Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)
Academic Year 2022-2023

Name : Sarvagya Singh

SAPID : 60009200030 BATCH : K1

Experiment - 1

AIM: To Perform Basic Image Processing Operations in Python

PROBLEM STATEMENT:

Read an Image

Display an Image

Convert in Grey Scale

Crop an Image

Arithmetic Operations

Logical Operations

THEORY:

Discuss about Different Image Processing Libraries available in Python.

Features & Limitations of each.

Libraries used:

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

Features:

• OpenCV is an open-source library, so it is free to use and can be easily customized to meet specific needs.

• OpenCV provides a comprehensive set of algorithms for image processing and computer vision, including object detection, recognition, tracking, and segmentation.

• Python is a popular programming language for scientific computing, and OpenCV's Python interface makes it easy to integrate computer vision algorithms with other Python libraries like NumPy, SciPy, and Matplotlib.

• OpenCV's Python bindings support both Python 2 and Python 3, making it accessible to a wide range of users.

• OpenCV is cross-platform, so code written in Python can be easily ported to other platforms like Linux, Windows, and macOS.

• OpenCV has a large and active community, so there are plenty of resources available for learning and troubleshooting, including documentation, tutorials, and forums.

Limitations:

• Speed: OpenCV in Python can be slower compared to other languages like C++ because of the GIL (Global Interpreter Lock) in Python. This can limit the performance of real-time applications.

• Memory management: Memory management in Python can be problematic, especially when working with large datasets. This can lead to slower processing times and memory leaks.

• Limited machine learning capabilities: Although OpenCV has some machine learning capabilities, it may not be as robust as other popular machine learning libraries like TensorFlow and PyTorch.

• Limited support for deep learning: OpenCV has limited support for deep learning, which may be a disadvantage for complex image processing tasks.

• Limited support for GPU processing: OpenCV in Python may not provide support for GPU processing, which may limit the speed of certain applications.

• Limited documentation: OpenCV documentation in Python may not be as extensive as in other languages, which may make it challenging for beginners to learn and use the library effectively.

RESULT:

Sarvagya Singh

600009200030 -- K1

IPCV -- lab1

In [ ]:
```
import cv2 as cv
import PIL
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

In [ ]:
```
%%capture
!wget https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTG8Oggi82WkRRVnsxMlr_WqdNwhVzo9vvW_42SZm8&s
```

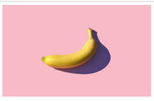
In [ ]:
```
# %%capture
# !wget
```

In [ ]:
```
# Displaying the image
path1 = "/content/image.jpg"
image_np = cv.imread(path1)
print(image_np.shape)
image_np[1][:5]
```

```
(122, 123, 3)
```

Out[ ]:
```
array([[136, 126, 139],
       [142, 132, 148],
       [116, 107, 127],
       [ 54,  49,  70],
       [ 83,  80, 105]], dtype=uint8)
```

In [ ]:
```
path2 = "/content/banana.jpg"
im_np2 = cv.imread(path)
im_np2.shape,image_np.shape
cv2_imshow(im_np2)
```



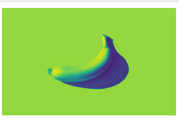
In [ ]:
```
# Displaying the image
im_rgb = cv.cvtColor(image_np, cv.COLOR_BGR2RGB)
plt.axis('off')
plt.imshow(im_rgb);
```

In [ ]:
```
cv2_imshow(image_np)
```

In [ ]:
```
im_gray1 = cv.cvtColor(im_rgb, cv.COLOR_BGR2GRAY)
im_gray2 = cv.cvtColor(im_np2, cv.COLOR_BGR2GRAY)
plt.axis('off')
plt.imshow(im_gray1);
```

In [ ]:
```
cv2_imshow(im_gray1)
```

In [ ]:
```
plt.axis('off')
plt.imshow(im_gray2);
```



In [ ]:
```
cv2_imshow(im_gray2)
```

In [ ]:
```
from PIL import Image
im1 = Image.open(path1)
# im1.show()
# cv2_imshow(im1)
print(im1.filename)
print(im1.size)
print(im1.info)
print(im1.palette)
```

```
/content/image.jpg
(123, 122)
{'jfif': 257, 'jfif_version': (1, 1), 'jfif_unit': 0, 'jfif_density': (1, 1)}
```

In [ ]:
```
#  Cropping an image
cropped_image = im_gray1[10:110, 20:110]
cv2_imshow(cropped_image)
```

In [ ]:
```
cropped_image = im_gray2[10:110, 20:110]
cv2_imshow(cropped_image)
```

In [ ]:
```
# Arithmetic operation
im_1 = im_gray1^2
cv2_imshow(im_1)
```

In [ ]:
```
im_2 = im_gray1+502
cv2_imshow(im_2)
```

In [ ]:
```
im_2 = im_gray1-50
cv2_imshow(im_2)
```

In [ ]:
```
im_2 = im_gray1+50
cv2_imshow(im_2)
```

In [ ]:
```
im_2 = im_gray1/2
cv2_imshow(im_2)
```

In [ ]:
```
resized_im1 = cv.resize(im_gray1,(122,122))
resized_im2 = cv.resize(im_gray2,(122,122))
resized_im2.shape,resized_im1.shape
```

Out[ ]:
```
((122, 122), (122, 122))
```

In [ ]:
```
bitwise_and_np = cv.bitwise_and(resized_im1, resized_im2)
cv2_imshow(bitwise_and_np)
```

In [ ]:
```
bitwise_xor_np = cv.bitwise_xor(resized_im1, resized_im2)
cv2_imshow(bitwise_xor_np)
```

In [ ]:
```
bitwise_not1 = cv.bitwise_not(resized_im1)
cv2_imshow(bitwise_not1)
```

In [ ]:
```
bitwise_not2 = cv.bitwise_not(resized_im2)
cv2_imshow(bitwise_not2)
```

In [ ]:
```

```

Name : Sarvagya Singh

SAPID : 60009200030 **BATCH: K1**

AIM: To Perform Image Enhancement(SD) Point Processing Techniques: Digital Negative, Thresholding, Grey Level Slicing with & without background

THEORY:

The principle objective of image enhancement is to process an image so that the result is more suitable than the original image for a specific application. Image enhancement approaches fall into two broad categories:

Spatial domain methods - which involve direct manipulation of pixels in an image

Frequency domain methods - which are based on modifying the Fourier transform of an image.

Point processing techniques are among the simplest of all image enhancement techniques. These techniques are basically zero memory operations where , every pixel is operated individually without the use of neighbouring pixels.

The values of pixels, before and after processing, will be called r and s, respectively. These values are related by an expression of the form

$$s = T(r)$$

where T is a transform function, it maps the pixel value r in to a pixel values.

1. Image Negatives

The negative of an image with gray levels in the range {0, L-1} is obtained by using the negative transformation given by the expression,

$s = (L – 1) – r$

where,

s = output gray level.

r = input gray level.

L – 1 = Highest gray level present in the image.

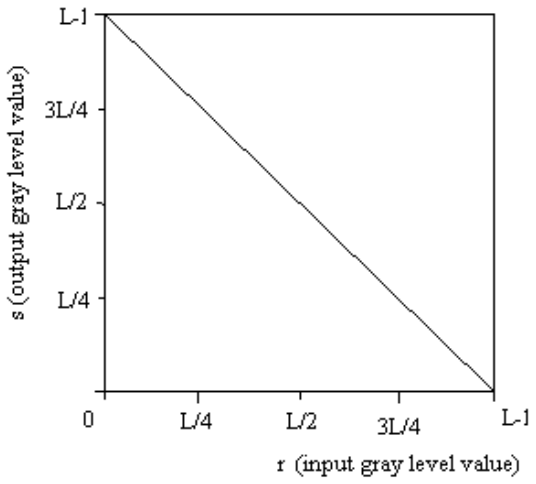


Fig 1. Negative Image Transformation

This type of processing is particularly suited for enhancing white or gray detail embedded in the dark regions of an image, especially when the black areas are dominant in size.

2.Thresholding

The input to a thresholding operation is typically a grayscale image. In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or *vice versa*).

For example, a simple mapping function is defined by the thresholding function. In contrast stretching if a = b , v = 0 & w = L - 1, the transformation function is such that ,

s = 0 for r < a

L – 1 for r >= a

The figure below shows the transformation function:

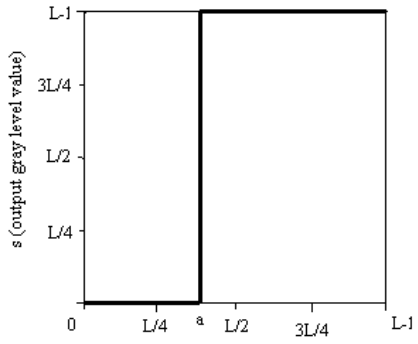


Fig 3. Thresholding

In simple implementations, the segmentation is determined by a single parameter known as the *intensity threshold*. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black.

3. Gray Level Slicing

Highlighting a specific range of gray levels in an image is often desired. Gray level slicing has two basic approaches for enhancement:

One is to display a high value for all gray levels in the range of interest and a low value for all other gray levels. This transformation produces a binary image.

Another approach brightens the desired range of gray levels but preserves the background and gray level tonalities in the image.

gray level slicing without background		gray level slicing with background	
s = L-1	a<= r < b	s = L-1	a< r <=b
s = 0	otherwise	s = r	otherwise

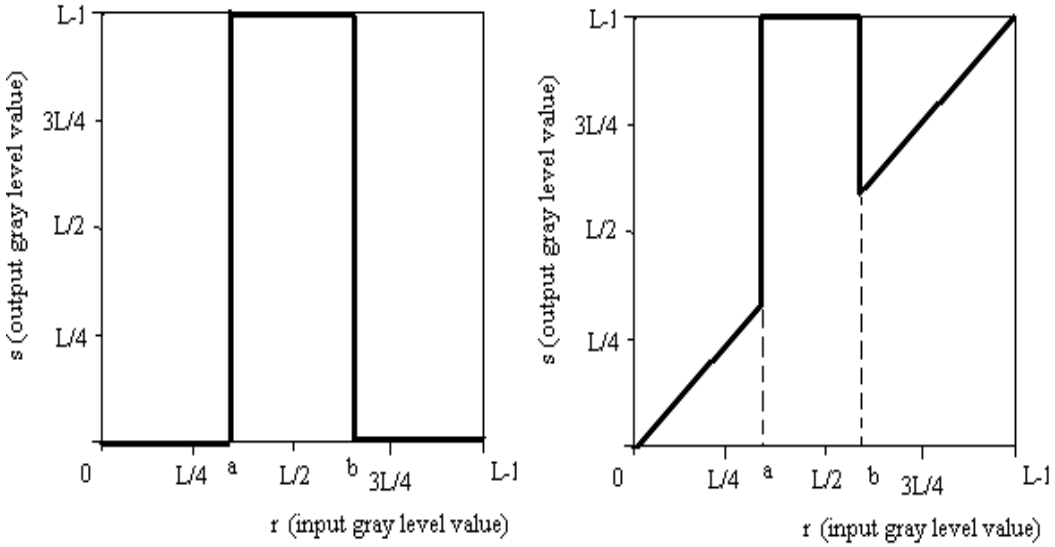


Fig 4. The gray level slicing operation (left), and with background operation (right).

RESULT:

Digital Negative

Sarvagya Singh

60009200030 - K1

IPCV -- lab2

In [1]:
```
!pip install opencv-python
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: opencv-python in /usr/local/lib/python3.8/dist-packages (4.6.0.66)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.8/dist-packages (from opencv-python) (1.22.4)
```

In [2]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

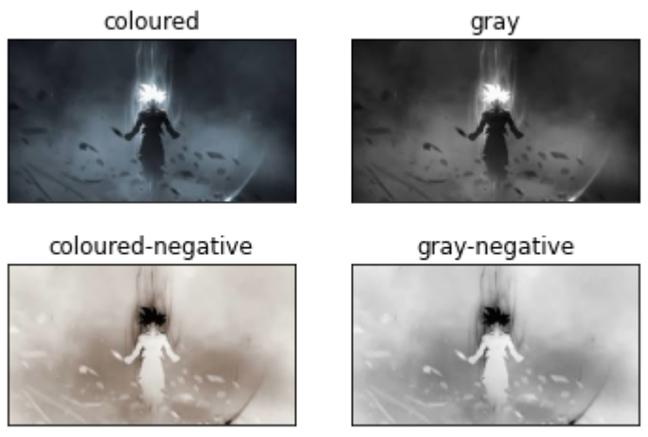
In [3]:
```
link = '/content/goku.jpeg'
image = cv2.imread(link)
```

In [4]:
```
img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

