MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

PG DEPARTMENT OF COMPUTER SCIENCE

SUBJECT NAME: DIGITAL IMAGE PROCESSING

SUBJECT CODE: PSD2B

SEMESTER: II

PREPARED BY: PROF. D.SELVARAJ

Digital Image Processing Syllabus
UNIT I
Introduction – steps in image processing, Image acquisition, representation, sampling and quantization, relationship between pixels. – color models – basics of color image processing.
UNIT II
Image enhancement in spatial domain – some basic gray level transformations – histogram processing – enhancement using arithmetic , logic operations – basics of spatial filtering and smoothing.
UNIT III
Image enhancement in Frequency domain – Introduction to Fourier transform: 1- D, 2 –D DFT and its inverse transform, smoothing and sharpening filters.
UNIT IV
Image restoration: Model of degradation and restoration process – noise models – restoration in the presence of noise- periodic noise reduction.. Image segmentation: Thresholding and region based segmentation.
UNIT V
Image compression: Fundamentals – models – information theory – error free compression –Lossy compression: predictive and transform coding. JPEG standard.

Digital Image Processing

UNIT I

Fundamental steps in Digital Image Processing :

1. Image Acquisition

This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling etc.

2. Image Enhancement

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

3. Image Restoration

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

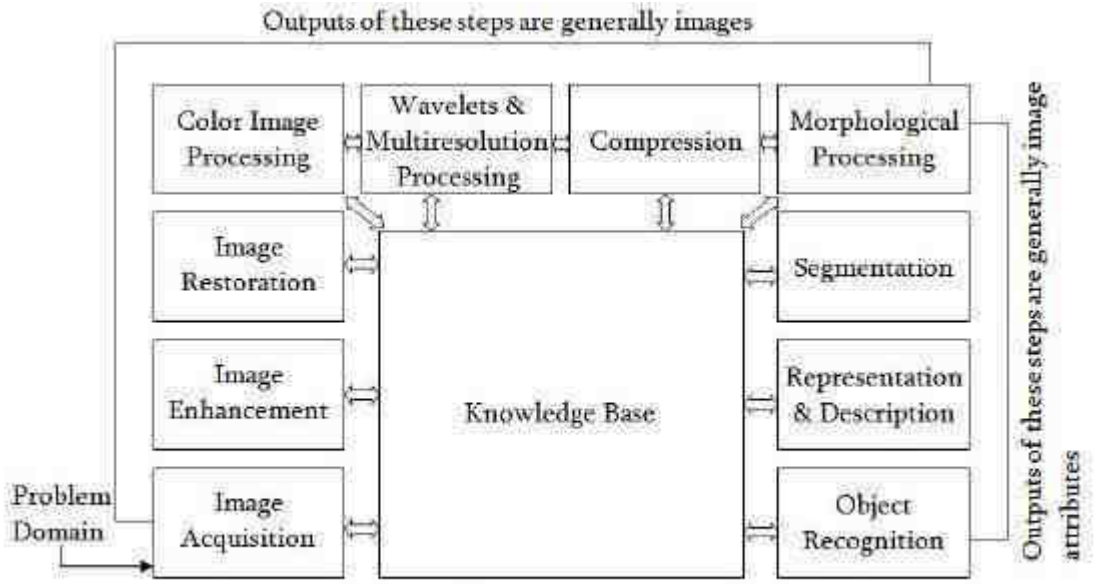


Figure 1

4. Color Image Processing

Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modeling and processing in a digital domain etc.

5. Wavelets and Multiresolution Processing

Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

6. Compression

Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

7. Morphological Processing

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

8. Segmentation

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

9. Representation and Description

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

10. Object recognition

Recognition is the process that assigns a label, such as, "vehicle" to an object based on its descriptors.

11. Knowledge Base:

Knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information. The knowledge base also can be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem or an image database containing high-resolution satellite images of a region in connection with change-detection applications.

Image Acquisition

Before any video or image processing can commence an image must be captured by a camera and converted into a manageable entity. This is the process known as image acquisition. The image acquisition process consists of three steps; energy reflected from the object of interest, an optical system which focuses the energy and finally a sensor which measures the amount of energy. In Fig. 2.1 the three steps are shown for the case of an ordinary camera with the sun as the energy source. In this topic each of these three steps are described in more detail.

Energy

In order to capture an image a camera requires some sort of measurable energy. The energy of interest in this context is light or more generally electromagnetic waves. An electromagnetic (EM) wave can be described as massless entity, a photon, whose electric and magnetic fields vary sinusoidally, hence the name wave. The photon belongs to the group of fundamental particles and can be described in three different ways:

- **A photon can be described by its energy** E, which is measured in electronvolts [eV]
- **A photon can be described by its frequency f**, which is measured in Hertz [Hz]. A frequency is the number of cycles or wave-tops in one second
- **A photon can be described** by its wavelength λ, which is measured in meters [m]. A wavelength is the distance between two wave-tops

The three different notations are connected through the speed of light c and Planck's constant h:

$$\lambda = \frac{c}{f}, \qquad E = h \cdot f \quad \Rightarrow \quad E = \frac{h \cdot c}{\lambda} \tag{2.1}$$

An EM wave can have different wavelengths (or different energy levels or different frequencies). When we talk about all possible wavelengths we denote this as the EM spectrum, see Fig. 2.2.

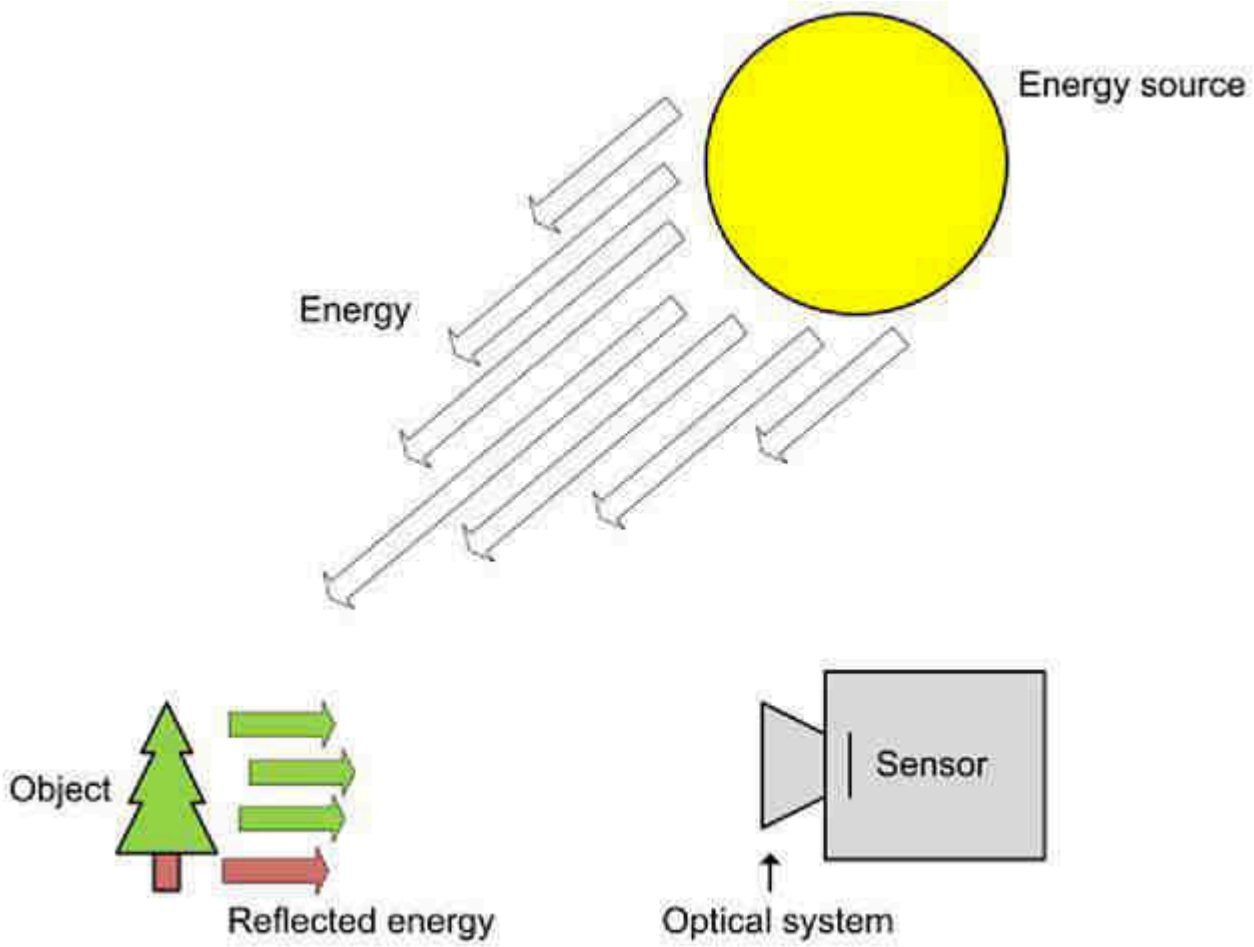


Fig. 2.1 Overview of the typical image acquisition process, with the sun as light source, a tree as object and a digital camera to capture the image. An analog camera would use a film where the digital camera uses a sensor.

In order to make the definitions and equations above more understandable, the EM spectrum is often described using the names of the applications where they are used in practice. For example, when you listen to FM-radio the music is transmitted through the air using EM waves around

100 · 106 Hz, hence this part of the EM spectrum is often denoted "radio". Other well-known applications are also included in the figure.

The range from approximately 400-700 nm (nm = nanometer = 10-9) is denoted the visual spectrum. The EM waves within this range are those your eye (and most cameras) can detect. This means that the light from the sun (or a lamp) in principle is the same as the signal used for transmitting TV, radio or for mobile phones etc. The only difference, in this context, is the fact that the human eye can sense EM waves in this range and not the waves used for e.g., radio. Or in other words, if our eyes were sensitive to EM waves with a frequency around 2 · 109 Hz, then your mobile phone would work as a flash light, and big antennas would be perceived as "small suns". Evolution has (of course) not made the human eye sensitive to such frequencies but rather to the frequencies of the waves coming from the sun, hence visible light.

Illumination

To capture an image we need some kind of energy source to illuminate the scene. In Fig. 2.1 the sun acts as the energy source. Most often we apply visual light, but other frequencies can also be applied, see Sect. 2.5.

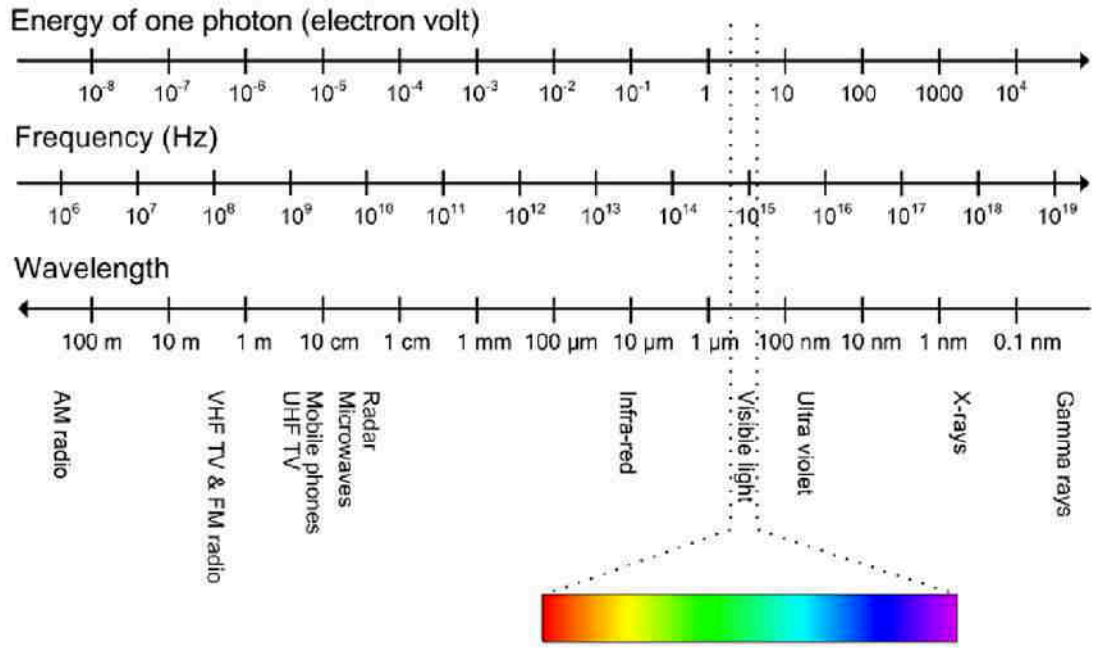


Fig. 2.2 A large part of the electromagnetic spectrum showing the energy of one photon, the frequency, wavelength and typical applications of the different areas of the spectrum

Fig. 2.3 The effect of illuminating a face from four different directions

If you are processing images captured by others there is nothing much to do about the illumination (although a few methods will be presented in later topics) which was probably the sun and/or some artificial lighting. When you, however, are in charge of the capturing process yourselves, it is of great importance to carefully think about how the scene should be lit. In fact, for the field of Machine Vision it is a rule-of-thumb that illumination is 2/3 of the entire system design and software only 1 /3. To stress this point have a look at Fig. 2.3. The figure shows four images of the same person facing the camera. The only difference between the four images is the direction of the light source (a lamp) when the images were captured!

Another issue regarding the direction of the illumination is that care must be taken when pointing the illumination directly toward the camera. The reason being that this might result in too bright an image or a nonuniform illumination, e.g., a bright circle in the image. If, however, the outline of the object is the only information of interest, then this way of illumination—denoted backlighting—can be an optimal solution, see Fig. 2.4.

Fig. 2.4 Backlighting. The light source is behind the object of interest, which makes the object stand out as a black silhouette. Note that the details inside the object are lost

Even when the illumination is not directed toward the camera overly bright spots in the image might still occur. These are known as highlights and are often a result of a shiny object surface,

which reflects most of the illumination (similar to the effect of a mirror). A solution to such problems is often to use some kind of diffuse illumination either in the form of a high number of less-powerful light sources or by illuminating a rough surface which then reflects the light (randomly) toward the object.

Even though this text is about visual light as the energy form, it should be mentioned that infrared illumination is sometimes useful. For example, when tracking the movements of human body parts, e.g. for use in animations in motion pictures, infrared illumination is often applied. The idea is to add infrared reflecting markers to the human body parts, e.g., in the form of small balls. When the scene is illuminated by infrared light, these markers will stand out and can therefore easily be detected by image processing. A practical example of using infrared illumination is given in Chap. 12.

The Optical System

After having illuminated the object of interest, the light reflected from the object now has to be captured by the camera. If a material sensitive to the reflected light is placed close to the object, an image of the object will be captured. However, as illustrated in Fig. 2.5, light from different points on the object will mix—resulting in a useless image. To make matters worse, light from the surroundings will also be captured resulting in even worse results. The solution is, as illustrated in the figure, to place some kind of barrier between the object of interest and the sensing material. Note that the consequence is that the image is upside-down. The hardware and software used to capture the image normally rearranges the image so that you never notice this.

The concept of a barrier is a sound idea, but results in too little light entering the sensor. To handle this situation the hole is replaced by an optical system. This section describes the basics behind such an optical system. To put it into perspective, the famous space-telescope—the Hubble telescope—basically operates like a camera, i.e., an optical system directs the incoming energy toward a sensor. Imagine how many man-hours were used to design and implement the Hubble telescope. And still, NASA had to send astronauts into space in order to fix the optical system due to an incorrect design. Building optical systems is indeed a complex science! We shall not dwell on all the fine details and the following is therefore not accurate to the last micro-meter, but the description will suffice and be correct for most usages.

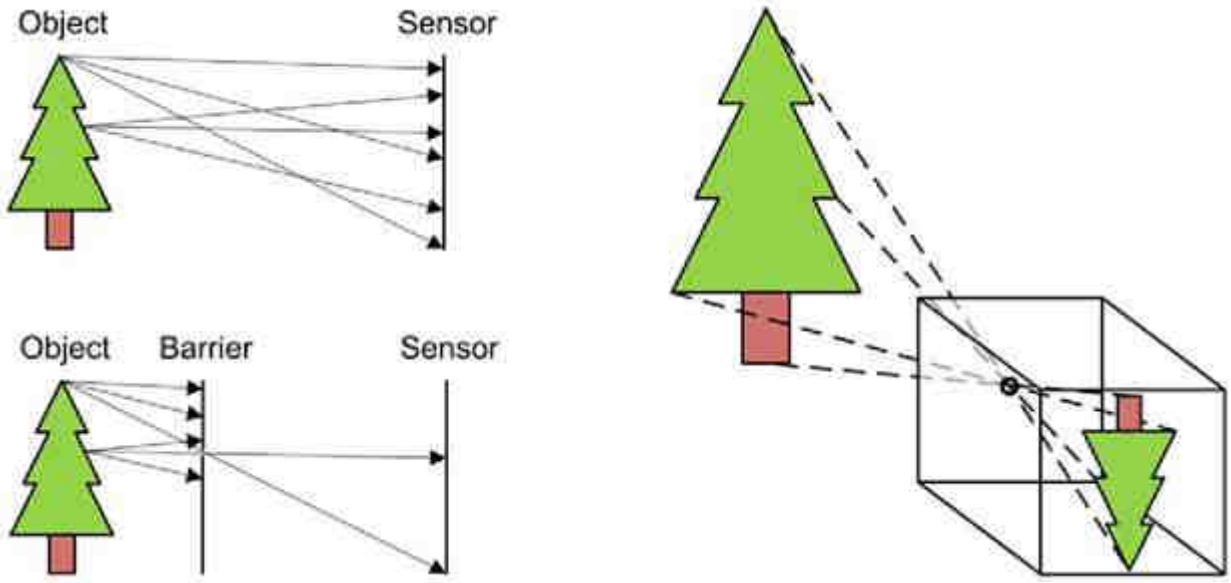


Fig. 2.5 Before introducing a barrier, the rays of light from different points on the tree hit multiple points on the sensor and in some cases even the same points. Introducing a barrier with a small hole significantly reduces these problems

The Lens

One of the main ingredients in the optical system is the lens. A lens is basically a piece of glass which focuses the incoming light onto the sensor, as illustrated in Fig. 2.6. A high number of light rays with slightly different incident angles collide with each point on the object's surface and some of these are reflected toward the optics. In the figure, three light rays are illustrated for two different points. All three rays for a particular point intersect in a point to the right of the lens. Focusing such rays is exactly the purpose of the lens. This means that an image of the object is formed to the right of the lens and it is this image the camera captures by placing a sensor at exactly this position. Note that parallel rays intersect in a point, F, denoted the Focal Point. The distance from the center of the lens, the optical center O, to the plane where all parallel rays intersect is denoted the Focal Lengthf. The line on which O and F lie is the optical axis.

Let us define the distance from the object to the lens as, g, and the distance from the lens to where the rays intersect as, b. It can then be shown via similar triangles, that

$$\frac{1}{g}+\frac{1}{b}=\frac{1}{f} \qquad (2.2)$$

f and b are typically in the range [1 mm, 100 mm]. This means that when the object is a few meters away from the camera (lens), then g has virtually no effect on the equation, i.e., b = f. What this tells us is that the image inside the camera is formed at a distance very close to the focal point. Equation 2.2 is also called the thin lens equation.

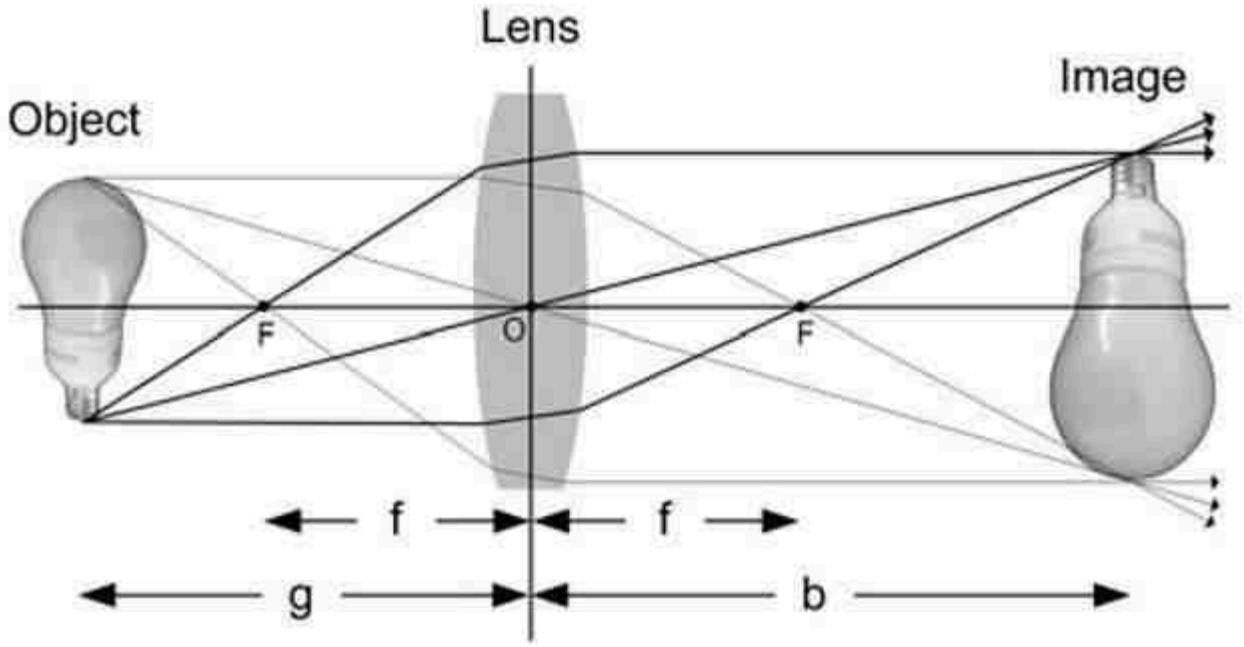


Fig. 2.6 The figure shows how the rays from an object, here a light bulb, are focused via the lens. The real light bulb is to the left and the image formed by the lens is to the right

Another interesting aspect of the lens is that the size of the object in the image, B, increases as f increased. This is known as optical zoom. In practice f is changed by rearranging the optics, e.g., the distance between one or more lenses inside the optical system.1 In Fig. 2.7 we show how optical zoom is achieved by changing the focal length. When looking at Fig. 2.7 it can be shown via similar triangles that

$$\frac{b}{B} = \frac{g}{G} \qquad (2.3)$$

where G is the real height of the object. This can for example be used to compute how much a physical object will fill on the imaging censor chip, when the camera is placed at a given distance away from the object.

Let us assume that we do not have a zoom-lens, i.e., f is constant. When we change the distance from the object to the camera (lens), g, Eq. 2.2 shows us that b should also be increased, meaning that the sensor has to be moved slightly further away from the lens since the image will be formed there. In Fig. 2.8 the effect of not changing b is shown. Such an image is said to be out of focus. So when you adjust focus on your camera you are in fact changing b until the sensor is located at the position where the image is formed.

The reason for an unfocused image is illustrated in Fig. 2.9. The sensor consists of pixels, as will be described in the next section, and each pixel has a certain size. As long as the rays from one point stay inside one particular pixel, this pixel will be focused. If rays from other points also intersect the pixel in question, then the pixel will receive light from more points and the resulting pixel value will be a mixture of light from different points, i.e., it is unfocused.

Referring to Fig. 2.9 an object can be moved a distance of gi further away from the lens or a distance of gr closer to the lens and remain in focus. The sum of gi and gr defines the total range an object can be moved while remaining in focus. This range is denoted as the depth-of-field.

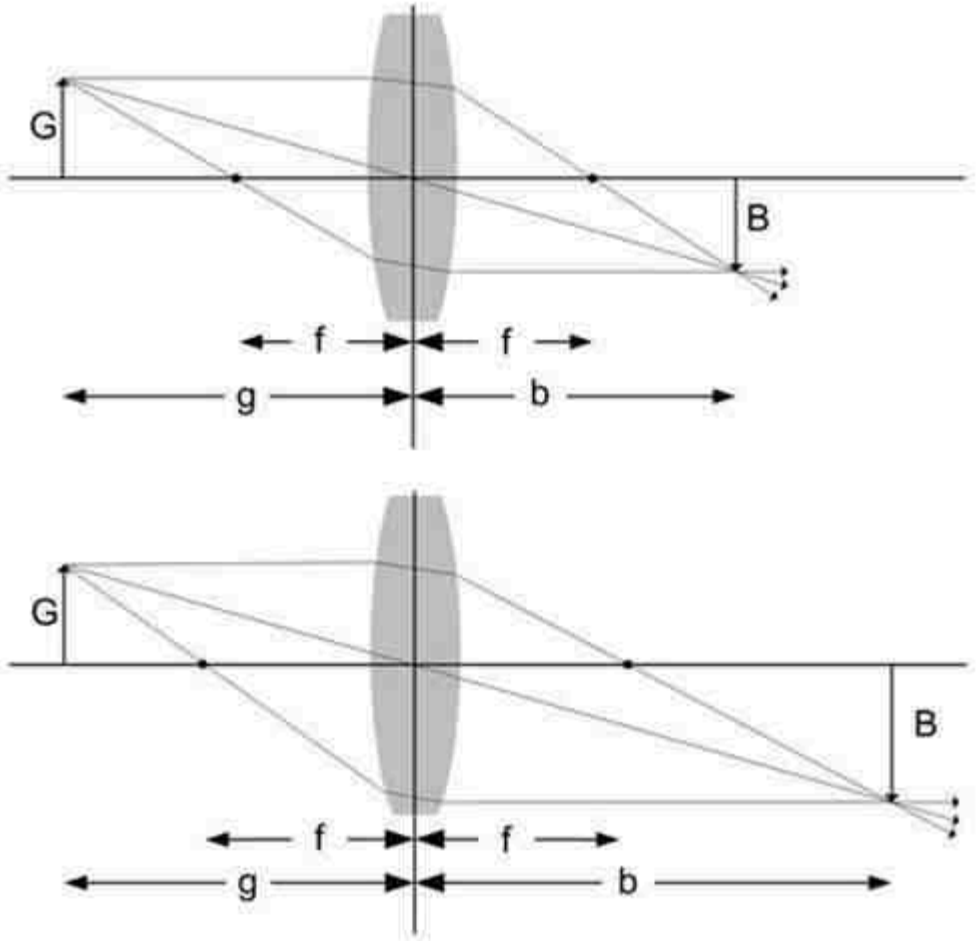


Fig. 2.7 Different focal lengths results in optical zoom

Fig. 2.8 A focused image (left) and an unfocused image (right). The difference between the two images is different values of b

A smaller depth-of-field can be achieved by increasing the focal length. However, this has the consequence that the area of the world observable to the camera is reduced. The observable area is expressed by the angle V in Fig. 2.10 and denoted the field-of-view of the camera. The field-of-view depends, besides the focal length, also on the physical size of the image sensor. Often the sensor is rectangular rather than square and from this follows that a camera has a field-of-view in both the horizontal and vertical direction denoted FOVx and FOVy, respectively. Based on right-angled triangles, these are calculated as

$$\begin{aligned} \mathrm{FOV}_x &= 2 \cdot \tan^{-1}\left(\frac{\text{width of sensor}/2}{f}\right) \\ \mathrm{FOV}_y &= 2 \cdot \tan^{-1}\left(\frac{\text{height of sensor}/2}{f}\right) \end{aligned} \tag{2.4}$$

where the focal length, f, and width and height are measured in mm.

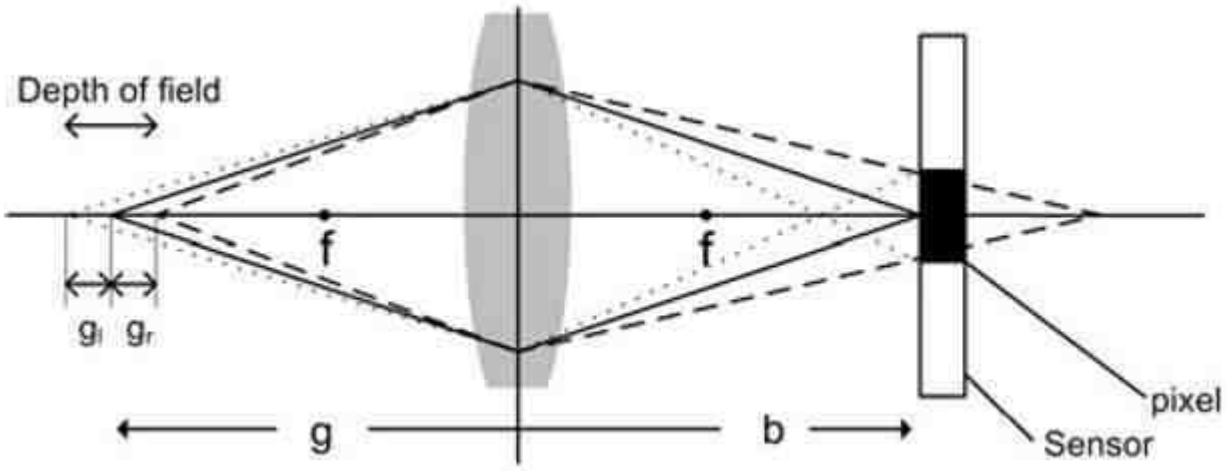


Fig. 2.9 Depth-of-field. The solid lines illustrate two light rays from an object (a point) on the optical axis and their paths through the lens and to the sensor where they intersect within the same pixel (illustrated as a black rectangle). The dashed and dotted lines illustrate light rays from two other objects (points) on the optical axis. These objects are characterized by being the most extreme locations where the light rays still enter the same pixel

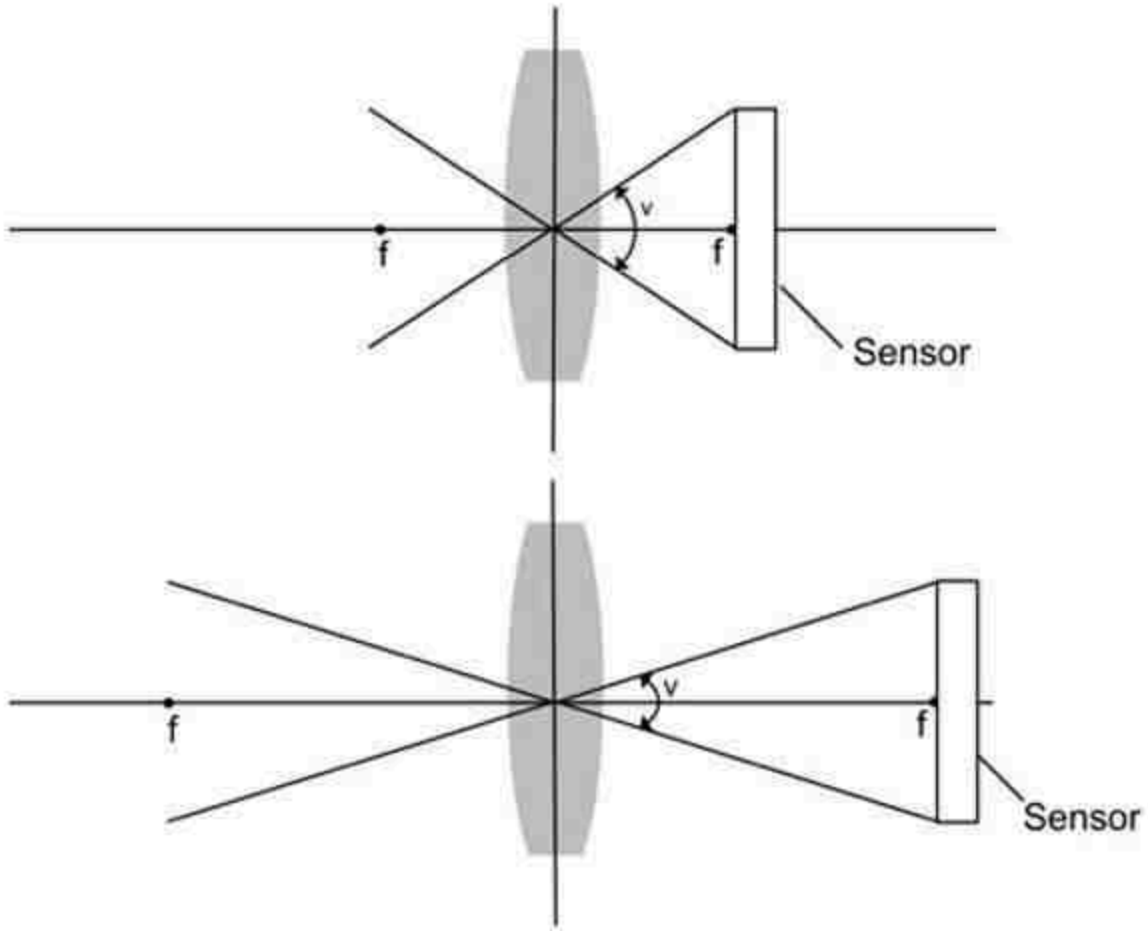


Fig. 2.10 The field-of-view of two cameras with different focal lengths. The field-of-view is an angle, V, which represents the part of the world observable to the camera. As the focal length increases so does the distance from the lens to the sensor. This in turn results in a smaller field-of-view. Note that both a horizontal field-of-view and a vertical field-of-view exist. If the sensor has equal height and width these two fields-of-view are the same, otherwise they are different

So, if we have a physical sensor with width = 14 mm, height = 10 mm and a focal length = 5 mm, then the fields-of-view will be

$$\mathrm{FOV}_x = 2 \cdot \tan^{-1}\left(\frac{7}{5}\right) = 108.9^\circ, \qquad \mathrm{FOV}_y = 2 \cdot \tan^{-1}(1) = 90^\circ \qquad (2.5)$$

Another parameter influencing the depth-of-field is the aperture. The aperture corresponds to the human iris, which controls the amount of light entering the human eye. Similarly, the

aperture is a flat circular object with a hole in the center with adjustable radius. The aperture is located in front of the lens and used to control the amount of incoming light. In the extreme case, the aperture only allows rays through the optical center, resulting in an infinite depth-of-field. The downside is that the more light blocked by the aperture, the lower shutter speed (explained below) is required in order to ensure enough light to create an image. From this it follows that objects in motion can result in blurry images.

Fig. 2.11 Three different camera settings resulting in three different depth-of-fields

To sum up, the following interconnected issues must be considered: distance to object, motion of object, zoom, focus, depth-of-field, focal length, shutter, aperture, and sensor. In Figs. 2.11 and 2.12 some of these issues are illustrated. With this knowledge you might be able to appreciate why a professional photographer can capture better images than you can!

Image Acquisition in Digital Image Processing – Buzztech

In image processing, it is defined as the action of retrieving an image from some source, usually a hardware-based source for processing. It is the first step in the workflow sequence because, without an image, no processing is possible. The image that is acquired is completely unprocessed.

Now the incoming energy is transformed into a voltage by the combination of input electrical power and sensor material that is responsive to a particular type of energy being detected. The output voltage waveform is the response of the sensor(s) and a digital quantity is obtained from each sensor by digitizing its response.

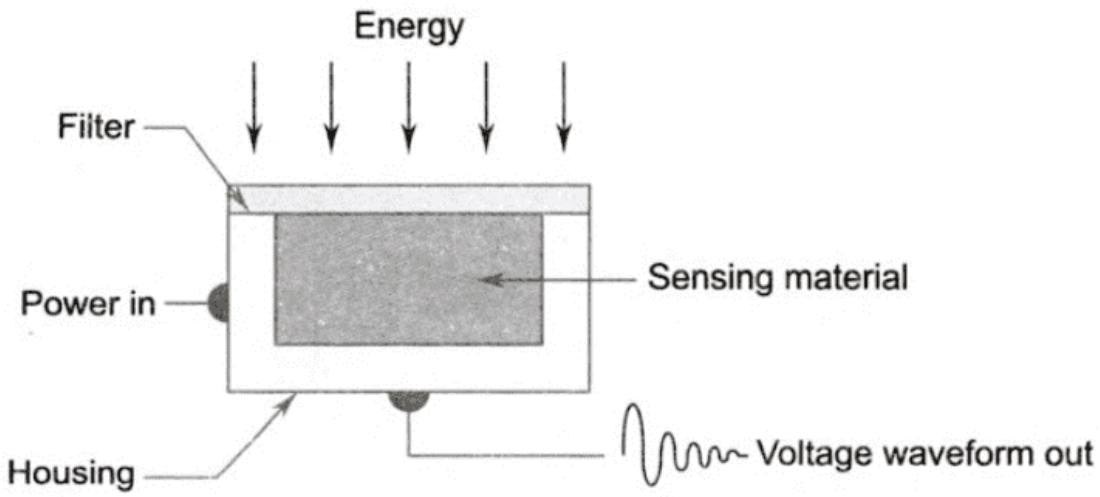


Fig: Single image sensor

Fig: Line sensor

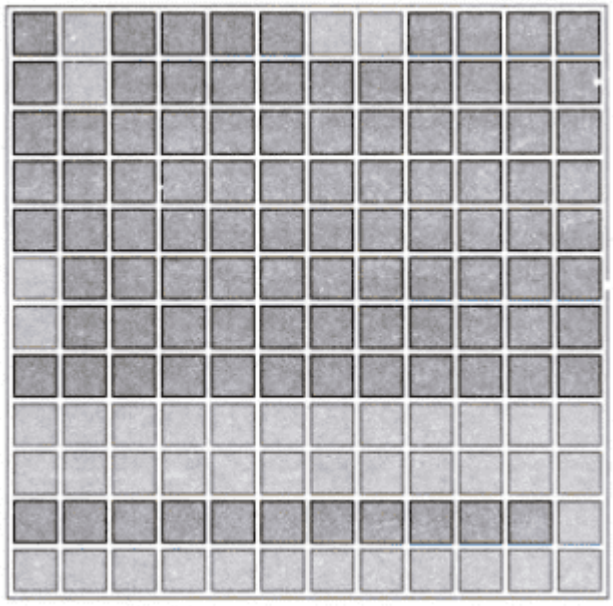


Fig: Array sensor

Image Acquisition using a single sensor:

Example of a single sensor is a photodiode. Now to obtain a two-dimensional image using a single sensor, the motion should be in both x and y directions.

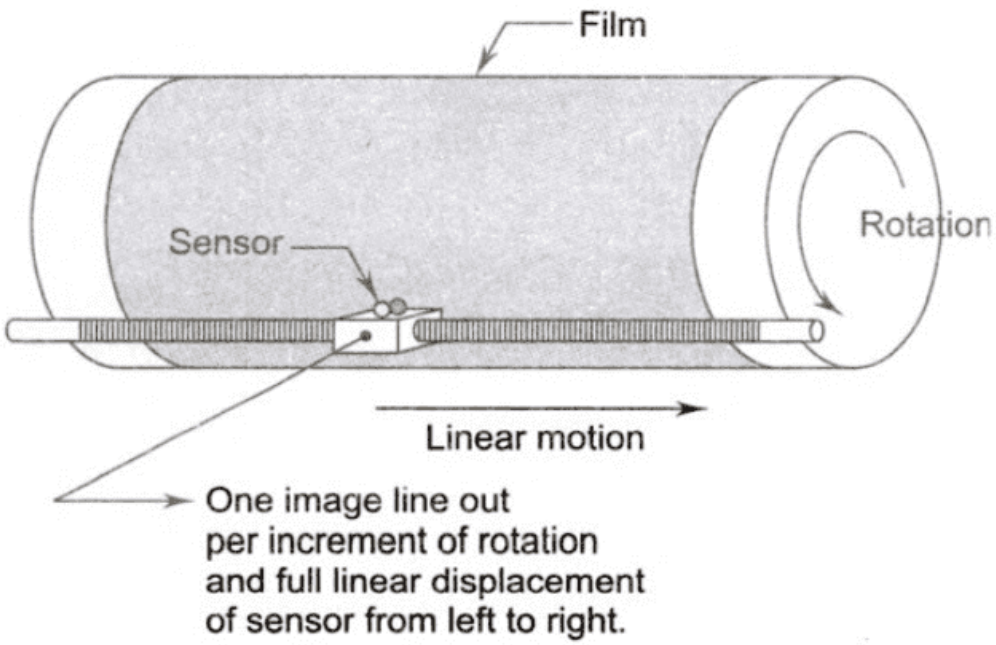


Fig:

Combining a single sensor with motion to generate a 2D image

This is an inexpensive method and we can obtain high-resolution images with high precision control. But the downside of this method is that it is slow.

Image Acquisition using a line sensor (sensor strips):

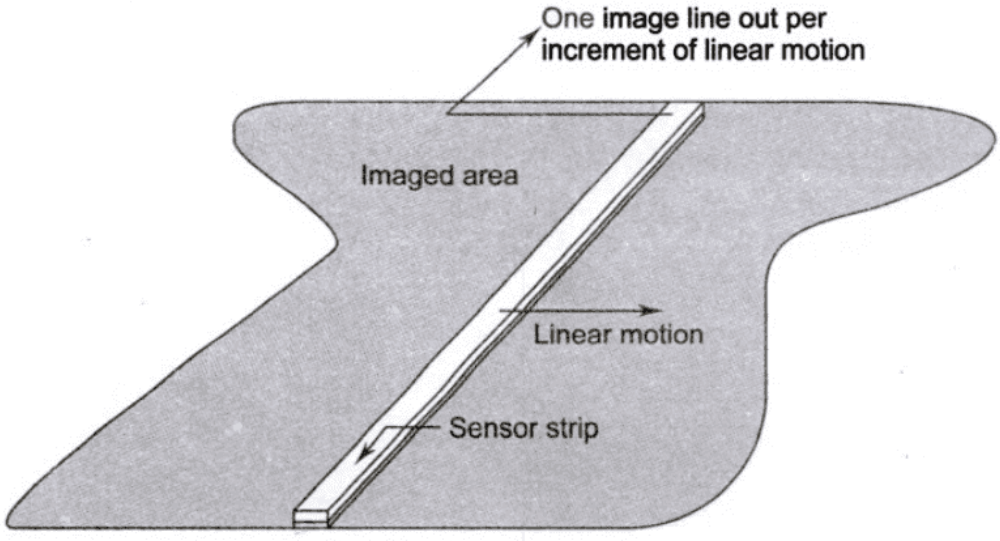


Fig:

Linear sensor strip

Image Acquisition using an array sensor:

In this, individual sensors are arranged in the form of a 2-D array. This type of arrangement is found in digital cameras. e.g. CCD array

In this, the response of each sensor is proportional to the integral of the light energy projected onto the surface of the sensor. Noise reduction is achieved by letting the sensor integrate the input light signal over minutes or ever hours.

Advantage: Since sensor array is 2D, a complete image can be obtained by focusing the energy pattern onto the surface of the array.

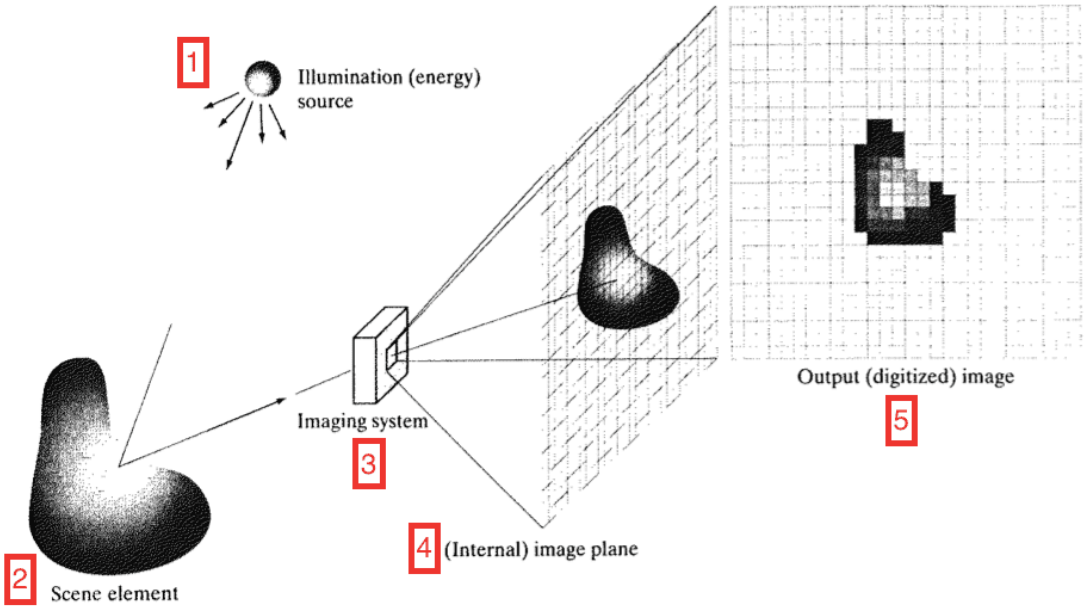


Fig: An example of digital image acquisition using array sensor

The sensor array is coincident with the focal plane, it produces an output proportional to the integral of light received at each sensor.

Digital and analog circuitry sweep these outputs and convert them to a video signal which is then digitized by another section of the imaging system. The output is a digital image.

Need of Sampling and Quantization in Digital Image Processing:

Mostly the output of image sensors is in the form of analog signal. Now the problem is that we cannot apply digital image processing and its techniques on analog signals.

This is due to the fact that we cannot store the output of image sensors which are in the form of analog signals because it requires infinite memory to store a signal that can have infinite values. So we have to convert this analog signal into digital signal.

To create a digital image, we need to convert the continuous data into digital form. This conversion from analog to digital involves two processes: sampling and quantization.

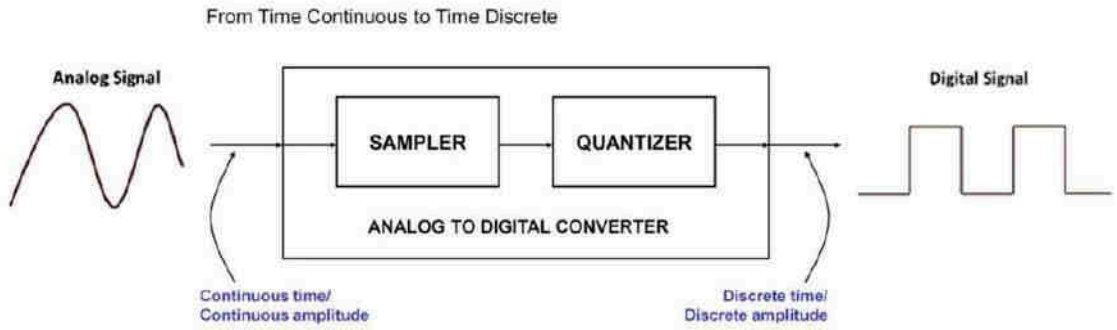


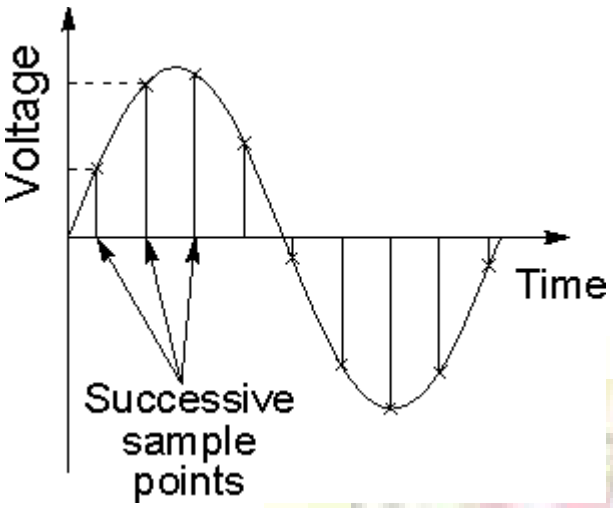
Fig: Analog to Digital Conversion

Sampling -> digitization of coordinate values

Quantization -> digitization of amplitude values

Sampling in Digital Image Processing:

- For e.g. if $y = sinx$, it is done on x variable.

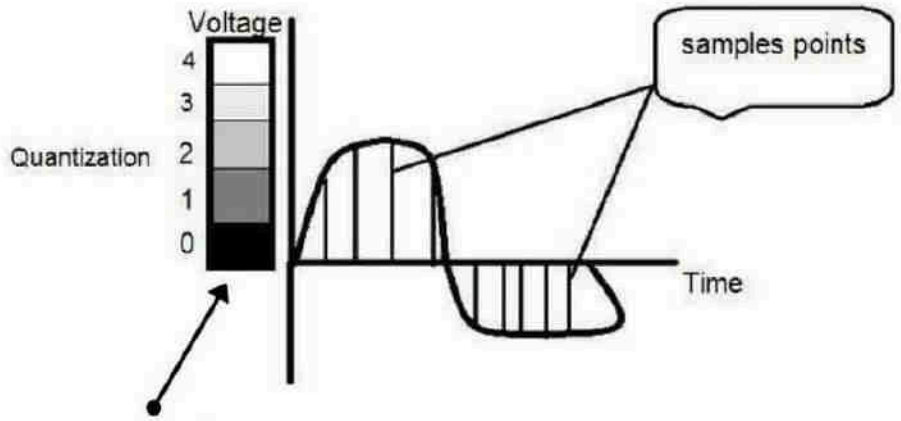


Total number of pixels = Total number of rows X Total number of columns

- ***For a CCD array,*** if the number of sensors on a CCD array is equal to the number of pixels and number of pixels is equal to the number of samples taken, therefore we can say that number of samples taken is equal to the number of sensors on a CCD array.

No. of sensors on a CCD array = No. of pixels = No. of samples taken

Quantization in Digital Image Processing:



Vertically ranging values have been quantized into 5 different levels or partitions.

Relation of Quantization and gray level resolution:

Number of quantas (partitions) = Number of gray levels

Where,

L = gray level resolution

k = gray level

Gray level = number of bits per pixel (BPP) = number of levels per pixel

Basic Relationships Between

Pixels

- ***Neighborhood***
- ***Adjacency***
- ***Connectivity***
- ***Paths***
- ***Regions and boundaries***

Neighbors of a Pixel

• Any pixel p(x, y) has two vertical and two horizontal neighbors, given by

(x+1, y), (x-1, y), (x, y+1), (x, y-1)

• This set of pixels are called the 4-neighbors of P, and is denoted by N4(P).

• Each of them are at a unit distance from P.

An image is denoted by f(x,y) and p,q are used to represent individual pixels of the image.

Neighbours of a pixel

A pixel p at (x,y) has 4-horizontal/vertical neighbours at (x+1,y), (x-1,y), (x,y+1) and (x,y-1). These are called the **4-neighbours of p : N4(p)**.

A pixel p at (x,y) has 4 diagonal neighbours at (x+1,y+1), (x+1,y-1), (x-1,y+1) and (x-1,y-1). These are called the **diagonal-neighbours of p : ND(p).**

The 4-neighbours and the diagonal neighbours of p are called **8-neighbours of p : N8(p).**

Adjacency between pixels

Let **V** be the set of intensity values used to define adjacency.

In a binary image, V ={1} if we are referring to adjacency of pixels with value 1. In a gray-scale image, the idea is the same, but set V typically contains more elements.

For example, in the adjacency of pixels with a range of possible intensity values 0 to 255, set V could be any subset of these 256 values.

We consider three types of adjacency:

a) 4-adjacency: Two pixels p and q with values from V are 4-adjacent if q is in the set N4(p).

b) 8-adjacency: Two pixels p and q with values from V are 8-adjacent if q is in the set N8(p).

c) m-adjacency(mixed adjacency): Two pixels p and q with values from V are m-adjacent if

1. q is in N4(p), or
2. 2) q is in ND(p) and the set N4(p)∩N4(q) has no pixels whose values are from V.

Connectivity between pixels

It is an important concept in digital image processing.

It is used for establishing boundaries of objects and components of regions in an image.

Two pixels are said to be connected:

- if they are adjacent in some sense(neighbour pixels,4/8/m-adjacency)
- if their gray levels satisfy a specified criterion of similarity(equal intensity level)

There are three types of connectivity on the basis of adjacency. They are:

a) 4-connectivity: Two or more pixels are said to be 4-connected if they are 4-adjacent with each others.

b) 8-connectivity: Two or more pixels are said to be 8-connected if they are 8-adjacent with each others.

c) m-connectivity: Two or more pixels are said to be m-connected if they are m-adjacent with each others.

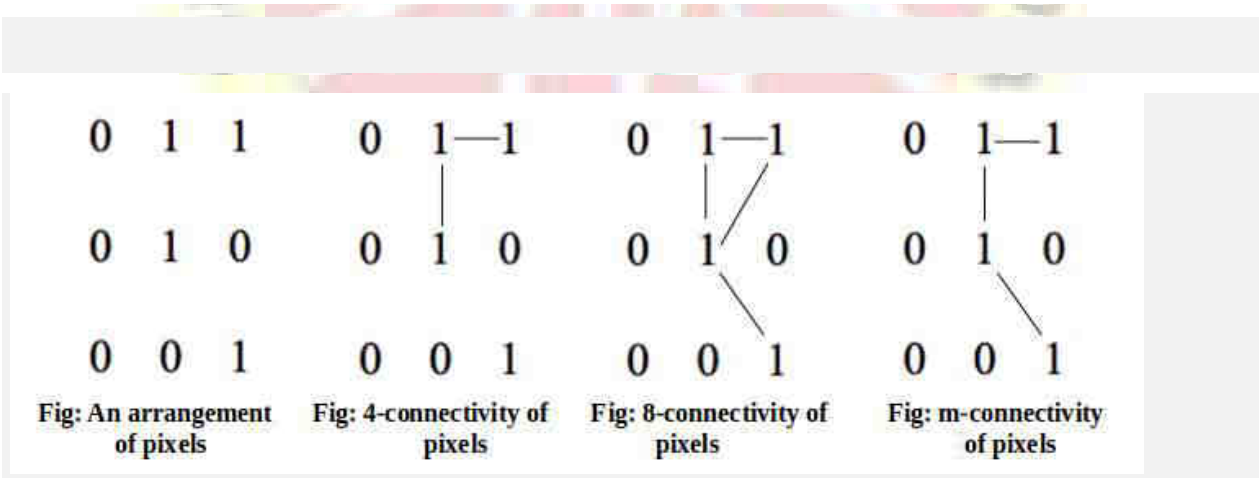


Fig: An arrangement of pixels | Fig: 4-connectivity of pixels | Fig: 8-connectivity of pixels | Fig: m-connectivity of pixels

Color Models

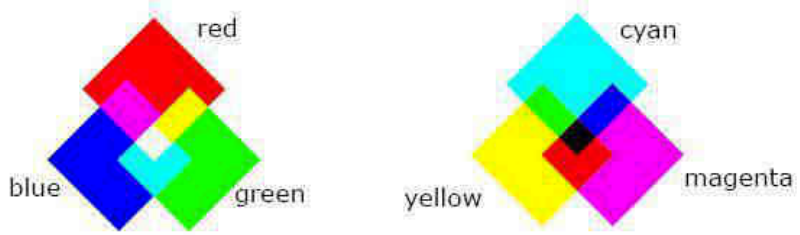
The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, the RGB color model is used in computer graphics, YUV or YCbCr are used in video systems, PhotoYCC* is used in PhotoCD* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

RGB Color Model

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure "Primary and Secondary Colors for RGB and CMYK Models") - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

Primary and Secondary Colors for RGB and CMYK Models



a) RGB model b) CMY(K) model

The color subspace of interest is a cube shown in Figure "RGB and CMY Color Models" (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

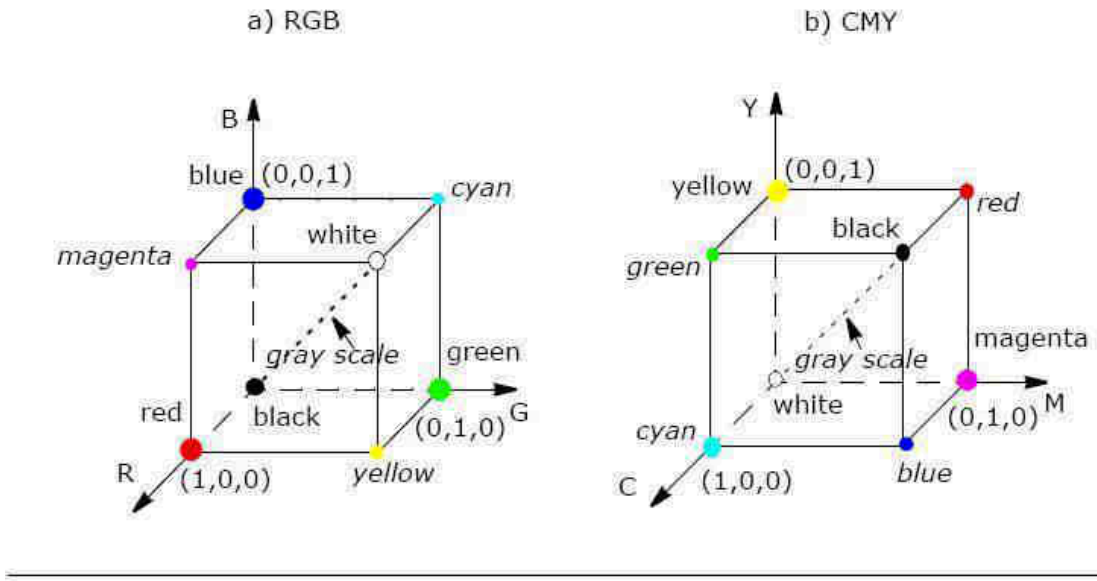
The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, the Intel IPP color conversion functions operate with non-linear gamma-corrected images R'G'B'.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.

RGB and CMY Color Models



However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see Figure "Primary and Secondary Colors for RGB and CMYK Models"), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY. The CMY cube is shown in Figure "RGB and CMY Color Models", in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is “derived” from the RGB space. It comprises the *luminance* (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or *chrominance*, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in the YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, because the possible RGB colors occupy only part of the YUV space limited by these ranges. Figure "RGB Colors Cube in the YUV Color Space" shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values that are normalized to [0..1]).

The Y'U'V' notation means that the components are derived from gamma-corrected R'G'B'. Weighted sum of these non-linear components forms a signal representative of luminance that is called *luma* Y'. (*Luma* is often loosely referred to as *luminance*, so you need to be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term *luminance*).

The Intel IPP functions use the following basic equation ([Jack01]) to convert between gamma-corrected R'G'B' and Y'U'V' models:

Y'= 0.299*R' + 0.587*G' + 0.114*B'

U'= -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'- Y')

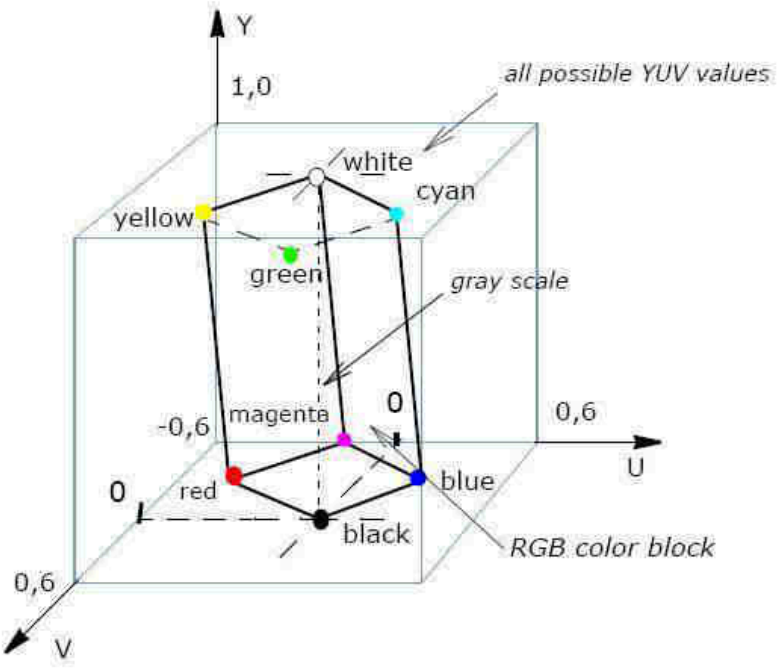
V'= 0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'- Y')

R' = Y' + 1.140*V'

G' = Y' - 0.394*U' - 0.581*V'

B' = Y' + 2.032*U'

RGB Colors Cube in the YUV Color Space



There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described in Image Downsampling.

YCbCr and YCCK Color Models

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations [Jack01] to convert between R'G'B' in the range 0-255 and Y'Cb'Cr' (this notation means that all components are derived from gamma-corrected R'G'B'):

Y' = 0.257*R' + 0.504*G' + 0.098*B' + 16

Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128

Cr' = 0.439*R' - 0.368*G' - 0.071*B' + 128

R' = 1.164*(Y'-16) + 1.596*(Cr'-128)

G' = 1.164*(Y'-16) - 0.813*(Cr'-128) - 0.392*(Cb'-128)

B' = 1.164*(Y'-16) + 2.017*(Cb'-128)

The Intel IPP color conversion functions specific for the JPEG codec use different equations:

Y = 0.299*R + 0.587*G + 0.114*B

Cb = -0.16874*R - 0.33126*G + 0.5*B + 128

Cr = 0.5*R - 0.41869*G - 0.08131*B + 128

R = Y + 1.402*Cr - 179,456

G = Y - 0.34414*Cb - 0.71414*Cr + 135.45984

B = Y + 1.772*Cb - 226.816

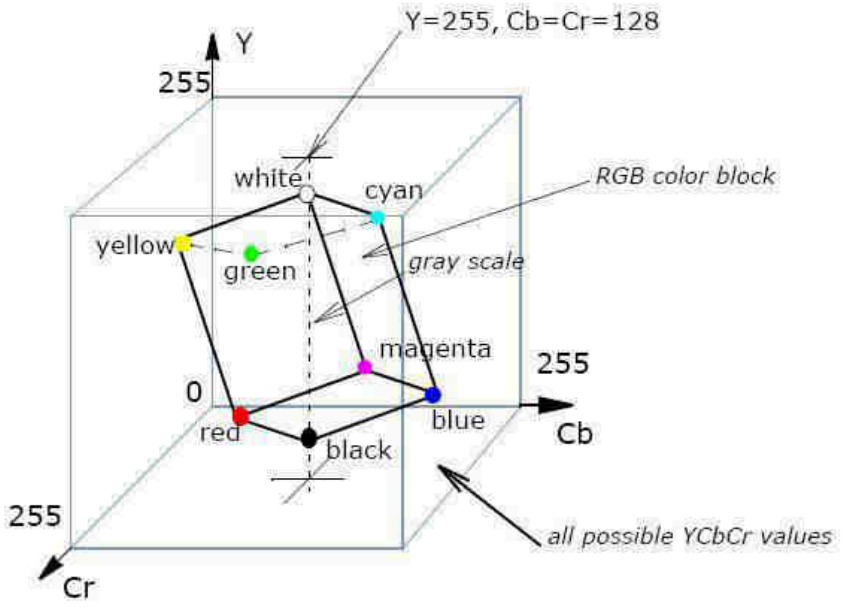
YCCK model is specific for the JPEG image compression. It is a variant of the YCbCr model containing an additional K channel (black). The fact is that JPEG codec performs more effectively if the luminance and color information are decoupled. Therefore, a CMYK image

must be converted to YCCK before JPEG compression (see description of the function ippiCMYKToYCCK_JPEG for more details).

Possible RGB colors occupy only part of the YCbCr color space (see Figure "RGB Colors Cube in the YCbCr Space") limited by the nominal ranges, therefore there are many YCbCr combinations that result in invalid RGB values.

There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described in Image Downsampling.

RGB Colors Cube in the YCbCr Space



PhotoYCC Color Model

The Kodak* PhotoYCC* was developed for encoding Photo CD* image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 ([ITU709]). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components. The PhotoYCC is optimized for the color

photographic material, and provides a color gamut that is greater than the one that can currently be displayed.

The Intel IPP functions use the following basic equations [Jack01] to convert non-linear gamma-corrected R'G'B' to Y'C'C':

Y' = 0.213*R' + 0.419*G' + 0.081*B'

C1' = -0.131*R' - 0.256*G' + 0.387*B'+ 0.612

C2' = 0.373*R' - 0.312*R' - 0.061*B' + 0.537

The equations above are given on the assumption that R',G', and B' values are normalized to the range [0..1].

Since the PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations [Jack01] are used in Intel IPP to generate R'G'B' values for driving a CRT display and require a unity relationship between the luma in the encoded image and the displayed image:

R' = 0.981 * Y + 1.315 * (C2 - 0.537)

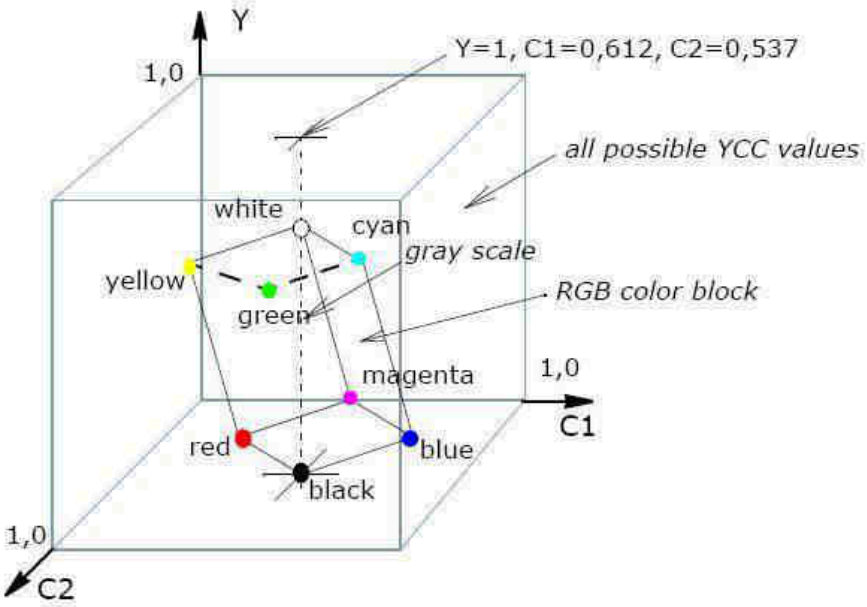
G' = 0.981 * Y - 0.311 * (C1 - 0.612)- 0.669 * (C2 - 0.537)

B' = 0.981 * Y + 1.601 * (C1 - 0.612)

The equations above are given on the assumption that source Y, C1 and C2 values are normalized to the range [0..1], and the display primaries have the chromaticity values in accordance with [ITU709] specifications.

The possible RGB colors occupy only part of the YCC color space (see Figure "RGB Colors in the YCC Color Space") limited by the nominal ranges, therefore there are many YCC combinations that result in invalid RGB values.

RGB Colors in the YCC Color Space



YCoCg Color Models

The YCoCg color model was developed to increase the effectiveness of the image compression [Malvar03]. This color model comprises the luminance (Y) and two color difference components (Co - offset orange, Cg - offset green).

The Intel IPP functions use the following simple basic equations [Malvar03] to convert between RGB and YCoCg:

$Y = R/4 + G/2 + B/4$

$Co = R/2 - B/2$

$Cg = -R/4 + G/2 - B/4$

$R = Y + Co - Cg$

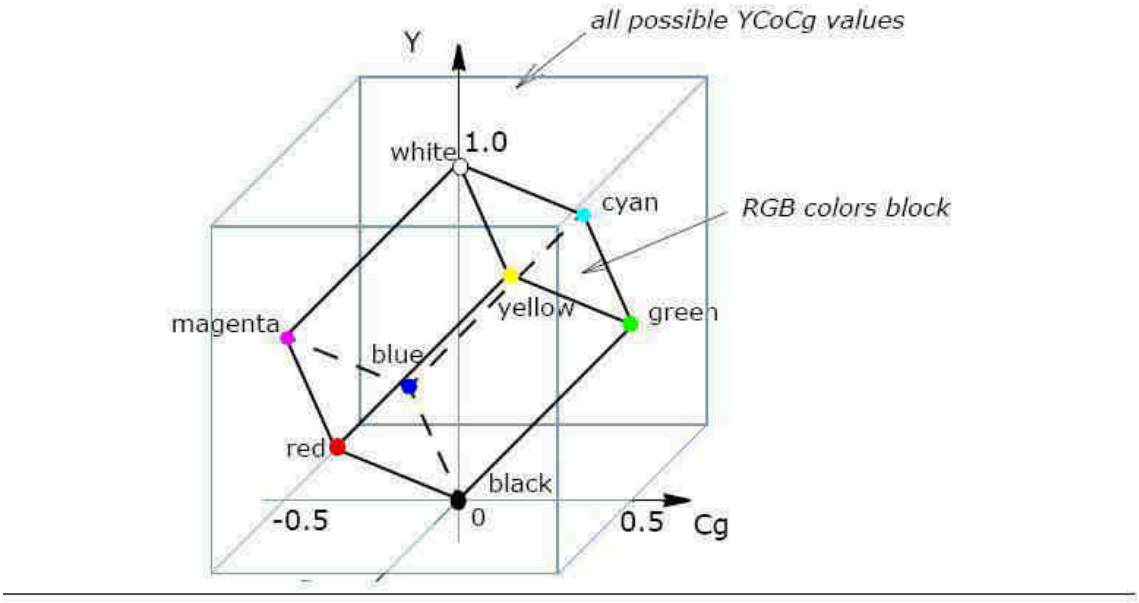
$G = Y + Cg$

$B = Y - Co - Cg$

A variation of this color space which is called YCoCg-R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg [Malvar03-1].

The possible RGB colors occupy only part of the YCoCg color space (see Figure "RGB Color Cube in the YCoCg Color Space") limited by the nominal ranges, therfore there are many YCoCg combinations that result in invalid RGB values.

RGB Color Cube in the YCoCg Color Space



HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more “intuitive” in manipulating with color and were designed to approximate the way humans perceive and interpret color.

Hue defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

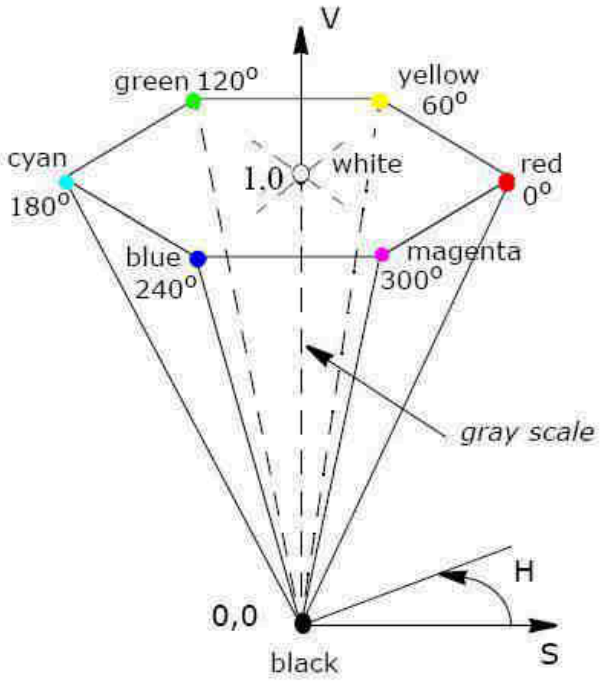
Saturation indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue (S=1) is at *value* V=1 (full illumination) in the HSV color model, and at *lightness* L=0.5 in the HLS color model.

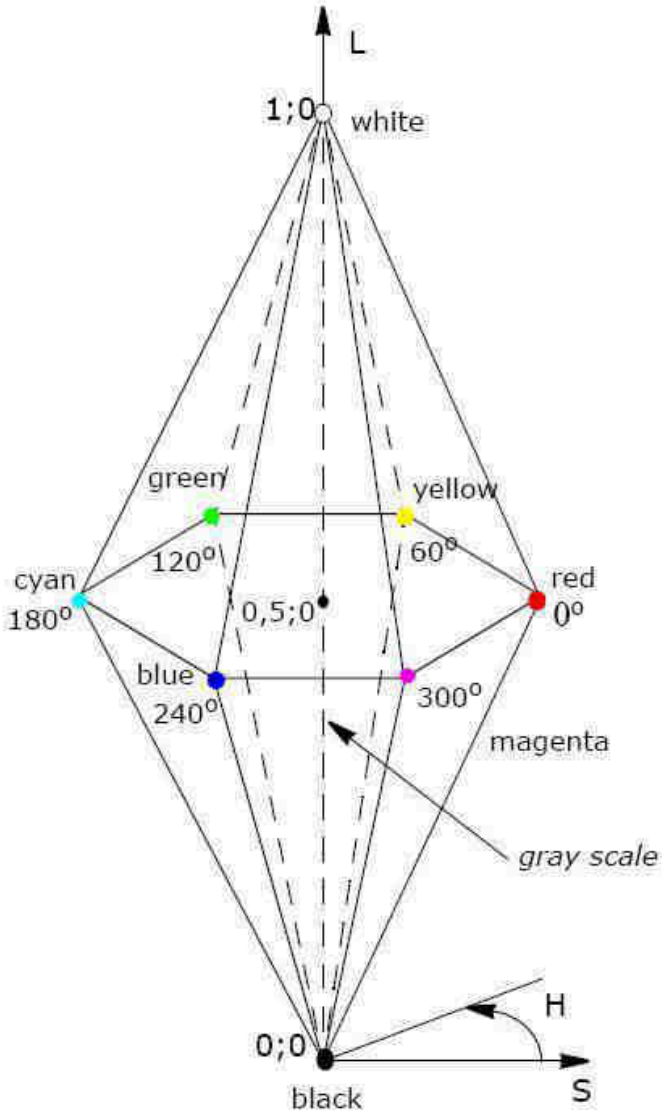
The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the Figure "HSV Solid", because the hexcone defines the subset of the HSV space with valid RGB values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone (Figure "HSV Solid") with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation. Conversions from RGBToHSV/RGBToHSV and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [Rogers85] given in the descriptions of corresponding conversion functions.

HSV Solid


V
green 120°
yellow
60°
cyan
180°
1.0
white
red
0°
blue
240°
magenta
300°
gray scale
H
0,0
black
S

HLS Solid



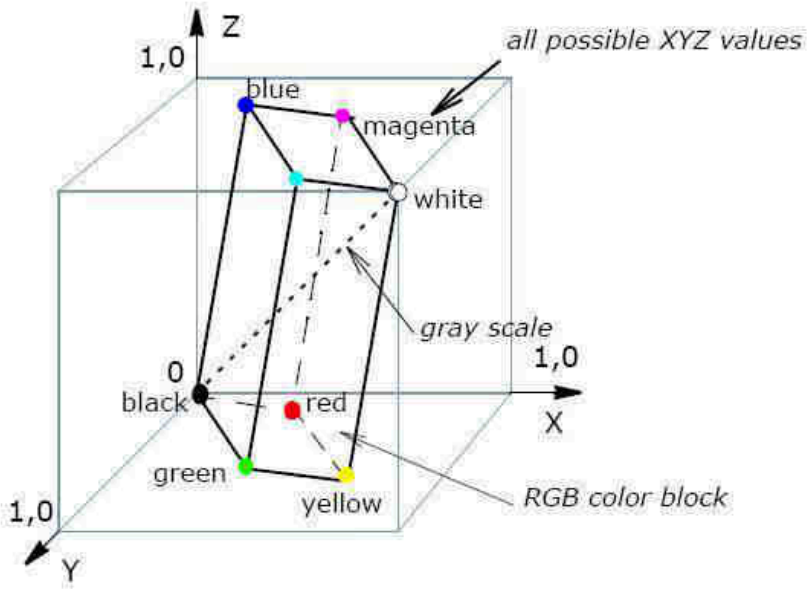
CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information. The main advantage of the CIE XYZ space (and any color space based on it)

is that this space is completely device-independent. The chromaticity diagram in Figure "CIE *xyY* Chromaticity Diagram and Color Gamut" is in fact a two-dimensional projection of the CIE XYZ sub-space. Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a "color" outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in Figure "RGB Colors Cube in the XYZ Color Space".

RGB Colors Cube in the XYZ Color Space



Intel IPP functions use the following basic equations [Rogers85], to convert between gamma-corrected R'G'B' and CIE XYZ models:

X = 0.412453*R' + 0.35758 *G' + 0.180423*B'

Y = 0.212671*R' + 0.71516 *G' + 0.072169*B'

Z = 0.019334*R' + 0.119193*G' + 0.950227*B'

The equations for X,Y,Z calculation are given on the assumption that R',G', and B' values are normalized to the range [0..1].

R' = 3.240479 * X - 1.53715 * Y - 0.498535 * Z

G' = -0.969256 * X + 1.875991 * Y + 0.041556 * Z

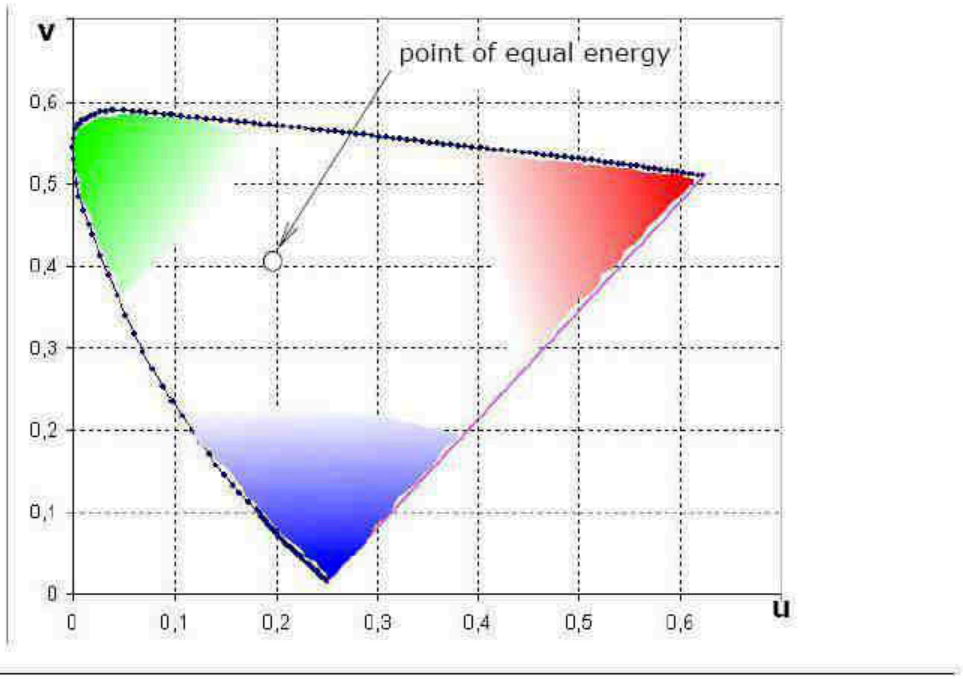
B' = 0.055648 * X - 0.204043 * Y + 1.057311 * Z

The equations for R',G', and B' calculation are given on the assumption that X,Y, and Z values are in the range [0..1].

CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. "Perceptually uniform" means that two colors that are equally distant in the color space are equally distant perceptually. To accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE (Figure "CIE *u'*,*v'* Uniform Chromaticity Scale Diagram"). The UCS diagram uses a mathematical formula to transform the XYZ values or x, y coordinates (Figure "CIE *xyY* Chromaticity Diagram and Color Gamut"), to a new set of values that present a visually more accurate two-dimensional model. The Y lightness scale is replaced with a new scale called L that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are U and V for CIE LUV, and *a* and *b* (referred to also respectively as red/blue and yellow/blue chrominances) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

CIE *u'*,*v'* Uniform Chromaticity Scale Diagram



The CIE LUV color space is derived from CIE XYZ as follows ([Rogers85]),

$L = 116. * (Y/Y_n)^{1/3.} - 16.$

$U = 13. * L * (u - u_n)$

$V = 13. * L * (v - v_n)$

where

$u = 4.*X / (X + 15.*Y + 3.*Z)$

$v = 9.*Y / (X + 15.*Y + 3.*Z)$

$u_n = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$

$v_n = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$

Inverse conversion is performed in accordance with equations:

$Y = Y_n * ((L + 16.) / 116.)^{3.}$

$X = -9.* Y * u / ((u - 4.)* v - u * v)$

$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$

where

$u = U / (13.* L) + u_n$

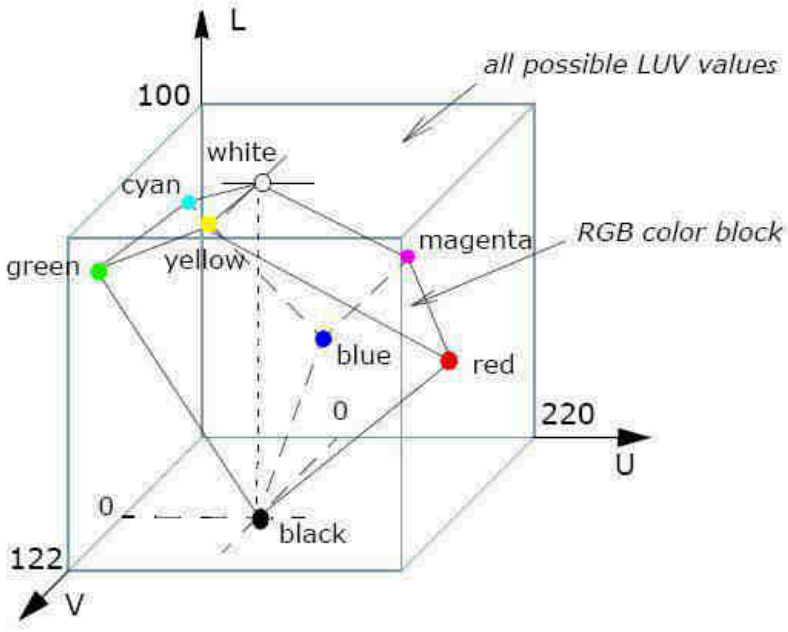
$v = V / (13.* L) + v_n$

and u_n, v_n are defined above.

Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point ([ITU709]), and $Y_n = 1.0$ is the luminance of the D65 white point. The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see Figure "RGB Color Cube in the CIE LUV Color Space") limited by the nominal ranges, therefore there are many LUV combinations that result in invalid RGB values.

RGB Color Cube in the CIE LUV Color Space



The CIE Lab color space is derived from CIE XYZ as follows:

$L = 116. * (Y/Y_n)^{1/3.} - 16$ for $Y/Y_n > 0.008856$

$L = 903.3 * (Y/Y_n)^{1/3.}$ for $Y/Y_n \leqslant 0.008856$

$a = 500. * [f(X/X_n)-f(Y/Y_n)]$

$b = 200. * [f(Y/Y_n)-f(Z/Z_n)]$

where

$f(t) = t^{1/3.} - 16$ for $t > 0.008856$

$f(t) = 7.787*t + 16/116$ for $t \leqslant 0.008856$

Here $Y_n = 1.0$ is the luminance, and $X_n = 0.950455$, $Z_n = 1.088753$ are the chrominances for the D65 white point.

The values of the L component are in the range [0..100], *a* and *b* component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$Y = Y_n * P^{3.}$

$X = X_n * (P + a/500.)^{3.}$

$Z = Z_n * (P - b/200.)^{3.}$

where

$P = (L +16)/116.$

UNIT II

Image enhancement

The aim of image enhancement is to improve the interpretability or perception of information in images for human viewers, or to provide `better' input for other automated image processing techniques.

Image enhancement techniques can be divided into two broad categories:

1.Spatial domain methods, which operate directly on pixels, and

2.frequency domain methods, which operate on the Fourier transform of an image.

Unfortunately, there is no general theory for determining what is `good' image enhancement when it comes to human perception. If it looks good, it is good! However, when image enhancement techniques are used as pre-processing tools for other image processing techniques, then quantitative measures can determine which techniques are most appropriate.

Spatial domain methods

The value of a pixel with coordinates (x,y) in the enhanced image $\hat{F}$ is the result of performing some operation on the pixels in the neighbourhood of (x,y) in the input image, F.

Neighbourhoods can be any shape, but usually they are rectangular.

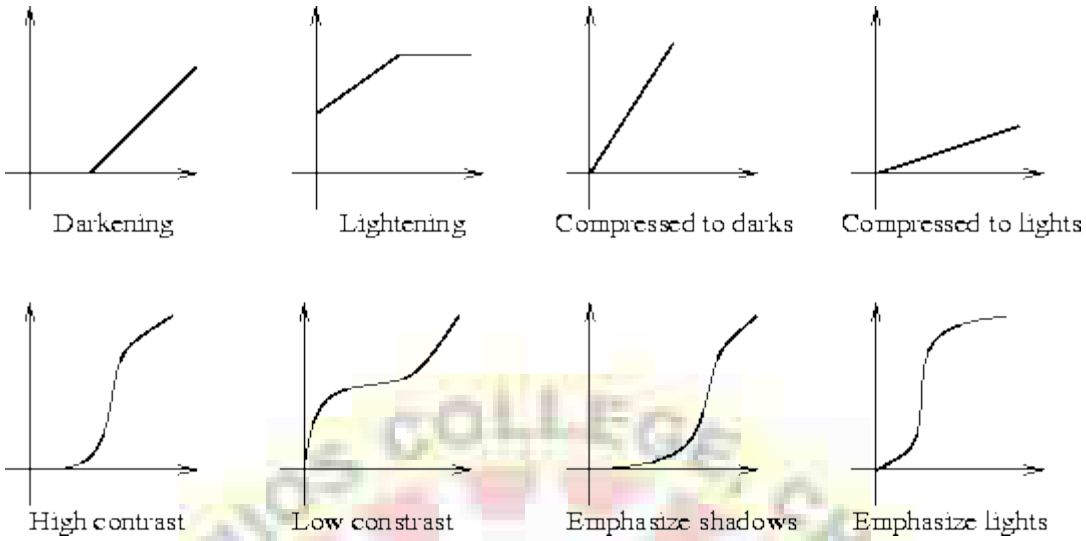
Grey scale manipulation

The simplest form of operation is when the operator T only acts on a 1×1 pixel neighbourhood in the input image, that is $\hat{F}(x,y)$ only depends on the value of F at (x,y). This is a *grey scale transformation* or mapping.

The simplest case is thresholding where the intensity profile is replaced by a step function, active at a chosen threshold value. In this case any pixel with a grey level below the threshold in the input image gets mapped to 0 in the output image. Other pixels are mapped to 255.

Other grey scale transformations are outlined in figure 1 below.

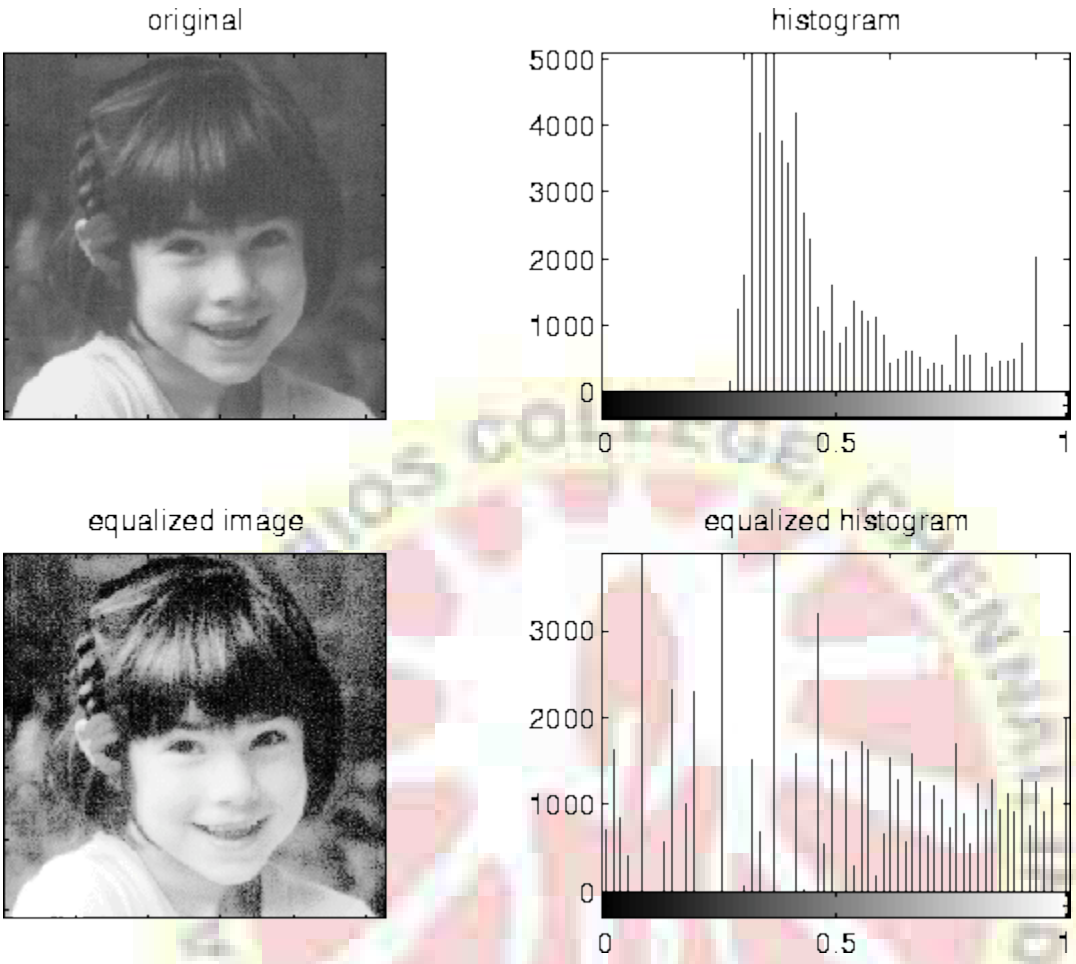
Figure 1: Tone-scale adjustments.



Histogram Equalization

Histogram equalization is a common technique for enhancing the appearance of images. Suppose we have an image which is predominantly dark. Then its histogram would be skewed towards the lower end of the grey scale and all the image detail is compressed into the dark end of the histogram. If we could `stretch out' the grey levels at the dark end to produce a more uniformly distributed histogram then the image would become much clearer.

Figure 2: The original image and its histogram, and the equalized versions. Both images are quantized to 64 grey levels.



Histogram equalization involves finding a grey scale transformation function that creates an output image with a *uniform histogram* (or nearly so).

How do we determine this grey scale transformation function? Assume our grey levels are continuous and have been normalized to lie between 0 and 1.

We must find a transformation T that maps grey values r in the input image F to grey values $s = T(r)$ in the transformed image $\hat{F}$.

It is assumed that

- T is single valued and monotonically increasing, and
- $0 \leq T(r) \leq 1$ for $0 \leq r \leq 1$.

The inverse transformation from s to r is given by

$r = T^{-1}(s)$.

If one takes the histogram for the input image and normalizes it so that the area under the histogram is 1, we have a probability distribution for grey levels in the input image $P_r(r)$.

If we transform the input image to get $s = T(r)$ what is the probability distribution $P_s(s)$?

From probability theory it turns out that

$$P_s(s) = P_r(r)\frac{dr}{ds},$$

where $r = T^{-1}(s)$.

Consider the transformation

$$s = T(r) = \int_0^r P_r(w)dw.$$

This is the cumulative distribution function of r. Using this definition of T we see that the derivative of s with respect to r is

$$\frac{ds}{dr} = P_r(r).$$

Substituting this back into the expression for P_s, we get

$$P_s(s) = P_r(r)\frac{1}{P_r(r)} = 1$$

for all $s, where 0 \leq s \leq 1$ .Thus, $P_s(s)$ is now a uniform distribution function, which is what we want.

Discrete Formulation

We first need to determine the probability distribution of grey levels in the input image. Now

$$P_r(r) = \frac{n_k}{N}$$

where n_k is the number of pixels having grey level k, and N is the total number of pixels in the image.

The transformation now becomes

$$s_k = T(r_k) = \sum_{i=0}^{k} \frac{n_i}{N}$$
$$= \sum_{i=0}^{k} P_r(r_i).$$

Note that $0 \leq r_k \leq 1$, the index $k = 0, 1, 2, \ldots, 255$,and $0 \leq s_k \leq 1$.

The values of s_k will have to be scaled up by 255 and rounded to the nearest integer so that the output values of this transformation will range from 0 to 255. Thus the discretization and rounding of s_k to the nearest integer will mean that the transformed image will not have a perfectly uniform histogram.

Image Smoothing

The aim of image smoothing is to diminish the effects of camera noise, spurious pixel values, missing pixel values etc. There are many different techniques for image smoothing; we will consider neighbourhood averaging and edge-preserving smoothing.

Neighbourhood Averaging

Each point in the smoothed image, $\hat{F}(x,y)$ is obtained from the average pixel value in a neighbourhood of (*x*,*y*) in the input image.

For example, if we use a 3×3 neighbourhood around each pixel we would use the mask

1/9 1/9 1/9

1/9 1/9 1/9

1/9 1/9 1/9

Each pixel value is multiplied by $\frac{1}{9}$, summed, and then the result placed in the output image. This mask is successively moved across the image until every pixel has been covered. That is, the image is *convolved* with this smoothing mask (also known as a spatial filter or kernel).

However, one usually expects the value of a pixel to be more closely related to the values of pixels close to it than to those further away. This is because most points in an image are spatially coherent with their neighbours; indeed it is generally only at edge or feature points where this hypothesis is not valid. Accordingly it is usual to weight the pixels near the centre of the mask more strongly than those at the edge.

Some common weighting functions include the rectangular weighting function above (which just takes the average over the window), a triangular weighting function, or a Gaussian.

In practice one doesn't notice much difference between different weighting functions, although Gaussian smoothing is the most commonly used. Gaussian smoothing has the attribute that the frequency components of the image are modified in a smooth manner.

Smoothing reduces or attenuates the higher frequencies in the image. Mask shapes other than the Gaussian can do odd things to the frequency spectrum, but as far as the appearance of the image is concerned we usually don't notice much.

Edge preserving smoothing

Neighbourhood averaging or Gaussian smoothing will tend to blur edges because the high frequencies in the image are attenuated. An alternative approach is to use *median filtering*. Here we set the grey level to be the median of the pixel values in the neighbourhood of that pixel.

The median *m* of a set of values is such that half the values in the set are less than *m* and half are greater. For example, suppose the pixel values in a 3×3 neighbourhood are (10, 20, 20, 15, 20, 20, 20, 25, 100). If we sort the values we get (10, 15, 20, 20, |20|, 20, 20, 25, 100) and the median here is 20.

The outcome of median filtering is that pixels with outlying values are forced to become more like their neighbours, but at the same time edges are preserved. Of course, median filters are non-linear.

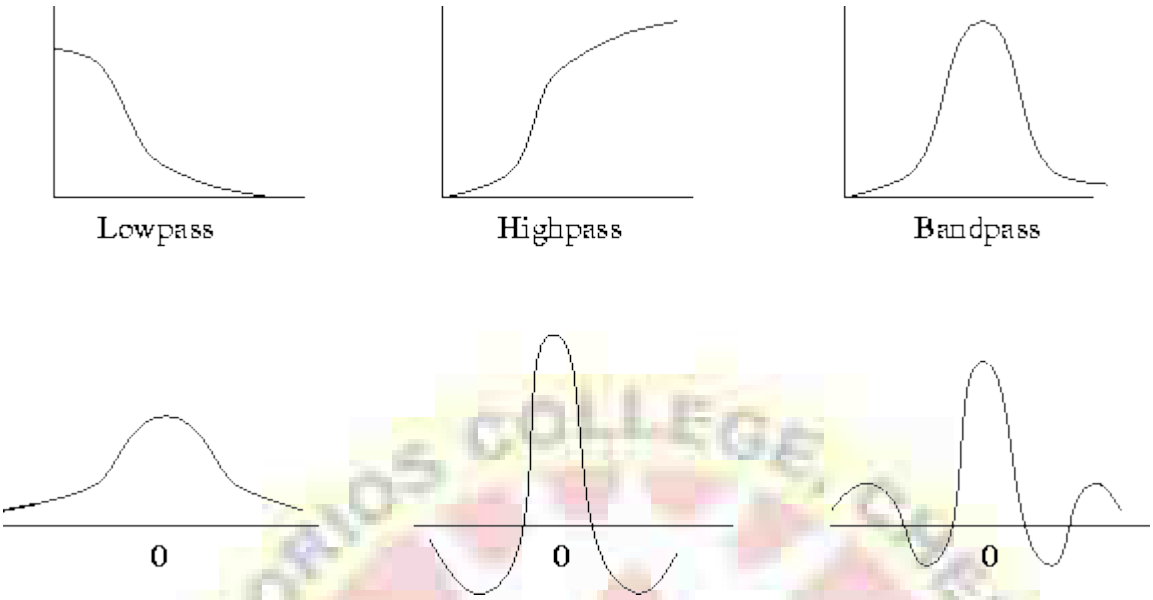
Median filtering is in fact a morphological operation. When we erode an image, pixel values are replaced with the smallest value in the neighbourhood. Dilating an image corresponds to replacing pixel values with the largest value in the neighbourhood. Median filtering replaces pixels with the median value in the neighbourhood. It is the rank of the value of the pixel used in the neighbourhood that determines the type of morphological operation.

Figure 3: Image of Genevieve; with salt and pepper noise; the result of averaging; and the result of median filtering.

Image sharpening

The main aim in image sharpening is to highlight fine detail in the image, or to enhance detail that has been blurred (perhaps due to noise or other effects, such as motion). With image sharpening, we want to enhance the high-frequency components; this implies a spatial filter shape that has a high positive component at the centre (see figure 4 below).

Figure 4: Frequency domain filters (top) and their corresponding spatial domain counterparts (bottom).



A simple spatial filter that achieves image sharpening is given by

-1/9 -1/9 -1/9

-1/9 8/9 -1/9

-1/9 -1/9 -1/9

Since the sum of all the weights is zero, the resulting signal will have a zero DC value (that is, the average signal value, or the coefficient of the zero frequency term in the Fourier expansion). For display purposes, we might want to add an offset to keep the result in the $0...255$ range.

High boost filtering

We can think of high pass filtering in terms of subtracting a low pass image from the original image, that is,

High pass = Original - Low pass.

However, in many cases where a high pass image is required, we also want to retain some of the low frequency components to aid in the interpretation of the image. Thus, if we multiply the

original image by an amplification factor A before subtracting the low pass image, we will get a *high boost* or *high frequency emphasis* filter. Thus,

$$\begin{aligned} \text{High boost} &= A.\text{Original} - \text{Low pass} \\ &= (A-1).(\text{Original}) + \text{Original} - \text{Low pass} \\ &= (A-1).\text{Original} + \text{High pass}. \end{aligned}$$

Now, if $A = 1$ we have a simple high pass filter. When $A > 1$ part of the original image is retained in the output.

A simple filter for high boost filtering is given by

-1/9 -1/9 -1/9

-1/9 ω /9 -1/9

-1/9 -1/9 -1/9

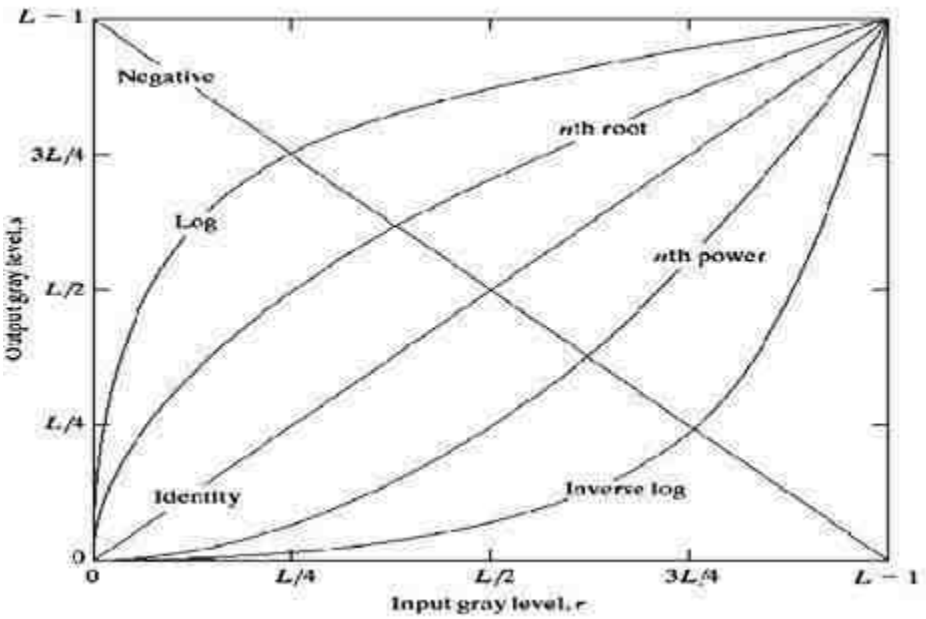
$$\omega = 9A - 1$$

Gray level transformation

There are three basic gray level transformation.

- Linear
- Logarithmic
- Power – law

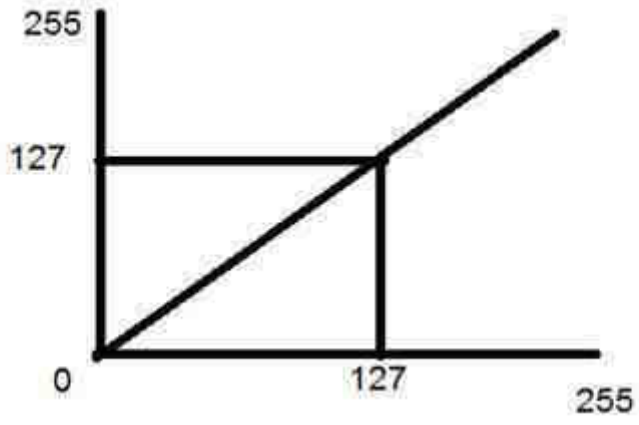
The overall graph of these transitions has been shown below.



Linear transformation

First we will look at the linear transformation. Linear transformation includes simple identity and negative transformation. Identity transformation has been discussed in our tutorial of image transformation, but a brief description of this transformation has been given here.

Identity transition is shown by a straight line. In this transition, each value of the input image is directly mapped to each other value of output image. That results in the same input image and output image. And hence is called identity transformation. It has been shown below:

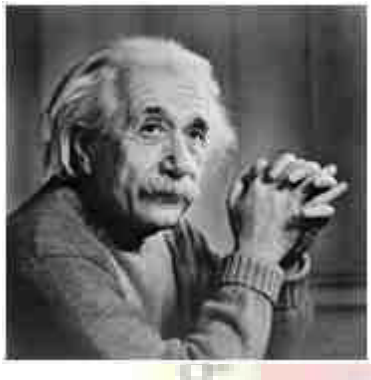


Negative transformation

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

The result is somewhat like this.

Input Image



Output Image

In this case the following transition has been done.

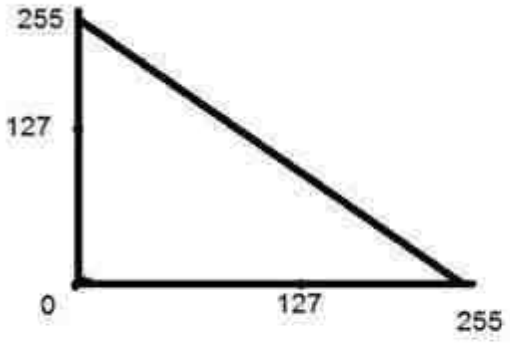
$s = (L - 1) - r$

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

$s = 255 - r$

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

It has been shown in the graph below.



Logarithmic transformations

Logarithmic transformation further contains two type of transformation. Log transformation and inverse log transformation.

Log transformation

The log transformations can be defined by this formula

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

Where s and r are the pixel values of the output and the input image and c is a constant. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement.

The value of c in the log transform adjust the kind of enhancement you are looking for.

Input Image

Log Tranform Image

The inverse log transform is opposite to log transform.

Power – Law transformations

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

s=cr^γ

This symbol γ is called gamma, due to which this transformation is also known as gamma transformation.

Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity.

This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different. For example Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.

Correcting gamma.

s=cr^γ

s=cr^(1/2.5)

The same image but with different gamma values has been shown here.

For example

Gamma = 10

Gamma = 8

Gamma = 6

Histograms

A histogram is a graph. A graph that shows frequency of anything. Usually histogram have bars that represent frequency of occurring of data in the whole data set.

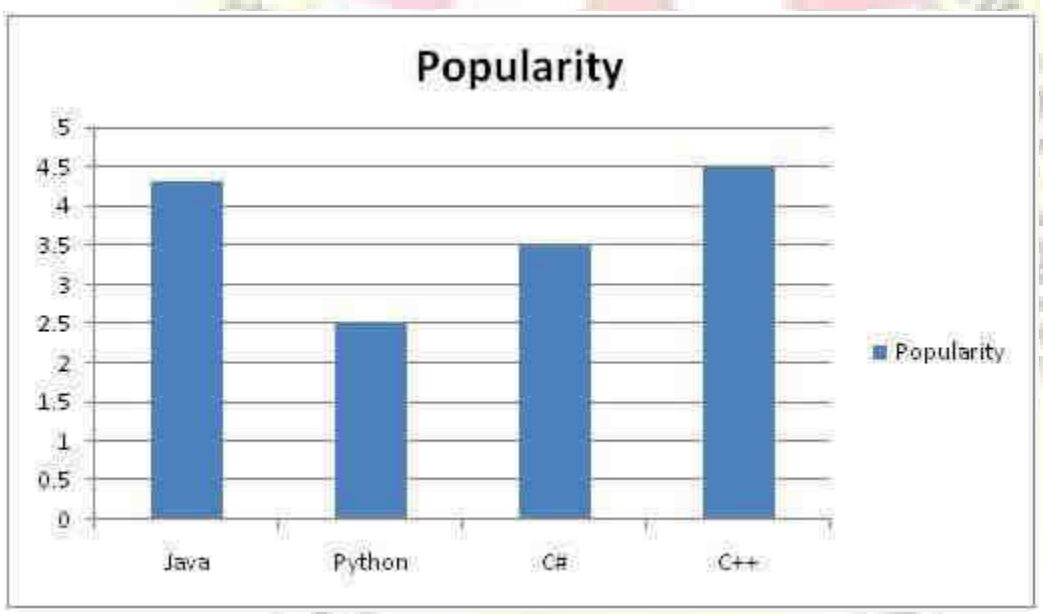
A Histogram has two axis the x axis and the y axis.

The x axis contains event whose frequency you have to count.

The y axis contains frequency.

The different heights of bar shows different frequency of occurrence of data.

Usually a histogram looks like this.



Now we will see an example of this histogram is build

Example

Consider a class of programming students and you are teaching python to them.

At the end of the semester, you got this result that is shown in table. But it is very messy and does not show your overall result of class. So you have to make a histogram of your result, showing the overall frequency of occurrence of grades in your class. Here how you are going to do it.

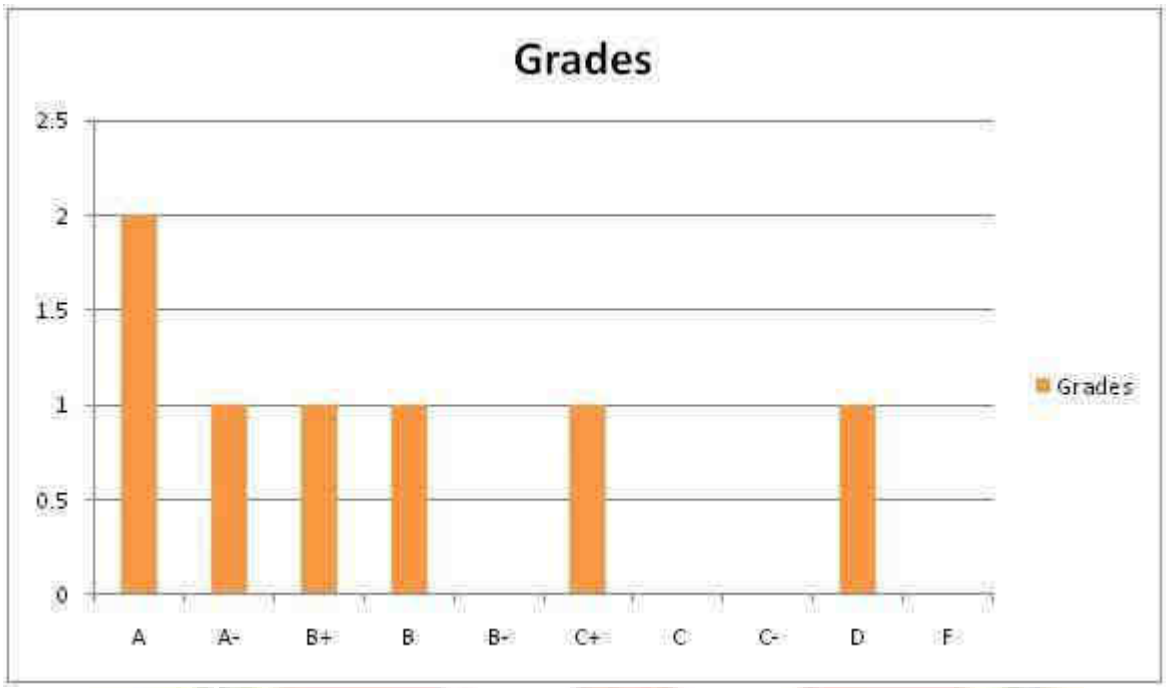
Result sheet

Name	Grade
John	A
Jack	D
Carter	B
Tommy	A
Lisa	C+
Derek	A-
Tom	B+

Histogram of result sheet

Now what you are going to do is, that you have to find what comes on the x and the y axis.

There is one thing to be sure, that y axis contains the frequency, so what comes on the x axis. X axis contains the event whose frequency has to be calculated. In this case x axis contains grades.

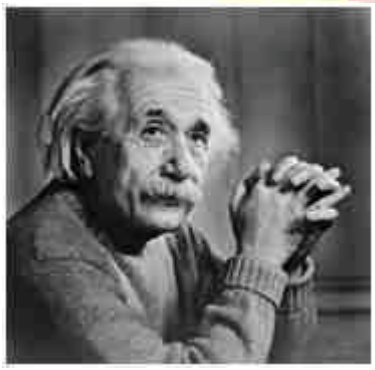


Now we will how do we use a histogram in an image.

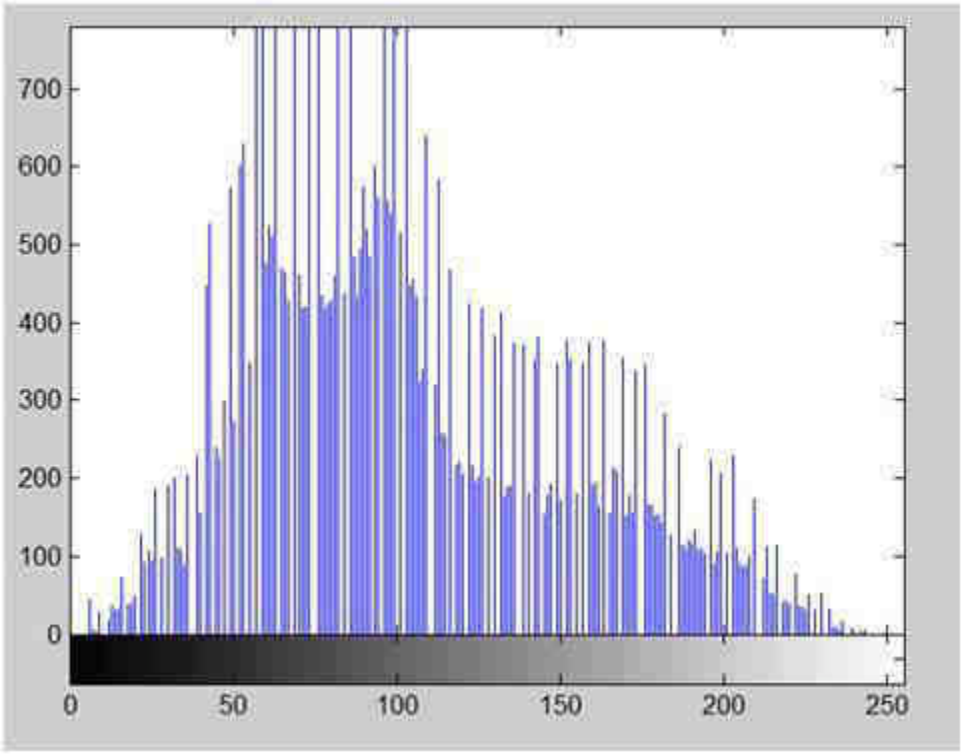
Histogram of an image

Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels intensity values. In an image histogram, the x axis shows the gray level intensities and the y axis shows the frequency of these intensities.

For example



The histogram of the above picture of the Einstein would be something like this



The x axis of the histogram shows the range of pixel values. Since its an 8 bpp image, that means it has 256 levels of gray or shades of gray in it. Thats why the range of x axis starts from 0 and end at 255 with a gap of 50. Whereas on the y axis, is the count of these intensities.

As you can see from the graph, that most of the bars that have high frequency lies in the first half portion which is the darker portion. That means that the image we have got is darker. And this can be proved from the image too.

Applications of Histograms

Histograms has many uses in image processing. The first use as it has also been discussed above is the analysis of the image. We can predict about an image by just looking at its histogram. Its like looking an x ray of a bone of a body.

The second use of histogram is for brightness purposes. The histograms has wide application in image brightness. Not only in brightness, but histograms are also used in adjusting contrast of an image.

Another important use of histogram is to equalize an image.

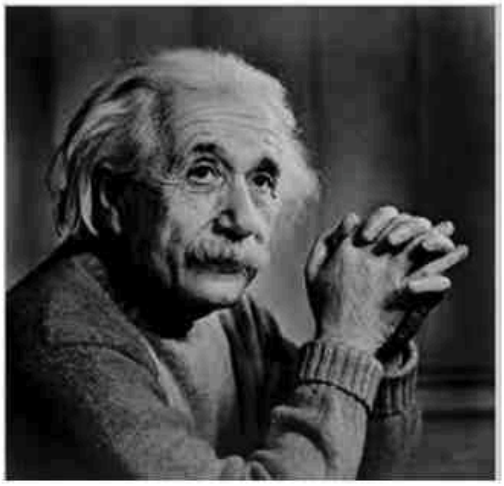
And last but not the least, histogram has wide use in thresholding. This is mostly used in computer vision.

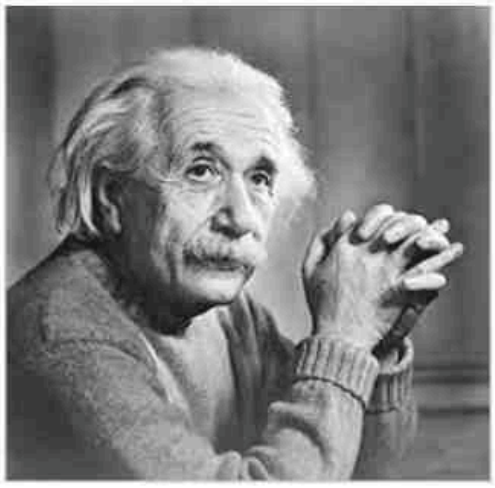
Brightness

Brightness is a relative term. It depends on your visual perception. Since brightness is a relative term, so brightness can be defined as the amount of energy output by a source of light relative to the source we are comparing it to. In some cases we can easily say that the image is bright, and in some cases, its not easy to perceive.

For example

Just have a look at both of these images, and compare which one is brighter.





We can easily see, that the image on the right side is brighter as compared to the image on the left.

But if the image on the right is made more darker then the first one, then we can say that the image on the left is more brighter then the left.

How to make an image brighter.

Brightness can be simply increased or decreased by simple addition or subtraction, to the image matrix.

Consider this black image of 5 rows and 5 columns

Since we already know, that each image has a matrix at its behind that contains the pixel values. This image matrix is given below.

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Since the whole matrix is filled with zero, and the image is very much darker.

Now we will compare it with another same black image to see this image got brighter or not.

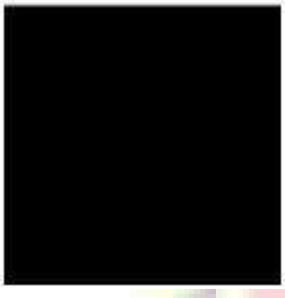
Image 1

Image 2

Still both the images are same, now we will perform some operations on image1 , due to which it becomes brighter then the second one.

What we will do is, that we will simply add a value of 1 to each of the matrix value of image 1. After adding the image 1 would something like this.

Now we will again compare it with image 2, and see any difference.



We see, that still we cannot tell which image is brighter as both images looks the same.

Now what we will do, is that we will add 50 to each of the matrix value of the image 1 and see what the image has become.

The output is given below.

Now again, we will compare it with image 2.

Now you can see that the image 1 is slightly brighter then the image 2. We go on, and add another 45 value to its matrix of image 1, and this time we compare again both images.

Now when you compare it, you can see that this image1 is clearly brighter then the image 2. Even it is brighter then the old image1. At this point the matrix of the image1 contains 100 at each index as first add 5, then 50, then 45. So 5 + 50 + 45 = 100.

Contrast

Contrast can be simply explained as the difference between maximum and minimum pixel intensity in an image.

For example.

Consider the final image1 in brightness.

The matrix of this image is:

100	100	100	100	100
100	100	100	100	100
100	100	100	100	100
100	100	100	100	100
100	100	100	100	100

The maximum value in this matrix is 100.

The minimum value in this matrix is 100.

Contrast = maximum pixel intensity(subtracted by) minimum pixel intensity

= 100 (subtracted by) 100

= 0

0 means that this image has 0 contrast.

Histogram sliding

In histogram sliding, we just simply shift a complete histogram rightwards or leftwards. Due to shifting or sliding of histogram towards right or left, a clear change can be seen in the image.In this tutorial we are going to use histogram sliding for manipulating brightness.

The term i-e: Brightness has been discussed in our tutorial of introduction to brightness and contrast. But we are going to briefly define here.

Brightness

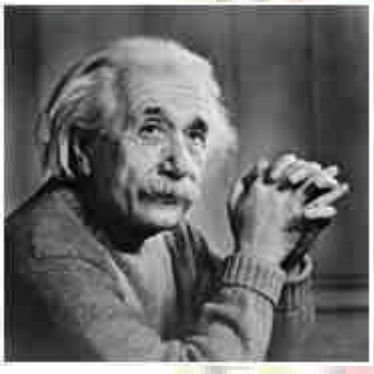
Brightness is a relative term. Brightness can be defined as intensity of light emit by a particular light source.

Contrast

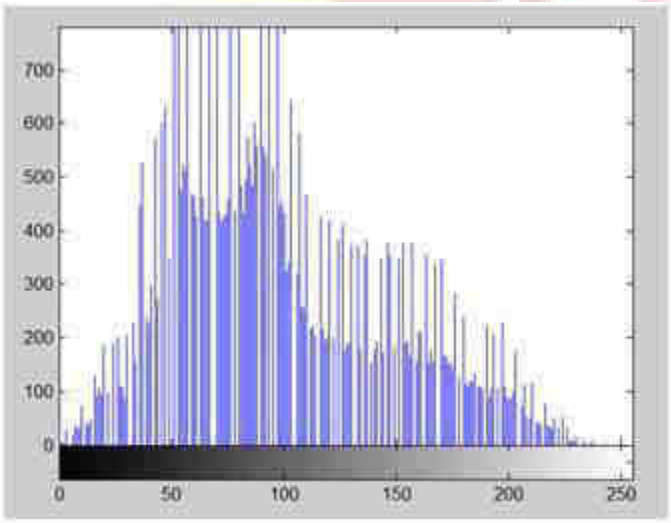
Contrast can be defined as the difference between maximum and minimum pixel intensity in an image.

Sliding Histograms

Increasing brightness using histogram sliding



Histogram of this image has been shown below.



On the y axis of this histogram are the frequency or count. And on the x axis, we have gray level values. As you can see from the above histogram, that those gray level intensities whose

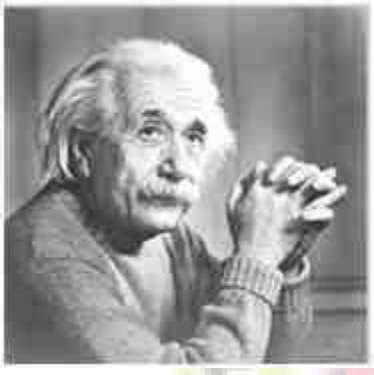
count is more then 700, lies in the first half portion, means towards blacker portion. Thats why we got an image that is a bit darker.

In order to bright it, we will slide its histogram towards right, or towards whiter portion. In order to do we need to add atleast a value of 50 to this image. Because we can see from the histogram above, that this image also has 0 pixel intensities, that are pure black. So if we add 0 to 50, we will shift all the values lies at 0 intensity to 50 intensity and all the rest of the values will be shifted accordingly.

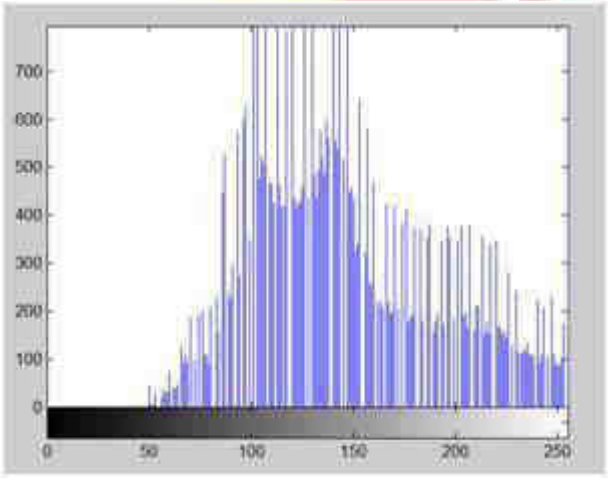
Lets do it.

Here what we got after adding 50 to each pixel intensity.

The image has been shown below.



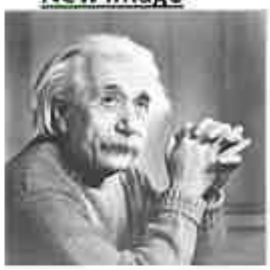
And its histogram has been shown below.



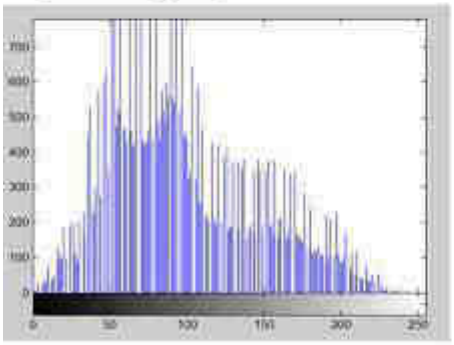
Lets compare these two images and their histograms to see that what change have to got.

Old image

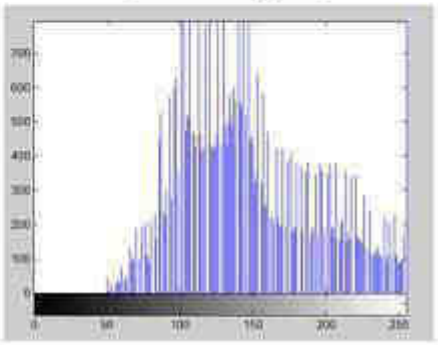
New image



Old histogram



New Histogram



Conclusion

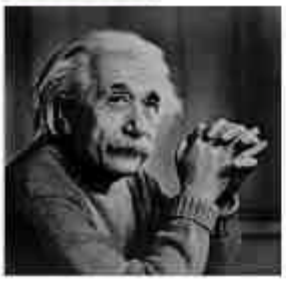
As we can clearly see from the new histogram that all the pixels values has been shifted towards right and its effect can be seen in the new image.

Decreasing brightness using histogram sliding

Now if we were to decrease brightness of this new image to such an extent that the old image look brighter, we got to subtract some value from all the matrix of the new image. The value which we are going to subtract is 80. Because we already add 50 to the original image and we got a new brighter image, now if we want to make it darker, we have to subtract at least more than 50 from it.

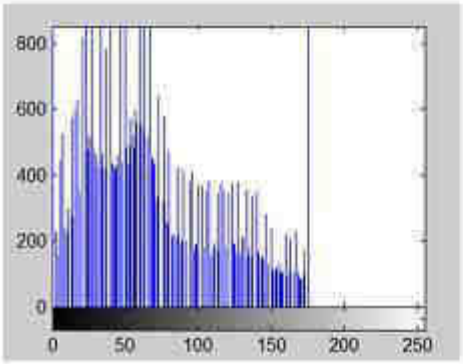
And this what we got after subtracting 80 from the new image.

New image.

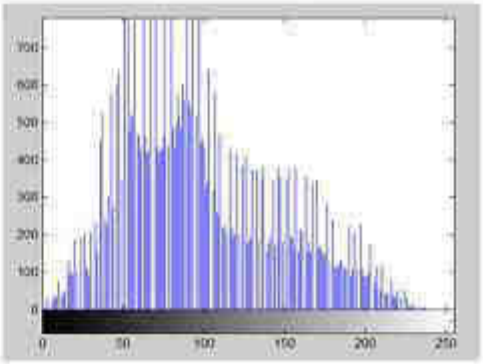


Original image.

New Histogram.



Original Histogram.



Conclusion

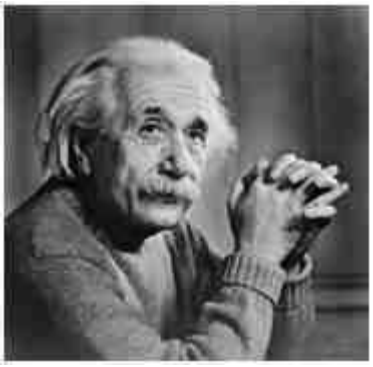
It is clear from the histogram of the new image, that all the pixel values has been shifted towards right and thus, it can be validated from the image that new image is darker and now the original image look brighter as compare to this new image.

Histogram Equalization

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram equalization can be worse. In that cases the contrast is decreased.

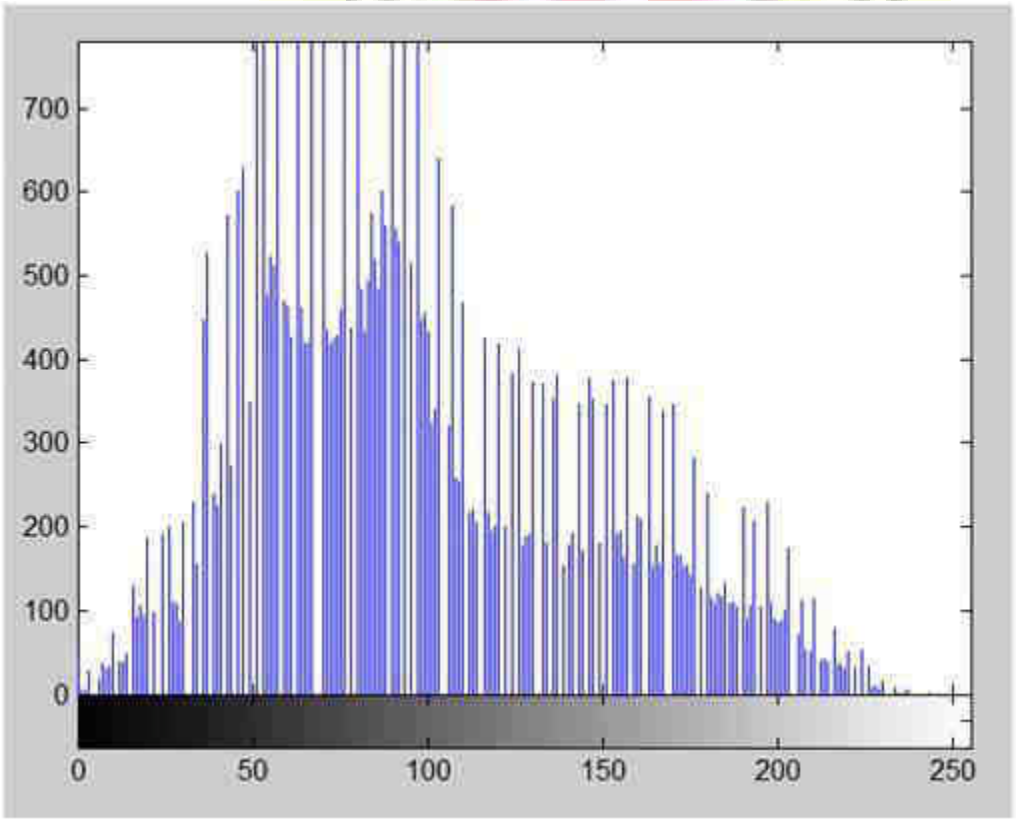
Lets start histogram equalization by taking this image below as a simple image.

Image



Histogram of this image

The histogram of this image has been shown below.



Now we will perform histogram equalization to it.

PMF

First we have to calculate the PMF (probability mass function) of all the pixels in this image. If you donot know how to calculate PMF, please visit our tutorial of PMF calculation.

CDF

Our next step involves calculation of CDF (cumulative distributive function). Again if you donot know how to calculate CDF , please visit our tutorial of CDF calculation.

Calculate CDF according to gray levels

Lets for instance consider this , that the CDF calculated in the second step looks like this.

Gray Level Value	CDF
0	0.11
1	0.22
2	0.55
3	0.66
4	0.77
5	0.88
6	0.99
7	1

Then in this step you will multiply the CDF value with (Gray levels (minus) 1) .

Considering we have an 3 bpp image. Then number of levels we have are 8. And 1 subtracts 8 is 7. So we multiply CDF by 7. Here what we got after multiplying.

Gray Level Value	CDF	CDF * (Levels-1)
0	0.11	0
1	0.22	1
2	0.55	3
3	0.66	4
4	0.77	5
5	0.88	6
6	0.99	6
7	1	7

Now we have is the last step, in which we have to map the new gray level values into number of pixels.

Lets assume our old gray levels values has these number of pixels.

Gray Level Value	Frequency
0	2

1	4
2	6
3	8
4	10
5	12
6	14
7	16

Now if we map our new values to , then this is what we got.

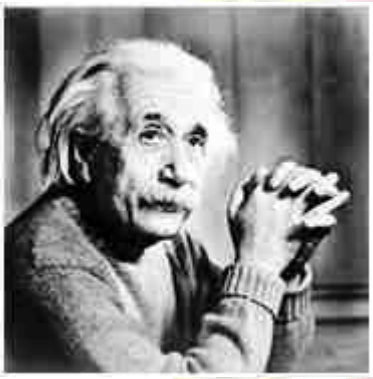
Gray Level Value	New Gray Level Value	Frequency
0	0	2
1	1	4
2	3	6
3	4	8
4	5	10

5	6	12
6	6	14
7	7	16

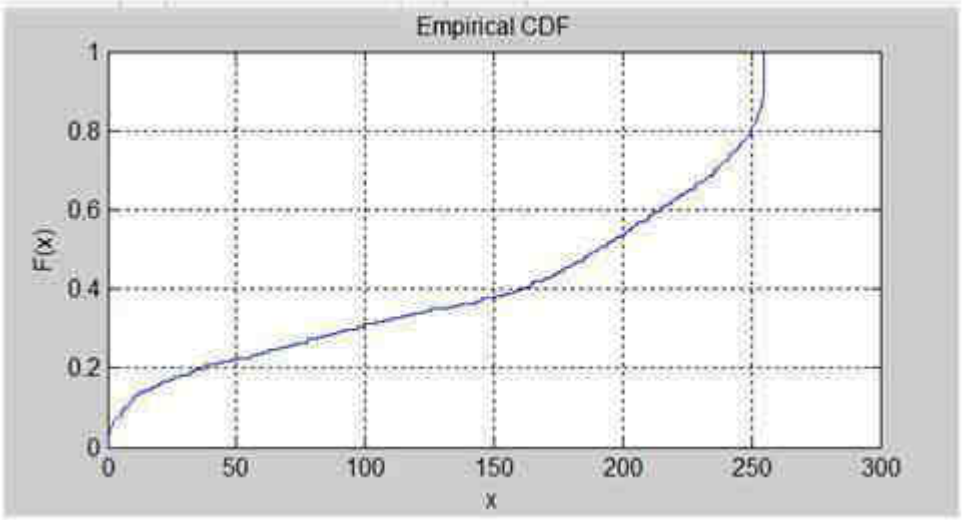
Now map these new values you are onto histogram, and you are done.

Lets apply this technique to our original image. After applying we got the following image and its following histogram.

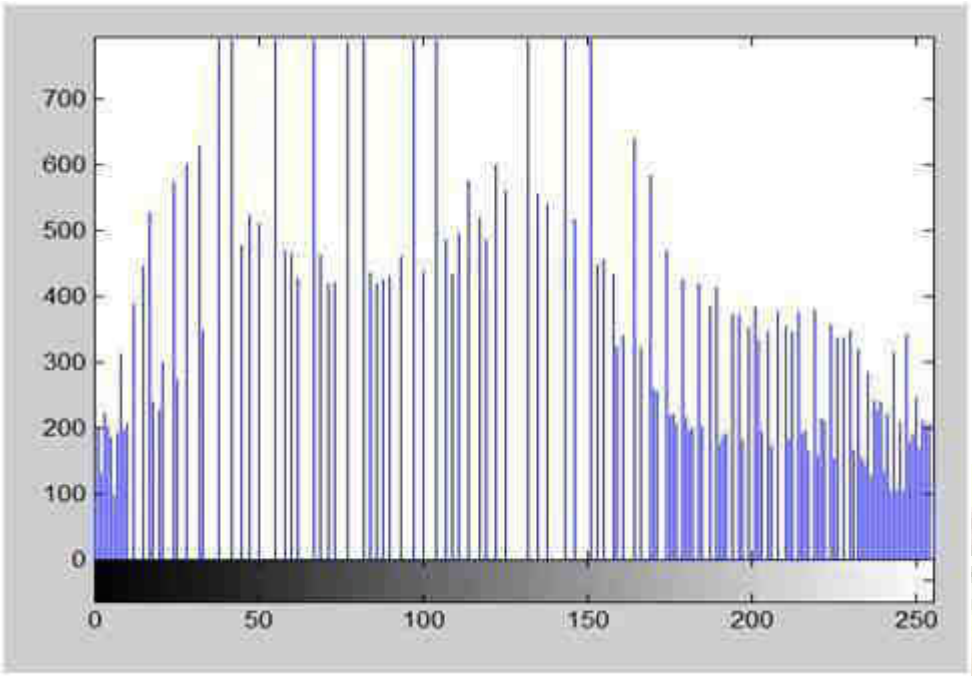
Histogram Equalization Image



Cumulative Distributive function of this image

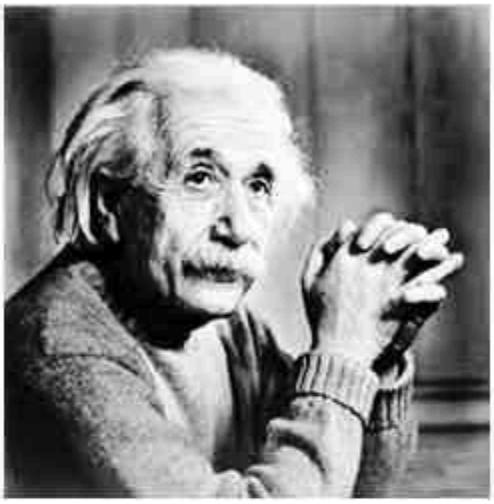


Histogram Equalization histogram

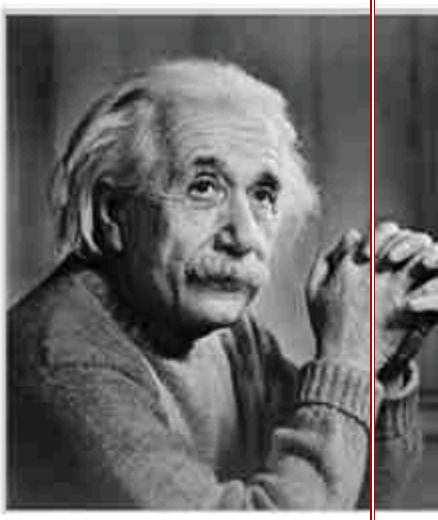


Comparing both the histograms and images

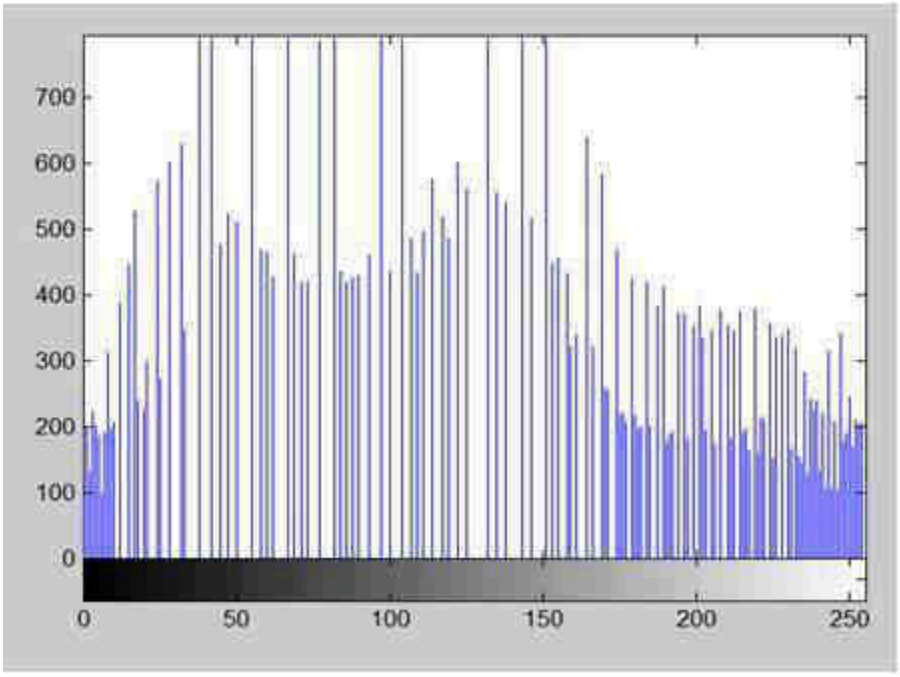
New Image



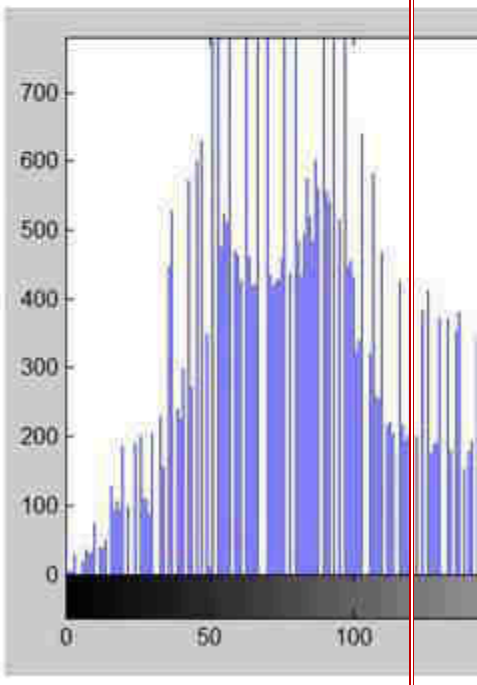
Old image



New Histogram



Old Histogram



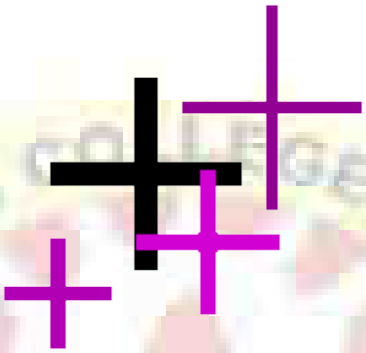
Conclusion

As you can clearly see from the images that the new image contrast has been enhanced and its histogram has also been equalized. There is also one important thing to be note here that during

histogram equalization the overall shape of the histogram changes, where as in histogram stretching the overall shape of histogram remains same.

Image Enhancement Using Arithmetic And Logic Operations

Pixel Addition



Common Names: Pixel Add, Sum, Offset

Brief Description

In its most straightforward implementation, this operator takes as input two identically sized images and produces as output a third image of the same size as the first two, in which each pixel value is the sum of the values of the corresponding pixel from each of the two input images. More sophisticated versions allow more than two images to be combined with a single operation.

A common variant of the operator simply allows a specified constant to be added to every pixel.

How It Works

The addition of two images is performed straightforwardly in a single pass. The output pixel values are given by:

$$Q(i,j) = P_1(i,j) + P_2(i,j)$$

Or if it is simply desired to add a constant value C to a single image then:

$$Q(i,j) = P_1(i,j) + C$$

If the pixel values in the input images are actually vectors rather than scalar values (*e.g.* for color images) then the individual components (*e.g.* red, blue and green components) are simply added separately to produce the output value.

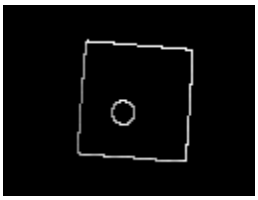
If the image format being used only supports, say 8-bit integer pixel values, then it is very easy for the result of the addition to be greater than the maximum allowed pixel value. The effect of this depends upon the particular implementation. The overflowing pixel values might just be set to the maximum allowed value, an effect known as saturation. Alternatively the pixel values might wrap around from zero again. If the image format supports pixel values with a much larger range, *e.g.* 32-bit integers or floating point numbers, then this problem does not occur so much.

Guidelines for Use

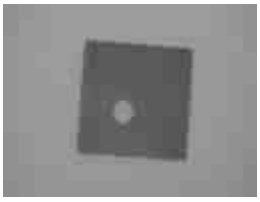
Image addition crops up most commonly as a sub-step in some more complicated process rather than as a useful operator in its own right. As an example we show how addition can be used to overlay the output from an edge detector on top of the original image after suitable masking has been carried out.

The image

shows a simple flat dark object against a light background. Applying the Canny edge detector to this image, we obtain

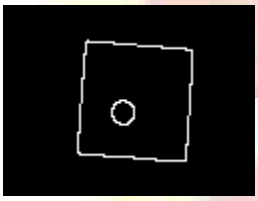


Suppose that our task is to overlay this edge data on top of the original image. The image

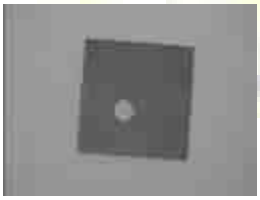


is the result of straightforwardly adding the two images. Since the sum of the edge pixels and the underlying values in the original is greater than the maximum possible pixel value, these pixels are (in this implementation) wrapped around. Therefore these pixels have a rather low pixel value and it is hard to distinguish them from the surrounding pixels. In order to avoid the pixel overflow we need to *replace* pixels in the original image with the corresponding edge data pixels, at every place where the edge data pixels are non-zero. The way to do this is to mask off a region of the original image before we do any addition.

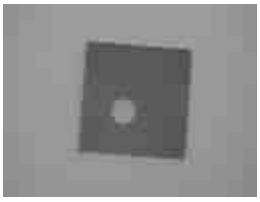
The mask is made by thresholding the edge data at a pixel value of 128 in order to produce



This mask is then inverted and subsequently ANDed with the original image to produce

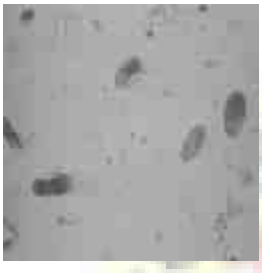


Finally, the masked image is added to the unthresholded edge data to produce

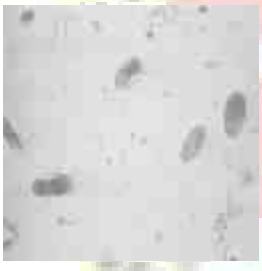


This image now clearly shows that the Canny edge detector has done an extremely good job of localizing the edges of the original object accurately. It also shows how the response of the edge detector drops off at the fuzzier left hand edge of the object.

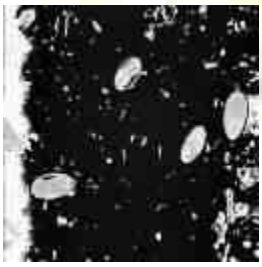
Other uses of addition include adding a constant offset to all pixels in an image so as to brighten that image. For example, adding a constant value of *50* to



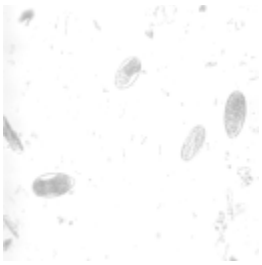
yields



It is important to realize that if the input images are already quite bright, then straight addition may produce a pixel value overflow. Image

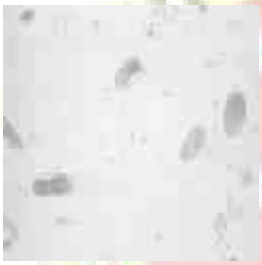


shows the results of adding *100* to the above image. Most of the background pixels are greater than the possible maximum (*255*) and therefore are (with this implementation of addition) wrapped around from zero. If we implement the operator in such a way that pixel values exceeding the maximum value are set to *255* (*i.e.* using a hard limit) we obtain

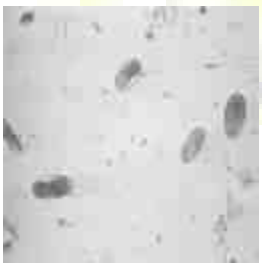


This image looks more natural than the wrapped around one. However, due to the saturation, we lose a certain amount of information, since all the values exceeding the maximum value are set to the same graylevel.

In this case, the pixel values should be scaled down before addition. The image



is the result of scaling the original with *0.8* and adding a constant value of *100*. Although the image is brighter than the original, it has lost contrast due to the scaling. In most cases, scaling the image with a factor larger than *1* without using addition at all provides a better way to brighten an image, as it increases the image contrast. For comparison,



is the original image multiplied with *1.3*.

Blending provides a slightly more sophisticated way of merging two images which ensures that saturation cannot happen.

When adding color images it is important to consider how the color information has been encoded. The section on 8-bit color images describes the issues to be aware of when adding such images.

Interactive Experimentation

You can interactively experiment with this operator by clicking here

Pixel Subtraction

Since we already know, that each image has a matrix at its behind that contains the pixel values. This image matrix is given below.

0	0	0	0	0
0	0	0	0	0

shows some text which has been badly illuminated during capture so that there is a strong illumination gradient across the image. If we wish to separate out the foreground text from the background page, then the obvious method for black on white text is simply to threshold the image on the basis of intensity. However, simple thresholding fails here due to the illumination gradient. A typical failed attempt looks like

Now it may be that we cannot adjust the illumination, but we can put different things in the scene. This is often the case with microscope imaging, for instance. So we replace the text with a sheet of white paper and without changing anything else we capture a new image, as shown in

This image is the *lightfield*. Now we can subtract the lightfield image from the original image to attempt to eliminate variation in the background intensity. Before doing that an offset of 100 is added to the first image to in order avoid getting negative numbers and we also use 32-bit integer pixel values to avoid overflow problems. The result of the subtraction is shown in

Note that the background intensity of the image is much more uniform than before, although the contrast in the lower part of the image is still poor. Straightforward thresholding can now achieve better results than before, as shown in

which is the result of thresholding at a pixel value of 80. Note that the results are still not ideal, since in the poorly lit areas of the image the contrast (*i.e.* difference between foreground and background intensity) is much lower than in the brightly lit areas, making a suitable threshold difficult or impossible to find. Compare these results with the example described under pixel division.

Absolute image differencing is also used for change detection. If the absolute difference between two frames of a sequence of images is formed, and there is nothing moving in the scene, then the output will mostly consist of zero value pixels. If however, there is movement going on, then pixels in regions of the image where the intensity changes spatially, will exhibit significant absolute differences between the two frames.

As an example of such change detection, consider

which shows an image of a collection of screws and bolts. The image

shows a similar scene with one or two differences. If we calculate the absolute difference between the frames as shown in

then the regions that have changed become clear. The last image here has been contrast-stretched in order to improve clarity.

Subtraction can also be used to estimate the temporal derivative of intensity at each point in a sequence of images. Such information can be used, for instance, in optical flow calculations.

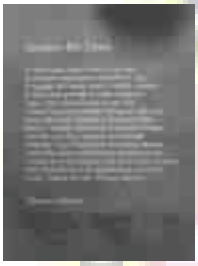
Simple subtraction of a constant from an image can be used to darken an image, although scaling is normally a better way of doing this.

It is important to think about whether negative output pixel values can occur as a result of the subtraction, and how the software will treat pixels that do have negative values. An example of what may happen can be seen in

which is the above *lightfield* directly subtracted from the text images. In the implementation of *pixel subtraction* which was used, negative values are wrapped around starting from the maximum value. Since we don't have exactly the same reflectance of the paper when taking the images of the lightfield and the text, the difference of pixels belonging to background is either slightly above or slightly below zero. Therefore the wrapping results in background pixels with either very small or very high values, thus making the image unsuitable for further processing (for example, thresholding). If we alternatively set all negative values to zero, the image would become completely black, because subtracting the pixels in the lightfield from the pixels representing characters in the text image yields negative results, as well.

In this application, a suitable way to deal with negative values is to use absolute differences, as can be seen in

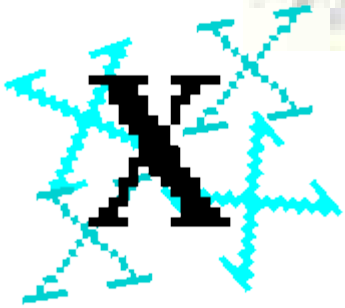
or as a gamma corrected version in



Thresholding this image yields similar good results as the earlier example.

If negative values are to be avoided then it may be possible to first add an offset to the first input image. It is also often useful if possible to convert the pixel value type to something with a sufficiently large range to avoid overflow, *e.g.* 32-bit integers or floating point numbers.

Pixel Multiplication and Scaling



Common Names: Pixel Multiplication, Graylevel scale

Brief Description

Like other image arithmetic operators, multiplication comes in two main forms. The first form takes two input images and produces an output image in which the pixel values are just those of the first image, multiplied by the values of the corresponding values in the second image. The second form takes a single input image and produces output in which each pixel value is multiplied by a specified constant. This latter form is probably the more widely used and is generally called *scaling*.

This *graylevel* scaling should not be confused with geometric scaling.

How It Works

The multiplication of two images is performed in the obvious way in a single pass using the formula:

$$Q(i,j) = P_1(i,j) \times P_2(i,j)$$

Scaling by a constant is performed using:

$$Q(i,j) = P_1(i,j) \times C$$

Note that the constant is often a floating point number, and may be less than one, which will reduce the image intensities. It may even be negative if the image format supports that.

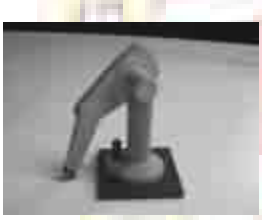
If the pixel values are actually vectors rather than scalar values (*e.g.* for color images) then the individual components (*e.g.* ref{rgb}{red, blue and green components}) are simply multiplied separately to produce the output value.

If the output values are calculated to be larger than the maximum allowed pixel value, then they may either be truncated at that maximum value, or they can `wrap around' and continue upwards from the minimum allowed number again.

Guidelines for Use

There are many specialist uses for scaling. In general though, given a scaling factor greater than one, scaling will brighten an image. Given a factor less than one, it will darken the image. Scaling generally produces a much more natural brightening/darkening effect than simply adding an offset to the pixels, since it preserves the relative contrast of the image better. For instance,

shows a picture of model robot that was taken under low lighting conditions. Simply scaling every pixel by a factor of 3, we obtain



which is much clearer. However, when using pixel multiplication, we should make sure that the calculated pixel values don't exceed the maximum possible value. If we, for example, scale the above image by a factor of 5 using a 8-bit representation, we obtain

All the pixels which, in the original image, have a value greater than 51 exceed the maximum value and are (in this implementation) wrapped around from *255* back to *0*.

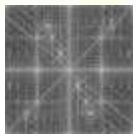
The last example shows that it is important to be aware of what will happen if the multiplications result in pixel values outside the range that can be represented by the image format being used. It is also very easy to generate very large numbers with pixel-by-pixel multiplication. If the image processing software supports it, it is often safest to change to an image format with a large range, *e.g.* floating point, before attempting this sort of calculation.

Scaling is also often useful prior to other image arithmetic in order to prevent pixel values going out of range, or to prevent integer quantization ruining the results (as in integer image division).

Pixel-by-pixel multiplication is generally less useful, although sometimes a binary image can be used to multiply another image in order to act as a mask. The idea is to multiply by 1 those pixels that are to be preserved, and multiply by zero those that are not. However for integer format images it is often easier and faster to use the logical operator AND instead.

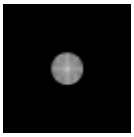
Another use for pixel by pixel multiplication is to filter images in the frequency domain. We illustrate the idea using the example of

First, we obtain



by applying the Fourier transform to the original image, and then we use pixel multiplication to attenuate certain frequencies in the Fourier domain. In this example we use a simple lowpass filter which (as a scaled version) can be seen in

The result of the multiplication is shown in



Finally, an inverse Fourier transform is performed to return to the spatial domain. The final result

shows the smoothing effect of a lowpass filter. More details and examples are given in the worksheets dealing with frequency filtering.

Interactive Experimentation

You can interactively experiment with this operator by clicking here.

Exercises

1. Overlay

and its skeleton

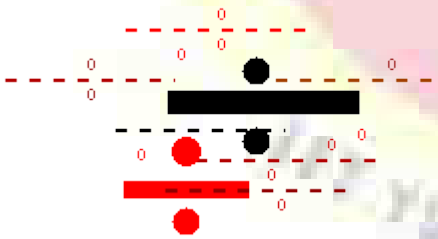
using pixel addition (the skeleton was derived from

which was produced by thresholding the input image at 110). Use image multiplication to scale the images prior to the addition in order to avoid the pixel values being out of range. What effect does this have on the contrast of the input images.

2. Use thresholding to segment the simple image

into foreground and background. Use scaling to set the foreground pixel value to 2, and the background pixel value to 0. Then use pixel-by-pixel multiplication to multiply this image with the original image. What has this process achieved and why might it be useful?

Pixel Division



Common Names: Pixel Division, Ratioing

Brief Description

The image division operator normally takes two images as input and produces a third whose pixel values are just the pixel values of the first image divided by the corresponding pixel values of the second image. Many implementations can also be used with just a single input image, in which case every pixel value in that image is divided by a specified constant.

How It Works

The division of two images is performed in the obvious way in a single pass using the formula:

$$Q(i,j) = P_1(i,j) \div P_2(i,j)$$

Division by a constant is performed using:

$$Q(i,j) = P_1(i,j) \div C$$

If the pixel values are actually vectors rather than scalar values (*e.g.* for color images) than the individual components (*e.g.* red, blue and green components) are simply divided separately to produce the output value.

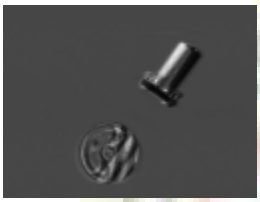
The division operator may only implement integer division, or it may also be able to handle floating point division. If only integer division is performed, then results are typically rounded down to the next lowest integer for output. The ability to use images with pixel value types other than simply 8-bit integers comes in very handy when doing division.

Guidelines for Use

One of the most important uses of division is in change detection, in a similar way to the use of subtraction for the same thing. Instead of giving the absolute change for each pixel from one frame to the next, however, division gives the fractional change or ratio between corresponding pixel values (hence the common alternative name of *ratioing*). The images

and

are of the same scene except two objects have been slightly moved between the exposures. Dividing the former by the latter using a floating point pixel type and then contrast stretching the resulting image yields



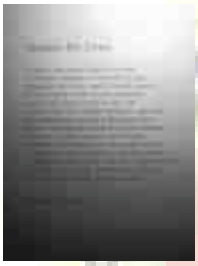
After the division, pixels which didn't change between the exposures have a value of 1, whereas if the pixel value increased after the first exposure the result of the division is clustered between 0 and 1, otherwise it is between 1 and 255 (provided the pixel value in the second image is not smaller than 1). That is the reason why we can only see the new position of the moved part in the contrast-stretched image. The old position can be visualized by histogram equalizing the division output, as shown in

Here, high values correspond to the new position, low values correspond to the old position, assuming that the intensity of the moved object is lower than the background intensity. Intermediate graylevels in the equalized image correspond to areas of no change. Due to noise, the image also shows the position of objects which were not moved.

For comparison, the absolute difference between the two images, as shown in

produces approximately the same pixel values at the old and the new position of a moved part.

Another application for pixel division is to separate the actual reflectance of an object from the unwanted influence of illumination. This image



shows a poorly illuminated piece of text. There is a strong illumination gradient across the image which makes conventional foreground/background segmentation using standard thresholding impossible. The image

shows the result of straightforward intensity thresholding at a pixel value of 128. There is no global threshold value that works over the whole of the image.

Suppose that we cannot change the lighting conditions, but that we can take several images with different items in the viewfield. This situation arises quite a lot in microscopy, for instance. We choose to take a picture of a blank sheet of white paper which should allow us to capture the incident illumination variation. This *lightfield* image is shown in

Now, assuming that we are dealing with a flat scene here, with points on the surface of the scene described by coordinates *x* and *y*, then the reflected light intensity *B(x,y)* depends upon the reflectance *R(x,y)* of the scene at that point and also on the incident illumination *I(x,y)* such that:

$$B(x,y) \propto I(x,y) \times R(x,y)$$

Using subscripts to distinguish the blank (lightfield) image and the original image, we can write:

$$\frac{B_{orig}(x,y)}{B_{blank}(x,y)} \propto \frac{I_{orig}(x,y) \times R_{orig}(x,y)}{I_{blank}(x,y) \times R_{blank}(x,y)}$$

But since *I(x,y)* is the same for both images, and assuming the reflectance of the blank paper to be uniform over its surface, then:

$$\frac{B_{orig}(x,y)}{B_{blank}(x,y)} \propto R_{orig}(x,y)$$

Therefore the division should allow us to segment the letters out nicely. In image

we see the result of dividing the original image by the lightfield image. Note that floating point format images were used in the division, which were then normalized to 8-bit integers for display. Virtually all the illumination gradient has been removed. The image

shows the result of thresholding this image at a pixel value of 160. While not fantastic, with a little work using morphological operations, the text could become quite legible. Compare the result with that obtained using subtraction.

As with other image arithmetic operations, it is important to be aware of whether the implementation being used does integer or floating point arithmetic. Dividing two similar images, as done in the above examples, results mostly in very small pixel values, seldom greater than 4 or 5. To display the result, the image has to be normalized to 8-bit integers. However, if the division is performed in an integer format the result is quantized before the normalization, hence a lot of information is lost. Image

shows the result of the above change detection if the division is performed in integer format. The maximum result of the division was less than 3, therefore the integer image contains only three different values, *i.e.* 0, 1 and 2 before the normalization. One solution is to multiply the first image (the numerator image) by a scaling factor before performing the division. Of course this is not generally possible with 8-bit integer images since significant scaling will simply saturate all the pixels in the image. The best method is, as was done in the above examples, to switch to a non-byte image type, and preferably to a floating point format. The effect is that the image is not quantized until the normalization and therefore the result does contain more graylevels. If floating point cannot be used, then use, say, 32-bit integers, and scale up the numerator image before dividing.

Logical AND/NAND

Common Names: AND, NAND

Brief Description

AND and NAND are examples of logical operators having the truth-tables shown in Figure 1.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Figure 1 Truth-tables for AND and NAND.

As can be seen, the output values of NAND are simply the inverse of the corresponding output values of AND.

The AND (and similarly the NAND) operator typically takes two binary or integer graylevel images as input, and outputs a third image whose pixel values are just those of the first image, ANDed with the corresponding pixels from the second. A variation of this operator takes just a single input image and ANDs each pixel with a specified constant value in order to produce the output.

How It Works

The operation is performed straightforwardly in a single pass. It is important that all the input pixel values being operated on have the same number of bits in them or unexpected things may happen. Where the pixel values in the input images are not simple 1-bit numbers, the AND operation is normally (but not always) carried out individually on each corresponding bit in the pixel values, in bitwise fashion.

Guidelines for Use

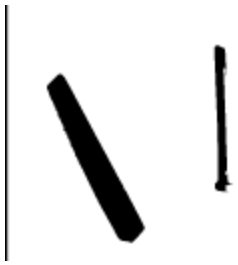
The most obvious application of AND is to compute the intersection of two images. We illustrate this with an example where we want to detect those objects in a scene which did not move between two images, *i.e.* which are at the same pixel positions in the first *and* the second image. We illustrate this example using

and

If we simply AND the two graylevel images in a bitwise fashion we obtain

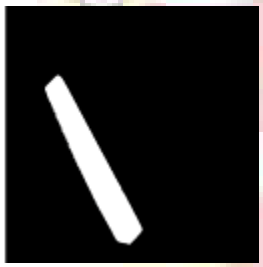
Although we wanted the moved object to disappear from the resulting image, it appears twice, at its old and at its new position. The reason is that the object has rather low pixel values (similar to a logical 0) whereas the background has a high values (similar to a logical 1). However, we normally associate an object with logical 1 and the background with logical 0, therefore we actually ANDed the negatives of two images, which is equivalent to NOR them. To obtain the desired result we have to invert the images before ANDing them, as it was done in

Now, only the object which has the same position in both images is highlighted. However, ANDing two graylevel images might still cause problems, as it is not guaranteed that ANDing two high pixel values in a bitwise fashion yields a high output value (for example, 128 AND 127 yields 0). To avoid these problems, it is best to produce a binary versions from the grayscale images using thresholding.



and

are the thresholded versions of the above images and



is the result of ANDing their negatives.

Although ANDing worked well for the above example, it runs into problems in a scene like

Here, we have two objects with the average intensity of one being higher than the background and the other being lower. Hence, we can't produce a binary image containing both objects using

simple thresholding. As can be seen in the following images, ANDing the grayscale images is not successful either. If in the second scene the light part was moved, as in

then the result of ANDing the two images is

It shows the desired effect of attenuating the moved object. However, if the second scene is somehow like

where the dark object was moved, we obtain

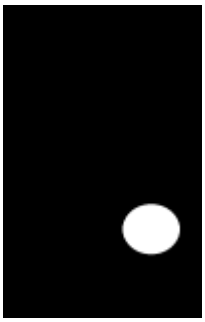
Here, the old and the new positions of the dark object are visible.

In general, applying the AND operator (or other logical operators) to two images in order to detect differences or similarities between them is most appropriate if they are binary or can be converted into binary format using thresholding.

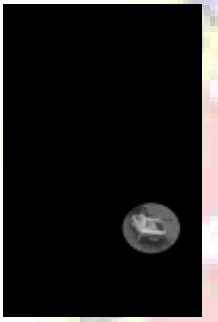
As with other logical operators, AND and NAND are often used as sub-components of more complex image processing tasks. One of the common uses for AND is for masking. For example, suppose we wish to selectively brighten a small region of

to highlight a particular car. There are many ways of doing this and we illustrate just one. First a paint program is used to identify the region to be highlighted. In this case we set the region to black as shown in

This image can then be thresholded to just select the black region, producing the mask shown in



The mask image has a pixel value of 255 (11111111 binary) in the region that we are interested in, and zero pixels (00000000 binary) elsewhere. This mask is then bitwise ANDed with the original image to just select out the region that will be highlighted. This produces



Finally, we brighten this image by scaling it by a factor of 1.1, dim the original image using a scale factor of 0.8, and then add the two images together to produce

AND can also be used to perform so called *bit-slicing* on an 8-bit image. To determine the influence of one particular bit on an image, it is ANDed in a bitwise fashion with a constant number, where the relevant bit is set to 1 and the remaining 7 bits are set to 0. For example, to obtain the bit-plane 8 (corresponding to the most significant bit) of

we AND the image with 128 (10000000 binary) and threshold the output at a pixel value of 1. The result, shown in

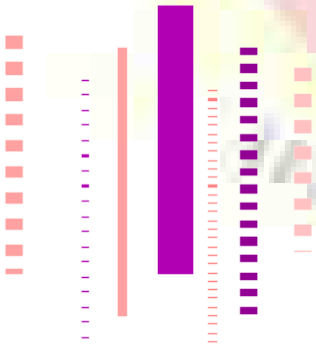
is equivalent to thresholding the image at a value of 128. Images

and

correspond to bit-planes 7, 6 and 4. The images show that most image information is contained in the higher (more significant) bits, whereas the less significant bits contain some of the finer details and noise. The image

shows bit-plane 1.

Logical OR/NOR



Common Names: OR, NOR

Brief Description

OR and NOR are examples of logical operators having the truth-tables shown in Figure 1.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

NOR

Figure 1 Truth-tables for OR and NOR.

As can be seen, the output values of NOR are simply the inverses of the corresponding output values of OR.

The OR (and similarly the NOR) operator typically takes two binary or graylevel images as input, and outputs a third image whose pixel values are just those of the first image, ORed with the corresponding pixels from the second. A variation of this operator takes just a single input image and ORs each pixel with a specified constant value in order to produce the output.

How It Works

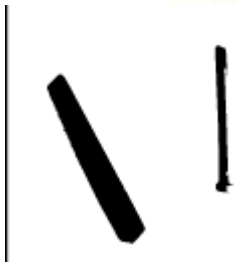
The operation is performed straightforwardly in a single pass. It is important that all the input pixel values being operated on have the same number of bits in them or unexpected things may happen. Where the pixel values in the input images are not simple 1-bit numbers, the OR operation is normally (but not always) carried out individually on each corresponding bit in the pixel values, in bitwise fashion.

Guidelines for Use

We can illustrate the function of the OR operator using

and

The images show a scene with two objects, one of which was moved between the exposures. We can use OR to compute the *union* of the images, *i.e.* highlighting all pixels which represent an object either in the first *or* in the second image. First, we threshold the images, since the process is simplified by use binary input. If we OR the resulting images



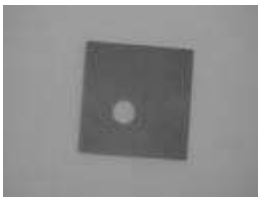
and

we obtain

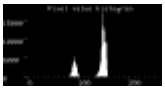
This image shows only the position of the object which was at the same location in both input images. The reason is that the objects are represented with logically 0 and the background is logically 1. Hence, we actually OR the background which is equivalent to NANDing the objects. To get the desired result, we first have to invert the input images before ORing them. Then, we obtain

Now, the output shows the position of the stationary object as well as that of the moved object.

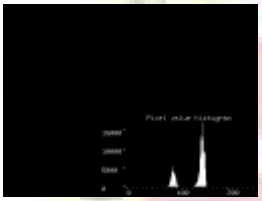
As with other logical operators, OR and NOR are often used as sub-components of more complex image processing tasks. OR is often used to merge two images together. Suppose we want to overlay



with its histogram, shown in



First, an image editor is used to enlarge the histogram image until it is the same size as the grayscale image as shown in



Then, simply ORing the two gives

The performance in this example is quite good, because the images contain very distinct graylevels. If we proceed in the same way with

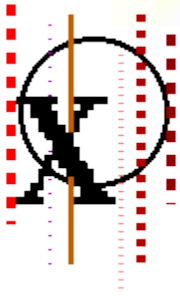
we obtain

Now, it is difficult to see the characters of the histogram (which have high pixel values) at places where the original image has high values, as well. Compare the result with that described under XOR.

Note that there is no problem of overflowing pixel values with the OR operator, as there is with the addition operator.

ORing is usually safest when at least one of the images is binary, *i.e.* the pixel values are 0000... and 1111... only. The problem with ORing other combinations of integers is that the output result can fluctuate wildly with a small change in input values. For instance 127 ORed with 128 gives 255, whereas 127 ORed with 126 gives 127.

Logical XOR/XNOR



Common Names: XOR, XNOR, EOR, ENOR

Brief Description

XOR and XNOR are examples of logical operators having the truth-tables shown in Figure 1.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

XOR

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

XNOR

Figure 1 Truth-tables for XOR and XNOR.

The XOR function is only true if just one (and only one) of the input values is true, and false otherwise. XOR stands for *eXclusive OR*. As can be seen, the output values of XNOR are simply the inverse of the corresponding output values of XOR.

The XOR (and similarly the XNOR) operator typically takes two binary or graylevel images as input, and outputs a third image whose pixel values are just those of the first image, XORed with the corresponding pixels from the second. A variation of this operator takes a single input image and XORs each pixel with a specified constant value in order to produce the output.

How It Works

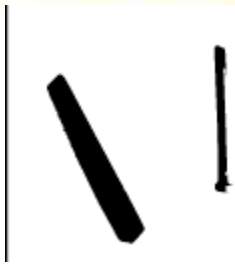
The operation is performed straightforwardly in a single pass. It is important that all the input pixel values being operated on have the same number of bits in them, or unexpected things may happen. Where the pixel values in the input images are not simple 1-bit numbers, the XOR operation is normally (but not always) carried out individually on each corresponding bit in the pixel values, in bitwise fashion.

Guidelines for Use

We illustrate the function of XOR using

and

Since logical operators work more reliably with binary input we first threshold the two images, thus obtaining



and

Now, we can use XOR to detect changes in the images, since pixels which didn't change output 0 and pixels which did change result in 1. The image

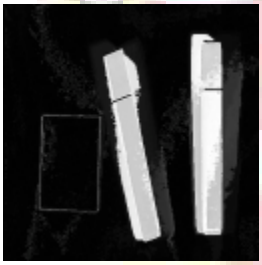
shows the result of XORing the thresholded images. We can see the old and the new position of the moved object, whereas the stationary object almost disappeared from the image. Due to the effects of noise, we can still see some pixels around the boundary of the stationary object, *i.e.* pixels whose values in the original image were close to the threshold.

In a scene like

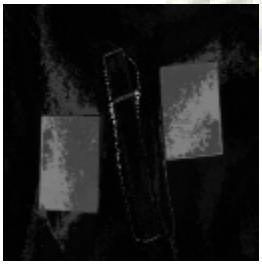
it is not possible to apply a threshold in order to obtain a binary image, since one of the objects is lighter than the background whereas the other one is darker. However, we can combine two grayscale images by XORing them in a bitwise fashion.

shows a scene where the dark object was moved and in

the light object changed its position. XORing each of them with the initial image yields



and



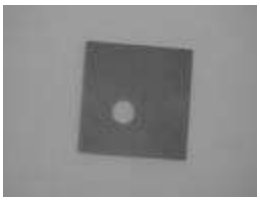
respectively. In both cases, the moved part appears at the old as well as at the new location and the stationary object almost disappears. This technique is based on the assumption that XORing two similar grayvalues produces a low output, whereas two distinct inputs yield a high output. However, this is not always true, *e.g.* XORing 127 and 128 yields 255. These effects can be seen

at the boundary of the stationary object, where the pixels have an intermediate graylevel and might, due to noise, differ slightly between two of the images. Hence, we can see a line with high values around the stationary object. A similar problem is that the output for the moved pen is much higher than the output for the moved piece of paper, although the contrast between their intensities and that of the background value is roughly the same. Because of these problems it is often better to use image subtraction or image division for change detection.

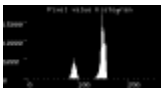
As with other logical operators, XOR and XNOR are often used as sub-components of more complex image processing tasks. XOR has the interesting property that if we XOR *A* with *B* to get *Q*, then the bits of *Q* are the same as *A* where the corresponding bit from *B* is zero, but they are of the opposite value where the corresponding bit from *B* is one. So for instance using binary notation, 1010 XORed with 1100 gives 0110. For this reason, *B* could be thought of as a *bit-reversal mask*. Since the operator is symmetric, we could just as well have treated *A* as the mask and *B* as the original.

Extending this idea to images, it is common to see an 8-bit XOR image mask containing only the pixel values 0 (00000000 binary) and 255 (11111111 binary). When this is XORed pixel-by-pixel with an original image it reverses the bits of pixels values where the mask is 255, and leaves them as they are where the mask is zero. The pixels with reversed bits normally `stand out' against their original color and so this technique is often used to produce a cursor that is visible against an arbitrary colored background. The other advantage of using XOR like this is that to undo the process (for instance when the cursor moves away), it is only necessary to repeat the XOR using the same mask and all the flipped pixels will become unflipped. Therefore it is not necessary to explicitly store the original colors of the pixels affected by the mask. Note that the flipped pixels are not always visible against their unflipped color --- light pixels become dark pixels and dark pixels become light pixels, but middling gray pixels become middling gray pixels!

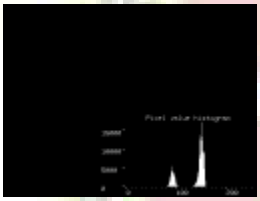
The image



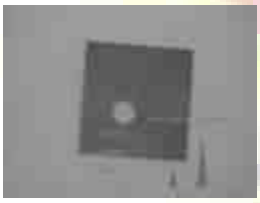
shows a simple graylevel image. Suppose that we wish to overlay this image with its histogram shown in



so that the two can be compared easily. One way is to use XOR. We first use an image editor to enlarge the histogram until it is the same size as the first image. The result is shown in



To perform the overlay we simply XOR this image with the first image in bitwise fashion to produce

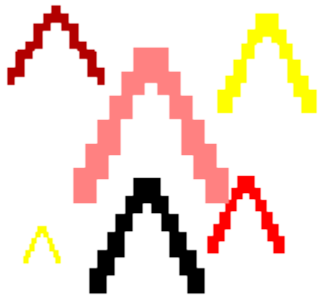


Here, the text is quite easy to read, because the original image consists of large and rather light or rather dark areas. If we proceed in the same way with

we obtain

Note how the writing is dark against light backgrounds and light against dark backgrounds and hardly visible against gray backgrounds. Compare the result with that described under OR. In fact XORing is not particularly good for producing easy to read text on gray backgrounds --- we might do better just to add a constant offset to the image pixels that we wish to highlight (assuming wraparound under addition overflow) --- but it is often used to quickly produce highlighted pixels where the background is just black and white or where legibility is not too important.

Invert/Logical NOT



Common Names: Logical NOT, invert, photographic negative

Brief Description

Logical NOT or *invert* is an operator which takes a binary or graylevel image as input and produces its photographic negative, *i.e.* dark areas in the input image become light and light areas become dark.

How It Works

To produce the photographic negative of a binary image we can employ the logical NOT operator. Its truth-table is shown in Figure 1.

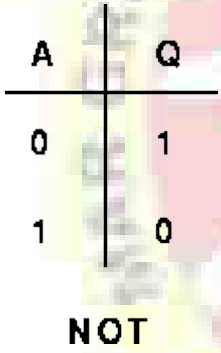


Figure 1 Truth-table for logical NOT.

Each pixel in the input image having a logical 1 (often referred to as foreground) has a logical 0 (associated with the background in the output image and *vice versa*. Hence, applying logical NOT to a binary image changes its polarity.

The logical NOT can also be used for a graylevel image being stored in *byte* pixel format by applying it in a bitwise fashion. The resulting value for each pixel is the input value subtracted from *255*:

$$Q(i,j) = 255 - P(i,j)$$

Some applications of *invert* also support *integer* or *float* pixel format. In this case, we can't use the logical NOT operator, therefore the pixel values of the inverted image are simply given by

$$Q(i,j) = -P(i,j)$$

If this output image is normalized for an 8-bit display, we again obtain the photographic negative of the original input image.

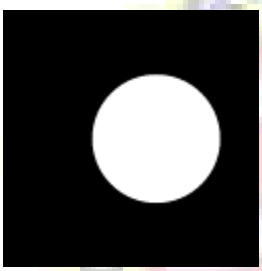
Guidelines for Use

When processing a binary image with a logical or morphological operator, its polarity is often important. Hence, the logical NOT operator is often used to change the polarity of a binary image as a part of some larger process. For example, if we OR

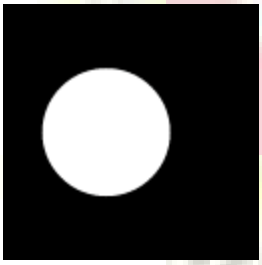
and

the resulting image,

shows the *union* of the background, because it is represented with a logical 1. However, if we OR



and



which are the inverted versions of the above image we obtain

Now, the result contains the union of the two circles.

We illustrate another example of the importance of the polarity of a binary image using the dilation operator. Dilation expands all white areas in a binary image. Hence, if we dilate

the object, being represented with a logical 1, grows and the holes in the object shrink. We obtain

If we dilate

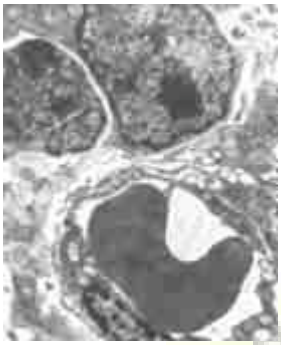
which was obtained by applying logical NOT to the original image, we get

Here, the background is expanded and the object became smaller.

Invert can be used for the same purpose on grayscale images, if they are processed with a morphological or logical operator.

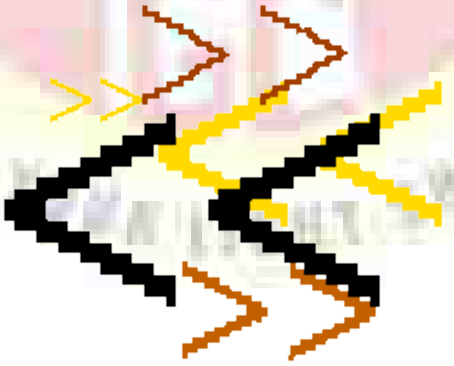
Invert is also used to print the photographic negative of an image or to make the features in an image appear clearer to a human observer. This can, for example, be useful for medical images, where the objects often appear in black on a white background. Inverting the image makes the

objects appear in white on a dark background, which is often more suitable for the human eye. From the original image



of a tissue slice, we obtain the photographic negative

Bitshift Operators



Common Names: Bitshifting

Brief Description

The bitshift operator works on images represented in byte or integer pixel format, where each pixel value is stored as a *binary* number with a fixed amount of bits. Bitshifting shifts the binary

representation of each pixel to the left or to the right by a pre-defined number of positions. Shifting a binary number by one bit is equivalent to multiplying (when shifting to the left) or dividing (when shifting to the right) the number by *2*.

How It Works

The operation is performed straightforwardly in a single pass. If the binary representation of a number is shifted in one direction, we obtain an empty position on the opposite side. There are generally three possibilities of how to fill in this empty position: we can pad the empty bits with a *0* or a *1* or we can wrap around the bits which are shifted out of the binary representation of the number on the other side. The last possibility is equivalent to *rotating* the binary number.

The choice of technique used depends on the implementation of the operator and on the application. In most cases, bitshifting is used to implement a fast multiplication or division. In order to obtain the right results for this application, we have to pad the empty bits with a *0*. Only in the case of dividing a negative number by a power of *2*, do we need to fill the left bits with a *1*, because a negative number is represented as the *two's-complement* of the positive number, *i.e.* the sign bit is a *1*. The result of applying bitshifting in this way is illustrated in the following formula:

$$\begin{aligned} \text{Shifting } i \text{ bits to the right} &\Leftrightarrow Q(i,j) = P(i,j) \div 2^i \\ \text{Shifting } i \text{ bits to the left} &\Leftrightarrow Q(i,j) = P(i,j) \times 2^i \end{aligned}$$

An example is shown in Figure 1.

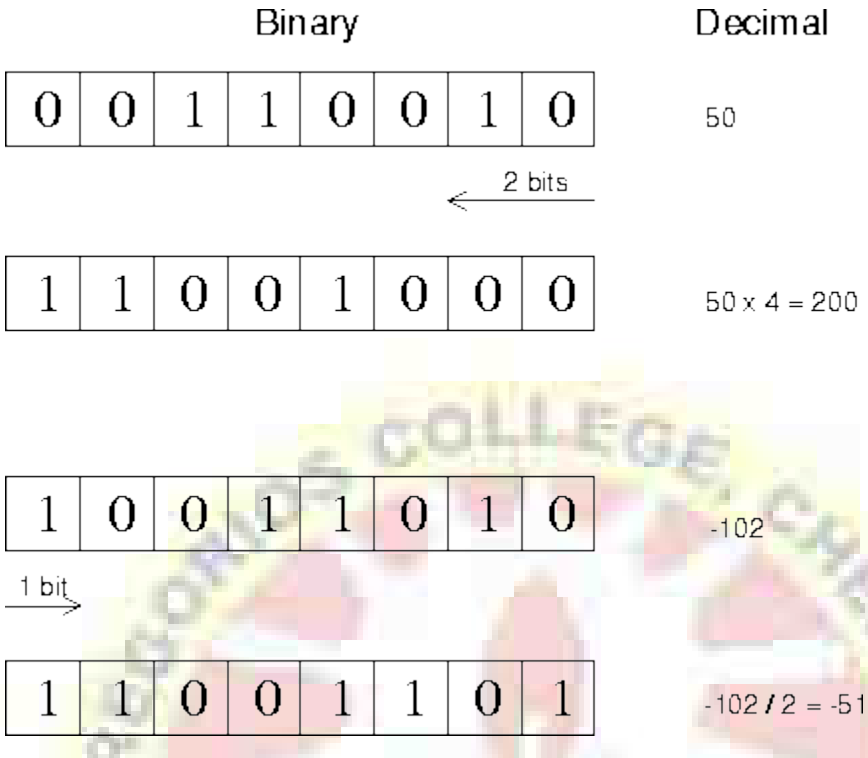


Figure 1 Examples for using bitshifting for multiplication and division. Note that the bottom example uses a signed-byte convention where a byte represents a number between -128 and +127

If bitshifting is used for multiplication, it might happen that the result exceeds the maximum possible pixel value. This is the case when a *1* is shifted out of the binary representation of the pixel value. This information is lost and the effect is that the value is wrapped around from zero.

Guidelines for Use

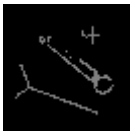
The main application for the bitshift operator is to divide or multiply an image by a power of *2*. The advantage over the normal pixel division and pixel multiplication operators is that bitshifting is computationally less expensive.

For example, if we want to add two images we can use bitshifting to make sure that the result will not exceed the maximum pixel value. We illustrate this example using

and

where the latter is the skeleton gained from the thresholded version of the former. To better visualize the result of the skeletonization we might want to overlay these two images. However, if we add them straightforwardly we obtain pixel values greater than the maximum value. First shifting both images to the right by one bit yields

and



which then can be added without causing any overflow problems. The result can be seen in

Here, we can see that shifting a pixel to the right does, as a normal pixel division, decrease the contrast in the image.

On the other hand, shifting the binary representation of a pixel to the left increases the image contrast, like the pixel multiplication. For example,

is an image taken under poor lighting conditions. Shifting each pixel in the image to the left by one bit, which is identical to multiplying it with *2*, yields

Although the operator worked well in this example, we have to be aware that the result of the multiplication might exceed the maximum pixel value. Then, the effect for the pixel value is that it is wrapped around from *0*. For example, if we shift each pixel in the above image by two bits, at some pixels a *1* is shifted out of the binary representation of the image, resulting in a loss of information. This can be seen in

In general, we should make sure that the values in the input image are sufficiently small or we have to be careful when we interpret the resulting image. Alternatively, we can change the pixel value format prior to applying the bitshift operator, *e.g.* change from *byte* format to *integer* format.

Although multiplication and division are the main applications for bitshifting it might also be used for other, often very specialized, purposes. For example, we can store two *4-bit* images in a byte array if we shift one of the two images by *4* bits and mask out the unused bits. Using the logical OR operator we can combine the two images into one without losing any information. Sometimes it might also be useful to rotate the binary representation of each bit, apply some other operator to the image and finally rotate the pixels back to the initial order.

Spatial Filtering and its Type

Spatial Filtering technique is used directly on pixels of an image. Mask is usually considered to be added in size so that it has specific center pixel. This mask is moved on the image such that the center of the mask traverses all image pixels.

Classification on the basis of linearity:

There are two types:

1. Linear Spatial Filter

2. Non-linear Spatial Filter

General Classification:

Smoothing Spatial Filter: Smoothing filter is used for blurring and noise reduction in the image. Blurring is pre-processing steps for removal of small details and Noise Reduction is accomplished by blurring.

Types of Smoothing Spatial Filter:

1. Linear Filter (Mean Filter)

2. Order Statistics (Non-linear) filter

These are explained as following below.

1. **Mean Filter:**

   Linear spatial filter is simply the average of the pixels contained in the neighborhood of the

filter mask. The idea is replacing the value of every pixel in an image by the average of the grey levels in the neighborhood define by the filter mask.

Types of Mean filter:

- **(i) Averaging filter:** It is used in reduction of the detail in image. All coefficients are equal.
- **(ii) Weighted averaging filter:** In this, pixels are multiplied by different coefficients. Center pixel is multiplied by a higher value than average filter.

2. **Order Statistics Filter:**

It is based on the ordering the pixels contained in the image area encompassed by the filter. It replaces the value of the center pixel with the value determined by the ranking result. Edges are better preserved in this filtering.

Types of Order statistics filter:

- **(i) Minimum filter:** 0th percentile filter is the minimum filter. The value of the center is replaced by the smallest value in the window.
- **(ii) Maximum filter:** 100th percentile filter is the maximum filter. The value of the center is replaced by the largest value in the window.
- **(iii) Median filter:** Each pixel in the image is considered. First neighboring pixels are sorted and original values of the pixel is replaced by the median of the list.

Sharpening Spatial Filter: It is also known as derivative filter. The purpose of the sharpening spatial filter is just the opposite of the smoothing spatial filter. Its main focus in on the removal of blurring and highlight the edges. It is based on the first and second order derivative.

First order derivative:

- Must be zero in flat segments.
- Must be non zero at the onset of a grey level step.
- Must be non zero along ramps.

First order derivative in 1-D is given by:

$f' = f(x+1) - f(x)$

Second order derivative:

- Must be zero in flat areas.

- Must be zero at the onset and end of a ramp.
- Must be zero along ramps.

Second order derivative in 1-D is given by:

f'' = f(x+1) + f(x-1) - 2f(x)

Smoothing Images

Goals

Learn to:

- Blur images with various low pass filters
- Apply custom-made filters to images (2D convolution)

2D Convolution (Image Filtering)

As in one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. LPF helps in removing noise, blurring images, etc. HPF filters help in finding edges in images.

OpenCV provides a function **cv.filter2D()** to convolve a kernel with an image. As an example, we will try an averaging filter on an image. A 5x5 averaging filter kernel will look like the below:

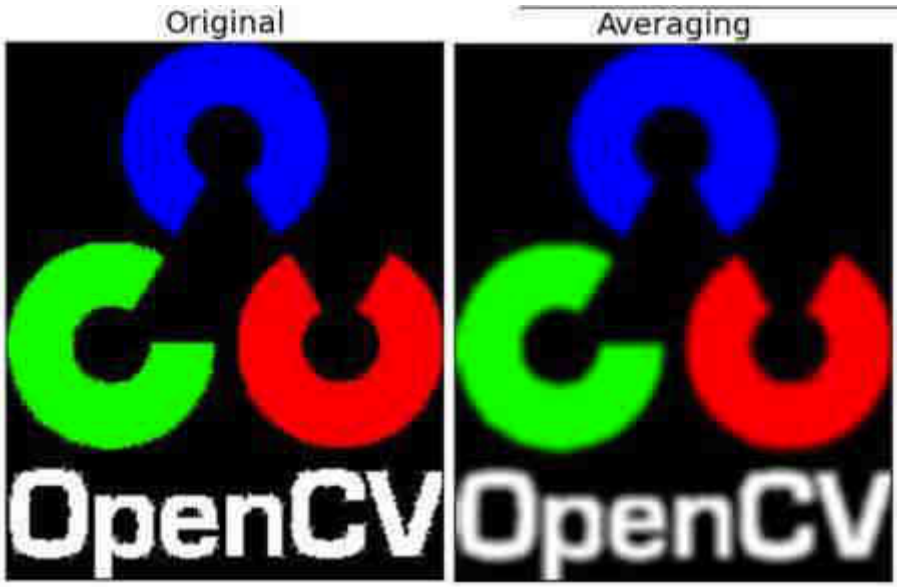
K=125⎡ ⌊ ⎢ ⎢

The operation works like this: keep this kernel above a pixel, add all the 25 pixels below this kernel, take the average, and replace the central pixel with the new average value. This operation is continued for all the pixels in the image. Try this code and check the result:

```
import numpy as np
import cv2 as cv
```

```
from matplotlib import pyplot as plt

img = cv.imread('opencv_logo.png')

kernel = np.ones((5,5),np.float32)/25

dst = cv.filter2D(img,-1,kernel)

plt.subplot(121),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(dst),plt.title('Averaging')

plt.xticks([]), plt.yticks([])

plt.show()
```

Result:



image

Image Blurring (Image Smoothing)

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image.

So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges). OpenCV provides four main types of blurring techniques.

1. Averaging

This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element. This is done by the function **cv.blur()** or **cv.boxFilter()**. Check the docs for more details about the kernel. We should specify the width and height of the kernel. A 3x3 normalized box filter would look like the below:

K=19[[| 111111111]] |

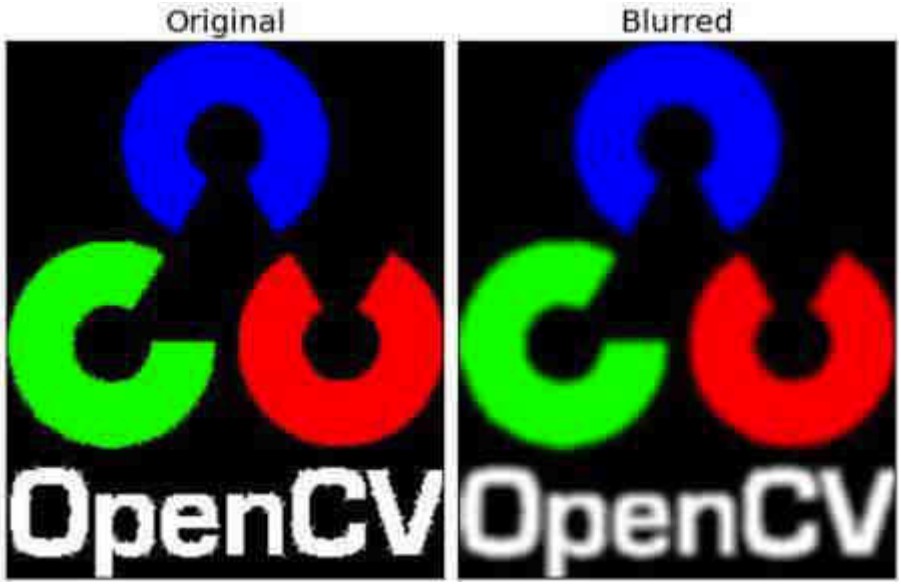
Note

> If you don't want to use a normalized box filter, use **cv.boxFilter()**. Pass an argument normalize=False to the function.

Check a sample demo below with a kernel of 5x5 size:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('opencv-logo-white.png')
blur = cv.blur(img,(5,5))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

Result:



image

2. Gaussian Blurring

In this method, instead of a box filter, a Gaussian kernel is used. It is done with the function, **cv.GaussianBlur()**. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.

If you want, you can create a Gaussian kernel with the function, **cv.getGaussianKernel()**.

The above code can be modified for Gaussian blurring:

```
blur = cv.GaussianBlur(img,(5,5),0)
```

Result:

image

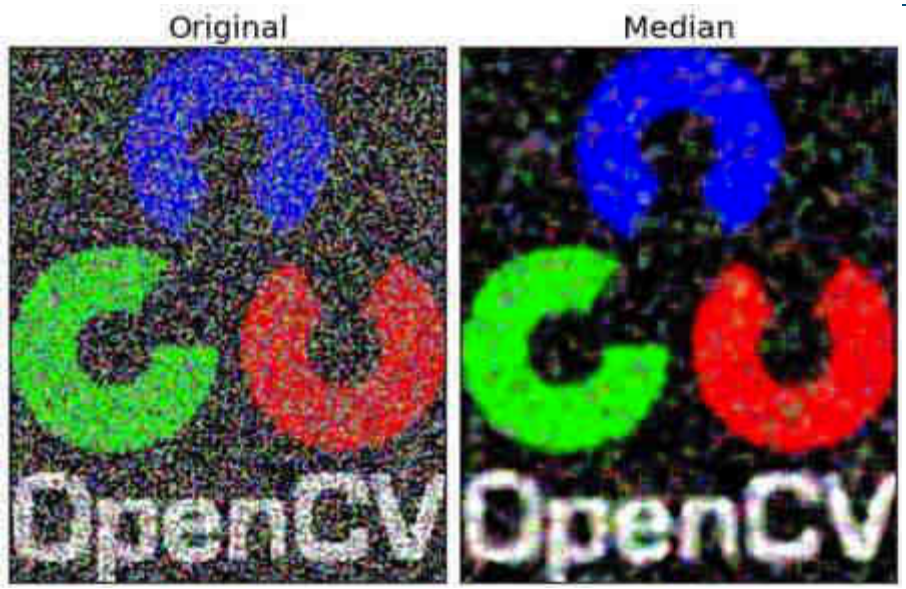
3. Median Blurring

Here, the function **cv.medianBlur()** takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer.

In this demo, I added a 50% noise to our original image and applied median blurring. Check the result:

```
median = cv.medianBlur(img,5)
```

Result:



image

4. Bilateral Filtering

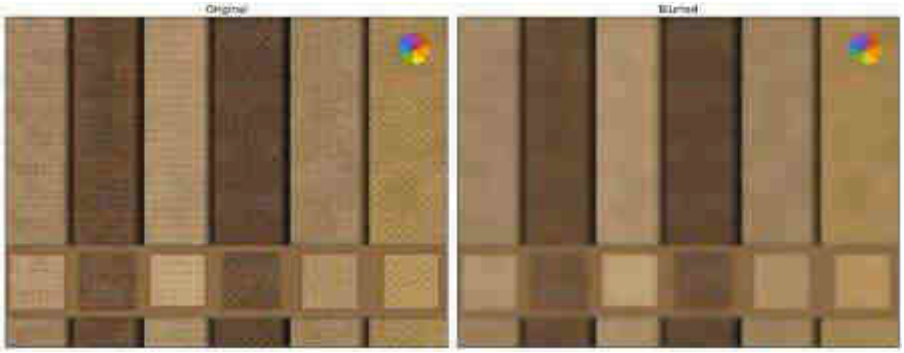
cv.bilateralFilter() is highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity. It doesn't consider whether a pixel is an edge pixel or not. So it blurs the edges also, which we don't want to do.

Bilateral filtering also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.

The below sample shows use of a bilateral filter (For details on arguments, visit docs).

```
blur = cv.bilateralFilter(img,9,75,75)
```

Result:



image

See, the texture on the surface is gone, but the edges are still preserved.

UNIT III

I

mage enhancement in Frequency domain

Frequency domain analysis

Till now, all the domains in which we have analyzed a signal , we analyze it with respect to time. But in frequency domain we don't analyze signal with respect to time, but with respect of frequency.

Difference between spatial domain and frequency domain

In spatial domain, we deal with images as it is. The value of the pixels of the image change with respect to scene. Whereas in frequency domain, we deal with the rate at which the pixel values are changing in spatial domain.

For simplicity, Let's put it this way.

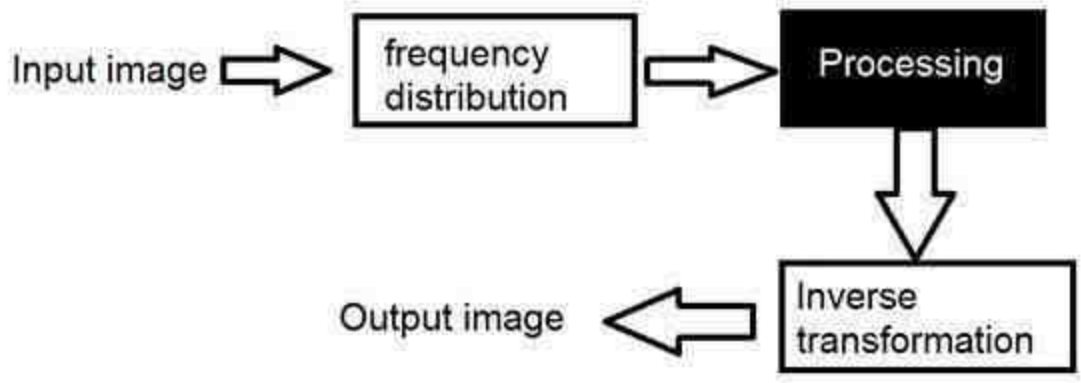
Spatial domain

In simple spatial domain, we directly deal with the image matrix. Whereas in frequency domain, we deal an image like this.

Frequency Domain

We first transform the image to its frequency distribution. Then our black box system perform what ever processing it has to performed, and the output of the black box in this case is not an image, but a transformation. After performing inverse transformation, it is converted into an image which is then viewed in spatial domain.

It can be pictorially viewed as



Here we have used the word transformation. What does it actually mean?

Transformation

A signal can be converted from time domain into frequency domain using mathematical operators called transforms. There are many kind of transformation that does this. Some of them are given below.

- Fourier Series
- Fourier transformation
- Laplace transform
- Z transform

Out of all these, we will thoroughly discuss Fourier series and Fourier transformation in our next tutorial.

Frequency components

Any image in spatial domain can be represented in a frequency domain. But what do this frequencies actually mean.

We will divide frequency components into two major components.

High frequency components

High frequency components correspond to edges in an image.

Low frequency components

Low frequency components in an image correspond to smooth regions.

Fourier

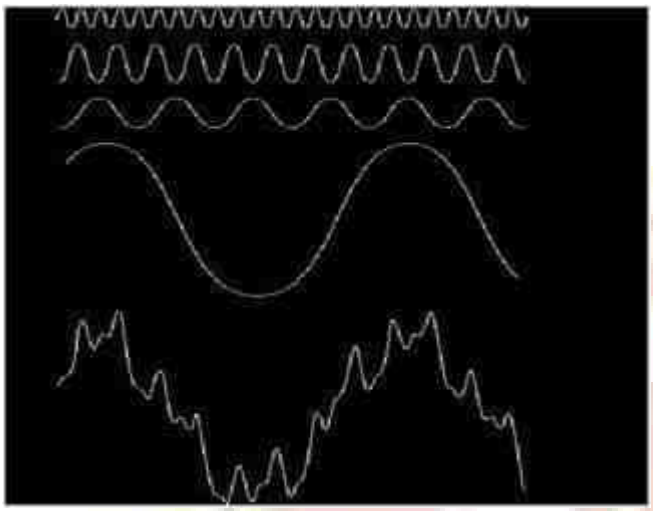
Fourier was a mathematician in 1822. He give Fourier series and Fourier transform to convert a signal into frequency domain.

Fourier Series

Fourier series simply states that, periodic signals can be represented into sum of sines and cosines when multiplied with a certain weight.It further states that periodic signals can be broken down into further signals with the following properties.

- The signals are sines and cosines
- The signals are harmonics of each other

It can be pictorially viewed as



In the above signal, the last signal is actually the sum of all the above signals. This was the idea of the Fourier.

How it is calculated

Since as we have seen in the frequency domain, that in order to process an image in frequency domain, we need to first convert it using into frequency domain and we have to take inverse of the output to convert it back into spatial domain. That's why both Fourier series and Fourier transform has two formulas. One for conversion and one converting it back to the spatial domain.

Fourier series

The Fourier series can be denoted by this formula.

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(ux+vy)} \, dx \, d$$

The inverse can be calculated by this formula.

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{j2\pi(ux+vy)} \, du \, dv.$$

Fourier transform

The Fourier transform simply states that that the non periodic signals whose area under the curve is finite can also be represented into integrals of the sines and cosines after being multiplied by a certain weight.

The Fourier transform has many wide applications that include, image compression (e.g JPEG compression), filtering and image analysis.

Difference between Fourier series and transform

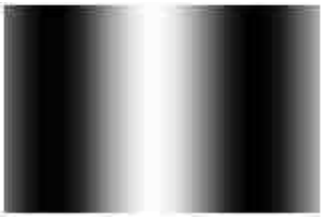
Although both Fourier series and Fourier transform are given by Fourier , but the difference between them is Fourier series is applied on periodic signals and Fourier transform is applied for non periodic signals

Which one is applied on images

Now the question is that which one is applied on the images , the Fourier series or the Fourier transform. Well, the answer to this question lies in the fact that what images are. Images are non – periodic. And since the images are non periodic, so Fourier transform is used to convert them into frequency domain.

Discrete fourier transform

Since we are dealing with images, and in fact digital images, so for digital images we will be working on discrete fourier transform



Consider the above Fourier term of a sinusoid. It include three things.

- Spatial Frequency
- Magnitude
- Phase

The spatial frequency directly relates with the brightness of the image. The magnitude of the sinusoid directly relates with the contrast. Contrast is the difference between maximum and minimum pixel intensity. Phase contains the color information.

The formula for 2 dimensional discrete Fourier transform is given below.

$$F(u,v)=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(ux/M+vy/N)}$$

The discrete Fourier transform is actually the sampled Fourier transform, so it contains some samples that denotes an image. In the above formula f(x,y) denotes the image, and F(u,v) denotes the discrete Fourier transform. The formula for 2 dimensional inverse discrete Fourier transform is given below.

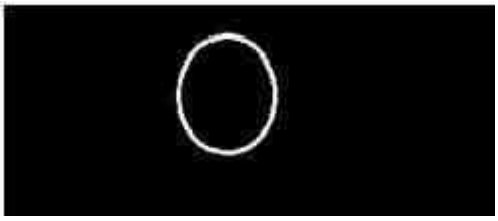
$$f(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi(ux/M+vy/N)}$$

The inverse discrete Fourier transform converts the Fourier transform back to the image

Consider this signal

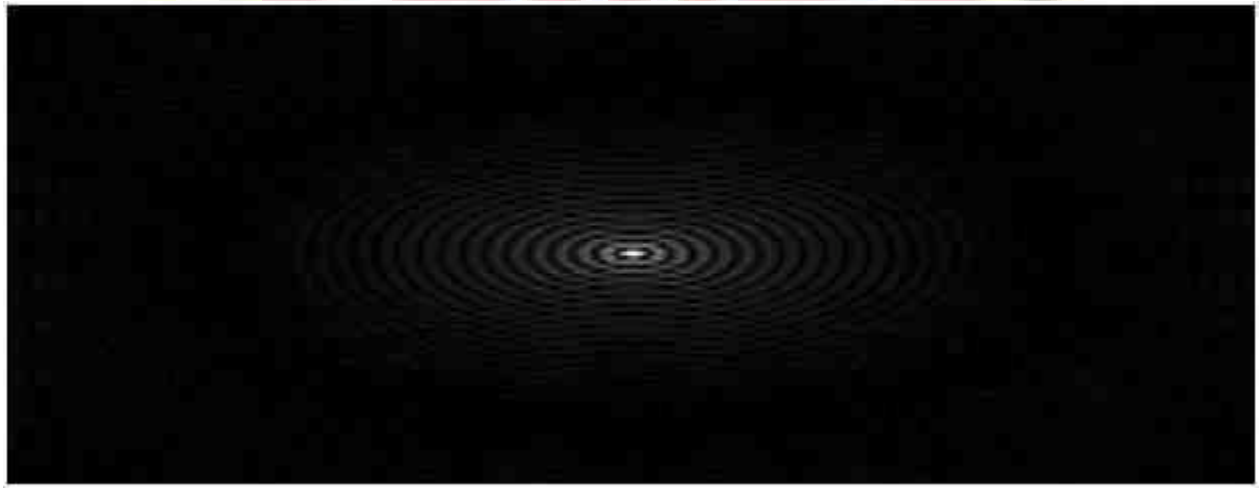
Now we will see an image, whose we will calculate FFT magnitude spectrum and then shifted FFT magnitude spectrum and then we will take Log of that shifted spectrum.

Original Image

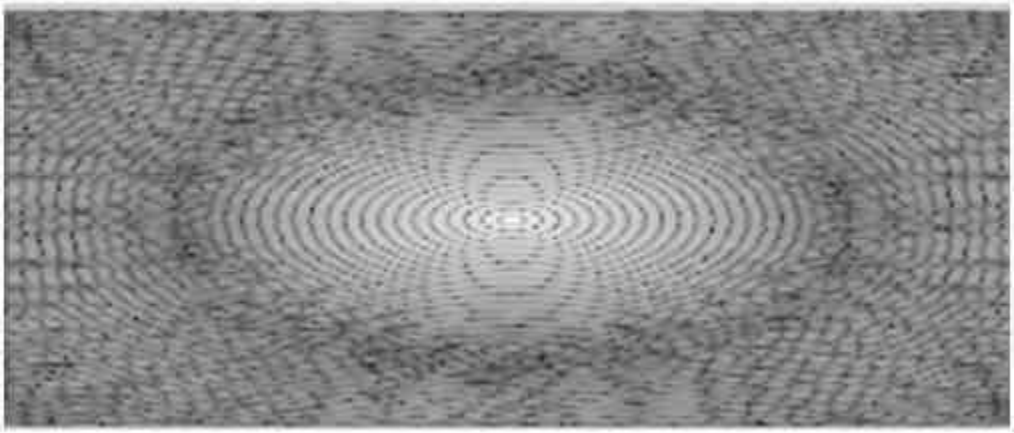


The Fourier transform magnitude spectrum

The Shifted Fourier transform

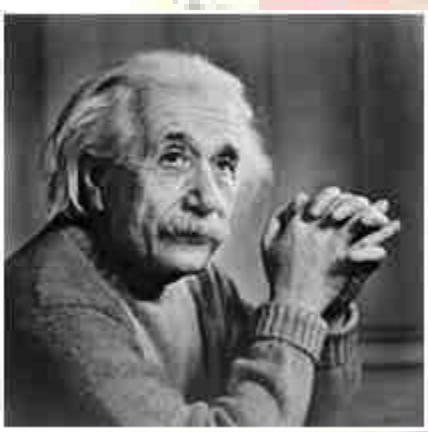


The Shifted Magnitude Spectrum



For example

Consider this example.



The same image in the frequency domain can be represented as.

Now what's the relationship between image or spatial domain and frequency domain. This relationship can be explained by a theorem which is called as Convolution theorem.

Convolution Theorem

The relationship between the spatial domain and the frequency domain can be established by convolution theorem.

The convolution theorem can be represented as.

$$f(x,y) * h(x,y) \leftrightarrow F(u,v)H(u,v)$$

$$f(x,y)h(x,y) \leftrightarrow F(u,v) * H(u,v)$$

$$h(x,y) \leftrightarrow H(u,v)$$

It can be stated as the convolution in spatial domain is equal to filtering in frequency domain and vice versa.

The filtering in frequency domain can be represented as following:

The steps in filtering are given below.

- At first step we have to do some pre – processing an image in spatial domain, means increase its contrast or brightness
- Then we will take discrete Fourier transform of the image
- Then we will center the discrete Fourier transform, as we will bring the discrete Fourier transform in center from corners
- Then we will apply filtering, means we will multiply the Fourier transform by a filter function
- Then we will again shift the DFT from center to the corners
- Last step would be take to inverse discrete Fourier transform, to bring the result back from frequency domain to spatial domain
- And this step of post processing is optional, just like pre processing , in which we just increase the appearance of image.

Filters

The concept of filter in frequency domain is same as the concept of a mask in convolution.

After converting an image to frequency domain, some filters are applied in filtering process to perform different kind of processing on an image. The processing include blurring an image, sharpening an image e.t.c.

The common type of filters for these purposes are:

- Ideal high pass filter
- Ideal low pass filter
- Gaussian high pass filter
- Gaussian low pass filter

In the next tutorial, we will discuss about filter in detail.

Blurring masks vs derivative masks

We are going to perform a comparison between blurring masks and derivative masks.

Blurring masks

A blurring mask has the following properties.

- All the values in blurring masks are positive
- The sum of all the values is equal to 1
- The edge content is reduced by using a blurring mask
- As the size of the mask grow, more smoothing effect will take place

Derivative masks

A derivative mask has the following properties.

- A derivative mask have positive and as well as negative values
- The sum of all the values in a derivative mask is equal to zero
- The edge content is increased by a derivative mask
- As the size of the mask grows , more edge content is increased

Relationship between blurring mask and derivative mask with high pass filters and low pass filters.

The relationship between blurring mask and derivative mask with a high pass filter and low pass filter can be defined simply as.

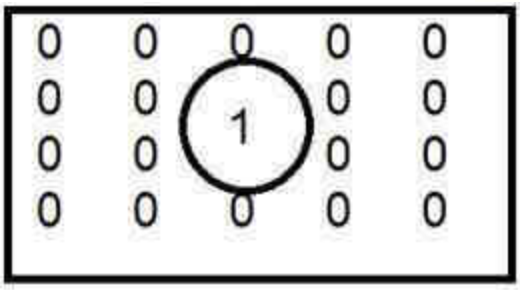
- Blurring masks are also called as low pass filter
- Derivative masks are also called as high pass filter

High pass frequency components and Low pass frequency components

The high pass frequency components denotes edges whereas the low pass frequency components denotes smooth regions.

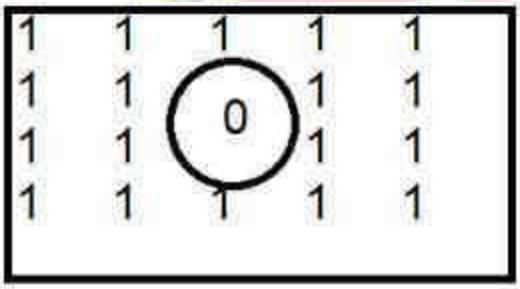
Ideal low pass and Ideal High pass filters

This is the common example of low pass filter.

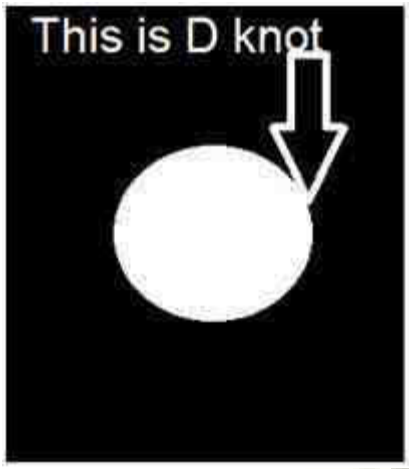


When one is placed inside and the zero is placed outside , we got a blurred image. Now as we increase the size of 1, blurring would be increased and the edge content would be reduced.

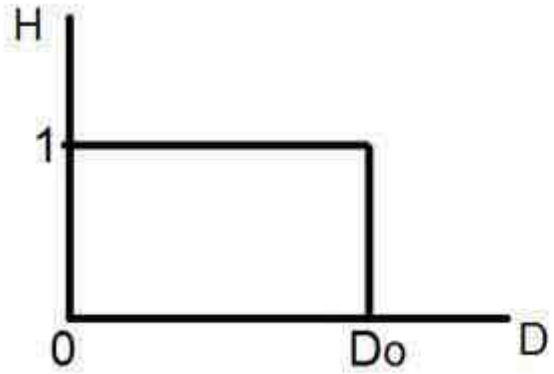
This is a common example of high pass filter.



When 0 is placed inside, we get edges, which gives us a sketched image. An ideal low pass filter in frequency domain is given below.



The ideal low pass filter can be graphically represented as



Now let's apply this filter to an actual image and let's see what we got.

Sample image

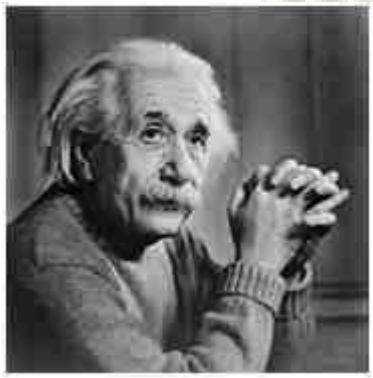
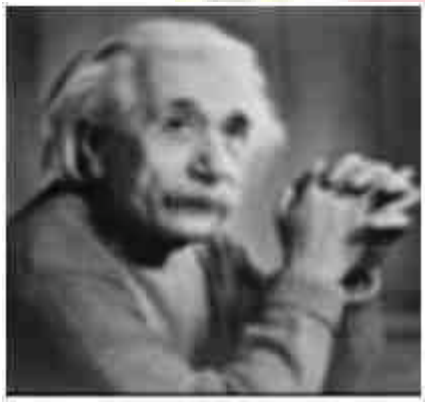


Image in frequency domain

Applying filter over this image

Resultant Image



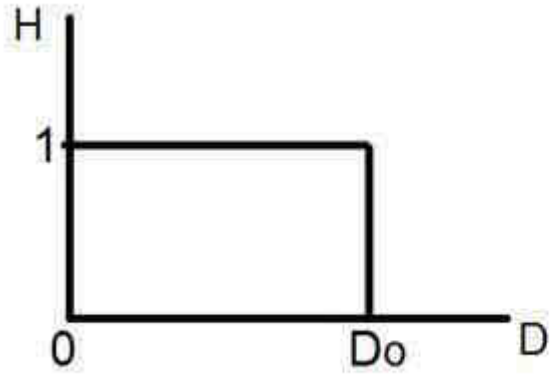
With the same way, an ideal high pass filter can be applied on an image. But obviously the results would be different as, the low pass reduces the edged content and the high pass increase it.

Gaussian Low pass and Gaussian High pass filter

Gaussian low pass and Gaussian high pass filter minimize the problem that occur in ideal low pass and high pass filter.

This problem is known as ringing effect. This is due to reason because at some points transition between one color to the other cannot be defined precisely, due to which the ringing effect appears at that point.

Have a look at this graph.



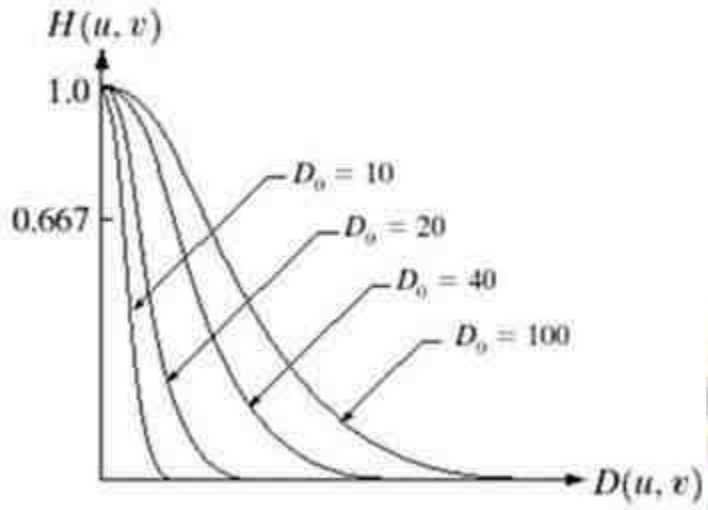
This is the representation of ideal low pass filter. Now at the exact point of Do, you cannot tell that the value would be 0 or 1. Due to which the ringing effect appears at that point.

So in order to reduce the effect that appears is ideal low pass and ideal high pass filter, the following Gaussian low pass filter and Gaussian high pass filter is introduced.

Gaussian Low pass filter

The concept of filtering and low pass remains the same, but only the transition becomes different and become more smooth.

The Gaussian low pass filter can be represented as



Note the smooth curve transition, due to which at each point, the value of Do, can be exactly defined.

Gaussian high pass filter

Gaussian high pass filter has the same concept as ideal high pass filter, but again the transition is more smooth as compared to the ideal one.

Continuous 1D Fourier Transform

The Fourier Series previously considered is intended for use with periodic signals. More general signals may exhibit some locally periodic components, but are not, in general, periodic. The Fourier transform and the inverse Fourier transform allow for the conversion of any signal to the frequency domain and back again to either the time or spatial domain.

We consider one dimensional signals only as steps towards the 2-D Fourier transform of images. One dimensional continuous and discrete signals provide simpler cases for learning some of the important properties of frequency domain data.

Fourier Transform

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\omega t}\, dt$$

Where ω is a continuous frequency in radians/sec.

Inverse Fourier Transform

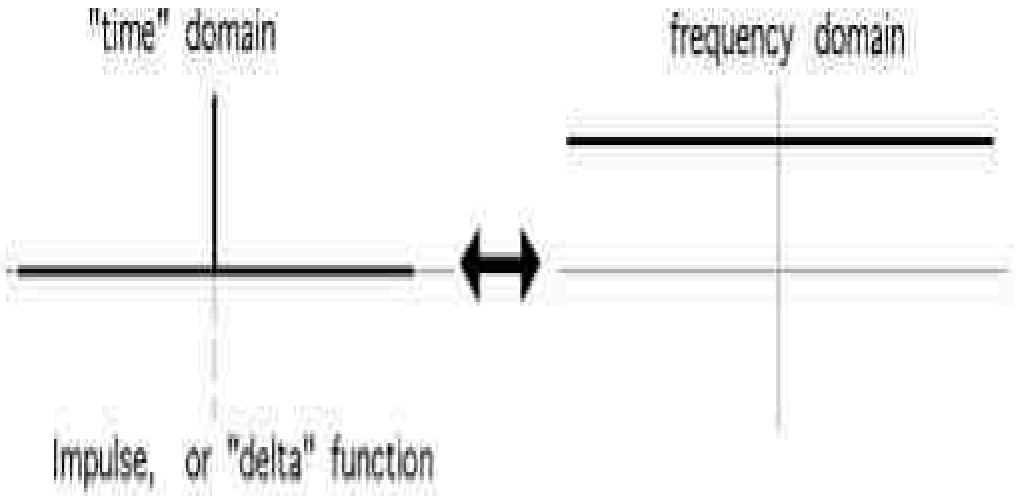
$$x'(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)\, e^{j\omega t}\, d\omega$$

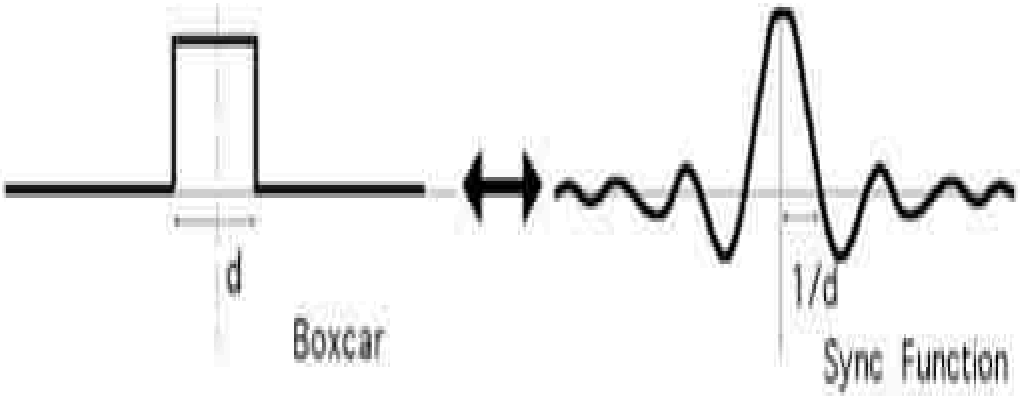
Properties

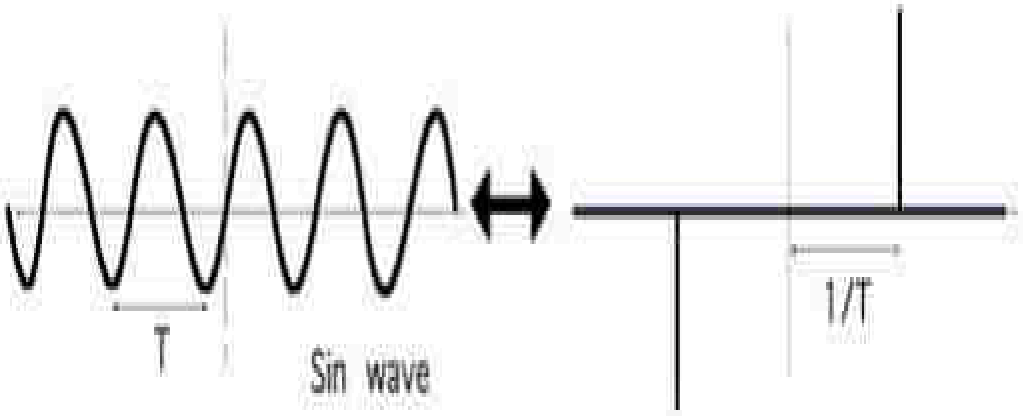
- $x(t)$ is continuous and may be complex.
- $X(\omega)$ is complex and continuous.
- $x'(t)$ is complex, but if $x(t)$ was real, then $x(t) = real\, x'(t)$.
- Although it is a bit hard to think about negative frequencies, they are a mathematical necessity. This is because of the cyclical nature of the complex exponential. The values of the complex exponential over the range 0 to 2π are the same as for the equivalent range $-\pi$ to π.

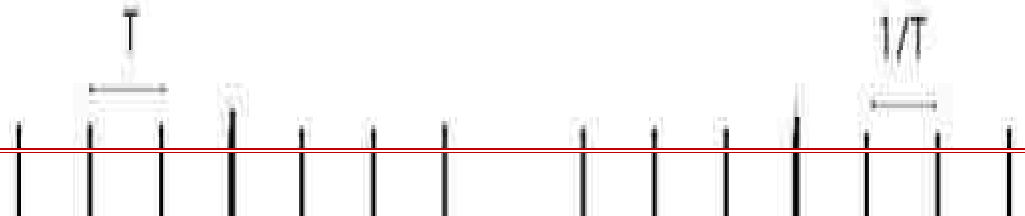
Common Fourier Transform Pairs

Below are some common frequency and time domain pairs. These results will be useful to us for explaining other properties later.


"time" domain
frequency domain
Impulse, or "delta" function


d
Boxcar
1/d
Sync Function


T
Sin wave
1/T


T
1/T

The Complex Exponential Basis Function

Note

Recall that a complex number is one containing a real component and an imaginary component. Imaginary numbers are multiplied by the imaginary number, $j = \sqrt{-1}$.

$$C = a + j\,b$$

Mathematicians usually use the variable i as the imaginary number. Engineers, however, prefer to use the variable j. Equations involving complex numbers often involve variables representing voltage and current in a circuit. In electrical circuits, i is always current, so to avoid confusion, the variable j is used for the imaginary number.

As we saw, the Fourier Series generates a periodic signal as a sum of weighted sinusoidal signals. The Fourier transform extends the Fourier series to convert any continuously integrable signals into the frequency domain. The coefficients in the frequency domain are complex numbers to quantify both the magnitude and phase of the spectrum over a range of frequencies. In both the Fourier transform (FT) and inverse Fourier transform (IFT), we use the complex exponential basis function for the sinusoidal foundation of the transforms. The coefficients are calculated as a sum of products between the signal and the set of complex, sinusoidal basis functions. The complex exponential basis function is defined by what is called Euler's formula.

Leonhard Euler (1707 - 1783)

$$e^{j\,\theta} = \cos(\theta) + j\,\sin(\theta)$$

Euler What?!

Most people find Euler's formula quite puzzling when they first see it. We know that j is the imaginary number $\sqrt{-1}$, but what is e and what does it have to do with the $\cos$ and $\sin$?

The number e

Like the number π, the number $e \approx 2.718281828$ is a constant irrational number with a significant influence on the mathematics of how things work. The value of e is defined in terms of a limit.

Here are the limits of exponentials that we already know about and also the limits defining the number e and also e^x.

$$\lim_{n\to\infty} a^n = 0, \qquad 0 < a < 1$$

$$\lim_{n\to\infty} a^n = 1, \qquad a = 1$$

$$\lim_{n\to\infty} (1 + 1/n)^n = e$$

$$\lim_{n\to\infty} (1 + x/n)^n = e^x, \quad \forall x$$

$$\lim_{n\to\infty} a^n = \infty, \qquad a > 1$$

The function e^x has a couple properties that make it special.

- e^x is the only known function that the derivative of the function is itself. This causes e^x to be in the solution to many differential equation problems. Some examples where you will find e^x are: the voltage on a capacitor as a function of time, the decay of radioactivity of the nucleus of an unstable atom, and the probability density function of a Gaussian random variable.
- Euler's formula, which we will tackle next.

Derivation of Euler's Formula

We can see how $e^{j\,x}$ relates to the $\sin(x)$ and $\cos(x)$ functions by looking at the MacLaurin series for these functions. A MacLaurin series is a Taylor series expansion about at the point zero.

If you are like I was as an undergraduate, Taylor and MacLaurin series expansions where among my least favorite parts of calculus class. However, they have two important virtues.

1. They are useful for numeric calculations. Many years ago, before the days of computers and calculators, tables were used to find the value of many math functions. The people with the jobs of computing these tables often used Taylor and MacLaurin series to calculate the values in the tables. Calculators and computers might also use these series for internally calculating the values of some functions.
2. There are some mathematical properties of functions that can only be observed by considering these series. This is the case with Euler's formula for complex exponentials.

The needed MacLaurin series are:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Now replace x in the equations for e^x with jx. Remember that $j^2 = -1$.

$$\begin{aligned} e^{jx} &= 1 + jx + \frac{(jx)^2}{2!} + \frac{(jx)^3}{3!} + \frac{(jx)^4}{4!} + \frac{(jx)^5}{5!} + \frac{(jx)^6}{6!} + \frac{(jx)^7}{7!} + \dots \\ &= 1 + jx - \frac{x^2}{2!} - \frac{jx^3}{3!} + \frac{x^4}{4!} + \frac{jx^5}{5!} - \frac{x^6}{6!} - \frac{jx^7}{7!} + \dots \\ &= \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots\right) + j\left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots\right) \\ &= \cos x + j \sin x \end{aligned}$$

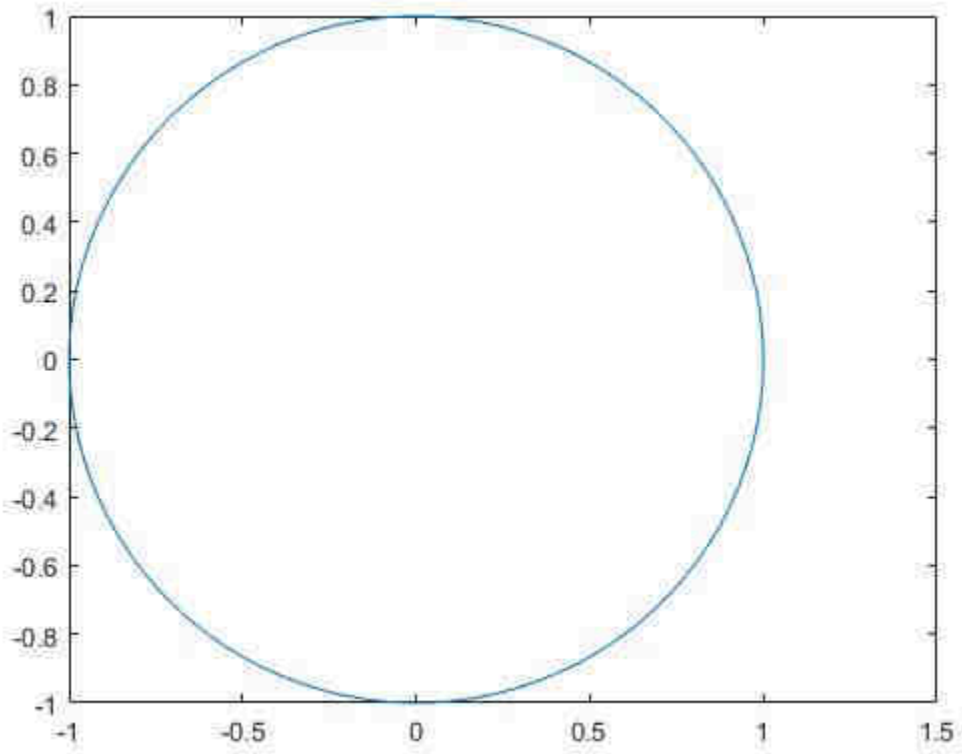
Numerical Proof

Okay, if the derivation from the MacLaurin series didn't convince you, we can try some numerical analysis to show that complex exponentials can actually produce complex, sinusoidal functions?

We can use MATLAB and the definition of e^x from a limit and verify if Euler knew what he was talking about or not. In the following MATLAB script, we just assign n to be a fairly large number. Since the definition uses a limit as n goes to infinity, the results become more accurate when larger value for n are used.

```
% let n = some big number
n=100000;
z=linspace(0,2*pi);% test 100 numbers between 0 and 2*pi
```

```
% Now show that e^jz = cos(z) + j*sin(z)
% Begin with definition of value of e^z.
%    e^z = lim(n = infinity) (1 + z/n)^n
eulerValues=complex(1,z/n).^n;

% This should plot a unit circle, just like
% figure, plot( cos(z), sin(z));
figure,plot(real(eulerValues),imag(eulerValues));
```



Points along a complex unit circle - Numerical proof of Euler's formula

Note

One might reasonably suspect that MATLAB, being aware of Euler's formula, cheated on calculating the complex exponential. That is, it probably converted the complex number to polar coordinates before doing the exponential calculation.

$$\text{let } C = r\,e^{j\theta}$$
$$\text{then, } C^n = r^n\,e^{j\,n\theta}$$

To complete the numerical proof, we can write a MATLAB function to do the complex exponential calculation by brute force. We will do this as a class activity and we will see the same result, which demonstrates the numerical validity of Euler's formula.

Two-Dimensional Fourier Transform

Fourier transform can be generalized to higher dimensions. For example, many signals $f(x,y)$ are functions of 2D space defined over an x-y plane. Two-dimensional Fourier transform also has four different forms depending on whether the 2D signal is periodic and discrete.

- **Aperiodic, continuous signal, continuous, aperiodic spectrum**

$$F(u,v) = \int\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}dx\,dy$$

$$f(x,y) = \int\int_{-\infty}^{\infty} F(u,v)e^{j2\pi(ux+vy)}du\,dv$$

where u and v are spatial frequencies in x and y directions, respectively, and $F(u,v)$ is the 2D spectrum of $f(x,y)$.

- **Aperiodic, discrete signal, continuous, periodic spectrum**

$$F(u,v) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m,n] e^{-j2\pi(umx_o + vny_o)}$$

$$f[m,n] = \frac{1}{UV} \int_0^U \int_0^V F(u,v) e^{j2\pi(umx_o + vny_o)} du\, dv$$

where x_o and y_o are the spatial intervals between consecutive signal samples in the x and y directions, respectively, and $U = 1/x_o$ and $V = 1/y_o$ are sampling rates in the two directions, and they are also the periods of the spectrum $F(u,v)$.

- **Periodic, continuous signal, discrete, aperiodic spectrum**

$$F[k,l] = \frac{1}{XY}\int_0^X \int_0^Y f_{XY}(x,y) e^{j2\pi(kxu_o + lyv_o)} dx\, dy$$

$$f_{XY}(x,y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} F[k,l] e^{-j2\pi(xku_o + ylv_o)}$$

where X and Y are periods of the signal in x and y directions, respectively, and $u_o = 1/X$ and $v_o = 1/Y$ are the intervals between consecutive samples in the spectrum $F[k,l]$.

- **Periodic, discrete signal, discrete and periodic spectrum**

$$F[k,l] = \frac{1}{\sqrt{MN}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[m,n] e^{-j2\pi(\frac{mk}{M} + \frac{nl}{N})}$$

$$f[m,n] = \frac{1}{\sqrt{MN}} \sum_{l=0}^{N-1} \sum_{k=0}^{M-1} F[k,l] e^{j2\pi(\frac{mk}{M} + \frac{nl}{N})}$$

$$(0 \le m,k \le M-1, \quad 0 \le n,l \le N-1)$$

where $M = X/x_0 = U/u_0$ and $N = Y/y_0 = V/v_0$ are the numbers of samples in x and y directions in both spatial and spatial frequency domains, respectively, and $F[k,l]$ is the 2D discrete spectrum of $f[m,n]$. Both $f[m,n]$ and $F[k,l]$ can be considered as elements of two M by N matrices $\mathbf{x}$ and $\mathbf{F}$, respectively.

Physical Meaning of 2DFT

Consider the Fourier transform of continuous, aperiodic signal (the result is easily generalized to other cases):

$$F(u,v) = \int\int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(xu+yv)} dx\, dy$$

$$f(x,y) = \int\int_{-\infty}^{\infty} F(u,v) e^{j2\pi(xu+yv)} du\, dv$$

The inverse transform represents the spatial function $f(x,y)$ as a linear combination of **complex exponentials** $e^{j2\pi(xu+yv)}$ with **complex weights** $F(u,v)$.

- The **Complex weight** can be represented in polar form as

$$F(u,v) = F_r(u,v) + jF_i(u,v) = |F(u,v)|e^{j\angle F(u,v)}$$

in terms of its **amplitude** $|F(u,v)|$ and **phase** $\angle F(u,v)$:

$$\begin{cases} |F(u,v)| \triangleq \sqrt{F_r(u,v)^2 + F_i(u,v)^2} \\ \angle F(u,v) \triangleq tan^{-1}[F_i(u,v)/F_r(u,v)] \end{cases}$$

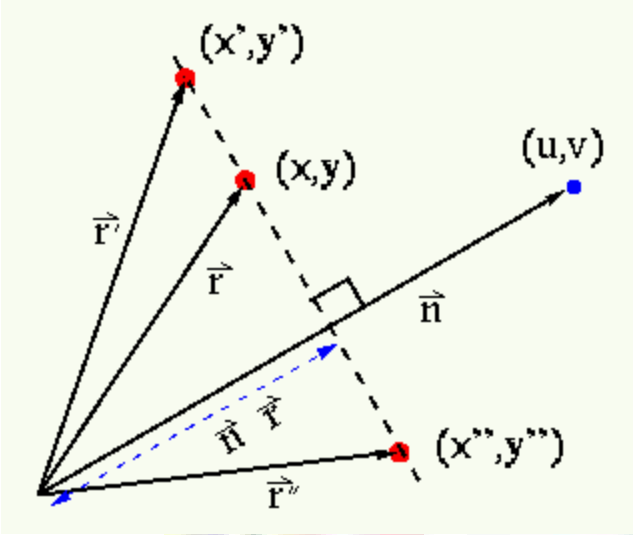
- The **Complex exponential** can be represented as

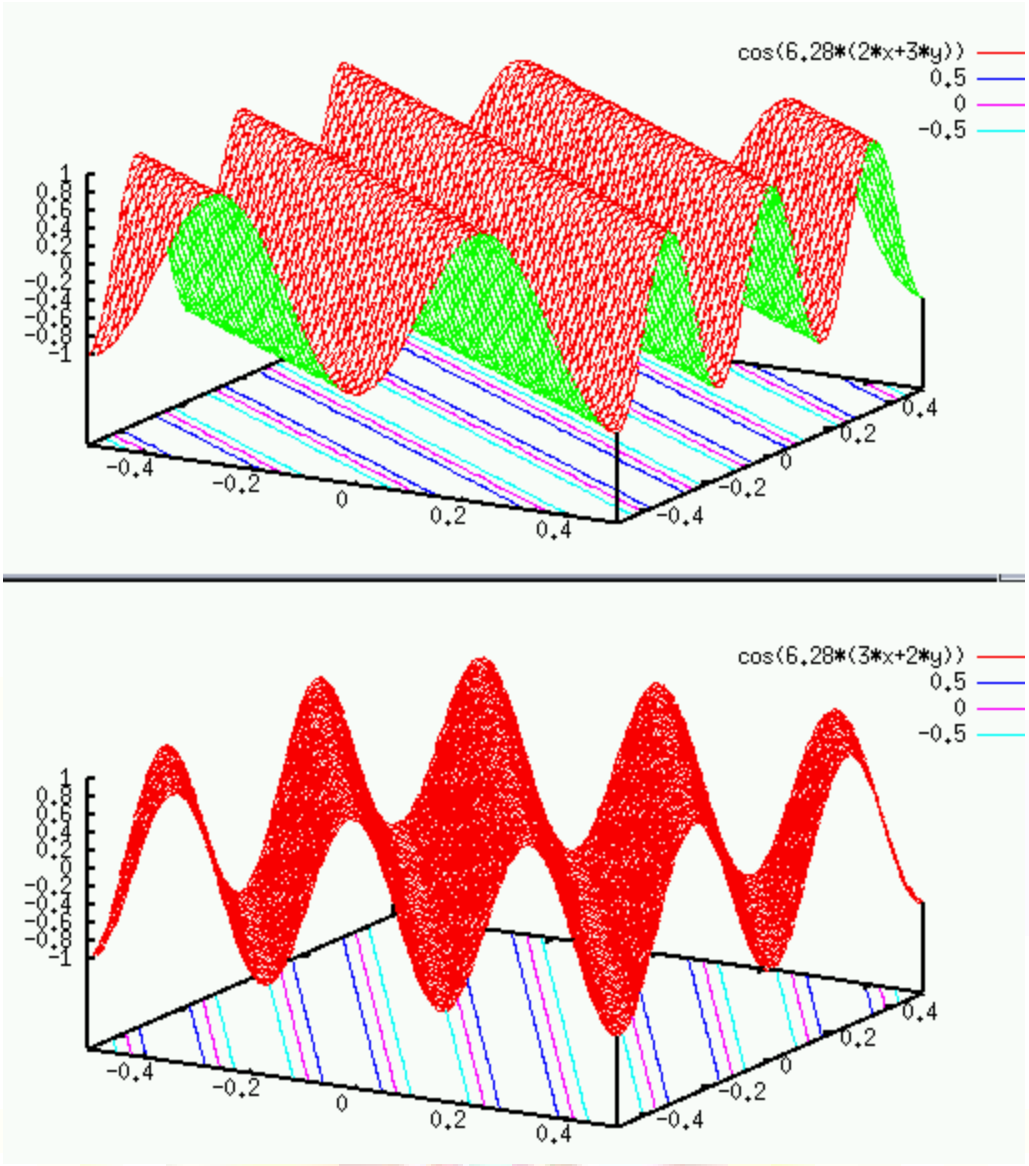
$$e^{j2\pi(xu+yv)} = e^{j2\pi w(xu/w+yv/w)} = e^{j2\pi w(\vec{r}\cdot\vec{n})}$$

where $w \triangleq \sqrt{u^2+v^2}$, and

  - $\vec{n} \triangleq (\frac{u}{w}, \frac{v}{w})$ is the unit vector along direction (u,v),
  - $\vec{r} \triangleq (x,y)$ is a vector along the direction (x,y) in the 2D spatial domain.

The inner product $(\vec{r}\cdot\vec{n})$ represents the projection of a spatial point $\vec{r} = (x,y)$ onto the direction of $\vec{n}$. As all points (x,y) on a straight line perpendicular to the direction of $\vec{n}$ have the same projection, $e^{j2\pi(ux+vy)} = e^{j2\pi w(\vec{r}\cdot\vec{n})}$ represents a planar sinusoid in the x-y plane along the **direction** $\theta = tan^{-1}(v/u)$ (i.e. $\vec{n}$) with **frequency** $w = \sqrt{u^2+v^2}$.


(x',y')
(x,y)
(u,v)
r'
r
n
n
r
(x'',y'')
r''



$$f(x,y) = cos(2\pi(2x+3y))$$

In the function on top, $u = 2$ (2 cycles per unit distance in x) and $v = 3$ and (3 cycles per unit distance in y), while in the

$$f(x,y) = cos(2\pi(3x+2y))$$

function at bottom, $u = 3$ (3 cycles per unit distance in x) and $v = 2$ (2 cycles per unit distance in y). But along their individual

$$tan^{-1}(v/u) \quad tan^{-1}(3/2) \quad tan^{-1}(2/3)$$

directions (and respectively), their spatial

$$w = \sqrt{2^2 + 3^2} = \sqrt{13}$$

frequencies are the same .

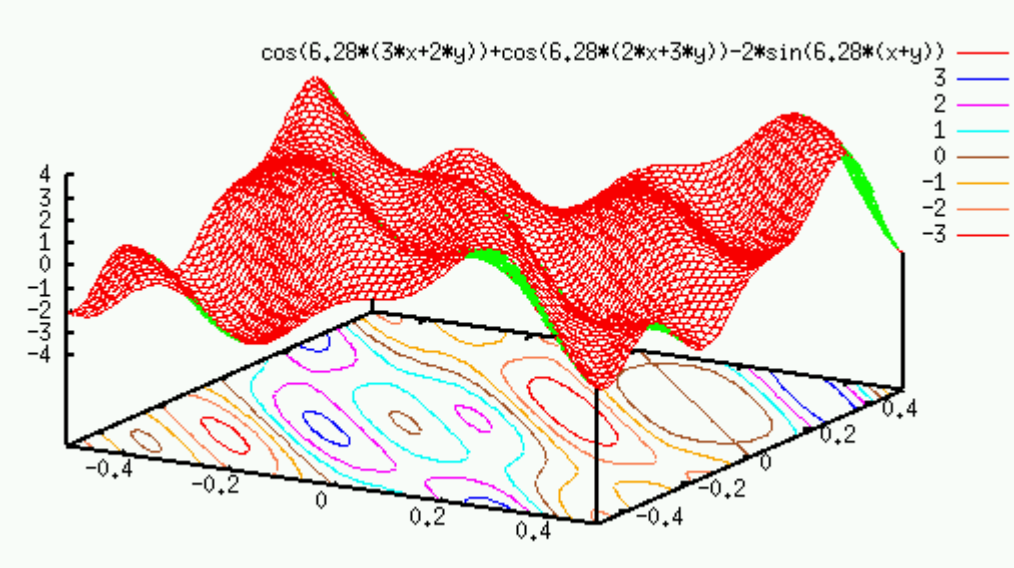
Now the 2DFT of a signal $f(x,y)$ can be written as:

$$f(x,y) = \int\int_{-\infty}^{\infty} |F(u,v)| e^{j\angle F(u,v)} e^{j2\pi(xu+yv)}\, du\, dv = \int\int_{-\infty}^{\infty} |F(u,v)| e^{j(\angle F(u,v)+2\pi w(\vec{r}\cdot\vec{n}))}\, du$$

which represents a signal $f(x,y)$ as a linear combination (integration) of infinite 2D spatial planar sinusoids $F(u,v)$ of

- **frequency:** $w = \sqrt{u^2+v^2}$,
- **direction:** $\vec{n} = tan^{-1}(v/u)$
- **amplitude:** $|F(u,v)| = \sqrt{F_r(u,v)^2 + F_i(u,v)}$,
- **phase:** $\angle F(u,v) = tan^{-1}(F_i(u,v)/F_r(u,v))$

The 2D function shown below contains three frequency components (2D sinusoidal waves) of different frequencies and directions:



Matrix Form of 2D DFT

Consider the 2D DFT:

$$X[k,l] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} [\frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} x[m,n] e^{-j2\pi \frac{mk}{M}}] e^{-j2\pi \frac{nl}{N}} = \sum_{n=0}^{N-1} w_N[n,l][\sum_{m=0}^{M-1} w_M[m,k] x[m,n]]$$

where, as defined before

$$w_M[m,k] \triangleq \frac{1}{\sqrt{M}} e^{-j2\pi mk/M}, \qquad w_N[n,l] \triangleq \frac{1}{\sqrt{N}} e^{-j2\pi nl/N}$$

We further define

$$X'[k,n] \triangleq \sum_{m=0}^{M-1} w_M[m,k] x[m,n], \quad (k = 0, 1, \cdots, M-1)$$

and rewrite the 2D transform as

$$X[k,l] = \sum_{n=0}^{N-1} w_N[n,l] X'[k,n], \quad (l = 0, 1, \cdots, N-1)$$

The above two equations are the two steps for a 2D transform:

1. **Column Transform:**

   First consider the expression for $X'[k,n]$. As the summation is with respect to the row index m of $x[m,n]$, the column index n can be treated as a parameter, and the expression is the 1D Fourier transform of the nth column vector of $\mathbf{x}$, which can be written in column vector (vertical) form for the nth column:

$$\begin{bmatrix} X'[0,n] \\ \cdots \\ X'[M-1,n] \end{bmatrix}_{M\times 1} = \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & w_M[m,k] & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}_{M\times M} \begin{bmatrix} x[0,n] \\ \cdots \\ x[M-1,n] \end{bmatrix}_{M\times 1}$$

   or more concisely

$$\mathbf{X'}_n = \mathbf{W}_M \mathbf{x}_n, \quad (n = 0, \cdots, N-1)$$

i.e., the nth column of $\mathbf{X}'$ is the 1D FT of the nth column of $\mathbf{x}$. Putting all N columns together, we have

$$[\mathbf{X}'_0, \cdots, \mathbf{X}'_{N-1}] = \mathbf{W}_M [\mathbf{x}_0, \cdots, \mathbf{x}_{N-1}]$$

or more concisely

$$\mathbf{X}' = \mathbf{W}_M \mathbf{x}$$

where $\mathbf{W}_M$ is a M by M Fourier transform matrix.

2. **Row Transform:**

Now we reconsider the 2DFT expression above

$$X[k,l] = \sum_{n=0}^{N-1} w_N[n,l] X'[k,n]$$

As the summation is with respective to the column index n of $X'[k,n]$, the row index k can be treated as a parameter, and the expression is the 1D Fourier transform of the kth row vector of $\mathbf{X}'$, which can be written in row vector (horizontal) form for the kth row:

$$[X[k,0], \cdots, X'[k,N-1]] = [X'[k,0], \cdots, X'[k,N-1]] \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & w_N[n,l] & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

or more concisely

$$\mathbf{X}_k^T = \mathbf{X}_k'^T \mathbf{W}_N, \quad (k = 0, \cdots, M-1)$$

i.e., the kth row of $\mathbf{X}$ is the 1D FT of the kth row of $\mathbf{X}'$. Putting all M rows together, we have

$$\begin{bmatrix} \mathbf{X}_0^T \\ . \\ . \\ \mathbf{X}_{M-1}^T \end{bmatrix} = \begin{bmatrix} \mathbf{X}_0'^T \\ . \\ . \\ \mathbf{X}_{M-1}'^T \end{bmatrix} \mathbf{W}_N$$

or more concisely

$$\mathbf{X} = \mathbf{X}'\mathbf{W}_N$$

But as $\mathbf{X}' = \mathbf{W}_M \mathbf{x}$, we finally have

$$\mathbf{X} = \mathbf{W}_M \mathbf{x} \mathbf{W}_N$$

This expression indicates that 2D DFT can be carried out by 1D transforming all the rows of the 2D signal $\mathbf{x}$ and then 1D transforming all the columns of the resulting matrix. The order of the steps is not important. The transform can also be carried out by the column transform followed by the row transform.

Similarly, the inverse 2D DFT can be written as

$$\mathbf{x} = \mathbf{W}_M^* \mathbf{X} \mathbf{W}_N^*$$

It is obvious that the complexity of 2D DFT is $O(N^3)$ (assuming $M = N$), which can be reduced to $O(N^2 log_2 N)$ if FFT is used.

A 2D DFT Example

Consider a real 2D signal:

$$\mathbf{x}_r = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 70.0 & 80.0 & 90.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 90.0 & 100.0 & 110.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 110.0 & 120.0 & 130.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 130.0 & 140.0 & 150.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

The imaginary part $\mathbf{x}_i = 0$. The 2D Fourier spectrum $\mathbf{X}$ of this signal can be found by 2D DFT. The real part of the spectrum is:

$$\mathbf{X}_r = \left[\begin{array}{rrr|r|rrr} 165.0 & -98.9 & 10.0 & -21.1 & 55.0 & -21.1 & 10.0 & -98.9 \\ \hline -63.1 & -11.3 & 27.7 & 13.2 & -21.0 & 1.6 & -32.7 & 85.7 \\ 15.0 & 0.0 & -5.0 & -2.9 & 5.0 & 0.0 & 5.0 & 17.1 \\ -41.9 & 16.8 & 2.7 & 6.3 & -14.0 & 4.3 & -7.7 & 33.4 \\ \hline 15.0 & -8.5 & 0.0 & -1.5 & 5.0 & -1.5 & 0.0 & -8.5 \\ \hline -41.9 & 33.4 & -7.7 & 4.3 & -14.0 & 6.3 & 2.7 & 16.8 \\ 15.0 & -17.1 & 5.0 & 0.0 & 5.0 & -2.9 & -5.0 & 0.0 \\ -63.1 & 85.7 & -32.7 & 1.6 & -21.0 & 13.2 & 27.7 & -11.3 \end{array}\right]$$

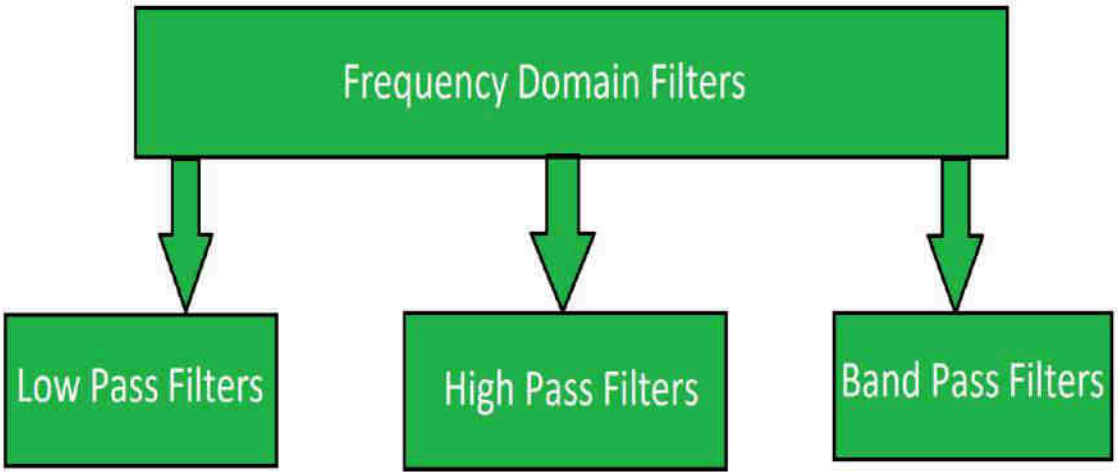
and the imaginary part of the spectrum is:

$$\mathbf{X}_i = \left[\begin{array}{rrrr|rrrr} 0.0 & -88.9 & 55.0 & 11.1 & 0.0 & -11.1 & -55.0 & 88.9 \\ \hline -90.5 & 89.2 & -27.1 & 6.9 & -30.2 & 16.8 & 15.0 & 19.9 \\ 15.0 & -17.1 & 5.0 & 0.0 & 5.0 & -2.9 & -5.0 & 0.0 \\ -15.5 & 31.9 & -15.0 & -0.8 & -5.2 & 4.9 & 12.9 & -13.2 \\ \hline 0.0 & -8.5 & 5.0 & 1.5 & 0.0 & -1.5 & -5.0 & -8.5 \\ \hline 15.0 & 13.2 & -12.9 & -4.9 & 5.2 & 0.8 & 15.0 & -31.9 \\ -15.5 & 0.0 & 5.0 & 2.9 & -5.0 & 0.0 & -5.0 & 17.1 \\ 90.5 & -19.9 & -15.0 & -16.8 & 30.2 & -6.9 & 27.1 & -89.2 \end{array}\right]$$

Pay close attention to the even and odd symmetry of the spectrum.

Frequency Domain Filters are used for smoothing and sharpening of image by removal of high or low frequency components. Sometimes it is possible of removal of very high and very low frequency. Frequency domain filters are different from spatial domain filters as it basically focuses on the frequency of the images. It is basically done for two basic operation i.e., Smoothing and Sharpening.

These are of 3 types:



Classification of Frequency Domain Filters

1. Low pass filter:

Low pass filter removes the high frequency components that means it keeps low frequency components. It is used for smoothing the image. It is used to smoothen the image by

attenuating high frequency components and preserving low frequency components.
Mechanism of low pass filtering in frequency domain is given by:

$G(u, v) = H(u, v) . F(u, v)$

where F(u, v) is the Fourier Transform of original image

and H(u, v) is the Fourier Transform of filtering mask

2. High pass filter:

High pass filter removes the low frequency components that means it keeps high frequency components. It is used for sharpening the image. It is used to sharpen the image by attenuating low frequency components and preserving high frequency components.
Mechanism of high pass filtering in frequency domain is given by:

$H(u, v) = 1 - H'(u, v)$

where H(u, v) is the Fourier Transform of high pass filtering

and H'(u, v) is the Fourier Transform of low pass filtering

3. Band pass filter:

Band pass filter removes the very low frequency and very high frequency components that means it keeps the moderate range band of frequencies. Band pass filtering is used to enhance edges while reducing the noise at the same time.

- Web Technologies
- School Learning
- Student
- Jobs
- GBlog
- Puzzles
- What's New ?

Related Articles

▲

Types of Restoration Filters

- Last Updated : 06 Dec, 2019

Restoration Filters are the type of filters that are used for operation of noisy image and estimating the clean and original image. It may consists of processes that are used for blurring or the reverse processes that are used for inverse of blur. Filter used in restoration is different from the filter used in enhancement process.

Types of Restoration Filters:

There are three types of Restoration Filters: Inverse Filter, Pseudo Inverse Filter, and Wiener Filter. These are explained as following below.

1. Inverse Filter:

Inverse Filtering is the process of receiving the input of a system from its output. It is the simplest approach to restore the original image once the degradation function is known.

It can be define as:

$H'(u, v) = 1 / H(u, v)$

Let,

$F'(u, v)$ -> Fourier transform of the restored image

$G(u, v)$ -> Fourier transform of the degraded image

$H(u, v)$ -> Estimated or derived or known degradation function

then **$F'(u, v) = G(u, v)/H(u, v)$**

where, $G(u, v) = F(u, v).H(u, v) + N(u, v)$

and $F'(u, v) = f(u, v) - N(u, v)/H(u, v)$

Note: Inverse filtering is not regularly used in its original form.

2. Pseudo Inverse Filter:

Pseudo inverse filter is the modified version of the inverse filter and stabilized inverse filter. Pseudo inverse filtering gives more better result than inverse filtering but both inverse and pseudo inverse are sensitive to noise.

Pseudo inverse filtering is defined as:

H'(u, v) = 1/H(u, v), H(u, v)!=0

H'(u, v) = 0, otherwise

3. Wiener Filter:

(Minimum Mean Square Error Filter). Wiener filter executes and optimal trade off between filtering and noise smoothing. IT removes the addition noise and inputs in the blurring simultaneously. Weiner filter is real and even.

It minimizes the overall mean square error by:

e^2 = F{(f-f')^2}

where, f -> original image

f' -> restored image

E{.} -> mean value of arguments

H(u, v) = H'(u, v)/(|H(u, v)|^2 + (Sn(u, v)/Sf(u, v))

where H(u, v) -> Transform of degradation function

Sn(u, v) -> Power spectrum of the noise

Sf(u, v) -> Power spectrum of the undergraded original image

Butterworth Lowpass Filter (BLPF)

In the field of Image Processing, **Butterworth Lowpass Filter (BLPF)** is used for image smoothing in the frequency domain. It removes high-frequency noise from a digital image and preserves low-frequency components. The transfer function of BLPF of order is defined as-

Where,

- is a positive constant. BLPF passes all the frequencies less than value without attenuation and cuts off all the frequencies greater than it.

- This is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as ***cutoff frequency***. But instead of making a sharp cut-off (like,Ideal Lowpass Filter (ILPF)), it introduces a smooth transition from 1 to 0 to reduce ringing artifacts.
- s the Euclidean Distance from any point (u, v) to the origin of the frequency plane, i.e,

Gaussian Low pass filter

Gaussian Smoothing

Common Names: Gaussian smoothing

Brief Description

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump. This kernel has some special properties which are detailed below.

How It Works

The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where is the standard deviation of the distribution. We have also assumed that the distribution has a mean of zero (*i.e.* it is centered on the line x=0). The distribution is illustrated in Figure 1.

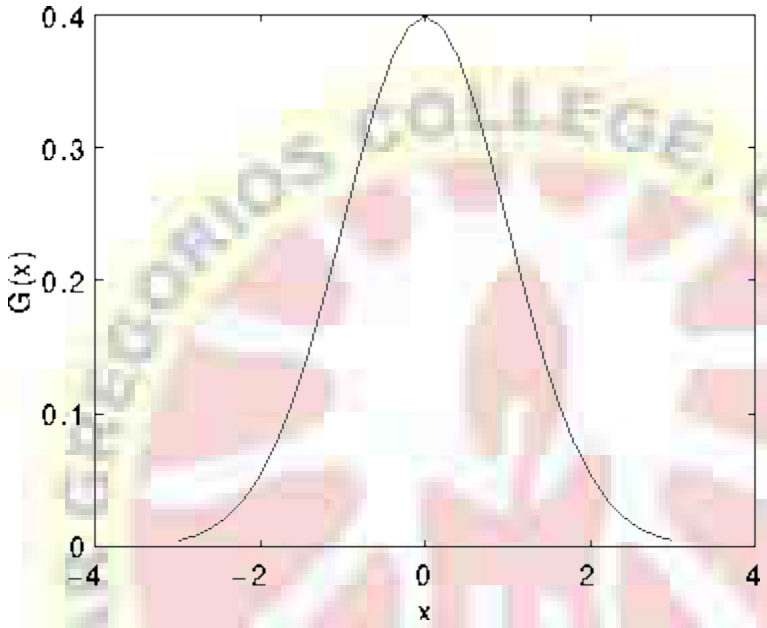


Figure 1 1-D Gaussian distribution with mean 0 and σ=1

In 2-D, an isotropic (*i.e.* circularly symmetric) Gaussian has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

This distribution is shown in Figure 2.

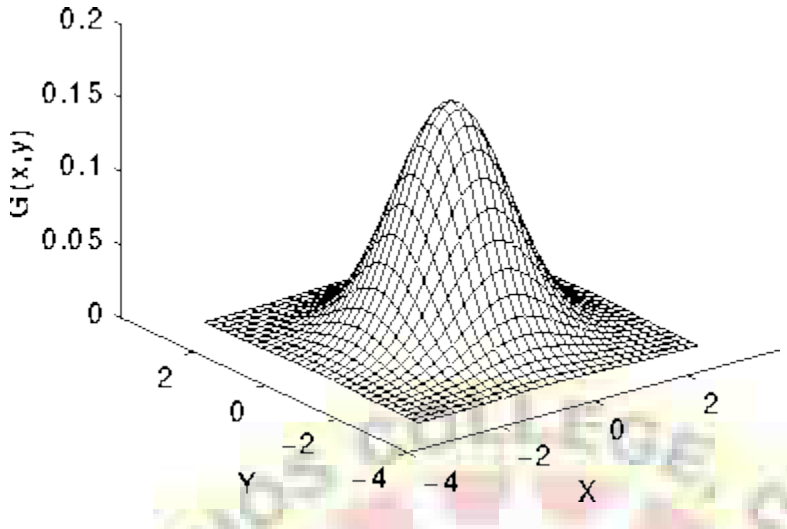


Figure 2 2-D Gaussian distribution with mean (0,0) and σ=1

The idea of Gaussian smoothing is to use this 2-D distribution as a `point-spread' function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is effectively zero more than about three standard deviations from the mean, and so we can truncate the kernel at this point. Figure 3 shows a suitable integer-valued convolution kernel that approximates a Gaussian with a σ of 1.0. It is not obvious how to pick the values of the mask to approximate a Gaussian. One could use the value of the Gaussian at the centre of a pixel in the mask, but this is not accurate because the value of the Gaussian varies non-linearly across the pixel. We integrated the value of the Gaussian over the whole pixel (by summing the Gaussian at 0.001 increments). The integrals are not integers: we rescaled the array so that the corners had the value 1. Finally, the 273 is the sum of all the values in the mask.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 3 Discrete approximation to Gaussian function with σ=1.0

Once a suitable kernel has been calculated, then the Gaussian smoothing can be performed using standard convolution methods. The convolution can in fact be performed fairly quickly since the equation for the 2-D isotropic Gaussian shown above is separable into *x* and *y* components. Thus the 2-D convolution can be performed by first convolving with a 1-D Gaussian in the *x* direction, and then convolving with another 1-D Gaussian in the *y* direction. (The Gaussian is in fact the *only* completely circularly symmetric operator which can be decomposed in such a way.) Figure 4 shows the 1-D *x* component kernel that would be used to produce the full kernel shown in Figure 3 (after scaling by 273, rounding and truncating one row of pixels around the boundary because they mostly have the value 0. This reduces the 7x7 matrix to the 5x5 shown above.). The *y* component is exactly the same but is oriented vertically.

.006	.061	.242	.383	.242	.061	.006

Figure 4 One of the pair of 1-D convolution kernels used to calculate the full kernel shown in Figure 3 more quickly.

A further way to compute a Gaussian smoothing with a large standard deviation is to convolve an image several times with a smaller Gaussian. While this is computationally complex, it can have applicability if the processing is carried out using a hardware pipeline.

The Gaussian filter not only has utility in engineering applications. It is also attracting attention from computational biologists because it has been attributed with some amount of biological plausibility, *e.g.* some cells in the visual pathways of the brain often have an approximately Gaussian response.

Guidelines for Use

The effect of Gaussian smoothing is to blur an image, in a similar fashion to the mean filter. The degree of smoothing is determined by the standard deviation of the Gaussian. (Larger standard deviation Gaussians, of course, require larger convolution kernels in order to be accurately represented.)

The Gaussian outputs a `weighted average' of each pixel's neighborhood, with the average weighted more towards the value of the central pixels. This is in contrast to the mean filter's uniformly weighted average. Because of this, a Gaussian provides gentler smoothing and preserves edges better than a similarly sized mean filter.

One of the principle justifications for using the Gaussian as a smoothing filter is due to its *frequency response*. Most convolution-based smoothing filters act as lowpass frequency filters. This means that their effect is to remove high spatial frequency components from an image. The frequency response of a convolution filter, *i.e.* its effect on different spatial frequencies, can be seen by taking the Fourier transform of the filter. Figure 5 shows the frequency responses of a 1-D mean filter with width 5 and also of a Gaussian filter with σ = 3.

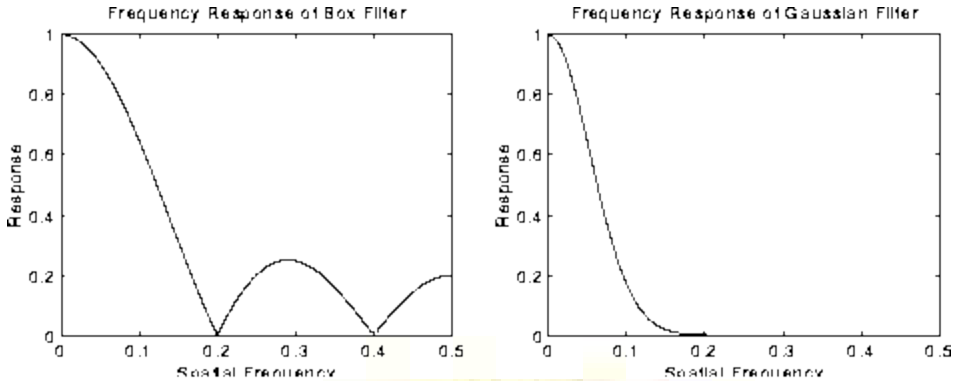


Figure 5 Frequency responses of Box (*i.e.* mean) filter (width 5 pixels) and Gaussian filter (σ = 3 pixels). The spatial frequency axis is marked in cycles per pixel, and hence no value above 0.5 has a real meaning.

Both filters attenuate high frequencies more than low frequencies, but the mean filter exhibits oscillations in its frequency response. The Gaussian on the other hand shows no oscillations. In fact, the shape of the frequency response curve is itself (half a) Gaussian. So by choosing an appropriately sized Gaussian filter we can be fairly confident about what range of spatial frequencies are still present in the image after filtering, which is not the case of the mean filter. This has consequences for some edge detection techniques, as mentioned in the section on zero crossings. (The Gaussian filter also turns out to be very similar to the optimal smoothing filter for edge detection under the criteria used to derive the Canny edge detector.)

Ideal High pass filter

In the field of Image Processing, **Ideal Highpass Filter (IHPF)** is used for image sharpening in the frequency domain. Image Sharpening is a technique to enhance the fine details and highlight the edges in a digital image. It removes low-frequency components from an image and preserves high-frequency components.

This ideal highpass filter is the reverse operation of the ideal lowpass filter. It can be determined using the following relation-

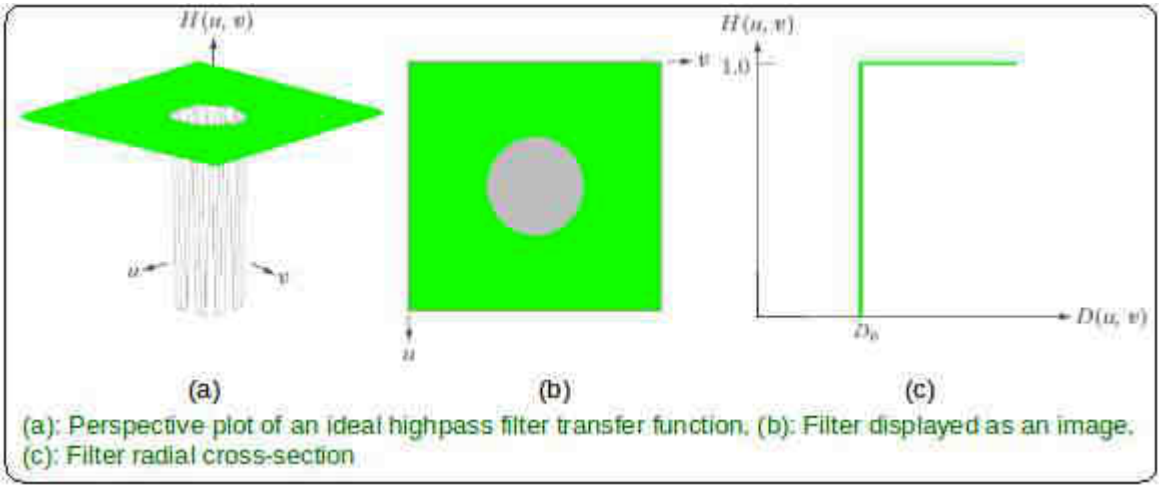
where, is the transfer function of the highpass filter

and is the transfer function of the corresponding lowpass filter.

The transfer function of the IHPF can be specified by the function-

Where,

- is a positive constant. IHPF passes all the frequencies outside of a circle of radius from the origin without attenuation and cuts off all the frequencies within the circle.
- This is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as ***cutoff frequency***.
- is the Euclidean Distance from any point (u, v) to the origin of the frequency plane, i.e,



(a): Perspective plot of an ideal highpass filter transfer function. (b): Filter displayed as an image. (c): Filter radial cross-section

In the field of Image Processing, **Butterworth Highpass Filter (BHPF)** is used for image sharpening in the frequency domain. Image Sharpening is a technique to enhance the fine details and highlight the edges in a digital image. It removes low-frequency components from an image and preserves high-frequency components.

This Butterworth highpass filter is the reverse operation of the Butterworth lowpass filter. It can be determined using the relation-

where, is the transfer function of the highpass filter and is the transfer function of the corresponding lowpass filter.

The transfer function of BHPF of order is defined as-

Where,

- is a positive constant. BHPF passes all the frequencies greater than value without attenuation and cuts off all the frequencies less than it.
- This is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as ***cutoff frequency***. But instead of making a sharp cut-off (like,Ideal Highpass Filter (IHPF)), it introduces a smooth transition from 0 to 1 to reduce ringing artifacts.
- is the Euclidean Distance from any point (u, v) to the origin of the frequency plane,.

UNIT IV

Image Restoration :

Image restoration is performed by reversing the process that blurred the**image** and such is performed by imaging a **point** source and use the **point** source **image**, which is called the **Point** Spread Function (PSF) to **restore**the **image** information lost to the blurring process

Mean Filters

In this section we discuss briefly the noise-reduction spatial filters introduced in Section 3.6 and develop several other filters whose performance is in many cases superior to the filters discussed in that section.

Arithmetic mean filter

This is the simplest of the mean filters. Let Sxv represent the set of coordinates in a rectangular subimage window of size m X n, centered at point (x, y).The arithmetic mean filtering process computes the average value of the corrupted image g(x, y) in the area defined by Sxy.The value of the restored image at any point (x, y) is simply the arithmetic mean computed using the pixels in the region defined by S. In other words.

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t). \qquad (5.3\text{-}3)$$

This operation can be implemented using a convolution mask in which ail coefficients have value 1/mn. As discussed in Section 3.6.1, a mean filter simply smoothes local variations in an image. Noise is reduced as a result of blurring.

Geometric mean filter

An image restored using a geometric mean filter is given by the expression

$$\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t) \right]^{\frac{1}{mn}}. \qquad (5.3\text{-}4)$$

Here, each restored pixel is given by the product of the pixels in the subimage window, raised to the power 1/mn. As shown in Example 52, a geometric mean filter achieves smoothing comparable to the arithmetic mean filter, but it tends to lose less image detail in the process.

Harmonic mean filter

The harmonic mean filtering operation is given by the expression

$$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}. \qquad (5.3\text{-}5)$$

The harmonic mean filter works well for salt noise, but fails for pepper noise. It does well also with other types of noise tike Gaussian noise.

Contraharmonic mean filter

The contraharmonic mean filtering operation yields a restored image based on the expression

$$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}} \qquad (5.3\text{-}6)$$

where Q is called the order of the filter. This filter is well suited for reducing or virtually eliminating the effects of salt-and-pepper noise. For positive values of Q, the filter eliminates pepper noise. For negative values of Q it eliminates salt noise. It cannot do both simultaneously. Note that the contraharmonic filter reduces to the arithmetic mean filter if Q = 0, and to the harmonic mean filter if Q = - 1

5.3.2 Order-Statistics Filters

Order-statistics filters were introduced in Section 3.6.2. We now expand the dis¬cussion in that section and introduce some additional order-statistics filters. As noted in Section 3.6.2, order-statistics filters are spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encom¬passed by the filter. The response of the filter at any point is determined by the ranking result

Median filter

The best-known order-statistics filter is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel:

$$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}. \qquad (5.3\text{-}7)$$

The original value of the pixel is included in the computation of the median. 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 filters are particularly effective in the presence of both bipolar and unipolar impulse noise. In fact, as Ex¬ample 5.3 shows, the median filter yields excellent results for images corrupted by this type of noise. Computation of the median and implementation of this filter are discussed in detail in Section 3.6.2.

Max and min filters

Although the median filter is by far the order-statistics filter most used in image processing.it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but the reader will recall from basic statis¬tics that ranking lends itself to many other possibilities. For example, using the 100th percentile results in the so-called max filter given by:

$$\hat{f}(x, y) = \max_{(s,t)\in S_{xy}} \{g(s, t)\}. \qquad (5.3\text{-}8)$$

This filter is useful for finding the brightest points in an image. Also, because pep¬per noise has very low values, it is reduced by this filter as a result of the max selection process in the subimage area S. The 0th percentile filter is the Min filter.

$$\hat{f}(x, y) = \min_{(s,t)\in S_{xy}} \{g(s, t)\}. \qquad (5.3\text{-}9)$$

Adaptive Filtering

Adaptive filters are designed to address this question, and are the third and final class of spatial image filters we shall explore. The premise behind adaptive image filtering is that by

varying the filtering method as the kernel slides across the image (in the same manner as the convolution operation), they are able to tailor themselves to the local properties and structures of an image. In essence, they can be thought of as self-adjusting digital filters. While certain types of adaptive filters may perform better than median filters at removing impulse noise (these are mostly variations on the basic median filtering scheme), they are most often used for denoising non-stationary images, which tend to exhibit abrupt intensity changes. Because the filtering operation is no longer purely uniform and instead modulated based on the local characteristics of the image, these filters can be employed effectively when there is little a priori knowledge of the signal being processed.

Adaptive filters find widespread use in countering the effects of so-called "speckle" noise, which afflicts coherent imaging systems like SAR and ultrasound. With these imaging techniques, scattered waves interfere with one another to contaminate an acquired image with multiplicative speckle noise. Various statistical models of speckle noise exist, with one of the more common being

$$g(x,y) = f(x,y) + f(x,y)n(x,y)$$

where g is the corrupted image and n(x,y) is drawn from a zero-mean Gaussian distribution with a given standard deviation. It is clear from the above model that speckle noise is dependent on the magnitude of the signal f, and in fact this type of noise is a serious impediment on the interpretability of image data because it degrades both spatial and contrast resolution. This situation is shown with a real-world example in Figure 4-24, where an ultrasound image exhibiting a fairly large amount of speckle noise is enhanced, illustrating the utility of adaptive filtering.

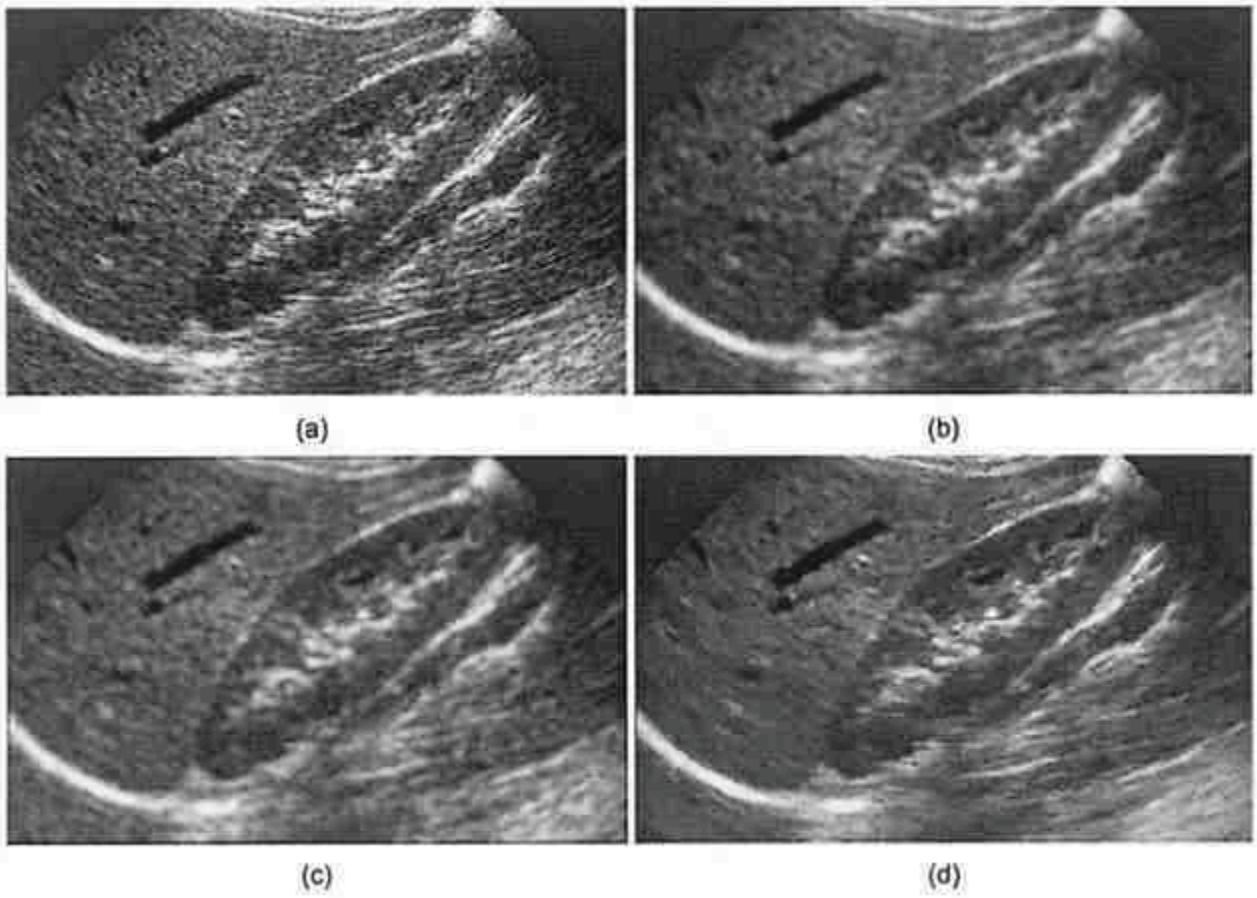


Figure 4-24. Ultrasound speckle noise removal, (a) Unprocessed ultrasound image of kidney anatomy, (b) Smoothed image (5×5 averaging filter), which while reducing speckle seriously degrades spatial resolution, (c) Result of adaptive filtering, using MATLAB wiener2 Image-Processing Toolkit function, (d) Commercial quality speckle tioise reduction, where image is despeckled using an adaptive geometric filter and then edge sharpened.

Adaptive filters are effective in reducing the deleterious effects of speckle noise because they are capable of adjusting themselves based on the signal content of the image. Thus it follows that they must use some measure of the local characteristics of the image in order to perform their job. Many adaptive filters are predicated on the use of local pixel statistics, primarily the mean and variance of the pixels within the current neighborhood. The local mean is simply the average pixel intensity of the neighborhood. The local variance is calculated in two stages from the pixels contained within the current neighborhood. First, the mean of the sum of the squares is computed, and then the square of the local mean is

subtracted from this number yielding a statistical quantity known as the variance, very often referred to as a2. In mathematical terms, these image statistics can be expressed as

$$local\ mean(i,j) = \mu = \frac{1}{NH^2} \sum_{i'=-\frac{NH}{2}}^{\frac{NH}{2}} \sum_{j'=-\frac{NH}{2}}^{\frac{NH}{2}} f(i+i', j+j')$$

$$local\ variance(i,j) = \sigma^2 = \left[\frac{1}{NH^2} \sum_{i'=-\frac{NH}{2}}^{\frac{NH}{2}} \sum_{j'=-\frac{NH}{2}}^{\frac{NH}{2}} f(i+i', j+j')^2 \right] - \mu^2$$

where /is the image and each neighborhood is of size NHxNH pixels. The standard deviation of the neighborhood is the square root of the variance, or a.

The Minimal Mean Square Error Filter

The adaptive filter we spend the majority of time on in this section is the Minimal Mean Square Error (MMSE) filter. This filter can be used to remove both additive white noise and speckle noise. Consider an observed image f(i,j) and a neighborhood L of size NHxNH. Let cr2 be the noise variance, be the local mean, and <j\ be the local variance. The cr2 parameter is the variance of a representative background area of the image containing nothing but noise (a technique for estimating this parameter is given later). The linear MMSE filter output is then given by26

$$g(i,j) = \left(1 - \frac{\sigma_n^2}{\sigma_L^2}\right) f(i,j) + \frac{\sigma_n^2}{\sigma_L^2} \mu_L$$

This equation describes a linear interpolation between the observed image /and a smoothed version off. Care should be taken to handle the case where

$$\sigma_L^2 << \sigma_n^2$$

in which case a negative output pixel may result (the MMSE filter implemented later in this section clamps the ratio σ_n^2/σ_L^2 to 1). The MMSE filter works as follows:

1. If the local variance is much greater than the noise variance, or if σ_n^2 is small or zero, it produces a value close to the input f(i,j). If $\sigma_L^2 >> \sigma_n^2$, this part of the image most likely contains an edge and this filter makes the assumption that it is best to leave that portion of the image alone.

2. If the noise variance dominates over the local variance, return the local mean.

3. If the two variance measures are more or less equal, the filter returns a mixture between the input and local mean.

This filter is edge-preserving because of (1), and consequently should tend to retain overall image sharpness, although noise will not be filtered from those portions of the image containing edges. Algorithm 4-2 describes the MMSE filter in pseudo-code form.

High-Pass Filters

The concept of high-pass filtering is to remove lower frequency content while keeping higher frequencies. With image processing, this, by it self, yields undesirable results. Particularly, removing the overall brightness represented at position (0, 0) of the image in the frequency domain is not desired. Thus, to preserve the low-frequency content while emphasising the high-frequency content we alter the transfer function with a high-frequency emphasis equation.

$$H_{hfe}(u, v) = a + b\,H(u, v)$$

A value of one for both a and b is common.

ideal high-pass filter

$$H_{IH}(u,\, v) = \begin{cases} 0 & \text{if } D(u,\, v) \leq D_0 \\ 1 & \text{if } D(u,\, v) > D_0 \end{cases}$$

Gaussian high-pass filter

$$H_{GH}(u,\, v) = 1 - e^{-(D(u,v)^2)/\, 2\sigma^2}$$

Butterworth high-pass filter

$$H_{BH}(u,\, v) = \frac{1}{1 + [D_0/\, D(u,v)]^{2n}}$$

Band-reject Filters

Band-reject and Band-Pass filters are used less in image processing than low-pass and high-pass filters.

Band-reject filters (also called band-stop filters) suppress frequency content within a range between a lower and higher cutoff frequency. The parameter D_0 here is the center frequency of the reject band.

ideal band-reject filter

$$H_{IBR}(u,\, v) = \begin{cases} 0 & \text{if } D_0 - W/2 \leq D(u,v) \leq D_0 + W/2 \\ 1 & \text{otherwise} \end{cases}$$

Gaussian band-reject filter

$$H_{GBR}(u,\, v) = 1 - e^{-\left(\frac{D(u,v)^2 - {D_0}^2}{D(u,v)\, W}\right)^2}$$

Butterworth band-reject filter

$$H_{BBR}(u,\, v) = \frac{1}{1 + \left[\frac{D(u,v)\, W}{D(u,v)^2 - {D_0}^2}\right]^{2n}}$$

Band-Pass Filters

A band-pass filter is the opposite of a band-reject filter. It only passes frequency content within a range. A band-pass filter is obtained from a band-reject filter.

Notch Filter

Notch filters, also commonly referred to as band-stop or band-rejection filters, are designed to transmit most wavelengths with little intensity loss while attenuating light within a specific wavelength range (the stop band) to a very low level. They are essentially the inverse of bandpass filters, which offer high in-band transmission and high out-of-band rejection so as to only transmit light within a small wavelength range. See the *Transmission Graphs* and *OD Graphs* tabs for the performance over the passbands and blocking region.

Notch filters are useful in applications where one needs to block light from a laser. For instance, to obtain good signal-to-noise ratios in Raman spectroscopy experiments, it is critical that light from the pump laser be blocked. This is achieved by placing a notch filter in the detection channel of the setup. In addition to spectroscopy, notch filters are commonly used in laser-based fluorescence instrumentation and biomedical laser systems.

As with all dielectric stack filters, the transmission is dependent on the angle of incidence (AOI). The central wavelength of the blocking region will shift to shorter wavelengths as the AOI is increased. The *AOI Graphs* tab shows the transmission of s- and p-polarized light as a function of wavelength changes for different AOI.

Thorlabs' notch filters feature a dielectric coating on a polished glass substrate, which has excellent environmental durability. The dielectric stack provides high rejection through destructive interference and reflection in the stop band: the optical density is greater than 6.0 (corresponding to a transmission of less than 0.0001%) within the stop band. See the table below for available stop band center wavelengths. Regardless of the filter chosen, the transmitted beam's wavefront error for light at normal incidence will be less than λ/2 at 633 nm. These filters also have an AR coating on the back surface to ensure >90% average transmission within the passbands.

Each filter is housed in a black anodized aluminum ring that is labeled with an arrow indicating the design propagation direction. The ring makes handling easier and enhances the blocking OD by limiting scattering. These filters can be mounted in our extensive line of filter mounts and wheels. As the mounts are not threaded, Ø1" retaining rings will be required to mount the filters in one of our internally-threaded SM1 lens tubes. We do not recommend removing the filter from its mount as the risk of damaging the filter is very high. However, select filters are available unmounted as well as in custom sizes; contact Tech Support for more details.

Image Segmentation

What is the Process of Image Segmentation?

A digital image is made up of various components that need to be "analysed", let's use that word for simplicity sake and the "analysis" performed on such components can reveal a lot of hidden information from them. This information can help us address a plethora of business problems – which is one of the many end goals that are linked with image processing.

Image Segmentation is the process by which a digital image is partitioned into various subgroups (of pixels) called Image Objects, which can reduce the complexity of the image, and thus analysing the image becomes simpler.

We use various image segmentation algorithms to split and group a certain set of pixels together from the image. By doing so, we are actually assigning labels to pixels and the pixels with the same label fall under a category where they have some or the other thing common in them.

Using these labels, we can specify boundaries, draw lines, and separate the most required objects in an image from the rest of the not-so-important ones. In the below example, from a main image on the left, we try to get the major components, e.g. chair, table etc. and hence all the chairs are colored uniformly. In the next tab, we have detected instances, which talk about individual objects, and hence the all the chairs have different colors.

This is how different methods of segmentation of images work in varying degrees of complexity and yield different levels of outputs.

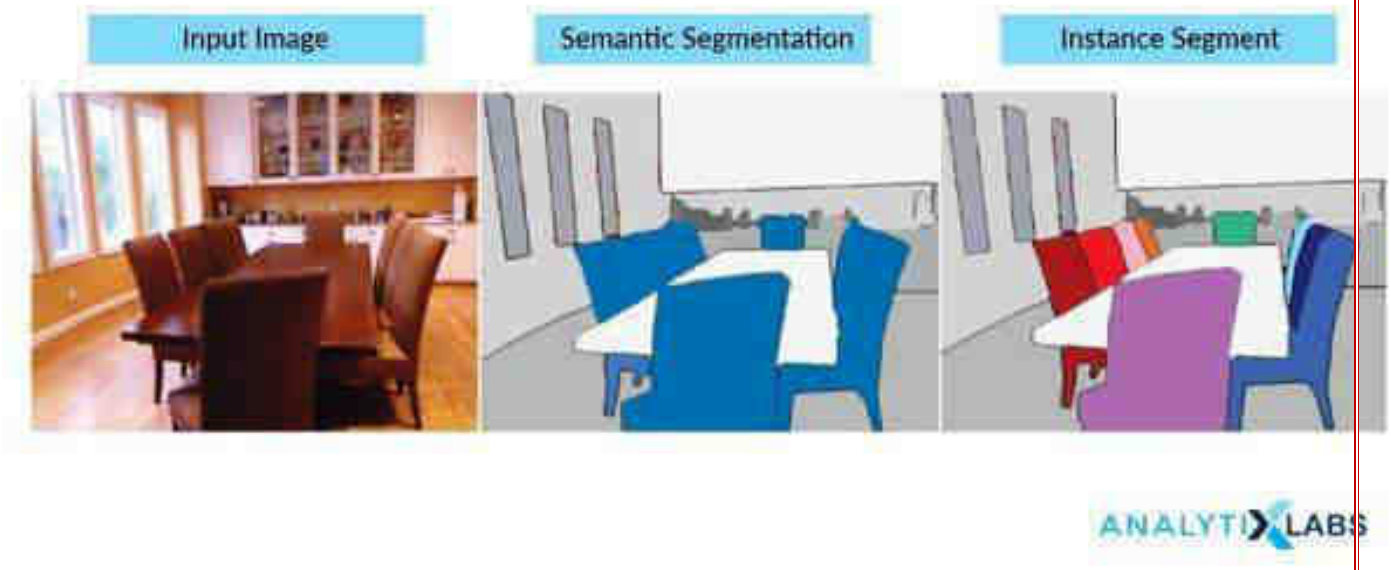


Image Source: stackexchange.com

From a machine learning point of view, later, these identified labels can be further used for both supervised and unsupervised training and hence simplifying and solving a wide variety of business problems. This is a simpler overview of segmentation in Image Processing. Let's try to understand the use cases, methodologies, and algorithms used in this article.

Need for Image Segmentation & Value Proposition

The concept of partitioning, dividing, fetching, and then labeling and later using that information to train various ML models have indeed addressed numerous business problems. In this section, let's try to understand what problems are solved by Image Segmentation.

A facial recognition system implements image segmentation, identifying an employee and enabling them to mark their attendance automatically. Segmentation in Image Processing is being used in the medical industry for efficient and faster diagnosis, detecting diseases,

tumors, and cell and tissue patterns from various medical imagery generated from radiography, MRI, endoscopy, thermography, ultrasonography, etc.

Satellite images are processed to identify various patterns, objects, geographical contours, soil information etc., which can be later used for agriculture, mining, geo-sensing, etc. Image segmentation has a massive application area in robotics, like RPA, self-driving cars, etc. Security images can be processed to detect harmful objects, threats, people and incidents. Image segmentation implementations in python, Matlab and other languages are extensively employed for the process.

A very interesting case I stumbled upon was a show about a certain food processing factory on the Television, where tomatoes on a fast-moving conveyer belt were being inspected by a computer. It was taking high-speed images from a suitably placed camera and it was passing instructions to a suction robot which was pick up rotten ones, unripe ones, basically, damaged tomatoes and allowing the good ones to pass on.

This is a basic, but a pivotal and significant application of Image Classification, where the algorithm was able to capture only the required components from an image, and those pixels were later being classified as the good, the bad, and the ugly by the system. A rather simple looking system was making a colossal impact on that business – eradicating human effort, human error and increasing efficiency.

Image Segmentation is very widely implemented in Python, along with other classical languages like Matlab, C/C++ etc. More likey so, Image segmentation in python has been the most sought after skill in the data science stack.

Types of Image Segmentation

1. The Approach

Whenever one tries to take a bird's eye view of the Image Segmentation tasks, one gets to observe a crucial process that happens here – object identification. Any simple to complex application areas, everything is based out of object detection.

And as we discussed earlier, detection is made possible because the image segmentation algorithms try to – if we put it in lay man's terms – collect similar pixels together and separate out dissimilar pixels. This is done by following two approaches based on the image properties:

1.1. Similarity Detection (Region Approach)

This fundamental approach relies on detecting similar pixels in an image – based on a threshold, region growing, region spreading, and region merging. Machine learning algorithms like clustering relies on this approach of similarity detection on an unknown set of features, so does classification, which detects similarity based on a pre-defined (known) set of features.

1.2. Discontinuity Detection (Boundary Approach)

This is a stark opposite of similarity detection approach where the algorithm rather searches for discontinuity. Image Segmentation Algorithms like Edge Detection, Point Detection, Line Detection follows this approach – where edges get detected based on various metrics of discontinuity like intensity etc.

Image Source: scikit-image

2. The Types of Techniques

Based on the two approaches, there are various forms of techniques that are applied in the design of the Image Segmentation Algorithms. These techniques are employed based on the type of image that needs to be processed and analysed and they can be classified into three broader categories as below:

2.1 Structural Segmentation Techniques

These sets of algorithms require us to firstly, know the structural information about the image under the scanner. This can include the pixels, pixel density, distributions, histograms, color distribution etc. Second, we need to have the structural information about the region that we are about to fetch from the image – this section deals with identifying our target area, which is highly specific to the business problem that we are trying to solve. Similarity based approach will be followed in these sets of algorithms.

2.2 Stochastic Segmentation Techniques

In these group of algorithms, the primary information that is required for them is to know the discrete pixel values of the full image, rather than pointing out the structure of the required portion of the image. This proves to be advantageous in the case of a larger group of images, where a high degree of uncertainty exists in terms of the required object within an object. ANN and Machine Learning based algorithms that use k-means etc. make use of this approach.

2.3 Hybrid Techniques

As the name suggests, these algorithms for image segmentation make use of a combination of structural method and stochastic methods i.e., use both the structural information of a region as well as the discrete pixel information of the image.

Image segmentation Techniques

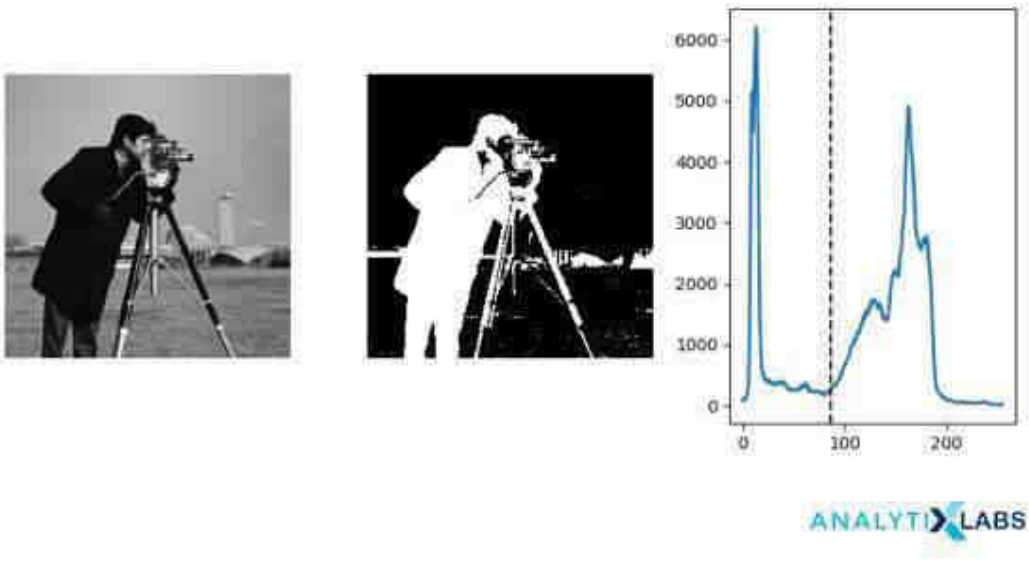
Based on the image segmentation approaches and the type of processing that is needed to be incorporated to attain a goal, we have the following techniques for image segmentation.

1. Threshold Method

2. Edge Based Segmentation
3. Region Based Segmentation
4. Clustering Based Segmentation
5. Watershed Based Method
6. Artificial Neural Network Based Segmentation

1. Threshold Method

This is perhaps the most basic and yet powerful technique to identify the required objects in an image. Based on the intensity, the pixels in an image get divided by comparing the pixel's intensity with a threshold value. The threshold method proves to be advantageous when the objects in the image in question are assumed to be having more intensity than the background (and unwanted components) of the image.



At its simpler level, the threshold value T is considered to be a constant. But that approach may be futile considering the amount of noise (unwanted information) that the image contains. So, we can either keep it constant or change it dynamically based on the image properties and thus obtain better results. Based on that, thresholding is of the following types:

1.1 Simple Thresholding

This technique replaces the pixels in an image with either black or white. If the intensity of a pixel ($I_{i,j}$) at position (i,j) is less than the threshold (T), then we replace that with black and if it is more, then we replace that pixel with white. This is a binary approach to thresholding.

1.2 Otsu's Binarization

In global thresholding, we had used an arbitrary value for threshold value and it remains a constant. The major question here is, how can we define and determine the correctness of the selected threshold? A simpler but rather inept method is to trial and see the error.

But, on the contrary, let us take an image whose histogram has two peaks (bimodal image), one for the background and one for the foreground. According to Otsu binarization, for that image, we can approximately take a value in the middle of those peaks as the threshold value. So in simply put, it automatically calculates a threshold value from image histogram for a bimodal image.

The disadvantage here, however, is for images that are not bimodal, the image histogram has multiple peaks, or one of the classes (peaks) present has high variance.

However, Otsu's Binarization is widely used in document scans, removing unwanted colors from a document, pattern recognition etc.

1.3 Adaptive Thresholding

A global value as threshold value may not be good in all the conditions where an image has different background and foreground lighting conditions in different actionable areas. We need an adaptive approach that can change the threshold for various components of the image. In this, the algorithm divides the image into various smaller portions and calculates the threshold for those portions of the image.

Hence, we obtain different thresholds for different regions of the same image. This in turn gives us better results for images with varying illumination. The algorithm can automatically calculate the threshold value. The threshold value can be the mean of neighborhood area or it can be the weighted sum of neighborhood values where weights are a Gaussian window (a window function to define regions).

2. Edge Based Segmentation

Edge detection is the process of locating edges in an image which is a very important step towards understanding image features. It is believed that edges consist of meaningful features and contains significant information. It significantly reduces the size of the image that will be processed and filters out information that may be regarded as less relevant, preserving and focusing solely on the important structural properties of an image for a business problem.

Edge-based segmentation algorithms work to detect edges in an image, based on various discontinuities in grey level, colour, texture, brightness, saturation, contrast etc. To further enhance the results, supplementary processing steps must follow to concatenate all the edges into edge chains that correspond better with borders in the image.

Image Source: researchgate.net

Edge detection algorithms fall primarily into two categories – Gradient based methods and Gray Histograms. Basic edge detection operators like sobel operator, canny, Robert's variable etc are used in these algorithms. These operators aid in detecting the edge discontinuities and hence mark the edge boundaries. The end goal is to reach at least a partial segmentation using

this process, where we group all the local edges into a new binary image where only edge chains that match the required existing objects or image parts are present.

3. Region Based Segmentation

The region based segmentation methods involve the algorithm creating segments by dividing the image into various components having similar characteristics. These components, simply put, are nothing but a set of pixels. Region-based image segmentation techniques initially search for some seed points – either smaller parts or considerably bigger chunks in the input image.

Next, certain approaches are employed, either to add more pixels to the seed points or further diminish or shrink the seed point to smaller segments and merge with other smaller seed points. Hence, there are two basic techniques based on this method.

3.1 Region Growing

It's a bottom to up method where we begin with a smaller set of pixel and start accumulating or iteratively merging it based on certain pre-determined similarity constraints. Region growth algorithm starts with choosing an arbitrary seed pixel in the image and compare it with its neighboring pixels.

If there is a match or similarity in neighboring pixels, then they are added to the initial seed pixel, thus increasing the size of the region. When we reach the saturation and hereby, the growth of that region cannot proceed further, the algorithm now chooses another seed pixel, which necessarily does not belong to any region(s) that currently exists and start the process again.

Region growing methods often achieve effective Segmentation that corresponds well to the observed edges. But sometimes, when the algorithm lets a region grow completely before trying other seeds, that usually biases the segmentation in favour of the regions which are segmented first. To counter this effect, most of the algorithms begin with the user inputs of similarities first, no single region is allowed to dominate and grow completely and multiple regions are allowed to grow simultaneously.

Region growth, also a pixel based algorithm like thresholding but the major difference is thresholding extracts a large region based out of similar pixels, from anywhere in the image whereas region-growth extracts only the adjacent pixels. Region growing techniques are preferable for noisy images, where it is highly difficult to detect the edges.

3.2 Region Splitting and Merging

The splitting and merging based segmentation methods use two basic techniques done together in conjunction – region splitting and region merging – for segmenting an image. Splitting involves iteratively dividing an image into regions having similar characteristics and merging employs combining the adjacent regions that are somewhat similar to each other.

A region split, unlike the region growth, considers the entire input image as the area of business interest. Then, it would try matching a known set of parameters or pre-defined similarity constraints and picks up all the pixel areas matching the criteria. This is a divide and conquers method as opposed to the region growth algorithm.

Now, the above process is just one half of the process, after performing the split process, we will have many similarly marked regions scattered all across the image pixels, meaning, the final segmentation will contain scattered clusters of neighbouring regions that have identical or similar properties. To complete the process, we need to perform merging, which after each split which compares adjacent regions, and if required, based on similarity degrees, it merges them. Such algorithms are called split-merge algorithms.

4. Clustering Based Segmentation Methods

Clustering algorithms are unsupervised algorithms, unlike Classification algorithms, where the user has no pre-defined set of features, classes, or groups. Clustering algorithms help in fetching the underlying, hidden information from the data like, structures, clusters, and groupings that are usually unknown from a heuristic point of view.

The clustering based techniques segment the image into clusters or disjoint groups of pixels with similar characteristics. By the virtue of basic Data Clustering properties, the data elements get split into clusters such that elements in same cluster are more similar to each other as compared to other clusters. Some of the more efficient clustering algorithms such as

k-means, improved k means, fuzzy c-mean (FCM) and improved fuzzy c mean algorithm (IFCM) are being widely used in the clustering based approaches proposed.

K means clustering is a chosen and popular method because of its simplicity and computational efficiency. The Improved K-means algorithm can minimize the number of iterations usually involved in a k-means algorithm. FCM algorithm allows data points, (pixels in our case) to belong to multiple classes with varying degrees of membership. The slower processing time of an FCM is overcome by improved FCM.

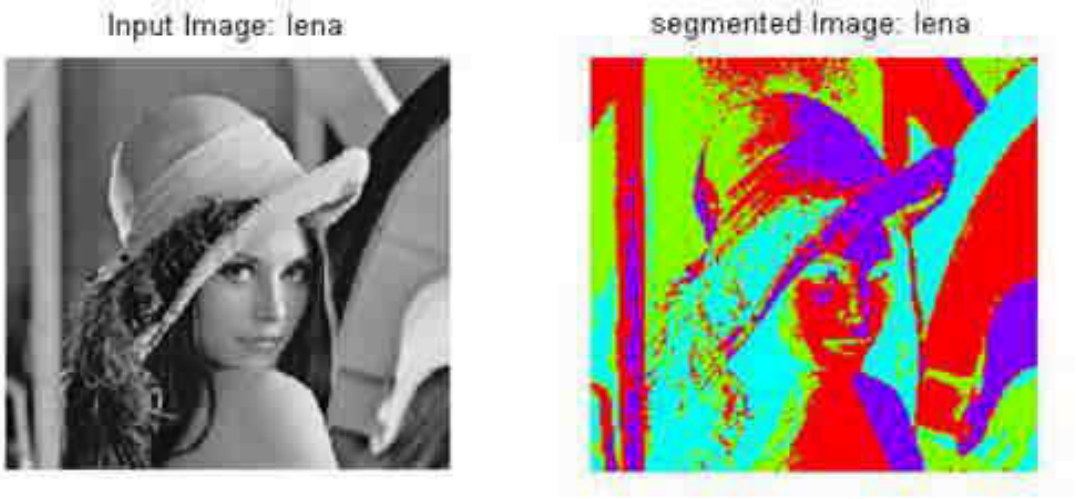


Image Source: researchgate.net

A massive value add of clustering based ML algorithms is that we can measure the quality of the segments that get generated by using several statistical parameters such as: Silhouette Coefficient, rand index (RI) etc.

4.1 k-means clustering

K-means is one of the simplest unsupervised learning algorithms which can address the clustering problems, in general. The process follows a simple and easy way to classify a given image through a certain number of clusters which are fixed apriori. The algorithm actually starts at this point where the image space is divided into k pixels, representing k group centroids. Now, each of the objects is then assigned to the group based on its distance from the cluster. When all the pixels are assigned to all the clusters, the centroids now move and are reassigned. These steps repeat until the centroids can no longer shift.

At the convergence of this algorithm, we have areas within the image, segmented into “K” groups where the constituent pixels show some levels of similarity.

4.2 Fuzzy C Means

k-means, as discussed in the previous section, allows for dividing and grouping together the pixels in an image that have certain degrees of similarity. One of the striking features in k-means is that the groups and their members are completely mutually exclusive. A Fuzzy C Means clustering technique allows the data points, in our case, the pixels to be clustered in more than one cluster. In other words, a group of pixels can belong to more than one cluster or group but they can have varying levels of associativity per group. The FCM algorithm has an optimization function associated with it and the convergence of the algorithm depends on the minimization of this function.

At the convergence of this algorithm, we have areas within the image, segmented into “C” groups where the constituent pixels inside a group show some levels of similarity, and also they will have a certain degree of association with other groups as well.

5. Watershed Based Methods

Watershed is a ridge approach, also a region-based method, which follows the concept of topological interpretation. We consider the analogy of geographic landscape with ridges and valleys for various components of an image. The slope and elevation of the said topography are distinctly quantified by the gray values of the respective pixels – called the gradient magnitude. Based on this 3D representation which is usually followed for Earth landscapes, the watershed transform decomposes an image into regions that are called “catchment basins”. For each local minimum, a catchment basin comprises all pixels whose path of steepest descent of gray values terminates at this minimum.

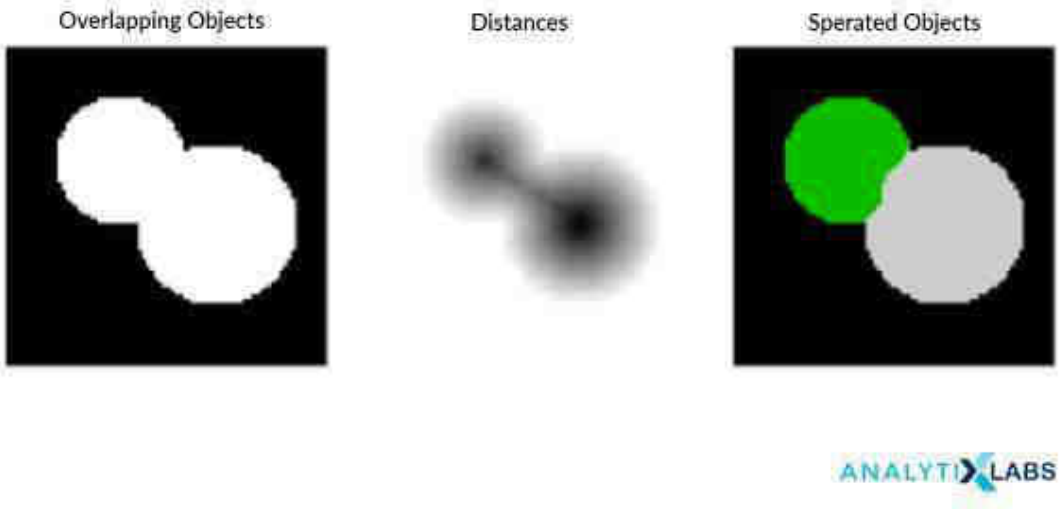


Image Source: scikit-image

In a simple way of understanding, the algorithm considers the pixels as a "local topography" (elevation), often initializing itself from user-defined markers. Then, the algorithm defines something called "basins" which are the minima points and hence, basins are flooded from the markers until basins meet on watershed lines. The watersheds that are so formed here, they separate basins from each other. Hence the picture gets decomposed because we have pixels assigned to each such region or watershed.

6. Artificial Neural Network Based Segmentation Method

The approach of using Image Segmentation using neural networks is often referred to as Image Recognition. It uses AI to automatically process and identify the components of an image like objects, faces, text, hand-written text etc. Convolutional Neural Networks are specifically used for this process because of their design to identify and process high-definition image data.

An image, based on the approach used, is considered either as a set of vectors (colour annotated polygons) or a raster (a table of pixels with numerical values for colors). The vector or raster is turned into simpler components that represent the constituent physical objects and features in an image. Computer vision systems can logically analyze these constructs, by

extracting the most important sections, and then by organizing data through feature extraction algorithms and classification algorithms.

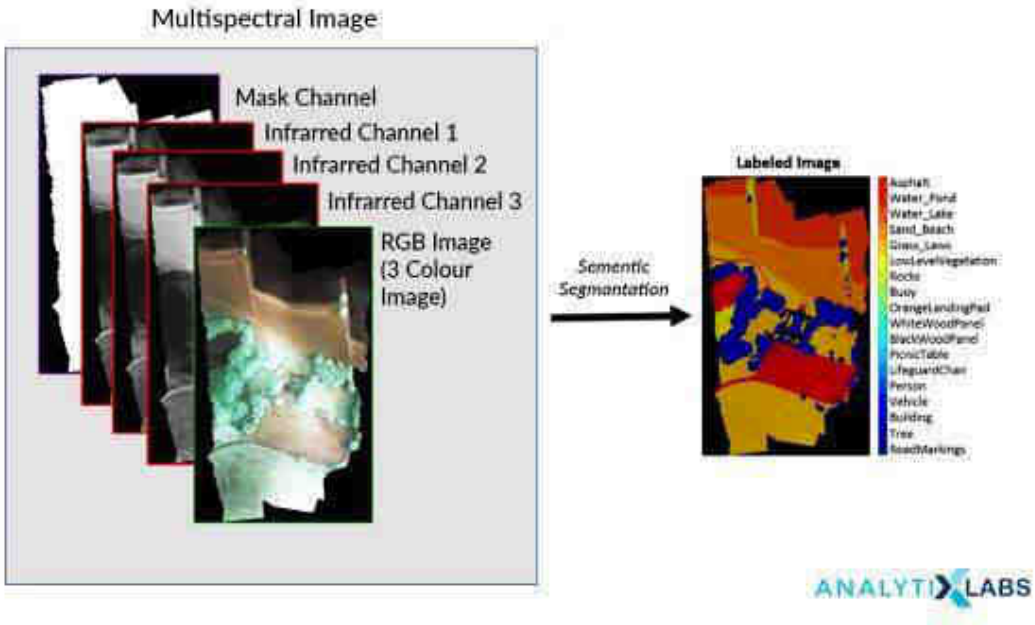


Image Source: mathworks.com

These algorithms are widely used in a variety of industries and applications. E-commerce industry uses it for providing relevant products to users for their search requirements and browsing history. The manufacturing industry uses it for anomaly detection, detecting damaged objects, ensuring worker safety etc. Image Recognition is famously used in education and training for visually impaired, speech impaired students. Although Neural Nets are time consuming when it comes to training the data, the end results have been very promising and the application of these has been highly successful.

UNIT V

Image Compression

Image compression is a type of data compression applied to digital images, to reduce their cost for storage or transmission. Algorithms may take advantage of visual perception and the statistical properties of image data to provide superior results compared with generic data compression methods which are used for other digital data.[1]

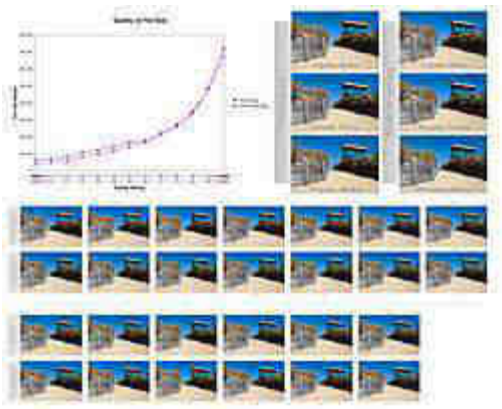


Image Compression Model

In the field of Image processing, the compression of images is an important step before we start the processing of larger images or videos. The compression of images is carried out by an encoder and output a compressed form of an image. In the processes of compression, the mathematical transforms play a vital role. A flow chart of the process of the compression of the image can be represented as:

In this article, we try to explain the overview of the concepts involved in the image compression techniques. The general representation of the image in a computer is like a vector of pixels. Each pixel is represented by a fixed number of bits. These bits determine the intensity of the color (on greyscale if a black and white image and has three channels of RGB if colored images.)

Why Do We Need Image Compression?

Consider a black and white image that has a resolution of 1000*1000 and each pixel uses 8 bits to represent the intensity. So the total no of bits req= 1000*1000*8 = 80,00,000 bits per image. And **consider if it is a video with 30 frames per second of the above-mentioned**

type images then the total bits for a video of 3 secs is: 3*(30*(8, 000, 000))=720, 000, 000 bits

As we see just to store a 3-sec video we need so many bits which is very huge. So, we need a way to have proper representation as well to store the information about the image in a minimum no of bits without losing the character of the image. Thus, image compression plays an important role.

Basic steps in image compression:

- Applying the image transform
- Quantization of the levels
- Encoding the sequences.

Transforming The Image

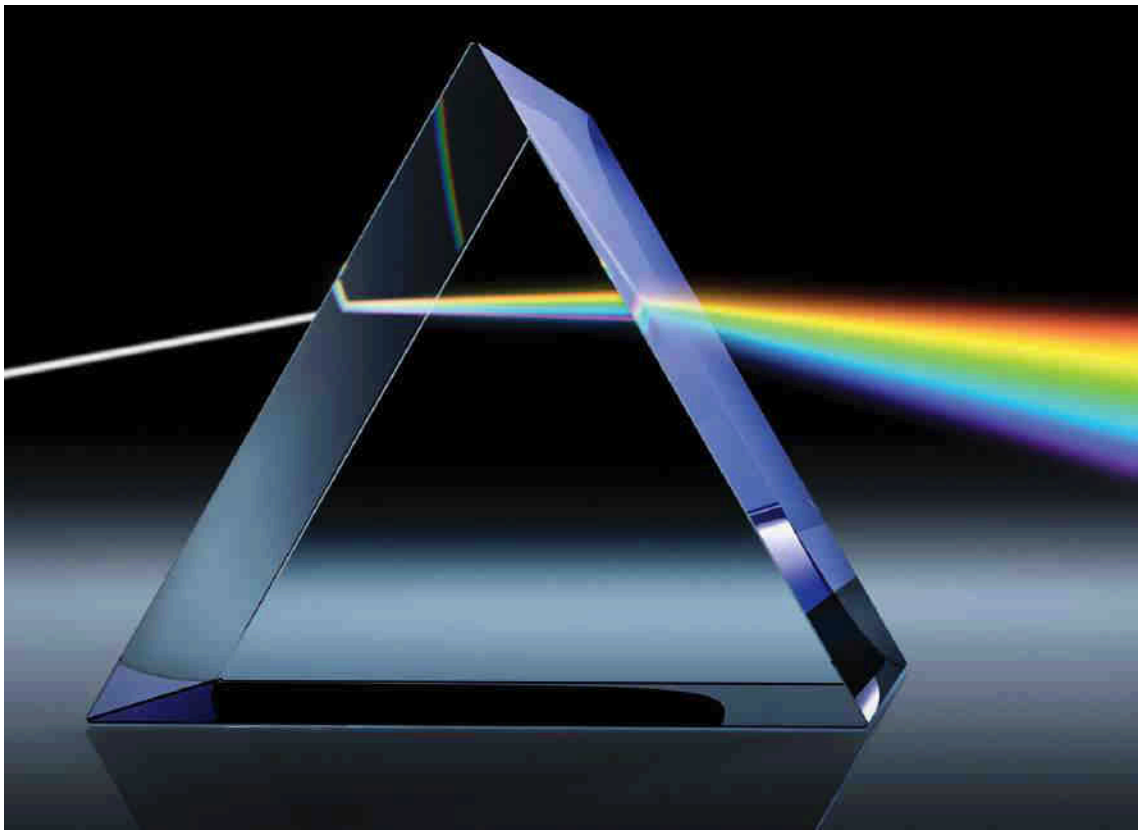
What is a transformation(Mathematically):

It a function that maps from one domain(vector space) to another domain(other vector space). Assume, **T is a transform, f(t):X->X' is a function then, T(f(t)) is called the transform of the function.**

In a simple sense, we can say that T changes the shape(representation) of the function as it i s a mapping from one vector space to another (without changing basic function f(t) i.e the r elationship between the domain and co-domain).

We generally carry out the transformation of the function from one vector space to the other because when we do that in the newly projected vector space we infer more information about the function.

A real life example of a transform:



Here we can say that the prism is a transformation function in which it splits the white light (f(t)) into its components i.e the representation of the white light. And we observe that we can infer more information about the light in its component representation than the white light one. This is how transforms help in understanding the functions in an efficient manner.

Transforms in Image Processing

The image is also a function of the location of the pixels. i.e I(x, y) where (x, y) are the coordinates of the pixel in the image. So we generally transform an image from the spatial domain to the frequency domain.

Why Transformation of the Image is Important:

- It becomes easy to know what all the principal components that make up the image and help in the compressed representation.
- It makes the computations easy.
    - Example: finding convolution in the time domain before the transformation:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau.$$

Finding convolution in the frequency domain after the transformation:

$$(f*g)(t)=F(s)G(s)$$

- So we can see that the computation cost has reduced as we switched to the frequency domain. We can also see that in the time domain the convolution was equivalent to an integration operator but in the frequency domain, it becomes equal to the simple product of terms. So, this way the cost of computation reduces.

So this way when we transform the image from domain to the other carrying out the spatial filtering operations becomes easier.

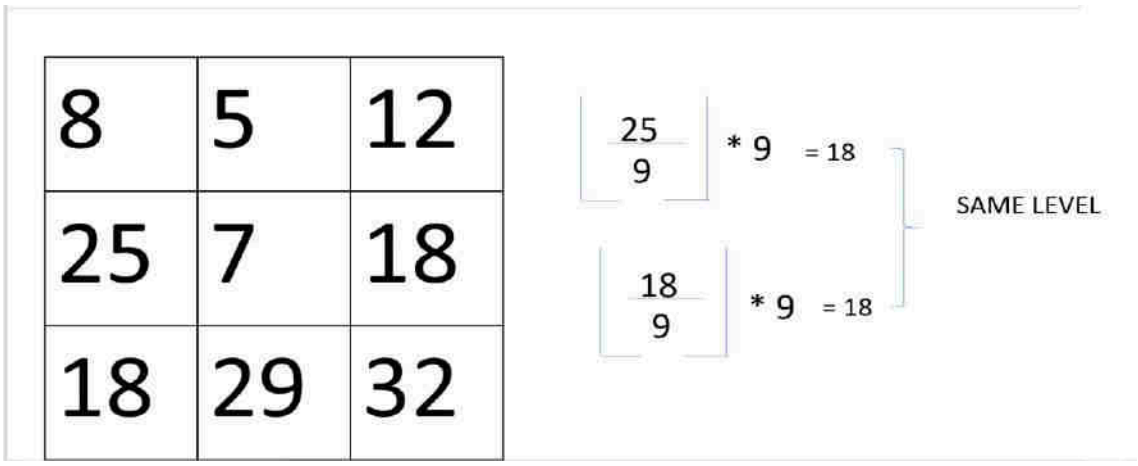
Quantization

The process quantization is a vital step in which the various levels of intensity are grouped into a particular level based on the mathematical function defined on the pixels. Generally, the newer level is determined by taking a fixed filter size of “m” and dividing each of the “m” terms of the filter and rounding it its closest integer and again multiplying with “m”.

*Basic quantization Function: [pixelvalue/m] * m*

So, the closest of the pixel values approximate to a single level hence as the no of distinct levels involved in the image becomes less. Hence we reduce the redundancy in the level of the intensity. So thus quantization helps in reducing the distinct levels.

Eg: (m=9)



Thus we see in the above example both the intensity values round up to 18 thus we reduce the number of distinct levels(characters involved) in the image specification.

Symbol Encoding

The symbol stage involves where the distinct characters involved in the image are encoded in a way that the no. of bits required to represent a character is optimal based on the frequency of the character's occurrence. In simple terms, In this stage codewords are generated for the different characters present. By doing so we aim to reduce the no. of bits required to represent the intensity levels and represent them in an optimum number of bits.

There are many encoding algorithms. Some of the popular ones are:

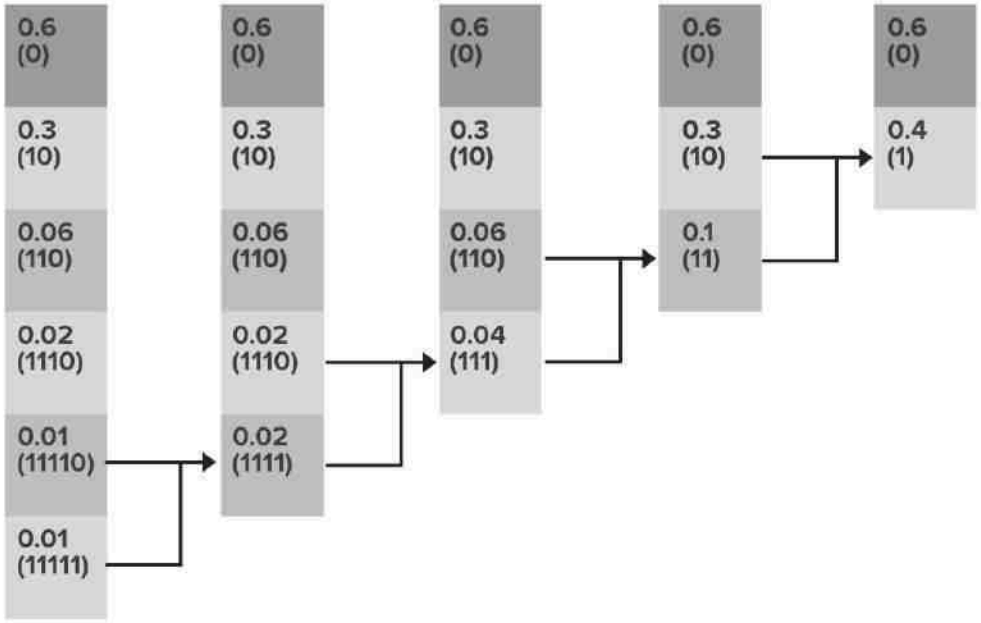
- Huffman variable-length encoding.
- Run-length encoding.

In the Huffman coding scheme, we try to find the codes in such a way that none of the codes are the prefixes to the other. And based on the probability of the occurrence of the character the length of the code is determined. In order to have an optimum solution the most probable character has the smallest length code.

Example:

SYMBOLS (with instensity value gray scale)	Probability (arranged in decreasing order)	Binary Code	Huffman code	Length of Huffman code
a1 - 18	0.6	00010010	0	1
a2 - 25	0.3	00011001	10	2
a3 - 255	0.06	11111111	110	3
A4 - 128	0.02	10000000	1110	4
a5 - 200	0.01	11001000	11110	5
a6 - 140	0.01	10001100	11111	5

We see the actual 8-bit representation as well as the new smaller length codes. The mechanism of generation of codes is:



So we see how the storage requirement for the no of bits is decreased as:

Initial representation–average code length: 8 bits per intensity level.

*After encoding–average code length: (0.6*1)+(0.3*2)+(0.06*3)+(0.02*4)+(0.01*5)+(0.01 *5)=1.56 bits per intensity level*

Thus the no of bits required to represent the pixel intensity is drastically reduced.

Thus in this way, the mechanism of quantization helps in compression. When the images are once compressed its easy for them to be stored on a device or to transfer them. And based on the type of transforms used, type of quantization, and the encoding scheme the decoders are designed based on the reversed logic of the compression so that the original image can be re-built based on the data obtained out of the compressed images

Information Theory

Information .eory (IT) tools, widely used in many scientific fields such as engineering, physics, genetics, neuroscience, and many others, are also useful transversal tools in image processing. In this book, we present the basic concepts of IT and how they have been used in the image processing areas of registration, segmentation, video processing, and computational aesthetics. Some of the approaches presented, such as the application of mutual information to registration, are the state of the art in the field. All techniques presented in this book have been previously published in peer-reviewed conference proceedings or international journals. We have stressed here their common aspects, and presented them in an unified way, so to make clear to the reader which problems IT tools can help to solve, which specific tools to use, and how to apply them. .e IT basics are presented so as to be self-contained in the book. .e intended audiences are students and practitioners of image processing and related areas such as computer graphics and visualization. In addition, students and practitioners of IT will be interested in knowing about these applications.

Lossy and Lossless Compression :

Data Compression refers to a technique where a large file to reduced to smaller sized file and can be decompressed again to the large file. Lossy compression restores the large file to its original form with loss of some data which can be considered as not-noticable while lossless compression restores the large file to its original form without any loss of data.

Following are some of the important differences between Lossy Compression and Lossless Compression.

Sr. No.	Key	Lossy Compression	Lossless Compression
1	Data Elimination	Lossy compression eliminates those bytes which are considered as not-noticable.	Lossless compression keeps even those bytes which are not-noticable.
2	Restoration	After lossy compression, a file cannot be restored to its original form.	After lossless compression, a file can be restored to its original form.
3	Quality	Lossy compression leads to compromise with quality.	No quality degradation happens in lossless compression.
4	Size	Lossy compression reduces the size of file to large extent.	Lossless compression reduces the size but less as compared to lossy compression.
5	Algorithm used	Transform coding, Discrete Cosine Transform, Discrete Wavelet transform, fractal compression etc.	Run length encoding, Lempel-Ziv-Welch, Huffman Coding, Arithmetic encoding etc.
6	Uses	Lossy compression is used to compress audio, video	Lossless compression is used to compress text,

Sr. No.	Key	Lossy Compression	Lossless Compression
		and images.	images and sound.
7	Capacity	Lossy compression technique has high data holding capacity.	Lossless compression has low data holding capacity as compared to lossy compression.

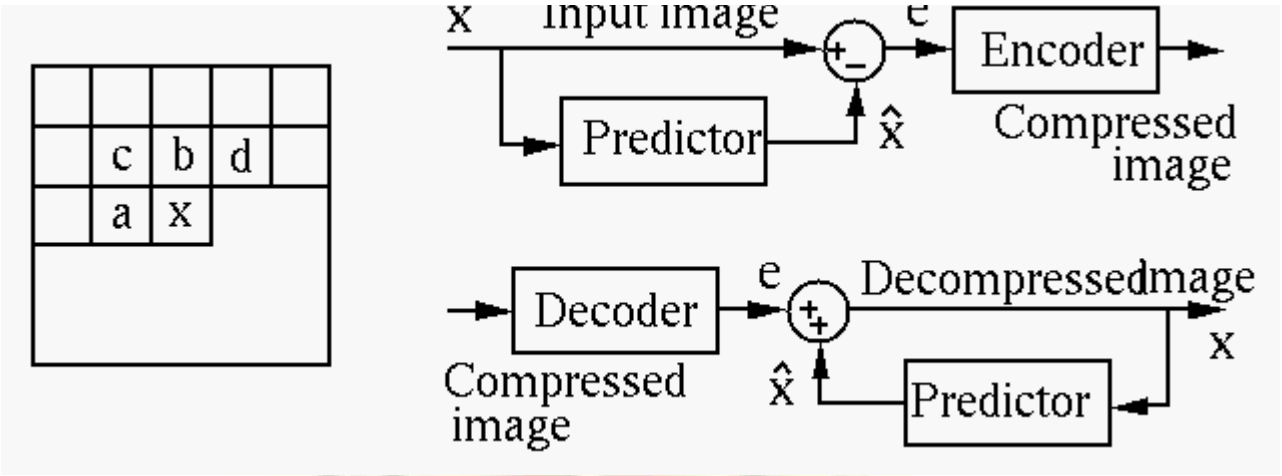
Predictive coding

As natural signals are highly correlated, the difference between neighboring samples is usually small. The value of a pixel x can be therefore predicted by its neighbors a, b, c, and d with a small error:

$$e_1 = x - a, \quad e_2 = x - b, \quad e_3 = x - \frac{a+b+c+d}{4}$$

In general, the predicted value $\hat{x}_0$ for a pixel x_0 is a linear combination of all available neighbors $a_i,\ (i = 1, \cdots, n)$:

$$\hat{x}_0 = \sum_{j=1}^{n} a_j x_j$$



The entropy of the histogram of the error image hist(e) is much smaller than that of the histogram of the original image hist(x), therefore Huffman coding will be much more effective for the error image than the original one.

Optimal predictive coding

The mean square error of the predictive error is:

$$E(e^2) = E[(x_0 - \hat{x}_0)^2] = E(x_0^2) - 2E(x_0\hat{x}_0) + E(\hat{x}_0^2)$$

where

$$\hat{x}_0 = \sum_{j=1}^{n} a_j x_j$$

To find the optimal coefficients a_i so that $E(e^2) \rightarrow min$ is minimized, we let

$$\frac{\partial}{\partial a_i} E(e^2) = \frac{\partial}{\partial a_i} E(\hat{x}_0^2) - 2\frac{\partial}{\partial a_i} E(x_0 \hat{x}_0) = 0$$

but as

$$2\frac{\partial}{\partial a_i} E(x_0 \hat{x}_0) = 2E(x_0 \frac{\partial}{\partial a_i} \sum_{j=1}^{n} a_j x_j) = 2E(x_0 \cdot x_i)$$

and

$$\frac{\partial}{\partial a_i} E(\hat{x}_0^2) = 2E(\hat{x}_0 \frac{\partial}{\partial a_i} \hat{x}_0) = 2E(\sum_{j=1}^{n} a_j x_j \cdot x_i) = 2\sum_{j=1}^{n} a_j E(x_j \cdot x_i)$$

we have

$$E(x_0 \cdot x_i) = \sum_{j=1}^{n} a_j E(x_j \cdot x_i)$$

Here

$$E(x_i x_j) = r_{ij}$$

is the correlation between x_i and x_j which can be estimated from data obtained from multiple trials:

$$\hat{r}_{ij} = \hat{E}(x_i x_j) = \frac{1}{K} \sum_{k=1}^{K} x_i^{(k)} x_j^{(k)}$$

Now the optimal prediction above can be written as

$$r_{0i} = \sum_{j=1}^{n} a_j r_{ij}, \quad (j = 1, \cdots, n)$$

which can be expressed in vector form:

$$\mathbf{r}_0 = R\mathbf{a}$$

where

$$\mathbf{r}_0 = \begin{bmatrix} r_{01} \\ \cdots \\ r_{0n} \end{bmatrix}, \qquad R = \begin{bmatrix} \ldots & \ldots & \ldots \\ \ldots & r_{ij} & \ldots \\ \ldots & \ldots & \ldots \end{bmatrix}$$

Then the coefficients can be found as

$$\mathbf{a} = R^{-1}\mathbf{r}_0$$

To prevent predictive error from being accumulated, we require

$$\sum_{j=1}^{n} a_j < 1$$

so that the errors will not propagate.

Examples

$$\hat{x}(i,j) = 0.9\, x(i,j-1)$$

$$\hat{x}(i,j) = 0.9\ x(i-1,j)$$

$$\hat{x}(i,j) = 0.7\ x(i,j-1) + 0.7\ x(i-1,j) - 0.5\ x(i-1,j-1)$$

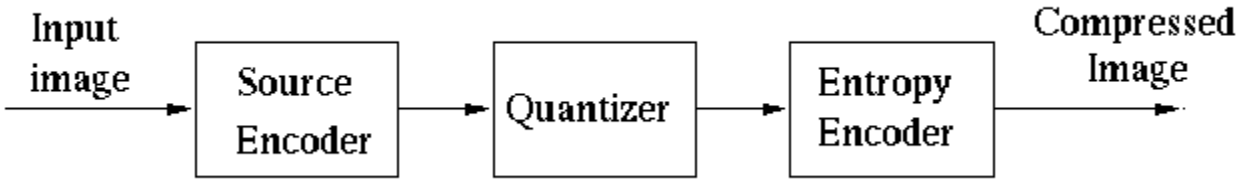
$$\hat{x}(i,j) = \begin{cases} 0.9\ x(i,j-1) & \text{if } \triangle h \leq \triangle v \\ 0.9\ x(i-1,j) & \text{else} \end{cases}$$

where

$$\triangle h = |x(i-1,j) - x(i-1,j-1)|, \quad \triangle v = |x(i,j-1) - x(i-1,j-1)|$$

Transform Coding (lossy) and JPEG Image Compression

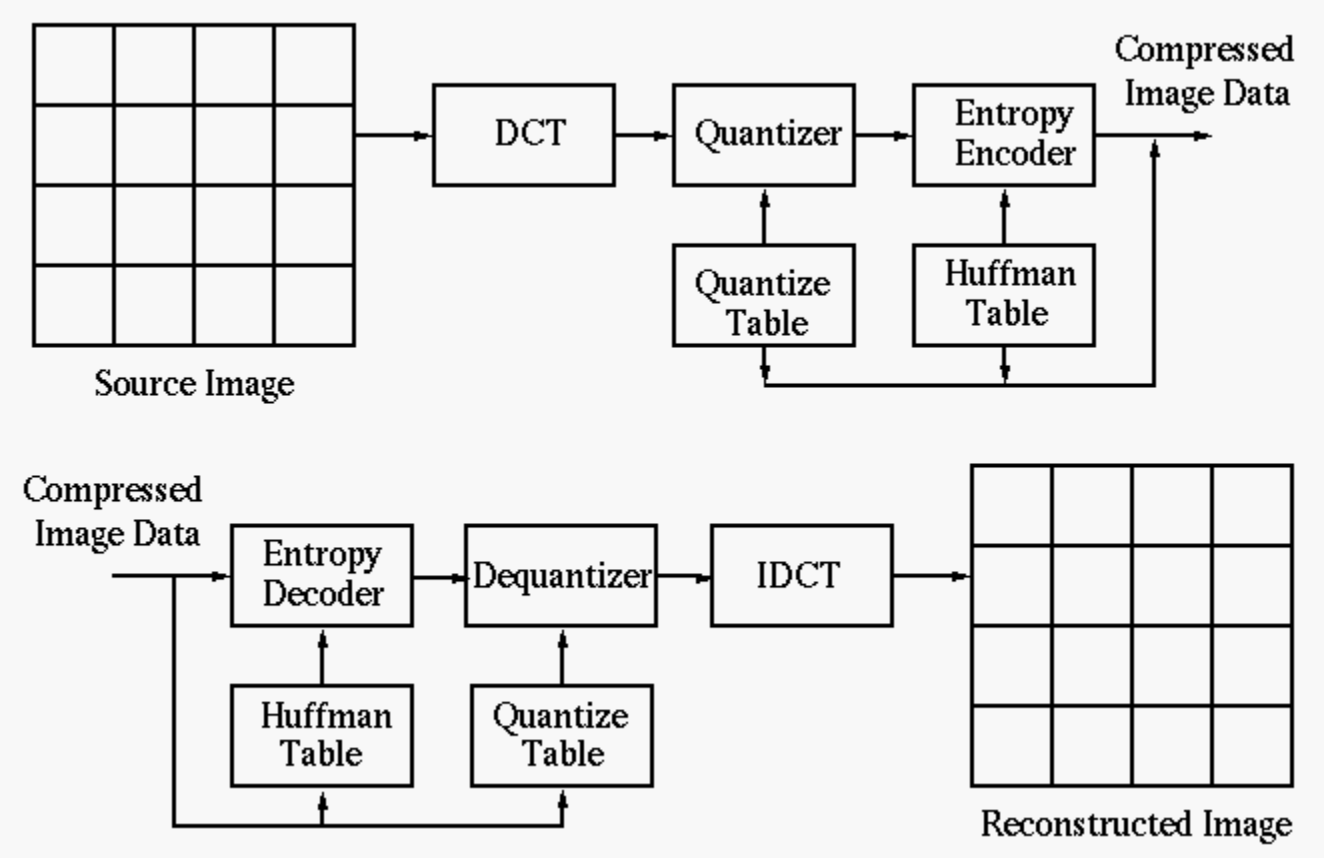
The Joint Photographic Experts Group (JPEG) is the working group of ISO, International Standard Organization, that defined the popular JPEG Imaging Standard for compression used in still image applications. The counter part in moving picture is the ``Moving Picture Experts Group" (MPEG).



JPEG compression is based on certain transform, either DCT or wavelet transform, due to the essential properties of orthogonal transforms in general:

- Decorrelation of the signal;
- Compaction of its energy.

Check this ACM page for review of DCT vs. wavelet transform used for image compression.



Here are the steps of JPEG image compression based on DCT:

1. Divide the image to form a set of 8×8 blocks and carry out 2D DCT transform of each block. The computational complexity for 2D DCT of an $N \times N$ image is $O(N^2 log_2 N)$, while the complexity of 2D DCT of all $N/8$ by $N/8$ blocks of image is

$$\frac{N^2}{8^2} O(8^2 log_2 8) = O(N^2)$$

The larger the image size N, the more saving by sub-block transform. As adjacent

pixels are highly correlated, most of energy in an 8 by 8 block is concentrated in the low frequency region of the spectrum (upper-left corner) and the rest transform coefficients are very close to zero.

2. Threshold all DCT coefficients smaller than a value T to zero, or alternatively, low-pass (either ideal or smooth) filter the 2D DCT spectrum of each sub-image;

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	2	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

3. Quantize remaining coefficients (convert floating-point values to integers). First, the elements in each block are divided (element-wise) by the elements in a quantization matrix Q:

$$y(i,j) = \left[\frac{x(i,j)}{Q(i,j)}\right]$$

where

$$Q = \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & q_{ij} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}_{8\times 8}$$

and each of the resulting 8 by 8 elements is rounded to the nearest integer (

$[x]$ represents rounding x to the closest integer). At the receiving end, the coefficients are recovered by:

$$\hat{x} = Q(i,j) \cdot y(i,j)$$

Two observations can be made:

- Larger $Q(i,j)$ causes larger error. Let $Q(i,j) = C$, and K be an integer as the rounding result of a pixel $x(i,j)$, then the possible value for the pixel is in the range:

$$KC - \frac{C}{2} \leq x(i,j) < KC + \frac{C}{2}$$

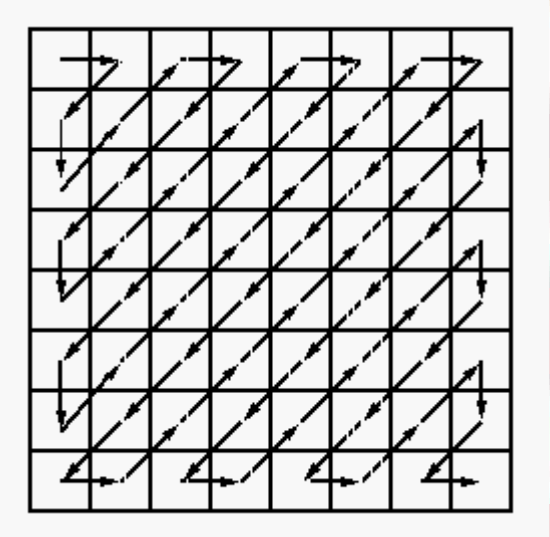
  i.e., the range of rounding error is proportional to $Q(i,j)$.

- Larger $Q(i,j) = C$ tends to suppress more pixels $x(i,j) < C/2$ to zero and they will not be recovered at the receiving end.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

In general, assign smaller numbers around the top-left corner (low frequency components) and larger ones around the lower-right corner (high frequency components). The values are also heuristically determined according to perceptual and psycho-visual tests.

4. Predictive code all DC components of the blocks (as the DC components are highly correlated);
5. Scan the rest coefficients in each block in a zigzag way (for higher probability of longer consecutive 0's) to code them by run-length encoding;



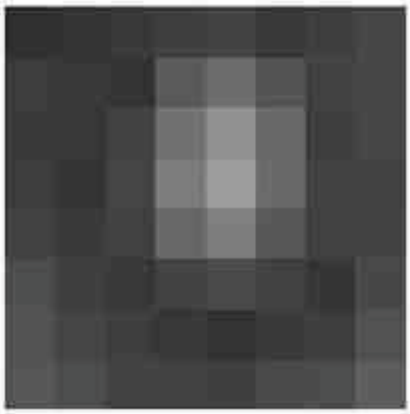
6. Huffman code the data stream;
7. Store and/or transmit the encoded image as well as the quantization matrix.

JPEG compression

JPEG stands for Joint photographic experts group. It is the first interanational standard in image compression. It is widely used today. It could be lossy as well as lossless . But the technique we are going to discuss here today is lossy compression technique.

How jpeg compression works

First step is to divide an image into blocks with each having dimensions of 8 x8.



Let's for the record, say that this 8x8 image contains the following values.

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

The range of the pixels intensities now are from 0 to 255. We will change the range from -128 to 127.

Subtracting 128 from each pixel value yields pixel value from -128 to 127. After subtracting 128 from each of the pixel value, we got the following results.

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

Now we will compute using this formula.

$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right]$$

$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

The result comes from this is stored in let's say A(j,k) matrix.

There is a standard matrix that is used for computing JPEG compression, which is given by a matrix called as Luminance matrix.

This matrix is given below

$$Q_{j,k} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Applying the following formula

$$B_{j,k} = \text{round}\left(\frac{A_{j,k}}{Q_{j,k}}\right)$$

We got this result after applying.

$$B_{j,k} = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now we will perform the real trick which is done in JPEG compression which is ZIG-ZAG movement. The zig zag sequence for the above matrix is shown below. You have to perform zig zag until you find all zeroes ahead. Hence our image is now compressed.

$$B_{j,k} = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Summarizing JPEG compression

The first step is to convert an image to Y'CbCr and just pick the Y' channel and break into 8 x 8 blocks. Then starting from the first block, map the range from -128 to 127. After that you have to find the discrete Fourier transform of the matrix. The result of this should be quantized. The last step is to apply encoding in the zig zag manner and do it till you find all zero.

Save this one dimensional array and you are done.

Note. You have to repeat this procedure for all the block of 8 x 8.

*****