DSE312 Assignment-2

Name : VK Susheel
Roll No. : 20297

Before Application:

As in the first assignment, the size of the Einstein image used is reduced to 500 ✕ 666 to easily visualize the output. Further, only grayscale images are used for all three questions to speed up calculations since the implementations are non-vectorized without the use of any inbuilt functions. To reduce information loss, the 32 bit floating point representation of the images is used. Lastly, for the Gaussian and Laplacian pyramids, we resize the corresponding images to 512 ✕ 512, which avoids edge cases where the images have odd rows or columns.

Code Description:

For the first question, relevant padding is applied in all 3 cases in both X and Y directions to ensure the output is of the same size as the input image. When displaying the gradient magnitude and orientation, the jet colormap is used to better visualize the differences in magnitude and orientation across the image.
For the second question, inbuilt functions, namely cv2. GaussianBlur() and cv2.Laplacian() are applied to compute the LoG and DoG. For the DoG, we use $\sigma = 2.4$ filter for the more blurred image, and $\sigma = 1.6$ image for the less blurred image. $\sigma = 1.6$ is also used for the LoG. For all GaussianBlur() calls, we use a 5 ✕ 5 filter size, and the BORDER_REPLICATE border type, which simply copies the border values when padding the input image. Since the Gaussian filter is separable, the OpenCV implementation computes filters in X and Y based on the σ value, and applies them individually, instead of convolving the image with a 2-D filter, which reduces complexity. The Laplacian function simply convolves the input image with the following kernel:

Laplacian:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

To compute the Gaussian pyramid, functions are written to apply the Gaussian filter and downsample the result as we move up the pyramid. Finally for the Laplacian, we use the same filter and upsample by linear interpolation of the rows and columns of the original image to form the odd rows and columns of the upsampled image.

Q1:

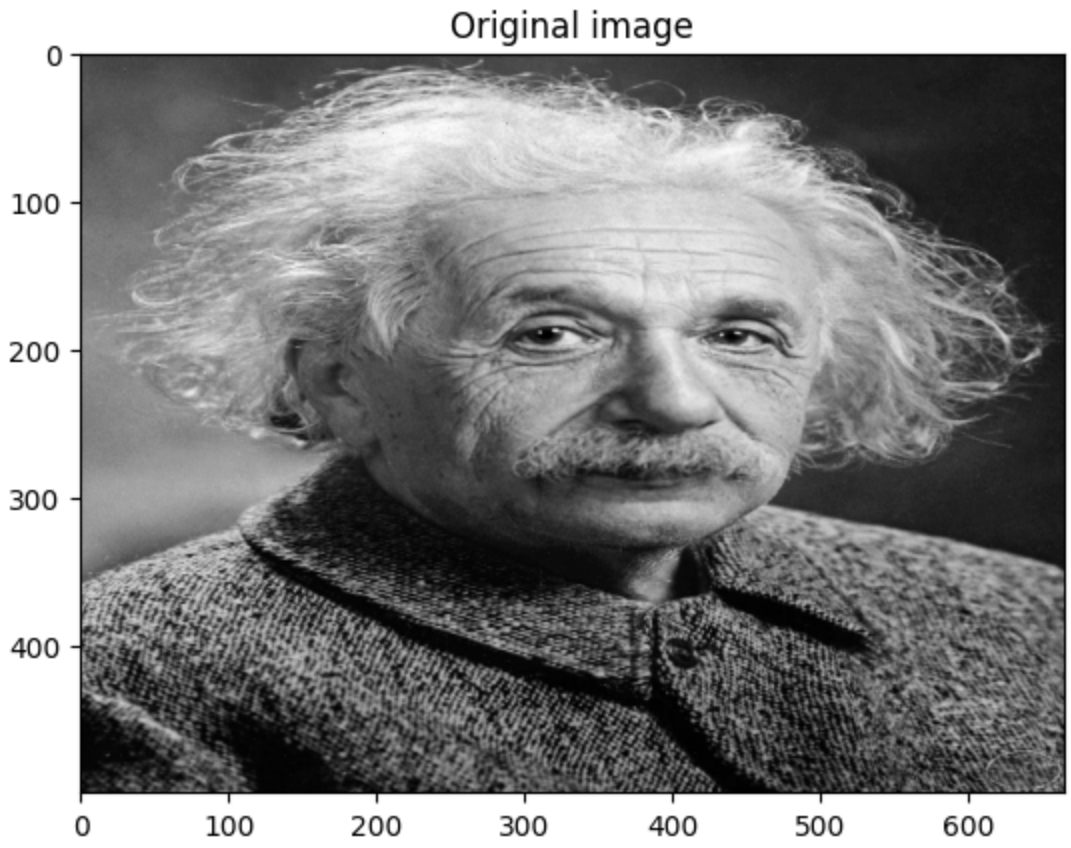
Forward difference : In X direction [1 -1], in Y direction $[1\ -1]^T$
Backward difference : In X direction [-1 1], in Y direction $[-1\ 1]^T$
Central difference : In X direction [-1 0 1], in Y direction $[-1\ 0\ 1]^T$

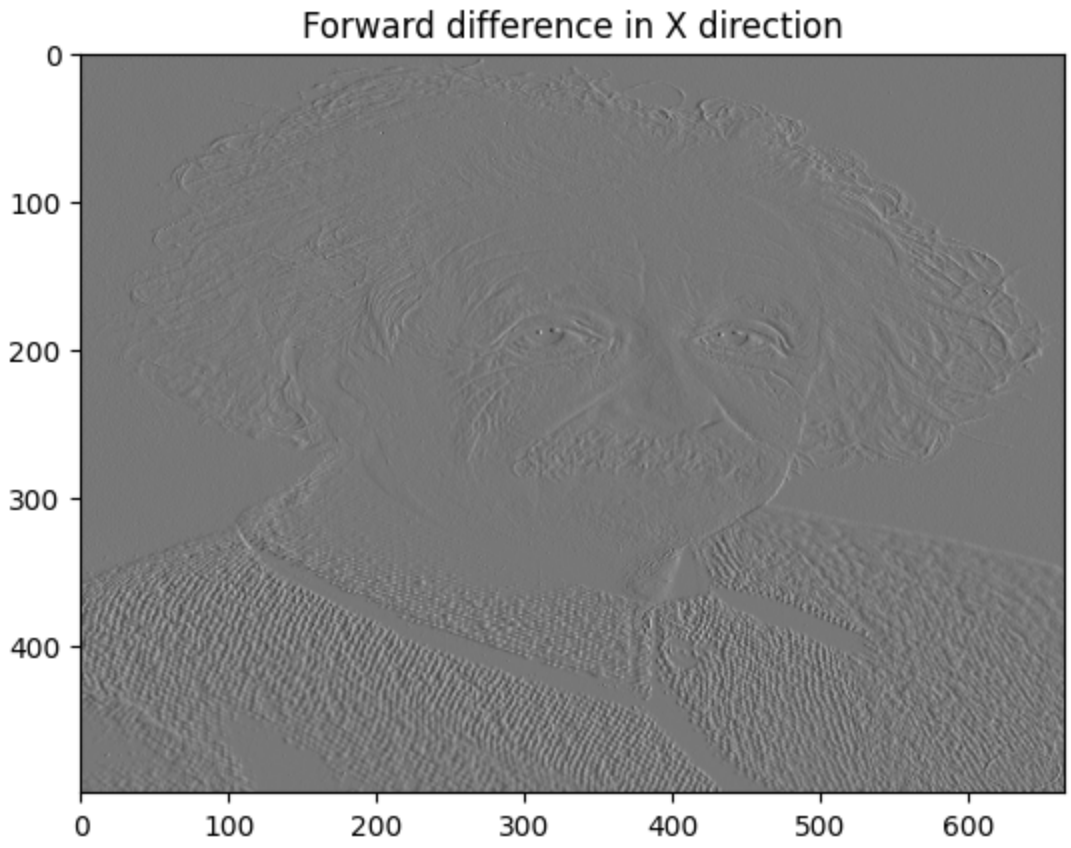
In both backward and forward differences, the pixel of interest is multiplied by 1 and then either the pixel to the left, right, above or below is subtracted from it, depending on the direction and axis of the filter. In other words, the 1 of the filter is placed on the pixel of interest. In the central difference, the filter value 0 is placed over the pixel of interest. Note that padding is required in all 6 cases to ensure the output image is of the same size as the input image, as shown in the code. Since we are applying the filters in X and Y direction independently, we can simply take the difference of the relevant columns for the X direction and difference of relevant rows for the Y direction.
To calculate the magnitude and orientation of the resulting gradient (difference), we use the following expressions:

