

Chapter Two

Image Analysis

2.1 Image Analysis

Image analysis involves manipulating the image data to determine exactly the information necessary to help solve a computer imaging problem. This analysis is typically part of a larger process, is iterative in nature and allows us to answer application specific questions: Do we need color information? Do we need to transform the image data into the frequency domain? Do we need to segment the image to find object information? What are the important features of the image?

Image analysis is primarily **data reduction** process. As we have seen, images contain enormous amount of data, typically on the order hundreds of kilobytes or even megabytes. Often much of this information is not necessary to solve a specific computer imaging problem, so primary part of the image analysis task is to determine exactly what information is necessary. Image analysis used both computer vision and image processing.

For computer vision, the end product is typically the extraction of high-level information for computer analysis or manipulation. This high level information may include shape parameter to control a robotics manipulator or color and texture features to help in diagnosis of a skin tumor.

In image processing application, image analysis methods may be used to help determine the type of processing required and the specific parameters needed for that processing. For example, determine the degradation function for an image restoration procedure, developing an enhancement algorithm and determining exactly what information is visually important for image compression methods.

2.2 System Model

The image analysis process can be broken down into three primary stages:

1. Preprocessing.
2. Data Reduction.
3. Features Analysis.

1. Preprocessing:

The preprocessing algorithm, techniques and operators are used to perform initial processing that makes the primary data reduction and analysis task easier. They include operations related to:

- Gray –level or spatial quantization (reducing the number of bits per pixel or the image size).
- Is used to remove noise and eliminate irrelevant, visually unnecessary information, (Noise is unwanted information that can result from the image acquisition process) □ Finding regions of interest for further processing.
- Performing basic algebraic operation on image.
- Enhancing specific image features.
- Reducing data in resolution and brightness.

Preprocessing is a stage where the requirements are typically obvious and simple, such as removal of artifacts from images or eliminating of image information that is not required for the application. For example, in one application we needed to eliminate borders from the images that have been digitized from film. Another example of preprocessing step involves a robotics gripper that needs to pick and place an object ; for this we reduce a gray-level image to binary (two-valued) image that contains all the information necessary to discern the object 's outlines.

2. Data Reduction:

Involves either reducing the data in the spatial domain or transforming it into another domain called the frequency domain, and then extraction features for the analysis process.

3. Features Analysis:

The features extracted by the data reduction process are examine and evaluated for their use in the application.

After preprocessing we can perform segmentation on the image in the spatial domain or convert it into the frequency domain via a mathematical transform. After these processes we may choose to filter the image. This filtering process further reduces the data and allows us to extract the feature that we may require for analysis.

Figure (2.1) illustrates the system model of image analysis

Image Analysis System Model

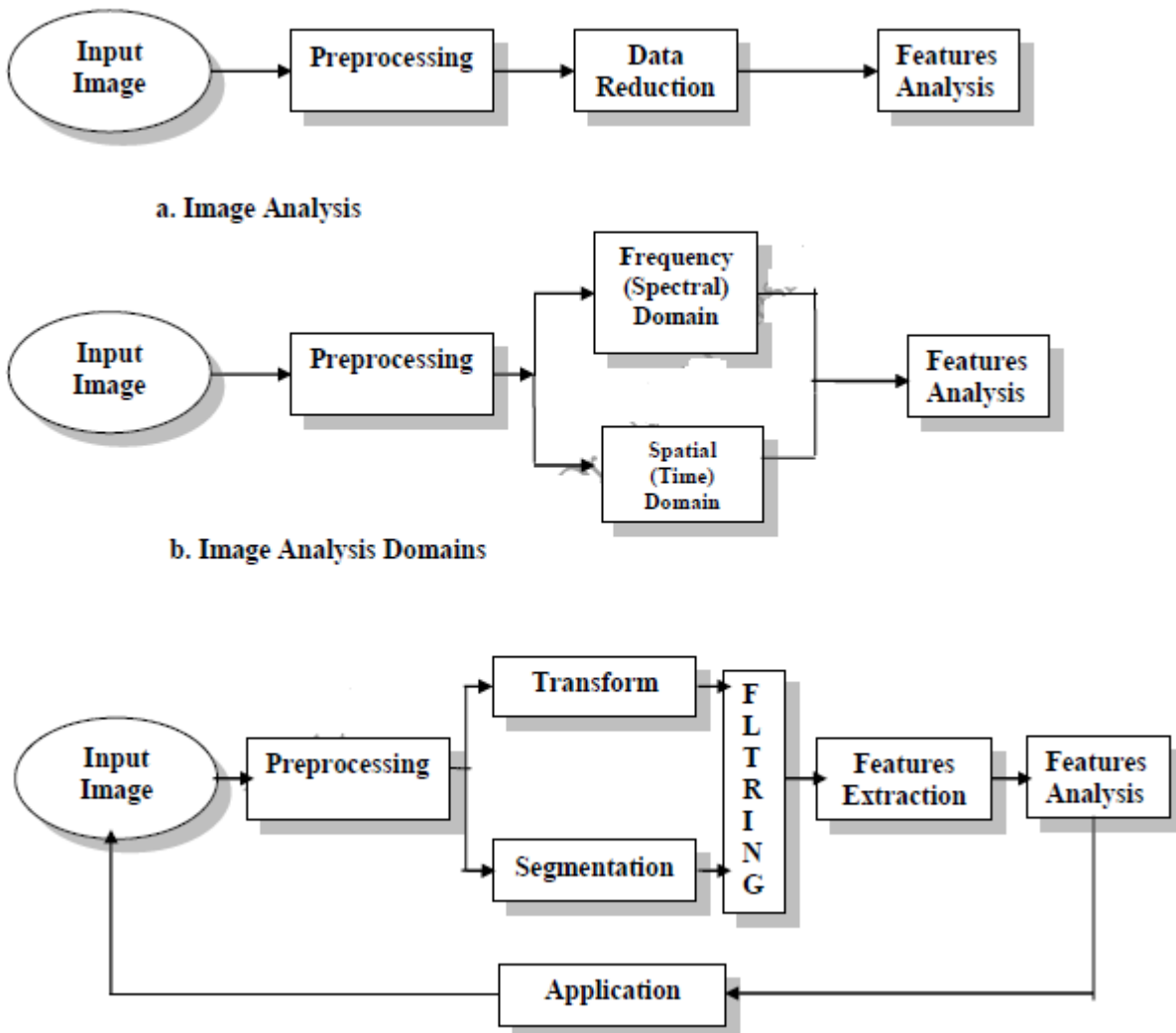


Figure (2.1): System Model of Image Analysis

2.3 Region –of-Interest (ROI)

For image analysis we want to investigate more interested area within the image, called region of interest (ROI). To do this we need operation that modifies the spatial coordinates of the image, and these are categorized as image geometry operations. The image geometry operations discussed here include:

Crop , Zoom, enlarge ,shrink, translate and rotate.

The image crop process is the process of selecting a small portion of the image, a sub image and cutting it away from the rest of the image. After we have cropped a sub image from the original image we can zoom in on it by enlarge it. The zoom process can be implemented by the following techniques:

- 1. Zero-Order Hold.**
 - 2. First _Order Hold.**
 - 3. Convolution.**
- 1. Zero-Order hold:** is achieved by repeating previous pixel values, thus creating a blocky effect as in the following figure:

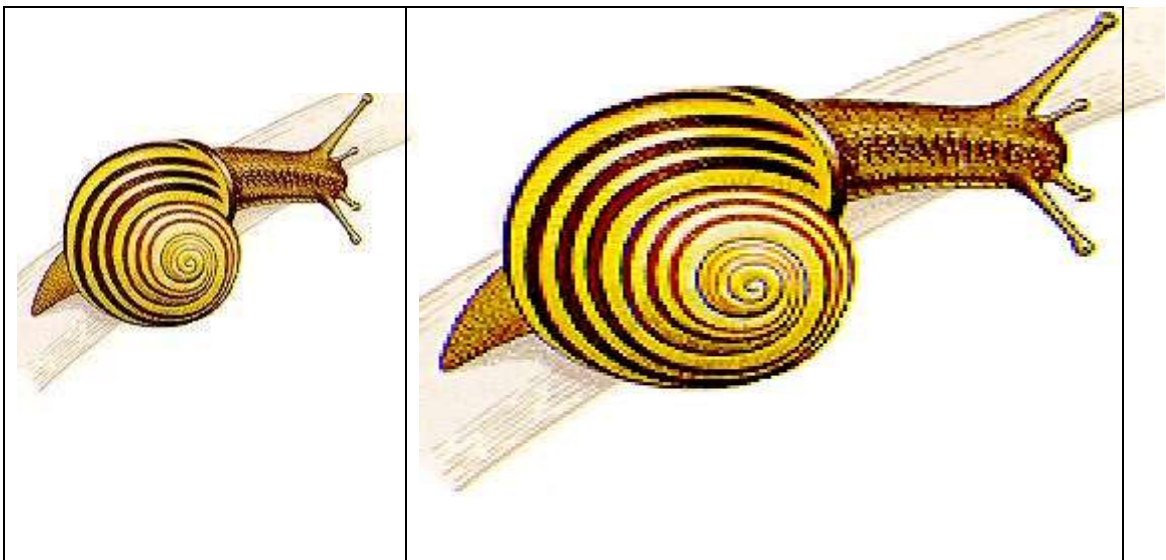


Figure (2.2): Zero Order Hold Method

- 2. First _Order Hold:** is performed by finding linear interpolation between adjacent pixels, i.e., finding the average value between two pixels and use that as the pixel value between those two, we can do this for the rows first as follows:

Original Image Array

$$\begin{bmatrix} 8 & 4 & 8 \\ 4 & 8 & 4 \\ 8 & 2 & 8 \end{bmatrix}$$

Image with Rows Expanded

$$\begin{bmatrix} 8 & 6 & 4 & 6 & 8 \\ 4 & 6 & 8 & 6 & 4 \\ 8 & 5 & 2 & 5 & 8 \end{bmatrix}$$

The first two pixels in the first row are averaged $(8+4)/2=6$, and this number is inserted between those two pixels. This is done for every pixel pair in each row.

Image with rows and columns expanded

$$\begin{bmatrix} 8 & & 6 & & 4 & & 6 & & 8 \\ & 6 & & 6 & & 6 & & 6 & \\ 4 & & 6 & & 8 & & 6 & & 4 \\ & 6 & & 5.5 = 6 & & 5 & & 5.5 = 6 & \\ 8 & & 5 & & 2 & & 5 & & 8 \end{bmatrix}$$

This method allows us to enlarge an $N \times N$ sized image to a size of $(2N-1) \times (2N-1)$ and be repeated as desired.

3- Convolution: this process requires a mathematical process to enlarge an image. This method required two steps:

1. Extend the image by adding rows and columns of zeros between the existing rows and columns.
2. Perform the convolution.

The image is extended as follows:

Original Image Array

$$\begin{pmatrix} 3 & 5 & 7 \\ 2 & 7 & 6 \\ 3 & 4 & 9 \end{pmatrix}$$

Image extended with zeros

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 7 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 4 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Next, we use convolution mask, which is slide a cross the extended image, and perform simple arithmetic operation at each pixel location

Convolution mask for first-order hold

$$\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

The convolution process requires us to overlay the mask on the image, multiply the coincident (متقابله) values and sum all these results. This is equivalent to finding the vector inner product of the mask with underlying sub image. The vector inner product is found by overlaying mask on sub image. Multiplying coincident terms, and summing the resulting products.

For example, if we put the mask over the upper-left corner of the image, we obtain (from right to left, and top to bottom):

$$1/4(0) + 1/2(0) + 1/4(0) + 1/2(0) + 1(3) + 1/2(0) + 1/4(0) + 1/2(0) + 1/4(0) = 3$$

Note that the existing image values do not change. The next step is to slide the mask over by one pixel and repeat the process, as follows:

$$1/4(0) + 1/2(0) + 1/4(0) + 1/2(3) + 1(0) + 1/2(5) + 1/4(0) + 1/2(0) + 1/4(0) = 4$$

Note this is the average of the two existing neighbors. This process continues until we get to the end of the row, each time placing the result of the operation in the location corresponding to center of the mask. When the end of the row is reached, the mask is moved down one row, and the process is repeated row by row. This procedure has been performed on the entire image, the process of **sliding, multiplying and summing** is called convolution.

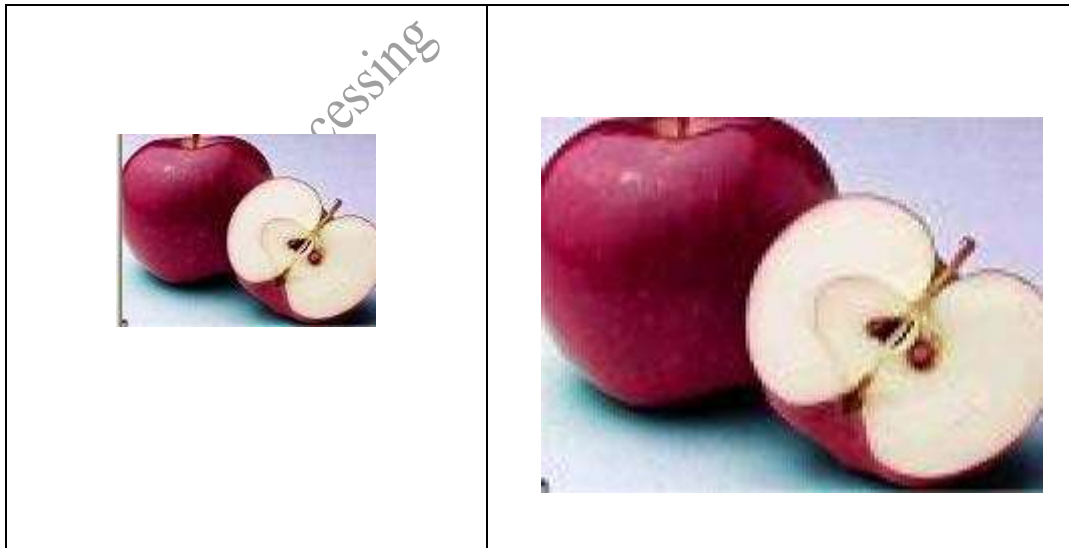
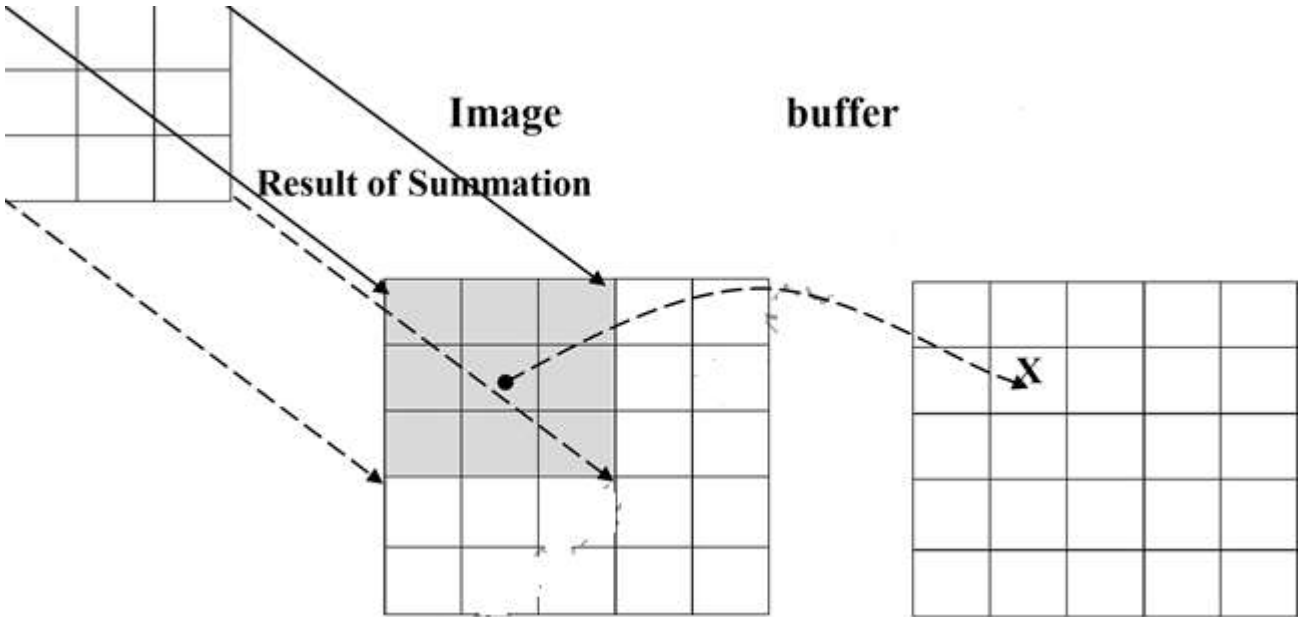


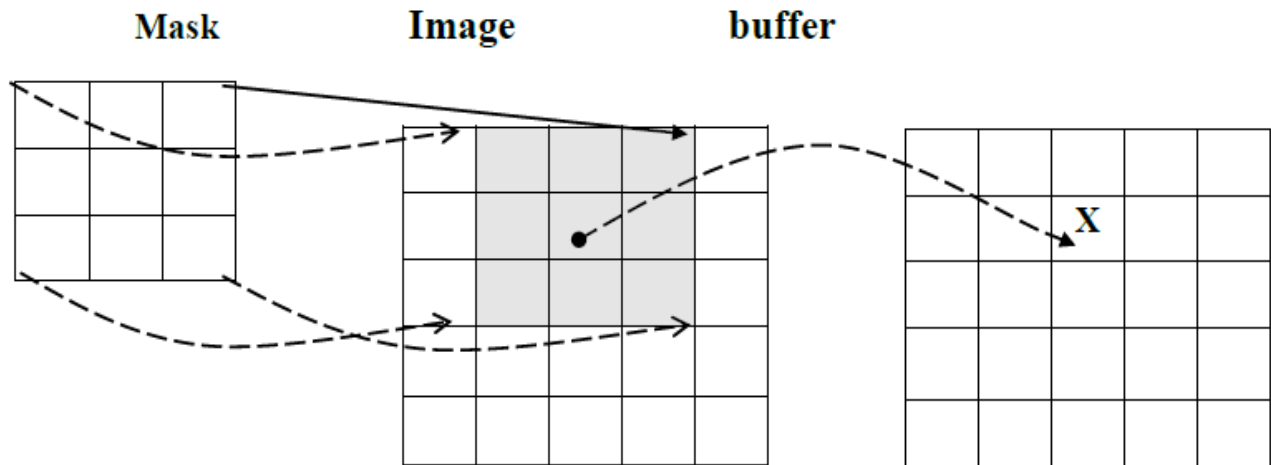
Figure (2.3): First _Order Hold Method

Note that the output image must be put in a separate image array called a buffer, so that the existing values are not overwritten during the convolution process.

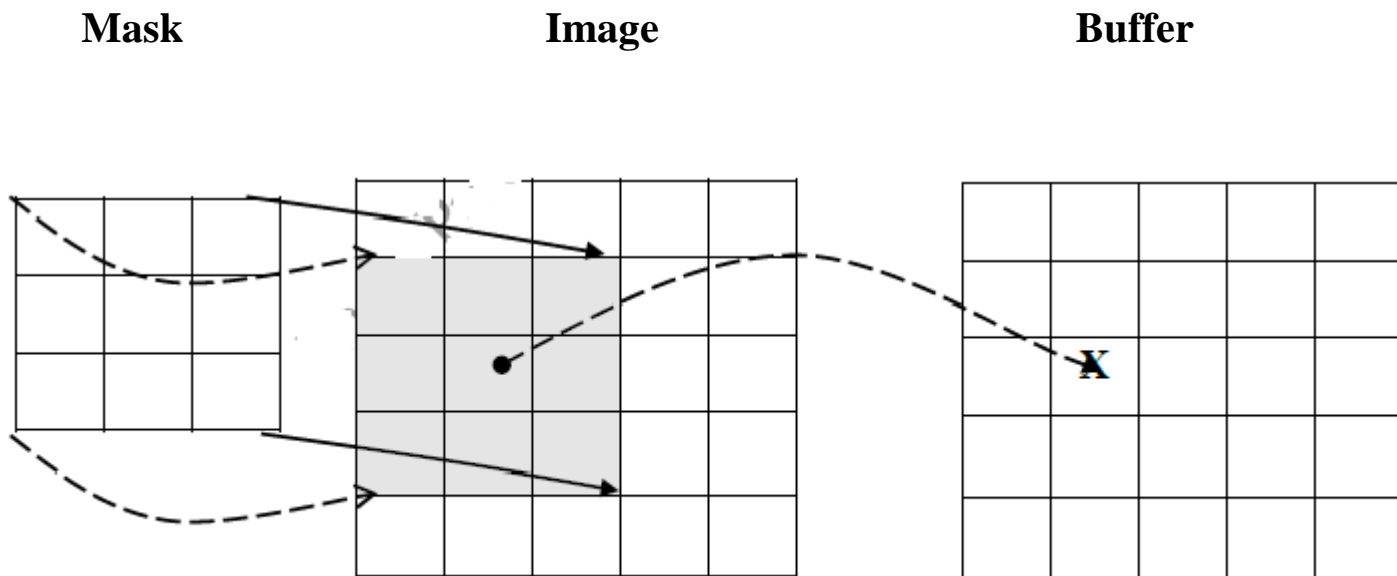
The convolution process Mask



- a. Overlay the convolution mask in the upper-left corner of the image. Multiply coincident terms, sum, and put the result into the image buffer at the location that corresponds to the masks current center, which is $(r,c)=(1,1)$.



- b.** Move the mask one pixel to the right , multiply coincident terms sum , and place the new results into the buffer at the location that corresponds to the new center location of the convolution mask which is now at $(r,c)=(1,2)$, continue to the end of the row.



- c. Move the mask down one row and repeat the process until the mask is convolved with the entire image. Note that we lose the outer row(s) and columns(s).

Why we use this convolution method when it require, so many more calculation than the basic averaging of the neighbors method? The answer is that many computer boards can perform convolution in hardware, which is generally very fast, typically much faster than applying a faster algorithm in software. Not, only first-order hold be performed via convolution, but zero-order hold can also achieved by extending the image with zeros and using the following convolution mask.

Zero-order hold convolution mask

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

Note that for this mask we will need to put the result in the pixel location corresponding to the lower-right corner because there is no center pixel.

These methods will only allows us to enlarge an image by a factor of $(2N-1)$, but what if we want to enlarge an image by something other than a factor of $(2N-1)$?

To do this we need to apply a more general method. We take two adjacent values and linearly interpolate more than one value between them. This is done by define an enlargement number k and then following this process:

1. Subtract the result by k .
2. Divide the result by k .

3. Add the result to the smaller value, and keep adding the result from the second step in a running total until all $(k-1)$ intermediate pixel locations are filled.

Example: we want to enlarge an image to three times its original size, and we have two adjacent pixel values 125 and 140.

1. Find the difference between the two values, $140-125=15$.
 2. The desired enlargement is $k=3$, so we get $15/3=5$.
 3. next determine how many intermediate pixel values we need : $K-1=3-1=2$. The two pixel values between the 125 and 140 are $125+5=130$ and $125+2*5=135$.
- We do this for every pair of adjacent pixels .first along the rows and then along the columns. This will allows us to enlarge the image by any factor of $K (N-1) +1$ where K is an integer and $N \times N$ is the image size.
 - To process opposite to enlarging an image is shrinking. This process is done by reducing the amount of data that need to be processed.
 - Two other operations of interest image geometry are: Translation and Rotation. These processes may be performed for many application specific reasons, for example to align an image with a known template in pattern matching process or make certain image details easier to see.

2.3.2 Image Shrinking

The process opposite to enlarge an image is shrinking it. This is not typically done to examine a ROI more closely but to reduce the amount of data that needs to be processed.

Image shrinking is accomplished by taking groups of pixels that are spatially adjacent and mapping them to one pixel. This can be done in one of three ways:

- **Averaging**
- **Median**
- **Decimation.**

For averaging method, we take all the pixels in each group (for example 2×2 block of pixels) and find the average gray level by summing the values select and divided by 2.

For the second method, median, we sort all the pixels values from lowest to highest and then select the middle value.

The third method decimation, also known as sub sampling, entails simply eliminating some of the data. For example to reduce the image by a factor of two, we simply take every other row and column and delete them.

EXAMPLE

Shrink the following sub image using the three shrinking methods (Averaging, Median, Decimation).

$$\begin{bmatrix} 6 & 8 \\ 1 & 9 \end{bmatrix}$$

I

Averaging

$$6+8+1+9=24/4=6$$

median

$$1, \underline{6}, 8, 9 = (6+8)/2=7$$

decimation

eliminate the first row and the first column and the result is 9

Example

if you have a sub image of size 2 × 2 , zoom the sub image to twice (2)times

using zero older hold method.

$$I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$I' = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \end{bmatrix} \text{ Row wise zooming}$$

$$I'' = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \text{ Row wise zooming}$$

Example

Enlarge the following sub image to three times its original size using the general method of zooming techniques.

$$I = \begin{bmatrix} 15 & 30 & 15 \\ 30 & 15 & 30 \end{bmatrix}_{2 \times 3}$$

Sol:

$$K=3, K-1=2$$

Row wise zooming:

Take the first two adjacent pixels. Which are 15 and 30.

Subtract 15 from 30: $30-15 = 15$.

Divide 15 by k: $15/k = 15/3 = 5 \text{ -----} \rightarrow \text{OP}$

Add **OP (5)** to the lower number: $15 + 5 = 20$.

Add **OP (5)** to 20 again. : $20 + 5 = 25$.

We do that 2 times because we have to insert **k-1** values.

Now repeat this step for the next two adjacent pixels. It is shown in the first table.

After inserting the values , you have to sort the inserted values in ascending order, so there remains a symmetry between them. It is shown in table 2

Table 1:

15	20	25	30	20	25	15
30	20	25	15	20	25	30

Table 2

15	20	25	30	<u>25</u>	<u>20</u>	15
30	<u>25</u>	<u>20</u>	15	20	25	30

Column wise zooming :

The same procedure has to be performed column wise. The procedure include taking the two adjacent pixel values, and then subtracting the smaller from the bigger one. Then after that , you have to divide it by k. Store the result as OP. Add OP to smaller one, and then again add OP to the value that comes in first addition of OP. Insert the new values;

The final output is :

Table 3

15	20	25	30	25	20	15
20	21	21	25	21	21	20
25	22	22	20	22	22	25
30	25	20	15	20	25	30

NOT: The dimensions of the original image were 2×3 . And the dimensions of the new image are 4×7 .

The formula thus is:

$(K \text{ (number of rows minus 1)} + 1) \times (K \text{ (number of cols minus 1)} + 1)$

H.W

Apply Zero-Order hold mask to zoom the following subimage>

$$I = \begin{bmatrix} 5 & 0 & 1 \\ 3 & 1 & 10 \\ 7 & 7 & 21 \end{bmatrix}_{3 \times 3}$$

2.4 Image Algebra

There are two primary categories of algebraic operations applied to image:

1. **Arithmetic operations.**
2. **Logic operations.**

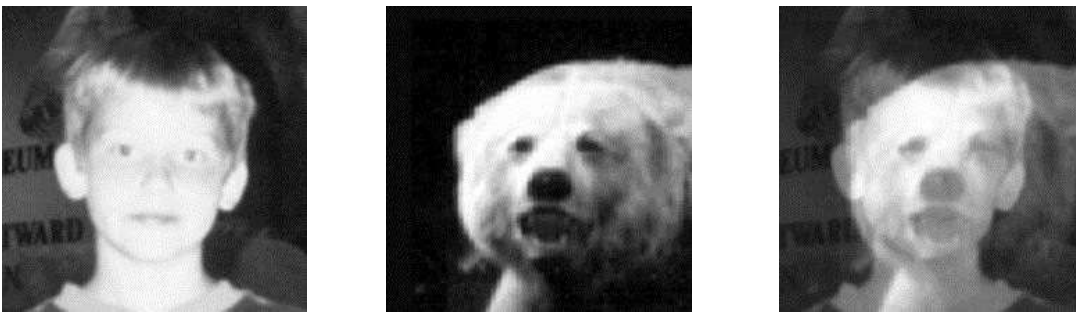
Addition, subtraction, division and multiplications comprise the **arithmetic operations**, while AND, OR and NOT makeup the **logic operations**. These operations which require only one image, and are done on a pixel –by-pixel basis.

To apply the arithmetic operations on two images, we simply operate on corresponding pixel values, which means that the value of a pixel in the output image depends only on the values of the corresponding pixels in the input images. Hence, the images normally have to be of the same size. For example to add image I_1 and I_2 to create I_3 :

$$\begin{matrix} I_1 & I_2 & I_3 \end{matrix}$$

$$\begin{pmatrix} 3 & 4 & 7 \\ 3 & 4 & 5 \\ 2 & 4 & 6 \end{pmatrix} + \begin{pmatrix} 6 & 6 & 6 \\ 4 & 2 & 6 \\ 3 & 5 & 5 \end{pmatrix} = \begin{pmatrix} 3+6 & 4+6 & 7+6 \\ 3+4 & 4+2 & 5+6 \\ 2+3 & 4+5 & 6+5 \end{pmatrix} = \begin{pmatrix} 9 & 10 & 13 \\ 7 & 6 & 11 \\ 5 & 9 & 11 \end{pmatrix}$$

- **Addition** is used to combine the information in two images as shown in figure (2.5). Or adding a constant value (scalar) to an image causes an increase (or decrease if the value is less than zero) in its overall brightness. Applications include development of image restoration algorithm for molding additive noise, and special effects, such as image morphing in motion pictures.
- **Subtraction** of two images is often used to **detect motion**, consider the case where nothing has changed in a sense; the image resulting from subtraction of two sequential image is filled with zero-a black image. If something has moved in the scene, subtraction produces a nonzero result at the location of movement. Subtraction process also used to detect the defects in the images. Applications include Object tracking , Medical imaging. Subtraction can result in a negative values for certain pixels. When this occurs with unsigned data types, such as uint8 or uint16, the imsubtract function truncates the negative value to zero (0), which displays as black



a. First Original image b. Second Original c. Addition of two images

Figure (2.5): Image Addition.



a. Original scene



b. Same scene later



c. Subtraction of scene a from scene b

Figure (2.6): Image Subtraction.

- Multiplication and Division are used **to adjust the brightness of an image**. (The Brightness adjustment is often used as a processing step in image enhancement). **Multiplication of the pixel values by a (scalar) smaller than one will darken the image, while multiplication by scalar greater than one will brighten the image.** Multiplication process often used for masking operations, see figure (2.9).



a. Cameraman image



b. X-ray image of hand



c. Multiplication of two images

Figure (2.7): Image Multiplication



a. Original image



b. Image divided by value < 1



c. Image divided by value > 1

Figure (2.8): Image Division



Figure 2.9 : (Image Multiplication by Scalar (smaller than one))

Logical operations apply only *to binary images*, whereas arithmetic operations apply to multi-valued pixels. Logical operations are basic tools in binary image processing, where they are used for tasks such as *masking*, *feature detection*, and *shape analysis*. Logical operations on entire image are performed pixel-by-pixel. Because the AND operation of two binary variables is 1 only when both variables are 1, the result at any location in a resulting AND image is 1 only if the corresponding pixels in the two input images are 1. As logical operation involve only one pixel location at a time, they can be done in place, as in the case of arithmetic operations. The XOR (exclusive OR) operation yields a 1 when one or other pixel (but not both) is 1, and it yields a 0 otherwise. The operation is unlike the OR operation, which is 1, when one or the other pixel is 1, or both pixels are 1.

	AND				OR				XOR			
input1	1	1	0	0	1	1	0	0	1	1	0	0
Input2	1	0	1	0	1	0	1	0	1	0	1	0
output	1	0	0	0	1	1	1	0	0	1	1	0

- Logical AND & OR operations are useful for the **masking and compositing** of images. For example, if we compute the AND of a binary image with **some** other image, then pixels for which the corresponding value in the binary image is 1 will be preserved, but pixels for which the corresponding binary value is 0 will be set to 0 (erased). Thus the binary image acts as a “**mask**” that **removes** information from certain parts of the image.
- On the other hand, if we compute the OR of a binary image with some other image, the pixels for which the corresponding value in the binary image is 0 will be **preserved**, but pixels for which the corresponding binary value is 1, will be set to 1 (cleared).

So, masking is a simple method to extract a region of interest (ROI) from an image

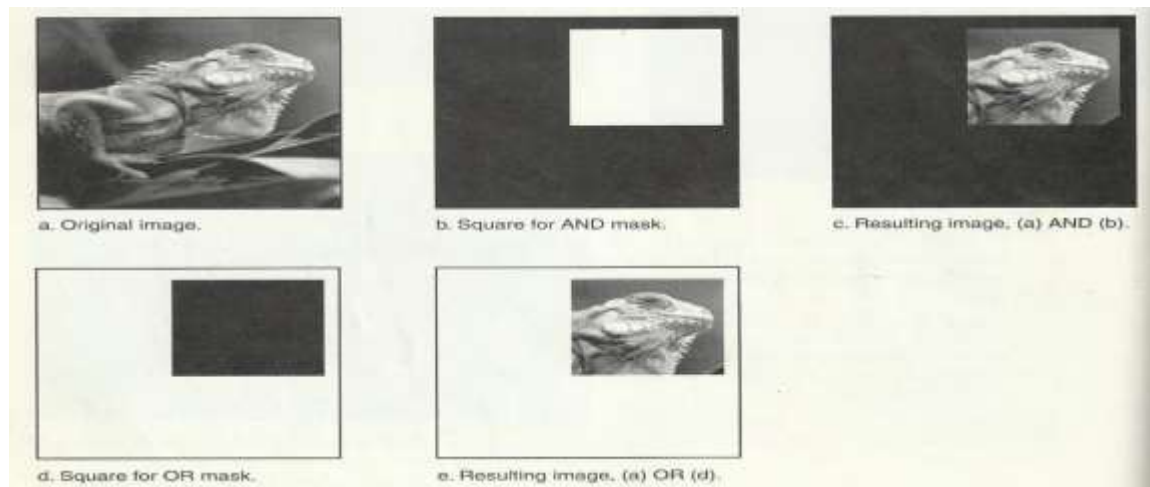


Figure (2.9): Image masking.

In addition to masking, logical operation can be used in feature detection. Logical operation can be used to compare between two images, as shown below:

- **AND**

This operation can be used to find the *similarity* white regions of two different images (it required two images).

$$g(x,y) = a(x,y) \wedge b(x,y)$$

- **Exclusive OR**

This operator can be used to find the *differences* between white regions of two different images(it requires two images).

$$g(x,y) = a(x,y) \otimes b(x,y)$$

- **NOT**

This operator can be performed on grey-level images, it's applied on only one image, and the result of this operation is the *negative* of the original image.

$$g(x,y) = 255 - f(x,y)$$

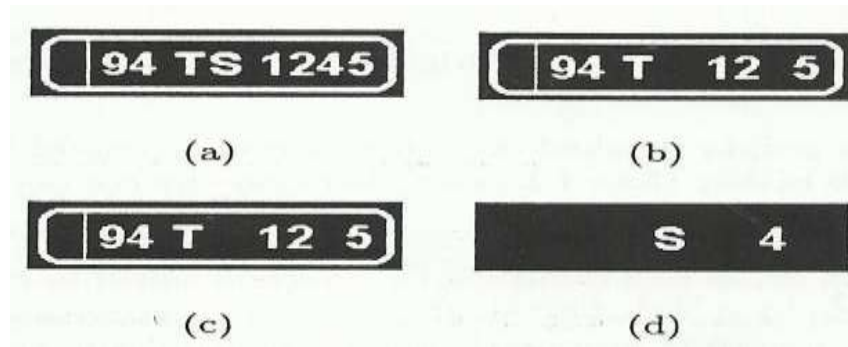
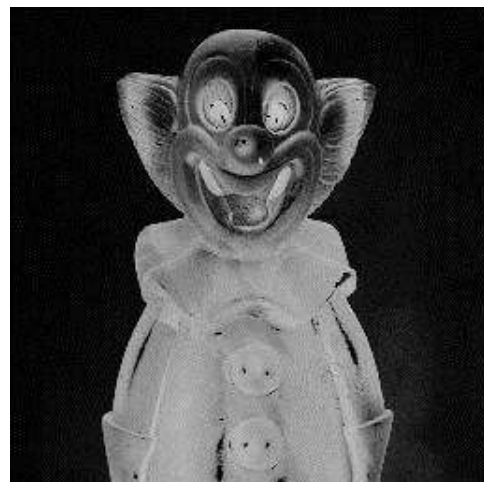


Figure 2.10: a) input image $a(x,y)$
b) input image $b(x,y)$, c) $a(x,y) \wedge b(x,y)$,
d) $a(x,y) \otimes b(x,y)$



a. Original image



b. Image after NOT operation.

Figure (2.11): Complement Image.

Example: A logic AND is performed on two images, suppose the two corresponding pixel values are $(111)_{10}$ is one image and $(88)_{10}$ in the second image. The corresponding bit strings are:

$$\begin{array}{rcl}
 (111)_{10} & \longrightarrow & 01101111_2 \\
 (88)_{10} & \longrightarrow & 01011000_2 \\
 & & \text{AND} \\
 & & \hline
 & & 01001000
 \end{array}$$

Example: Find the output image for the AND logical operation between the following sub images:

$$A = \begin{bmatrix} 50_{10} & 11_{10} \\ 18_{10} & 22_{10} \end{bmatrix}, \quad B = \begin{bmatrix} 10_{10} & 60_{10} \\ 33_{10} & 130_{10} \end{bmatrix}$$

2.5 Image Restoration: Image restoration methods are used to improve the appearance of an image by application of a restoration process that use *mathematical model* for image degradation.

Example of the type of degradation:

1. Blurring caused by motion or atmospheric disturbance.
2. Geometrics distortion caused by imperfect lenses.
3. Superimposed interface patterns caused by mechanical systems.
4. Noise from electronic source.

2.5.1 Noise Definition

Noise is any undesired information that contaminates an image. Noise appears in image from a variety of source.-The digital image a acquisition process, which converts an optical image into a continuous electrical

signal that is then sampled is the primary process by which noise appears in digital images.

At every step in the process there are fluctuations (تذبذب) caused by natural phenomena (ظاهرة) that add a random value to exact brightness value for a given pixel. In typical image the noise can be modeled with one of the following distribution:

1. Gaussian (“normal”) distribution.
2. Uniform distribution.
3. Salt _and _pepper distribution.



Figure (2. : Image Noise.

2.5.2 Noise Removal using Spatial Filters:

Spatial filtering is typically done for:

1. Remove various types of noise in digital images.
2. Perform some type of image enhancement.

[These filters are called spatial filter to distinguish them from frequency domain filter].

The three types of filters are:

1. Mean filters
2. Median filters (order filter)
3. Enhancement filters

Mean and median filters are used primarily to conceal or remove noise, although they may also be used for special applications. For instance, a mean filter adds “softer” look to an image. The enhancement filter highlights edges and details within the image.

Spatial filters are implemented with convolution masks. Because convolution mask operation provides a result that is weighted sum of the values of a pixel and its neighbours, it is called a linear filter.

Overall effects the convolution mask can be predicated based on the general pattern. For example:

- If the coefficients of the mask sum-to one, the average brightness of the image will be retained.
- If the coefficients of the mask sum to zero, the average brightness will be lost and will return a dark image.
- If the coefficients of the mask are alternatively positive and negative, the mask is a filter that returns edge information only.

- If the coefficients of the mask are all positive, it is a filter that will blur the image.

The mean filters, are essentially averaging filter. They operate on local groups of pixel called neighborhoods and replace the center pixel with an average of the pixels in this neighborhood. This replacement is done with a convolution mask such as the following 3×3 mask

Arithmetic mean filter smoothing or low-pass filter.

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

Note that the coefficient of this mask sum to one, so the image brightness will be retained, and the coefficients are all positive, so it will tend to blur the image. This type of mean filter soothes out local variations within an image, so it essentially a low pass filter. **So a low filter can be used to attenuate image noise that is composed primarily of high frequencies components.**



a. Original image



b. Mean filtered image

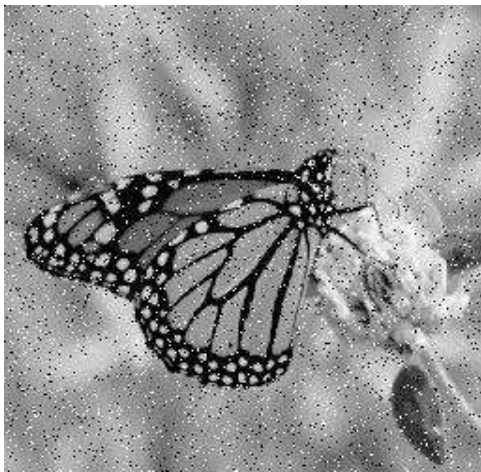
Figure (2.13): Mean Filter.

The median filter is a non linear filter (order filter). These filters are based on as specific type of image statistics called order statistics.

Typically, these filters operate on small sub image, “Window”, and replace the centre pixel value (similar to the convolution process). **Order statistics** is a technique that arranges the entire pixel in sequential order, given an $N \times N$ window (W) the pixel values can be ordered from smallest to the largest.

$$I_1 \leq I_2 \leq I_3 \dots \dots \dots < I_N$$

Where $I_1, I_2, I_3 \dots \dots \dots, I_N$ are the intensity values of the subset of pixels in t



a. Salt and pepper noise



b. Median filtered image (3x3)

Figure (2.14): Median Filter

Example : Given the following 3×3 neighborhood

$$\begin{pmatrix} 5 & 5 & 6 \\ 3 & 4 & 5 \\ 3 & 4 & 7 \end{pmatrix}$$

Solution:

1. sort the value in order of size (3,3,4,4,5,5,5,6,7) ;
2. then we select the middle value ,in this case it is 5.
3. The middle value 5 is then placed in center location.

H.w/ Apply a median filter (Order Statistic) on the following assumed image, using (3×3) Window size:

20	44	56	100	90
37	45	55	200	90
22	80	100	100	50
20	90	200	200	100
20	33	210	180	230
22	80	100	100	150

Solution:

0	0	0	0	0
0	45	80	90	0
0	55	100	100	0
0	80	100	180	0
0	80	100	180	0

- **Not:** A median filter can use a neighborhood of any size, but 3X3, 5X5 and 7X7. The output image must be written to a separate image (a buffer); so that the results are not corrupted as this process is performed. (The median filtering operation is performed on an image by applying the sliding window concepts).

- **The Procedure of median filter:**

- The window is overlaid on the upper left corner of the image, and the median is determined.
- This value is put into the output image (buffer) corresponding to the centre location of the window.
- The window is then slide one pixel over, and the process is repeated
- When the end of the row is reached, the window is slide back to the left side of the image and down one row, and the process is repeated.
- This process continues until the entire image has been processed.

Not: the outer rows and columns are not replaced.. And these “wasted” rows and columns are often filled with zeros (or cropped off the image). For example, with 3X3 mask, we lose one outer row and column, a 5X5 mask we lose two rows and columns.

The maximum and minimum filters: are two order filters that can be used for elimination of salt- and-pepper noise.

- **The maximum filter** selects the largest value within an ordered window of pixels values and replaces the central pixel with the lightest one
- **The minimum filter** selects the smallest value within an ordered window of pixels values and replaces the central pixel with the darkest one in the ordered window

The minimum filters works best for salt- type noise, and the maximum filters work best for pepper-type noise.

The procedure of minimum and maximum filter:

- minimum and maximum filter order filters can be defined to select a specific pixel rank within the ordered set. For example we may find for certain type

of pepper noise that selecting the second highest values works better than selecting the maximum value.

Not: This type of ordered selection is very sensitive to type of images and their use it is application specific. In general a minimum or low rank filter will tend to darken an image and a maximum or high rank filter will tend to brighten an image

The **midpoint filter** : It is actually both order and mean filter because it rely on ordering the pixel values , but then calculated by an averaging process. This midpoint filter is the average of the maximum and minimum within the window as follows:

$$\text{Order set} = I_1 \leq I_2 \leq I_3 \dots \dots \dots \leq I_N^2.$$

$$\text{Midpoint} = (I_1 + I_N^2) / 2$$

The midpoint filter is most useful for Gaussian and uniform noise.

2.5.3 Enhancement filter:

The enhancement filters are:

1. Laplacian type.
2. Difference filter.

These filters will tend to bring out, or enhance details in the image. Example of convolution masks for the Laplacian-type filters are:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} -2 & 1 & -2 \\ 1 & 5 & 1 \\ -2 & 1 & -2 \end{pmatrix}$$



a. Original image



b. Laplacian filtered image

Figure (2.15): Laplacian Filter.

NOT: The Laplacian type filters will enhance details in all directions equally. T

The difference filters will enhance details in the direction specific to the mask selected. There are four different filter convolution masks, corresponding to lines in the vertical, horizontal and two diagonal directions.

Vertical $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$	Horizontal $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$
Diaogonal1 $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$	Diaogonal2 $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$

Figure (2.16): Difference Filter

2.6 Image Quantization

Image quantization is the process of reducing the image data by removing some of the detail information by mapping group of data points to a single point. This can be done by:

1. **Gray_Level reduction (reduce pixel values $I(r, c)$).**
2. **Spatial reduction (reduce the spatial coordinate (r, c)).**

The simplest method of gray-level reduction is **Thresholding**.

The Procedure:

- We select a threshold gray_level
- and set everything above that value equal to “1” and everything below the threshold equal to “0”.

NOT: This effectively turns a gray_level image into a binary (two level) image

Application: is often used as a preprocessing step in the extraction of object features, such as shape, area, or perimeter. Also, the gray_level reduction is the process of taking the data and reducing the number of bits per pixel. **This can be done very efficiency by masking the lower bits via an AND operation. Within this method, the numbers of bits that are masked determine the number of gray levels available.**

Example:

We want to reduce 8_bit information containing 256 possible gray_level values down to 32 gray levels possible values.

This can be done by applying the (AND) logical operation for each 8-bit value with the bit string **1111000**. this is equivalent to dividing by eight(2^3), corresponding to the lower three bits that we are masking and then shifting

the result left three times. [Gray_level in the image 0-7 are mapped to 0, gray_level in the range 8-15 are mapped to 8 and so on].

We can see that by masking the lower three bits we reduce 256 gray levels to 32 gray levels:

$$256 \div 8 = 32$$

The general case requires us to mask k bits, where 2^k is divided into the original gray-level range to get the quantized range desired. Using this method, we can reduce the number of gray levels to any power of 2: 2,4,6,8, 16, 32, 64 or 128.

- Image quantization by masking to 128 gray levels, this can be done by ANDing each 8-bit value with bit string 11111110(2^1).
- Image quantization by masking to 64 gray_level. This can be done by ANDing each 8-bit value with bit string 11111100(2^2).

As the number of gray levels decreases, we can see increase in a phenomenon called contouring.

Contouring appears in the image as false edges, or lines as a result of the gray_level quantization method.

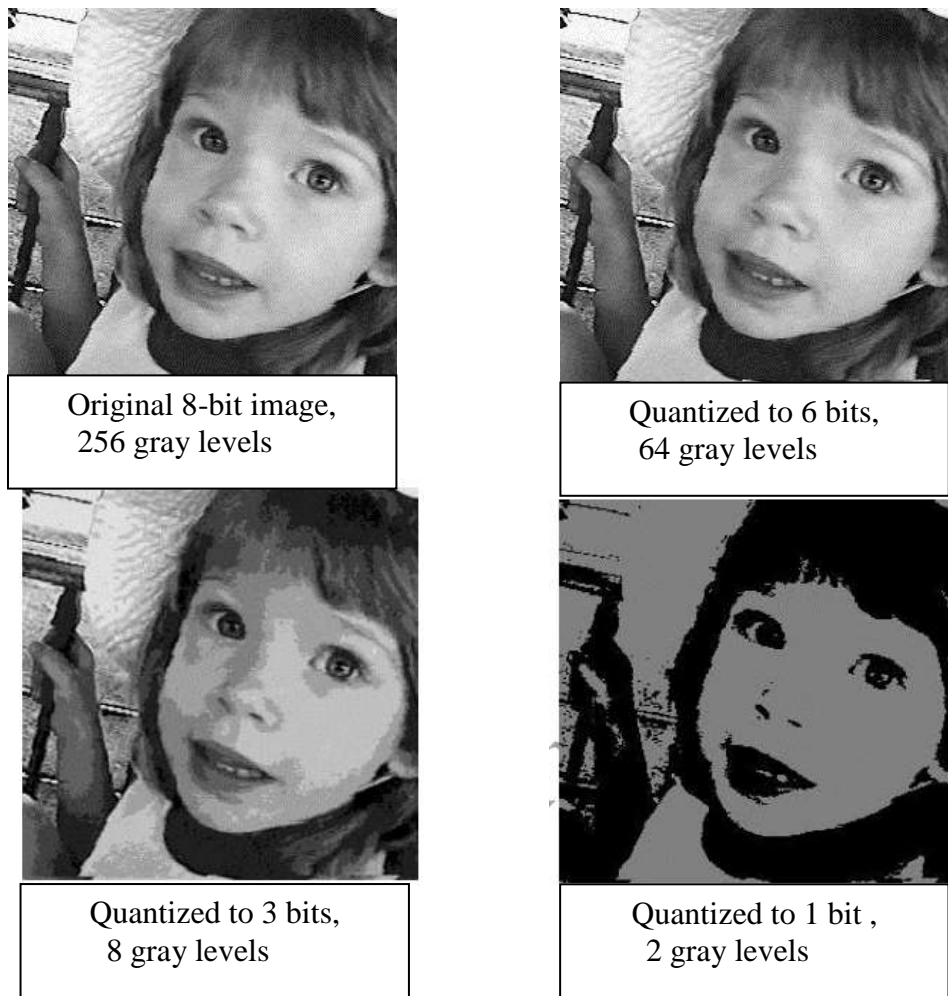
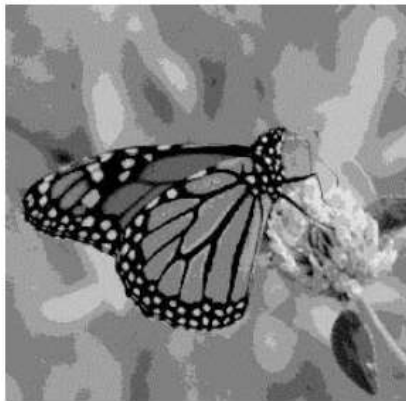


Figure (2-17): False Contouring

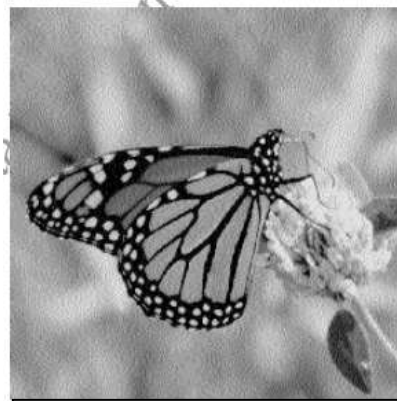
The **false contouring** effect can be visually improved upon by using an **IGS (improved gray-scale) quantization method**. In this method (IGS) the improvement will be by adding a small random number to each pixel before quantization, which results in a more visually pleasing appearance.



Original Image



Uniform quantization
to 8 levels (3 bits)



IGS quantization
to 8 levels (3 bits)

Figure (2-18): IGS quantization

2.7 Edge Detection

Detecting edges is a basic operation in image processing. The edges of items in an image hold much of the information in the image.

The edges tell you where:

- Items are.
- Their size.
- shape
- something about their texture.

Application : Edge detection methods are used as a first step in the line detection processes, and they are used to find object boundaries by marking potential edge points corresponding to place in an image where rapid changes in brightness occur. After these edge points have been marked, they can be merged to form lines and objects outlines.

Edge detection operations are based on the idea that edge information in an image is found by looking at the relationship a pixel has with its neighbors. If a pixel gray_level values similar to those around it, there is probably not an edge at that point. However, if a pixel has neighbors with widely varying gray levels, it may represent an edge point.

In general, an edge is defined by a discontinuity in gray-level values. Ideally, an edge separates two distinct objects. **In practice, edges are caused by:**

- Change in color or texture or
- Specific lighting conditions present during the image acquisition process.

The following figure illustrates the difference between an ideal edge and a real edge.

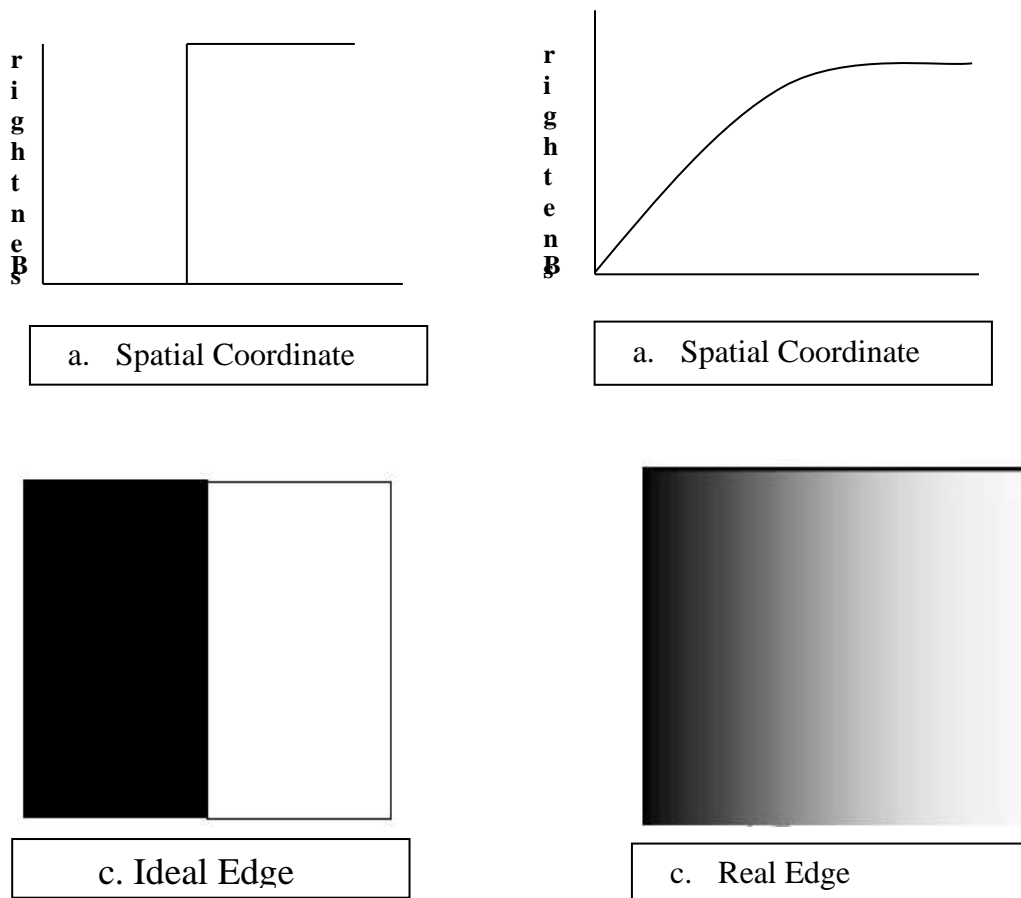


Figure (2-19): Ideal vs. Real Edge

The vertical axis represents brightness, and the horizontal axis shows the spatial coordinates. The rapid change in brightness characterizes an ideal edge. In the figure (b) we see the representation of real edge, which change gradually. This gradual change is a minor form of blurring caused by:

- Imaging devices
- The lenses
- Or the lighting and it is typical for real world (as opposed to computer _generated) images.

An edge is where the gray level of the image moves from an area of low values to high values or vice versa. The edge itself is at the centre of this transition. The detected edge gives a bright spot at edge and dark area everywhere else. This means it is the slope or rate of change of the gray level in the edge.

- **The calculation of the derivative (the slope) of an image in all direction is achieved based on : [Convolution of the image with masks]**

The Procedure:

- take a 3×3 array of numbers
- and multiply it point by point with 3×3 section of the image you sum the products and place the result in the centre point of the image.

- **The question in this operation is how to choose the 3×3 mask?**

There are several masks that amplify the slope of the edge. Take the simple one-dimensional case and take as an example points on the ideal edge near the edge.

Example : They could have values such as [3 5 7]. The slope through these three points is $(7-3)/2=2$. If you convolve these three points with [-1 0 1] you have $-3+7=4$.

- **The convolution amplified the slope, and the result is a large number at the transition point in the edge.**

There are two basic principles for each edge detector mask:

- **First:** the number in the mask sum to zero. If 3×3 areas of an image contains a constant value (such as all ones), then there are no edges in that area. The result of convolving that area with a mask should be zero. If the numbers in the mask sum to zero, then convolving the mask with a constant area will result in the correct answer of zeros.
- **Second:** the masks should approximate differentiation or amplify the slope of the edge. The simple example $[-1 \ 0 \ 1]$ given earlier showed how to amplify the slope of the edge.

2.7.1 Edge Detection masks

1- Sobel Operator: The Sobel edge detection masks look for edges in both the horizontal and vertical directions and then combine this information into a single metric. These two masks are as follows:

Row Mask $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	Column Mask $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
---	--

The Procedure:

- These masks are each convolved with the image.
- At each pixel location we now have two numbers: **S1**, corresponding to the result from the row mask and **S2** from the column mask.

- Use S_1, S_2 to compute two matrices, the edge magnitude and the edge direction (angle of orientation of the edge), which are defined as follows:

$$\text{Edge Magnitude (EM)} = \sqrt{S_1^2 + S_2^2}$$

$$\text{Edge Direction (ED)} = \tan^{-1} \left[\frac{S_1}{S_2} \right]$$

Example: determine the edges of the following sub image using Sobel edge detector.

$$\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & 1 & 0 & 3 \\ 2 & 4 & 1 & 5 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

Row Mask

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Column Mask

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Solution:

$$S_1 = (2+8+1) - (1+4+2) = 4,$$

$$S_2 = (2+0+1) - (1+2+2) = -2,$$

$$\text{EM}(1,1) = \sqrt{4^2 + (-2)^2} = 4.47$$

$$\text{EM}(1,2) =$$

$$\text{EM}(2,1) =$$

$$\text{EM}(2,2) =$$

.....

2- Prewitt Operator: The Prewitt is similar to the Sobel but with different mask coefficients. The masks are defined as follows:

Row Mask	Column Mask
$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

The Procedure:

- These masks are each convolved with the image.
- At each pixel location we find two numbers: **P1** corresponding to the result from the row mask and **P2** from the column mask.
- Use P1, P2 to **determine two metrics, the edge magnitude and edge direction (angle of orientation of the edge),** which are defined as follows:

$$\text{Edge Magnitude} = \sqrt{P_1^2 + P_2^2}$$

$$\text{Edge Direction} = \text{Tan}^{-1} \left[\frac{P_1}{P_2} \right]$$

3- Kirsch Compass (محيط) Mask: the Kirsch edge detection masks are called compass masks because they are defined by taking a single mask and rotating it to the eight major compass orientations (اتجاهات):

North, north-east, east, south-east, south, south-west, and west and northwest edges in an image. The masks are defined as follows:

$$\begin{array}{cccc}
\begin{pmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{pmatrix} & \begin{pmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{pmatrix} & \begin{pmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix} & \begin{pmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix} \\
\mathbf{K}_0 & \mathbf{K}_1 & \mathbf{K}_3 & \mathbf{K}_4 \\
\\
\begin{pmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{pmatrix} & \begin{pmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{pmatrix} & \begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{pmatrix} & \begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{pmatrix} \\
\mathbf{K}_4 & \mathbf{K}_5 & \mathbf{K}_6 & \mathbf{K}_7
\end{array}$$

The edge magnitude is defined as the maximum value found by the convolution of each of the mask, with the image.

[Given a pixel, there are eight directions you can travel to a neighboring pixel (above, below , left ,right ,upper left, upper right, lower left, lower right). Therefore there are eight possible directions for an edge. The directional edge detectors can detect an edge in only one of the eight directions. If you want to detect only left to right edges, you would use only one of eight masks. If; however you want to detect all of the edges, you would need to perform convolution over an image eight times using each of the eight masks.

4- Robinson Compass Masks: the Robinson compass masks are used in a manner similar to the Kirsch masks but are easier to implement because they rely only on coefficient of 0, 1 and 2, and are symmetrical about their directional axis-the axis with the zeros, we only need to compute the result

on four of the mask, the results. From the other four can be obtained by negating the results from the first four. The masks are as follows:

$$\begin{array}{cccc}
 \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} & \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{pmatrix} \\
 \mathbf{R}_0 & \mathbf{R}_1 & \mathbf{R}_3 & \mathbf{R}_4 \\
 \\
 \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} & \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix} & \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} & \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix} \\
 \mathbf{R}_5 & \mathbf{R}_6 & \mathbf{R}_7 &
 \end{array}$$

The edge magnitude is defined as the maximum value found by the convolution of each of the masks with the-image. The edge detection is defined by the mask that produces the maximum magnitude.

It's interesting to note that masks \mathbf{R}_0 and \mathbf{R}_7 are the same as the Sobel masks. We can see that any of the edge detection masks can be extended by rotating them in a manner like these compass masks which allow us to extract explicit information about edge in any direction.

5- Laplacian Operators: the Laplacian operator described here are similar to the ones used for pre-processing (as described in enhancement filter). The three Laplacian masks that follow represent different approximation of the Laplacian masks are rationally symmetric, which means edges at all

orientation contribute to the result. They are applied by selecting one mask and convolving it with the image selecting one mask and convolving it with the image.

Laplacian masks

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

These masks differ from the Laplacian type previously described in that the centre coefficients have been decreased by one. So, if we are only interested in edge information, the sum of the coefficients should be **zero**. If we want to retain most of the information the coefficient should sum to a number greater than zero. Consider an extreme example in which the centre coefficient value will depend most heavily up on the current value, with only minimal contribution from surrounding pixel values.

6- Other Edge Detection Methods

Edge detection can also be performed by *subtraction*. Two methods that use subtraction to detect the edge are **Homogeneity operator** and **Difference operator**.

The workflow of **homogeneity operator** is illustrated as follows:

- subtracts each of the pixels next to the center of the $n \times n$ area (where n is usually 3) from the center pixel.

- The result is the maximum of the absolute value of these subtractions. Subtraction in a homogenous region produces zero and indicates an absence of edges.
- A high maximum of the subtractions indicates an edge. This is a quick operator since it performs only subtraction- eight operations per pixel and no multiplication.
- This operator then requires thresholding.

If there is no thresholding then the resulting image looks like a faded copy of the original. Generally thresholding at 30 to 50 gives good result. The thresholding can be varied depending upon the extent of edge detection desired.

The workflow of **difference operator** based on differentiation through:

- calculating the differences between the pixels that surround the centre pixel of an $n \times n$ area.
- This operator finds the absolute value of the difference between the opposite pixels, the upper left minus the lower right, upper right minus the lower left, left minus right, and top minus bottom. The result is the maximum absolute value. as in the homogeneity case,
- this operator requires thresholding. But it is quicker than the homogeneity operator since it uses four integer subtractions as against eight subtractions in homogeneity operator per pixel.

Example : Shown below is how the two operators detect the edge:

Consider an image block with centre pixel intensity 5,

1	2	3
4	5	6
7	8	9

Output of *homogeneity operator* is:

$$\text{Max of } \{ |5-1|, |5-2|, |5-3|, |5-4|, |5-6|, |5-7|, |5-8|, |5-9| \} = 4$$

Output of *difference operator* is:

$$\text{Max of } \{ |1-9|, |7-3|, |4-6|, |2-8| \} = 8$$


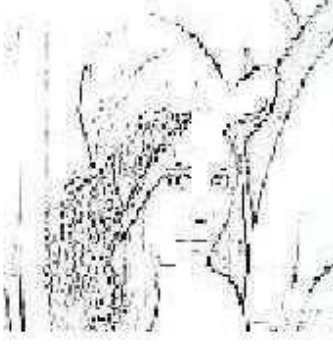



		
Original Image	Edge detect by Sobel' horizontal mask	Edge detect by Sobel overall mask
		
Edge detect by Gaussian mask	Edge detect by Prewitt mask	

Figure (2-) : Example of Edge Operators