Drishti Narwal
AI & ML (A2)
PRN No. 22070126037

Aim:

To explore and implement various image processing techniques in Python using libraries such as PIL, OpenCV, and skimage for tasks like image loading, manipulation, and analysis.

Objectives:

1. To understand the fundamental concepts of image processing. 2. To implement image loading and display techniques using PIL, OpenCV, and matplotlib. 3. To perform basic image transformations such as resizing, cropping, and filtering. 4. To convert images into numerical arrays using NumPy for further processing. 5. To compare the functionality of different image processing libraries and their use cases.

Theory:

Image processing is a technique used to enhance and analyze images through various computational methods. Some of the key libraries used in Python for image processing include:

1. PIL (Python Imaging Library):

Used for basic image manipulation tasks such as opening, resizing, cropping, and filtering. Provides an easy-to-use API for handling image formats like JPEG, PNG, and BMP.

2. OpenCV (Open Source Computer Vision Library):

A powerful library widely used for real-time computer vision applications. Supports advanced image processing techniques such as edge detection, feature extraction, and object tracking.

3. Skimage (Scikit-Image):

A collection of algorithms designed for image analysis. Provides functions for segmentation, filtering, and morphological operations.

4. NumPy:

Used to convert images into numerical arrays for mathematical manipulation. Facilitates operations like pixel-wise transformations and matrix operations on images. These libraries form the foundation for various applications such as medical imaging, object detection, and pattern recognition

Many ways to open images in Python.

1. PIL
2. matplotlib
3. skimage
4. openCV

Using PIL, Python Imaging Library

Pillow is an image manipulation and processing library

1. You can use pillow to crop, resize images and to do basic filtering.
2. For advanced tasks that require computer vision or machine learning we have other packages such as openCV, scikit image and scikit learn.
3. To install pillow, pip install Pillow
4. To import the package you need to use import PIL

In [ ]:
```
from PIL import Image
import numpy as np   #Use numpy to convert images to arrays

# Read image
img = Image.open("/content/water_body_1.jpg") #Not a numpy
print(type(img))

# Output Images
img.show()

# prints format of image
print(img.format)

# prints mode of image
print(img.mode)

#PIL is not by default numpy array but can convert PIL image to numpy array.
img1 = np.asarray(img)
print(type(img1))
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
JPEG
RGB
<class 'numpy.ndarray'>
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

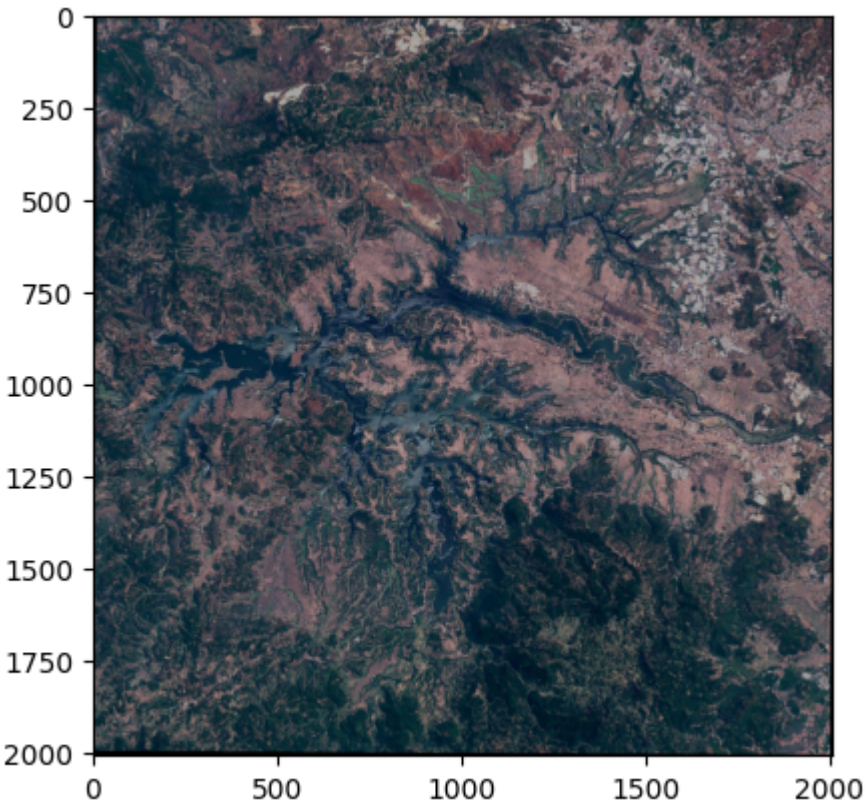
Using Matplotlib

1. Matplotlib is a plotting library for the Python programming language
2. Pyplot is a Matplotlib module which provides a MATLAB-like interface
3. Pyplot is commonly used not just to generate plots and graphs but also to visualize images because visualizing images is nothing but plotting data in 2D.
4. To install matplotlib, pip install matplotlib
5. To import the package you need to use import matplotlib

In [ ]:
```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img = mpimg.imread("/content/water_body_1.jpg")  #this is a numpy array
print(type(img))
print(img)

print(img.shape)

plt.imshow(img)
#plt.colorbar()   #Puts a color bar next to the image.
```

```
<class 'numpy.ndarray'>
[[[7 0 1]
  [7 0 1]
  [7 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[7 0 1]
  [5 0 1]
  [7 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[5 0 1]
  [4 0 1]
  [5 0 0]
  ...
  [0 1 0]
  [0 1 0]
  [0 1 0]]

 ...

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]]
(2009, 2007, 3)
```

Out[ ]: <matplotlib.image.AxesImage at 0x7802f47a6e30>



Using scikit image

1. To install matplotlib, pip install scikit-image
2. To import the package you need to use import skimage
3. scikit image is an image processing library that includes alforithms for
4. segmentation, geometric transformation, color space manipulation, analysis, filtering,
5. feature detection, and more.
6. A very good package for traditional machine learning, using Random forest or SVM

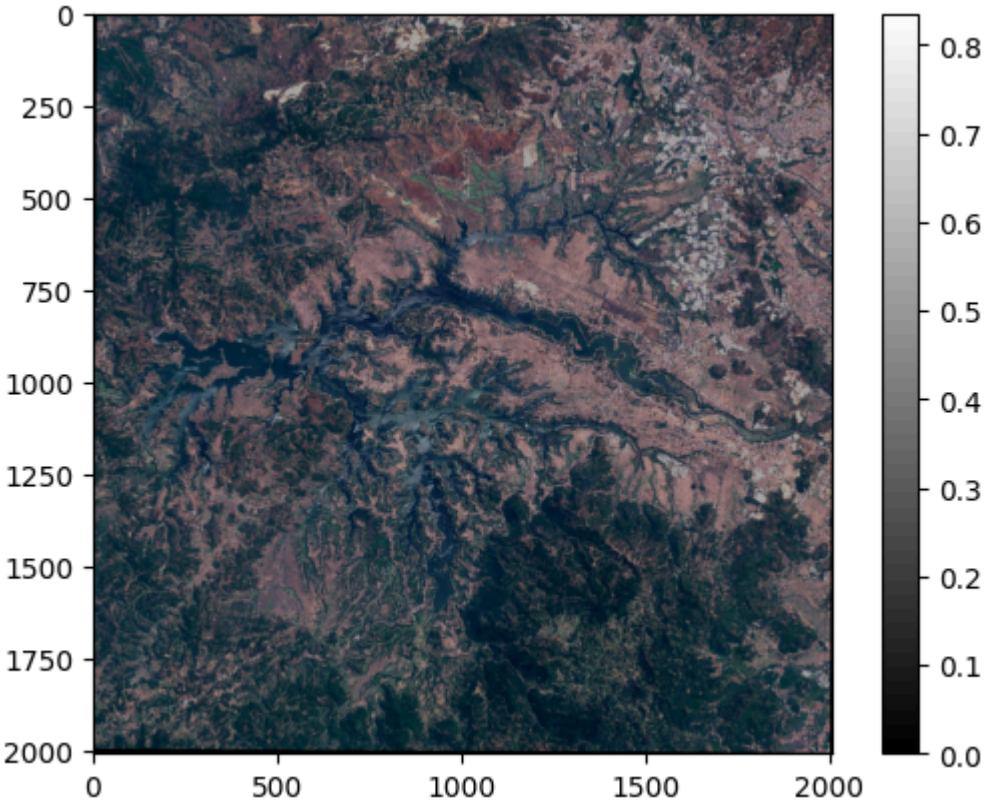
"""

In [14]:
```
from skimage import io, img_as_float, img_as_ubyte
import numpy as np
import matplotlib.pyplot as plt

image = img_as_float(io.imread("/content/water_body_1.jpg"))

#image2 = io.imread("images/test_image.jpg").astype(np.float)
#avoid using astype as it violates assumptions about dtype range.
#for example float should range from 0 to 1 (or -1 to 1) but if you use
#astype to convert to float, the values do not lie between 0 and 1.
#print(image.shape)
#plt.imshow(img)

print(image)

#print(image2)
#image8byte = img_as_ubyte(image)
#print(image8byte)
```

```
plt.imshow(image,'gray')
plt.colorbar()   #Puts a color bar next to the image.

#End of Skimage
```

```
[[[0.02745098 0.         0.00392157]
  [0.02745098 0.         0.00392157]
  [0.02745098 0.         0.        ]
  ...
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.02745098 0.         0.00392157]
  [0.01960784 0.         0.00392157]
  [0.02745098 0.         0.        ]
  ...
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.01960784 0.         0.00392157]
  [0.01568627 0.         0.00392157]
  [0.01960784 0.         0.        ]
  ...
  [0.         0.00392157 0.        ]
  [0.         0.00392157 0.        ]
  [0.         0.00392157 0.        ]]

 ...

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  ...
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  ...
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  ...
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]]
```

Out[14]: <matplotlib.colorbar.Colorbar at 0x7802c771fcd0>



Using openCV

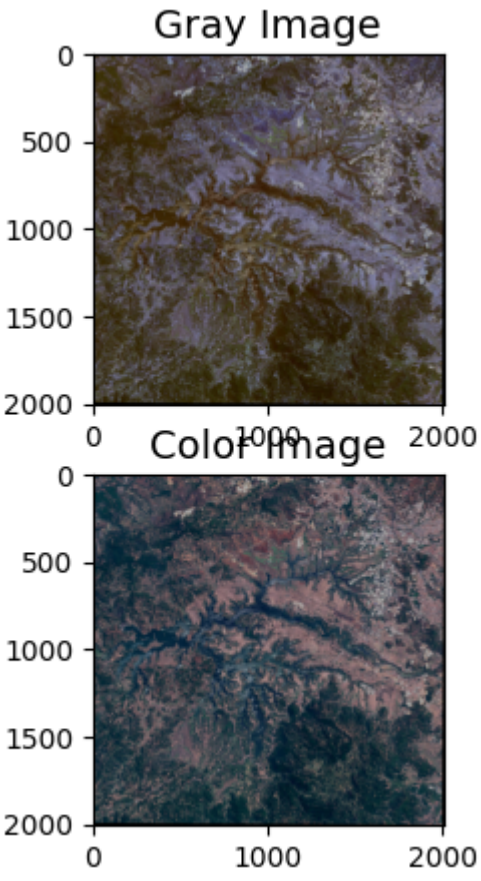
1. to install open CV : pip install opencv-python
2. to import the package you need to use import cv2
3. openCV is a library of programming functions mainly aimed at computer vision.
4. Very good for images and videos, especially real time videos.
5. It is used extensively for facial recognition, object recognition, motion tracking,
6. optical character recognition, segmentation, and even for artificial neural netwroks.
7. You can import images in color, grey scale or unchanged using individual commands i) cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag. ii) cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode iii) cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel Instead of these three flags, you can simply pass integers 1, 0 or -1 respectively.

In [16]:
```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt


grey_img = cv2.imread("/content/water_body_1.jpg")
color_img = cv2.imread("/content/water_body_1.jpg", 1)[:, :, ::-1]

#images opened using cv2 are numpy arrays
print(type(grey_img))
print(type(color_img))

# Use the function cv2.imshow() to display an image in a window.
```

```
# First argument is the window name which is a string. second argument is our im

plt.figure(figsize=(5,5))

plt.subplot(2,1,1)
plt.imshow(grey_img,'gray')
plt.title('Gray Image', fontsize=14)
plt.axis('on')

plt.subplot(2,1,2)
plt.imshow(color_img)
plt.title('Color Image', fontsize=14)
plt.axis('on')
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

Out[16]: (-0.5, 2006.5, 2008.5, -0.5)



In [ ]:
```
import cv2
```

In [17]:
```
img = cv2.imread("/content/water_body_1.jpg")
```

In [18]:
```
print(img)
```

```
[[[1 0 7]
  [1 0 7]
  [0 0 7]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[1 0 7]
  [1 0 5]
  [0 0 7]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[1 0 5]
  [1 0 4]
  [0 0 5]
  ...
  [0 1 0]
  [0 1 0]
  [0 1 0]]

 ...

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]]
```

In [19]:
```
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
```

In [20]:
```
cv2_imshow(img)
```

In [ ]:
```
plt.imshow(img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7802b67c4c10>
```

In [ ]:
```
img1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

In [ ]:
```
plt.imshow(img1)
plt.axis("off")
```

In [ ]:
```
import numpy as np
np.shape(img1)
```

In [ ]:
```
img2 = cv2.cvtColor(img1,cv2.COLOR_RGB2GRAY)
```

In [ ]:
```
np.shape(img2)
```

In [ ]:
```
plt.imshow(img2,"gray")
plt.axis("off")
plt.title("Gray scale image")
```

In [ ]:
```
cv2.imwrite("/content/Mona_Lisa1.jpg",img2)
```

Code for reading multiple files

Reading multiple images from a folder

1. The glob module finds all the path names matching a specified pattern according to the rules used by the Unix shell
2. The glob.glob returns the list of files with their full path

In [ ]:
```
#import the library opencv
import cv2
from google.colab.patches import cv2_imshow
import glob
#select the path
path = "/content/drive/MyDrive/FIPLab/Dataset/*.*"
for file in glob.glob(path):
    print(file)     #just to see all file names are printed
    a= cv2.imread(file)  #now, we can read each file since we have the full path
    print(a)  #print numpy arrays for each file
#process each image - change color from BGR to RGB.
    c = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
    cv2_imshow(c)
```

SciPy

In [ ]:
```
from scipy import misc
import imageio
import matplotlib.pyplot as plt

# reads a raccoon face
face = misc.face()

# save the image
imageio.imsave('/content/drive/MyDrive/0.png', face)

plt.imshow(face)
plt.show()
```

In [ ]:
```

```

Color Coversion

1. BGR to Gray

Gray-scale Images:

A gray-scale (or gray level) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is

that less information needs to be provided for each pixel. In a gray-scale image, each pixel has a value between 0 and 255, where zero corresponds to “black” and 255 corresponds to “white”. The values in between 0 and 255 are varying shades of gray, where values closer to 0 are darker and values closer to 255 are lighter.

In [ ]:
```python
# Python program to explain cv2.cvtColor() method

# importing cv2
import cv2
from google.colab.patches import cv2_imshow

# path
path = r'/content/drive/MyDrive/FIPLab/Dataset/monalisa.jpg'

# Reading an image in default mode
src = cv2.imread(path)

# Window name in which image is displayed
window_name = 'Image'

# Using cv2.cvtColor() method
# Using cv2.COLOR_BGR2GRAY color space
# conversion code
image = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY )

# Displaying the image
cv2_imshow(image)
```

In [ ]:
```python
# Using the cv2.imread() function with flag=zero
# Import opencv
import cv2

# Use the second argument or (flag value) zero
# that specifies the image is to be read in grayscale mode
img = cv2.imread('C:\\Documents\\full_path\\tomatoes.jpg', 0)

cv2.imshow('Grayscale Image', img)
```

In [ ]:
```python
# import required libraries
import cv2
import matplotlib.pyplot as plt
```

```
# load the input image
img = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/raccoon.png')

# convert the input image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# apply thresholding to convert grayscale to binary image
ret, thresh = cv2.threshold(gray,127,255,0)

# convert BGR to RGB to display using matplotlib
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display Original, Grayscale and Binary Images
plt.subplot(131),plt.imshow(imgRGB),plt.title('Original Image'), plt.axis('off')
plt.subplot(132),plt.imshow(gray,'gray'),plt.title('Grayscale Image'),plt.axis('
plt.subplot(133),plt.imshow(thresh,'gray'),plt.title('Binary Image'),plt.axis('o
plt.show()
print(imgRGB)
print(gray)
print(thresh)
```

In [ ]:
```
help(cv2.threshold)
```

2. BGR to HSV:

HSV Color Model:

An HSV color model is the most accurate color model as long as the way humans perceive colors. How humans perceive colors is not like how RGB or CMYK make colors. They are just primary colors fused to create the spectrum. The H stands for Hue, S stands for Saturation, and the V stand for value.

Hue: Hue tells the angle to look at the cylindrical disk. The hue represents the color. The hue value ranges from o to 360 degrees.

Saturation: The saturation value tells us how much quantity of respective color must be added. A 100% saturation means that complete pure color is added, while a 0% saturation means no color is added, resulting in grayscale.

Value: The value represents the brightness concerning the saturation of the color. the value 0 represents total black darkness, while the value 100 will mean a full brightness and depend on the saturation.

In [ ]:
```
# Python program to explain cv2.cvtColor() method

# importing cv2
import cv2

# path
path = r'/content/drive/MyDrive/FIPLab/Dataset/monalisa.jpg'

# Reading an image in default mode
src = cv2.imread(path)

# Window name in which image is displayed
```

```
window_name = 'Image'

# Using cv2.cvtColor() method
# Using cv2.COLOR_BGR2HSV color space
# conversion code
image = cv2.cvtColor(src, cv2.COLOR_BGR2HSV )

# Displaying the image
cv2_imshow(image)
print(image)

bgrimg = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
cv2_imshow(bgrimg)
```

Image Operations

1. Image translation

In [ ]:
```
import cv2
import numpy as np
image = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/agri_0_1017.jpeg')
# Store height and width of the image
height, width = image.shape[:2]
quarter_height, quarter_width = height / 4, width / 4
T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])
# We use warpAffine to transform
# the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height),0.1)

cv2_imshow(image)
cv2_imshow(img_translation)
```

2 Image rotation

In [ ]:
```
import cv2
from google.colab.patches import cv2_imshow

# Reading the image
image = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/raccoon.png')

height, width = image.shape[:2]
# get the center coordinates of the image to create the 2D rotation matrix
center = (width/2, height/2)

# using cv2.getRotationMatrix2D() to get the rotation matrix
rotate_matrix = cv2.getRotationMatrix2D(center=center, angle=45, scale=1)
# rotate the image using cv2.warpAffine
rotated_image = cv2.warpAffine(src=image, M=rotate_matrix, dsize=(width, height))

cv2_imshow(image)
print(rotate_matrix)
cv2_imshow(rotated_image)
cv2.imwrite('rotated_image.jpg', rotated_image)
```

Output hidden; open in https://colab.research.google.com to view.

Using cv2.rotate() function

cv2.rotate(img, rotateCode)

rotateCode is a rotate flag specifying how to rotate the array. The three rotate flags are as below –

1. cv2.ROTATE_90_CLOCKWISE

2. cv2.ROTATE_180

3. cv2.ROTATE_90_COUNTERCLOCKWISE

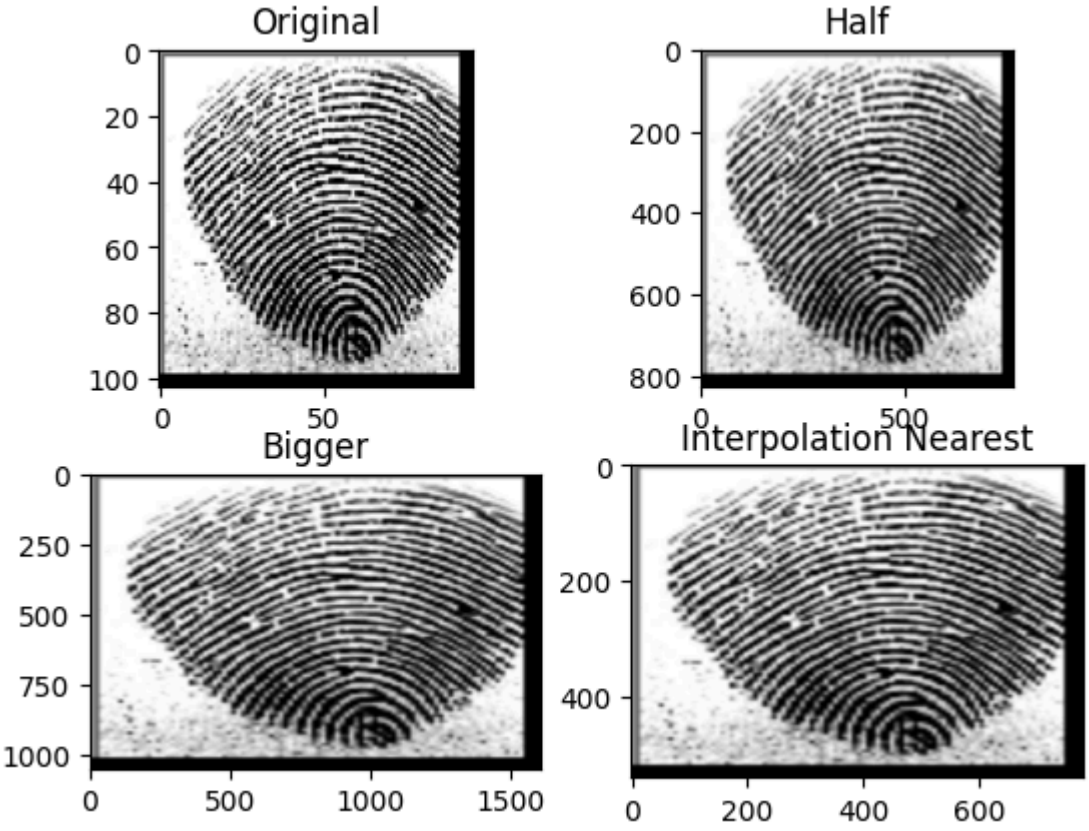
In [ ]:
```
# import required libraries
import cv2
from google.colab.patches import cv2_imshow
# load the input image
img = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/agri_0_1017.jpeg')

# rotate the image by 180 degree clockwise
img_cw_180 = cv2.rotate(img, cv2.ROTATE_180)

# display the rotated image
print('Original')
cv2_imshow(img)
print('Rotated')
cv2_imshow(img_cw_180)
```

```
Original
```

Rotated

3 Image scaling and resizing

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread(r"/content/drive/MyDrive/FIPLab/Dataset/finger_print2.bmp")

half = cv2.resize(image, (0, 0), fx = 8, fy = 8)
bigger = cv2.resize(image, (1610,1050))

stretch_near = cv2.resize(image, (780, 540),
                          interpolation = cv2.INTER_LINEAR)

Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]
images =[image, half, bigger, stretch_near]
count = 4

for i in range(count):
        plt.subplot(2, 2, i + 1)
        plt.title(Titles[i])
        plt.imshow(images[i])


plt.show()
```



In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
image = cv2.imread("/content/drive/MyDrive/FIPLab/Dataset/Mona_Lisa.jpg")
```

In [3]:
```
np.shape(image)
```

Out[3]: (599, 396, 3)

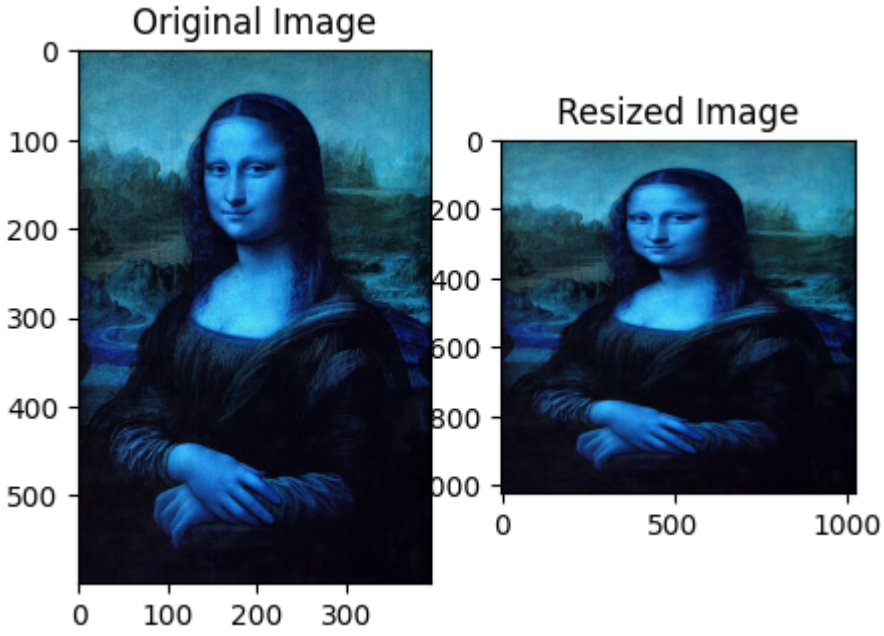
In [7]:
```
img1 = cv2.resize(image, (1024,1024))
```

In [8]:
```
np.shape(img1)
```

Out[8]: (1024, 1024, 3)

In [16]:
```
plt.figure(figsize = (5,5))
plt.subplot(121)
plt.imshow(image)
plt.title("Original Image")
plt.axis("on")
plt.subplot(122)
plt.imshow(img1)
plt.title("Resized Image")
plt.axis("on")
```

Out[16]: (-0.5, 1023.5, 1023.5, -0.5)

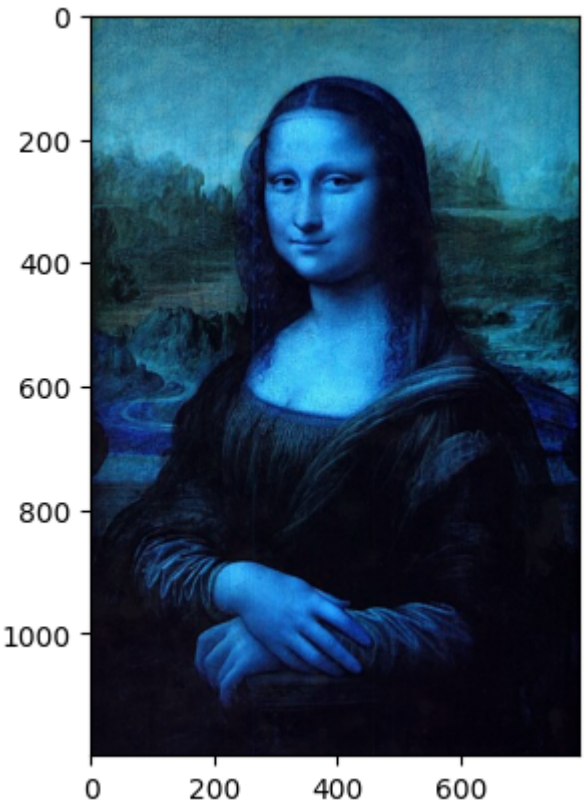


In [17]:
```
cv2.imwrite("/content/drive/MyDrive/FIPLab/Dataset/Mona_Lisa1.jpg",image)
cv2.imwrite("/content/drive/MyDrive/FIPLab/Dataset/Mona_Lisa2.jpg",img1)
```

Out[17]: True

In [20]:
```
half = cv2.resize(image, (0, 0), fx = 2, fy = 2)
plt.imshow(half)
np.shape(half)
```

Out[20]: (1198, 792, 3)



Histogram of an Image

In [ ]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
I =cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/bubbles.tif')
```

In [ ]:
```
plt.imshow(I)
plt.show()
```

In [ ]:
```
print(I.shape)
```

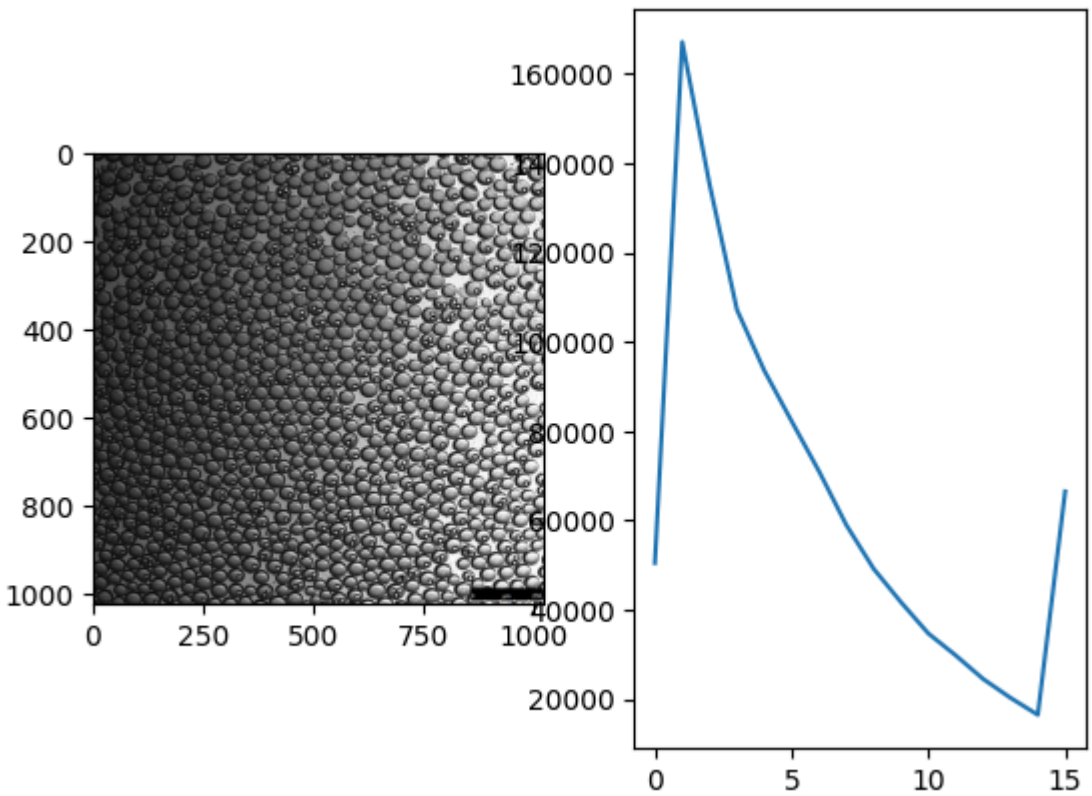
```
(1024, 1024, 3)
```

In [ ]:
```
I1=cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
```

In [ ]:
```
print(I1.shape)
```

```
(1024, 1024)
```

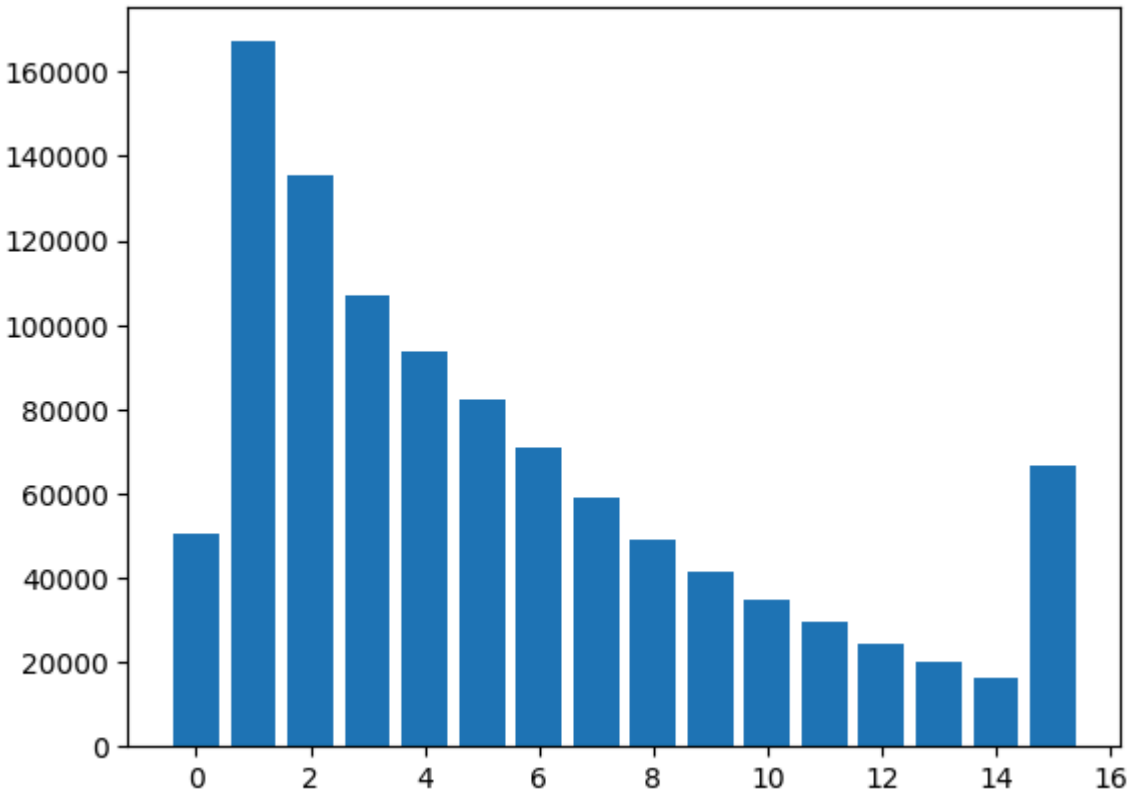
In [ ]:
```
histr = cv2.calcHist([I1],[0],None,[16],[0,256])
plt.subplot(121)
plt.imshow(I,cmap='gray')
plt.subplot(122)
plt.plot(histr)
plt.show()
```



In [ ]:
```
print(histr)
```

In [ ]:
```
print(histr)
```

In [ ]:
```
x= list(range(0,16))
histr = np.reshape(histr,-1)
histr.shape
plt.bar(x,histr)
plt.show()
```



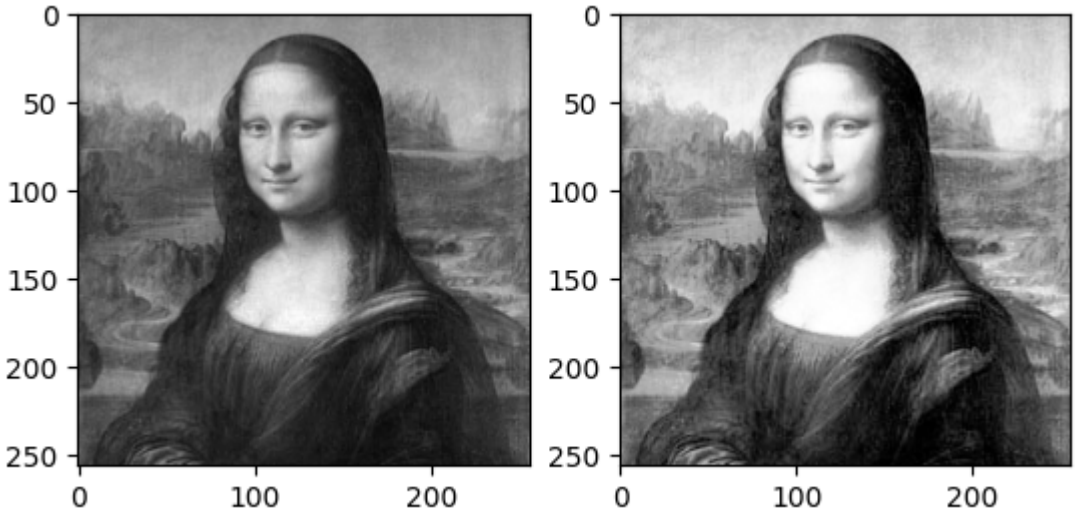
In [ ]:
```
plt.hist(I1.ravel(),256,[0,256])
plt.show()
```

In [ ]:
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('/content/drive/MyDrive/FIPLab/Dataset/monalisa.jpg', cv.IMREAD_
plt.subplot(121)
plt.imshow(img,'gray')
plt.subplot(122)
plt.hist(img.ravel(),256,[0,256])
plt.show()
```

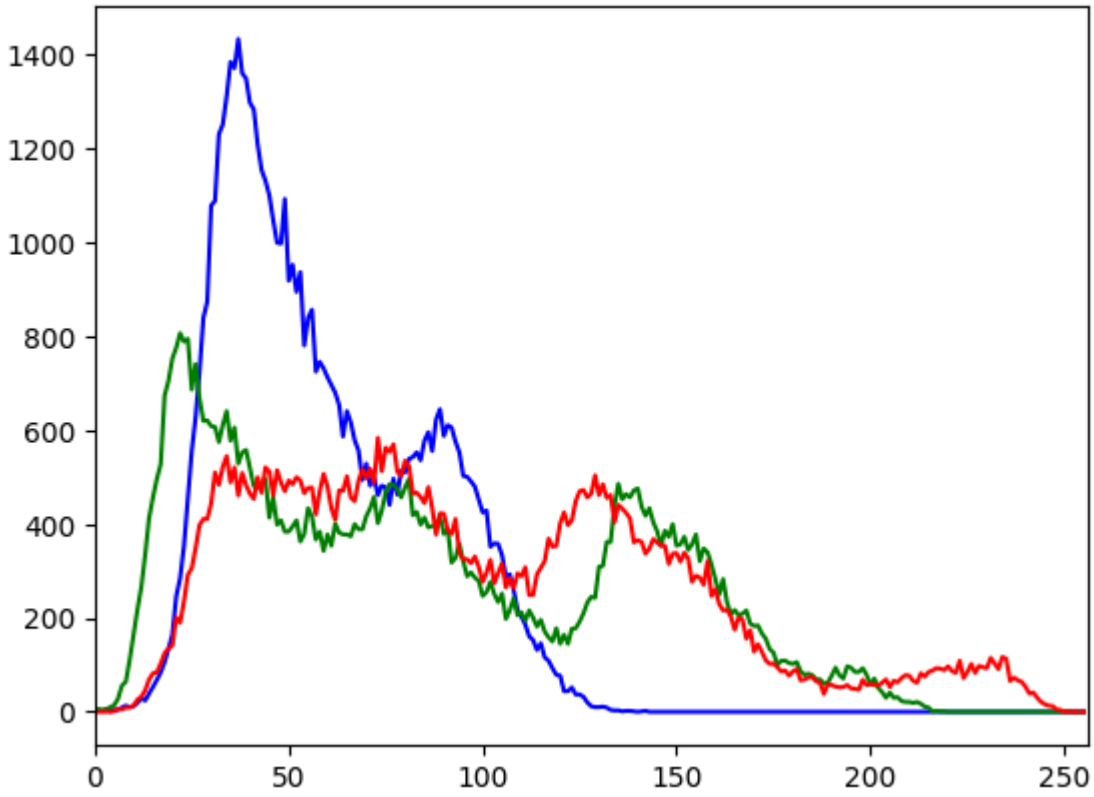
In [ ]:
```
I3 = cv.equalizeHist(img)
```

In [ ]:
```
plt.hist(I3.ravel(),256,[0,256])
plt.show()
```

In [ ]:
```
plt.subplot(121)
plt.imshow(img,'gray')
plt.subplot(122)
plt.imshow(I3,cmap='gray')
plt.show()
```

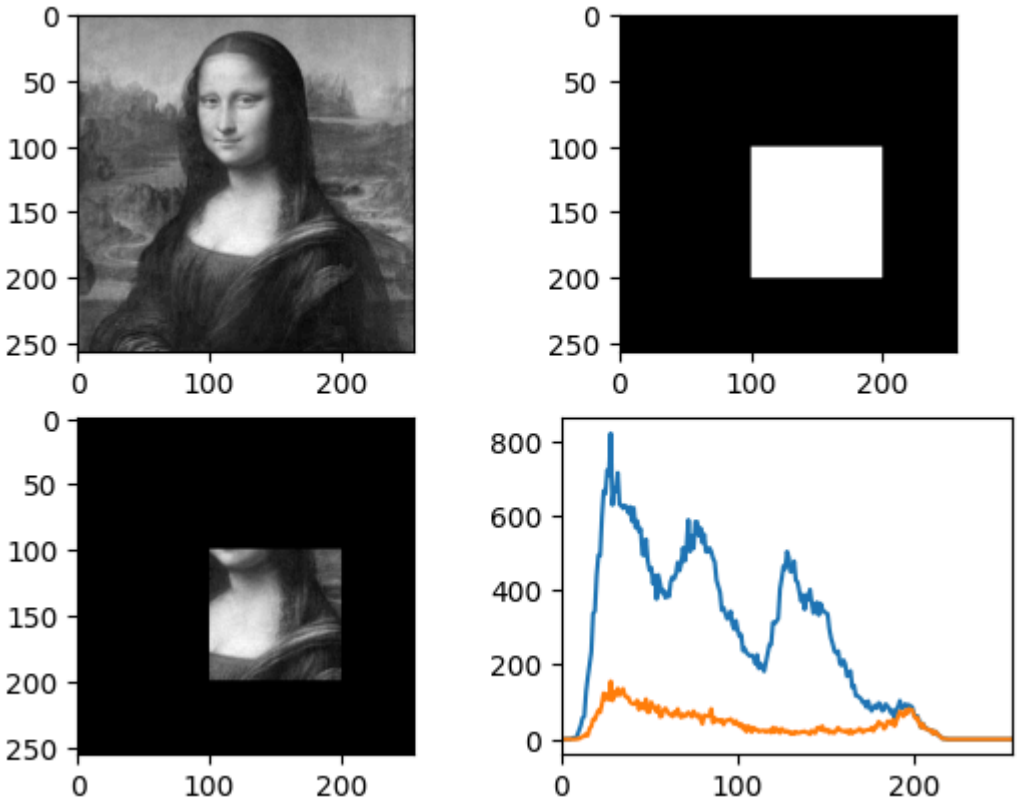


In [ ]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/monalisa.jpg')
color = ('b','g','r')
for i,col in enumerate(color):
 histr = cv2.calcHist([img],[i],None,[256],[0,256])
 plt.plot(histr,color = col)
 plt.xlim([0,256])
plt.show()
```



In [ ]:
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('/content/drive/MyDrive/FIPLab/Dataset/monalisa.jpg', cv.IMREAD_
# create a mask
mask = np.zeros(img.shape, np.uint8)
mask[100:200, 100:200] = 255
```

```
masked_img = cv.bitwise_and(img,img,mask = mask)
# Calculate histogram with mask and without mask
# Check third argument for mask
hist_full = cv.calcHist([img],[0],None,[256],[0,256])
hist_mask = cv.calcHist([img],[0],mask,[256],[0,256])
plt.subplot(221), plt.imshow(img, 'gray')
plt.subplot(222), plt.imshow(mask,'gray')
plt.subplot(223), plt.imshow(masked_img, 'gray')
plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask)
plt.xlim([0,256])
plt.show()
```



In [ ]:
```
img.shape
```

Out[ ]: (256, 256)

Image Smoothing / Blurring

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

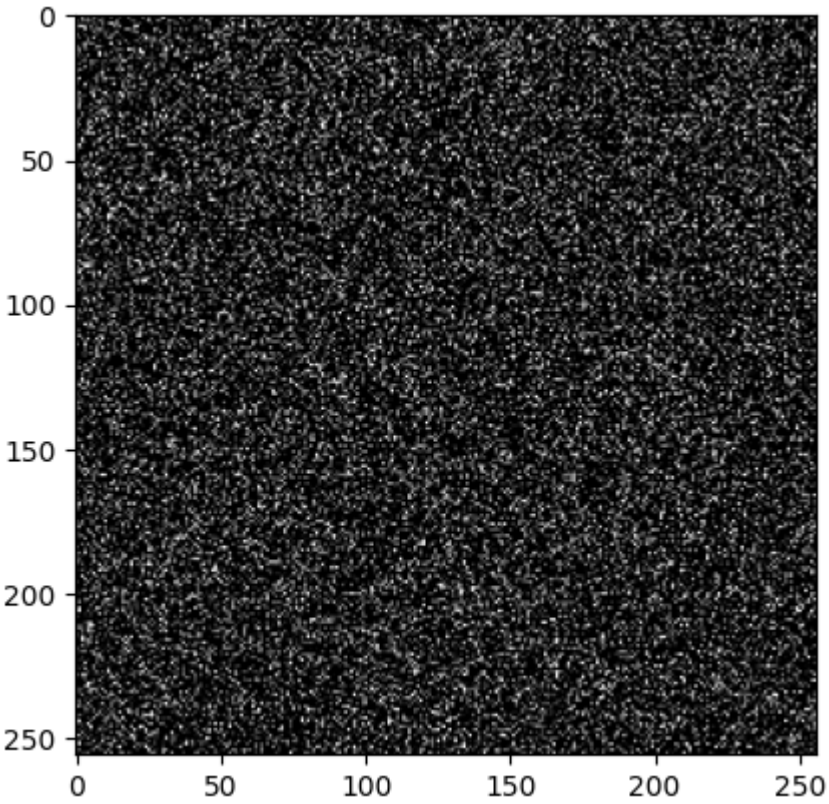
In [6]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
img = cv2.imread('/content/cameraman123.jpg',0)
```

In [7]:
```
img.shape
```

Out[7]: (256, 256)

In [8]:
```
# Generate random Gaussian noise
mean = 0
stddev = 100
noise = np.zeros(img.shape, np.uint8)
cv2.randn(noise, mean, stddev)
plt.imshow(noise,'gray')
```

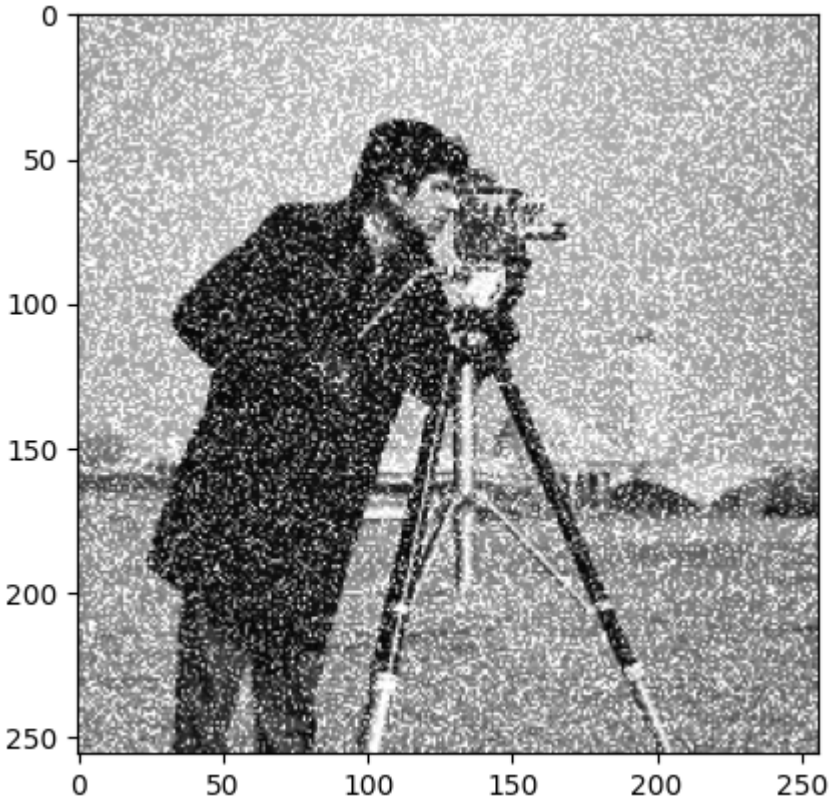
Out[8]: <matplotlib.image.AxesImage at 0x7eac9021e620>



In [11]:
```
# Add noise to image
noisy_img = cv2.add(img, noise)
```

In [19]:
```
plt.imshow(noisy_img,'gray')
```

Out[19]: <matplotlib.image.AxesImage at 0x7eac8c2c8d00>

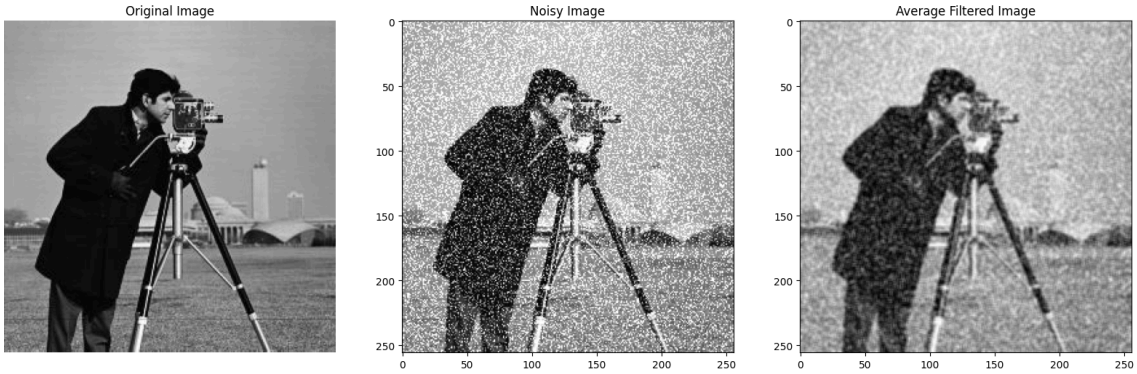


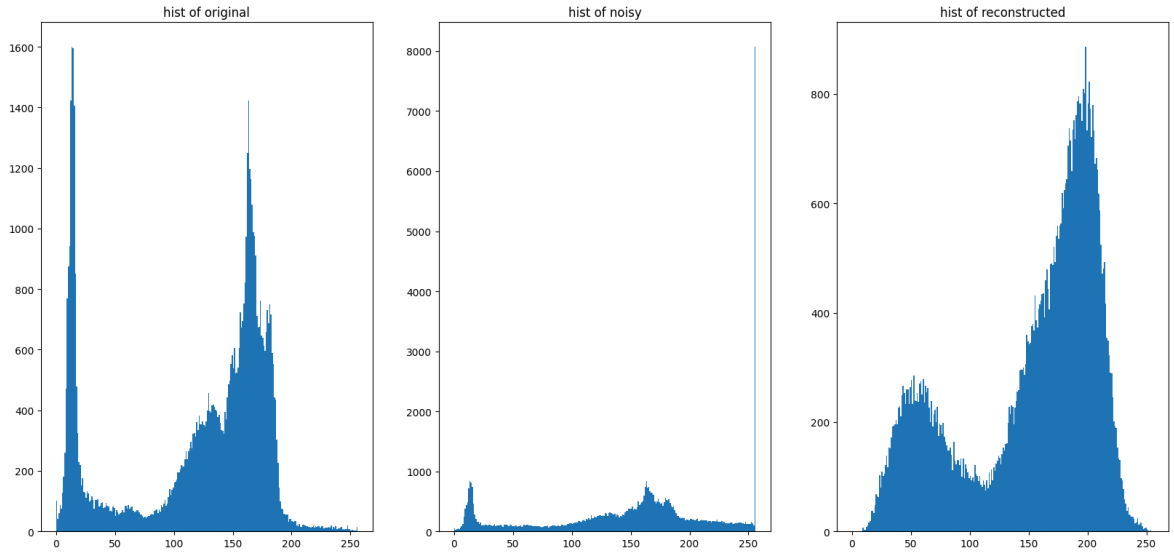
In [13]:
```
# Smoothing/low-pass filter kernel
kernel = np.ones((3,3),np.float32)/9
kernel
```

Out[13]:
```
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]], dtype=float32)
```

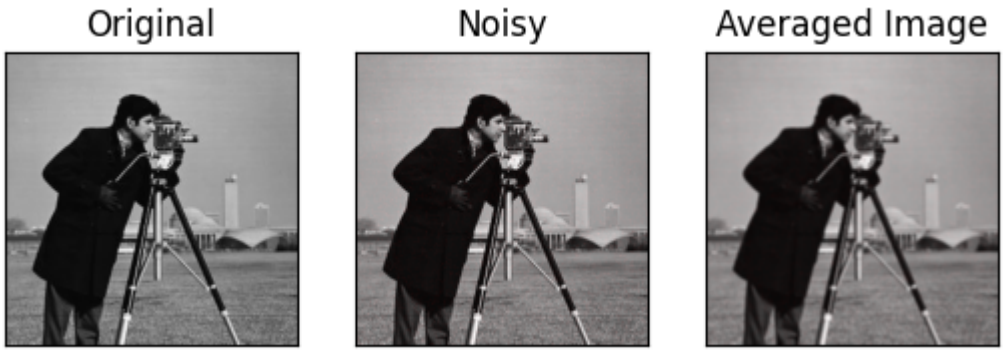
In [14]:
```
# Smoothing/low-pass filter using filter2D command
dst = cv2.filter2D(noisy_img,-1, kernel)
```

In [15]:
```
plt.figure(figsize=(20,20))
plt.subplot(231), plt.imshow(img,'gray'), plt.title('Original Image'),plt.axis('
plt.subplot(232), plt.imshow(noisy_img,'gray'), plt.title('Noisy Image')
plt.subplot(233),plt.imshow(dst, 'gray'),plt.title('Average Filtered Image')
plt.subplot(234),plt.hist(img.ravel(),256,[0,256]),plt.title('hist of original')
plt.subplot(235),plt.hist(noisy_img.ravel(),256,[0,256]),plt.title('hist of nois
plt.subplot(236),plt.hist(dst.ravel(),256,[0,256]),plt.title('hist of reconstruc
plt.show()
```



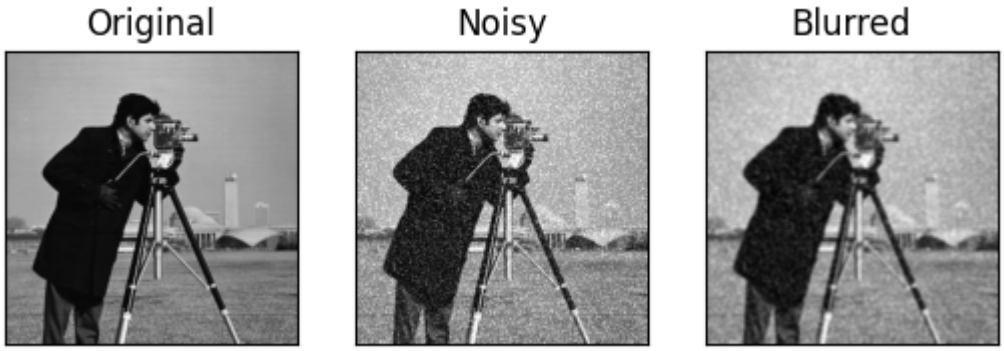


In [ ]:
```
# blur() function for averaging filter in opencv
blur = cv2.blur(noisy_img,(3,3))
plt.subplot(131),plt.imshow(img,'gray'),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(noisy_img,'gray'),plt.title('Noisy')
plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(blur,'gray'),plt.title('Averaged Image')
plt.xticks([]), plt.yticks([])
plt.show()
```



In [ ]:
```
# GaussianBlur() function for guassian filter in opencv
blur = cv2.GaussianBlur(noisy_img,(5,5),0)
plt.subplot(131),plt.imshow(img,'gray'),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(noisy_img,'gray'),plt.title('Noisy')
```

```
plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(blur,'gray'),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```



In [ ]:
```
# Adding uniform noise
uni_noise=np.zeros(img.shape,dtype=np.uint8)
cv2.randu(uni_noise,0,255)
```

In [ ]:
```
plt.hist(uni_noise.ravel(),256,[0,256])
plt.show()
```

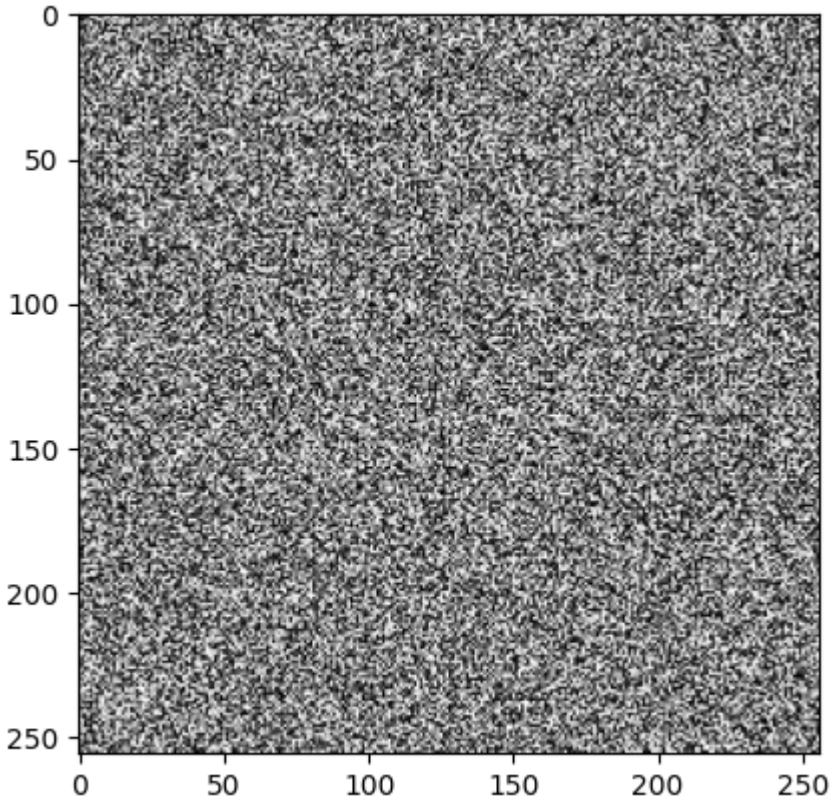
In [ ]:
```
# Add uniform noise to the image
un_img=cv2.add(img,uni_noise)
```

In [ ]:
```
uni_filtered =cv2.GaussianBlur(un_img,(3,3),0)
```

In [ ]:
```
plt.subplot(1,4,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

plt.subplot(1,4,2)
plt.imshow(uni_noise,cmap='gray')
plt.axis("off")
plt.title("Uniform Noise")

plt.subplot(1,4,3)
plt.imshow(un_img,cmap='gray')
plt.axis("off")
plt.title("Combined")

plt.subplot(1,4,4)
plt.imshow(uni_filtered,cmap='gray')
plt.axis("off")
plt.title("Filtered")
```

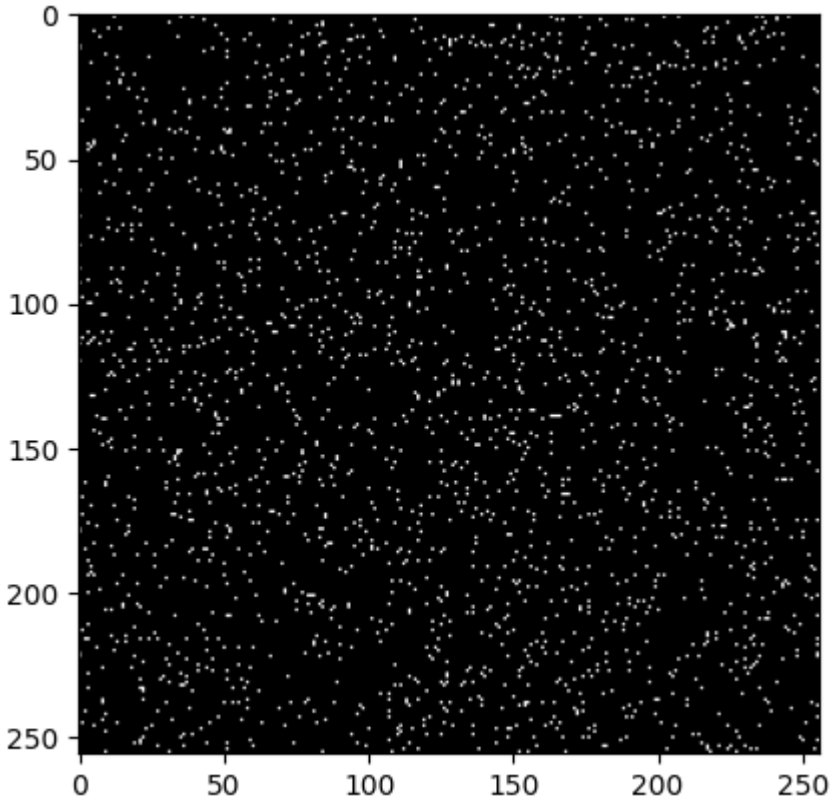
In [ ]:
```
imp_noise=np.zeros(img.shape,dtype=np.uint8)
cv2.randu(imp_noise,0,255)
plt.imshow(imp_noise,'gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7c6550f55ea0>
```



In [ ]:
```
imp_noise=cv2.threshold(imp_noise,245,255,cv2.THRESH_BINARY)[1]
plt.imshow(imp_noise,'gray')
```

Out[ ]: <matplotlib.image.AxesImage at 0x7c6550c7b910>

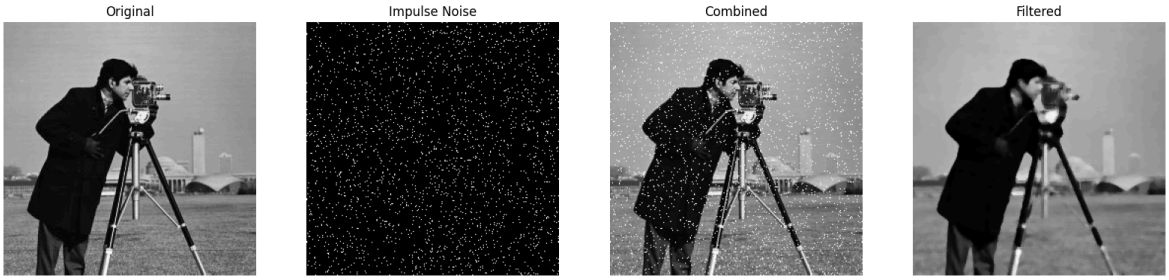


In [ ]:
```
noise_img=cv2.add(img,imp_noise)
```

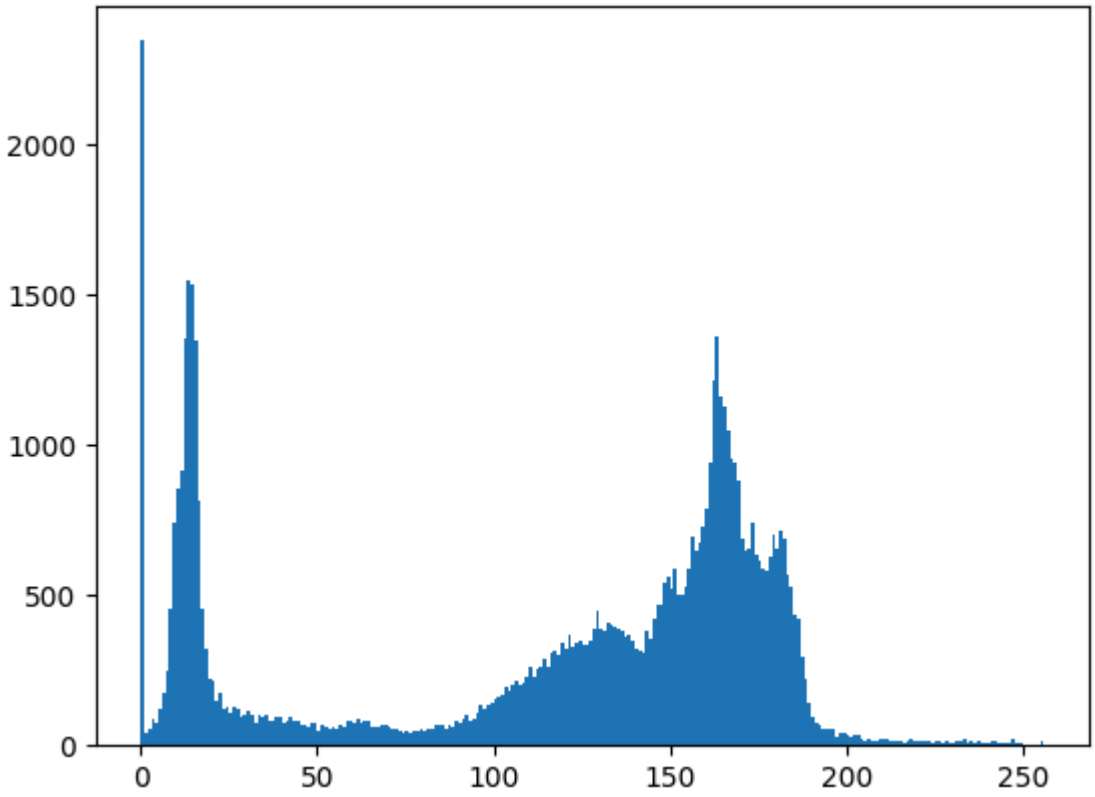
In [ ]:
```
median = cv2.medianBlur(noise_img,5)
```

In [ ]:
```
fig = plt.figure(figsize=(20,20))
plt.subplot(1,4,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

plt.subplot(1,4,2)
plt.imshow(imp_noise,cmap='gray')
plt.axis("off")
plt.title("Impulse Noise")

plt.subplot(1,4,3)
plt.imshow(noise_img,cmap='gray')
plt.axis("off")
plt.title("Combined")

plt.subplot(1,4,4)
plt.imshow(median,cmap='gray')
plt.axis("off")
plt.title("Filtered")
```

Out[ ]:
```
Text(0.5, 1.0, 'Filtered')
```



In [ ]:
```
# Speckle noise - Multiplicative noise
noisy_img = img + img * imp_noise
plt.hist(noisy_img.ravel(),256,[0,256])
plt.show()
```



In [ ]:
```
# GaussianBlur() function for guassian filter in opencv
blur = cv2.medianBlur(noisy_img,3)
fig = plt.figure(figsize=(20,20))
plt.subplot(131),plt.imshow(img,'gray'),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(noisy_img,'gray'),plt.title('Noisy')
plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(blur,'gray'),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

Shapening Filter

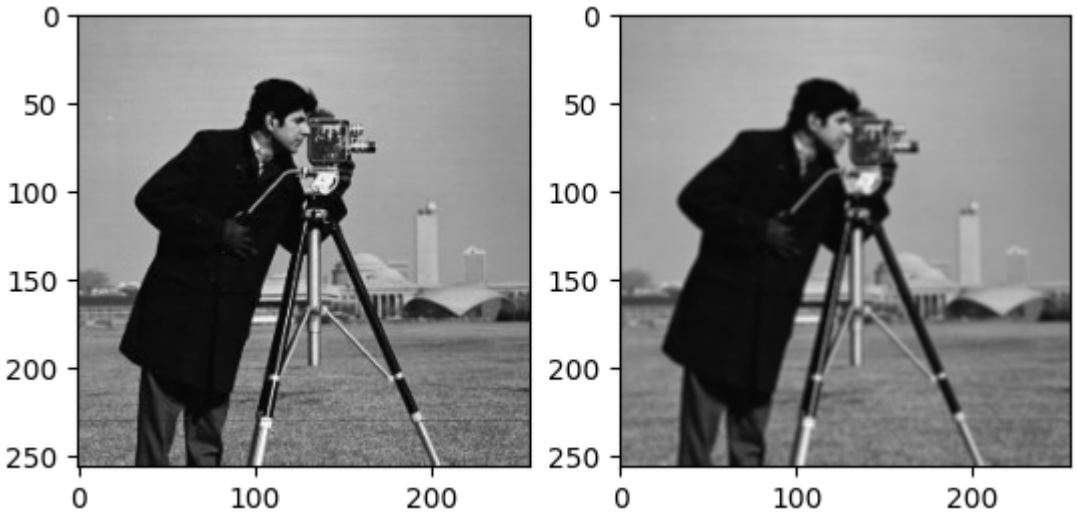
In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the image
img = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/cameraman.jpg',0)
```

In [ ]:
```
# Smoothing/low-pass filter kernel
kernel = np.ones((3,3),np.float32)/9
kernel
```

In [ ]:
```
dst = cv2.filter2D(img,-1,kernel)
```

In [ ]:
```
plt.subplot(121),plt.imshow(img,'gray')
plt.subplot(122),plt.imshow(dst,'gray')
```

Out[ ]:
```
(<Axes: >, <matplotlib.image.AxesImage at 0x7ea1581119f0>)
```



In [ ]:
```
#Create the sharpening kernel
kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
# Apply the sharpening kernel to the image using filter2D
sharpened = cv2.filter2D(dst, -1, kernel)
```

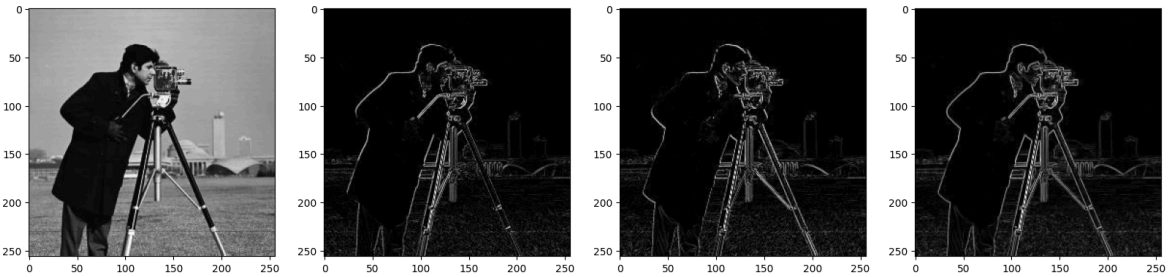
In [ ]:
```
plt.figure(figsize=(20,20))
plt.subplot(131), plt.imshow(img,'gray'), plt.title('Original Image'),plt.axis('
plt.subplot(132),plt.imshow(dst,'gray'), plt.title('Blurred Image'),plt.axis('of
plt.subplot(133),plt.imshow(sharpened, 'gray'),plt.title('Sharpened Image'),plt.
```

Out[ ]:
```
(<Axes: title={'center': 'Sharpened Image'}>,
 <matplotlib.image.AxesImage at 0x7ea1583c1d50>,
 Text(0.5, 1.0, 'Sharpened Image'),
 (-0.5, 255.5, 255.5, -0.5))
```

1. Robert Operator

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
roberts_cross_v = np.array( [[1, 0 ],
                                                    [0,-1 ]] )
roberts_cross_h = np.array( [[ 0, 1 ],
                                                    [ -1, 0 ]] )
img = cv2.imread("/content/drive/MyDrive/FIPLab/Dataset/cameraman.jpg",0).astype
img/=255.0
vertical = cv2.filter2D(img, -1, roberts_cross_v)
horizontal = cv2.filter2D(img, -1, roberts_cross_h)
edged_img_v = np.sqrt(np.square(vertical))
edged_img_h = np.sqrt(np.square(horizontal))
edged_img = np.sqrt(np.square(vertical)+np.square(horizontal))
edged_img_v*=255
edged_img_h*=255
plt.figure(figsize=(20,20))
plt.subplot(141)
plt.imshow(img,'gray')

plt.subplot(142)
plt.imshow(edged_img_v,'gray')

plt.subplot(143)
plt.imshow(edged_img_h,'gray')

plt.subplot(144)
plt.imshow(edged_img,'gray')
print(roberts_cross_v,roberts_cross_h)
```

```
[[ 1  0]
 [ 0 -1]] [[ 0  1]
 [-1  0]]
```



2. Prewitt Operator

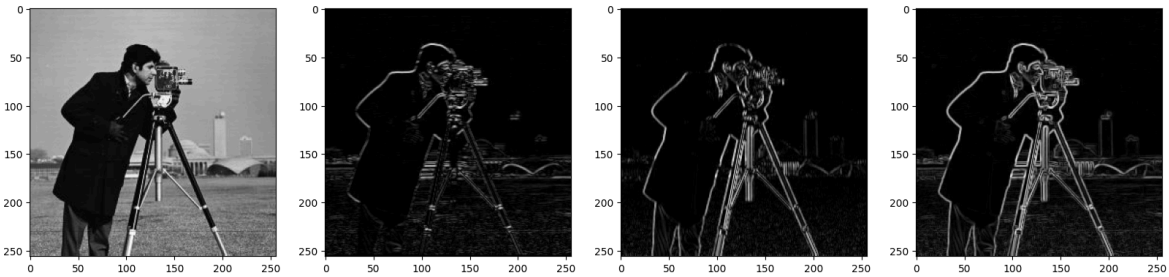
In [ ]:
```
#prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
img_prewittx = cv2.filter2D(img, -1, kernelx)
img_prewitty = cv2.filter2D(img, -1, kernely)
#img_prewittx = np.sqrt(np.square(img_prewittx))
#img_prewitty = np.sqrt(np.square(img_prewitty))
img_prewitt = np.sqrt(np.square(img_prewittx)+np.square(img_prewitty))
```

In [ ]:
```
kernelx,kernely
```

In [ ]:
```
plt.figure(figsize=(20,20))
plt.subplot(141)
```

```
plt.imshow(img,'gray')

plt.subplot(142)
plt.imshow(img_prewittx,'gray')

plt.subplot(143)
plt.imshow(img_prewitty,'gray')

plt.subplot(144)
plt.imshow(img_prewitt,'gray')
```

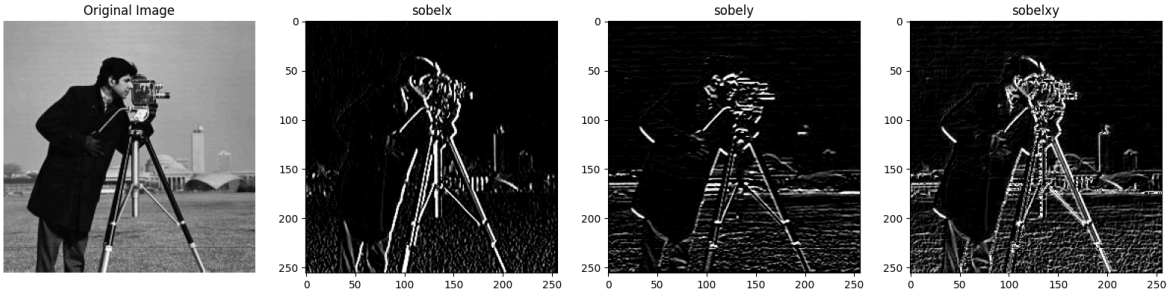
Out[ ]: <matplotlib.image.AxesImage at 0x7ea157e1be20>



3. Sobel Operator

In [ ]:
```
# Sobel Edge Detection
sobelx = cv2.Sobel(src=img, ddepth=-1, dx=1, dy=0, ksize=3) # Sobel Edge Detecti
sobely = cv2.Sobel(src=img, ddepth=-1, dx=0, dy=1, ksize=3) # Sobel Edge Detecti
sobelxy = cv2.Sobel(src=img, ddepth=-1, dx=1, dy=1, ksize=3) # Combined X and Y

sobelx = np.abs(sobelx)
sobely = np.abs(sobely)
sobelxy = np.abs(sobelx+sobely)

plt.figure(figsize=(20,20))
plt.subplot(141), plt.imshow(img,'gray'), plt.title('Original Image'),plt.axis('
plt.subplot(142),plt.imshow(sobelx,'gray'),plt.title('sobelx')
plt.subplot(143),plt.imshow(sobely,'gray'),plt.title('sobely')
plt.subplot(144),plt.imshow(sobelxy,'gray'),plt.title('sobelxy')
```

Out[ ]:
```
(<Axes: title={'center': 'sobelxy'}>,
 <matplotlib.image.AxesImage at 0x7c69b4e8a740>,
 Text(0.5, 1.0, 'sobelxy'))
```



In [ ]:
```
sobelxy
```

Out[ ]:
```
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
        -2.35294118e-02, -2.35294118e-02,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
        -3.92156863e-03, -7.84313725e-03,  0.00000000e+00],
       ...,
       [ 0.00000000e+00, -2.35294118e-02,  2.07843137e-01, ...,
         2.74509804e-02,  2.74509804e-02,  0.00000000e+00],
       [ 0.00000000e+00,  3.52941176e-02,  2.62745098e-01, ...,
         1.09803922e-01, -5.55111512e-17,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])
```

In [ ]:
```
sobelxy
```

Out[ ]:
```
array([[0.        , 0.        , 0.        , ..., 0.02352941, 0.02352941,
        0.        ],
       [0.01568627, 0.01568627, 0.01568627, ..., 0.05490196, 0.00784314,
        0.01568627],
       [0.        , 0.        , 0.        , ..., 0.01568627, 0.03137255,
        0.00784314],
       ...,
       [0.01568627, 0.04705882, 0.01568627, ..., 0.03921569, 0.2745098 ,
        0.0627451 ],
       [0.01568627, 0.11764706, 0.65882353, ..., 0.03921569, 0.32941176,
        0.04705882],
       [0.        , 0.10196078, 0.57254902, ..., 0.22745098, 0.28235294,
        0.        ]])
```

Erosion - It erodes away the boundaries of foreground object

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('/content/drive/MyDrive/FIPLab/Dataset/morph_tophat.png', cv2.I
```

In [ ]:
```
img
```

In [ ]:
```
# apply thresholding to convert grayscale to binary image
ret, thresh = cv2.threshold(img,180,255,cv2.THRESH_BINARY)
```

In [ ]:
```
thresh
```

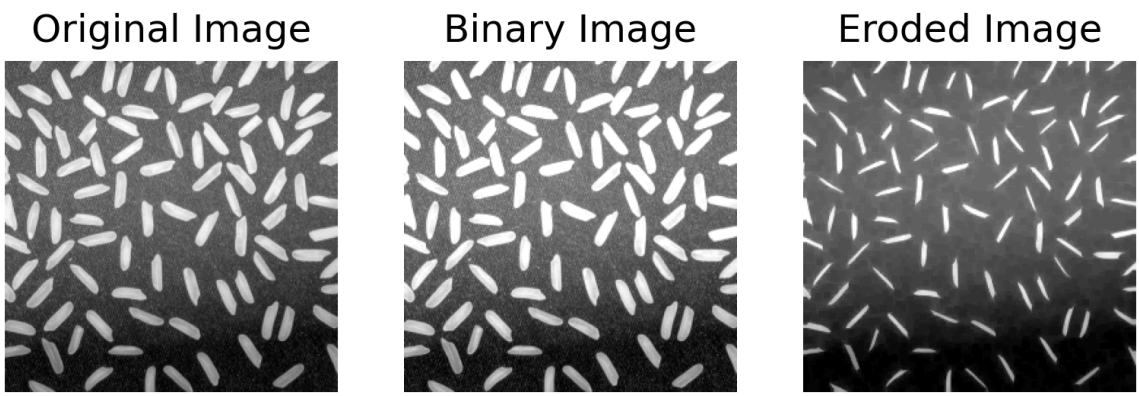
In [ ]:
```
ret
```

Out[ ]:
```
180.0
```

In [ ]:
```
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(thresh,kernel,iterations =1)
```

In [ ]:
```
kernel
```

In [ ]:
```
fig = plt.figure(dpi=300)
plt.subplot(131)
plt.imshow(img,'gray')
plt.axis('off')
plt.title('Original Image')

plt.subplot(132)
plt.imshow(thresh,'gray')
plt.axis('off')
plt.title('Binary Image')

plt.subplot(133)
plt.imshow(erosion,'gray')
plt.axis('off')
plt.title('Eroded Image')
plt.show()
```

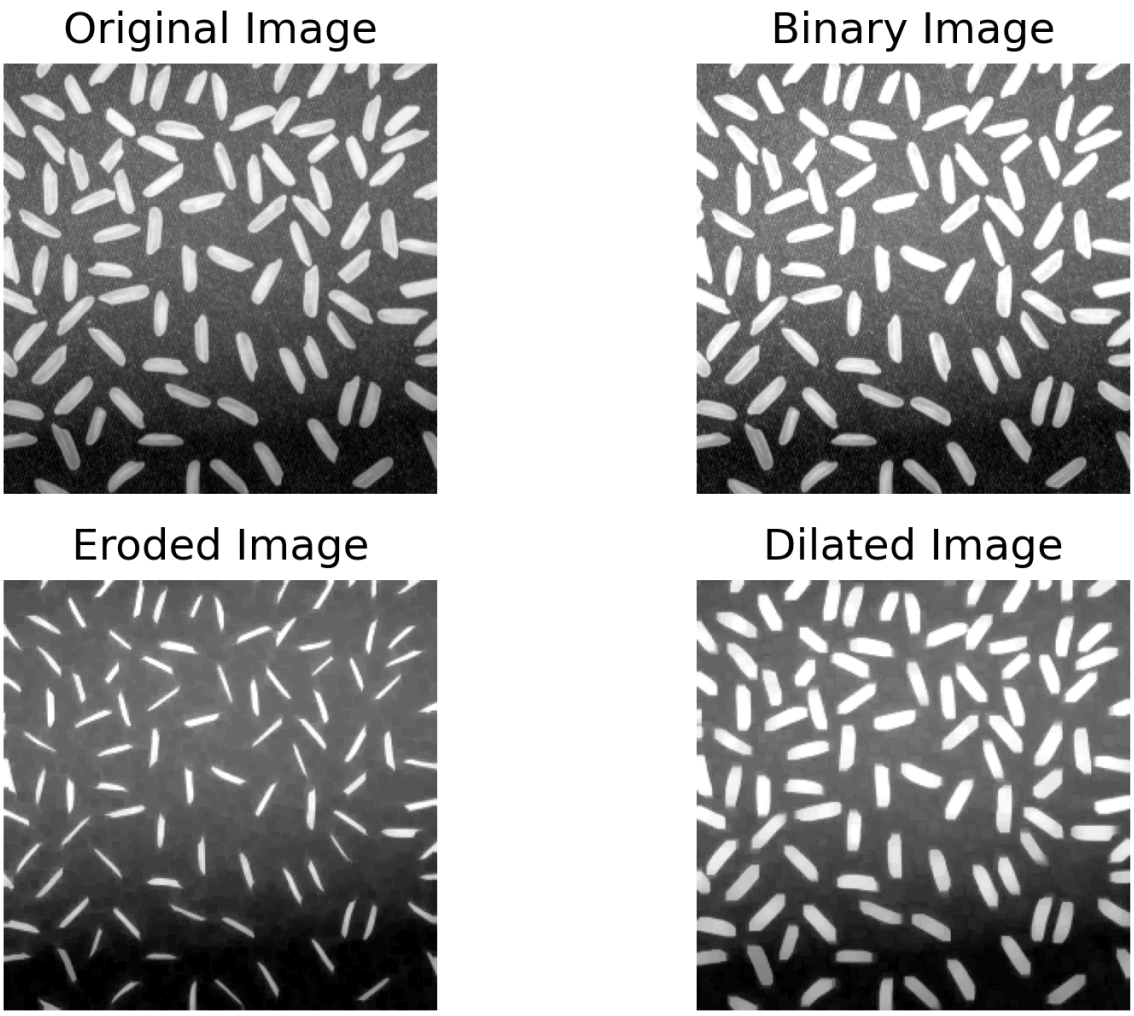


Dilation

In [ ]:
```
dilation = cv2.dilate(erosion,kernel,iterations = 1)
```

In [ ]:
```
fig = plt.figure(dpi=300)
plt.subplot(221)
plt.imshow(img,'gray')
plt.axis('off')
plt.title('Original Image')

plt.subplot(222)
plt.imshow(thresh,'gray')
plt.axis('off')
plt.title('Binary Image')

plt.subplot(223)
plt.imshow(erosion,'gray')
plt.axis('off')
plt.title('Eroded Image')

plt.subplot(224)
```

```
plt.imshow(dilation,'gray')
plt.axis('off')
plt.title('Dilated Image')
plt.show()
```



Opening

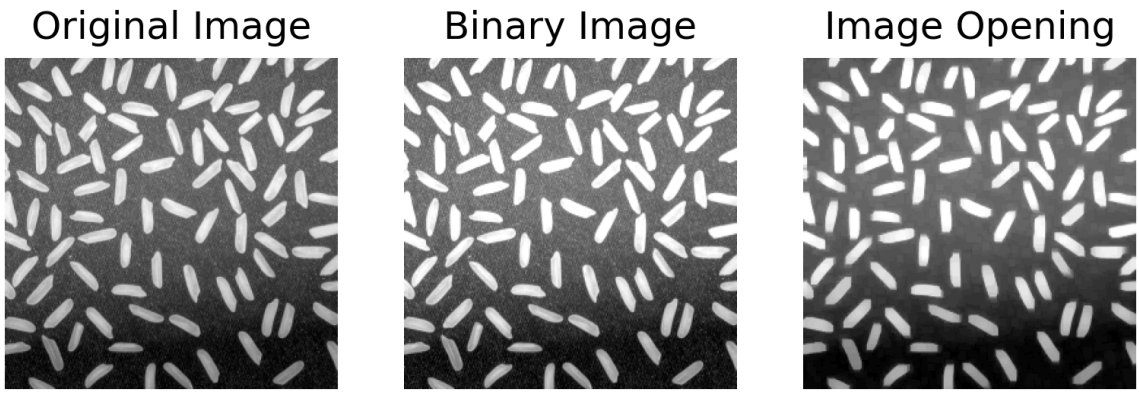
The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations. Morphological opening is useful for removing small objects and thin lines from an image while preserving the shape and size of larger objects in the image.

In [ ]:
```
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations = 1)
```

In [ ]:
```
fig = plt.figure(dpi=300)
plt.subplot(131)
plt.imshow(img,'gray')
plt.axis('off')
plt.title('Original Image')

plt.subplot(132)
plt.imshow(thresh,'gray')
plt.axis('off')
plt.title('Binary Image')
```

```
plt.subplot(133)
plt.imshow(opening,'gray')
plt.axis('off')
plt.title('Image Opening')
```

Out[ ]: Text(0.5, 1.0, 'Image Opening')



Closing

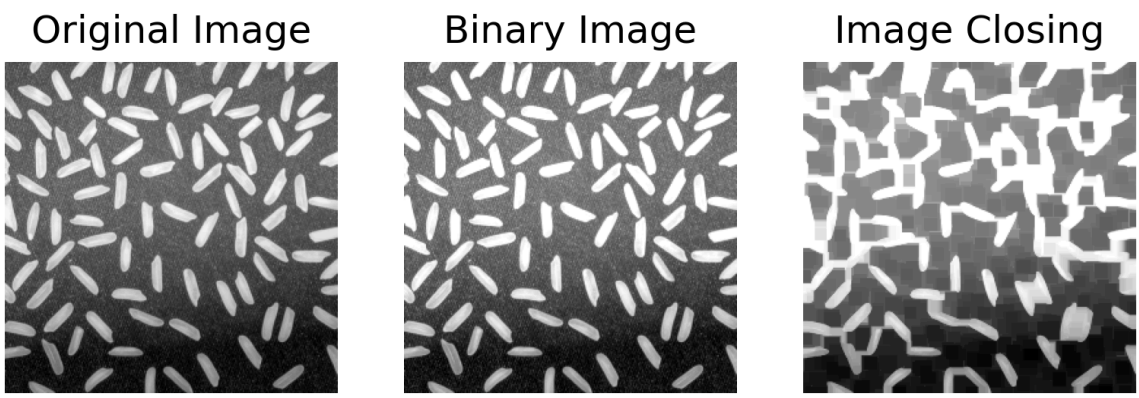
The closing operation dilates an image and then erodes the dilated image, using the same structuring element for both operations.

Morphological closing is useful for filling small holes in an image while preserving the shape and size of large holes and objects in the image.

In [ ]:
```
closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel,iterations=2)
```

In [ ]:
```
fig = plt.figure(dpi=300)
plt.subplot(131)
plt.imshow(img,'gray')
plt.axis('off')
plt.title('Original Image')

plt.subplot(132)
plt.imshow(thresh,'gray')
plt.axis('off')
plt.title('Binary Image')

plt.subplot(133)
plt.imshow(closing,'gray')
plt.axis('off')
plt.title('Image Closing')
```

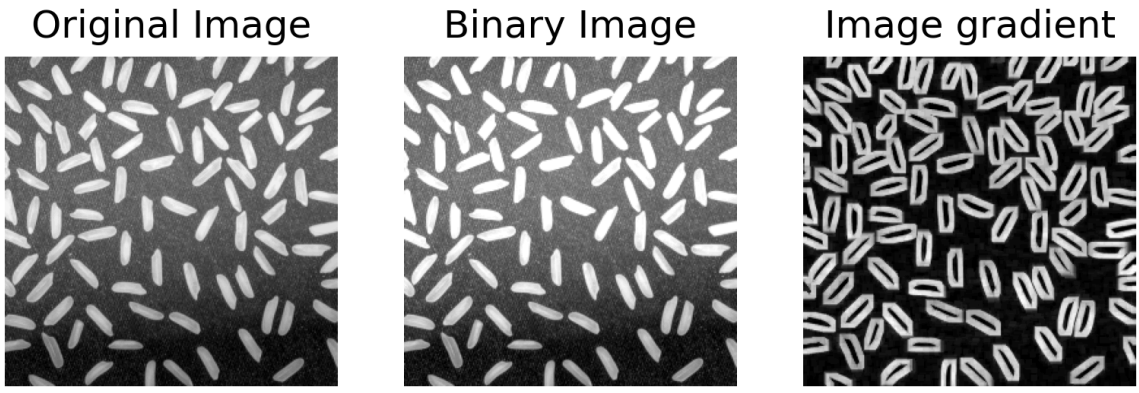
Out[ ]: Text(0.5, 1.0, 'Image Closing')



In [ ]:
```
gradient = cv2.morphologyEx(opening, cv2.MORPH_GRADIENT, kernel)
```

In [ ]:
```
fig = plt.figure(dpi=300)
plt.subplot(131)
plt.imshow(img,'gray')
plt.axis('off')
plt.title('Original Image')

plt.subplot(132)
plt.imshow(thresh,'gray')
plt.axis('off')
plt.title('Binary Image')

plt.subplot(133)
plt.imshow(gradient,'gray')
plt.axis('off')
plt.title('Image gradient')
```

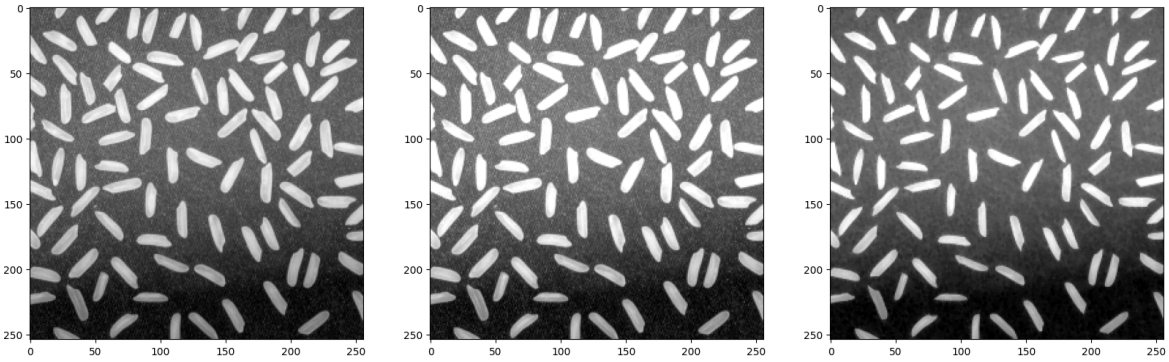
Out[ ]:
```
Text(0.5, 1.0, 'Image gradient')
```



In [ ]:
```
kernel = np.array((
 [0, 1, 0],
 [1, 1, 1],
 [0, 1, 0]), dtype="int")

output_image = cv2.morphologyEx(thresh, cv2.MORPH_HITMISS, kernel)

fig = plt.figure(figsize = (20,20))
plt.subplot(131)
plt.imshow(img,'gray')
plt.subplot(132)
plt.imshow(thresh,'gray')
```

```
plt.subplot(133)
plt.imshow(output_image,'gray')
```

Out[ ]: <matplotlib.image.AxesImage at 0x7b31605bed70>



In [ ]:
```
! jupyter nbconvert --to html Drishtti_Narwal_CV_Lab_Exp2.ipynb
```