ST. FRANCIS INSTITUTE OF TECHNOLOGY

Mount Poinsur, S.V.P. Road, Borivali (W), Mumbai - 400 103.

CLASS / BATCH: TE- EXTCA /TA-3 SEM: VI

NAME: Om Kadam ROLL NO: 41

SUBJECT: IPMV

EXPERIMENT NO. : 03

TITLE: To perform Image enhancement using masking process

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

Correction Parameters	Marks Allotted	Correction Parameters	Marks Allotted
Submission on time [10%]		Experimental Result [30%]	
Conclusion / Result Analysis/ Discussion [30%]		Post Experiment Exercise [20%]	
Presentation of Write-Up [10%]			

TOTAL MARKS:

FACULTY SIGNATURE (WITH DATE)

Roll No. ____

Experiment – 03: To Perform Image Enhancement using Masking Process

Date: _____

1. **Aim :** To perform image enhancement using neighborhood processing technoques
   a. Low Pass Filter, b. High Pass Filter c. Un-sharp filter
   d. Sobel operator

2. **Requirements:** Python

3. **Pre-Experiment Exercise**
   3.1 Brief Theory

The filtering process simply consists of moving the filter mask from point to point in an image. At each point, the response of the filter at that point is calculated using a predefined relationship. For linear spatial filtering, the response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask. The expression to find the response R, of an m x n image at any point (x, y) is as shown below:

$$R = w_1z_1 + w_2z_2 + \ldots + w_{mn}z_{mn}$$
$$= \sum_{i=1}^{mn} w_i z_i$$

where, the w's are mask coefficients, the z's are the values of the image gray levels corresponding to those coefficients, and mn is the total number of coefficients in the mask.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

a. Low-Pass or Smoothing Spatial Filters

The output of a smoothing spatial low-pass filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters are sometimes are called *averaging filters*. The idea behind smoothing is to replace the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask. This process results in an image with reduced "sharp" transitions in gray levels. Smoothing filters are used for blurring and for noise reduction. Two such 3x3 smoothing filter mask are given below:

1	1	1
1	1	1
1	1	1

Average filter mask

1	2	1
2	4	2
1	2	1

weighted average filter mask

Note that the constant multiplier in front of each mask is equal to the sum of the values of its coefficient, as it is required to compute an average.

b. High-Pass or Sharpening Spatial Filters

The sharpening of an image can be accomplished by spatial differentiation. We use filters that are based on first- and second- order derivatives for image sharpening. The filter employing the second derivative is preferred over the one employing derivative because of the ability of the former to enhance fine details. A Laplacian shows filter mask employed are shown below:

1	1	1
1	-8	1
1	1	1

c. Un-sharp filter

The unsharp filter is a simple sharpening operator which derives its name from the fact that it enhances edges (and other high frequency components in an image) via a procedure which subtracts an unsharp, or smoothed, version of an image from the original image. The unsharp filtering technique is commonly used in the photographic and printing industries for crispening edges.

d. Sobel operator

The most commonly used discontinuity based edge detection techniques are Sobel Edge Detection, Prewitt edge detection techniques. Such discontinuities are detected by using first and second derivatives. The first order derivative of choice in image processing is the gradient. The gradient of a 2-D function, f(x, y), is defined as the vector. The magnitude of this vector is approximately equal to

$$\nabla f = |Gx| + |Gy|$$

These approximations still behaves as derivatives; that is, they are zero in areas of constant intensity and their values are related to the degree of intensity change in areas of variable intensity. The Sobel technique performs a 2-D spatial gradient quantity on an image and so highlights regions of high spatial frequency that correspond to edges. In general it is used to find the estimated absolute gradient magnitude at each point in n input grayscale image. In conjecture at least the operator consists of a pair of 3x3 complication kernels as given away in under table. One kernel is simply the other rotated by 90°. This is very alike to the Roberts Cross operator. The operator consists of a pair of 3×3 convolution kernels as shown in below figure. One kernel is simply the other rotated by 90°.

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

Gy

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

Gx

It provides both a differentiating and a smoothing effect, which is particularly attractive as derivatives typically enhance noise.

4. **Laboratory Exercise**

4.1 **Algorithm:**

1. Take a gray scale image as an input and define 3 X 3 mask filter.
2. Apply different types of filter masks as mentioned above on an input image with zero padding. Enter the horizontal and vertical size of the mask.
3. Display all the output images using different mask.

5. **Post-Expriment Exercice**

5.1 **Conclusion:**

This experiment demonstrated the effectiveness of masking processes to enhance image quality through spatial frequency and morphological operations. Spatial filtering reduced noise, improved clarity, frequency domain filtering selectively enhanced specific features.

5.2 Questions:

1. Write short note on: Spatial domain filtering.
2. Give the difference between first order derivative filter and second order derivative filter.
3. Apply Low and High Pass Spatial masks on the following image matrix.

   Assume virtual Rows and Columns.

   f(x,y) = 30 51 52
   33 120 30
   32 52 31

5. Prove that High Pass filter = Original image – Low pass filter.

1) Spatial domain filtering is a fundamental technique used in image processing for modifying images, it operate directly on the pixel value of an image in spatial domain rather than the transforming it into another domain like frequency. This ~~technology~~ technique involves applying the filter which is typically a matrix of coefficient to each pixel of the image.

2)

First Order Derivative Filter	Second Order Derivative Filter
① It measures the rate of change of a quantity over time or input.	It measures the rate of change of the first order derivative.
② It cannot be used to measure acceleration	It can be used to measure acceleration.
③ It is more sensitive to small changes in the input signal	It is more sensitive to large changes in the input signal.
④ Commonly used in physics and engineering	Used in more advanced applications such as control systems and signal processing.

3) Applying lowpass filter, $\begin{bmatrix} 30 & 31 & 32 \\ 33 & 120 & 30 \\ 32 & 32 & 31 \end{bmatrix}$

$$\text{MatMx} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 30 & 31 & 32 \\ 33 & 120 & 30 \\ 33 & 32 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 11 & 20 & 10 \\ 11 & 20 & 10 \\ 11 & 20 & 10 \end{bmatrix}$$

Applying highpass filter,

$$= \frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$= \frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 30 & 31 & 32 \\ 33 & 120 & 30 \\ 33 & 32 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} -4 & -13 & -3 \\ 22 & 100 & 20 \\ -10 & -20 & -10 \end{bmatrix}$$

4) Original Image

Let LPF

4) Original Image - LPF = HPF

Let LPF be $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

$$HPF = \frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{bmatrix}$$

$$z_5 = \left\{ \frac{z_1}{9} + \frac{z_2}{9} + \frac{z_3}{9} + \frac{z_4}{9} + \frac{z_5}{9} + \frac{z_6}{9} + \frac{z_7}{9} + \frac{z_8}{9} \right\}$$

$$= -\frac{z_1}{9} - \frac{z_2}{9} - \frac{z_3}{9} - \frac{z_4}{9} + \frac{8z_5}{9} - \frac{z_6}{9} - \frac{z_7}{9} - \frac{z_8}{9}$$

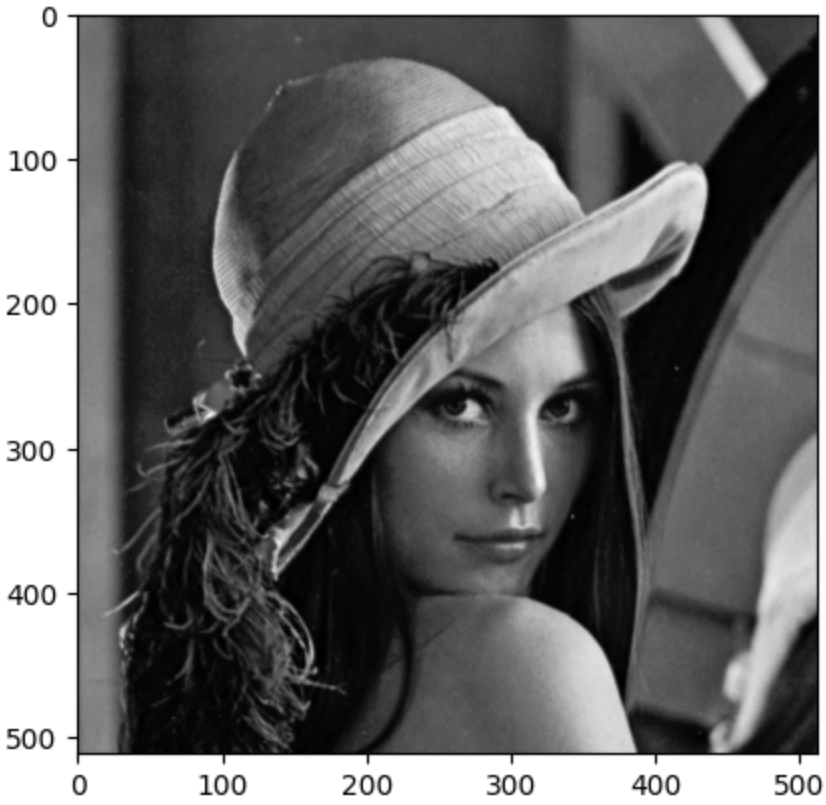
$$HPF = \frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Hence, proved.

In [4]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
from google.colab.patches import cv2_imshow
from google.colab import files
# uploaded = files.upload()
```

In [5]:
```
img = cv2.imread('lena_gray.tif')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img,cmap=plt.cm.gray)
```

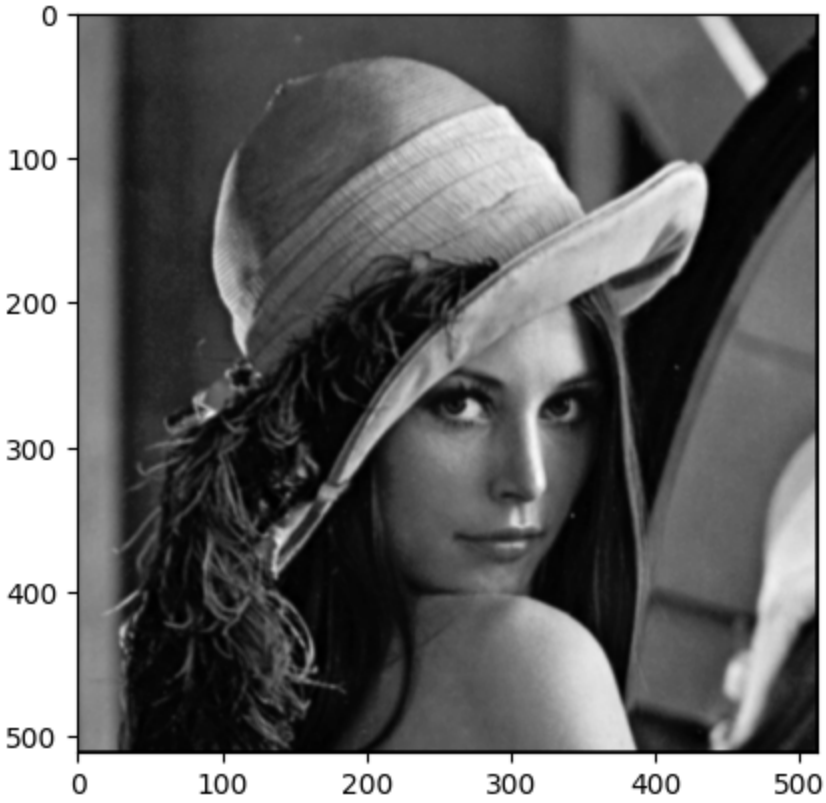
Out[5]: <matplotlib.image.AxesImage at 0x785b60a40490>



In [6]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask = np.array([[1,2,1],[2,4,2],[1,2,1]])
mask = mask / 16

#Rmask=np.array([[1,0],[0,-1]])
mask

img_new = np.zeros([m, n])

img_new
```

```
for i in range(1, m-1):
  for j in range(1, n-1):
    temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[0,
    img_new[i, j]= temp

img_new = img_new.astype(np.uint8)
plt.imshow(img_new,cmap=plt.cm.gray)

img_new
```

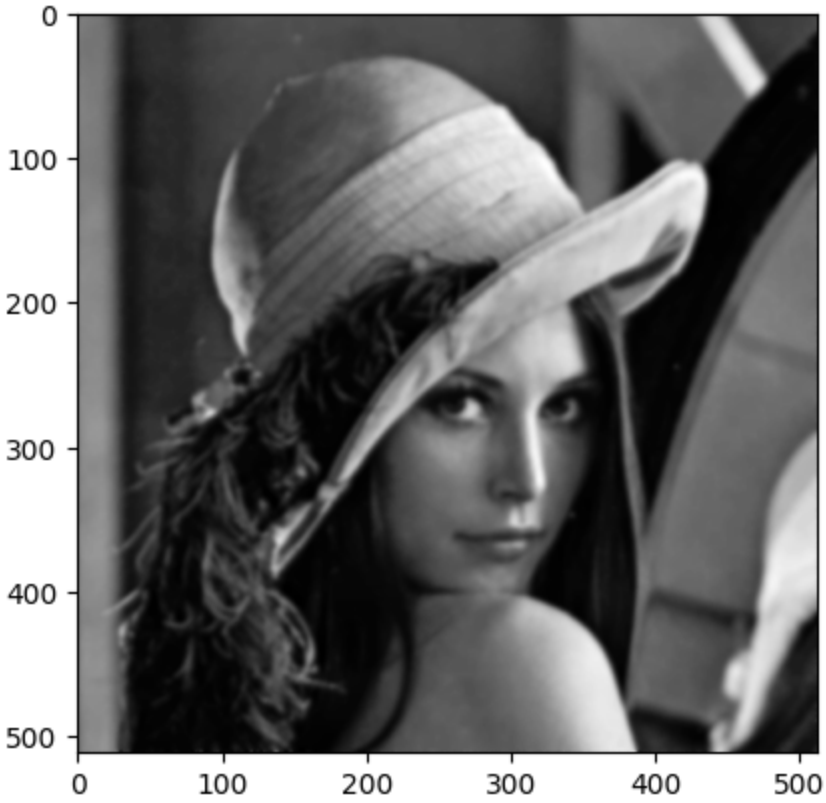
Out[6]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  0, 105, 106, ...,  59,  59,   0],
       [  0, 106, 106, ...,  59,  59,   0],
       ...,
       [  0,  96, 102, ...,  19,  23,   0],
       [  0,  95, 100, ...,  21,  26,   0],
       [  0,   0,   0, ...,   0,   0,   0]], dtype=uint8)
```



In [7]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask=np.array([[1,2,1],[2,4,2],[1,2,1]])
mask = mask / 16

#Rmask=np.array([[1,0],[0,-1]])
mask

img1=img
```

```
for c in range(1,5):
  for i in range(1, m-1):
    for j in range(1, n-1):
      temp = img1[i-1, j-1]*mask[0, 0]+img1[i-1, j]*mask[0, 1]+img1[i-1, j + 1]*mas
      img1[i, j]= temp
img1 = img1.astype(np.uint8)
plt.imshow(img1,cmap=plt.cm.gray)

img1
```

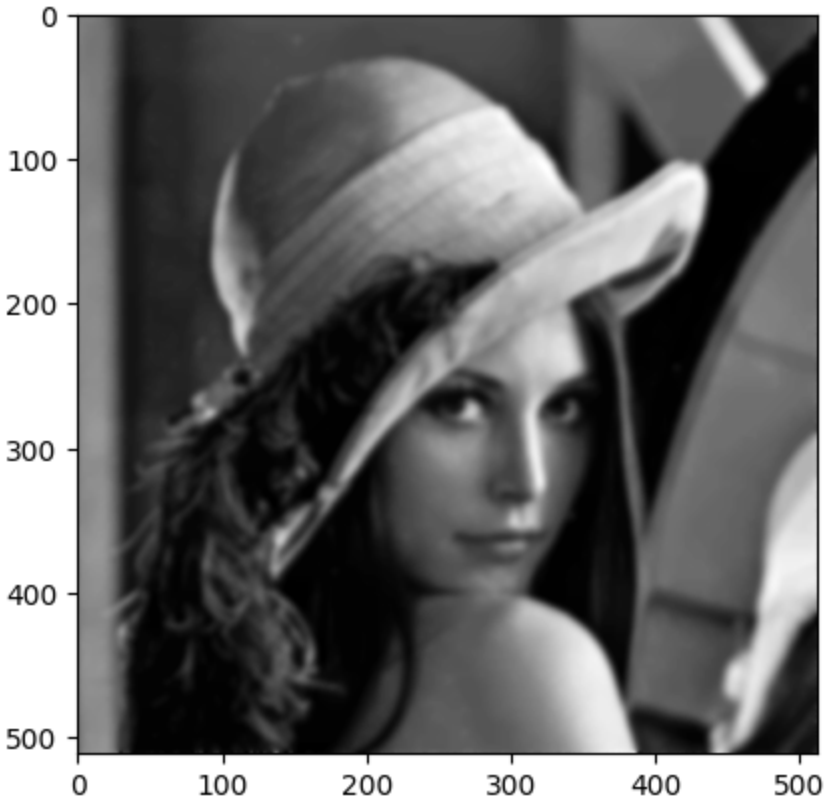
Out[7]:
```
array([[105, 105, 106, ...,  60,  59,  59],
       [106, 105, 105, ...,  58,  59,  60],
       [106, 104, 104, ...,  58,  59,  60],
       ...,
       [ 89,  93,  98, ...,  20,  24,  30],
       [ 88,  94,  99, ...,  23,  28,  32],
       [ 89,  98, 101, ...,  28,  34,  39]], dtype=uint8)
```



In [8]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask=np.array([[1,2,1],[2,4,2],[1,2,1]])
mask = mask / 16
img3 = cv2.imread('lena-gray.jpg')

#Rmask=np.array([[1,0],[0,-1]])
mask
```

```
img2=img1


for c in range(1,5):
  for i in range(1, m-1):
    for j in range(1, n-1):
      temp = img2[i-1, j-1]*mask[0, 0]+img2[i-1, j]*mask[0, 1]+img2[i-1, j + 1]*mas
      img2[i, j]= temp
img2 = img2.astype(np.uint8)
plt.imshow(img1,cmap=plt.cm.gray)

img2
```

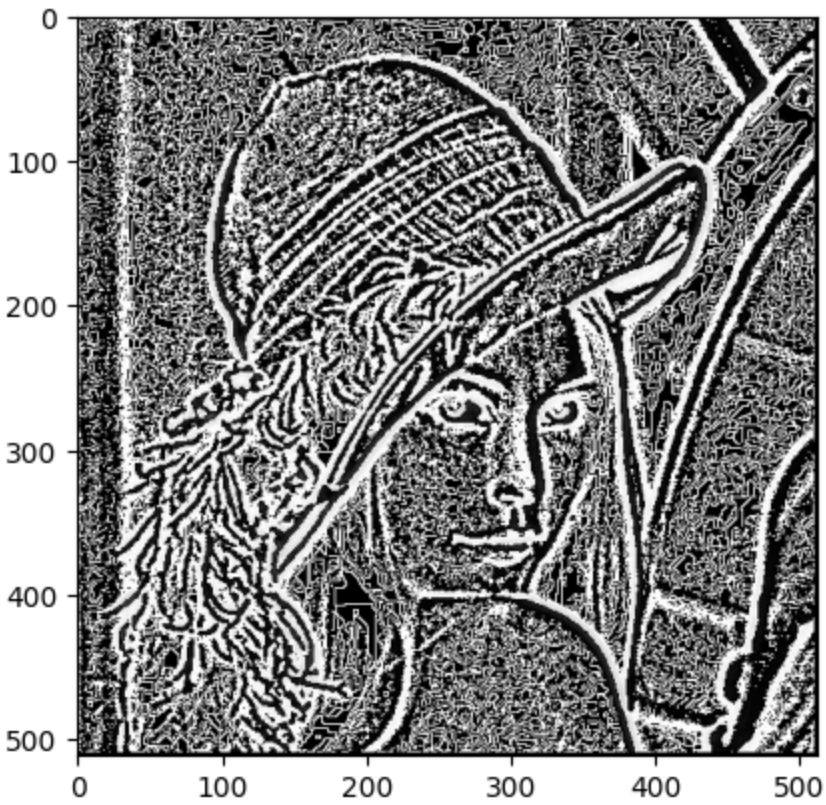
Out[8]:
```
array([[105, 105, 106, ...,  60,  59,  59],
       [106, 104, 104, ...,  58,  59,  60],
       [106, 104, 103, ...,  58,  59,  60],
       ...,
       [ 89,  92,  96, ...,  20,  24,  30],
       [ 88,  93,  98, ...,  23,  28,  32],
       [ 89,  98, 101, ...,  28,  34,  39]], dtype=uint8)
```



In [9]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask=np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])

#Rmask=np.array([[1,0],[0,-1]])
mask
```

```
img_new = np.zeros([m, n])

img_new

for i in range(1, m-2):
  for j in range(1, n-2):
    temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[0,
    img_new[i, j]= temp
img_new = img_new.astype(np.uint8)
plt.imshow(img_new,cmap=plt.cm.gray)

img_new
```

