

# MODULE 8 비즈니스 문제 IMAGE

OpenCV with Python

# 강의 내용

## Chapter1 Open CV –python

- OpenCV –python 시작하기
- OpenCV – python 기초 사용법
- 기본적인 영상 처리 기법
- 필터링
- 기하학적 변화
- 영상의 특징 추출
- 이진 영상 처리
- 영상 분할과 객체 검출
- 특징점 검출과 매칭
- 객체 추적과 모션 벡터

## Chapter2 python with machine learning and deep learning

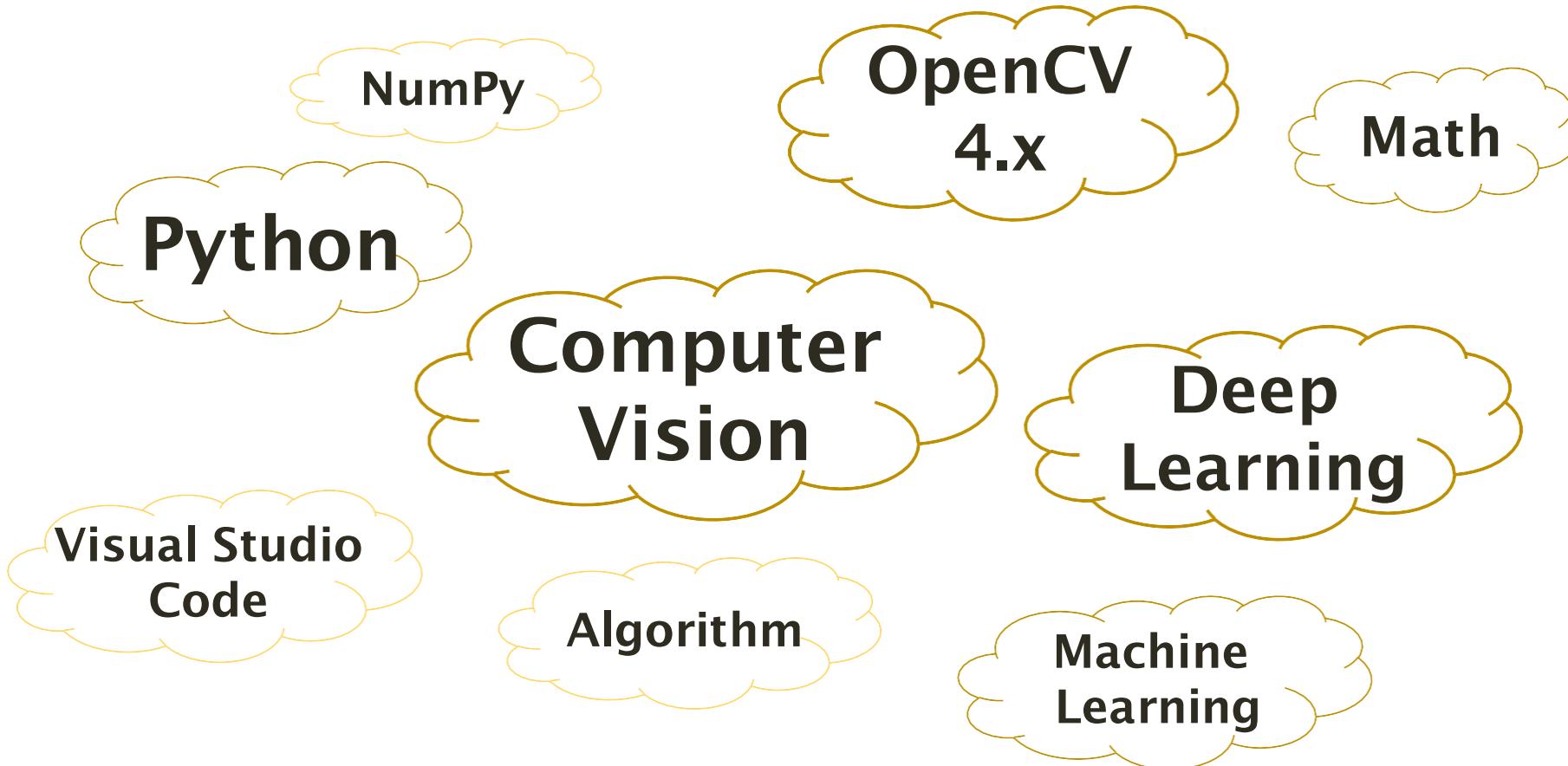
- 머신러닝
- 딥러닝
- 딥러닝 활용 객체 검출

# 1장 OpenCV-Python

# 1. OpenCV-Python 시작하기

1) 전체 코스와 컴퓨터 비전 소개

# 강의 개요



# 강의 개요

컴퓨터 비전이론의  
체계적인 학습

OpenCV 프로그래밍  
기초부터 활용까지

머신 러닝과 딥러닝

## 컴퓨터 비전 기초

- OpenCV-Python 시작하기
- OpenCV-Python 기초 사용법
- 기본적인 영상 처리 기법
- 필터링과 모폴로지
- 기하학적 변환

## 컴퓨터 비전 심화

- 영상의 특징추출
- 이진 영상처리
- 영상 분할 및 검출
- 키포인트 매칭
- 객체 추적과 모션 벡터

## 머신 러닝과 딥러닝

- 머신 러닝
- 딥러닝 영상 인식
- 딥러닝과 객체 검출

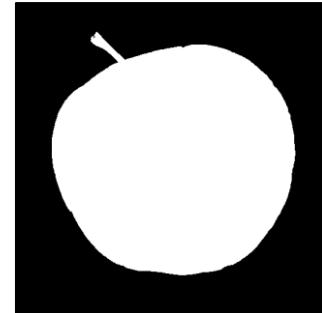
# 컴퓨터 비전

## ■ 컴퓨터 비전(Computervision)

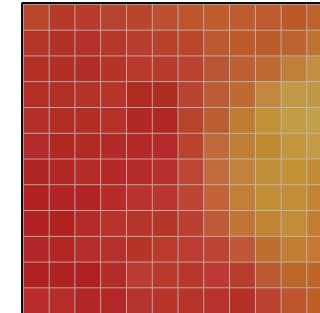
- 컴퓨터를 이용하여 정지 영상 또는 동영상으로부터 의미 있는 정보를 추출하는 방법을 연구하는 학문
- 즉, 사람이 눈으로 사물을 보고 인지하는 작업을 컴퓨터가 수행하게끔 만드는 학문



사과?



둥글다?



빨간색이다?



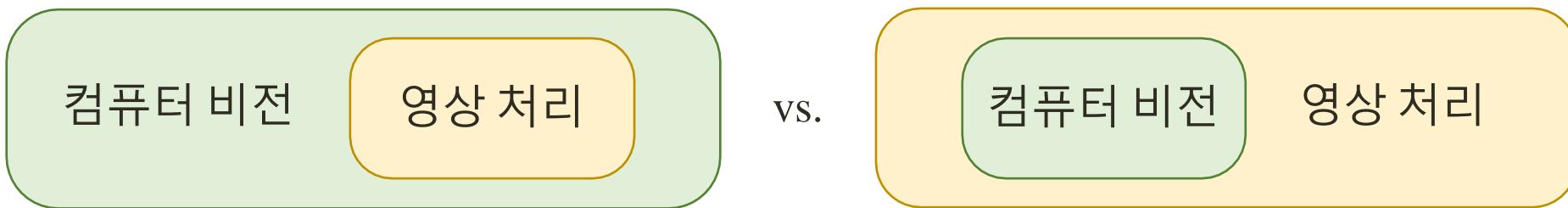
꼭지 모양?



사과가 몇 개??

# 컴퓨터 비전

## ■ 컴퓨터 비전과 영상 처리(image processing)

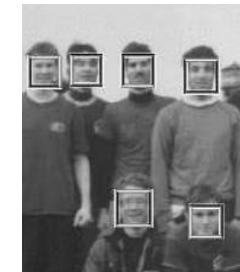
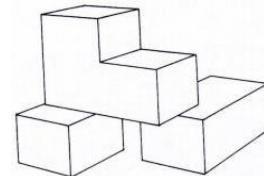


- 영상 처리는 영상을 입력으로 받아 화질을 개선하는 등의 처리를 하여 다시 영상을 출력으로 내보내는 작업
- 영상 처리는 컴퓨터 비전을 위한 전처리 작업
- 영상 처리는 영상을 다루는 모든 학문과 응용을 통틀어 지칭
- 컴퓨터 비전은 영상 인식과 같은 고수준의 영상 처리를 지칭

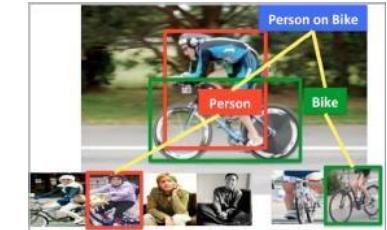
컴퓨터 비전 ≈ 영상 처리

# 컴퓨터 비전 역사

1966년 MIT “The Summer Vision Project” 연구가 컴퓨터 비전의 시초



IMAGENET



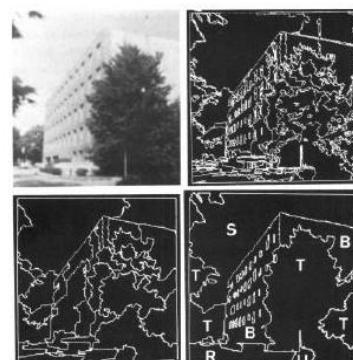
1960'

1970'

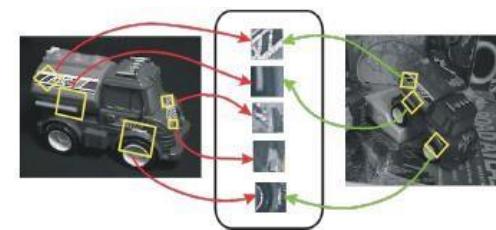
2000'

2010'

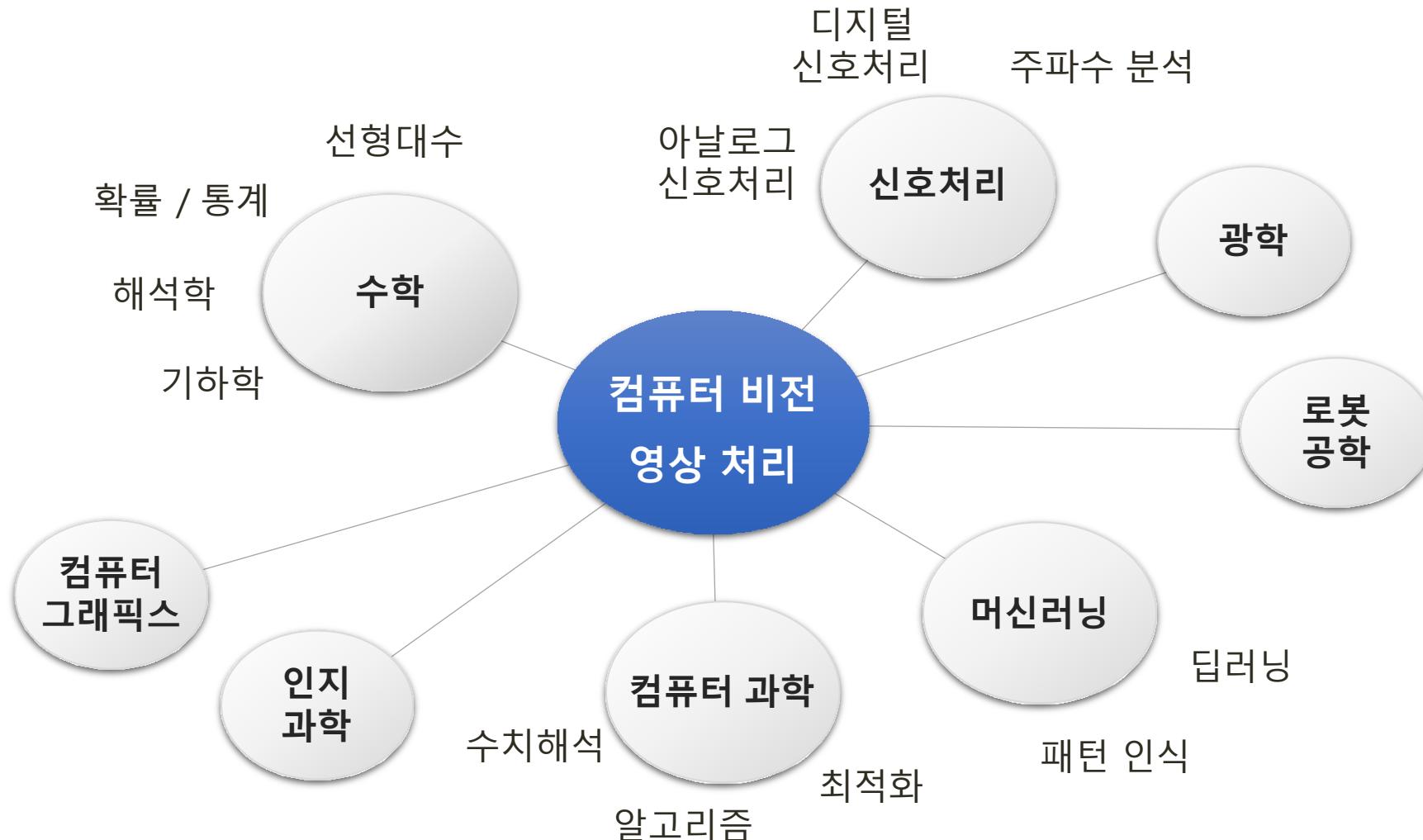
아날로그 방식의  
사진 편집



위성으로부터  
전송 받은 달 표면  
사진의 화질 복원

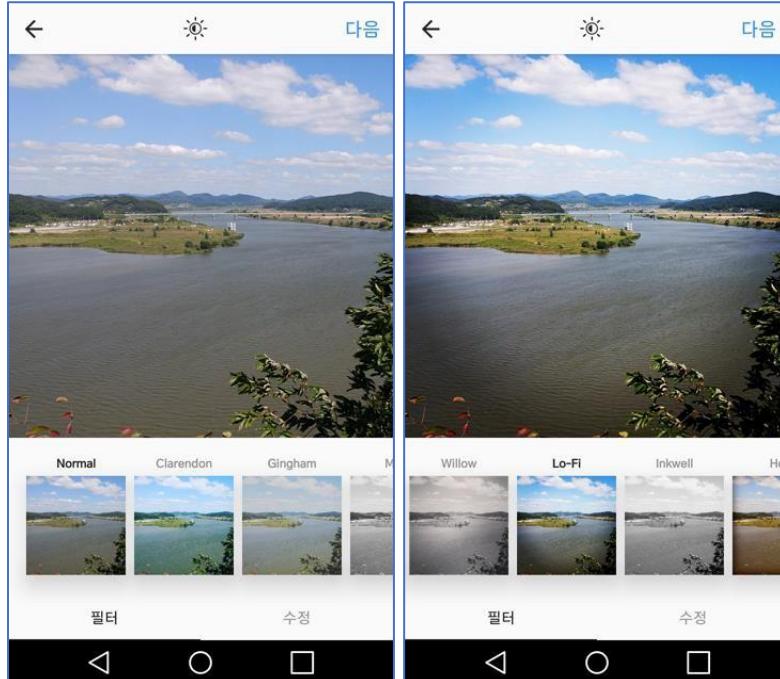


# 컴퓨터 비전 관련 분야

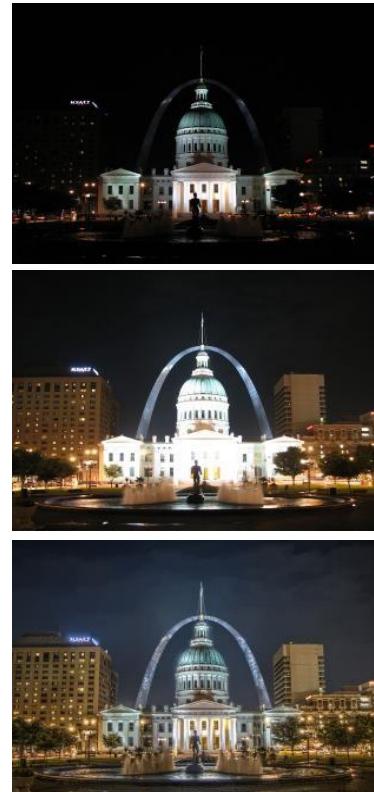


# 컴퓨터 비전 연구 분야

## ■ 영상의 화질 개선



Filtering App



HDR



Image Noise Reduction

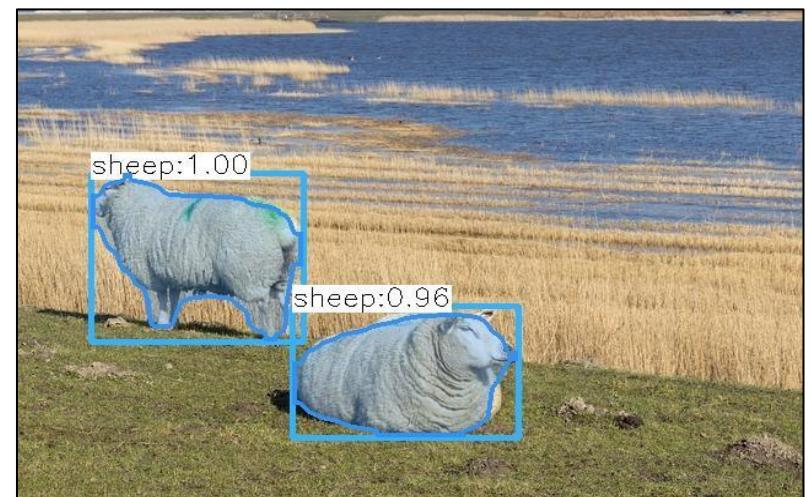
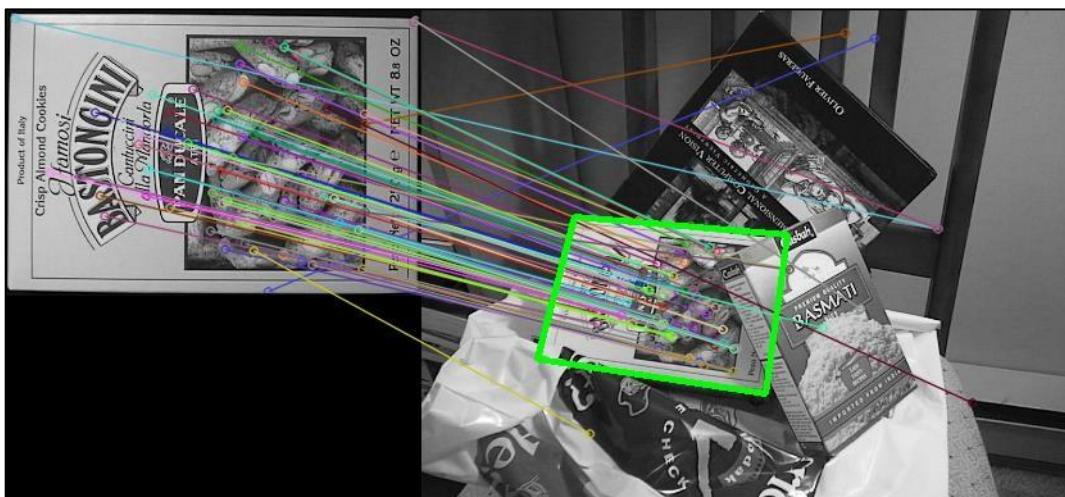
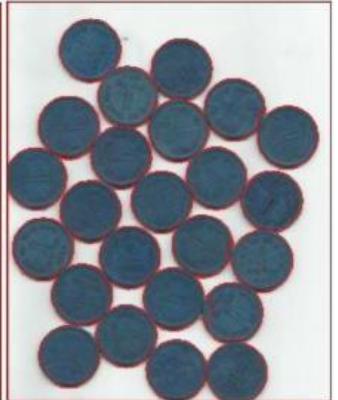
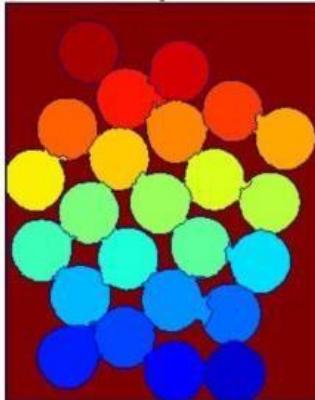
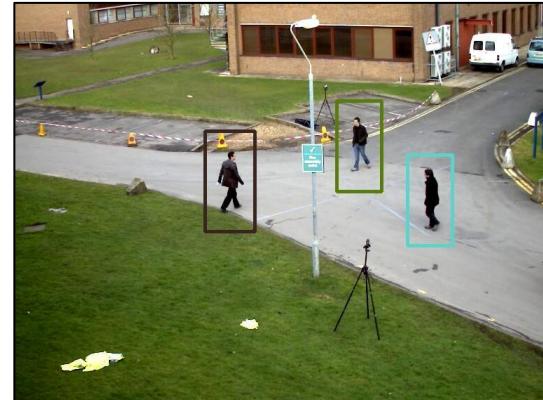
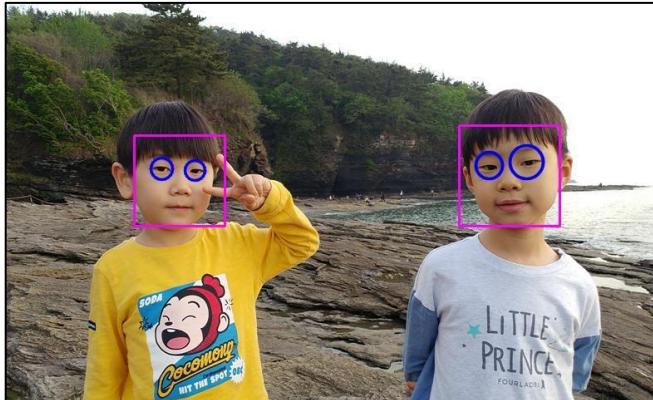


Super Resolution

[https://en.wikipedia.org/wiki/High-dynamic-range\\_imaging](https://en.wikipedia.org/wiki/High-dynamic-range_imaging), <https://arxiv.org/pdf/1707.02921.pdf>

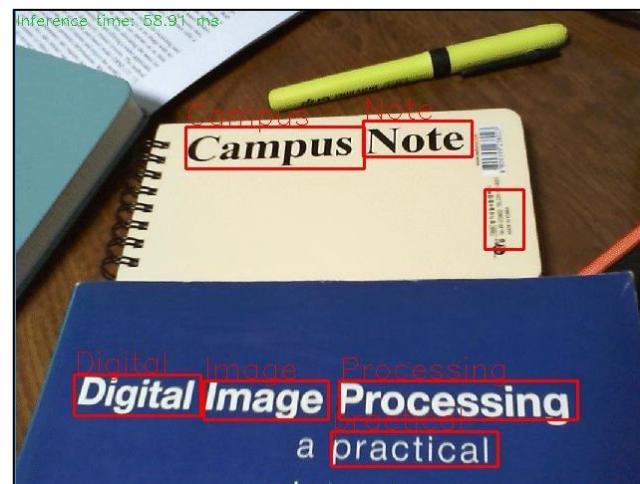
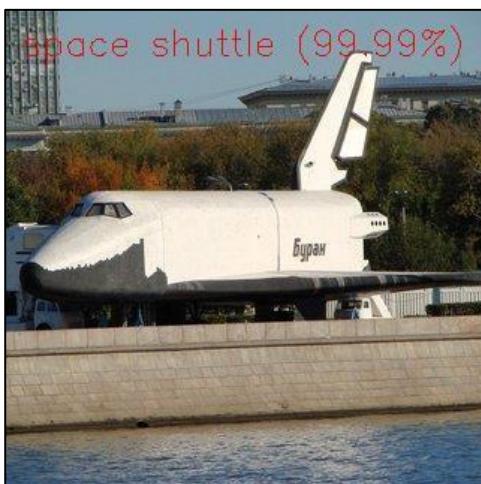
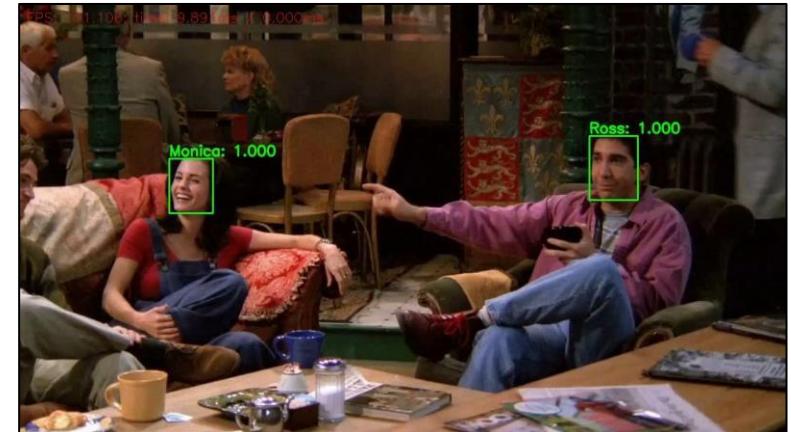
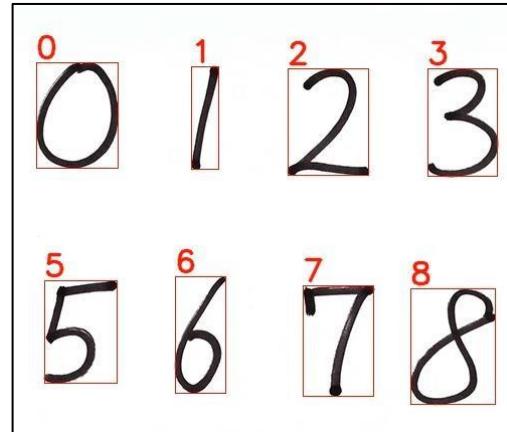
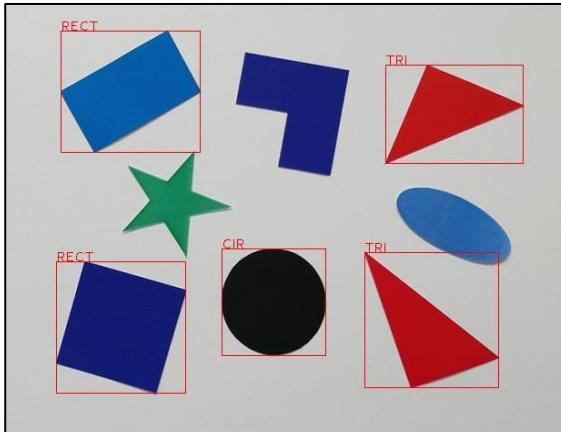
# 컴퓨터 비전 연구 분야

## ■ 객체 검출(Object detection)과 영상 분할



# 컴퓨터 비전 연구 분야

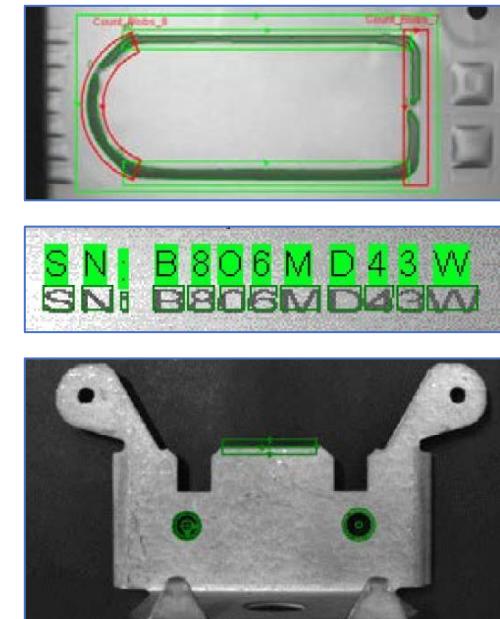
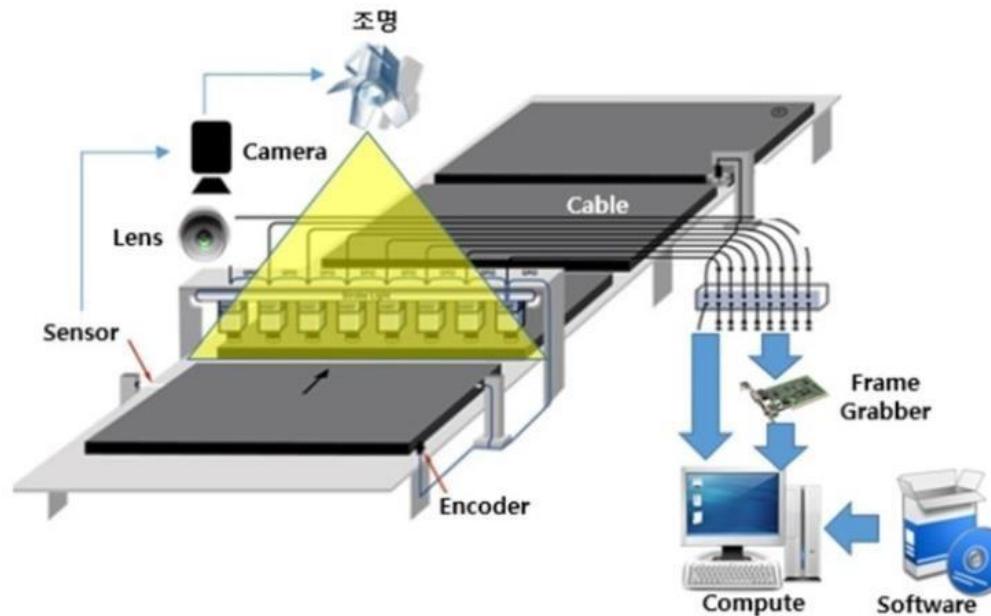
## ■ 인식(Recognition)



# 컴퓨터 비전 응용 분야

## ■ 머신 비전(machine vision)

- 공장 자동화: 제품의 불량 검사, 위치 확인, 측정 등
- 높은 정확도와 빠른 처리 시간 요구
- 조명, 렌즈, 필터, 실시간 (Real-time) 처리



<https://laonple.blog.me/>, <http://www.cognex.com>

# 컴퓨터 비전 응용 분야

## ■ 인공지능 서비스

- 입력 영상을 객체와 배경으로 분할 ⑦ 객체와 배경 인식 ⑦ 상황 짐  
⑦ 로봇과 자동차의 행동 지시
- Computer Vision + Sensor Fusion + Deep Learning
- 인공지능 로봇, Amazon Go, 구글/테슬라의 자율 주행 자동차



<https://youtu.be/NrmMk1Myrxc?t=26>



<https://youtu.be/wuhbqcMzOaw?t=7>

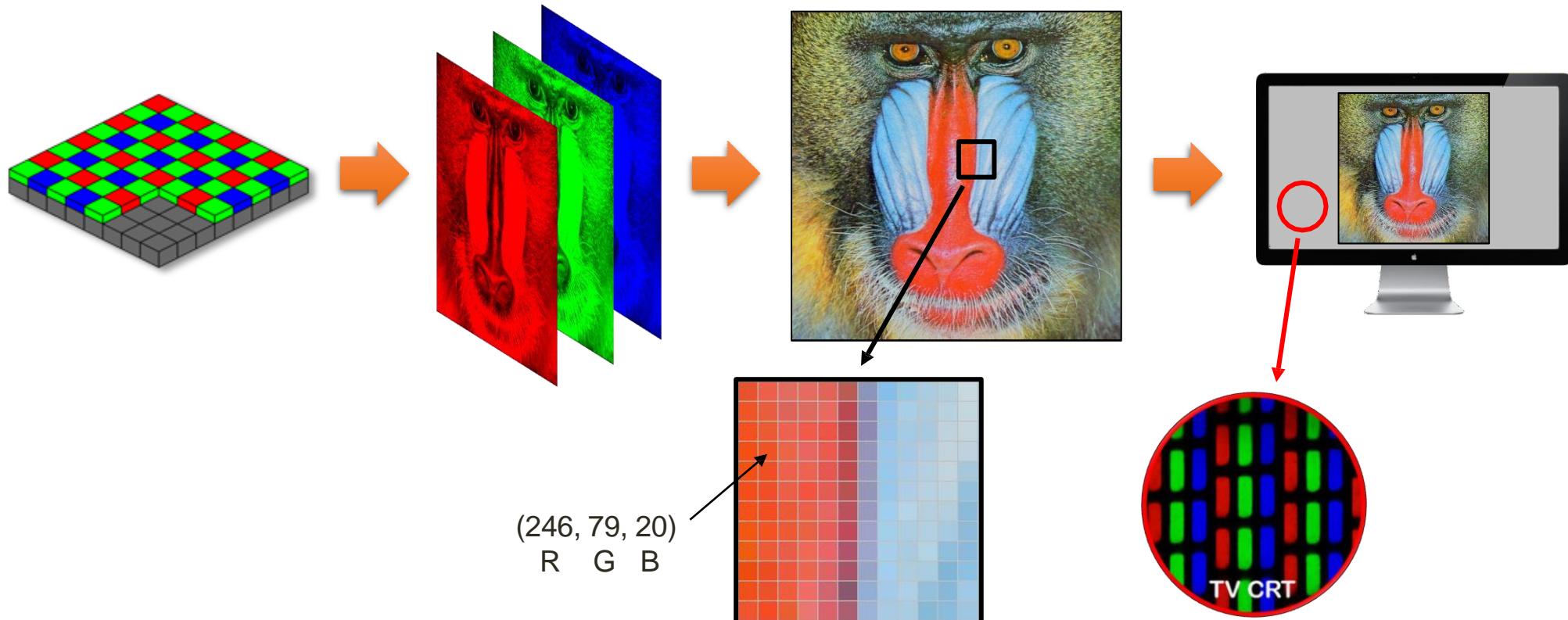
# 1. OpenCV-Python 시작하기

2) 영상의 구조와 표현

# 영상의 표현 방법

## ■ 영상(image)이란?

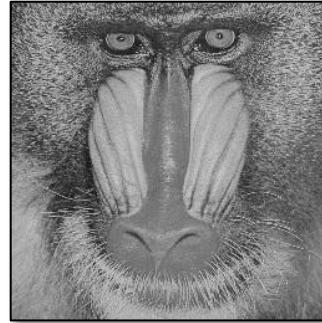
- 픽셀(pixel)이 바둑판 모양의 격자에 나열되어 있는 형태 (2차원 행렬)
- 픽셀: 영상의 기본 단위, picture element, 화소(畫素)



# 영상의 표현 방법

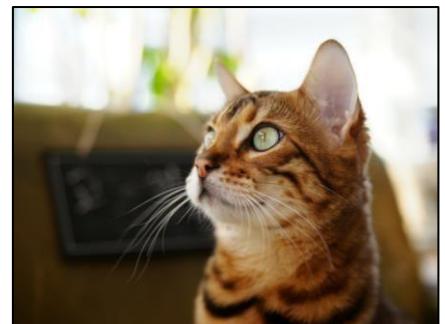
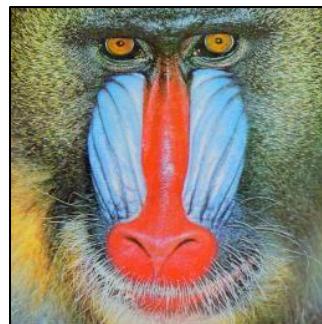
## ■ 그레이스케일(grayscale) 영상

- 흑백 사진처럼 색상 정보가 없이 오직 밝기 정보만으로 구성된 영상
- 밝기 정보를 256 단계로 표현



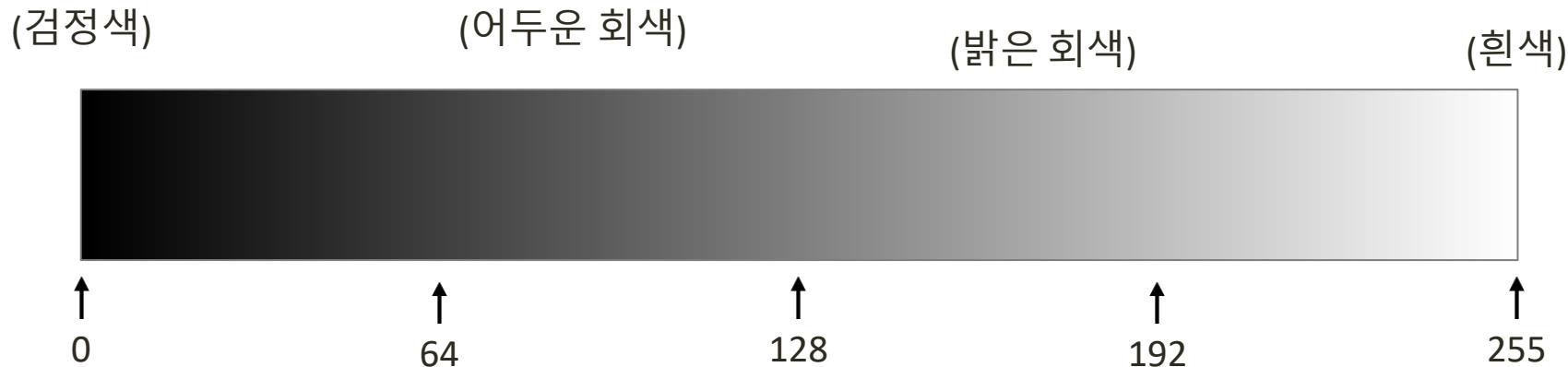
## ■ 트루컬러(truecolor) 영상

- 컬러 사진처럼 색상 정보를 가지고 있어서 다양한 색상을 표현할 수 있는 영상
  - Red, Green, Blue 색 성분을 256 단계로 표현
- ⑦  $256^3 = 16,777,216$  색상 표현 가능



# 영상의 표현 방법

- 그레이스케일 영상의 픽셀 값 표현
  - 밝기 성분을 0 ~ 255 범위의 정수로 표현

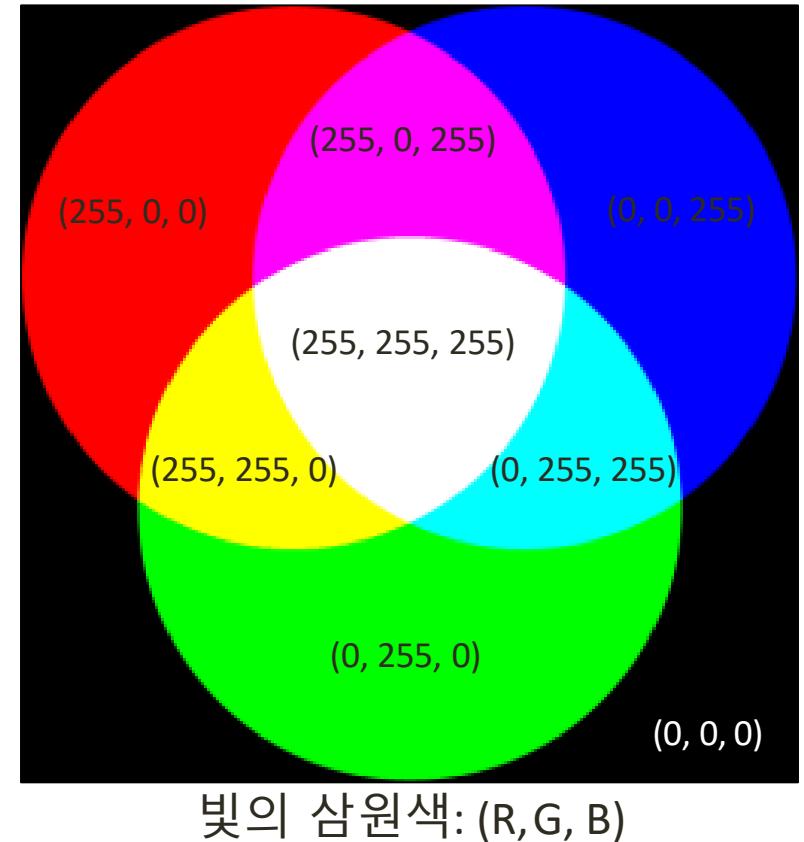


- 프로그래밍 언어에서 표현 방법: 1Byte 사용
  - C/C++ → `unsigned char`
  - Python → `numpy.uint8`

# 영상의 표현 방법

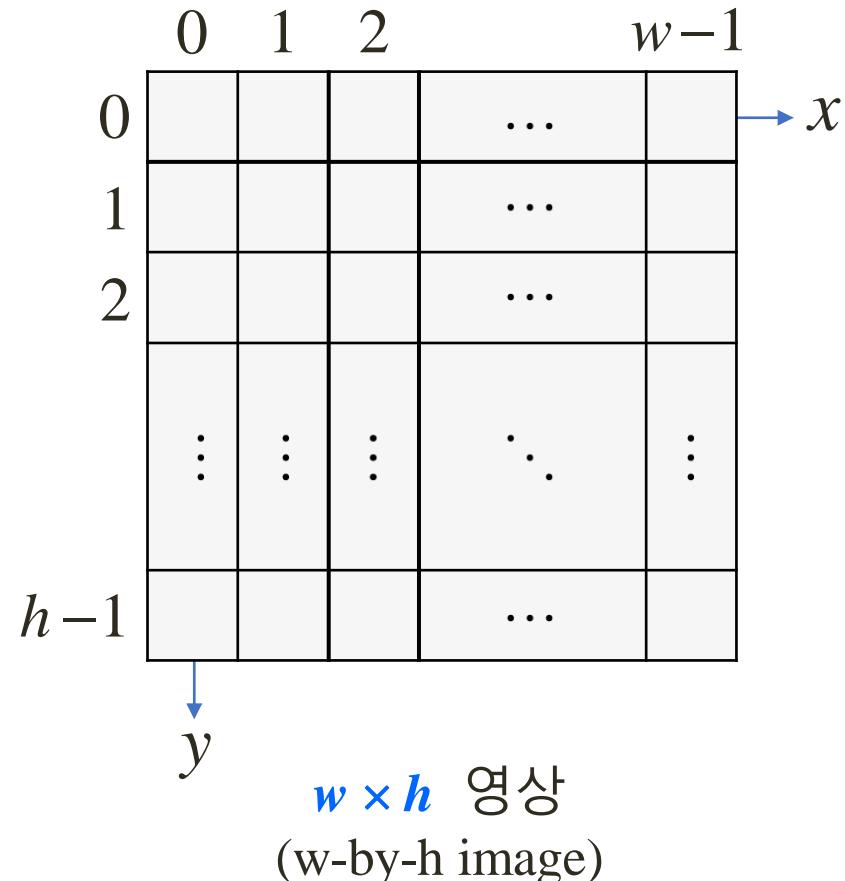
## ■ 컬러 영상의 픽셀 값 표현

- R, G, B 색 성분의 크기를 각각 0 ~ 255 범위의 정수로 표현
  - 0 : 해당 색 성분이 전혀 없는 상태
  - 255 : 해당 색 성분이 가득 있는 상태
- 프로그래밍 언어에서 표현 방법: 3Bytes 사용
  - C/C++ → 구조체, 클래스
  - Python → 튜플, numpy.ndarray



# 영상의 표현 방법

- 영상에서 주로 사용되는 좌표계



$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & a_{M,2} & \cdots & a_{M,N} \end{bmatrix}$$

$M \times N$  행렬  
(m-by-n matrix)

# 영상의 표현 방법

## ■ 그레이스케일 영상에서 픽셀 값 분포의 예

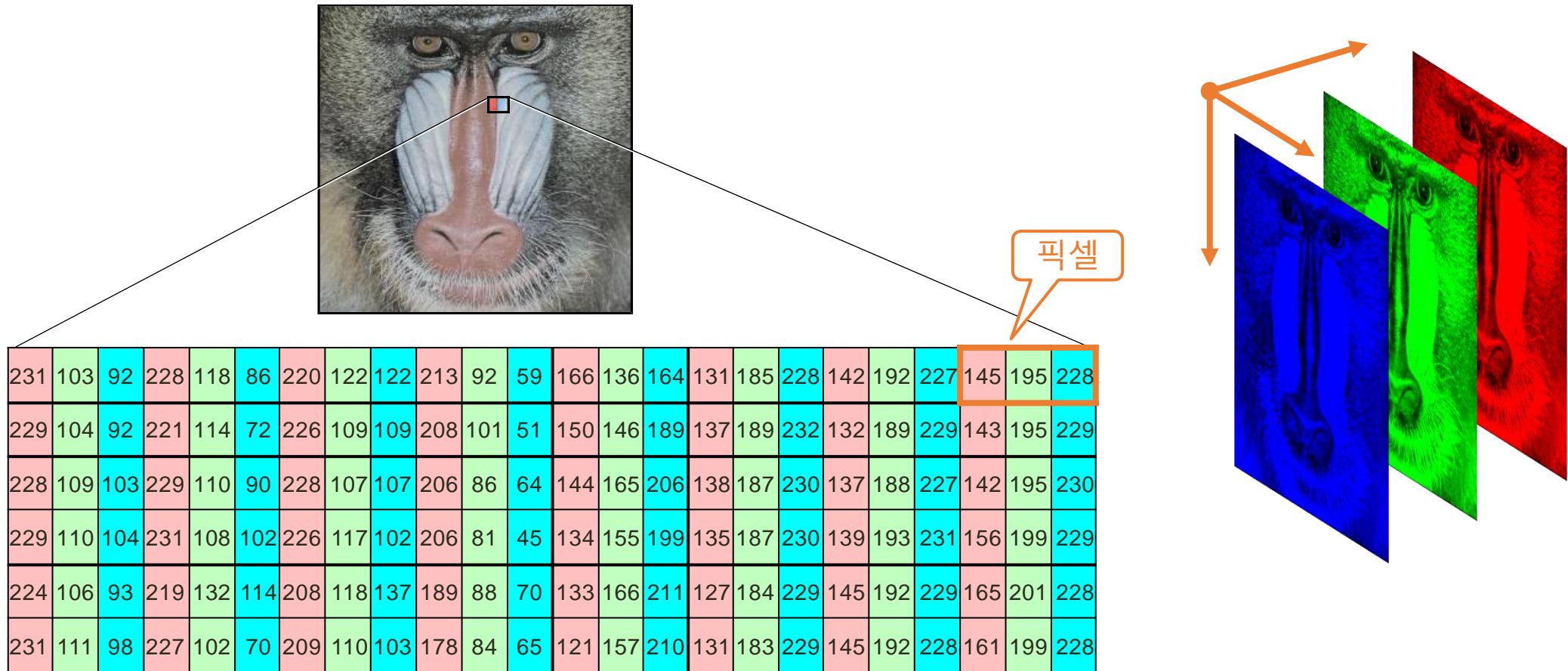


187	187	187	194	197	173	77	25	19	19
190	187	190	191	158	37	15	14	20	20
187	182	180	127	32	16	13	16	14	12
184	186	172	100	20	13	15	18	13	18
186	190	187	127	18	14	15	14	12	10
189	192	192	148	16	15	11	10	10	9
192	195	181	37	13	10	10	10	10	10
189	194	54	14	11	10	10	10	9	8
189	194	19	16	11	11	10	10	9	9
192	88	12	11	11	10	10	10	9	9

픽셀

# 영상의 표현 방법

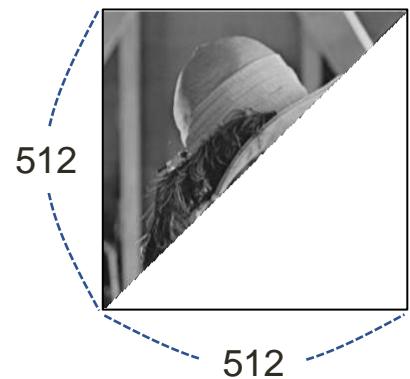
## ■ 트루컬러 영상에서 픽셀 값 분포의 예



# 영상 데이터의 크기

## ■ 영상 데이터 크기 분석

- 그레이스케일 영상: (가로 크기) × (세로 크기) Bytes
- 트루컬러 영상: (가로 크기) × (세로 크기) × 3 Bytes



$$512 \times 512 = 262144 \text{ Bytes}$$



$$1920 \times 1080 \times 3 = 6220800 \text{ Bytes} \\ \approx 6 \text{ MBytes}$$

# 영상 파일 형식 특징

## BMP

- 픽셀 데이터를 압축하지 않고 그대로 저장  
**⑦** 파일 용량이 큰 편
- 파일 구조가 단순해서 별도의 라이브러리 도움 없이 파일 입출력 프로그래밍 가능

## JPG

- 주로 사진과 같은 컬러 영상을 저장
- 손실 압축(lossy compression)
- 압축률이 좋아서 파일 용량이 크게 감소  
디지털 카메라 사진 포맷

## GIF

- 256 색상 이하의 영상을 저장  
**⑦** 일반 사진을 저장 시 화질 열화
- 무손실 압축(lossless compression)
- 움직이는 GIF 지원

## PNG

- Portable Network Graphics
- 무손실 압축 (컬러 영상도 무손실 압축)
- 알파 채널(투명도)을 지원

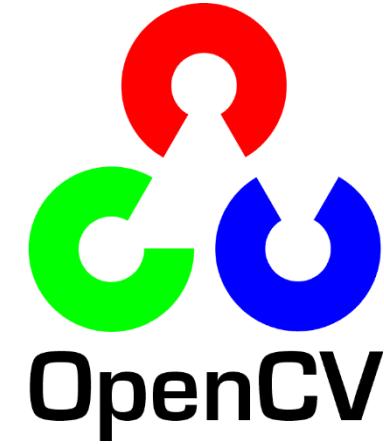
# 1. OpenCV-Python 시작하기

## 3) OpenCV 소개와 설치

# OPENCV 개요

## ■ What is OpenCV?

- Open source
- Computer vision & machine learning
- Software library

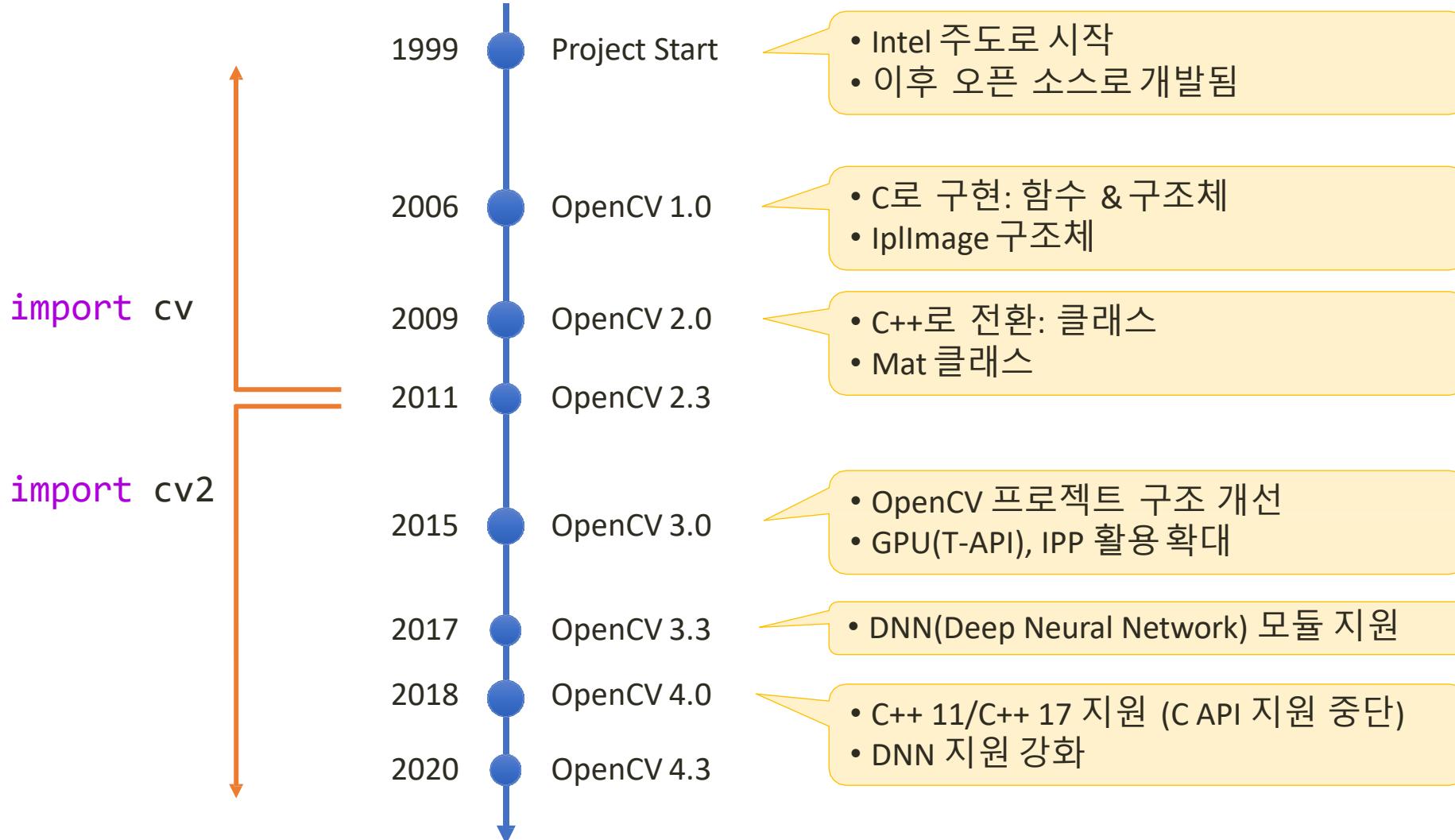


<http://www.opencv.org/>

## ■ Why OpenCV?

- BSD license ... **Free for academic & commercial use**
- Multiple interface ... **C, C++, Python, Java, JavaScript, MATLAB, etc.**
- Multiple platform ... Windows, Linux, Mac OS, iOS, Android
- Optimized ... **CPU instructions, Multi-core processing, OpenCL, CUDA**
- Popular ... More than 18 million downloads
- Usage ... Stitching streetview images, detecting intrusions, monitoring mine equipment, helping robots navigate and pick up objects, Interactive art, etc.

# OPENCV 역사



# OPENCV 구성

## ■ OpenCV main modules

- Core, widely used, infrastructures
- <https://github.com/opencv/opencv/>

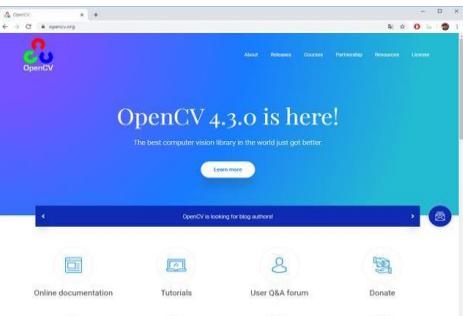
calib3d, core, dnn, features2d, flann, gapi, highgui, imgcodecs, imgproc, java, js, ml, objdetect, photo, python, stitching, ts, video, videoio, world

## ■ OpenCV extra modules

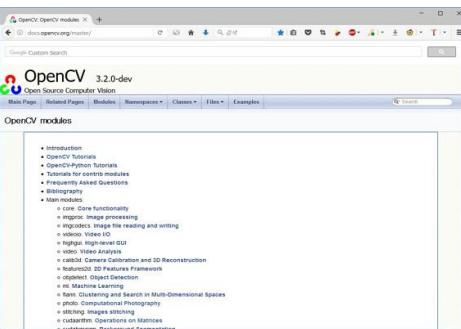
- Brand new, unpopular, non-free, HW dependency, etc.
- [https://github.com/opencv/opencv\\_contrib/](https://github.com/opencv/opencv_contrib/)

aruco, bgsegm, bioinspired, ccalib, cnn\_3dobj, cudaarithm, cudabgsegm, ... , cudawarping, cudev, cvv, datasets, dnns\_easily\_fooled, dnn\_objdetect, dpm, face, freetype, fuzzy, hdf, hfs, img\_hash, line\_descriptor, matlab, optflow, ovis, phase\_unwrapping, plot, reg, rgbd, saliency, sfm, shape, stereo, structured\_light, superres, surface\_matching, text, tracking, videotab, viz, xfeatures2d, ximgproc, xobjdetect, xphoto

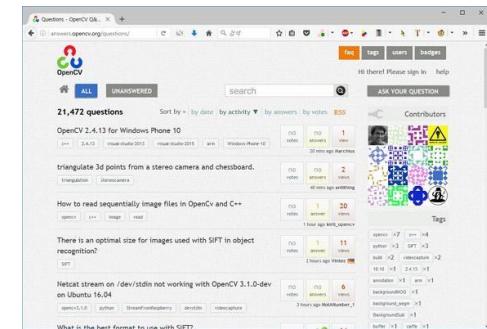
# OpenCV 관련 사이트



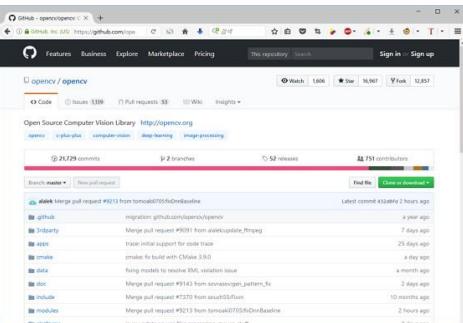
<http://opencv.org/>



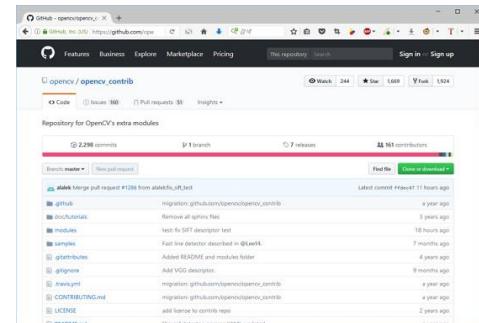
<http://docs.opencv.org/master/>



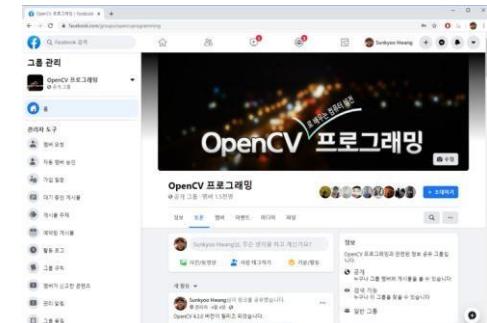
<http://answers.opencv.org/questions/>



<https://github.com/opencv/opencv/>



[https://github.com/opencv/opencv\\_contrib/](https://github.com/opencv/opencv_contrib/)



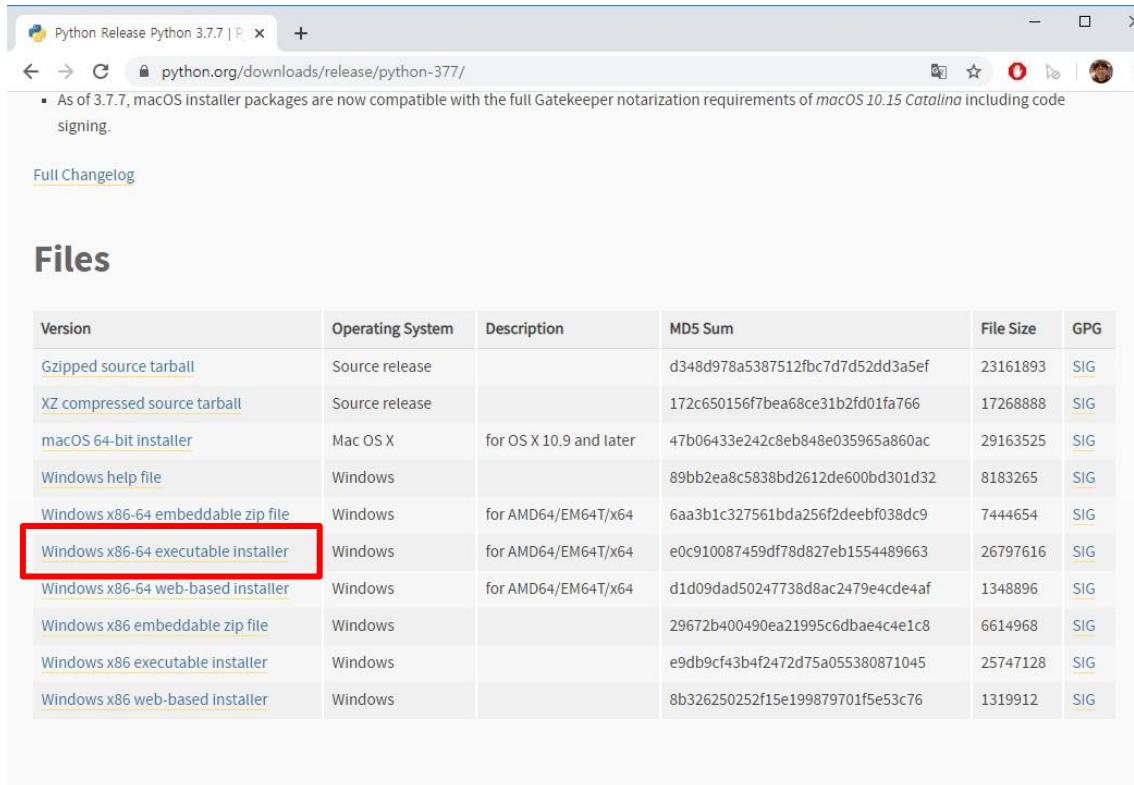
<https://www.facebook.com/groups/opencvprogramming/>

# PYTHON 설치

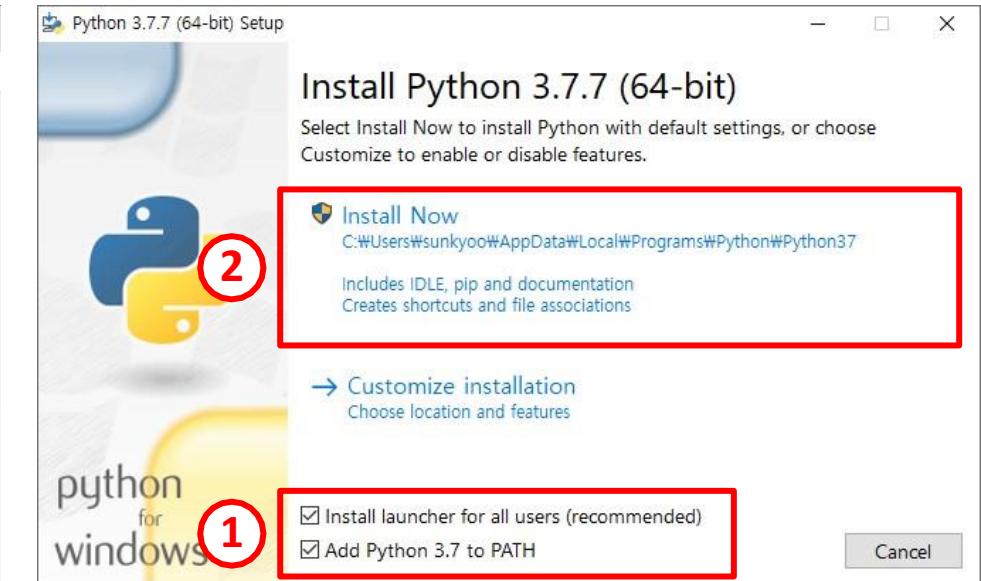
## ■ Python 설치하기

- Python 3.7, 64-bit 설치 실행 파일 다운로드:

<https://www.python.org/downloads/release/python-377/>



Version	Operating System	Description	MD5 Sum	File Size	PGP
Gzipped source tarball	Source release		d348d978a5387512fbc7d7d52dd3a5ef	23161893	SIG
XZ compressed source tarball	Source release		172c650156f7bea68ce31b2fd01fa766	17268888	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	47b06433e242c8eb848e035965a860ac	29163525	SIG
Windows help file	Windows		89bb2ea8c5838bd2612de600bd301d32	8183265	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	6aa3b1c327561bda256f2deebf038dc9	7444654	SIG
<b>Windows x86-64 executable installer</b>	Windows	for AMD64/EM64T/x64	e0c910087459df78d827eb1554489663	26797616	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	d1d09dad50247738d8ac2479e4cde4af	1348896	SIG
Windows x86 embeddable zip file	Windows		29672b400490ea21995c6dbae4c4e1c8	6614968	SIG
Windows x86 executable installer	Windows		e9db9cf43b4f2472d75a055380871045	25747128	SIG
Windows x86 web-based installer	Windows		8b326250252f15e199879701f5e53c76	1319912	SIG



# OPENCV-PYTHON 설치

## ■ pip 명령으로 설치하기

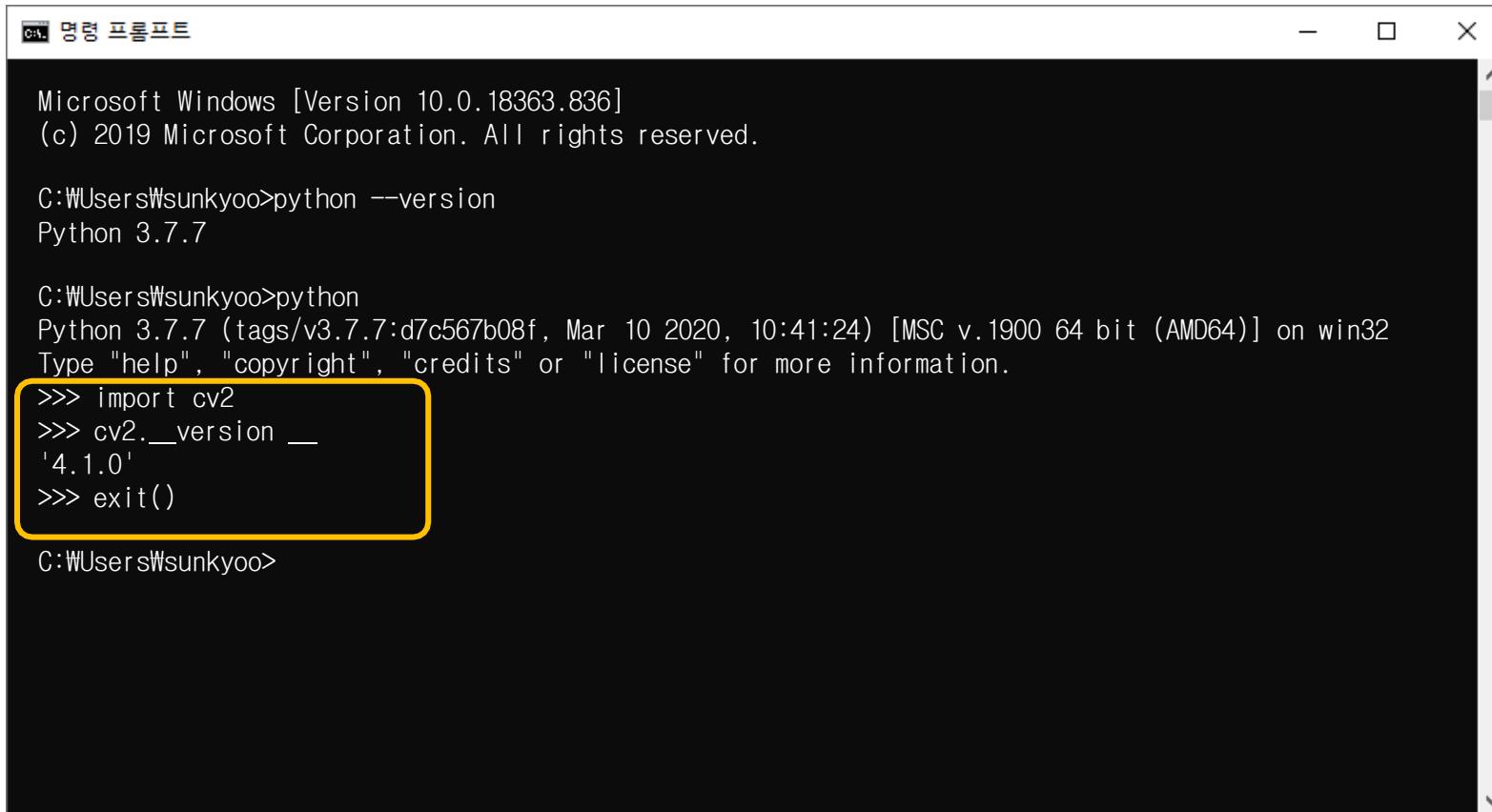
```
> pip install opencv-python
```

- <PYTHON\_PATH>\Lib\site-packages 폴더 아래에 cv,opencv\_python-4.2.0.34.dist-info 폴더가 생성되고, 그 아래에 cv2.cp37-win\_amd64.pyd 파일이 저장됨.
- 설치되는 OpenCV 버전은 <https://pypi.org/>에서 확인 가능
- OpenCV 추가 모듈도 함께 사용하려면 [opencv-contrib-python](#) 패키지를 설치
- OpenCV 4.2.0 버전에서 cv2.imread() 함수를 사용하여 영상을 그레이스케일 형식으로 불러올 때 픽셀 값이 잘못 설정되는 문제가 있음  
**⑦** cv2.imread() 함수가 정상 동작하는 OpenCV를 설치하려면 아래 명령 사용

```
> pip install opencv-python==4.3.0.xx
```

# OPENCV-PYTHON 설치

- pip 명령으로 opencv-python 설치 확인하기
  - 명령 프롬프트에서 확인



```
C:\ 명령 프롬프트
Microsoft Windows [Version 10.0.18363.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\sunkyoo>python --version
Python 3.7.7

C:\Users\sunkyoo>python
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.1.0'
>>> exit()

C:\Users\sunkyoo>
```

# 1. OpenCV-Python 시작하기

4) VS Code 설치와 개발 환경 설정

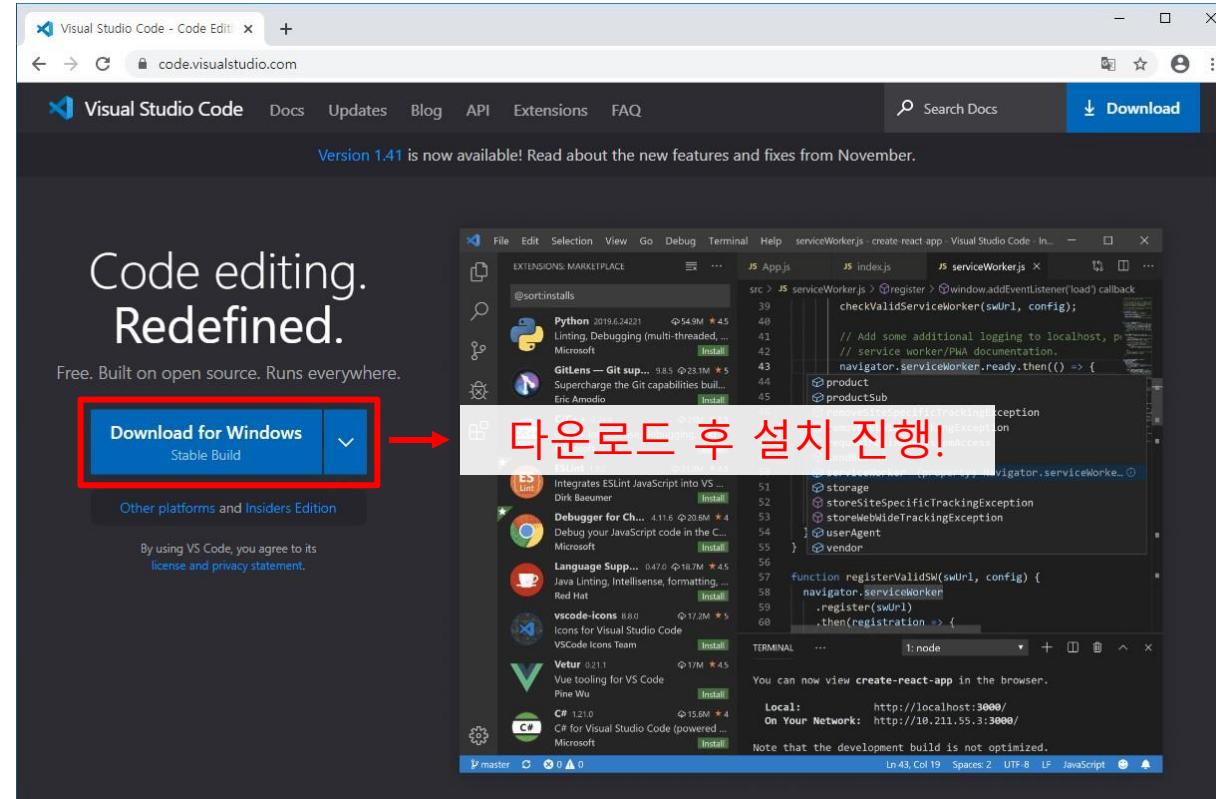
# OPENCV-PYTHON 코딩 작업 환경

- 메모장 + 명령프롬프트
- 주피터 노트북(Jupyter Notebook)
  - 웹 브라우저에서 파이썬 코드를 작성 & 블록 단위 실행
  - 마크업 언어와 그림 등을 활용한 설명 추가가 쉬움
- 파이썬 IDE
  - PyCharm, Visual Studio Code, Spider 등
  - 편리한 디버깅
  - OpenCV에서 제공하는 GUI 기능 사용

# VS CODE 설치

## ■ Visual Studio Code

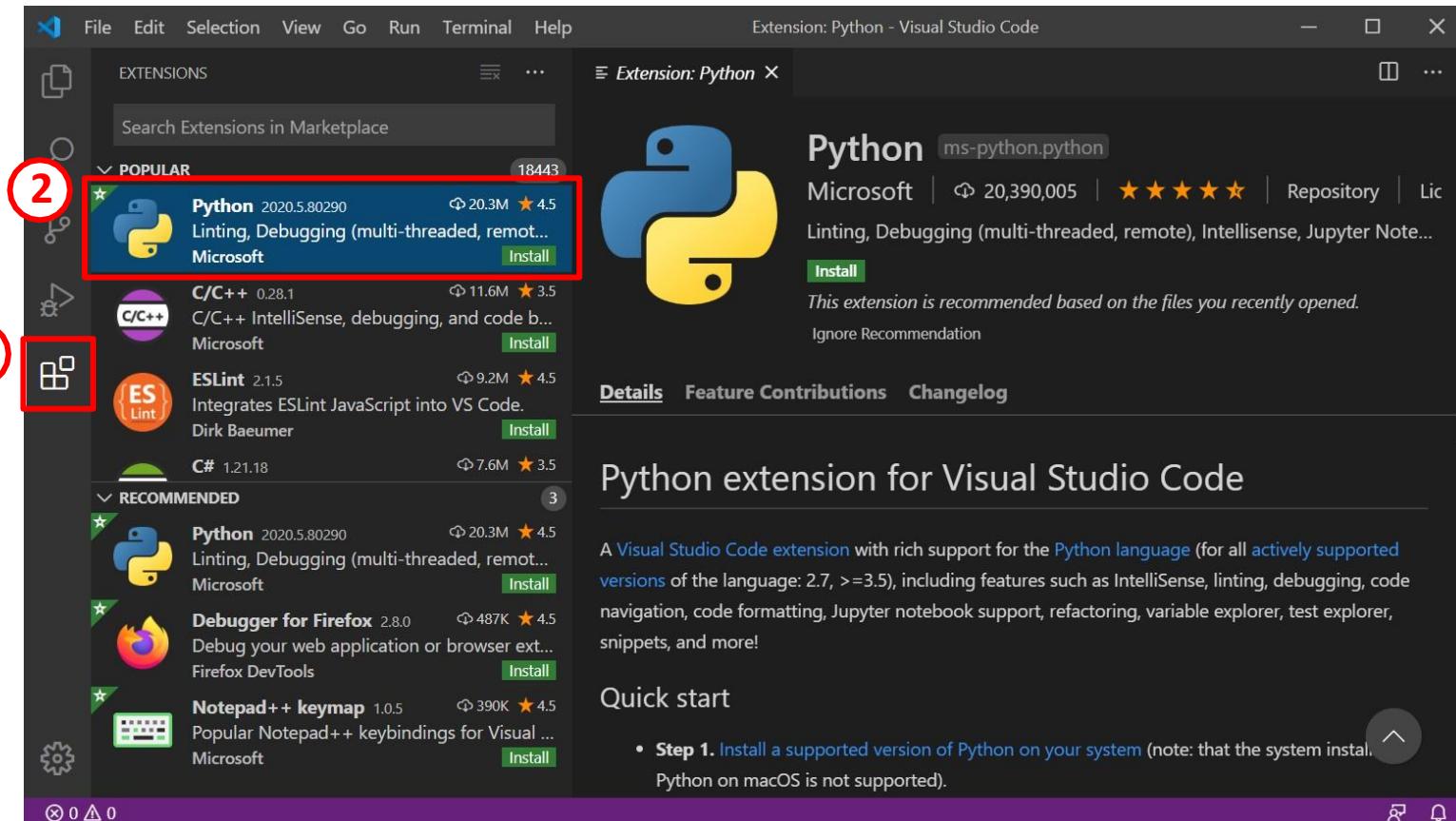
- Microsoft에서 개발한 범용 소스 코드 편집기 (Windows, MacOS, Linux)
- 설치 프로그램 다운로드: <https://code.visualstudio.com/>



# VS CODE 설치

## ■ Python 확장 설치

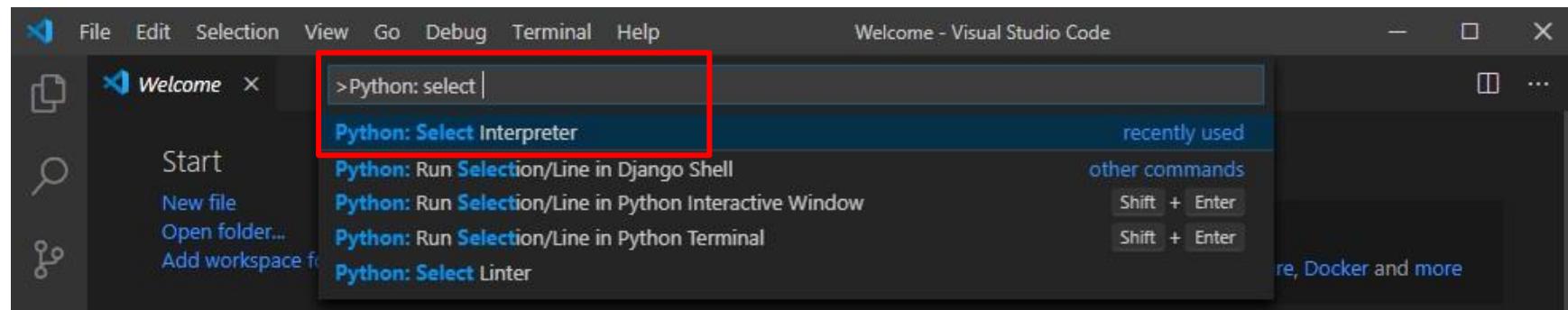
- Extensions 뷰에서 Python (by Microsoft) 설치



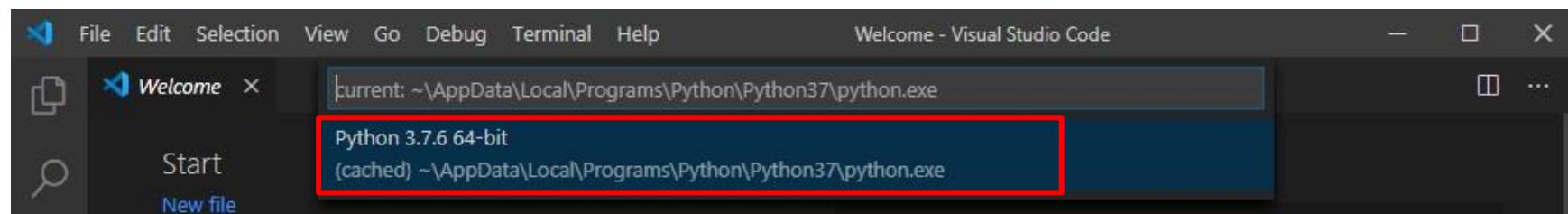
# VS CODE 설치

## ■ Python 인터프리터 선택

- [View] → [Command Palette...] 메뉴 선택 후 "Python: Select Interpreter" 입력 & 선택



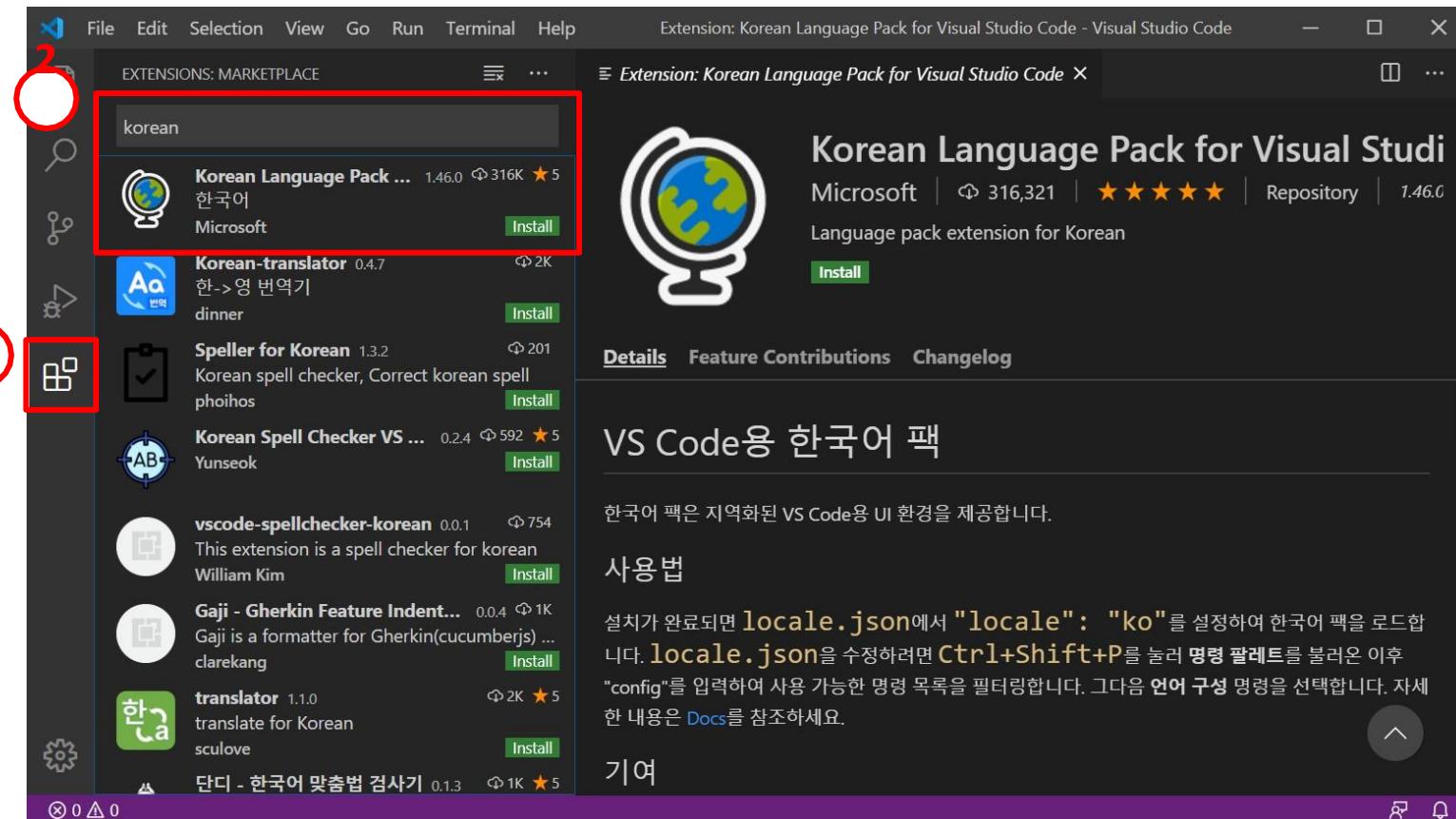
- 설치된 Python 인터프리터를 선택



# VS CODE 설치

## ■ 한글 언어팩 설치

- Extensions 뷰에서 "Korean Language Pack" 검색하여 설치



# HELLOCV.PY 프로그램 만들기

실습: HelloCV.py

## ■ VS Code에서 새 Python 프로그램 만들기

- VS Code에서 [파일] ⑦ [폴더 열기] 메뉴 선택한 후, 강의 예제 파일 폴더 [ch01] 선택
  - (e.g.) C:\coding\python\opencv\ch01
- 아래 그림처럼 [새 파일] 버튼 클릭 후, **HelloCV.py** 파일 이름 입력



- 편집창에 소스 코드 입력

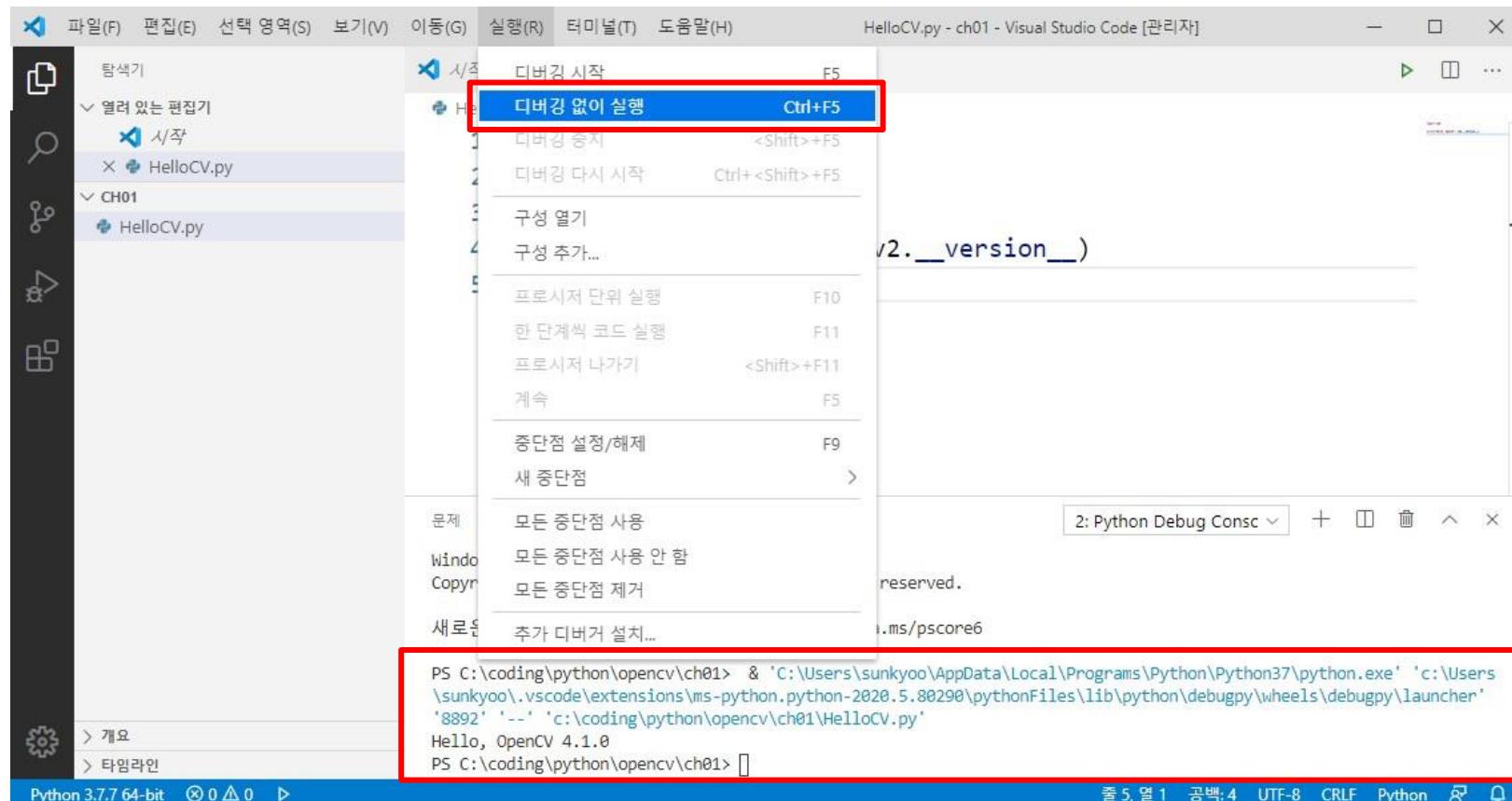
```
import cv2

print('Hello OpenCV', cv2.__version_)
```

# HELLOCV.PY 프로그램 만들기

## ■ VS Code에서 Python 프로그램 실행하기

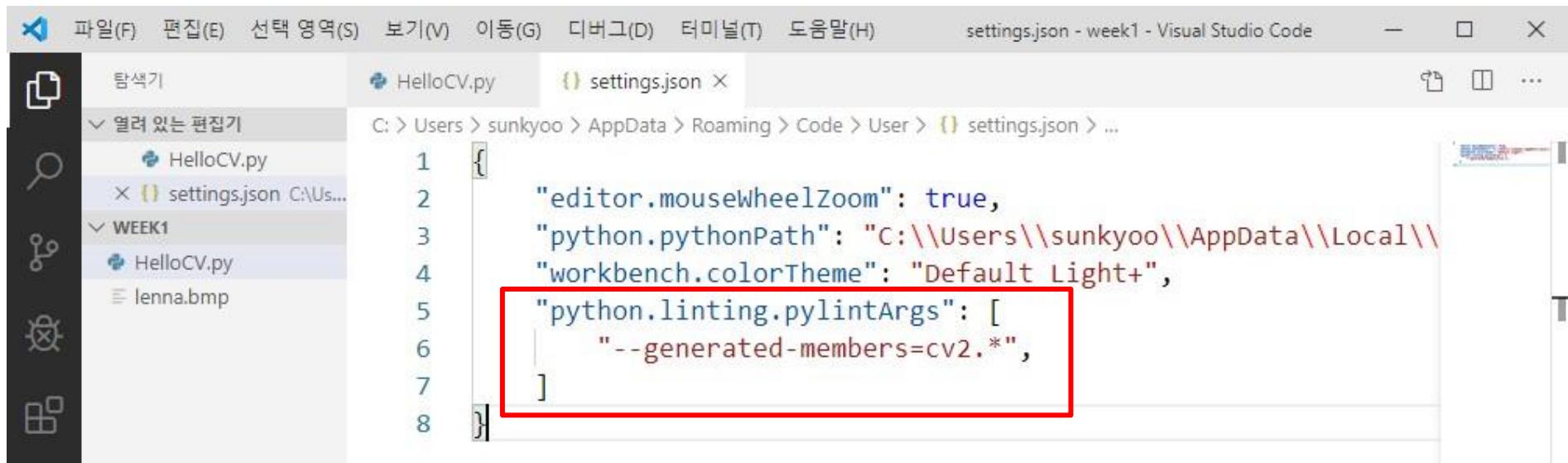
- [실행] ⑦ [디버깅 없이 실행 (CTRL+F5)] 메뉴 선택



# HELLOCV.PY 프로그램 만들기

## ■ VS Code 편집창에서 cv2 에러 없애기

- VS Code pylint에서 cv2 모듈의 멤버를 제대로 인식하지 못하는 문제가 있음  
(실행은 정상적으로 동작함)
- [보기] ⑦ [명령 팔레트] (Ctrl + Shift + P)
- "Preferences: Open Settings (JSON)" 선택
- "python.linting.pylintArgs": [ "--generated-members=cv2.\*" ] 문장 추가



```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 디버그(D) 터미널(T) 도움말(H) settings.json - week1 - Visual Studio Code
탐색기 열려 있는 편집기 HelloCV.py settings.json
C: > Users > sunkyoo > AppData > Roaming > Code > User > settings.json > ...
1 {
2     "editor.mouseWheelZoom": true,
3     "python.pythonPath": "C:\\\\Users\\\\sunkyoo\\\\AppData\\\\Local\\\\",
4     "workbench.colorTheme": "Default Light",
5     "python.linting.pylintArgs": [
6         "--generated-members=cv2.*",
7     ]
8 }
```

# HELLOCV.PY 프로그램 만들기

## ■ vscode\_settings.json 설정의 예

```
{  
    "editor.fontFamily": "Consolas, D2Coding, 'Courier New', monospace",  
    "editor.fontSize": 20,  
    "editor.formatOnPaste": true,  
    "editor.minimap.enabled": false,  
    "editor.mouseWheelZoom": true,  
    "editor.wordWrap": "on",  
    "editor.wordWrapColumn": 120,  
    "explorer.confirmDelete": false,  
    "explorer.sortOrder": "type",  
    "python.pythonPath": "C:\\\\Users\\\\sunkyoo\\\\AppData\\\\Local\\\\Programs\\\\Python  
\\\\Python37\\\\python.exe",  
    "window.restoreWindows": "none",  
    "workbench.colorTheme": "Default Light+",  
    "workbench.editor.enablePreview": false  
}
```

# 1. OpenCV-Python 시작하기

5) 영상 파일 불러와서 출력하기

# HELLOCV.PY 프로그램 만들기

실습: HelloCV.py

- BMP 파일을 불러와서 출력하는 소스 코드 추가 입력

```
import sys
import cv2

print('Hello OpenCV', cv2.__version__)

img = cv2.imread('cat.bmp') ← cat.bmp 파일을 불러와 img 변수에 저장

if img is None: ← 영상 파일 불러오기가 실패하면 에러 메시지
    print('Image load failed!')
    sys.exit()

cv2.namedWindow('image')
cv2.imshow('image', img) ← "image"라는 이름의 새 창을 만들고,
cv2.waitKey()           이 창에 img 영상을 출력하고, 키
                        보드 입력이 있을 때까지 대기.

cv2.destroyAllWindows() ← 생성된 모든 창을 닫음
```

# 영상 불러와서 출력하기

## ■ VS Code에서 HelloCV.py 프로그램 실행 화면

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with a folder "CH01" containing files "cat.bmp" and "HelloCV.py".
- Code Editor:** Displays the Python script "HelloCV.py":

```
1 import sys
2 import cv2
3
4
5 print('Hello, OpenCV', cv2.__version__)
6
7 img = cv2.imread('cat.bmp')
8
9 if img is None:
10     print('Image load failed!')
11     sys.exit()
12
13 cv2.namedWindow('image')
14 cv2.imshow('image', img)
15 cv2.waitKey()
16
17 cv2.destroyAllWindows()
```

- Terminal:** Shows the command line output of the Python script execution:

```
PS C:\coding\python\opencv\ch01> cd 'c:\coding\python\opencv\ch01'; & 'C:\Users\sunkyoo\AppData\Local\Programs\Python\Python37\python.exe' 'c:\Users\sunkyoo\.vscode\extensions\ms-python.python-2020.5.80290\pythonFiles\lib\python\debugpy\wheels\debugpy\launcher' '8989' '--'
Hello, OpenCV 4.1.0
```

- Output Panel:** Shows the status message "Hello, OpenCV 4.1.0".
- Status Bar:** Shows "Python 3.7.7 64-bit" and other status indicators.