In [5]:
```
gray = cv2.imread(link, 0)
```

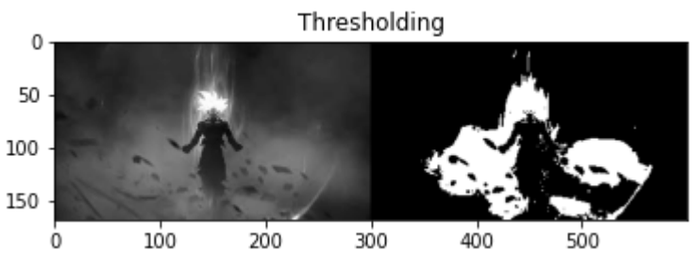
In [6]:
```
colored_negative = abs(255-img)
gray_negative = abs(255-gray)
```

In [7]:
```
imgs = [img, gray, colored_negative, gray_negative]
title = ['coloured', 'gray', 'coloured-negative', 'gray-negative']


plt.subplot(2, 2, 1)
plt.title(title[0])
plt.imshow(imgs[0])
plt.xticks([])
plt.yticks([])

plt.subplot(2, 2, 2)
plt.title(title[1])
plt.imshow(imgs[1], cmap='gray')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 2, 3)
plt.title(title[2])
plt.imshow(imgs[2])
plt.xticks([])
plt.yticks([])

plt.subplot(2, 2, 4)
plt.title(title[3])
plt.imshow(imgs[3], cmap='gray')
plt.xticks([])
plt.yticks([])

plt.show()
```



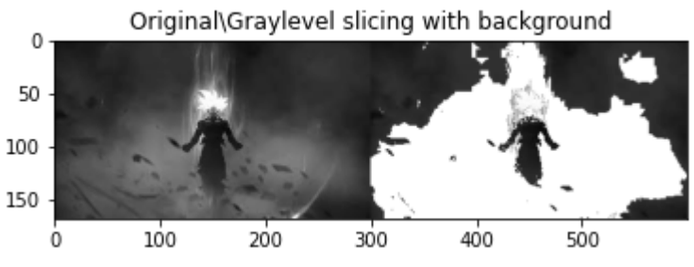
Thresholding

In [11]:
```
image = cv2.imread(link, 0)
x,y=image.shape
threshold = 100
z=np.zeros((x,y))
for i in range(0,x):
    for j in range(0,y):
        if(image[i][j]>threshold):
            z[i][j]=255
        else:
            z[i][j]=0
equ=np.hstack((image,z))
plt.title('Thresholding')
plt.imshow(equ,'gray')
plt.show()
```



Gray Level Slicing

In [12]:
```
import numpy as np
image=cv2.imread(link,0)
x,y=image.shape
z=np.zeros((x,y))
for i in range(0,x):
    for j in range(0,y):
        if(image[i][j]>50 and image[i][j]<150):
            z[i][j]=255
        else:
            z[i][j]=image[i][j]
equ=np.hstack((image,z))
plt.title('Original\Graylevel slicing with background')
plt.imshow(equ,'gray')
plt.show()
```



In [13]:
```
image=cv2.imread(link,0)
x,y=image.shape
z=np.zeros((x,y))
for i in range(0,x):
    for j in range(0,y):
        if(image[i][j]>50 and image[i][j]<150):
            z[i][j]=255
        else:
            z[i][j]=0
equ=np.hstack((image,z))
plt.title('Original\Graylevel slicing without background')
plt.imshow(equ,'gray')
plt.show()
```

Name: Sarvayga Singh

SAPID : 60009200030 **BATCH : K1**

AIM: To Perform Image Enhancement(SD) Point Processing Techniques: Contrast Stretching, Log Transformation, Power Law Transformation

THEORY:

1. Contrast Stretching

Low - contrast images can result from poor illumination, lack of dynamic range in the imaging sensor or wrong setting of a lens aperture during image acquisition. Contrast stretching is employed to increase the dynamic range of the gray levels in the image being processed.

The figure illustrates the form of the transformation function for contrast stretching:

The locations of points (r_1, s_1) and (r_2, s_2) control the shape of the transformation and thus the contrast of the output image.

If $r_1 = s_1$ and $r_2 = s_2$, the transformation is a linear function that produces no changes in the gray levels.

If $r_1 = r_2$, $s_1 = 0$ and $s_2 = L-1$, the transformation becomes a thresholding function that creates a binary image.

Intermediate values of (r_1, s_1) and (r_2, s_2) produce various degrees of spread in the gray levels of the output image, thus affecting its contrast.

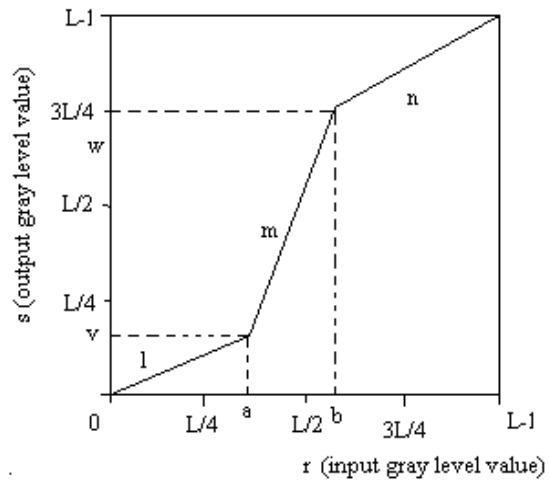


Fig 2. Contrast Stretching

In general, $r_1 \leq r_2$ and $s_1 \leq s_2$ is assumed to make the function single valued and monotonically increasing which preserves the order of gray levels thus preventing the creation of intensity artefacts in the processed image.

2. Dynamic Range Compression (Log Transformation)

The log transformations can be defined by this formula

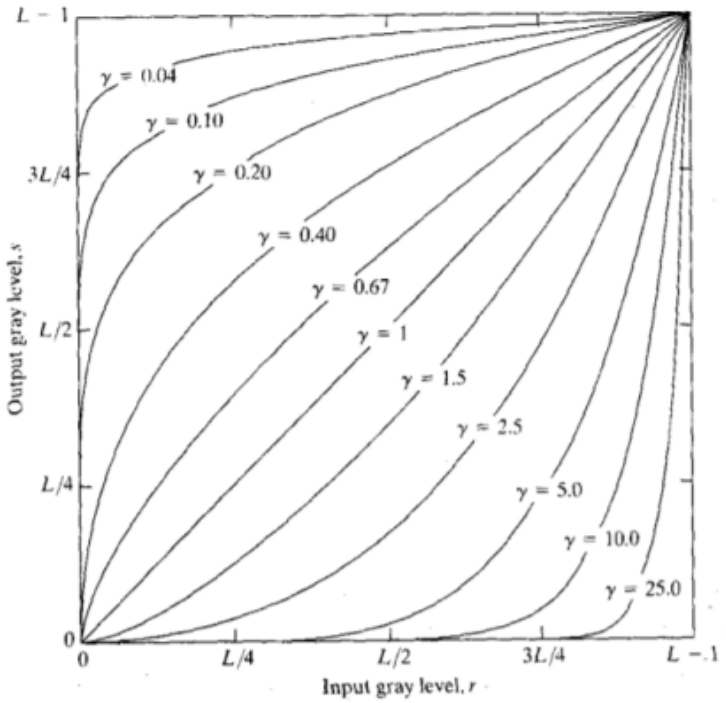
$$\mathbf{s = c \log(r + 1)}$$

Where s and r are the pixel values of the output and the input image and c is a constant. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation.

3. Power Law Transformation:

$s=r^{\gamma}$



RESULT:

Sarvagya Singh
60009200030 - K1
IPCV -- lab2

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

Contrast Stretching

In [ ]:
```
# Function to map each intensity level to output intensity level.
def pixelVal(pix, a, b, l, m, n):
  v = l*a
  w = m*(b-a) + v
  if (0 <= pix and pix <= a):
      return (v / a)*pix
  elif (a < pix and pix <= b):
      return ((w - v)/(b - a)) * (pix - a) + v
  else:
      return ((255 - w)/(255 - b)) * (pix - b) + w
```

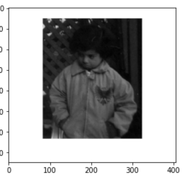
In the range

- l, n < 1
- m >= 1

In [ ]:
```
img = cv2.imread('/content/StandardDeviationBasedImageStretchingExample_01.png', 0)

a = 60
b = 140
l_1 = 0.1
m_1 = 1.5
n_1 = 0.6
```

In [ ]:
```
plt.imshow(img, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc5403c50a0>
```



In [ ]:
```
img.min()
```

Out[ ]:
```
74
```

In [ ]:
```
img.max()
```

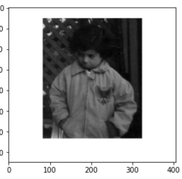
Out[ ]:
```
255
```

In [ ]:
```
pixelVal_vec = np.vectorize(pixelVal)
```

In [ ]:
```
contrast_stretched_1 = pixelVal_vec(img, a, b, l_1, m_1, n_1)
```

In [ ]:
```
plt.imshow(contrast_stretched_1, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc541d04580>
```



In [ ]:
```
contrast_stretched_1.min()
```

Out[ ]:
```
27.0
```

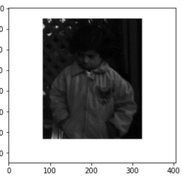
Outside the range

- l, n > 1
- m < 1

In [ ]:
```
a = 60
b = 140
l_2 = 1.5
m_2 = 0.5
n_2 = 2
```

In [ ]:
```
contrast_stretched_2 = pixelVal_vec(img, a, b, l_2, m_2, n_2)
plt.imshow(contrast_stretched_2, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc5403465e0>
```



In [ ]:
```
contrast_stretched_2.min()
```

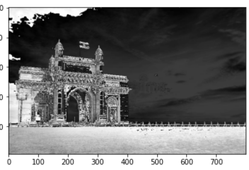
Out[ ]:
```
97.0
```

Log Transformation

In [ ]:
```
c = 150
log_transformed = c*np.log(img + 1)

log_transformed = np.array(log_transformed, dtype = np.uint8)

plt.imshow(log_transformed, cmap='gray')
```

```
<ipython-input-25-012fe7bb06f9>:2: RuntimeWarning: divide by zero encountered in log
  log_transformed = c*np.log(img + 1)
```

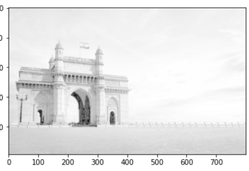
Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc542100610>
```



Power Law Transformation

In [ ]:
```
gamma_1 = 0.2
gamma_2 = 2.5

power_transformed_1 = img**gamma_1
plt.imshow(power_transformed_1, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc5421b3820>
```



In [ ]:
```
power_transformed_2 = img**gamma_2
plt.imshow(power_transformed_2, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc542192d30>
```

In [ ]:
```
```

Sarvagya Singh
60009200030 - K1
IPCV -- lab2

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

Contrast Stretching

In [ ]:
```
# Function to map each intensity level to output intensity level.
def pixelVal(pix, a, b, l, m, n):
  v = l*a
  w = m*(b-a) + v
  if (0 <= pix and pix <= a):
      return (v / a)*pix
  elif (a < pix and pix <= b):
      return ((w - v)/(b - a)) * (pix - a) + v
  else:
      return ((255 - w)/(255 - b)) * (pix - b) + w
```

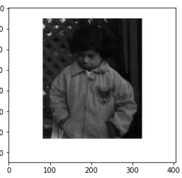
In the range

- l, n < 1
- m >= 1

In [ ]:
```
img = cv2.imread('/content/StandardDeviationBasedImageStretchingExample_01.png', 0)

a = 60
b = 140
l_1 = 0.1
m_1 = 1.5
n_1 = 0.6
```

In [ ]:
```
plt.imshow(img, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc5403c50a0>
```



In [ ]:
```
img.min()
```

Out[ ]:
```
74
```

In [ ]:
```
img.max()
```

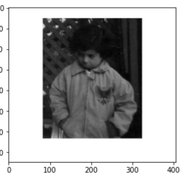
Out[ ]:
```
255
```

In [ ]:
```
pixelVal_vec = np.vectorize(pixelVal)
```

In [ ]:
```
contrast_stretched_1 = pixelVal_vec(img, a, b, l_1, m_1, n_1)
```

In [ ]:
```
plt.imshow(contrast_stretched_1, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc541d04580>
```



In [ ]:
```
contrast_stretched_1.min()
```

Out[ ]:
```
27.0
```

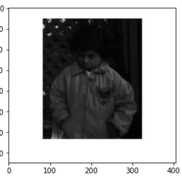
Outside the range

- l, n > 1
- m < 1

In [ ]:
```
a = 60
b = 140
l_2 = 1.5
m_2 = 0.5
n_2 = 2
```

In [ ]:
```
contrast_stretched_2 = pixelVal_vec(img, a, b, l_2, m_2, n_2)
plt.imshow(contrast_stretched_2, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc5403465e0>
```



In [ ]:
```
contrast_stretched_2.min()
```

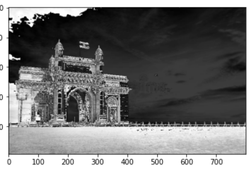
Out[ ]:
```
97.0
```

Log Transformation

In [ ]:
```
c = 150
log_transformed = c*np.log(img + 1)

log_transformed = np.array(log_transformed, dtype = np.uint8)

plt.imshow(log_transformed, cmap='gray')
```

```
<ipython-input-25-012fe7bb06f9>:2: RuntimeWarning: divide by zero encountered in log
  log_transformed = c*np.log(img + 1)
```

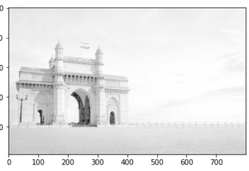
Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc542100610>
```



Power Law Transformation

In [ ]:
```
gamma_1 = 0.2
gamma_2 = 2.5

power_transformed_1 = img**gamma_1
plt.imshow(power_transformed_1, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc5421b3820>
```



In [ ]:
```
power_transformed_2 = img**gamma_2
plt.imshow(power_transformed_2, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fc542192d30>
```

In [ ]:
```
```

Name : Sarvagya Singh **LAB4** **SAPID : 60009200030(K1)**

AIM: To perform Image Enhancement(SD) Neighbourhood Processing Techniques: Smoothing Operators

PROBLEM STATEMENT:

Add Gaussian Noise, Remove using Averaging Filter

Add Salt & Pepper Noise, Remove using Median Filter

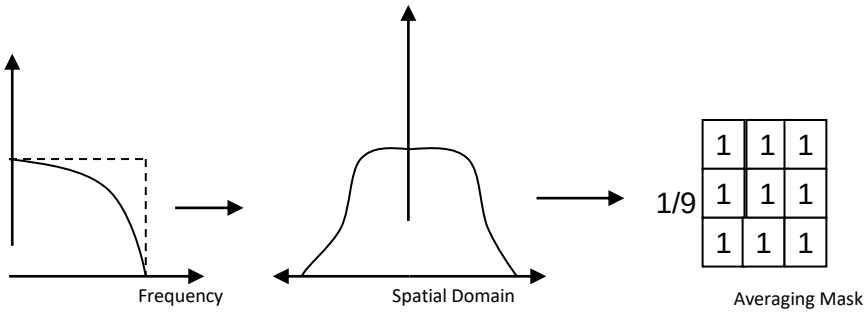
THEORY:

Smoothing

Low pass filtering removes the high frequency content from the image. It is used to remove the noise present in the image which is generally a high frequency signal.

1. **Averaging Filter**

If an image has Gaussian noise, a low pass averaging filter is used to remove the noise. The frequency response and spatial response is show below.



From spatial response we generate the mask that would give us the low pass filtering operation. Here all the coefficients are positive and the multiplying factor for a M X N matrix is 1 / (M X N).

2. **Median Filtering**

The averaging filter removes the noise by filtering it till it is no longer seen. But in the process it also blurs the edges. If e use averaging filter to remove salt and pepper noise from an image it will blur the noise but will also ruin the edges. Hence when we need to remove salt and pepper noise we use non-linear filter known as Median Filter. They are also called order statistical filters because their response is based on the ordering and ranking of the pixels contained within the mask.

Steps to perform median filtering:

1. Assume 3 X 3 empty mask.
2. Place it on let hand side corner.
3. Arrange the 9 pixels in ascending or descending order
4. Choose the median from these nine values.
5. Place this median at the centre.
6. Move the mask in similar fashion as the averaging mask

In median filtering the grey level of the centre pixel is replaced by the median filter value of the neighbourhood.

RESULT:

The average filters seems to blur the image and reduce the overall quality of the image. Whereas the gaussian filters sharpness the image too much thereby making it tough to be visible and get any information. Thou by combining both of the filters the overall quality of the image increases and it also creates a good amount of depth in the image.The median filter seems to be just as like as the average i.e. it reduces the quality of the image.

Sarvagya Singh

60009200030 -- K1

IPCV - LAB4

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

In [ ]:
```
img = cv2.imread('/content/gateway.jpeg', 0)
cv2_imshow(img)
```

In [ ]:
```
img.shape
```

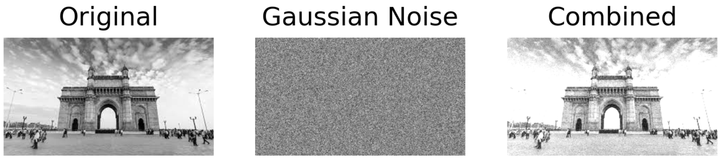
Out[ ]:
```
(168, 300)
```

In [ ]:
```
gauss_noise=np.zeros((168,300),dtype=np.uint8)
cv2.randn(gauss_noise,128,20)
gauss_noise=(gauss_noise*0.5).astype(np.uint8)
```

In [ ]:
```
gn_img=cv2.add(img, gauss_noise)
```

In [ ]:
```
fig=plt.figure(dpi=300)

fig.add_subplot(1,3,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

fig.add_subplot(1,3,2)
plt.imshow(gauss_noise,cmap='gray')
plt.axis("off")
plt.title("Gaussian Noise")

fig.add_subplot(1,3,3)
plt.imshow(gn_img,cmap='gray')
plt.axis("off")
plt.title("Combined")
```

Out[ ]:
```
Text(0.5, 1.0, 'Combined')
```



In [ ]:
```
filter = np.array([[1/9 for i in range(3)] for j in range(3)])
filter
```

Out[ ]:
```
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]])
```

In [ ]:
```
gn_img
```

Out[ ]:
```
array([[255, 253, 237, ..., 152, 164, 191],
       [238, 237, 239, ..., 180, 167, 199],
       [238, 228, 236, ..., 193, 157, 213],
       ...,
       [242, 255, 250, ..., 237, 235, 224],
       [236, 255, 249, ..., 235, 241, 251],
       [252, 255, 233, ..., 241, 240, 227]], dtype=uint8)
```

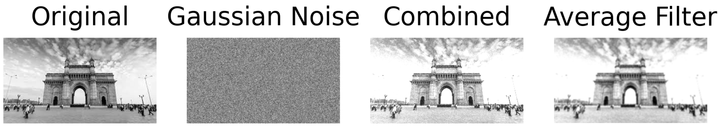
Average Filter

In [ ]:
```
output = []
for r in range(gn_img.shape[0]-2):
  temp = []
  for c in range(gn_img.shape[1]-2):
    lhs = gn_img[r:r+3, c:c+3]
    ans = 0
    for r_f in range(filter.shape[0]):
      for c_f in range(filter.shape[1]):
        ans += lhs[r_f][c_f]*filter[r_f][c_f]
    temp.append(ans)
  output.append(temp)
```

In [ ]:
```
output = np.array(output)
cv2_imshow(output)
```

In [ ]:
```
fig=plt.figure(dpi=400)

fig.add_subplot(1,4,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

fig.add_subplot(1,4,2)
plt.imshow(gauss_noise,cmap='gray')
plt.axis("off")
plt.title("Gaussian Noise")

fig.add_subplot(1,4,3)
plt.imshow(gn_img,cmap='gray')
plt.axis("off")
plt.title("Combined")

fig.add_subplot(1,4,4)
plt.imshow(output,cmap='gray')
plt.axis("off")
plt.title("Average Filter")
```

Out[ ]:
```
Text(0.5, 1.0, 'Average Filter')
```



In [ ]:

Median Filter

In [ ]:
```
import random
salt_and_pepper_img = img.copy()
salt_and_pepper_img
```

Out[ ]:
```
array([[193, 188, 180, ..., 107, 100, 111],
       [182, 179, 173, ..., 111, 106, 115],
       [170, 168, 165, ..., 117, 115, 123],
       ...,
       [182, 182, 181, ..., 173, 167, 161],
       [189, 188, 187, ..., 177, 176, 175],
       [176, 175, 173, ..., 161, 166, 171]], dtype=uint8)
```

In [ ]:
```
for i in range(200):
  row = random.randint(0, img.shape[0]-1)
  col = random.randint(0, img.shape[1]-1)
  n = random.randint(0,1)
  if(n==0):
    salt_and_pepper_img[row][col] = 0
  else:
    salt_and_pepper_img[row][col] = 255
```

In [ ]:
```
cv2_imshow(salt_and_pepper_img)
```

In [ ]:
```
output = []
for r in range(salt_and_pepper_img.shape[0]-2):
  temp = []
  for c in range(salt_and_pepper_img.shape[1]-2):
    lhs = salt_and_pepper_img[r:r+3, c:c+3]
    lhs_flatten = lhs.flatten()
    lhs_flatten.sort()
    temp.append(lhs_flatten[4])
  output.append(temp)
```

In [ ]:
```
output = np.array(output)
```

In [ ]:
```
fig=plt.figure(dpi=400)

fig.add_subplot(1,3,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

fig.add_subplot(1,3,2)
plt.imshow(salt_and_pepper_img,cmap='gray')
plt.axis("off")
plt.title("Combined")

fig.add_subplot(1,3,3)
plt.imshow(output,cmap='gray')
plt.axis("off")
plt.title("Median Filter")
```

Out[ ]:
```
Text(0.5, 1.0, 'Median Filter')
```

Name : Sarvagya Singh **LAB5** **SAPID : 60009200030(K1)**

AIM: To perform Image Enhancement(SD) Neighbourhood Processing Techniques: Sharpening Operators

THEORY:

1. **Basic High Pass Filter**

The principal objective of high pass (sharpening) filter is to highlight fine detail in an image or to enhance detail that has been blurred, either in error or as a natural effect of a particular method of image acquisition. Uses of image sharpening vary and include applications ranging from electronic printing and medical imaging to industrial inspection and autonomous target detection in military systems.

The shape of the impulse response needed to have a high pass (sharpening) spatial filter indicates that the filter should have positive coefficients in the outer periphery. For a 3 x 3 mask, choosing a positive value in the centre location with negative coefficients in the rest of the mask meets this condition. Thus when the mask is over an area of constant or slowly varying gray level, the output of the mask is zero or very small. This result is consistent with what is expected from the corresponding frequency domain filter.

-1	-1	-1
-1	8	-1
-1	-1	-1

Fig: A high Pass {Laplacian} filter mask

2. **High Boost Filtering**

A process used for many years in the publishing industry to sharpen images consists of subtracting a blurred version of an image from the image itself.

This process, called unsharp masking, is expressed as:

$$f_s(x,y) = f(x, y) - f'(x,y)$$

where, $f_s(x,y)$ denotes the sharpened image obtained by unsharp masking and $f'(x,y)$ is a blurred version of $f(x,y)$. The origin of unsharp masking is in dark room photography, where it consists of clamping together a blurred negative to corresponding positive film and then developing this combination to produce a sharper image.

A further generalization of unsharp masking is called high-boost filtering. A high-boost filtered image, f_{hb}, is defined at any point (x, y) as

$$f_{hb}(x, y) = A f(x,y) - f'(x,y)$$

where A=>1 and as before $f'(x,y)$ is the blurred image of $f(x,y)$

The final expression for high-boost image is given as

$$f_{hb}(x, y) = (A-1) f(x,y) + f_s(x,y)$$

The above given equation is applicable in general and does not state explicitly how the sharp image is obtained.

The mask for high-boost filtering is given as:

-1	-1	-1
-1	A+8	-1
-1	-1	-1

Fig: A mask for High-Boost filtering

One of the principal applications of high-boost filtering is when the input image is darker than desired. By varying the boost co-efficient, it generally is possible to obtain an overall increase in average gray level of the image, thus helping to brighten the final result.

RESULT:

A high pass filter is the basis for most sharpening methods. The high pass filters only allow those pixels with values above more values. The high pass filters have negative as well as positive values in the pixels. Due to this it creates a net values which results in highly distorted images. The high boost filter can be used to enhance the high frequency components. We can sharpen edges of a image through the amplification and obtain a more clear image. The high boost filter is simply the sharpening operator in image processing. High boost filter is a popular method that allows sharpening in high detail areas but little or no sharpening in flat or smooth areas. A high boost filter is used to retain some of the low-frequency components to and in the interpretation of a image.

Sarvagya Singh

60009200030 -- K1

IPCV - LAB5

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

In [ ]:
```
img = cv2.imread('/content/6.jpg', 0)
cv2_imshow(img)
```

In [ ]:
```
img.shape
```

Out[ ]:
```
(534, 800)
```

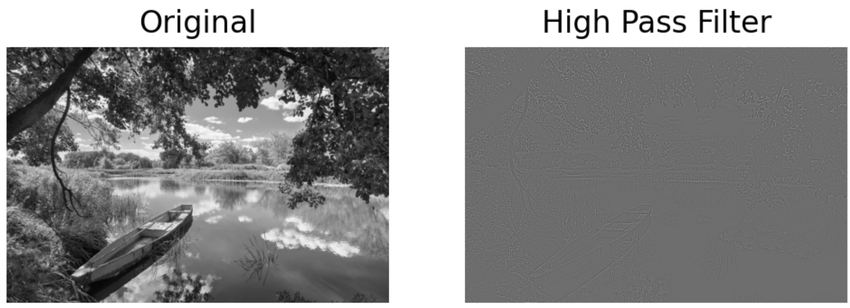
In [ ]:
```
filter = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
filter
```

Out[ ]:
```
array([[-1, -1, -1],
       [-1,  9, -1],
       [-1, -1, -1]])
```

In [ ]:
```
output = []
for r in range(img.shape[0]-2):
  temp = []
  for c in range(img.shape[1]-2):
    lhs = img[r:r+3, c:c+3]
    ans = 0
    for r_f in range(filter.shape[0]):
      for c_f in range(filter.shape[1]):
        ans += lhs[r_f][c_f]*filter[r_f][c_f]
    temp.append(ans)
  output.append(temp)
```

In [ ]:
```
fig=plt.figure(dpi=200)

fig.add_subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

fig.add_subplot(1,2,2)
plt.imshow(output,cmap='gray')
plt.axis("off")
plt.title("High Pass Filter")
```

Out[ ]:
```
Text(0.5, 1.0, 'High Pass Filter')
```



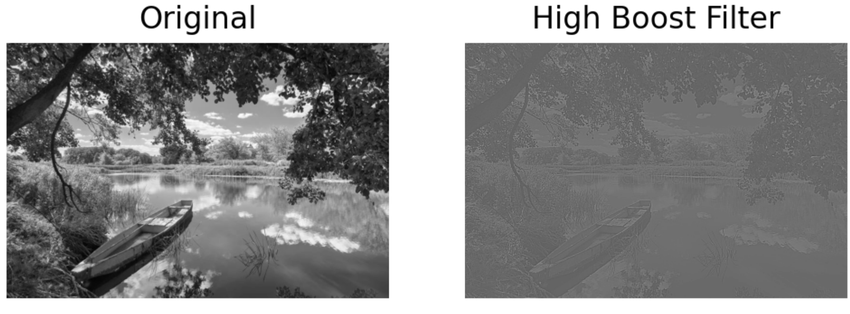
In [ ]:
```
filter_hb = np.array([[-1, -1, -1], [-1, 8.9, -1], [-1, -1, -1]])
filter_hb
```

Out[ ]:
```
array([[-1. , -1. , -1. ],
       [-1. ,  8.9, -1. ],
       [-1. , -1. , -1. ]])
```

In [ ]:
```
output_hb = []
for r in range(img.shape[0]-2):
  temp = []
  for c in range(img.shape[1]-2):
    lhs = img[r:r+3, c:c+3]
    ans = 0
    for r_f in range(filter_hb.shape[0]):
      for c_f in range(filter_hb.shape[1]):
        ans += lhs[r_f][c_f]*filter_hb[r_f][c_f]
    temp.append(ans)
  output_hb.append(temp)
```

In [ ]:
```
fig=plt.figure(dpi=200)

fig.add_subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.axis("off")
plt.title("Original")

fig.add_subplot(1,2,2)
plt.imshow(output_hb,cmap='gray')
plt.axis("off")
plt.title("High Boost Filter")
```

Out[ ]:
```
Text(0.5, 1.0, 'High Boost Filter')
```



In [ ]:
```
!jupyter nbconvert --to html ''
```

Name : Sarvagya Singh **SAPID : 60009200030**

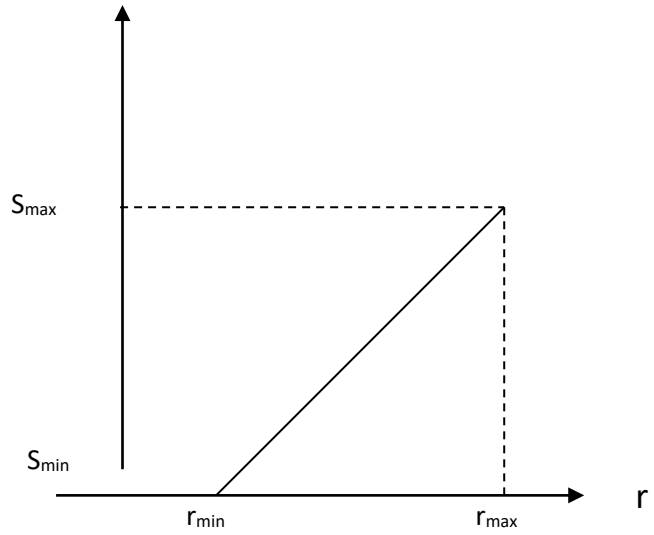
AIM: To Perform Histogram Stretching

THEORY:

Histogram Stretching

It is a method to increase the dynamic range of the image. Here we do not alter the basic shape of the histogram, but we spread it so as to cover the entire dynamic range. We do this by using a straight line equation having a slope

$(s_{max} - s_{min})/ (r_{max} - r_{min})$



s_{max} = Maximum grey level of output image

s_{min} = Minimum grey level of output image.

r_{max} = Maximum grey level of input image

r_{min} = Minimum grey level of input image.

$$S = T(r) = \frac{(smax - smin)}{(rmax - rmin)} (r - r_{min}) + smin$$

This transformation stretches and shifts the grey level range of input image to occupy the entire dynamic range (s_{max}, s_{min}).

RESULT:

So we have performed the image contrast stretching and thereby doing it the low contrast images start to have a larger range of image pixel values. This increases the range of intensity of the pixel values.The Histogram stretching increases the pixel range of the images but the distribution of the image remains the same thereby making sure not to create any anomaly.

Sarvayga Singh

60009200030 - K1

IPCV - lab6

Contrast Stretching

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

In [2]:
```
img = cv2.imread('/content/imager.jpeg',0)
img
```

Out[2]:
```
array([[206, 206, 206, ..., 207, 207, 207],
       [206, 206, 206, ..., 207, 207, 207],
       [206, 206, 206, ..., 207, 207, 207],
       ...,
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255]], dtype=uint8)
```

In [3]:
```
np.shape(img)
```

Out[3]:
```
(280, 600)
```

In [4]:
```
final_img = [[0 for j in range(np.shape(img)[1])] for i in range(np.shape(img)[0])]
np.shape(final_img)
```

Out[4]:
```
(280, 600)
```

In [5]:
```
image = cv2_imshow(img)
```

In [6]:
```
rmax,rmin = np.max(img),np.min(img)
rX = (rmax , rmin)
rX
```

Out[6]:
```
(255, 31)
```

In [7]:
```
smax,smin = 255,0
sX = (smax , smin)
sX
```

Out[7]:
```
(255, 0)
```

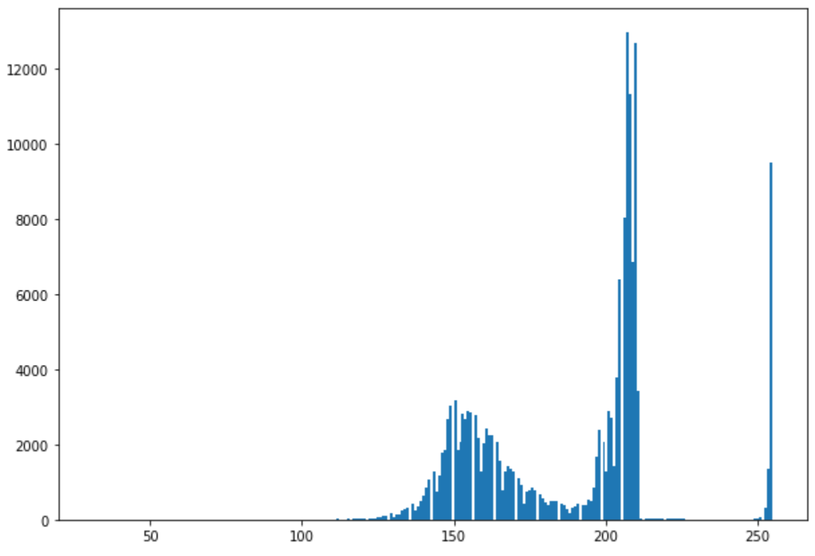
In [8]:
```
ratio = (sX[0]-sX[1])/(rX[0]-rX[1])
ratio
```

Out[8]:
```
1.1383928571428572
```

In [9]:
```
imgX = np.reshape(img,-1)
imgX
```

Out[9]:
```
array([206, 206, 206, ..., 255, 255, 255], dtype=uint8)
```

In [10]:
```
plt.figure(figsize=(10,7))
plt.hist(imgX, bins=256);
```



In [11]:
```
for i,lister in enumerate(img):
  for j,value in enumerate(lister):
    final_img[i][j] = ratio*(value-rmin) + smin

final_img = np.array(final_img)
```

In [12]:
```
np.shape(final_img)
```

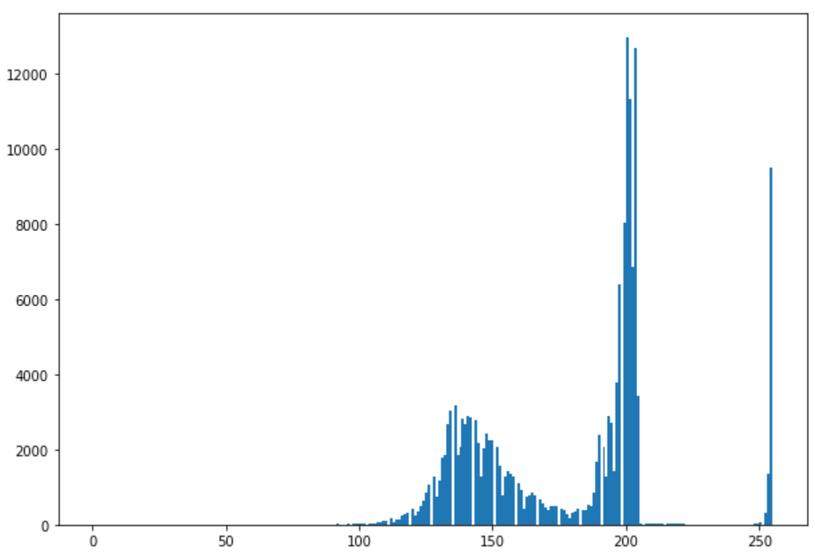
Out[12]:
```
(280, 600)
```

In [13]:
```
final_imgX = np.reshape(final_img,-1)
final_imgX
```

Out[13]:
```
array([199.21875, 199.21875, 199.21875, ..., 255.     , 255.     ,
       255.     ])
```

In [14]:
```
cv2_imshow(final_img)
```

In [16]:
```
plt.figure(figsize=(10,7))
plt.hist(final_imgX, bins=256);
```



In [17]:
```
np.max(final_imgX),np.min(final_imgX)
```

Out[17]:
```
(255.0, 0.0)
```

Name : Sarvagya Singh – K1 Lab7 **SAPID : 60009200030**

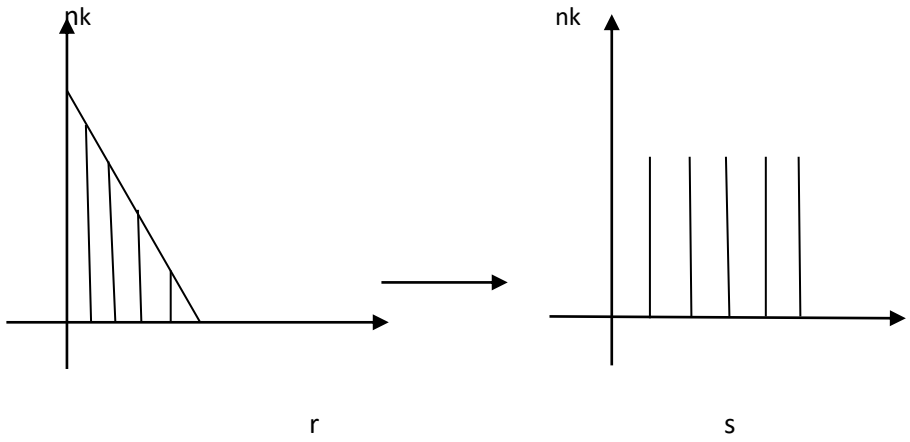
AIM: To Perform Histogram Equalization

THEORY:

Histogram Equalisation

There are many applications wherein we need a flat histogram. This cannot be achieved by histogram stretching. Hence histogram equalization was introduced.

A perfect image is one which has equal number of pixels in all its grey levels. Hence our objective is not only to spread the dynamic range, but also to have pixels in all grey levels. This technique is known as Histogram equalization.



The transformation must satisfy the two conditions:

- T (r) must be single valued and monotonically increasing in the interval $0 < r < 1$ and,
- $0 < T(r) < 1$ for $0 < r < 1$.

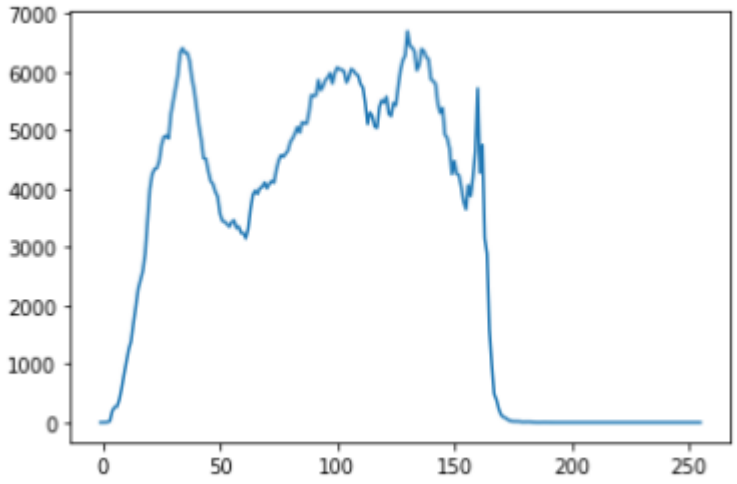
Hence the range of r is taken as [0,1]. This is called the normalized range. This range is taken for simplicity. So instead of r being in the range [0,255] we take [0,1].

RESULT:

The Histogram Equalization is a powerful way to stretch the histogram to any possible mostly between 0 and 256. The Histogram equalization can be used to understand and uniform the pixels value of different images like having it in the range of 256.It can be used to make much more powerful algorithms .

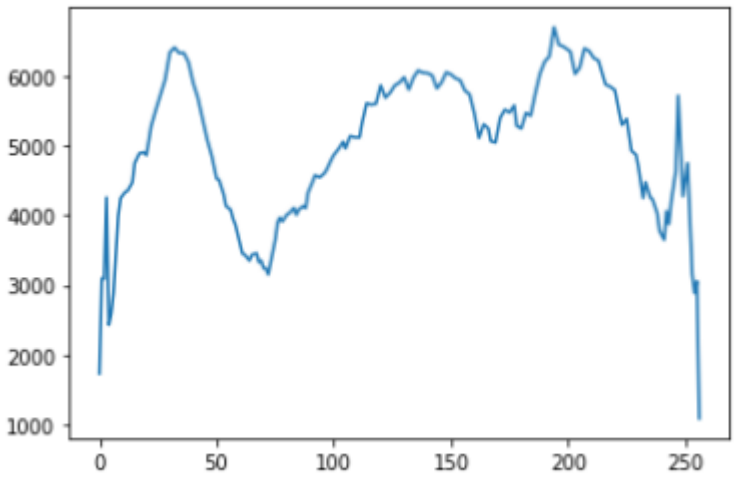
Before histogram Equalization :

```
plt.plot(list(Dict_values.keys()),list(Dict_values.value
```




```
Dict_values.keys()
```

After the Histogram Equalization :



Sarvayga Singh

60009200030 - K1

IPCV - lab6

Histogram Equalisation

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

In [2]:
```
img_size = (300,300)
img = [[0 for i in range(img_size[0])] for j in range(img_size[1])]
print(np.shape(img))
```

```
(300, 300)
```

In [3]:
```
total = img_size[0]*img_size[1]
count = 0
for i in range(img_size[0]):
  for j in range(img_size[1]):
    img[i][j] = count*(255/total)
    count += 1

img = np.array(img)
```

In [4]:
```
imgX = np.reshape(img,-1)
imgX
```

Out[4]:
```
array([0.00000000e+00, 2.83333333e-03, 5.66666667e-03, ...,
       2.54991500e+02, 2.54994333e+02, 2.54997167e+02])
```

In [5]:
```
np.shape(imgX)
```

Out[5]:
```
(90000,)
```

In [6]:
```
cv2_imshow(img)
```

In [7]:
```
img = cv2.imread('/content/imager.jpeg',0)
# plt.imshow(img);
cv2_imshow(img)
```

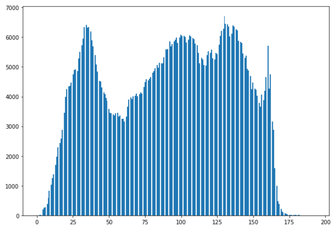
In [8]:
```
np.shape(img)
```

Out[8]:
```
(751, 1000)
```

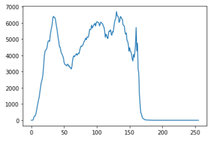
In [9]:
```
print(np.max(img),np.min(img))
```

```
193 0
```

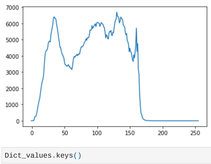
In [10]:
```
imgX = np.reshape(img,-1)
plt.figure(figsize=(10,7))
plt.hist(imgX, bins=256);
```



In [11]:
```
hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
plt.plot(hist1);
```



In [18]:
```
def create_Dist(img):
  indexer = {}
  for i,value1 in enumerate(img):
    for j,value2 in enumerate(value1):
      if value2 not in indexer:
        indexer[value2] = 1
      else:
        indexer[value2] += 1
  indexer[-1] = 0
  for i in range(256):
    try:
      indexer[i] = indexer[i]*1
      pass
    except:
      indexer[i] = 0

  lister = sorted(list(indexer.keys()))
  final = {i:indexer[i] for i in lister}
  return final
Dict_values = create_Dist(img)
plt.plot(list(Dict_values.keys()),list(Dict_values.values()));
```



In [ ]:
```
Dict_values.keys()
```

In [57]:
```
def stepper(num):
  num_list = str(num).split('.')
  if num_list[1][0]>='5':
    return int(num_list[0])+1
  else:
    return int(num_list[0])
# stepper(0.6003408778828229028)
```

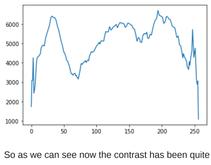
In [97]:
```
def Hist_equalizer(Dict_values):
  sum = 0
  cdf = {}
  # pdf_list = [0]
  for i in Dict_values.keys():
    sum += Dict_values[i]

  pdf = {i:(Dict_values[i]/sum) for i in Dict_values.keys()}
  pdf_list = list(pdf.values())
  cdf[-1] = 0
  # cdf = {i:(pdf_list[i]+cdf[i-1]) for i in range(5)} This wont work as this is dict comprehension as it does all of the computing at once so it does stores i
  for i in range(256):
    cdf[i] = pdf_list[i]+cdf[i-1]
  fcdf = {i:cdf[i]*256 for i in range(256)}
  hist_equalizer = map(lambda x: stepper(x), list(fcdf.values()))
  return hist_equalizer

Hist_equi = list(Hist_equalizer(Dict_values))
```

In [99]:
```
len(Hist_equi)
```

Out[99]:
```
256
```

In [ ]:
```
finalx = {}
pixels = list(Dict_values.values())
for index,i in enumerate(Hist_equi):
  if i not in finalx.keys():
    finalx[i] = pixels[index+1]
  else:
    finalx[i] += pixels[index+1]
```

In [102]:
```
plt.plot(list(finalx.keys()),list(finalx.values()));
```



So as we can see now the contrast has been quite significantly in comparison to that of the original image

In [ ]:
```
# img_2 = cv2.equalizeHist(img)
# cv2_imshow(img_2)

# hist2 = cv2.calcHist([img_2],[0],None,[256],[0,256])
# plt.plot(hist2);
```

In [ ]:
```
!jupyter nbconvert --to html ''
```

Name : Sarvagya Singh **SAPID : 60009200030** **Batch : K1**

AIM: To implement Ideal Low Pass & High Pass Filtering on an image

THEORY:

1. Ideal Low Pass Filter

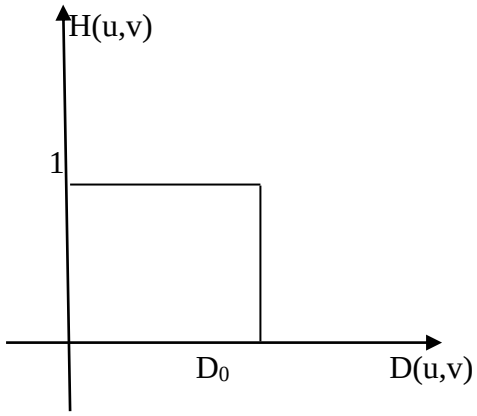
This filter cuts off all high frequency components of the Fourier transform that are at a distance greater than a specified distance D_0.

$$H(u,v) = 1; \text{ if } D(u,v) < D_0$$

$$= 0; \text{ if } D(u,v) > D_0$$

Where,

D_0 is the specified non negative distance.



Response of Ideal Low Pass Filter

D(u,v) is the distance from the point (u,v) to the origin of the frequency rectangle for an M X N image.

$$D(u,v)=[(u-M/2)^2 + (v-N/2)^2]^{\wedge} \frac{1}{2}$$

Therefore ,

For an image, when u=M/2 , v=N/2

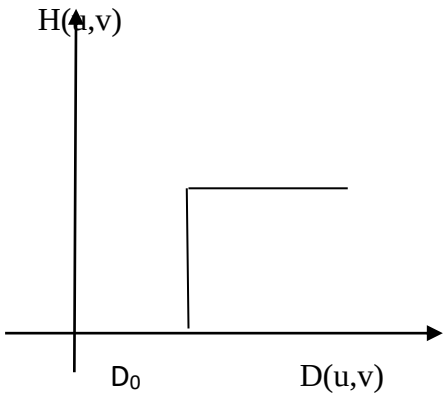
$$D(u,v)=0$$

This formula centers our H(u,v).

D(u,v) gives us concentric rings with each ring having a fixed value.

2. Ideal High Pass Filter

This filter cuts off all high frequency components of the Fourier transform that are at a distance greater than a specified distance D_0.



H(u,v) = 0; if D(u,v)< D0

= 1; if D(u,v)> D0

Where,

D0 is the specified non negative distance.

D(u,v) is the distance from the point (u,v) to the origin of the frequency rectangle for an M X N image.

RESULT:

An ideal low-pass filter completely eliminates all frequencies above the cutoff frequency while passing those below unchanged.Ideal low pass filters blurs the image where the High pass filters only shows/highlights the part of the image which is important

Ideal low pass filter and High pass filter

Sarvayga Singh -- 60009200030 (K1)

Lab8

In [1]:
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
```

In [2]:
```
def step(num):
  num_list = num.split('.')
  if num_list[1] > 5:
    return int(num_list(1))+1
  else:
    return int(num_list[0])
```

In [ ]:
```
img = cv2.imread('/content/image.jpg',0)
cv2_imshow(img)
```

In [ ]:
```
M,N = img.shape[0],img.shape[1]
M,N
```

Out[ ]:
```
(564, 849)
```

In [ ]:
```
np.max(img),np.min(img)
```

Out[ ]:
```
(218, 169)
```

In [ ]:
```
def dist(u,v):
  return np.sqrt((u - M/2)**2+(v-N/2)**2)
```

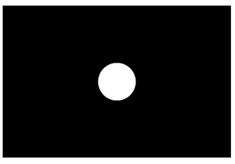
In [ ]:
```
F = np.fft.fft2(img)
Fshift = np.fft.fftshift(F)
```

In [ ]:
```
h = [[0 for i in range(N)] for j in range(M)]
h = np.array(h)
threshold = 70
print(len(h[0]),len(h))
```

```
849 564
```

In [ ]:
```
for i in range(0,M): #y axis
  for j in range(0,N): # x axis
    d = dist(i,j)
    if d<=threshold:
      h[i,j] = 1
    else:
      h[i,j] = 0
```

In [ ]:
```
plt.imshow(h, cmap='gray')
plt.axis('off')
plt.show()

Gshift = Fshift * h
G = np.fft.ifftshift(Gshift)
g = np.abs(np.fft.ifft2(G))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()
```

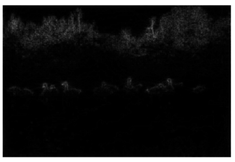


In [ ]:
```
np.shape(h)
```

Out[ ]:
```
(564, 849)
```

In [ ]:
```
i = [[1 for i in range(N)] for j in range(M)]
```

In [ ]:
```
# cv2_imshow(np.array(h))
# Filter: High pass filter
h = 1 - h
plt.imshow(h, cmap='gray')
plt.axis('off')
plt.show()
Gshift = Fshift * h
G = np.fft.ifftshift(Gshift)
g = np.abs(np.fft.ifft2(G))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()
```



In [ ]:
```
# calculating the discrete Fourier transform
DFT = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

# reposition the zero-frequency component to the spectrum's middle
shift = np.fft.fftshift(DFT)
row, col = img.shape
center_row, center_col = row // 2, col // 2

# create a mask with a centered square of 1s
mask = np.zeros((row, col, 2), np.uint8)
mask[center_row - 30:center_row + 30, center_col - 30:center_col + 30] = 1

# put the mask and inverse DFT in place.
fft_shift = shift * mask
fft_ifft_shift = np.fft.ifftshift(fft_shift)
imageThen = cv2.idft(fft_ifft_shift)

# calculate the magnitude of the inverse DFT
imageThen = cv2.magnitude(imageThen[:,:,0], imageThen[:,:,1])

# visualize the original image and the magnitude spectrum
plt.figure(figsize=(10,10))
plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(imageThen, cmap='gray')
plt.title('Low pass filter'), plt.xticks([]), plt.yticks([])
plt.show()
```

In [ ]:
```
```

Name : Sarvagya Singh **SAPID : 60009200030** **Batch : K1**

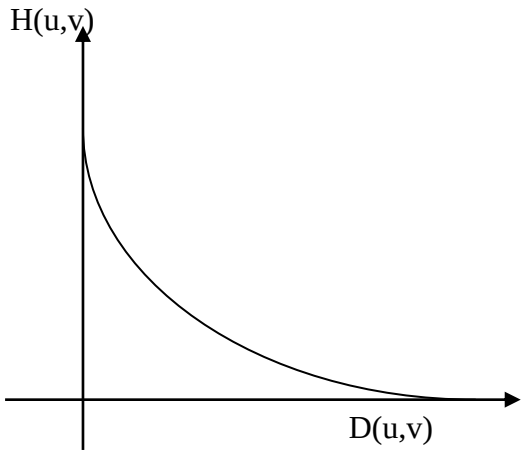
AIM: To implement Gaussian Low Pass & High Pass Filtering techniques on an image

THEORY:

1. Gaussian Low Pass Filter

Gaussian LPF is given by,

$$H(u,v) = e^{\wedge} -D^2 (u,v)/2\sigma^2$$



Where, σ is the standard deviation and is a measure of spread of the Gaussian curve. If we put $\sigma = D_0$ we get,

$$H(u,v) = e^{\wedge} -D^2 (u,v)/2D_0^2$$

The response of the Gaussian LPF is similar to that of BLPF but there are no ringing effects.

2. Gaussian High Pass Filter

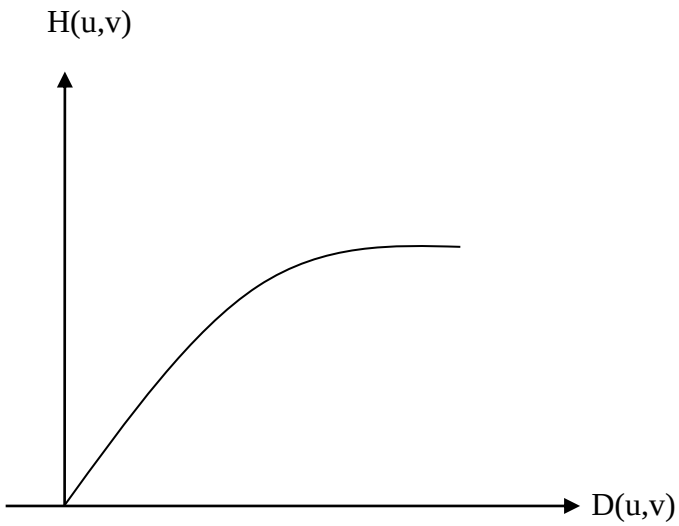
The basic formula is,

$$H\ hp\ (u,v) = 1- H\ lp\ (u,v)$$

Therefore ,

$$H\ Gaussian\ hp\ (u,v) = 1- H\ Gaussian\ lp\ (u,v)$$

$$H\ GHPF = 1- e^{\wedge} -D^2 (u,v)/2D0^2$$



The results of Gaussian high pass filter are smoother and also cleaner.

RESULT:

The Gaussian filters are generally used to reduced the size of the image thereby blurring the image. When downsampling an image, it is common to apply a low-pass filter to the image prior to resampling. This is to ensure that spurious high-frequency information does not appear in the downsampled image.

Ideal Gaussian low pass filter

Sarvayga Singh -- 60009200030 (K1)

Lab9

In [1]:
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread('/content/image.png',0)
# cv2_imshow(img)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()
```

In [6]:
```
m,n = img.shape[0],img.shape[1]
m,n
```

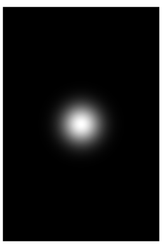
Out[6]:
```
(900, 600)
```

In [4]:
```
np.max(img),np.min(img)
```

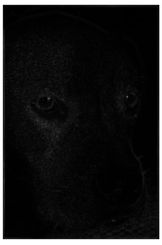
Out[4]:
```
(255, 0)
```

In [5]:
```
def dist(u,v):
  return np.sqrt((u - M/2)**2+(v-N/2)**2)
```

In [7]:
```
D0 = 50
F = np.fft.fft2(img)
Fshift = np.fft.fftshift(F)
H = np.zeros((m,n), dtype=np.float32)
for u in range(m):
    for v in range(n):
        D = np.sqrt((u-m/2)**2 + (v-n/2)**2)
        H[u,v] = np.exp(-(D**2)/(2*(D0**2)))

plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()

plt.imshow(H, cmap='gray')
plt.axis('off')
plt.show()

Gshift = Fshift * H
G = np.fft.ifftshift(Gshift)
g = np.abs(np.fft.ifft2(G))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()
```



In [8]:
```
# Filter: High pass filter
H = 1 - H
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()

plt.imshow(H, cmap='gray')
plt.axis('off')
plt.show()

Gshift = Fshift * H
G = np.fft.ifftshift(Gshift)
g = np.abs(np.fft.ifft2(G))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()
```



In [ ]:
```
```