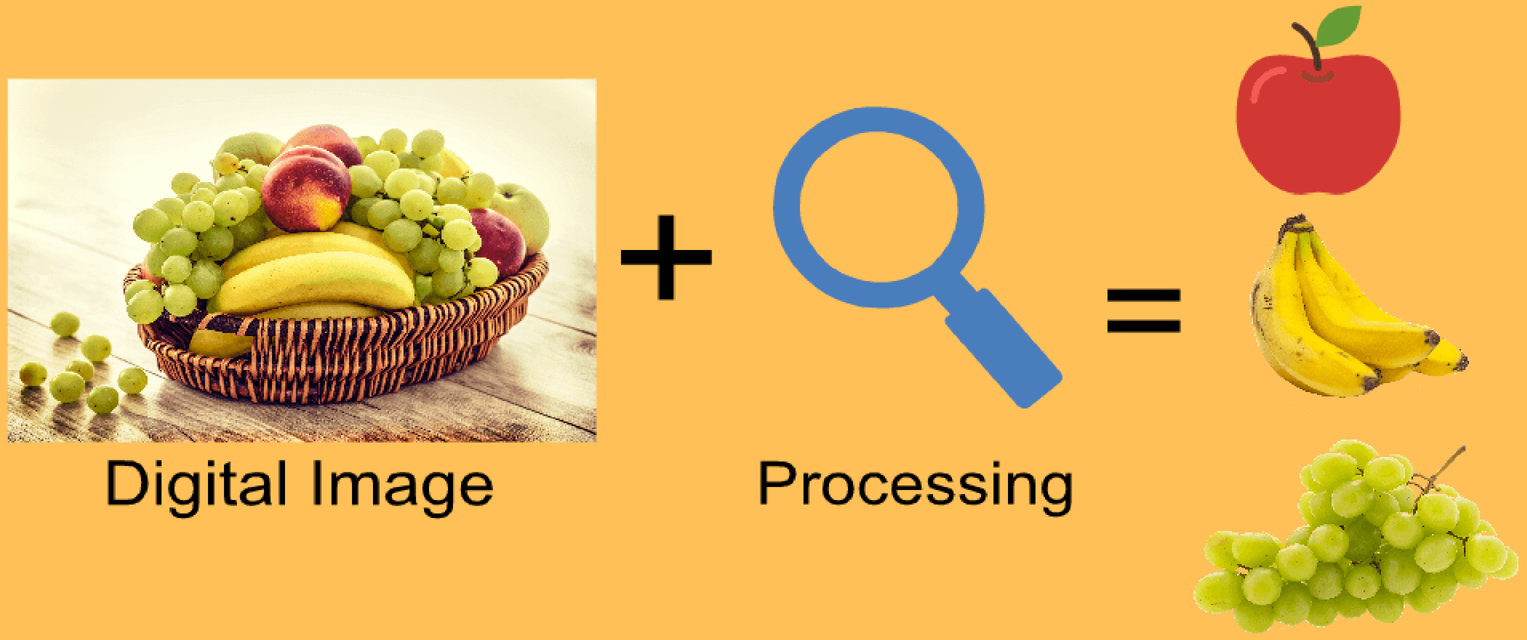
Digital Image Processing



Digital Image

Processing

Apple

Banana

Grape

Digital Image Processing

Lec. 3: Image Filtering

Assist. Prof. Dr. Saad Albawi

Three Views of Filtering :

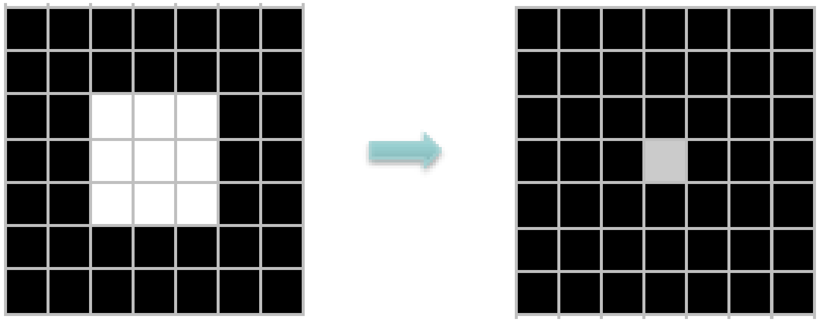
- Image filters in spatial domain
  - Filter is a mathematical operation of a grid of numbers.
  - Smoothing, sharpening, measuring texture.

- Image filters in the frequency domain
  - Filtering is a way to modify the frequencies of images
  - Denoising, sampling, image compression

- Image pyramids
  - Scale-space representation allows coarse-to-fine operations.

Neighborhood processing

- Define a reference point in the input image, $f(x_0, y_0)$.
- Perform an operation that involves only pixels within a neighborhood around the reference point in the input image.
- Apply the result of that operation to the pixel of same coordinates in the output image, $g(x_0, y_0)$.
- Repeat the process for every pixel in the input image.



Neighborhood processing

- **Linear & shift-invariant (LTI) filters**: the resulting output pixel is computed using a weighted average of neighboring pixel values with a fixed kernel.
- **Linear & locally adaptive filters**: the resulting output pixel is computed using a weighted average of neighboring pixel values where the kernel weights may vary depending on the pixel location.
- **Nonlinear filters**: the resulting output pixel is computed via a nonlinear combination of neighboring pixel values.

10.2.1 Convolution in the One-Dimensional Domain

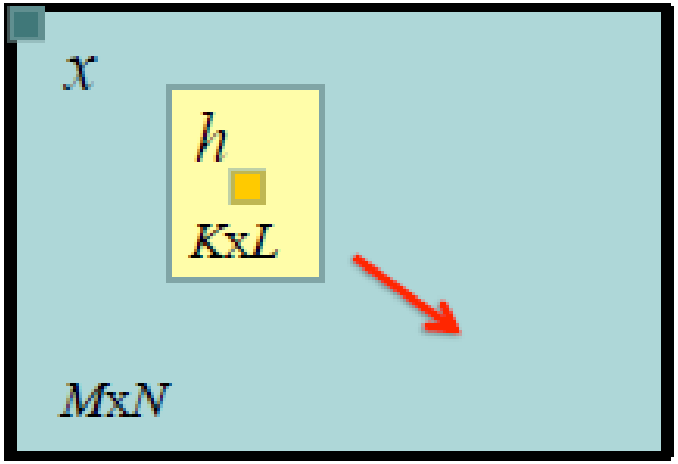
The convolution between two discrete one-dimensional (1D) arrays $A(x)$ and $B(x)$, denoted by $A * B$, is mathematically described by the equation

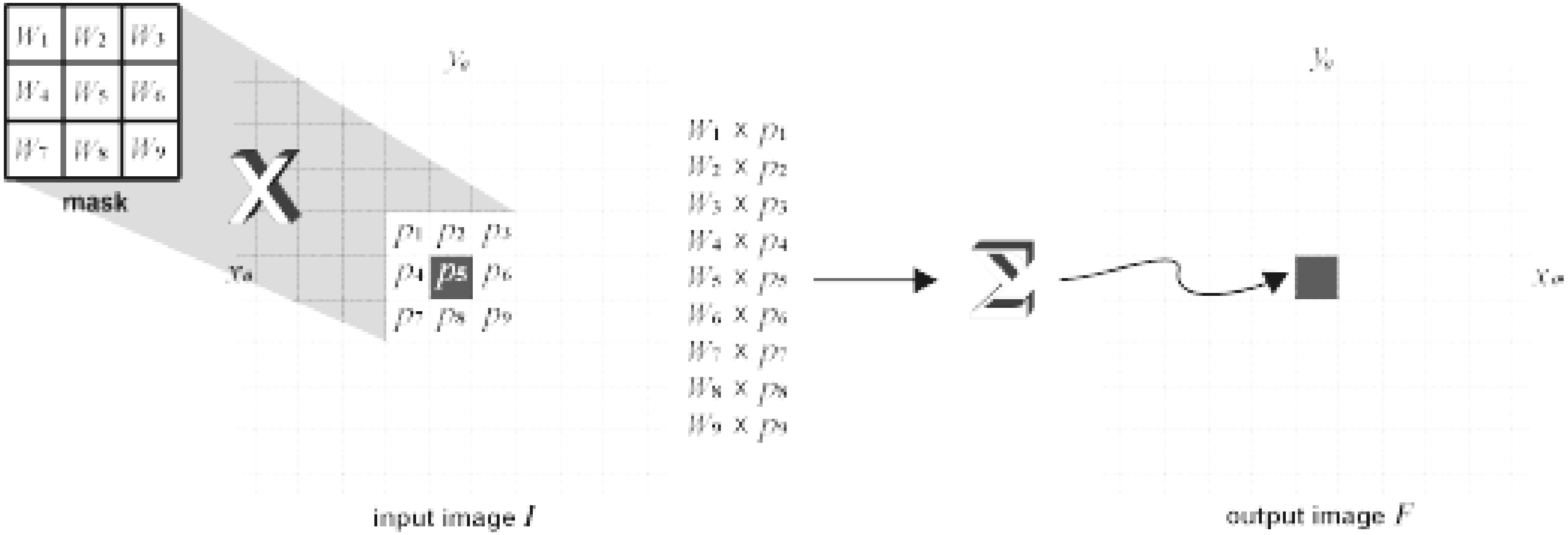
$$A * B = \sum_{j=-\infty}^{\infty} A(j) \cdot B(x - j) \tag{10.1}$$

[1] We shall see additional examples of neighborhood operations for different purposes—for example, image restoration (Chapter 12) and edge detection (Chapter 14)—later in the book.

2D Convolution (LTI Filtering)

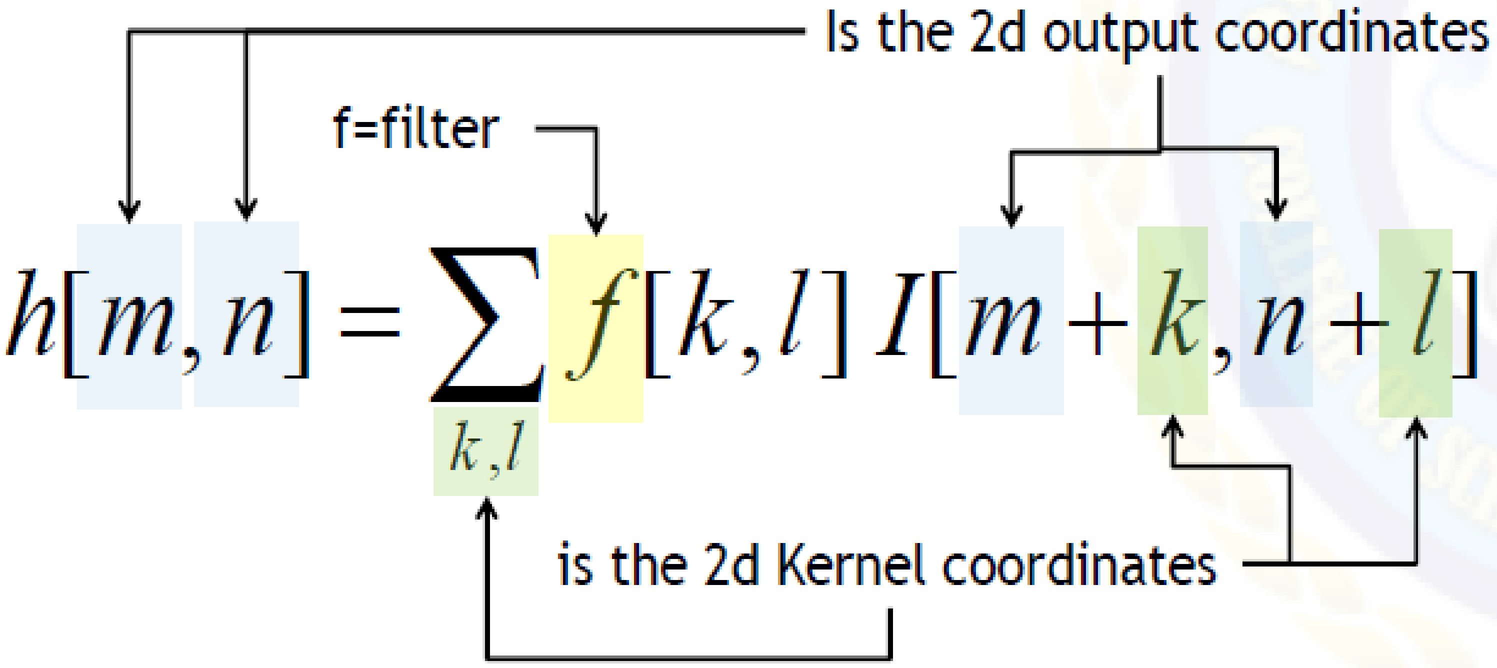
$$y[m,n] = \sum_i \sum_j h[m-i, n-j]\,x[i,j]$$





Three Views of Filtering :

- Image filtering:
  - Compute function of local neighborhood at each position.



Is the 2d output coordinates

f=filter

$$h[m,n]=\sum_{k,l} f[k,l]\,I[m+k,n+l]$$

is the 2d Kernel coordinates

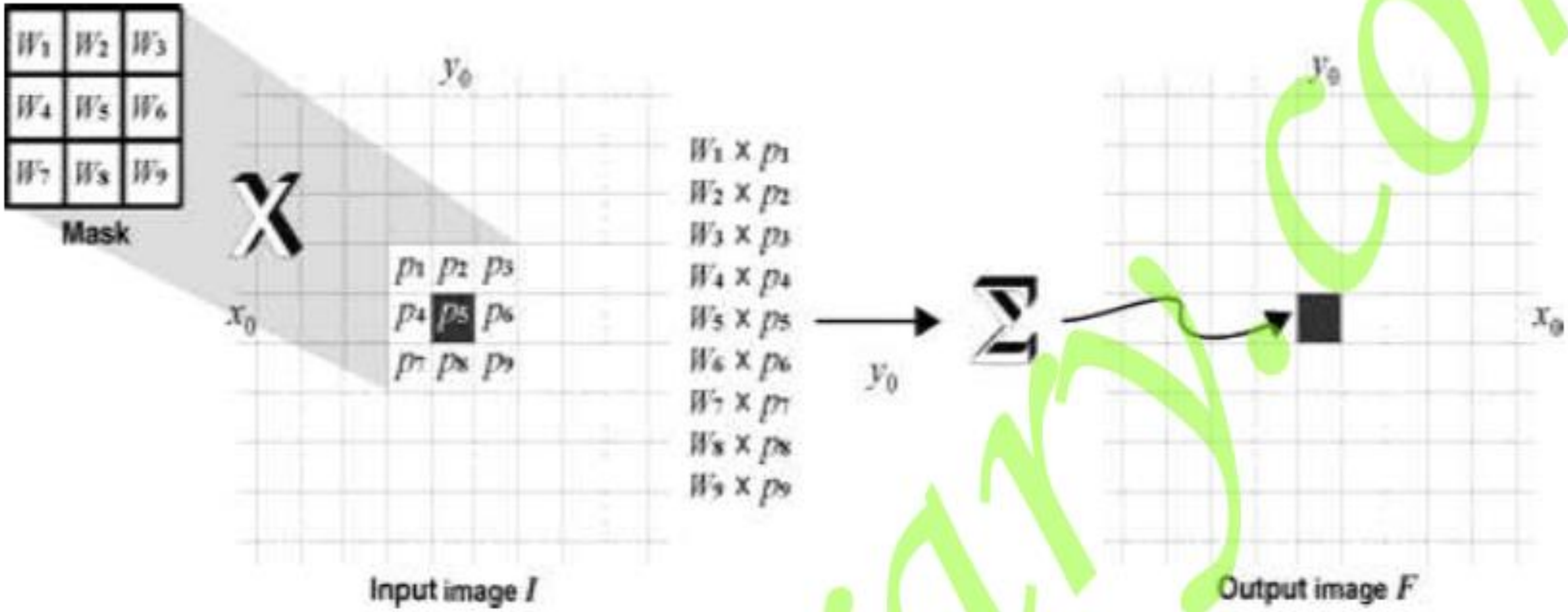
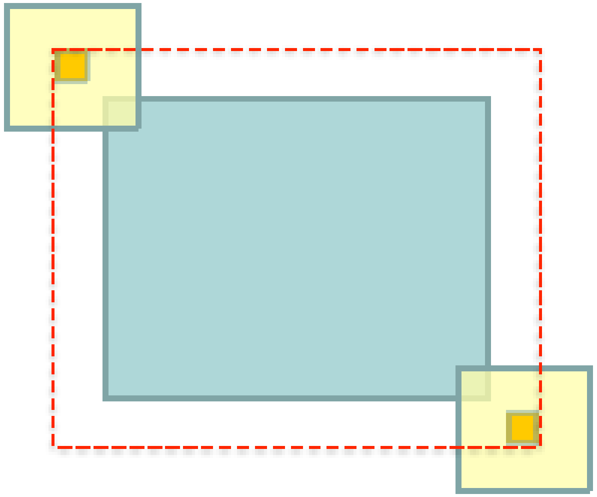
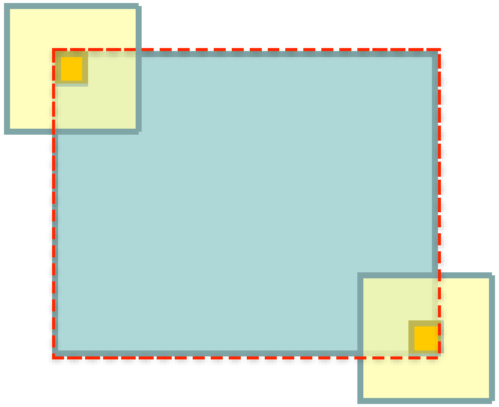


FIGURE 10.1 Neighborhood processing for the case of linear filtering.

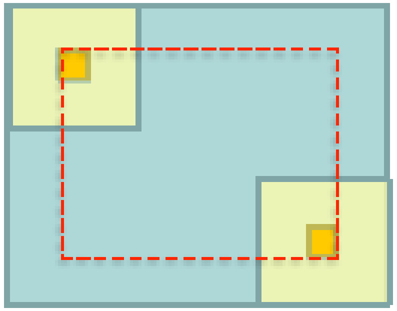
Output Image Size Options



$(M+K-1)\text{x}(N+L-1)$



$M\text{x}N$



$(M-K+1)\text{x}(N-L+1)$

L EXAMPLE 10.1

In this example, we show how the result of a 1D convolution operation can be obtained step-by-step. Let $A = \{0, 1, 2, 3, 2, 1, 0\}$ and $B = \{1, 3, -1\}$. The partial results of multiplying elements in A with corresponding elements in B, as B shifts from $-\infty$ to ∞, are displayed below.

1. Initially, we mirror array B and align its center (reference) value with the first (leftmost) value of array A.[2] The partial result of the convolution calculation $(0 \times (-1)) + (0 \times 3) + (1 \times 1) = 1$ (where empty spots are assumed as zero) is stored in the resulting array $(A * B)$.

A		0	1	2	3	2	1	0
B	-1	3	1					
$A * B$		1						

2. Array B is shifted one position to the right. The partial result of the convolution calculation $(0 \times (-1)) + (1 \times 3) + (2 \times 1) = 5$ is stored in the resulting array $(A * B)$.

A	0	1	2	3	2	1	0
B	−1	3	1				
$A * B$	1	5					

3. Array B is shifted another position to the right. The partial result of the convolution calculation $(1 \times (-1)) + (2 \times 3) + (3 \times 1) = 8$ is stored in the resulting array $(A * B)$.

A	0	1	2	3	2	1	0
B		−1	3	1			
$A * B$	1	5	8				

4. Array B is shifted another position to the right. The partial result of the convolution calculation $(2 \times (-1)) + (3 \times 3) + (2 \times 1) = 9$ is stored in the resulting array $(A * B)$.

A	0	1	2	3	2	1	0
B			−1	3	1		
$A * B$	1	5	8	8			

5. Array B is shifted another position to the right. The partial result of the convolution calculation $(3 \times (-1)) + (2 \times 3) + (1 \times 1) = 4$ is stored in the resulting array $(A * B)$.

A	0	1	2	3	2	1	0
B				−1	3	1	
$A * B$	1	5	8	8	4		

6. Array B is shifted another position to the right. The partial result of the convolution calculation $(2 \times (-1)) + (1 \times 3) + (0 \times 1) = 1$ is stored in the resulting array $(A * B)$.

A	0	1	2	3	2	1	0
B					−1	3	1
$A * B$	1	5	8	8	4	1	

10.2.2 Convolution in the Two-Dimensional Domain

The mathematical definition for 2D convolution is

$$g(x, y) = \sum_{k=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(j, k) \cdot f(x - j, y - k) \tag{10.2}$$

In practice, this is rewritten as

$$g(x, y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j, k) \cdot f(x - j, y - k) \tag{10.3}$$

where m_2 is equal to half of the mask's width and n_2 is equal to half of the mask's height, that is,

$$m_2 = \lfloor m/2 \rfloor \tag{10.4}$$

and

$$n_2 = \lfloor n/2 \rfloor \tag{10.5}$$

where $\lfloor x \rfloor$ is the *floor* operator, which rounds a number to the nearest integer less than or equal to x.

Example: Box Filter:

- Image filtering:
  - Compute function of local neighborhood at each position.

$$f[\cdot,\cdot]$$

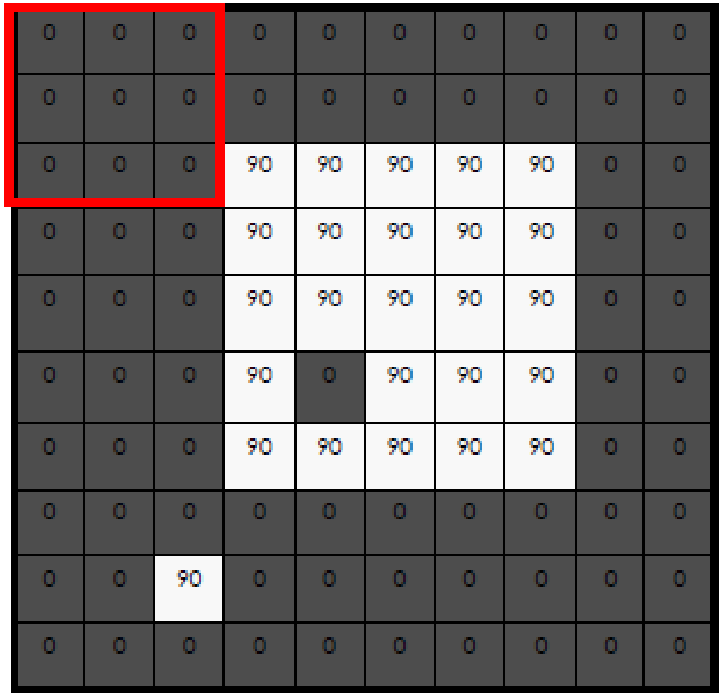
$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Image filtering

$f[\cdot\,,\cdot\,]\ \frac{1}{9}$

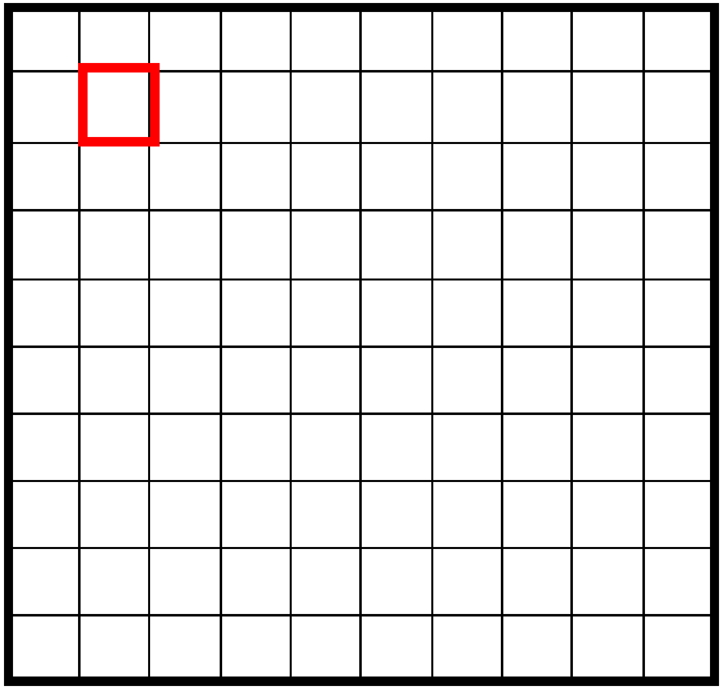
1	1	1
1	1	1
1	1	1

$I[.,.]$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$



$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

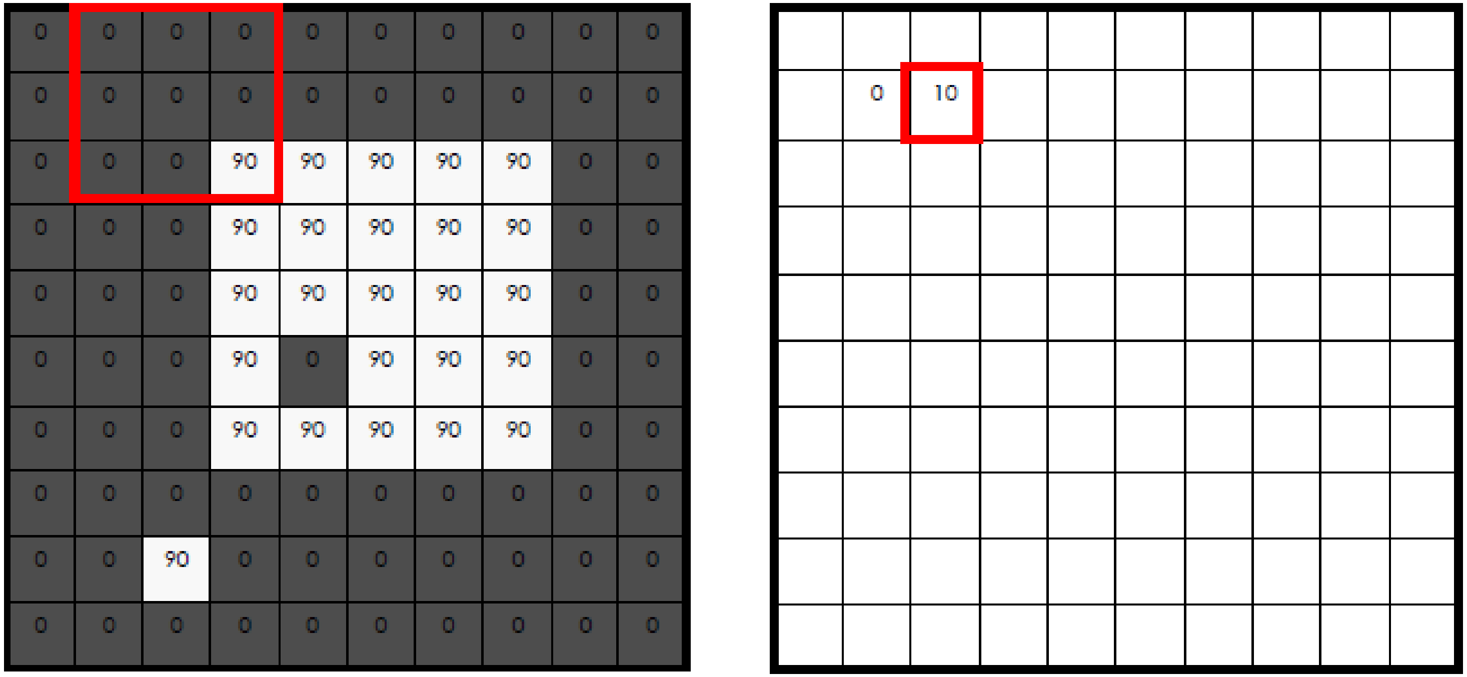
Credit: S. Seitz

Image filtering

$f[\cdot\,,\cdot\,]\,\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[.,.]$ $h[.,.]$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10							

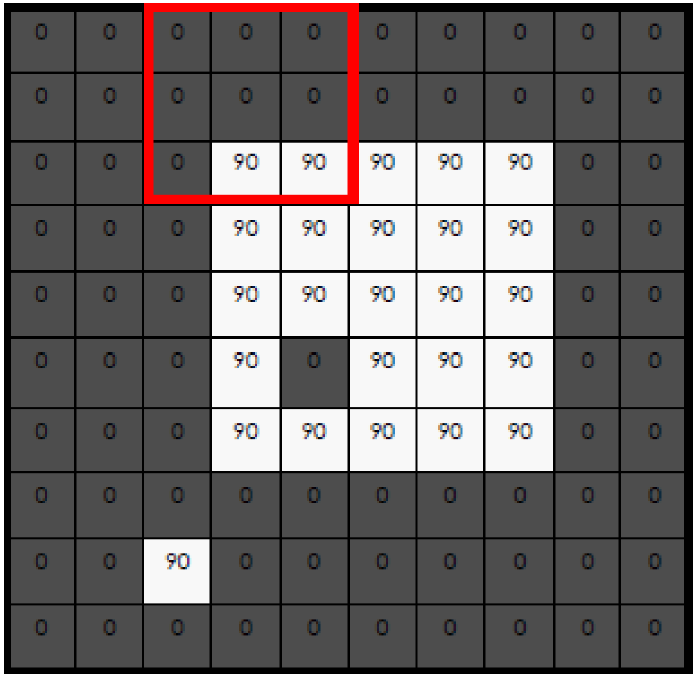
$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

Image filtering

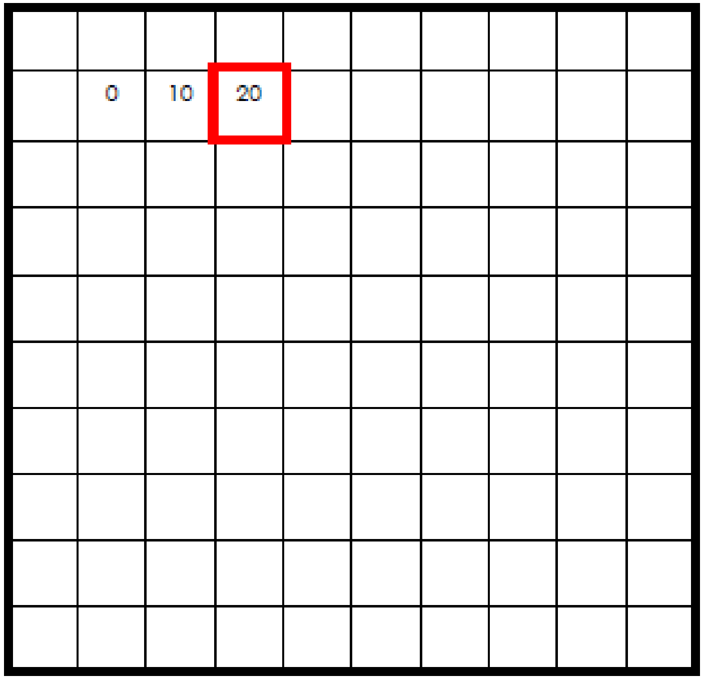
$$f[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$I[.,.]$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$



	0	10	20						

$$h[m,n]=\sum_{k,l} f[k,l]\, I[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$f[\cdot\,,\cdot\,] \quad \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$I[.,.]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$h[.,.]$$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

∟ EXAMPLE 10.2

Let

$$A = \begin{bmatrix} 5 & 8 & 3 & 4 & 6 & 2 & 3 & 7 \\ 3 & 2 & 1 & 1 & 9 & 5 & 1 & 0 \\ 0 & 9 & 5 & 3 & 0 & 4 & 8 & 3 \\ 4 & 2 & 7 & 2 & 1 & 9 & 0 & 6 \\ 9 & 7 & 9 & 8 & 0 & 4 & 2 & 4 \\ 5 & 2 & 1 & 8 & 4 & 1 & 0 & 9 \\ 1 & 8 & 5 & 4 & 9 & 2 & 3 & 8 \\ 3 & 7 & 1 & 2 & 3 & 4 & 4 & 6 \end{bmatrix}$$

and

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

The result of the convolution $A * B$ will be

$$A * B = \begin{bmatrix} 20 & 10 & 2 & 26 & 23 & 6 & 9 & 4 \\ 18 & 1 & -8 & 2 & 7 & 3 & 3 & -11 \\ 14 & 22 & 5 & -1 & 9 & -2 & 8 & -1 \\ 29 & 21 & 9 & -9 & 10 & 12 & -9 & -9 \\ 21 & 1 & 16 & -1 & -3 & -4 & 2 & 5 \\ 15 & -9 & -3 & 7 & -6 & 1 & 17 & 9 \\ 21 & 9 & 1 & 6 & -2 & -1 & 23 & 2 \\ 9 & -5 & -25 & -10 & -12 & -15 & -1 & -12 \end{bmatrix}$$

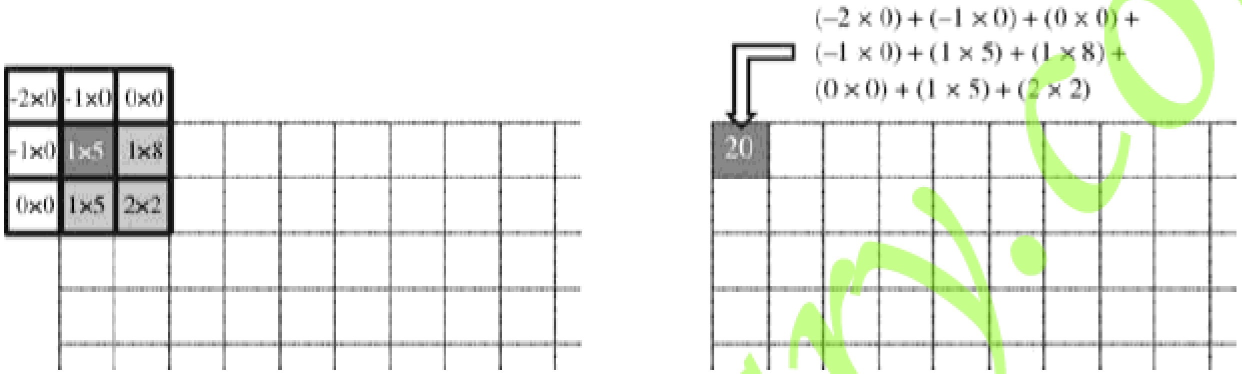


FIGURE 10.2 Two-dimensional convolution example.

TABLE 10.1 Examples of Convolution Masks

Low-Pass Filter	High-Pass Filter	Horizontal Edge Detection
$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

In practice, this is rewritten as

$$g(x, y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j, k) \cdot f(x + j, y + k) \tag{10.8}$$

where m_2 and n_2 are as defined earlier.

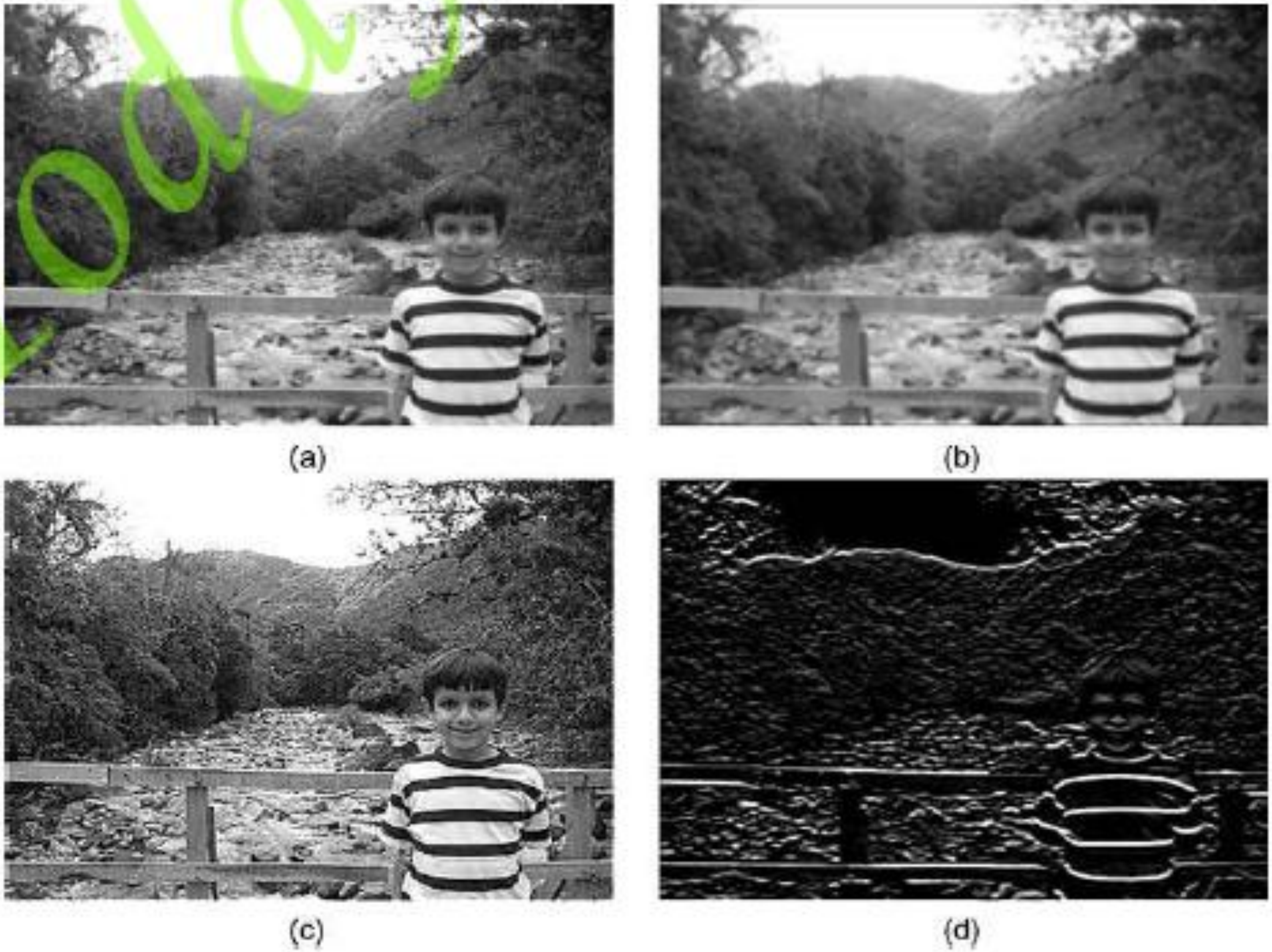


FIGURE 10.3 Applying different convolution masks to the same input image: (a) original image; (b–d) result of 2D convolution using the masks in Table 10.1.

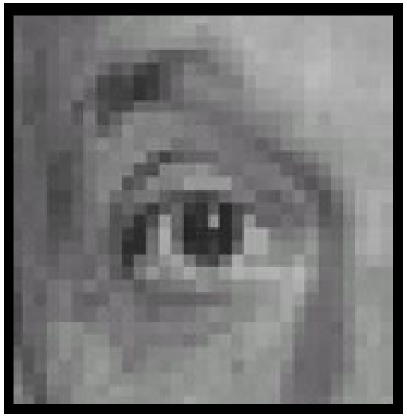
Example: Box Filter:

- Image filtering:
  - Compute function of local neighborhood at each position

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

- Really important!
  - Enhance images
    - Denoise, resize, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching

Think-Pair-Share Time:



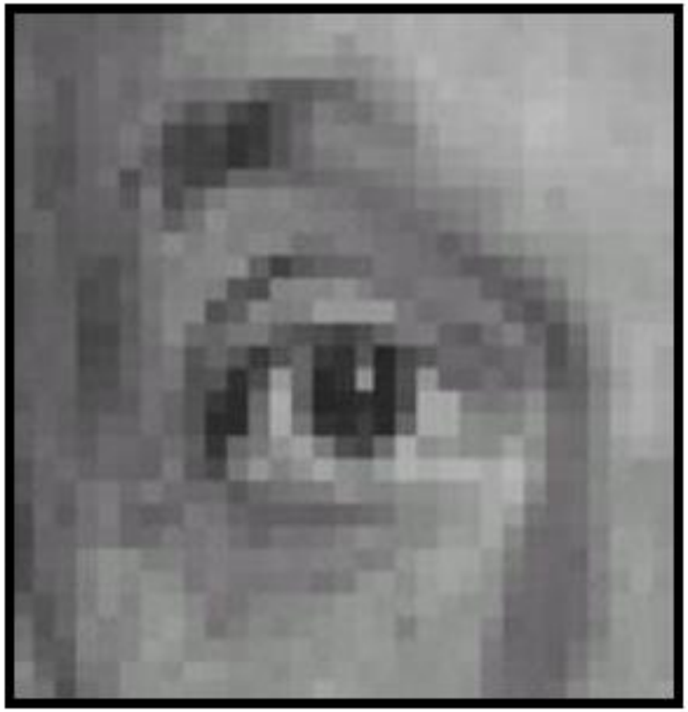
1. $$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

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

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

4. $$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Practice with Linear Filters



Original

0	0	0
0	1	0
0	0	0

?

Practice with Linear Filters

Original

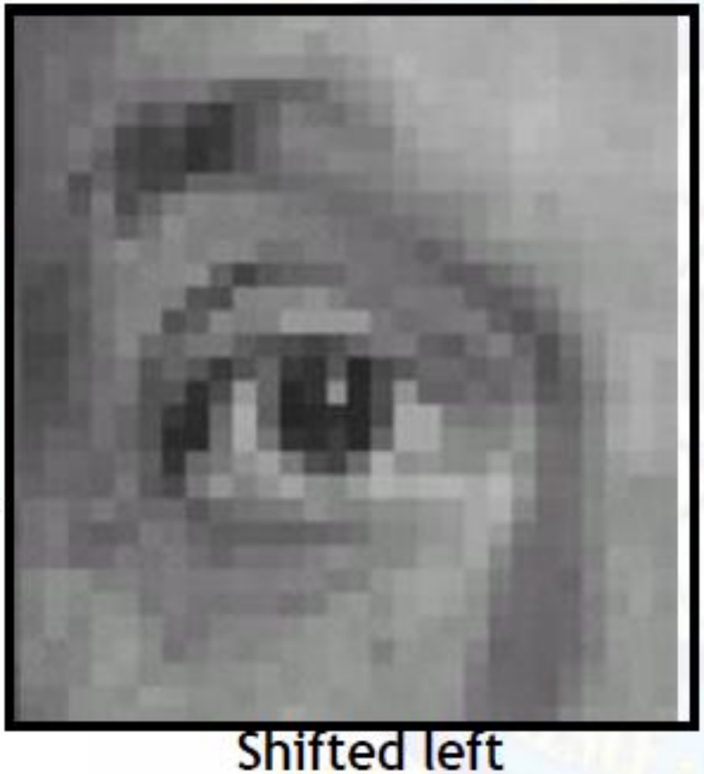
0	0	0
0	1	0
0	0	0

Filtered
(no change)

1. Practice with Linear Filters

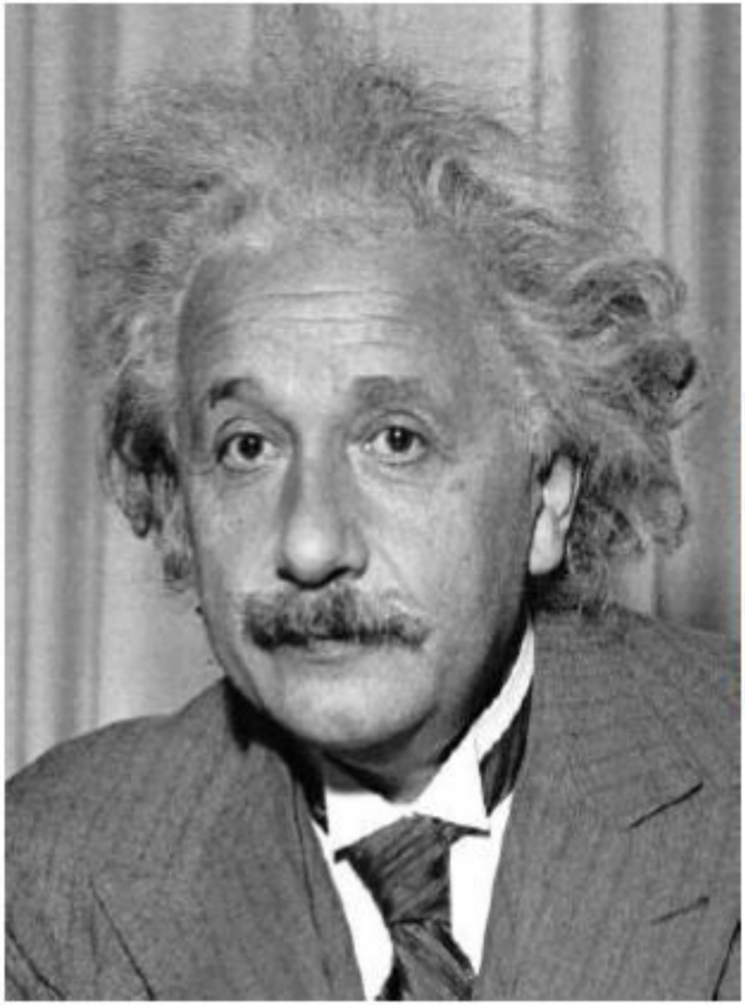
Original

0	0	0
0	0	1
0	0	0



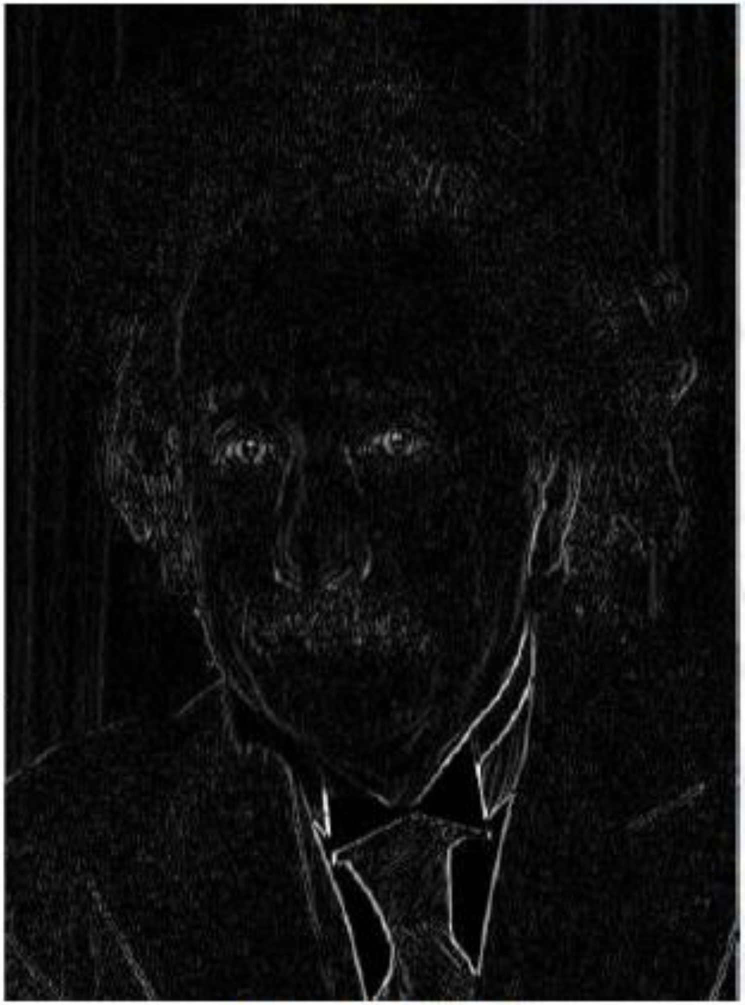
Shifted left
By 1 pixel

Practice with Linear Filters



1	0	-1
2	0	-2
1	0	-1

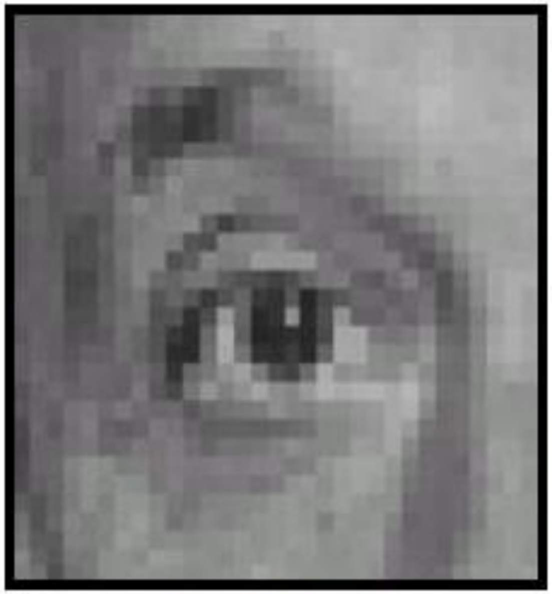
Sobel



Vertical Edge
(absolute value)

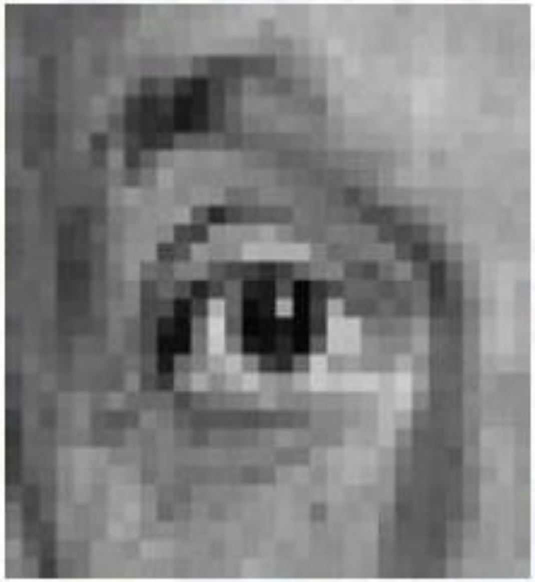
Practice with Linear Filters:

- Sharpening filter
  - Accentuates differences with local average



Original

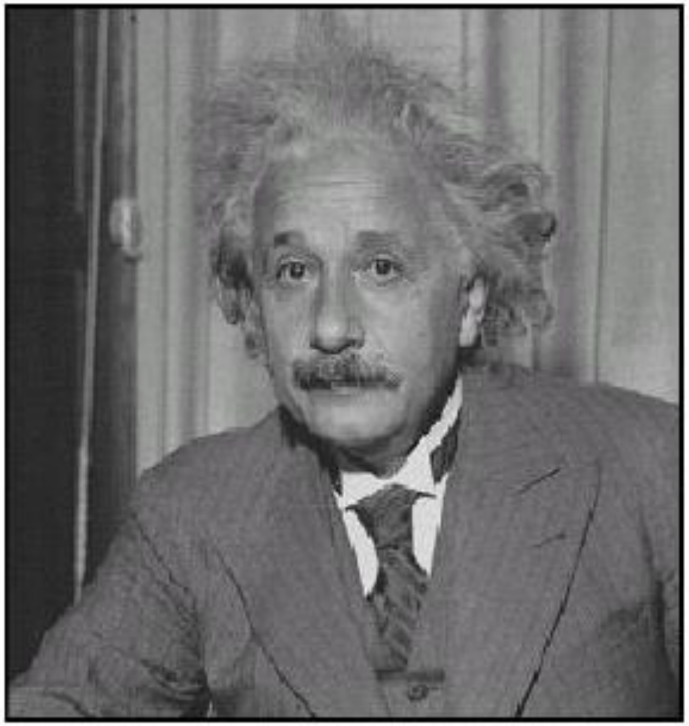
$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$



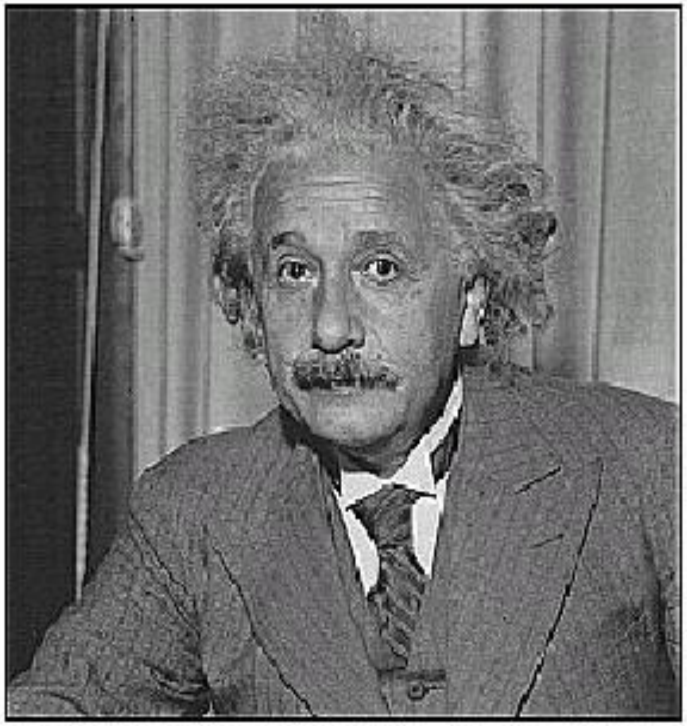
Sharpening filter

Practice with Linear Filters:

- Sharpening filter
  - Accentuates differences with local average



before



after

Separable 2-D Filter

$$y[m,n] = \sum_i \sum_j h[m-i, n-j]\, x[i,j]$$

separable ⟹ $h[m,n] = h_1[m]h_2[n]$

Low-pass filter	High-pass filter	Horizontal edge detection
$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
separable	non-separable	separable

Separable Smoothing Kernel Examples

$$h[m,n] = h_1[m]h_2[n]$$

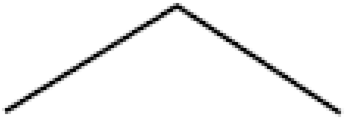
$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

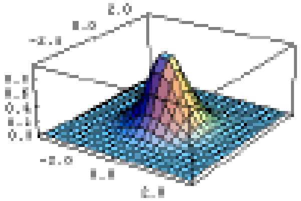
$$\frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

Mean filter

Bilinear filter

Gaussian filter





Implementation of a Separable 2-D Filter

$$
\begin{aligned}
y[m,n] &= \sum_i \sum_j h_1[m-i]h_2[n-j]x[i,j] \\
&= \sum_j h_2[n-j]\underbrace{\sum_i h_1[m-i]x[i,j]}_{} \quad \longleftarrow \quad \text{rows} \\
&= \sum_j h_2[n-j]\bar{x}[m,j] \quad \longleftarrow \quad \text{columns}
\end{aligned}
$$

Two 1-D convolution computations: $2N\log N$

Rather than one 2-D convolution: $N^2\log N^2$

Convolution Properties:

- Commutative: a * b = b * a
  - Conceptually no difference between filter and signal
  - But particular filtering implementations might break this equality, e.g., image edges

- Associative: a * (b * c) = (a * b) * c
  - Often apply several filters one after another: (((a * b1) b2) * b3)
  - This is equivalent to applying one filter: a * (b1 * b2 * b3)
  - Correlation is _not_ associative (rotation effect)
  - Why important?

Convolution Properties:

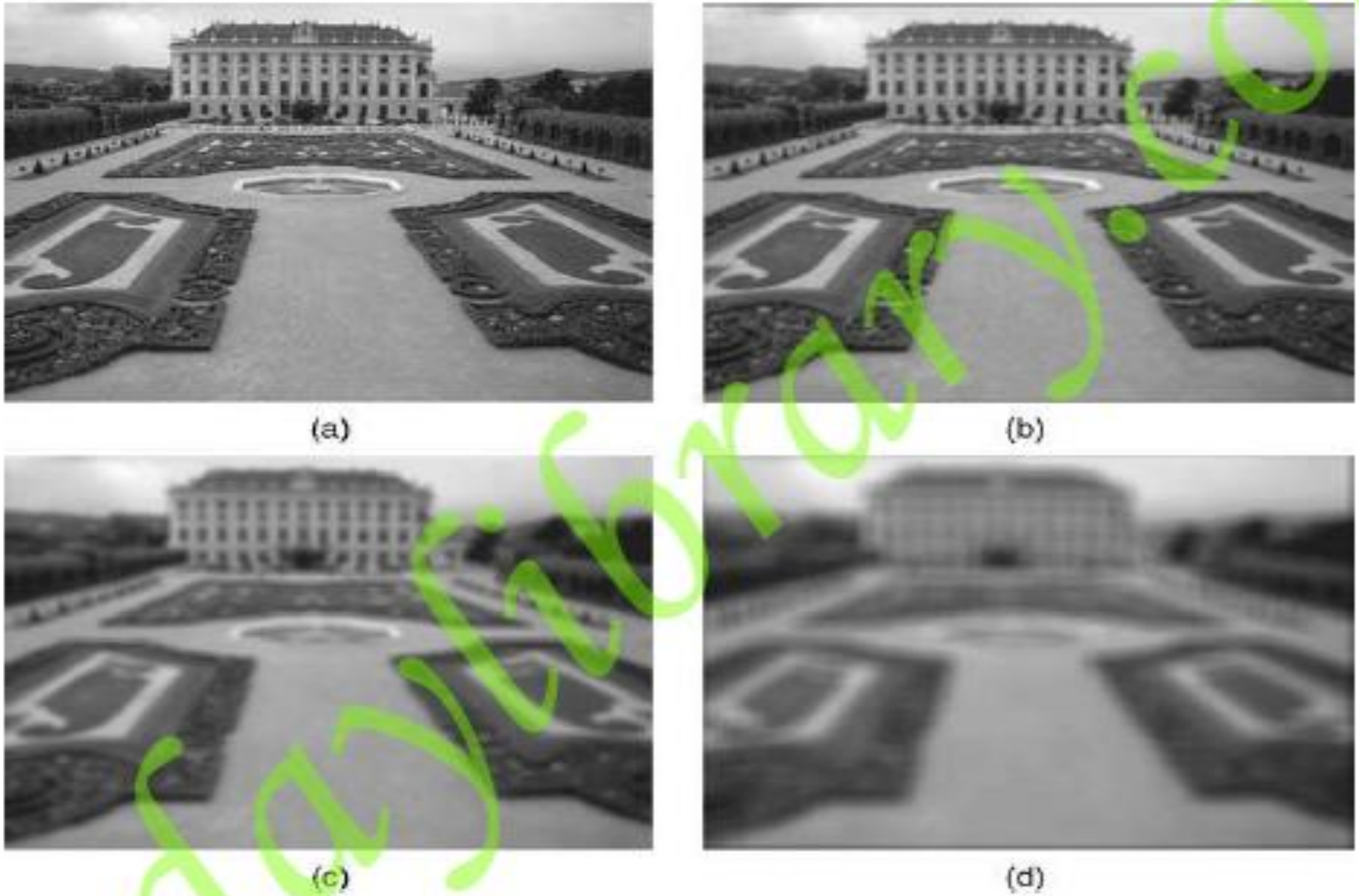
- Commutative: a * b = b * a
  - Conceptually no difference between filter and signal
  - But particular filtering implementations might break this equality, e.g., image edges
- Associative: a * (b * c) = (a * b) * c
  - Often apply several filters one after another: (((a * b1) * b2) * b3)
  - This is equivalent to applying one filter: a * (b1 * b2 * b3)
  - Correlation is _not_ associative (rotation effect)
  - Why important?
- Distributes over addition: a * (b + c) = (a * b) + (a * c)
- Scalars factor out: ka * b = a * kb = k (a * b)
- Identity: unit impulse e = [0, 0, 1, 0, 0], a * e = a

2D Filters

Mean (averaging) filter

- The simplest and most widely known spatial smoothing filter.
- It uses convolution with a mask whose coefficients have a value of 1, and divides the result by a scaling factor (the total number of elements in the mask).
- Also known as *box filter*.

$$h(x,y) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



(a)

(b)

(c)

(d)

FIGURE 10.5 Examples of applying the averaging filter with different mask sizes: (a) input image (899 × 675 pixels); (b–d) output images corresponding to averaging masks of size 7 × 7, 15 × 15, and 31 × 31.

Mean Filter Variations

- Modified mask coefficients, e.g. Give more importance to the center pixel:

$$h(x,y) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & 0.2 & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

- Directional averaging: rectangular mask for blurring is done in a specific direction.
- Selective application of averaging calculation results:
  - if the difference between original and processed values is larger than T, keep the original pixel (preserves important edges)
- Removal of outliers before calculating the average

Separability Example:

2D convolution
(center location only)

$$\begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&4&2\\\hline 1&2&1\\\hline\end{array} * \begin{array}{|c|c|c|}\hline 2&3&3\\\hline 3&5&5\\\hline 4&4&6\\\hline\end{array}$$

$$\begin{aligned}&=2+6+3=11\\&=6+20+10=36\\&=4+8+6=18\\\hline&\quad 65\end{aligned}$$

The filter factors
into a product of 1D
filters:

$$\begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&4&2\\\hline 1&2&1\\\hline\end{array} = \begin{array}{|c|}\hline 1\\\hline 2\\\hline 1\\\hline\end{array} \times \begin{array}{|c|c|c|}\hline 1&2&1\\\hline\end{array}$$

Perform convolution
along rows:

$$\begin{array}{|c|c|c|}\hline 1&2&1\\\hline\end{array} * \begin{array}{|c|c|c|}\hline 2&3&3\\\hline 3&5&5\\\hline 4&4&6\\\hline\end{array} = \begin{array}{|c|c|c|}\hline &11&\\\hline &18&\\\hline &18&\\\hline\end{array}$$

Followed by convolution
along the remaining column:

$$\begin{array}{|c|}\hline 1\\\hline 2\\\hline 1\\\hline\end{array} * \begin{array}{|c|c|c|}\hline &11&\\\hline &18&\\\hline &18&\\\hline\end{array} = \begin{array}{|c|c|c|}\hline &&\\\hline &65&\\\hline &&\\\hline\end{array}$$

Separability

- Why is separability useful in practice?
- If K is width of convolution kernel:
  - 2D convolution = K^2 multiply-add operations
  - 2x 1D convolution: 2K multiply-add operations

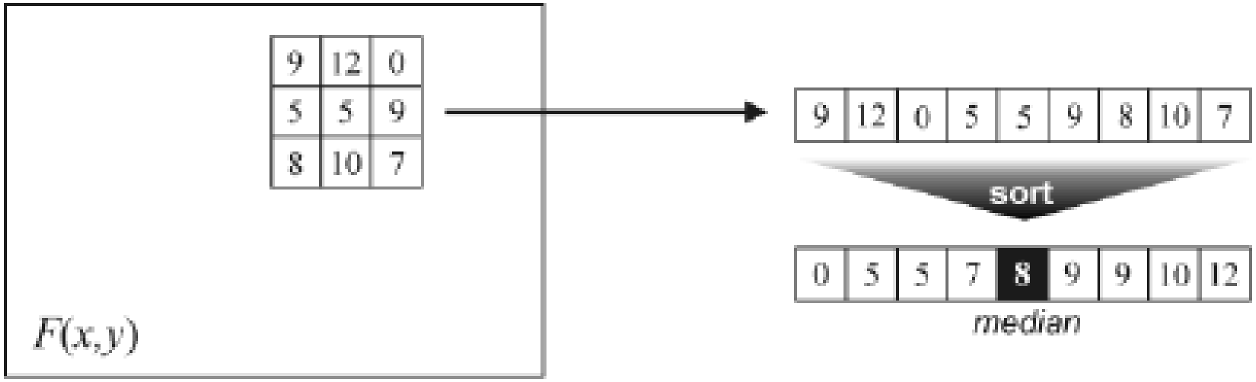
Practical matters

How big should the filter be?

- Values at edges should be near zero
- Gaussians have infinite extent…
- Rule of thumb for Gaussian: set filter half-width to about 3 σ

Median filter

- Works by sorting the pixel values within a neighborhood, finding the median value and replacing the original pixel value with the median of that neighborhood.



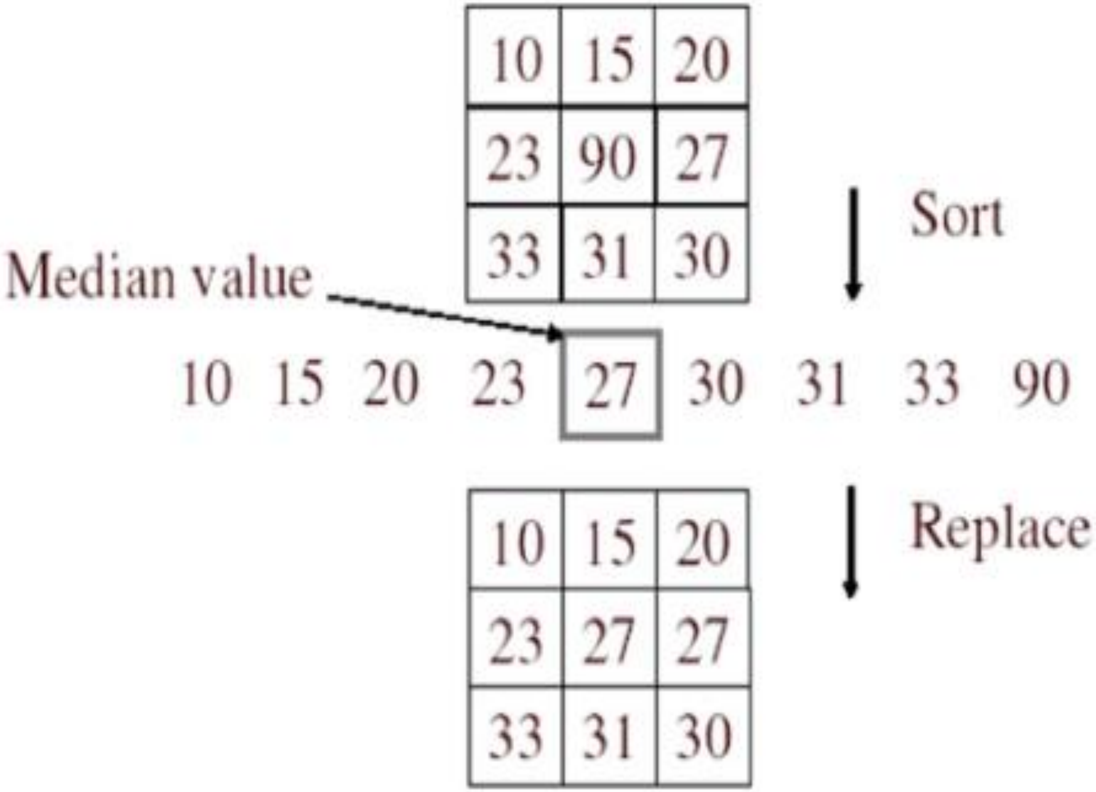
9	12	0
5	5	9
8	10	7

9	12	0	5	5	9	8	10	7

0	5	5	7	8	9	9	10	12

Median filter in 2D

- A **median filter** operates over a window by selecting the median intensity in the window



10	15	20
23	90	27
33	31	30

Sort

Median value

10 15 20 23 27 30 31 33 90

Replace

10	15	20
23	27	27
33	31	30

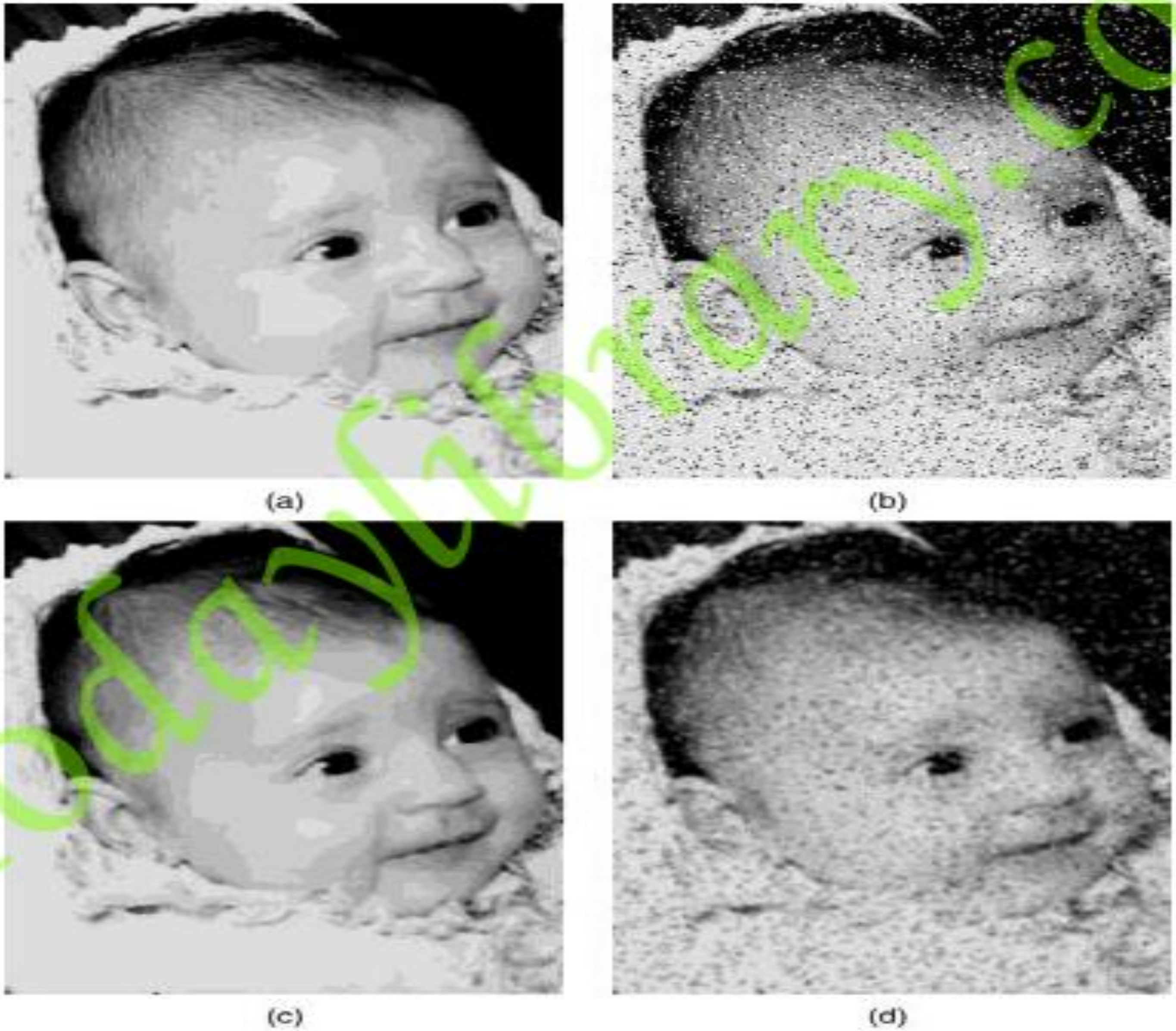


FIGURE 10.9 (a) Original image; (b) image with salt and pepper noise; (c) result of 3 × 3 median filtering; (d) result of 3 × 3 neighborhood averaging.

10.4 IMAGE SHARPENING (HIGH-PASS FILTERS)

We call *high-pass* filters those spatial filters whose effect on an image is equivalent to preserving or emphasizing its high-frequency components (i.e., fine details, points, lines, and edges), that is, to highlight transitions in intensity within the image.

Linear HPFs can be implemented using 2D convolution masks with positive and negative coefficients, which correspond to a digital approximation of the *Laplacian*, a simple, *isotropic* (i.e., *rotation invariant*) second-order derivative that is capable of responding to intensity transitions in any direction.

10.3.3 Gaussian Blur Filter

The Gaussian blur filter is the best-known example of a LPF implemented with a nonuniform kernel. The mask coefficients for the Gaussian blur filter are samples from a 2D Gaussian function (plotted in Figure 10.6):

$$h(x, y) = \exp \frac{-(x^2 + y^2)}{2\sigma^2} \tag{10.11}$$

The parameter σ controls the overall shape of the curve: the larger the value of σ, the flatter the resulting curve.

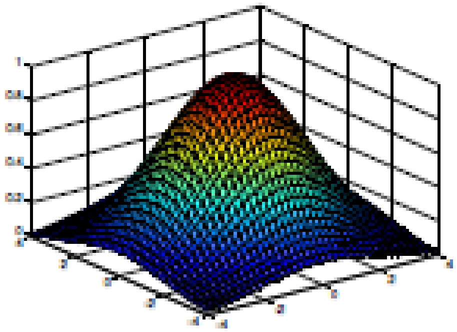
Gaussian blur filter

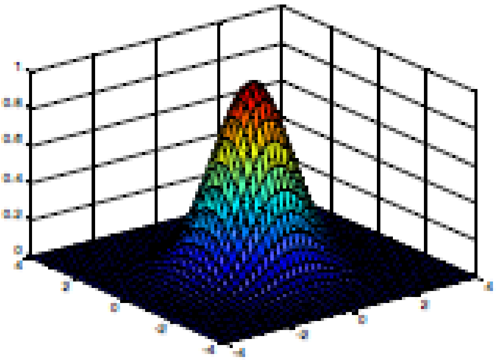
- The best-known example of a LPF implemented with a non-uniform kernel.
- The mask coefficients for the Gaussian blur filter are samples from a 2D Gaussian function:

$$h(x, y) = \exp\left[\frac{-(x^2 + y^2)}{2\sigma^2}\right]$$

- The parameter sigma controls the overall shape of the curve. The larger the sigma, the flatter the resulting curve.

$\sigma = 2$





$\sigma = 1$

Some of the most notable properties of the Gaussian blur filter are as follows:

- The kernel is symmetric with respect to rotation; therefore, there is no directional bias in the result.
- The kernel is separable, which can lead to fast computational implementations.
- The kernel's coefficients fall off to (almost) zero at the kernel's edges.

- The Fourier transform (FT) of a Gaussian filter is another Gaussian (this will be explained in Chapter 11).
- The convolution of two Gaussians is another Gaussian.
- The output image obtained after applying the Gaussian blur filter is more pleasing to the eye than the one obtained using other low-pass filters.

Gaussian Filter

$$\sigma = 1 \qquad N = 3 \qquad Z = \begin{matrix} \boxed{0.0751} & 0.1238 & 0.0751 \\ 0.1238 & \boxed{0.2042} & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{matrix}$$

$$\sigma = 2 \qquad N = 3 \qquad Z = \begin{matrix} \boxed{0.1019} & 0.1154 & 0.1019 \\ 0.1154 & \boxed{0.1308} & 0.1154 \\ 0.1019 & 0.1154 & 0.1019 \end{matrix}$$

Gaussian Filter Kernel Size?

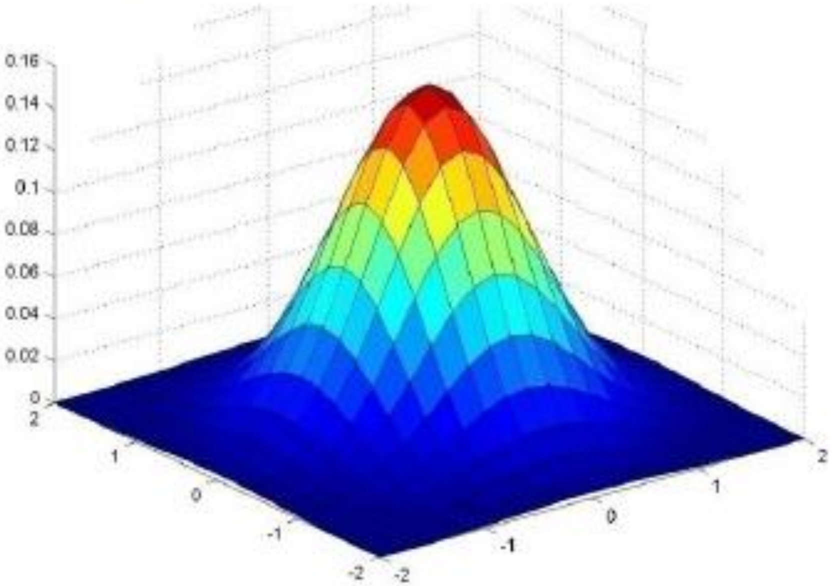
- Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel
- In practice it is effectively zero more than about three standard deviations from the mean, and so we can truncate the kernel at this point.

$$\sigma = 1$$
$$\Rightarrow N \cong 3 \times \sigma \times 2$$
$$\Rightarrow N = 5 \;\; \text{(should be odd)}$$

```
Z =   0.0030   0.0133   0.0219   0.0133   0.0030
      0.0133   0.0596   0.0983   0.0596   0.0133
      0.0219   0.0983   0.1621   0.0983   0.0219
      0.0133   0.0596   0.0983   0.0596   0.0133
      0.0030   0.0133   0.0219   0.0133   0.0030
```

Important filter: Gaussian:

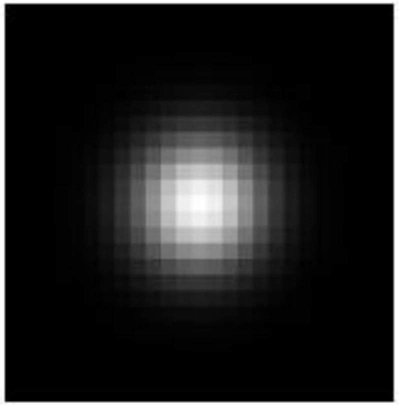
- Weight contributions of neighboring pixels by nearness



x

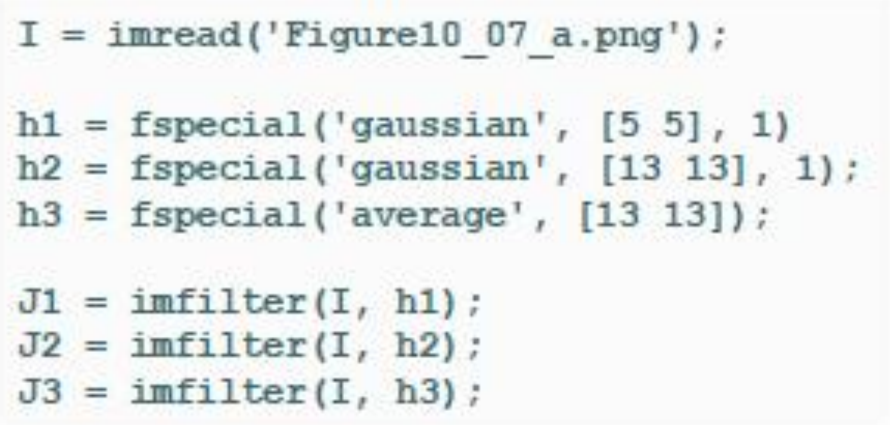
	0.003	0.013	0.022	0.013	0.003
	0.013	0.059	0.097	0.059	0.013
y	0.022	0.097	0.159	0.097	0.022
	0.013	0.059	0.097	0.059	0.013
	0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1



$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

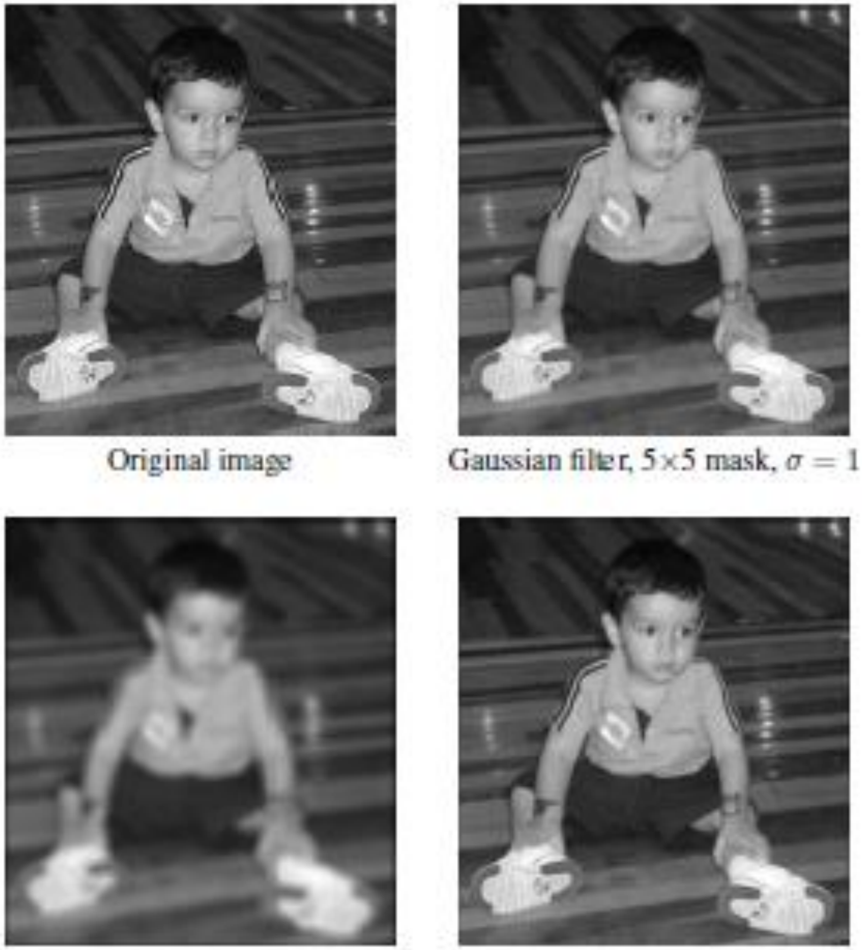
Gaussian Blur Filter



```
I = imread('Figure10_07_a.png');

h1 = fspecial('gaussian', [5 5], 1)
h2 = fspecial('gaussian', [13 13], 1);
h3 = fspecial('average', [13 13]);

J1 = imfilter(I, h1);
J2 = imfilter(I, h2);
J3 = imfilter(I, h3);
```



Original image

Gaussian filter, 5×5 mask, $\sigma = 1$

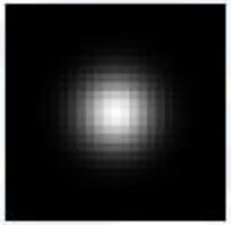
Mean filter, 13×13 mask

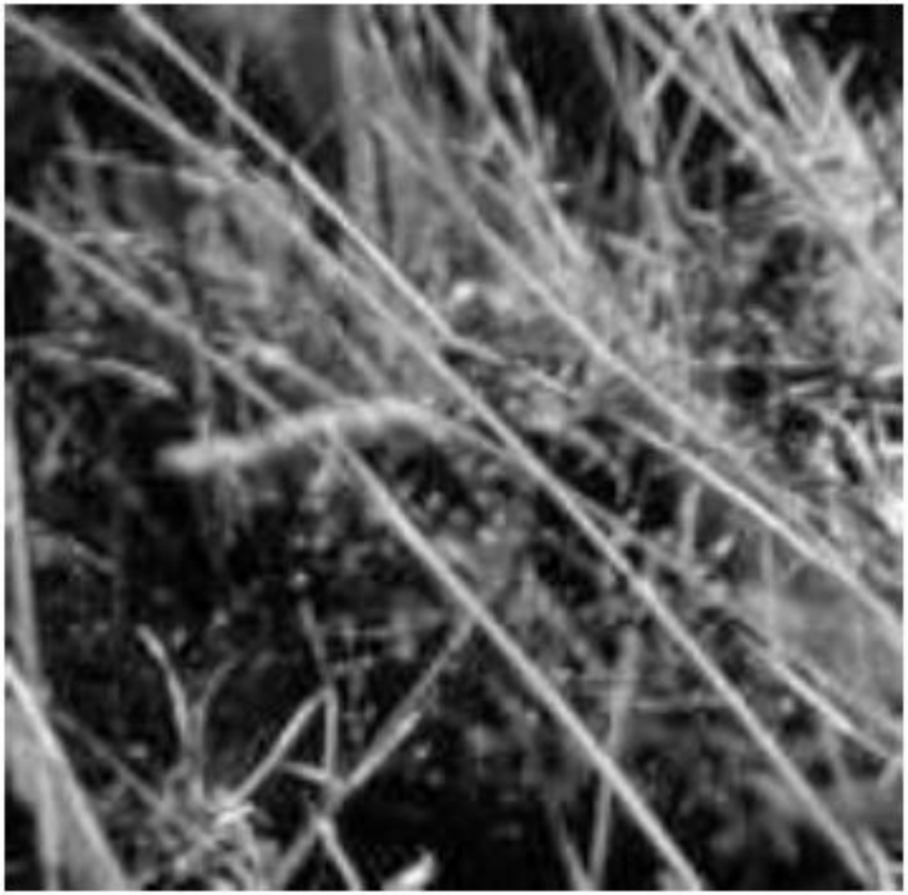
Gaussian filter, 13×13 mask, $\sigma = 1$

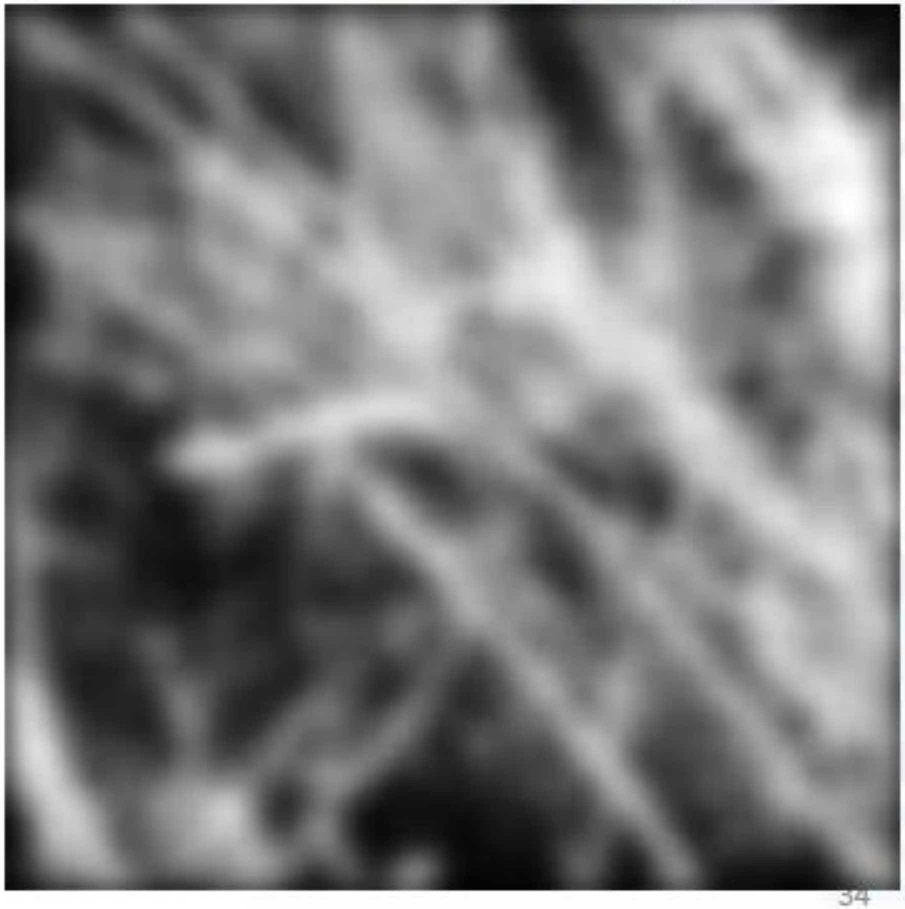
By Oge Marques

Important filter: Gaussian:

- Smoothing with Gaussian filter.

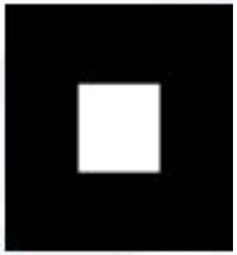


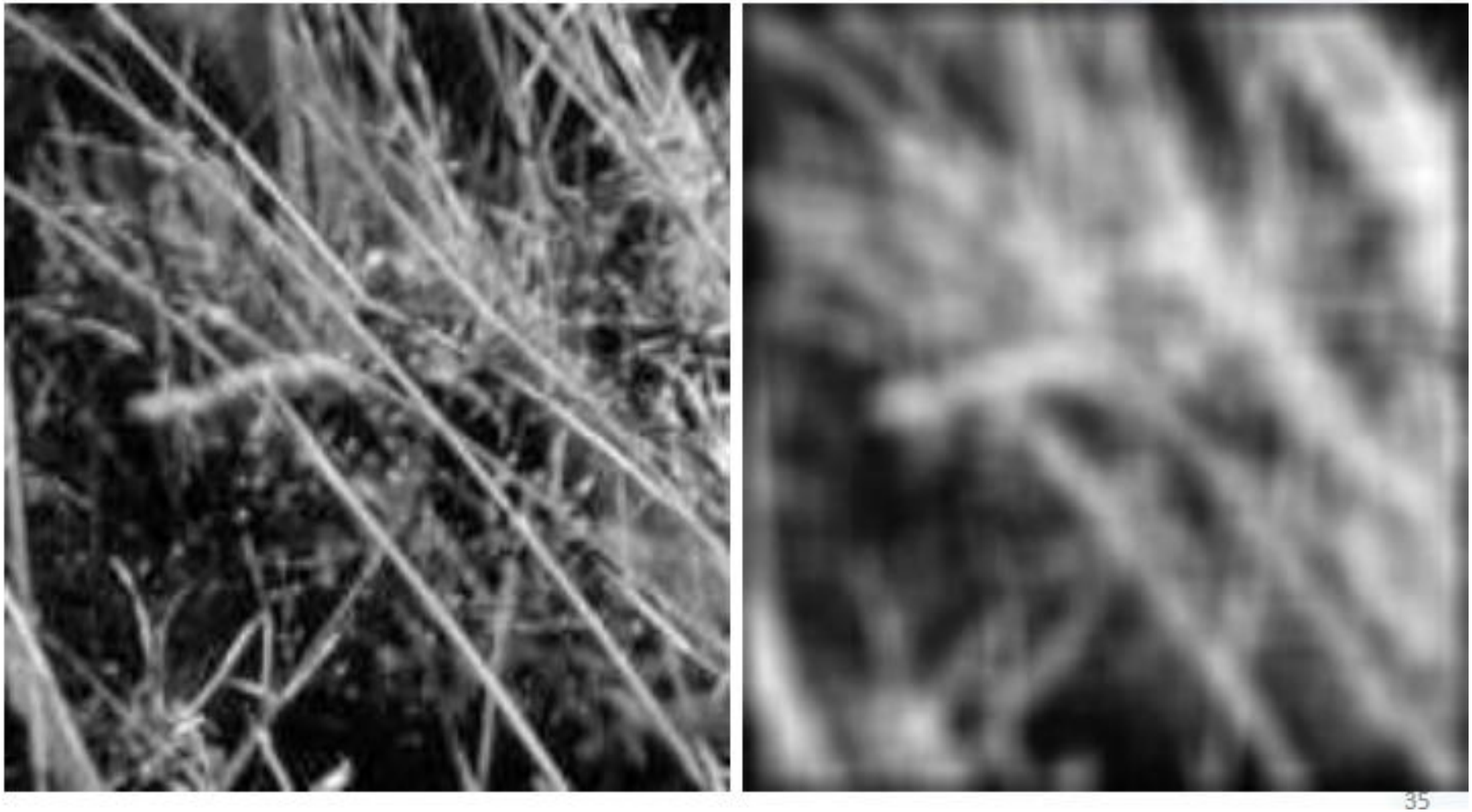




Important filter: Gaussian:

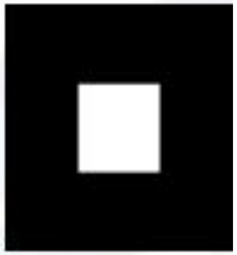
- Smoothing with Gaussian filter.

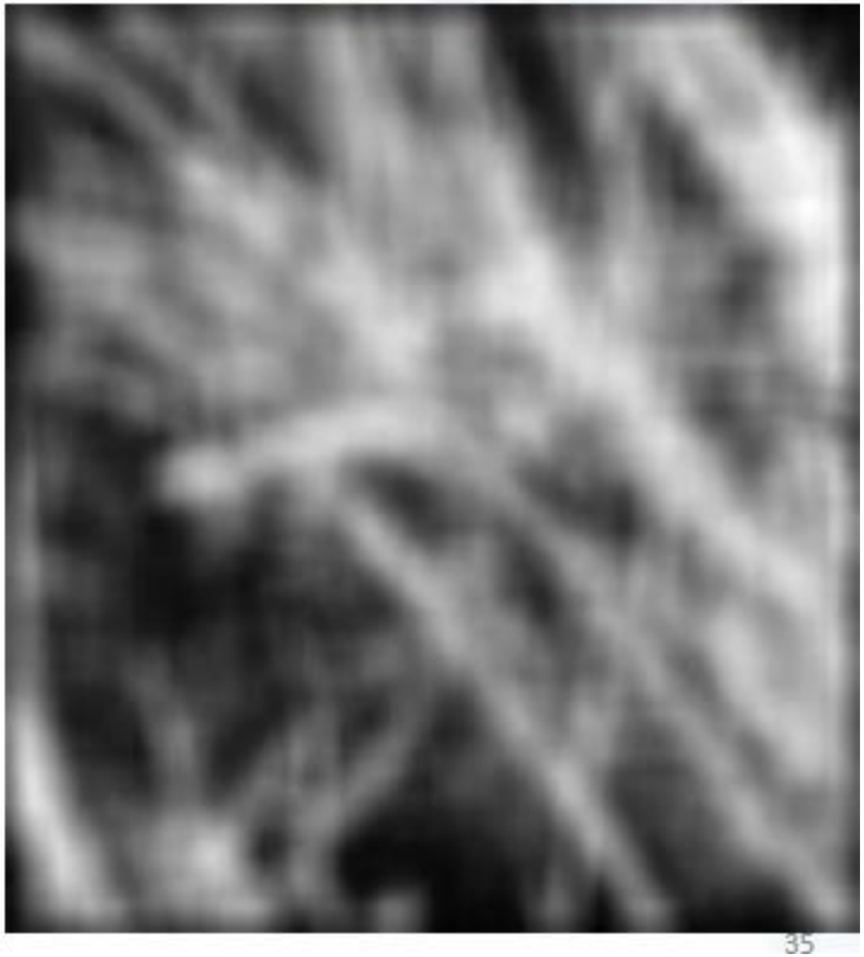




Important filter: Gaussian:

- Smoothing with Gaussian filter.





Gaussian Filters:

- Remove “high-frequency” components from the image (low-pass filter)
  - Images become more smooth
- Convolution with self is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- Separable kernel
  - Factors into product of two 1D Gaussians

Gaussian Filters:

- Remove “high-frequency” components from the image (low-pass filter)

$$
\begin{aligned}
G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\
&= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right)
\end{aligned}
$$

The 2D Gaussian can be expressed as the product of two functions, one a function of *x* and the other a function of *y*

In this case, the two functions are the (identical) 1D Gaussian

The Laplacian

- The Laplacian operator is defined as

$$\nabla^2(x, y) = \frac{\partial^2(x, y)}{\partial x^2} + \frac{\partial^2(x, y)}{\partial y^2}$$

- The Laplacian of an image is approximated as

$$\nabla^2(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

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

Laplacian Mask

- An alternative digital implementation of the Laplacian takes into account all eight neighbors of the reference pixel and can be implemented using:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

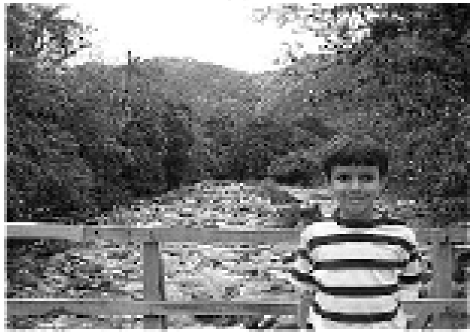
Composite Laplacian Mask

- Goal is to restore the gray-level tonality that was lost in Laplacian calculations.
- Laplacian mask produces results centered around zero, and hence very dark images.

$$g(x, y) = f(x, y) + c\left[\nabla^2(x, y)\right]$$

- For *c*=1: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

(a)



(c)

It is also common to factor equation (10.16) into the design of the mask, which produces the *composite* Laplacian mask below:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

L EXAMPLE 10.5

Figure 10.10 shows an example (using the `imfilter` and `fspecial` functions in MATLAB) of applying a high-pass filter to enhance (sharpen) a monochrome image. Figure 10.10a shows the original image. Figure 10.10b shows the resulting enhanced image obtained by applying equation (10.16) with $c = -1$ and Figure 10.10c shows the result of using the eight-directional Laplacian operator instead. It can be claimed that the results in part (c) are crisper than the ones obtained in part (b).

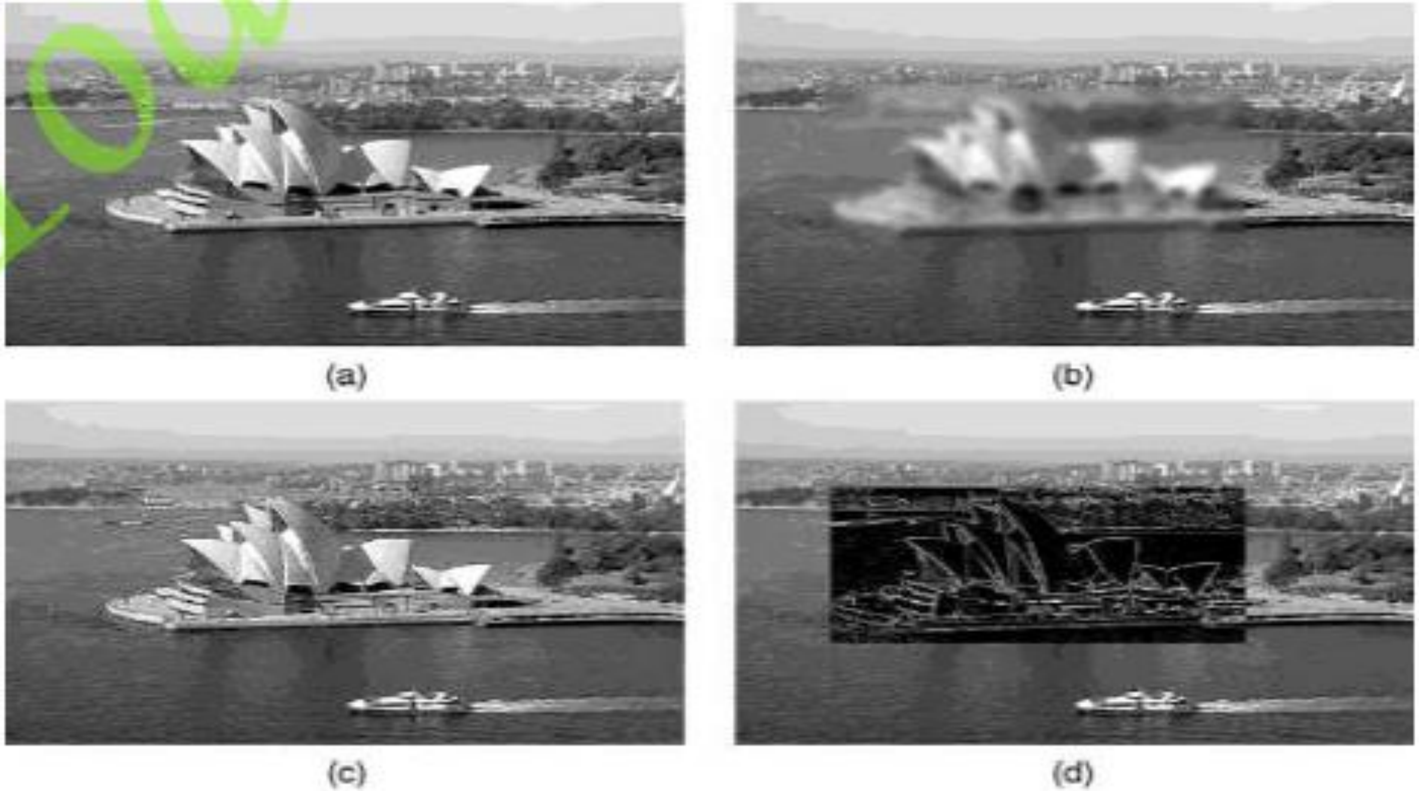
High-Boost Filtering

$$\frac{1}{c-8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & c & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- where: c ($c > 8$) is a coefficient ("amplification factor") that controls how much weight is given to the original image and the high-pass filtered version of that image.
    - For c=9, the result would be equivalent to that seen on the previous page.
    - Greater values of c will cause less sharpening.

L EXAMPLE 10.6

Figure 10.11 shows an example of ROI processing using the `roifilt2` function in MATLAB.



(a)

(b)

(c)

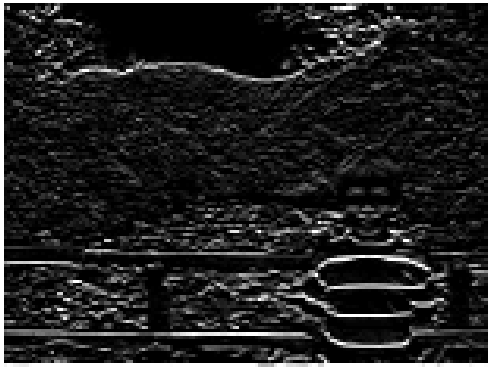
(d)

FIGURE 10.11 Example of region of interest processing: (a) original image; (b) result of applying a Gaussian blur to a selected ROI; (c) result of applying a HPF to a selected ROI; (d) result of applying a Laplacian mask to a selected ROI.

Directional Difference Filters

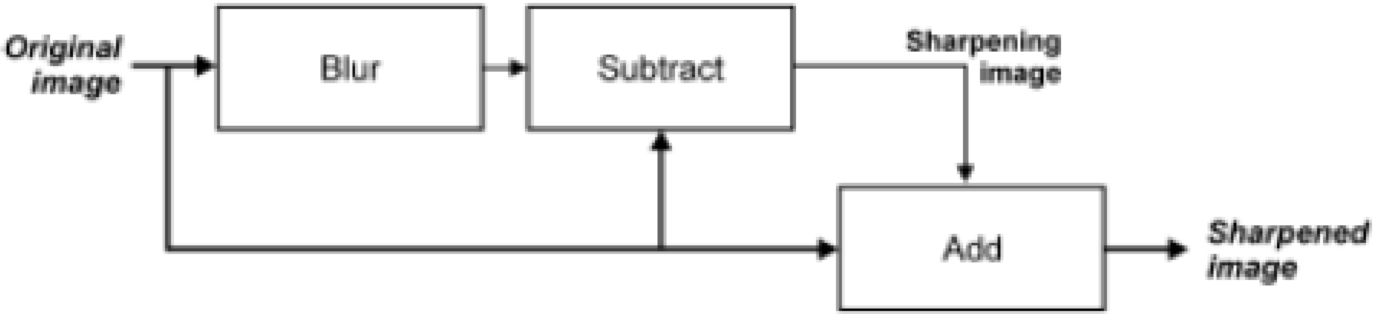
- Similar to the Laplacian high-frequency filter.
  - Main difference: directional difference filters emphasize edges in a specific direction.
- Examples:

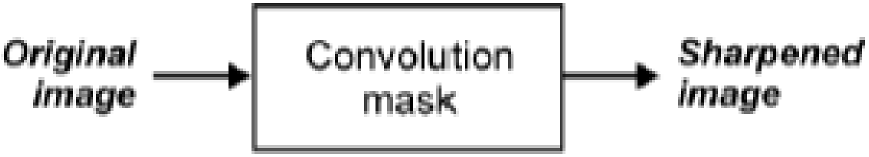
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$



Horizontal edge detection
$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Unsharp Masking





Unsharp Masking

- Blur the image $\overline{f}(x,y)$
- Obtain the unsharp mask: $g_{mask}(x,y) = f(x,y) - \overline{f}(x,y)$
- Add a weighted portion of the mask back to the original image $g(x,y) = f(x,y) + kg_{mask}(x,y)$
  - If k = 1, we have unsharp masking
  - If k>1, it is called **highboost filtering.**

- Unsharp mask is very similar to what we would obtain using a second order derivative:

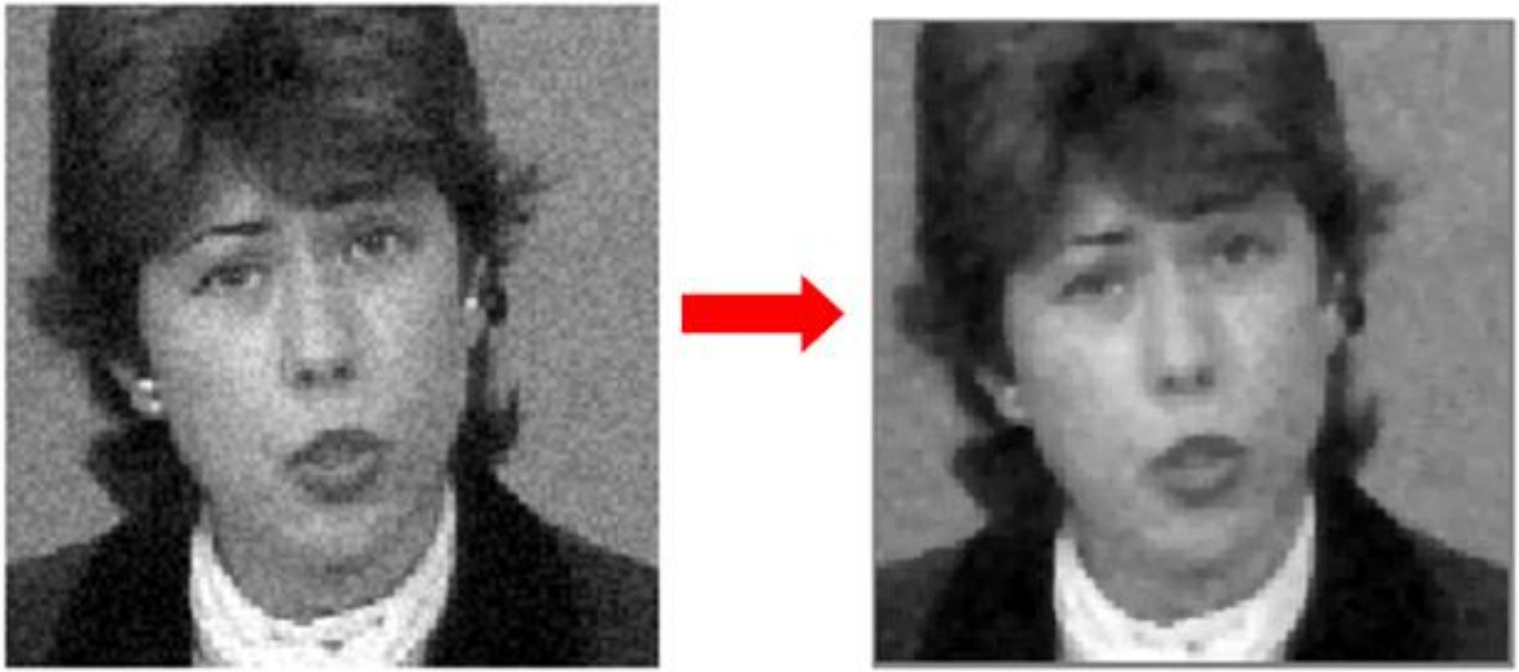
$$g(x,y) = f(x,y) + c\left[\nabla^2(x,y)\right]$$

Nonlinear Filters

- Nonlinear filters also work at a neighborhood level, but do not process the pixel values using the convolution operator.
- Rank filters apply a ranking (sorting) function to the pixel values within the neighborhood and select a value from the sorted list.
  - Examples: median filter, max and min filters

Gaussian noise

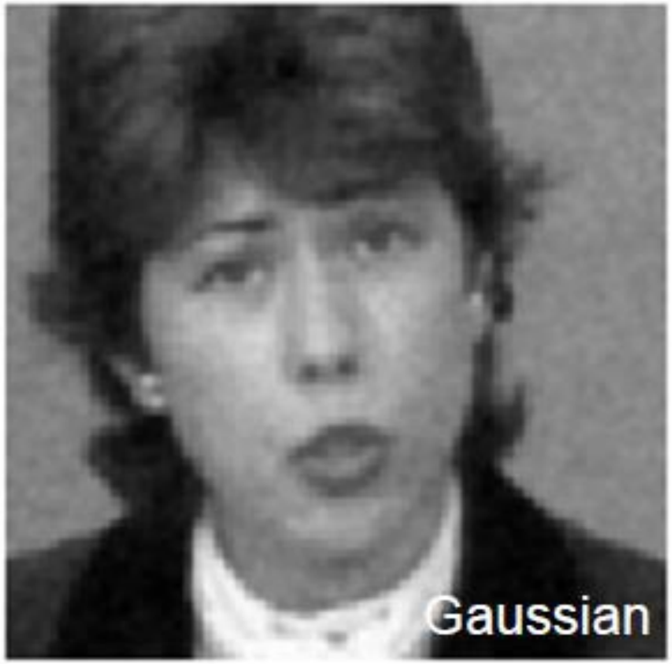
2D median filter, 3 x 3 neighbourhood



- sharpens edges, reduces noise, but …
- generates jagged edges

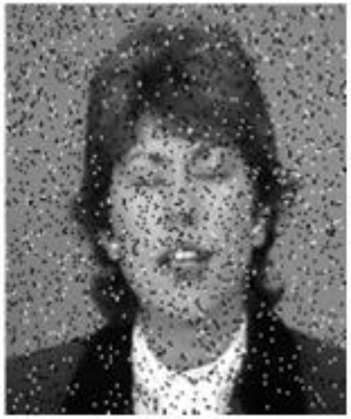
Gaussian noise

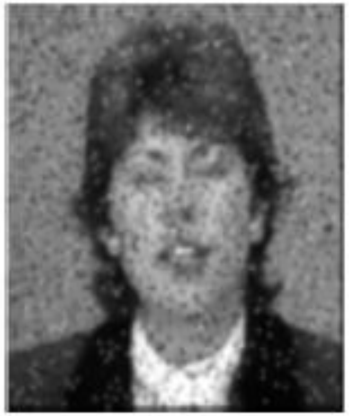
Comparison with Gaussian filter



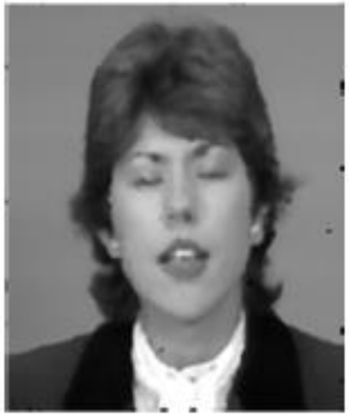
Gaussian: upper lip smoother, eye better preserved

Salt and Pepper noise



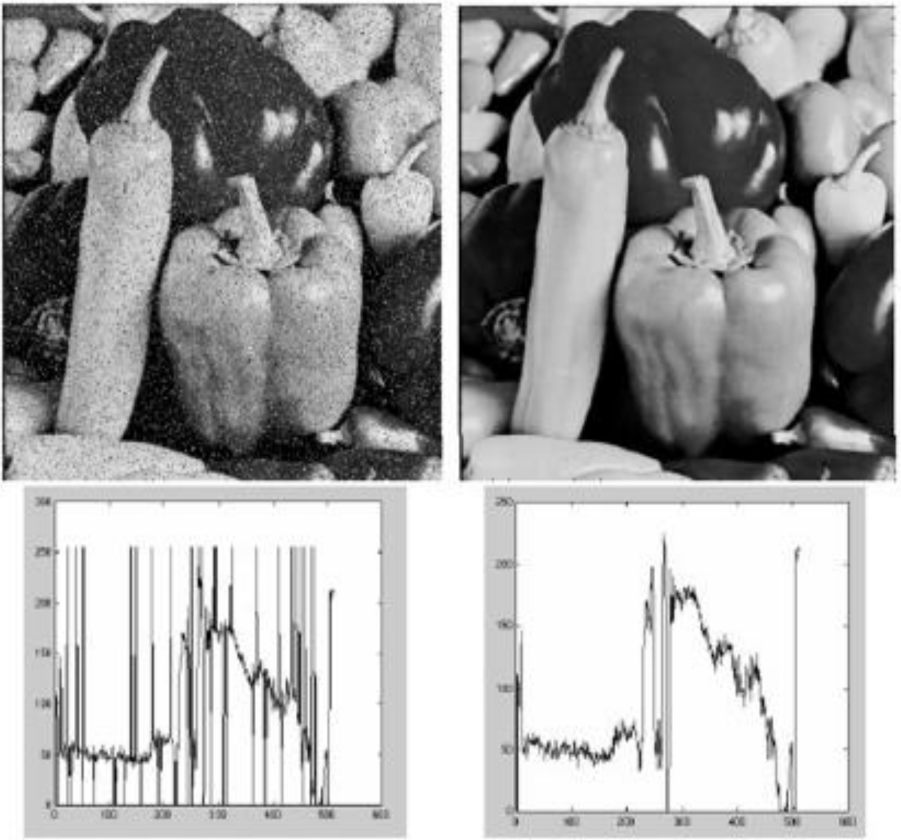


Gaussian
ρ = 1 pixel



3 x 3
median

Median filter



Salt and pepper noise

Median filtered

Plots of a row of the image

Matlab: output im = medfilt2(im, [h w]);

Source: M. Hebert

Image Sharpening (High-Pass Filters)

- Spatial filters whose effect on the output image is equivalent to emphasizing its high-frequency components (e.g., fine details, points, lines, and edges).
- Linear HPFs can be implemented using 2D convolution masks which correspond to a digital approximation of the *Laplacian* operator