A separate window titled "image" displays a close-up photograph of a brown tabby cat's face, looking upwards.

# 1. OpenCV-Python 시작하기

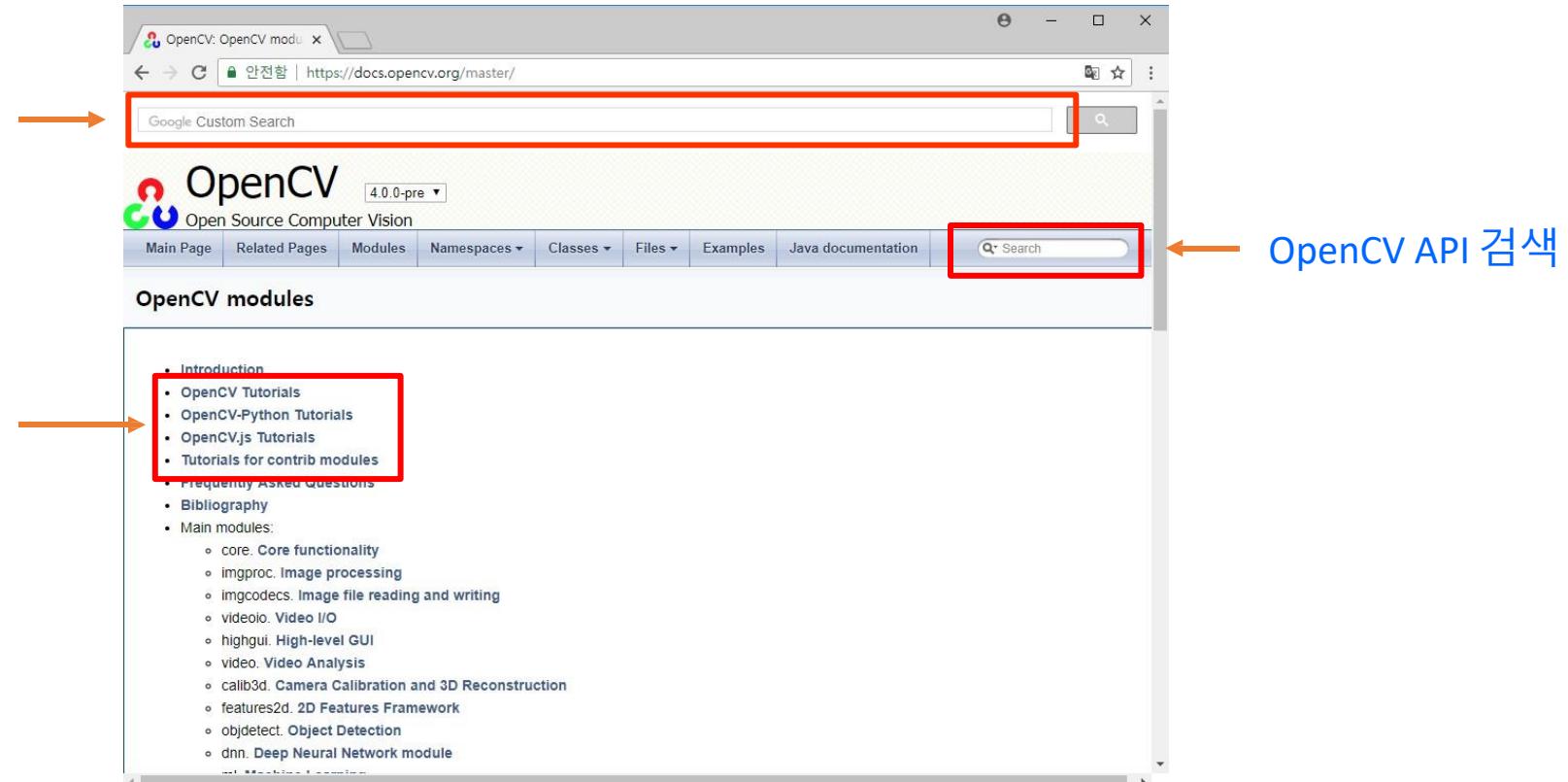
## 6) OpenCV 주요 함수 설명

# OPENCV 주요 함수 설명

## ■ OpenCV API 도움말 찾기

- OpenCV 최신 도움말: <http://docs.opencv.org/master/>
- OpenCV 도움말 사이트에서 우측 상단 검색창 활용

OpenCV 사이트  
내부에서 구글 검색



# OPENCV API

## ■ 영상 파일 불러오기

```
cv2.imread(filename, flags=None) -> retval
```

- filename: 불러올 영상 파일 이름 (문자열)
- flags: 영상 파일 불러오기 옵션 플래그

cv2.IMREAD_COLOR	BGR 컬러 영상으로 읽기 (기본값) <i>shape = (rows, cols, 3)</i>
cv2.IMREAD_GRAYSCALE	그레이스케일 영상으로 읽기 <i>shape = (rows, cols)</i>
cv2.IMREAD_UNCHANGED	영상 파일 속성 그대로 읽기 (e.g.) 투명한 PNG 파일: <i>shape = (rows, cols, 4)</i>

- retval: 불러온 영상 데이터 ([numpy.ndarray](#))

# OPENCV API

## ■ 영상 파일 저장하기

```
cv2.imwrite(filename, img, params=None) -> retval
```

- filename: 저장할 영상 파일 이름 (문자열)
- img: 저장할 영상 데이터 ([numpy.ndarray](#))
- params: 파일 저장 옵션 지정 (속성 & 값의 정수 쌍)  
e.g) [cv2.IMWRITE\_JPEG\_QUALITY, 90] : JPG 파일 압축률을 90%로 지정
- retval: 정상적으로 저장하면 True, 실패하면 False.

# OPENCV API

## ■ 새 창띄우기

```
cv2.namedWindow(winname, flags=None) -> None
```

- **winname:** 창 고유 이름 (문자열)
- **flags:** 창 속성 지정 플래그

cv2.WINDOW_NORMAL	영상 크기를 창 크기에 맞게 지정
cv2.WINDOW_AUTOSIZE	창 크기를 영상 크기에 맞게 변경 (기본값)

# OPENCV API

## ■ 창 닫기

```
cv2.destroyWindow(winname) -> None  
cv2.destroyAllWindows() -> None
```

- **winname:** 닫고자 하는 창 이름
- 참고사항
  - cv2.destroyWindow() 함수는 지정한 창 하나만 닫고, cv2.destroyAllWindows() 함수는 열려 있는 모든 창을 닫음.
  - 일반적인 경우 프로그램 종료 시 운영 체제에 의해 열려 있는 모든 창이 자동으로 닫힘

# OPENCV API

## ■ 창 위치 이동

```
cv2.moveWindow(winname, x, y) -> None
```

- **winname:** 창 이름
- **x, y:** 이동할 위치좌표

# OPENCV API

## ■ 창 크기 변경

```
cv2.resizeWindow(winname, width, height) -> None
```

- winname: 창 이름
- width: 변경할 창의 가로 크기
- height: 변경할 창의 세로 크기
- 참고 사항
  - 창 생성 시 cv2.WINDOW\_NORMAL 속성으로 생성되어야 동작
  - 영상 출력 부분의 크기만을 고려함 (제목 표시줄, 창 경계는 고려되지 않음)

# OPENCV API

## ■ 영상 출력하기

```
cv2.imshow(winname, mat) -> None
```

- winname: 영상을 출력할 대상 창 이름
- mat: 출력할 영상 데이터 ([numpy.ndarray](#))
- 참고 사항
  - uint16, int32 자료형 행렬의 경우, 행렬 원소 값을 255로 나눠서 출력
  - float32, float64 자료형 행렬의 경우, 행렬 원소 값에 255를 곱해서 출력
  - 만약 winname에 해당하는 창이 없으면 창을 새로 만들어서 영상을 출력함
  - Windows 운영체제에서는 Ctrl + C (복사), Ctrl + S (저장) 지원
  - 실제로는 cv2.waitKey() 함수를 호출해야 화면에 영상이 나타남

# OPENCV API

## ■ 키보드 입력 대기

```
cv2.waitKey(delay=None) -> retval
```

- delay: 밀리초 단위 대기 시간.  $delay \leq 0$  이면 무한히 기다림. 기본값은 0.
- retval: 눌린 키 값(ASCII code). 키가 눌리지 않으면 -1.
- 참고 사항
  - cv2.waitKey() 함수는 OpenCV 창이 하나라도 있을 때 동작함
  - 특정 키 입력을 확인하려면 ord() 함수를 이용

```
while True:  
    if cv2.waitKey() == ord('q'):  
        break
```

- 주요 특수키 코드: 27(ESC), 13(ENTER), 9(TAB)

# 1. OpenCV-Python 시작하기

7) Matplotlib 사용하여 영상 출력하기

# MATPLOTLIB을 이용한 영상 출력

## ■ Matplotlib 라이브러리

- 함수 그래프, 차트(chart), 히스토그램(histogram) 등의 다양한 그리기 기능을 제공하는 Python 패키지

```
> pip install matplotlib
```

## ■ 컬러 영상 출력

- 컬러 영상의 색상 정보가 **RGB 순서**이어야 함
- cv2.imread() 함수로 불러온 영상의 색상 정보는 BGR 순서이므로 이를 RGB 순서로 변경해야 함 ⑦ **cv2.cvtColor()** 함수 이용

## ■ 그레이스케일 영상 출력

- plt.imshow() 함수에서 컬러맵을 **cmap='gray'** 으로 지정

# MATPLOTLIB을 이용한 영상 출력

실습: matplotlib.py

- Matplotlib을 이용하여 영상 출력하기

```
import matplotlib.pyplot as plt
import cv2

# 컬러 영상 출력
imgBGR = cv2.imread('cat.bmp')
imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)

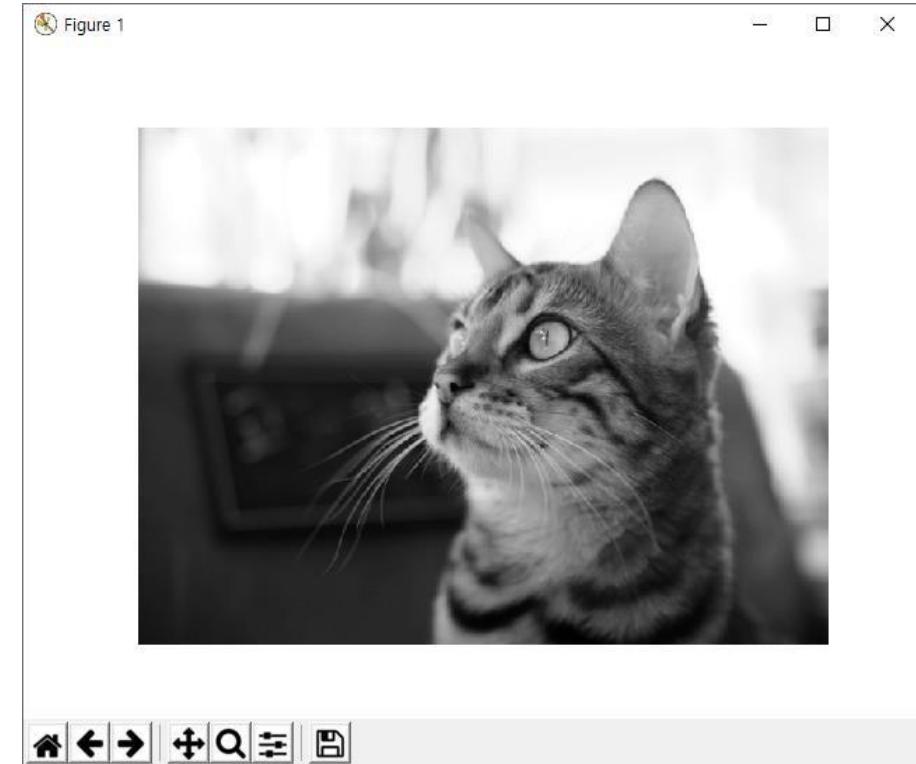
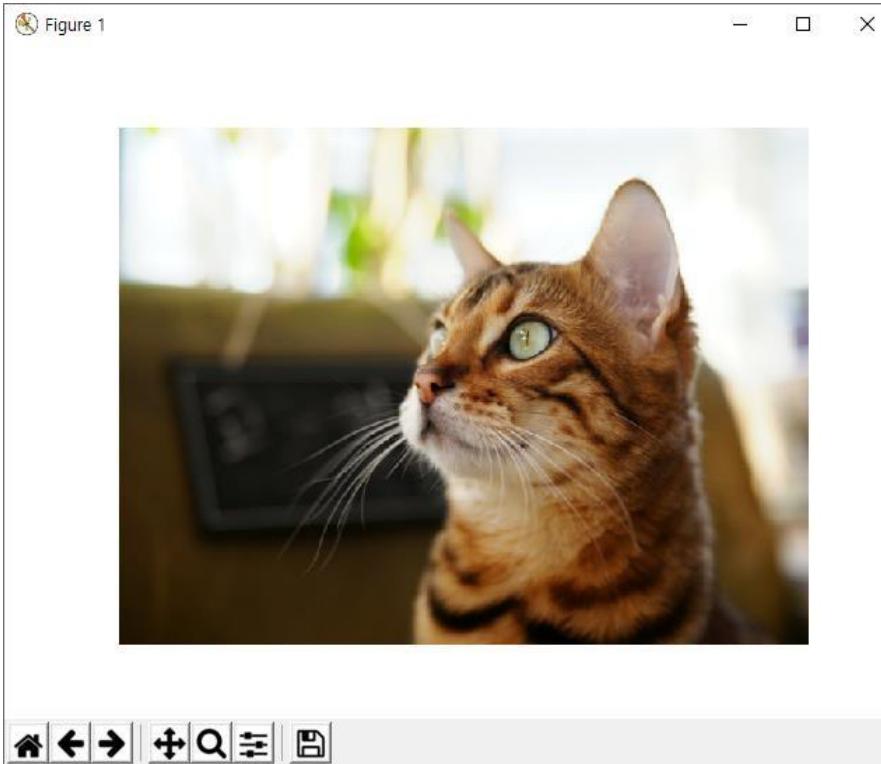
plt.axis('off')
plt.imshow(imgRGB)
plt.show()

# 그레이스케일 영상 출력
imgGray = cv2.imread('cat.bmp', cv2.IMREAD_GRAYSCALE)

plt.axis('off')
plt.imshow(imgGray, cmap='gray')
plt.show()
```

# MATPLOTLIB을 이용한 영상 출력

- Matplotlib을 이용하여 영상 출력하기



# MATPLOTLIB을 이용한 영상 출력

- Matplotlib을 이용하여 창 하나에 여러 개의 이미지 출력하기

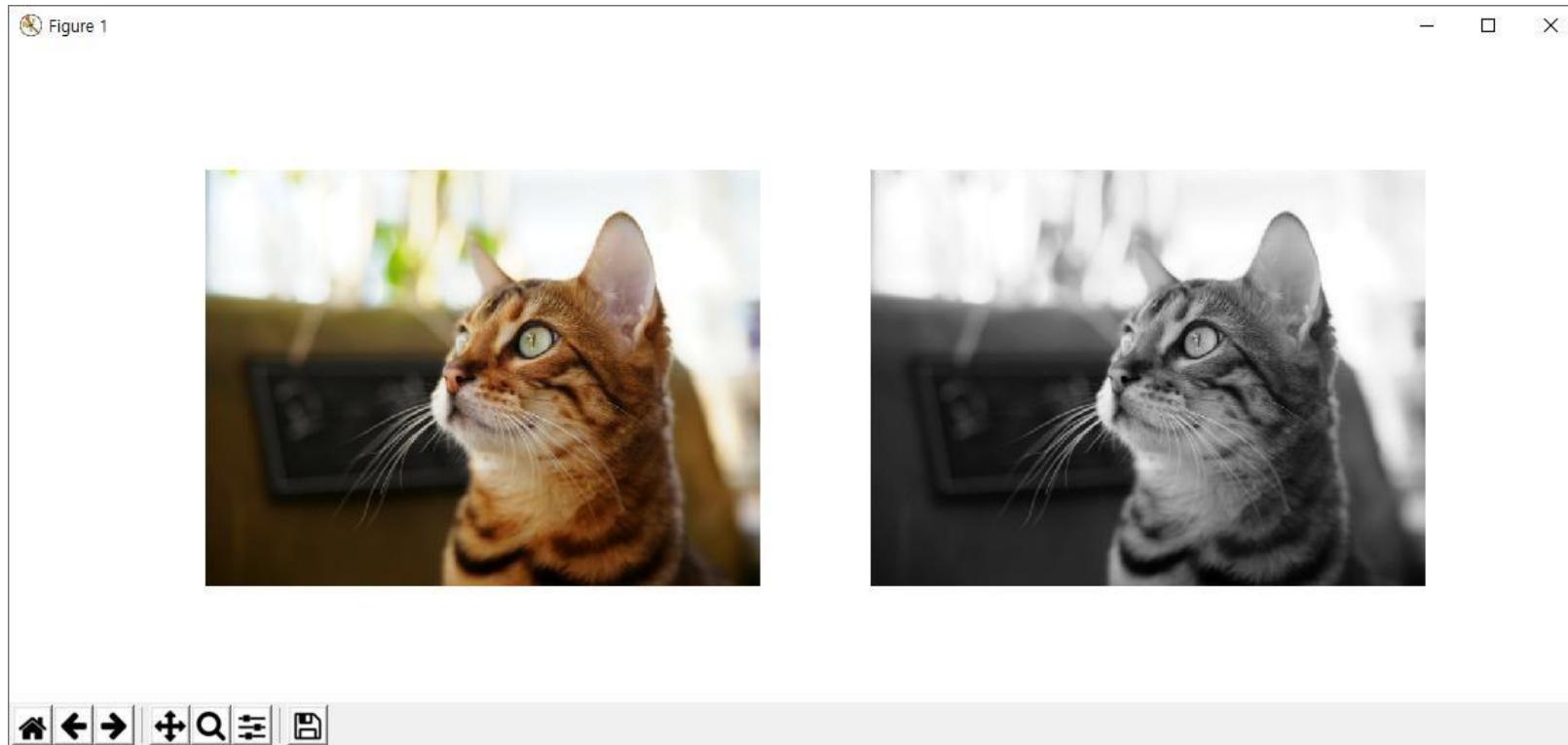
```
import matplotlib.pyplot as plt
import cv2

# 컬러 영상 & 그레이스케일 영상 불러오기
imgBGR = cv2.imread('cat.bmp')
imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
imgGray = cv2.imread('cat.bmp', cv2.IMREAD_GRAYSCALE)

# 두 개의 영상을 함께 출력
plt.subplot(121), plt.axis('off'), plt.imshow(imgRGB)
plt.subplot(122), plt.axis('off'), plt.imshow(imgGray, cmap='gray')
plt.show()
```

# MATPLOTLIB을 이용한 영상 출력

- Matplotlib을 이용하여 창 하나에 여러 개의 이미지 출력하기



# MATPLOTLIB을 이용한 영상 출력

- Jupyter Notebook에서 Matplotlib으로 영상 출력하기

The screenshot shows a Jupyter Notebook interface. The title bar says "jupyter - Jupyter Notebook". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Run, and Code buttons. The menu bar has File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The notebook area contains two code cells:

```
In [1]: import cv2
import matplotlib.pyplot as plt

In [2]: img = cv2.imread('cat.bmp')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

Cell In [2] displays an image of a brown tabby cat looking upwards. The image has axes ranging from 0 to 400 on the y-axis and 0 to 600 on the x-axis.

# 1. OpenCV-Python 시작하기

8) 실전 코딩: 이미지 슬라이드쇼

# [실전 코딩] 이미지 슬라이드쇼

실습: SlideShow.py

- 이미지 슬라이드쇼
  - 특정 폴더에 있는 모든 이미지 파일을 이용하여 슬라이드쇼를 수행
- 구현 할 기능
  - 특정 폴더에 있는 이미지 파일 목록 읽기
  - 이미지를 전체 화면으로 출력하기
  - 일정 시간동안 이미지를 화면에 출력하고, 다음 이미지로 교체하기 (무한루프)

# [실전 코딩] 이미지 슬라이드쇼

- 특정 폴더에 있는 이미지 파일(\*.jpg) 목록 읽기

- os.listdir()

```
import os

file_list = os.listdir('.\\images')
img_files = [file for file in file_list if file.endswith('.jpg')]
```

- glob.glob()

```
import glob

img_files = glob.glob('.\\images\\*.jpg')
```

# [실전 코딩] 이미지 슬라이드쇼

## ■ 전체 화면 영상 출력 창 만들기

- 먼저 cv2.WINDOW\_NORMAL 속성의 창을 만든 후, cv2.setWindowProperty() 함수를 사용하여 전체 화면 속성으로 변경

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.setWindowProperty('image', cv2.WND_PROP_FULLSCREEN,
                      cv2.WINDOW_FULLSCREEN)
```

# [실전 코딩] 이미지 슬라이드쇼

- 불러온 영상을 반복적으로 출력하기

```
cnt = len(img_files)
idx = 0

while True:
    img = cv2.imread(img_files[idx])

    if img is None:
        print('Image load failed!')
        break

    cv2.imshow('image', img)
    if cv2.waitKey(1000) >= 0:
        break

    idx += 1
    if idx >= cnt:
        idx = 0
```

## 2. OpenCV-Python 기초 사용법

1) 영상의 속성과 픽셀 값 참조

# 영상의 속성과 픽셀 값 참조

- OpenCV는 영상 데이터를 **numpy.ndarray**로 표현

```
import cv2

img1 = cv2.imread('cat.bmp', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('cat.bmp', cv2.IMREAD_COLOR)
```

numpy.ndarray

- ndim: 차원 수. `len(img.shape)`과 같음.
- shape: 각 차원의 크기. (h, w) 또는 (h, w, 3)
  - 컬러 영상
  - 그레이스케일 영상
- size: 전체 원소 개수
- dtype: 원소의 데이터 타입. 영상 데이터는 `uint8`.

# 영상의 속성과 픽셀 값 참조

## ■ OpenCV 영상 데이터 자료형과 NumPy 자료형

OpenCV 자료형 (1채널)	NumPy 자료형	구분
cv2.CV_8U	numpy.uint8	8비트 부호없는 정수
cv2.CV_8S	numpy.int8	8비트 부호있는 정수
cv2.CV_16U	numpy.uint16	16비트 부호없는 정수
cv2.CV_16S	numpy.int16	16비트 부호있는 정수
cv2.CV_32S	numpy.int32	32비트 부호있는 정수
cv2.CV_32F	numpy.float32	32비트 부동소수형
cv2.CV_64F	numpy.float64	64비트 부동소수형
cv2.CV_16F	numpy.float16	16비트 부동소수형

- 그레이스케일 영상: `cv2.CV_8UC1` ⑦ `numpy.uint8, shape = (h, w)`
- 컬러 영상: `cv2.CV_8UC3` ⑦ `numpy.uint8, shape = (h, w, 3)`

# 영상의 속성과 픽셀 값 참조

실습: img\_info.py

## ■ 영상의 속성 참조 예제

```
img1 = cv2.imread('cat.bmp', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('cat.bmp', cv2.IMREAD_COLOR)

print('type(img1):', type(img1)) # type(img1): <class 'numpy.ndarray'>
print('img1.shape:', img1.shape) # img1.shape: (480, 640)
print('img2.shape:', img2.shape) # img2.shape: (480, 640, 3)
print('img2.dtype:', img2.dtype) # img2.dtype: uint8

h, w = img2.shape[:2] # h: 480, w: 640
print('img2 size: {} x {}'.format(w, h))

if len(img1.shape) == 2:
    print('img1 is a grayscale image')
elif len(img1.shape) == 3:
    print('img1 is a truecolor image')
```

# 영상의 속성과 픽셀 값 참조

실습: img\_info.py

## ■ 영상의 픽셀 값 참조 예제

```
img1 = cv2.imread('cat.bmp', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('cat.bmp', cv2.IMREAD_COLOR)

for y in range(h):
    for x in range(w):
        img1[y, x] = 255
        img2[y, x] = [0, 0, 255]

# img1[:, :] = 255
# img2[:, :] = (0, 0, 255)
```

} for문으로 픽셀 값을 변경하는 작업은 매우 느리므로,  
픽셀 값 참조 방법만 확인하고 실제로는 사용 금지

## 2. OpenCV-Python 기초 사용법

2) 영상의 생성, 복사, 부분 영상 추출

# 영상의 생성, 복사, 부분 영상 추출

## ■ 지정한 크기로 새 영상 생성하기

```
numpy.empty(shape, dtype=float, ...) -> arr  
numpy.zeros(shape, dtype=float, ...) -> arr  
numpy.ones(shape, dtype=None, ...) -> arr  
numpy.full(shape, fill_value, dtype=None, ...) -> arr
```

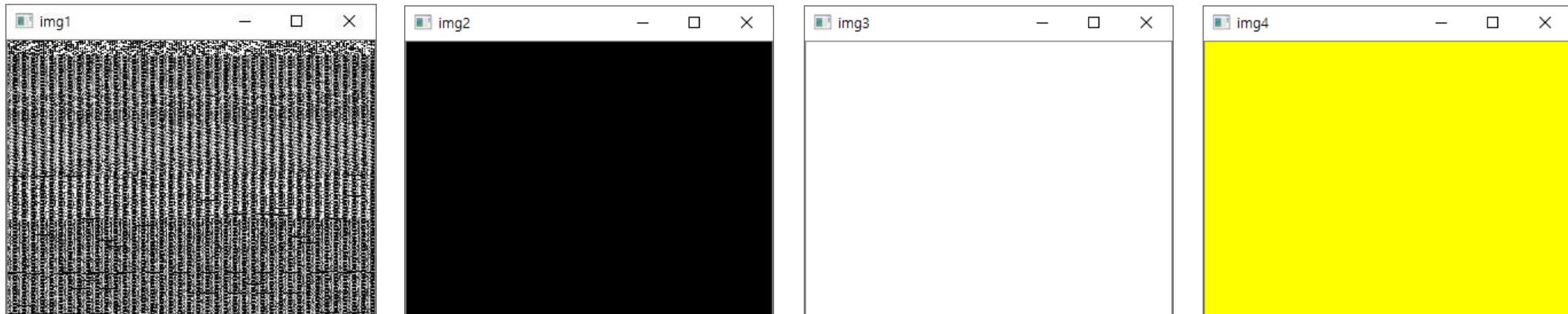
- shape: 각 차원의 크기. ([h, w](#)) 또는 ([h, w, 3](#))
- dtype: 원소의 데이터 타입. 일반적인 영상이면 [numpy.uint8](#) 지정
- arr: 생성된 영상([numpy.ndarray](#))
- 참고사항:
  - [numpy.empty\(\)](#) 함수는 임의의 값으로 초기화된 배열을 생성
  - [numpy.zeros\(\)](#) 함수는 0으로 초기화된 배열을 생성
  - [numpy.ones\(\)](#) 함수는 1로 초기화된 배열을 생성
  - [numpy.full\(\)](#) 함수는 [fill\\_value](#)로 초기화된 배열을 생성

# 영상의 생성, 복사, 부분 영상 추출

실습: img\_ops.py

## ■ 영상의 생성 예제 코드

```
img1 = np.empty((480, 640), dtype=np.uint8)      # grayscale image
img2 = np.zeros((480, 640, 3), dtype=np.uint8)    # color image
img3 = np.ones((480, 640), dtype=np.uint8) * 255 # white
img4 = np.full((480, 640, 3), (0, 255, 255), dtype=np.uint8) # yellow
```

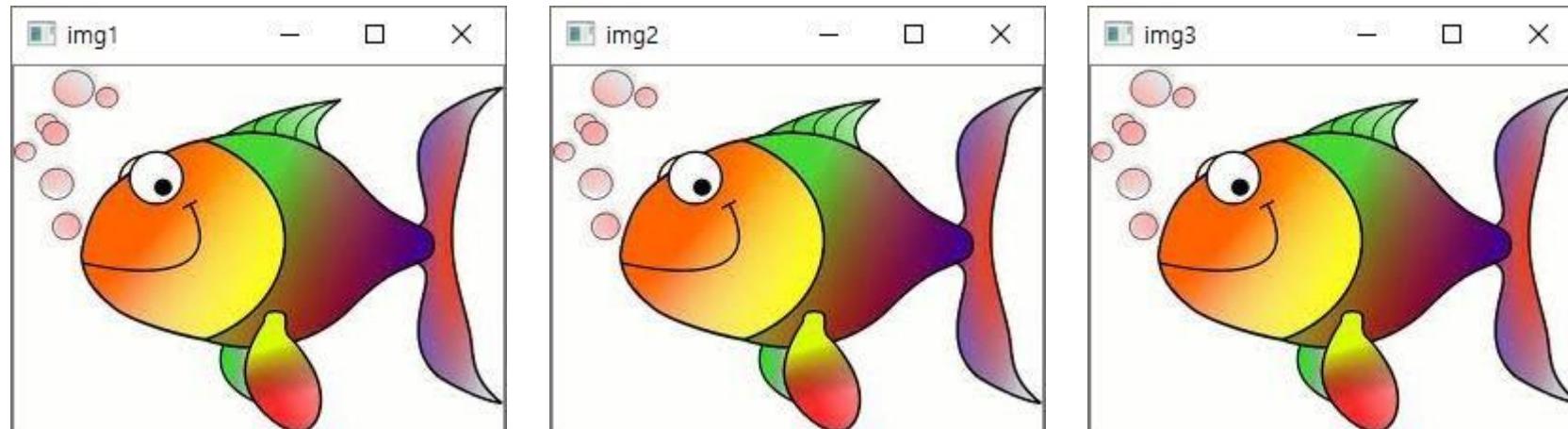


# 영상의 생성, 복사, 부분 영상 추출

실습: img\_ops.py

- 영상의 참조 및 복사 예제 코드

```
img1 = cv2.imread('HappyFish.jpg')  
  
img2 = img1  
img3 = img1.copy()
```



# 영상의 생성, 복사, 부분 영상 추출

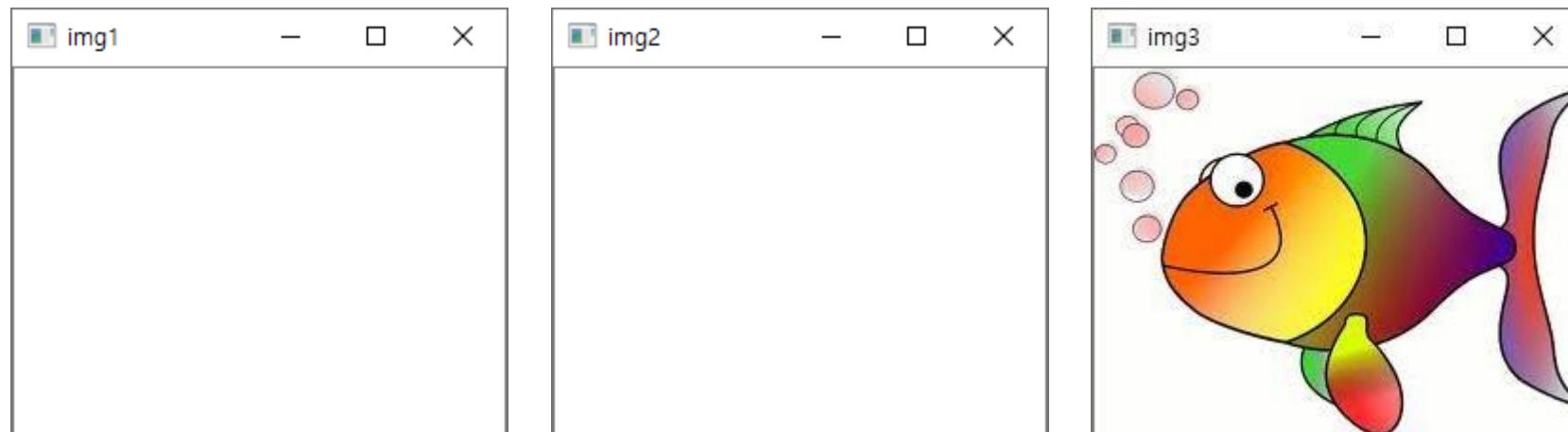
실습: img\_ops.py

- 영상의 참조 및 복사 예제 코드

```
img1 = cv2.imread('HappyFish.jpg')

img2 = img1
img3 = img1.copy()

img1.fill(255)
```



# 영상의 생성, 복사, 부분 영상 추출

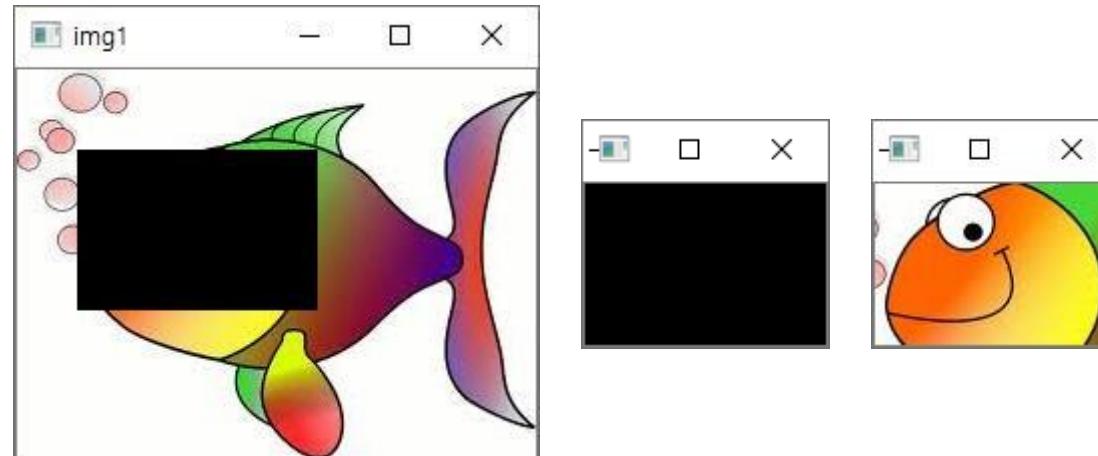
실습: img\_ops.py

## ■ 부분 영상 추출

```
img1 = cv2.imread('HappyFish.jpg')

img2 = img1[40:120, 30:150] # numpy.ndarray의 슬라이싱
img3 = img1[40:120, 30:150].copy()

img2.fill(0)
```



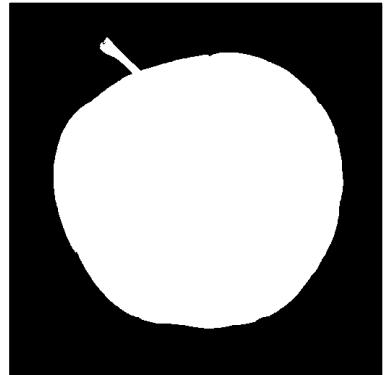
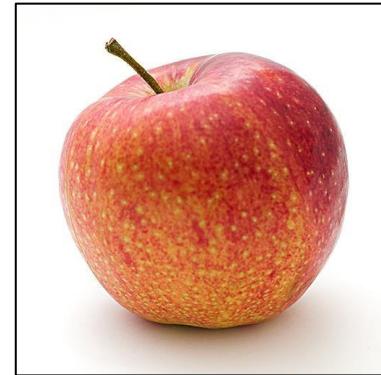
## 2. OpenCV-Python 기초 사용법

### 3) 마스크 연산과 ROI

# 마스크 연산과 ROI

## ■ ROI

- Region of Interest, 관심 영역
- 영상에서 특정 연산을 수행하고자 하는  
임의의 부분영역



## ■ 마스크 연산

- OpenCV는 일부 함수에 대해 ROI 연산을 지원하며, 이때 **마스크 영상**을 인자로 함께 전달해야 함  
(e.g.) `cv2.copyTo()`, `cv2.calcHist()`, `cv2.bitwise_or()`, `cv2.matchTemplate()`, etc.
- 마스크 영상은 `cv2.CV_8UC1` 타입(그레이스케일 영상)
- 마스크 영상의 픽셀 값이 0이 아닌 위치에서만 연산이 수행됨  
**⑦ 보통 마스크 영상으로는 0 또는 255로 구성된 이진 영상(binary image)을 사용**

# 마스크 연산과 ROI

## ■ 마스크 연산을 지원하는 픽셀 값 복사 함수

```
cv2.copyTo(src, mask, dst=None) -> dst
```

- src: 입력 영상
- mask: 마스크 영상. cv2.CV\_8U. (numpy.uint8)  
0이 아닌 픽셀에 대해서만 복사 연산을 수행.
- dst: 출력 영상. 만약 src와 크기 및 타입이 같은 dst를 입력으로 지정하면  
dst를 새로 생성하지 않고 연산을 수행.  
그렇지 않으면 dst를 새로 생성하여 연산을 수행한 후 반환함.

# 마스크 연산과 ROI

실습: mask\_op.py

## ■ 마스크 연산 예제

```
src = cv2.imread('airplane.bmp', cv2.IMREAD_COLOR)
mask = cv2.imread('mask_plane.bmp', cv2.IMREAD_GRAYSCALE)
dst = cv2.imread('field.bmp', cv2.IMREAD_COLOR)
```

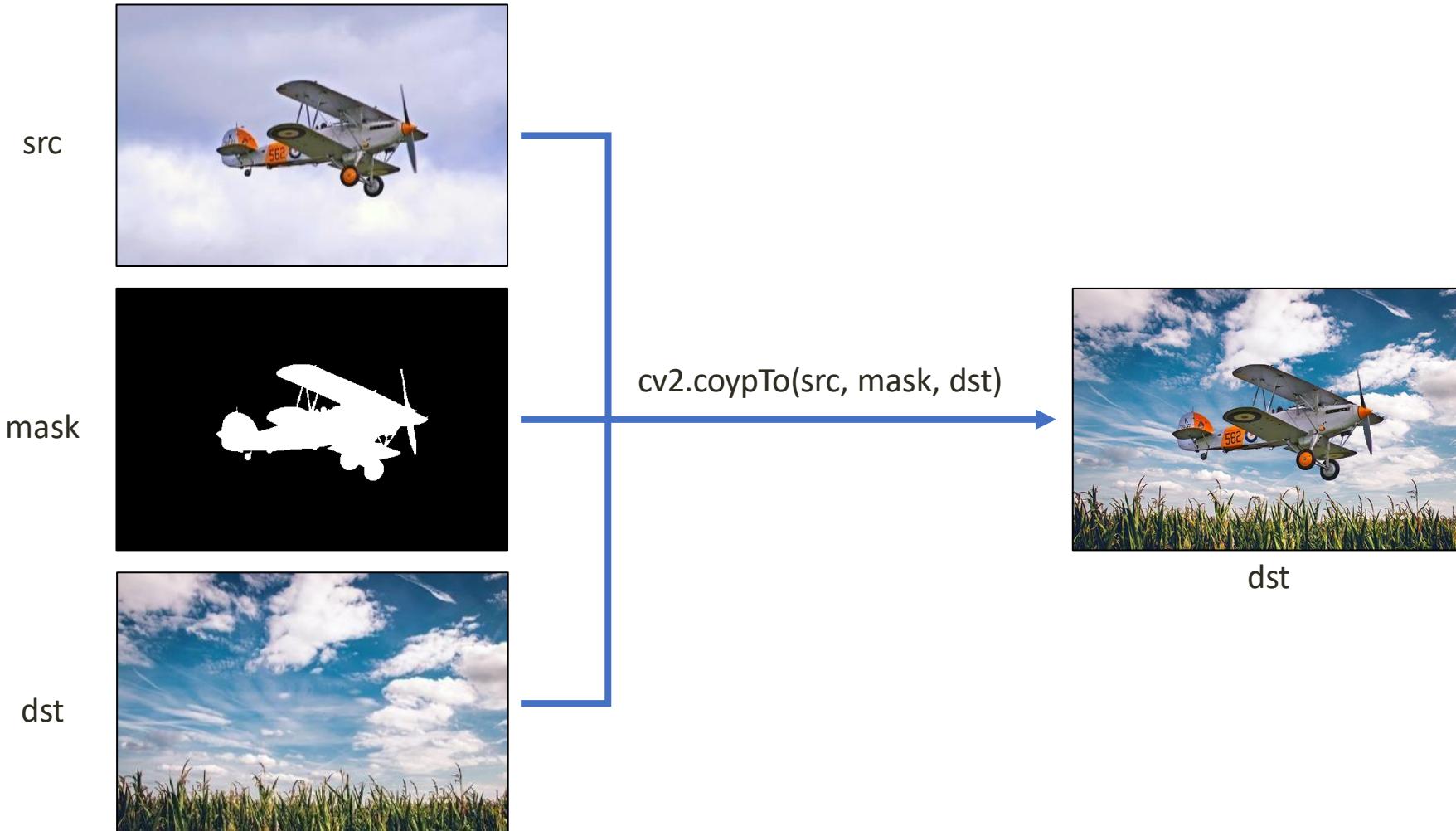
```
cv2.copyTo(src, mask, dst)
```

src, mask, dst는 모두 크기가 같아야 함.  
src와 dst는 같은 타입이어야 하고, mask는 그레이스케일 타입의 이진 영상.

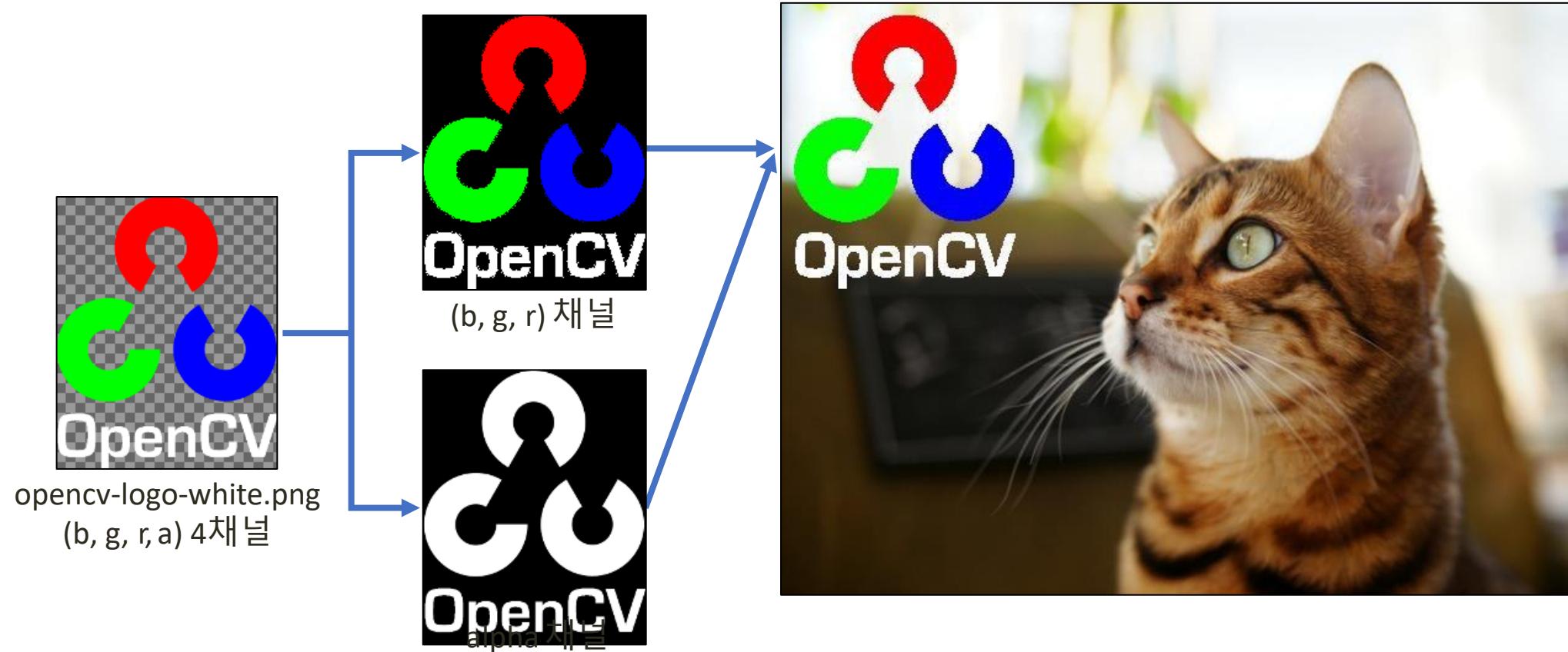
- NumPy의 불리언 인덱싱(Boolean indexing)을 이용한 마스크 연산

```
dst[mask > 0] = src[mask > 0]
```

# 마스크 연산과 ROI



# 마스크 연산과 ROI



## 2. OpenCV-Python 기초 사용법

### 4) OpenCV 그리기 함수

# OPENCV 그리기 함수

## ■ OpenCV 그리기 함수

- OpenCV는 영상에 선, 도형, 문자열을 출력하는 그리기 함수를 제공
- 선 그리기: 직선, 화살표, 마커 등
- 도형 그리기: 사각형, 원, 타원, 다각형 등
- 문자열 출력

## ■ 그리기 함수 사용 시 주의할 점

- 그리기 알고리즘을 이용하여 영상의 픽셀 값 자체를 변경  
**⑦** 원본 영상이 필요하면 복사본을 만들어서 그리기 & 출력
- 그레이스케일 영상에는 컬러로 그리기 안 됨  
**⑦** cv2.cvtColor() 함수로 BGR 컬러 영상으로 변환한 후 그리기 함수 호출

# OPENCV 그리기 함수

## ■ 직선 그리기

```
cv2.line(img, pt1, pt2, color, thickness=None, lineType=None,  
        shift=None) -> img
```

- img: 그림을 그릴 영상
- pt1, pt2: 직선의 시작점과 끝점. (x, y) 튜플.
- color: 선 색상 또는 밝기. (B, G, R) 튜플 또는 정수값.
- thickness: 선 두께. 기본값은 1.
- lineType: 선 타입. cv2.LINE\_4, cv2.LINE\_8, cv2.LINE\_AA 중 선택.  
 기본값은 cv2.LINE\_8
- shift: 그리기 좌표 값의 축소 비율. 기본값은 0.

# OPENCV 그리기 함수

## ■ 사각형 그리기

```
cv2.rectangle(img, pt1, pt2, color, thickness=None, lineType=None,  
             shift=None) -> img  
cv2.rectangle(img, rec, color, thickness=None, lineType=None,  
             shift=None) -> img
```

- img: 그림을 그릴 영상
- pt1, pt2: 사각형의 두 꼭지점 좌표. (x, y) 튜플.
- rec: 사각형 위치 정보. (x, y, w, h) 튜플.
- color: 선 색상 또는 밝기. (B, G, R) 튜플 또는 정수값.
- thickness: 선 두께. 기본값은 1. 음수(-1)를 지정하면 내부를 채움.
- lineType: 선 타입. cv2.LINE\_4, cv2.LINE\_8, cv2.LINE\_AA 중 선택.  
기본값은 cv2.LINE\_8
- shift: 그리기 좌표 값의 축소 비율. 기본값은 0.

# OPENCV 그리기 함수

## ■ 원 그리기

```
cv2.circle(img, center, radius, color, thickness=None, lineType=None,  
          shift=None) -> img
```

- img: 그림을 그릴 영상
- center: 원의 중심 좌표. (x, y) 튜플.
- radius: 원의 반지름
- color: 선 색상 또는 밝기. (B, G, R) 튜플 또는 정수값.
- thickness: 선 두께. 기본값은 1. 음수(-1)를 지정하면 내부를 채움.
- lineType: 선 타입. cv2.LINE\_4, cv2.LINE\_8, cv2.LINE\_AA 중 선택.  
기본값은 cv2.LINE\_8
- shift: 그리기 좌표 값의 축소 비율. 기본값은 0.

# OPENCV 그리기 함수

## ■ 다각형 그리기

```
cv2.polyline(img, pts, isClosed, color, thickness=None, lineType=None,  
            shift=None) -> img
```

- img: 그림을 그릴 영상
- pts: 다각형 외곽 점들의 좌표 배열. [numpy.ndarray의 리스트](#).  
(e.g.) [np.array([[10, 10], [50, 50], [10, 50]]), dtype=np.int32)]
- isClosed: 폐곡선 여부. True 또는 False 지정.
- color: 선 색상 또는 밝기. (B, G, R) 튜플 또는 정수값.
- thickness: 선 두께. 기본값은 1. 음수(-1)를 지정하면 내부를 채움.
- lineType: 선 타입. cv2.LINE\_4, cv2.LINE\_8, cv2.LINE\_AA 중 선택.  
기본값은 cv2.LINE\_8
- shift: 그리기 좌표 값의 축소 비율. 기본값은 0.

# OPENCV 그리기 함수

## ■ 문자열 출력

```
cv2.putText(img, text, org, fontFace, fontScale, color, thickness=None,  
           lineType=None, bottomLeftOrigin=None) -> img
```

- img: 그림을 그릴 영상
- text: 출력할 문자열
- org: 영상에서 문자열을 출력할 위치의 좌측 하단 좌표. (x, y) 튜플.
- fontFace: 폰트 종류. `cv2.FONT_HERSHEY_SIMPLEX`로 시작하는 상수 중 선택
- fontScale: 폰트 크기 확대/축소 비율
- color: 선 색상 또는 밝기. (B, G, R) 튜플 또는 정수값.
- thickness: 선 두께. 기본값은 1. 음수(-1)를 지정하면 내부를 채움.
- lineType: 선 타입. `cv2.LINE_4`, `cv2.LINE_8`, `cv2.LINE_AA` 중 선택.
- bottomLeftOrigin: True이면 영상의 좌측 하단을 원점으로 간주. 기본값은 False.

# OPENCV 그리기 함수

- cv2.putText()에서 지원하는 fontFace 상수와 실제 출력 모양

FONT\_HERSHEY\_SIMPLEX

FONT\_HERSHEY\_PLAIN

FONT\_HERSHEY\_DUPLEX

FONT\_HERSHEY\_COMPLEX

FONT\_HERSHEY\_TRIPLEX

FONT\_HERSHEY\_COMPLEX\_SMALL

FONT\_HERSHEY\_SCRIPT\_SIMPLEX

FONT\_HERSHEY\_SCRIPT\_COMPLEX

FONT\_HERSHEY\_COMPLEX | FONT\_ITALIC

# OPENCV 그리기 함수

실습: drawing.py

## ■ 다양한 그리기 함수 실행 예제

```
img = np.full((400, 400, 3), 255, np.uint8)

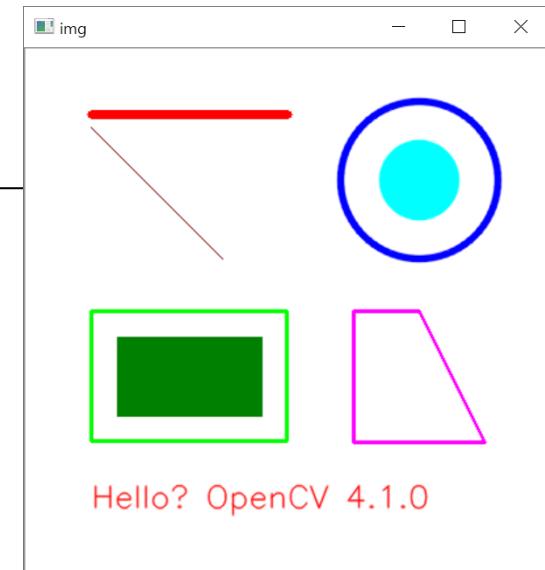
cv2.line(img, (50, 50), (200, 50), (0, 0, 255), 5)
cv2.line(img, (50, 60), (150, 160), (0, 0, 128))

cv2.rectangle(img, (50, 200, 150, 100), (0, 255, 0), 2)
cv2.rectangle(img, (70, 220), (180, 280), (0, 128, 0), -1)

cv2.circle(img, (300, 100), 60, (255, 0, 0), 3, cv2.LINE_AA)
cv2.circle(img, (300, 100), 30, (255, 255, 0), -1, cv2.LINE_AA)

pts = np.array([[250, 200], [300, 200], [350, 300], [250, 300]])
cv2.polylines(img, [pts], True, (255, 0, 255), 2)

text = 'Hello? OpenCV ' + cv2.__version__
cv2.putText(img, text, (50, 350), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
            (0, 0, 255), 1, cv2.LINE_AA)
```



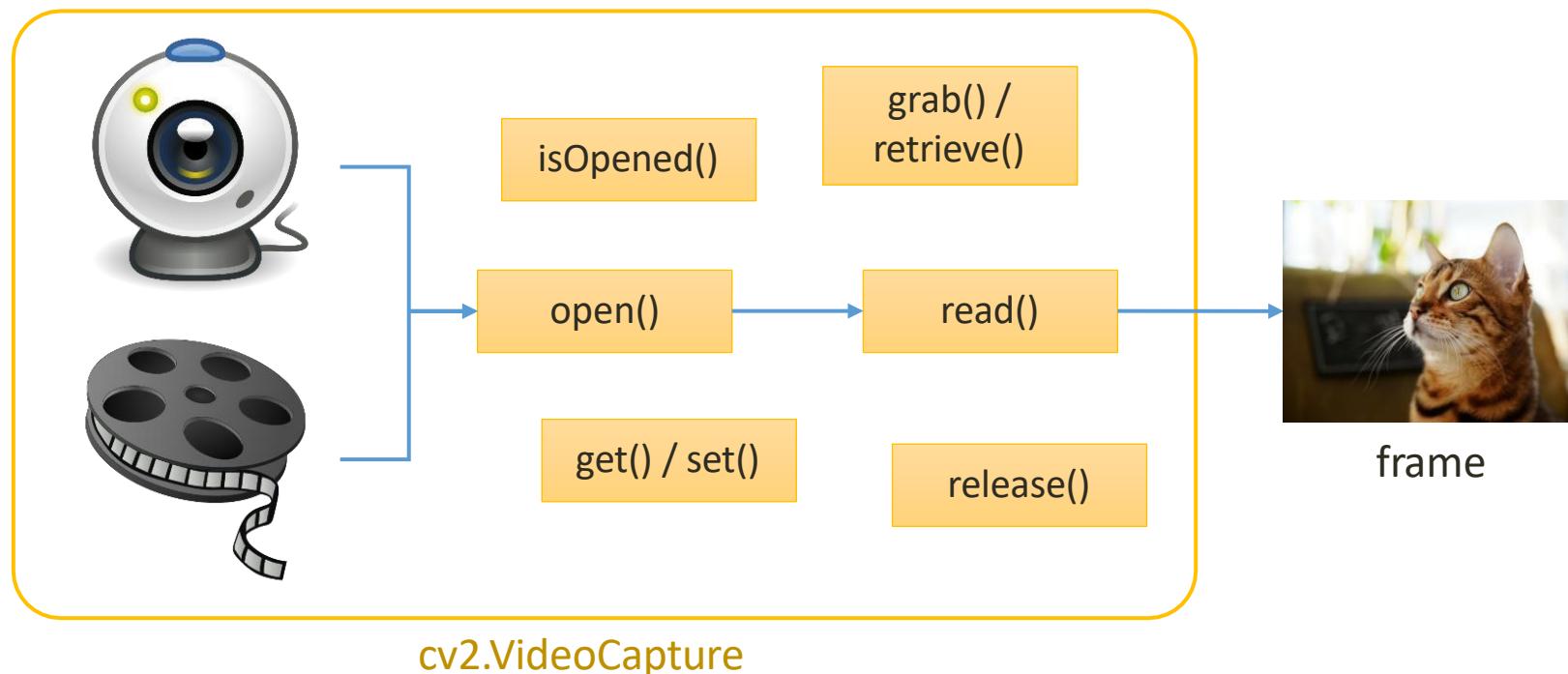
## 2. OpenCV-Python 기초 사용법

### 5) 카메라와 동영상 처리하기 1

# 카메라와 동영상 처리하기 1

## ■ cv2.VideoCapture 클래스

- OpenCV에서는 카메라와 동영상으로부터 프레임(frame)을 받아오는 작업을 cv2.VideoCapture 클래스 하나로 처리함



# 카메라와 동영상 처리하기 1

## ■ 카메라 열기

```
cv2.VideoCapture(index, apiPreference=None) -> retval
```

- index: camera\_id + domain\_offset\_id  
시스템 기본 카메라를 기본 방법으로 열려면 index에 0을 전달
- apiPreference: 선호하는 카메라 처리 방법을 지정
- retval: cv2.VideoCapture 객체

```
cv2.VideoCapture.open(index, apiPreference=None) -> retval
```

- retval: 성공하면 True, 실패하면 False.

# 카메라와 동영상 처리하기 1

## ■ 동영상, 정지 영상 시퀀스, 비디오 스트림 열기

```
cv2.VideoCapture(filename, apiPreference=None) -> retval
```

- filename: 비디오 파일 이름, 정지 영상 시퀀스, 비디오 스트림 URL 등  
(e.g) 'video.avi', 'img\_%02d.jpg', 'protocol://host:port/script?params|auth'
- apiPreference: 선호하는 동영상 처리 방법을 지정
- retval: cv2.VideoCapture 객체

```
cv2.VideoCapture.open(filename, apiPreference=None) -> retval
```

- retval: 성공하면 True, 실패하면 False.

# 카메라와 동영상 처리하기 1

## ■ 비디오 캡쳐가 준비되었는지 확인

```
cv2.VideoCapture.isOpened() -> retval
```

- retval: 성공하면 True, 실패하면 False.

## ■ 프레임 받아오기

```
cv2.VideoCapture.read(image=None) -> retval, image
```

- retval: 성공하면 True, 실패하면 False.
- image: 현재 프레임 (numpy.ndarray)

# 카메라와 동영상 처리하기 1

## ■ 카메라, 비디오 장치 속성 값 참조

```
cv2.VideoCapture.get(propId) -> retval
```

- propId: 속성 상수. ([OpenCV 문서](#) 참조)

CAP_PROP_FRAME_WIDTH	프레임 가로크기
CAP_PROP_FRAME_HEIGHT	프레임 세로크기
CAP_PROP_FPS	초당 프레임수
CAP_PROP_FRAME_COUNT	비디오 파일의 총 프레임 수
CAP_PROP_POS_MSEC	밀리초 단위로 현재 위치
CAP_PROP_POS_FRAMES	현재 프레임 번호
CAP_PROP_EXPOSURE	노출

- retval: 성공하면 해당 속성 값, 실패하면 0.

# 카메라와 동영상 처리하기 1

## ■ 카메라, 비디오 장치 속성 값 참조

```
cv2.VideoCapture.set(propId, value) -> retval
```

- propId: 속성 상수
- value: 속성 값
- retval: 성공하면 True, 실패하면 False.

# 카메라와 동영상 처리하기 1

실습: camera\_in.py

## ■ 카메라 처리 예제

```
import sys
import cv2

cap = cv2.VideoCapture(0)           ← 기본 카메라 장치 열기.

while True:
    ret, frame = cap.read()        ← 카메라로부터 프레임을 정상적으로 받아오면
                                    ret에는 True, frame에는 해당 프레임이 저장됨.

    inversed = ~frame            ← 현재 프레임 반전

    cv2.imshow('frame', frame)
    cv2.imshow('inversed', inversed)

    if cv2.waitKey(10) == 27:      ← 일정 시간(e.g. 10ms) 기다린 후 다음 프레임 처리.
                                만약 ESC 키를 누르면 while 루프 종료.
        break

cap.release()
cv2.destroyAllWindows()
```

# 카메라와 동영상 처리하기 1

실습: camera\_in.py

## ■ 카메라 처리 예제

```
import sys
import cv2

cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    inversed = ~frame

    cv2.imshow('frame', frame)
    cv2.imshow('inversed', inversed)

    if cv2.waitKey(10) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

The diagram illustrates the flow of control in the Python script. It starts with the line `cap = cv2.VideoCapture(0)`. A blue arrow points from this line to the first conditional block, which contains the code: `if not cap.isOpened():`, `print("Camera open failed!")`, and `exit()`. Another blue arrow points from the line `ret, frame = cap.read()` to the second conditional block, which contains the code: `if not ret:` and `break`.

# 카메라와 동영상 처리하기 1

실습: video\_in.py

## ■ 동영상 처리 예제

```
cap = cv2.VideoCapture('video1.mp4')

fps = round(cap.get(cv2.CAP_PROP_FPS))
delay = round(1000 / fps)

while True:
    ret, frame = cap.read()

    inversed = ~frame

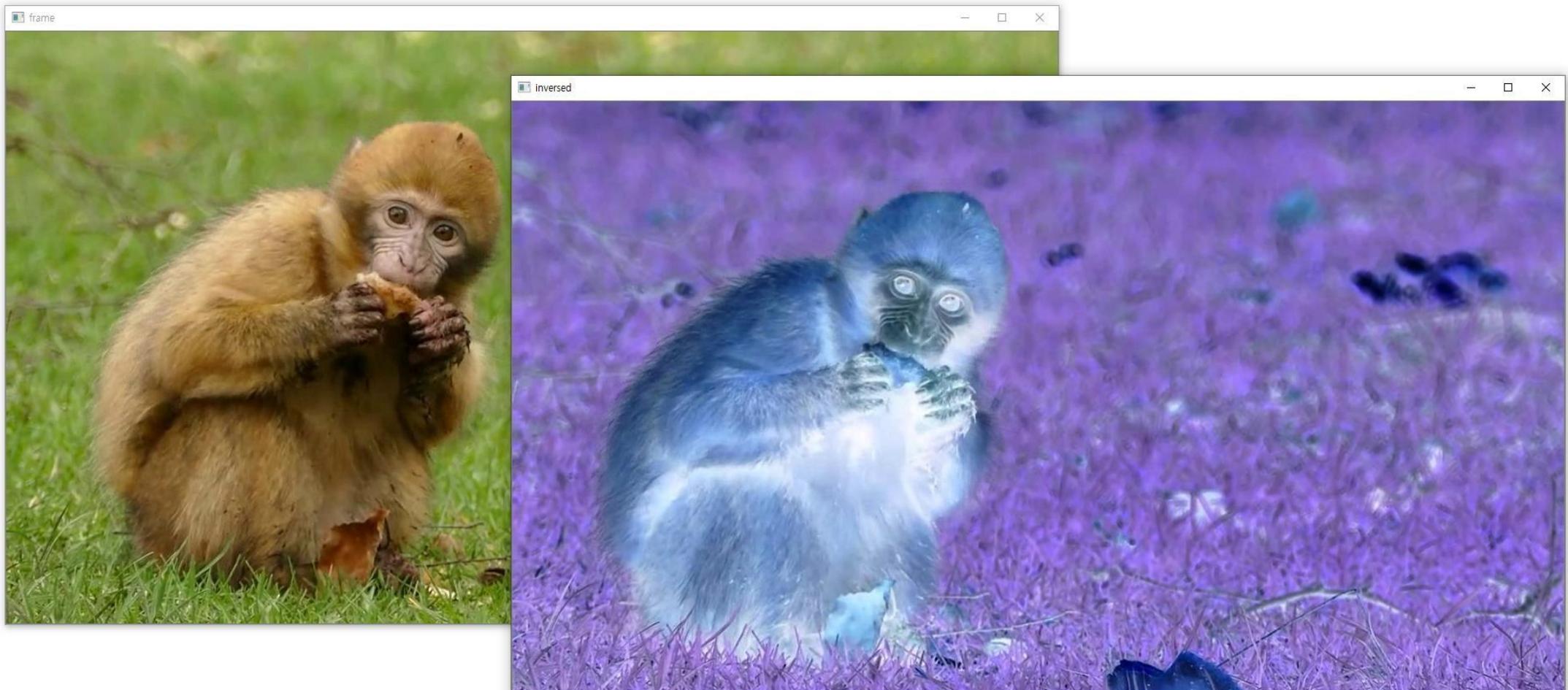
    cv2.imshow('frame', frame)
    cv2.imshow('inversed', inversed)

    if cv2.waitKey(delay) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

# 카메라와 동영상 처리하기 1

## ■ 동영상 처리 예제 실행 결과



## 2. OpenCV-Python 기초 사용법

### 5) 카메라와 동영상 처리하기 2

# 카메라와 동영상 처리하기 2

## ■ cv2.VideoWriter 클래스

- OpenCV에서는 cv2.VideoWriter 클래스를 이용하여 일련의 프레임을 동영상 파일로 저장할 수 있음
- 일련의 프레임은 모두 크기와 데이터 타입이 같아야 함

## ■ Fourcc (4-문자 코드, four charactercode)

- 동영상 파일의 코덱, 압축 방식, 색상, 픽셀 포맷 등을 정의하는 정수 값

cv2.VideoWriter_fourcc(*'DIVX')	DIVX MPEG-4 코덱
cv2.VideoWriter_fourcc(*'XVID')	XVID MPEG-4 코덱
cv2.VideoWriter_fourcc(*'FMP4')	FFMPEG MPEG-4 코덱
cv2.VideoWriter_fourcc(*'X264')	H.264/AVC 코덱
cv2.VideoWriter_fourcc(*'MJPG')	Motion-JPEG 코덱

<http://www.fourcc.org/codecs.php>

# 카메라와 동영상 처리하기 2

## ■ 저장을 위한 동영상 파일 열기

```
cv2.VideoWriter(filename, fourcc, fps, frameSize, isColor=None) -> retval
```

- filename: 비디오 파일 이름 (e.g. 'video.mp4')
- fourcc: fourcc (e.g. cv2.VideoWriter\_fourcc(\*'DIVX'))
- fps: 초당 프레임 수 (e.g. 30)
- frameSize: 프레임 크기. (width, height) 튜플.
- isColor: 컬러 영상이면 True, 그렇지 않으면 False.
- retval: cv2.VideoWriter 객체

```
cv2.VideoWriter.open(filename, fourcc, fps, frameSize, isColor=None) -> retval
```

- retval: 성공하면 True, 실패하면 False.

# 카메라와 동영상 처리하기 2

## ■ 비디오 파일이 준비되었는지 확인

```
cv2.VideoCapture.isOpened() -> retval
```

- retval: 성공하면 True, 실패하면 False.

## ■ 프레임 저장하기

```
cv2.VideoWriter.write(image) -> None
```

- image: 저장할 프레임 (numpy.ndarray)

# 카메라와 동영상 처리하기 2

실습: video\_out.py

## ■ 웹카메라 입력을 동영상으로 저장하기

```
cap = cv2.VideoCapture(0)

w = round(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = round(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fourcc = cv2.VideoWriter_fourcc(*'DIVX') # *'DIVX' == 'D', 'I', 'V', 'X'
out = cv2.VideoWriter('output.avi', fourcc, 30, (w, h))

while True:
    ret, frame = cap.read()

    inversed = ~frame
    out.write(inversed)

    cv2.imshow('frame', frame)
    cv2.imshow('inversed', inversed)
    if cv2.waitKey(10) == 27:
        break

...
```

## 2. OpenCV-Python 기초 사용법

7) 키보드 이벤트 처리하기

# 키보드 이벤트 처리하기

## ■ 키보드 입력 대기 함수

```
cv2.waitKey(delay=None) -> retval
```

- delay: 밀리초 단위 대기 시간.  $delay \leq 0$  이면 무한히 기다림. 기본값은 0.
- retval: 눌린 키 값(ASCII code). 키가 눌리지 않으면 -1.
- 참고 사항
  - cv2.waitKey() 함수는 OpenCV 창이 하나라도 있을 때 동작함
  - 특정 키 입력을 확인하려면 ord() 함수를 이용

```
while True:  
    if cv2.waitKey() == ord('q'):  
        break
```

- 주요 특수키 코드: 27(ESC), 13(ENTER), 9(TAB)

# 키보드 이벤트 처리하기

## ■ 키보드 특수키 입력 처리하기

- Windows 운영체제에서 방향키, 함수키 등의 특수키 입력은 [cv2.waitKeyEx\(\)](#) 함수 사용

▼ 표 4-7 주요 특수 키에 해당하는 waitKeyEx() 함수 반환값

특수 키	waitKeyEx() 반환값	특수 키	waitKeyEx() 반환값
Insert	0x2d0000	F1	0x700000
Delete	0x2e0000	F2	0x710000
Home	0x240000	F3	0x720000
End	0x230000	F4	0x730000
Page Up	0x210000	F5	0x740000
Page Down	0x220000	F6	0x750000
←	0x250000	F7	0x760000
↑	0x260000	F8	0x770000
→	0x270000	F9	0x780000
↓	0x280000	F10	0x790000
		F11	0x7a0000
		F12	0x7b0000

# 키보드 이벤트 처리하기

실습: keyboard.py

- 키보드에서 'i' 또는 'I' 키를 누르면 영상을 반전

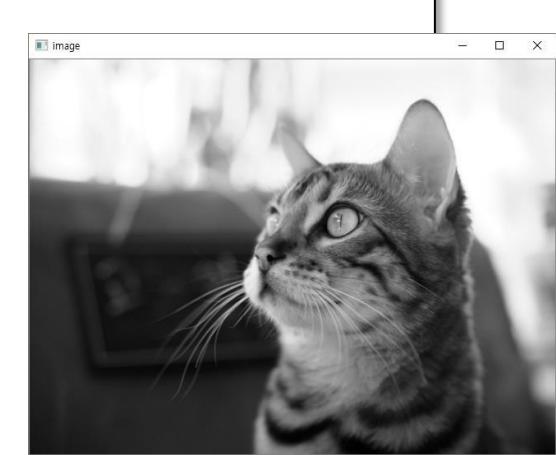
```
import cv2

img = cv2.imread('cat.bmp', cv2.IMREAD_GRAYSCALE)

cv2.imshow('image', img)

while True:
    keycode = cv2.waitKey()
    if keycode == ord('i') or keycode == ord('I'):
        img = ~img
        cv2.imshow('image', img)
    elif keycode == 27:
        break

cv2.destroyAllWindows()
```



## 2. OpenCV-Python 기초 사용법

### 8) 마우스 이벤트 처리하기

# 마우스 이벤트 처리하기

## ■ 마우스 이벤트 콜백함수 등록 함수

```
cv2.setMouseCallback(windowName, onMouse, param=None) -> None
```

- windowName: 마우스 이벤트 처리를 수행할 창 이름
- onMouse: 마우스 이벤트 처리를 위한 콜백 함수 이름.  
마우스 이벤트 콜백 함수는 다음 형식을 따라야 함.

```
onMouse(event, x, y, flags, param) -> None
```

- param: 콜백 함수에 전달할 데이터

# 마우스 이벤트 처리하기

## ■ 마우스 이벤트 처리 함수(콜백 함수) 형식

```
onMouse(event, x, y, flags, param) -> None
```

- event: 마우스 이벤트 종류. cv2.EVENT\_로 시작하는 상수.
- x: 마우스 이벤트가 발생한 x 좌표
- y: 마우스 이벤트가 발생한 y 좌표
- flags: 마우스 이벤트 발생 시 상태. cv2.EVENT\_FLAG\_로 시작하는 상수
- param: cv2.setMouseCallback() 함수에서 설정한 데이터.

# 마우스 이벤트 처리하기

## ■ 마우스 이벤트 처리 함수의 event 인자

MouseEventTypes 열거형 상수	값	설명
cv2.EVENT_MOUSEMOVE	0	마우스가 창 위에서 움직이는 경우
cv2.EVENT_LBUTTONDOWN	1	마우스 왼쪽 버튼이 눌려지는 경우
cv2.EVENT_RBUTTONDOWN	2	마우스 오른쪽 버튼이 눌려지는 경우
cv2.EVENT_MBUTTONDOWN	3	마우스 가운데 버튼이 눌려지는 경우
cv2.EVENT_LBUTTONUP	4	마우스 왼쪽 버튼이 떼어지는 경우
cv2.EVENT_RBUTTONUP	5	마우스 오른쪽 버튼이 떼어지는 경우
cv2.EVENT_MBUTTONUP	6	마우스 가운데 버튼이 떼어지는 경우
cv2.EVENT_LBUTTONDOWNDBLCLK	7	마우스 왼쪽 버튼을 더블클릭하는 경우
cv2.EVENT_RBUTTONDOWNDBLCLK	8	마우스 오른쪽 버튼을 더블클릭하는 경우
cv2.EVENT_MBUTTONDOWNDBLCLK	9	마우스 가운데 버튼을 더블클릭하는 경우
cv2.EVENT_MOUSEWHEEL	10	마우스 휠을 앞뒤로 돌리는 경우
cv2.EVENT_MOUSEHWHEEL	11	마우스 휠을 좌우로 움직이는 경우

# 마우스 이벤트 처리하기

## ■ 마우스 이벤트 처리 함수의 flags 인자

MouseEventFlags 열거형 상수	값	설명
cv2.EVENT_FLAG_LBUTTON	1	마우스 왼쪽 버튼이 눌려져 있음
cv2.EVENT_FLAG_RBUTTON	2	마우스 오른쪽 버튼이 눌려져 있음
cv2.EVENT_FLAG_MBUTTON	4	마우스 가운데 버튼이 눌려져 있음
cv2.EVENT_FLAG_CTRLKEY	8	CTRL 키가 눌려져 있음
cv2.EVENT_FLAG_SHIFTKEY	16	SHIFT 키가 눌려져 있음
cv2.EVENT_FLAG_ALTKEY	32	ALT 키가 눌려져 있음

# 마우스 이벤트 처리하기

실습: mouse.py

## ■ 마우스를 이용한 그리기 예제

```
oldx = oldy = -1

def on_mouse(event, x, y, flags, param):
    global oldx, oldy

    if event == cv2.EVENT_LBUTTONDOWN:
        oldx, oldy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if flags & cv2.EVENT_FLAG_LBUTTON:
            cv2.line(img, (oldx, oldy), (x, y), (0, 0, 255), 4, cv2.LINE_AA)
            cv2.imshow('image', img)
        oldx, oldy = x, y

img = np.ones((480, 640, 3), dtype=np.uint8) * 255

cv2.imshow('image', img)
cv2.setMouseCallback('image', on_mouse)
cv2.waitKey()
```

# 마우스 이벤트 처리하기

실습: mouse.py

- 마우스를 이용한 그리기 예제



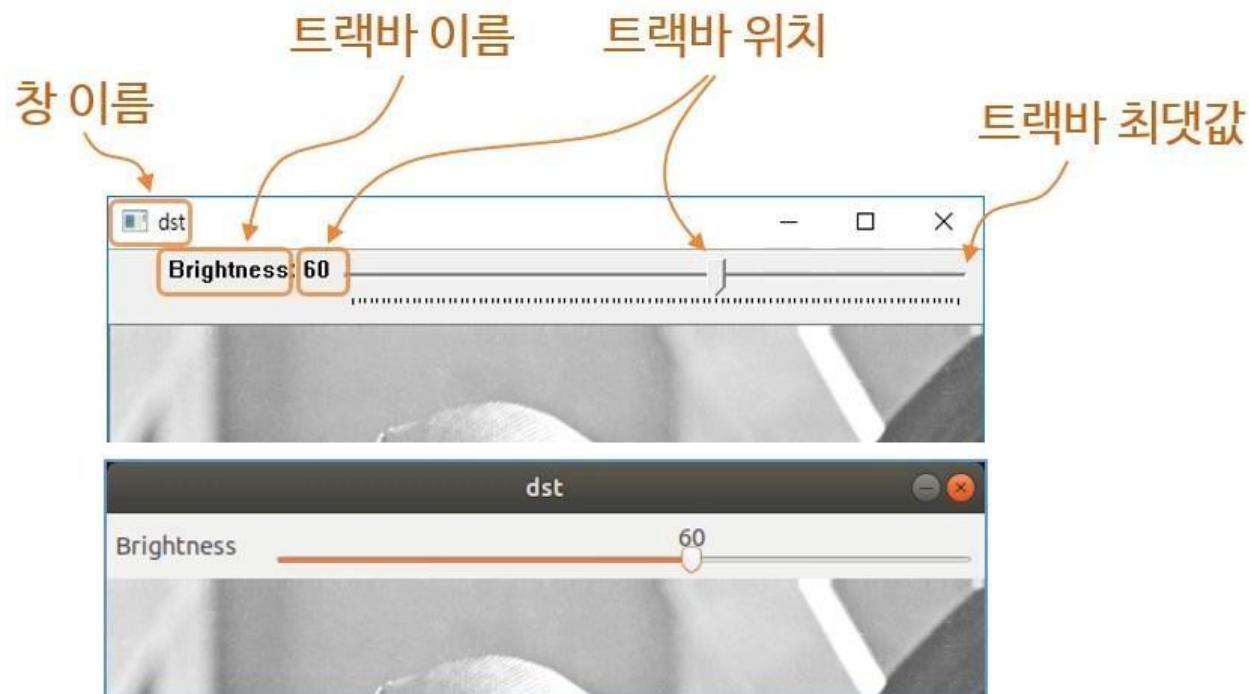
## 2. OpenCV-Python 기초 사용법

### 9) 트랙바 사용하기

# 트랙바 사용하기

## ■ 트랙바(Trackbar)란?

- 프로그램 동작 중 사용자가 지정한 범위 안의 값을 선택할 수 있는 컨트롤
- OpenCV에서 제공하는 (유일한?) 그래픽 사용자 인터페이스



# 트랙바 사용하기

## ■ 트랙바 생성 함수

```
cv2.createTrackbar(trackbarName, windowName, value, count, onChange) -> None
```

- trackbarName: 트랙바 이름
- windowName: 트랙바를 생성할 창 이름.
- value: 트랙바 위치 초기값
- count: 트랙바 최댓값. 최솟값은 항상 0.
- onChange: 트랙바 위치가 변경될 때마다 호출할 콜백 함수 이름  
트랙바 이벤트 콜백 함수는 다음 형식을 따름.

```
onChange(pos) -> None
```

# 트랙바 사용하기

실습: trackbar.py

- 트랙바를 이용한 그레이스케일 레벨 표현

```
def on_level_change(pos):
    value = pos * 16
    if value >= 255:
        value = 255

    img[:] = value
    cv2.imshow('image', img)

img = np.zeros((480, 640), np.uint8)

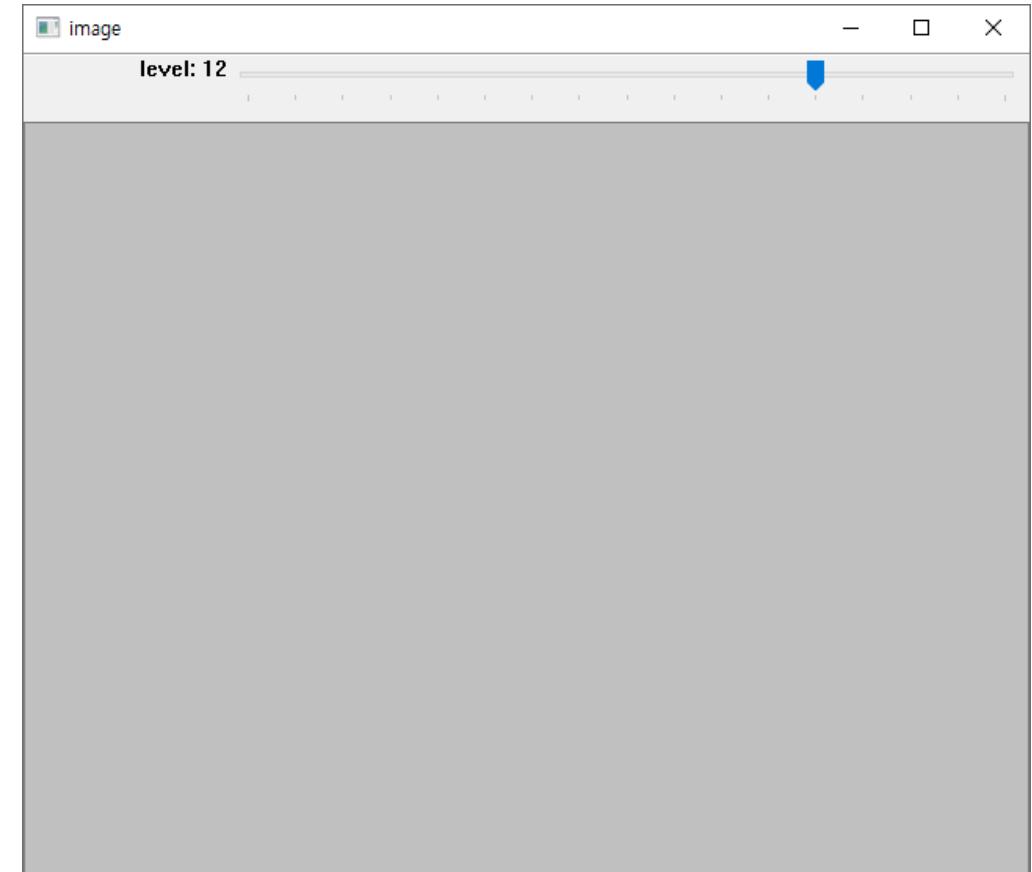
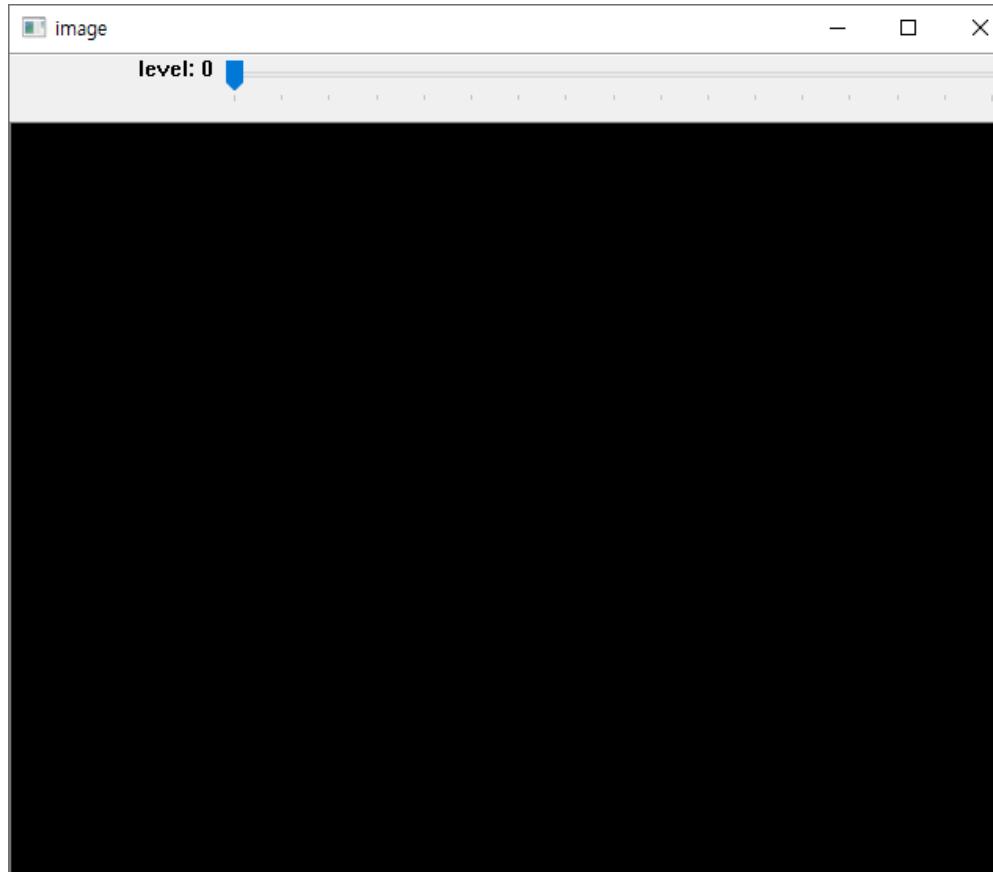
cv2.namedWindow('image')
cv2.createTrackbar('level', 'image', 0, 16, on_level_change)

cv2.imshow('image', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

# 트랙바 사용하기

실습: trackbar.py

- 트랙바를 이용한 그레이스케일 레벨 표현

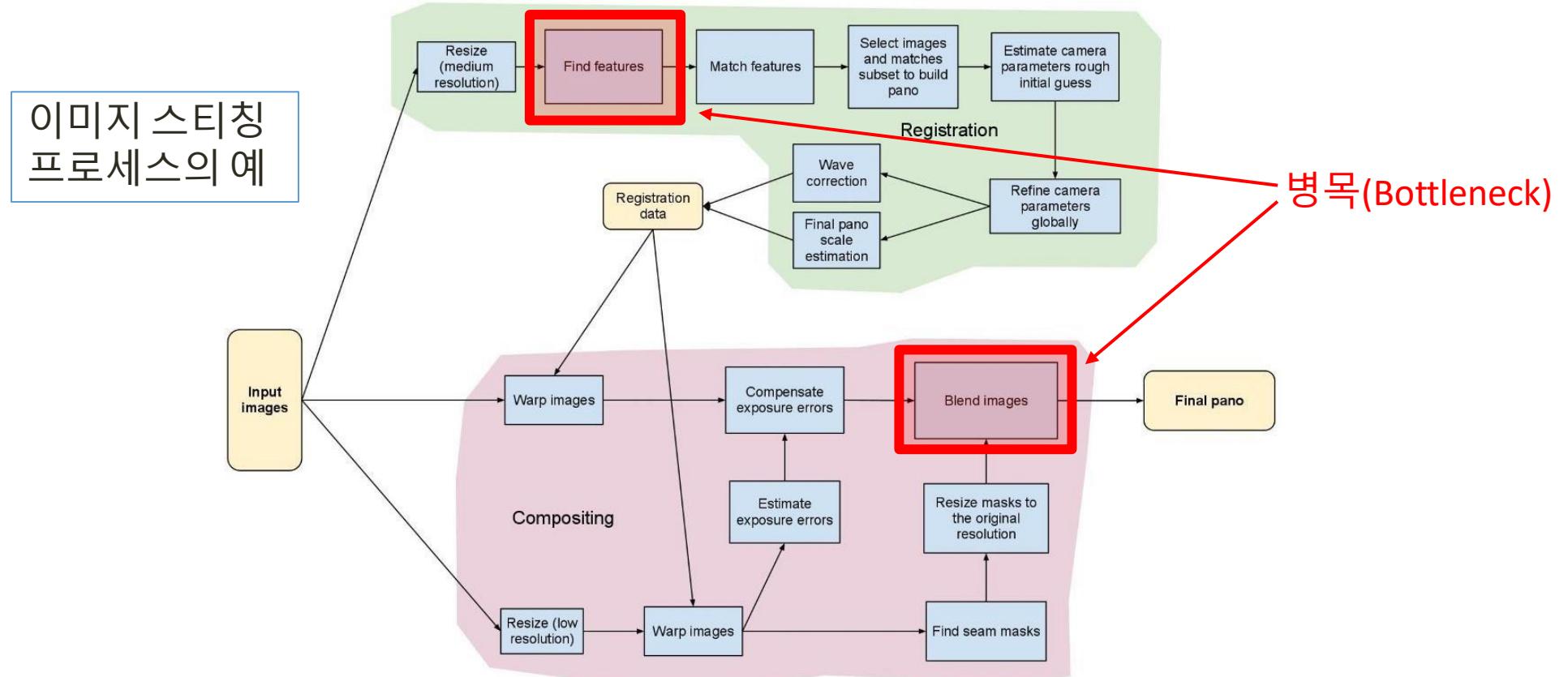


# OpenCV-Python 기초 사용법

## 10) 연산 시간 측정 방법

# 연산 시간 측정 방법

- 컴퓨터 비전은 대용량 데이터를 다루고, 일련의 과정을 통해 최종 결과를 얻으므로 매 단계에서 연산 시간을 측정하여 관리할 필요가 있음



# 연산 시간 측정 방법

- OpenCV에서는 TickMeter 클래스를 이용하여 연산 시간을 측정

```
cv2.TickMeter() -> tm
```

- tm: cv2.TickMeter 객체
  - tm.start(): 시간 측정 시작
  - tm.stop(): 시간 측정 끝 시
  - tm.reset(): 간 측정초기화
- tm.getTimeSec(): 측정 시간을 초 단위로 반환
- tm.getTimeMilli(): 측정 시간을 밀리 초 단위로 반환
- tm.getTimeMicro(): 측정 시간을 마이크로 초 단위로 반환

# 연산 시간 측정 방법

실습: time\_check.py

## ■ 특정 연산의 시간 측정 예제

```
img = cv2.imread('hongkong.jpg')

tm = cv2.TickMeter()
tm.start()

edge = cv2.Canny(img, 50, 150)

tm.stop()

print('Elapsed time: {}ms.'.format(tm.getTimeMilli()))
```

## 2. OpenCV-Python 기초 사용법

11) 실전 코딩: 동영상 전환 이펙트

# [실전 코딩] 동영상 전환 이펙트

실습: video\_effect.py

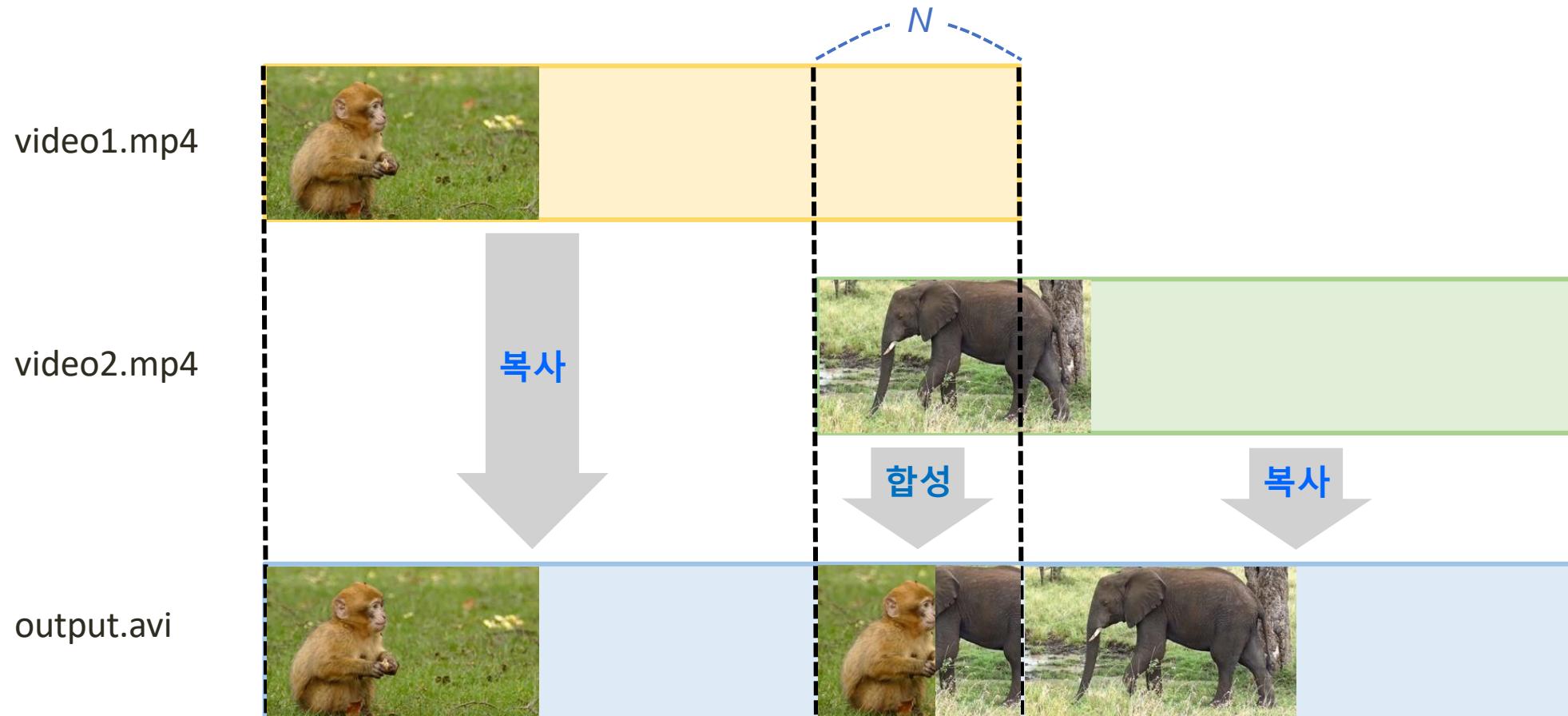
## ■ 동영상 전환 이펙트

- 두 동영상 클립 사이에 추가되는 애니메이션 효과
- 페이드-인(fade-in), 페이드-아웃(fade-out), 디졸브(dissolve), 밀기, 확대 등

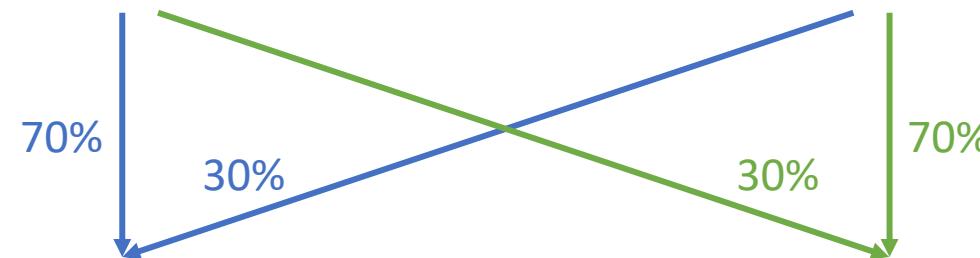
## ■ 구현 할 기능

- 두 개의 동영상 동시 열기
- 첫 번째 동영상의 마지막 N개 프레임과 두 번째 동영상의 처음 N개 프레임을 합성
- 합성된 영상을 동영상으로 저장하기

# [실전 코딩] 동영상 전환 이펙트



# [실전 코딩] 동영상 전환 이펙트



### 3. 기본적인 영상 처리 기법

1) 영상의 밝기 조절

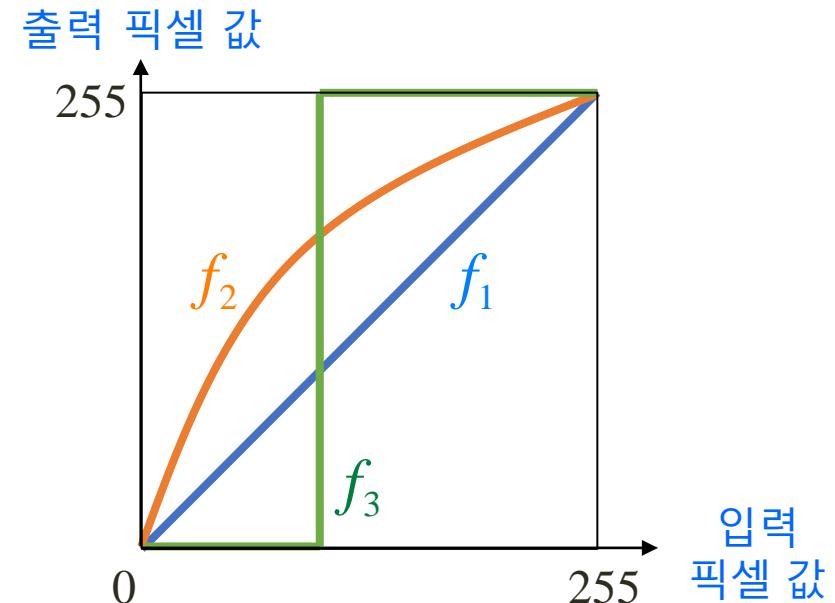
# 영상의 화소 처리 기법

## ■ 화소 처리(Point processing)

- 입력 영상의 특정 좌표 픽셀 값을 변경하여 출력 영상의 해당 좌표 픽셀 값으로 설정하는 연산

$$dst(x, y) = f(\text{src}(x, y))$$

변환 함수  
(transfer function)



- 결과 영상의 픽셀 값이 정해진 범위(e.g. 그레이스케일)에 있어야 함
- 반전, 밝기 조절, 명암비 조절 등

# 영상의 밝기 조절

## ■ 밝기 조절이란?

- 영상을 전체적으로 더욱 밝거나 어둡게 만드는 연산



원본 영상

밝기 -50 조절



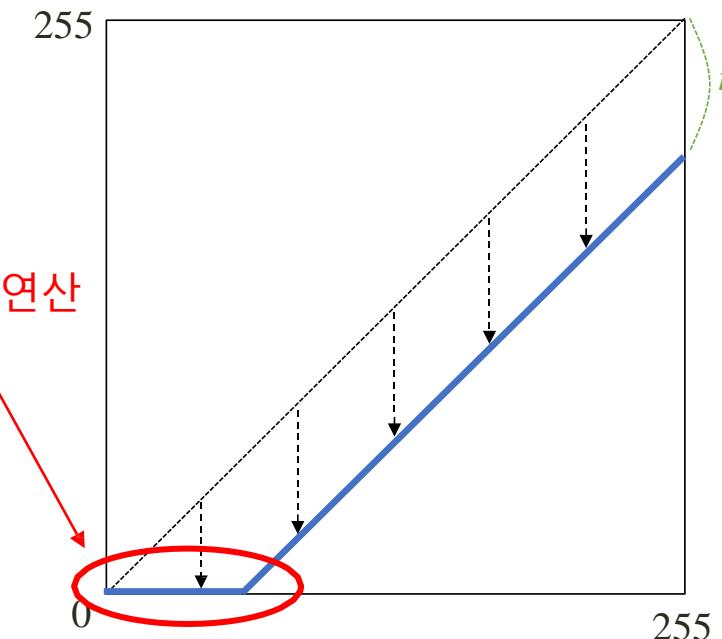
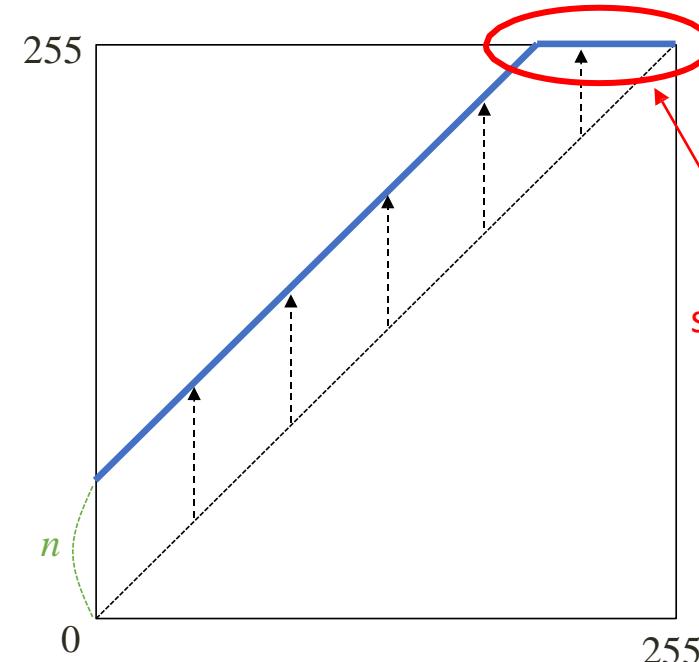
밝기 +50 조절



# 영상의 밝기 조절

## ■ 밝기 조절 수식

$$dst(x, y) = \text{saturate}(\text{src}(x, y) + n)$$



# 영상의 밝기 조절

## ■ 영상의 밝기 조절을 위한 영상의 덧셈 연산

```
cv2.add(src1, src2, dst=None, mask=None, dtype=None) -> dst
```

- src1: (입력) 첫 번째 영상 또는 스칼라
- src2: (입력) 두 번째 영상 또는 스칼라
- dst: (출력) 덧셈 연산의 결과 영상
- mask: 마스크 영상
- dtype: 출력 영상(dst)의 타입. (e.g.) cv2.CV\_8U, cv2.CV\_32F 등
- 참고사항
  - 스칼라(Scalar)는 실수 값 하나 또는 실수 값 네 개로 구성된 튜플
  - dst를 함수 인자로 전달하려면 dst의 크기가 src1, src2와 같아야 하며, 타입이 적절해야 함

# 영상의 밝기 조절

실습: brightness.py

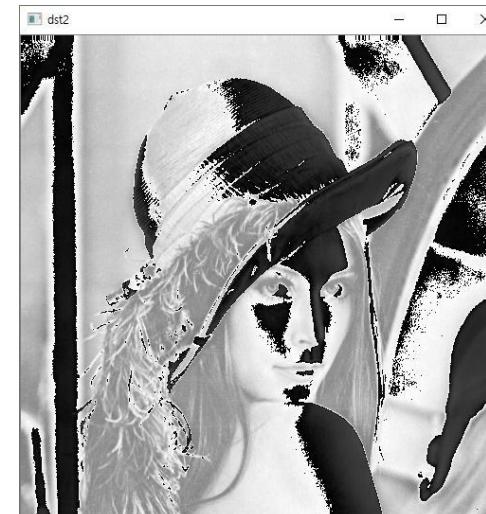
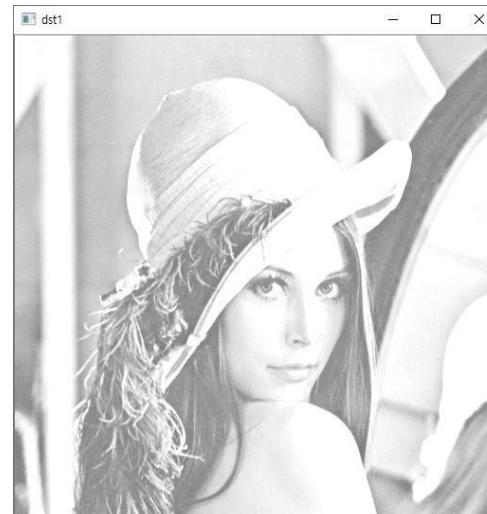
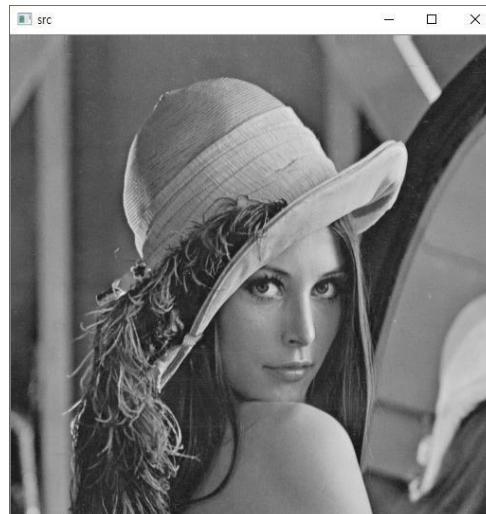
- 그레이스케일 영상의 밝기 100만큼 증가시키기

```
src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

dst1 = cv2.add(src, 100)

dst2 = src + 100

#dst2 = np.clip(src + 100., 0, 255).astype(np.uint8)
```



# 영상의 밝기 조절

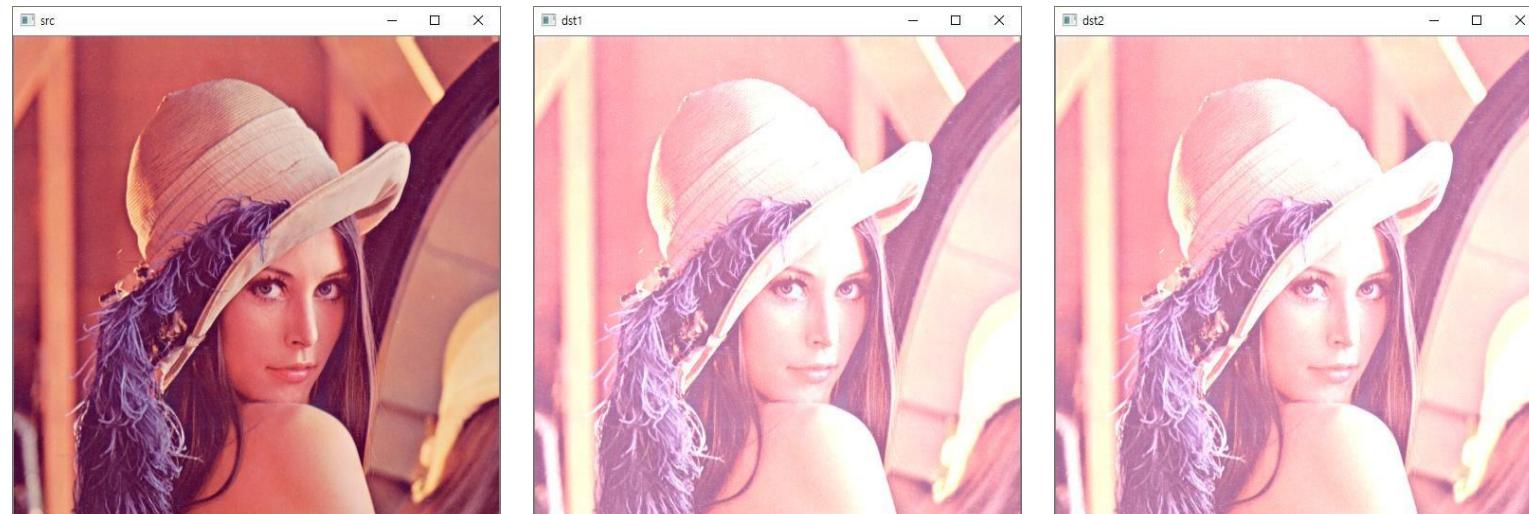
실습: brightness.py

- 컬러 영상의 밝기 100만큼 증가시키기

```
src = cv2.imread('lenna.bmp')

dst1 = cv2.add(src, (100, 100, 100, 0))

dst2 = np.clip(src + 100., 0, 255).astype(np.uint8)
```



### 3. 기본적인 영상 처리 기법

2) 영상의 산술 및 논리 연산

# 영상의 산술 연산

## ■ 덧셈 연산

```
cv2.add(src1, src2, dst=None, mask=None, dtype=None) -> dst
```

- 두 영상의 같은 위치에 존재하는 픽셀 값을 더하여 결과 영상의 픽셀 값으로 설정
- 덧셈 결과가 255보다 크면 픽셀 값을 255로 설정 (포화 연산)



+



=



# 영상의 산술 연산

## ■ 덧셈 연산

```
cv2.add(src1, src2, dst=None, mask=None, dtype=None) -> dst
```

- src1: (입력) 첫 번째 영상 또는 스칼라
- src2: (입력) 두 번째 영상 또는 스칼라
- dst: (출력) 덧셈 연산의 결과 영상
- mask: 마스크 영상
- dtype: 출력 영상(dst)의 타입. (e.g.) cv2.CV\_8U, cv2.CV\_32F 등
- 참고사항
  - 스칼라(Scalar)는 실수 값 하나 또는 실수 값 네 개로 구성된 튜플
  - dst를 함수 인자로 전달하려면 dst의 크기가 src1, src2와 같아야 하며, 타입이 적절해야 함

# 영상의 산술연산

## ■ 가중치 합(weighted sum)

$$dst(x, y) = \text{saturate}(\alpha \cdot \text{src1}(x, y) + \beta \cdot \text{src2}(x, y))$$

- 두 영상의 같은 위치에 존재하는 픽셀 값에 대하여 가중합을 계산하여 결과 영상의 픽셀 값으로 설정
- 보통  $\alpha + \beta = 1$  이 되도록 설정 → 두 입력 영상의 평균 밝기를 유지

## ■ 평균 연산(average)

- 가중치를  $\alpha = \beta = 0.5$  로 설정한 가중치 합

$$dst(x, y) = \frac{1}{2} (\text{src1}(x, y) + \text{src2}(x, y))$$

# 영상의 산술연산

## ■ 가중치 합(weighted sum)



$\alpha=1, \beta=0$



$\alpha=0.75, \beta=0.25$



$\alpha=0.5, \beta=0.5$



$\alpha=0.25, \beta=0.75$



$\alpha=0, \beta=1$

# 영상의 산술연산

## ■ 가중치 합(weighted sum)

```
cv2.addWeighted(src1, alpha, src2, beta, gamma, dst=None, dtype=None) -> dst
```

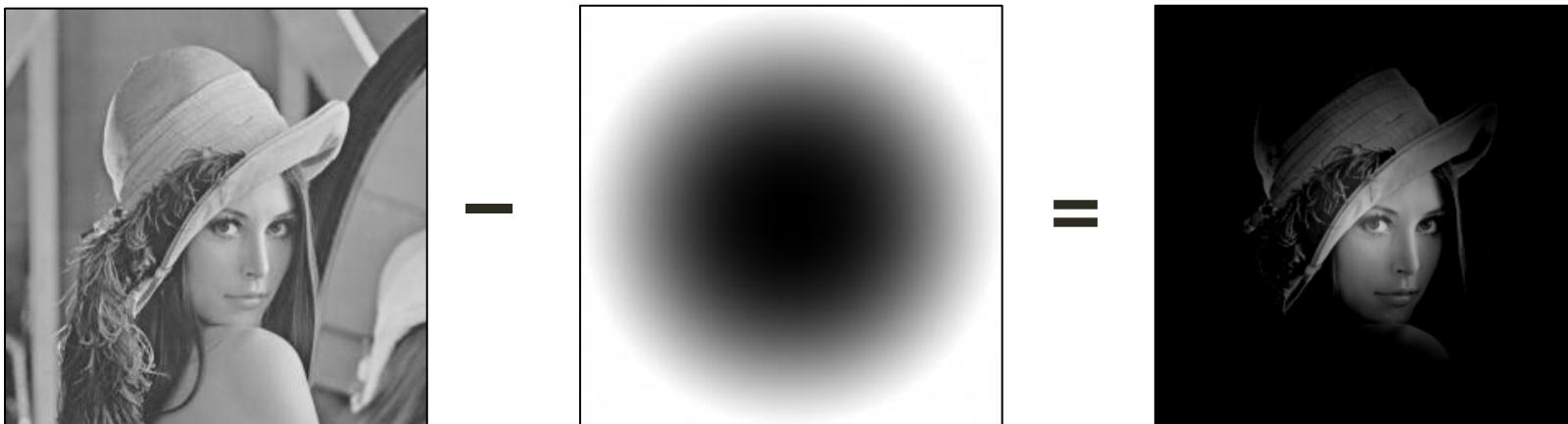
- src1: 첫 번째 영상
- alpha: 첫 번째 영상 가중치
- src2: 두 번째 영상. src1과 같은 크기 & 같은 타입
- beta: 두 번째 영상 가중치
- gamma: 결과 영상에 추가적으로 더할 값
- dst: 가중치 합 결과 영상
- dtype: 출력 영상(dst)의 타입

# 영상의 산술 연산

## ■ 뺄셈 연산



- 두 영상의 같은 위치에 존재하는 픽셀 값에 대하여 뺄셈 연산을 수행하여 결과 영상의 픽셀 값으로 설정
- 뺄셈 결과가 0보다 작으면 픽셀 값을 0으로 설정 (포화 연산)



# 영상의 산술 연산

## ■ 뺄셈 연산

```
cv2.subtract(src1, src2, dst=None, mask=None, dtype=None) -> dst
```

- src1: 첫 번째 영상 또는 스칼라
- src2: 두 번째 영상 또는 스칼라
- dst: 뺄셈 연산 결과 영상 마스크
- mask: 스크 영상
- dtype: 출력 영상(dst)의 타입

# 영상의 산술 연산

## ■ 차이 연산

$$dst(x, y) = |src1(x, y) - src2(x, y)|$$

- 두 영상의 같은 위치에 존재하는 픽셀 값에 대하여 뺄셈 연산을 수행한 후, 그 절댓값을 결과 영상의 픽셀 값으로 설정
- 뺄셈 연산과 달리 입력 영상의 순서에 영향을 받지 않음



# 영상의 산술 연산

## ■ 차이 연산

```
cv2.absdiff(src1, src2, dst=None) -> dst
```

- src1: 첫 번째 영상 또는 스칼라
- src2: 두 번째 영상 또는 스칼라
- dst: 차이 연산 결과 영상(차영상)

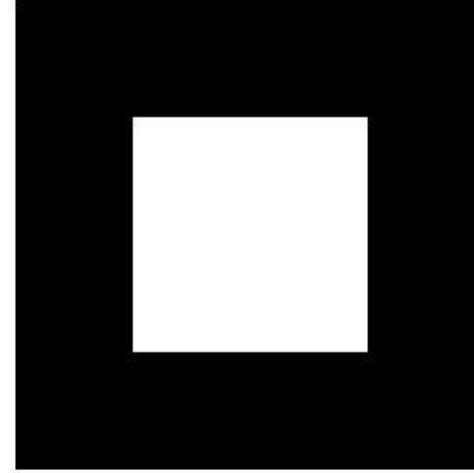
# 영상의 산술연산

실습: arithmetic.py

src1



src2



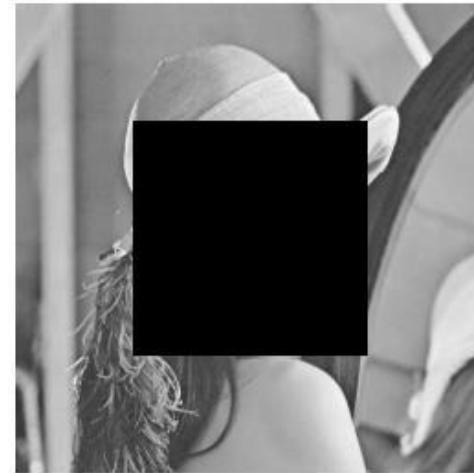
add



addWeighted



subtract



absdiff



# 영상의 논리 연산

## ■ 비트단위 AND, OR, XOR, NOT 연산

```
cv2.bitwise_and(src1, src2, dst=None, mask=None) -> dst  
cv2.bitwise_or(src1, src2, dst=None, mask=None) -> dst  
cv2.bitwise_xor(src1, src2, dst=None, mask=None) -> dst  
cv2.bitwise_not(src1, dst=None, mask=None) -> dst
```

- src1: 첫 번째 영상 또는 스칼라
- src2: 두 번째 영상 또는 스칼라
- dst: 출력 영상
- mask: 마스크 영상
- 참고사항
  - 각각의 픽셀 값을 이진수로 표현하고, 비트(bit) 단위 논리 연산을 수행

입력 비트		논리 연산 결과			
a	b	a AND b	a OR b	a XOR b	NOT a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

### 3. 기본적인 영상 처리 기법

#### 3) 컬러 영상과 색 공간

# 컬러 영상과 색 공간

## ■ OpenCV와 컬러 영상

- 컬러 영상은 3차원 `numpy.ndarray`로 표현. `img.shape = (h, w, 3)`
- OpenCV에서는 RGB 순서가 아니라 **BGR** 순서를 기본으로 사용

## ■ OpenCV에서 컬러 영상 다루기

```
img1 = cv2.imread('lenna.bmp', cv2.IMREAD_COLOR)

img2 = np.zeros((480, 640, 3), np.uint8)

img3 = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
img4 = cv2.cvtColor(img3, cv2.COLOR_GRAY2BGR)
```

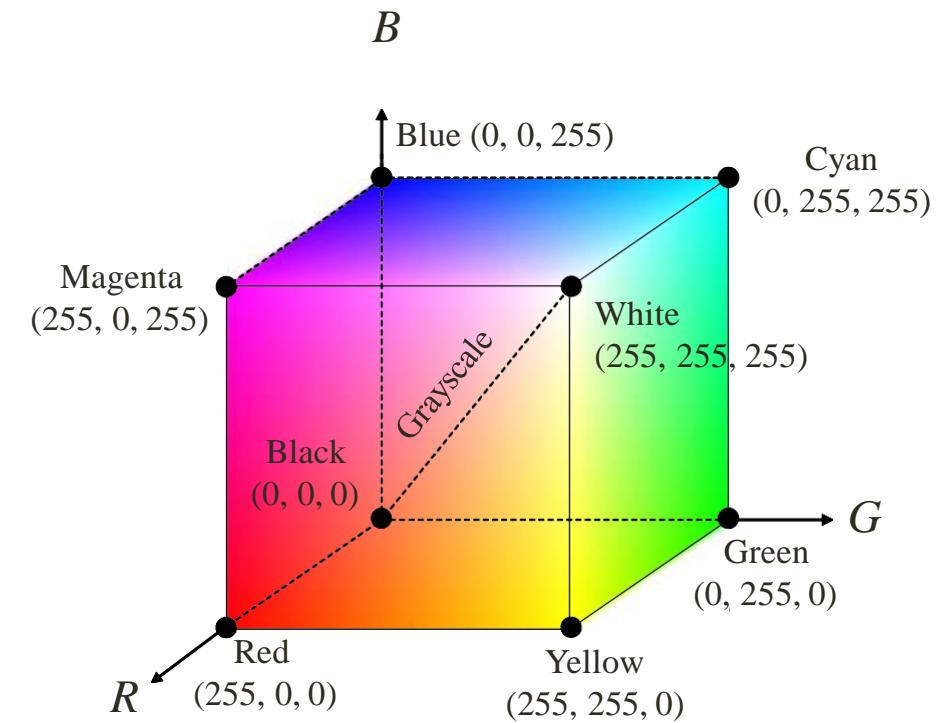
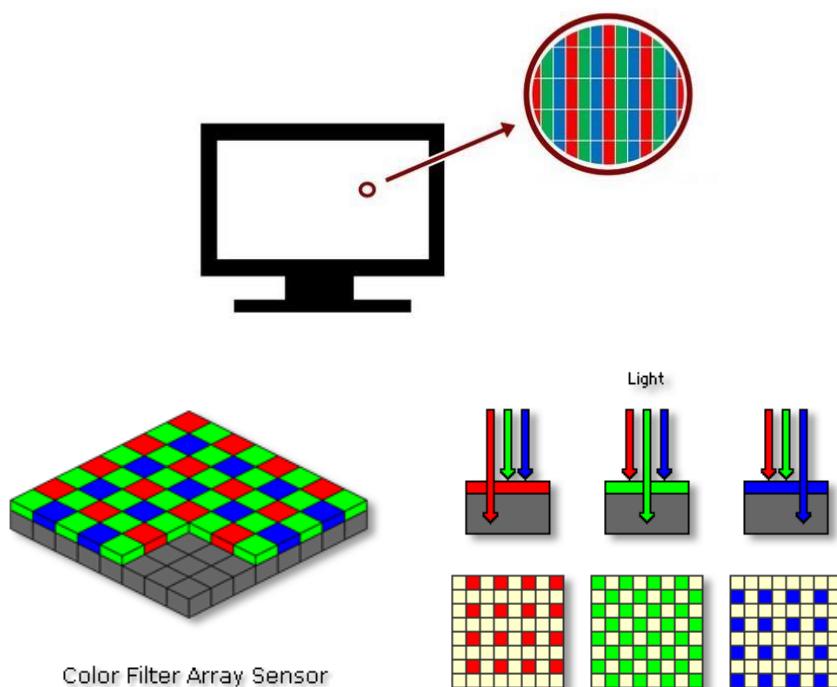


이 경우 img4 영상의 각 픽셀은 B, G, R, 색 성분 값이 모두 같게 설정됨

# 컬러 영상과 색 공간

## ■ RGB 색 공간

- 빛의 삼원색인 빨간색(R), 녹색(G), 파란색(B)을 혼합하여 색상을 표현 (가산 혼합)
- TV & 모니터, 카메라 센서 Bayer 필터, 비트맵



# 컬러 영상과 색 공간

## ■ (색상) 채널 분리

```
cv2.split(m, mv=None) -> dst
```

- m:                  다채널 영상 (e.g.) (B, G, R)로 구성된 컬러 영상
- mv:                  출력 영상
- dst:                출력 영상의 리스트

## ■ (색상) 채널 결합

```
cv2.merge(mv, dst=None) -> dst
```

- mv:                입력 영상 리스트 또는 튜플
- dst:               출력 영상

# 컬러 영상과 색 공간

실습: color.py

## ■ RGB 색상 평면 나누기

```
src = cv2.imread('candies.png', cv2.IMREAD_COLOR)

# 컬러 영상 속성 확인
print('src.shape:', src.shape) # src.shape: (480, 640, 3)
print('src.dtype:', src.dtype) # src.dtype: uint8

# RGB 색 평면 분할
b_plane, g_plane, r_plane = cv2.split(src)

cv2.imshow('src', src)
cv2.imshow('B_plane', b_plane)
cv2.imshow('G_plane', g_plane)
cv2.imshow('R_plane', r_plane)
```

```
b_plane = src[:, :, 0]
g_plane = src[:, :, 1]
r_plane = src[:, :, 2]
```

# 컬러 영상과 색 공간

## ■ RGB 색상 평면



B 평면



G 평면

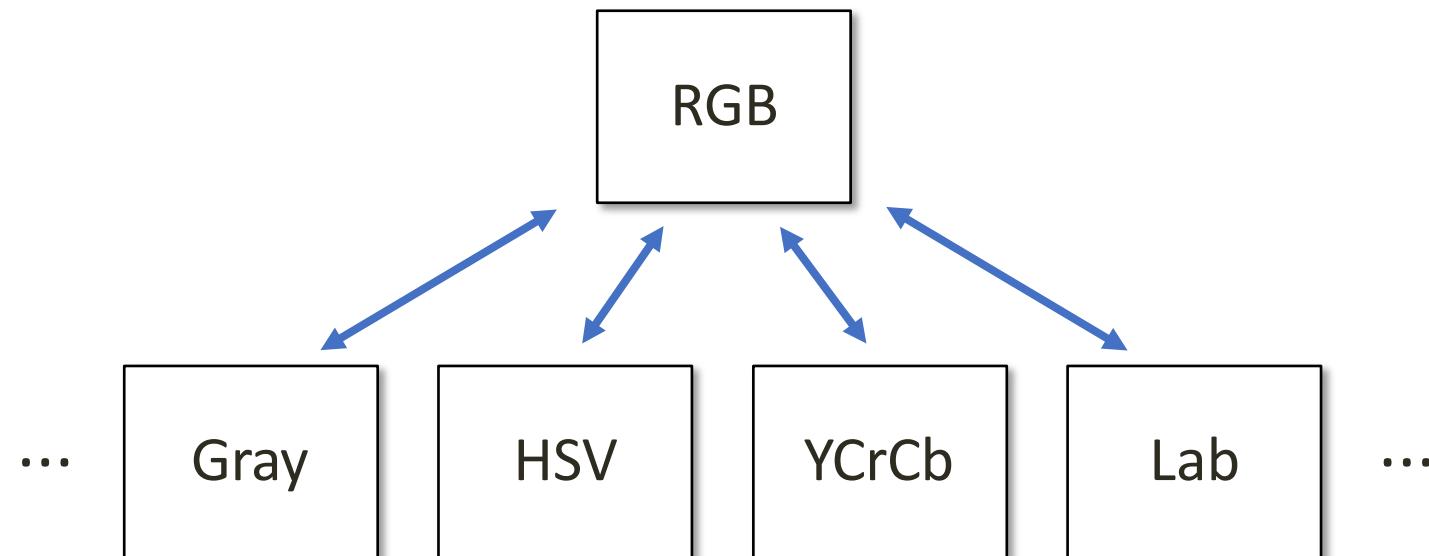


R 평면

# 컬러 영상과 색 공간

## ■ 색 공간 변환

- 영상 처리에서는 특정한 목적을 위해 RGB 색 공간을 HSV, YCrCb, Grayscale 등의 다른 색 공간으로 변환하여 처리
- OpenCV 색 공간 변환 방법
  - [https://docs.opencv.org/master/de/d25/imgproc\\_color\\_conversions.html](https://docs.opencv.org/master/de/d25/imgproc_color_conversions.html) 참고



# 컬러 영상과 색 공간

## ■ 색 공간 변환 함수

```
cv2.cvtColor(src, code, dst=None, dstCn=None) -> dst
```

- src: 입력 영상
- code: 색 변환 코드 ([OpenCV 문서 페이지](#) 참고)

cv2.COLOR_BGR2GRAY / cv2.COLOR_GRAY2BGR	BGR ↔ GRAY
cv2.COLOR_BGR2RGB / cv2.COLOR_RGB2BGR	BGR ↔ RGB
cv2.COLOR_BGR2HSV / cv2.COLOR_HSV2BGR	BGR ↔ HSV
cv2.COLOR_BGR2YCrCb / cv2.COLOR_YCrCb2BGR	BGR ↔ YCrCb

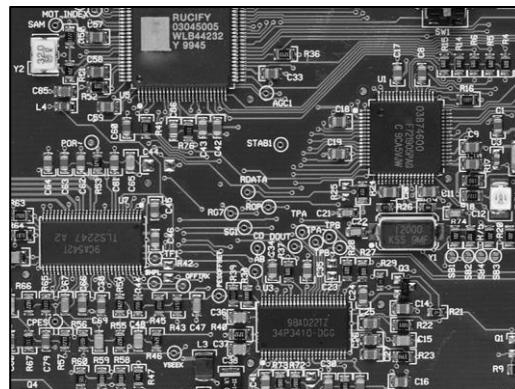
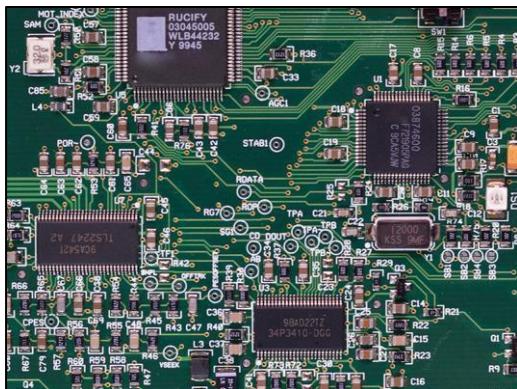
- dstCn: 결과 영상의 채널 수. 0이면 자동 결정.
- dst: 출력 영상

# 컬러 영상과 색 공간

- RGB 색상을 그레이스케일로 변환

$$Y = 0.299R + 0.587G + 0.114B$$

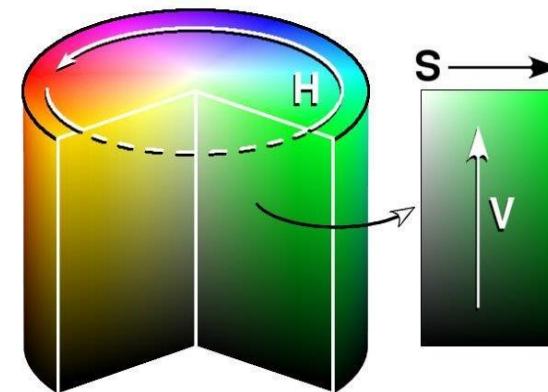
- 장점: 데이터 저장 용량 감소, 데이터 처리 속도 향상
- 단점: 색상 정보 손실



# 컬러 영상과 색 공간

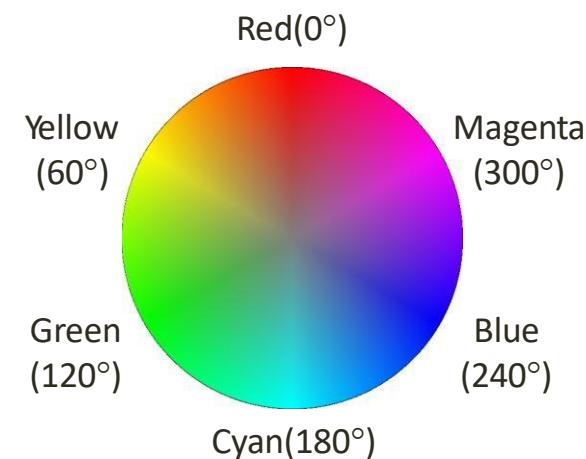
## ■ HSV 색 공간

- Hue: 색상, 색의 종류
- Saturation: 채도, 색의 탁하고 선명한 정도
- Value: 명도, 빛의 밝기



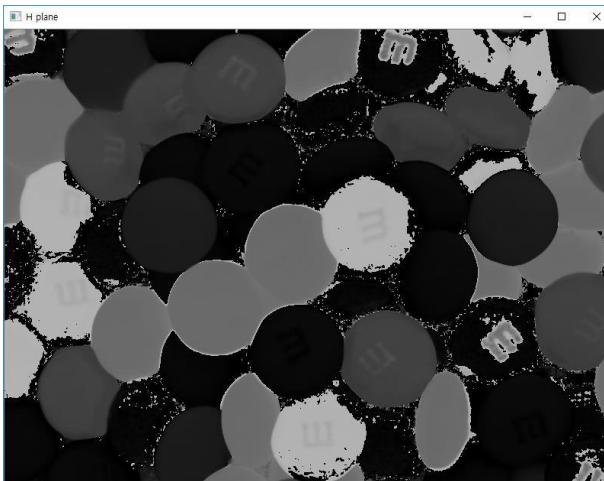
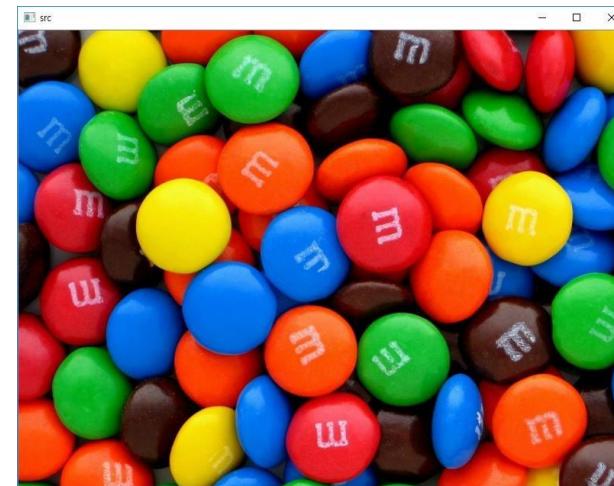
## ■ HSV 값 범위

- cv2.CV\_8U 영상의 경우
  - $0 \leq H \leq 179$
  - $0 \leq S \leq 255$
  - $0 \leq V \leq 255$

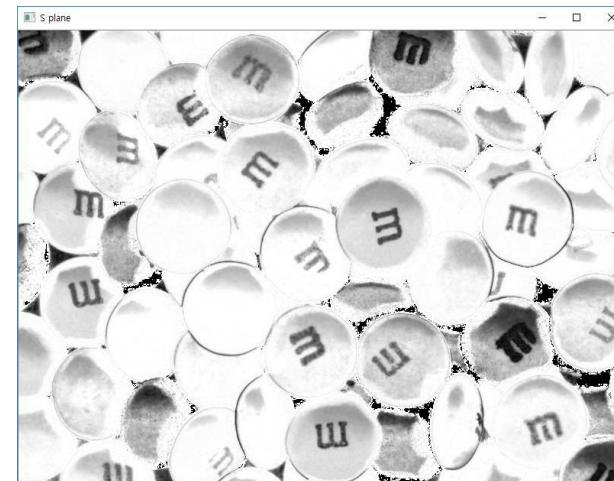


# 컬러 영상과 색 공간

## ■ HSV 색상 평면



H 평면



S 평면



V 평면

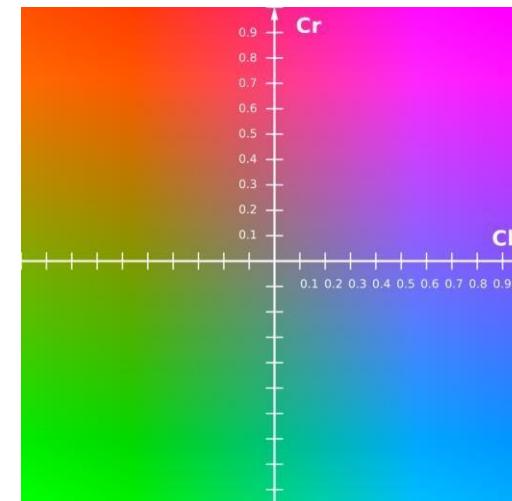
# 컬러 영상과 색 공간

## ■ YCrCb 색 공간

- PAL, NTSC, SECAM 등의 컬러 비디오 표준에 사용되는 색 공간
- 영상의 밝기 정보와 색상 정보를 따로 분리하여 부호화 (흑백 TV 호환)
- Y: 밝기 정보(luma)
- Cr, Cb: 색차(chroma)

## ■ YCrCb 값 범위

- cv2.CV\_8U 영상의 경우
  - $0 \leq Y \leq 255$
  - $0 \leq Cr \leq 255$
  - $0 \leq Cb \leq 255$

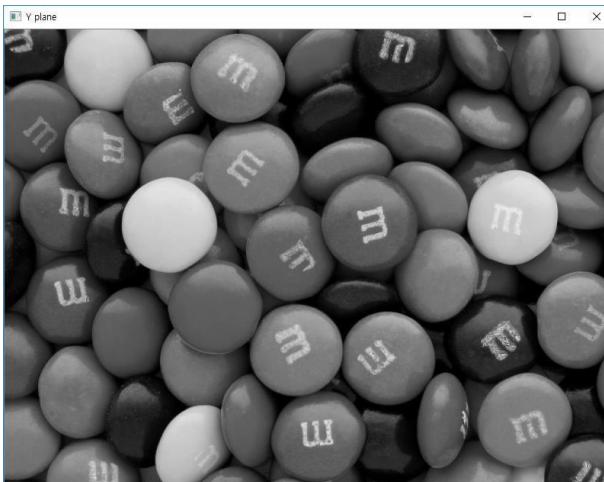
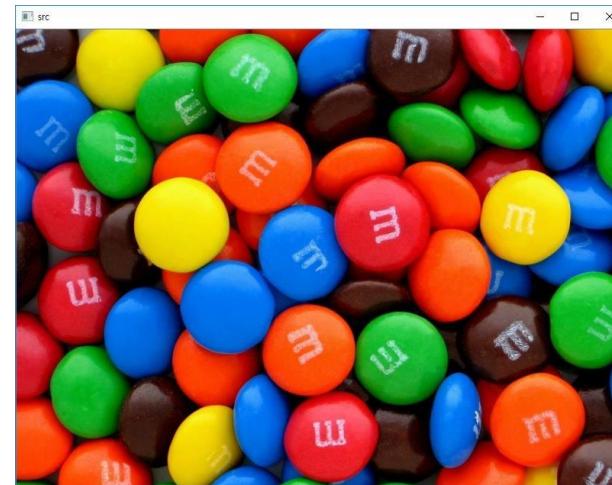


CbCr 평면 ( $Y=0.5$  고정)

<https://ko.wikipedia.org/wiki/YUV>

# 컬러 영상과 색 공간

## ■ YCrCb 색상 평면



Y 평면



Cr 평면



Cb 평면

### 3. 기본적인 영상 처리 기법

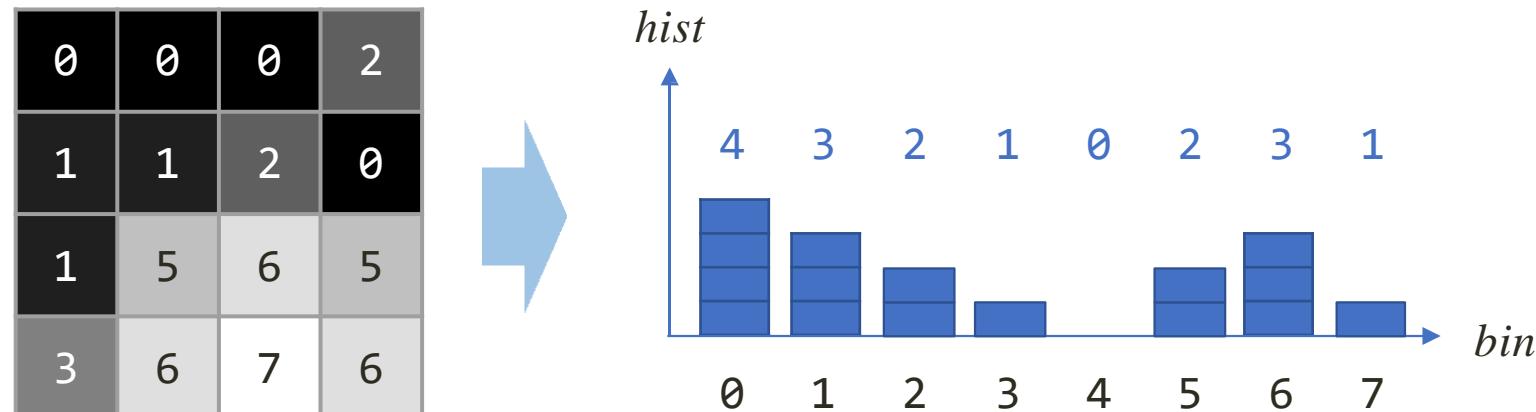
#### 4) 히스토그램 분석

# 히스토그램 분석

## ■ 히스토그램(Histogram)

- 영상의 픽셀 값 분포를 그래프의 형태로 표현한 것
- 예를 들어 그레이스케일 영상에서 각 그레이스케일 값에 해당하는 픽셀의 개수를 구하고, 이를 막대 그래프의 형태로 표현

$$h(g) = N_g$$



# 히스토그램 분석

## ■ 정규화된 히스토그램(Normalized histogram)

- 각 픽셀의 개수를 영상 전체 픽셀 개수로 나누어준 것
- 해당 그레이스케일 값을 갖는 픽셀이 나타날 확률

$$p(g) = \frac{N_g}{w \times h} \rightarrow \sum_{g=0}^{L-1} p(g)$$

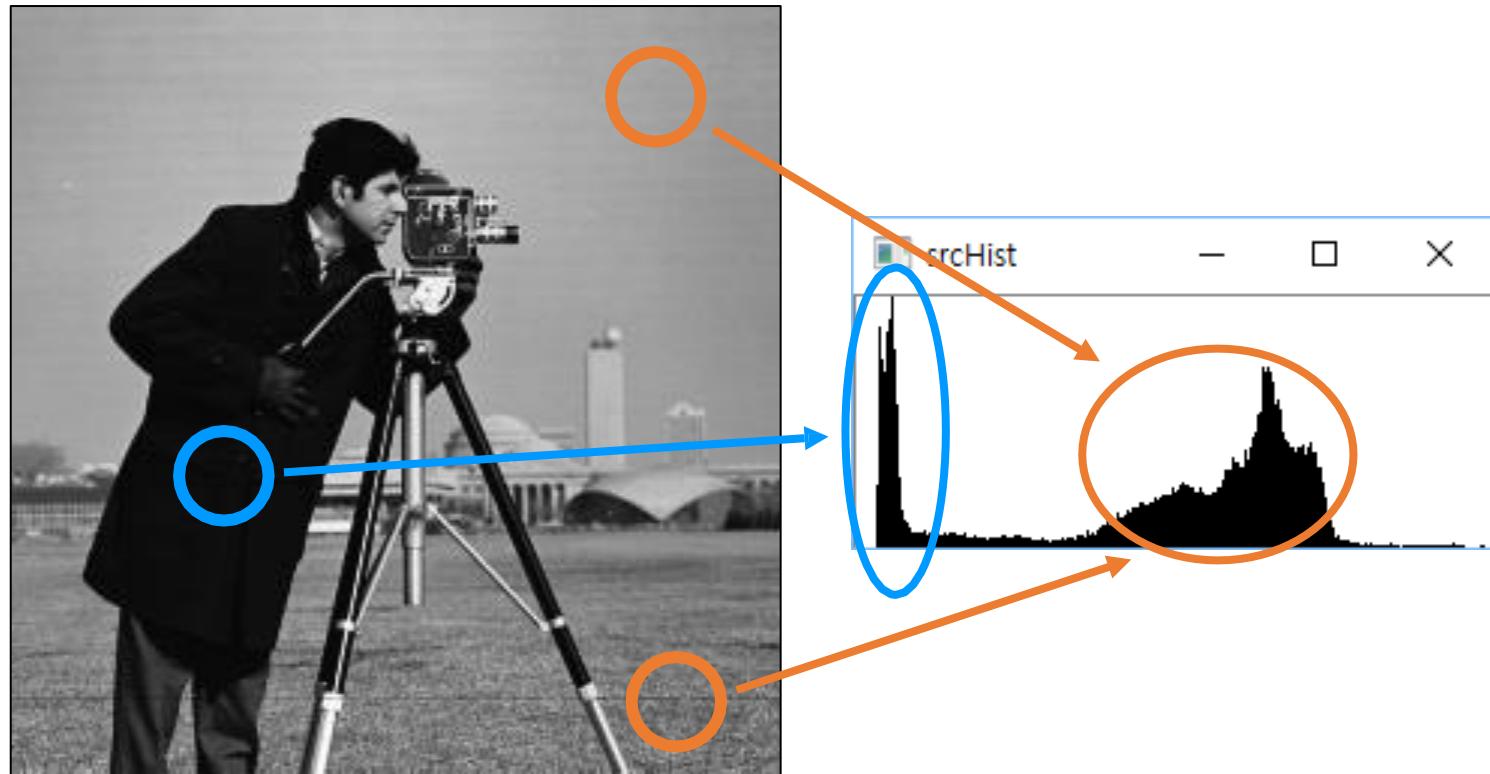
0	0	0	2
1	1	2	0
1	5	6	5
3	6	7	6



$g$	0	1	2	3	4	5	6	7
$h(g)$	4	3	2	1	0	2	3	1
$p(g)$	4	3	2	1	0	2	3	1
	16	16	16	16	0	16	16	16

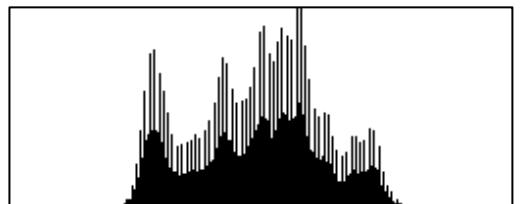
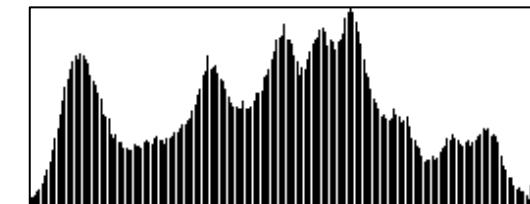
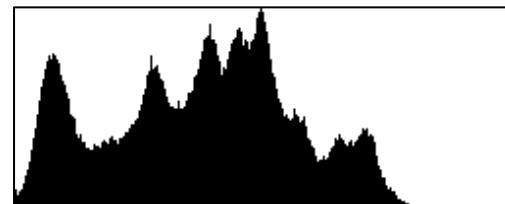
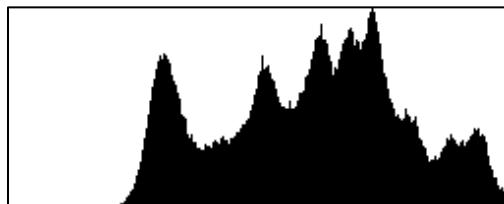
# 히스토그램 분석

- 영상과 히스토그램의 관계



# 히스토그램 분석

- 영상과 히스토그램의 관계



# 히스토그램 분석

## ■ 히스토그램 구하기

```
cv2.calcHist(images, channels, mask, histSize, ranges, hist=None,  
            accumulate=None) -> hist
```

- images: 입력 영상 리스트
- channels: 히스토그램을 구할 채널을 나타내는 리스트
- mask: 마스크 영상. 입력 영상 전체에서 히스토그램을 구하려면 **None** 지정.
- histSize: 히스토그램 각 차원의 크기(빈(bin)의 개수)를 나타내는 리스트 히스토그램 각 차원의 최솟값과 최댓값으로 구성된 리스트
- ranges: 계산된 히스토그램 ([numpy.ndarray](#))
- hist: 기존의 hist 히스토그램에 누적하려면 True, 새로 만들려면 False.
- accumulate:

# 히스토그램 분석

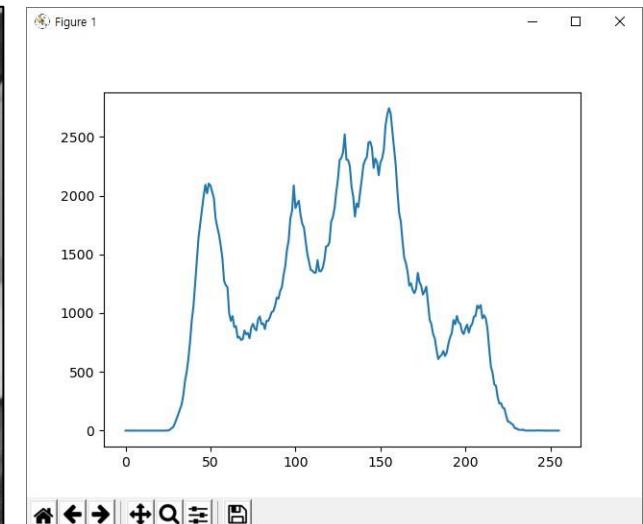
실습: histogram1.py

- 그레이스케일 영상의 히스토그램 구하기

```
src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

hist = cv2.calcHist([src], [0], None, [256], [0, 256])

plt.plot(hist)
plt.show()
```



# 히스토그램 분석

실습: histogram1.py

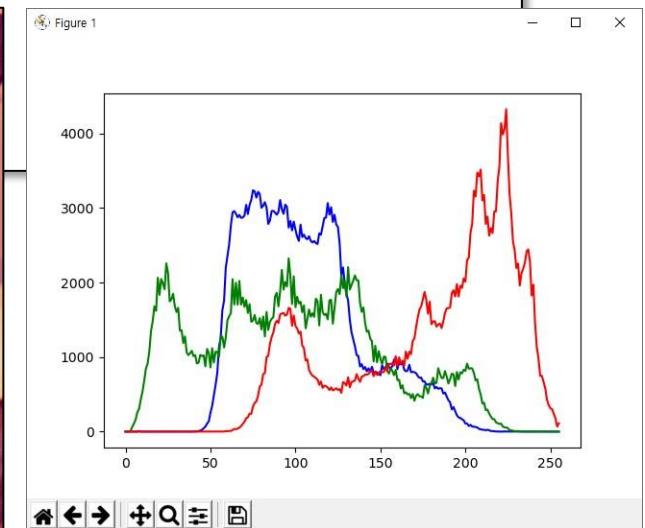
## ■ 컬러 영상의 히스토그램 구하기

```
src = cv2.imread('lenna.bmp')

colors = ['b', 'g', 'r']
bgr_planes = cv2.split(src)

for (p, c) in zip(bgr_planes, colors):
    hist = cv2.calcHist([p], [0], None, [256], [0, 256])
    plt.plot(hist, color=c)

plt.show()
```



# 히스토그램 분석

실습: histogram2.py

- OpenCV 그리기 함수로 그레이스케일 영상의 히스토그램 나타내기

```
def getGrayHistImage(hist):
    imgHist = np.full((100, 256), 255, dtype=np.uint8)

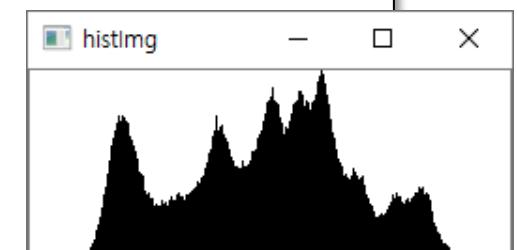
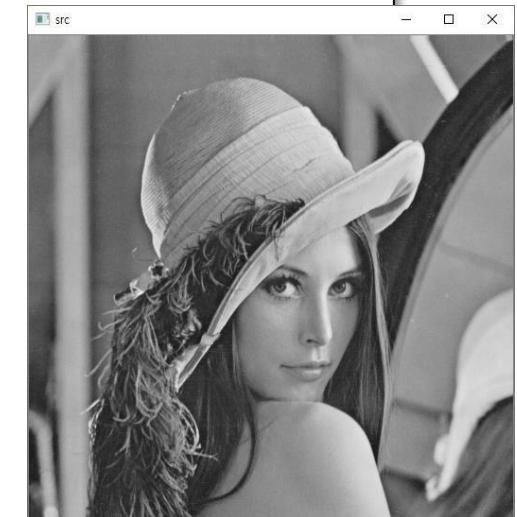
    histMax = np.max(hist)
    for x in range(256):
        pt1 = (x, 100)
        pt2 = (x, 100 - int(hist[x, 0] * 100 / histMax))
        cv2.line(imgHist, pt1, pt2, 0)

    return imgHist

src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

hist = cv2.calcHist([src], [0], None, [256], [0, 256])

histImg = getGrayHistImage(hist)
```



### 3. 기본적인 영상 처리 기법

#### 5) 영상의 명암비 조절

# 영상의 명암비 조절

## ■ 명암비(Contrast)란?

- 밝은 곳과 어두운 곳 사이에 드러나는 밝기 정도의 차이
- 컨트라스트, 대비



명암비가 낮은 영상

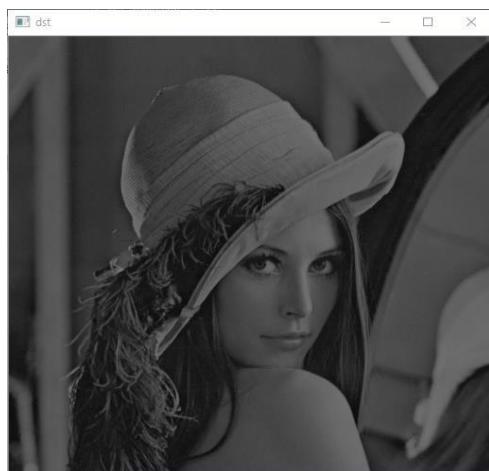


명암비가 높은 영상

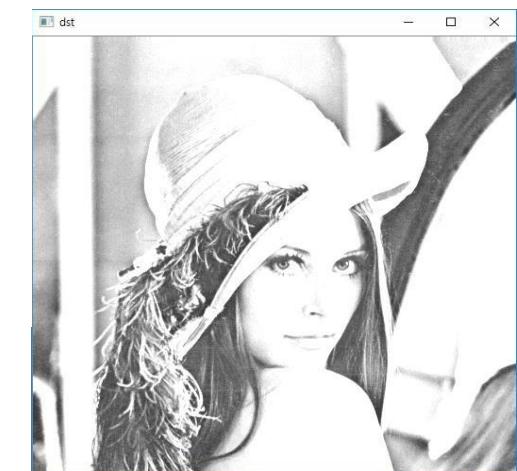
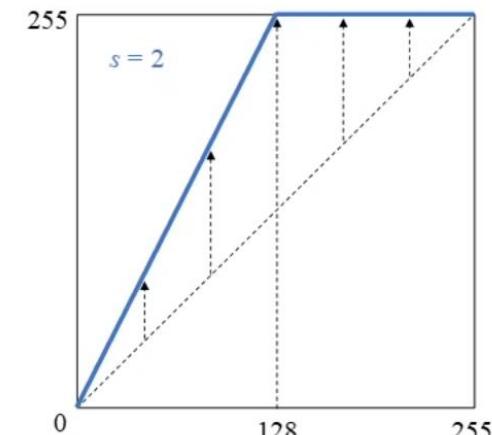
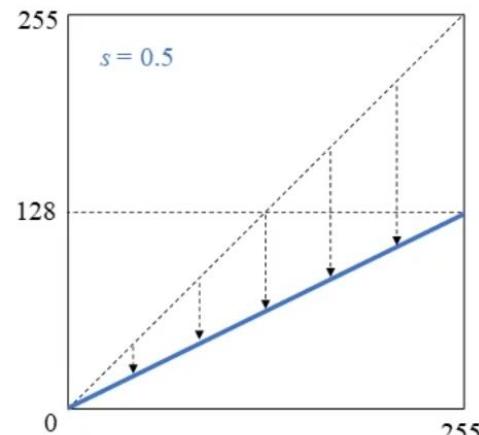
# 영상의 명암비 조절

## ■ 기본적인 명암비 조절 함수

$$dst(x, y) = \text{saturate}(s \cdot \text{src}(x, y))$$



$s = 0.5$

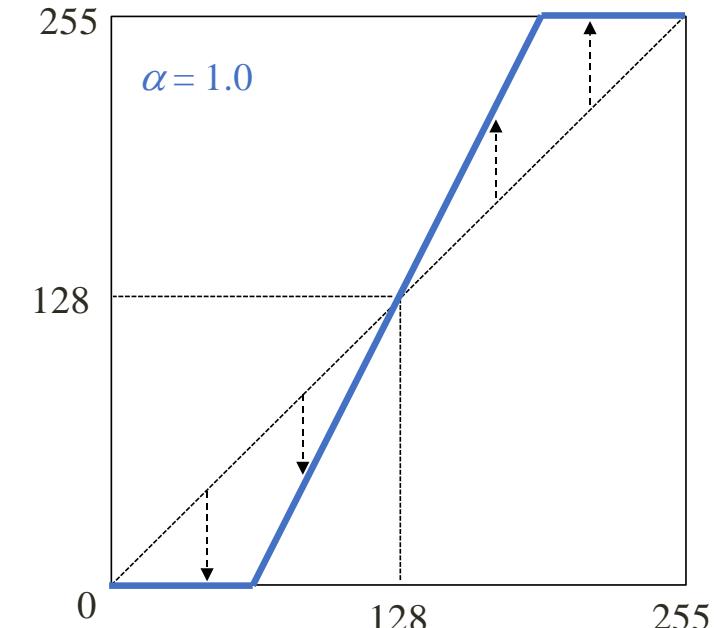
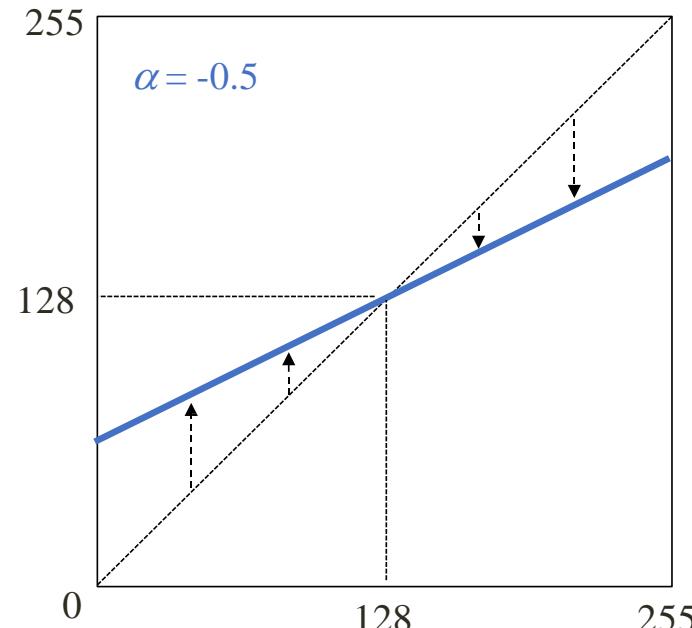


$s = 2.0$

# 영상의 명암비 조절

- 효과적인 명암비 조절 함수

$$\text{dst}(x, y) = \text{saturate}(\text{src}(x, y) + (\text{src}(x, y) - 128) \cdot \alpha)$$

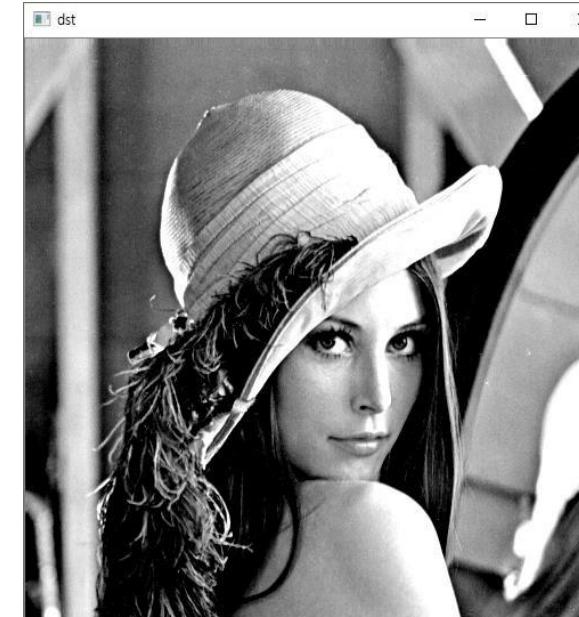
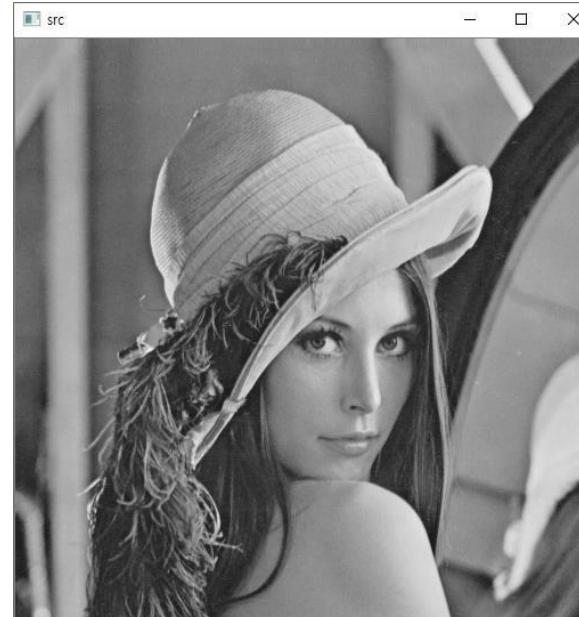


# 영상의 명암비 조절

실습: contrast1.py

## ■ 기본적인 명암비 조절 예제

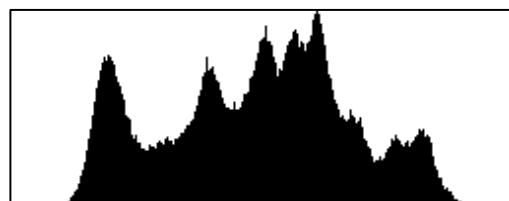
```
src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)  
  
alpha = 1.0  
dst = np.clip((1+alpha)*src - 128*alpha, 0, 255).astype(np.uint8)
```



# 영상의 자동 명암비 조절

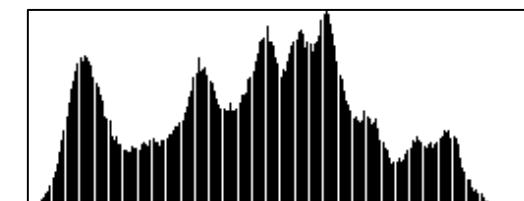
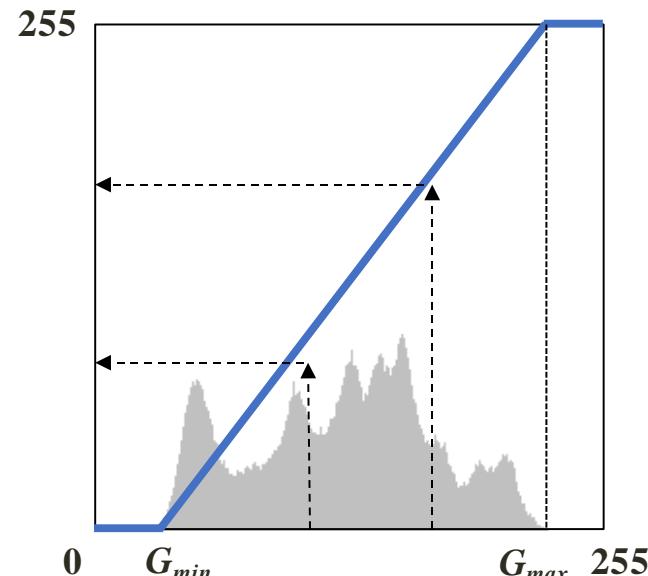
## ■ 히스토그램 스트레칭(Histogram stretching)

- 영상의 히스토그램이 그레이스케일 전 구간에서 걸쳐 나타나도록 변경하는 선형 변환기법



$G_{min}$

$G_{max}$



$G_{min} = 0$

$G_{max} = 255$

# 영상의 자동 명암비 조절

## ■ 정규화 함수

```
cv2.normalize(src, dst, alpha=None, beta=None, norm_type=None, dtype=None,  
mask=None) -> dst
```

- src: 입력 영상
- dst: 결과 영상
- alpha: (노름 정규화인 경우) 목표 노름 값,  
(원소 값 범위 정규화인 경우) 최솟값
- beta: (원소 값 범위 정규화인 경우) 최댓값
- norm\_type: 정규화 타입. NORM\_INF, NORM\_L1, NORM\_L2, NORM\_MINMAX.
- dtype: 결과 영상의 타입
- mask: 마스크 영상

# 영상의 자동 명암비 조절

실습: contrast2.py

## ■ 히스토그램 스트레칭을 이용한 명암비 자동 조절

```
src = cv2.imread('Hawkes.jpg', cv2.IMREAD_GRAYSCALE)  
  
dst = cv2.normalize(src, None, 0, 255, cv2.NORM_MINMAX)
```



# 영상의 자동 명암비 조절

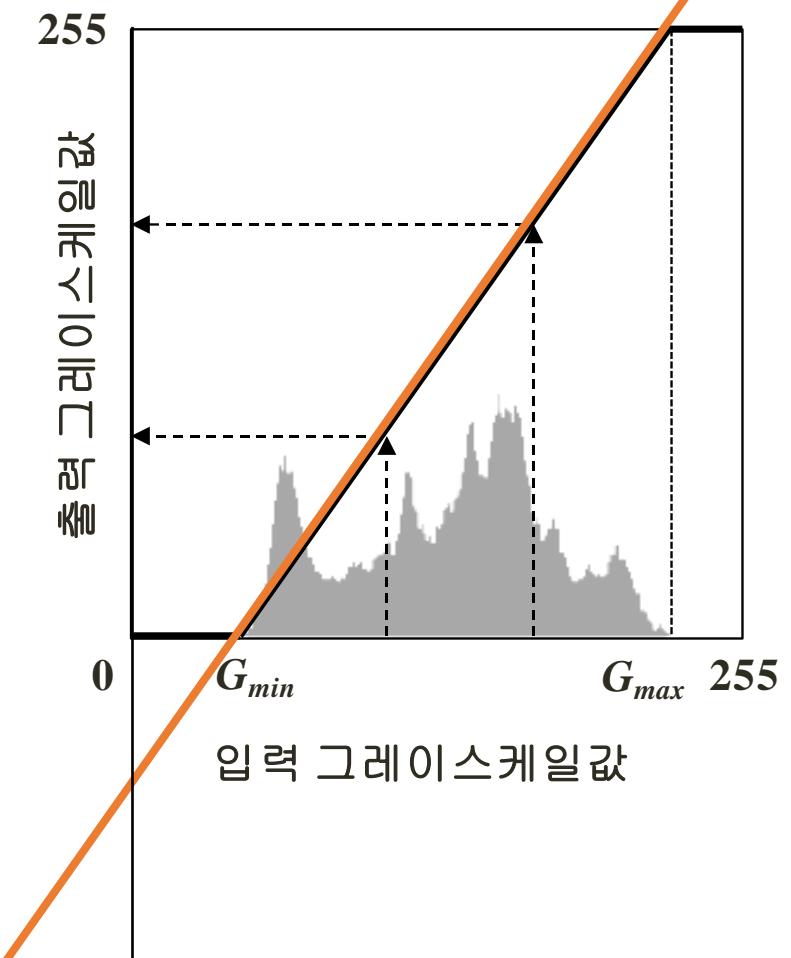
## ■ 히스토그램 스트레칭 변환 함수

- 변환 함수의 직선의 방정식 구하기

- 기울기:  $\frac{255}{G_{\max} - G_{\min}}$

- y 절편:  $-\frac{255 \times G_{\min}}{G_{\max} - G_{\min}}$

$$\begin{aligned}g(x, y) &= \frac{255}{G_{\max} - G_{\min}} \times f(x, y) - \frac{255 \times G_{\min}}{G_{\max} - G_{\min}} \\&= \frac{f(x, y) - G_{\min}}{G_{\max} - G_{\min}} \times 255\end{aligned}$$



### 3. 기본적인 영상 처리 기법

#### 6) 히스토그램 평활화

# 히스토그램 평활화

## ■ 히스토그램 평활화(Histogram equalization)

- 히스토그램이 그레이스케일 전체 구간에서 균일한 분포로 나타나도록 변경하는 명암비 향상 기법
- 히스토그램 균등화, 균일화, 평탄화

## ■ 히스토그램 평활화를 위한 변환 함수 구하기

- 히스토그램 함수 구하기:  $h(g) = N_g$
- 정규화된 히스토그램 함수 구하기:  $p(g) = \frac{h(g)}{w \times h}$
- 누적 분포 함수(cdf) 구하기:  $cdf(g) = \sum_{0 \leq i < g} p(i)$
- 변환 함수:  $dst(x, y) = round(cdf(src(x, y)) \times L_{\max})$

# 히스토그램 평활화

## ■ 히스토그램 평활화 계산 방법

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



bin	0	1	2	3	4	5	6	7
$h(g)$	4	3	2	1	0	2	3	1
$p(g)$	4/16	3/16	2/16	1/16	0	2/16	3/16	1/16



$cdf(g)$	4/16	7/16	9/16	10/16	10/16	12/16	15/16	1

2	2	2	4
3	3	4	2
3	5	7	5
4	7	7	7

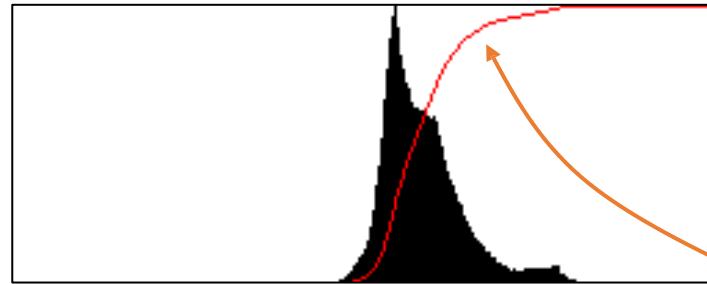


$cdf(g) * L_{max}$	1.75	3.06	3.94	4.38	4.38	5.25	6.56	7
	2	3	4	4	4	5	7	7

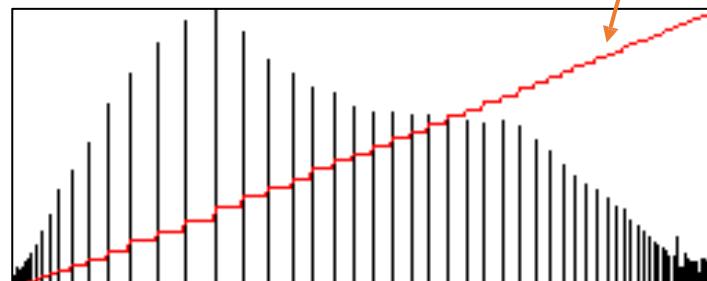
$$g(x, y) = cdf(f(x, y)) \times L_{max}$$

# 히스토그램 평활화

- 히스토그램 평활화와 히스토그램 누적 분포 함수와의 관계



히스토그램  
누적 분포 함수



[https://en.wikipedia.org/wiki/Histogram\\_equalization](https://en.wikipedia.org/wiki/Histogram_equalization)

# 히스토그램 평활화

## ■ 히스토그램 평활화

```
cv2.equalizeHist(src, dst=None) -> dst
```

- src: 입력 영상. 그레이스케일 영상.
- dst: 결과 영상.

# 히스토그램 평활화

실습: equalize.py

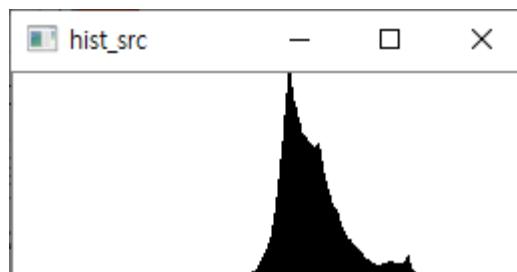
## ■ 히스토그램 평활화 예제

```
src = cv2.imread('Hawkes.jpg', cv2.IMREAD_GRAYSCALE)  
  
dst = cv2.equalizeHist(src)
```

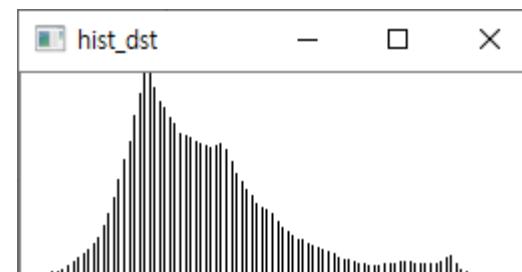


# 히스토그램 평활화

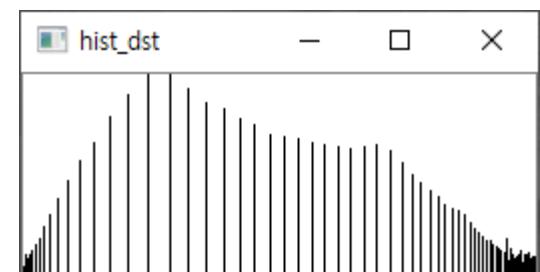
## ■ 히스토그램 스트레칭과 평활화 비교



입력 영상



히스토그램 스트레칭

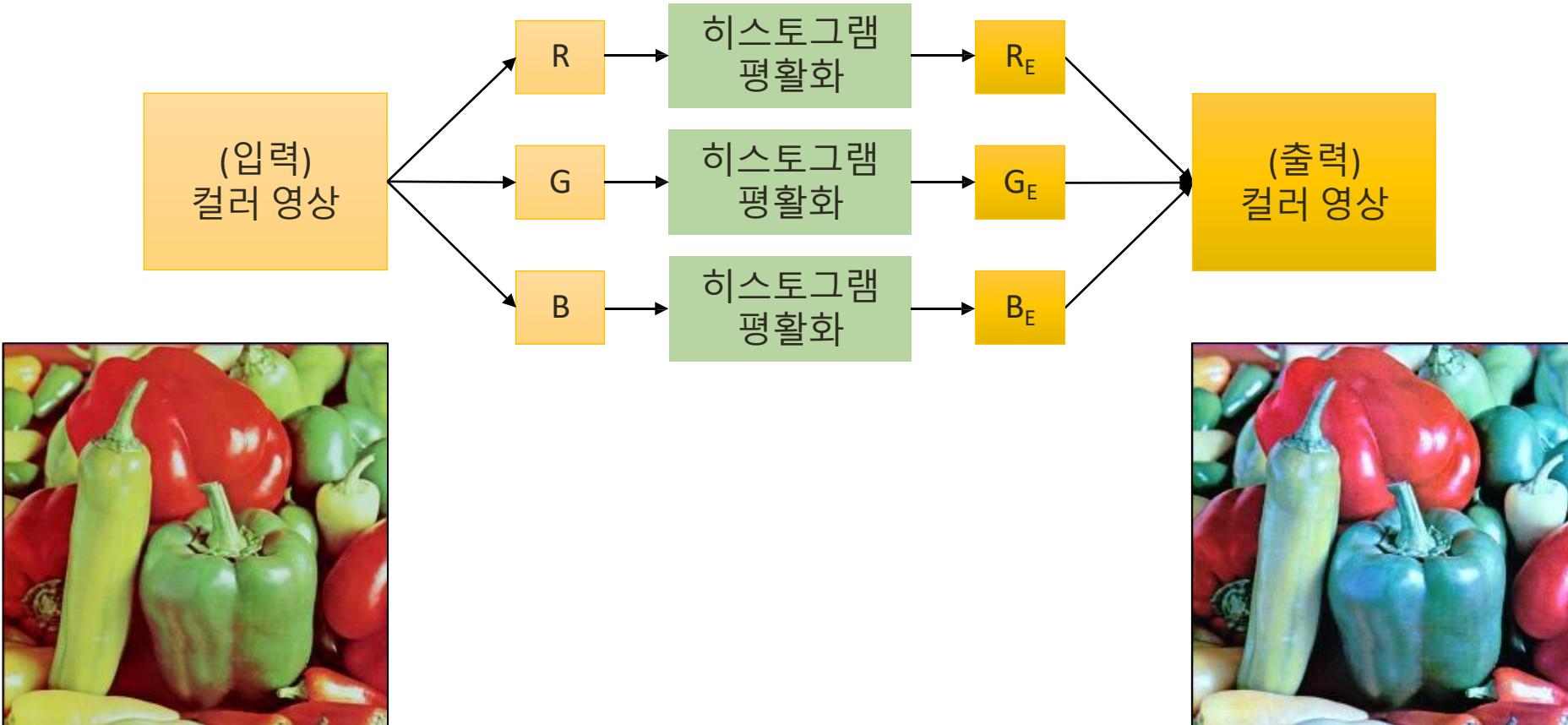


히스토그램 평활화

# 컬러 영상의 히스토그램 평활화

## ■ 컬러 히스토그램 평활화

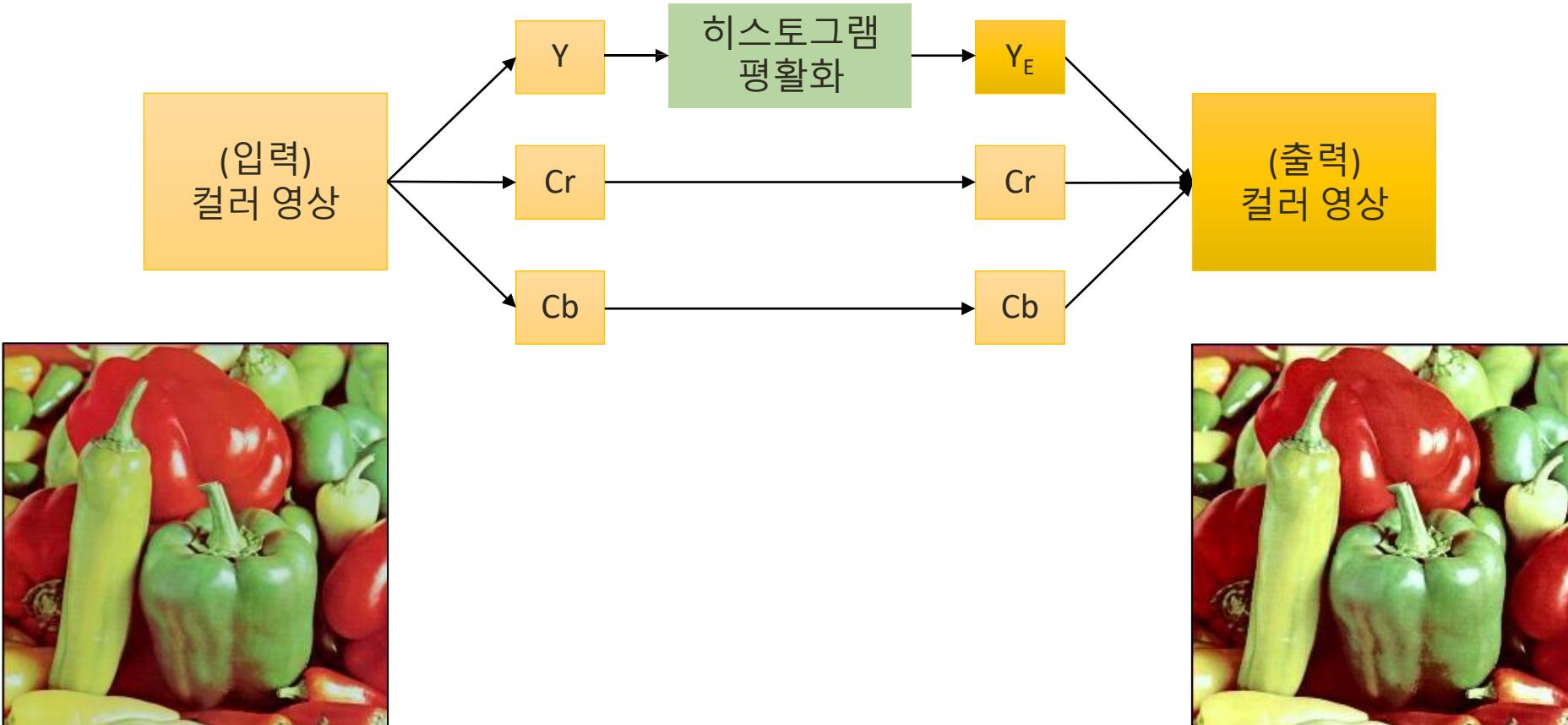
- 직관적 방법: R, G, B 각 색 평면에 대해 히스토그램 평활화



# 컬러 영상의 히스토그램 평활화

## ■ 컬러 히스토그램 평활화

- 밝기 성분에 대해서만 히스토그램 평활화 수행 (색상 성분은 불변)



# 컬러 영상의 히스토그램 평활화

실습: equalize.py

## ■ 컬러 영상의 히스토그램 평활화

```
src = cv2.imread('field.bmp')

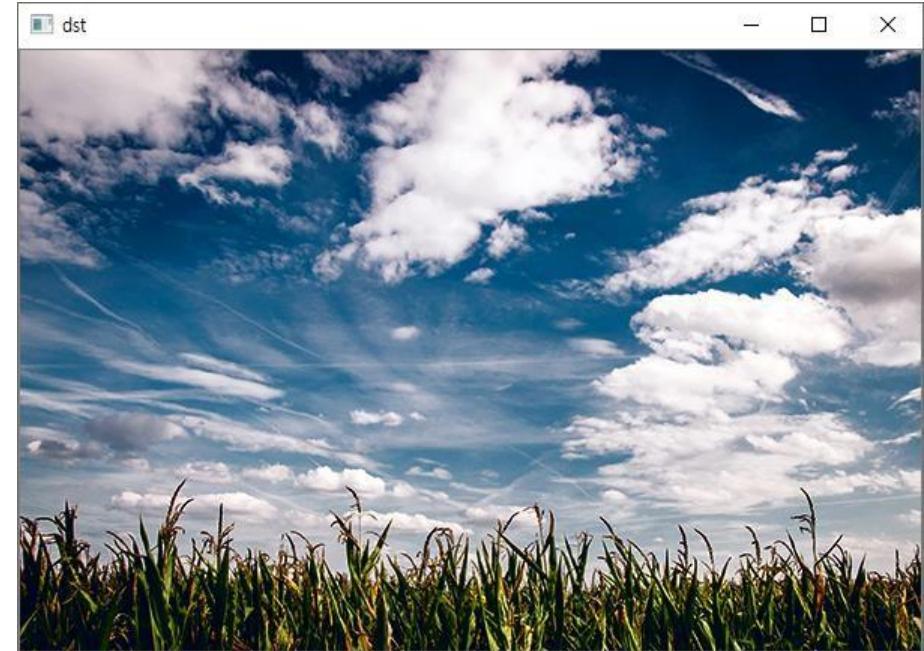
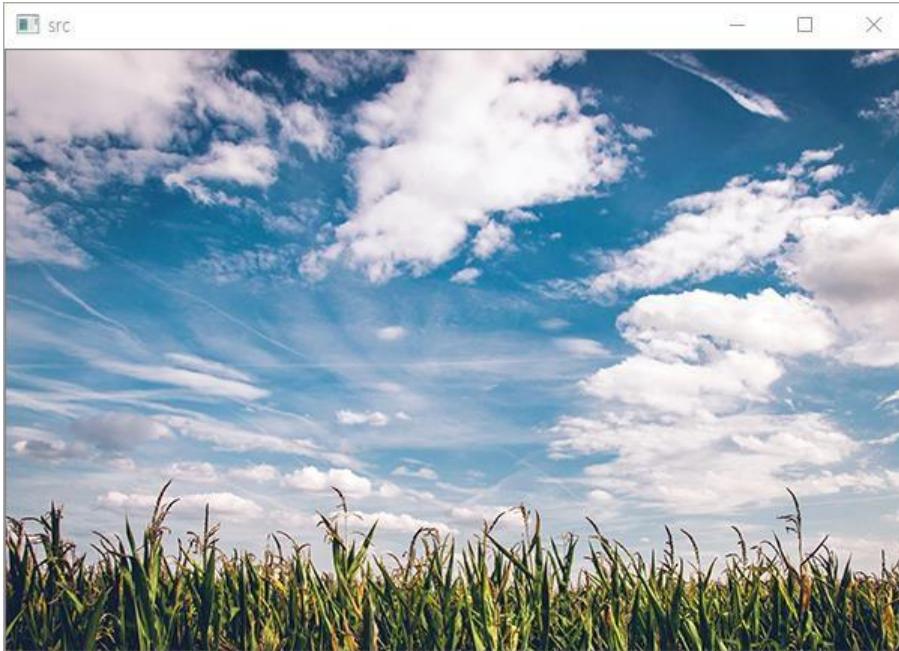
src_ycrcb = cv2.cvtColor(src, cv2.COLOR_BGR2YCrCb)
ycrcb_planes = cv2.split(src_ycrcb)

# 밝기 성분에 대해서만 히스토그램 평활화 수행
ycrcb_planes[0] = cv2.equalizeHist(ycrcb_planes[0])

dst_ycrcb = cv2.merge(ycrcb_planes)
dst = cv2.cvtColor(dst_ycrcb, cv2.COLOR_YCrCb2BGR)
```

# 컬러 영상의 히스토그램 평활화

## ■ 컬러 영상의 히스토그램 평활화 결과



### 3. 기본적인 영상 처리 기법

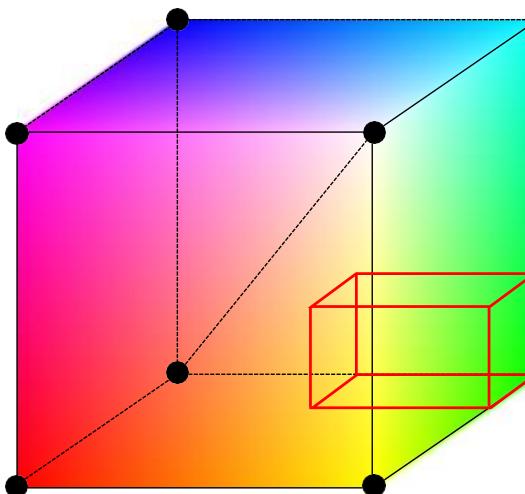
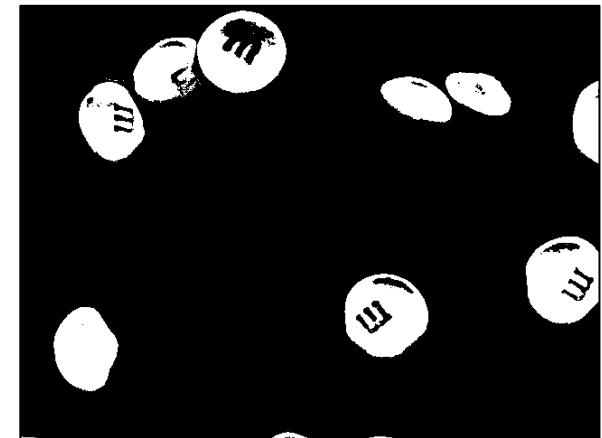
7) 특정 색상 영역 추출

# 특정 색상 영역 추출

- RGB 색 공간에서 녹색 영역 추출하기



$$\begin{aligned}0 &\leq R \leq 100 \\128 &\leq G \leq 255 \\0 &\leq B \leq 100\end{aligned}$$

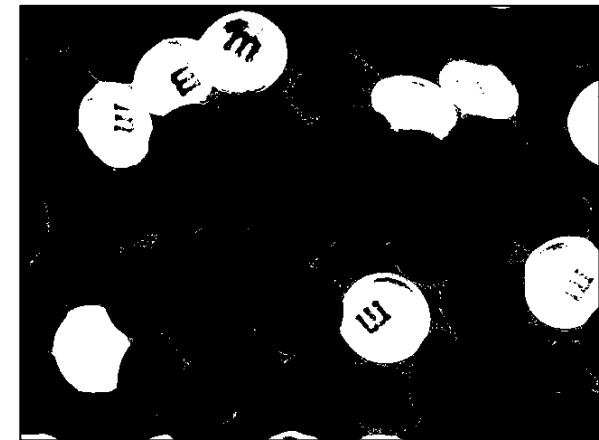
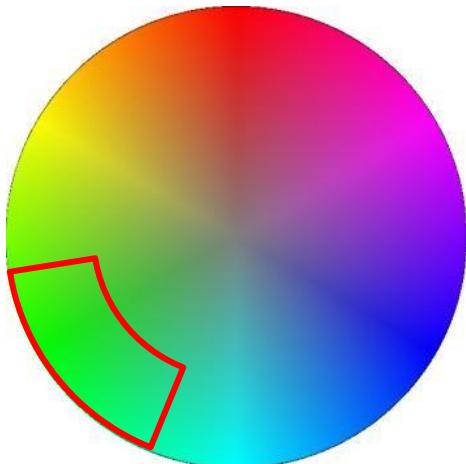


# 특정 색상 영역 추출

- HSV 색 공간에서 녹색 영역 추출하기



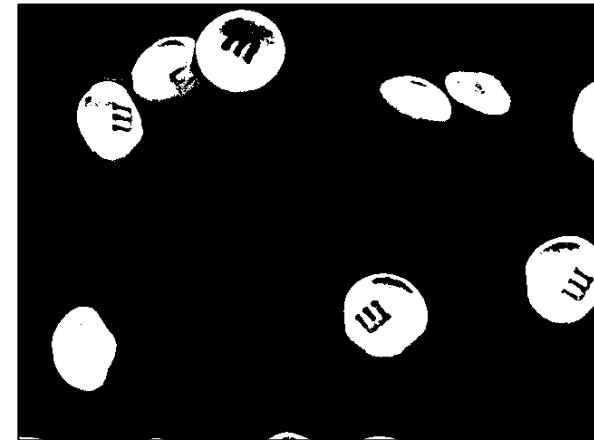
$50 \leq H \leq 80$   
 $150 \leq S \leq 255$   
 $0 \leq V \leq 255$



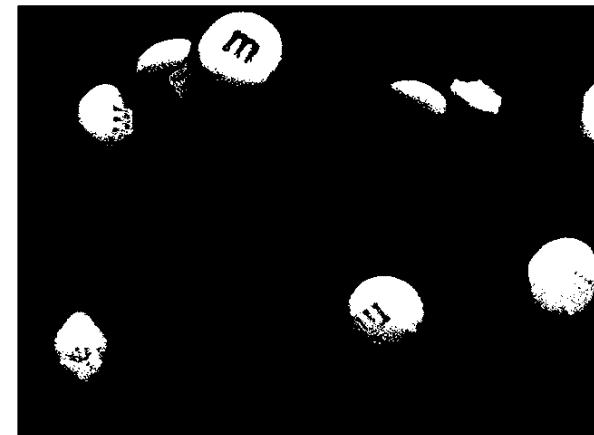
# 특정 색상 영역 추출

실습: inrange1.py

- RGB 색 공간에서 녹색 영역 추출하기



$$\begin{aligned}0 &\leq R \leq 100 \\128 &\leq G \leq 255 \\0 &\leq B \leq 100\end{aligned}$$



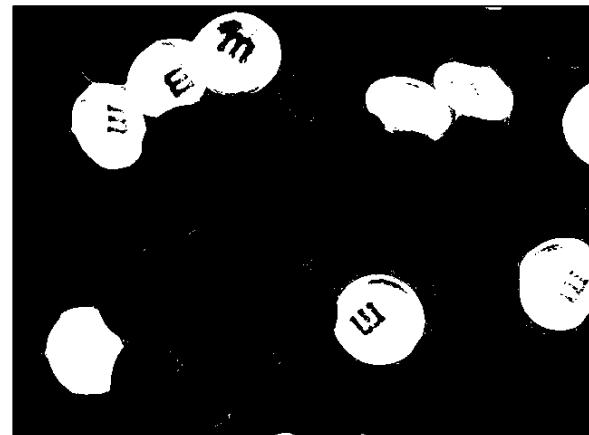
# 특정 색상 영역 추출

실습: inrange1.py

- HSV 색 공간에서 녹색 영역 추출하기



$$\begin{aligned}50 \leq H &\leq 80 \\150 \leq S &\leq 255 \\0 \leq V &\leq 255\end{aligned}$$



# 특정 색상 영역 추출

## ■ 특정 범위 안에 있는 행렬 원소 검출

```
cv2.inRange(src, lowerb, upperb, dst=None) -> dst
```

- src: 입력 행렬
- lowerb: 하한 값 행렬 또는 스칼라
- upperb: 상한 값 행렬 또는 스칼라
- dst: 입력 영상과 같은 크기의 마스크 영상. (numpy.uint8)  
범위 안에 들어가는 픽셀은 255, 나머지는 0으로 설정.

▪ 단일 채널:  $\text{dst}(I) = \text{lowerb}(I)_0 \leq \text{src}(I)_0 \leq \text{upperb}(I)_0$

▪ 다중 채널:  $\text{dst}(I) = \text{lowerb}(I)_0 \leq \text{src}(I)_0 \leq \text{upperb}(I)_0 \wedge$   
 $\text{lowerb}(I)_1 \leq \text{src}(I)_1 \leq \text{upperb}(I)_1 \wedge$   
...

# 특정 색상 영역 추출

실습: inrange2.py

## ■ 트랙바를 이용한 특정 색상 영역 추출

```
src = cv2.imread('candies.png')
src_hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)

def on_trackbar(pos):
    hmin = cv2.getTrackbarPos('H_min', 'dst')
    hmax = cv2.getTrackbarPos('H_max', 'dst')

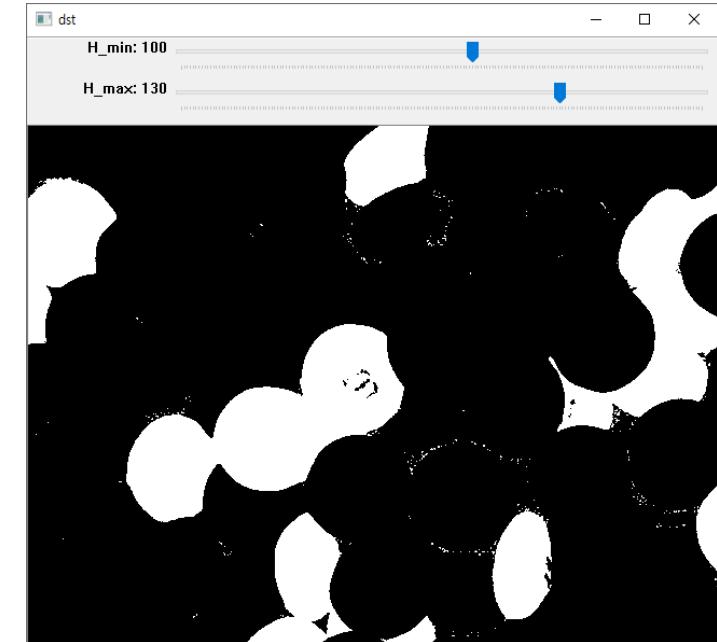
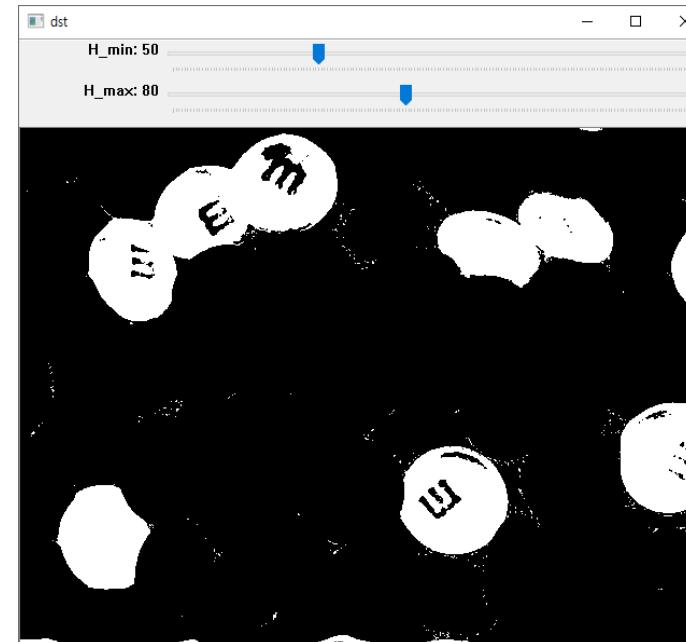
    dst = cv2.inRange(src_hsv, (hmin, 150, 0), (hmax, 255, 255))
    cv2.imshow('dst', dst)

cv2.imshow('src', src)
cv2.namedWindow('dst')

cv2.createTrackbar('H_min', 'dst', 50, 179, on_trackbar)
cv2.createTrackbar('H_max', 'dst', 80, 179, on_trackbar)
on_trackbar(0)
cv2.waitKey()
```

# 특정 색상 영역 추출

- 트랙바를 이용한 특정 색상 영역 추출 실행 결과



### 3. 기본적인 영상 처리 기법

8) 실전 코딩: 크로마 키 합성

# [실전 코딩] 크로마 키 합성

실습: chroma\_key.py

## ■ 크로마 키(Chroma key) 합성이란?

- 녹색 또는 파란색 배경에서 촬영한 영상에 다른 배경 영상을 합성하는 기술



## ■ 구현 할 기능

- 녹색 스크린 영역 추출하기
- 녹색 영역에 다른 배경 영상을 합성하여 저장하기
- 스페이스바를 이용하여 크로마 키 합성 동작 제어하기

# [실전 코딩] 크로마 키 합성

## ■ 녹색 스크린 영역 추출하기

- 크로마 키 영상을 HSV 색 공간으로 변환
- cv2.inRange() 함수를 사용하여  $50 \leq H \leq 80, 150 \leq S \leq 255, 0 \leq V \leq 255$  범위의 영역을 검출



# [실전 코딩] 크로마 키 합성

- 녹색 영역에 다른 배경 영상을 합성하기
  - 마스크 연산을 지원하는 cv2.copyTo() 함수 사용



## 4. 필터링

### 1) 필터링 이해하기

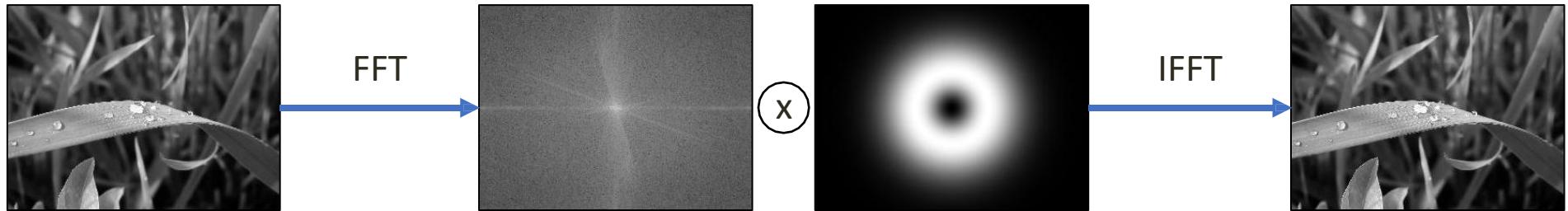
# 필터링 이해하기

- 영상의 필터링(image filtering)
  - 영상에서 필요한 정보만 통과시키고 원치 않는 정보는 걸러내는 작업



# 필터링 이해하기

## ■ 주파수 공간에서의 필터링 (Frequency domain filtering)

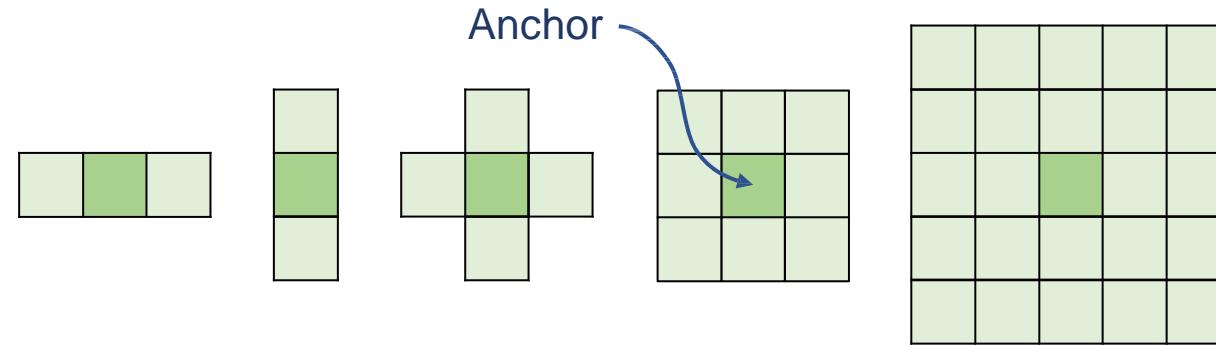


## ■ 공간적 필터링 (Spatial domain filtering)

- 영상의 픽셀 값을 직접 이용하는 필터링 방법
  - 대상 좌표의 픽셀 값과 주변 픽셀 값을 동시에 사용
- 주로 마스크(mask) 연산을 이용함  
(마스크 = 커널(kernel) = 윈도우(window) = 템플릿(template))

# 필터링 이해하기

## ■ 다양한 모양과 크기의 마스크

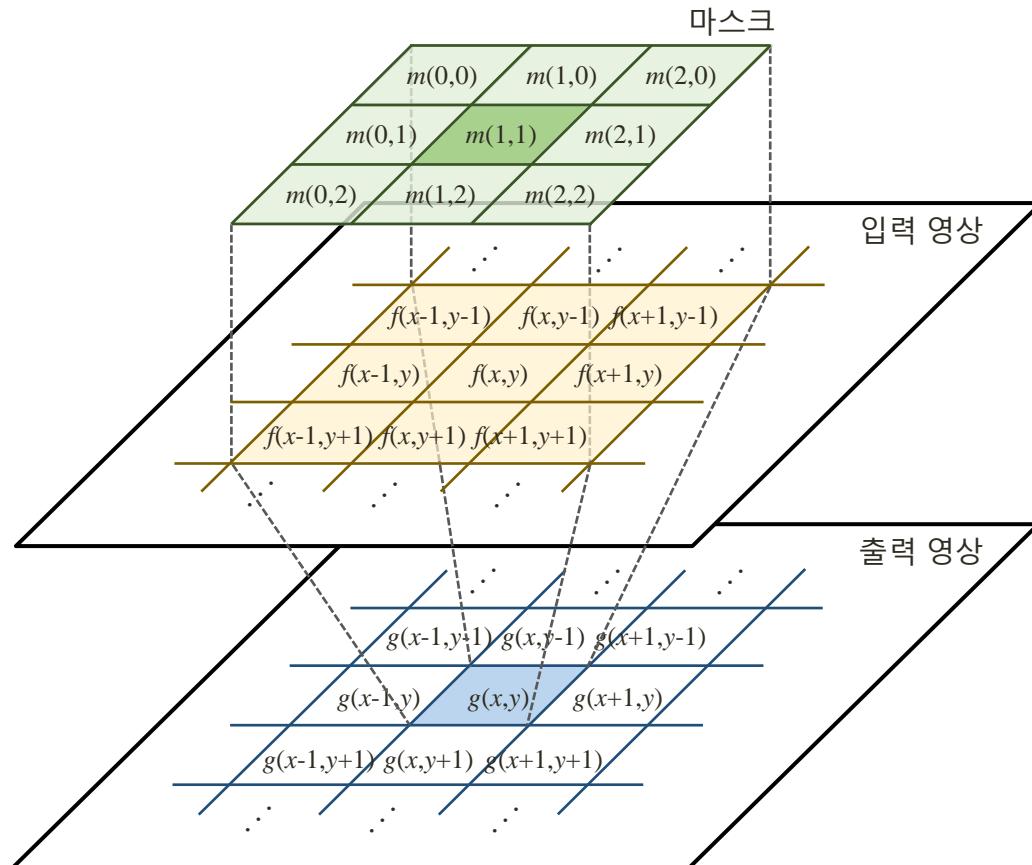


## ■ 마스크의 형태와 값에 따라 필터의 역할이 결정됨

- 영상 부드럽게 만들기
- 영상 날카롭게 만들기
- 에지(edge) 검출
- 잡음 제거

# 필터링 이해하기

## ■ 3×3 크기의 마스크를 이용한 공간적 필터링



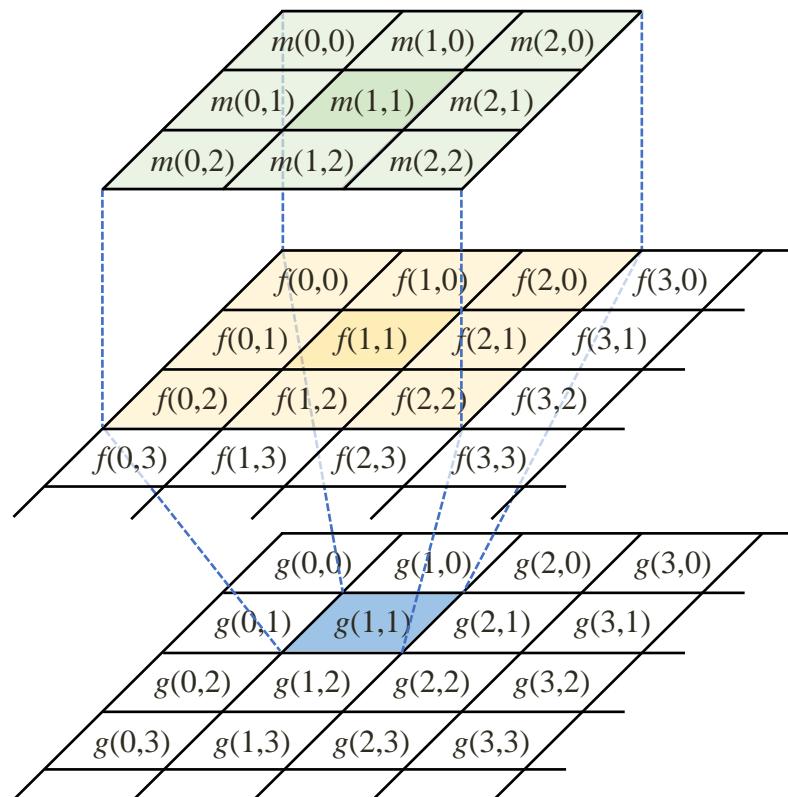
$$\begin{aligned}
 g(x, y) = & m(0,0)f(x-1, y-1) \\
 & +m(1,0)f(x, y-1) \\
 & +m(2,0)f(x+1, y-1) \\
 & +m(0,1)f(x-1, y) \\
 & +m(1,1)f(x, y) \\
 & +m(2,1)f(x+1, y) \\
 & +m(0,2)f(x-1, y+1) \\
 & +m(1,2)f(x, y+1) \\
 & +m(2,2)f(x+1, y+1)
 \end{aligned}$$

Correlation  
(Convolution)

$$g(x, y) = \sum_{j=0}^2 \sum_{i=0}^2 m(i, j) f(x + i - 1, y + j - 1)$$

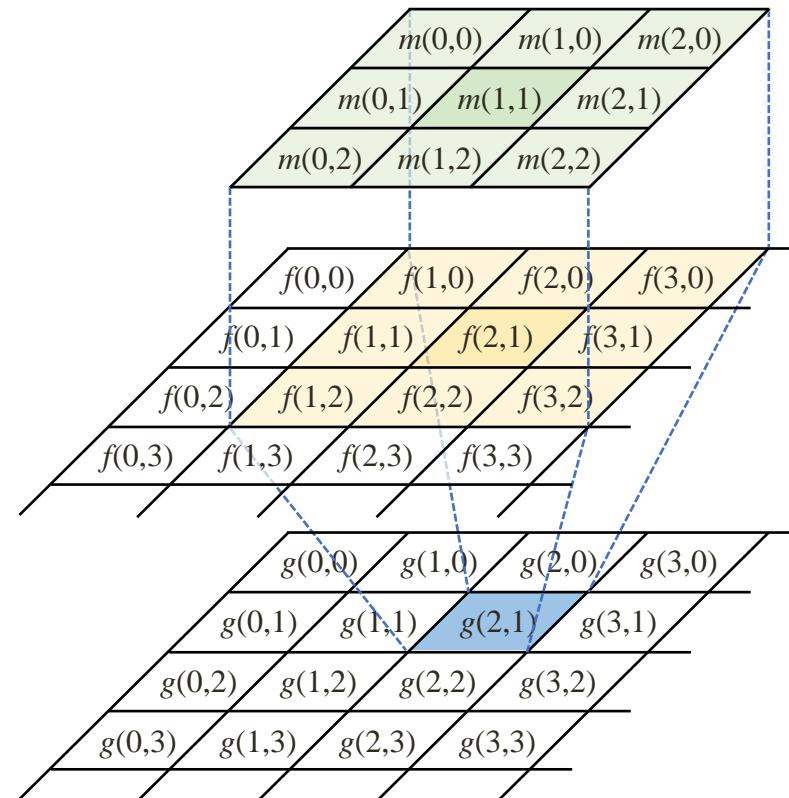
# 필터링 이해하기

- $3 \times 3$  크기의 마스크를 이용한 공간적 필터링 (Con't)
  - (1, 1) 좌표에서 필터링



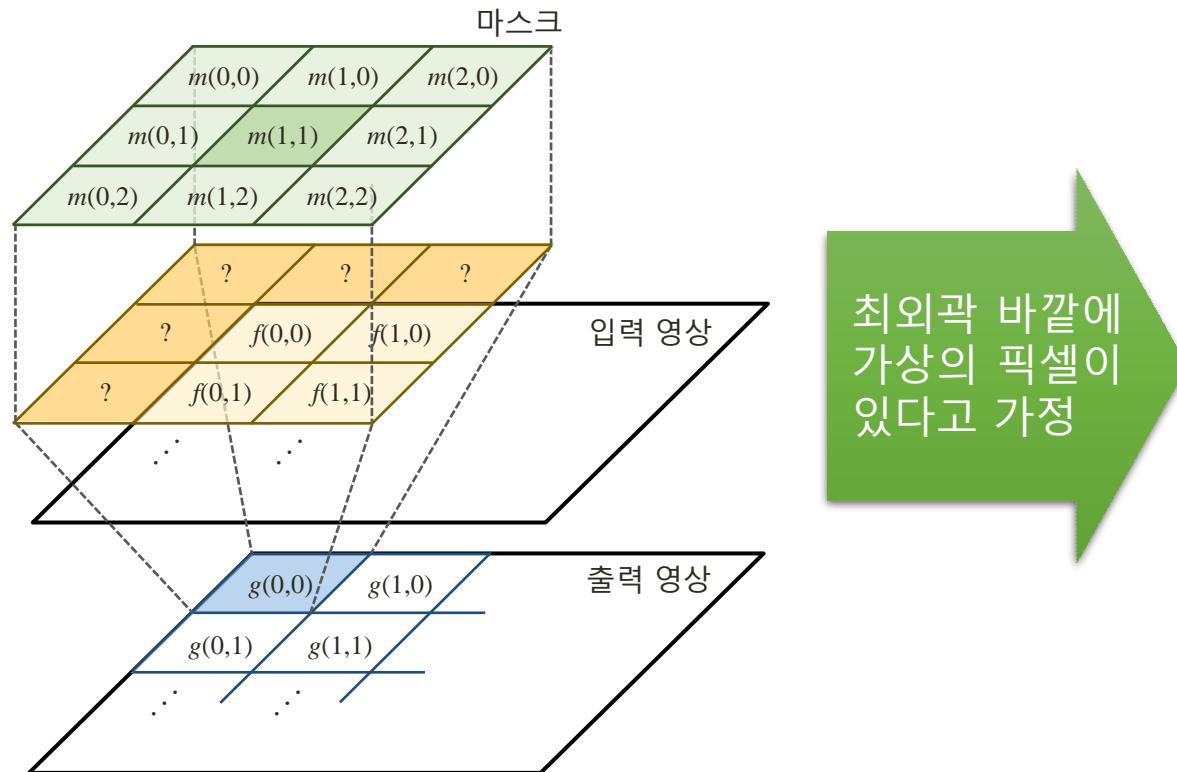
# 필터링 이해하기

- $3 \times 3$  크기의 마스크를 이용한 공간적 필터링 (Con't)
  - (2, 1) 좌표에서 필터링



# 필터링 이해하기

## ■ 최외곽 픽셀 처리



i	h	g	h	i	...
f	e	d	e	f	...
c	b	a	b	c	...
f	e	d	e	f	...
i	h	g	h	i	...
⋮	⋮	⋮	⋮	⋮	⋮

# 필터링 이해하기

## ■ OpenCV 필터링에서 지원하는 가장자리 픽셀 확장 방법

BorderTypes 열거형 상수	설명
BORDER_CONSTANT	
BORDER_REPLICATE	
BORDER_REFLECT	
BORDER_REFLECT_101	
BORDER_REFLECT101	BORDER_REFLECT_101과 같음
BORDER_DEFAULT	BORDER_REFLECT_101과 같음

# 필터링 이해하기

## ■ 기본적인 2D 필터링

```
cv2.filter2D(src, ddepth, kernel, dst=None, anchor=None, delta=None,  
            borderType=None) -> dst
```

- src: 입력 영상
- ddepth: 출력 영상 데이터 타입. (e.g) cv2.CV\_8U, cv2.CV\_32F, cv2.CV\_64F  
-1을 지정하면 src와 같은 타입의 dst 영상을 생성
- kernel: 필터 마스크 행렬. 실수형.
- anchor: 고정점 위치. (-1, -1)이면 필터 중앙을 고정점으로 사용
- delta: 추가적으로 더할 값
- borderType: 가장자리 픽셀 확장 방식
- dst: 출력 영상

## 4. 필터링

2) 블러링(1): 평균값 필터

# 블러링(1): 평균 값 필터

## ■ 평균 값 필터(Mean filter)

- 영상의 특정 좌표 값을 주변 픽셀 값들의 산술 평균으로 설정
- 픽셀들 간의 그레이스케일 값 변화가 줄어들어 날카로운 에지가 무뎌지고, 영상에 있는 잡음의 영향이 사라지는 효과

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

$$\frac{1}{25} \times \begin{array}{|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

# 블러링(1): 평균 값 필터

- 실제 영상에 평균 값 필터를 적용한 결과



원본 영상



3x3 크기의 마스크



5x5 크기의 마스크

- 마스크 크기가 커질수록 평균 값 필터 결과가 더욱 부드러워짐
- ⑦ 더 많은 연산량이 필요!

# 블러링(1): 평균 값 필터

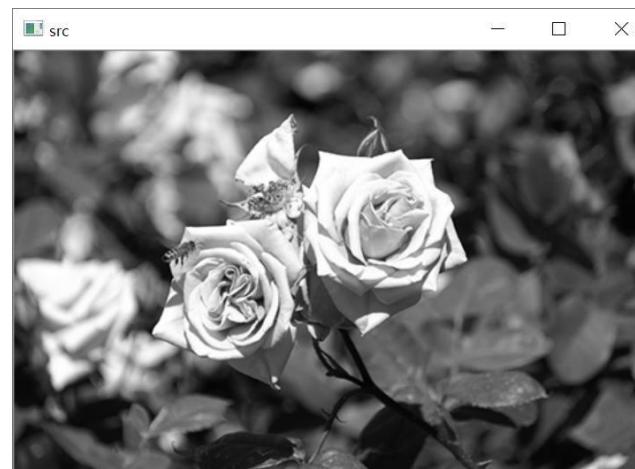
실습: blurring1.py

- filter2D() 함수를 이용한 평균값 필터링 예제

```
src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)

kernel = np.array([[1/9, 1/9, 1/9],
                  [1/9, 1/9, 1/9],
                  [1/9, 1/9, 1/9]])

dst = cv2.filter2D(src, -1, kernel)
```



# 블러링(1): 평균 값 필터

## ■ 평균 값 필터링 함수

```
cv2.blur(src, ksize, dst=None, anchor=None, borderType=None) -> dst
```

- src: 입력 영상
- ksize: 평균값 필터 크기. (width, height) 형태의 튜플.
- dst: 결과 영상. 입력 영상과 같은 크기 & 같은 타입.

$$\text{kernel} = \frac{1}{\text{ksize.width} \times \text{ksize.height}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

# 블러링(1): 평균 값 필터

실습: blurring2.py

- 다양한 크기의 커널을 사용한 평균값 필터링 예제

```
src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)

cv2.imshow('src', src)

for ksize in (3, 5, 7):
    dst = cv2.blur(src, (ksize, ksize))

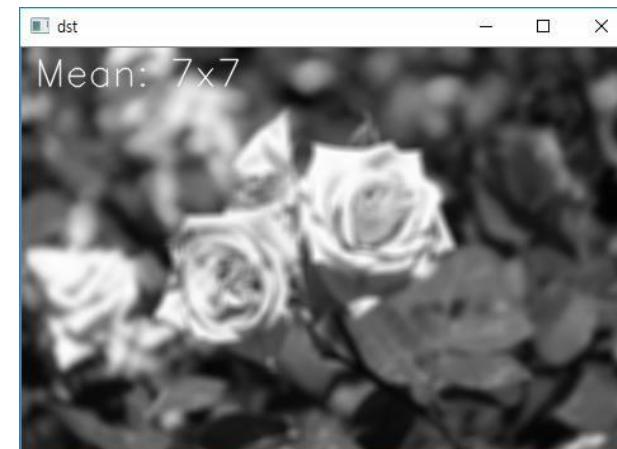
    desc = 'Mean: {}x{}'.format(ksize, ksize)
    cv2.putText(dst, desc, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                1.0, 255, 1, cv2.LINE_AA)

    cv2.imshow('dst', dst)
    cv2.waitKey()

cv2.destroyAllWindows()
```

# 블러링(1): 평균 값 필터

- 다양한 크기의 커널을 사용한 평균값 필터링 예제 결과



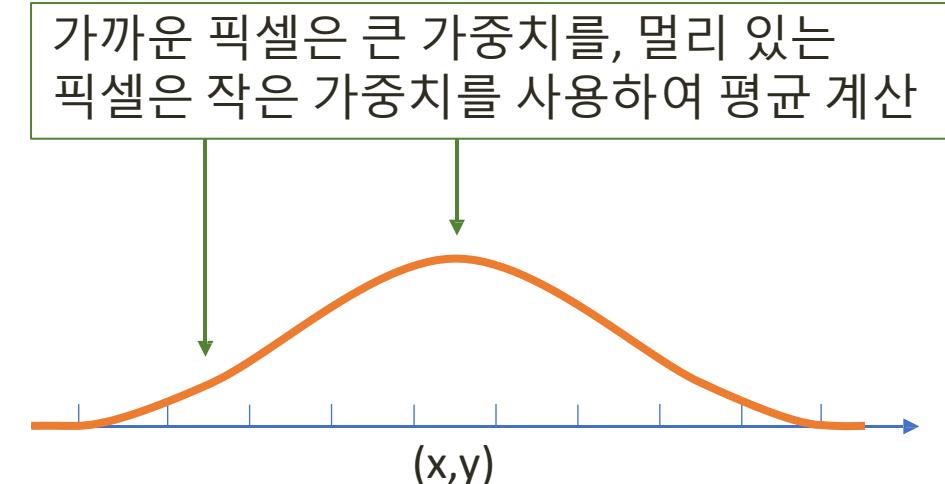
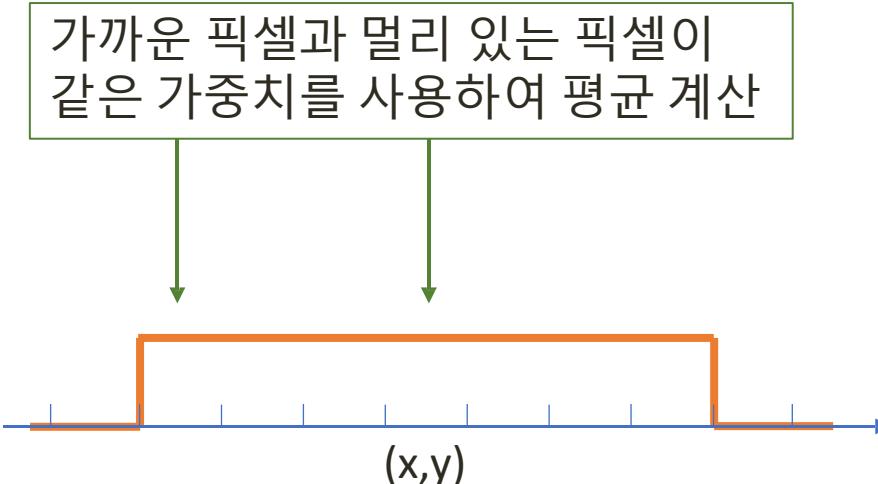
## 4. 필터링

3) 블러링(2): 가우시안 필터

# 블러링(2): 가우시안 필터

## ■ 평균값 필터에 의한 블러링의 단점

- 필터링 대상 위치에서 가까이 있는 픽셀과 멀리 있는 픽셀이 모두 같은 가중치를 사용하여 평균을 계산
- 멀리 있는 픽셀의 영향을 많이 받을 수 있음

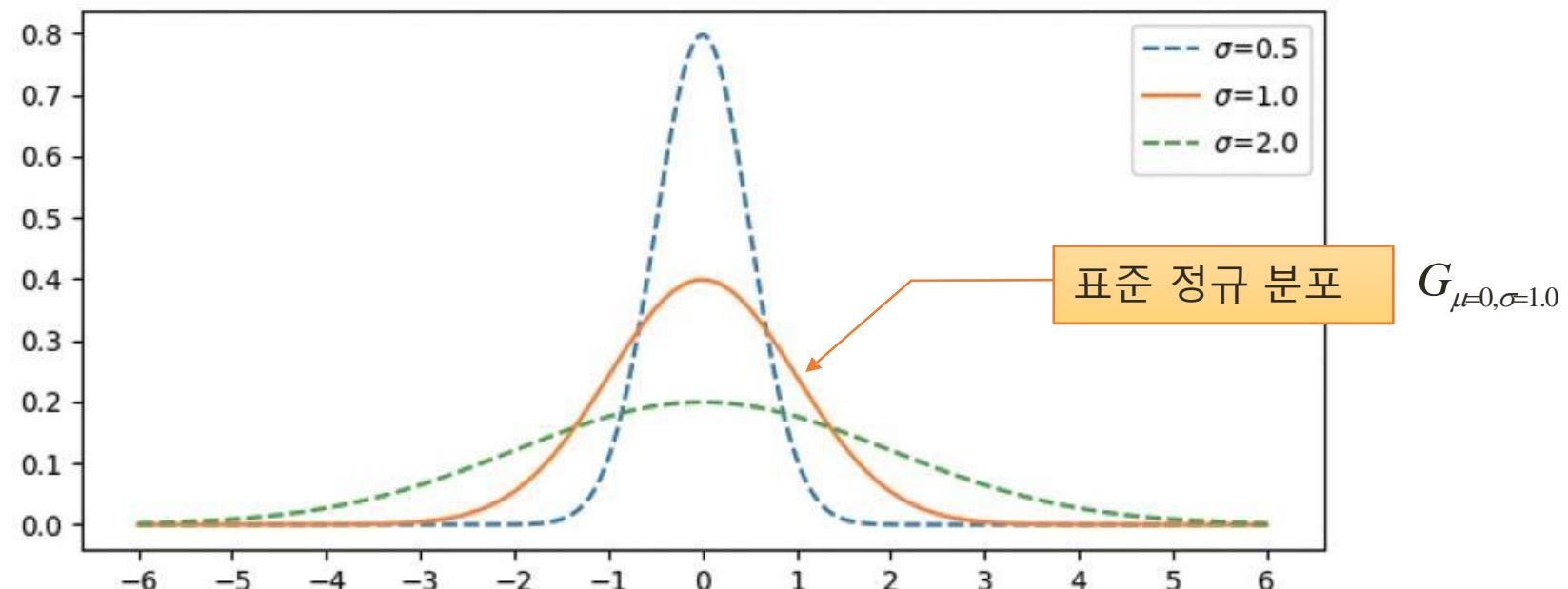


# 블러링(2): 가우시안 필터

## ■ (1차원) 가우시안 함수 (Gaussian function)

$$G_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- $\mu$ : 평균
- $\sigma$ : 표준편차

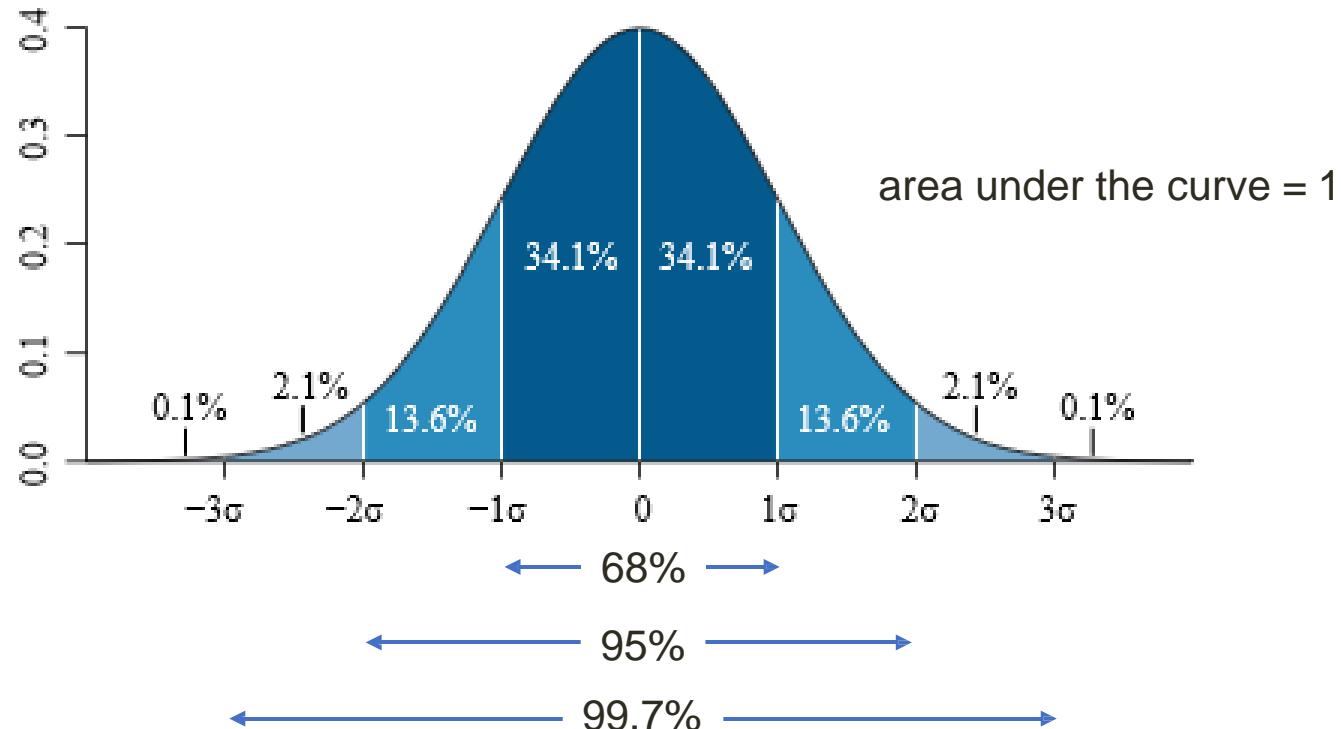


# 블러링(2): 가우시안 필터

## ■ 가우시안 함수의 특징

Symmetric (bell curve) shape around the mean

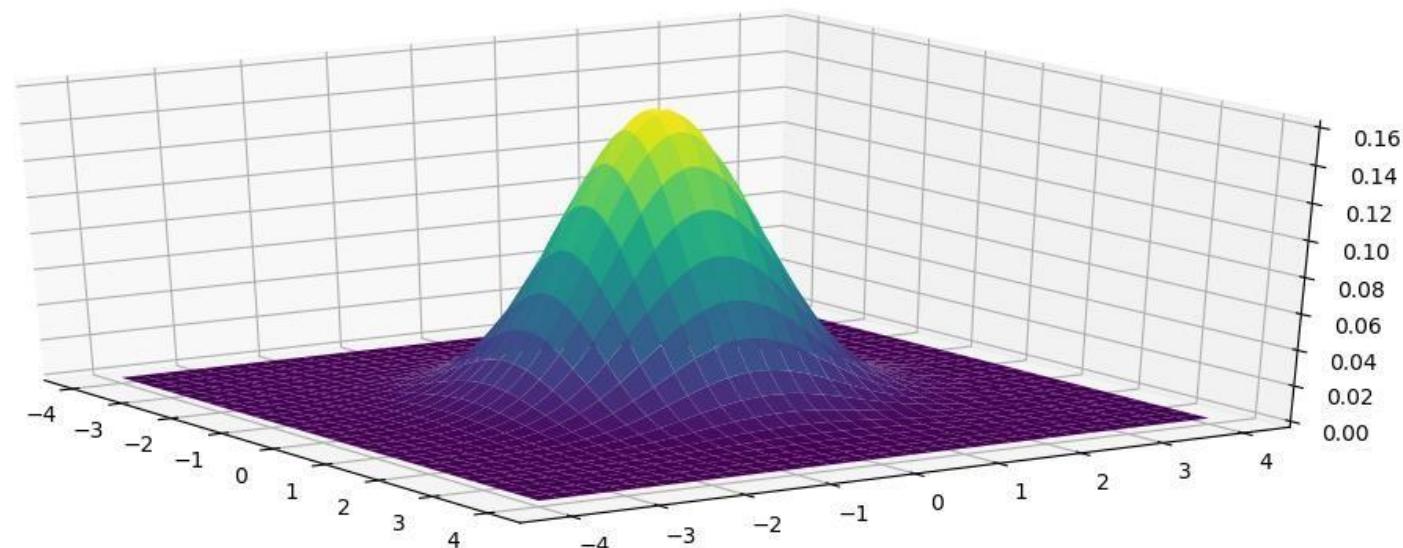
mean = median = mode



# 블러링(2): 가우시안 필터

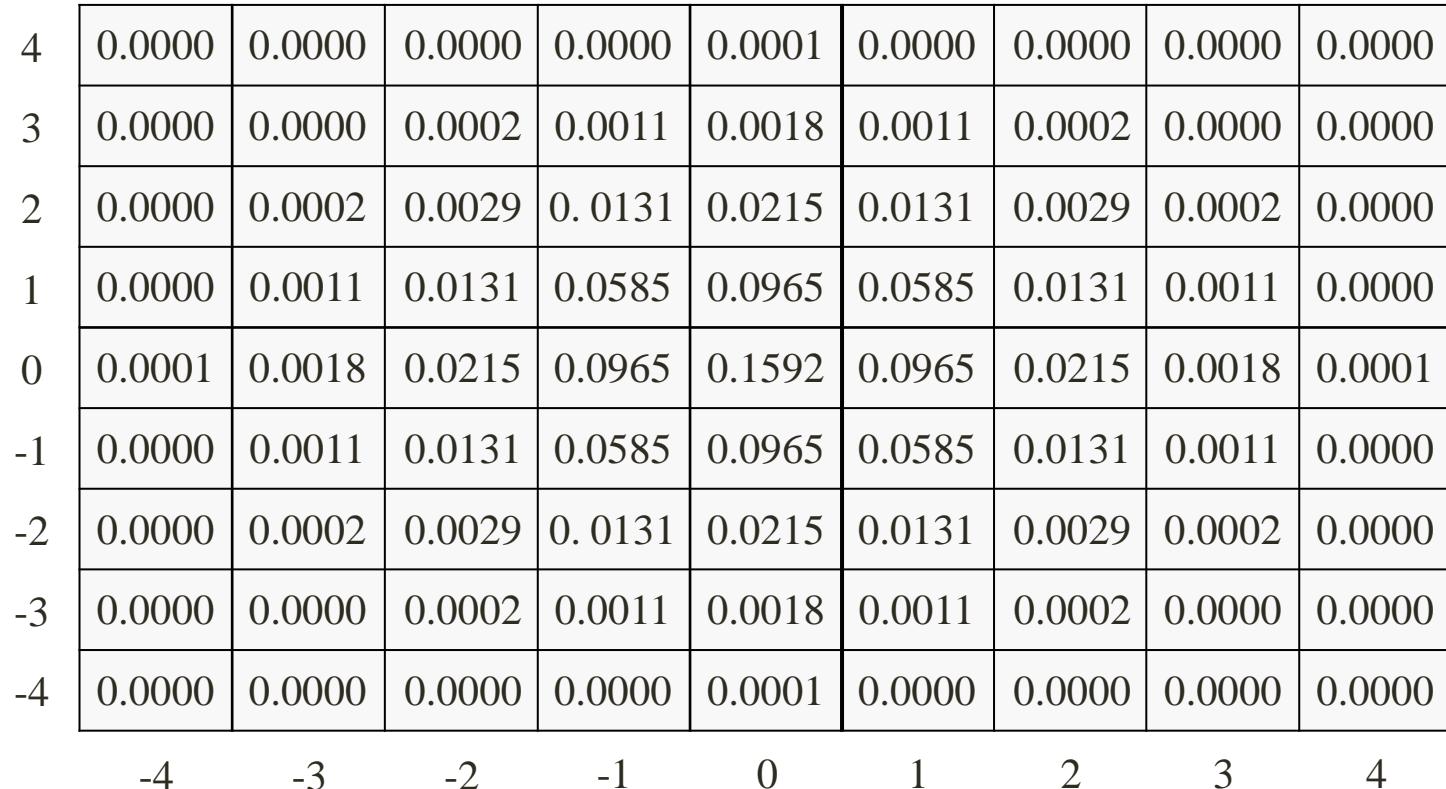
- 2차원 가우시안 함수 ( $\mu_x = \mu_y = 0, \sigma_x = \sigma_y = \sigma$ )

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}$$
$$\begin{cases} \mu_x = \mu_y = 0 \\ \sigma_x = \sigma_y = \sigma \end{cases}$$



# 블러링(2): 가우시안 필터

- 2차원 가우시안 필터 마스크 ( $\sigma=1.0$ )
  - 필터 마스크 크기:  $(8\sigma + 1)$  또는  $(6\sigma + 1)$



# 블러링(2): 가우시안 필터

## ■ 가우시안 필터링 함수

```
cv2.GaussianBlur(src, ksize, sigmaX, dst=None, sigmaY=None,  
                  borderType=None) -> dst
```

- src: 입력 영상. 각 채널 별로 처리됨.
- dst: 출력 영상. src와 같은 크기, 같은 타입.
- ksize: 가우시안 커널 크기. (0, 0)을 지정하면 sigma 값에 의해 자동 결정됨.
- sigmaX: x방향 sigma.
- sigmaY: y방향 sigma. 0이면 sigmaX와 같게 설정.
- borderType: 가장자리 픽셀 확장 방식.

# 블러링(2): 가우시안 필터

실습: gaussian.py

- 다양한 크기의 sigma를 사용한 가우시안 필터링

```
src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)

cv2.imshow('src', src)

for sigma in range(1, 6):
    dst = cv2.GaussianBlur(src, (0, 0), sigma)

    desc = 'sigma = {}'.format(sigma)
    cv2.putText(dst, desc, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                1.0, 255, 1, cv2.LINE_AA)

    cv2.imshow('dst', dst)
    cv2.waitKey()

cv2.destroyAllWindows()
```

# 블러링(2): 가우시안 필터

- 다양한 크기의 sigma를 사용한 가우시안 필터링 실행 결과



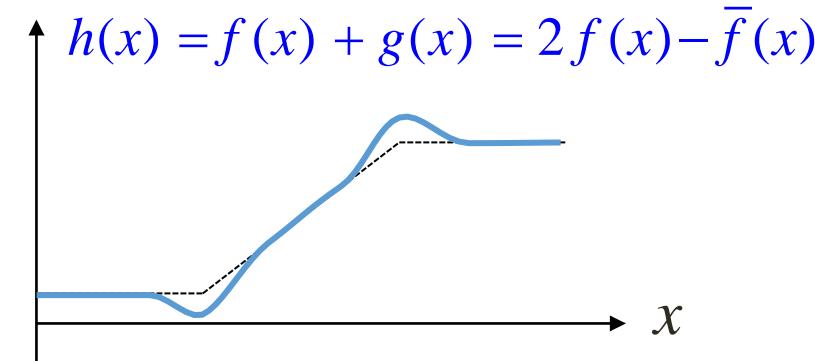
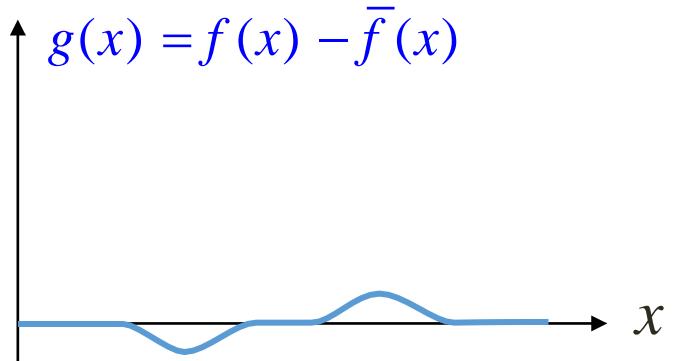
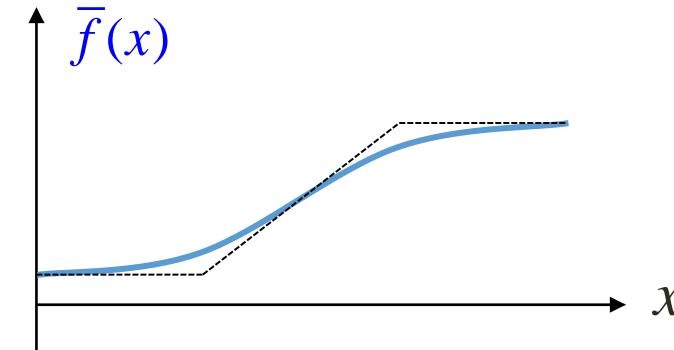
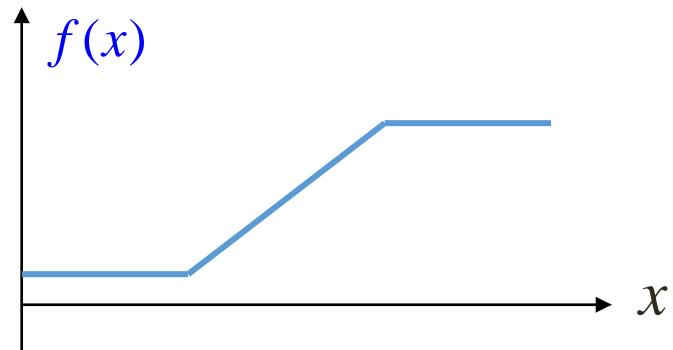
## **4. 필터링**

**4) 샤프닝: 언샤프 마스크 필터**

# 샤프닝: 언샤프 마스크 필터

## ■ 언샤프 마스크(Unsharp mask) 필터링

- 날카롭지 않은(unsharp) 영상, 즉, 부드러워진 영상을 이용하여 날카로운 영상을 생성



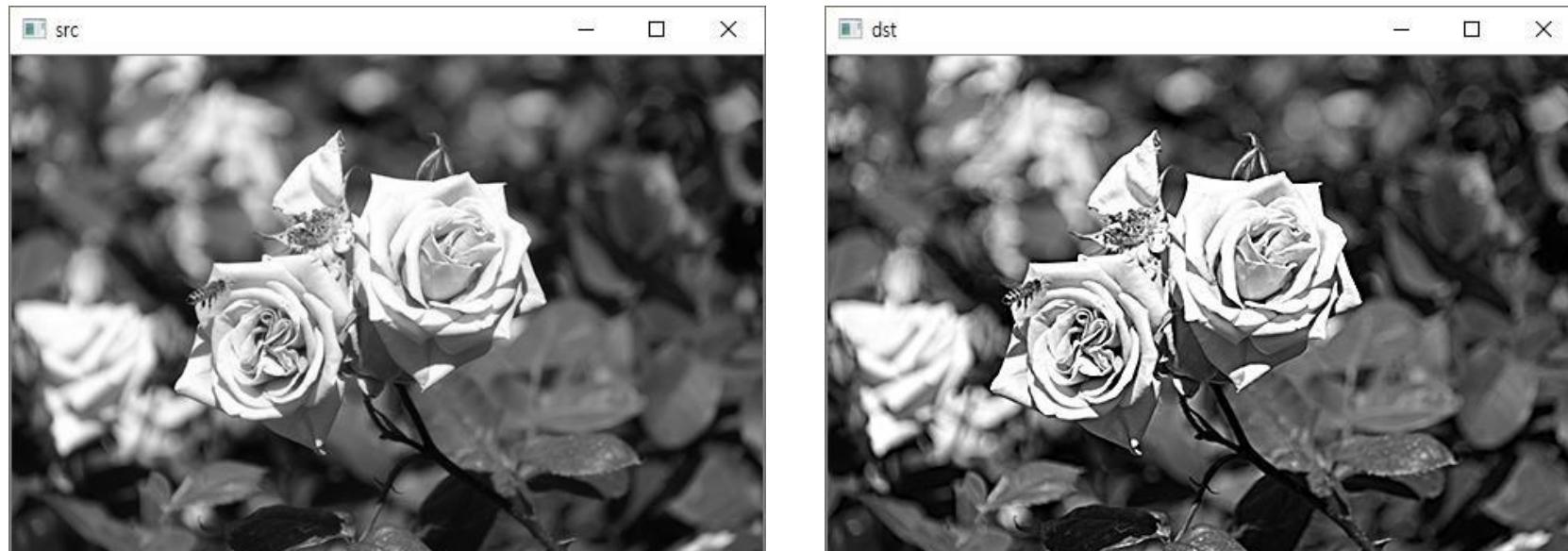
# 샤프닝: 언샤프 마스크 필터

실습: sharpening1.py

## ■ 언샤프 마스크 필터 구현하기

```
src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)

src_f = src.astype(np.float32)
blr = cv2.GaussianBlur(src_f, (0, 0), 2.0)
dst = np.clip(2. * src_f - blr, 0, 255).astype(np.uint8)
```



# 샤프닝: 언샤프 마스크 필터

실습: sharpening.py

- 언샤프 마스크 필터 구현하기
  - 샤프닝 정도를 조절할 수 있도록 수식 변경

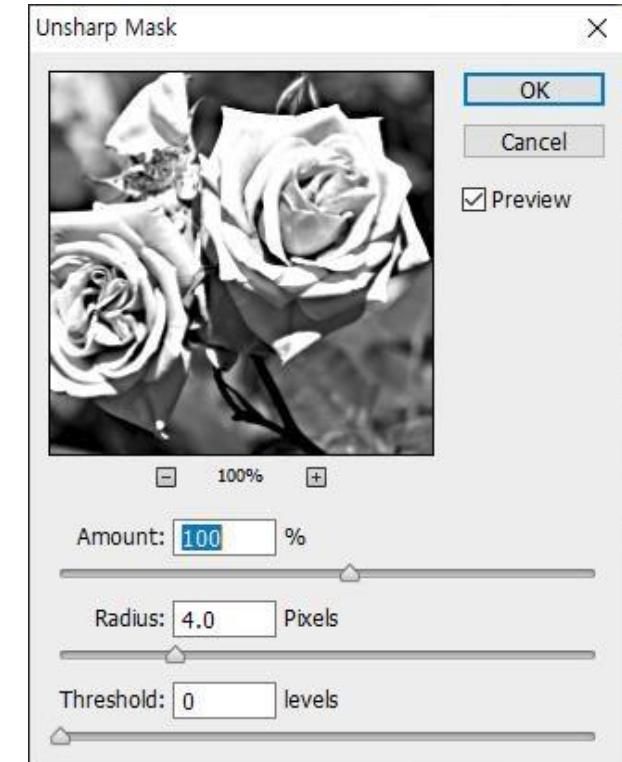
$$h(x, y) = f(x, y) + \alpha \cdot g(x, y)$$



$$\begin{aligned} h(x, y) &= f(x, y) + \alpha(f(x, y) - \bar{f}(x, y)) \\ &= (1 + \alpha)f(x, y) - \alpha \cdot \bar{f}(x, y) \end{aligned}$$



$$h(x, y) = (1 + \alpha)f(x, y) - \alpha \cdot G_\sigma(f(x, y))$$



# 샤프닝: 언샤프 마스크 필터

실습: sharpening2.py

- 컬러 영상에 대한 언샤프 마스크 필터 구현하기

```
src = cv2.imread('rose.bmp')

src_ycrcb = cv2.cvtColor(src, cv2.COLOR_BGR2YCrCb)

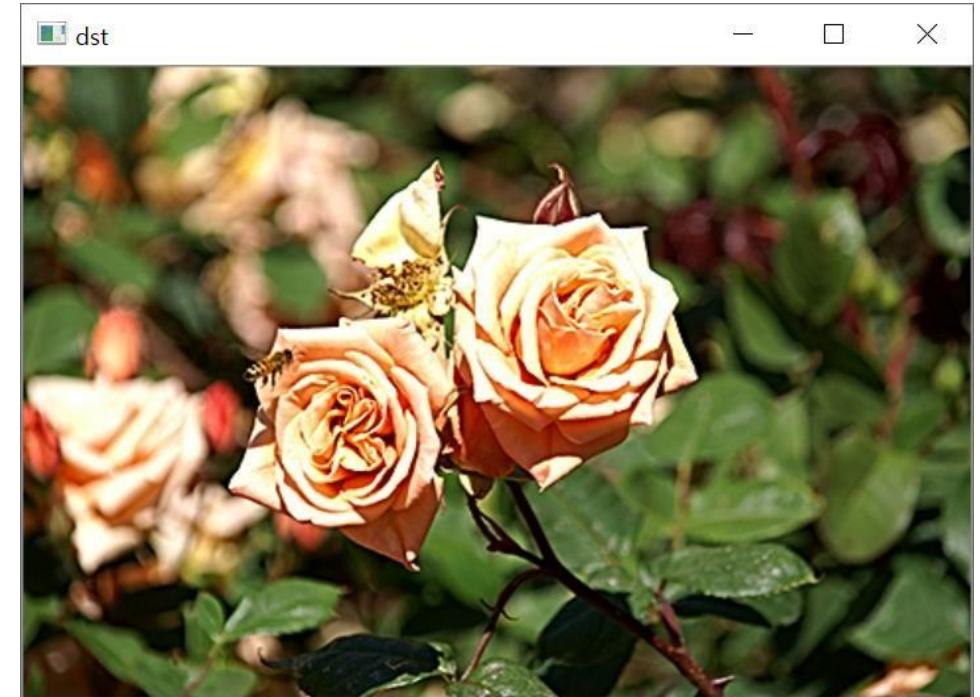
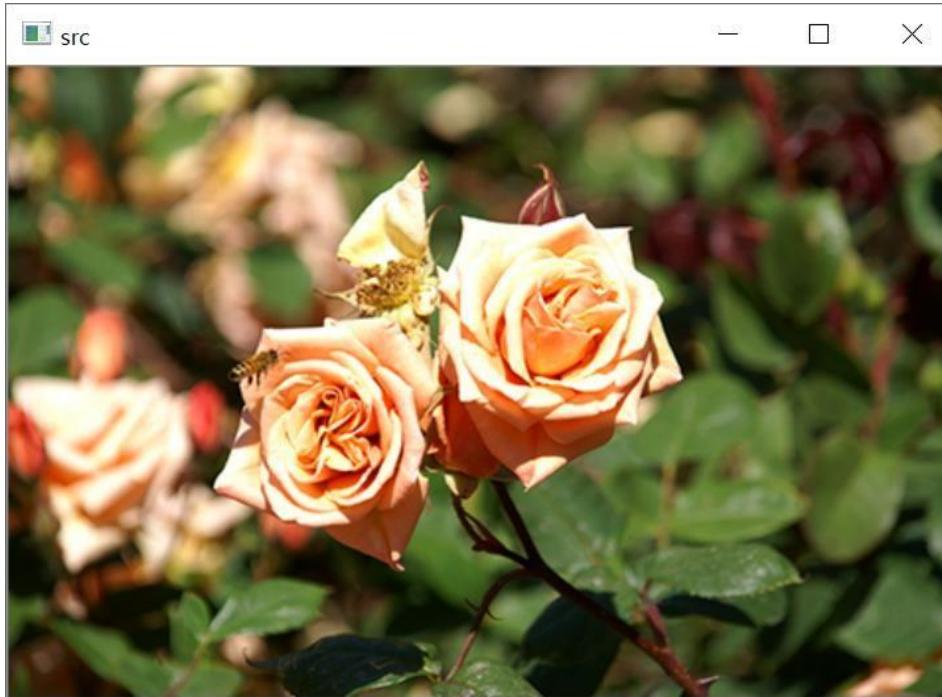
src_f = src_ycrcb[:, :, 0].astype(np.float32)
blr = cv2.GaussianBlur(src_f, (0, 0), 2.0)
src_ycrcb[:, :, 0] = np.clip(2. * src_f - blr, 0, 255).astype(np.uint8)

dst = cv2.cvtColor(src_ycrcb, cv2.COLOR_YCrCb2BGR)
```

# 샤프닝: 언샤프 마스크 필터

실습: sharpening2.py

- 컬러 영상에 대한 언샤프 마스크 필터 구현하기



## 4. 필터링

5) 잡음 제거(1): 미디언 필터

# 영상의 잡음

### ■ 영상의 잡음(Noise)

- 영상의 픽셀 값에 추가되는 원치 않는 형태의 신호

## ■ 잡음의 종류

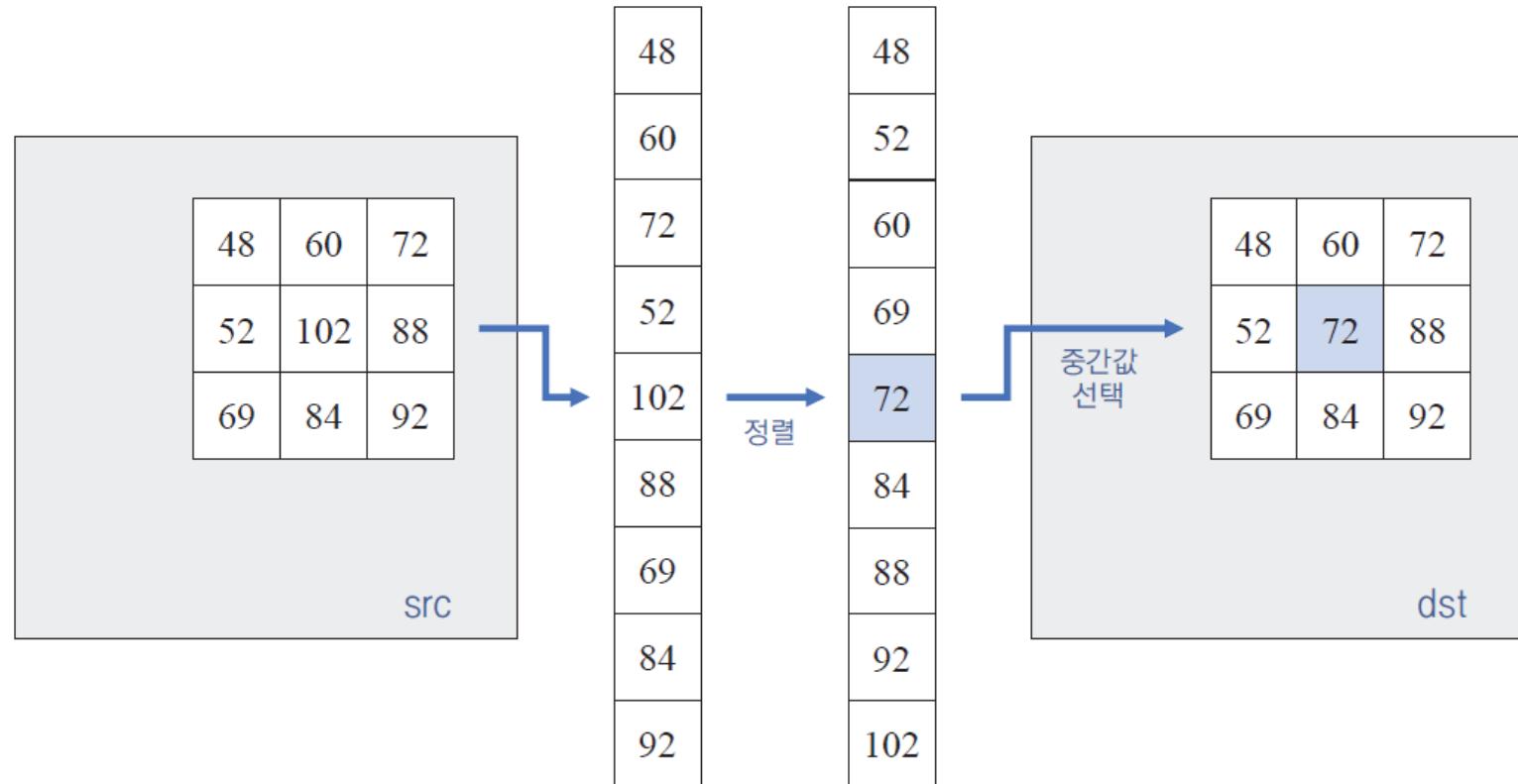
- 가우시안 잡음 (Gaussian noise)
  - 소금&후추 잡음 (Salt&Pepper)



# 잡음 제거(1): 미디언 필터

## ■ 미디언 필터(Median filter)

- 주변 픽셀들의 값을 정렬하여 그 중앙값(median)으로 픽셀 값을 대체
- 소금-후추 잡음 제거에 효과적



# 잡음 제거(1): 미디언 필터

## ■ 미디언 필터링 함수

```
cv2.medianBlur(src, ksize, dst=None) -> dst
```

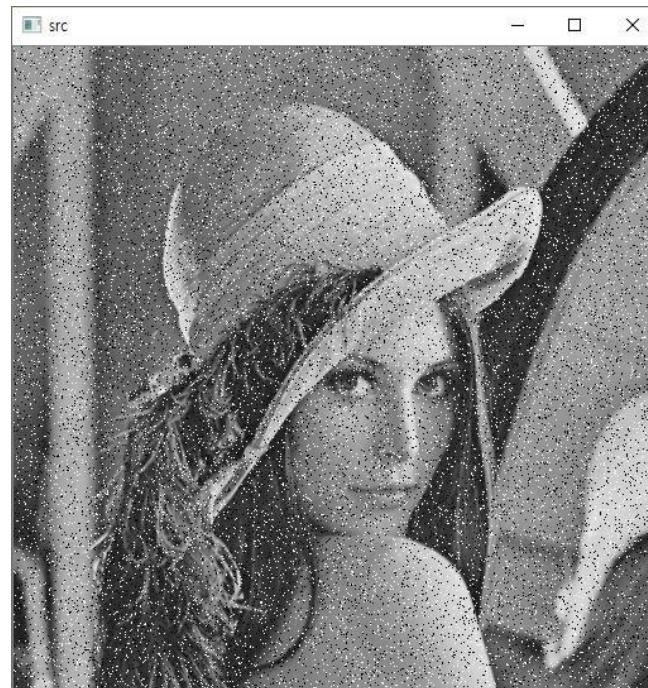
- src: 입력 영상. 각 채널 별로 처리됨.
- ksize: 커널 크기. 1보다 큰 홀수를 지정.
- dst: 출력 영상. src와 같은 크기, 같은 타입.

# 잡음 제거(1): 미디언 필터

실습: median.py

## ■ 미디언 필터링 예제

```
src = cv2.imread('noise.bmp', cv2.IMREAD_GRAYSCALE)  
  
dst = cv2.medianBlur(src, 3)
```

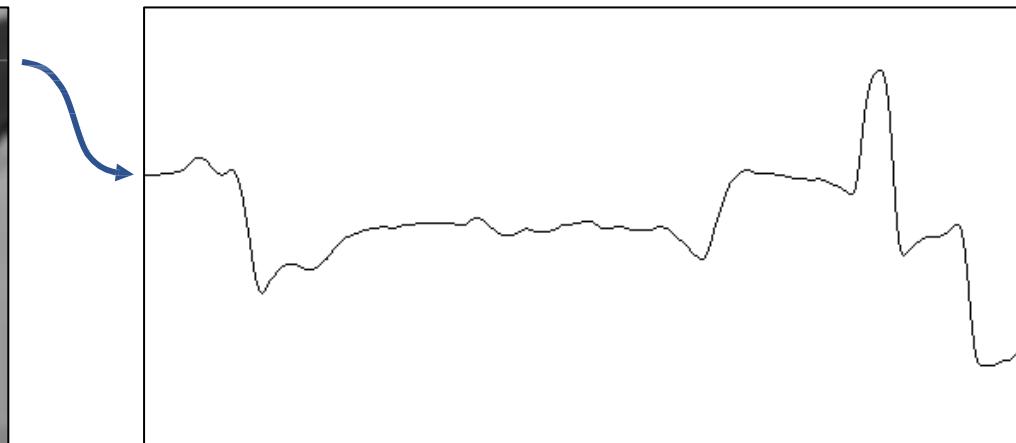
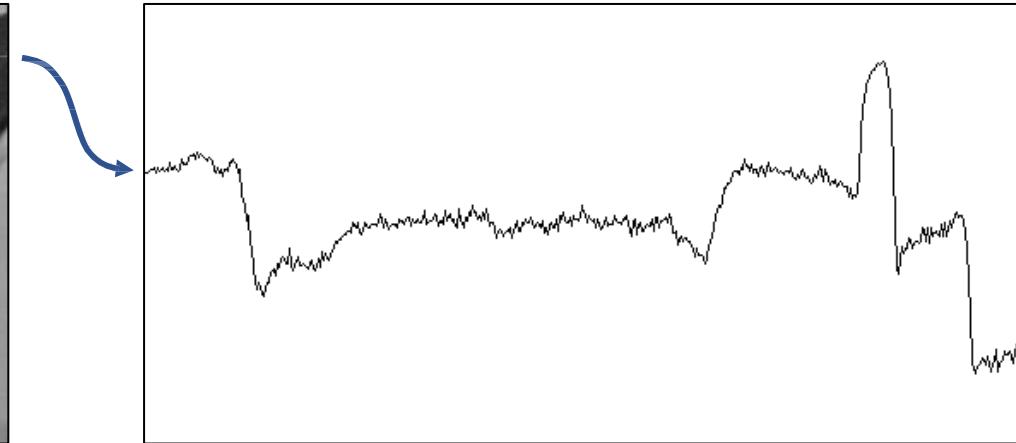


## **4. 필터링**

**5) 잡음 제거(2): 양방향 필터**

# 가우시안 필터

- 가우시안 잡음 제거에는 가우시안 필터가 효과적



# 잡음 제거(2): 양방향 필터

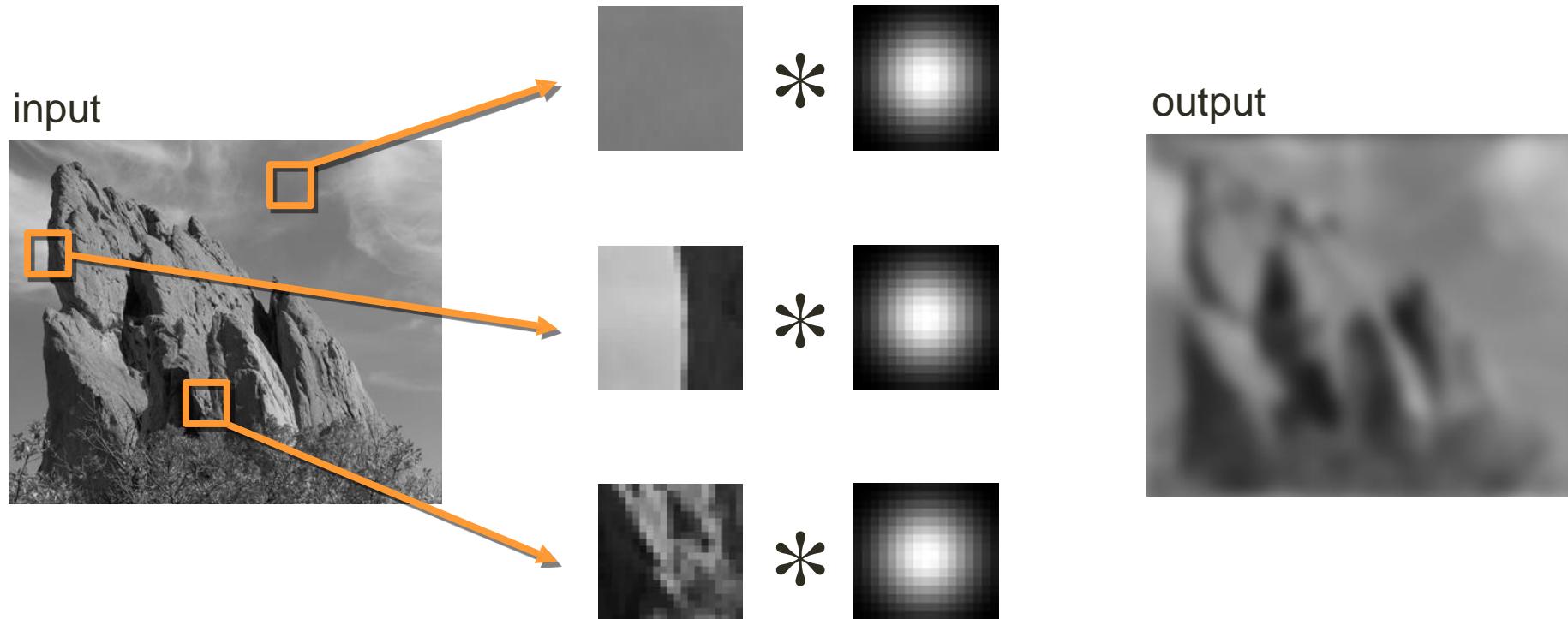
## ■ 양방향 필터(Bilateral filter)

- 에지 보전 잡음 제거 필터(edge-preserving noise removal filter)의 하나
- 평균 값 필터 또는 가우시안 필터는 에지 부근에서도 픽셀 값을 평탄하게 만드는 단점이 있음
- 기준 픽셀과 이웃 픽셀과의 거리, 그리고 픽셀 값의 차이를 함께 고려하여 블러링 정도를 조절

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

# 잡음 제거(2): 양방향 필터

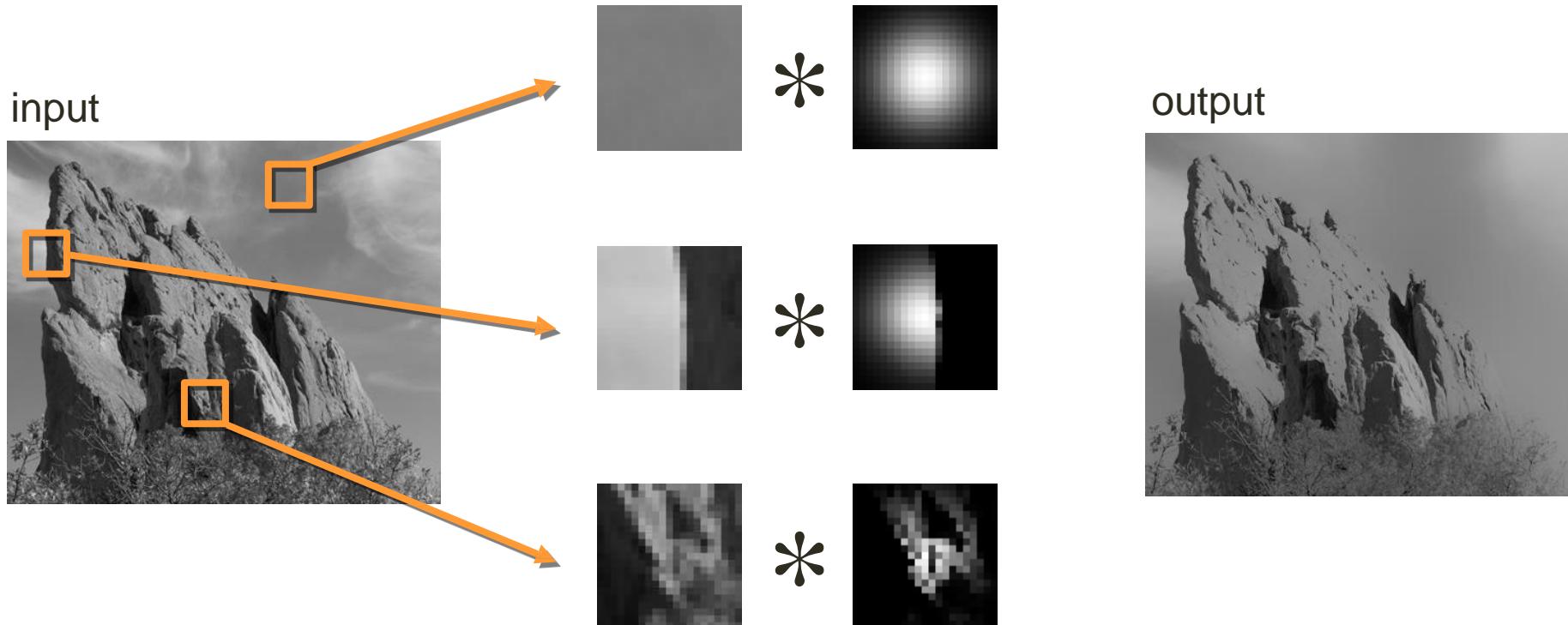
- (일반적인) 가우시안 필터링: 영상 전체에서 blurring



Same Gaussian kernel everywhere.

# 잡음 제거(2): 양방향 필터

- 양방향 필터: 에지가 아닌 부분에서만 blurring



The kernel shape depends on the image content.

# 잡음 제거(2): 양방향 필터

## ■ 양방향 필터링 함수

```
cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace, dst=None,  
                     borderType=None) -> dst
```

- src: 입력 영상. 8비트 또는 실수형, 1채널 또는 3채널.
- d: 필터링에 사용될 이웃 픽셀의 거리(지름).  
음수(-1)를 입력하면 sigmaSpace 값에 의해 자동 결정됨.
- sigmaColor: 색 공간에서 필터의 표준 편차
- sigmaSpace: 좌표 공간에서 필터의 표준 편차
- dst: 출력 영상. src와 같은 크기, 같은 타입.
- borderType: 가장자리 픽셀 처리 방식

# 잡음 제거(2): 양방향 필터

실습: bilateral.py

## ■ 양방향 필터링 예제

```
src = cv2.imread('lenna.bmp')  
  
dst = cv2.bilateralFilter(src, -1, 10, 5)
```

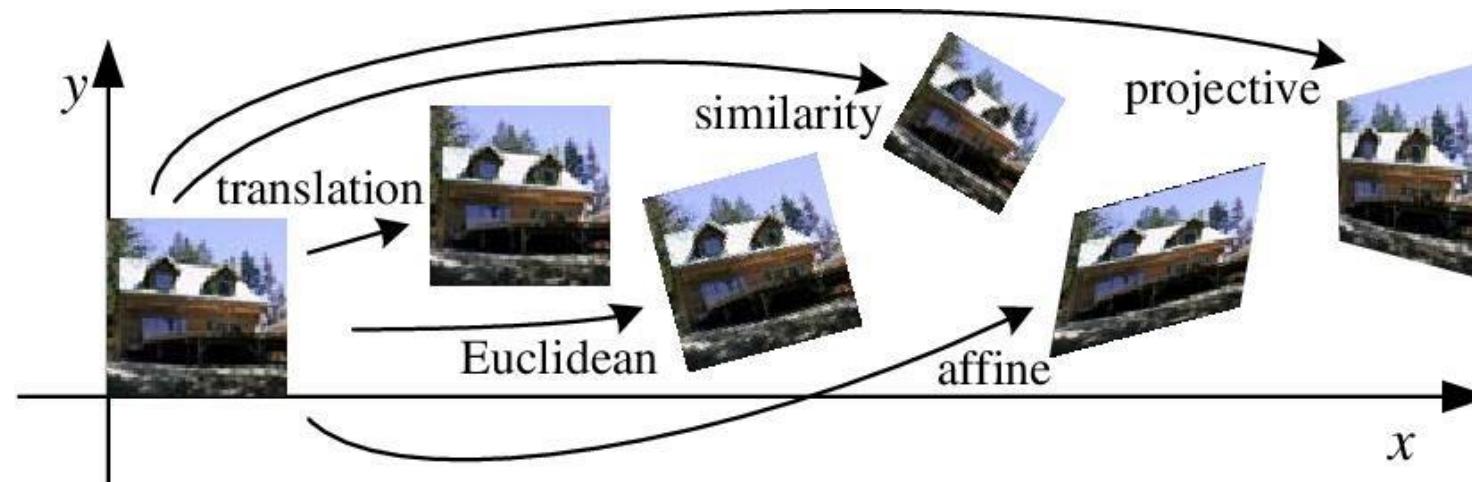


# 5. 기하학적 변환

1) 영상의 이동 변환과 전단 변환

# 영상의 기하학적 변환

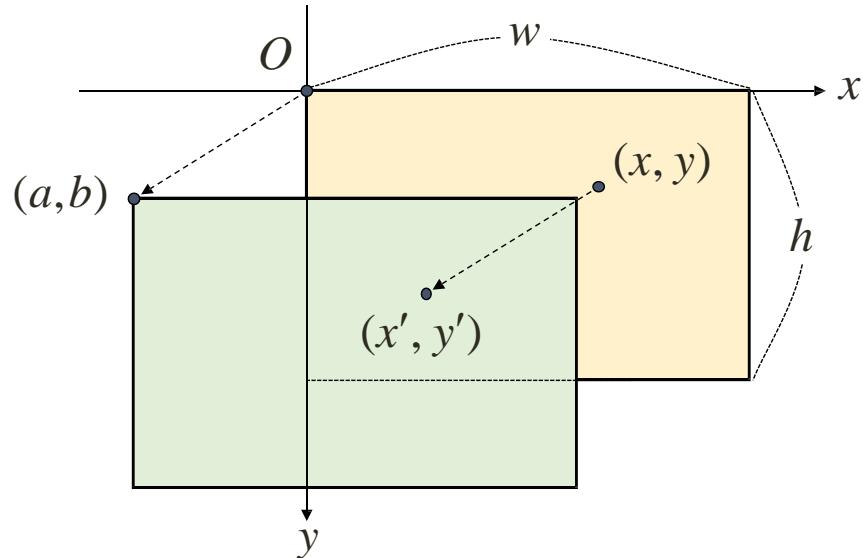
- 영상의 기하학적 변환(geometric transformation)이란?
  - 영상을 구성하는 픽셀의 배치 구조를 변경함으로써 전체 영상의 모양을 바꾸는 작업
  - Image registration, removal of geometric distortion, etc.



# 영상의 이동 변환

## ■ 이동 변환(Translation transformation)

- 가로 또는 세로 방향으로 영상을 특정 크기만큼 이동시키는 변환
- x축과 y축 방향으로의 이동 범위를 지정



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

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

↑  
**2x3 어파인 변환행렬**

# 영상의 이동 변환

## ■ 영상의 어파인 변환 함수

```
cv2.warpAffine(src, M, dsize, dst=None, flags=None,  
                borderMode=None, borderValue=None) -> dst
```

- src: 입력 영상
- M: 2x3 어파인 변환 행렬. 실수형.
- dsize: 결과 영상 크기. (w, h) 튜플. (0, 0)이면 src와 같은 크기로 설정.
- dst: 출력 영상
- flags: 보간법. 기본값은 cv2.INTER\_LINEAR.
- borderMode: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_CONSTANT.
- borderValue: cv2.BORDER\_CONSTANT일 때 사용할 상수 값. 기본값은 0.

# 영상의 이동 변환

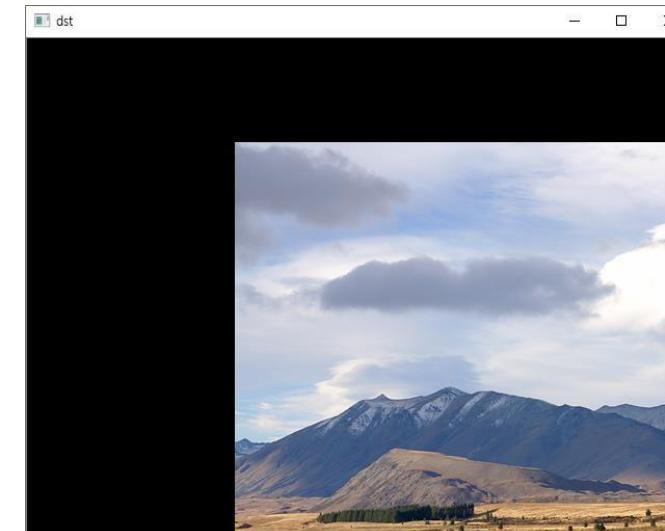
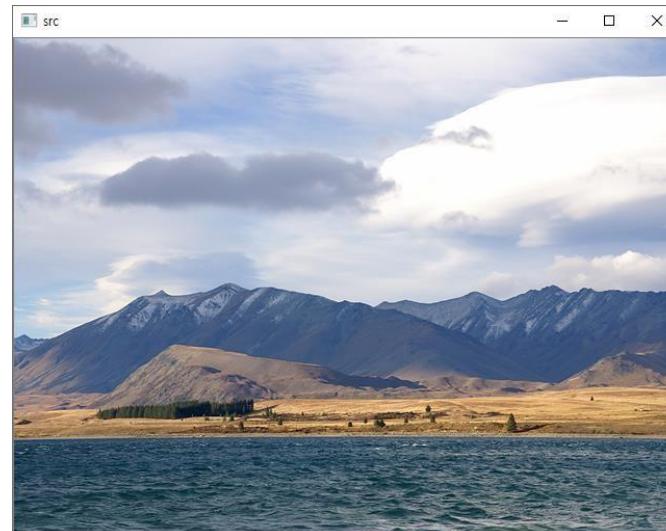
실습: translate.py

## ■ 영상의 이동 변환 예제

```
src = cv2.imread('tekapo.bmp')

aff = np.array([[1, 0, 200],
               [0, 1, 100]], dtype=np.float32)

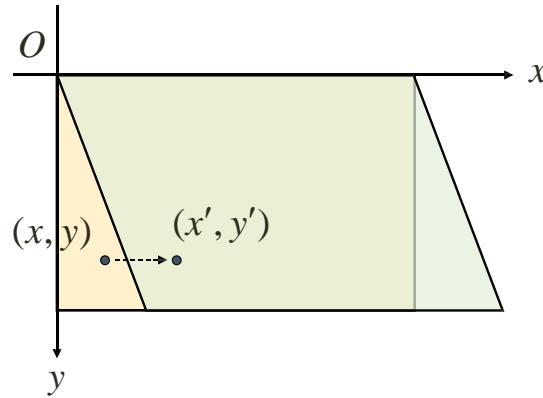
dst = cv2.warpAffine(src, aff, (0, 0))
```



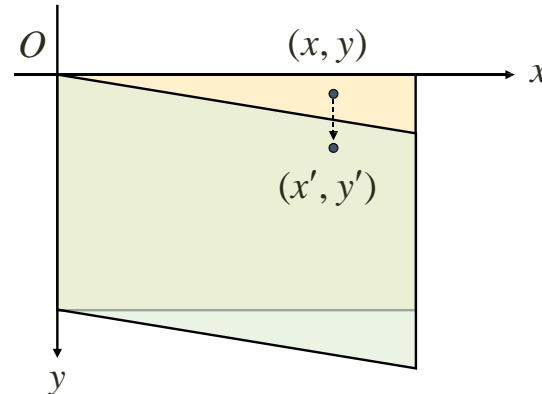
# 영상의 전단 변환

## ■ 전단 변환 (Shear transformation)

- 층 밀림 변환. x축과 y축 방향에 대해 따로 정의.



$$\begin{cases} x' = x + my \\ y' = y \end{cases} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & m & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$\begin{cases} x' = x \\ y' = mx + y \end{cases} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# 영상의 전단 변환

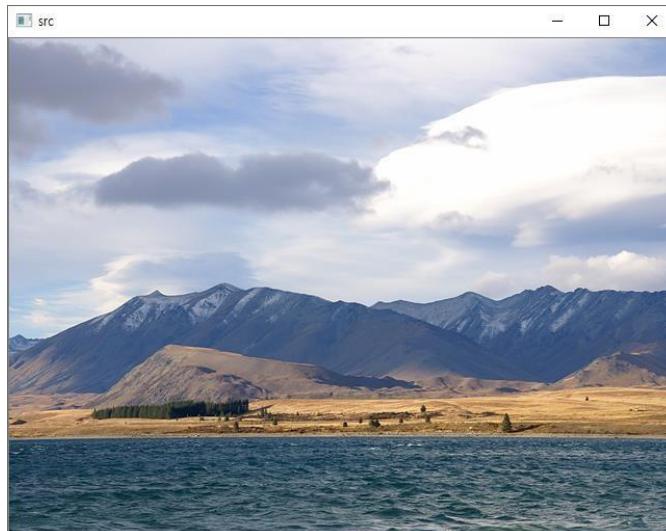
실습: shear.py

## ■ 영상의 전단 변환 예제

```
src = cv2.imread('tekapo.bmp')

aff = np.array([[1, 0.5, 0], [0, 1, 0]], dtype=np.float32)

h, w = src.shape[:2]
dst = cv2.warpAffine(src, aff, (w + int(h * 0.5), h))
```



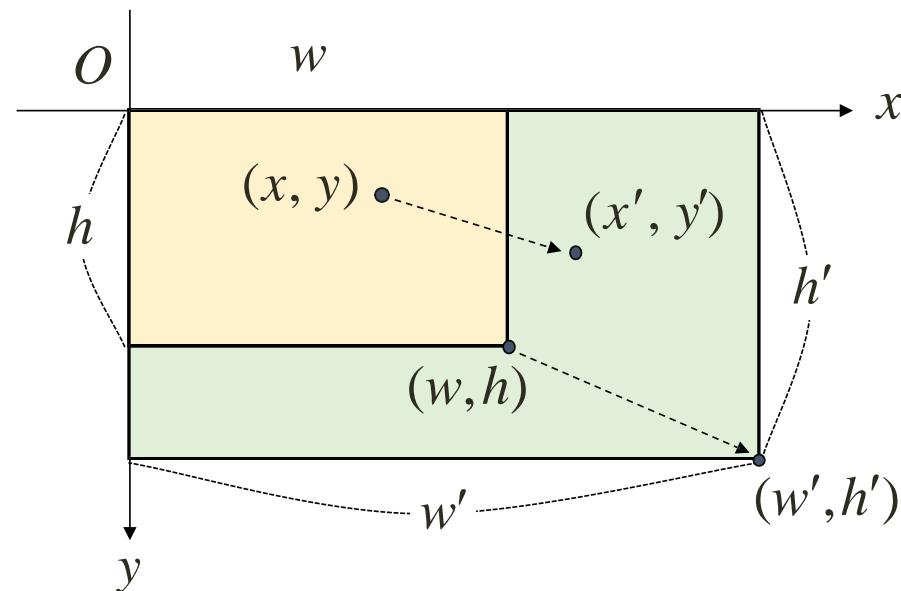
## 5. 기하학적 변환

2) 영상의 확대와 축소

# 영상의 확대와 축소

## ■ 크기 변환(Scale transformation)

- 영상의 크기를 원본 영상보다 크게 또는 작게 만드는 변환
- x축과 y축 방향으로의 스케일 비율(scale factor)을 지정



$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases} \quad \begin{cases} s_x = w' / w \\ s_y = h' / h \end{cases}$$

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

# 영상의 확대와 축소

## ■ 영상의 크기 변환

```
cv2.resize(src, dsize, dst=None, fx=None, fy=None, interpolation=None) -> dst
```

- src: 입력 영상
- dsize: 결과 영상 크기. (w, h) 튜플. (0, 0)이면 fx와 fy 값을 이용하여 결정.
- dst: 출력 영상
- fx, fy: x와 y방향 스케일 비율(scale factor). (dsize 값이 0일 때 유효)
- interpolation: 보간법 지정. 기본값은 cv2.INTER\_LINEAR

cv2.INTER_NEAREST	최근방 이웃보간법
cv2.INTER_LINEAR	양선형 보간법 (2x2 이웃 픽셀 참조)
cv2.INTER_CUBIC	3차회선 보간법 (4x4 이웃 픽셀 참조)
cv2.INTER_LANCZOS4	Lanczos 보간법 (8x8 이웃 픽셀 참조)
cv2.INTER_AREA	영상 축소 시 효과적

# 영상의 확대와 축소

실습: scaling.py

## ■ 영상의 크기 변환 예제

```
src = cv2.imread('rose.bmp') # 480x320

dst1 = cv2.resize(src, (0, 0), fx=4, fy=4, interpolation=cv2.INTER_NEAREST)
dst2 = cv2.resize(src, (1920, 1280)) # cv2.INTER_LINEAR
dst3 = cv2.resize(src, (1920, 1280), interpolation=cv2.INTER_CUBIC)
dst4 = cv2.resize(src, (1920, 1280), interpolation=cv2.INTER_LANCZOS4)

cv2.imshow('src', src)
cv2.imshow('dst1', dst1[500:900, 400:800])
cv2.imshow('dst2', dst2[500:900, 400:800])
cv2.imshow('dst3', dst3[500:900, 400:800])
cv2.imshow('dst4', dst4[500:900, 400:800])
cv2.waitKey()

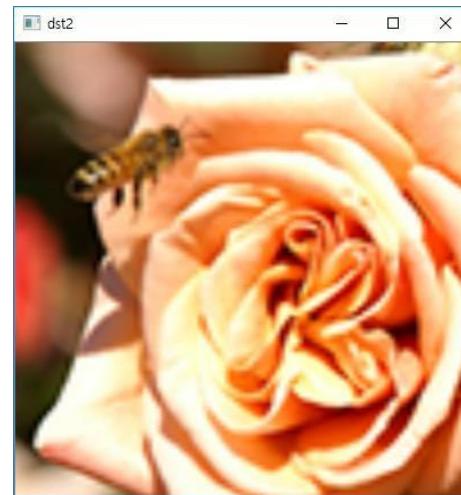
cv2.destroyAllWindows()
```

# 영상의 확대와 축소

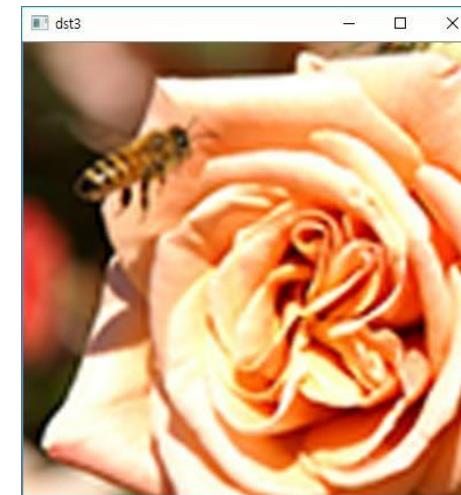
## ■ 영상의 크기 변환 예제 결과



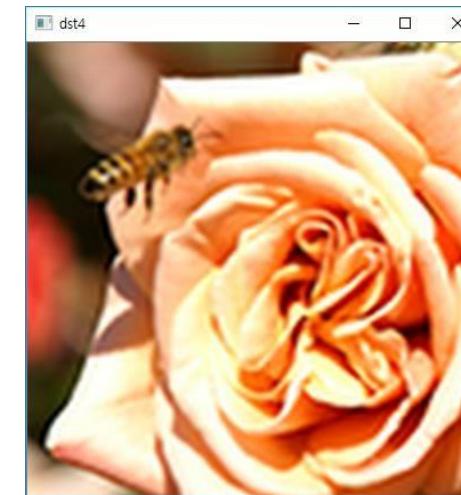
cv2.INTER\_NEAREST  
(by one neighbors)



cv2.INTER\_LINEAR  
(by 2x2 neighbors)



cv2.INTER\_CUBIC  
(by 4x4 neighbors)

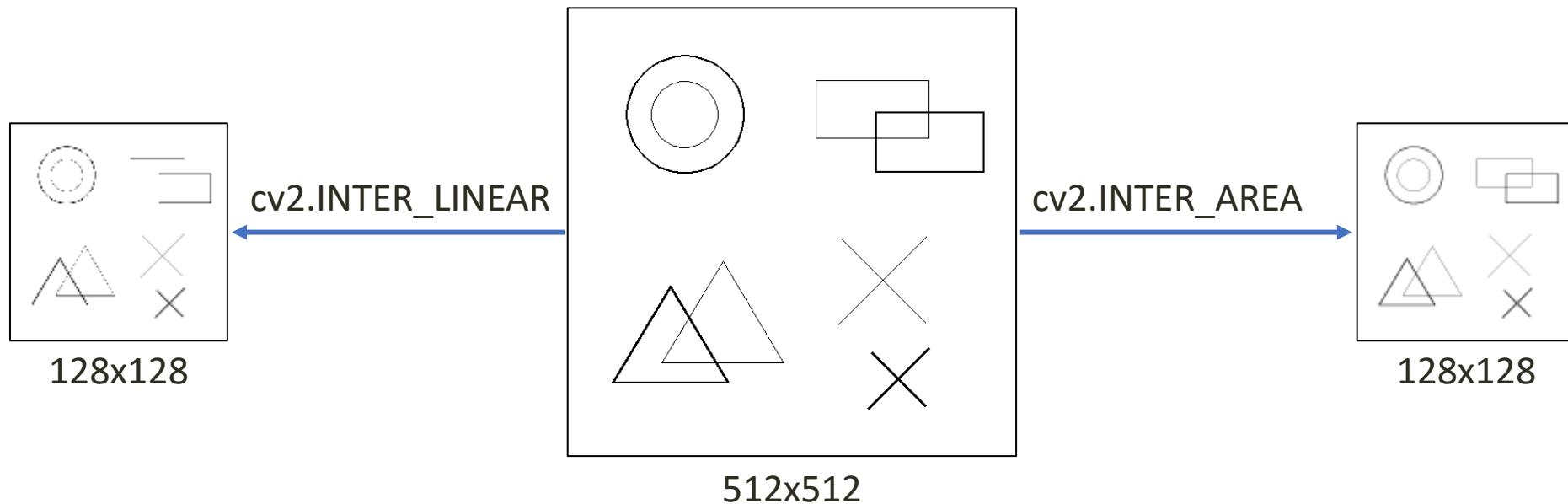


cv2.INTER\_LANCZOS4  
(by 8x8 neighbors)

# 영상의 확대와 축소

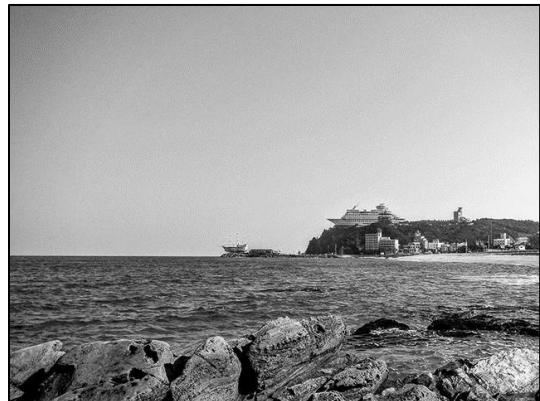
## ■ 영상의 축소 시 고려할 사항

- 영상 축소 시 디테일이 사라지는 경우가 발생 (e.g. 한 픽셀로 구성된 선분)
- 입력 영상을 부드럽게 필터링한 후 축소, 다단계 축소
- OpenCV의 cv2.resize() 함수에서는 cv2.INTER\_AREA 플래그를 사용



# 영상의 대칭

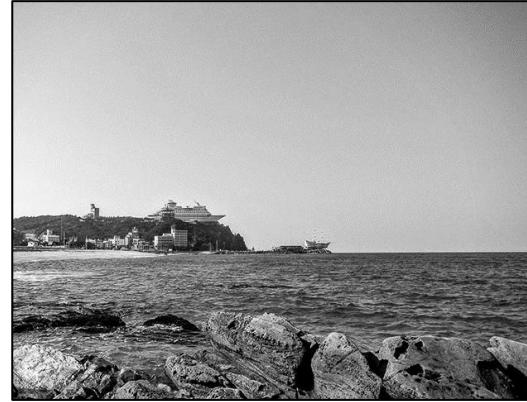
- 영상의 대칭 변화 (flip, reflection)



좌우 대칭

상하 대칭

좌우&상하 대칭



# 영상의 대칭

## ■ 영상의 대칭 변환

```
cv2.flip(src, flipCode, dst=None) -> dst
```

- src: 입력 영상 대
- flipCode: 칭 방향지정

양수 (+1)	좌우 대칭
0	상하 대칭
음수 (-1)	좌우 & 상하 대칭

- dst: 출력 영상

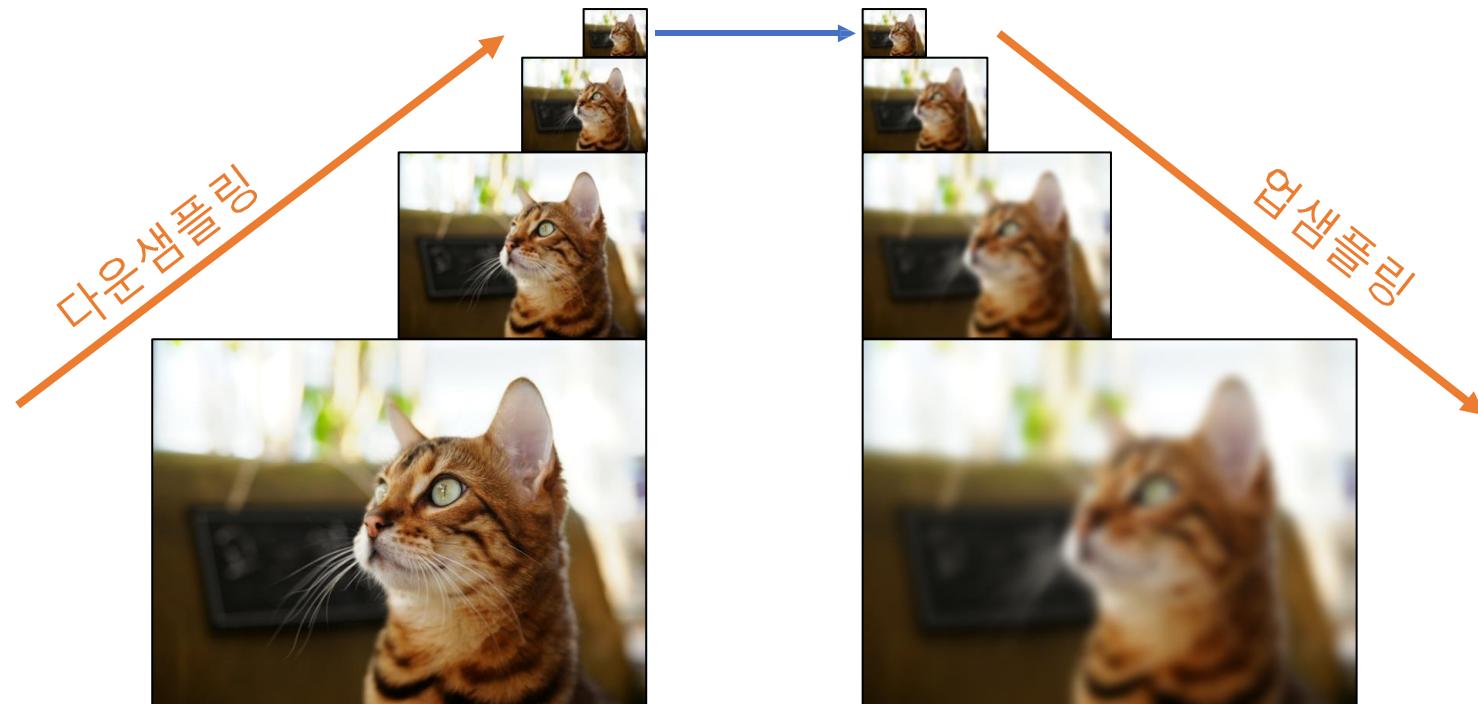
## 5. 기하학적 변환

3) 이미지 피라미드

# 이미지 피라미드

## ■ 이미지 피라미드(Image pyramid)란?

- 하나의 영상에 대해 다양한 해상도의 영상 세트를 구성한 것
- 보통 가우시안 블러링 & 다운샘플링 형태로 축소하여 구성



# 이미지 피라미드

## ■ 영상 피라미드 다운샘플링

```
cv2.pyrDown(src, dst=None, dstsize=None, borderType=None) -> dst
```

- src: 입력 영상
- dst: 출력 영상
- dstsize: 출력 영상 크기.  
따로 지정하지 않으면 입력 영상의 가로, 세로 크기의 1/2로 설정.
- borderType 가장자리 픽셀 확장 방식
- 참고 사항
  - 먼저 5x5 크기의 가우시안 필터를 적용
  - 이후 짹수 행과 열을 제거하여 작은 크기의 영상을 생성

# 이미지 피라미드

## ■ 영상 피라미드 업샘플링

```
cv2.pyrUp(src, dst=None, dstsize=None, borderType=None) -> dst
```

- src: 입력 영상
- dst: 출력 영상
- dstsize: 출력 영상 크기.  
따로 지정하지 않으면 입력 영상의 가로, 세로 크기의 2배 설정.
- borderType 가장자리 픽셀 확장 방식

# 이미지 피라미드

실습: pyramid.py

## ■ 피라미드 영상에 사각형 그리기 예제

```
src = cv2.imread('cat.bmp')

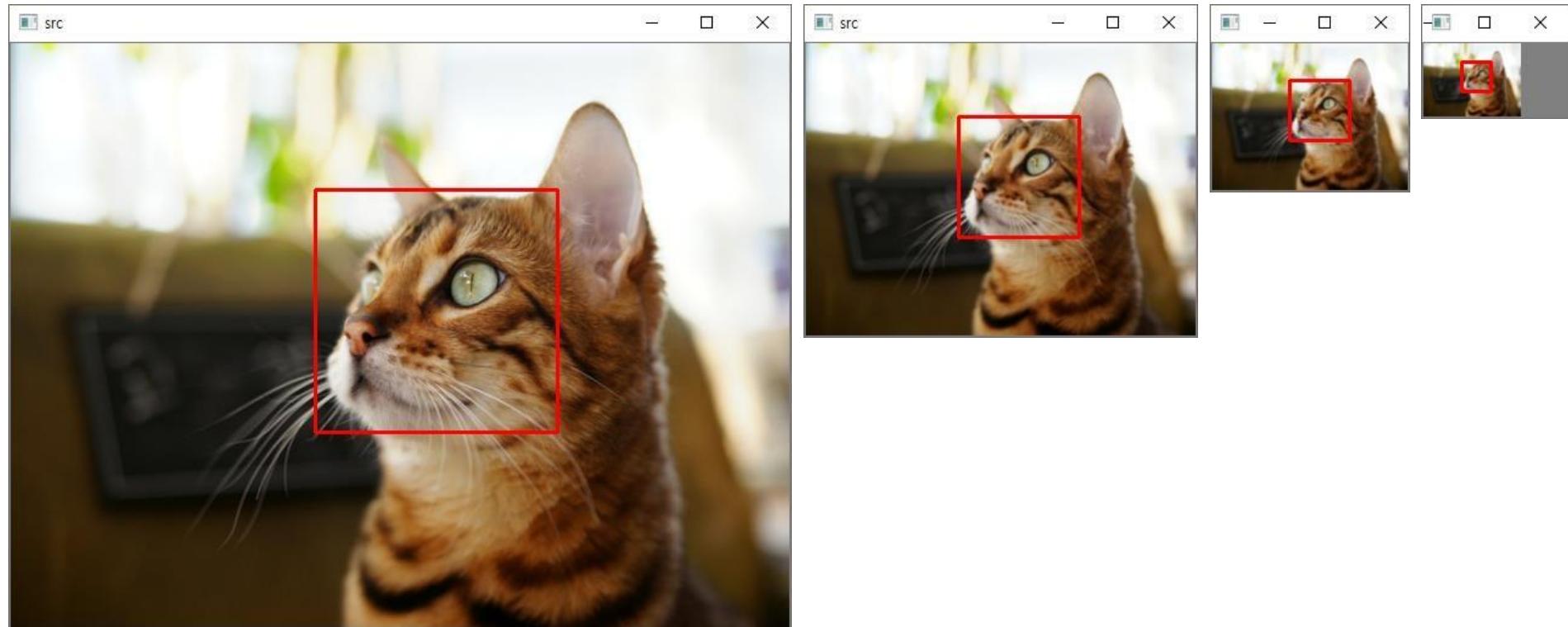
rc = (250, 120, 200, 200) # rectangle tuple

cpy = src.copy()
cv2.rectangle(cpy, rc, (0, 0, 255), 2)
cv2.imshow('src', cpy)
cv2.waitKey()

for i in range(1, 4):
    src = cv2.pyrDown(src)
    cpy = src.copy()
    cv2.rectangle(cpy, rc, (0, 0, 255), 2, shift=i)
    cv2.imshow('src', cpy)
    cv2.waitKey()
```

# 이미지 피라미드

- 피라미드 영상에 사각형 그리기 예제 실행 결과



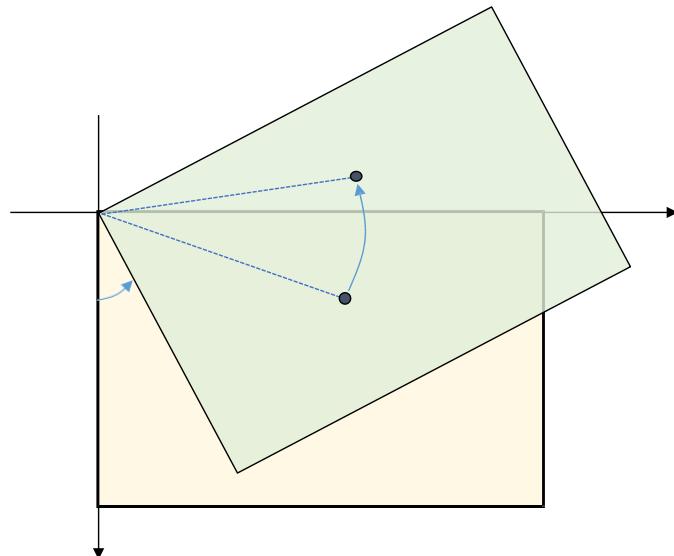
## 5. 기하학적 변환

4) 영상의 회전

# 영상의 회전

## ■ 회전 변환(rotation transformation)

- 영상을 특정 각도만큼 회전시키는 변환 (반시계 방향)



$$\begin{cases} x' = \cos\theta \cdot x + \sin \\ y' = - \end{cases}$$

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

# 영상의 회전

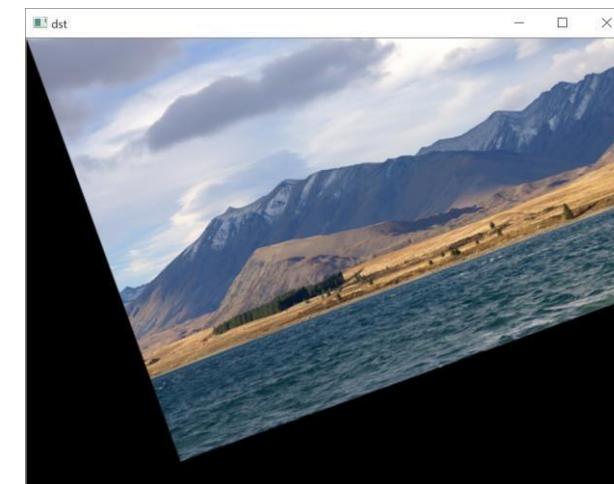
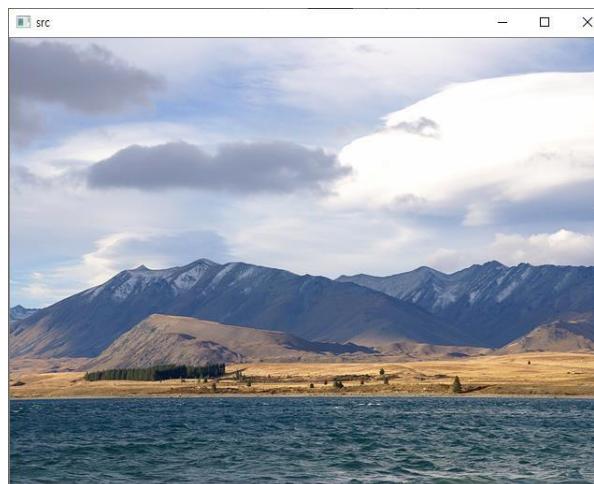
실습: rotation1.py

## ■ 영상의 회전 예제

```
src = cv2.imread('tekapo.bmp')

rad = 20 * math.pi / 180
aff = np.array([[math.cos(rad), math.sin(rad), 0],
               [-math.sin(rad), math.cos(rad), 0]], dtype=np.float32)

dst = cv2.warpAffine(src, aff, (0, 0))
```



# 영상의 회전

## ■ 영상의 회전 변환 행렬 구하기

```
cv2.getRotationMatrix2D(center, angle, scale) -> retval
```

- center: 회전 중심 좌표. (x, y) 튜플.
- angle: (반시계 방향) 회전 각도(degree). 음수는 시계 방향.
- scale: 추가적인 확대 비율
- retval: 2x3 어파인 변환행렬. 실수형.

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot \text{center}.x - \beta \cdot c \\ -\beta & \alpha & \beta \end{bmatrix}$$

where  $\begin{cases} \alpha = \text{scale} \cdot \cos(\text{angle}) \\ \beta = \text{scale} \cdot \sin(\text{angle}) \end{cases}$

# 영상의 회전

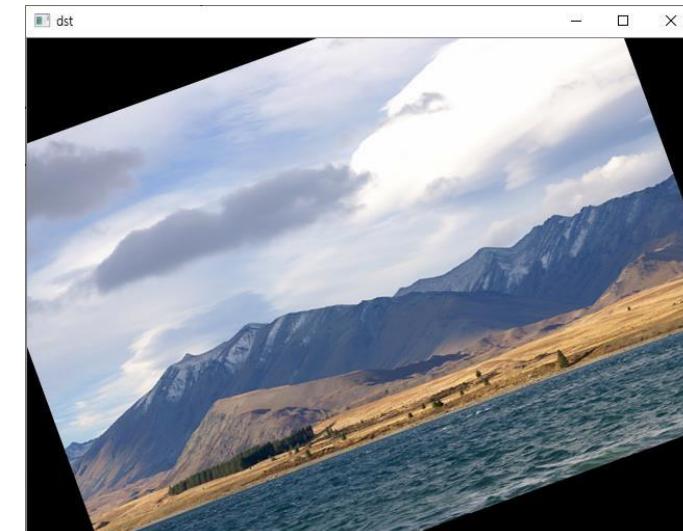
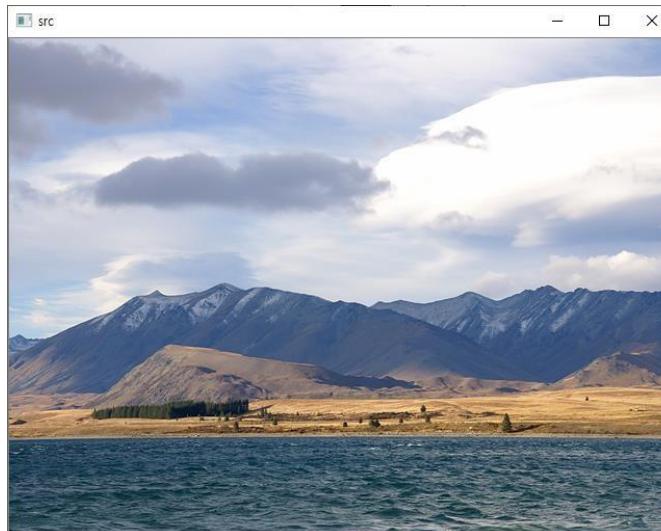
실습: rotation2.py

## ■ 영상의 중앙 기준 회전 예제

```
src = cv2.imread('tekapo.bmp')

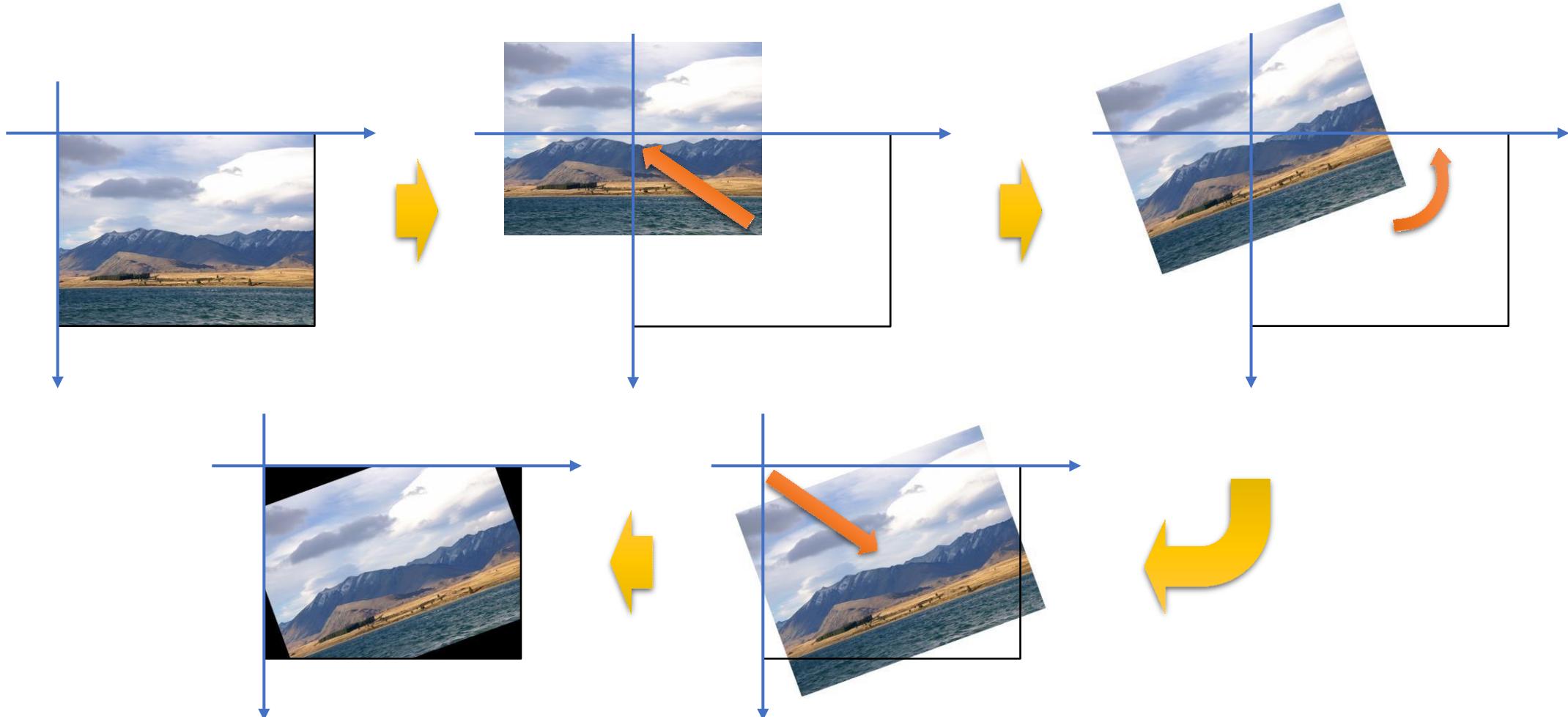
cp = (src.shape[1] / 2, src.shape[0] / 2)
rot = cv2.getRotationMatrix2D(cp, 20, 1)

dst = cv2.warpAffine(src, rot, (0, 0))
```



# 영상의 회전

## ■ 영상의 중앙 기준 회전 예제 동작



# 5. 기하학적 변환

5) 리매핑

# 리매핑

## ■ 리매핑(remapping)

- 영상의 특정 위치 픽셀을 다른 위치에 재배치하는 일반적인 프로세스

$$dst(x, y) = \text{src}(\text{map}_x(x, y), \text{map}_y(x, y))$$

- 어파인 변환, 투시 변환을 포함한 다양한 변환을 리매핑으로 표현 가능
- examples)

$$\begin{cases} \text{map}_x(x, y) = x - 200 \\ \text{map}_y(x, y) = y - 100 \end{cases}$$

$$\begin{cases} \text{map}_x(x, y) = w - 1 - x \\ \text{map}_y(x, y) = y \end{cases}$$

# 리매핑

## ■ 리매핑 함수

```
cv2.remap(src, map1, map2, interpolation, dst=None, borderMode=None,  
          borderValue=None) -> dst
```

- src: 입력 영상
- map1: 결과 영상의 (x, y) 좌표가 참조할 입력 영상의 x좌표.  
입력 영상과 크기는 같고, 타입은 np.float32인 numpy.ndarray.
- map2: 결과 영상의 (x, y) 좌표가 참조할 입력 영상의 y좌표.
- interpolation: 보간법
- dst: 출력 영상
- borderMode: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_CONSTANT.
- borderValue: cv2.BORDER\_CONSTANT일 때 사용할 상수 값. 기본값은 0.

# 리매핑

실습: remap.py

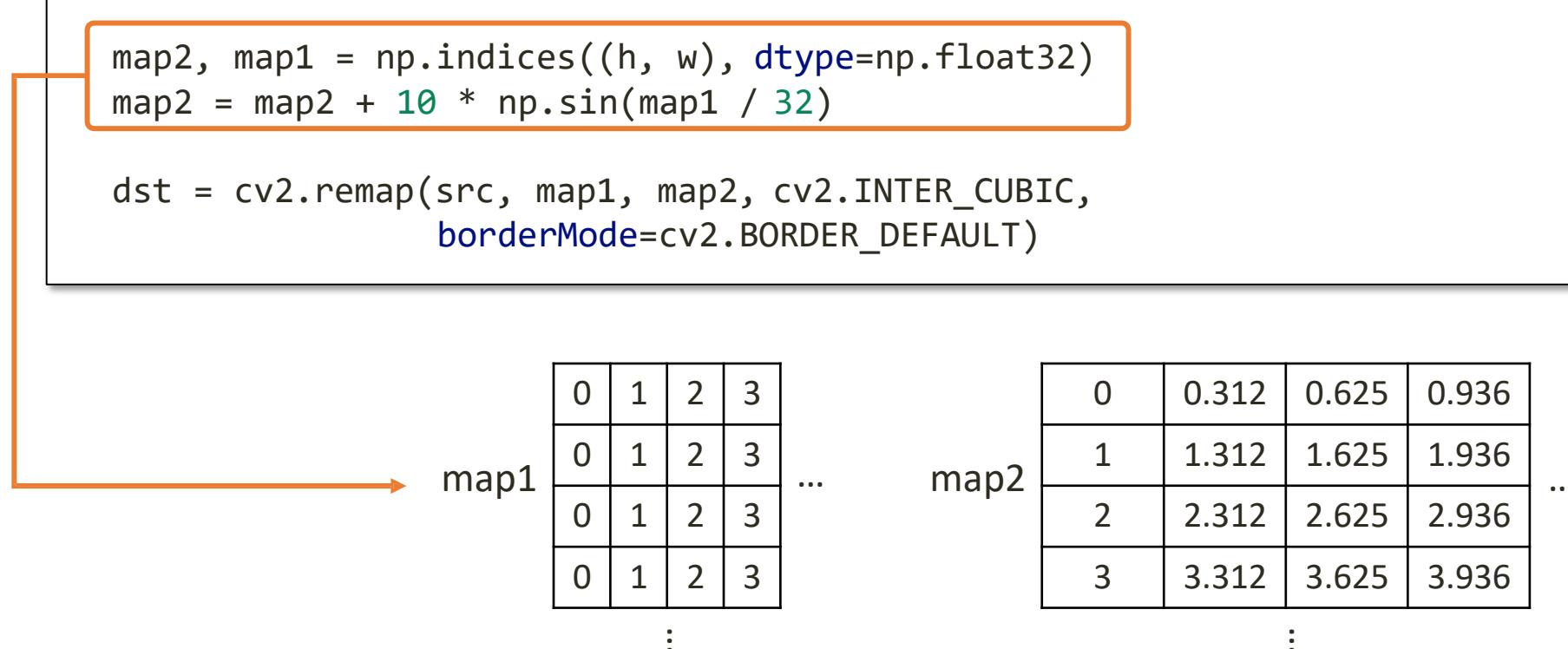
## ■ 삼각함수를 이용한 리매핑 예제

```
src = cv2.imread('tekapo.bmp')

h, w = src.shape[:2]

map2, map1 = np.indices((h, w), dtype=np.float32)
map2 = map2 + 10 * np.sin(map1 / 32)

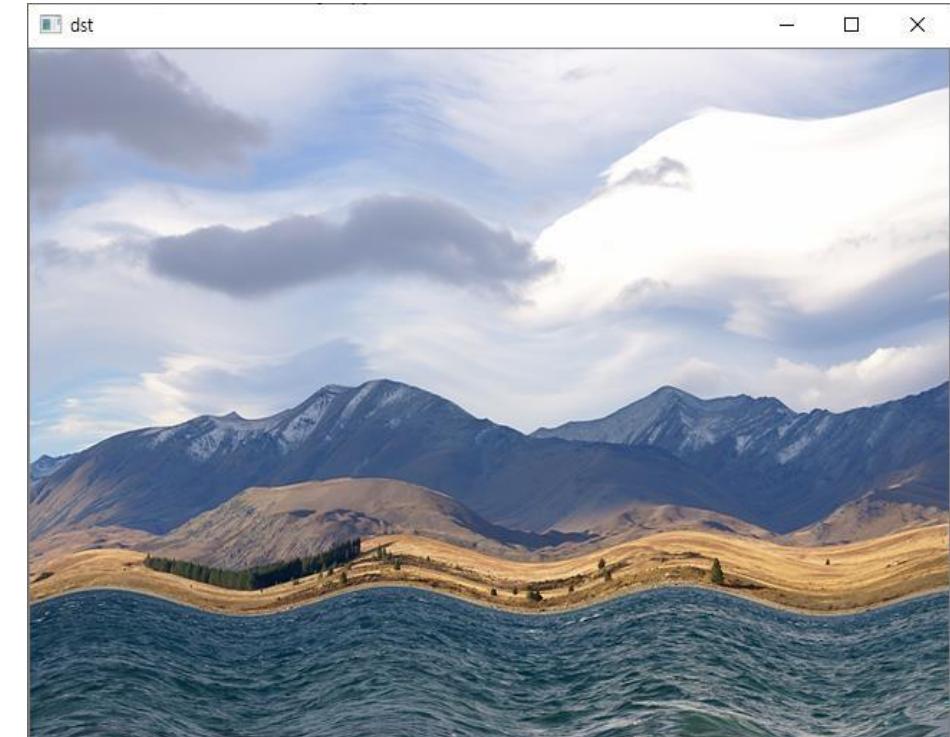
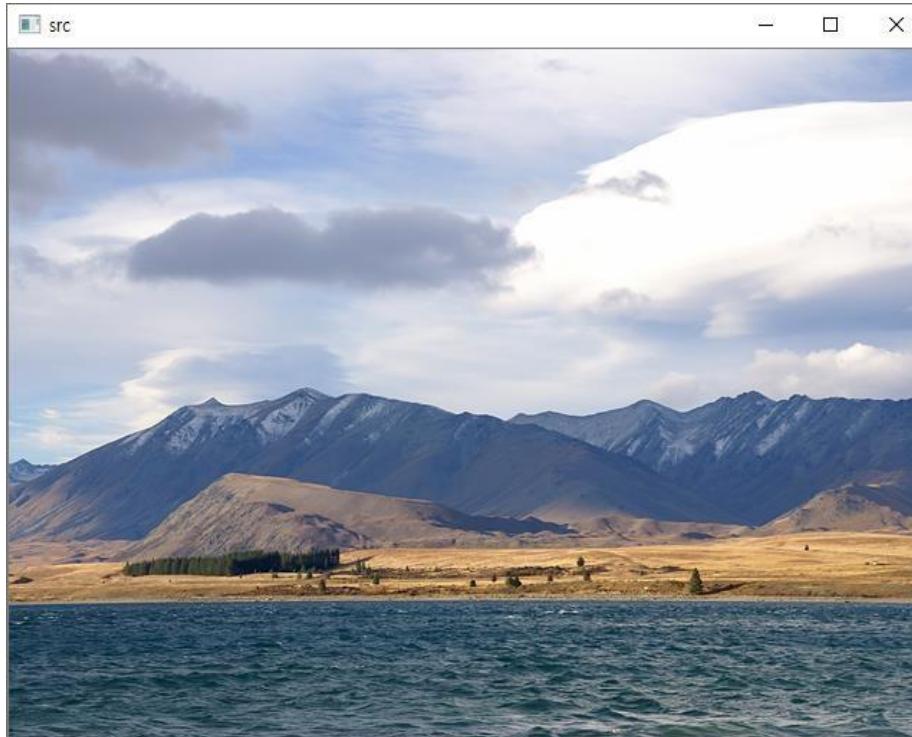
dst = cv2.remap(src, map1, map2, cv2.INTER_CUBIC,
                 borderMode=cv2.BORDER_DEFAULT)
```



# 리매핑

실습: perspective.py

- 삼각함수를 이용한 리매핑 예제



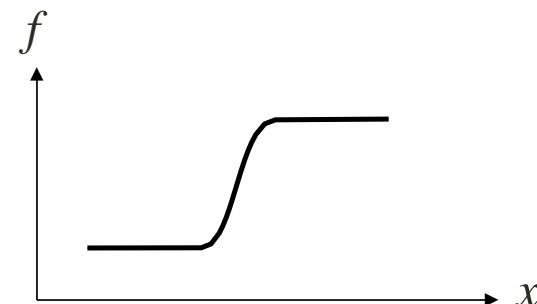
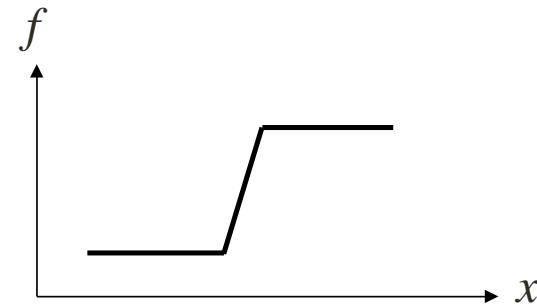
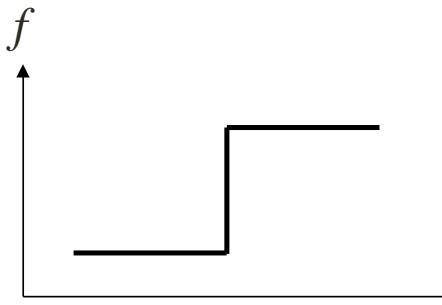
## 6. 영상의 특징 추출

1) 영상의 미분과 소벨 필터

# 에지 검출과 미분

## ■ 에지(edge)

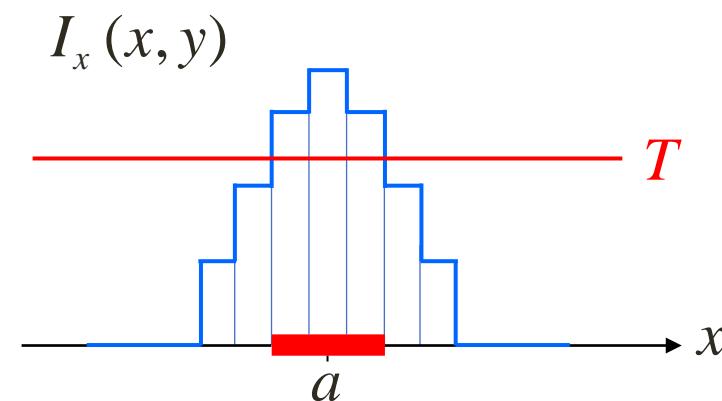
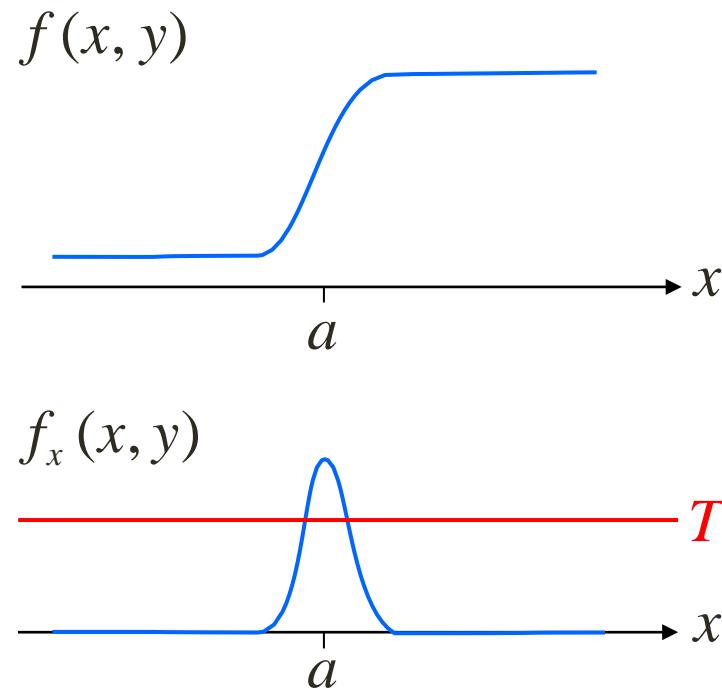
- 영상에서 픽셀의 밝기 값이 급격하게 변하는 부분
- 일반적으로 배경과 객체, 또는 객체와 객체의 경계



# 에지 검출과 미분

## ■ 기본적인 에지 검출 방법

- 영상  $(x, y)$  변수의 함수로 간주했을 때, 이 함수의 1차 미분(1<sup>st</sup> derivative) 값이 크게 나타나는 부분을 검출



# 영상의 미분과 소벨 필터

## ■ 1차 미분의 근사화(approximation)

- 전진 차분  
(Forward difference):

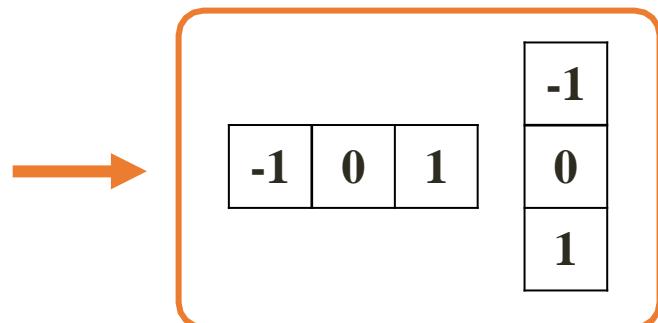
$$\frac{\partial I}{\partial x} \cong \frac{I(x+h) - I(x)}{h}$$

- 후진 차분  
(Backward difference):

$$\frac{\partial I}{\partial x} \cong \frac{I(x) - I(x-h)}{h}$$

- 중앙 차분  
(Centered difference):

$$\frac{\partial I}{\partial x} \cong \frac{I(x+h) - I(x-h)}{2h}$$



# 영상의 미분과 소벨 필터

## ■ 다양한 미분마스크

가로 방향:

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

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

-3	0	3
-10	0	10
-3	0	3

세로 방향:

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

Prewitt

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

Sobel

-3	-10	-3
0	0	0
3	10	3

Scharr

# 영상의 미분과 소벨 필터

## ■ 소벨 필터를 이용한 미분 함수

```
cv2.Sobel(src, ddepth, dx, dy, dst=None, ksize=None, scale=None,  
          delta=None, borderType=None) -> dst
```

- src: 입력 영상
- ddepth: 출력 영상 데이터 타입. -1이면 입력 영상과 같은 데이터 타입을 사용.
- dx: x 방향 미분 차수.
- dy: y 방향 미분 차수.
- dst: 출력 영상(행렬)
- ksize: 커널 크기. 기본값은 3.
- scale: 연산 결과에 추가적으로 곱할 값. 기본값은 1.
- delta: 연산 결과에 추가적으로 더할 값. 기본값은 0.
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_DEFAULT.

대부분 dx=1, dy=0, ksize=3 또는  
dx=0, dy=1, ksize=3 으로 지정.

# 영상의 미분과 소벨 필터

## ■ 샤르 필터를 이용한 미분 함수

```
cv2.Scharr(src, ddepth, dx, dy, dst=None, scale=None,  
           borderType=None) -> dst
```

- src: 입력 영상
- ddepth: 출력 영상 데이터 타입. -1이면 입력 영상과 같은 데이터 타입을 사용.
- dx: x 방향 미분 차수.
- dy: y 방향 미분 차수.
- dst: 출력 영상(행렬)
- scale: 연산 결과에 추가적으로 곱할 값. 기본값은 1.
- delta: 연산 결과에 추가적으로 더할 값. 기본값은 0.
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_DEFAULT.

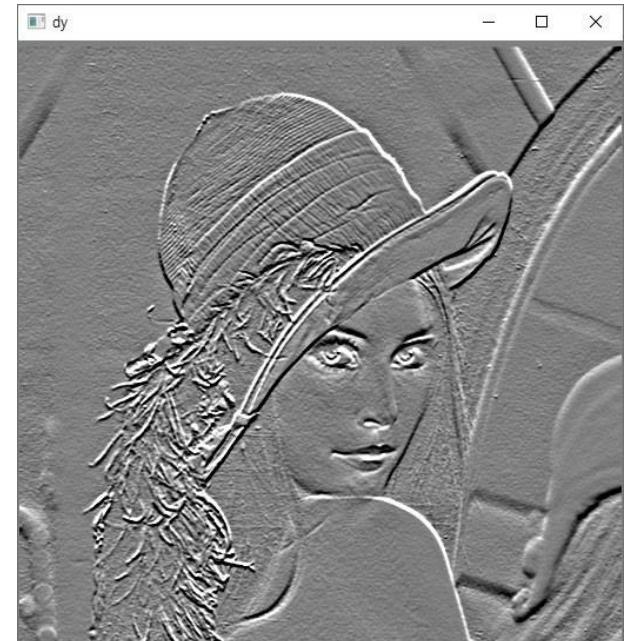
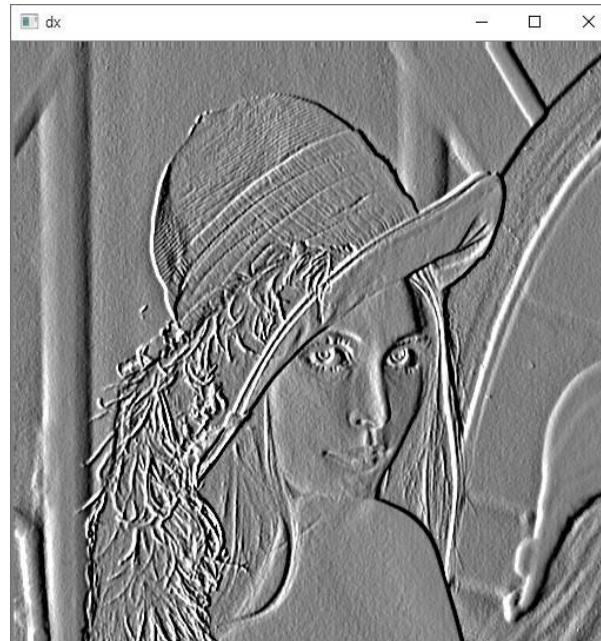
# 영상의 미분과 소벨 필터

실습: sobel.py

- 소벨 필터를 이용한 영상의 미분 예제

```
src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

dx = cv2.Sobel(src, -1, 1, 0, delta=128)
dy = cv2.Sobel(src, -1, 0, 1, delta=128)
```



## 6. 영상의 특징 추출

2) 그래디언트와 에지 검출

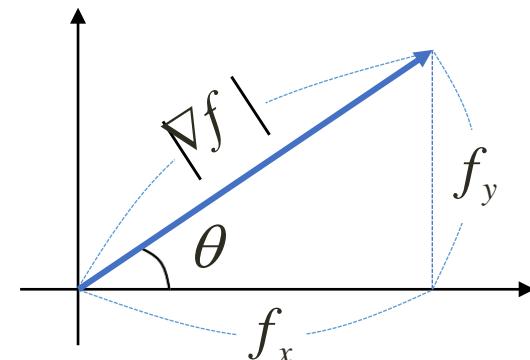
# 그래디언트

## ■ 영상의 그래디언트(gradient)

- 함수  $f(x, y)$ 를  $x$ 축과  $y$  축으로 각각 편미분(partial derivative)하여 벡터 형태로 표현한 것

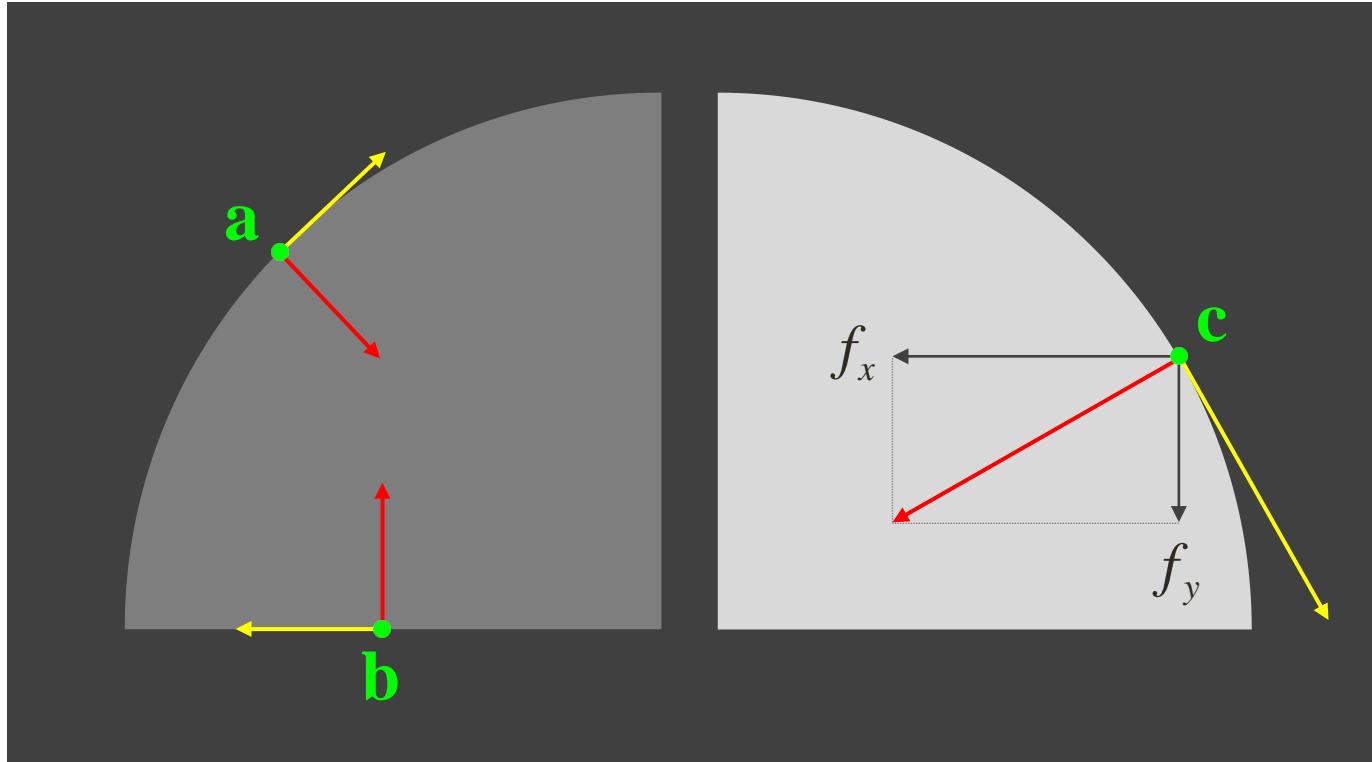
$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = f_x \mathbf{i} + f_y \mathbf{j}$$

- 그래디언트 크기:  $|\nabla f| = \sqrt{f_x^2 + f_y^2}$
- 그래디언트 방향:  $\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$



# 그래디언트

- 실제 영상에서 구한 그래디언트 크기와 방향
  - 그래디언트 크기: 픽셀 값의 차이 정도, 변화량
  - 그래디언트 방향: 픽셀 값이 가장 급격하게 증가하는 방향



# 그래디언트

## ■ 2D 벡터의 크기 계산 함수

```
cv2.magnitude(x, y, magnitude=None) -> magnitude
```

- x: 2D 벡터의 x 좌표 행렬. 실수형.
- y: 2D 벡터의 y 좌표 행렬. x와 같은 크기. 실수형.
- magnitude: 2D 벡터의 크기 행렬. x와 같은 크기, 같은 타입.  
$$\text{magnitude}(II) = \sqrt{x(II)^2 + y(II)^2}$$

# 그래디언트

## ■ 2D 벡터의 방향 계산 함수

```
cv2.phase(x, y, angle=None, angleInDegrees=None) -> angle
```

- x: 2D 벡터의 x 좌표 행렬. 실수형.
- y: 2D 벡터의 y 좌표 행렬. x와 같은 크기. 실수형.
- angle: 2D 벡터의 크기 행렬. x와 같은 크기, 같은 타입.  
$$\text{angle}(I) = \text{atan2}(y(I), x(I))$$

만약  $x(I)=y(I)=0$ 이면 angle은 0으로 설정됨.
- angleInDegrees: True이면 각도 단위, False이면 래디언 단위.

# 그래디언트와 에지 검출

실습: sobel\_edge.py

## ■ 소벨 필터를 이용한 에지 검출 예제

```
src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

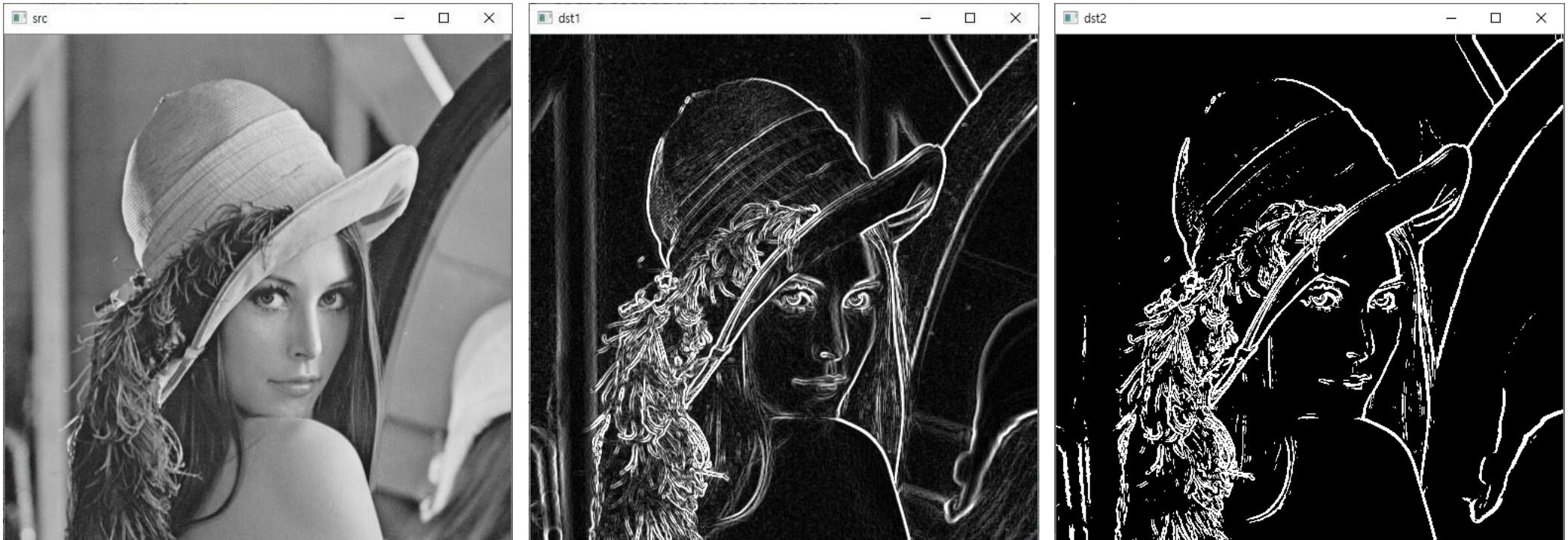
dx = cv2.Sobel(src, cv2.CV_32F, 1, 0)
dy = cv2.Sobel(src, cv2.CV_32F, 0, 1)

mag = cv2.magnitude(dx, dy)
mag = np.clip(mag, 0, 255).astype(np.uint8)

dst = np.zeros(src.shape[:2], np.uint8)
dst[mag > 120] = 255
#_, dst = cv2.threshold(mag, 120, 255, cv2.THRESH_BINARY)
```

# 그래디언트와 에지 검출

- 소벨 필터를 이용한 에지 검출 예제 실행 결과



## 6. 영상의 특징 추출

3) 캐니 에지 검출

# 캐니 에지 검출

## ■ 좋은 에지 검출기의 조건 (J. Canny)

- 정확한 검출(**Good detection**):      에지가 아닌 점을 에지로 찾거나 또는 에지인데 에지로 찾지 못하는 확률을 최소화
- 정확한 위치(**Good localization**):      실제 에지의 중심을 검출
- 단일 에지(**Single edge**):      하나의 에지는 하나의 점으로 표현

John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

# 캐니 에지 검출

## ■ 캐니 에지 검출 1단계

- 가우시안 필터링
  - (Optional) 잡음 제거 목적

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



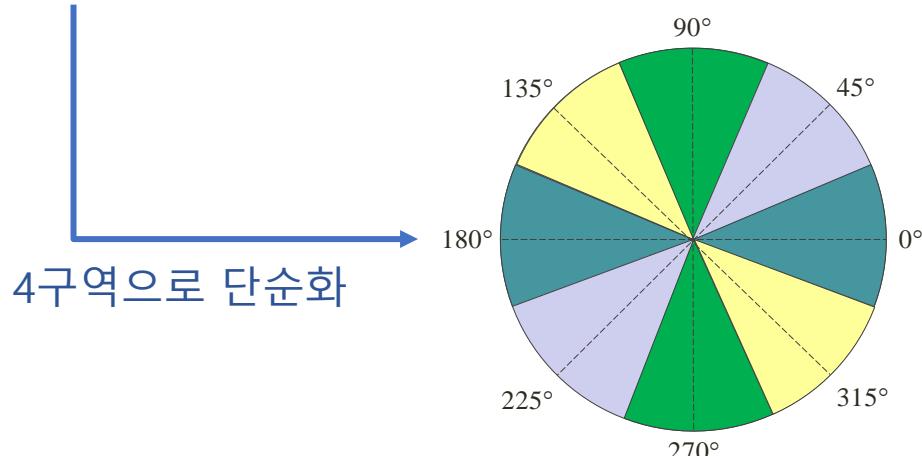
# 캐니 에지 검출

## ■ 캐니 에지 검출 2단계

- 그래디언트 계산
  - 주로 소벨 마스크를 사용

- 크기:  $\|f\| = \sqrt{f_x^2 + f_y^2}$

- 방향:  $\theta = \tan^{-1} (f_y / f_x)$



가우시안 필터링

그래디언트 계산  
(크기 & 방향)

비최대 억제

이중 임계값을 이용한  
히스테리시스 에지트래킹

# 캐니 에지 검출

## ■ 캐니 에지 검출 3단계

- 비최대 억제(Non-maximum suppression)
  - 하나의 에지가 여러 개의 픽셀로 표현되는 현상을 없애기 위하여 그래디언트 크기가 국지적 최대(local maximum)인 픽셀만을 에지 픽셀로 설정
  - 그래디언트 방향에 위치한 두 개의 픽셀을 조사하여 국지적 최대를 검사

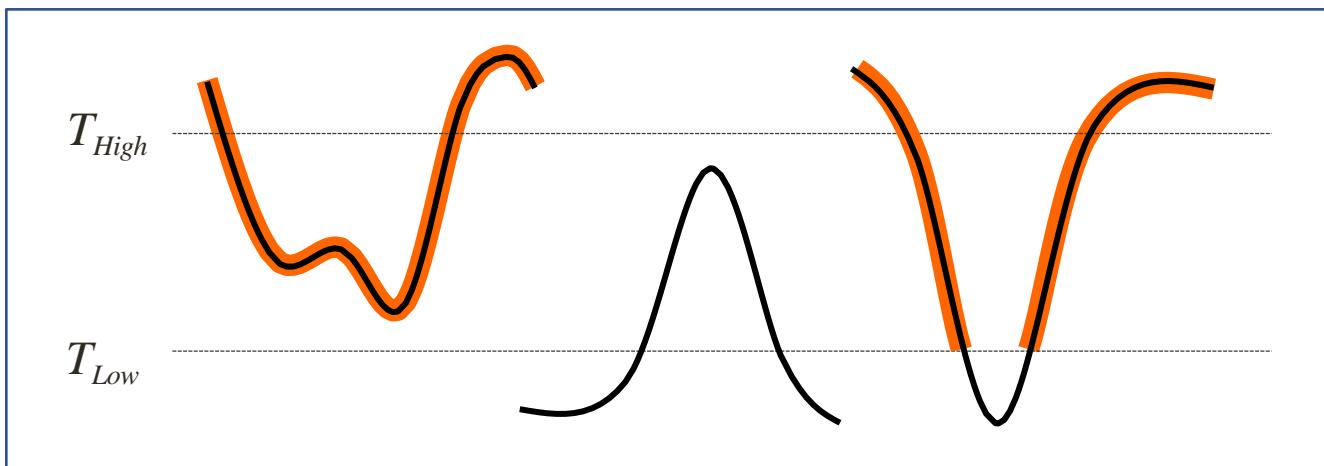


# 캐니 에지 검출

## ■ 캐니 에지 검출 4단계

- 히스테리시스 에지 트래킹 (Hysteresis edge tracking)

- 두 개의 임계값을 사용:  $T_{Low}$ ,  $T_{High}$
- 강한 에지:  $\|f\| \geq T_{High} \rightarrow$  최종 에지로 선정
- 약한 에지:  $T_{Low} \leq \|f\| < T_{High}$   
 $\rightarrow$  강한 에지와 연결되어 있는 픽셀만 최종 에지로 선정



가우시안 필터링

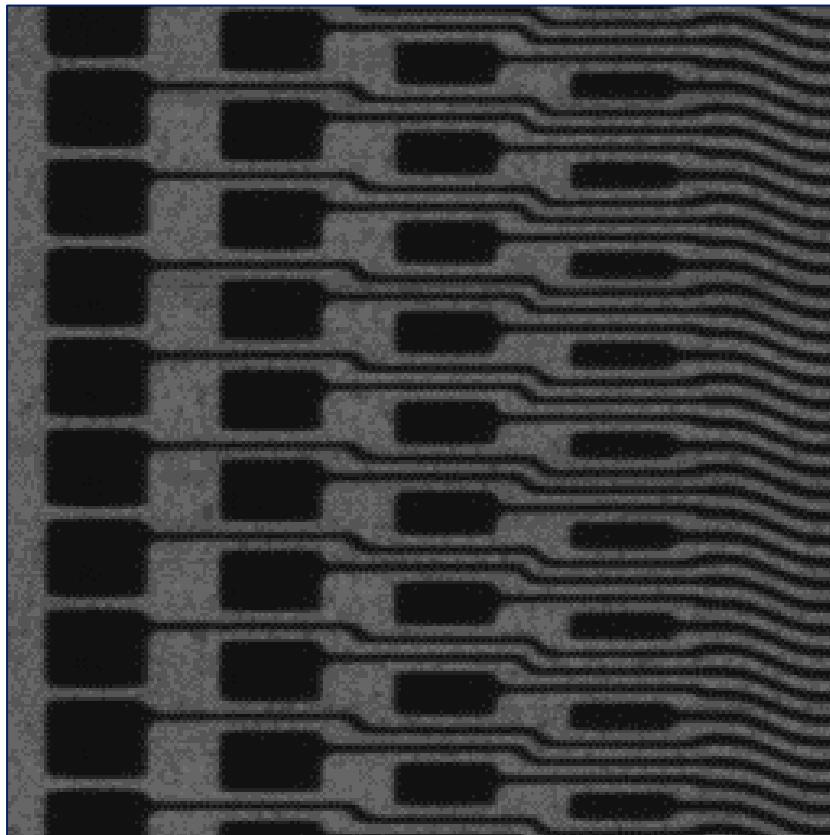
그래디언트 계산  
(크기 & 방향)

비최대 억제

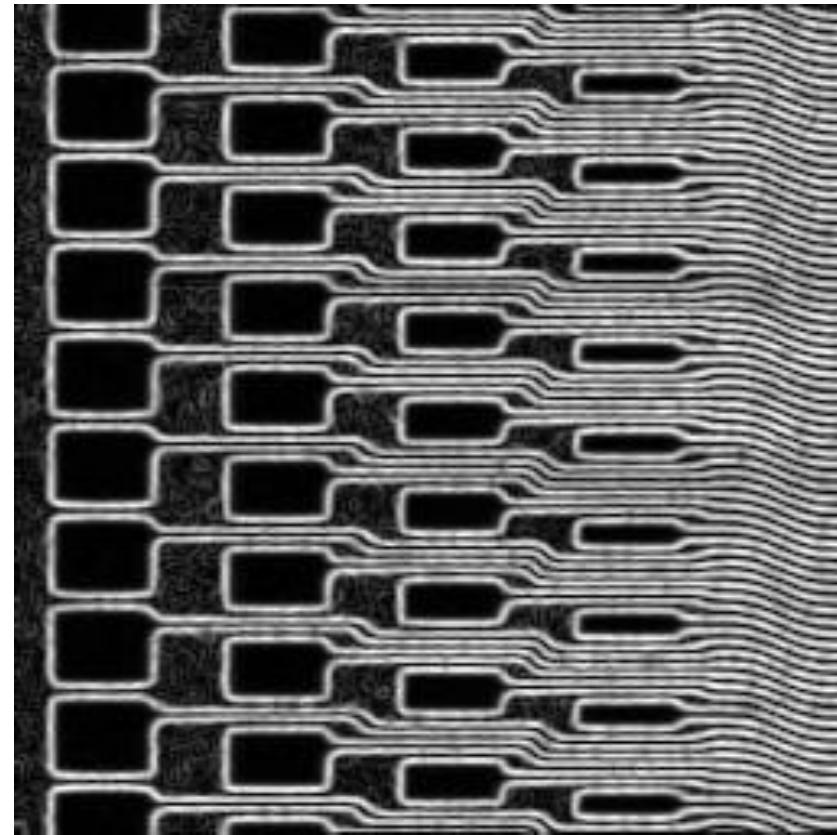
이중 임계값을 이용한  
히스테리시스 에지트래킹

# 캐니 에지 검출

- 캐니 에지 검출 과정



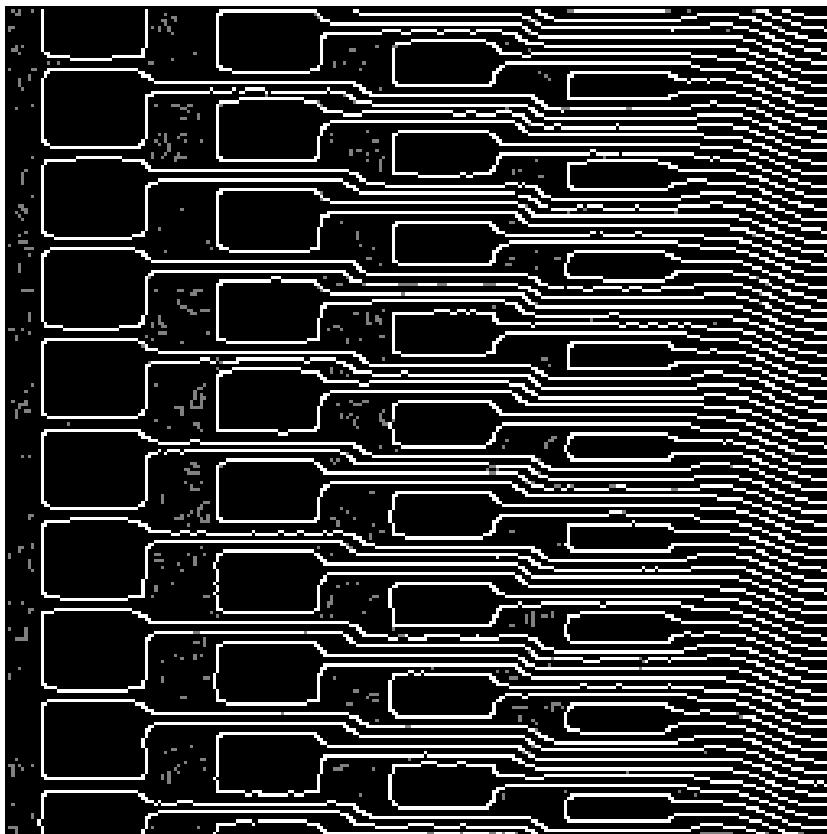
입력 영상



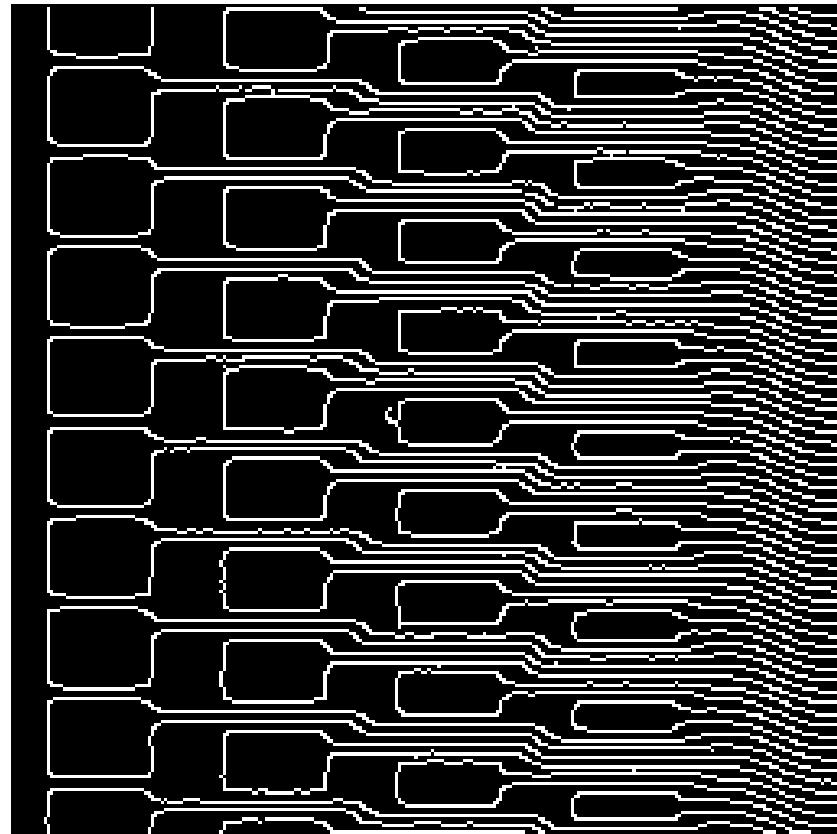
그래디언트 크기

# 캐니 에지 검출

## ■ 캐니 에지 검출 과정



비최대 억제



히스테리시스 에지 트래킹

# 캐니 에지 검출

## ■ 캐니 에지 검출 함수

```
cv2.Canny(image, threshold1, threshold2, edges=None, apertureSize=None,  
          L2gradient=None) -> edges
```

- image: 입력 영상
- threshold1: 하단 임계값
- threshold2: 상단 임계값 } threshold1:threshold2 = 1:2 또는 1:3
- edges: 에지 영상
- apertureSize: 소벨 연산을 위한 커널 크기. 기본값은 3.
- L2gradient: True이면 L2 norm 사용, False이면 L1 norm 사용. 기본값은 False.

$$L_2 \text{ norm} = \sqrt{(dd//ddd)^2 + (dd//ddd)^2}, L_1 \text{ norm} = |dd//ddd| + |dd//ddd|$$

# 캐니 에지 검출

실습: canny.py

## ■ 캐니 에지 검출 예제

```
src = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)  
  
dst = cv2.Canny(src, 50, 150)
```



## 6. 영상의 특징 추출

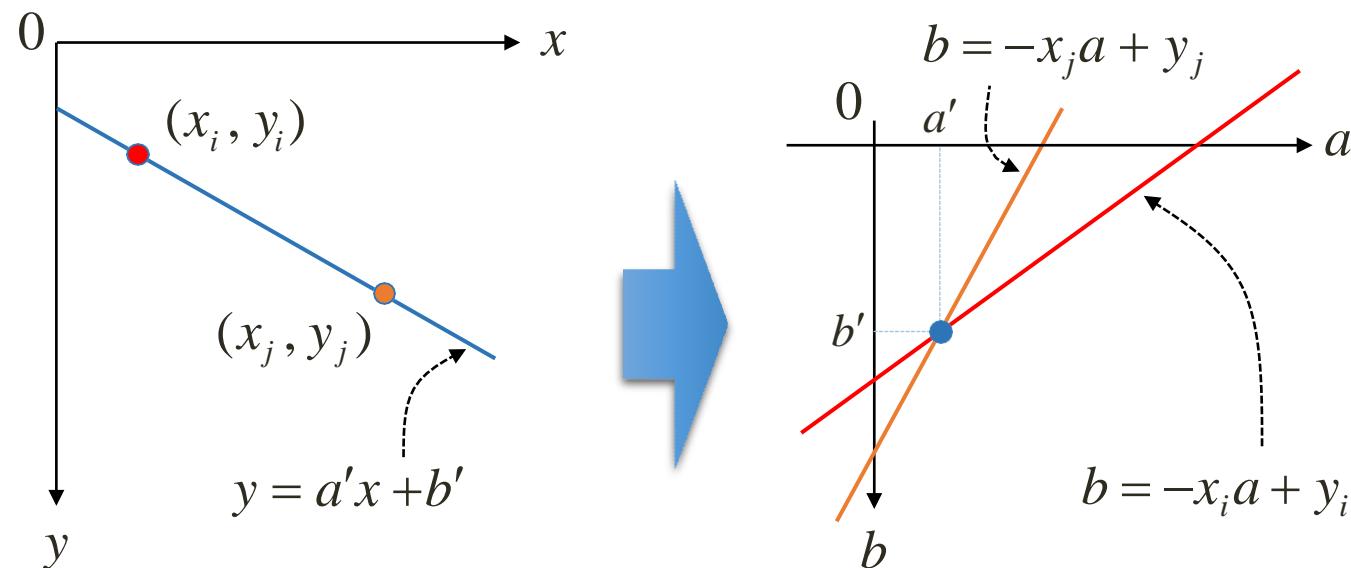
4) 허프 변환: 직선 검출

# 허프 변환: 직선 검출

## ■ 허프 변환(Hough transform) 직선 검출이란?

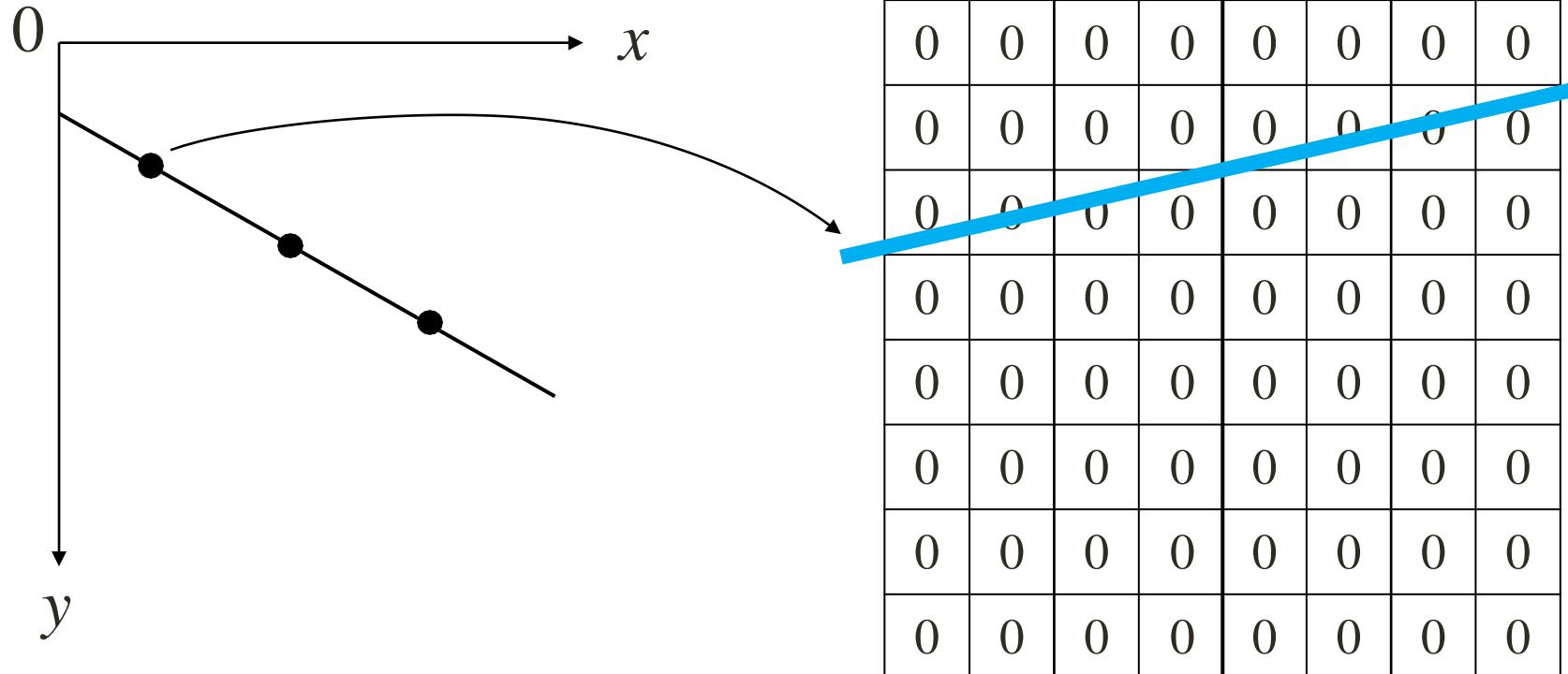
- 2차원 영상 좌표에서의 직선의 방정식을 파라미터(parameter) 공간으로 변환하여  
직선을 찾는 알고리즘

$$y = ax + b \Leftrightarrow b = -xa + y$$



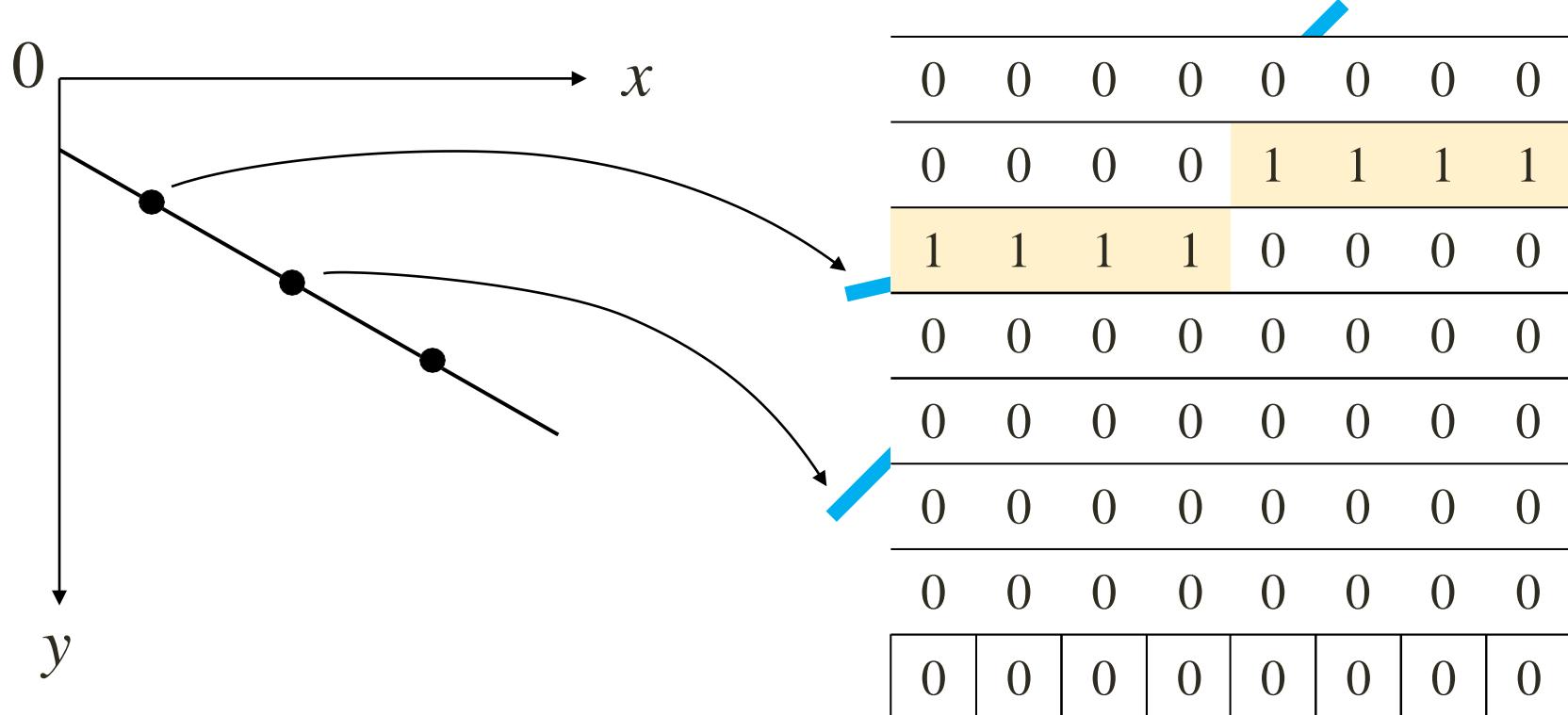
# 허프 변환: 직선 검출

- 축적 배열(accumulation array)
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



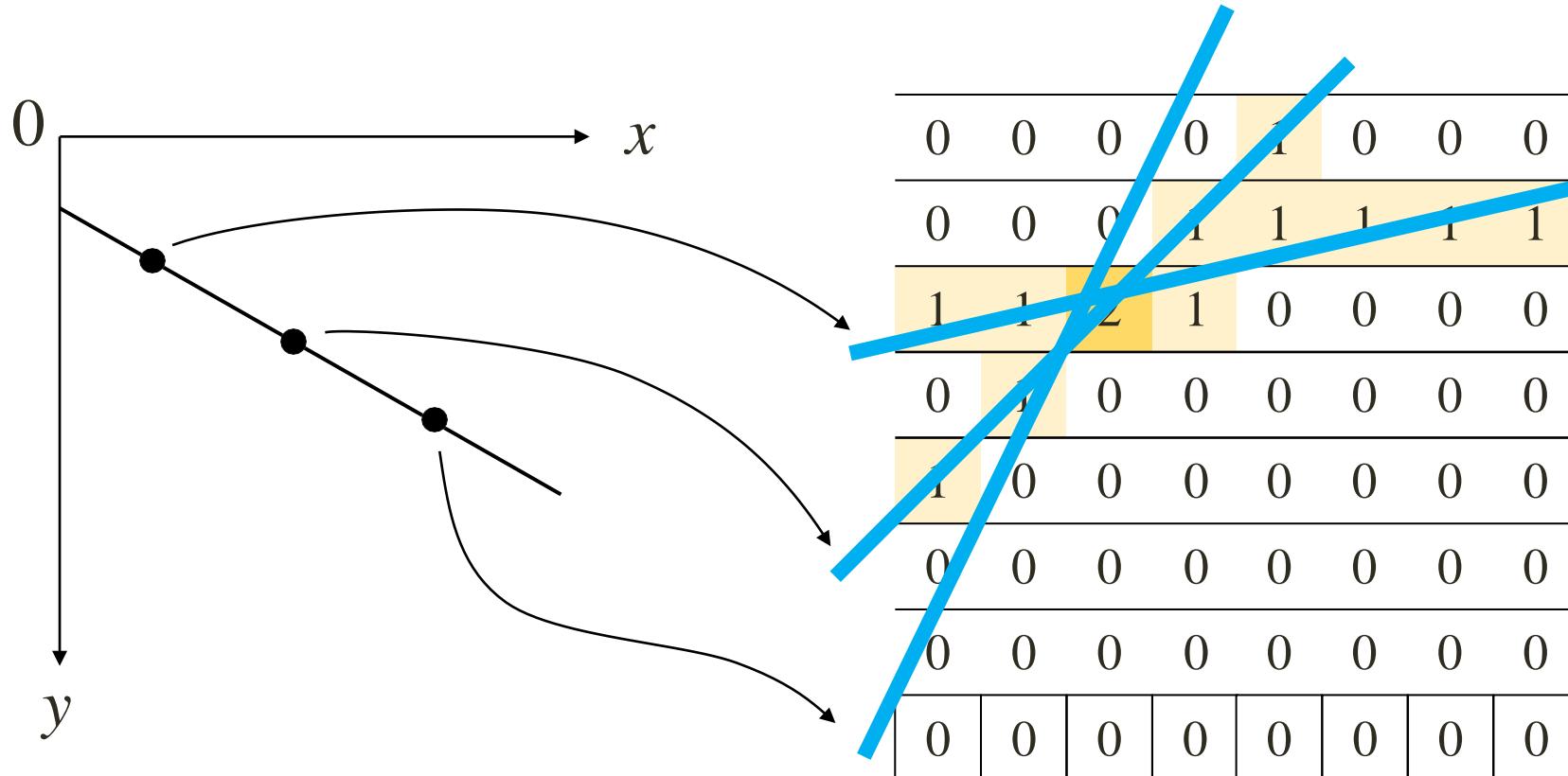
# 허프 변환: 직선 검출

- 축적 배열(accumulation array)
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



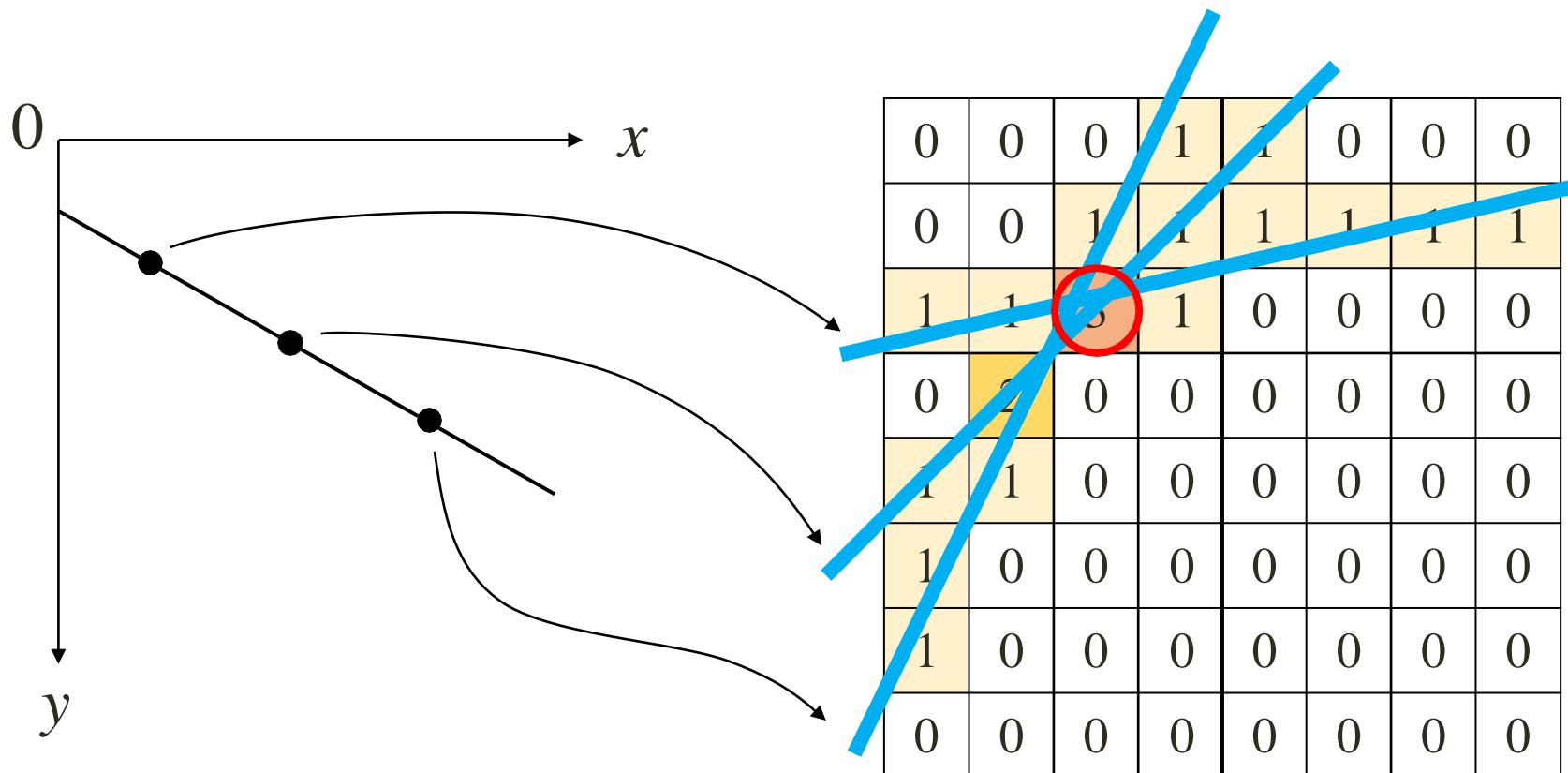
# 허프 변환: 직선 검출

- 축적 배열(accumulation array)
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



# 허프 변환: 직선 검출

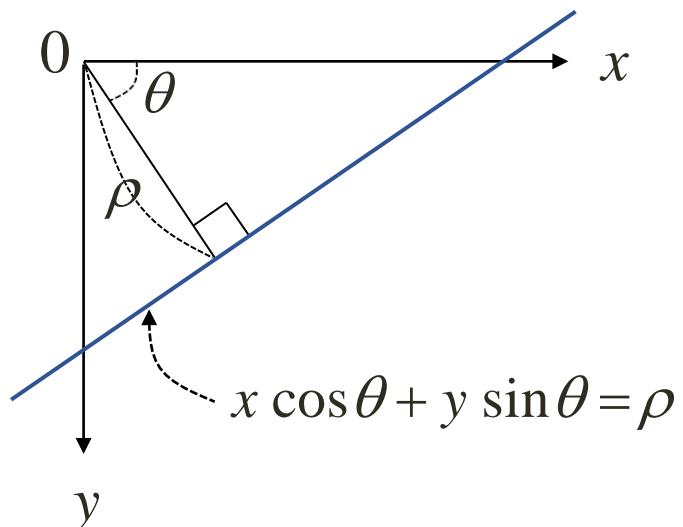
- 축적 배열(accumulation array)
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



# 허프 변환: 직선 검출

- 직선의 방정식  $y = ax + b$  를 사용할 때의 문제점
  - $y$  축과 평행한 수직선을 표현하지 못함  $\rightarrow$  극좌표계 직선의 방정식을 사용

$$x \cos \theta + y \sin \theta = \rho$$



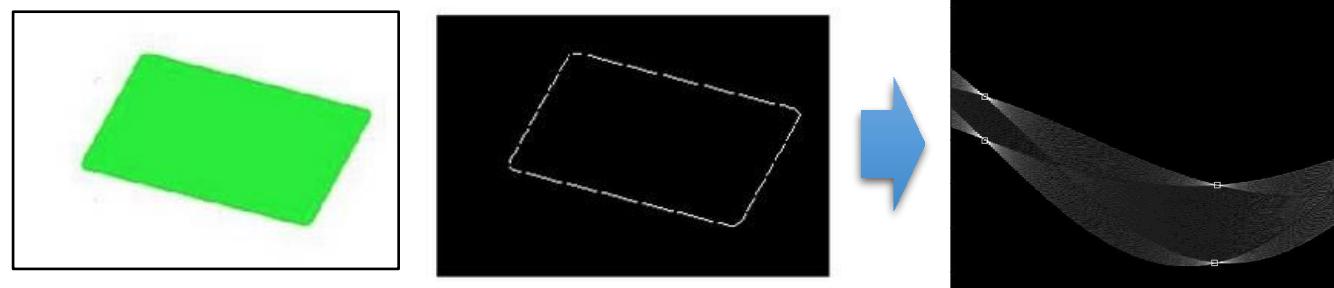
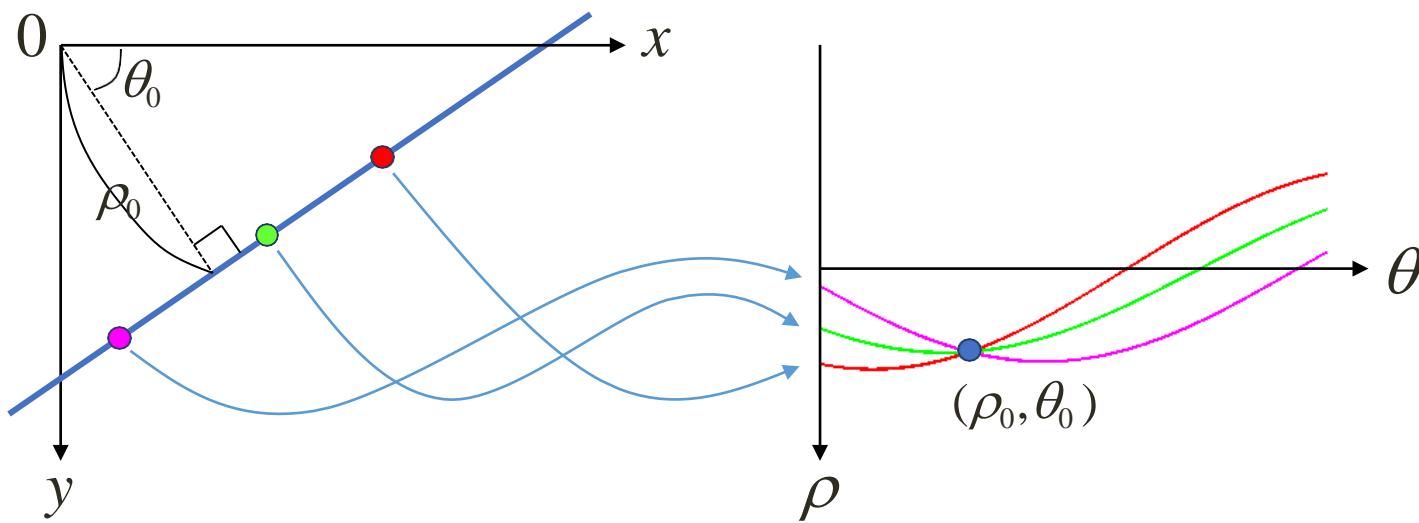
$$\begin{cases} x_{\text{율기}} = -\frac{\cos \theta}{\sin \theta} \\ y_{\text{절편}} = \frac{\rho}{\sin \theta} \end{cases}$$

$$y = -\frac{\cos \theta}{\sin \theta} x + \frac{\rho}{\sin \theta}$$

$$\rightarrow x \cos \theta + y \sin \theta = \rho$$

# 허프 변환: 직선 검출

- $x\cos\theta + y\sin\theta = \rho$  방정식에 의한 파라미터 공간으로의 변환



# 허프 변환: 직선 검출

## ■ 허프 변환에 의한 선분 검출

```
cv2.HoughLines(image, rho, theta, threshold, lines=None, srn=None, stn=None,  
min_theta=None, max_theta=None) -> lines
```

- image: 입력 에지 영상
- rho: 축적 배열에서 rho 값의 간격. (e.g.) 1.0 → 1픽셀 간격.
- theta: 축적 배열에서 theta 값의 간격. (e.g.) np.pi / 180 → 1° 간격.
- threshold: 축적 배열에서 직선으로 판단할 임계값
- lines: 직선 파라미터(rho, theta) 정보를 담고 있는 numpy.ndarray.  
shape=(N, 1, 2). dtype=numpy.float32.
- srn, stn: 멀티 스케일 허프 변환에서 rho 해상도, theta 해상도를 나누는 값.  
기본값은 0이고, 이 경우 일반 허프 변환 수행.
- min\_theta, max\_theta: 검출할 선분의 최소, 최대 theta 값

# 허프 변환: 직선 검출

## ■ 확률적 허프 변환에 의한 선분 검출

```
cv2.HoughLinesP(image, rho, theta, threshold, lines=None,  
minLineLength=None, maxLineGap=None) -> lines
```

- image: 입력 에지 영상
- rho: 축적 배열에서 rho 값의 간격. (e.g.) 1.0 → 1픽셀 간격.
- theta: 축적 배열에서 theta 값의 간격. (e.g.) np.pi / 180 → 1° 간격.
- threshold: 축적 배열에서 직선으로 판단할 임계값
- lines: 선분의 시작과 끝 좌표(x1, y1, x2, y2) 정보를 담고 있는 numpy.ndarray.  
shape=(N, 1, 4). dtype=numpy.int32.
- minLineLength: 검출할 선분의 최소 길이
- maxLineGap: 직선으로 간주할 최대 에지 점 간격

# 허프 변환: 직선 검출

실습: hough\_lines.py

## ■ 확률적 허프 변환 직선 검출 예제

```
src = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)

edges = cv2.Canny(src, 50, 150)

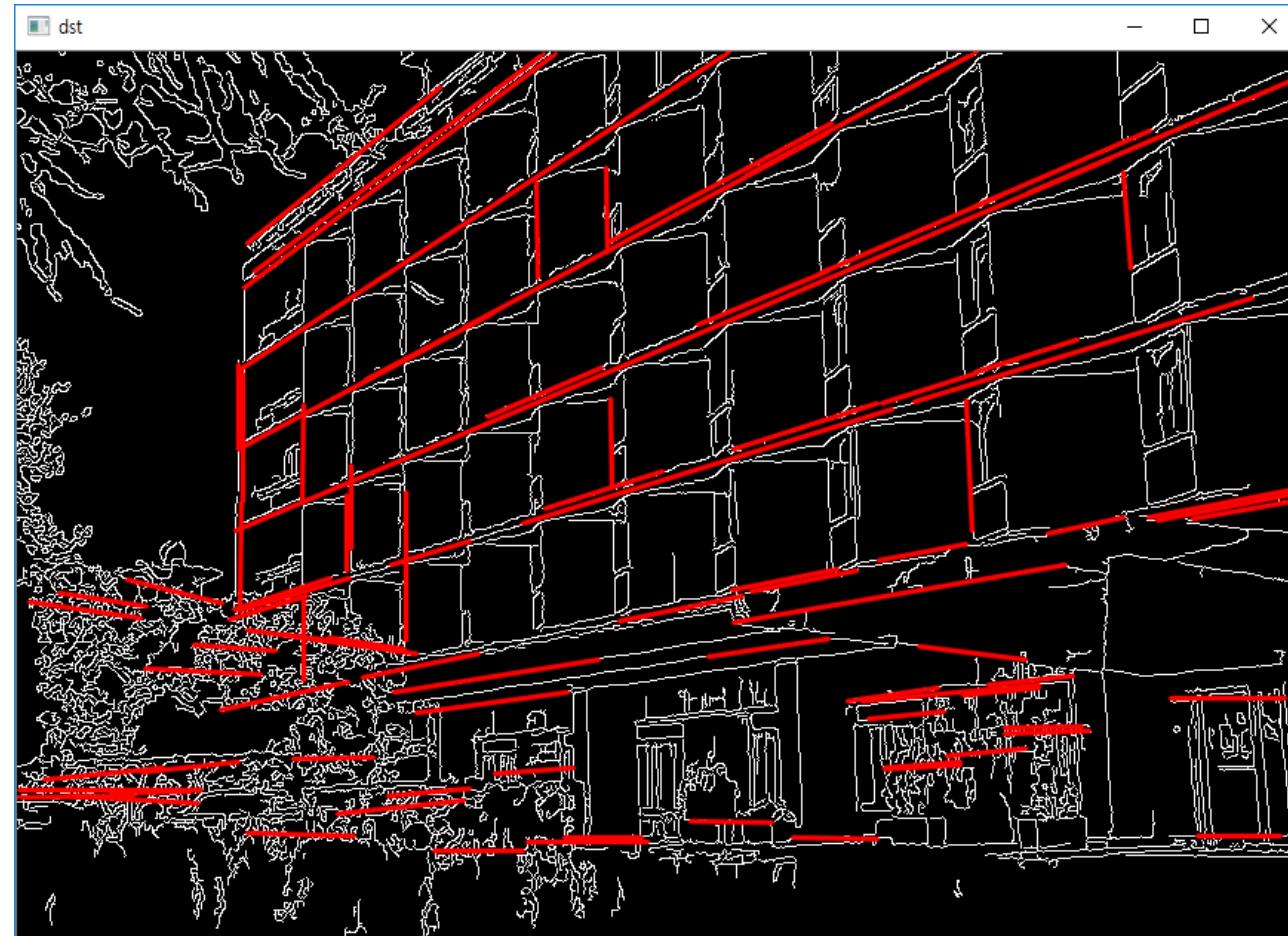
lines = cv2.HoughLinesP(edges, 1.0, np.pi / 180., 160,
                        minLineLength=50, maxLineGap=5)

dst = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

if lines is not None:
    for i in range(lines.shape[0]):
        pt1 = (lines[i][0][0], lines[i][0][1]) # 시작점 좌표
        pt2 = (lines[i][0][2], lines[i][0][3]) # 끝점 좌표
        cv2.line(dst, pt1, pt2, (0, 0, 255), 2, cv2.LINE_AA)
```

# 허프 변환: 직선 검출

- 확률적 허프 변환 직선 검출 예제 실행 결과



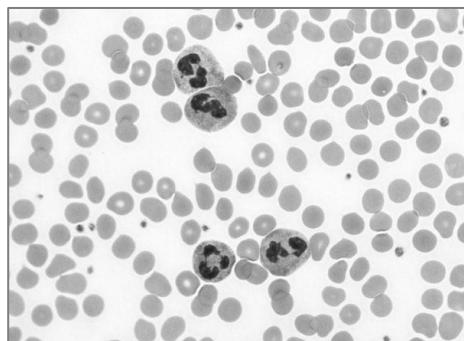
# 7. 이진 영상처리

## 1) 영상의 이진화

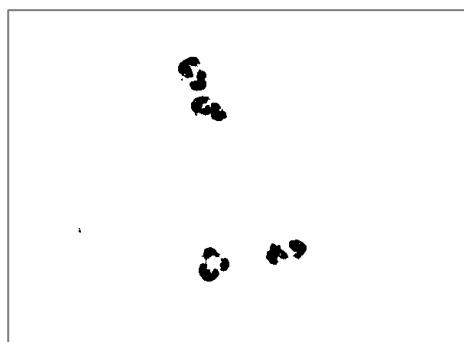
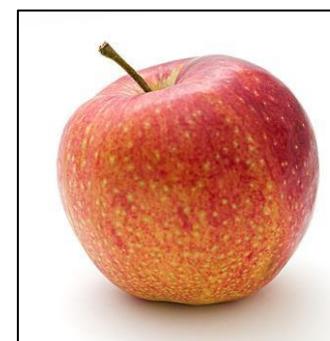
# 영상의 이진화

## ■ 영상의 이진화(Binarization)란?

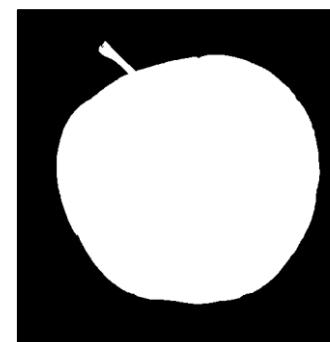
- 영상의 픽셀 값을 0 또는 255(1)로 만드는 연산
  - 배경(background) vs. 객체(object)
  - 관심 영역 vs. 비관심 영역



motivation for this come  
e segmentation and regio  
ic circumstances. By re  
peating methods are app  
variety of sources, occu  
t to the majority of dat  
mostly structured da



motivation for this come  
e segmentation and regio  
ic circumstances. By re  
peating methods are app  
variety of sources, occu  
t to the majority of dat  
mostly structured da

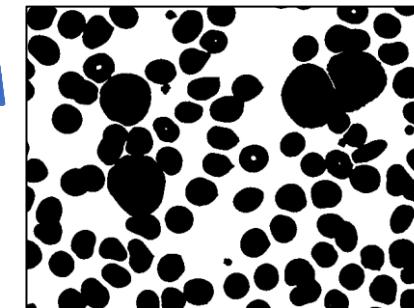
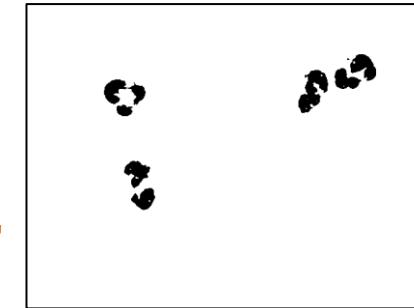
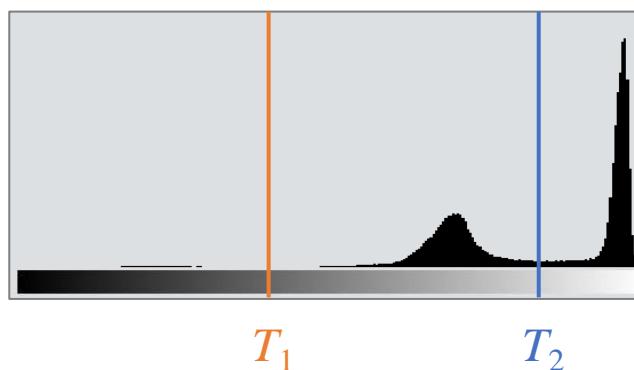
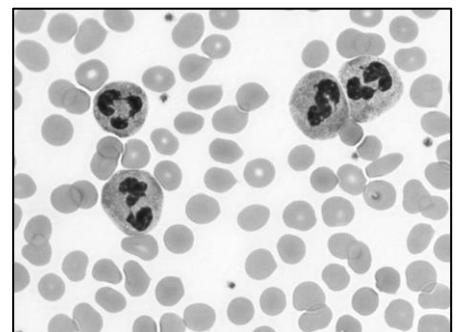


# 영상의 이진화

## ■ 그레이스케일 영상의 이진화

$$g(x, y) = \begin{cases} 0 & \text{if } f(x, y) \leq T \\ 255 & \text{if } f(x, y) > T \end{cases}$$

- $T$ : 임계값, 문턱치, threshold



# 영상의 이진화

## ■ 임계값 함수

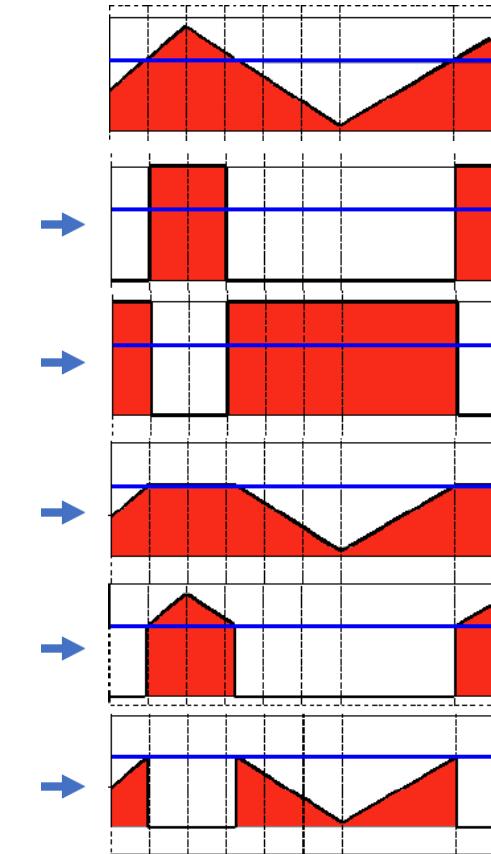
```
cv2.threshold(src, thresh, maxval, type, dst=None) -> retval, dst
```

- src: 입력 영상. 다채널, 8비트 또는 32비트 실수형.
- thresh: 사용자 지정 임계값
- maxval: cv2.THRESH\_BINARY 또는 cv2.THRESH\_BINARY\_INV 방법 사용 시 최댓값. 보통 255로 지정.
- type: cv2.THRESH\_로 시작하는 플래그. 임계값 함수 동작 지정 또는 자동 임계값 결정 방법 지정
- retval: 사용된 임계값
- dst: 출력 영상. src와 동일 크기, 동일 타입, 같은 채널 수.

# 영상의 이진화

- cv2.threshold() 함수 동작 타입

cv2.THRESH_BINARY	$dst(x,y) = \begin{cases} maxval & \text{if } src(x,y) > thresh \\ 0 & \text{otherwise} \end{cases}$
cv2.THRESH_BINARY_INV	$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > thresh \\ maxval & \text{otherwise} \end{cases}$
cv2.THRESH_TRUNC	$dst(x,y) = \begin{cases} threshold & \text{if } src(x,y) > thresh \\ src(x,y) & \text{otherwise} \end{cases}$
cv2.THRESH_TOZERO	$dst(x,y) = \begin{cases} src(x,y) & \text{if } src(x,y) > thresh \\ 0 & \text{otherwise} \end{cases}$
cv2.THRESH_TOZERO_INV	$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > thresh \\ src(x,y) & \text{otherwise} \end{cases}$
cv2.THRESH_OTSU	Otsu 알고리즘으로 임계값 자동 결정
cv2.THRESH_TRIANGLE	삼각 알고리즘으로 임계값 자동 결정

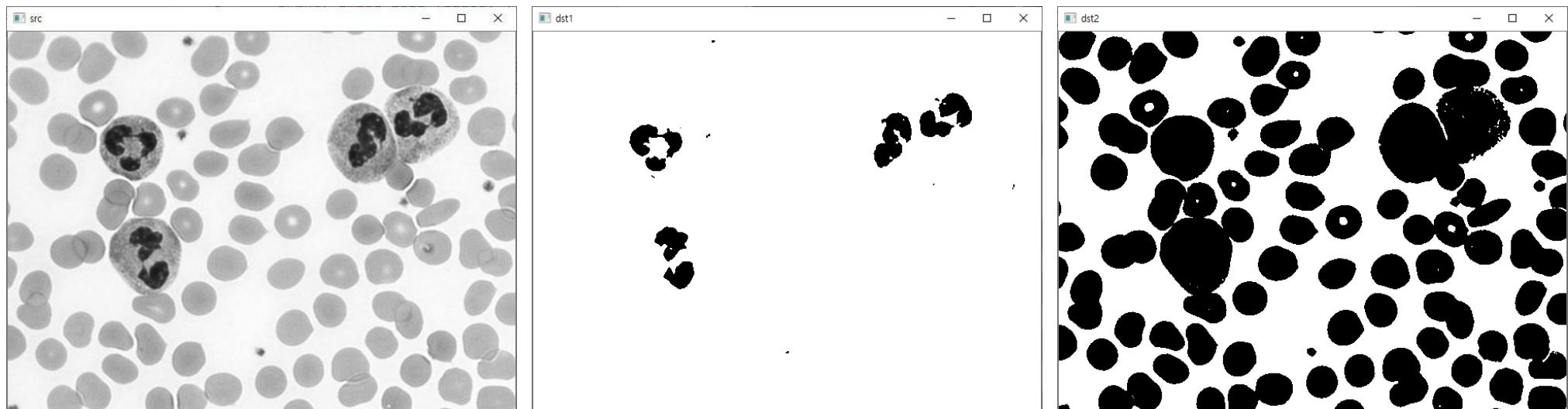


# 영상의 이진화

실습: threshold1.py

## ■ 영상의 이진화 예제

```
src = cv2.imread('cells.png', cv2.IMREAD_GRAYSCALE)  
  
_, dst1 = cv2.threshold(src, 100, 255, cv2.THRESH_BINARY)  
_, dst2 = cv2.threshold(src, 210, 255, cv2.THRESH_BINARY)
```



# 영상의 이진화

실습: threshold2.py

- 트랙바를 이용한 영상의 이진화 예제

```
src = cv2.imread('cells.png', cv2.IMREAD_GRAYSCALE)

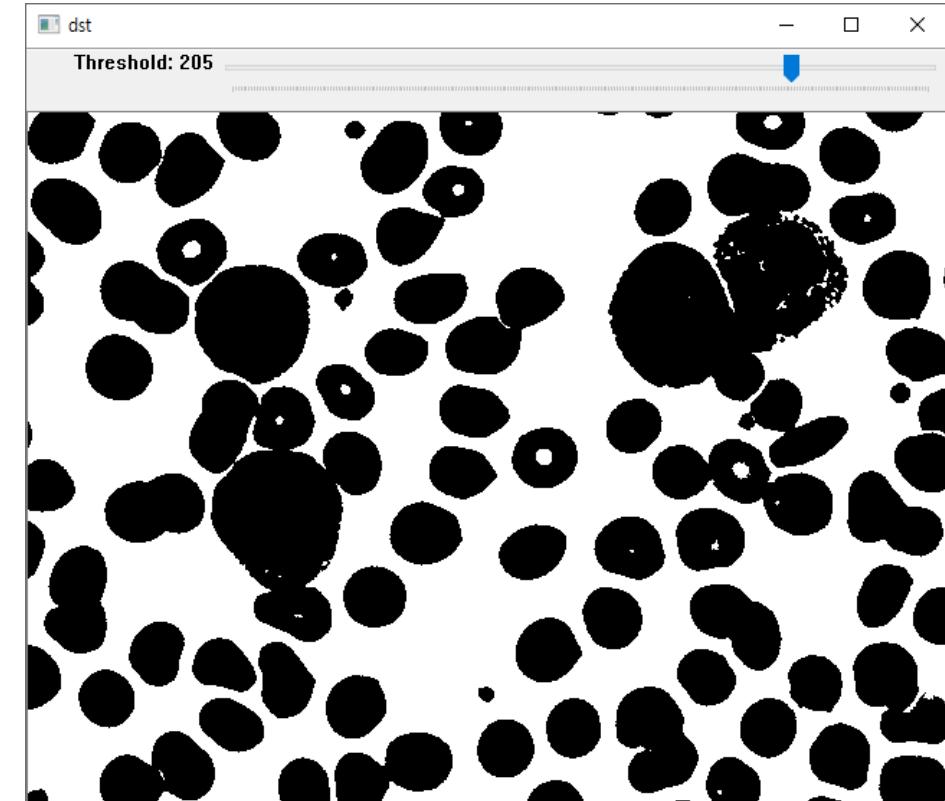
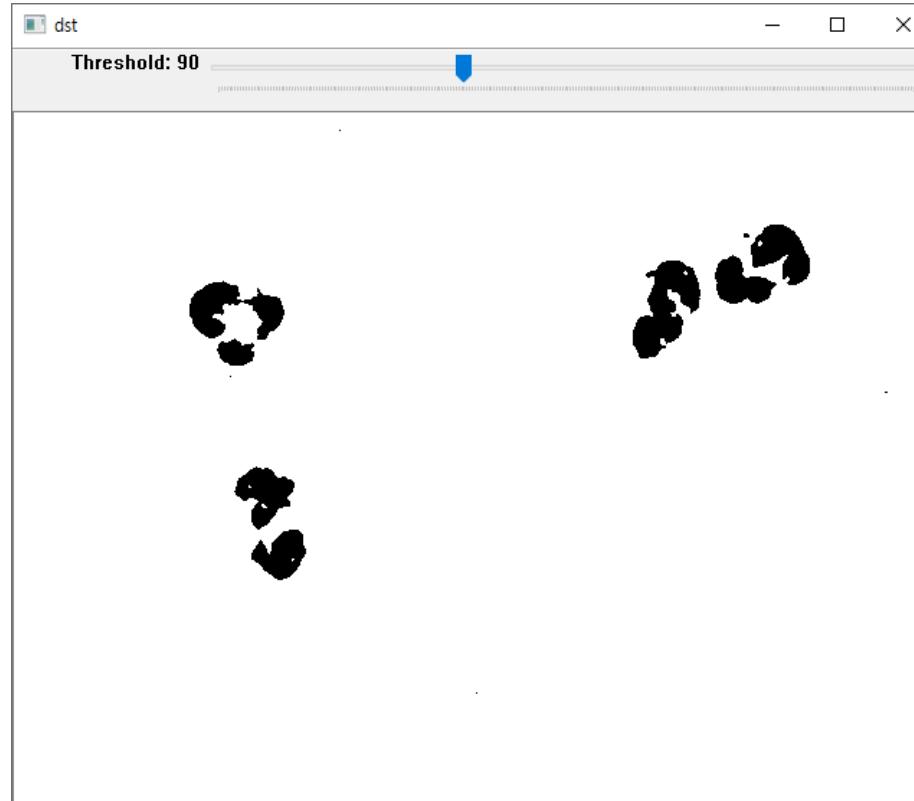
def on_threshold(pos):
    _, dst = cv2.threshold(src, pos, 255, cv2.THRESH_BINARY)
    cv2.imshow('dst', dst)

cv2.imshow('src', src)
cv2.namedWindow('dst')
cv2.createTrackbar('Threshold', 'dst', 0, 255, on_threshold)
cv2.setTrackbarPos('Threshold', 'dst', 128)

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

# 영상의 이진화

- 트랙바를 이용한 영상의 이진화 예제 실행 결과



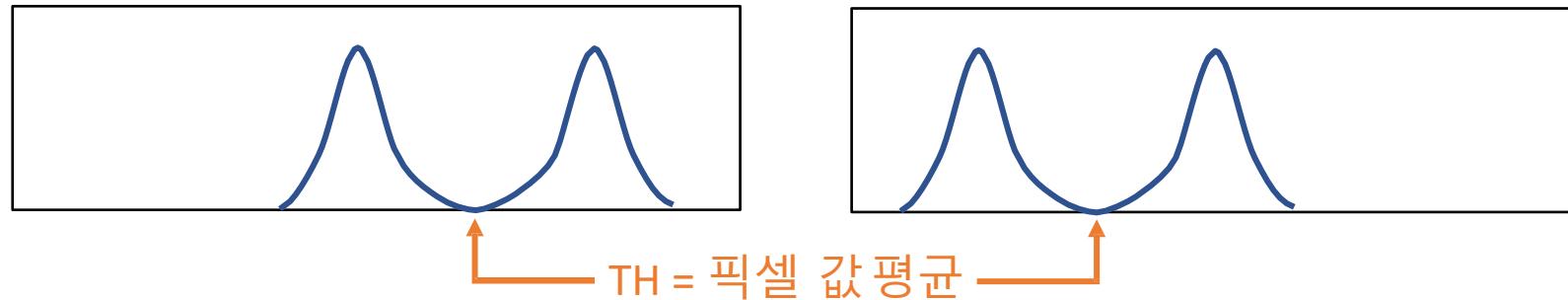
## 7. 이진 영상처리

2) 자동 이진화: Otsu 방법

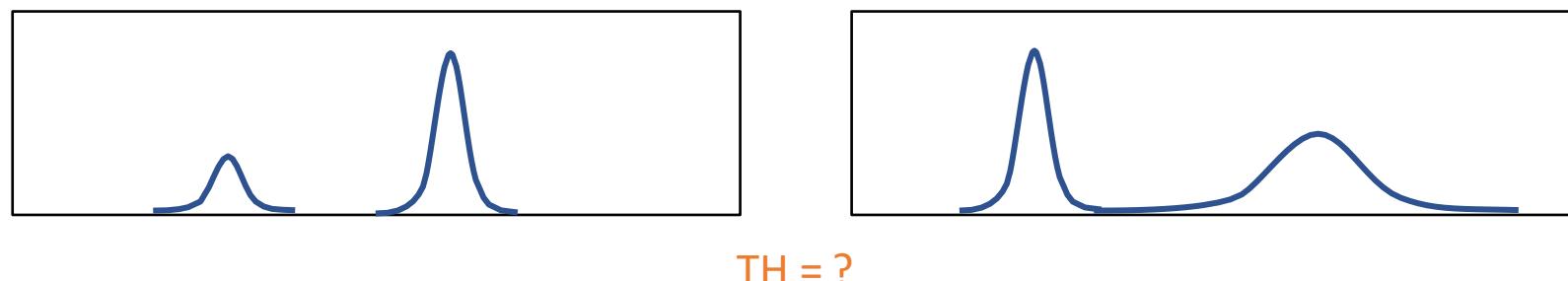
# 자동 이진화

## ■ 임계값 자동 결정 방법

- 영상의 히스토그램이 bimodal이고, 전경&배경 픽셀 분포가 비슷하다면?



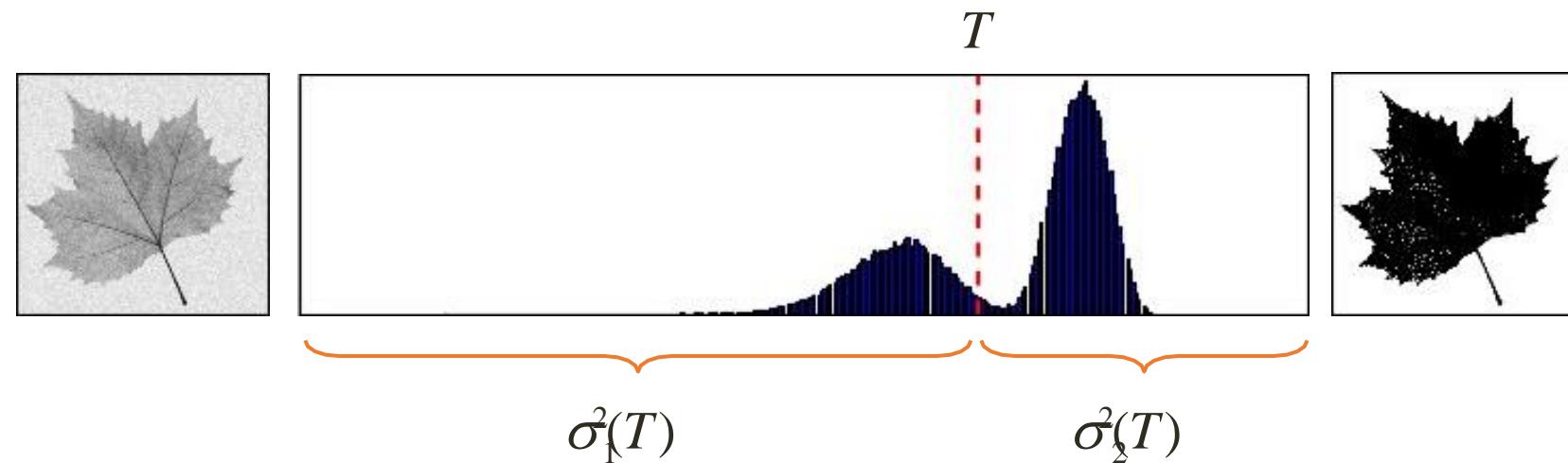
- 히스토그램이 bimodal이지만, 전경&배경 픽셀 분포가 크게 다르다면?



# 자동 이진화: OTSU 방법

## ■ Otsu 이진화 방법

- 입력 영상이 배경(background)과 객체(object) 두 개로 구성되어 있다고 가정
- ⑦ Bimodal histogram
- 임의의 임계값  $T$ 에 의해 나눠지는 두 픽셀 분포 그룹의 분산이 최소가 되는  $T$ 를 선택
- 일종의 최적화 알고리즘(optimization algorithm)



# 자동 이진화: OTSU 방법

## ■ Otsu 이진화 방법

- Within-class variance:

$$\sigma_{\text{Within}}^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

$$\omega_1(t) = \sum_{i=0}^t p(i), \quad \omega_2(t) = \sum_{i=t+1}^{L-1} p(i)$$

↑  
1번 클래스 가중치  
↑  
피셀 값  $i$ 가 나타날 확률  
(정규화된 히스토그램 값)  
↑  
2번 클래스 가중치

- Otsu 임계값:

$$T = \operatorname{argmin}_{t \in \{0,1,\dots,L-1\}} \sigma_{\text{Within}}^2(t)$$

# 자동 이진화: OTSU 방법

## ■ Otsu 이진화 방법

- Within-class variance 최소화 ⑦ Between-class variance 최대화

$$\begin{aligned}\sigma_{\text{Between}}^2(t) &= \sigma^2 - \sigma_{\text{Within}}^2(t) \\ &= \omega(t) (-\omega_1(t))(\mu(t) - \mu_1)^2\end{aligned}$$

- 모든  $t$  값에 대해  $\sigma_{\text{Between}}^2(t)$ 를 구하여 최적의  $T$ 를 선택 ⑦ Show
- Recursion을 이용한 효율적 계산 ⑦ Eat

$$\alpha(0) = p(0), \mu(0) = 0$$

$$\alpha(t) = \alpha(t-1) + p(t)$$

$$\mu(t) = \frac{\alpha(t-1)\mu(t-1) + tp(t)}{\alpha(t)} \quad \mu_1(t) = \frac{\mu - \alpha(t)\mu_1(t)}{1 - \alpha(t)}$$

# 자동 이진화: OTSU 방법

## ■ Otsu 이진화 방법

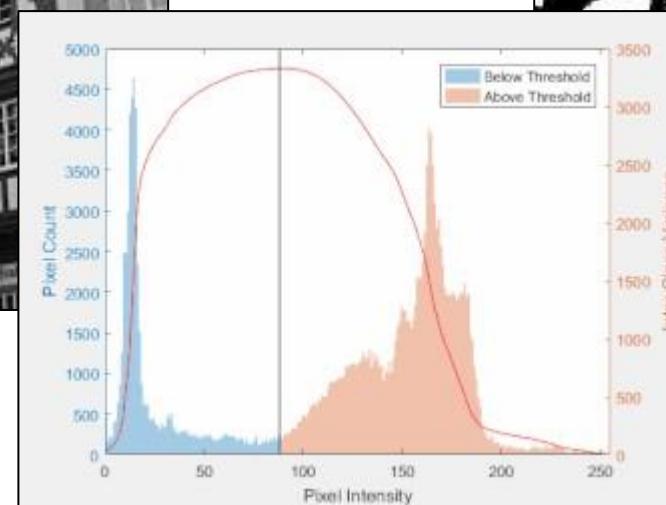


Image from [https://en.wikipedia.org/wiki/Otsu%27s\\_method](https://en.wikipedia.org/wiki/Otsu%27s_method)

# 자동 이진화: OTSU 방법

실습: otsu.py

## ■ Otsu 방법을 이용한 자동 이진화

```
src = cv2.imread('rice.png', cv2.IMREAD_GRAYSCALE)

th, dst = cv2.threshold(src, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
print("otsu's threshold:", th) # 131
```

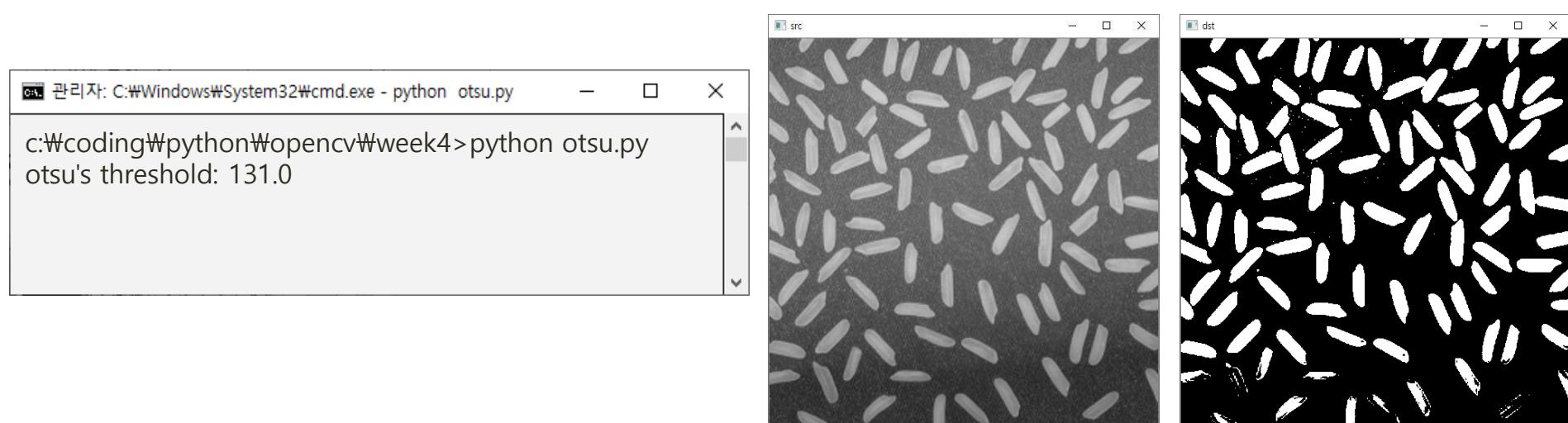


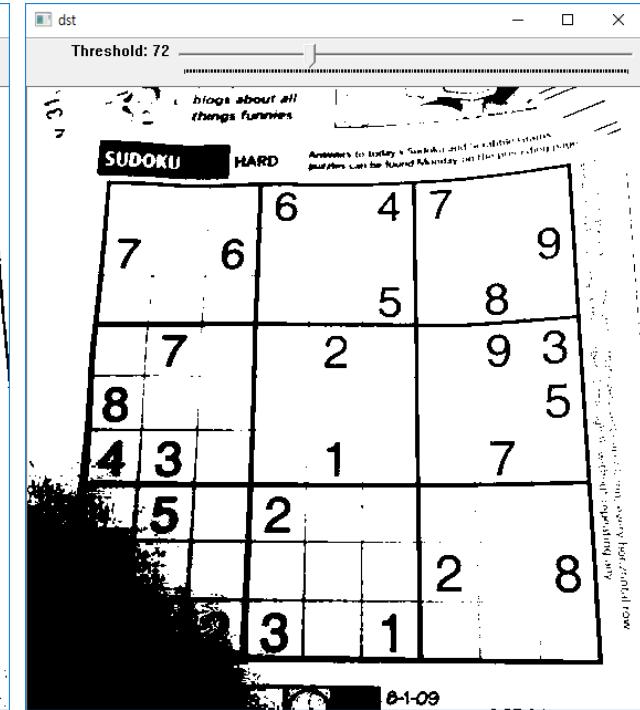
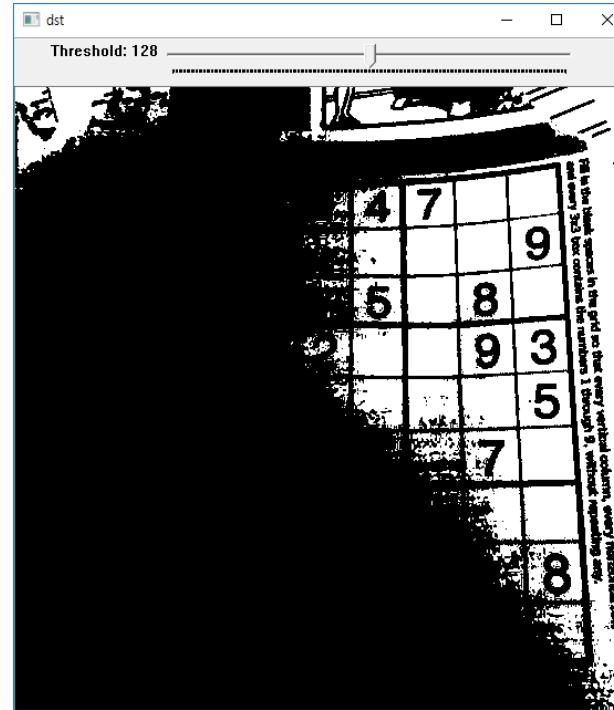
Image from <https://www.mathworks.com/company/newsletters/articles/new-features-for-high-performance-image-processing-in-matlab.html>

## 7. 이진 영상처리

### 3) 지역 이진화

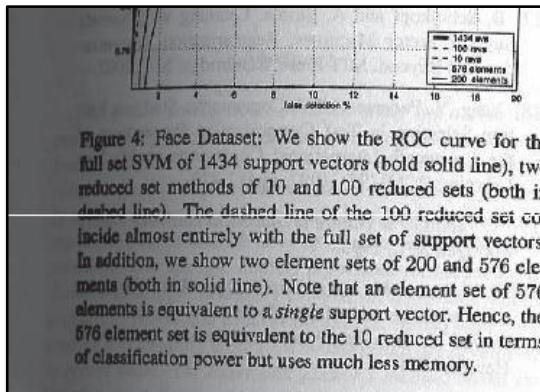
# 지역 이진화

- 균일하지 않은 조명 환경에서 촬영된 영상의 이진화
  - threshold2.py 프로그램에 sudoku.jpg 파일을 입력으로 사용

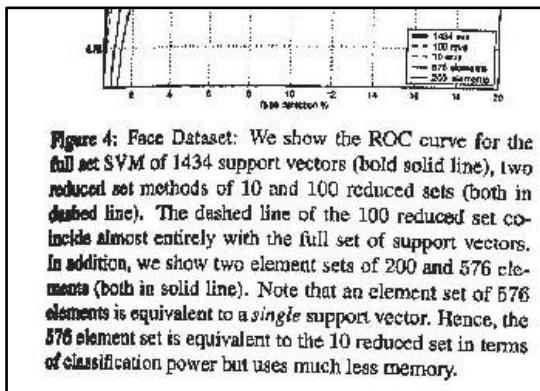
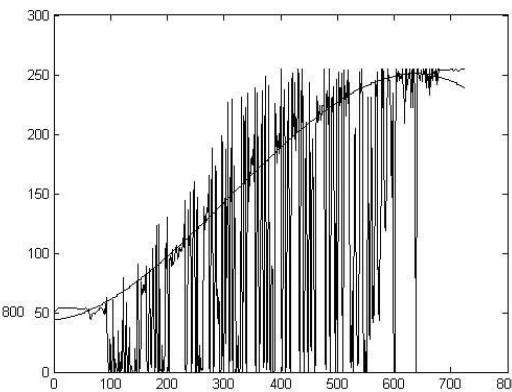
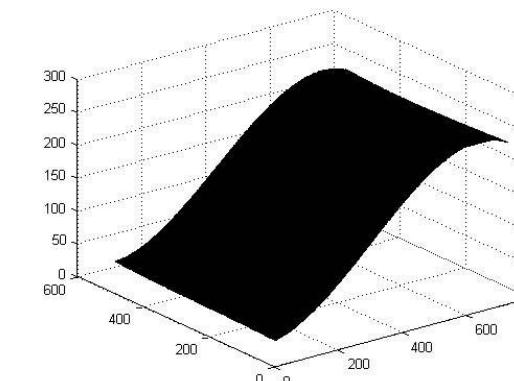


# 지역 이진화

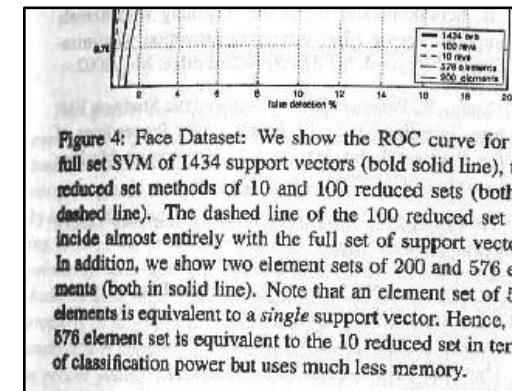
- 균일하지 않은 조명의 영향을 해결하려면?
  - 불균일한 조명 성분을 보상한 후 전역 이진화 수행



Surface  
Fitting



Global  
Thresholding



Shading  
Compensation

# 지역 이진화

- 균일하지 않은 조명의 영향을 해결하려면?
  - 픽셀 주변에 작은 윈도우를 설정하여 지역 이진화 수행
    - 윈도우의 크기는?
    - 윈도우 형태는? Uniform? Gaussian?
    - 윈도우를 겹칠 것인가? Overlap? Non-overlap?
    - 윈도우 안에 배경 또는 객체만 존재한다면?



# 지역 이진화

실습: local\_th.py

## ■ 지역 이진화 예제

```
src = cv2.imread('rice.png', cv2.IMREAD_GRAYSCALE)

# 전역 이진화 by Otsu's method
_, dst1 = cv2.threshold(src, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

# 지역 이진화 by Otsu's method
dst2 = np.zeros(src.shape, np.uint8)

bw = src.shape[1] // 4
bh = src.shape[0] // 4

for y in range(4):
    for x in range(4):
        src_ = src[y*bh:(y+1)*bh, x*bw:(x+1)*bw]
        dst_ = dst2[y*bh:(y+1)*bh, x*bw:(x+1)*bw]
        cv2.threshold(src_, 0, 255, cv2.THRESH_BINARY|cv2.THRESH_OTSU, dst_)
```

# 지역 이진화

- 지역 이진화 예제 실행 결과



# 지역 이진화

## ■ OpenCV 적응형 이진화

```
cv2.adaptiveThreshold(src, maxValue, adaptiveMethod,  
                      thresholdType, blockSize, C, dst=None) -> dst
```

- src: 입력 영상. 그레이스케일 영상.
- maxValue: 임계값 함수 최댓값. 보통 255.
- adaptiveMethod: 블록 평균 계산 방법 지정. cv2.ADAPTIVE\_THRESH\_MEAN\_C는 산술 평균, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C는 가우시안 가중치 평균
- thresholdType: cv2.THRESH\_BINARY 또는 cv2.THRESH\_BINARY\_INV 지정
- blockSize: 블록 크기. 3 이상의홀수.
- C: 블록 내 평균값 또는 블록 내 가중 평균값에서 뺄 값.  
 $(x, y)$  픽셀의 임계값으로  $T(x, y) = \mu_B(x, y) - C$  를 사용

# 지역 이진화

실습: adaptive\_th.py

## ■ OpenCV 적용형 이진화 예제

```
src = cv2.imread('sudoku.jpg', cv2.IMREAD_GRAYSCALE)

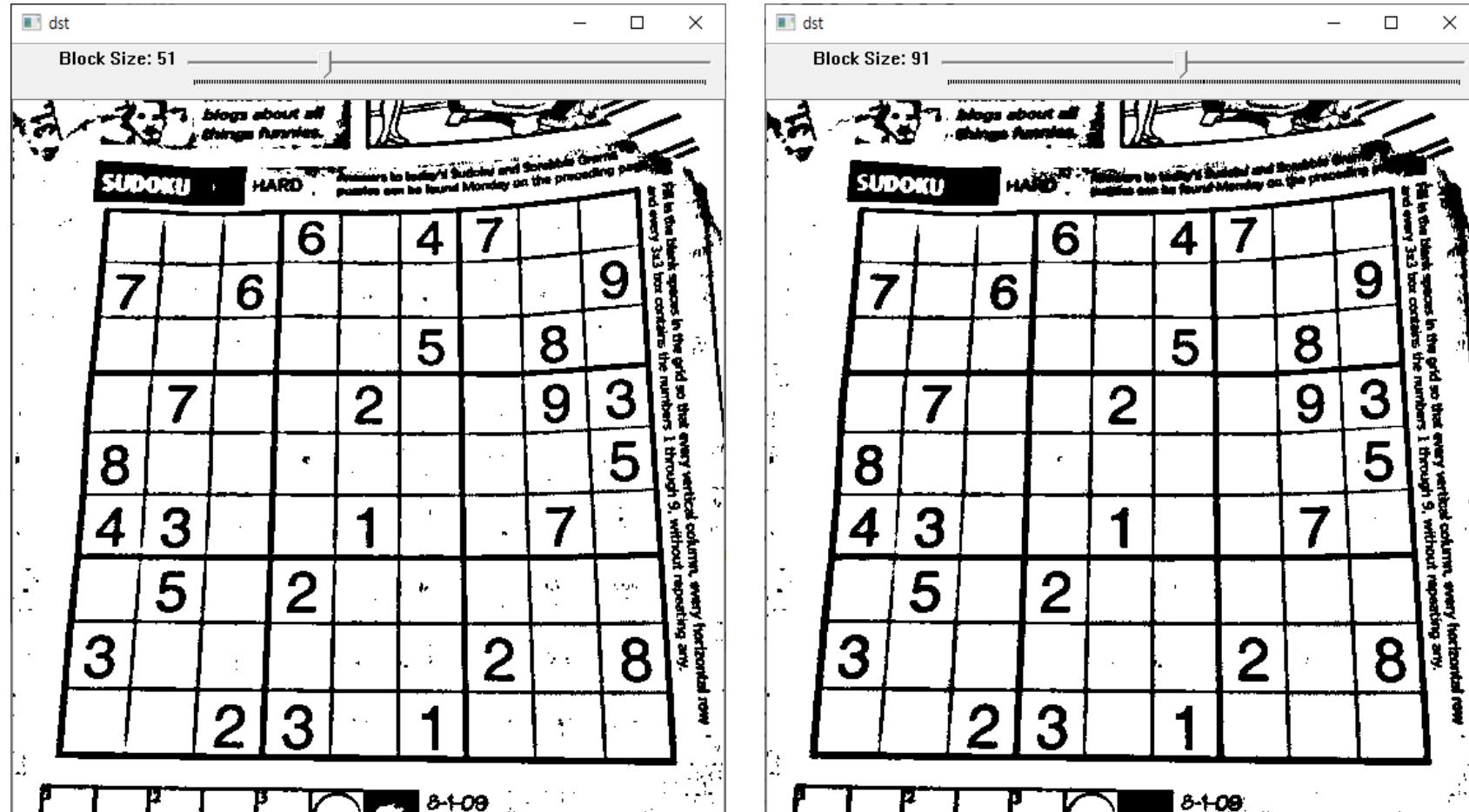
def on_trackbar(pos):
    bsize = pos
    if bsize % 2 == 0:
        bsize = bsize - 1
    if bsize < 3:
        bsize = 3

    dst = cv2.adaptiveThreshold(src, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                cv2.THRESH_BINARY, bsize, 5)
    cv2.imshow('dst', dst)

    cv2.imshow('src', src)
    cv2.namedWindow('dst')
    cv2.createTrackbar('Block Size', 'dst', 0, 200, on_trackbar)
    cv2.setTrackbarPos('Block Size', 'dst', 11)
```

# 지역 이진화

## ■ OpenCV 적응형 이진화 예제 실행 결과



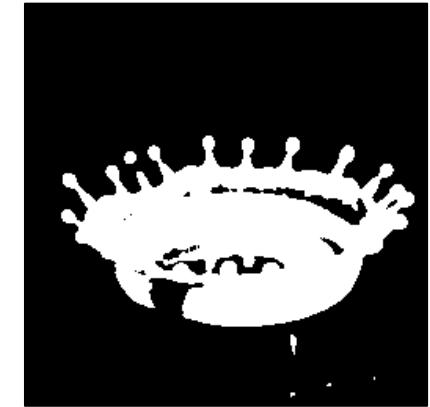
## 7. 이진 영상처리

4) 모폴로지 (1): 침식과팽창

# 모폴로지

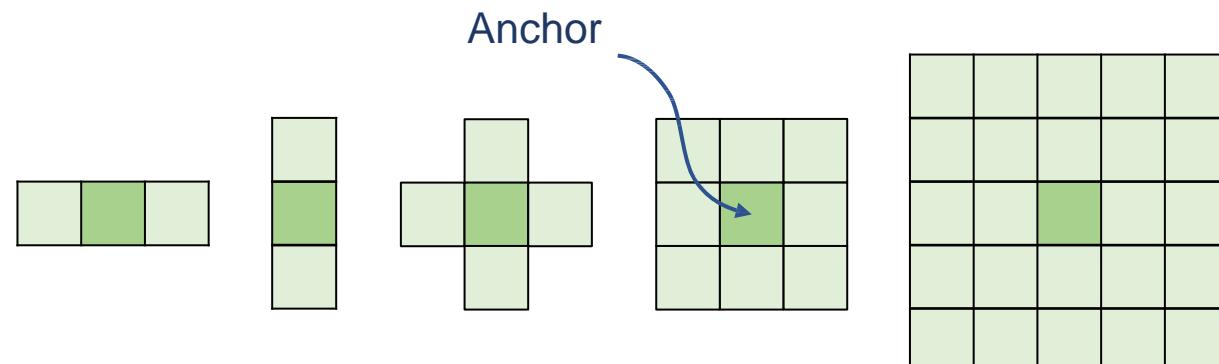
## ■ 모폴로지(Morphology) 연산이란?

- 영상을 형태학적인 측면에서 다루는 기법
- 다양한 영상 처리 시스템에서 전처리(pre-processing) 또는 후처리(post-processing) 형태로 널리 사용
- 수학적 모폴로지(mathematical morphology)



## ■ 구조 요소(Structuring element)

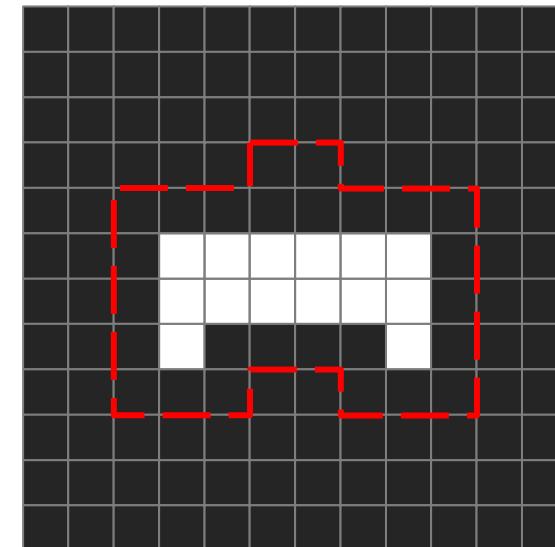
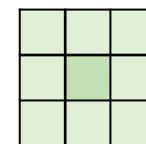
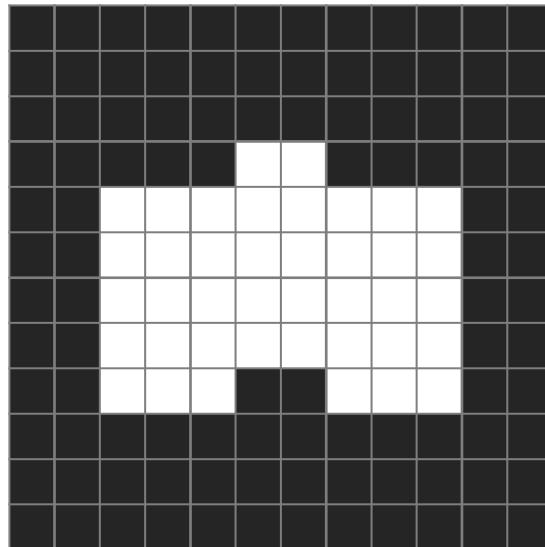
- 모폴로지 연산의 결과를 결정하는 커널, 마스크, 윈도우



# 모폴로지 (1): 침식과 팽창

## ■ 이진 영상의 침식(erosion) 연산

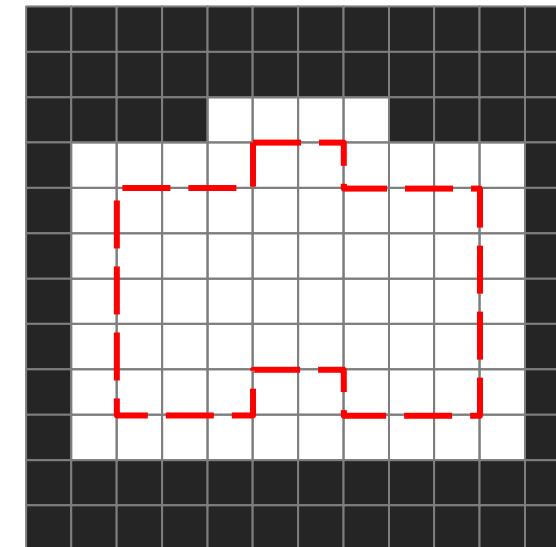
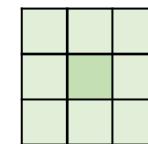
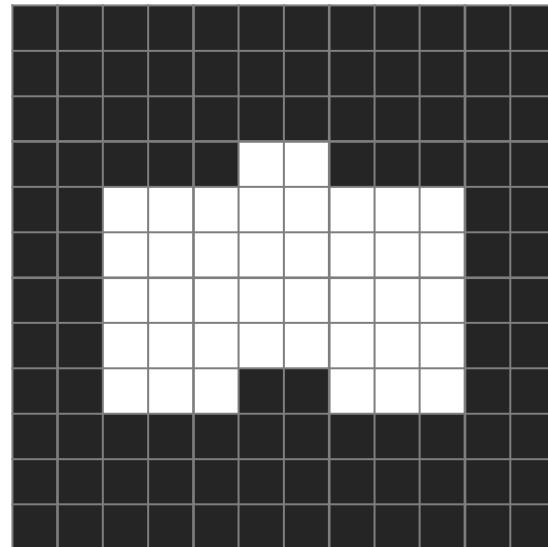
- 구조 요소가 객체 영역 내부에 완전히 포함될 경우 고정점 픽셀을 255로 설정
- 침식 연산은 객체 외곽을 깎아내는 연산 ⑦ 객체 크기는 감소 & 배경은 확대



# 모폴로지 (1): 침식과 팽창

## ■ 이진 영상의 팽창(dilation) 연산

- 구조 요소와 객체 영역이 한 픽셀이라도 만날 경우 고정점 픽셀을 255로 설정
- 팽창 연산은 객체 외곽을 확대시키는 연산 ⑦ 객체 크기는 감소 & 배경은 확대



# 모폴로지 (1): 침식과 팽창

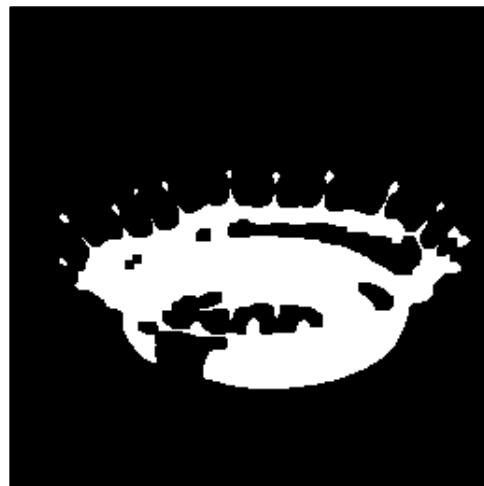
- 실제 영상의 침식 연산 결과
  - 객체 영역(흰색)이 점점 줄어듦
  - 작은 크기의 객체(잡음) 제거 효과



입력 영상



침식 1회



침식 2회



침식 3회

# 모폴로지 (1): 침식과 팽창

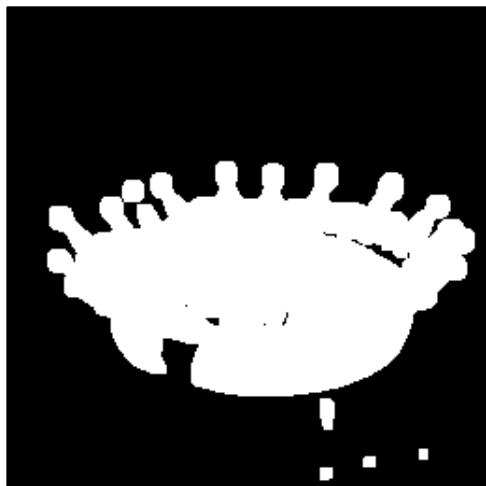
- 실제 영상의 팽창 연산 결과
  - 객체 영역(흰색)이 점점 불어남
  - 객체 내부의 홀(구멍)이 채워짐



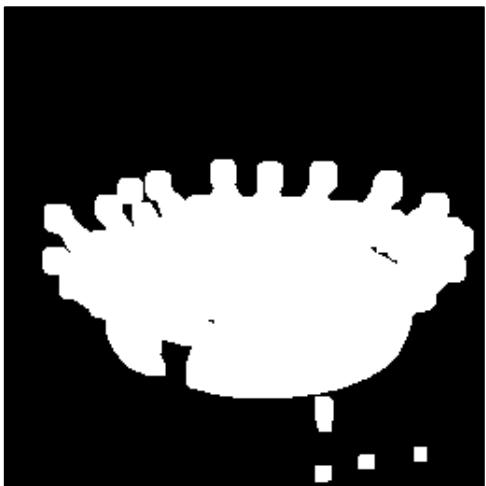
입력 영상



팽창 1회



팽창 2회



팽창 3회

# 모폴로지 (1): 침식과 팽창

## ■ 모폴로지 침식 연산

```
cv2.erode(src, kernel, dst=None, anchor=None, iterations=None,  
          borderType=None, borderValue=None) -> dst
```

- src: 입력 영상.
- kernel: 구조 요소. getStructuringElement() 함수에 의해 생성 가능.  
만약 **None**을 지정하면 3x3 사각형 구성 요소를 사용.
- dst: 출력 영상. src와 동일한 크기와 타입
- anchor: 고정점 위치. 기본값 (-1, -1)을 사용하면 중앙점을 사용.
- iterations: 반복 횟수. 기본값은 1.
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_CONSTANT.
- borderValue: cv2.BORDER\_CONSTANT인 경우, 확장된 가장자리 픽셀을 채울 값.

# 모폴로지 (1): 침식과 팽창

## ■ 모폴로지 팽창 연산

```
cv2.dilate(src, kernel, dst=None, anchor=None, iterations=None,  
           borderType=None, borderValue=None) -> dst
```

- src: 입력 영상.
- kernel: 구조 요소. getStructuringElement() 함수에 의해 생성 가능.  
만약 **None**을 지정하면 3x3 사각형 구성 요소를 사용.
- dst: 출력 영상. src와 동일한 크기와 타입
- anchor: 고정점 위치. 기본값 (-1, -1)을 사용하면 중앙점을 사용.
- iterations: 반복 횟수. 기본값은 1.
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_CONSTANT.
- borderValue: cv2.BORDER\_CONSTANT인 경우, 확장된 가장자리 픽셀을 채울 값.

# 모폴로지 (1): 침식과 팽창

## ■ 모폴로지 구조 요소 (커널) 생성

```
cv2.getStructuringElement(shape, ksize, anchor=None) -> retval
```

- shape: 구조 요소 모양을 나타내는 플래그

cv2.MORPH_RECT	사각형 모양
cv2.MORPH_CROSS	십자가 모양
cv2.MORPH_ELLIPSE	사각형에 내접하는 타원

- ksize: 구조 요소 크기. (width, height) 튜플.
- anchor: MORPH\_CROSS 모양의 구조 요소에서 고정점 좌표.  
(-1, -1)을 지정하면 구조 요소의 중앙을 고정점으로 사용.
- retval: 0과 1로 구성된 cv2.CV\_8UC1 타입 행렬. [numpy.ndarray](#).  
(1의 위치가 구조 요소 모양을 결정.)

# 모폴로지 (1): 침식과 팽창

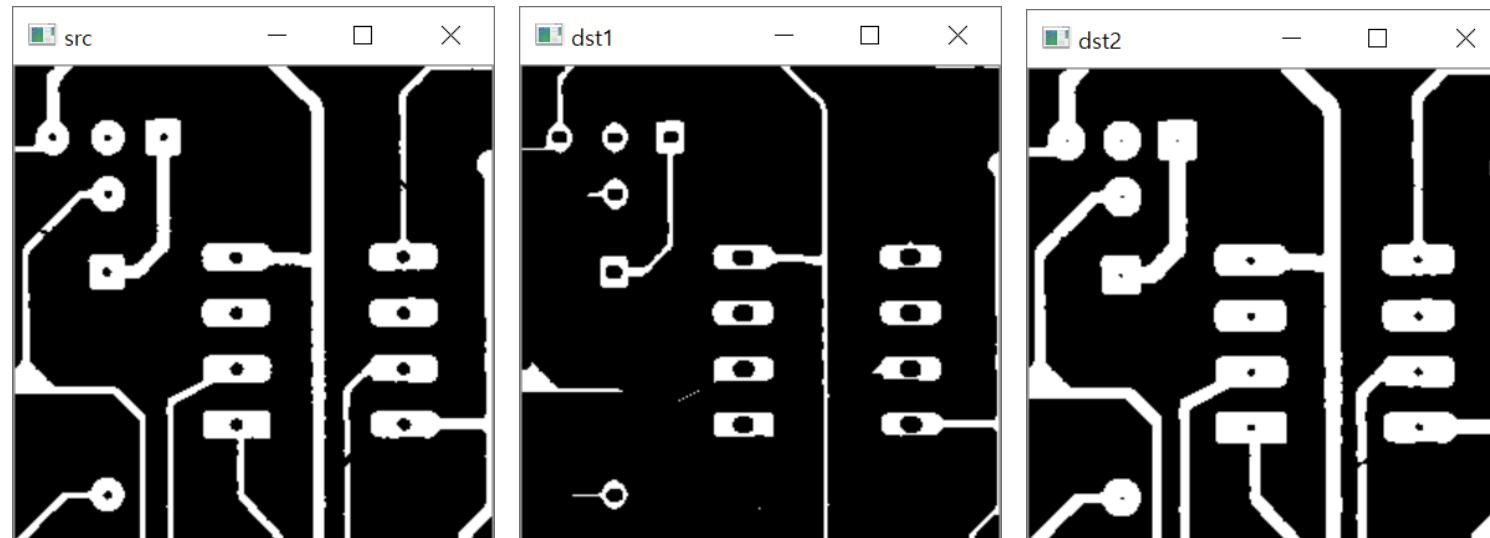
실습: morphology.py

## ■ 이진 영상의 침식과 팽창 예제

```
src = cv2.imread('circuit.bmp', cv2.IMREAD_GRAYSCALE)

se = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 3))
dst1 = cv2.erode(src, se)

dst2 = cv2.dilate(src, None)
```



## 7. 이진 영상처리

5) 모폴로지 (2): 열기와 닫기

## 모폴로지 (2): 열기와 닫기

- 이진 영상의 열기(opening) 연산

열기 = 침식 ⑦ 팽창

Opening = Erosion ⑦ Dilation

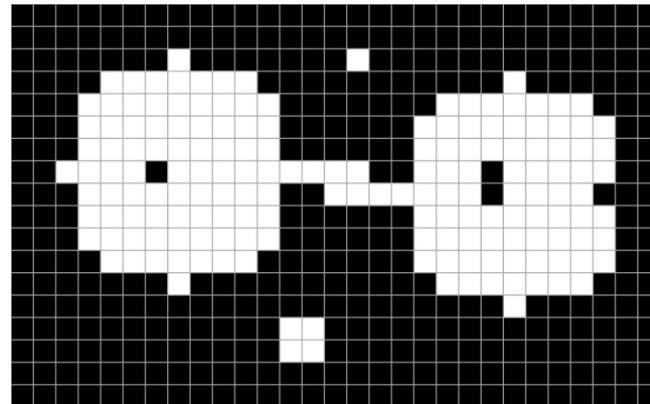
- 이진 영상의 닫기(closing) 연산

닫기 = 팽창 ⑦ 침식

Closing = Dilation ⑦ Erosion

# 모폴로지 (2): 열기와 닫기

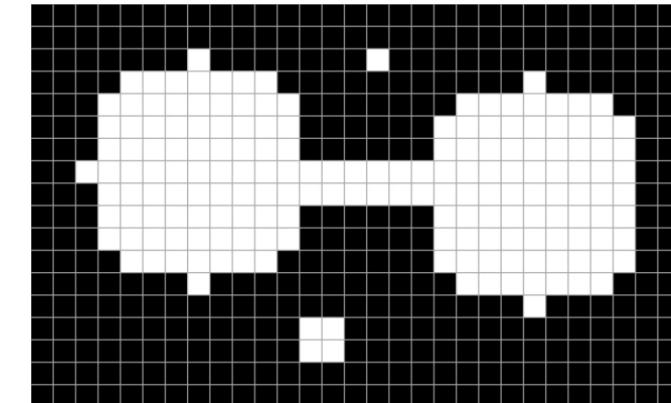
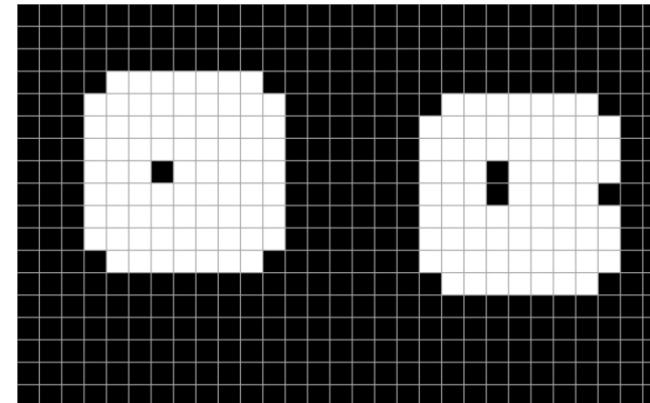
- 열기와 닫기 연산 효과 ( $3 \times 3$  구조 요소 사용)



열기

닫기

작은 돌기, 작은  
객체가 사라지고,  
얇은 연결선이  
끊어짐



작은 홈, 작은  
홀이 사라지고,  
얇은 연결선이  
두꺼워짐

# 모폴로지 (2): 열기와 닫기

## ■ 범용 모폴로지 연산 함수

```
cv2.morphologyEx(src, op, kernel, dst=None, anchor=None, iterations=None,  
borderType=None, borderValue=None) -> dst
```

- src: 입력 영상.
- op: 모폴로지 연산 플래그.

cv2.MORPH_ERODE	침식
cv2.MORPH_DILATE	팽창
cv2.MORPH_OPEN	열기
cv2.MORPH_CLOSE	닫기
cv2.MORPH_GRADIENT	모폴로지 그래디언트 = 팽창 - 침식

- kernel: 커널 출
- dst: 력 영상

# 모폴로지 (2): 열기와 닫기

실습: ricecount.py

## ■ 열기 연산을 이용한 잡음 제거 예제

```
src = cv2.imread('rice.png', cv2.IMREAD_GRAYSCALE)

dst1 = np.zeros(src.shape, np.uint8)

# src 영상에 지역 이진화 수행 (local_th.py 참고)

cnt1, _ = cv2.connectedComponents(dst1)
print('cnt1:', cnt1)

dst2 = cv2.morphologyEx(dst1, cv2.MORPH_OPEN, None)
#dst2 = cv2.erode(dst1, None)
#dst2 = cv2.dilate(dst2, None)

cnt2, _ = cv2.connectedComponents(dst2)
print('cnt2:', cnt2)
```

## 모폴로지 (2): 열기와 닫기

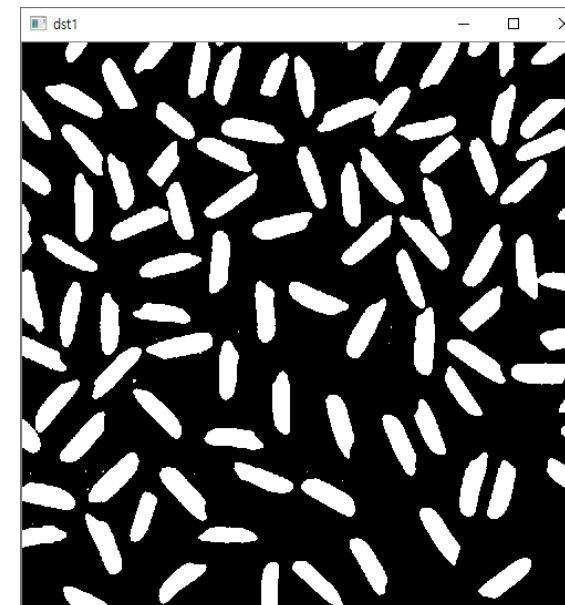
- 열기 연산을 이용한 잡음 제거 예제 실행 결과

## 지역 이진화



cnt1: 113

열기



cnt2: 99

## 7. 이진 영상처리

6) 레이블링

# 객체 단위 분석

## ■ 객체 단위 분석

- (흰색) 객체를 분할하여 특징을 분석
- 객체 위치 및 크기 정보, ROI 추출, 모양 분석 등



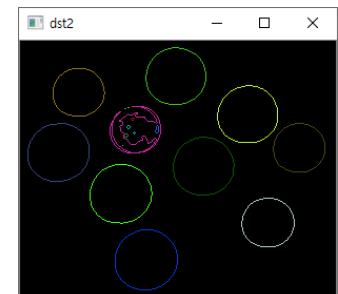
## ■ 레이블링(Connected Component Labeling)

- 서로 연결되어 있는 객체 픽셀에 고유한 번호를 지정 (레이블맵)
- 영역 기반 모양 분석
- 레이블맵, 바운딩 박스, 픽셀 개수, 무게 중심 좌표를 반환



## ■ 외곽선 검출(Contour Tracing)

- 각 객체의 외곽선 좌표를 모두 검출
- 외곽선 기반 모양 분석
- 다양한 외곽선 처리 함수에서 활용 가능 (근사화, 컨벡스헬 등)



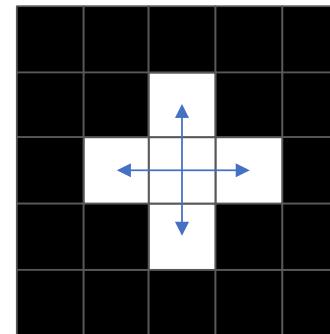
# 레이블링

## ■ 레이블링(Labeling)이란?

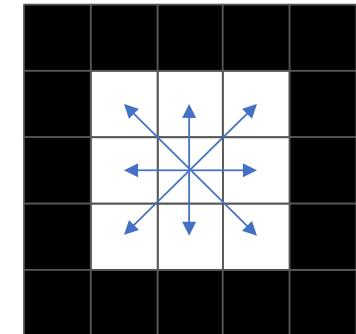
- 동일 객체에 속한 모든 픽셀에 고유한 번호를 매기는 작업
- 일반적으로 이진 영상에서 수행
- OpenCV에는 3.x 버전부터 최신 논문 기반의 레이블링 알고리즘 함수를 제공
- Connected component labeling

## ■ 픽셀의 연결 관계

- 4-이웃 연결관계(4-neighbor connectivity)
- 8-이웃 연결관계(8-neighbor connectivity)



4-이웃 연결 관계

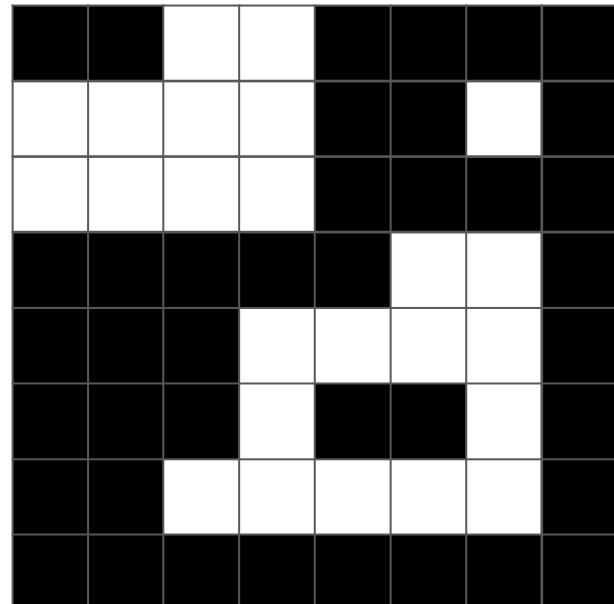


8-이웃 연결 관계

# 레이블링

## ■ 레이블링 알고리즘의 입력과 출력

입력



이진 영상  
(zero: 배경, non-zero: 객체)

출력

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

레이블 맵 (label map)  
(2차원 정수 행렬)

# 레이블링

## ■ 레이블링 함수

```
cv2.connectedComponents(image, labels=None, connectivity=None,  
ltype=None) -> retval, labels
```

- image: 8비트 1채널 영상
- labels: 레이블 맵 행렬. 입력 영상과 같은 크기. [numpy.ndarray](#).
- connectivity: 4 또는 8. 기본값은 8.
- ltype: labels 타입. cv2.CV\_32S 또는 cv2.CV\_16S. 기본값은 cv2.CV\_32S.
- retval: 객체 개수. N을 반환하면 [0, N-1]의 레이블이 존재 하며,  
0은 배경을 의미. (실제 흰색 객체 개수는 N-1개)

# 레이블링

## ■ 객체 정보를 함께 반환하는 레이블링 함수

```
cv2.connectedComponentsWithStats(image, labels=None, stats=None,  
                                 centroids=None, connectivity=None, ltype=None)  
    -> retval, labels, stats, centroids
```

- image: 8비트 1채널 영상
- labels: 레이블 맵 행렬. 입력 영상과 같은 크기. `numpy.ndarray`.
- stats: 각 객체의 바운딩 박스, 픽셀 개수 정보를 담은 행렬.  
`numpy.ndarray. shape=(N, 5), dtype=numpy.int32`.
- centroids: 각 객체의 무게 중심 위치 정보를 담은 행렬  
`numpy.ndarray. shape=(N, 2), dtype=numpy.float64`.
- ltype: labels 행렬 타입. `cv2.CV_32S` 또는 `cv2.CV_16S`. 기본값은 `cv2.CV_32S`.

# 레이블링

실습: labeling.py

- cv2.connectedComponentsWithStats() 함수 수행 결과의 예

```
retval, labels, stats, centroids = cv2.connectedComponentsWithStats(src)
```

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

retval = 4

stats				
0	0	8	8	38
0	0	4	3	10
6	1	1	1	1
2	3	5	4	14

← 배경  
← 1번 객체  
← 2번 객체  
← 3번 객체

← 면적

← 바운딩 박스 정보  
(x, y, width, height)

centroids	
3.615	3.692
1.7	1.2
6	1
4.285	4.785

← 무게 중심의 y 좌표  
← 무게 중심의 x 좌표

# 레이블링

실습: keyboard.py

## ■ 키보드 영상에서 문자 영역 분할 예제

```
src = cv2.imread('keyboard.bmp', cv2.IMREAD_GRAYSCALE)

_, src_bin = cv2.threshold(src, 0, 255, cv2.THRESH_OTSU)
cnt, labels, stats, centroids = cv2.connectedComponentsWithStats(src_bin)

dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)

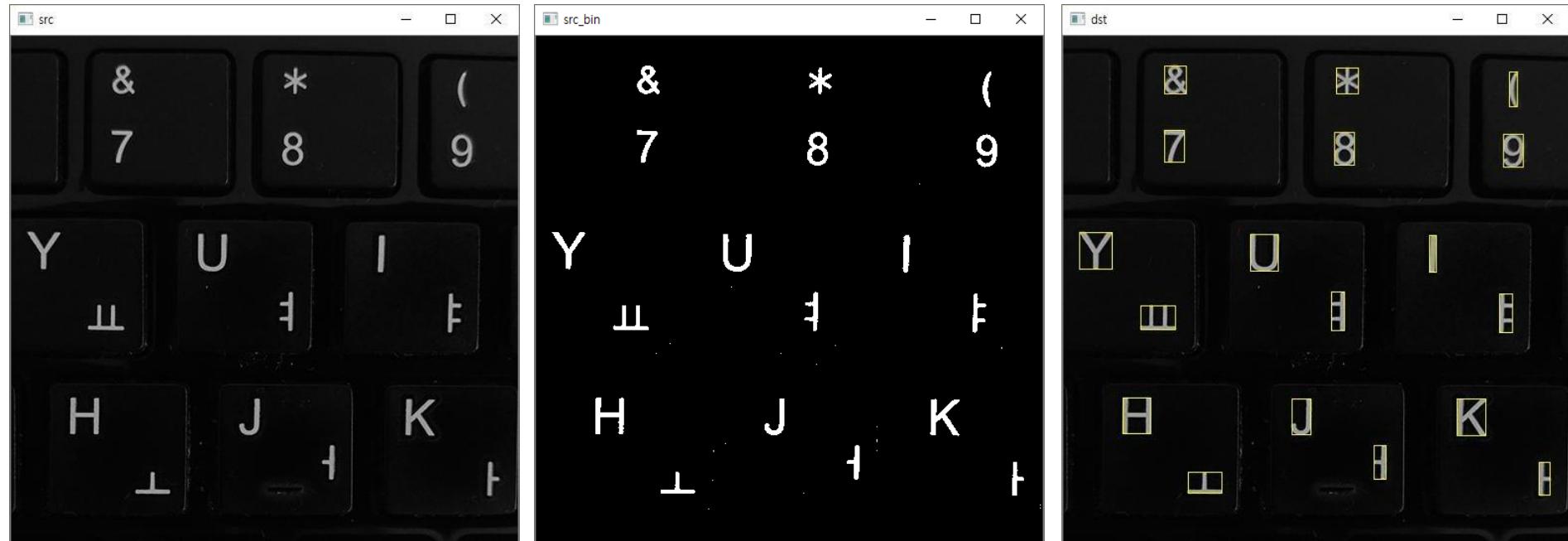
for i in range(1, cnt):
    (x, y, w, h, area) = stats[i]

    if area < 20:
        continue

    cv2.rectangle(dst, (x, y, w, h), (0, 255, 255))
```

# 레이블링

- 키보드 영상에서 문자 영역 분할 예제 실행 결과



# 8. 영상 분할과 객체 검출

1) 그랩컷

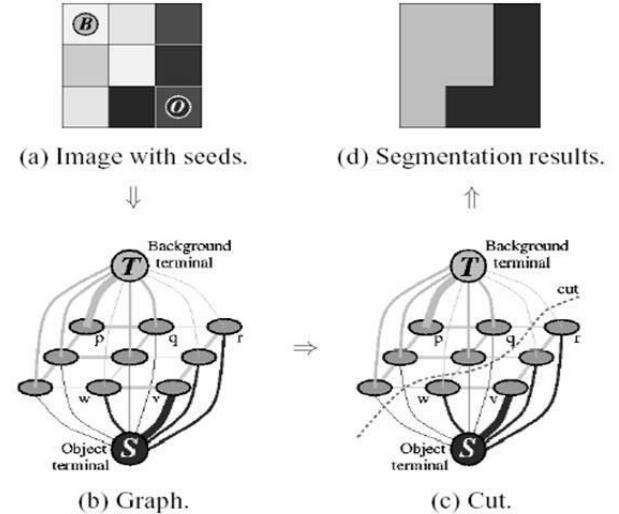
# 그랩컷

## ■ 그랩컷(GrabCut)이란?

- 그래프 컷(graph cut) 기반 영역 분할 알고리즘
- 영상의 픽셀을 그래프 정점으로 간주하고, 픽셀들을 두 개의 그룹으로 나누는 최적의 컷(Max Flow Minimum Cut)을 찾는 방식

## ■ 그랩컷 영상 분할 동작 방식

- 사각형 지정 자동 분할
- 사용자가 지정한 전경/배경 정보를 활용하여 영상분할



C. Rother, V. Kolmogorov, and A. Blake, "GrabCut: Interactive foreground extraction using iterated graph cuts," ACM Trans. Graph., vol. 23, pp. 309–314, 2004. /  
<https://grabcut.weebly.com/background--algorithm.html> / <https://www.cs.ru.ac.za/research/g02m1682/>

# 그랩컷

## ■ 그랩컷 함수

```
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, iterCount, mode=None)
    -> mask, bgdModel, fgdModel
```

- img: 입력 영상. 8비트3채널.
- mask: 입출력 마스크. cv2.GC\_BGD(0), cv2.GC\_FGD(1), cv2.GC\_PR\_BGD(2),  
cv2.GC\_PR\_FGD(3) 네 개의 값으로 구성됨.  
cv2.GC\_INIT\_WITH\_RECT 모드로 초기화.
- rect: ROI 영역. cv2.GC\_INIT\_WITH\_RECT 모드에서만 사용됨
- bgdModel: 임시 배경 모델 행렬. 같은 영상 처리 시에는 변경 금지.
- fgdModel: 임시 전경 모델 행렬. 같은 영상 처리 시에는 변경 금지.
- iterCount: 결과 생성을 위한 반복 횟수.
- mode: cv2.GC\_로 시작하는 모드 상수. 보통 cv2.GC\_INIT\_WITH\_RECT 모드로  
초기화하고, cv2.GC\_INIT\_WITH\_MASK 모드로 업데이트함.

# 그랩컷

실습: grabcut1.py

## ■ 그랩컷 영상 분할 예제

```
src = cv2.imread('nemo.jpg')

rc = cv2.selectROI(src)
mask = np.zeros(src.shape[:2], np.uint8)

cv2.grabCut(src, mask, rc, None, None, 5, cv2.GC_INIT_WITH_RECT)
```

# 0: cv2.GC\_BGD, 2: cv2.GC\_PR\_BGD

```
mask2 = np.where((mask == 0) | (mask == 2), 0, 1).astype('uint8')
```

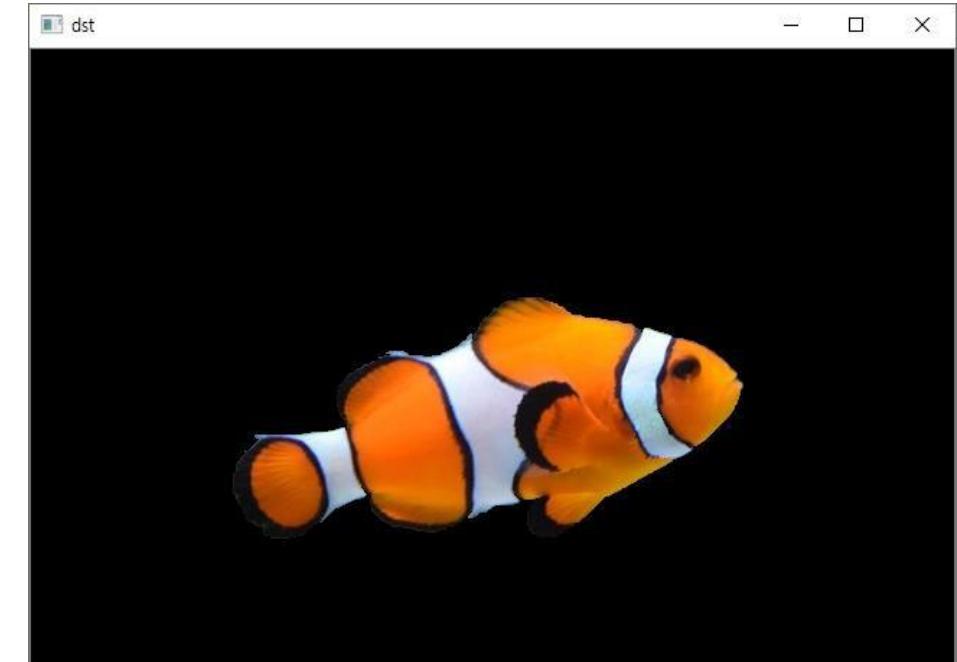
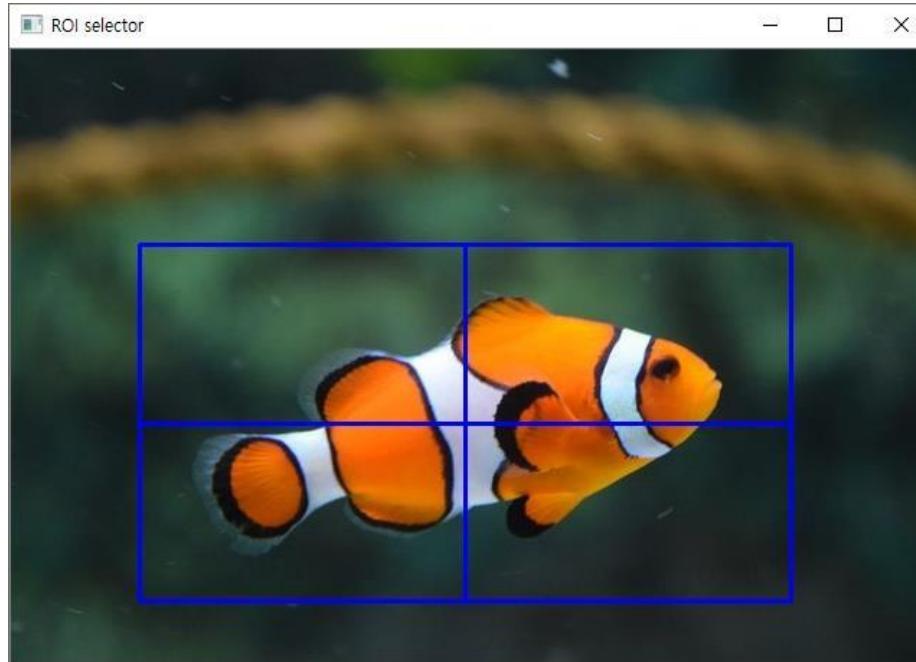
```
dst = src * mask2[:, :, np.newaxis]
```



mask 행렬에서 값이 0 또는 2인 원소는  
0으로, 그렇지 않은 원소는 1로 설정

# 그랩 컷

- 그랩컷 영상 분할 예제 실행 결과

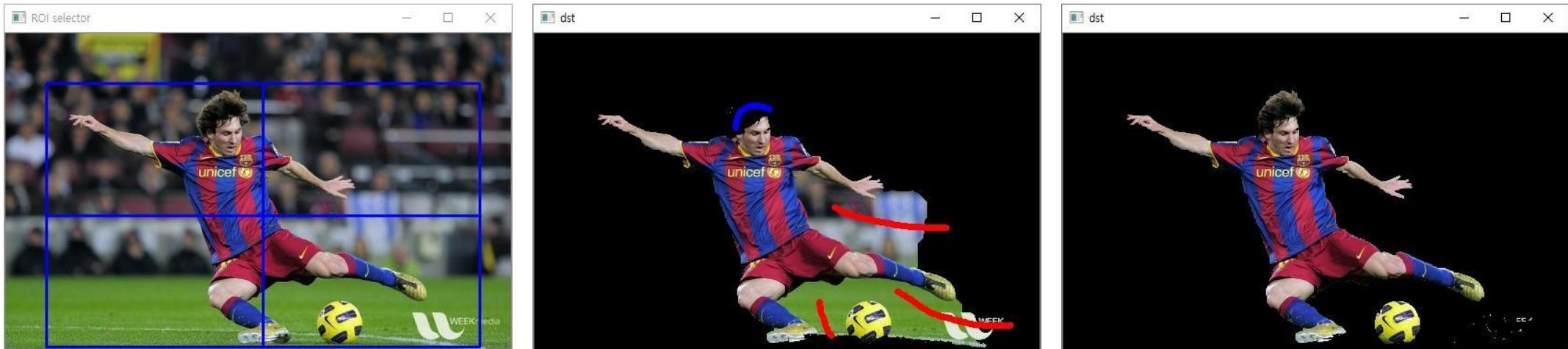


# 그랩컷

실습: grabcut2.py

## ■ 마우스를 활용한 그랩컷 영상 분할 예제

- 소스 코드는 예제 파일 참고: [grabcut2.py](#)
- 초기 영역은 ROI selector 창에서 사각형 지정
- 초기 분할 결과 dst 창에서 전경은 마우스 왼쪽 버튼 드래그(파란색), 배경은 마우스 오른쪽 버튼 드래그(빨간색) ⑦ ENTER 키 입력 시 영상 재분할



[https://docs.opencv.org/trunk/d8/d83/tutorial\\_py\\_grabcut.html](https://docs.opencv.org/trunk/d8/d83/tutorial_py_grabcut.html)

## 8. 영상 분할과 객체 검출

2) HOG 보행자 검출

# HOG 보행자 검출

## ■ HOG(Histogram of Oriented Gradients)란?

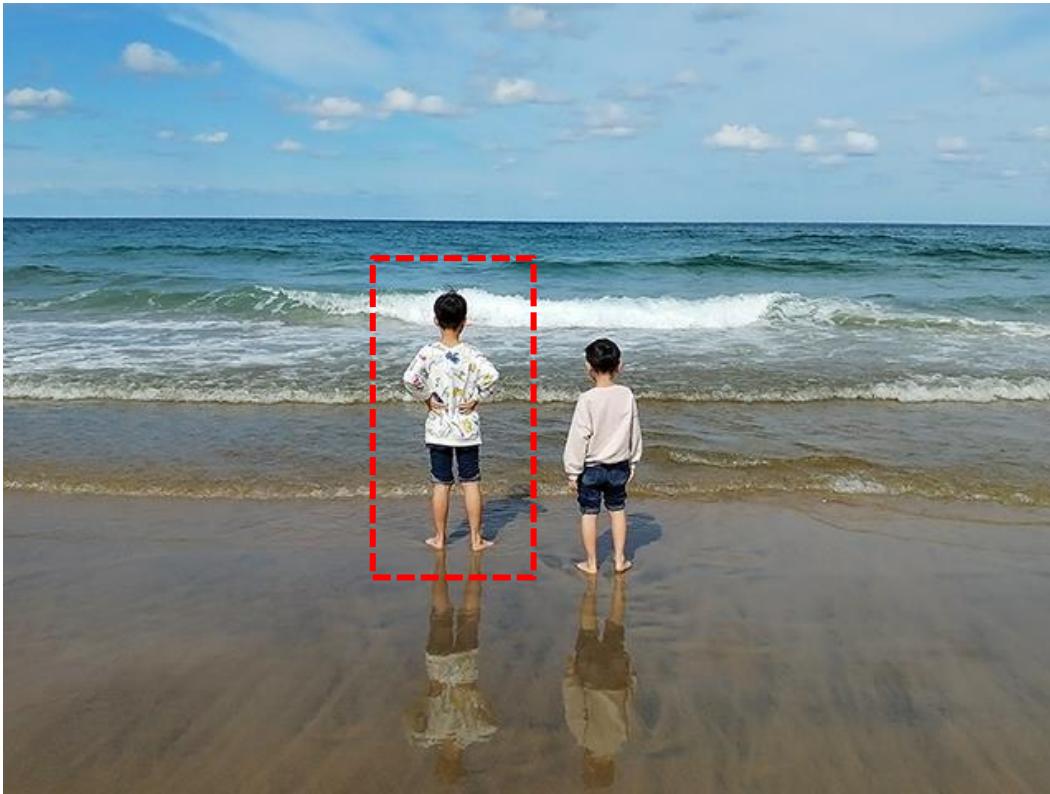
- 영상의 지역적 그래디언트 방향 정보를 특징 벡터로 사용
- 2005년 CVPR 학회에서 보행자 검출 방법으로 소개되어 널리 사용되기 시작함
- 이후 다양한 객체 인식에서 활용됨



N. Dalal and B. Triggs, "Histogram of oriented gradients for human detection," *Computer Vision and Pattern Recognition 2005*, pp. 886-893.

# HOG 보행자 검출

## ■ HOG 알고리즘



부분 영상  
추출



크기 정규화  
64x128

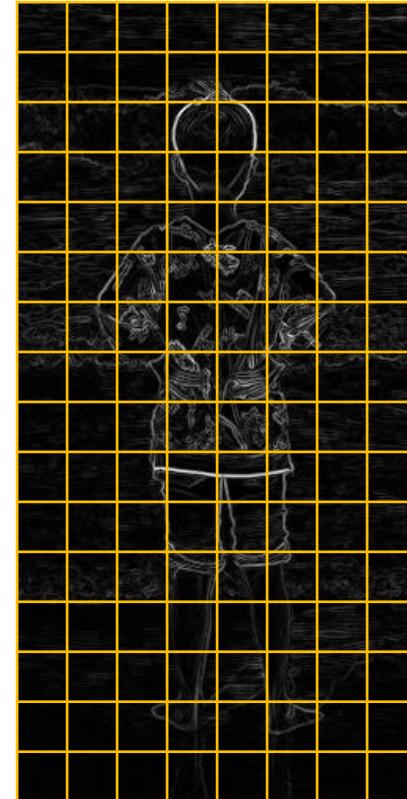
# HOG 보행자 검출

## ■ HOG 알고리즘



64x128 영상

그레이드  
언트  
계산



8x8 크기의  
셀(cell) 분할

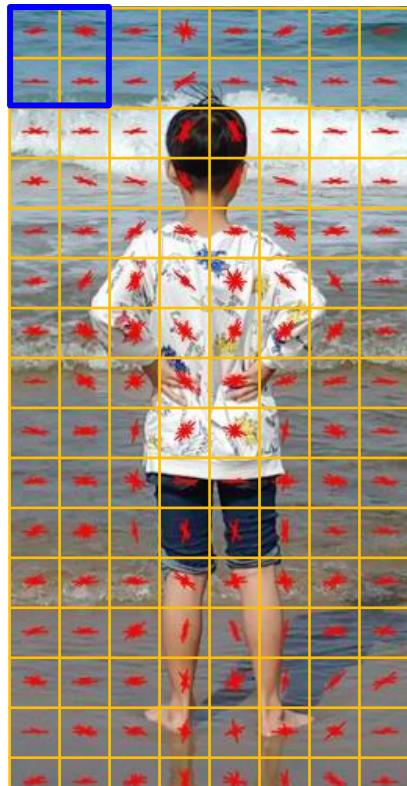
각 셀마다  
방향과 크기 성  
분을 이용하여  
방향 히스토그램  
계산



방향 히스토그램의  
빈 개수 = 9

# HOG 보행자 검출

## ■ HOG 알고리즘



### [블록 히스토그램 구하기]

- 8x8 셀 4개를 하나의 블록을 지정
- ⑦ 즉, 블록 하나의 크기는  $16 \times 16$
- ⑦ 8픽셀 단위로 이동: `stride = 8`
- ⑦ 각 블록의 히스토그램 빈(bin) 개수는  $4 \times 9 = 36$ 개

### [특징 벡터의 차원]

하나의 부분 영상 패치에서의 특징 벡터 크기

$$⑦ 7 \times 15 \times 36 = 3780$$

# HOG 보행자 검출

- HOG 기술자 객체 생성 및 보행자 검출을 위해 학습된 분류기 계수 불러오기

```
cv2.HOGDescriptor() -> <object>
```

```
cv2.HOGDescriptor_getDefaultPeopleDetector() -> retval
```

- retval : 미리 훈련된 특징 벡터. `numpy.ndarray`. shape=(3781, 1).  
`dtype=numpy.float32`.

- SVM 분류기 계수등록하기

```
cv2.HOGDescriptor.setSVMClassifier(svmdetector) -> None
```

- svmdetector: 선형 SVM 분류기를 위한 계수

# 캐스케이드 분류기: 얼굴 검출

## ■ HOG 멀티스케일 객체 검출 함수

```
cv2.HOGDescriptor.detectMultiScale(img, hitThreshold=None, winStride=None,  
padding=None, scale=None, finalThreshold=None,  
useMeanshiftGrouping=None) -> foundLocations, foundWeights
```

- img: 입력 영상. cv2.CV\_8UC1 또는 cv2.CV\_8UC3.
- hitThreshold: 특징 벡터와 SVM 분류 평면까지의 거리에 대한 임계값
- winStride: 셀 윈도우 이동 크기. (0, 0) 지정 시 셀 크기와 같게 설정.
- padding: 패딩 크기
- scale: 검색 윈도우 크기 확대 비율. 기본값은 1.05.
- finalThreshold: 검출 결정을 위한 임계값
- useMeanshiftGrouping: 겹쳐진 검색 윈도우를 합치는 방법 지정 플래그
- foundLocations: (출력) 검출된 사각형 영역 정보
- foundWeights: (출력) 검출된 사각형 영역에 대한 신뢰도

# HOG 사람 검출

실습: peopledetect.py

## ■ HOG 보행자 검출 예제

```
cap = cv2.VideoCapture('vtest.avi')

hog = cv2.HOGDescriptor()
hog.setSVMClassifier(cv2.HOGDescriptor_getDefaultPeopleDetector())

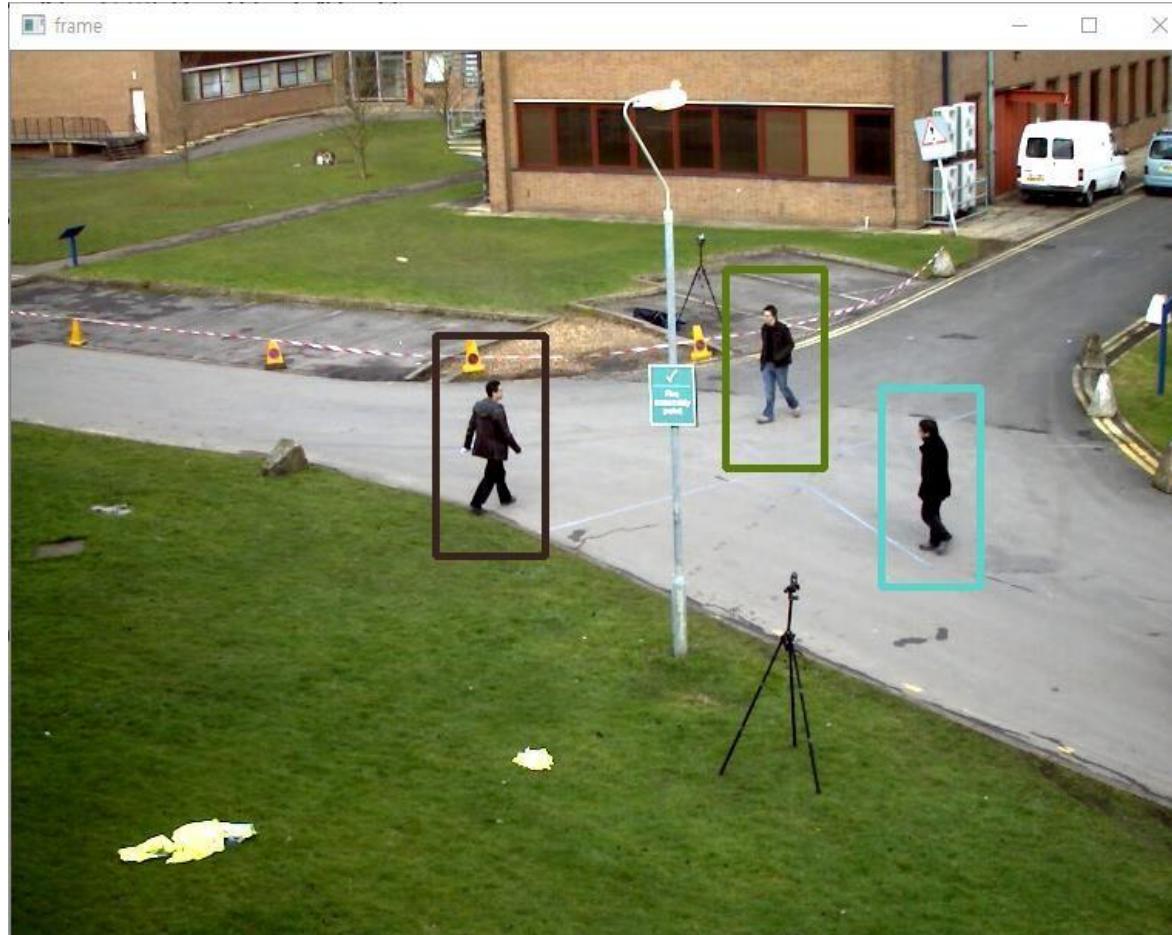
while True:
    ret, frame = cap.read()

    detected, _ = hog.detectMultiScale(frame)

    for (x, y, w, h) in detected:
        c = (random.randint(0, 255), random.randint(0, 255),
              random.randint(0, 255))
        cv2.rectangle(frame, (x, y), (x + w, y + h), c, 3)
```

# HOG 사람 검출

- HOG 보행자 검출 예제 실행 결과



# 9. 특징점 검출과 매칭

1) 코너 검출

# 코너 검출

## ■ 코너의 특징

- 평탄한 영역(flat) & 에지(edge) 영역은 고유한 위치를 찾기 어려움
- 코너(corner)는 변별력이 높은 편이며, 영상의 이동, 회전 변환에 강인함



# 코너 검출

## ■ 다양한 코너 검출 방법

코너 검출 방법	특징
해리스 (Harris)	<ul style="list-style-type: none"><li>영상 내부 작은 영역이 모든 방향에 대해 변화가 큰 경우 코너로 규정</li><li>코너 응답 함수 <math>R</math>을 반환 ⑦ <math>R(x,y)</math>가 충분히 크면 코너로 구분</li><li><code>cv2.cornerHarris()</code> 함수 사용</li></ul>
추적하기 좋은 특징 (Good Features to Track)	<ul style="list-style-type: none"><li>해리스 코너 검출 방법을 기반으로 향상된 방법</li><li>비최대 억제수행</li><li>코너 품질 함수를 정의 ⑦ 가장 값이 큰 순서대로 정렬하여 반환</li><li><code>cv2.goodFeaturesToTrack()</code> 함수 사용</li></ul>
FAST (Features from Accelerated Segment Test)	<ul style="list-style-type: none"><li>주변 16개 픽셀 값 크기를 분석</li><li>기준 픽셀(<math>p</math>)보다 충분히 밝거나(<math>&gt;p+t</math>) 또는 충분히 어두운(<math>&lt;p-t</math>) 픽셀이 <math>n</math>개 연속으로 나타나면 코너로 인식 (<math>n</math>은 보통 9)</li><li>해리스, GFTT 방법보다 매우 빠르게 동작</li><li><a href="https://www.edwardrosten.com/work/fast.html">https://www.edwardrosten.com/work/fast.html</a></li></ul>

# 코너 검출

## ■ 해리스 코너 응답 함수 계산

```
cv2.cornerHarris(src, blockSize, ksize, k, dst=None, borderType=None) -> dst
```

- src: 입력 단일채널 8비트 또는 실수형 영상
- blockSize: 코너 응답 함수 계산에서 고려할 이웃 픽셀 크기. 보통 2~5.
- ksize: (미분을 위한) 소벨 연산자를 위한 커널 크기. 보통 3. 해
- k: 리스 코너 검출 상수 (보통 0.04~0.06)
- dst: 해리스 코너 응답 계수. src와 같은 크기의 행렬([numpy.ndarray](#)).  
dtype=[numpy.float32](#).
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER\_DEFAULT.

# 코너 검출

## ■ 추적하기 좋은 특징 코너 검출

```
cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance,  
corners=None, mask=None, blockSize=None,  
useHarrisDetector=None, k=None) -> corners
```

- image: 8비트 또는 32비트 실수, 단일채널 영상
- maxCorners: 최대 코너 개수. maxCorners <=0 이면 무제한.
- qualityLevel: 코너점 결정을 위한 값. 보통 0.01 ~ 0.1.
- minDistance: 코너점 사이의 최소 거리
- corners: 검출된 코너점 좌표. `numpy.ndarray`. `shape=(N, 1, 2)`. `dtype=numpy.float32`.
- mask: 마스크 영상
- blockSize: 코너 검출을 위한 블록 크기. 기본값은 3.
- useHarrisDetector: 해리스 코너 방법 사용 여부. 기본값은 False.
- k: 해리스 코너 검출 시 사용할 k 값

# 코너 검출

## ▪ FAST 코너 검출

```
cv2.FastFeatureDetector_create(, threshold=None, nonmaxSuppression=None,  
                                type=None) -> retval
```

```
cv2.FastFeatureDetector.detect(image) -> keypoints
```

- threshold: 중심 픽셀 값과 주변 픽셀 값과의 차이 임계값. 기본값은 10.
- nonmaxSuppression: 비최대 억제 수행 여부. 기본값은 True.
- type: 코너 검출 방법. 기본값은 cv2.FAST\_FEATURE\_DETECTOR\_TYPE\_9\_16.
- retval: FastFeatureDetector 객체
- image: (입력) 그레이스케일 영상
- keypoints: (출력) 검출된 코너점 정보. cv2.KeyPoint 객체를 담은 리스트.  
cv2.KeyPoint의 pt 멤버를 이용하여 코너 좌표 추출.  
pt[0]은 x좌표, pt[1]은 y좌표.

# 코너 검출

실습: corners.py

## ■ GFTT와 FAST 코너 검출 예제

```
src = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)

corners = cv2.goodFeaturesToTrack(src, 400, 0.01, 10)

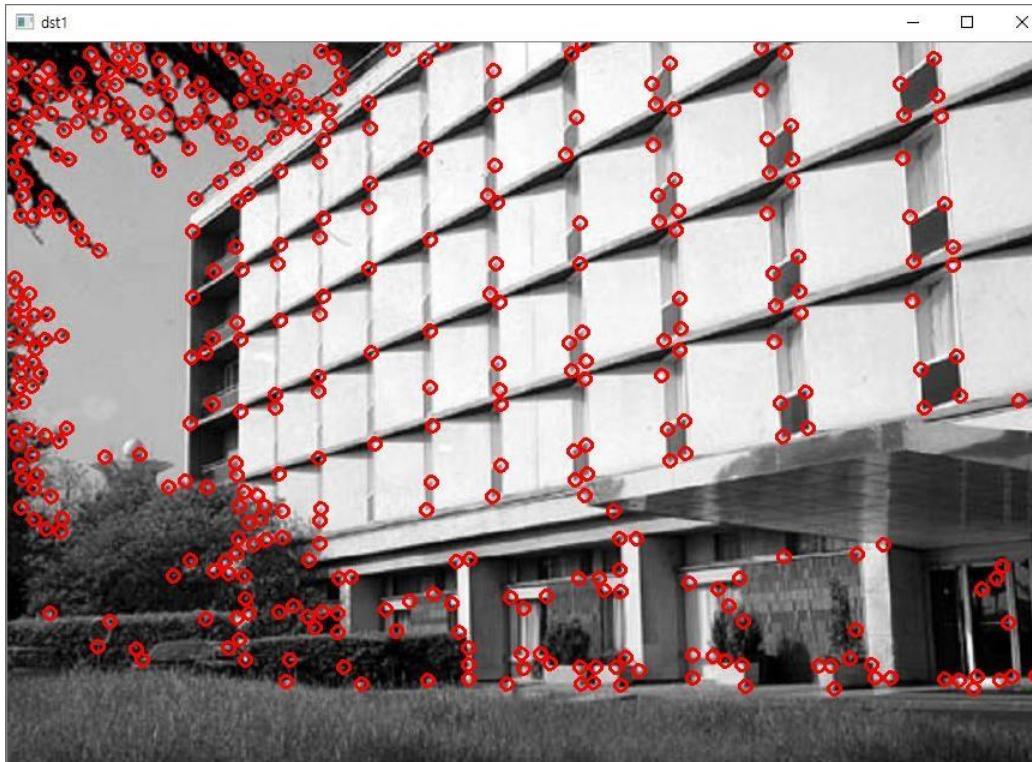
dst1 = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)
if corners is not None:
    for i in range(corners.shape[0]):
        pt = (int(corners[i, 0, 0]), int(corners[i, 0, 1]))
        cv2.circle(dst1, pt, 5, (0, 0, 255), 2)

fast = cv2.FastFeatureDetector_create(60)
keypoints = fast.detect(src)

dst2 = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)
for kp in keypoints:
    pt = (int(kp.pt[0]), int(kp.pt[1]))
    cv2.circle(dst2, pt, 5, (0, 0, 255), 2)
```

# 코너 검출

## ■ GFTT와 FAST 코너 검출 예제



GFTT: 18.665ms.

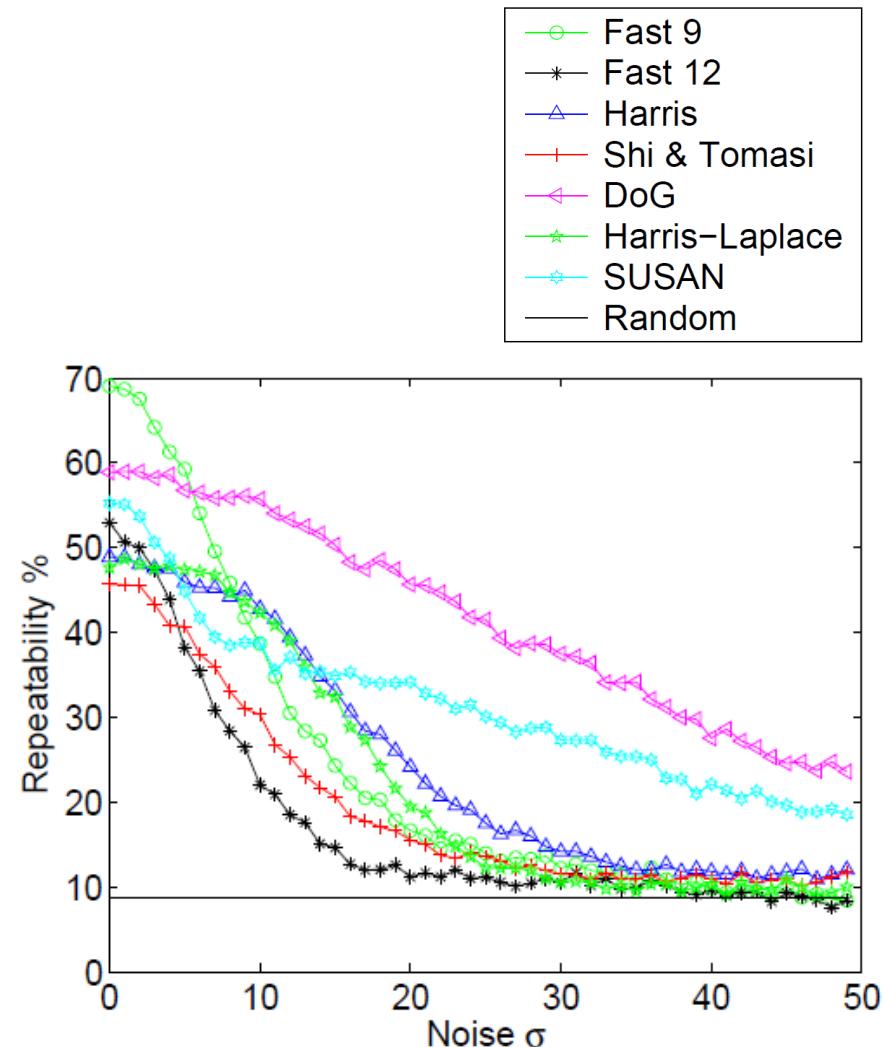
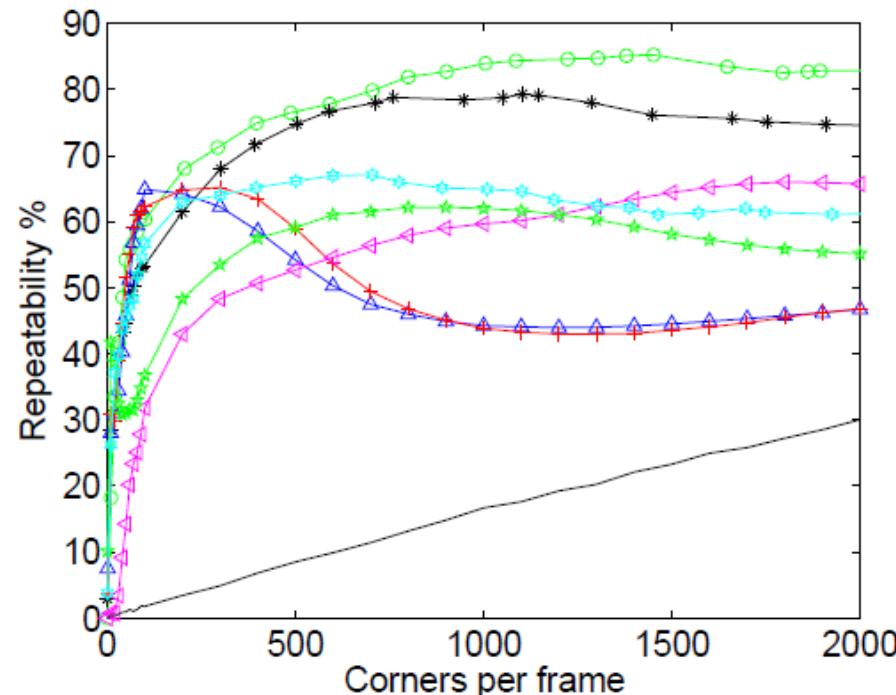


FAST: 0.6577ms.

# 코너 검출 방법 성능 비교

## ■ 코너 검출 반복성 비교

- FAST 방법의 반복 검출률이 대체로 높음
- 다만 FAST 방법은 노이즈에 민감함



<https://www.edwardrosten.com/work/fast.html>

# 9. 특징점 검출과 매칭

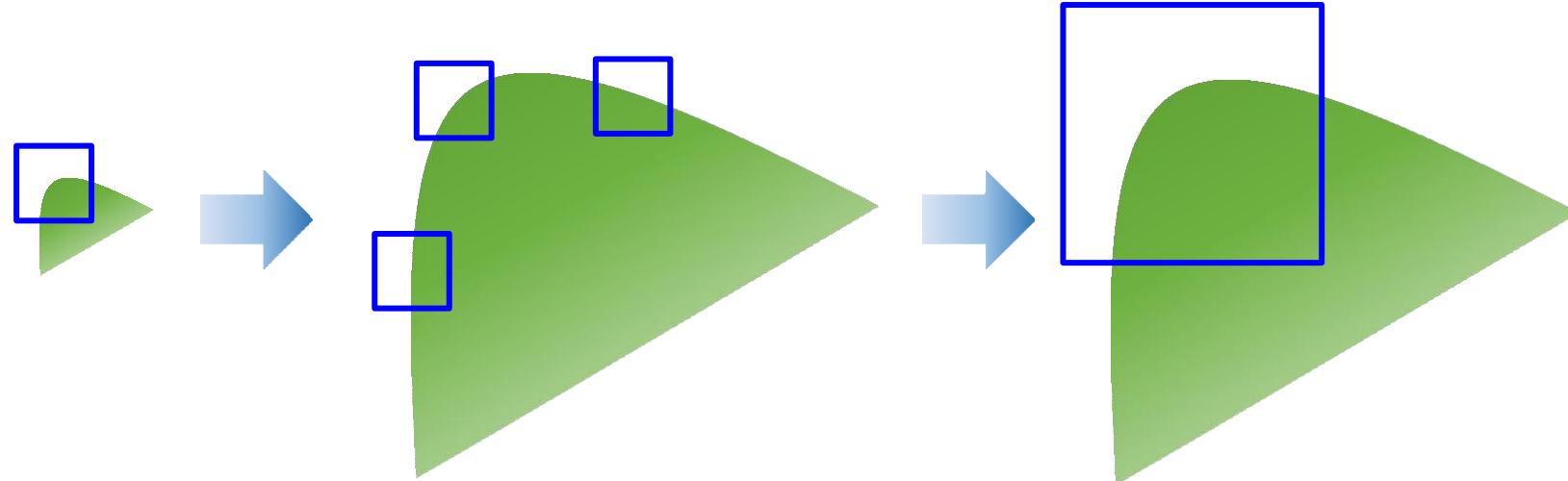
2) 특징점 검출

# 특징점 검출

## ■ Harris, GFTT, FAST 코너의 문제점

- 이동, 회전 변환에 강인
- 크기 변환에 취약

다양한 크기 관점에서  
특징 검출 필요!



- 특징점(feature point) ≈ 키포인트(keypoint) ≈ 관심점(interest point)
- 기술자(descriptor) ≈ 특징 벡터(feature vector)

# 특징점 검출

## ■ 크기 불변 특징점 검출 방법

- SIFT, KAZE, AKAZE, ORB 등 다양한 특징점 검출 방법에서 스케일 스페이스(scale-space), 이미지 피라미드(image pyramid)를 구성하여 크기 불변 특징점을 검출



Scale-Space

*"The maxima and minima of  $\sigma^2 \nabla G$  produce the most stable image features..."*

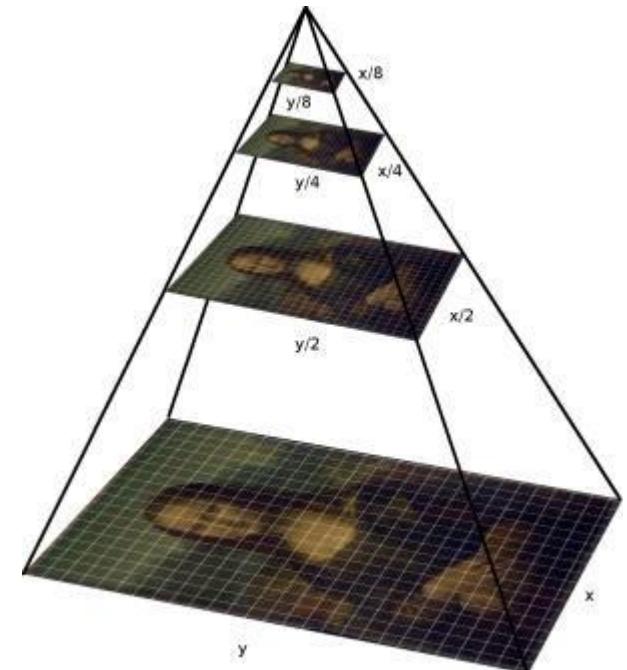
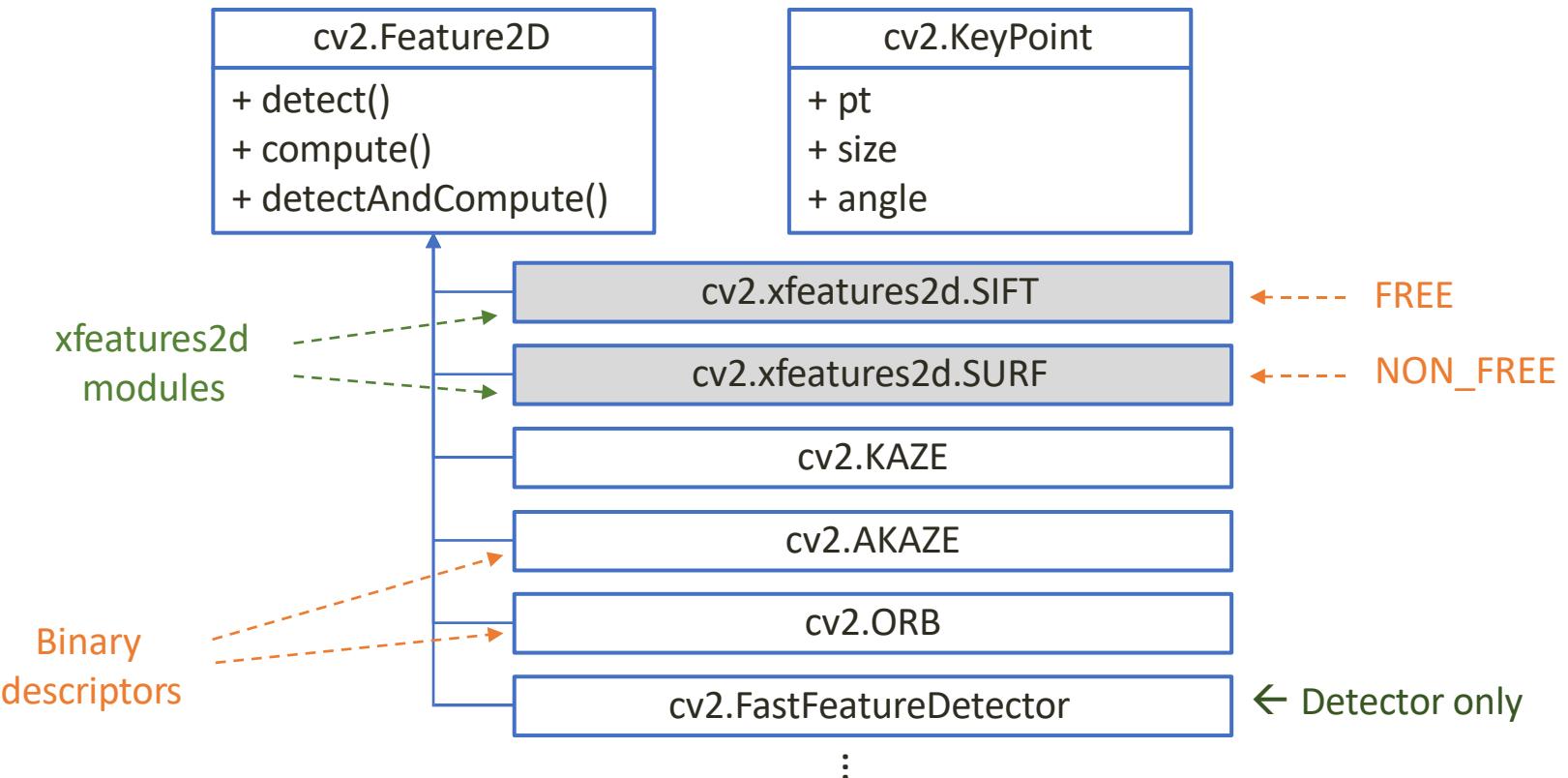


Image Pyramid

# 특징점 검출

- OpenCV 특징점 검출 클래스: Feature2D 클래스와 파생 클래스



[https://docs.opencv.org/master/d0/d13/classcv\\_1\\_1Feature2D.html](https://docs.opencv.org/master/d0/d13/classcv_1_1Feature2D.html)

# 특징점 검출

## ■ 특징점 검출 알고리즘 객체 생성

```
cv2.KAZE_create(, ...) -> retval  
cv2.AKAZE_create(, ...) -> retval  
cv2.ORB_create(, ...) -> retval  
cv2.xfeatures2d.SIFT_create(, ...) -> retval  
...
```

- retval: 각 특징점 검출 알고리즘 객체
- 참고사항
  - 각각의 알고리즘은 고유한 파라미터를 인자로 받을 수 있음
  - 대부분의 인자는 기본값을 가지고 있으므로 함수 인자 없이 호출 가능

# 특징점 검출

## ■ 특징점 검출 함수

```
cv2.Feature2D.detect(image, mask=None) -> keypoints
```

- image: 입력 영상
- mask: 마스크 영상
- keypoints: 검출된 특징점 정보. cv2.KeyPoint 객체의 리스트.

# 특징점 검출

## ■ 검출된 특징점 그리기 함수

```
cv2.drawKeypoints(image, keypoints, outImage, color=None, flags=None)  
    -> outImage
```

- image: 입력 영상
- keypoints: 검출된 특징점 정보. cv2.KeyPoint 객체의 리스트.
- outImage: 출력 영상
- color: 특징점 표현 색상.  
기본값은 (-1, -1, -1, -1)이며, 이 경우 임의의 색상으로 표현.
- flags: 특징점 표현 방법.

cv2.DRAW_MATCHES_FLAGS_DEFAULT	특징점 위치만을 표현하는 작은 크기의 원
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS	특징점의 크기와 방향을 반영한 원

# 특징점 검출

실습: keypoints.py

## ■ 특징점 검출 예제

```
# 영상 불러오기
src1 = cv2.imread('graf1.png', cv2.IMREAD_GRAYSCALE)
src2 = cv2.imread('graf3.png', cv2.IMREAD_GRAYSCALE)

# 두 영상에서 특징점 검출 & 출력 (KAZE, AKAZE, ORB 등)
feature = cv2.KAZE_create()
#feature = cv2.AKAZE_create()
#feature = cv2.ORB_create()

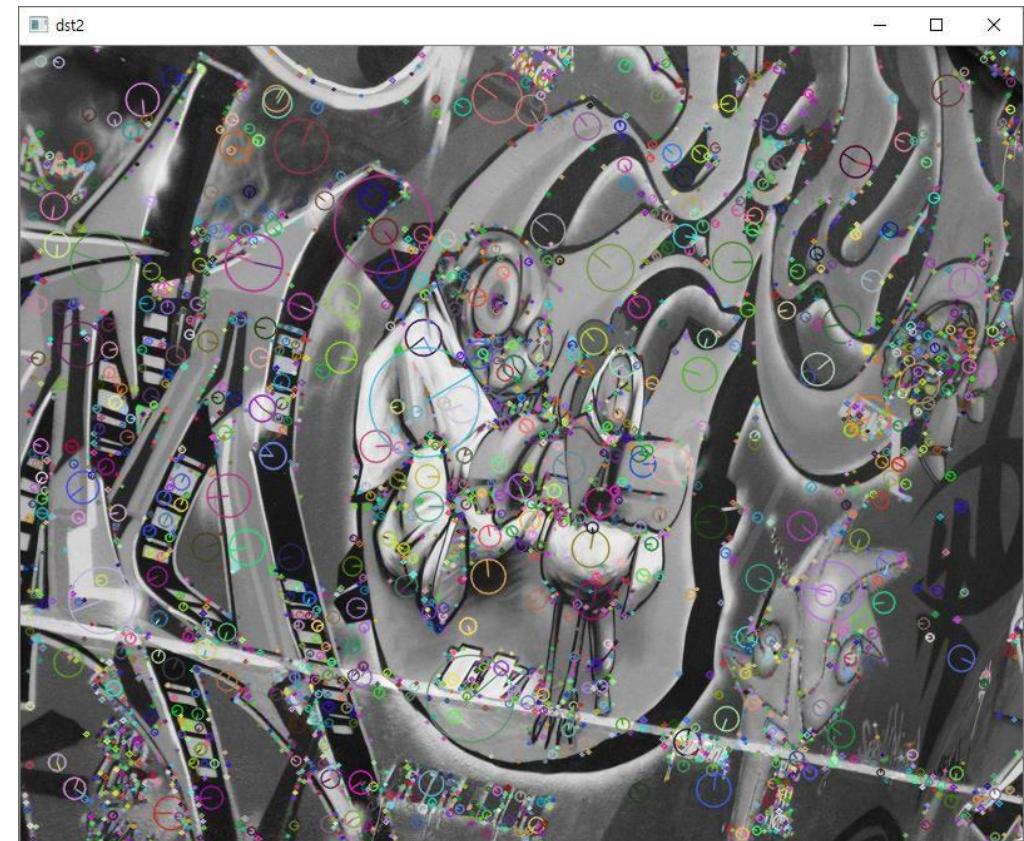
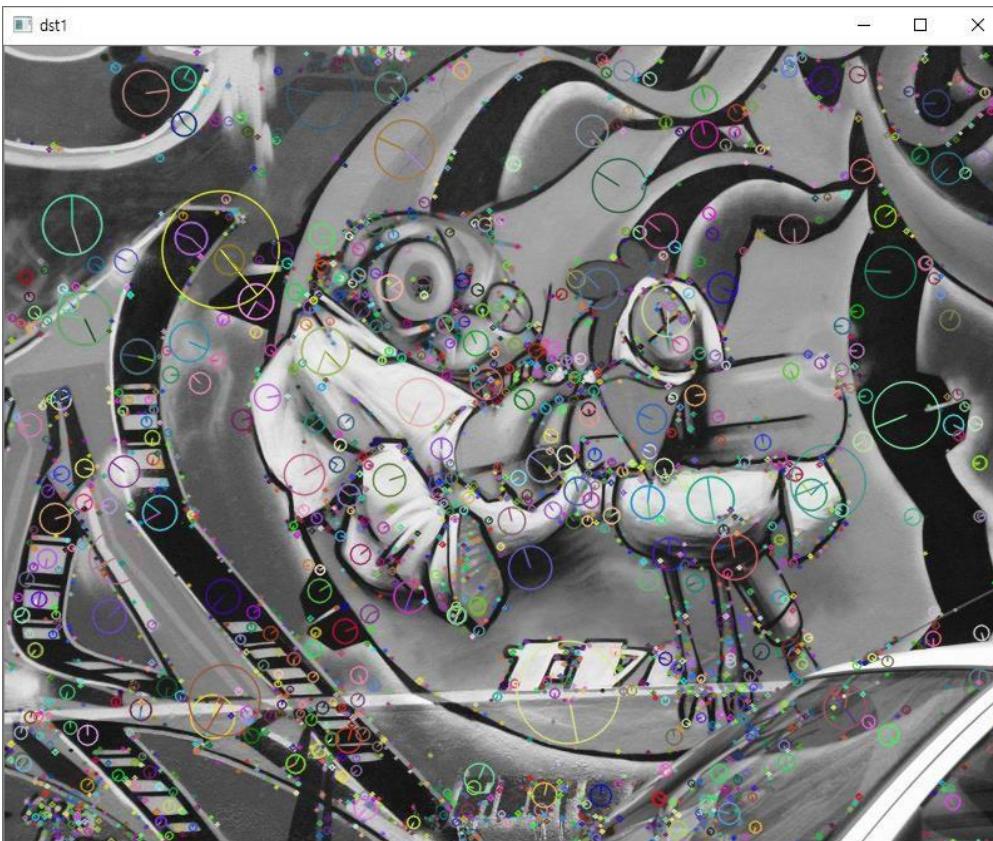
kp1 = feature.detect(src1)
kp2 = feature.detect(src2)

# 검출된 특징점 출력 영상 생성
dst1 = cv2.drawKeypoints(src1, kp1, None,
                         flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
dst2 = cv2.drawKeypoints(src2, kp2, None,
                         flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

# 특징점 검출

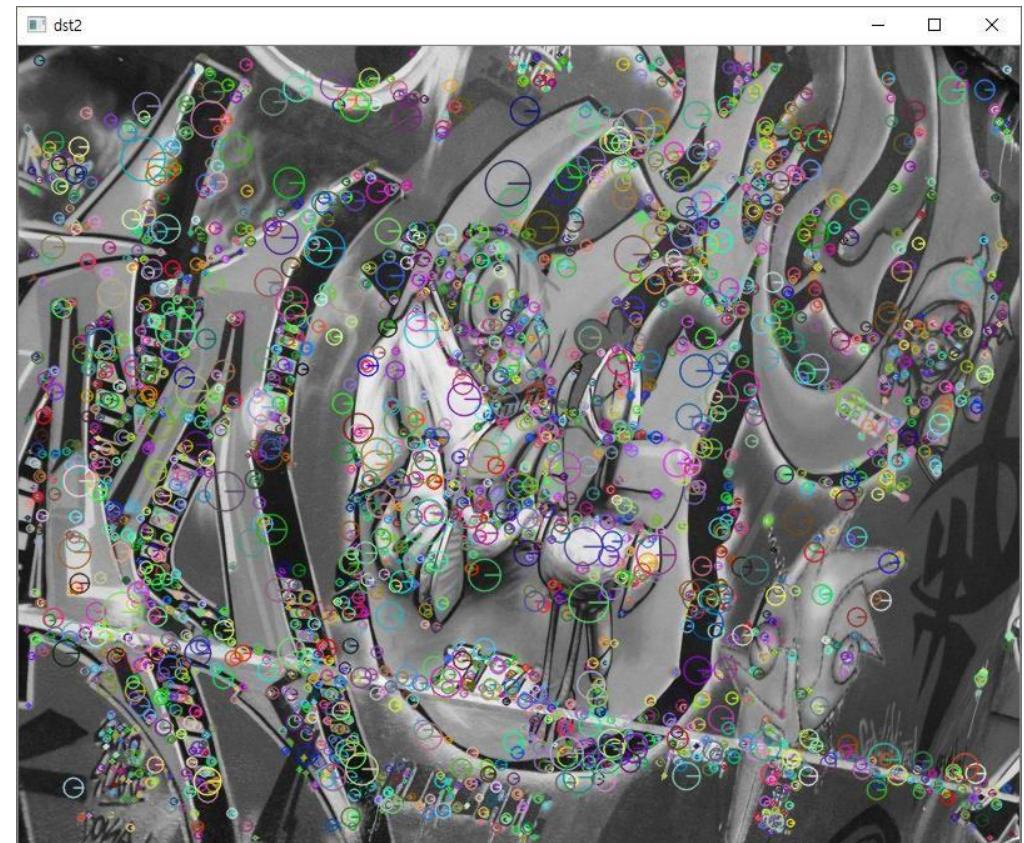
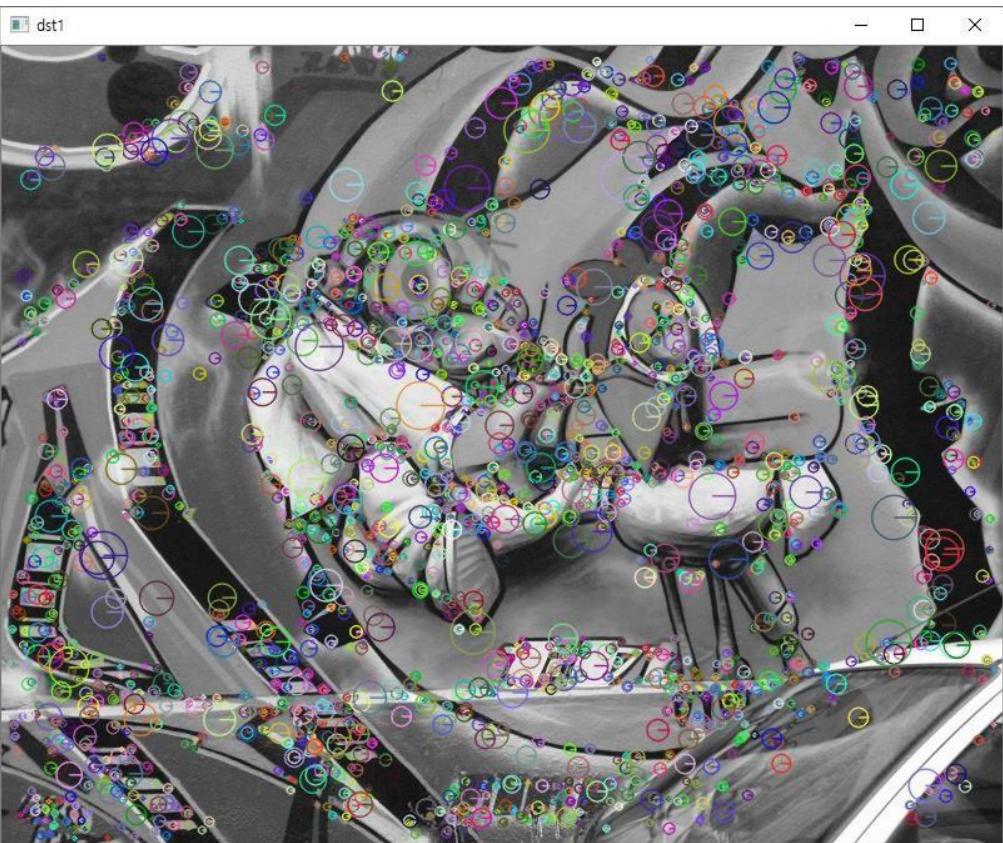
## ■ SIFT 특징점 검출 결과

- cv2.xfeatures2d.SIFT\_create() 함수 사용 (소스 코드 직접 빌드 필요)



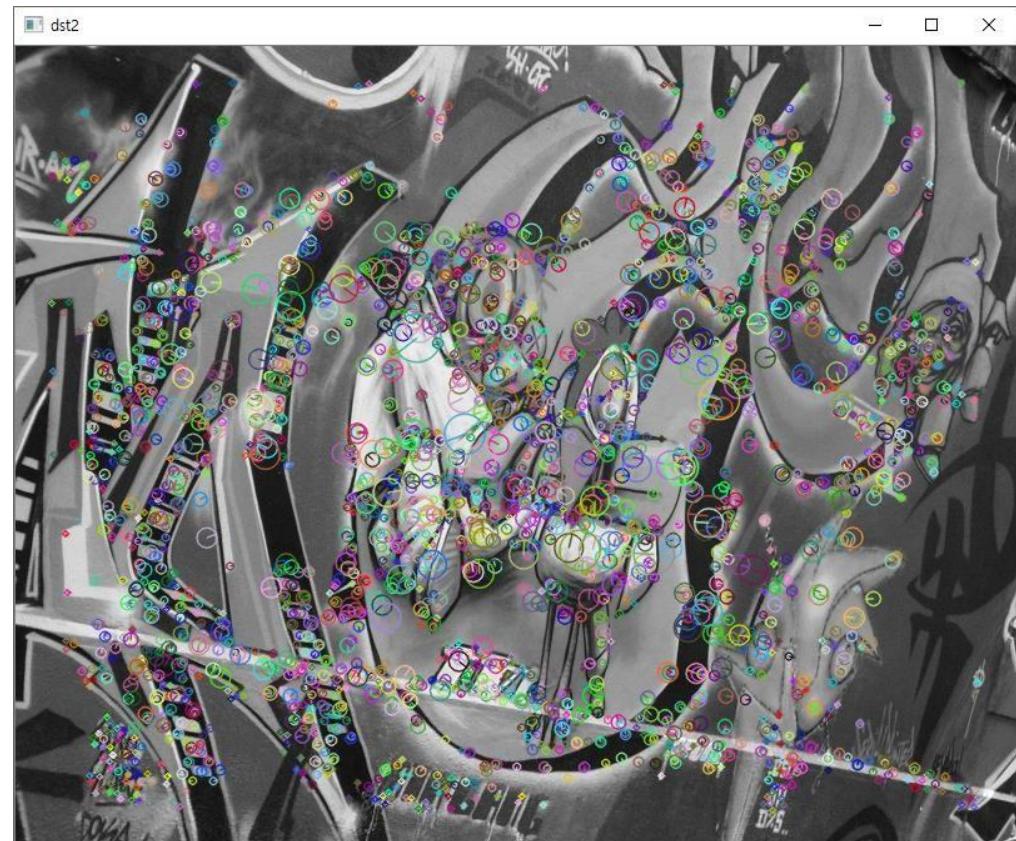
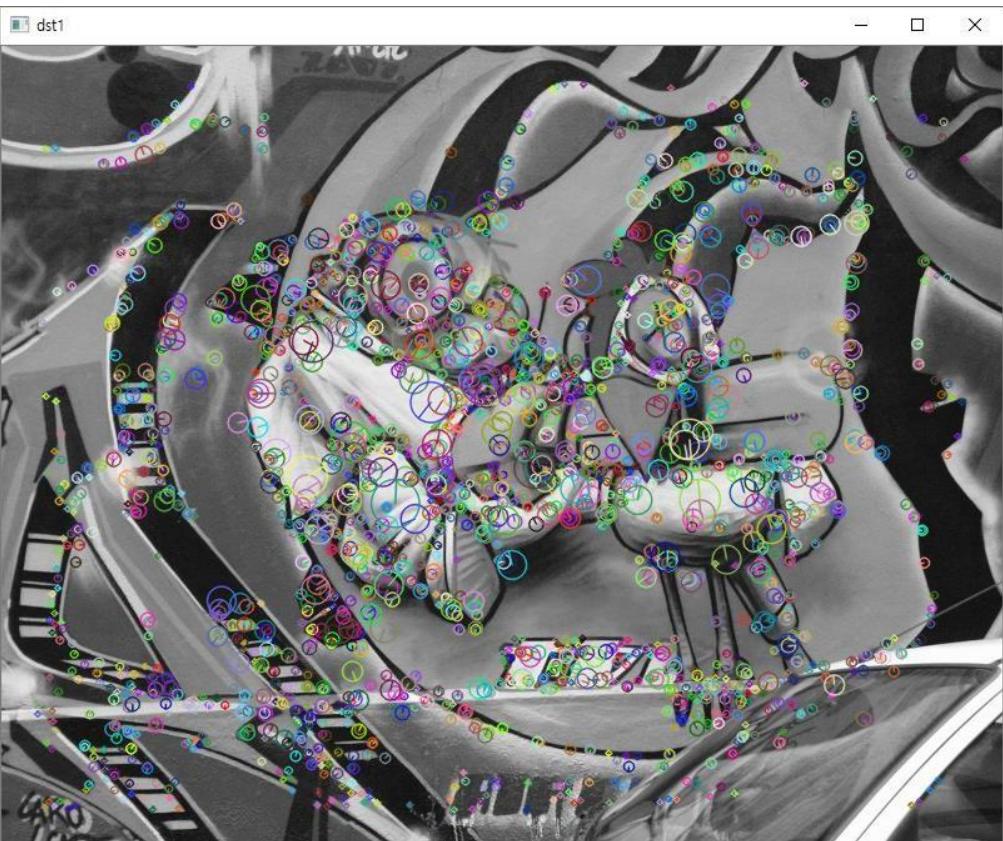
# 특징점 검출

- KAZE 특징점 검출 결과
  - cv2.KAZE\_create() 함수 사용



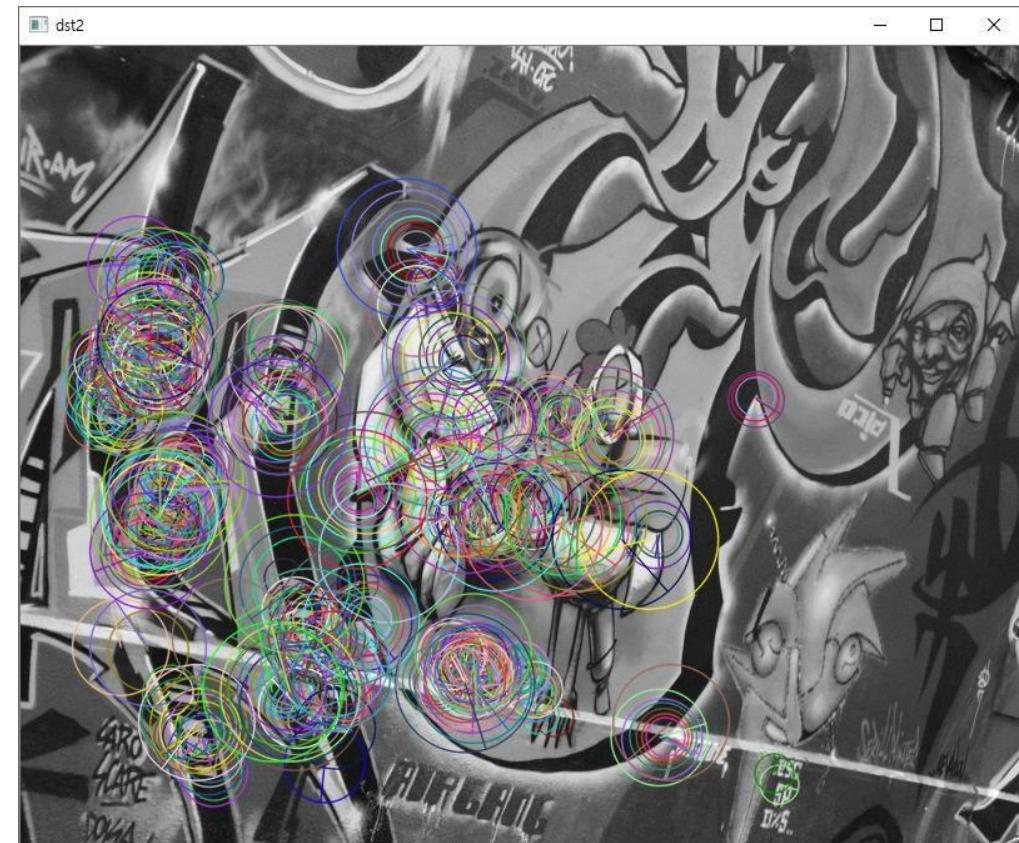
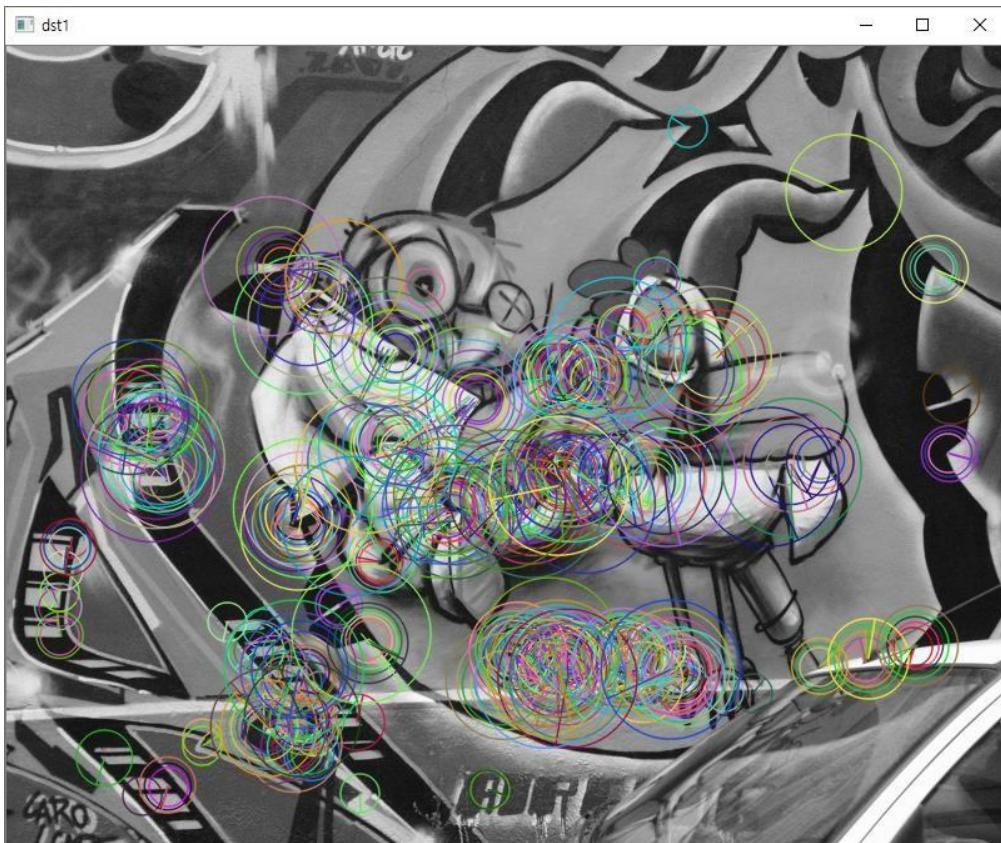
# 특징점 검출

- AKAZE 특징점 검출 결과
  - cv2.AKAZE\_create() 함수 사용



# 특징점 검출

- ORB 특징점 검출 결과
  - cv2.ORB\_create() 함수 사용



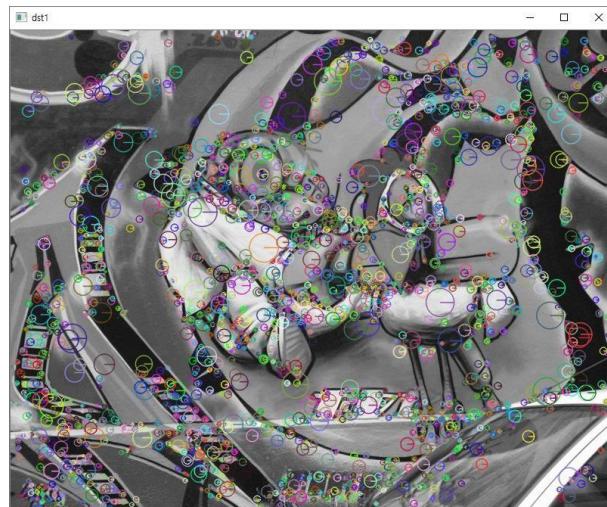
# 9. 특징점 검출과 매칭

3) 특징점 기술

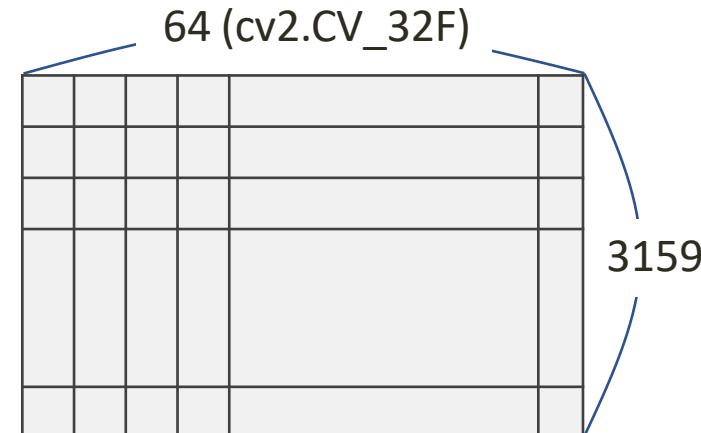
# 특징점 기술

## ■ 기술자(Descriptor, feature vector)

- 특징점 근방의 부분 영상을 표현하는 실수 또는 이진 벡터
- OpenCV에서는 2차원 행렬(numpy.ndarray)로 표현
  - 행 개수: 특징점 개수 / 열 개수: 특징점 기술자 알고리즘에 의해 정의됨
  - 실수 기술자: numpy.float32 / 이진 기술자: numpy.uint8



KAZE (3159개)

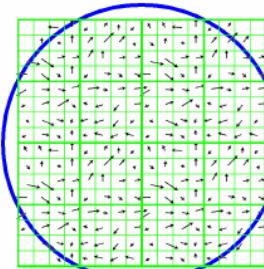
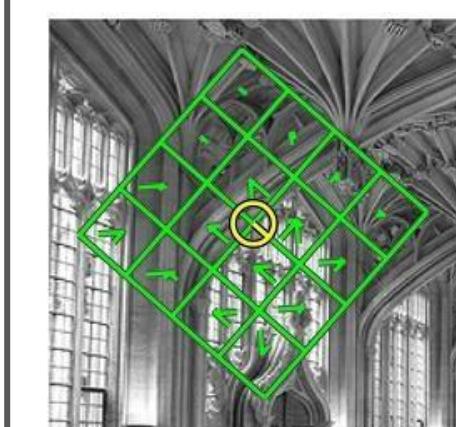


기술자 행렬

# 특징점 기술

## ■ 실수 기술자

- 주로 특징점 부근 부분 영상의 방향 히스토그램을 사용



→

*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

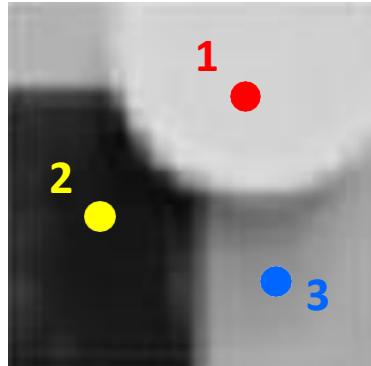
- 특징점 근방 부분 영상의 주 방향 성분을 계산하여 보정
- 보정된 사각형 영역을  $4 \times 4$  구역으로 분할 ⑦ 각 구역에서 8방향 히스토그램을 구함
- $4 \times 4 \times 8 = 128$  차원의 실수 벡터 (알고리즈다다름)

- 보통 `numpy.float32` 자료형을 사용하여 실수 정보를 저장하는 방식
- 실수 기술자를 사용하는 알고리즘: `SIFT`, `SURF`, `KAZE` 등
- 실수 기술자는 보통 `L2 노름(L2 norm)`을 사용하여 유사도를 판단

# 특징점 기술

## ■ 이진 기술자(Binary descriptor)

- 이진 테스트(Binary test)를 이용하여 부분 영상의 특징을 기술



- 특징점 근방 패치( $p$ )에서 point pair ( $x, y$ )의 픽셀 값 크기 테스트:

$$\tau(p; x, y) := \begin{cases} 1 & \text{if } p(x) < p(y) \\ 0 & \text{otherwise} \end{cases}$$

- $n_d$  차원 특징 벡터(기술자):

$$f_{n_d}(p) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(p; x_i, y_i)$$

- 보통 `numpy.uint8` 자료형을 사용하여 비트 단위로 영상 특징 정보를 저장하는 방식
- 이진 기술자를 사용하는 알고리즘: [AKAZE](#), [ORB](#), [BRIEF](#) 등
- 이진 기술자는 [해밍 거리\(Hamming distance\)](#)를 사용하여 유사도를 판단

# 특징점 기술

## ■ 특징점 기술자 계산 함수

```
cv2.Feature2D.compute(image, keypoints, descriptors=None)  
    -> keypoints, descriptors
```

- image: 입력 영상
- keypoints: 검출된 특징점 정보. cv2.KeyPoint 객체의 리스트.
- descriptors: 특징점 기술자 행렬

# 특징점 기술

## ■ 특징점 검출 및 기술자 계산 함수

```
cv2.Feature2D.detectAndCompute(image, mask=None, descriptors=None)  
    -> keypoints, descriptors
```

- **image:** 입력 영상
- **mask:** 마스크 영상
- **keypoints:** 검출된 특징점 정보. cv2.KeyPoint 객체의 리스트.
- **descriptors:** 특징점 기술자 행렬

# 특징점 기술

실습: descriptors.py

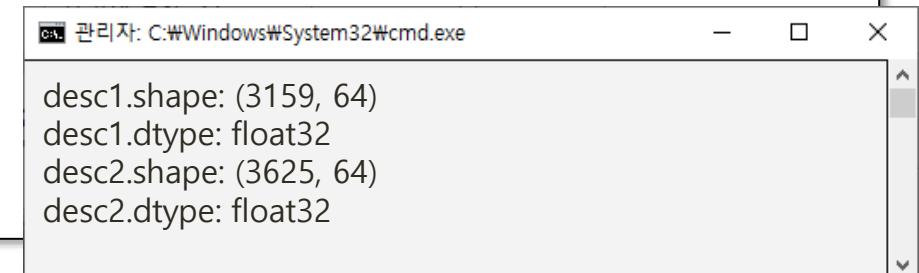
## ■ 특징점 기술자 계산 예제

```
# 영상 불러오기
src1 = cv2.imread('graf1.png', cv2.IMREAD_GRAYSCALE)
src2 = cv2.imread('graf3.png', cv2.IMREAD_GRAYSCALE)

