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 using Spatial Domain

- Image Enhancement is a process that improves the quality of an image for a particular application. Image enhancement is needed
  - To highlight the important details
  - To remove noise in the image

Different methods for image enhancement

1. **Spatial Domain:** Direct Manipulation of pixel values
2. **Frequency Domain:** Modifying the Fourier Transform of the image. First Spatial domain is converted to frequency domain using Fourier transform and then inverse transform is applied to convert back to spatial domain.
3. **Combination Method:** combination of both Spatial Domain and Frequency Domain.

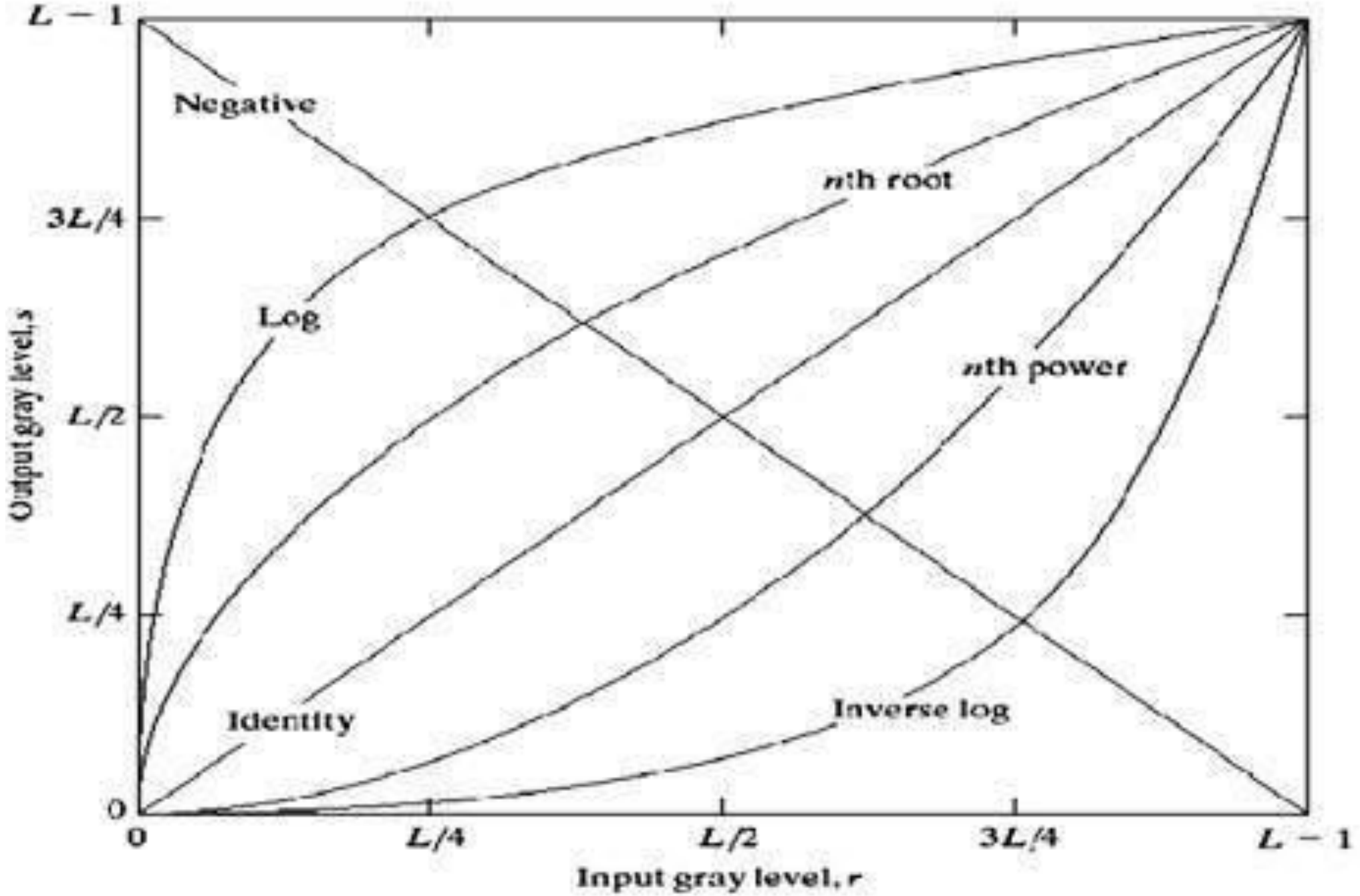
Basic Gray Level Transformations

- Image enhancement can be done through gray level transformations
- Enhancing an image provides better contrast and a more detailed image as compare to non enhanced image. Image enhancement has very applications. It is used to enhance medical images, images captured in remote sensing, images from satellite e.t.c
- The transformation function has been given below
- s = T (r)

where r is the pixels of the input image and s is the pixels of the output image. T is a transformation function that maps each value of r to each value of s.

- There are three basic gray level transformation.

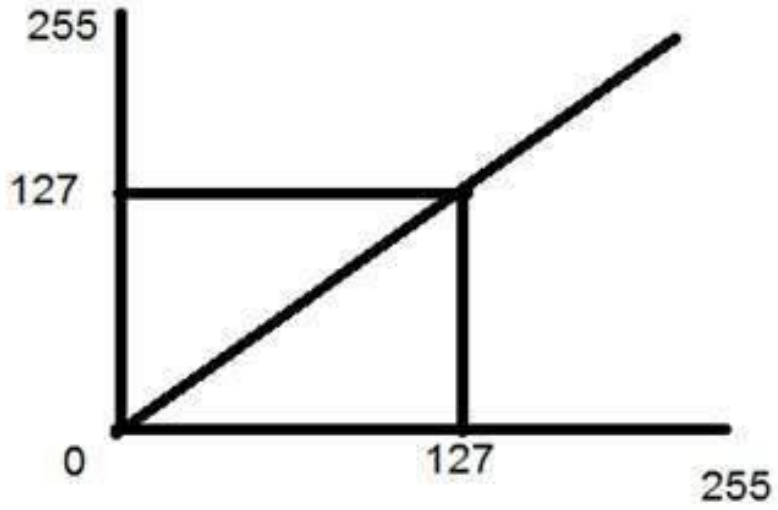
1. Linear
2. Logarithmic
3. Power – law



0 represents black
L-1 represents White

1. Linear transformation (identity & Negative)

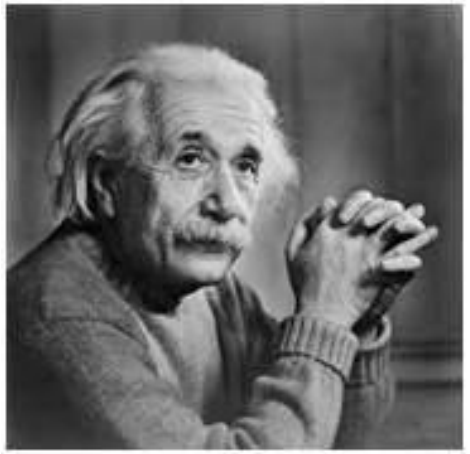
- Linear transformation includes simple identity and negative transformation.
- Identity transition is shown by a straight line. In this transition, each value of the input image is directly mapped to each other value of output image. That results in the same input image and output image. And hence is called identity transformation.
- It is very less used because same type of image output. If L/4 is mapped (i.e. 256/4), then same L/4 image is the output. For white image, white image is the output



Negative transformation

- The second linear transformation is negative transformation, which is invert of identity transformation. In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image. The result is somewhat like this.

Input Image



Output Image

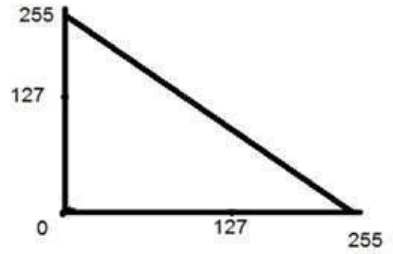
In this case the following transition has been done.

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

since the input image of Einstein is an 8 bpp image, so the number of levels in this image are 256. i.e. 2 power 8
Putting 256 in the equation, we get this

$$s = 255 - r$$

So each value is subtracted by 255 and the result image has been shown above. So what happens is that, the lighter pixels become dark and the darker picture becomes light. And it results in image negative. It has been shown in the graph below.



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

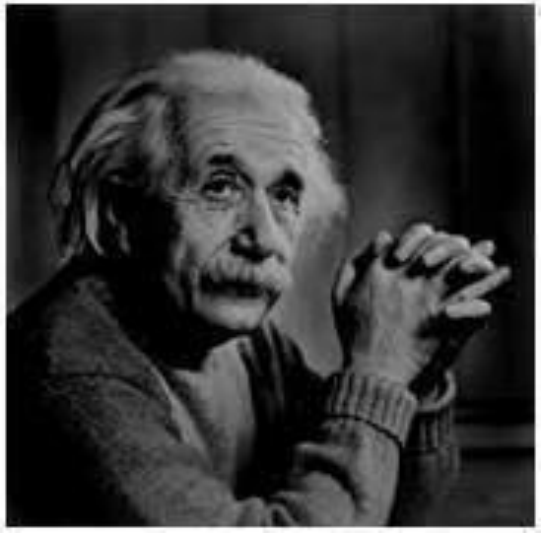
output image

75, 166.33)

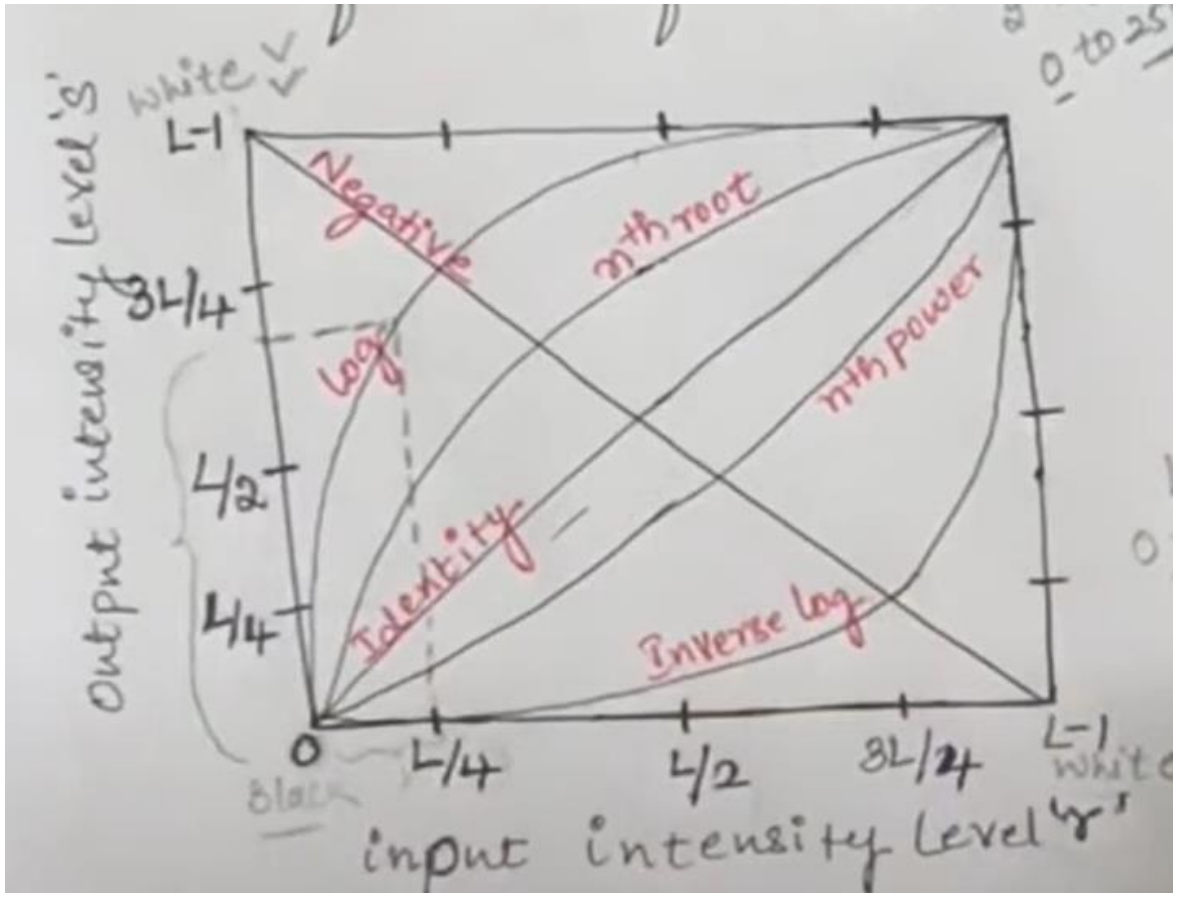
2. Logarithmic transformations (Log & Inverse Log)

- The log transformations can be defined by this formula
- $s = c \log(r + 1)$.
- 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. This result in following image enhancement.
- The value of c in the log transform adjust the kind of enhancement you are looking for.

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:

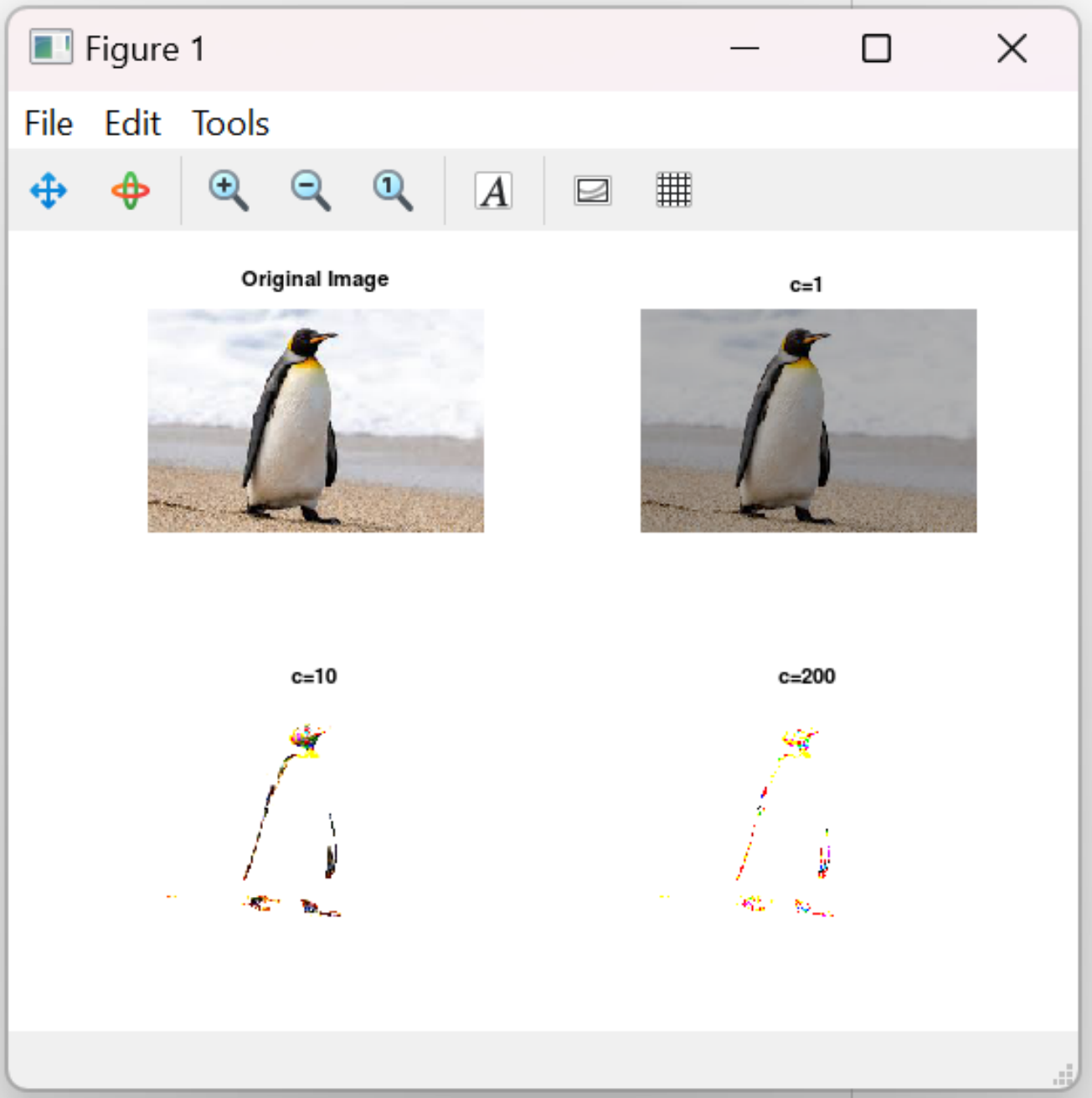
Similarly, for nearly 3L/4 input image, output is L/4 i.e. for high amount of image, less amount of image is the output. So it will produce less contrast image.

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';
```



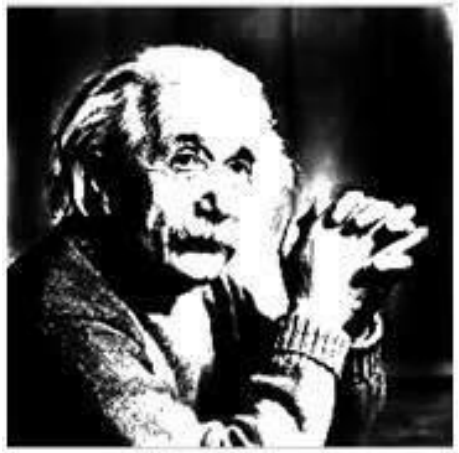
3. Power – Law transformations

- There are further two transformation is power law transformations, that include nth power and nth root transformation. These transformations can be given by the expression:

  $s=cr^{\gamma}$

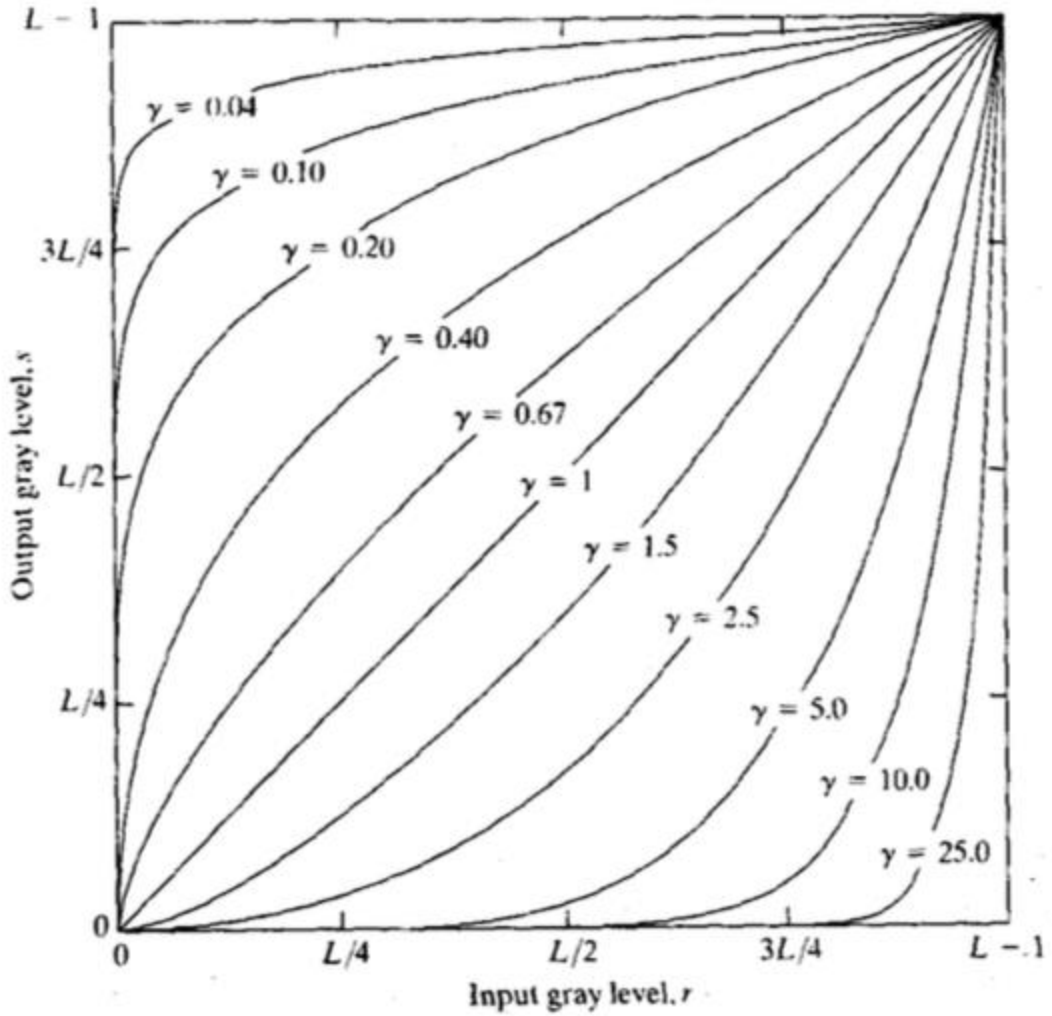
- This symbol γ is called gamma, due to which this transformation is also known as gamma transformation.
- Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity.
- This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different. For example Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.

Gamma = 10



Gamma = 8

Gamma = 6



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

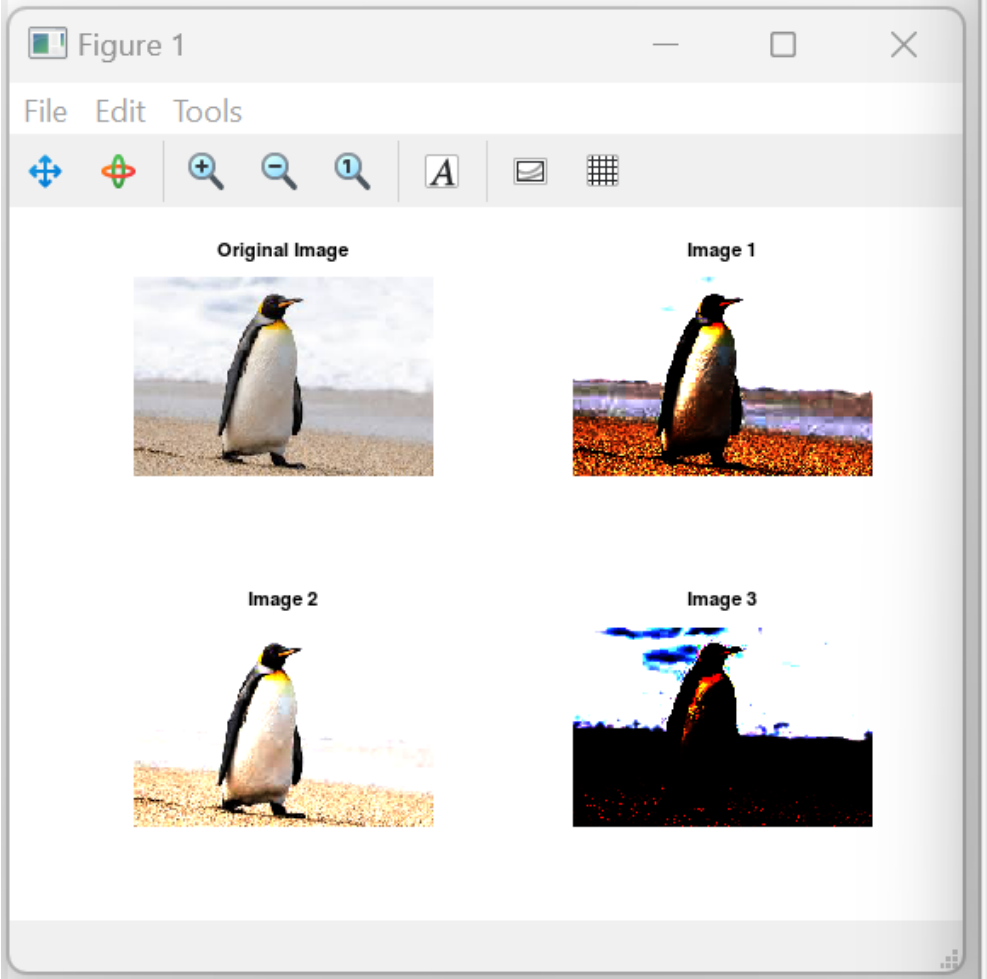
For same input L/4, the output will be different depending upon the value of gamma.

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';
```

	Linear	Logarithmic	Power-law
Linearity	Linear relation between input and output pixels	A non-linear relation between input and output pixels	A non-linear relation between input and output pixels
Changes	Can change contrast and brightness	Enhances details in darker regions	Can selectively enhance or compress different intensity ranges
Applications	Brightness and contrast adjustments	Used for increasing visibility in dim areas	Used for fine-tuning contrast for specific details

Piecewise Linear Transformations

Piece-wise Linear Transformation is type of gray level transformation that is used for image enhancement. It is a spatial domain method. It is used for manipulation of an image so that the result is more suitable than the original for a specific application. Some commonly used piece-wise linear transformations are:

1. Contrast Stretching:

Low contrast image occur often due to improper illumination or non-linearly or small dynamic range of an imaging sensor. It increases the dynamic range of grey levels in the image. Contrast Stretching Transform is given by:

In Contrast Stretching, darker images are made more darker and bright portion are made more brighter.

Contrast stretching (often called normalization) is a simple image enhancement technique that attempts to improve the contrast in an image by `stretching' the range of intensity values it contains to span a desired range of values, e.g. the the full range of pixel values that the image type concerned allows.

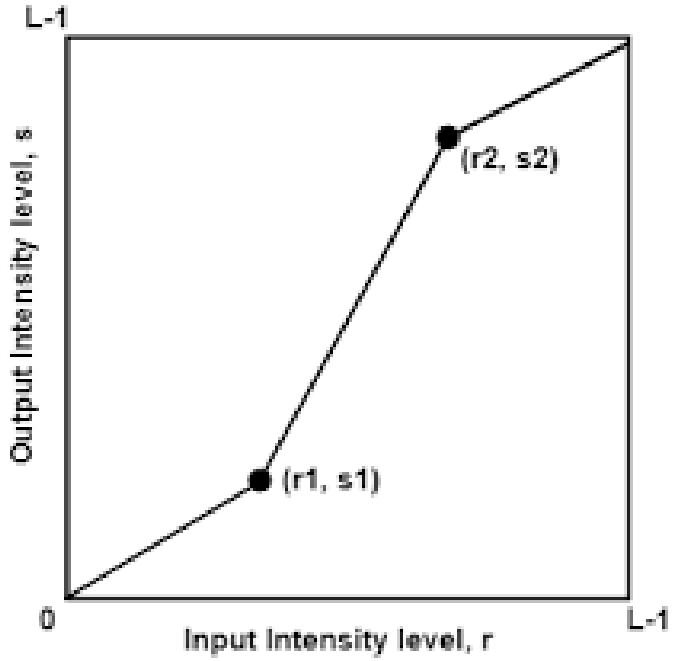


Fig. 2(a). Original Image Fig. 2(b). Contrast Stretched Image

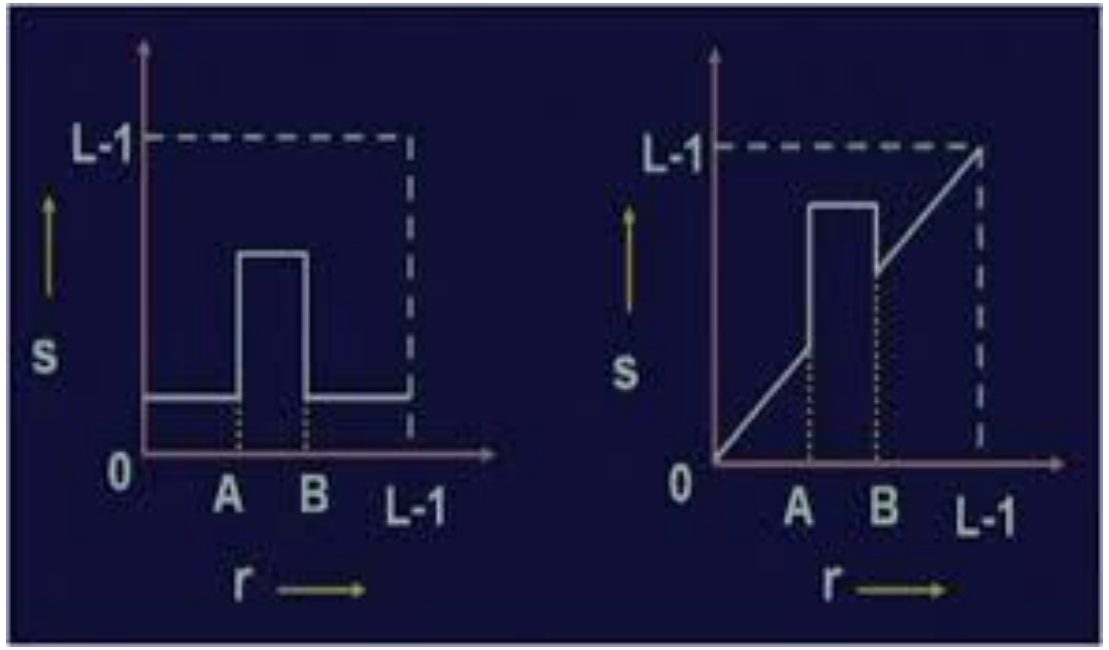
730 ×

Almost invisible image is stretched
& in next image, thresholding

2. Grey level slicing:

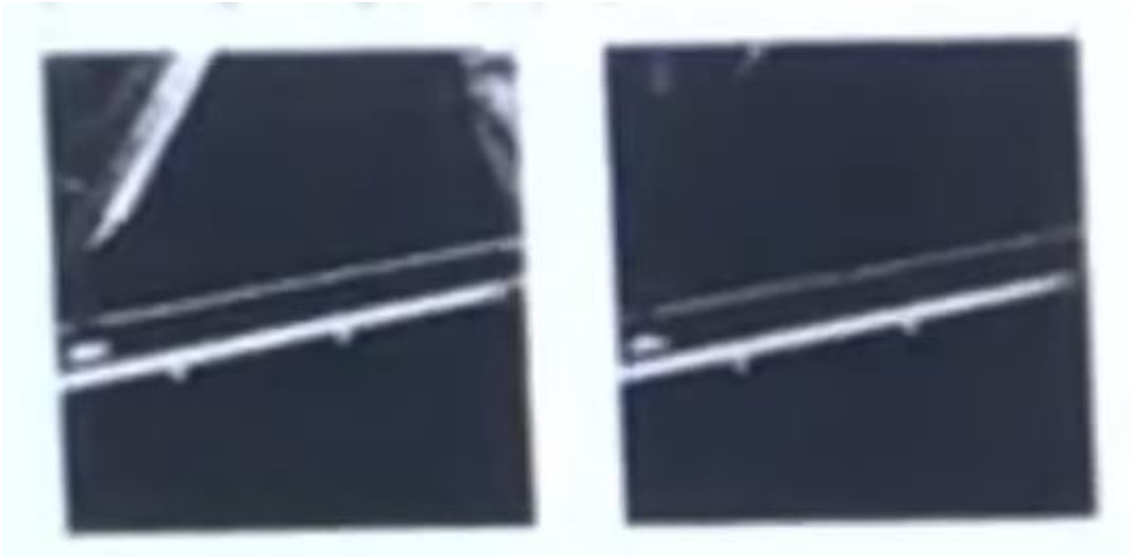
Highlighting a specific range of grey level in an image.

- **Case-I:**
  - To display a high value for all grey levels in the range of interest.
  - To display a low value for all grey levels.
- **Case-II:**
  - Brighten the desired range of grey level.
  - Preserve the background quality in the range.



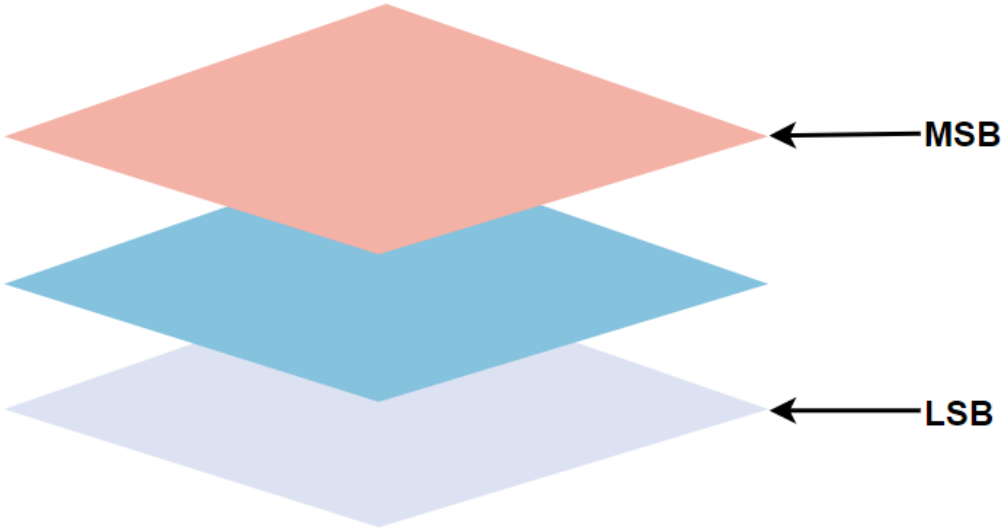
The first approach, shown in first image, produces high value for region A to B while remaining region will be low value. So it produces binary images.

The second approach, shown in second image, produces high value for region A to B while remaining region will be unchanged. So it only highlights the region focused, while others are 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

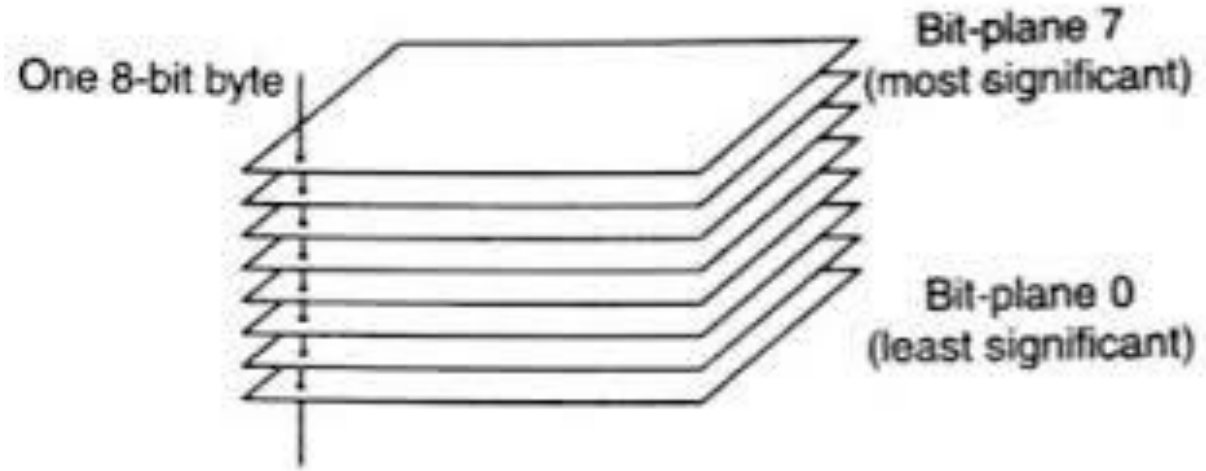


Fig no: 7

the orginal

image of bit-1

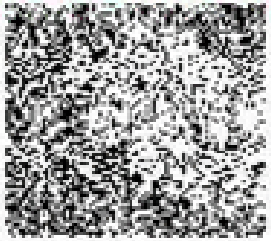


image of bit-2

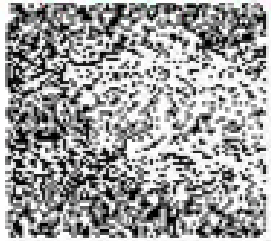


image of bit-3

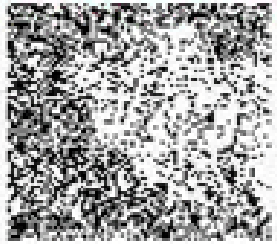


image of bit-4

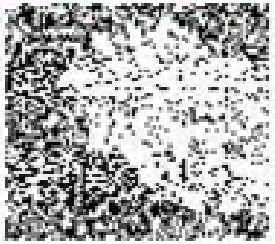


image of bit-5

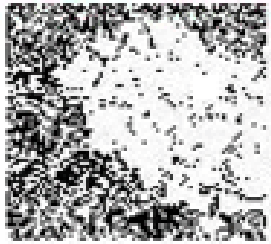


image of bit-6

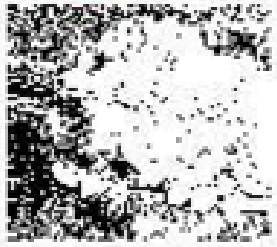


image of bit-7

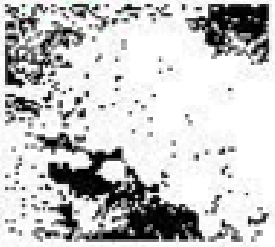
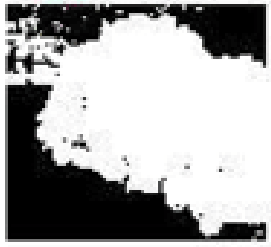
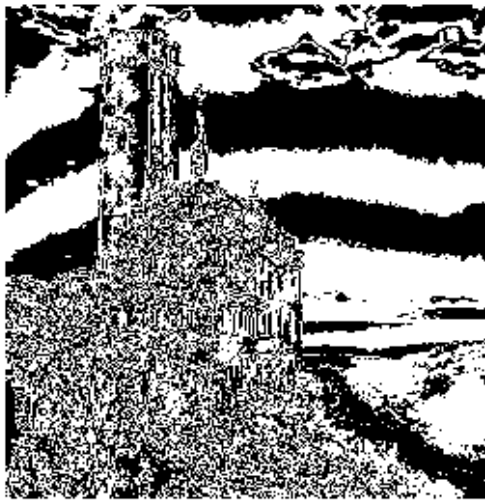
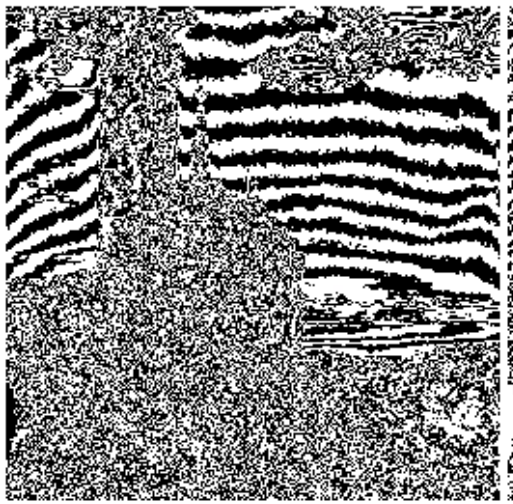
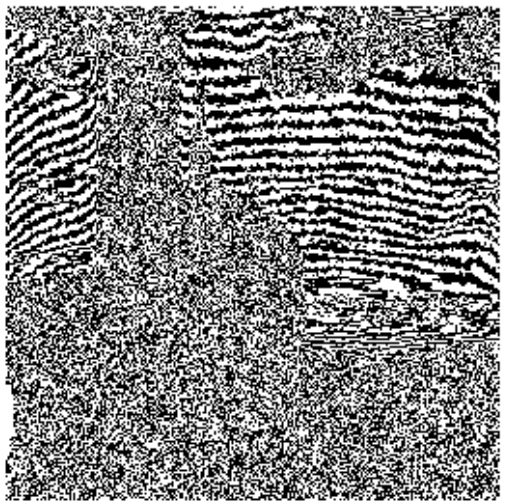


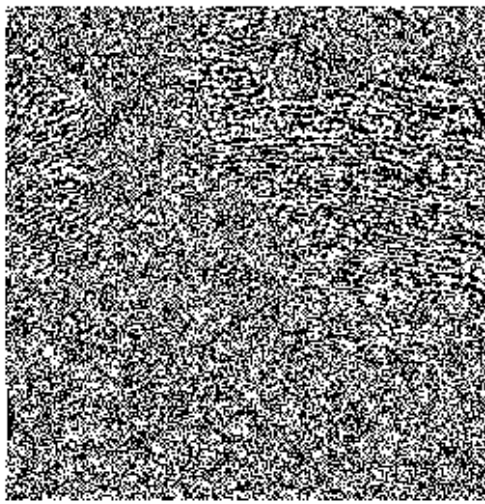
image of bit-8







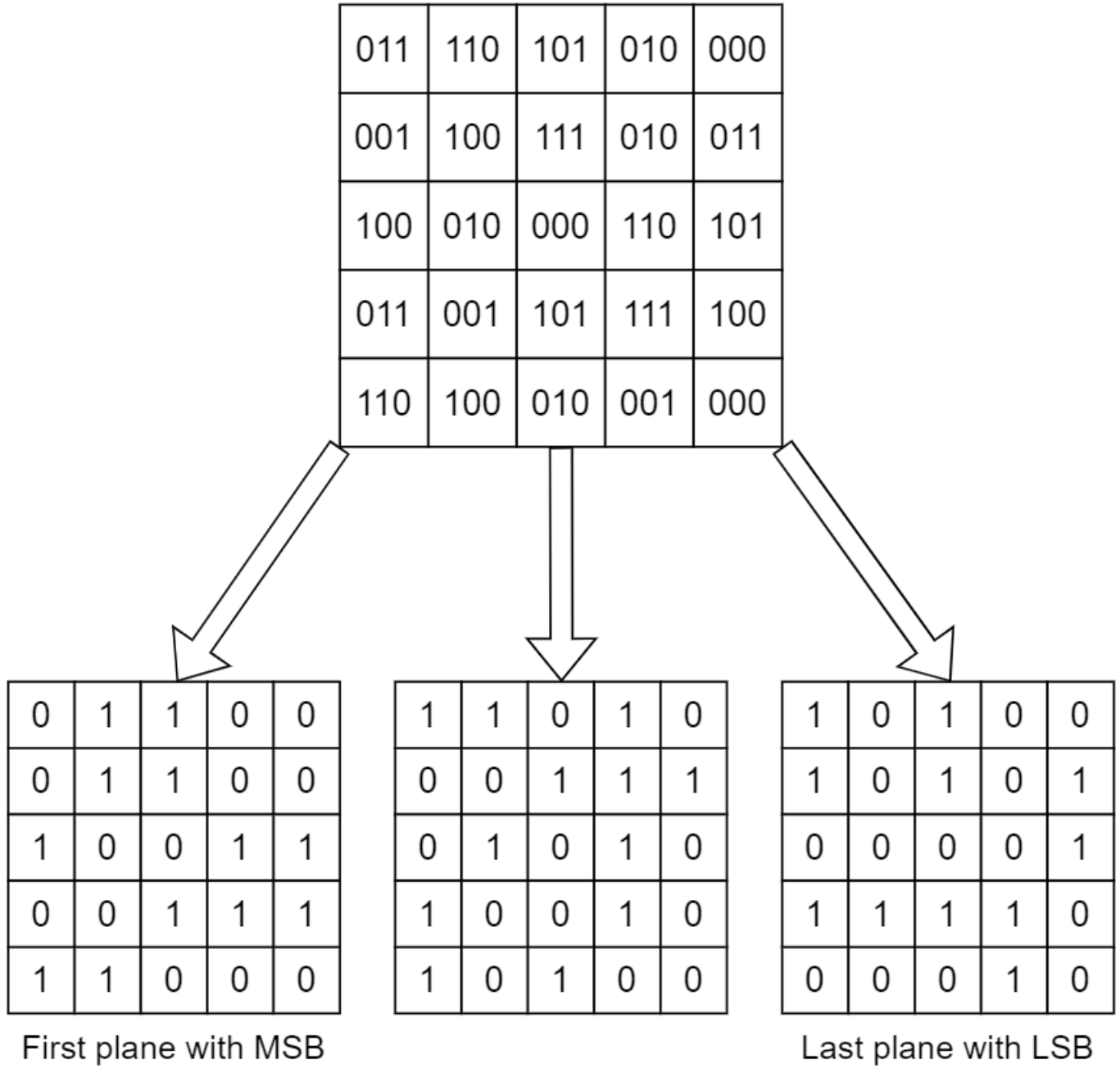




1. **Convert to binary**: Convert the value of each pixel in the image to its binary representation.
2. **Bit-plane extraction**: The number of planes will equal the number of bits used to represent pixel values. Each bit is assigned to a separate plane. The MSB is assigned to the first-bit plane and 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

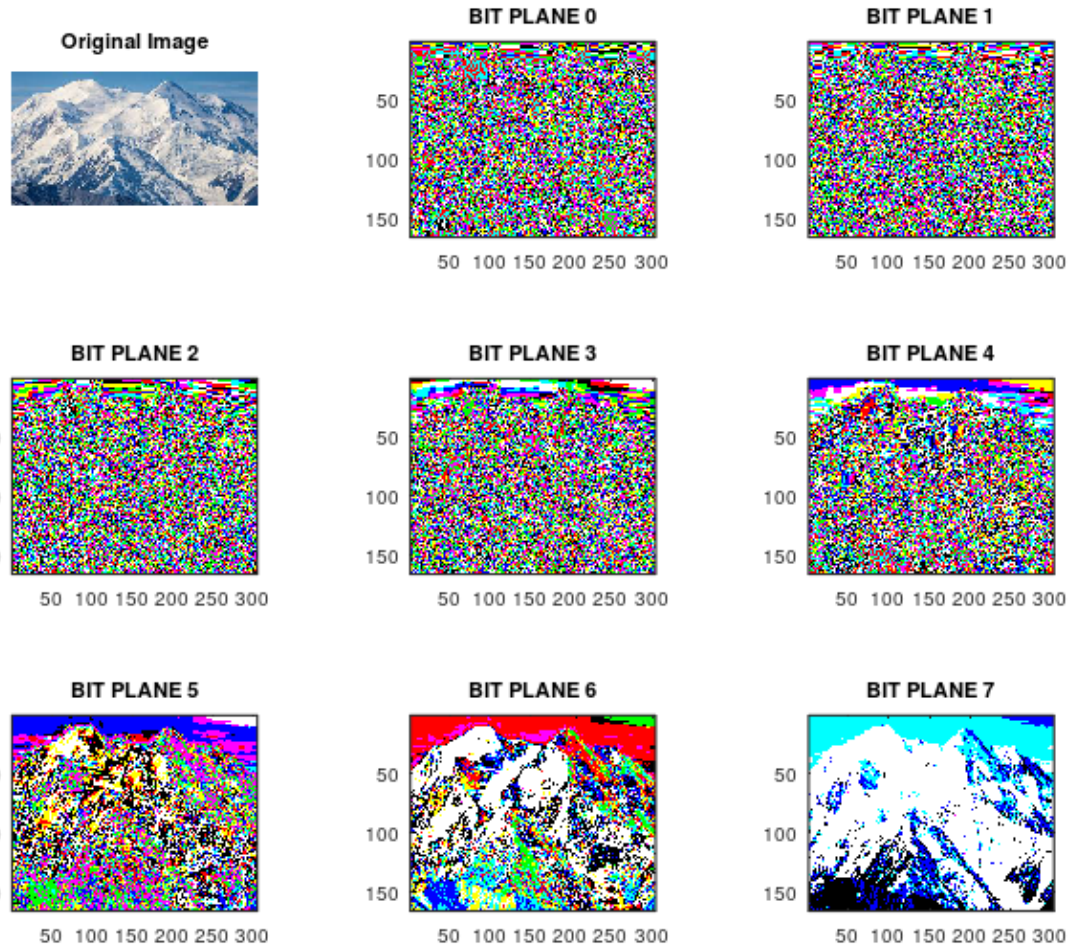
bit_plane.m | negative_linear_transformation.m

```
clear all; % clear all variables
close all; % close all figures
clc;       % clear command window
% import image package
pkg load image;
% read image
i=imread("mountain.png");
b0=double(bitget(i,1));
b1=double(bitget(i,2));
b2=double(bitget(i,3));
b3=double(bitget(i,4));
b4=double(bitget(i,5));
b5=double(bitget(i,6));
b6=double(bitget(i,7));
b7=double(bitget(i,8));
subplot(3,3,1);imshow(i);title('Original Image');
subplot(3,3,2);subimage(b0);title('BIT PLANE 0');
subplot(3,3,3);subimage(b1);title('BIT PLANE 1');
subplot(3,3,4);subimage(b2);title('BIT PLANE 2');
subplot(3,3,5);subimage(b3);title('BIT PLANE 3');
subplot(3,3,6);subimage(b4);title('BIT PLANE 4');
subplot(3,3,7);subimage(b5);title('BIT PLANE 5');
subplot(3,3,8);subimage(b6);title('BIT PLANE 6');
subplot(3,3,9);subimage(b7);title('BIT PLANE 7');

```

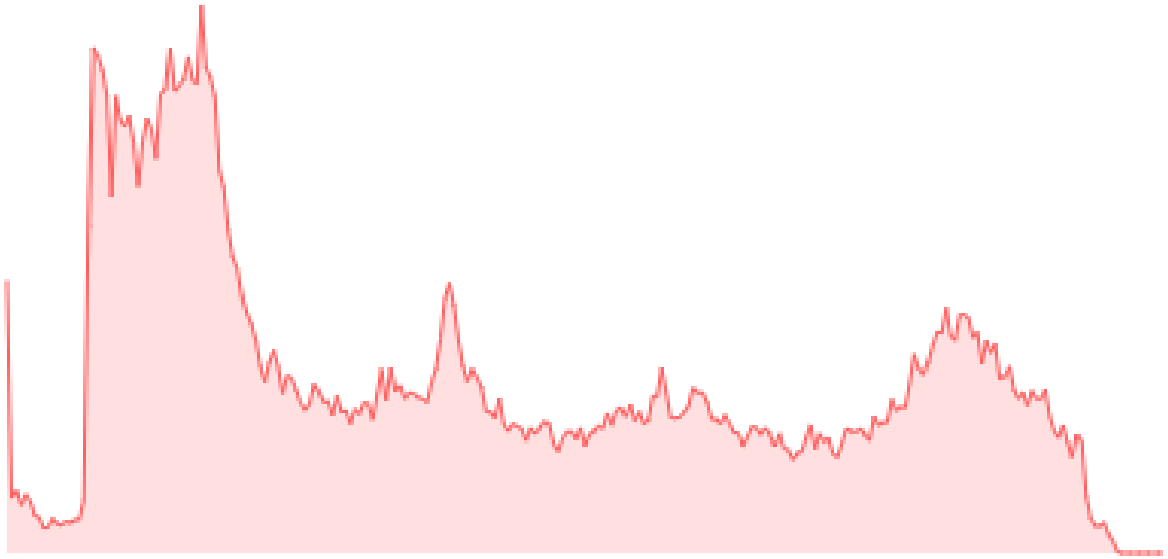
Figure 1

File Edit Tools



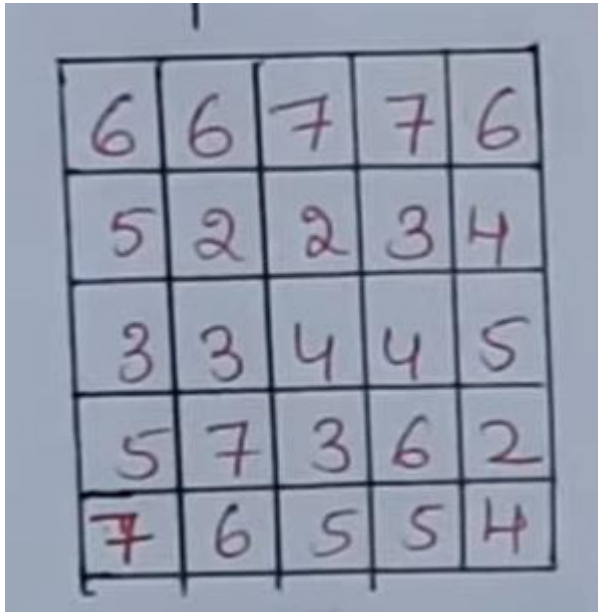
Histogram Processing

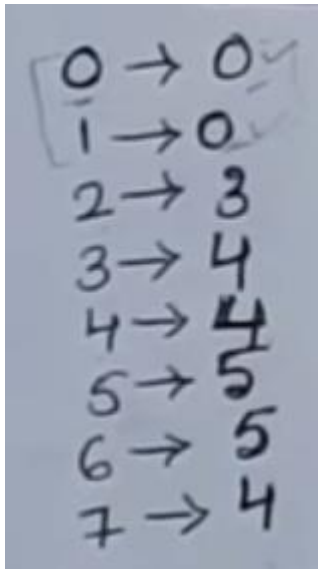
- In digital image processing, the histogram is used for graphical representation of a digital image. A graph is a plot by the number of pixels for each tonal value. Nowadays, image histogram is present in digital cameras. Photographers use them to see the distribution of tones captured.
- In a graph, the horizontal axis of the graph is used to represent tonal variations whereas the vertical axis is used to represent the number of pixels in that particular pixel. Black and dark areas are represented in the left side of the horizontal axis, medium grey color is represented in the middle, and the vertical axis represents the size of the area.

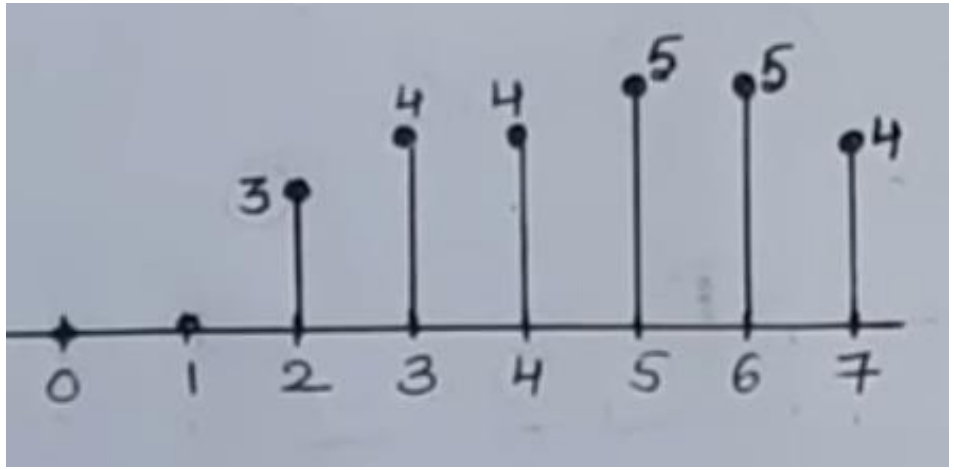


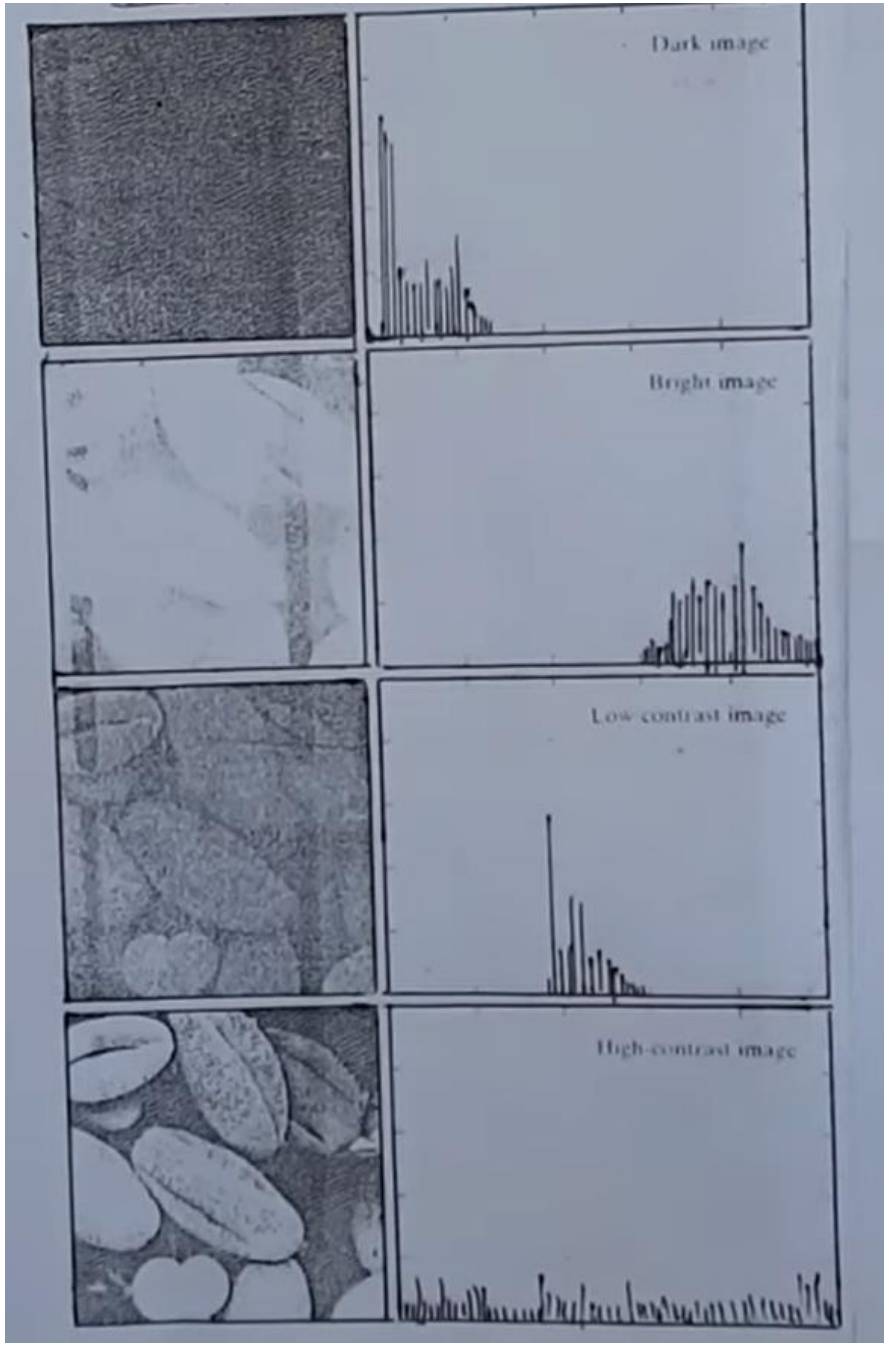
Histogram of the above scenery

- Histogram is the graphical representation of data, which is used for image enhancement
- It represents the frequency of occurrence of various gray levels.
- It is used for manipulating contrast and brightness.
- To get a high quality image, the histogram should be distributed over a place or in a flat profile.









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

Applications of Histograms

1. In digital image processing, histograms are used for simple calculations in software.
2. It is used to analyze an image. Properties of an image can be predicted by the detailed study of the histogram.
3. The brightness of the image can be adjusted by having the details of its histogram.
4. The contrast of the image can be adjusted according to the need by having details of the x-axis of a histogram.
5. It is used for image equalization. Gray level intensities are expanded along the x-axis to produce a high contrast image.
6. Histograms are used in thresholding as it improves the appearance of the image.
7. If we have input and output histogram of an image, we can determine which type of transformation is applied in the algorithm.

Histogram Processing Techniques

1. Histogram Sliding:

In Histogram sliding, the complete histogram is shifted towards rightwards or leftwards. When a histogram is shifted towards the right or left, clear changes are seen in the brightness of the image. The brightness of the image is defined by the intensity of light which is emitted by a particular light source.

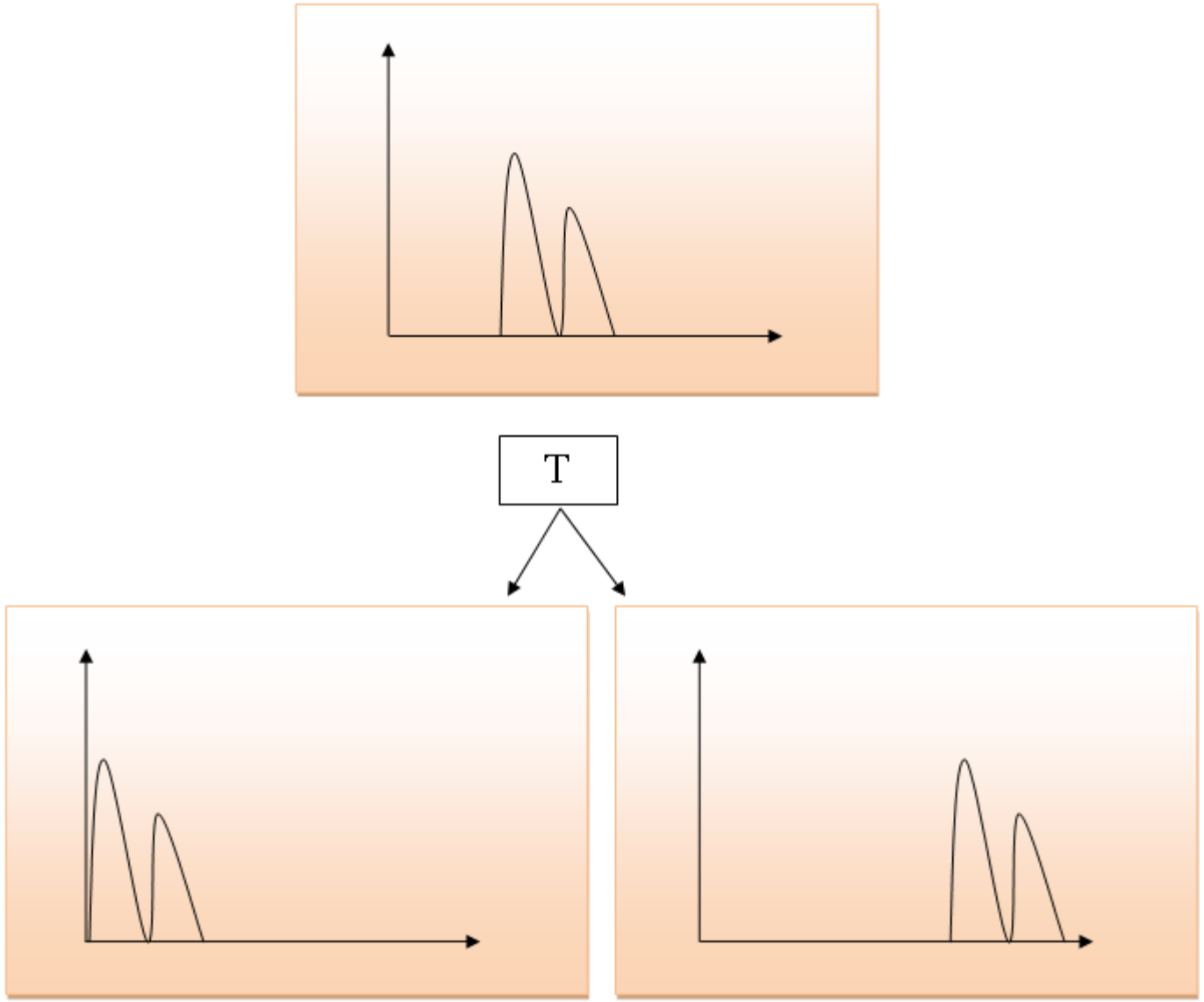
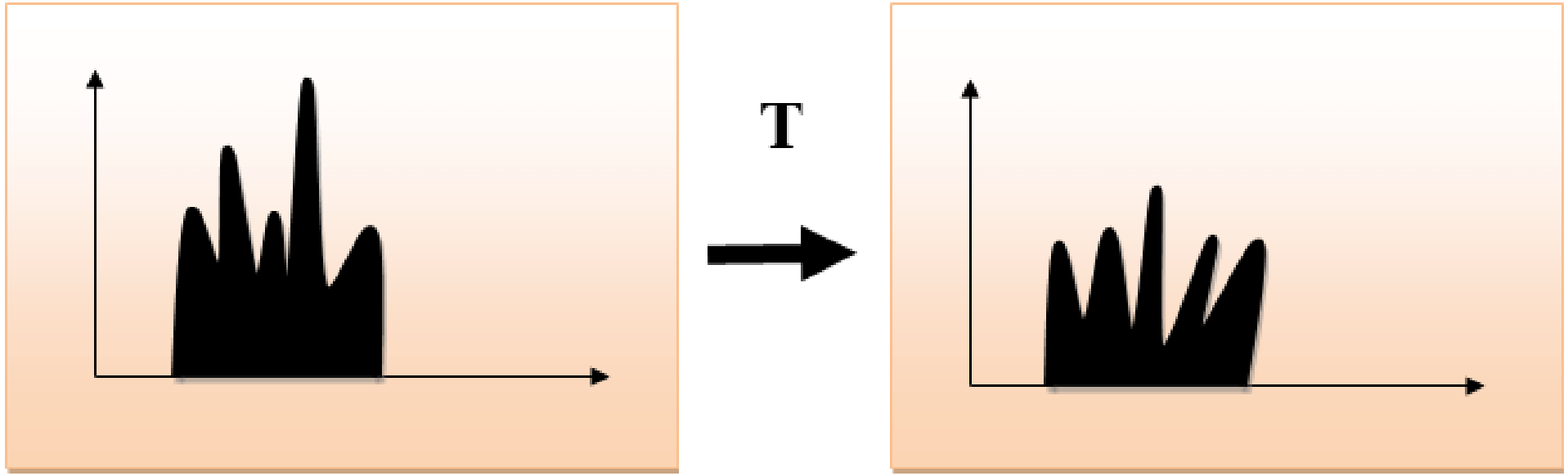


Fig. Histogram Sliding

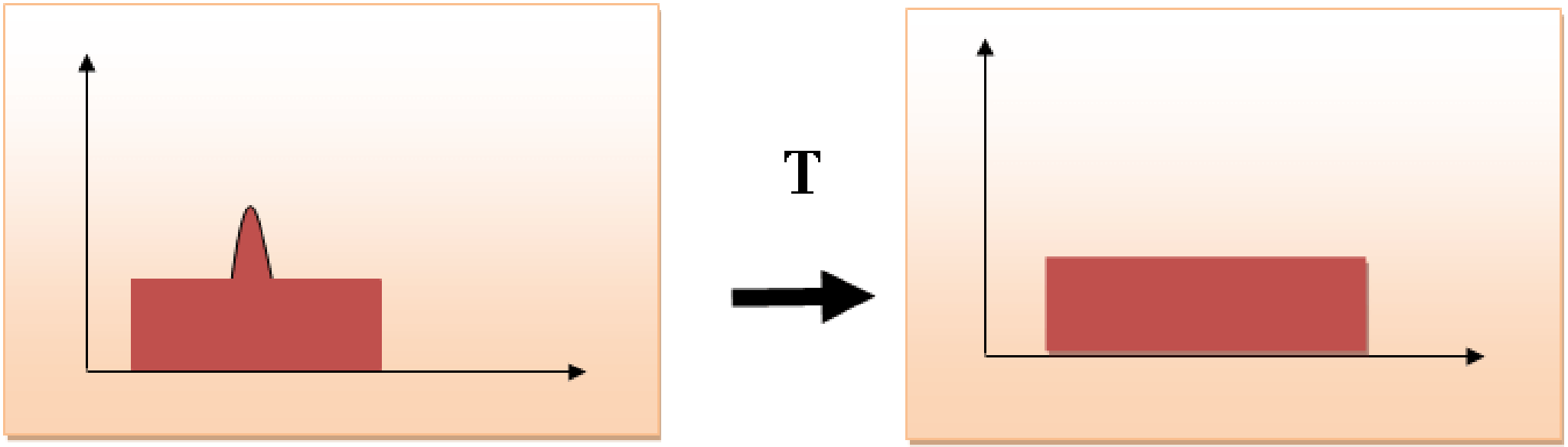
2. Histogram Stretching

- In histogram stretching, contrast of an image is increased. The contrast of an image is defined between the maximum and minimum value of pixel intensity.
- If we want to increase the contrast of an image, histogram of that image will be fully stretched and covered the dynamic range of the histogram.
- From histogram of an image, we can check that the image has low or high contrast.



3. Histogram Equalization

- Histogram equalization is used for equalizing all the pixel values of an image. Transformation is done in such a way that uniform flattened histogram is produced.
- Histogram equalization increases the dynamic range of pixel values and makes an equal count of pixels at each level which produces a flat histogram with high contrast image.
- While stretching histogram, the shape of histogram remains the same whereas in Histogram equalization, the shape of histogram changes and it generates only one image.



Discuss the algorithm for histogram equalizations

Histogram equalization is one of the Pixel brightness transformations techniques. It is a well-known contrast enhancement technique due to its performance on almost all types of image.

Steps Involved

1. Get the input image
2. Generate the histogram for the image
3. Find the local minima of the image
4. Divide the histogram based on the local minima
5. Have the specific gray levels for each partition of the histogram
6. Apply the histogram equalization on each partition

Algorithm

- Compute the histogram of pixel values of the input image. The histogram places the value of each pixel $f[x,y]$ into one of L uniformly-spaced buckets $h[i]$

Where L=2^8 and the image dimension is $M \times N$

- Calculate the cumulative distribution function
- Scale the input image using the cumulative distribution function to produce the output image.

Where CDFmin is the smallest non-zero value of the cumulative distribution function

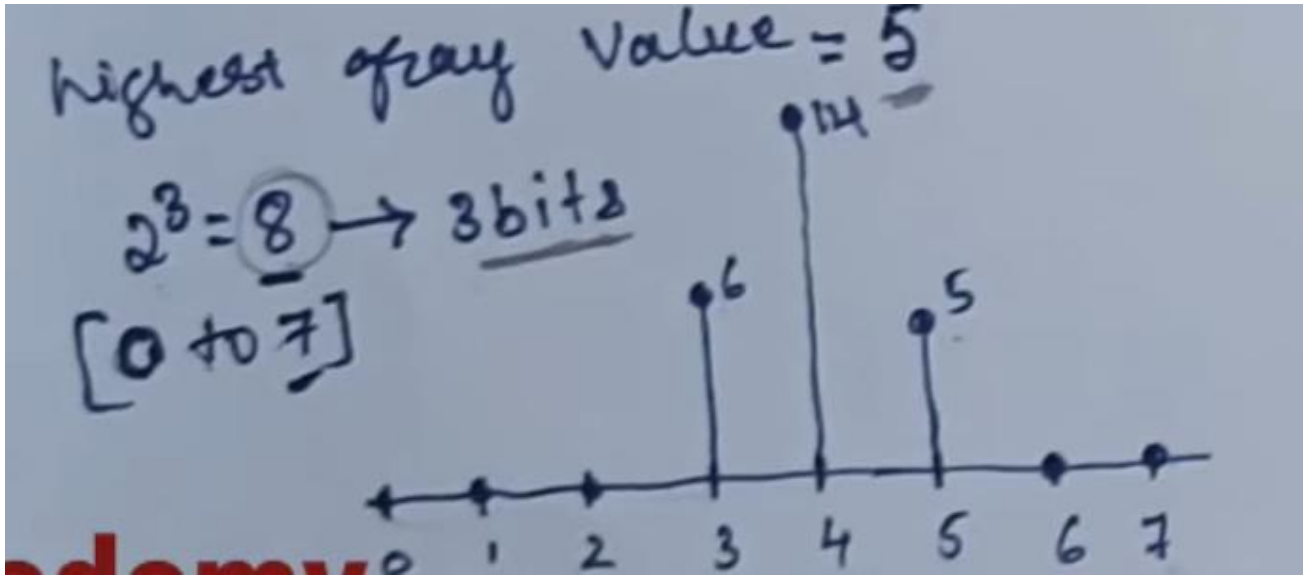
Example: Perform the histogram equalization of following pixel values

$f(x,y)=$

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

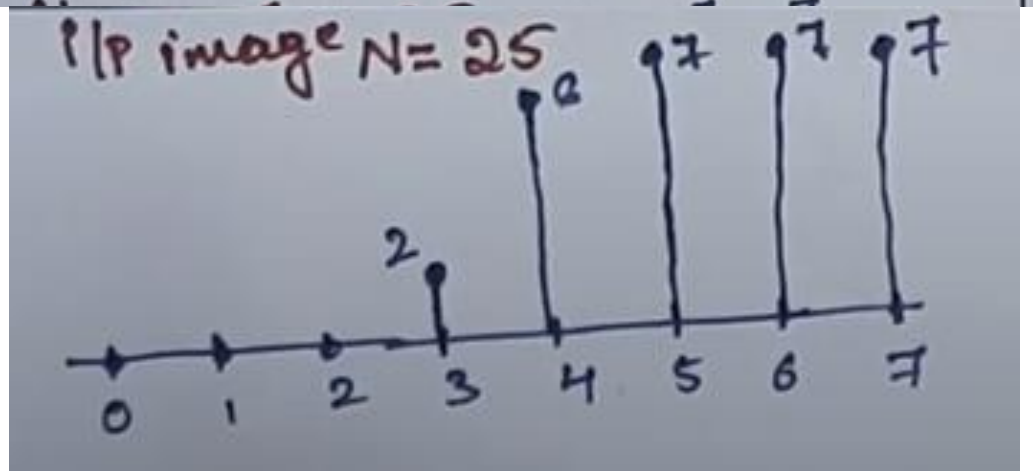


Gray levels	0	1	2	3	4	5	6	7
No of pixels n_k	0	0	0	6	14	5	0	0

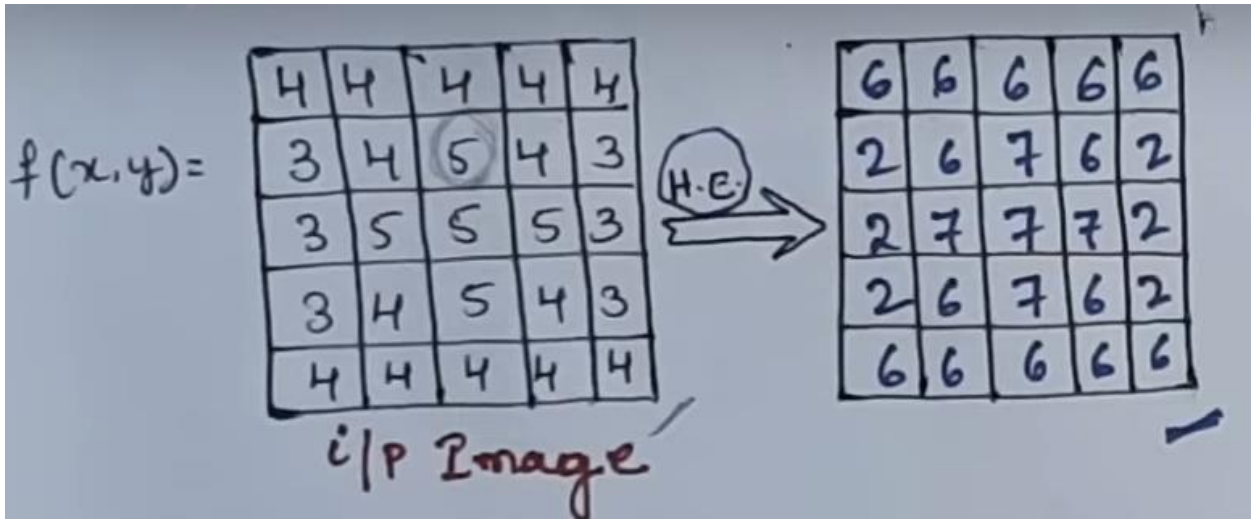
Histogram of input image.

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

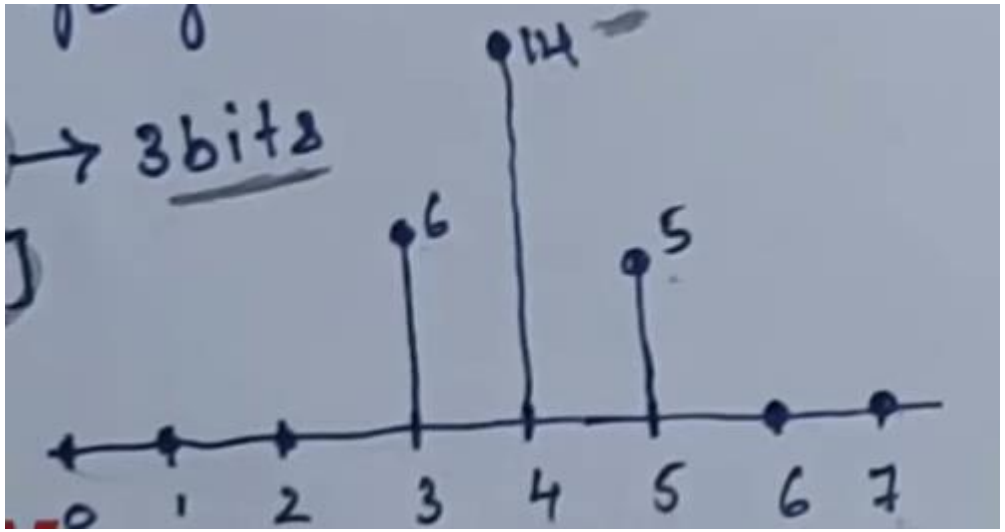
Gray level	No. of pixels n_k	PDF = n_k/sum	CDF = S_k	$S_k \times 7$	Histogram equal. level
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	6	$6/25 = 0.24$	0.24	1.68	2
4	14	$\frac{14}{25} = 0.56$	0.8	5.6	6
5	5	$5/25 = 0.2$	1.0	7	7
6	0	0	1.0	7	7
7	0	0	1.0	7	7.

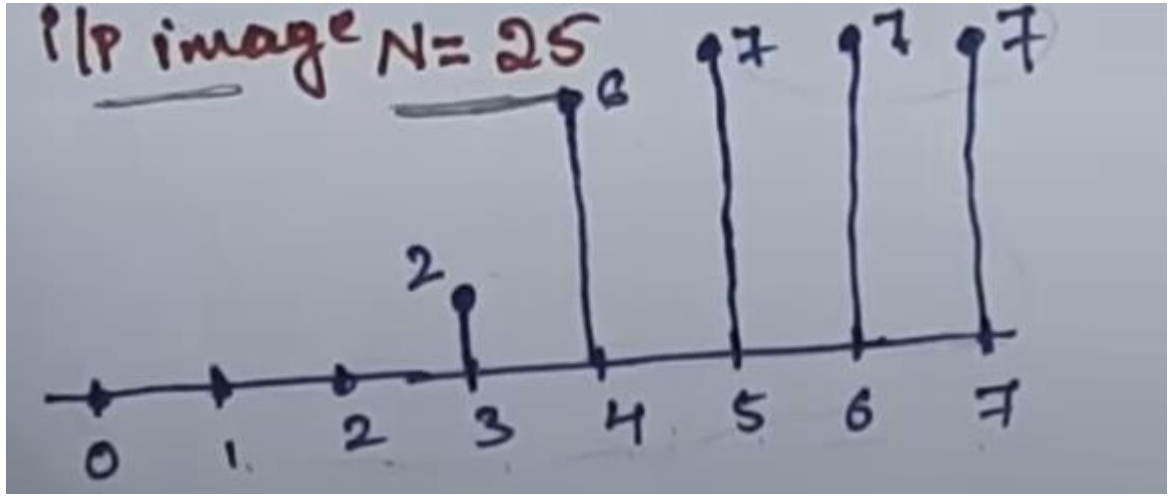


PDF = Probability Distribution Function
CDF = Cumulative Distribution Function



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





Function File: imadjust *(I)*

Function File: imadjust *(I, [low_in; high_in])*

Function File: imadjust *(I, [low_in; high_in],[low_out; high_out])*

Function File: imadjust *(..., gamma)*

Function File: imadjust *(cmap, ...)*

Function File: imadjust *(RGB, ...)*

The values are rescaled according to the input and output limits, *low_in* and *high_in*,
and *low_out* and *high_out* respectively.

Function File: *J* = histeq *(I, n)*

Equalize histogram of grayscale image.

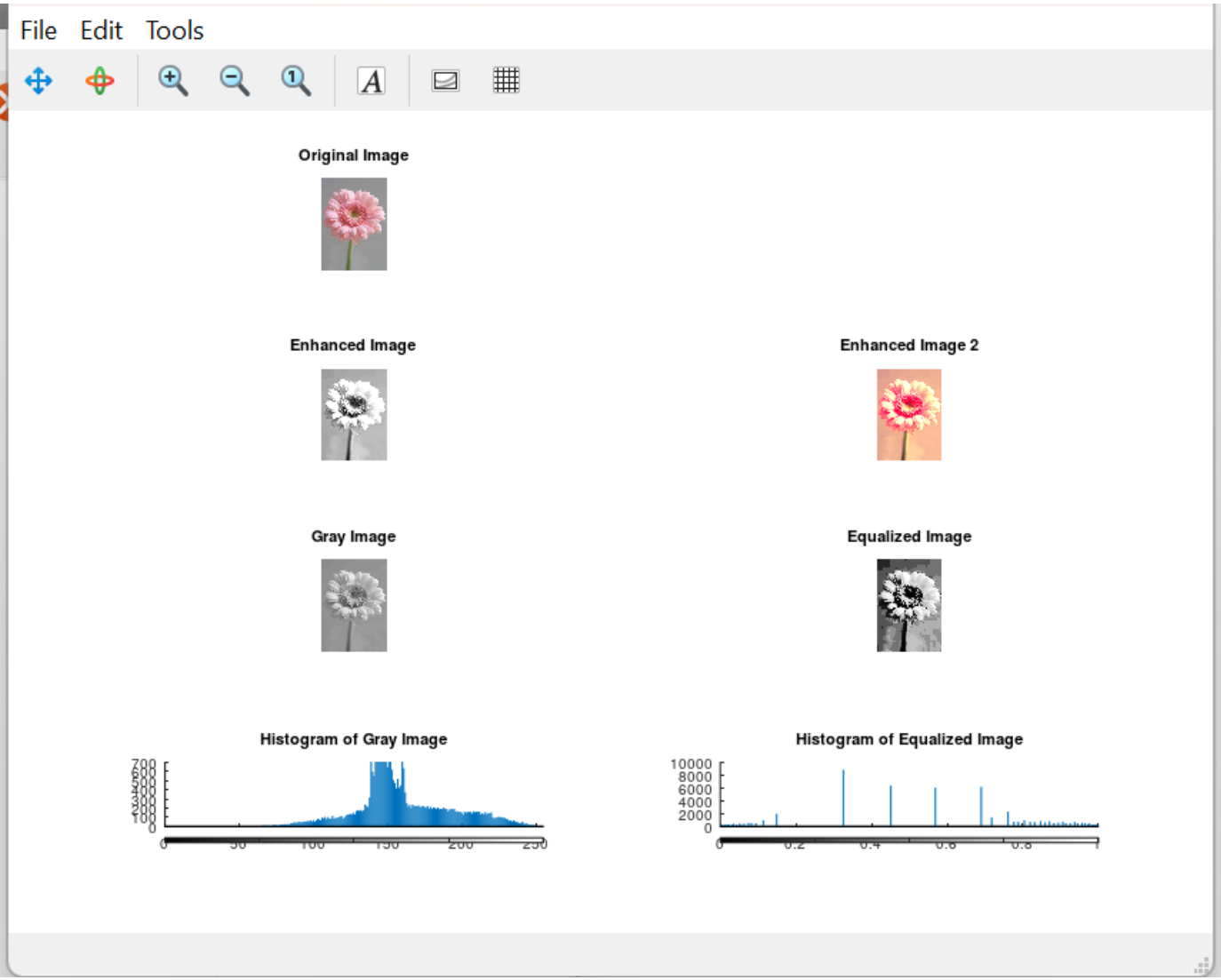
The histogram contains *n* bins, which defaults to 64.

I: Image in double format, with values from 0.0 to 1.0.

J: Returned image, in double format as well.

Lab 06: Histogram Equalization

```
Edit View Debug Run Help
t_plane.m | negative_linear_transformation.m | histogram.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 for Image Enhancement
7  I=imread('flower.png');
8  subplot(4,2,1); imshow(I); title('Original Image');
9  g=rgb2gray(I);
10 subplot(4,2,5); imshow(g); title('Gray Image');
11 J=imadjust(g,[0.3 0.7],[]);
12 subplot(4,2,3); imshow(J); title('Enhanced Image');
13 D= imadjust(I,[0.2 0.3 0; 0.6 0.7 1],[]);
14 subplot(4,2,4);imshow(D);title('Enhanced Image 2');
15 % Histogram and Histogram Equalization
16 subplot(4,2,7); imhist(g); title('Histogram of Gray Image');
17 m=histeq(g);
18 subplot(4,2,6); imshow(m); title('Equalized Image');
19 subplot(4,2,8); imhist(m); title('Histogram of Equalized Image');
20
```



Spatial Operations in DIP

Filtering in the spatial domain (Spatial Filtering)

refers to image operators that change the gray value at any pixel (x,y) depending on the pixel values in a square neighborhood centered at (x,y) using a fixed integer matrix of the same size. The integer matrix is called a *filter*, *mask*, *kernel* or a *window*.

The mechanism of spatial filtering, shown below, consists simply of moving the filter mask from pixel to pixel in an image. At each pixel (x,y), the response of the filter at that pixel is calculated using a predefined relationship (linear or nonlinear).

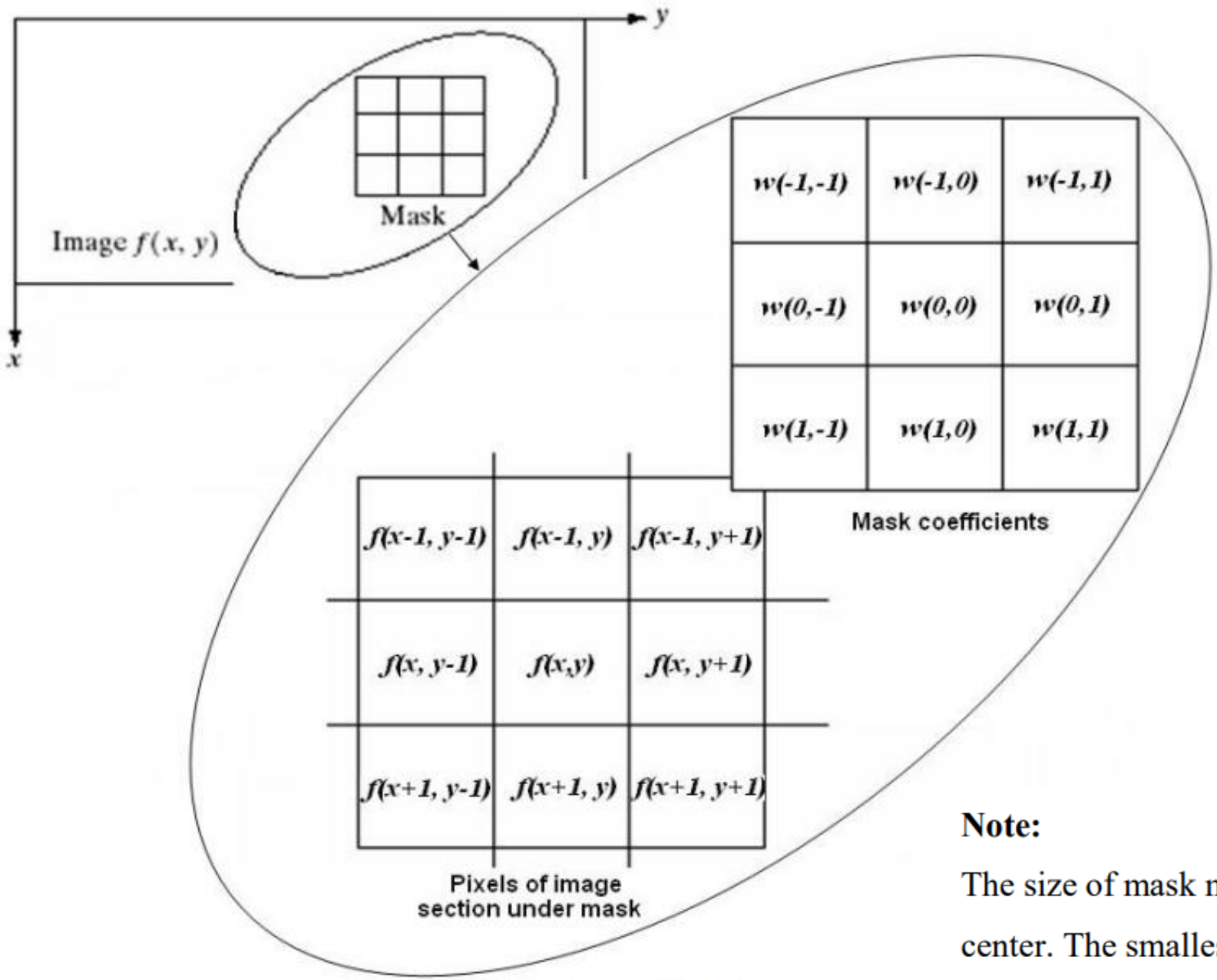


Figure 6.1 Spatial filtering

Note:

The size of mask must be odd (i.e. 3×3, 5×5, etc.) to ensure it has a center. The smallest meaningful size is 3×3.

Linear Spatial Filtering (Convolution)

The process consists of moving the filter mask from pixel to pixel in an image. At each pixel (x,y), the response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask.

Nonlinear Spatial Filtering

The operation also consists of moving the filter mask from pixel to pixel in an image. The filtering operation is based conditionally on the values of the pixels in the neighborhood, and they do not explicitly use coefficients in the sum-of-products manner.

For example, noise reduction can be achieved effectively with a nonlinear filter whose basic function is to compute the median gray-level value in the neighborhood in which the filter is located. Computation of the median is a nonlinear operation.

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:

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

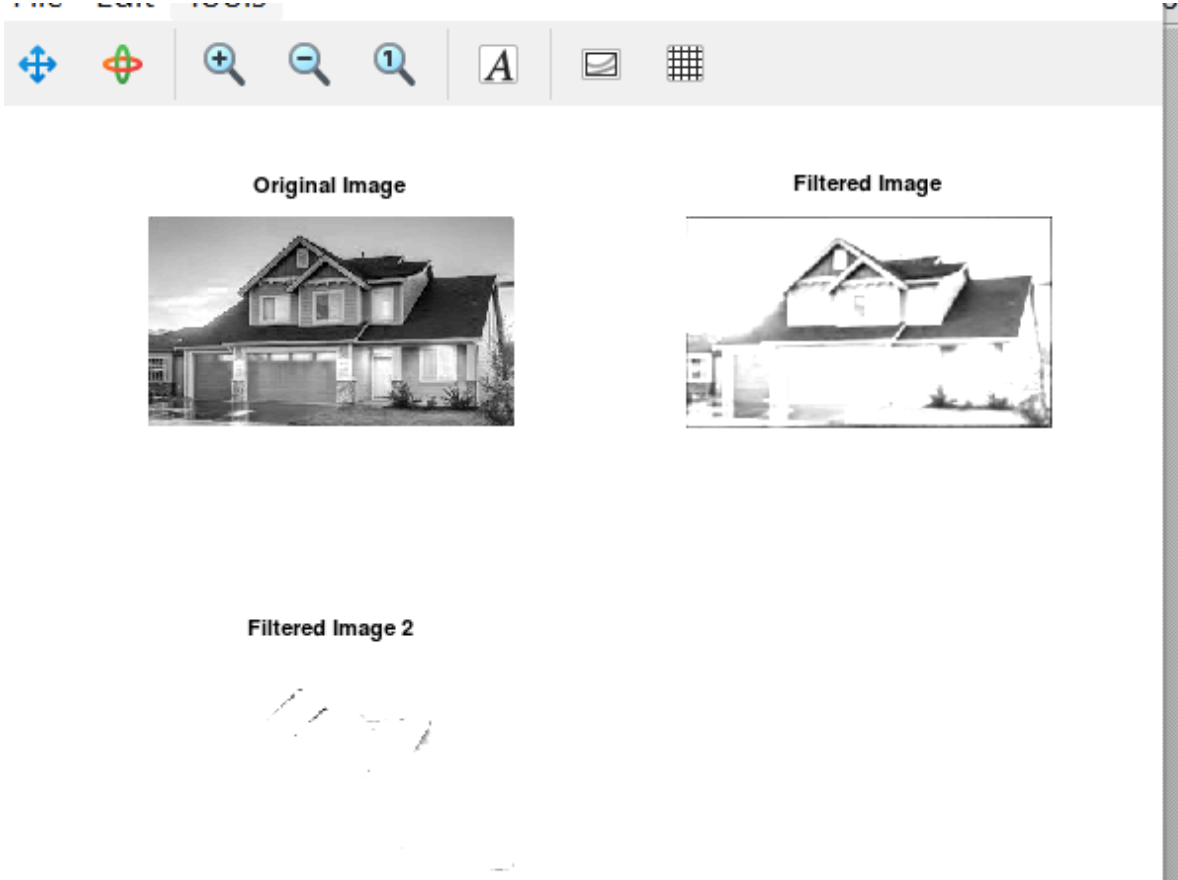
$$0 \times 40 + \frac{1}{6} \times 50 + 0 \times 70 + \frac{1}{6} \times 50 + \frac{1}{3} \times 80 + \frac{1}{6} \times 60 + 0 \times 255 + \frac{1}{6} \times 70 + 0 \times 0 = 65$$

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

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

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');
```

Spatial Filters

Spatial filters can be classified by effect into:

1. **Smoothing Spatial Filters:** also called lowpass filters. They include:
    1.1 Averaging linear filters
    1.2 Order-statistics nonlinear filters.
2. **Sharpening Spatial Filters:** also called highpass filters. For example, the Laplacian linear filter.

Smoothing Spatial Filters

are used for blurring and for noise reduction. Blurring is used in preprocessing steps to:

- remove small details from an image prior to (large) object extraction
- bridge small gaps in lines or curves.

Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering.

Averaging linear filters

The response of averaging filter is simply the average of the pixels contained in the neighborhood of the filter mask.

The output of averaging filters is a smoothed image with reduced "sharp" transitions in gray levels.

Noise and edges consist of sharp transitions in gray levels. Thus smoothing filters are used for noise reduction; however, they have the undesirable side effect that they blur edges.

The figure below shows two 3×3 averaging filters.

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

1	1	1
1	1	1
1	1	1

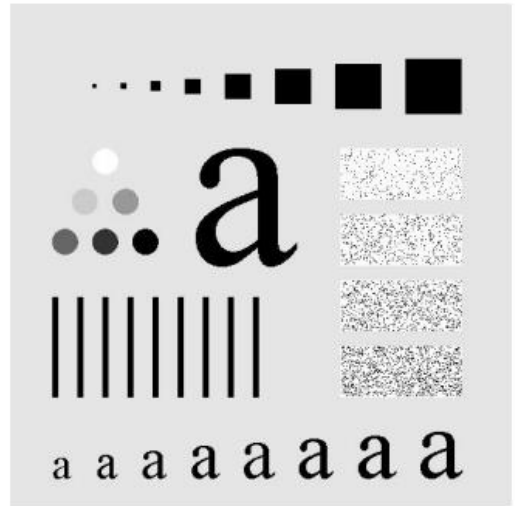
Standard average filter

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

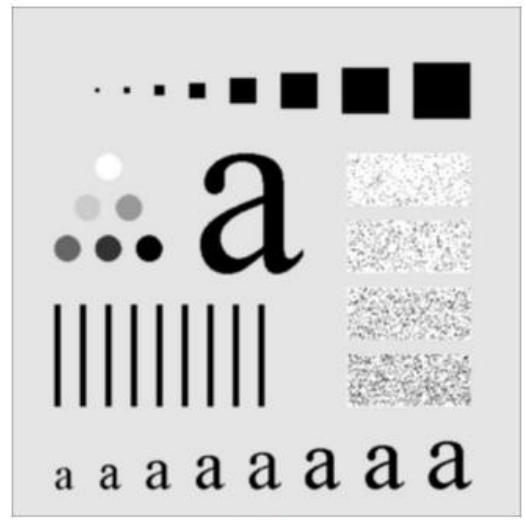
1	2	1
2	4	2
1	2	1

Weighted average filter

Figure below shows an example of applying the standard averaging filter.



(a)



(b)

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.

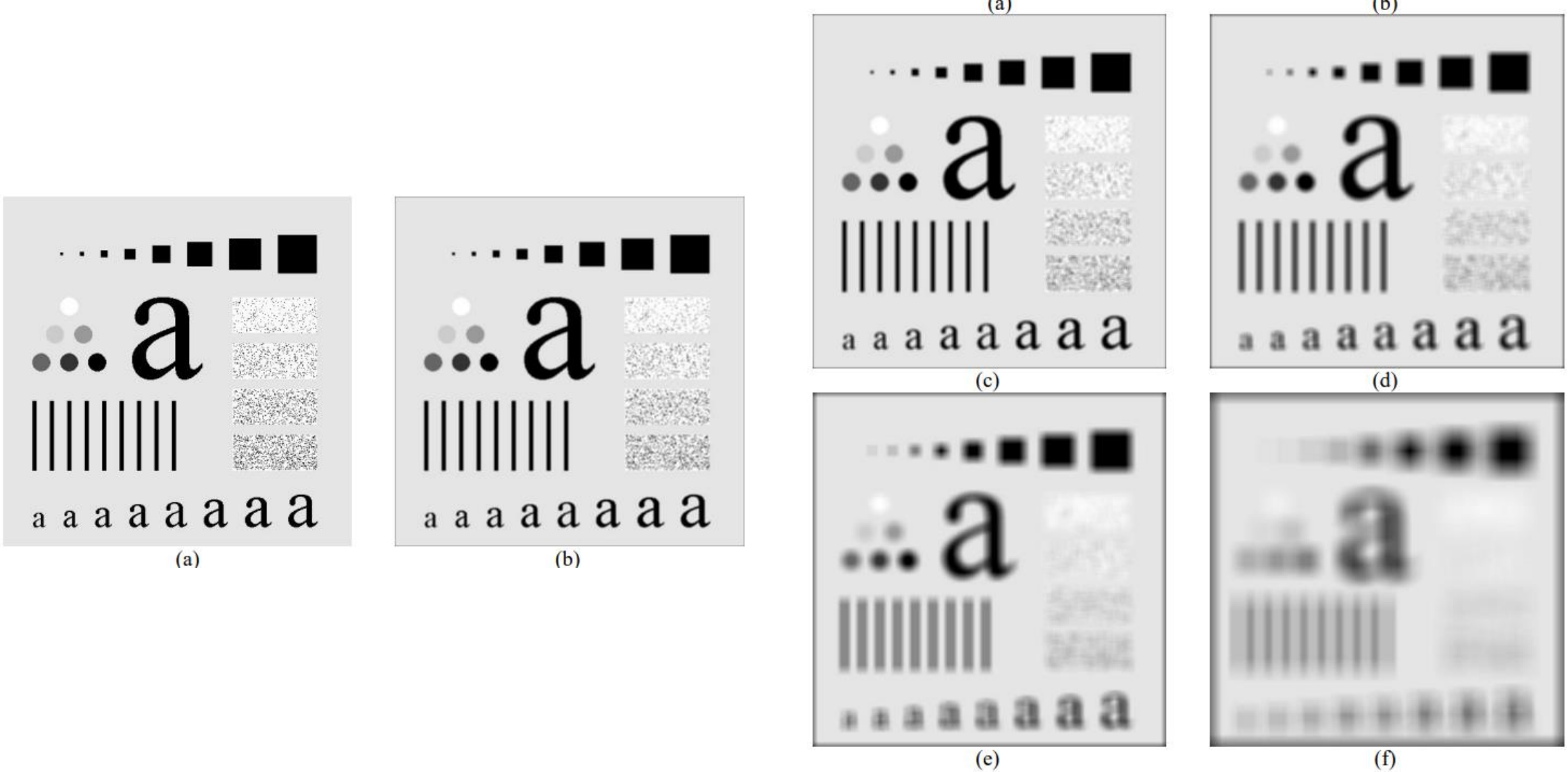


Figure 6.2 Effect of averaging filter. (a) Original image. (b)-(f) Results of smoothing with square averaging filter masks of sizes n = 3,5,9,15, and 35, respectively.

As shown in the figure, the effects of averaging linear filter are:

1. Blurring which is increased whenever the mask size increases.
2. Blending (removing) small objects with the background. The size of the mask establishes the relative size of the blended objects.
3. Black border because of padding the borders of the original image.
4. Reduced image quality.

Order-statistics filters

are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the neighborhood, and then replacing the value of the center pixel with the value determined by the ranking result.

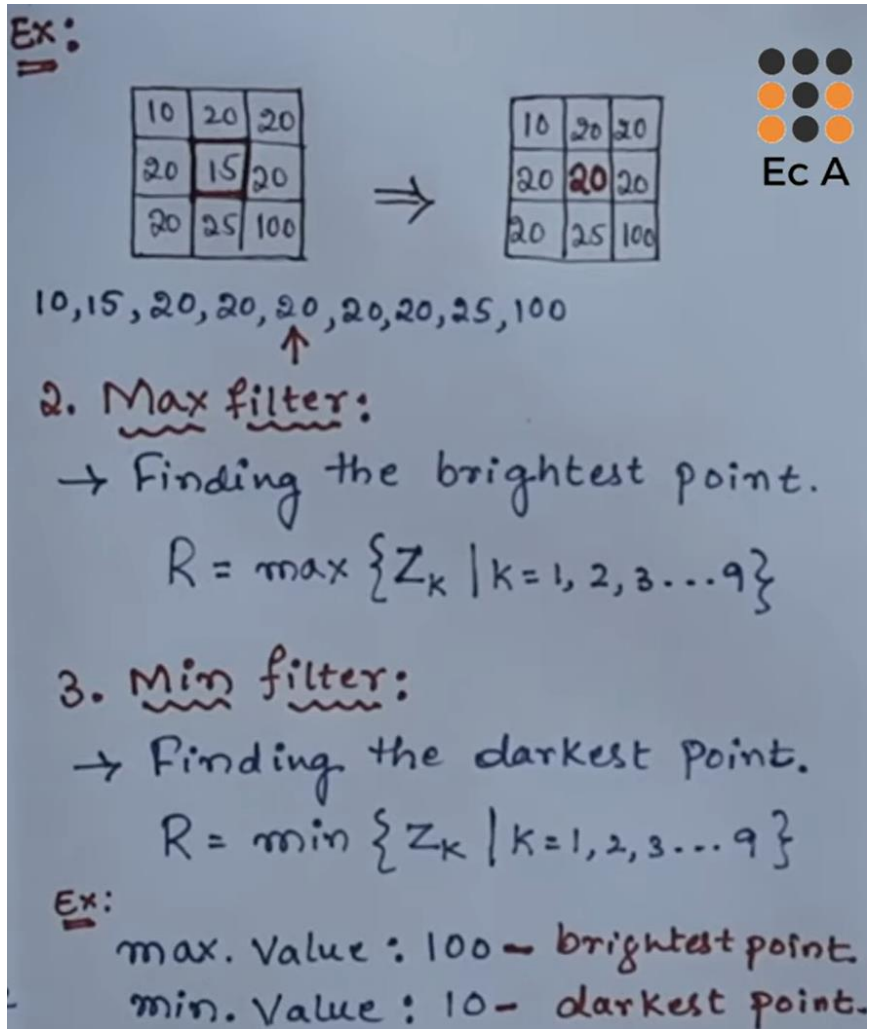
Examples include Max, Min, and Median filters.

Median filter

It replaces the value at the center by the median pixel value in the neighborhood, (i.e. the middle element after they are sorted). Median filters are particularly useful in removing impulse noise (also known as salt-and-pepper noise). Salt = 255, pepper = 0 gray levels.

In a 3×3 neighborhood the median is the 5th largest value, in a 5×5 neighborhood the 13th largest value, and so on.

For example, suppose that a 3×3 neighborhood has gray levels (10, 20, 0, 20, 255, 20, 20, 25, 15). These values are sorted as (0,10,15,20,20,20,20,25,255), which results in a median of 20 that replaces the original pixel value 255 (salt noise).



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

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

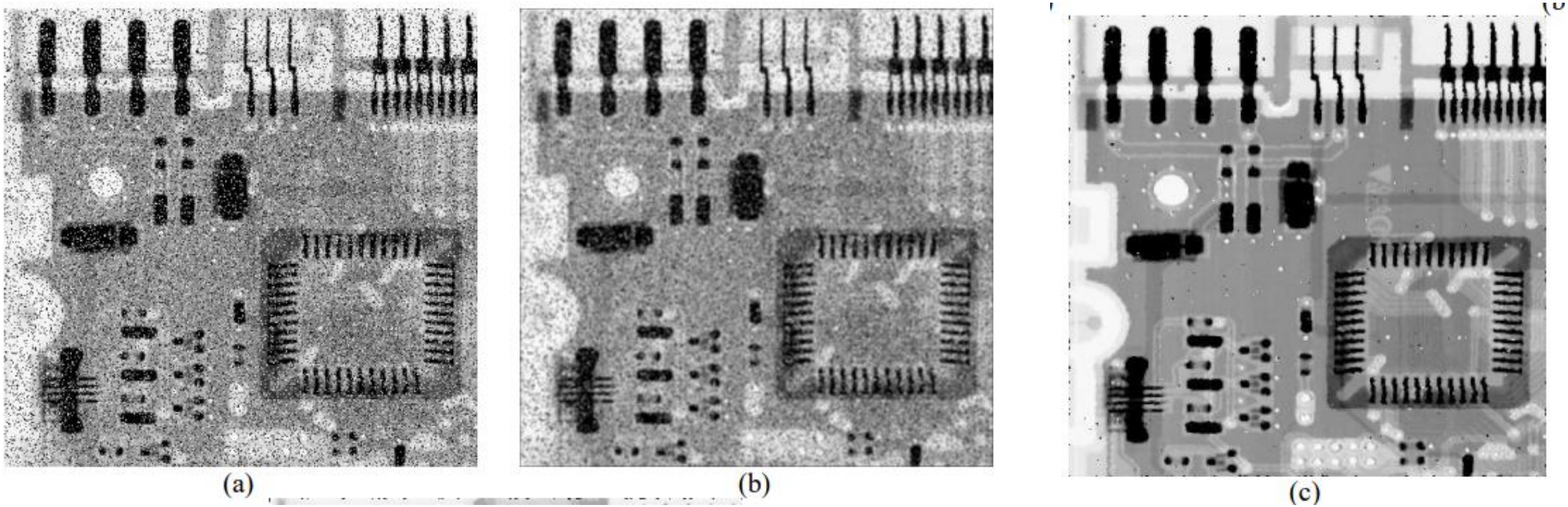
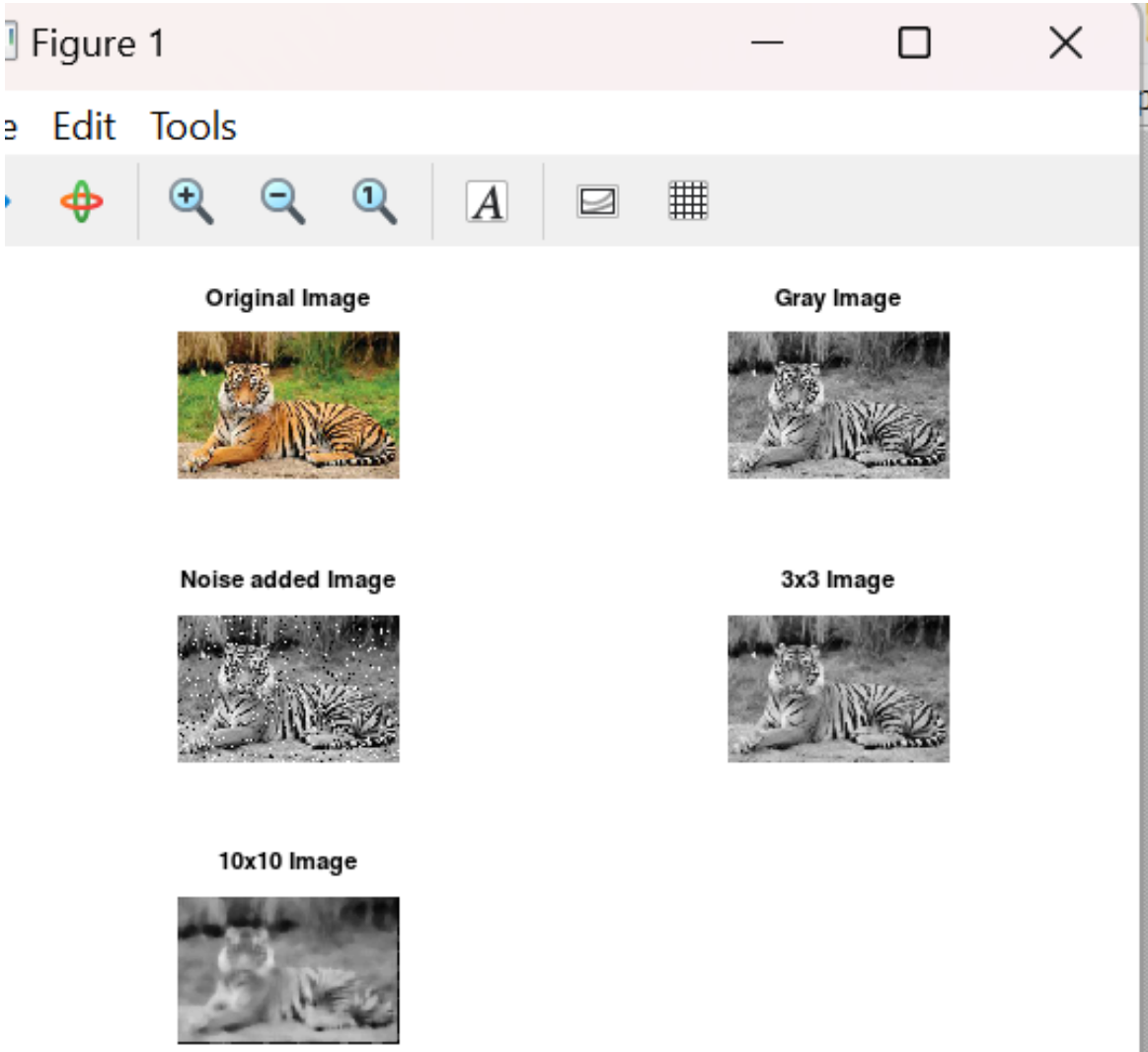


Figure 6.3 Effect of median filter. (a) Image corrupted by salt & pepper noise. (b) Result of applying 3×3 standard averaging filter on (a). (c) Result of applying 3×3 median filter on (a).

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');
```

As shown in the figure, the effects of median filter are:

1. Noise reduction
2. Less blurring than averaging linear filter

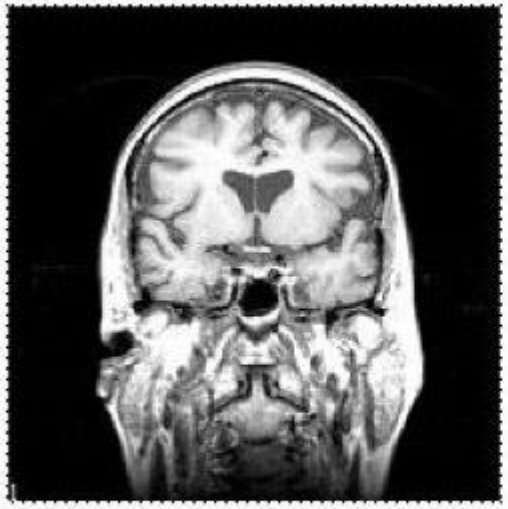
Sharpening Spatial Filters

Sharpening aims to highlight fine details (e.g. edges) in an image, or enhance detail that has been blurred through errors or imperfect capturing devices.

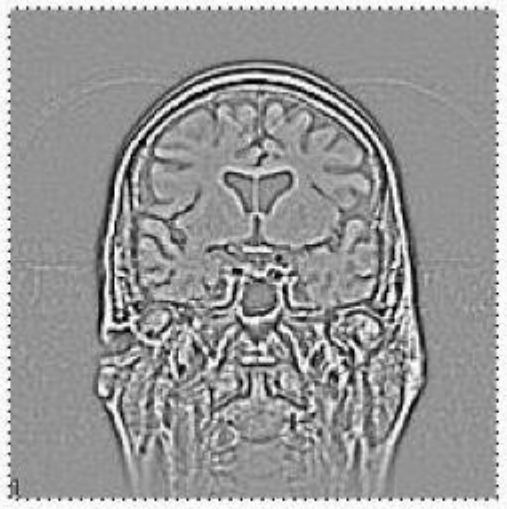
Image blurring can be achieved using averaging filters, and hence sharpening can be achieved by operators that invert averaging operators. In mathematics, averaging is equivalent to the concept of integration, and differentiation inverts integration. Thus, sharpening spatial filters can be represented by partial derivatives.

First Derivative Vs Second Derivative Filters

- **Sensitivity to Noise**: Second derivative filters are more sensitive to noise compared to first derivative filters because they respond to changes in the rate of change, which can be affected by small fluctuations in intensity.
- **Edge Detection vs. Detail Enhancement**: First derivative filters are primarily used for edge detection due to their ability to highlight rapid changes in intensity. Second derivative filters, meanwhile, can highlight finer details and are used for both edge detection and image enhancement.
- **Response Pattern**: First derivative filters produce a response that is directly proportional to the edge slope, resulting in thicker edges for gradual intensity changes. Second derivative filters produce a zero-crossing at the edge location, which can result in finer edge representations but may also detect additional, non-edge features.



(A) Original MR image



(B) Laplacian results

(C) Extraction of the zero crossing of the Laplacian (object edges)

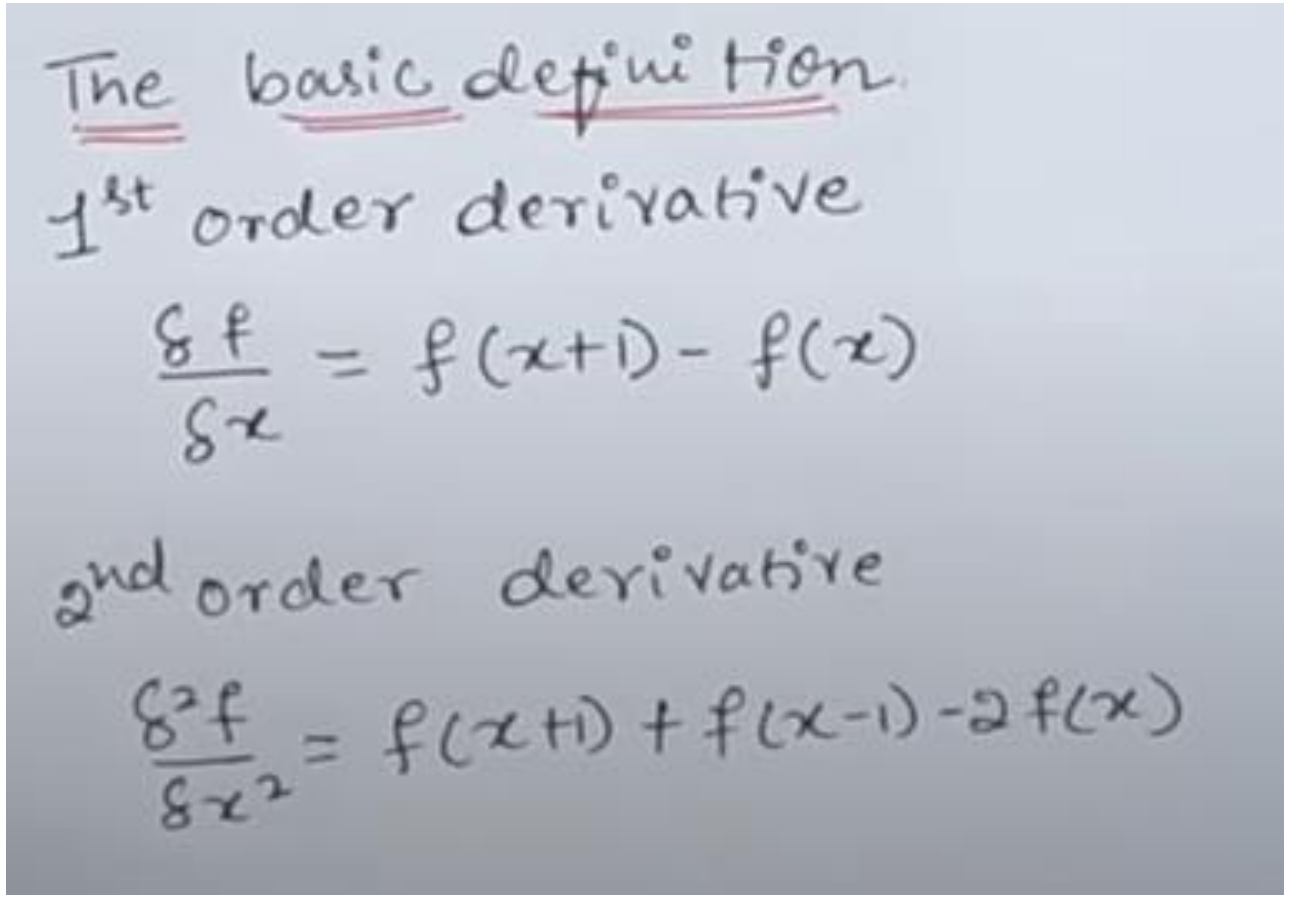
- For First Derivative:
  - Must be zero on flat segments
  - Must be non-zero at onset of a gray level or on ramp
  - Must be non-zero along ramp

For Second Derivate, following properties apply:

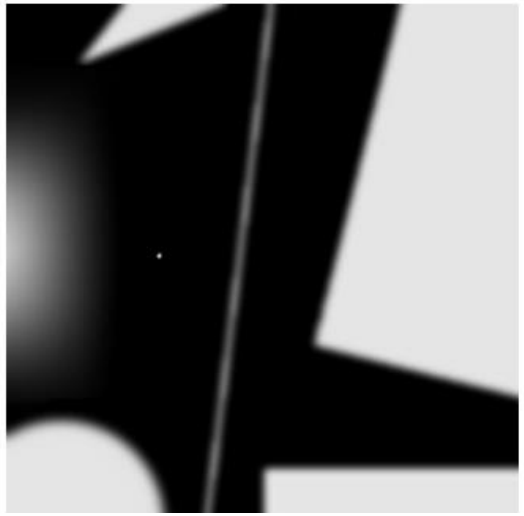
Must be zero in flat area

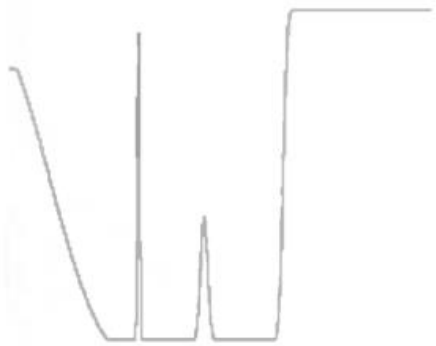
Must be non-zero at the onset and end of gray level step or ramp

Must be zero along ramps of constant slope



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.

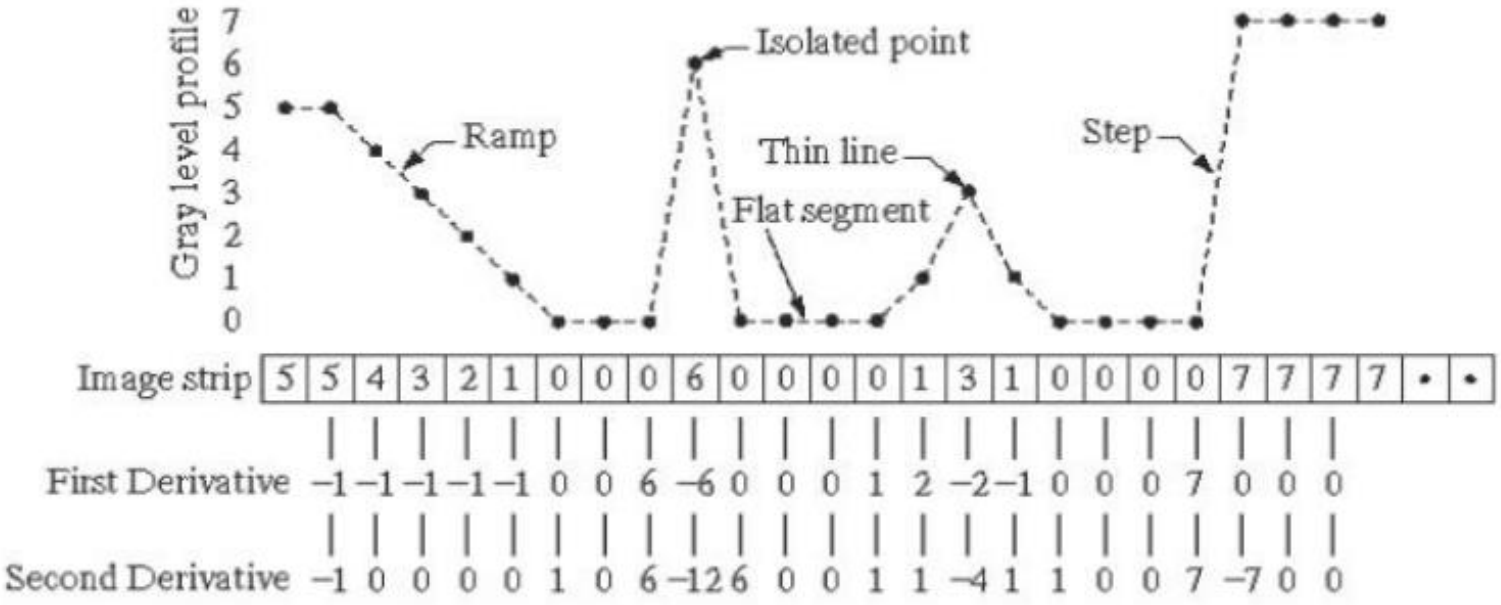


Figure 6.4 Example of partial derivatives

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)

Laplacian Filter in Image Processing

The Laplacian filter is a second derivative filter used in image processing for edge detection. It measures the rate at which the first derivatives change, allowing us to find regions of rapid change (edges) in images. The Laplacian operator uses the second spatial derivative of an image to enhance regions of rapid intensity change.

Derivation of the Laplacian Filter Mask

The Laplacian ∇^2 of a function $f(x, y)$ in two dimensions is given by the sum of the second partial derivatives with respect to x and y:

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

In discrete 2D space (like an image grid), we approximate these second derivatives using finite differences. For a pixel $f(x, y)$, the second derivative with respect to x and y can be approximated as:

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

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

Adding these two approximations gives us the Laplacian:

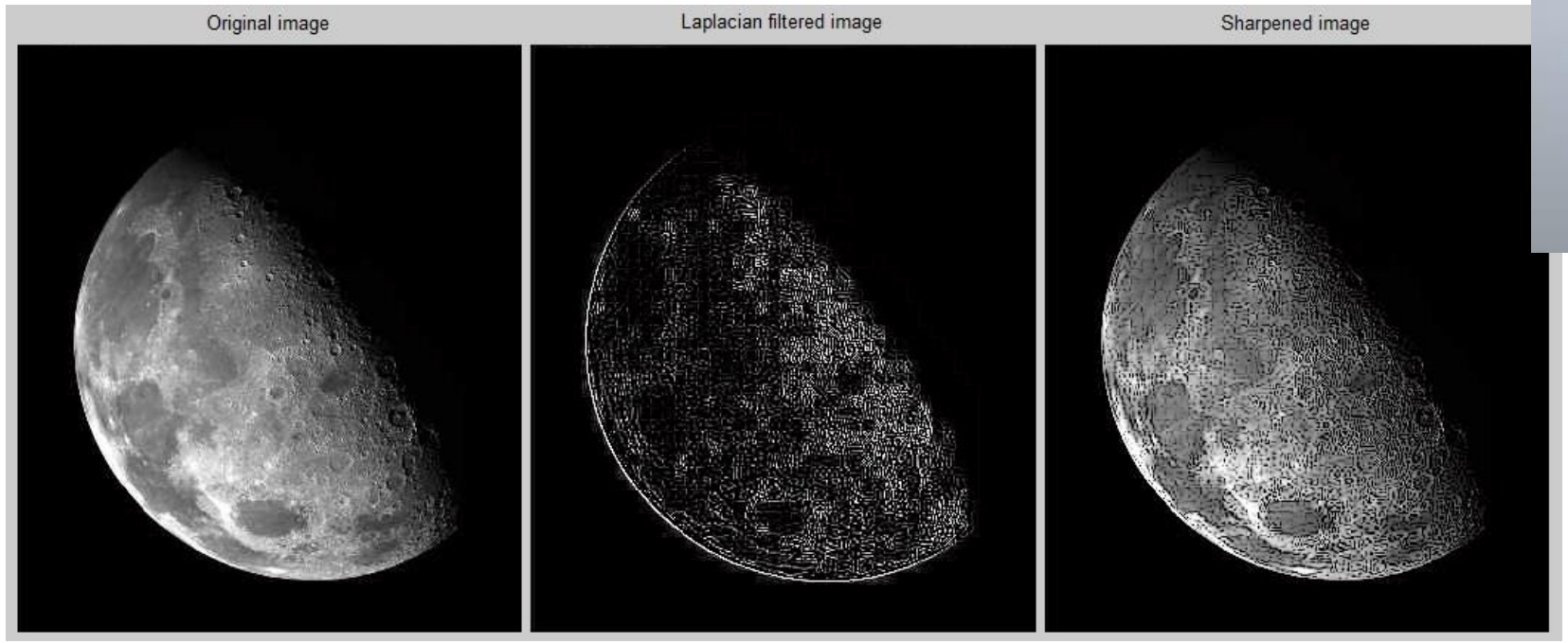
$$\nabla^2 f \approx f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4f(x, y)$$

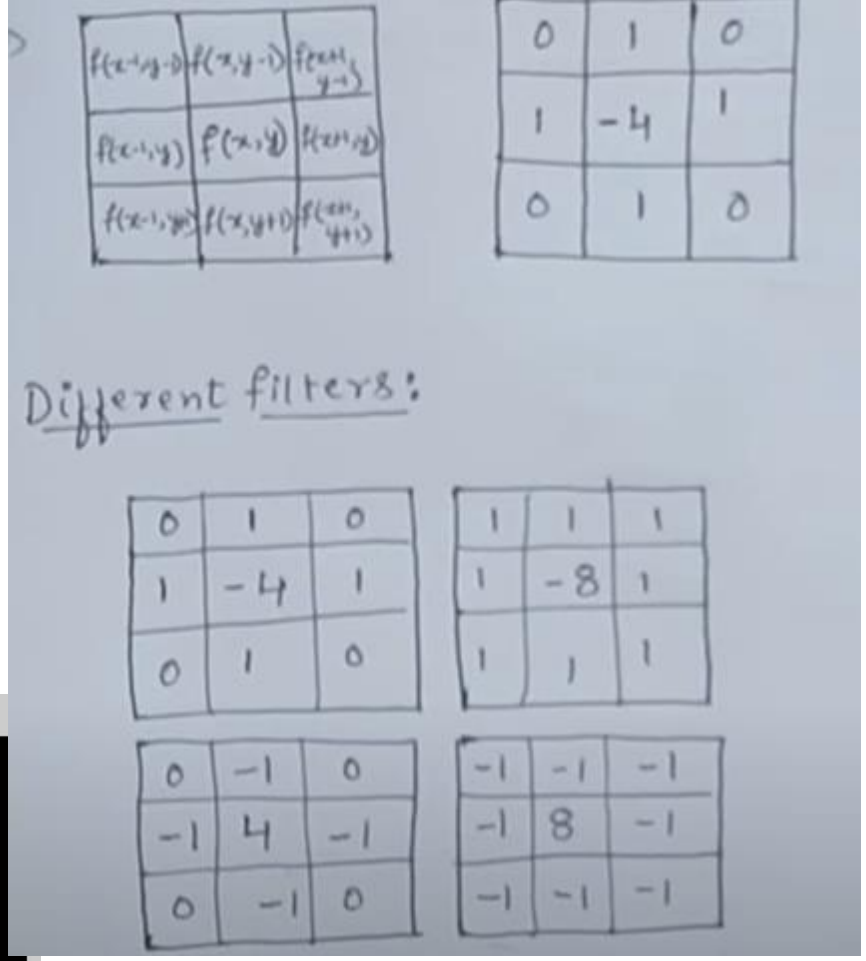
This can be represented as a filter mask (or kernel) for a 3×3 neighborhood:

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

Or, including diagonal neighbors (which is another common representation):

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





In table1: equal importance is given to surroundings while less importance given to center of image. Exact opposite in table 3. In the last table, only center will be highlighted while others remain the same.

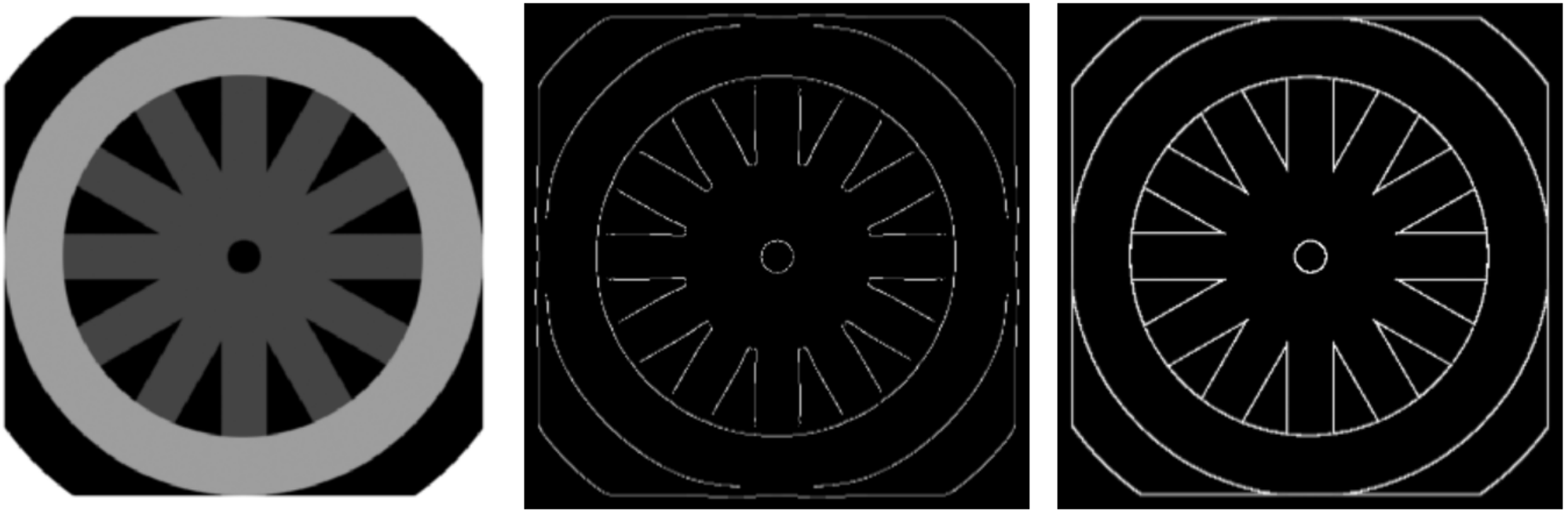
Algorithm for Implementing the Laplacian Filter

1. **Prepare the Image**: Optionally, apply a Gaussian blur to the image. This step reduces noise which can significantly affect the results of the Laplacian filter due to its sensitivity to noise.
2. **Select the Laplacian Filter Mask**: Choose the appropriate Laplacian filter mask. The choice between the two common masks (4-neighbor or 8-neighbor) depends on whether diagonal changes in intensity are important for your application.
3. **Apply the Filter to Each Pixel**:
   - For each pixel in the image (excluding the border pixels, unless padding is applied), take the sum of the product of the Laplacian filter mask and the corresponding neighborhood pixels.
   - Replace the current pixel value with this sum.
4. **Handle Negative Values**: The Laplacian filter may produce negative values, depending on the intensity changes. These need to be handled appropriately, either by taking the absolute value, scaling, or thresholding, to produce a valid image.
5. **Post-Processing**: Additional steps may be necessary depending on the application, such as thresholding the result to highlight edges.

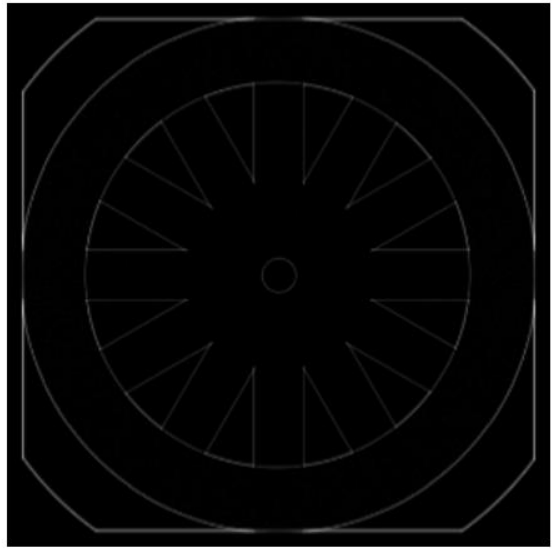
Edge Detection Filter

- Edges are usually one of the most important features in a structure, and can often be used for measurements after appropriate edge detection has been applied.
- The edge detection filters available in Dragonfly can be used to emphasize the edges and transitions in an image. Refer to the comparison chart below for a quick overview of the available filters. The edge detection filter comparisons are as:

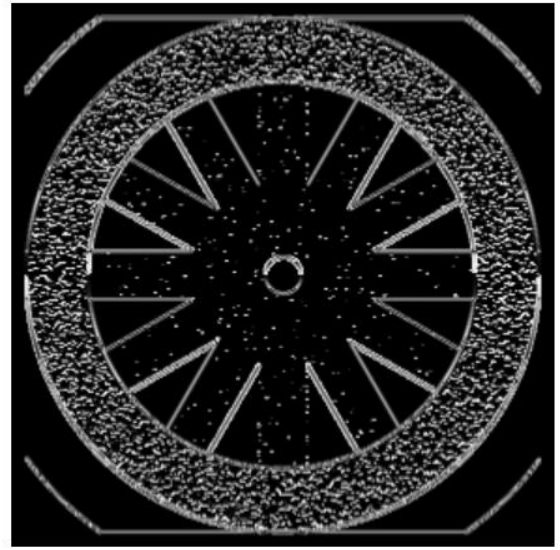
Edge detection filter comparison



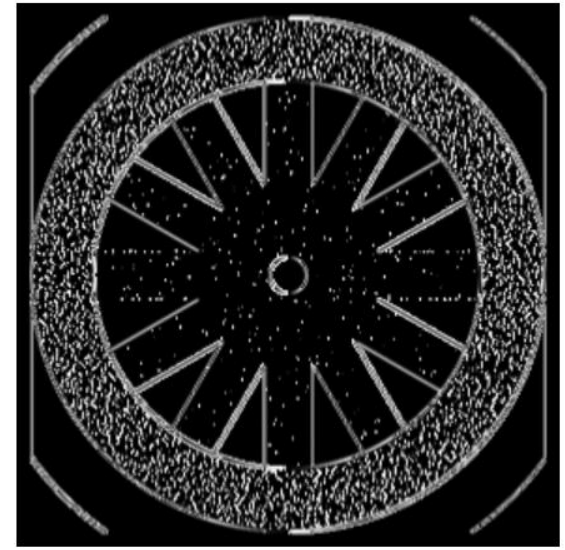
Original image | Canny | Difference of Gaussian (DoG)



Laplacian

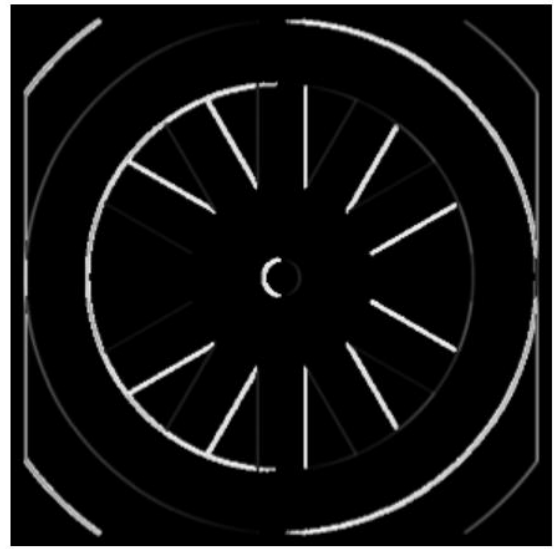


Prewitt (Horizontal)

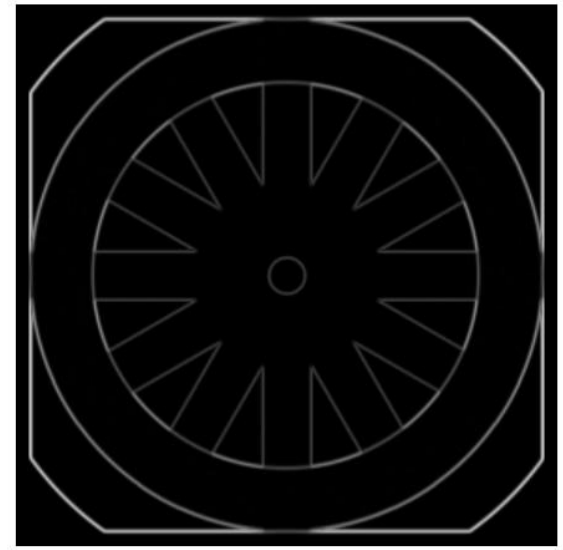


Prewitt (Vertical)

Scharr (horizontal)



Scharr (vertical)



Sobel

- Edge detection is a fundamental tool in image processing and computer vision, aimed at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. **The primary purpose of edge detection is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing.**
- Several algorithms and techniques can be used for edge detection, and they vary in terms of complexity and the type of edges they can detect. The most common methods include:
    1. **Sobel Operator**: Emphasizes edges in the vertical and horizontal direction by convolving the image with a pair of 3x3 kernels. It is relatively simple but effective for many applications.
    2. **Canny Edge Detector**: Aims to satisfy three main criteria: low error rate (detect as many real edges as possible while minimizing the detection of false edges), good edge localization (detected edges are as close as possible to the true edges), and minimal response (only a single edge detector response to a single edge). It uses a multi-stage algorithm to detect a wide range of edges in images.
    3. **Prewitt Operator:** Similar to the Sobel operator but uses a different kernel for detecting horizontal and vertical edges. It is also simple and effective for basic edge detection tasks.
    4. **Laplacian of Gaussian (LoG):** Involves smoothing the image with a Gaussian filter and then applying the Laplacian operator to detect edges. This method is particularly good at detecting blob-like structures in addition to edges.

Magnification in Image Processing

- Magnification in image processing is a technique used to increase the size of an image.
- It's a crucial part of many applications, from enhancing digital photographs to allowing detailed analysis in scientific research.
- There are several methods to perform magnification, but two common ones are pixel replication (also known as nearest neighbor interpolation) and interpolation (with methods like bilinear, bicubic, and Lanczos).
- Each method has its own set of advantages and drawbacks.

1) Pixel Replication (Nearest Neighbor Interpolation)

Pixel replication is the simplest form of image magnification. It involves duplicating the pixels in an image to enlarge it without introducing any new colors or gradients. Here's how it works:

1. **Determine the Scale Factor**: First, you decide how much larger you want the image to be. For example, a scale factor of 2 means each pixel will be duplicated to create an image twice the original size in each dimension.
2. **Duplicate Pixels**: For each pixel in the original image, create new pixels in the enlarged image that have the same color value. If your scale factor is 2, each pixel in the original image becomes a 2x2 block of pixels in the enlarged image with the same color.

- **Advantages**:
  - Simplicity: It's straightforward to implement.
  - Speed: It's faster than more complex interpolation methods because it doesn't involve any calculations to create new pixel values.
- **Drawbacks**:
  - Quality: The resulting image can appear blocky or pixelated, especially if the scale factor is large. It doesn't handle diagonal lines or curves well.

Magnification using interpolation

Interpolation methods are more sophisticated and aim to create new pixel values that make the enlarged image appear smoother and more continuous than what pixel replication can achieve. The main types include:

1. **Bilinear Interpolation**: This method considers the closest 2x2 neighborhood of known pixel values surrounding the unknown pixel. It then performs a linear interpolation first in one direction and then in the other to generate the new pixel value.
2. **Bicubic Interpolation**: Bicubic interpolation goes a step further by considering a 4x4 neighborhood of pixels for interpolating. It uses cubic polynomials, leading to smoother images compared to bilinear interpolation.
3. **Lanczos Interpolation**: This method uses a sinc function to interpolate the new pixel values. It's often considered to produce the highest quality results, especially for images with high contrast, but it's also the most computationally intensive.

Advantages:

- Improved Quality: Interpolation generally produces smoother and more visually appealing images than pixel replication, especially for higher magnification levels.
- Flexibility: Different interpolation methods offer a balance between computational complexity and image quality, allowing for optimization based on the application's needs.

Drawbacks:

- Complexity: Interpolation methods are more complex to implement than pixel replication.
- Computational Cost: These methods are slower and more computationally intensive, especially for large images or high magnification levels.

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.