National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

EE 433: Digital Image Processing

Name: Amal Saqib

CMS: 282496

Class: BSCS 9C

Lab 9: Spatial Filtering Basics-2

Date: 15th November 2021

Time: 2.00Pm to 5.00Pm

Instructor: Dr. Imran Malik

Task 1

In [1]:

```
from PIL import Image
import numpy as np
```

Open image

In [2]:

```
input_image = Image.open("DIP Lab 9/unsharpmasking.tif")
input_array = np.asarray(input_image)
```

Using smoothing filter of size 3

In [3]:

```
# returns the filter of size nxn
def generateFilter(filterSize):
    filter_array = [[1 for j in range(filterSize)] for i in range(filterSize)]
    return filter_array
```

In [4]:

```
filter_size = 3
filter_array = generateFilter(filter_size)
filter_array
```

Out[4]:

```
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

In [5]:

```
# calculates the pixel value by averaging
def averaging(inputArray, filterArray):
    pixel_value = np.sum(np.multiply(inputArray, filterArray) / np.sum(filterArray))
    return pixel_value
```

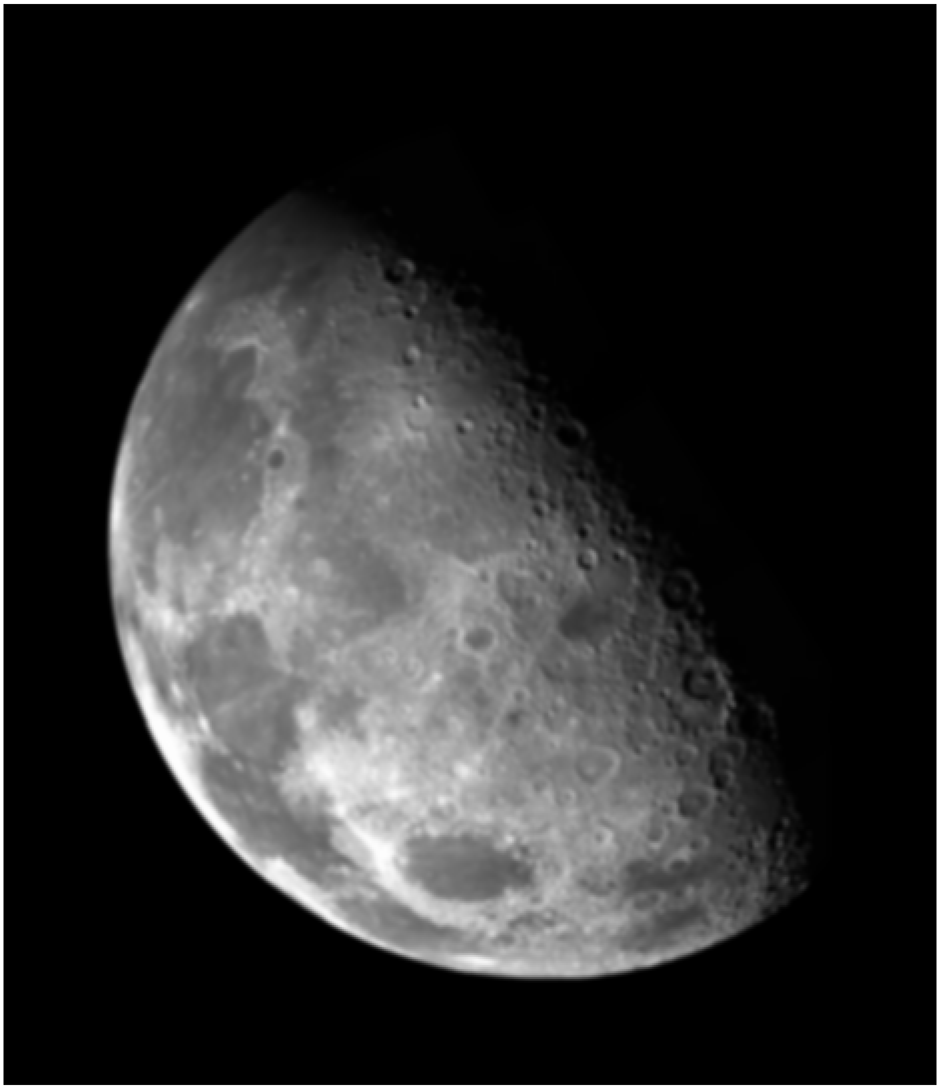
In [6]:

```
# applies the filter on the image
def applyingFilter(inputArray, filterArray, filterSize):
    # zero padding
    padding = int((filterSize - 1) / 2)
    inputArray = np.pad(inputArray, padding)

    height, width = inputArray.shape

    outputArray = inputArray.copy()

    # iterate the original image
    for x in range(padding, height - padding):
        for y in range(padding, width - padding):
            # gets the part of the image the size of the filter matrix
            neighbourhood_array = inputArray[x - padding:x + padding + 1, y - padding:
            outputArray[x][y] = averaging(neighbourhood_array, filterArray)

