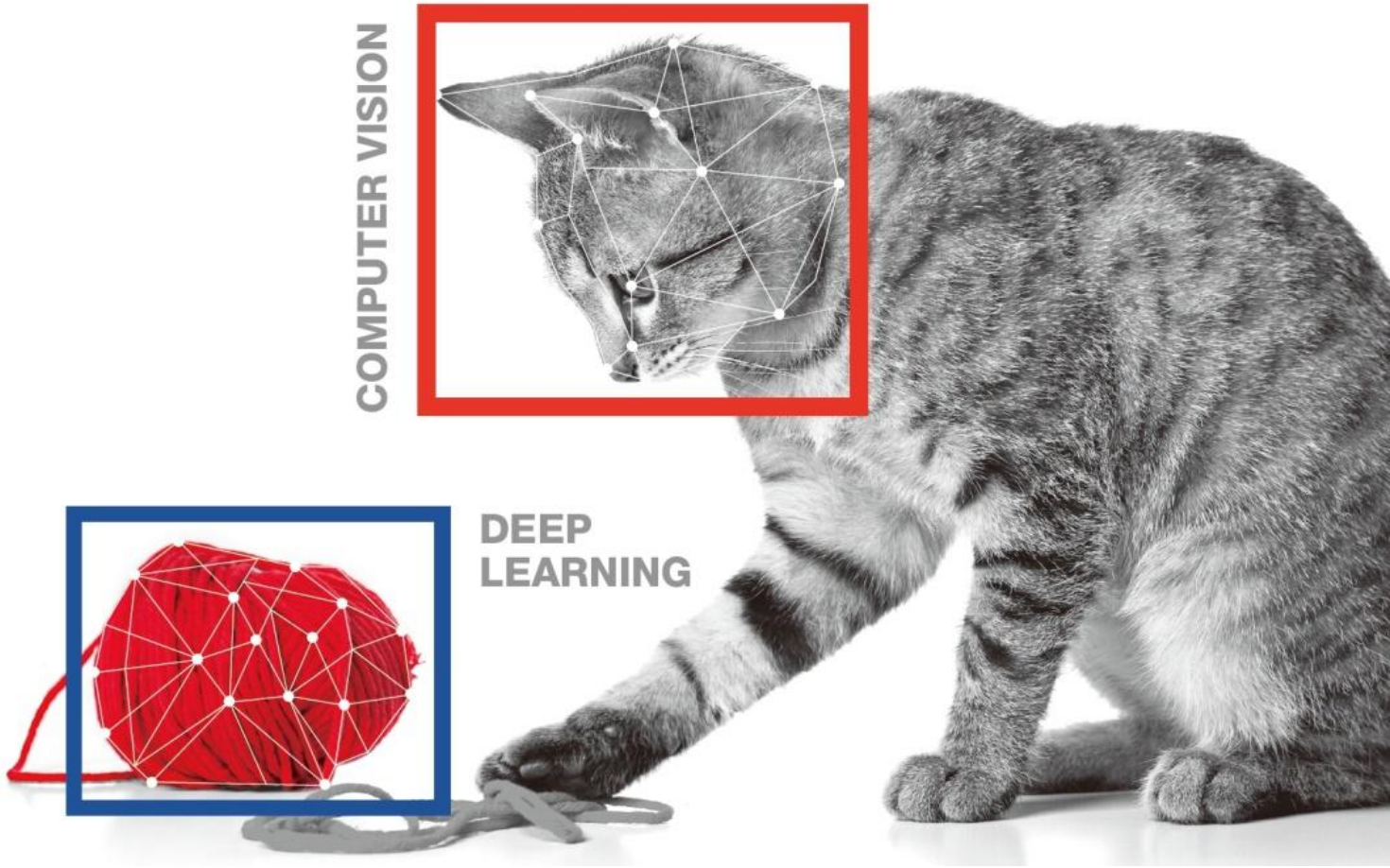


이미지딥러닝

07주차

영상 처리I

목차

- OpenCV Preview
- 디지털 영상 기초

OpenCV Preview

- 현대는 양호한 프로그래밍 환경
  - 예전에는 알고리즘을 바닥부터 구현
  - 현대는 함수 호출로 영상 처리하는 시대. 대표적 컴퓨터 비전 라이브러리는 OpenCV

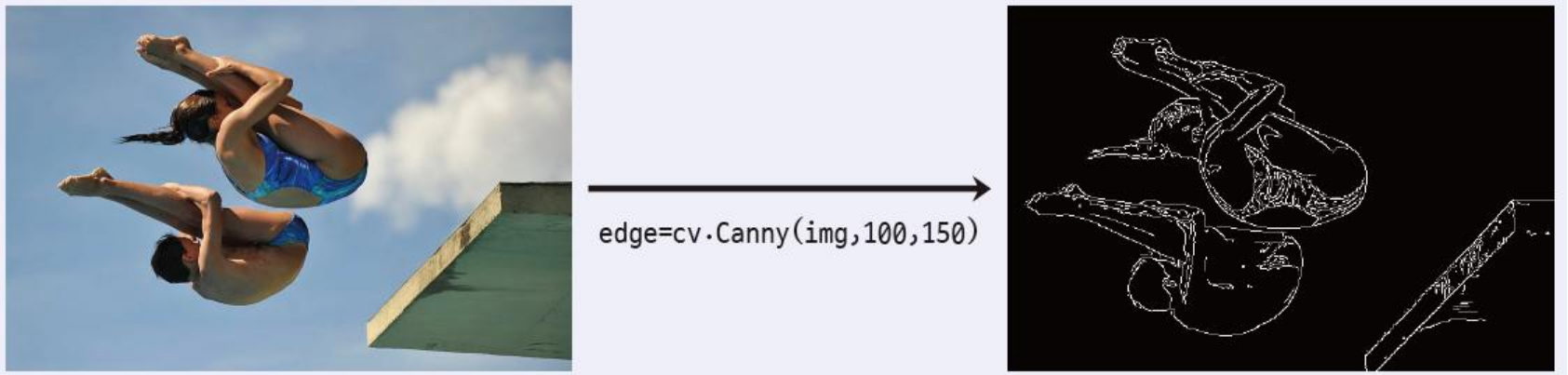


그림 2-1 에지 검출이라는 큰 일을 거뜬히 수행하는 OpenCV의 Canny 함수

- 파이썬 언어와 OpenCV 라이브러리로 컴퓨터 비전 세계 시작!
- 인텔이 만들어 공개한 OpenCV
  - 바퀴를 다시 발명 reinventing the wheel하는 쓸데없는 노력을 방지할 목적
  - 인텔 칩의 성능을 평가할 목적

OpenCV 개요

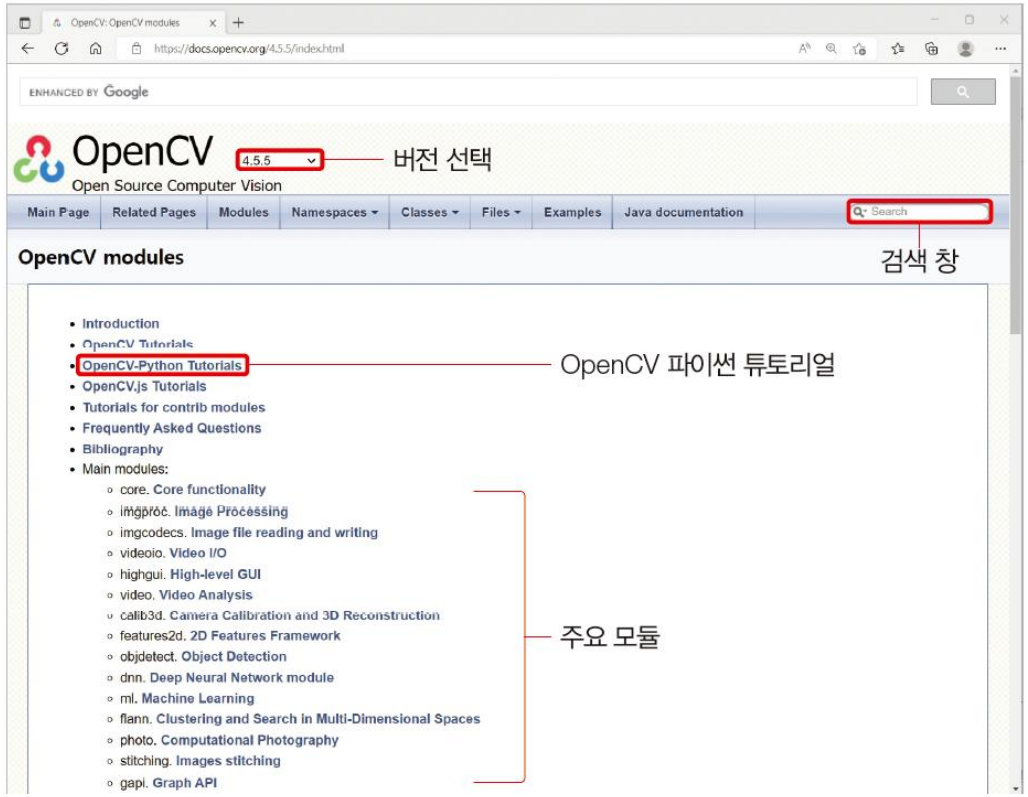
- 개요
  - 클래스와 함수는 C와 C++로 개발. 전체 코드는 180만 라인 이상
  - 인터페이스 언어는 C, C++, 자바, 자바스크립트, 파이썬
  - OS 플랫폼은 윈도우, 리눅스, macOS, 안드로이드, iOS
  - 교차 플랫폼 지원
  - 교육과 상업 목적 모두 무료

OpenCV 역사

연도	사건
1998	• 인텔 직원인 개리 브라드스키(Gary Bradski)가 아이디어 제안
1999	• 오픈 소스로 공개하기로 결정하고 이름을 OpenCV로 정함
2000	• CVPR 컨퍼런스에서 알파 버전 발표
2001-2005	• 5개의 베타 버전 발표
2005	• 스탠퍼드 대학교의 자율주행차인 스탠리의 개발 팀에 합류해 그랜드 챌린지 우승 • OpenCV Korea 출범(https://cafe.naver.com/opencv)
2006	• OpenCV 1.0(C 인터페이스) 공개 • 로고 완성
2009	• OpenCV 2.0(C++ 인터페이스) 공개 • 파이썬과 자바 인터페이스 지원
2012	• 안드로이드와 iOS 지원 시작 • 깃허브로 마이그레이션
2015	• OpenCV 3.0 공개
2016	• 자바스크립트 인터페이스 지원 시작 • 딥러닝을 지원하는 DNN 모듈 추가
2018	• OpenCV 4.0 공개 • 고속 처리를 지원하는 OpenVINO 공개
2020	• Computer Vision and Deep Learning 코스 개설 • 전용 보드인 OpenCV AI Kit 출시
2022	• OpenCV 4.6 공개

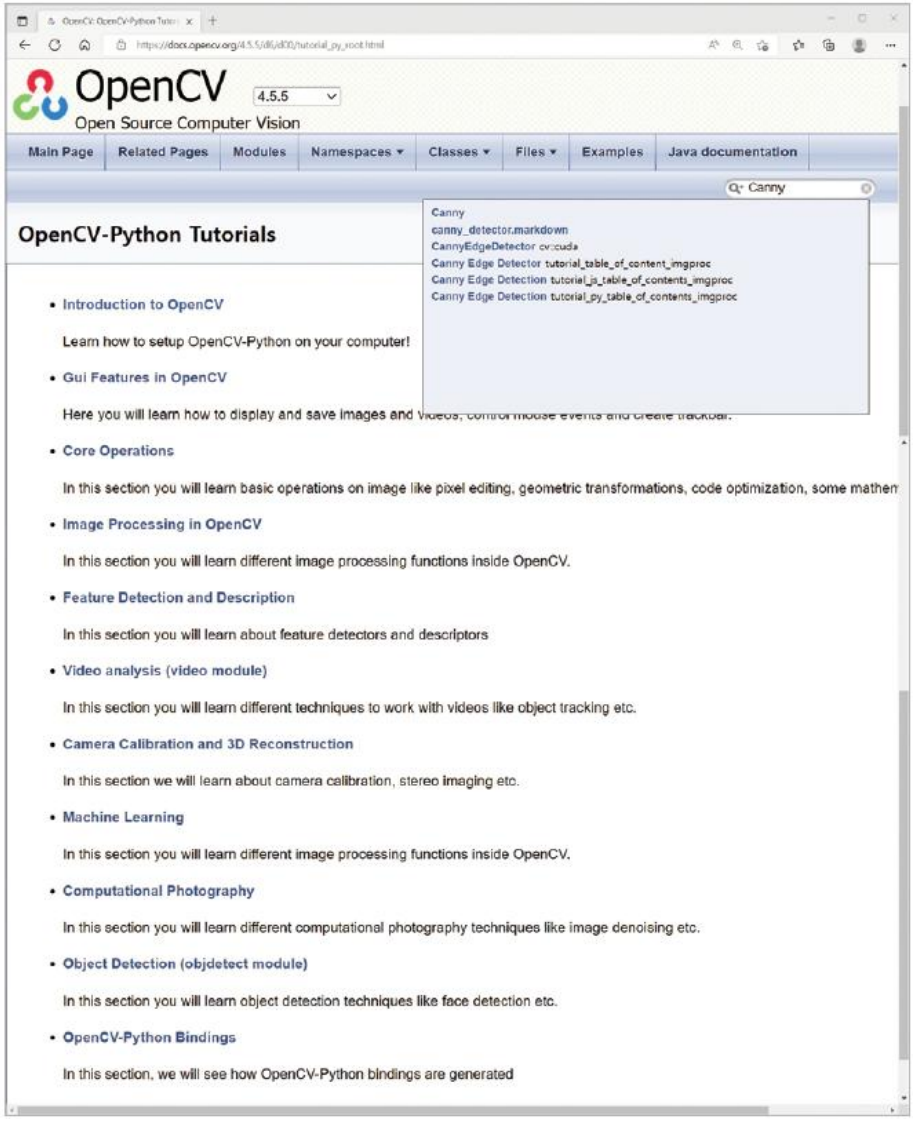
공식 사이트

- OpenCV를 지원하는 사이트
  - 공식 홈페이지(https://opencv.org)
  - 매뉴얼 사이트: 프로그래밍할 때 가장 많은 도움(https://docs.opencv.org)
  - 깃허브
  - 대한민국 OpenCV 사이트(https://cafe.naver.com/opencv)

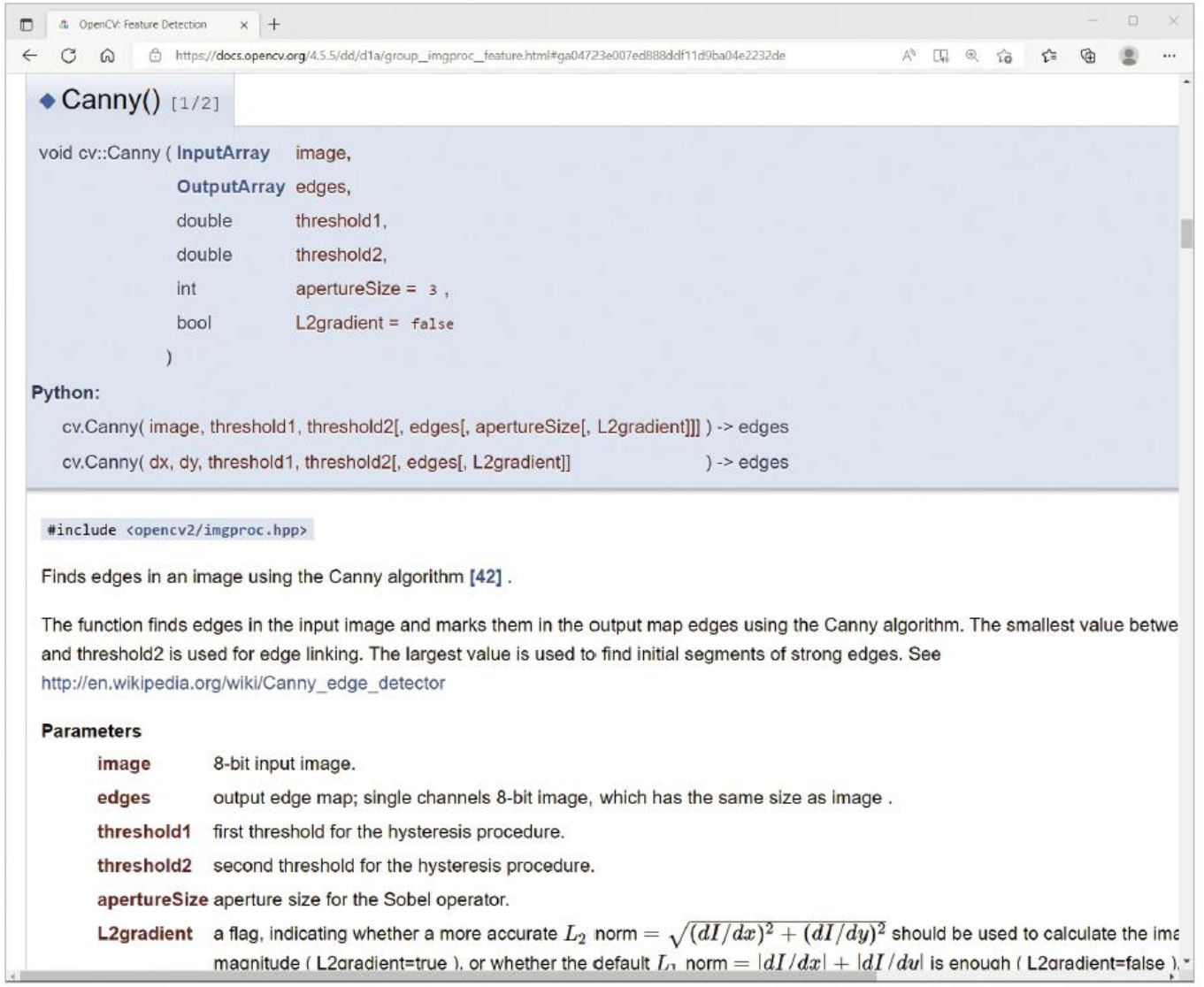


OpenCV 매뉴얼 활용하기

- OpenCV-Python 튜토리얼



- 함수 선언



영상을 읽고 표시하기

In

```
import cv2 as cv
import sys

img=cv.imread('C:/cv_workspace/data/soccer.jpg')     # 영상 읽기

if img is None:
    sys.exit('파일을 찾을 수 없습니다.')

cv.imshow('Image Display',img)  # 윈도우에 영상 표시

cv.waitKey()
cv.destroyAllWindows()
```

Out

OpenCV에서 영상은 numpy.ndarray 클래스 형의 객체

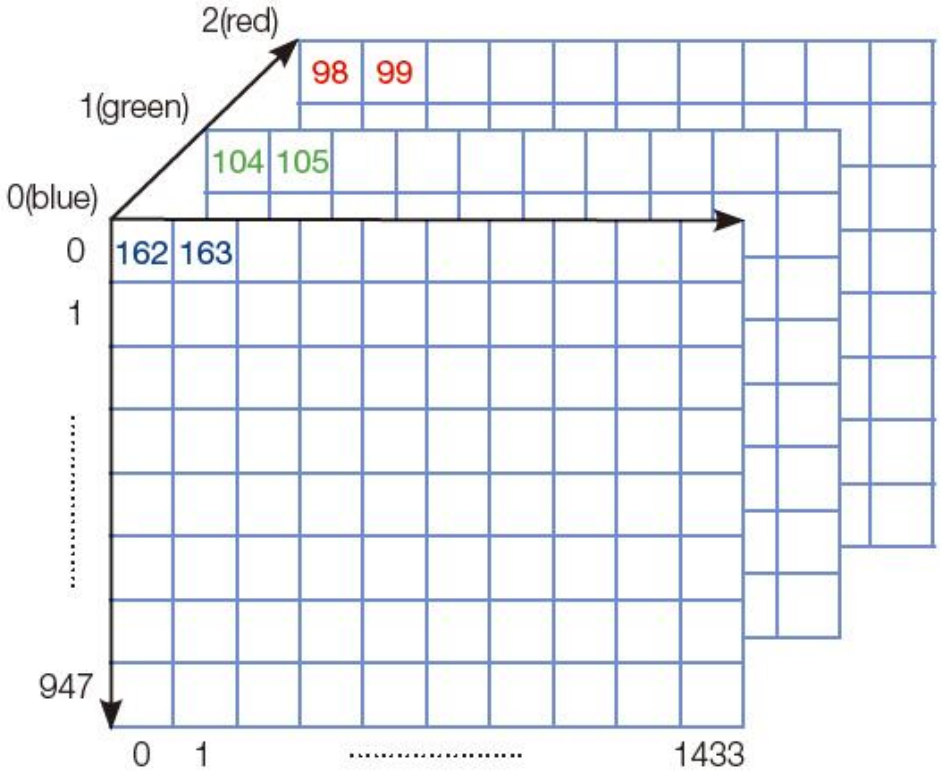
- numpy는 다차원 배열을 위한 사실상 표준 모듈
  - 이런 이유로 OpenCV는 영상을 넘파이 배열numpy.ndarray로 표현
  - OpenCV가 다루는 영상은 numpy가 제공하는 다양한 기능(함수)을 사용할 수 있음

In	type(img)
Out	numpy.ndarray
In	img.shape
Out	(948, 1434, 3)

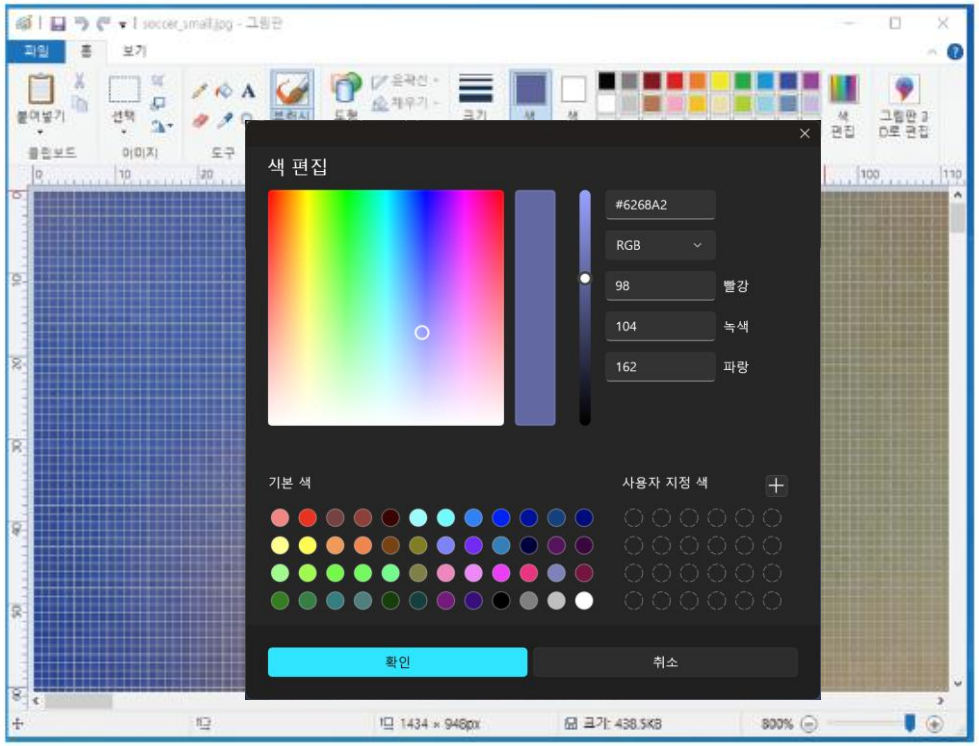
OpenCV에서 영상은 numpy.ndarray 클래스 형의 객체

- 영상의 표현
  - 화소의 위치 (r,c) 또는 (y,x)
  - 화솟값RGB 조사

In	# (0, 0) 화소 조사 print(img[0, 0, 0], img[0, 0, 1], img[0, 0, 2])
Out	162 104 98
In	# (0, 1) 화소 조사 print(img[0, 1, 0], img[0, 1, 1], img[0, 1, 2])
Out	163 105 99



(a) 프로그램으로 조사



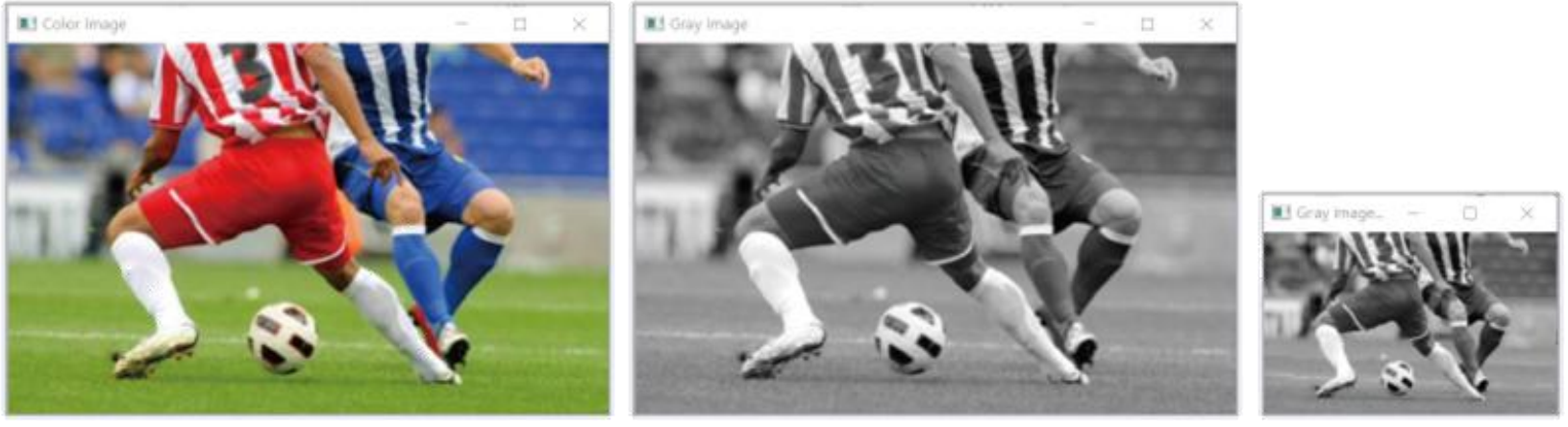
(b) 그림판으로 조사

영상 형태 변환하고 크기 축소하기

In

```
img=cv.imread('C:/cv_workspace/data/soccer.jpg')

if img is None:
    sys.exit('파일을 찾을 수 없습니다.')

gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)    # BGR 컬러 영상을 명암 영상으로 변환
gray_small=cv.resize(gray,dsize=(0,0),fx=0.5,fy=0.5) # 반으로 축소

cv.imwrite('soccer_gray.jpg',gray)         # 영상을 파일에 저장
cv.imwrite('soccer_gray_small.jpg',gray_small)

cv.imshow('Color image',img)
cv.imshow('Gray image',gray)
cv.imshow('Gray image small',gray_small)

cv.waitKey()
cv.destroyAllWindows()
```

Out



cvtColor 함수가 컬러 영상을 명암 영상으로 바꾸는 방법

$$\text{RGB[A] to Gray:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

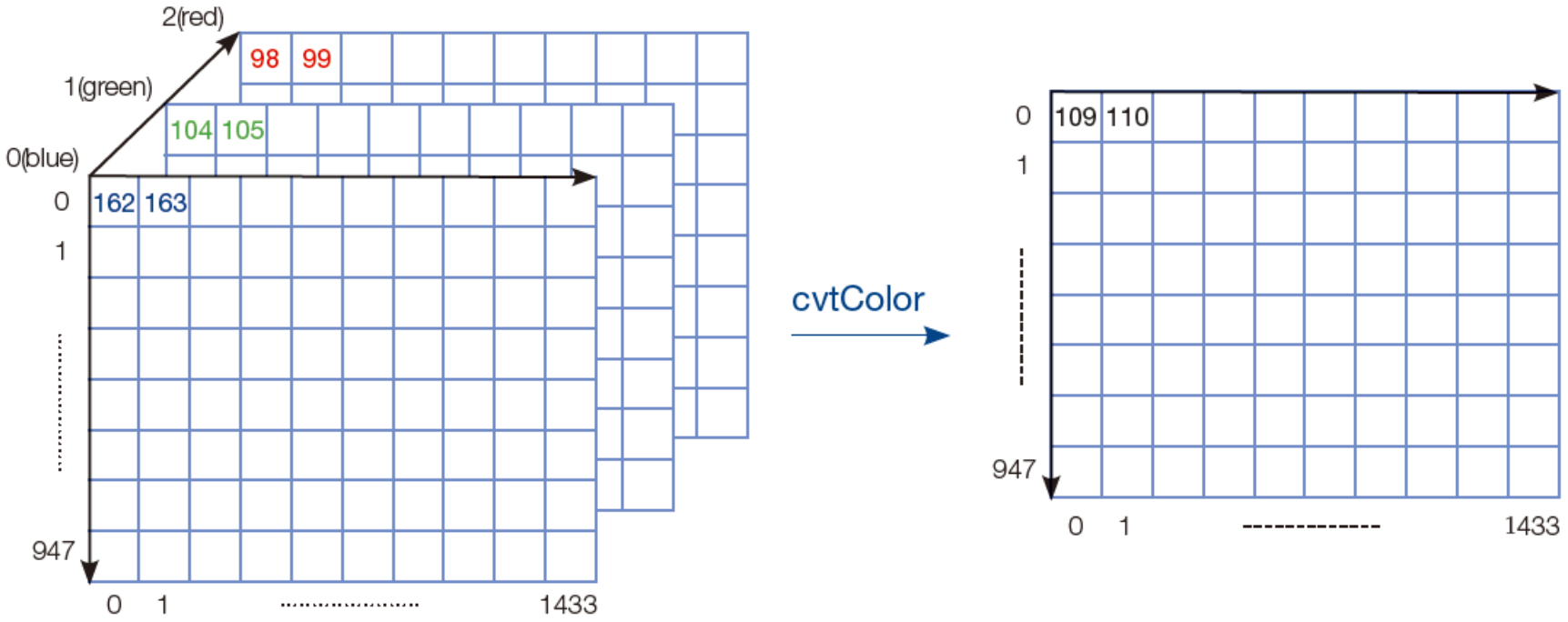


그림 2-10 BGR 컬러 영상을 명암 영상으로 변환

$$\text{Gray to RGB[A]:} \quad R \leftarrow Y, G \leftarrow Y, B \leftarrow Y, A \leftarrow \max(ChannelRange)$$

- 참조 https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

비디오에서 영상 수집하기

- 적어도 3장 이상 'c'버튼 눌러서 캡쳐하기.

In

```
import numpy as np
cap=cv.VideoCapture(0,cv.CAP_DSHOW)        # 카메라와 연결 시도

if not cap.isOpened():
    sys.exit('카메라 연결 실패')

frames=[]
while True:
    ret,frame=cap.read()                              # 비디오를 구성하는 프레임 획득

    if not ret:
        print('프레임 획득에 실패하여 루프를 나갑니다.')
        break

    cv.imshow('Video display',frame)

    key=cv.waitKey(1)# 1밀리초 동안 키보드 입력 기다림
    if key==ord('c'):# 'c' 키가 들어오면 프레임을 리스트에 추가
        frames.append(frame)
    elif key==ord('q'):          # 'q' 키가 들어오면 루프를 빠져나감
        break

cap.release()                                     # 카메라와 연결을 끊음
cv.destroyAllWindows()
```

비디오에서 영상 수집하기

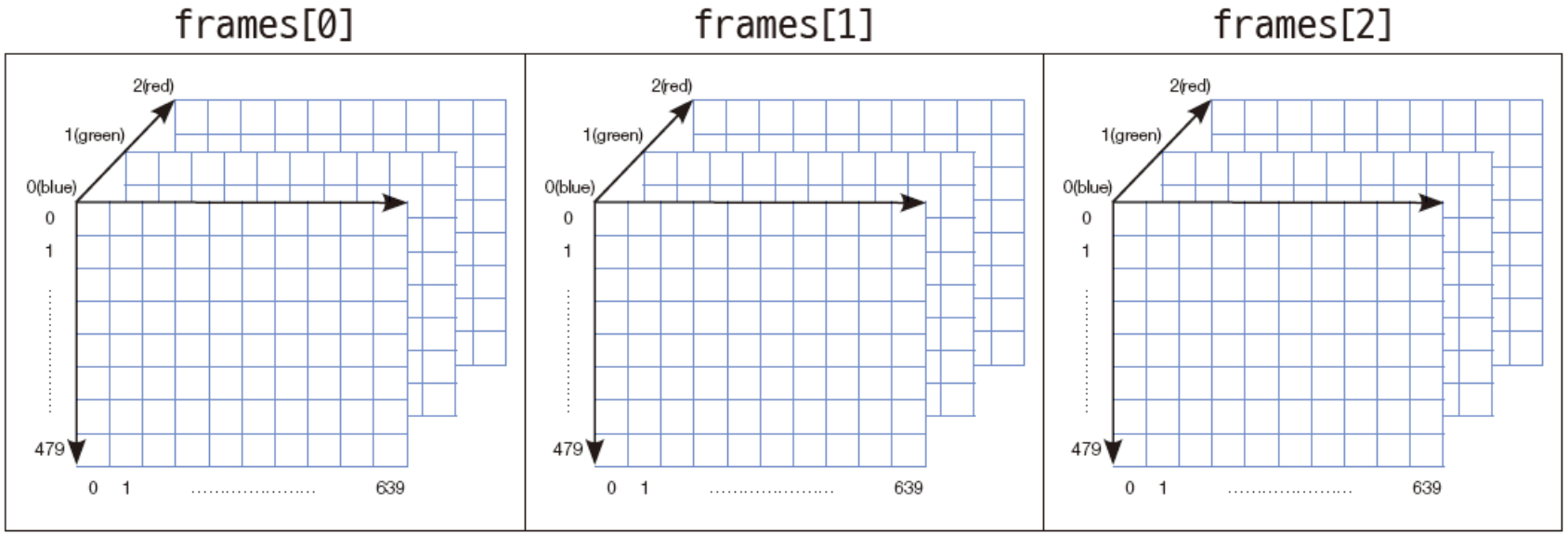
- 저장한 프레임 확인

In

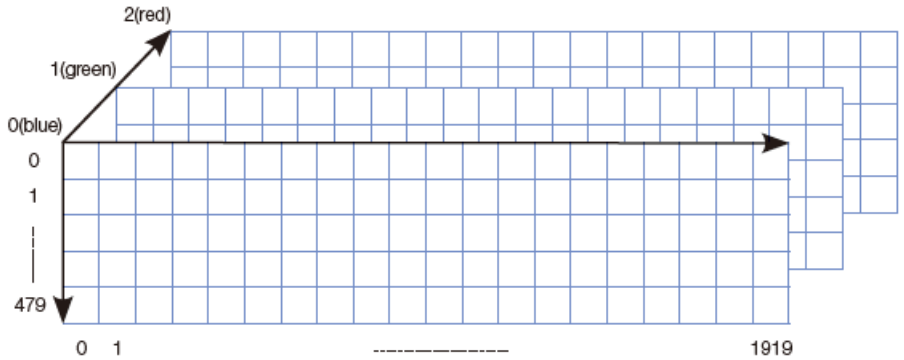
```
if len(frames)>0:                # 수집된 영상이 있으면
    imgs=frames[0]
    for i in range(1,min(3,len(frames))):  # 최대 3개까지 이어 붙임
        imgs=np.hstack((imgs,frames[i]))

    cv.imshow('collected images',imgs)

    cv.waitKey()
    cv.destroyAllWindows()

else:
    print("[WARN] 수집된 프레임이 없습니다.")
```

Out

영상의 자료구조



(a) frames 리스트



(b) imgs 배열

- 저장된 프레임 개수

In	len(frames)
Out	10

- 첫 번째 프레임의 형태 확인

In	frames[0].shape
Out	(480, 640, 3)

- imgs의 타입 확인

In	type(imgs)
Out	numpy.ndarray

- imgs의 형태 확인

In	type(imgs)
Out	numpy.ndarray

OpenCV의 그래픽 기능

- 영상에 글씨나 도형을 넣는데 유용
  - line, rectangle, polylines, circle, ellipse, putText 함수
- 영상에 도형을 그리고 글씨 쓰기

In

```
img=cv.imread('C:/cv_workspace/data/girl_laughing.jpg')

if img is None:
    sys.exit('파일을 찾을 수 없습니다.')

cv.rectangle(img,(830,30),(1000,200),(0,0,255),2)# 직사각형 그리기
cv.putText(img,'laugh',(830,24),cv.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2)# 글씨 쓰기

cv.imshow('Draw',img)

cv.waitKey()
cv.destroyAllWindows()
```

OpenCV의 그래픽 기능

```
img=cv.imread('girl_laughing.jpg')

if img is None:
    sys.exit('파일을 찾을 수 없습니다.')

cv.rectangle(img,(830,30),(1000,200),(0,0,255),2)   # 직사각형 그리기
cv.putText(img,'laugh',(830,24),cv.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2) # 글씨 쓰기

cv.imshow('Draw',img)

cv.waitKey()
cv.destroyAllWindows()
```

red

굵기

blue

영상 처리

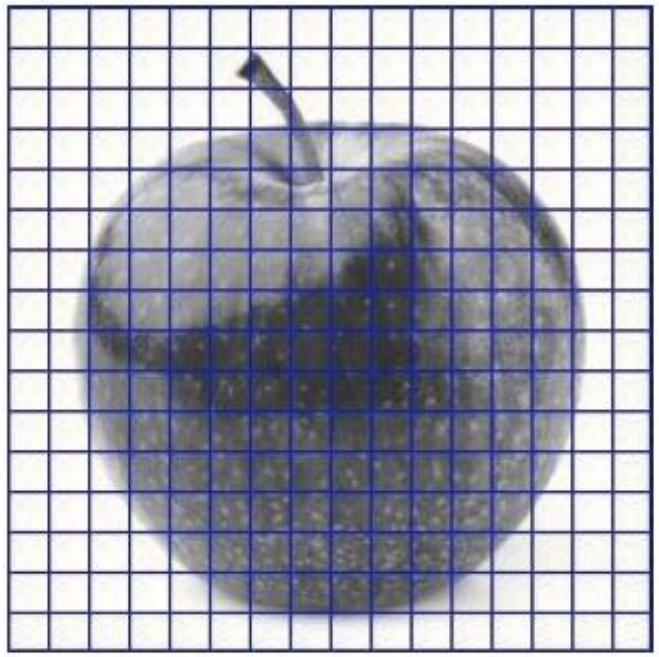
- 영상 처리: 특정 목적을 달성하기 위해 원래 영상을 개선된 새로운 영상으로 변환하는 작업

(a) 안개 낀 도로 영상 (b) 히스토그램 평활화로 개선한 영상

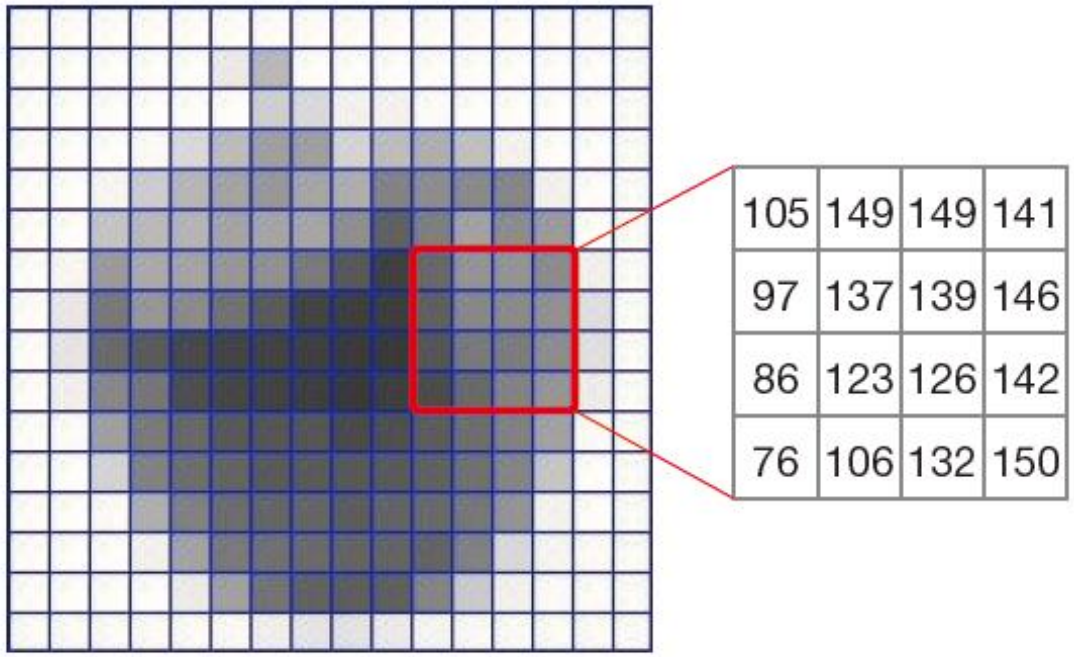
- 화질 개선 자체가 목적인 경우
  - 예) 도주 차량의 번호판 식별. 병변 위치 찾기 등
- 컴퓨터 비전은 전처리로 활용하여 인식 성능을 향상
- 현대는 인터넷에 수많은 영상이 쌓임
  - 컴퓨터 비전 알고리즘을 개발하는데 중요한 실험 데이터로 활용됨

영상 획득과 표현

- 디지털 변환
  - M*N 영상으로 샘플링sampling
  - L 단계로 양자화quantization



(a) 샘플링

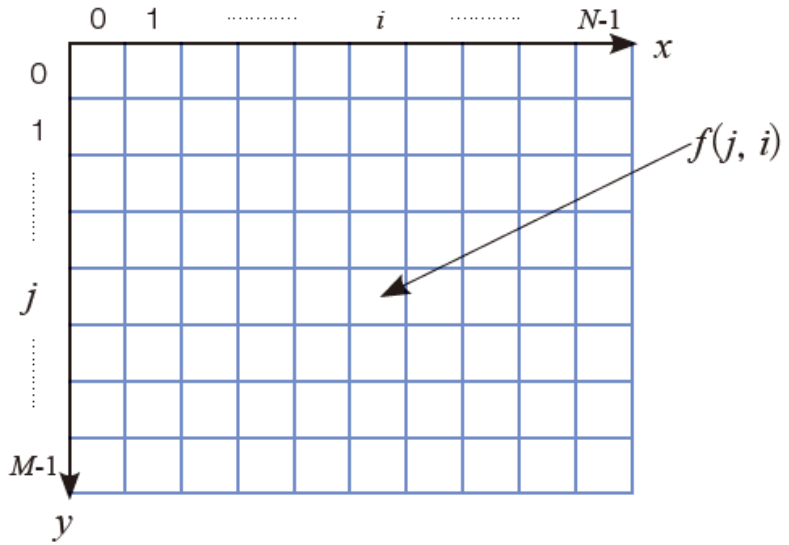


(b) 양자화

TIP 엄밀히 말해 해상도는 물리적 단위 공간에서 식별 가능한 점의 개수를 뜻한다. 예를 들어 인치 당 점의 개수를 뜻하는 dpi(dot per inch)는 해상도다. 이 책에서는 화소의 개수를 해상도라고 부른다.

영상 획득과 표현

- 영상 좌표계
  - 왼쪽 위 구석이 원점
  - (y,x) 표기

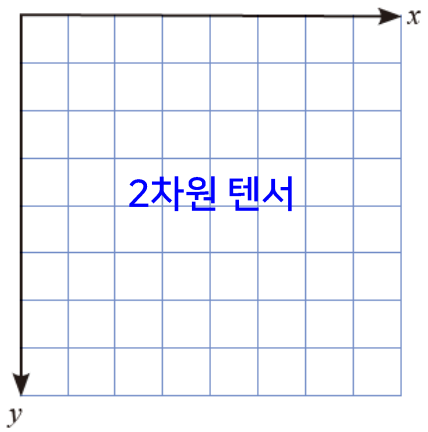



```
cv.line(img,(10,20),(100,20),….)
```

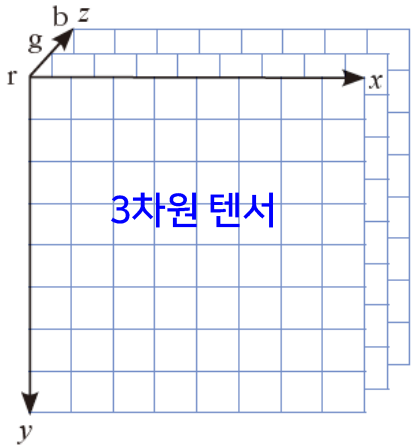
  - 함수에 따라 (x,y) 표기 사용하니 주의할 필요.
  - 예) cv.line 함수

- OpenCV는 numpy.ndarray로 영상 표현
  - numpy.ndarray가 지원하는 다양한 함수를 사용할 수 있다는 큰 장점
  - 예) min, max, argmin, argmax, mean, sort, reshape, transpose, … …

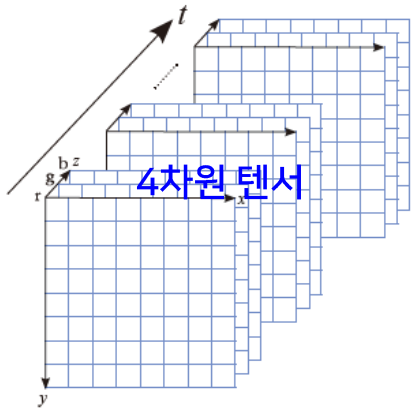
다양한 종류의 영상



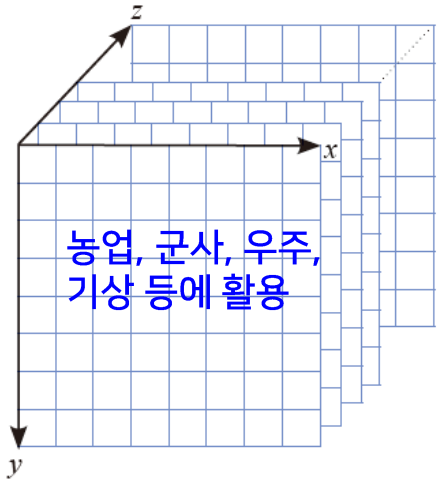
(a) 명암 영상



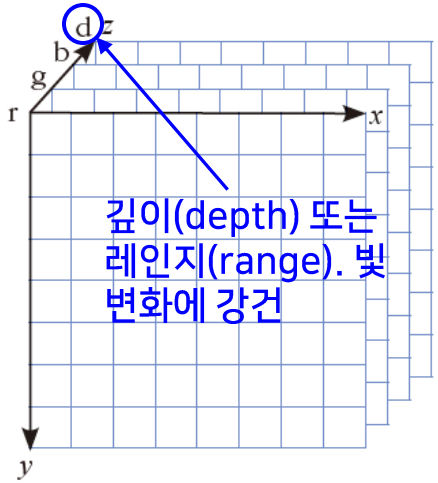
(b) 컬러 영상



(c) 컬러 동영상



(d) 다분광/초분광/MR/CT 영상

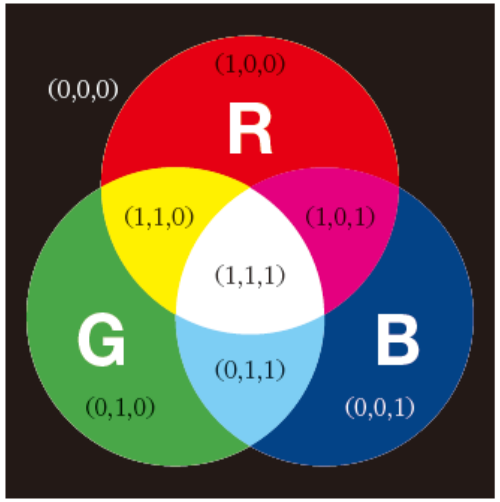


(e) RGB-D 영상

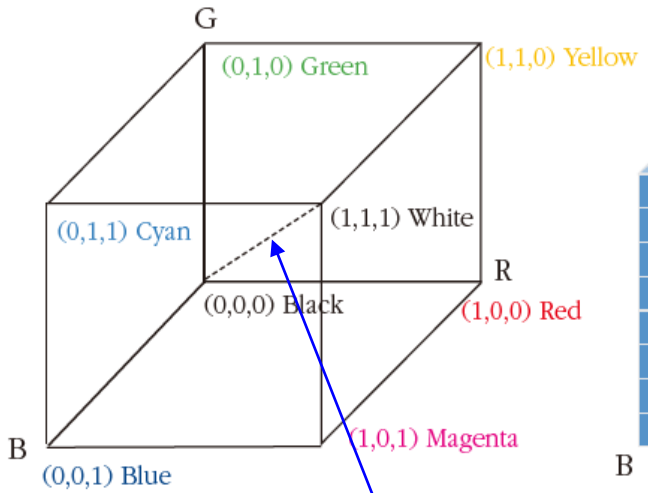
(f) 점 구름 영상

컬러 모델

- RGB 컬러 모델

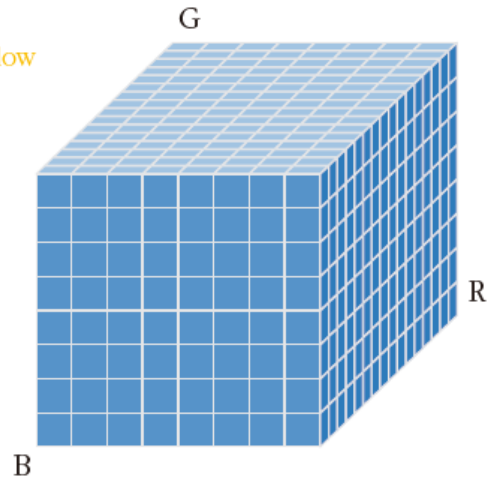


(a) RGB 삼원색의 혼합



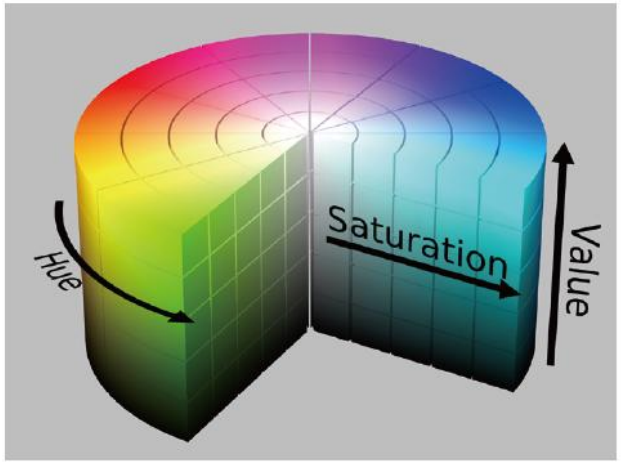
(b) RGB 큐브

명암(grayscale)



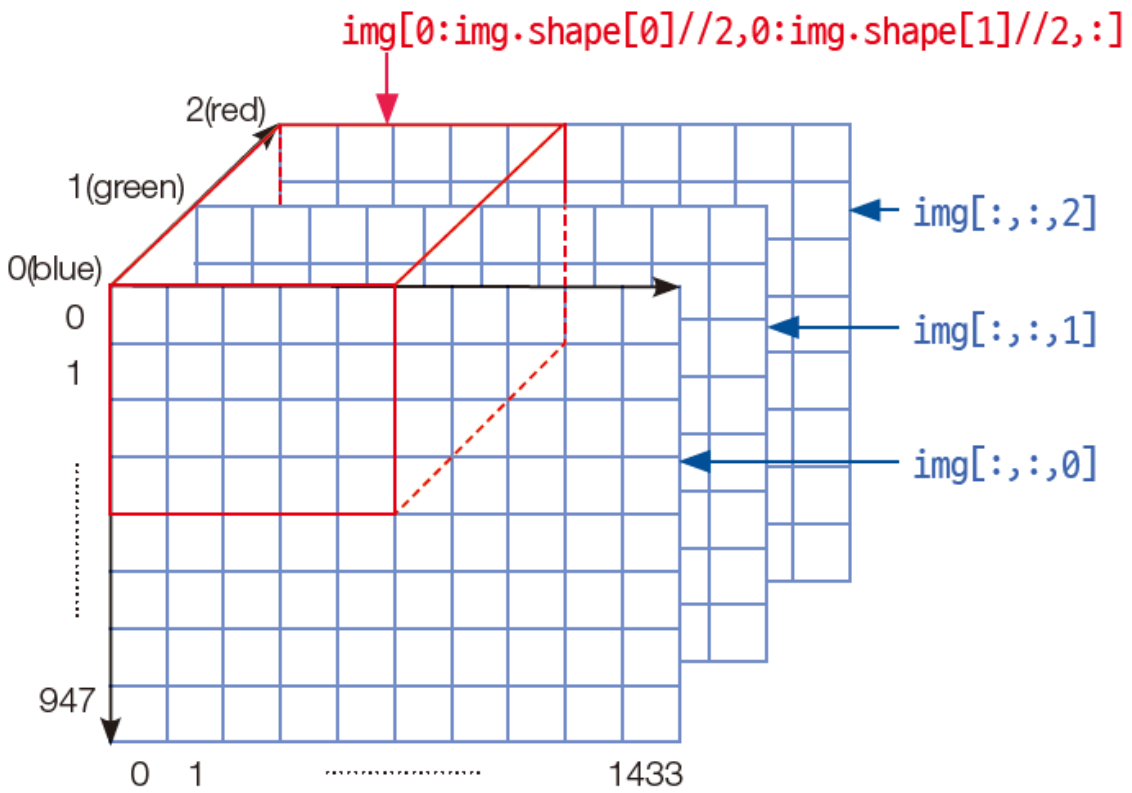
(c) 양자화된 RGB 큐브

- HSV 컬러 모델
  - 빛의 밝기가 V 요소에 집중
  - RGB보다 빛 변환에 강건



RGB 채널별로 디스플레이

- numpy의 슬라이싱 기능을 이용하여 RGB 채널별로 디스플레이




```
In [0]: img.shape
(948,1434,3)
```

RGB 채널별로 디스플레이

In

```
img=cv.imread('C:/cv_workspace/data/soccer.jpg')
if img is None:
    sys.exit('파일을 찾을 수 없습니다.')

cv.imshow('original_RGB',img)
cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,img.shape[1]//4:3*img.shape[1]//4,:])

cv.imshow('R channel',img[:,:,2])
cv.imshow('G channel',img[:,:,1])
cv.imshow('B channel',img[:,:,0])

cv.waitKey()
cv.destroyAllWindows()
```

히스토그램 계산

- 히스토그램
  - [0,L-1] 사이의 명암값 각각이 영상에 몇 번 나타나는지 표시
  - 히스토그램 h와 정규화 히스토그램

$$h(l) = |\{(j,i) \mid f(j,i) = l\}| \tag{2.1}$$

$$\hat{h}(l) = \frac{h(l)}{M \times N} \tag{2.2}$$

알고리즘 2-1 명암 영상에서 히스토그램 계산

입력 : 명암 영상 $f(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

출력 : 히스토그램 $h(l)$과 정규 히스토그램 $\hat{h}(l), 0 \le l \le L-1$

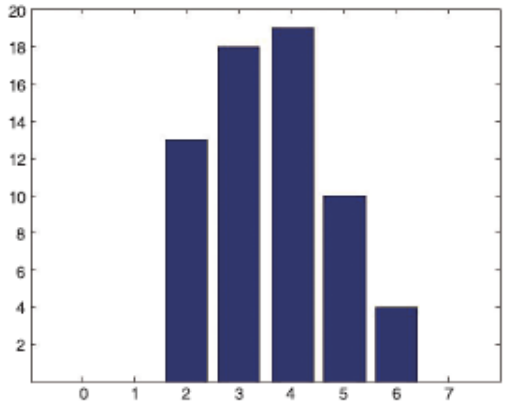
```
1  for(l=0 to L-1)    h(l)=0; // 초기화
2  for(j=0 to M-1)
3    for(i=0 to N-1)  // f의 화소 (j,i) 각각에 대해
4      h(f(j,i))++; // 그곳 명암값에 해당하는 히스토그램 칸을 1만큼 증가
5  for(l=0 to L-1)
6    ĥ(l)=h(l)/(M×N); // 정규화한다.
```

히스토그램 계산

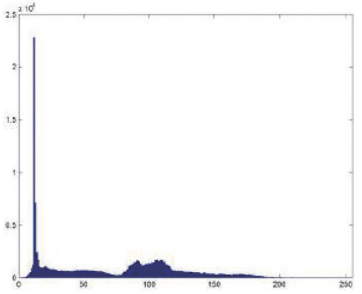
[그림 2-7(a)]는 M과 N이 8이고 L=8인 아주 작은 영상이다. 이 영상에서 명암값이 2인 화소는 13개이므로 $h(2)=13$이다. 다른 명암값에 대해서도 화소의 개수를 세어보면 $h=(0,0,13,18,19,10,4,0)$이고, $\hat{h}(l)=(0,0,0.203,0.281,0.297,0.156,0.063,0)$이다. 이것을 그래프로 그리면 [그림 2-7(b)]와 같다.

3	2	2	2	2	3	3	4
3	2	2	2	3	4	3	3
4	3	3	4	4	4	3	3
5	4	4	4	5	4	3	3
5	4	3	4	5	4	3	2
6	5	4	4	5	4	3	2
6	6	5	5	4	3	2	2
6	5	4	5	4	3	2	2

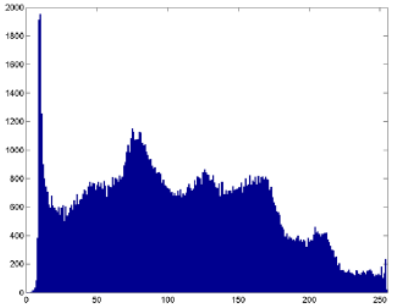
(a) 8×8 영상(8 명암 단계)



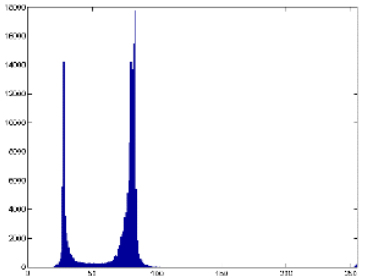
(b) 히스토그램



(a) 어두운 영상

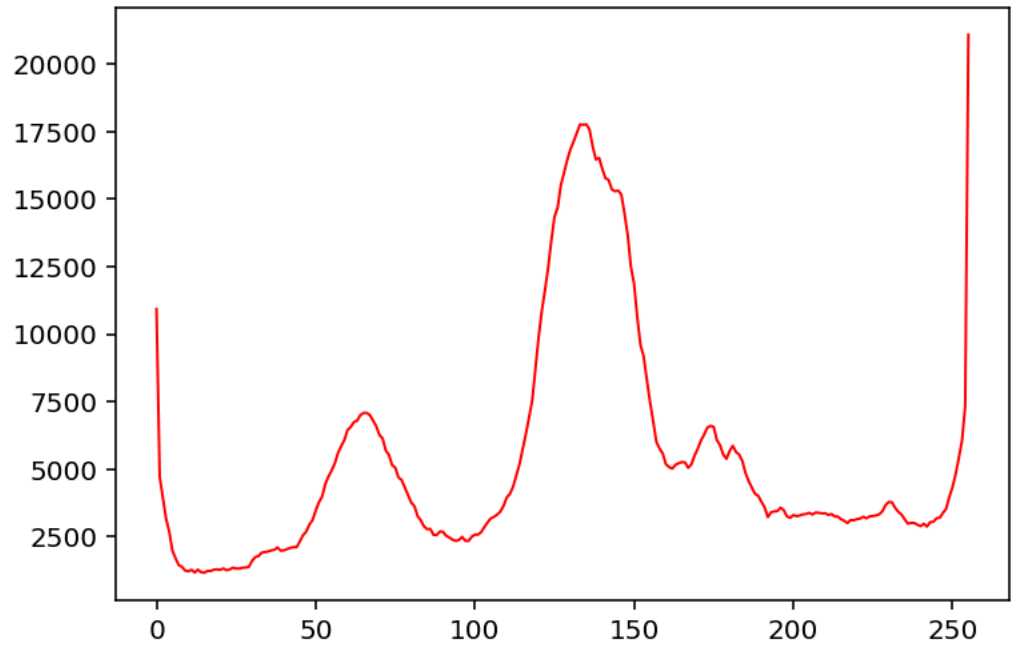


(b) 비교적 균등한 영상



(c) 봉우리 사이의 계곡이 선명한 영상

히스토그램 계산 코드

In	img=cv.imread('C:/cv_workspace/data/soccer.jpg') h=cv.calcHist([img],[2],None,[256],[0,256]) # 2번 채널인 R 채널에서 히스토그램 구함 plt.plot(h,color='r',linewidth=1)
Out	

히스토그램 용도

- 히스토그램 평활화
  - 히스토그램을 평평하게 만들어 주는 연산
  - 명암의 동적 범위를 확장하여 영상의 품질을 향상시켜줌
  - 누적 히스토그램 $c(.)$를 매핑 함수로 사용

$$l_{out} = T(l_{in}) = round(c(l_{in}) \times (L-1)) \quad (2.3)$$

$$\text{이때 } c(l_{in}) = \sum_{l=0}^{l_{in}} \hat{h}(l)$$

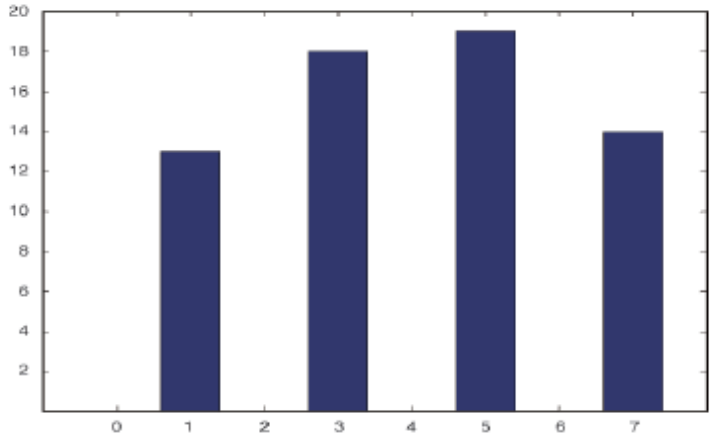
[그림 2-9(a)]에 제시된 표는 매핑 함수 $T(.)$를 구하는 과정을 보여준다. 결국 입력 영상의 명암값 0은 0, 1은 0, 2는 1, 3은 3, …, 7은 7로 매핑해 주는 함수를 얻었다. [그림 2-9(b)]는 매핑하여 얻은 평활화된 영상이다. [그림 2-9(c)]는 새로운 영상의 히스토그램이다. 이 히스토그램을 이전 영상의 히스토그램인 [그림 2-7(b)]와 비교해 보자. 이전 것은 동적 범위가 [2, 6]이었는데 새로운 영상은 [1, 7]로 보다 넓어졌음을 알 수 있다.

l_{in}	$\hat{h}(l_{in})$	$c(l_{in})$	$c(l_{in}) \times 7$	l_{out}
0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0
2	0.203	0.203	1.421	1
3	0.281	0.484	3.388	3
4	0.297	0.781	5.467	5
5	0.156	0.937	6.559	7
6	0.063	1.0	7.0	7
7	0.0	1.0	7.0	7

(a) 매핑 표 $T(.)$

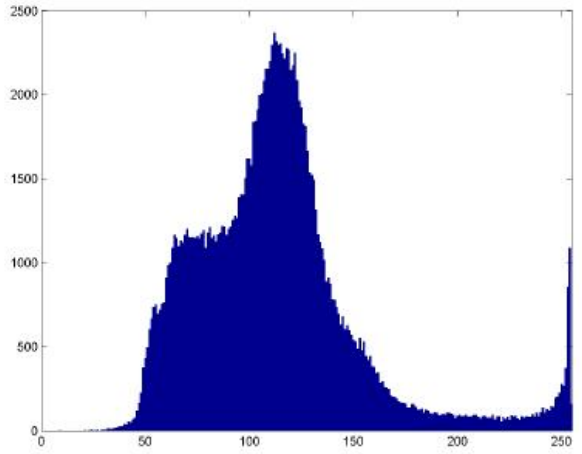
3	1	1	1	1	3	3	5
3	1	1	1	3	5	3	3
5	3	3	5	5	5	3	3
7	5	5	5	7	5	3	3
7	5	3	5	7	5	3	1
7	7	5	5	7	5	3	1
7	7	7	7	5	3	1	1
7	7	5	7	5	3	1	1

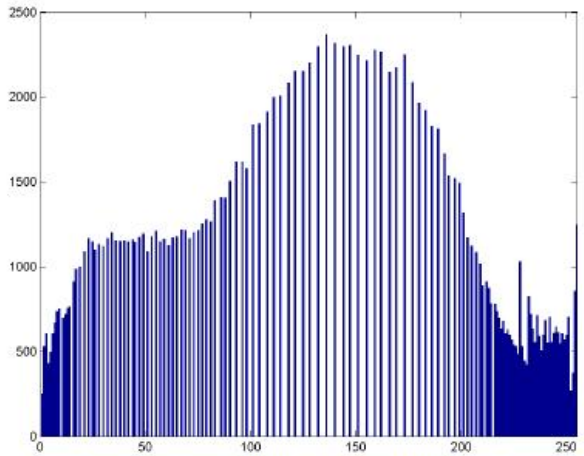
(b) 평활화된 영상



(c) 평활화된 영상의 히스토그램

히스토그램 용도





(a) 원래 영상 (b) 히스토그램 평활화된 영상

그림 2-10 히스토그램 평활화를 적용해 품질이 향상된 예

(a) 원래 영상 (b) 히스토그램 평활화된 영상

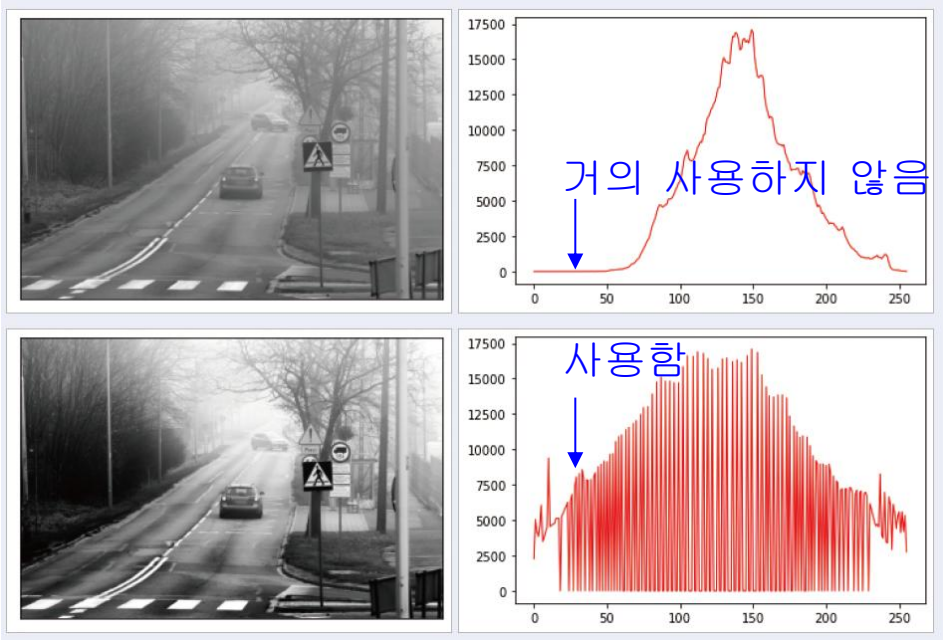
그림 2-11 히스토그램 평활화를 적용해 시각적 느낌이 나빠진 예

히스토그램 평활화

In

```
img=cv.imread('C:/cv_workspace/data/mistyroad.jpg')

gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)                # 명암 영상으로 변환하고 출력
plt.imshow(gray,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()

h=cv.calcHist([gray],[0],None,[256],[0,256])           # 히스토그램을 구해 출력
plt.plot(h,color='r',linewidth=1), plt.show()

equal=cv.equalizeHist(gray)                            # 히스토그램을 평활화하고 출력
plt.imshow(equal,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()

h=cv.calcHist([equal],[0],None,[256],[0,256])          # 히스토그램을 구해 출력
plt.plot(h,color='r',linewidth=1), plt.show()
```

Out

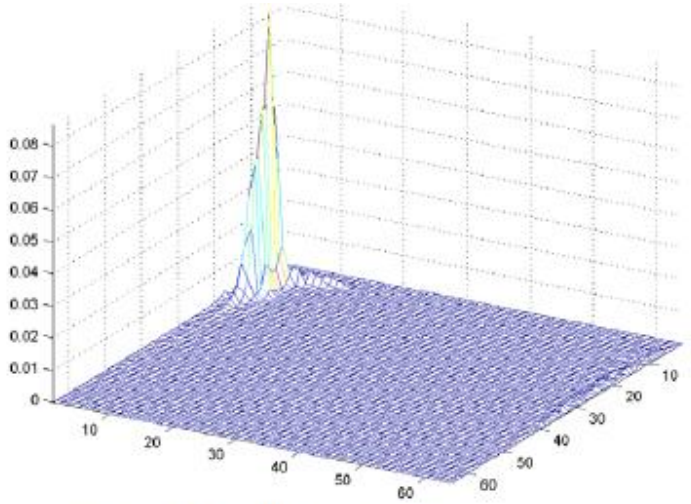


히스토그램 역투영과 얼굴 검출

- 히스토그램 역투영
  - 히스토그램을 매핑 함수로 사용하여, 화소 값을 신뢰도 값으로 변환

- 얼굴 검출 예: 모델 얼굴과 2차원 히스토그램

(a) 모델 얼굴



(b) 2차원 히스토그램(HS 공간)

그림 2-12 얼굴 검출을 위한 모델 얼굴과 히스토그램

알고리즘 2-2 2차원 히스토그램 계산(HS 공간)

입력 : H와 S채널 영상 $f_H(j,i), f_S(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

출력 : 히스토그램 $h(j,i)$와 정규 히스토그램 $\hat{h}(j,i), 0 \le j,i \le q-1$ // L단계를 q단계로 양자화

```
1  h(j,i), 0≤j,i≤q−1을 0으로 초기화한다.
2  for(j=0 to M−1)
3    for(i=0 to N−0)  // 화소 (j,i) 각각에 대해
4      h(quantize(f_H(j,i)), quantize(f_S(j,i)))++;  // 해당 칸을 1 증가시킴
5  for(j=0 to q−1)
6    for(i=0 to q−1)
7      ĥ(j,i)=h(j,i)/(M×N);  // 정규화
```

히스토그램 역투영과 얼굴 검출

- 얼굴 검출
  - 모델 얼굴에서 구한 히스토그램 h_m은 화소의 컬러 값을 얼굴에 해당하는 신뢰도 값으로 변환해줌
  - 실제로는 비율 히스토그램 h_r을 사용

$$h_r(j,i) = \min\left(\frac{\hat{h}_m(j,i)}{\hat{h}_i(j,i)}, 1.0\right), \quad 0 \le j, \quad i \le q-1 \qquad (2.4)$$

알고리즘 2-3 히스토그램 역투영

입력 : H와 S채널 영상 $g_H(j,i), g_S(j,i), 0 \le j \le M-1, 0 \le i \le N-1$ // 얼굴을 검출하려는 영상
모델 히스토그램 $\hat{h}_m(j,i), 0 \le j, i \le q-1$

출력 : 가능성 맵 $o(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

```
1  영상 g_H, g_S에 [알고리즘 2-2]를 적용하여 정규 히스토그램 ĥ_i를 만든다.
2  식 (2.4)를 이용하여 ĥ_r을 구한다.
3  for (j=0 to M-1)
4    for (i=0 to N-1)
5      o(j,i)=ĥ_r(quantize(g_H(j,i)), quantize(g_S(j,i))); // 역투영
```

히스토그램 역투영과 얼굴 검출

- 히스토그램 역투영 결과
  - 얼굴 영역은 높은 신뢰도 값, 손 영역도 높은 값
  - 한계: 비슷한 색 분포를 갖는 다른 물체 구별 못함. 검출 대상이 여러 색 분포를 갖는 경우 오류 가능성
  - 장점: 배경을 조정할 수 있는 상황에 적합 (이동과 회전에 불변, 가림occlusion에 강인)

(a) 입력 영상 (b) 역투영 영상

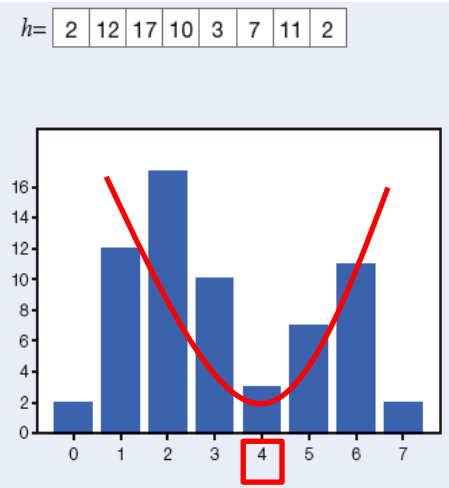
그림 2-13 히스토그램 역투영을 이용한 얼굴 검출

이진 영상

- 화소가 0(흑) 또는 1(백)인 영상
- 1비트면 저장할 수 있는데, 편의상 1바이트 사용하는 경우 많음
- 에지 검출 결과를 표시하거나 물체와 배경을 구분하여 표시하는 응용 등에 사용
- 알고리즘
  - 임계값 T보다 큰 화소는 1, 그렇지 않은 화소는 0으로 바꿈. 임계값 결정이 중요 $b(j,i)=\begin{cases}1, f(j,i)\geq T\\0, f(j,i)<T\end{cases}$
  - 히스토그램에서 계곡 부근으로 결정하는 방법

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

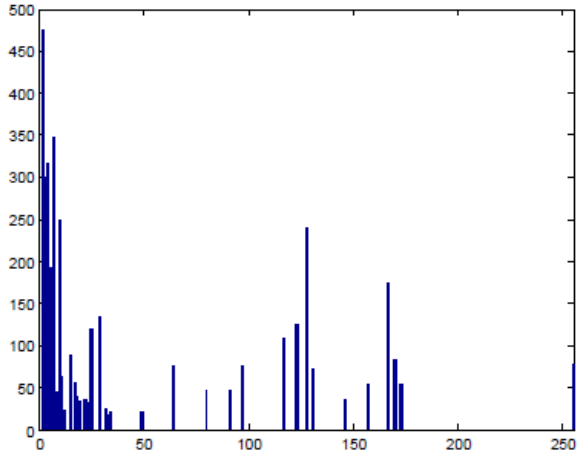
(a) 입력 영상



(b) 히스토그램

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

(c) 이진 영상



(a) 히스토그램

(b) 임계값을 이용하여 구한 이진 영상(T=50)

* 실제 영상에서는 계곡이 아주 많이 나타나서 구현이 쉽지 않음

오츄 알고리즘

- 이진화를 최적화 문제로 바라봄. 최적값 $\hat{t}$을 임계값 T로 이용

$$\hat{t} = \underset{t\in\{0,1,2,\cdots,L-1\}}{\operatorname{argmin}} J(t) \qquad (3.2)$$

  - 목적 함수 $J(t)$는 임계값 t의 좋은 정도를 측정함(작을수록 좋음)
    - t로 이진화했을 때 0이 되는 화소들의 분산($v_0(t)$)과 1이 되는 화소들의 분산($v_1(t)$)의 가중치($n_0(t)$와 $n_1(t)$) 합을 J로 사용

$$J(t) = w_0(t)v_0(t) + w_1(t)v_1(t)$$

$$w_0(t) = \sum_{i=0}^{t}\hat{h}(i), \qquad w_1(t) = \sum_{i=t+1}^{L-1}\hat{h}(i)$$

- 이진화 했을 때 흑 그룹과 백 그룹 각각이 균일할수록 좋다는 원리에 근거
- 균일성은 분산으로 측정 (분산이 작을수록 균일성 높음)
- 분산의 가중치 합 *J(t)*를 목적 함수로 이용한 최적화 알고리즘

오츄 알고리즘 코드

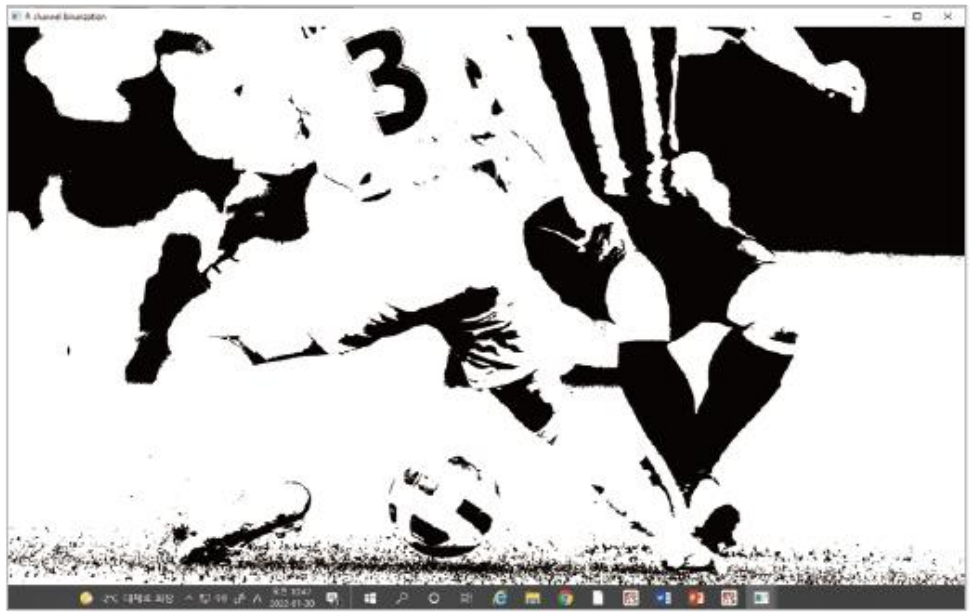
- 프로그래밍 실습

In

```
img=cv.imread('C:/cv_workspace/data/soccer.jpg')

t,bin_img=cv.threshold(img[:,:,2],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print('오츄 알고리즘이 찾은 최적 임곗값=',t)

cv.imshow('R channel',img[:,:,2])                              # R 채널 영상
cv.imshow('R channel binarization',bin_img)          # R 채널 이진화 영상

cv.waitKey()
cv.destroyAllWindows()
```

Out

```
오츄 알고리즘이 찾은 최적 임곗값= 113.0 ①
```



오츄 알고리즘

최적화 문제를 푸는 알고리즘

- 오츄는 최적화를 구현하는데 낱낱 탐색$_{\text{exhaustive search}}$ 알고리즘 사용
  - 매개변수 t가 해공간을 구성하는데, 해공간이 작아 낱낱 탐색 가능
  - L이 256이라면 해공간은 {0,1,2,3,...,255}
- 컴퓨터 비전은 문제를 최적화로 푸는 경우가 많음. 해공간이 커서 낱낱 탐색 불가능하여 부최적해$_{\text{sub-optimal solution}}$를 찾는 효율적인 알고리즘 사용
  - 스네이크 (물체 외곽선을 찾는 알고리즘. 4.5.1절)는 탐욕 알고리즘 사용
  - 역전파 알고리즘(기계 학습을 위해 미분하는 알고리즘. 7.6절)은 미분 사용

연결 요소

4-연결성과 8-연결성

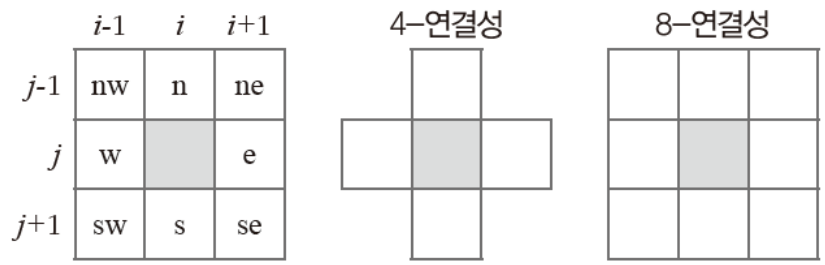


그림 3-10 화소의 연결성

연결 요소? 같은 덩어리

[예시 3-2] 연결 요소

[그림 3-11]에서 (a)는 입력 이진 영상이고, (b)와 (c)는 각각 4-연결성과 8-연결성으로 찾은 연결 요소다. 연결 요소는 고유한 정수 번호로 구분한다.

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

(a) 입력 이진 영상

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	0	2	2	0	0
0	1	1	0	2	2	0	0
0	0	0	0	2	2	0	0
0	0	0	0	0	0	0	0
0	0	3	3	3	0	4	0
0	0	0	3	3	0	4	0

(b) 4-연결성으로 찾은 연결 요소

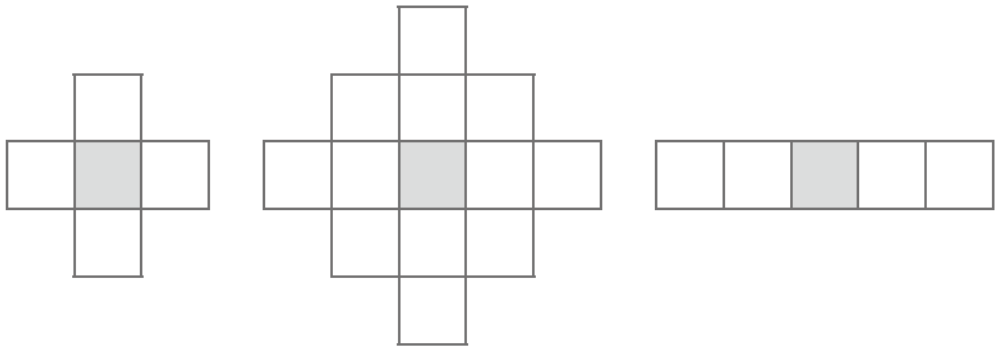
0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	0	1	1	0	0
0	1	1	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	2	2	2	0	3	0
0	0	0	2	2	0	3	0

(c) 8-연결성으로 찾은 연결 요소

그림 3-11 연결 요소 찾기

모폴로지

- 모폴로지는 구조 요소structuring element를 이용하여 영역의 모양을 조작



  - 팽창:작은 홈을 메우거나 끊어진 영역을 연결하는 효과. 영역을 키움
  - 침식:경계에 솟은 돌출 부분을 깎는 효과. 영역을 작게 만듦
  - 열림:침식한 결과에 팽창 적용. 원래 영역 크기 유지
  - 닫힘:팽창한 결과에 침식을 적용. 원래 영역 크기 유지

모폴로지

팽창과 침식 연산

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

(a) 입력 영상과 구조 요소

0	0	0	0	0	0	0	0
0	0	0	P 1	0	0	0	0
0	1	1	1	1	1	0	0
0	1	1	0	1	1	1	0

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

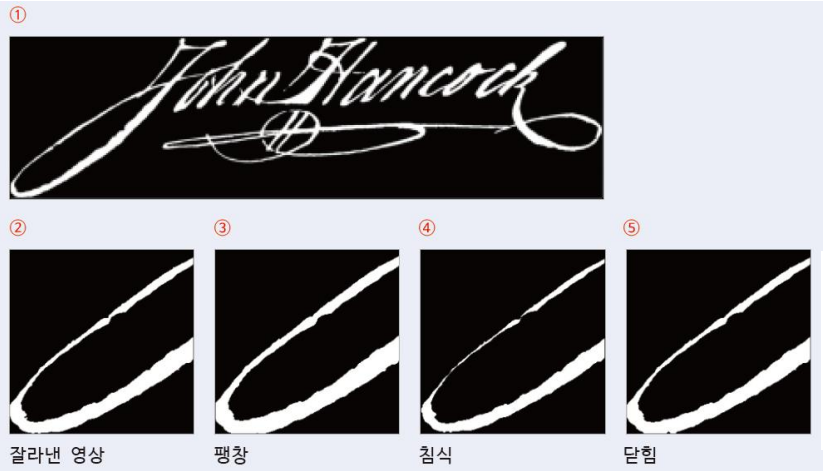
(b) 팽창

0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	P 1	1	1	1	1	0	0
0	1	1	0	1	1	1	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	0	0	0	1	0	0

(c) 침식

모폴로지



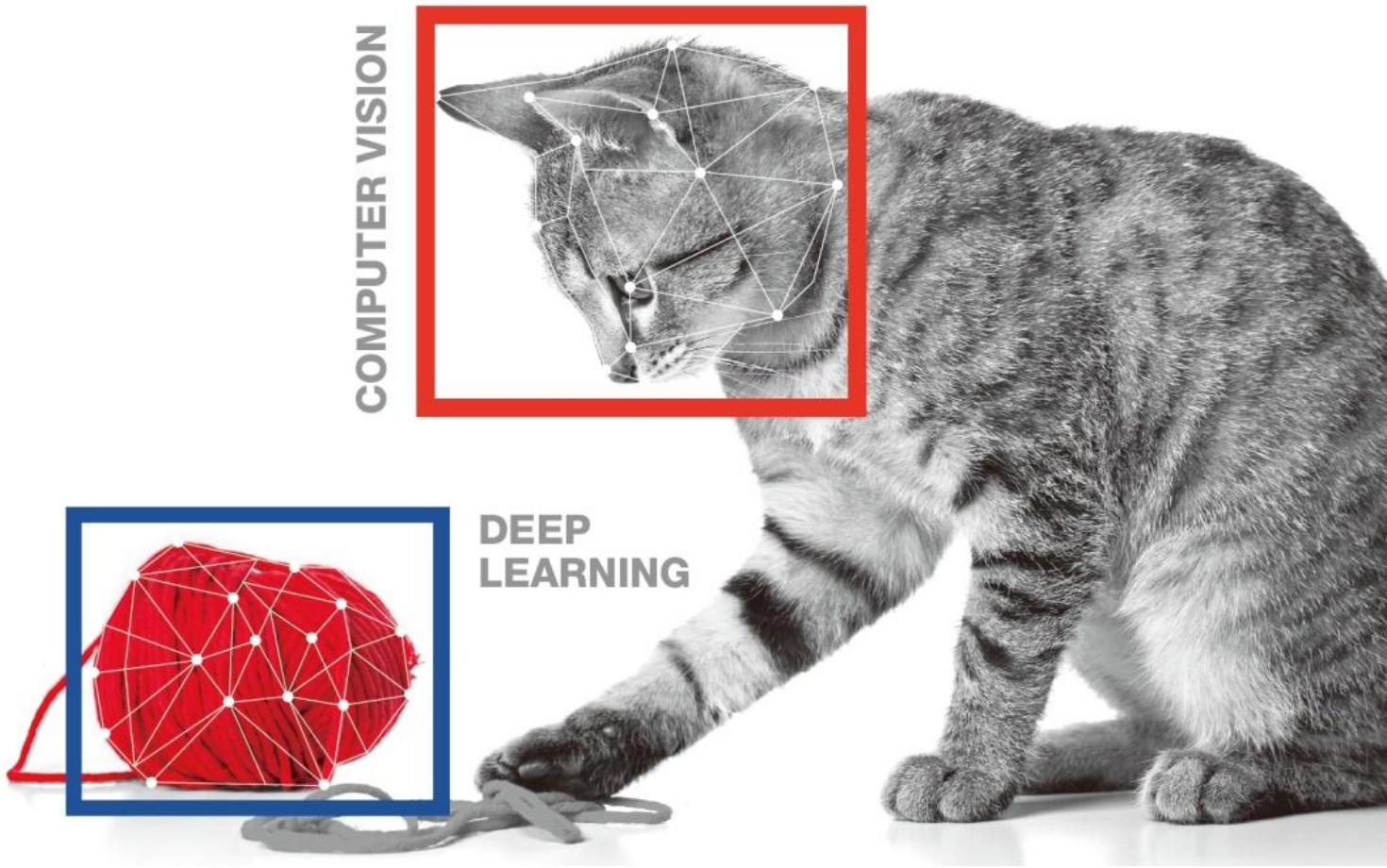
팽창: 객체를 굵게 끊어진 부분 연결
침식: 객체를 얇게 노이즈 제거
닫힘: 끊어진 부분/구멍 메우기
열림: 작은 노이즈 제거

In

```
img=cv.imread('C:/cv_workspace/data/JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)

t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b=bin_img[bin_img.shape[0]//2:bin_img.shape[0],0:bin_img.shape[0]//2+1]
plt.imshow(b,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

se=np.uint8([[0,0,1,0,0],                    # 구조 요소
             [0,1,1,1,0],
             [1,1,1,1,1],
             [0,1,1,1,0],
             [0,0,1,0,0]])

b_dilation=cv.dilate(b,se,iterations=1)    # 팽창
plt.imshow(b_dilation,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b_erosion=cv.erode(b,se,iterations=1)      # 침식
plt.imshow(b_erosion,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b_closing=cv.erode(cv.dilate(b,se,iterations=1),se,iterations=1) # 닫기
plt.imshow(b_closing,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b_opening = cv.dilate(cv.erode(b, se, iterations=1), se, iterations=1)#열림
plt.imshow(b_opening, cmap='gray'); plt.xticks([]); plt.yticks([])
plt.show()
```

요약

- OpenCV: 범용 영상처리 라이브러리 / C++ 기반 / Python 지원 / 교차플랫폼 / 무료
- 영상 표현: numpy.ndarray 구조 / (y,x) 좌표 / 색상 변환(cvtColor) / 크기 조정(resize)
- 컬러 모델 & 히스토그램: RGB·HSV 색공간 / 명암분포 표현 / 히스토그램 평활화로 화질 향상
- 히스토그램 역투영 & 얼굴 검출
  - HS 히스토그램으로 피부색 확률모델 생성
  - 역투영으로 얼굴 후보영역 탐색
  - 이동·회전 불변 / 조명·색 유사물체엔 취약
- 이진화 & 오츄 알고리즘: 임계값 기반 흑백 분리 / 자동 임계값 결정(분산 최소화)
- 모폴로지
  - 구조요소 활용 형태조작
  - 팽창·침식·열림·닫힘 / 노이즈 제거·형태 보정



Thank You:)

Any Question?