223060A1012

Sr. No.	Title/Aim of The Practical	Page No.	Date	Signature
1	Write program to demonstrate the following aspects of signal processing on suitable data. 1. Upsampling and downsampling on Image/speech Signal. 2. Fast Fourier Transform to compute DFT.	1	16/02/2023	
2	Write program to perform Convolution and correlation of gray scale image.	5	16/02/2023	
3	Write program to perform the DFT of 4x4 Gray Scale Image.	7	16/02/2023	
4	Write program to implement point/pixel intensity transformations such as, 1. Log and Power-law transformations 2. Contrast adjustments 3. Histogram equalization 4. Thresholding, and halftoning operations	9	16/02/2023	
5	Write a program to apply various enhancements on images using image derivatives by implementing Gradient and Laplacian operations.	15	18/03/2023	
6	Write a program to apply various image enhancement using image derivatives by implementing smoothing, sharpening, and unsharp masking filters for generating suitable images for specific application requirements.	17	18/03/2023	
7	Write a program to Apply edge detection techniques such as Sobel and Canny to extract meaningful information from the given image samples.	19	13/04/2023	
8	Write the program to implement various morphological image processing techniques.	23	13/04/2023	
9	Image Segmentation.	25	13/04/2023	

16/2/23

DIP

Pract 1

Write program to demonstrate the following aspects of signal processing on data

1] Upsampling and downsampling

- Upsampling and downsampling are techniques used to change resolution of digital image and speech signals
- Upsampling involves incresing sample rate of signal by inserting new samples between existing samples
- Downsampling on other hand, involves reducing sample rate of signal by removing samples from signals

2] Fast Fourier Transform to compute DFT

- The fourier transform is mathematical technique used to analyze signals in the frequency domain
- The Discrete Fourier Transform (DFT) is specific type of Fourier transform used to analyze discrete signals such as digital image and speech signals
- The FFT takes advantage of periodicity of DFT of signal quickly and efficiency
- The FFT is commonly used in digital signal processing for application such as spectral analysis, filtering and compression.

16/2/23

Pract 2

Write program to perform convolution and correlation of grey scale image

- Convolution and correlation are two important operations in digital image processing that involves applying a kernel or filter to image
- Convolution involves taking small matrix called kernel or filter, and sliding it over image, computing sum of element wise products between kernel and corrosponding pixels in image
- Correlation on other hand is similer to convolution, but kernel is not flipped before sliding over image.
- Correlation can be used to perform tasks such as template matching and pattern recognition
- In grey scale image correlation, the kernel is typycally small matrix of values that represent pattern or feature to search for in image

16/2/23

Practical 3

Write program to perform DFT of 4x4 Grey scale image

- In digital image processing, the DFT of 4x4 greyscale image is mathematical operation that transform spatial domain representation of image into frequency domain representation
- The DFT of image is complex valued function that represents frequency components of image
- To compute DFT of 4x4 array greyscale image, the image is first divided into 16 equal sized blocks, each size 1x1. The DFT is then applied to each of this block individually, resulting in 4x4 array of complex values
- The DFT of grey scale image is useful in many digital image processing applications
- It allows us to analyze frequency content of an image and selectively remove or enhance certain frequency components to achive desired result.

16/2/23

Practical 4

Aim: Write program to implement point/pixel intensity transformed such as

1] Log and power law transform
- These transformation are used to adjust dynamic range of an image. Logarithmic transformation compress higher intensity values of an image, whereas power law transformation can either compress or expand intensity range of image

2] Contrast adjustments
- These transformation are used to adjust contrast of image. Contrast streaching increses difference between darkest and lighest part.

3] Histogram equalization
- This transformation is used to improve contrast and overall appearence of an image

4] Thresholding:
- These transformation are used to convert grey scale image into binary image.
- Thresholding involves setting threshold value and classifying all pixels with intensities above or below value

1&13123

Practical 5

Aim: Write program to apply various ehhencement of image using image derivatives by implementing Gradiant and laplacian

1] Gradient operation

- The gradient operation is used to compute first order derivative of image
- The gradient can be computed in both x and y direction using finite differences
- The magnetude ~~of~~ and direction of gradient can be used to detect edges and calculate feature in an image

2] Laplacian operation:

- The laplacian operation is used to compute second order derivative of an image
- The laplacian operator is 3X3 matrix that is applied to each pixel in image to compute second derivative.
- The laplacian is used to detect edges and to sharpen or smooth an image.

18131123

Practical 6:

Aim: Write program to apply various image enhancement using image derivatives by implementing smoothing, sharpning, and unsharp masking filters for generating suitable images for application

1] Smoothing filters:
- Smoothing filters are used to remove noise and blur an image by averaging intensity values of neighbouring pixels.
- The Gaussian filter can be implemented using derivatives by convolving image with gaussian kernel

2] Sharpning filters
- Sharpning filters are used to enhance edges and high frequency details in an image by incrising contrast between adjesent pixels

3] Unsharp masking filters
- Unsharp masking filters are used to enhance ~~edj~~ edges and details of image by subtracting smoothed version of image from orignal image.

13/4/23

Practical 7 DIP

Aim: Write program to apply edge detection techniques such as sobel and canny to extract meaningful information from given image samples

1] Sobel edge detector: The Sobel edge detector is simple and effective edge detection technique that computes gradient of image set using set of convolution kernels

- The sobel operator consist of two 3x3 kernels that are applied to image in both x and y direction

2] Canny edge detector :-

- The canny edge detector is more complex edge detection technique that is widely used in image processing applications.
- The canny detector works by first applying Gaussian filter to image to remove noise and then computing gradient of image using Sobel operator.
- The canny detector is powerful edge detection technique that is robust to noise and can detect edges with high accuracy

13/4/23

Practical 8

Aim: Write Program to implement various morphological image processing techinque

- Here are some morphological image processing technique

1] Erosion:
- Erosion is morphological operation that shrinks boundries of an object in image

2] Dialation:
Dialation is operation that expands boundries of an object in image

3] Opening:
opening is morphological operation consist of performing erosion followed by dialation

4] Closing
closing is morphological operation that consist of performing diletion following by errosion

5] Morphological Gradiant
- The morphological gradient that compute differences between dialation and erosion of image

6] Top Hat transform:
It subtracts opening of image from orignal image

13/4/23

Practical 9 Image Segmentation

— Image segmentation is fundamental task in digital image processing that involves dividing image into multiple regions or segments based on their visual characteristics

1] Thresholding:
- Thresholding involves setting threshold value and classifying pixels in image as foreground or background based on wheather intensity values are above or below

2] Region based segmentation:
- Region based segmentation involves grouping pixels into region based on visual similarity

3] Edge based segmentation
- Edge based segmentation involves detecting edges in image and grouping adjacent pixels into region based on edges.

4] Watershed segmentation:
- Watershed segmentation is technique that involves treating intensity values in image as topographical surface and then flooding surface with water

Practical No: 1 Ninad Karlekar 22306A1012 Date: 16/02/2023

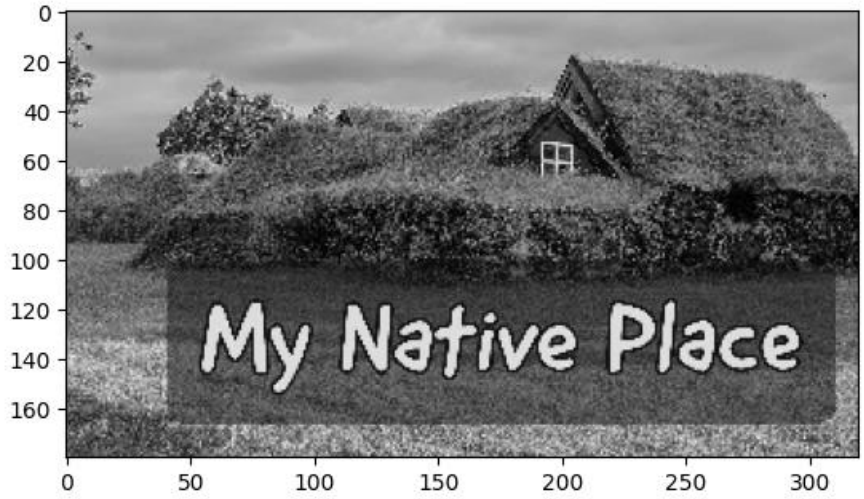
Aim: Write program to demonstrate the following aspects of signal processing on suitable data.

Practical 1 a)Upsampling and downsampling on Image/speech Signal.

Code:

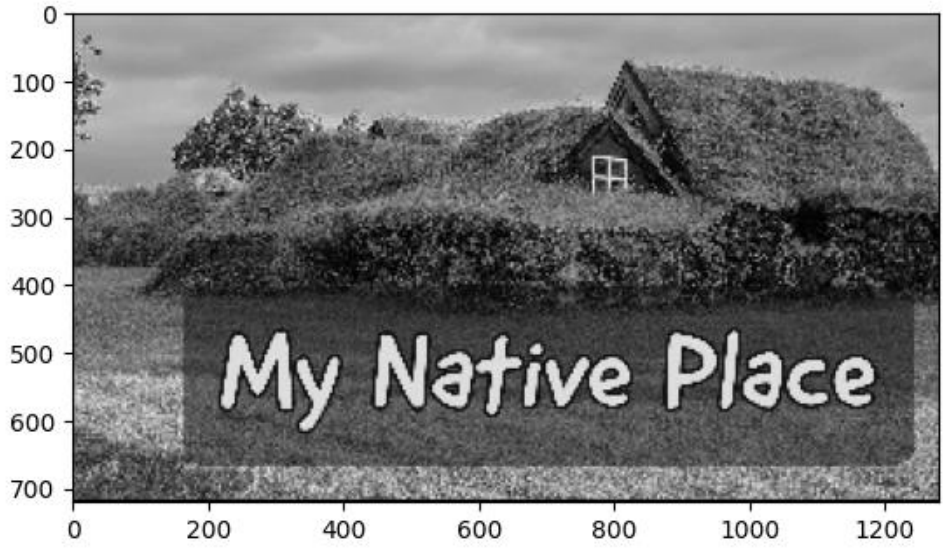
```
import os
os.sys.path
import cv2
import matplotlib.pyplot as plt
import numpy as np

img1 = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/nativeplace.jpg', 0)
[m, n] = img1.shape
print('Original Image Shape:', m, n)
print('Original Image:')
plt.imshow(img1, cmap="gray")
f = 4
img2 = np.zeros((m//f, n//f), dtype=int)
for i in range(0, m, f):
 for j in range(0, n, f):
  try:
   img2[i//f][j//f] = img1[i][j]
  except IndexError:
   pass

[a, b] = img2.shape
print('Down Sampled Image Shape:', a, b)
print("-----------------------")
print('Down Sampled Image:')
plt.imshow(img2, cmap="gray")
```

```
Original Image Shape: 720 1280
Original Image:
Down Sampled Image Shape: 180 320
-----------------------
Down Sampled Image:
```




```
# Upsampling
img3 = np.zeros((m, n), dtype=int)

for i in range(0, m-1, f):
    for j in range(0, n-1, f):
        try:
            img3[i, j] = img2[i//f][j//f]
        except IndexError:
            pass

for i in range(1, m-(f-1), f):
    for j in range(0, n-(f-1)):
        img3[i:i+(f-1), j] = img3[i-1, j]

for i in range(0, m-1):
    for j in range(1, n-1, f):
        img3[i, j:j+(f-1)] = img3[i, j-1]

[c, d] = img3.shape

print('Original Image Shape:', m, n)
print("-----------------------")
print('Down Sampled Image Shape:', a, b)
print("-----------------------")
print('UP Sampled Image Shape:', c, d)
print("-----------------------")

print('Up Sampled Image:')
plt.imshow(img3, cmap="gray")
```

```
Original Image Shape: 720 1280
-----------------------
Down Sampled Image Shape: 180 320
-----------------------
UP Sampled Image Shape: 720 1280
-----------------------
Up Sampled Image:
```



Aim: 2. Fast Fourier Transform to compute DFT.

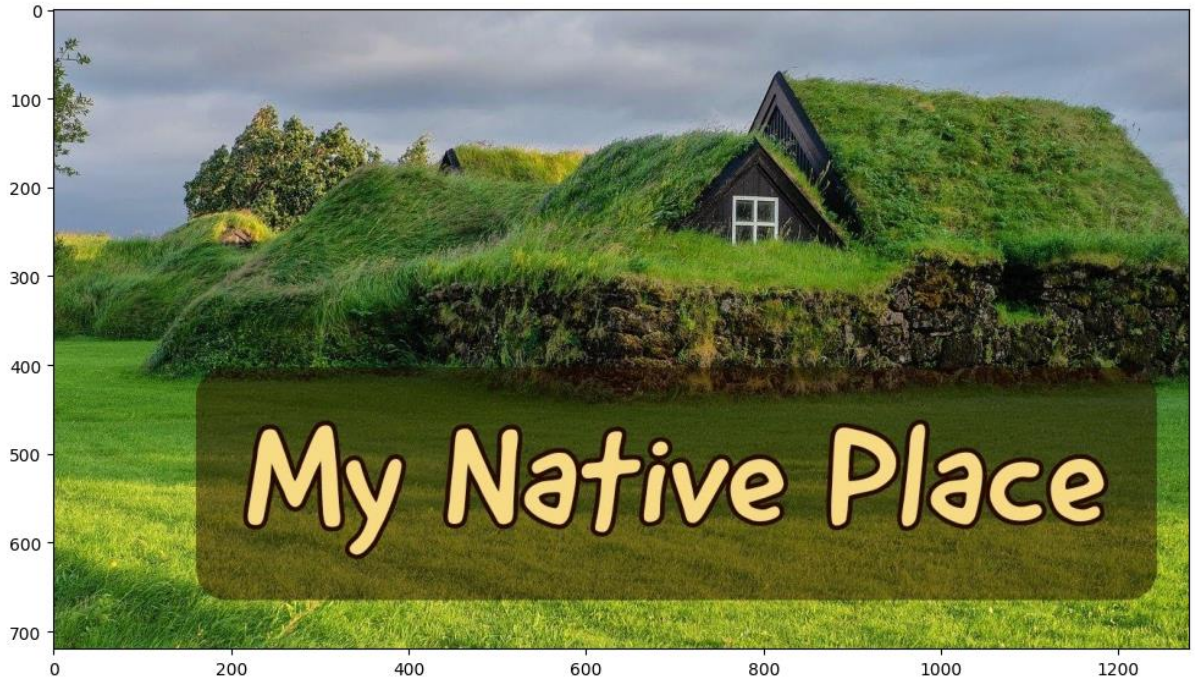
CODE:

```
import os
os.sys.path
import cv2
import matplotlib.pyplot as plt
import numpy as np
# scipy.stats.signaltonoise() was deprecated in scipy 0.16.0 and removed in 1.0.0.
import numpy as np
def signaltonoise(a, axis=0, ddof=0):
    a = np.asanyarray(a)
    m = a.mean(axis)
    sd = a.std(axis=axis, ddof=ddof)
    return np.where(sd == 0, 0, m/sd)
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

im = np.array(Image.open('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/elephant.jpg').convert('L'))
freq = np.fft.fft2(im)
im1 = np.fft.ifft2(freq).real
snr = signaltonoise(im1, axis=None)

print('SNR for the image obtained after reconstruction = ' + str(snr))
assert(np.allclose(im, im1))

plt.figure(figsize=(20, 10))
plt.subplot(121), plt.imshow(im, cmap='gray'), plt.axis('off')
plt.title('Original Image', size=20)
plt.subplot(122), plt.imshow(im1, cmap='gray'), plt.axis('off')
plt.title('Image Obtained after Reconstruction', size=20)
plt.show()
```

Practical No: 2 Ninad Karlekar 22306A1012 Date: 16/02/2023

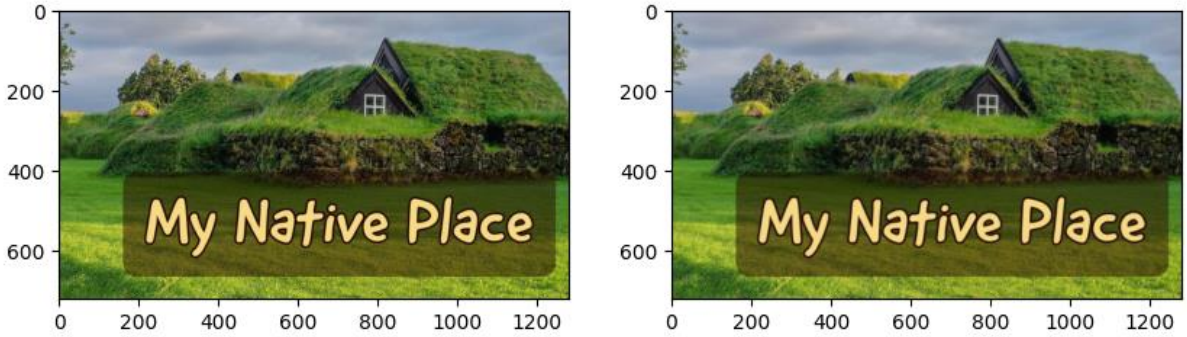
Aim: Write program to perform Convolution and correlation of gray scale image.

```
# Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester 2/ImageProcessing/Dataset/nativeplace.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
fig, ax = plt.subplots(1, figsize=(12,8))
plt.imshow(image)
```

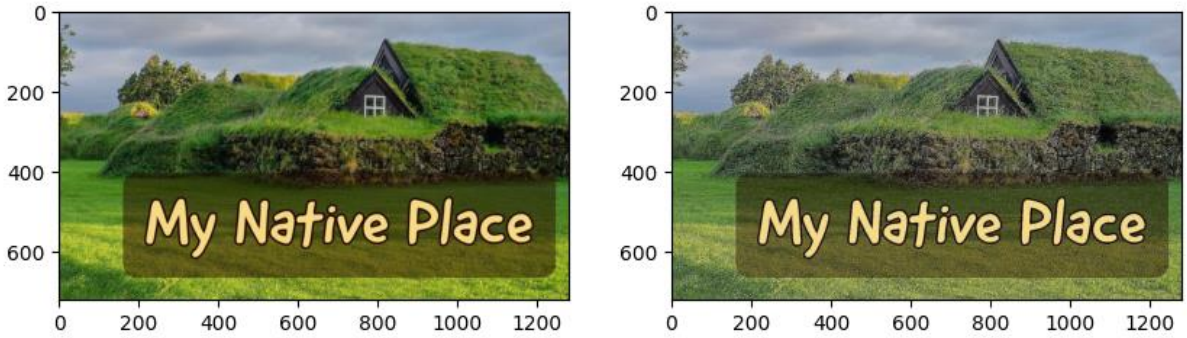



```
abc=np.ones((3,3))
kernel = np.ones((3, 3), np.float32) / 9
img = cv2.filter2D(image, -1, kernel)
```

```
fig, ax = plt.subplots(1,2,figsize=(10,6))
ax[0].imshow(image)
ax[1].imshow(img)
```




```
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
img = cv2.filter2D(image, -1, kernel)
fig, ax = plt.subplots(1,2,figsize=(10,6))
ax[0].imshow(image)
ax[1].imshow(img)
```



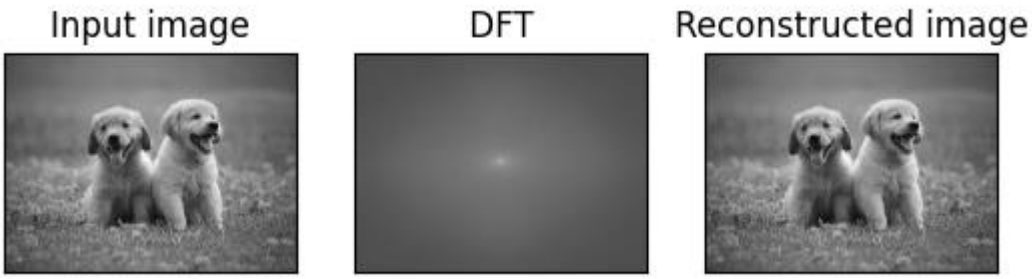
Practical No: 3 Ninad Karlekar 22306A1012 Date: 16/02/2023

Aim: Write program to perform the DFT of 4x4 Gray Scale Image.

CODE:

```
#importing packages
import numpy as np
import cv2
from matplotlib import pyplot as plt

#getting the input image and convert to grayscale
img = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester 2/ImageProcessing/Dataset/Dog.jpg', 0)

# Transform the image to improve the speed in the Fourier transform calculation
rows, cols = img.shape
optimalRows = cv2.getOptimalDFTSize(rows)
optimalCols = cv2.getOptimalDFTSize(cols)
optimalImg = np.zeros((optimalRows, optimalCols))
optimalImg[:rows, :cols] = img

# Calculate the discrete Fourier transform
dft = cv2.dft(np.float32(optimalImg), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

# output of CV2.dft() function will be 3-D numpy array, for 2-D Output, 2D DFT as two-part complex and real part.
f_complex = dft_shift[:, :, 0] + 1j * dft_shift[:, :, 1]
f_abs = np.abs(f_complex) + 1 # lie between 1 and 1e6
f_bounded = 20 * np.log(f_abs)
f_img = 255 * f_bounded / np.max(f_bounded)
f_img = f_img.astype(np.uint8)

# Reconstruct the image using the inverse Fourier transform
```

```
i_shift = np.fft.ifftshift(dft_shift)
result = cv2.idft(i_shift)
result = cv2.magnitude(result[:, :, 0], result[:, :, 1])

# #Displaying input image, grayscale image, DFT of the Input Image
images = [optimalImg, f_img, result]
imageTitles = ['Input image', ' DFT ', 'Reconstructed image']

for i in range(len(images)):
    plt.subplot(1, 3, i + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title(imageTitles[i])
    plt.xticks([])
    plt.yticks([])
plt.show()
# for hold the Display until key press
cv2.waitKey()
cv2.destroyAllWindows()
```

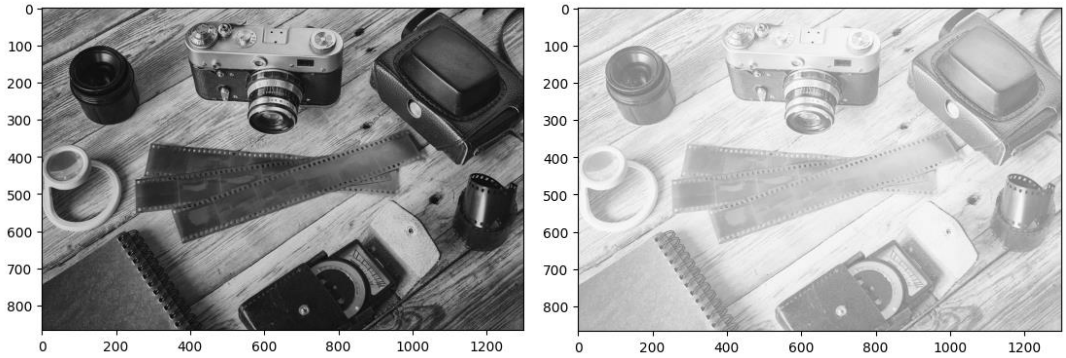


Practical No: 4 Ninad Karlekar 22306A1012 Date: 16/02/2023

Aim: Write program to implement point/pixel intensity transformations such as,
1. Log and Power-law transformations

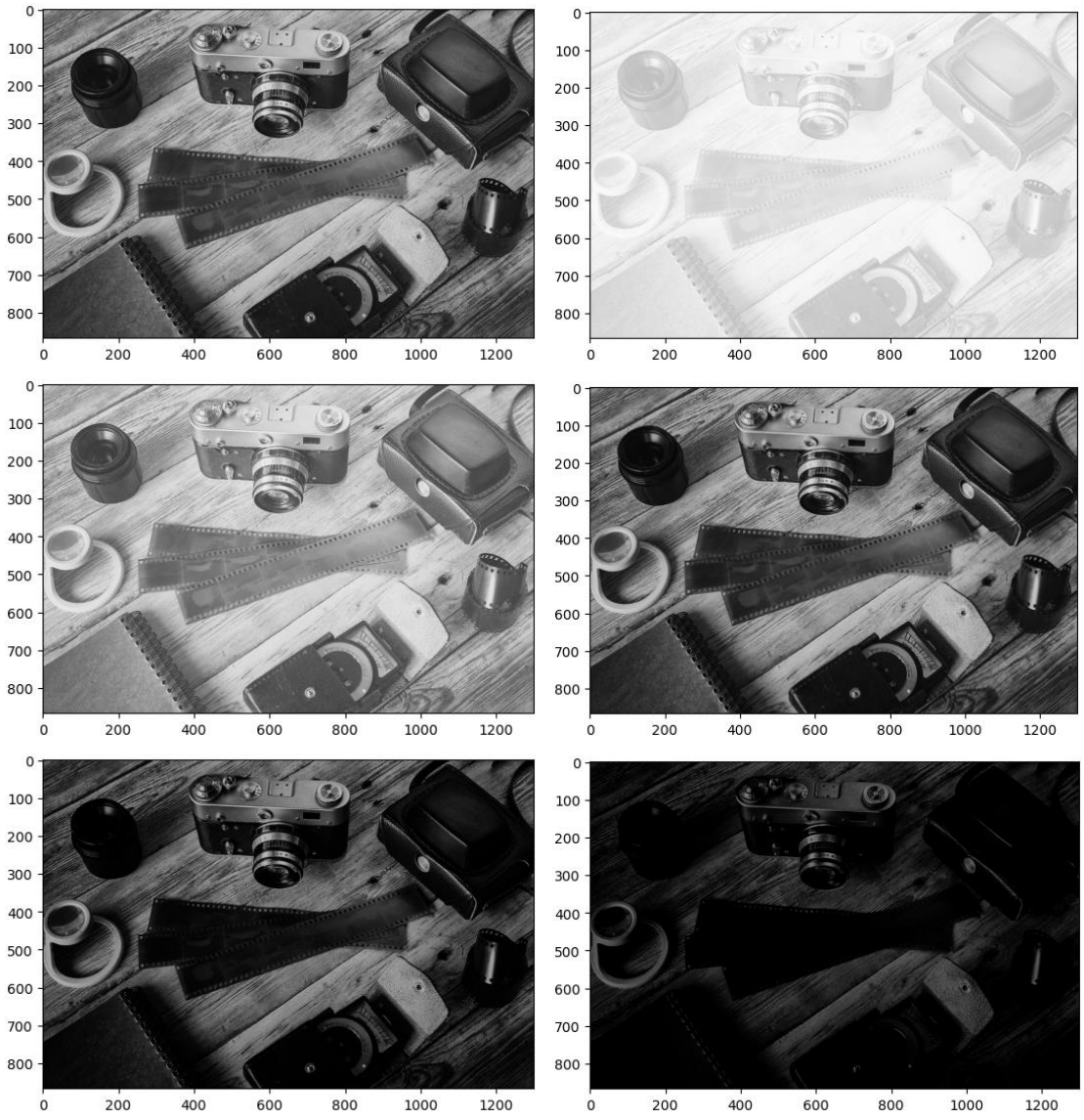
CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Open the image.
img = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sample.jpg')

# Apply log transform.
c = 255/(np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)

# Specify the data type.
log_transformed = np.array(log_transformed, dtype = np.uint8)

# Save the output.
cv2.imwrite('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/log_transformed.jpg', log_transformed)

plt.imshow(img)
plt.show()
plt.imshow(log_transformed)
plt.show()
```




```
import cv2
import numpy as np

# Open the image.
img = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sample.jpg')
plt.imshow(img)
plt.show()
# Trying 4 gamma values.
for gamma in [0.1, 0.5, 1.2, 2.2,5]:
```

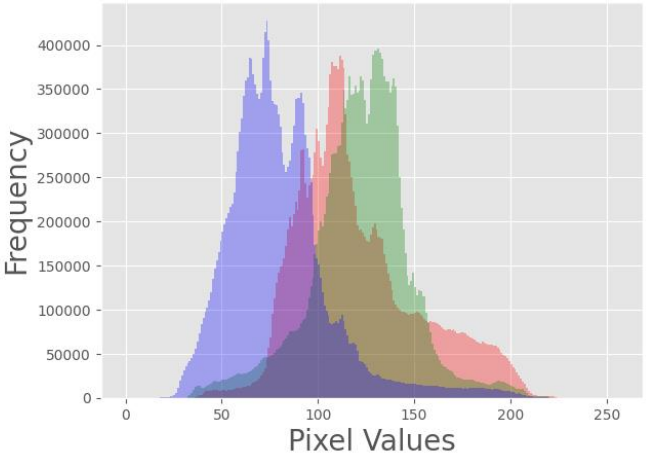
```
    # Apply gamma correction.
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')

    # Save edited images.
    cv2.imwrite('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/gamma_transformed'+str(gamma)+'.jpg', gamma_corrected)

    plt.imshow(gamma_corrected)
    plt.show()
```

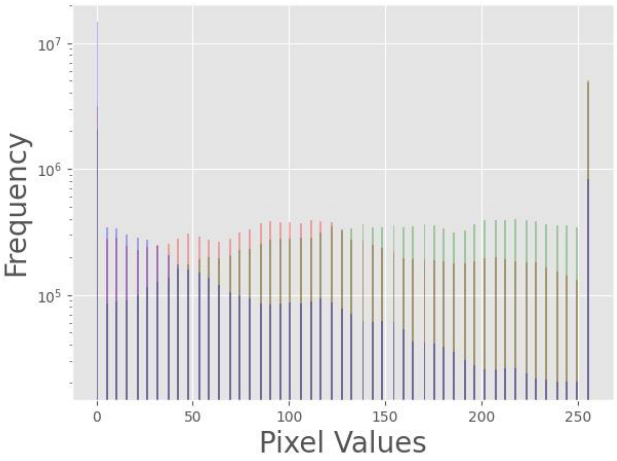


2. Contrast adjustments

```
import numpy as np
from skimage.io import imread
from skimage.color import rgb2gray
from skimage import data, img_as_float, img_as_ubyte, exposure, io, color
from PIL import Image, ImageEnhance, ImageFilter
from scipy import ndimage, misc
import matplotlib.pyplot as pylab
import cv2

def plot_image(image, title=""):
  pylab.title(title, size=10)
  pylab.imshow(image)
  pylab.axis('off')

def plot_hist(r,g,b,title=""):
   r,g,b=img_as_ubyte(r),img_as_ubyte(g),img_as_ubyte(b)
   pylab.hist(np.array(r).ravel(),bins=256, range=(0,256),color='r',alpha=0.3)
   pylab.hist(np.array(g).ravel(),bins=256, range=(0,256),color='g',alpha=0.3)
   pylab.hist(np.array(b).ravel(),bins=256, range=(0,256),color='b',alpha=0.3)
   pylab.xlabel('Pixel Values', size=20)
   pylab.ylabel('Frequency',size=20)
   pylab.title(title,size=10)

im=Image.open('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/Dog.jpg')
im_r,im_g,im_b=im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(im)
pylab.subplot(122)
plot_hist(im_r,im_g,im_b)
pylab.show()
def contrast(c):
  return 0 if c<50 else (255 if c>150 else int((255*c-22950)/48))


imc=im.point(contrast)
im_rc,im_gc,im_bc=imc.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(imc)
pylab.subplot(122)
plot_hist(im_rc,im_gc,im_bc)
pylab.yscale('log')
pylab.show()
```

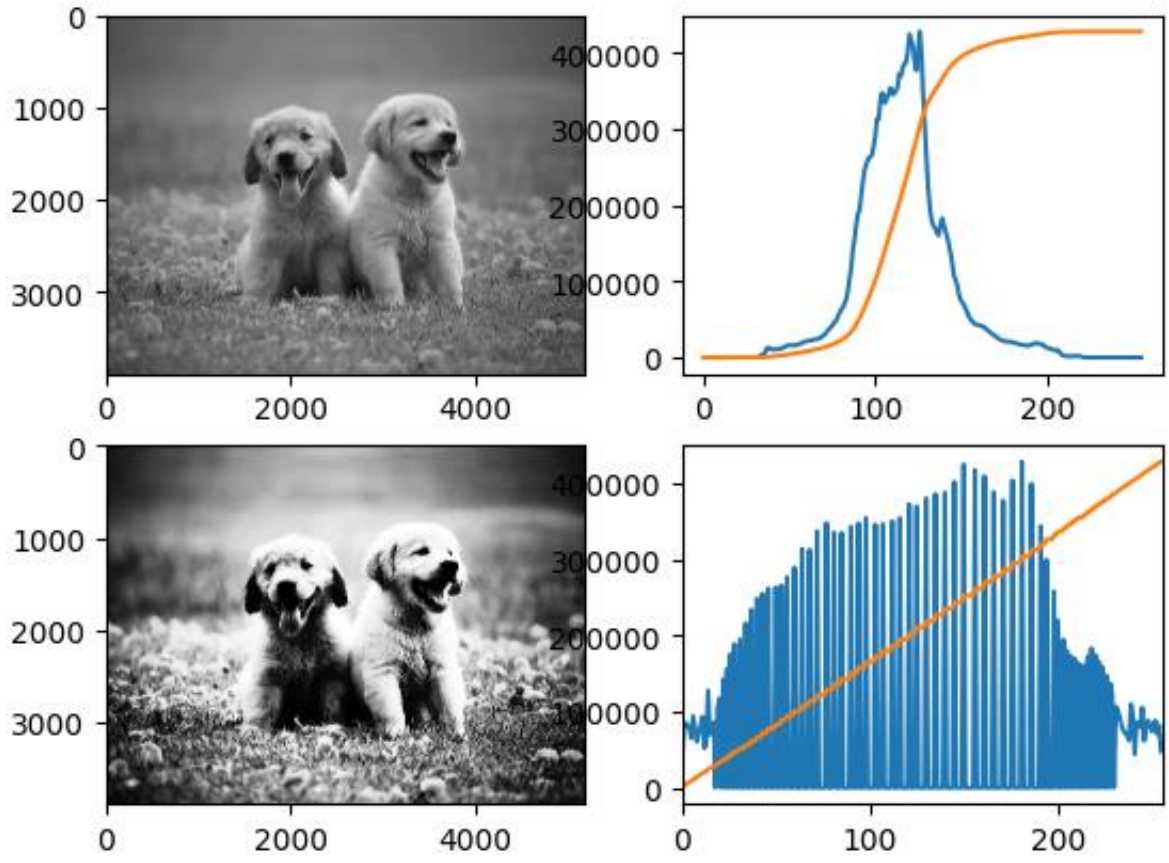
OUTPUT:


Frequency
400000
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
Pixel Values


Frequency
10^7
10^6
10^5
0
50
100
150
200
250
Pixel Values

3. Histogram equalization

```
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/Dog.jpg',0)

hist = cv2.calcHist([img],[0],None,[256],[0,256])
eq = cv2.equalizeHist(img)
cdf = hist.cumsum()
cdfnmhist = cdf * hist.max()/ cdf.max()
histeq = cv2.calcHist([eq],[0],None,[256],[0,256])
cdfeq = histeq.cumsum()
cdfnmhisteq = cdfeq * histeq.max()/ cdf.max()
plt.subplot(221), plt.imshow(img,'gray')
plt.subplot(222), plt.plot(hist), plt.plot(cdfnmhist)
plt.subplot(223), plt.imshow(eq,'gray')
plt.subplot(224), plt.plot(histeq), plt.plot(cdfnmhisteq)
plt.xlim([0,256])
```

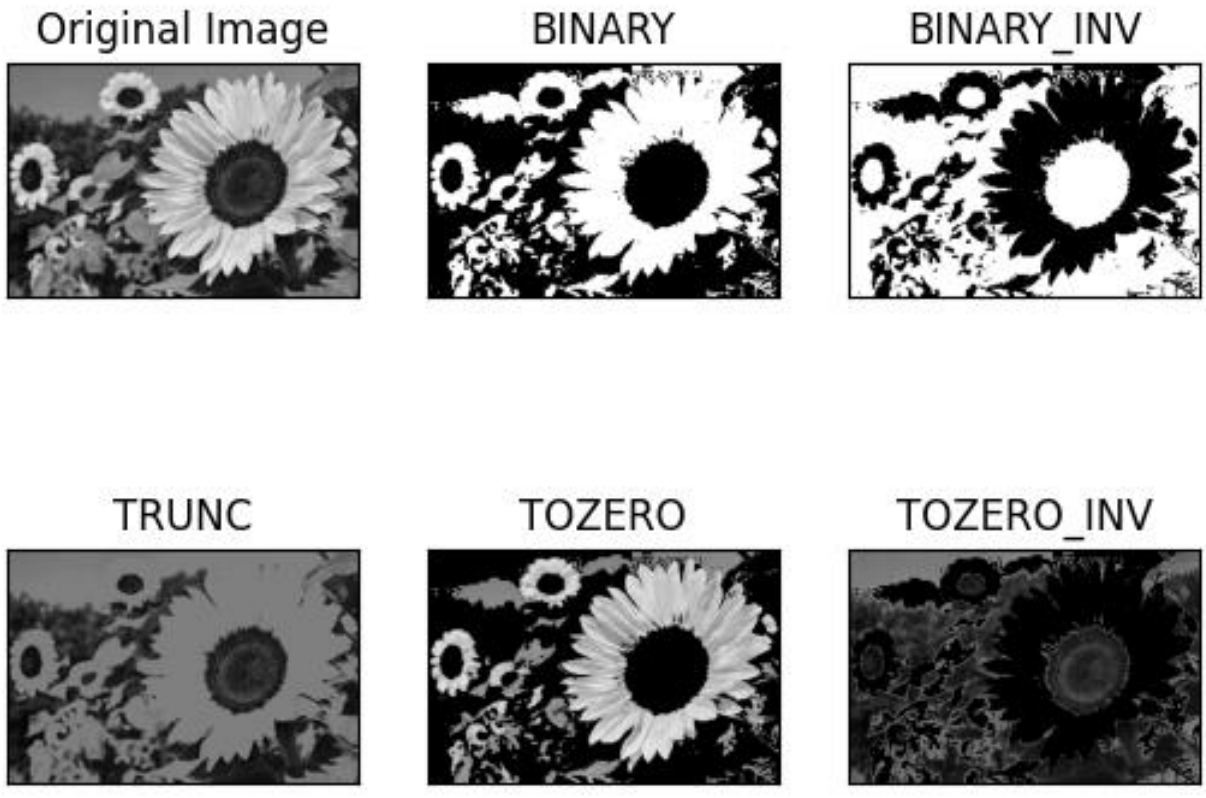


4. Thresholding

CODE:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sunflower.jpg',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

OUTPUT:



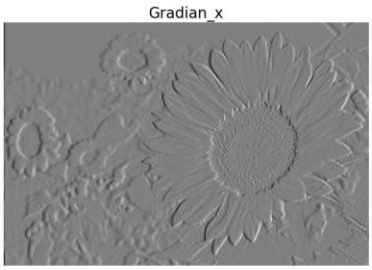
Practical No: 5 Ninad Karlekar 22306A1012 Date: 18/03/2023

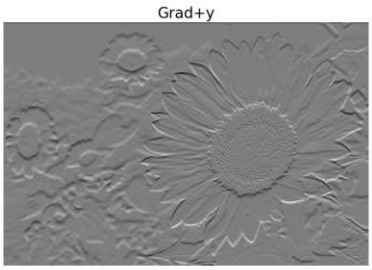
AIM: Practical 5: Write a program to apply various enhancements on images using image derivatives by implementing Gradient and Laplacian operations.

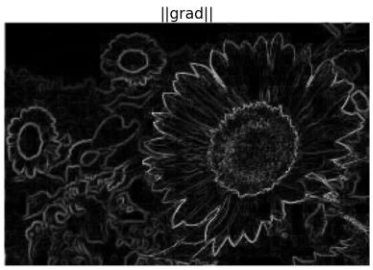
```
import numpy as np
from scipy import signal, misc, ndimage
from skimage import filters, feature, img_as_float
from skimage.io import imread
from skimage.color import rgb2gray
from PIL import Image, ImageFilter
import matplotlib.pylab as pylab
from skimage.transform import rescale

def plot_image(image, title=""):
 pylab.title(title, size=20),
 pylab.imshow(image)
 pylab.axis('off')
def plot_hist(r,g,b,title=""):
   r,g,b=img_as_ubyte(r),img_as_ubyte(g),img_as_ubyte(b)
   pylab.hist(np.array(r).ravel(),bins=256, range=(0,256),color='r',alpha=0.3)
   pylab.hist(np.array(g).ravel(),bins=256, range=(0,256),color='g',alpha=0.3)
   pylab.hist(np.array(b).ravel(),bins=256, range=(0,256),color='b',alpha=0.3)
   pylab.xlabel('Pixel Values', size=20)
   pylab.ylabel('Frequency',size=20)
   pylab.title(title,size=10)
ker_x=[[-1,1]]
ker_y=[[-1],[1]]
im=rgb2gray(imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sunflower.jpg'))
im_x=signal.convolve2d(im,ker_x,mode='same')
im_y=signal.convolve2d(im,ker_y,mode='same')
im_mag=np.sqrt(im_x**2+im_y**2)
im_dir=np.arctan(im_y/im_x)
pylab.gray()
pylab.figure(figsize=(30,20))
pylab.subplot(231)
plot_image(im,'Original')
pylab.subplot(232)
plot_image(im_x,'Gradian_x')
pylab.subplot(233)
plot_image(im_y,'Grad+y')
pylab.subplot(234)
plot_image(im_mag,'||grad||')
pylab.subplot(235)
plot_image(im_dir, r'$\theta$')
pylab.subplot(236)
pylab.plot(range(im.shape[1]), im[0,:], 'b-', label=r'$f(x,y)|_{x=0}$', linewidth=5)
pylab.plot(range(im.shape[1]), im_x[0,:], 'r-', label=r'$grad_x (f(x,y))|_{x=0}$')
pylab.title(r'$grad_x (f(x,y))|_{x=0}$',size=30)
pylab.legend(prop={'size':20})
```

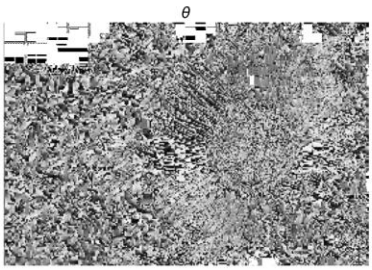
pylab.show()

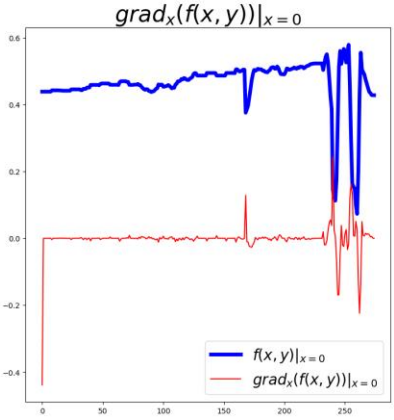
OUTPUT:











Practical No: 6 Ninad Karlekar 22306A1012 Date: 18/03/2023

AIM: Practical 6:Write a program to apply various image enhancement using image derivatives by implementing smoothing, sharpening, and unsharp masking filters for generating suitable images for specific application requirements.

CODE:

```
import numpy as np
from scipy import signal, misc, ndimage
from skimage import filters, feature, img_as_float
from skimage.io import imread
from skimage.color import rgb2gray
from PIL import Image, ImageFilter
import matplotlib.pylab as pylab
from skimage.transform import rescale

def plot_hist(r,g,b,title=""):
   r,g,b=img_as_ubyte(r),img_as_ubyte(g),img_as_ubyte(b)
   pylab.hist(np.array(r).ravel(),bins=256, range=(0,256),color='r',alpha=0.3)
   pylab.hist(np.array(g).ravel(),bins=256, range=(0,256),color='g',alpha=0.3)
   pylab.hist(np.array(b).ravel(),bins=256, range=(0,256),color='b',alpha=0.3)
   pylab.xlabel('Pixel Values', size=20)
   pylab.ylabel('Frequency',size=20)
   pylab.title(title,size=10)

def plot_image(image, title=""):
  pylab.title(title, size=10)
  pylab.imshow(image)
  pylab.axis('off')

# sharpening of images
from skimage.filters import laplace
im=rgb2gray(imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sunflower.jpg'))
im1=np.clip(laplace(im)+im,0,1)
pylab.figure(figsize=(10,15))
pylab.subplot(121), plot_image(im, 'Original Image')
pylab.subplot(122), plot_image(im1,'Sharpened Image')
pylab.tight_layout()
pylab.show()
```

OUTPUT:

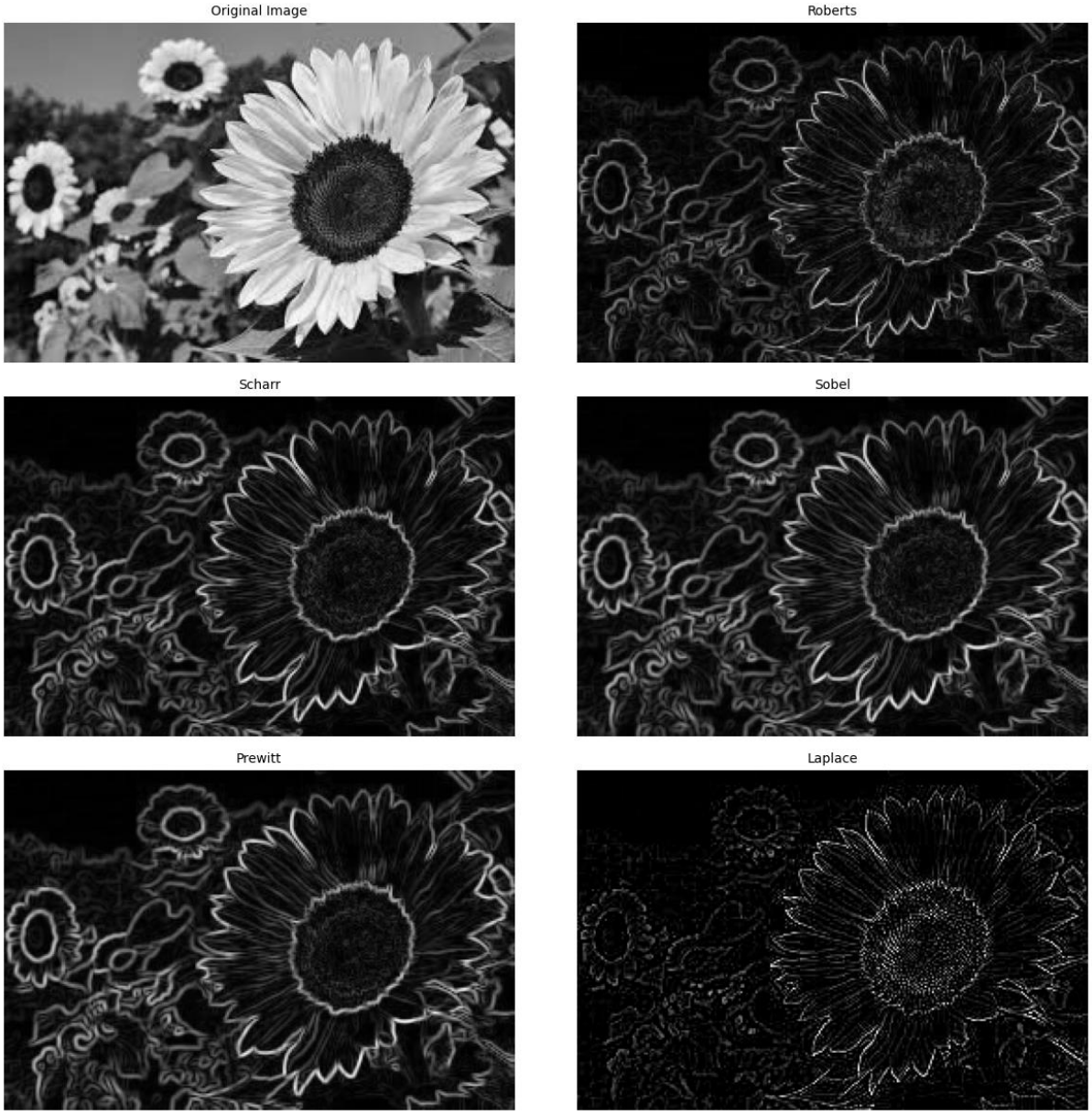
Practical No: 7 Ninad Karlekar 22306A1012 Date: 13/04/2023

Aim: Write a program to Apply edge detection techniques such as Sobel and Canny to extract meaningful information from the given image samples.

CODE:

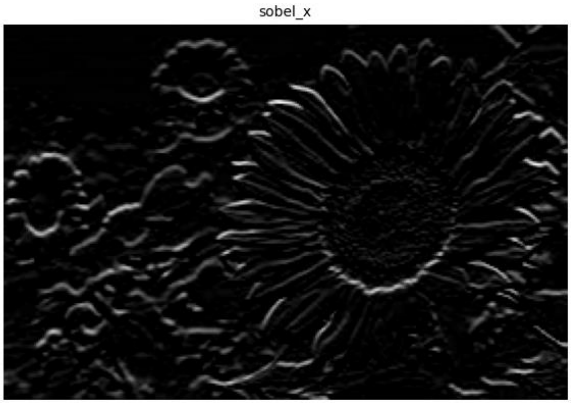
```
import numpy as np
from scipy import signal, misc, ndimage
from skimage import filters, feature, img_as_float
from skimage.io import imread
from skimage.color import rgb2gray
from PIL import Image, ImageFilter
import matplotlib.pylab as pylab
from skimage.transform import rescale

def plot_image(image, title=""):
  pylab.title(title, size=10)
  pylab.imshow(image)
  pylab.axis('off')

def plot_hist(r,g,b,title=""):
   r,g,b=img_as_ubyte(r),img_as_ubyte(g),img_as_ubyte(b)
   pylab.hist(np.array(r).ravel(),bins=256, range=(0,256),color='r',alpha=0.3)
   pylab.hist(np.array(g).ravel(),bins=256, range=(0,256),color='g',alpha=0.3)
   pylab.hist(np.array(b).ravel(),bins=256, range=(0,256),color='b',alpha=0.3)
   pylab.xlabel('Pixel Values', size=20)
   pylab.ylabel('Frequency',size=20)
   pylab.title(title,size=10)

# Edge Detectors with scikit-image-Prewitt, roberts, sobel, scharr, laplace
im=Image.open('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sunflower.jpg').convert('L')
im = img_as_float(im)  # convert to floating point dtype
pylab.gray()
pylab.figure(figsize=(15,15))
pylab.subplot(3,2,1), plot_image(im,'Original Image')
edges=filters.roberts(im)
pylab.subplot(3,2,2), plot_image(edges,'Roberts')

edges=filters.scharr(im)
pylab.subplot(3,2,3), plot_image(edges,'Scharr')

edges=filters.sobel(im)
pylab.subplot(3,2,4), plot_image(edges,'Sobel')

edges=filters.prewitt(im)
pylab.subplot(3,2,5), plot_image(edges,'Prewitt')

edges=np.clip(filters.laplace(im), 0,1)
```

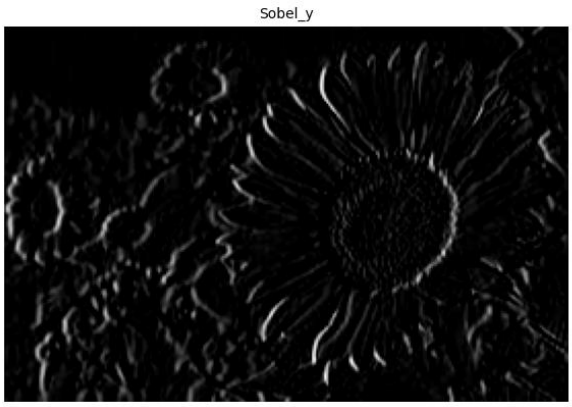
```
pylab.subplot(3,2,6), plot_image(edges,'Laplace')
pylab.subplots_adjust(wspace=0.1,hspace=0.1)
pylab.show()
```




```
#SOBEL
im = Image.open('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/sunflower.jpg').convert('L')
im_array = np.array(im)
pylab.gray()
pylab.figure(figsize=(15,15))
pylab.subplot(2,2,1), plot_image(im,'Original')
pylab.subplot(2,2,2)
edges_x=filters.sobel_h(im_array)
plot_image(np.clip(edges_x,0,1),'sobel_x')

pylab.subplot(2,2,3)
```

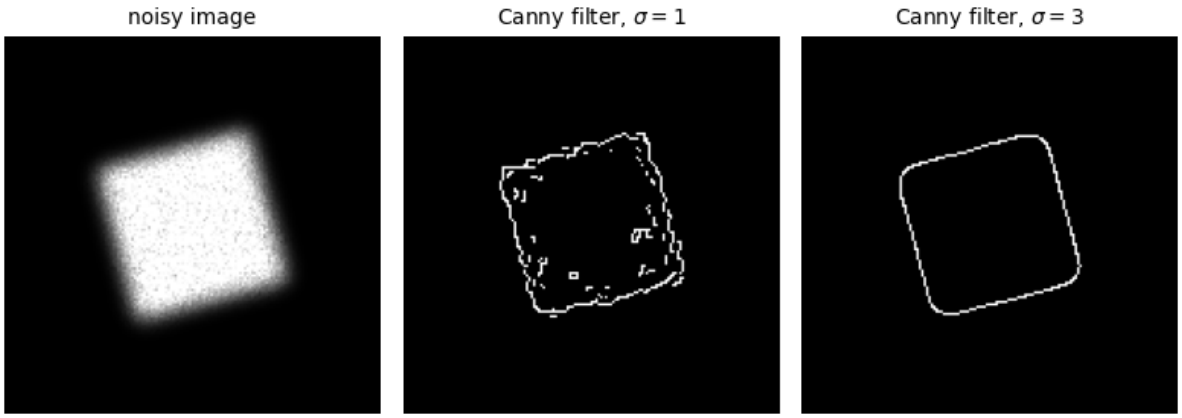
```
edges_y=filters.sobel_v(im_array)
plot_image(np.clip(edges_y,0,1),'Sobel_y')

pylab.subplot(2,2,4)
edges=filters.sobel(im_array)
plot_image(np.clip(edges,0,1),'Sobel')

pylab.subplots_adjust(wspace=0.1,hspace=0.1)
pylab.show()
```






```
#CANNY
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from skimage.util import random_noise
from skimage import feature

# Generate noisy image of a square
image = np.zeros((128, 128), dtype=float)
image[32:-32, 32:-32] = 1

image = ndi.rotate(image, 15, mode='constant')
image = ndi.gaussian_filter(image, 4)
image = random_noise(image, mode='speckle', mean=0.05)
```

```
# Compute the Canny filter for two values of sigma
edges1 = feature.canny(image)
edges2 = feature.canny(image, sigma=3)

# display results
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(8, 3))

ax[0].imshow(image, cmap='gray')
ax[0].set_title('noisy image', fontsize=10)

ax[1].imshow(edges1, cmap='gray')
ax[1].set_title(r'Canny filter, $\sigma=1$', fontsize=10)

ax[2].imshow(edges2, cmap='gray')
ax[2].set_title(r'Canny filter, $\sigma=3$', fontsize=10)

for a in ax:
    a.axis('off')
fig.tight_layout()
plt.show()
```



Practical No: 8 Ninad Karlekar 22306A1012 Date: 13/04/2023

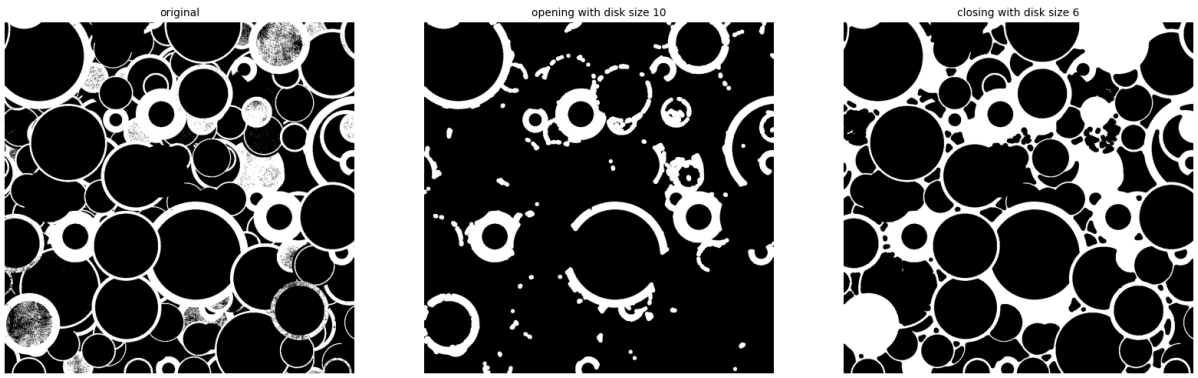
Aim: Write the program to implement various morphological image processing techniques.

CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
# For colab
# from google.colab.patches import cv2_imshow
%matplotlib inline
from matplotlib import pyplot as plt
img = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/411525.jpg', 0)
ret, bw_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
# converting to its binary form
bw = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((5, 5), np.uint8)
img_erosion = cv2.erode(img, kernel,iterations=1)
img_dilation = cv2.dilate(img, kernel,iterations=1)
plt.figure(figsize=(5,5))
plt.imshow(img,cmap="gray")
plt.axis('off')
plt.title("ORIGINAL IMAGE")
plt.show()
plt.figure(figsize=(5,5))
plt.imshow(img_erosion,)
plt.axis('off')
plt.title("EROSION")
plt.show()
plt.figure(figsize=(5,5))
plt.imshow(img_dilation,cmap="gray")
plt.axis('off')
plt.title("DILATION")
plt.show()
cv2.waitKey(0)
```

DILATION

```
#Image opening and closing
from skimage.morphology import binary_opening, binary_closing, binary_erosion, binary_dilation, disk
from skimage.color import rgb2gray
from skimage.io import imread
from PIL import Image, ImageFilter
import matplotlib.pylab as pylab
from skimage.transform import rescale
import numpy as np
from scipy import signal, misc, ndimage
from skimage import filters, feature, img_as_float
def plot_image(image, title=""):
  pylab.title(title, size=10)
  pylab.imshow(image)
  pylab.axis('off')
im = rgb2gray(imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/circles1.jpg'))
im[im <= 0.5] = 0
im[im > 0.5] = 1
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(1,3,1), plot_image(im, 'original')
im1 = binary_opening(im, disk(6))
pylab.subplot(1,3,2), plot_image(im1, 'opening with disk size ' + str(10))
im1 = binary_closing(im, disk(6))
pylab.subplot(1,3,3), plot_image(im1, 'closing with disk size ' + str(6))
pylab.show()
```



Practical No: 9 Ninad Karlekar 22306A1012 Date: 13/04/2023

Aim: Image Segmentation.

Code:

```
#Loading original image
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('F:/GitHub/Practical_BscIT_MscIT_Ninad/MscIT/Semester
2/ImageProcessing/Dataset/Original_img_segmentation.png')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.imshow(img,cmap="gray")
plt.axis('off')
plt.title("Original Image")
plt.show()
```


```
#Converting to gray scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(8,8))
plt.imshow(gray,cmap="gray")
plt.axis('off')
plt.title("GrayScale Image")
plt.show()
```

```
#Converting to binary inverted image
ret, thresh = cv2.threshold(gray, 0, 255,cv2.THRESH_BINARY_INV +cv2.THRESH_OTSU)
plt.figure(figsize=(8,8))
plt.imshow(thresh,cmap="gray")
plt.axis('off')
plt.title("Threshold Image")
plt.show()
```


```
#Segmenting the images
kernel = np.ones((3, 3), np.uint8)
closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE,kernel, iterations = 15)
bg = cv2.dilate(closing, kernel, iterations = 1)
dist_transform = cv2.distanceTransform(closing, cv2.DIST_L2, 0)
ret, fg = cv2.threshold(dist_transform, 0.02*dist_transform.max(), 255, 0)
#cv2.imshow('image', fg)
plt.figure(figsize=(8,8))
plt.imshow(fg,cmap="gray")
plt.axis('off')
plt.title("Segmented Image")
plt.show()
```


```
#Final code
plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.axis('off')
```

```
plt.title("Original Image")
plt.imshow(img,cmap="gray")
plt.subplot(2,2,2)
plt.imshow(gray,cmap="gray")
plt.axis('off')
plt.title("GrayScale Image")
plt.subplot(2,2,3)
plt.imshow(thresh,cmap="gray")
plt.axis('off')
plt.title("Threshold Image")
plt.subplot(2,2,4)
plt.imshow(fg,cmap="gray")
plt.axis('off')
plt.title("Segmented Image")
plt.show()
```

OUTPUT: