Bansilal Ramnath Agarwal Charitable Trust's

VISHWAKARMA INSTITUTE OF TECHNOLOGY

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

DEPARTMENT OF E&TC ENGINEERING

LABORATORY JOURNAL REPORT

A.Y.2019-20 (SEM-II)

Digital Image Processing

Batch: 2

Sampada Petkar (K 62, 1710536)

Priyanka Sargam (K 71, 1710433)

T.Y. B.Tech.

Guide: Prof. Ashutosh Marathe

INDEX

Note:

- All codes are done using Python version 3.6.0.
- Libraries OpenCV version 4.1.1.26, Matplotlib version 3.1.1 and Numpy 1.17.2 are used.
- During any practical, masks used are of 3 X 3 size. Masks are listed with array elements whenever they are used.

Experiment No. 1

Title: Up and down sampling of given image

Aim: To implement downsampling of the greyscale image

Code:

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

image=cv2.imread('dog.jfif', 0)
fig=plt.figure()

ht,width=image.shape
x=int(width/2)
y=int(ht/2)

newimg=np.zeros([ht,width,3],dtype=np.uint8)
newimg4=np.zeros([int(y/2),int(x/2),3],dtype=np.uint8)
i=0
while(i<ht-1):
   j=0
   k=0

   while j<width-1:
      newimg[i][k]=image[i][j]
      newimg[i+1][k]=image[i][j]
      newimg[i][k+1]=image[i][j]
      newimg[i+1][k+1]=image[i][j]
      k=k+2;
```

```
        j=j+2;

    i=i+2

ax1 = fig.add_subplot(2,2,2)
ax1.title.set_text("Downsampled by 2")
plt.imshow(newimg, cmap = "gist_gray")
i=0
while i<ht-3:
    j=0
    k=0

    while j<width-3:
        for x in range (0,3):
            for y in range (0,3):
                newimg[i+x][k+y]=image[i][j]
        k=k+4;
        j=j+4;
    i=i+4
ax2 = fig.add_subplot(2,2,3)
ax2.title.set_text("Downsampled by 4")
plt.imshow(newimg, cmap = "gist_gray")
i=0
while i<ht-7:
    j=0
    k=0
    while j<width-7:
        for x in range (0,7):
            for y in range (0,7):
                newimg[i+x][k+y]=image[i][j]
```

```
        k=k+8;
        j=j+8;
    i=i+8
ax3 = fig.add_subplot(2,2,4)
ax3.title.set_text("Downsampled by 8")
plt.imshow(newimg, cmap = "gist_gray")
ax4 = fig.add_subplot(2,2,1)
ax4.title.set_text("Input image")
plt.imshow(image, cmap = "gist_gray")

plt.show()
```

Results:

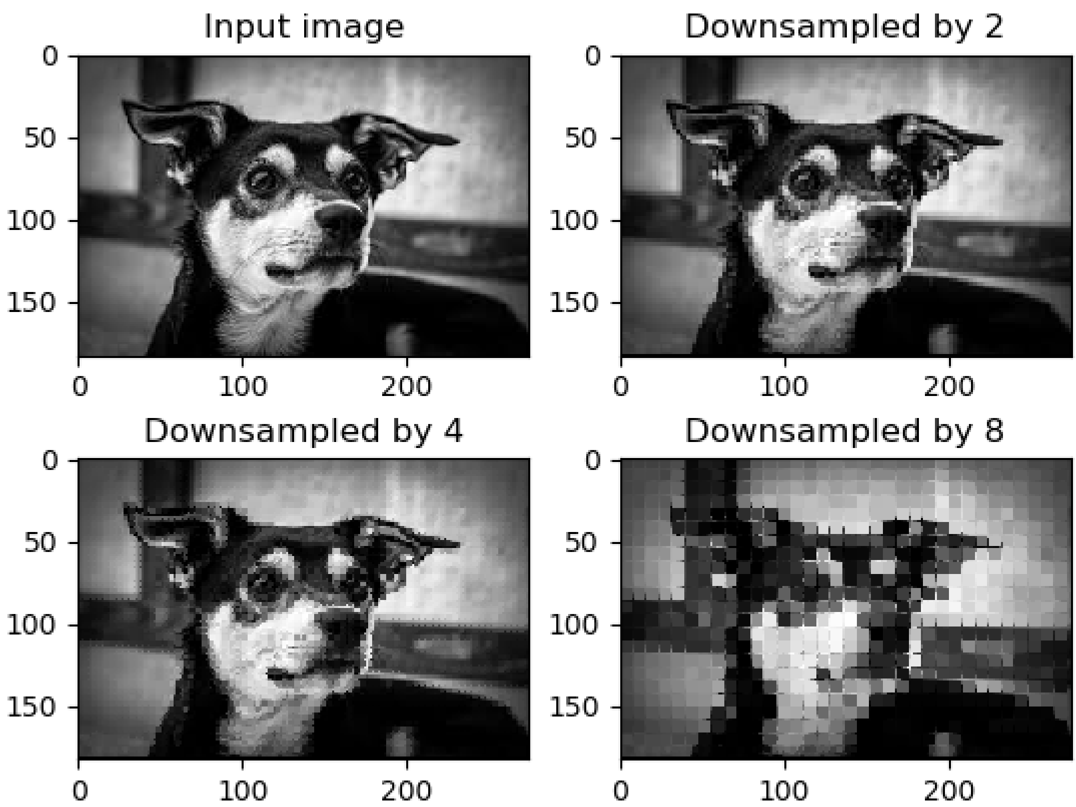


Fig. 1.1. Results

Experiment No. 2

Title: Quantization effects and false contouring

Aim : To implement false contouring on given image

Code:

```
import cv2

q=2
factor=2**(8-q)

image=cv2.imread('floww.jfif', 0)
image=cv2.resize(image,(500,300),interpolation=cv2.INTER_AREA)
cv2.imshow('Original',image)

#print(image)
def falseContour(image):
    ht,width=image.shape
    for i in range (0,ht):
        for j in range (0,width):
            image[i][j]=int(image[i][j]/factor)*factor

    cv2.imshow('Quantized',image)

falseContour(image)
```

Results:

Fig. 2.1. Input Image

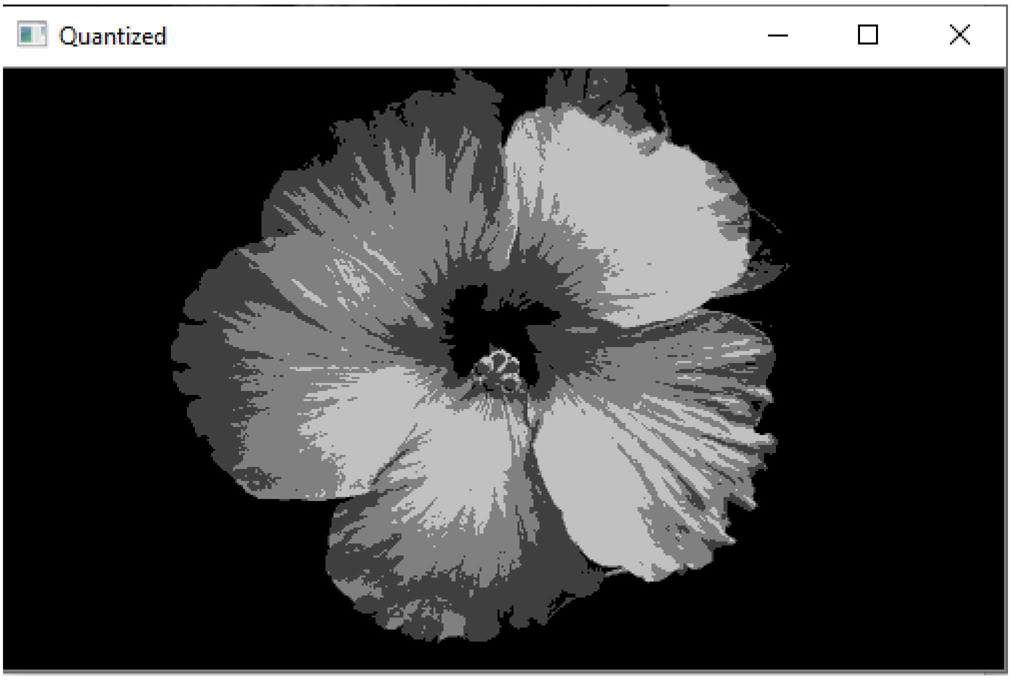


Fig. 2.2. Output Image

Experiment No. 3 A

Title: Image Enhancement

Aim: To implement

a) identity transformation

b) negative transformation,

c) power-log (gamma) transformation,

d) nth root transformation,

e) log transformation

f) inverse log transformation on input image.

A. Identity Transformation

Code:

```
import cv2
img=cv2.imread(r'D:\6th Semester\DIP\LAB\standard_test_images\lena_gray_512.tif',0)
cv2.imshow('Input',img)
img2=img
cv2.imshow('Output',img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results:

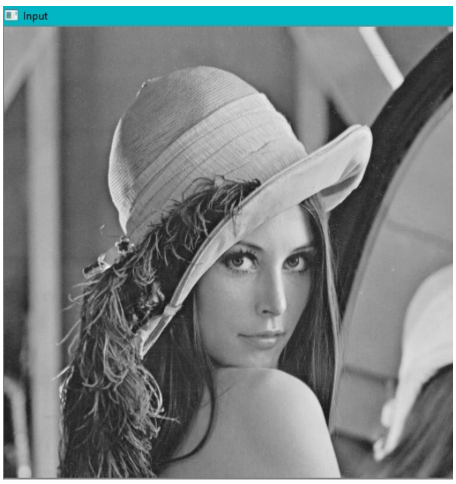


Fig. 3.A.1. Input image

Fig. 3.A.2. output image

B. Negetive Transformation

Code:

```
import cv2
import numpy as np
img_1 = cv2.imread(r'D:\6th Semester\DIP\LAB\standard_test_images\vessel.jpg')
img_2 = 255 - img_1
cv2.imshow('input_image',img_1)
cv2.imshow('image_negative',img_2)
cv2.waitKey(100000)
cv2.destroyAllWindows()
```

Results:

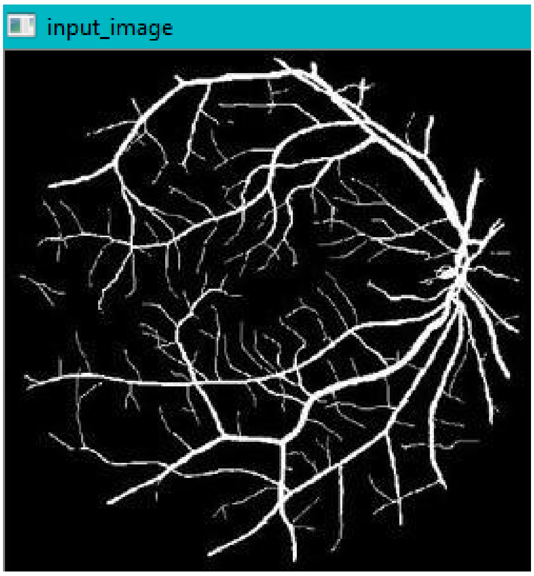


Fig. 3.A.3. Input image

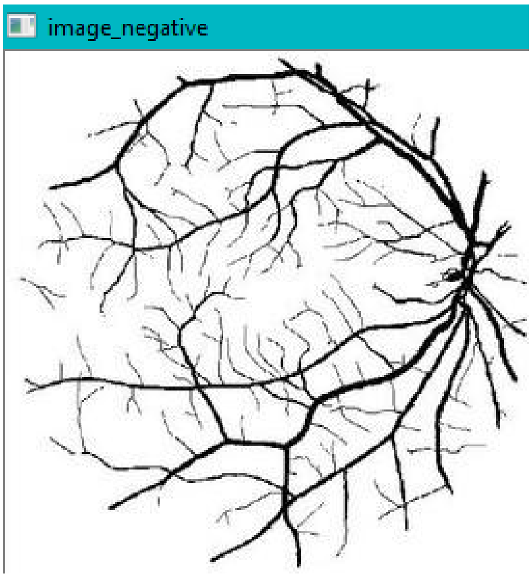


Fig. 3.A.4. output image

C. Power Log Transformation

Code:

```
import cv2
import numpy as np
# Open the image.
```

```
img = cv2.imread(r'D:\6th
Semester\DIP\LAB\standard_test_images\woman_darkhair.tif',1)
cv2.imshow('Input',img)
cv2.waitKey(10000)
gamma=2
img2 = np.power(img,gamma)
gamma=3
img3 = np.power(img,gamma)
gamma=4
img4 = np.power(img,gamma)
cv2.imshow('gamma2',img2)
cv2.waitKey(10000)
cv2.imshow('gamma3',img3)
cv2.waitKey(10000)
cv2.imshow('gamma4',img4)
cv2.waitKey(10000)
cv2.destroyAllWindows()
```

Results:

Fig. 3.A.5. Input image

Fig. 3.A.6. output image, $\gamma = 2$

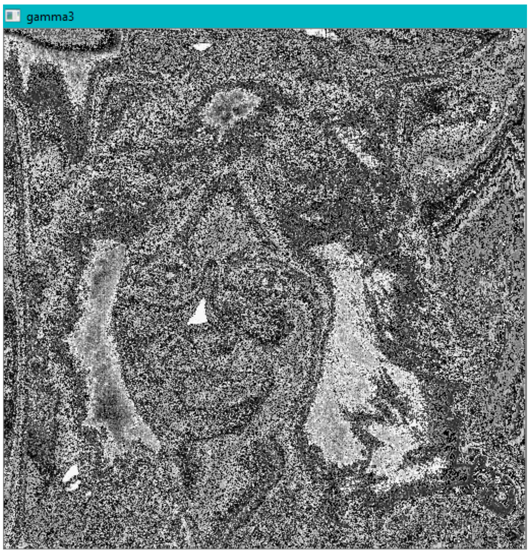


Fig. 3.A.7. output image, γ = 3

Fig. 3.A.8. output image, γ = 4

D. Nth Root Transformation

Code:

```
import cv2
import numpy as np
# Open the image.
img = cv2.imread(r'D:\6th Semester\DIP\LAB\standard_test_images\scientist.tif',0)
cv2.imshow('Input',img)
gamma=2
img2 = np.power(img,(1/gamma))
gamma=3
img3 = np.power(img,(1/gamma))
gamma=4
img4 = np.power(img,(1/gamma))
cv2.imshow('gamma2',img2)
cv2.imshow('gamma3',img3)
```

```
cv2.imshow('gamma4',img4)
cv2.waitKey(10000)
cv2.destroyAllWindows()
```

Results:

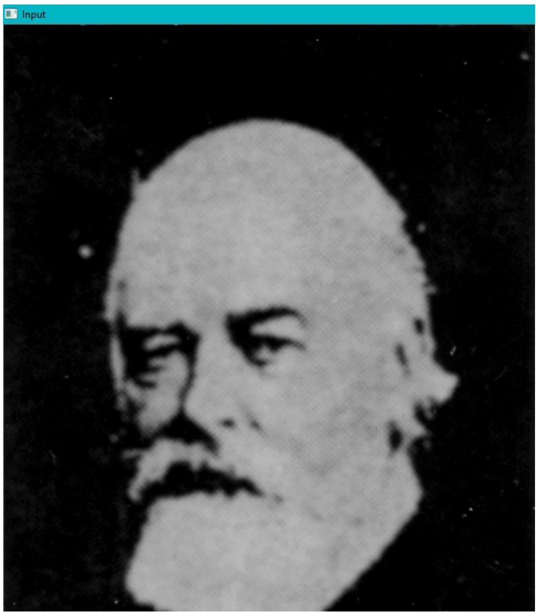


Fig. 3.A.10. Input image

Fig. 3.A.11. output image, γ = 2

Fig. 3.A.12. output image, γ = 3

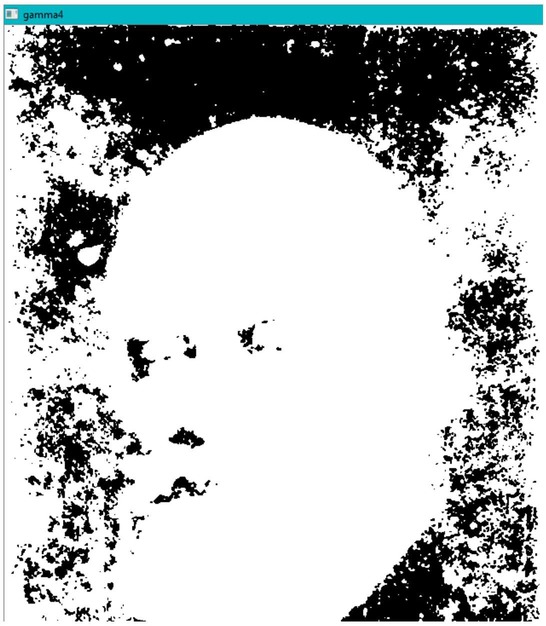


Fig. 3.A.13. output image, γ = 4

E. Log Transformation

Code:

```
import cv2
import numpy as np
img_1 = cv2.imread(r'D:\6th Semester\DIP\LAB\standard_test_images\scientist.tif',0)
c=2
img_2 = np.uint8(np.log1p(img_1))
thresh = 1
img_3 = cv2.threshold(img_2,thresh,255,cv2.THRESH_BINARY)[1]
cv2.imshow('input_image',img_1)
cv2.waitKey(10000)
cv2.imshow('log_transformation',img_3)
cv2.waitKey(10000)
cv2.destroyAllWindows()
```

Results:

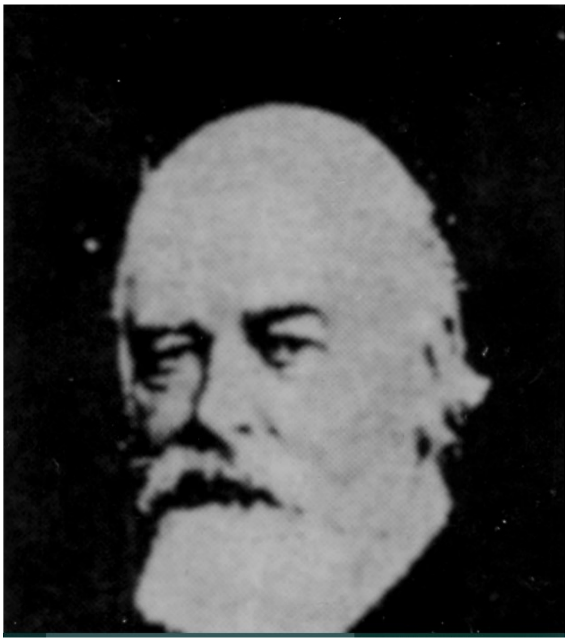


Fig. 3.A.14. Input image

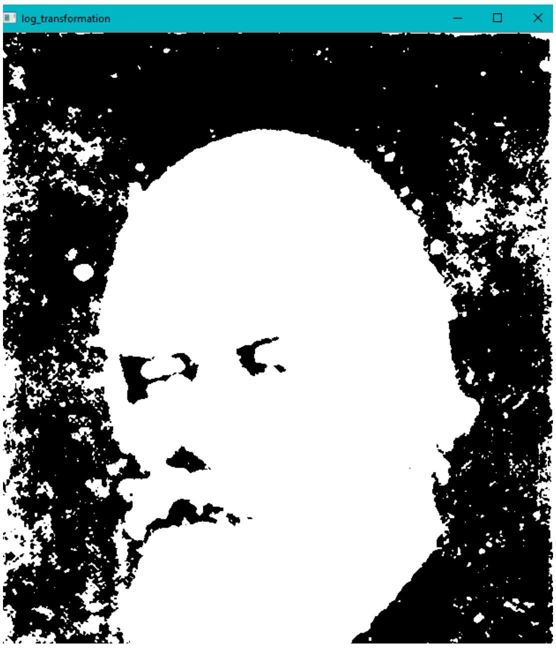


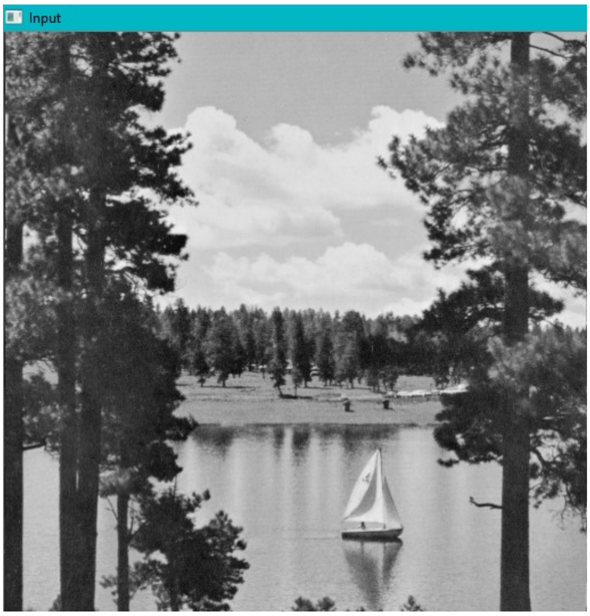
Fig. 3.A.15. output image

F. Inverse Log Transformation

Code:

```
import cv2
import numpy as np
# Open the image.
img = cv2.imread(r'D:\6th Semester\DIP\LAB\standard_test_images\lake.tif',1)
cv2.imshow('Input',img)
# Apply log transform.
c = 255 / (np.log(1 + np.max(img)))
ilog_transformed = np.exp(img/c)-1
# Specify the data type.
ilog_transformed = np.array(ilog_transformed, dtype=np.uint8)
cv2.imshow('ILog',ilog_transformed)
cv2.waitKey(100000)
cv2.destroyAllWindows()
```

Results:



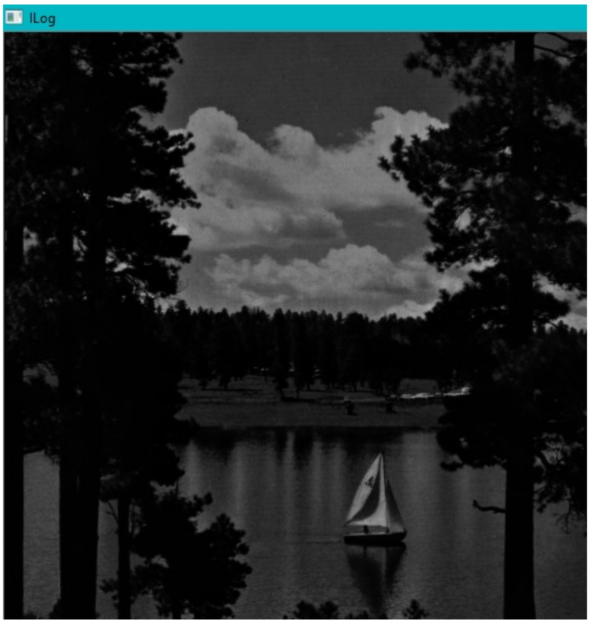


Fig. 3.A.16. Input image

Fig. 3.A.17. output image

Experiment No. 3 B

Title: Implementation of Histogram equalization

Aim: To implement histogram equalization on the given greyascale image.

Code:

```
# import Opencv
import cv2

# import Numpy
import numpy as np

# read a image using imread
img = cv2.imread(r'D:\6th
Semester\DIP\LAB\standard_test_images\peppers_gray.tif', 0)
cv2.imshow('Input', img)
# creating a Histograms Equalization
# of a image using cv2.equalizeHist()
equ = cv2.equalizeHist(img)

# stacking images side-by-side
res = np.hstack((img, equ))

# show image input vs output
cv2.imshow('Output', equ)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results:

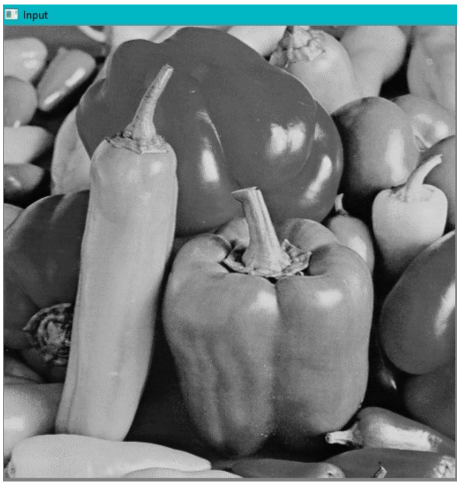


Fig. 3.B.1. Input image

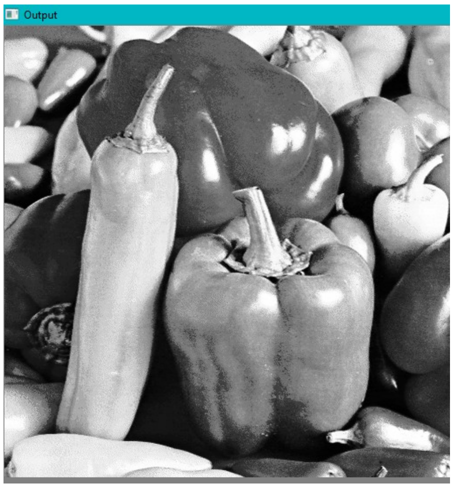


Fig. 3.B.2. Output image

Experiment No. 4

Title: Contrast Stretching

Aim: To implement contrast stretching for low contrast and high contrast images

Code:

```
import cv2
import numpy as np
img = cv2.imread('paw.jfif')

original = img.copy()
cv2.imshow("original_LH", original)

xp = [0, 64, 128, 192, 255]
fp = [0, 16, 128, 240, 255]
x = np.arange(256)
table = np.interp(x, xp, fp).astype('uint8')
img = cv2.LUT(img, table)
cv2.imshow("Output - LH", img)
img2 = cv2.imread('flower.jfif')
original2 = img2.copy()
cv2.imshow("original HL", original2)

fp2 = [0, 64, 128, 192, 255]
xp2 = [0, 20, 128, 240, 255]
x = np.arange(256)
table = np.interp(x, xp2, fp2).astype('uint8')
img2 = cv2.LUT(img2, table)
cv2.imshow("Output - HL", img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results:

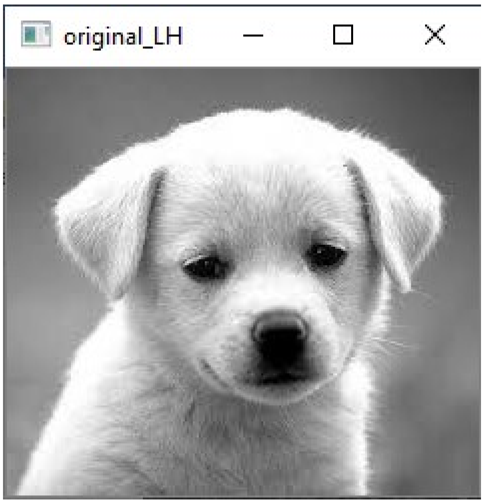


Fig. 4.1. Input image(low to high)

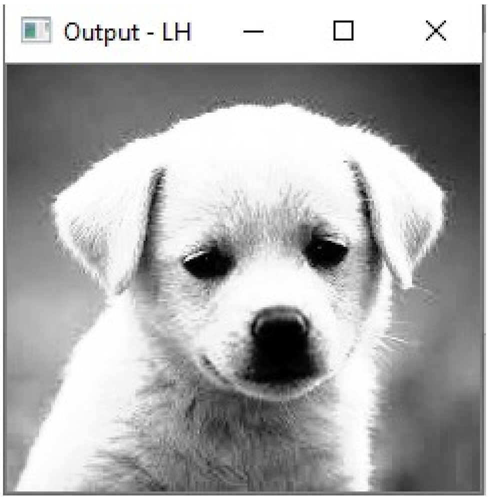


Fig. 4.2. output image(low to high)

Fig. 4.3. Input image (high to low)

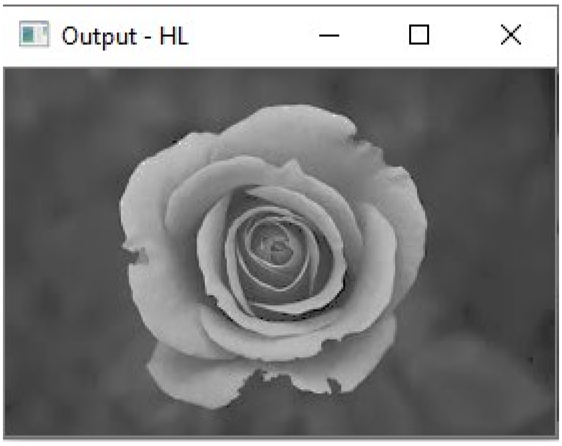


Fig. 4.3. output image(high to low)

Experiment No. 5

Title: Bit plane slicing

Aim: To implement the bit plane slicing on a greyscale image

Code:

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

img = cv2.imread('flower.jfif', 0)
fig=plt.figure(figsize=(8, 8))

for k in range(0, 8):
    plane = np.full((img.shape[0], img.shape[1]), 2 ** k, np.uint8)
    res = cv2.bitwise_and(plane, img)
    x = res * 255
    ax = fig.add_subplot(3,3,k+2)
    ax.title.set_text("Plane " + str(k+1))
    plt.imshow(x, cmap = "gist_gray")

ax1 = fig.add_subplot(3,3,1)
ax1.title.set_text("Input Image")
plt.imshow(img, cmap = "gist_gray")

plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

Results:

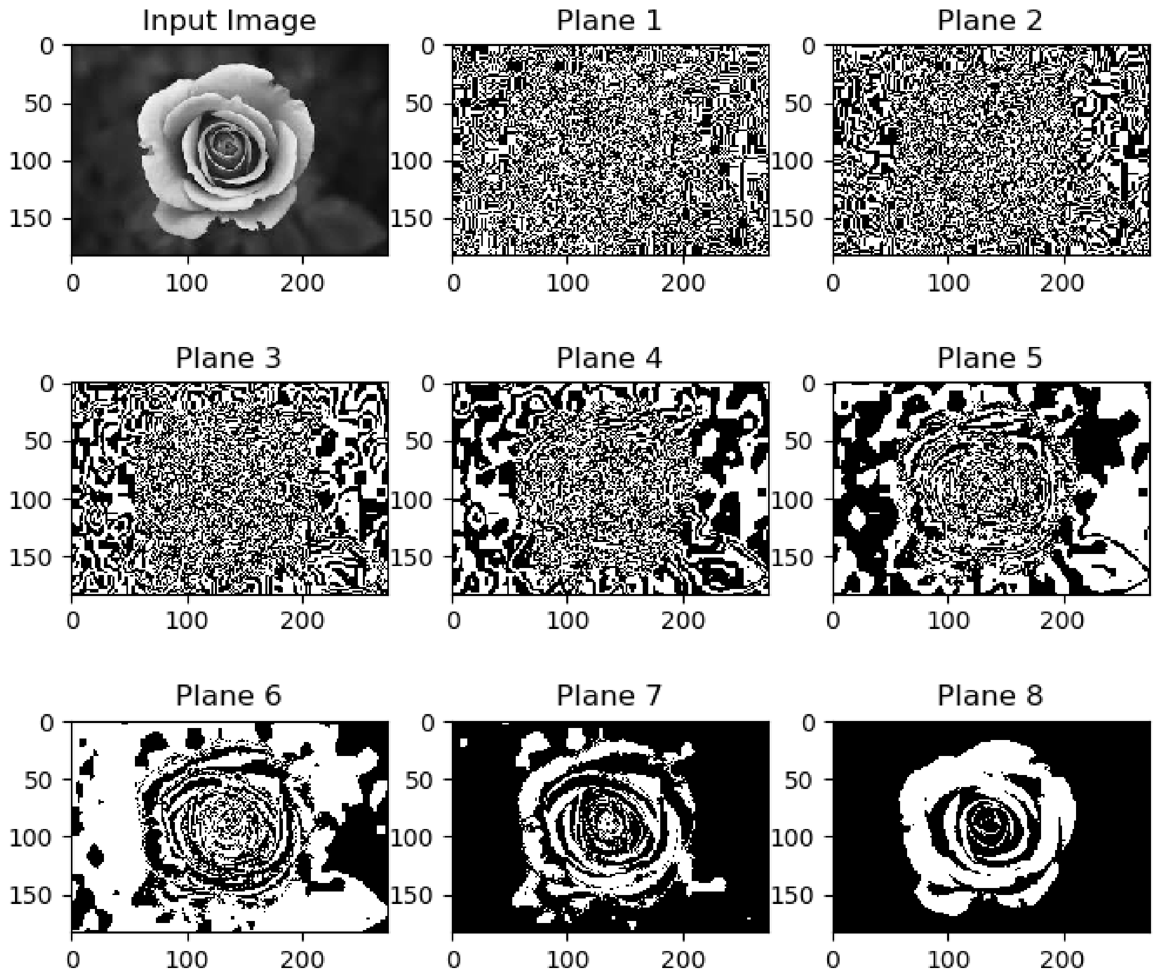


Fig. 5.1. Results

Experiment No. 6

Title: Masking Techniques

Aim: Implement weighted average mask and average mask for given image.

Code:

```
import cv2
import numpy as np

image = cv2.imread('flower.jfif', 0)
cv2. imshow('original', image)

ht,wd =image.shape
newimg = np.zeros([ht,wd,3],dtype=np.uint8)
newimg_w = np.zeros([ht,wd,3],dtype=np.uint8)

def average(image):
   for i in range (0, ht -1):
      for j in range (0, wd - 1):
         avg = int((int(image[i-1][j-1]) + int(image[i-1][j]) + int(image[i-1][j+1])
                 + int(image[i][j-1]) + int(image[i][j]) + int(image[i][j+1])
                 + int(image[i+1][j-1]) + int(image[i+1][j]) + int(image[i+1][j+1])) / 9)
         if(avg > 255):
            avg = 255
         newimg[i][j] =avg

   cv2. imshow('average', newimg)
def weighted(image):
   mask = [[3, 2, 3], [2, 1, 2], [3, 2, 3]]
   for i in range (0, ht -1):
      for j in range (0, wd - 1):
```

```
            w_avg = int((int(image[i-1][j-1])* mask[0][0] + int(image[i-1][j])* mask[0][1]
+ int(image[i-1][j+1])* mask[0][2]
                  + int(image[i][j-1])* mask[1][0] + int(image[i][j])* mask[1][1] +
int(image[i][j+1])* mask[1][2]
                  + int(image[i+1][j-1])* mask[2][0] + int(image[i+1][j])* mask[2][1] +
int(image[i+1][j+1]* mask[2][2])) / 9)
            if(w_avg > 255):
               w_avg = 255
            newimg_w[i][j] = w_avg
   cv2. imshow('weighted average', newimg_w)
average(image)
weighted(image)
```

Results:

Fig. 6.1. Input image

Fig. 6.2. Output image- Average mask

Fig. 6.3. Output image- weighted average mask

Experiment No. 7

Title: Laplacian mask

Aim : To implement laplacian mask

Code:

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

image=cv2.imread('dog.jfif')

image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow('original',image)
ht,width=image.shape
lp=[[0,-1,0],[-1,4,-1],[0,-1,0]]
k=np.zeros([ht,width],dtype=float)

temp=k[0][0]

for i in range (1,ht-1):
    for j in range (1,width-1):
        temp=0

        for x in range(0,3):
            for y in range(0,3):
                temp=temp+image[i-1+x][j-1+y]*lp[x][y]
                if(temp<0):
                    k[i][j]=0
```

```
            else:
                k[i][j]=temp;

x=np.amax(k)
print(x)

for i in range (0,ht):
    for j in range(0,width):
        image[i][j]=np.uint8(k[i][j]*255/x)

cv2.imshow('Laplacian',image)
```

Results:

Fig. 7.1. Input image

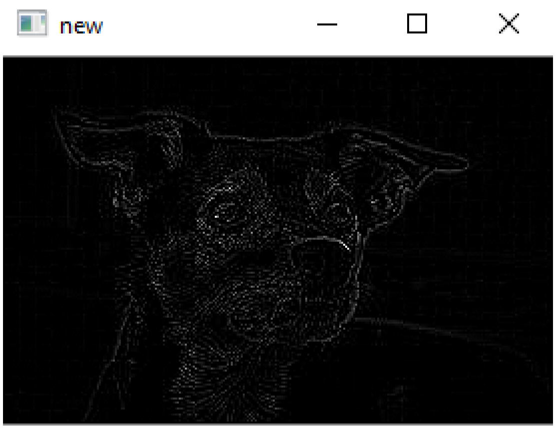


Fig. 7.2. Output image

Experiment No. 8

Title: Morphological Transformations – I on Binary images

Aim: To implement morphological transformations

a) Erosion

b) Dilation

c) Opening

d) Closing on binary images.

Code:

```
import cv2
import numpy as np
image = cv2.imread('sampada.png', 0)
cv2. imshow('original', image)
ht,wd =image.shape
mask = [[1, 1, 1], [1, 1, 1], [1, 1, 1]]
newimg_e = np.zeros([ht,wd,3],dtype=np.uint8)

for i in range (0, ht -1):
    for j in range (0, wd - 1):
        flag = ((mask[0][0] & image[i-1][j-1]) & (mask[0][1] & image[i-1][j]) & (mask[0][0] & image[i-1][j+1])
                & (mask[1][0] & image[i][j-1]) & (mask[1][1] & image[i][j]) & (mask[1][0] & image[i][j+1])
                & (mask[2][0] & image[i+1][j-1]) & (mask[2][1] & image[i+1][j]) & (mask[2][0] & image[i+1][j+1]))

        if flag == 1:
            newimg_e[i][j] = 255
        else:
            newimg_e[i][j] = 0
```

```
cv2.imshow('erosion', newimg_e)
flag = 0
mask_d = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
newimg_d = np.zeros([ht,wd,3],dtype=np.uint8)

for i in range (0, ht -1):
   for j in range (0, wd - 1):
      flag = ((mask_d[0][0] | image[i-1][j-1]) | (mask_d[0][1] | image[i-1][j]) | (mask_d[0][0] | image[i-1][j+1])
              | (mask_d[1][0] | image[i][j-1]) | (mask_d[1][1] | image[i][j]) | (mask_d[1][0] | image[i][j+1])
              | (mask_d[2][0] | image[i+1][j-1]) | (mask_d[2][1] | image[i+1][j]) | (mask_d[2][0] | image[i+1][j+1]))

      if flag != 0:
         newimg_d[i][j] = 255

      else:
         newimg_d[i][j] = 0

cv2.imshow('dilation', newimg_d)

flag = 0
newimg_o = np.zeros([ht,wd,3],dtype=np.uint8)

for i in range (0, ht -1):
   for j in range (0, wd - 1):
      flag = ((mask_d[0][0] | newimg_e[i-1][j-1][0]) | (mask_d[0][1] | newimg_e[i-1][j][0]) | (mask_d[0][0] | newimg_e[i-1][j+1][0])
              | (mask_d[1][0] | newimg_e[i][j-1][0]) | (mask_d[1][1] | newimg_e[i][j][0]) | (mask_d[1][0] | newimg_e[i][j+1][0])
```

```
                    | (mask_d[2][0] | newimg_e[i+1][j-1][0]) | (mask_d[2][1] |
newimg_e[i+1][j][0]) | (mask_d[2][0] | newimg_e[i+1][j+1][0]))

            if flag != 0:
                newimg_o[i][j] = 255

            else:
                newimg_o[i][j] = 0

    cv2.imshow('opening', newimg_o)

    newimg_c = np.zeros([ht,wd,3],dtype=np.uint8)

    for i in range (0, ht -1):
        for j in range (0, wd - 1):
            flag = ((mask[0][0] & newimg_d[i-1][j-1][0]) & (mask[0][1] & newimg_d[i-
1][j][0]) & (mask[0][0] & newimg_d[i-1][j+1][0])
                    & (mask[1][0] & newimg_d[i][j-1][0]) & (mask[1][1] & newimg_d[i][j][0])
& (mask[1][0] & newimg_d[i][j+1][0])
                    & (mask[2][0] & newimg_d[i+1][j-1][0]) & (mask[2][1] &
newimg_d[i+1][j][0]) & (mask[2][0] & newimg_d[i+1][j+1][0]))

            if flag == 1:
                newimg_c[i][j] = 255

            else:
                newimg_c[i][j] = 0

    cv2.imshow('closing', newimg_c)
```

Results:

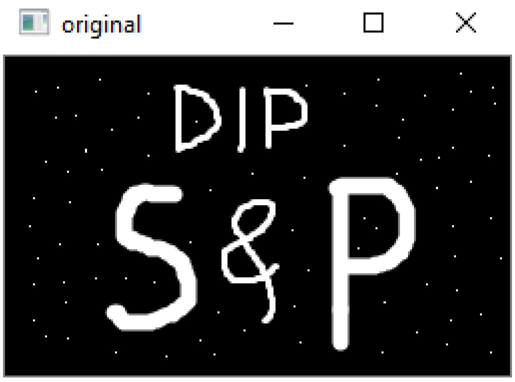


Fig. 8.1. Input image

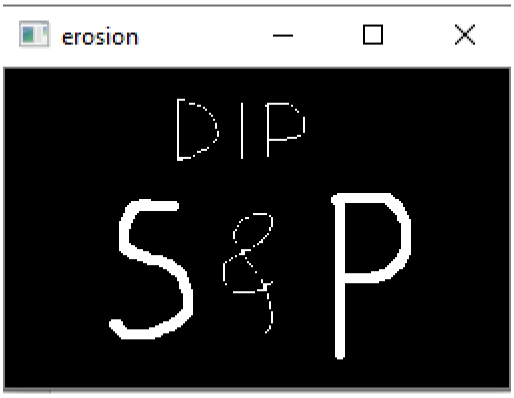


Fig. 8.2. Erosion

Fig. 8.3. Dilation

Fig. 8.4. Opening

Fig. 8.5. Closing

Experiment No. 9

Title: Morphological Transformations - II on Greyscale images

Aim: To implement morphological transformations

a) Erosion

b) Dilation

c) Opening

d) Closing on greyscale images.

Code:

```
import cv2
import numpy as np
image = cv2.imread('tulips.jfif', 0)
cv2.imshow('original',image)
ht,wd =image.shape

mask = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
newimg_e = np.zeros([ht,wd],dtype=np.uint8)

def gray_erosion(image_in):
    #flag=0;
    image=image_in.copy()
    for i in range (1, ht -1):
        for j in range (1, wd - 1):
            flag = [mask[0][0] + image[i-1][j-1],mask[0][1]+image[i-1][j],mask[0][2]+image[i-1][j+1],
                    mask[1][0] + image[i][j-1],mask[1][1] + image[i][j],mask[1][0] + image[i][j+1],
                    mask[2][0] + image[i+1][j-1],mask[2][1] + image[i+1][j],mask[2][0] + image[i+1][j+1]]
```

```
            newimg_e[i][j] = min(flag)


def gray_dilation(image_in):
    image=image_in.copy()
    for i in range (1, ht -1):
        for j in range (1, wd - 1):
            flag = [mask[0][0] + image[i-1][j-1],mask[0][1]+image[i-
1][j],mask[0][2]+image[i-1][j+1],
                    mask[1][0] + image[i][j-1],mask[1][1] + image[i][j],mask[1][0] +
image[i][j+1],
                    mask[2][0] + image[i+1][j-1],mask[2][1] + image[i+1][j],mask[2][0] +
image[i+1][j+1]]



            newimg_e[i][j] = max(flag)


gray_erosion(image)
cv2.imshow('Erosion',newimg_e)



gray_dilation(image)
cv2.imshow('Dilation',newimg_e)



gray_erosion(image)
gray_dilation(newimg_e)
cv2.imshow('Opening',newimg_e)


gray_dilation(newimg_e)
gray_erosion(image)
cv2.imshow('Closing',newimg_e)
```

Results:

Fig. 9.1. Input image

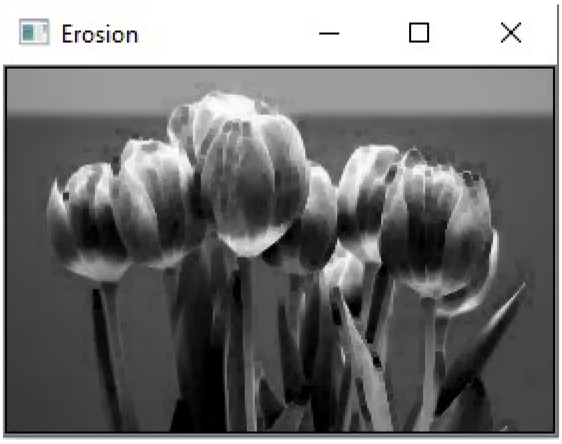


Fig. 9.2. Erosion

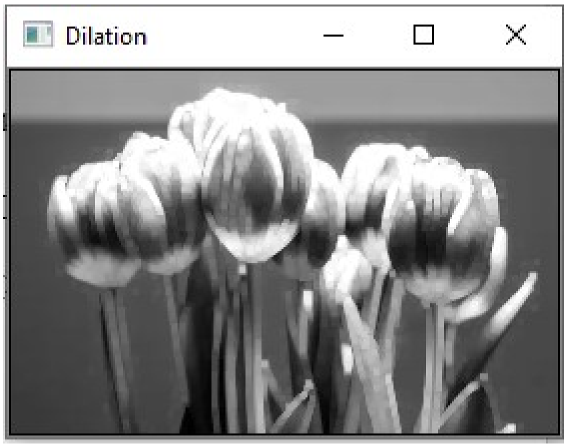


Fig. 9.3. Dilation

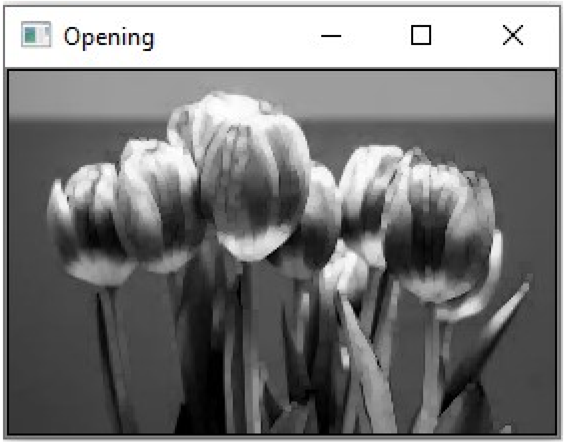


Fig. 9.4. Opening

Fig. 9.5. Closing

Experiment No. 10

Title: Marphological transformations – III on greyscale images

Aim: To implement Top hat and Well transformations on greyscale images

Code:

```
import cv2
import numpy as np

image = cv2.imread('tulips.jfif', 0)
cv2.imshow('original',image)
ht,wd =image.shape

mask = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

newimg_e = np.zeros([ht,wd],dtype=np.uint8)

def gray_erosion(image_in):
    #flag=0;
    image=image_in.copy()

    for i in range (1, ht -1):
        for j in range (1, wd - 1):
            flag = [mask[0][0] + image[i-1][j-1],mask[0][1]+image[i-
1][j],mask[0][2]+image[i-1][j+1],
                    mask[1][0] + image[i][j-1],mask[1][1] + image[i][j],mask[1][0] +
image[i][j+1],
                    mask[2][0] + image[i+1][j-1],mask[2][1] + image[i+1][j],mask[2][0] +
image[i+1][j+1]]
```

```
            newimg_e[i][j] = min(flag)



def gray_dilation(image_in):
    #flag=0;
    image=image_in.copy()
    for i in range (1, ht -1):
        for j in range (1, wd - 1):
            flag = [mask[0][0] + image[i-1][j-1],mask[0][1]+image[i-
1][j],mask[0][2]+image[i-1][j+1],
                    mask[1][0] + image[i][j-1],mask[1][1] + image[i][j],mask[1][0] +
image[i][j+1],
                    mask[2][0] + image[i+1][j-1],mask[2][1] + image[i+1][j],mask[2][0] +
image[i+1][j+1]]

            newimg_e[i][j] = max(flag)



def subtract(image1,image2,result):
    for i in range (0,ht-1):
        for j in range (0,wd-1):
            result[i][j]= int(image1[i][j])- int(image2[i][j])
            if(result[i][j] <0):
                result[i][j] = 0



gray_erosion(image)
gray_dilation(newimg_e)



newimg_tp = np.zeros([ht,wd],dtype=np.uint8)
```

```
subtract(image,newimg_e,newimg_tp)
cv2.imshow('Top Hat',newimg_tp)

gray_dilation(image)
gray_erosion(newimg_e)

newimg_w = np.zeros([ht,wd],dtype=np.uint8)
subtract(image,newimg_e,newimg_w)
cv2.imshow('Well',newimg_w)
```

Results:

Fig. 10.1. Input image

Fig. 10.4. Top Hat transformation

Fig. 10.5. Well Transformation