college-assignment (/github/sachi1406/college-assignment/tree/main)
/
IP_Expt_5_Sachi_Shah.ipynb (/github/sachi1406/college-assignment/tree/main/IP_Expt_5_Sachi_Shah.ipynb)

(https://colab.research.google.com/github/sachi1406/college-assignment/blob/main/IP_Expt_5_Sachi_Shah.ipynb)

IP Experiment No. 05

Image Enhancement using spatial domain filters (neighborhood processing)

- Name: Sachi Shah
- Roll No.: C094
- Batch: EB1
- Sap Id: 70321018081

Aim:

Apply a suitable spatial domain filtering technique to three images.
i. Blur the image.
ii. Sharpen the image.
iii. Remove salt and pepper noise

Neighborhood processing in spatial domain:

Here, to modify one pixel, we consider values of the immediate neighboring pixels also. For this purpose, 3X3, 5X5 or 7X7 neighborhood mask can be considered. Example of 3X3 mask is shown below.

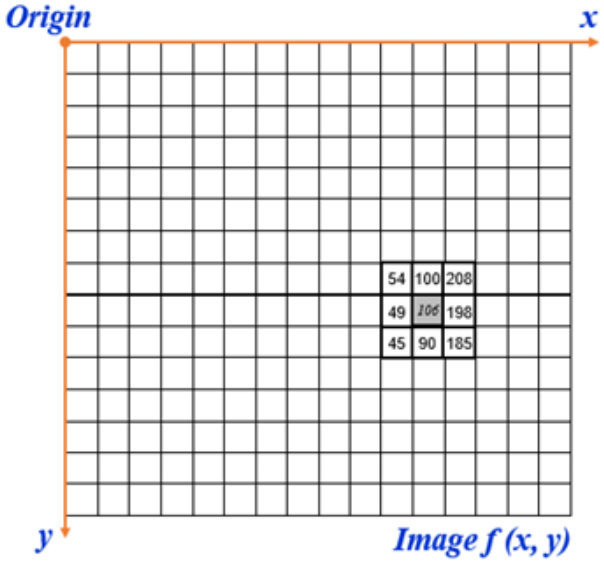
f(x-1,y-1) f(x-1,y) f(x-1,y+1)
f(x,y-1) f(x,y) f(x,y+1)
f(x+1,y-1) f(x+1,y) f(x+1,y+1)

Low Pass filtering for blurring

It is also known as smoothing filter. It removes the high frequency content from the image. It is also used to blurr an image. Low pass averaging filter mask is as shown.

1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9

the pixel which is to be filtered is processed with mask as shown in example below
The pixel with value 106 after applying mask is converted to 115.
The mask is shifted pixel by pixel and the process is repeated till all pixels are covered.



54	100	208
49	106	198
45	90	185

Original Image Pixels

* 1/9

1	1	1
1	1	1
1	1	1

Filter

3*3 Smoothing Filter

$$e = {}^{1}/_{9}[106 + 54 + 100 + 208 + 49 + 198 + 45 + 90 + 185] = 115$$

In [ ]:

