AI 기반 영상 데이터 분석 실습
- Day 2 -

Course overview

Course overview

Day	Morning session (이론)	Afternoon Lab session (실습)
Jan. 19 (Mon)	• 인공지능 개요 • Vision task 소개 • MNIST Classification review	• GitHub 활용법 • 주제 선정 • 데이터 수집
Jan. 20 (Tue)	• 영상처리 기초 • 이미지 전처리 기법	• 전처리, 증강, 시각화 • Dataset split • Data set class & Data loading
Jan. 21 (Wed)	• Fundamentals on CNN • Basic architectures	• CNN architecture 구현 (Resnet)
Jan. 22 (Thu)	• Weight update (Gradient descent, Optimizers) • Loss monitoring	• Torch model • 아키텍처 선택 및 구현 (Resnet + @) • 모델 학습 및 하이퍼파라미터 튜닝
Jan. 26 (Mon)	• Model evaluation • Grad-CAM	• 모델 평가 • Grad-CAM 실습
Jan. 27 (Tue)	• Github에 프로젝트 업로드 • 프로젝트 발표	

Digital image fundamentals

Elements of Visual Perception

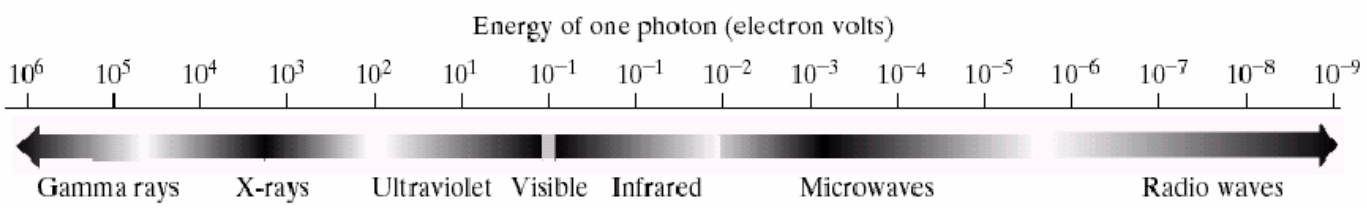
Image

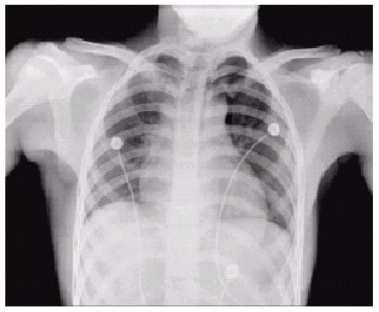
- **General meaning**
  - A representation of optical phenomena in the natural world, captured through sensing visible light and expressed as data in two or more dimensions.

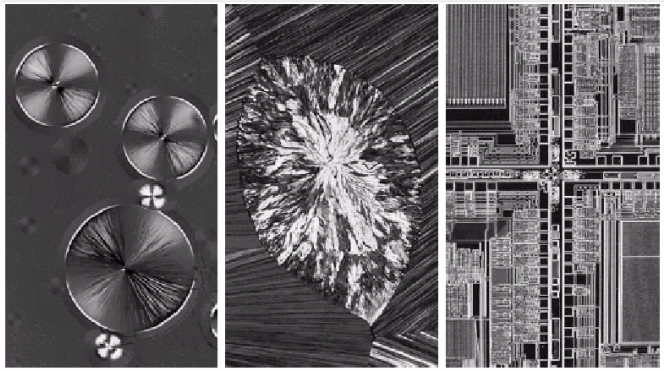
Elements of Visual Perception

Image

- **Broad Definition of Image**
  - An image created by sensing beyond the visible spectrum or by using computer graphics to visualize waves or signals.
  - Examples:
    - X-ray: Used in medical imaging, airport security, etc.
    - Infrared Imaging: Employed for various purposes in military, aviation,







Elements of Visual Perception

Electromagnetic spectrum

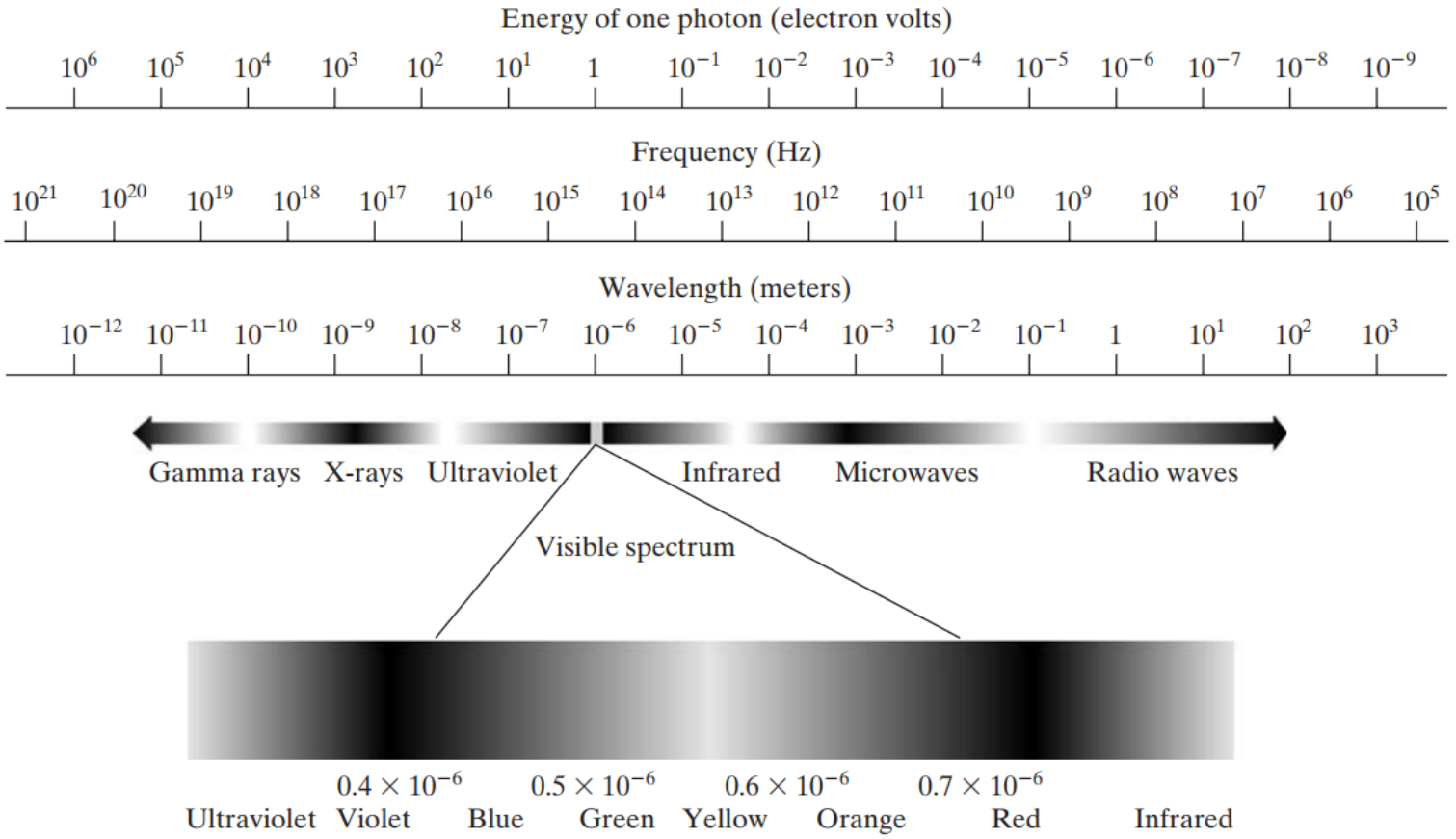


FIGURE 2.10 The electromagnetic spectrum. The visible spectrum is shown zoomed to facilitate explanation, but note that the visible spectrum is a rather narrow portion of the EM spectrum.

Elements of Visual Perception

Electromagnetic spectrum

- **Light: the visible spectrum**
  - Visible range: 0.43μm(violet)-0.78μm(red)
  - Six bands: violet, blue, green, yellow, orange, red
  - The color of an object is determined by the nature of the light reflected by the object
  - Monochromatic light (Light that is void of color)
    → intensity(gray level): only attribute

Elements of Visual Perception

Human eye structure

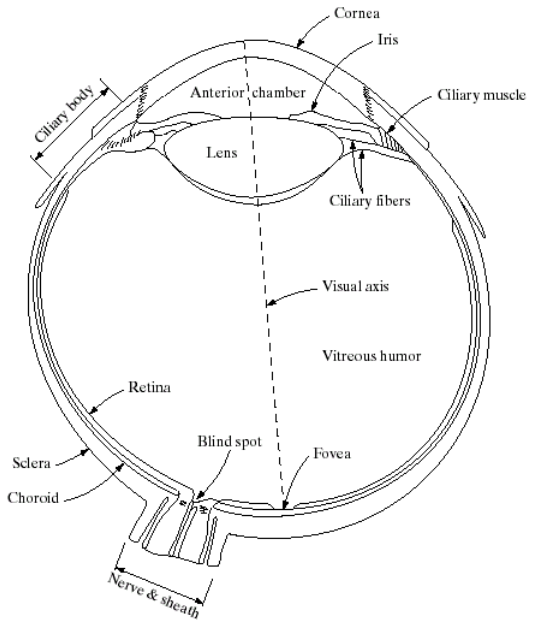


FIGURE 2.1 Simplified diagram of a cross section of the human eye.

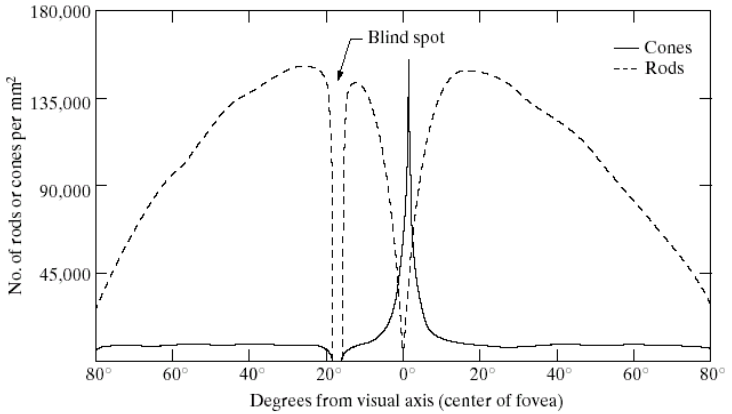


FIGURE 2.2 Distribution of rods and cones in the retina.

Cone: 6-7 million in each eye, central part of retina (fovea) and highly sensitive to color
Rod: 75-150 million, all over the retina surface and sensitive to low levels of illumination

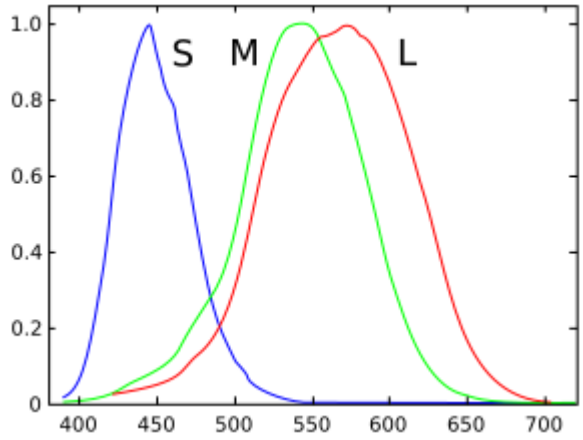
Cornea – 각막, Iris – 홍채, Lens – 수정체, Retina – 망막

When the eye is properly focused, light from an outside object is imaged on the retina

Elements of Visual Perception

Human eye structure

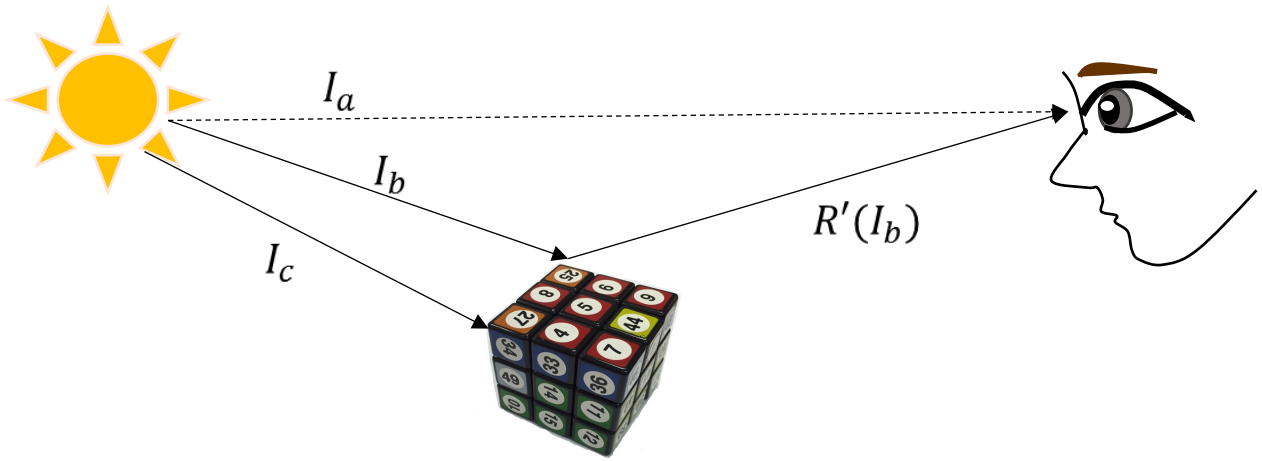
- L-cones (Long-wavelength): Primarily sensitive to longer wavelengths of light, roughly corresponding to red.
- M-cones (Medium-wavelength): Primarily sensitive to medium wavelengths, roughly corresponding to green.
- S-cones (Short-wavelength): Primarily sensitive to shorter wavelengths, roughly corresponding to blue.



Elements of Visual Perception

Image formation in the Eye

- The process by which spatial information from the real world is projected into the human eye.



- I_a: Light from a source directly to the eye
- I_b, I_c: Light from a source to object
- $R'(I_c)$: A portion of the light reflected from an object and reaches the eye

- Camera sensors are designed to mimic the human retina in sensing images.

Elements of Visual Perception

Image formation in the Eye

- The process by which spatial information from the real world is projected into the human eye.

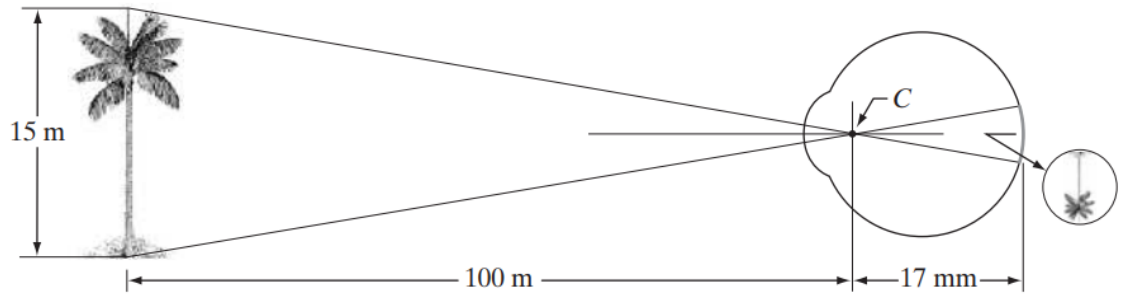


FIGURE 2.3 Graphical representation of the eye looking at a palm tree. Point *C* is the optical center of the lens.

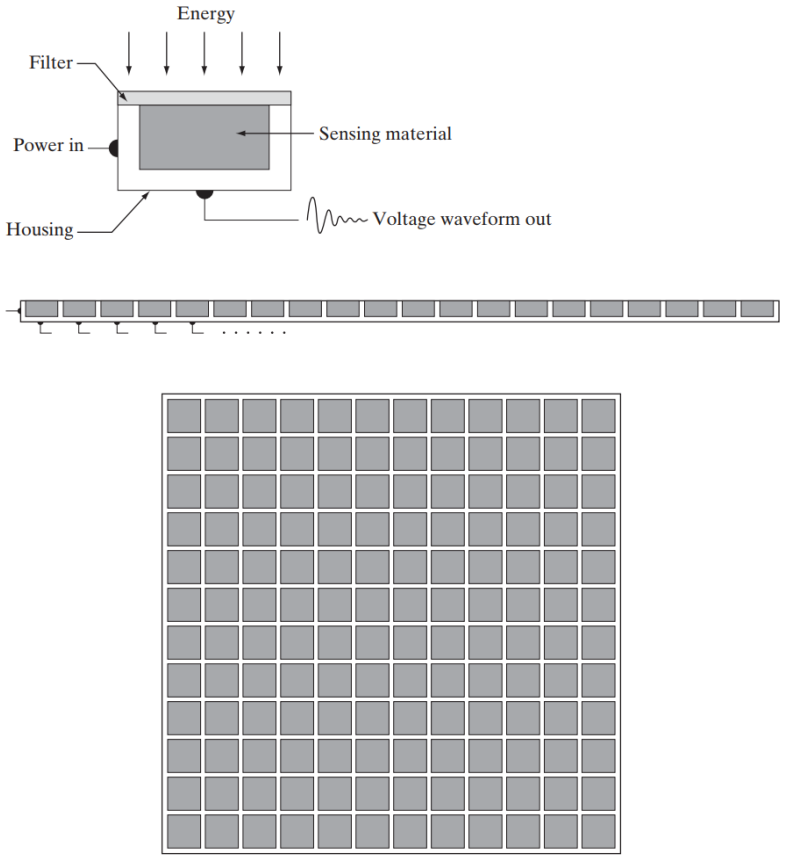
  - Inverted Image as the Default State
  - The eye adjusts focal length (position C) by changing the shape of the lens.
  - In a relaxed state (when viewing objects over 3 meters away), the human eye's focal length is approximately 14–17mm.
  - In the image above, the size of the tree projected on the retina is around 2.55mm.

$$100m : 15m = 17mm : H\ mm$$

Image sensing and acquisition

Image sensing and acquisition

- **Imaging sensor arrangements**



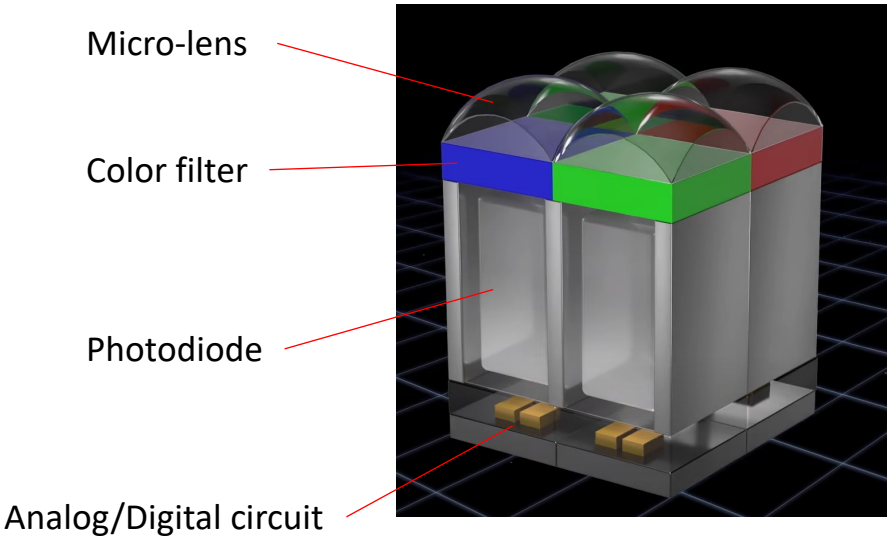
a
b
c

FIGURE 2.12
(a) Single imaging sensor.
(b) Line sensor.
(c) Array sensor.

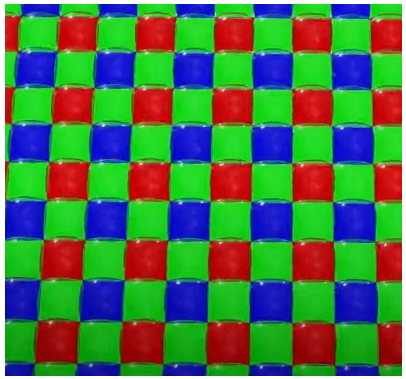
Image sensing and acquisition

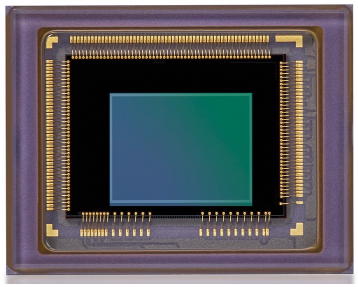
Image sensing and acquisition

- **Image sensor**
  - Mimicking human eye



<CMOS image sensor>



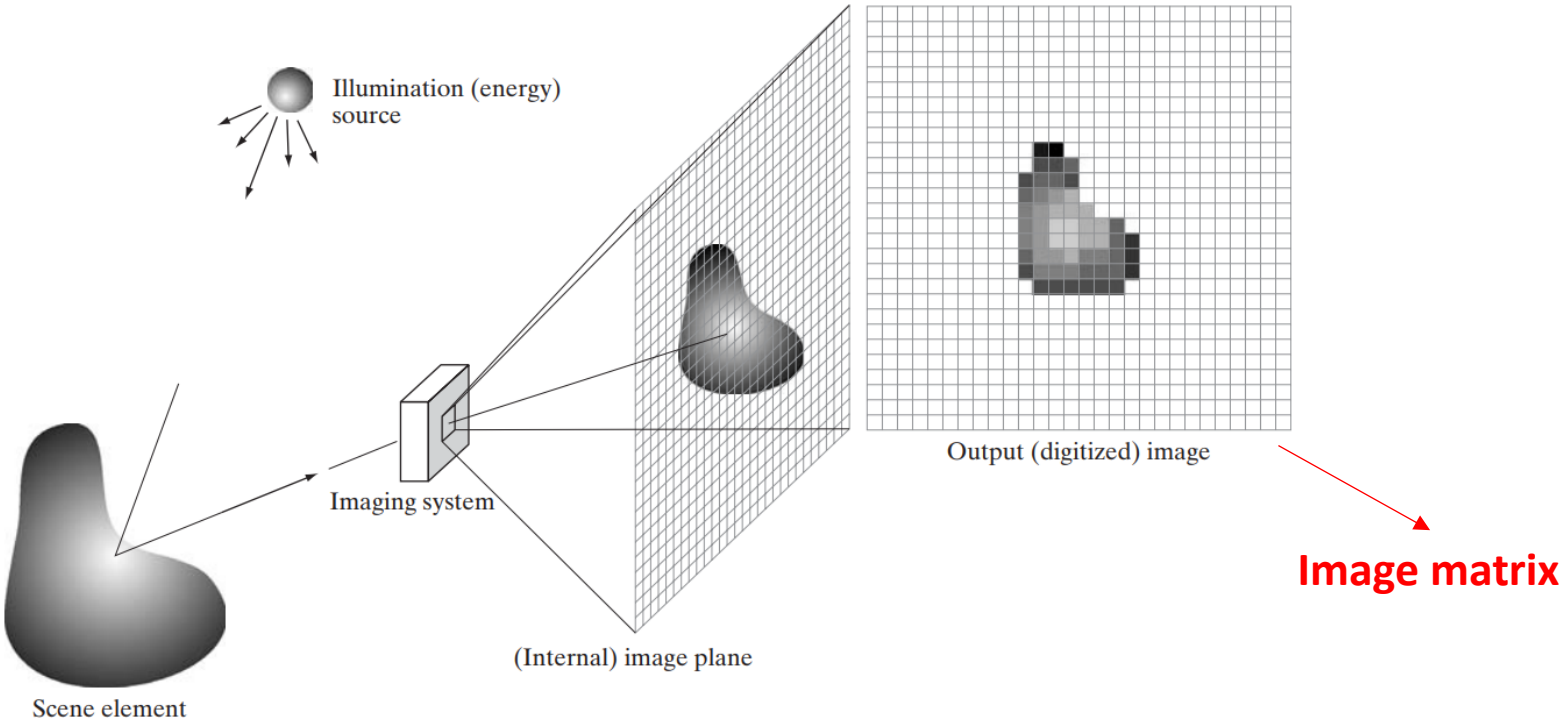


Sony YouTube(https://www.youtube.com/@Sony)

Image sensing and acquisition

Image sensing and acquisition

- Image acquisition process



a b c d e

FIGURE 2.15 An example of the digital image acquisition process. (a) Energy ("illumination") source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

Image sensing and acquisition

Image sensing and acquisition

- Image acquisition result

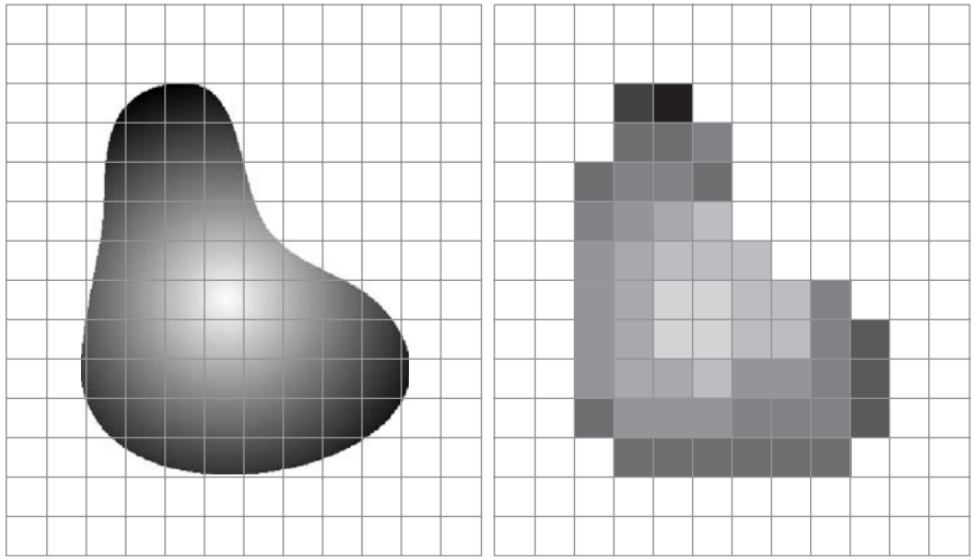


Image:
Discrete space!

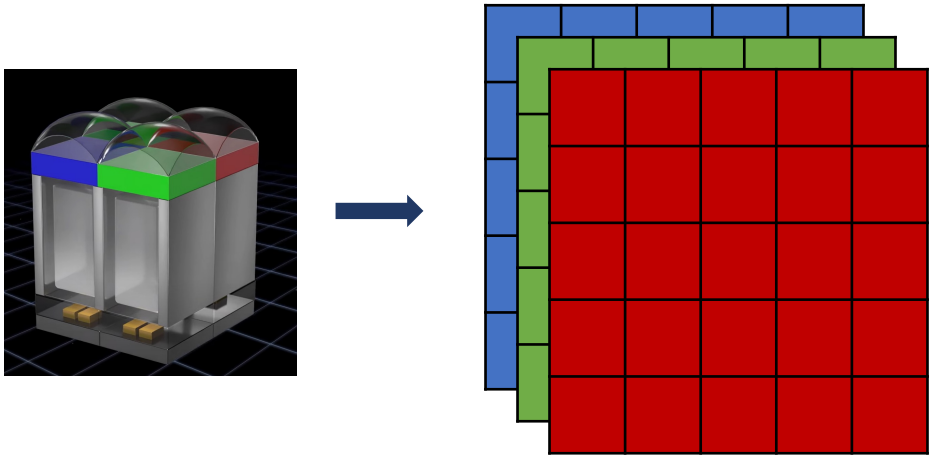
a b

FIGURE 2.17 (a) Continuous image projected onto a sensor array. (b) Result of image sampling and quantization.

Image sensing and acquisition

Natural (color) image

- Composed of three 2D matrices representing color channels.
  - R, G, B channels



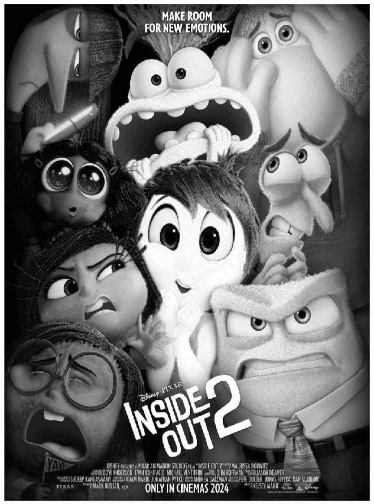
  - Shape of image matrix: (Width) x (Height) x (Channel) → 3D

- Each channel captures intensity values for its respective color, and when combined, they create the full-color image.

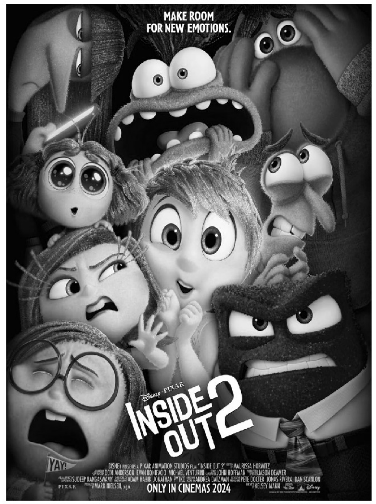
Image sensing and acquisition

Natural (color) image

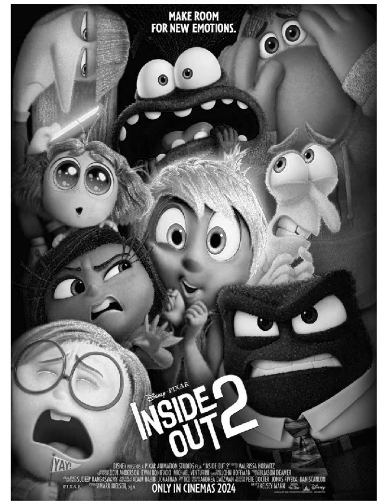
Red channel



Green channel



Blue channel



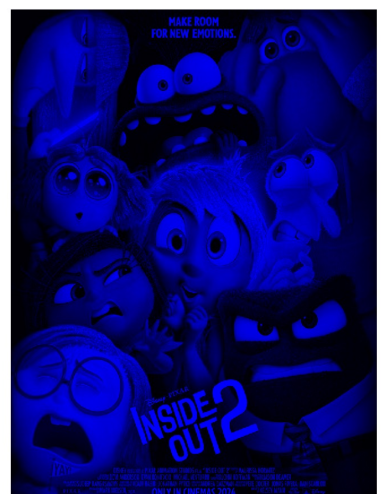


Image sensing and acquisition

In short,

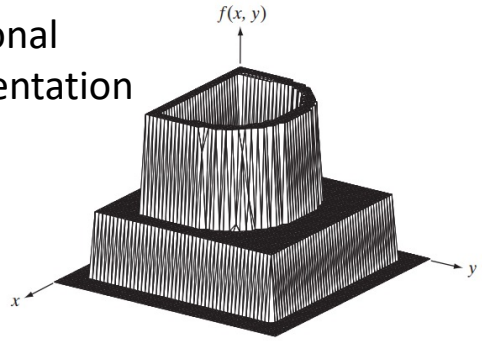
"An image is a matrix"

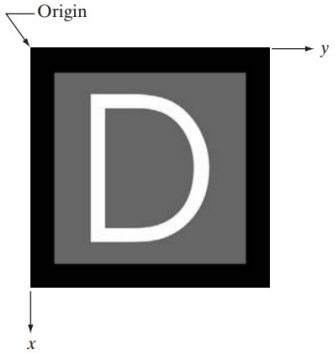
- Element of matrix = Pixel
- Numbers in matrix = Brightness of each pixel (or location)

Representing Digital Images

Representing Digital Images

Functional representation





Visual intensity

Numerical array (Matrix)

a
b c

FIGURE 2.18
(a) Image plotted as a surface.
(b) Image displayed as a visual intensity array.
(c) Image shown as a 2-D numerical array (0, .5, and 1 represent black, gray, and white, respectively).

Visual intensity

- A monitor or printer simply converts these three values to black, gray, or white, respectively, as Fig. 2.18(b) shows.

Numerical array

- When developing algorithms, however, this representation is quite useful when only parts of the image are printed and analyzed as numerical values.

Representing Digital Images

Representing Digital Images

Thus, we normally express image as numerical array

Pixel

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \quad (2.4\text{-}1)$$

Image

- Intensity level
  - Due to storage and quantizing hardware considerations, the number of intensity levels typically is an integer power of 2.

$$L = 2^k$$

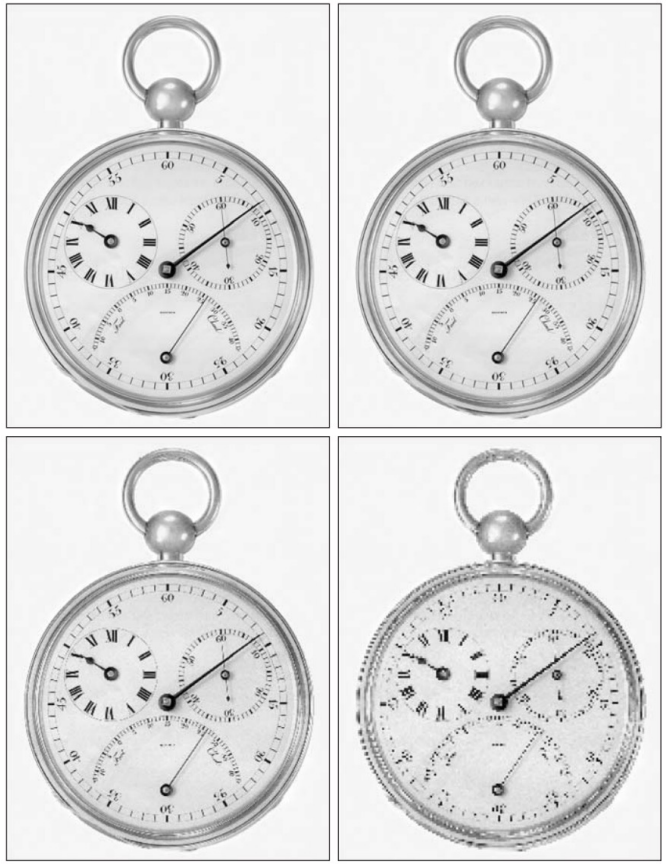
  - Dynamic range: [0,L-1]

- Total bit - $M \times N \times k$

Spatial and Intensity Resolution

Spatial resolution

- The smallest distinguishable elements or features that can be visually resolved
  - pixels per unit distance
    - (e.g. 3 pixel per 1mm)
  - Line pairs per unit distance
    - (e.g. 100 line pair per 1 mm)
  - US measure – dots per inch (dpi)
  - Sometimes.. Physical size of pixel
    - 3x3mm per pixel
      → Pixel spacing



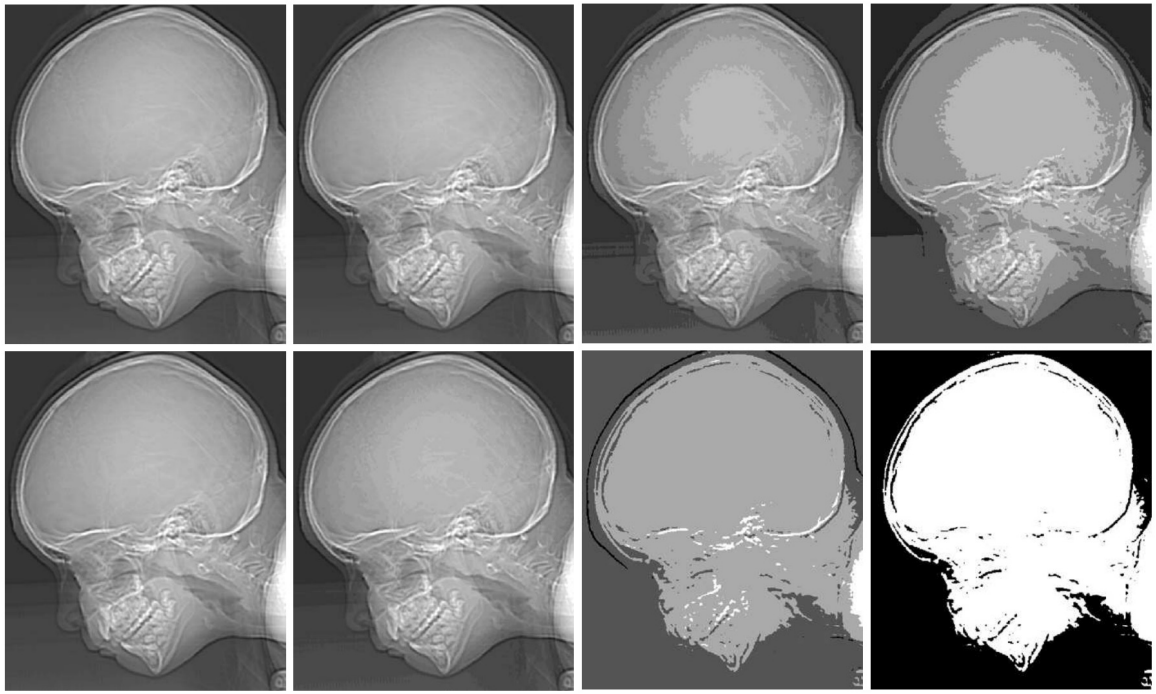
a b
c d

FIGURE 2.20 Typical effects of reducing spatial resolution. Images shown at: (a) 1250 dpi, (b) 300 dpi, (c) 150 dpi, and (d) 72 dpi. The thin black borders were added for clarity. They are not part of the data.

Spatial and Intensity Resolution

Intensity resolution (Bit-depth resolution)

- The smallest distinguishable elements or features that can be visually resolved
  - Smallest discernible change in intensity level
  - it is common practice to refer to the number of bits used to quantize intensity as the *intensity resolution*
  - Most common intensity level: 8bit (256 level)



Spatial and Intensity Resolution

Intensity resolution (Bit-depth resolution)

- The smallest distinguishable elements or features that can be visually resolved
  - Smallest discernible change in intensity level
  - it is common practice to refer to the number of bits used to quantize intensity as the *intensity resolution*
  - Most common intensity level: 8bit (256 level)

(Original – 8bit(256lv)/channel)

(3bit(8lv)/channel)

(2bit(4lv)/channel)

Basic image processing methods

Image processing methods

Index

- Image interpolation
- Intensity transformation – Spatial filtering (Image filtering)
    - Smoothing
    - Edge detection and Sharpening
- Spatial transformation
    - Translation
    - Rotation
    - Scaling
    - Affine transformation
- ETC
    - Crop
    - Reflection (Flip)

Interpolation

Image interpolation

Image Interpolation – zoom, shrink, rotation etc.

- Zooming and shrinking image (resampling)

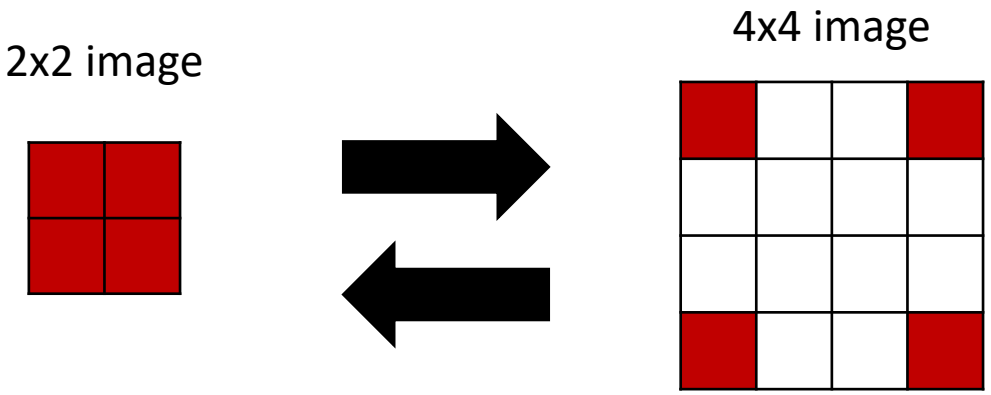
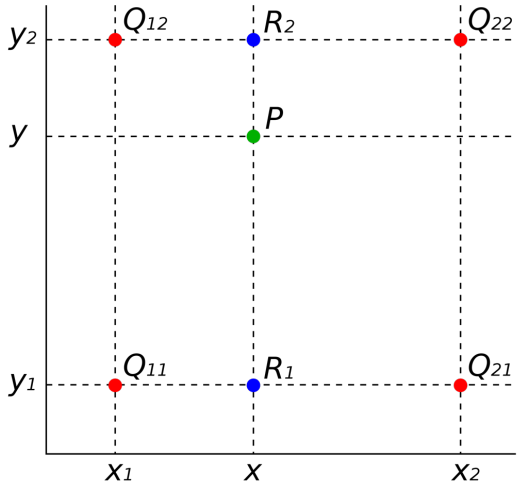


Image interpolation

Image Interpolation – zoom, shrink, rotation etc.

- Fundamentally, interpolation is the process of using known data to estimate values at unknown locations



Nearest neighbor interpolation

- It assigns to each new location the intensity of its nearest neighbor in the original image

Bilinear interpolation

- Linearly estimate new location's intensity
- Using 4 neighbor points

$$v(x, y) = ax + by + cxy + d$$

Bicubic interpolation

- Using 16 neighbor point

$$v(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

Image interpolation

Exercise

Nearest neighbor interpolation

2x2 image

1	2
3	4

4x4 image

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Image interpolation

Exercise

Bilinear interpolation

Step:

1. Approximate the R_1 using a linear function of x between points Q_{11} and Q_{21}
2. Approximate the R_2 using a linear function of x between points Q_{21} and Q_{22}
3. Approximate the P using linear function of y between points R_1 and R_2

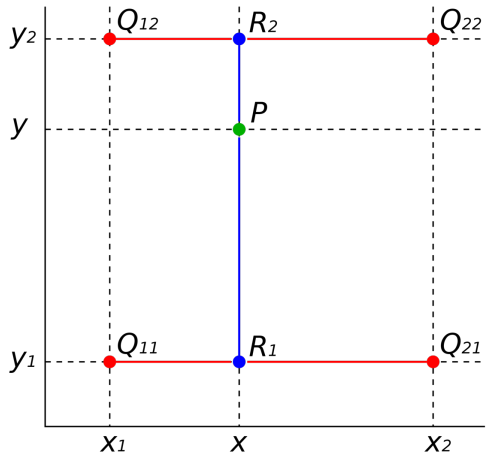


Image interpolation

Exercise

Bilinear interpolation

2x2 image

1	4
4	7

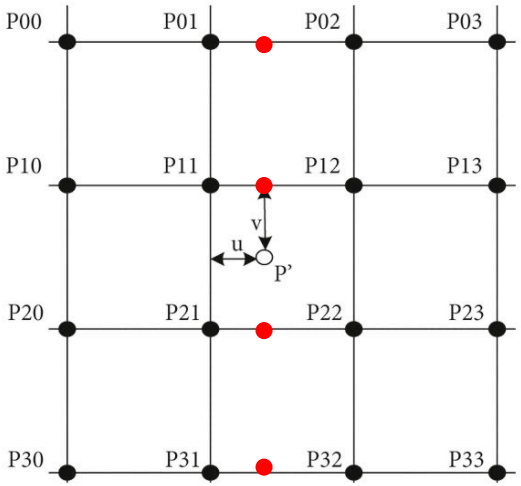
4x4 image

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

Image interpolation

Bicubic interpolation

1. Approximate R_0 using a cubic function of x between points P_{00} ~ P_{03}.
2. Approximate R_1 using a cubic function of x between points P_{10} ~ P_{13}.
3. Approximate R_2 using a cubic function of x between points P_{20} ~ P_{23}.
4. Approximate R_3 using a cubic function of x between points P_{30} ~ P_{33}.
5. Approximate P' using a cubic function of y between points R_0 and R_3.



Red dots: R_0, R_1, R_2, R_3

Image interpolation

Summary

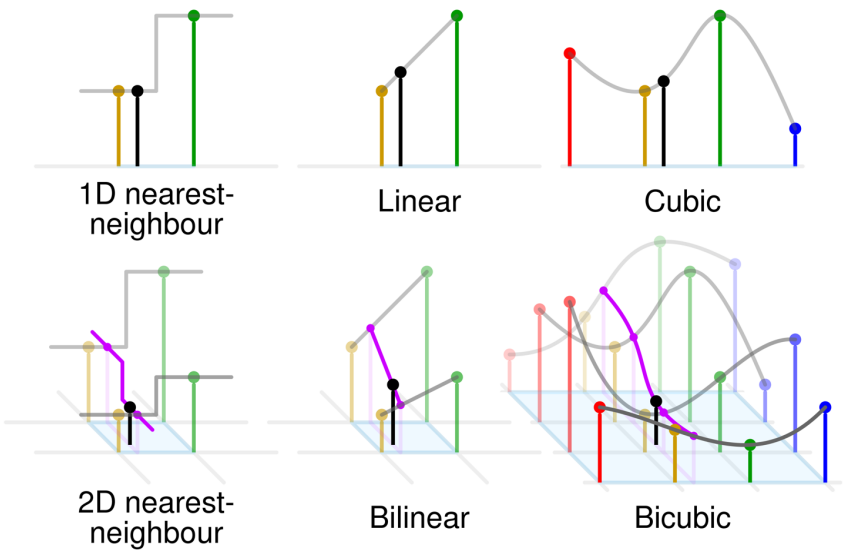
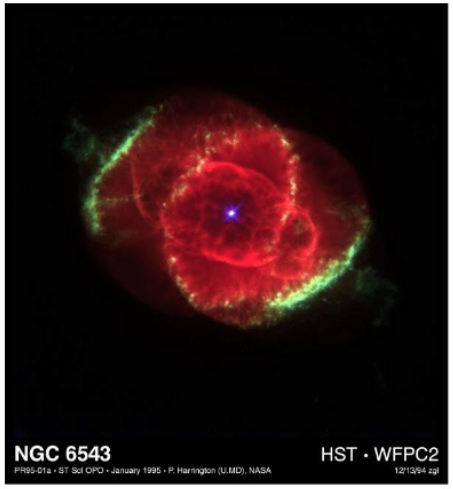


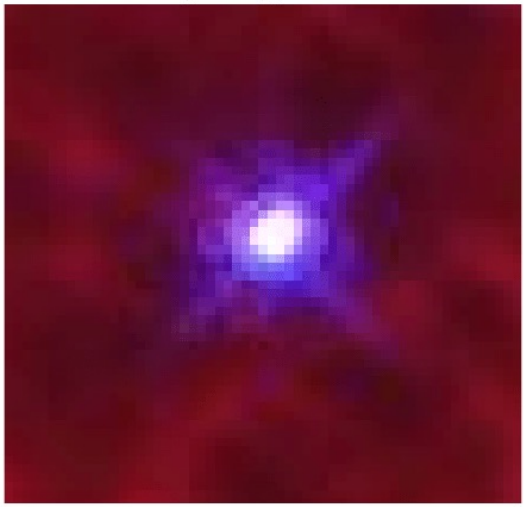
Image interpolation

Zoom with interpolation

Lower Resolution



Original Image, 10x Zoom

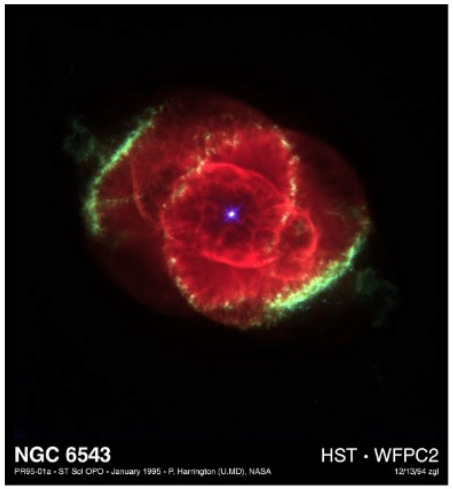


Linear method

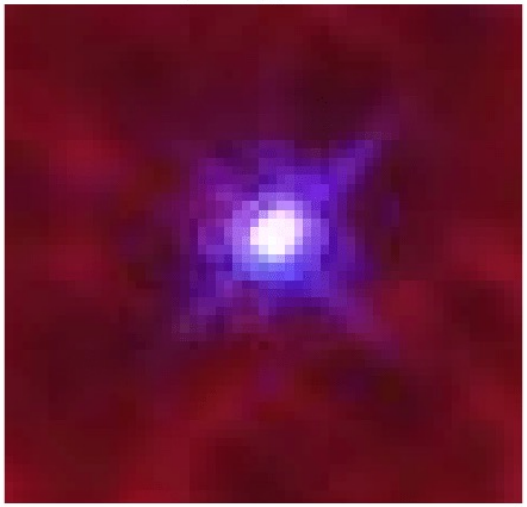
Image interpolation

Zoom with interpolation

Lower Resolution



Original Image, 10x Zoom



Cubic method

Image interpolation

Check the effect of each interpolation

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import resize
from skimage import data

# Load sample data
img = data.coffee() / 255.0

# Downsampling -> Interpolation
img_resampled = img[::2,::2,:] # down sampling

new_shape = (img_resampled.shape[0]*2, img_resampled.shape[1]*2)

img_nn  = resize(img_resampled, new_shape, order=0) # Nearest Neighbor
img_lin = resize(img_resampled, new_shape, order=1) # Bi-linear
img_bic = resize(img_resampled, new_shape, order=3) # Bi-cubic
```

```
# Visualization
plt.figure(figsize=(8, 6))

plt.subplot(2, 2, 1)
plt.title("Original")
plt.imshow(img, cmap="gray", interpolation=None)
plt.axis("off")

plt.subplot(2, 2, 2)
plt.title("Nearest (order=0)")
plt.imshow(img_nn, cmap="gray", interpolation=None)
plt.axis("off")

plt.subplot(2, 2, 3)
plt.title("Bilinear (order=1)")
plt.imshow(img_lin, cmap="gray", interpolation=None)
plt.axis("off")

plt.subplot(2, 2, 4)
plt.title("Bicubic (order=3)")
plt.imshow(img_bic, cmap="gray", interpolation=None)
plt.axis("off")

plt.tight_layout()
plt.show()
```

Image operation 함수 쓸 때 주의점:

- Input data type 제약이 있는가?
- Input data value range 제약이 있는가?
- Output data type은 어떻게 되는가?
- Output data value range는 어떻게 되는가?

Image interpolation

Check the effect of each interpolation

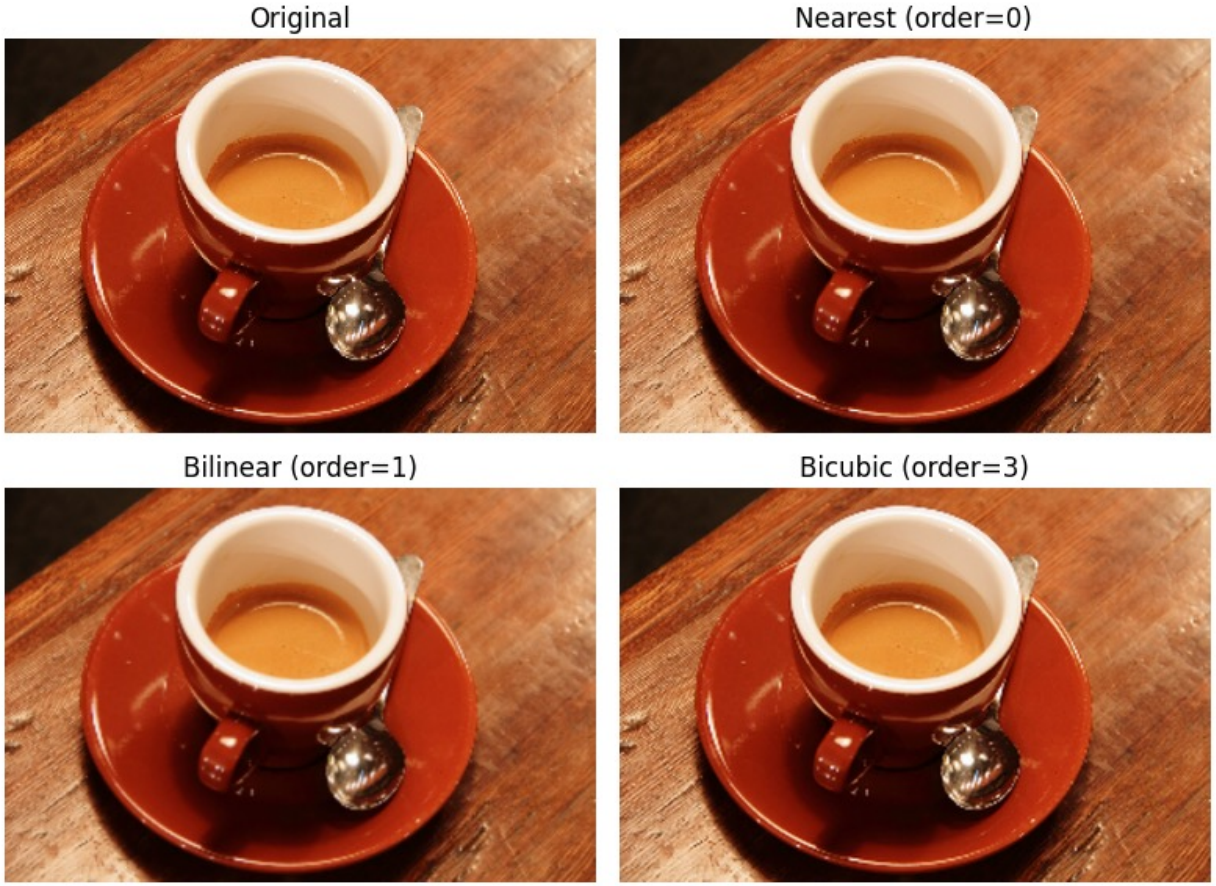
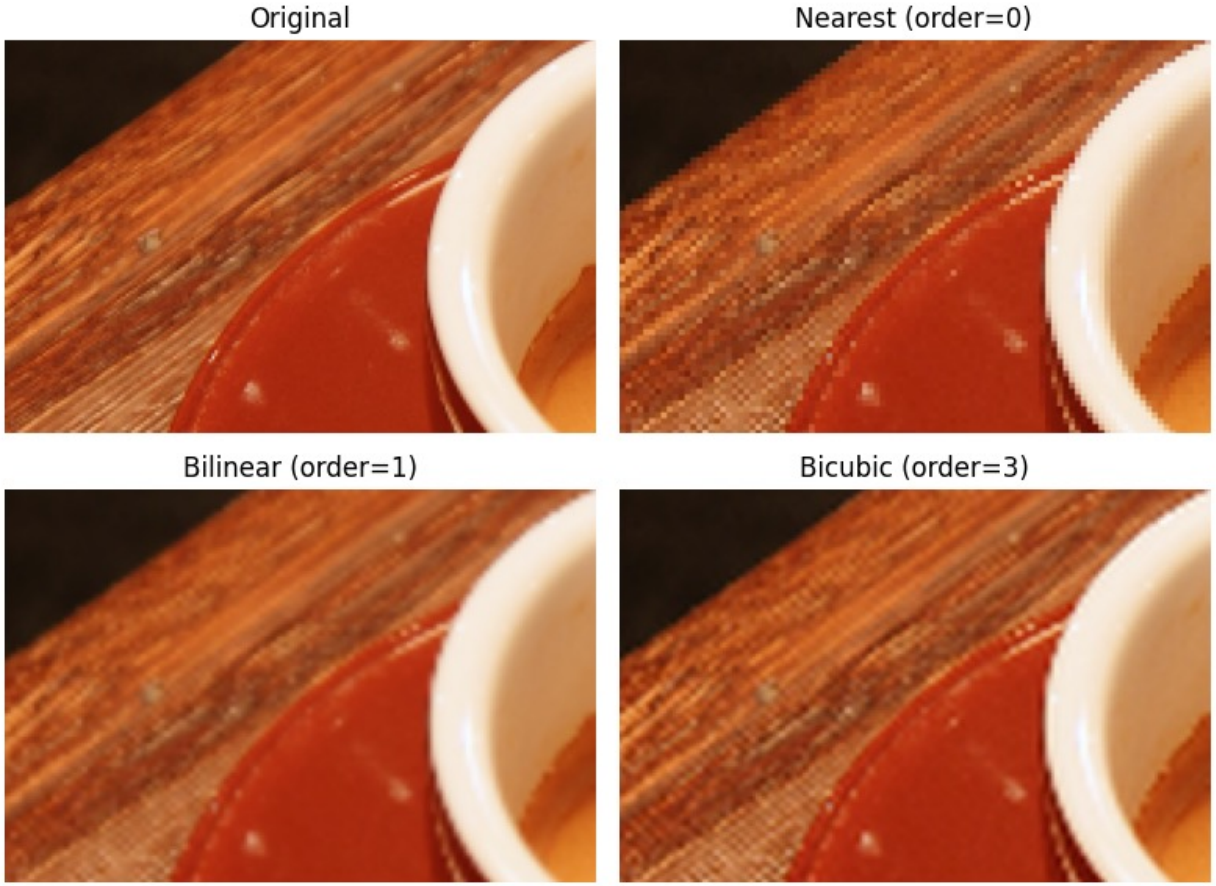


Image interpolation

Check the effect of each interpolation

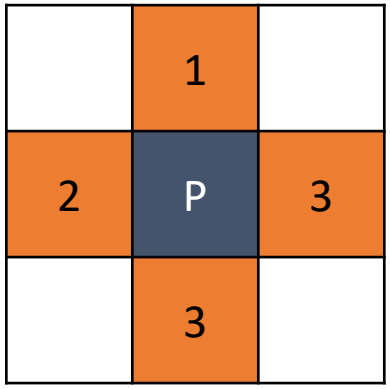


Spatial filtering

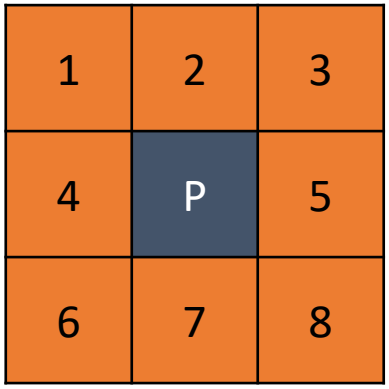
Neighbor of a Pixel

Neighbor of a Pixel

4-Neighbor



8-Neighbor



$$N_4(p) = \{(i-1,j),(i+1,j),(i,j-1),(i,j+1)\}$$

$$N_8(p) = \{(i-1,j),(i+1,j),(i,j-1),(i,j+1),(i-1,j-1),(i-1,j+1),(i+1,j-1),(i+1,j+1)\}$$

Adjacency

4-adjacency: p,q are 4-adjacent if p is in the set $N_4(q)$

8-adjacency: p,q are 8-adjacent if p is in the set $N_8(q)$

Note that if p is in $N_{4/8}(q)$, then q must be also in $N_{4/8}(p)$

Fundamentals of Spatial filtering

The basics of Intensity Transformation and Spatial filtering

- Spatial domain processing

$$g(x,y) = T\big(f(x,y)\big)$$

  - g: processed image, f: input image
  - T: operator defined over a neighborhood of point (x,y)
    → Spatial filtering: neighborhood >= 1x1

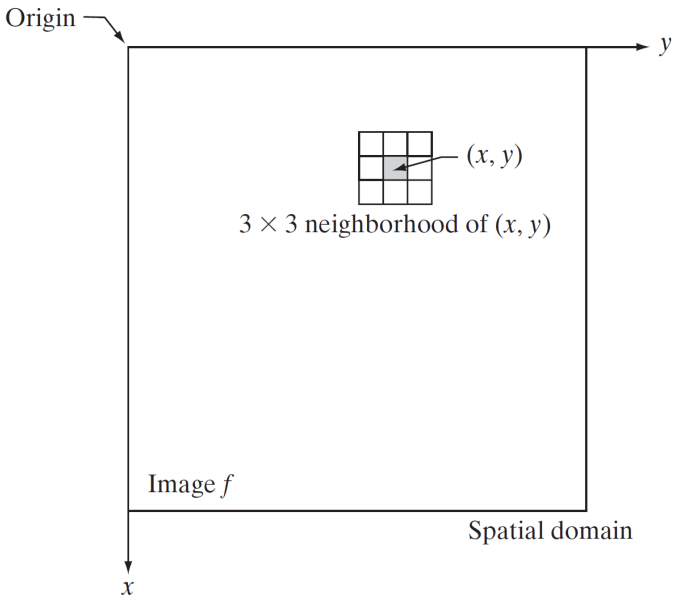


FIGURE 3.1 A 3×3 neighborhood about a point (x, y) in an image in the spatial domain. The neighborhood is moved from pixel to pixel in the image to generate an output image.

Fundamentals of Spatial filtering

Spatial filtering

- Principal tool to enhance image
- Name "filter" borrowed from frequency domain approach
    - Freq domain approach rejects/keeps only certain information
    - Smoothing (low-pass filtering), Edge enhancement (high-pass filtering)
- Filtering creates a new pixel with coordinates equal to the coordinates of the center of the neighborhood, and whose value is the result of the filtering operation
- Mechanics:
    - Predefined neighborhood: 4 neighbor, 8 neighbor, squared region
    - Operation: mathematical operation

Fundamentals of Spatial filtering

The mechanics of spatial filtering

- Linear spatial filter – linear operation
  - Consider a mask (filter) of odd size m × n (m = 2a + 1, n = 2b + 1 for a, b > 0 ∈ Z)

*Mask = spatial filter, kernel, window, template

  - *In general,* linear filtering of an image *f* of size *M X N* with a filter mask of size *m X n* is given by the expression

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

called convolution mask

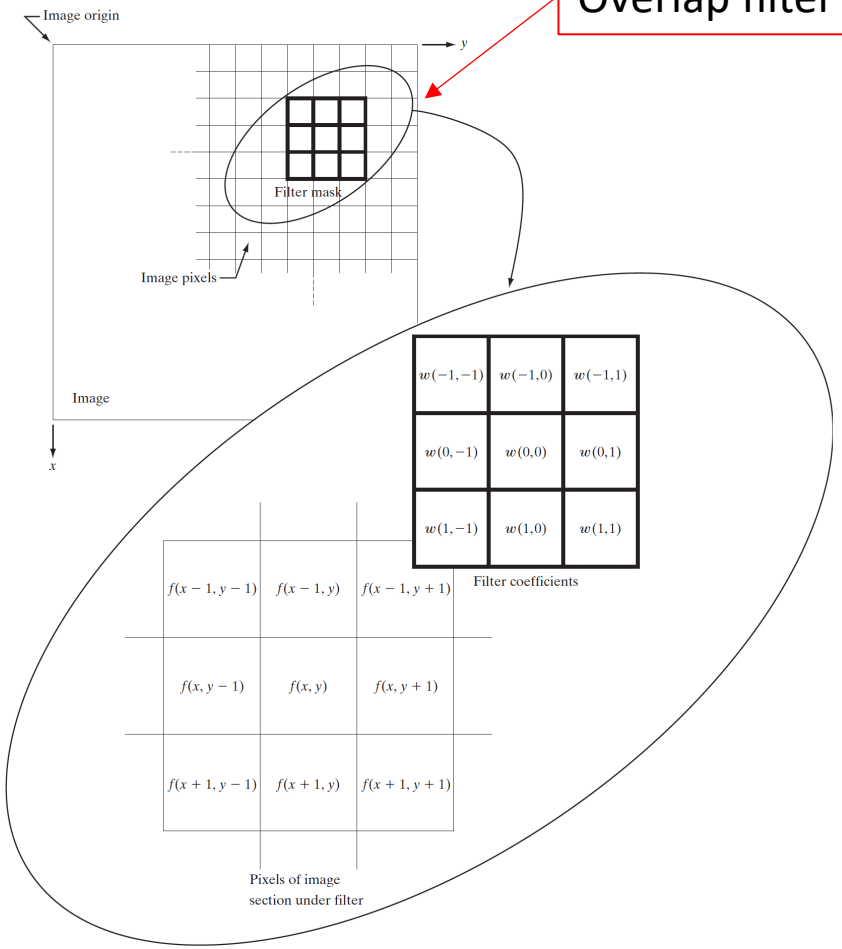
where $a = \frac{m-1}{2}$, $b = \frac{n-1}{2}$

⇒ linear spatial filtering = convolving a mask with an image

Fundamentals of Spatial filtering

The mechanics of spatial filtering

Overlap filter on the image



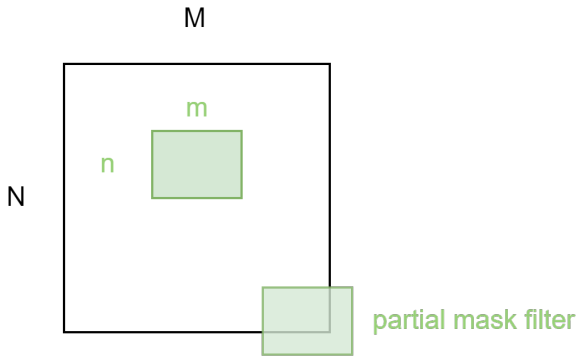
Filtered image g(x)

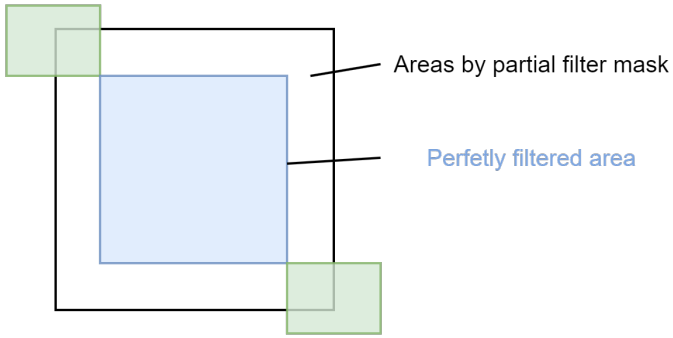
$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

$f(x_0,y_0) \rightarrow g(x_0,y_0)$

Fundamentals of Spatial filtering

The mechanics of spatial filtering





In general, images get small after filtering operation

Fundamentals of Spatial filtering

Spatial filtering with hard coding

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data

# Load image and crop
img = data.checkerboard()
H, W = img.shape[:]
img = img[W//3:2*W//3, :H//3]
print(img.shape)

# Filter define
filter_size = 5
filter_r = filter_size // 2
filter = np.ones((filter_size,filter_size)) / filter_size**2

# Filtering
filtered_img = np.zeros_like(img)
for i in range(filter_r, img.shape[0] - filter_r):
    for j in range(filter_r, img.shape[1] - filter_r):
        filtered_img[i, j] = np.sum(img[i - filter_r : i + filter_r + 1, j - filter_r : j + filter_r + 1] * filter)

# Visualization
plt.figure(figsize=(8,4))

plt.subplot(1,2,1)
plt.imshow(img, cmap='gray', interpolation=None)
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(filtered_img, cmap='gray', interpolation=None)
plt.axis('off')
```

Smoothing Spatial filtering

Smoothing Linear Filters

- Reduces sharp transitions but has the undesirable side effect that they blur edges
- Used for blurring and for noise reduction
  - prior to object extraction and bridging of small gaps in lines or curves

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

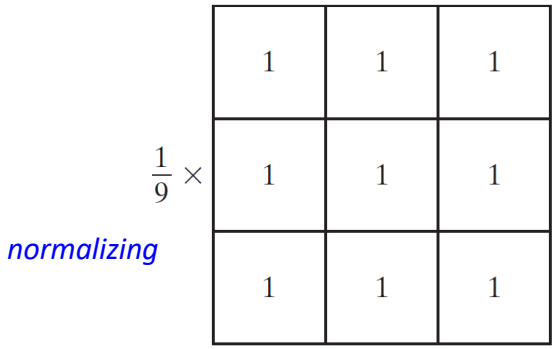
normalizing

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

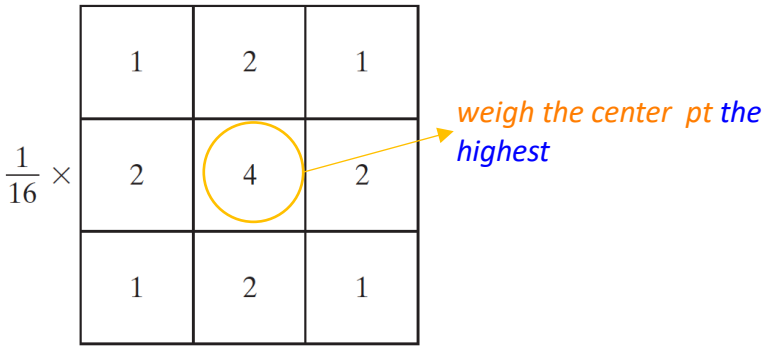
weigh the center pt the highest

Smoothing Spatial filtering

Smoothing Linear Filters



normalizing



box filter

whose all coeff are equal

weighted averaging filter

it reduces blurring in the smoothing process

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

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

the sum of the mask coeff

Smoothing Spatial filtering

Smoothing Linear Filters

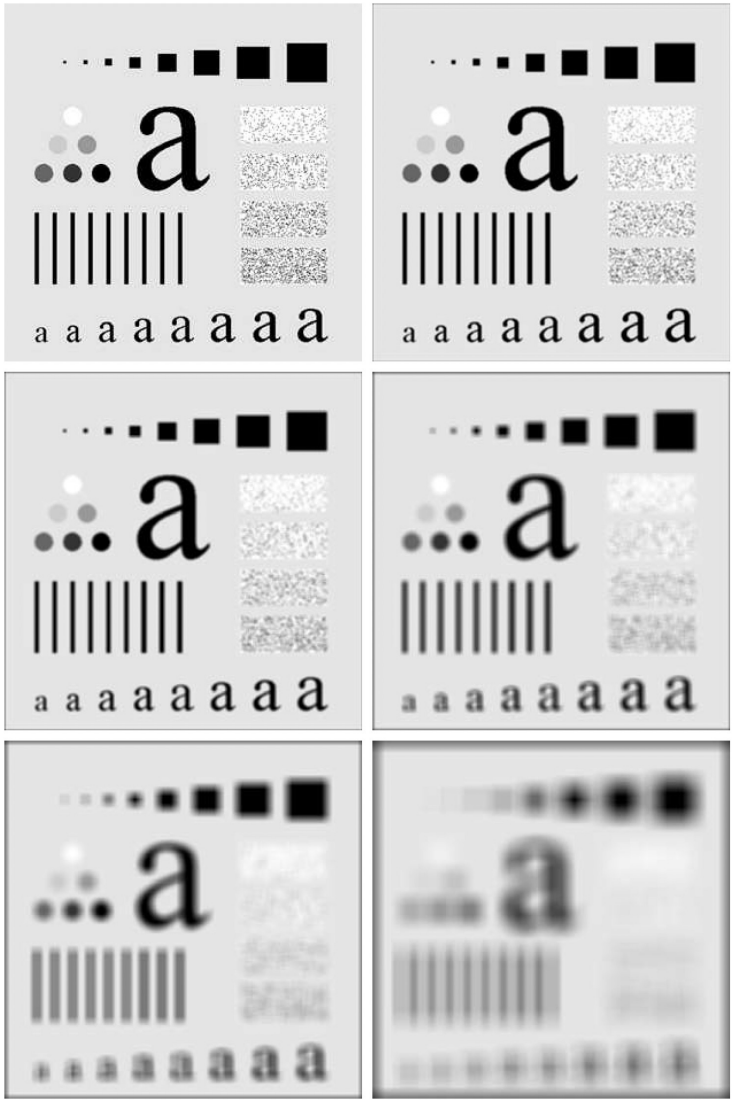


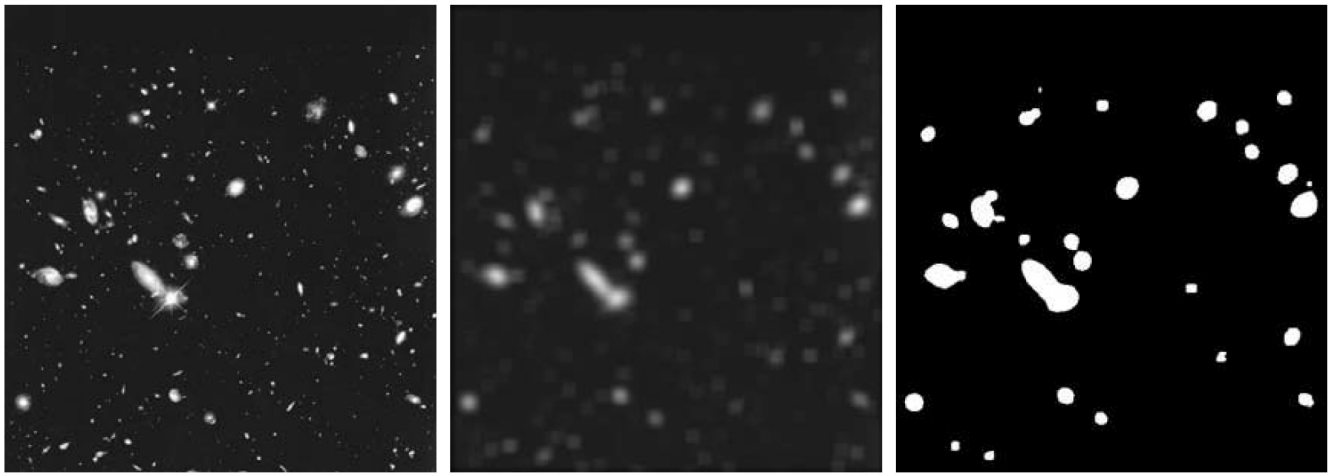
FIGURE 3.33 (a) Original image, of size 500 × 500 pixels. (b)–(f) Results of smoothing with square averaging filter masks of sizes $m = 3, 5, 9, 15$, and 35, respectively. The black squares at the top are of sizes 3, 5, 9, 15, 25, 35, 45, and 55 pixels, respectively; their borders are 25 pixels apart. 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.

a b
c d
e f

Smoothing Spatial filtering

Smoothing Linear Filters

- Ex) blurring coupled with thresholding
  - Blurring:
    The intensity of small objects blends with the background and larger objects become "bloblike". And thus easy to detect.
  - Choose:
    the size of mask ≈ size of the objects that will be blended with the background



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.)

Smoothing Spatial filtering

Order-statistic (Nonlinear) Filters

- Order-statistic filter are nonlinear spatial filter whose response is based on ordering (ranking)

- Ex) Median filter for smoothing impulse noise (Salt & pepper noise)
  - *popular because* they provide excellent noise reduction capabilities with less blurring than linear spatial filter

  - Suppose A = {a_1, a_2,…, a_K } are the pixel values in a neighborhood of a given pixel with $a_1 \le a_2 \le \ldots \le a_K$. Then

$$median(A) = \begin{cases} a_{K/2} & \text{for K even} \\ a_{(K+1)/2} & \text{for K odd} \end{cases}$$

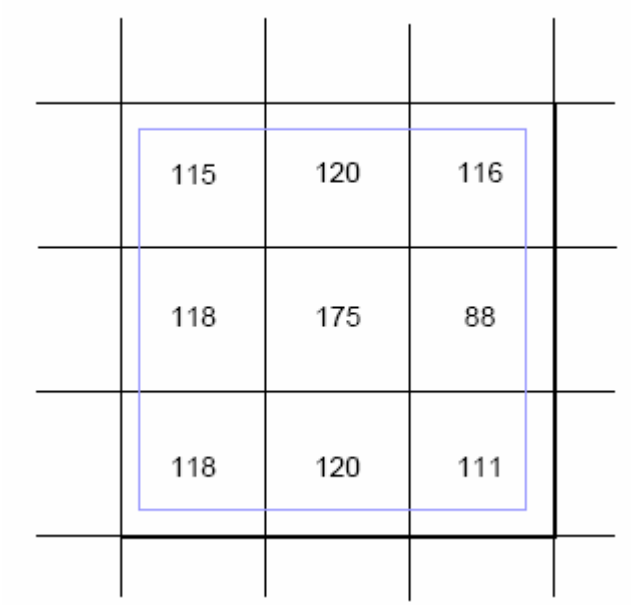
  - Note: Median of a set of values is the "center value," after sorting.

Smoothing Spatial filtering

Order-statistic (Nonlinear) Filters

- Median filtering example

$$\hat{f}(x,y) = \underset{(s,t) \in S_{xy}}{median}[g(s,t)]$$

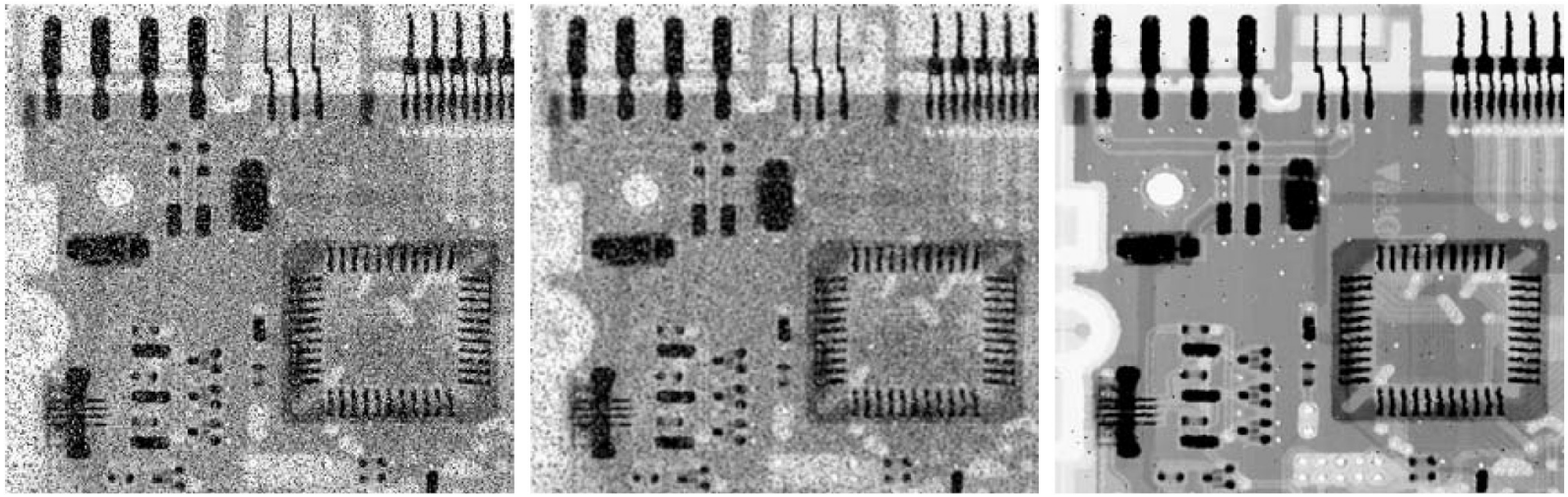


- Example:
  - Sort: 88, 111, 115, 116, 118, 118, 120, 120, 175.
  - Median = 118.

Smoothing Spatial filtering

Order-statistic (Nonlinear) Filters

- Median filtering example



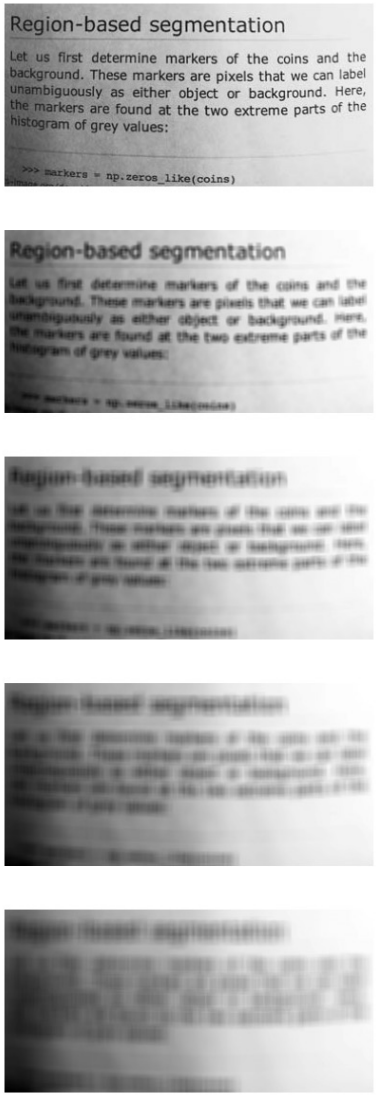
a b c

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.)

Smoothing Spatial filtering

Smoothing implementation (mean filter)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from scipy.ndimage import convolve

img = data.page()

filter_sizes = [5, 9, 13, 17]
filtered_imgs = []

for filter_size in filter_sizes:
    filter = np.ones((filter_size, filter_size))/ filter_size**2
    filtered_imgs.append(convolve(img, filter))

plt.figure(figsize=(5,15))
plt.subplot(5,1,1)
plt.axis('off')
plt.imshow(img, cmap='gray', interpolation=None)
for idx, filtered_img in enumerate(filtered_imgs):
    plt.subplot(5,1,idx + 2)
    plt.imshow(filtered_img, cmap='gray', interpolation=None)
    plt.axis('off')
```



Sharpening Spatial filtering

Foundation

Blurring ⟷ Sharpening

Spatial averaging (Integration)

Spatial differentiation (Derivatives)

- 1st order derivative of 1-d f(x):

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

- 2nd order derivative of 1-d f(x)

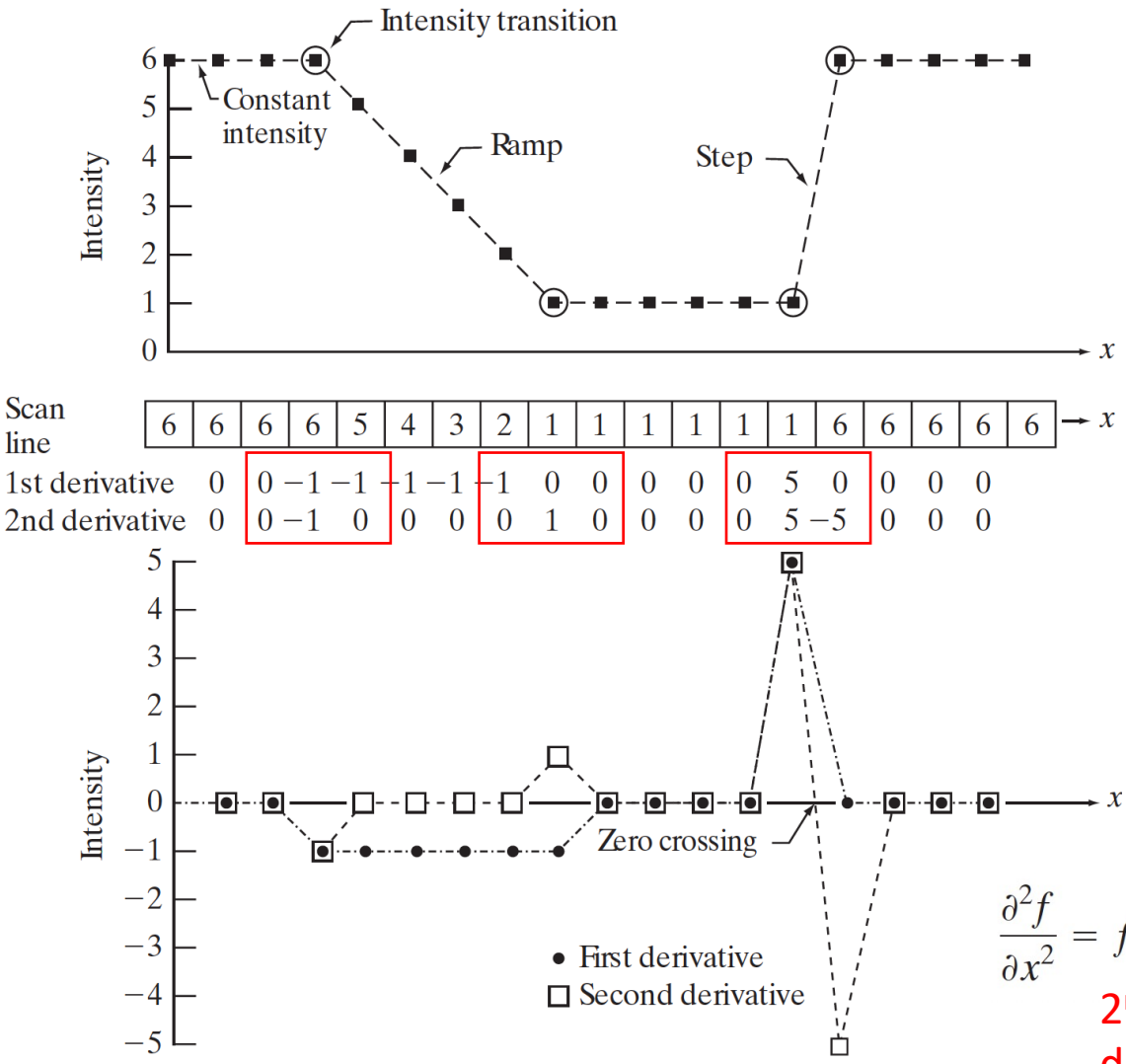
Can be used to extract edge component

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

Sharpening Spatial filtering

Foundation

Edge(fine detail) → Where the intensity profile changes rapidly



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.

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

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

2nd derivative enhance fine detail much better

Sharpening Spatial filtering

Using the 2nd derivative for image sharpening (The Laplacian)

$$\nabla^2 f(x, y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y) \quad (3.6\text{-}6)$$

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

0	−1	0
−1	4	−1
0	−1	0

−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 filtering

Using the 2nd derivative for image sharpening (The Laplacian)

gives an isotropic filter for increments of 90^0

gives an isotropic filter for increments of 45^0

Laplacian masks →

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−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 →

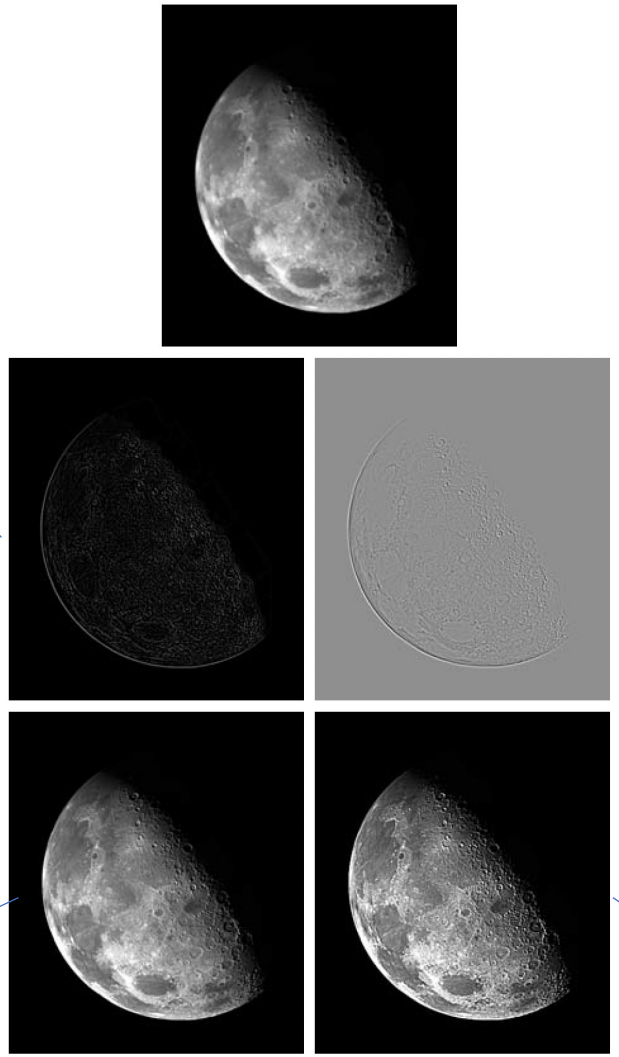
$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y) & \text{if the center coeff is negative} \\ f(x,y) + \nabla^2 f(x,y) & \text{if the center coeff is positive} \end{cases}$$

Sharpening Spatial filtering

Using the 2nd derivative for image sharpening (The Laplacian)

$\nabla^2 f(x,y)$

0	1	0
1	−4	1
0	1	0



$\nabla^2 f(x,y)$

1	1	1
1	−8	1
1	1	1

$f(x,y) - \nabla^2 f(x,y)$

$f(x,y) - \nabla^2 f(x,y)$

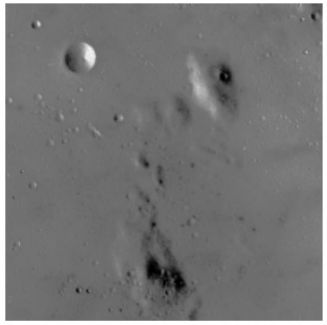
a
b c
d e

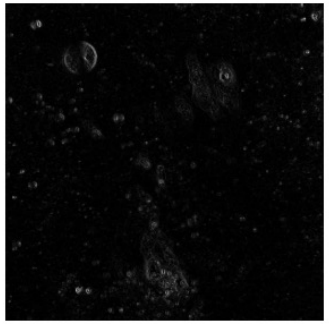
FIGURE 3.38
(a) Blurred image of the North Pole of the moon. (b) Laplacian without scaling. (c) Laplacian with scaling. (d) Image sharpened using the mask in Fig. 3.37(a). (e) Result of using the mask in Fig. 3.37(b). (Original image courtesy of NASA.)

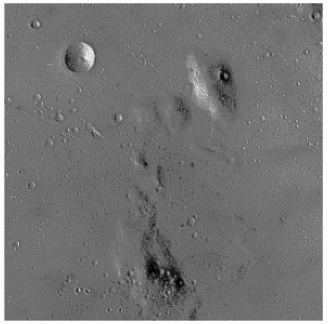
Sharpening Spatial filtering

Using the 2nd derivative for image sharpening (The Laplacian)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from scipy.ndimage import convolve

img = data.moon() / 255.0 # normalize for operation

filter = np.array([[1, 1, 1],
                   [1, -8, 1],
                   [1, 1, 1]], dtype=float)

filtered_img = convolve(img, filter)

edge_enhanced = img - filtered_img
edge_enhanced[edge_enhanced<0] = 0 # negative signal clipping
edge_enhanced[edge_enhanced>1] = 1 # over max signal clipping

# Visualization
plt.figure(figsize=(8,8))

plt.subplot(2,2,1)
plt.imshow(img, cmap='gray', interpolation=None)
plt.axis('off')

# for visualization, abs laplacian
plt.subplot(2,2,3)
plt.imshow(np.abs(filtered_img), cmap='gray', interpolation=None)
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(edge_enhanced, cmap='gray', interpolation=None)
plt.axis('off')
```







Sharpening Spatial filtering

Using the 2nd derivative for image sharpening (The Laplacian)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2gray
from scipy.ndimage import convolve

img = data.cat() / 255.0 # normalize for operation
img = rgb2gray(img)

filter = np.array([[1, 1, 1],
                   [1, -8, 1],
                   [1, 1, 1]], dtype=float)

filtered_img = convolve(img, filter)

edge_enhanced = img - filtered_img
edge_enhanced[edge_enhanced<0] = 0 # negative signal clipping
edge_enhanced[edge_enhanced>1] = 1 # over max signal clipping

# Visualization
plt.figure(figsize=(12,8))

plt.subplot(2,2,1)
plt.imshow(img, cmap='gray', interpolation=None)
plt.axis('off')

# for visualization, abs laplacian
plt.subplot(2,2,3)
plt.imshow(np.abs(filtered_img), cmap='gray', interpolation=None)
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(edge_enhanced, cmap='gray', interpolation=None)
plt.axis('off')
```

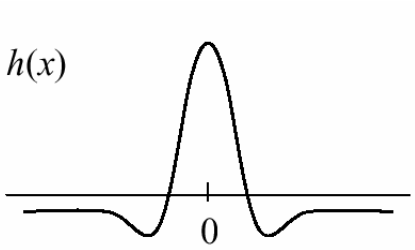
Sharpening Spatial filtering

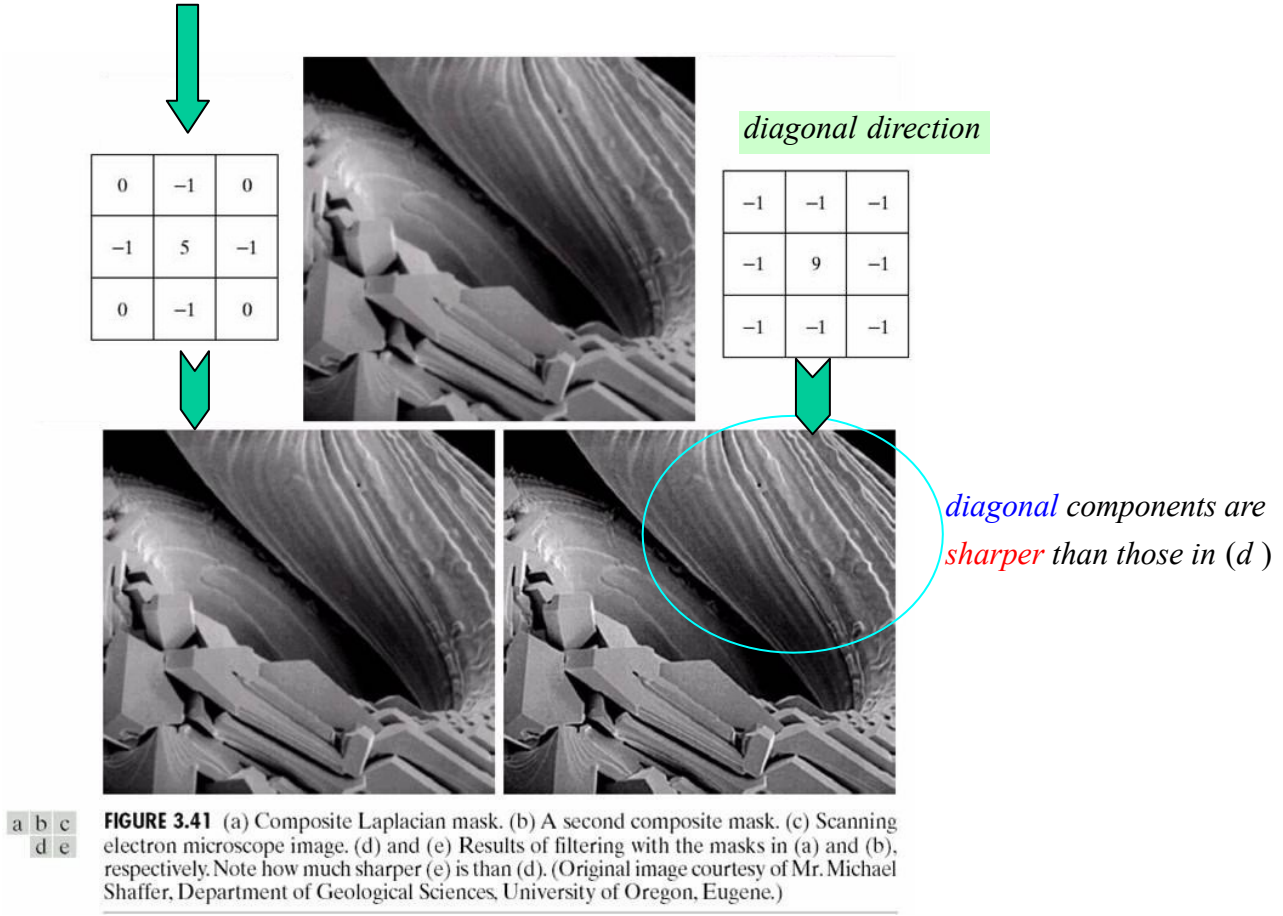
Using the 2nd derivative for image sharpening (The Laplacian)

- *Ex2)*

$$g(x,y) = f(x,y) - \nabla^2 f$$

$$\therefore \quad = 5f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)]$$





a b c d e **FIGURE 3.41** (a) Composite Laplacian mask. (b) A second composite mask. (c) Scanning electron microscope image. (d) and (e) Results of filtering with the masks in (a) and (b), respectively. Note how much sharper (e) is than (d). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

Other edge filters

Using 1st order derivative for image sharpening (The Gradient)

- Gradient

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

(Non-isotropic linear operator)

- Magnitude

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

simple to implement, *isotropic for multiples of* 90^0

(Isotropic, not linear operator)

Other edge filters

Using 1st order derivative for image sharpening (The Gradient)

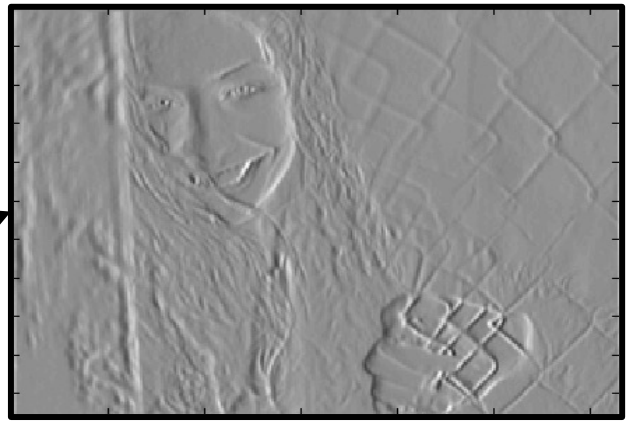
- Ex)

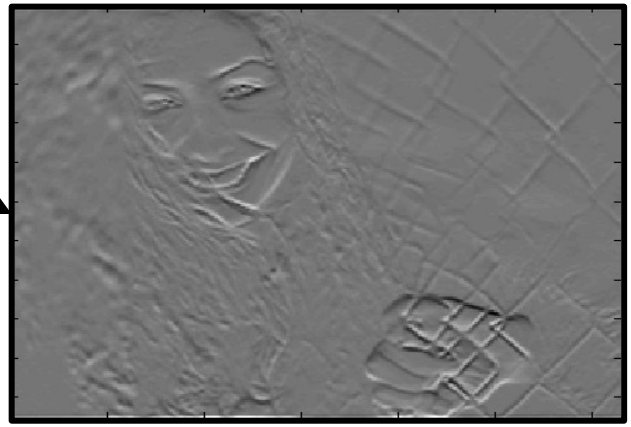
$\frac{\partial f}{\partial x}$

$\frac{\partial f}{\partial y}$

Vertical edge enhanced

Vertical edge



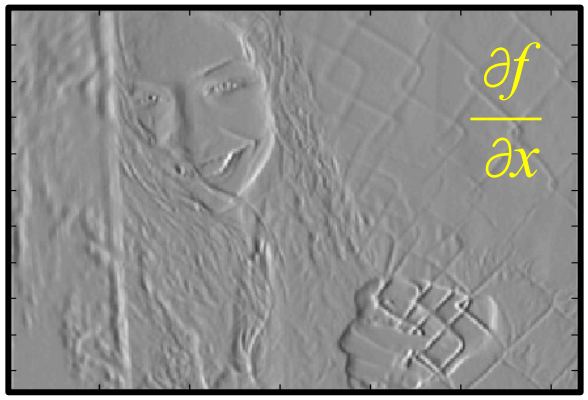


Horizontal edge

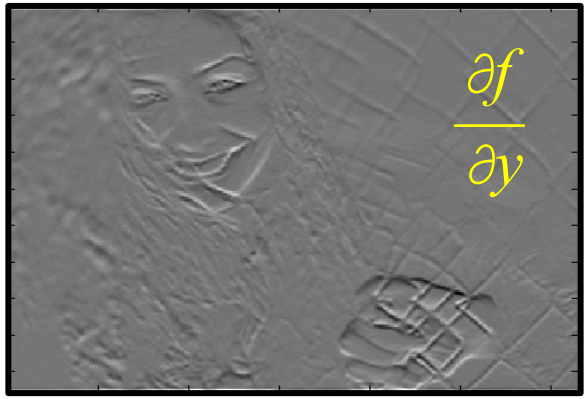
Other edge filters

Using 1st order derivative for image sharpening (The Gradient)

- Ex)



$$: \sqrt{\frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2}$$

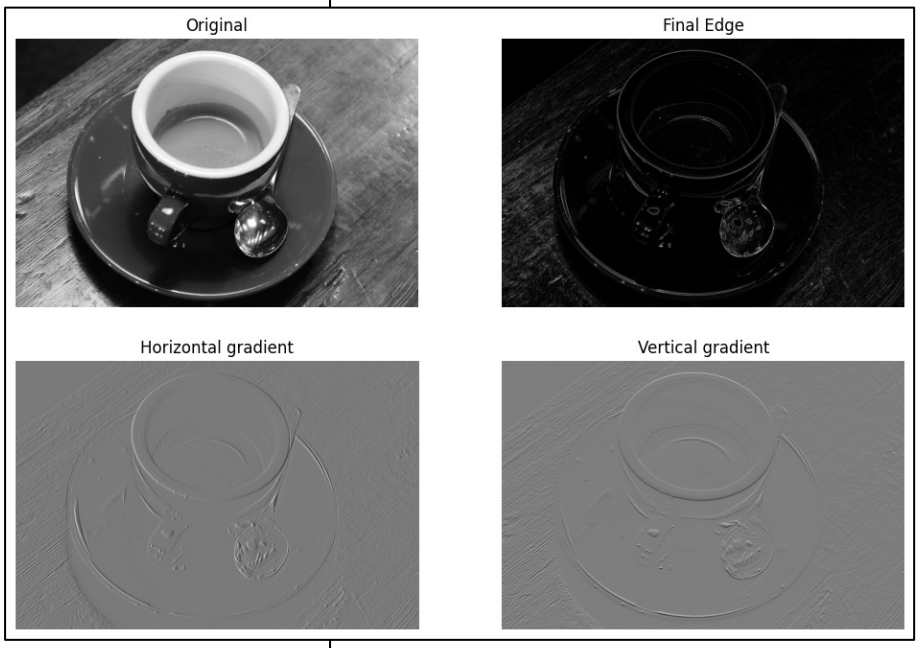


final edge

Other edge filters

Using 1st order derivative for image sharpening (The Gradient)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2gray
from scipy.ndimage import convolve

img = data.coffee() / 255.0 # normalize for operation
img = rgb2gray(img)

filter_grad_h = np.array([[1,- 1]], dtype=float)
filter_grad_v = np.array([[1],
                          [- 1]], dtype=float)

filtered_img_h = convolve(img, filter_grad_h)
filtered_img_v = convolve(img, filter_grad_v)

# Visualization
plt.figure(figsize=(12,8))

plt.subplot(2,2,1)
plt.imshow(img, cmap='gray', interpolation=None)
plt.title('Original')
plt.axis('off')

plt.subplot(2,2,2)
plt.imshow(np.sqrt(filtered_img_h**2 + filtered_img_v**2), cmap='gray', interpolation=None)
plt.title('Final Edge')
plt.axis('off')

plt.subplot(2,2,3)
plt.imshow(filtered_img_h, cmap='gray', interpolation=None)
plt.title('Horizontal gradient')
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(filtered_img_v, cmap='gray', interpolation=None)
plt.title('Vertical gradient')
plt.axis('off')
```



Other edge filters

Sobel filter (1st derivative)

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

$$\nabla f(x,y) = |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)| + |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)|$$

$$h_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad h_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Vertical edge Horizontal edge

the coeff' s sum is to zero

⇒ *gives a response of* 0 *in an area of const gray level*

(as expected of *a derivative operator*)

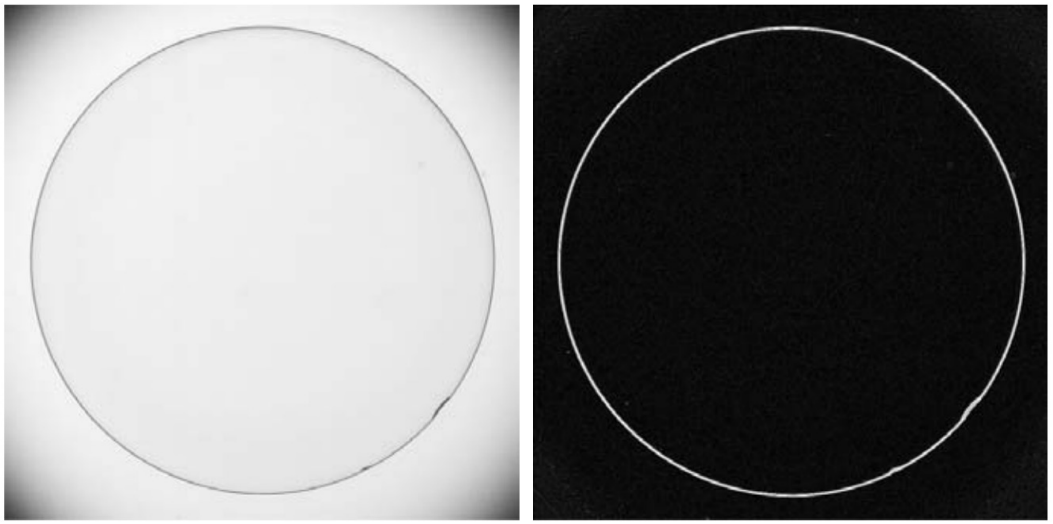
Other edge filters

Sobel filter (1st derivative)

- Ex) use of gradient for edge enhancement

a b

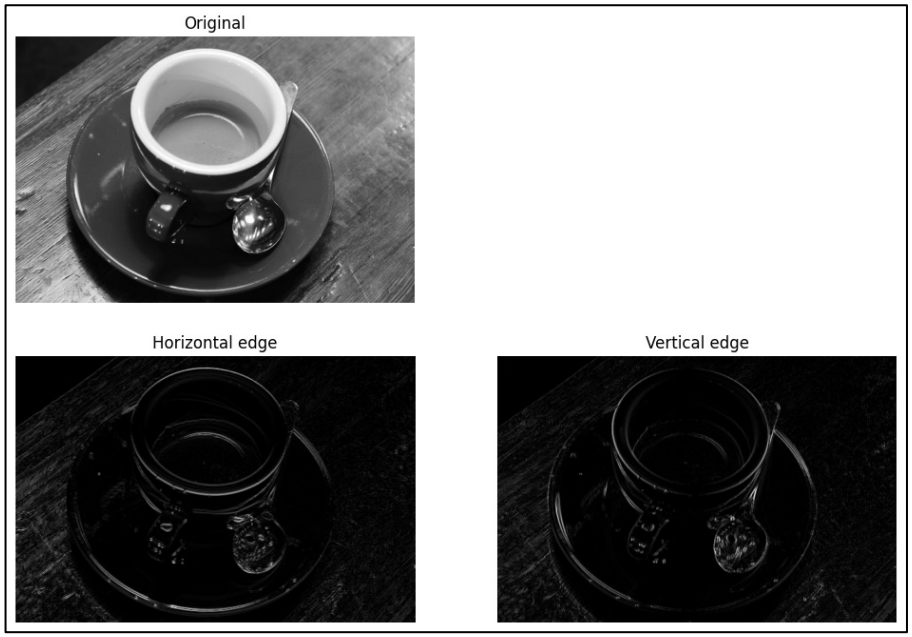
FIGURE 3.42
(a) Optical image of contact lens (note defects on the boundary at 4 and 5 o'clock). (b) Sobel gradient. (Original image courtesy of Pete Sites, Perceptics Corporation.)



Other edge filters

Sobel filter (1st derivative)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2gray
from scipy.ndimage import convolve

img = data.coffee() / 255.0 # normalize for operation
img = rgb2gray(img)

sobel_h = np.array([[-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]], dtype=float)
sobel_v = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]], dtype=float)

edge_h = convolve(img, sobel_h)
edge_v = convolve(img, sobel_v)

# Visualization
plt.figure(figsize=(12,8))

plt.subplot(2,2,1)
plt.imshow(img, cmap='gray', interpolation=None)
plt.title('Original')
plt.axis('off')

plt.subplot(2,2,3)
plt.imshow(np.abs(edge_h), cmap='gray', interpolation=None)
plt.title('Horizontal edge')
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(np.abs(edge_v), cmap='gray', interpolation=None)
plt.title('Vertical edge')
plt.axis('off')
```



Spatial transformation

Spatial transformation

Homogeneous coordinates

- A mathematical tool for expressing linear and affine transformations using matrix multiplication.
  - 2D transformation → 3 x 3 matrix, 3D transformation → 4 x 4 matrix
- Ex) $(x, y) \rightarrow (x + 1, 2y + 3)$

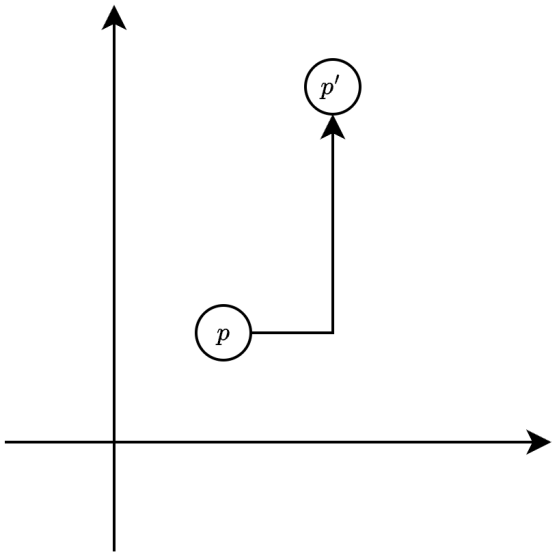
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

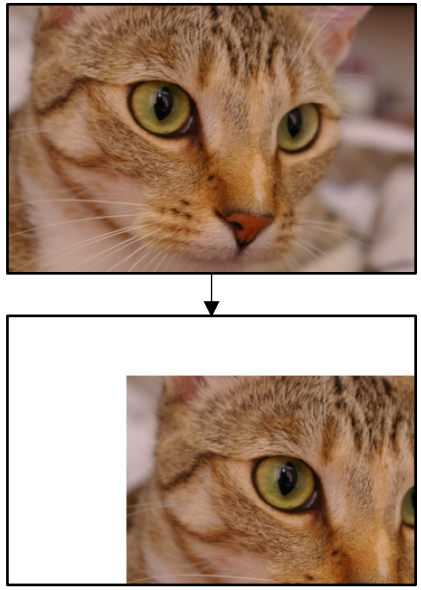
Spatial transformation

Linear transformation

- Translation
- Ex) $(x, y) \rightarrow (x + a, y + b)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$





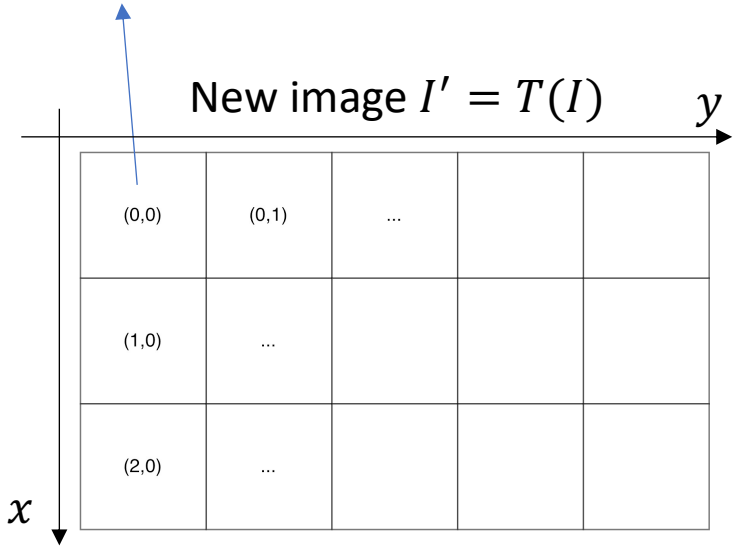
Spatial transformation

Linear transformation

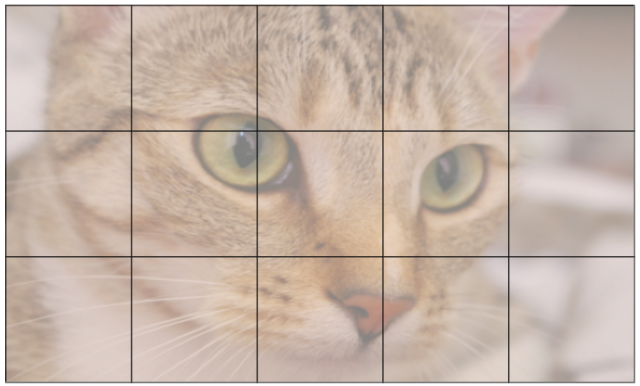
- Image spatial transformation mechanism
- Ex) $(x, y) \rightarrow (x - 1, y - 2)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Original location (x, y)?



Original image I



Spatial transformation

Linear transformation

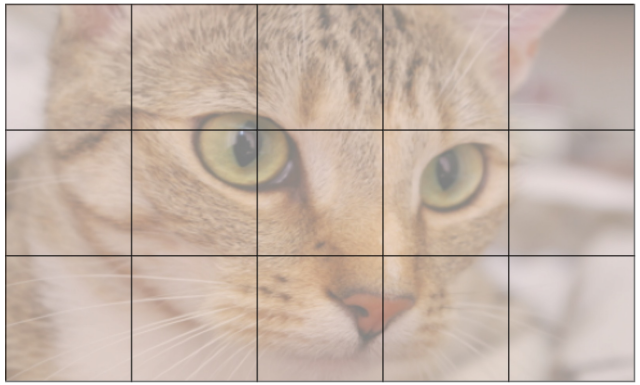
- Image spatial transformation mechanism
- Ex) $(x, y) \rightarrow (x-1, y-2)$

$$\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}1 & 0 & -1\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & -1\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}^{-1}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & 1\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}1\\2\\1\end{bmatrix} = \begin{bmatrix}x\\y\\1\end{bmatrix}$$

New image $I' = T(I)$

(0,0)	(0,1)	...		
(1,0)	...			
(2,0)	...			

Original image I



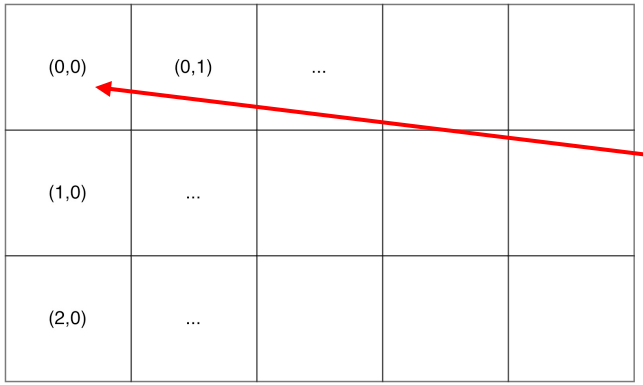
Spatial transformation

Linear transformation

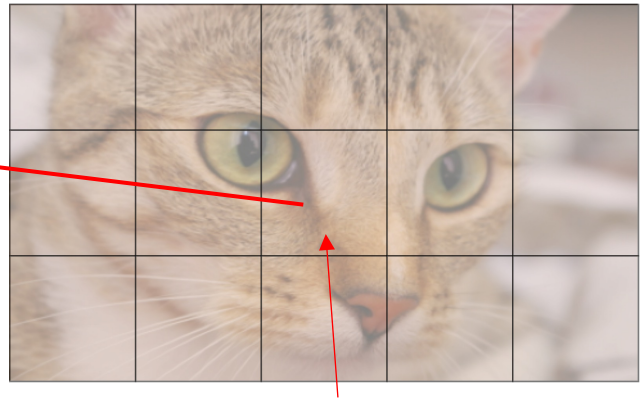
- Image spatial transformation mechanism
- Ex) $(x, y) \rightarrow (x-1, y-2)$

$$\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}1 & 0 & -1\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & -1\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}^{-1}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & 1\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}1\\2\\1\end{bmatrix} = \begin{bmatrix}x\\y\\1\end{bmatrix}$$

New image $I' = T(I)$



Original image I



Original location

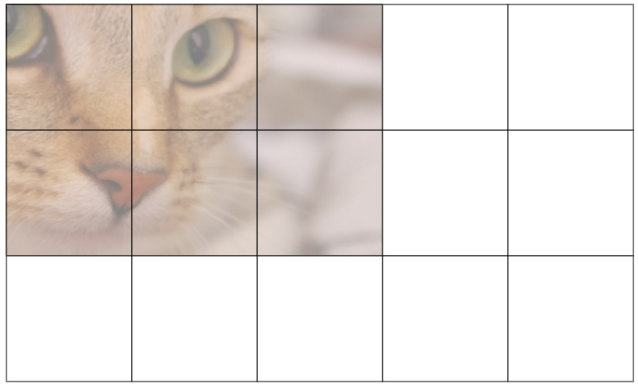
Spatial transformation

Linear transformation

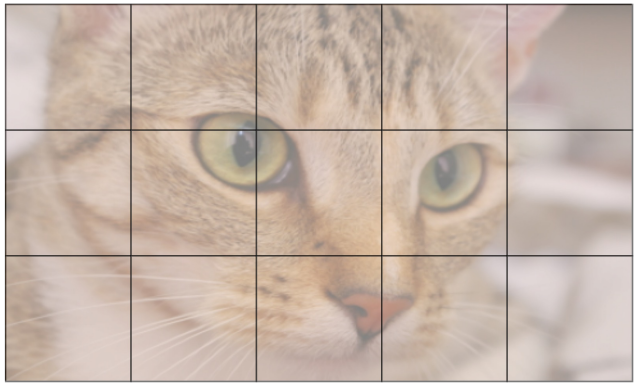
- Image spatial transformation mechanism
- Ex) $(x, y) \rightarrow (x-1, y-2)$

$$\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}1 & 0 & -1\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & -1\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}^{-1}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & 1\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}1\\2\\1\end{bmatrix} = \begin{bmatrix}x\\y\\1\end{bmatrix}$$

New image $I' = T(I)$



Original image I

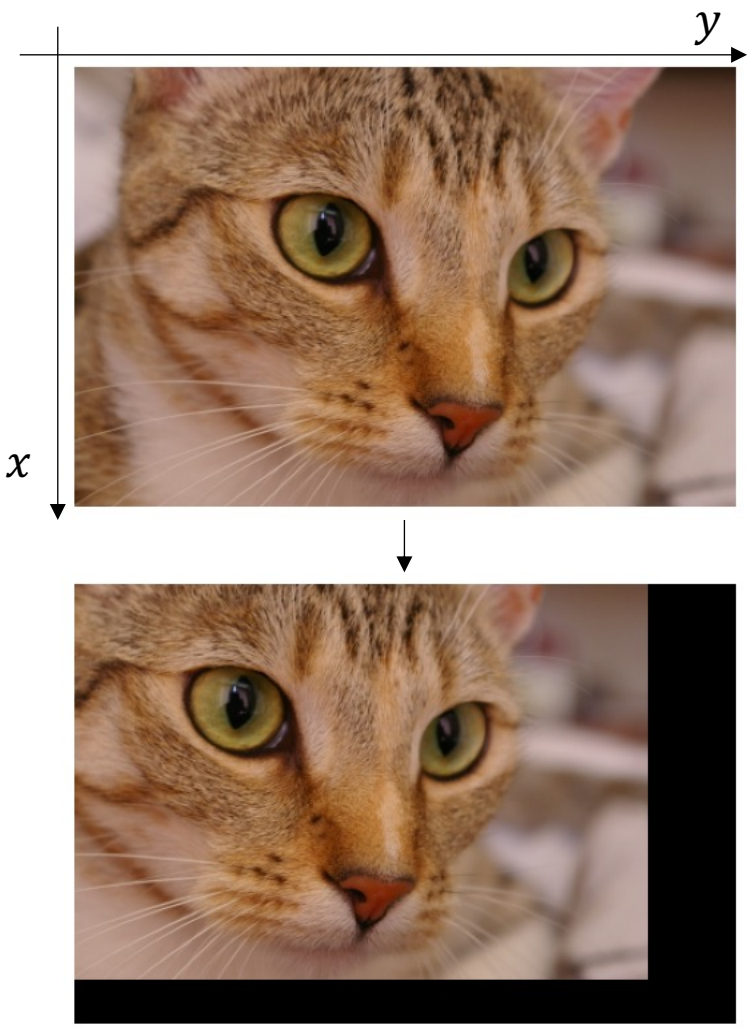


Spatial transformation

Linear transformation

- Image translation (hard coding)

```
import numpy as np
from skimage import data
from skimage.color import rgb2gray
import matplotlib.pyplot as plt

img = data.cat()
new_img = np.zeros_like(img)

tx, ty = -30, -60
T = np.array([[1, 0, tx],
              [0, 1, ty],
              [0, 0, 1]], dtype=float)
T_inv = np.linalg.inv(T)

H, W, _ = img.shape
for i in range(H):
    for j in range(W):
        hcoord = np.array([i, j, 1.0])
        i_org_f, j_org_f, _ = T_inv @ hcoord

        i_org = int(round(i_org_f))
        j_org = int(round(j_org_f))

        if 0 <= i_org < H and 0 <= j_org < W:
            new_img[i, j, :] = img[i_org,
j_org, :]

plt.imshow(new_img, cmap="gray")
plt.axis("off")
plt.show()
```

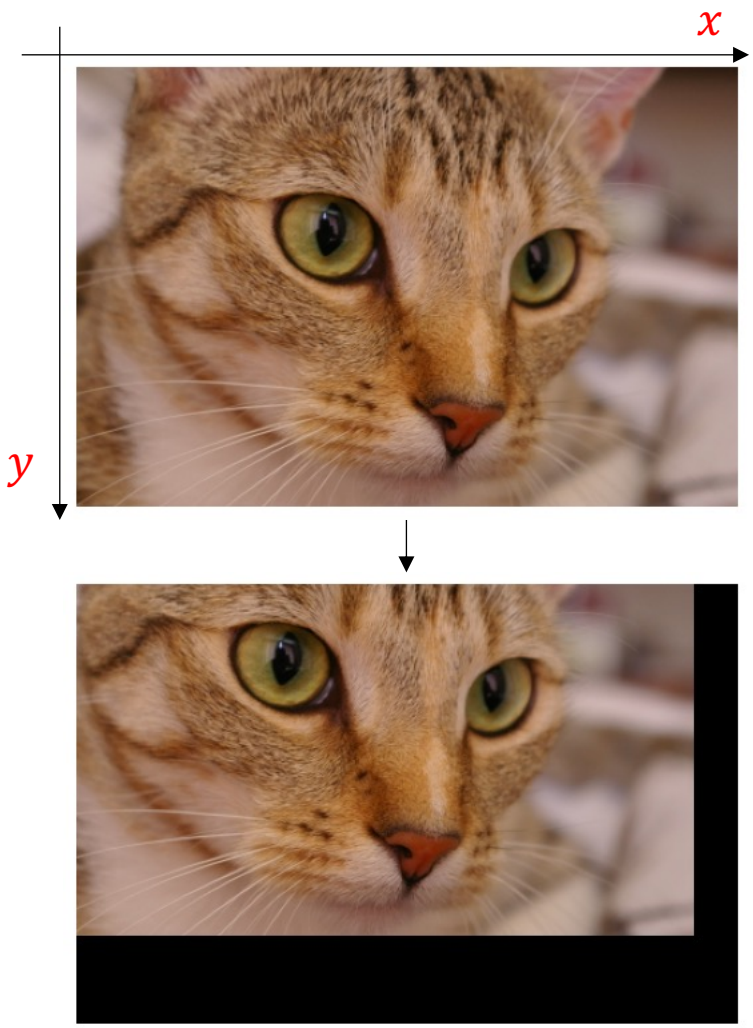


Spatial transformation

Linear transformation

- Image translation (using skimage)

```
import numpy as np
from skimage import data
from skimage.transform import warp, AffineTransform
import matplotlib.pyplot as plt

image = data.cat()

tx, ty = -30, -60

T = np.array([
    [1, 0, tx],
    [0, 1, ty],
    [0, 0, 1]])

tform = AffineTransform(matrix=T)
# inverse mapping
warped = warp(image, tform.inverse)

plt.imshow(warped)
plt.axis("off")
plt.show()
```



주의) library 별로 image의 x, y축 방향, 순서, 원점의 위치 등 설정이 다르므로 사용 전 반드시 샘플로 테스트 후 확인 필요

→ Skimage: x 방향이 오른쪽, y방향이 아래쪽, 원점 왼-위

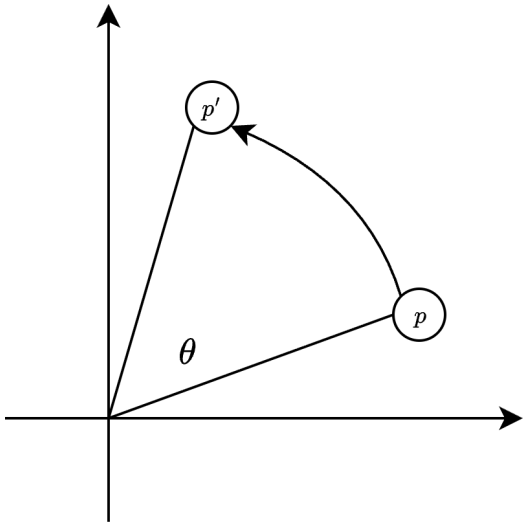
→ Opencv, scipy 다 세팅이 다름

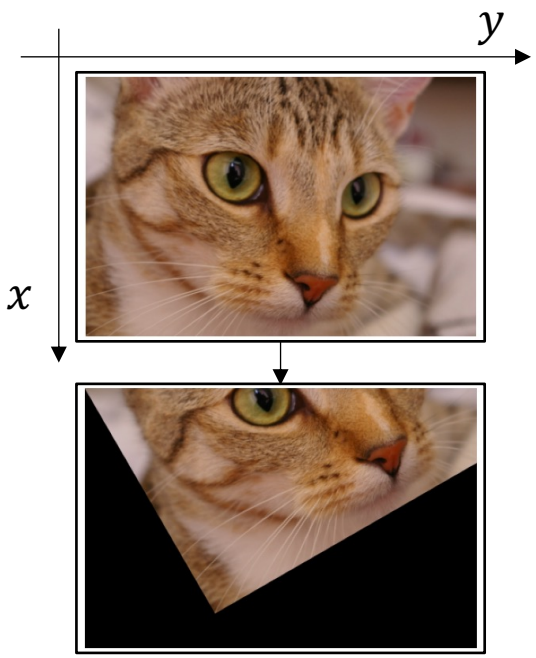
Spatial transformation

Linear transformation

- Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 1 \\ \sin\theta & \cos\theta & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



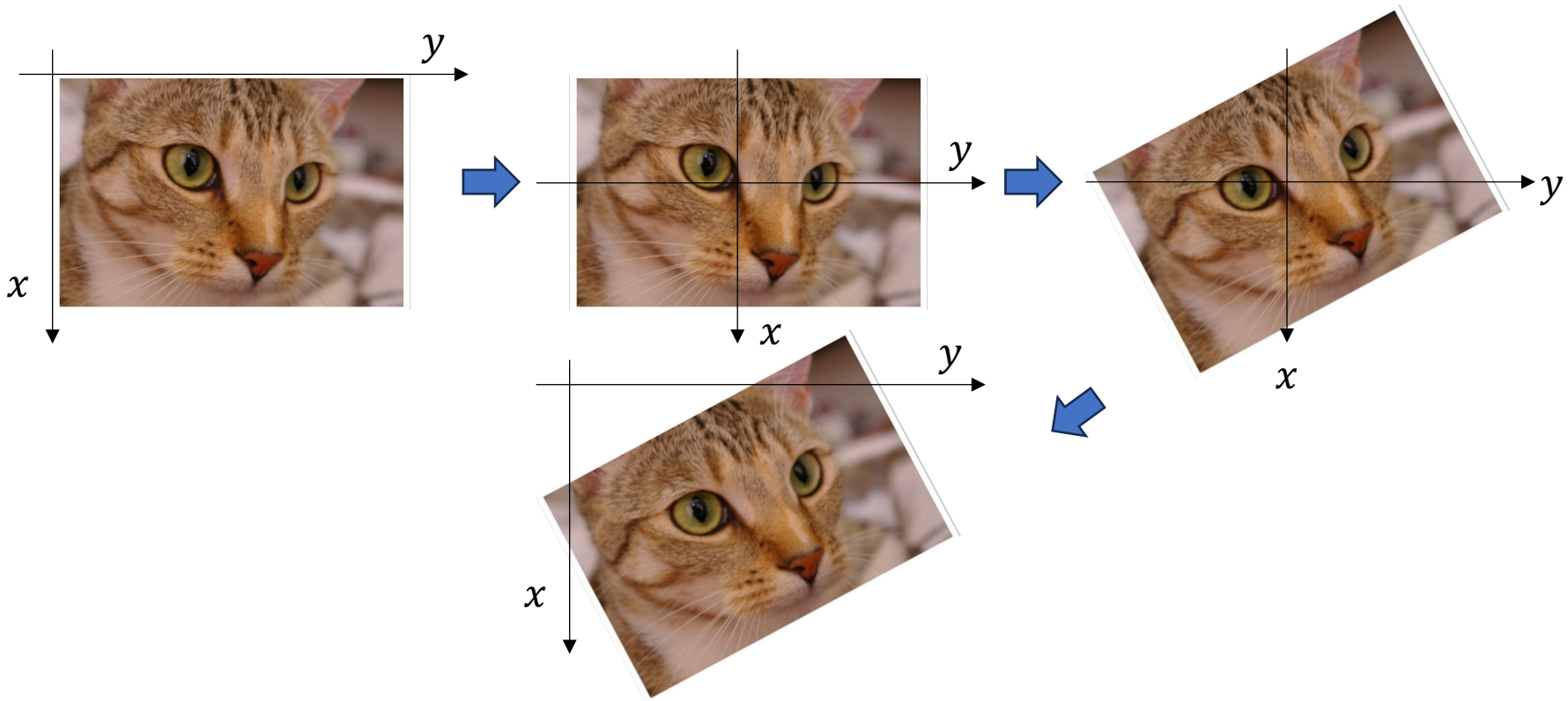


Spatial transformation

Linear transformation

- Rotation about center
  - $(x, y) \rightarrow (x - c_x, y - c_y) \rightarrow Rotation \rightarrow (x + c_x, y + c_y)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 1 \\ \sin\theta & \cos\theta & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

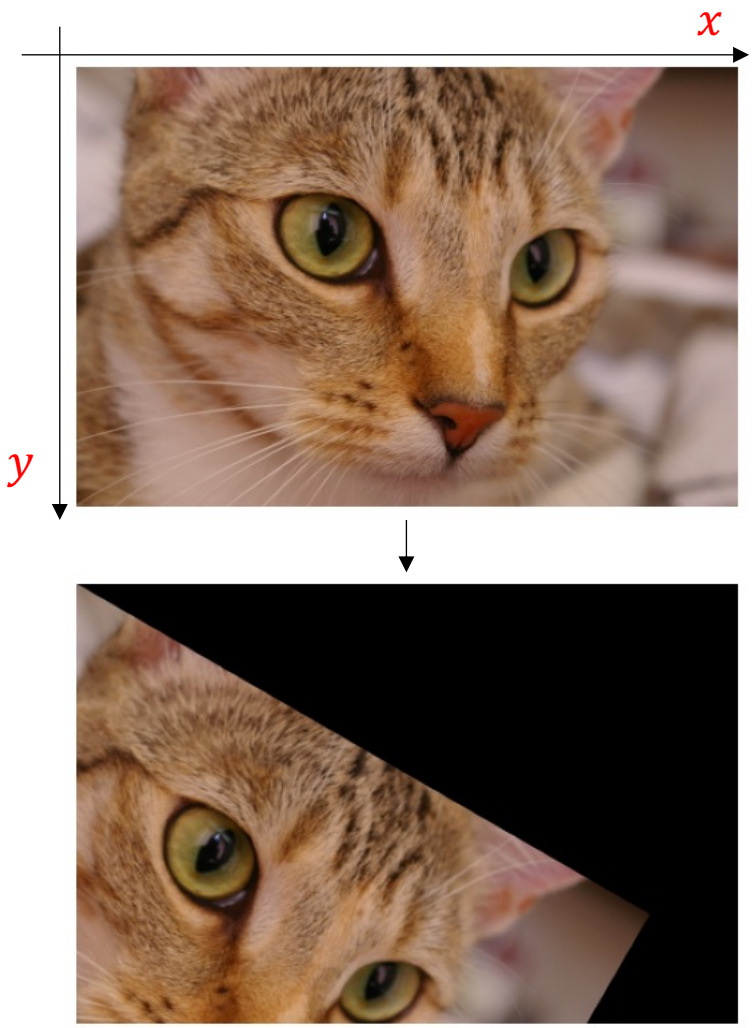


Spatial transformation

Linear transformation

- Image rotation (using skimage)

```
import numpy as np
from skimage import data
from skimage.transform import warp, AffineTransform
import matplotlib.pyplot as plt

image = data.cat()

theta = np.deg2rad(30)
T = np.array([
    [np.cos(theta), -np.sin(theta), 0],
    [np.sin(theta),  np.cos(theta), 0],
    [0,              0,             1]
])

tform = AffineTransform(matrix=T)
warped = warp(image, tform.inverse)

plt.imshow(warped)
plt.axis("off")
plt.show()
```

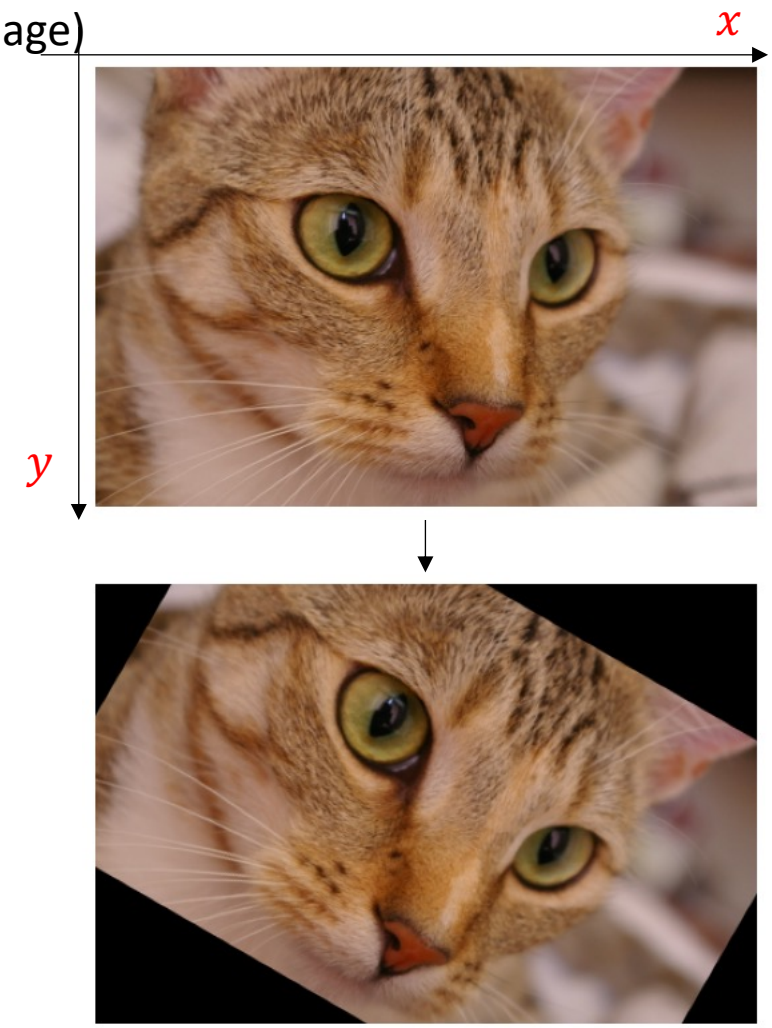


Spatial transformation

Linear transformation

- Image rotation about center (using skimage)

```
import numpy as np
from skimage import data
from skimage.transform import warp, AffineTransform
import matplotlib.pyplot as plt

image = data.cat()

cx = image.shape[1] // 2
cy = image.shape[0] // 2
T1 = np.array([
    [1, 0, -cx],
    [0, 1, -cy],
    [0, 0, 1]
])
T1_inv = np.linalg.inv(T1)

theta = np.deg2rad(30)
T2 = np.array([
    [np.cos(theta), -np.sin(theta), 0],
    [np.sin(theta),  np.cos(theta), 0],
    [0,              0,             1]
])

T = T1_inv @ T2 @ T1
tform = AffineTransform(matrix=T)
warped = warp(image, tform.inverse)

plt.imshow(warped)
plt.axis("off")
plt.show()
```



Spatial transformation

Linear transformation

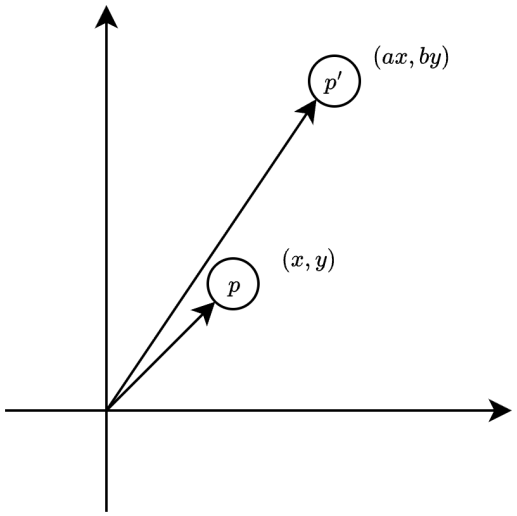
- Translation + Rotation → Rigid body transform
    - It does not change shape or scale

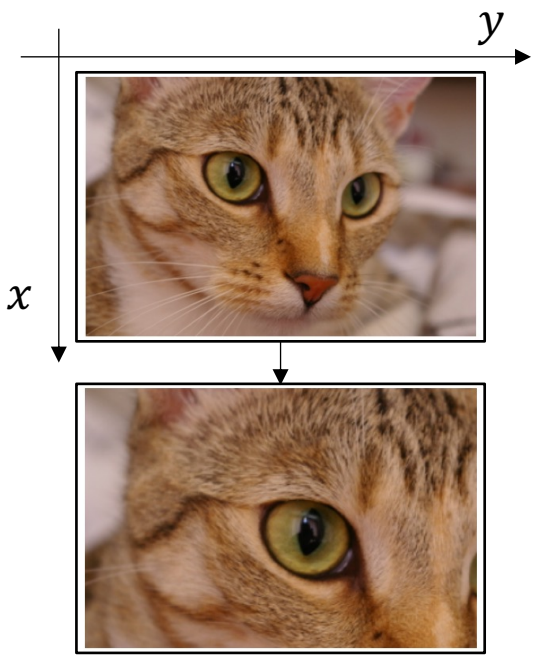
Spatial transformation

Linear transformation

- Scaling
- $(x, y) \rightarrow (ax, by)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



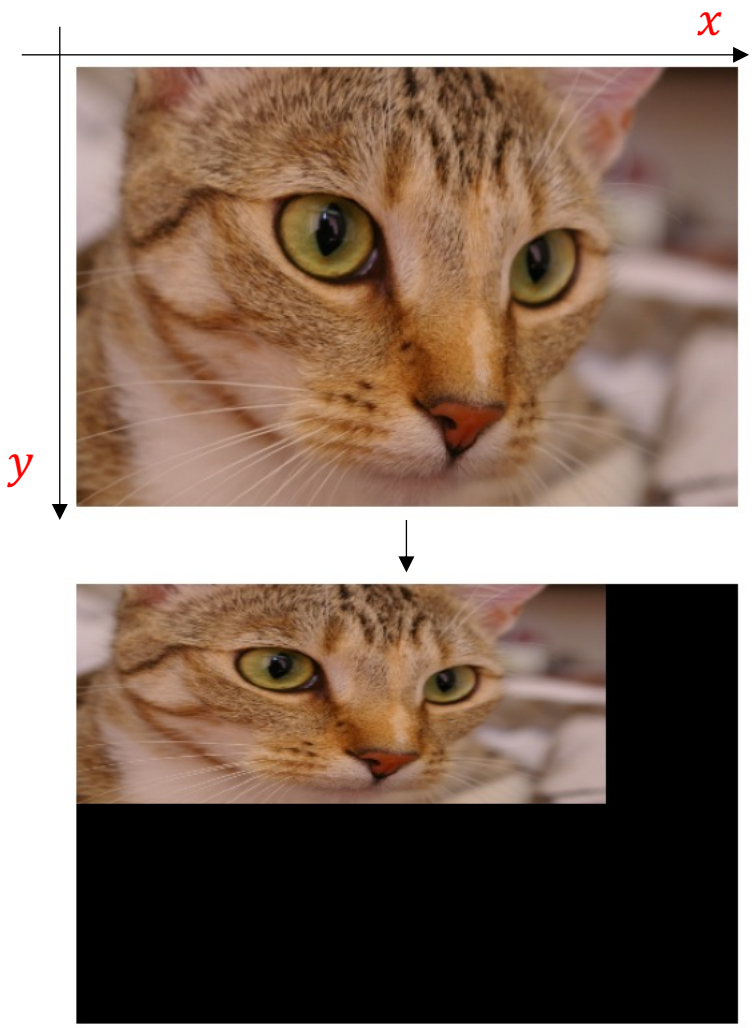


Spatial transformation

Linear transformation

- Image scaling (using skimage)

```
import numpy as np
from skimage import data
from skimage.transform import warp, AffineTransform
import matplotlib.pyplot as plt

image = data.cat()

x_scale = 0.8
y_scale = 0.5
T = np.array([
    [x_scale, 0, 0],
    [0, y_scale, 0],
    [0, 0, 1]
])

tform = AffineTransform(matrix=T)
warped = warp(image, tform.inverse)

plt.imshow(warped)
plt.axis("off")
plt.show()
```



예제) 중심 기준으로 scaling은 어떻게?

Spatial transformation

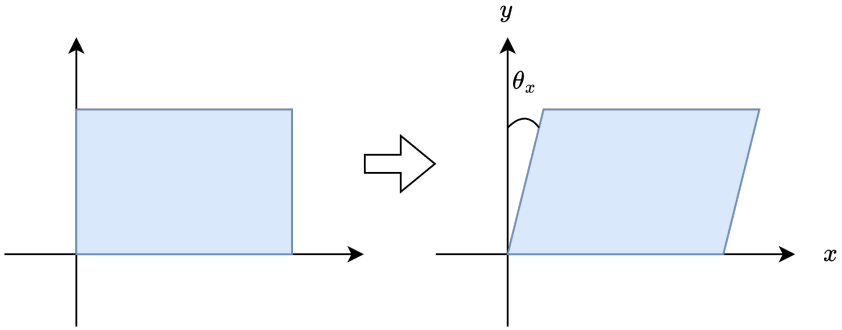
Linear transformation

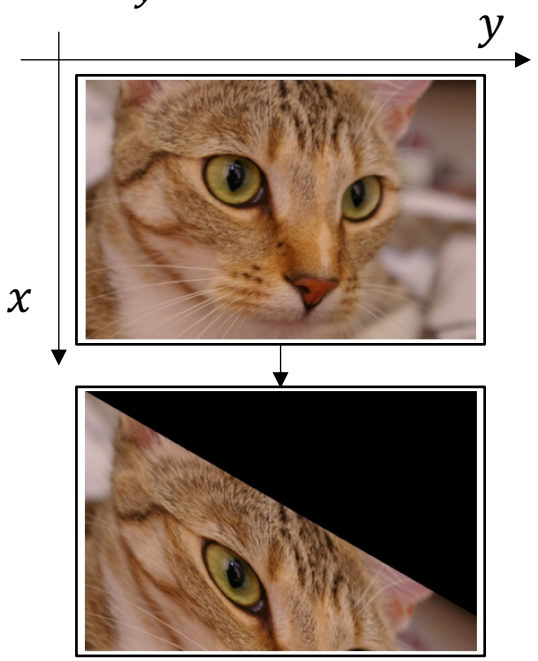
Note. Shearing 의 DOF는 사실상 1
→ s_x, s_y를 모두 사용할 때에는 shearing과 rotation이 섞인 것으로 봄

- Shearing

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_x & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$s_x = \tan\theta_x\,, \qquad s_y = \tan\theta_y$$



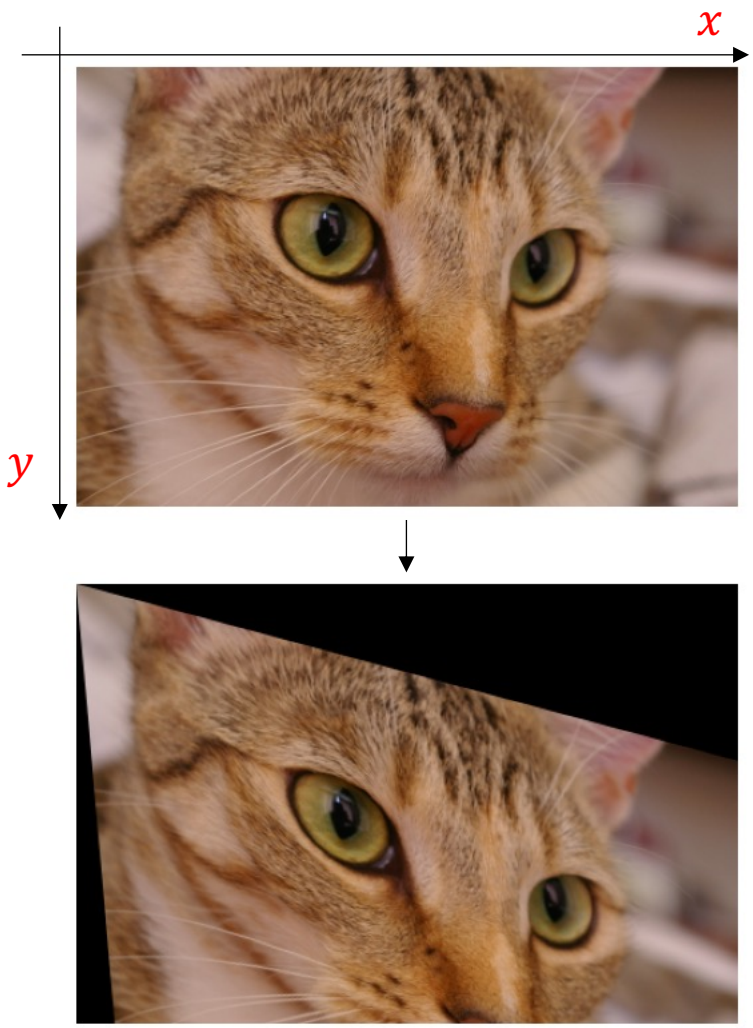


Spatial transformation

Linear transformation

- Image shearing (using skimage)

```
import numpy as np
from skimage import data
from skimage.transform import warp, AffineTransform
import matplotlib.pyplot as plt

image = data.cat()

theta_x = np.deg2rad(5)
theta_y = np.deg2rad(15)
T = np.array([
    [1,               np.tan(theta_x), 0],
    [np.tan(theta_y), 1,               0],
    [0,               0,               1]
])

tform = AffineTransform(matrix=T)
warped = warp(image, tform.inverse)

plt.imshow(warped)
plt.axis("off")
plt.show()
```



Spatial transformation

Linear transformation

- General Affine (6-parameters in 2D space)
  - Translation + Rotation + Scaling + Shearing

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  - 6 independent parameters
  - DOF: 2 + 1 + 2 + 1

Other transformations

Various image augmentation