Arithmetic and Logical Operations on Images (Image Algebra)

These operations are applied on pixel-by-pixel basis. So, to add two images together, we add the value at pixel (0 , 0) in image 1 to the value at pixel (0 , 0) in image 2 and store the result in a new image at pixel (0 , 0). Then we move to the next pixel and repeat the process, continuing until all pixels have been visited.

Clearly, this can work properly only if the two images have identical dimensions. If they do not, then combination is still possible, but a meaningful result can be obtained only in ***the area of overlap***. If our images have dimensions of w_1*h_1, and w_2*h_2 and we assume that their origins are aligned, then the new image will have dimensions w*h, where:

$$w = min\ (w_1, w_2)$$

$$h = min\ (h_1, h_2)$$

Addition and Averaging

If we add two 8-bit gray scale images, then pixels in the resulting image can have values in the range 0-510. We should therefore choose a 16-bit representation for the output image or divide every pixel value by two. If we do the later, then we are computing an average of the two images.

The main application of image averaging is ***noise removal***. Every image acquired by a real sensor is afflicted to some degree of random noise. However, the level of noise is represented in the image can be reduced, provided that the scene is static and unchanging, by the averaging of multiple observations of that scene. This works because the noisy distribution can be regarded as approximately symmetrical with a mean of zero. As a result, positive perturbations of a pixel's value by a given amount are just as likely as negative perturbations by the same amount, and there will be a tendency for the perturbations to cancel out when several noisy values are added.

Addition can also be used to ***combine the information of two images***, such as an image morphing, in motion pictures.

Algorithm 1: image addition

```
read input-image1 into in-array1;
read input-image2 into in- array2;
for i = 1    to no-of-rows do
for j=1 to no-of-columns do begin
out-array (i,j) = in-array1(i,j) + in-array2(i,j);
if ( out-array (i,j) > 255 ) then            out-array (i,j) = 255;
    end
write out-array to out-image;
```

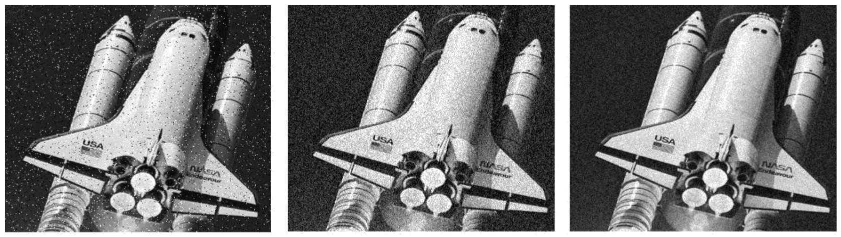


Figure (4) a) noisy image b) average of five observation c) average of ten observation

Subtraction

Subtracting two 8-bit grayscale images can produce values between - 225 and +225. This necessitates the use of 16-bit signed integers in the output image – unless sign is unimportant, in which case we can simply take the modulus of the result and store it using 8-bit integers:

$g(x,y) = |f_1 (x,y) - f_2 (x,y)|$

The main application for image subtraction is in ***change detection*** (or ***motion detection***). If we make two observations of a scene and compute their difference using the above equation, then changes will be indicated by pixels in the difference image which have ***non-zero values***. Sensor noise, slight changes in illumination and various other factors can result in small differences which are of no significance so it is usual to apply a threshold to the difference image. Differences below this threshold are set to zero. Difference above the threshold can, if desired, be set to the maximum pixel value. Subtraction can also be used in ***medical imaging to remove static background information.***

Algorithm2: image subtraction

```
read input-image1 into in-array1;
read input-image2 into in- array2;
for i = 1    to no-of-rows do
    for j=1 to no-of-columns do
       begin
          out-array (i,j) = in-array1(i,j) - in-array2(i,j);
          if ( out-array (i,j) < 0 ) then out-array (i,j) = 0;
      end
write out-array to out-image;
```

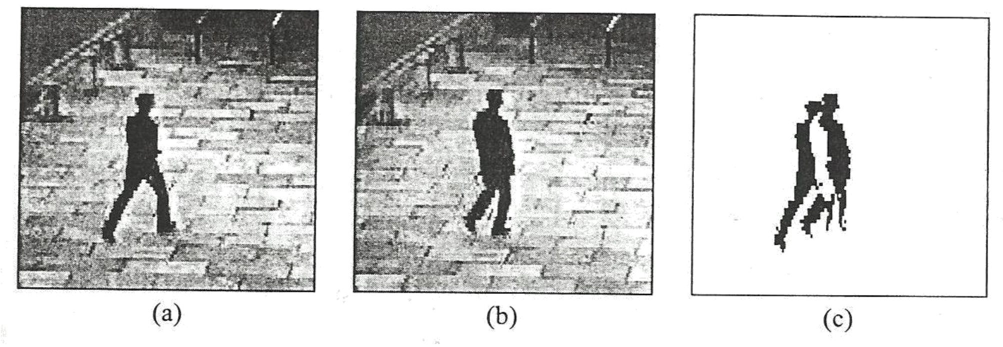


Figure (5) a, b) two frames of video sequence c) their difference

Multiplication and Division

Multiplication and division can be used to adjust brightness of an image. Multiplication of pixel values by a number greater than one will brighten the image, and division by a factor greater than one will darken the image. Brightness adjustment is often used as a ***preprocessing step*** in image enhancement.

One of the principle uses of image multiplication (or division) is to ***correct grey-level shading*** resulting from non uniformities in illumination orin the sensor used to acquire the image.

(a) *(b)* *(c)*

Figure a) original image b) image multiplied by 2 c) image divided by 2

15. Logical Operation:

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

	AND					OR					XOR			
Input 1	1	1	0	0		1	1	0	0		1	1	0	0
Input 2	1	0	1	0		1	0	1	0		1	0	1	0
output	1	0	0	0		1	1	1	0		0	1	1	0

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

On the other hand, if we compute the OR of a binary image with some other image , the pixels for which the corresponding value in the binary image is 0 will be ***preserved***, but pixels for which the corresponding binary value is 1, will be set to 1 (cleared).

So, masking is a simple method to extract a region of interest from an image.

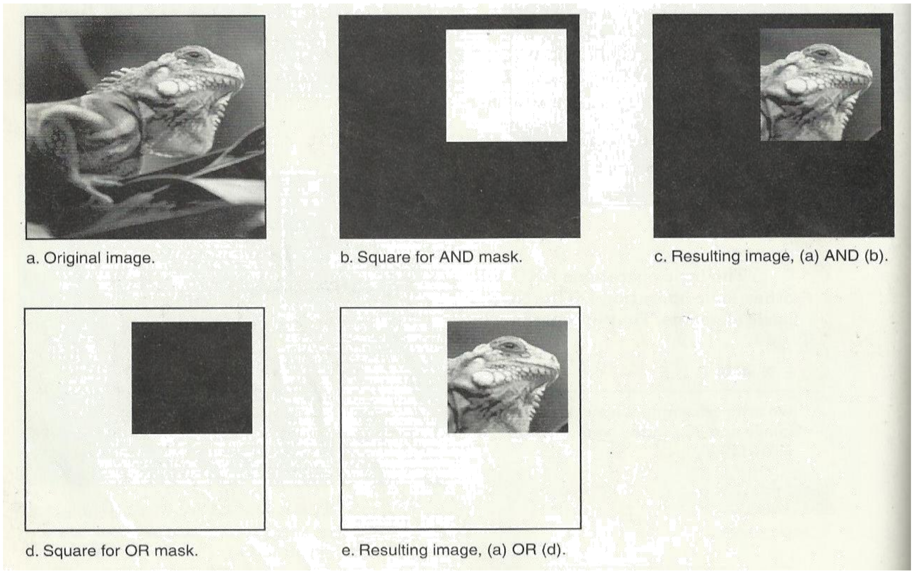

a. Original image.
b. Square for AND mask.
c. Resulting image, (a) AND (b).
d. Square for OR mask.
e. Resulting image, (a) OR (d).

Figure: image masking

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

AND^

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

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

Exclusive OR $\otimes$

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

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

NOT

NOT operation can be performed on gray-level images, it's applied on only one image, and the result of this operation is the ***negative*** of the original image.

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

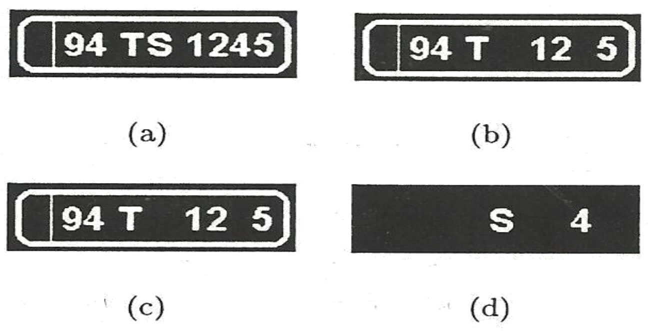


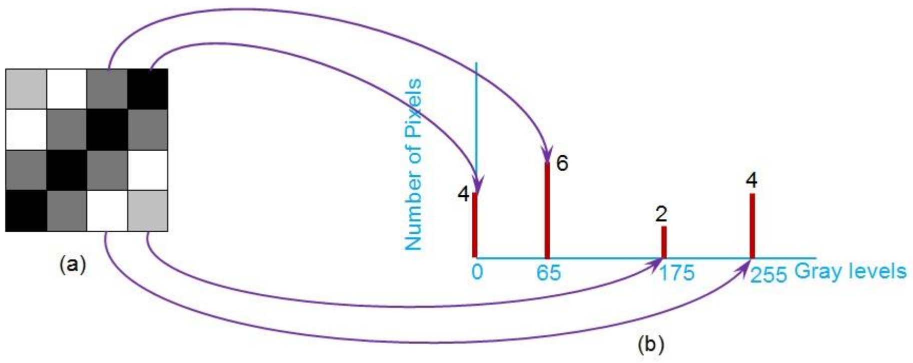
Figure a) input image a(x,y); b) input image b(x,y) ; c) a(x,y) ^ b(x,y) ; d) a(x,y) ^ ~ b(x,y)

16. Image Histogram

A histogram is a graph that shows the frequency of anything.

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable). A histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. It's a bar chart of the count of pixels of every tone of gray that occurs in the image.

For an 8-bit grayscale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those grayscale values.



The gray level ***histogram*** is showing, the gray level, for each pixel in the image.

The histogram of an image records the frequency distribution of gray levels

The histogram of an 8-bit image, can be though of as a table with 256 entries, or " bins", indexed from 0 to 255. in bin 0 we record the number of times a gray level of 0 occurs; in bin 1 we record the number of times a gray level of 1 occurs, and so on, up to bin 255.

An algorithm below shows how we can accumulate in a histogram from an image.

ALGORITHM: for Calculating image Histogram

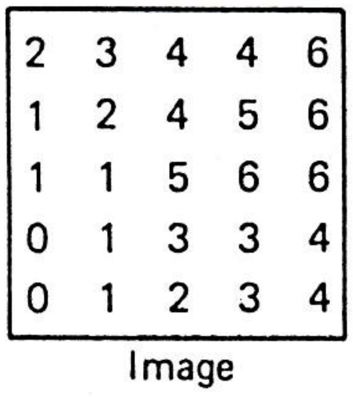
```
create an array histogram.
For all gray levels ,I,do
Histogram [I] =0
Endfor
For all pixels coordinates, x and y , do
```

- The histogram of a digital image with *L* total possible intensity levels in the range [*0, G*] is defined as the discrete function

$$h(r_k) = n_k$$

- Where r_k is the *k*th intensity level in the interval [*0,G*] and n_k is the number of pixels in the image whose intensity level is r_k.

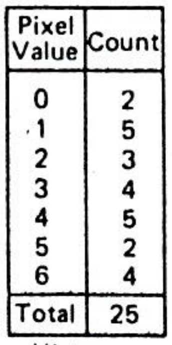
Example: Figure shows an image and its histogram.



2	3	4	4	6
1	2	4	5	6
1	1	5	6	6
0	1	3	3	4
0	1	2	3	4

Image

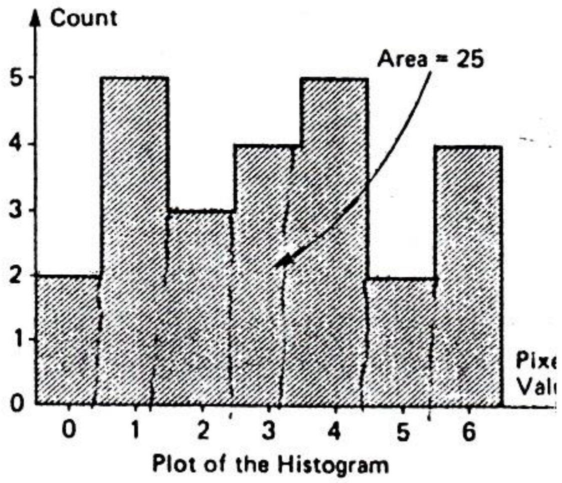
(a)



Pixel Value	Count
0	2
1	5
2	3
3	4
4	5
5	2
6	4
Total	25

Histogram

(b)



(c)

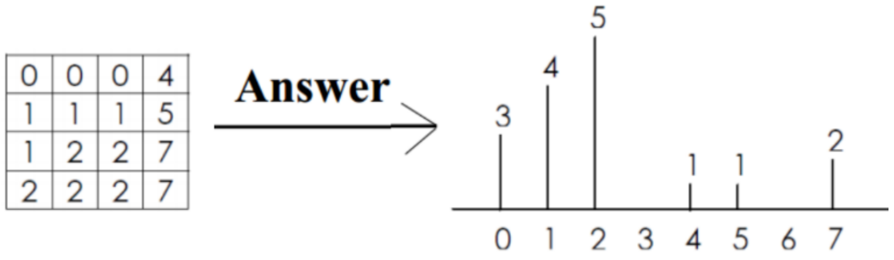
Figure: sub image and its histogram

The shape of the histogram provides us with information about the nature of the image, or sub image if we are considering an object in the image. For example, a ***very narrow*** histogram implies a low contrast, a histogram ***skewed toward the right*** implies a bright image, a histogram ***skewed toward the left*** implies a dark image, and a histogram with ***two major peaks***, implies an object that in contrast with the background.

A color histogram counts pixels with a given pixel value in red, green, and blue (RGB). For example, in pseudocode, for images with 8-bit values in each of R, G, B, we can fill a histogram that has 256^3 bins:

```
inthist[256][256][256]; // reset to 0

//image is an appropriate struct

//with byte fields red,green,blue

fori=0..(MAX_Y-1)

for j=0..(MAX_X-1)

{

R = image[i][j].red;

G = image[i][j].green;
```

Example : Plot the Histogram of the following example with 4x4 matrix of a 3-bit image.



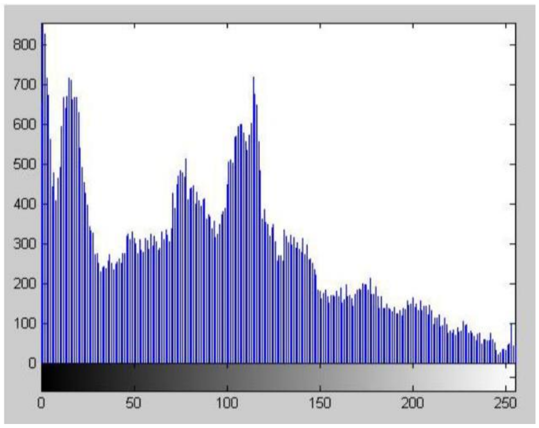
Properties and usage of histogram

One of the ***principle use*** of the histogram is in the selection of ***threshold*** parameter.

The histogram of an image provides a useful indication of the relative importance of different gray levels in an image, indeed, it is sometimes possible to ***determine*** whether ***brightnes***s or ***contras***t ***adjustment*** is necessary merely by ***examining*** the ***histogram*** and ***not the image*** itself.

When an image is condensed into a histogram, ***all spatialinformation is discarded.*** The histogram specifies the number of pixelshaving each gray level but ***gives no hint*** as to ***where those pixels arelocated*** within the image. Thus the histogram is ***unique*** for any particularimage, but the ***reverse is not true***. Vastly different images could have identical histograms. Such operations as moving objects around within an image typically have no effect on the histogram.

Histograms are used in numerous image processing techniques, such as image enhancement, compression and segmentation.



Note that the horizontal axis of the histogram plot (Figure (b) above) represents gray level values, k, from 0 to 255. The vertical axis represents the values of $h(k)$ i.e. the number of pixels which have the gray level k.

It is customary to **“normalize”** a histogram by dividing each of its values by the total number of pixels in the image, i.e. uses the probability distribution (previously stated) as:

$$p(k) = \frac{h(k)}{M \times N}$$

Thus, $p(k)$ represents the probability of occurrence of gray level k.

As with any probability distribution:

- ***All the values of a normalized histogram $p(k)$ are less than or equal to 1.***
- ***The sum of all $p(k)$ values is equal to 1***

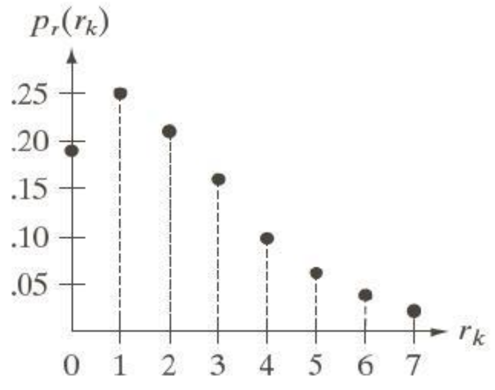
Another way of getting histogram is to plot pixel intensities vs. pixel probabilities. However, probability histogram should be used when comparing the histograms of images with different sizes.

Example: Suppose that a 3-bit image (L = 8) of size 64 × 64 pixels has the gray level (intensity) distribution shown in the table below.***Perform normalized histogram.***

r_k	n_k
$r_0 = 0$	790
$r_1 = 1$	1023
$r_2 = 2$	850
$r_3 = 3$	656
$r_4 = 4$	329
$r_5 = 5$	245
$r_6 = 6$	122
$r_7 = 7$	81

Solution:*M × N = 4096 .* We compute the normalized histogram:

r_k	n_k	$p(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02



Normalized histogram