DSE 312 Assignment-1: Filters

Name: VK Susheel
Roll No. : 20297

Before Application:

Since the given image is 2666 × 2000, it is too large to easily view. Further, the effects of the filters are negligible. For this reason, the image is resized to 666 × 500 (dividing rows and columns by 4) before applying the filter.
Also, both the grayscale and original color images are used for different filters. The filters that shift the image, corresponding to a 1 at locations f and g according to the convention given in the question, do not show easily visible differences when applied just once to the code. Hence, these filters are applied repeatedly to show the resulting shift in the image. To reduce complexity, these filters are applied on the grayscale image since the added dimension corresponding to the color channel increases the time taken to implement these filters repeatedly. However, the implementation of the filters for shifting as described below will work for both color and grayscale images, although for brevity, only the results for the grayscale image will be shown. For the remaining filters, the color image is used, since the filter is applied only once. For locations d and c, due to the implementation of the filter using nested for loops as opposed to vectorization implementation that computes the resulting image in place, the approach mentioned above will not work. This is because once the filter is applied recursively, the values of the padded rows and columns spread throughout the image, leading to a completely dark image. For these filters, we use a cropped image centered around the eye to show the right and diagonally downwards shift. For the second question, since the output obtained from applying both filters can be greater than 255 or be less than 0, we convert the type of the image array to 32 bit floating point, to allow for greater precision. The output array is then clipped to between 0 and 255 and converted back to unsigned 8 bit integer type.

Code Description:

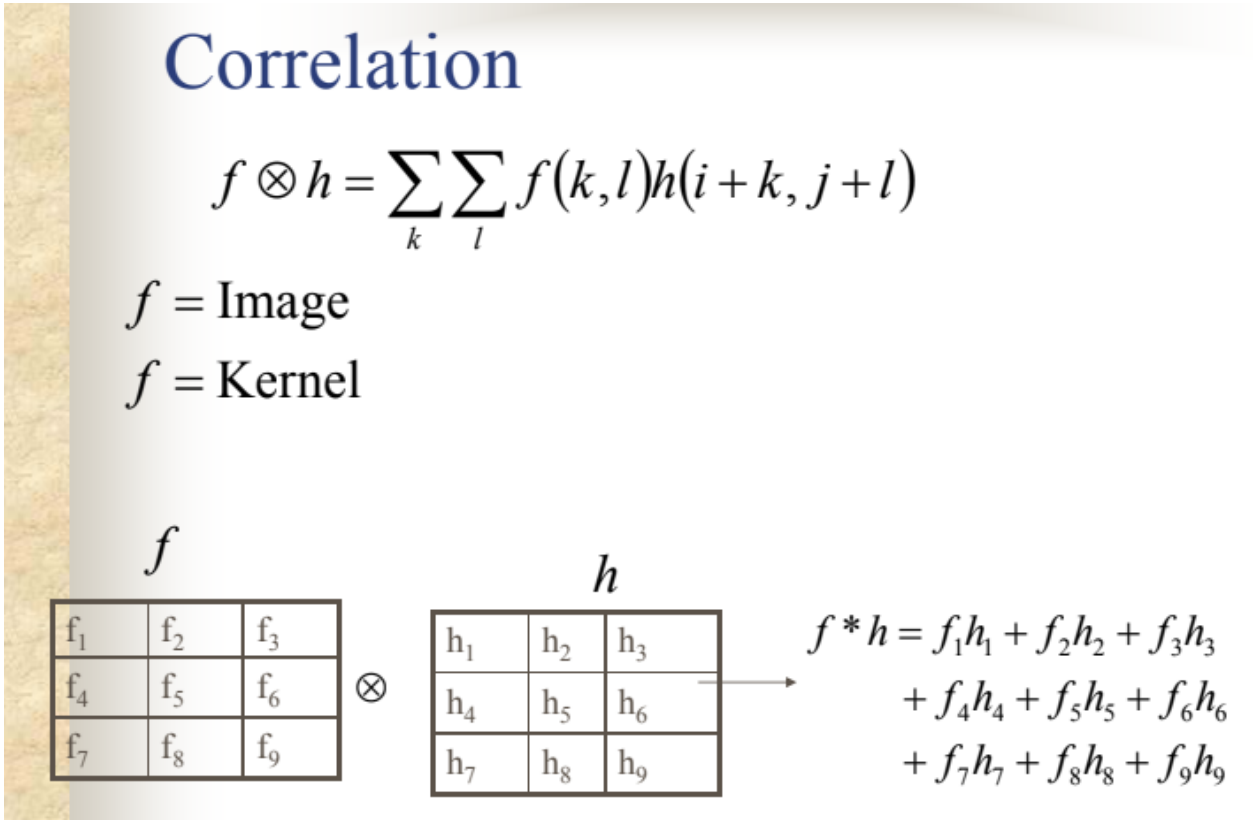
The filters are implemented using functions called apply_filter_color, apply_filter_color_float or apply_filter_gray depending on the type of filter, as mentioned above. Both functions take as input the source image, the kernel to be applied and the number of times it should be applied. The input image is padded with zeros according to the number of times the filter is applied, which is one for all filters except for the shifting filters. This allows us to obtain an output image with the same size as the input image. The filtering is then done by using a nested for loop, which multiplies each value from the 3 × 3 kernel with the corresponding pixel value in the image. The center of the kernel is placed on the pixel of interest in the image, and the kernel is

