MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Lab Report

Course Code	: CSE-4106
Course Title	: Digital Image Processing and Computer Vision Lab
Report Title	: Image Processing Operations with Python

Submission Date: May 2019

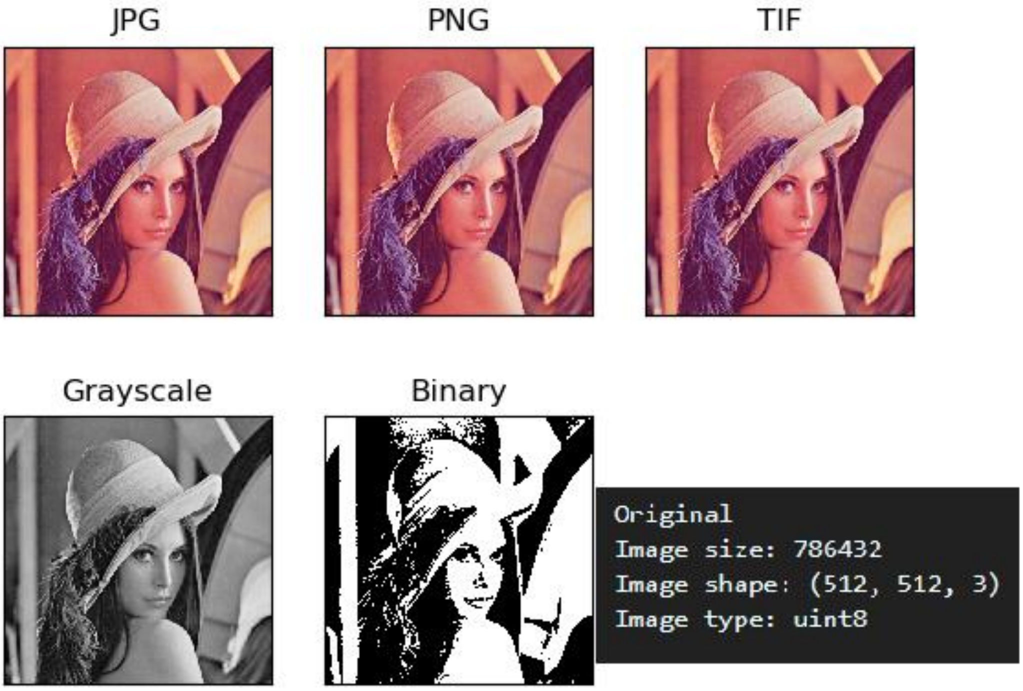
Submitted by	Submitted to
Name : Md. Ashef Shahrior **ID** : CE-15007 **Batch** : 4th YEAR 1st SEMESTER **Department** : Computer Science and Engineering	**Dr. Mohammad Motiur Rahman** **Professor** **Dept. Of Computer Science & Engineering** **MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY**

Lab 1 Solution

Code

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

def info(img):
        print("Original")
        print('Image size:', img.size)
        print('Image shape:', img.shape)
        print('Image type:', img.dtype)

def display(imgs):
        plt.figure('Basic Image Formats')
        titles = ['JPG', 'PNG', 'TIF', 'Grayscale', 'Binary']

        for i in range(5):
                plt.subplot(2,3,i+1), plt.imshow(cv2.cvtColor(imgs[i],
cv2.COLOR_BGR2RGB)), plt.title(titles[i])
                plt.xticks([]), plt.yticks([])
        plt.show()

def binaryImage(img):
        gre = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        bina = np.zeros(gre.shape,gre.dtype)
        for i in range(gre.shape[0]):
                for j in range(gre.shape[1]):
                        if gre[i,j] > 127:
                                bina[i,j] = 255
                        else: bina[i,j] = 0
        return bina

if __name__ == '__main__':
        img = cv2.imread('lena_color.jpg')
        info(img)
        cv2.imwrite('Lena_png.png', img)
        cv2.imwrite('Lena_tif.tif', img)
        png = cv2.imread('Lena_png.png')
        tif = cv2.imread('Lena_tif.tif')
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        binary = binaryImage(img)

        imgs = [img, png, tif, gray, binary]
        display(imgs)
```

Output



```
Original
Image size: 786432
Image shape: (512, 512, 3)
Image type: uint8
```

Lab 2 Solution

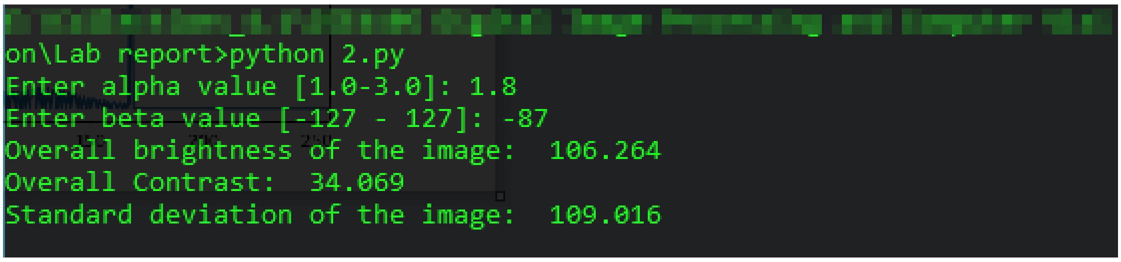
Code

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

# g(x) = alpha*f(x) + beta
# g(x) - output image and f(x) - input image
# alpha - contrast
# beta - brightness
# alpha 1  beta 0      --> no change
# 0 < alpha < 1        --> lower contrast
# alpha > 1            --> higher contrast
# -127 < beta < +127   --> good range for brightness values

def modify(img):
        des = cv2.multiply(img, np.array([alpha]))
        des = cv2.add(des,beta)
        return des

def display_img(src,des):
        plt.figure('Brightness and Contrast Analysis')
        plt.subplot(221), plt.imshow(cv2.cvtColor(src, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
        plt.xticks([]), plt.yticks([])

        plt.subplot(222), plt.imshow(cv2.cvtColor(des, cv2.COLOR_BGR2RGB)),
plt.title('Modified Brightness and Contrast')
        plt.xticks([]), plt.yticks([])
```

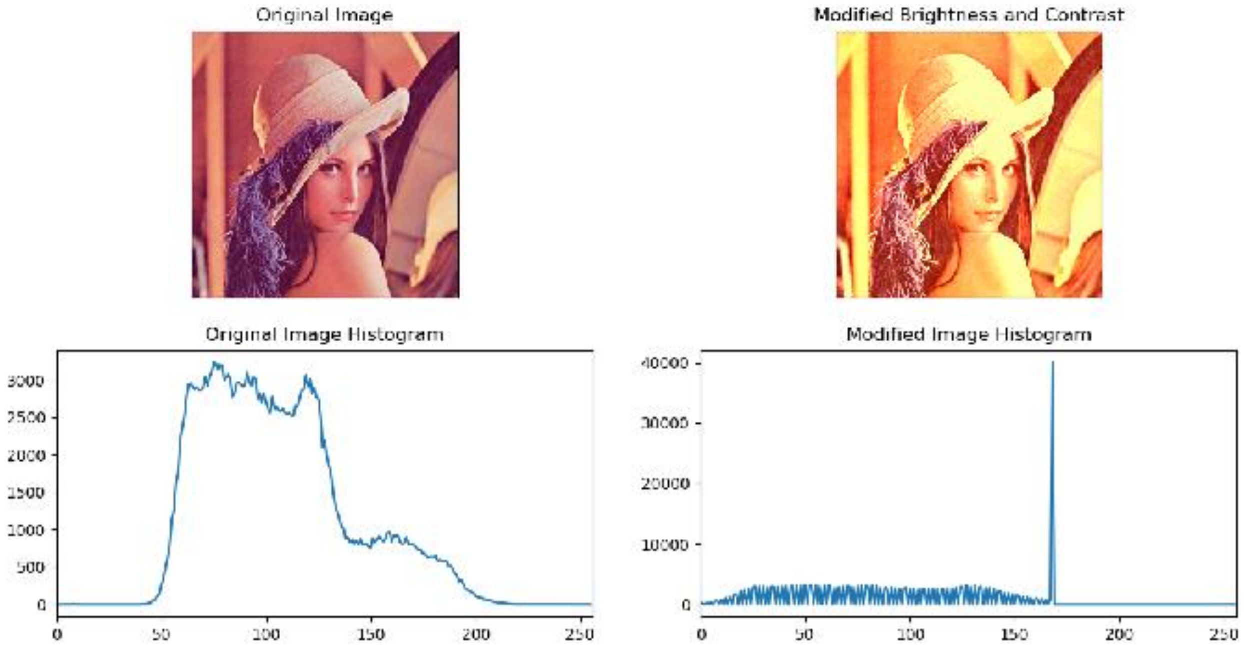
```
def display_hist(src,des):
        hist_src = cv2.calcHist([src],[0],None,[256],[0,256])
        hist_des = cv2.calcHist([des],[0],None,[256],[0,256])

        plt.subplot(223), plt.plot(hist_src), plt.title('Original Image
Histogram')
        plt.xlim([0,256])
        plt.subplot(224), plt.plot(hist_des), plt.title('Modified Image
Histogram')
        plt.xlim([0,256])

def calc_brightness(src):
        total_pixels = 0
        total_brightness = 0
        for i in range(src.shape[0]):
                for j in range(src.shape[1]):
                        brightness = 0
                        r,g,b = src[i][j]
                        total_brightness += r*0.2126 + g * 0.7152 + b *
0.0722
                        # The formula reflects the luminosity function:
                        # green light contributes the most to the intensity
perceived by humans,
                        # and blue light the least.
                        total_pixels +=1
        # print(total_pixels)
        global pixels
        pixels = total_pixels
        avg_brightness = total_brightness/total_pixels
        avg_brightness = round(avg_brightness,3)
        print('Overall brightness of the image: ',avg_brightness)
        return avg_brightness

# calculating contrast
# sqrt(1/(h*w)*(f(x)-beta)**2) - overall contrast value

def calc_contrast(src, brightness):
        arr = np.array(src)
        t = 0
        for i in range(arr.shape[0]):
                for j in range(arr.shape[1]):
                        t += (arr[(i,j,0)]-brightness)**2
        sz = pixels
        contrast = t/sz
        contrast = math.sqrt(contrast)
        print('Overall Contrast: ', round(contrast,3))

def calc_standard_deviation(src, brightness):
        dif = (src - brightness)**2
        total_sum = 0
        for i in range(dif.shape[0]):
                for j in range(dif.shape[1]):
                        r,g,b = dif[i][j]
                        total_sum+= r + g + b

        variance = total_sum/pixels
        st_deviation = math.sqrt(variance)
        print('Standard deviation of the image: ',round(st_deviation,3))

pixels = 0
```

```
if __name__ == '__main__':
        alpha = 1.0
        beta = 0.0

        alpha = float(input('Enter alpha value [1.0-3.0]: '))
        beta = int(input('Enter beta value [-127 - 127]: '))

        src = cv2.imread('lena_color.jpg')

        des = modify(src)

        display_img(src,des)

        display_hist(src,des)

        brightness = calc_brightness(src)

        calc_contrast(src, brightness)

        calc_standard_deviation(src, brightness)

        plt.show()

        cv2.waitKey()
```

Output



```
[redacted]
on\Lab report>python 2.py
Enter alpha value [1.0-3.0]: 1.8
Enter beta value [-127 - 127]: -87
Overall brightness of the image:  106.264
Overall Contrast:  34.069
Standard deviation of the image:  109.016
```

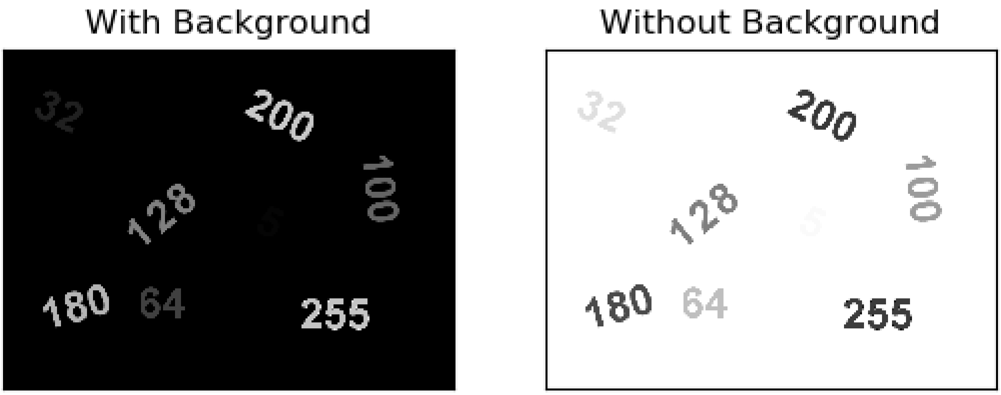


Lab 3 Solution

Background Elimination Code

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

def display(img1,img2):
        plt.figure('Background Elimination')
        plt.subplot(121), plt.imshow(cv2.cvtColor(img1,
cv2.COLOR_BGR2RGB)), plt.title('With Background')
        plt.xticks([]), plt.yticks([])
        plt.subplot(122), plt.imshow(cv2.cvtColor(img2,
cv2.COLOR_BGR2RGB)), plt.title('Without Background')
        plt.xticks([]), plt.yticks([])
        plt.show()

def modify(img1,img2):
        flag = False
        dif = cv2.subtract(img2,img1)
        #cv2.imshow('demo',d)
        return dif


if __name__== '__main__':
        img1 = cv2.imread('numbers.png',0)

        x,y = img1.shape
        img2 = np.ones((x,y,1),np.uint8)*255

        display(img1,modify(img1,img2))

        cv2.waitKey()
```

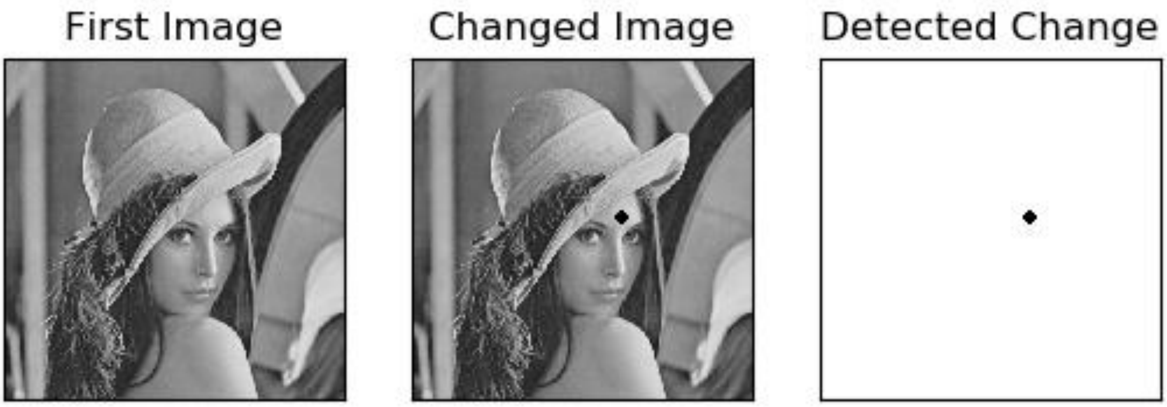
Output



Change Detection Code

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

def display(img1,img2, img3):
        plt.figure('Change Detection')
        plt.subplot(131), plt.imshow(cv2.cvtColor(img1,
cv2.COLOR_BGR2RGB)), plt.title('First Image')
        plt.xticks([]), plt.yticks([])
        plt.subplot(132), plt.imshow(cv2.cvtColor(img2,
cv2.COLOR_BGR2RGB)), plt.title('Changed Image')
        plt.xticks([]), plt.yticks([])
        plt.subplot(133), plt.imshow(cv2.cvtColor(img3,
cv2.COLOR_BGR2RGB)), plt.title('Detected Change')
        plt.xticks([]), plt.yticks([])
        plt.show()

def modify(img1,img2,img3):
        flag = False
        dif = cv2.subtract(img1,img2)
        flag = np.any(img3)
        for i in range(img1.shape[0]):
                for j in range(img2.shape[1]):
                        if img1[i,j] == img2[i,j]:
                                img3[i,j] = 255
                        else:
                                img3[i,j] = 0
        if(flag):
                print("Change is found")
        else: print("Change not found")

if __name__== '__main__':
        img1 = cv2.imread('lena_color.jpg',0)

        img2 = cv2.imread('lena_color2.jpg',0)

        x,y = img1.shape
        #img3 = np.ones((x,y,1),np.uint8)*255
        img3 = np.ones(img1.shape,img1.dtype)

        modify(img1,img2,img3)

        display(img1,img2, img3)

        cv2.waitKey()
```

Output

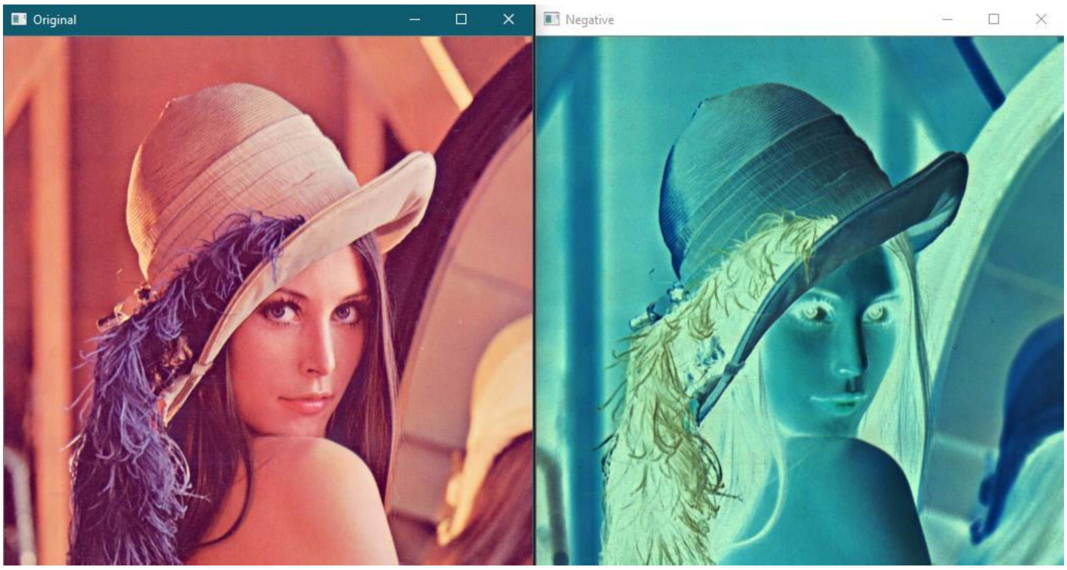


Lab 4 Solution

Image Negative Code

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

img1 = cv2.imread('lena_color.jpg')
img2 = 255 - img1

cv2.imshow('Original',img1)
cv2.imshow('Negative',img2)

cv2.waitKey()
```

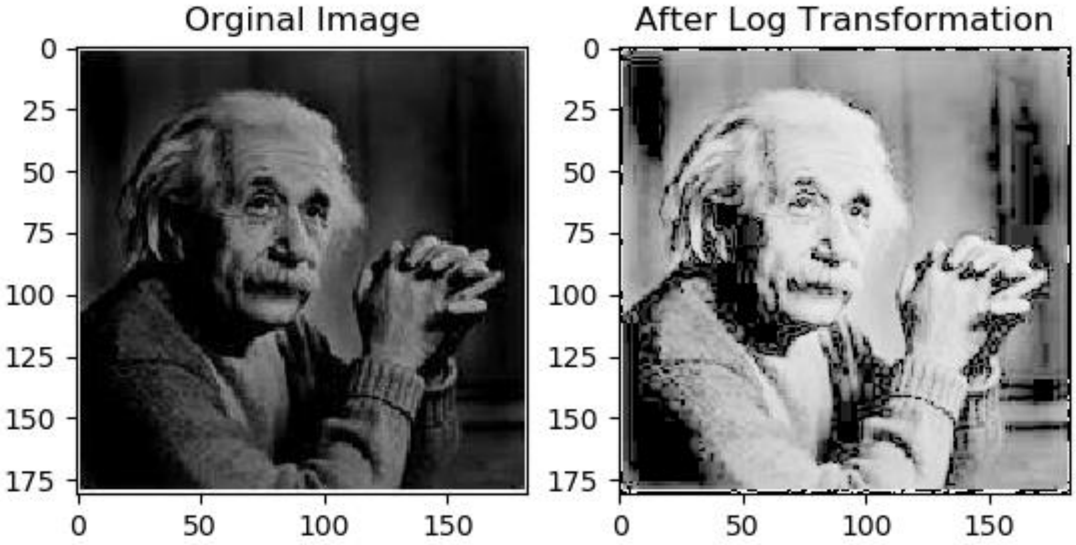
Output



Log Transformation Code

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

img1 = cv2.imread('einstein.jpg')
img2 = np.zeros(img1.shape,img1.dtype)  # Null pixel value

#For log transformation the formula is s=c*log(1+r) where c and r are
constant and r is the pixel value.
#c = 255/(log(1 + max_input_pixel_value))

c = 255/(np.log(1+np.max(img1)))

for i in range(img1.shape[0]):
    for j in range(img1.shape[1]):       # s=c*log(1+image)
        img2[i][j]=c*np.log(1+img1[i][j])


plt.figure('Log Transformation')
plt.subplot(1,2,1).set_title('Orginal Image')
plt.imshow(img1)

plt.subplot(1,2,2).set_title('After Log Transformation')#Showing Image as
ploating
plt.imshow(img2)
plt.show()

cv2.waitKey()
```

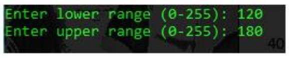
Output



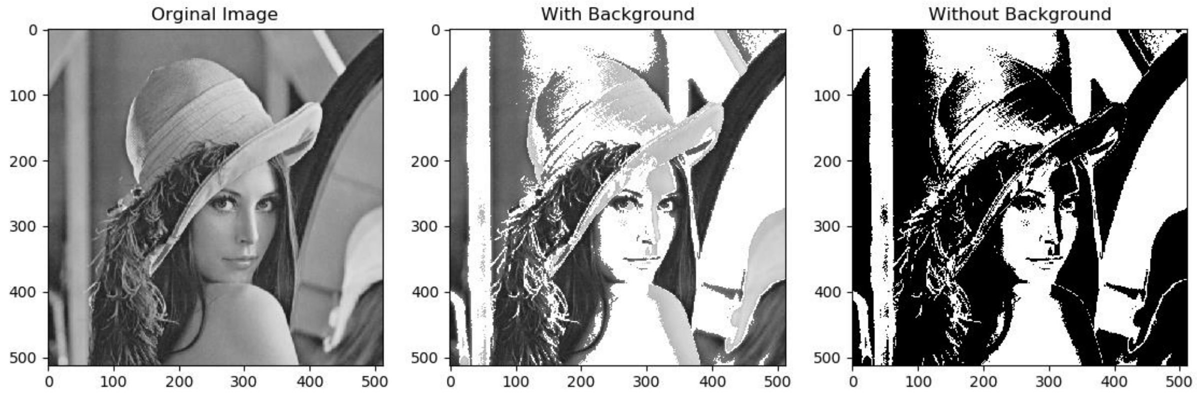
Grey Level Slicing Code

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

img = cv2.imread('lena_color.jpg',0)

img1 = np.zeros(img.shape,img.dtype)
img2 = np.zeros(img.shape,img.dtype)

r1 = int(input('Enter lower range (0-255): '))
r2 = int(input('Enter upper range (0-255): '))

if r1>r2:
        r1,r2 = r2,r1

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if img[i][j] >= r1 and img[i][j] <= r2:
            img1[i][j]=255
            img2[i][j]=255
        else:
            img1[i][j] = img[i][j]      # With background, Set L-1 if
r1<=r<=r2 else set the same pixel value
            img2[i][j] = 0            # Without background, Set L-1 if
r1<=r<=r2 else set 0


plt.figure('Gray Level Slicing')
plt.subplot(1,3,1).set_title('Orginal Image')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

plt.subplot(1,3,2).set_title('With Background')#Showing Image as ploating
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))

plt.subplot(1,3,3).set_title('Without Background')#Showing Image as
ploating
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))

plt.show()

cv2.waitKey()
```

Output



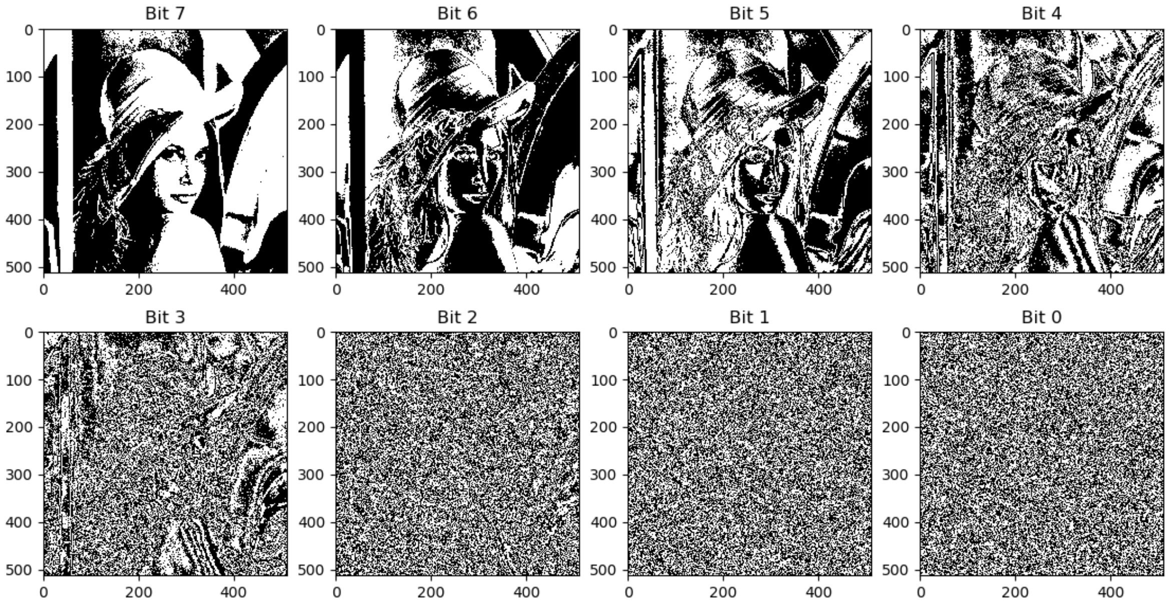
```
Enter lower range (0-255): 120
Enter upper range (0-255): 180
```



Bit plane Slicing Code

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

if __name__ == '__main__':
        img = cv2.imread('lena_color.jpg',cv2.IMREAD_GRAYSCALE) # Read the
image in greyscale
        ims = cv2.resize(img,(256,256))
        #cv2.imshow('Original Image',ims)
        eight_bit_img = seven_bit_img = six_bit_img = five_bit_img =
four_bit_img = three_bit_img = two_bit_img = one_bit_img = 0
        img_bits = [eight_bit_img, seven_bit_img, six_bit_img,
five_bit_img, four_bit_img, three_bit_img, two_bit_img, one_bit_img]
        val = [128, 64, 32, 16, 8, 4, 2, 1]
        # Iterate over each pixel and change pixel value to binary using
np.binary_repr() and store it in a list.
        lst = []
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                lst.append(np.binary_repr(img[i][j], width=8))   # width =
no. of bits
        for j in range(8):
                img_bits[j] = (np.array([int(i[j]) for i in lst],
dtype=np.uint8) * val[j]).reshape(img.shape[0], img.shape[1])
        plt.figure("The Image Slices")
        nm = ['Bit 7','Bit 6','Bit 5','Bit 4','Bit 3','Bit 2','Bit 1','Bit
0']
        bits = [eight_bit_img, seven_bit_img, six_bit_img, five_bit_img,
four_bit_img, three_bit_img, two_bit_img, one_bit_img]

        for i in range(8):
                plt.subplot(2,4,i+1).set_title(nm[i])
                plt.imshow(img_bits[i], cmap='gray')
        plt.show()
```

Output

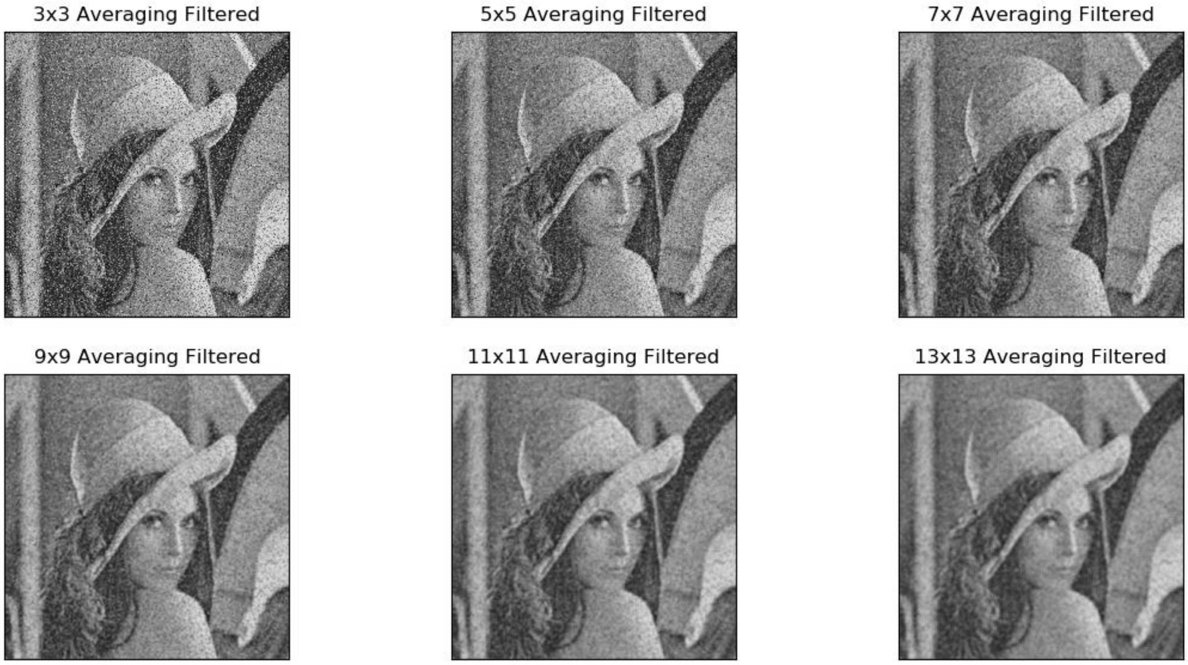


Lab 5 Solution

Averaging Filtering (I) Code

```
#averaging filter with different mask size

import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('lena-noise.jpg')

mask_size = [3,5,7,9,11,13]
x = 231

ims = cv2.resize(img,(256,256))
cv2.imshow('Original Image',ims)

for n in mask_size:
    #mask = np.ones((n,n),np.float32)/(n*n)
    #averaging = cv2.filter2D(img,-1,mask)
    averaging = cv2.blur(img, (n, n))
    plt.subplot(x),plt.imshow(averaging),plt.title('%dx%d Averaging
Filtered'%(n,n))
    plt.xticks([]), plt.yticks([])
    x+=1

plt.show()
```

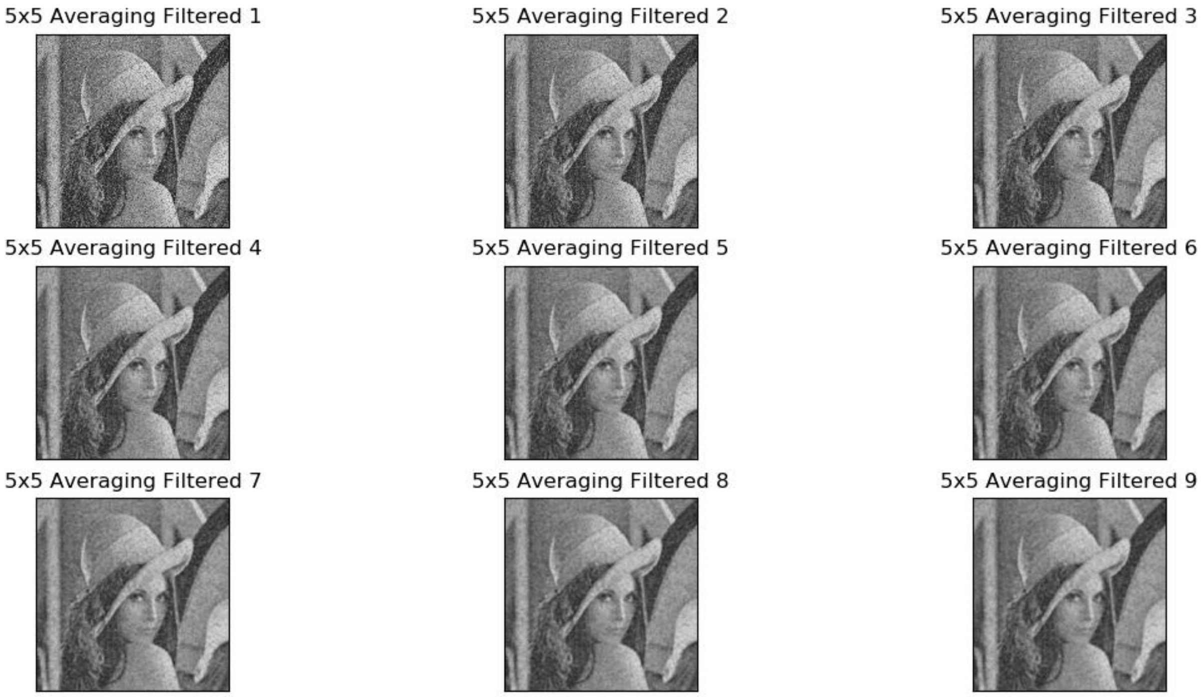
Output



Averaging Filtering (II) Code

```
# averaging filter with same mask size multiple times
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('lena-noise.jpg')
n = int(input('Enter mask size: '))
x = 331

ims = cv2.resize(img,(256,256))
cv2.imshow('Original Image',ims)

mask = np.ones((n,n),np.float32)/(n*n)
averaging = cv2.filter2D(img,-1,mask)

for i in range(9):
    plt.subplot(x),plt.imshow(averaging),plt.title('%dx%d Averaging
Filtered %d'%(n,n,i+1))
    plt.xticks([]), plt.yticks([])
    x+=1
    averaging = cv2.filter2D(averaging,-1,mask)
plt.show()
```

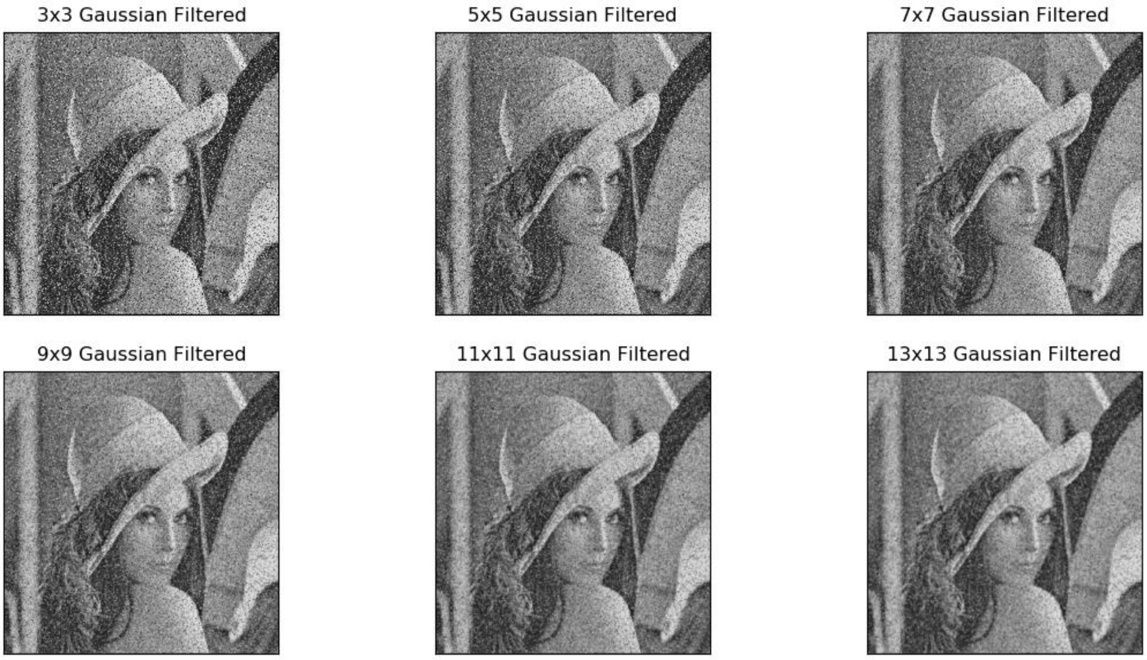
Output



Gaussian Filter with Different Mask Size Code

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

img = cv2.imread('lena-noise.jpg')

mask_size = [3,5,7,9,11,13]
x = 231

cv2.imshow('Original Image',cv2.resize(img,(256,256)))

for n in mask_size:
    gauss = cv2.GaussianBlur(img,(n,n),0)
    plt.subplot(x),plt.imshow(gauss),plt.title('%dx%d Gaussian
Filtered'%(n,n))
    plt.xticks([]), plt.yticks([])
    x+=1
plt.show()
```

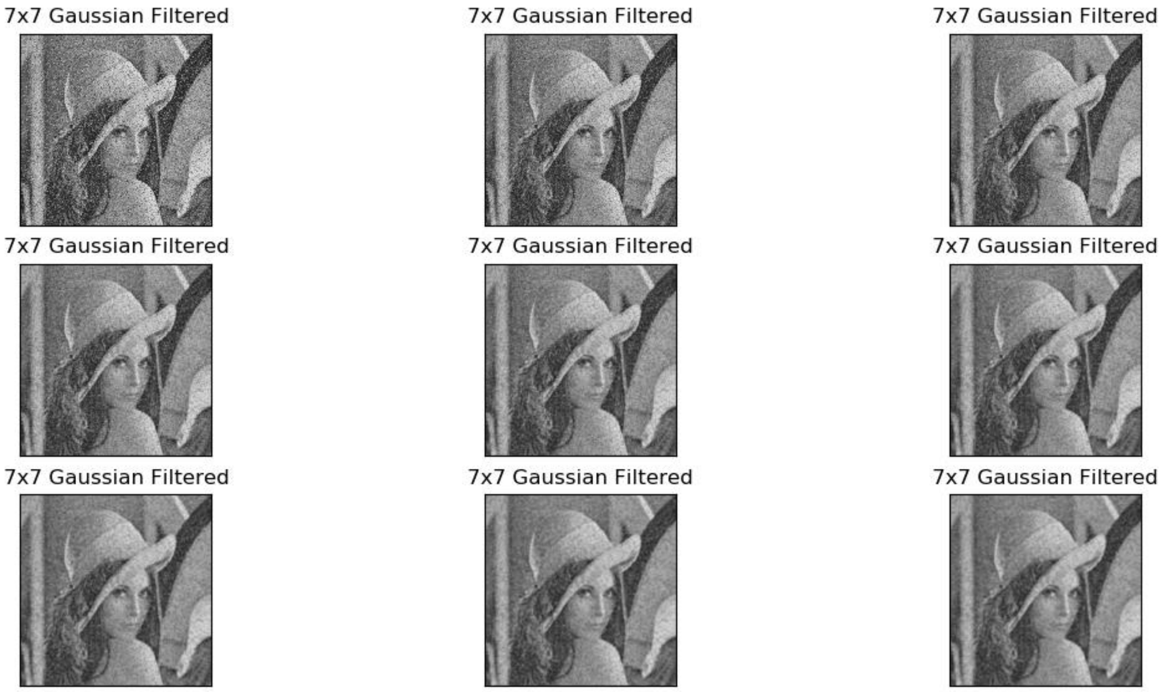
Output



Gaussian Filter Iteratively with Same Mask Size Code

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

img = cv2.imread('lena-noise.jpg')
n = int(input('Enter mask size: '))
x = 331

cv2.imshow('Original Image',cv2.resize(img,(256,256)))

gauss = cv2.GaussianBlur(img,(n,n),0)

for i in range(9):
    plt.subplot(x),plt.imshow(gauss),plt.title('%dx%d Gaussian
Filtered'%(n,n))
    plt.xticks([]), plt.yticks([])
    gauss = cv2.GaussianBlur(gauss,(n,n),0)
    x+=1
plt.show()
```

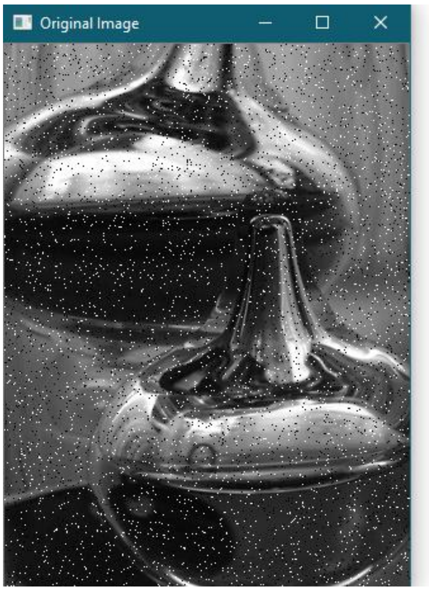
Output

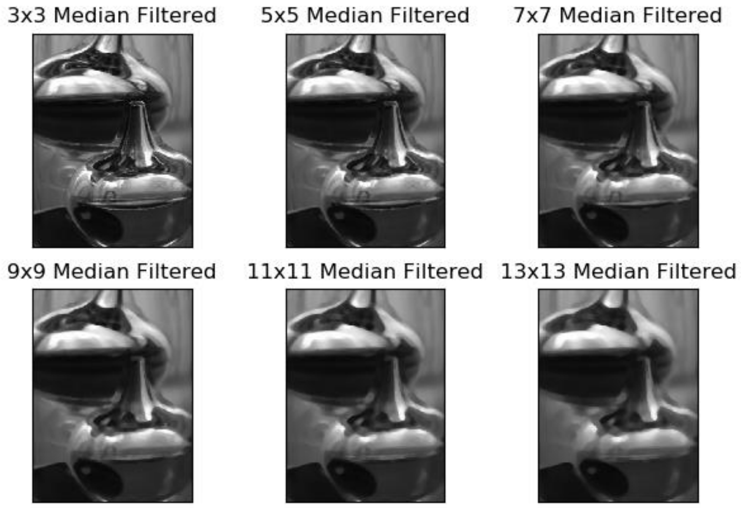


Median Filtering with Different Mask Size Code

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

img = cv2.imread('salt-pepper.png')

mask_size = [3,5,7,9,11,13]
x = 231

cv2.imshow('Original Image',img)

for n in mask_size:
    median = cv2.medianBlur(img,n)
    plt.subplot(x),plt.imshow(median),plt.title('%dx%d Median
Filtered'%(n,n))
    plt.xticks([]), plt.yticks([])
    x+=1

plt.show()
```

Output

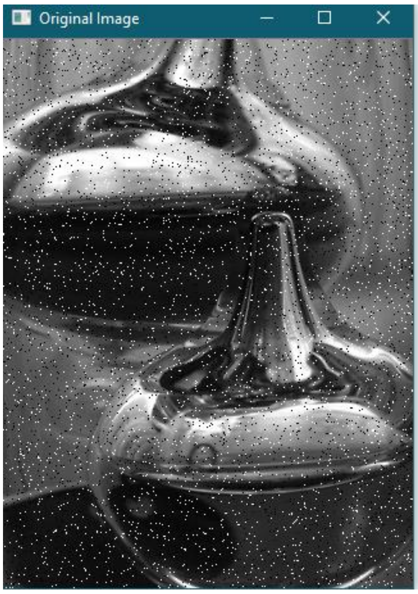


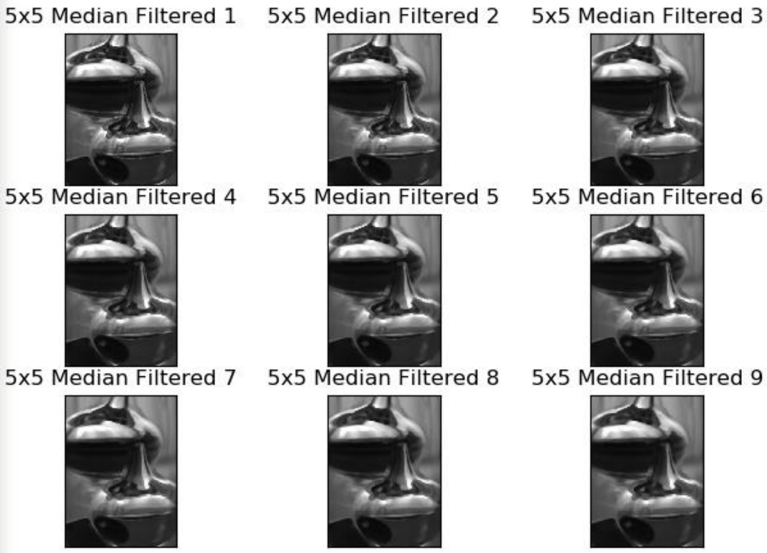


Median Filtering with Same Mask Size Iteration Code

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

img = cv2.imread('salt-pepper.png')
n = int(input('Enter mask size: '))
x = 331

cv2.imshow('Original Image',img)

median = cv2.medianBlur(img,n)

for i in range(9):
    plt.subplot(x),plt.imshow(median),plt.title('%dx%d Median Filtered
%d'%(n,n,i+1))
    plt.xticks([]), plt.yticks([])
    x+=1
    median = cv2.medianBlur(median,n)
plt.show()
```

Output



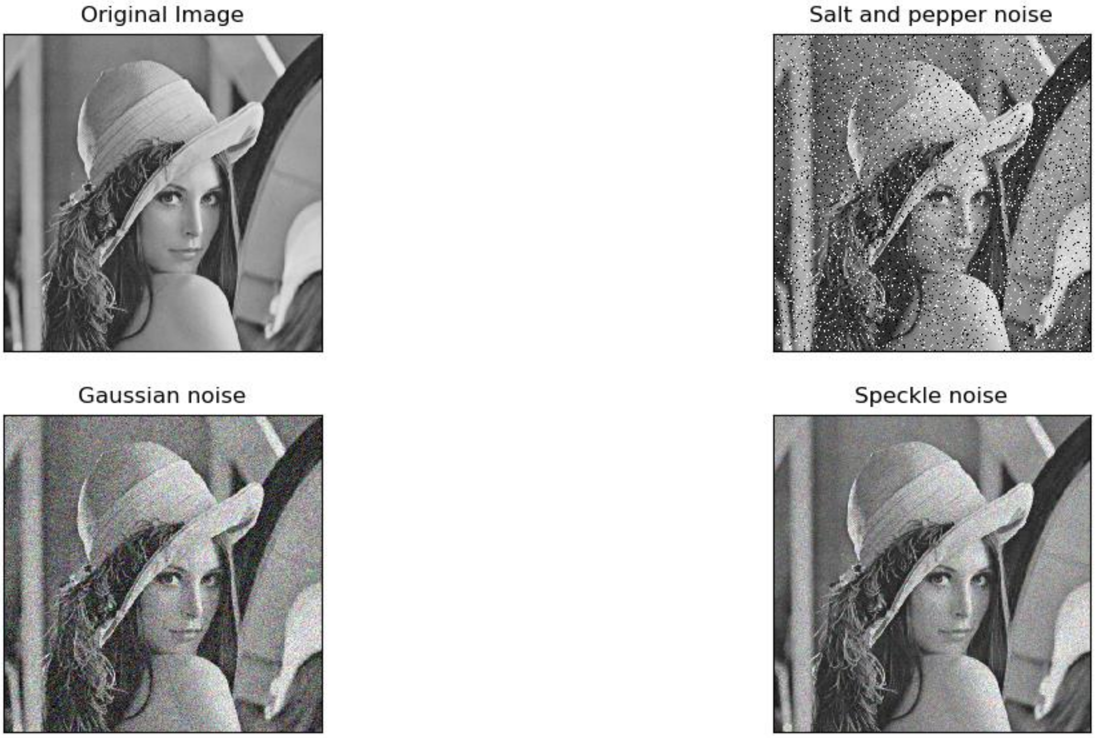


Lab 6 Solution

Various types of Noise Code

```
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from scipy.ndimage import filters
from scipy import misc
import cv2
import random

def to_std_float(img):
        #Converts img to 0 to 1 float to avoid wrapping that occurs with
uint8
        img.astype(np.float16, copy = False)
        img = np.multiply(img, (1/255))
        return img

def to_std_uint8(img):
        # Properly handles the conversion to uint8
        img = cv2.convertScaleAbs(img, alpha = (255/1))
        return img

def noise_generator (noise_type,image):
        row, col, ch = image.shape
        mean = 0.
        var = 0.01
        sigma = var**0.5
        if noise_type == "s&p":
                output = np.zeros(image.shape,np.uint8)
                prob = 0.05
                thres = 1 - prob
                for i in range(image.shape[0]):
                    for j in range(image.shape[1]):
                        rdn = random.random()
                        if rdn < prob:
                            output[i][j] = 0
                        elif rdn > thres:
                            output[i][j] = 255
                        else:
                            output[i][j] = image[i][j]
                return output
        elif noise_type == "gauss":
                image = to_std_float(image)
                gauss = np.random.normal(mean, sigma, image.shape)
                noisy = image + gauss
                #return noisy.astype('uint8')
                return to_std_uint8(noisy)
        elif noise_type == "speckle":
                image = to_std_float(image)
                noise = np.random.normal(mean, sigma, image.shape)
                out = image + image * noise
                return to_std_uint8(out)
        else:
                return image

im = cv2.imread('lena_gray.jpg')
sp_im = noise_generator('s&p', im)
gauss_im = noise_generator('gauss', im)
speckle_im = noise_generator('speckle', im)

imgs = [im, sp_im, gauss_im, speckle_im]
titles = ['Original Image','Salt and pepper noise','Gaussian
noise','Speckle noise']

plt.figure('Noises')
for i in range(4):
        plt.subplot(2,2,1+i), plt.imshow(cv2.cvtColor(imgs[i],
cv2.COLOR_BGR2RGB)), plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
plt.show()
```

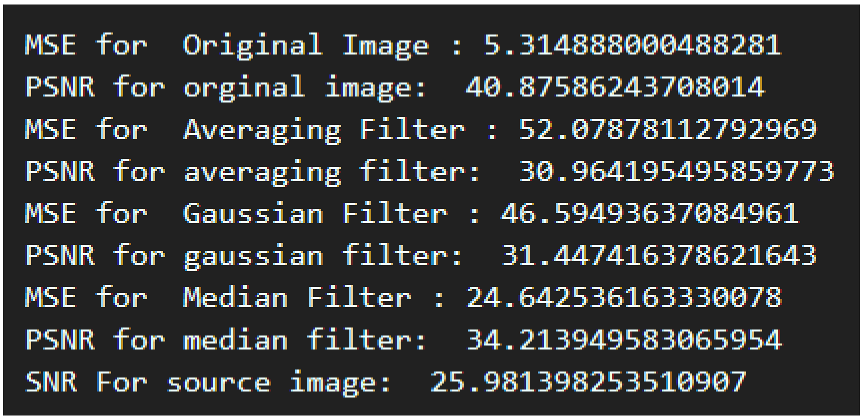
Output



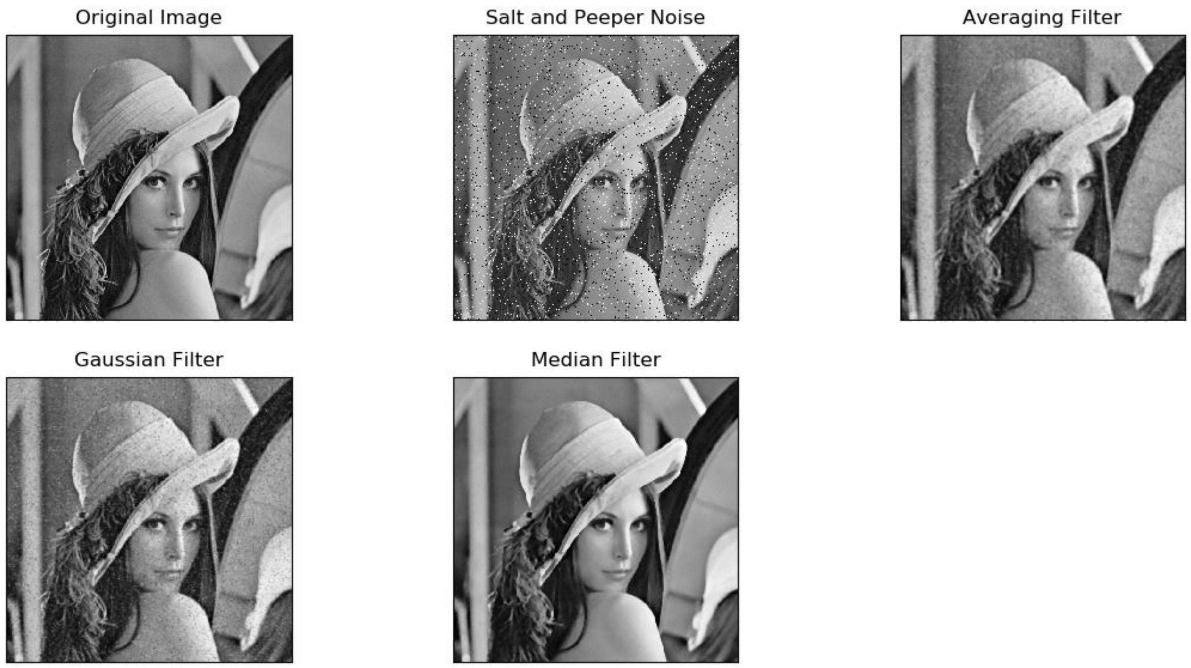
Calculate SNR, MSE and PSNR Code

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


def add_noise(img):
  p=.05
  output=np.zeros(img.shape,np.uint8)
  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      r=random.random()
      if r<p/2:
        #peeper sprinkled
        output[i][j]=0
      elif r<p:
        #salt sprinkled
        output[i][j]=255
      else:
        output[i][j]=img[i][j]
  return output

def psnr(img1, img2, str):
  mse = np.mean((img1-img2)**2)
  print("MSE for ",str,":",mse)
  if mse == 0:
    return 10
  PIXEL_MAX = 255.0
  return 20 * math.log10(PIXEL_MAX/ math.sqrt(mse))

def snr(img1, img2):
  mse = np.mean((img1)** 2)
  mse2=np.mean((img1-img2)**2)
  return 20 * math.log10(mse/mse2)

if __name__ == '__main__':
  img=cv2.imread("lena_color.jpg",0)
  noisy = add_noise(img)
  blur = cv2.blur(noisy,(5,5))
  gauss = cv2.GaussianBlur(noisy,(5,5),0)
  median = cv2.medianBlur(noisy,5)

  print("PSNR for orginal image: ",psnr(img,noisy,'Original Image'))
  print("PSNR for averaging filter: ",psnr(blur,noisy,'Averaging Filter'))
  print("PSNR for gaussian filter: ",psnr(gauss,noisy,'Gaussian Filter'))
  print("PSNR for median filter: ",psnr(median,noisy,'Median Filter'))
  print("SNR For source image: ",snr(img,noisy))

  imgs = [img, noisy, blur, gauss, median]
  titles = ['Original Image', 'Salt and Peeper Noise', 'Averaging Filter',
'Gaussian Filter', 'Median Filter']

  plt.figure("Noise removing techniques")
  for i in range(5):
    plt.subplot(2,3,i+1).set_title(titles[i])
    plt.imshow(imgs[i],cmap="gray")
    plt.xticks([]), plt.yticks([])
  plt.show()
```

Output



```
MSE for  Original Image : 5.314888000488281
PSNR for orginal image:  40.87586243708014
MSE for  Averaging Filter : 52.07878112792969
PSNR for averaging filter:  30.964195495859773
MSE for  Gaussian Filter : 46.59493637084961
PSNR for gaussian filter:  31.447416378621643
MSE for  Median Filter : 24.642536163330078
PSNR for median filter:  34.213949583065954
SNR For source image:  25.981398253510907
```



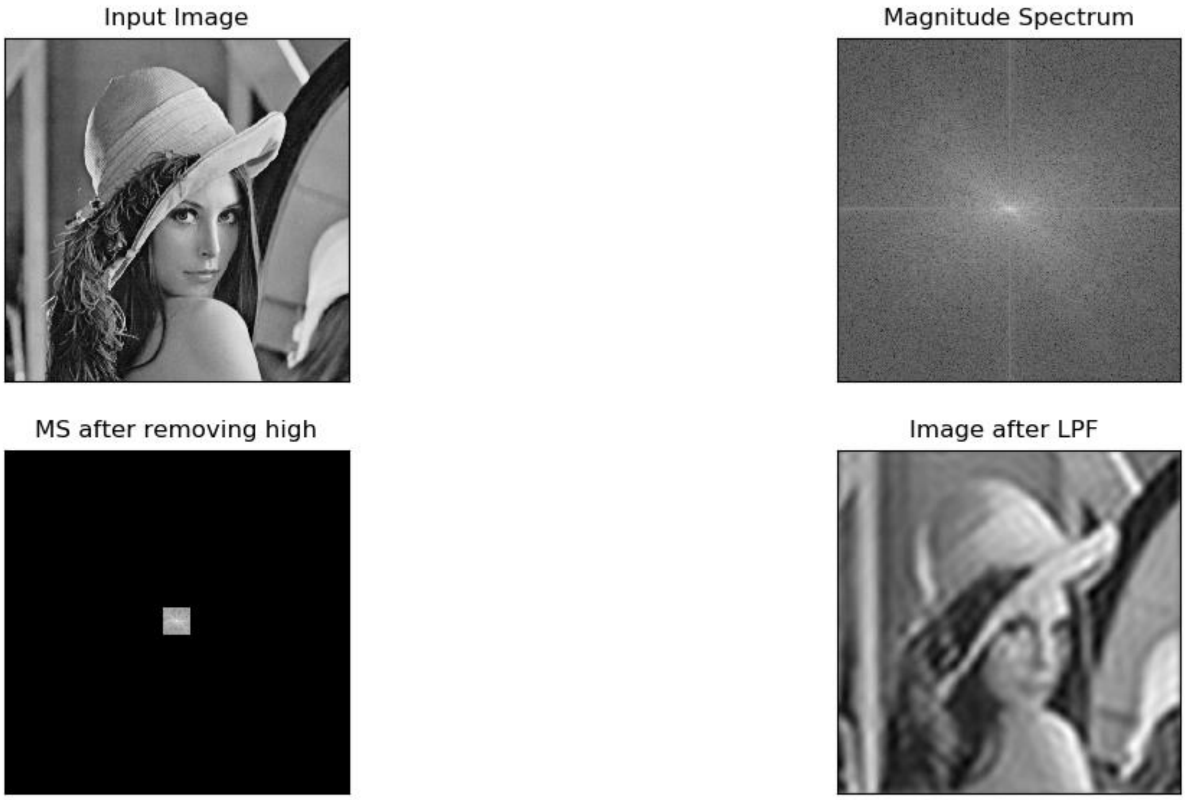
Lab 7 solution

LPF Code

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

def apply_mask(img, fshift):
        rows, cols = img.shape
        crow = (int)(rows/2)
        ccol = (int)(cols/2)

        # 80x80 mask to remove high frequency
        rad = 20 # Radius
        for i in range(0,crow-rad):
                for j in range(0,cols):
                        fshift[i,j] = 1

        for i in range(crow+rad,rows):
                for j in range(0,cols):
                        fshift[i,j] = 1

        for i in range(crow-rad,crow+rad):
                for j in range(0,ccol-rad):
                        fshift[i,j] = 1

        for i in range(crow-rad,crow+rad):
                for j in range(ccol+rad,cols):
                        fshift[i,j] = 1
        return fshift
```

```
def display(img, magnitude_spectrum, magnitude_spectrum1, img_back):
        plt.subplot(221),plt.imshow(img, cmap = 'gray')
        plt.title('Input Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(222),plt.imshow(magnitude_spectrum, cmap = 'gray')
        plt.title('Magnitude Spectrum'),plt.xticks([]), plt.yticks([])
        plt.subplot(223),plt.imshow(magnitude_spectrum1, cmap = 'gray')
        plt.title('MS after removing high'), plt.xticks([]), plt.yticks([])
        plt.subplot(224),plt.imshow(img_back, cmap = 'gray')
        plt.title('Image after LPF '), plt.xticks([]), plt.yticks([])
        plt.show()

if __name__ == '__main__':
        img = cv2.imread('lena_gray.jpg',0)
        f = np.fft.fft2(img) # returns an array with complex value
        fshift = np.fft.fftshift(f) # making the center low frequency
        magnitude_spectrum = 20*np.log(np.abs(fshift)) # taking the
Magnitude
        fshift = apply_mask(img, fshift)
        magnitude_spectrum1 = 20*np.log(np.abs(fshift)) # magnitude after
removing the High frequency

        # applying the reverse fourier Transform
        f_ishift = np.fft.ifftshift(fshift)
        img_back = np.fft.ifft2(f_ishift)
        img_back = np.abs(img_back)

        display(img, magnitude_spectrum, magnitude_spectrum1, img_back)
```

Output

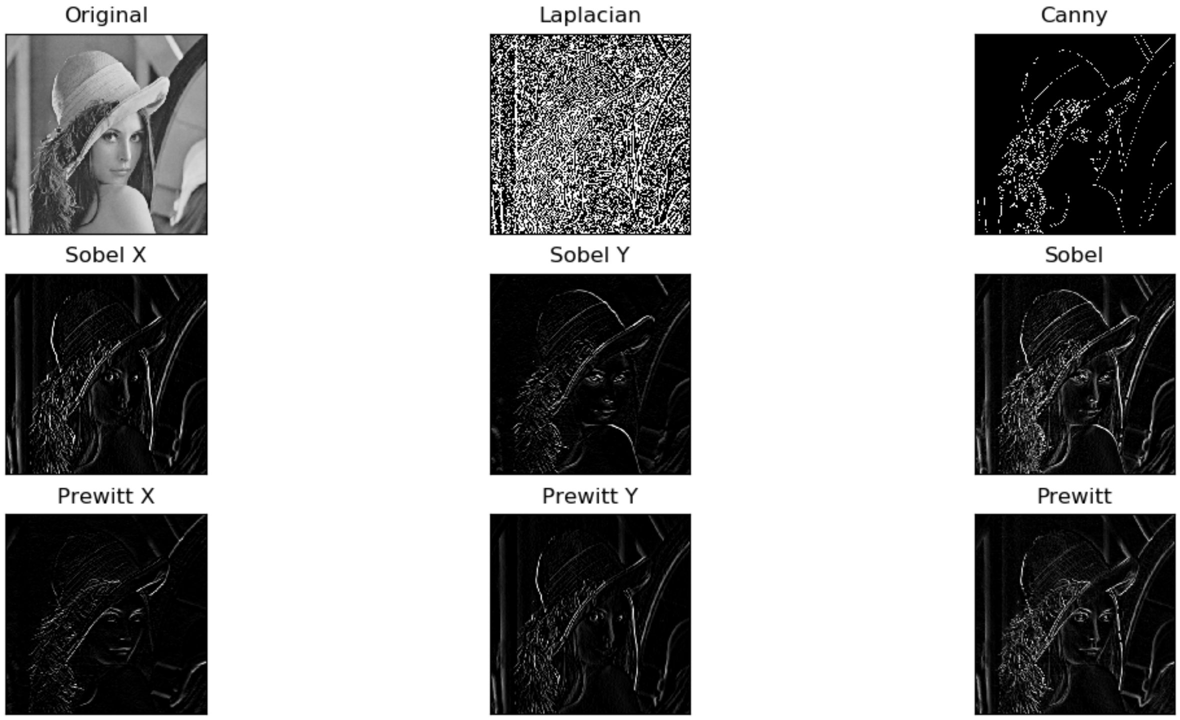


Lab 8 solution

Different Derivative Filters Code

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

src = cv2.imread('lena_gray.jpg')

img = cv2.GaussianBlur(src,(3,3),0) #remove noise

laplacian = cv2.Laplacian(img,cv2.CV_64F)

img_canny = cv2.Canny(img,100,200)

img_sobelx = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=3)
img_sobely = cv2.Sobel(img,cv2.CV_8U,0,1,ksize=3)
img_sobel = img_sobelx + img_sobely

kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
img_prewittx = cv2.filter2D(img, -1, kernelx)
img_prewitty = cv2.filter2D(img, -1, kernely)
img_prewitt = img_prewittx + img_prewitty

imgs = [src, laplacian, img_canny, img_sobelx, img_sobely, img_sobel,
img_prewittx, img_prewitty, img_prewitt]
titles = ['Original','Laplacian','Canny','Sobel X','Sobel
Y','Sobel','Prewitt X','Prewitt Y','Prewitt']

for i in range(9):
        plt.subplot(3,3,i+1),plt.imshow(imgs[i],cmap="gray"),plt.title(titl
es[i])
        plt.xticks([]), plt.yticks([])
plt.show()
```

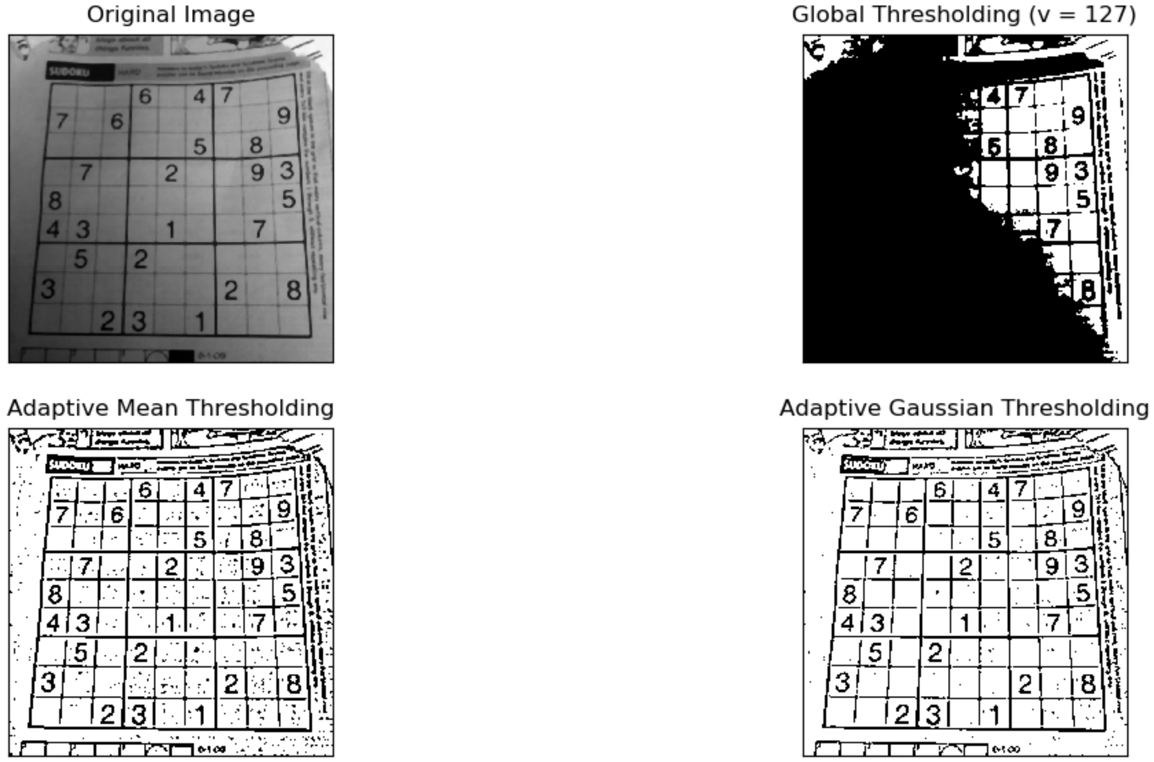
Output



Adaptive Thresholding Code

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

img = cv2.imread('sudoku.jpg',0)
img = cv2.medianBlur(img,5)
ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

Output

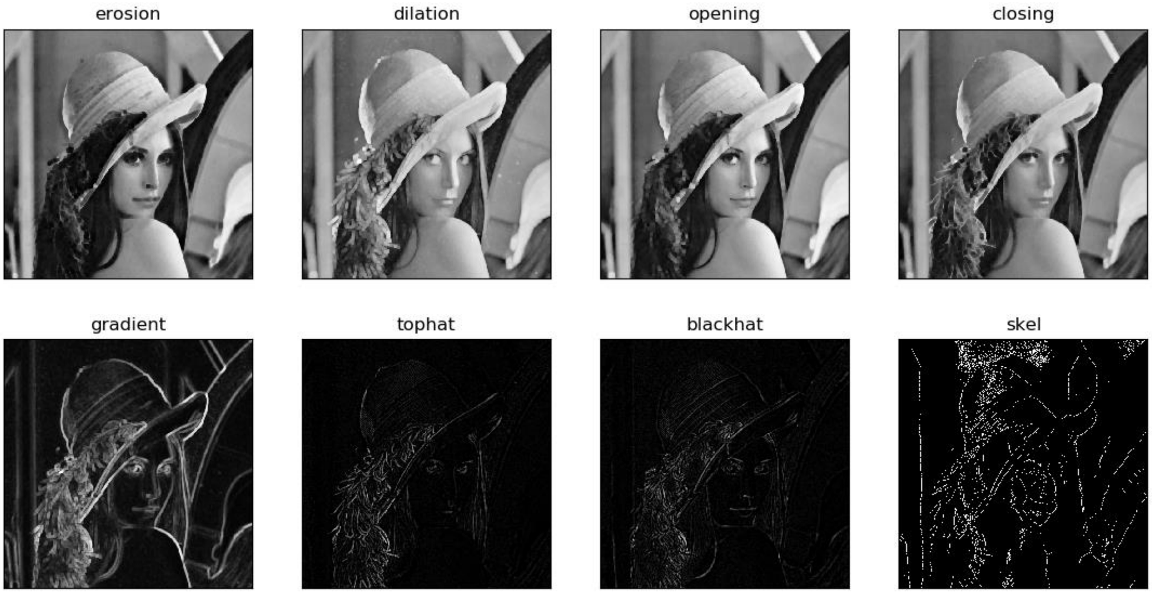

Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Lab 9 Solution

Morphological Operation Code

```
import cv2
import numpy as np
from skimage.morphology import skeletonize
from skimage import data
import matplotlib.pyplot as plt
from skimage.util import invert

img = cv2.imread('lena_gray.jpg',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
dilation = cv2.dilate(img,kernel,iterations = 1)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)

size = np.size(img)
skel = np.zeros(img.shape,np.uint8)

ret,img = cv2.threshold(img,127,255,0)
element = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
done = False
while( not done):
    eroded = cv2.erode(img,element)
    temp = cv2.dilate(eroded,element)
    temp = cv2.subtract(img,temp)
    skel = cv2.bitwise_or(skel,temp)
    img = eroded.copy()
    zeros = size - cv2.countNonZero(img)
    if zeros==size:
        done = True

ops = [erosion, dilation, opening, closing, gradient, tophat, blackhat,
skel]
titles = ["erosion", "dilation", "opening", "closing", "gradient",
"tophat", "blackhat", "skel"]

for i in range(8):
    plt.subplot(2,4,i+1),plt.imshow(ops[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()

cv2.waitKey()
```

Output



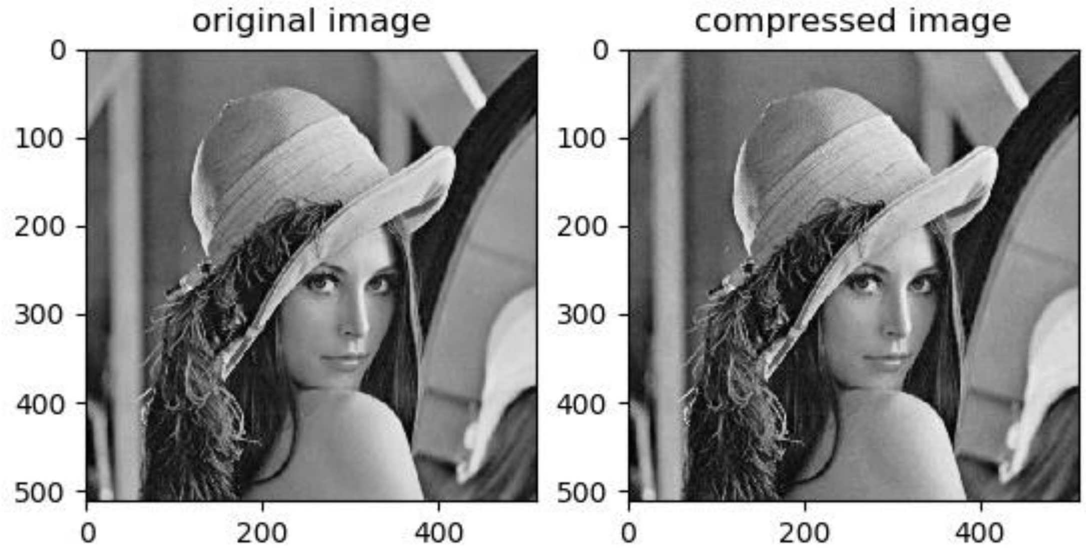
Lab 10 Solution

Bit Plane Slicing Compression Code

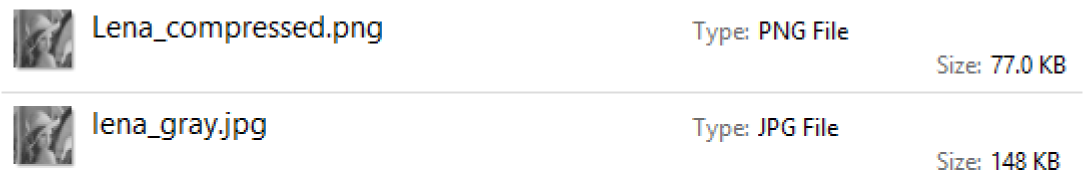
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

```
def apply_mask(img, fshift):
        rows, cols = img.shape
        crow = (int)(rows/2)
        ccol = (int)(cols/2)

        # 80x80 mask to remove high frequency
        rad = 20 # Radius
        for i in range(0,crow-rad):
                for j in range(0,cols):
                        fshift[i,j] = 1

        for i in range(crow+rad,rows):
                for j in range(0,cols):
                        fshift[i,j] = 1

        for i in range(crow-rad,crow+rad):
                for j in range(0,ccol-rad):
                        fshift[i,j] = 1

        for i in range(crow-rad,crow+rad):
                for j in range(ccol+rad,cols):
                        fshift[i,j] = 1
        return fshift

def display(img, magnitude_spectrum, magnitude_spectrum1, img_back):
        plt.subplot(221),plt.imshow(img, cmap = 'gray')
        plt.title('Input Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(222),plt.imshow(magnitude_spectrum, cmap = 'gray')
        plt.title('Magnitude Spectrum'),plt.xticks([]), plt.yticks([])
        plt.subplot(223),plt.imshow(magnitude_spectrum1, cmap = 'gray')
        plt.title('MS after removing high'), plt.xticks([]), plt.yticks([])
        plt.subplot(224),plt.imshow(img_back, cmap = 'gray')
        plt.title('Image after LPF '), plt.xticks([]), plt.yticks([])
        plt.show()

if __name__ == '__main__':
        img = cv2.imread('lena_gray.jpg',0)
        f = np.fft.fft2(img) # returns an array with complex value
        fshift = np.fft.fftshift(f) # making the center low frequency
        magnitude_spectrum = 20*np.log(np.abs(fshift)) # taking the
Magnitude
        fshift = apply_mask(img, fshift)
        magnitude_spectrum1 = 20*np.log(np.abs(fshift)) # magnitude after
removing the High frequency

        # applying the reverse fourier Transform
        f_ishift = np.fft.ifftshift(fshift)
        img_back = np.fft.ifft2(f_ishift)
        img_back = np.abs(img_back)

        display(img, magnitude_spectrum, magnitude_spectrum1, img_back)
```

Output



After compression of the source image -


Lena_compressed.png Type: PNG File Size: 77.0 KB

lena_gray.jpg Type: JPG File Size: 148 KB

Huffman Compression Code

```
import numpy as np
from scipy.misc import imread,imresize
import matplotlib.pyplot as plt
from operator import itemgetter, attrgetter
import queue

class Node:
        def __init__(self):
                self.prob = None
                self.code = None
                self.data = None
                self.left = None
                self.right = None      # the color (the bin value) is only
required in the leaves
        def __lt__(self, other):
                if (self.prob < other.prob):          # define rich
comparison methods for sorting in the priority queue
                        return 1
                else:
                        return 0
        def __ge__(self, other):
                if (self.prob > other.prob):
                        return 1
                else:
                       return 0
def rgb2gray(img):
        gray_img = np.rint(img[:,:,0]*0.2989 + img[:,:,1]*0.5870 +
img[:,:,2]*0.1140)
        gray_img = gray_img.astype(int)
        return gray_img

def get2smallest(data):                       # can be used instead of
inbuilt function get(). was not used in  implementation
    first = second = 1;
    fid=sid=0
    for idx,element in enumerate(data):
        if (element < first):
            second = first
            sid = fid
            first = element
            fid = idx
        elif (element < second and element != first):
            second = element
    return fid,first,sid,second

def tree(probabilities):
        prq = queue.PriorityQueue()
        for color,probability in enumerate(probabilities):
                leaf = Node()
                leaf.data = color
                leaf.prob = probability
                prq.put(leaf)

        while (prq.qsize()>1):
                newnode = Node()              # create new node
                l = prq.get()
                r = prq.get()                 # get the smalles probs in
the leaves
                                              # remove the smallest two
leaves
                newnode.left = l              # left is smaller
                newnode.right = r
                newprob = l.prob+r.prob       # the new prob in the new
node must be the sum of the other two
                newnode.prob = newprob
                prq.put(newnode)      # new node is inserted as a leaf,
replacing the other two
        return prq.get()              # return the root node - tree is
complete

def huffman_traversal(root_node,tmp_array,f):        # traversal of the
tree to generate codes
        if (root_node.left is not None):
                tmp_array[huffman_traversal.count] = 1
                huffman_traversal.count+=1
                huffman_traversal(root_node.left,tmp_array,f)
                huffman_traversal.count-=1
        if (root_node.right is not None):
                tmp_array[huffman_traversal.count] = 0
                huffman_traversal.count+=1
                huffman_traversal(root_node.right,tmp_array,f)
                huffman_traversal.count-=1
        else:
                huffman_traversal.output_bits[root_node.data] =
huffman_traversal.count               #count the number of bits for each
```

```
color
                bitstream = ''.join(str(cell) for cell in
tmp_array[1:huffman_traversal.count])
                color = str(root_node.data)
                wr_str = color+' '+ bitstream+'\n'
                f.write(wr_str)       # write the color and the code to a
file
        return

# Read an bmp image into a numpy array
img = imread('tiger.bmp')
#img = imresize(img,10)               # resize to 10% (not strictly
necessary - done for faster computation)

# convert to grayscale
gray_img = rgb2gray(img)

# compute histogram of pixels
hist = np.bincount(gray_img.ravel(),minlength=256)

probabilities = hist/np.sum(hist)             # a priori probabilities from
frequencies

root_node = tree(probabilities)                       # create the tree
using the probs.
tmp_array = np.ones([64],dtype=int)
huffman_traversal.output_bits = np.empty(256,dtype=int)
huffman_traversal.count = 0
f = open('codes.txt','w')
huffman_traversal(root_node,tmp_array,f)              # traverse the tree
and write the codes

input_bits = img.shape[0]*img.shape[1]*8      # calculate number of bits in
grayscale
compression = (1-np.sum(huffman_traversal.output_bits*hist)/input_bits)*100
print('Compression is ',compression,' percent')
```

Output

```
Compression is  9.664828266523184  percent
```