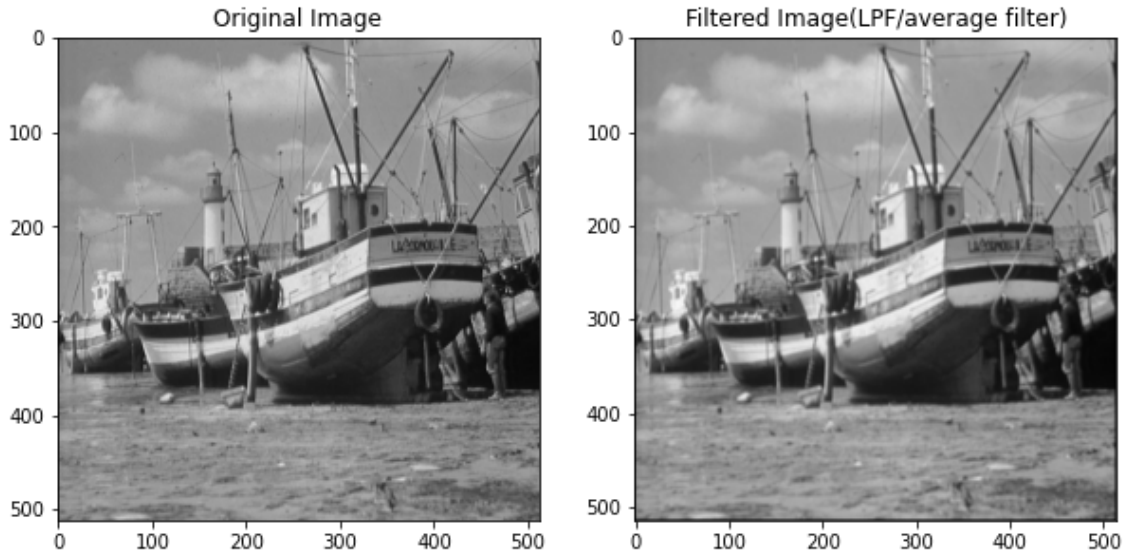
```
#import cv2, numpy, matplotlib
import cv2
from skimage import io
from skimage.color import rgb2gray
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
```

In [ ]:

```
img = cv2.imread('boat.png',0)
#create filter mask
mask = np.ones([3,3],dtype = int)
mask  = mask / 9
#aply convolution between image and mask
output_img = signal.convolve2d(img,mask)

#plot original and modified image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(LPF/average filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(LPF/average filter)')
```



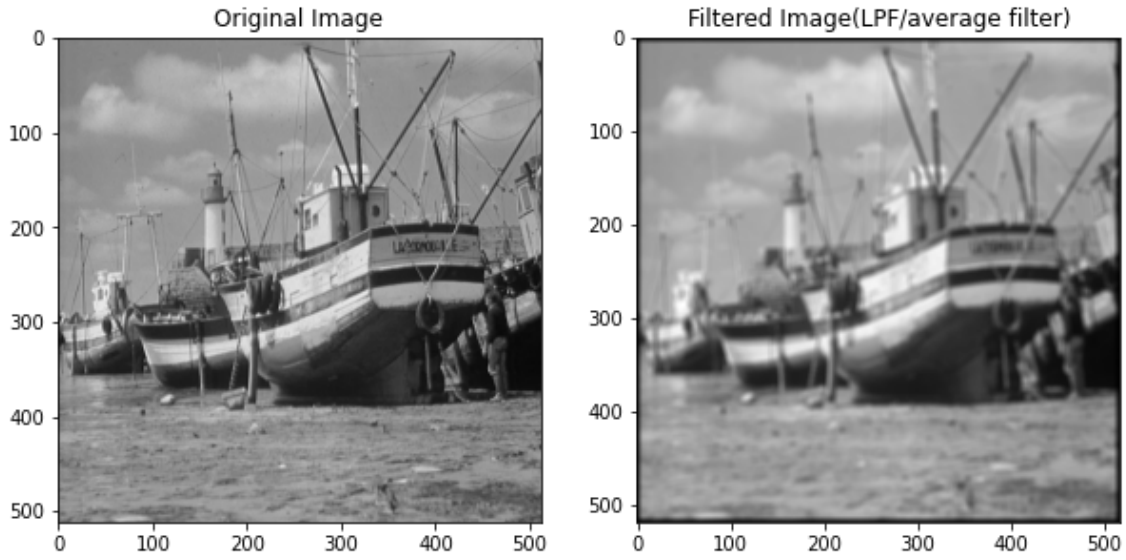
Increase in the mask size, increases the blurring

In [ ]:

```
img = cv2.imread('boat.png',0)
#create filter mask
mask = np.ones([7,7],dtype = int)
mask  = mask / 49
#aply convolution between image and mask
output_img = signal.convolve2d(img,mask)

#plot original and modified image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(LPF/average filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(LPF/average filter)')
```



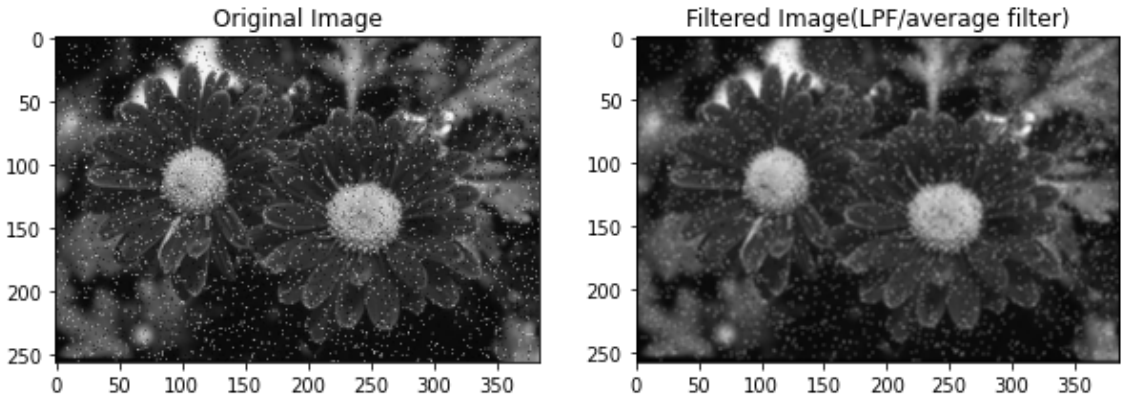
Average filter blures the salt and pepper noice as well as the main image. bigger mask can remove salt and pepper noise at the cost of increase in blurring of main image.

In [ ]:

```
img = cv2.imread('saltnpepper.png',0)
#create filter mask
mask = np.ones([3,3],dtype = int)
mask  = mask / 9
#aply convolution between image and mask
output_img = signal.convolve2d(img,mask)

#plot original and modified image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(LPF/average filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(LPF/average filter)')
```



High Pass Filtering in Spatial Domain- Laplacian Mask

It eliminates low frequency regions while retaining or enhancing the high frequency components (sharp edges). High pass filtering is done using Laplacian filtering mask as shown below :

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

The high pass filtering operation for a pixel is as illustrated below:

1	4	5	2	7
0	4	0	6	2
3	2	1	0	2
7	5	2	3	1
4	3	2	5	1

*

0	1	0
1	-4	1
0	1	0

=

1	4	5	2	7
0	0	0	6	2
3	2	1	0	2
7	5	2	0	1
4	3	2	5	1

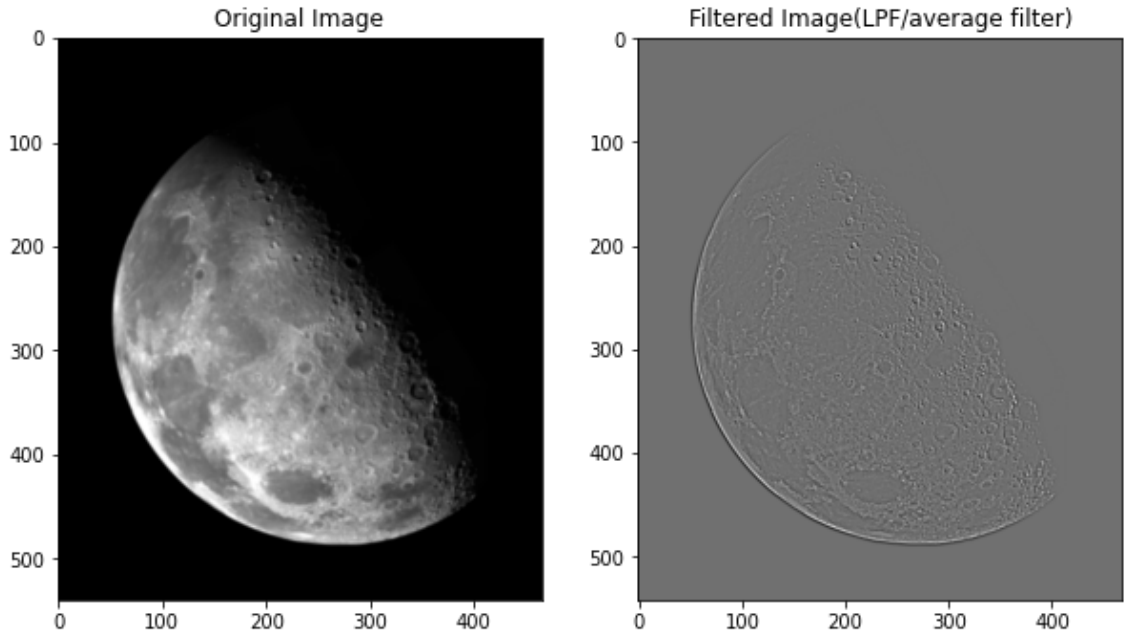
The high pass filtering process (convolution) of each pixel with the mask is carried out till all pixels are covered as shown below:

In [ ]:

```
# Read image 'Blurr_moon.tif'
img = cv2.imread('Blurr_moon.tif',0)

# create Laplacian mask
mask = [[0,-1,0],[-1,4,-1],[0,-1,0]]

# convolve image and mask
output_img = signal.convolve2d(img,mask)

# plot the original and resultant image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(HPF/Laplacian filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(LPF/average filter)')
```

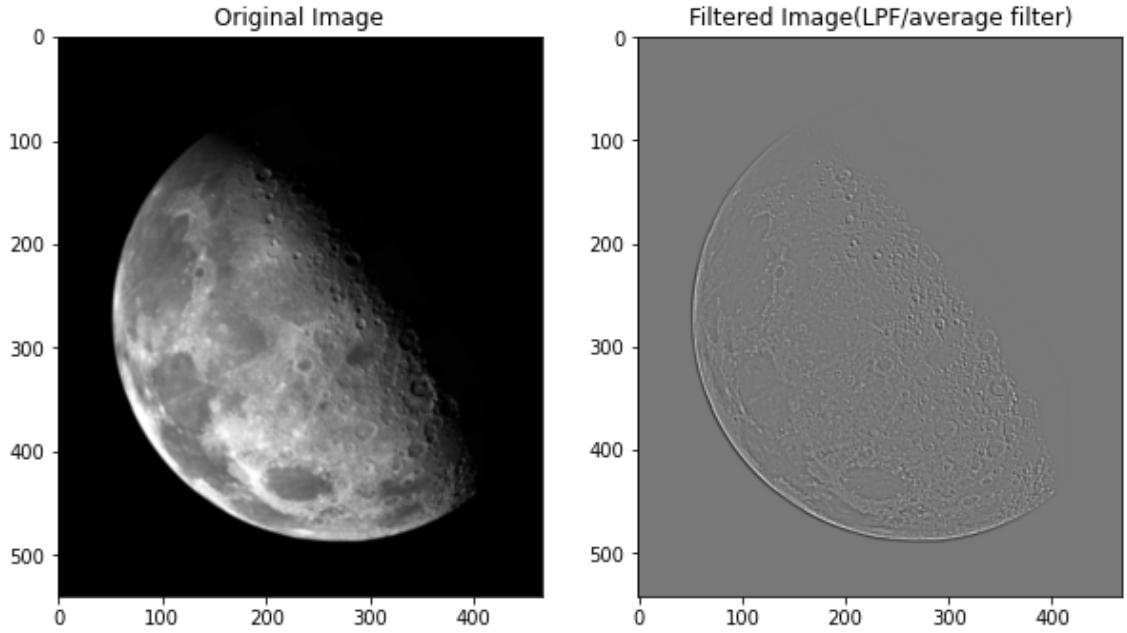


In [ ]:

```
# Read image 'Blurr_moon.tif'
img = cv2.imread('Blurr_moon.tif',0)

# create Laplacian mask
mask = [[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]

# convolve image and mask
output_img = signal.convolve2d(img,mask)

# plot the original and resultant image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(HPF/Laplacian filter)')
```

Out[ ]:

Text(0.5, 1.0, 'Filtered Image(LPF/average filter)')

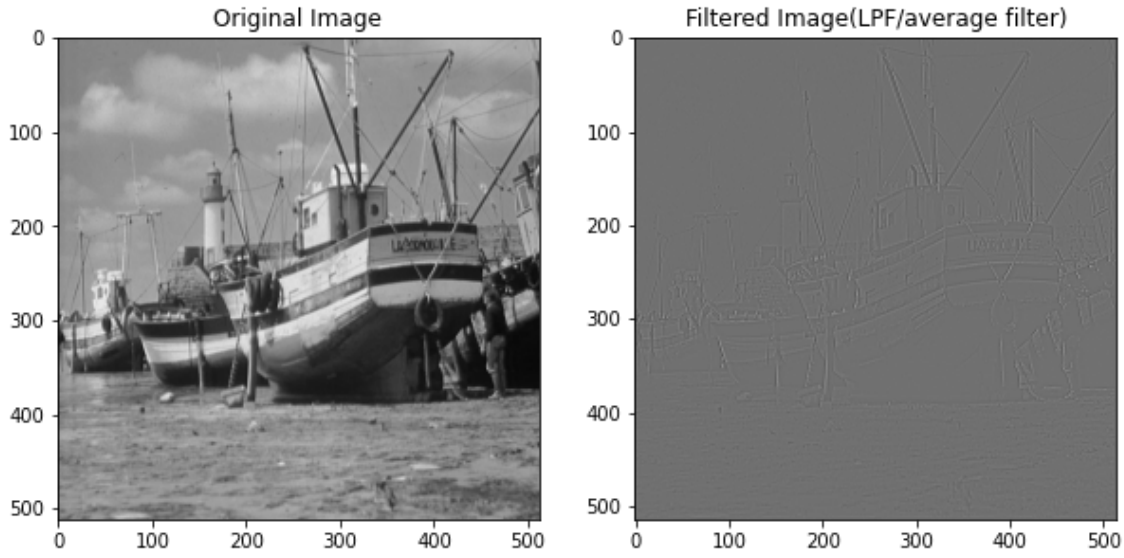


In [ ]:

```
# Read image 'Blurr_moon.tif'
img = cv2.imread('boat.png',0)

# create Laplacian mask
mask = [[0,-1,0],[-1,4,-1],[0,-1,0]]

# convolve image and mask
output_img = signal.convolve2d(img,mask)

# plot the original and resultant image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(HPF/Laplacian filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(LPF/average filter)')
```



The above result of appling HPF to the image provide sharp edges.
However we want the sharpened image for this we used enhanced laplacian mask

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -5 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The pixel by pixel processing of image using the enhanced lapacian mask is as showon below:

1	4	5	2	7
0	4	0	6	2
3	2	1	0	2
7	5	2	3	1
4	3	2	5	1

$*$

-1	-1	-1
-1	8	-1
-1	-1	-1

$=$

1	4	5	2	7
0	16	-24	29	2
3	-6	-14	-17	2
7	-16	-5	10	1
4	3	2	5	1

the above Laplacian mask provides the edges. However we want the sharpened image we have to use ENHANCED LAPLACIAN MASK AS SHOWN BELOW:

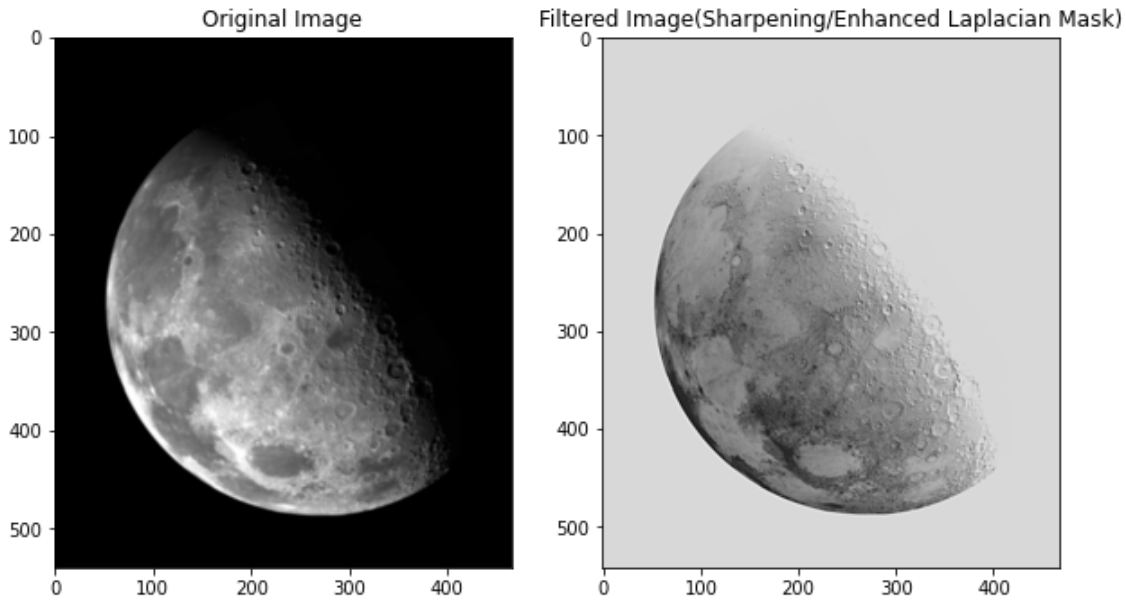
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -5 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

In [ ]:

```
# Read image 'Blurr_moon.tif'
img = cv2.imread('Blurr_moon.tif',0)

# create enhanced Laplacian mask
mask = [[0,1,0],[1,-5,1],[0,1,0]]

# convolve image and mask
output_img = signal.convolve2d(img,mask)

# plot the original and resultant image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(Sharpening/Enhanced Laplacian Mask)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(Sharpening/Enhanced Laplacian Mask)')
```

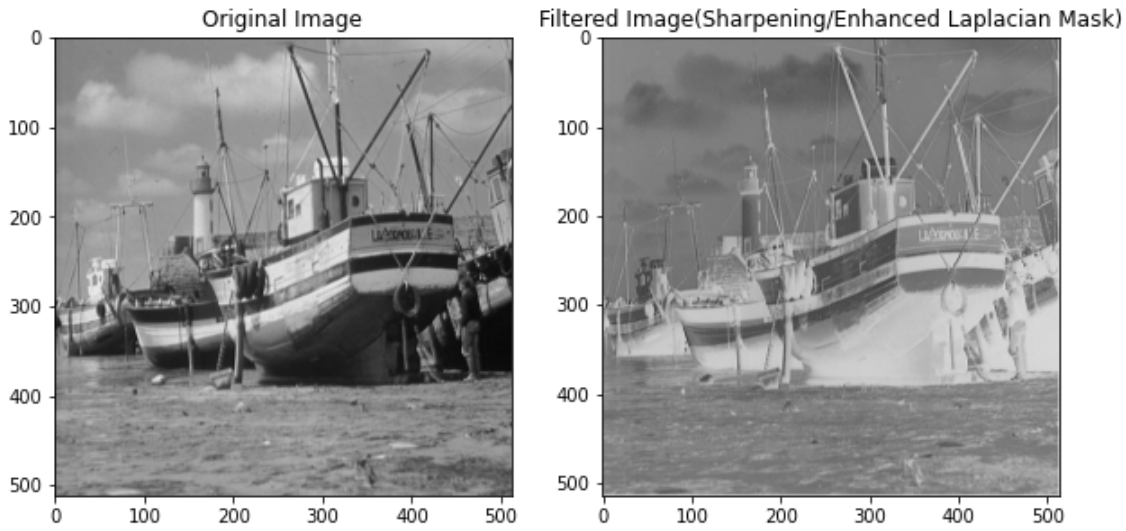


In [ ]:

```
# Read image 'Blurr_moon.tif'
img = cv2.imread('boat.png',0)

# create enhanced Laplacian mask
mask = [[0,1,0],[1,-9,1],[0,1,0]]
#use different variant of mask to sharpen
# convolve image and mask
output_img = signal.convolve2d(img,mask)

# plot the original and resultant image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output_img,cmap='gray')
plt.title('Filtered Image(Sharpening/Enhanced Laplacian Mask)')
```

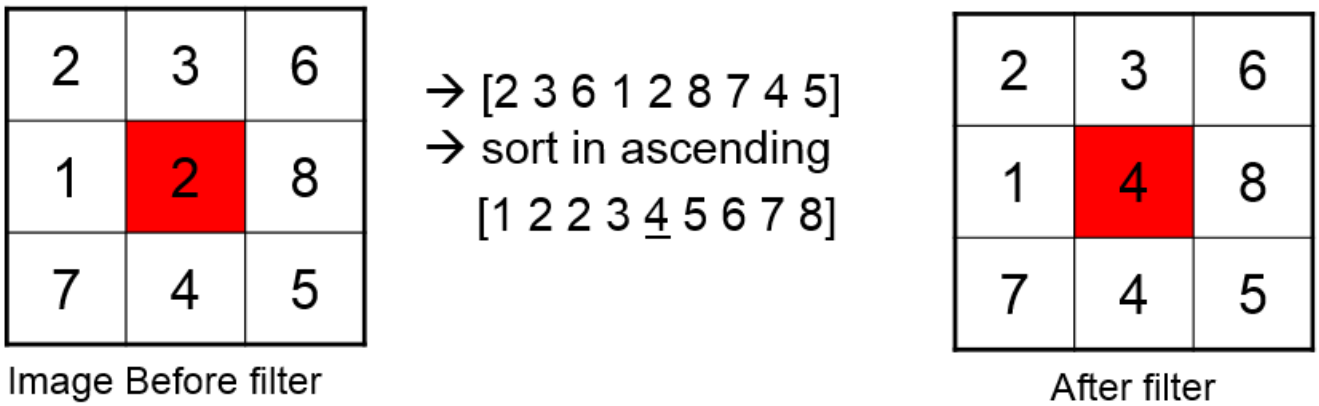
Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(Sharpening/Enhanced Laplacian Mask)')
```



Median Filtering for salt and pepper image

It is used to eliminate salt and pepper noise. Here the pixel value is replaced by median value of the neighbouring pixel.

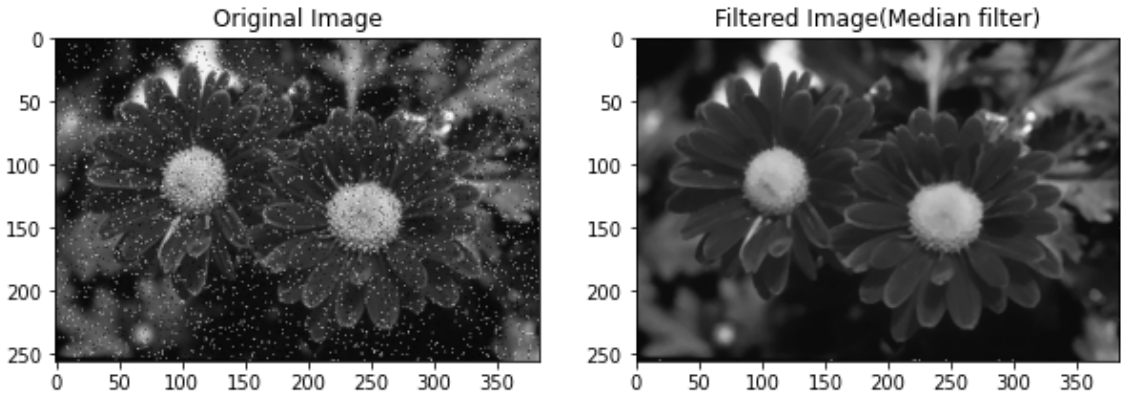


In [ ]:

```
# read the image
img = cv2.imread('saltnpepper.png',0)

# take the rows and columns of the image
r,c = img.shape

# create an array of zeros of same size as image
img_copy = img.copy()


#apply for loop, take the 8 neighbours of each pixel, sort the values and pick the mid
#loop starting from 1 cause edges are not considered
for i in range(1,r-1):
  for j in range(1,c-1):
    temp = [img[i-1][j-1],img[i-1][j],img[i-1][j+1],
            img[i][j-1],img[i][j],img[i][j+1],
            img[i+1][j-1],img[i+1][j],img[i+1][j+1]
            ]
    s = sorted(temp)
    img[i][j] = s[4]

# plot the original and filtered image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img_copy,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(img,cmap='gray')
plt.title('Filtered Image(Median filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(Median filter)')
```



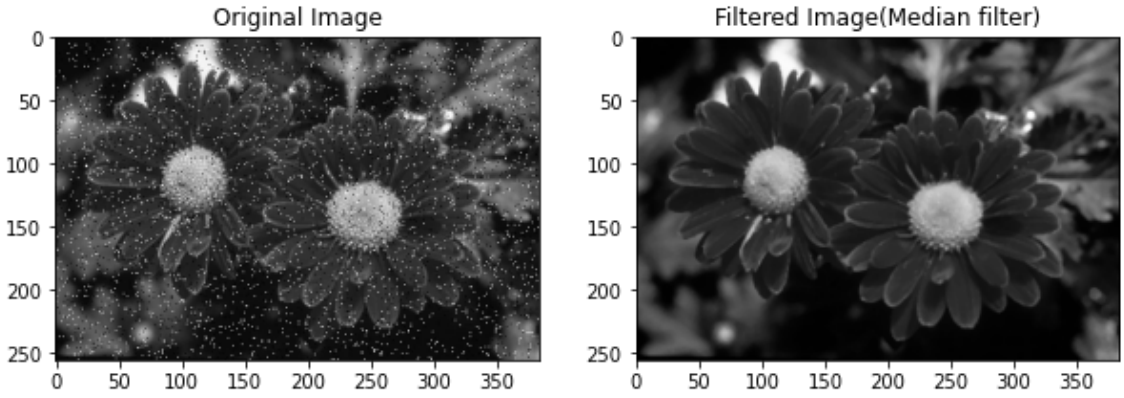
Using Python function to apply Median Filter

In [ ]:

```
# read the image
img = cv2.imread('saltnpepper.png',0)

# take the rows and columns of the image
r,c = img.shape

# create an array of zeros of same size as image
output = cv2.medianBlur(img,3)

# plot the original and filtered image
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(img_copy,cmap='gray')
plt.title('Original Image')
plt.subplot(122)
plt.imshow(output,cmap='gray')
plt.title('Filtered Image(Median filter)')
```

Out[ ]:

```
Text(0.5, 1.0, 'Filtered Image(Median filter)')
```



Conclusion :

1. We implemented the code for Low Pass Filtering using Average filter with a 3*3 mask and 7*7 mask*
2. It was observed that average filter creates a blurring effect
3. Average fliter is used for noise removal as it blurs the noise
4. A larger size mask results in more blurring, therefore for effective noise removal we should use a 3*3 mask.
5. The result of average filter was also tested on a salt and pepper noisy image.
6. Average filter is not a good choice for salt and pepper noise removal.
7. We also implemented the code for high pass filtering of an image using Laplacian mask and its variance.
8. Laplacian Mask provides edge detection.
9. High pass filtering (sharpened image) is obtained by using enhanced laplacian mask.

10. We implemented the code for median filtering by testing the code on salt and pepper noisy image. Median filter successlully removes the noise as compared to average filter as it selects the middle value and discards the lowest and highest value.