then shifted to cover the entire image. If the filter is applied multiple times, an additional loop is required. Note that when the filter is applied multiple times, the application becomes recursive, so the output of one iteration becomes the input to the next, till the outermost loop ends. As mentioned before, we convert the input to floating point for the filters in the second question. The final image is then converted to uint8 type or 8 bit integer and returned.
The implementation is standardized for all filters except for the Gaussian. For the other filters the 3 × 3 kernel is implemented using a numpy array, with the values filled at the same time as the definition. For the Gaussian filter, a nested loop is used to fill values according to the expression:

$$G(x, y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

, where x and y are the coordinates of the elements of the kernel (note: we start indexing from (-1, 1), with the last coordinate being (1, 1). σ is the standard deviation of the distribution, for this implementation, σ = 1 is taken. For Gaussian, Mean and Sobel filters the results are compared to that obtained using the inbuilt function cv.filter2D, with the appropriate kernel.

Note: The formula for correlation used to apply the filter is as follows:



(Taken from the slides of Lecture 7 - Filtering)

Q1.

In this question we apply 5 3 × 3 filters. The general filter is given below:

a	b	c
d	e	f
g	h	i

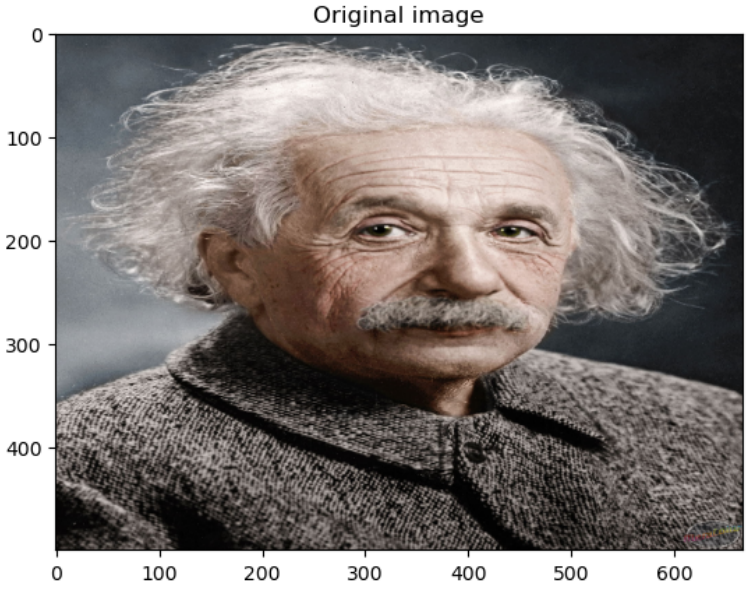
The five filters correspond to one at positions e, f, d, g and c, with zeros in the other 8 positions.

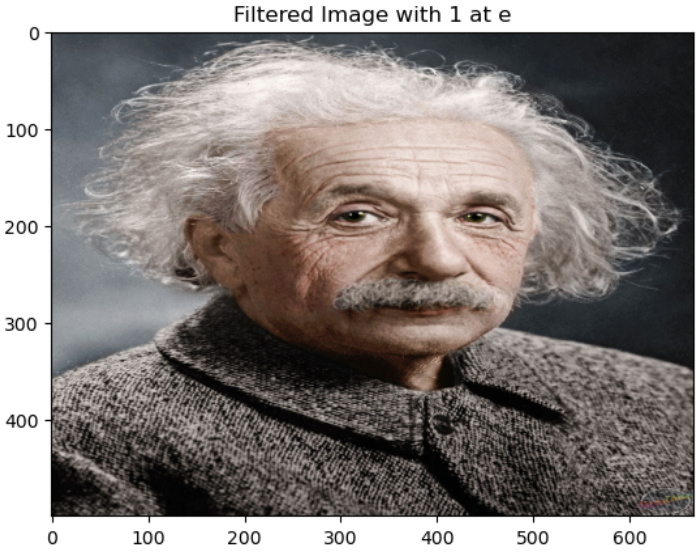
One at (e):

Kernel:

0	0	0
0	1	0
0	0	0

Since all elements of the kernel are 0 except for the center which is 1, the result when the filter is applied to a particular pixel is simply the pixel itself. This holds for all the pixels in the image, and so the resulting image is identical to the input image, as is shown below:



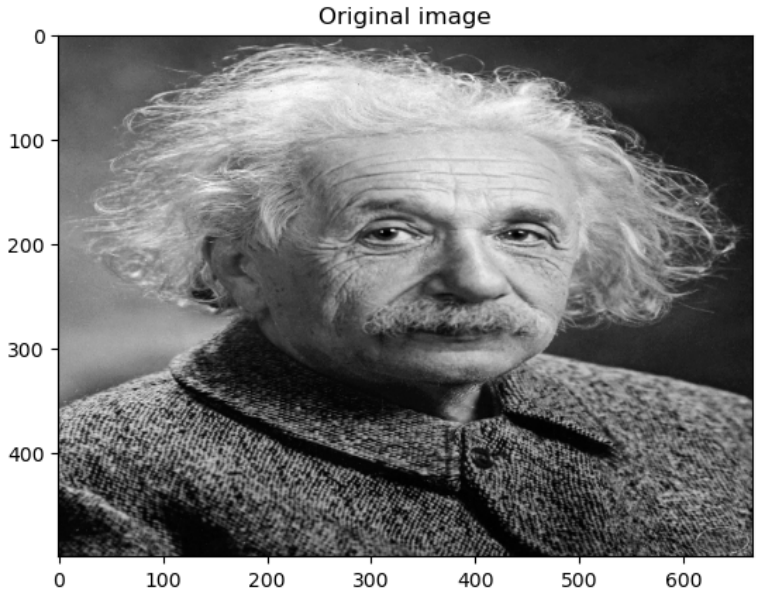


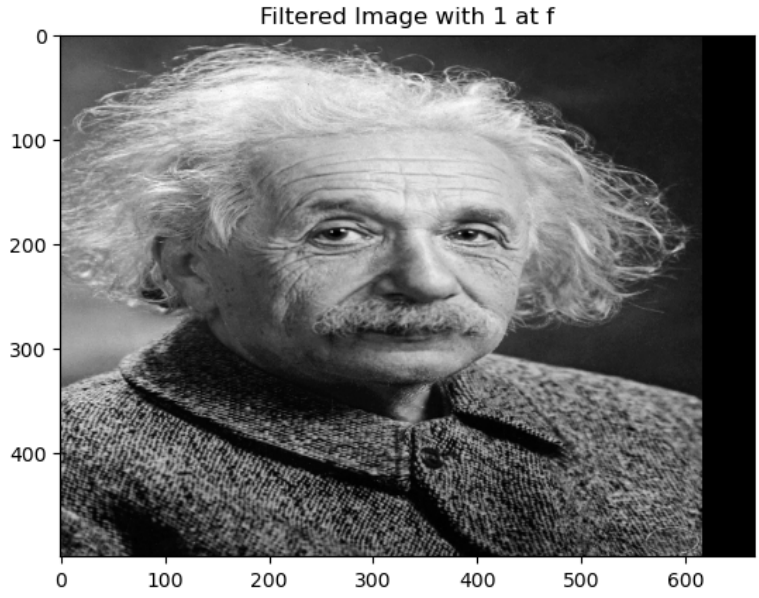
One at (f):

Kernel:

0	0	0
0	0	1
0	0	0

Since all elements of the kernel are 0 except for the pixel to the immediate right of the center which is 1, the result when the filter is applied to a particular pixel is the value of the pixel to the immediate right of the pixel of interest, i.e. the pixel in the same row and next column. This holds for all the pixels in the image, and so the resulting image is a right-shift of the original image. As mentioned earlier, we have padded the input image and applied the filter multiple (50) times. This allows us to clearly visualize the shift as the leftmost section of the image (for 50 columns) will be entirely black, as shown below:



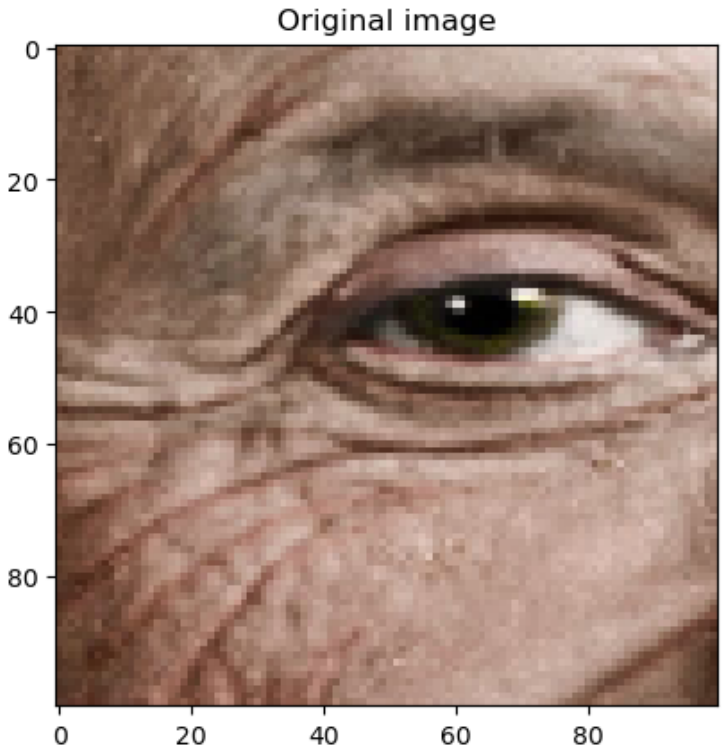


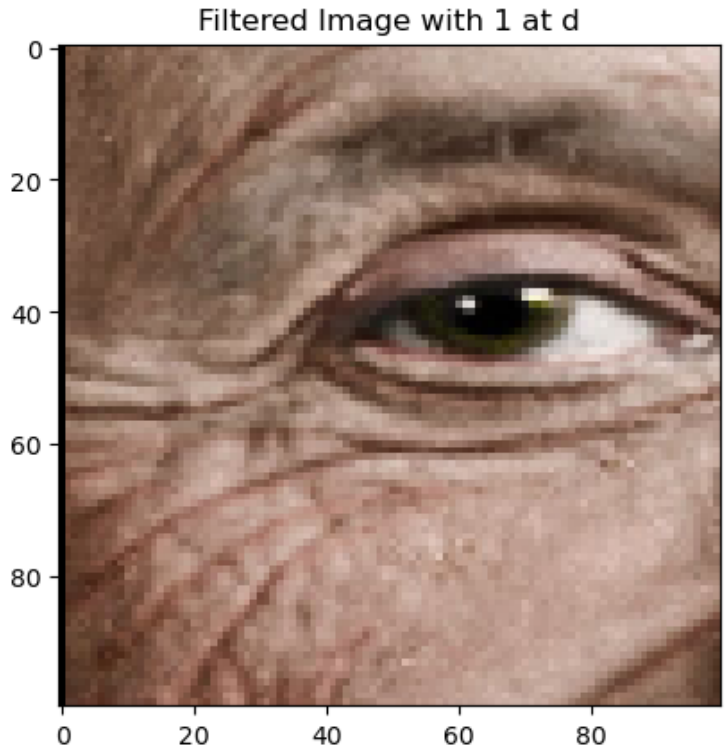
One at (d):

Kernel:

0	0	0
1	0	0
0	0	0

Since all elements of the kernel are 0 except for the pixel to the immediate left of the center which is 1, the result when the filter is applied to a particular pixel is the value of the pixel to the immediate left of the pixel of interest, i.e. the pixel in the same row and previous column. This holds for all the pixels in the image, and so the resulting image is a left-shift of the original image. As mentioned earlier, we use a cropped version of the original color image to visualize this difference:



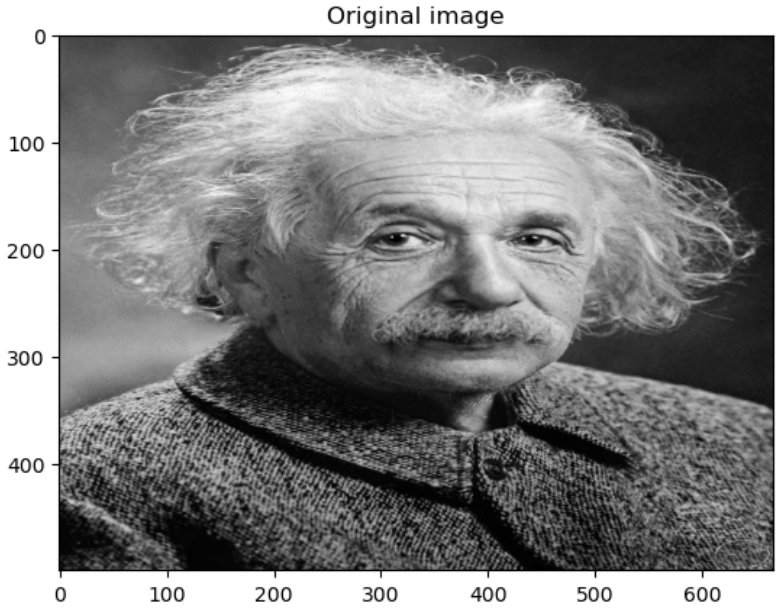


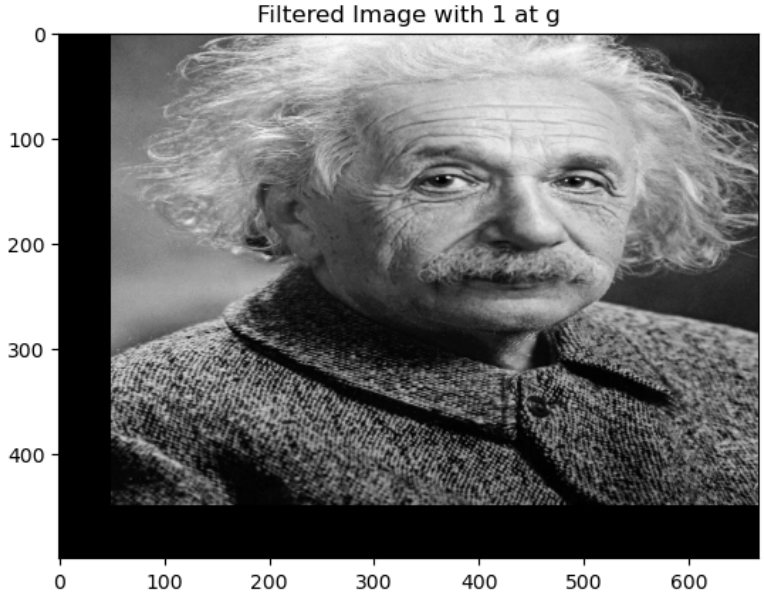
One at (g):

Kernel:

0	0	0
0	0	0
1	0	0

Since all elements of the kernel are 0 except for the pixel to the immediate left and below the center which is 1, the result when the filter is applied to a particular pixel is the value of the pixel to the immediate left and below the pixel of interest, i.e. the pixel in the previous row and previous column. This holds for all the pixels in the image, and so the resulting image is a diagonally upward shift of the original image. As mentioned earlier, we have padded the input image and applied the filter multiple (50) times. This allows us to clearly visualize the shift as the left and bottom borders of the image (for 50 rows and columns) will be entirely black, as shown below:



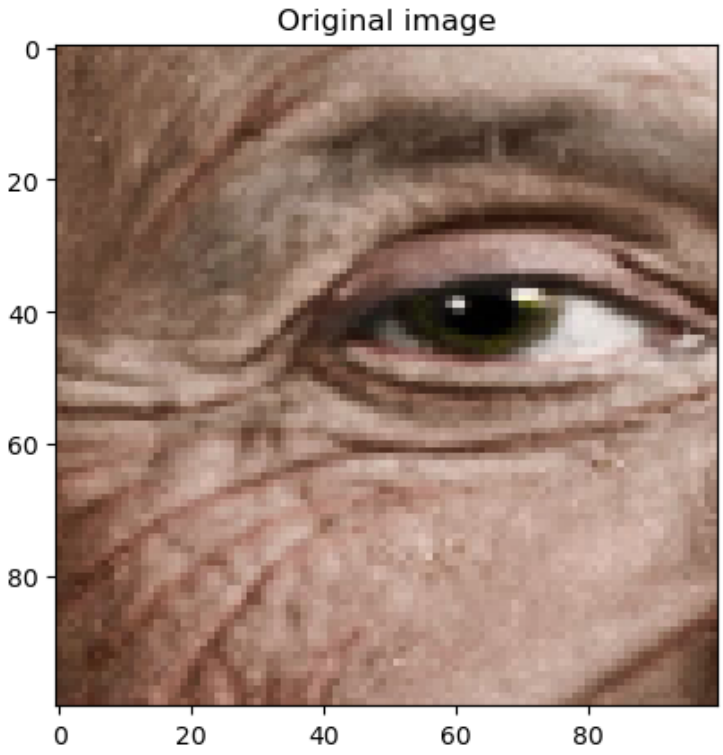


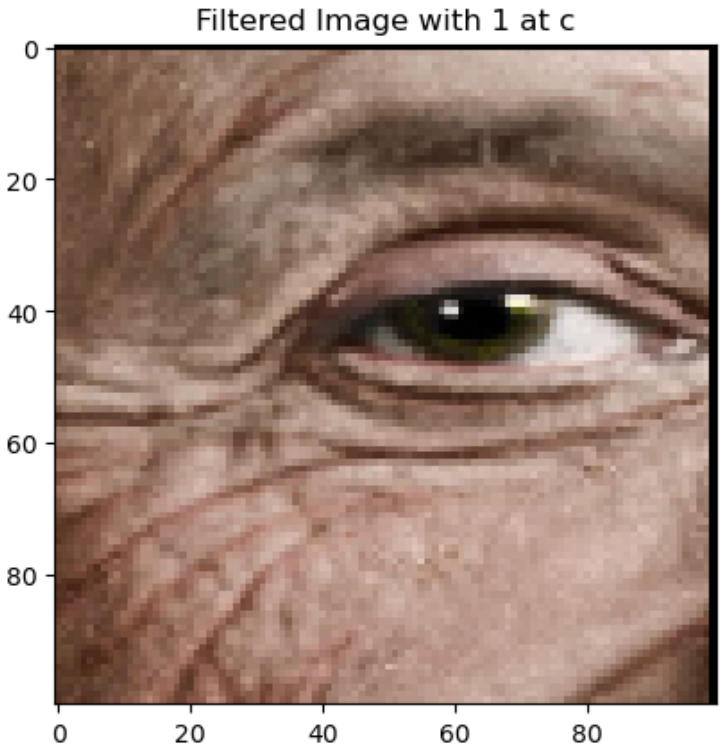
One at (c):

Kernel:

0	0	1
0	0	0
0	0	0

Since all elements of the kernel are 0 except for the pixel to the immediate right and above the center which is 1, the result when the filter is applied to a particular pixel is the value of the pixel to the immediate right and above the pixel of interest, i.e. the pixel in the next row and next column. This holds for all the pixels in the image, and so the resulting image is a diagonally downward shift of the original image. As mentioned earlier, we use a cropped version of the original color image to visualize this difference:





Q2.

The filters to be implemented are:

1	0	-1
2	0	-2
1	0	-1

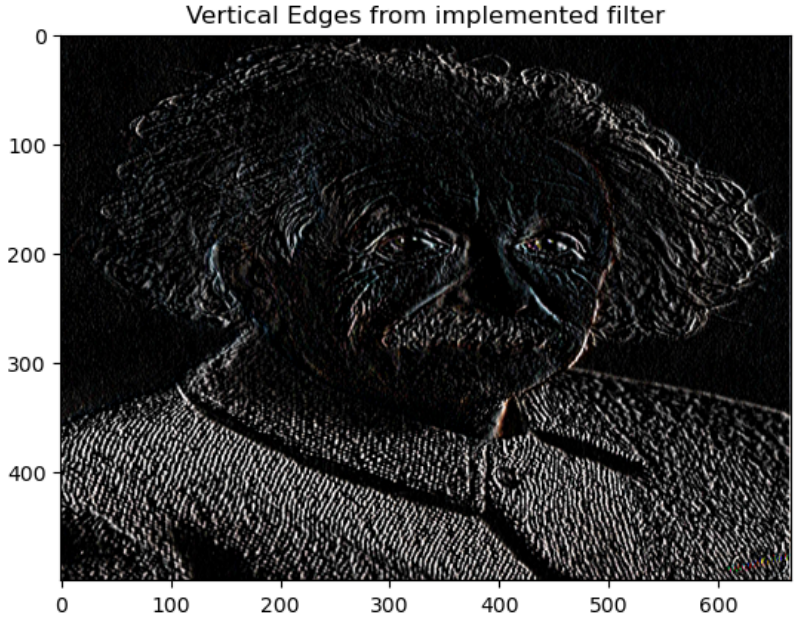
and

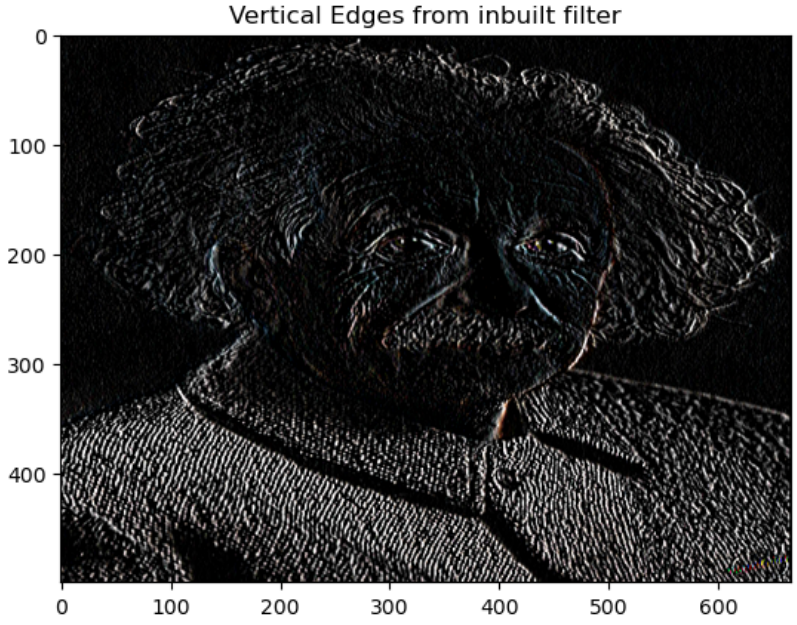
1	2	1
0	0	0
-1	-2	-1

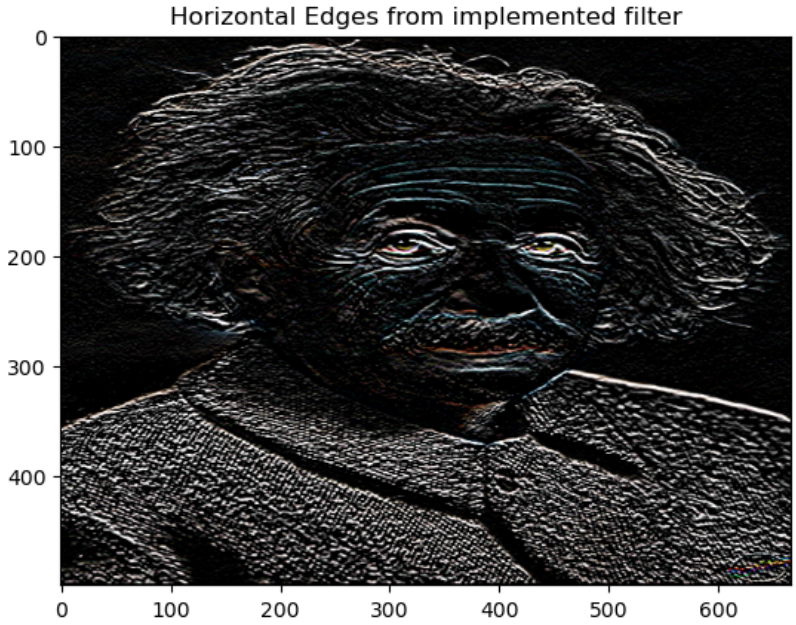
Clearly, both filters are the transpose of one another, and both compute the gradients of intensities. The first computes the gradient in the horizontal direction, while the second computes the gradient in the vertical direction. The type of gradient is a central difference, where the difference in intensities between the pixels above and below or to the left and right of the pixel of interest is considered. Here, the increasing direction of pixel coordinates are from the bottom left towards the top right. We also see that the central difference of the central pixel is more heavily weighted than the central difference of the pixels above and below or to the left or right of the center pixel.

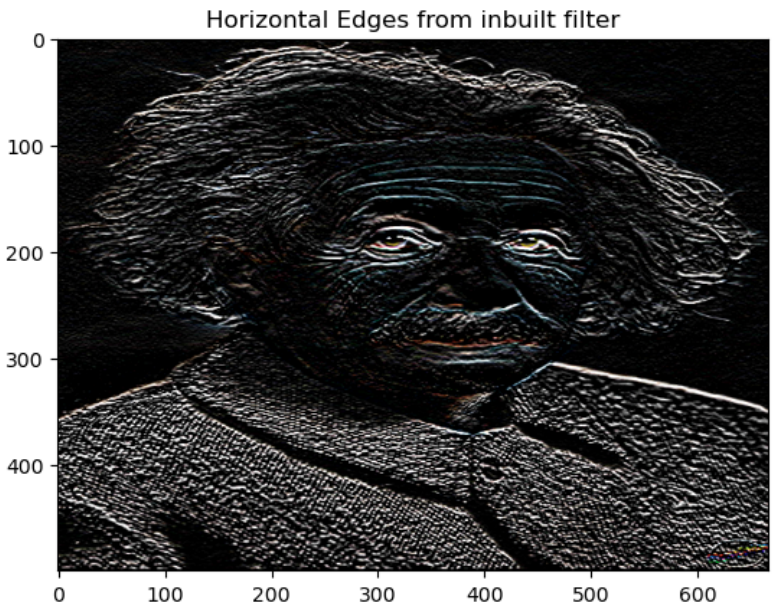
The intuition behind both filters are the same. Horizontal edges produce a vertical gradient in an image, since we would expect to see a large change in intensities as we move from below the edge to above it and vice versa. For the same reason, vertical edges produce a horizontal gradient. Hence, the first filter that computes the horizontal gradient detects the vertical edges and the second filter that computes the vertical gradient detects the horizontal edges. The results are shown below:


Original image
0
100
200
300
400
0
100
200
300
400
500
600


Vertical Edges from implemented filter
0
100
200
300
400
0
100
200
300
400
500
600


Vertical Edges from inbuilt filter
0
100
200
300
400
0
100
200
300
400
500
600


Horizontal Edges from implemented filter
0
100
200
300
400
0
100
200
300
400
500
600

