Indian Institute of Science Education and Research Bhopal

Computer Vision (DSE/EECS-312)

Assignment-1: Answer Sheet

Name: Jiya Sinha
Roll No.: 22161
Time of submission: Marks Obtained:

Please follow the instructions given in the assignment carefully.

Please provide your detailed answers and any explanations or diagrams directly below each question in the 'Answers' section.

1. Apply the filters mentioned below on the image attached and analyze their impact. Describe what you found after applying each filter and why certain phenomena are happening. ***(Marks:)***

1.

-1	0	1
-1	0	1
-1	0	1

2.

1	1	1
0	0	0
-1	-1	-1

Since the above filters are of dimension 3×3. Construct the same filters of dimension 5×5 and do the above experiments.

Answer:

- **Original Image:**

The original image before applying filters.
The matrix corresponding to the original image:

$$\begin{bmatrix} 162 & 162 & 162 & \cdots & 170 & 155 & 128 \\ 162 & 162 & 162 & \cdots & 170 & 155 & 128 \\ 162 & 162 & 162 & \cdots & 170 & 155 & 128 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 43 & 43 & 50 & \cdots & 104 & 100 & 98 \\ 44 & 44 & 55 & \cdots & 104 & 105 & 108 \\ 44 & 44 & 55 & \cdots & 104 & 105 & 108 \end{bmatrix}$$

- **After applying the first filter:**

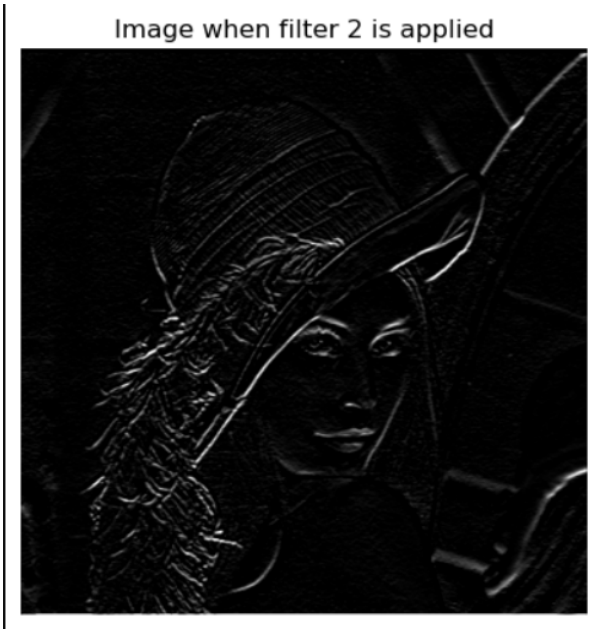
The image matrix after applying this filter becomes:

$$\begin{bmatrix} 324. & 0. & -2. & \cdots & -32. & -84. & -310. \\ 486. & 0. & -3. & \cdots & -48. & -126. & -465. \\ 486. & 0. & -3. & \cdots & -48. & -126. & -465. \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 133. & 22. & 15. & \cdots & 3. & -8. & -303. \\ 131. & 29. & 33. & \cdots & 10. & 2. & -310. \\ 88. & 22. & 29. & \cdots & 10. & 8. & -210. \end{bmatrix}$$

After applying the first filter, we observe the following effects:

- This is a Prewitts' filter for edge detection in horizontal-direction.
- This filter shows the change in the intensity or color along the horizontal direction.
- We can see the image the vertical edges are obtained by application the horizontal filter.

- **After applying the second filter:**



The matrix after applying this filter is:

$$\begin{bmatrix} -324. & -486. & -485. & \cdots & -496. & -453. & -283. \\ 0. & 0. & 0. & \cdots & 0. & 0. & 0. \\ 0. & 0. & 0. & \cdots & 0. & 0. & 0. \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 4. & -1. & -4. & \cdots & -9. & -21. & -19. \\ -2. & -7. & -25. & \cdots & -5. & -15. & -15. \\ 88. & 143. & 150. & \cdots & 309. & 317. & 213. \end{bmatrix}$$

After applying the second filter, we observe the following effects:

- This is a Prewitts' filter for edge detection in Vertical-direction.
- This filter shows the change in the intensity or color along the vertical direction.
- We can see in the image that the horizontal edges are obtained by application the vertical filter.

- **Constructed 5×5 Filters:**
  - In x-direction:

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

  - Result of applying this filter:

Image when filter 3 is applied

The matrix after applying this filter is:

$$\begin{bmatrix} 1458. & 966. & -3. & \cdots & -294. & -1152. & -1485. \\ 1944. & 1288. & -4. & \cdots & -392. & -1536. & -1980. \\ 2430. & 1610. & -5. & \cdots & -490. & -1920. & -2475. \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 747. & 549. & 102. & \cdots & 42. & -987. & -1516. \\ 597. & 461. & 111. & \cdots & 34. & -804. & -1236. \\ 451. & 357. & 95. & \cdots & 30. & -598. & -934. \end{bmatrix}$$

* This is a Prewitts' filter (5x5) for edge detection in Horizontal-direction using this we can see the change in the intensity or color values in wider range of pixels(Localization increased).

– In y-direction:

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

– Result of applying this filter:

The matrix after applying this filter is:

$$\begin{bmatrix} -1458. & -1941. & -2427. & \cdots & -2379. & -1872. & -1359. \\ -972. & -1294. & -1618. & \cdots & -1586. & -1248. & -906. \\ 0. & 0. & 0. & \cdots & 0. & 0. & 0. \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 25. & -14. & -18. & \cdots & -164. & -121. & -101. \\ 277. & 358. & 462. & \cdots & 955. & 777. & 577. \\ 415. & 560. & 722. & \cdots & 1517. & 1221. & 921. \end{bmatrix}$$

* This is a Prewitts' filter (5x5) for edge detection in Vertical-direction using this we can see the change in the intensity or color values in wider range of pixels(Localization increased).

2. Sobel Edge Detection Implementation. ***(Marks:)***

   (a) Write a Python function to apply the Sobel filter given below to detect edges along the x-direction and y-direction. Combine these to compute the gradient magnitude image.

1.

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

2.

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

3.

1	2	3	2	1
2	3	5	3	2
0	0	0	0	0
-2	-3	-5	-3	-2
-1	-2	-3	-2	-1

4.

-1	-2	0	2	1
-2	-3	0	3	2
-3	-5	0	5	3
-2	-3	0	3	2
-1	-2	0	2	1

(b) Apply your function to an image and display the gradient magnitude image alongside the original. Vary the size of the kernel and document the effects.

(c) Manually implement thresholding on the gradient magnitude to create a binary edge image. Experiment with different thresholds and show the results.

(d) Apply the Sobel edge detector on a noisy image (you may add synthetic noise to an attached clean image). Discuss how noise affects edge detection and the visual quality of the output images.

Answer:

(a)

To apply the filter on an given image, we need to do 2 steps.

1. Padding
2. Convolution

Given below are the 2 function written in python to do the same:

```python
def padding(img, filter_size):
    pad = int(((filter_size) - 1) // 2)   # Calculating the padding needed from the filter size

    (ih, iw) = (img.shape)    # Store the image size for the loop
    (nh, nw) = (ih + (pad * 2), iw + (pad * 2))  # Image size after padding

    p = np.full((nh, nw), 0, dtype=np.uint8)  # Result array to store 0
                                                values
    for i in range(0, ih):   # Loop for height
        for j in range(0, iw):  # Loop for width
            p[i + pad][j + pad] = img[i][j]  # Padded Image
    return p
```

Listing 1: Padding Function

```python
def conv(filter_matrix,img):
    filter_size = filter_matrix.shape[0]
    pad = ((filter_size)-1)//2  #Calculating the padding
    padded_img = padding(img,filter_size)  #performing the padding
    conv_img = np.zeros(img.shape)    #array of zeros of size of original image
    for i in range(pad,padded_img.shape[0]-pad):  #Loop for height
        for j in range(pad,padded_img.shape[1]-pad):  #Loop for width
            g = padded_img[i-pad:i+pad+1,j-pad:j+pad+1]  # Extracting the Region to be convoluted by filter
            h = np.sum(filter_matrix * g)  #Performing convolution
            conv_img[i-pad][j-pad] = h   #adding the pixel value to the result image
    return conv_img   #returning the result image.
```

Listing 2: Convolution Function

```
def grad_magnitude(x,y):
    #x is the image after applying horizontal filtering.
    #y is the image after applying vertical filtering.
    (h,w) = x.shape
    grad = np.zeros((h,w)) #array of zeros of shape of original
   image.
    for i in range (h):
        for j in range (w):
            grad[i][j] = sqrt(((x[i][j])**2) + ((y[i][j])**2))  #
   finding gradient magnitude.
    return grad
```

Listing 3: Function to find gradient magnitude

```
def show_image(array,title=None):
    im = Image.fromarray(array)
    plt.imshow(im,cmap="gray")
    plt.title(title)
    plt.axis("off")
    plt.show()
```

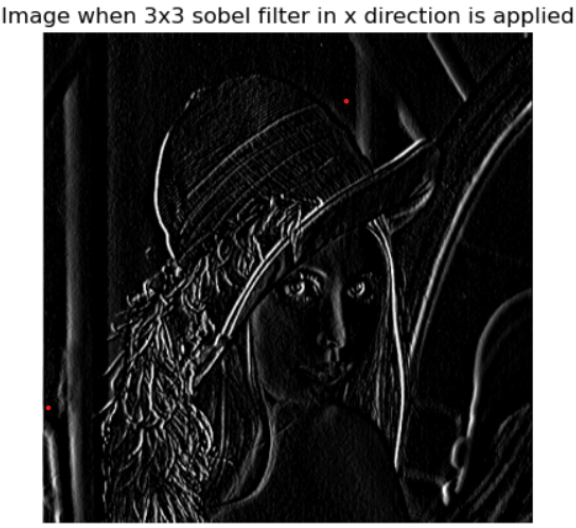
Listing 4: Function to show images

- (a)1.

```
#Defining the filter
sobel_filter_3x = np.asarray([
    [-1,0,1],
    [-2,0,2],
    [-1,0,1]
])
#Performing Convolution
sobel3x = conv(sobel_filter_3x,img_ndarray)
show_image(sobel3x,"Image when 3x3 sobel filter in x direction
   is applied")
```

Listing 5: Applying 3x3 Sobel x filter

Result of applying this filter:



- (a)2.

```
sobel_filter_3y = np.asarray([
    [1,2,1],
    [0,0,0],
    [-1,-2,-1]
])
sobel3y = conv(sobel_filter_3y,img_ndarray)
show_image(sobel3y,"Image when 3x3 sobel filter in y direction
   is applied")

```

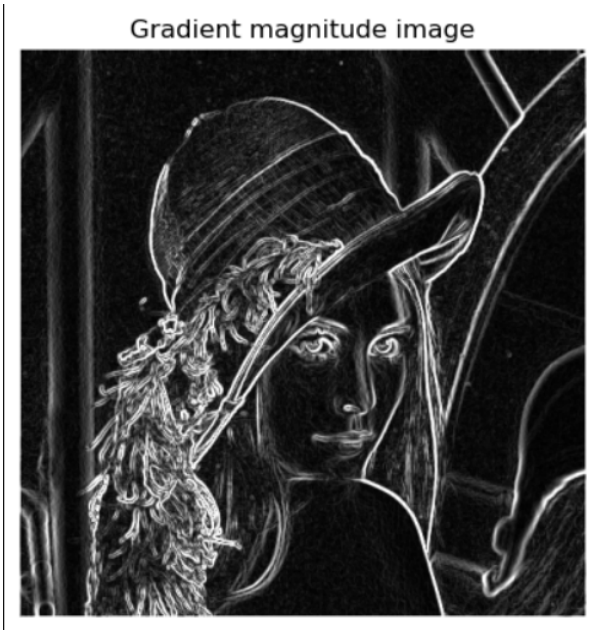
Listing 6: Applying 3x3 Sobel y filter

Result of applying this filter:

- (a)1,2 gradient magnitude:

```
grad3 = grad_magnitude(sobel3x, sobel3y)
show_image(grad3,"Gradient magnitude image of 3x3 Sobel filter"
   )
```

Listing 7: Gradient magnitude of 3x3 Sobel filter

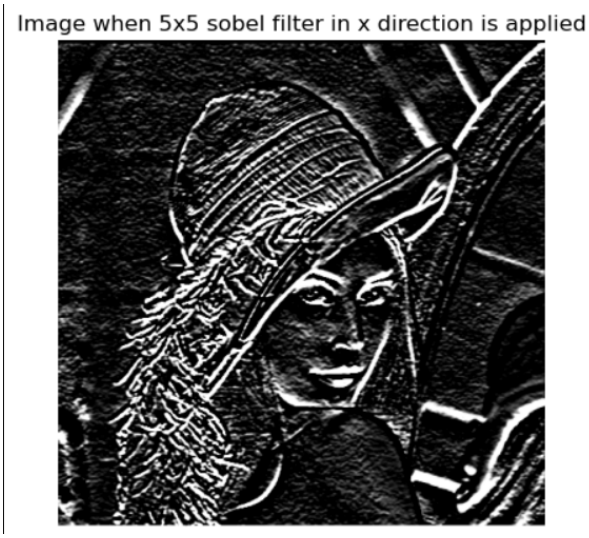


- (a)3.

```
#Defining the filter
sobel_filter_5x = np.asarray([
    [1,2,3,2,1],
    [2,3,5,3,2],
    [0,0,0,0,0],
    [-2,-3,-5,-3,-2],
    [-1,-2,-3,-2,-1]
])
#Performing Convolution
sobel5x = conv(sobel_filter_5x,img_ndarray)
show_image(sobel5x,"Image when 5x5 sobel filter in x direction
    is applied")
```

Listing 8: Applying 5x5 Sobel x filter

Result of applying this filter:

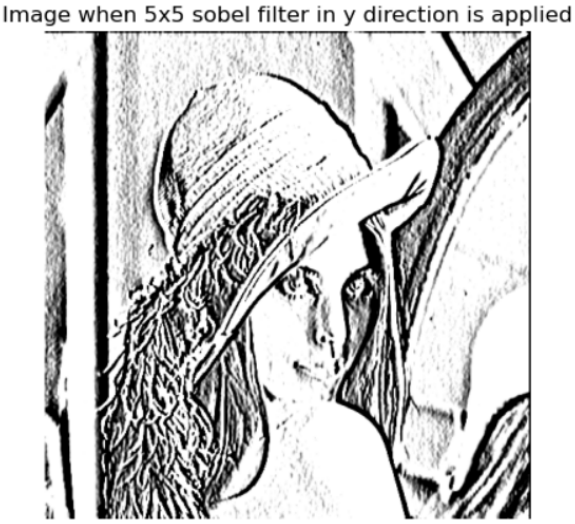


- (a)4.

```
sobel_filter_5y = np.asarray([
    [1,-2,0,2,1],
    [-2,-3,0,3,2],
    [-3,-5,0,5,3],
    [-2,-3,0,3,2],
    [-1,-2,0,2,1]
])
sobel5y = conv(sobel_filter_5y,img_ndarray)
show_image(sobel5y,"Image when 5x5 sobel filter in y direction
    is applied")

```

Listing 9: Applying 5x5 Sobel y filter

Result of applying this filter:



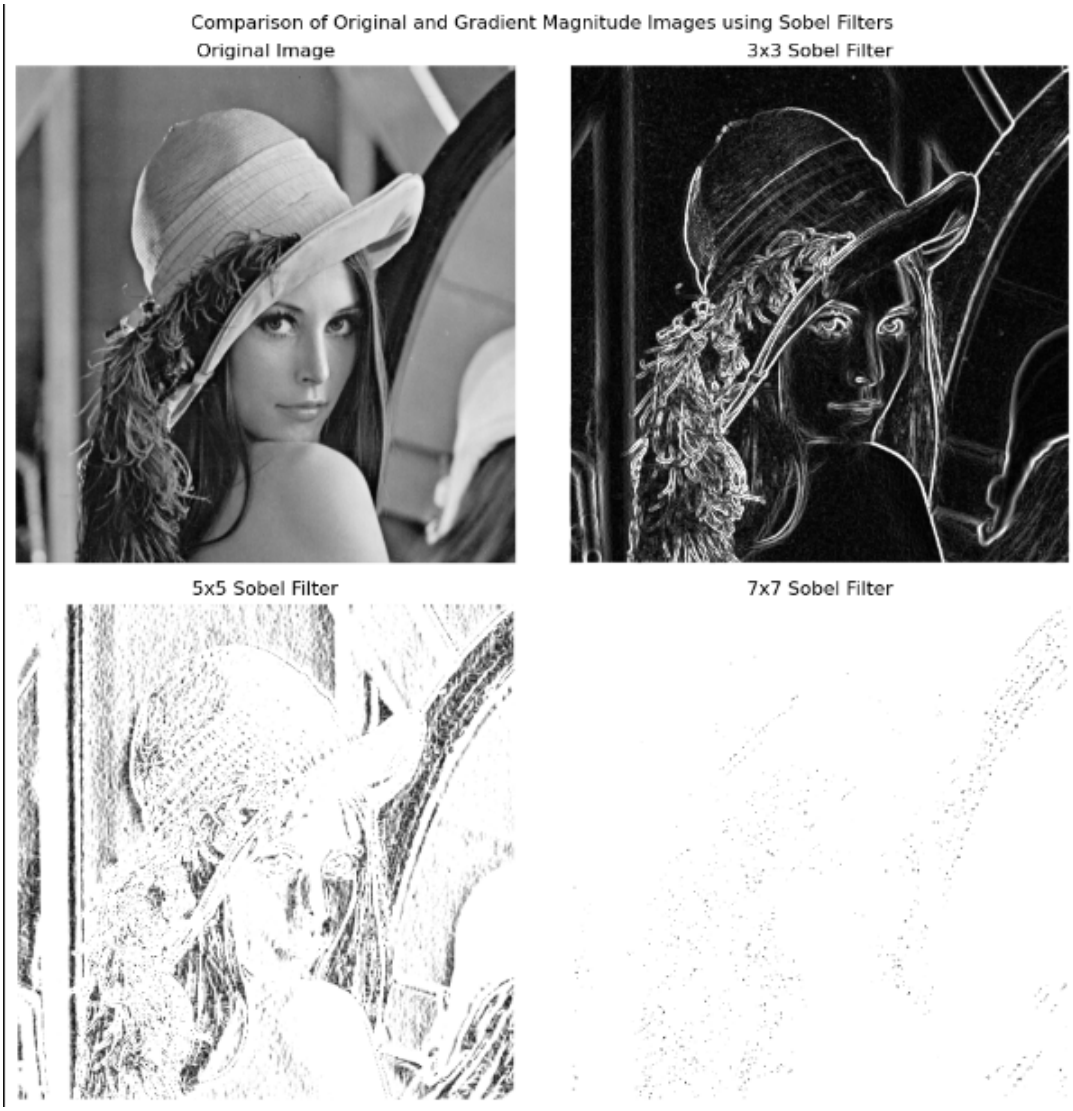
- (a)3,4 gradient magnitude:

```
grad5 = grad_magnitude(sobel5x, sobel5y)
show_image(grad5,"Gradient magnitude image of 5x5 Sobel filter"
    )
```

Listing 10: Gradient magnitude of 5x5 Sobel filter

(b)

- Comparison of gradient magnitude image with varying kernel size (3x3, 5x5, 7x7):



- Observations:
  - 3x3 filter has the best filter for this image for edge detection.
  - It is less robust to noise and edge detected are very clear.
  - The 5x5 filter is not very good at detection of noise as well as edge detection, The image contains a lot of noise and edges are not clear. It has very poor localization.
  - The 7x7 filter, being a very big kernel does not provide good localization and the edge detection is not good.

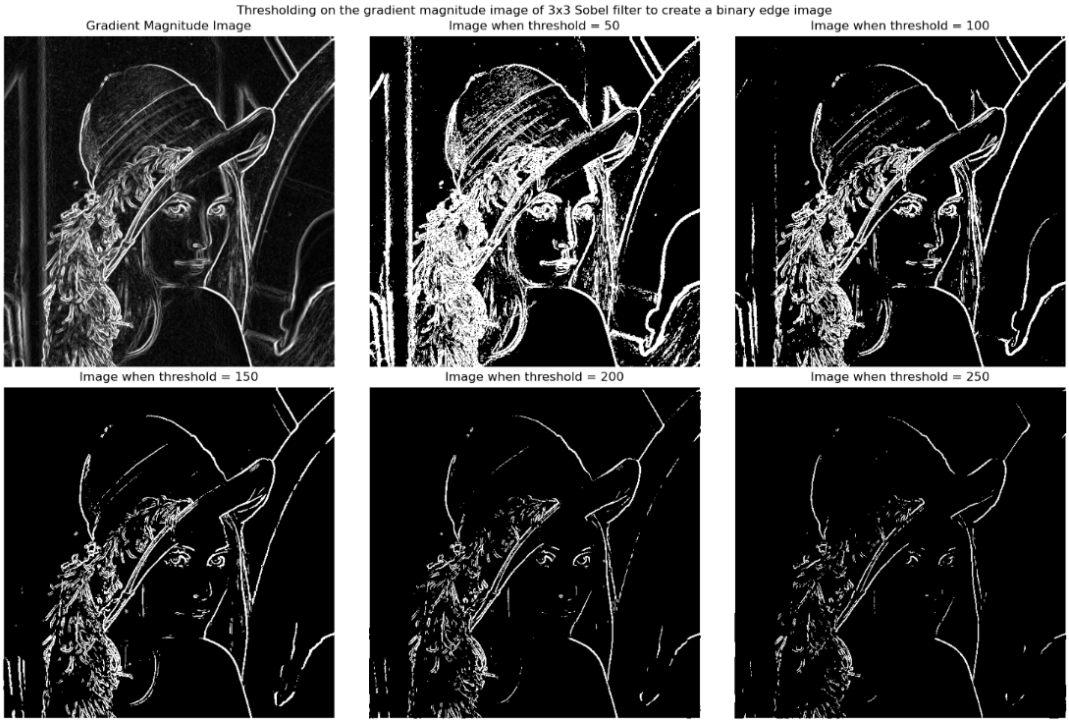
(c)

3. Performing thresholding on the gradient magnitude to create binary image:

```
def compare_threshold(grad,threshold,s):
    d = {}

    grad1= Image.fromarray(grad)
    fig, axes = plt.subplots(2,3, figsize=(15, 10))
    axes = axes.flatten()  # Flatten the 2D array of axes for easy
   indexing
    axes[0].imshow(grad1, cmap='gray')
    axes[0].set_title("Gradient Magnitude Image")
    axes[0].axis('off')
```

```
10
11      for j, i in enumerate(threshold, start=1):   #j is the index
        and i is the value. value of j starts from 1.
12          title = f"Image when threshold = {i}"
13          d[i] = np.where(grad > i, 255, 0)  #It sets values in the d
         array to 255 where grad > i and 0 where grad < i.
14          ax = axes[j]
15          ax.imshow(d[i], cmap='gray')
16          ax.set_title(title)
17          ax.axis('off')
18
19      plt.suptitle(f"Thresholding on the gradient magnitude image of
        {s}x{s} Sobel filter to create a binary edge image")
20      plt.tight_layout()
21      plt.show()
```

Listing 11: Code for performing threshold



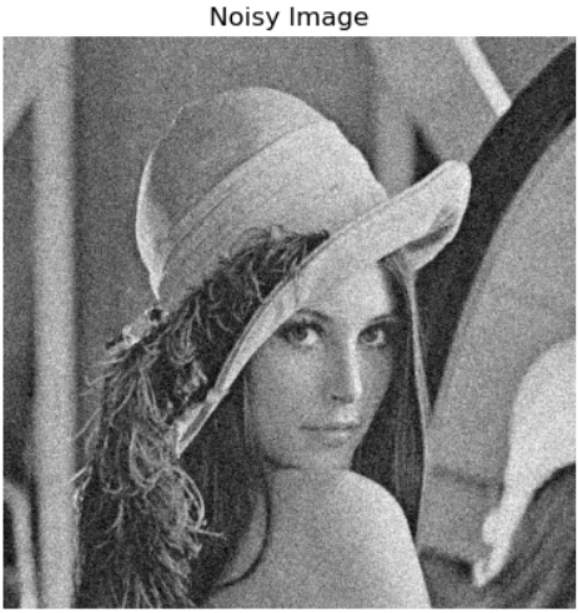
- We see that at higher threshold values, many edges gets removed whereas for lower threshold values, the image contains a lot of noise.

(d) Adding synthetic noise:

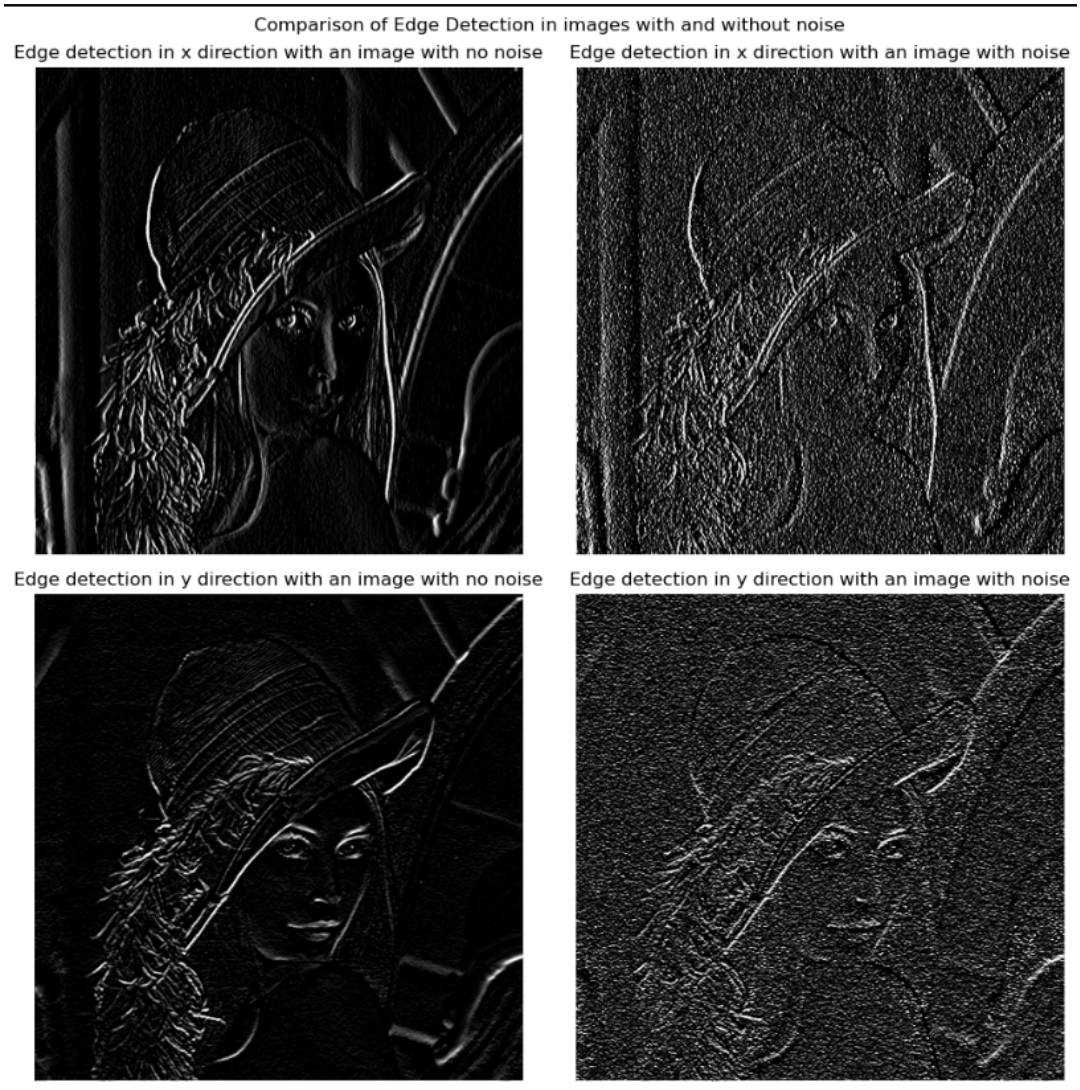
```
1  def add_gaussian_noise(img_ndarray,sigma,mean):
2
3      size = img_ndarray.shape
4      noise = np.random.normal(mean, sigma, size)  #Draw random
       samples from a normal (Gaussian) distribution.
5      noisy_img = img_ndarray + noise #add noise to the image
6      return noisy_img
```

Listing 12: Function to add Gaussian noise to an image

- Result of adding Gaussian Noise of mean=0, sigma = 25:



- Comparision of edge detection being applied on images with noise and images with no noise.



- We see that on application of noise, the edge detection is also considering it as

edge thus producing a noisy image.

4. Laplacian of Gaussian Edge Detection (follow the class notes). ***(Marks:)***

   (a) Implement Gaussian smoothing from scratch. Apply your Gaussian filter given below to smooth an image before edge detection.

   (b) Develop the Laplacian filter and apply it to the smoothed image from 3 (a) to detect edges via zero-crossings. Describe how you detect zero-crossings in your implementation.

   (c) Display the edges detected from the smoothed image alongside the edges detected from the non-smoothed image. Discuss the differences and the impact of noise.