    # removes padding from output
    outputArray = outputArray[padding: -padding, padding: -padding]

    return outputArray
```

In [7]:

```
blurred_array = applyingFilter(input_array, filter_array, filter_size)
blurred_image = Image.fromarray(blurred_array)
blurred_image
```

Out[7]:



Applying Unsharpen Masking

In [8]:

```
def mask(inputArray, outputArray):
    gMask = inputArray - outputArray
    return gMask
```

In [9]:

```
gmask = mask(input_array, blurred_array)
```

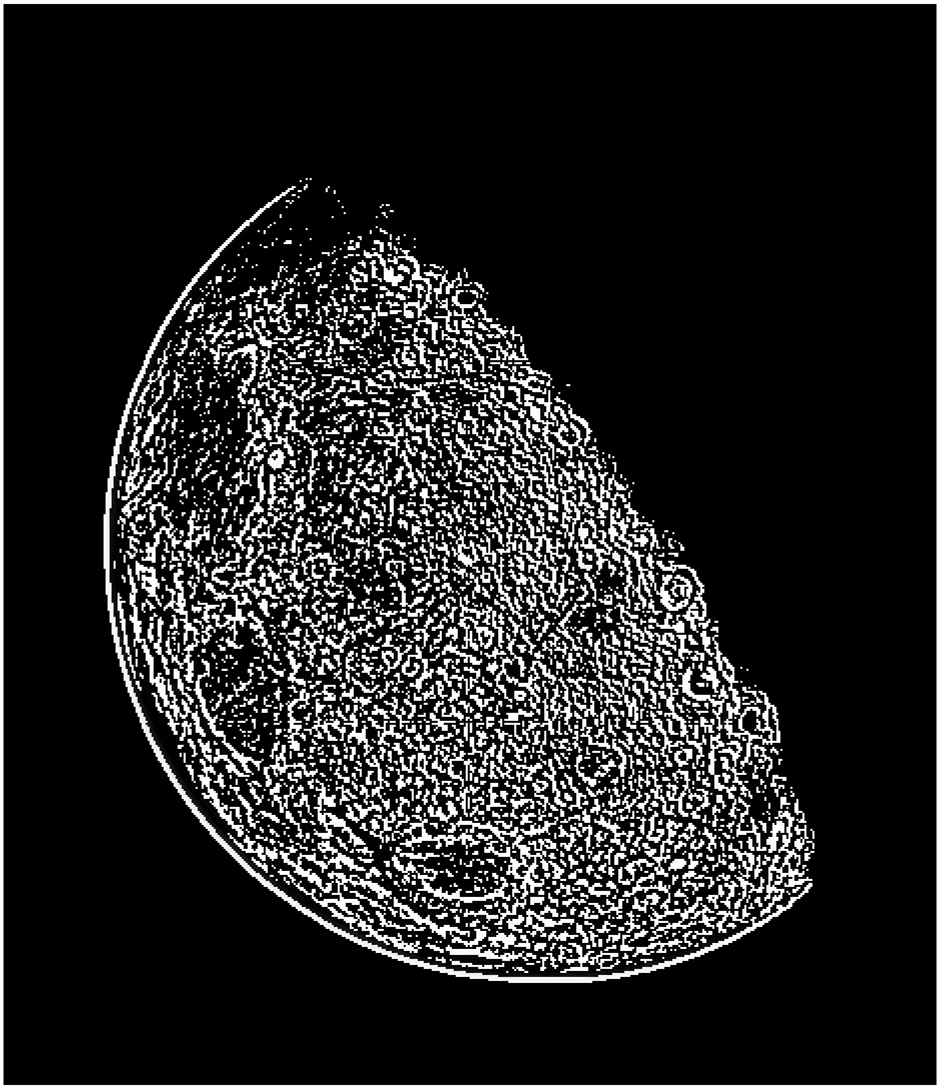
In [10]:

```
def unsharpening(inputArray, gMask, k):
    kTimesGmask = k * gMask
    unsharpArray = inputArray + kTimesGmask
    return unsharpArray
```

In [11]:

```
subtraction_image = Image.fromarray(gmask)
subtraction_image
```

Out[11]:

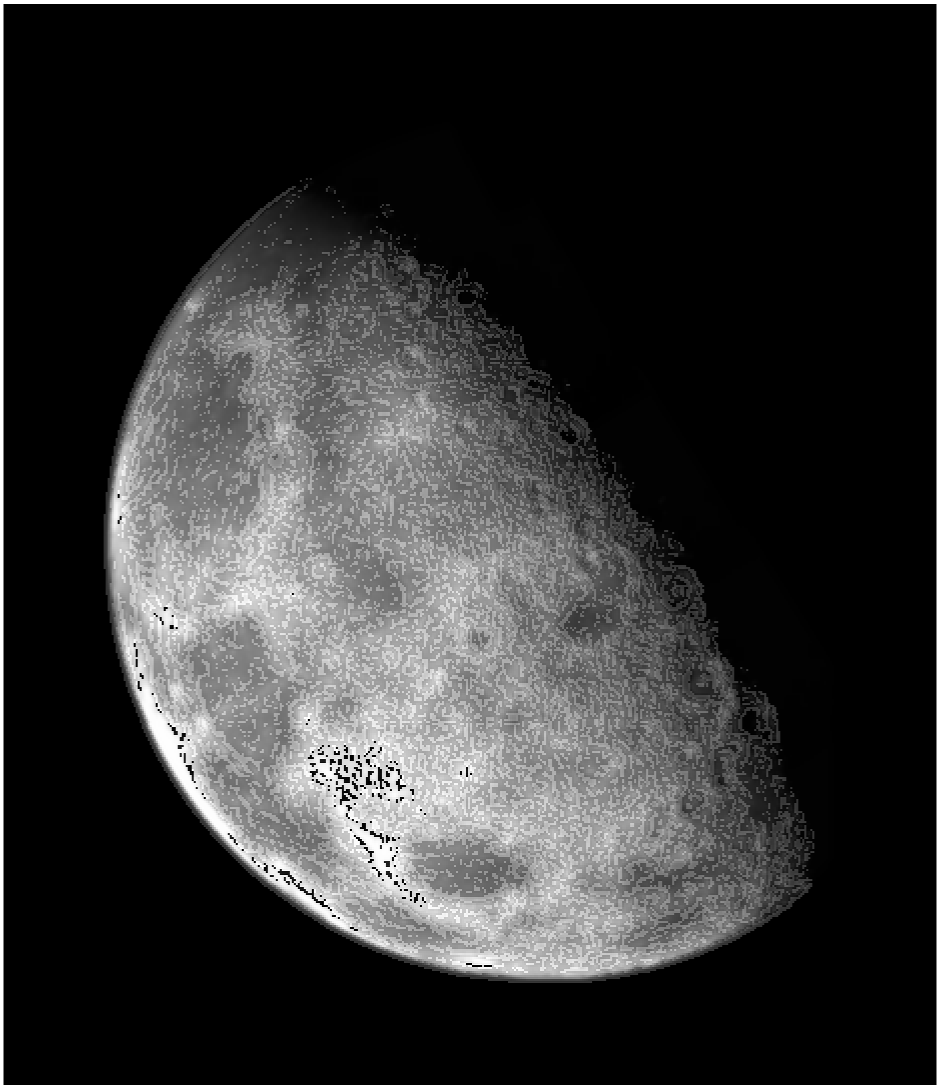


k = 0.2

In [12]:

```
k = 0.2
unsharp_output = unsharpening(input_array, gmask, k)
unsharp_image = Image.fromarray(unsharp_output.astype('uint8'))
unsharp_image
```

Out[12]:

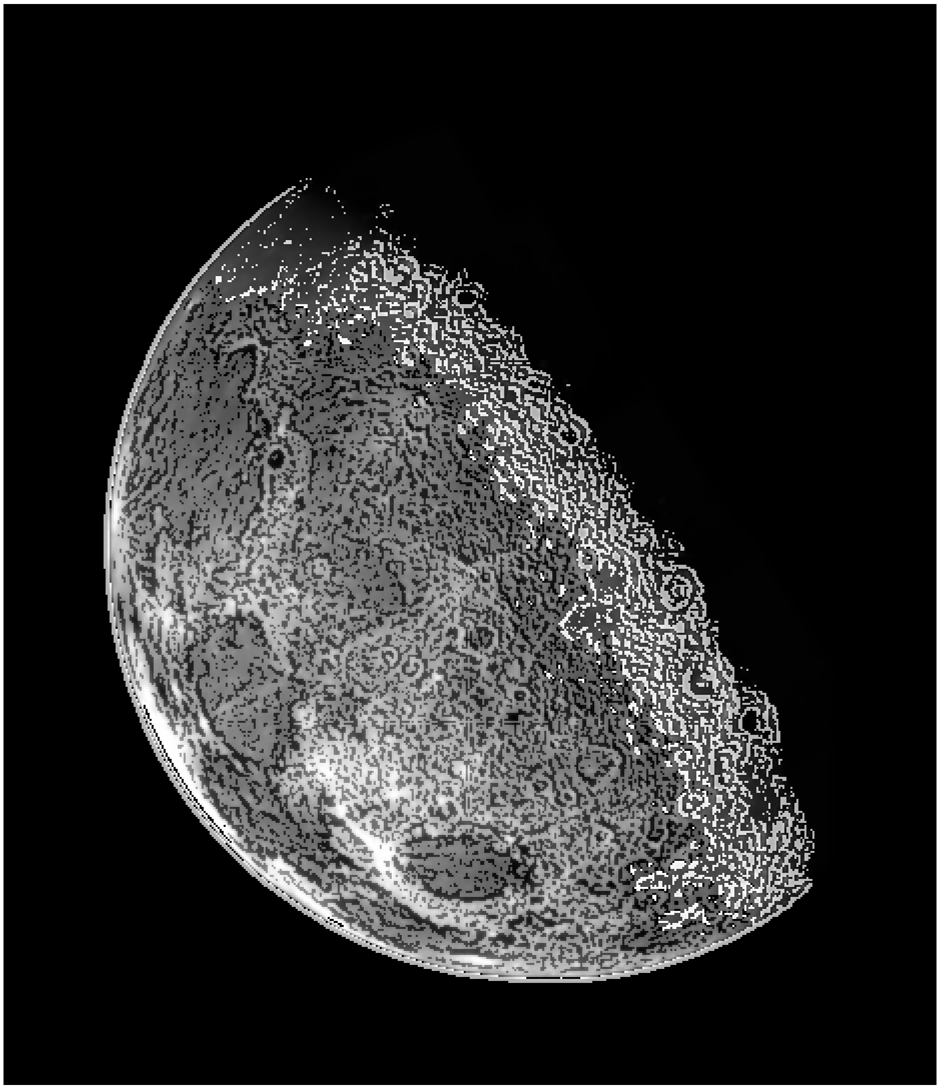


k = 0.7

In [13]:

```
k = 0.7
unsharp_output = unsharpening(input_array, gmask, k)
unsharp_image = Image.fromarray(unsharp_output.astype('uint8'))
unsharp_image
```

Out[13]:



Using smoothing filter of size 15

In [14]:

```
filter_size = 15
filter_array = generateFilter(filter_size)
filter_array
```

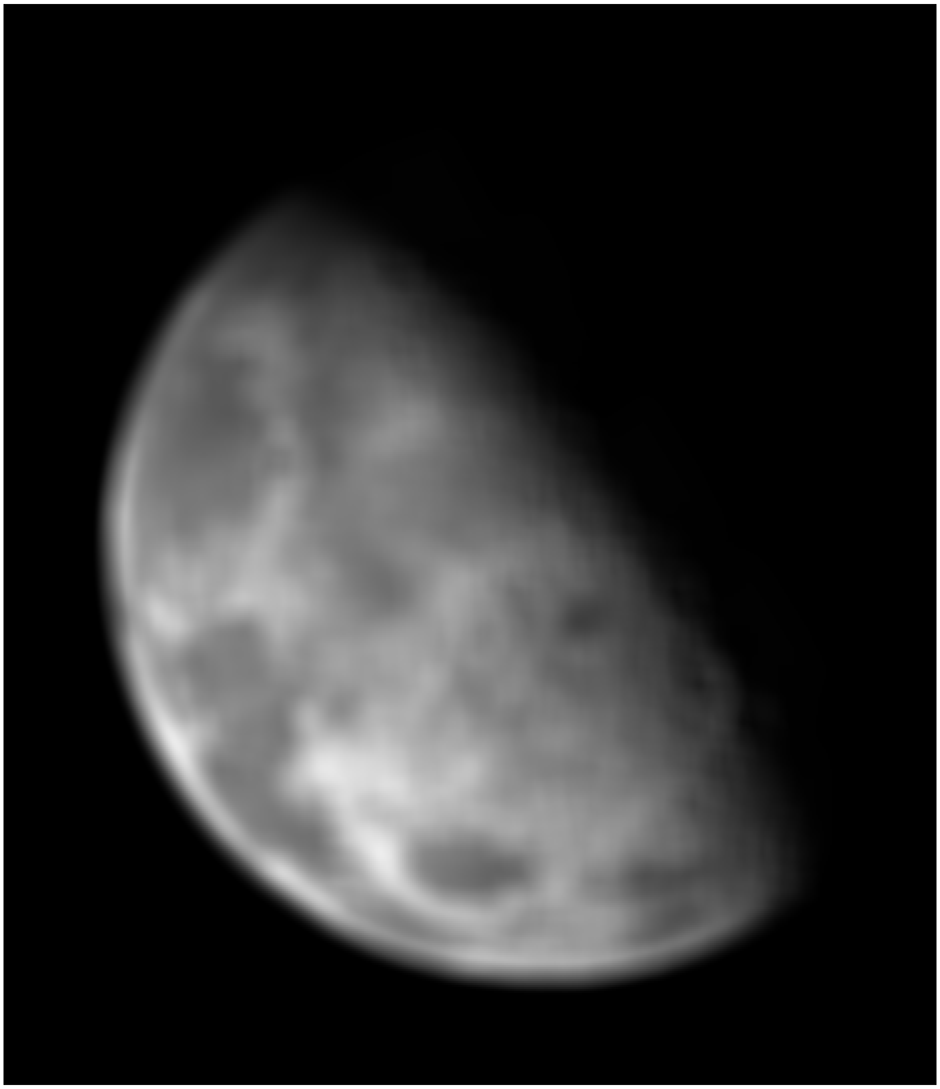
Out[14]:

```
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

In [15]:

```
blurred_array = applyingFilter(input_array, filter_array, filter_size)
blurred_image = Image.fromarray(blurred_array)
blurred_image
```

Out[15]:



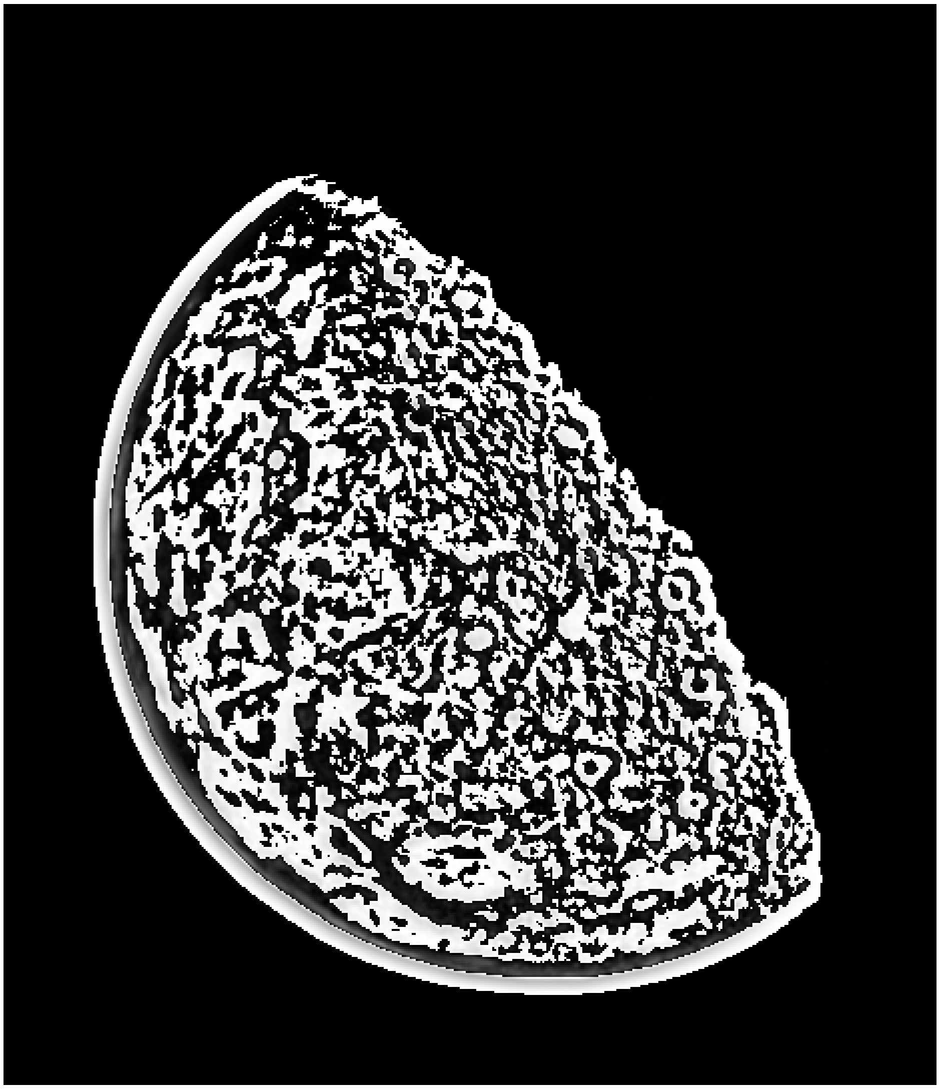
In [16]:

```
gmask = mask(input_array, blurred_array)
```

In [17]:

```
subtraction_image = Image.fromarray(gmask)
subtraction_image
```

Out[17]:

