```
import cv2
import numpy as np
import numpy.linalg as la

def MyConvolve(img, ff):
     #chooses the filter function according to the user's input
     result = np.zeros(img.shape)
     if ff.lower() == 'prewitt':
       result = prewitt_convolution(img)
     elif ff.lower() == 'sobel':
       result = sobel_convolution(img)
     else:
       result = False

     return result


def prewitt_convolution(img):
     # here we use the prewitt filter i.e.:
     # -1   -1   -1
     #  0    0    0      :  to detect horizontal edges
     #  1    1    1
     #
     #  -1    0    1
     #  -1    0    1     :  to detect vertical edges
     #  -1    0    1
     #
     # P.S. I didn't know we didn't have to check for boundaries before
     # the consultation and I had already implemented it, sorry about that
     # Skip to the else section in the for loop
     # to see how convolution is performed on the pixel in the middle
     # MARKED "OVER HERE"

     horizontal_edge_strength = np.zeros(img.shape)
     vertical_edge_strength = np.zeros(img.shape)
     edge_detected = np.zeros(img.shape)


     rows_for_img = len(img[:,0])
     columns_for_img = len(img[0,:])

     for k in range(rows_for_img):
       for h in range(columns_for_img):

              string = check_bounds(k, h, rows_for_img, columns_for_img)

              if string == "left_and_top_most":
                     #horizontal
                     g_y = (0.0*img[k][h+1]) + (1.0*img[k+1][h]) +
(1.0*img[k+1][h+1])
                     #vertical
                     g_x = (1.0*img[k][h+1]) + (0.0*img[k+1][h]) +
(1.0*img[k+1][h+1])

              elif string == "right_and_top_most":
                     #horizontal
                     g_y = (0.0*img[k][h-1]) + (1.0*img[k+1][h-1]) +
(1.0*img[k+1][h])
                     #vertical
                     g_x = (-1.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) +
(0.0*img[k+1][h])
```

```
            elif string == "left_and_bottom_most":
                  #horizontal
                  g_y = (-1.0*img[k-1][h]) + (-1.0*img[k-1][h+1]) +
(0.0*img[k][h+1])
                  #vertical
                  g_x = (0.0*img[k-1][h]) + (1.0*img[k-1][h+1]) +
(1.0*img[k][h+1])

            elif string == "right_and_bottom_most":
                  #horizontal
                  g_y = (-1.0*img[k-1][h-1]) + (0.0*img[k][h-1]) + (-
1.0*img[k-1][h])
                  #vertical
                  g_x = (-1.0*img[k-1][h-1]) + (-1.0*img[k][h-1]) +
(0.0*img[k-1][h])

            elif string == "top_most":
                  #horizontal
                  g_y = (0.0*img[k][h-1]) + (1.0*img[k+1][h-1]) +
(1.0*img[k+1][h]) + (0.0*img[k][h+1]) + (1.0*img[k+1][h+1])
                  #vertical
                  g_x = (-1.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) +
(0.0*img[k+1][h]) + (1.0*img[k][h+1]) + (1.0*img[k+1][h+1])

            elif string == "bottom_most":
                  #horizontal
                  g_y = (-1.0*img[k-1][h-1]) + (0.0*img[k][h-1]) + (-
1.0*img[k-1][h]) + (-1.0*img[k-1][h+1]) + (0.0*img[k][h+1])
                  #vertical
                  g_x = (-1.0*img[k-1][h-1]) + (-1.0*img[k][h-1]) +
(0.0*img[k-1][h]) + (1.0*img[k-1][h+1]) + (1.0*img[k][h+1])
            elif string == "left_most":
                  #horizontal
                  g_y = (-1.0*img[k-1][h]) + (-1.0*img[k-1][h+1]) +
(0.0*img[k][h+1]) + (1.0*img[k+1][h]) + (1.0*img[k+1][h+1])
                  #vertical
                  g_x = (0.0*img[k-1][h]) + (1.0*img[k-1][h+1]) +
(1.0*img[k][h+1]) + (0.0*img[k+1][h]) + (1.0*img[k+1][h+1])

            elif string == "right_most":
                  #horizontal
                  g_y = (-1.0*img[k-1][h-1]) + (-1.0*img[k-1][h]) +
(0.0*img[k][h-1]) + (1.0*img[k+1][h-1]) + (1.0*img[k+1][h])
                  #vertical
                  g_x = (-1.0*img[k-1][h-1]) + (0.0*img[k-1][h]) + (-
1.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) + (0.0*img[k+1][h])

            # OVER HERE
            else:
                  #horizontal
                  g_y = (-1.0*img[k-1][h-1]) + (-1.0*img[k-1][h]) + (-
1.0*img[k-1][h+1]) + (0.0*img[k][h-1]) + (0.0*img[k][h+1]) + (1.0*img[k+1][h-
1]) + (1.0*img[k+1][h]) + (1.0*img[k+1][h+1])
                  #vertical
                  g_x = (-1.0*img[k-1][h-1]) + (0.0*img[k-1][h]) +
(1.0*img[k-1][h+1]) + (-1.0*img[k][h-1]) + (1.0*img[k][h+1]) + (-
1.0*img[k+1][h-1]) + (0.0*img[k+1][h]) + (1.0*img[k+1][h+1])

            addition_of_squares = np.square(g_y) + np.square(g_x)
            square_root = np.sqrt(addition_of_squares)
```

```
                edge_detected[k][h] = square_root
                horizontal_edge_strength[k][h] = g_y
                vertical_edge_strength[k][h] = g_x

        return edge_detected

def sobel_convolution(img):
        # here we use the sobel filter i.e.:
        #  1    2    1
        #  0    0    0      :  to detect horizontal edges
        #  -1   -2   -1
        #
        #  -1    0    1
        #  -2    0    2     :  to detect vertical edges
        #  -1    0    1
        #
        # P.S. I didn't know we didn't have to check for boundaries before
        # the consultation and I had already implemented it, sorry about that
        # Skip to the else section in the for loop
        # to see how convolution is performed on the pixel in the middle
        # MARKED "OVER HERE"
        # also the direction of my sobel filter and prewitt filter is reversed

        horizontal_edge_strength = np.zeros(img.shape)
        vertical_edge_strength = np.zeros(img.shape)
        edge_detected = np.zeros(img.shape)
        rows_for_img = len(img[:,0])
        columns_for_img = len(img[0,:])

        for k in range(rows_for_img):
          for h in range(columns_for_img):

                string = check_bounds(k, h, rows_for_img, columns_for_img)

                if string == "left_and_top_most":
                        #horizontal
                        g_y = (0.0*img[k][h+1]) + (-2.0*img[k+1][h]) + (-
1.0*img[k+1][h+1])
                        #vertical
                        g_x = (2.0*img[k][h+1]) + (0.0*img[k+1][h]) +
(1.0*img[k+1][h+1])

                elif string == "right_and_top_most":
                        #horizontal
                        g_y = (0.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) + (-
2.0*img[k+1][h])
                        #vertical
                        g_x = (-2.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) +
(0.0*img[k+1][h])

                elif string == "left_and_bottom_most":
                        #horizontal
                        g_y = (2.0*img[k-1][h]) + (1.0*img[k-1][h+1]) +
(0.0*img[k][h+1])
                        #vertical
                        g_x = (0.0*img[k-1][h]) + (1.0*img[k-1][h+1]) +
(2.0*img[k][h+1])

                elif string == "right_and_bottom_most":
                        #horizontal
```

```
                    g_y = (1.0*img[k-1][h-1]) + (0.0*img[k][h-1]) + (2.0*img[k-
1][h])
                    #vertical
                    g_x = (-1.0*img[k-1][h-1]) + (-2.0*img[k][h-1]) +
(0.0*img[k-1][h])

             elif string == "top_most":
                    #horizontal
                    g_y = (0.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) + (-
2.0*img[k+1][h]) + (0.0*img[k][h+1]) + (-1.0*img[k+1][h+1])
                    #vertical
                    g_x = (-2.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) +
(0.0*img[k+1][h]) + (2.0*img[k][h+1]) + (1.0*img[k+1][h+1])

             elif string == "bottom_most":
                    #horizontal
                    g_y = (1.0*img[k-1][h-1]) + (0.0*img[k][h-1]) + (2.0*img[k-
1][h]) + (1.0*img[k-1][h+1]) + (0.0*img[k][h+1])
                    #vertical
                    g_x = (-1.0*img[k-1][h-1]) + (-2.0*img[k][h-1]) +
(0.0*img[k-1][h]) + (1.0*img[k-1][h+1]) + (2.0*img[k][h+1])
             elif string == "left_most":
                    #horizontal
                    g_y = (2.0*img[k-1][h]) + (1.0*img[k-1][h+1]) +
(0.0*img[k][h+1]) + (-2.0*img[k+1][h]) + (-1.0*img[k+1][h+1])
                    #vertical
                    g_x = (0.0*img[k-1][h]) + (1.0*img[k-1][h+1]) +
(2.0*img[k][h+1]) + (0.0*img[k+1][h]) + (1.0*img[k+1][h+1])

             elif string == "right_most":
                    #horizontal
                    g_y = (1.0*img[k-1][h-1]) + (2.0*img[k-1][h]) +
(0.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) + (-2.0*img[k+1][h])
                    #vertical
                    g_x = (-1.0*img[k-1][h-1]) + (0.0*img[k-1][h]) + (-
2.0*img[k][h-1]) + (-1.0*img[k+1][h-1]) + (0.0*img[k+1][h])

             # OVER HERE
             else:
                    #horizontal
                    g_y = (1.0*img[k-1][h-1]) + (2.0*img[k-1][h]) + (1.0*img[k-
1][h+1]) + (0.0*img[k][h-1]) + (0.0*img[k][h+1]) + (-1.0*img[k+1][h-1]) + (-
2.0*img[k+1][h]) + (-1.0*img[k+1][h+1])
                    #vertical
                    g_x = (-1.0*img[k-1][h-1]) + (0.0*img[k-1][h]) +
(1.0*img[k-1][h+1]) + (-2.0*img[k][h-1]) + (2.0*img[k][h+1]) + (-
1.0*img[k+1][h-1]) + (0.0*img[k+1][h]) + (1.0*img[k+1][h+1])

             addition_of_squares = np.square(g_y) + np.square(g_x)
             square_root = np.sqrt(addition_of_squares)

             edge_detected[k][h] = square_root
             horizontal_edge_strength[k][h] = g_y
             vertical_edge_strength[k][h] = g_x

     return edge_detected

def edge_thinning(edge_detected):
     # brief description: first we thin the vertical edges, and so we traverse
to the right
     # for every row, and find the first instance of a pixel with a value
```

```
higher than the threshold_value.
     # I have set a threshold_value of 138.0 as pixels stored in my edge
detection image
     # do not have an absolute value of 0, thus the threshold_value is meant to
find darker pixels.
     # once the first instance is found, we store it as the max value. Then, we
continue tranversing to
     # the right and performing the max function between the current pixel
value and max value to find the
     # pixel with the max value. At the same time, we store the index of that
pixel. If the current pixel
     # is smaller or equal to the max value, we set it to zero. We carry on
until we find a pixel that has a
     # value lower than the threshold_value. Then we go back to the pixel of
the index with the max value, and we
     # traverse to the left until we reach the start index. While traversing
left we turn the pixels black.
     # And that's how vertical edges are thinned.
     # implementation for the thinning of horizontal edges is somewhat similar.

     no_of_rows = edge_detected.shape[0]
     no_of_columns = edge_detected.shape[1]
     threshold_value  = 138.0

     for i in range(1, no_of_rows):
       for j in range(1, no_of_columns-1):

              #if it's black we skip the pixel and change it to black
              if edge_detected[i][j][0] <= threshold_value:
                     edge_detected[i][j] = [0.0, 0.0, 0.0]
                     continue

              max_value = 0.0
              max_row_index = 0
              max_col_index = 0
              #if pixel is the left edge pixel of an edge, we traverse right to
find the max
              if edge_detected[i][j-1][0] <= threshold_value and
edge_detected[i][j][0] > threshold_value:
                     current_pixel_row_index = i
                     current_pixel_col_index = j
                     start_pixel_row_index = i
                     start_pixel_col_index = j
                     # if current pixel is not a black pixel, we compare to the
max
                     #if it is black , we break out of the while loop

     while(edge_detected[current_pixel_row_index][current_pixel_col_index][0] >
threshold_value):
                            current_pixel_val =
edge_detected[current_pixel_row_index][current_pixel_col_index][0]

                            if current_pixel_val <= max_value:

     edge_detected[current_pixel_row_index][current_pixel_col_index] = [0.0,
0.0, 0.0]
                            else:
                                   max_value = max(max_value, current_pixel_val)
                                   max_row_index = current_pixel_row_index
                                   max_col_index = current_pixel_col_index
```

```
                        if current_pixel_col_index == (no_of_columns - 1):
                                break
                        current_pixel_col_index = current_pixel_col_index + 1

                current_pixel_row_index = max_row_index
                current_pixel_col_index = max_col_index

                while(current_pixel_col_index != start_pixel_col_index):

    edge_detected[current_pixel_row_index][current_pixel_col_index-1] = [0.0,
0.0, 0.0]
                        current_pixel_col_index = current_pixel_col_index - 1
                        if current_pixel_col_index == 0:
                                break

    for k in range(1, no_of_columns):
      for h in range(1, no_of_rows-1):

            #if it's black we skip the pixel and change it to black
            if edge_detected[h][k][0] <= threshold_value:
                    edge_detected[h][k] = [0.0, 0.0, 0.0]
                    continue

            max_value = 0.0
            max_row_index = 0
            max_col_index = 0
            #if pixel is the left edge pixel of an edge, we traverse right to
find the max
            if edge_detected[h][k-1][0] <= threshold_value and
edge_detected[h][k][0] > threshold_value:
                current_pixel_row_index = h
                current_pixel_col_index = k
                start_pixel_row_index = h
                start_pixel_col_index = k
                # if current pixel is not a black pixel, we compare to the
max
                #if it is black , we break out of the while loop

    while(edge_detected[current_pixel_row_index][current_pixel_col_index][0] >
threshold_value):
                        current_pixel_val =
edge_detected[current_pixel_row_index][current_pixel_col_index][0]

                        if current_pixel_val <= max_value:

    edge_detected[current_pixel_row_index][current_pixel_col_index] = [0.0,
0.0, 0.0]
                        else:
                                max_value = max(max_value, current_pixel_val)
                                max_row_index = current_pixel_row_index
                                max_col_index = current_pixel_col_index

                        if current_pixel_row_index == (no_of_rows - 1):
                                break
                        current_pixel_row_index = current_pixel_row_index + 1

                current_pixel_row_index = max_row_index
                current_pixel_col_index = max_col_index

                while(current_pixel_row_index != start_pixel_row_index):
```

```
     #while(edge_detected[current_pixel_row_index][current_pixel_col_index-
1][0] > 138.0):
                              #pixel_to_the_left_val =
edge_detected[current_pixel_row_index][current_pixel_col_index-1][0]
                              #if pixel_to_the_left_val <= max_value:

     edge_detected[current_pixel_row_index][current_pixel_col_index-1] = [0.0,
0.0, 0.0]
                              current_pixel_row_index = current_pixel_row_index - 1
                              if current_pixel_row_index == 0:
                                     break

     return edge_detected


def check_bounds(index_from_top, index_from_left, no_of_rows, no_of_cols):
     string = ""

     #left and top most
     if (index_from_top == 0) and (index_from_left == 0):
       string = "left_and_top_most"
     elif (index_from_top == 0) and (index_from_left == (no_of_cols-1)):
       string = "right_and_top_most"
     elif (index_from_top == (no_of_rows-1)) and (index_from_left == 0):
       string = "left_and_bottom_most"
     elif (index_from_top == (no_of_rows-1)) and (index_from_left ==
(no_of_cols-1)):
       string = "right_and_bottom_most"
     elif (index_from_top == 0):
       string = "top_most"
     elif (index_from_top == (no_of_rows-1)):
       string = "bottom_most"
     elif (index_from_left == 0):
       string = "left_most"
     elif (index_from_left == (no_of_cols-1)):
       string = "right_most"
     else:
       string = "center"

     return string


def get_grayscale_image(img):
     rows_for_img = len(img[:,0])
     columns_for_img = len(img[0,:])
     grayscale_image = np.zeros(img.shape)

     for i in range(rows_for_img):
       for j in range(columns_for_img):
              current_pixel = img[i][j]
              red = current_pixel[0]
              green = current_pixel[1]
              blue = current_pixel[2]

              r_val = red / 255.0
              g_val = green / 255.0
              b_val = blue / 255.0

              c_max = max(r_val, g_val, b_val)
              v_value = c_max*255
```

```
                grayscale_image[i][j] = v_value

        return grayscale_image


file_name = raw_input('Enter file name: ')
filter_function = raw_input('Enter filter function (Prewitt or Sobel): ')
thinning_function = raw_input('Thinning? (Y/N): ')

image = cv2.imread(file_name)

#we first obtain the grayscale image, i.e. V value like lab 2
grayscale_image = get_grayscale_image(image)

#then we pass through the convolution function
convolution_result_array = MyConvolve(grayscale_image, filter_function)
if filter_function.lower() == 'prewitt':
        if file_name == 'example.jpg':
          cv2.imwrite('example_prewitt.jpg', convolution_result_array)
        elif file_name == 'test1.jpg':
          cv2.imwrite('test1_prewitt.jpg', convolution_result_array)
        elif file_name == 'test2.jpg':
          cv2.imwrite('test2_prewitt.jpg', convolution_result_array)
        elif file_name == 'test3.jpg':
          cv2.imwrite('test3_prewitt.jpg', convolution_result_array)
elif filter_function.lower() == 'sobel':
        if file_name == 'example.jpg':
          cv2.imwrite('example_sobel.jpg', convolution_result_array)
        elif file_name == 'test1.jpg':
          cv2.imwrite('test1_sobel.jpg', convolution_result_array)
        elif file_name == 'test2.jpg':
          cv2.imwrite('test2_sobel.jpg', convolution_result_array)
        elif file_name == 'test3.jpg':
          cv2.imwrite('test3_sobel.jpg', convolution_result_array)

if thinning_function.lower() == 'y':
        if filter_function.lower() == 'prewitt':
          if file_name == 'example.jpg':
                edge_detected_image = cv2.imread('example_prewitt.jpg')
                thinned_edge = edge_thinning(edge_detected_image)
                cv2.imwrite('example_prewitt_thinning.jpg', thinned_edge)
          elif file_name == 'test1.jpg':
                edge_detected_image = cv2.imread('test1_prewitt.jpg')
                thinned_edge = edge_thinning(edge_detected_image)
                cv2.imwrite('test1_prewitt_thinning.jpg', thinned_edge)
          elif file_name == 'test2.jpg':
                edge_detected_image = cv2.imread('test2_prewitt.jpg')
                thinned_edge = edge_thinning(edge_detected_image)
                cv2.imwrite('test2_prewitt_thinning.jpg', thinned_edge)
          elif file_name == 'test3.jpg':
                edge_detected_image = cv2.imread('test3_prewitt.jpg')
                thinned_edge = edge_thinning(edge_detected_image)
                cv2.imwrite('test3_prewitt_thinning.jpg', thinned_edge)

        elif filter_function.lower() == 'sobel':
          if file_name == 'example.jpg':
                edge_detected_image = cv2.imread('example_sobel.jpg')
                thinned_edge = edge_thinning(edge_detected_image)
                cv2.imwrite('example_sobel_thinning.jpg', thinned_edge)
          elif file_name == 'test1.jpg':
                edge_detected_image = cv2.imread('test1_sobel.jpg')
```

```
        thinned_edge = edge_thinning(edge_detected_image)
        cv2.imwrite('test1_sobel_thinning.jpg', thinned_edge)
    elif file_name == 'test2.jpg':
        edge_detected_image = cv2.imread('test2_sobel.jpg')
        thinned_edge = edge_thinning(edge_detected_image)
        cv2.imwrite('test2_sobel_thinning.jpg', thinned_edge)
    elif file_name == 'test3.jpg':
        edge_detected_image = cv2.imread('test3_sobel.jpg')
        thinned_edge = edge_thinning(edge_detected_image)
        cv2.imwrite('test3_sobel_thinning.jpg', thinned_edge)
```