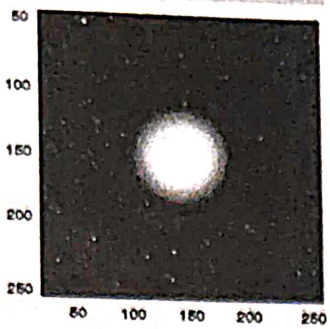
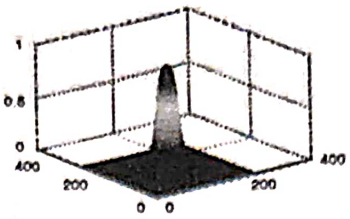
figure (3) , imshow(uint8(X))

figure (4) , imagesc(H),colormap(gray)



(a) Original image (b) 2-D Butterworth filter

Fig. 6.5.12 Continued...



(c) Butterworth frequency response

(d) Butterworth low-pass filtered image order = 2, cut-off = 25

Fig. 6.5.12

6.5.3 Gaussian Low Pass Filter (GLPF)

Gaussian LPF is given by,

$$H(u, v) = e^{-D^2(u, v)/2\sigma^2} \quad ...(6.5.4)$$

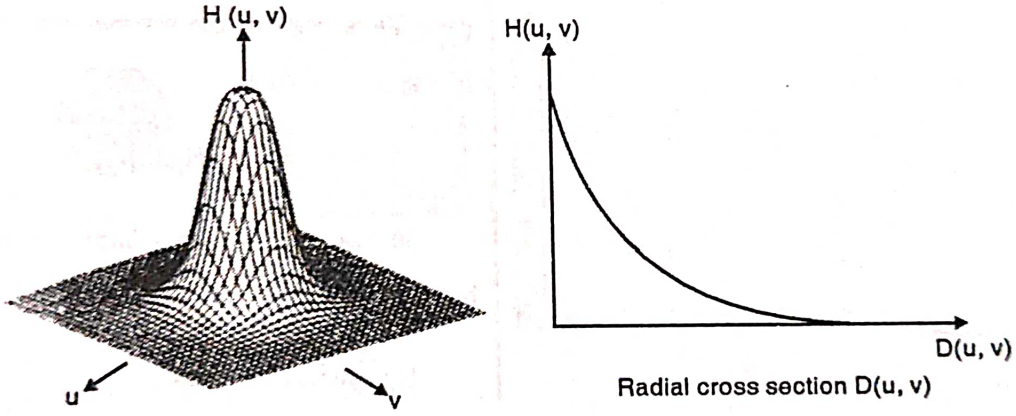


Fig. 6.5.13

Here σ is the standard deviation and is a measure of spread of the Gaussian curve.

If we put $\sigma = D_0$ we get,

$$H(u, v) = e^{-D^2(u, v)/2D_0^2}$$

The response of the Gaussian LPF is similar to that of BLPF but there are no ringing effects at all !!!!

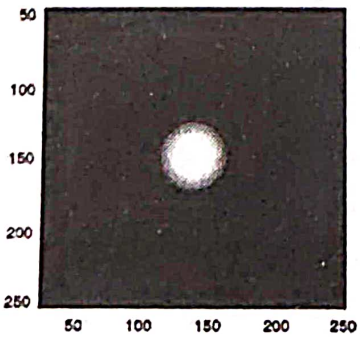
The reason for studying the Gaussian low pass filter is that it has a very important property in the Fourier domain which can be used as an analysis tool. We shall explain that when we try to study the relationship between the spatial domain masks and their frequency response.

MATLAB program for Gaussian LPF on square images

```
%% Low pass Gaussian filter %%
clear all
clc
a = mread('cameraman.tif') ;
a = double(a) ;
c = size(a) ;
N = c(1)
```

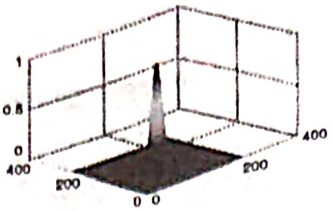
```
D0 = input ('Enter the cut-off frequency / standard deviation');
for u = 1:1:c(1)
    for v = 1:1:c(2)
    Dx = ( (u – (N/2)) ^2 + (v – (N/2) ) ^2) ^0.5 ;
    D = Dx*Dx
        %% In the formula we need to take the square %%
    H(u,v) = exp(– D / (2*D0*D0) ) ; %% D0 is the standard
                                       deviation,..
                                    %%..sigma%%
  end
end
vv = fft2(a);
vc = fftshift(vv);
x = vc.*H;
X = abs(ifft2(x));
%% Plotting
figure (1) , imshow(uint8(a))
figure (2) , mesh(H)
figure (3) , imshow(uint8(X))
figure (4) , imagesc(H),colormap(gray)
```

Main advantage of a Gaussian LPF over a Butterworth LPF is that we are assured that there will be no ringing effects no matter what filter order we choose to work with.



(a) Original image **(b) 2-D Gaussian filter**

Fig. 6.5.14 Continued...



(c) Gaussian filter response

(d) Gaussian low pass filtered image

Fig. 6.5.14

6.6 High Pass Frequency Domain Filters

Edges and other abrupt changes in the grey levels are associated with high frequency components while the slowly varying changes are the low frequency areas. Since high pass filters perform the reverse operation of the low pass filters, we can write.

$$H_{hp}(u, v) = 1 - H_{lp}(u, v) \quad \ldots(6.6.1)$$

Just as in the low pass filter case, we shall define three basic types of high pass filters

(a) Ideal High Pass

(b) Butterworth High Pass

(c) Gaussian High Pass

6.6.1 Ideal High Pass Filters (IHPF)

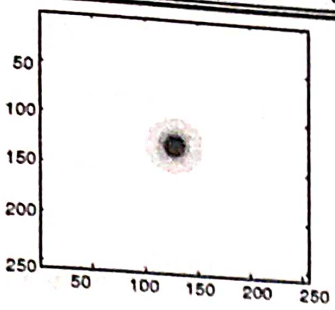
A 2-D ideal high pass filter is defined as

$$H(u, v) = 0 ; \text{ if } D(u, v) \leq D_0 \quad \ldots(6.6.2)$$

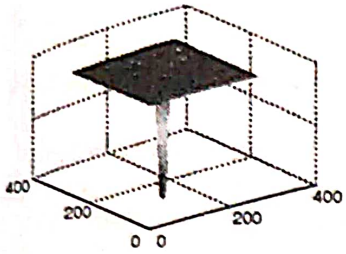
$$= 1 ; \text{ if } D(u, v) > D_0$$

where D_0 is called the cut-off. D(u, v) is the same as defined in the low pass filtering section.

(a) Original image



(b) 2-D Butterworth high pass filter



(c) Butterworth high pass filter response

(d) Butterworth high pass image cut-off 10

Fig. 6.6.5

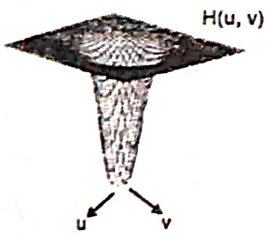
6.6.3 Gaussian High Pass Filters (GHPF)

We again use the basic formula

$$H_{hp}(u, v) = 1 - H_{lp}(u, v)$$

$$\therefore H_{GaussianHP}(u, v) = 1 - H_{GaussianLP}(u, v)$$

$$H_{GHPF} = 1 - e^{-D^2(u, v)/2D_0^2} \quad ...(6.6.4)$$



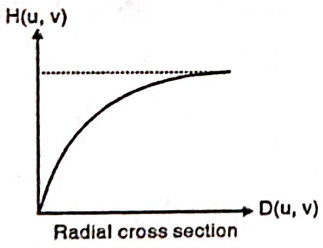


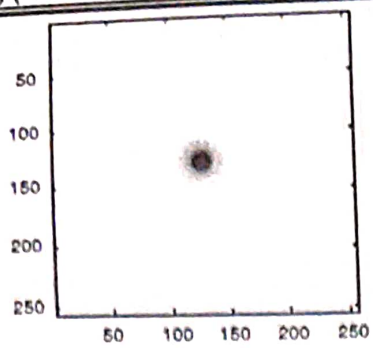
Fig. 6.6.6

The results of Gaussian high pass filter are shown in Fig. 6.6.7. The results are smoother than the ones discussed previously. The results are also cleaner.

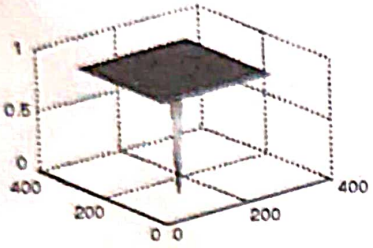
MATLAB code for Gaussian high pass filter on square images

```
%% Gaussian high pass filter %%
clear all
clc
a = imread('cameraman. tif ') ;
a = double(a) ;
c = size(a) ;
N = c(1)
D0 = input('Enter the cut-off frequency/standard deviation ');
for u = 1:1:c(1)
    for v = 1:1:c(2)
    Dx = ((u – (N/2)) ^ 2+(v – (N/2)) ^ 2) ^ 0.5;
    D = Dx*Dx; %% In the formula we need to take the
                square %%
    H(u,v) = 1-exp(-D/(2*D0*D0));
  % Here D0 is taken as the standard % deviation sigma %%
    end
end
vv = fft2(a);
vc = fftshift(vv)
x = vc.*H;
X = abs(ifft2(x));
%%Plotting%%
figure (1) , imshow (uint8(a) )
figure (2) , mesh(H)
figure (3) , imshow(uint8(X) )
figure (4) , imagesc(H),colormap (gray)
```

(a) Original image



(b) 2-D Gaussian high pass filter



(c) Gaussian high passed filter response

(d) Gaussian high passed image

Fig. 6.6.7

6.6.4 High Boost Filtering (Unsharp Masking / High Frequency Emphasis)

We have already seen what we mean by high boost filtering in the spatial domain. Since high pass filters completely remove the low frequency components, we get only the edges. There are applications where we need to preserve some of the low frequency components. This can be achieved by high boost filtering. We shall use the following abbreviated forms in the frequency domain.

F_{HP} → High pass filtered image

F_{LP} → Low pass filtered image

H_{LP} → Low pass filter

H_{HP} → High pass filter

H_{HB} → High boost filter

We have seen that in the spatial domain.

High boost = (A – 1) Original + High pass

i.e. $F_{HB}(x, y) = (A - 1) f(x, y) + F_{HP}(x, y)$

What happens to this in the frequency domain ?

We know, High pass = Original – Low pass

i.e., $F_{HP}(x, y) = f(x, y) - F_{LP}(x, y)$

Taking the Fourier transform

$$F_{HP}(u, v) = F(u, v) - F_{LP}(u, v)$$

$$\text{But } F_{LP}(u, v) = H_{LP}(u, v) \cdot F(u, v)$$

$$\therefore F_{HP}(u, v) = [1 - H_{LP}(u, v)] F(u, v)$$

$$\therefore H_{HP}(u, v) \cdot F(u, v) = [1 - H_{LP}(u, v)] F(u, v)$$

$$\therefore H_{HP}(u, v) = [1 - H_{LP}(u, v)]$$

In a similar manner

$$H_{HB}(u, v) = (A - 1) + H_{HP}(u, v); \quad A \geq 1$$

MATLAB code for high boost filtering on square image

```
%% High boost filtering in the frequency domain %%
clear all
clc
a = imread('tire.tif');
a = double(a);
[row col]= size (a);
for u = 1:1:row
for v = 1:1:col
D = ((u– (row/2)) ^ 2+(v-(col/2)) ^ 2) ^ 0.5;
  if D<10 ;%% Cut-off is taken as 10. It could be changed
  H(u,v) = 0 ;          %% .. as per the requirement
  else
  H(u,v) = 1 ;
    end
end
end
A = 1.5 ;
      %% This is the high boost factor. It could be changed
H1 = (A-1)+ H ;      %% .. as per requirement
```

6.7 Homomorphic Filtering

Homomorphic filter uses a slightly different approach compared to the frequency domain filters studied so far. Before studying the homomorphic filter, we need to study the image formation model.

Image Formation Model

We have seen that an image is a 2-D function. A grey level (monochrome) image is given as f(x, y) where f is the grey level and x and y are the spatial coordinates.

Images that we see are formed due to the fact that there is some amount of light falling on the object and some amount of light reflecting from the object.

Light falling on the object is called illumination and light reflecting from the object is called reflectance.

Since images that we see fall in the electromagnetic spectrum,

$$0 < f(x, y) < \infty$$

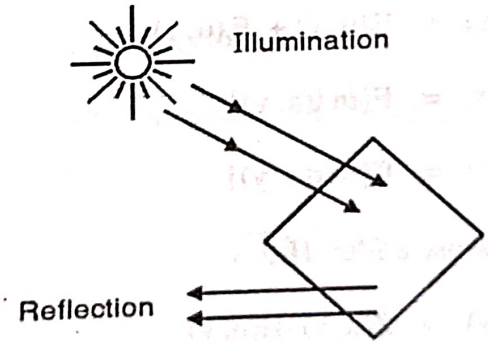


Fig. 6.7.1

f(x, y) is characterized by two components :

(1) The amount of light incident on the scene/object. (illumination)

(2) Amount of light reflected by the object. (reflection)

They are called illumination and reflectance components i(x, y) and r(x, y) respectively.

These two combine as a product to form f(x, y)

$$\therefore \quad f(x, y) = i(x, y)\, r(x, y) \qquad \ldots(6.7.1)$$

where $0 < i(x, y) < \infty$

Review Questions

Q. 1 Explain the frequency domain concept.

Q. 2 What is advantage of processing an image in the frequency domain ?

Q. 3 Describe the basic principles of image enhancement by

(a) Spatial domain methods

(b) Frequency domain methods

Q. 4 Derive the Fourier transform of the given pulse. Explain the Fourier spectrum that you obtain.

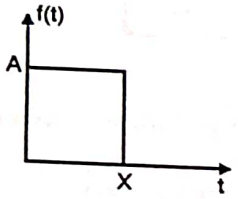


Fig. Q. 4

Q. 5 What are blurring and ringing effects ? How can they be avoided ?

Q. 6 Compare the basic frequency domain filters

(a) Ideal low pass

(b) Butterworth low pass

(c) Gaussian low pass.

Q. 7 Explain the concept of the power spectrum. What information does it yield ?

Q. 8 Explain high boost filter in the frequency domain.

Q. 9 Explain the image model. What do the illumination and reflectance components denote in terms of frequency ?

Q. 10 Explain the homomorphic filter.

Q. 11 Explain the various properties of the Fourier transform. Explain the relevance of each in image processing.

Q. 12 A 2-dimensional-DFT can be obtained using a 1-dimensional-DFT algorithm twice; Explain.

Q. 13 Explain convolution in the spatial and the frequency domain. Derive the relationship between the two domains.

Chapter Ends...

We shall explain segmentation based on discontinuities first and then proceed to explain segmentation by similarities later.

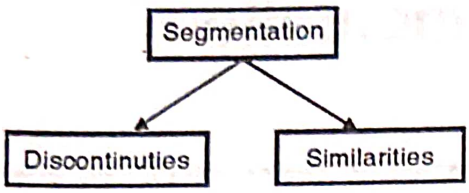


Fig. 7.1.1

7.2 Image Segmentation based on Discontinuties

When we look at an image, we immediately observe 3 basic types of discontinuities in the grey level viz. points, lines and edges. The easiest way is to use spatial masks which have properties to detect these discontinuities.

7.2.1 Point Detection

Remember one thing, points, lines and edges are all high frequency components and hence we need masks which are basically high pass. Hence the masks that we present here for point, line and edge detection have the properties of a high mask mainly, the sum of the coefficients of the mask has to be equal to zero in all the three cases.

Detecting points is fairly simple. We use the standard high pass mask for it. And so that this mask detects only points and not lines, we set a threshold value i.e., we say a point has been detected at the location on which the mask is centered only if

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

$$|R| \geq T \qquad \ldots(7.2.1)$$

Where R is derived from the standard convolution formula

$$R = w_1 z_1 + w_2 z_2 + \ldots\ldots\ldots + w_9 z_9$$

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

We take | R | because we want to detect both the kinds of points i.e. white points on a black background as well as black points on a white background.

T is a non negative threshold which is defined by the user.

7.2.2 Line Detection

Detection of lines can be done using the masks shown below. In an image, lines can be in any direction and detecting these lines would need different masks.

-1 -1 -1	-1 -1 2	-1 2 -1	2 -1 -1
2 2 2	-1 2 -1	-1 2 -1	-1 2 -1
-1 -1 -1	2 -1 -1	-1 2 -1	-1 -1 2
Horizontal	+ 45°	Vertical	– 45°

We notice that all these masks have a sum equal to zero, and hence all of them are high pass masks. The first mask would respond strongly to lines that are oriented horizontally. The second mask would respond to lines at an angle of + 45°, the third mask would respond strongly to vertical lines and the last mask would respond strongly to lines at an angle of – 45°.

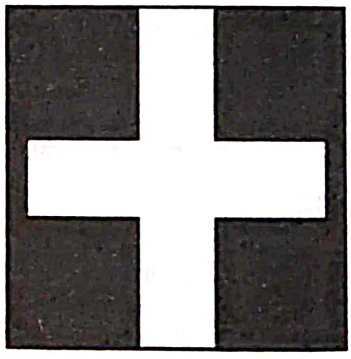


Fig. 7.2.1

Let us take a simple example of a 8×8 pseudo image.

0	0	0	10	0	0	0	0
0	0	0	10	0	0	0	0
0	0	0	10	0	0	0	0
0	0	0	10	0	0	0	0
0	10	10	10	10	10	10	0
0	0	0	10	0	0	0	0
0	0	0	10	0	0	0	0
0	0	0	10	0	0	0	0

It is quite evident that in this image we have two lines, one horizontal and the second vertical.

Let us see what happens when we move the first (Horizontal) mask over the image. By now you should be quite comfortable with moving masks on the entire image. We leave the borders and hence start from the encircled pixel.

The result that we obtain is as shown

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	−20	−20	−20	−20	−30	−20	0
0	+40	+40	+40	+40	+60	+40	0
0	−20	−20	−20	−20	−30	−20	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Since line detection masks are high pass masks, we encounter negative values. These values are made zero as was discussed in chapter of Image Enhancement in Spatial Domain. Hence the final image obtained is –

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	40	40	40	40	60	40	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

We see that the horizontal mask detects lines only in the horizontal direction and cleans up the other lines (vertical line in this case).

In a similar fashion we can show that vertical, + 45°, and – 45° masks respond strongly to lines in their respective directions. It would be a good idea to try this out yourself.

7.2.3 Edge Detection

More than isolated points and lines, it is the detection of edges that form an important part of image segmentation. An edge can be defined as a set of connected pixels that form a boundary between two disjoint regions.

A typical example of an edge is shown in Fig. 7.2.2.

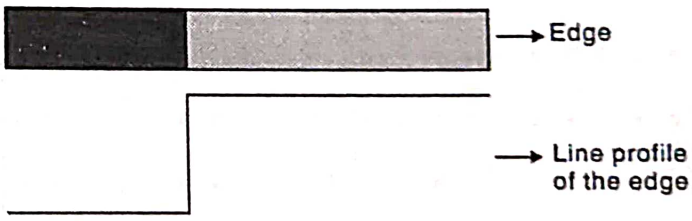


Fig. 7.2.2

Here the set of pixels that separate the black region from the grey region is called an *edge*. The line profile of such an edge is shown along with edge.

The image shown has a step edge. Such step edges occur only in artificially generated images such as test patterns. Images obtained from digitization of optical images of real scenes do not possess step edges. Recollect what happens when we digitize an image. We first sample the image using the Nyquist condition and then quantize it.

CHAPTER 8

Image Transforms

8.1 Introduction

We have studied the Discrete Fourier Transform (DFT) in chapter of Image Enhancement in Frequency Domain. We have also seen specific applications where the DFT is used.

Apart from the DFT, there are a number of discrete linear transformations that prove useful in digital image processing. In this chapter, we shall develop and examine several other transforms which regularly feature in image processing literature.

The term image transform usually refers to a class of unitary matrices used for representing images. Just as a one-dimensional signal can be represented by an orthogonal series of basis functions, an image can also be expanded in terms of a discrete set of basis arrays called basis images.

8.2 Linear Transformations

8.2.1 One Dimensional Discrete Linear Transformations

If x is a $N \times 1$ vector and T is a $N \times N$ matrix, then

$$y_i = \sum t_{i,j} \cdot x_j \qquad i = 0, 1, ..., N-1$$

$$y = Tx \qquad ...(8.2.1)$$

defines a linear transformation of the vector x. The matrix T is called the kernel matrix of the transformation and the resultant y is another $N \times 1$ vector.

The transformation $y = Tx$ is said to be linear because the output y is formed by a first order summation of input elements. Each element of y (y_i) is the inner product of the input vector x with the i^{th} row of T.

8.2.2 Unitary and Orthogonal Matrices

We have seen that $y = Tx$ represents a linear transformation. Now what should be the values of T ? For a given vector length N, there are infinitely many transformation matrices T that could be used. Commonly used T are the ones that belong to a class of matrices having certain properties.

If T is a unitary matrix, then

$$T^{*\prime} = T^{-1}$$

Multiplying both sides by T, we get,

$$T \cdot T^{*\prime} = T \cdot T^{-1} \qquad ...(8.2.2)$$

$$\therefore T \cdot T^{*\prime} = I \qquad ...(8.2.3)$$

Where * indicates complex conjugate of T.

Hence to check whether T is a unitary matrix we need to use the relationship $T\,T^{*\prime} = I$.

If T is unitary and has only real elements, then it is an orthogonal matrix and it follows that

$$T^{\prime} = T^{-1}$$

$$\therefore T \cdot T^{\prime} = I \qquad ...(8.2.4)$$

In other words, an orthogonal matrix is a square matrix whose transpose is its inverse.

Here the rows of T are a set of orthonormal vectors.

To cut a long story short, T is unitary if $T\,T^{*\prime} = I$ and T is orthogonal if $T \cdot T^{\prime} = I$. Always remember this.

Ex. 8.2.1

Check whether the one-dimensional DFT is an example of a unitary transform.

Soln. :

We know that the Fourier transform can be represented as F = Wf where f is the input and W is the DFT matrix. Taking N = 4, we form a DFT matrix.

(This was covered in chapter of Image Enhancement in Frequency Domain.)

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

To check whether W is unitary or not, we need to check the relationship.

$$W \cdot W^{*'} = I$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$$W \qquad W^{*'} \qquad = \quad 4\,[I]$$

Hence the DFT is an example of a unitary transform.

4 indicates that the DFT matrix is not normalised. We compensate for this by using $\frac{1}{N}$ in the forward or inverse DFT formula.

Hence the DFT is an example of a unitary transform.

If f is an input sequence in column form, then the 1-D DFT can be computed using the formula

$$F = Wf \qquad \ldots(8.2.5)$$

8.2.3 Two Dimensional Discrete Linear Transformations

In two dimensions, a linear transformation which transforms an input image f (N × N) into the transformed matrix F (which is also N × N) is given by the equation,

$$F_{m,n} = \sum_{i=0}^{N-1} \sum_{k=0}^{N-1} f_{i,k}\ \Omega(i, k, m, n)$$

Where i, k, m and n are the discrete variables that range from 0 to N – 1. Ω (i, k, m, n) is the kernel function of the transformation.

Ω (i, k, m, n) can be thought of as an $N^2 \times N^2$ block matrix having N rows of N blocks, each of which is a N × N matrix. The blocks are indexed by m, n and the elements of each N × N submatrix by i, k.

$$\begin{array}{c} \\ m=1 \\ m=2 \\ \vdots \\ m=N \end{array} \begin{array}{c} \begin{array}{cccc} n=1 & n=2 & & n=N \end{array} \\ \begin{bmatrix} [\ ]_{i,k} & [\ ]_{i,k} & \cdots & [\ ]_{i,k} \\ [\ ]_{i,k} & [\ ]_{i,k} & \cdots & [\ ]_{i,k} \\ \vdots & \vdots & \cdots & \vdots \\ [\ ]_{i,k} & [\ ]_{i,k} & & [\ ]_{i,k} \end{bmatrix} \end{array}$$

The transformation Ω (i, k, m, n) is called separable if it can be separate into the product of row-wise and column-wise component functions i.e., if

$$\Omega(i, k, m, n) = T_{row}(i, m)\, T_{col}(k, n)$$

This simply means that the transformation can be carried out in two steps – A row wise operation followed by a column wise operation (could be the other way too).

$$\therefore F_{m,n} = \sum_{i=0}^{N-1} \left[\sum_{k=0}^{N-1} f_{i,k}\, T_{row}(i, m) \right] T_{col}(k, n)$$

Further, if the two components i.e., T_{col} and T_{row} are identical, the transform is called symmetric,

$$\Omega(i, k, m, n) = T(i, m)\, T(k, n)$$

$$\therefore F_{m,n} = \sum_{i=0}^{N-1} T(i, m) \left[\sum_{k=0}^{N-1} f_{i,k}\, T_{row}(k, n) \right]$$

$$\text{i.e., } F = T f T \qquad \ldots(8.2.6)$$

The DFT matrix is both symmetric as well as separable and hence can be obtained using the above relationship.

This is the standard formula for computing the transform of a 2D signal. Here T is transformation matrix which is both symmetric and separable. If the transformation matrix is not symmetric, the equation gets modified to

$$F = T f T' \qquad \ldots(8.2.7)$$

Part B

We now compute the 2-D-IDFT of the obtained result.

$$f = \frac{1}{N}[T \cdot F \cdot T']$$

As T is symmetric, we have $f = \frac{1}{N}[T \cdot F \cdot T]$

Since the size of the image is 4×4, $N = 16$

$$f = \frac{1}{16}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}\begin{bmatrix} 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$\therefore f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

8.3 Discrete Cosine Transform (DCT)

The Cosine Transform published by N. Ahmed, T. Natrajan and K. R. Rao has found applications in image compression. All JPEG images use the Discrete Cosine Transform as the initial stage for compression.

Just as the Fourier transform uses sines and cosines waves to represent a signal, the DCT uses only cosine waves. Hence the DCT is purely real unlike the DFT which is complex (has magnitude and phase).

The one dimensional DCT of a sequence f(x), $0 \le x \le N-1$ is defined as,

$$F(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{\pi (2x+1) u}{2N}\right]; \quad 0 \le u \le N-1$$

Where,

$$\alpha(0) = \sqrt{\frac{1}{N}}; \quad \alpha(u) = \sqrt{\frac{2}{N}} \quad \text{for } 1 \le u \le N-1$$

The inverse transformation is given by,

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) F(u) \cos\left[\frac{\pi (2x+1) u}{2N}\right]; \quad 0 \le x \le N-1$$

The corresponding 2-D DCT pair is,

$$F(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{(2x+1) u \pi}{2N}\right] \cos\left[\frac{(2y+1) v \pi}{2N}\right]$$

for $u, v = 0, 1, 2, \ldots N-1$ and

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v) F(u,v) \cos\left[\frac{(2x+1) u\pi}{2N}\right] \cos\left[\frac{(2y+1) v\pi}{2N}\right]$$

for $x, y = 0, 1, 2, \ldots N-1$.

The $N \times N$ Cosine Transform Matrix $C = \{C(u, v)\}$ is defined as,

$$C(u,v) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0, 0 \le v \le N-1 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{\pi (2v+1) u}{2N}\right] & 1 \le u \le N-1, 0 \le v \le N-1 \end{cases} \quad \ldots(8.3.1)$$

It is this equation that we shall use to compute the DCT of a given sequence.

The cosine transform matrix is real and orthogonal but not symmetric.

$$\therefore C = C^* \Rightarrow C^{-1} = C'$$

$$\therefore C \cdot C' = I \quad \text{... from Equation (8.2.3)}$$

The DCT of a column sequence f(x); $0 \le x \le N-1$ can be computed

$$F = C \cdot f \quad \ldots(8.3.2)$$

Similarly, since the discrete cosine transform is not symmetric, the 2-dimensional DCT of an image can be generated using Equation (8.2.7).

$$\therefore F = C f C' \quad \ldots(8.3.3)$$

If all this sounds confusing, a few examples would make you feel at ease.

Ex. 8.3.1

Find the DCT of the following sequence $f(x) = \{1, 2, 4, 7\}$

Soln. :

We shall find the DCT using the matrix notation.

$$F = C \cdot f$$

$$C(u,v) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0, 0 \le v \le N-1 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{\pi (2v+1) u}{2N}\right] & 1 \le u \le N-1, 0 \le v \le N-1 \end{cases}$$

CHAPTER 9

Image Compression

9.1 Introduction

Data Storage and Transmission cost money. In spite of this, most digital data are not stored in a compact form. Rather, they are stored in whatever way that makes them easy to use such as: ASCII text for word processors, binary code that can be executed on a computer, BMP formats for images etc. These easy to use encoding methods require data files about twice as large as actually needed to represent the information. Data compression is the term used for various algorithms developed to address this problem.

We have seen that the amount of storage space required to store images is enormous (Remember $A \times B \times C \times D$?).

Since images are quite data intensive, reducing their size can produce results that are more ambitious than would otherwise be practical. Image compression addresses the problem of reducing the amount of data required to represent a image.

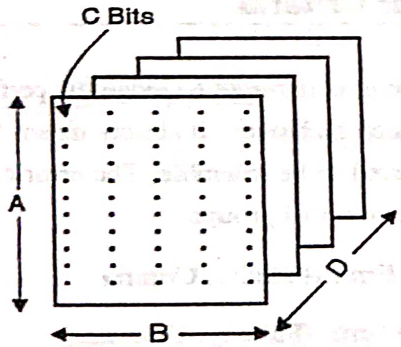


Fig. 9.1.1

9.2 Redundant and Irrelevant Data

Image files commonly contain considerable amount of data that is redundant and much that is irrelevant. Due to this, images are prime candidates for compression. Let us take a simple example to see how compression works. Mr. Rahul who is in Calcutta for a Business meeting hails from Bombay. He is supposed to be boarding the train back to Bombay today evening. He has received a telegram today morning which has the following message.

"Your wife Shilpa will receive you at C.S.T. station in Bombay at 4.30 PM tomorrow afternoon."

Now if this telegraphic message has to be shortened, we need to get rid of information that is already known. Surely Rahul knows that C.S.T. is in Bombay as he stays there. He also does not need to be reminded that his wife's name is Shilpa. When the time is written as 4:30 PM, the word afternoon can be dropped off. Eliminating all this data, we get,

"Your wife will receive you at C.S.T. Station at 4:30 PM tomorrow"

Here we have compressed the data without any loss of information. Hence data compression techniques exploit inherent redundancies and irrelevancies by transforming a data file into a smaller file from which the original file can later be reconstructed exactly or approximately. The ratio of the two file sizes (known as compression ratio) specify the degree of compression. There exists two different categories of data compression algorithms. They could either be Loss-less or they could be Lossy.

The loss-less algorithm eliminates only redundant information, so that one can recover the image exactly upon decompression of the file. In other words the restored image is identical to the original image.

For loss-less algorithms, $f(x, y) = g(x, y)$

Lossy algorithms eliminate redundant as well as irrelevant information and thus permit only an approximate reconstruction of the original image rather than an exact duplicate. Information is lost when lossy algorithms are used.

For Lossy compression $f(x, y) \neq g(x, y)$. $g(x, y)$ is smaller compared to $f(x, y)$.

Loss-less algorithms are absolutely necessary for many types of data, for example : Executable codes (EXE's), word processing files, tabulated numbers etc. In comparison, data files that represent images and other acquired signals do not have to be kept in perfect condition for storage and transmission. For images, a slight loss of fidelity is often an acceptable trade off for a much higher degree of compactness.

As one might expect, lossy compression algorithm achieve higher compression ratios. Images transmitted over the Internet are an excellent example of why image compression is so important. To understand the basic concept of compression in mathematical terms, let us consider the following example in which a 1-dimensional signal (the discussion could be extended to 2–dimensional signals as well) is represented as a vector f having N elements, i.e.,

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

Where f_j represents the j^{th} sampled value.

If every element, f_0, f_1, f_2 ... , is coded by b bits, the total number of bits required to represent this digital signal is Nb. Now suppose that the vector f is transformed to another vector F, by a transformation matrix T i.e.,

$$F = Tf$$

Where F is represented by K bits and $K < Nb$. This may be achieved either by reducing N to N' or by reducing b to b'. Thus the amount of data reduction is given by

$$C = \frac{Nb - K}{Nb} \cdot 100\ \% \qquad \ldots(9.2.1)$$

Notice, if $K = Nb$, $C = 0$ i.e., there is no data reduction. We can now reconstruct the digital signal f from F using inverse transformation. Let us see where we are. We have a original signal f which is represented by Nb bits. This vector is transformed to another vector F which is represented by K bits ($K < Nb$). We now reconstruct the digital signal f from F.

Any discrepancy between the original signal f and the new signal $\tilde{f}$ is the error introduced due to the process of compression. Usually amount of error increases with the reduction in the amount of data i.e., error increases as the difference between Nb and K increases. The objective of compression is to achieve maximum C without introducing objectionable error. If the error introduced is zero i.e., $\tilde{f}$ is exactly equal to f, we call it loss-less compression otherwise it is called lossy compression.

Another simple formula to measure compression ratio is

$$CR = \frac{\text{Number of bits in input image}}{\text{Number of bits in output image}} \qquad \ldots(9.2.2)$$

9.3 Error Criteria

Error criteria is required to judge the performance of a lossy compression technique. It simply means how much of error is considered to be tolerable. The error criteria can be classified into two broad groups.

(1) Objective Error (Fidelity) Criteria

(2) Subjective Error (Fidelity) Criteria.

9.3.1 Objective Error Criteria

This is a mathematical formulation to quantify the error introduced in Lossy compression. Let us denote the