Name: Shreya Rane Roll No: 71

Experiment 1: Image Enhancement techniques

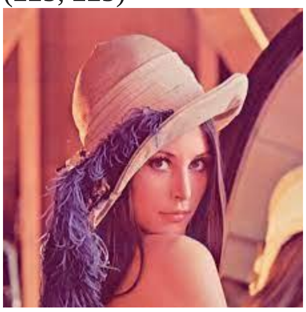
```
# Program to mount the drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Program to read and display color and gray scale image
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
im=cv2.imread('/content/Lena.jfif')
#im=cv2.imread('/content/drive/MyDrive/im_files/rosepic12.jpg')
#new_image = np.copy(im)
gray_img = cv2.cvtColor(im, cv2.COLOR_RGB2GRAY)
print(im.shape)
cv2_imshow(im)
print(im.shape)
cv2_imshow(gray_img)
print(gray_img.shape)

im1=np.copy(gray_img)
```

(225, 225)



(225, 225, 3)

```
# Program to display image negative of given image
im2=255-im1
cv2_imshow(im2)
```

```
#Program for gray level slicing with background
im11=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)
m,n = im1.shape
min=132
max=180
for i in range(m):
    for j in range(n):
        if im1[i,j] >min and im1[i,j]<max:
            im11[i,j]= 255
else:
        im11[i,j] = im1[i,j]
        cv2_imshow(im11)
```

```
#Program for gray level slicing without background
im11=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)
m,n = im1.shape
min=132
max=180
for i in range(m):
    for j in range(n):
        if im1[i,j] >min and im1[i,j]<max:
            im11[i,j]= 0
else:
  im11[i,j] = im1[i,j]
cv2_imshow(im11)
```

```
#Program for image Tresholding
T=100
for i in range(m):
    for j in range(n):
      if im1[i,j] <  T:
        im11[i,j]= 0
else:
    im11[i,j] = 255
    cv2_imshow(im11)
```

```
#Program for power law transformation(gamma Correction)
im12=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)
im22=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)
#m,n = im1.shape
gamma1=1.2
im12=np.power(im1,gamma1)
#cv2_imshow(im1)
gamma2=0.8
im22=np.power(im1,gamma2)
#cv2_imshow(im2)
hor_stack=np.row_stack((im12,im22))
cv2_imshow(hor_stack)
```

```
#Program for Log transformation
import math
import numpy as np
import cv2
L=255
d=np.zeros((225,225),np.uint8)
l1= math.log(L,10)
print(l1)
c=L/l1
print(c)
new=im1+1;
new1=np.log10(new)
d=c*new1
cv2_imshow(d)
```

2.4065401804339546
105.96124763394461

```
img = np.copy(im1)
lst = []
m,n = im1.shape
for i in range(m):
    for j in range(n):
         lst.append(np.binary_repr(img[i][j] ,width=8)) # width =
no. of bits

# We have a list of strings where each string represents binary
pixel value.
#To extract bit planes we need to iterate over the strings and
#store the characters corresponding to bit planes into lists.
# Multiply with 2^(n-1) and reshape to reconstruct the bit image.
eight_bit_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8)
* 128).reshape(img.shape[0],img.shape[1])
seven_bit_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8)
* 64).reshape(img.shape[0],img.shape[1])
six_bit_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) *
32).reshape(img.shape[0],img.shape[1])
five_bit_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8)
* 16).reshape(img.shape[0],img.shape[1])
four_bit_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8)
* 8).reshape(img.shape[0],img.shape[1])
three_bit_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8)
* 4).reshape(img.shape[0],img.shape[1])
```

```
two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) *
2).reshape(img.shape[0],img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) *
1).reshape(img.shape[0],img.shape[1])

#Concatenate these images for ease of display using cv2.hconcat()
finalr =
cv2.hconcat([eight_bit_img,seven_bit_img,six_bit_img,five_bit_img])
finalv
=cv2.hconcat([four_bit_img,three_bit_img,two_bit_img,one_bit_img])

# Vertically concatenate
final = cv2.vconcat([finalr,finalv])

# Display the images
cv2_imshow(final)
```

```
img = np.copy(im1)
lst = []
m,n = im1.shape
for i in range(m):
    for j in range(n):
         lst.append(np.binary_repr(img[i][j] ,width=8)) # width =
no. of bits
```

```
# We have a list of strings where each string represents binary
pixel value.
#To extract bit planes we need to iterate over the strings and
#store the characters corresponding to bit planes into lists.
# Multiply with 2^(n-1) and reshape to reconstruct the bit image.
eight_bit_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8)
* 128).reshape(img.shape[0],img.shape[1])
seven_bit_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8)
* 64).reshape(img.shape[0],img.shape[1])
six_bit_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) *
32).reshape(img.shape[0],img.shape[1])
five_bit_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8)
* 16).reshape(img.shape[0],img.shape[1])
four_bit_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8)
* 8).reshape(img.shape[0],img.shape[1])
three_bit_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8)
* 4).reshape(img.shape[0],img.shape[1])
two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) *
2).reshape(img.shape[0],img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) *
1).reshape(img.shape[0],img.shape[1])

#Concatenate these images for ease of display using cv2.hconcat()
finalr =
cv2.hconcat([eight_bit_img,seven_bit_img,six_bit_img,five_bit_img])
finalv
=cv2.hconcat([four_bit_img,three_bit_img,two_bit_img,one_bit_img])

# Vertically concatenate
final = cv2.vconcat([finalr,finalv])

# Display the images
cv2_imshow(final)
```