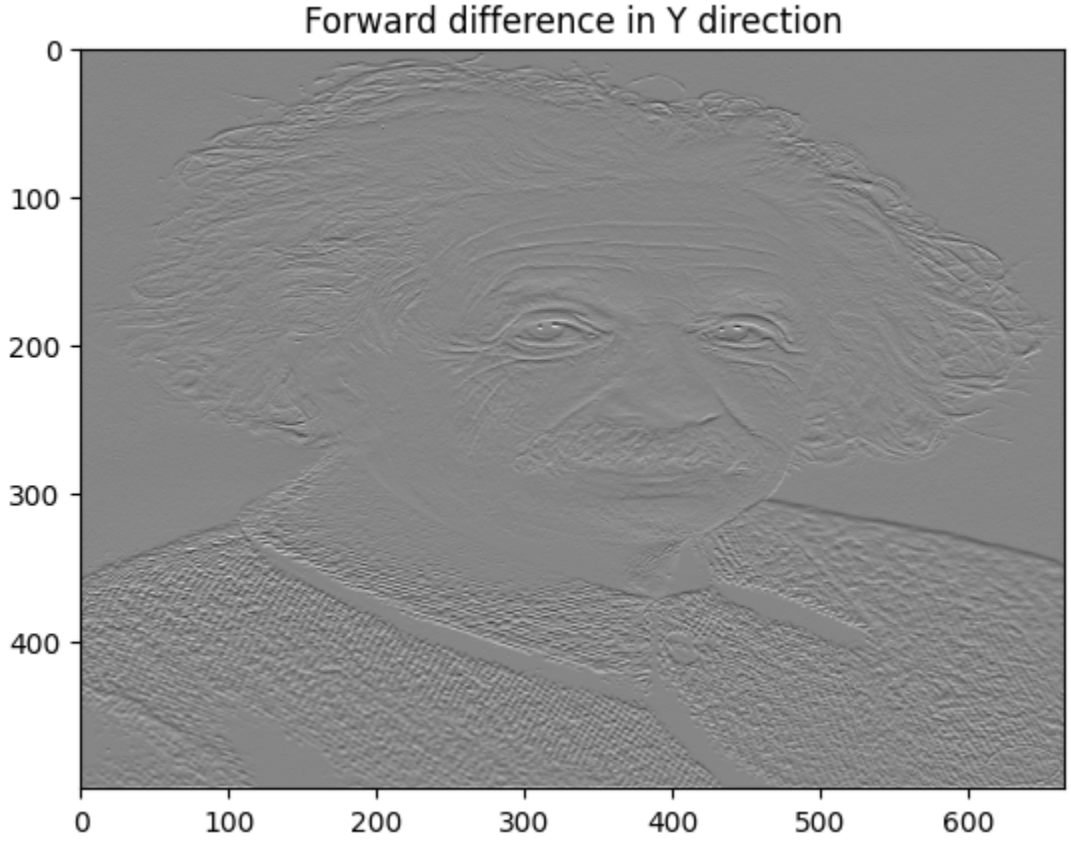
$$|\nabla f(x,\, y)| \;=\; \sqrt{f_x^2 + f_y^2}$$

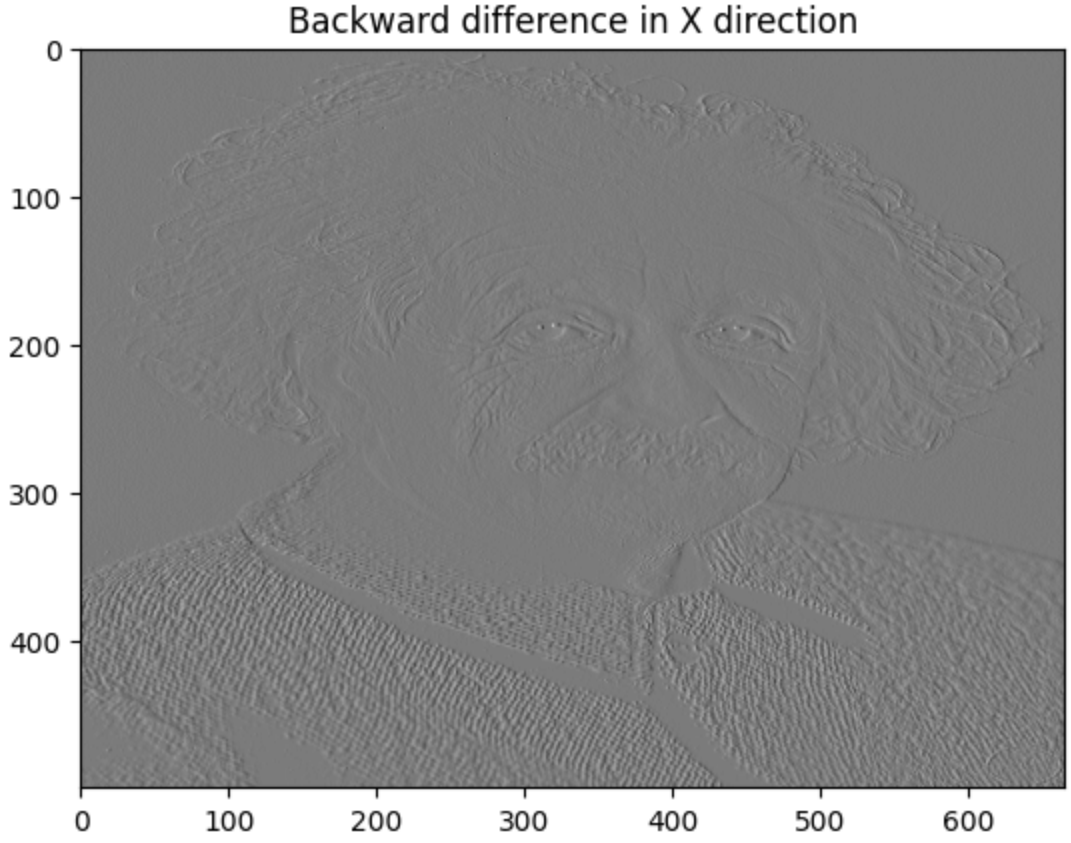
$$\theta(x,\, y) \;=\; \arctan\left(\frac{f_y}{f_x}\right)$$

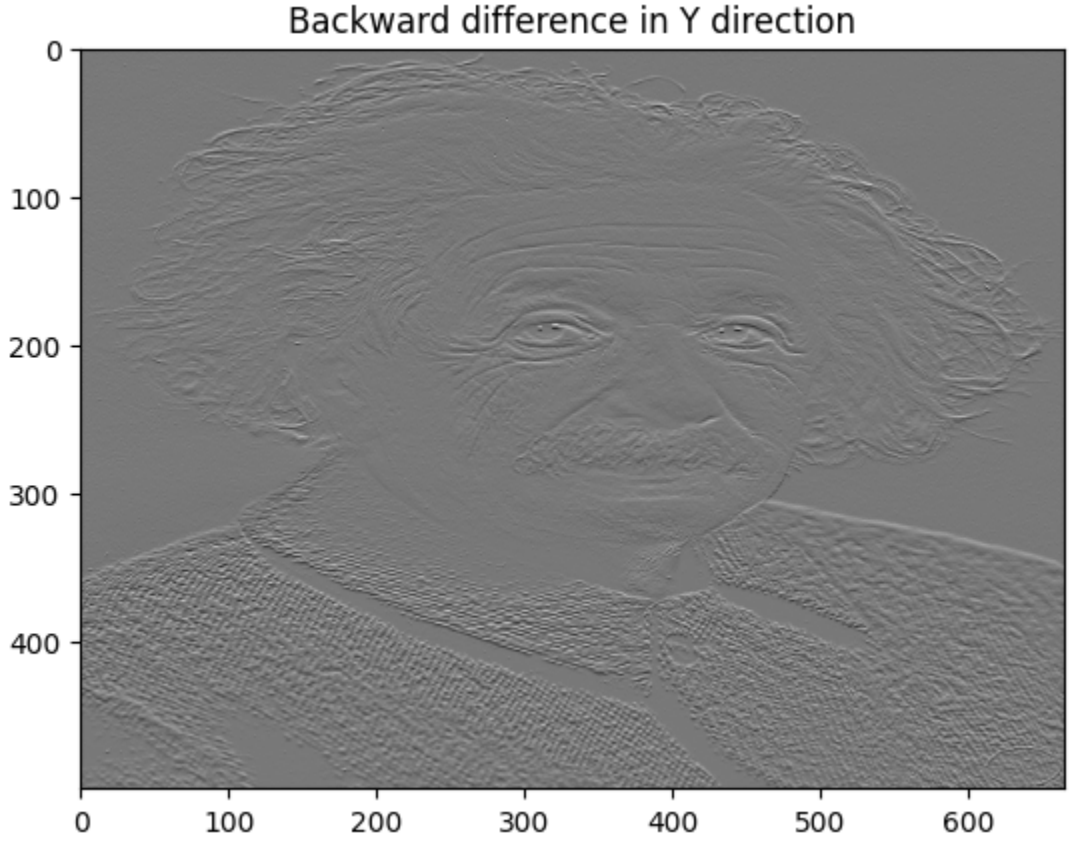
Results:

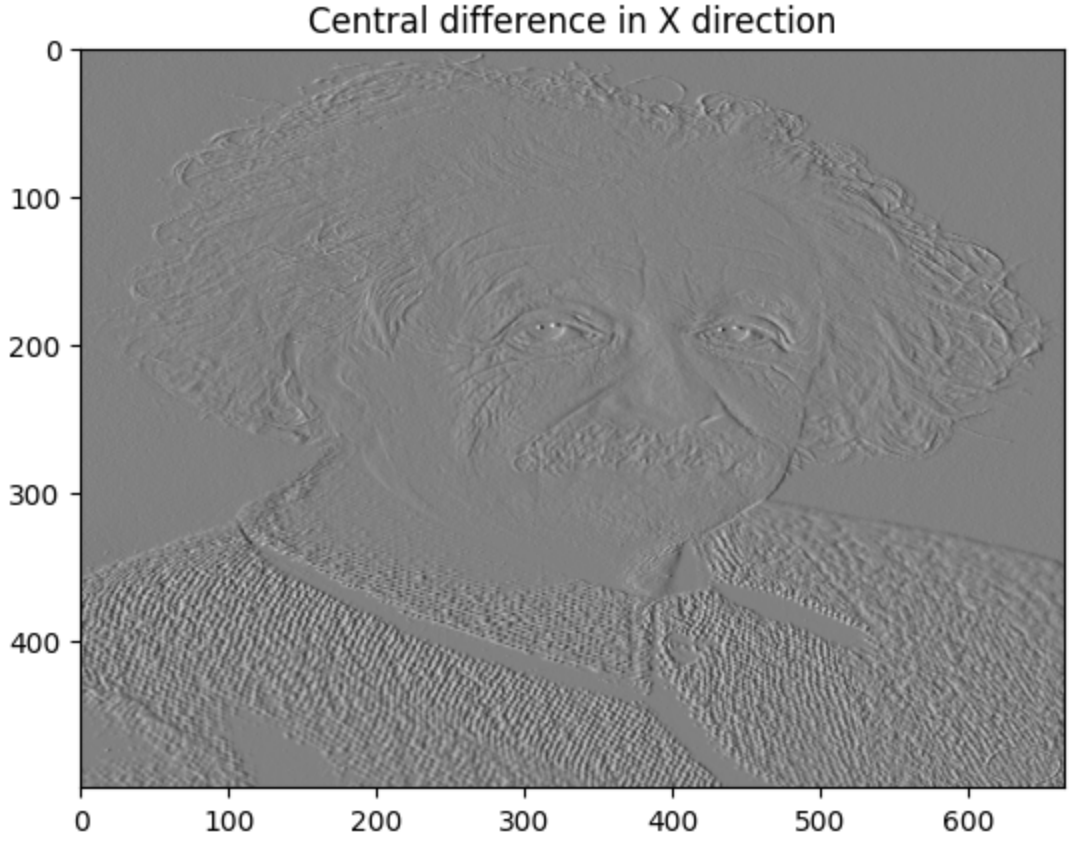

Original image
0
100
200
300
400
0
100
200
300
400
500
600

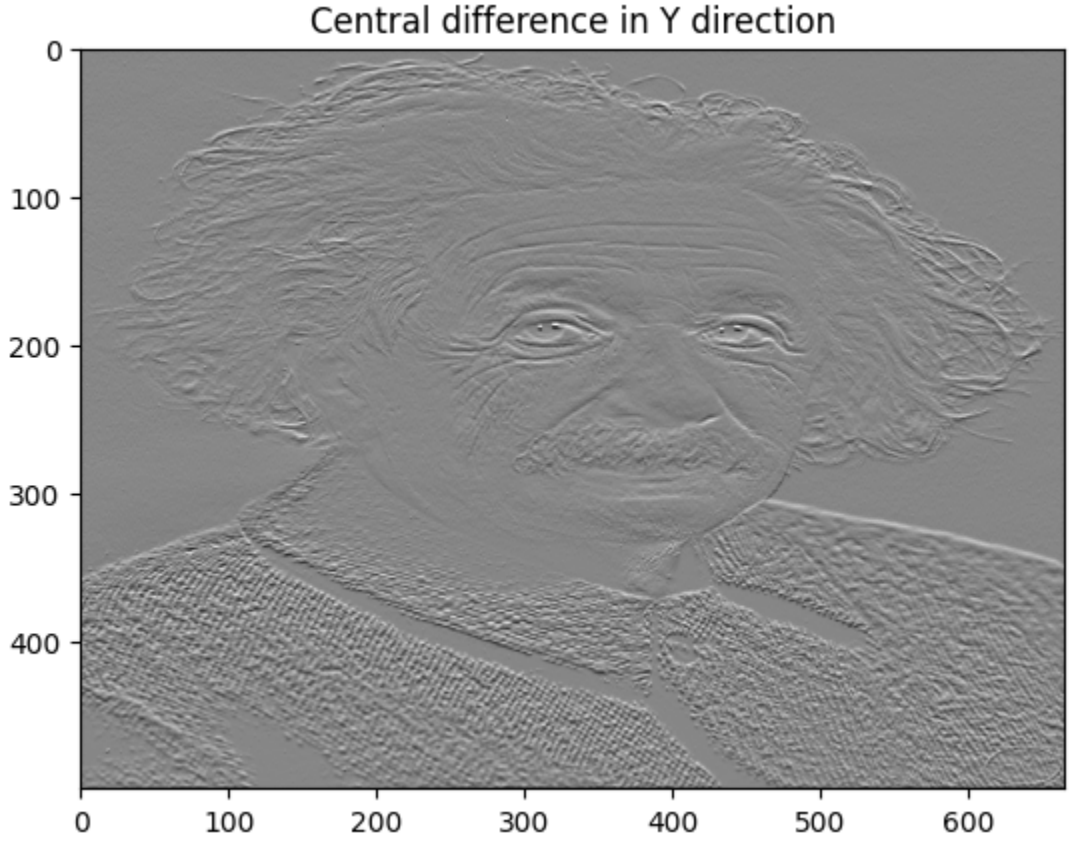

Forward difference in X direction
0
100
200
300
400
0
100
200
300
400
500
600

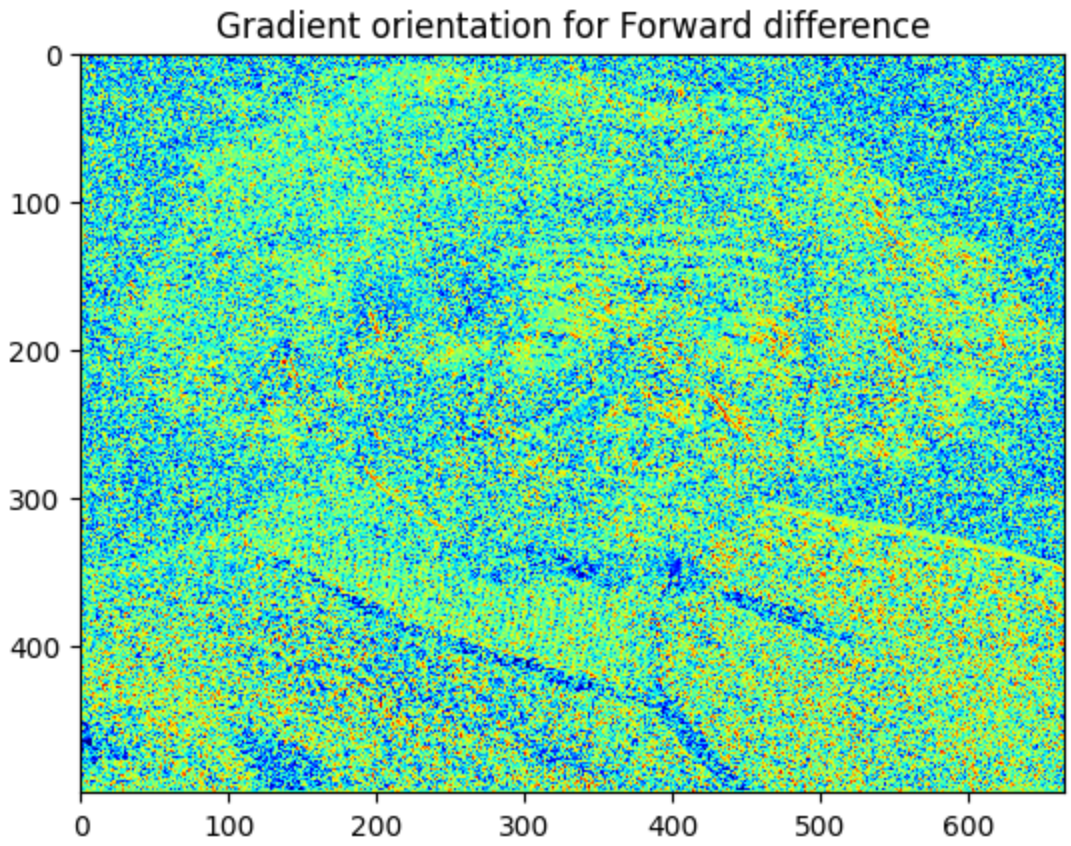

Forward difference in Y direction
0
100
200
300
400
0
100
200
300
400
500
600

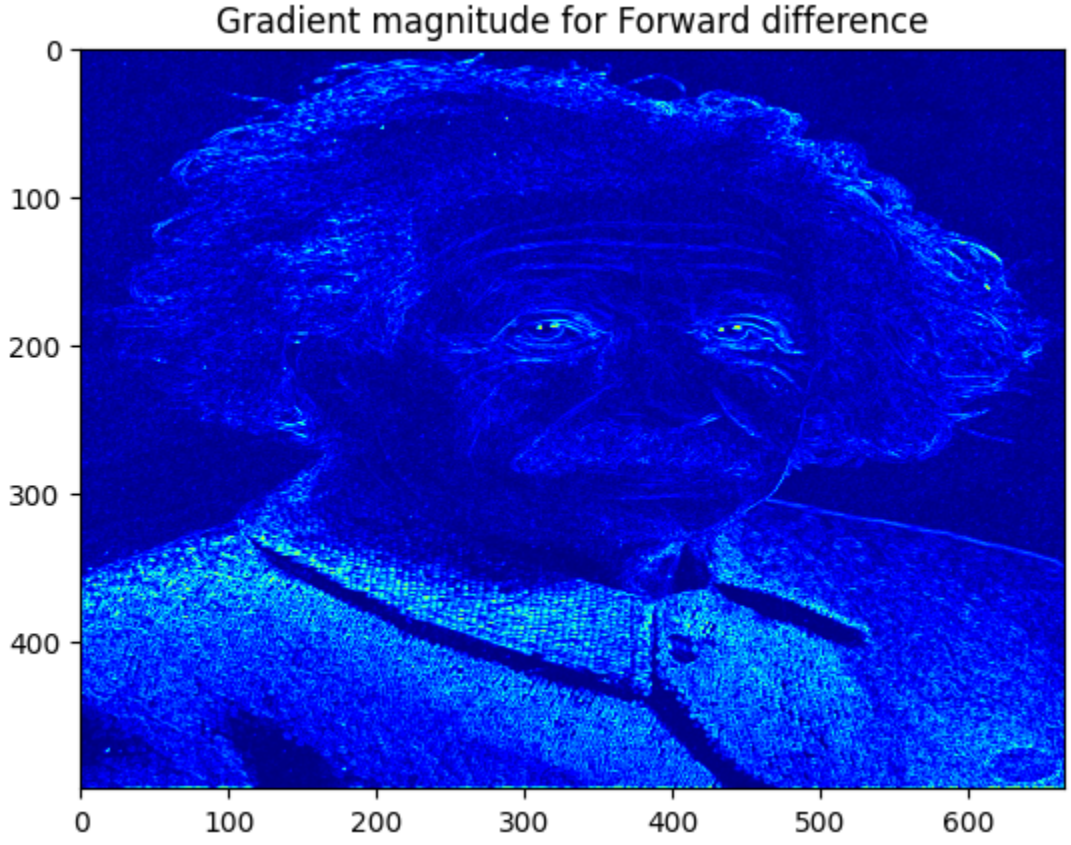

Backward difference in X direction
0
100
200
300
400
0
100
200
300
400
500
600

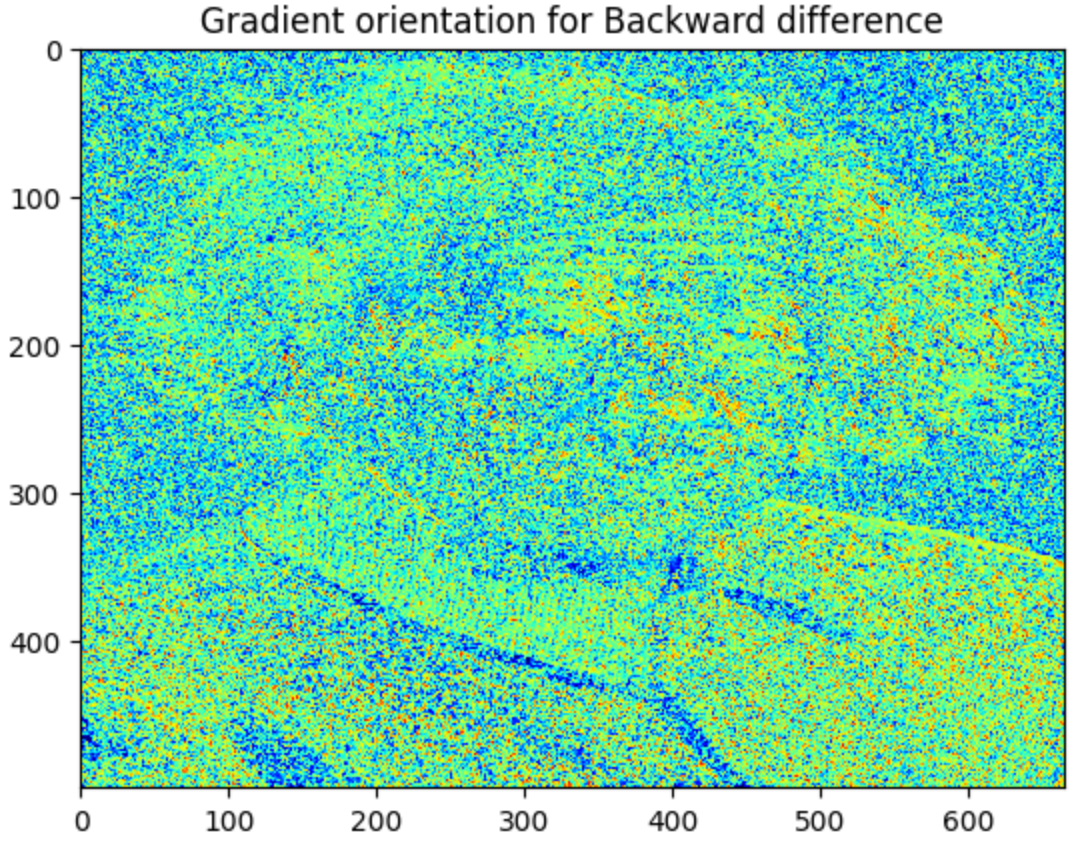

Backward difference in Y direction
0
100
200
300
400
0
100
200
300
400
500
600

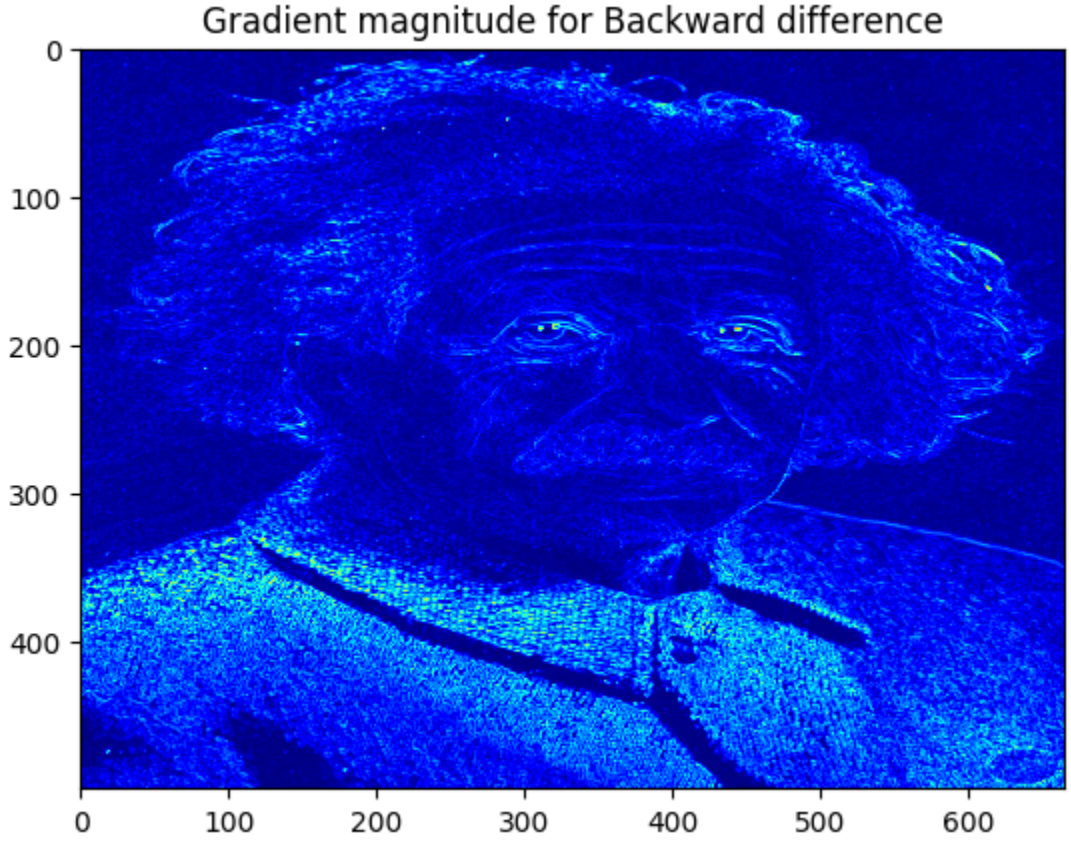

Central difference in X direction
0
100
200
300
400
0
100
200
300
400
500
600

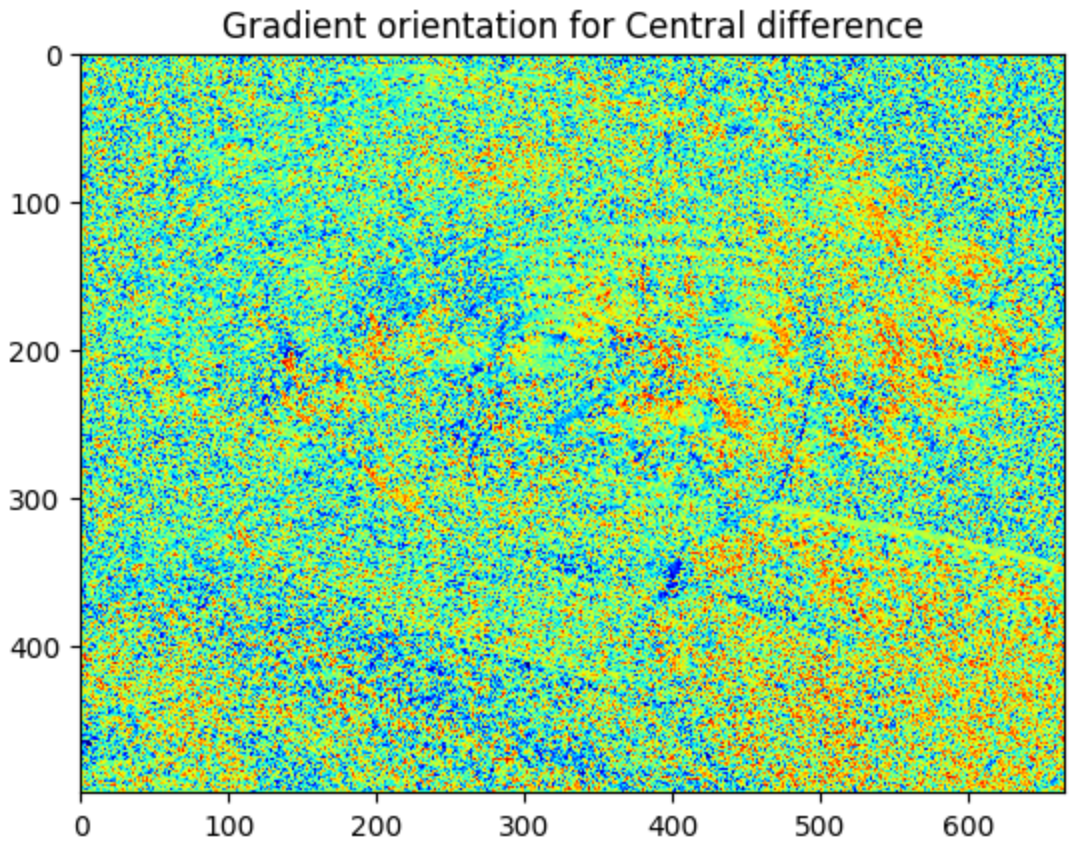

Central difference in Y direction
0
100
200
300
400
0
100
200
300
400
500
600

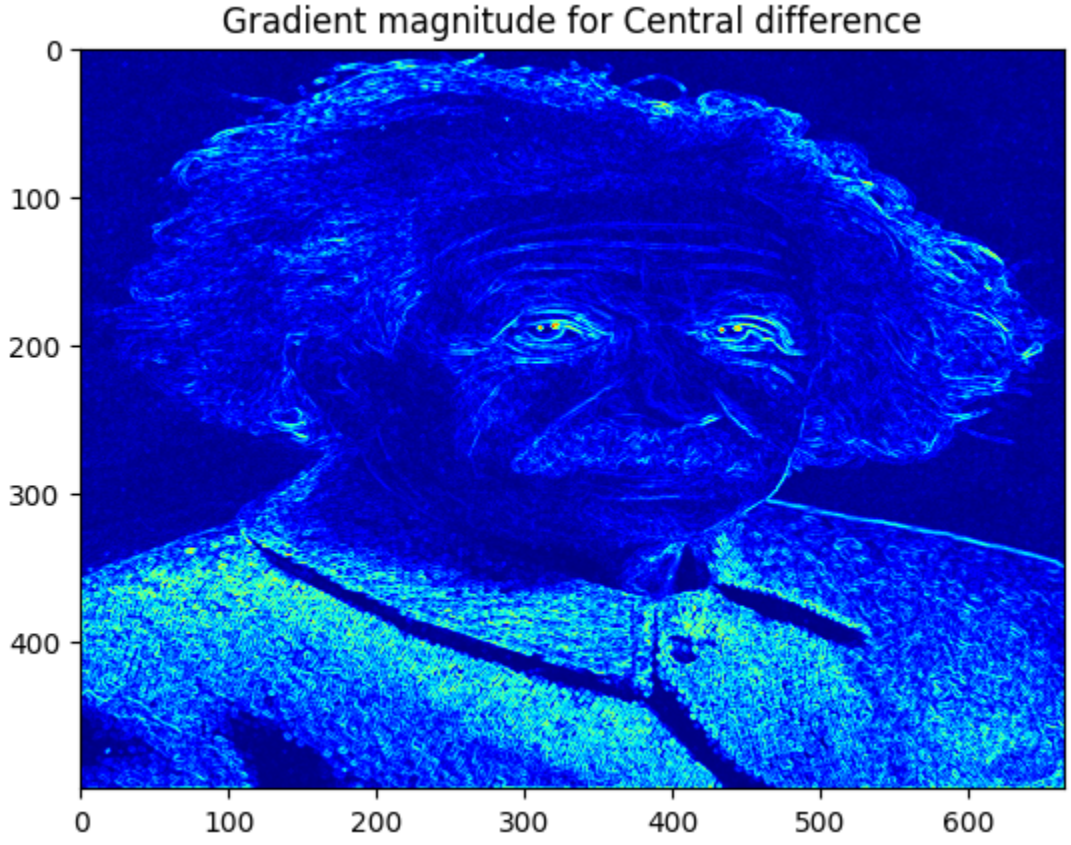

Gradient orientation for Forward difference
0
100
200
300
400
0
100
200
300
400
500
600


Gradient magnitude for Forward difference
0
100
200
300
400
0
100
200
300
400
500
600


Gradient orientation for Backward difference
0
100
200
300
400
0
100
200
300
400
500
600


Gradient magnitude for Backward difference
0
100
200
300
400
0
100
200
300
400
500
600


Gradient orientation for Central difference
0
100
200
300
400
0
100
200
300
400
500
600


Gradient magnitude for Central difference
0
100
200
300
400
0
100
200
300
400
500
600

Note: In the jet colormap, colors range from dark blue(minimum intensity) to dark red(maximum intensity):

Observe that the central difference gives sharper edges and a brighter image, due to the larger window considered during evaluation. The forward and backward differences are largely identical. This is also reflected in the gradient magnitude obtained in all three cases. However, this has a negative effect on the orientation for the central difference, as the Einstein outline is less clearly visible.

The difference in value between the gradient magnitude and gradient orientation is clear from the plots above. Theoretically, only the gradient magnitude gives definitive edge information about the edge, since varying magnitudes are more indicative of the presence of edges than varying orientation, a fact that is used in both the Prewitt and Sobel filters, where central difference is used, along with thresholding of the gradient magnitude to detect edges.

Note: The primary difference between the applied filters and Prewitt and Sobel, is that here, filters have been applied in X and Y directions independently, whereas Prewitt and Sobel use a single 2-D filter that averages and applies the central difference.

The importance of gradient magnitude is clear from the images obtained, as the background (which has no edges, apart from with the boundary of Einstein), and the image of Einstein are clearly differentiable, due to the lack of variation in intensity values in the background. In contrast, the orientation information has less discernible pattern, and the outline of Einstein is less clearly differentiable from the background. However, there are patterns that are clearly visible, for instance the orientation of the hair or wrinkles. These properties are combined in the Canny edge detector, where low intensity points are suppressed based on the gradient orientations, and edges are detected by considering the neighborhood of the pixels with highest magnitude of gradient.

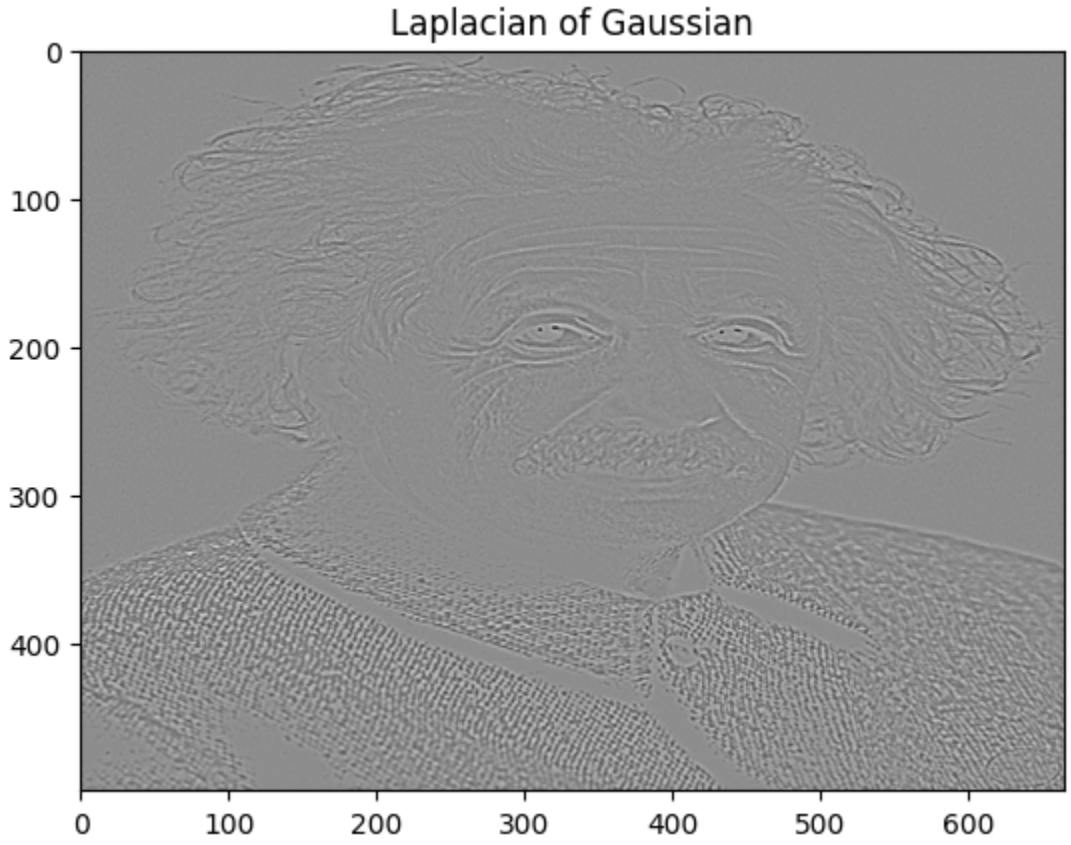
Q2:

Gaussian filter expression:

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

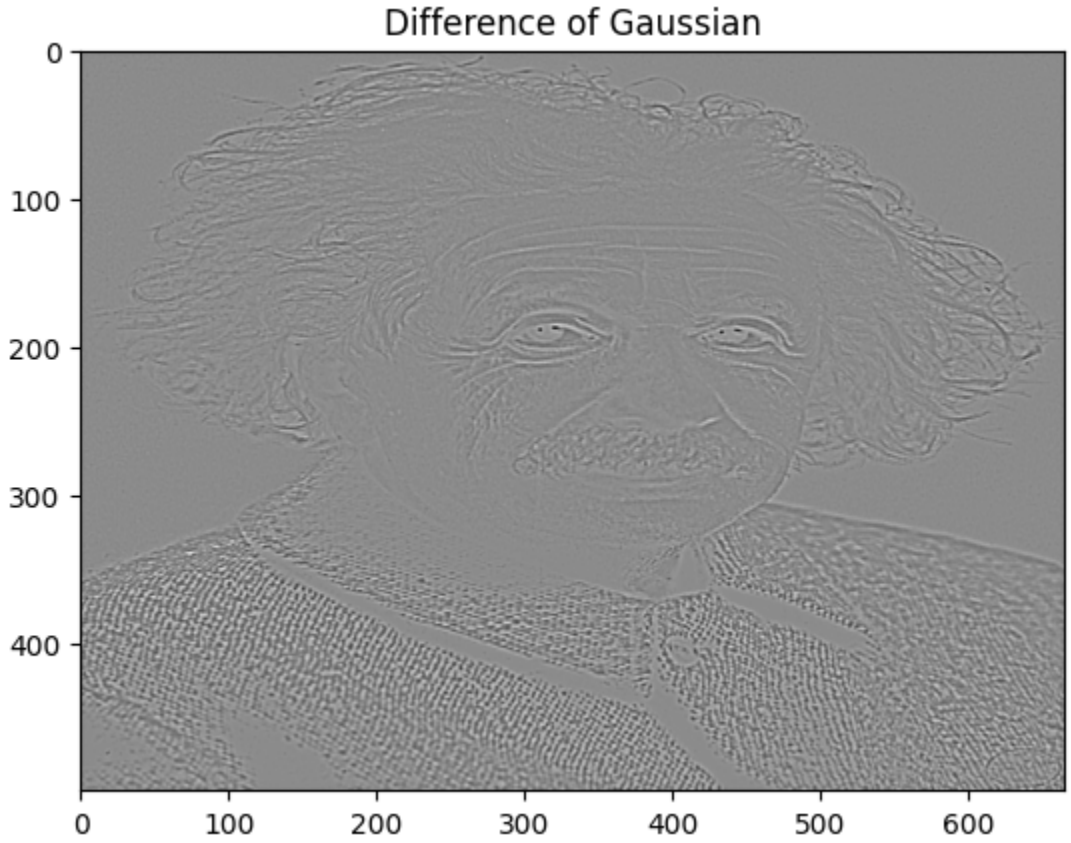
LoG:

The Laplacian of Gaussian, as the name suggests, is computed by first applying the Gaussian filter on the image and then taking the Laplacian, or second derivative, of the result.



DoG:

The difference of Gaussian (DoG) is also self-explanatory. It is the subtraction of one Gaussian blurred version of an image from another, less blurred version of the original. Here, more and less blurred correspond to a higher and lower sigma value, respectively.



Note: Theoretically, the LoG can be approximated as a DoG, up to a scaling factor of $(k - 1)\sigma^2$, where k is the scale applied to σ for the more blurred image. The similarity that is expected from this result is visible in the obtained result.

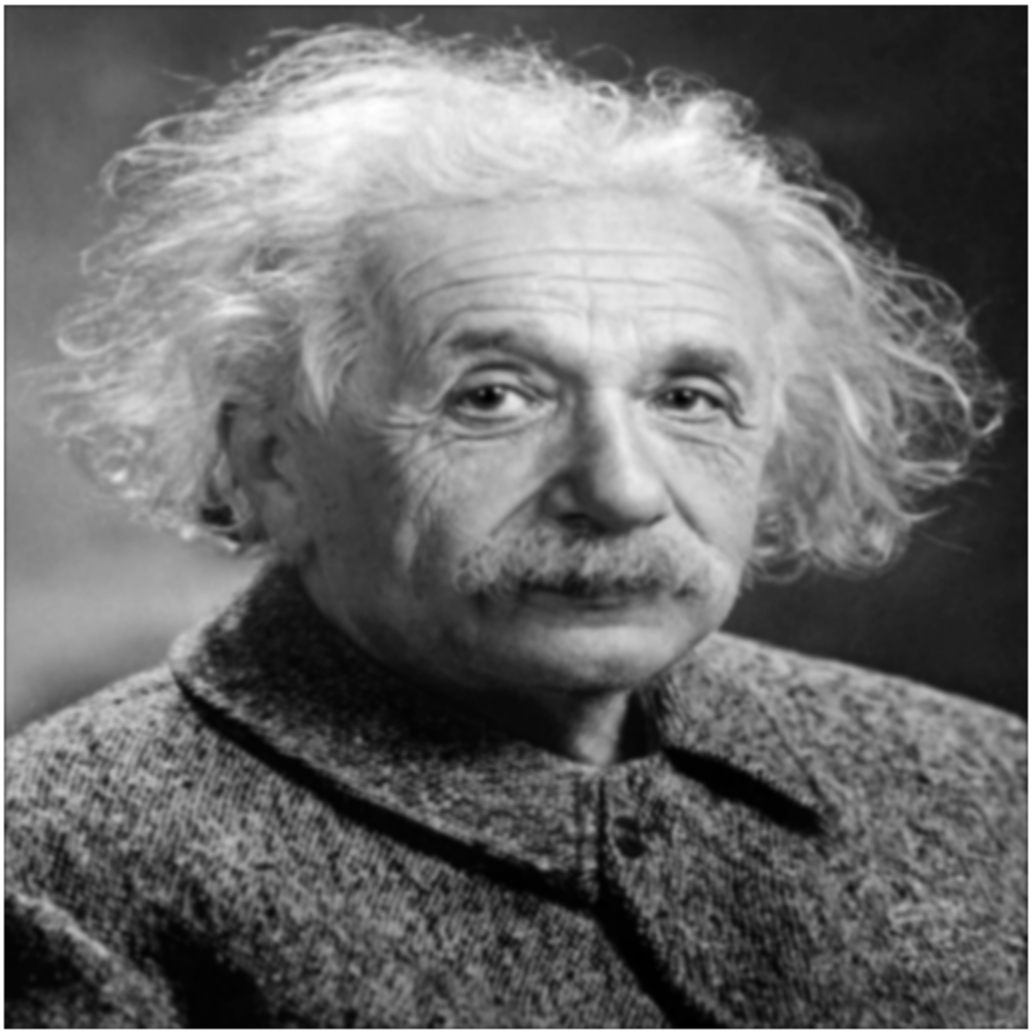
Gaussian Pyramid:

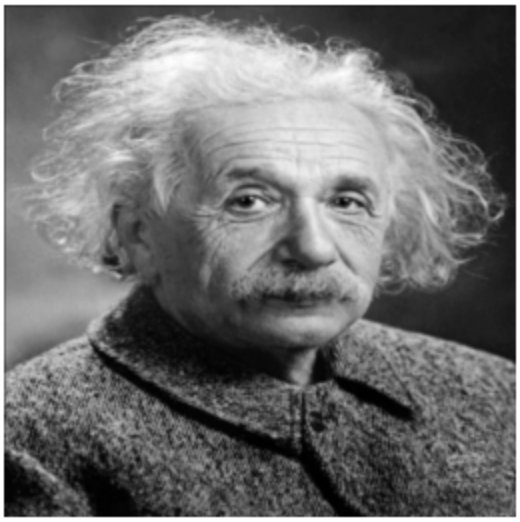
The Gaussian pyramid is more complex. It comprises multiple versions of the same image at different resolutions and scales. The lowest level is the Gaussian blurred version of the original image, and subsequent levels are formed by applying the Gaussian on the preceding level, and then downsampling, i.e. halving the rows and columns. This is done by simply selecting only the even rows and columns from the larger image, resulting in an image that is one-fourth of the original image. These can be combined in one equation as follows:

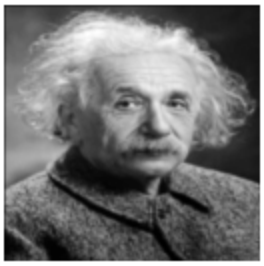
$$g_l(i, j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m, n)\, g_{l-1}(2i + m, 2j + n)$$

, where g_l is the Gaussian image at the l^{th} level, and w is the 5 × 5 Gaussian filter.

Here, four levels are shown, from 512 × 512 to 64 × 64.







The reduction in size, and increase in smoothing is clear for each subsequent layer.

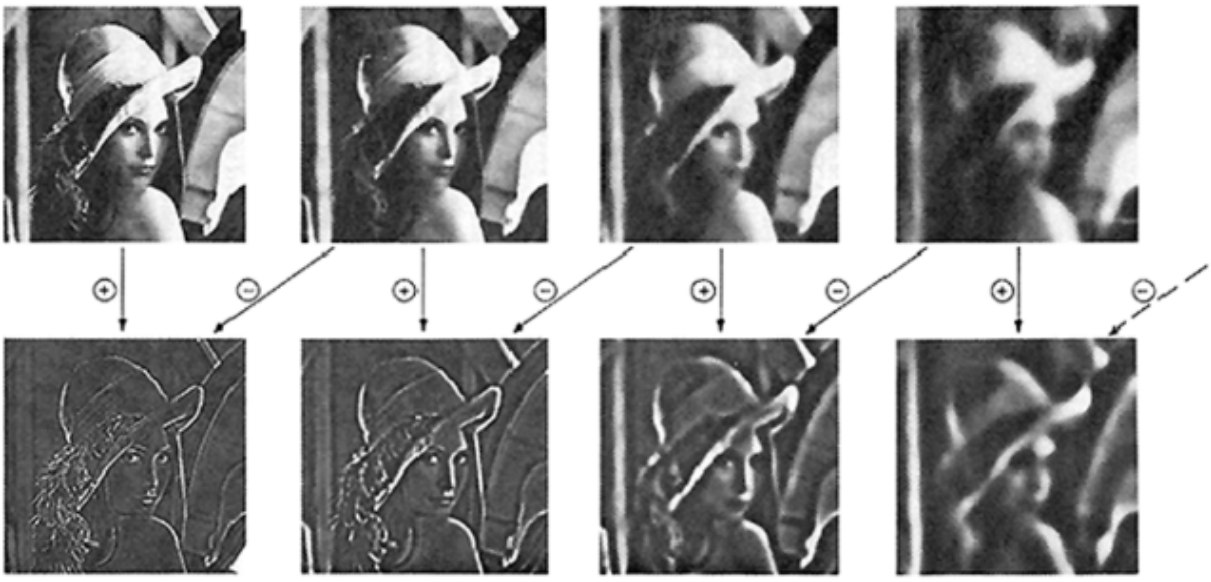
Q3:

The procedure to merge an image of an apple and orange is as follows:

Algorithm

- Generate Laplacian pyramid Lo of orange image.
- Generate Laplacian pyramid La of apple image.
- Generate Laplacian pyramid Lc by
  - copying left half of nodes at each level from apple and
  - right half of nodes from orange pyramids.
- Reconstruct combined image from Lc.

