CPE409 Image Processing

Part 5
Spatial Filtering

Assist. Prof. Dr. Caner ÖZCAN

If the facts don't fit the theory, change the facts.
~Einstein

Outline

3. Intensity Transformations and Spatial Filtering
    - Some Basic Intensity Transformation Functions
    - Histogram Processing
    - Fundamentals of Spatial Filtering
    - Smoothing Spatial Filters
    - Sharpening Spatial Filters
    - Combining Spatial Enhancement Methods
    - *Using Fuzzy Techniques for Intensity Transformations and Spatial Filtering*

Importance of Neighborhood

- Both zebras and dalmatians have black and white pixels in similar numbers.
- The difference between the two is the characteristic appearance of small group of pixels rather than individual pixel values.

Spatial Filtering

- A spatial filter consists of (a) **a neighborhood**, and (b) **a predefined operation**

- Linear spatial filtering of an image of size MxN with a filter of size mxn is given by the expression

$$g(x,y)=\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)\,f(x+s,y+t)$$

Spatial Filtering

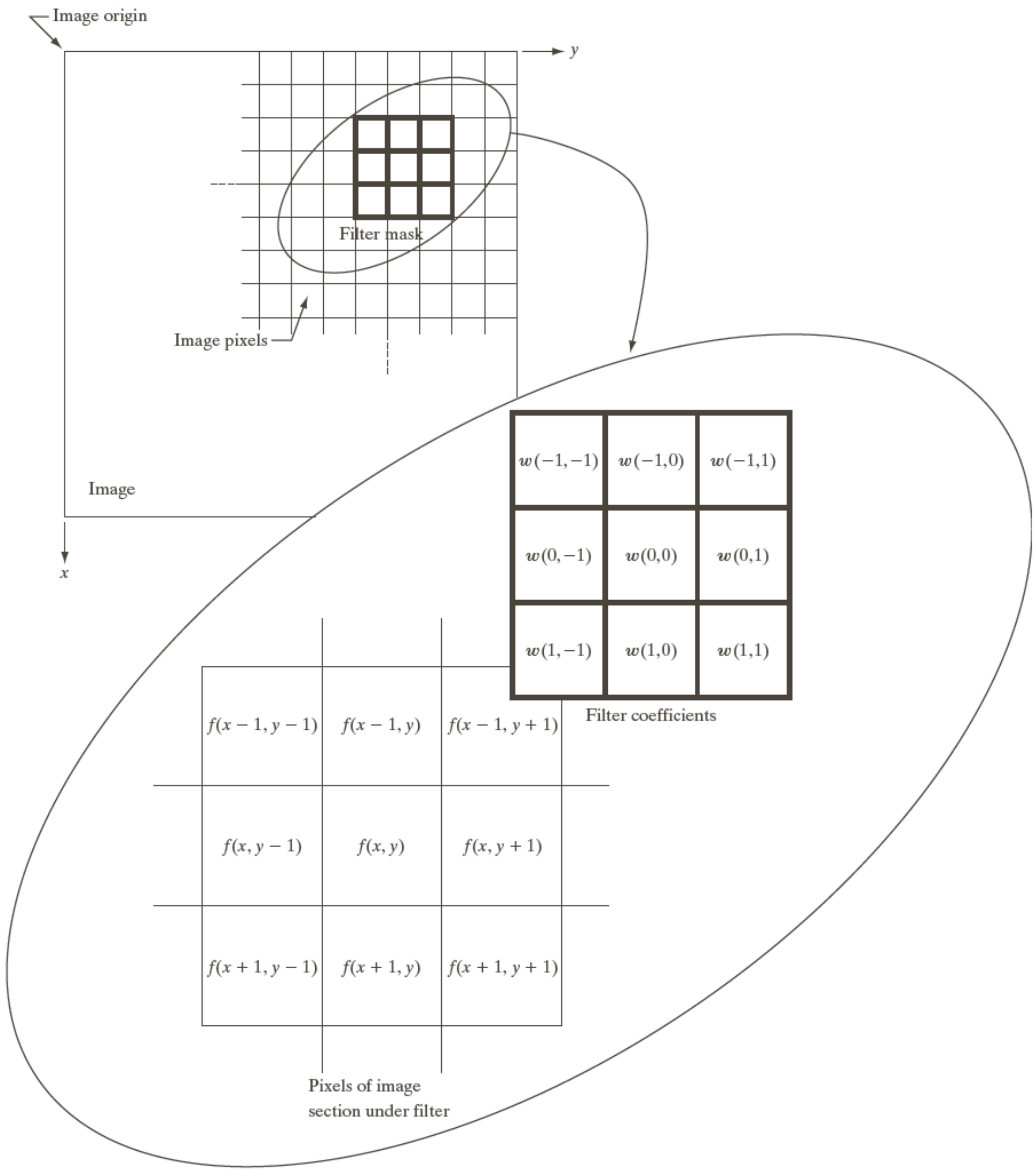


FIGURE 3.28 The mechanics of linear spatial filtering using a 3×3 filter mask. The form chosen to denote the coordinates of the filter mask coefficients simplifies writing expressions for linear filtering.

Spatial Filtering (Moving Average In 2D)

$F[x, y]$

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

$G[x, y]$

	0								

Slide credit: S. Seitz

Spatial Filtering (Moving Average In 2D)

$F[x, y]$

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

$G[x, y]$

	0	10							

Slide credit: S. Seitz

Spatial Filtering (Moving Average In 2D)

$F[x, y]$

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

$G[x, y]$

	0	10	20						

Slide credit: S. Seitz

Spatial Filtering (Moving Average In 2D)

$F[x, y]$

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

$G[x, y]$

	0	10	20	30					

Slide credit: S. Seitz

Spatial Filtering (Moving Average In 2D)

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

Slide credit: S. Seitz

Spatial Filtering (Moving Average In 2D)

$F[x, y]$

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

$G[x, y]$

	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	

Slide credit: S. Seitz

Smoothing Spatial Filters

- Smoothing filters are used for blurring and for noise reduction.

- Blurring is used in removal of small details and bridging of small gaps in lines or curves.

- Smoothing spatial filters include linear filters and nonlinear filters.

Two Smoothing Averaging Filter Masks

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

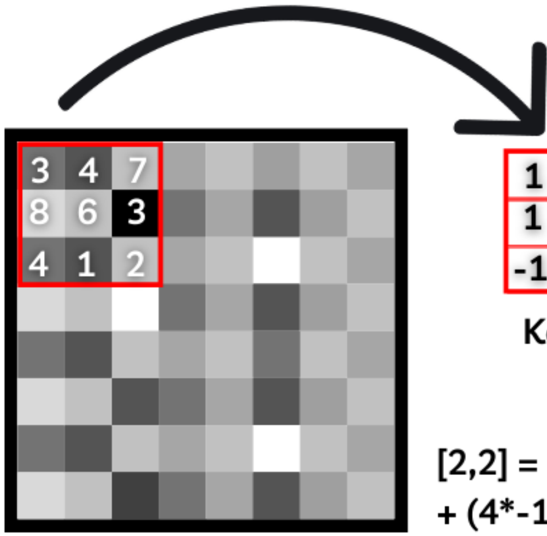
a b

FIGURE 3.32 Two 3×3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to 1 divided by the sum of the values of its coefficients, as is required to compute an average.

<u>Python Code:</u>

```
kernel1 = np.ones((3, 3))
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel1)
```

Two Smoothing Averaging Filter Masks



Image

1	-1	-1
1	-4	1
-1	1	-1

Kernel

3*1	4*-1	7*-1
8*1	6*-4	3*1
4*-1	1*1	2*-1

[2,2] = (3*1) + (4*-1) +(7*1) +(8*1) + (6*-4) + (3*1)
+ (4*-1) + (1*1) + (2*-1)

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

Identity kernel

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

Edge detection

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

Sharpen kernel

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

Box blur

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Gaussian blurr kernel

Blur an image with a 2D convolution matrix

```
# importing the modules needed
import cv2
import numpy as np

# Reading the image
image = cv2.imread('image.png')

# Creating the kernel(2d convolution matrix)
kernel1 = np.ones((5, 5), np.float32)/30

# Applying the filter2D() function
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel1)

# Shoeing the original and output image
cv2.imshow('Original', image)
cv2.imshow('Kernel Blur', img)

cv2.waitKey()
cv2.destroyAllWindows()
```

Smoothing (Averaging) Filter Masks

- Ortalama filtresini gerçekleştirmek için *cv2.blur()* ve *cv2.boxFilter()* işlevleri kullanılabilir.
- Her iki işlev de çekirdeği kullanarak bir görüntüyü düzgünleştirir.

Syntax of cv2.blur()

```
cv2.blur(image, ksize)
```

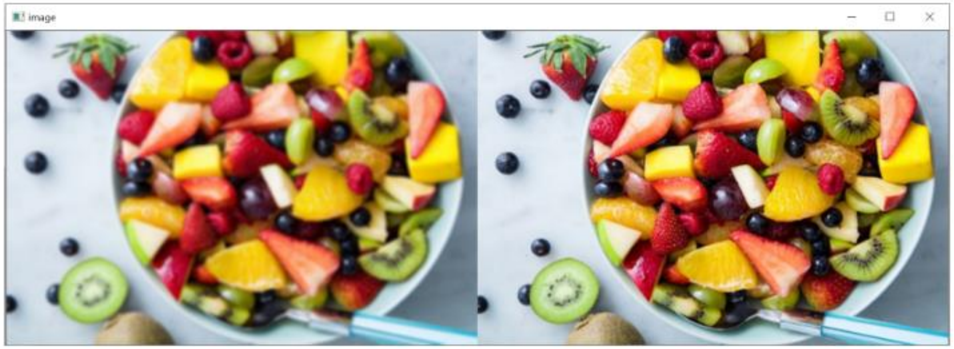
Syntax of cv2.boxFilter()

```
cv2.boxFilter(src, dst, depth, ksize, anchor, normalize, bordertype)
```

Smoothing (Averaging) Filter Masks

Example of Averaging Filter

```
import cv2
import numpy as np

# image path
path = r'salad.jpg'

# using imread()
img = cv2.imread(path)

im1 = cv2.blur(img,(5,5))
im2 = cv2.boxFilter(img, -1, (2, 2), normalize=True)

cv2.imshow('image', np.hstack((im1, im2)))
cv2.waitKey(0);
cv2.destroyAllWindows();
cv2.waitKey(1)
```



Spatial Smoothing Linear Filters

The general implementation for filtering an $\mathrm{M} \times \mathrm{N}$ image with a weighted averaging filter of size $\mathrm{m} \times \mathrm{n}$ is given

$$g(x, y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)}$$

where $m = 2a+1, \quad n = 2b+1.$

Spatial Smoothing Linear Filters

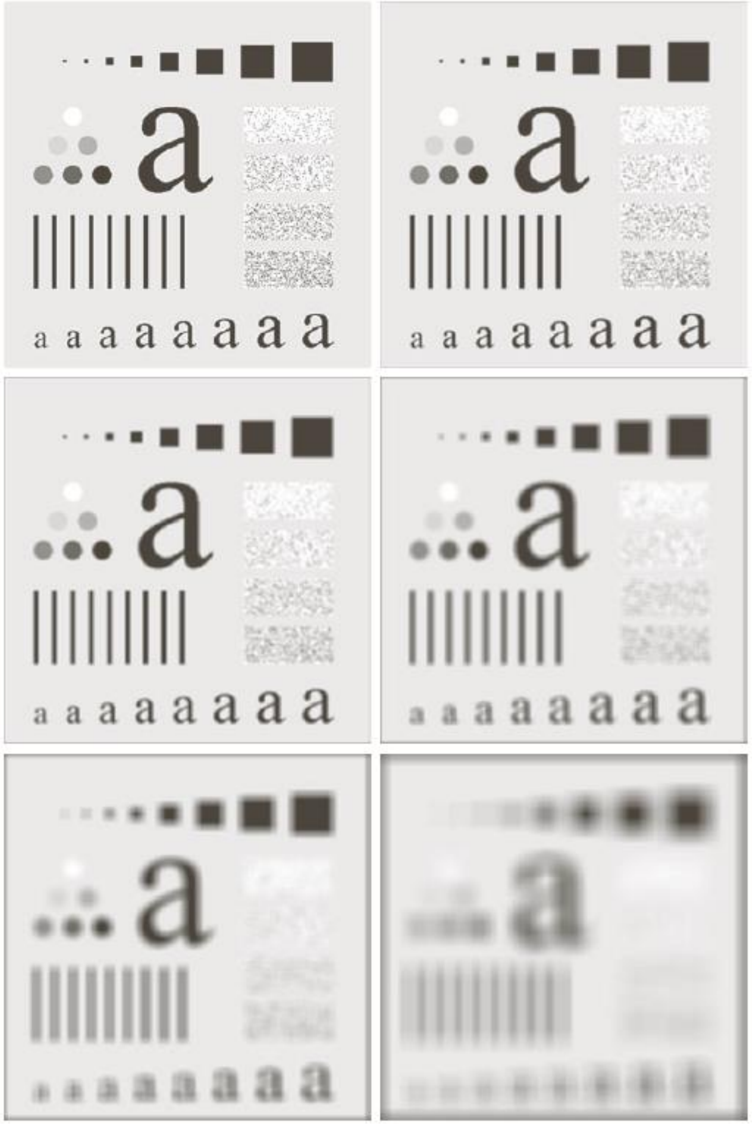
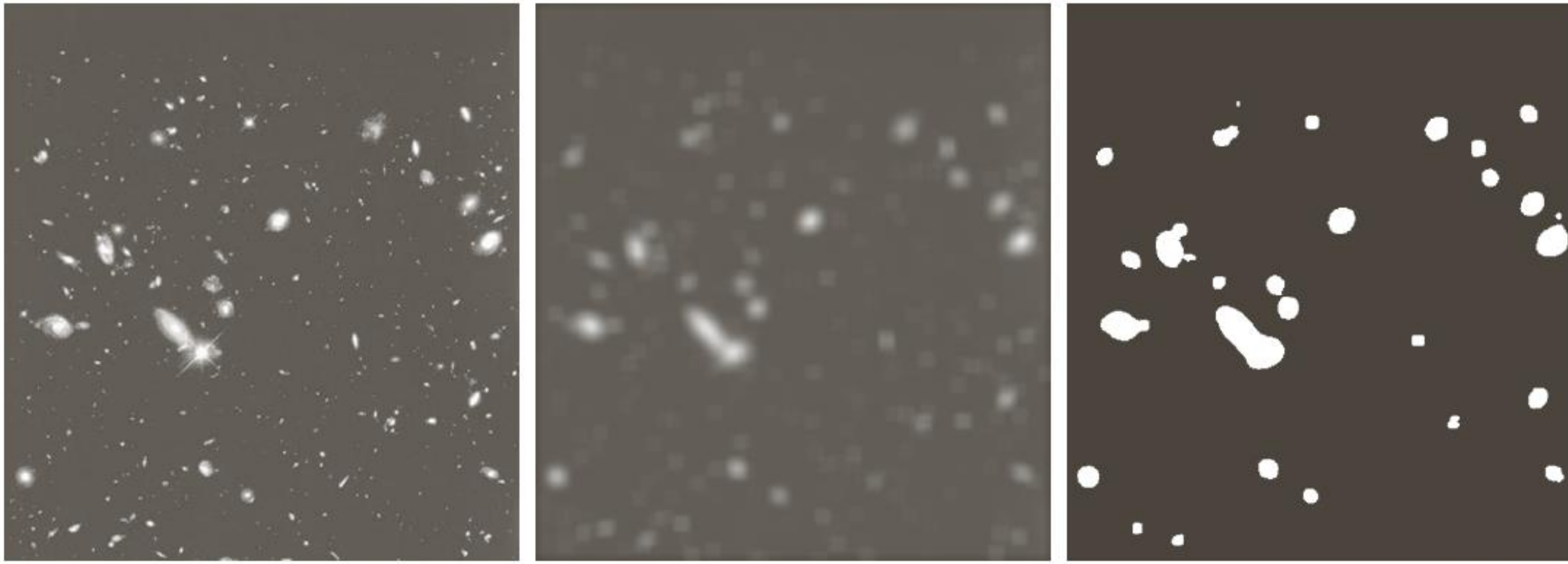


FIGURE 3.33 (a) Original image, of size 500 × 500 pixels. (b)–(f) Results of smoothing with square averaging filter masks of sizes $m = 3, 5, 9, 15$, and 35, respectively. The black squares at the top are of sizes 3, 5, 9, 15, 25, 35, 45, and 55 pixels, respectively; their borders are 25 pixels apart. The letters at the bottom range in size from 10 to 24 points, in increments of 2 points; the large letter at the top is 60 points. The vertical bars are 5 pixels wide and 100 pixels high; their separation is 20 pixels. The diameter of the circles is 25 pixels, and their borders are 15 pixels apart; their intensity levels range from 0% to 100% black in increments of 20%. The background of the image is 10% black. The noisy rectangles are of size 50 × 120 pixels.

Example: Gross Representation of Objects



a b c

FIGURE 3.34 (a) Image of size 528 × 485 pixels from the Hubble Space Telescope. (b) Image filtered with a 15 × 15 averaging mask. (c) Result of thresholding (b). (Original image courtesy of NASA.)

Order-statistic (Nonlinear) Filters

— Nonlinear

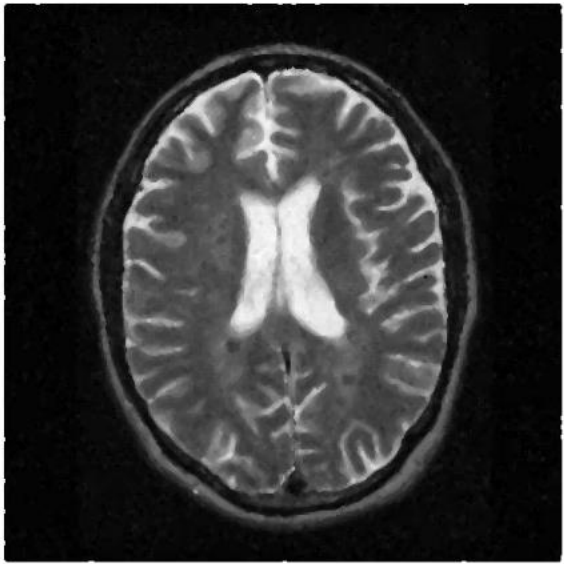
— Based on ordering (ranking) the pixels contained in the filter mask

— Replacing the value of the center pixel with the value determined by the ranking result

E.g., median filter, max filter, min filter

Order-statistic (Nonlinear) Filters

```
import cv2
import numpy as np

img = cv2.imread('brain.jpg')
median = cv2.medianBlur(img, 5)
compare = np.concatenate((img, median), axis=1) #side by side comparison

cv2.imshow('img', compare)
cv2.waitKey(0)
cv2.destroyAllWindows
```

Example: Use of Median Filtering for Noise Reduction

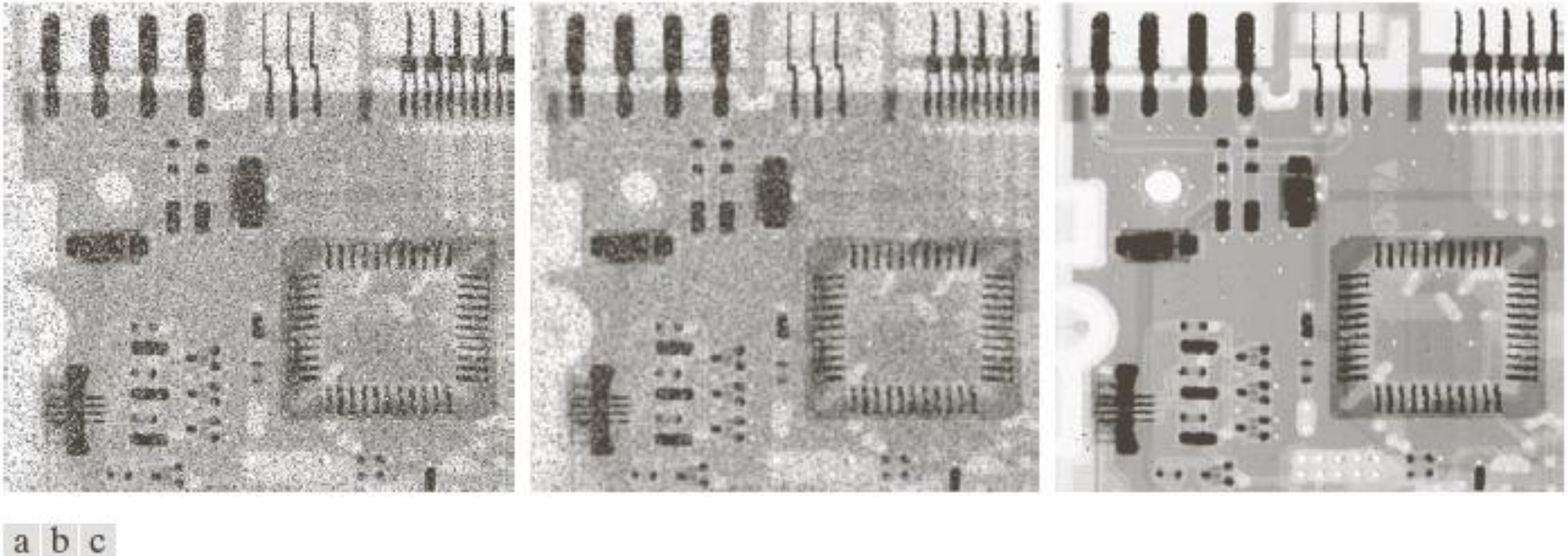


FIGURE 3.35 (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3×3 averaging mask. (c) Noise reduction with a 3×3 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

Python Code (median):

```
medianBlur(source_image, kernel_size)
```

Sharpening Spatial Filters

- The aim is to emphasize the transitions in intensity.

- Laplacian Operator

- Unsharp Masking and Highboost Filtering

- Using First-Order Derivatives for Nonlinear Image Sharpening — The Gradient

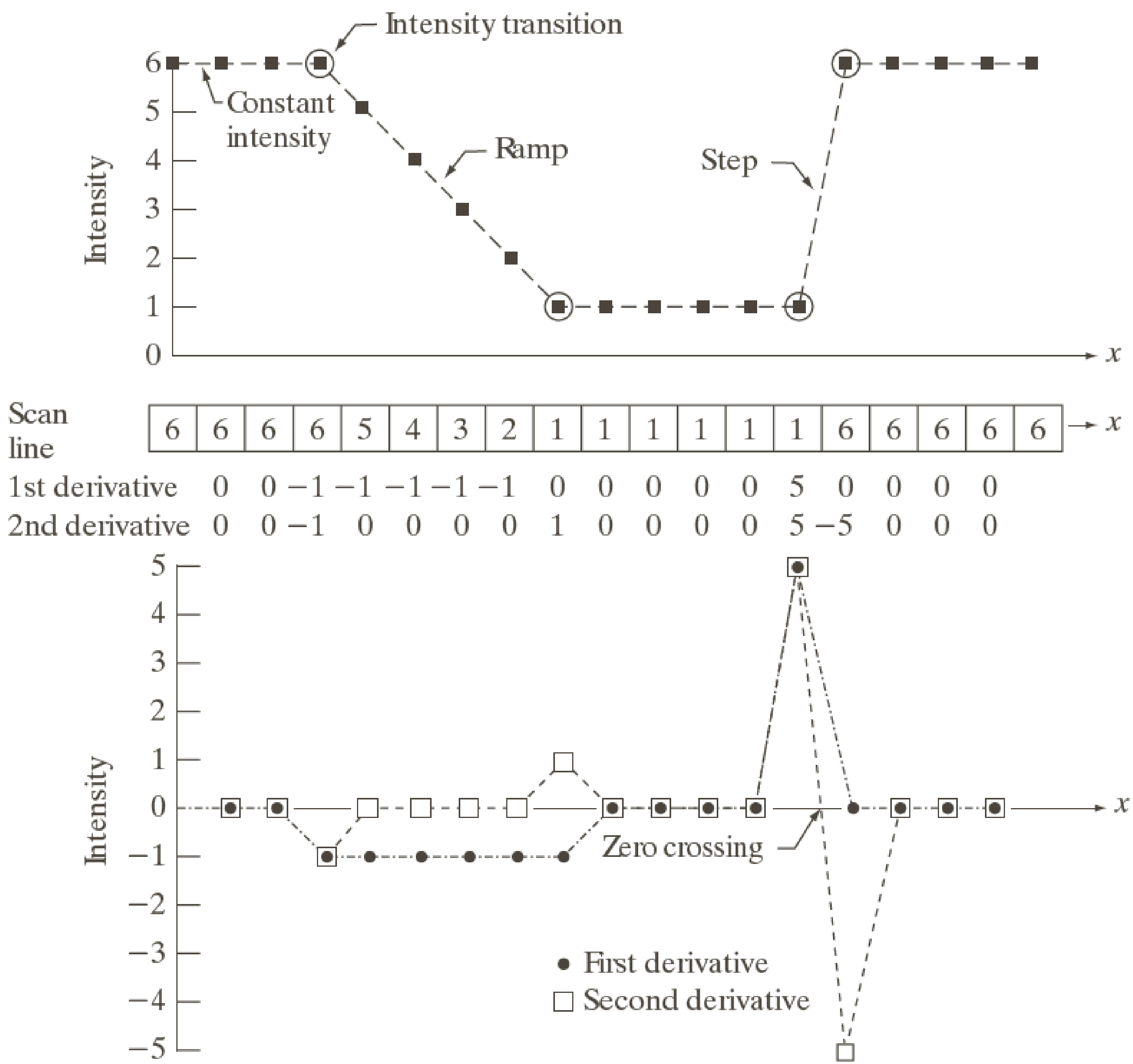
Sharpening Spatial Filters: Foundation

- The first-order derivative of a one-dimensional function f(x) is the difference

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

- The second-order derivative of f(x) as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$



a
b
c

FIGURE 3.36 Illustration of the first and second derivatives of a 1-D digital function representing a section of a horizontal intensity profile from an image. In (a) and (c) data points are joined by dashed lines as a visualization aid.

Sharpening Spatial Filters: Laplace Operator

- The second-order isotropic derivative operator is the Laplacian for a function (image) f(x,y)

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

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

Sharpening Spatial Filters: Laplace Operator

(a)

0	1	0
1	−4	1
0	1	0

(b)

1	1	1
1	−8	1
1	1	1

(c)

0	−1	0
−1	4	−1
0	−1	0

(d)

−1	−1	−1
−1	8	−1
−1	−1	−1

a	b
c	d

FIGURE 3.37 (a) Filter mask used to implement Eq. (3.6-6). (b) Mask used to implement an extension of this equation that includes the diagonal terms. (c) and (d) Two other implementations of the Laplacian found frequently in practice.

Sharpening Spatial Filters: Laplace Operator

- Image sharpening in the way of using the Laplacian:

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

where,

$f(x,y)$ is input image,

$g(x,y)$ is sharpenend images,

$c =$ -1 if $\nabla^2 f(x,y)$ corresponding to Fig. 3.37(a) or (b)

and $c = 1$ if either of the other two filters is used.

Sharpening Spatial Filters: Laplace Operator

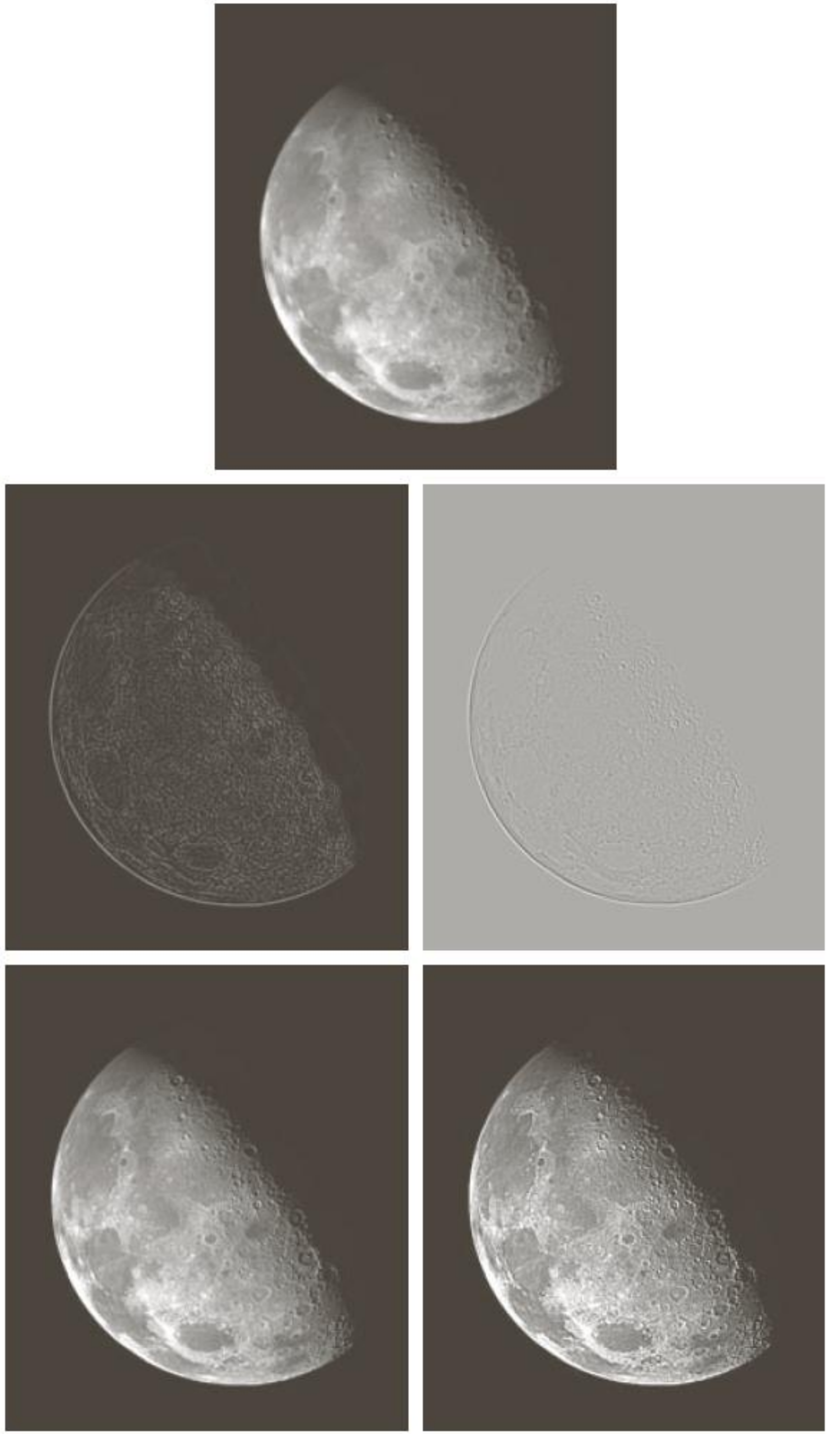
Python Code:

Laplacian(src_gray, dst, ddepth, kernel_size, scale, delta, BORDER_DEFAULT);

- *src_gray*: The input image.
- *dst*: Destination (output) image
- *ddepth*: Depth of the destination image. Since our input is *CV_8U* we define *ddepth* = *CV_16S* to avoid overflow
- *kernel_size*: The kernel size of the Sobel operator to be applied internally. We use 3 in this example.
- *scale*, *delta* and *BORDER_DEFAULT*: We leave them as default values.

Sharpening Spatial Filters: Laplace Operator

```python
"""
@file laplace_demo.py
@brief Sample code showing how to detect edges using the Laplace operator
"""
import sys
import cv2 as cv

def main(argv):
    # [variables]
    # Declare the variables we are going to use
    ddepth = cv.CV_16S
    kernel_size = 3
    window_name = "Laplace Demo"
    # [variables]

    # [load]
    imageName = argv[0] if len(argv) > 0 else 'lena.jpg'

    src = cv.imread(cv.samples.findFile(imageName), cv.IMREAD_COLOR) # Load an image

    # Check if image is loaded fine
    if src is None:
        print ('Error opening image')
        print ('Program Arguments: [image_name -- default lena.jpg]')
        return -1
    # [load]

    # [reduce_noise]
    # Remove noise by blurring with a Gaussian filter
    src = cv.GaussianBlur(src, (3, 3), 0)
    # [reduce_noise]

    # [convert_to_gray]
    # Convert the image to grayscale
    src_gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
    # [convert_to_gray]

    # Create Window
    cv.namedWindow(window_name, cv.WINDOW_AUTOSIZE)

    # [laplacian]
    # Apply Laplace function
    dst = cv.Laplacian(src_gray, ddepth, ksize=kernel_size)
    # [laplacian]

    # [convert]
    # converting back to uint8
    abs_dst = cv.convertScaleAbs(dst)
    # [convert]

    # [display]
    cv.imshow(window_name, abs_dst)
    cv.waitKey(0)
    # [display]

    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```



	a
b	c
d	e

FIGURE 3.38
(a) Blurred image of the North Pole of the moon. (b) Laplacian without scaling. (c) Laplacian with scaling. (d) Image sharpened using the mask in Fig. 3.37(a). (e) Result of using the mask in Fig. 3.37(b). (Original image courtesy of NASA.)

Image Sharpening based on First-Order Derivatives

For function $f(x, y)$, the gradient of f at coordinates (x, y) is defined as

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}$$

The *magnitude* of vector ∇f, denoted as $M(x, y)$

Gradient Image

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^{\ 2} + g_y^{\ 2}}$$

Image Sharpening based on First-Order Derivatives

The *magnitude* of vector ∇f, denoted as $M(x, y)$

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{{g_x}^2 + {g_y}^2}$$

$$M(x, y) \approx |g_x| + |g_y|$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

$$M(x, y) = |z_8 - z_5| + |z_6 - z_5|$$

Image Sharpening based on First-Order Derivatives

(a)

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

(b)

−1	0
0	1

(c)

0	−1
1	0

(d)

−1	−2	−1
0	0	0
1	2	1

(e)

−1	0	1
−2	0	2
−1	0	1

a
b c
d e

FIGURE 3.41 A 3×3 region of an image (the zs are intensity values). (b)–(c) Roberts cross gradient operators. (d)–(e) Sobel operators. All the mask coefficients sum to zero, as expected of a derivative operator.

Image Sharpening based on First-Order Derivatives

Roberts Cross-gradient Operators

$$M(x, y) \approx |z_9 - z_5| + |z_8 - z_6|$$

Sobel Operators

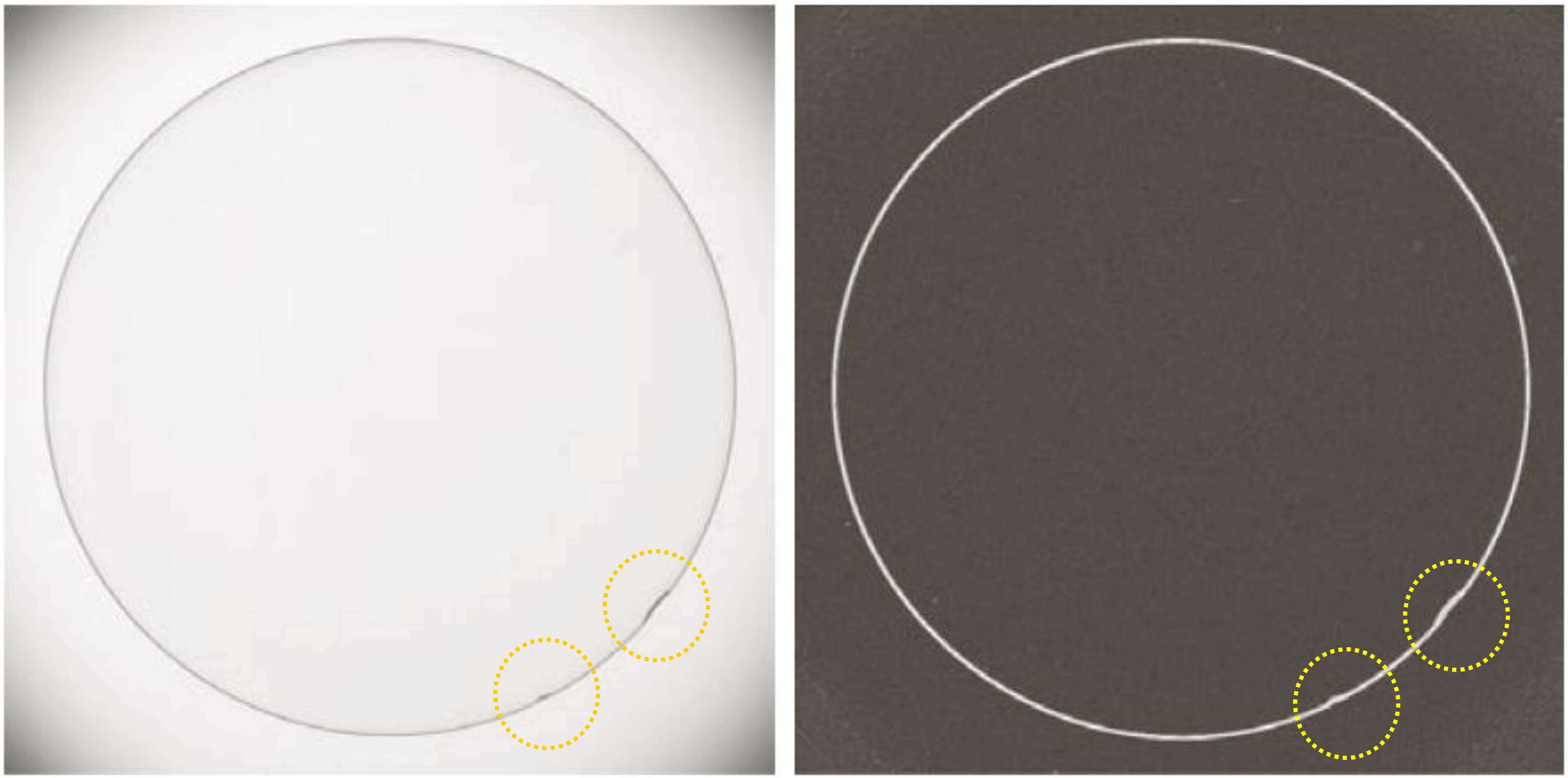
$$M(x, y) \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)|$$
$$+ |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Example

a b

FIGURE 3.42
(a) Optical image of contact lens (note defects on the boundary at 4 and 5 o'clock). (b) Sobel gradient. (Original image courtesy of Pete Sites, Perceptics Corporation.)

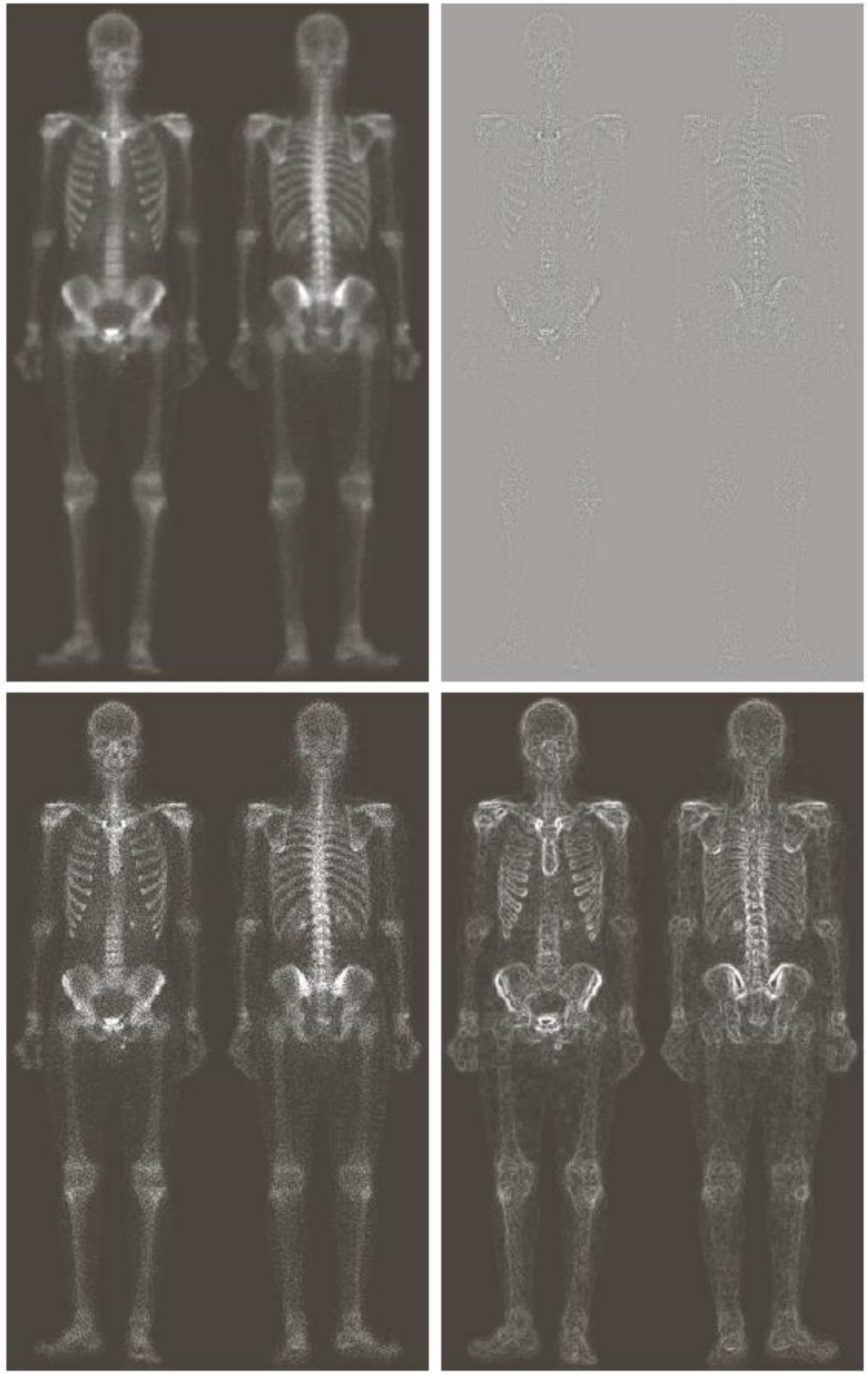


Example:

Combining Spatial Enhancement Methods

Goal:

Enhance the image by sharpening it and by bringing out more of the skeletal detail



a b
c d

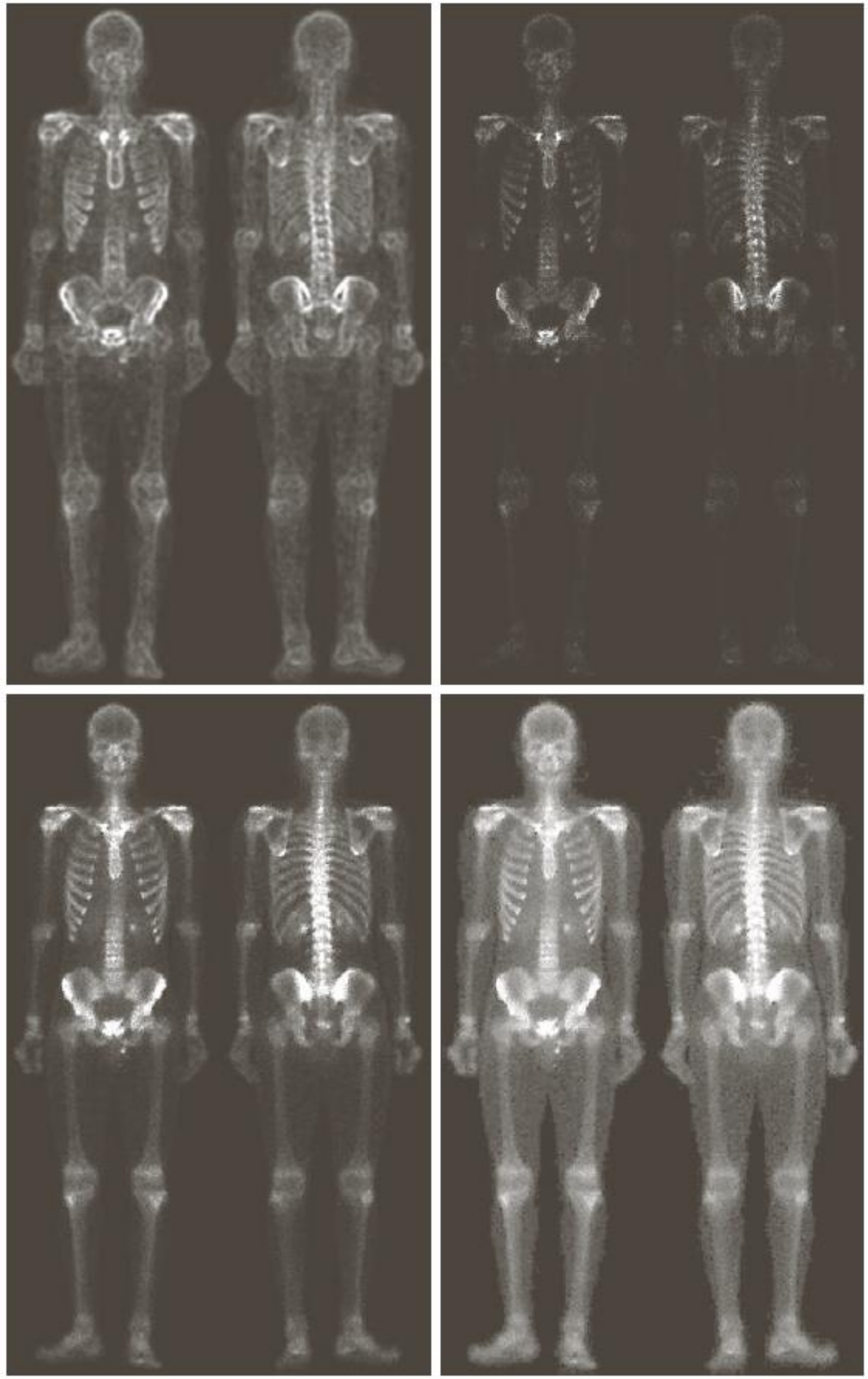
FIGURE 3.43 (a) Image of whole body bone scan. (b) Laplacian of (a). (c) Sharpened image obtained by adding (a) and (b). (d) Sobel gradient of (a).

Example:

Combining Spatial Enhancement Methods

Goal:

Enhance the image by sharpening it and by bringing out more of the skeletal detail



e f
g h

FIGURE 3.43 *(Continued)* (e) Sobel image smoothed with a 5 × 5 averaging filter. (f) Mask image formed by the product of (c) and (e). (g) Sharpened image obtained by the sum of (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare (g) and (h) with (a). (Original image courtesy of G.E. Medical Systems.)

References

- Sayısal Görüntü İşleme, Palme Publishing, Third Press Trans. (*Orj: R.C. Gonzalez and R.E. Woods: "Digital Image Processing", Prentice Hall, 3rd edition, 2008*).
- "Digital Image Processing Using Matlab", Gonzalez & Richard E. Woods, Steven L. Eddins, Gatesmark Publishing, 2009
- Lecture Notes, CS589-04 Digital Image Processing, Frank (Qingzhong) Liu, http://www.cs.nmt.edu/~ip
- Lecture Notes, BIL717-Image Processing, Erkut Erdem
- Lecture Notes, EBM537-Image Processing, F.Karabiber https://docs.opencv.org/
- https://www.geeksforgeeks.org/python-opencv-filter2d-function/
- https://medium.com/@florestony5454/median-filtering-with-python-and-opencv-2bce390be0d1
- Bekir Aksoy, Python ile İmgeden Veriye Görüntü İşleme ve Uygulamaları, Nobel Akademik Yayıncılık