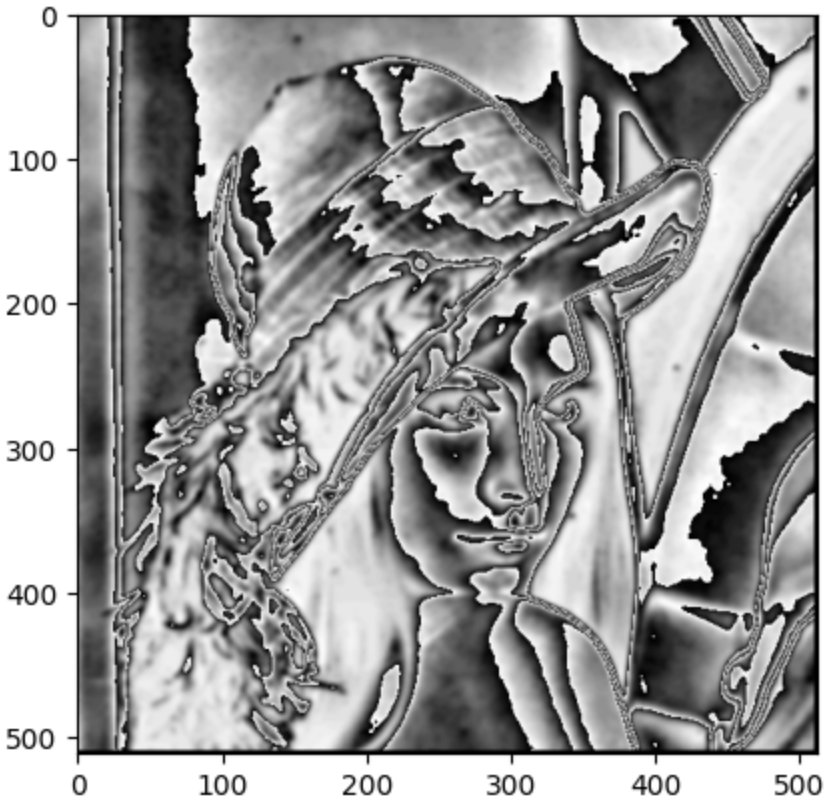
Out[9]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  0, 255,   3, ..., 250,   0,   0],
       [  0, 251,   2, ..., 255,   0,   0],
       ...,
       [  0,   0,   1, ..., 251,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0]], dtype=uint8)
```



In [10]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask=np.array([[-1,0,-1],[-1,0,-1],[-1,0,-1]])

#Rmask=np.array([[1,0],[0,-1]])
mask
```

```
img_new = np.zeros([m, n])

img_new

for i in range(1, m-2):
  for j in range(1, n-2):
    temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[0,
    img_new[i, j]= temp
img_new = img_new.astype(np.uint8)
plt.imshow(img_new,cmap=plt.cm.gray)

img_new
```