Answer: (a)

- Gaussian Function:

```
def gaussian_fn (x, y, sigma):
#function to calculate the gaussian function
    sq = pow(sigma,2)
    n = (1/(2* (np.pi) * sq))
    num = (pow(x,2) + pow(y,2) )
    den = 2 * sq
    f = num/den
    e = exp(-f)
    return n*e
```

- Gaussian Filter:

```
def g_filter(size_of_filter, sigma):
    g_filter = np.zeros((size_of_filter, size_of_filter))  # Initialize a square filter with zeros
    for i in range(size_of_filter):
        for j in range(size_of_filter):
            g_filter[i, j] = gaussian_fn(i - size_of_filter // 2, j - size_of_filter // 2, sigma)  # (x,y) would be the distance from the center of the filter
    sum = np.sum(g_filter)  #for normalization
    return g_filter/(sum)

```

- Gaussian Smoothing

```
def gaussian_smoothing (size, sigma, img_ndarray):
    f = g_filter(size,sigma)    #getting the gaussian filter
    return conv(f,img_ndarray)  #performing convolution
```

- Output of the Gaussian Smoothing:

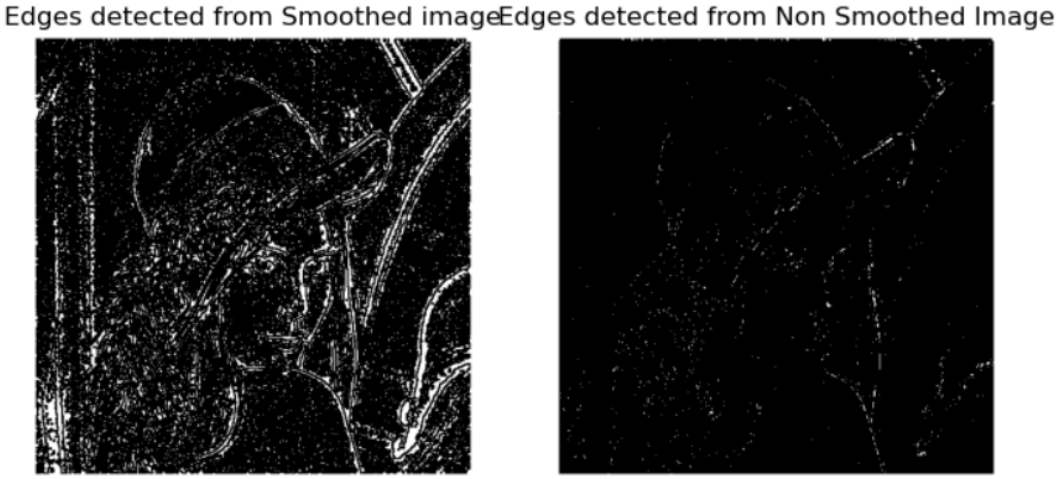
(b)

```
def zero_crossing(img):
    laplacian_kernel = np.array([[0, 1, 0],            #Laplacian
   Kernel
                                 [1, -4, 1],
                                 [0, 1, 0]])
    laplacian_image = conv(laplacian_kernel, img)      #Performing
   Convolution

    (ih,iw)=laplacian_image.shape
    zero = np.full((ih,iw),255)

    for i in range(1,ih-1):
        for j in range(1,iw-1):
            a = laplacian_image[i-1:i+2,j-1:j+2]       #Region of
   Interest
            mina = a.min()                             #max value
            maxa = a.max()                             #min value

            if mina<0 and maxa > 0:                    #if sign
   changes, it means it crossed zero.
                zero[i,j] = 0                          #if crossed
   zero, asign value 0 in the matrix

    return zero
```

- Applying Zero Crossing to a smooth image:

- Implementation:
  - Define a Laplacian kernel.
  - Apply the Laplacian kernel to the smoothed image.
  - Create an array named "zero" of the size = image size. This stores the values of the zero crossing for a particular pixel.
  - For each pixel location, check all the 9 closest value(that pixel included).
  - Store the minimum and maximum value from the above 9 values.
  - If both of these values are opposite signs, it would mean that the pixels had a zero crossing.
  - Update the value to 0 where the pixel crosses zero crossing, else 255 (already assigned).

(c)



- We can see that while detecting the edges using second derivative (Laplacian), the smooth image gives better edges where as edge detection in a non-smoothed images lead a lot of information loss compared to former.