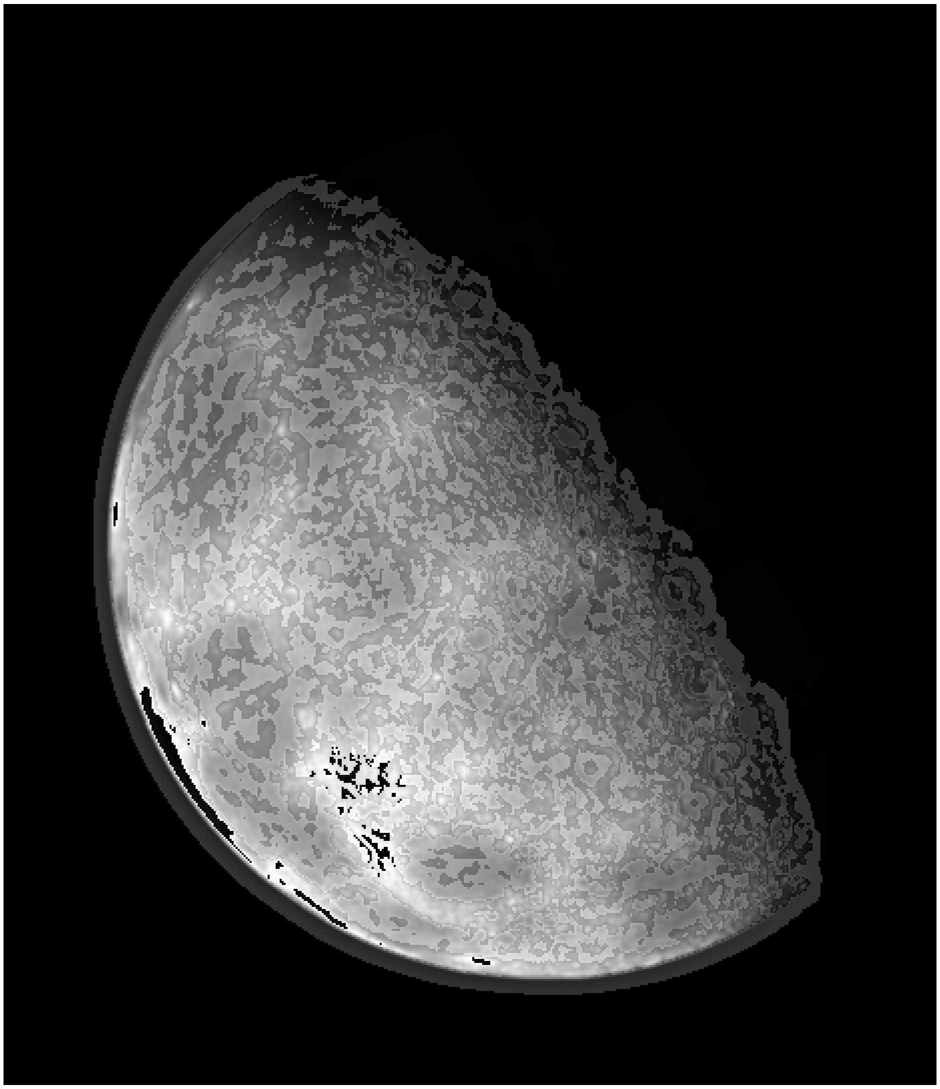


k = 0.2

In [18]:

```
k = 0.2
unsharp_output = unsharpening(input_array, gmask, k)
unsharp_image = Image.fromarray(unsharp_output.astype('uint8'))
unsharp_image
```

Out[18]:

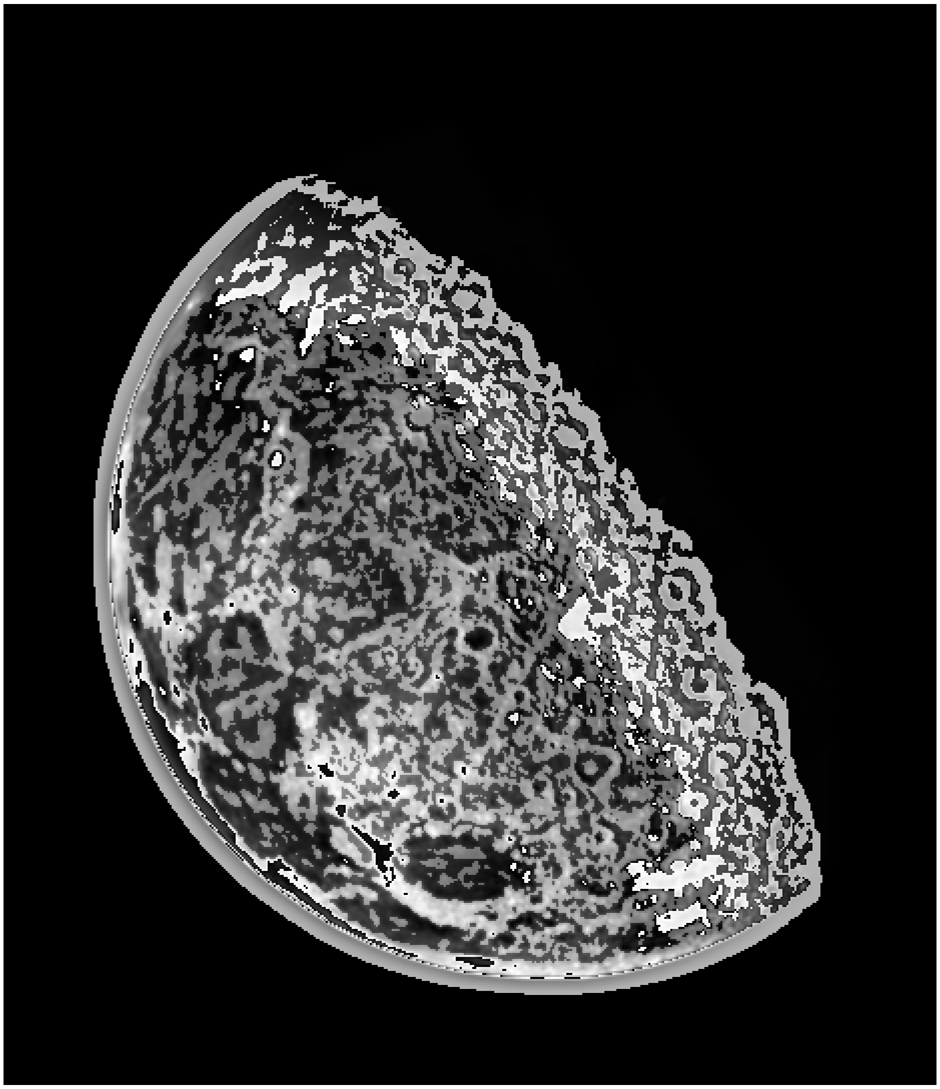


k = 0.7

In [19]:

```
k = 0.7
unsharp_output = unsharpening(input_array, gmask, k)
unsharp_image = Image.fromarray(unsharp_output.astype('uint8'))
unsharp_image
```

Out[19]:



As k increases, the edges become more defined, resulting in a more sharpened image

Task 2

In [20]:

```
def median(inputArray):
    array_1D = inputArray.flatten()
    array_1D.sort()
    if len(array_1D) % 2 != 0:
        mid_value = int((len(array_1D) + 1) / 2)
        pixel_value = array_1D[mid_value - 1]
    else:
        mid_value = int(len(array_1D) / 2)
        pixel_value = (array_1D[mid_value - 1] + array_1D[mid_value]) / 2
    return pixel_value
```

In [21]:

```
# applies the filter on the image
def applyingMedianFilter(inputArray, filterSize):
    # zero padding
    padding = int((filterSize - 1) / 2)
    inputArray = np.pad(inputArray, padding)
    height, width = inputArray.shape

    outputArray = inputArray.copy()

    # iterate the original image
    for x in range(padding, height - padding):
        for y in range(padding, width - padding):
            # gets the part of the image the size of the filter matrix
            neighbourhood_array = inputArray[x - padding:x + padding + 1, y - padding:
            outputArray[x][y] = median(neighbourhood_array)

    # removes padding from output
    outputArray = outputArray[padding: -padding, padding: -padding]

    return outputArray
```

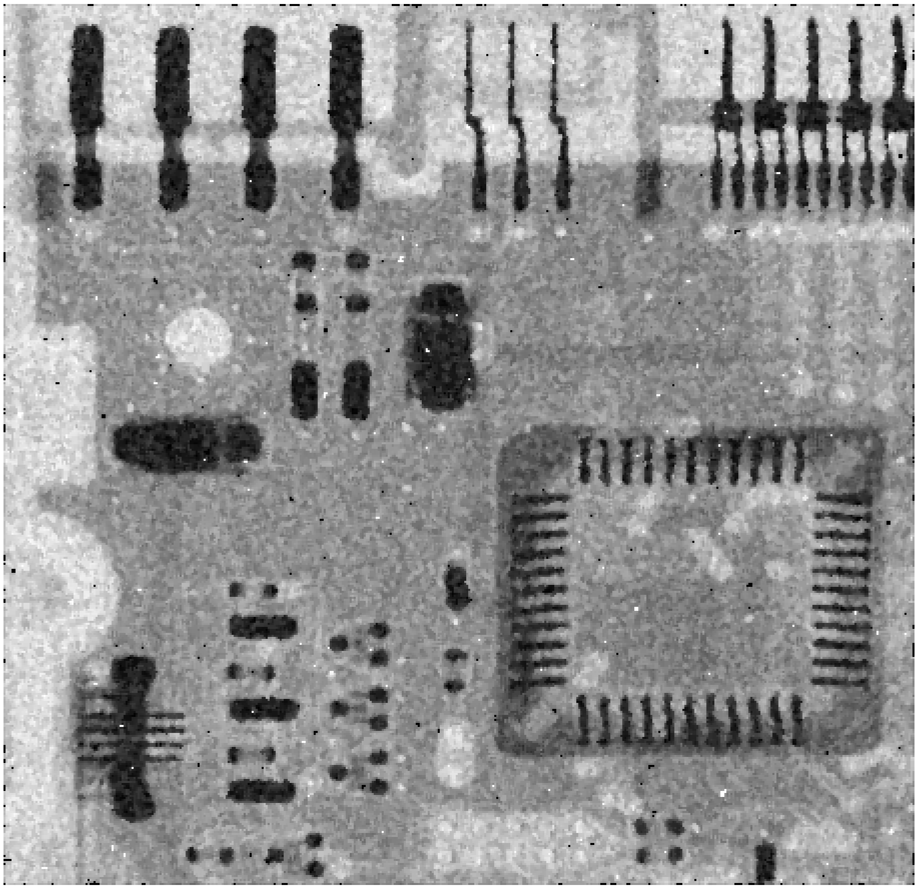
In [22]:

```
input_image = Image.open("DIP Lab 9/saltandpaper.tif")
input_array = np.asarray(input_image)
filter_size = 3
```

