough space by calculating max rho the hough space is made into theta and rho axis. Every pixel in the edge map is checked and mapped into the parameter space incrementing its votes at each sinsouidal wave generated by a pixel from the edge map then return the space.

6. Extract_lines: Takes the hough space and number of needed votes for making a line, the funcrion goes through the hough space and checks the votes of each rho and theta if they meets the given votes or higher they are added into a list. Returning a list of rho and theta with needed votes.

7. Code operation: First thing is to load an image and by applying every function in order and feeding each with the right parameter given at each one the result should be a list of rhos and thetas.

8. The next thing to do is to calculate the points to draw the lines. The calculation is done using the normal form formulas resulting in long lines. To minimise the lines length to the region of interset, points are taken and by calculating their slope and intersection point and as region of interest gives ymin and ymax, xmin and xmax can be calculated and resulting in two new points.

Representative Results: