INDEX

Sr. No.	TOPICS	Page No.
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	4
2	Write program to demonstrate the following aspects of signal on sound/image data 1. Convolution operation 2. Template Matching	9
3	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	11
4	Write a program to apply various enhancements on images using image derivatives by implementing Gradient and Laplacian operations.	18
5	Write a program to implement linear and nonlinear noise smoothing on suitable image or sound signal.	23
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	28
7	Write a program to Apply edge detection techniques such as Sobel and Canny to extract meaningful information from the given image samples	31
8	Write the program to implement various morphological image processing techniques.	33
9	Write the program to extract image features by implementing methods like corner and blob detectors, HoG and Haar features.	42
10	Write the program to apply segmentation for detecting lines, circles, and other shapes/objects. Also, implement edge-based and region-based segmentation.	46

Applied Signal & Image Processing

Practicals

Practical-1:

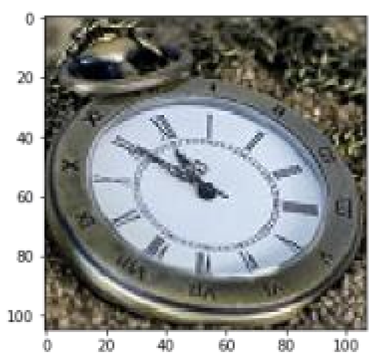
Write program to demonstrate the following aspects of signal processing on suitable data.

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

- **Up-Sampling**

im = Image.open("../images/clock.jpg") # *the original small clock image*

pylab.imshow(im), pylab.show()



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

im1 = im.resize((im.width*5, im.height*5), Image.BILINEAR) # *up-sample with bi-linear interpolation* pylab.figure(figsize=(10,10)), pylab.imshow(im1), pylab.show()

im.resize((im.width*10, im.height*10), Image.BICUBIC).show() # *bi-cubic interpolation* pylab.figure(figsize=(10,10)), pylab.imshow(im1), pylab.show()

- **Down-sampling**

```
im = im.resize((im.width//5, im.height//5))
pylab.figure(figsize=(15,10)), pylab.imshow(im), pylab.show()
```

```
im = Image.open("../images/tajmahal.jpg")
im = im.resize((im.width//5, im.height//5), Image.ANTIALIAS)
pylab.figure(figsize=(15,10)), pylab.imshow(im), pylab.show()
```

```
im = imread('../images/umbc.png')
im1 = im.copy()
pylab.figure(figsize=(20,15))

for i in range(4):
    pylab.subplot(2,2,i+ 1), pylab.imshow(im1, cmap='gray'), pylab.axis('off')
    pylab.title('imagesize = ' + str(im1.shape[1]) + 'x' + str(im1.shape[0]))
    im1 = rescale(im1, scale = 0.5, multichannel=True, anti_aliasing=False)
pylab.subplots_adjust(wspace=0.1, hspace=0.1)
pylab.show()
```

- **FFT with the numpy.fft module**

```
import numpy.fft as fp
im1 = rgb2gray(imread('../images/house.png'))
pylab.figure(figsize=(12,10))
freq1 = fp.fft2(im1)
im1_ = fp.ifft2(freq1).real
```

```
pylab.subplot(2,2,1), pylab.imshow(im1, cmap='gray'), pylab.title('Original Image', size=20)
pylab.subplot(2,2,2), pylab.imshow(20*np.log10( 0.01 +
np.abs(fp.fftshift(freq1))), cmap='gray')
pylab.title('FFT Spectrum Maginitude', size=20)
pylab.subplot(2,2,3), pylab.imshow(np.angle(fp.fftshift(freq1)),cmap='gray')
pylab.title('FFT Phase', size=20)
pylab.subplot(2,2,4), pylab.imshow(np.clip(im1_,0,255), cmap='gray')
pylab.title('Reconstructed Image', size=20)
pylab.show()
```

Image 1:

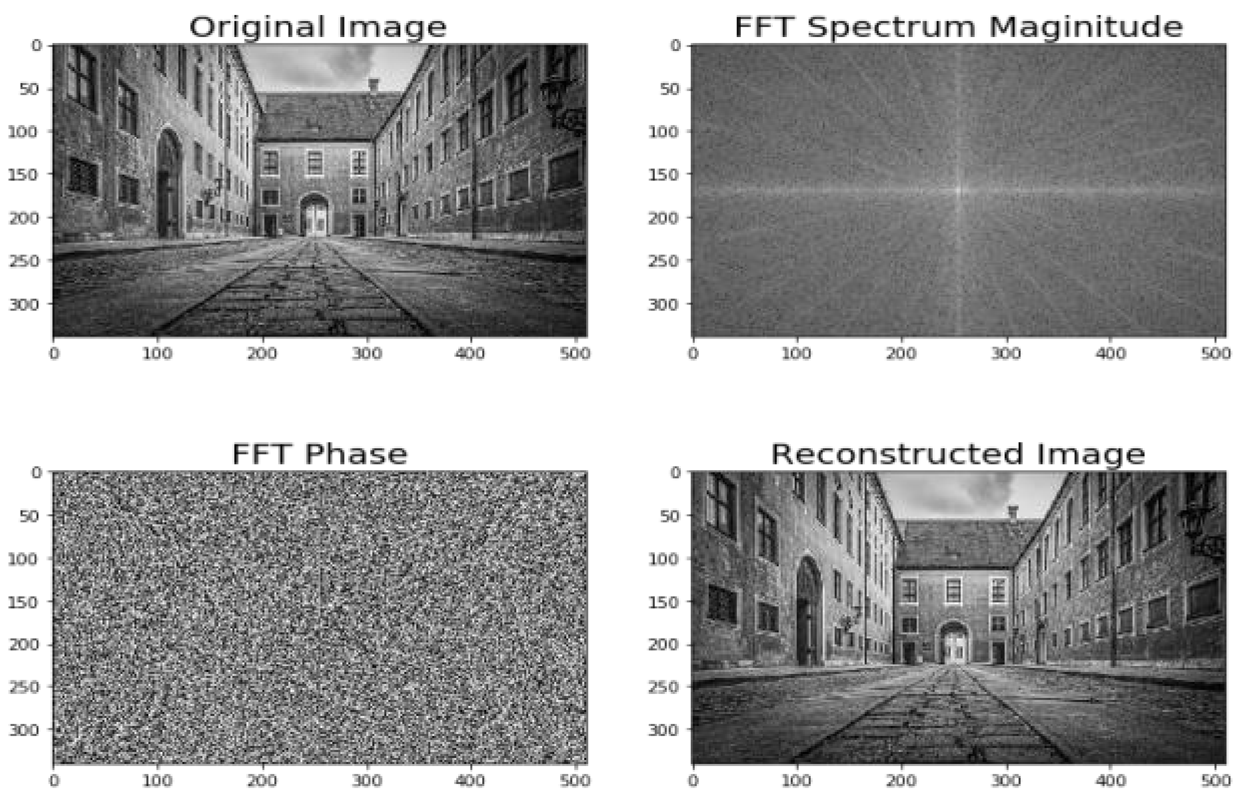
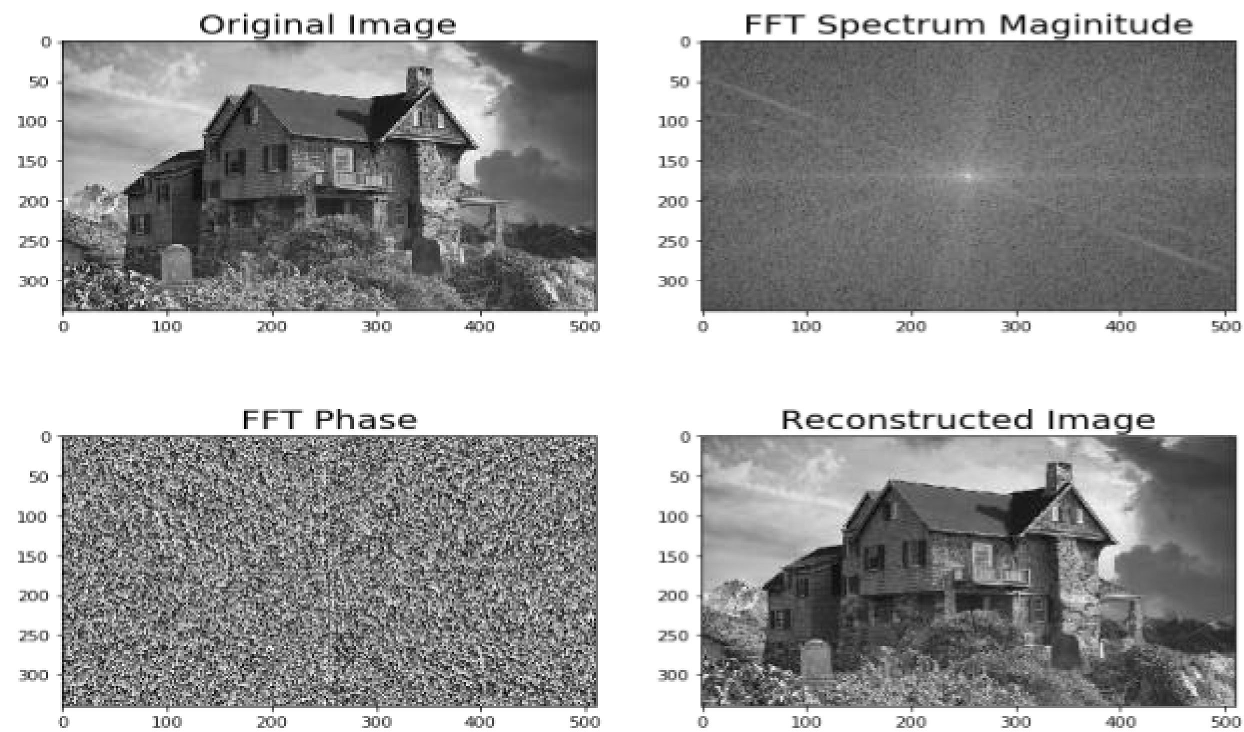
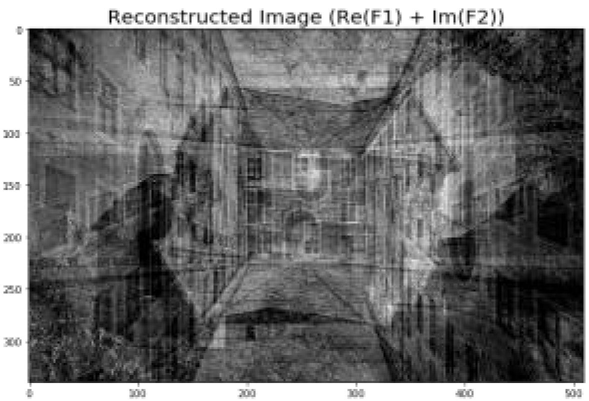
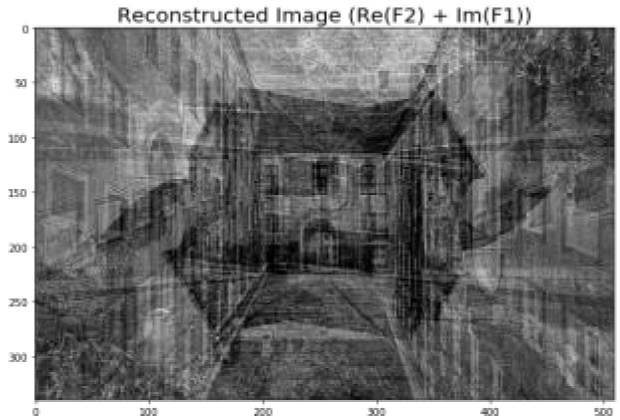


Image 2:




```
pylab.figure(figsize=(20,15))
im1_ = fp.ifft2(np.vectorize(complex)(freq1.real, freq2.imag)).real
im2_ = fp.ifft2(np.vectorize(complex)(freq2.real, freq1.imag)).real
pylab.subplot(211), pylab.imshow(np.clip(im1_,0,255), cmap='gray')
pylab.title('Reconstructed Image (Re(F1) + Im(F2))', size=20)
pylab.subplot(212), pylab.imshow(np.clip(im2_,0,255), cmap='gray')
pylab.title('Reconstructed Image (Re(F2) + Im(F1))', size=20)
pylab.show()
```


Reconstructed Image (Re(F1) + Im(F2))
0
50
100
150
200
250
300
0
100
200
300
400
500


Reconstructed Image (Re(F2) + Im(F1))
0
50
100
150
200
250
300
0
100
200
300
400
500

Practical-2:

Write a program to demonstrate the following aspects of signal on sound/image data.

1. Convolution operation
2. Template Matching

- **Applying convolution to a grayscale image**

```
im = rgb2gray(imread('../images/cameraman.jpg')).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('Original Image', size=20)
axes[1].imshow(im_blurred, cmap=pylab.cm.gray)
axes[1].set_title('Box Blur', size=20)
axes[2].imshow(im_edges, cmap=pylab.cm.gray)
axes[2].set_title('Laplace Edge Detection', size=20)
for ax in axes:
    ax.axis('off')
pylab.show()
```

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

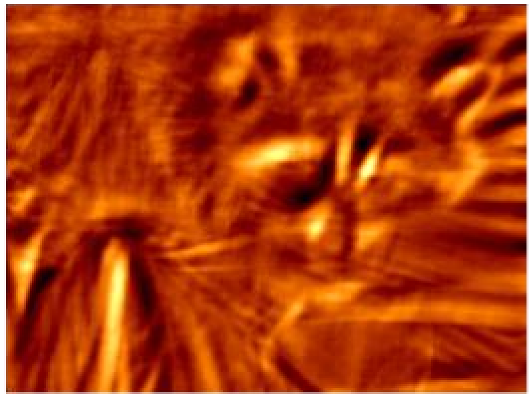
```
face_image = misc.face(gray=True) - misc.face(gray=True).mean()
template_image = np.copy(face_image[300:365, 670:750]) # right eye
template_image -= template_image.mean()
face_image = face_image + np.random.randn(*face_image.shape) * 50 # add random noise
```

```
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()
```

Original

Template

Cross-correlation



Practical-03:

Write program to implement point/pixel intensity transformations such as

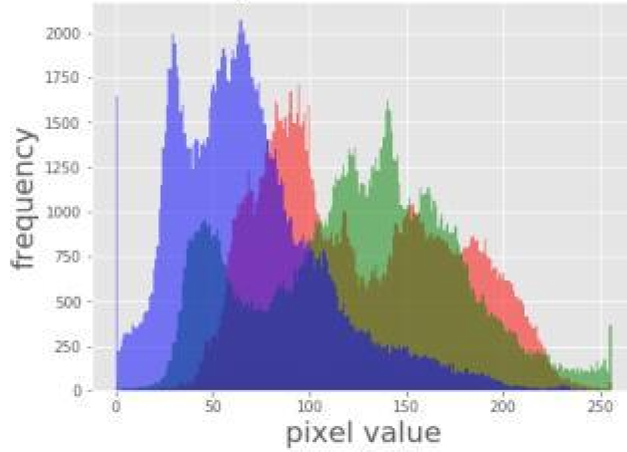
1. Log and Power-law transformations
2. Contrast adjustments
3. Histogram equalization
4. Thresholding, and halftoning operations

- **Log transform**

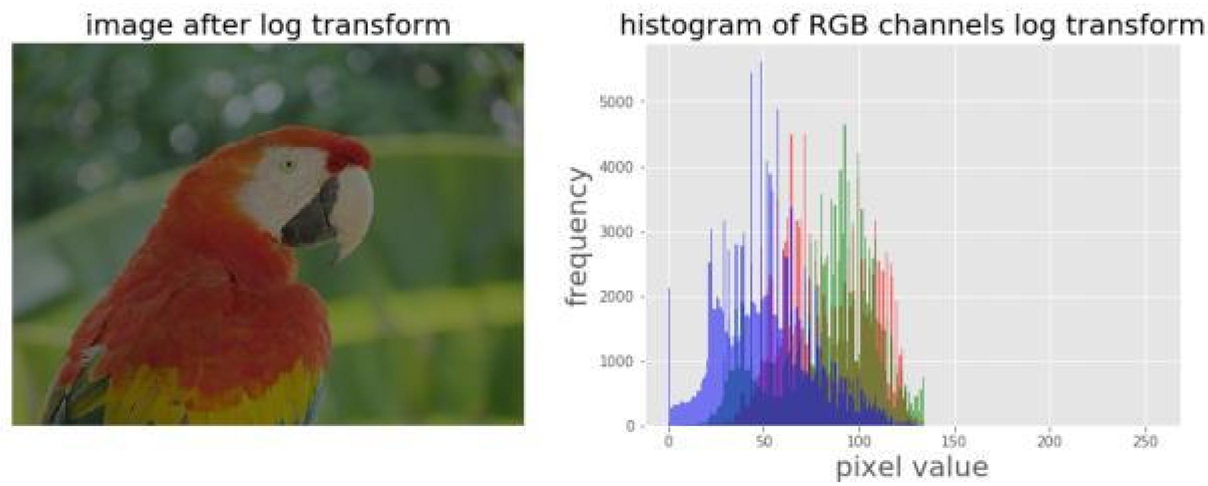
```
def plot_image(image, title=''):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis('off') # comment this line if you want axis ticks

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)

im = Image.open("../images/parrot.png")
im_r, im_g, im_b = im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121), plot_image(im,'original image')
pylab.subplot(122), plot_hist(im_r, im_g, im_b,'histogram for RGB channels')
pylab.show()
```

original image

histogram for RGB channels

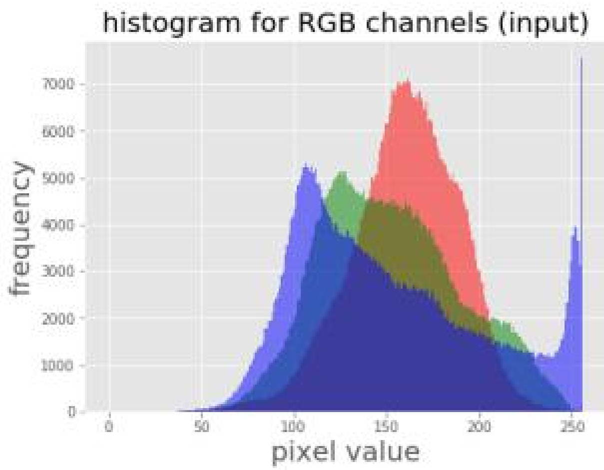


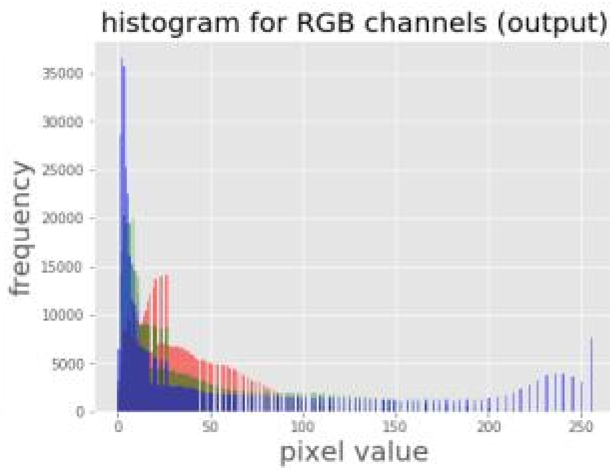
```
im = im.point(lambda i: 255*np.log(1+i/255))
im_r, im_g, im_b = im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121), plot_image(im,'image after log transform')
pylab.subplot(122), plot_hist(im_r, im_g, im_b,'histogram of RGB channels log transform')
pylab.show()
```



- **Power-law transform**

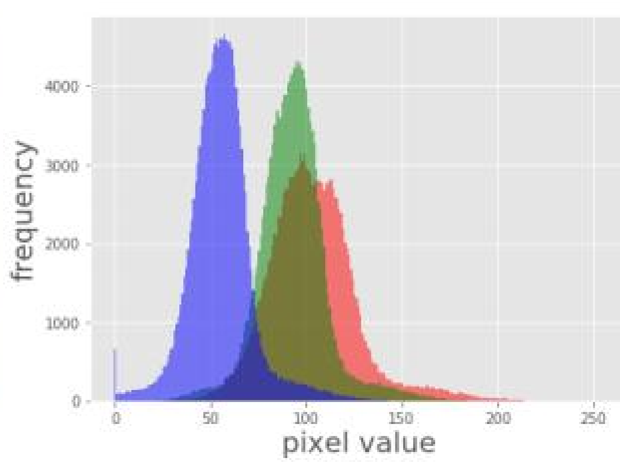
```
im = img_as_float(imread('../images/earthfromsky.jpg'))

gamma = 5
im1 = im**gamma
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121), plot_hist(im[...,0], im[...,1], im[...,2],'histogram for RGB channels (input)')
pylab.subplot(122), plot_hist(im1[...,0], im1[...,1], im1[...,2],'histogram for RGB channels (output)')
pylab.show()
```


histogram for RGB channels (input)
frequency
pixel value
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250


histogram for RGB channels (output)
frequency
pixel value
0
5000
10000
15000
20000
25000
30000
35000
0
50
100
150
200
250

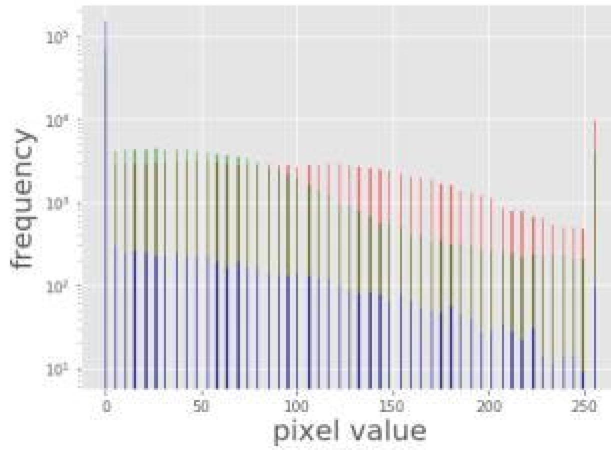
- **Contrast stretching with PIL as a point operation**

```
im = Image.open('../images/cheetah.png')
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()
```



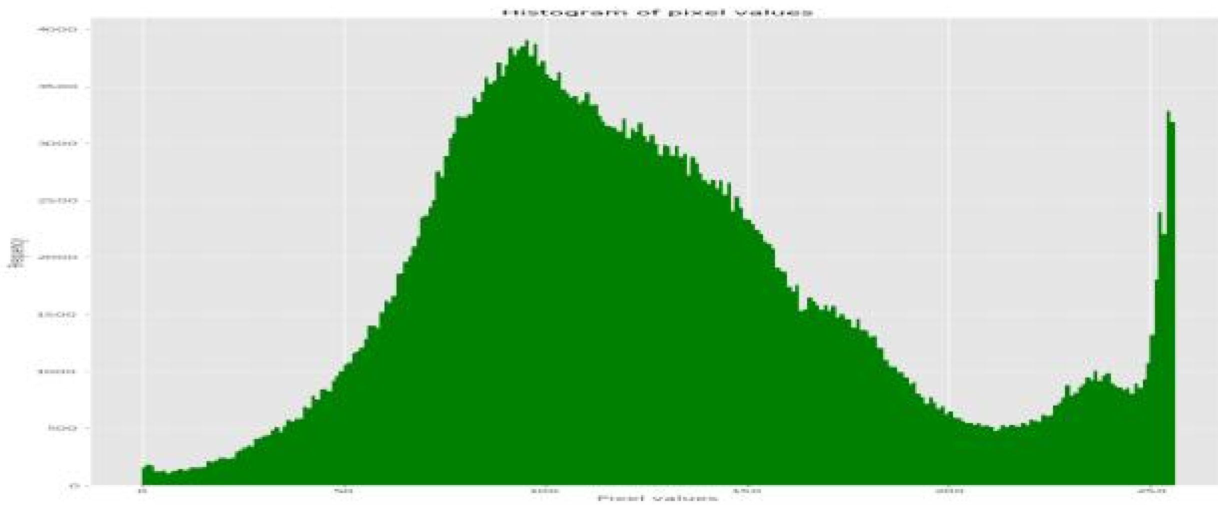
```python
def contrast(c):
    return 0 if c < 70 else (255 if c > 150 else (255*c - 22950) / 48) # piece-wise linear function

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)
```

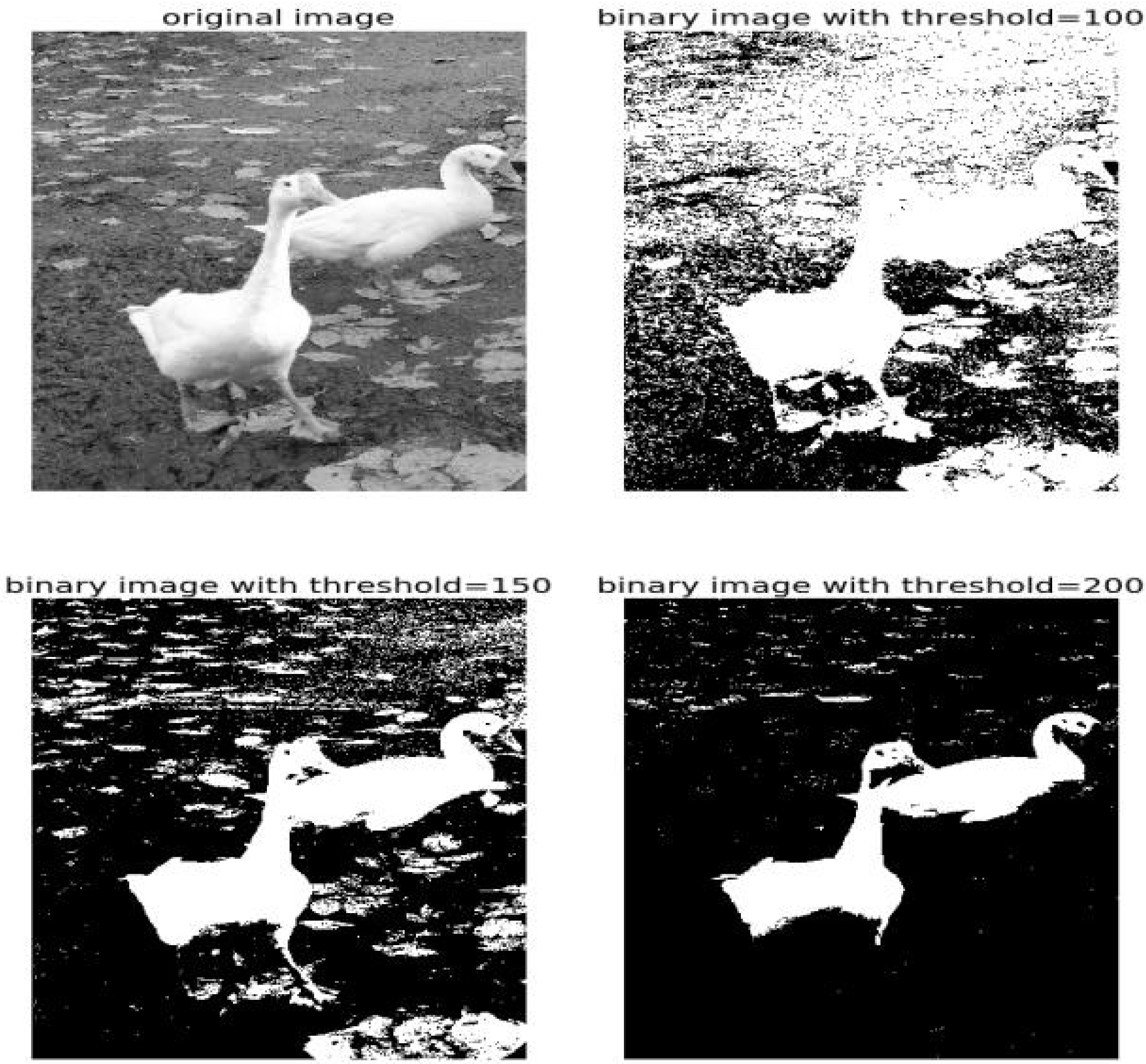
```
pylab.subplot(122)
plot_hist(im_r, im_g, im_b)
pylab.yscale('log',basey= 10)
pylab.show()
```



- **Thresholding With a fixed threshold**

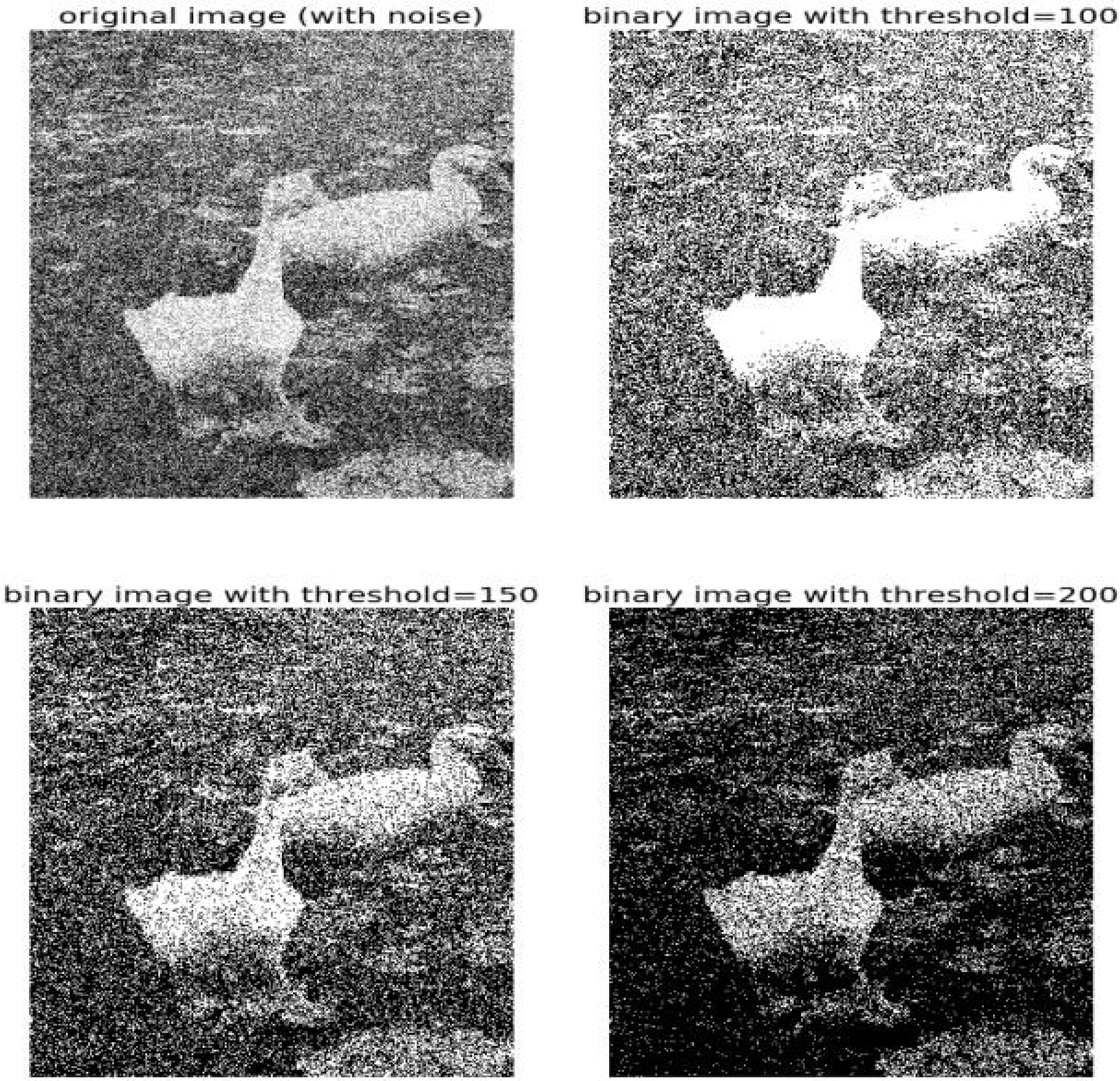
```
im = Image.open('../images/swans.jpg').convert('L')
pylab.hist(np.array(im).ravel(), bins=256, range=(0, 256), color='g')
pylab.xlabel('Pixel values'), pylab.ylabel('Frequency'),
pylab.title('Histogram of pixel values')
pylab.show()
pylab.figure(figsize=(12,18))
pylab.gray()
pylab.subplot(221), plot_image(im,'original image'), 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,'binary image with threshold=' + str(th[i]))
pylab.show()
```


Histogram of pixel values
Pixel values


original image
binary image with threshold=100
binary image with threshold=150
binary image with threshold=200

- **Half-toning**

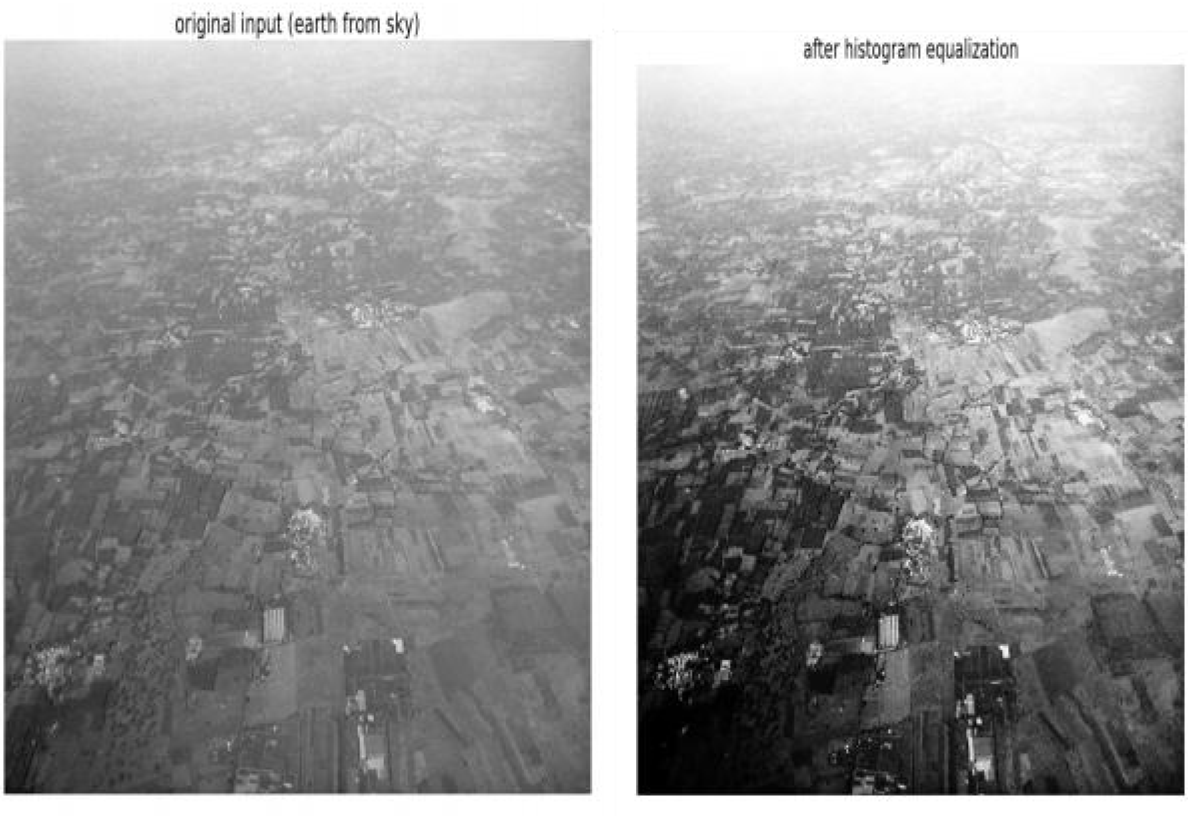
```
im = Image.open('../images/swans.jpg').convert('L')
im = Image.fromarray(np.clip(im + np.random.randint(- 128, 128, (im.height, im.width)), 0,
255).astype(np.uint8))
pylab.figure(figsize=(12,18))
pylab.subplot(221), plot_image(im,'original image (with noise)')
th = [0, 50, 100, 150, 200]

for i in range(2, 5):
    im1 = im.point(lambda x: x > th[i])
    pylab.subplot(2,2,i), plot_image(im1,'binary image with threshold=' +   str(th[i]))
pylab.show()
```


original image (with noise)
binary image with threshold=100
binary image with threshold=150
binary image with threshold=200

- **Histogram equalization with scikit-image**

```
img = rgb2gray(imread('../images/earthfromsky.jpg'))
# histogram equalization
img_eq = exposure.equalize_hist(img)
# adaptive histogram equalization
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)

pylab.gray()
images = [img, img_eq, img_adapteq]
titles = ['original input (earth from sky)','after histogram equalization','after adaptive histogram equalization']
for i in range(3):
    pylab.figure(figsize=(20,10)), plot_image(images[i], titles[i])
```

```
pylab.figure(figsize=(15,5))

for i in range(3):
    pylab.subplot(1,3,i+1), pylab.hist(images[i].ravel(), color='g'), pylab.title(titles[i], size=15)
pylab.show()
```



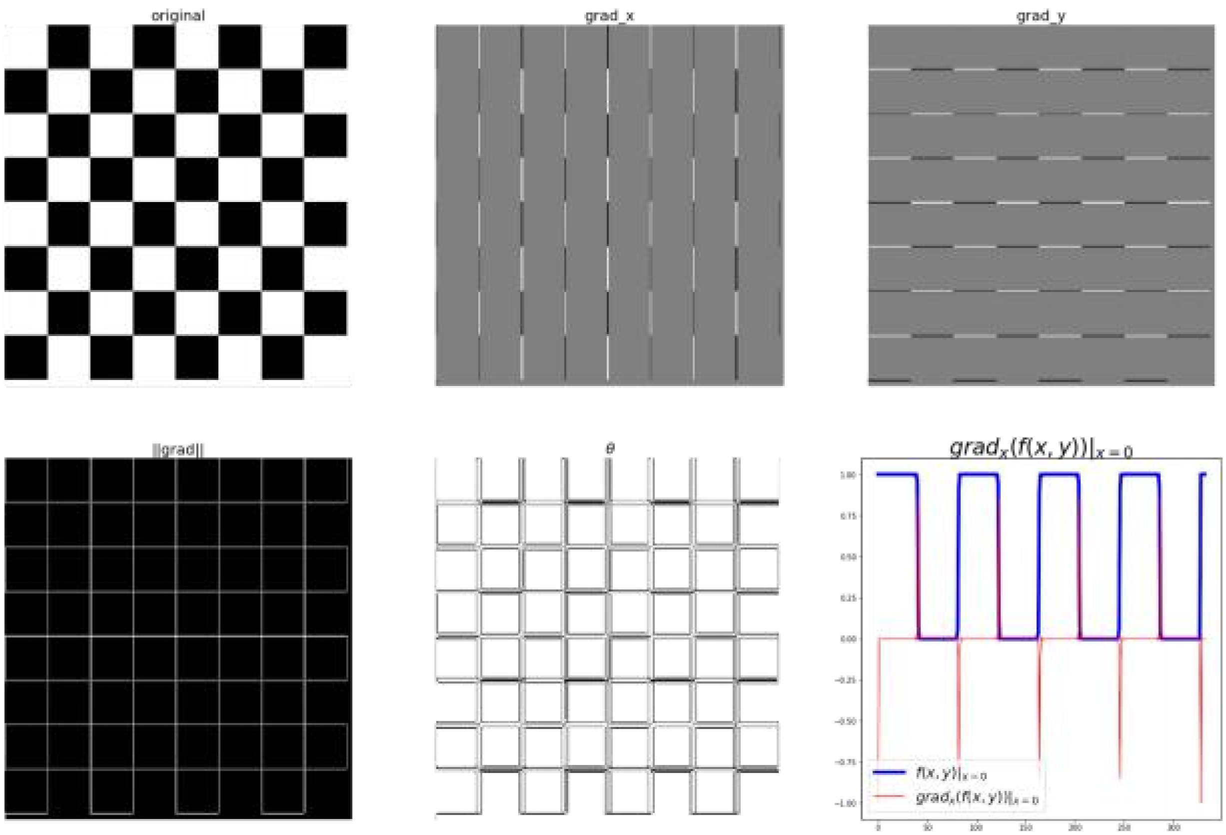
Practical-04:

Write a program to apply various enhancements on images using image derivatives by implementing Gradient and Laplacian Operations.

- **Derivatives and gradients**

```
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('../images/chess.png'))
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()
```



- **Displaying the magnitude and the gradient on the same image**

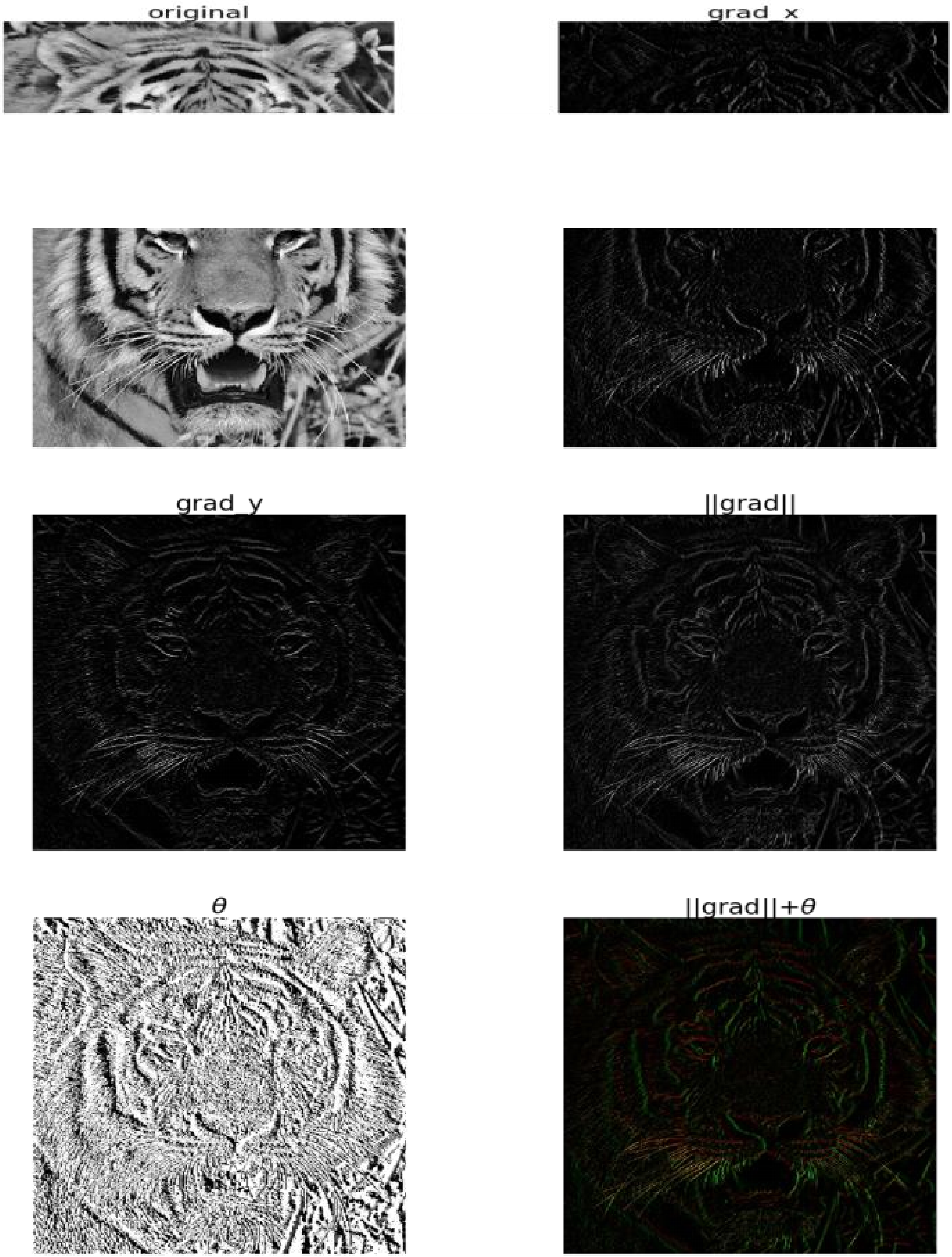
from skimage.io **import** imread

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('../images/tiger3.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()
```

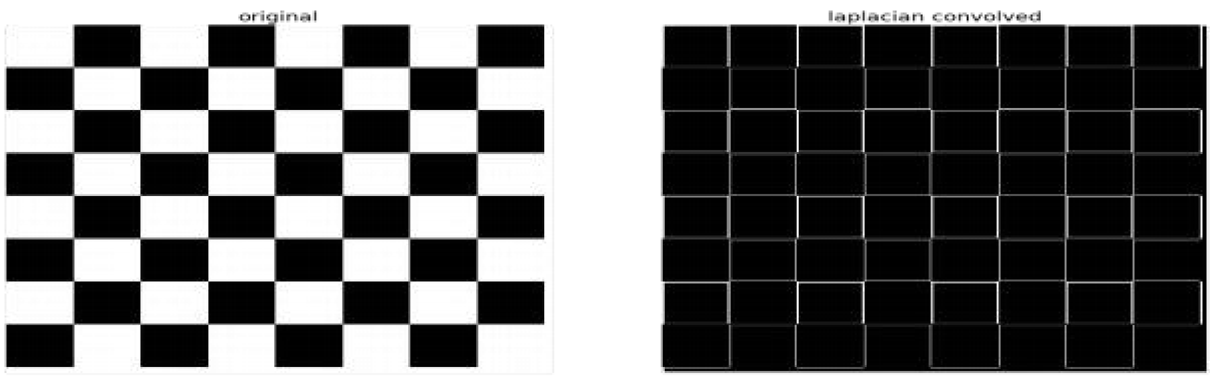
```
plt.figure(figsize=(20,30))
plt.subplot(321)

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()
```


original
grad_x
grad_y
||grad||
θ
||grad||+θ

- **Laplacian**

ker_laplacian **=** [[0,**-**1,0],[**-**1, 4, **-**1],[0,**-**1,0]]

```
im = rgb2gray(imread('../images/chess.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,'laplacian convolved')

pylab.show()
```



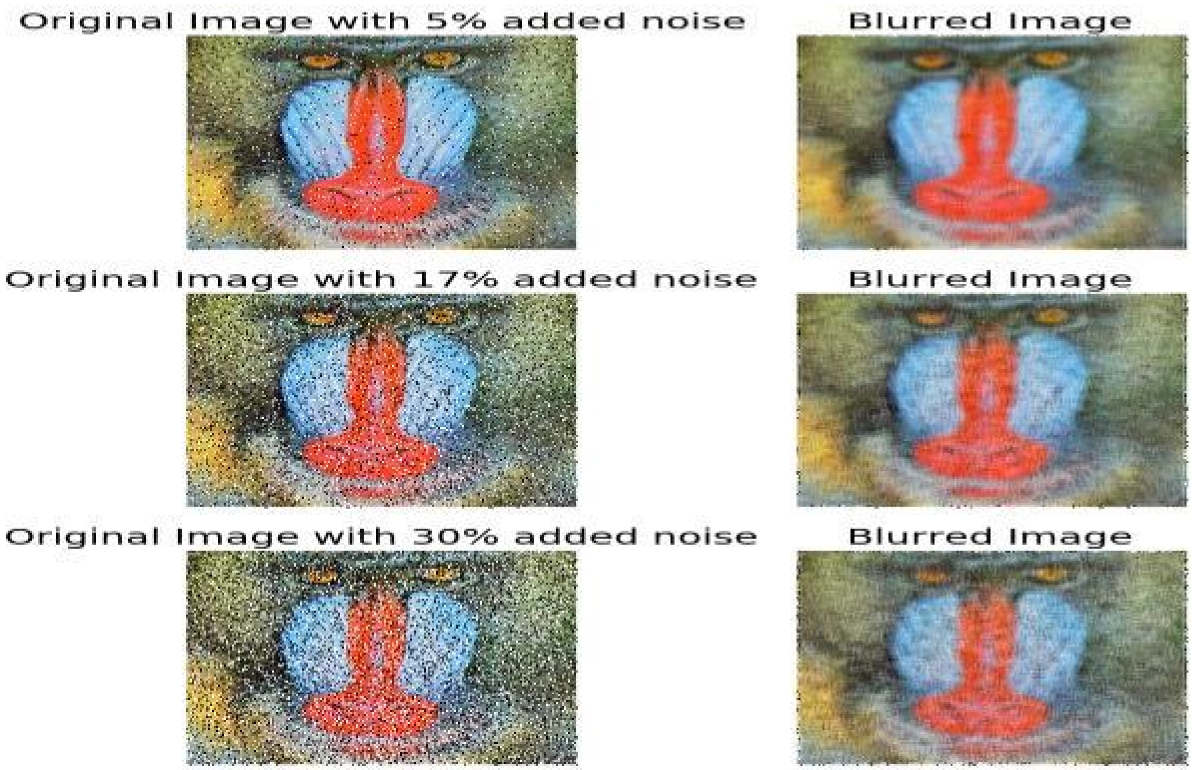
Practical-05:

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

- **Linear Smoothing:**

Smoothing with ImageFilter.BLUR

```
i = 1
pylab.figure(figsize=(10,25))
for prop_noise in np.linspace(0.05,0.3,3):

    im = Image.open('../images/mandrill.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('../images/mandrill_spnoise_' + 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()
```



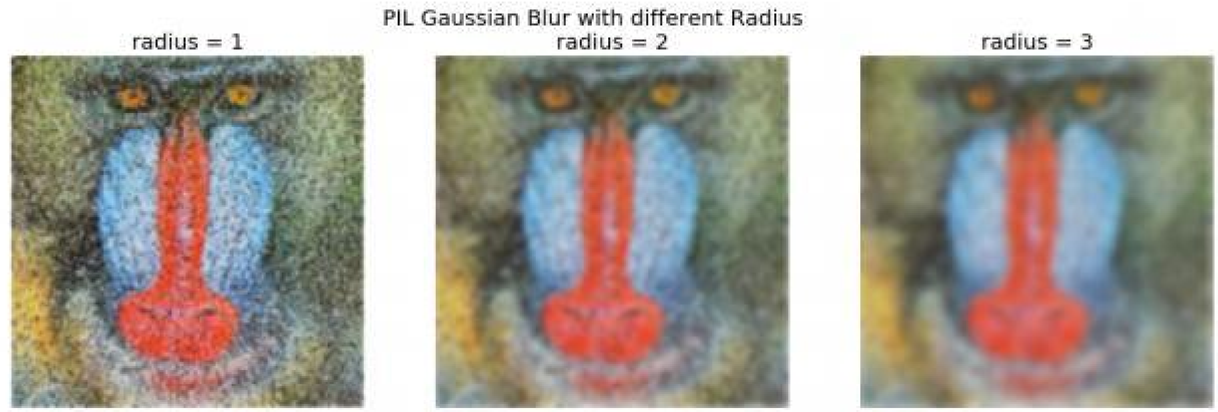
Smoothing by averaging with the box blur kernel

```
im = Image.open('../images/mandrill_spnoise_0.1.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()
```

Smoothing with the Gaussian blur filter

```
im = Image.open('../images/mandrill_spnoise_0.2.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()
```



Comparing smoothing with box and Gaussian kernels using SciPy ndimage

```
from scipy import misc, ndimage
import matplotlib.pylab as pylab
im = misc.imread('../images/mandrill_spnoise_0.1.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()
```

- **Non-Linear Smoothing:**

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('../images/mandrill.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('../images/mandrill_spnoise_' + str(prop_noise) + '.jpg')
   pylab.subplot(6,4,i)
   plot_image(im,'Original Image with ' + str(int(100*prop_noise)) + '%added noise') 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()
```

Using max and min filter

```
im = Image.open('../images/mandrill_spnoise_0.1.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))
```

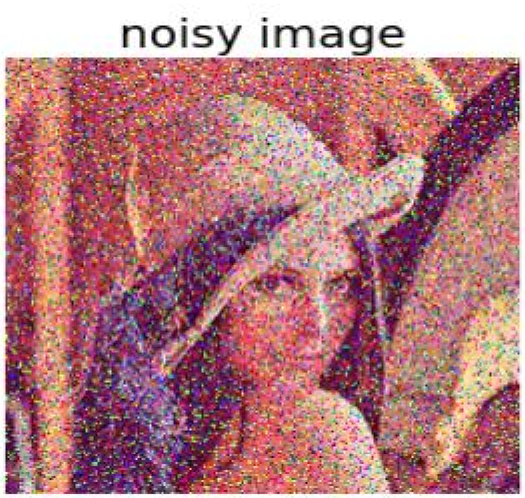
```
pylab.subplot(1,3,3), plot_image(im1,'Output (Min Filter size=' + str(sz)+ ')')

pylab.show()
```

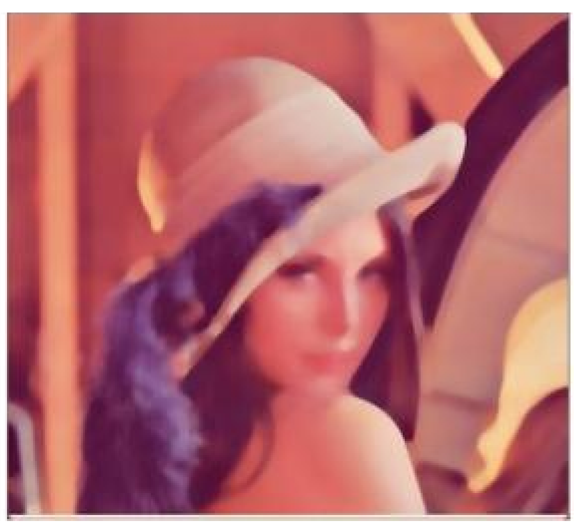
Practical-06:

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.

- **Smoothing with scipy ndimage**

```
lena = misc.imread('../images/lena.jpg')
# add salt-and-pepper noise to the input image
noise = np.random.random(lena.shape)
lena[noise > 0.9] = 255
lena[noise < 0.1] = 0
plot_image(lena,'noisy image')
pylab.show()
fig = pylab.figure(figsize=(20,15))

i = 1
for p in range(25, 100, 25):
    for k in range(5, 25, 5):
        pylab.subplot(3,4,i)
        filtered = ndimage.percentile_filter(lena, percentile=p, size=(k,k,1))
        plot_image(filtered, str(p) + ' percentile, ' + str(k) + 'x' + str(k) + ' kernel')
        i += 1
pylab.show()
```





- **Sharpening with Laplacian**

```
from skimage.filters import laplace
im = rgb2gray(imread('../images/me8.jpg'))
im1 = np.clip(laplace(im) + im, 0, 1)
```

```
pylab.figure(figsize=(10,15))
pylab.subplot(211), plot_image(im,'original image')
pylab.subplot(212), plot_image(im1,'sharpened image')
pylab.tight_layout()

 pylab.show()
```

- **Unsharp masking**

```
def rgb2gray(im):
    '''
    the input image is an RGB image
    with pixel values for each channel in [0,1]
    '''
    return np.clip(0.2989 * im[...,0] + 0.5870 * im[...,1] + 0.1140 * im[...,2], 0, 1)

im = rgb2gray(img_as_float(misc.imread('../images/me4.jpg')))
im_blurred = ndimage.gaussian_filter(im, 5)
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))
```

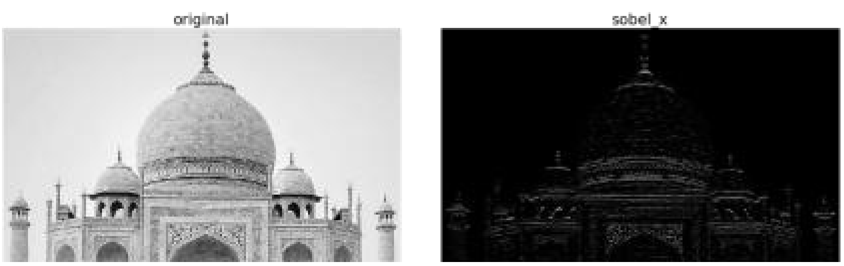
```
axes = axes.ravel()
axes[0].set_title('Original image', size= 15), axes[0].imshow(im)
axes[1].set_title('Blurred image, sigma=5', size= 15),
axes[1].imshow(im_blurred)
axes[2].set_title('Detail image', size= 15), axes[2].imshow(im_detail)
alpha = [1, 5, 10]

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()
```

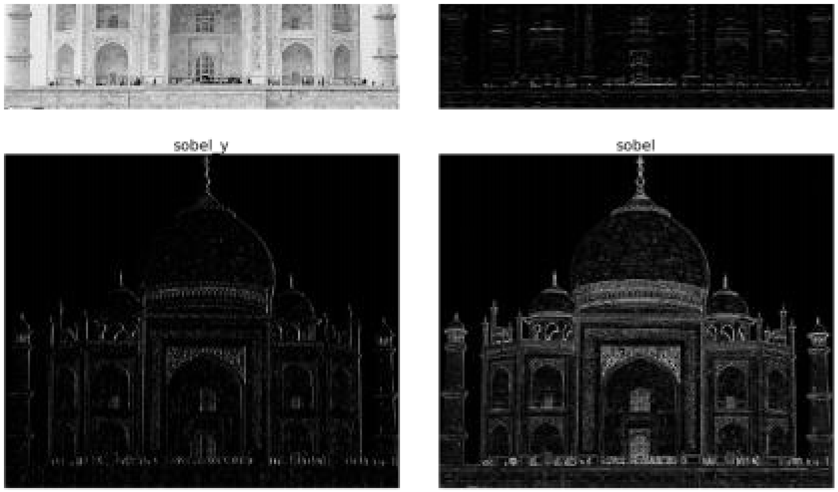
Practical-07:

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

- ## Sobel edge detector with scikit-image

```
im = rgb2gray(imread('../images/tajmahal1.jpg')) # 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()
```

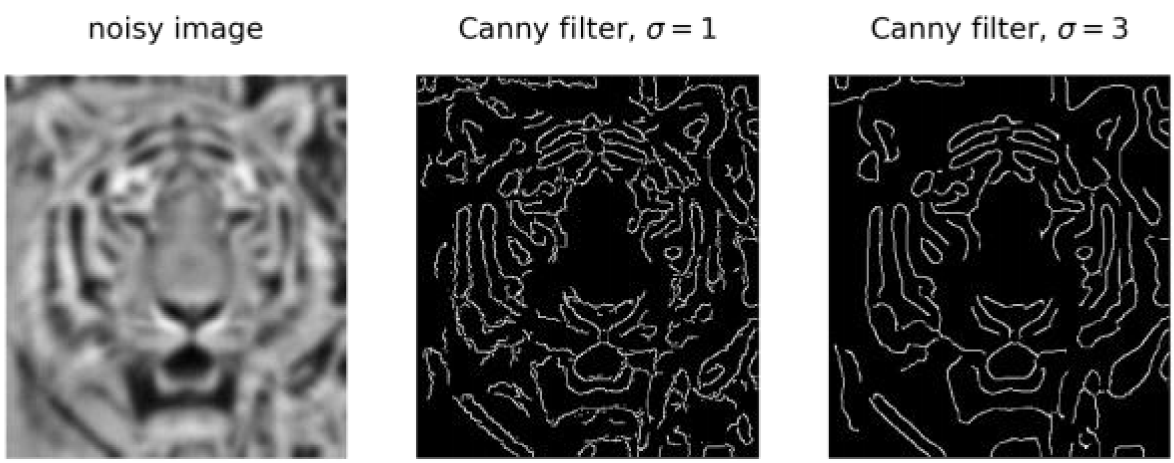




- **The Canny edge detector with scikit-image**

im = rgb2gray(imread('../images/tiger3.jpg'))

```
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()
```

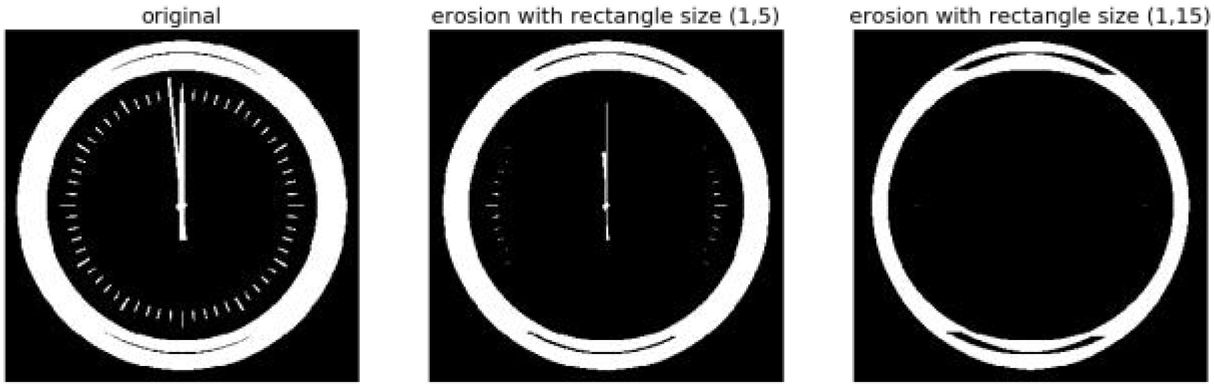


Practical-08:

Write the program to implement various morphological image processing techniques.

- **Erosion**

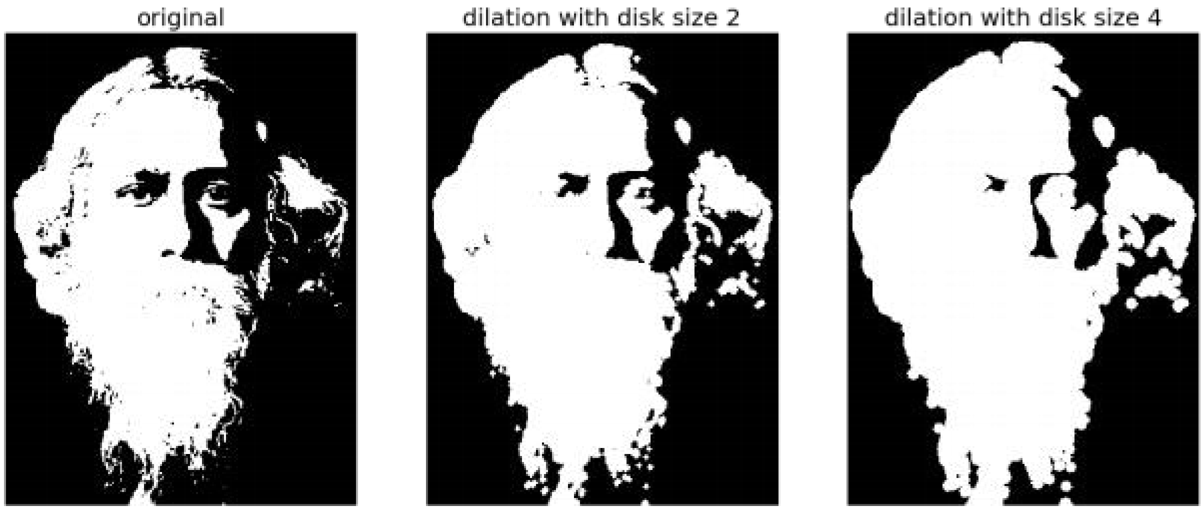
```
% matplotlib inline
from skimage.io import imread
from skimage.color import rgb2gray

import matplotlib.pylab as pylab
from skimage.morphology import binary_erosion, rectangle


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('../images/clock2.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()
```



- **Dilation**

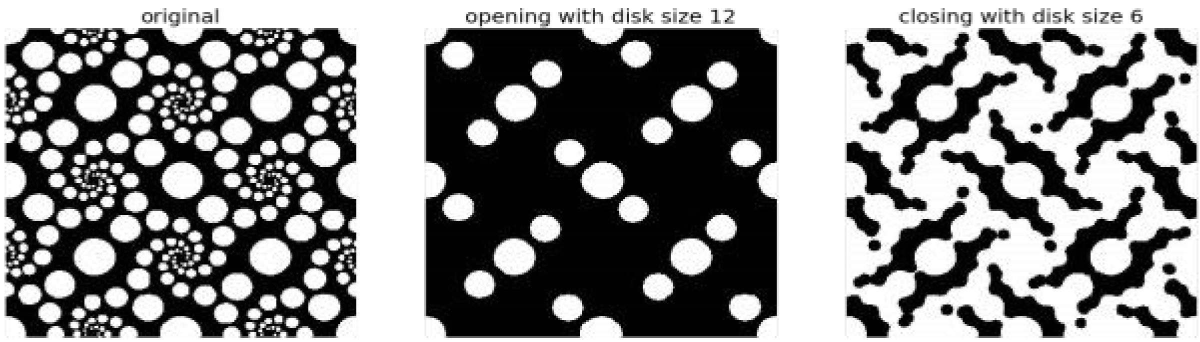
```
from skimage.morphology import binary_dilation, disk

from skimage import img_as_float
im = img_as_float(imread('../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('../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()
```

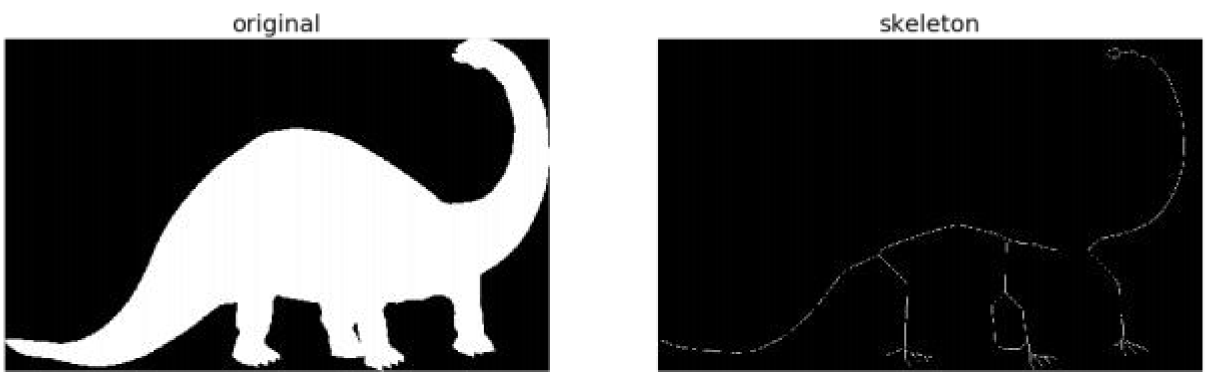


- **Skeletonizing**

```
def plot_images_horizontally(original, filtered, filter_name, sz=(18,7)):

    pylab.gray()
```

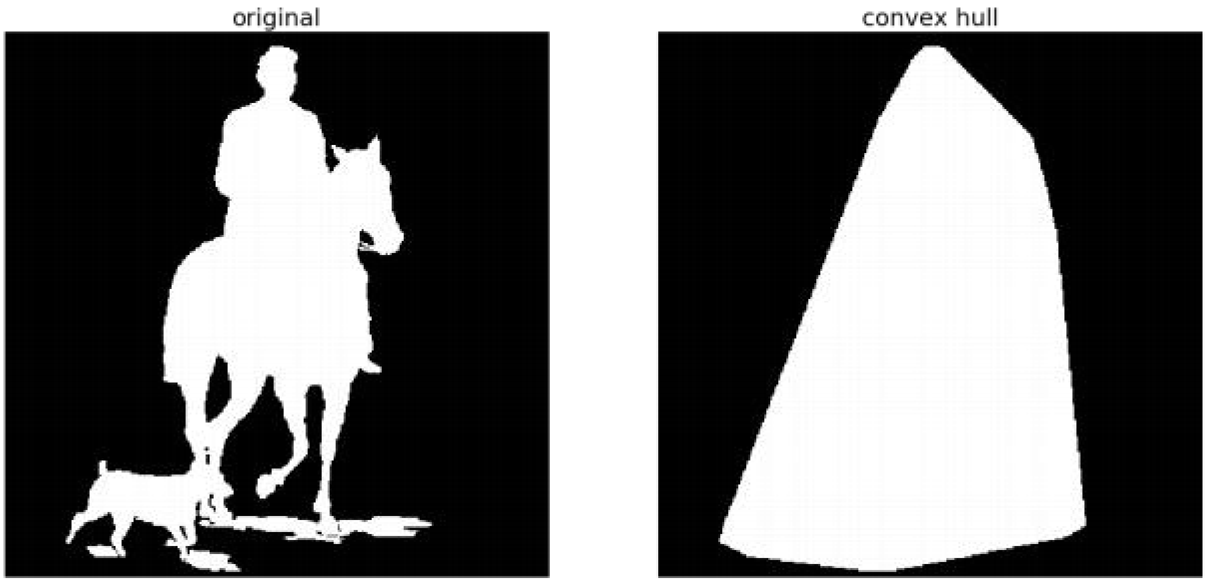
```
pylab.figure(figsize = sz)
pylab.subplot(1,2,1), plot_image(original,'original')
pylab.subplot(1,2,2), plot_image(filtered, filter_name)
pylab.show()
```

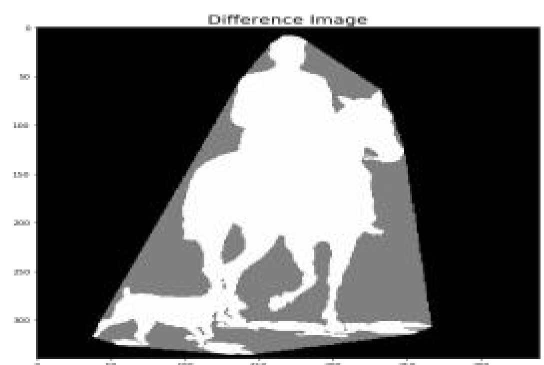
```
from skimage.morphology import skeletonize
im = img_as_float(imread('../images/dynasaur.png')[...,3])
threshold = 0.5
im[im <= threshold] = 0
im[im > threshold] = 1
skeleton = skeletonize(im)
plot_images_horizontally(im, skeleton,'skeleton',sz=(18,9))
```



- **Computing the convex hull**

```
from skimage.morphology import convex_hull_image
im = rgb2gray(imread('../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))
```



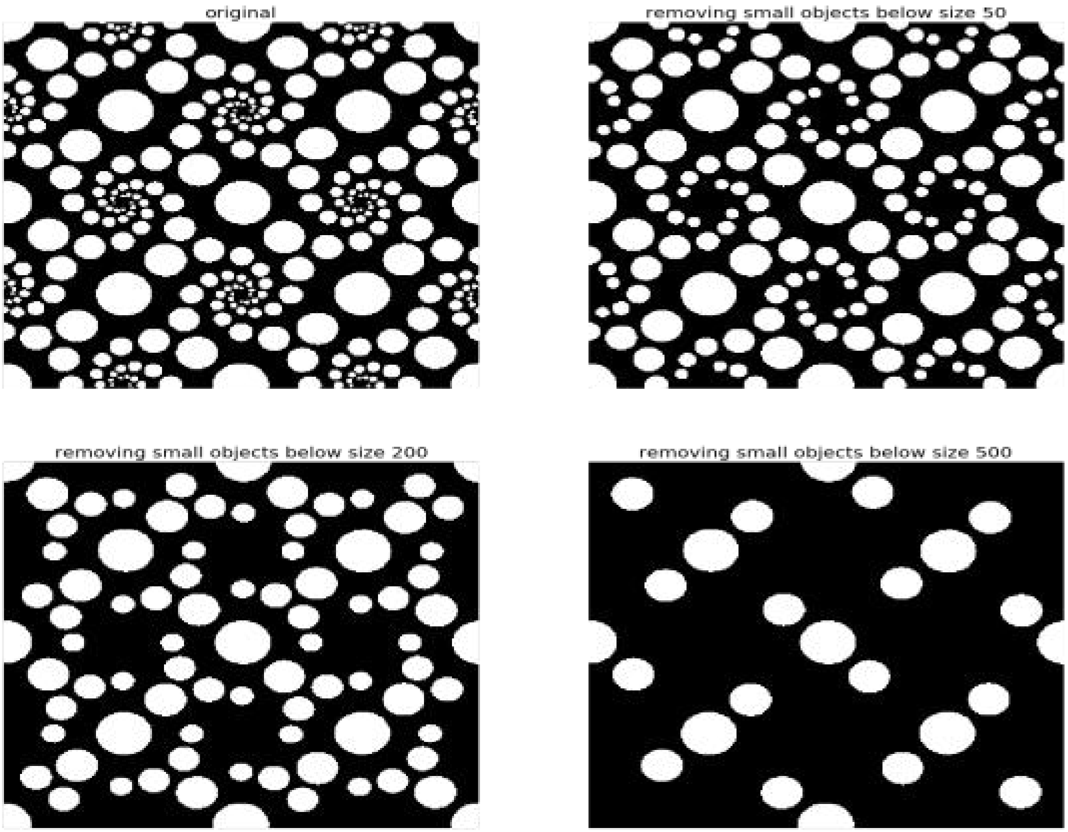



```
im = im.astype(np.bool)
chull_diff = img_as_float(chull.copy())
chull_diff[im] = 2
pylab.figure(figsize=(20, 10))
pylab.imshow(chull_diff, cmap=pylab.cm.gray,
interpolation='nearest')
pylab.title('Difference Image', size=20)
pylab.show()
```

- **Removing small objects**

```
from skimage.morphology import remove_small_objects
im = rgb2gray(imread('../images/circles.jpg'))
im[im > 0.5] = 1 # create binary image by thresholding with fixed threshold

0.5
im[im <= 0.5] = 0
im = im.astype(np.bool)
```

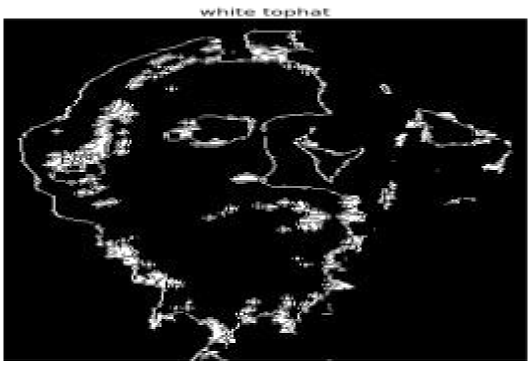
```
pylab.figure(figsize=(20,20))
pylab.subplot(2,2, 1), plot_image(im, 'original')

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('../images/tagore.png')[...,3]
im[im <= 0.5] = 0
im[im > 0.5] = 1
im1 = white_tophat(im, square(5))
```

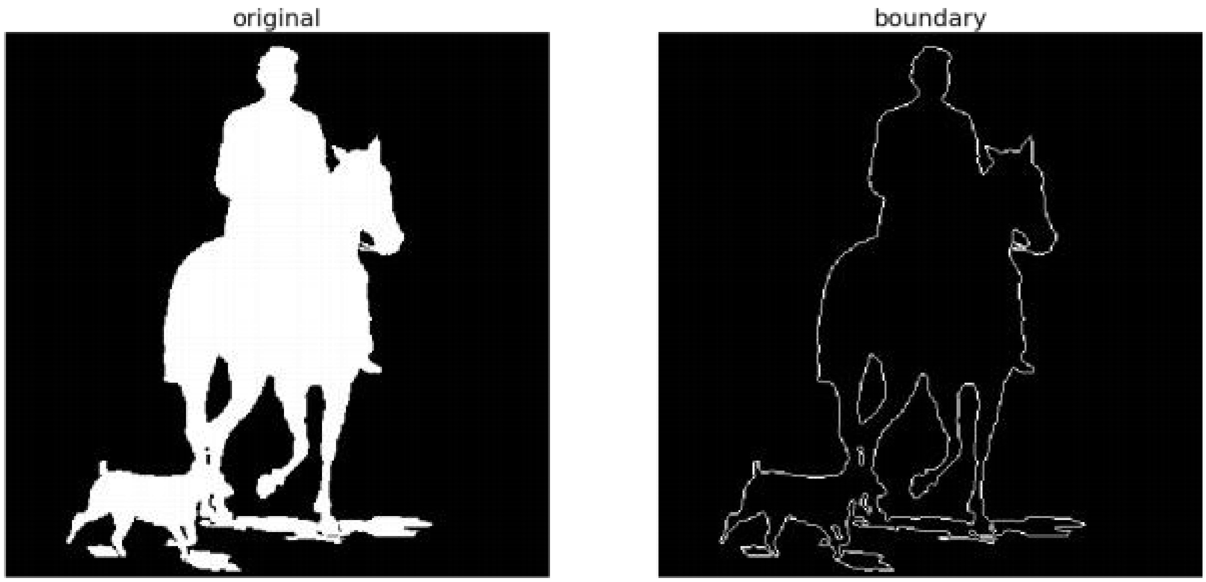
```
im2 = black_tophat(im, square(5))
pylab.figure(figsize=(20,15))
pylab.subplot(1,2,1), plot_image(im1,'white tophat')
pylab.subplot(1,2,2), plot_image(im2,'blacktophat')
pylab.show()
```



- **Extracting the boundary**

```
from skimage.morphology import binary_erosion
```

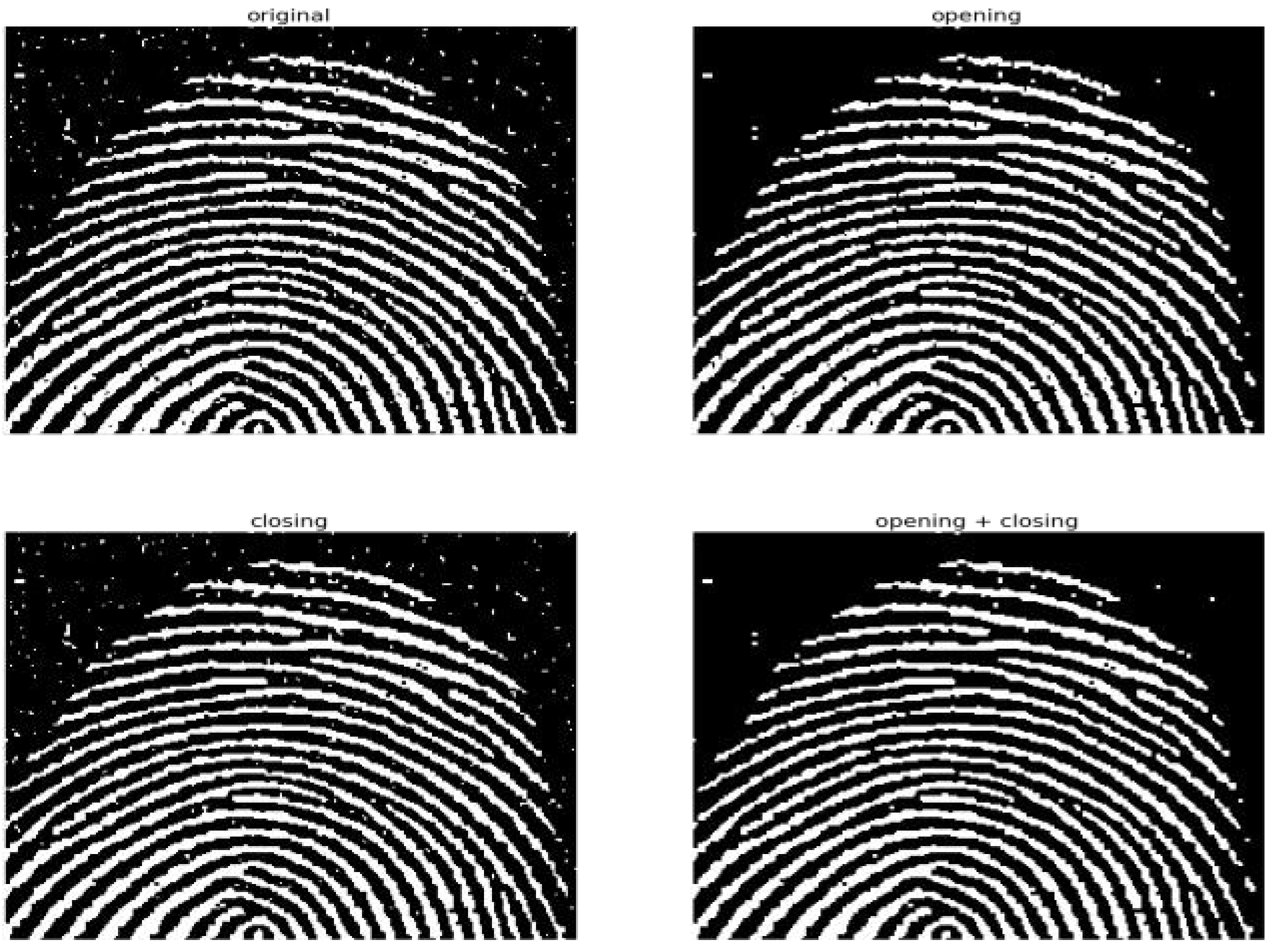
```
im = rgb2gray(imread('../images/horse-dog.jpg'))
threshold = 0.5
im[im < threshold] = 0
im[im >= threshold] = 1
boundary = im - binary_erosion(im)
plot_images_horizontally(im, boundary, 'boundary',sz=(18,9))
```



- **Fingerprint cleaning with opening and closing**

```
im = rgb2gray(imread('../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()
```


original
opening
closing
opening + closing

- **Morphological contrast enhancement**

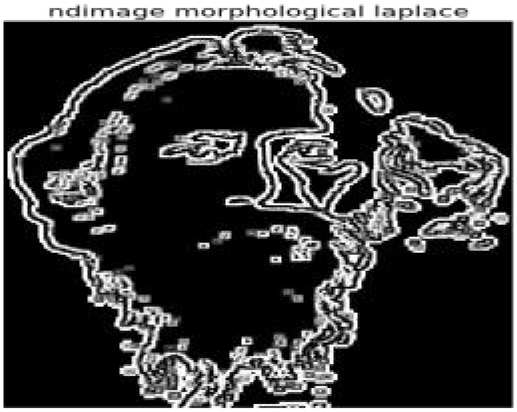
```
from skimage.filters.rank import enhance_contrast
from skimage import exposure
def plot_gray_image(ax, image, title):
    ax.imshow(image, cmap=pylab.cm.gray),
    ax.set_title(title), ax.axis('off')

    ax.set_adjustable('box-forced')

image = rgb2gray(imread('../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 contrast enhancement')

plot_gray_image(axes3, equalized_image,'Adaptive Histogram equalization')
```

- **Computing the morphological Laplace**

```
im = imread('../images/tagore.png')[...,3]
im_g = ndimage.morphological_gradient(im, size=(3,3))

im_l = ndimage.morphological_laplace(im, size=(5,5))
pylab.figure(figsize=(15,10))
pylab.subplot(121), pylab.title('ndimage morphological laplace', size=20), pylab.imshow(im_l)
pylab.axis('off')
pylab.subplot(122), pylab.title('ndimage morphological gradient', size=20),

pylab.imshow(im_g)
pylab.axis('off')

pylab.show()
```

ndimage morphological laplace



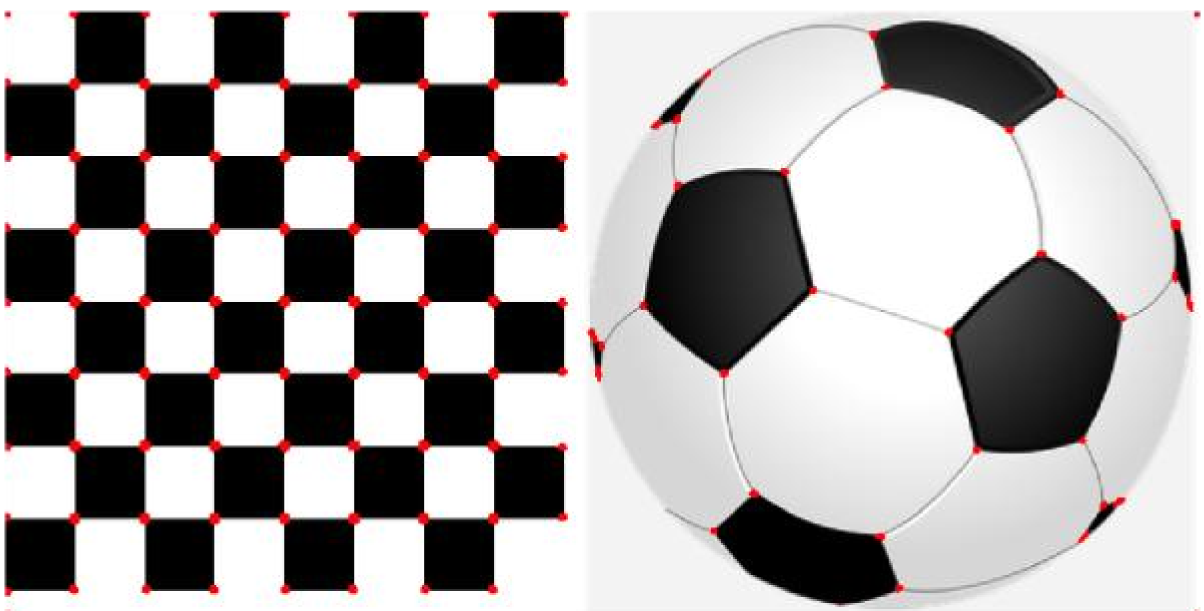
ndimage morphological gradient

Practical-09:

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

- ## Harris Corner Detector With scikit-image

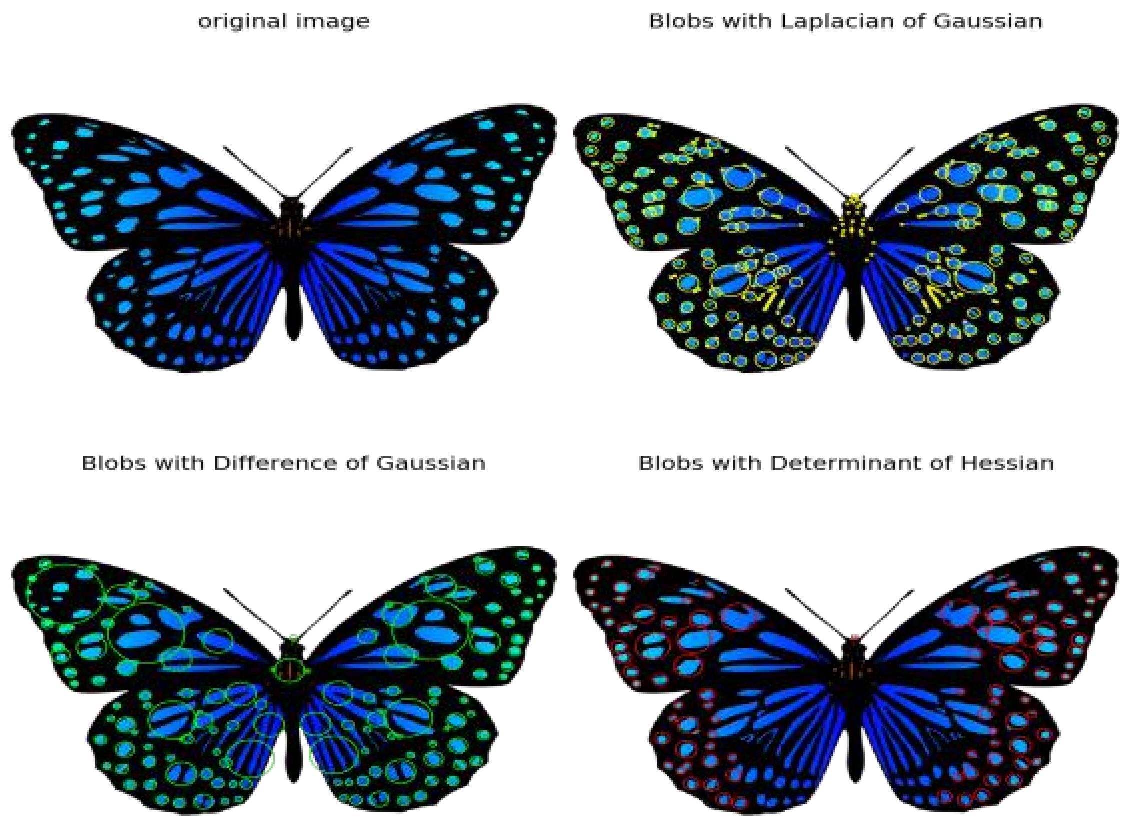
```
image = imread('../images/chess_football.png') # RGB image
image_gray = rgb2gray(image)
coordinates = corner_harris(image_gray, k =0.001)
image[coordinates>0.01*coordinates.max()]=[255,0,0,255]

pylab.figure(figsize=(20,10))
pylab.imshow(image), pylab.axis('off'), pylab.show()
```



- **Blob detectors with LoG, DoG and DoH**

```
from numpy import sqrt
from skimage.feature import blob_dog, blob_log, blob_doh

im = imread('../images/butterfly.png')
im_gray = rgb2gray(im)
log_blobs = blob_log(im_gray, max_sigma=30, num_sigma=10, threshold=.1)

log_blobs[:, 2] = sqrt(2) * log_blobs[:, 2] # Compute radius in the 3rd column
dog_blobs = blob_dog(im_gray, max_sigma=30, threshold=0.1)
dog_blobs[:, 2] = sqrt(2) * dog_blobs[:, 2]
doh_blobs = blob_doh(im_gray, max_sigma=30, threshold=0.005)

list_blobs = [log_blobs, dog_blobs, doh_blobs]
```

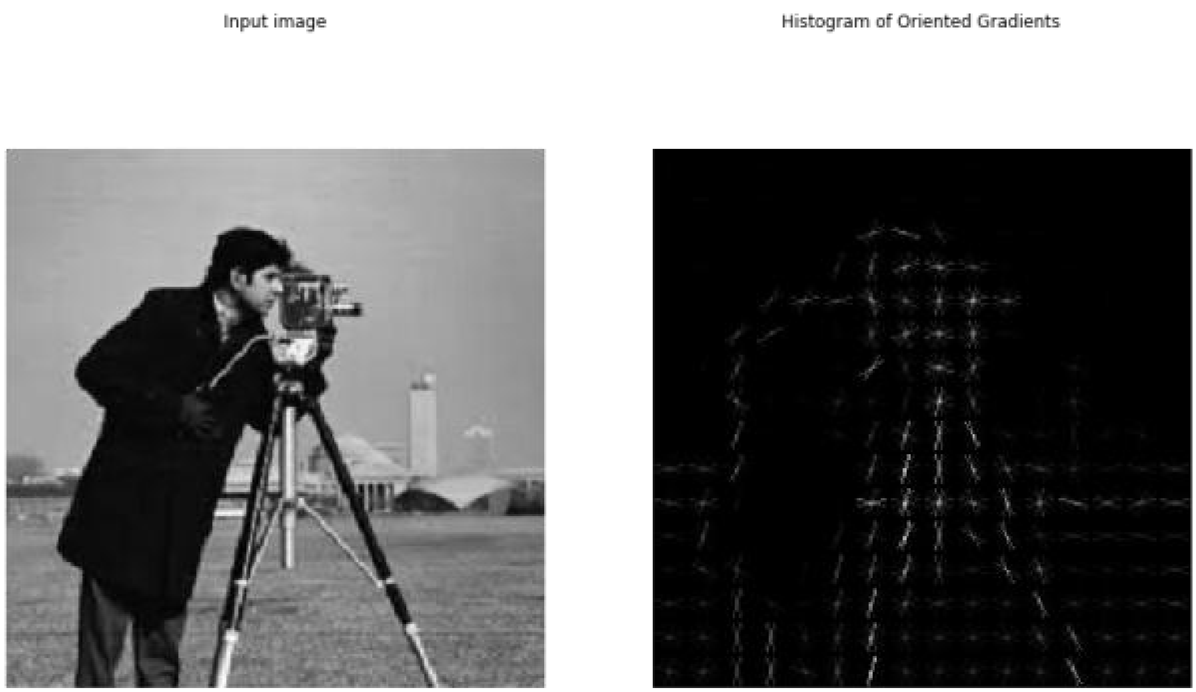
```
colors, titles = ['yellow','lime','red'], ['Laplacian of Gaussian', 'Difference of Gaussian', 'Determinant of Hessian']
sequence = zip(list_blobs, colors, titles)
fig, axes = pylab.subplots(2, 2,figsize=(20, 20),sharex=True, sharey=True)

axes = axes.ravel()
axes[0].imshow(im, interpolation='nearest')
axes[0].set_title('original image', size=30), axes[0].set_axis_off()

for idx, (blobs, color, title) in enumerate(sequence):
    axes[idx+1].imshow(im, interpolation='nearest')

    axes[idx+1].set_title('Blobs with ' + title, size=30)
    for blob in blobs:

        y, x, row = blob
        col = pylab.Circle((x, y), row, color=color, linewidth=2, fill=False)

        axes[idx+1].add_patch(col), axes[idx+1].set_axis_off()
pylab.tight_layout(), pylab.show()
```



- ## Compute HOG descriptors with scikit-image

```
from skimage.feature import hog
from skimage import exposure
image = rgb2gray(imread('../images/cameraman.jpg'))
fd,hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
```

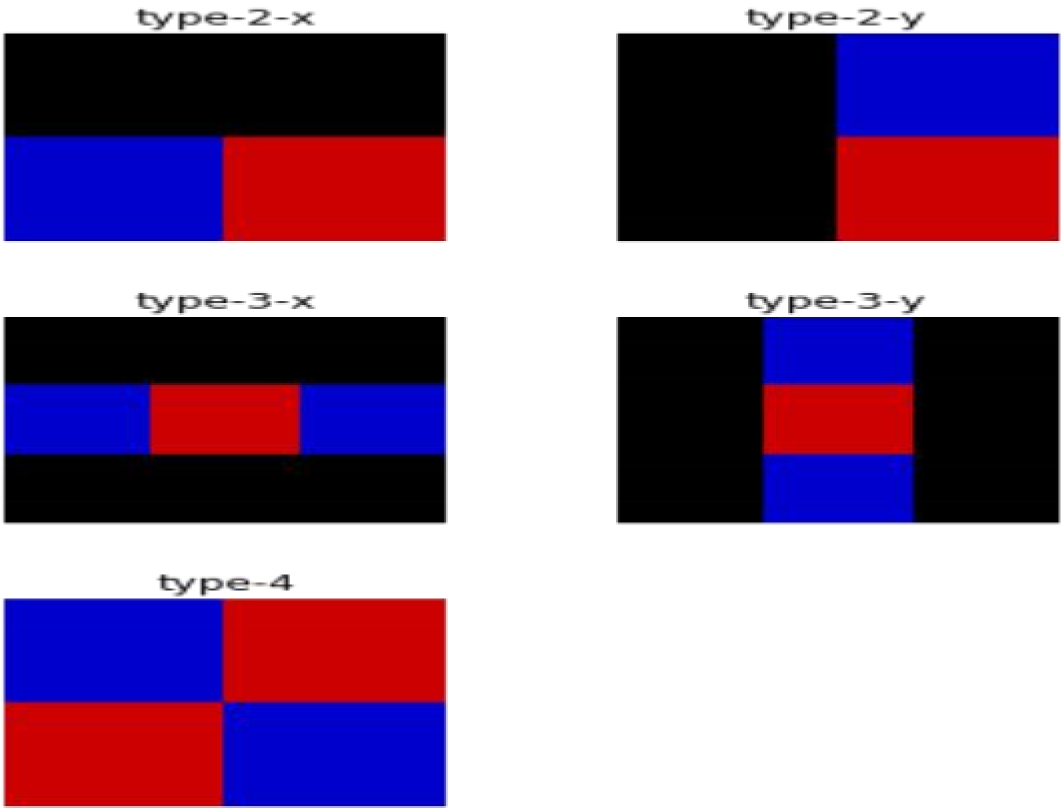
```
cells_per_block=(1, 1), visualize=True)

print(image.shape, len(fd))
# ((256L, 256L), 2048)
fig, (axes1, axes2) = pylab.subplots(1, 2,figsize=(15, 10),sharex=True, sharey=True)
axes1.axis('off'), axes1.imshow(image, cmap=pylab.cm.gray), axes1.set_title('Input image')

hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
axes2.axis('off'), axes2.imshow(hog_image_rescaled, cmap=pylab.cm.gray),
axes2.set_title('Histogram of Oriented Gradients')

pylab.show()
```



- **Haar-like feature descriptor with scikit-image**

```
from skimage.feature import haar_like_feature_coord
from skimage.feature import draw_haar_like_feature
images = [np.zeros((2, 2)), np.zeros((2, 2)), np.zeros((3, 3)),

np.zeros((3, 3)), np.zeros((2, 2))]
feature_types = ['type-2-x','type-2-y','type-3-x','type-3-y','type-4']

fig, axes = pylab.subplots(3, 2,figsize=(5,7))
for axes, img, feat_t in zip(np.ravel(axes), images, feature_types):
    coordinates, _ = haar_like_feature_coord(img.shape[0], img.shape[1], feat_t)
    haar_feature = draw_haar_like_feature(img, 0, 0, img.shape[0],img.shape[1], coordinates, max_n_features=1,
random_state=0, color_positive_block=(1.0, 0.0, 0.0), color_negative_block=(0.0, 0.0, 1.0), alpha=0.8)
    axes.imshow(haar_feature), axes.set_title(feat_t), axes.set_axis_off()
```

```
#fig.suptitle('Different Haar-like feature descriptors')
pylab.axis('off'), pylab.tight_layout(), pylab.show()
```



Practical-10:

Write the program to apply segmentation for detecting lines, circles, and other shapes/objects. Also, implement edge-based and region-based segmentation.

- ## Hough transform – detecting lines and circles

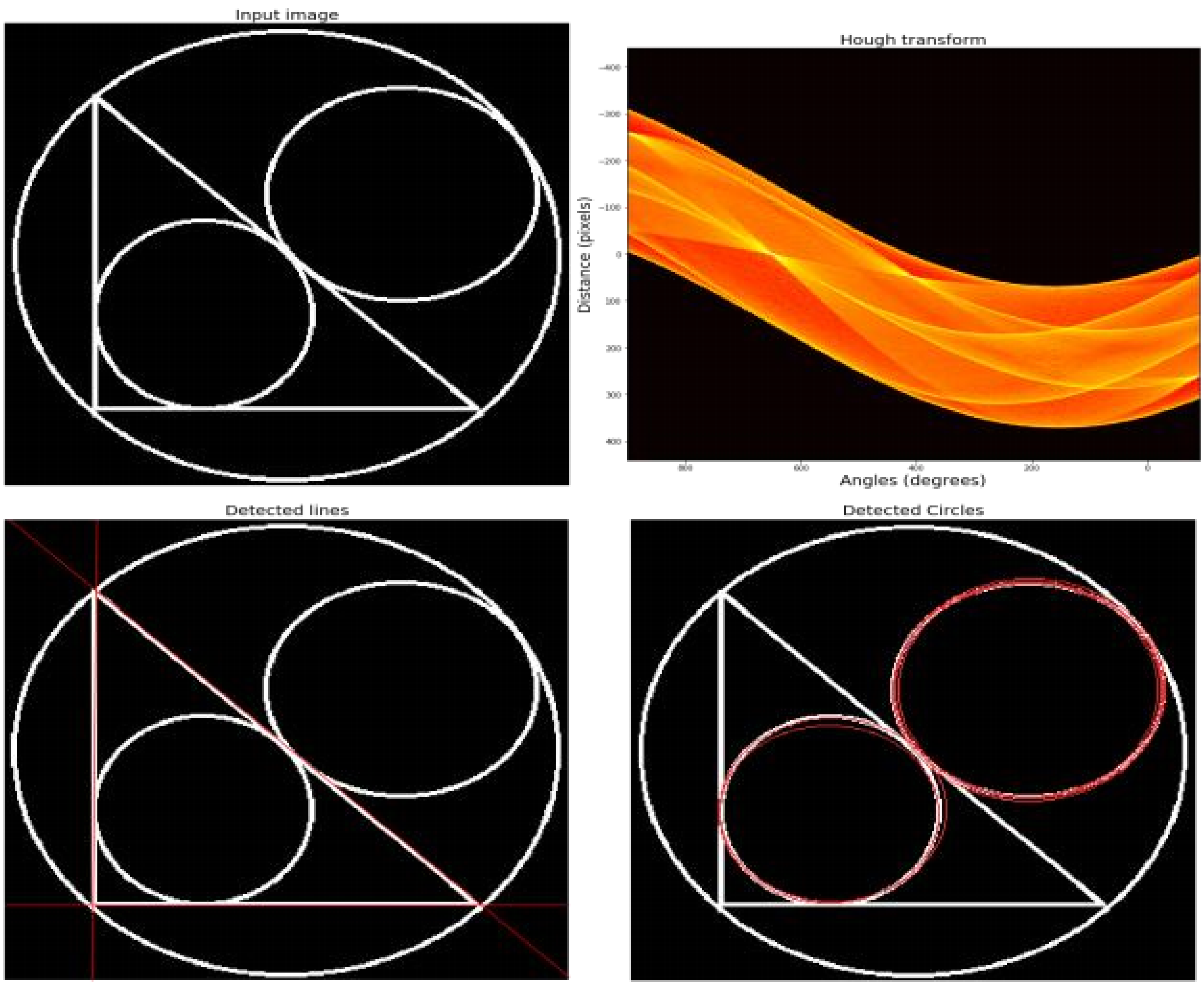
```
image = rgb2gray(imread('../images/triangle_circle.png'))

# Classic straight-line Hough transform
h, theta, d = hough_line(image)

# Generating figure 1
fig, axes = plt.subplots(2, 2,figsize=(20, 20))

ax = axes.ravel()

ax[0].imshow(image, cmap=cm.gray)

ax[0].set_title('Input image', size=20)
ax[0].set_axis_off()

ax[1].imshow(np.log(1 + h),
        extent=[10*np.rad2deg(theta[-1]), np.rad2deg(theta[0]), d[-1], d[0]],

        cmap=cm.hot, aspect=1/1.5)
ax[1].set_title('Hough transform', size=20)

ax[1].set_xlabel('Angles (degrees)', size=20)
ax[1].set_ylabel('Distance (pixels)', size=20)
ax[1].axis('image')

ax[2].imshow(image, cmap=cm.gray)
for _, angle, dist in zip(*hough_line_peaks(h, theta, d)):

  y0 = (dist - 0 * np.cos(angle)) / np.sin(angle)
  y1 = (dist - image.shape[1] * np.cos(angle)) / np.sin(angle)

  ax[2].plot((0, image.shape[1]), (y0, y1), '-r')
ax[2].set_xlim((0, image.shape[1]))
```

```
ax[2].set_ylim((image.shape[0], 0))
ax[2].set_axis_off()
ax[2].set_title('Detected lines', size=20)
```

```
hough_radii = np.arange(50, 100, 2)
hough_res = hough_circle(image, hough_radii)

# Select the most prominent 5 circles
accums, cx, cy, radii = hough_circle_peaks(hough_res, hough_radii, total_num_peaks=6)

image = gray2rgb(image)
for center_y, center_x, radius in zip(cy, cx, radii):
    circy, circx = circle_perimeter(center_y, center_x, radius)

    image[circy, circx] = (0.9, 0.2, 0.2)

ax[3].imshow(image, cmap=plt.cm.gray)

ax[3].set_axis_off()
ax[3].set_title('Detected Circles', size=20)

plt.tight_layout()

plt.show()

image = rgb2gray(imread('../images/coins.png'))

fig, axes = plt.subplots(1, 2,figsize=(20, 10),sharex=True, sharey=True)

ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)

ax[0].set_axis_off()
ax[0].set_title('Original Image', size=20)

hough_radii = np.arange(65, 75, 1)
hough_res = hough_circle(image, hough_radii)

# Select the most prominent 5 circles
accums, cx, cy, radii = hough_circle_peaks(hough_res, hough_radii, total_num_peaks=4)

image = gray2rgb(image)
for center_y, center_x, radius in zip(cy, cx, radii):
    circy, circx = circle_perimeter(center_y, center_x, radius)

    image[circy, circx] = (1, 0, 0)
```

```
ax[1].imshow(image, cmap=plt.cm.gray)
```

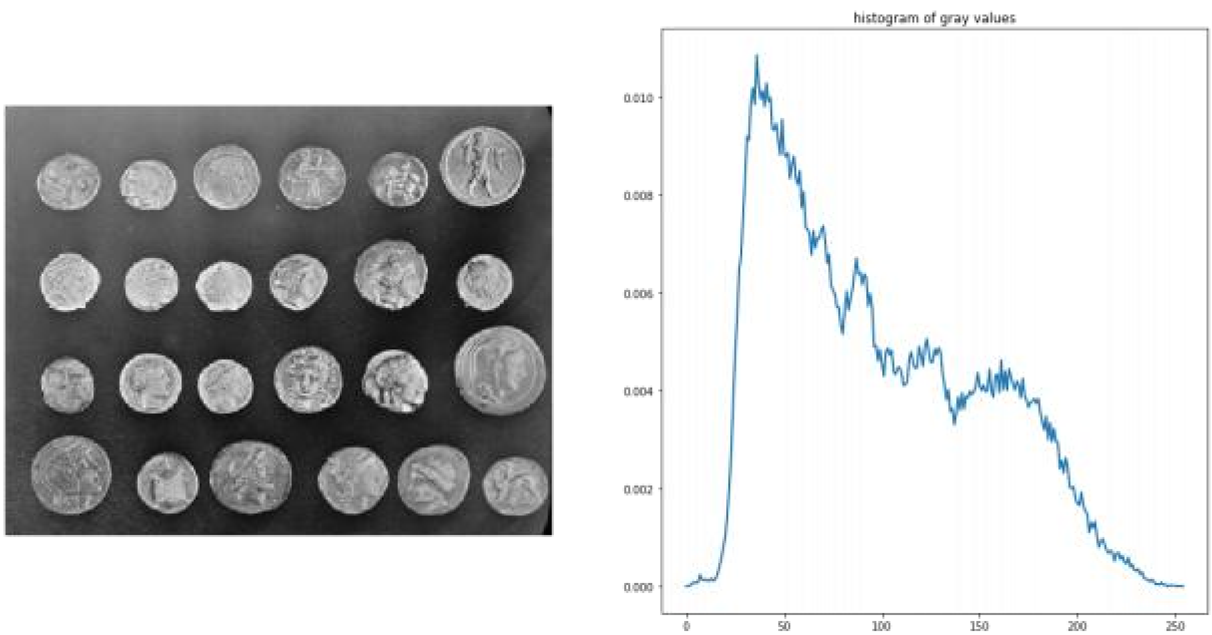
```
ax[1].set_axis_off()
ax[1].set_title('Detected Circles', size=20)

plt.tight_layout()

plt.show()
```



- **Edges-based/region-based segmentation**

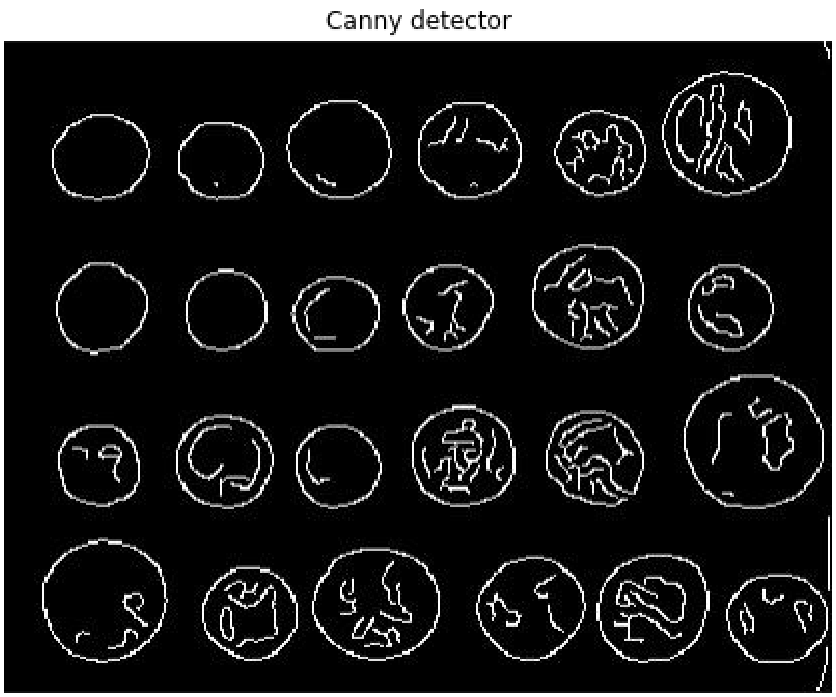
coins **=** data**.**coins()

```
#print(np.max(coins), np.min(coins), np.mean(coins))
hist = np.histogram(coins, bins=np.arange(0, 256), normed=True)
#print(hist)

fig, axes = plt.subplots(1, 2,figsize=(20, 10))
axes[0].imshow(coins, cmap=plt.cm.gray, interpolation='nearest')

axes[0].axis('off')
axes[1].plot(hist[1][:-1], hist[0], lw=2)
axes[1].set_title('histogram of gray values')

 plt.show()
```

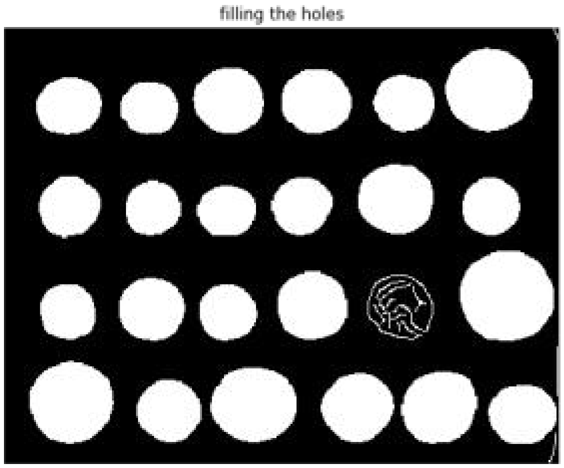



```
edges = canny(coins, sigma=2)


fig, ax = plt.subplots(figsize=( 10, 6))
```

```
ax.imshow(edges, cmap=plt.cm.gray, interpolation='nearest')
ax.set_title('Canny detector')
ax.axis('off')
plt.show()
```




```
from scipy import ndimage as ndi

fill_coins = ndi.binary_fill_holes(edges)

fig, ax = plt.subplots(figsize=( 10, 6))
ax.imshow(fill_coins, cmap=plt.cm.gray, interpolation='nearest')
ax.set_title('filling the holes')
ax.axis('off')
plt.show()
```



```
ax.set_title('segmentation')

ax.axis('off')
plt.show()
```