We have already covered how to construct the Gaussian pyramid for a given image. In order to construct the Laplacian pyramid for the same image, we can use the concept mentioned earlier, namely that the LoG can be approximated as a DoG. Hence, taking the difference between the Guassian image at level l, and the upsampled version of the Gaussian image at level l+1 will approximate the Laplacian image at level l:



The smallest Laplacian image is set to be equal to the smallest Gaussian image as the base case for forming the Laplacian pyramid:

$L_n = g_n$

$L_i = g_i$ - Smooth(Upsample(g_{i+1}))

, where g_i is the Gaussian image at the i^{th} level, and smoothing is done using the Gaussian filter. The equation to smooth and upsample from level l + 1 to level l is as follows:

$$g_l(i, j) = \sum_{p=-2}^{2} \sum_{q=-2}^{2} w(m, n)\, g_{l+1}\left(\left(\frac{i-p}{2}\right), \left(\frac{j-q}{2}\right)\right)$$

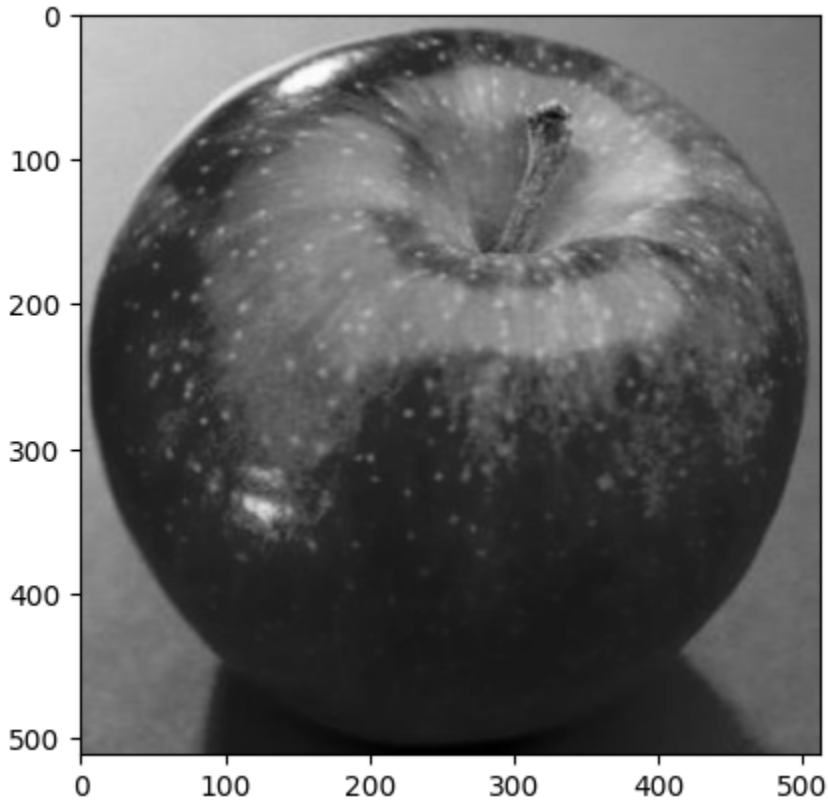
Note: While upsampling, we first place the original image in the even rows and columns of the upsampled image. The remaining values are obtained by linear interpolation along first the rows and then the columns, i.e. taking the average of the pixels above and below the pixel of interest for each odd row, and then taking the average of the pixels to the left and right of the pixel of interest for each odd column. While going up the pyramid, or reducing, we downsample after smoothing the image at the current level, whereas when going down the pyramid, or expanding, we upsample and then apply the Gaussian, to obtain the lower level.

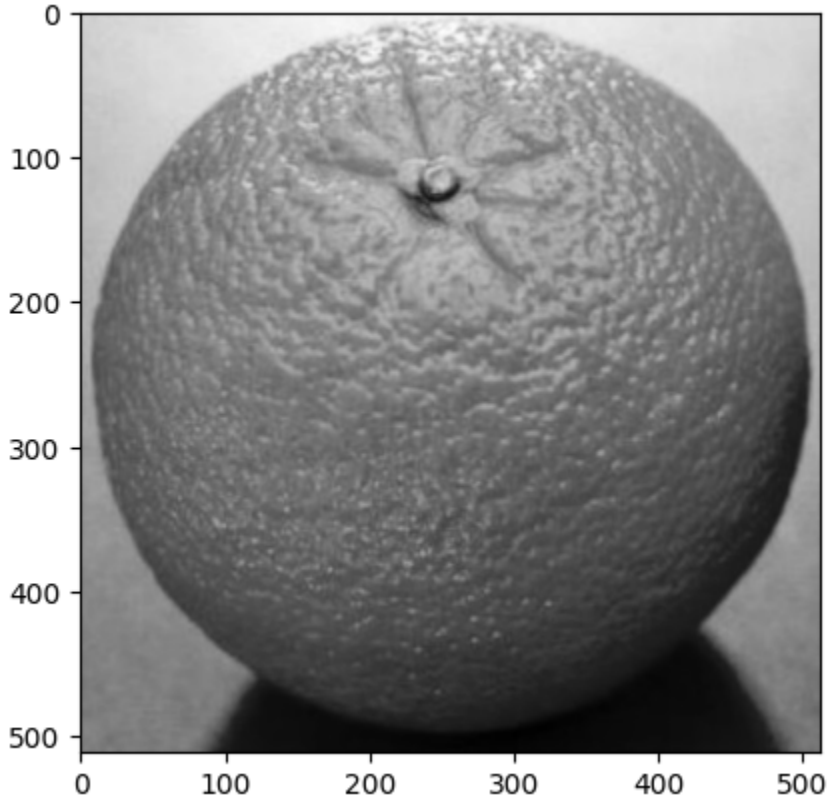
Once the combined Laplacian pyramid is obtained from the individual Laplacian pyramids, we decode the Laplacian pyramid back to the Gaussian pyramid, by reversing the above equations, i.e.:

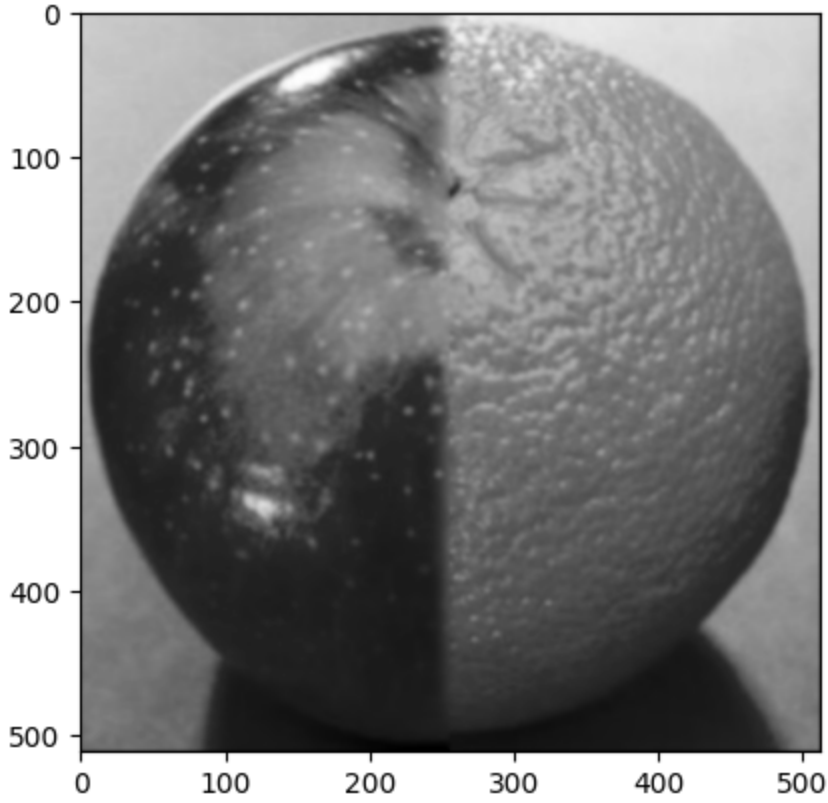
$g_n = L_n$

$g_i = L_i$ + Smooth(Upsample(g_{i+1}))

The Gaussian image at the lowest level is the combined image


0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500



The combined image is clearly half of each of the original images.