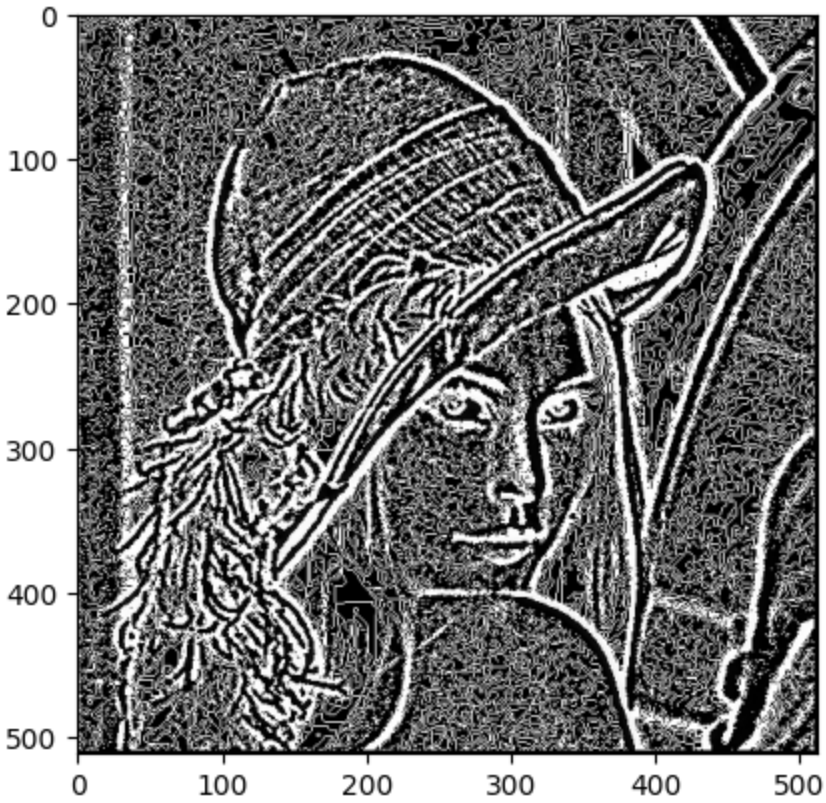
Out[10]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  0, 136, 141, ..., 160,   0,   0],
       [  0, 139, 146, ..., 163,   0,   0],
       ...,
       [  0, 210, 181, ..., 132,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0]], dtype=uint8)
```



In [11]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask=np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])

#Rmask=np.array([[1,0],[0,-1]])
mask
```

```
img_new = np.zeros([m, n])

img_new

for i in range(1, m-2):
  for j in range(1, n-2):
    temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[0,
    img_new[i, j]= temp
img_new = img_new.astype(np.uint8)
plt.imshow(img_new,cmap=plt.cm.gray)

img_new
```

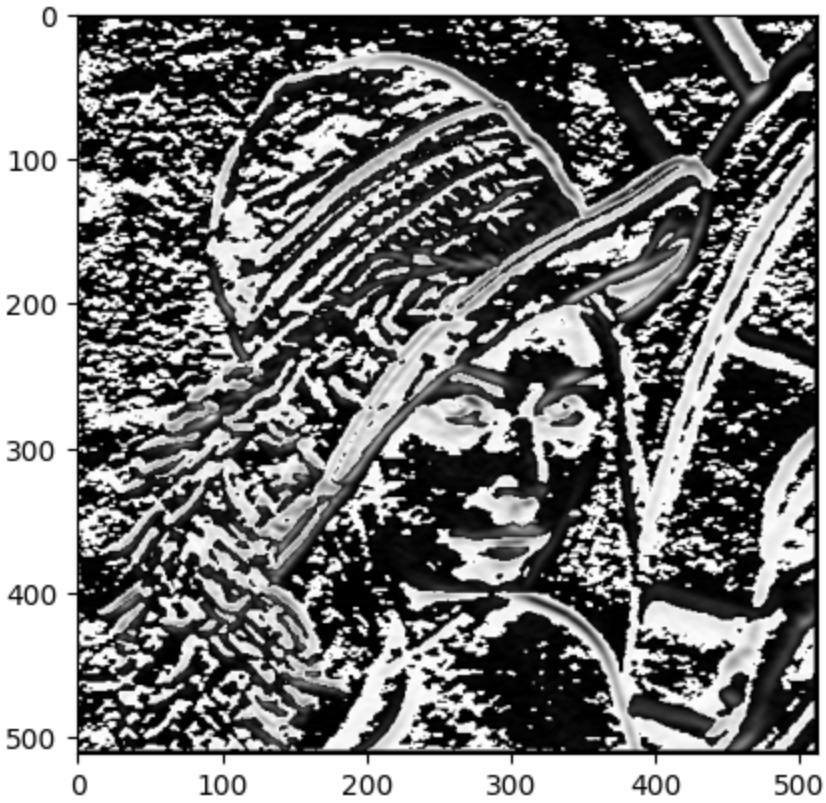
Out[11]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   1, ..., 253,   0,   0],
       [  0, 254,   1, ...,   0,   0,   0],
       ...,
       [  0, 255,   1, ..., 255,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0]], dtype=uint8)
```



In [12]:
```
[m,n] = img.shape;

