UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

M.Sc. Computer Science – Semester I

APPLIED SIGNAL AND IMAGE PROCESSING

JOURNAL

2022-2023

Seat No. _____

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the work entered in this journal was done in the University Department of Computer Science laboratory by Mr./Ms._____Seat No. _____ for the course of M.Sc. Computer Science - Semester I (CBCS) (Revised) during the academic year 2022- 2023 in a satisfactory manner.

_____ _____

Subject In-charge **Head of Department**

_____

External Examiner

Index

PRACTICAL-01

AIM :- Write program to demonstrate the following aspects ofsignal processing on suitable data

1. Upsampling and downsampling on Image/speechsignal.
2. Fast Fourier Transform to compute DFT

Theory:

THEORY:-

Down-sampling

In the down-sampling technique, the number of pixels in the given image is reduced depending on the sampling frequency. Due to this, the resolution and size of the image decrease.

Up-sampling

The number of pixels in the down-sampled image can be increased by using up-sampling interpolation techniques. The up-sampling technique increases the resolution as well as the size of the image.

Fast Fourier Transform

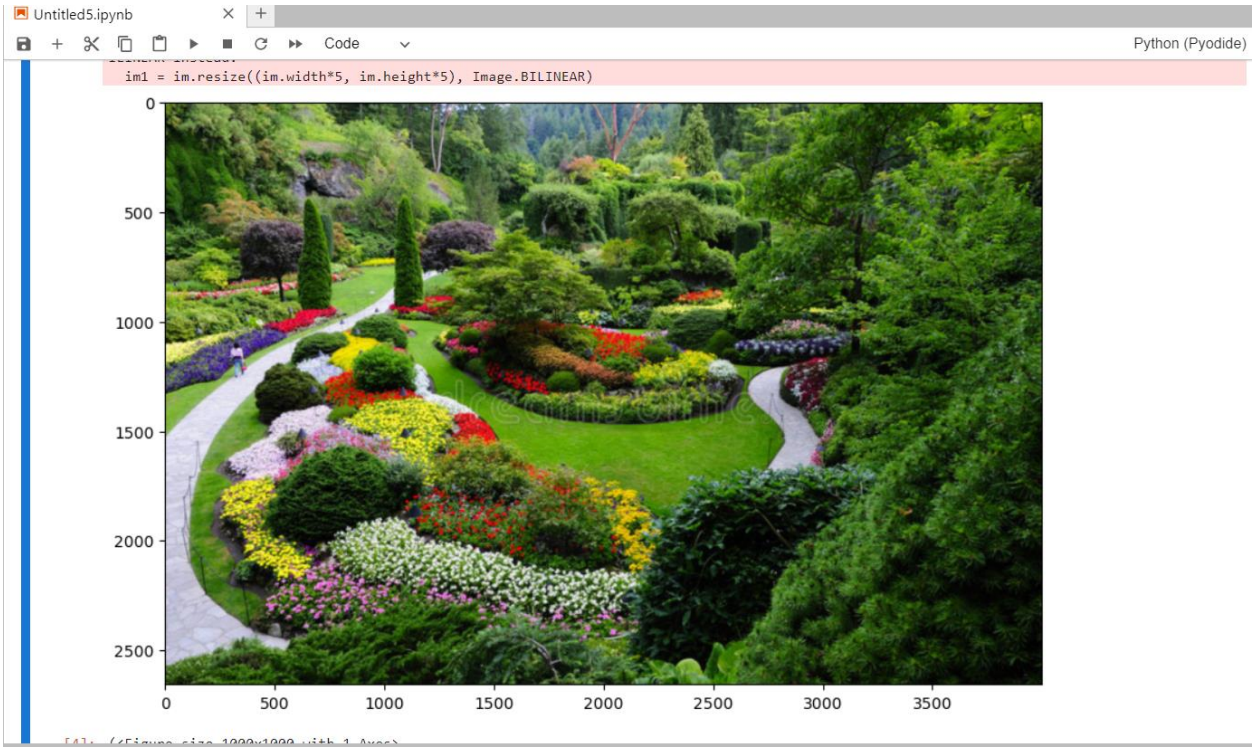
To compute the DFT of an N-point sequence using equation (1) would take O(N2) mul- tiplies and adds. The FFT algorithm computes the DFT using O(N log N) multiplies and adds. There are many variants of the FFT algorithm.

```
from PIL import Image
from skimage.io import imread, imshow, show
import scipy.fftpack as fp
from scipy import ndimage, misc, signal
from skimage import data, img_as_float
from skimage.color import rgb2gray
from skimage.transform import rescale
import matplotlib.pylab as pylab
import numpy as np
import numpy.fft
import timeit

im = Image.open("garden2.jpg")
pylab.imshow(im), pylab.show()
```


```
: (<matplotlib.image.AxesImage at 0x452ebf0>, None)
```

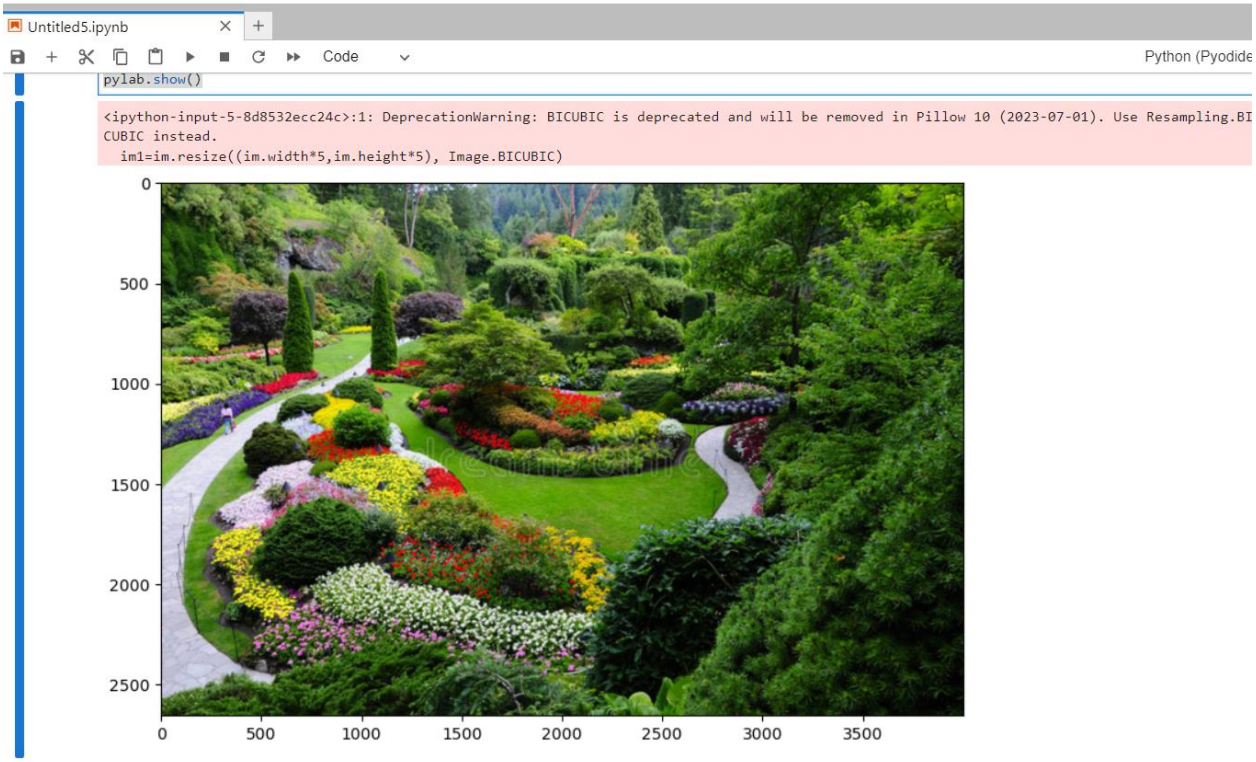
```
im1=im.resize((im.width*5, im.height*5), Image.NEAREST)
pylab.figure(figsize=(10,10)), pylab.imshow(im1), pylab.show()
```


```
im1 = im.resize((im.width*5, im.height*5), Image.BILINEAR)
pylab.figure(figsize=(10,10)), pylab.imshow(im1), pylab.show()
```



(<Figure size 1000x1000 with 1 Axes>,
<matplotlib.image.AxesImage at 0x491a340>,
None)

```
im1=im.resize((im.width*5,im.height*5), Image.BICUBIC)
pylab.figure(figsize=(9,9)), pylab.imshow(im1),
pylab.show()
```

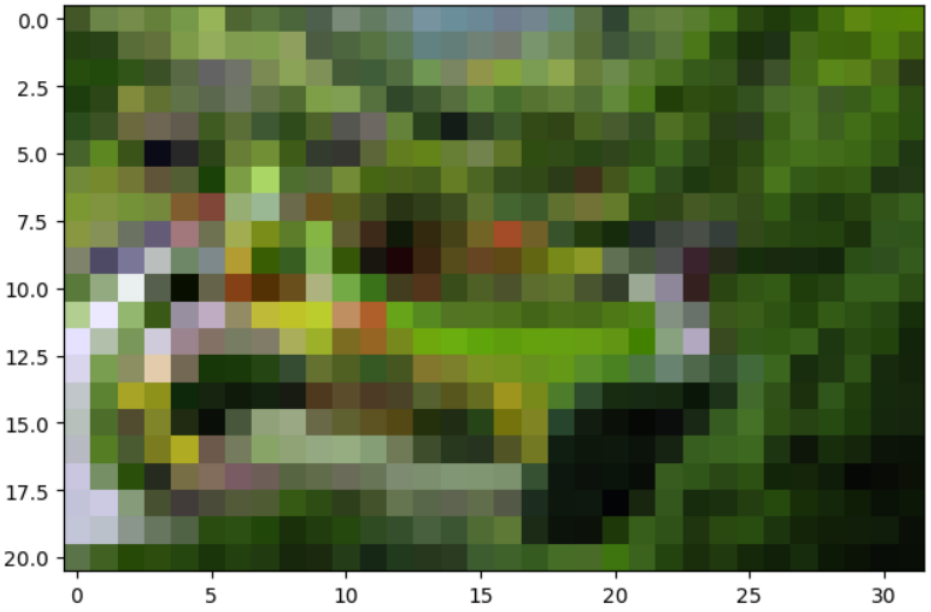


```
im=Image.open("garden2.jpg")
im=im.resize((im.width//5,im.height//5))
pylab.figure(figsize=(10,5)),pylab.imshow(im),
pylab.show()
```

```
pylab.show()
```

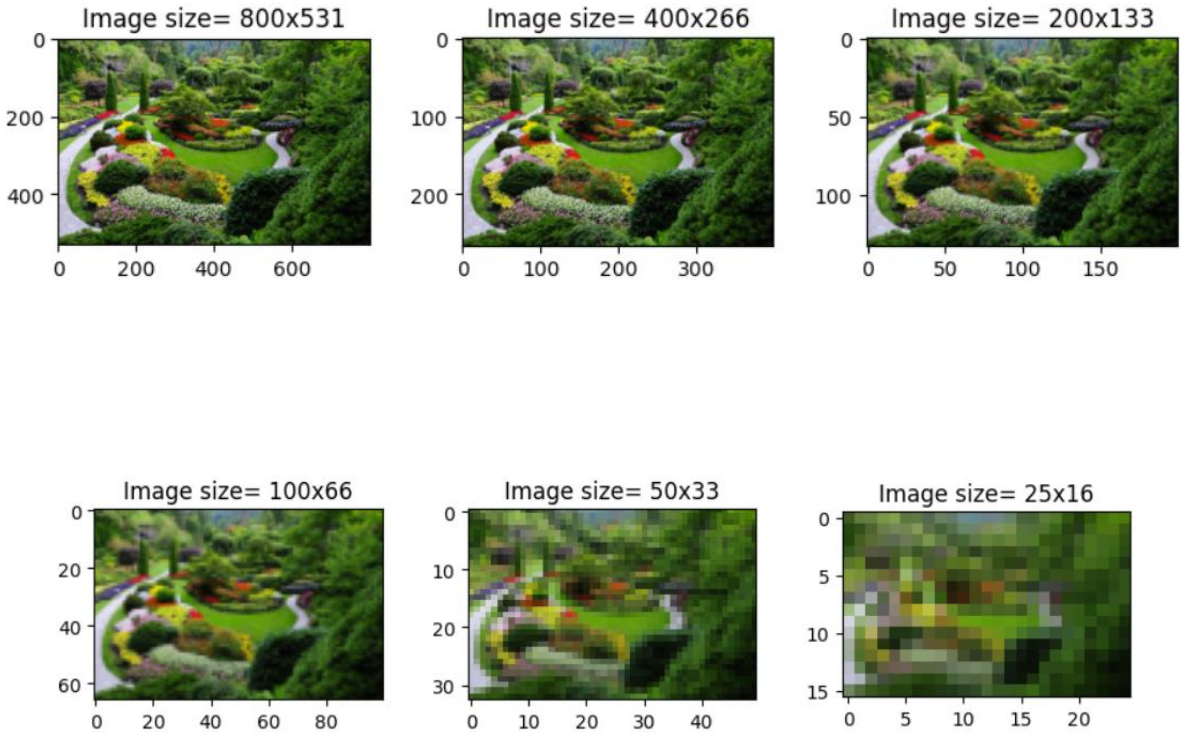

```
im=im.resize((im.width//5,im.height//5), Image.ANTIALIAS)
pylab.figure(figsize=(10,5)),pylab.imshow(im),pylab.show()
```

```
LANCZOS instead.
  im=im.resize((im.width//5,im.height//5), Image.ANTIALIAS)
```




```
[7]: (<Figure size 1000x500 with 1 Axes>,
      <matplotlib.image.AxesImage at 0x4aa02d8>,
      None)
```

```
im=imread('garden2.jpg')
im1=im.copy()
pylab.figure(figsize=(10,10))
for i in range(6):
    pylab.subplot(2,3,i+1),
    pylab.imshow(im1,cmap='Spectral_r'),pylab.axis('on')
    pylab.title('Image size= '+str(im1.shape[1])+'x'+str(im1.shape[0]))
    im1=rescale(im1,scale=0.5,multichannel=True, anti_aliasing=True)
pylab.subplots_adjust(wspace=0.3, hspace=0.3),
pylab.show()
```

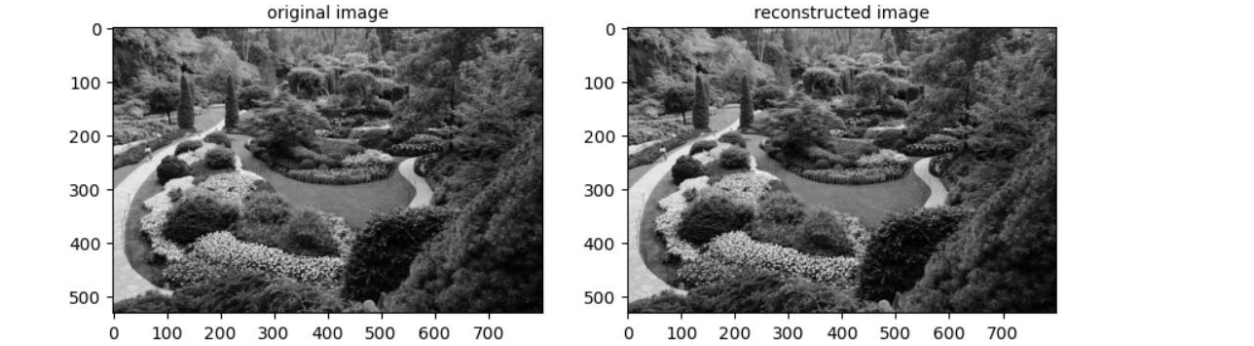



```
def signaltonoise(a,axis=0,ddof=0):
    a=np.asanyarray(a)
```

```
    n=a.mean(axis)
    sd=a.std(axis=axis,ddof=ddof)
    return np.where (sd==0, 0, n/sd)
pylab.figure(figsize=(10,10))
num_colors_list=[1<<n for n in range (8,0,-1)]
snr_list=[]
i=1
for num_colors in num_colors_list:
    im1=im.convert('P',palette=Image.ADAPTIVE, colors=num_colors)
    pylab.subplot(4,2,i),pylab.imshow(im1), pylab.axis('off')
    snr_list.append(signaltonoise(im1,axis=None))
    pylab.title('Image with $ colors = '+str(num_colors)+ 'SNR=' + str(np.round(snr_list[i-
1],3)),size=10)
    i+=1
pylab.subplots_adjust(wspace=0.2, hspace=0.2)
pylab.show()


pylab.plot(num_colors_list,snr_list,'r.-')
pylab.xlabel('#colorsintheimage')
pylab.ylabel('SNR')
pylab.title('ChangeinSNRw.r.t.#colors')
pylab.xscale('log',basex=2)
pylab.gca().invert_xaxis()
pylab.show()
```

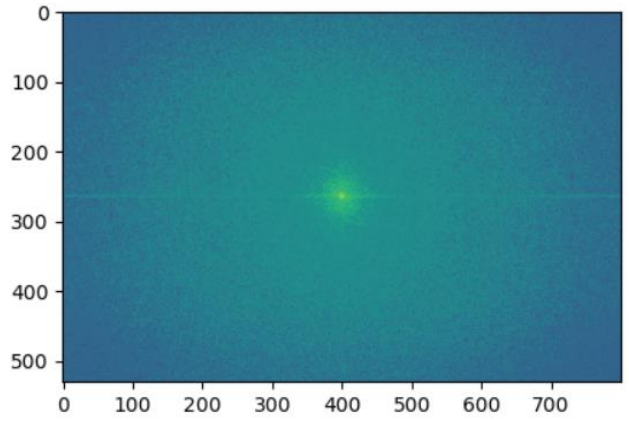
```
# b) Fast Fourier Transform to compute DFT # FFT Operations
im=np.array(Image.open("garden2.jpg").convert('L'))
snr=signaltonoise(im, axis=None)
print("SNR for the original Image =" +str(snr))
freq=fp.fft2(im)
im1=fp.ifft2(freq).real
```

```
snr=signaltonoise(im1,axis=None)
print('SNR for the original Image =' +str(snr))
assert(np.allclose(im,im1))
pylab.figure(figsize=(10,10))
pylab.subplot(121),
pylab.imshow(im, cmap='gray'),
pylab.axis('on')
pylab.title('original image' ,size=10)
pylab.subplot(122),
pylab.imshow(im, cmap='gray'),
pylab.axis('on')
pylab.title('reconstructed image' ,size=10)
#SNR for the original Image =2.4397639089933434
#SNR for the original Image =2.439763908993343

#Text(0.5, 1.0, 'reconstructed image')
```



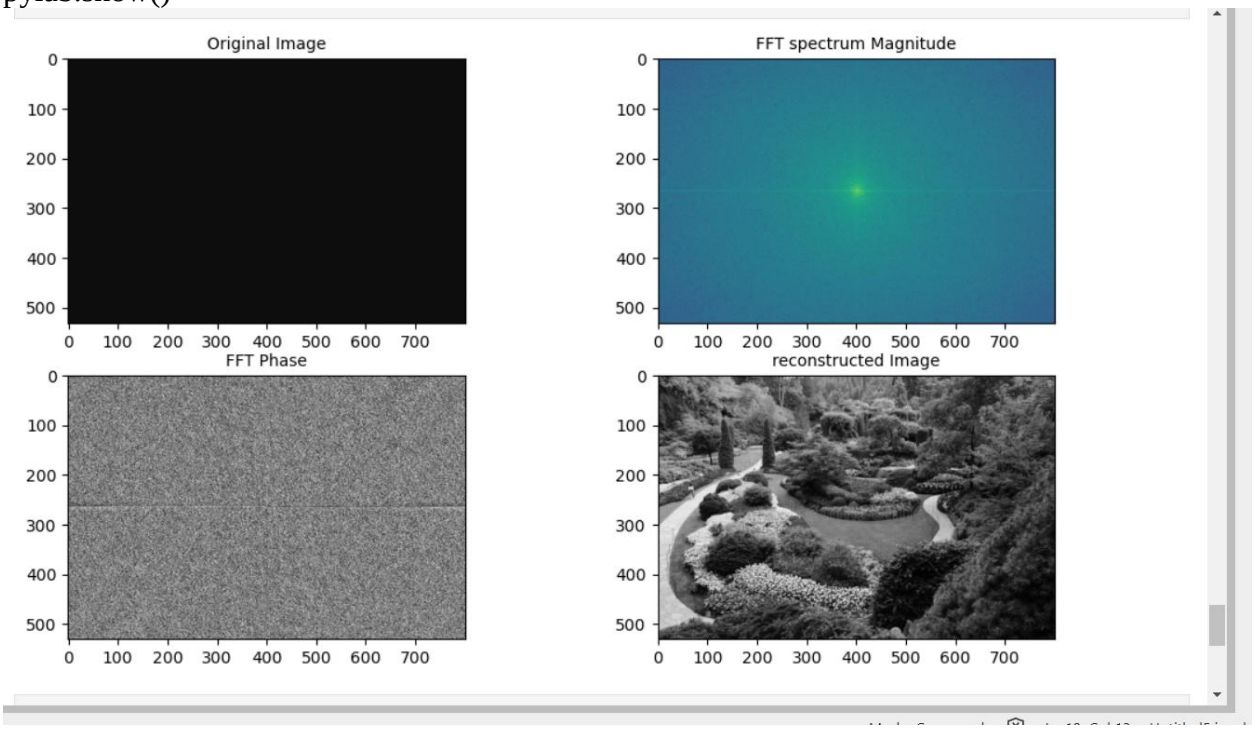

```
freq2=fp.fftshift(freq)
pylab.figure(figsize=(5,5)),
pylab.imshow(20*np.log10(0.1+freq2).astype(int)),
pylab.show()
```

```
<ipython-input-33-d40bb676eb3b>:3: ComplexWarning: Casting complex values to real discards the imaginary part
  pylab.imshow(20*np.log10(0.1+freq2).astype(int)),
```



In [43]: #FFT with numpy

```
import numpy.fft as fp
im1=rgb2gray(imread("garden2.jpg"))
pylab.figure(figsize=(12,10))
freq1=fp.fft2(im1)
im1=fp.ifft2(im1).real
pylab.subplot(3,2,1),
pylab.imshow(im1, cmap='gray')
pylab.title('Original Image', size=10)
pylab.subplot(3,2,2),
pylab.imshow(20*np.log10(0.01 + np.abs(fp.fftshift(freq1))))
pylab.title('FFT spectrum Magnitude', size=10)
pylab.subplot(3,2,3),
pylab.imshow(np.angle(fp.fftshift(freq1)), cmap='gray')
pylab.title('FFT Phase' , size=10)
pylab.subplot(3,2,4),
pylab.imshow(np.clip(im,0,255), cmap='gray')
pylab.title('reconstructed Image', size=10),
pylab.show()
```



Practical no 2

AIM :- Write program to demonstrate the following aspects of signal on sound/image data

1.Convolution operation
2.Template Matching

Theory:

CONVOLUTION OPERATION:-

Convolution is **a mathematical operation that allows the merging of two sets of information**. In the case of CNN, convolution is applied to the input data to filter the information and produce a feature map. This filter is also called a kernel, or feature detector, and its dimensions can be, for example, 3x3.

We've already described how convolution layers work above. They are at the center of CNNs, enabling them to autonomously recognize features in the images.

TEMPLATE MATCHING:-

Template matching is the process of moving the template over the entire image and calculating the similarity between the template and the covered window on the image. Template matching is implemented through two dimensional convolution .In convolution, the value of an output pixel is computed by multiplying elements of two matrices and summing the results. One of these matrices represents the image itself, while the other matrix is the template, which is known as a convolution kernel.

Code:-

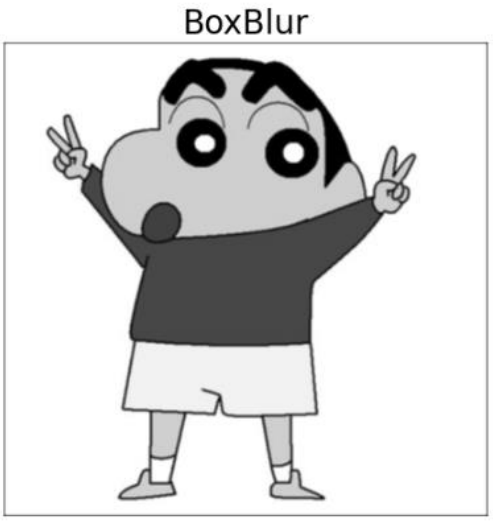
```
from skimage.io import imread,imshow,show
from skimage.color import rgb2gray
import numpy as np
from scipy import ndimage,misc,signal
import matplotlib.pylab as pylab
###Convolution on grey and Color Images
im = rgb2gray(imread("Untitled Folder/schinchan.png")).astype(float)
print(np.max(im))
print(im.shape)
blur_box_kernel=np.ones((3,3))/9
edge_laplace_kernel=np.array([[0,1,0],[1,-4,1],[0,1,0]])
```

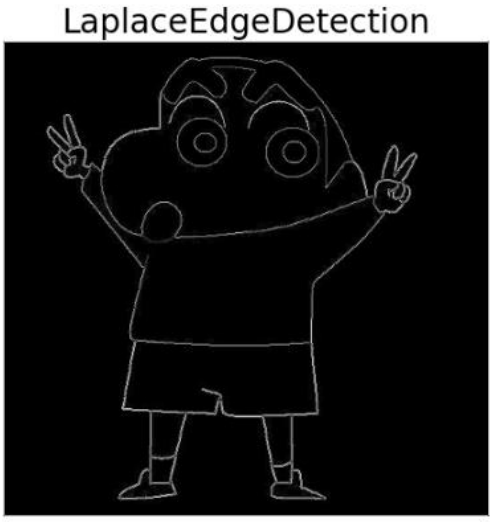
```
im_blurred = signal.convolve2d(im,blur_box_kernel)
im_edges=np.clip(signal.convolve2d(im,edge_laplace_kernel),0,1)
fig,axes=pylab.subplots(ncols=3,sharex=True,sharey=True,figsize=(18,6))
axes[0].imshow(im,cmap=pylab.cm.gray)
axes[0].set_title('OriginalImage',size=20)
axes[1].imshow(im_blurred,cmap=pylab.cm.gray)
axes[1].set_title('BoxBlur',size=20)
axes[2].imshow(im_edges,cmap=pylab.cm.gray)
axes[2].set_title('LaplaceEdgeDetection',size=20)
for ax in axes:
 ax.axis('off')
pylab.show()
```

OUTPUT

```
1.0
(525, 535)
```


```
(280, 390)
```





#Applying convolution to a color (RGB) image

```
im = imread("Untitled Folder/schinchan.png").astype(np.float)
print(np.max(im))
sharpen_kernel=np.array([0,-1,0,-1,5,-1,0,-1,0]).reshape((3,3,1))
emboss_kernel=np.array(np.array([[-2,-1,0],[-1,1,1],[0,1,2]])).reshape((3,3,1))
im_sharp=ndimage.convolve(im,sharpen_kernel,mode='nearest')
im_sharp=np.clip(im_sharp,0,255).astype(np.uint8)
255.0
#clip(0to255)andconverttounsignedint
im_emboss=ndimage.convolve(im,emboss_kernel,mode='nearest')
im_emboss=np.clip(im_emboss,0,255).astype(np.uint8)
pylab.figure(figsize=(10,15))
pylab.subplot(131),
```

```
pylab.imshow(im.astype(np.uint8)),
pylab.axis('off')
pylab.title('OriginalImage',size=25)
pylab.subplot(132),
pylab.imshow(im_sharp),
pylab.axis('off')
pylab.title('SharpenedImage',size=25)
pylab.subplot(133),
pylab.imshow(im_emboss),
pylab.axis('off')
pylab.title('EmbossedImage',size=25)
pylab.tight_layout()
pylab.show()
im_gray=ndimage.convolve(im,emboss_kernel,mode='nearest')
im_gray=np.clip(im_gray,0,255).astype(np.uint8)
pylab.figure(figsize=(10,15))
pylab.subplot(133),
pylab.imshow(im_gray),
pylab.axis('off')
pylab.title('grayImage',size=25)
pylab.tight_layout()
pylab.show()
```

OUTPUT:-

OriginalImage

SharpenedImage

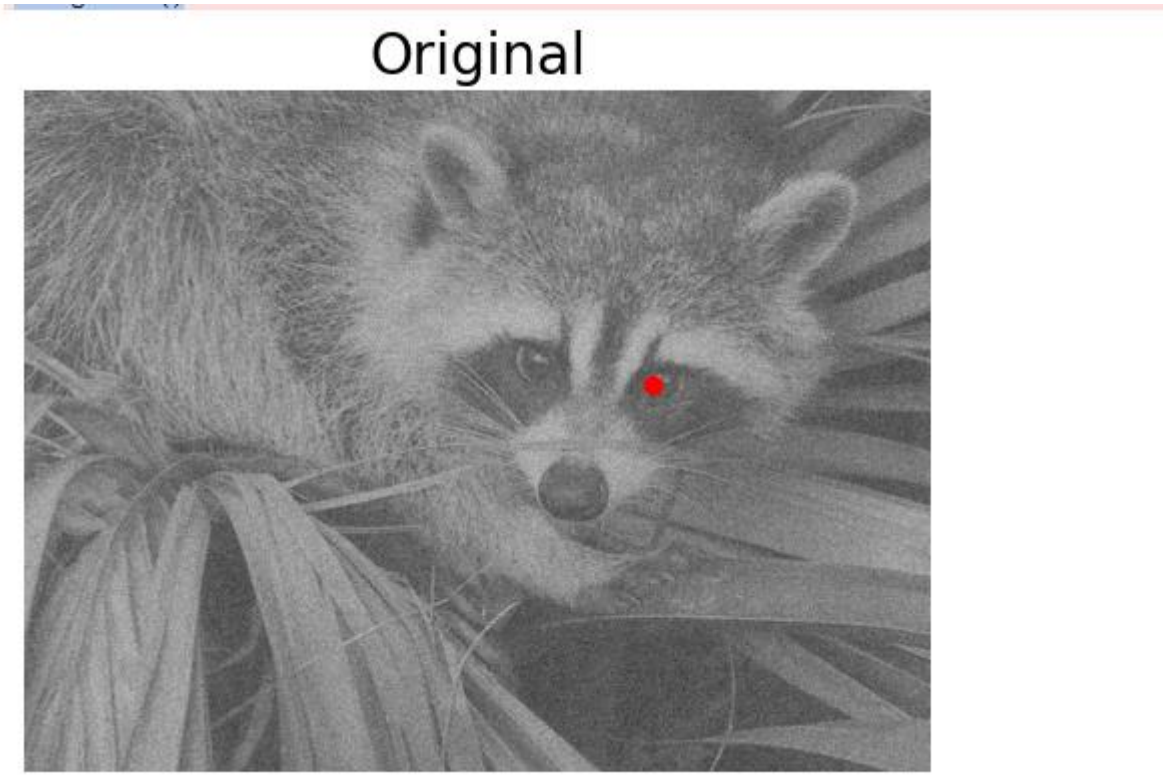
EmbossedImage

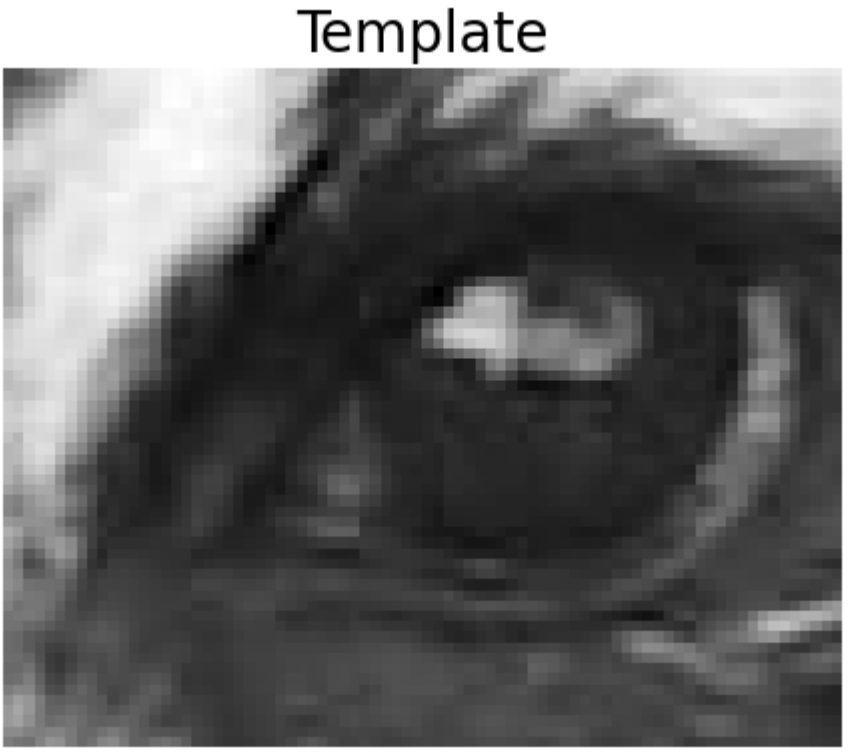
grayImage

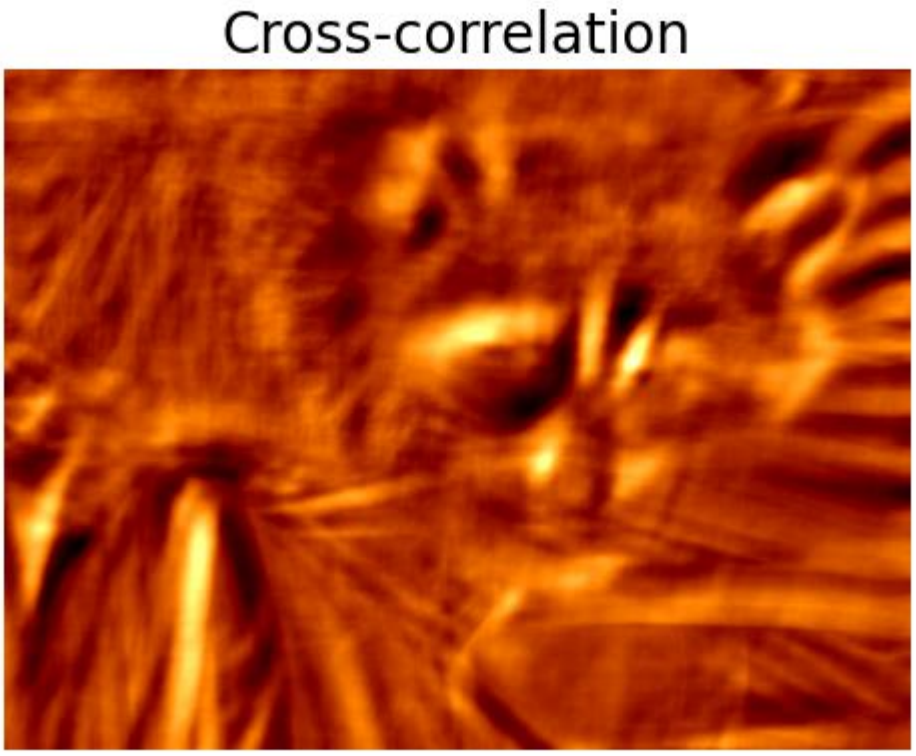
##Template Matching

#Template matching with cross-correlation between the image and template

```
face_image=misc.face(gray=True)-misc.face(gray=True).mean()
template_image=np.copy(face_image[300:365,670:750])
#righteye
template_image-=template_image.mean()
face_image=face_image+np.random.randn(*face_image.shape)*50
#add random noise
correlation=signal.correlate2d(face_image,template_image,boundary='symm',mode='same')
y,x=np.unravel_index(np.argmax(correlation),correlation.shape)
#find the match
fig,(ax_original,ax_template,ax_correlation)=pylab.subplots(3,1,figsize=(6,15))
ax_original.imshow(face_image,cmap='gray')
ax_original.set_title('Original',size=20)
ax_original.set_axis_off()
ax_template.imshow(template_image,cmap='gray')
ax_template.set_title('Template',size=20)
ax_template.set_axis_off()
ax_correlation.imshow(correlation,cmap='afmhot')
ax_correlation.set_title('Cross-correlation',size=20)
ax_correlation.set_axis_off()
ax_original.plot(x,y,'ro')
fig.show()
```




Template


Cross-correlation

PRACTICAL-3

Aim:- 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

Theory:

Log and Power-law transformations :-

The general form of the log transformation is s = c * log (1 + r) The log transformation maps [5] a narrow range of low input grey level values into a wider range of output values. The inverse log transformation performs the opposite transformation. Log functions are particularly useful when the input grey level values may have an extremely large range of values. In the following example the Fourier transform of an image is put through a log transform to reveal more detail.

The nth power and nth root curves shown in fig. A can be given by the expression, $s = cr^{\gamma}$. This transformation function is also called as gamma correction [$s = cr^{\gamma}$]. For various values of γ different levels of enhancements can be obtained. This technique is quite commonly called as Gamma Correction. If you notice, different display monitors display images at different intensities and clarity.

The difference between the logtransformation function and the power-law functions is that using the power-law function a family of possible transformation curves can be obtained just by varying

the λ. These are the three basic image enhancement functions for grey scale images that can be applied easily for any type of image for better contrast and highlighting.

Contrast Adjustment:-

Contrast adjustment remaps image intensity values to the full display range of the data type. An image with good contrast has sharp differences between black and white. To illustrate, the image on the left has poor contrast, with intensity values limited to the middle portion of the range.

Histogram equalization

Histogram Equalization is a computer image processing technique used to improve contrast in images . It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image.

A color histogram of an image represents the number of pixels in each type of color component. Histogram equalization cannot be applied separately to the Red, Green and Blue components of the image as it leads to dramatic changes in the image's color balance. However, if the image is first converted to another color space, like HSL/HSV color space, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image. The histogram of an image gives important information about the grayscale and contrast of the image. If the entire histogram of an image is centered towards the left end of the x-axis, then it implies a dark image. If the histogram is more inclined towards the right end, it signifies a white or bright image. A narrow-width histogram plot at the center of the intensity axis shows a low-contrast image, as it has a few levels of grayscale. On the other hand, an evenly distributed histogram over the entire x-axis gives a high-contrast effect to the image.

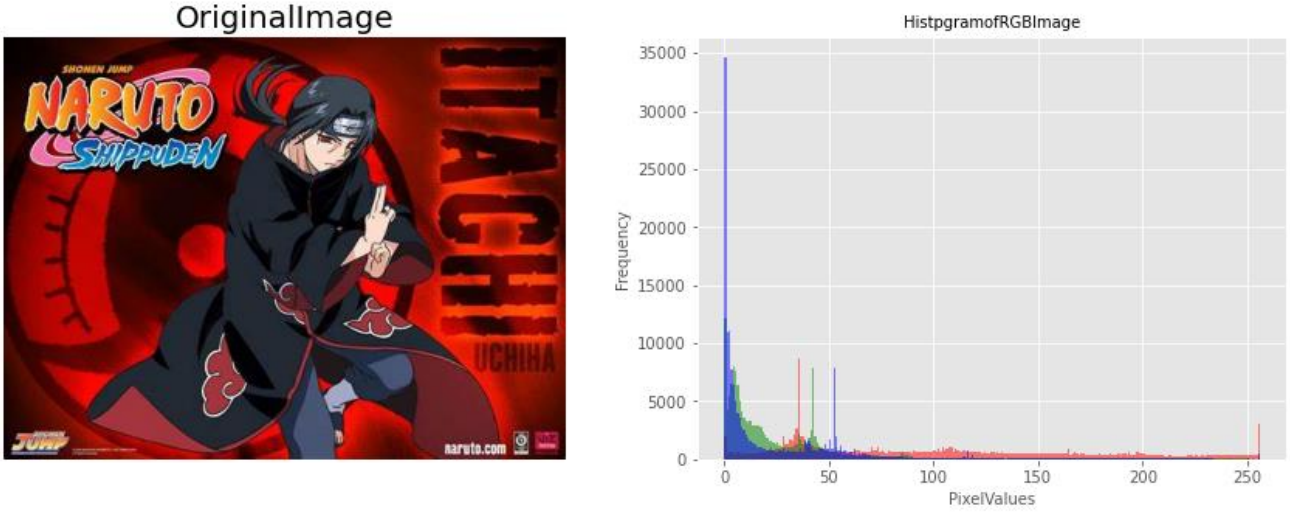
Thresholding, and halftoning operations:-

Thresholding is a type of image segmentation, where we change the pixels of an image to make the image easier to analyze. In thresholding, we convert an image from colour or grayscale into a binary image, i.e., one that is simply black and white. The downside of the simple thresholding technique is that we have to make an educated guess about the threshold t by inspecting the histogram. There are also *automatic thresholding* methods that can determine the threshold automatically for us. One such method is *Otsu's method*. It is particularly useful for situations where the grayscale histogram of an image has two peaks that correspond to background and objects of interest.

Halftoning or analog halftoning is a process that simulates shades of gray by varying the size of tiny black dots arranged in a regular pattern. This technique is used in printers, as well as the publishing industry. If you inspect a photograph in a newspaper, you will notice that the picture is composed of black dots even though it appears to be composed of grays. This is possible because of the spatial integration performed by our eyes.

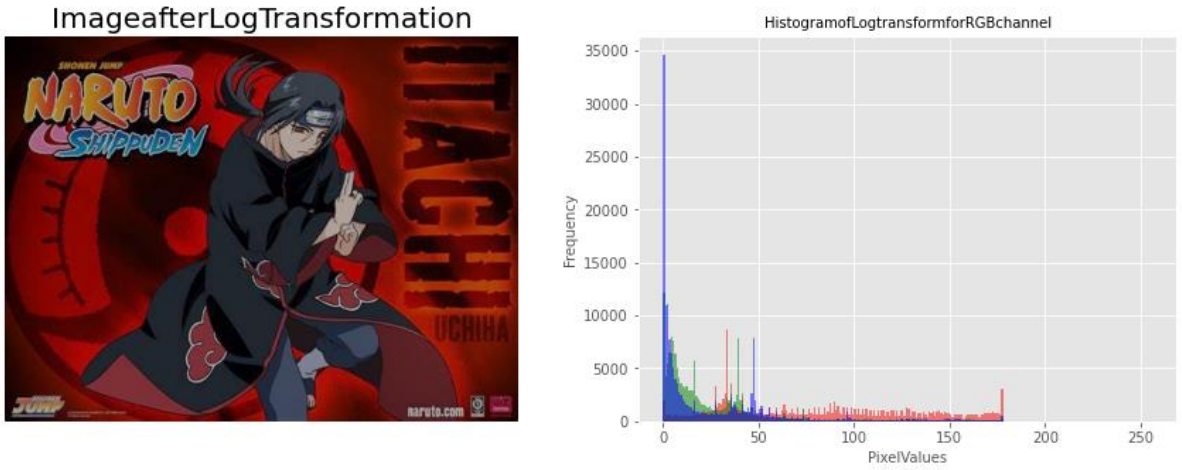
```
import numpy as np
from skimage import data,img_as_float,img_as_ubyte,exposure,io,color
from skimage.io import imread
from skimage.exposure import cumulative_distribution
from skimage.restoration import denoise_bilateral,denoise_nl_means,estimate_sigma
#from skimage.measure importcompare_psnr
from skimage.util import random_noise
from skimage.color import rgb2gray
from PIL import Image,ImageEnhance,ImageFilter
```

```
from scipy import ndimage,misc
import matplotlib.pylab as pylab

#loadimageandplothistogram #functiontodisplayimage
def plot_image (image,title = "Itachi Uchiha" ):
    pylab.title(title,size=20),pylab.imshow(image)
    pylab.axis('off')

#functiontodisplayhistogram
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=0.5)
    pylab.hist(np.array(g).ravel(),bins=256,range=(0,256),color='g',alpha=0.5)
    pylab.hist(np.array(b).ravel(),bins=256,range=(0,256),color='b',alpha=0.5)
    pylab.xlabel('PixelValues',size=10),pylab.ylabel('Frequency',size=10)
    pylab.title(title,size=10)
im=Image.open("/home/ubuntu/Downloads/itachi.jpeg")
im_r,im_g,im_b=im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121),plot_image(im,'OriginalImage')
pylab.subplot(122),plot_hist(im_r,im_g,im_b,'HistpgramofRGBImage')
pylab.show()
```

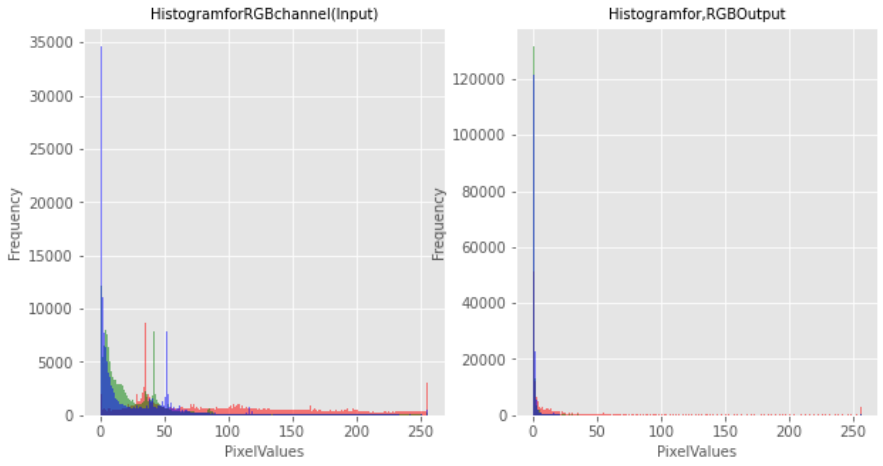



```
##Log and Power-law transformations
#log transformation
im=im.point(lambda i:255*np.log(1+i/255))
im_r,im_g,im_b=im.split()
pylab.style.use('ggplot')
```

```
pylab.figure(figsize=(15,5))
pylab.subplot(121),plot_image(im,'ImageafterLogTransformation')
pylab.subplot(122),plot_hist(im_r,im_g,im_b,'HistogramofLogtransformforRGBchannel')
pylab.show()
```

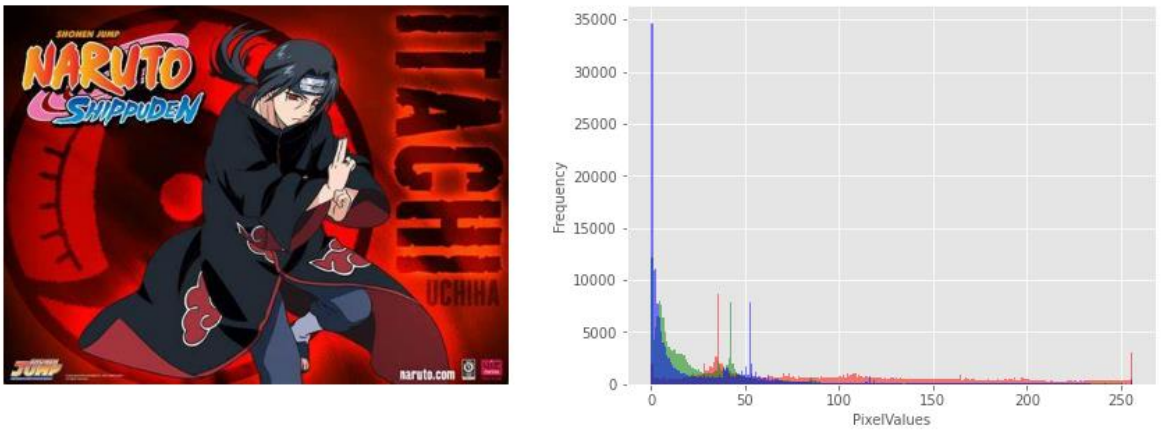



```
#power law transform
im=img_as_float(imread("/home/ubuntu/Downloads/itachi.jpeg"))
#im_r, im_g, im_b = im.split()
gamma=3.5
im1=im**gamma
pylab.style.use('ggplot')
pylab.figure(figsize=(10,5))
pylab.subplot(121),plot_hist(im[...,0],im[...,1],im[...,2],'HistogramforRGBchannel(Input)')
pylab.subplot(122),plot_hist(im1[...,0],im1[...,1],im1[...,2],'Histogramfor,RGBOutput')
pylab.show()
```

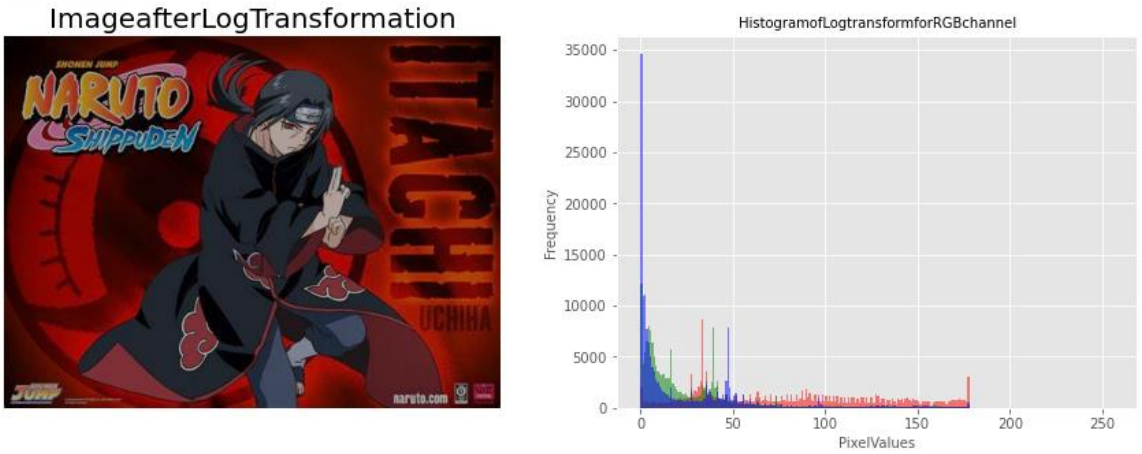



```
pylab.figure(figsize=(10,5))
pylab.subplot(121),plot_image(im,'Imageoriginal')
pylab.subplot(122),plot_image(im1,'Output')
pylab.show()
```

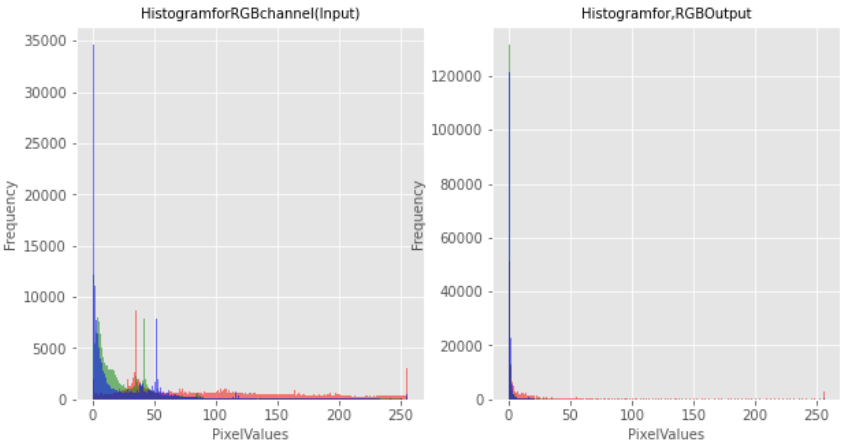
##Contrast adjustments
#constrast streching
im=Image.open("/home/ubuntu/Downloads/itachi.jpeg")
im_r,im_g,im_b=im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(im)
pylab.subplot(122)
plot_hist(im_r,im_g,im_b)
pylab.show()



##Log and Power-law transformations
#log transformation
im=im.point(lambda i:255*np.log(1+i/255))
im_r,im_g,im_b=im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121),plot_image(im,'ImageafterLogTransformation')
pylab.subplot(122),plot_hist(im_r,im_g,im_b,'HistogramofLogtransformforRGBchannel')
pylab.show()



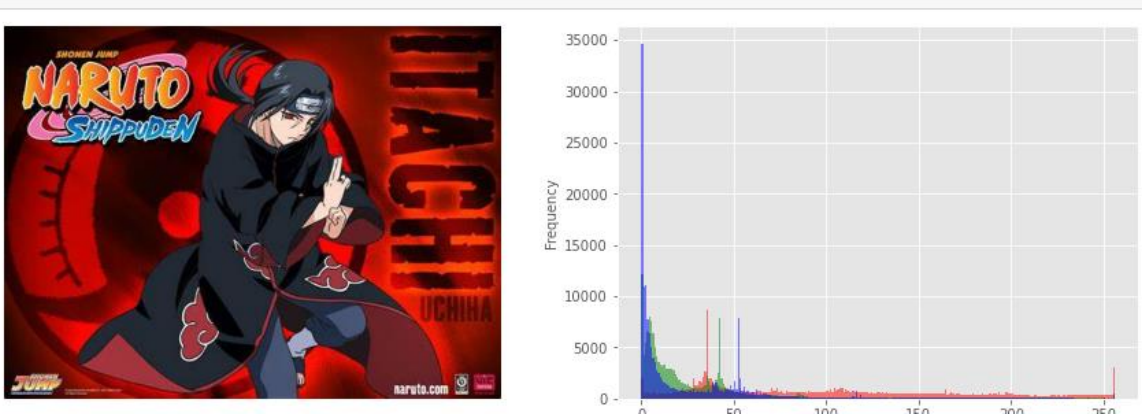

```
#power law transform
im=img_as_float(imread("/home/ubuntu/Downloads/itachi.jpeg"))
#im_r, im_g, im_b = im.split()
gamma=3.5
im1=im**gamma
pylab.style.use('ggplot')
pylab.figure(figsize=(10,5))
pylab.subplot(121),plot_hist(im[...,0],im[...,1],im[...,2],'HistogramforRGBchannel(Input)')
pylab.subplot(122),plot_hist(im1[...,0],im1[...,1],im1[...,2],'Histogramfor,RGBOutput')
pylab.show()
```



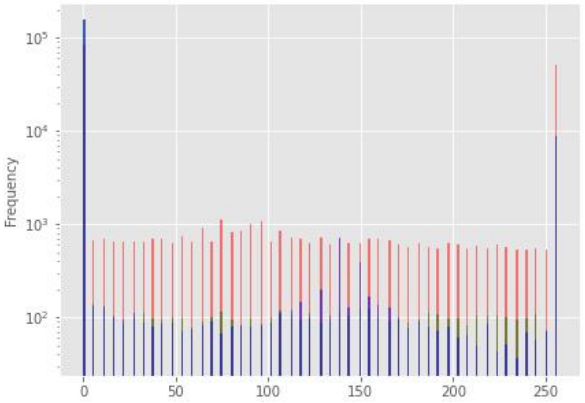

```
pylab.figure(figsize=(10,5))
pylab.subplot(121),plot_image(im,'Imageoriginal')
pylab.subplot(122),plot_image(im1,'Output')
pylab.show()
```

##Contrast adjustments
#constrast streching
im=Image.open("/home/ubuntu/Downloads/itachi.jpeg")
im_r,im_g,im_b=im.split()
pylab.style.use('ggplot')

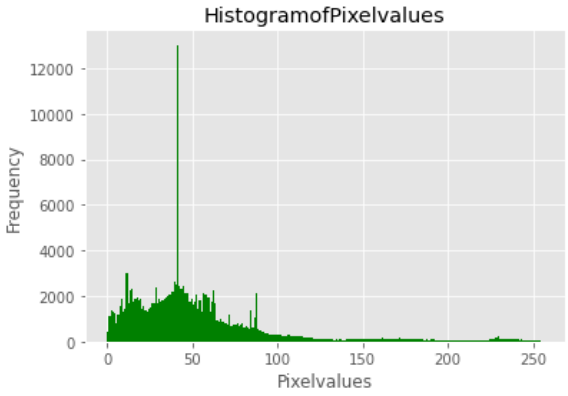
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(im)
pylab.subplot(122)
plot_hist(im_r,im_g,im_b)
pylab.show()



#contrast
def contrast (c):
    return 0 if c<50 else (255 if c>150 else(255*c-22950)/48)
im1=im.point(contrast)
im_r,im_g,im_b=im1.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(im1)
pylab.subplot(122)
plot_hist(im_r,im_g,im_b)
pylab.yscale('log',base=10)
pylab.show()




```
##Thresholding, and halftoning operations
#thresholding operations
im=Image.open("/home/ubuntu/Downloads/itachi.jpeg").convert('L')
pylab.hist(np.array(im).ravel(),bins=256,range=(0,256),color='g')
pylab.xlabel('Pixelvalues'),pylab.ylabel('Frequency')
pylab.title('HistogramofPixelvalues')
pylab.show()
pylab.figure(figsize=(12,18))
pylab.gray()
pylab.subplot(221),plot_image(im,'OriginalImage')
pylab.axis('off')
th=[0,50,100,150,200]
for i in range (2,5):
    im1=im.point(lambda x:x>th[i])
    pylab.subplot(2,2,i),plot_image(im1,'binaryImagewiththreshold='+str(th[i]))
pylab.show()
```



binaryImagewiththreshold=150

binaryImagewiththreshold=200

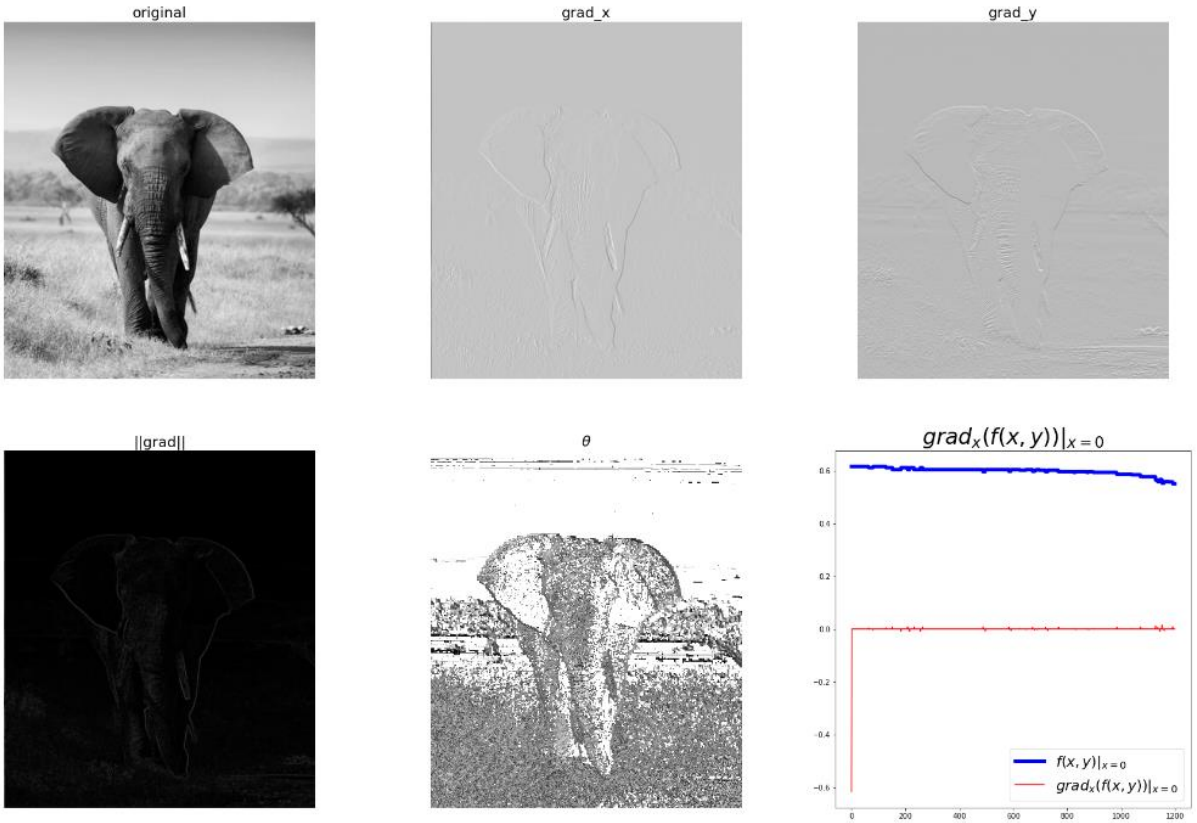
PRACTICAL-4

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

Theory:

Recall that the gradient of a two-dimensional function, f, is given by: Then, the Laplacian (that is, the divergence of the gradient) of f can be defined by the sum of unmixed second partial derivatives: It can, equivalently, be considered as the trace (tr) of the function's Hessian, H(f).
Gradient descent is the method that iteratively searches for a minimizer by looking in the gradient direction. Conjugate gradient is similar, but the search directions are also required to be orthogonal to each other in the sense that pTiApj=0∀i,j.

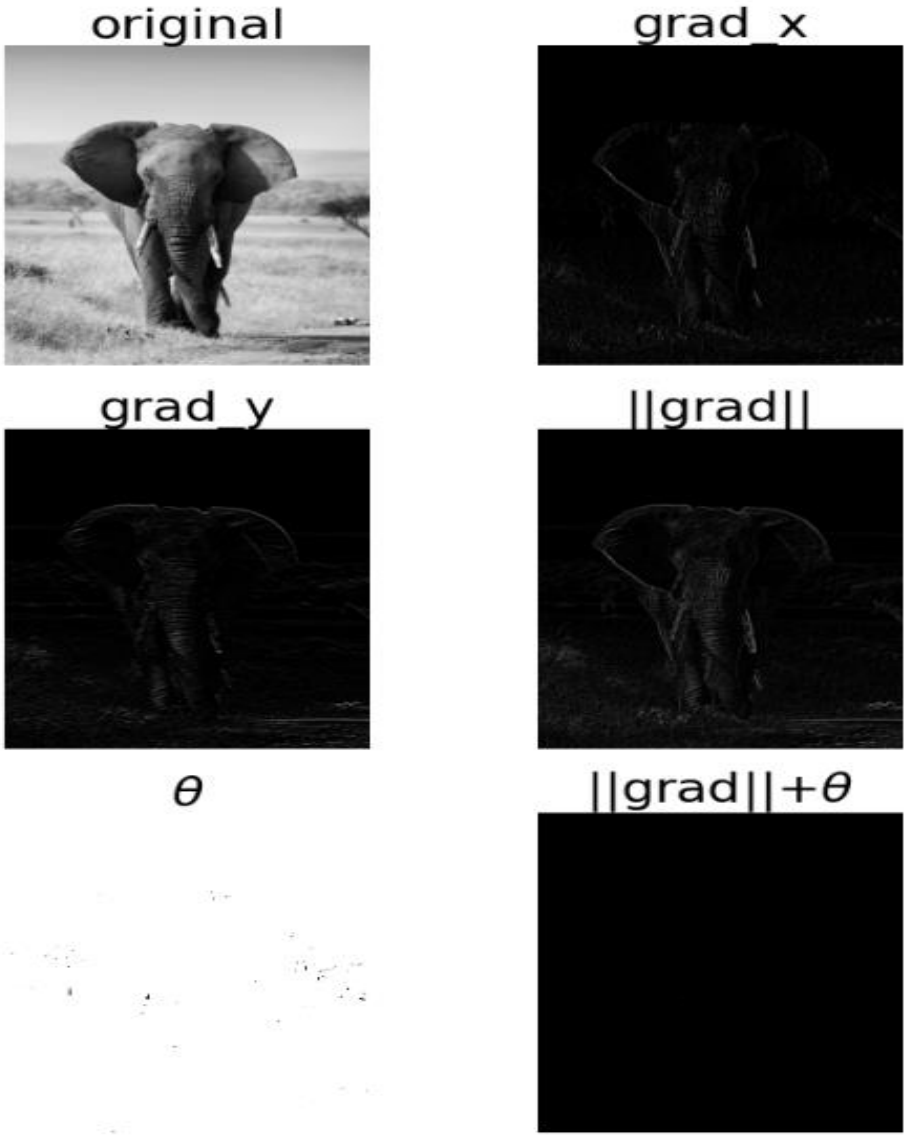
```
#% matplotlib inline
import numpy as np
from scipy import signal,misc,ndimage
from skimage import filters,feature,img_as_float
from skimage.io import imread
from skimage.color import rgb2gray
from PIL import Image,ImageFilter
import matplotlib.pylab as pylab

#Derivativesandgradients
def plot_image(image,title): pylab.imshow(image),pylab.title(title,size=20),pylab.axis('off')
ker_x=[[-1,1]]
ker_y=[[-1],[1]]
im=rgb2gray(imread('file:///C:/Users/varun/Downloads/images/images/elephant.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,'grad_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()
```



#Displayingthemagnitudeandthegradientonthesameimage

```
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.util import random_noise
from skimage.filters import gaussian
import matplotlib.pylab as plt
from scipy import signal
import numpy as np
ker_x=[[-1,1]]
ker_y=[[-1],[1]]
im=rgb2gray(imread('file:///C:/Users/varun/Downloads/images/images/elephant.jpg'))
#im=random_noise(im,var=sigma**2) #im=gaussian(im,sigma=0.25) print(np.max(im))
im_x=np.clip(signal.convolve2d(im,ker_x,mode='same'),0,1)
im_y=np.clip(signal.convolve2d(im,ker_y,mode='same'),0,1)
im_mag=np.sqrt(im_x**2+im_y**2)
im_ang=np.arctan(im_y/im_x)
plt.gray()
plt.figure(figsize=(10,15))
plt.subplot(321)
plt.imshow(im)
plt.title('original',size=30)
plt.axis('off')
plt.subplot(322)
plt.imshow(im_x)
plt.title('grad_x',size=30)
plt.axis('off')
plt.subplot(323)
```

```
plt.imshow(im_y)
plt.title('grad_y',size=30)
plt.axis('off')
plt.subplot(324)
plt.imshow(im_mag)
plt.title('||grad||',size=30)
plt.axis('off')
plt.subplot(325)
plt.imshow(im_ang)
plt.title(r'$\theta$',size=30)
plt.axis('off')
plt.subplot(326)
im=np.zeros((im.shape[0],im.shape[1],3))
im[...,0]=im_mag*np.sin(im_ang)
im[...,1]=im_mag*np.cos(im_ang)
plt.imshow(im)
plt.title(r'||grad||+$\theta$',size=30)
plt.axis('off')
plt.show()
```



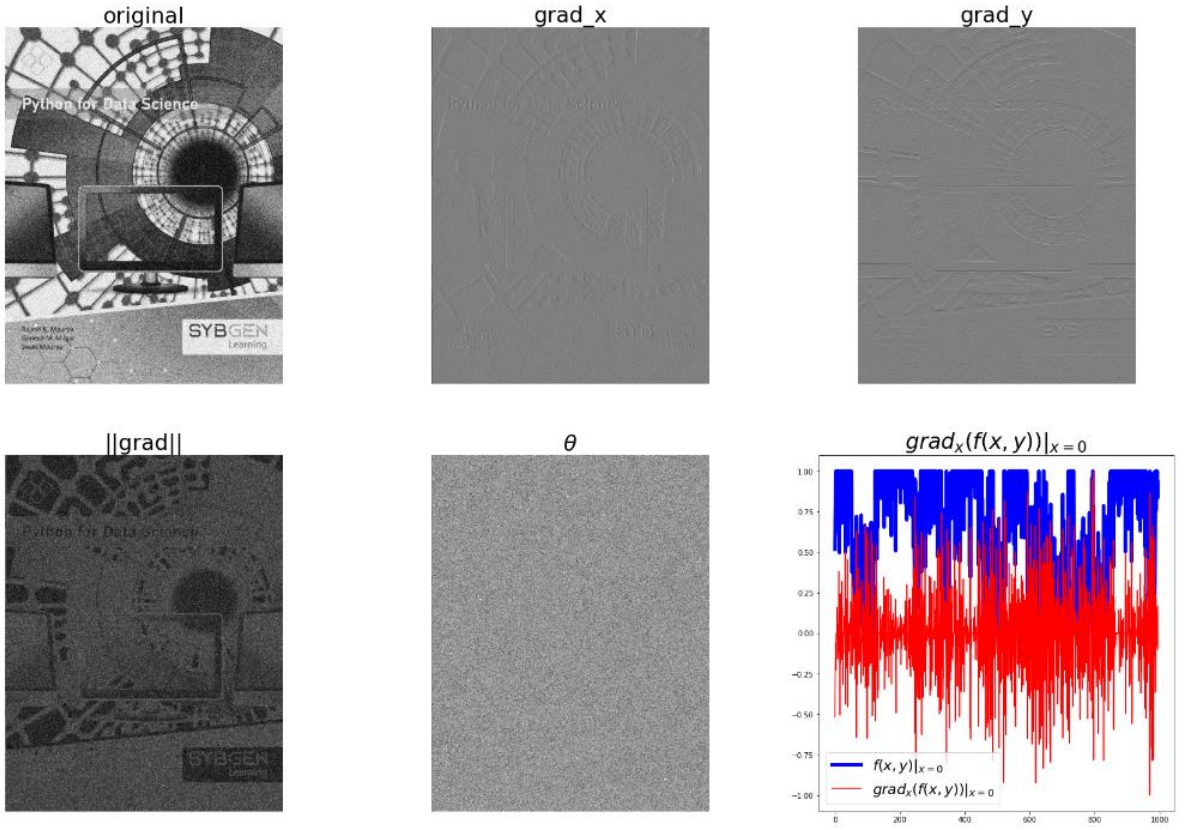
#Laplacian

```
ker_laplacian=[[0,-1,0],[-1,4,-1],[0,-1,0]]
im=rgb2gray(imread('images/cgvr.png'))
im1=np.clip(signal.convolve2d(im,ker_laplacian,mode='same'),0,1)
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(121),plot_image(im,'original')
pylab.subplot(122),plot_image(im1,'laplacianconvolved')
pylab.show()
```

#Effectsofnoiseongradientcomputation

```
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.util import random_noise
from skimage.filters import gaussian
import matplotlib.pylab as plt
from scipy import signal
import numpy as np
ker_x =[[-1,1]]
ker_y=[[-1],[1]]
im=rgb2gray(imread('C:/Users/varun/Downloads/images/images/PythonDS.jpg'))
sigma=0.25
sign=np.random.random(im.shape)
sign[sign<=0.5]=-1
sign[sign>0.5]=1
im=random_noise(im,var=sigma**2)
im=gaussian(im,sigma=0.25)
print(np.max(im))
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_ang=np.arctan(im_y/im_x)
plt.gray()
plt.figure(figsize=(30,20))
plt.subplot(231)
plt.imshow(im)
plt.title('original',size=30)
plt.axis('off')
plt.subplot(232)
plt.imshow(im_x)
plt.title('grad_x',size=30)
plt.axis('off')
```

```
plt.subplot(233)
plt.imshow(im_y)
plt.title('grad_y',size=30)
plt.axis('off')
plt.subplot(234)
plt.imshow(im_mag)
plt.title('||grad||',size=30)
plt.axis('off')
plt.subplot(235)
plt.imshow(im_ang)
plt.title(r'$\theta$',size=30)
plt.axis('off')
plt.subplot(236)
plt.plot(range(im.shape[1]),im[0,:],'b-',label=r'$f(x,y)|_{x=0}$',linewidth=5)
plt.plot(range(im.shape[1]),im_x[0,:],'r-',label=r'$grad_x(f(x,y))|_{x=0}$')
plt.title(r'$grad_x(f(x,y))|_{x=0}$',size=30)
plt.legend(prop={'size':20})
plt.show()
```



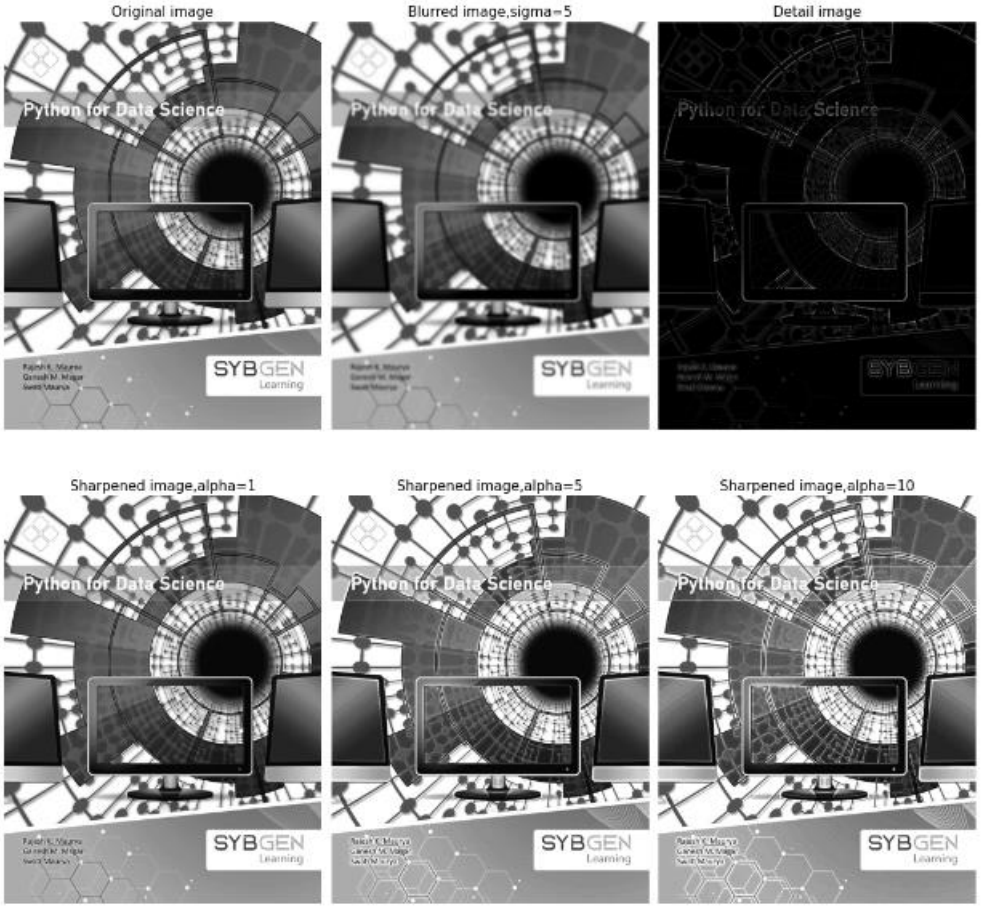
#SharpeningwithLaplacian

```
from skimage.filters import laplace
im=rgb2gray(imread('C:/Users/varun/Downloads/images/images/PythonDS.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()
```

#Unsharpmasking

```
def rgb2gray (im):
#heinputimageisanRGBimage #withpixelvaluesforeachchannelin[0,1]
    return np.clip(0.2989*im[...,0]+0.5870*im[...,1]+0.1140*im[...,2],0,1)
im=rgb2gray(img_as_float(imread('C:/Users/varun/Downloads/images/images/PythonDS.jpg')))
im_blurred=ndimage.gaussian_filter(im,3)
im_detail=np.clip(im-im_blurred,0,1)
pylab.gray()
fig,axes=pylab.subplots(nrows=2,ncols=3,sharex=True,sharey=True,figsize=(15,15))
axes=axes.ravel()
axes[0].set_title('Original image',size=15),axes[0].imshow(im)
axes[1].set_title('Blurred image,sigma=5',size=15),axes[1].imshow(im_blurred)
axes[2].set_title('Detail image',size=15),axes[2].imshow(im_detail)
alpha=[1,5,10]
for i in range(3):
    im_sharp=np.clip(im+alpha[i]*im_detail,0,1)
    axes[3+i].imshow(im_sharp),axes[3+i].set_title('Sharpened
image,alpha='+str(alpha[i]),size=15)
for ax in axes:
    ax.axis('off')
fig.tight_layout()
pylab.show()
```


Original image
Blurred image,sigma=5
Detail image
Sharpened image,alpha=1
Sharpened image,alpha=5
Sharpened image,alpha=10
Python for Data Science
SYBGEN Learning

PRACTICAL-5

Aim :Write a program to implement linear and nonlinear noise smoothing on suitable image or sound signal.

Theory:

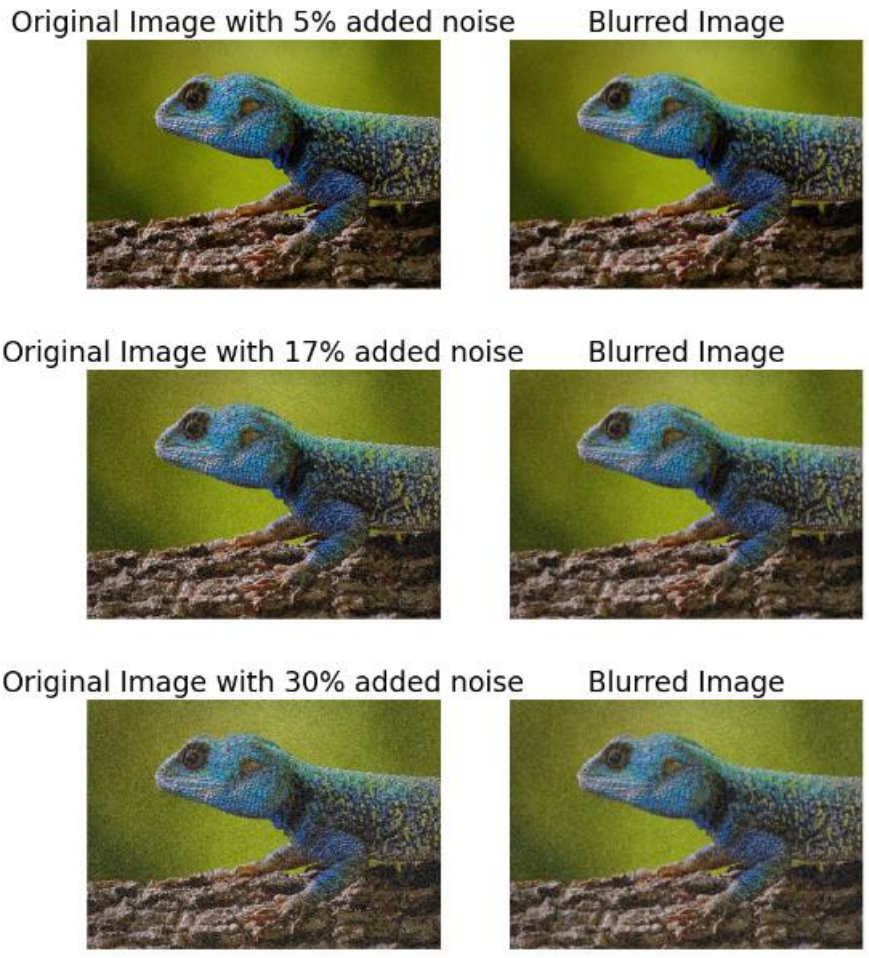
Consider a dataset with p features(or independent variables) and one response(or dependent variable). Also, the dataset contains n rows/observations. We define: X (feature matrix) = a matrix of size n X p where x_{ij} denotes the values of jth feature for ith observation.
Linear means something related to a line. All the linear equations are used to construct a line. A non-linear equation is such which does not form a straight line. It looks like a curve in a graph and has a variable slope value.
It forms a straight line or represents the equation for the straight line. It does not form a straight line but forms a curve.
In smoothing, the data points of a signal are modified so individual points higher than the adjacent points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. Smoothing is a signal processing technique typically used to remove noise from signals.

Code:

```
import numpy as np
from skimage.io import imread
from skimage.restoration import denoise_bilateral, denoise_nl_means,estimate_sigma
#from skimage.measure import compare_psnr
from skimage.util import random_noise
from skimage.color import rgb2gray
from PIL import Image, ImageEnhance, ImageFilter
from scipy import ndimage, misc
import matplotlib.pylab as pylab

def plot_image(image, title=''):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis('off') # comment this line if you want axis ticks
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=0.5)
    pylab.hist(np.array(g).ravel(), bins=256, range=(0, 256), color='g',alpha=0.5)
    pylab.hist(np.array(b).ravel(), bins=256, range=(0, 256), color='b',alpha=0.5)
    pylab.xlabel('pixel value', size=20), pylab.ylabel('frequency', size=20)
   pylab.title(title, size=20)


# Smoothing with ImageFilter.BLUR
i = 1
pylab.figure(figsize=(10,25))
for prop_noise in np.linspace(0.05,0.3,3): # creating numeric sequences evenly spaced
```

```
numbers structured as a NumPy array
    im = Image.open("C:/Users/kalpesh/Desktop/msc-1/dsip/Asip/lizard.jpg")
# choose 5000 random locations inside image
    n = int(im.width * im.height * prop_noise)
    x, y = np.random.randint(0, im.width, n), np.random.randint(0, im.height, n)
    for (x,y) in zip(x,y):
        im.putpixel((x, y), ((0,0,0) if np.random.rand() < 0.5 else(255,255,255))) # generate
salt-and-pepper noise
    im.save("C:/Users/kalpesh/Desktop/msc-1/dsip/Asip" + str(prop_noise) + '.jpg')
    pylab.subplot(6,2,i), plot_image(im, 'Original Image with ' + str(int(100*prop_noise)) +
'% added noise')
    i += 1
    im1 = im.filter(ImageFilter.BLUR)
    pylab.subplot(6,2,i), plot_image(im1, 'Blurred Image')
    i += 1
pylab.show()
```

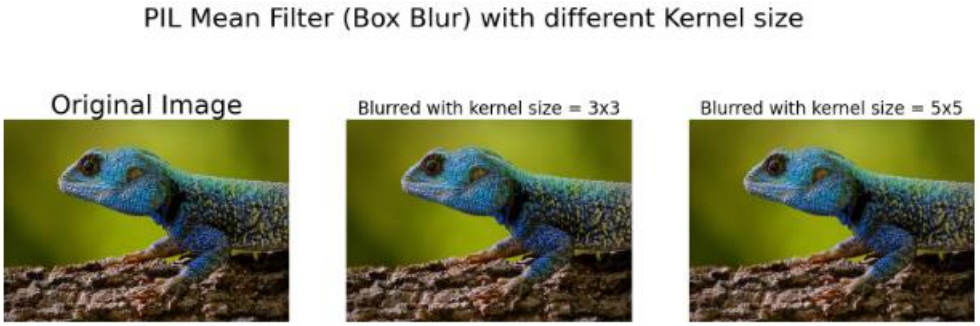
OUTPUT




```
# Smoothing by averaging with the box blur kernel
im = Image.open('C:/Users/kalpesh/Desktop/msc-1/dsip/Asip/lizard.jpg')
```

```
pylab.figure(figsize=(20,7))
pylab.subplot(1,3,1), pylab.imshow(im), pylab.title('Original Image', size=30),pylab.axis('off')
for n in [3,5]:
    box_blur_kernel = np.reshape(np.ones(n*n),(n,n)) / (n*n)
    im1 = im.filter(ImageFilter.Kernel((n,n), box_blur_kernel.flatten()))
    pylab.subplot(1,3,(2 if n==3 else 3))
    plot_image(im1, 'Blurred with kernel size = ' + str(n) + 'x' + str(n))
pylab.suptitle('PIL Mean Filter (Box Blur) with different Kernel size',size=30)
pylab.show()
```

OUTPUT




```
 # Smoothing with the Gaussian blur filter
im = Image.open('C:/Users/kalpesh/Desktop/msc-1/dsip/Asip/lizard.jpg')
pylab.figure(figsize=(20,6))
i = 1
for radius in range(1, 4):
    im1 = im.filter(ImageFilter.GaussianBlur(radius))
    pylab.subplot(1,3,i), plot_image(im1, 'radius = ' +str(round(radius,2)))
    i += 1
pylab.suptitle('PIL Gaussian Blur with different Radius', size=20)
pylab.show()
```

OUTPUT

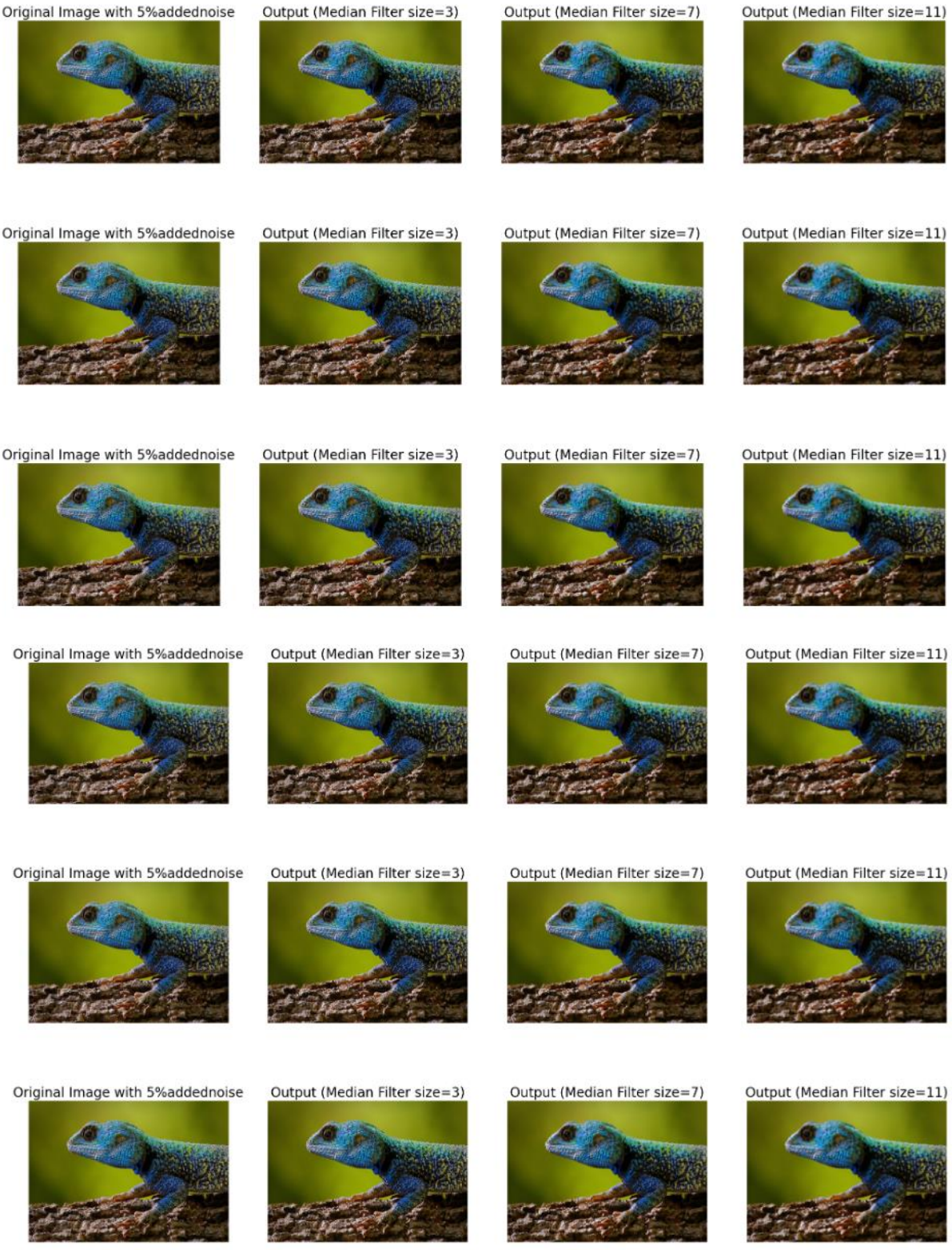

```
#Comparing smoothing with box and Gaussian kernels using SciPy ndimage
from scipy import misc, ndimage
import matplotlib.pylab as pylab
im = imread('C:/Users/kalpesh/Desktop/msc-1/dsip/Asip/lizard.jpg')
k = 7 # 7x7 kernel
```

```
im_box = ndimage.uniform_filter(im, size=(k,k,1))
s = 2 # sigma value
t = (((k - 1)/2)-0.5)/s # truncate parameter value for a kxk gaussian kernel with sigma s
im_gaussian = ndimage.gaussian_filter(im, sigma=(s,s,0), truncate=t)
fig = pylab.figure(figsize=(30,10))
pylab.subplot(131), plot_image(im, 'original image')
pylab.subplot(132), plot_image(im_box, 'with the box filter')
pylab.subplot(133), plot_image(im_gaussian, 'with the gaussian filter')
pylab.show()
```

OUTPUT


```
# Using the median filter
i = 1
pylab.figure(figsize=(25,35))
for prop_noise in np.linspace(0.05,0.3,3):
    im = Image.open('C:/Users/kalpesh/Desktop/msc-1/dsip/Asip/lizard.jpg')
# choose 5000 random locations inside image
    n = int(im.width * im.height * prop_noise)
    x, y = np.random.randint(0, im.width, n), np.random.randint(0, im.height, n)
    for (x,y) in zip(x,y):
        im.putpixel((x, y), ((0,0,0) if np.random.rand() < 0.5 else(255,255,255))) # geenrate
salt-and-pepper noise
        im.save('C:/Users/kalpesh/Desktop/msc-1/dsip/Asip' + str(prop_noise) + '.jpg')
        pylab.subplot(6,4,i)
        plot_image(im, 'Original Image with ' + str(int(100*prop_noise)) + '%addednoise' )
        i += 1
        for sz in [3,7,11]:
            im1 = im.filter(ImageFilter.MedianFilter(size=sz))
            pylab.subplot(6,4,i), plot_image(im1, 'Output (Median Filter size=' + str(sz) + ')')
            i += 1
pylab.show()
```

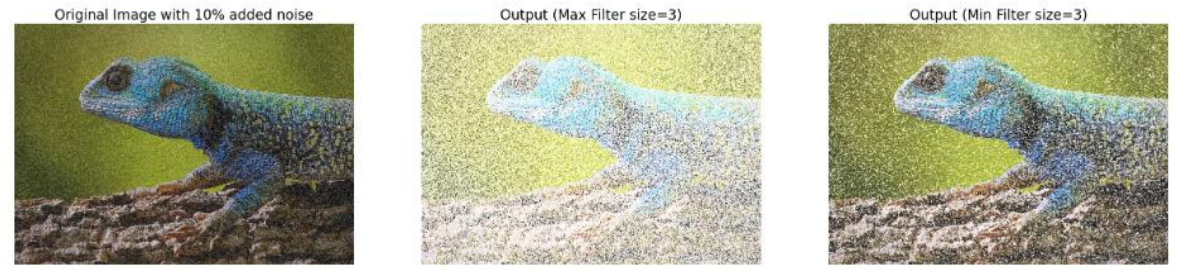
OUTPUT




```
# Using max and min filter
im = Image.open('C:/Users/kalpesh/Desktop/msc-1/dsip/Asip0.3.jpg')
pylab.figure(figsize=(30,10))
sz = 3
pylab.subplot(1,3,1)
plot_image(im, 'Original Image with 10% added noise')
```

```
im1 = im.filter(ImageFilter.MaxFilter(size=sz))
pylab.subplot(1,3,2), plot_image(im1, 'Output (Max Filter size=' + str(sz)+ ')')
im1 = im1.filter(ImageFilter.MinFilter(size=sz))
pylab.subplot(1,3,3), plot_image(im1, 'Output (Min Filter size=' + str(sz)+ ')')
pylab.show()
```

OUTPUT



PRACTICAL-6

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

Theory:

• Output is simply the average of the pixels contained in the neighborhood of the filter mask. Also called averaging filters or low pass filters

• Replacing the value of every pixel in an image by the average of the gray levels in the neighborhood will reduce the “sharp ” transitions in gray levels.

• Sharp transitions

– random noise in the image

– edges of objects in the image

– Smoothing can reduce noises (desirable) and blur edges (undesirable)

$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ box filter

$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ weighted average

Sharpen

Given the fact that we can blur an image, it seems that one should be able to sharpen an image as well. To a certain extent this is possible, although the most commonly used method is really somewhat of a trick. The Sharpen operator actually works by increasing the contrast between areas of transition in an image. This, in turn, is perceived by the eye as an increased sharpness. Sharpening can either be done by using a convolve with a filter such as

−1	−1	−1
−1	9	−1
−1	−1	−1

or via other techniques, such as unsharp masking.

Unsharp masking, an old technique known to photographers, is used to change the relative highpass content in an image by subtracting a blurred (lowpass filtered) version of the image [5]. This can be done optically by first developing an unsharp picture on a negative film and then using this film as a mask in a second development step. Mathematically, unsharp masking can be expressed as

$$\hat{f} = \alpha f - \beta f_{lp}$$

where α and β are positive constants, α ≥ β. When processing digital image data, it is desirable to keep the local mean of the image unchanged. If the coefficients in the lowpass filter f_{lp} are normalized, i.e., their sum equals 1, the following formulation of unsharp masking ensures unchanged local mean in the image:

$$\hat{f} = \frac{1}{\alpha-\beta}(\alpha f - \beta f_{lp}).$$

By expanding the expression in the parentheses $(\alpha f - \beta f_{lp}) = \alpha f_{lp} + \alpha(f - f_{lp}) - \beta f_{lp}$, we can write Eq. (9) as

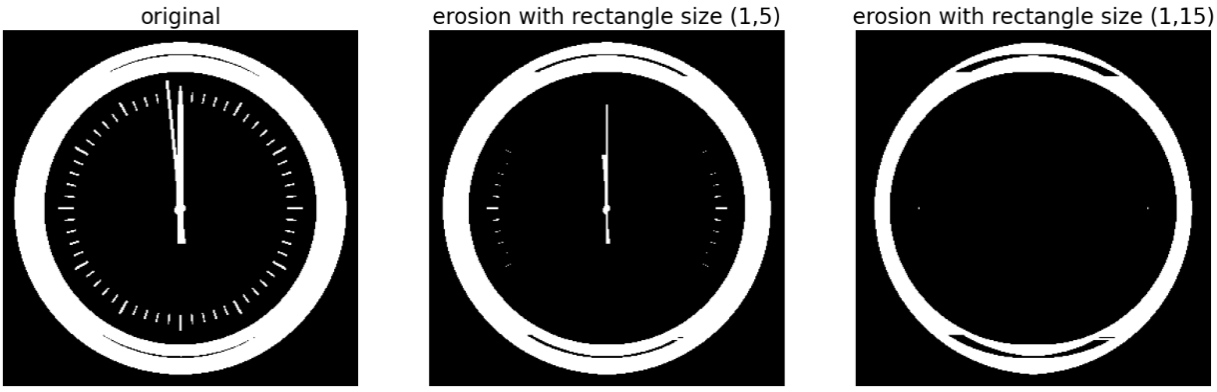
$$\hat{f} = f_{lp} + \frac{\alpha}{\alpha-\beta}(f - f_{lp}),$$

which provides a more intuitive way of writing unsharp masking. Further rewriting yields

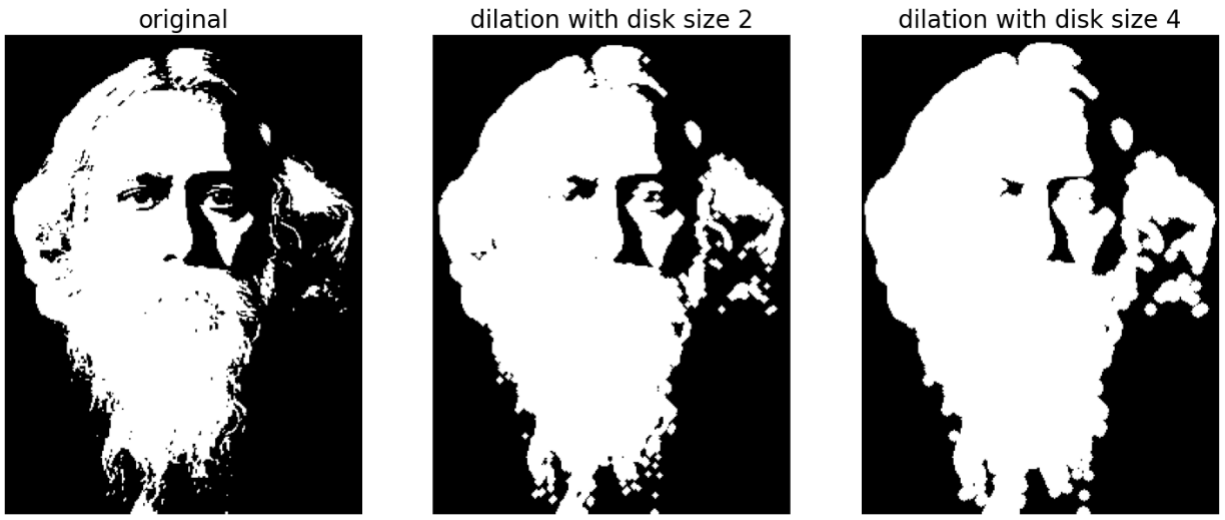
```
!pip install scikit-image
from skimage.io import imread
from skimage.color import rgb2gray
import matplotlib.pylab as pylab
from skimage.morphology import binary_erosion, rectangle
import numpy as np
from scipy.ndimage.morphology import binary_fill_holes
```

```
# erosion
def plot_image(image, title=''):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis('off') # comment this line if you want axis ticks
im=rgb2gray(imread('C:/Users/kalpesh/Desktop/msc1/dsip/practicals/images/images/clock
2.jpg'))
im[im <= 0.5] = 0 # create binary image with fixed threshold 0.5
```

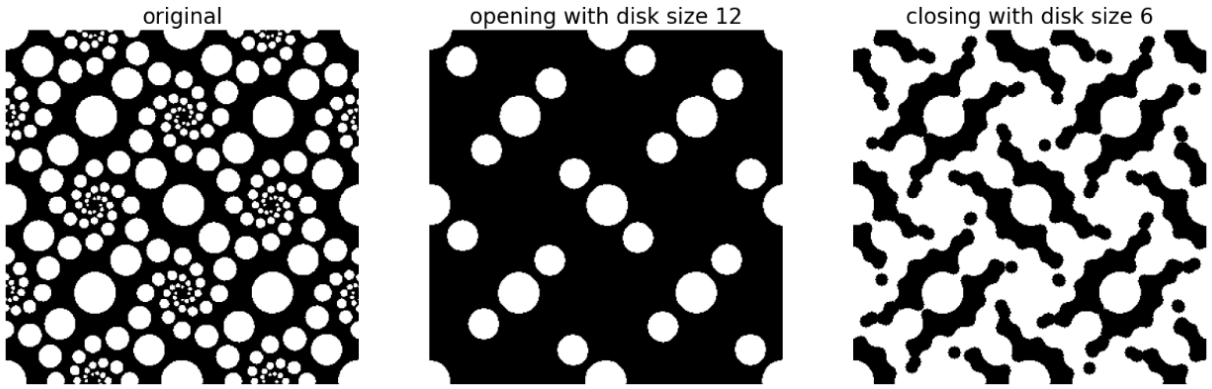
```
im[im > 0.5] = 1
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(1,3,1), plot_image(im, 'original')
im1 = binary_erosion(im, rectangle(1,5))
pylab.subplot(1,3,2), plot_image(im1, 'erosion with rectangle size (1,5)')
im1 = binary_erosion(im, rectangle(1,15))
pylab.subplot(1,3,3), plot_image(im1, 'erosion with rectangle size (1,15)')
pylab.show()
```




```
# Dialation
from skimage.morphology import binary_dilation, disk
from skimage import img_as_float
im = img_as_float(imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/tagore.png'))
im = 1 - im[...,3]
im[im <= 0.5] = 0
im[im > 0.5] = 1
pylab.gray()
pylab.figure(figsize=(18,9))
pylab.subplot(131)
pylab.imshow(im)
pylab.title('original', size=20)
pylab.axis('off')
for d in range(1,3):
    pylab.subplot(1,3,d+1)
    im1 = binary_dilation(im, disk(2*d))
    pylab.imshow(im1)
    pylab.title('dilation with disk size ' + str(2*d), size=20)
    pylab.axis('off')
pylab.show()
```

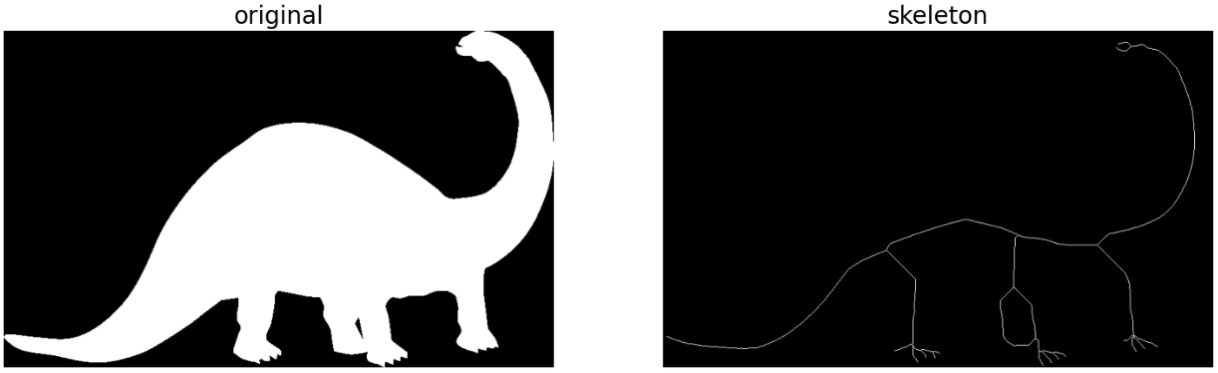



```
# Opening and closing
from skimage.morphology import binary_opening, binary_closing,
binary_erosion,binary_dilation, disk
im= rgb2gray(imread('C:/Users/kalpesh/Desktop/msc-
1/dsip/practicals/images/images/circles.jpg'))
im[im <= 0.5] = 0
im[im > 0.5] = 1
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(1,3,1), plot_image(im, 'original')
im1 = binary_opening(im, disk(12))
pylab.subplot(1,3,2), plot_image(im1, 'opening with disk size ' + str(12))
im1 = binary_closing(im, disk(6))
pylab.subplot(1,3,3), plot_image(im1, 'closing with disk size ' + str(6))
pylab.show()
```

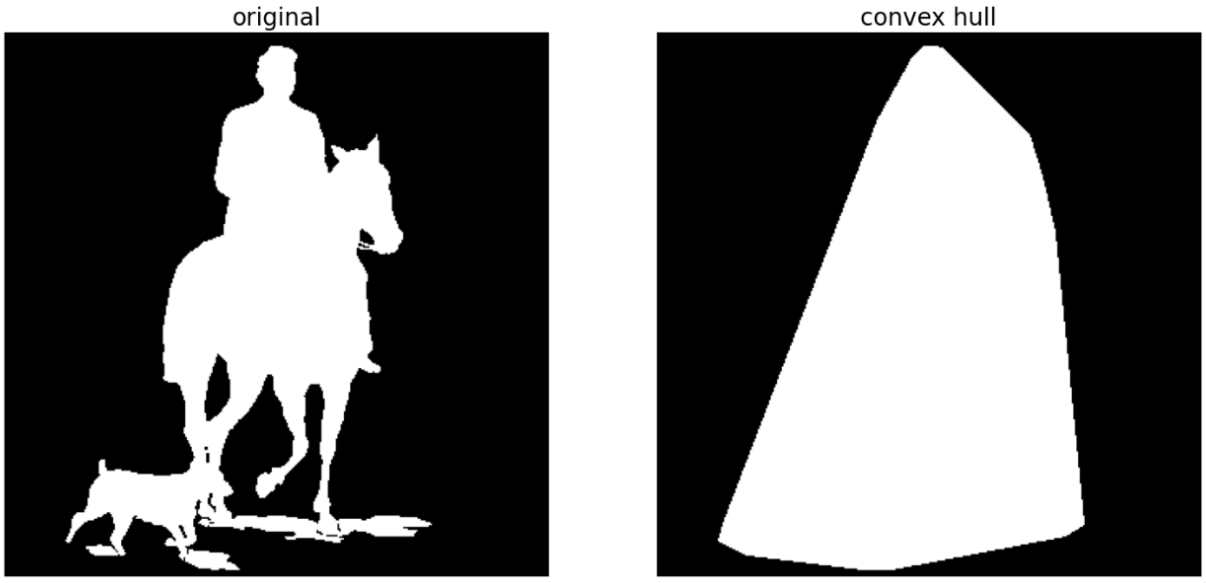



```
# Skeletonizing
def plot_images_horizontally(original, filtered, filter_name, sz=(18,7)):
    pylab.gray()
    pylab.figure(figsize = sz)
    pylab.subplot(1,2,1), plot_image(original, 'original')
    pylab.subplot(1,2,2), plot_image(filtered, filter_name)
    pylab.show()
from skimage.morphology import skeletonize
```

```
im = img_as_float(imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/dynasaur.png')[...,3])
threshold = 0.5
im[im <= threshold] = 0
im[im > threshold] = 1
skeleton = skeletonize(im)
plot_images_horizontally(im, skeleton, 'skeleton',sz=(18,9))
```

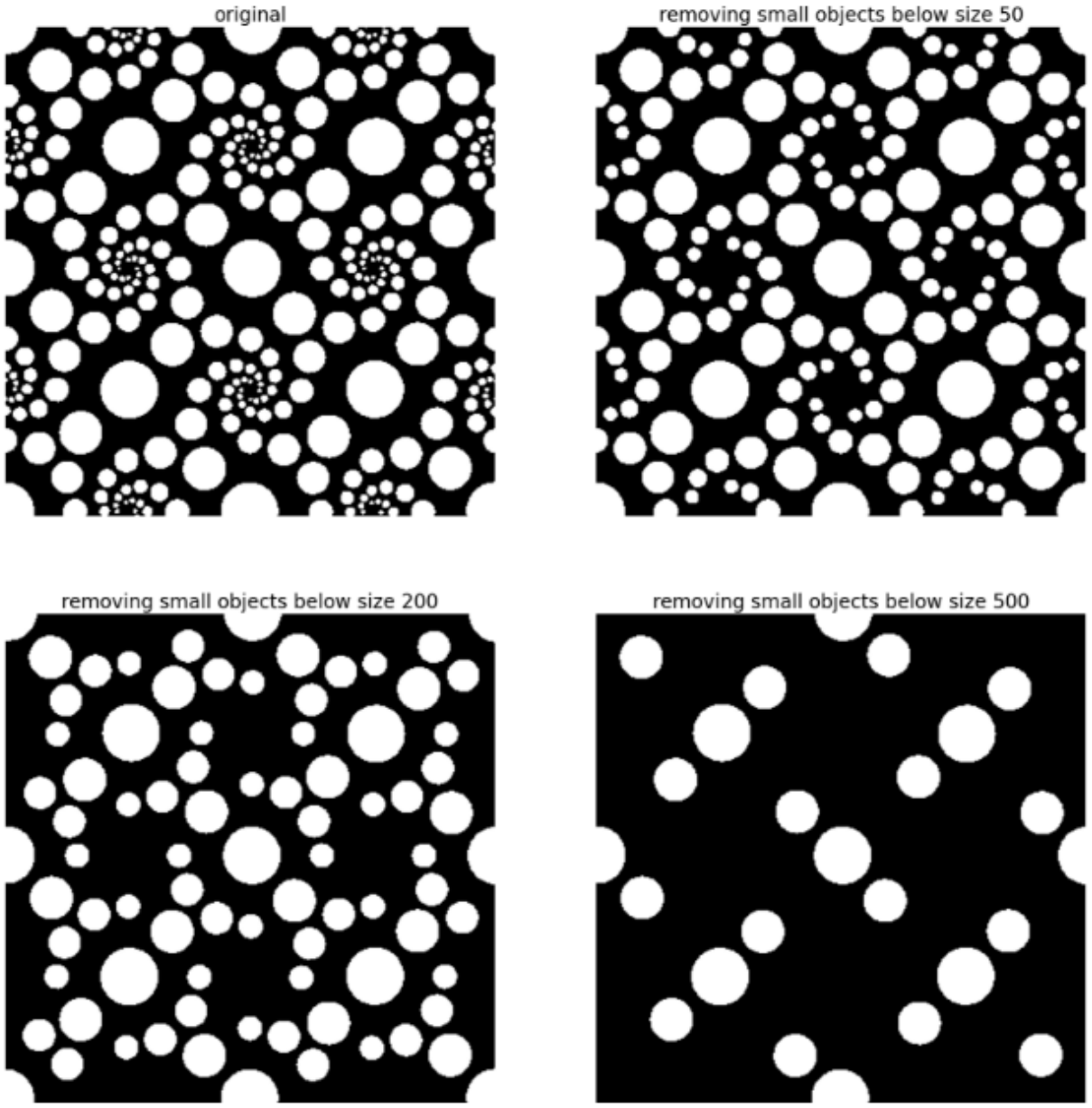



```
 # Computing the convex hull
from skimage.morphology import convex_hull_image
im = rgb2gray(imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/horse-dog.jpg'))
threshold = 0.5
im[im < threshold] = 0 # convert to binary image
im[im >= threshold] = 1
chull = convex_hull_image(im)
plot_images_horizontally(im, chull, 'convex hull', sz=(18,9))
```



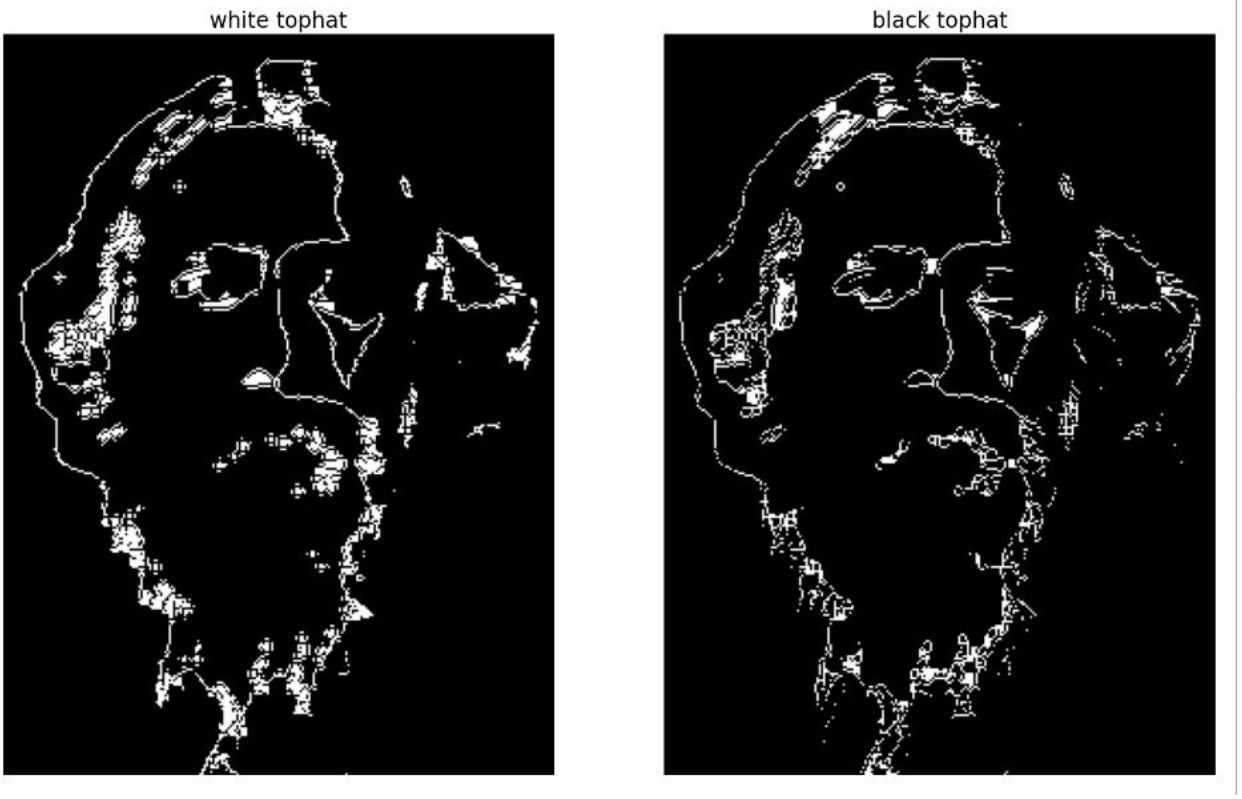

```
# Removing small objects
from skimage.morphology import remove_small_objects
im=rgb2gray(imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/circles.jpg'))
im[im > 0.5] = 1 # create binary image by thresholding with fixed threshold
0.5
5
```

```
im[im <= 0.5] = 0
im = im.astype(np.bool)
pylab.figure(figsize=(20,20))
pylab.subplot(2,2,1), plot_image(im, 'original')
i = 2
for osz in [50, 200, 500]:
    im1 = remove_small_objects(im, osz, connectivity=1)
    pylab.subplot(2,2,i), plot_image(im1, 'removing small objects below size '+ str(osz))
    i += 1
pylab.show()
```

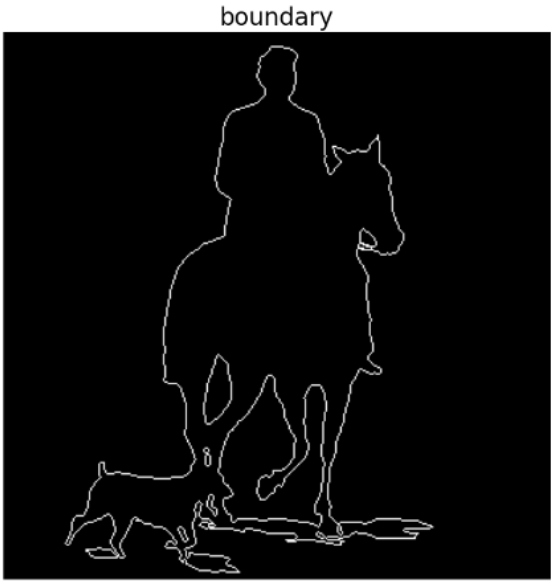



```
#White and black top-hats
from skimage.morphology import white_tophat, black_tophat, square
im = imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/tagore.png')[...,3]
im[im <= 0.5] = 0
im[im > 0.5] = 1
```

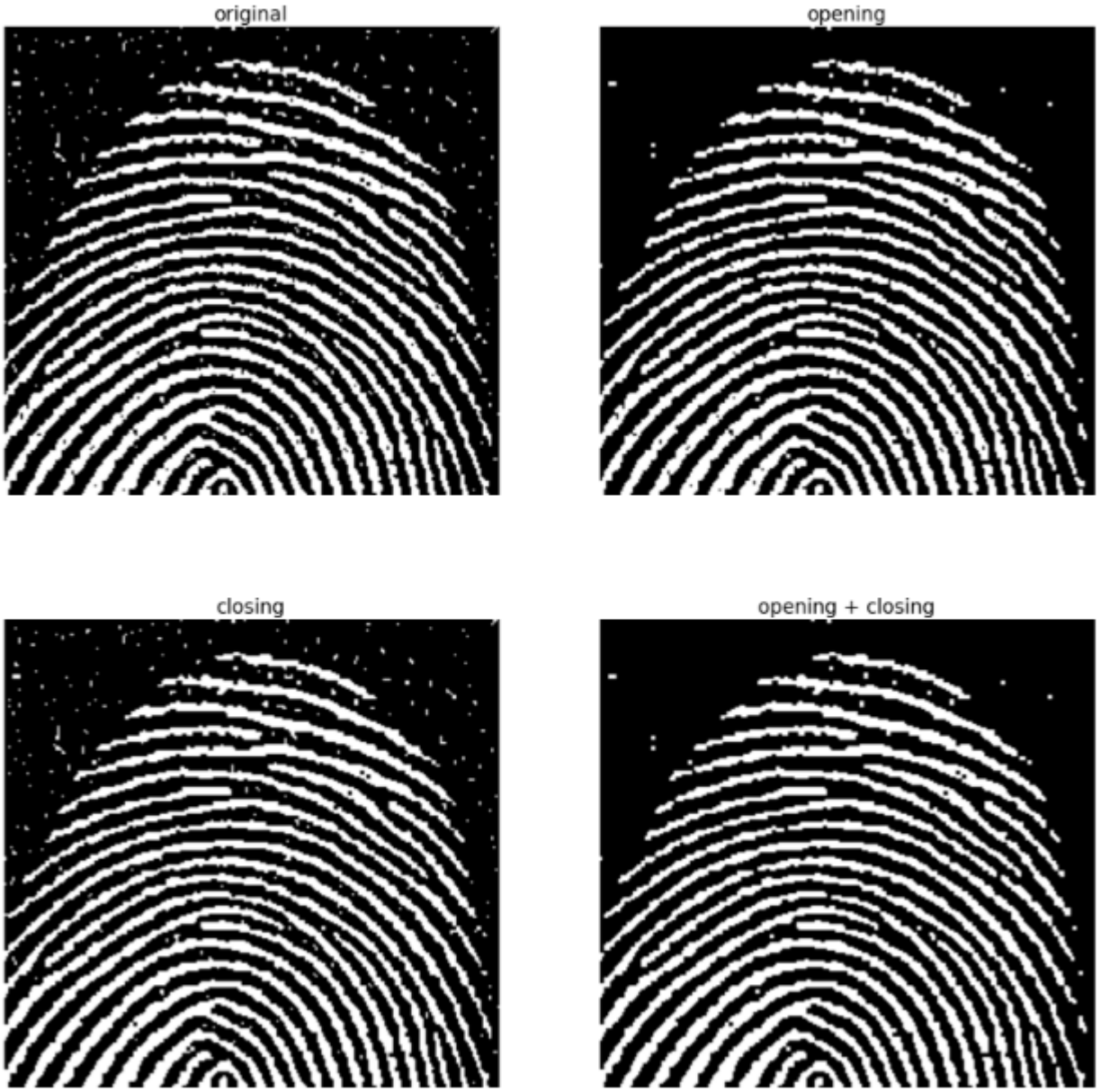
```
im1 = white_tophat(im, square(5))
im2 = black_tophat(im, square(5))
pylab.figure(figsize=(20,15))
pylab.subplot(1,2,1), plot_image(im1, 'white tophat')
pylab.subplot(1,2,2), plot_image(im2, 'black tophat')
pylab.show()
```




```
#Extracting the boundary
from skimage.morphology import binary_erosion
im = rgb2gray(imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/horse-dog.jpg'))
threshold = 0.5
im[im < threshold] = 0
im[im >= threshold] = 1
boundary = im - binary_erosion(im)
plot_images_horizontally(im, boundary, 'boundary',sz=(18,9))
```




```
#Fingerprint cleaning with opening and closing
im = rgb2gray(imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/fingerprint.jpg'))
im[im <= 0.5] = 0 # binarize
im[im > 0.5] = 1
im_o = binary_opening(im, square(2))
im_c = binary_closing(im, square(2))
im_oc = binary_closing(binary_opening(im, square(2)), square(2))
pylab.figure(figsize=(20,20))
pylab.subplot(221), plot_image(im, 'original')
pylab.subplot(222), plot_image(im_o, 'opening')
pylab.subplot(223), plot_image(im_c, 'closing')
pylab.subplot(224), plot_image(im_oc, 'opening + closing')
pylab.show()
```




```
# Grayscale operations
from skimage.morphology import dilation, erosion, closing, opening, square
im = imread('C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/zebras.jpg')
im = rgb2gray(im)
struct_elem = square(5)
eroded = erosion(im, struct_elem)
plot_images_horizontally(im, eroded, 'erosion')
```


```
dilated = dilation(im, struct_elem)
plot_images_horizontally(im, dilated, 'dilation')
```

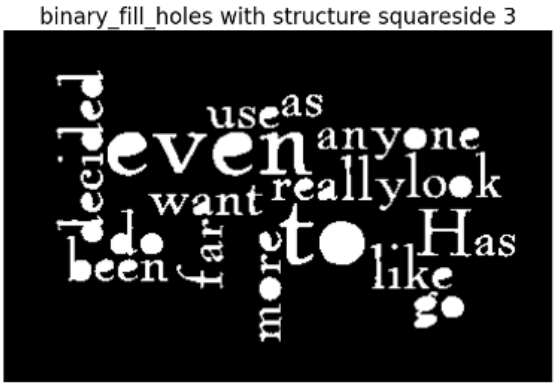

```
opened = opening(im, struct_elem)
plot_images_horizontally(im, opened, 'opening')
```

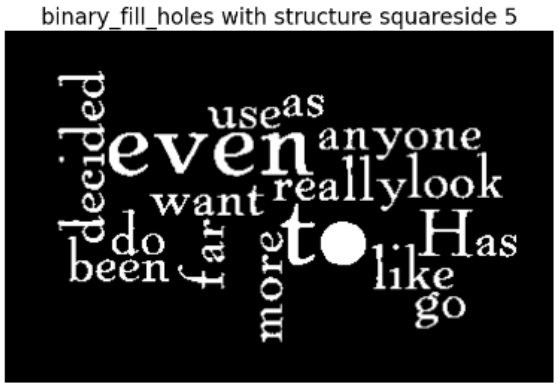

```
closed = closing(im, struct_elem)
plot_images_horizontally(im, closed, 'closing')
```

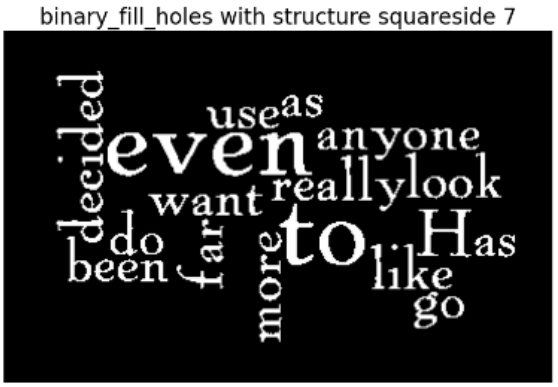

```
# Morphological contrast enhancement
from skimage.filters.rank import enhance_contrast
from skimage import exposure
def plot_gray_image(ax, image, title):
    ax.imshow(image, cmap=pylab.cm.gray),
```

```
    ax.set_title(title), ax.axis('off')
    ax.set_adjustable('box')
image=rgb2gray(imread('C:/Users/kalpesh/Desktop/msc-
1/dsip/practicals/images/images/squirrel.jpg'))
sigma = 0.05
noisy_image = np.clip(image + sigma * np.random.standard_normal(image.shape),0, 1)
enhanced_image = enhance_contrast(noisy_image, disk(5))
equalized_image = exposure.equalize_adapthist(noisy_image)
fig, axes = pylab.subplots(1, 3, figsize=[18, 7], sharex='row',sharey='row')
axes1, axes2, axes3 = axes.ravel()
plot_gray_image(axes1, noisy_image, 'Original')
plot_gray_image(axes2, enhanced_image, 'Local morphological contrastenhancement')
plot_gray_image(axes3, equalized_image, 'Adaptive Histogram equalization')
```


```
# Filling holes in binary objects
from scipy.ndimage.morphology import binary_fill_holes
im=rgb2gray(imread('file:///C:/Users/kalpesh/Desktop/msc-
1/dsip/practicals/images/images/OIP.jpg'))
im[im <= 0.5] = 0
im[im > 0.5] = 1
pylab.figure(figsize=(20,15))
pylab.subplot(221), pylab.imshow(im), pylab.title('original', size=20),pylab.axis('off')
i = 2
for n in [3,5,7]:
    pylab.subplot(2, 2, i)
    im1 = binary_fill_holes(im, structure=np.ones((n,n)))
    pylab.imshow(im1), pylab.title('binary_fill_holes with structure squareside ' + str(n),
size=20)
    pylab.axis('off')
    i += 1
pylab.show()
```





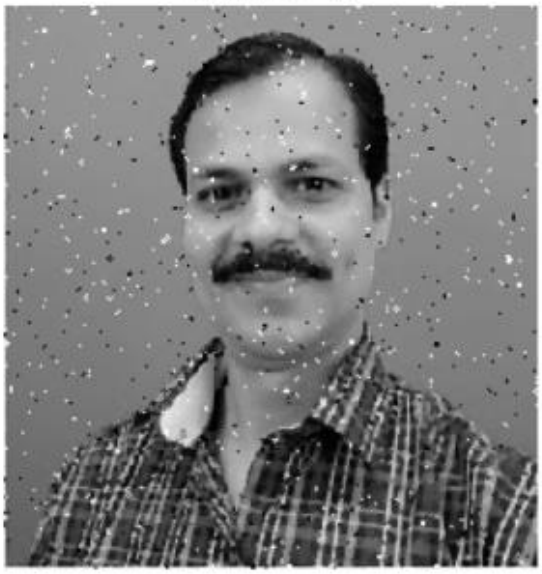



```
 # Noise removal with the median filter
from skimage.filters.rank import median
from skimage.morphology import disk
noisy_image = (rgb2gray(imread('file:///C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/rkm.jpg'))*255).astype(np.uint8)
noise = np.random.random(noisy_image.shape)
noisy_image[noise > 0.9] = 255
noisy_image[noise < 0.1] = 0
fig, axes = pylab.subplots(2, 2, figsize=(10, 10), sharex=True, sharey=True)
axes1, axes2, axes3, axes4 = axes.ravel()
plot_gray_image(axes1, noisy_image, 'Noisy image')
plot_gray_image(axes2, median(noisy_image, disk(1)), 'Median $r=1$')
plot_gray_image(axes3, median(noisy_image, disk(5)), 'Median $r=5$')
plot_gray_image(axes4, median(noisy_image, disk(20)), 'Median $r=20$')
from skimage.filters.rank import median
from skimage.morphology import disk
noisy_image=(rgb2gray(imread('file:///C:/Users/kalpesh/Desktop/msc-1/dsip/practicals/images/images/rkm.jpg'))*255).astype(np.uint8)
noise = np.random.random(noisy_image.shape)
noisy_image[noise > 0.9] = 255
noisy_image[noise < 0.1] = 0
fig, axes = pylab.subplots(2, 2, figsize=(10, 10), sharex=True, sharey=True)
axes1, axes2, axes3, axes4 = axes.ravel()
plot_gray_image(axes1, noisy_image, 'Noisy image')
plot_gray_image(axes2, median(noisy_image, disk(1)), 'Median $r=1$')
plot_gray_image(axes3, median(noisy_image, disk(5)), 'Median $r=5$')
```

plot_gray_image(axes4, median(noisy_image, disk(20)), 'Median $r=20$')

Noisy image

Median $r = 1$



Median $r = 5$

Median $r = 20$

PRACTICAL-7

AIM : - Write a program to Apply edge detection techniquessuch as Sobel and Canny to extract meaningful information from the given image samples.

Theory:

SOBEL

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary Feldman, colleagues at the Stanford Artificial Intelligence Laboratory (SAIL). Sobel and Feldman presented the idea of an "Isotropic 3 × 3 Image Gradient Operator" at a talk at SAIL in 1968.[1] Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:[2]

{\displaystyle \mathbf {G} _{x}={\begin{bmatrix}+1&0&-1\\+2&0&-2\\+1&0&-1\end{bmatrix}}*\mathbf {A} \quad {\mbox{and}}\quad \mathbf {G} _{y}={\begin{bmatrix}+1&+2&+1\\0&0&0\\-1&-2&-1\end{bmatrix}}*\mathbf {A} }

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

where {\displaystyle *} here denotes the 2-dimensional signal processing convolution operation.

Canny edge detection

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:

Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible

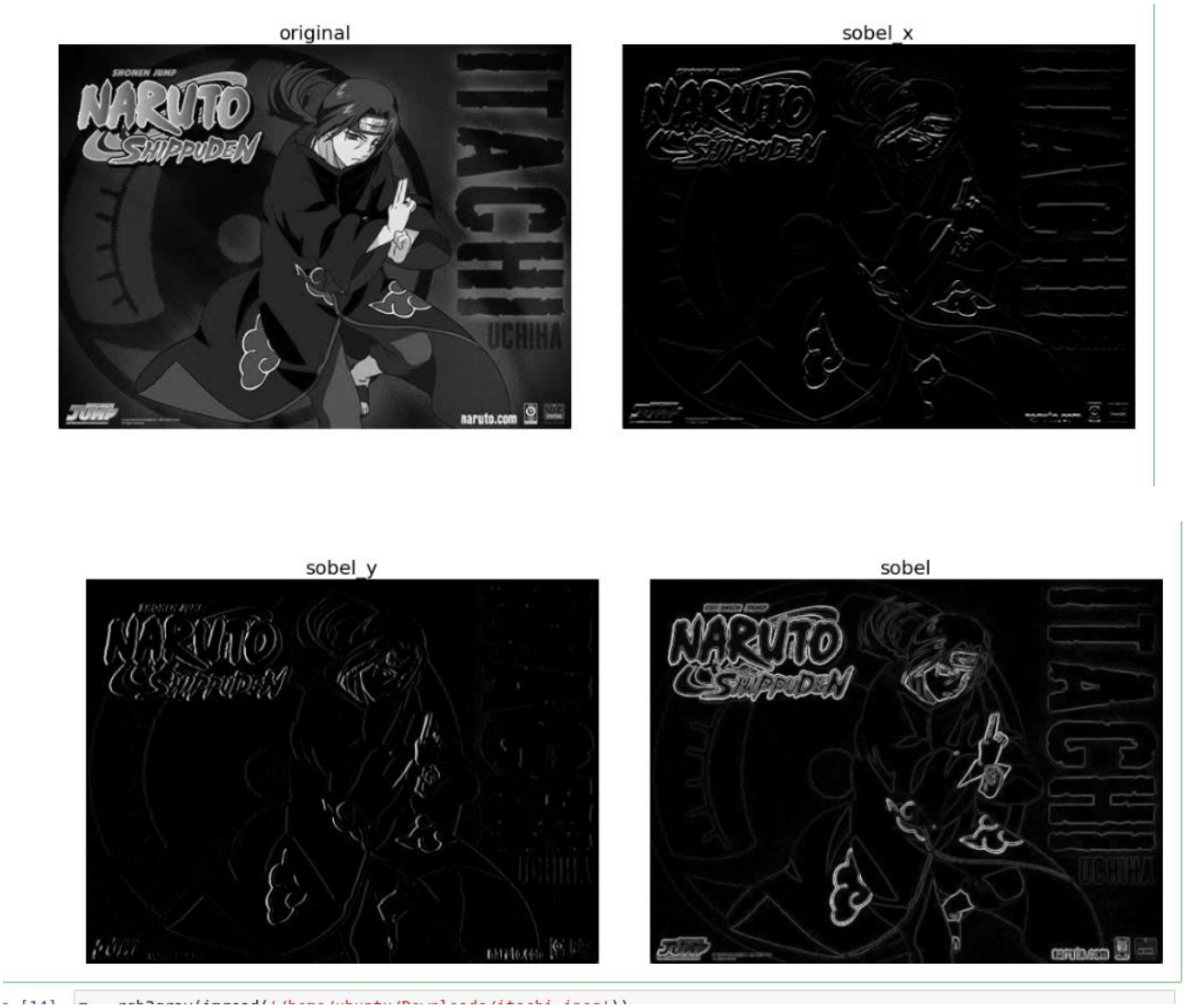
The edge point detected from the operator should accurately localize on the center of the

edge.
A given edge in the image should only be marked once, and where possible, image noise should not create false edges.
To satisfy these requirements Canny used the calculus of variations – a technique which finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.
Among the edge detection methods developed so far, Canny edge detection algorithm is one of the most strictly defined methods that provides good and reliable detection. Owing to its optimality to meet with the three criteria for edge detection and the simplicity of process for implementation, it became one of the most popular algorithms for edge detection.

Sobel Edge detector with scikit-image

```
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

im = rgb2gray(imread('/home/ubuntu/Downloads/itachi.jpeg')) # RGB image to grayscale
pylab.gray()
pylab.figure(figsize=(20,18))
pylab.subplot(2,2,1)
plot_image(im, 'original')
pylab.subplot(2,2,2)
edges_x = filters.sobel_h(im)
plot_image(np.clip(edges_x,0,1), 'sobel_x')
pylab.subplot(2,2,3)
edges_y = filters.sobel_v(im)
plot_image(np.clip(edges_y,0,1), 'sobel_y')
pylab.subplot(2,2,4)
edges = filters.sobel(im)
plot_image(np.clip(edges,0,1), 'sobel')
pylab.subplots_adjust(wspace=0.1, hspace=0.1)
pylab.show()
```

OUTPUT

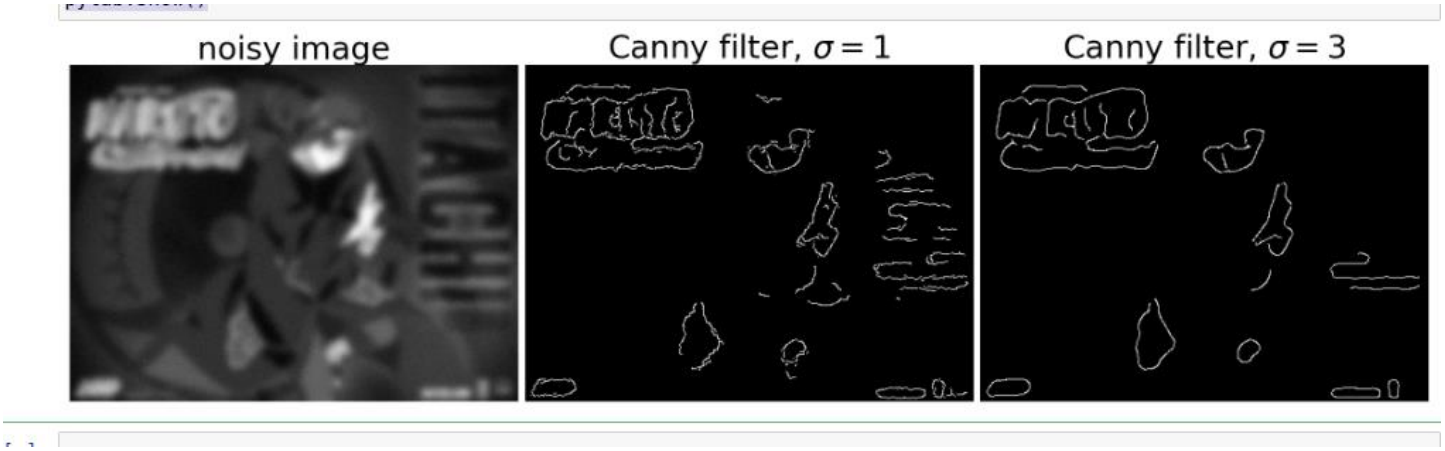


Canny edge detector with scikit -image

```
m = rgb2gray(imread('/home/ubuntu/Downloads/itachi.jpeg'))
im = ndimage.gaussian_filter(im, 4)
im += 0.05 * np.random.random(im.shape)
edges1 = feature.canny(im)
edges2 = feature.canny(im, sigma=2)
fig, (axes1, axes2, axes3) = pylab.subplots(nrows=1, ncols=3, figsize=(30, 12),
sharex=True, sharey=True)
axes1.imshow(im, cmap=pylab.cm.gray), axes1.axis('off'), axes1.set_title('noisy
image', fontsize=50)
```

```
axes2.imshow(edges1, cmap=pylab.cm.gray), axes2.axis('off')
axes2.set_title('Canny filter, $\sigma=1$', fontsize=50)
axes3.imshow(edges2, cmap=pylab.cm.gray), axes3.axis('off')
axes3.set_title('Canny filter, $\sigma=3$', fontsize=50)
fig.tight_layout()
pylab.show()
```

OUTPUT



PRACTICAL-8

AIM :- Write the program to implement various morphologicalimage processing techniques.

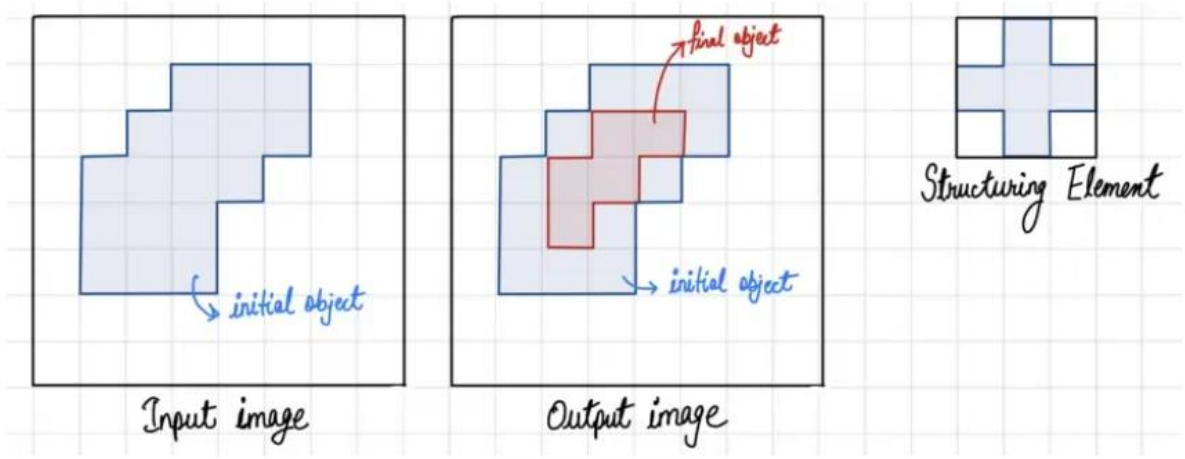
Theory:

1. Erosion

Erosion shrinks the image pixels, or erosion removes pixels on object boundaries. First, we traverse the structuring element over the image object to perform an erosion operation, as shown in Figure 4. The output pixel values are calculated using the following equation.

Pixel (output) = 1 {if FIT}

Pixel (output) = 0 {otherwise}

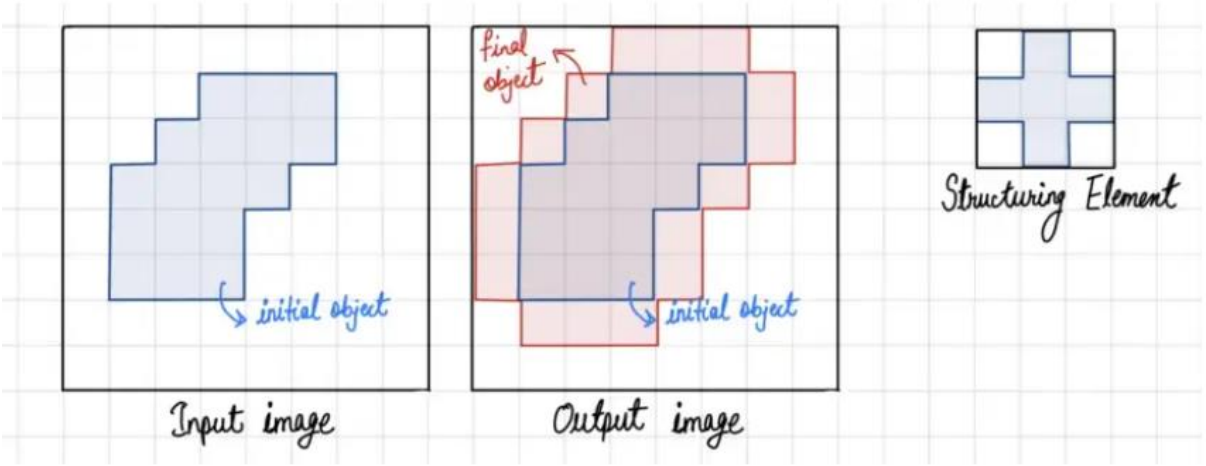


2. Dilation

Dilation expands the image pixels, or it adds pixels on object boundaries. First, we traverse the structuring element over the image object to perform an dilation operation, as shown in Figure 7. The output pixel values are calculated using the following equation.

Pixel (output) = 1 {if HIT}

Pixel (output) = 0 {otherwise}

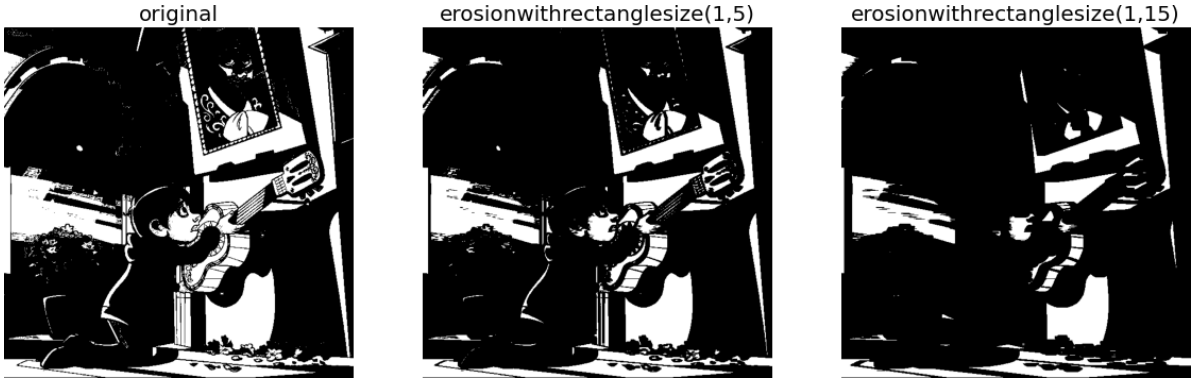


- **Open:** The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations.
- **Close:** The closing operation dilates an image and then erodes the dilated image, using the same structuring element for both operations.

```
!pip install scikit-image
from skimage.io import imread
import numpy as np
from skimage.color import rgb2gray
import matplotlib.pylab as pylab
from skimage.morphology import binary_erosion,rectangle
```

In [4]:

```
#erosion
def plot_image(image,title=''):
        pylab.title(title,size=20),
        pylab.imshow(image)
pylab.axis('off')
#commenthis line if you want axis ticks
im=rgb2gray(imread('../Desktop/College/MSC/MSC-I/DIP/practical/pracimg.jpeg'))
im[im<=0.5]=0
#createbinaryimagewithfixedthreshold0.5
im[im>0.5]
=1
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(1,3,1),plot_image(im,'original')
im1=binary_erosion(im,rectangle(1,5))
pylab.subplot(1,3,2),plot_image(im1,'erosionwithrectangles
ize(1,5)') im1=binary_erosion(im,rectangle(1,15))
pylab.subplot(1,3,3),plot_image(im1,'erosionwithrectangles
ize(1,15)')pylab.show()
```




```
#Dialation
from skimage.morphology import binary_dilation,disk
from skimage import img_as_float
im=img_as_float(imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg'))im=1-im[...,2]
```

```
im[im<=0.5]
=0
im[im>0.5]=
1
pylab.gray()
pylab.figure(figsize=(18
,9)) pylab.subplot(131)
pylab.imshow(im)
pylab.title('original',size
=20)pylab.axis('off')
for d in range(1,3):
    pylab.subplot(1,3,d+1)
    im1=binary_dilation(im,disk(
    2*d))pylab.imshow(im1)
    pylab.title('dilationwithdisksize'+str(2*d),size=20),pylab.axis(off)
pylab.show()
```

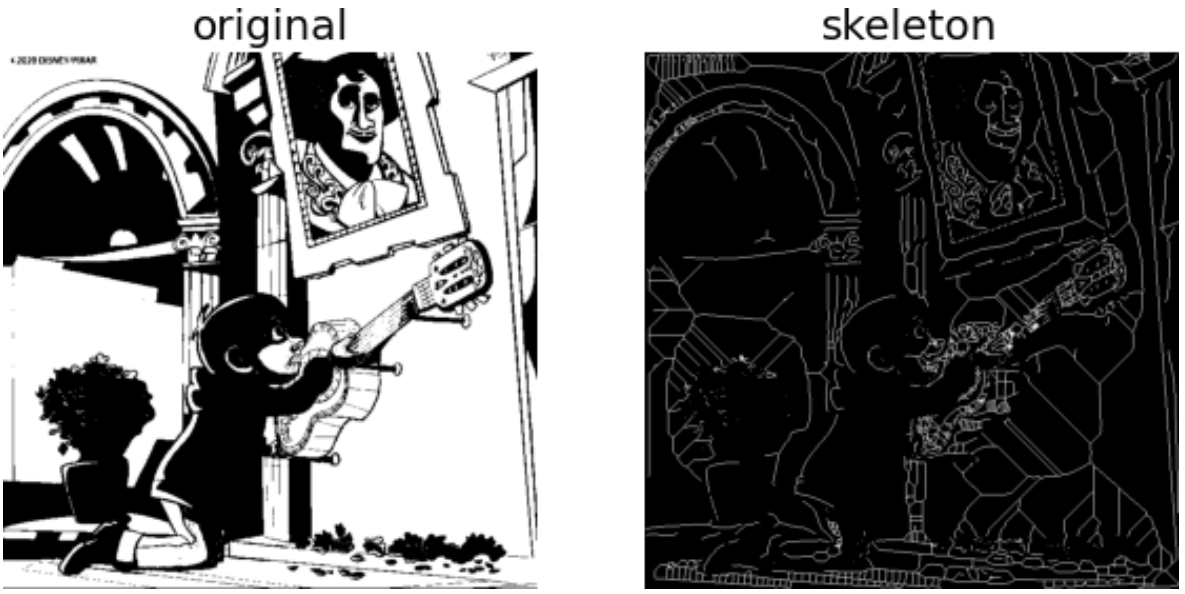

```
from skimage.morphology import
binary_opening,binary_closing,binary_erosion,binary_dilation,disk
im=rgb2gray(imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg')) im[im<=0.5]=0
im[im>0.5]=1
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(1,3,1),plot_image(im,'original')
```

```
im1=binary_opening(im,disk(12))
pylab.subplot(1,3,2),plot_image(im1,'openingwithdisksize'+str(12))
im1=binary_closing(im,disk(6))
pylab.subplot(1,3,3),plot_image(im1,'closingwithdisksize'+str(6))
pylab.show()
```

<Figure size 432x288 with 0 Axes>

```
pylab.subplot(1,2,2),plot_image(filtered,filter
_name)pylab.show()
from skimage.morphology import skeletonize
im=img_as_float(imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg')[...,2])threshold=0.5
im[im<=threshold]=0
im[im>threshold]=1
skeleton=skeletonize(
im)
plot_images_horizontally(im,skeleton,'skeleton',sz=(10,5))
```

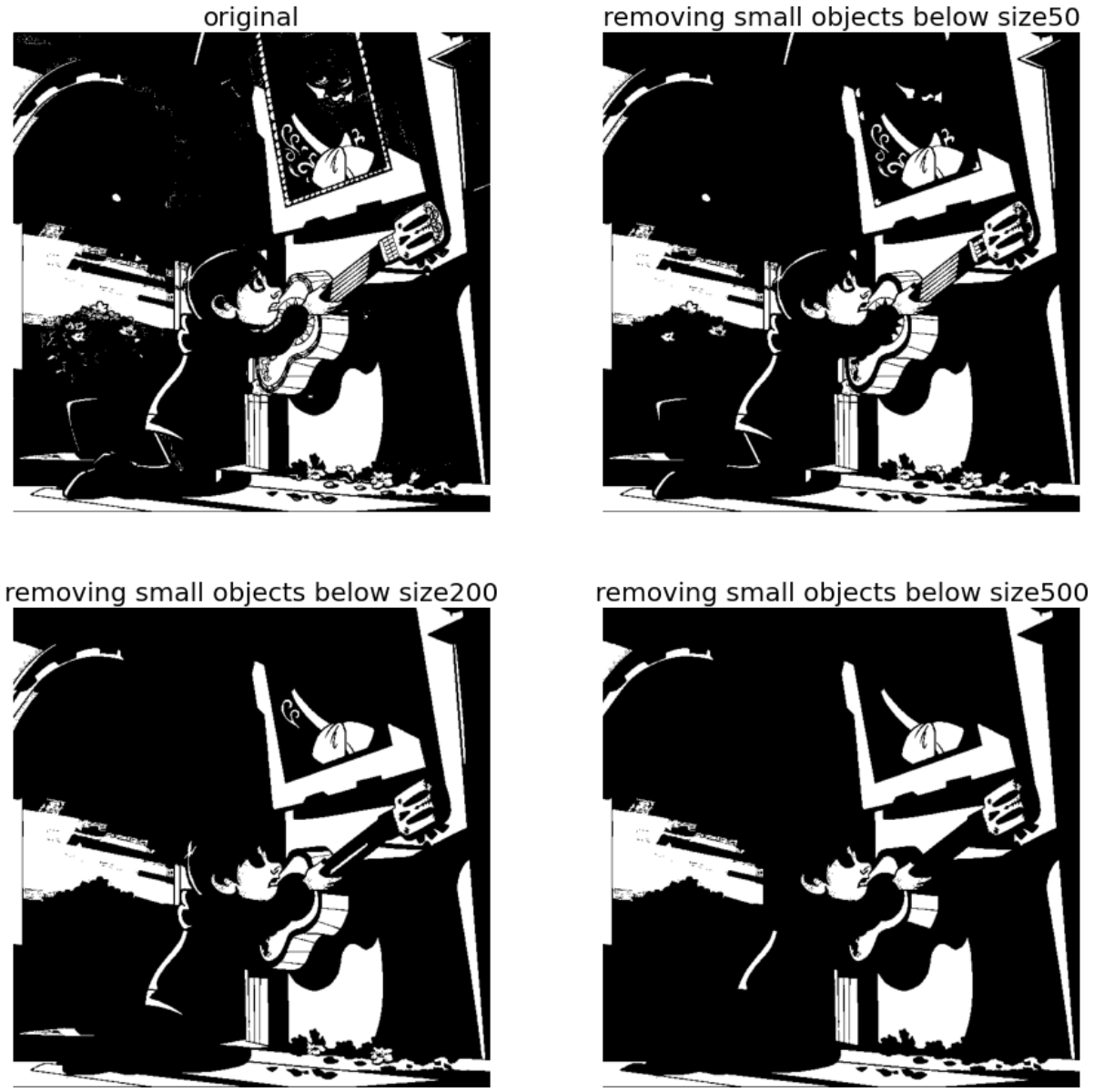
<Figure size 432x288 with 0 Axes>




```
from skimage.morphology import convex_hull_image
im=rgb2gray(imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg'))threshold=0.5
im[im<threshold]=0
#converttobinaryimage
im[im>=threshold]=1
chull=convex_hull_image(im)
plot_images_horizontally(im,chull,'convexhull',sz=(18,9))
```

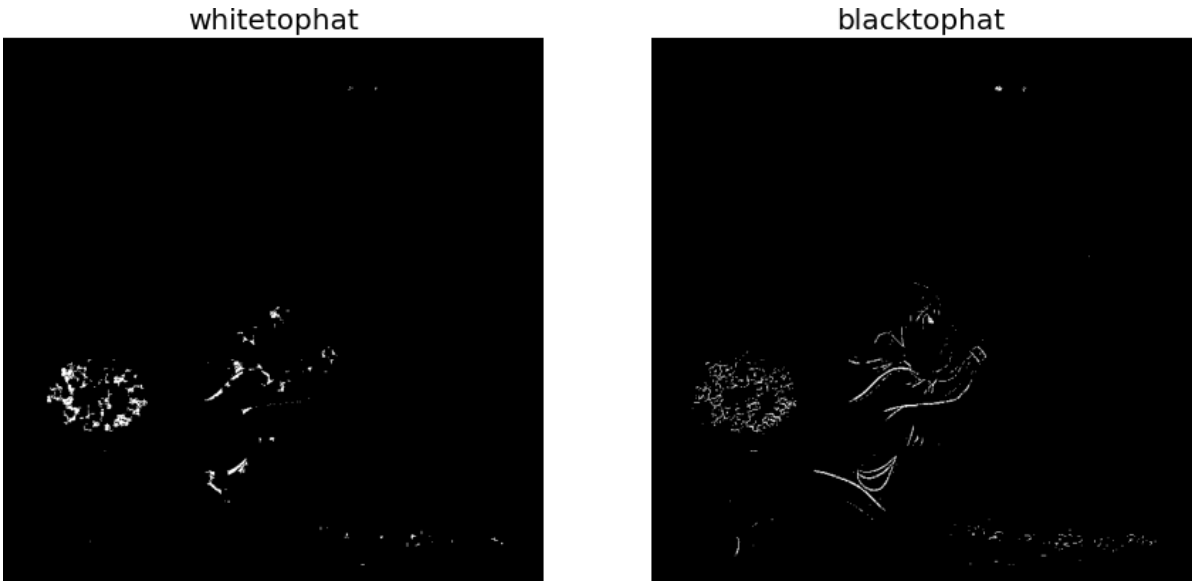
<Figure size 432x288 with 0 Axes>


```
#Removingsmallobjects
from skimage.morphology import remove_small_objects
im=rgb2gray(imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg'))im[im>0.5]=1
#createbinaryimagebythresholdingwithfixedthreshold
0.5
im[im<=0.5]=0
im=im.astype(np.bool)
pylab.figure(figsize=(15,
15))
pylab.subplot(2,2,1),plot_image(im,'or
iginal')i=2
for osz in[50,200,500]:
   im1=remove_small_objects(im, osz,
   connectivity=1)
   pylab.subplot(2,2,i),plot_image(im1,'removing small objects below size'
   +str(osz
   ))i+=1
pylab.show()
```



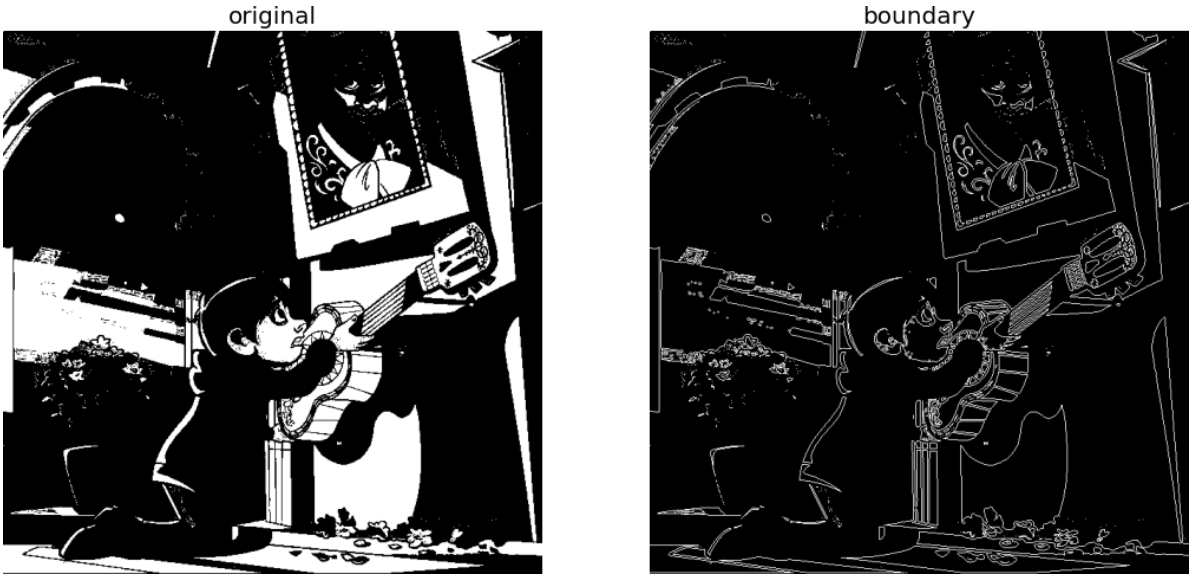

```
#Whiteandblacktop-hats
from skimage.morphology import
white_tophat,black_tophat,square
im=imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg')[...,2]im[im<=0.5]=0
im[im>0.5]=1
im1=white_tophat(im,squar
e(5))
im2=black_tophat(im,squar
e(5))
pylab.figure(figsize=(15,15
))
pylab.subplot(1,2,1),plot_image(im1,'white
tophat')
```

pylab.subplot(1,2,2),plot_image(im2,'black tophat')pylab.show()



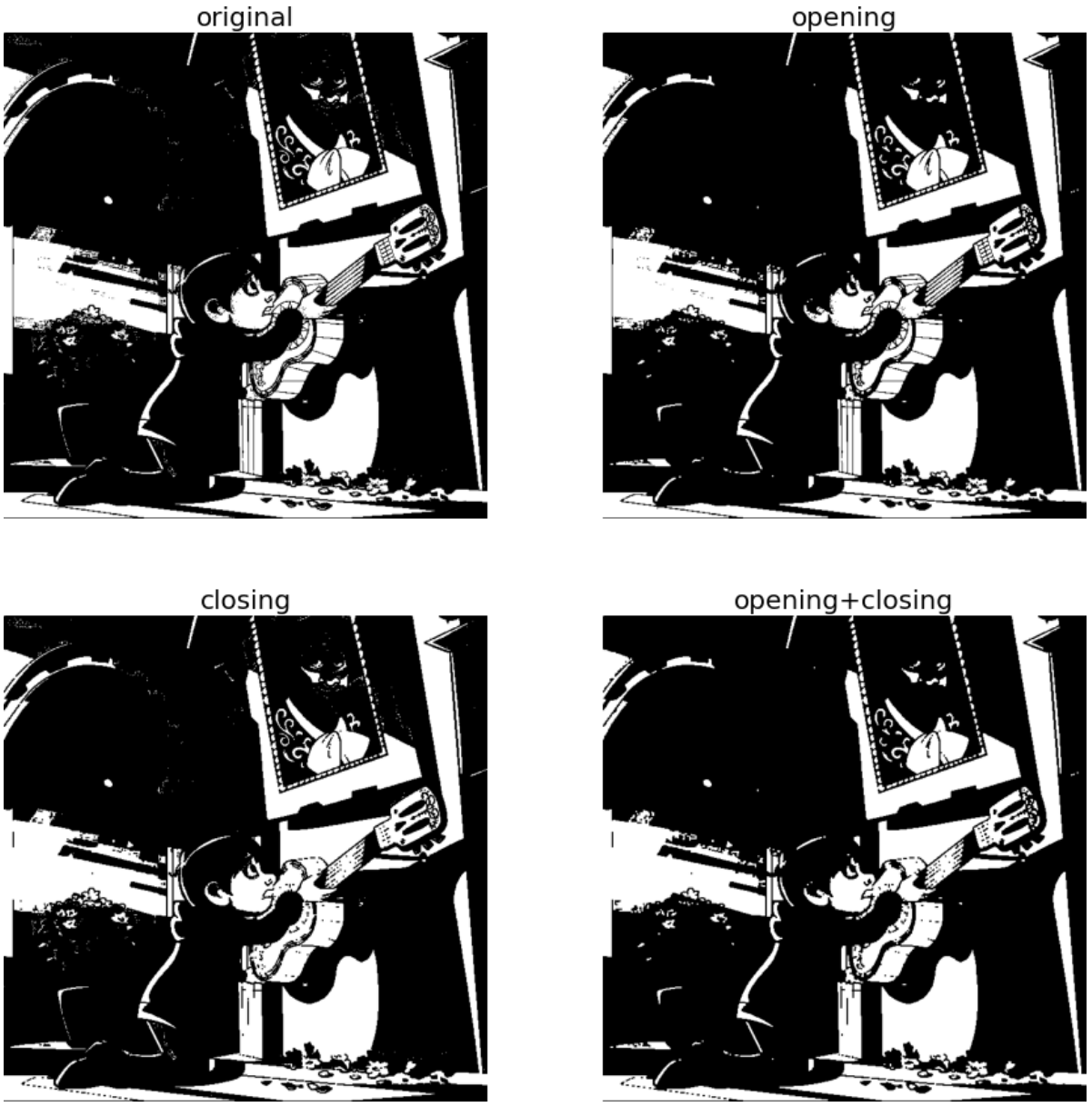
from skimage.morphology **import** binary_erosion
im=rgb2gray(imread('../Desktop/College/MSC/MSC-I/DIP/practical/pracimg.jpeg')) threshold=0.5
im[im<threshold]=0
im[im>=threshold]=1 boundary=im - binary_erosion(im)
plot_images_horizontally(im,boundary,'boundary',sz=(18,9))

<Figure size 432x288 with 0 Axes>



#Fingerprintcleaningwithopeningandclosing
im=rgb2gray(imread('../Desktop/College/MSC/MSC-I/DIP/practical/pracimg.jpeg'))
im[im<=0.5]=0

```
#binarize
im[im>0.5]=1
im_o=binary_opening(im,square(2))
im_c=binary_closing(im,square(2))
im_oc=binary_closing(binary_opening(im,square(2)),square(2))
pylab.figure(figsize=(15,15))
pylab.subplot(221),plot_image(im,'original')
pylab.subplot(222),plot_image(im_o,'opening')
pylab.subplot(223),plot_image(im_c,'closing')
pylab.subplot(224),plot_image(im_oc,'opening+closing')
pylab.show()
```



#Grayscale operations

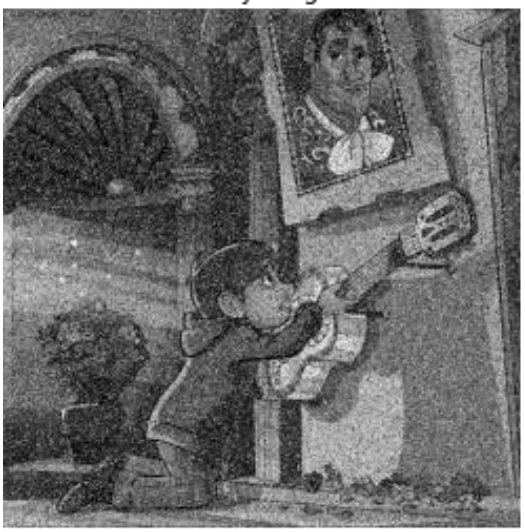
from skimage.morphology **import** dilation,erosion,closing,opening,square

im=imread('../Desktop/College/MSC/MSC-

```
I/DIP/practical/pracimg.jpeg')im=rgb2gray(im)
struct_elem=square(5)
eroded=erosion(im,struct_elem)
plot_images_horizontally(im,eroded,'er
osion')
```

<Figure size 432x288 with 0 Axes>

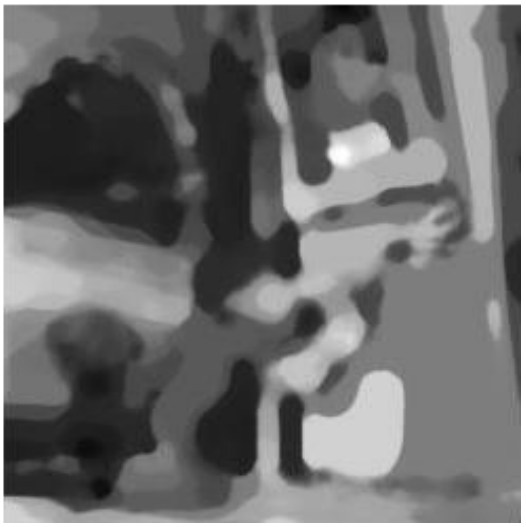
Noisyimage



Medianr = 1

Medianr = 5

Medianr = 20



PRATICAL-9

AIM :- Write the program to extract image features by implementing methods like corner and blob detectors, HoG and Haar features.

```
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
#TheLoGandDoGfilters
from scipy.signal import convolve2d
#fromscipy.miscimportimread
from scipy.ndimage import gaussian_filter
from numpy import pi
def plot_kernel (kernel,s,name):
    pylab.imshow(kernel,cmap='Yl
    OrRd')
def LOG(k=12,s=3):
    n=2*k+1
#sizeofthekernel
    kernel=np.zeros((n,n))
    for i in range (n):
        for j in range(n):
            kernel[i,j]=-(1-((i-k)**2+(j-k)**2)/(2.*s**2))*np.exp(-((i-k)**2+(j-
k)**2)/(2.*s**2))/(pi*s**4)
        kernel=np.round(kernel/np.sqrt((kernel**2).sum()),3)
        return kernel
def DOG(k=12,s=3):
    n=2*k+1
#size of the kernel
    s1,s2=s*np.sqrt(2),s/np.sqrt(2)
    kernel=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            kernel[i,j]=np.exp(-((i-k)**2+(j-k)**2)/(2.*s1**2))/(2*pi*s1**2)-np.exp(-((i-
k)**2+(j-k)**2)/(2.*s2**2))/(2*pi*s2**2)
        kernel=np.round(kernel/np.sqrt((kernel**2).sum()),3)
        return kernel
s=3
#sigmavalueforLoG
img=rgb2gray(imread('../Desktop/College/MSC/MSC-
I/DIP/practical/pracimg.jpeg'))kernel=LOG()
outimg=convolve2d(img,kernel)
```

```
pylab.figure(figsize=(20,20))


pylab.subplot(221),pylab.title('LOG
kernel',size=20),plot_kernel(kernel,s,'DOG')
pylab.subplot(222),pylab.title('outputimagewithLOG',size=20)
pylab.imshow(np.clip(outimg,0,1),cmap='gray')
#clipthepixelvaluesinbetween0and1
kernel=DOG()
outimg=convolve2d(img,D
OG())
pylab.subplot(223),pylab.title('DOG
kernel',size=20),plot_kernel(kernel,s,'DOG')
pylab.subplot(224),pylab.title('output image with DOG',size=20)
pylab.imshow(np.clip(outimg,0,1),cmap='gray')
pylab.show()
```