Image Enhancement and Filter in Spatial Domain

UNIT 2 | Yuba Raj Devkota

8 hrs | NCCS

Unit 2	**Image Enhancement and Filter in Spatial Domain**	**Teaching Hours (8)**
Basic Gray Level Transformations	Point operations, Contrast stretching, clipping and thresholding, digital negative, intensity level slicing, log transformation, power log transformation, bit plane slicing	2 hrs.
Histogram Processing	Unnormalized and Normalized Histogram, Histogram Equalization, Use of Histogram Statistics for Image Enhancement	1 hr
Spatial operations	Basics of Spatial Filtering, Linear filters, Spatial Low pass smoothing filters, Averaging, Weighted Averaging, Non-Linear filters, Median filter, Maximum and Minimum filters, High pass sharpening filters, High boost filter, high frequency emphasis filter, Gradient based filters, Robert Cross Gradient Operators, Prewitt filters, Sobel filters, Second Derivative filters, Laplacian filters	4 hrs.
Magnification	Magnification by replication and interpolation	1 hr

Image Enhancement

Image enhancement is the process of improving the visual appearance of an image or making it more suitable for analysis by humans or machines.
It does **not add new information**, but highlights or suppresses certain features (e.g., contrast, edges, brightness).

Examples:

- Increasing contrast of a dark image
- Sharpening blurred edges
- Removing noise

Spatial domain enhancement works directly on pixels and is simpler.

Frequency domain enhancement works on frequency components and is powerful for noise removal and sharpening.

Image Enhancement in Spatial Domain

In **spatial domain enhancement**, operations are performed **directly on image pixels**.

Basic idea:

$$g(x, y) = T[f(x, y)]$$

where

- $f(x, y)$ = original image
- $g(x, y)$ = enhanced image
- T = transformation on pixel values

Common techniques:

- Image negative
- Log & power-law (gamma) transformation
- Contrast stretching
- Histogram equalization
- Spatial filtering (smoothing, sharpening)

Key point:

✔ Direct manipulation of pixel values

Image Enhancement in Frequency Domain

In **frequency domain enhancement**, the image is first transformed into frequency components using a transform (e.g., **Fourier Transform**).

Basic steps:

1. Apply Fourier Transform
2. Modify frequency components
3. Apply Inverse Transform

Common techniques:

- Low-pass filtering (smoothing)
- High-pass filtering (edge enhancement)
- Band-pass / Band-stop filtering

Key point:

✔ Manipulates image based on **rate of intensity changes (frequencies)**

Aspect	Spatial Domain	Frequency Domain
Operation	On pixels directly	On transformed image
Domain	Image space (x, y)	Frequency space (u, v)
Complexity	Simple, intuitive	More complex
Suitable for	Basic enhancement	Periodic noise, global features
Examples	Histogram equalization	Low-pass, High-pass filters

Basic Gray Level Transformations

Gray level transformations are **point processing techniques** where each pixel value is transformed independently to enhance an image.

General form:

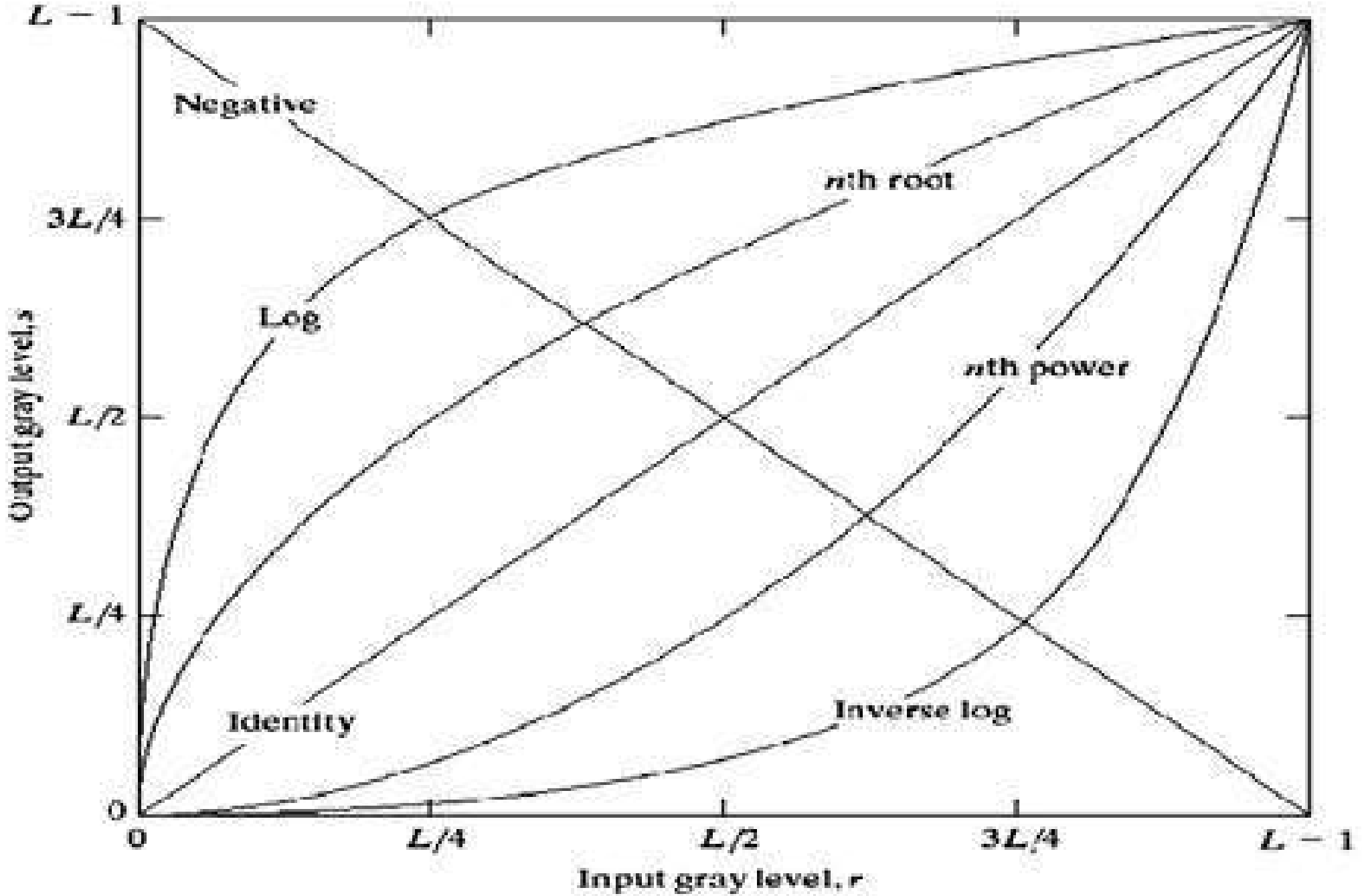
$$s = T(r)$$

where

- r = input gray level
- s = output gray level

There are three basic gray level transformation.

1. Linear
2. Logarithmic
3. Power – law



0 represents black
L-1 represents White

1. Linear transformation (identity & Negative)

Linear transformations change pixel values **linearly**. The most common ones are **identity**, **negative**, and **contrast stretching**.

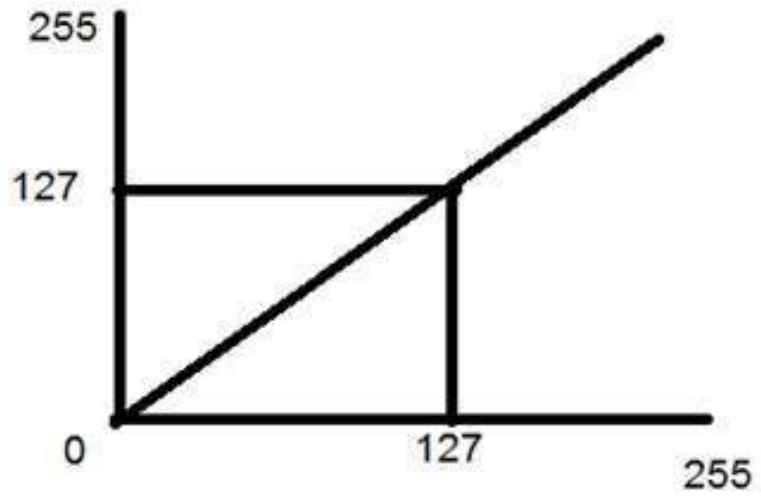
(a) Identity Transformation

$$s = r$$

✔ Image remains unchanged.

Example:

If pixel value = 120 → Output = 120



Negative transformation

(b) Image Negative

$$s = (L - 1) - r$$

where L = number of gray levels (for 8-bit, $L = 256$)

Purpose:

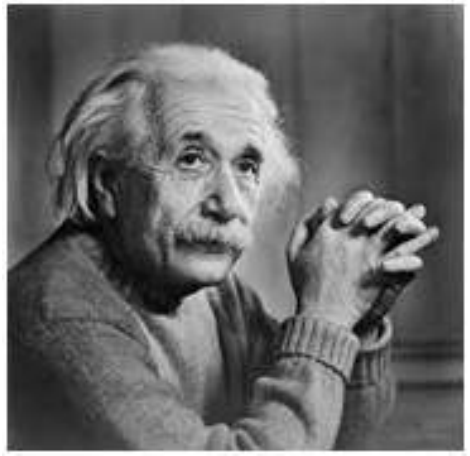
- Highlights details in dark regions
- Used in medical images (X-rays)

Example (8-bit image):

If $r = 50$

$$s = 255 - 50 = 205$$

Input Image



Output Image

Lab 02: Negative Linear Transformation

```
clear all; % clear all variables
close all; % close all figures
clc;       % clear command window
% import image package
pkg load image;
% read image
img = imread("mountain.png");
figure
imshow(img);
% convert image into gray and then
grayscale_img = rgb2gray(img);
% show grayscale image
figure
imshow(grayscale_img);
title("grayscale image");
imwrite(grayscale_img, "original.jp
# calculate negative of the image
output = 255-grayscale_img;
% show output image
figure
imshow(uint8(output));
title("output image");
imwrite(uint8(output), "negative_ta
```

Figure 2

File Edit Tools

(163.03, 161.43)

grayscale image

output image

75, 166.33)

2. Logarithmic Transformation

$$s = c \log(1 + r)$$

Purpose:

- Enhances **low-intensity (dark)** pixels
- Compresses **high-intensity** values

Applications:

- Fourier spectrum visualization
- Images with large dynamic range

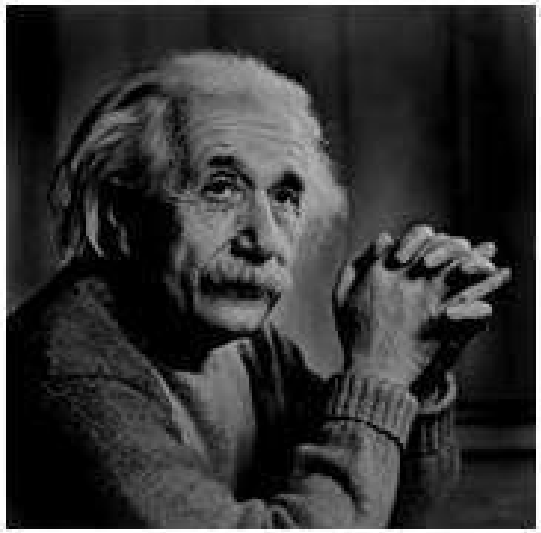
Example:

Let $r = 10, c = 1$

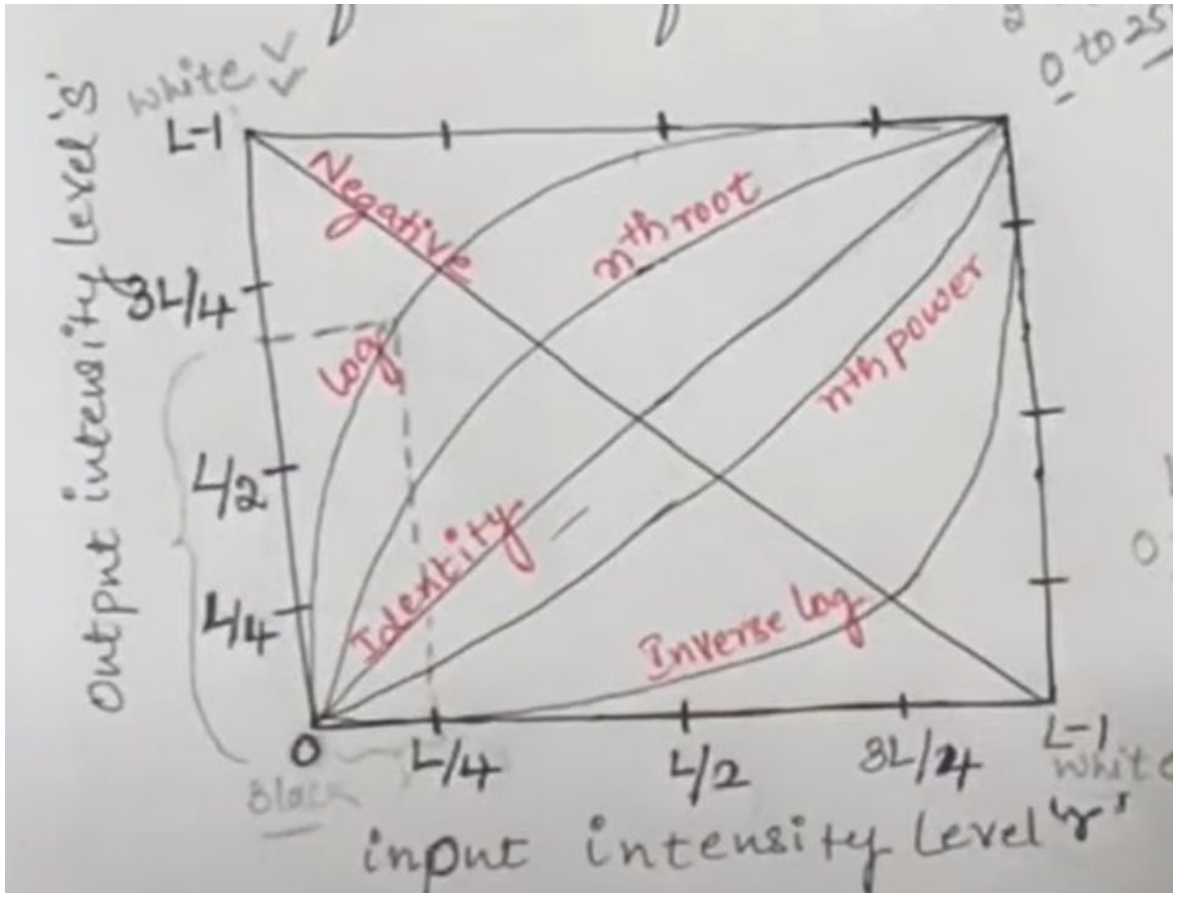
$$s = \log(11) \approx 1.04$$

✔ Dark regions become more visible.

Input Image



Log Tranform Image



Log:

For L/4 Input, output is nearly 3L/4, which means that for less amount of image as input, more amount of image is the output. That's why it results in high contrast image.

Inverse Log:

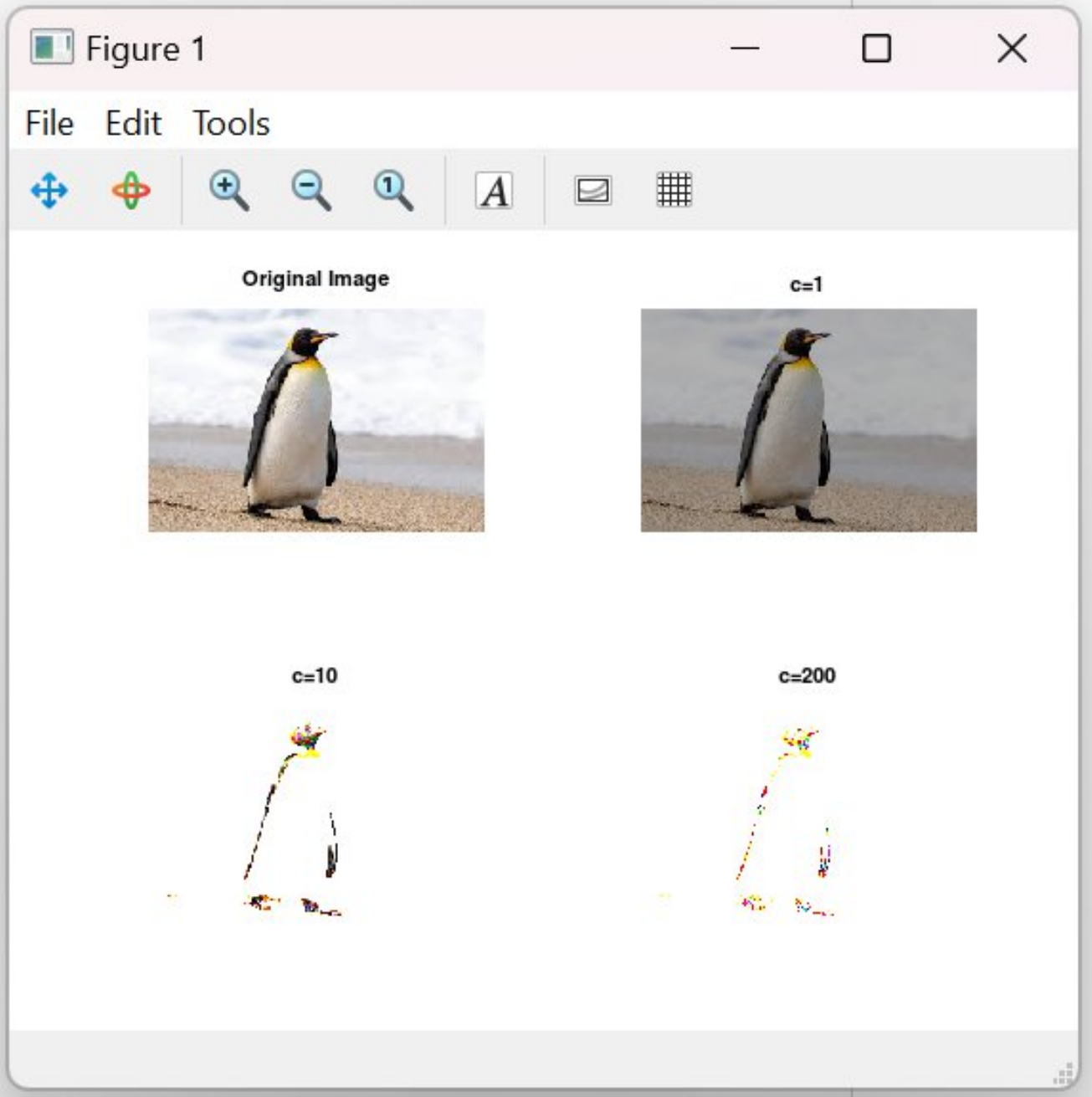
Similarly, for nearly 3L/4 input image, output is L/4 i.e. for high amount of image, less amount of image is the output. So it will produce less contrast image.

Lab 03:
Log Transformation

```
clear all; % clear all variables
close all; % close all figures
clc;       % clear command window
a=imread('penguines.png');
subplot(2,2,1);
imshow(a);
title 'Original Image';
b=im2double(a);
s=(1*log(1+b))*256;
s1=uint8(s);
subplot(2,2,2);
imshow(s1);
title 'c=1';

sp=(10*log(1+b))*256;
s2=uint8(sp);
subplot(2,2,3);
imshow(s2);
title 'c=10';


sp2=(200*log(1+b))*256;
s3=uint8(sp2);
subplot(2,2,4);
imshow(s3);
title 'c=200';
```



3. Power Law (Gamma) Transformation

$$s = c\,r^{\gamma}$$

Purpose:

- Controls image brightness
- Used in **gamma correction**

Case 1: $\gamma < 1$

✔ Image becomes **brighter**

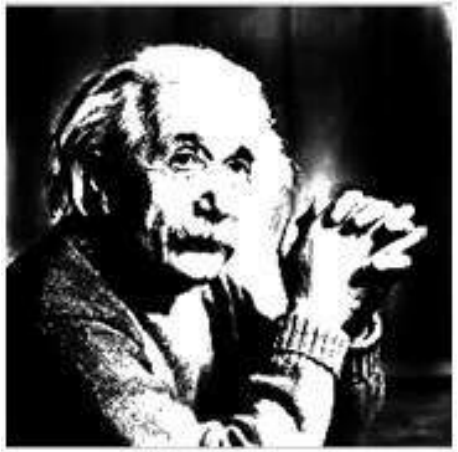
Case 2: $\gamma > 1$

✔ Image becomes **darker**

Real-life use:

- TV, monitors, mobile displays

Gamma = 10



Gamma = 8

Gamma = 6

Example:

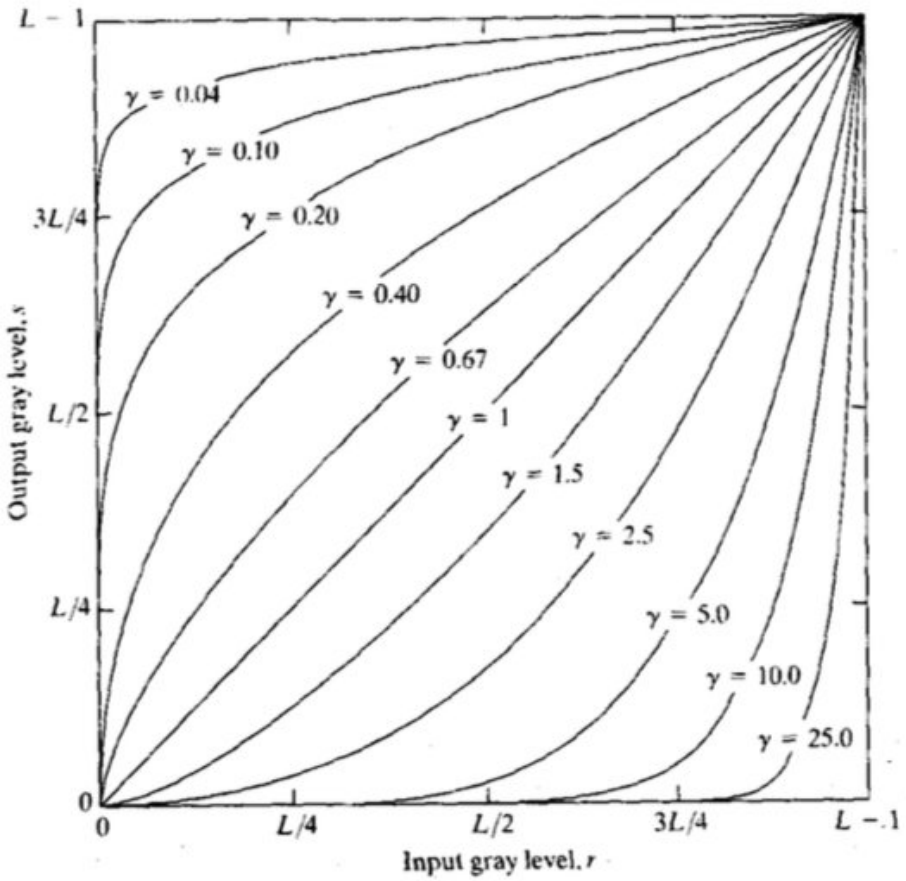
Let $r = 0.5, c = 1$

- If $\gamma = 0.5$:

$$s = (0.5)^{0.5} \approx 0.71 \quad \text{(brighter)}$$

- If $\gamma = 2$:

$$s = (0.5)^{2} = 0.25 \quad \text{(darker)}$$



This transformation is similar to log transformation but different values of gamma

For same input L/4, the output will be different depending upon the value of gamma.

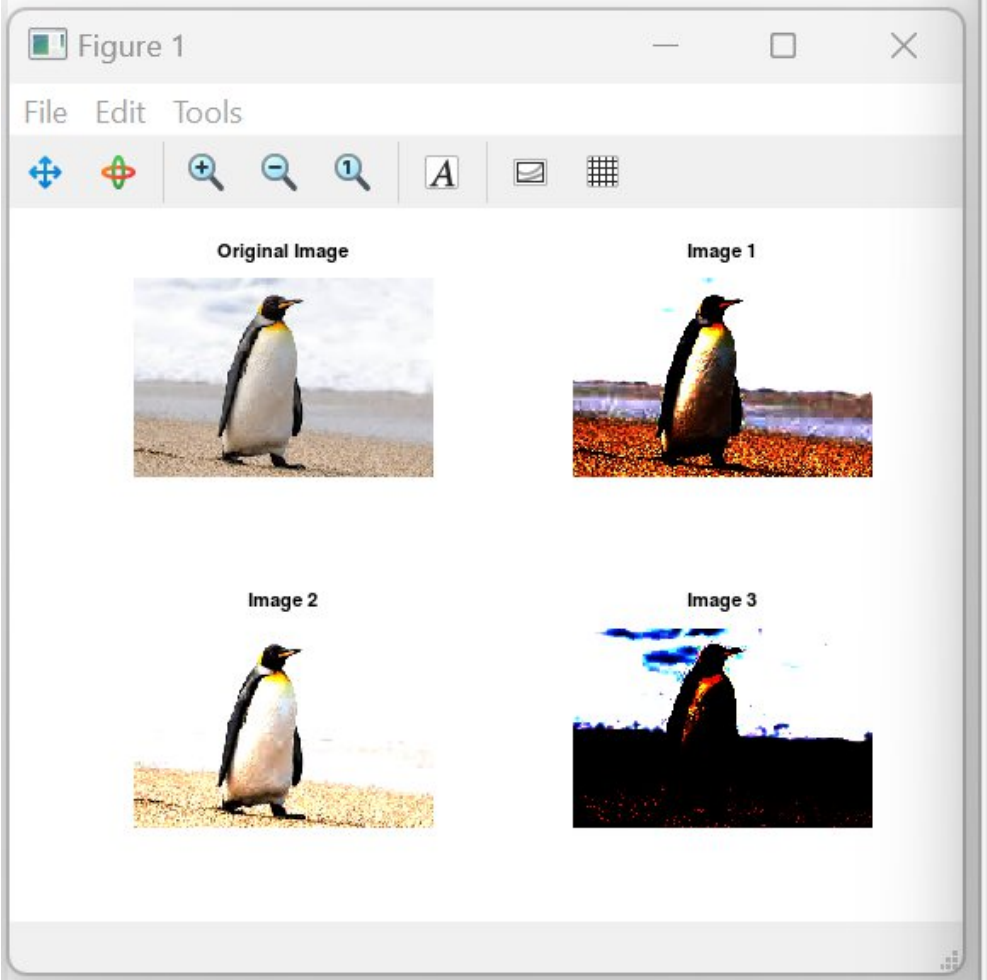
Transformation	Formula	Main Use
Linear	$s = ar + b$	Contrast improvement
Logarithmic	$s = c\log(1 + r)$	Enhance dark areas
Power Law	$s = cr^{\gamma}$	Brightness control

Lab 04:
Gamma Transformation

Courses\Image Processing\lab_ip

Command Window

```
enter value a10
enter value gamma10
enter value a2
enter value gamma2
enter value a50
enter value gamma50
>>
```



File Edit View Debug Run Help

ge_filter.m | convolution_filters.m | gamma_tranformations.m

```
1  clear all; % clear all variables
2  close all; % close all figures
3  clc;       % clear command window
4  % import image package
5  pkg load image;
6  % read image
7  a=imread('penguines.png');
8  subplot(2,2,1);
9  imshow(a);
10 title 'Original Image';
11 b=im2double(a);
12 a1=input('enter value a');
13 ga1=input('enter value gamma');
14 s=(a1*(b.^ga1))*256;
15 s1=uint8(s);
16 subplot(2,2,2);
17 imshow(s1);
18 title 'Image 1';
19
20 a2=input('enter value a');
21 ga2=input('enter value gamma');
22 sp=(a2*(b.^ga2))*256;
23 s2=uint8(sp);
24 subplot(2,2,3);
25 imshow(s2);
26 title 'Image 2';
27
28
29 a3=input('enter value a');
30 ga3=input('enter value gamma');
31 sp2=(a3*(b.^ga3))*256;
32 s3=uint8(sp2);
33 subplot(2,2,4);
34 imshow(s3);
35 title 'Image 3';
```

Piecewise Linear Transformations

Piecewise Linear Transformations are gray level transformations where the transformation function is defined by **multiple straight-line segments** instead of a single line.
They are used to **selectively enhance specific ranges of gray levels**.

✔ Very useful when only certain intensity ranges contain important information.

1. Contrast Stretching

Contrast Stretching improves image contrast by **expanding a narrow range of gray levels** to a wider range.

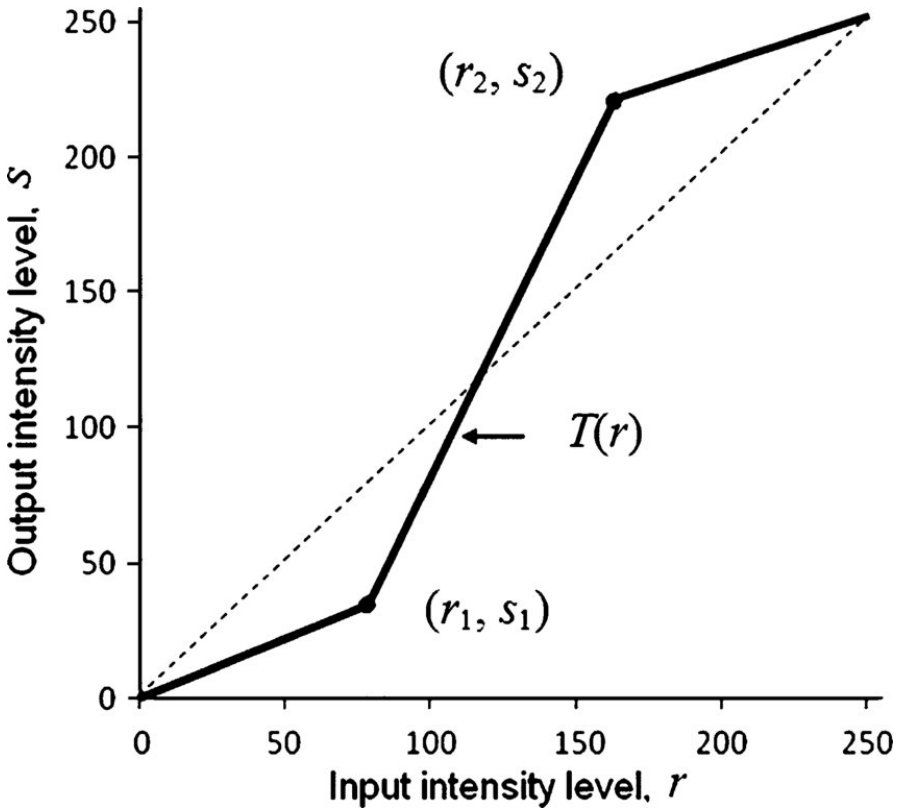
Concept

Low-contrast images often have pixel values clustered in a small range. Contrast stretching spreads them over the full range.

Transformation

Defined by two break points (r_1, s_1) and (r_2, s_2).

- Gray levels below r_1 → dark
- Between r_1 and r_2 → stretched
- Above r_2 → bright

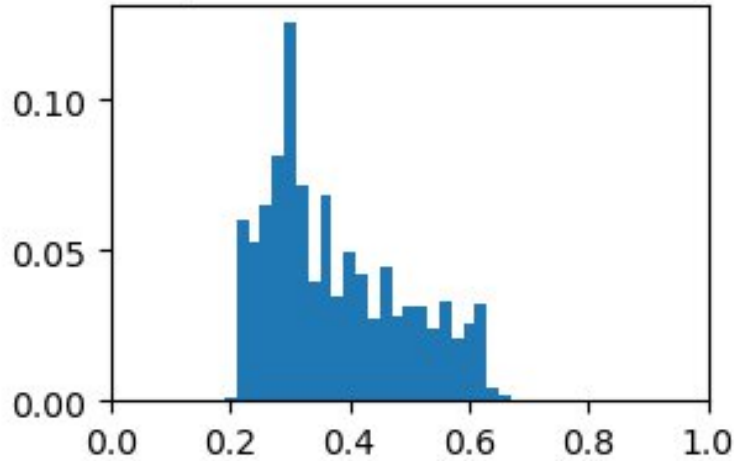


Example: If an image has gray levels mostly between **80 and 150**, stretch them to **0–255**.

✔ Used in satellite images and medical imaging.

Low contrast orginal

Histogram of low contrast image



Contrast Stretched

Histogram of contrast stretched image

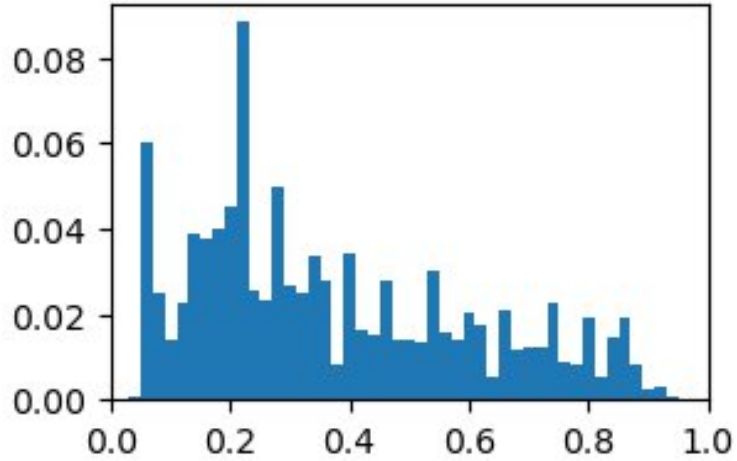
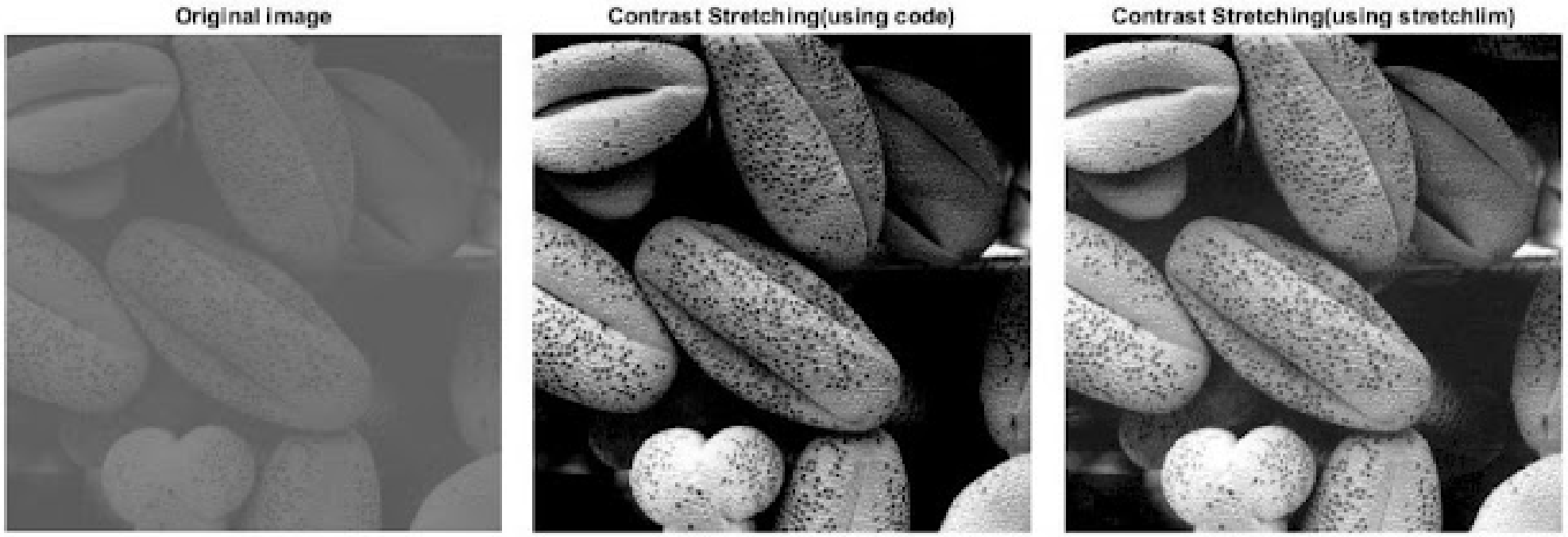


Fig. 2(a). Original Image Fig. 2(b). Contrast Stretched Image

730 ×

Almost invisible image is stretched
& in next image, thresholding

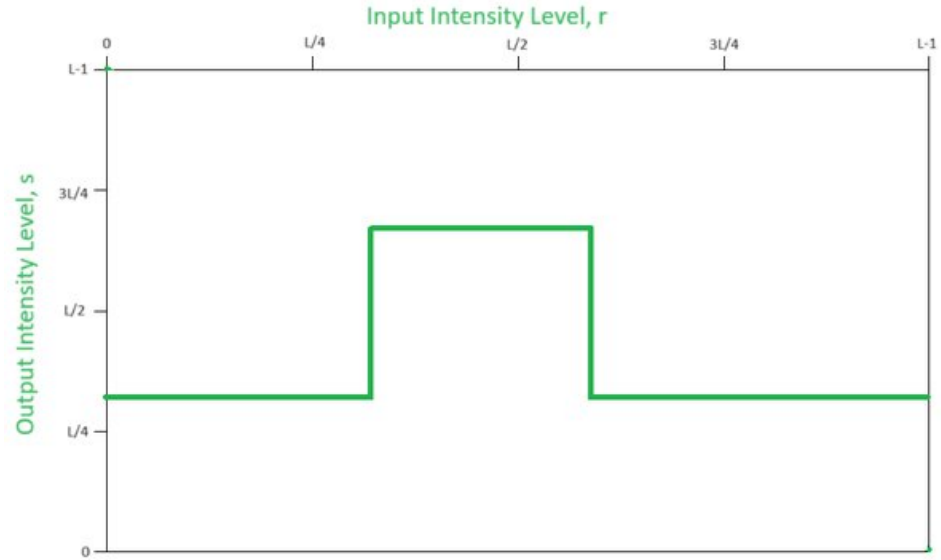


2. Gray Level Slicing

Gray Level Slicing highlights a **specific range of gray levels**, suppressing others.

Purpose

- Emphasize important features
- Ignore irrelevant background

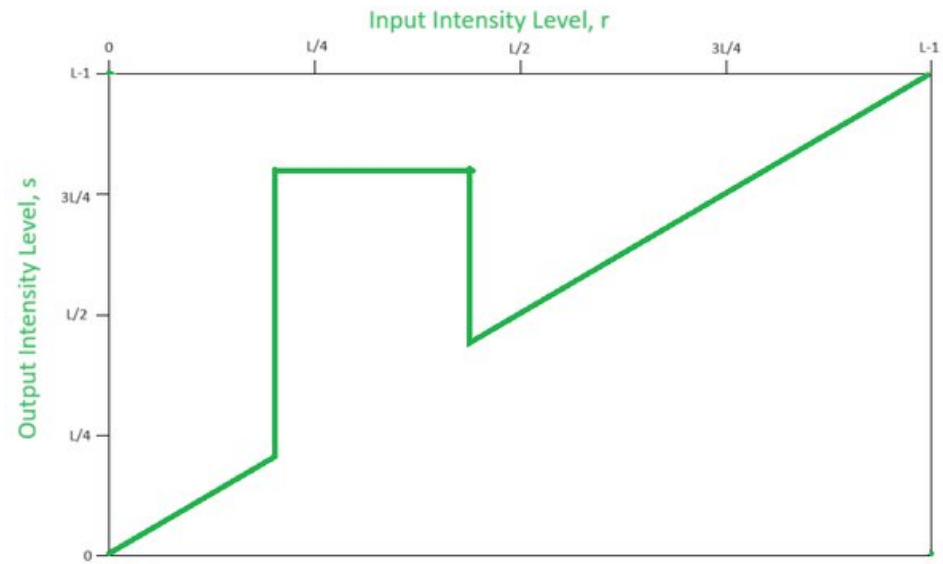


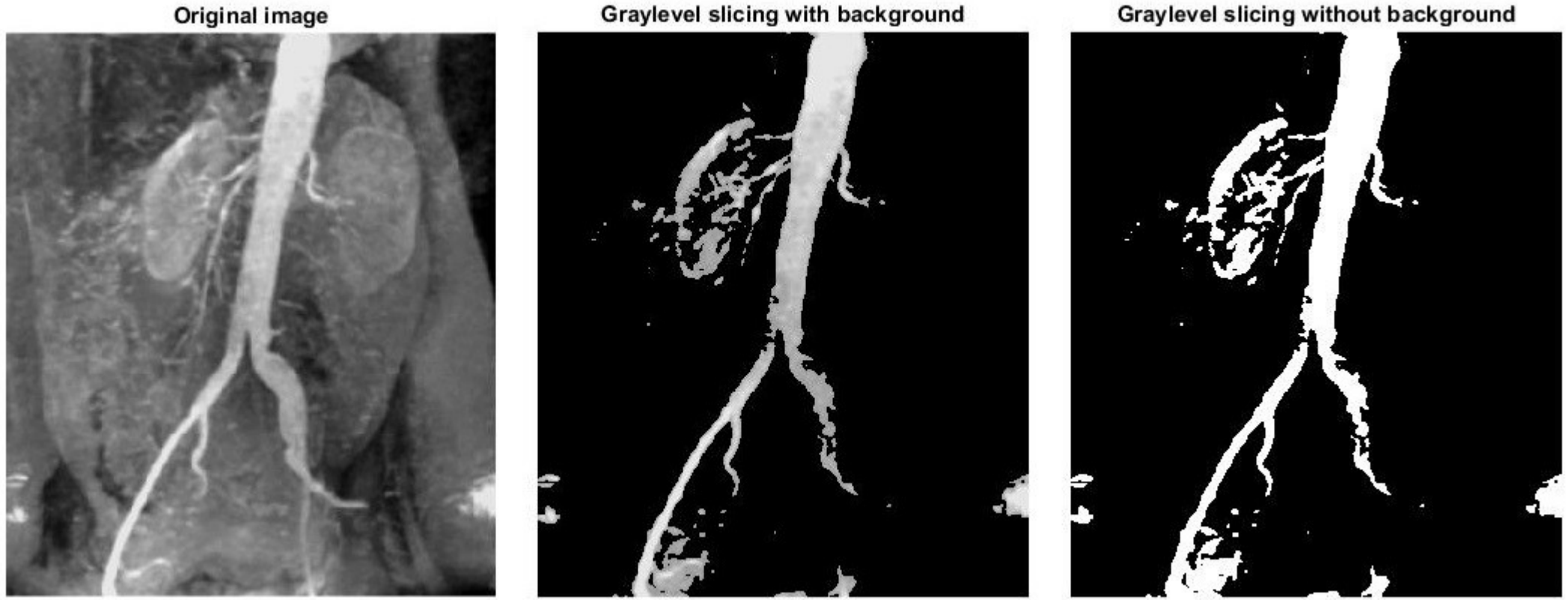
Example

Highlight gray levels **120–160** to detect:

- Tumors in medical images
- Roads/buildings in satellite images

✔ Very useful in feature detection.





Two types

(a) With Background Suppression

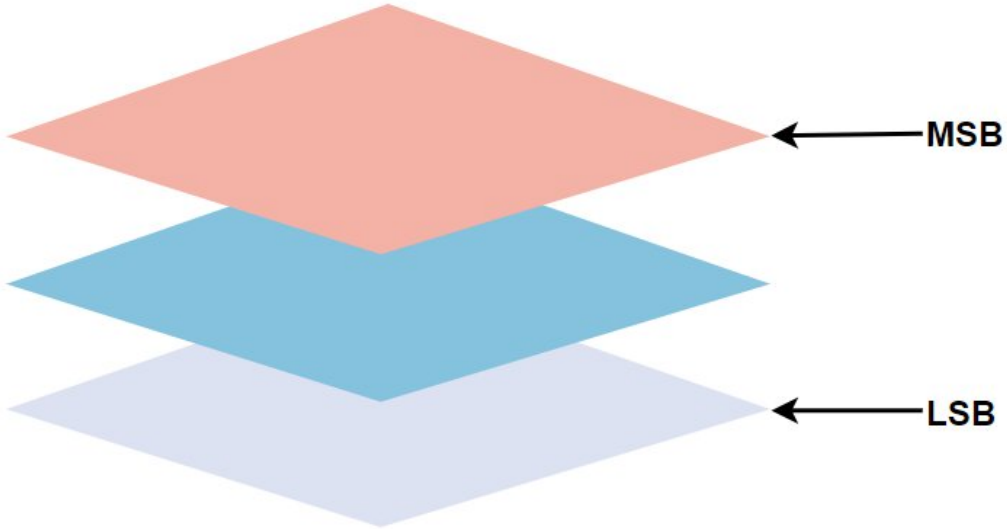
Pixels in the selected range are set to high value (bright), others to low value (dark).

(b) Without Background Suppression

Selected range is highlighted, but background is kept unchanged.

3. Bit-Plane Slicing

- **Bit-plane slicing** is a technique used in digital image processing to analyze the individual bits in the pixel values, where each pixel is represented by a binary number, commonly using 8 bits for grayscale images or 24 bits for color images. Mostly used in image compression.
- The process of bit-plane slicing involves separating and displaying each bit of the pixel values as a separate image. It creates a set of binary images, each corresponding to a single bit of the original pixel values. The most significant bit (MSB) is the leftmost bit, representing the highest order bit; the least significant bit (LSB) is the rightmost bit, representing the lowest order bit.



3-bit grayscale image

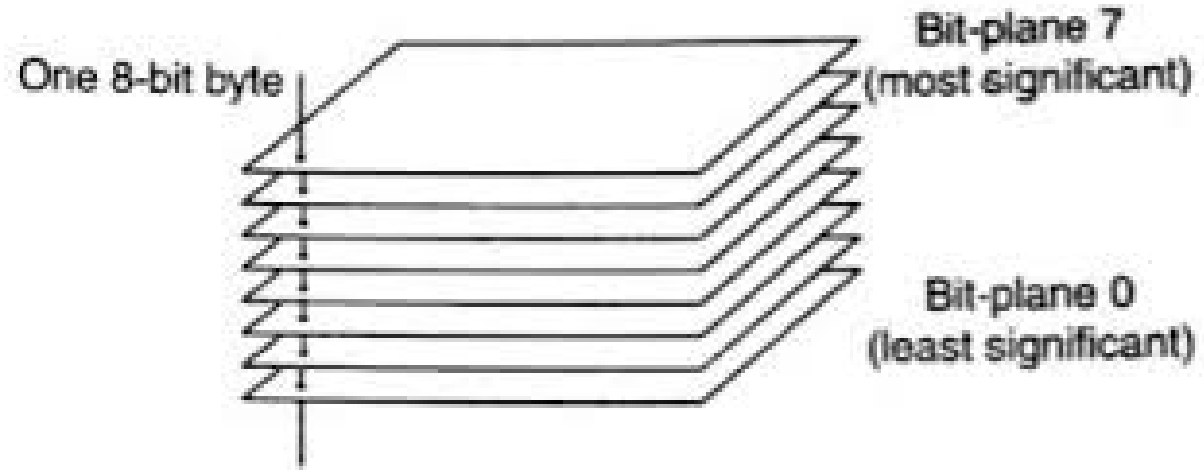


Fig no: 7

the orginal

image of bit-1

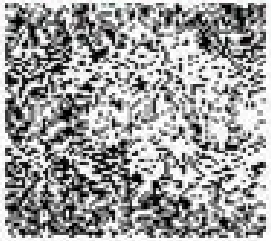


image of bit-2

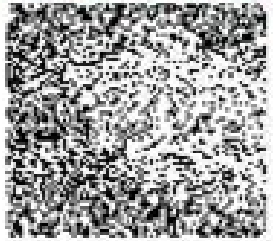


image of bit-3

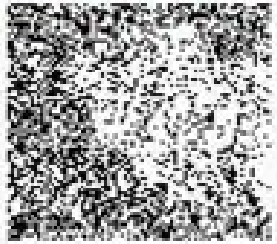


image of bit-4

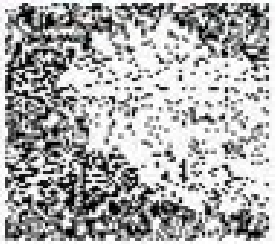


image of bit-5

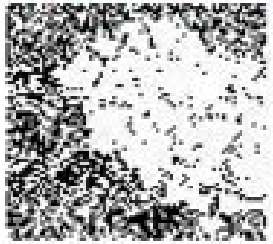


image of bit-6

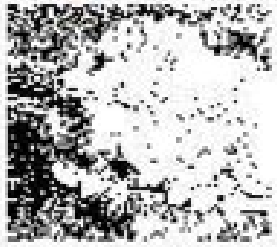


image of bit-7

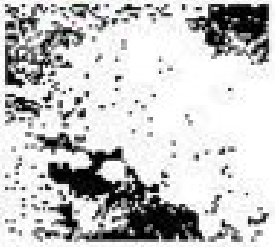
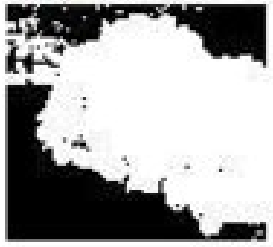
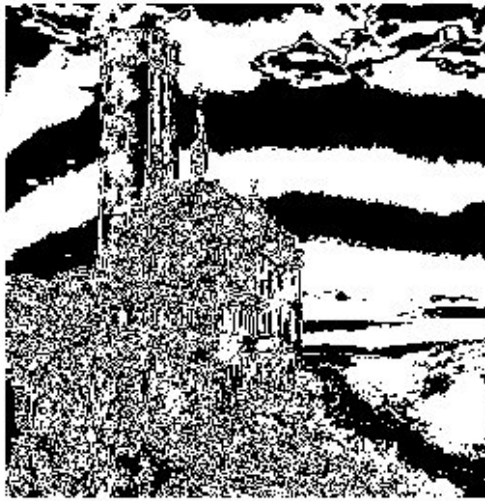
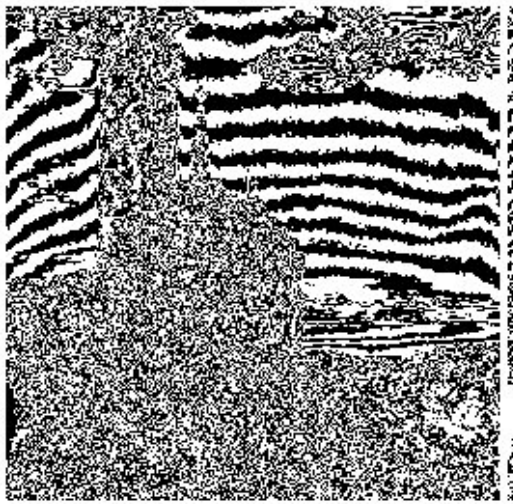
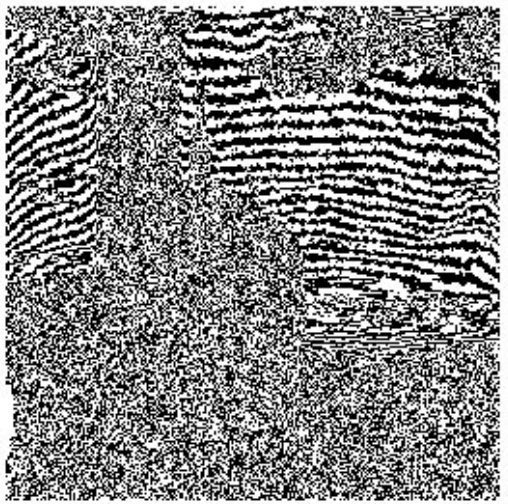


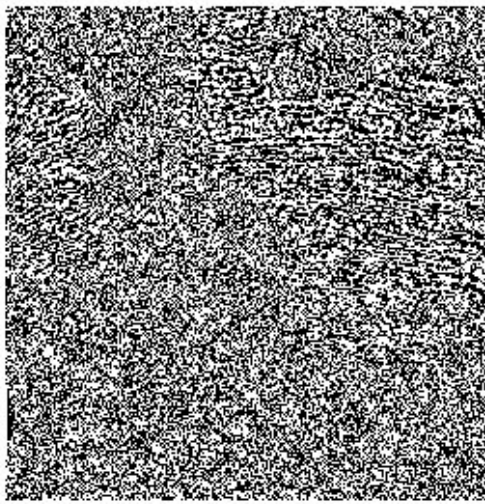
image of bit-8







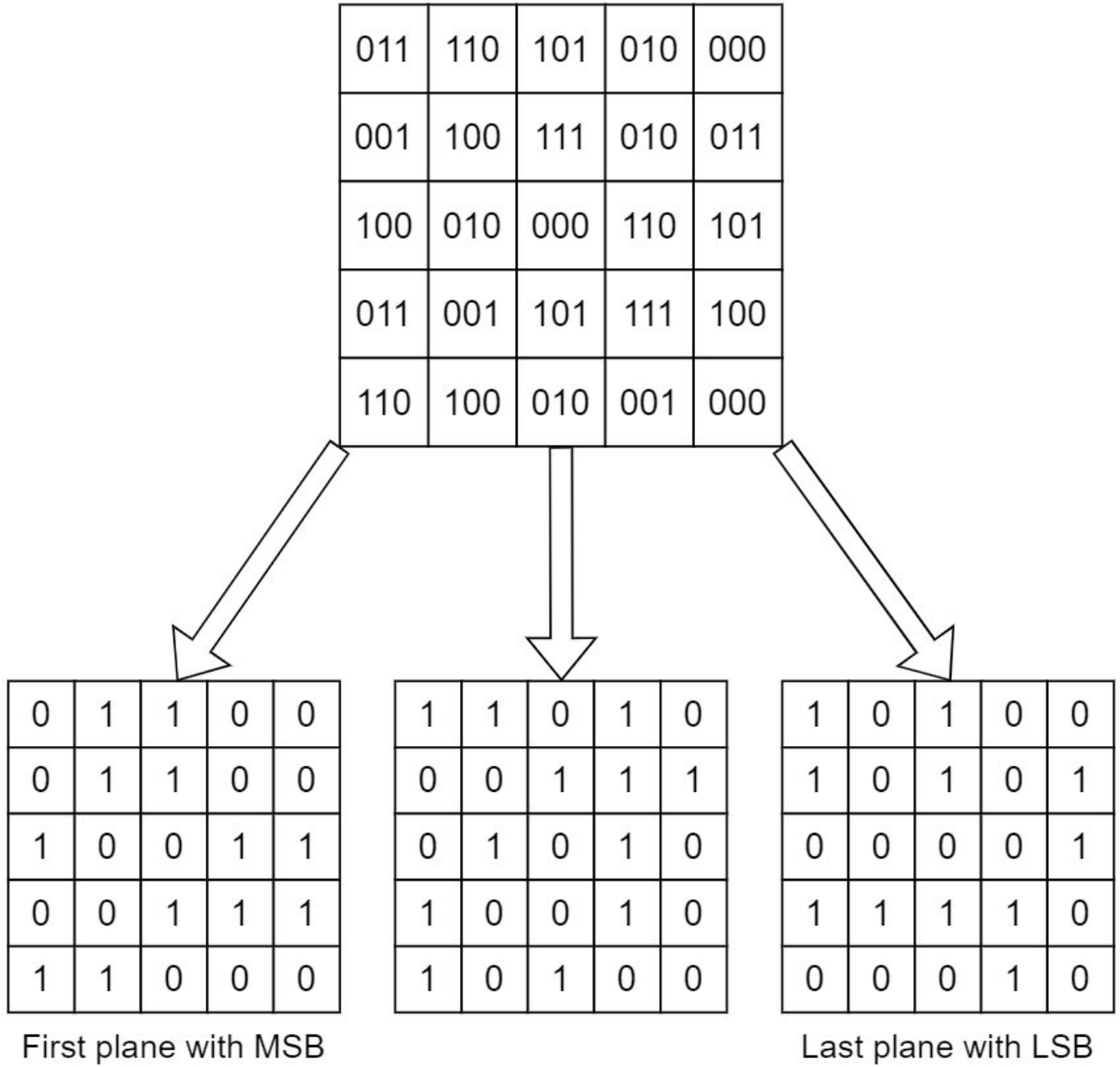




1. **Convert to binary**: Convert the value of each pixel in the image to its binary representation.
2. **Bit-plane extraction**: The number of planes will equal the number of bits used to represent pixel values. Each bit is assigned to a separate plane. The MSB is assigned to the first-bit plane and the LSB to the last-bit plane.
3. **Create bit plane images**: Create separate images for each extracted bit plane. These images will be binary, with pixel values of 0 and 1, representing the presence or absence of the corresponding bit in the original pixel values.

3	6	5	2	0
1	4	7	2	3
4	2	0	6	5
3	1	5	7	4
6	4	2	1	0

011	110	101	010	000
001	100	111	010	011
100	010	000	110	101
011	001	101	111	100
110	100	010	001	000


011 110 101 010 000
001 100 111 010 011
100 010 000 110 101
011 001 101 111 100
110 100 010 001 000
0 1 1 0 0
0 1 1 0 0
1 0 0 1 1
0 0 1 1 1
1 1 0 0 0
1 1 0 1 0
0 0 1 1 1
0 1 0 1 0
1 0 0 1 0
1 0 1 0 0
1 0 1 0 0
1 0 1 0 1
0 0 0 0 1
1 1 1 1 0
0 0 0 1 0
First plane with MSB
Last plane with LSB

Lab 05: Bit Plane Slicing

Editor

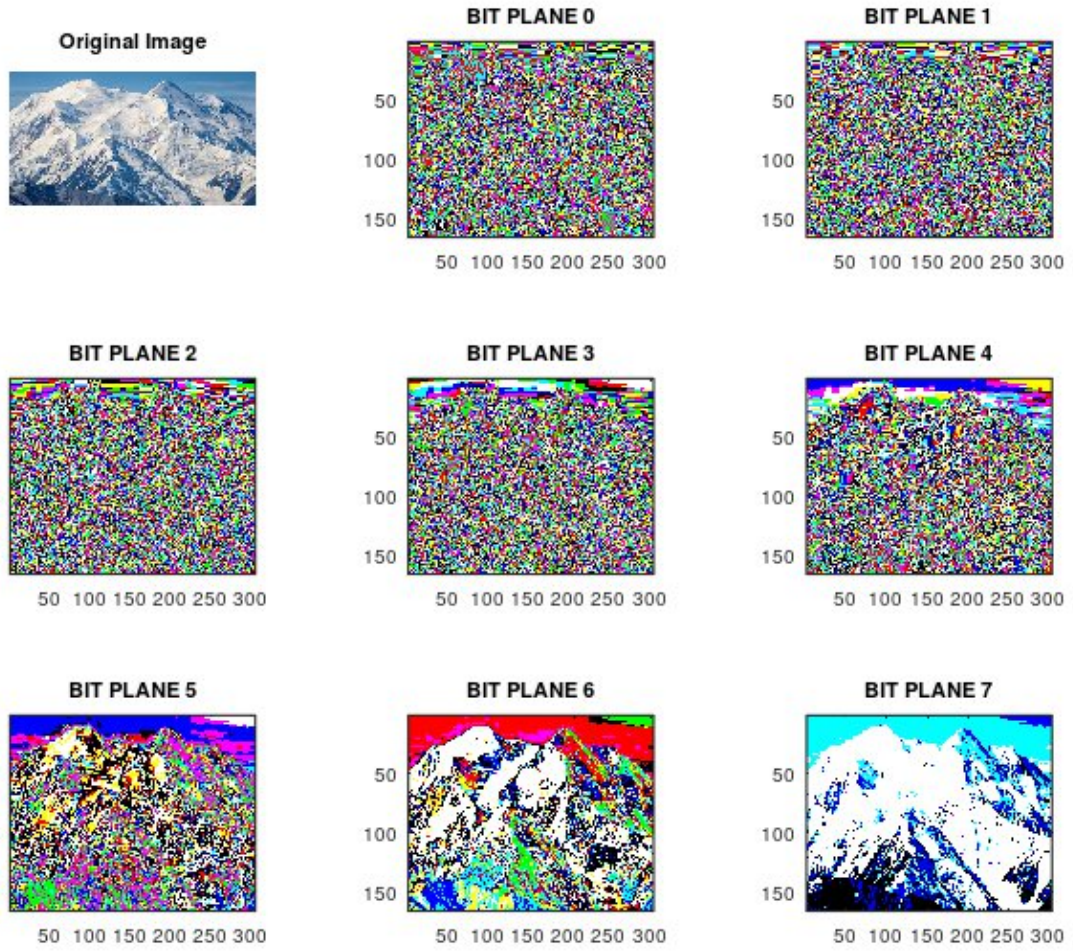
File Edit View Debug Run Help

bit_plane.m | negative_linear_transformation.m

```
1  clear all; % clear all variables
2  close all; % close all figures
3  clc;       % clear command window
4  % import image package
5  pkg load image;
6  % read image
7  i=imread("mountain.png");
8  b0=double(bitget(i,1));
9  b1=double(bitget(i,2));
10 b2=double(bitget(i,3));
11 b3=double(bitget(i,4));
12 b4=double(bitget(i,5));
13 b5=double(bitget(i,6));
14 b6=double(bitget(i,7));
15 b7=double(bitget(i,8));
16 subplot(3,3,1);imshow(i);title('Original Image');
17 subplot(3,3,2);subimage(b0);title('BIT PLANE 0');
18 subplot(3,3,3);subimage(b1);title('BIT PLANE 1');
19 subplot(3,3,4);subimage(b2);title('BIT PLANE 2');
20 subplot(3,3,5);subimage(b3);title('BIT PLANE 3');
21 subplot(3,3,6);subimage(b4);title('BIT PLANE 4');
22 subplot(3,3,7);subimage(b5);title('BIT PLANE 5');
23 subplot(3,3,8);subimage(b6);title('BIT PLANE 6');
24 subplot(3,3,9);subimage(b7);title('BIT PLANE 7');
25
```

Figure 1

File Edit Tools

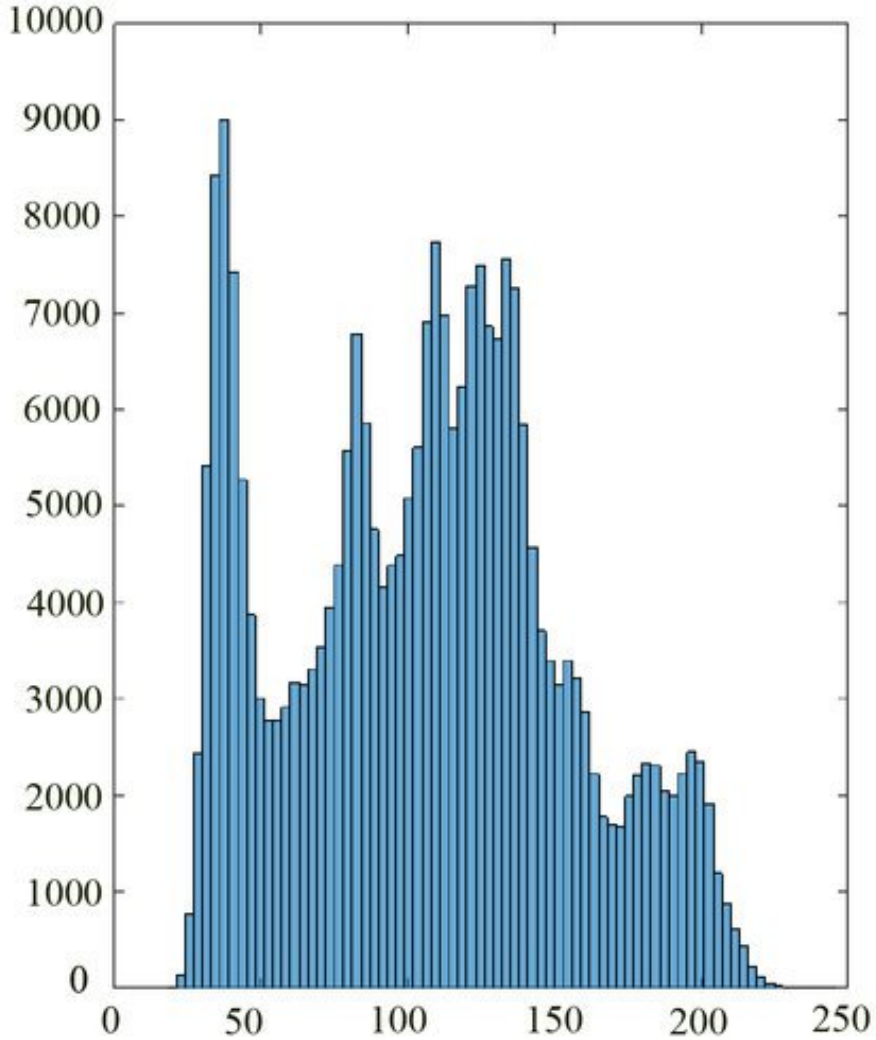


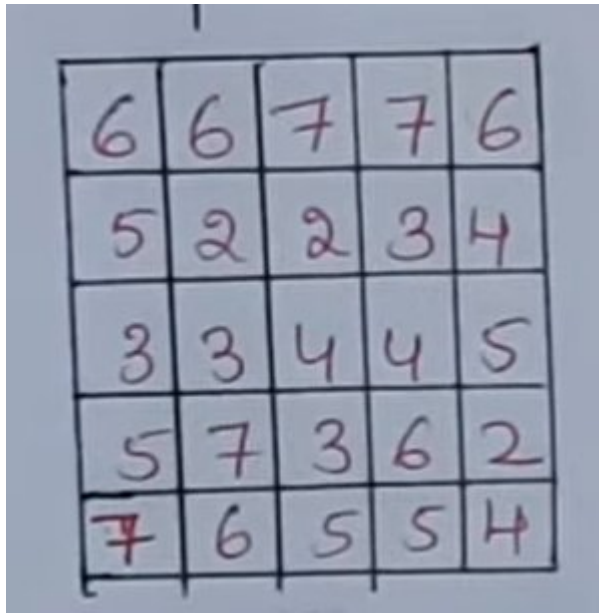
Histogram in Digital Image Processing

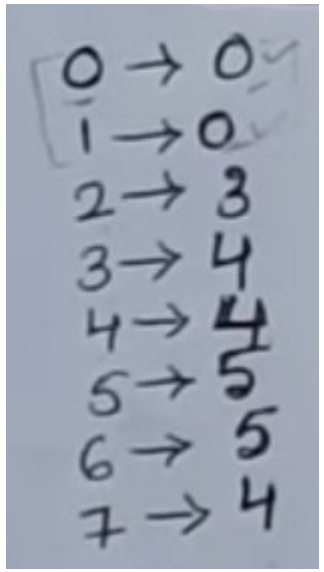
An image histogram is a graphical representation of the distribution of gray levels (intensity values) in an image.

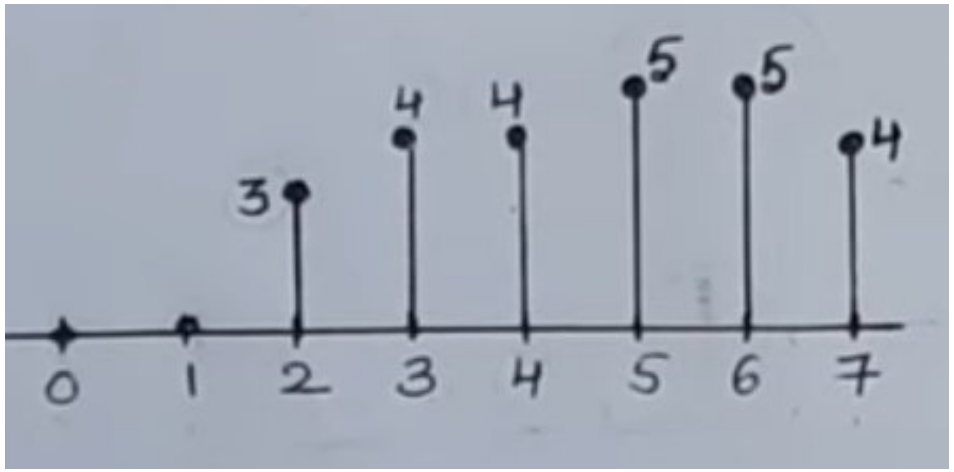
- *X-axis: Gray levels (0–255 for 8-bit image)*
- *Y-axis: Number of pixels (frequency)*

✔ ***It gives a global description of image brightness and contrast.***









✔ The histogram shows how many pixels have each gray level.

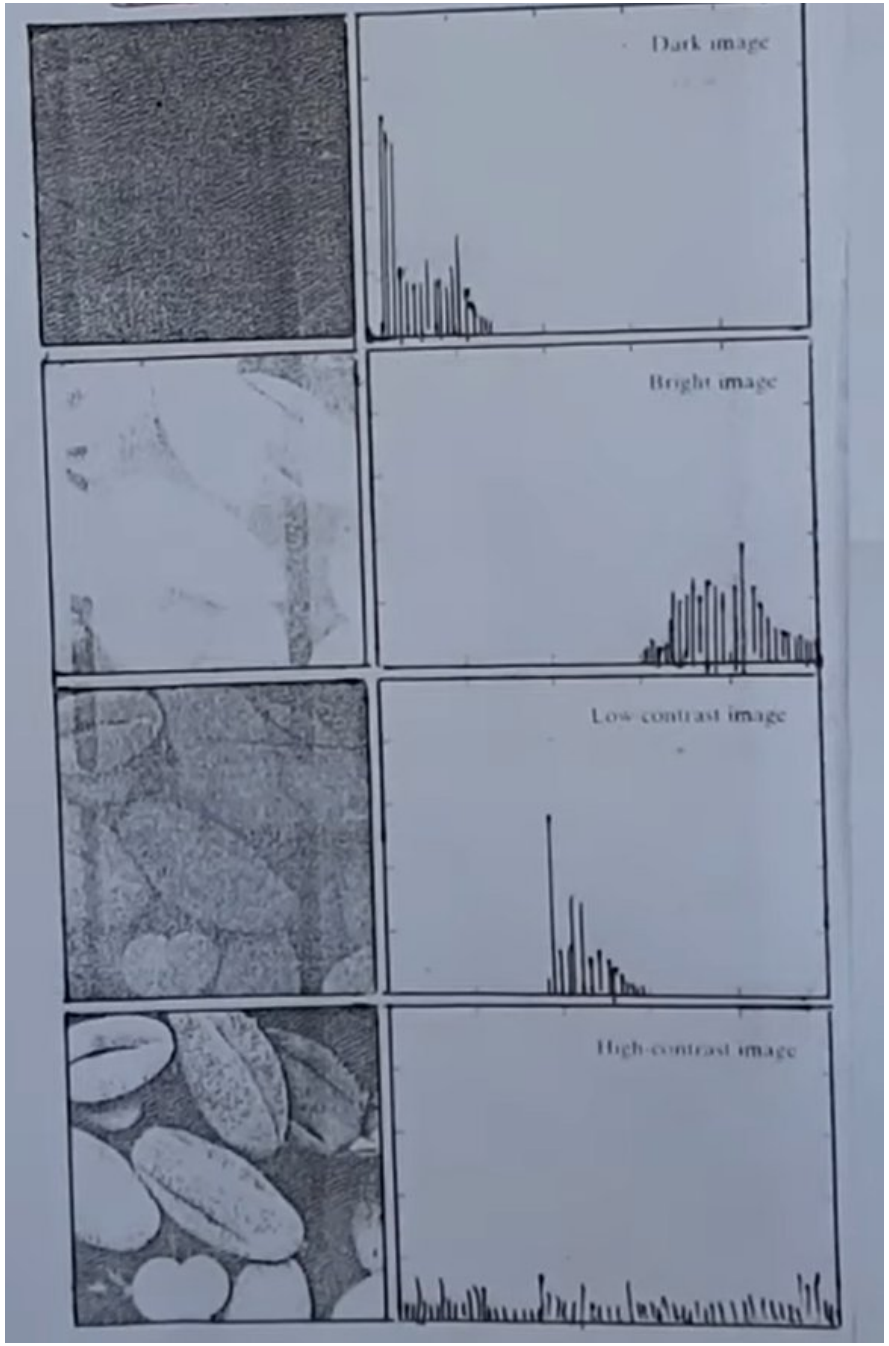
Interpretation:

- Histogram skewed left → dark image
- Histogram skewed right → bright image
- Narrow histogram → low contrast
- Spread histogram → high contrast

Applications of Histogram

Histograms are widely used because they are simple and powerful.

1.Image contrast analysis
2.Brightness adjustment
3.Image enhancement
4.Threshold selection (image segmentation)
5.Quality inspection (medical, satellite images)
6.Automatic image correction in cameras



In black image, the histogram values are placed towards 0 level i.e. left

In black image, the histogram values are placed towards 0 level i.e. left

In white image, the histogram values are placed towards 255 level i.e. right

For low contrast image, the histogram values are at the center and not distributed

High quality image consists of histogram normalized in flat surface; i.e. distributed among all level 0 to 255

1. Histogram Equalization

Redistributes gray levels so that the histogram becomes **approximately uniform**.

Purpose:

✔ Improve overall contrast

Working idea:

- Uses cumulative distribution function (CDF)
- Spreads pixel values across full range

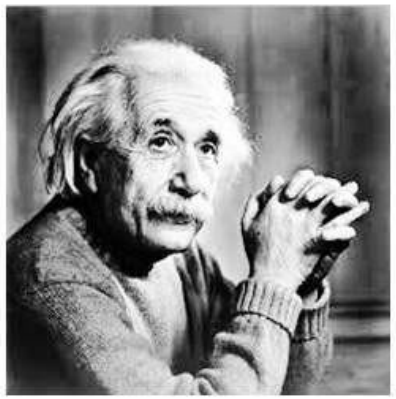
Result:

- Dark regions become clearer
- Bright regions become balanced

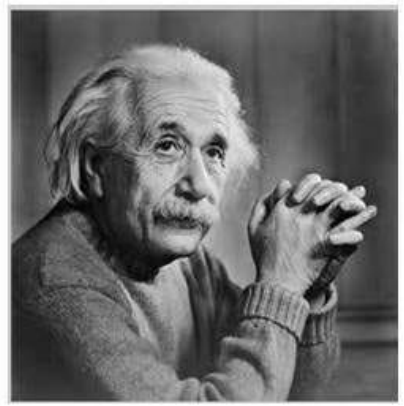
Application:

- Medical images
- Low-contrast photographs

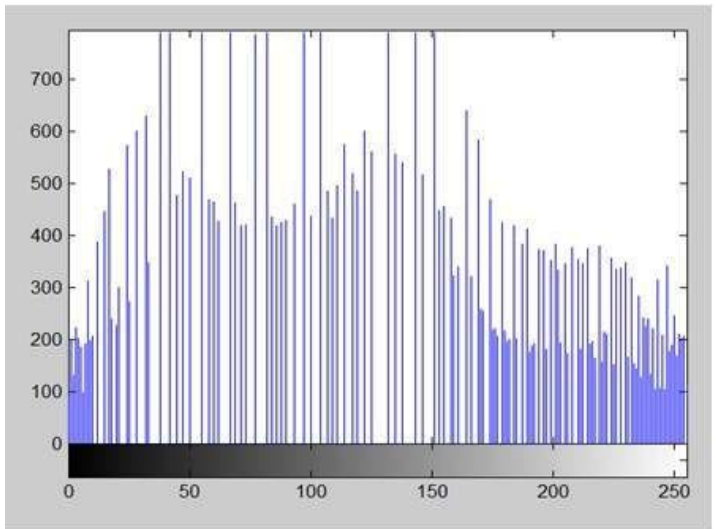
New Image



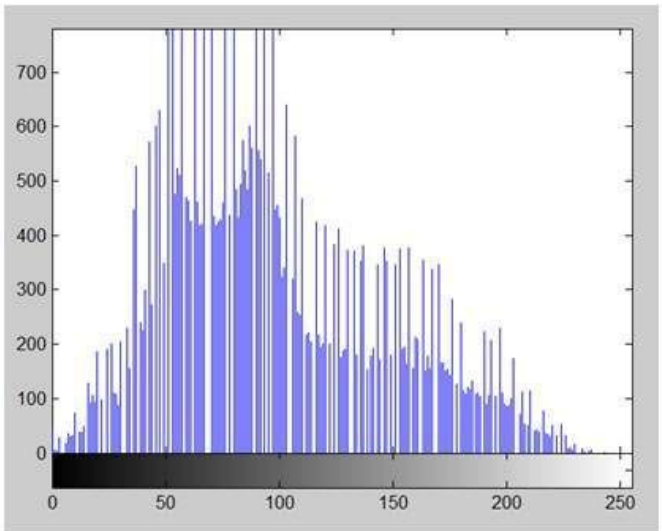
Old image



New Histogram

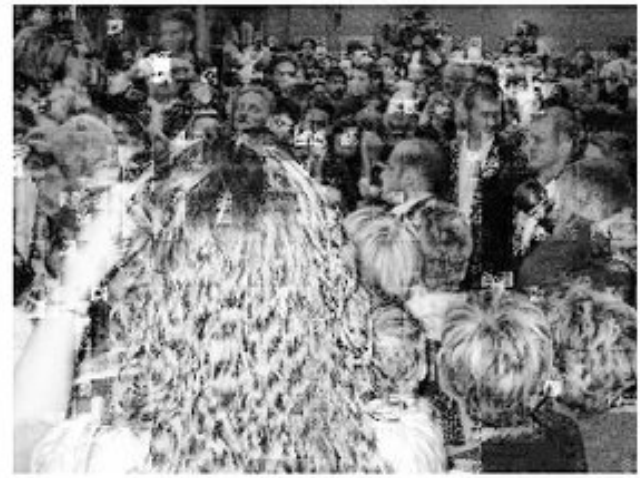


Old Histogram

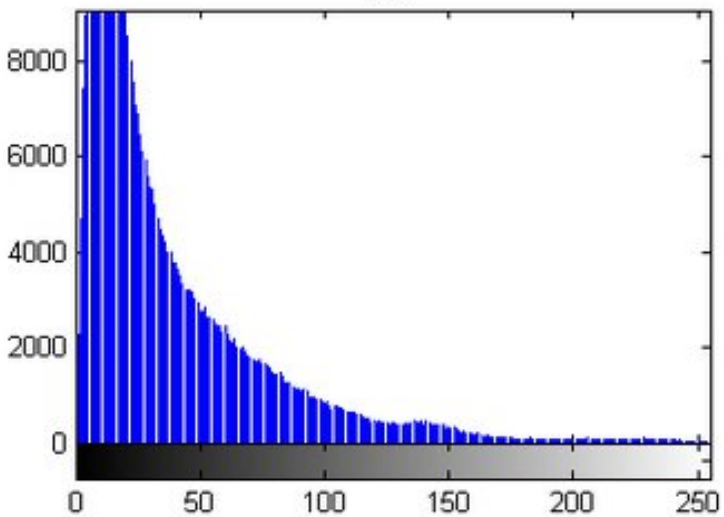


(a)

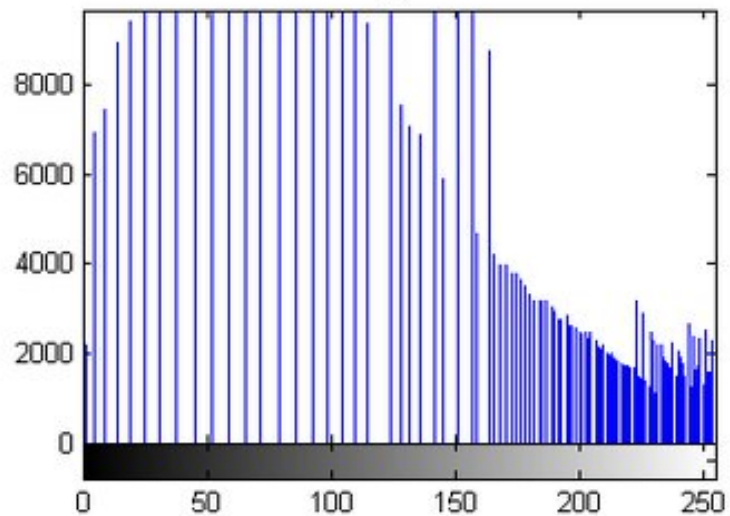
(b)



(c)


8000
6000
4000
2000
0
0
50
100
150
200
250

(d)


8000
6000
4000
2000
0
0
50
100
150
200
250

Discuss the algorithm for histogram equalizations

Algorithm for Histogram Equalization

Histogram equalization is a **global contrast enhancement technique** that redistributes gray levels so that the histogram becomes **approximately uniform**.

Assume:

- Image size = $M \times N$
- Gray levels = $0, 1, 2, \ldots, L-1$ (for 8-bit, $L = 256$)

Step-by-Step Algorithm

1. **Read the grayscale image**

   Let r_k be the k-th gray level.
2. **Compute the histogram**

   Count number of pixels n_k having gray level r_k.

3. **Normalize the histogram (PDF)**

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

4. **Compute the cumulative distribution function (CDF)**

$$CDF(r_k) = \sum_{j=0}^{k} p(r_j)$$

5. **Compute the transformation function**

$$s_k = (L-1) \times CDF(r_k)$$

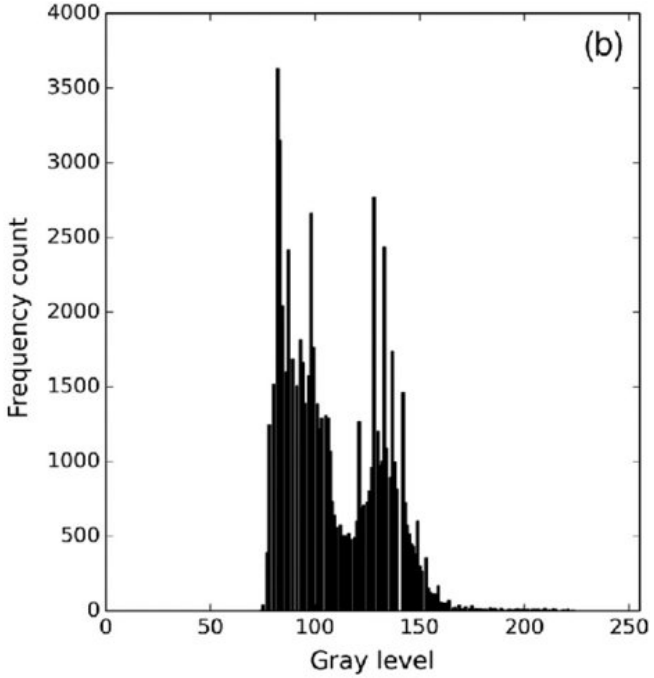
(round to nearest integer)

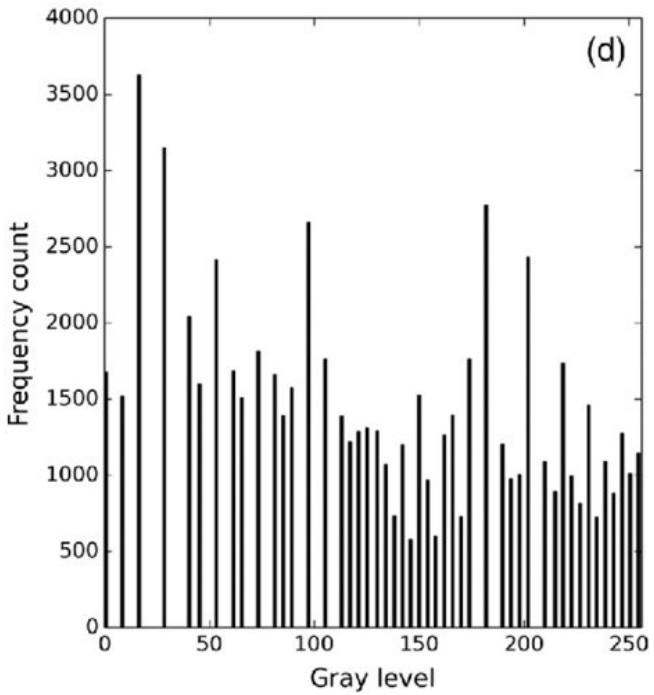
6. **Map old gray levels to new gray levels**

   Replace each pixel of value r_k by s_k.

7. **Obtain the enhanced image**

   The result has improved contrast.





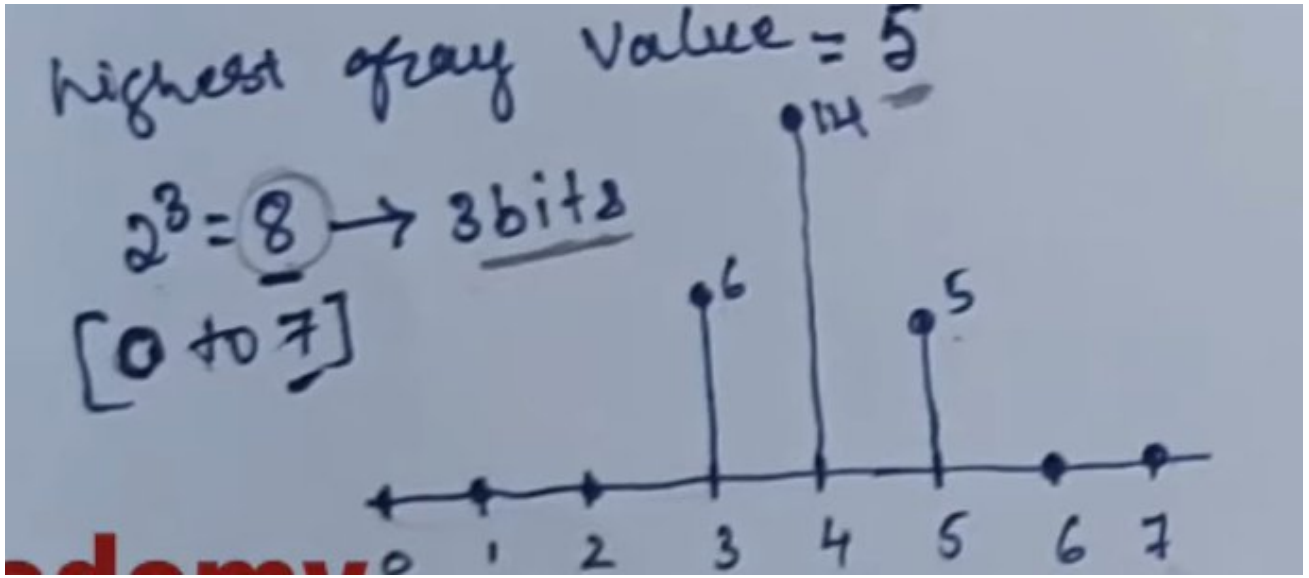
Example: Perform the histogram equalization of following pixel values

$f(x,y)=$

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

H.E.

i/p Image

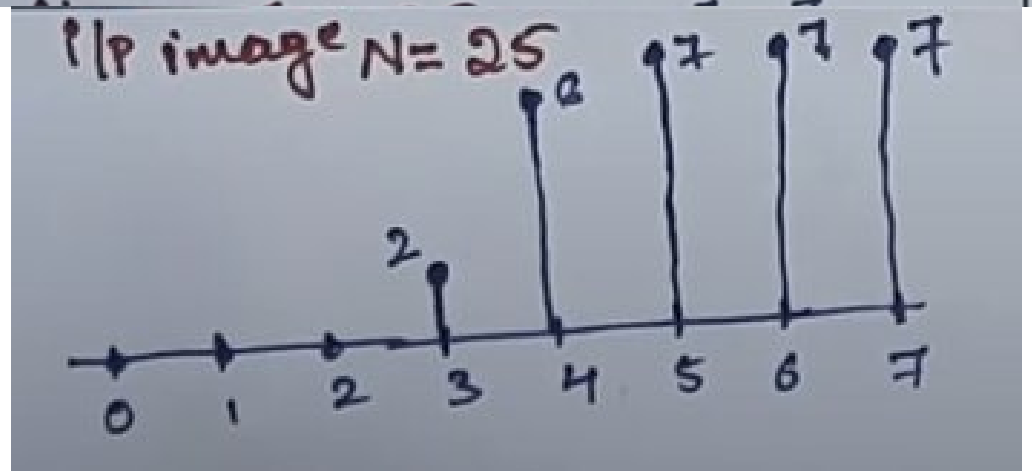


Gray levels	0	1	2	3	4	5	6	7
No of pixels n_k	0	0	0	6	14	5	0	0

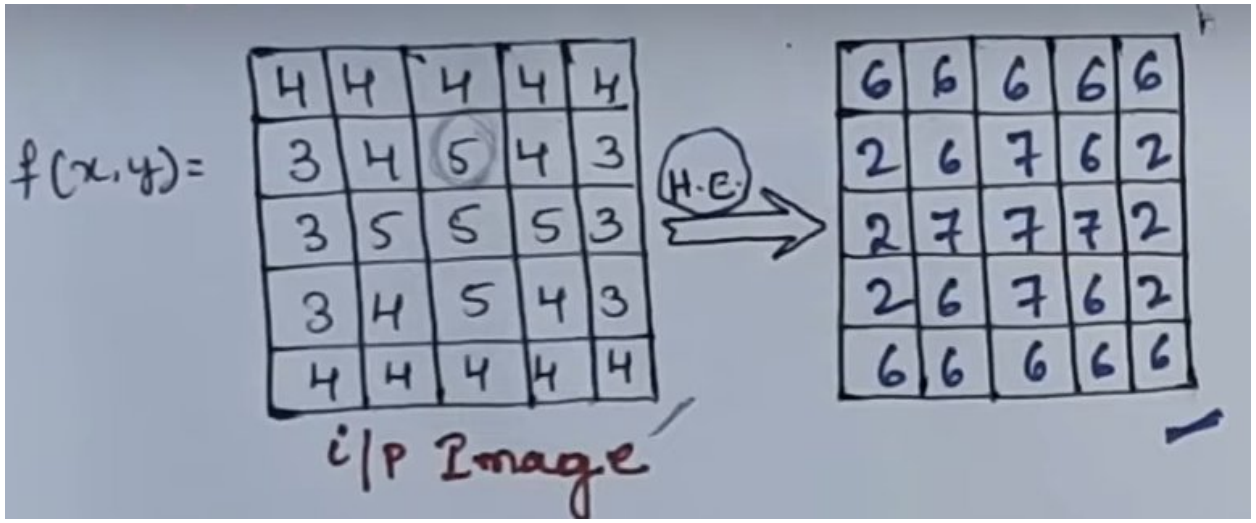
Histogram of input image.

Now needs to find the histogram equalization for this input histogram.

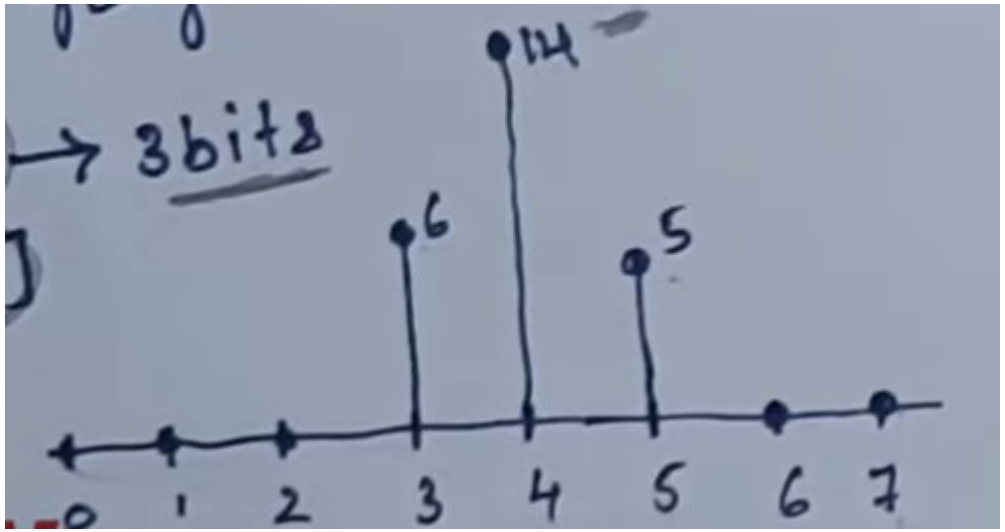
Gray level	No. of pixels n_k	PDF = n_k/sum	CDF = S_k	$S_k \times 7$	Histogram equal. level
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	6	$6/25 = 0.24$	0.24	1.68	2
4	14	$\frac{14}{25} = 0.56$	0.8	5.6	6
5	5	$5/25 = 0.2$	1.0	7	7
6	0	0	1.0	7	7
7	0	0	1.0	7	7.

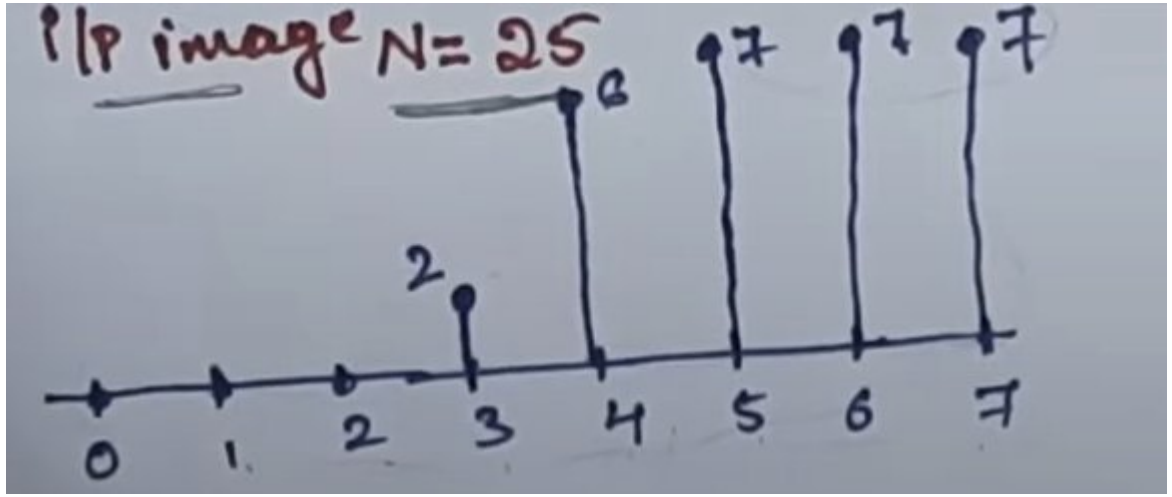


PDF = Probability Distribution Function
CDF = Cumulative Distribution Function



As we can see that histogram of output function is somewhat more distributed than input image function.

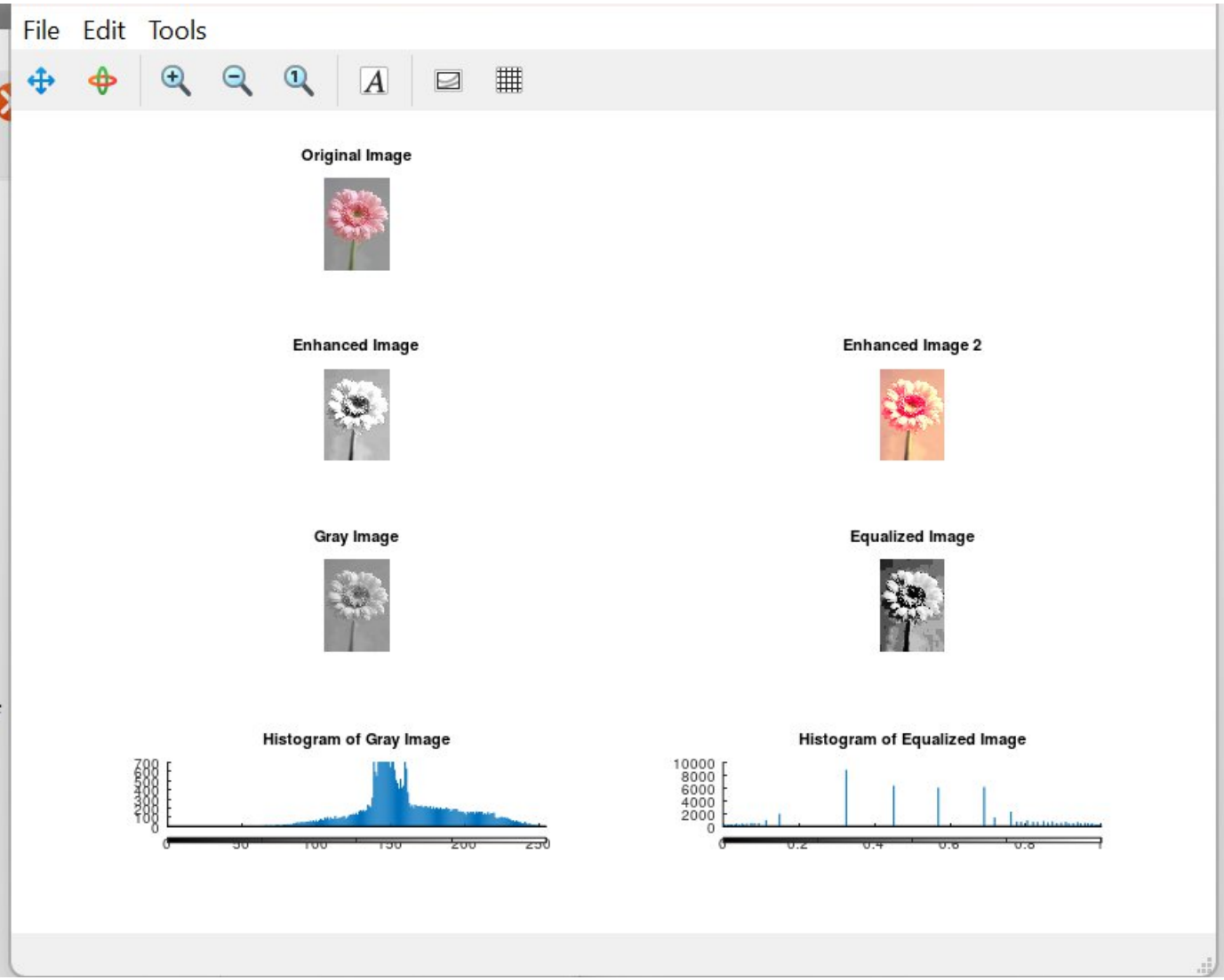




Lab 06: Histogram Equalization

```
Edit View Debug Run Help
t_plane.m | negative_linear_transformation.m | histogram.m

clear all; % clear all variables
close all; % close all figures
clc;       % clear command window
% import image package
pkg load image;
% read image for Image Enhancement
I=imread('flower.png');
subplot(4,2,1); imshow(I); title('Original Image');
g=rgb2gray(I);
subplot(4,2,5); imshow(g); title('Gray Image');
J=imadjust(g,[0.3 0.7],[]);
subplot(4,2,3); imshow(J); title('Enhanced Image');
D= imadjust(I,[0.2 0.3 0; 0.6 0.7 1],[]);
subplot(4,2,4);imshow(D);title('Enhanced Image 2');
% Histogram and Histogram Equalization
subplot(4,2,7); imhist(g); title('Histogram of Gray Image');
m=histeq(g);
subplot(4,2,6); imshow(m); title('Equalized Image');
subplot(4,2,8); imhist(m); title('Histogram of Equalized Image');
```



Spatial Operations in DIP

Spatial operations are image processing techniques applied **directly on pixels in the spatial domain** using a small neighborhood (window or mask) around each pixel.

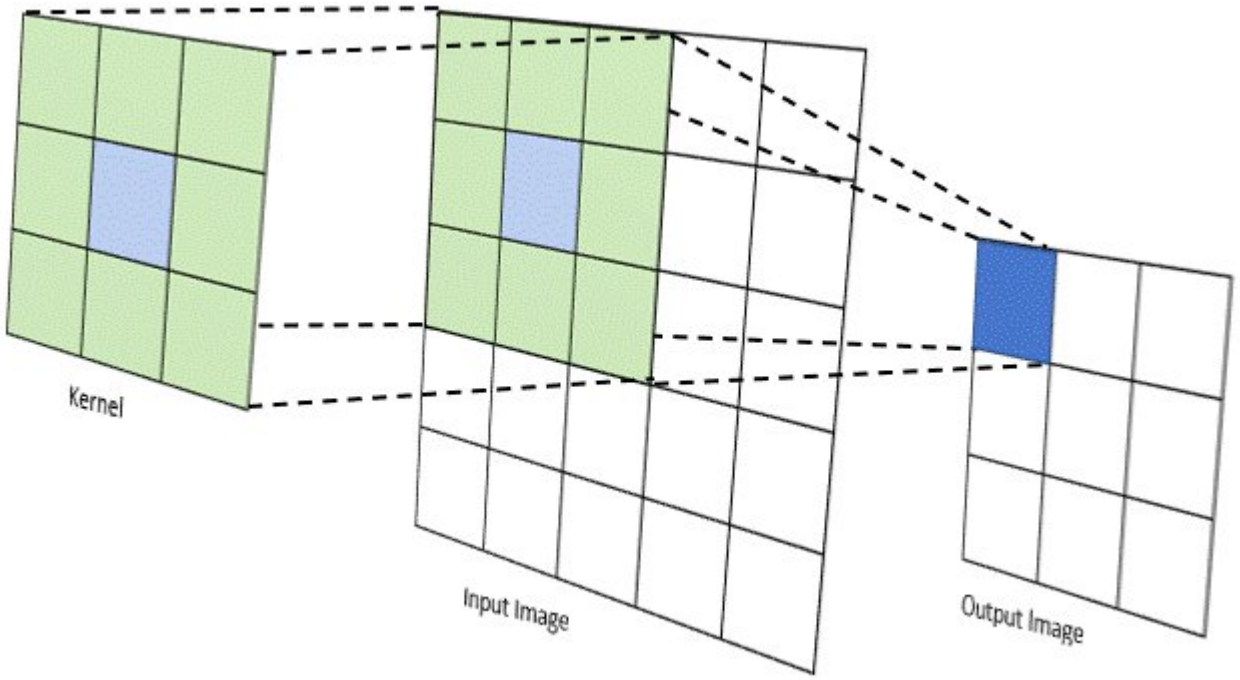
General form:

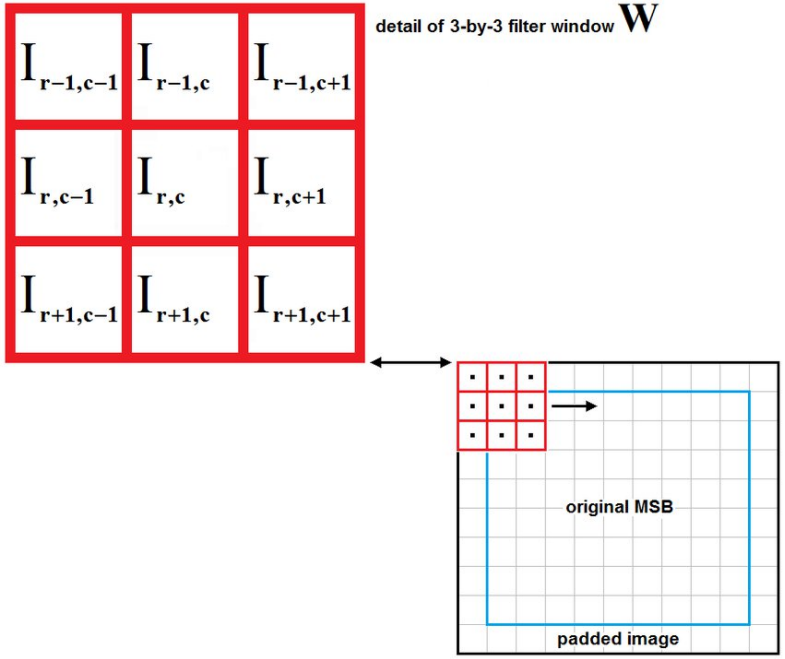
$$g(x,y) = T\{f(x,y)\}$$

where

- $f(x,y)$ = input image
- $g(x,y)$ = output image
- T = operation on neighborhood of (x,y)

✔ Used for **smoothing, sharpening, and noise reduction.**





Types of Spatial Filters

Spatial filters are mainly classified into:

1. **Linear Filters**
2. **Non-Linear Filters**

1. Linear Filters

A filter is **linear** if the output pixel is a **linear combination (weighted sum)** of input pixels.

Mathematical form:

$$g(x,y) = \sum_i \sum_j w(i,j)\, f(x+i, y+j)$$

Characteristics:

- Uses **addition and multiplication**
- Obeys **superposition principle**
- Easy to implement

Examples of Linear Filters

(a) Averaging (Mean) Filter – Smoothing

Reduces noise by replacing each pixel with the **average of neighbors**.

3×3 Mean Filter:

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

✔ Reduces noise

✗ Blurs edges

The figure below shows two 3×3 averaging filters.

$\frac{1}{9} \times$

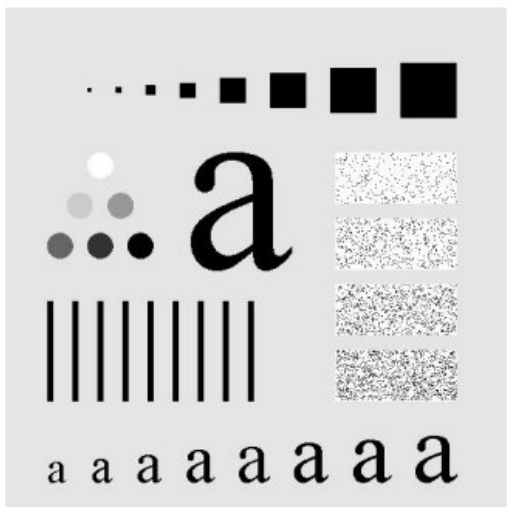
1	1	1
1	1	1
1	1	1

Standard average filter

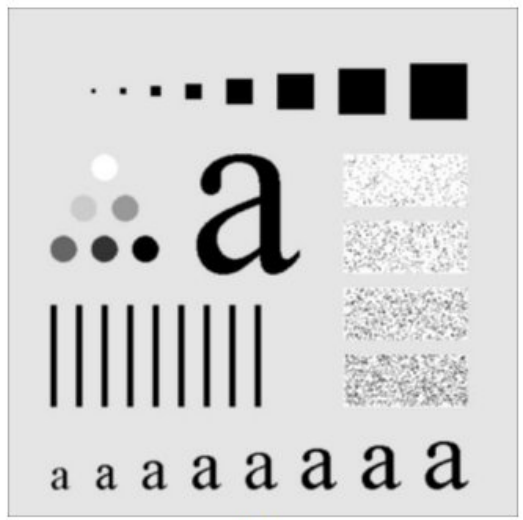
$\frac{1}{16} \times$

1	2	1
2	4	2
1	2	1

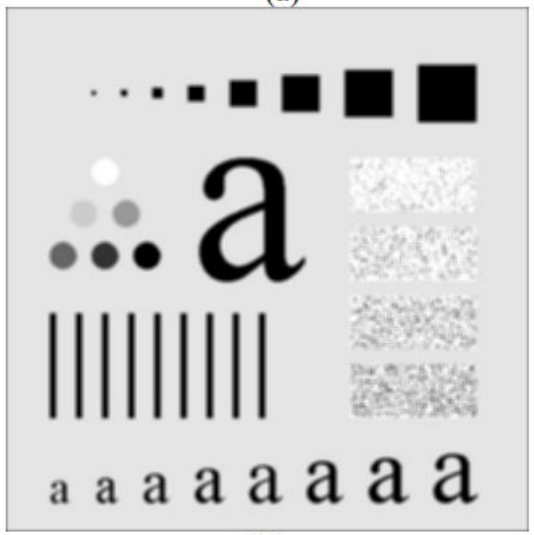
Weighted average filter



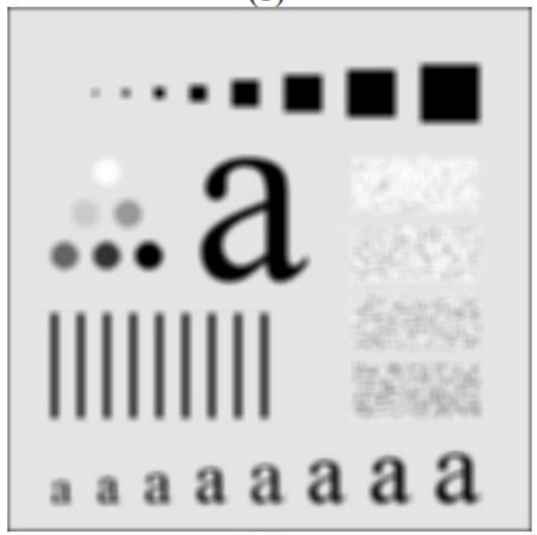
(a)



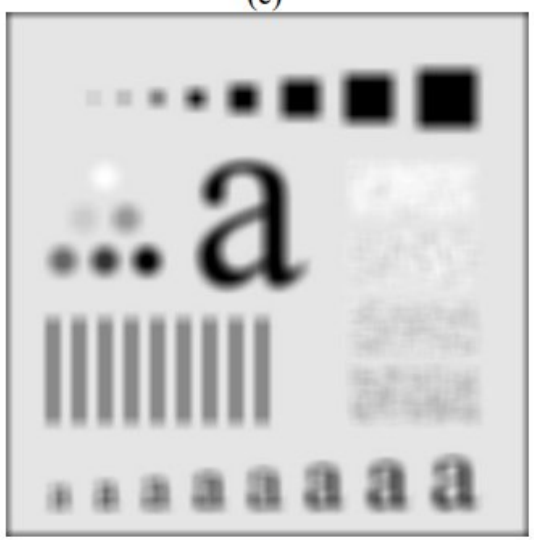
(b)



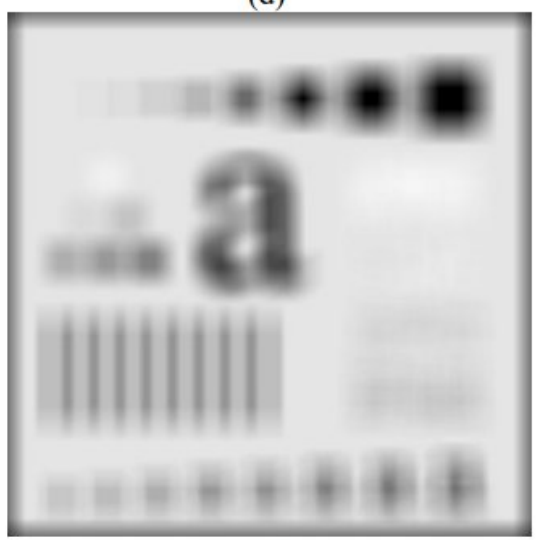
(c)



(d)



(e)



(f)

Figure 6.2 Effect of averaging filter. (a) Original image. (b)-(f) Results of smoothing with square averaging filter masks of sizes n = 3,5,9,15, and 35, respectively.

(b) Sharpening Filter (High-pass)

Enhances edges and fine details.

Example Mask:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

✔ Enhances edges

Applications of Linear Filters

- Noise reduction (Gaussian, mean)
- Image blurring
- Edge enhancement

Example:

Use the following 3×3mask to perform the convolution process on the shaded pixels in the 5×5 image below. Write the filtered image.

0	1/6	0
1/6	1/3	1/6
0	1/6	0

3×3 mask

30	40	50	70	90
40	50	80	60	100
35	255	70	0	120
30	45	80	100	130
40	50	90	125	140

5×5 image

Filtered image =

30	40	50	70	90
40	**85**	**65**	**61**	100
35	**118**	**92**	**58**	120
30	**84**	**77**	**89**	130
40	50	90	125	140

Solution:

$$0 \times 30 + \frac{1}{6} \times 40 + 0 \times 50 + \frac{1}{6} \times 40 + \frac{1}{3} \times 50 + \frac{1}{6} \times 80 + 0 \times 35 + \frac{1}{6} \times 255 + 0 \times 70 = 85$$

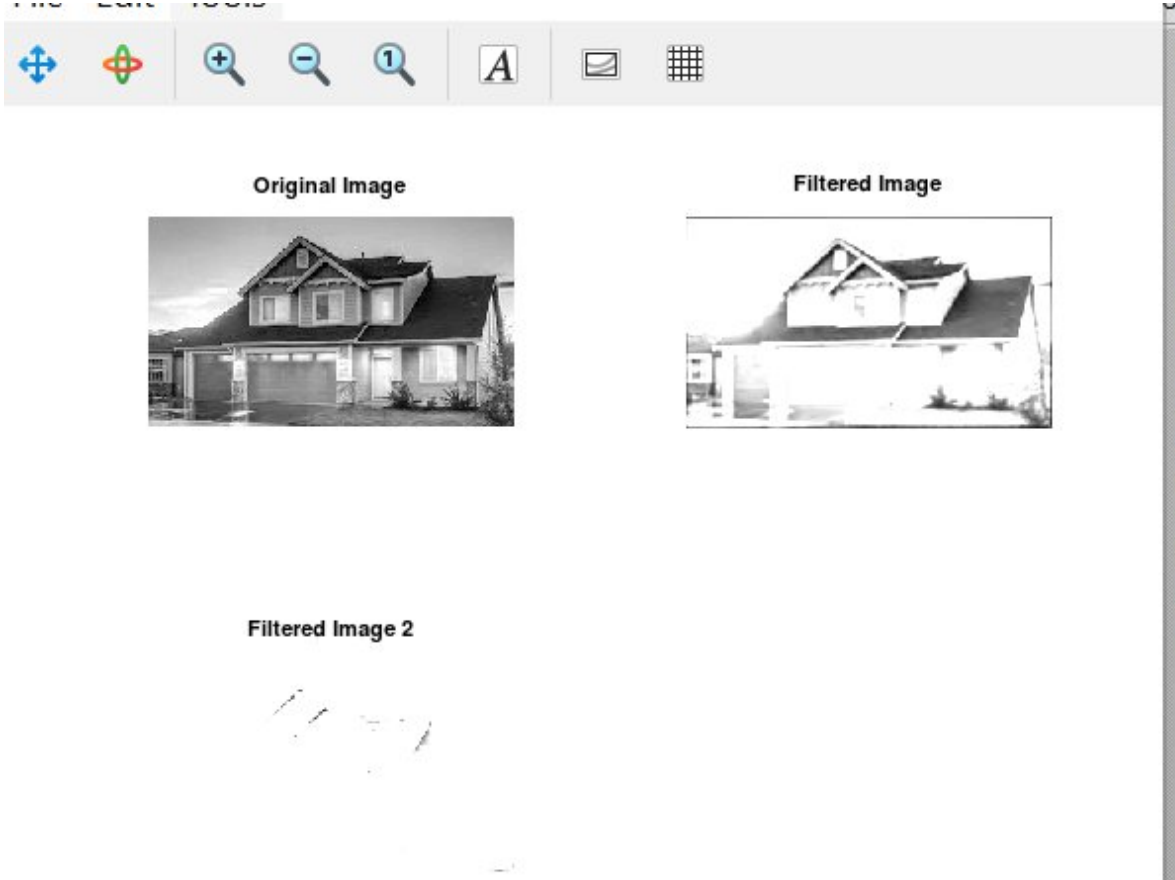
$$0 \times 40 + \frac{1}{6} \times 50 + 0 \times 70 + \frac{1}{6} \times 50 + \frac{1}{3} \times 80 + \frac{1}{6} \times 60 + 0 \times 255 + \frac{1}{6} \times 70 + 0 \times 0 = 65$$

$$0 \times 50 + \frac{1}{6} \times 70 + 0 \times 90 + \frac{1}{6} \times 80 + \frac{1}{3} \times 60 + \frac{1}{6} \times 100 + 0 \times 70 + \frac{1}{6} \times 0 + 0 \times 120 =$$

$$0 \times 40 + \frac{1}{6} \times 50 + 0 \times 80 + \frac{1}{6} \times 35 + \frac{1}{3} \times 255 + \frac{1}{6} \times 70 + 0 \times 30 + \frac{1}{6} \times 45 + 0 \times 80 = 118$$

and so on …

Lab 07: Convolution Filtering




```
clear all; % clear all variables
close all; % close all figures
clc;       % clear command window
% import image package
pkg load image;
% read image
%Implementation of filter using Convolution
#figure;
I= imread('house.png');
I=I(:,:,1); subplot(2,2,1); imshow(I); title('Original Image');
a=[0.001 0.001 0.001; 0.001 0.001 0.001; 0.001 0.001 0.001];
R=conv2(a,I);
subplot(2,2,2); imshow(R); title('Filtered Image');
b=[0.005 0.005 0.005; 0.005 0.005 0.005; 0.005 0.005 0.005];
R1=conv2(b,I);
subplot(2,2,3); imshow(R1); title('Filtered Image 2');
```

2. Non-Linear Filters

A filter is **non-linear** if the output pixel is **not a linear combination** of input pixels.

Characteristics:

- Uses **sorting, comparison, logical operations**
- Does **not obey superposition**
- Better at **preserving edges**

Examples of Non-Linear Filters

(a) Median Filter

Replaces the center pixel with the **median** of neighborhood values.

Example (3×3 window):

Pixels: {10, 20, 20, 25, 100, 22, 21, 23, 24}

Median = **22**

- ✔ Excellent for **salt-and-pepper noise**
- ✔ Preserves edges better than mean filter

Example:

Consider the following 5×5 image:

20	30	50	80	100
30	20	80	100	110
25	255	70	0	120
30	30	80	100	130
40	50	90	125	140

Filtered Image =

20	30	50	80	100
30	20	80	100	110
25	**30**	**80**	**100**	120
30	30	80	100	130
40	50	90	125	140

Apply a 3×3 median filter on the shaded pixels, and write the filtered image.

Solution

20	30	50	80	100
30	20	80	100	110
25	255	70	0	120
30	30	80	100	130
40	50	90	125	140

Sort:
20, 25, 30, 30, **30**, 70, 80, 80, 255

20	30	50	80	100
30	20	80	100	110
25	255	70	0	120
30	30	80	100	130
40	50	90	125	140

Sort
0, 20, 30, 70, **80**, 80, 100, 100, 255

20	30	50	80	100
30	20	80	100	110
25	255	70	0	120
30	30	80	100	130
40	50	90	125	140

Sort
0, 70, 80, 80, **100**, 100, 110, 120, 130

(b) Max and Min Filters

- **Max filter:** brightens image, removes pepper noise
- **Min filter:** darkens image, removes salt noise

Applications of Non-Linear Filters

- Noise removal
- Image enhancement
- Morphological preprocessing

- *Spatial operations work directly on pixels.*
- *Linear filters use weighted sums (mean, sharpening).*
- *Non-linear filters use ranking or logic (median).*

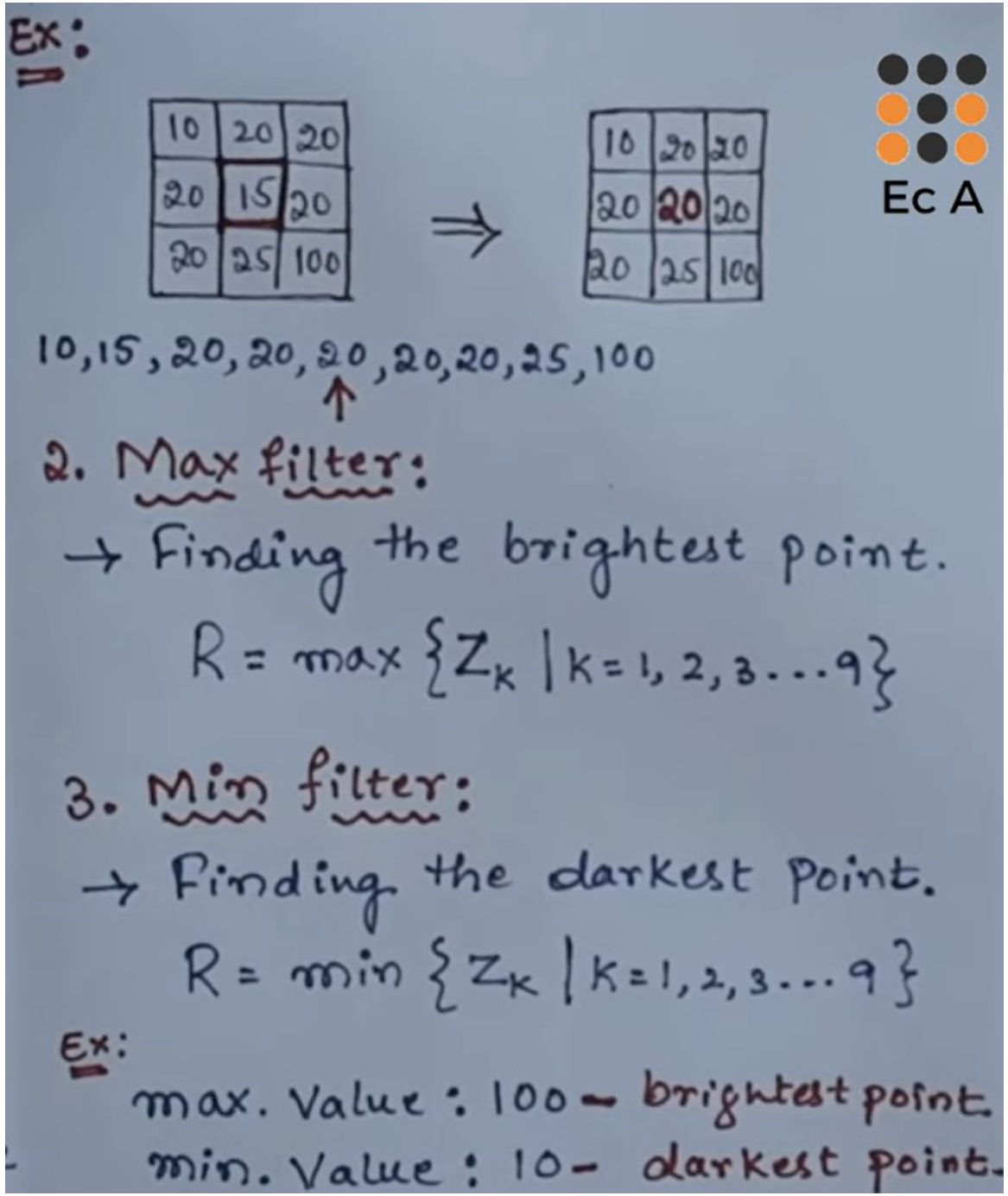


Figure below shows an example of applying the median filter on an image corrupted with salt-and-pepper noise.

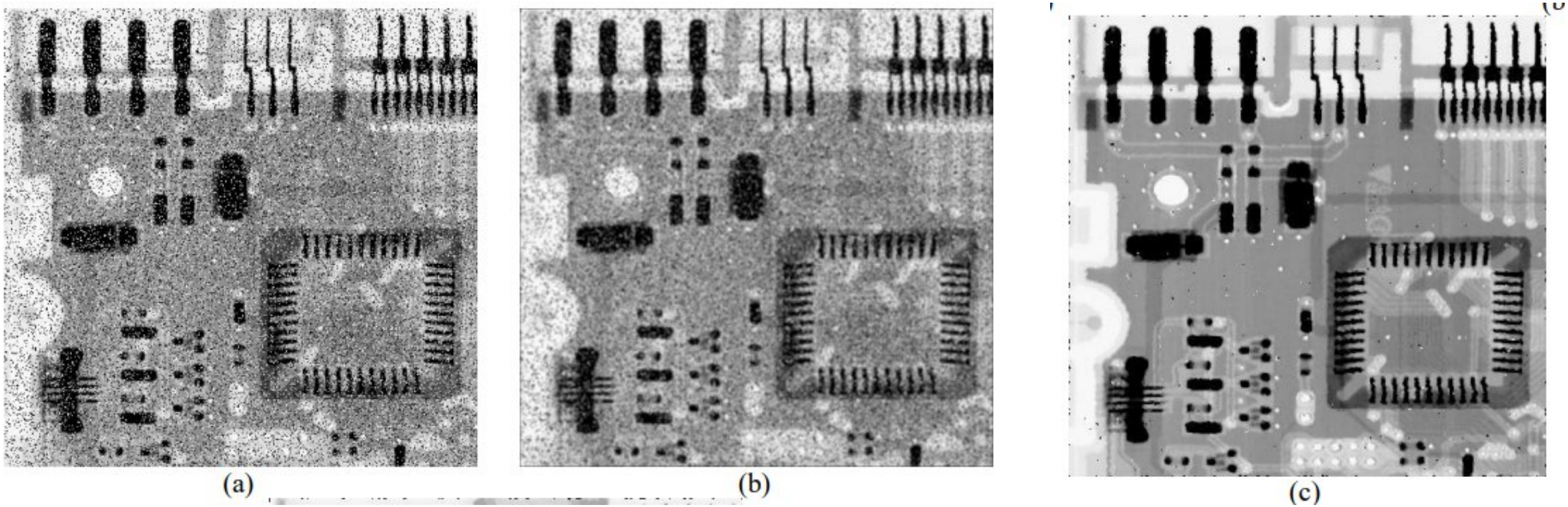
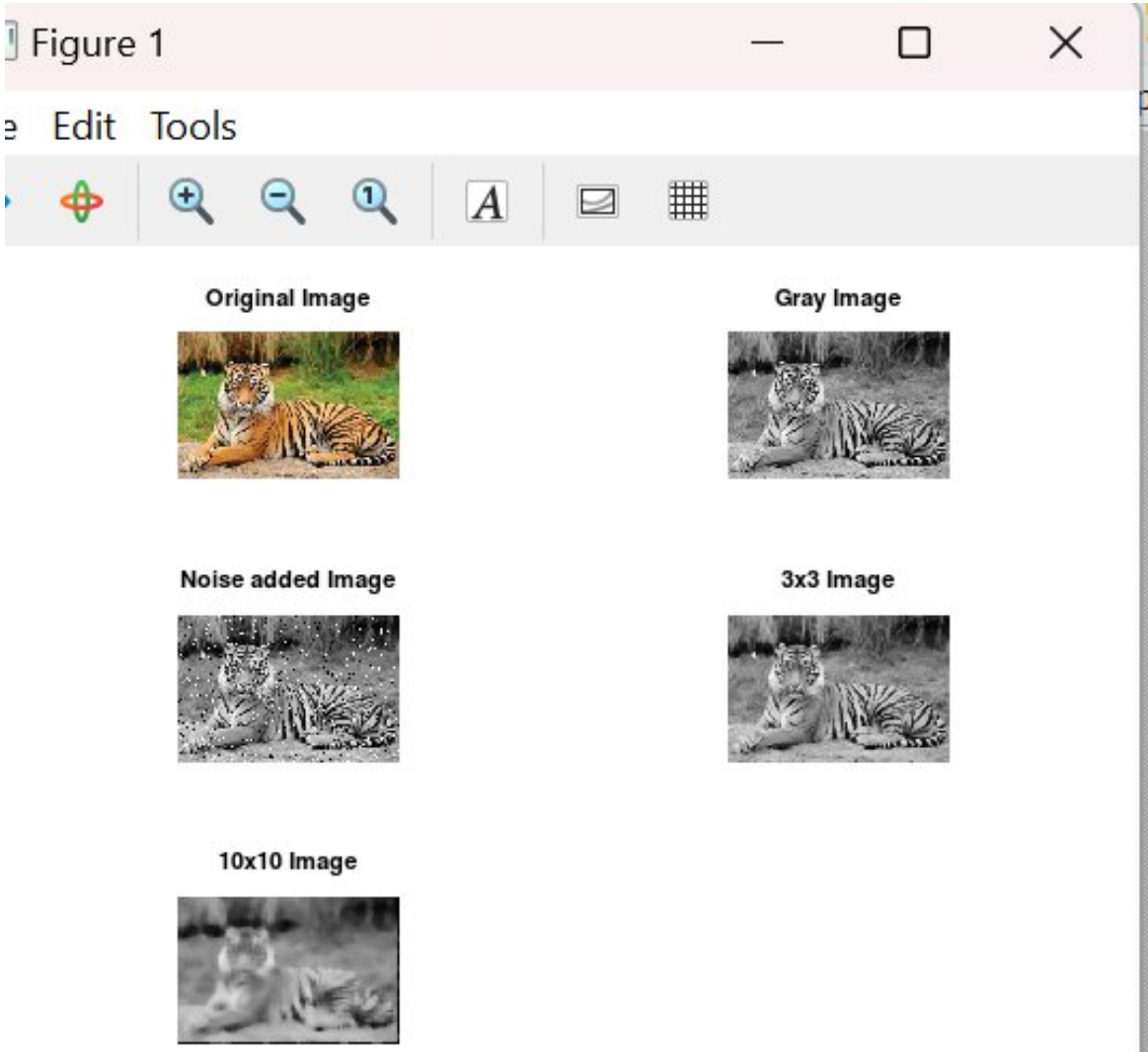


Figure 6.3 Effect of median filter. (a) Image corrupted by salt & pepper noise. (b) Result of applying 3×3 standard averaging filter on (a). (c) Result of applying 3×3 median filter on (a).

Lab 08: Median Filter



plane.m | negative_linear_transformation.m | histogram.m | FFT

```
clear all; % clear all variables
close all; % close all figures
clc;       % clear command window
% import image package
pkg load image;
% read image
% Median Filters
I=imread('tiger.jpg');
K = rgb2gray(I);
J= imnoise(K ,'salt & pepper',0.05);
f= medfilt2(J,[3,3]);
f1=medfilt2(J,[10,10]);
subplot(3,2,1); imshow(I); title('Original Image');
subplot(3,2,2); imshow(K); title('Gray Image');
subplot(3,2,3); imshow(J); title('Noise added Image');
subplot(3,2,4); imshow(f); title('3x3 Image');
subplot(3,2,5); imshow(f1); title('10x10 Image');
```

First Derivative Vs Second Derivative Filters

(Edge detection & image enhancement in Digital Image Processing)

1 First Derivative Filters

What is First Derivative?

The **first derivative** of an image measures the **rate of change of gray level intensity**.

It becomes **large at edges**, where intensity changes suddenly.

Mathematically:

$$\frac{\partial f(x,y)}{\partial x}, \quad \frac{\partial f(x,y)}{\partial y}$$

In DIP, the **gradient** is the first derivative.

Characteristics of First Derivative Filters

- *Zero in constant intensity regions*
- *Large at edges*
- *Directional (detects edge orientation)*
- *Produces thick edges*
- *Less sensitive to fine noise than second derivative*

If pixel values change as:

```
10  10  10
10  10  10
10  200 200
```

The sudden jump from 10 → 200 results in a large gradient, hence an edge.

2️⃣ Second Derivative Filters

What is Second Derivative?

The **second derivative** measures the **rate of change of the first derivative**.

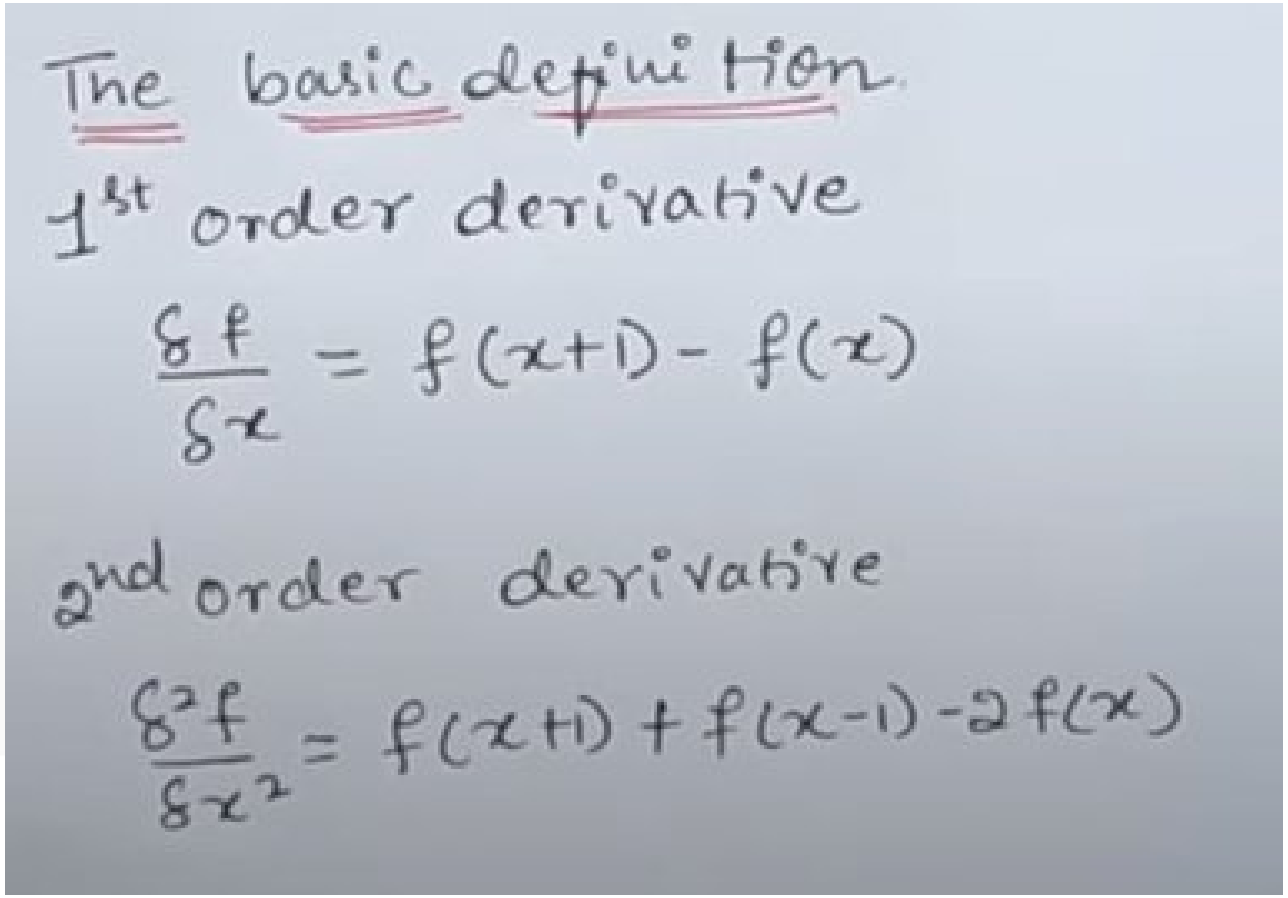
It highlights regions where the intensity **changes rapidly twice**, such as **thin edges and fine details**.

Mathematically:

$$\frac{\partial^2 f(x,y)}{\partial x^2}, \quad \frac{\partial^2 f(x,y)}{\partial y^2}$$

Characteristics of Second Derivative Filters

- *Zero in constant regions*
- *Strong response at edges*
- *Produces thin edges*
- *Very sensitive to noise*
- *Edges detected using zero-crossings*



If intensity changes smoothly:

```
10 → 50 → 100 → 150
```

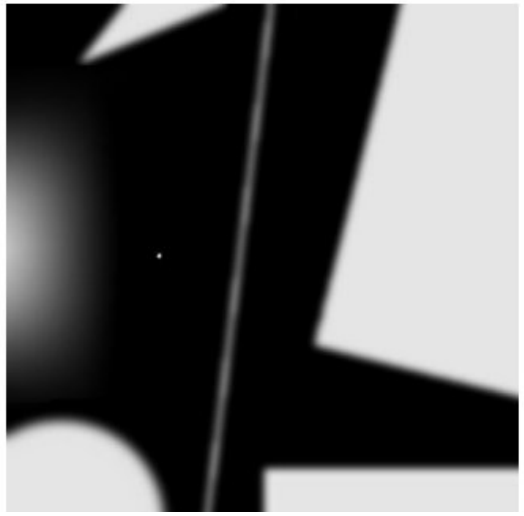
Second derivative is small.

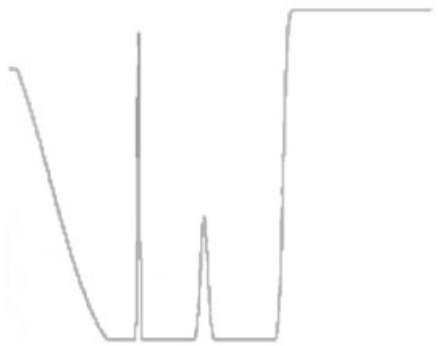
If intensity changes abruptly:

```
10 → 10 → 200 → 10
```

Second derivative is very high.

Consider the example below:





We conclude that:

- 1st derivative detects thick edges while 2nd derivative detects thin edges.
- 2nd derivative has much stronger response at gray-level step than 1st derivative.

Thus, we can expect a second-order derivative to enhance fine detail (thin lines, edges, including noise) much more than a first-order derivative.

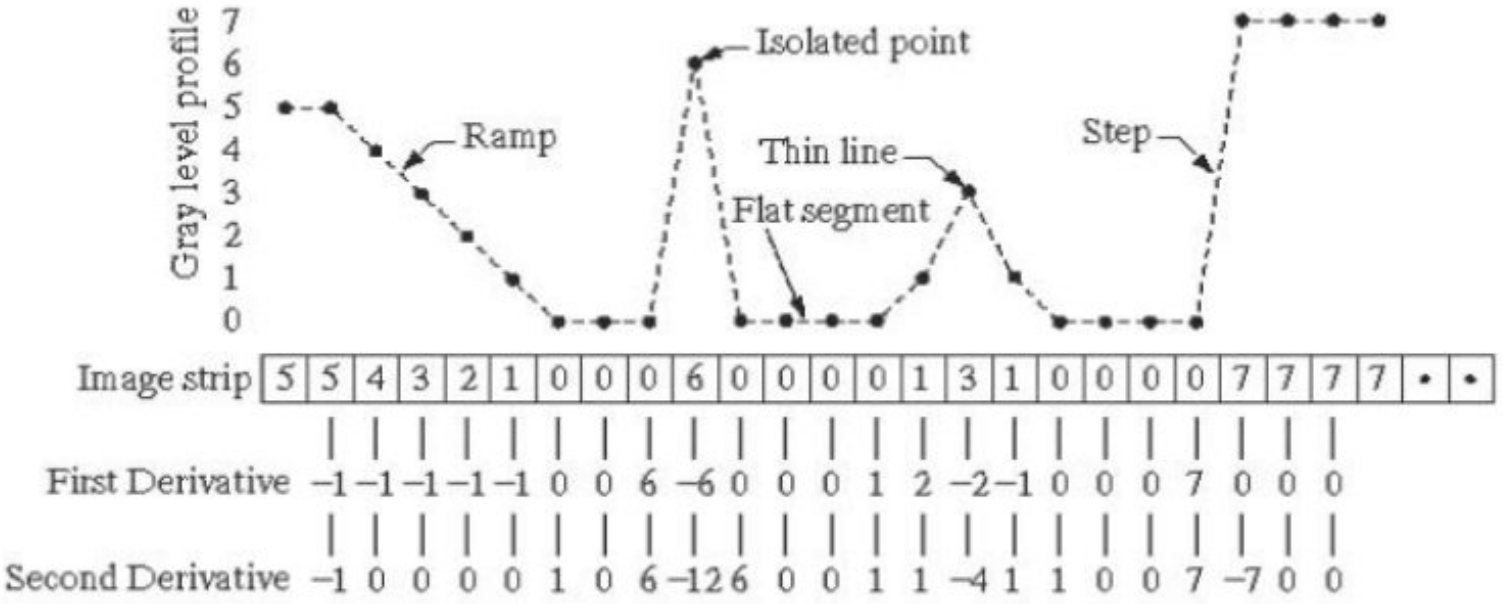


Figure 6.4 Example of partial derivatives

Second figure is the one dimensional horizontal line of the image. And last figure is the simplified representation of second figure

In 1D: values are 0 on flat surfaces; non-zero on slope or ramp. So thicker edges.
In Isolated point, the response of 2D(i.e. -12) is much stronger than 1D (-6).

In 2D: values are 0 on flat surfaces; non-zero on starting and ending point of ramp. So finer edges. In Step, 1D & 2D remains same (i.e. 7)

Gradient Based Filters (Edge Detection) (1st Derivative)

Gradient-Based Filters (Edge Detection)

Gradient-based filters are **first-order derivative operators** used in **Digital Image Processing (DIP)** to detect **edges**.

Edges correspond to **sudden changes in gray level intensity**, and the gradient measures how fast the intensity changes in the **x and y directions**.

Basic Idea of Gradient

For an image $f(x, y)$:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Gradient magnitude:

$$|\nabla f| = \sqrt{G_x^2 + G_y^2}$$

(Practically approximated as $|G_x| + |G_y|$)

- **Large gradient value → Edge**
- **Small gradient value → Smooth region**

1 Roberts Cross Operator

Description

- Oldest and simplest gradient operator
- Uses **2×2 masks**
- Detects edges along **diagonals**
- Sensitive to noise

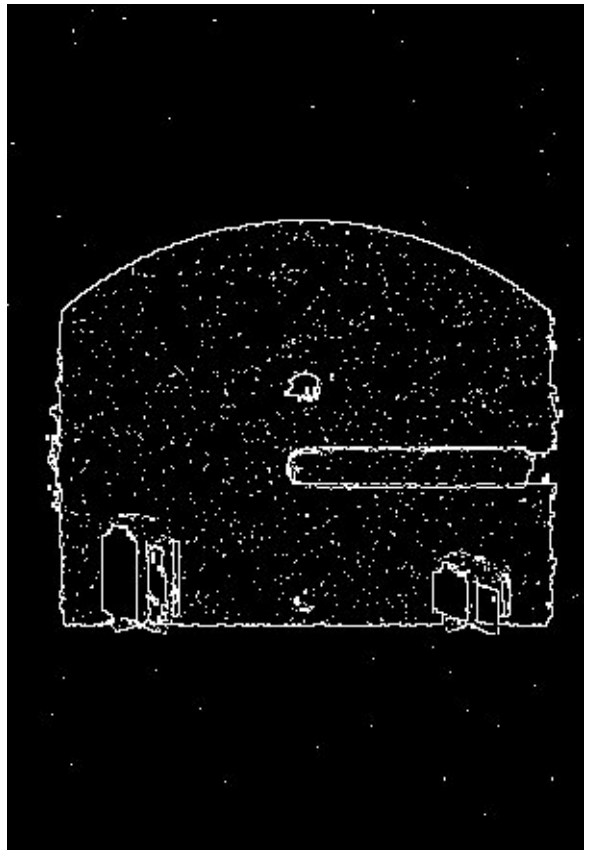
Masks

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Gradient Approximation

$$|G| = |G_x| + |G_y|$$

Example: If intensity changes sharply between diagonal pixels, Roberts operator highlights that location as an edge.



Advantages

- Simple and fast
- Low computation

Disadvantages

- Very noise-sensitive
- Poor edge localization for larger images

2️⃣ Prewitt Operator

Description

- Uses **3×3 masks**
- Detects **horizontal and vertical edges**
- Slight smoothing effect → less noise than Roberts

Masks

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Example

- Vertical edges → highlighted by G_x
- Horizontal edges → highlighted by G_y

Advantages

- Simple
- Better noise resistance than Roberts

Disadvantages

- Not optimal for detecting diagonal edges
- Edge thickness is slightly larger

3 Sobel Operator

Description

- Most commonly used gradient filter
- Uses **3×3 masks with weighting**
- Emphasizes central pixels → **better noise suppression**

Masks

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Example

Sobel produces **strong and thick edges**, suitable for real-world images.

Advantages

- Better noise reduction
- Strong edge detection
- Widely used in practice

Disadvantages

- Slightly more computation than Prewitt

origional image

Final Image

Orginal Image

Canny

Laplacian of Gaussian

Roberts

Sobel

v=0.8

Original Image (3×3 window)

Assume a small part of an image:

$$I = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 50 & 50 \\ 10 & 50 & 50 \end{bmatrix}$$

There is a clear **vertical edge** between low (10) and high (50).

1️⃣ Roberts Operator (2×2)

Masks

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Take top-left 2×2 pixels:

$$\begin{bmatrix} 10 & 10 \\ 10 & 50 \end{bmatrix}$$

Compute G_x

$$(10 \times 1) + (10 \times 0) + (10 \times 0) + (50 \times -1) = 10 - 50 = -40$$

Compute G_y

$$(10 \times 0) + (10 \times 1) + (10 \times -1) + (50 \times 0) = 10 - 10 = 0$$

Gradient Magnitude

$$|G| = |-40| + |0| = 40$$

✅ Edge detected

2️⃣ Prewitt Operator (3×3)

Masks

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Apply G_x on the image

$$\begin{aligned} &(-1 \times 10) + (0 \times 10) + (1 \times 10) \\ &+(-1 \times 10) + (0 \times 50) + (1 \times 50) \\ &+(-1 \times 10) + (0 \times 50) + (1 \times 50) \end{aligned}$$

$$= (0) + (40) + (40) = 80$$

Apply G_y

$$(10 + 10 + 10) - (10 + 50 + 50) = 30 - 110 = -80$$

Gradient Magnitude

$$|G| = |80| + |-80| = 160$$

✅ Strong edge detected

3️⃣ Sobel Operator (3×3)

Masks

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Apply G_x

$$\begin{aligned} &(-1 \times 10) + (0 \times 10) + (1 \times 10) \\ &+(-2 \times 10) + (0 \times 50) + (2 \times 50) \\ &+(-1 \times 10) + (0 \times 50) + (1 \times 50) \end{aligned}$$

$$= 0 + 80 + 40 = 120$$

Apply G_y

$$(10 + 20 + 10) - (10 + 100 + 50) = 40 - 160 = -120$$

Gradient Magnitude

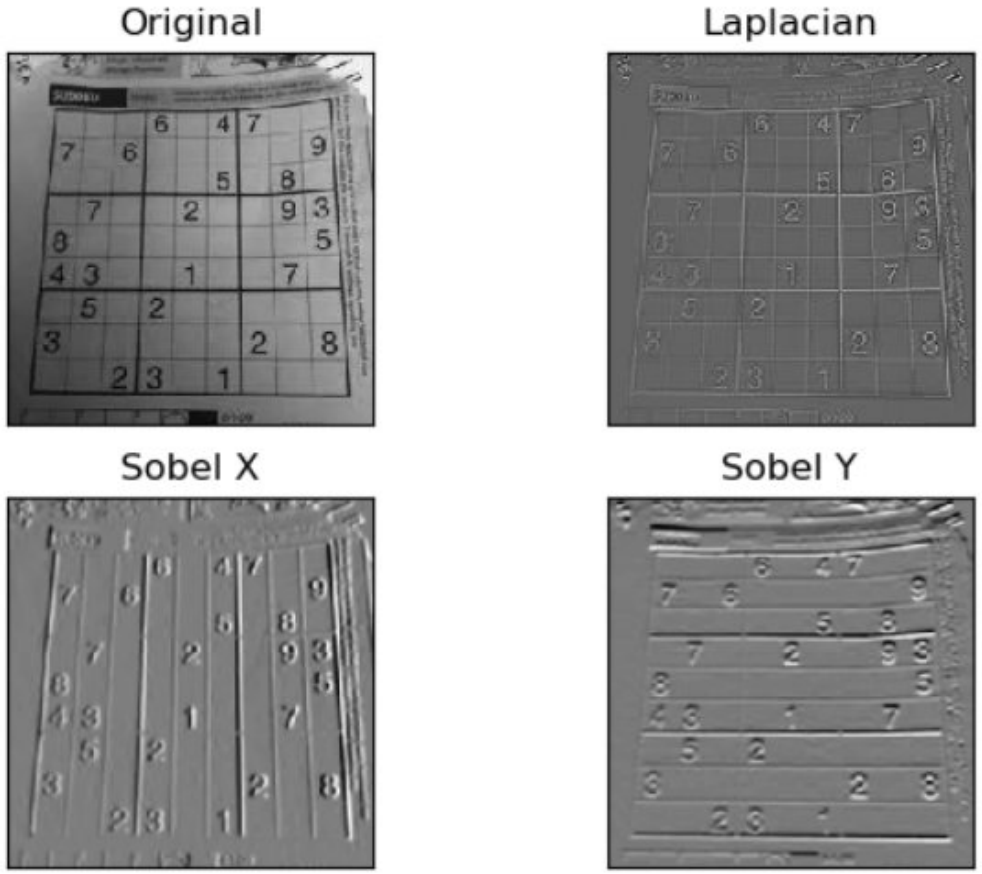
$$|G| = |120| + |-120| = 240$$

✅ Very strong and smooth edge

Laplacian Filters (2nd Derivative)

The **Laplacian filter** is a **second-order derivative operator** that detects edges by calculating the **sum of second derivatives** in both x and y directions.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$



Common Laplacian Masks

4-neighbor Laplacian

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

8-neighbor Laplacian

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

1 Derivation of Laplacian Filter Equation

The **Laplacian** is a **second-order derivative operator** used for **edge detection and image sharpening**.

Continuous Form

For an image $f(x, y)$, the Laplacian is defined as:

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

Discrete Approximation (Digital Image)

Using second-order finite differences:

$$\frac{\partial^2 f}{\partial x^2} \approx f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} \approx f(x, y+1) + f(x, y-1) - 2f(x, y)$$

Combine Both Directions

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

2 Laplacian Mask Derivation

From the above equation, the **4-neighbour Laplacian mask** is obtained:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

If diagonal neighbors are included, the **8-neighbour Laplacian mask** is:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

3 Algorithm for Laplacian Filter Implementation

Step-by-Step Algorithm

1. Read the input image $f(x, y)$
2. Convert image to **grayscale** (if needed)
3. Choose Laplacian mask (4- or 8-neighbour)
4. Apply convolution between image and Laplacian mask
5. Obtain Laplacian image $\nabla^2 f(x, y)$
6. (For sharpening) compute:

$$g(x, y) = f(x, y) - \nabla^2 f(x, y)$$

7. Normalize the output image
8. Display the result

Given Image (3×3)

$$I = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 50 & 10 \\ 10 & 10 & 10 \end{bmatrix}$$

Laplacian Mask (4-neighbour)

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Convolution at Center Pixel

$$\begin{gathered}(0 \times 10) + (-1 \times 10) + (0 \times 10) \\ +(-1 \times 10) + (4 \times 50) + (-1 \times 10) \\ +(0 \times 10) + (-1 \times 10) + (0 \times 10)\end{gathered}$$

$$= -10 - 10 + 200 - 10 - 10 = 160$$

Result

- Laplacian output at center = **160**
- Large value ⇒ **edge present**

Image Sharpening

$$g(x, y) = f(x, y) - \nabla^2 f(x, y)$$

$$g(2, 2) = 50 - 160 = -110$$

Image Magnification (Zooming)

Image magnification is the process of **increasing the spatial size (resolution)** of an image so that objects appear **larger**, without changing the scene content.

When we magnify an image, **new pixel values must be estimated** — this is done using **interpolation techniques**.

Why Interpolation is Needed

After magnification, the new image grid has more pixels than the original. Since those pixel values do not exist, they must be computed from neighboring pixels.

1️⃣ Nearest Neighbor Interpolation (Pixel Replication)

Idea

- Each new pixel takes the value of the **closest original pixel**
- Simplest and fastest method

Example (2× Magnification)

Original (2×2):

$$\begin{bmatrix} 10 & 50 \\ 80 & 120 \end{bmatrix}$$

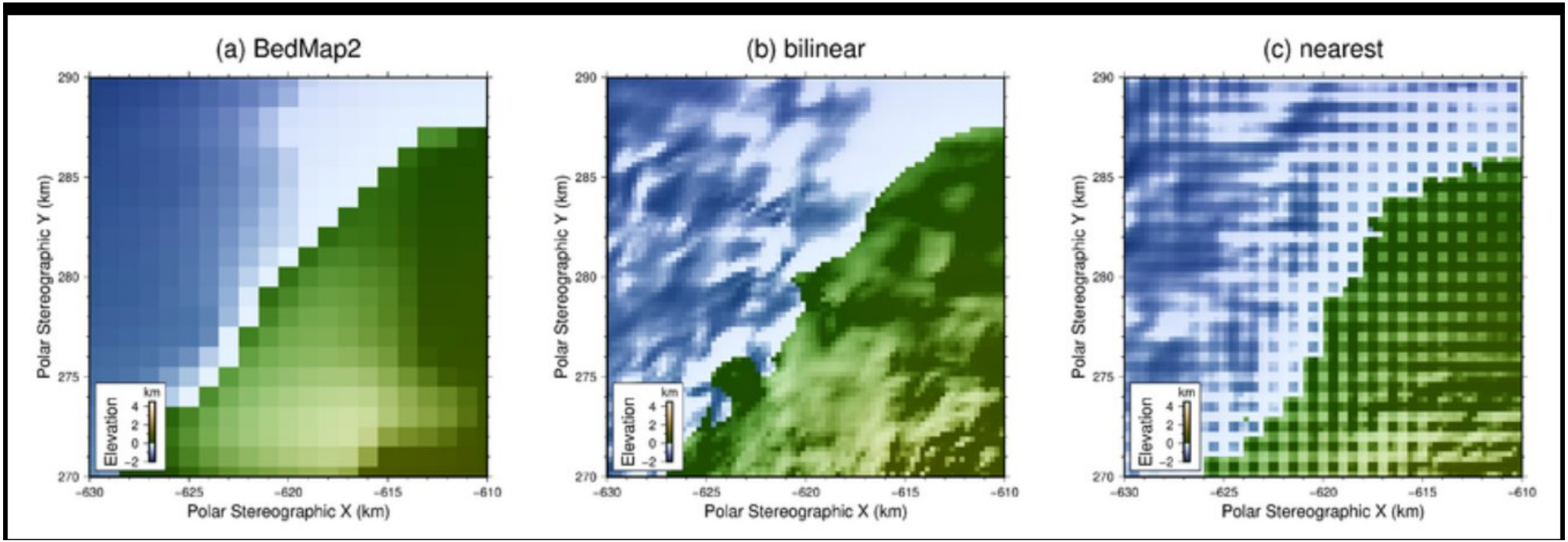
After 2× magnification:

$$\begin{bmatrix} 10 & 10 & 50 & 50 \\ 10 & 10 & 50 & 50 \\ 80 & 80 & 120 & 120 \\ 80 & 80 & 120 & 120 \end{bmatrix}$$

2 Bilinear Interpolation

In **bilinear interpolation**:

- New pixels are created during magnification
- Each new pixel value is calculated using the **weighted average of 4 nearest neighboring pixels**
- This produces a **smooth transition** between pixels (no blocky effect)



$$I = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

- Horizontal interpolation → average left & right
- Vertical interpolation → average top & bottom
- Center pixel → average of all 4

Step A: Copy original pixels to corners

```
10   ?   ?   20
 ?   ?   ?    ?
 ?   ?   ?    ?
30   ?   ?   40
```

Step B: Horizontal interpolation

```
10   15   15   20
 ?    ?    ?    ?
 ?    ?    ?    ?
30   35   35   40
```

Step C: Vertical interpolation

```
10   15   15   20
20    ?    ?   30
20    ?    ?   30
30   35   35   40
```

Step D: Center pixels (average of 4 neighbors)

$$(10 + 20 + 30 + 40)/4 = 25$$

$$\begin{bmatrix} 10 & 15 & 15 & 20 \\ 20 & 25 & 25 & 30 \\ 20 & 25 & 25 & 30 \\ 30 & 35 & 35 & 40 \end{bmatrix}$$

1. Explain "power law transformation" techniques for the purpose of image enhancement. Explain the mean filter along with suitable algorithm for its implementation(4+6)

5. Discuss the algorithm for histogram equalization.

6. Explain the first derivative filter with a suitable example.

11. Discuss the magnification of image using interpolation technique.

3. What is an edge detection filter? Differentiate between the first derivative and second derivative filter? Derive the filter mask for Laplacian filter and write the algorithm for its implementation. [1+2+7=10]

4. Explain the term Contrast Stretching and the Histogram Equalization.(3+1)

10. Derive the equation for Laplacian filter and write the algorithm for its implementation.(6)

4. Explain the Bit plane slicing technique for image enhancement.