m,n

#mask = np.ones([3, 3], dtype = int)
mask=np.array([[1,2,1],[0,0,0],[-1,-2,-1]])

#Rmask=np.array([[1,0],[0,-1]])
mask
```

```
img_new = np.zeros([m, n])

img_new

for i in range(1, m-2):
  for j in range(1, n-2):
    temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[0,
    img_new[i, j]= temp
img_new = img_new.astype(np.uint8)
plt.imshow(img_new,cmap=plt.cm.gray)

img_new
```

Out[12]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  0,   3,   8, ...,   7,   0,   0],
       [  0,   7,   7, ...,   1,   0,   0],
       ...,
       [  0, 249, 248, ..., 236,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0]], dtype=uint8)
```

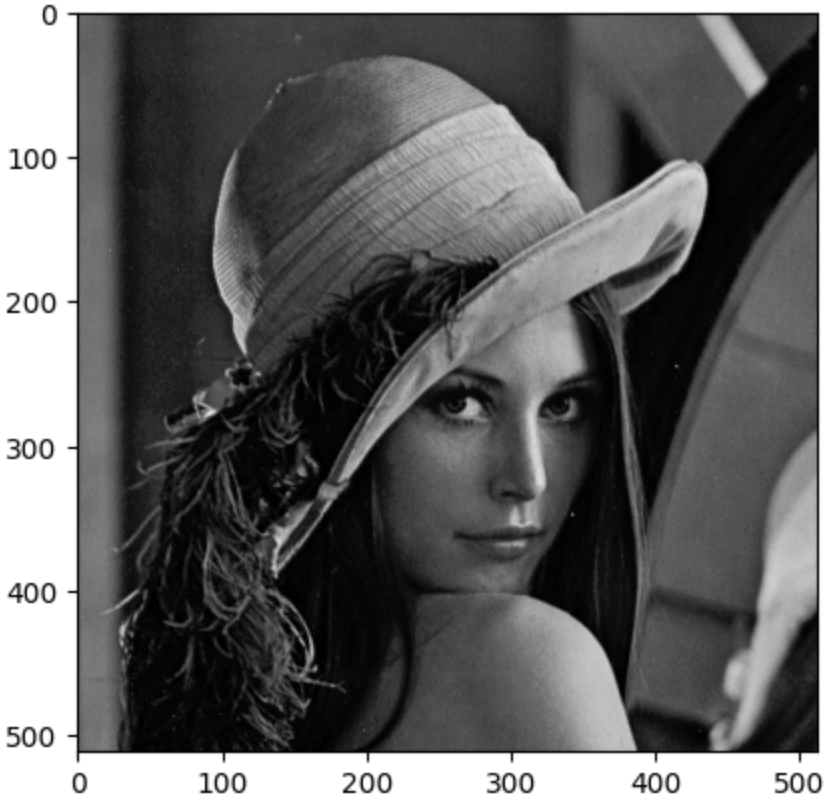


In [15]:
```
hb=0.9*img+0.5*img_new
hb
cv2_imshow(hb)
```

In [16]:
```
cv2_imshow(img3)
```

In [17]:
```
import cv2

# Load the image
image = cv2.imread("lena_gray.tif")
# Blur the image
gauss = cv2.GaussianBlur(image, (3,3), 0)
# Apply Unsharp masking
unsharp_image = cv2.addWeighted(image, 2, gauss, -1, 0)
plt.imshow(unsharp_image)
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x785b52db4850>
```


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