EXPERIMENT NO. 2

AIM: To perform point operations on an image

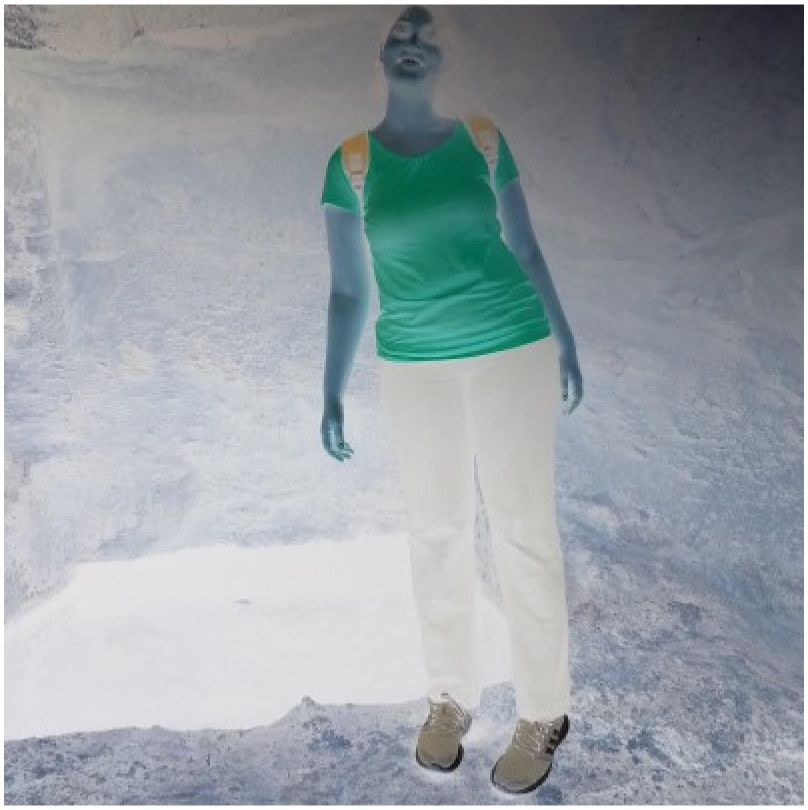
IMPLEMENTATION:

Image Specifications:
Taken on OnePlus 5 device
Image format is .jpg
Image dimensions are 400x400

```
# Importing libraries
from numpy import *
from cv2 import *
from google.colab.patches import *
from warnings import *
filterwarnings('ignore')
# Reading image and storing it in memory
image = imread('/content/drive/My Drive/IP/Prac 2/rebecca.jpg')
cv2_imshow(image)
```

DIGITAL NEGATIVE

```
# Subtract the img from max value(calculated from dtype)
neg_img = 255 - image
cv2_imshow(neg_img)
```



ANALYSIS

Negative of the image has been performed by doing 255 - the pixel bits value. This directly affects the RGB values of the image.

THRESHOLDING

```
# Convert image to grayscale
img = cvtColor(image, COLOR_BGR2GRAY)

# Pixel intensity is greater than threshold value then value is truncated to
threshold. Other values remain the same
ret, thresh1 = threshold(img, 120, 255, THRESH_TRUNC)
# Pixel intensity is greater than threshold value then value is set to zero
else value is set to 255
ret, thresh2 = threshold(img, 120, 255, THRESH_BINARY_INV)
```

```
# Pixel intensity is greater than threshold value then value is set to 255 else
value is set to zero
ret, thresh3 = threshold(img, 120, 255, THRESH_BINARY)

cv2_imshow(thresh1)
cv2_imshow(thresh2)
cv2_imshow(thresh3)
```

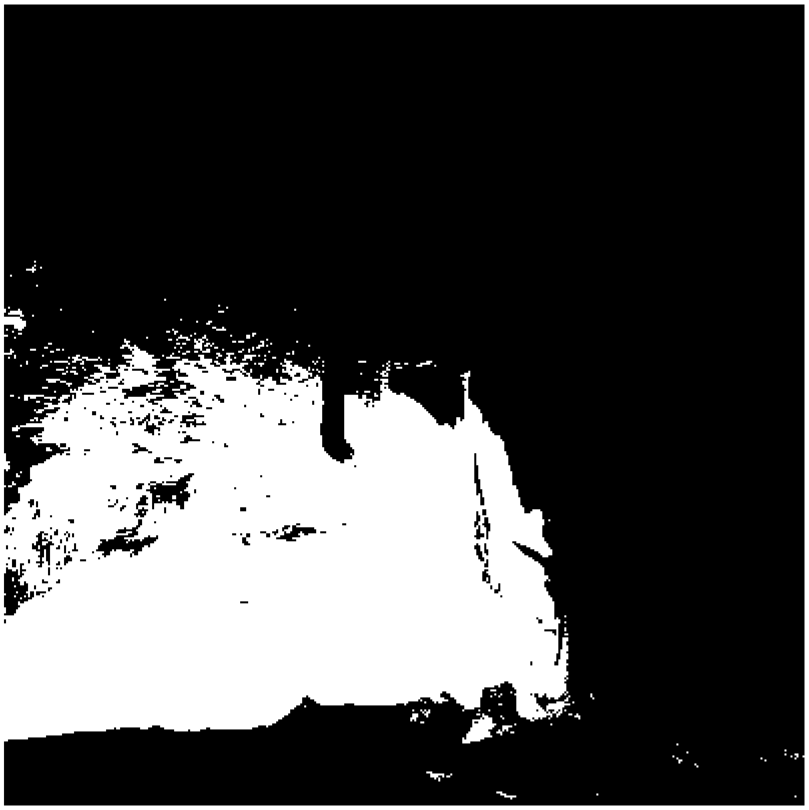
ANALYSIS

Thresholding has been used to divide the pixels into two categories - black or white. This has been achieved by specifying the threshold value.

GRAY LEVEL SLICING

```
# Convert image to grayscale
img = cvtColor(image, COLOR_BGR2GRAY)

row, column = img.shape
# Create an zeros array to store the sliced image
gray_slice_img = zeros((row,column),dtype = 'uint8')
# Specify the min and max range
min_range = 10
max_range = 60
# Loop over the input image and if pixel value lies in desired range set it to
255 otherwise set it to 0.
for i in range(row):
    for j in range(column):
        if img[i,j]>min_range and img[i,j]<max_range:
            gray_slice_img[i,j] = 255
        else:
            gray_slice_img[i,j] = 0
# Display the image
cv2_imshow(gray_slice_img)
```



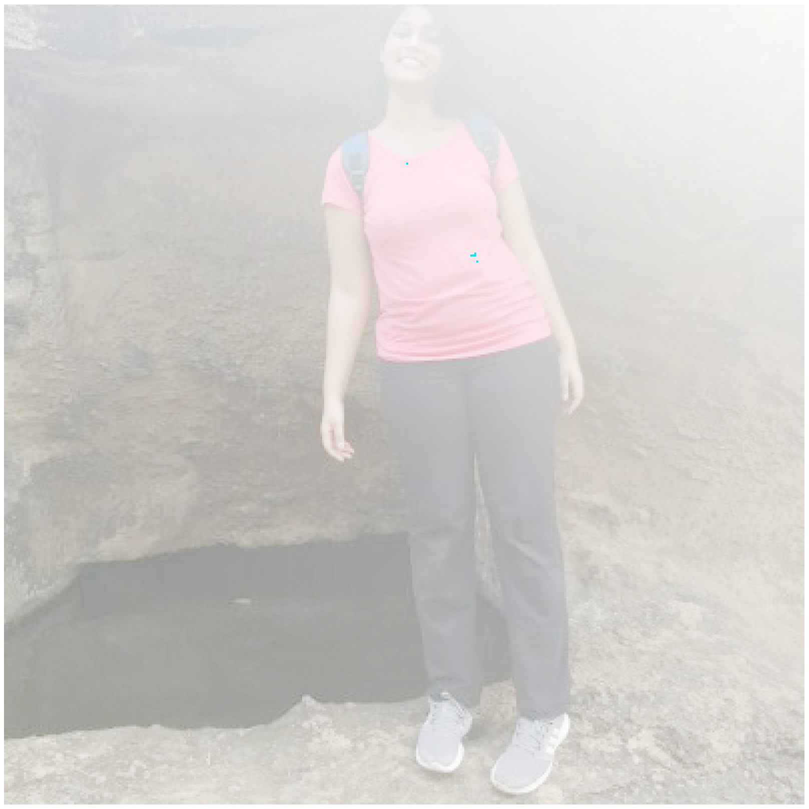
ANALYSIS

Highlighting a specific features, eg masses of water in satellite imagery. This specific feature has been specified by the range of the pixel values.

DYNAMIC RANGE COMPRESSION: LOG TRANSFORMATION

```
from numpy import max, log

img = image
# Apply log transform
log_img = (log(img+1)/(log(1+max(img))))*255
# Specify the data type
log_img = array(log_img,dtype=uint8)
cv2_imshow(log_img)
```

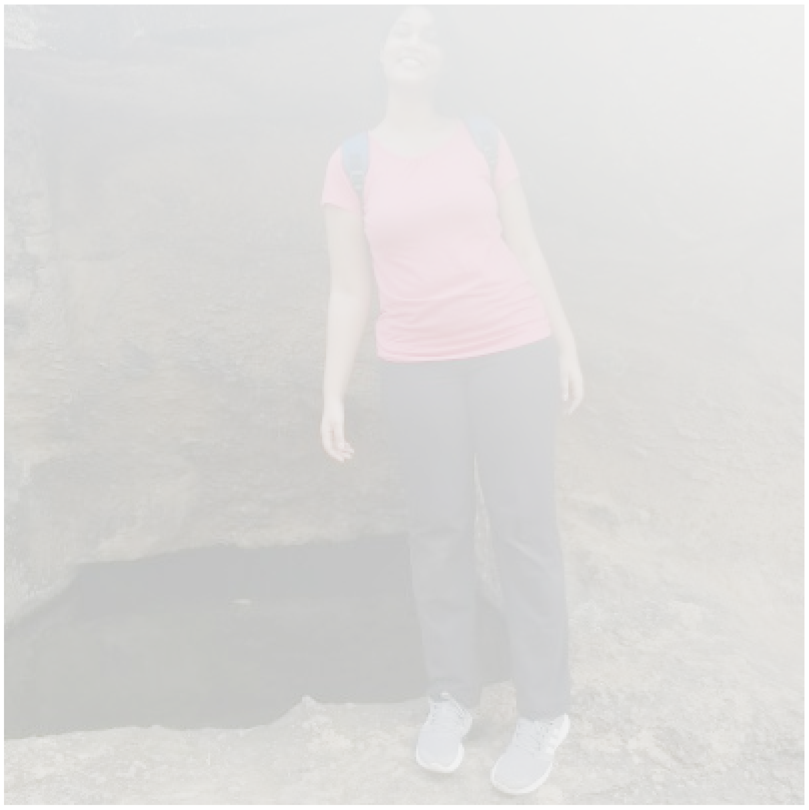


ANALYSIS

Expand the values of dark pixels. Maps a narrow range of low gray level values in the input image into a wider range of output levels.

POWER LAW TRANSFORMATION: GAMMA CORRECTION

```
for gamma in [0.1, 0.5, 1.2, 2.2]:
  # Apply Gamma valye on the normalised image and then multiply by scaling
constant (For 8 bit, c=255)
  gamma_corrected = array(244*(image/255) ** gamma, dtype = 'uint8')
  print("Gamma = ", gamma)
  cv2_imshow(gamma_corrected)
```



ANALYSIS

Gamma correction has been performed for gamma values [0.1, 0.5, 1.2, 2.2]. Used to display images on screen in MRI scans.

BIT PLANE SLICING

```
# Convert image to grayscale
img = cvtColor(image, COLOR_BGR2GRAY)

lst = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
         lst.append(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 = (array([int(i[0]) for i in lst],dtype = uint8) *
128).reshape(img.shape[0],img.shape[1])
seven_bit_img = (array([int(i[1]) for i in lst],dtype = uint8) *
64).reshape(img.shape[0],img.shape[1])
six_bit_img = (array([int(i[2]) for i in lst],dtype = uint8) *
32).reshape(img.shape[0],img.shape[1])
```

```
five_bit_img = (array([int(i[3]) for i in lst],dtype = uint8) *
16).reshape(img.shape[0],img.shape[1])
four_bit_img = (array([int(i[4]) for i in lst],dtype = uint8) *
8).reshape(img.shape[0],img.shape[1])
three_bit_img = (array([int(i[5]) for i in lst],dtype = uint8) *
4).reshape(img.shape[0],img.shape[1])
two_bit_img = (array([int(i[6]) for i in lst],dtype = uint8) *
2).reshape(img.shape[0],img.shape[1])
one_bit_img = (array([int(i[7]) for i in lst],dtype = uint8) *
1).reshape(img.shape[0],img.shape[1])

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

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

# Display the images
cv2_imshow(final)
```

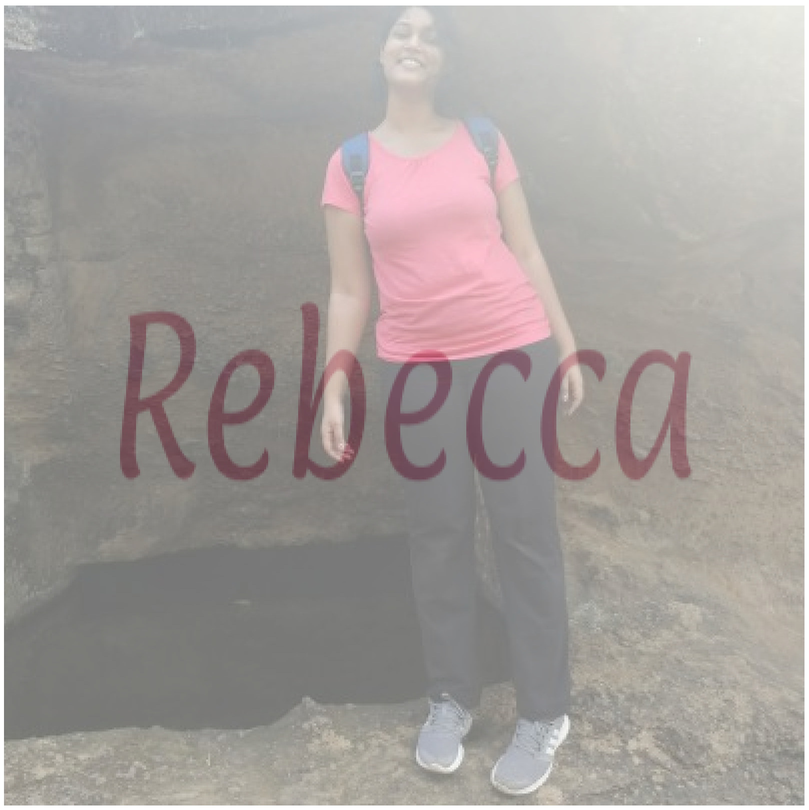
ANALYSIS

Bit plane slicing has been performed on each bit.

WATERMARKING

```
img = image
watermark = imread('/content/drive/My Drive/IP/Prac 2/watermark.png')
cv2_imshow(watermark)
```

```
resized_watermark = resize(watermark,(400,400))
watermarked_img = (0.6* img) +  (0.4 * resized_watermark)
cv2_imshow(watermarked_img)
```



ANALYSIS

Watermarking has been performed by reducing the intensity of both images, and then merging them together.

CONCLUSION

Thus the point operations have been performed on the image.