Baghdad university — Fourth Year
College of education — Lecturer: Dr. Hussein L. Hussein
Image processing

<u>Syllabus(Image processing)</u>

Introduction to Computer Vision and Image Processing

What Is Digital Image Processing?

An image may be defined as a two-dimensional function, *f*(*x*, *y*) ,where *x* and *y* are *spatial* (plane) coordinates, and the amplitude of (*f*) at any pair of coordinates (*x*, *y*) is called the *intensity* or *gray level* of the image at that point.

When *x*, *y*, and the intensity values of *f* are all finite, discrete quantities, we call the image a *digital image*.

Note that a digital image is a representation of a two-dimensional image as a finite set of digital values composed of a finite number of elements, each of which has a particular location and value. These elements are called *picture elements*, *image elements*, , and *pixels*. *Pixel* is the term used most widely to denote the elements of a digital image

One picture is worth more than ten thousand words.

The Human Visual System

The Human Visual System (HVS) has two primary components:

• Eye.

• Brian.

The structure that we know the most about is the image receiving sensors (the human eye).

The brain can be thought as being an information-processing unit analogous to the computer in our computer imaging system. These two are connected by the optic nerve, which is really a bundle of nerves that contains the pathways for visual information to travel from the receiving sensor (the eye) to the processor (the brain).

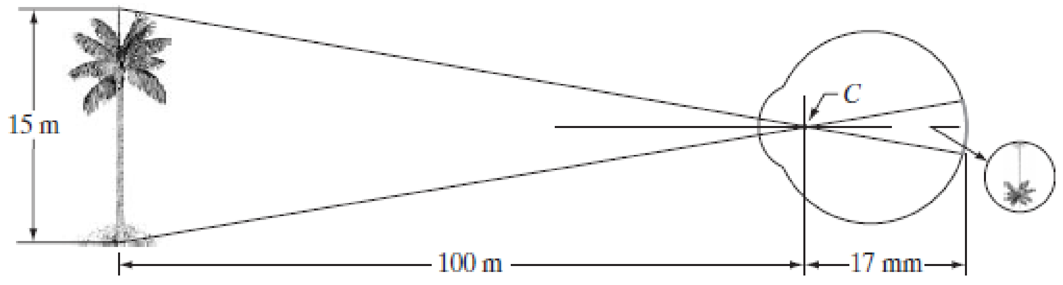


Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

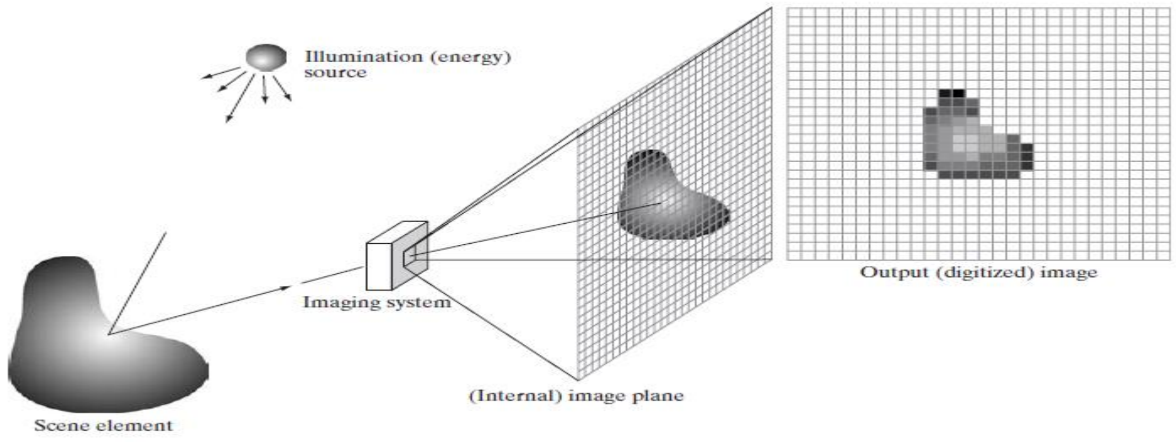


FIGURE 1.1 An example of the digital image acquisition process.
(a) Energy ("illumination") source. (b) An element of a scene. (c) Imaging system.
(d) Projection of the scene onto the image plane. (e) Digitized image.

1.1 Computer Imaging

Can be defined an acquisition and processing of visual information by computer. Computer representation of an image requires the equivalent of many thousands of words of data, so the massive amount of data required for image is a primary reason for the development of many sub areas with field of computer imaging, such as image compression and segmentation .Another important aspect of computer imaging involves the ultimate "receiver" of visual information in some case the human visual system and in some cases the human visual system and in others the computer itself.

Computer imaging can be separate into two primary categories:

1. **Computer Vision.** **2. Image Processing.**

(In computer vision application the processed images output for use by a computer, whereas in image processing applications the output images are for human consumption).

These two categories are not totally separate and distinct. The boundaries that separate the two are fuzzy, but this definition allows us to explore the differences between the two and to explore the difference between the two and to understand how they fit together (Figure 1.1).

Computer imaging can be separated into two different but overlapping areas.

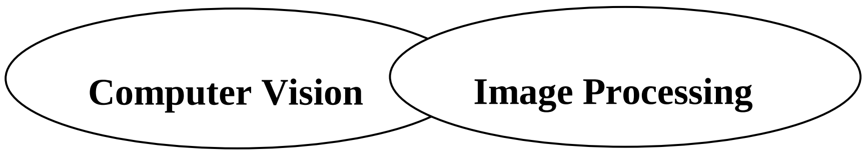


Figure (1.1): Computer Imaging.

Historically, the field of image processing grew from electrical engineering as an extension of the signal processing branch, whereas are the computer science discipline was largely responsible for developments in computer vision.

1.2 Computer Vision

Computer vision emulate human vision, that's mean: understanding the scene based on image data.One of the major topics within this field of computer vision is image analysis.

Image Analysis: involves the examination of the image data to facilitate solving vision problem.

The image analysis process involves two other topics:

- **Feature Extraction**: is the process of acquiring higher level image information, such as shape or color information.
- **Pattern Classification**: is the act of taking this higher –level information and identifying objects within the image.

Computer vision systems are used in many and various types of environments, such as:

1. Manufacturing Systems: computer vision is often used for quality control, where the computer vision system will scan manufactured items for defects, and provide control signals to a robotics manipulator to remove detective part automatically.
2. Medical Community: current example of medical systems to aid neurosurgeons (جراحة الاعصاب) during brain surgery, systems to diagnose skin tumors (سرطان الجلد) automatically.

3. The field of Law Enforcement (تنفيذ القانون) and security in an active area for computer vision system development, with application ranging from automatic identification of fingerprints to DNA analysis.
4. Infrared Imaging (صورِاشعة تحت الحمراء)
5. Satellites orbiting (مدار الفضائيات).

1.3. Image Processing

Image processing does some transformations on image. That means may be it does some smoothing, sharpening, contrasting, stretching on the image for making image more enhancive & readable that is input and output of a process are images.. In other words the image are to be examined and a acted upon by people.

The major topics within the field of image processing include:

1. Image restoration.
2. Image enhancement.
3. Image compression.

1.3.1 Image Restoration

Is the process of taking an image with some known, or estimated degradation, and restoring it to its original appearance. Image restoration is often used in the field of photography or publishing where an image was somehow degraded but needs to be improved before it can be printed (Figure 1.2).

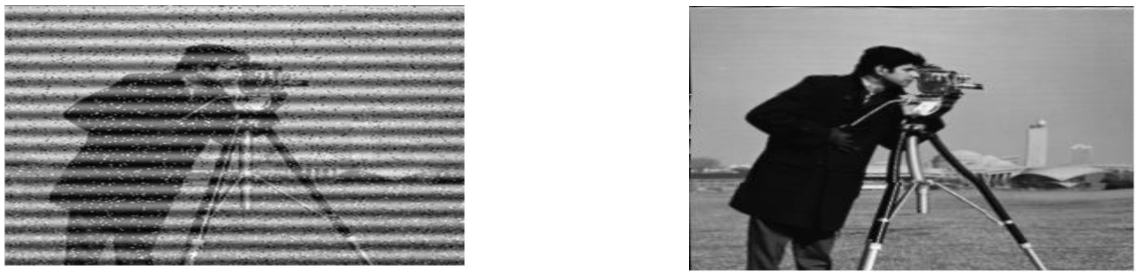

a. Image with distortion **b. Restored image**

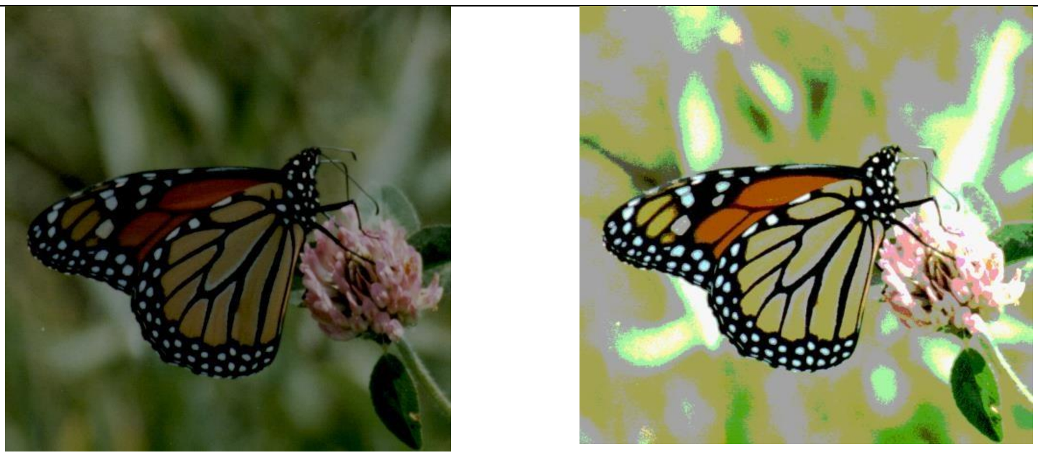
Figure (1.2) Image Restoration

1.3.2 Image Enhancement

Involves taking an image and improving it visually, typically by taking advantages of human Visual Systems responses. One of the simplest enhancement techniques is to simply stretch the contrast of an image.

Enhancement methods tend to be problem specific. For example, a method that is used to enhance satellite images may not suitable for enhancing medical images.

Although enhancement and restoration are similar in aim, to make an image look better. Restoration method attempt to model the distortion to the image and reverse the degradation, where enhancement methods use knowledge of the human visual systems responses to improve an image visually.



a. image with poor contrast **b. Image enhancement by contrast**

Figure (1.3) Image Enhancement

1.3.3 <u>Image Compression</u>

Involves reducing the typically massive amount of data needed to represent an image. This done by eliminating data that are visually unnecessary and by taking advantage of the redundancy that is inherent in most images. Image data can be reduced 10 to 50 times, and motion image data (video) can be reduced by factors of 100 or even 200.

a. Image before compression. b. Image after compression.

92KB 6.59 KB

1.4. Image Processing Applications

Image processing systems are used in many and various types of environments, such as:

1. Medical community has many important applications for image processing involving various type diagnostics imaging , as an example Magnetic Resonance Imaging (MRI)scanning, that allows the medical professional to look into the human body without the need to cut it open, CT scanning, X-RAY imaging .
2. Computer – Aided Design (CAD), which uses tools from image processing and computer graphics, allows the user to design a new building or spacecraft and explore it from the inside out.
3. Virtual Reality is one application that exemplifies (يمثل) future possibilities
4. Machine/Robot vision: Make robot able to see things , identify them , identify the hurdles (العقبات)

1.5 Computer Imaging Systems

Computer imaging systems are comprised of two primary components types, hardware and software. The hard ware components can be divided into image acquiring sub system (computer, scanner, and camera) and display devices (monitor, printer).The software allows us to manipulate the image and perform any desired processing on the image data.

1.6. Digitization

The process of transforming a standard video signal into digital image. This transformation is necessary because the standard video signal in analog (continuous) form and the computer requires a digitized or sampled version of that continuous signal. The analog video signal is turned into a digital image by sampling the continuous signal at affixed rate. In the figure below we see one line of a video signal being sampled (digitized) by instantaneously measuring the voltage of the signal

(amplitude) at fixed intervals in time.

The value of the voltage at each instant is converted into a number that is stored, corresponding to the brightness of the image at that point. Note that **the image brightness of the image at that point** depends on both the intrinsic properties of the object and the lighting conditions in the scene.

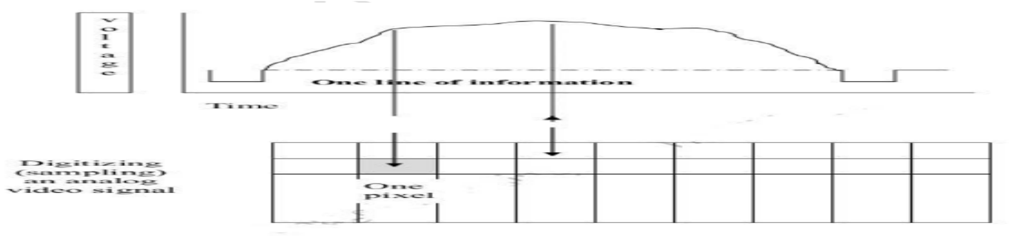


Figure (1.5) Digitizing (Sampling) an Analog Video Signal

The image can now be accessed as a two-dimension array of data , where each data point is referred to a pixel (picture element).for digital images we will use the following notation :

I(r,c) = The brightness of image at the point (r,c)

Where r = row and c = column.

"When we have the data in digital form, we can use the software to process the data". The digital image is 2D- array as:

$$\begin{pmatrix} I(0,0) & I(0,1) & \ldots\ldots\ldots\ldots & I(0,N-1) \\ I(1,0) & I(1,1) & \ldots\ldots\ldots\ldots & I(1,N-1) \\ \ldots & \ldots & \ldots\ldots\ldots\ldots & \ldots \\ \ldots & \ldots & \ldots\ldots\ldots\ldots & \ldots \\ I(N-1,0) & I(N-1,1) & \ldots\ldots\ldots\ldots & I(N-1,N-1) \end{pmatrix}$$

In above image matrix, the image size is (N×N) [matrix dimension] then:

$Ng = 2^{m}$ ………..(1)

Where Ng denotes the number of gray levels m, where m is the no. of bits contains in digital image matrix.

<u>Example</u> : If we have (6 bit) in 128 * 128 image .Find the no. of gray levels to represent it ,then find the no. of bit in this image?

<u>Solution</u>:

$N_g = 2^{6} = 64$ Gray Level

$N_b = 128 * 128 * 6 = 9.8304 * 10^{4}$ bit

1.7 Image Resolution

Pixels are the building blocks of every digital image. Clearly defined squares of light and color data are stacked up (مكدسة) next to one another both horizontally and vertically. Each **picture element (pixel for short)** has a dark to light value from 0 (solid black) to 255 (pure white).

That is, there are 256 defined values. **A gradient** (ميل, نسبة الانحدار) is the gradual transition from one value to another in sequence.

The resolution has to do with ability to separate two adjacent pixels as being separate, and then we can say that we can resolve the two. The concept of resolution is closely tied to the concepts of spatial frequency. There are three types of image resolution:

1- ***Vertical resolution***: the number of M rows in the image (image scan line).

2- ***Horizontal resolution***: the number of N columns in the image.

3- ***Spatial frequency resolution***: It is represent by the multiplication (M x N) and its closely tied to the concept of spatial frequency that refers to how rapidly the signal is changing in space, and the signal has two values for brightness 0 and maximum. If we use this signal for one line (row) of an image and then repeat the line down the entire image, we get an image of vertical stripes. If we increase this frequency the strips get closer and closer together, until they finally blend together as shown in figure below, note that the higher the resolution the more details (high frequency).

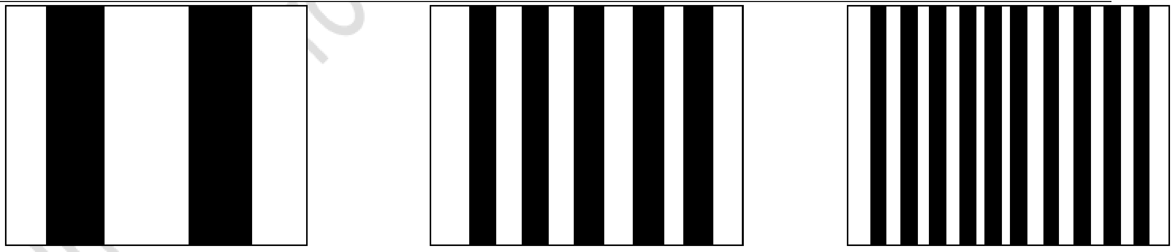


Figure (1.4) Resolution and Spatial frequency.

In computers, resolution is the number of pixels (individual points of color) contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis.

The sharpness of the image on a display depends on the resolution and the size of the monitor. The same pixel resolution will be sharper on a smaller monitor and gradually lose sharpness on larger monitors because the same numbers of pixels are being spread out over a larger number of **inches**. Display resolution is not measured in dots per inch as it usually is with printers ***(*****We measure resolution in pixels per inch or more commonly, dots per inch (dpi)).**

In computers, resolution is the number of pixels (individual points of color) contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis.

The sharpness of the image on a display depends on the resolution and the size of the monitor.

Display resolution is not measured in dots per inch as it usually is with printers *(We measure resolution in pixels per inch or more commonly, dots per inch (dpi)).*

- A display with 240 pixel columns and 320 pixel rows would generally be said to have a resolution of **240×320.**
- Resolution can also be used to refer to the total number of pixels in a digital camera image. For example, a camera that can create images of 1600x1200 pixels will sometimes be referred to as a 2 megapixel resolution camera since 1600 x 1200 = 1,920,000 pixels, or roughly 2 million pixels. Where a megapixel (that is, a million pixels) is a unit of image sensing capacity in a digital camera. In general, the more megapixels in a camera, the better the resolution when printing an image in a given size.

Below is an illustration of how the same image might appear at different pixel resolutions, if the pixels were poorly rendered as sharp squares (normally, a smooth image reconstruction from pixels would be preferred, but for illustration of pixels, the sharp squares make the point better).

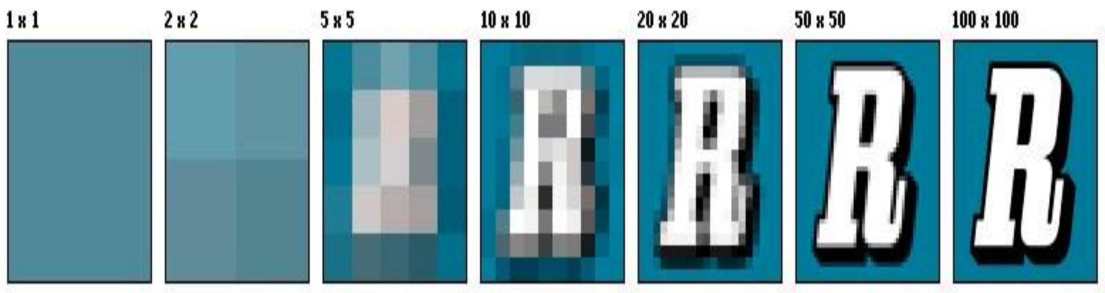


Figure (1.6) : Image Resolution.

An image that is 2048 pixels in width and 1536 pixels in height has a total of 2048×1536 = 3,145,728 pixels or 3.1 megapixels. One could refer to it as 2048 by 1536 or a 3.1-megapixel image.

1.8. Image Representation

We have seen that the **H**uman **V**isual **S**ystem (HVS) receives an input image as a collection of spatially distributed light energy; this is form is called an optical image. Optical images are the type we deal with every day –cameras captures them, monitors display them, and we see them [we know that these optical images are represented as video information in the form of analog electrical signals and have seen how these are sampled to generate the digital image I(r , c).

The digital image I (r, c) is represented as a two- dimensional array of data, where each pixel value corresponds to the brightness of the image at the point (r, c). in linear algebra terms , a two-dimensional array like our image model I(r, c) is referred to as a matrix , and one row (or column) is called a vector.

The image types we will consider are:

***1.8.1. Binary* Image**

Binary images are the simplest type of images and can take on two values, typically black and white, or ‘0’ and ‘1’. A binary image is referred to as a 1 bit/pixel image because it takes only 1 binary digit to represent each pixel.

These types of images are most frequently in computer vision application where the only information required for the task is general shapes, or outlines information. For example, to position a robotics gripper to grasp (يمسك) an object or in optical character recognition (OCR).

Binary images are often created from gray-scale images via a threshold value is, those values above it are turned white (‘1’), and those below it are turned black (‘0’).

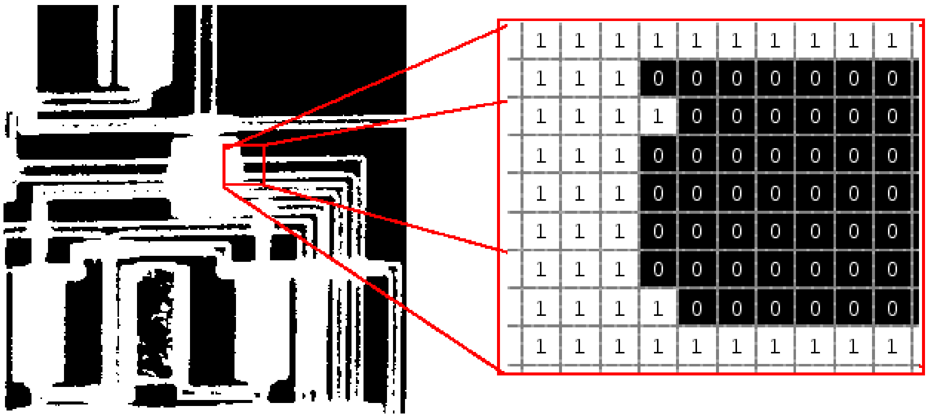


Figure (1.7): Binary Images.

1.8.2.Gray-scale images.

Gray-scale image is a range of monochromatic shades from black to white. Therefore, a grayscale image contains only shades of gray (brightness information only) and no color information. The number of different brightness level available. (0) value refers to black color, (255) value refers to white color, and all intermediate values are different shades of gray varying from black to white. The typical image contains 8 bit/ pixel (data, which allows us to have (0-255) different brightness (gray) levels. The 8 bit representation is typically due to the fact that the byte, which corresponds to 8-bit of data, is the standard small unit in the world of digital computer.

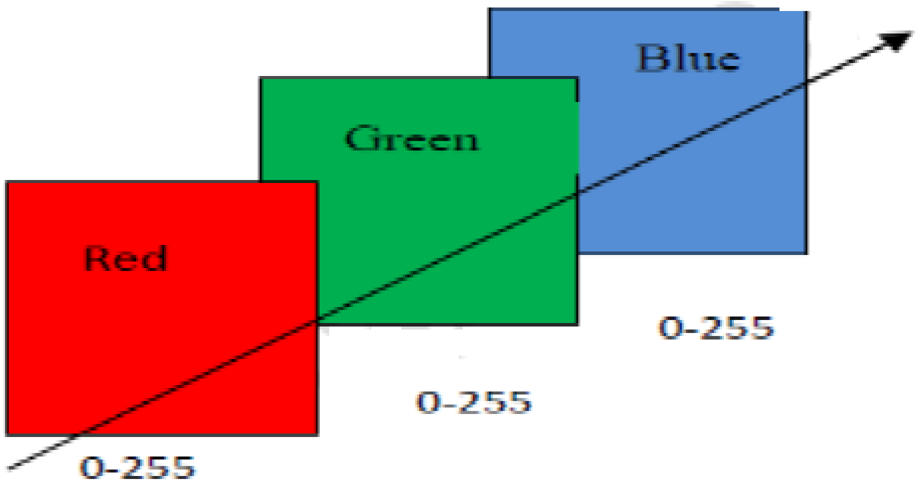
Figure (1.8) : Gray Scale Image

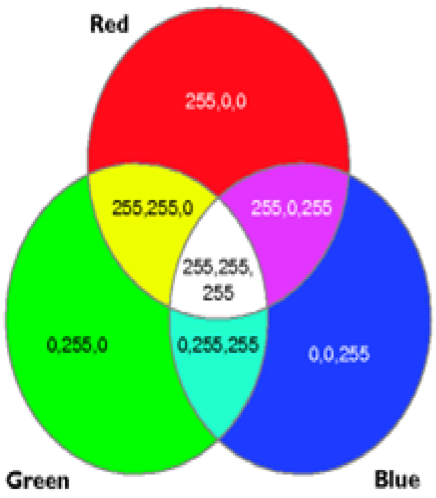
1.8.3.Color image.

Color image can be modeled as three band monochrome image data, where each band of the data corresponds to a different color. The actual information stored in the digital image data is brightness information in each spectral band. When the image is displayed, the corresponding brightness information is displayed on the screen by picture elements that emit light energy corresponding to that particular color. Typical color images are represented as red, green, and blue or RGB images. Using the 8-bit monochrome standard as a model, the corresponding color image would have 24 bit/pixel – 8 bit for each color bands (red, green and blue). The following figure we see a representation of a typical RGB color image.

IR(r,c) IG(r,c) IB(r,c)

The following figure illustrate that in addition to referring to arrow or column as a vector, we can refer to a single pixel red ,green, and blue values as a color pixel vector –(R,G,B).





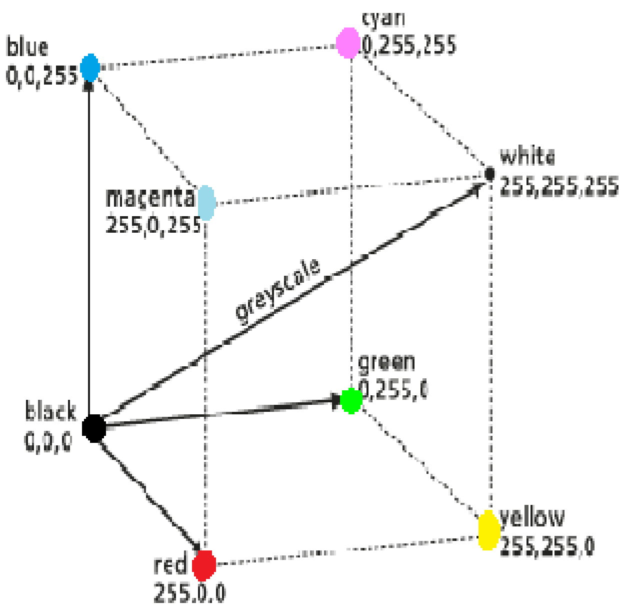


Figure (1.8): a color pixel vector consists of the red, green and blue pixel values (R, G, B) at one given row/column pixel coordinate (r , c).

The RGB color model used in color CRT monitor, in this model ,Red,Grren and Blue are added together to get the resultant color White.

For many applications, RGB color information is transformed into mathematical space that decouples the brightness information from the color information.

The lightness is the brightness of the color, and the hue is what we normally think of as “color” and the hue (ex: green, blue, red, and orange). The saturation is a measure of how much white is in the color (ex: Pink is red with more white, so it is less saturated than a pure red). Most people relate to this method for describing color.

Example: “a deep, bright orange” would have a large intensity (“bright”), a hue of “orange”, and a high value of saturation (“deep”).we can picture this color in our minds, but if we defined this color in terms of its RGB components, R=245, G=110 and B=20, most people have no idea how this color appears. Modeling the color information creates a more people oriented way of describing the colors.

1.9. Multispectral images.

Multispectral images typically contain information outside the normal human perceptual range. This may include infrared (تحت الحمراء),ultraviolet (فوق البنفسجية), X-ray, acoustic or radar data. These are not images because the information represented is not directly visible by the human system. Source of these types of image include satellite systems, underwater sonar systems and medical diagnostics imaging systems.

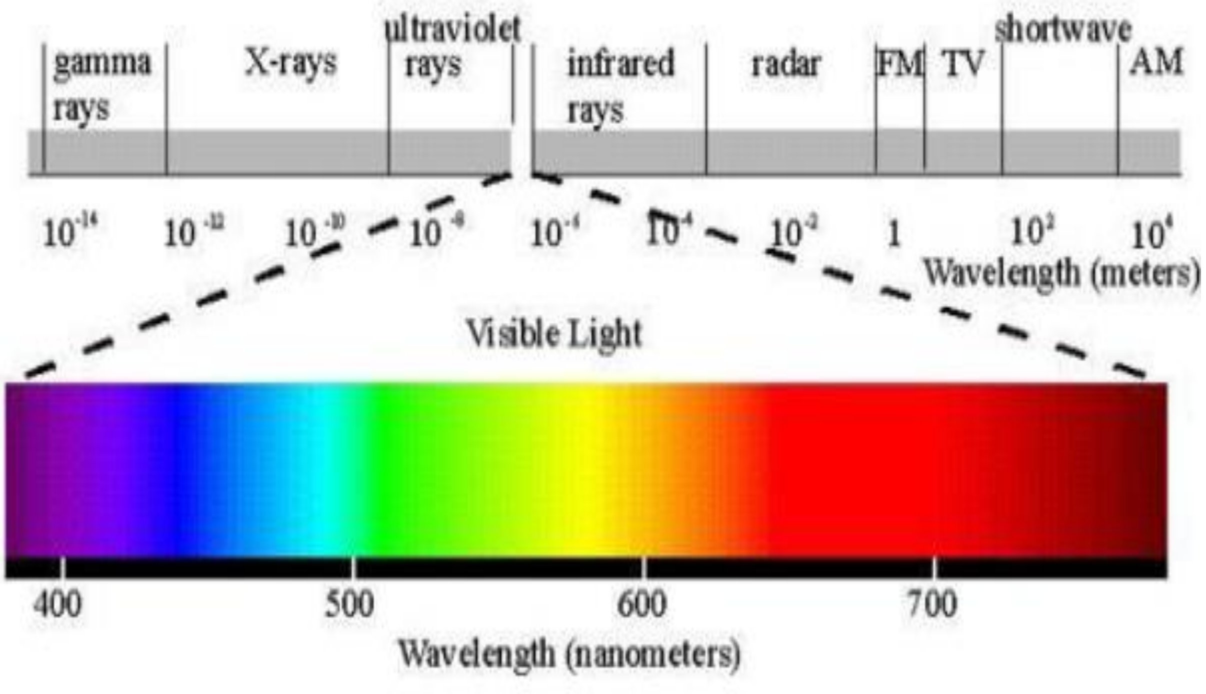


Figure (1.9) Electromagnetic spectrum.

1.10. Digital Image File Format

Why do we need so many different types of image file format?

• *The short answer* is that there are many different types of images and application with varying requirements.

• *A more complete answer*, also considers market share proprietary information, and a lack of coordination within the imaging industry. Many image types can be converted to one of other type by easily available image conversion software. Field related to computer imaging is that computer graphics.

1.12.1 Computer Graphics:

Computer graphics is a specialized field within that refers to the computer science realm that refers to the reproduction of visual data through the use of computer. In computer graphics, types of image data are divided into two primarily categories:

1. *Bitmap Image* (or Raster Image): can represented by our image model I(r, c), where we have pixel data and corresponding brightness values stored in file format.

2. *Vector Images*: refer to the methods of representing lines, curves shapes by storing only the key points. These key points are sufficient to define the shapes, and the process of turning theses into an image is called rendering. After the image has been rendered, it can be thought of as being in bitmap format where each pixel has specific values associated with it.

The differences between vector and raster graphics are:

1- Raster graphics are composed of pixels, while vector graphics are composed of paths.

2- A raster graphic, such as a gif or jpeg, is an array of pixels of various colors, which together form an image.
A vector graphic, such as an .eps file or Adobe Illustrator file, is composed of paths, or lines, that are either straight or curved.

3- Because vector graphics are not made of pixels, the images can be scaled to be very large without losing quality. Raster graphics, on the other hand, become "blocky," since each pixel increases in size as the image is made larger.

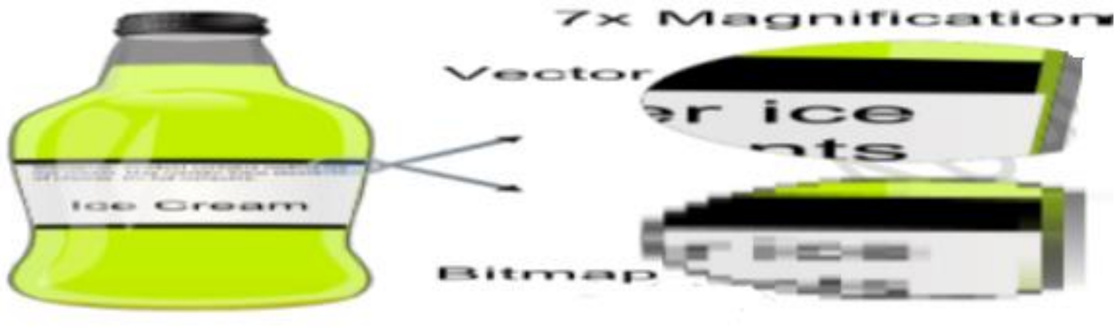


Figure (1.10): vector and bitmap image.

Image file formats are standardized means of organizing and storing digital images. Image files are composed of digital data in one of these formats that can be rasterized (نقطية) for use on a computer display or printer. An image file format may store data in uncompressed, compressed, or vector formats. Once rasterized, an image becomes a grid of pixels, each of which has a number of bits to designate its color equal to the color depth of the device displaying it.

Many image types can be converted to one of other type by easily available image conversion software. Field related to computer imaging is that computer graphics.

The most the type of file format falls into category of bitmap images. In general, these types of images contain both header information and the raw pixel data. The header information contains information regarding:

1. The number of rows (height)
2. The number of columns (Width)
3. The number of bands.
4. The number of bit per pixel.
5. The file type.
6. Additionally, with some of the more complex file formats, the header may contain information about the type of compression used and other necessary parameters to create the image, I(r, c).

<u>1.12.2. Image File Format:</u>

1. BMP format:

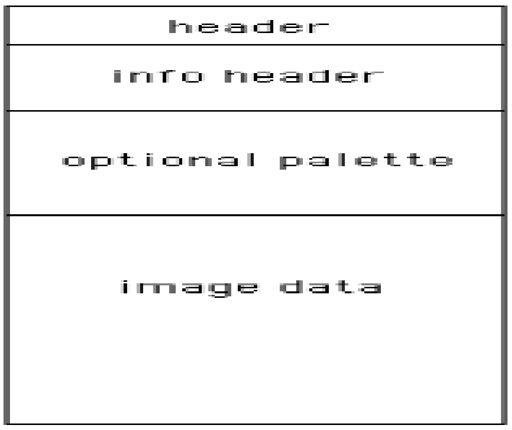
It's a compressed format and the data of image are located in the field of data while there are two fields one for header (54 byte) that contains the image information such as (height ,width , no. of bits per pixel, no of bands , the file type). The second field is the color map or color palette for gray level image, where its length is 0-255).

The **BMP file format**, also known as **bitmap image file** or **device independent bitmap (DIB) file format** or simply a **bitmap**, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems.

The BMP file format is capable of storing 2D digital images of arbitrary width, height, and resolution, both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles. BMP files are an historic (but still commonly used) file format for the operating system called "Windows". BMP images can range from black and white (1 bit per pixel) up to 24 bit colour (16.7 million colours). While the images can be compressed, this is rarely used in practice and won't be discussed in detail here.

Structure

A BMP file consists of either 3 or 4 parts as shown in the following diagram



The first part is a header, this is followed by an information section, if the image is indexed color then the palette follows, and last of all is the pixel data. Information such as the image width and height, the type of compression, the number of colors is contained in the information header.

Header

The header consists of the following fields. Note that we are assuming short int of 2 bytes, int of 4 bytes, and long int of 8 bytes.

1. Magic identifier (1+1= 2 Byte).
2. File Size in byte(4 Byte)
3. Resereved1+ Reserved2 (2+2 Byte)
4. Offset to image data (4 Byte)

14 byte

Information

The image info data that follows is 40 bytes in length, structure given below. The fields of most interest below are the image width and height, the number of bits per pixel (should be 1, 4, 8 or 24), the number of planes (assumed to be 1 here), and the compression type (assumed to be 0 here).

1. Header size in bytes (4 Byte)
2. Width and height of image (4 + 4 Byte)
3. Number of colour planes (2 Byte)
4. Bits per pixel (2 Byte)
5. Compression type (4 Byte)
6. Image size in bytes (4 Byte)
7. Pixels per meter (x_ resolution , y_ resolution) (4 + 4 Byte)
8. Number of colours (4 Byte)
9. Important colours (4 Byte)

40 bytes

The compression types supported by BMP are listed below:

0 - no compression

1 - 8 bit run length encoding

2 - 4 bit run length encoding

3 - RGB bitmap

2. TIFF (Tagged Image File Format) and GIF (Graphics Interchange Format):

It is one of the most popular and flexible of the current public domain raster file formats. They are used on World Wide Web (WWW). GIF files are limited to a maximum of 8 bits/pixel and allows for a type of compression called LZW. The GIF image header is 13 byte long & contains basic information.

3. JPEG (Joint photo Graphic Experts Group):

This is the right format for those photo images which must be very small files, for example, for web sites or for email. JPG is often used on digital camera memory cards. The JPG file is wonderfully small, often compressed to perhaps only 1/10 of the size of the original data, which is a good thing when modems are involved. However, this fantastic compression efficiency comes with a high price. JPG uses lossy compression (lossy meaning "with losses to quality"). Lossy means that some image quality is lost when the JPG data is compressed and saved, and this quality can never be recovered. JPEG images compression is being used extensively on the WWW. It’s, flexible, so it can create large files with excellent image equality.

4. VIP (visualization in image processing) formats:

It is developed for the CVIP tools software, when performing temporary images are created that use floating point representation, which is beyond the standard 8-bit/pixel. To represent this type of data the remapping is used, which is the process of taking original image and adding an equation to translate it to the range (0-225).

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 equations: 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 is used both computer vision and image processing.

For computer vision, the 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:

Is used to remove noise and eliminate irrelevant, visually unnecessary information. Preprocessing steps might make 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.
- Finding regions of interest for further processing.

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.

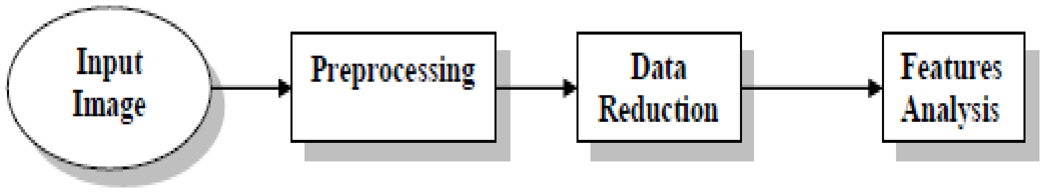
2. **Data Reduction:**

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.

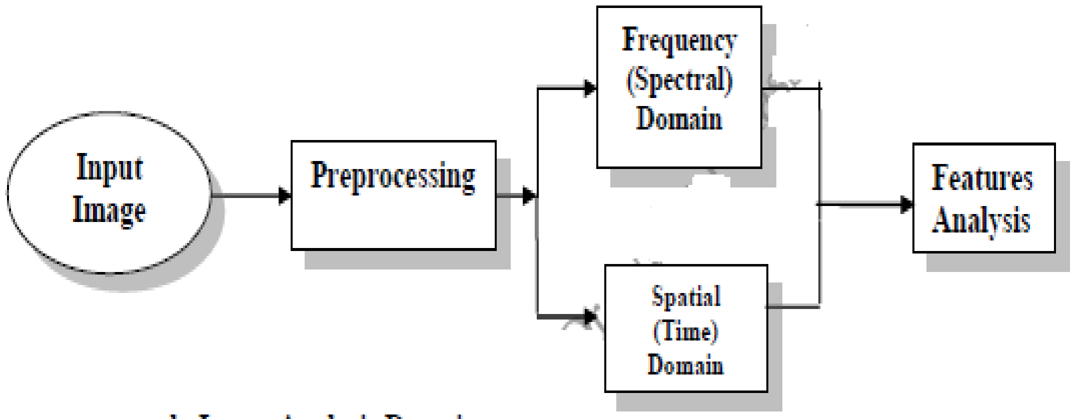
3. **Features Analysis:**

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.



a. Image Analysis



b. Image Analysis Domains

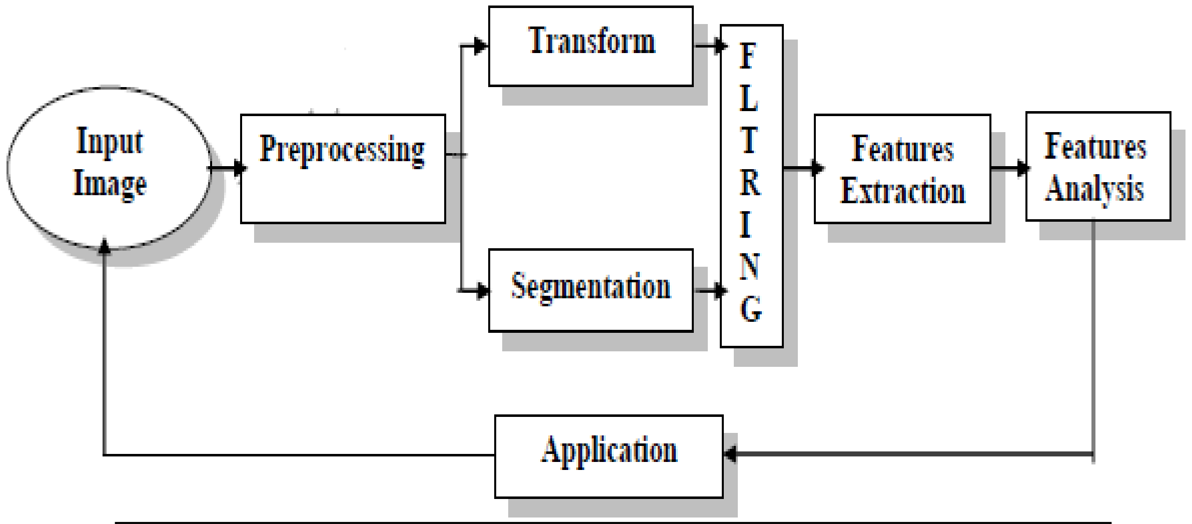


Figure (2.1): System Model of Image Analysis

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:

Original Image Array

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

Image After Applying Zero-Order-Hold

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

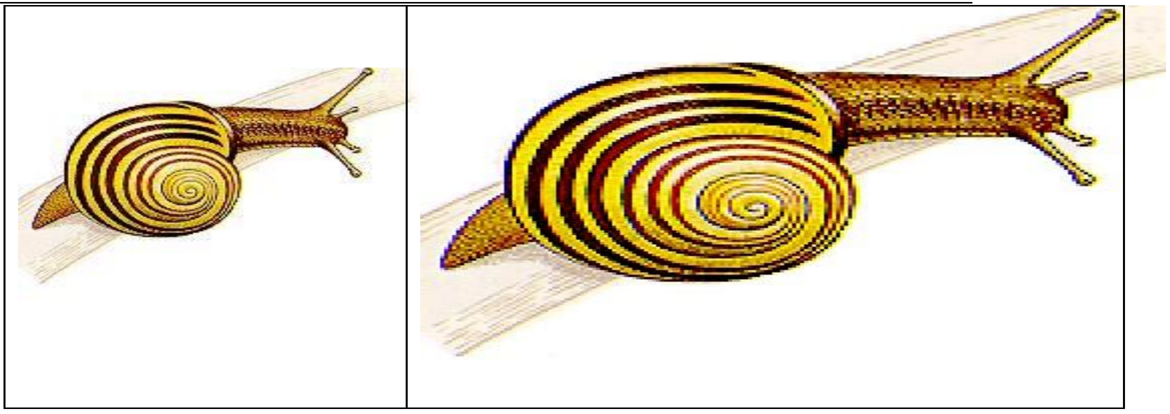


Figure (2.2): Zero _Order Hold Method

2. ***First _Order Hold*: ** is performed by finding linear interpolation between an adjacent pixel, 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.

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

Image with rows and columns expanded

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

Example: Enlarge the following sub image using:

1- First –Order- Hold.

2- Zero-Order _Hold.

$$\begin{bmatrix} 150 & 50 & 30 \\ 40 & 100 & 80 \\ 120 & 60 & 60 \end{bmatrix}$$

Solution:

a. First-Order Hold.

Size of original image is: 3x3

Size of the resultant image is: 5x5

(150+50)/2=100 (50+30)/2=40 (40+100)/2=70

(100+80)/2=90 (120+60)/2=90 (60+60)/2=60

Image with row expanded

$$\begin{bmatrix} 150 & 100 & 50 & 40 & 30 \\ 40 & 70 & 100 & 90 & 80 \\ 120 & 90 & 60 & 60 & 60 \end{bmatrix}$$

(150+40)/2=95 (100+70)/2=85 (50+100)/2=75

(40+90)/2=66 (30 + 80)/2=55 (40 +120)/2=80

(70 + 90)/2=80 (100 + 60)/2=80 (90 + 60)/2=75 (80 +60)/2=70

Image with row and column expanded

$$\begin{bmatrix} 150 & 100 & 50 & 40 & 30 \\ 95 & 85 & 75 & 66 & 55 \\ 40 & 70 & 100 & 90 & 80 \\ 80 & 80 & 80 & 75 & 70 \\ 120 & 90 & 60 & 60 & 60 \end{bmatrix}$$

b. Zero-Order Hold

The original image size is 3x3.

After enlargement, the size will be 6x6

$$\begin{bmatrix} 150 & 150 & 50 & 50 & 30 & 30 \\ 150 & 150 & 50 & 50 & 30 & 30 \\ 40 & 40 & 100 & 100 & 80 & 80 \\ 40 & 40 & 100 & 100 & 80 & 80 \\ 120 & 120 & 60 & 60 & 60 & 60 \\ 120 & 120 & 60 & 60 & 60 & 60 \end{bmatrix}$$

***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

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

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 on 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***.

In this process, 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.

If we call the convolution mask M (r, c) and the image I (r, c), the convolution equation is given by:

$$\sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} I(r-x,c-y)M(x,y)$$

<u>The steps of the convolution process can be summarized as follows:</u>

<u>a.</u> 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).

<u>b.</u> 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.

<u>c.</u> 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).

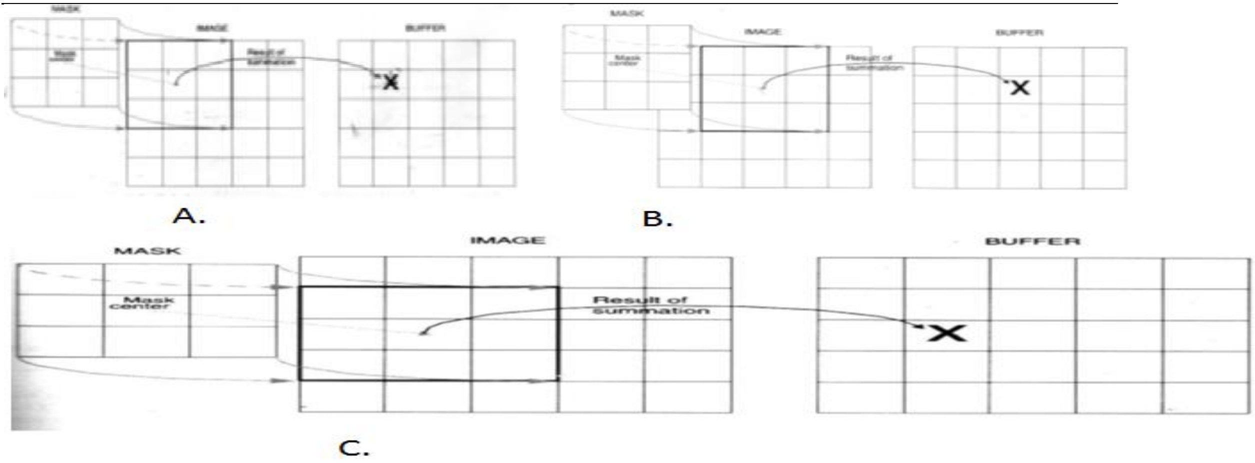


Figure (2.2. a,b and c) the convolution process.

These methods will only allow 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: Enlarge the following sub image three times its original size using k method.

$$\begin{bmatrix} 130 & 160 & 190 \\ 160 & 190 & 220 \end{bmatrix}$$

Solution:

The size of the result image will be k(N-1) +1, where k=3

So the sub image with size 2x3 will be

3(2-1) +1=4 row and 3(3-1) +1=7 column (4 x 7)

For column:

1: 130-160=30 K=3 then 30/3=10

k-1=3-1=2 no. of column to be inserted

130+10*1=140, 130+10*2=150

2: 160-190=30/3=10 160+10 *1=170, 160+10*2=180

3: 160-190=30/3=10 160+10 *1=170, 160+10*2=180

4: 190-220=30/3=10 190+10 *1=200, 190+10*2=210

The same thing repeated for the row. The resultant image is:

$$\begin{bmatrix} \boxed{130} & 140 & 150 & \boxed{160} & 170 & 180 & \boxed{190} \\ 140 & 150 & 160 & 170 & 180 & 190 & 200 \\ 150 & 160 & 170 & 180 & 190 & 200 & 210 \\ \boxed{160} & 170 & 180 & \boxed{190} & 200 & 210 & \boxed{220} \end{bmatrix}$$

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 4.

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).

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

Averaging	**median**	**decimation**
6+8+1+9=24/4=6	$1,\underline{6,8},9 = (6+8)/2=7$	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' = $\begin{bmatrix} 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \end{bmatrix}$ $Row\ wisng\ e\ zooming$

I''= $\begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix}$ $column\ wisng\ e\ zooming$

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 ------→ 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	**25**	**20**	15
30	**25**	**20**	15	20	25	30

Column wise zooming:

The same procedure has to be performed column wise. The procedure includes 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

2.4. Image Algebra

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

1. **Arithmetic operations. (**Addition, subtraction, division and multiplications)
2. **Logic operations. (**AND, OR and NOT)

These operations which require only one image, and are done on a pixel–by-pixel basis.

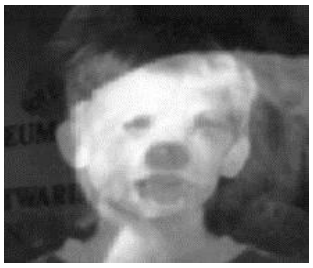
To apply the arithmetic operations on two images, we simply operate on corresponding pixel values, which means that the value of a pixel in the output image depends only on the values of the corresponding pixels in the input images. Hence, the images normally have to be of the same size. For example,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

$$\underset{I_1}{\begin{pmatrix} 3 & 4 & 7 \\ 3 & 4 & 5 \\ 2 & 4 & 6 \end{pmatrix}} + \underset{I_2}{\begin{pmatrix} 6 & 6 & 6 \\ 4 & 2 & 6 \\ 3 & 5 & 5 \end{pmatrix}} = \underset{I_3}{\begin{pmatrix} 3+6 & 4+6 & 7+6 \\ 3+4 & 4+2 & 5+6 \\ 2+3 & 4+5 & 6+5 \end{pmatrix}} = \begin{pmatrix} 9 & 10 & 13 \\ 7 & 6 & 11 \\ 5 & 9 & 11 \end{pmatrix}$$

- **<u>Addition</u>** 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.
- **<u>Subtraction</u>** 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

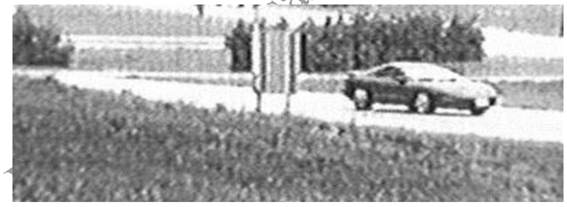
nonzero result at the location of movement. Subtraction process also used to detect the defects in the images. Applications include Object tracking, Medical imaging. Subtraction can result in a negative value for certain pixels. When this occurs with unsigned data types, such as uint8 or uint16, the subtract function truncates the negative value to zero (0), which displays as black.

- ***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 on will brightness the image**. Multiplication process often used for masking operations, see figure (2.7).



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

a. Original scene


b. Same scene later

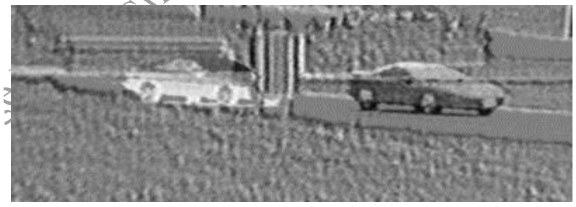

c. Subtraction of scene a from scene b

Figure (2.5): Image Addition, Image Subtraction.

a. Cameraman image

b.X-ray image of hand

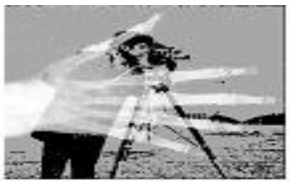
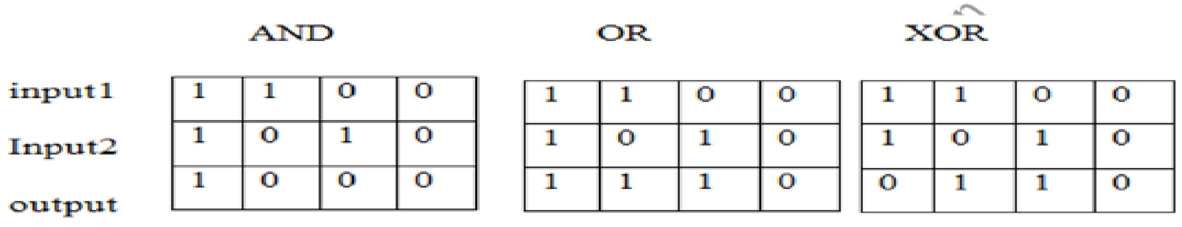

c. Multiplication of two images

Figure (2.6) Image Multiplication.

Figure (2.7) Image Division.

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 variablesis1only when both variables are 1, the result at any location in a resulting AND image is1 only if the corresponding pixels in the two input images are 1. As logical operation involves 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, whichis1, when one or the other pixel is1, 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.

Example: A white square ANDed with an image will allow only the portion of the image coincident with the square to appear in the output image with the background turned black; and a black square ORd with an image will allow only the part of the image corresponding to the black square to appear in the output image but will turn the rest of the image white. This process is called ***image masking*** as shown in figure (2.8).

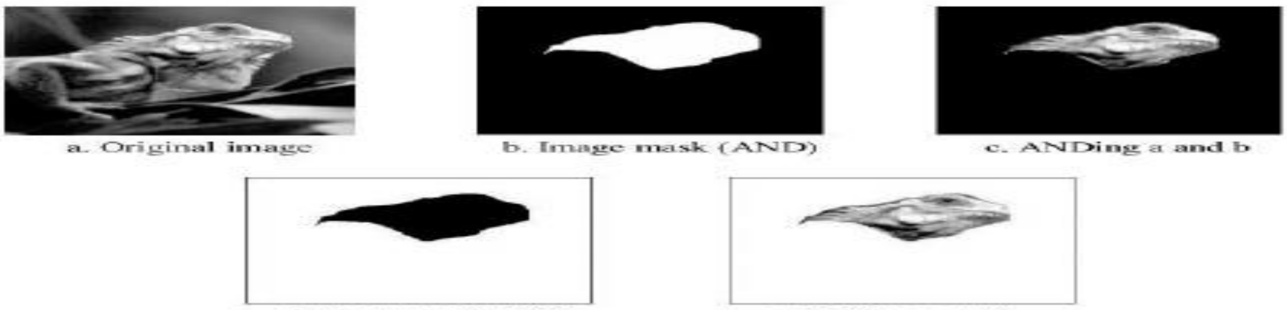


Figure (2.8) Image Masking.

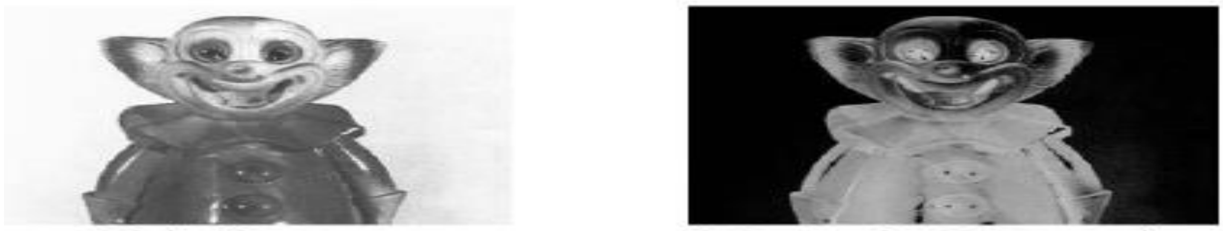


Figure (2.9) Complement Image.

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

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) ^ b (x, y)

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:

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

- **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.

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

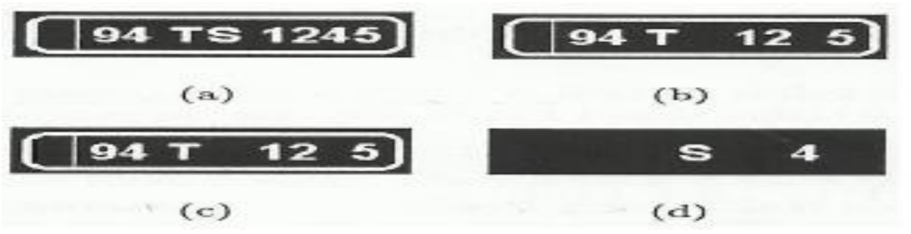


Figure 2.10: a) input image1 $a(x,y)$
b) input image2 $b(x,y)$
c) $a(x,y) \wedge b(x,y)$
d) $a(x,y) \otimes b(x,y)$

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

$$\left.\begin{array}{lcr} \mathbf{(111)_{10}} & \longrightarrow & \mathbf{01101111_2} \\ \mathbf{(88)_{10}} & \longrightarrow & \mathbf{01011000_2} \end{array}\right] \textbf{AND}$$

$$\overline{\mathbf{01001000}}$$

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

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

2.5 <u>Image Restoration:</u> 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 <u>Noise Definition</u>

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 (ظاهره) 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.

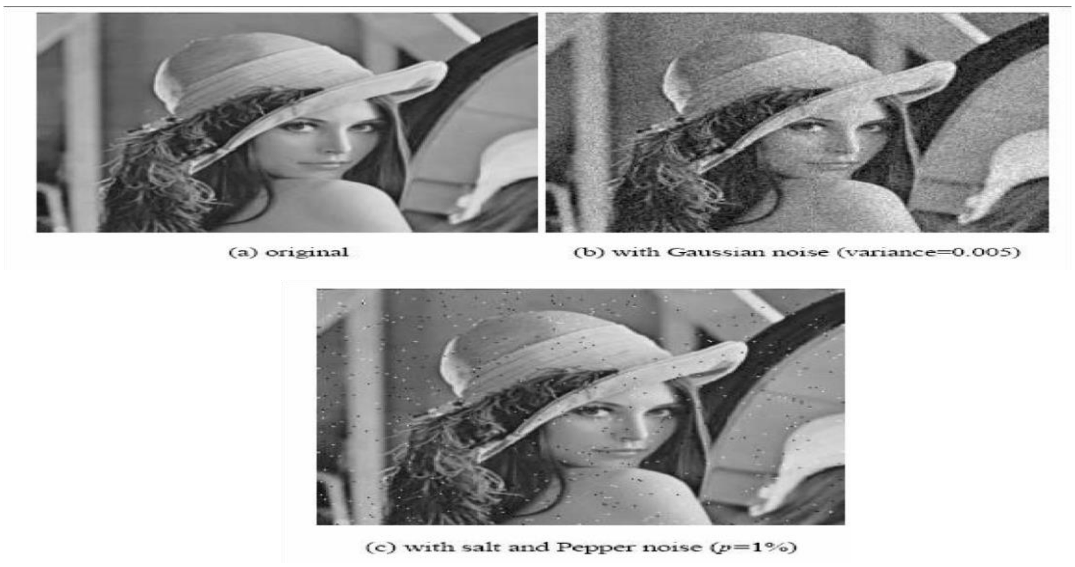

(a) original
(b) with Gaussian noise (variance=0.005)
(c) with salt and Pepper noise (p=1%)

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 high lights edge 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 neighbors, 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 (2.11): Mean Filter.

The median filter is a nonlinear 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 center pixel value (similar to the convolution process). **Order statistics** is a technique that arranges the entire pixel in sequential order, given an N×N window (W) the pixel values can be ordered from smallest to the largest.

$\mathbf{I_1 \le I_2 \le I_3 \ldots\ldots < I_N}$

Where $\mathbf{I_1, I_2, I_3 \ldots\ldots, I_N}$ are the intensity values of the subset of pixels in the image.

Example: Given the following 3×3 neighborhood.

5	**5**	**6**
3	**4**	**5**
3	**4**	**7**

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.

- **Note:** A median filter can use a neighborhood of any size, but 3*3, 5*5 and 7*7. 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 center 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.

Note: the outer rows and columns are not replaced. Moreover, 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 work 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.

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".

NOTE: <u>This effectively turns a gray_level image into a binary (two level) image</u>

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. **<u>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</u>**.

<u>Example</u>:

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 ÷ 8= 32**

The general case requires us to mask k bits, where $2^{\mathbf{k}}$ 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($2^{\mathbf{1}}$).
- 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 meth~~o~~d.

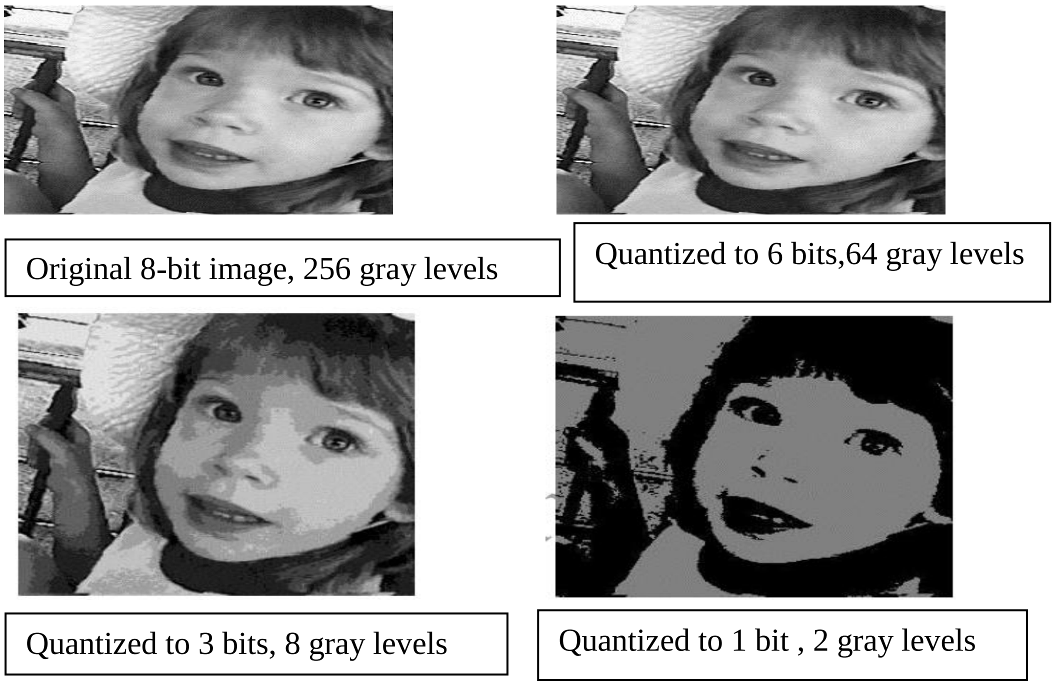

Original 8-bit image, 256 gray levels
Quantized to 6 bits,64 gray levels
Quantized to 3 bits, 8 gray levels
Quantized to 1 bit , 2 gray levels

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.

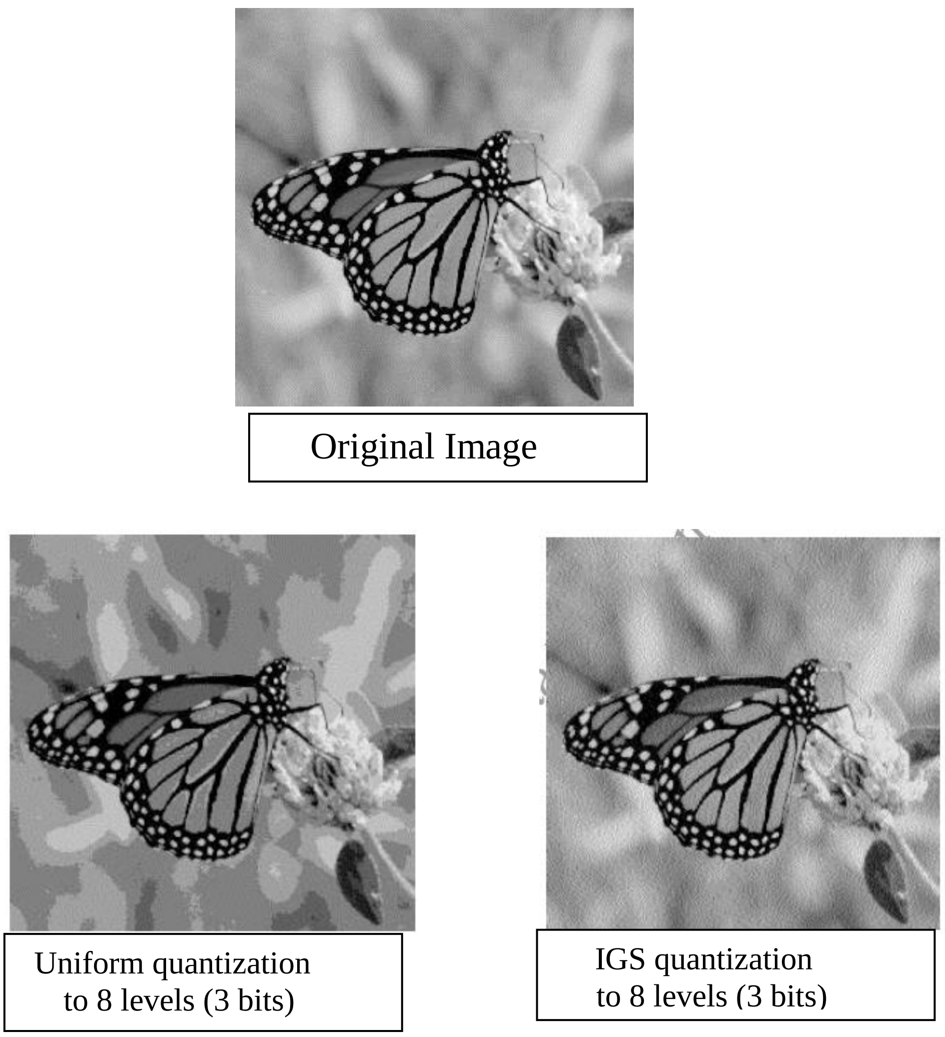

Original Image
Uniform quantization to 8 levels (3 bits)
IGS quantization to 8 levels (3 bits)

Figure (2-18): IGS quantization

2.6. 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.

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.

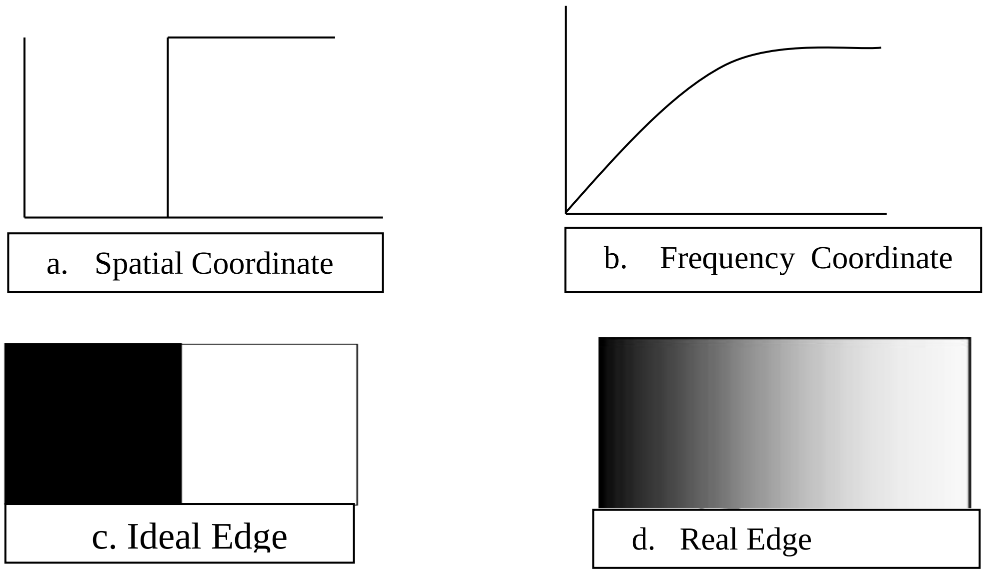


Figure (2-19): Ideal vs. Real Edge.

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.

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 form the row mask and **S2** from the column mask.
- Use S1, S2 to compute two matrices,

- **<u>the edge magnitude and the edge direction (angle of orientation of the edge),</u>** which are defined as follows:

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

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

Example: determine the edges of the following sub image using <u>Sobel edge detector.</u>

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

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:

S1= (2+8+1) – (1+4+2) = 4, **S2=(2+0+1) - (1+2+2) = -2,**

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

EM(2,1)= **EM(2,2)=** **…….**

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

Row Mask

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

Column Mask

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

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:

Edge Magnitude = $\sqrt{P_1{}^2 + P_2{}^2}$ **Edge Direction** = Tan $^{-1}$ $\left[\frac{P_1}{P_2}\right]$

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

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

$$\begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix} \qquad \begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix} \qquad \begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

K1 **K2** **K3**

$$\begin{matrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{matrix} \qquad \begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix} \qquad \begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix}$$

K4 **K5** **K6**

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix} \qquad \begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}$$

K7 **K8**

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 lef~~t~~, 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:

$$\begin{matrix} -1 & 0 & 5 \\ -2 & 0 & 5 \\ -1 & 0 & 5 \end{matrix} \qquad \begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix} \qquad \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

R1 **R2** **R3**

$$\begin{matrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{matrix} \qquad \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \qquad \begin{matrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{matrix}$$

R4 **R5** **R6**

$$\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \qquad \begin{matrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{matrix}$$

R7 **R8**

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 R_0 and $R_7$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.

<u>5- Laplacian Operators:</u> 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 center 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 center 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×n area (where n s 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 center pixel of an n×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. However, it is quicker than the homogeneity operator since it uses four integer subtractions as against eight subtractions in homogeneity operator per pixel.

<u>Example: Shown</u> below is how the two operators detect the edge:

Consider an image block with centre pixel intensity 5,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Output of *homogeneity operator* is:

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

Output of *difference operator* is:

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

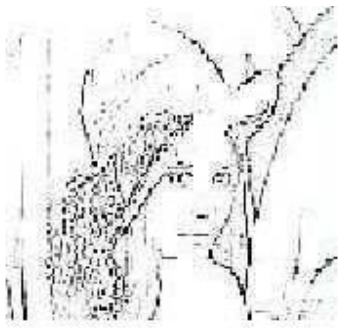
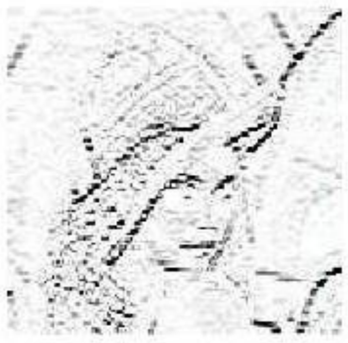
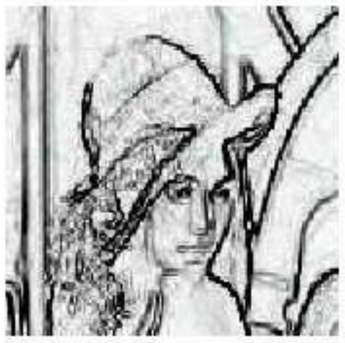
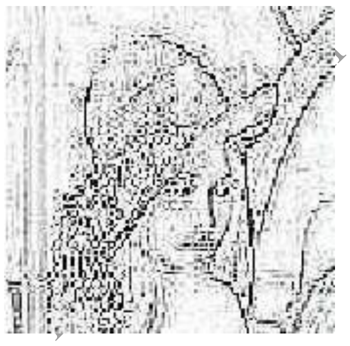
		
Original Image	**Edge detect by Sobel' horizontal mask**	**Edge detect by Sobel overall mask**
		
Edge detect by Gaussian mask	**Edge detect by Prewitt mask**	

Figure (2-20) : Example of Edge Operators

Chapter Threee: Image Restoration

3.1 Introduction

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.

It is assumed that the degradation model is known or can be estimated. The idea is to model the degradation process and then apply the inverse process to restore the original image.

3.2. Noise

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 (ظاهره) 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.

3.2.1.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.

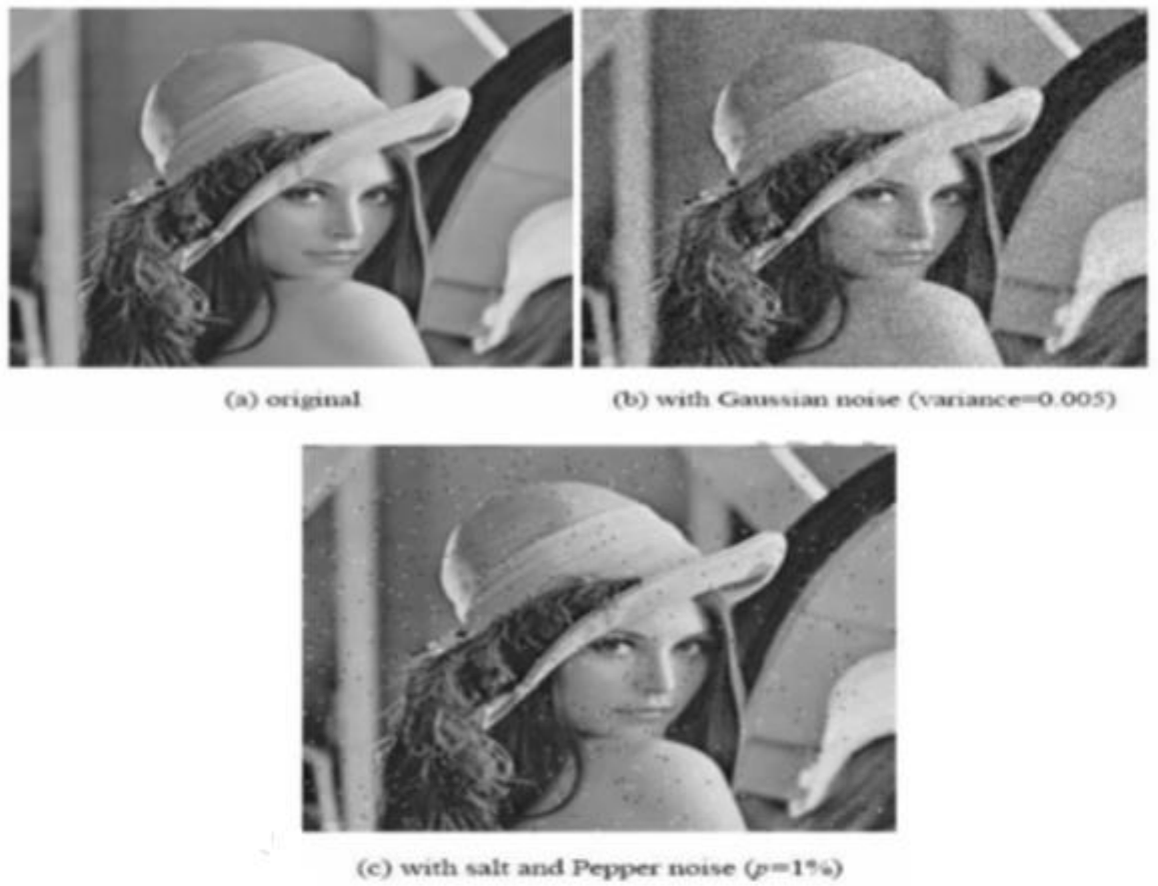
These filters are called spatial filter to distinguish them from frequency domain filter and typically operate on small neighborhood s, 3x3 to 11x11, and some can be implemented as convolution masks.

The types of filters are:

1. Mean filters.

2. Order filters (Median filer).

3. Enhancement filters.


(a) original (b) with Gaussian noise (variance=0.005)

(c) with salt and Pepper noise (p=1%)

Figure (3.1) Image noise

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 filters high lights edges and details within the image. Many 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 neighbors, 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.

3.2.1.1. mean filters

They are essentially *averaging filter*s. They operate on local groups of pixels called neighborhoods and replace the centre pixel with an average of the pixels in this neighborhood. This replacement is done with a convolution mask such as the following 3X3 mask:

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

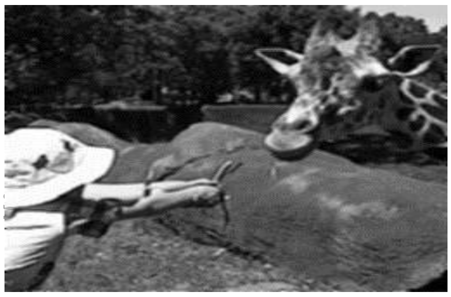
Arithmetic mean filter smoothing or low-pass filter.

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 smooth 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.

Mean filter ***works well with Gaussian or Uniform noise***, and has the disadvantage of blurring the image edge, or details.

a.

Original image



b. Mean filtered image

Figure (3.2): Mean Filter.

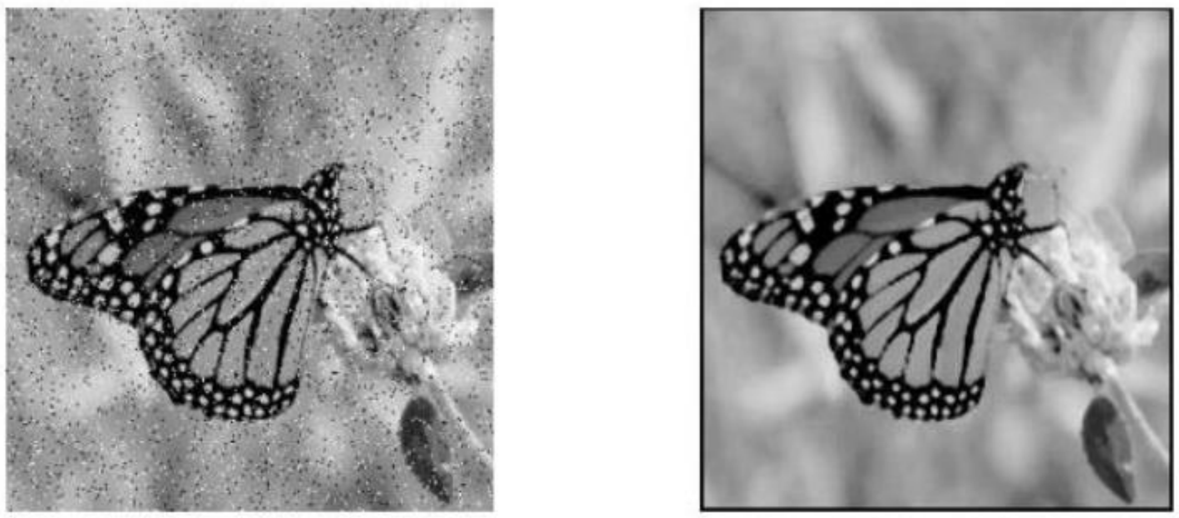
<u>3.2.1.2 median filter.</u>

It 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 center pixel value (similar to the convolution process). Order statistics is a technique that arranges the entire pixel in sequential order, given an NXN window (W) the pixel values can be ordered from smallest to the largest.

I1 ≤ I**2** ≤ I**3...………………….≤ IN2**

Where I1**, I2, I3...……, I N2** are the intensity values of the subset of pixels in the image.

Median filters are quite popular because, for certain types of random noise, they provide excellent noise reduction capabilities with considerably less blurring than linear smoothing filters of similar size. Median filter are particularly effective in the presence of its appearance as white and black dots super imposed on an image, i.e. median filter **works best with Salt and Pepper noise.**



a. Salt and pepper noise   b. Median filtered image (3x3)

Figure (3.2) Median filter

***Example* 3.1**

Given the following 3x3 sub image:

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

We first sort the value in order of size as (3,3,4,4,5,5,5,6,7); the we select the middle value, in this case it is **5** because there are four values above it and four values below it . This 5 is then placed in centre location. A median filter can use a neighborhood of any size, but 3*3, 5*5 and 7*7 are typical.

The median filtering operation is performed on an image by applying the sliding window concepts, similar to what is done with convolution. The outer rows and columns of an image are not replaced. In practice this is usually not a problem due to the fact that the images are much larger than the masks. And these “wasted” rows and columns are often filled with zeros (or cropped off the image) as mentioned before.

<u>3.2.1.3The maximum filter:</u> It is a Nonlinear order filter and works by selects the **<u>largest value</u>** within an ordered window of pixels’ values and replaces the central pixel with the ***<u>lightest one</u>***.

The maximum filters work best for pepper-type noise (low values).

<u>3.2.1.4 The minimum filter:</u> It is a Nonlinear order filter and works by selects the smallest value within an ordered window of pixels' values and replaces the central pixel with the **<u>darkest one</u>** in the ordered window.

The minimum filters works best for salt- type noise (high values).

<u>3.2.1.5. The Enhancement filter</u>

The enhancement filters are:

1. Laplacian- type filters.
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:

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

The ***Laplacian- type filters*** will enhance details in **all directions** equally**.** ***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:

$$\begin{array}{cccc} \textbf{VERTICAL} & \textbf{HORIZANTAL} & \textbf{DIAGONAL1} & \textbf{DIAGONAL2} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \end{array}$$

Example: Apply the following on the following sub image:

$$\begin{bmatrix} 100 & 30 & 20 & 40 & 15 \\ 15 & 130 & 220 & 30 & 40 \\ 30 & 200 & 40 & 120 & 20 \\ 15 & 50 & 20 & 60 & 50 \end{bmatrix}$$

1- Minimum filter to remove salt-type noise.
Solution:
g(1,1)= $\underline{15}$ 20 30 30 40 100 130 200 220.
g(1,2)= $\underline{20}$ 30 30 40 40 120 130 200 220
g(1,3)= $\underline{15}$ 20 20 30 40 40 40 120 220
g(2,1)= $\underline{15}$ 15 20 30 40 50 130 200 220
g(2,2)= $\underline{20}$ 30 40 50 60 120 130 200 220
g(2,3)= $\underline{20}$ 20 30 40 40 50 60 120 220

The resultant sub image is:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 15 & 20 & 15 & 0 \\ 0 & 15 & 20 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2- Maximum filter to remove pepper-type noise.
Solution:
g(1,1)= 15 20 30 30 40 100 130 200 $\underline{220}$.
g(1,2)= 20 30 30 40 40 120 130 200 $\underline{220}$
g(1,3)= 15 20 20 30 40 40 40 120 $\underline{220}$
g(2,1)= 15 15 20 30 40 50 130 200 $\underline{220}$
g(2,2)= $\underline{20}$ 30 40 50 60 120 130 200 $\underline{220}$
g(2,3)= $\underline{20}$ 20 30 40 40 50 60 120 $\underline{220}$

The resultant sub image is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 220 & 220 & 220 & 0 \\ 0 & 220 & 220 & 220 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3- Median filter to remove salt and pepper-type noise.
Solution:

g(1,1)= 15 20 30 30 $\underline{40}$ 100 130 200 220.
g(1,2)= 20 30 30 40 $\underline{40}$ 120 130 200 220
g(1,3)= 15 20 20 30 $\underline{40}$ 40 40 120 220
g(2,1)= 15 15 20 30 $\underline{40}$ 50 130 200 220
g(2,2)= 20 30 40 50 $\underline{60}$ 120 130 200 220
g(2,3)= 20 20 30 40 $\underline{40}$ 50 60 120 220

The resultant sub image is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 40 & 40 & 40 & 0 \\ 0 & 40 & 60 & 40 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3- Midpoint filter to remove Gaussian or Uniform noise.
Solution:

g(1,1)= $\underline{15}$ 20 30 30 40 100 130 200 $\underline{220}$.$\longrightarrow$ (15+220)/2=117.5

g(1,2)= $\underline{20}$ 30 30 40 40 120 130 200 $\underline{220.}$$\longrightarrow$(20+220)/2=120

g(1,3)= $\underline{15}$ 20 20 30 40 40 40 120 $\underline{220.}$ $\longrightarrow$(15+220)/2=117.5

g(2,1)= $\underline{15}$ 15 20 30 40 50 130 200 $\underline{220.}$ $\longrightarrow$(15+220)/2=117.5

g(2,2)= $\underline{20}$ 30 40 50 60 120 130 200 $\underline{220.}$$\longrightarrow$(20+220)/2=120

g(2,3)= $\underline{20}$ 20 30 40 40 50 60 120 $\underline{220.}$ $\longrightarrow$(20+220)/2=120

The resultant sub image is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 117.5 & 120 & 117.5 & 0 \\ 0 & 117.5 & 120 & 120 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4- Mean filter

g(1,1)= 1/9(100+30+20+15+130+220+30+200+40)=87.2 ≅ 87
g(1,2)= 1/9(30+20+40+130+220+30+200+40+120)=92.2 ≅ 92
g(1,3)= 1/9(20+40+15+220+30+40+40+120+20)=60.5≅ 61
g(2,1)= 1/9(15+130+220+30+200+40+15+50+20)=80
g(2,2)= 1/9(130+220+30+200+40+120+50+20+60)=96.6≅ 97
g(2,3)= 1/9(220+30+40+40+120+20+20+60+50)=66.6≅ 67

The resultant sub image is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 87 & 92 & 61 & 0 \\ 0 & 80 & 97 & 67 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5- Laplacian filter with the following mask:

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

Solution:

g(1,1)= (100*0)+(30*-1)+(20*0)+(15*-1)+(130*5)+(220*-1)+(30*0)+(200*-1)+(40*0)=185

g(1,2)= (30*0)+(20*-1)+(40*0)+(130*-1)+(220*5)+(30*-1)+(200*0)+(40*-1)+(120*0)=880

=255

g(1,3)= (20*0)+(40*-1)+(15*0)+(220*-1)+(30*5)+(40*-1)+(40*0)+(120*-1)+(20*0)=-270 =0

g(2,1)= (15*0)+(130*-1)+(220*0)+(30*-1)+(200*5)+(40*-1)+(15*0)+(50*-1)+(20*0)=750=255

g(2,2)= (130*0)+(220*-1)+(30*0)+(200*-1)+(40*5)+(120*-1)+(50*0)+(20*-1)+(60*0)=-360=0

g(2,3)= (220*0)+(30*-1)+(40*0)+(40*-1)+(120*5)+(20*-1)+(20*0)+(60*-1)+(50*0)=450=255

The resultant sub image is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 185 & 255 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

6- Difference filter that enhance the image in the vertical direction:

Solution:

The mask will use is

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

g(1,1)= (100*0)+(30*1)+(20*0)+(15*0)+(130*1)+(220*0)+(30*0)+(200*-1)+(40*0)=-40=0

g(1,2)= (30*0)+(20*-1)+(40*0)+(130*-1)+(220*5)+(30*-1)+(200*0)+(40*-1)+(120*0)=880

=255
g(1,3)= (20*0)+(40*-1)+(15*0)+(220*-1)+(30*5)+(40*-1)+(40*0)+(120*-1)+(20*0)=-200=0
g(2,1)= (15*0)+(130*-1)+(220*0)+(30*-1)+(200*5)+(40*-1)+(15*0)+(50*-1)+(20*0)=-380=0
g(2,2)= (130*0)+(220*-1)+(30*0)+(200*-1)+(40*5)+(120*-1)+(50*0)+(20*-1)+(60*0)=700 =0
g(2,3)= (220*0)+(30*-1)+(40*0)+(40*-1)+(120*5)+(20*-1)+(20*0)+(61*-1)+(50*0)=499=255

The resultant sub image is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Chapter Four: Image Compression

4.1 Introduction

Image compression is one of the image processing fields, the growth in the multimedia market, and the advent of the World Wide Web, which makes the Internet easily accessible for everyone and the advances in video technology, including high-definition television, are creating a demand for new, better, and faster image compression algorithms. Compression algorithm development starts with applications to two-dimensional (2-D) still images because video and television signals consist of consecutive frames of 2-D image data.

What is image compression? *Image compression* involves reducing the size of image data files, while retaining necessary information. The reduced file is called the *compressed file* and is used to reconstruct the image, resulting in the *decompressed image.* The original image, before any compression is performed, is called the *uncom-pressed* image file. The ratio of the original, uncompressed image file and the com-pressed file is referred to as the *compression ratio.* The compression ratio is denoted by:

$$Compression\ Ration = \frac{Uncompressed\ File\ Size}{Compressed\ File\ Size} = \frac{SIZE_U}{SIZE_C}$$

It is often written as $Size_U : Size_C$

EXAMPLE (4-1):

The original image is 256 * 256 pixels, single-band (gray-scale), 8 bits per pixel. This file is 65,536 bytes (64k). After compression, the image file is 6,554 bytes.

The compression ratio is:

C = 65536/6554 = 9.999 = 10. This can also be written as 10:1.

This is called a "10 to 1 compression" or a "10 times compression," or it can be stated as "compressing the image to 1/10 its original size."

Another way to state the compression is to use the terminology of *bits per pixel.* For an *N* * *N* image

$$Bits\ per\ Pixel = \frac{Number\ of\ Bits}{Number\ of\ Pixels} = \frac{8(Number\ of\ Bytes)}{NxN}$$
$$= \frac{6554\ x8}{256\ x256} = \frac{52432}{65536} = 0.8$$

EXAMPLE (4-2):

Using the preceding example, with a compression ratio of 65,536/6,554 bytes, we want to express this as bits per pixel. This is done by:

- **First:** finding the number of pixels in the image: 256 x 256 = 65,536 pixels.

- **Second:** find the number of bits in the compressed image file:

(6,554 bytes) * (8 bits/byte) = 52,432 bits.

- **Third:** find the bits per pixel by taking the ratio: 52,432/ 65,536 = 0.8 bits/pixel.

The reduction in file size is necessary to meet:

1- The bandwidth requirements for many transmission systems.

2- The storage requirements in computer databases.

The amount of data required for digital images is enormous. *For example*, a single 512 x 512, 8-bit image requires 2,097,152 bits for storage. If we wanted to transmit this image over the World Wide Web, it would probably take minutes for transmission—too long for most people to wait.

EXAMPLE (4-3)

To transmit an RGB (color) 512 x 512, 24-bit (8 bits/pixel/color) image via modem at 28.8 kbaud (kilobits/second), it would take about

$$\frac{(512x512\ pixels)(24\ bis/pixel)}{(28.8x1024\ bits/second)} = 213\ seconds = 3.6\ minutes$$

EXAMPLE (4-4)

To transmit a digitized color 35mm slide scanned at 3,000 * 2,000 pixels, and 24 bits, at 28.8 kb and would take about:

$$\frac{(3000\text{x}2000 \text{ pixels})\ (24 \text{ bits/pixel})}{(28.8x1024\ bits/second)} = 4883\ seconds = 81\ minutes$$

Couple this result with transmitting multiple images, or motion images, and the necessity of image compression can be appreciated.

The key to a successful compression scheme comes with the second part of the definition—*retaining necessary information.* To understand this we must differentiate between data and information.

For digital images, *data* refers to the pixel gray-level values that correspond to the brightness of a pixel at a point in space. *Information is* an interpretation of the data in a meaningful way. Data are used to convey information, much like the way the alphabet is used to convey information via words. Information is an elusive concept; it can be application specific. For example, in a binary image that contains text only, the necessary information may only involve the text being readable, whereas for a medical image the necessary information may be every minute detail in the original image.

There are two primary types of image compression methods—those that preserve the data and those that allow some loss of data.

1- ***Lossless** methods*: because no data are lost, and the original image can be recreated exactly from the compressed data. For complex images these methods are limited to compressing the image file to about one half to one third its original size (2:1 to 3:1); often the achievable compression is much less. For simple images such as text-only images, lossless methods may achieve much higher compression.

2- ***Lossy methods***: because they allow a loss in the actual image data, so the original uncompressed image cannot be created *exactly* from the compressed file. For complex images these techniques can achieve compression ratios of 100 or 200 and still retain high-quality visual information. For simple images or lower-quality results, compression ratios as high as 100 to 200 can be attained.

Compression algorithms are developed by taking advantage of the redundancy that is inherent in image data. Three primary types of redundancy can be found in images:

1) coding.

2) interpixel,
3) psychovisual redundancy.

Table (6.1) show differences between lossy and loosless compression.

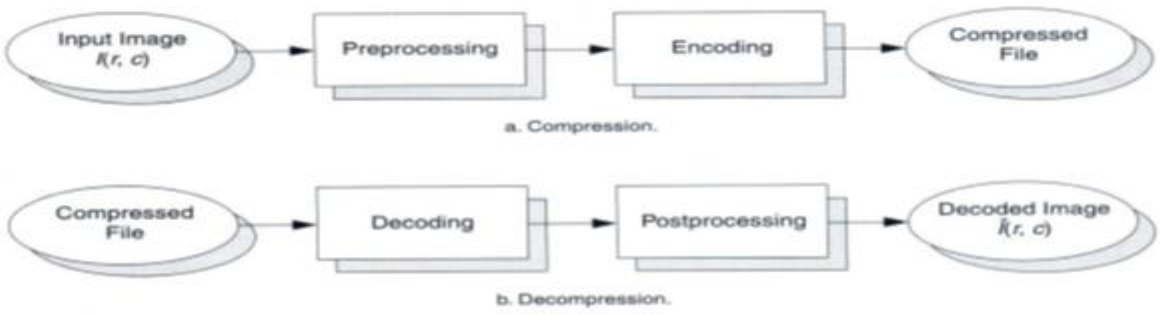
Lossless compression	***Lossy compression***
1. All original data can be recovered when the file is uncompressed every single bit of data that was originally in the file remains after the file is uncompressed.	**1.** Reduces a file by permanently eliminating certain information, especially redundant information
2. All of the information is completely restored	**2.** When the file is uncompressed, only a part of the original information is still there (although the user may not notice it)
3. This is generally the technique of choice for text or spreadsheet files, where losing words or financial data could pose a problem	**3.** Lossy compression is generally used for video and sound where a certain amount of information loss will not be detected by most users
4. The Graphics Interchange File (GIF) is an image format used on the Web that provides lossless compression.	**4.** The JPEG image file, commonly used for photographs and other complex still images on the Web, is an image that has lossy compression.

4.2 Compression System Model

The compression system model consists of two parts: the compressor and the decompressor.

- The ***compressor*** consists of a preprocessing stage and encoding stage.

- The ***decompressor*** consists of a decoding stage followed by a post processing stage as shown in (Figure 4.1).



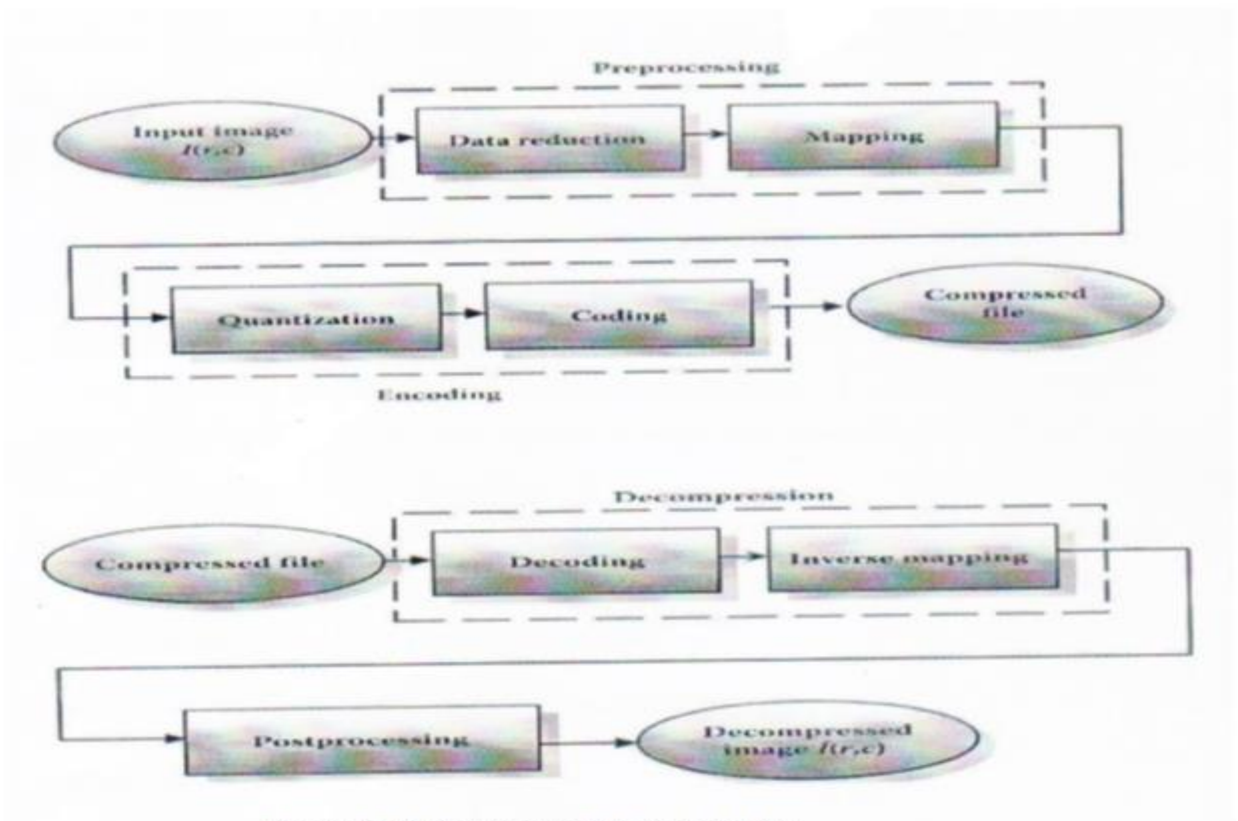


Figure (5.1) Compression System model

Before encoding, preprocessing is performed to prepare the image for the encoding process, and consists of any number of operations that are application specific. After the compressed file has been decoded, post processing can be performed to eliminate some of the potentially undesirable artifacts brought about by the compression process. Often, many practical compression algorithms are a combination of a number of different individual compression techniques.

4.3. LOSSLESS COMPRESSION METHODS

Lossless compression methods are necessary in some imaging applications. For example, with medical images, the law requires that any archived medical images are stored without any data loss. Many of the lossless techniques were developed for non-image data and, consequently, are not optimal for image compression. In general, the lossless techniques alone provide marginal compression of complex image data, often in the range of only a 10% reduction in file size. However, lossless compression techniques may be used for both preprocessing and post processing in image compression algorithms. Additionally, for simple images the lossless techniques can provide substantial compression.

An important concept here is the idea of measuring the average information in an image, referred to as the *entropy*. The entropy for *an N*N* image can be calculated by this equation:

$$Entropy = -\sum_{i=0}^{L-1} p_i \; log_2\left(p_i\right) \qquad \left(in \frac{bit}{pixel}\right)$$

Where pi = the probability of the $_i$th gray level= nk/N^2.

$_nk$ = the total number of pixels with gray value k

L = the total number of gray levels (e.g., 256 for 8 bits)

This measure provides us with a theoretical minimum for the average number of bits per pixel that could be used to encode the image. This number is theoretically optimal and can be used as a metric for judging the success of a coding scheme.

EXAMPLE (4-5):

Let *L* = *8,* meaning that there are 3 bits/pixel in the original image. Now, lets say that the number of pixels at each gray-level value is equal (they have the same probability), that is:

$Po = Pi = \ldots = P7 = 1/8$

Now, we can calculate the entropy as follows:

$$Entropy = -\sum_{i=0}^{L-1} p_i \, log_2\,(p_i) = -\sum_{i=0}^{7} \frac{1}{8} log_2\left(\frac{1}{8}\right) = 3$$

This tells us that the theoretical minimum for lossless coding for this image is 3 bits/pixel. In other words, there is no code that will provide better results than the one currently used (called the natural code, since 0002 = 0, 00 12 = 1, 0102 = 2, ..., 1112 = 7). This example illustrates that the image with the most random distribution of gray levels, a uniform distribution, has the highest entropy.

EXAMPLE (4-6):

Let *L* = 8, thus we have a natural code with 3 bits/pixel in the original image. Now let's say that the entire image has a gray level of 2, so

$p2=1, p0= p1= p3= p4= p5= p6=p7= 0$

And the entropy is

$$Entropy = -\sum_{i=0}^{L-1} p_i \, log_2\,(p_i) = -(1) log_2\,(1) + 0 + \cdots + 0 = 0$$

This tells us that the theoretical minimum for coding this image is 0 bits/pixel, because the gray-level value is known to be 2. To code the entire image, we need only one value. This is called the certain event; it has a probability of 1.

The two preceding examples illustrate the range of the entropy:

$0 \leq \text{Entropy} \leq \log_2(L)$

The examples also illustrate the information theory perspective on information and randomness. The more randomness that exists in an image, the more evenly distributed the gray levels, and the more bits per pixel are required to represent the data.

The entropy measure also provides us with a metric to evaluate coder performance. We can measure the average number of bits per pixel (Length) in a coder by the following:

$$L_{ave} = \sum_{i=0}^{L-1} l_i \, p_i$$

Where *li* = length in bits of the code for ith gray level
pi = histogram-probability of ith gray level
This can then be compared to the entropy, which provides the theoretical minimum. The closer Lave is to the entropy, the better the coder.

4.3.1. Huffman Coding

The Huffman code, developed by D. Huffman in 1952, is a minimum length code. This means that given the statistical distribution of the gray levels (the histogram), the Huffman algorithm will generate a code that is as close as possible to the minimum bound, the entropy. This method results in a *variable length code*, where the code words are of unequal length. For complex images, Huffman coding alone will typically reduce the file by 10 to 50% (1.1:1 to 1.5:1), but this ratio can be improved to 2:1 or 3:1 by preprocessing for irrelevant information removal.

The Huffman algorithm can be described in five steps:

1. Find the gray-level probabilities for the image by finding the histogram.
2. Order the input probabilities (histogram magnitudes) from smallest to largest.
3. Combine the smallest two by addition.
4. GOTO step 2, until only two probabilities are left.
5. By working backward along the tree, generate code by alternating assignment of
0 and 1.

EXAMPLE (4-7):

We have an image with 2 bits/pixel, giving four possible gray levels. The image is 10 rows by 10 columns.

Step 1: we find the histogram for the image. This is shown in Figure (5.2-a), where we see that gray level 0 has 20 pixels, gray level 1 has 30 pixels, gray level 2 has 10 pixels, and gray level 3 has 40 pixels with the value. These are converted into probabilities by normalizing to the total number of pixels in the image.

Step 2: the probabilities are ordered as in Figure (5.2-b).

Step 3: we combine the smallest two by addition.

Step 4: repeats steps 2 and 3, where we reorder (if necessary) and add the two smallest probabilities as in Figure (5.2-d). This step is repeated until only two values remain.

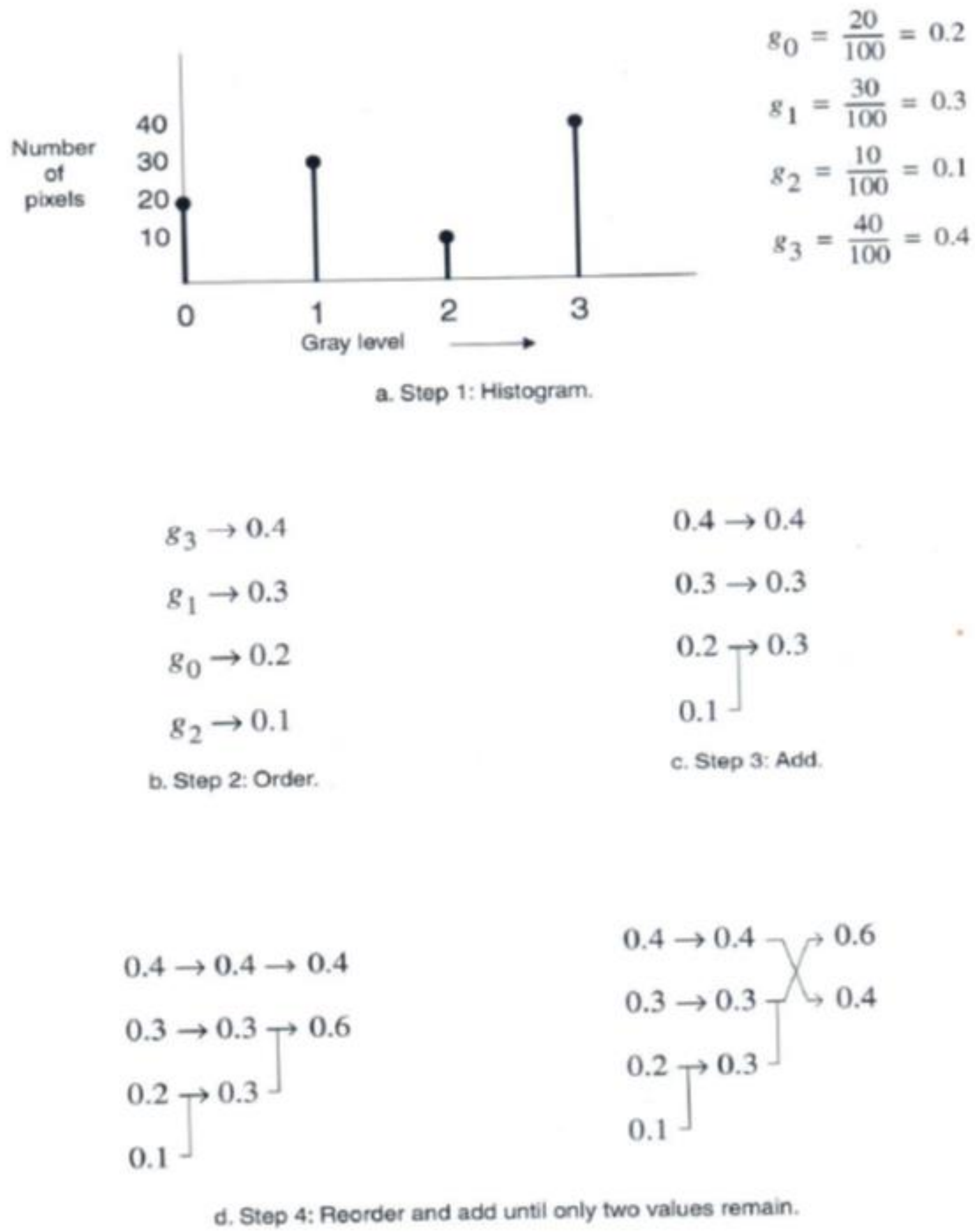


Figure (4.2) Huffman Coding Example

Because we have only two left in our example, we can continue to **step 5** where the actual code assignment is made. The code assignment is shown in Figure (5.3). We start on the right-hand side of this tree and assign 0's and 1's, working our way back to the original probabilities. Figure (5.3-a) shows the first assignment of 0 and 1. A 0 is assigned to the 0.6 branch, and a 1, to the 0.4 branch. In Figure (5.3-b) the assigned 0 and 1 are brought back along the tree, and wherever a branch occurs the code is put on both branches. Now (Figure 5.3-c) we assign the 0 and 1 to the branches labeled 0.3, appending to the existing code. Finally (Figure 5.3-d), the codes are brought back one more level, and where the branch splits another assignment of 0 and 1 occurs (at the 0.1 and 0.2 branch). Now we have the Huffman code for this image as shown in Table (4.2-1).

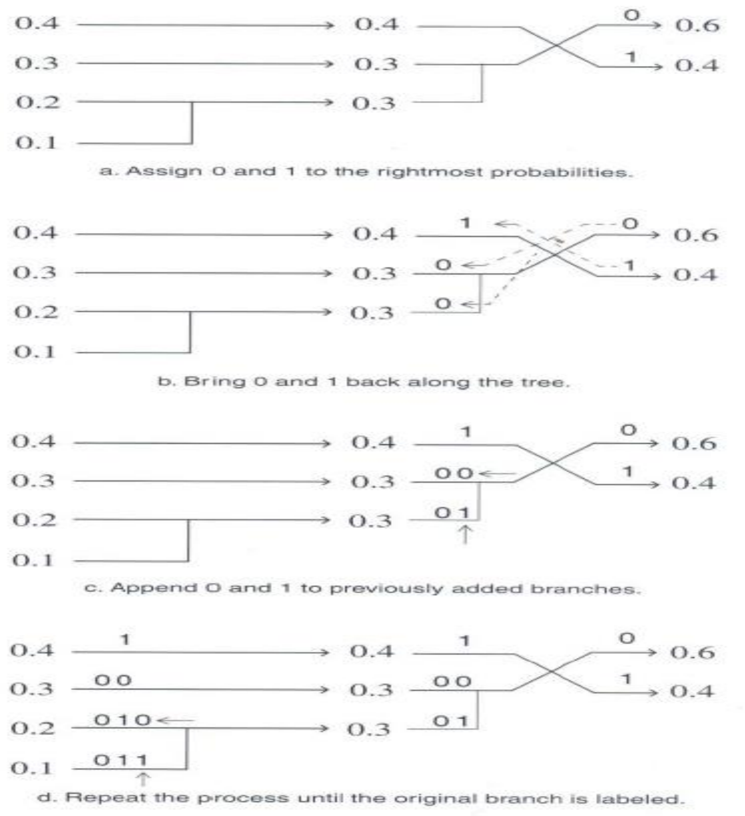

0.4
0.3
0.2
0.1
0.4
0.3
0.3
0
1
0.6
0.4
a. Assign 0 and 1 to the rightmost probabilities.
b. Bring 0 and 1 back along the tree.
c. Append 0 and 1 to previously added branches.
00
01
010
011
d. Repeat the process until the original branch is labeled.

Figure (4.3) Huffman Coding Example, Step 5

Original Gray level	**(Natural Code)**	**probability**	**Huffman Code**
0	**002**	**0.2**	**0102**
1	**012**	**0.3**	**002**
2	**102**	**0.1**	**0112**
3	**112**	**0.4**	**12**

Table (4.2-1) Huffman Code

Note that two of the gray levels now have 3 bits assigned to represent them, but one gray level only has 1 bit assigned to represent it. The gray level represented by 1 bit, *g3*, is the most likely to occur (40% of the time). The symbols with less information require fewer bits to represent them. The original image had an average of 2 bits/pixel.

The following example will examine the entropy in bits per pixel and average bit length for the Huffman coded image file.

EXAMPLE (4-8):

$$\text{Entropy} = -\sum_{i=0}^{3} p_i \log_2(p_i)$$

$$= -[(0.2)\log_2(0.2) + (0.3)\log_2(0.3) + (0.1)\log_2(0.1) + (0.4)\log_2(0.4)$$

$$\approx 1.846 \text{ bits/pixel}$$

(Note: $\log_2(x)$ can be found by taking $\log_{10}(x)$ and multiplying by 3.322)

$$L_{\text{ave}} = \sum_{i=0}^{L-1} l_i p_i$$

$$= 3(0.2) + 2(0.3) + 3(0.1) + 1(0.4)$$

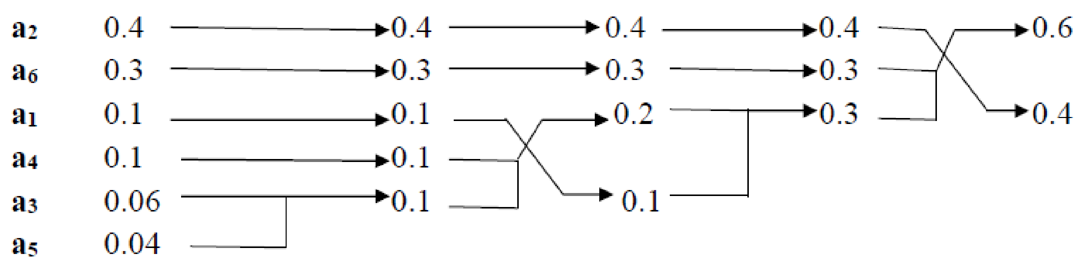
$$= 1.9 \text{ bits/pixel} \quad \text{(Average length with Huffman code)}$$

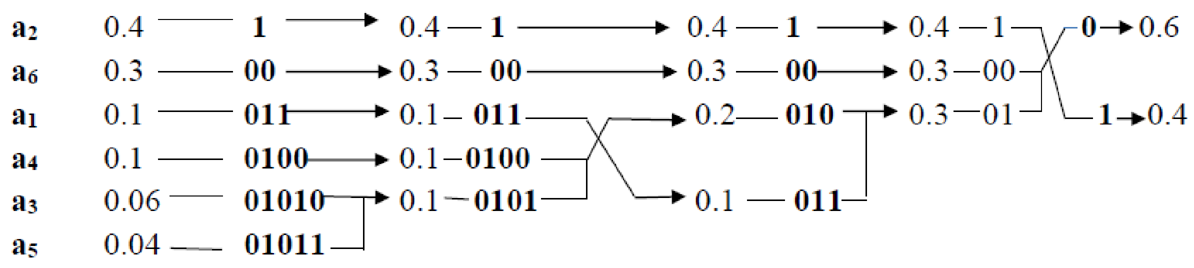
In the example, we observe a 2.0:1.9 compression, which is about a 1.05 compres-sion ratio, providing about 5% compression. From the example we can see that the Huffman code is highly dependent on the histogram, so any preprocessing to the histo-gram may help the compression ratio.

EXAMPLE (4-9): find the Huffman codes for the following symbols with the following probability.

Symbol	Probability
a_1	0.1
a_2	0.4
a_3	0.06
a_4	0.1
a_5	0.04
a_6	0.3

Solution:




a2 0.4 — 1 → 0.4 — 1 → 0.4 — 1 → 0.4 — 1 — 0 → 0.6
a6 0.3 — 00 → 0.3 — 00 → 0.3 — 00 → 0.3 — 00 — 1 → 0.4
a1 0.1 — 011 → 0.1 — 011 → 0.2 — 010 → 0.3 — 01
a4 0.1 — 0100 → 0.1 — 0100 → 0.1 — 011
a3 0.06 — 01010 → 0.1 — 0101
a5 0.04 — 01011

4.3.2 Run-length coding

Run-length coding (RLC) is an image compression method that works by counting the number of adjacent pixels with the same gray-level value. This count, called the *run length,* is then coded and stored. The basic methods are used primarily for binary (two-valued) images and extended versions for gray-scale images, but can work with complex images that have been preprocessed by thresholding to reduce the number of gray levels to two.

- ***Binary image run-length Code***: where the image is in a binary (0, 1). Starting

With 0 level, what is required in coding is only to send the length until a black pixel and so on.

- The first step is to define the required parameters. We can use either horizontal RLC, counting along the rows, or vertical RLC, counting along the columns. In basic horizontal RLC the number of bits used for the coding depends on the number of pixels in a row. If the row has 2*n* pixels, then the required number of bits is *n*, so that a run that is the length of the entire row can be coded.

EXAMPLE (4-10)

A 256 x 256 image requires 8 bits, since $2^8 = 256$.

EXAMPLE (4-11)

A 512 x 512 image requires 9 bits, since $2^9 = 512$.

- The next step is to define a convention for the first RLC number in a row—does it represent a run of 0's or 1's? Defining the convention for the first RLC number to represent 0's, we can look at the following example.

EXAMPLE (4-12)

The image is an 8 x 8 binary image, which requires 3 bits for each run-length coded word. To apply RLC to this image, using horizontal RLC:

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

The RLC numbers are:

First row: 8

Second row: 0, 4, 4

Third row: 1, 2, 5

Fourth row: 1, 5, 2

Fifth row: 1, 3, 2, 1, 1

Sixth row: 2, 1, 2, 2, 1

Seventh row: 0, 4, 1, 1, 2

Eighth row: 8

Note that in the second and seventh rows, the first RLC number is 0, since we are using the convention that the first number corresponds to the number of zeros in a run.

- ***Gray level run-length code*:** In this method, two parameters are used to characterize the run. The pair (G, *L)* correspond to the gray-level value G and the run-length L. It works by starting at the first point in the upper left corner and looking at the first pixel and its following neighbors across the line, determines how many following pixels have the same brightness as the first one. According to that a new code will be. This technique is only effective with images containing a small number of gray levels.

The Run-Length method provides around 1.5:1 compression ratio on gray-scale image, ratios from 4:1 to greater than 10:1 on binary image. Applying bit plane run length coding to gray –scale images yield up to 2:1.

EXAMPLE (5-12)

Given the following 8 x 8, 4-bit images.

$$\begin{bmatrix}
10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\
10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\
0 & 0 & 0 & 10 & 10 & 10 & 0 & 0 \\
5 & 5 & 5 & 0 & 0 & 0 & 0 & 0 \\
5 & 5 & 5 & 10 & 10 & 9 & 9 & 10 \\
5 & 5 & 5 & 4 & 4 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

First row: 10,8

Second row: 10,5 12,3

Third row: 10,5 12,3

Fourth row: 0,3 10,3 0,3

Fifth row: 5,3 0,5

Sixth row: 5,3 10,2 9,2 10,1

Seventh row: 5,3 4,3 0,2

Eighth row: 0,8

These numbers are then stored in the RLC compressed file as:

10, 8, 10, 5, 12, 3, 10, 5, 12, 3, 0, 3, 10, 3, 0, 3, 5, 3, 0, 5, 5, 3, 10, 2, 9, 2, 10, 1, 5, 3, 4, 3, 0, 2, 0, 8

The corresponding gray-level pairs are as follows.

(10,8) (10,5)............(0,8)

Chapter Five: Transformation

5.1 Colors Transforms

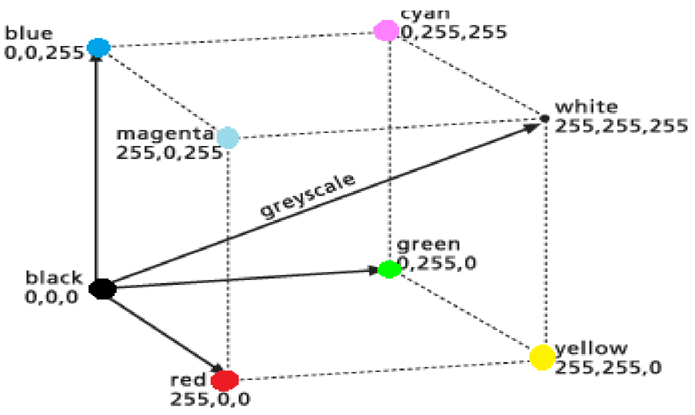
Any color that can be represented on a computer monitor is specified by means of the three basic colors- Red, Green and Blue called the RGB colors. By mixing appropriate percentages of these basic colors, one can design almost any color one ever imagines.

5.1-1 RGB color space

The model of designing colors based on the intensities of their RGB components is called the RGB model (as shown in Figure 6.1), and it's a fundamental concept in computer graphics. Each color, therefore, is represented by a triplet (Red, Green, and Blue), in which red, green and blue has three bytes that represent the basic color components. The smallest value, 0, indicates the absence of color. The largest value, 255, indicates full intensity or saturation. The triplet (0, 0, 0) is black, because all colors are missing, and the triplet (255, 255, 255) is white. Other colors have various combinations:
(255,0,0) is pure red, (0,255,255) is a pure cyan (what one gets when green and blue are mixed), and (0,128,128) is a mid-cyan (a mix of mid-green and mid-blue tones). The possible combinations of the three basic color components are 256 x 256 x 256, or 16,777,216 colors.

The process of generating colors with three basic components is based on the RGB Color cube (as shown in figure 5.1). The three dimensions of the color cube correspond to the three basic colors. The cube's corners are assigned each of the three primary colors, their complements, and the colors black and white. Complementary colors are easily calculated by subtracting the Color values from 255. For example, the color (0, 0,255) is a pure blue tone. Its complementary color is (255-0,255-0,255-255), or (255, 255, 0), which is a pure yellow tone. Blue and Yellow are complementary colors, and they are mapped to opposite corners of the cube. The same is true for red and cyan, green and magenta, and black and white. Adding a color to its complement gives white. It is noticed that the components of the colors at the corners of the cube have either zero or full intensity. As we move from one corner to another along the same edge of the cube, only one of its components changes value. For example, as we move from the Green to the Yellow corner, the Red component changes from 0 to 255. The other two components remain the same. As we move between these two corners, we get all the available tones from Green to Yellow (256 in all). Similarly, as one moves from the Yellow to the Red corner, the only component that changes are the Green, and one gets all the available shades from Yellow to Red. This range of similar colors is called gradient.

Although we can specify more than 16 million colors, we can’t have more than 256 shades of Gray. The reason is that a gray tone, including the two extremes (black and white), is made up of equal values of all the three primary colors. This is seen from the RGB cube as well. Gray shades lie on the cube’s diagonal that goes from black to white. As we move along this path, all three basic components change value, but they are always equal. The value (128,128,128) is a mid-gray tone, but the values (129,128,128) aren’t gray tones, although they are too close for the human eye to distinguish. That's why it is wasteful to store grayscale pictures using 16-million color True Color file formats. A 256-color file format stores a grayscale just as accurately and more compactly. Once an image is known in grayscale, we needn't store all three bytes per pixel. One value is adequate (the other two components have the same value).



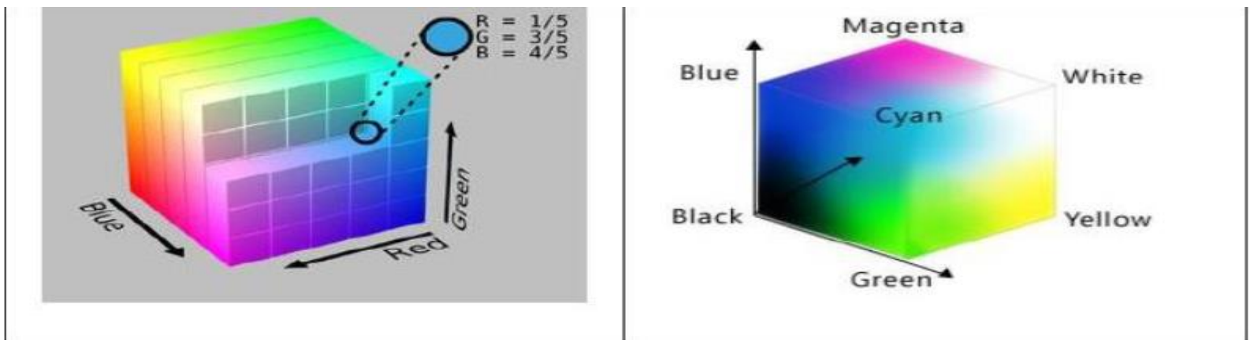


Figure (5.1) color specification of RGB color cube

5.1-2 HSV Color Format

HSV color space HSL (Hue-Saturation-Lightness) or HSI (Hue-Saturation-Intensity) is one color space, which describes colors as perceived by human beings. HSI (or HSV) stands for Hue (H), (S) Saturation and Intensity (I) (or value V). For example, a blue car reflects blue hue. Moreover is also attributing of the human perception. The hue which is essentially the chromatic component of our perception may again be considered as weak hue or strong hue. The colorfulness of a color is described by the saturation component.

For example, the color from a single monochromatic source of light, which produce colors of a single wavelength only , is highly saturated , while the colors comprising hues of different wavelengths have little chroma and have less saturation. The gray colors do not have any hue and hence they have less saturation or unsaturated. Saturation is thus a measure of colorfulness or whiteness in the color perceived.

The lightness (L) or intensity (I) or value (V) essentially provides a measure of the brightness of colors. This gives a measure of how much light is reflected from the object or how much light is emitted from a region.

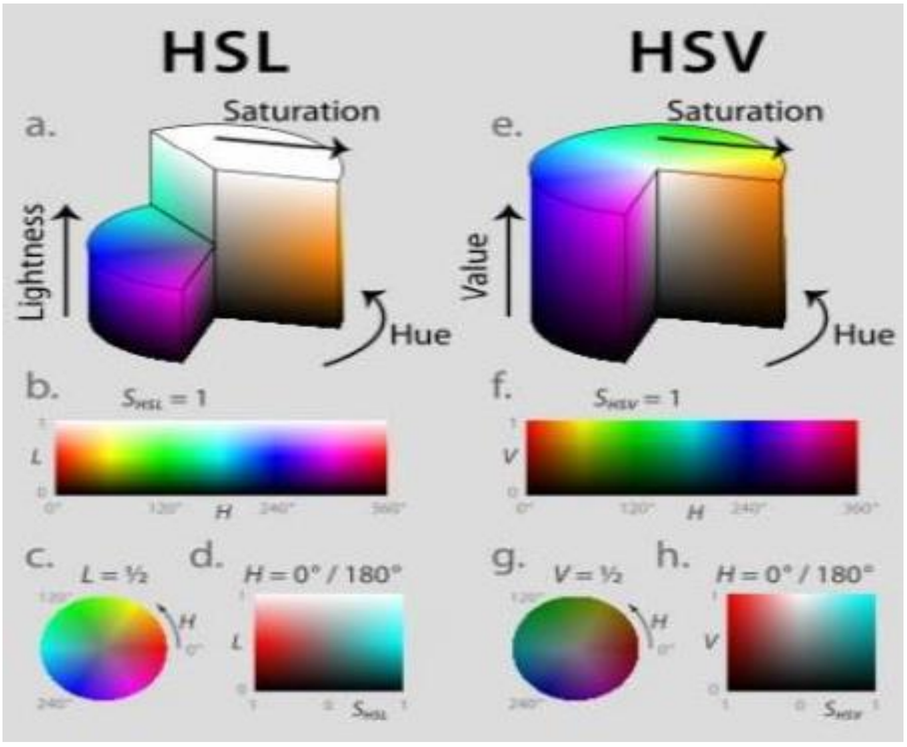


Figure (5.2) HSL and HSV Color Space

The HSV image may be computed from RGB image using different transformation. Some of them are as follows:

- The simplest form of HSV transformation is:

H= tan [3(G-B)/(R-G) +(R-B)]

S= 1-(min(R, G, B)/V)

V= (R+G+B/3)

However, the hue (H) becomes undefined when saturation S=0.

- The most popular form of HSV transformation is shown next, where the r,g,b values are first obtained by normalizing each pixel such that

r=(R/(R+G+B)), g=(G/(R+G+B)), b=(B/(R+G+B))

Accordingly, the H, S and V value can be computed as:

V=max (r, g, b)

$$S = \begin{cases} 0, & if \;\; V = 0 \\ V - \frac{\min(r,g,b)}{V}, & if \;\; V > 0 \end{cases}$$

$$H = \begin{cases} 0, & if\; S = 0 \\ 60 * \left[\frac{(g-b)}{s} * V\right] & if\; V = r \\ 60 * \left[2 + \left(\frac{(b-r)}{s} * V\right)\right] & if\; V = g \\ 60 * \left[4 + \left(\frac{(r-g)}{s} * V\right)\right] & if\;\; V = b \end{cases}$$

H=H+360 if H<0

The HSV color space is compatible with human color perception.

5.1.3. YCbCr Color Format

Another color space in which luminance and chrominance are separately represented is the YCbCr. It is appropriate for digital coding of standard TV images. The Y component takes values from

16 to 235, while Cb and Cr take values from 16 to 240. This color space was used in a process of video sequence encoding on videodisks. Other current applications in image compression (e.g. JPEG format) often employ YCbCr space as a quantization space. They are obtained from R, G, B values as follows:

Y= 0.299R + 0.587G + 0.114B

Cb= - 0.169R – 0.331G + 0.500B

Cr= 0.500R – 0.418G – 0.081B

5.2. Discrete Transforms

The transform maps image data into a different mathematical space via a transformation equation.However, the color transformations mapped data from one color space to another color space with a one-to-one correspondence between a pixel in the input and the output. Here, we are mapping the image data from the spatial المكانية (time) domain spectral الطيفية domain, where all the pixels in the input (spatial domain) contribute to each value in the output (frequency domain) as illustrated in the figure (6.3) and (6.4) below:

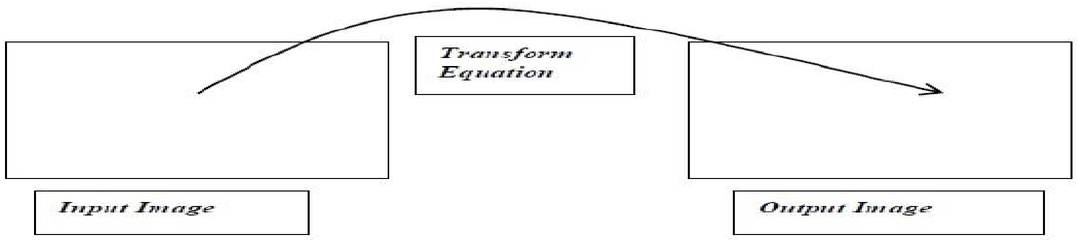


Figure (5.3) color transform use a single –pixel to single –pixel mapping

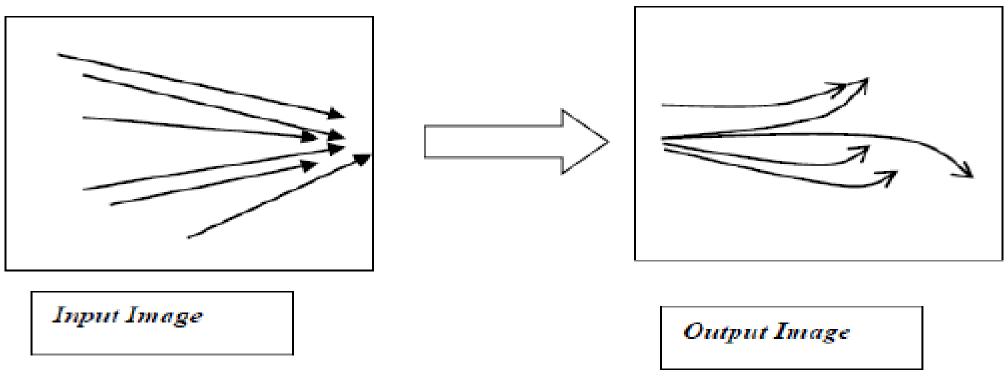


Figure (5.4) All pixel in the input image contribute to each value in the output image color transform use a single-pixel to single-pixel mapping.

The most distinguished information is hidden in the frequency content. Image transforms are designed to have two properties:

1- Reduce image redundancy by reducing the sizes of most pixels.
2- Identify the less important parts of the image by isolating the various frequencies of the image.

Question: Why image frequencies are important?

Answer: Because it depends on the fact that low frequencies correspond to the important image features, whereas high frequencies correspond to the details of the image, which are less important. Thus, when a transform isolates the various image frequencies, pixels that correspond to high frequencies can be quantized تكميم heavily بشكل كبير , whereas pixels that correspond to low frequencies should be quantized lightly or not at all. This is how a transform can compress an image very effectively by losing information, but only information associated with unimportant image details. 5.2-1 Discrete Fourier Transform (FT)

The Fourier transform is the most well-known, and the most widely used transform. It was developed by Baptiste Joseph Fourier (1768-1830) to explain the distribution of temperature and heat conduction (توصيل حراري). Since that time the Fourier transform has found numerous uses, including vibration (الاهتزاز) analysis in mechanical engineering, circuit analysis in electrical engineering, and here in computer imaging. In mathematics, a Fourier series decomposes periodic functions or periodic signals into the sum of a (possibly infinite) set of simple oscillating تتأرجح functions, namely sine and cosine (or complex exponentials).

This transform allows for decomposition of an image into weighted sum of 2-d sinusoidal جيبية terms. Assuming an N×N image, the equation for the 2-D discrete Fourier

$$F(u,v)=\frac{1}{N}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I(r,c)\, e^{-j2\pi\frac{(ur+vc)}{N}}$$

The base of the natural logarithmic function e is about 2.71828; j, the imaginary coordinate for a complex number, equals $\sqrt{-1}$. The basic functions are sinusoidal in nature, as can be seen by Euler's identity:

$$e^{-jx}=\cos x - j\sin x$$

So we can also write the Fourier transform equation as

$$F(u,v)=\frac{1}{N}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I(r,c)\left[cos\left(\frac{2\pi}{N}(ur+vc)\right)-j\,sin\left(\frac{2\pi}{N}(ur+vc)\right)\right]$$

In this case, F(u, v) is also complex, with the real part corresponding to the cosine terms and the imaginary part corresponding to the sine terms. If we represent a complex spectral component by: F(u, v) = R(u, v) +j I(u, v)

Where R(u, v) is the real part and I(u, v) is the imaginary part, then we can define the magnitude and phase of a complex spectral component as:

$$MAGNITUDE = |F(u,v)| = \sqrt{[R(u,v)]^2 + [I(u,v)]^2}$$

and

$$PHASE = \phi(u,v) = tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right]$$

The magnitude of a sinusoid is simply its peak value, the phase, data contain information about where objects are in an image. After we perform the transform, if we want to get our original image back, we need to apply the inverse transform. The inverse Fourier transform is given by:

$$F^{-1}[F(u,v)] = I(r,c) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi\frac{(ur+vc)}{N}}$$

The F^{-1}[ ] notation represents the inverse transform.

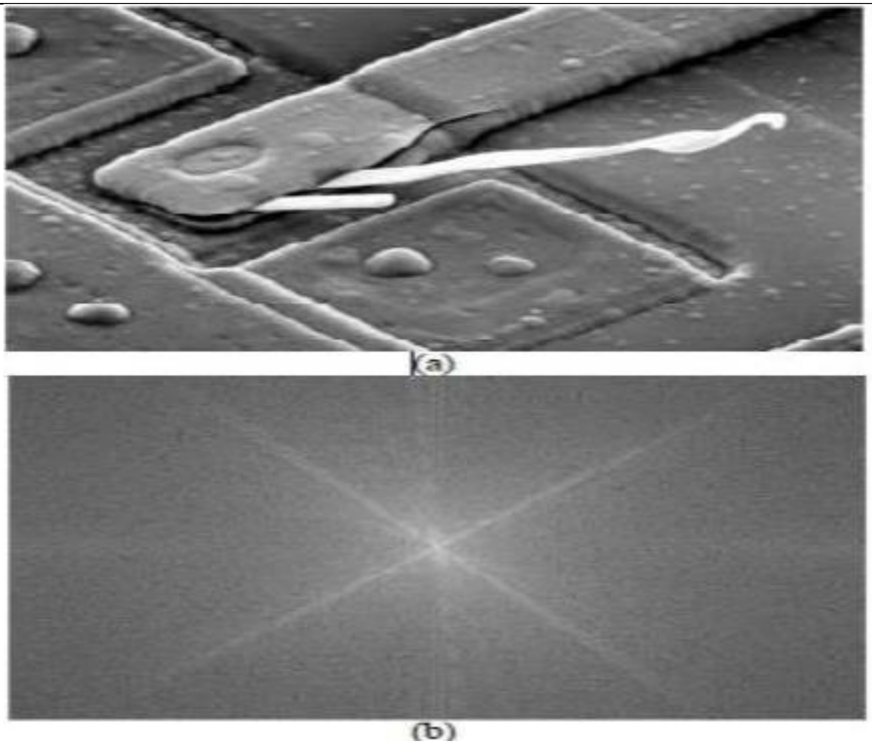


Figure (5.5) : (a) gray image (b) Fourier spectrum

5.2-2 Discrete Cosine Transform (DCT)

The cosine transform, like the Fourier transform, uses sinusoidal basis functions. The difference is that the cosine transform basis functions are not complex; they use only cosine functions and not sine functions. Assuming an N x N image, the discrete cosine transform equation is given by

$$C(u,v) = \alpha(u)\alpha(v)\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I(r,c)\cos\left[\frac{(2r+1)u\pi}{2N}\right]\left[\frac{(2c+1)v\pi}{2N}\right]$$

Where

$$\alpha(u)\alpha(v) = \begin{cases} \sqrt{\frac{1}{N}} & for\ u,v = 0 \\ \sqrt{\frac{2}{N}} & for\ u,v = 0,1,\ldots,N-1 \end{cases}$$

Because this transform uses only the cosine function, it can be calculated using only real arithmetic (not complex). This also affects the implied symmetry of the transform, which is explored in the filtering section.

The important feature of cosine transform is often used in image compression, in particular the Joint Photographic Experts Group (JPEG) image compression method, which has been established as an international standard. The transform coefficients C(u, v) tell us the amount of that particular basis image that the original image I(r, c) contains. The inverse cosine transform is given by

$$C^{-1}[C(u,v)] = I(r,c) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v)cos\left[\frac{(2r+1)u\pi}{2N}\right]cos\left[\frac{(2c+1)v\pi}{2N}\right]$$

5.3 Wavelet Transform

- Wavelet transforms are based on sub-sampling high and low pass filters.
- These filters are matched in such a way that they split the data into high and low pass bands with or without losing any information.
- Wavelet filters have been designed for a wide range of applications and many different sets.
- Wavelets are functions defined over a finite interval. The purpose of wavelet transform is to change the data from Time-space domain to Time-frequency domain which makes better compression results. The simplest form of wavelets, the Haar wavelet function is defined as:

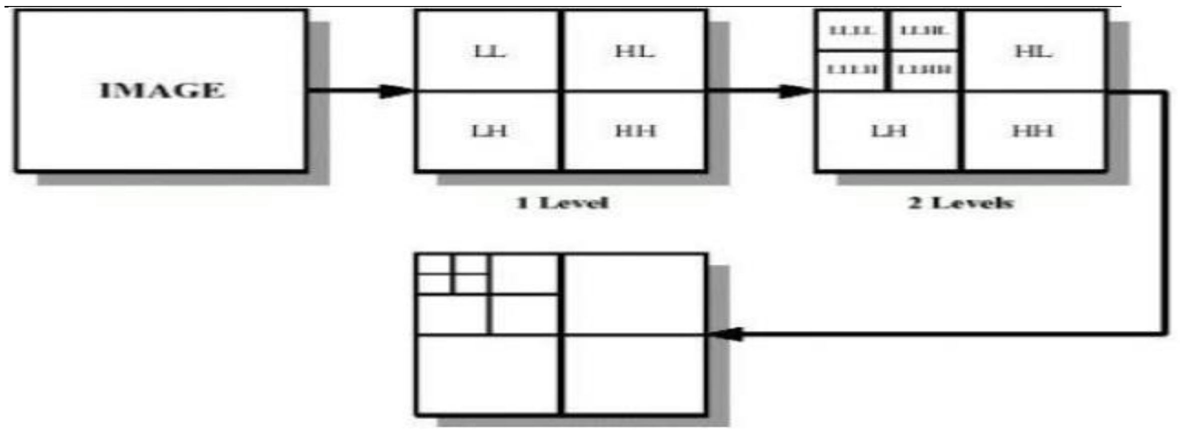


Figure (5.6) block diagram of 3-level wavelet transform

The four bands that are formed are referred to as the Low-Low (LL), Low-High (LH), High-Low (HL) and High-High (HH). The LL band still has image-like information and so it is possible to apply the set of wavelet filters, in the same way as applied to the original image. This process of dividing the image into sub-bands can be continued as far as desired (to the resolution of the image), but for image compression it is usually only continued to 4 or 5 levels. A typical final image is shown in figure below:

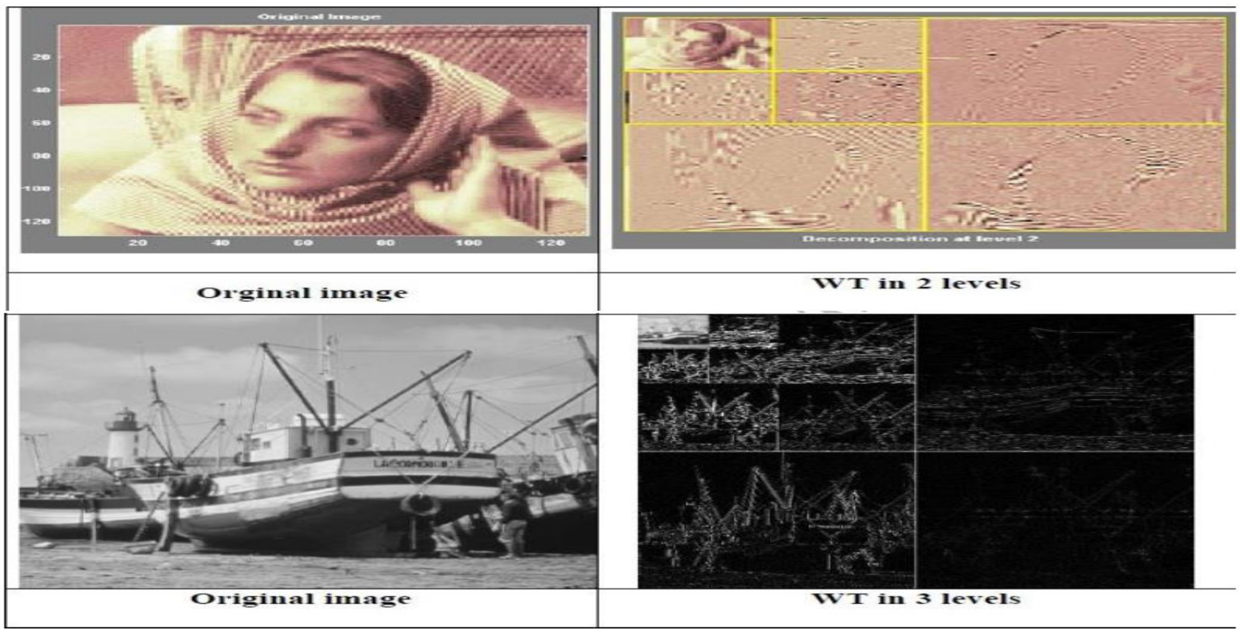


Figure (5.7) example of Wavelet Transform

Haar Wavelet:

The Haar transform works mainly on pairs of data items from the signal begin processed and execute two step of calculation, the first step is:

Li= 1/ √2 (X_{2i}+X_{2i}+1)

Which result an approximation (average) of the two data items, the second is:

Hi= 1/ √2 (X_{2i}+X_{2i}+1)

Which also results an approximation (difference) of the two data items

Where

X is the signal data with length N

L is the Low sub –band with length N/2

H is the High sub-band with length N/2

i = 0…….N/2

The invers Haar transform formula is:

X_{2i}= 1/ √2 (Li+Hi)

X_{2i}+1= 1/ √2 (Li-Hi)

The Haar wavelet transform has a number of advantages:

- It is conceptually simple.
- It is fast.
- It is memory efficient, since it can be calculated in place without a temporary Array.
- It is exactly reversible without the edge effects that are a problem with other Wavelet

Figure (6.6) example of applying 3-level of Wavelet Transform

The attracting features of the Haar transform is:

- Fast and simple for implementation.
- Efficient memory because it can be calculated without temporary array.

- It is exactly reversible without the edge effects which is a problem with other wavelet transform.
- Ability to analyze the local feature, make it a potential candidate in modern electrical and computer engineering applications, such as signal and image compression.

Example (1): Apply 1- Level Harr Wavelet Transform on the following 4x4 image:

137	134	129	131
135	141	133	132
138	134	134	131
131	129	133	131

Solution:

-لايجاد جزء ال (L) low سنأخذ كل نقطتين متجاورتين لعمودين ونقوم بجمعهما وقسمتهما على 2

ملاحظة: نأخذ فقط الجزء الصحيح ونهمل الكسر عند ايجاد النتيجة. -

(137+134)/2=135.5=135   (129+131)/2=130

(135+141)/2=138   (133+132)/2=132.5=132

(138+134)/2=136   (134+131)/2=132.5=132

(135+129)/2=132   (133+131)/2=132

-لايجاد جزء الhigh (H) سنأخذ كل نقطتين متجاورتين لعمودين ونقوم بطرحهما وقسمتهما على 2

137-134)/2=1.5=1   (129 -131)/2=1

(135 -141)/2= - 3   (133 - 132)/2=0.5=0

(138 -134)/2= 2   (134 -131)/2=1.5=1

(135 -129)/2= 3   (133 - 131)/2=1

L		H	
135	130	1	-1
138	132	-3	0
136	132	2	1
132	132	3	1

-نلاحظ ان الصورة انقسمت الى جزئين.

-المرحلة الثانية نقوم بتكرار نفس العملية السابقة على الصورة الناتجة لتقسيمها الى اربعة اجزاء ولكن

بأخذ نقطتين لصفين وكالتالي:

(LL)	**(HL)**
(135+138)/2=136.5=136	(1+(-3))/2= -1
(136+132)/2=134	(2+3)/2= 2.5=2
(130+132)/2=131	(-1+0)/2= - 0.5=0
(132+133)/2=132	(1+1)/2=1
(LH)	**(HH)**
(135 -138)/2= -1.5= -1	(1-(-3))/2= 2
(136 -132)/2= 2	(2-3)/2= -0.5=0
(130 - 132)/2= -1	(-1- 0)/2= - 0.5=0
(132-133)/2= 0	(1-1)/2=-0.5=0

LL		HL	
136	131	-1	0
134	132	2	1
LH		**HH**	
-1	-1	2	0
2	0	0	0