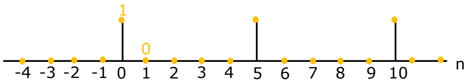
Chapter 5: Neighborhood Processing

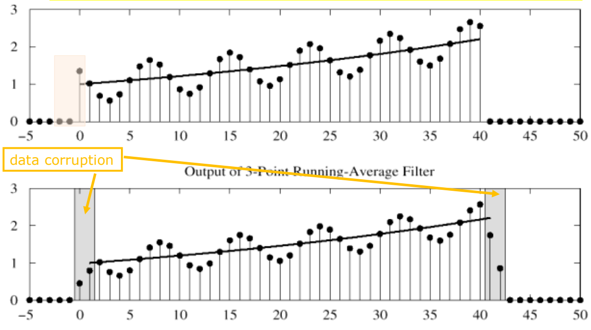
Basic Theory: Math. Notation of Discrete Signal



- discrete signal x[n] = {1 0 0 0 0 1 0 0 0 0 1}
- x[0] = 1, x[1] = 0, x[-1] = 0
- Generalized form of a value of x : x[n]
- If n = 0, x[n-1] = x[-1] = 0.
- If n = 0, x[n+1] = x[1] = 0.

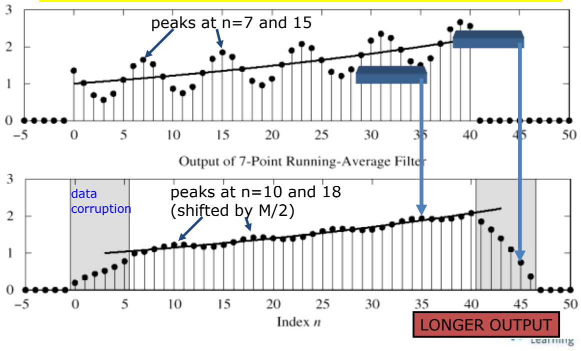
Basic Theory: 3-pt AVG EXAMPLE

$$\text{Input}: x[n] = (1.02)^n + \cos(2\pi n/8 + \pi/4) \quad \text{for } 0 \le n \le 40$$



Basic Theory: 7-pt AVG Example

$$\text{Input}: x[n] = (1.02)^n + \cos(2\pi n/8 + \pi/4) \quad \text{for } 0 \le n \le 40$$



4-point Averaging Input Signal $x[n]$

- $y[n]$: 4-point average value from x[n-3] to x[n]

$$x[n] = \{1,1,1,1,1,1\}$$
$$y[n] = \tfrac{1}{4}(x[n] + x[n-1] + x[n-2] + x[n-3])$$
$$y[0] = \tfrac{1}{4}x[0] + \tfrac{1}{4}x[-1] + \tfrac{1}{4}x[-2] + \tfrac{1}{4}x[-3]$$
$$y[1] = \tfrac{1}{4}x[1] + \tfrac{1}{4}x[0] + \tfrac{1}{4}x[-1] + \tfrac{1}{4}x[-2]$$
.....
$$y[6] = \tfrac{1}{4}x[6] + \tfrac{1}{4}x[5] + \tfrac{1}{4}x[4] + \tfrac{1}{4}x[3]$$
.....
$$y[8] = \tfrac{1}{4}x[8] + \tfrac{1}{4}x[7] + \tfrac{1}{4}x[6] + \tfrac{1}{4}x[5]$$

$$y[n] = \sum_{k=0}^{3} b_k x[n-k]$$

where,

$$b_k = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\}$$

Convolution

- Output signal of the filter, $y[n]$, is obtained by convolution of $x[n]$ and the impulse response of the filter, $h[n]$.

filter length-1

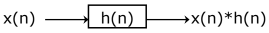
$$y[n] = h[n] * x[n] = \sum_{k=0}^{M} h[k]x[n-k]$$

Same as b_k

Convolution Example 1

- one of the most common methods for *filtering* a function

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$



if n is contant, the equation is the function of k

- x(n) = 1 0 0 0 0 1 0 0 0 0 1, h(n) = 1 2 3 1

  x*h = 1 2 3 1 0 1 2 3 1 0 1 2 3 1
- x(n) = 1 1 1, h(n) = 1 1

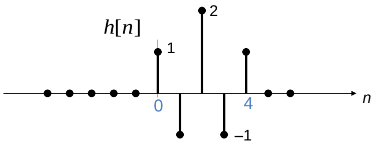
  x*h = 1 2 2 1
- Number of multiplications : $M(M+N-1)$

Convolution Example 2

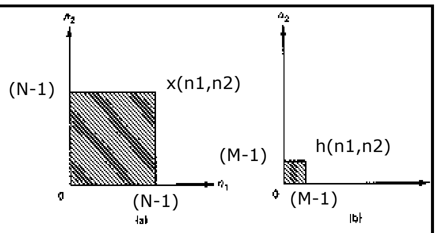
[Q] FIR system has impulse response h[n] as follows.
When x[n]={1,2,3}, determine y[n].