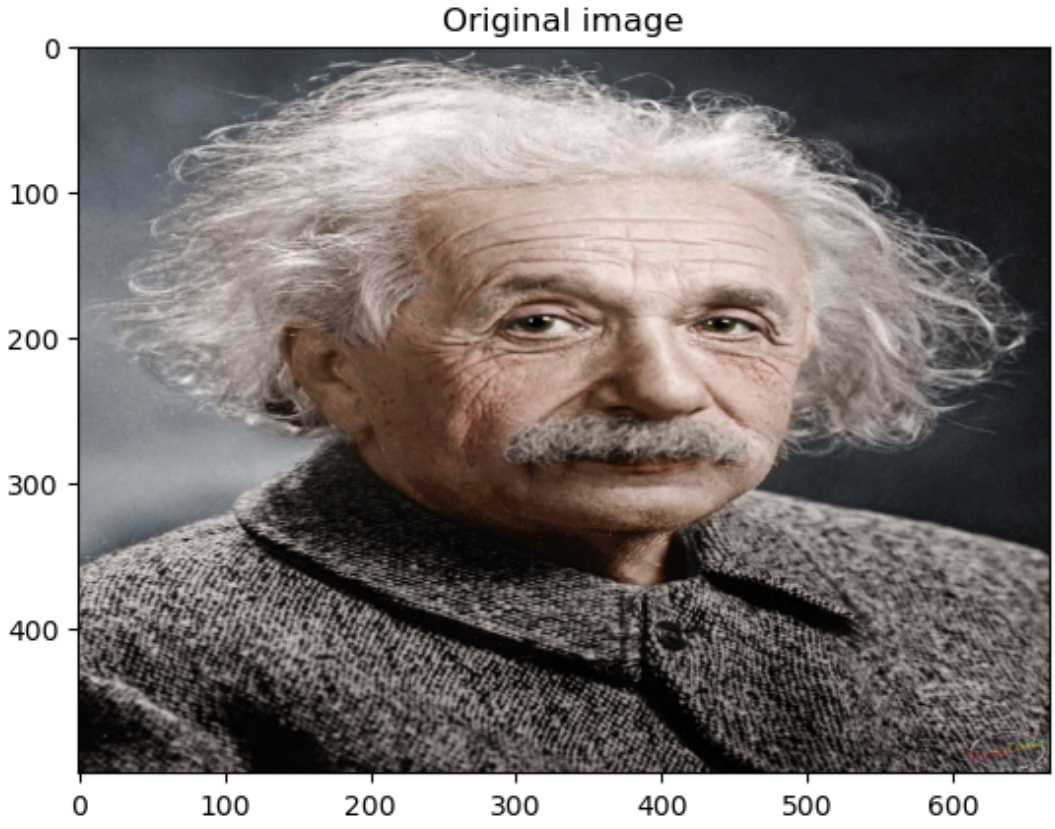

Horizontal Edges from inbuilt filter
0
100
200
300
400
0
100
200
300
400
500
600

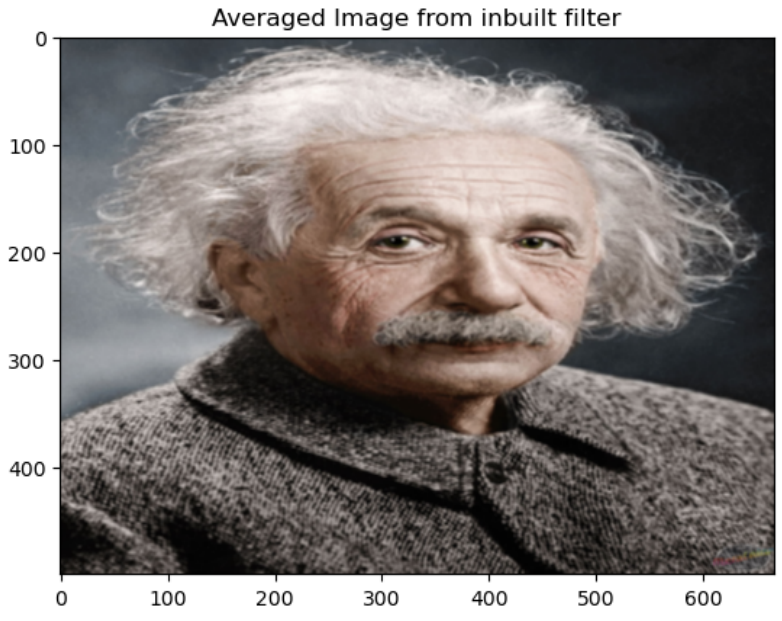
Q3.

The average filter implemented is:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The filter, known as the box filter, averages the intensities of the 9 pixels on which the filter is placed, and has a blurring or smoothing effect on the image since large discrepancies in intensities are reduced, leading to a reduction in sharp features. The scaling factor of (1 / 9) is also important to ensure that the R, G and B values for color images and intensity values for grayscale images do not exceed the maximum allowed values (255 for 8 bit integers). A smaller scaling factor would diminish the intensities, while a larger scaling factor could allow overflow to occur. This method of scaling, i.e. dividing the filter elements with the sum of the elements of the filter is also followed in the implementation of the Gaussian filter. The results of the averaging filter are as follows:





The second filter to be applied in this question is the Gaussian filter, which is the only non-linear filter in this assignment. The formula describing the filter was given above, and involves taking the negative exponential of the sum of squares of the filter coordinates to determine the value of the filter for each coordinate, i.e:

$$G(x,\ y)\ =\ e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Here, we consider σ = 1 as mentioned earlier. The resulting filter is as follows:

0.07511361	0.123814	0.07511361
0.1238412	0.20417996	0.1238412
0.07511361	0.123814	0.07511361

The filter was obtained by first applying the formula given above and then normalizing by dividing all the entries by the sum of all the values of the array.

There are two important properties that are immediately apparent. The first is the symmetry, with the first and third rows being identical, as well as the first and last elements of each row being equal with the middle element approximately double the other values in the row.
The other important feature is the fact that the values in the second row are approximately double the values of the other two rows.
The impact of these features can be determined by a comparison to the averaging filter applied earlier, which had equal values in all 9 coordinates. Similar to the box filter, the Gaussian filter also has a smoothing and blurring effect. However, due to the asymmetry described earlier, there are subtle differences in the output obtained from both filters. The Gaussian filter, like the box filter, computes a weighted average of the 9 pixels on which it is applied. Unlike the box filter which gives equal weight to all 9 pixels, the Gaussian filter gives maximum weightage to the central pixel, second highest to the pixels to the immediate left and right as well as above and below the center pixel. The pixels that are diagonal to the center pixel are given the minimum weightage. This means that the Gaussian filter provides gentler smoothing and preserves edges better. This property can be visualized by the results given below: