Pro.Dr. Azmi Tawfiq. Hussain Al-Rawi,
University of Anbar of Commuter science & I.T.
Department of Computer science \4th stage
CVIP

Point Processing

image processing operations may be divided into three classes based on the information required to perform the transformation.

1. **Transforms**: We require a knowledge of all the grey levels in the entire image to transform the image. In other words, the entire image is processed as a single large block. This may be illustrated by the diagram shown in figure (1)
2. **Spatial filters**: To change the grey level of a given pixel we need only know the value of the grey levels in a small neighborhood of pixels around the given pixel.
3. **Point operations**: A pixel's grey value is changed without any knowledge of its surrounds.

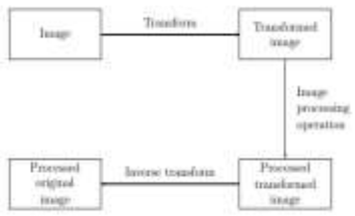


Figure (1): Schema for transform processing.

Arithmetic operations:
These operations act by applying a simple function
Y=f(x)
to each grey value in the image. Thus f(x) is a function which maps the range 0…….255 on to itself.
Simple functions include adding or subtract a constant value to each pixel:

1-Linear gray scale transformation:

Y=x+c or y=x-c
" or multiplying each pixel by a constant:
Y=cx
the values by setting:
y=255 if y>255, y=0 if y<0

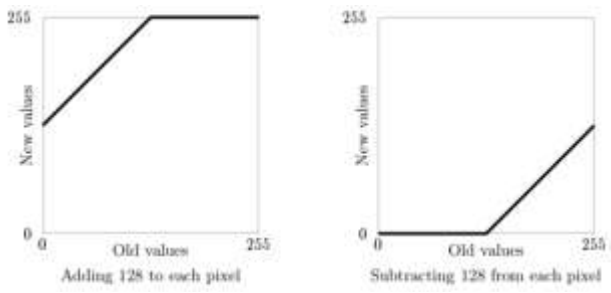


Figure (2): Adding and subtracting constant.

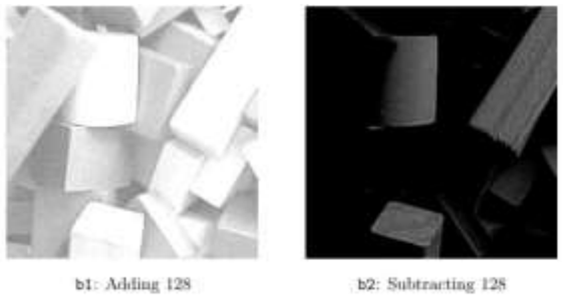
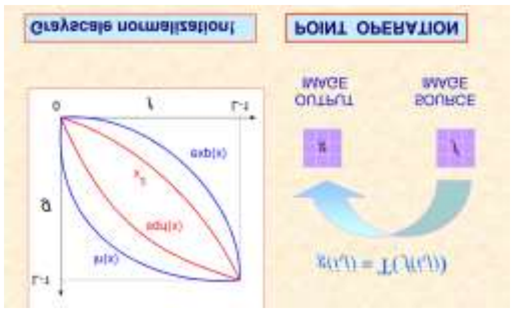
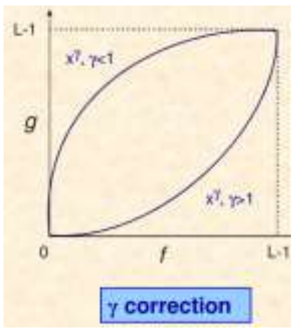


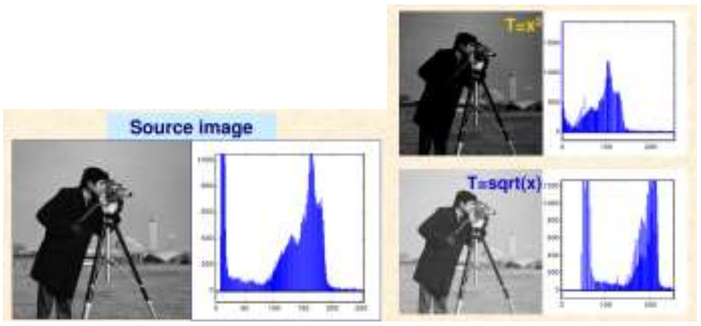
Figure 3: Arithmetic operations on an image: adding or subtracting a constant.

2-Non Linear gray scale transformation:





Examples



Histograms:

Given a greyscale image, its histogram consists of the histogram of its grey levels; that is, a graph indicating the number of times each grey level occurs in the image.

- In a dark image, the grey levels (and hence the histogram) would be clustered at the lower end:

- In a uniformly bright image, the grey levels would be clustered at the upper end:

- In a well contrasted image, the grey levels would be well spread out over much of the range.

There are **two** ways to enhance its contrast, by spreading out its histogram:

1- Histogram stretching (Contrast stretching):

The function for Histogram stretching can be found by the following equation:

$$\boldsymbol{Stretch}\ (\boldsymbol{I}(\boldsymbol{r},\boldsymbol{c})) = \frac{\boldsymbol{I}(\boldsymbol{r},\boldsymbol{c}) - \boldsymbol{I}(\boldsymbol{r},\boldsymbol{c})_{\boldsymbol{min}}}{(\boldsymbol{I}(\boldsymbol{r},\boldsymbol{c})_{\boldsymbol{max}} - \boldsymbol{I}(\boldsymbol{r},\boldsymbol{c})_{\boldsymbol{min}})}(\boldsymbol{L}_{\boldsymbol{max}} - \boldsymbol{L}_{\boldsymbol{min}}) + \boldsymbol{L}_{\boldsymbol{min}}$$

Where $I(r,c)_{max}$ is the largest gray-level in the image I(r,c).
$I(r,c)_{min}$ is the smallest gray-level in the image I(r,c).
L_{max} and L_{min} corresponding to the maximum and minimum gray-level values possible for an 8-bit image these are 255 and 0

Example: Suppose an image on 4-bits gray level with size 3x3. Apply contrast stretching.

12	1	14
4	5	8
3	2	9

Solution:

$$\boldsymbol{Stretch\ (I(r,c)) = \frac{I(r,c) - I(r,c)_{min}}{(I(r,c)_{max} - I(r,c)_{min})}(L_{max} - L_{min}) + L_{min}}$$

$I(r,c)_{min}$ =1, $I(r,c)_{max}$=14 , L_{min}=0 ,L_{max}=15

$$\boldsymbol{Stretch\ (I(0,0,)) = \frac{12-1}{14-1}(15-0)+0 = 13}$$

$$\boldsymbol{Stretch\ (I(0,1,)) = \frac{1-1}{14-1}(15-0)+0 = 0}$$

$$\boldsymbol{Stretch\ (I(0,2,)) = \frac{14-1}{14-1}(15-0)+0 = 15}$$

$$\boldsymbol{Stretch\ (I(1,0,)) = \frac{4-1}{14-1}(15-0)+0 = 3}$$

$$\boldsymbol{Stretch\ (I(1,1,)) = \frac{5-1}{14-1}(15-0)+0 = 5}$$

$$\boldsymbol{Stretch\ (I(1,2)) = \frac{8-1}{14-1}(15-0)+0 = 8}$$

$$\boldsymbol{Stretch\ (I(2,0,)) = \frac{3-1}{14-1}(15-0)+0 = 2}$$

$$\boldsymbol{Stretch\ (I(2,1)) = \frac{2-1}{14-1}(15-0)+0 = 1}$$

$$\boldsymbol{Stretch\ (I(2,2)) = \frac{9-1}{14-1}(15-0)+0 = 9}$$

After processing:

12	1	14
4	5	8
3	2	9

13	0	15
3	5	8
2	1	9

Example 2: Suppose an image on 8-bits gray level. Find stretching histogram.

G. L	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fre.	15	0	0	0	0	70	110	45	70	35	0	0	0	0	0	15

Sol.

$$\boldsymbol{Stretch\ (I(r,c)) = \frac{I(r,c) - I(r,c)_{min}}{(I(r,c)_{max} - I(r,c)_{min})}(L_{max} - L_{min}) + L_{min}}$$

$I(r,c)_{min}$ =0, $I(r,c)_{max}$=15 , L_{min}=0 ,L_{max}=255

$$\boldsymbol{Stretch\ (0) = \frac{0-0}{15-0}(255-0)+0 = 0}$$

$$\boldsymbol{Stretch\ (5) = \frac{5-0}{15-0}(255-0)+0 = 85}$$

$$\boldsymbol{Stretch\ (6) = \frac{6-0}{15-0}(255-0)+0 = 102}$$

$$\boldsymbol{Stretch\ (7) = \frac{7-0}{15-0}(255-0)+0 = 119}$$

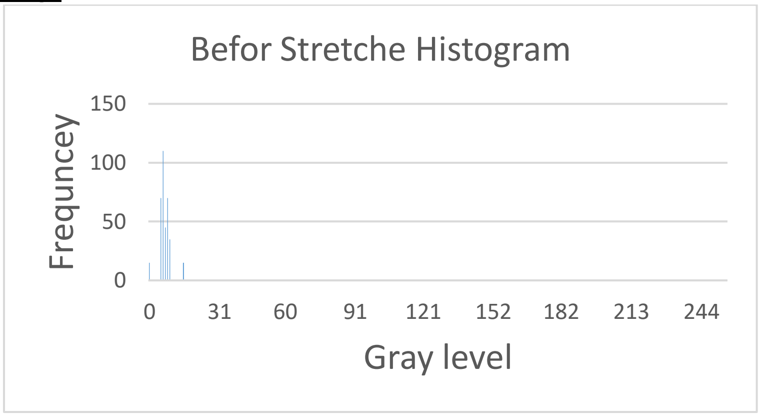
$$\boldsymbol{Stretch\ (8) = \frac{8-0}{15-0}(255-0)+0 = 136}$$

$$\boldsymbol{Stretch\ (9) = \frac{9-0}{15-0}(255-0)+0 = 153}$$

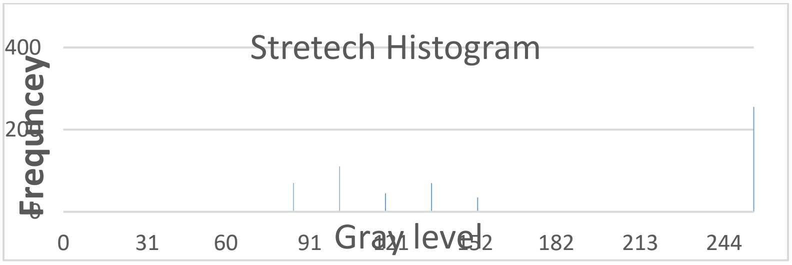
$$\boldsymbol{Stretch\ (15) = \frac{15-0}{15-0}(255-0)+0 = 255}$$

i	**0**	**5**	**6**	**7**	**8**	**9**	**15**
j	**0**	**85**	**102**	**119**	**136**	**153**	**255**

<u>Before Processing:</u>



<u>After processing:</u>



2-Histogram Equalization:

Sometimes a better approach is provided by histogram equalization. To transform the grey levels to obtain a better contrast image, we change gray level I to

$$1/n \quad (n_0+ n_1+ n_2+ n_3+\ldots\ldots\ldots\ldots+ n_i)(L-1)$$

And this number is rounded to the nearest integer.

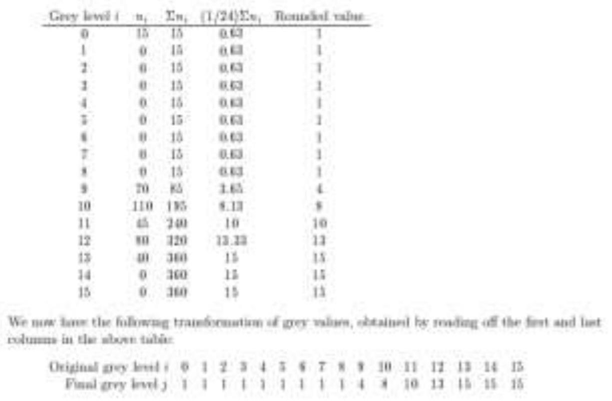
Example

Suppose our images 4-bits.

Gray level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_i	15	0	0	0	0	0	0	0	0	70	110	45	80	40	0	0

n_i=15+0+0++0+0+0+0……………..+80+40+0+0=360
L-1=15

To equalize this histogram, we form running totals of the n_i , 15/360=1/24



Grey level i	n_i	Σn_i	$(1/24)\Sigma n_i$	Rounded value
0	15	15	0.63	1
1	0	15	0.63	1
2	0	15	0.63	1
3	0	15	0.63	1
4	0	15	0.63	1
5	0	15	0.63	1
6	0	15	0.63	1
7	0	15	0.63	1
8	0	15	0.63	1
9	70	85	3.54	4
10	110	195	8.13	8
11	45	240	10	10
12	80	320	13.33	13
13	40	360	15	15
14	0	360	15	15
15	0	360	15	15

We now have the following transformation of grey values, obtained by reading off the first and last columns in the above table:

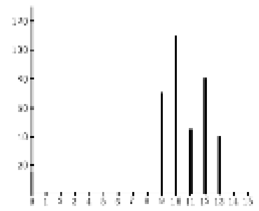
Original grey level i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Final grey level j	1	1	1	1	1	1	1	1	1	4	8	10	13	15	15	15

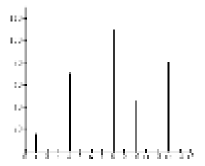
<u>Before</u>

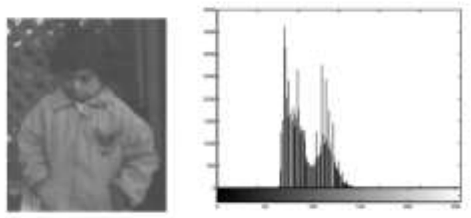
Gray level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_i	15	0	0	0	0	0	0	0	0	70	110	45	80	40	0	0

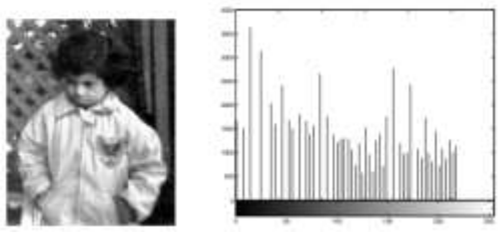
After

Gray level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_i	0	15	0	0	70	0	0	0	110	0	45	0	0	80	0	0


Before Histogram Equalization


After Histogram Equalization


Before Histogram Equalization



After Histogram Equalization

Thresholding :

1 -Single thresholding: A greyscale image is turned into a binary (black and white) image by first choosing a grey level in the original image, and then turning every pixel black or white according to whether its grey value is greater than or less than T:
A pixels become becomes white if its gray level > T.
A pixels become becomes black if its gray level <=T
Thresholding is a vital part of image segmentation, where we wish to isolate objects from the background. It is also an important component of robot vision.

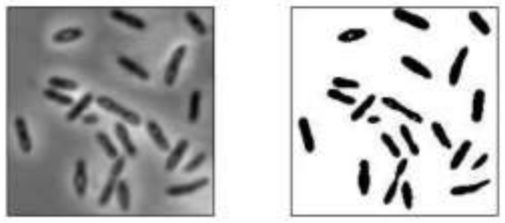


Figure (5): Thresholded image of bacteria at T>100

Thresholding provides a very simple way of showing hidden aspects of an image. For example, the image paper as shown in figure (6):

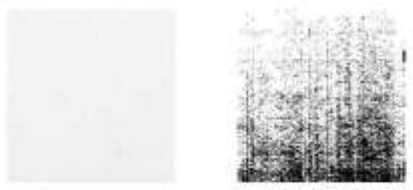


Figure (6): The paper image and result after thresholding.

2- Double thresholding:
Here we choose two values and T_1 and T_2 apply a thresholding operation.
A pixels become becomes white its gray level between T_1 and T_2.
A pixels become becomes black if its gray level is otherwise.

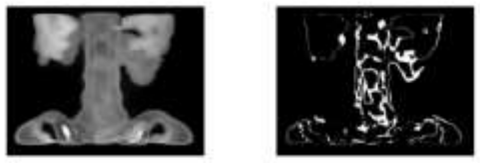


Figure (7): The image spins the result after double thresholding.

Applications of thresholding:

1-To remove unnecessary detail from an image, to concentrate on essentials. But this information may be all we need to investigate sizes, shapes, or numbers of blobs.
2-To bring out hidden detail. This was illustrated with paper and spine images. In both, the detail was obscured because of the similarity of the grey levels involved.
3-When we want to remove a varying background from text or a drawing. We can simulate a varying background by taking the image text as shown figure (8).

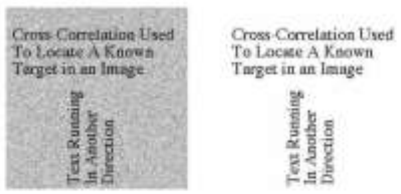
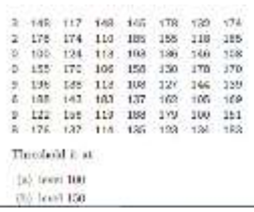


Figure (8): Text on a varying background, and threshold

Exercises :

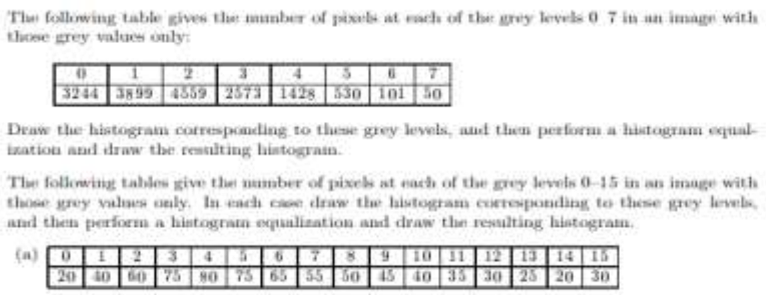
1- Consider the following 8x8 image.



3	148	117	148	145	178	132	174
2	176	174	110	185	155	118	165
0	100	124	113	193	136	146	108
0	155	170	106	158	130	178	170
9	196	138	113	108	127	144	139
6	188	143	183	137	162	105	169
9	122	156	119	188	179	100	151
8	176	137	114	135	123	134	183

Threshold it at

(a) level 100
(b) level 150



The following table gives the number of pixels at each of the grey levels 0–7 in an image with those grey values only:

0	1	2	3	4	5	6	7
3244	3899	4559	2573	1428	530	101	50

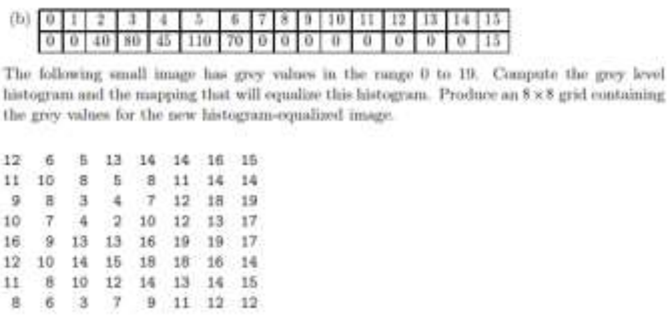
Draw the histogram corresponding to these grey levels, and then perform a histogram equalization and draw the resulting histogram.

The following tables give the number of pixels at each of the grey levels 0–15 in an image with those grey values only. In each case draw the histogram corresponding to these grey levels, and then perform a histogram equalization and draw the resulting histogram.

(a)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20	40	60	75	80	75	65	55	50	45	40	35	30	25	20	30

2,3



(b)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	40	80	45	110	70	0	0	0	0	0	0	0	0	15

9. The following small image has grey values in the range 0 to 19. Compute the grey level histogram and the mapping that will equalize this histogram. Produce an 8 × 8 grid containing the grey values for the new histogram-equalized image.

12	6	5	13	14	14	16	15
11	10	8	5	8	11	14	14
9	8	3	4	7	12	18	19
10	7	4	2	10	12	13	17
16	9	13	13	16	19	19	17
12	10	14	15	18	18	16	14
11	8	10	12	14	13	14	15
8	6	3	7	9	11	12	12

Noise in Images

We may define noise to be any degradation in the image signal, caused by external disturbance. The errors will appear on the image output in different ways depending on the type of disturbance in the signal. Cleaning an image corrupted by noise is thus an important area of image restoration.

Types of noise:

There are four different noise types:

1-Salt and pepper noise:

Also called impulse noise, shot noise, or binary noise. This degradation can be caused by sharp, sudden disturbances in the image signal; its appearance is randomly scattered white or black (or both) pixels over the image. As shown in figure (1)

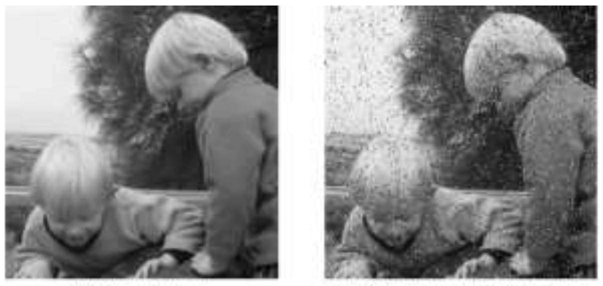


Figure (1) Noise on an image

2 -Gaussian noise:

Gaussian noise is an idealized form of white noise, which is caused by random fluctuations in the signal (white noise by watching a television) is normally distributed. we can model a noisy image by simply adding the image I and noise N: I+N

Assume that I is a matrix whose elements are the pixel values of our image, and N is a matrix whose elements are normally distributed.

3- Speckle Noise:

Whereas Gaussian noise can be modelled by random values added to an image; speckle noise (or more simply just speckle) can be modelled by random values multiplied by pixel values, hence it is also called multiplicative noise. Speckle noise is a major problem in some radar applications. As shown in figure (2)

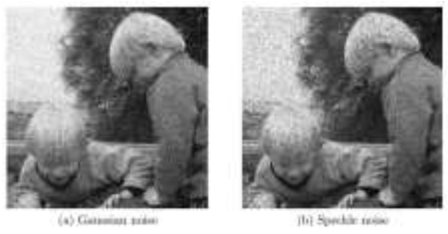


Figure (2): The twins image corrupted by Gaussian and speckle noise.

Although Gaussian noise and speckle noise appear superficially similar, they are formed by two totally different methods.

4- Periodic noise:

If the image signal is subject to a periodic, rather than a random disturbance, we might obtain an image corrupted by periodic noise. The effect is of bars over the image. As shown in figure (3)

Salt and pepper noise, Gaussian noise and speckle noise can all be cleaned by using spatial filtering techniques. Periodic noise, however, requires image transforms for best results.

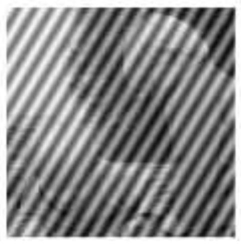
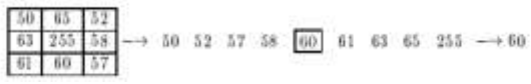


Figure (3): The twins image corrupted by periodic noise.

Cleaning salt and pepper noise:

1- **Low pass filtering:** Given that pixels corrupted by salt and pepper noise are high frequency components of an image, we should expect a low-pass filter should reduce them. As shown in figure (4).

2-Median filtering: Median filtering seems almost tailor-made for removal of salt and pepper noise. A median filter is an example of a non-linear spatial filter. The result is shown in figure (5). The result is a vast improvement on using averaging filters. For example.



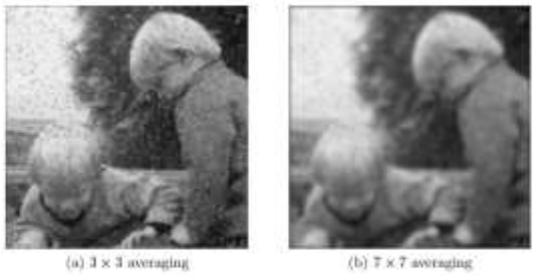


Figure (4): Attempting to clean salt & pepper noise with average filtering.

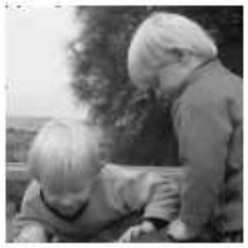
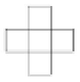


Figure (5): Cleaning salt and pepper noise with a median filter.

3-Rank-order filter: Median filtering is a special case of a more general process called rank-order filtering. Rather than take the median of a set, we order the set and take the nth value, for some predetermined value of n. Thus median filtering using a 3x3 mask is equivalent to rank-order filtering with n=5. Similarly, median filtering using a 5x5 mask is equivalent to rank-order filtering with n=13. t2. There is only one reason for using rank-order filtering instead of median filtering, and that is that it allows us to choose the median of non-rectangular masks. For example, if we decided to use as a mask a 3x3 cross shape.



4-An outlier method:

Applying the median filter can in general be a slow operation: each pixel requires the sorting of at least nine values. To overcome this difficulty. We use of cleaning salt and pepper noise by treating noisy pixels as **outliers**. The following approach for noise cleaning:

1. Choose a threshold value D.

2. For a given pixel, compare its value p with the mean m of the values of its eight neighbors.

3. If |p-m|>D then classify the pixel as noisy, otherwise not.

4. If the pixel is noisy, replace its value with mi; otherwise leave its value unchanged.

Cleaning Gaussian noise:

Image averaging: It may sometimes happen that instead of just one image corrupted with Gaussian noise, we have many different copies of it. An example is satellite imaging; if a satellite passes over the same spot many times, we will obtain many different images of the same place. And other example is microscopy with different images.

suppose we have 100 copies of our image, each with noise; then the i -th noisy image well be:

$M+N_i$

where M is the matrix of original values, and N_i is a matrix of normally distributed random values with mean 0. We can find the mean $M^{'}$ of these images by the usual add and divide method.

$$\begin{aligned} M' &= \frac{1}{100}\sum_{i=1}^{100}(M+N_i) \\ &= \frac{1}{100}\sum_{i=1}^{100}M + \frac{1}{100}\sum_{i=1}^{100}N_i \\ &= M + \frac{1}{100}\sum_{i=1}^{100}N_i \end{aligned}$$

Since N_i is normally distributed with mean 0, it can be readily shown that the mean of all the N_i 's will be close to zero—the greater the number of N_i's; the closer to zero. Thus

$M' \approx M$

and the approximation is closer for larger number of images $M+N_i$ see fig.6

We can demonstrate this with the twins image. We first need to create different versions with Gaussian noise, and then take the average of them.

Average filtering: If the Gaussian noise has mean 0, then we would expect that an average filter would average the noise to 0. The larger the size of the filter mask, the closer to zero.

(a) 10 images (b) 100 images

Figure (6): Image averaging to remove Gaussian noise.

Exercises

1. The arrays below represent small greyscale images. Compute the 4×4 image that would result in each case if the middle 16 pixels were transformed using a 3×3 median filter:

8	17	4	10	15	12
10	12	15	7	3	10
15	10	50	5	3	12
4	8	11	4	1	8
16	7	4	3	0	7
16	24	19	3	20	10

1	1	2	5	3	1
3	20	5	6	4	6
4	6	4	20	2	2
4	3	3	5	1	5
6	5	20	2	20	2
6	3	1	4	1	2

7	8	11	12	13	9
8	14	0	9	7	10
11	23	10	14	1	8
14	7	11	8	9	11
13	13	18	10	7	12
9	11	14	12	13	10

2. Using the same images as in question 1, transform them by using a 3×3 averaging filter.

3. Use the outlier method to find noisy pixels in each of the images given in question 1. What are the reasonable values to use for the difference between the grey value of a pixel and the average of its eight 8-neighbours?

Edge of images

Edges contain some of the most useful information in an image. We may use edges to measure the size of objects in an image; to isolate particular objects from their background; to recognize or classify objects. There are a large number of edge-finding algorithms in this field.

An edge may be loosely defined as a line of pixels showing an observable difference. For example, consider the two blocks of pixels shown in figure 1.

51	52	53	59
54	52	53	62
50	52	53	68
55	52	53	55

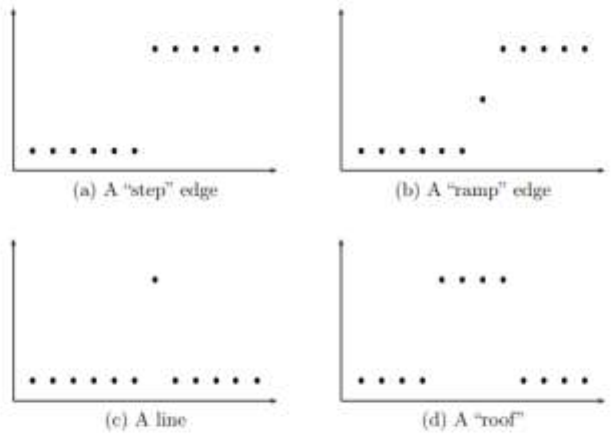
50	53	150	160
51	53	150	170
52	53	151	190
51	53	152	155

Figure 1: Blocks of pixels.

In the right hand block, there is a clear difference between the grey values in the second and third columns.

Differences and edges:

If we consider the grey values along this line, and plot their values, we will have something similar to that shown in figure 2 (a) or (b). If we now plot the differences between each grey value and its predecessor from the ramp edge, we would obtain a graph similar to that shown in figure 3



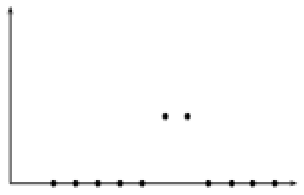


Figure 3: Differences of the edge function

To see where this graph comes from, suppose that the values of the "ramp" edge in figure 2(b) are, from left to right:

30, 30, 30, 30, 30, 30, 100, 180, 180, 180, 180, 180

If we form the differences, by subtracting each value from its successor, we obtain:

0, 0, 0, 0, 0, 80, 80, 0, 0, 0, 0

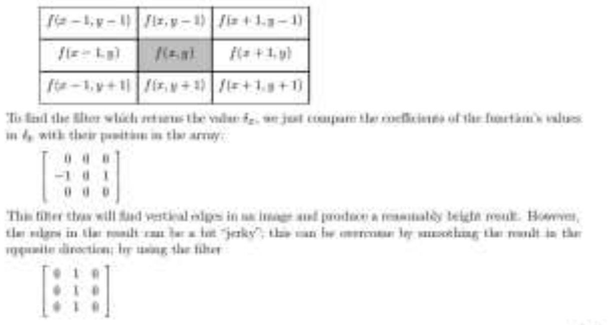
and it is these values which are plotted in figure 3. It appears that the difference tends to enhance edges, and reduce other components. So if we could "difference" the image, we would obtain the effect we want. We can define the difference in three separate ways:

- the *forward difference*: $\Delta f(x) = f(x+1) - f(x)$,
- the *backward difference*: $\nabla f(x) = f(x) - f(x-1)$,
- the *central difference*: $\delta f(x) = f(x+1) - f(x-1)$.

However, an image is a function of two variables, so we can generalize these definitions to include both the x and y values:

$$\begin{aligned} \Delta_x f(x,y) &= f(x+1,y) - f(x,y) & \Delta_y f(x,y) &= f(x,y+1) - f(x,y) \\ \nabla_x f(x,y) &= f(x,y) - f(x-1,y) & \nabla_y f(x,y) &= f(x,y) - f(x,y-1) \\ \delta_x f(x,y) &= f(x+1,y) - f(x-1,y) & \delta_y f(x,y) &= f(x,y+1) - f(x,y-1) \end{aligned}$$

Some difference filters: To see how we might use δx to determine edges in the x direction, consider the function values around a point p(x,y).



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

To find the filter which returns the value δ_x, we just compare the coefficients of the function's values in δ_x with their position in the array:

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

This filter thus will find vertical edges in an image and produce a reasonably bright result. However, the edges in the result can be a bit "jerky"; this can be overcome by smoothing the result in the opposite direction; by using the filter

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

Both filters can be applied at once, using the combined filter:

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

This filter, and its companion for finding horizontal edges:

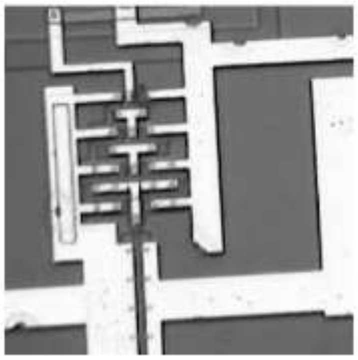
$$P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

This Prewitt filters for edge detection.

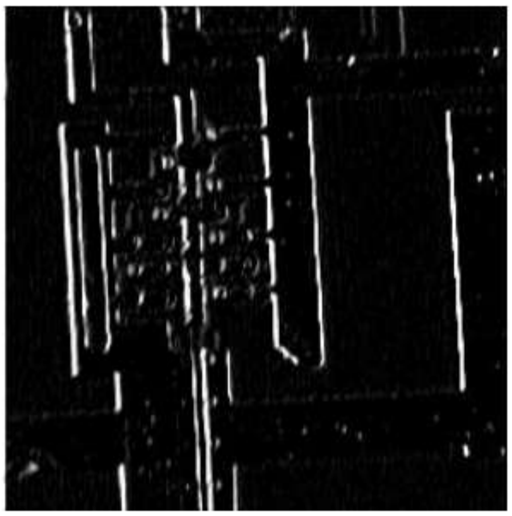
So if P_x and were the grey values produced by applying P_y and to an image, then the output grey value can be chosen by any of these methods

1. $v = \max\{|p_x|, |p_y|\}$,
2. $v = |p_x| + |p_y|$,
3. $v = \sqrt{p_x^2 + p_y^2}$.

Applying each of P_x and P_y individually provides the results shown in below figures

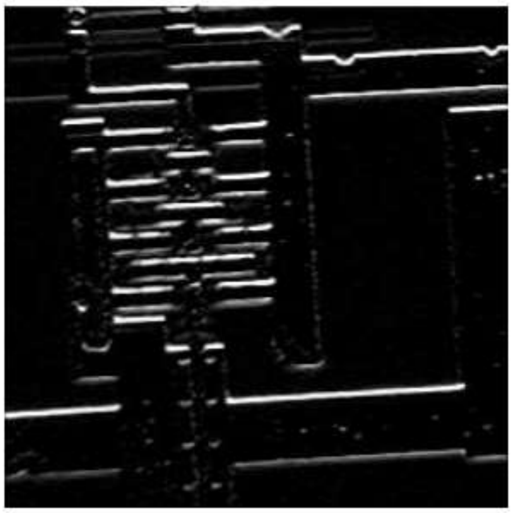


Original image integrated circuit



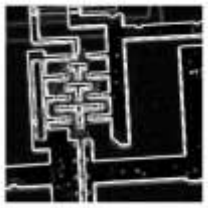
(a)

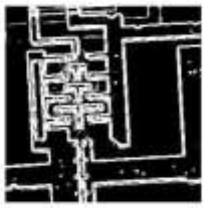
P_x

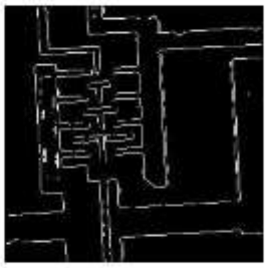


(b)

P_y


(a)


(b)



The prewitt option of edge after convert to binary image

Slightly different edge finding filters are the *Roberts cross-gradient filters*:

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

and the *Sobel filters*:

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

The *Sobel filters* are similar to the Prewitt filters, in that they apply a smoothing filter in the opposite direction to the central difference filter. In the Sobel filters, the smoothing takes the form

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

takes the form which gives slightly more prominence to the central pixel.

Fourier Transform

Fourier transform the signals between two different domains, such as transforming signal from frequency domain to time domain or vice versa. Fourier transform has many applications in Engineering and Physics, such as signal processing, RADAR, and so on.

The Fourier transform is of fundamental importance to image processing. It allows us to perform tasks which would be impossible to perform any other way; its efficiency allows us to perform other tasks more quickly. The Fourier is a powerful alternative to linear spatial filtering; it is more efficient to use the Fourier transform than a spatial filter for a large filter.

Definition of the one dimensional discrete Fourier transform (DFT):

Suppose

$$\mathbf{f}=[f_0, f_1, f_2, \ldots\ldots f_{N-1}] \ldots\ldots\ldots\ldots\ldots\ldots(1)$$

is a sequence of length N. We define its discrete Fourier transform to be the sequence

$$\mathbf{F}=[F_0, F_1, F_2, \ldots\ldots, F_{N-1}] \ldots\ldots\ldots\ldots\ldots\ldots(2)$$

where

$$F_u = \frac{1}{N}\sum_{x=0}^{N-1} \exp\left[-2\pi i \frac{xu}{N}\right] f_x \ldots\ldots\ldots\ldots\ldots(3)$$

The inverse DFT:

The formula for the inverse DFT is very similar to the forward transform

$$x_u = \frac{1}{N}\sum_{x=0}^{N-1} \exp\left[2\pi i \frac{xu}{N}\right] F_u \ldots\ldots\ldots\ldots\ldots\ldots(4)$$

Properties of the one-dimensional DFT:

Linearity: Suppose f and g are two vectors of equal length, p and q are scalars, with h=pf+qg . If F, G and H are the DFT's of f,g and h respectively, we

$$H=PF+qG \ldots\ldots\ldots\ldots\ldots\ldots(5)$$

This follows from the definition

F=ƒf ,G=ƒg , H=ƒG ……….(6)

Shifting: we change the sign of every second element x by $(-1)^n$. Let the resulting vector be denoted $x^/$. The DFT $X^/$ of $x^/$ is equal to the DFT X of x DFT with the swapping of the left and right halves.

The Fast Fourier Transform(FFT):

There are a number of extremely fast and efficient algorithms for computing a DFT; such an algorithm is called a fast Fourier transform, or FFT. The use of an FFT vastly reduces the time needed to compute a DFT. One FFT method works recursively by dividing the original vector into two halves, computing the FFT of each half.

For a vector of length 2^n, the direct method takes $(2^n)^2=2^{2n}$ multiplications; the FFT only $n2^n$. The saving in time is thus of an order of $2^n/n$. Clearly the advantage of using an FFT algorithm becomes greater as the size of the vector increases.

Because of this computational advantage, any implementation of the DFT will use an FFT algorithm:

Table 1: Comparison of FFT and direct arithmetic.

n	**2^n**	**Direct arithmetic (2^{2n})**	**FFT ($n2^n$)**	**Increase in speed =$(2^{2n})/(n2^n)$**
2	4	16	8	2.0
3	8	64	24	2.67
4	16	256	64	4.0
5	32	1024	160	6.4
6	64	4096	384	10.67
7	128	16384	896	18.3
8	256	65536	2048	32.0
9	512	262144	4608	56.9
10	1024	1048576	10240	102.4

The two-dimensional DFT in two dimensions:

the DFT takes a matrix as input, and returns another matrix, of the same size, as output. If f(x,y) the original matrix values are where x and y are the indices, then the output matrix values are F(u,v)

The forward and inverse transforms for an MxN matrix, where for notational convenience we assume that the x indices are from 0 to M-1 and the y indices are from to 0 to N-1, are

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\exp[-2\pi i(\frac{xu}{M}+\frac{yv}{N})]……(7)$$

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v)\exp[2\pi i(\frac{xu}{M}+\frac{yv}{N})] \ldots\ldots\ldots(8)$$

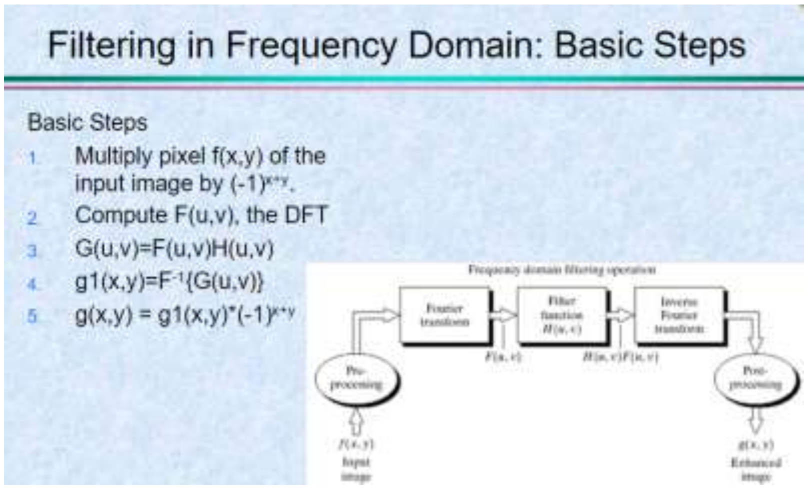
DFT a spatial filter:

Note that the values

$$\exp[\pm 2\pi i(\frac{xu}{M}+\frac{yv}{N})] \ldots\ldots\ldots\ldots(9)$$

are independent of the values *f* or F. as above. It also means that every value is obtained by multiplying every value of *f(x,y)* by a fixed value, and adding up all the results.

What a linear spatial filter does? it multiplies all elements under a mask with fixed value and adds them all up. Thus we can consider the DFT as a linear spatial filter which is as big as the image.



Filtering in Frequency Domain: Basic Steps

Basic Steps

1. Multiply pixel f(x,y) of the input image by $(-1)^{x+y}$.
2. Compute F(u,v), the DFT
3. G(u,v)=F(u,v)H(u,v)
4. $g1(x,y)=F^{-1}\{G(u,v)\}$
5. $g(x,y) = g1(x,y)*(-1)^{x+y}$

Some properties of the two dimensional Fourier transform:

All the properties of the one-dimensional DFT transfer into two dimensions, but there are some further properties as:

1-Similarity: First notice that the forward and inverse transforms are very similar, with the exception of the scale factor 1/MN in the inverse transform, and the negative sign in the exponent of the forward transform.

2-Separability: Notice that the Fourier transform "filter elements" can be expressed as products:

$$\exp\left[2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)\right] = \exp[2\pi i(\frac{xu}{M})\exp[2\pi i(\frac{yv}{N})] \ldots\ldots\ldots(10)$$

The first product value

$$\exp[2\pi i(\frac{xu}{M})]$$

depends only on x and u, and independent of y and v. Conversely, the second product value

$$\exp[2\pi i(\frac{yv}{N})]$$

depends only on y and v, and is independent of x and u. This means that we can break down formulas above to simpler formulas that work single rows or columns:

$$F(u) = \sum_{x=0}^{M-1} f(\text{x})\exp[-2\pi i\frac{xu}{M}]................(11)$$

$$f(x) = \frac{1}{M}\sum_{u=0}^{M-1} \exp[-2\pi i\frac{xu}{N}]F(u)..............(12)$$

The 2-D, DFT can be calculated by using this property of "separability"; to obtain the 2-D DFT of a matrix, we first calculate the DFT of all the rows, and then calculate the DFT of all the columns of the result, as shown in figure 1

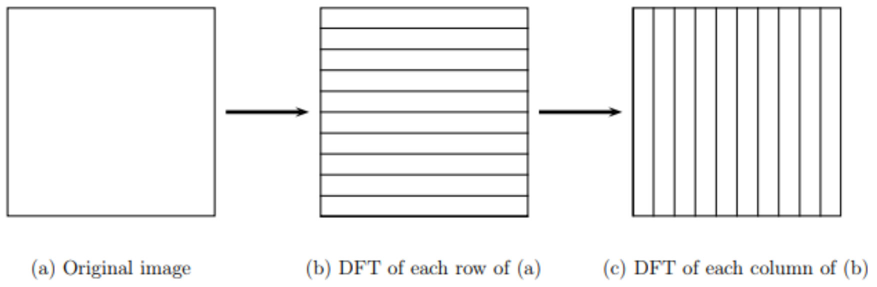


Figure 1 : Calculating of DFT

3-Linearity: An important property of the DFT is its linearity;

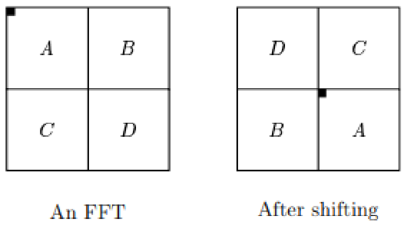
$\digamma(f+g)=\digamma(f)+\digamma(g)$ ………..(13)

$\digamma(\text{kf})=k\digamma(f)$ ………...(14)

The DC coefficient. The value F(0,0) of the DFT is called the DC coefficient;

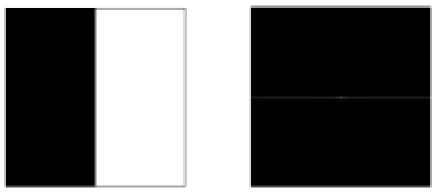
$$F(0,0) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\exp(0) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y).....(15)$$

4- Shifting: For purposes of display, it is convenient to have the DC coefficient in the center of the matrix. This will happen if all elements in the matrix are multiplying by $(-1)^{x+y}$ before transform.

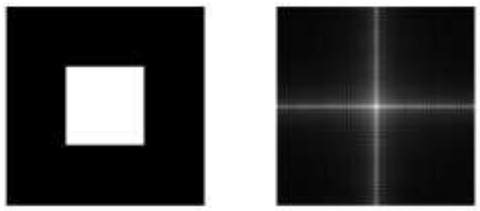


Fourier transforms of images:

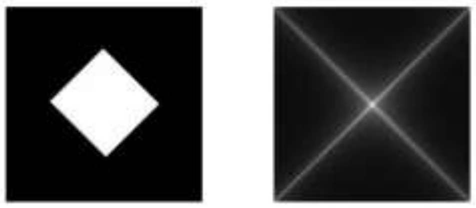
Example 1: produce a simple image consisting of a single edge and its DFT.



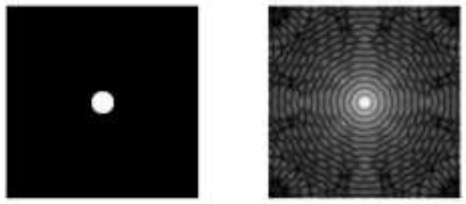
Example 2: create a box, and then its Fourier transform



Example 3: A box rotate $45^0$and its DFT



Example 4: Create a circle and its DFT



-An image can be expanded in terms of a discrete set of basis arrays called basis images. Unitary matrices can generate these basis images. An image transform provides a set of coordinates or basis vectors for vector space.

Applications of Transform:

To reduce band width

To reduce redundancy

To extract feature.

The Conditions for Perfect Transform

Transpose of matrix = *Inverse of a matrix** *Orthogonality*.

Discrete Cosine Transform:

its computational efficiency the DFT is very popular represent the signal in the frequency domain. This is very important for image compression. A much better transform.

The DCT equation (eq.16) computes the I,jth entry of the DCT of an image

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right] \ldots\ldots(16)$$

$$C(u)=\left\{\begin{matrix} \frac{1}{\sqrt{2}} & if\ u=0 \\ 1 & if\ u>0 \end{matrix}\right\} \ldots\ldots\ldots\ldots\ldots\ldots(17)$$

p(x,y) is the x,yth element of the image represented by the matrix p and N is the block size that is DCT done.

The properties of the Cosine Transform:

-Real & orthogonal.

-Fast transform.

-Has excellent energy compaction for highly correlated data

Fast transform. For 8x8 block, there we D(i,j) would be in eq.18

$$D(i,j) = \frac{1}{4} C(i)C(j) \sum_{x=0}^{7} \quad \sum_{y=0}^{N7} p(x,y) \cos\left[\frac{(2x+1)i\pi}{16}\right] \cos[\frac{(2y+1)j\pi}{16}] \ldots\ldots(18$$

Image compression

Image compression is the process of encoding or converting an image file in such a way that it consumes less space than the original file.

compression technique that reduces the size of an image or is minimizing the size in bytes of a graphics file without affecting or degrading its quality to a greater extent.

Image compression is typically performed through an image/data compression algorithm or codec. Typically, such codecs/algorithms apply different techniques to reduce the image size.

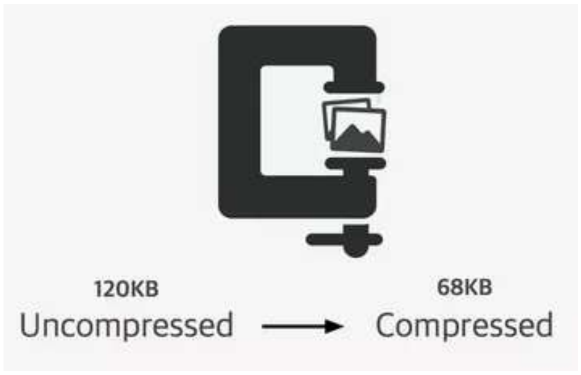


Image Compression model:

image compression is very important task in image processing. Images and videos require lots of space and large transmission time.

data compression is the mathematical process of transforming data to a smaller representation from the original

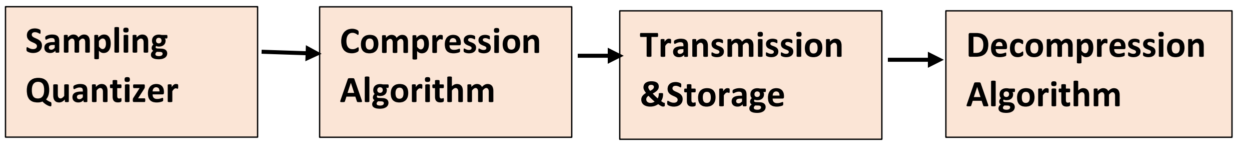
Why data compression?

Storage

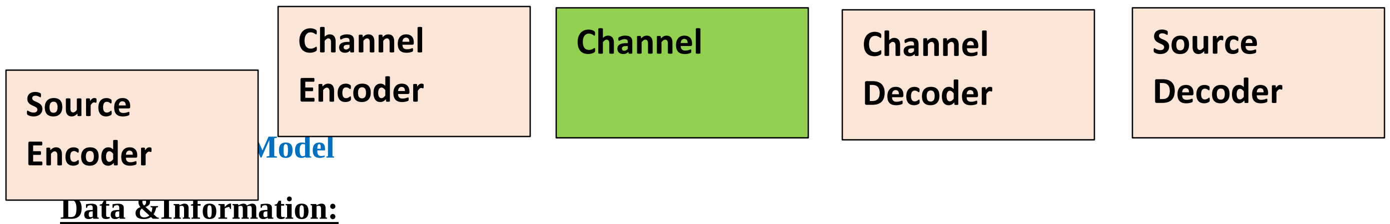
Transmission

Faster computation

Compression Scheme



Two main component of image compressor model are : **Encoder** and **Decoder**



Model

Data &Information:

-Data and information are two different things:

-Data is raw facts which are encountered in image processing.

-Information is an interpretation of the data in a meaningful way.

-Data is the means by which information is conveyed.

-Data redundancy is a central issue in digital image compression.

There are types of data:

-Text data----flat files

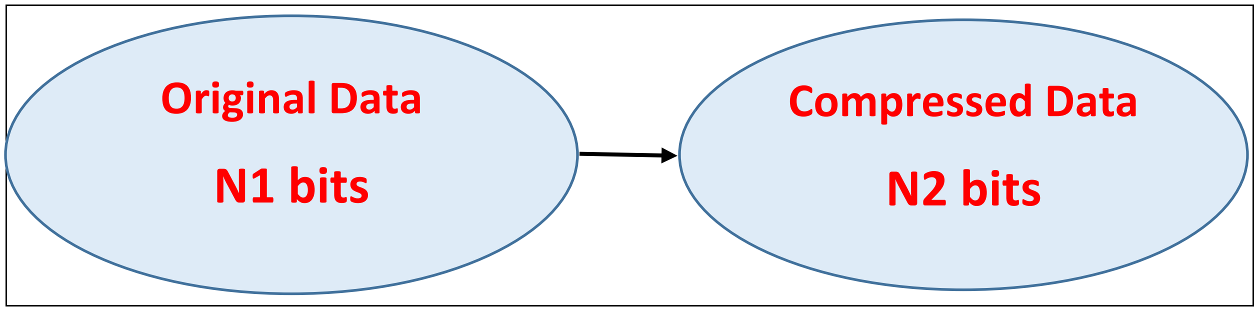
-Binary data—machine can interpret it

-Image data---pixel data

-Graphic data ---vector form

-Sound data----audio information

-Video data ----video information



The relative data redundancy R_D:

$$\boldsymbol{R_D = 1 - \frac{1}{C_R}} \ldots\ldots\ldots\ldots\text{(1)}$$

where C_R is compression ratio

$$\boldsymbol{C_R = \frac{N1}{N2}} \ \ldots\ldots\ldots\ldots\ldots\text{(2)}$$

If N1==N2	C_R=1 There is no compression &no redundancy data
If N2<<N1	C_R= There is significant compression& High redundant data
If N2>>N1	C_R=0 Data Explosions& unbearable data

Basic Redundancies:

Redundancy means repetitive data. This may be data that share some common characteristics or overlapped information.

Redundancy may be implicitly and explicitly.

Explicitly

Ex. AAAAABB

{(A,5),(B,2)}

Ex. Image

$$\begin{bmatrix} 5 & 8 & 5 \\ 8 & 5 & 8 \\ 5 & 5 & 8 \end{bmatrix}$$

5----5

8----4

<u>Implicitly</u>

Ex.

I=$\begin{bmatrix} 00 & 11 \\ 00 & 10 \end{bmatrix}$ I= $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ and I=$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

Coding Redundancy: The average length of the code words assigned to the various gray level values:

$$\boldsymbol{L_{av} = \sum_{k=0}^{L-1} I(r_k) P_k(r_k)} \quad \text{k=0,1,2,….,L-1} \;\ldots\ldots\ldots\ldots(3)$$

where $I(r_{k)}$ no. of bits used to represent each gray

$$P_k(r_k)\ probability\ that\ grray\ level\ occurs$$

$$\boldsymbol{P_k(r_k) = \frac{N_k}{N}} \;\ldots\ldots\ldots\ldots\ldots\ldots(4)$$

where L is the number gray levels.

N_k is the number of times that K^th gray level appears in image

N is the total number of pixel in the image.

Example: Suppose sub image 3bits

1	**5**	**2**	**0**
2	**7**	**3**	**1**
0	**3**	**0**	**5**
3	**3**	**2**	**1**

k	**N_k**	**P_k**	**Code1**(Before compression)	**I1(r_k)**	**Code2**(After compression)	**I2(r_k)**
0	**3**	**.1875**	**000**	**3**	**11**	**2**
1	**3**	**.1875**	**001**	**3**	**01**	**2**
2	**3**	**.1875**	**010**	**3**	**10**	**2**
3	**4**	**.25**	**011**	**3**	**001**	**3**
4	**0**	**0**	**100**	**3**	**0001**	**4**
5	**2**	**.125**	**101**	**3**	**00001**	**5**
6	**0**	**0**	**110**	**3**	**000001**	**6**
7	**1**	**.0625**	**111**	**3**	**000000**	**6**

By using equation (3)

L_{av}=2*.1875 +2*.1875 +2*.1875 +3*.25 +4*0+5*.125+6*0+6*.0625=2.875 bits

By using equation (2)

$$C_R=3/2.875=1.04$$

By using equation (1)

$$R_D = 1 - \frac{1}{C_R} = 1 - \frac{1}{1.09} = .038$$

Spatial and Temporal Redundancy (i.e. video sequence)

Irreverent information (i.e. information that ignored by human visual system)

Applications of data compression:

personal commination like facsimile, voice mail and telephony.

computer networks-internet

multimedia applications.

image and signal processing.

digital and satellite TV.

Video conferencing and digital library.

Types of image compression:

There are two kinds of image compression methods:

1-**Lossless compression** is preferred for archival purposes and often for medical imaging, technical drawings, clip art, or comics.

2-**Lossy compression** methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications.

Methods for lossless image compression are:

-Run-length encoding (RLE)runs of data are stored as a single data value and count, rather than as the original run. This is most efficient on data that contains many such runs, for example simple graphic images such as icons, line drawings. That takes redundant strings or runs of data and stores them as one unit. Example you have a picture of red and white stripes, and there are 12 white pixels and 12 red pixels. Normally, the data for it would be written as WWWWWWWWWWWWRRRRRRRRRRRR, with W representing the white pixel and R the red pixel. Run length would put the data as 12W and 12R.

Used in default method in PCX and as one of possible in BMP, TGA, TIFF

-Area image compression

-Predictive Coding used in *Differential pulse-code modulation* (**DPCM**)is a signal encoder that uses the baseline of pulse-code modulation (PCM) but adds some functionalities based on the prediction of the samples of the signal. The input can be an analog signal or a digital signal

-Entropy encoding the two most common entropy encoding techniques are *arithmetic coding* and *Huffman coding*.

Adaptive dictionary algorithms such as LZW – used in GIF and TIFF

-DEFLATE used in PNG, MNG, and TIFF .It uses a combination of LZ77 and Huffman coding to achieve compression results that do not affect the quality of the image.

-Chain codes

is a lossless compression algorithm for monochrome images. The basic principle of chain codes is to separately encode each connected component, or "blob", in the image.

Methods for lossy compression:

Reducing the color space to the most common colors in the image. The selected colors are specified in the color palette in the header of the compressed image. Each pixel just references the index of a color in the color palette, this method can be combined with dithering to avoid pasteurization.

Fractal compression*:* The method is best suited for textures and natural images, relying on the fact that parts of an image often resemble other parts of the same image. Fractal algorithms convert these parts into mathematical data called "fractal codes" which are used to recreate the encoded image..

-Transform coding is the most commonly used method. There are millions of shades of colors, and transform encoding takes colors that have similar shades and makes them one single value.

-Discrete Cosine Transform(DCT*)* is the most widely used form of lossy compression. It is a type of Fourier-related transform. DCT is used in JPEG, the most popular lossy format, and the more recent HEIF.

The more recently developed **wavelet transform** is also used extensively, followed by quantization and entropy coding.

-Chroma subsampling is the practice of encoding images by implementing less resolution for chroma information than for luma information, taking advantage of the human visual system's lower acuity for color differences than for luminance. It is used in many video encoding schemes – both analog and digital, and also in JPEG encoding

Pro.Dr. Azmi Tawfiq. Hussain Al-Rawi,
University of Anbar of Commuter science & I.T.
Department of Computer science \4th stage
CVIP

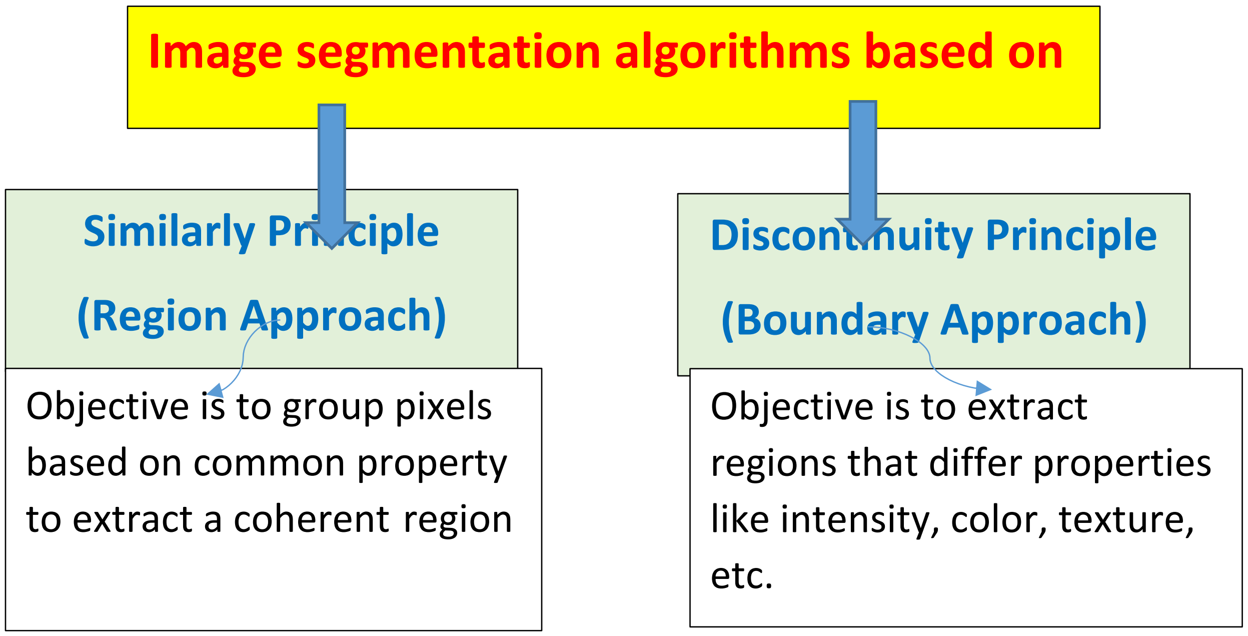
Image Segmentation

segmentation is the process to partition a digital images into multiple regions and extracting the meaningful region which is known as region of interest(ROI).

Region of interest varies with applications.

In fact, no signal universal segmentation algorithm exists for segmenting the ROI in all images.

therefore, many segmentation algorithms need to apply and pick that algorithm which performs the best for given requirement.



Objective is to group pixels based on common property to extract a coherent region

Objective is to extract regions that differ properties like intensity, color, texture, etc.

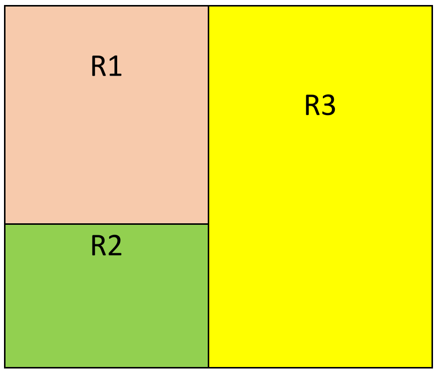
Definition of image segmentation:

An image can be portioned into to many regions R1,R2,..Rn

R1	R21	R22
	R23	R24
R2	R4	

Example

15	15	30	30	30	30
15	15	30	30	30	30
15	15	30	30	30	30
15	15	30	30	30	30
20	20	30	30	30	30
20	20	30	30	30	30



Characteristics of Segmentation Process:

Let R be entire image region and segmentation is partitioning R into n subgroups Ri

*$\cup_{i=1}^{n} Ri = R \quad i = 1,2,3, \dots \dots n$

*Ri should be connected region , i=1,2,3,…,n

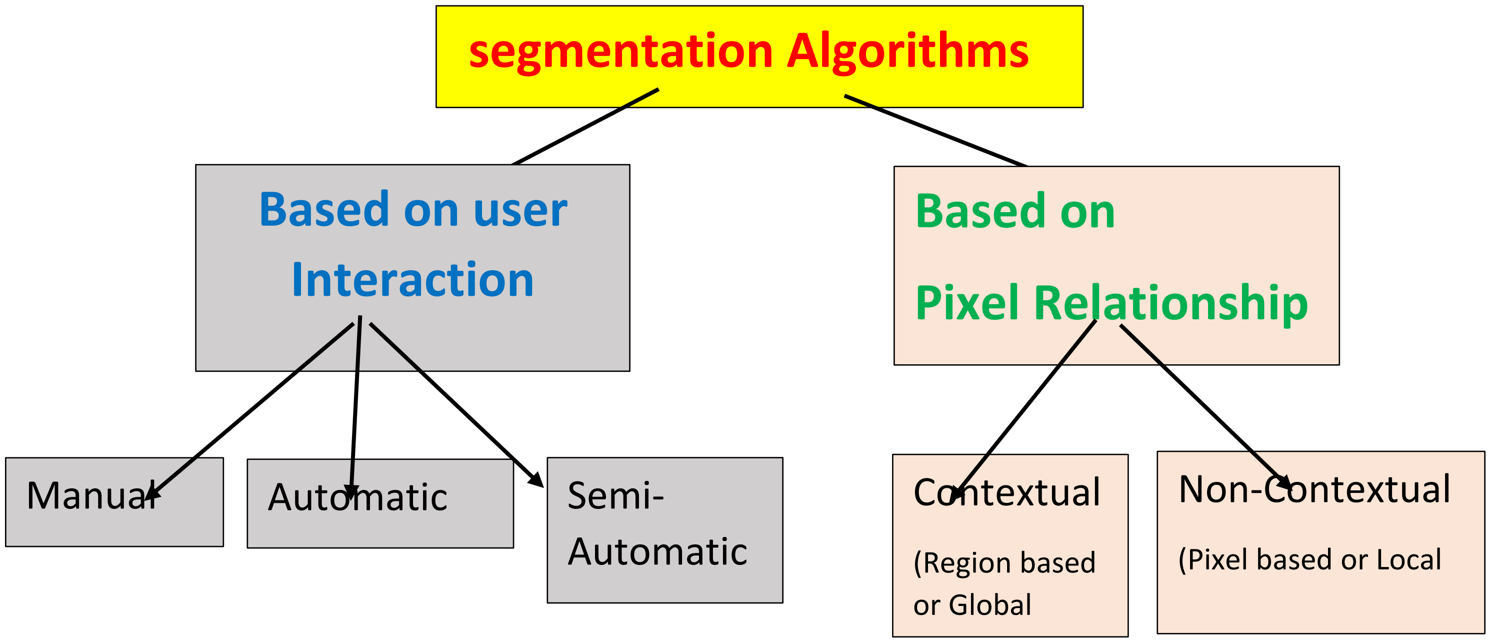
*$Ri \cap R1 = \emptyset (for\ all\ i\ and\ j,\ i \neq j)$

*$P(Ri) = True\ for\ i = 1,2,3, .., n$

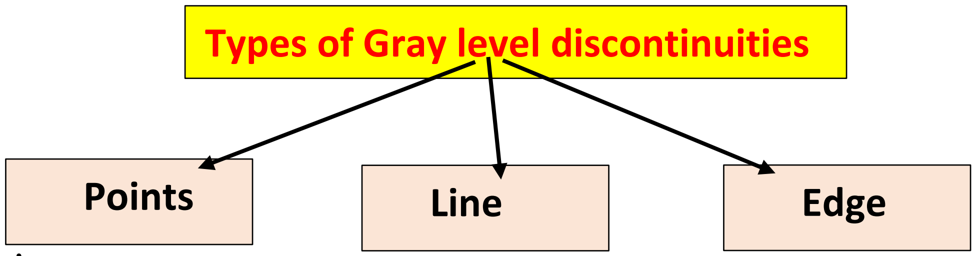
*$P(Ri \cup Rj) = False\ for\ any\ adjusten\ region\ Ri\ and\ Rj, i \neq j$

Here *P(Rj)* is predicated that indicated some property over this region

Classification of image segmentation Algorithms:



Detection of discontinuities:



Isolated points:

An isolated point is a point whose gray level is significantly different from its background in a homogenous area

mask

m1	m2	m3
m4	m5	m6
m7	m8	m9

Image

x1	x2	x3
x4	x5	x6
x7	x8	x9

The response of the mask is:

$$R = \sum_{i=1}^{9} m_i x_i \text{ ....(1)}$$

if $|R| \geq T$,

$$a\ point\ is\ detected\ where\ T\ is\ a\ non\ negative\ integer$$

Example: Mask for point detection

-1	-1	-1
-1	8	-1
-1	-1	-1

Line detection:

There are four types of masks are used to get the responses (R1,R2,R3,R4) for the directions

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal

-1	2	-1
-1	2	-1
-1	2	-1

Vertical

-1	-1	2
-1	2	-1
2	-1	-1

+45°

2	-1	-1
-1	2	-1
-1	-1	2

-45°

The response of the mask:

$$R_k = \sum_{k=1}^{4} m_k x_k \text{ ....(2)}$$

If a certain point in the image,$|R_i|>|R_j|$ for all $i \neq j$, that point is said to be more likely associated with a line in the direction of mask

Edge detection

An edge is a set of connected pixels that lies on the boundary between two regions which differ in gray value. pixels on edge is known as edge points.

Edges provide an outline of the object.

in physical plane, edge corresponds to the discontinuities in depth, surface orientation changes in material properties, light variation etc.

It locates sharp changes in the intensity function.

Edges are pixels where brightness changes abruptly

An edge can be extracted by computing the derivative of the image function

Magnitude of the derivative indicates the strength or contrast of edge

Direction of the derivative vector indicates the edge orientation.

There are some of the commonly encountered edges in image processing (the details in lect.12):

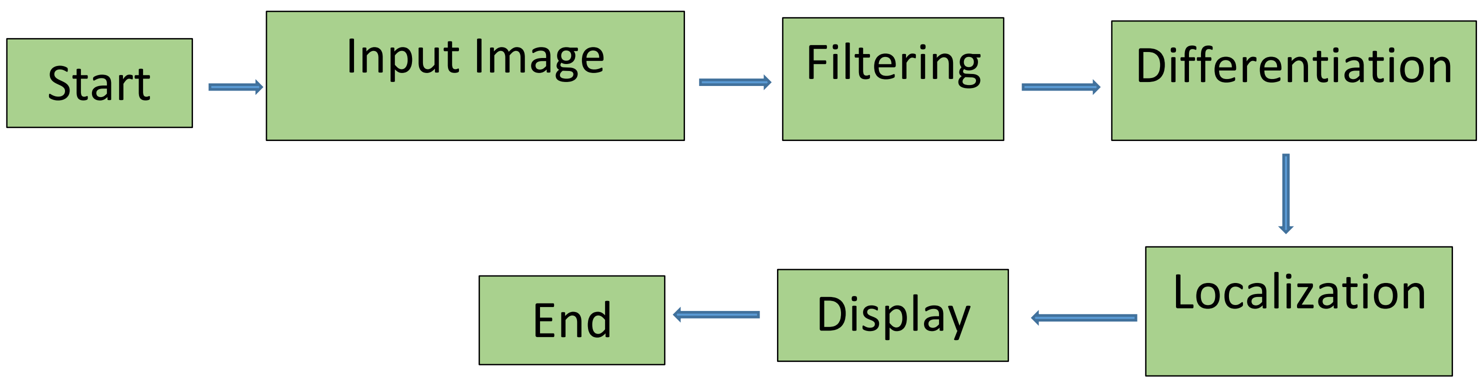
Step Edge

Ramp Edge

Spike Edge

Roof Edge

Stages in Edge detection



Filtering: It movies smoothing. This stage may be performed explicitly or implicitly.

Differentiation: It distinguishes the edge pixels from other pixel.

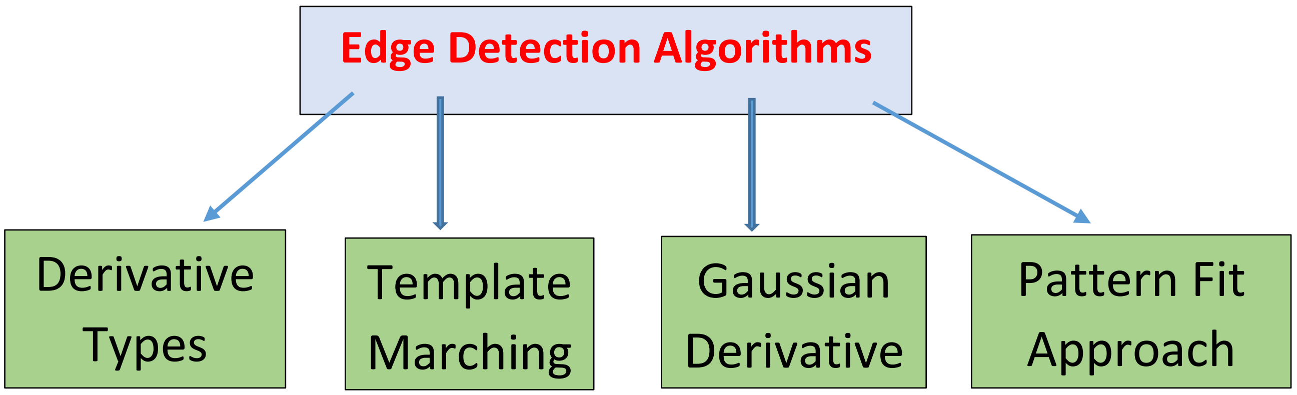
If image *f(x,y)*

$$Vector : \nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$Magnitude : \nabla f = mag(\nabla \mathbf{f}) = [G_x^2 + G_y^2]^{\frac{1}{2}}$$

$$Direction : tan^{-1}(G_y / G_x)$$

Localization: determine the exact location of edge.



Region Based Segmentation-Basics

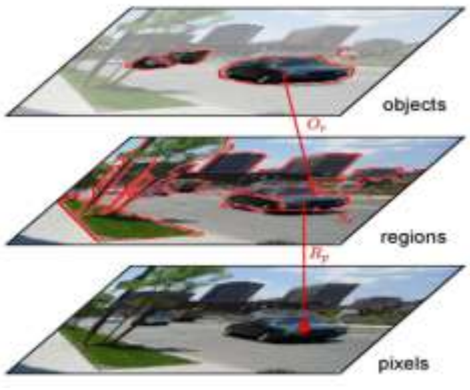
Region: A group of connected pixels with similar properties (*closed boundaries*).

Region growing is a procedure that groups pixels or subregions into larger regions based on predefined criteria for growth (*Principle of Similarity*).

principle of similarity states that a region is coherent if all pixels of that region are homogeneous is used as the main segmentation criterion in region growing

(gray level , color, texture , shape model, et.c..)

Computation of regions is **based on** similarity regions may correspond to objects in a scene or parts of objects



Digital Image Algorithms:

1-Seed Pixels

The basic approach is to start with a set of "seed" points, and from these grow regions by appending to each seed those neighboring pixels that have predefined properties similar to the seed (such as ranges of intensity or color)

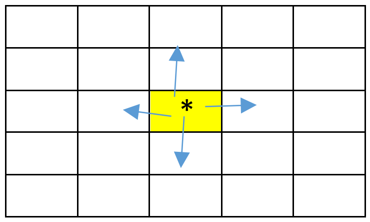
For segment generation in grey-level or color images, we may start at one seed pixel (x,y,I(x,y)) and add recursively adjacent pixels that satisfy a "similarity criterion" with pixels contained in the so-far grown region around the seed pixel.

Seeded Segmentation Algorithm:

1. Choose the initial seed pixel

2. Check the neighboring pixels and add them to the region if they are similar to the seed.

3. Repeat step 2 for each of the newly added pixels; stop if no more pixels can be added.



|f(x,y)-f(x/,y/)|<=T

T is thresholding

|neighboring pixels- seed| <= Threshold

<u>Example:</u>

1	0	7	8	7
0	1	8	9	8
0	0	7	9	8
0	1	8	8	9
1	2	8	8	9

seed pixels :

s1=9, s2=1

thresholding, T<=4

|f(x,y)-f(x/,y/)|<=4

<u>s1=9</u>

|f(x,y)-9|<=4

f(x,y)={5,6,7,8,9} …………A

<u>s2=1</u>

|f(x,y)-1|<=4

f(x,y)={0,1,2,3,4,5} ………B

B	B	A	A	A
B	B	A	A	A
B	B	A	A	A
B	B	A	A	A
B	B	A	A	A

2-Region splitting and Merging Segmentation:

A-Region splitting:

Unlike region growing, which starts from a set of seed points, region splitting starts with the whole image as a single region and subdivides it into subsidiary regions recursively while a condition of homogeneity is not satisfied.

B-Region merging:

Region merging is the opposite of splitting, and works as a way of avoiding over-segmentation

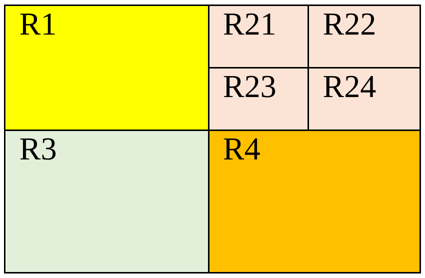
Algorithm:

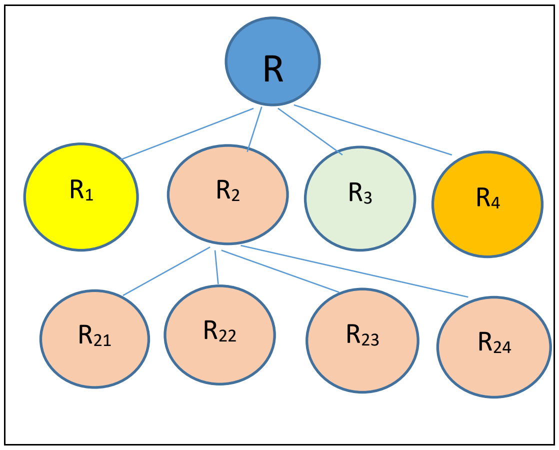
Start with small regions (2x2 or 4x4 regions) and merge the regions that have similar characteristics (such as gray level, variance).

-If a region R is inhomogeneous P(R)=FALSA, then R is split into four sub-regions

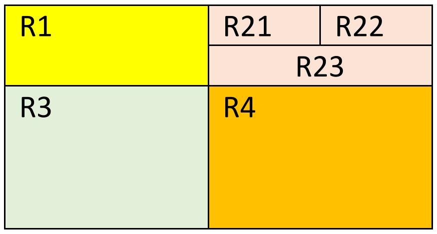
-If two adjacent region R_i, R_j are homogeneous $P(R_i \cup R_j)$ =TRUE , then merged

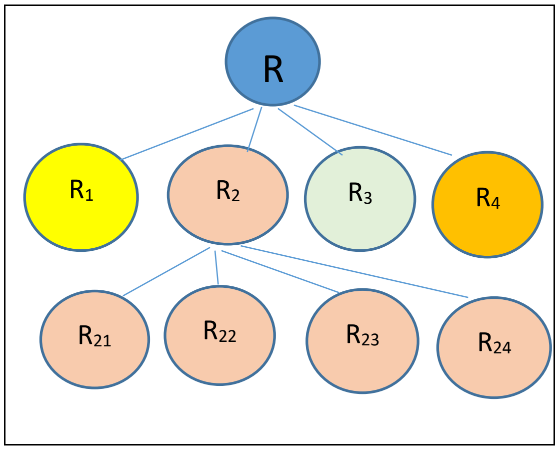
-The algorithm stops when no further splitting or merging is possible





quadtree of splitting





quad tree of splitting and merging

Example :For a given image apply split and merge algorithm

6	5	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

let threshold T <=3

max. value =7

min. value =0

the difference between max. and min. pixels = 7-0=7

	6	5	6	6	7	7	6	6	
	6	7	6	7	5	5	4	7	
A	6	6	4	4	3	2	5	6	C
	5	4	5	4	2	3	4	6	
	0	3	2	3	3	2	4	7	
B	0	0	0	0	2	2	5	6	D
	1	1	0	1	0	3	4	4	
	1	0	1	0	2	3	5	4	

At region A :7-4=3

At region B: 3-0=0

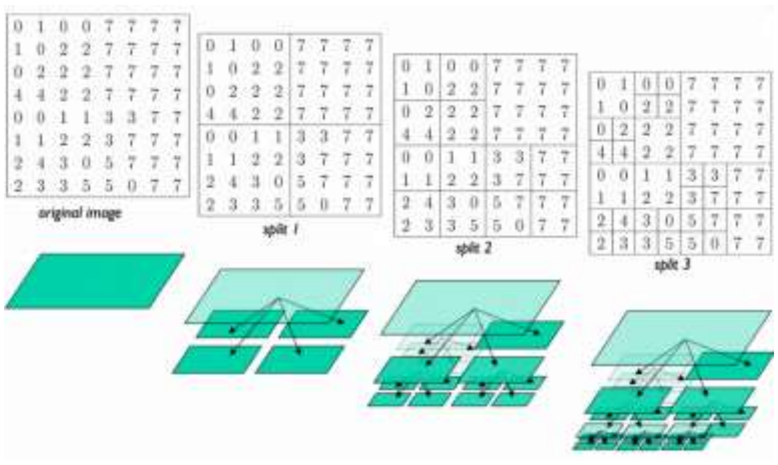
At region C :7-2=5

At region D: 7-0=0

6	5	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

6	5	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

Example



0	1	0	0	7	7	7	7
1	0	2	2	7	7	7	7
0	2	2	2	7	7	7	7
4	4	2	2	7	7	7	7
0	0	1	1	3	3	7	7
1	1	2	2	3	7	7	7
2	4	3	0	5	7	7	7
2	3	3	5	5	0	7	7

References:

Dr. Sapana Kaityar https://www.youtube.com/watch?v=i3ANIRt9qRg.

Gonzales R.C. and Woods P., Digital Image Processing, Addison- Wesley,4th edition, 2018

=Alasdair McAndrew, An Introduction to Digital Image Processing with Matlab Notes for SCM2511 Image Processing