In [23]:

```
median_blurred_array = applyingMedianFilter(input_array, filter_size)
median_blurred_image = Image.fromarray(median_blurred_array)
median_blurred_image
```

Out[23]:



Task 3

In [24]:

```
def pixelValue(inputArray, filterArray, filterSize):
    values = []
    for i in range(3):
        for j in range(3):
            product = inputArray[i][j] * filterArray[i][j]
            values.append(product)
    return abs(sum(values))
```

In [25]:

```
def prewitt(inputArray, filterArray, filterSize):
    # zero padding
    padding = int((filterSize - 1) / 2)
    inputArray = np.pad(inputArray, padding)

    height, width = inputArray.shape

    outputArray = inputArray.copy()

    # iterate the original image
    for x in range(padding, height - padding):
        for y in range(padding, width - padding):
            # gets the part of the image the size of the filter matrix
            neighbourhood_array = inputArray[x - padding:x + padding + 1, y - padding:
            outputArray[x][y] = pixelValue(neighbourhood_array, filterArray, filterSize

    # removes padding from output
    outputArray = outputArray[padding: -padding, padding: -padding]

    return outputArray
```

In [26]:

```
input_image = Image.open("DIP Lab 9/two_cats.jpg").convert('L')
input_array = np.asarray(input_image)
input_array
```

Out[26]:

```
array([[ 96,  58,  86, ..., 223, 223, 223],
       [110,  57, 116, ..., 224, 223, 223],
       [123,  85, 111, ..., 224, 224, 224],
       ...,
       [ 61,  58,  55, ...,  80,  27,  77],
       [ 69,  61,  54, ...,  81,  69,  91],
       [ 49,  56,  62, ...,  84,  93,  78]], dtype=uint8)
```

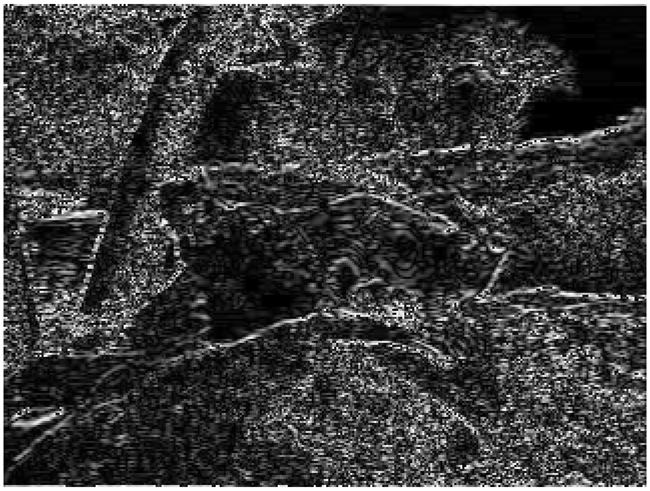
In [27]:

```
horizontal_filter = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
vertical_filter = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
```

In [28]:

```
horizontal_array = prewitt(input_array, horizontal_filter, 3)
horizontalEdges_image = Image.fromarray(horizontal_array)
horizontalEdges_image
```

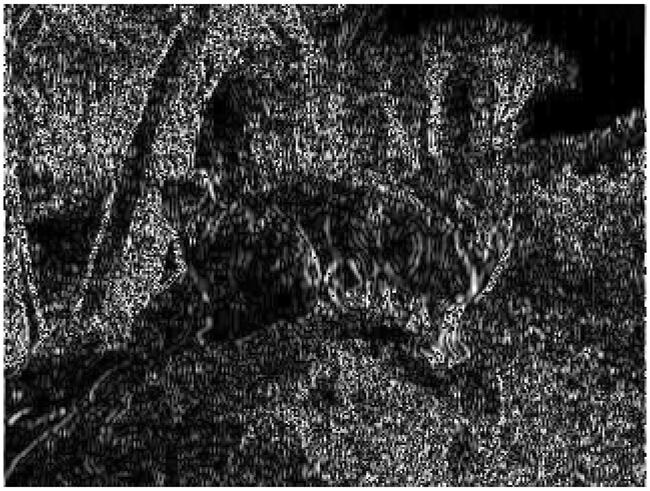
Out[28]:



In [29]:

```
vertical_array = prewitt(input_array, vertical_filter, 3)
verticalEdges_image = Image.fromarray(vertical_array)
verticalEdges_image
```

Out[29]:



In [30]:

```
final_array = np.add(horizontal_array, vertical_array)
final_image = Image.fromarray(final_array)
final_image
```

Out[30]: