Module 2

IMAGE ENHANCEMENT IN THE SPATIAL DOMAIN

Prepared By,

Sandesha Karanth P.K

Assistant Professors, Dept. Of CSE,

VCET, Puttur

writetokaranth@gmail.com[1], rkk4691@gmail.com[2]

Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term ***spatial domain*** refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. ***Frequency domain*** processing techniques are based on modifying the Fourier transform of an image.

The term *spatial domain* refers to the aggregate of pixels composing an image. Spatial domain methods are procedures that operate directly on these pixels. Spatial domain processes will be denoted by the expression

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

where f(x, y) is the input image, g(x, y) is the processed image, and *T* is an operator on *f*, defined over some neighborhood of (x, y). In addition, *T* can operate on a *set* of input images, such as performing the pixel-by-pixel sum of *K* images for noise reduction.

The principal approach in defining a neighborhood about a point (x, y) is to use a square or rectangular sub image area centered at (x, y), as shown in below figure 1.

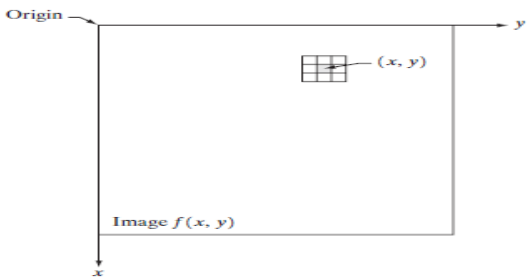

Figure 1: 3*3 neighborhood about a point (x, y) in an image.

The center of the sub image is moved from pixel to pixel starting, say, at the top left corner. The operator *T* is applied at each location (x, y) to yield the output, *g*, at that location. The process utilizes only the pixels in the area of the image spanned by the neighborhood. Although other neighborhood shapes, such as approximations to a circle, sometimes are used, square and rectangular arrays are by far the most predominant because of their ease of implementation.

The simplest form of *T* is when the neighborhood is of size 1*1 (that is, a single pixel). In this case, *g* depends only on the value of *f* at (x, y), and *T* becomes a *gray-level* (also called an *intensity* or *mapping*) *transformation function* of the form **s = T(r)** where, for simplicity in notation, *r* and *s* are variables denoting, respectively, the gray level of f(x, y) and g(x, y) at any point (x, y).

For example, if T(r) has the form shown in Fig. 2(a), the effect of this transformation would be to produce an image of higher contrast than the original by darkening the levels below *m* and brightening the levels above *m* in the original image. In this technique, known as ***contrast stretching,*** the values of *r* below *m* are compressed by the transformation function into a narrow range of *s*, toward black. The opposite effect takes place for values of *r* above *m*. In the limiting case shown in Fig. 2(b), T(r) produces a two-level (binary) image. A mapping of this form is called a ***thresholding*** function.

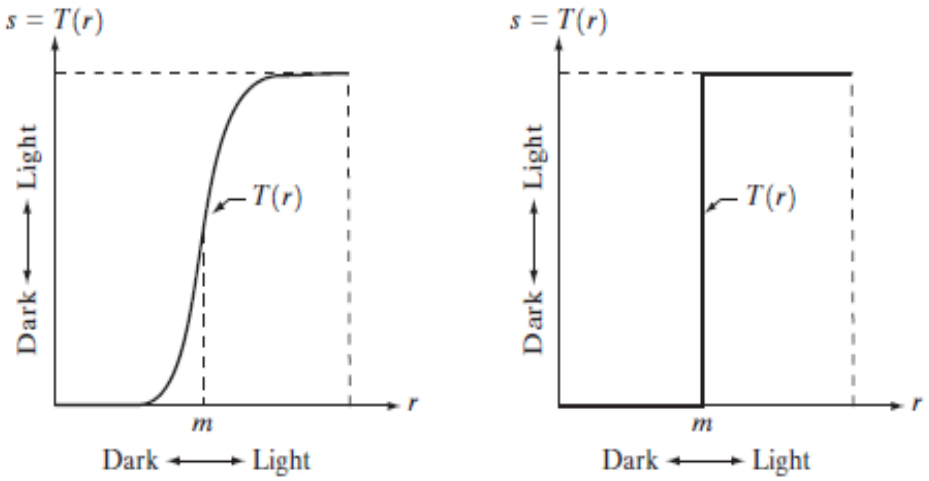

Figure 2 a,b : Gray level transformation functions for contrast enhancement.

Some Basic Gray Level Transformations:

1. Image Negatives:

The negative of an image with gray levels in the range [0,L-1]is obtained by using the negative transformation shown in Fig. 3.3, which is given by the expression

$$s = L - 1 - r$$

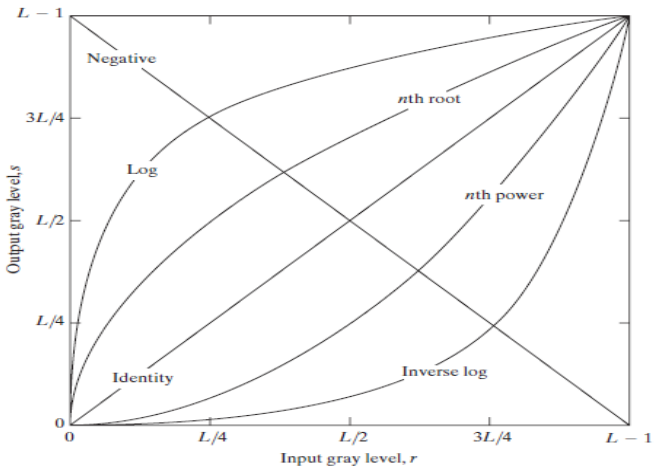

Figure 3: Some basic gray-level transformation functions used for image enhancement.

Reversing the intensity levels of an image produces the equivalent of a photographic negative. This type of processing is particularly suited for enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size.

2. Log Transformations:

The general form of the log transformation

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

where c is a constant, and it is assumed that $r \geq 0$.

The shape of the log curve in Fig. 3 shows that this transformation maps a narrow range of low gray-level values in the input image into a wider range of output levels. The opposite is true of higher values of input levels. We would use a transformation of this type to expand the values of dark pixels in an image while compressing the higher-level values. The opposite is true of the inverse log transformation.

3. Power-Law Transformations

Power-law transformations have the basic form

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

where ***c*** and γ are positive constants, Also can be represented as

$$\mathbf{s = c\ (r+\varepsilon)^{\gamma}}$$

An offset measurable when input is zero

We see in Fig. 4 that curves generated with values of g>1 have exactly the opposite effect as those generated with values of g<1. Finally, we note that from above equation reduces to the identity transformation when c=g=1.

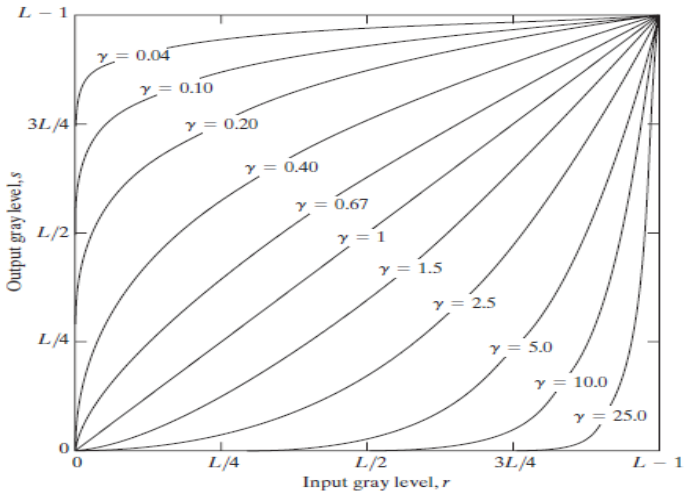

Figure 4: Plots of the equation $\mathbf{s = c\ r^{\gamma}}$ for various values of g (c=1 in all cases).

Plots of *s* versus *r* for various values of g are shown in Fig. 4. As in the case of the log transformation, power-law curves with fractional values of g map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels.

A variety of devices used for image capture, printing, and display respond according to a power law. By convention, the exponent in the power-law equation is referred to as *gamma.* The process used to correct this power-law response phenomenon is called *gamma correction*. Gamma correction is straightforward. All we need to do is preprocess the input image before inputting it into the monitor by performing the transformation

$$s = r^{1/2.5} = r^{0.4}$$

Gamma correction is important if displaying an image accurately on a computer screen is of concern. Images that are not corrected properly can look either bleached out, or, what is more likely, too dark.

For example, cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5.With reference to the curve for g=2.5 in Fig. 3.6, we see that such display systems would tend to produce images that are darker than intended.

Contrast stretching

One of the simplest piecewise linear functions is a contrast-stretching transformation. Low-contrast images can result from poor illumination, lack of dynamic range in the imaging sensor, or even wrong setting of a lens aperture during image acquisition. The idea behind contrast stretching is to increase the dynamic range of the gray levels in the image being processed.

- If r1,s1 and r2,s2 control the shape of the transformation function.and if r1=s1 and r2=s2 the transformation is a linear function that produces no changes in intensity levels !
- If r1=r2 and s1=0 and s2 =L-1, the transformation becomes a thresholding function that creates a binary image.
- In general r1≤r2 and s1 ≤s2 is assumed so that the function is single valued and monotonically increasing. This preserves the order of intensity levels, thus preventing the creation of intensity artifacts in the processed image.

Gray level slicing

It is highlighting a specific range of intensities in an image often is of interest. Its applications include enhancing features such as masses of water in satellite imagery and enhancing flaws in X-ray images.

- The intensity level slicing process can be implemented for these defects/noise etc.
- It can be done in several ways

Ex: One is to display in one value (say, white), all the values in the range of interest and in another (say, black), all other intensities.

Bit-plane slicing

Instead of highlighting gray-level ranges, highlighting the contribution made to total image appearance by specific bits might be desired. Suppose that each pixel in an image is represented by 8 bits. Imagine that the image is composed of eight 1-bit planes, ranging from bit-plane 0 for the least significant bit to bit plane 7 for the most significant bit. In terms of 8-bit bytes, plane 0 contains all the lowest order bits in the bytes comprising the pixels in the image and plane 7 contains all the high-order bits.

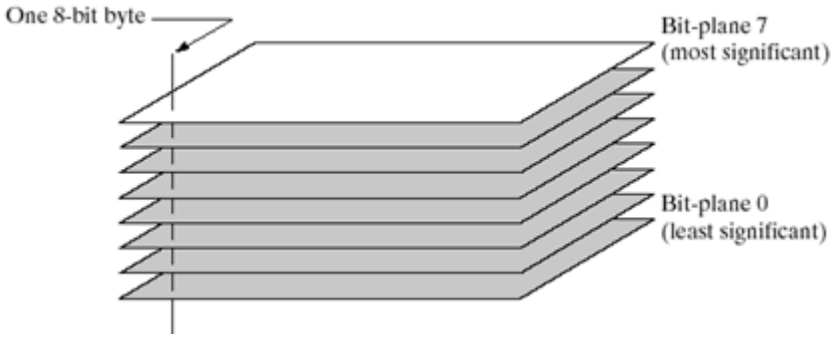

Figure 5: Bit-plane representation of an 8-bit image.

Decomposing an image into its bit planes is useful for analyzing the relative importance of each bit in the image, a process that aids in determining the adequacy of the number of bits used to quantize the image. It is useful in image compression.

- The reconstruction is done by using few planes only.
- It is done by multiplying the pixels of the n^{th} plane by a constant 2^{n-1}.
- All the generated planes are added together (few of 8 planes)
- If we use bit plane 7 and 8, multiply bit plane 8 by 128 and plane 7 by 64 and then added together.

Histogram Processing

The histogram of a digital image with gray levels in the range [0, L-1] is a discrete function $h(r_k)=n_k$, where r_k is the k^{th} gray level and n_k is the number of pixels in the image having gray level r_k. Histograms are the basis for numerous spatial domain processing techniques. Histogram manipulation can be used effectively for image enhancement. Histograms are simple to calculate

in software and also lend themselves to economic hardware implementations, thus making them a popular tool for real-time image processing.

The horizontal axis of each histogram plot corresponds to gray level values, r_k. The vertical axis corresponds to values of

$$\mathbf{H(r_k)=n_k \text{ or } p(r_k)=n_k/n} \text{ if the values are normalized.}$$

Thus, as indicated previously, these histogram plots are simply plots of $\mathbf{h(r_k)=n_k}$ versus $\mathbf{r_k}$ or $\mathbf{p(r_k)=n_k/n}$ versus $\mathbf{r_k}$.

Histogram Equalization

Consider for a moment continuous functions, and let the variable *r* represent the gray levels of the image to be enhanced. In the initial part of our discussion we assume that *r* has been normalized to the interval [0, 1], with r=0 representing black and r=1 representing white. Later, we consider a discrete formulation and allow pixel values to be in the interval [0, L-1]. For any *r* satisfying the aforementioned conditions, we focus attention on transformations of the form

$$\mathbf{s=T(r) \quad 0 \le r \le 1 \quad \ldots\ldots(1)}$$

that produce a level *s* for every pixel value *r* in the original image. For reasons that will become obvious shortly, we assume that the transformation function T(r) satisfies the following conditions:

a. T(r) is a **monotonically increasing** function in the interval $0 \le r \le L-1$:

   T(r) be single valued is needed to guarantee that the inverse transformation will exist, and the monotonicity condition preserves the increasing order from black to white in the output image.

b. $\mathbf{0 \le T(r) \le L-1}$ for $0 \le r \le L-1$:

   It guarantees that the output gray levels will be in the same range as the input levels.

Figure 6 gives an example of a transformation function that satisfies these two conditions.

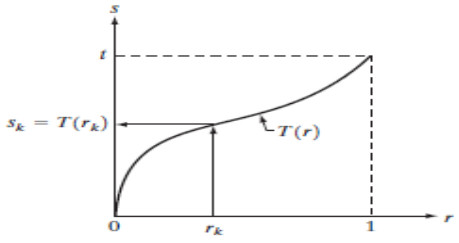

Figure 6 A gray-level transformation functions that is both single valued and monotonically increasing.

The inverse transformation from *s* back to *r* is denoted

$$\mathbf{r = T^{-1}(s) \qquad 0 \le s \le 1 \quad \ldots\ldots(2)}$$

The gray levels in an image may be viewed as random variables in the interval [0, 1]. One of the most fundamental descriptors of a random variable is its probability density function (PDF). Let $p_r(r)$ and $p_s(s)$ denote the probability density functions of random variables *r* and *s*, respectively, where the subscripts on *p* are used to denote that pr and p_s are different functions. A basic result from an elementary probability theory is that, if $p_r(r)$ and T(r) are known and $T^{-1}(s)$ satisfies condition(a) specified, then the probability density function ps(s) of the transformed variable *s* can be obtained using a rather simple formula:

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| \quad \ldots\ldots\ldots(3)$$

A transformation function of particular importance in image processing has the form

$$s = T(r) = \int_0^r p_r(w)\, dw \quad \ldots\ldots\ldots(4)$$

where w is a dummy variable of integration. From Leibniz's rule in calculus

$$\begin{aligned}\frac{ds}{dr} &= \frac{dT(r)}{dr} \\ &= \frac{d}{dr}\left[\int_0^r p_r(w)\, dw\right] \\ &= p_r(r). \end{aligned} \quad \ldots\ldots\ldots(5)$$

Substituting this result for *dr*/*ds* into Eq. (3), and keeping in mind that all probability values are positive, yields

$$\begin{aligned} p_s(s) &= p_r(r)\left|\frac{dr}{ds}\right| \\ &= p_r(r)\left|\frac{1}{p_r(r)}\right| \\ &= 1 \qquad 0 \le s \le 1. \end{aligned} \quad \ldots.(6)$$

For discrete values we deal with probabilities and summations instead of probability density functions and integrals. The probability of occurrence of gray level r_k in an image is approximated by

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L-1 \qquad \ldots\ldots(7)$$

The discrete version of the transformation function

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j)$$
$$= \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0, 1, 2, \ldots, L-1 \qquad \ldots..(8)$$

The transformation (mapping) given in Eq.(8) is called ***histogram equalization*** or ***histogram linearization***.

The inverse transformation from *s* back to *r* is denoted by

$$r_k = T^{-1}(s_k) \qquad k = 0, 1, 2, \ldots, L-1 \qquad \ldots\ldots.(9)$$

Histogram Matching

As indicated in the preceding discussion, histogram equalization automatically determines a transformation function that seeks to produce an output image that has a uniform histogram. When automatic enhancement is desired, this is a good approach because the results from this technique are predictable and the method is simple to implement. But in few applications, its required to extract specified histogram then the method used to generate a processed image that has a specified histogram is called ***histogram matching*** or ***histogram specification***.

Let us return for a moment to continuous gray levels r and z and let $p_r(r)$ and $p_z(z)$ denote their corresponding continues probability density functions. In this notation, r and z denote the gray levels of the input and output (processed) images, respectively. We can estimate $p_r(r)$ from the given input image, while $p_z(z)$ is the *specified* probability density function that we wish the output image to have.

Let *s* be a random variable with the property

$$s = T(r) = \int_0^r p_r(w)\, dw \qquad \ldots\ldots.(10)$$

where ‘w’(omega)is a dummy variable of integration.

Suppose next that we define a random variable *z* with the property

$$G(z) = \int_0^z p_z(t)\,dt = s \qquad \ldots\ldots(11)$$

where *t* is a dummy variable of integration. It then follows from these two equations that G(z)=T(r) and, therefore, that *z* must satisfy the condition

$$z = G^{-1}(s) = G^{-1}[T(r)]. \qquad \ldots..(12)$$

The transformation T(r) can be obtained from Eq. (10) once $p_r(r)$ has been estimated from the input image. Similarly, the transformation function G(z) can be obtained using Eq. (11) because $p_z(z)$ is given.

The discrete formulation of Eq. (3.3-10) is given by Eq. (3.3-8), which we repeat here for convenience:

$$\begin{aligned} s_k = T(r_k) &= \sum_{j=0}^{k} p_r(r_j) \\ &= \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0, 1, 2, \ldots, L-1 \end{aligned} \qquad \ldots.(13)$$

where *n* is the total number of pixels in the image, nj is the number of pixels with gray level r_j , and *L* is the number of discrete gray levels. Similarly, the discrete formulation of Eq. (11) is obtained from the given histogram $p_z(z_i)$,i=0,1, 2,p ,L-1, and has the form

$$v_k = G(z_k) = \sum_{i=0}^{k} p_z(z_i) = s_k \qquad k = 0, 1, 2, \ldots, L-1. \qquad \ldots..(14)$$

Finally, the discrete version of Eq. (12) is given by

$$z_k = G^{-1}[T(r_k)] \qquad k = 0, 1, 2, \ldots, L-1 \qquad \ldots\ldots(15)$$

or, from Eq. (13),

$$z_k = G^{-1}(s_k) \qquad k = 0, 1, 2, \ldots, L-1 \qquad \ldots\ldots.(16)$$

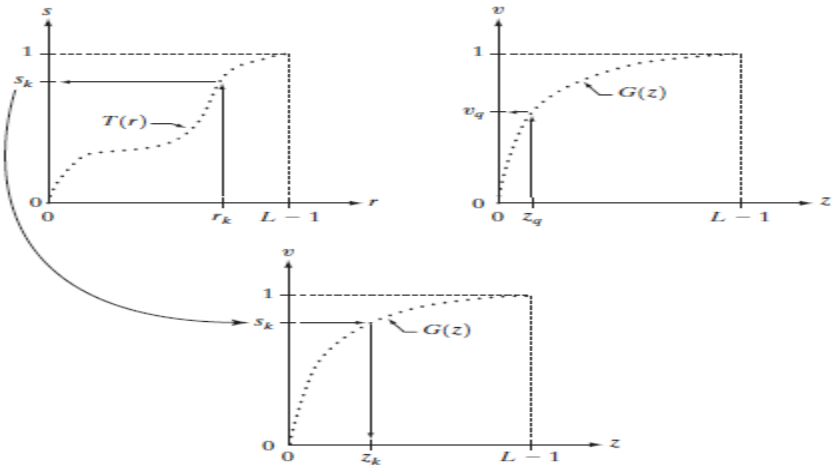

Figure 7(a) Graphical interpretation of mapping from r_k to s_k via T(r). (b) Mapping of z_q to its corresponding value v_q via G(z). (c) Inverse mapping from s_k to its corresponding value of z_k.

The procedure we have just developed for histogram matching may be summarized as follows:

1. Obtain the histogram of the given image.

2. Use Eq. (13) to pre-compute a mapped level s_k for each level r_k.

3. Obtain the transformation function *G* from the given $p_z(z)$ using Eq. (14).

4. Pre-compute z_k for each value of s_k using the iterative scheme defined in connection with Eq.

5. For each pixel in the original image, if the value of that pixel is r_k, map this value to its corresponding level s_k; then map level s_k into the final level z_k. Use the pre-computed values from Steps (2) and (4) for these mappings.

Enhancement Using Arithmetic/Logic Operations

Arithmetic/logic operations involving images are performed on a pixel-by-pixel basis between two or more images (this excludes the logic operation NOT, which is performed on a single image). Logic operations similarly operate on a pixel-by-pixel basis. We need only be concerned with the ability to implement the AND, OR, and NOT logic operators because these three operators are functionally complete. When dealing with logic operations on gray-scale images, pixel values are processed as strings of binary numbers. For example, performing the NOT operation on a black, 8-bit pixel (a string of eight 0's) produces a white pixel (a string of eight 1's). Intermediate values are processed the same way, changing all 1's to 0's and vice versa.

The four arithmetic operations, subtraction and addition (in that order) are the most useful for image enhancement.

$$s(x, y) = f(x, y) + g(x, y)$$
$$d(x, y) = f(x, y) - g(x, y)$$
$$p(x, y) = f(x, y) \times g(x, y)$$
$$v(x, y) = f(x, y) \div g(x, y)$$

Image Subtraction:

The difference between two images f(x, y) and h(x, y), expressed as

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

And it is obtained by computing the difference between all pairs of corresponding pixels from *f* and *h*. The key usefulness of subtraction is the enhancement of differences between images.

One of the most commercially successful and beneficial uses of image subtraction is in the area of medical imaging called mask mode radiography. In this case h(x, y), the *mask*, is an X-ray image of a region of a patient's body captured by an intensified TV camera (instead of traditional X-ray film) located opposite an X-ray source. The procedure consists of injecting a contrast medium into the patient's bloodstream, taking a series of images of the same anatomical region as h(x, y), and subtracting this mask from the series of incoming images after injection of the contrast medium. The net effect of subtracting the mask from each sample in the incoming stream of TV images is that the areas that are different between f(x, y) and h(x, y) appear in the output image as enhanced detail.

Image Averaging:

Consider a noisy image g(x, y) formed by the addition of noise h(x, y) to an original image f(x, y); that is,

$$g(x, y) = f(x, y) + h(x, y$$

where the assumption is that at every pair of coordinates (x, y) the noise is uncorrelated and has zero average value. The objective of the following procedure is to reduce the noise content by adding a set of noisy images, $\{g_i(x, y)\}$.

If the noise satisfies the constraints just stated, it can be shown that if an image $\bar{g}(x, y)$ is formed by averaging *K* different noisy images,

$$\bar{g}(x, y) = \frac{1}{K} \sum_{i=1}^{K} g_i(x, y)$$

An important application of image averaging is in the field of astronomy, where imaging with very low light levels is routine, causing sensor noise frequently to render single images virtually useless for analysis.

Multiplication and Division:

We consider division of two images simply as multiplication of one image by the reciprocal of the other. Aside from the obvious operation of multiplying an image by a constant to increase its average gray level, image multiplication finds use in enhancement primarily as a masking operation that is more general than the logical masks discussed in the previous paragraph. In other words, multiplication of one image by another can be used to implement gray-level, rather than binary, masks.

Basics of Spatial Filtering:

Some neighborhood operations work with the values of the image pixels in the neighborhood and the corresponding values of a sub image that has the same dimensions as the neighborhood. The sub image is called a filter, mask, kernel, template, or window, with the first three terms being the most prevalent terminology. The values in a filter sub image are referred to as coefficients, rather than pixels.

The mechanics of spatial filtering are illustrated in Fig 8.The process consists simply of moving the filter mask from point to point in an image. At each point (x, y), the response of the filter at that point is calculated using a predefined relationship. For linear spatial, the response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask.

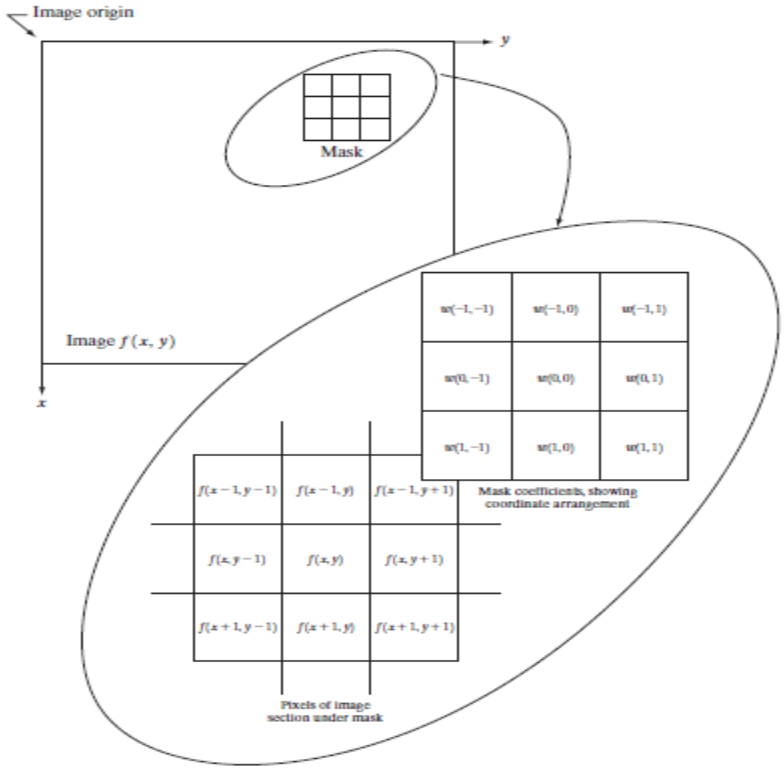

Figure 8 The mechanics of spatial filtering. The magnified drawing shows a 3*3 mask and the image section directly under it; the image section is shown displaced out from under the mask for ease of readability.

For the 3*3 mask shown in Fig. 3.32, the result (or response), *R*, of linear filtering with the filter mask at a point (x, y) in the image is

$$R = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \cdots + w(0,0)f(x,y) + \cdots + w(1,0)f(x+1,y) + w(1,1)f(x+1,y+1)$$

which we see is the sum of products of the mask coefficients with the corresponding pixels directly under the mask. Note in particular that the coefficient w(0, 0) coincides with image value f(x, y), indicating that the mask is centered at (x, y) when the computation of the sum of products takes place. For a mask of size m*n, we assume that m=2a+1 and n=2b+1,where a and b are nonnegative integers. All this says is that our focus in the following discussion will be on masks of *odd* sizes, with the smallest meaningful size being 3*3 (we exclude from our discussion the trivial case of a 1*1 mask).

In general, linear filtering of an image f of size M*Nwith a filter mask of size m*n is given by the expression:

$$g(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+s,y+t)$$

where, from the previous paragraph, a=(m-1)/2 and b=(n-1)/2. To generate a complete filtered image this equation must be applied for x=0, 1, 2, p , M-1 and y=0, 1, 2, p , N-1.

When interest lies on the response, *R*, of an m*n mask at any point (x, y), and not on the mechanics of implementing mask convolution, it is common practice to simplify the notation by using the following expression:

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} = \sum_{i=1}^{mn} w_i z_i$$

where the w's are mask coefficients, the *z*'s are the values of the image gray levels corresponding to those coefficients, and mn is the total number of coefficients in the mask.

Smoothing Spatial Filter

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing steps, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering.

Smoothing Linear Filters:

The output (response) of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called averaging filters.

The idea behind smoothing filters is straightforward. By replacing the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in gray levels. Because random noise typically consists of sharp transitions in gray levels, the most obvious application of smoothing is noise reduction. However, edges (which almost always are desirable features of an image) also are characterized by sharp transitions in gray levels, so averaging filters have the undesirable side effect that they blur edges.

A major use of averaging filters is in the reduction of "irrelevant" detail in an image. By "irrelevant" we mean pixel regions that are small with respect to the size of the filter mask. The

Figure shows two 3*3 smoothing filters. Use of the first filter yields the standard average of the pixels under the mask.

$$R = \frac{1}{9}\sum_{i=1}^{9} z_i,$$

This is the average of the gray levels of the pixels in the 3*3 neighborhood defined by the mask. Note that, instead of being 1_9, the coefficients of the filter are all 1's. An m*n mask would have a normalizing constant equal to 1_mn. A spatial averaging filter in which all coefficients are equal is sometimes called a box filter.

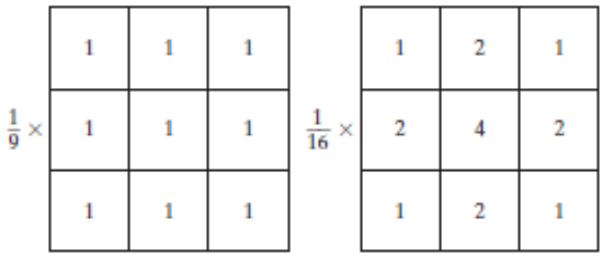

Figure 9 Two 3*3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to the sum of the values of its coefficients, as is required to compute an average.

The second mask shown in Fig. 9 is a little more interesting. This mask yields a so-called *weighted average*, terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask shown in Fig. 9(b) the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average. The other pixels are inversely weighted as a function of their distance from the center of the mask.

The general implementation for filtering an M*N image with a weighted averaging filter of size m*n (m and *n* odd) is given by the expression

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t) f(x+s, y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)}$$

For x=0, 1, 2, p , M-1 and y=0, 1, 2, p , N-1.

Sharpening Spatial Filters:

The principal objective of sharpening is to highlight fine detail in an image or to enhance detail that has been blurred, either in error or as a natural effect of a particular method of image acquisition.

Image sharpening vary and include applications ranging from electronic printing and medical imaging to industrial inspection and autonomous guidance in military systems.

Sharpening filters that are based on first- and second-order derivatives. The derivatives of a digital function are defined in terms of differences. There are various ways to define these differences. However, we require that any definition we use for a first derivative (1) must be zero in flat segments (areas of constant gray-level values); (2) must be nonzero at the onset of a gray-level step or ramp; and (3) must be nonzero along ramps. Similarly, any definition of a second derivative (1) must be zero in flat areas; (2) must be nonzero at the onset and end of a gray-level step or ramp; and (3) must be zero along ramps of constant slope.

A basic definition of the first-order derivative of a one-dimensional function f(x) is the difference

$$\frac{\partial f}{\partial x} = f(x+1) - f(x).$$

Similarly, we define a second-order derivative as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

It is easily verified that these two definitions satisfy the conditions stated previously regarding derivatives of the first and second order.

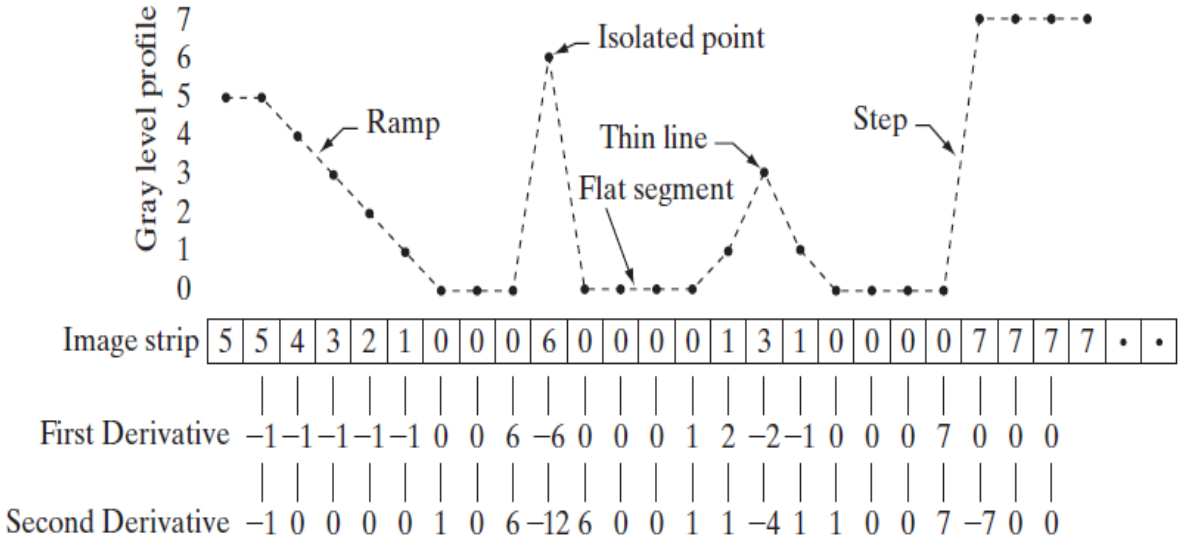

Figure 10 Simplified profile (the points are joined by dashed lines to simplify interpretation).

Figure 10 shows a simplification of the profile, with just enough numbers to make it possible for us to analyze how the first- and second-order derivatives behave as they encounter a noise point, a line, and then the edge of an object. In our simplified diagram the transition in the ramp spans four pixels, the noise point is a single pixel, the line is three pixels thick, and the transition into the gray-level step takes place between adjacent pixels. The number of gray levels was simplified to only eight levels.

Let us consider the properties of the first and second derivatives as we traverse the profile from left to right.

1. The first-order derivative is nonzero along the entire ramp,
2. The second-order derivative is nonzero only at the onset and end of the ramp.
   We conclude that first-order derivatives produce "thick" edges and second-order derivatives, much finer ones.
3. We encounter the isolated noise point. Here, the response at and around the point is much stronger for the second- than for the first-order derivative.
4. A second-order derivative is much more aggressive than a first-order derivative in enhancing sharp changes. Thus, we can expect a second-order derivative to enhance fine detail (including noise) much more than a first-order derivative.
5. The thin line is a fine detail, and we see essentially the same difference between the two derivatives. If the maximum gray level of the line had been the same as the isolated point, the response of the second derivative would have been stronger for the latter.
6. Finally, in this case, the response of the two derivatives is the same at the gray-level step. We also note that the second derivative has a transition from positive back to negative.

Comparison B/N first- and second-order derivatives response

1. First-order derivatives generally produce thicker edges in an image.
2. Second-order derivatives have a stronger response to fine detail, such as thin lines and isolated points.
3. First order derivatives generally have a stronger response to a gray-level step.
4. Second-order derivatives produce a double response at step changes in gray level.

Use of Second Derivatives for Enhancement–The Laplacian

The approach basically consists of defining a discrete formulation of the second-order derivative and then constructing a filter mask based on that formulation. We are interested in *isotropic* filters, whose response is independent of the direction of the discontinuities in the image to which the filter is applied. In other words, isotropic filters are *rotation invariant*, in the sense that rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.

It can be shown that the simplest isotropic derivative operator is the *Laplacian*, which, for a function (image) f(x, y) of two variables, is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad \ldots\ldots(1)$$

Because derivatives of any order are linear operations, the Laplacian is a linear operator**.** The definition of the digital second derivative given in that section is one of the most used. Taking into account that we now have two variables, we use the following notation for the partial second-order derivative in the *x*-direction:

$$\frac{\partial^2 f}{\partial^2 x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad \ldots..(2)$$

and, similarly in the *y*-direction, as

$$\frac{\partial^2 f}{\partial^2 y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad \ldots..(3)$$

The digital implementation of the two-dimensional Laplacian in Eq. (1) is obtained by summing these two components:

$$\nabla^2 f = \left[f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)\right] - 4f(x, y) \quad \ldots..(4)$$

This equation can be implemented using the mask shown in Fig. 3.39(a),

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

Figure 11Filter mask used to implement the digital Laplacian, as defined in Eq. (4). (b) Mask used to implement an extension of this equation that includes the diagonal neighbors. (c) and (d) Two other implementations of the Laplacian.

Because the Laplacian is a derivative operator, its use highlights gray-level discontinuities in an image and deemphasizes regions with slowly varying gray levels.This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be “recovered” while still preserving the sharpening effect of the Laplacian operation simply by adding the original and Laplacian images.

As noted in the previous paragraph, it is important to keep in mind which definition of the Laplacian is used. If the definition used has a negative center coefficient, then we *subtract*, rather than add, the Laplacian image to obtain a sharpened result. Thus, the basic way in which we use the Laplacian for image enhancement is as follows:

$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y) & \text{if the center coefficient of the Laplacian mask is negative} \\ f(x,y) + \nabla^2 f(x,y) & \text{if the center coefficient of the Laplacian mask is positive.} \end{cases} \quad \ldots.(5)$$

Unsharp masking and high-boost filtering

A process used for many years in the publishing industry to sharpen images consists of subtracting a blurred version of an image from the image itself.This process, called *unsharp masking*, is expressed as

$$f_s(x, y) = f(x, y) - \bar{f}(x, y) \quad \text{....(7)}$$

where $f_s(x, y)$ denotes the sharpened image obtained by unsharp masking, and $\bar{f}(x, y)$ is a blurred version of f(x, y).The origin of unsharp masking is in darkroom photography, where it consists of clamping together a blurred negative to a corresponding positive film and then developing this combination to produce a sharper image.

A slight further generalization of unsharp masking is called *high-boost filtering*.A high-boost filtered image, f_{hb}, is defined at any point (x, y) as

$$f_{hb}(x, y) = Af(x, y) - \bar{f}(x, y) \quad \text{.....(8)}$$

where $A \geq 1$ and, as before, is a blurred version of f. This equation may be written as

$$f_{hb}(x, y) = (A - 1)f(x, y) + f(x, y) - \bar{f}(x, y) \quad \text{.....(9)}$$

By using Eq. (7), we obtain

$$f_{hb}(x, y) = (A - 1)f(x, y) + f_s(x, y) \quad \text{.....(10)}$$

as the expression for computing a high-boost-filtered image.

Equation (10) is applicable in general and does not state explicitly how the sharp image is obtained. If we elect to use the Laplacian, then we know that $f_s(x, y)$ can be obtained using Eq. (5). In this case, Eq. (10) becomes

$$f_{hb} = \begin{cases} Af(x, y) - \nabla^2 f(x, y) & \text{if the center coefficient of the Laplacian mask is negative} \\ Af(x, y) + \nabla^2 f(x, y) & \text{if the center coefficient of the Laplacian mask is positive.} \end{cases} \quad \text{...(11)}$$

Use of First Derivatives for Enhancement—The Gradient

First derivatives in image processing are implemented using the magnitude of the gradient. For a function f(x, y), the gradient of *f* at coordinates (x, y) is defined as the two-dimensional column *vector*.

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad \ldots(12)$$

The magnitude of this vector is given by

$$\begin{aligned} \nabla f &= \text{mag}(\nabla \mathbf{f}) \\ &= \left[G_x^2 + G_y^2\right]^{1/2} \\ &= \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \end{aligned} \quad \ldots.(13)$$

The components of the gradient vector itself are linear operators, but the magnitude of this vector obviously is not because of the squaring and square root operations. On the other hand, the partial derivatives in Eq. (3.7-12) are not rotation invariant (isotropic), but the magnitude of the gradient vector is. Although it is not strictly correct, the magnitude of the gradient vector often is referred to as the *gradient*.

The computational burden of implementing Eq. (13) over an entire image is not trivial, and it is common practice to approximate the magnitude of the gradient by using absolute values instead of squares and square roots:

$$\nabla f \approx |G_x| + |G_y| \quad \ldots.(14)$$