Preview

■ 영상 처리

- 특정 목적을 달성하기 위해 원래 영상을 개선된 새로운 영상으로 변환하는 작업

(a) 안개 낀 도로 영상　　(b) 히스토그램 평활화로 개선한 영상

그림 3-1 영상 처리로 화질 개선

■ 화질 개선 자체가 목적인 경우

- 예) 도주 차량의 번호판 식별. 병변 위치 찾기 등

■ 컴퓨터 비전은 전처리로 활용하여 인식 성능을 향상

각 절에서 다루는 내용

- 모폴로지
  - 이진 모폴로지, 명암 모폴로지
- 영상 처리의 세 가지 기본 연산

  →점연산, 영역연산, 기하연산
- OpenCV의 시간 효율

모폴로지

- 모폴로지
  - 원래 생물학에서 생물의 모양 변화를 표현하는 기법
  - 영상을 표현하는데 유용한 영상 구성 성분을 추출하기 위하여 사용하는 영상처리기법
    - 구조 요소(Structuring Element)를 이용하여 영상의 구성 성분을 추출.
- 이진 모폴로지, 명암 모폴로지

- 집합 이론
  - 소문자 $a=(a_x,a_y)$: 특정 픽셀
  - 대문자 A : 서로 연결된 픽셀의 집합
- 팽창$_{\text{dilation}}$, 침식$_{\text{erosion}}$, 열림$_{\text{opening}}$, 닫힘$_{\text{closing}}$
  - 팽창은 작은 홈을 메우거나 끊어진 영역을 연결하는 효과. 영역을 키움
  - 침식은 경계에 솟은 돌출 부분을 깎는 효과. 영역을 작게 만듦
  - 열림은 침식한 결과에 팽창 적용. 원래 영역 크기 유지
  - 닫힘은 팽창한 결과에 침식을 적용. 원래 영역 크기 유지

이진 영상의 모폴로지 연산

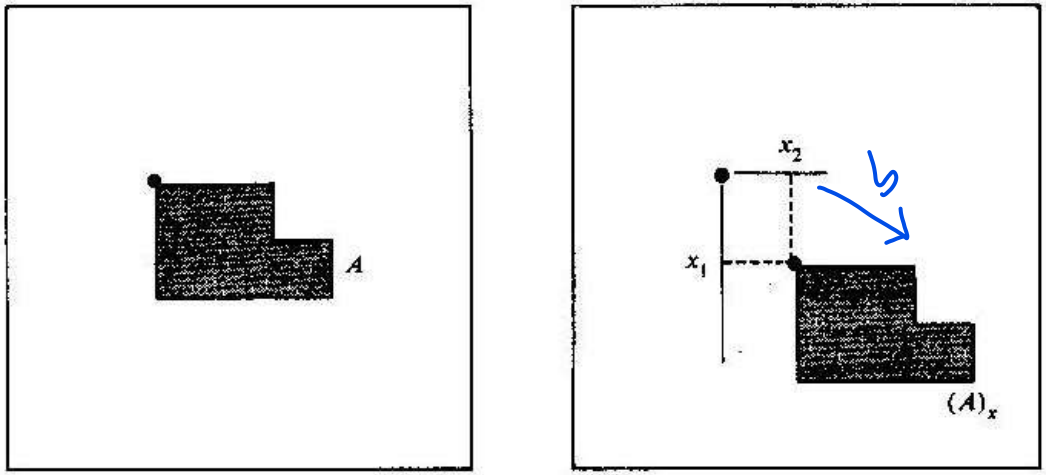
- 집합 이론(con't)
  - 포함 관계 $a \in A,\ a \notin A,\ B \subset A,\ B \subseteq A$
  - 이동(translation) $(A)_b = \{c \mid c = a + b,\ \text{for } a \in A\}$ (A픽셀을 b만큼 이동)
  - 대칭(reflection) $\hat{B} = \{x \mid x = -b,\ \text{for } b \in B\}$
  - 여집합(complement) $A^c = \{x \mid x \notin A\}$
  - 차집합(difference) $A - B = \{x \mid x \in A, x \notin B\} = A \cap B^c$

A, B : binary images

- basic definition

1) translation of A by b

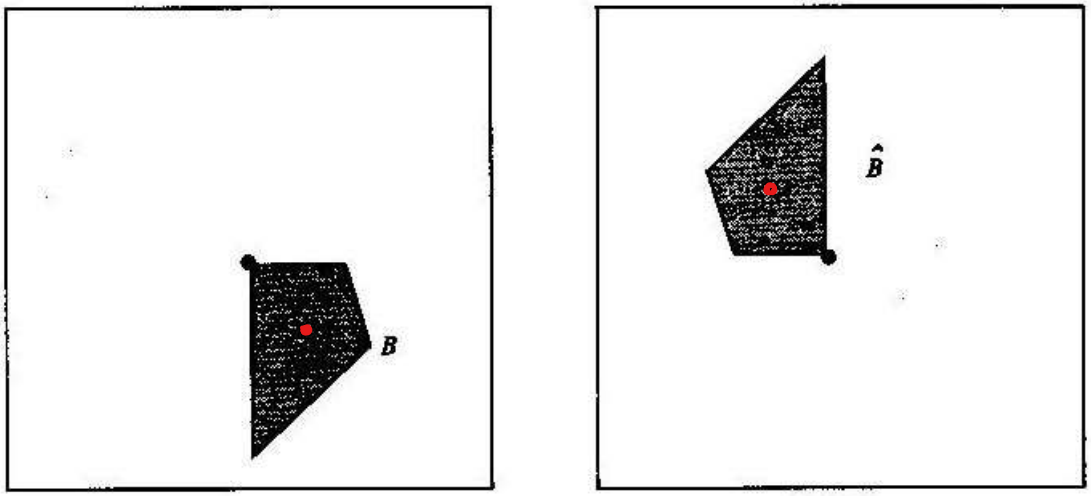
$(A)_b=\{c|c=a+b, \text{ for } a\in A\}$



2) reflection of B

$$\hat{B} = \{x \mid x = -b, \quad \text{for} \quad b \in B\}$$

참고로 원점은 물체의 중심



(c) (d)

이진 모폴로지

- 구조 요소: 모폴로지 연산의 결과를 조절하는 커널, 어두운 픽셀이 원점

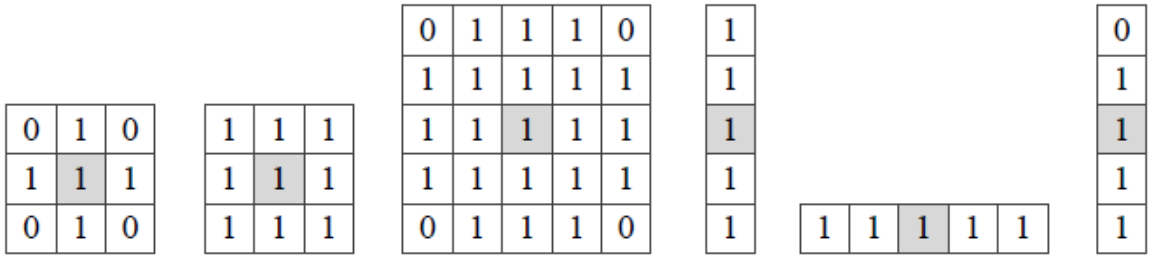


그림 2-36 몇 가지 대표적인 구조요소

- 팽창dilation, 침식erosion, 열림opening, 닫힘closing

$$S_x = \{s + x \mid s \in S\}$$

f의 팽창: $f \oplus S = \mathrm{U}_{x \in f} S_x$

f의 침식: $f \ominus S = \mathrm{U}_{x \in f}\{x \mid S_x \subset f\}$

열기: $f \circ S = (f \ominus S) \oplus S$ ⊖ 침식

닫기: $f \cdot S = (f \oplus S) \ominus S$ ⊕ 팽창

이진 모폴로지

예제 2-5 모폴로지 연산(팽창, 침식, 열기, 닫기)

[그림 2-37]은 간단한 예제 영상과 1×3 크기의 가로 방향의 구조요소를 보여준다.

$f=$

0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	0
0	1	1	1	0	0	1	0
0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0
0	0	0	0	0	0	0	0

(a) 원래 영상

a의 1에, b를 올림

$S=$ | 1 | 1 | 1 |

(b) 구조요소

그림 2-37 예제 영상과 구조요소

0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1
1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0

(a) 팽창($f \oplus S$)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0

(b) 침식($f \ominus S$)

a에 안정적으로 들어가는 위치

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0
0	0	0	0	0	0	0	0

(c) 열기($f \circ S$)

1. 침식
2. 팽창

0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0
0	0	0	0	0	0	0	0

(d) 닫기($f \cdot S$)

1. 팽창
2. 침식

← 효과를 분석해 보자.

그림 2-38 모폴로지 연산 적용 결과

이진 영상의 모폴로지 연산

- 이진 영상의 침식(erosion) 연산 : 객체 축소

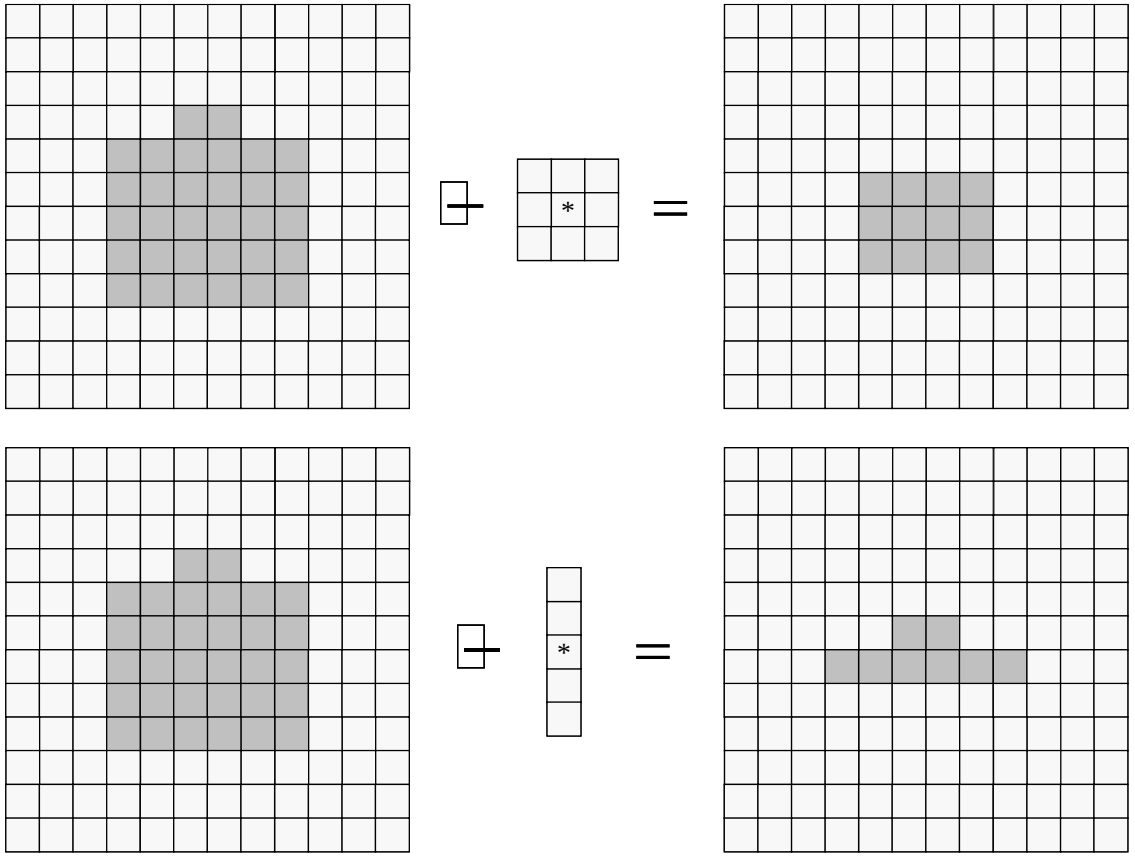
$$A \boxminus B = \{x \mid (B)_x \subset A\}$$

- 이진 영상의 팽창(dilation) 연산 : 객체 확대

$$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \varnothing\} = \{x \mid [(\hat{B})_x \cap A] \subseteq A\}$$

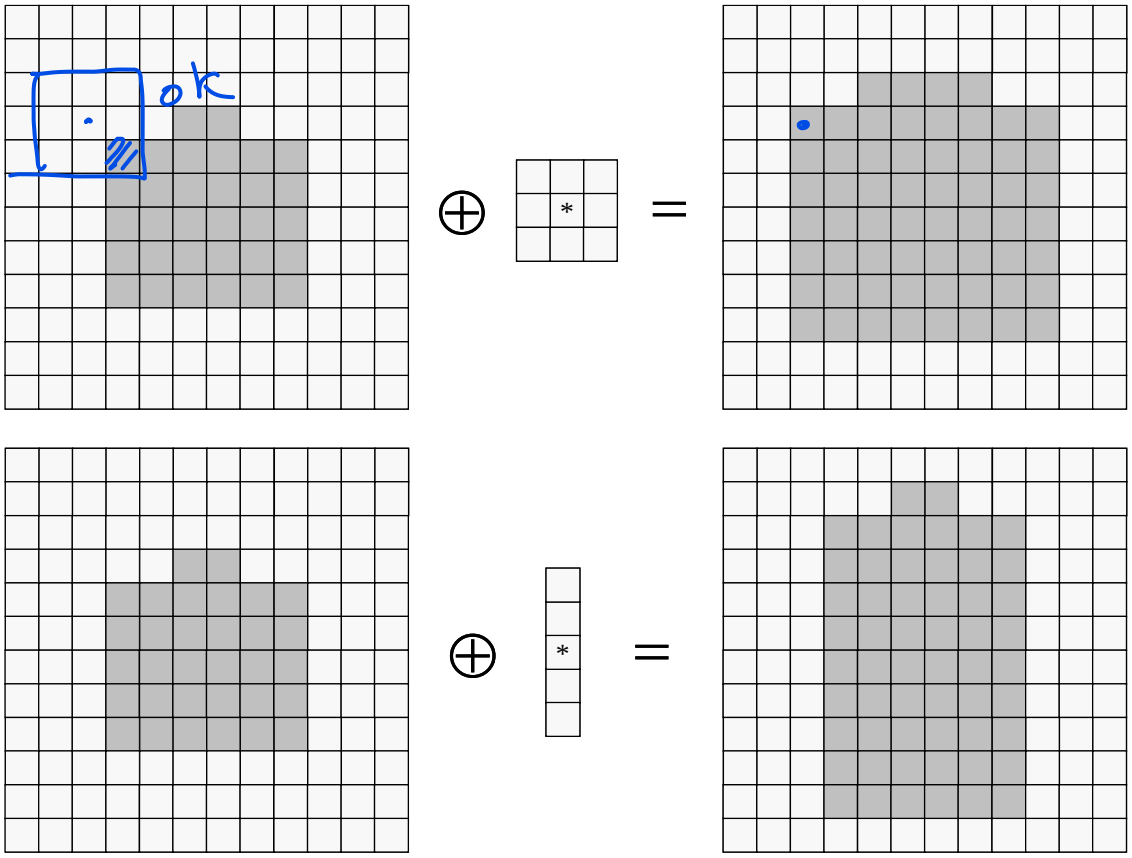
이진 영상의 모폴로지 연산

- 다양한 구성 요소에 따른 침식 연산 결과



이진 영상의 모폴로지 연산

- 다양한 구성 요소에 따른 팽창 연산 결과



형태학 필터를 이용한 영상 열림과 닫힘

- 이진 영상의 열기(opening) 연산
  - 침식 연산 후 팽창 연산 수행

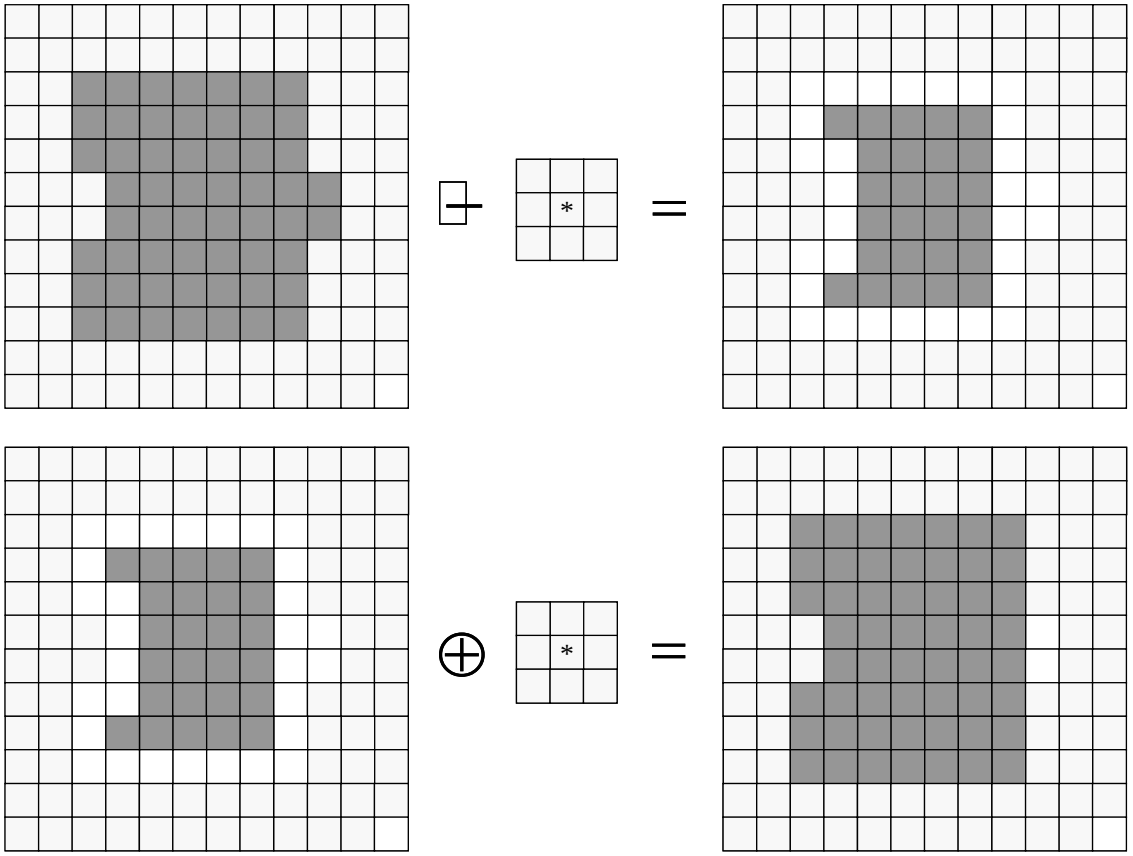
$$A \circ B = (A \ominus B) \oplus B$$

- 이진 영상의 닫기(closing) 연산
  - 팽창 연산 후 침식 연산 수행

$$A \bullet B = (A \oplus B) \ominus B$$

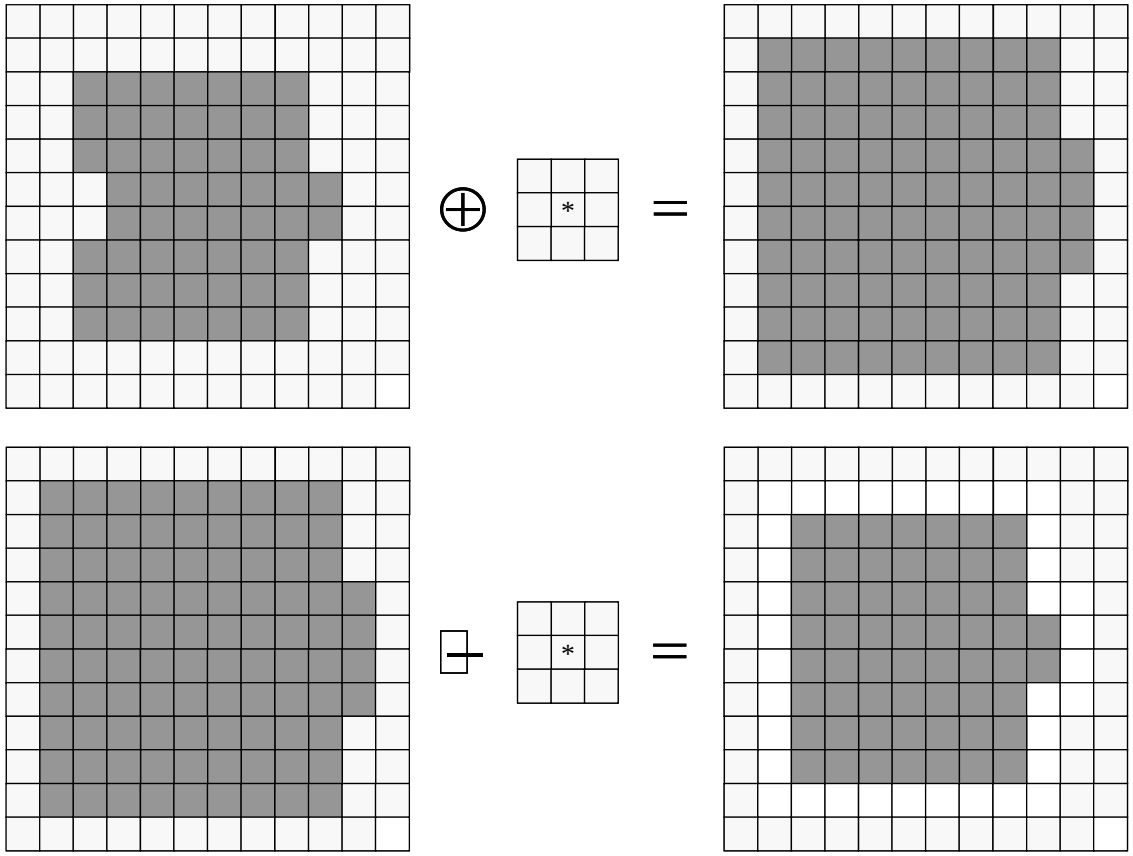
형태학 필터를 이용한 영상 열림과 닫힘

- 이진 영상의 열기 연산 과정 및 결과



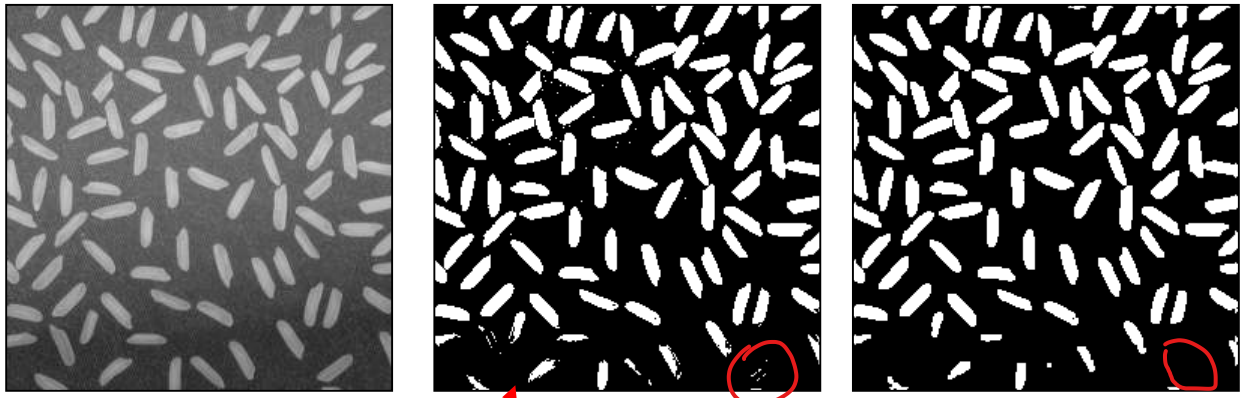
형태학 필터를 이용한 영상 열림과 닫힘

- 이진 영상의 닫기 연산 과정 및 결과



형태학 필터를 이용한 영상 열림과 닫힘

- 열기 연산을 이용한 잡음 제거



레이블링 수행 결과, 레이블의 개수가 186 개로 나타남 (잡음의 영향)

열기 연산 수행 후 레이블링 결과, 레이블의 개수가 96개로 나타남 (비교적 정확한 결과)

Opening and Closing

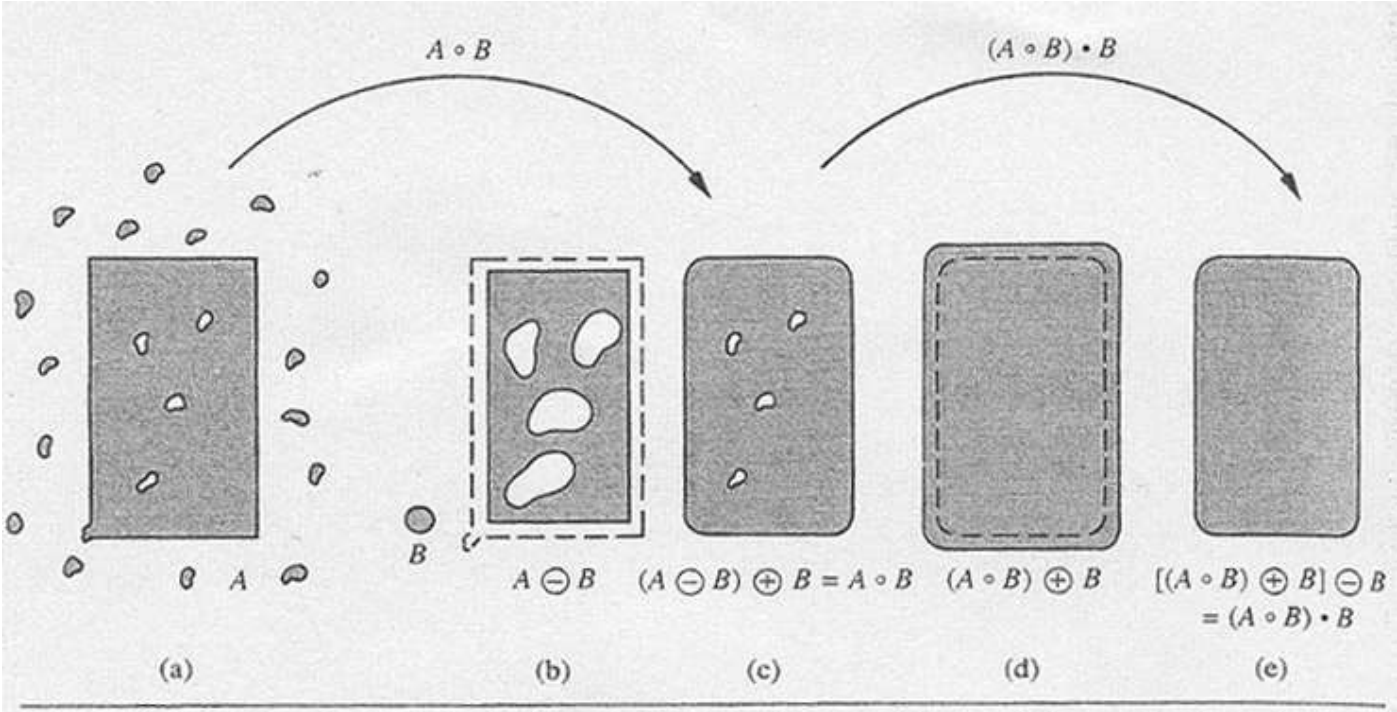


Figure 8.31 *Morphological filtering: (a) original, noisy image; (b) result of erosion; (c) opening of A; (d) result of performing dilation on the opening; (e) final result showing the closing of the opening. (Adapted from Giardina and Dougherty [1988].)*

◆ erode()

dilate() 는 erode()와 동일한 매개변수

```
void cv::erode ( InputArray     src,
                 OutputArray    dst,
                 InputArray     kernel,
                 Point          anchor = Point(-1,-1) ,
                 int            iterations = 1 ,
                 int            borderType = BORDER_CONSTANT ,
                 const Scalar & borderValue = morphologyDefaultBorderValue()
               )
```

Python:

```
cv.erode( src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]] ) -> dst
```

- **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F` or ``CV_64F.
- **dst** – output image of the same size and type as src.
- **element(kernel)** – structuring element used for erosion; if element=Mat() , a 3 x 3 rectangular structuring element is used.
- **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center.
- **iterations** – number of times erosion is applied.
- **borderType** – pixel extrapolation method (see borderInterpolate() for details).
- **borderValue** – border value in case of a constant border (see createMorphologyFilter() for details).

3.2.4 모폴로지

프로그램 3-4 모폴로지 연산 적용하기

```
01  import cv2 as cv
02  import numpy as np
03  import matplotlib.pyplot as plt
04
05  img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
06
07  t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
08  plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([]) ①
09  plt.show()
```

matplotlib.pyplot.imshow(*X, cmap=None, norm=None, *, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, interpolation_stage=None, filternorm=True, filterrad=4.0, resample=None, url=None, data=None, **kwargs*) [sourc

Display data as an image, i.e., on a 2D regular raster.

The input may either be actual RGB(A) data, or 2D scalar data, which will be rendered as a pseudocolor image. For displaying a grayscale image set up the colormapping using the parameters `cmap='gray', vmin=0, vmax=255`.

matplotlib.pyplot.xticks(*ticks=None, labels=None, *, minor=False, **kwargs*)

Get or set the current tick locations and labels of the x-axis. [so

3.2.4 모폴로지

```
10
11  b=bin_img[bin_img.shape[0]//2:bin_img.shape[0],0:bin_img.shape[0]//2+1]
12  plt.imshow(b,cmap='gray'), plt.xticks([]), plt.yticks([]) ②
13  plt.show()
14
15  se=np.uint8([[0,0,1,0,0],                                    # 구조 요소
16               [0,1,1,1,0],
17               [1,1,1,1,1],
18               [0,1,1,1,0],
19               [0,0,1,0,0]])
20
21  b_dilation=cv.dilate(b,se,iterations=1)                       # 팽창
22  plt.imshow(b_dilation,cmap='gray'), plt.xticks([]), plt.yticks([]) ③
23  plt.show()
24
25  b_erosion=cv.erode(b,se,iterations=1)                         # 침식
26  plt.imshow(b_erosion,cmap='gray'), plt.xticks([]), plt.yticks([]) ④
27  plt.show()
28
29  b_closing=cv.erode(cv.dilate(b,se,iterations=1),se,iterations=1) # 닫기
30  plt.imshow(b_closing,cmap='gray'), plt.xticks([]), plt.yticks([]) ⑤
31  plt.show()
```

3.2.4 모폴로지

①

②

잘라낸 영상

③

팽창

④

침식

⑤

닫힘

영상 처리의 세 가지 기본 연산

점 연산

- 오직 자신의 명암 값에 따라 새로운 값을 결정

영역 연산

- 이웃 화소의 명암 값에 따라 새로운 값 결정

기하 연산

- 일정한 기하 규칙으로 결정된 화소(다른 픽셀)의 명암 값에 따라 새로운 값 결정

점 연산

- 점 연산을 식으로 쓰면,
  - 대부분은 k=1 (즉 한 장의 영상을 변환)

$$f_{out}(j,i) = t(f_1(j,i), f_2(j,i), \cdots f_k(j,i)) \qquad (2.10)$$

- 선형 연산
  - 예)

$$f_{out}(j,i) = t(f(j,i))$$

$$= \begin{cases} \min(f(j,i)+a, L-1), & \text{(밝게)} \\ \max(f(j,i)-a, 0), & \text{(어둡게)} \\ (L-1)-f(j,i), & \text{(반전)} \end{cases}$$

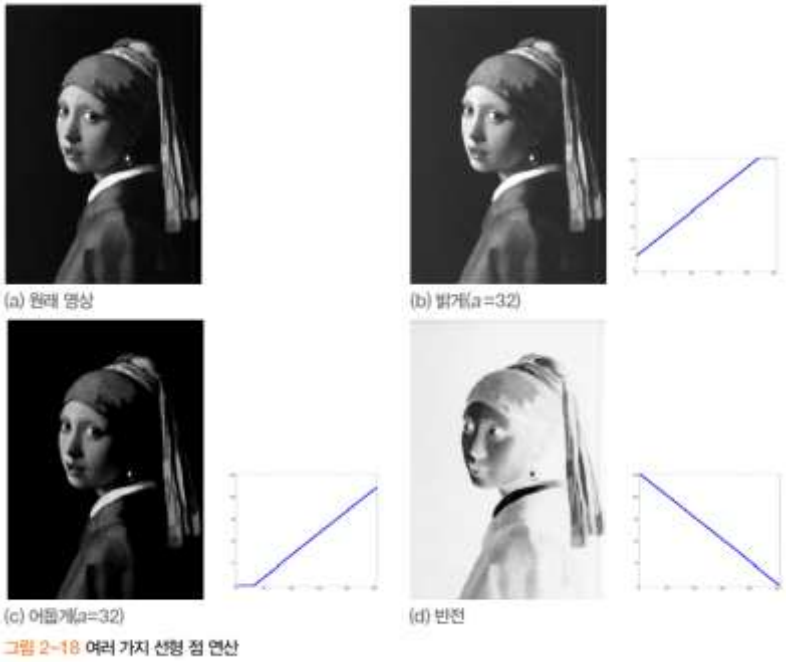


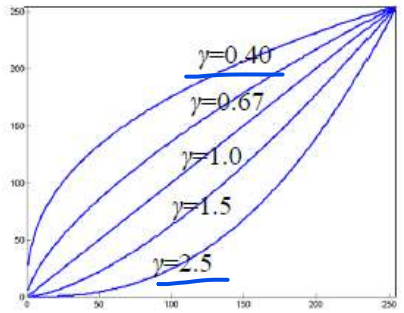
그림 2-18 여러 가지 선형 점 연산

점 연산

- 비선형 연산
  - 예) 감마 수정 (모니터나 프린터 색상 조절에 사용): 모니터는 실제 밝기 보다 어둡게 표현
  - 이를 바로 잡아주는 것이 감마 수정임; $\gamma = 1/2.5$

$$f_{out}(j,i) = (L-1) \times (\hat{f}(j,i))^{\gamma} \quad \text{이때} \quad \hat{f}(j,i) = \frac{f(j,i)}{(L-1)}$$

γ 클수록 어두워짐



(a) 감마값에 따른 변환 함수의 모양

(b) γ=0.40

(c) γ=0.67

(d) γ=1.0(원본 영상)

(e) γ=1.5

(f) γ=2.5

그림 2-19 감마 수정

점 연산

- 장면 디졸브: 영상 f_1 이 영상 f_2로 서서히 전환 됨
  - k=2인 경우 α=1.0 에서 0.2씩 감소

$$f_{out}(j,i) = \alpha f_1(j,i) + (1-\alpha) f_2(j,i) \qquad (2.13)$$

그림 2-21 디졸브 효과

3.3.1 명암 조절

■ 프로그래밍 실습: 감마 보정

프로그램 3-5 감마 보정 실험하기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')
05  img=cv.resize(img,dsize=(0,0),fx=0.25,fy=0.25)
06
07  def gamma(f,gamma=1.0):
08      f1=f/255.0                          # L=256이라고 가정
09      return np.uint8(255*(f1**gamma))
10
11  gc=np.hstack((gamma(img,0.5),gamma(img,0.75),gamma(img,1.0),gamma(img,2.0),gamma
            (img,3.0)))
12  cv.imshow('gamma',gc)
13
14  cv.waitKey()
15  cv.destroyAllWindows()
```

numpy.float64 형

numpy.uint8 형으로 변환

numpy.hstack 함수로 이어붙이기

3.3.2 히스토그램 평활화

■ **히스토그램 평활화**histogram equalization

- 히스토그램이 평평하게 되도록 영상을 조작해 영상의 명암 대비를 높이는 기법

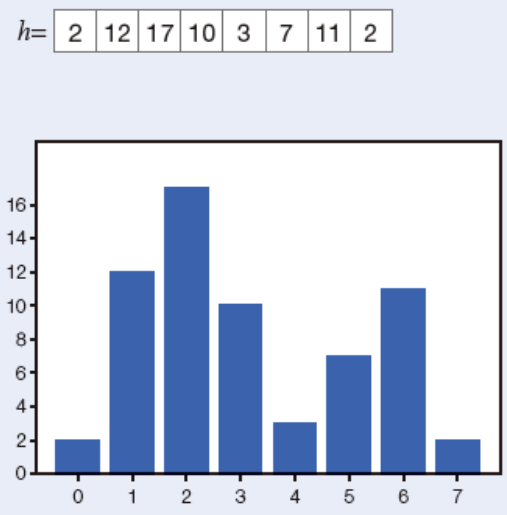
$$l' = \text{round}\left(\ddot{h}(l) \times (L-1)\right) \qquad (3.6)$$

누적 정규화 히스토그램 사용

■ **[예시 3-4] ([그림 3-9]를 재활용)**

1	2	2	2	1	1	2	0
2	6	7	6	6	4	3	0
2	6	7	6	6	4	3	2
2	5	6	6	6	4	3	2
2	5	6	6	5	5	3	2
2	5	5	5	3	3	3	2
2	2	3	3	3	1	1	1
2	2	1	1	1	1	1	1

(a) 입력 영상



(b) 히스토그램

[그림 3-9]

3.3.2 히스토그램 평활화

l	h	$\dot{h}$	$\ddot{h}$	$\ddot{h}\times 7$	l'
0	2	0.03125	0.03125	0.21875	0
1	12	0.1875	0.21875	1.53125	2
2	17	0.265625	0.484375	3.390625	3
3	10	0.15625	0.640625	4.484375	4
4	3	0.046875	0.6875	4.8125	5
5	7	0.109375	0.796875	5.578125	6
6	11	0.171875	0.96875	6.78125	7
7	2	0.03125	1.0	7.0	7

2	3	3	3	2	2	3	0
3	7	7	7	7	5	4	0
3	7	7	7	7	5	4	3
3	6	7	7	7	5	4	3
3	6	7	7	6	6	4	3
3	6	6	6	4	4	4	3
3	3	4	4	4	2	2	2
3	3	2	2	2	2	2	2

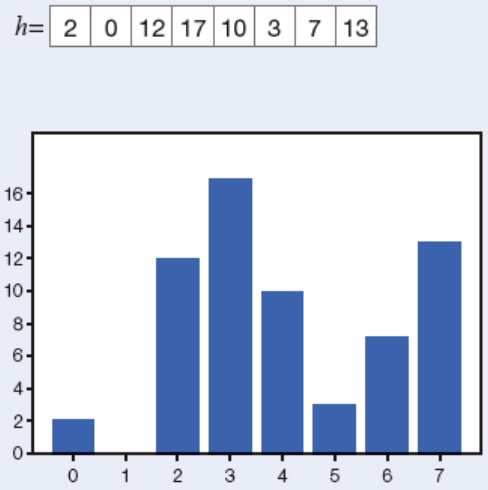


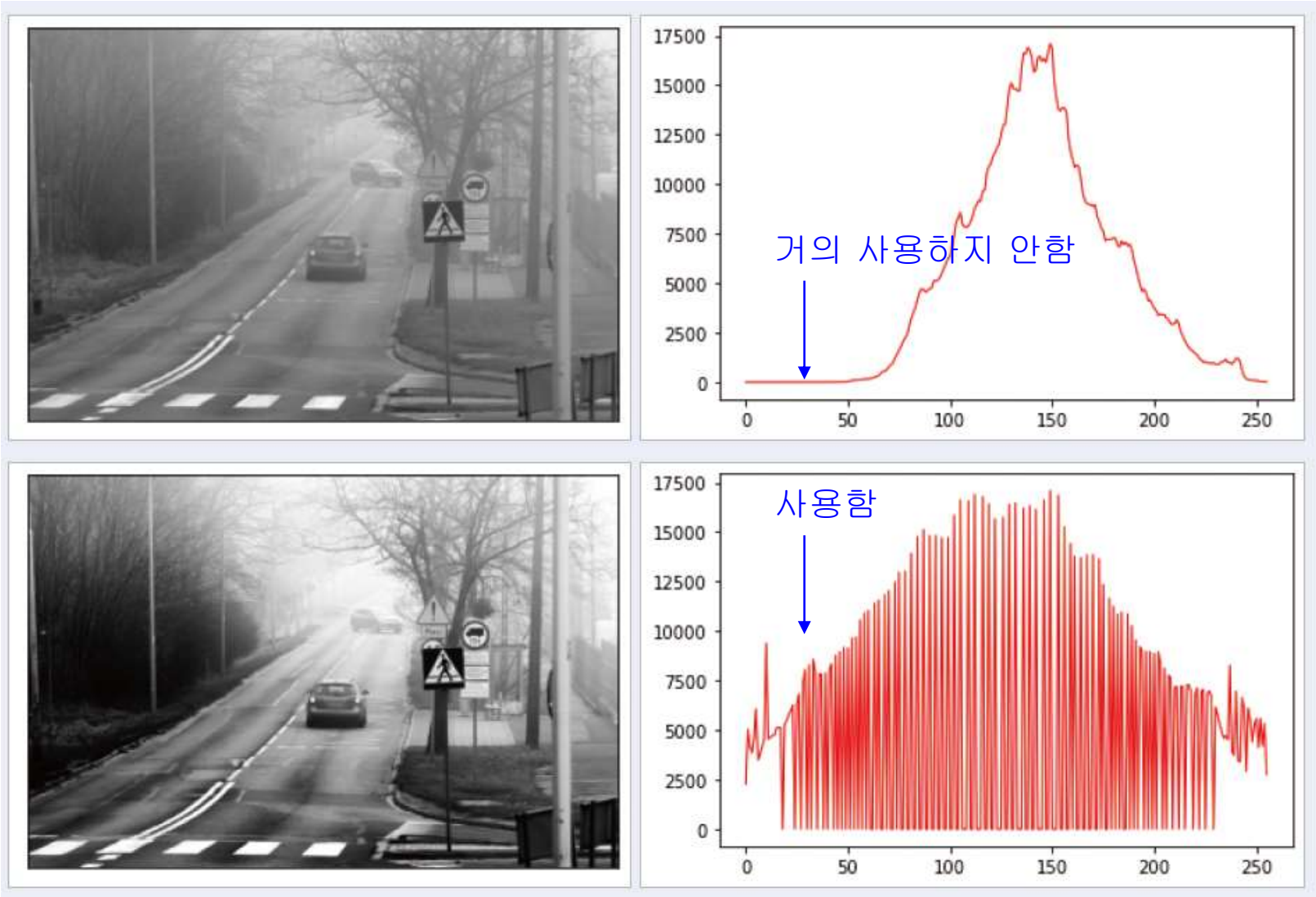
그림 3-15 히스토그램 평활화된 영상

3.3.2 히스토그램 평활화

프로그램 3-6 히스토그램 평활화하기

```
01  import cv2 as cv
02  import matplotlib.pyplot as plt
03
04  img=cv.imread('mistyroad.jpg')
05
06  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)          # 명암 영상으로 변환하고 출력
07  plt.imshow(gray,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()
08
09  h=cv.calcHist([gray],[0],None,[256],[0,256])     # 히스토그램을 구해 출력
10  plt.plot(h,color='r',linewidth=1), plt.show()
11
12  equal=cv.equalizeHist(gray)                      # 히스토그램을 평활화하고 출력
13  plt.imshow(equal,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()
14
15  h=cv.calcHist([equal],[0],None,[256],[0,256])    # 히스토그램을 구해 출력
16  plt.plot(h,color='r',linewidth=1), plt.show()
```

3.3.2 히스토그램 평활화



◆ calcHist() [3/3]

```
void cv::calcHist ( InputArrayOfArrays        images,
                    const std::vector< int > & channels,
                    InputArray                mask,
                    OutputArray               hist,
                    const std::vector< int > & histSize,
                    const std::vector< float > & ranges,
                    bool                      accumulate = false
                  )
```

Python:

```
cv.calcHist( images, channels, mask, histSize, ranges[, hist[, accumulate]] ) -> hist
```

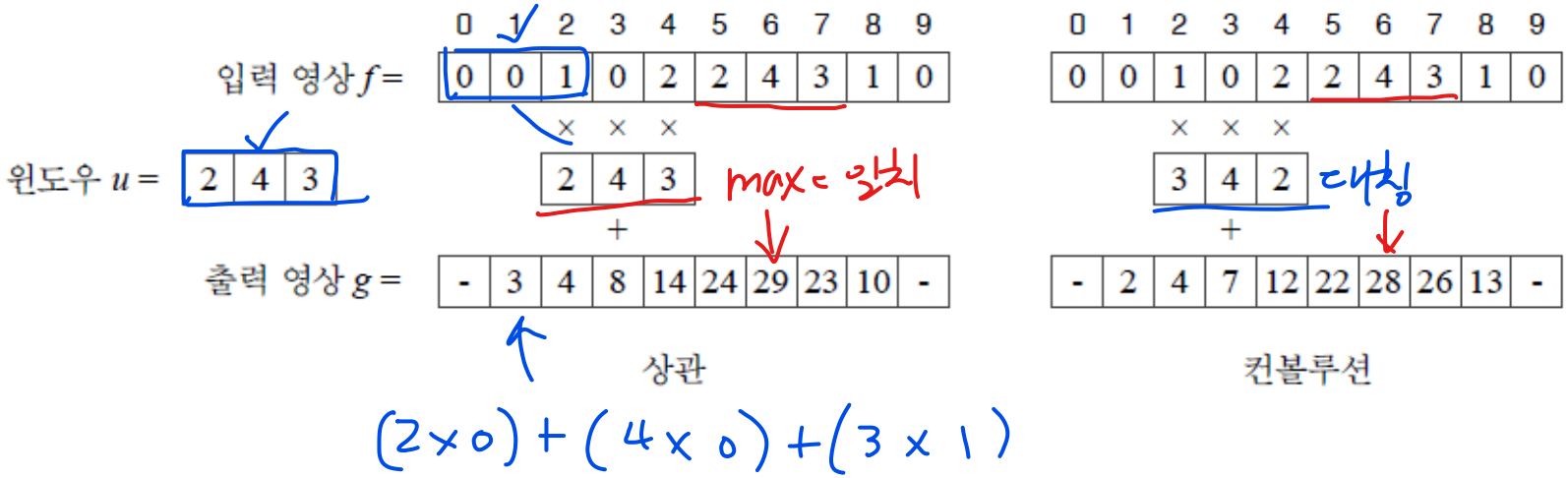
아래는 calcHist의 파라미터 입니다. 배열이라고 표시한 부분은 반드시 리스트로 입력해야 합니다.

- images : 히스토그램을 계산할 영상의 배열입니다. 영상은 같은 사이트, 깊이의 8bit unsigned 정수 또는 32bit 실수형 입니다.
- channels : 히스토그램을 계산할 channel의 배열. (**배열 형태로 입력 필요함**), RGB면 channels이 3개입니다.
- mask : images[i]와 같은 크기의 8bit 이미지로, mask(x, y)가 0이 아닌 경우에만 image[i](x,y)을 히스토그램 계산에 사용합니다.
    - mask = None이면 마스크를 사용하지 않고, 모든 화소에서 히스토그램을 계산합니다.
- histSize : 히스토그램 hist (return 값)의 각 빈(bin) 크기에 대한 정수 배열 입니다.
- ranges : 히스토그램 각 빈의 경계값에 대한 배열입니다. opencv는 기본적으로 등간격 히스토그램을 제공합니다.
- accumulate : True 이면 calcHist() 함수를 수행할 때, 히스토그램을 초기화 하지 않고, 이전 값을 계속 누적합니다.
- hist : 히스토그램 리턴값

영역 연산

상관

- 원시적인 매칭 연산 (물체를 윈도우 형태로 표현하고 물체를 검출): f 에서 u를 검출
- 아래 예에서는 최대값 29를 갖는 위치 6에서 물체가 검출됨

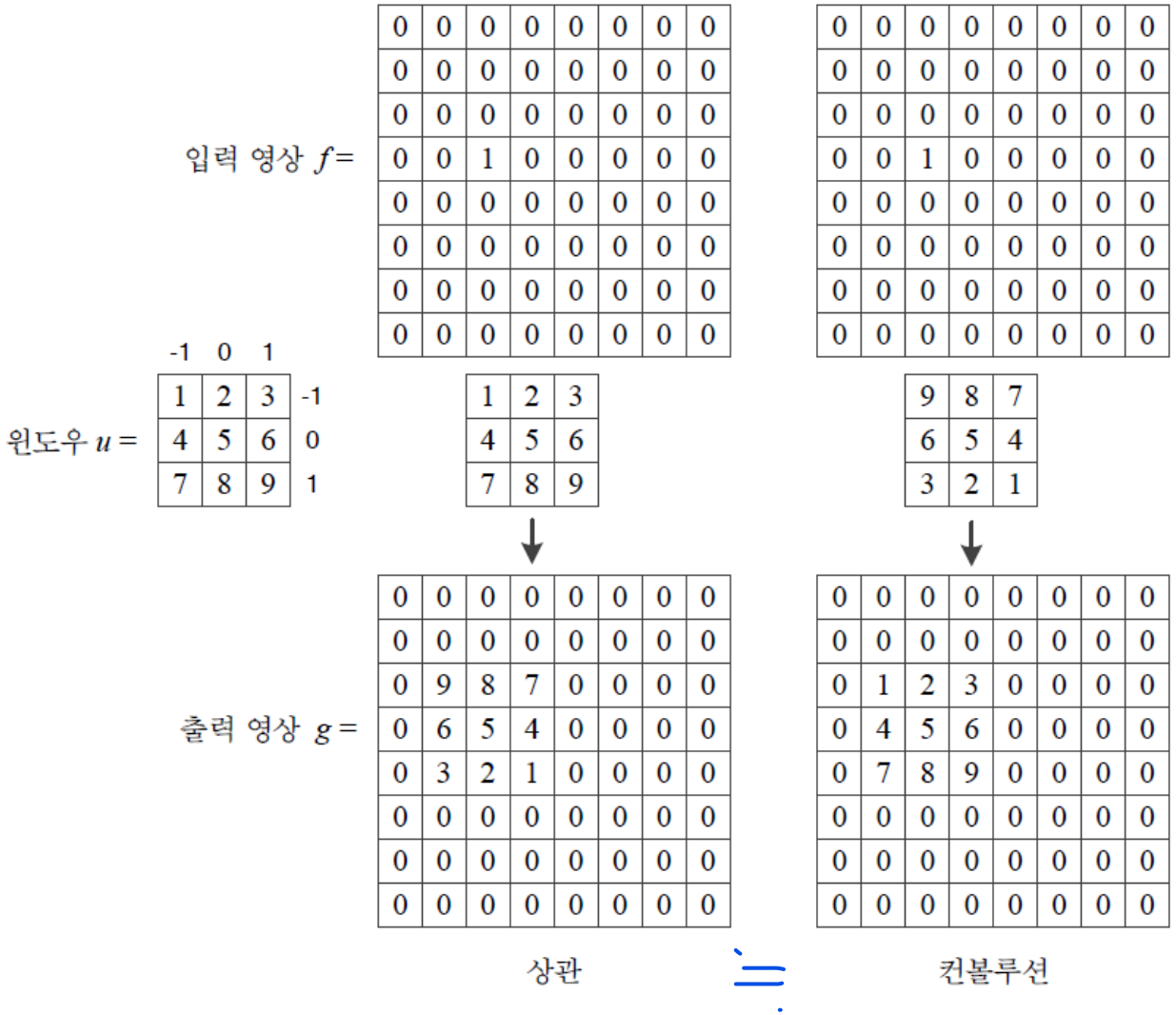


컨볼루션

- 윈도우를 원점에 대해 대칭 이동 후 상관 적용
- 임펄스 반응 함수

영역 연산

- 2차원 영상
  - 단위 임펄스인 경우



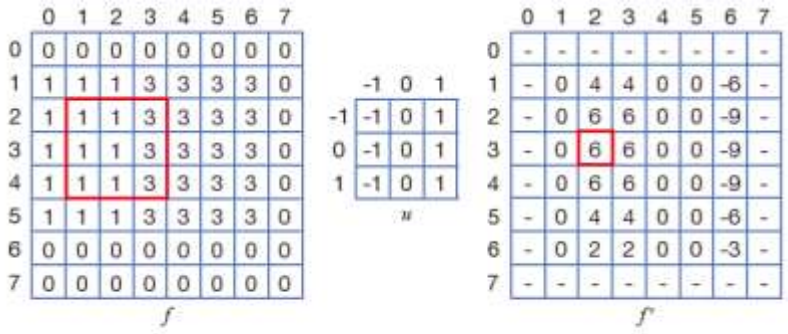
영역 연산

- 수식으로 쓰면,

$$\text{상관 } g(i) = u \otimes f = \sum_{x=-(w-1)/2}^{(w-1)/2} u(x) f(i+x)$$

$$\text{컨볼루션 } g(i) = u \circledast f = \sum_{j=-(w-1)/2}^{(w-1)/2} u(x) f(i-x)$$

1차원



(b) 2차원 영상에 컨볼루션 적용

$$\text{상관 } g(j,i) = u \otimes f = \sum_{y=-(h-1)/2}^{(h-1)/2} \sum_{x=-(w-1)/2}^{(w-1)/2} u(y,x) f(j+y, i+x)$$

$$\text{컨볼루션 } g(j,i) = u \circledast f = \sum_{y=-(h-1)/2}^{(h-1)/2} \sum_{x=-(w-1)/2}^{(w-1)/2} u(y,x) f(j-y, i-x)$$

2차원

- 이 책은 둘 구분하지 않고 상관을 사용하는데 모두 컨볼루션이라는 용어를 사용
- 윈도우는 마스크, 커널, 템플릿, 필터라고도 불림.

영역 연산

- 컨볼루션 예제
  - 오른쪽 박스와 가우시안은 스무딩 효과
  - 샤프닝은 명암 대비 강조 효과
  - 수평 에지와 수직 에지는 에지 검출 효과
  - 모션은 45도 방향의 모션 효과
- 컨볼루션은 선형 연산

박스

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

가우시안

.0000	.0000	.0002	.0000	.0000
.0000	.0113	.0837	.0113	.0000
.0002	.0837	.6187	.0837	.0002
.0000	.0113	.0837	.0113	.0000
.0000	.0000	.0002	.0000	.0000

샤프닝

0	-1	0
-1	5	-1
0	-1	0

수평 에지

1	1	1
0	0	0
-1	-1	-1

수직 에지

1	0	-1
1	0	-1
1	0	-1

모션

.0304	.0501	0	0	0
.0501	.1771	.0519	0	0
0	.0519	.1771	.0519	0
0	0	.0519	.1771	.0501
0	0	0	.0501	.0304

(a) 원래 영상과 여러 가지 마스크들

> 박스

> 가우시안

> 샤프닝

> 수평 에지

> 수직 에지

> 모션

(b) 다양한 마스크로 컨볼루션한 영상들

영역 연산

- 비선형 연산
  - 예) 메디안 필터
    - 솔트페퍼 잡음에 효과적임
    - 메디안은 가우시안에 비해 에지 보존 효과 뛰어남

(a) 원래 영상 (b) 솔트페퍼 잡음

(c) 가우시안 필터 (d) 메디안 필터

그림 2-25 가우시안과 메디안 필터의 비교

3.4.2 다양한 필터

■ 가우시안 필터

$$\text{1차원 가우시안: } g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

$$\text{2차원 가우시안: } g(y,x) = \frac{1}{\sigma^2 2\pi} e^{-\frac{y^2+x^2}{2\sigma^2}} \qquad (3.9)$$

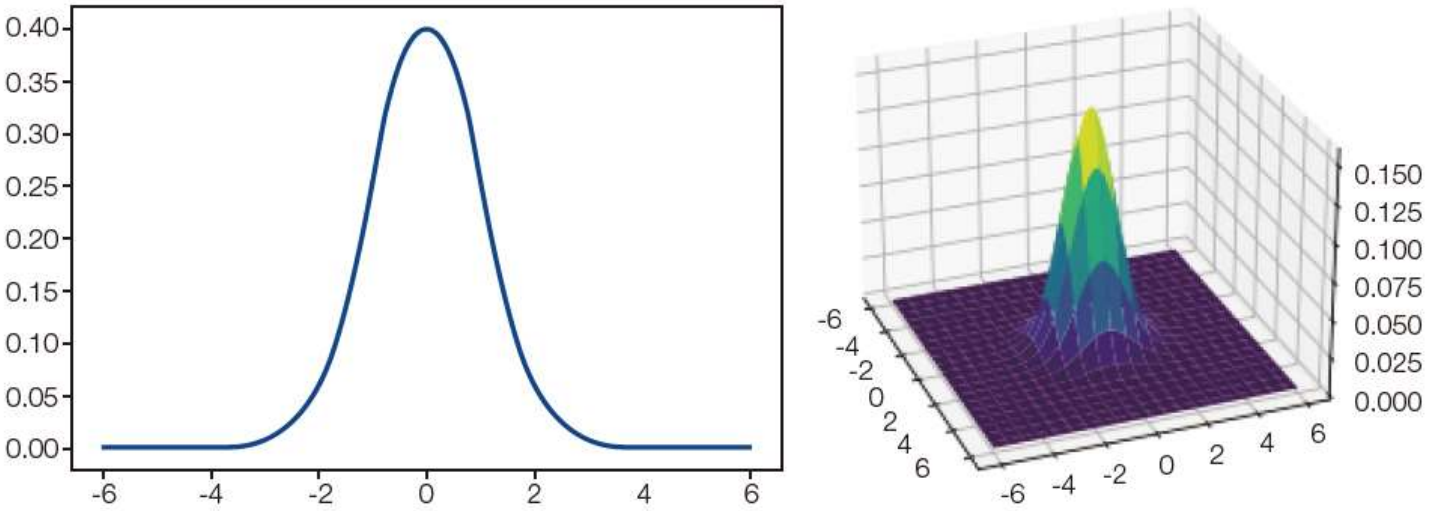


그림 3-18 1차원과 2차원 가우시안 함수

Gaussian filter

◆ GaussianBlur()

```
void cv::GaussianBlur ( InputArray  src,
                        OutputArray dst,
                        Size        ksize,
                        double      sigmaX,
                        double      sigmaY = 0 ,
                        int         borderType = BORDER_DEFAULT
                      )
```

Python:

```
cv.GaussianBlur( src, ksize, sigmaX[, dst[, sigmaY[, borderType]]] ) -> dst
```

- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
- **dst** – output image of the same size and type as src.
- **ksize** – Gaussian kernel size. ksize.width and ksize.height can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from sigma* .
- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.height , respectively (see getGaussianKernel() for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, sigmaX, and sigmaY.

데이터 형과 컨볼루션

■ 연산 결과를 저장하는 변수의 유효 값 범위

- OpenCV는 주의를 기울여 작성되어 있음
- 때로 프로그래머가 직접 신경 써야 하는 경우 있음. 예) filter2D 함수

■ 데이터 형

- Opencv는 영상 화소를 주로 numpy.uint8 형으로 표현 ([0,255] 범위)

```
In [1]: print(type(img[0,0,0]))
        numpy.uint8
```

- [0,255] 범위를 벗어나는 경우 문제 발생

```
In [2]: a=np.array([-3,-2,-1,0,1,254,255,256,257,258],dtype=np.uint8)
In [3]: print(a)
        [253 254 255   0   1 254 255   0   1   2]
```

- 예) 엠보싱의 경우 [-255~255] 발생하는데 어떻게 해결하나?

3.4.3 데이터 형과 컨볼루션

프로그램 3-7 컨볼루션 적용(가우시안 스무딩과 엠보싱)하기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')
05  img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
06  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
07  cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
08  cv.imshow('Original',gray) ①
09
10  smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),cv.
                  GaussianBlur(gray,(9,9),0.0),cv.GaussianBlur(gray,(15,15),0.0)))
11  cv.imshow('Smooth',smooth) ②
12
13  femboss=np.array([[-1.0, 0.0, 0.0],
14                    [ 0.0, 0.0, 0.0],
15                    [ 0.0, 0.0, 1.0]])
16
17  gray16=np.int16(gray)
18  emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
19  emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
20  emboss_worse=cv.filter2D(gray,-1,femboss)
21
22  cv.imshow('Emboss',emboss) ③
23  cv.imshow('Emboss_bad',emboss_bad) ④
24  cv.imshow('Emboss_worse',emboss_worse) ⑤
25
26  cv.waitKey()
27  cv.destroyAllWindows()
```

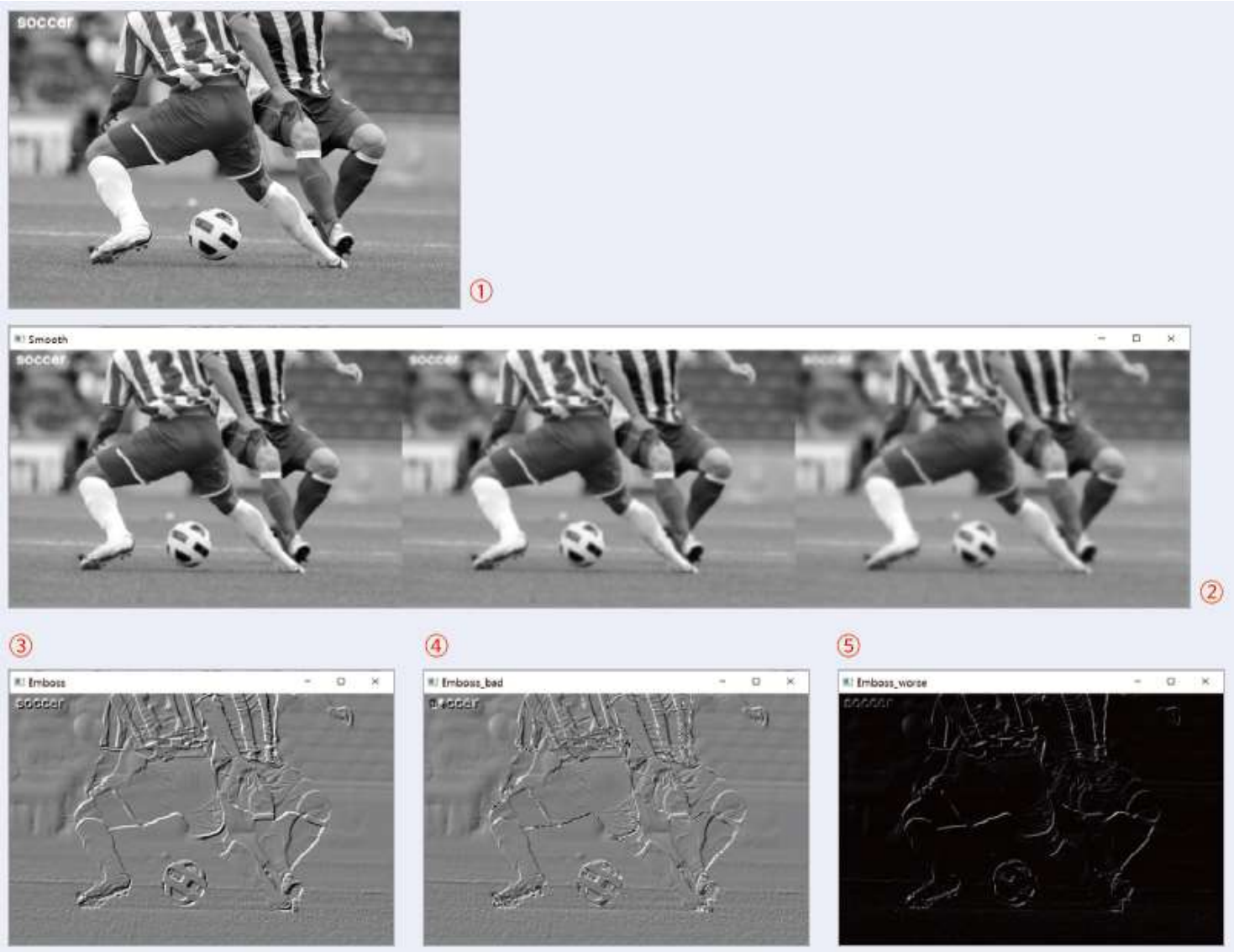
numpy.clip(array, min, max)

array 내의 element들에 대해서

min 값 보다 작은 값들을 min값으로 바꿔주고

max 값 보다 큰 값들을 max값으로 바꿔주는 함수.

3.4.3 데이터 형과 컨볼루션



엠보싱(embossing)

- **엠보싱** (embossing) 은 직물, 종이, 금속 등의 표면에 돋을새김으로 무늬를 찍어내는 가공 방법
- 입력영상에서 픽셀 값 변화가 적은 평탄한 영역은 회색으로 설정하고 객체의 경계 부분은 좀 더 밝거나 어둡게 설정

dst = cv2.filter2D(src, ddepth, kernel, dst, anchor, delta, borderType)
src: 입력 영상, Numpy 배열
ddepth: 출력 영상의 dtype (-1: 입력 영상과 동일)
kernel: 컨볼루션 커널, float32의 n x n 크기 배열
dst(optional): 결과 영상
anchor(optional): 커널의 기준점, default: 중심점 (-1, -1)
delta(optional): 필터가 적용된 결과에 추가할 값
borderType(optional): 외곽 픽셀 보정 방법 지정

```
-1 -1 0        0 -1 -1
-1  0 1        1  0 -1
 0  1 1        1  1  0
```

3.5 기하 연산

- **영상 처리 연산: 화소에 새로운 값을 결정하는 과정**
- **기하 연산**
  - 기하학적 변환에 따라 정해진 위치의 화소에서 값을 가져옴
  - 주로 물체의 이동, 크기, 회전에 따른 기하 변환

3.5.1 동차 좌표와 동차 행렬

■ 동차 좌표homogeneous coordinate(전위 곱셈)

- (x, y)과 같은 2차원 좌표에 1을 추가해 (x, y, 1)과 같이 3차원 벡터로 표현
- 3개 요소에 같은 값을 곱하면 같은 좌표.
- 예) (-2,4,1), (-4,8,2) 과 (-0.2, 0.4, 0.1)은 (-2,4)에 해당(모두 같은 좌표임)

$$\bar{p} = (x, y, 1) \qquad (3.10)$$

■ 여러 가지 기하 변환

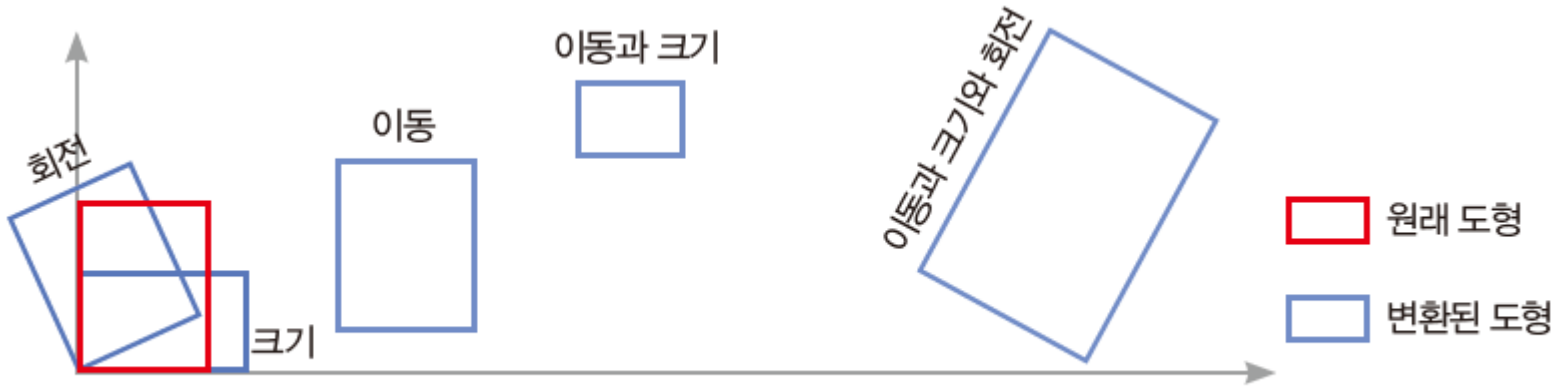


그림 3-19 여러 가지 기하 변환

3.5.1 동차 좌표와 동차 행렬

■ 기하 변환을 동차 행렬homogeneous matrix로 표현(전위 곱셈)

- [표 3-1] 변환은 모두 어파인 변환affine transform: 평행을 평행으로 유지

표 3-1 3가지 기하 변환

기하 변환	동차 행렬	설명
이동	$T(t_x,t_y)=\begin{pmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{pmatrix}$	x 방향으로 t_x, y 방향으로 t_y만큼 이동
회전	$R(\theta)=\begin{pmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}$	원점을 중심으로 반시계 방향으로 θ만큼 회전
크기	$S(s_x,s_y)=\begin{pmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{pmatrix}$	x 방향으로 s_x, y 방향으로 s_x만큼 크기 조정(1보다 크면 확대, 1보다 작으면 축소)

3.5.1 동차 좌표와 동차 행렬

■ **[예시 3-5] 동차 행렬을 이용한 기하 변환**

- 정사각형을 x 방향으로 2, y 방향으로 -1만큼 이동한 다음 반시계 방향으로 30도 회전

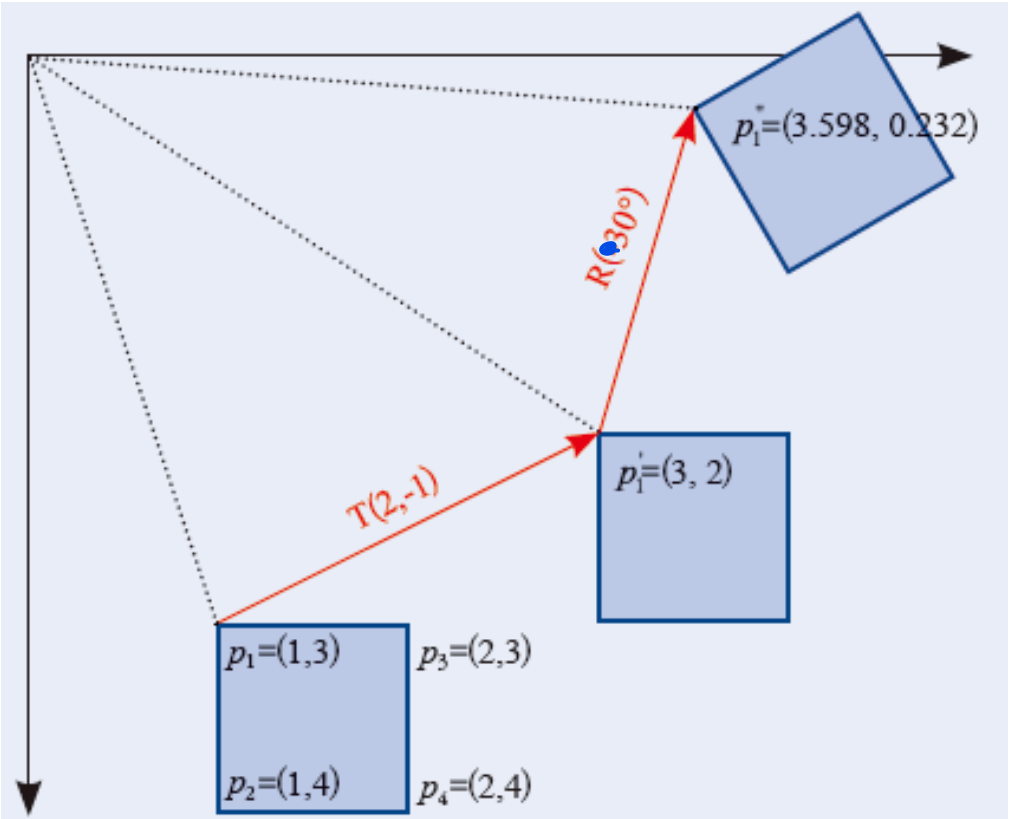


그림 3-20 동차 행렬을 이용한 기하 변환

3.5.1 동차 좌표와 동차 행렬

■ 변환을 위한 동차 행렬(전위 곱셈)

$$\mathrm{T}(2,-1)=\begin{pmatrix}1 & 0 & 2\\ 0 & 1 & -1\\ 0 & 0 & 1\end{pmatrix},\quad \mathrm{R}(30^\circ)=\begin{pmatrix}0.8660 & 0.5000 & 0\\ -0.5000 & 0.8660 & 0\\ 0 & 0 & 1\end{pmatrix}$$

■ 이동 적용

$$\overline{p}_1'^{\mathrm{T}}=\mathrm{T}(2,-1)\,\overline{p}_1^{\mathrm{T}}=\begin{pmatrix}1 & 0 & 2\\ 0 & 1 & -1\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\ 3\\ 1\end{pmatrix}=\begin{pmatrix}3\\ 2\\ 1\end{pmatrix}$$

■ 회전 적용

$$\overline{p}_1''^{\mathrm{T}}=\mathrm{R}(30^\circ)\,\overline{p}_1'^{\mathrm{T}}=\begin{pmatrix}0.8660 & 0.5000 & 0\\ -0.5000 & 0.8660 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}3\\ 2\\ 1\end{pmatrix}=\begin{pmatrix}3.598\\ 0.232\\ 1\end{pmatrix}$$

3.5.1 동차 좌표와 동차 행렬

■ 동차 행렬을 이용하면 계산이 효율적임(전위 곱셈)

- 복합 변환을 위한 행렬을 미리 곱해 놓으면, 모든 점에 대해 한번의 행렬 곱셈으로 기하 변환 가능(행렬 곱셈은 결합 법칙 성립)

$$\mathbf{A} = \mathrm{R}(30°)\mathrm{T}(2,-1) = \begin{pmatrix} 0.8660 & 0.5000 & 0 \\ -0.5000 & 0.8660 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.8660 & 0.5000 & 1.232 \\ -0.5000 & 0.8660 & -1.866 \\ 0 & 0 & 1 \end{pmatrix}$$

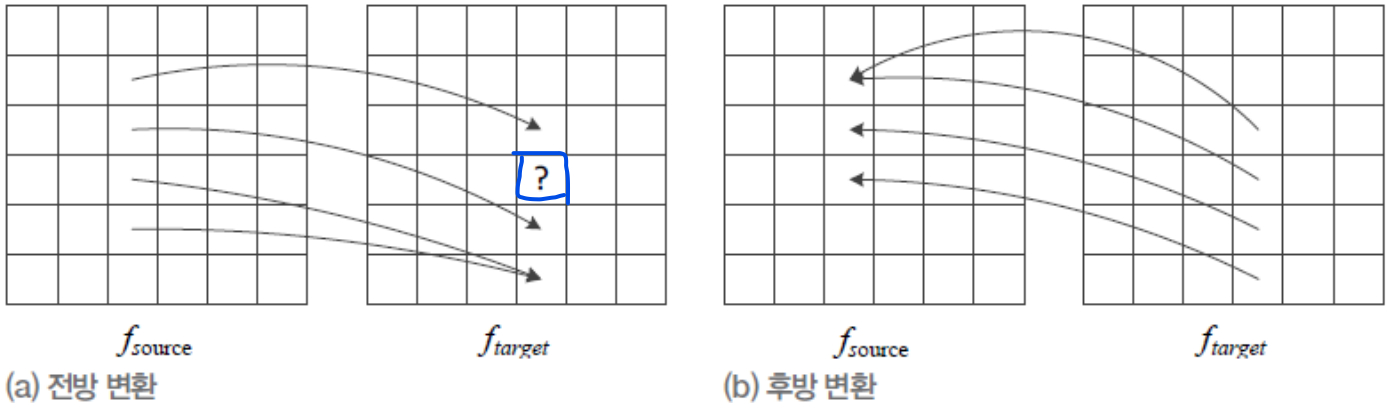
$$\mathbf{A}\bar{p}_1^{\mathrm{T}} = \begin{pmatrix} 0.8660 & 0.5000 & 1.232 \\ -0.5000 & 0.8660 & -1.866 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 3.598 \\ 0.232 \\ 1 \end{pmatrix}$$

■ 임의의 점 (c_x, c_y)를 중심으로 회전

- $T(c_x, c_y)R(\theta)T(-c_x, -c_y)$

3.5.2 영상의 기하 변환

- 기하 연산을 영상에 적용
  - 전방 변환은 값을 받지 못하는 화소가 생기는 에일리어싱aliasing 현상
  - 후방 변환을 이용한 안티 에일리어싱



(a) 전방 변환 (b) 후방 변환

빈 곳의 색깔을 뭐로 채울지 정해줘야

(a) 전방 변환 (b) 후방 변환 거꾸로 찾아감

그림 3-21 영상의 기하 변환

3.5.3 영상 보간

- 보간에 의한 안티 에일리어싱
  - 실수 좌표를 반올림(최근접 이웃)하여 정수로 변환하는 과정에서 에일리어싱 발생
  - 주위 화소 값을 이용한 보간으로 안티 에일리어싱
  - 양선형 보간법: 걸치는 비율에 따라 선형 곱을 함으로써 안티 에일리어싱
  - 양3차 보간: 주변 16개 화소 사용

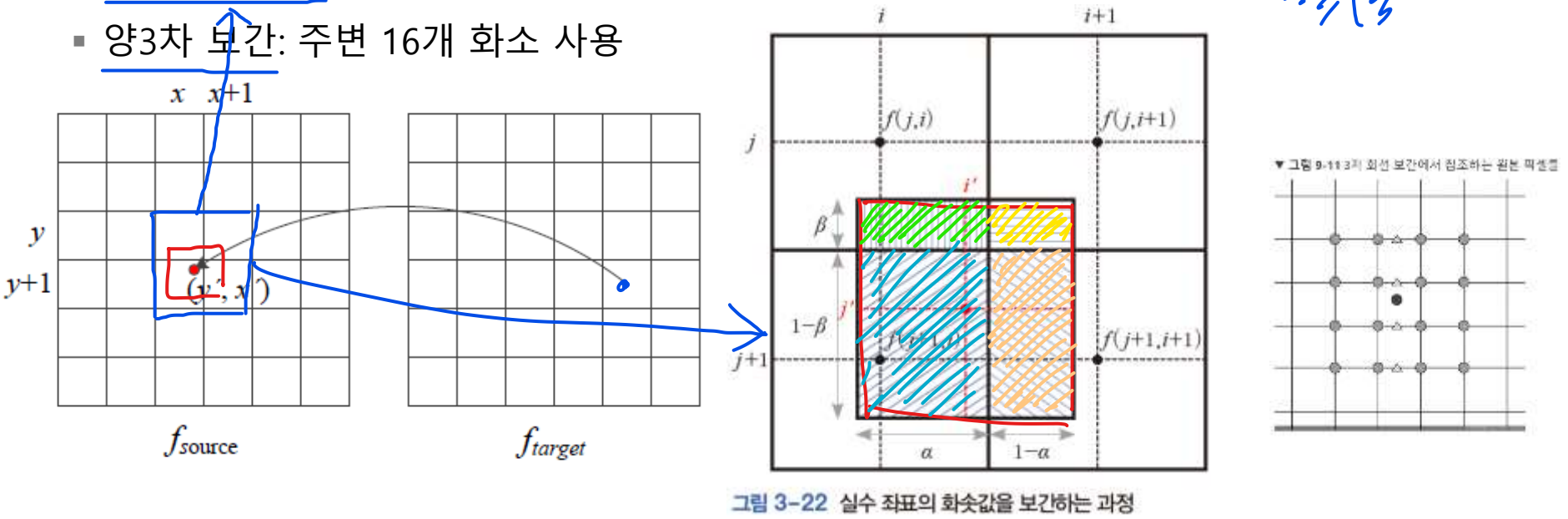


그림 3-22 실수 좌표의 화솟값을 보간하는 과정

$$f(j',i') = \alpha\beta f(j,i) + (1-\alpha)\beta f(j,i+1) + \alpha(1-\beta) f(j+1,i) + (1-\alpha)(1-\beta) f(j+1,i+1) \quad (3.11)$$

$$f(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & 0 \le |x| < 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases}$$

a는 -1.0 ≤ a ≤ -0.5

3.5.3 영상 보간

프로그램 3-8 보간을 이용해 영상의 기하 변환하기

```
01  import cv2 as cv
02
03  img=cv.imread('rose.png')
04  patch=img[250:350,170:270,:]
05
06  img=cv.rectangle(img,(170,250),(270,350),(255,0,0),3)
07  patch1=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_NEAREST)
08  patch2=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_LINEAR)
09  patch3=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_CUBIC)
10
11  cv.imshow('Original',img)
12  cv.imshow('Resize nearest',patch1)
13  cv.imshow('Resize bilinear',patch2)
14  cv.imshow('Resize bicubic',patch3)
15
16  cv.waitKey()
17  cv.destroyAllWindows()
```

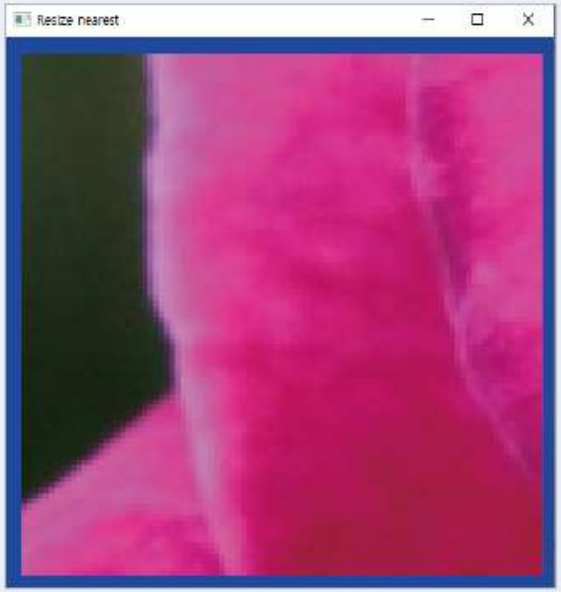
크기변환

① 최근접

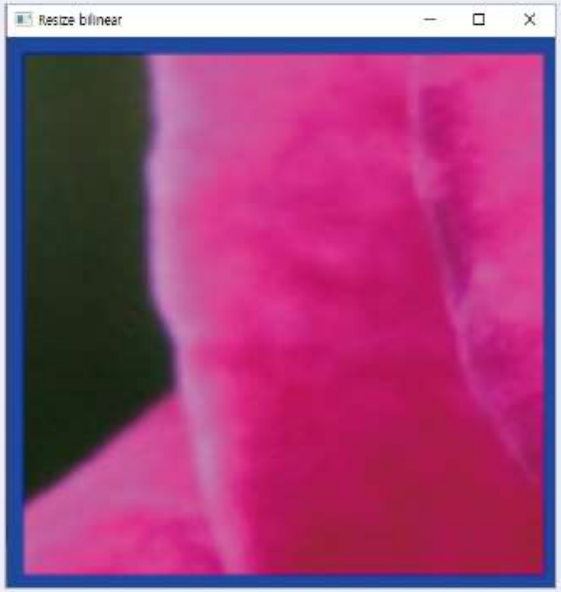
② 양선형 보간법

③ 양3차 보간법

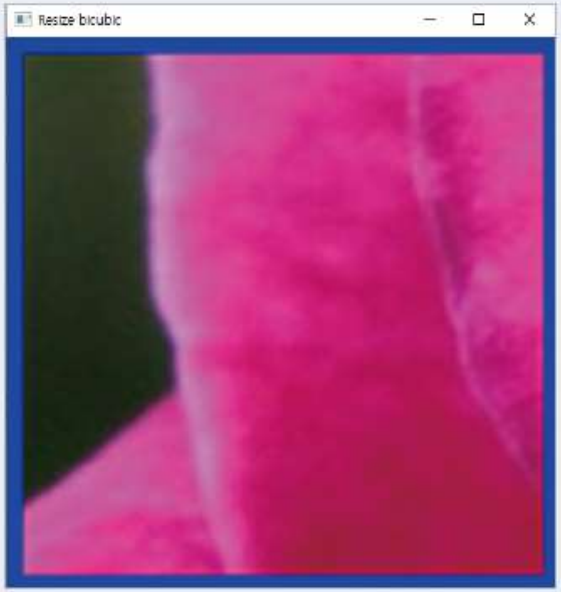
3.5.3 영상 보간



최근접 이웃



양선형 보간



양3차 보간

기하 연산

- 최근접 이웃, 양선형 보간, 양 3차 보간의 비교

(a) 원래 영상과 조각 영상

> 최근접 이웃

> 양선형 보간

> 양 3차 보간

(b) 10° 회전한 영상

3.6 OpenCV의 시간 효율

- 컴퓨터 비전은 인식 정확률 뿐 아니라 시간 효율도 중요
  - 특히 실시간 처리가 요구되는 응용

- **OpenCV**는 효율적으로 구현되었기 때문에 **OpenCV** 함수를 사용하는 것이 유리
  - C/C++로 구현하고 인텔 마이크로프로세서에 최적화

- 직접 구현하는 경우 파이썬의 배열 연산 사용하는 것이 유리

3.6 OpenCV의 시간 효율

프로그램 3-9 직접 작성한 함수와 OpenCV가 제공하는 함수의 시간 비교하기

```
01  import cv2 as cv
02  import numpy as np
03  import time
04
05  def my_cvtGray1(bgr_img):
06      g=np.zeros([bgr_img.shape[0],bgr_img.shape[1]])
07      for r in range(bgr_img.shape[0]):
08          for c in range(bgr_img.shape[1]):
09              g[r,c]=0.114*bgr_img[r,c,0]+0.587*bgr_img[r,c,1]+0.299*bgr_img[r,c,2]
10      return np.uint8(g)
11
12  def my_cvtGray2(bgr_img):
13      g=np.zeros([bgr_img.shape[0],bgr_img.shape[1]])
14      g=0.114*bgr_img[:,:,0]+0.587*bgr_img[:,:,1]+0.299*bgr_img[:,:,2]
15      return np.uint8(g)
16
```

3.6 OpenCV의 시간 효율

```
17  img=cv.imread('girl_laughing.png')
18
19  start=time.time()
20  my_cvtGray1(img)
21  print('My time1:',time.time()-start) ①
22
23  start=time.time()
24  my_cvtGray2(img)
25  print('My time2:',time.time()-start) ②
26
27  start=time.time()
28  cv.cvtColor(img,cv.COLOR_BGR2GRAY)
29  print('OpenCV time:',time.time()-start) ③
```

```
My time1: 4.798288106918335 ①
My time2: 0.015836000442504883 ②
OpenCV time: 0.013601541519165039 ③
```