Computer Vision (CSE3010)

Dr. Susant Kumar Panigrahi
Assistant Professor
School of Electrical & Electronics Engineering

Module-1 Syllabus

Digital Image Formation And Low Level Processing:

- Overview and State-of-the-art, Fundamentals of Image Formation, Transformation: Orthogonal, Euclidean, Affine, Projective, Fourier Transform,
- Convolution and Filtering, Image Enhancement, Restoration, Histogram Processing.

Module-2 Syllabus

Depth Estimation And Multi-Camera Views:

Depth Estimation and Multi-Camera Views: Perspective, Binocular Stereopsis: Camera and Epipolar Geometry; Homography, Rectification, DLT, RANSAC, 3-D reconstruction framework; Auto-calibration. apparel.

Module-3 Syllabus

Feature Extraction And Image Segmentation:

- **Feature Extraction**: Edges - Canny, LOG, DOG; Line detectors (Hough Transform), Corners - Harris and Hessian Affine, Orientation Histogram, SIFT, SURF, HOG, GLOH, Scale-Space Analysis- Image Pyramids and Gaussian derivative filters, Gabor Filters and DWT.
- **Image Segmentation:** Region Growing, Edge Based approaches to segmentation, Graph-Cut, Mean-Shift, MRFs, Texture Segmentation; Object detection.

Module-4 Syllabus

Pattern Analysis And Motion Analysis:

- **Pattern Analysis**: Clustering: K-Means, K-Medoids, Mixture of Gaussians, Classification: Discriminant Function, Supervised, Un-supervised, Semi-supervised; Classifiers: Bayes, KNN, ANN models;
- **Dimensionality Reduction**: PCA, LDA, ICA; Non-parametric methods. Motion Analysis: Background Subtraction and Modelling, Optical Flow, KLT, Spatio-Temporal Analysis, Dynamic Stereo; Motion parameter estimation.

Module-5 Syllabus

Shape From X:

Light at Surfaces; Phong Model; Reflectance Map;

Albedo estimation; Photometric Stereo; Use of Surface Smoothness

Constraint; Shape from Texture, color, motion and edges.

Guest Lecture on Contemporary Topics

Text Books

1. Richard Szeliski, Computer Vision: Algorithms and Applications, Springer-Verlag London Limited 2011.
2. Computer Vision: A Modern Approach, D. A. Forsyth, J. Ponce, Pearson Education, 2003.

Reference Book(s):

1. R.C. Gonzalez and R.E. Woods, Digital Image Processing, Addison- Wesley, 1992.
2. Richard Hartley and Andrew Zisserman, Multiple View Geometry in Computer Vision, Second Edition, Cambridge University Press, March 2004.
3. K. Fukunaga; Introduction to Statistical Pattern Recognition, Second Edition, Academic Press, Morgan Kaufmann, 1990.

Required Tools/Software/IDLE:

1. Python/jupyter-notebook/google-colab
2. OpenCV
3. MATLAB

Indicative List of Experiments:

1. Implement image preprocessing and Edge
2. Implement camera calibration methods
3. Implement Projection
4. Determine depth map from Stereo pair
5. Construct 3D model from Stereo pair
6. Implement Segmentation methods
7. Construct 3D model from defocus image
8. Construct 3D model from Images
9. Implement optical flow method
10. Implement object detection and tracking from video
11. Face detection and Recognition
12. Object detection from dynamic Background for Surveillance
13. Content based video retrieval
14. Construct 3D model from single image

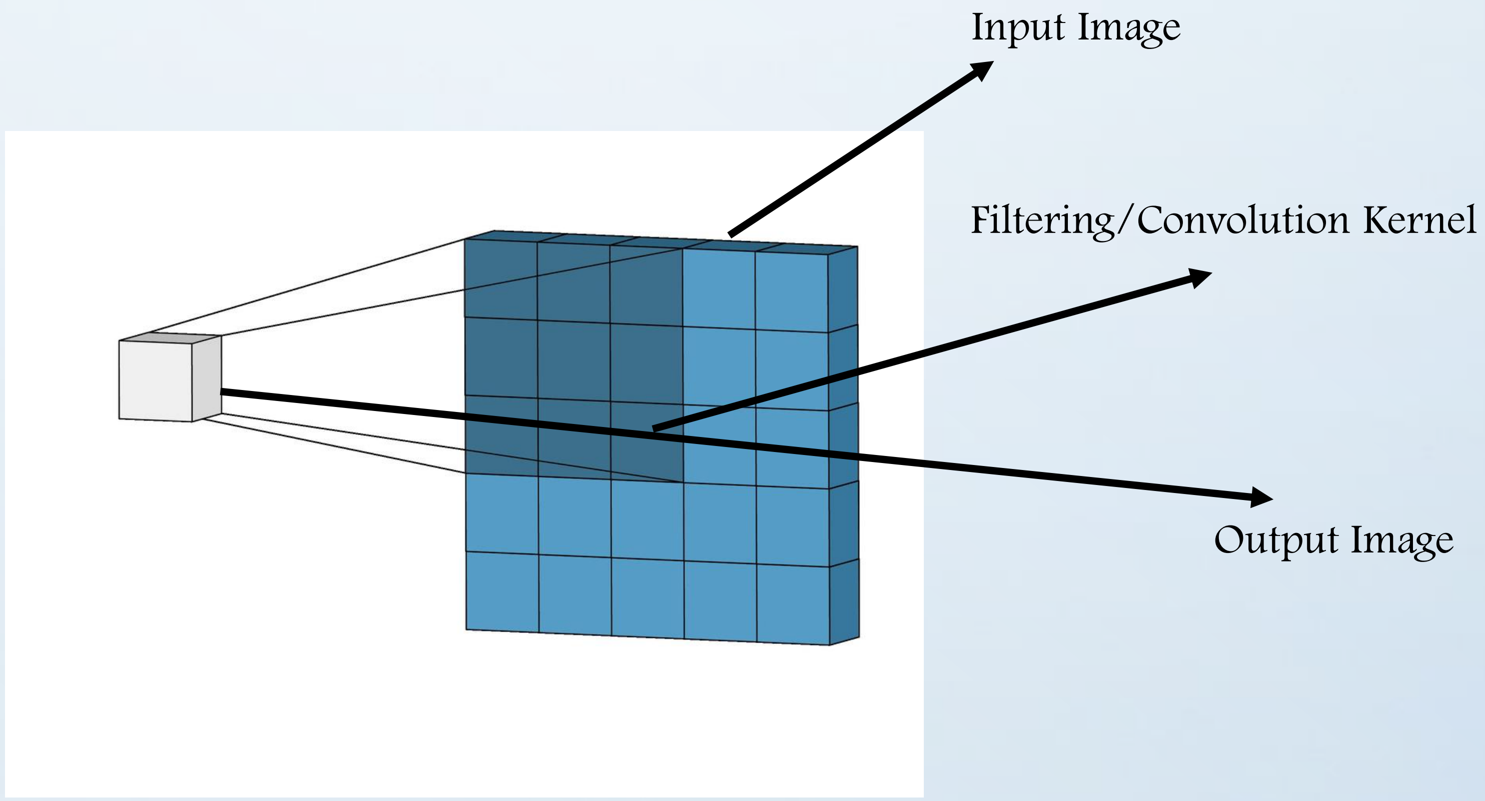
Computer Vision

Unit – 01

Low-Level Vision (Image Filtering and Histogram Processing)

Convolution and Filtering

Weighted average between image and filtering kernel. Depending on the choice of kernels the filtered output image may change. It is one of the most fundamental building block for many image processing and computer-vision algorithms that expressed how the shape/structure of the input image is modified by the convolution kernel.



Source : https://miro.medium.com/max/2340/1*Fw-ehcNBR9byHtho-Rxbtw.gif

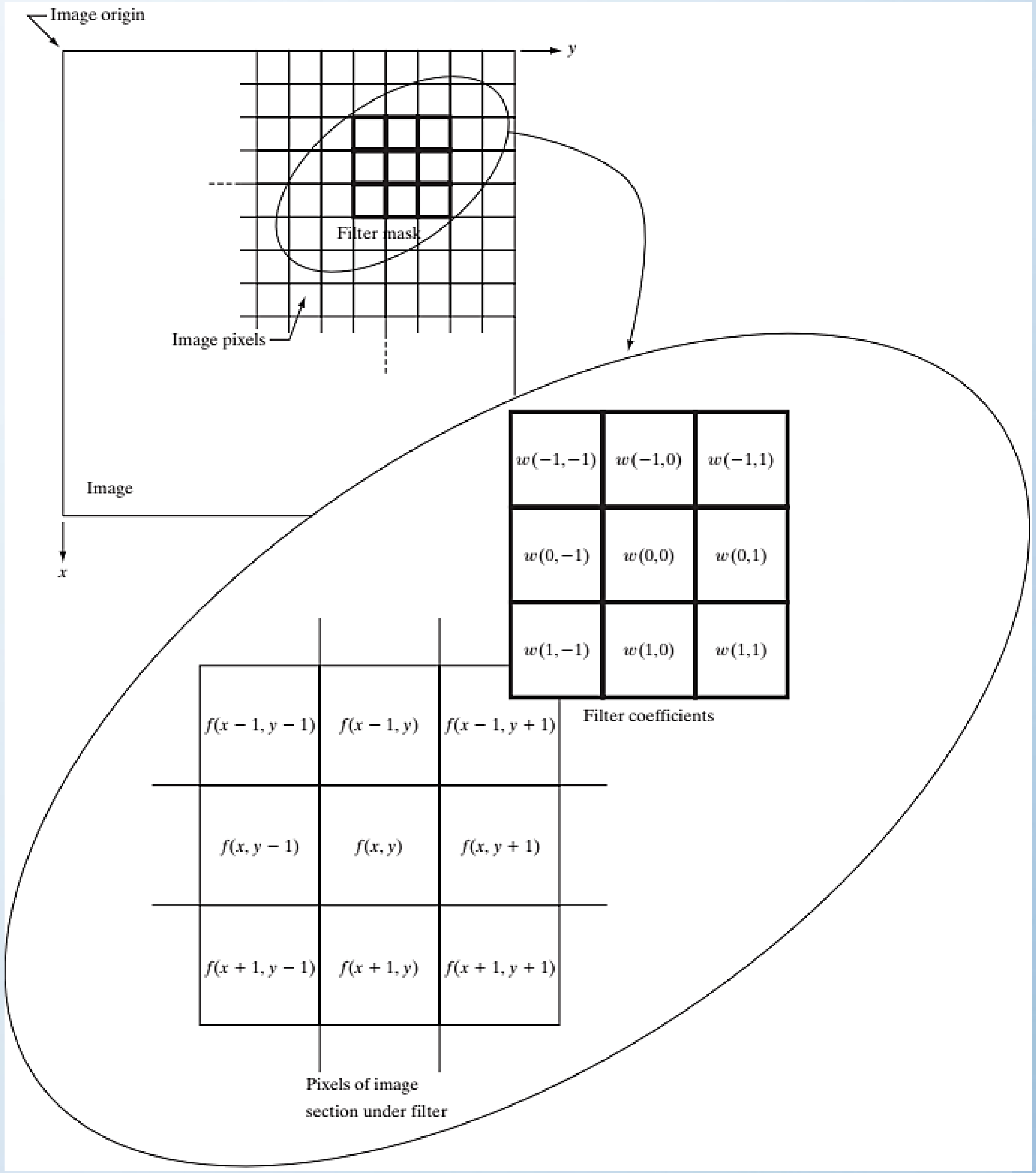
Convolution (Neighborhood Operations)

Neighborhood processing is an appropriate name because you define a center point and perform an operation (or apply a filter known as convolution) to only those pixels in predetermined neighborhood of that center point.

- As illustrated in the figure; the image f at any pixel center location (x,y) produces output image g as the weighted average between f and the filter w: Mathematically:

$$g(x, y) = f(x, y) * w(x, y)$$

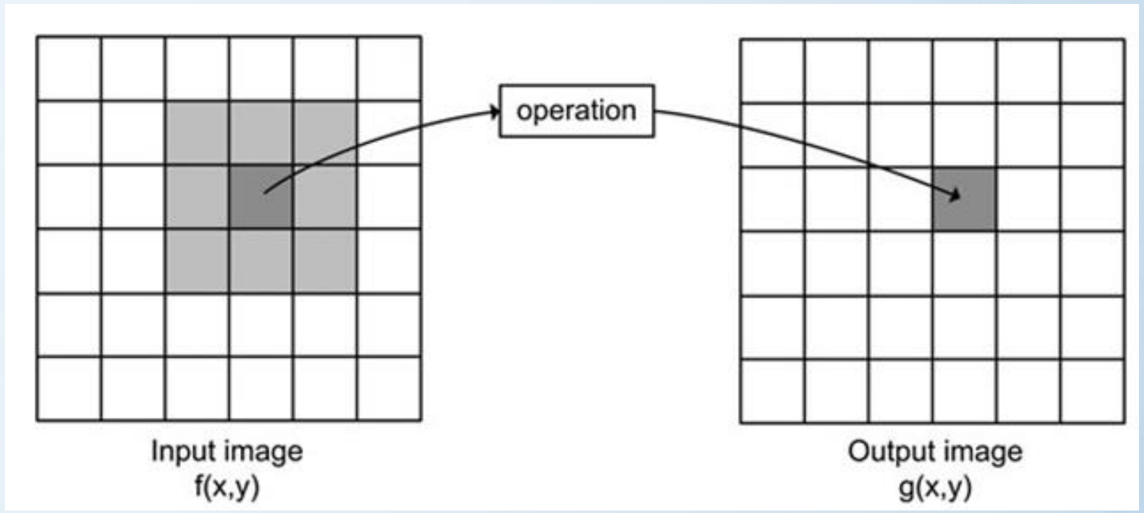
$$= \sum_{s=-a}^{a} \sum_{s=-b}^{b} w(s,t) f(x+s, y+t)$$



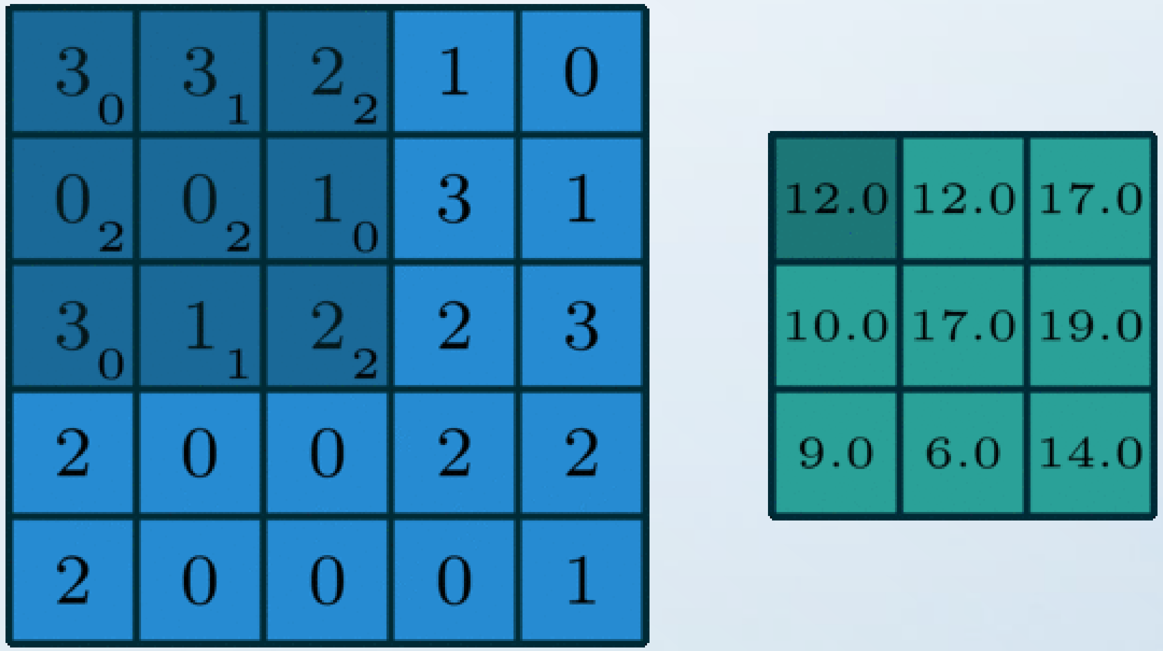
Convolution (Neighborhood Operations)

Example:

The fixed window or kernel slides over the image and performs element-wise multiplication to produce the resultant sum as the output pixel value at the center location: as represented below:



Sample model of a 3 × 3 convolution kernel sliding over a 6 × 6 image



3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

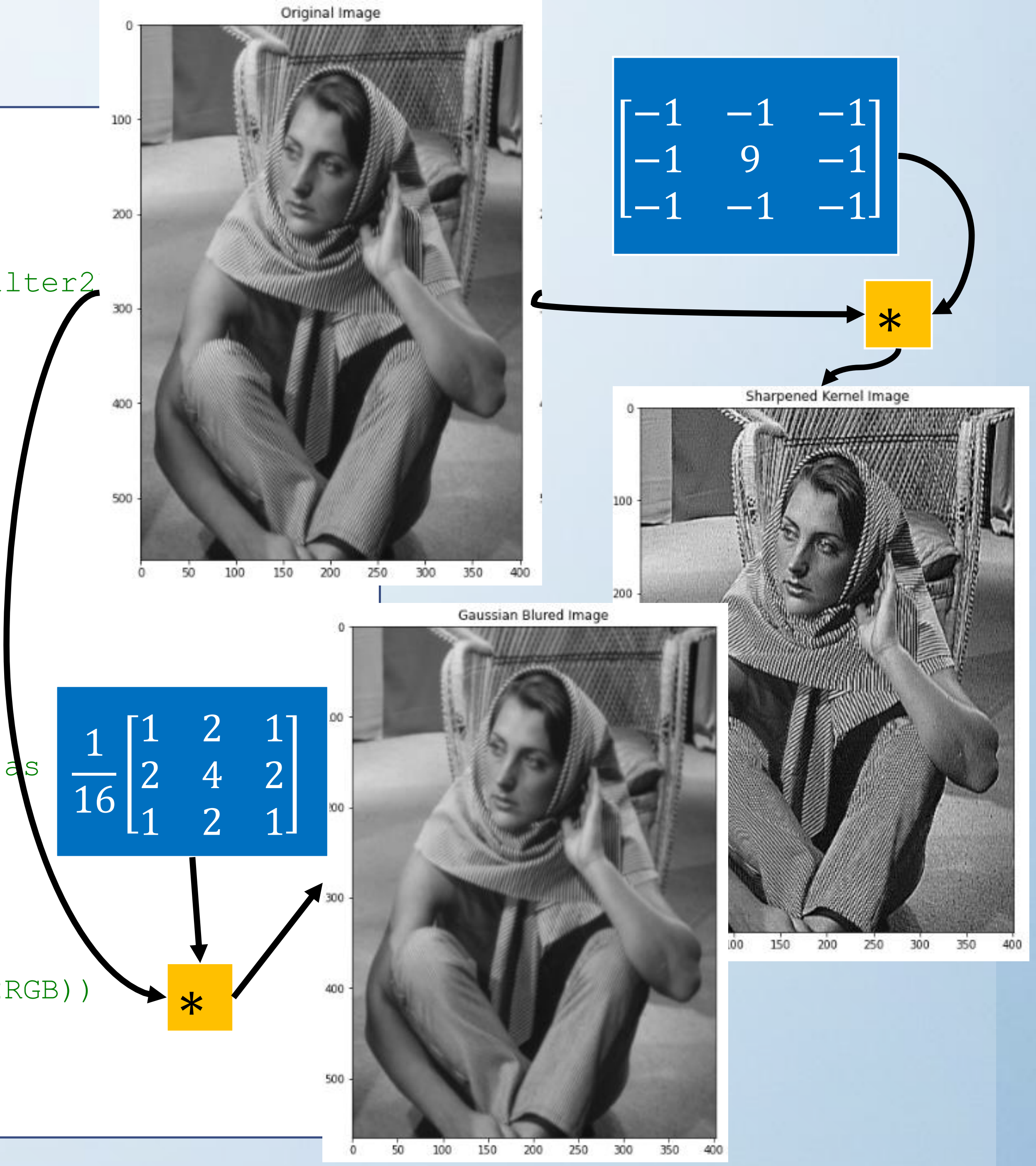
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Demonstration of simple convolution operation.
Source: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Applications of Convolution:
- Image Smoothing
- Noise Reduction
- Image Sharpening
- Edge detection

Convolution (Neighborhood Operations)

```
image = cv2.imread("/content/barbara.tif")

fig, ax = plt.subplots(1, 3, figsize=(16, 8))
fig.tight_layout()
# To conovolve the kernel on an image we can use cv.filter2
#ax[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
ax[0].imshow(image)
ax[0].set_title('Original Image')

kernel_sharpening = np.array([[-1, -1, -1],
                             [-1, 9, -1],
                             [-1, -1, -1]])

kernel_GaussBlur = (1/16)*np.array([[1, 2, 1],
                             [2, 4, 2],
                             [1, 2, 1]])
# Convolution operation using cv2.filter2D ---------
# depth = -1 will give the output image depth as same as
sharpened = cv2.filter2D(image, -1, kernel_sharpening)
ax[1].imshow(sharpened)
ax[1].set_title('Sharpened Kernel Image')

GaussBlur = cv2.filter2D(image, -1, kernel_GaussBlur)
#ax[2].imshow(cv2.cvtColor(sharpened_2, cv2.COLOR_BGR2RGB))
ax[2].imshow(GaussBlur)
ax[2].set_title('Gaussian Blured Image')
plt.show()
```



$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix}$$

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Spatial Domain Filtering (Blurring and Smoothing)

Convolution Kernel	Filtering Operation
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	Identity Filter
$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	Average Filtering or Box Blur
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	Sharpening Filter
$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	Gaussian Blur
$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	Unsharp Mask 5×5

Image Enhancement by Point Processing

Image enhancement (that directly applied on the image intensity values) simply means, transforming an image f into image g using T, (Where T is the transformation). The values of pixels in images f and g are denoted by r and s, respectively. As said, the pixel values r and s are related by the expression

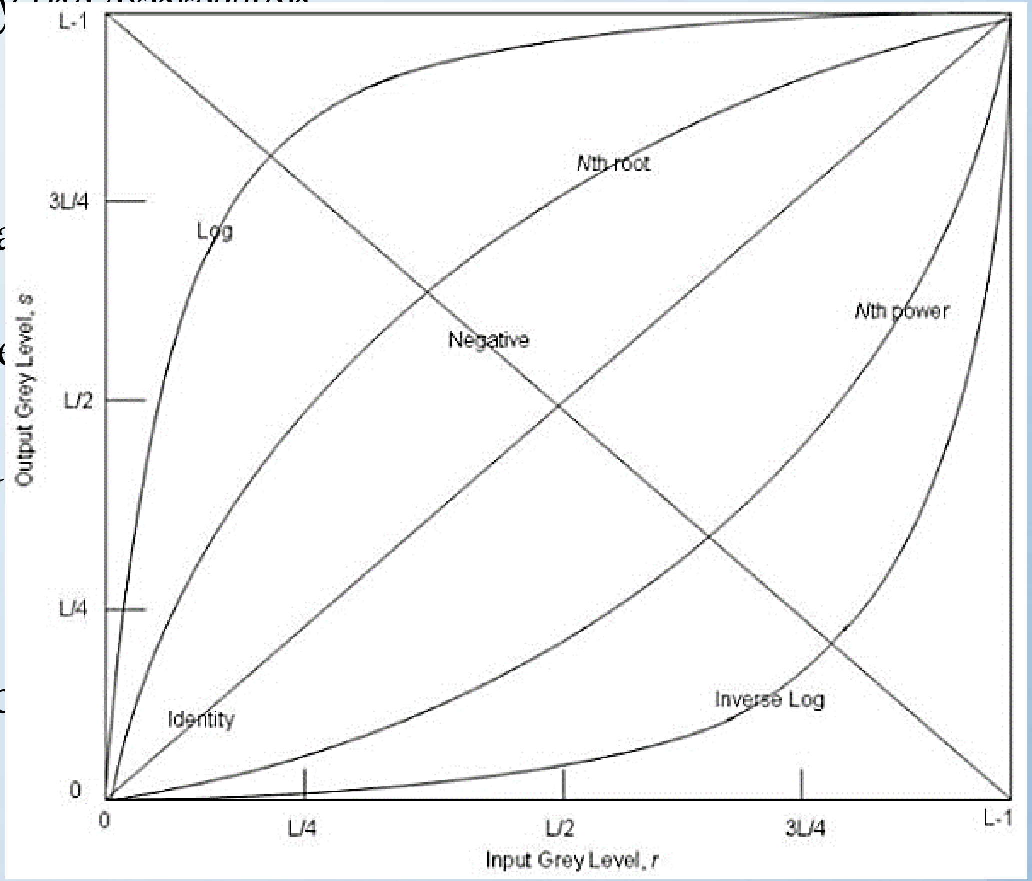
$$s = T(r)$$

where T is a transformation that maps a pixel value r into a pixel va

mapped into the grayscale range as we are dealing here only with gre

back into the range $[0, L-1]$, where $L = 2^k$, k being the num

So, for instance, for an 8-bit image the range of pixel values will be [0



a. Image Negatives

The negative of an image with grey levels in the range $[0, L-1]$ is obtained by the negative transformation, which is given by the expression,

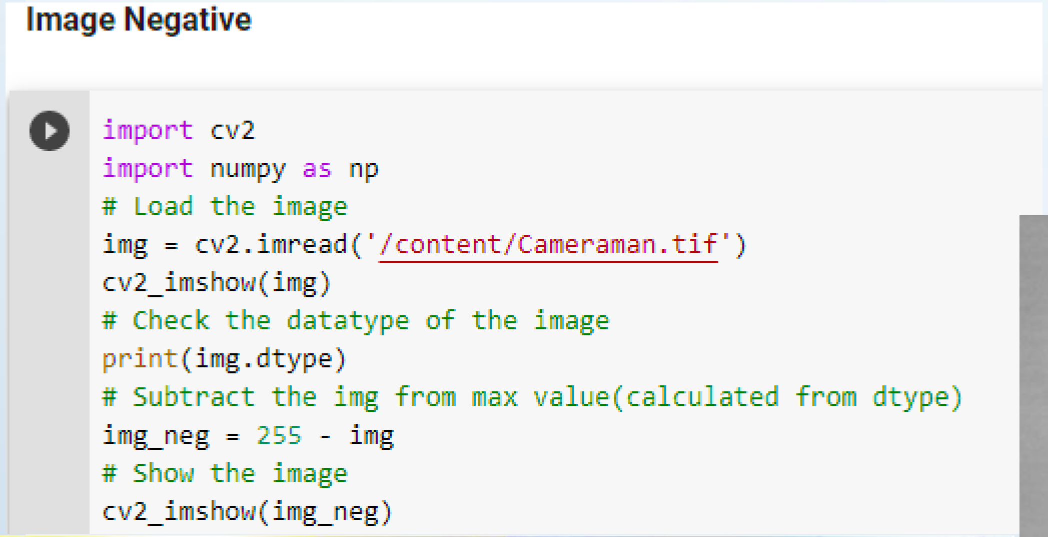
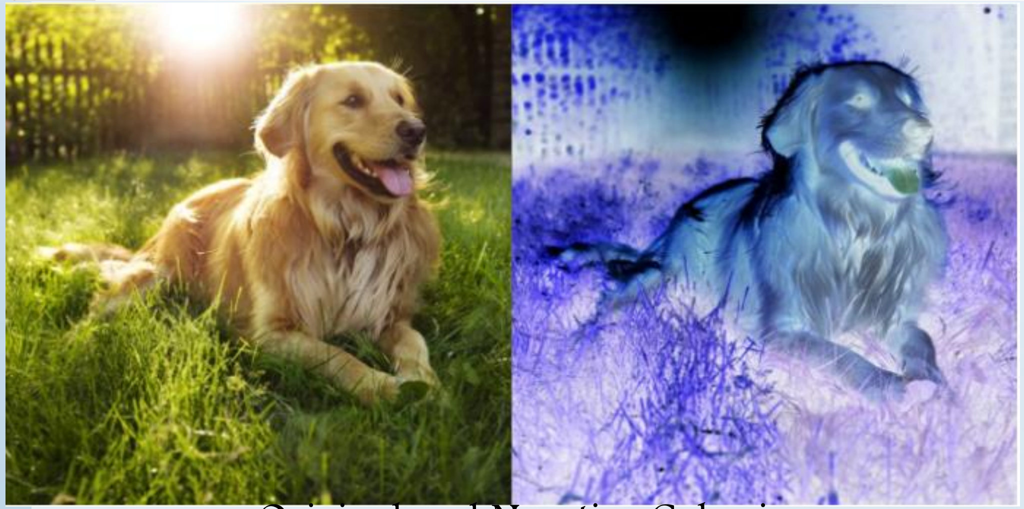


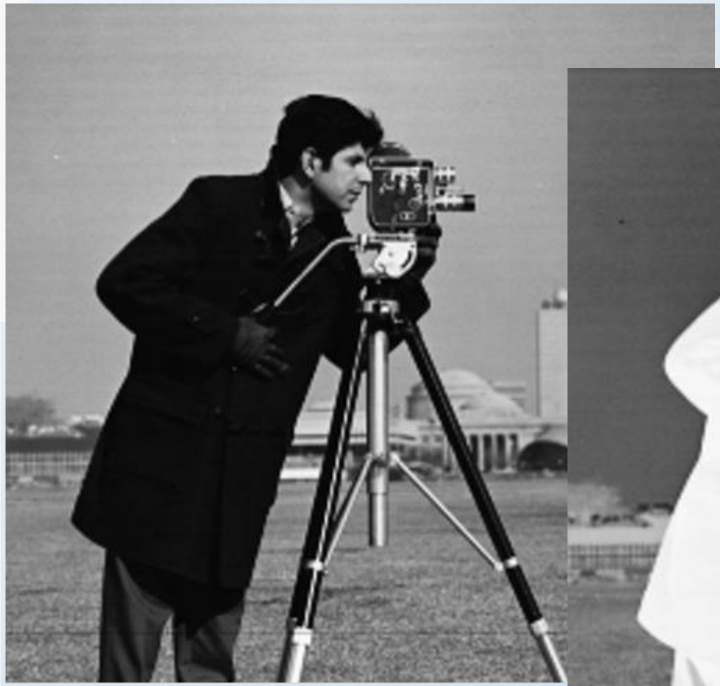
Image Negative

```
import cv2
import numpy as np
# Load the image
img = cv2.imread('/content/Cameraman.tif')
cv2_imshow(img)
# Check the datatype of the image
print(img.dtype)
# Subtract the img from max value(calculated from dtype)
img_neg = 255 - img
# Show the image
cv2_imshow(img_neg)
```

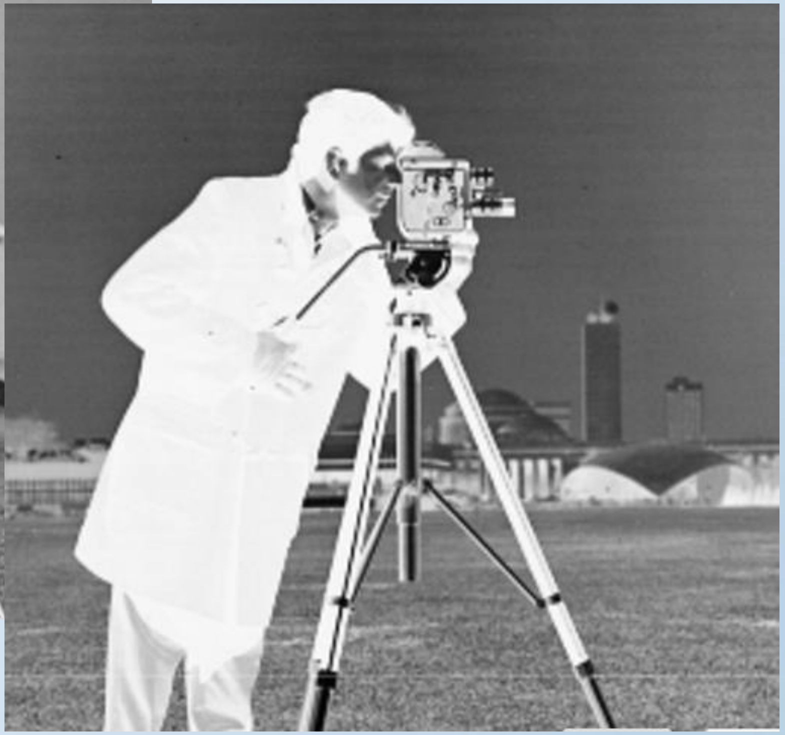
$$s = L - 1 - r$$



Original and Negative Color image



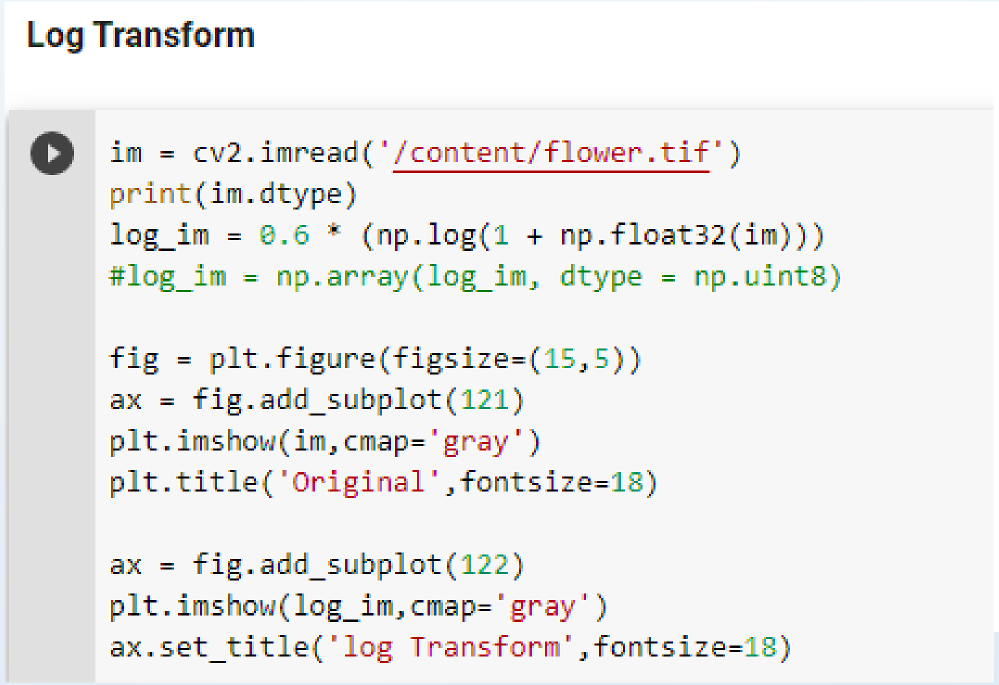
Original Grayscale image



Negative Grayscale image

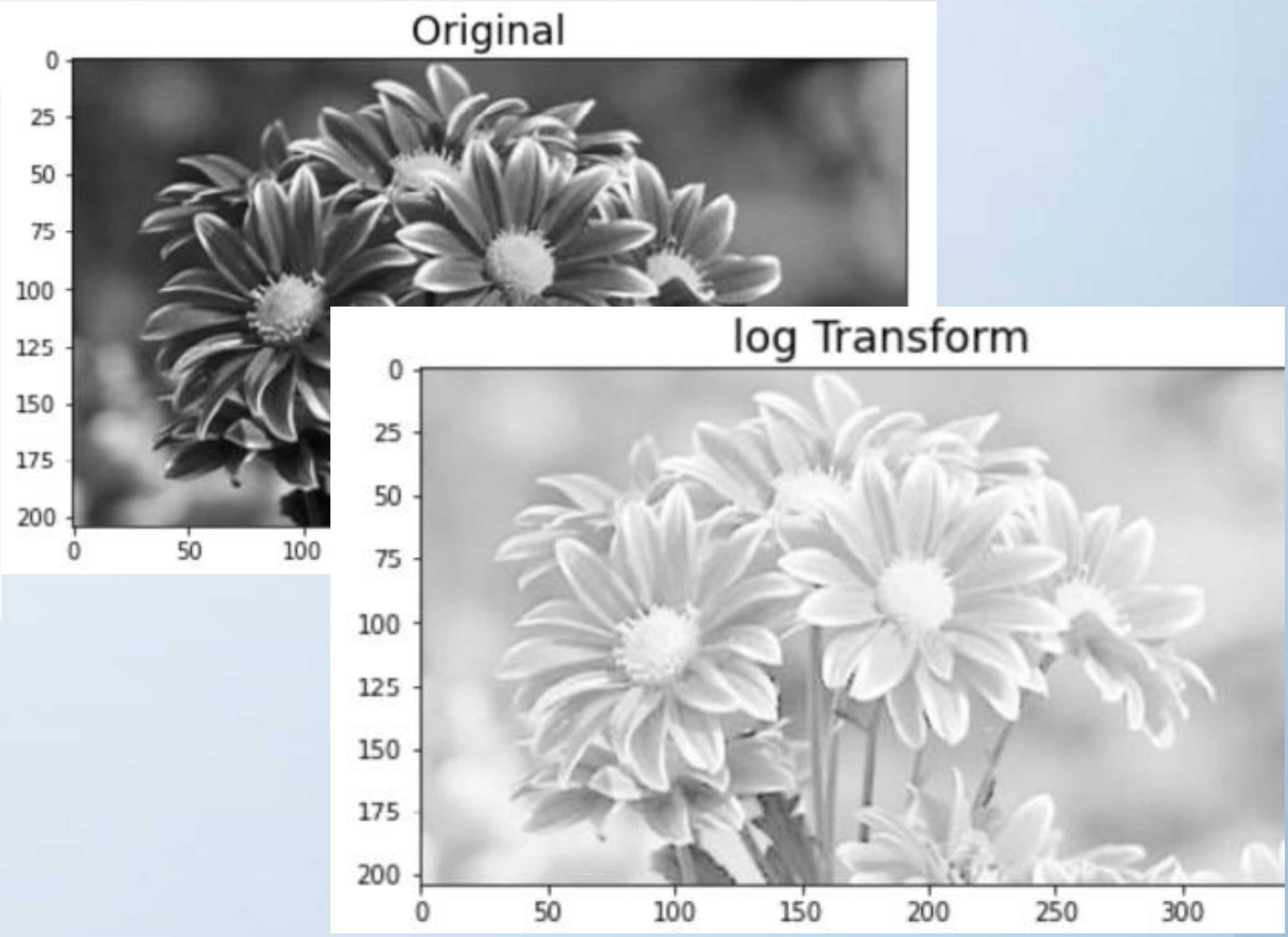
b. Log Transform

For an image having intensity ranging from $[0 \; L-1]$, log transformation is given by $s = c\log(1+r)$ where c is a constant. It compresses the dynamic range of images with a large variation in the pixel value.

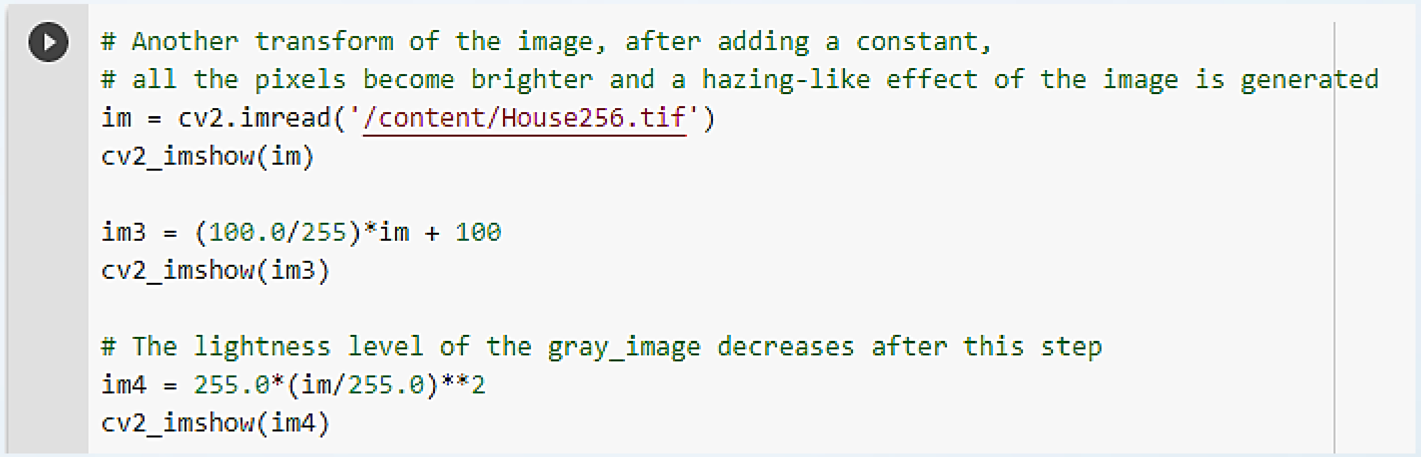


Log Transform

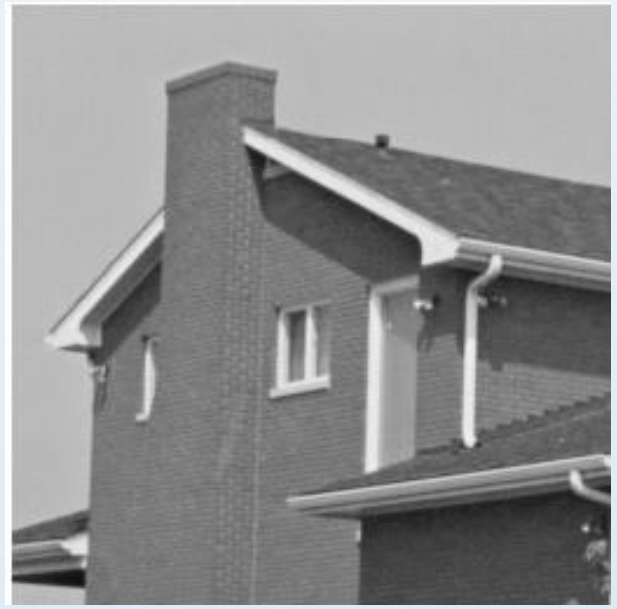
```
im = cv2.imread('/content/flower.tif')
print(im.dtype)
log_im = 0.6 * (np.log(1 + np.float32(im)))
#log_im = np.array(log_im, dtype = np.uint8)

fig = plt.figure(figsize=(15,5))
ax = fig.add_subplot(121)
plt.imshow(im,cmap='gray')
plt.title('Original',fontsize=18)

ax = fig.add_subplot(122)
plt.imshow(log_im,cmap='gray')
ax.set_title('log Transform',fontsize=18)
```



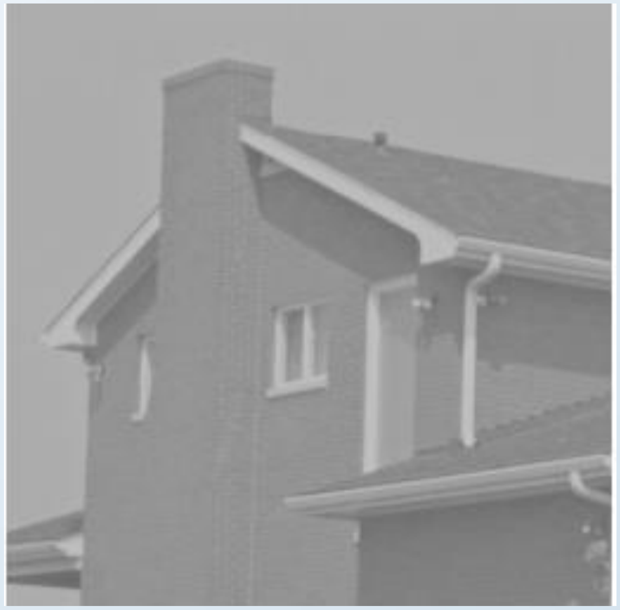
c. Brightness and Linear Transform



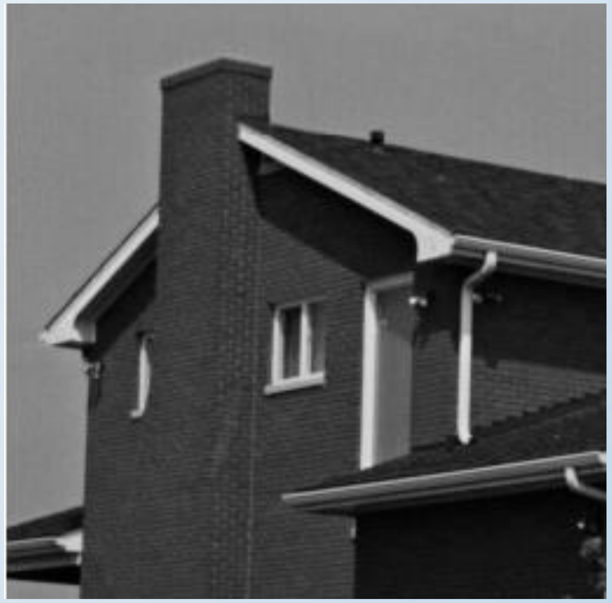
```
# Another transform of the image, after adding a constant,
# all the pixels become brighter and a hazing-like effect of the image is generated
im = cv2.imread('/content/House256.tif')
cv2_imshow(im)

im3 = (100.0/255)*im + 100
cv2_imshow(im3)

# The lightness level of the gray_image decreases after this step
im4 = 255.0*(im/255.0)**2
cv2_imshow(im4)
```



Original Grayscale image



Output image linear transform-1 (addition and brighter pixels)

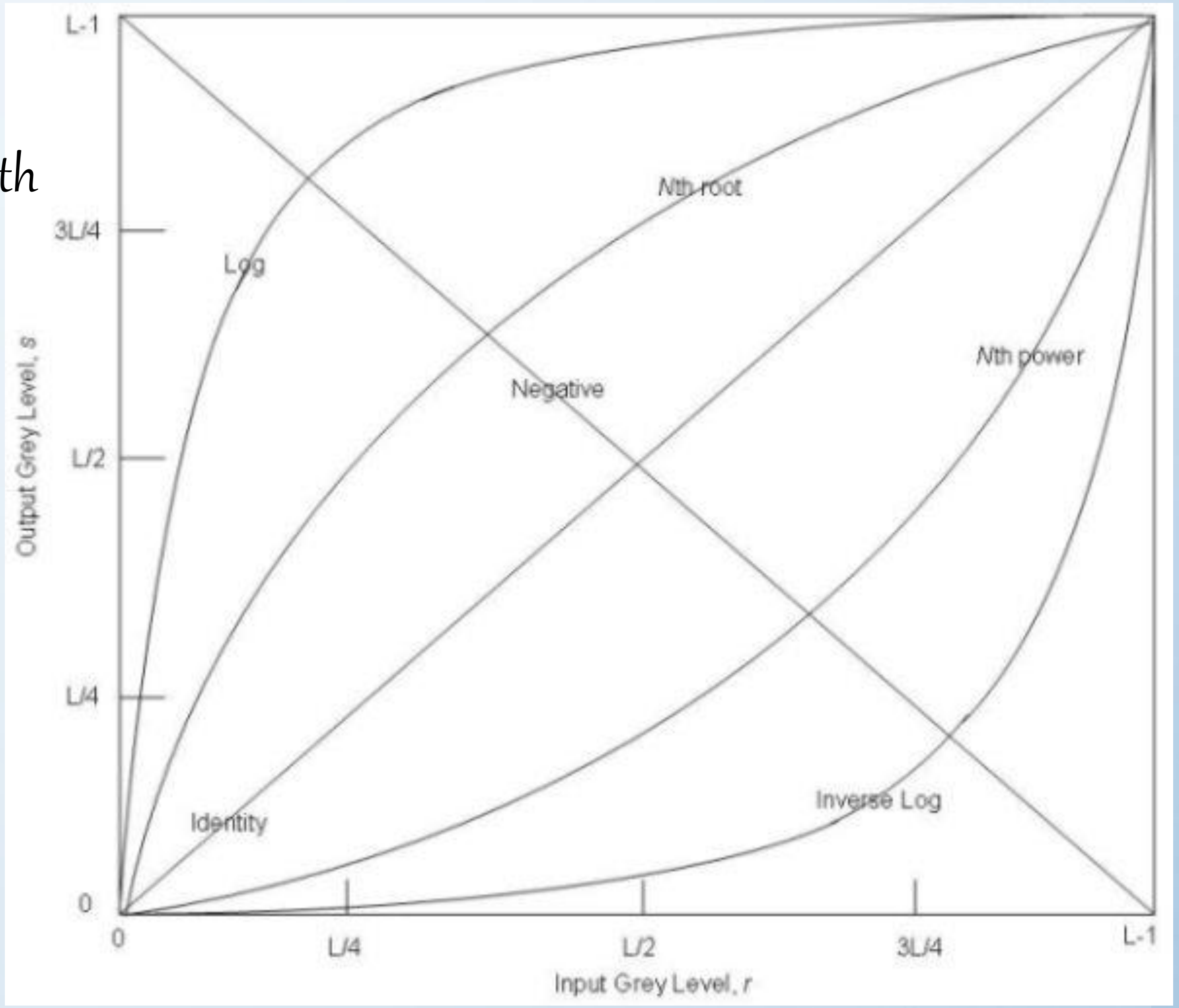


Output image linear transform-2 (Lightness pixels)

d. Power Law

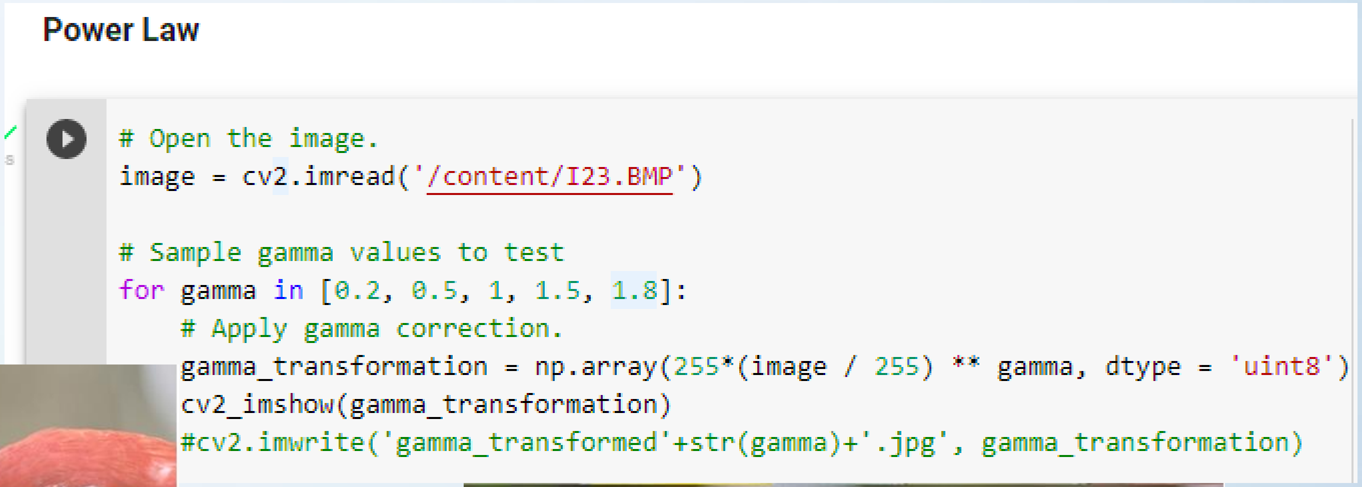
The pixel intensity values can be transformed by finding nth power of the input image, as given below:

$$s = cr^{\gamma}$$



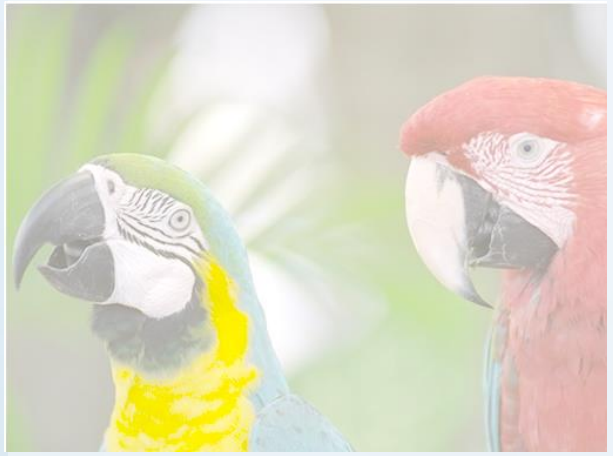
Plot of various (Linear and Non-Linear) transformation function

d. Power Law

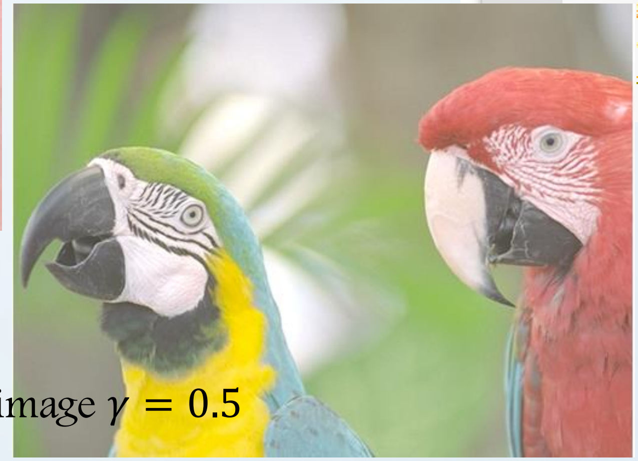


Power Law

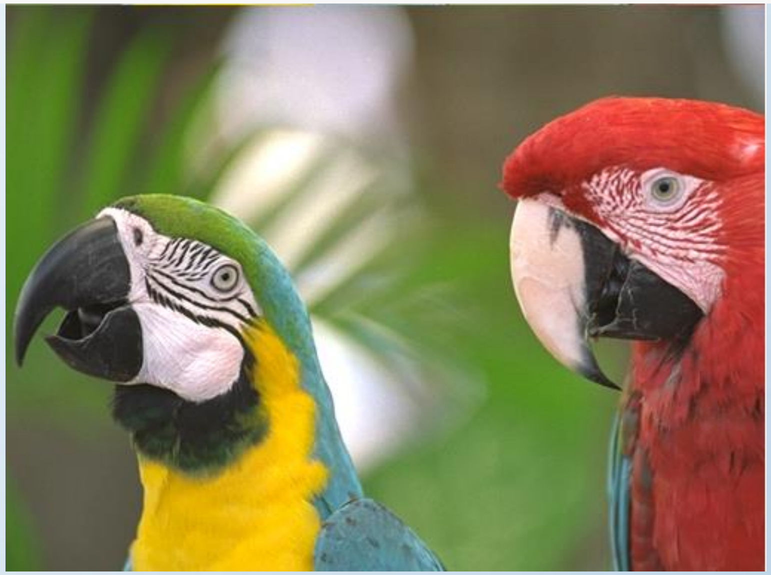
```
# Open the image.
image = cv2.imread('/content/I23.BMP')

# Sample gamma values to test
for gamma in [0.2, 0.5, 1, 1.5, 1.8]:
    # Apply gamma correction.
    gamma_transformation = np.array(255*(image / 255) ** gamma, dtype = 'uint8')
    cv2_imshow(gamma_transformation)
    #cv2.imwrite('gamma_transformed'+str(gamma)+'.jpg', gamma_transformation)
```



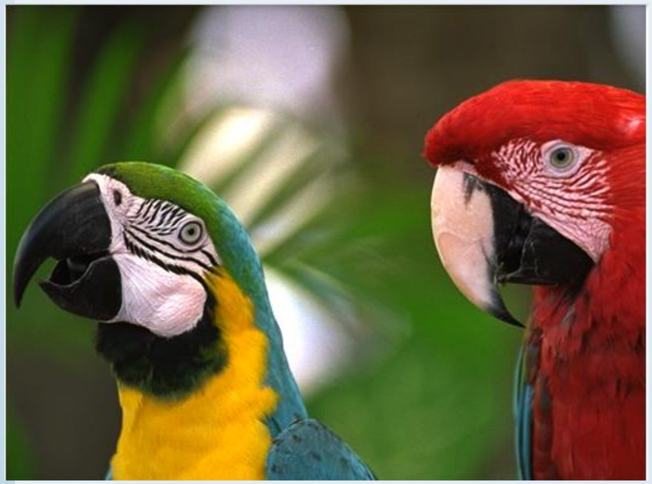
Output image $\gamma = 0.2$



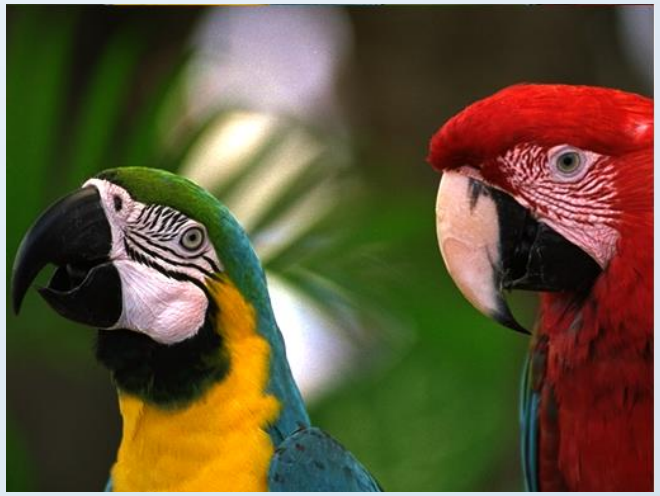
Output image $\gamma = 0.5$



Original image $\gamma = 1$



Output image $\gamma = 1.5$



Output image $\gamma = 1.8$

Image Enhancement by Histogram Processing

Image histogram represents the number of pixels in an image at each different intensity value found in that image.

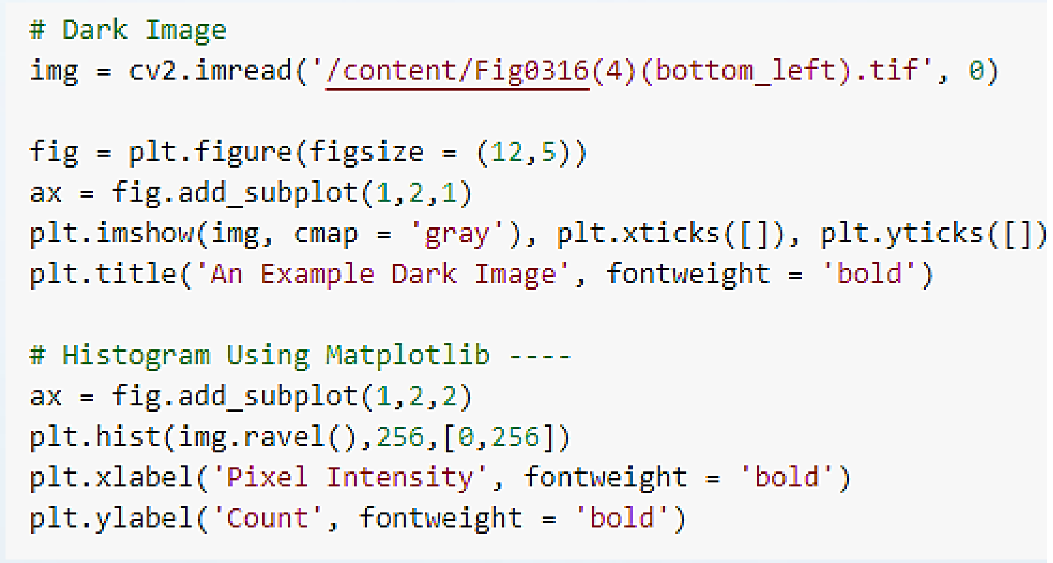
Assuming an image, $f(x, y)$ with intensity values, r_k for $k = 0,1,2, \dots L - 1$ (L represents gray-levels), the normalized histogram can be defined as:

$$p(r_k) = \frac{h(r_k)}{MN}$$

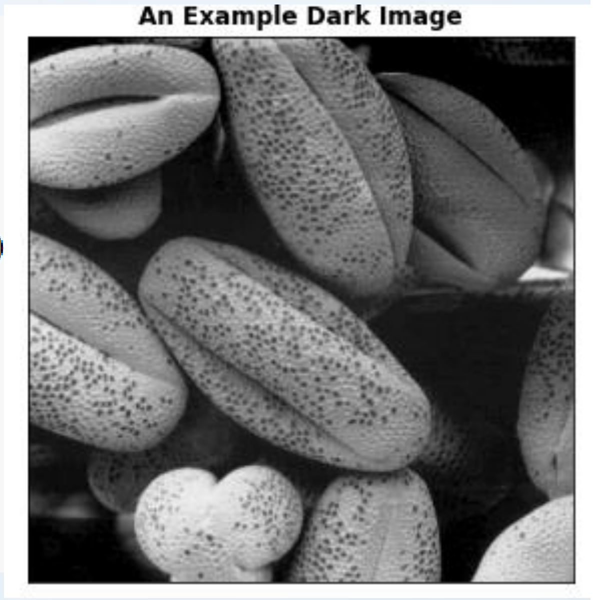
Where, $h_r(r_k) = n_k, \; k = 0, 1, 2, \dots L - 1.$ $h(r_k)$ represents the number of repetitions of each intensity values in the image $f(x, y)$ of dimension $M \times N$ also known as histogram of the image.

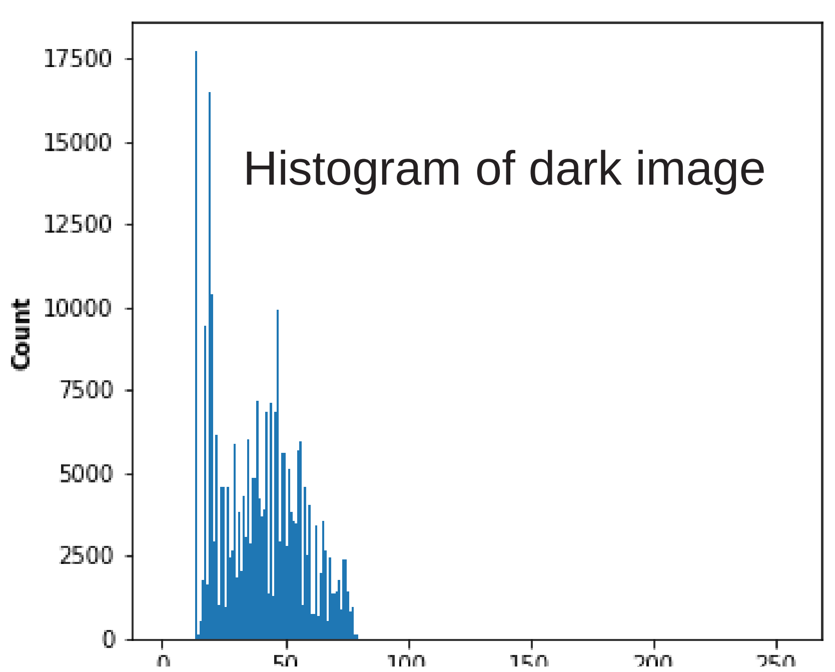
- The normalized histogram represents the probability density function (PDF) of image.

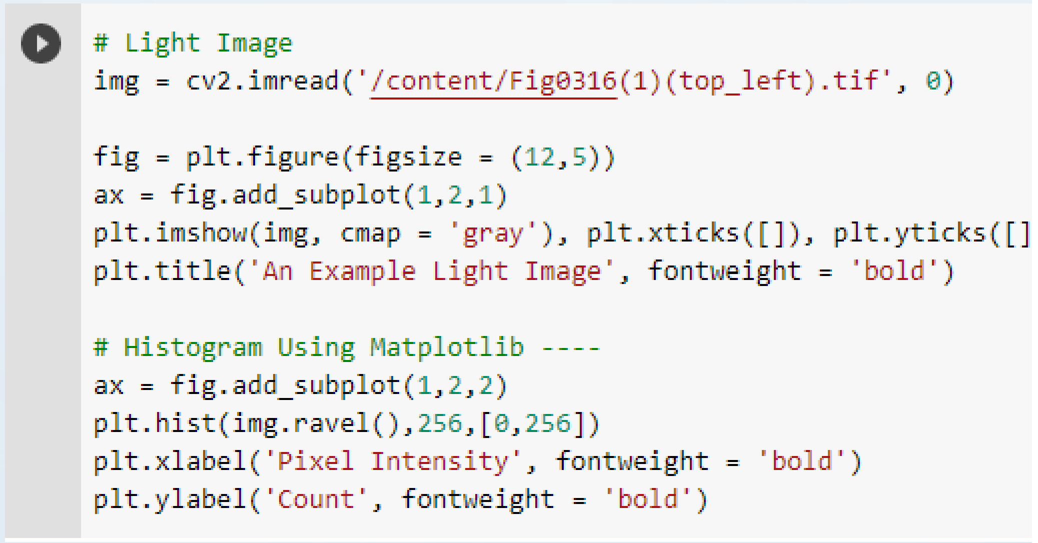
Analyzing Histograms of Monochrome Images



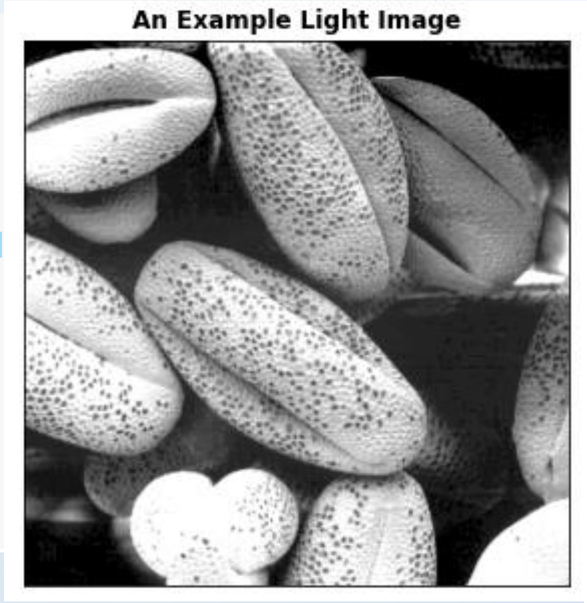
```
# Dark Image
img = cv2.imread('/content/Fig0316(4)(bottom_left).tif', 0)

fig = plt.figure(figsize = (12,5))
ax = fig.add_subplot(1,2,1)
plt.imshow(img, cmap = 'gray'), plt.xticks([]), plt.yticks([])
plt.title('An Example Dark Image', fontweight = 'bold')

# Histogram Using Matplotlib ----
ax = fig.add_subplot(1,2,2)
plt.hist(img.ravel(),256,[0,256])
plt.xlabel('Pixel Intensity', fontweight = 'bold')
plt.ylabel('Count', fontweight = 'bold')
```

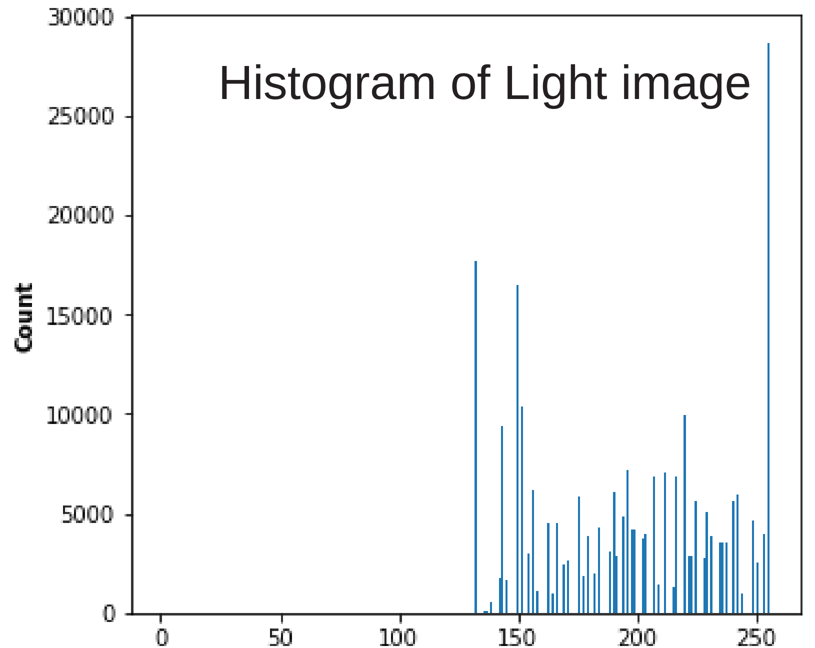


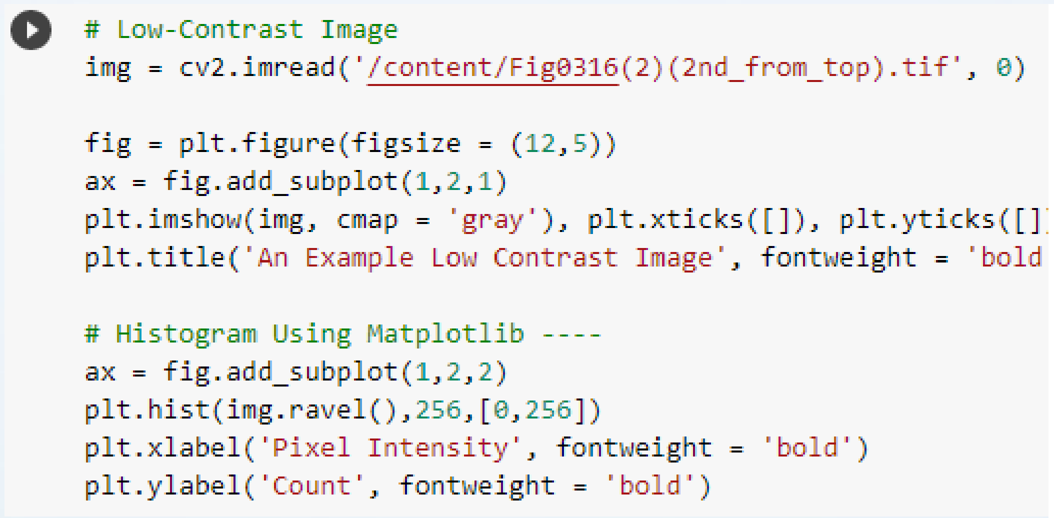




```
# Light Image
img = cv2.imread('/content/Fig0316(1)(top_left).tif', 0)

fig = plt.figure(figsize = (12,5))
ax = fig.add_subplot(1,2,1)
plt.imshow(img, cmap = 'gray'), plt.xticks([]), plt.yticks([]
plt.title('An Example Light Image', fontweight = 'bold')

# Histogram Using Matplotlib ----
ax = fig.add_subplot(1,2,2)
plt.hist(img.ravel(),256,[0,256])
plt.xlabel('Pixel Intensity', fontweight = 'bold')
plt.ylabel('Count', fontweight = 'bold')
```

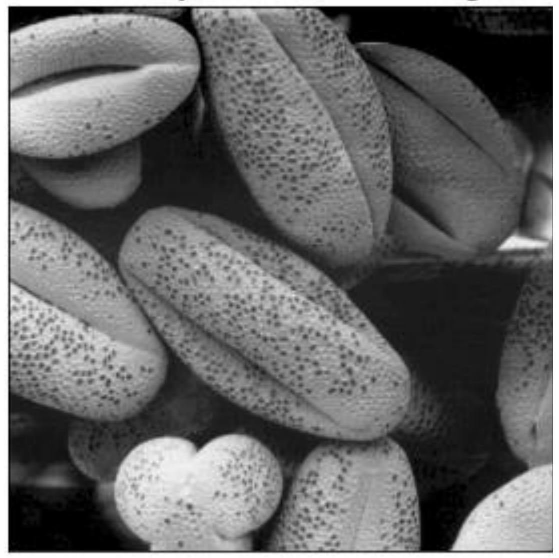


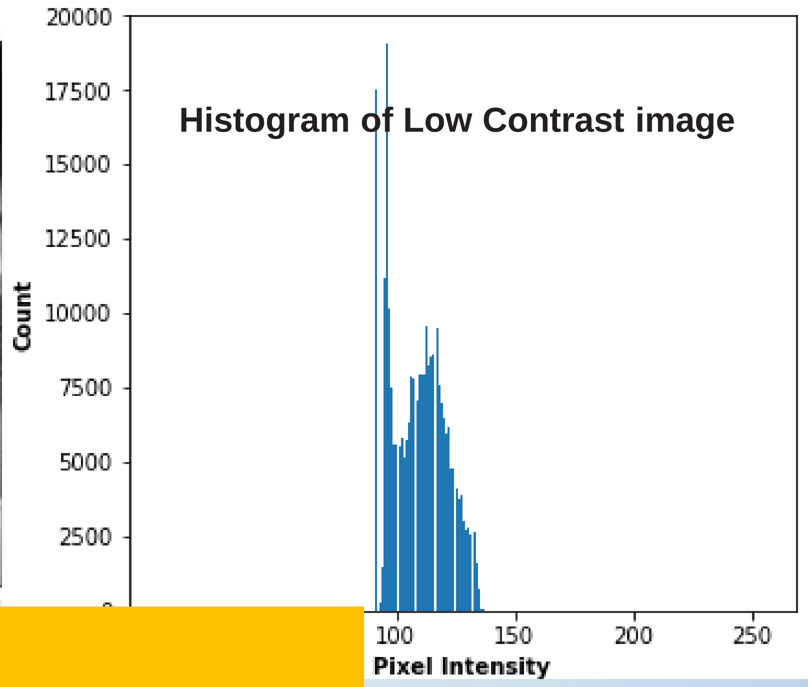




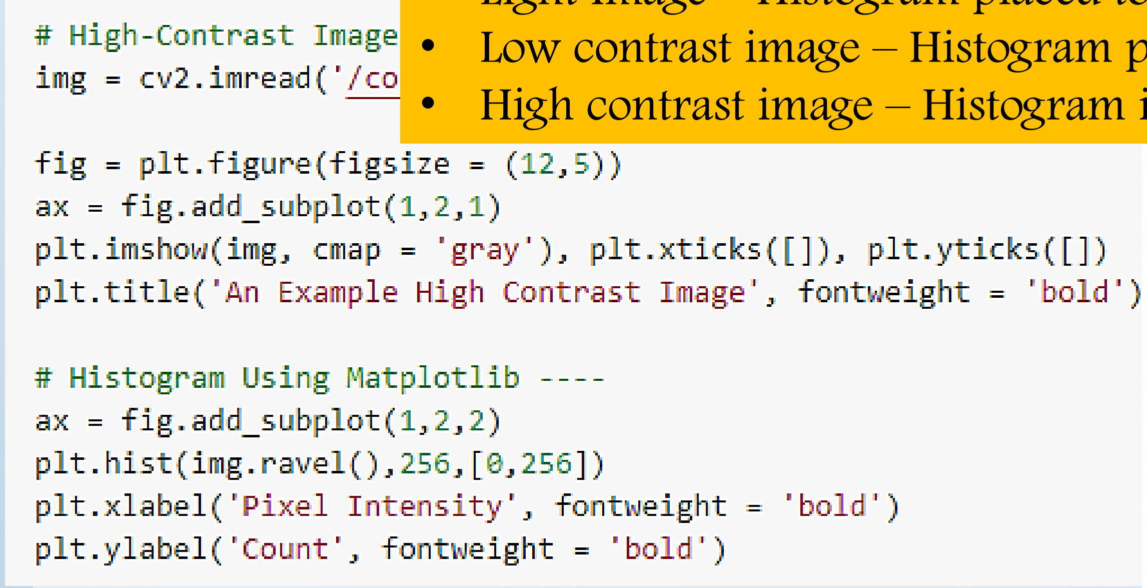
```
# Low-Contrast Image
img = cv2.imread('/content/Fig0316(2)(2nd_from_top).tif', 0)

fig = plt.figure(figsize = (12,5))
ax = fig.add_subplot(1,2,1)
plt.imshow(img, cmap = 'gray'), plt.xticks([]), plt.yticks([]
plt.title('An Example Low Contrast Image', fontweight = 'bold

# Histogram Using Matplotlib ----
ax = fig.add_subplot(1,2,2)
plt.hist(img.ravel(),256,[0,256])
plt.xlabel('Pixel Intensity', fontweight = 'bold')
plt.ylabel('Count', fontweight = 'bold')
```

An Example Low Contrast Image

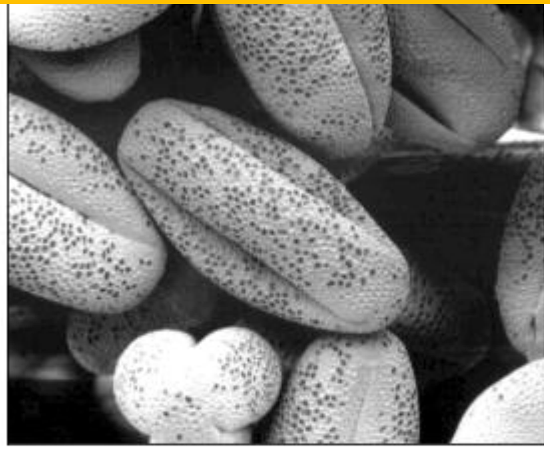


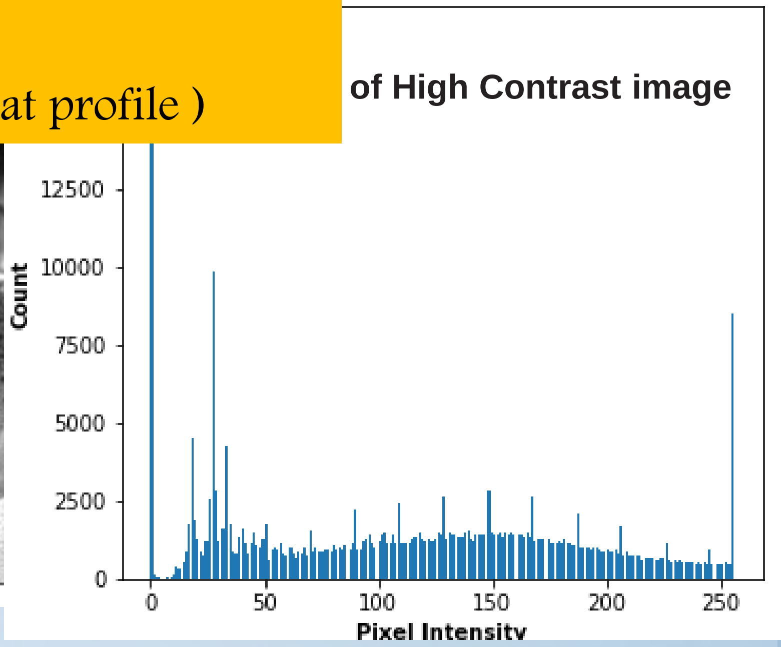


- Dark image – histogram placed at 0
- Light Image – Histogram placed toward 255
- Low contrast image – Histogram placed at center
- High contrast image – Histogram is placed on entire plane (flat profile)



```
# High-Contrast Image
img = cv2.imread('/co

fig = plt.figure(figsize = (12,5))
ax = fig.add_subplot(1,2,1)
plt.imshow(img, cmap = 'gray'), plt.xticks([]), plt.yticks([])
plt.title('An Example High Contrast Image', fontweight = 'bold')

# Histogram Using Matplotlib ----
ax = fig.add_subplot(1,2,2)
plt.hist(img.ravel(),256,[0,256])
plt.xlabel('Pixel Intensity', fontweight = 'bold')
plt.ylabel('Count', fontweight = 'bold')
```





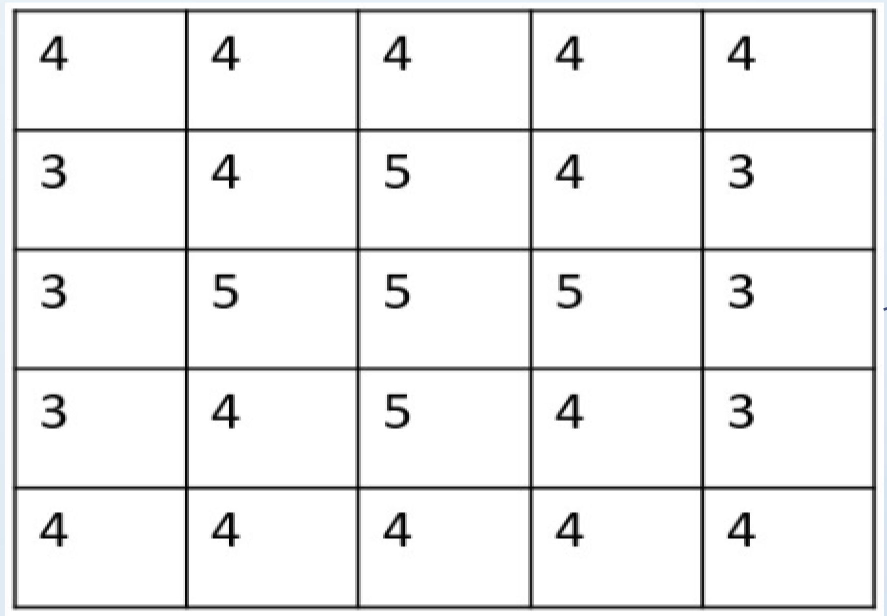
Histogram Equalization & Image Contrast

- Histogram equalization is used increase the global contrast of the input image. It is a process used to obtain flat image intensity profile of the input image.

- The output image is supposed to have high dynamic range. The probability of occurrence of each pixel intensities is expected to be same.

- Thus to obtain a uniform PDF of output image.

Example



4	4	4	4	4
3	4	5	4	3
3	5	5	5	3
3	4	5	4	3
4	4	4	4	4

- A 3-bit input image with minimum pixel intensity 3 and maximum intensity 5.

- For 3-bit image the intensity range 0 to $2^3 - 1 = 7$. Total Gray levels, $L = 8$.

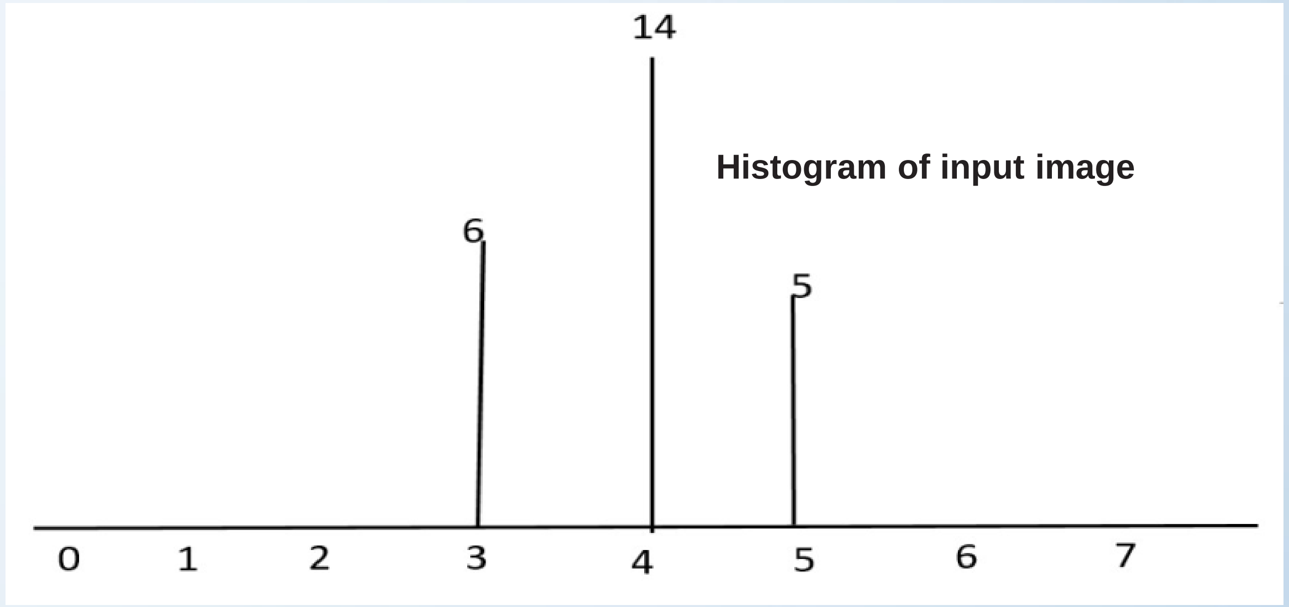
- $M \times N = 5 \times 5$: Input image dimension & $3 \leq r_k \leq 5$.

Example

Gray Levels r_k	0	1	2	3	4	5	6	7
Frequency of occurrence $h(r_k)$	0	0	0	6	14	5	0	0
PDF (Normalized Histogram) $p_r(r_k) = \frac{h(r_k)}{MN}$	$\frac{0}{25} = 0$	$\frac{0}{25} = 0$	$\frac{0}{25} = 0$	$\frac{6}{25} = .24$	$\frac{14}{25} = .56$	$\frac{5}{25} = .2$	$\frac{0}{25} = 0$	$\frac{0}{25} = 0$
Cumulative Distribution function (CDF) $P_r(r) = \sum_{r_k \leq r} p_r(r_k)$	0	0	0	0.24	0.8	1	1	1
$P_r(r) \times 7$ (max. possible gray level)	0	0	0	1.68	5.6	7	7	7
Intensity of Histogram Equalized image	0	0	0	2	6	7	7	7

4	4	4	4	4
3	4	5	4	3
3	5	5	5	3
3	4	5	4	3
4	4	4	4	4

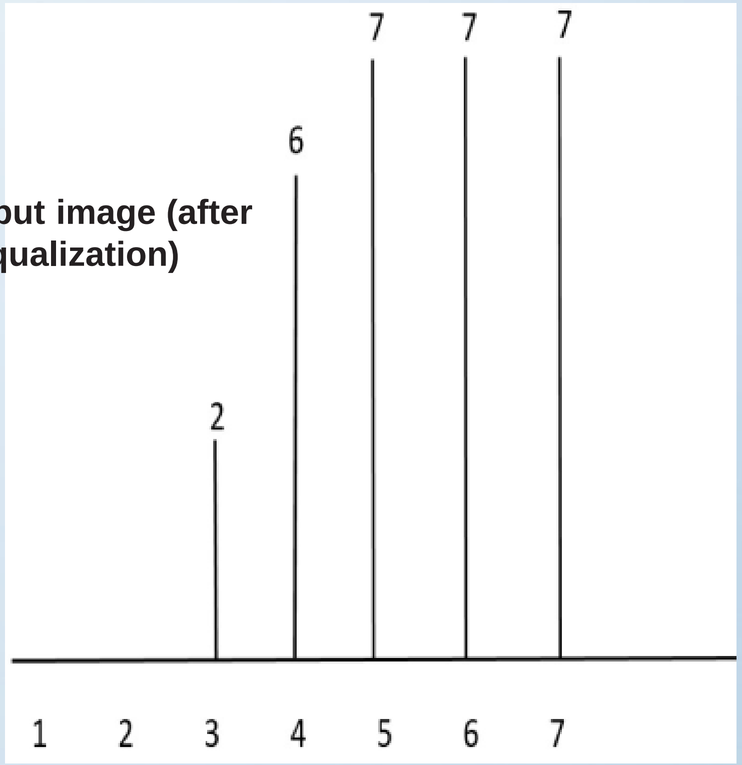
Input Image



Histogram of input image

6	6	6	6	6
2	6	7	6	2
2	7	7	7	2
2	6	7	6	2
6	6	6	6	6

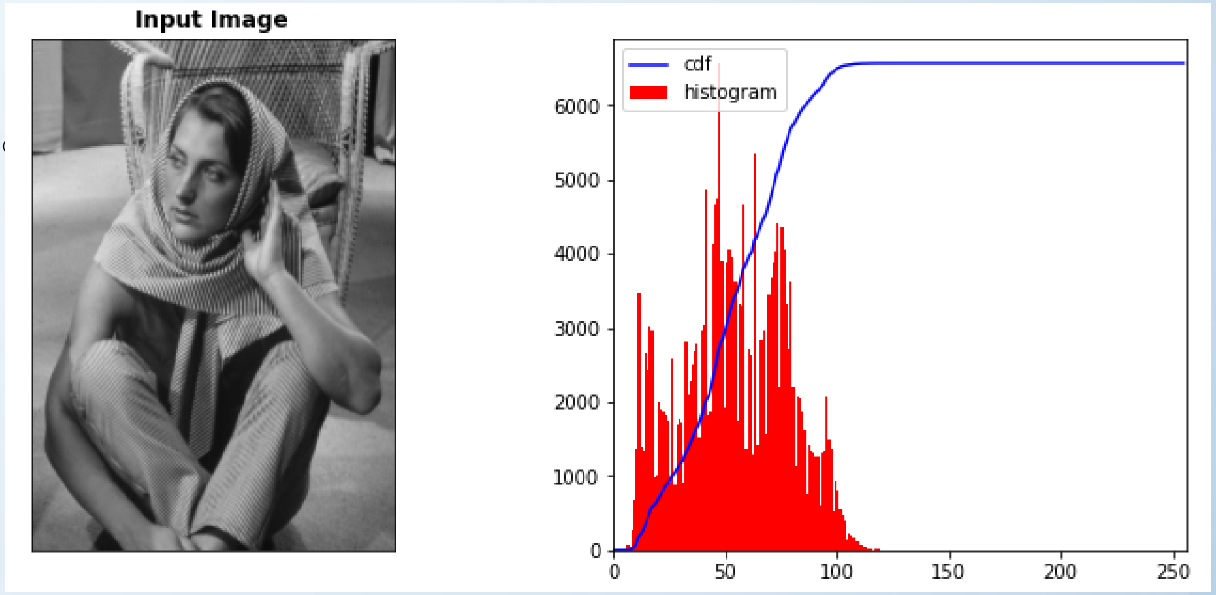
Histogram Equalized Image

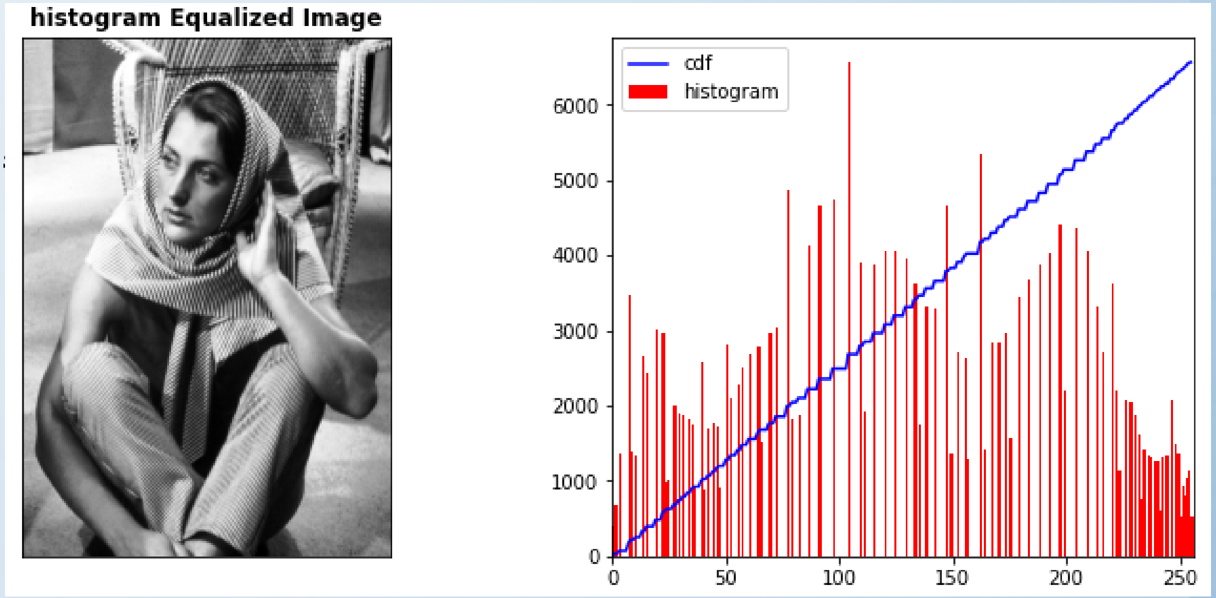


Histogram of output image (after histogram Equalization)

Python (OpenCV) implementation

```
img_gray = cv2.imread('/content/barbara_Dark.tif', 0)

fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(1,2,1)
plt.imshow(img_gray, cmap = 'gray'), plt.xticks([]), plt.yti
plt.title('Input Image', fontweight = 'bold')

hist,bins = np.histogram(img_gray.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
ax = fig.add_subplot(1,2,2)
plt.plot(cdf_normalized, color = 'b')
plt.hist(img_gray.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()

# Histogram Equalization
HE_img = cv2.equalizeHist(np.uint8(img_gray))
fig = plt.figure(figsize = (12, 5))
ax = fig.add_subplot(1,2,1)
plt.imshow(HE_img, cmap = 'gray'), plt.xticks([]), plt.yticks
plt.title('histogram Equalized Image', fontweight = 'bold')

hist,bins = np.histogram(HE_img.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
ax = fig.add_subplot(1,2,2)
plt.plot(cdf_normalized, color = 'b')
plt.hist(HE_img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
```


Input Image
cdf
histogram


histogram Equalized Image
cdf
histogram