$$b_k = \{1, -1, 2, -1, 1\}$$

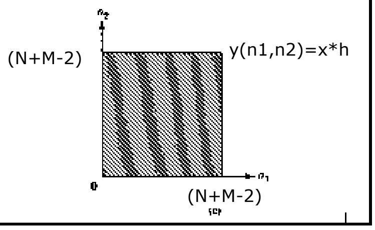
$$h[n] = \delta[n] - \delta[n-1] + 2\delta[n-2] - \delta[n-3] + \delta[n-4]$$



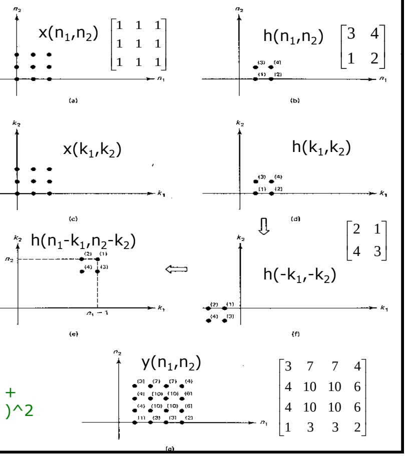
y[n]={ 1 1 3 0 5 -1 3 }



A Typical 2-D Convolution



A 2-D Convolution Example



$$x(n_1,n_2) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad h(n_1,n_2) = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}$$

$$y(n_1,n_2) = \begin{bmatrix} 3 & 7 & 7 & 4 \\ 4 & 10 & 10 & 6 \\ 4 & 10 & 10 & 6 \\ 1 & 3 & 3 & 2 \end{bmatrix}$$

$$y(n_1,n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1,k_2) h(n_1-k_1, n_2-k_2)$$

Number of multiplications :

$$M^2(M+N-1)^2$$

kernel size : MxM

(image size + zeropadding)^2

NP is Nothing But a Convolution.

- Move a **mask**
  - ✓ A rectangle (usually with sides of odd length) or other shape over the given image

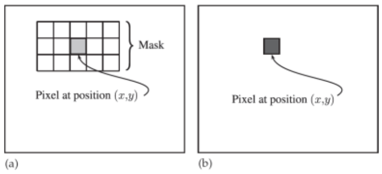


FIGURE 5.1 *Using a spatial mask on an image. (a) Original image. (b) Image after filtering.*

Notation of Masks and Pixel Values

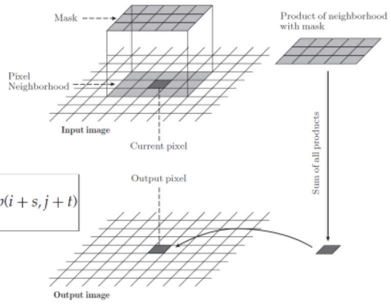
mask values

$m(-1,-2)$	$m(-1,-1)$	$m(-1,0)$	$m(-1,1)$	$m(-1,2)$
$m(0,-2)$	$m(0,-1)$	$m(0,0)$	$m(0,1)$	$m(0,2)$
$m(1,-2)$	$m(1,-1)$	$m(1,0)$	$m(1,1)$	$m(1,2)$

corresponding pixel values

$p(i-1,j-2)$	$p(i-1,j-1)$	$p(i-1,j)$	$p(i-1,j+1)$	$p(i-1,j+2)$
$p(i,j-2)$	$p(i,j-1)$	$p(i,j)$	$p(i,j+1)$	$p(i,j+2)$
$p(i+1,j-2)$	$p(i+1,j-1)$	$p(i+1,j)$	$p(i+1,j+1)$	$p(i+1,j+2)$

NP: How it works ?



$$\sum_{s=-1}^{1}\sum_{t=-2}^{2} m(s,t)p(i+s,j+t)$$

FIGURE 5.2 *Performing spatial filtering.*

CENGAGE Learning

Spatial Filtering: Convolution

- Allied to spatial filtering is spatial **convolution**
  - ✓ The filter must be rotated by 180° (flips in both horizontal and vertical directions) before multiplying and adding

$$\sum_{s=-1}^{1}\sum_{t=-2}^{2} m(-s,-t)p(i+s,j+t)$$

$$\sum_{s=-1}^{1}\sum_{t=-2}^{2} m(s,t)p(i-s+j-t)$$

CENGAGE Learning

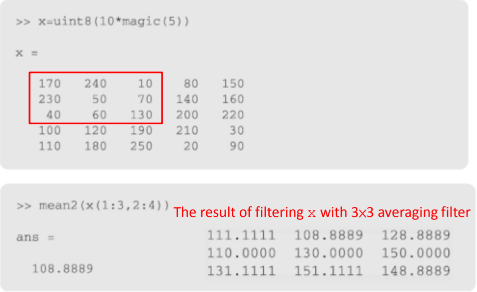
Spatial Filtering

- **EXAMPLE** One important linear filter is to use a 3x3 mask and take the average of all nine values within the mask

$$\begin{array}{|c|c|c|} \hline a & b & c \\ \hline d & e & f \\ \hline g & h & i \\ \hline \end{array} \longrightarrow \frac{1}{9}(a+b+c+d+e+f+g+h+i)$$

CENGAGE Learning

Spatial Filtering



```
>> x=uint8(10*magic(5))

x =

   170   240    10    80   150
   230    50    70   140   160
    40    60   130   200   220
   100   120   190   210    30
   110   180   250    20    90
```

```
>> mean2(x(1:3,2:4))

ans =

  108.8889
```

The result of filtering x with 3x3 averaging filter

```
111.1111   108.8889   128.8889
110.0000   130.0000   150.0000
131.1111   151.1111   148.8889
```

CENGAGE Learning

Spatial Filtering

- It is convenient to describe a linear filter simply in terms of the coefficients of all the gray values of pixels within the mask
  - ✓ The averaging filter

$$\frac{1}{9}a+\frac{1}{9}b+\frac{1}{9}c+\frac{1}{9}d+\frac{1}{9}e+\frac{1}{9}f+\frac{1}{9}g+\frac{1}{9}h+\frac{1}{9}i$$

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

CENGAGE Learning

Spatial Filtering

✓ **EXAMPLE** The filter

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

would operate on gray values as

$$\begin{array}{|c|c|c|} \hline a & b & c \\ \hline d & e & f \\ \hline g & h & i \\ \hline \end{array} \longrightarrow a-2b+c-2d+4e-2f+g-2h+i$$

CENGAGE Learning

Edges of the Image

- What happens at the edge of the image, where the mask partly falls outside the image?

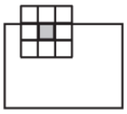
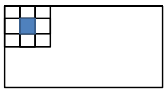


FIGURE 5.3 *A mask at the edge of an image.*

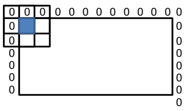
- There are a number of different approaches to dealing with this problem

Edges of the Image

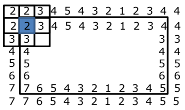
- **Ignore the edges**



- **Pad with zeros**

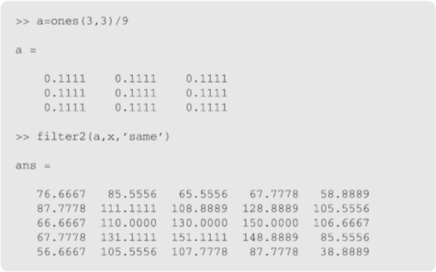


- **Mirroring**



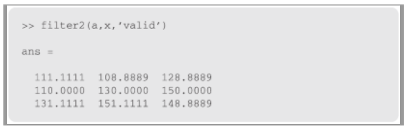
5.3 Filtering in MATLAB

- `filter2(filter,image,shape)` The result is a matrix of data type double!!
- `shape` is optional and describes the method for dealing with edges.
  - ✓ `'same'` –**pad with zeros**
  - ✓ `'valid'` –**ignore the edges**



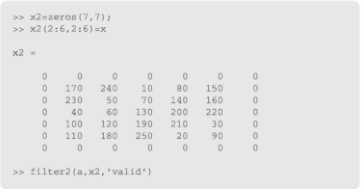
```
>> a=ones(3,3)/9

a =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>> filter2(a,x,'same')

ans =

   76.6667   85.5556   65.5556   67.7778   58.8889
   87.7778  111.1111  108.8889  128.8889  105.5556
   66.6667  110.0000  130.0000  150.0000  106.6667
   67.7778  131.1111  151.1111  148.8889   85.5556
   56.6667  105.5556  107.7778   87.7778   38.8889
```

5.3 Filtering in MATLAB



```
>> filter2(a,x,'valid')

ans =

  111.1111  108.8889  128.8889
  110.0000  130.0000  150.0000
  131.1111  151.1111  148.8889
```

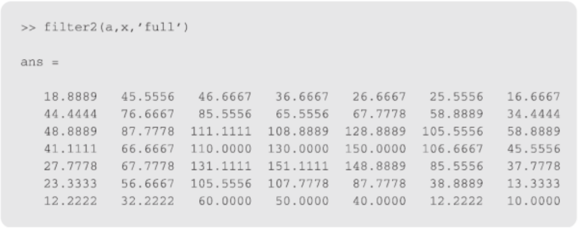
- The result of `'same'` may also be obtained by padding with zeros and using `'valid'`:



```
>> x2=zeros(7,7);
>> x2(2:6,2:6)=x

x2 =

     0     0     0     0     0     0     0
     0   170   240    10    80   150     0
     0   230    50    70   140   160     0
     0    40    60   130   200   220     0
     0   100   120   190   210    30     0
     0   110   180   250    20    90     0
     0     0     0     0     0     0     0

>> filter2(a,x2,'valid')
```

5.3 Filtering in MATLAB

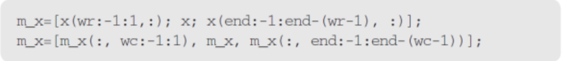
- `filter2(filter,image,'`**`full`**`')` returns a result larger than the original.
- It does this by padding with zero and applying the filter at all places on and around the image where the mask intersects the image matrix.



```
>> filter2(a,x,'full')

ans =

   18.8889   45.5556   46.6667   36.6667   26.6667   25.5556   16.6667
   44.4444   76.6667   85.5556   65.5556   67.7778   58.8889   34.4444
   48.8889   87.7778  111.1111  108.8889  128.8889  105.5556   58.8889
   41.1111   66.6667  110.0000  130.0000  150.0000  106.6667   45.5556
   27.7778   67.7778  131.1111  151.1111  148.8889   85.5556   37.7778
   23.3333   56.6667  105.5556  107.7778   87.7778   38.8889   13.3333
   12.2222   32.2222   60.0000   50.0000   40.0000   12.2222   10.0000
```

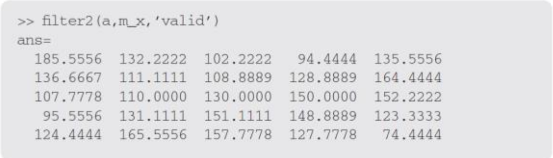
5.3 Filtering in MATLAB

- `filter2` provides no mirroring option
- The mirroring approach can be realized by placing the following codes before `filter2 (filter,image,'valid')`

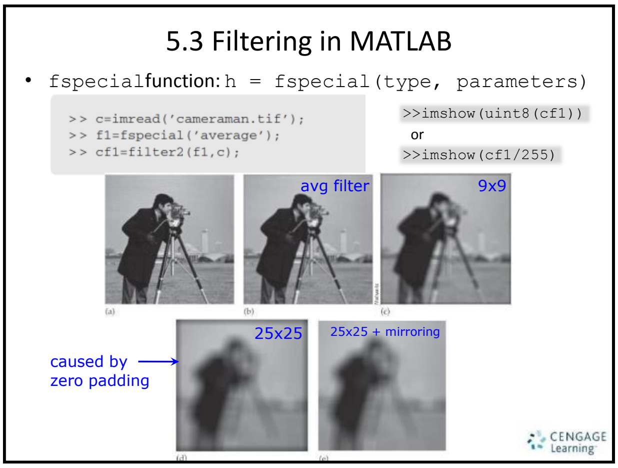


```
m_x=[x(wr:-1:1,:); x; x(end:-1:end-(wr-1), :)];
m_x=[m_x(:, wc:-1:1), m_x, m_x(:, end:-1:end-(wc-1))];
```

- Where matrix `x` is extended to `m_x`, `wr/wc` is defined as one half total column/row number of the mask (chopping the decimal)



```
>> filter2(a,m_x,'valid')
ans=
  185.5556  132.2222  102.2222   94.4444  135.5556
  136.6667  111.1111  108.8889  128.8889  164.4444
  107.7778  110.0000  130.0000  150.0000  152.2222
   95.5556  131.1111  151.1111  148.8889  123.3333
  124.4444  165.5556  157.7778  127.7778   74.4444
```



5.3 Filtering in MATLAB

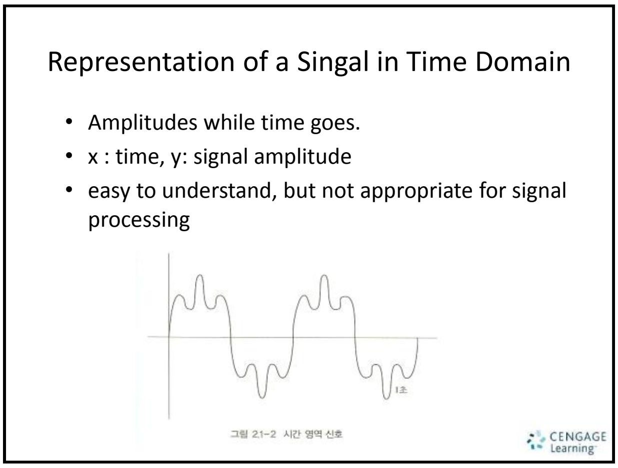
- `fspecial`function: `h = fspecial(type, parameters)`

```
>> c=imread('cameraman.tif');
>> f1=fspecial('average');
>> cf1=filter2(f1,c);
```

```
>>imshow(uint8(cf1))
```

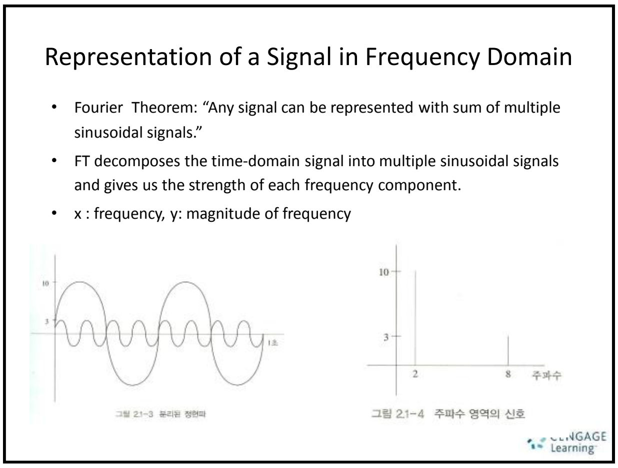
or

```
>>imshow(cf1/255)
```



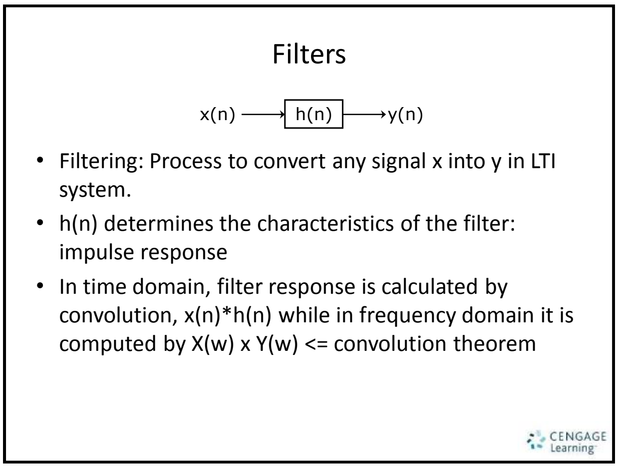
Representation of a Singal in Time Domain

- Amplitudes while time goes.
- x : time, y: signal amplitude
- easy to understand, but not appropriate for signal processing



Representation of a Signal in Frequency Domain

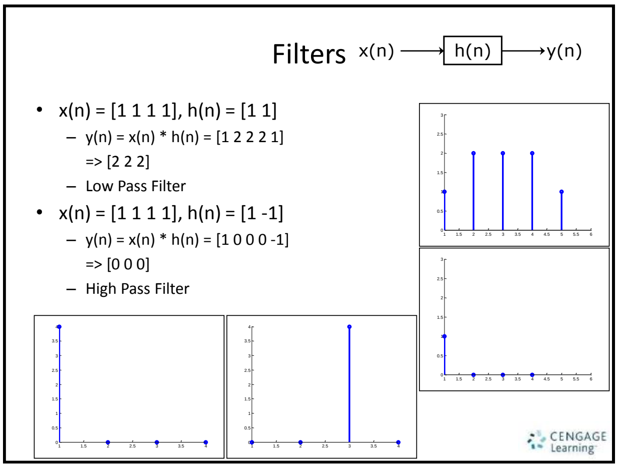
- Fourier Theorem: "Any signal can be represented with sum of multiple sinusoidal signals."
- FT decomposes the time-domain signal into multiple sinusoidal signals and gives us the strength of each frequency component.
- x : frequency, y: magnitude of frequency



Filters

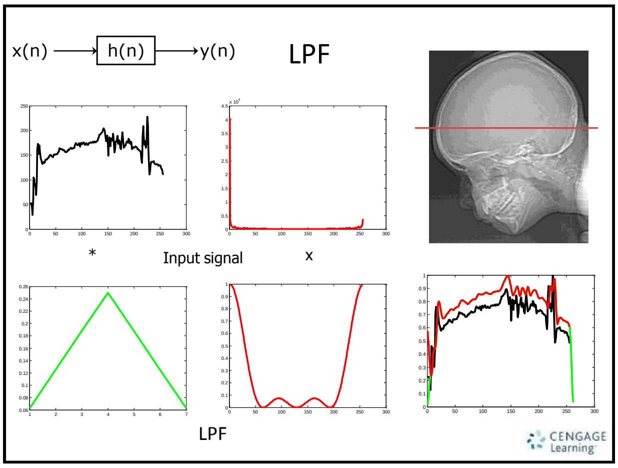
$x(n) \rightarrow \boxed{h(n)} \rightarrow y(n)$

- Filtering: Process to convert any signal x into y in LTI system.
- h(n) determines the characteristics of the filter: impulse response
- In time domain, filter response is calculated by convolution, x(n)*h(n) while in frequency domain it is computed by X(w) x Y(w) <= convolution theorem



Filters $x(n) \rightarrow \boxed{h(n)} \rightarrow y(n)$

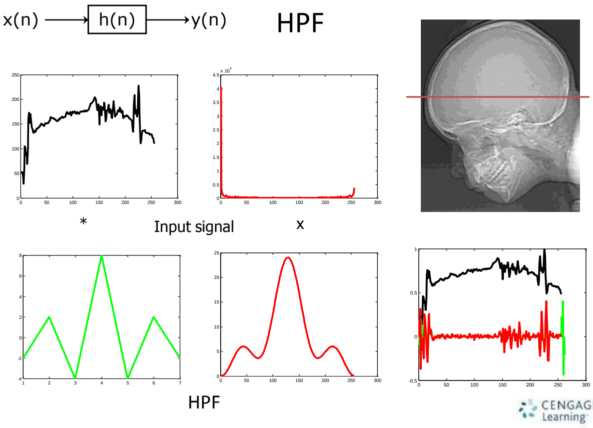
- x(n) = [1 1 1 1], h(n) = [1 1]
  - y(n) = x(n) * h(n) = [1 2 2 2 1]
    => [2 2 2]
  - Low Pass Filter
- x(n) = [1 1 1 1], h(n) = [1 -1]
  - y(n) = x(n) * h(n) = [1 0 0 0 -1]
    => [0 0 0]
  - High Pass Filter



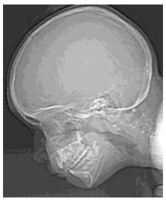
$x(n) \rightarrow \boxed{h(n)} \rightarrow y(n)$ **LPF**

$x(n) \longrightarrow \boxed{h(n)} \longrightarrow y(n)$

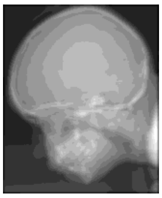
HPF



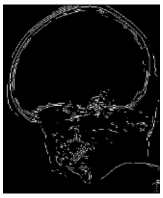
LPF & HPF



ORG.



LPF



HPF

5.4 Frequencies: Low- and High-Pass Filters

- Frequencies of an image are a measure of the amount by which gray values change with distance

✓ **high-pass filter** $\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$

✓ **low-pass filter** $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Lowpass Filtering for Noise Smoothing

- Most image energy : e,g, bit plane slicing
  -> is in its **low frequency components** (high spatial correlation among neighboring pixels),
- Most image noise
  -> **wideband** (e.g., white Gaussian noise)
  -> lowpass filtering (LPF) required
  (it cannot remove the noise in the low frequency range though !).
- LPFs cause **blurring**
  - critical for some images with sharp edges
  - not critical for images of smooth contrast.
- **Another use:** remove blocky effect (e.g., 8 by 8 DCT based coding) created in lossy image coding.

Some Typical Filter Masks of LPFs:

- Averaging LPFs:

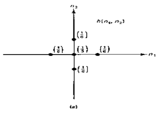
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- Some Other (Gaussian) LPFs:

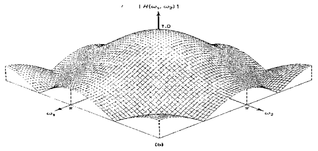
$$\frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

A Lowpass Filter (= Averaging Filter)

$$\frac{1}{9}\times\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \qquad \frac{1}{16}\times\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$



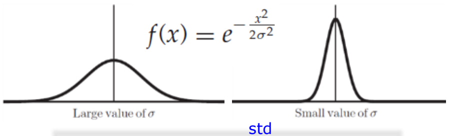
$$\begin{bmatrix} 0 & 1/6 & 0 \\ 1/6 & 1/3 & 1/6 \\ 0 & 1/6 & 0 \end{bmatrix}$$



Weighted Averaging Filter

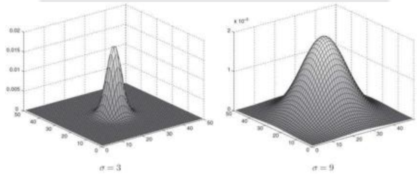
Gaussian Lowpass Filters in Matlab

$$f(x) = e^{-\frac{x^2}{2\sigma^2}}$$



std

```
>> a=50;s=3;
>> g=fspecial('gaussian',[a a],s);
>> surf(1:a,1:a,g)
>> s=9;
>> g2=fspecial('gaussian',[a a],s);
>> figure,surf(1:a,1:a,g2)
```

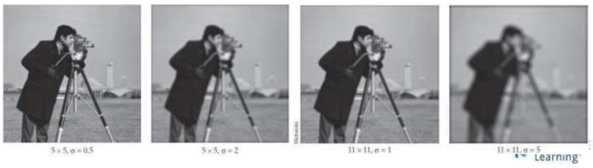


CENGAGE Learning

Gaussian Lowpass Filters

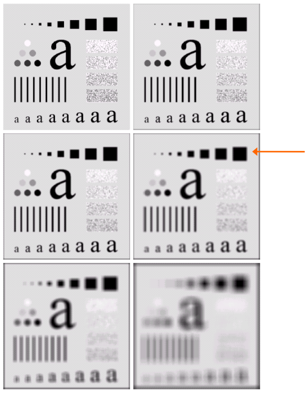
```
>> g1=fspecial('gaussian',[5,5]);
>> g2=fspecial('gaussian',[5,5],2);
>> g3=fspecial('gaussian',[11,11],1);
>> g4=fspecial('gaussian',[11,11],5);
```

```
>> imshow(filter2(g1,c)/256)
>> figure,imshow(filter2(g2,c)/256)
>> figure,imshow(filter2(g3,c)/256)
>> figure,imshow(filter2(g4,c)/256)
```



Learning

original image



Low-pass Filtering

mask size :
3x3, 5x5, 9x9, 15x15,
35x35

square size :
3x3, 5x5, 9x9, 15x15,
25x25, 35x35, 45x45,
55x55

CENGAGE Learning

Lowpass Filtering : Noise Reduction

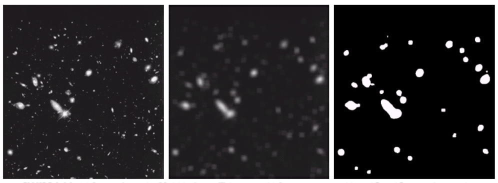
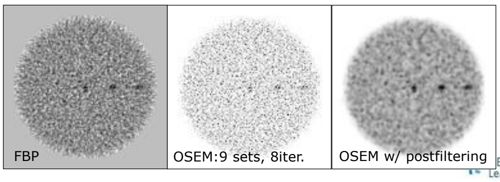


FIGURE 3.36 (a) Image from the Hubble Space Telescope. (b) Image processed by a 15 × 15 averaging mask. (c) Result of thresholding (b). (Original image courtesy of NASA.)



CENGAGE Learning

Highpass Filter : Derivative Filter

Averaging (=blurring=LPF) : analogous to "integration(적분)"

sharpening(=HPF) : analogous to "differentiation(미분)"

1st derivative (= gradient) of an image $x(t_1,t_2)$

$$\nabla_x = \begin{bmatrix} \frac{\partial x(t_1,t_2)}{\partial t_1} \\ \frac{\partial x(t_1,t_2)}{\partial t_2} \end{bmatrix}$$

두 점 간의 변화율 또는 기울기 -> 1차 편미분

2nd derivative (= Laplacian) of an image $x(t_1,t_2)$

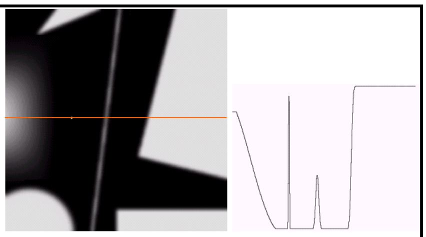
$$\nabla_x^2 = \begin{bmatrix} \frac{\partial^2 x(t_1,t_2)}{\partial t_1^2} \\ \frac{\partial^2 x(t_1,t_2)}{\partial t_2^2} \end{bmatrix}$$

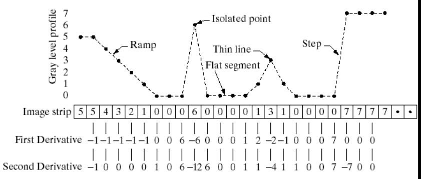
두 점 간의 변화율 또는 기울기 -> 2차 편미분

1,2차 편미분을 두 점간의 차(difference)의 연산으로 단순화 시킬 수 있다. (See the figure in next slide)

CENGAGE Learning

High-pass Filter : Derivative Filter





1st and 2nd derivatives represent simplified edge information.

Edge Detection by Gradient (1st derivative)

- **Edges:** the image portions which have large gradients

$x(n_1,n_2)$ magnitude of gradient

$$\nabla_x(n_1,n_2) = mag(\nabla_x(n_1,n_2)) = \sqrt{\left[\frac{\partial x(n_1,n_2)}{\partial n_1}\right]^2 + \left[\frac{\partial x(n_1,n_2)}{\partial n_2}\right]^2}$$

$$\nabla_x(n_1,n_2) \approx \left|\frac{\partial x(n_1,n_2)}{\partial n_1}\right| + \left|\frac{\partial x(n_1,n_2)}{\partial n_2}\right|$$

You can apply the directional derivative edge detector only or apply two directional detectors together and sum their square magnitudes to get the omni-directional (isotropic) edges.

1st Derivative (gradient) Filters : Prewitt

- Typically, we want to average over several neighboring rows, i.e.,

$$\frac{\partial x(t_1,t_2)}{\partial t_1} \approx [x(n_1+1,n_2-1) - x(n_1-1,n_2-1)] + [x(n_1+1,n_2) - x(n_1-1,n_2)] + [x(n_1+1,n_2+1) - x(n_1-1,n_2+1)]$$

- These result in **Prewitt** operators (directional filters):

gradient for y direction $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ gradient for x direction

1st Derivative (gradient) Filters : Sobel

- How about emphasize the center row more:

$$\frac{\partial x(t_1,t_2)}{\partial t_1} \approx [x(n_1+1,n_2-1) - x(n_1-1,n_2-1)] + 2[x(n_1+1,n_2) - x(n_1-1,n_2)] + [x(n_1+1,n_2+1) - x(n_1-1,n_2+1)]$$

- These result in **Sobel** operators:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

1st Derivative (gradient) Filters : Robert

- We can also deal with both derivatives simultaneously, i.e. emphasize diagonal edge information.:

$$\frac{\partial x(t_1,t_2)}{\partial t_1} + \frac{\partial x(t_1,t_2)}{\partial t_2} \approx [x(n_1,n_2) - x(n_1-1,n_2-1)]$$

- This results in **Robert** operators:

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

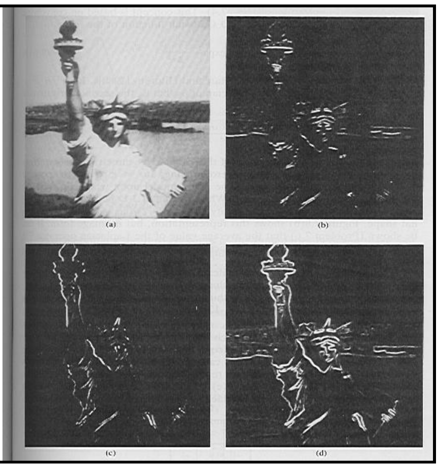
Edge Extraction via Gradients

(a) original

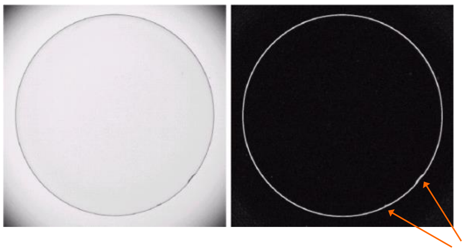
(b) vertical Sobel filtering

(c) horizontal Sobel filter

(d) magnitude of gradients



An Application for the Edge Detection



Optical image of contact lens

After Sobel filtering

Detected defects

Laplacian Edge Detectors: 2nd Derivative

$$\nabla_x^2(n_1,n_2) \cong \frac{\partial^2 x(n_1,n_2)}{\partial n_1^2} + \frac{\partial^2 x(n_1,n_2)}{\partial n_2^2}$$

Difference Eq. Approximation of Laplacian:

$$\frac{\partial x(n_1,n_2)}{\partial n_1} \approx x(n_1+1,n_2) - x(n_1,n_2)$$

$$\frac{\partial^2 x(n_1,n_2)}{\partial n^2} \approx \left[x(n_1+1,n_2) - x(n_1,n_2)\right] - \left[x(n_1,n_2) - x(n_1-1,n_2)\right]$$

$$= x(n_1+1,n_2) - 2x(n_1,n_2) + x(n_1-1,n_2)$$

$$\nabla_x^2(n_1,n_2) \cong x(n_1+1,n_2) + x(n_1-1,n_2) + x(n_1,n_2+1) + x(n_1,n_2-1) - 4x(n_1,n_2)$$

Isotropic (omni directional) edge detection

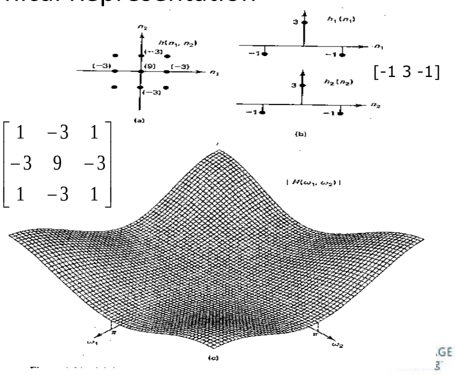
Some Laplacian Operators

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

- Note that: The edges are indicated by **zero crossing,** instead of **large gradients** (1st derivative).
- **Detecting the zero crossing:** low operated values with large variance in a local 5-by-5 window.

$$\sigma_x^2 = \frac{1}{(2M+1)^2} \sum_{i=n_1-M}^{n_1+M} \sum_{j=n_2-M}^{n_2+M} \left[x(i,j) - \bar{x}(n_1,n_2)\right]^2$$

A Laplacian Filter Mask : Graphical Representation

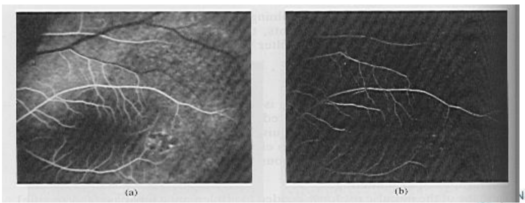


[-1 3 -1]

$$\begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -3 & 1 \\ -3 & 9 & -3 \\ 1 & -3 & 1 \end{bmatrix}$$

HPF : Laplacian Filter Mask

- **Goal:** highlight or enhance fine details in an image.
- A **basic highpass spatial filtering**: $\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
- Note that the sum of coefficients is **0**! The DC term of H(w1,w2) is zero, i.e., the global contrast of the images is reduced.
- Results need scaling or clipping to remain in [0,L-1].



Summary

- **Chap 4. Point Processing**
  - ✓ Histogram(Contrast) Stretching
  - ✓ Piecewise Contrast Stretching
  - ✓ Histogram Equalization & Specification
- **Chap 5. Neighborhood Processing**
  - ✓ Convolution for Spatial Filtering
  - ✓ Lowpass Filtering: Gaussian filter
  - ✓ Highpass Filtering: Laplacian, LoG filter
  - ✓ Edge Sharpening by Unsharp Masking
  - ✓ High Boost Filtering
  - ✓ Nonlinear Filtering: median, min, max