# 두 영상에서 특징점 검출 & 출력 (KAZE, AKAZE, ORB 등)
feature = cv2.KAZE_create()

# 두 가지 기술자 계산 예제 코드
kp1 = feature.detect(src1)
_, desc1 = feature.compute(src1, kp1)
kp2, desc2 = feature.detectAndCompute(src2, None)

print('desc1.shape:', desc1.shape)
print('desc1.dtype:', desc1.dtype)
print('desc2.shape:', desc2.shape)
print('desc2.dtype:', desc2.dtype)
```



# 특징점 기술

## ■ OpenCV 주요 특징점 알고리즘과 기술자 특성

특징점 알고리즘	기술자 차원	데이터 타입	이진 기술자	Extra Module	비고
SIFT	128	numpy.float32	X	O	알고리즘 특허 만료. OpenCV 4.4 버전부터 main module로 통합.
SURF	64	numpy.float32	X	O	알고리즘 특허.
KAZE	64	numpy.float32	X	X	
AKAZE	61	numpy.uint8	O	X	
ORB	32	numpy.uint8	O	X	가장 빠름
BRISK	64	numpy.uint8	O	X	

# 특징점 기술

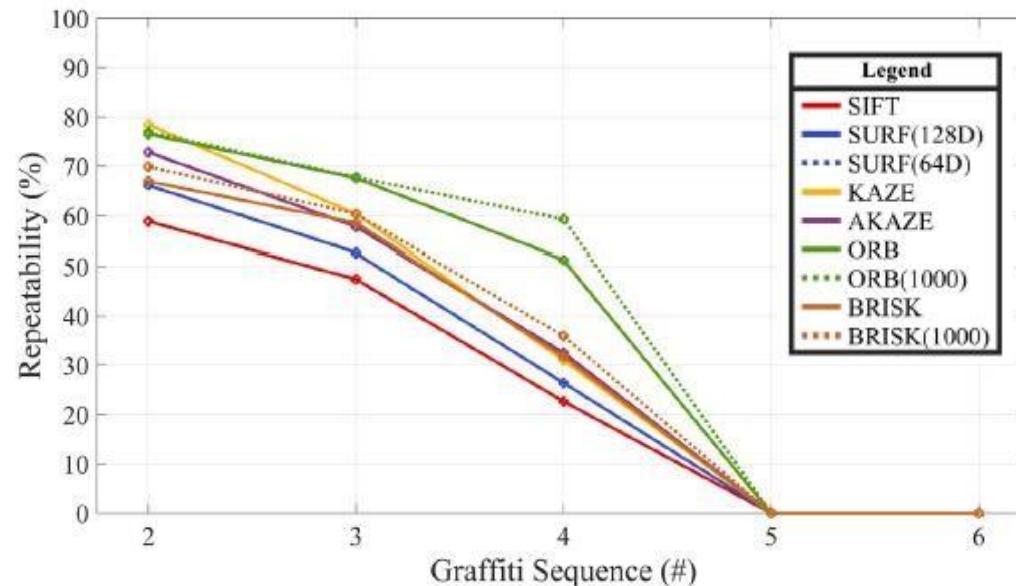
- 특징점 검출 알고리즘 성능 비교
  - 연산 시간 비교

Algorithm	Features Detected in the Image Pairs		Feature Detection & Description Time (s)		Total Image Matching Time (s)
	1 <sup>st</sup> Image	2 <sup>nd</sup> Image	1 <sup>st</sup> Image	2 <sup>nd</sup> Image	
<b>Mean Values for All Image Pairs</b>					
SIFT	3125.7	3662.8	0.2827	0.3119	<b>1.1212</b>
SURF(128D)	4317.8	4744.7	0.1847	0.2003	<b>1.1934</b>
SURF(64D)	4317.8	4744.7	0.1806	0.1954	<b>0.8092</b>
KAZE	1517.5	1660.7	0.2902	0.2941	<b>0.6904</b>
AKAZE	1633.0	1776.0	0.0995	0.1013	<b>0.2502</b>
ORB	9782.7	10828.0	0.0385	0.0427	<b>1.1410</b>
ORB(1000)	955.2	955.7	0.0129	0.0133	<b>0.0426</b>
BRISK	6612.3	7476.8	0.1097	0.1253	<b>1.1735</b>
BRISK(1000)	966.0	958.7	0.0200	0.0206	<b>0.0609</b>

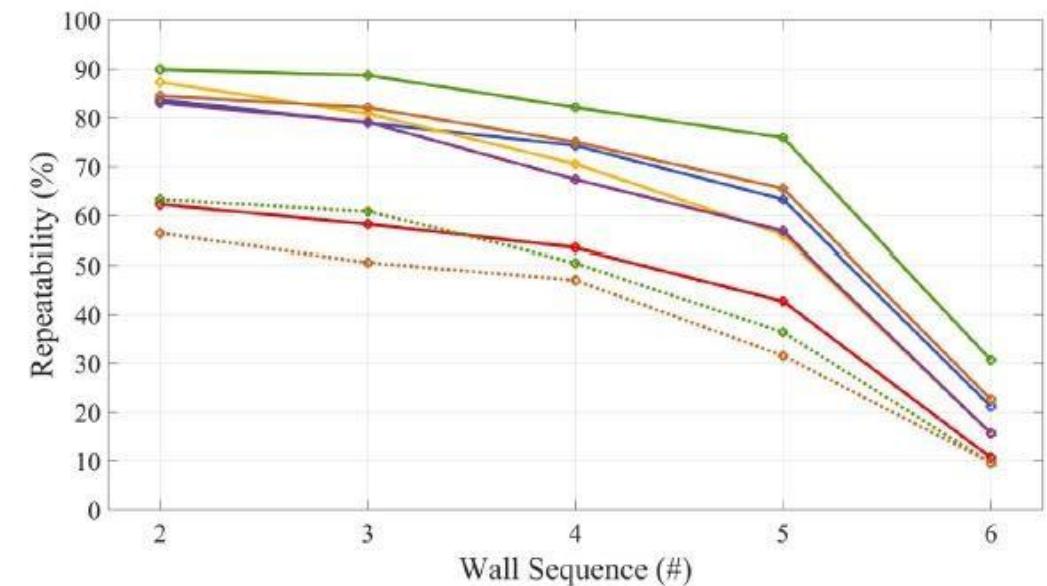
S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Sukkur, 2018

# 특징점 기술

- 특징점 검출 알고리즘 성능 비교
  - 반복 검출율



(c) Repeatability of feature detectors for complete *Graffiti sequence* (viewpoint changes).

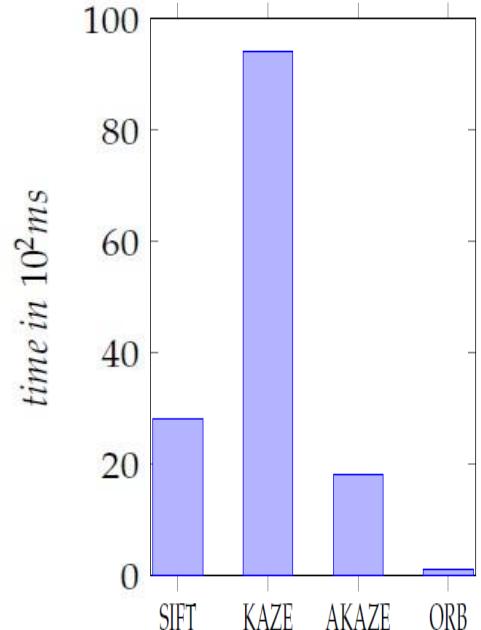
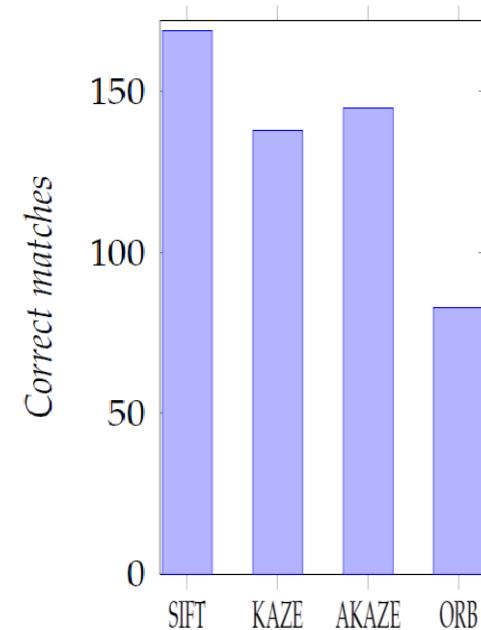
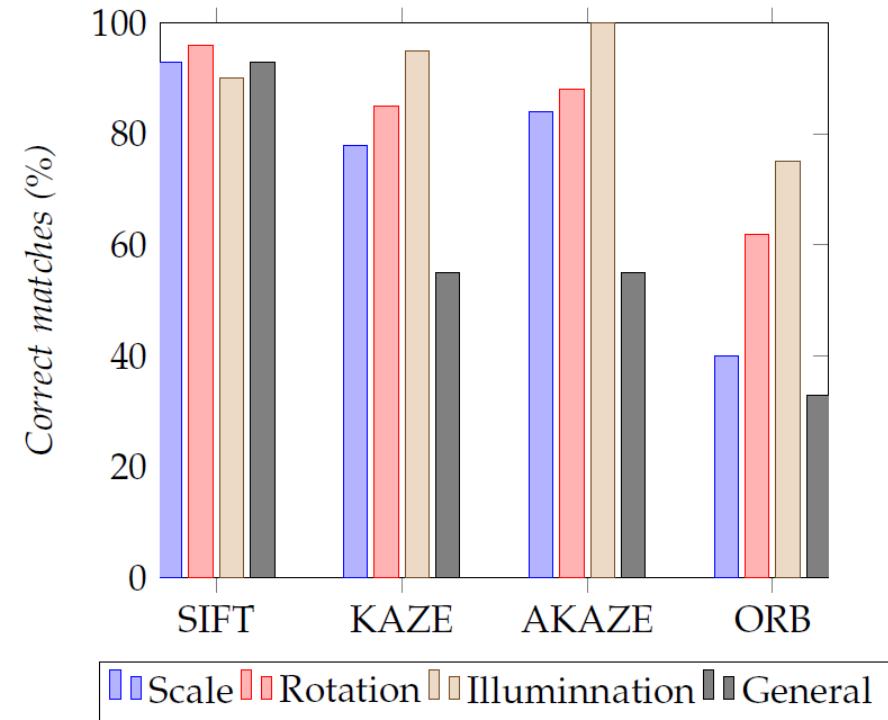


(d) Repeatability of feature detectors for complete *Wall sequence* (viewpoint changes).

S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Sukkur, 2018

# 특징점 기술

## ■ 특징점 검출 알고리즘 성능 비교



<http://www.diva-portal.org/smash/get/diva2:927480/FULLTEXT01.pdf>

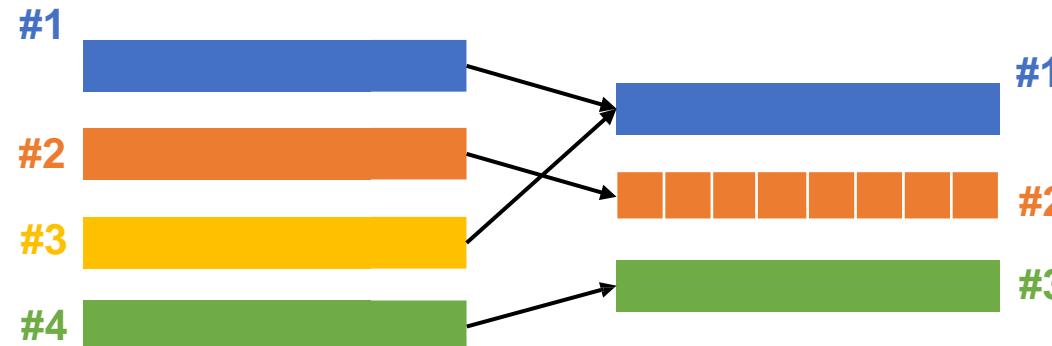
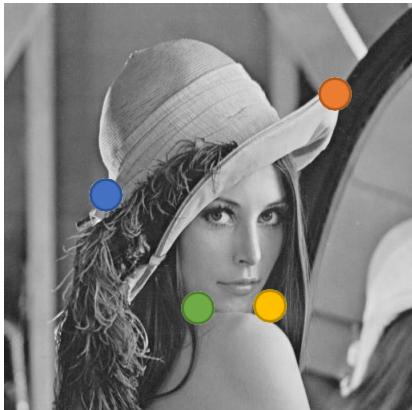
## 9. 특징점 검출과 매칭

4) 특징점 매칭

# 특징점 매칭

## ■ 특징점 매칭 (feature point matching)

- 두 영상에서 추출한 특징점 기술자를 비교하여 서로 유사한 기술자를 찾는 작업

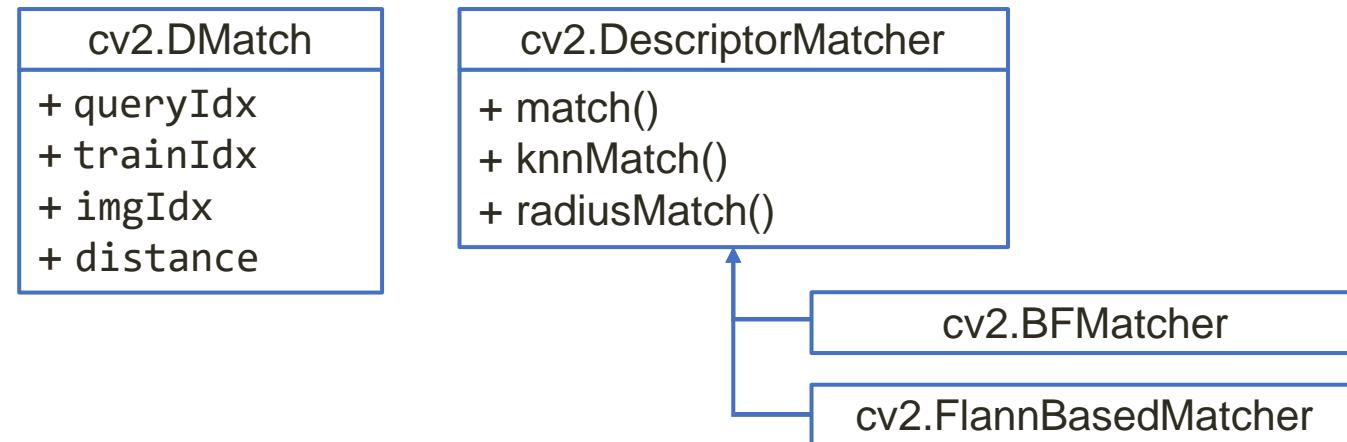


## ■ 특징 벡터 유사도 측정 방법

- 실수 특징벡터: L2 노름(L2 norm) 사용
- 이진 특징 벡터: 해밍 거리(hamming distance) 사용

# 특징점 매칭

## ■ OpenCV 특징점 매칭 클래스



- BF: Brute-force (전수 조사)
- Flann: Fast Library for Approximate Nearest Neighbor (K-D Tree 사용)

[https://docs.opencv.org/master/db/d39/classcv\\_1\\_1DescriptorMatcher.html](https://docs.opencv.org/master/db/d39/classcv_1_1DescriptorMatcher.html)

# 특징점 매칭

## ■ 특징점 매칭 알고리즘 객체 생성

```
cv2.BFMatcher_create(, normType=None, crossCheck=None) -> retval
```

- normType: 특징점 기술자 거리 계산 방식 지정. 기본값은 cv2.NORM\_L2.

cv2.NORM_L1	L1 노름(L1 norm) 사용
cv2.NORM_L2	L2 노름(L2 norm) 사용
cv2.NORM_HAMMING	해밍 거리 사용
cv2.NORM_HAMMING2	두 비트를 한 단위로 취급하여 해밍 거리 계산

- crossCheck: 이 값이 True이면 양방향 매칭 결과가 같은 경우만 반환함. 기본값은 False.

# 특징점 매칭

## ■ 특징점 검출 알고리즘 객체 생성

```
cv2.DescriptorMatcher.match(queryDescriptors, trainDescriptors, mask=None)
                            -> matches
```

- queryDescriptors: (기준 영상 특징점) 질의 기술자 행렬
- trainDescriptors: (대상 영상 특징점) 학습 기술자 행렬
- mask: 매칭 진행 여부를 지정하는 행렬 마스크.
- matches: 매칭 결과. cv2.DMatch 객체의 리스트.

# 특징점 매칭

## ■ 특징점 검출 알고리즘 객체 생성

```
cv2.DescriptorMatcher.knnmatch(queryDescriptors, trainDescriptors, k,  
                                mask=None, compactResult=None) -> matches
```

- queryDescriptors: (기준 영상 특징점) 질의 기술자 행렬
- trainDescriptors: (대상 영상 특징점) 학습 기술자 행렬
- k: 질의 기술자에 대해 검출할 매칭 개수
- mask: 매칭 수행 여부를 지정하는 행렬 마스크
- compactResult: mask가 None이 아닐 때 사용되는 파라미터. 기본값은 False이며, 이 경우 결과 matches는 기준 영상 특징점과 같은 크기를 가짐.
- matches: 매칭 결과. cv2.DMatch 객체의 리스트의 리스트.

# 특징점 매칭

## ■ 특징점 매칭 결과 영상 생성

```
cv2.drawMatches(img1, keypoints1, img2, keypoints2, matches1to2, outImg,  
               matchColor=None, singlePointColor=None, matchesMask=None,  
               flags=None) -> outImg
```

- img1, keypoints1: 기준 영상과 기준 영상에서 추출한 특징점 정보
- img2, keypoints2: 대상 영상과 대상 영상에서 추출한 특징점 정보
- matches1to2: 매칭 정보. cv2.DMatch의 리스트.
- outImg: 출력 영상
- matchColor: 매칭된 특징점과 직선 색상
- singlePointColor: 매칭되지 않은 특징점 색상
- matchesMask: 매칭 정보를 선택하여 그릴 때 사용할 마스크
- flags: 매칭 정보 그리기 방법.  
기본값은 cv2.DRAW\_MATCHES\_FLAGS\_DEFAULT.

# 특징점 매칭

실습: matching.py

## ■ 특징점 매칭 예제

```
src1 = cv2.imread('box.png', cv2.IMREAD_GRAYSCALE)
src2 = cv2.imread('box_in_scene.png', cv2.IMREAD_GRAYSCALE)

detector = cv2.KAZE_create()

kp1, desc1 = detector.detectAndCompute(src1, None)
kp2, desc2 = detector.detectAndCompute(src2, None)

matcher = cv2.BFMatcher_create()

matches = matcher.match(desc1, desc2)

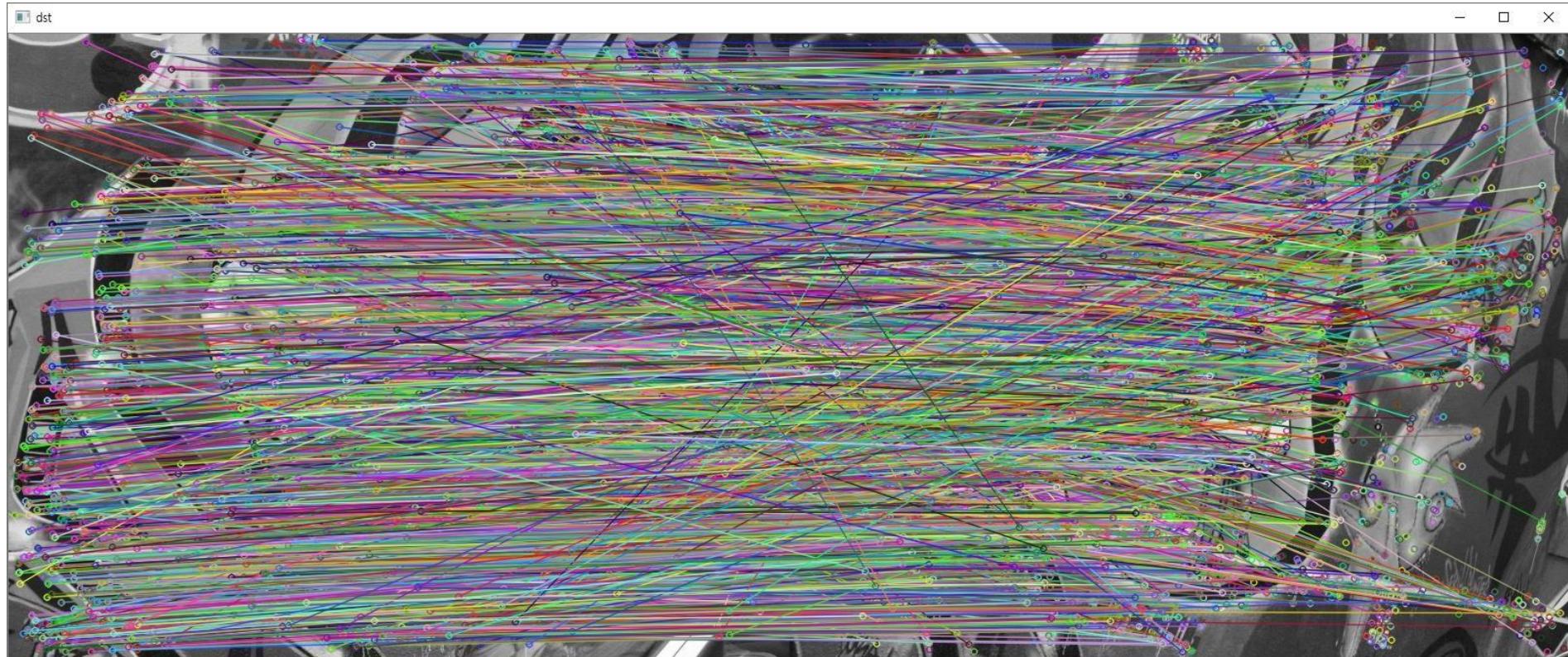
print('# of kp1:', len(kp1))
print('# of kp2:', len(kp2))
print('# of matches:', len(matches))

dst = cv2.drawMatches(src1, kp1, src2, kp2, matches, None)
```



# 특징점 매칭

- 특징점 매칭 예제 실행 결과



## 9. 특징점 검출과 매칭

5) 이미지 스티칭

# 이미지 스티칭

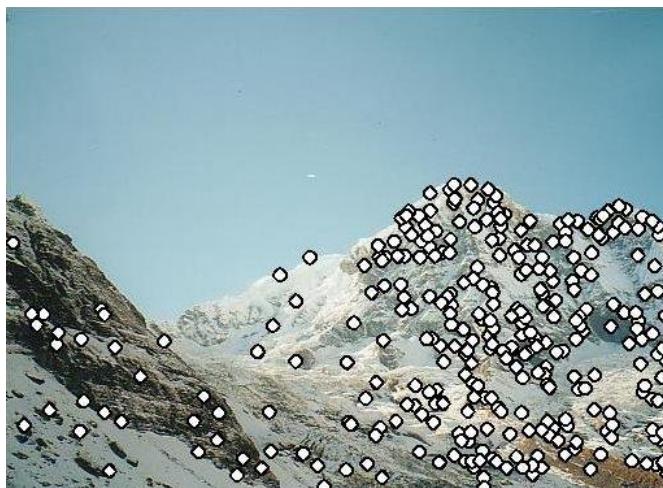
## ■ 이미지 스티칭(Image Stitching)이란?

- 동일 장면의 사진을 자연스럽게(seamless) 붙여서 한 장의 사진으로 만드는 기술
- 사진 이어 붙이기, 파노라마 영상(Panorama image)

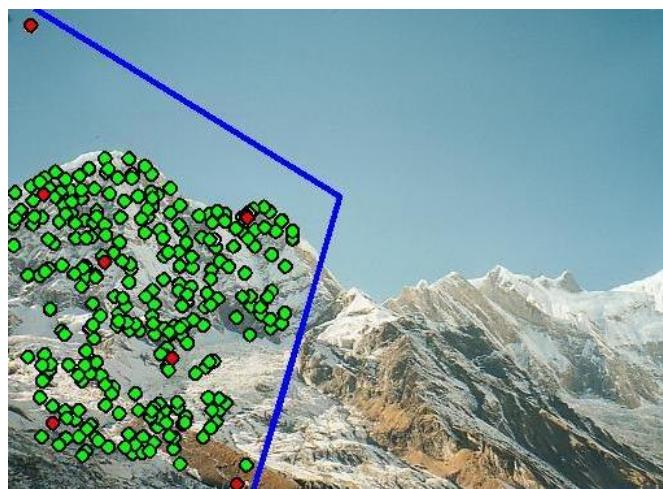
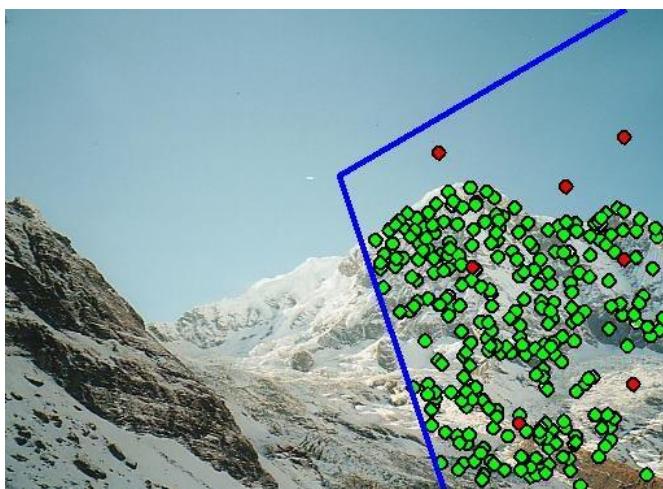


<http://matthewalunbrown.com/papers/ijcv2007.pdf>

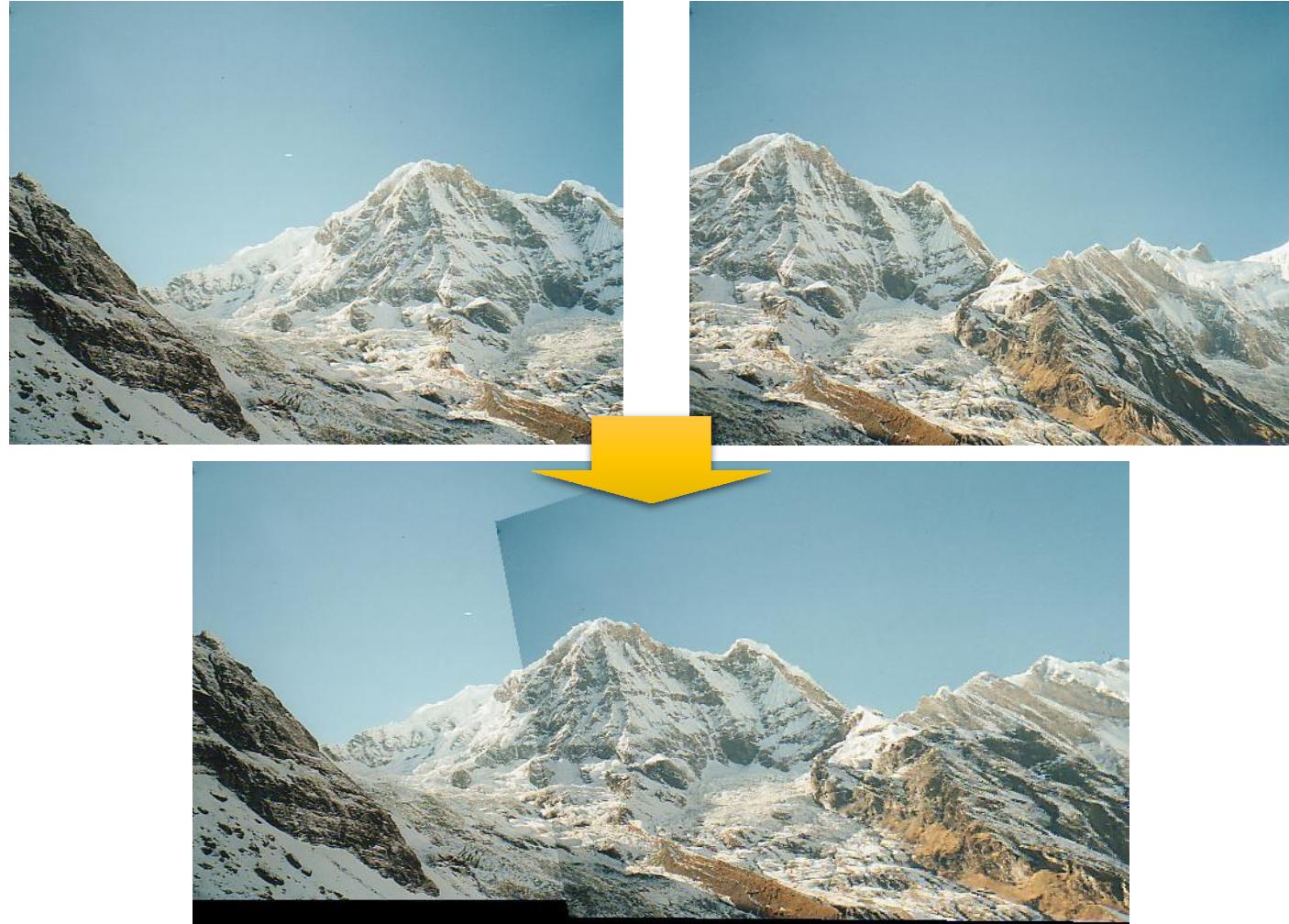
# 이미지 스티칭



# 이미지 스티칭

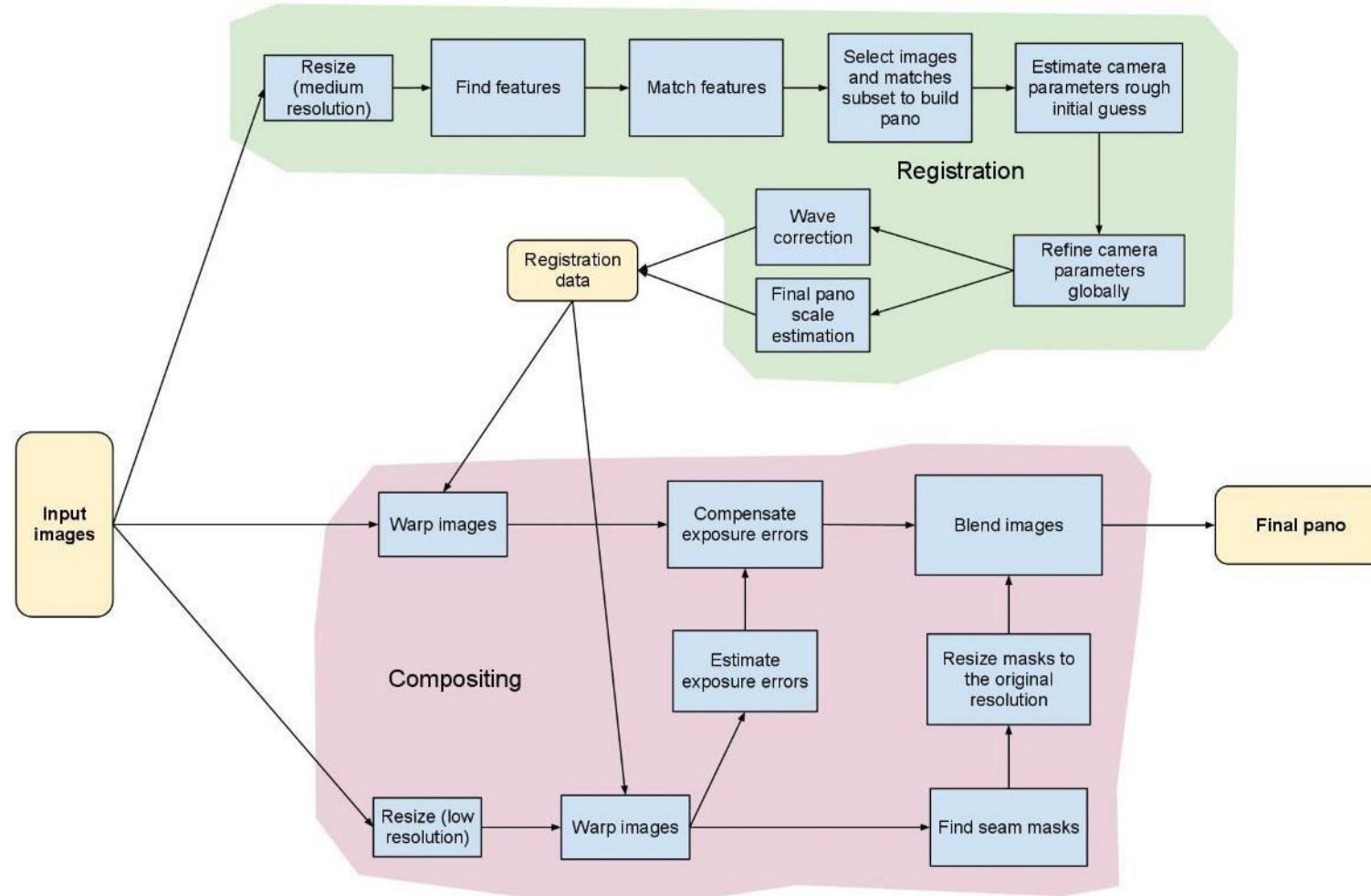


# 이미지 스티칭



# 이미지 스티칭

OPENCV  
를



[https://docs.opencv.org/master/d1/d46/group\\_stitching.html](https://docs.opencv.org/master/d1/d46/group_stitching.html)

# 이미지 스티칭

## ■ 이미지 스티칭 객체 생성

```
cv2.Stitcher_create(, mode=None) -> retval
```

- mode: 스티칭 모드. cv2.PANORAMA 또는 cv2.SCANS 중 하나 선택.  
기본값은 cv2.PANORAMA.
- retval: cv2.Stitcher 클래스 객체

# 이미지 스티칭

## ■ 이미지 스티칭 함수

```
cv2.Stitcher.stitch(images, pano=None) -> retval, pano
```

- images: 입력 영상 리스트 성공
- retval: 하면 cv2.Stitcher\_OK. 파
- pano: 노라마 영상

# 이미지 스티칭

실습: stitching.py

## ■ 이미지 스티칭 예제

```
img_names = ['img1.jpg', 'img2.jpg', 'img3.jpg']

imgs = []
for name in img_names:
    img = cv2.imread(name)
    imgs.append(img)

stitcher = cv2.Stitcher_create()
_, dst = stitcher.stitch(imgs)

cv2.imwrite('output.jpg', dst)
```

# 이미지 스티칭

## ■ 이미지 스티칭 예제 실행 결과



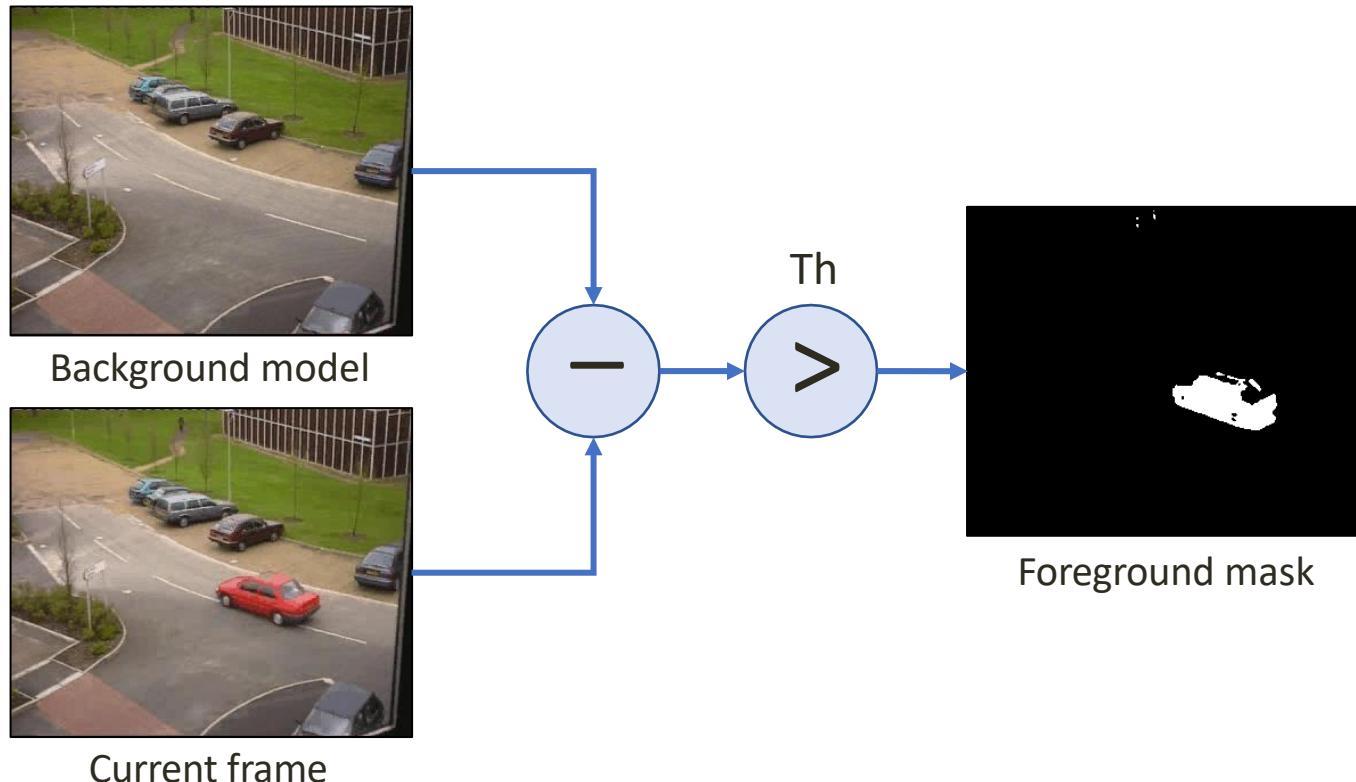
# 10. 객체 추적과 모션 벡터

1) 배경 차분: 정적 배경 차분

# 배경 차분: 정적 배경 차분

## ■ 배경 차분(Background Subtraction: BS)

- 등록된 배경 모델과 현재 입력 프레임과의 차영상을 이용하여 전경 객체를 검출
- 움직이는 전경 객체 검출을 위한 기본적인 방법



# 배경 차분: 정적 배경 차분

실습: bs\_static1.py

## ■ 정적 배경을 이용한 전경 객체 검출

```
cap = cv2.VideoCapture('PETS2000.avi')

# 배경 영상 등록
ret, back = cap.read()
back = cv2.cvtColor(back, cv2.COLOR_BGR2GRAY)
#back = cv2.GaussianBlur(back, (0, 0), 1.0)

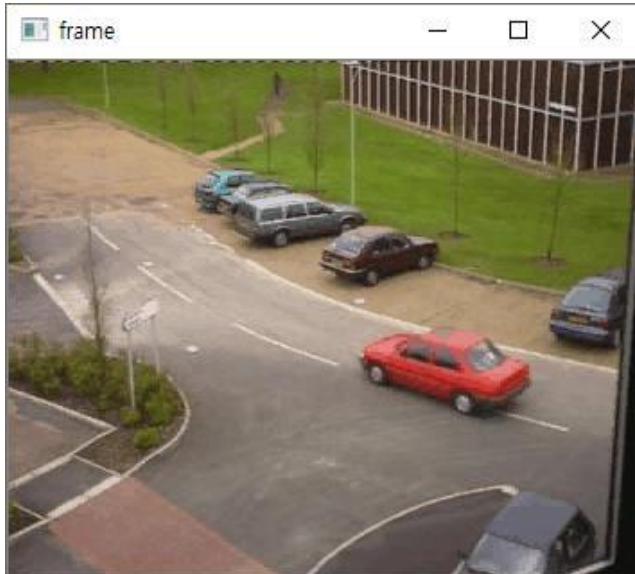
while True:
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #gray = cv2.GaussianBlur(gray, (0, 0), 1.0)

    # 차영상 구하기 & 이진화
    diff = cv2.absdiff(blr, back)
    _, diff = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)
```

# 배경 차분: 정적 배경 차분

- 정적 배경을 이용한 전경 객체 검출 실행 결과
  - 배경 영상과 현재 프레임에 대해 각각 가우시안 필터링을 수행하여 잡음 제거



현재 프레임(#130)



차영상

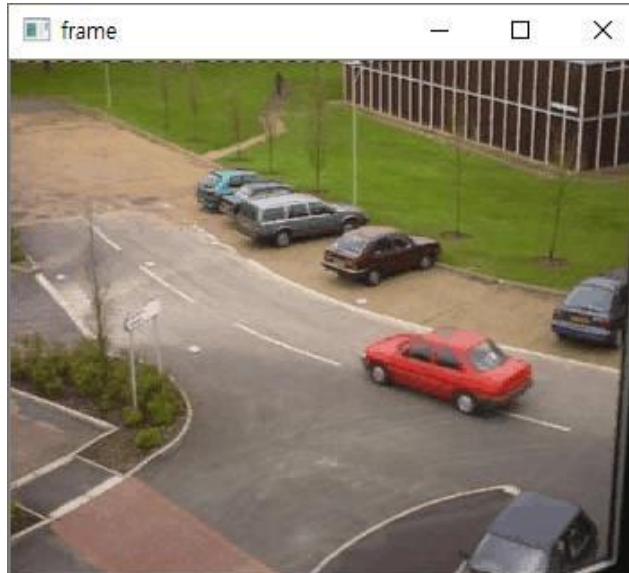


가우시안 필터 사용

# 배경 차분: 정적 배경 차분

실습: bs\_static2.py

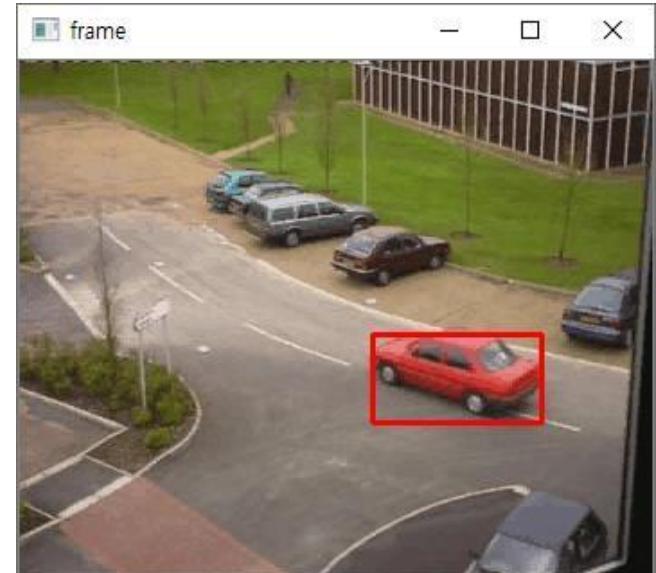
- 정적 배경을 이용한 전경 객체 검출 후 주요 객체 바운딩 박스 표시
  - 레이블링 수행 후 픽셀 개수가 100 이상인 객체에 바운딩 박스 표시



현재 프레임(#130)



가우시안 필터& 차영상



바운딩 박스 표시

# 10. 객체 추적과 모션 벡터

2) 배경 차분: 이동 평균 배경

# 배경 차분: 이동 평균 배경

## ■ 정적 배경 모델 사용 시 문제점

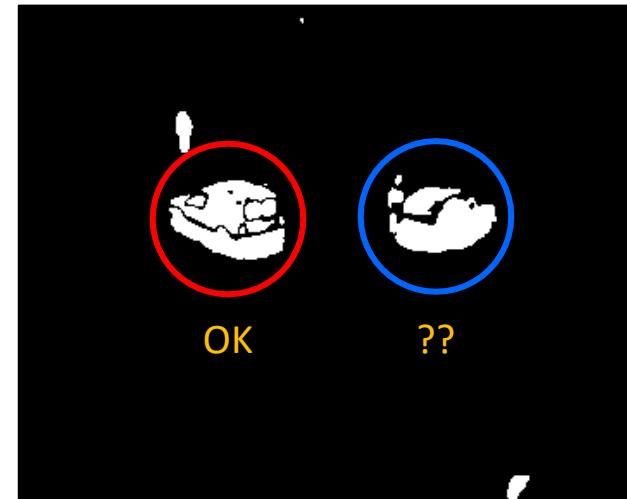
- 미리 등록된 기준 영상이 실제 배경과 크게 달라질 경우 오동작  
(e.g) 그림자 등의 영향으로 인한 조도 변경, 새로운 객체가 화면에 고정될 경우



고정 객체 발생



현재 프레임 (#840)



차영상

# 배경 차분: 이동 평균 배경

- 평균 연산에 의한 배경 영상 생성
  - 움직이는 객체가 존재하는 수백 장의 입력 영상으로부터 평균 영상을 구하면?



- 수백 장의 이전 프레임을 버퍼에 저장하려면 대용량 메모리가 필요

# 배경 차분: 이동 평균 배경

## ■ 이동 평균(Moving average)이란?

- 수백 장의 영상을 저장하는 대신 매 프레임이 들어올 때마다 평균 영상을 갱신

$$B(x, y, t) = \alpha \cdot I(x, y, t) + (1 - \alpha) \cdot B(x, y, t-1)$$

갱신된 배경 영상

현재 프레임

이전 프레임까지의 배경 영상

현재 프레임에 대한 가중치 ( $0 < \alpha < 1$ )

The diagram shows the formula for moving average background subtraction. Three blue arrows point from labels below the equation to its components: an arrow points to the first term  $B(x, y, t)$  labeled '갱신된 배경 영상' (updated background image); another arrow points to the second term  $I(x, y, t)$  labeled '현재 프레임' (current frame); and a third arrow points to the third term  $B(x, y, t-1)$  labeled '이전 프레임까지의 배경 영상' (background image from previous frames). Below the second term, there is also a label '현재 프레임에 대한 가중치 ( $0 < \alpha < 1$ )' (weighting factor for the current frame).

- 대용량 버퍼 메모리가 필요하지 않음

# 배경 차분: 이동 평균 배 경

- 이동 평균 계산을 위한 가중치 누적 함수

```
cv2.accumulateWeighted(src, dst, alpha, mask=None) -> dst
```

- src: 입력 영상. 1 또는 3채널. 8비트 또는 32비트 실수형
- dst: 축적 영상. 입력 영상과 동일 채널 개수.  
32비트 또는 64비트 실수형
- alpha: (입력 영상에 대한) 가중치
- mask: 마스크 영상

$$dst(x, y) = (1 - \alpha) \cdot dst(x, y) + \alpha \cdot src(x, y) \quad \text{if} \quad mask(x, y) \neq 0$$

# 배경 차분: 이동 평균 배경

실습: bs\_runave.py

## ■ 이동 평균에 의한 배경 차분 예제

```
# 배경 영상 등록
ret, back = cap.read()
back = cv2.cvtColor(back, cv2.COLOR_BGR2GRAY)
back = cv2.GaussianBlur(back, (0, 0), 1.0)
fback = back.astype(np.float32) # 실수형 배경 영상

while True:
    ret, frame = cap.read()

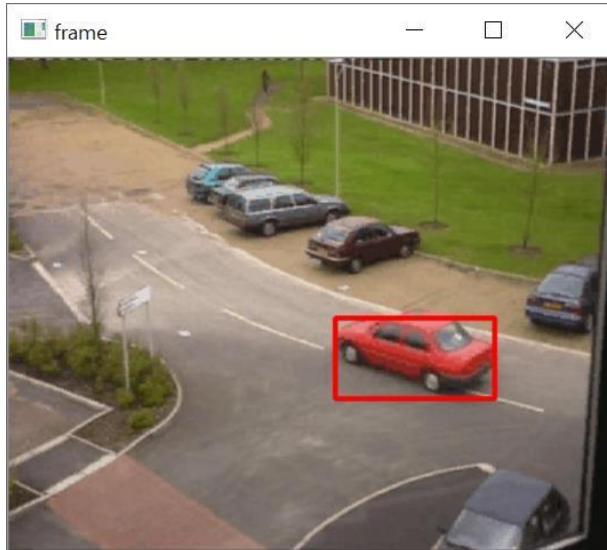
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (0, 0), 1.0)

    cv2.accumulateWeighted(gray, fback, 0.01) # 이동 평균
    back = fback.astype(np.uint8)

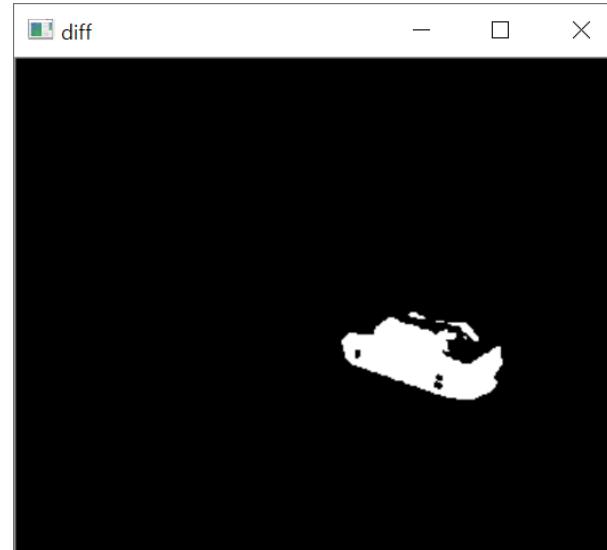
    diff = cv2.absdiff(gray, back)
    _, diff = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)
```

# 배경 차분: 이동 평균 배경

- 이동 평균에 의한 배경 차분 예제 실행 결과



현재 프레임



차영상



이동 평균 배경 영상

# 10. 객체 추적과 모션 벡터

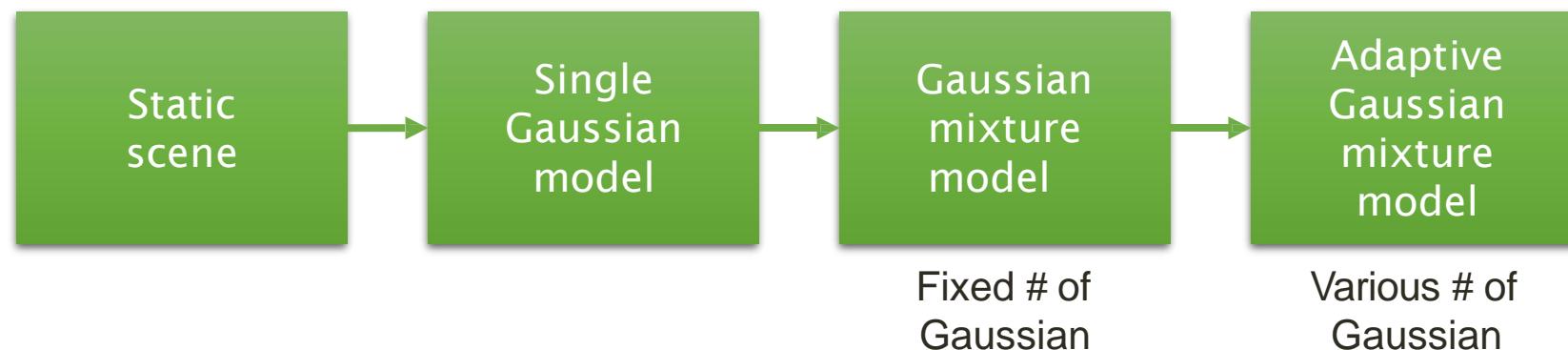
3) 배경 차분: MOG 배경모델

# 배경 차분: MOG 배경 모델

## ■ MOG란?

- Mixture of Gaussian, GMM(Gaussian Mixture Model)
- 각 픽셀에 대해 MOG 확률 모델을 설정하여 배경과 전경을 구분

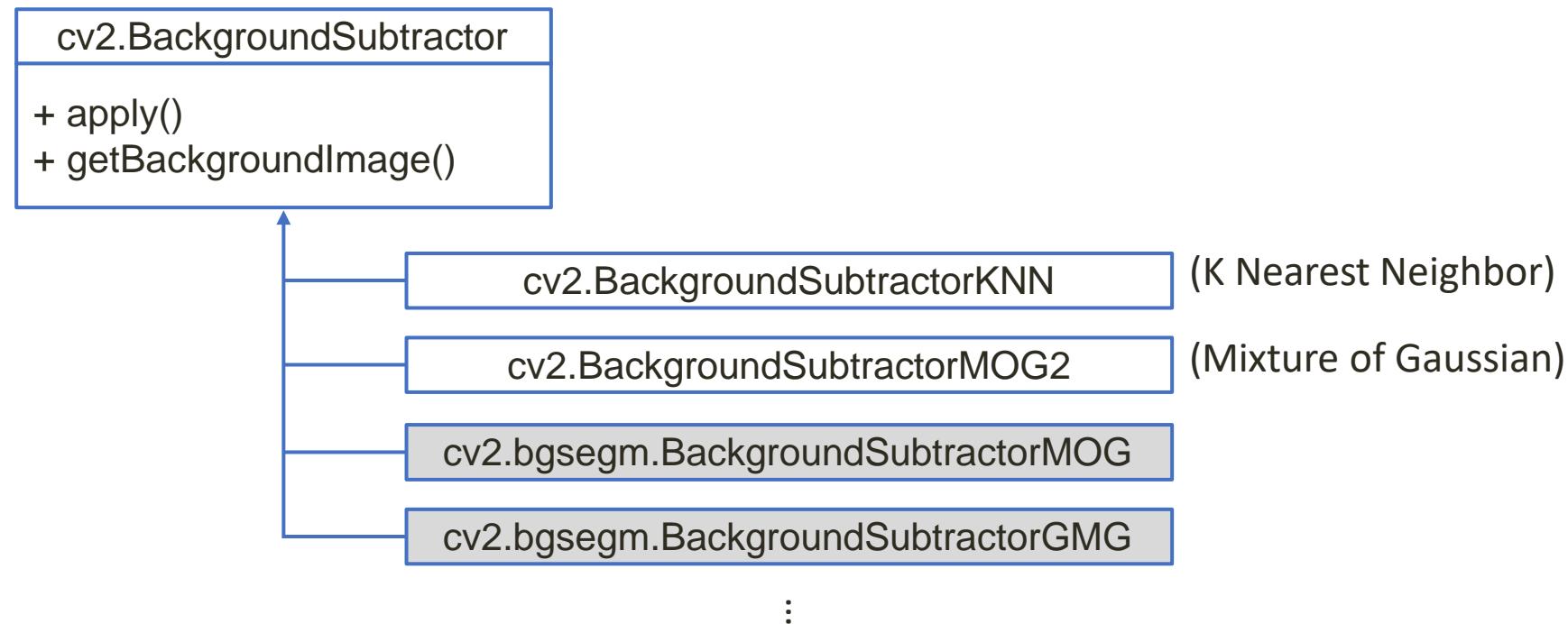
## ■ 다양한 배경 모델 구성 방법



Zoran Zivkovic, “Improved adaptive Gaussian mixture model for background subtraction,” *International Conference on Pattern Recognition*, vol. 2, p. 28–31. IEEE, 2004.

# 배경 차분: MOG 배경 모델

- OpenCV에서 제공하는 배경 추정 알고리즘



# 배경 차분: MOG 배경모델

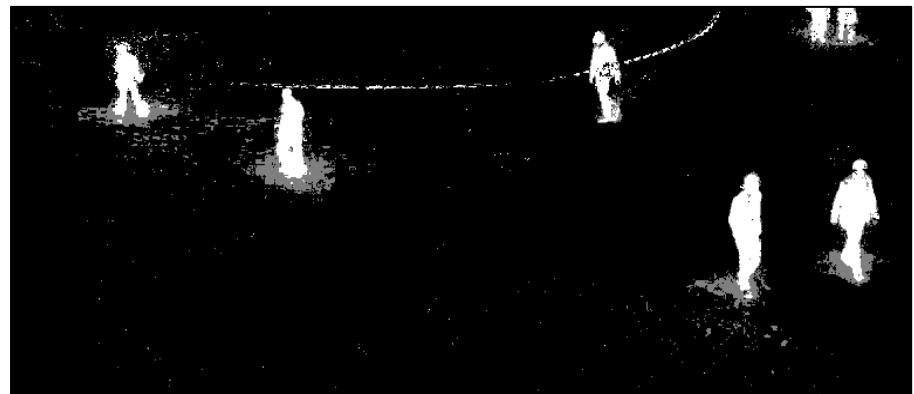
- 움직이는 전경 객체 마스크 영상



입력 프레임 (#200)



BackgroundSubtractorKNN



BackgroundSubtractorMOG2

# 배경 차분: MOG 배경 모델

## ■ BackgroundSubtractorMOG2 클래스 생성 함수

```
cv2.createBackgroundSubtractorMOG2(, history=None, varThreshold=None,  
detectShadows=None) -> dst
```

- history: 히스토리 길이. 기본값은 500.
- varThreshold: 픽셀과 모델 사이의 마할라노비스 거리(Mahalanobis distance) 제곱에 대한 임계값. 해당 픽셀이 배경 모델에 의해 잘 표현되는지를 판단. 기본값은 16.
- detectShadows: 그림자 검출 여부. 기본값은 True.

# 배경 차분: MOG 배경 모델

## ■ 전면 객체 마스크 생성 함수

```
cv2.BackgroundSubtractor.apply(image, fgmask=None, learningRate=None)  
    -> fgmask
```

- image: (입력) 다음 비디오프레임
- fgmask: (출력) 전경 마스크 영상. 8비트 이진 영상.
- learningRate: 배경 모델 학습 속도 지정 (0~1 사이의 실수). 기본값은 -1.

0	배경 모델을 갱신하지 않음
1	매 프레임마다 배경 모델을 새로 만듦
음수	자동으로 결정됨

# 배경 차분: MOG 배경 모델

## ■ 배경 영상 반환 함수

```
cv2.BackgroundSubtractor.getBackgroundImage(, backgroundImage=None)  
          -> backgroundImage
```

- backgroundImage: (출력) 학습된 배경 영상

# 배경 차분: MOG 배경 모델

실습: bs\_mog2.py

- MOG 기법을 이용한 배경 생성 및 전경 객체 검출

```
cap = cv2.VideoCapture('PETS2000.avi')

bs = cv2.createBackgroundSubtractorMOG2()
#bs = cv2.createBackgroundSubtractorKNN()
#bs.setDetectShadows(False)

while True:
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    fgmask = bs.apply(gray)
    back = bs.getBackgroundImage()
    ...
```

# 배경 차분: MOG 배경 모델

- BackgroundSubtractorMOG2 방법에 의해 추출된 전경 & 배경 영상



현재 프레임



전경 마스크영상  
(회색: 그림자)



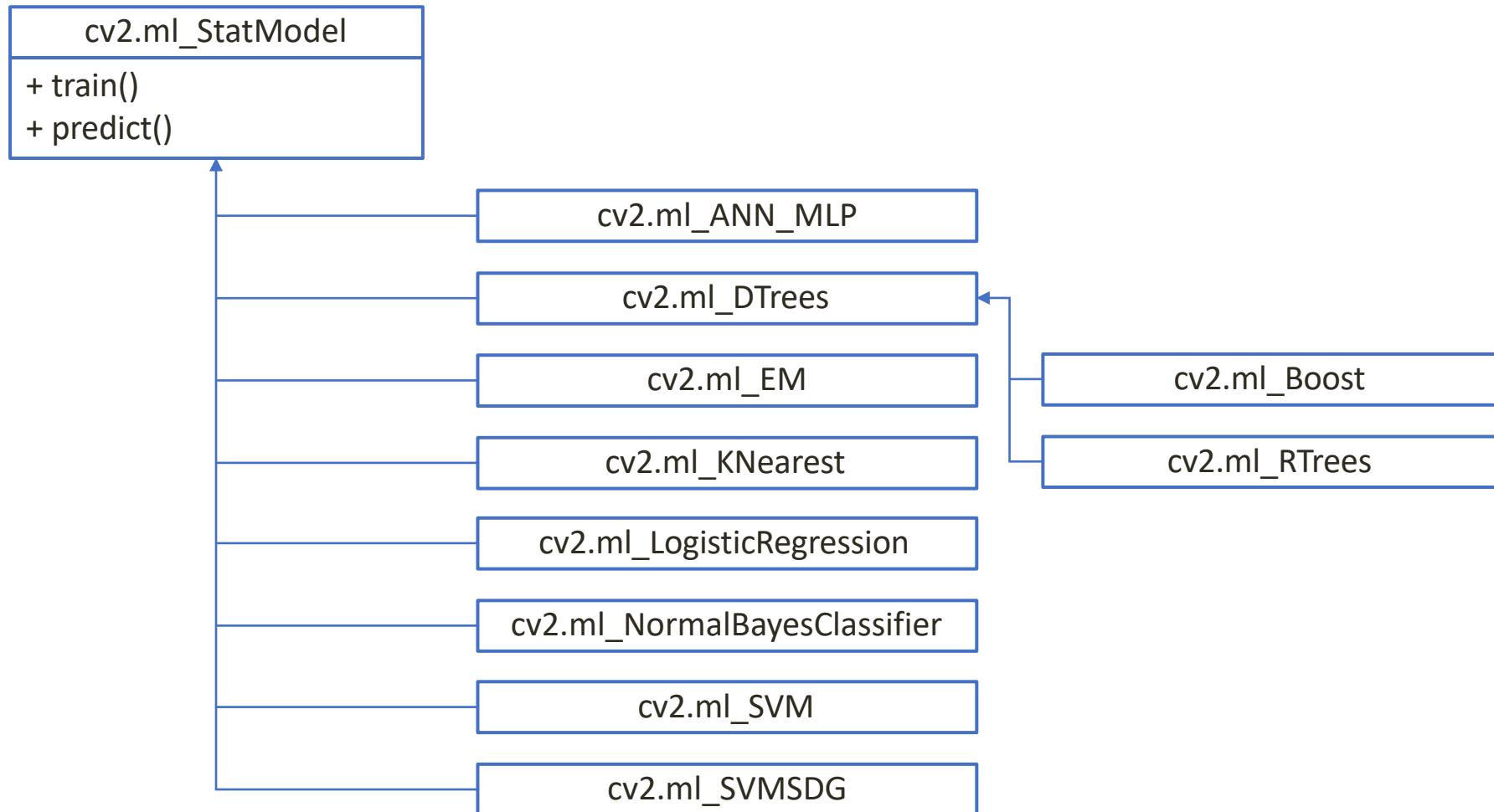
MOG 배경 영상

**2장 OpenCV with 머신 러닝과 딥러닝!**

# 11. 머신 러닝

1) OpenCV 머신 러닝 클래스

# OPENCV 머신 러닝 클래스



# OPENCV 머신 러닝 클래스

## ■ OpenCV 머신 러닝 클래스 설명

클래스 이름	설명
ANN_MLP	인공 신경망(artificial neural network) 다층 퍼셉트론(multi-layer perceptron). 여러 개의 은닉층을 포함한 신경망을 학습시킬 수 있고, 입력 데이터에 대한 결과를 예측할 수 있습니다.
DTrees	이진 의사 결정 트리(decision trees) 알고리즘. DTrees 클래스는 다시 부스팅 알고리즘을 구현한 ml::Boost 클래스와 랜덤 트리(random tree) 알고리즘을 구현한 ml::RTree 클래스의 부모 클래스 역할을 합니다.
Boost	부스팅(boosting) 알고리즘. 다수의 약한 분류기(weak classifier)에 적절한 가중치를 부여하여 성능이 좋은 분류기를 만드는 방법입니다.
RTrees	랜덤 트리(random tree) 또는 랜덤 포레스트(random forest) 알고리즘. 입력 특징 벡터를 다수의 트리로 예측하고, 그 결과를 취합하여 분류 또는 회귀를 수행합니다.
EM	기댓값 최대화(Expectation Maximization). 가우시안 혼합 모델(Gaussian mixture model)을 이용한 군집화 알고리즘입니다.

# OPENCV 머신 러닝 클래스

## ■ OpenCV 머신 러닝 클래스 설명

클래스 이름	설명
KNearest	K 최근접 이웃(K-Nearest Neighbors) 알고리즘. K 최근접 이웃 알고리즘은 샘플 데이터와 인접한 k개의 학습 데이터를 찾고, 이 중 가장 많은 개수에 해당하는 클래스를 샘플 데이터 클래스로 지정합니다.
LogisticRegression	로지스틱 회귀(logistic regression). 이진 분류 알고리즘의 일종입니다.
NormalBayesClassifier	정규 베이즈 분류기. 정규 베이즈 분류기는 각 클래스의 특징 벡터가 정규 분포를 따른다고 가정합니다. 따라서 전체 데이터 분포는 가우시안 혼합 모델로 표현 가능합니다. 정규 베이즈 분류기는 학습 데이터로부터 각 클래스의 평균 벡터와 공분산 행렬을 계산하고, 이를 예측에 사용합니다.
SVM	서포트 벡터 머신(support vector machine) 알고리즘. 두 클래스의 데이터를 가장 여유 있게 분리하는 초평면을 구합니다. 커널 기법을 이용하여 비선형 데이터 분류에도 사용할 수 있으며, 다중 클래스 분류 및 회귀에도 적용할 수 있습니다.
SVMSDG	통계적 그래디언트 하향(stochastic gradient descent) SVM. 통계적 그래디언트 하향 방법을 SVM에 적용함으로써 대용량 데이터에 대해서도 빠른 학습이 가능합니다.

# OPENCV 머신 러닝 API

## ■ 머신 러닝 알고리즘 객체 생성

```
cv2.ml.ANN_MLP_create() -> retval  
cv2.ml.KNearest_create() -> retval  
cv2.ml.SVM_create() -> retval  
...
```

- retval: 각 머신 러닝 알고리즘 객체

# OPENCV 머신 러닝 API

## ■ 머신 러닝 알고리즘 학습

```
cv2.ml_StatModel.train(samples, layout, responses) -> retval
```

- samples: 학습 데이터 행렬. `numpy.ndarray`. `shape=(N, d)`, `dtype=numpy.float32`.
- layout: 학습 데이터 배치 방법

cv2.ROW_SAMPLE	하나의 데이터가 한 행으로 구성됨
cv2.COL_SAMPLE	하나의 데이터가 한 열로 구성됨

- responses: 각 학습 데이터에 대응되는 응답(레이블) 행렬.  
`numpy.ndarray`. `shape=(N, 1)`, `dtype=numpy.int32` 또는 `numpy.float32`.
- retval: 학습이 성공하면 `True`.

# OPENCV 머신 러닝 API

## ■ 머신 러닝 알고리즘 예측

```
cv2.ml_StatModel.predict(samples, results=None, flags=None) -> retval, results
```

- samples:      입력 벡터가 행 단위로 저장된 행렬.  
`numpy.ndarray. shape=(N, d), dtype=numpy.float32.`
- results:      각 입력 샘플에 대한 예측(분류 또는 회귀) 결과를 저장한 행렬.  
`numpy.ndarray. shape=(N, ) 또는 (N,1).`  
`dtype=numpy.int32 또는 numpy.float32.`
- flags:        추가적인 플래그. 기본값은 0.  
cv2.ml.STAT\_MODEL\_RAW\_OUTPUT을 지정하면 클래스 레이블이 아닌  
실제 계산 결과 값을 출력.
- retval:        알고리즘에 따라 다름

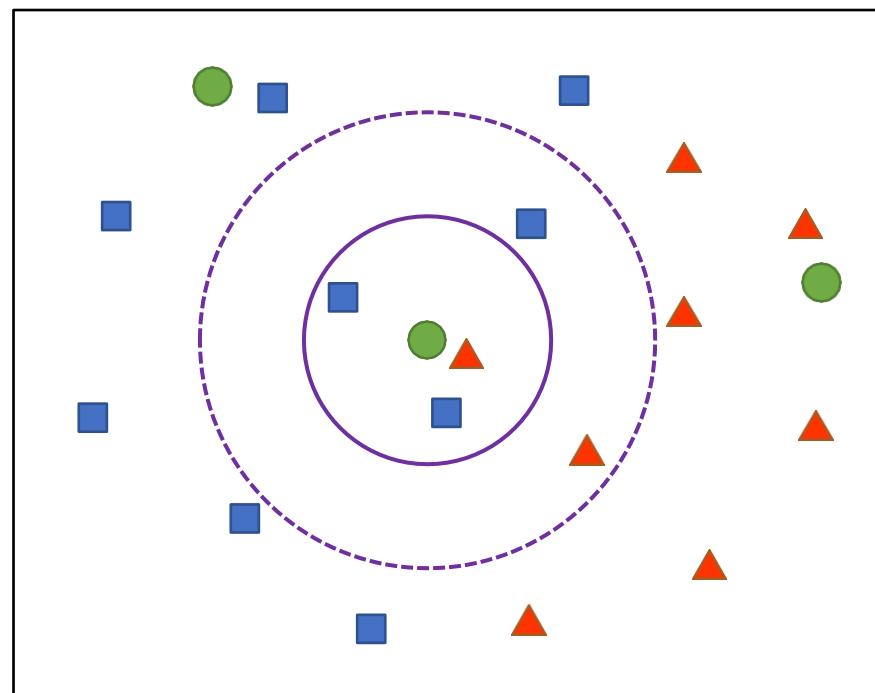
# 11. 머신 러닝

2) k 최근접 이웃 알고리즘

# K 최근접 이웃 알고리즘

## ■ k 최근접 이웃(kNN, k-Nearest Neighbor) 알고리즘이란?

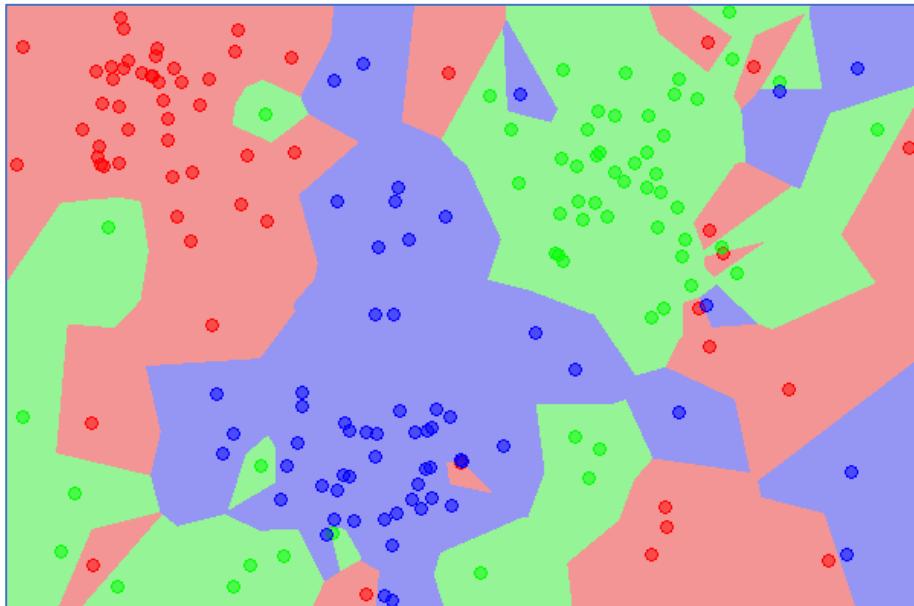
- 특징 공간에서 테스트 데이터와 가장 가까이 있는 k개의 학습 데이터를 찾아 분류 또는 회귀를 수행하는 지도 학습 알고리즘의 하나



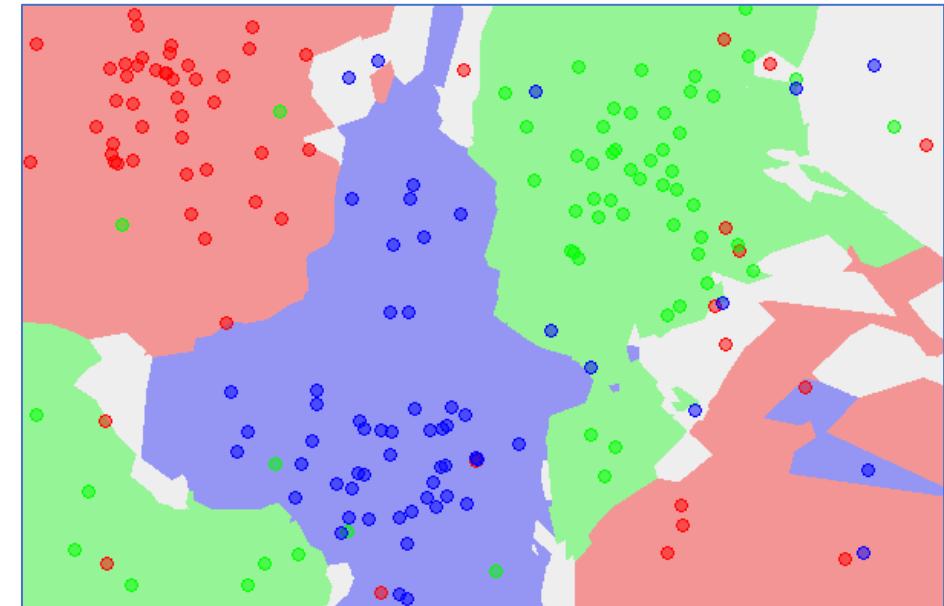
# K 최근접 이웃 알고리즘

## ■ NN vs. kNN

- NN: Nearest neighbor ( $k = 1$ )



NN classifier



5-NN classifier

[https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)

# K 최근접 이웃 알고리즘

## ■ KNN 알고리즘 객체 생성

```
cv2.ml.KNearest_create() -> retval
```

- retval: cv2.ml\_KNearest 객체

# K 최근접 이웃 알고리즘

## ■ KNN 알고리즘으로 입력 데이터의 클래스 예측

```
cv.ml_KNearest.findNearest(samples, k, results=None,  
                           neighborResponses=None, dist=None , flags=None)  
                           -> retval, results, neighborResponses, dist
```

- samples:      입력 벡터가 행 단위로 저장된 입력 샘플 행렬.  
`numpy.ndarray. shape=(N, d), dtype=numpy.float32.`
- k:              사용할 최근접 이웃 개수
- results:        각 입력 샘플에 대한 예측(분류 또는 회귀) 결과를 저장한 행렬.  
`numpy.ndarray. shape=(N, 1), dtype=numpy.float32.`
- neighborResponses:      예측에 사용된 k개의 최근접 이웃 클래스 정보 행렬.  
`numpy.ndarray. shape=(N, k), dtype=numpy.float32.`
- dist:              입력 벡터와 예측에 사용된 k개의 최근접 이웃과의 거리를 저장한 행렬.  
`numpy.ndarray. shape=(N, k), dtype=numpy.float32.`
- retval:          입력 벡터가 하나인 경우에 대한 응답

# K 최근접 이웃 알고리즘

실습: knnplane.py

## ■ KNN 알고리즘 점 분류 예제

```
# 학습 데이터 & 레이블
train = []
label = []
NUM = 30
rn = np.zeros((NUM, 2), np.int32)

def addPoint(x, y, c):
    train.append([x, y])
    label.append([c])

# (150, 150) 근방의 점은 0번 클래스로 설정
cv2.randn(rn, 0, 50)
for i in range(NUM):
    addPoint(rn[i, 0] + 150, rn[i, 1] + 150, 0)
...
```

# K 최근접 이웃 알고리즘

## ■ KNN 알고리즘 점 분류 예제

```
def trainAndDisplay():
    trainData = np.array(train, dtype=np.float32)
    labelData = np.array(label, dtype=np.int32)

    knn.train(trainData, cv2.ml.ROW_SAMPLE, labelData)
```

trainData: shape = (90, 2), dtype=float32  
labelData: shape = (90, ), dtype=int32

```
h, w = img.shape[:2]
for y in range(h):
    for x in range(w):
        sample = np.array([[x, y]]).astype(np.float32)

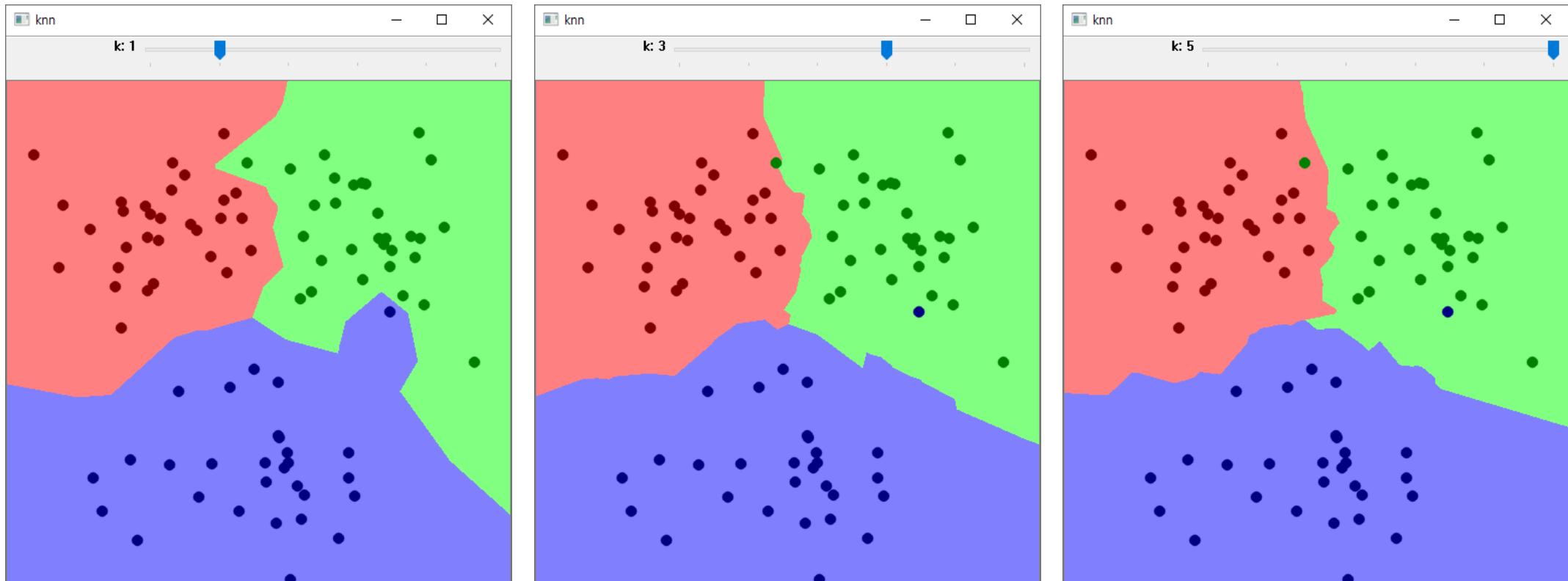
        ret, _, _, _ = knn.findNearest(sample, k_value)
        ret = int(ret)
```

sample에 하나의 데이터만 있으므로  
ret 반환값 사용 가능.

```
if ret == 0:    img[y, x] = (128, 128, 255)
elif ret == 1:   img[y, x] = (128, 255, 128)
elif ret == 2:   img[y, x] = (255, 128, 128)
```

# K 최근접 이웃 알고리즘

## ■ KNN 알고리즘 점 분류 예제 실행 결과



# 11. 머신 러닝

3) k-평균 알고리즘

# K-평균 알고리즘

## ■ k-평균(k-means) 알고리즘

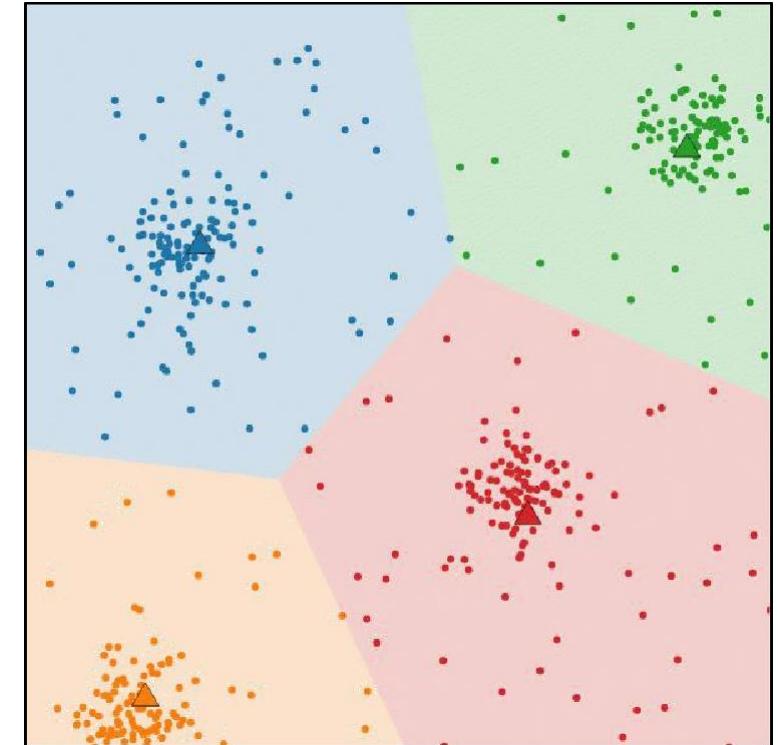
- 주어진 데이터를 k개의 구역으로 나누는 군집화 알고리즘

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

- k: 사용자 지정 파라미터

## ■ 동작 순서

- 임의의 K개 중심을 선정
- 모든 데이터에 대하여 가장 가까운 중심을 선택
- 각 군집에 대해 중심을 다시 계산
- 중심이 변경되면 2~3 과정을 반복
- 그렇지 않으면 종료

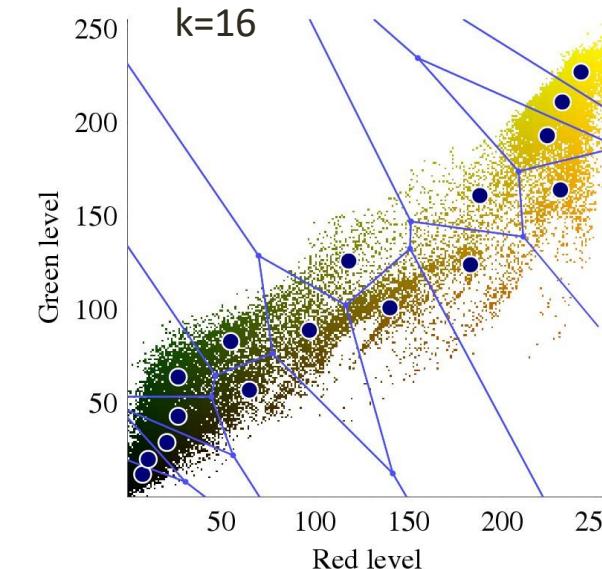


Images from [https://ko.wikipedia.org/wiki/K-%ED%8F%89%EA%B7%A0\\_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98](https://ko.wikipedia.org/wiki/K-%ED%8F%89%EA%B7%A0_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

# K-평균 알고리 즘

## ■ k-평균 알고리즘을 이용한 컬러 영상 분할

- 입력 영상의 각 픽셀 값을 색 공간 상의 한 점으로 표현  
(e.g.) RGB 3차원 공간에서의 한 점, HS 2차원 공간에서의 한 점
- 색 공간에서 k-평균 알고리즘 수행
- 각 픽셀 값을 k개의 대표 색상으로 치환



Images from [https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)

# K-MEAN 알고리즘

## ■ K-mean 군집화 함수

```
cv2.kmeans(data, K, bestLabels, criteria, attempts, flags, centers=None)
           -> retval, bestLabels, centers
```

- data: 학습 데이터 행렬. `numpy.ndarray`. `shape=(N, d)`, `dtype=numpy.float32`.
- K: 군집 개수
- bestLabels: 각 샘플의 군집 번호 행렬. `numpy.ndarray`. `shape=(N, 1)`, `dtype=np.int32`.
- criteria: 종료 기준. (`type`, `maxCount`, `epsilon`) 튜플. 다
- attempts: 른 초기 레이블을 이용해 반복 실행할 횟수.
- flags: 초기 중앙 설정 방법. `cv2.KMEANS_RANDOM_CENTERS`,  
`cv2.KMEANS_PP_CENTERS`, `cv2.KMEANS_USE_INITIAL_LABELS` 중 하나.
- centers: 군집 중심을 나타내는 행렬. `np.ndarray`. `shape=(N, d)`, `dtype=np.float32`.
- retval: Compactness measure =  $\sigma_i \| \text{samples}_i - \text{centers}_{\text{labels}} \|^2$

# K-MEAN 알고리즘

실습: kmeans.py

## ■ k-means 알고리즘을 이용한 컬러 영상 분할 예제

```
src = cv2.imread('flowers.jpg')

data = src.reshape((-1, 3)).astype(np.float32)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

for K in range(2, 9):
    ret, label, center = cv2.kmeans(data, K, None, criteria, 10,
                                    cv2.KMEANS_RANDOM_CENTERS)

    center = np.uint8(center)
    dst = center[label.flatten()] # 각 픽셀을 K개 군집 중심 색상으로 치환
    dst = dst.reshape((src.shape))

cv2.imshow('src', src)
cv2.imshow('dst', dst)
cv2.waitKey()
```

# K-MEAN 알고리즘

- k-means 알고리즘을 이용한 컬러 영상 분할 예제 실행 결과



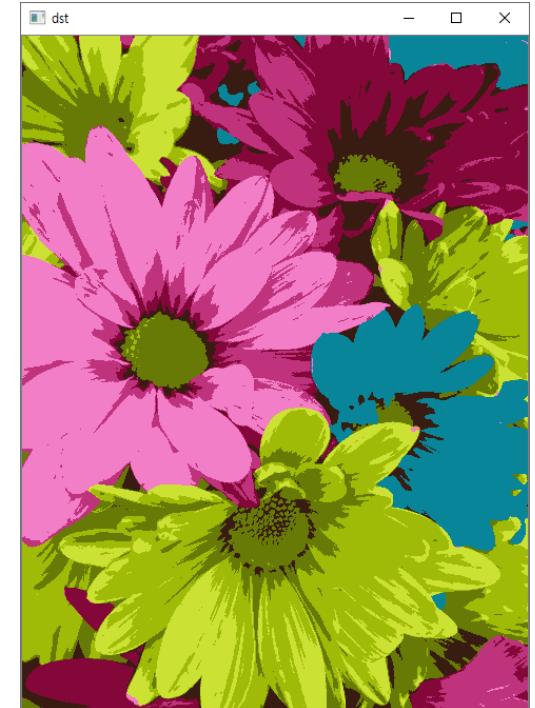
입력 영상



$K = 2$



$K = 5$



$K = 8$

# 12. 딥러닝

1) OpenCV DNN 모듈

# OPENCV DNN 모듈

## ■ OpenCV DNN(Deep Neural Network) 모듈

- 미리 학습된 딥러닝 모델을 이용하여 실행(forward pass, inference) 하는 기능
- 학습은 지원하지 않음
- OpenCV 3.3 버전부터 기본 기능으로 제공
- OpenCV 4.3 버전부터 GPU(CUDA) 지원 (소스 코드 직접 빌드 필요)
- 참고: <https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV>

## ■ 지원하는 딥러닝 프레임워크

Caffe

TensorFlow

torch

Darknet



ONNX

# OPENCV DNN 모듈

## ■ 검증된 딥러닝 네트워크

### [Image classification]

- [AlexNet](#)
- [GoogLeNet](#)
- [VGG](#)
- [ResNet](#)
- [SqueezeNet](#)
- [DenseNet](#)
- [ShuffleNet](#)
- [Inception](#)
- [MobileNet](#)
- [Darknet](#)
- [Other ONNX formats](#)

### [Object detection]

- [SSD VGG](#)
- [MobileNet-SSD](#)
- [Faster-RCNN](#)
- [R-FCN](#)
- [OpenCV face detector](#)
- [SSD, Faster-RCNN and Mask-RCNN](#)
- [EAST](#)
- [YOLOv2, tiny YOLO, YOLOv3](#)

### [Semantic segmentation]

- [FCN](#)
- [ENet](#)

### [Pose estimation]

- [OpenPose](#)

### [Image processing]

- [Colorization](#)
- [Fast-Neural-Style](#)

### [Person identification]

- [OpenFace](#)

# OPENCV DNN API

## ■ 네트워크 불러오기

```
cv2.dnn.readNet(model, config=None, framework=None) -> retval
```

- model: 훈련된 가중치를 저장하고 있는 이진 파일 이름
- config: 네트워크 구성을 저장하고 있는 텍스트 파일 이름
- framework: 명시적인 딥러닝 프레임워크 이름
- retval: cv2.dnn\_Net 클래스 객체

딥러닝 프레임워크	model 파일 확장자	config 파일 확장자	framework 문자열
카페	*.caffemodel	*.prototxt	"caffe"
텐서플로우	*.pb	*.pbtxt	"tensorflow"
토치	*.t7 또는 *.net		"torch"
다크넷	*.weights	*.cfg	"darknet"
DLDT	*.bin	*.xml	"dldt"
ONNX	*.onnx		"onnx"

# OPENCV DNN API

## ■ 네트워크 입력 블롭(blob) 만들기

```
cv2.dnn.blobFromImage(image, scalefactor=None, size=None, mean=None,  
                      swapRB=None, crop=None, ddepth=None) -> retval
```

- image: 입력 영상
- scalefactor: 입력 영상 픽셀 값에 곱할 값. 기본값은 1.
- size: 출력 영상의 크기. 기본값은 (0, 0).
- mean: 입력 영상 각 채널에서 뺄 평균 값. 기본값은 (0, 0, 0, 0).
- swapRB: R과 B 채널을 서로 바꿀 것인지를 결정하는 플래그. 기본값은 False.
- crop: 크롭(crop) 수행 여부. 기본값은 False.
- ddepth: 출력 블롭의 깊이. CV\_32F 또는 CV\_8U. 기본값은 CV\_32F.
- retval: 영상으로부터 구한 블롭 객체.  
`numpy.ndarray. shape=(N,C,H,W). dtype=numpy.float32.`

# OPENCV DNN API

## ■ 네트워크 입력 설정하기

```
cv2.dnn_Net.setInput(blob, name=None, scalefactor=None, mean=None) -> None
```

- blob: 블롭 객체
- name: 입력 레이어 이름
- scalefactor: 추가적으로 픽셀 값에 곱할 값
- mean: 추가적으로 픽셀 값에서 뺄 평균 값

# OPENCV DNN API

## ■ 네트워크 순방향 실행 (추론)

```
cv2.dnn_Net.forward(outputName=None) -> retval  
cv2.dnn_Net.forward(outputNames=None, outputBlobs=None) -> outputBlobs
```

- `outputName`: 출력 레이어 이름
- `retval`: 지정한 레이어의 출력 블롭. 네트워크마다 다르게 결정됨.
- `outputNames`: 출력 레이어 이름 리스트
- `outputBlobs`: 지정한 레이어의 출력 블롭 리스트

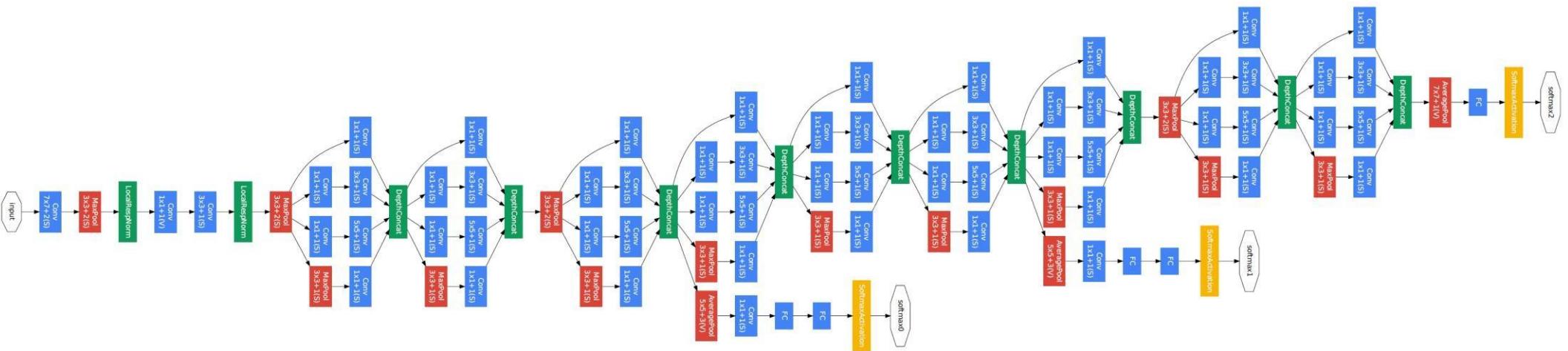
# 12. 딥러닝

2) GoogLeNet 영상 인식

# GOOGLENET 영상 인식

## ■ GoogLeNet 영상 인식

- 2014년 ILSVRC(ImageNet Large Scale Visual Recognition Competition) 영상 인식 분야 1위
    - 1000개의 카테고리, 120만개의 훈련 영상, 15만개의 테스트 영상
  - 입력: 224x224, BGR 컬러 영상, 평균 값 = (104, 117, 123)
  - 출력: 1x1000 행렬, 1000개 클래스에 대한 확률 값



<https://arxiv.org/pdf/1409.4842>

# GOOGLENET 영상 인식

## ■ 미리 학습된 GoogLeNet 학습 모델 및 구성 파일 다운로드

- Caffe Model Zoo: <https://github.com/BVLC/caffe>
  - 모델 파일: [http://dl.caffe.berkeleyvision.org/bvlc\\_googlenet.caffemodel](http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel)
  - 설정 [https://github.com/BVLC/caffe/blob/master/models/bvlc\\_googlenet/deploy.prototxt](https://github.com/BVLC/caffe/blob/master/models/bvlc_googlenet/deploy.prototxt)
- ONNX model zoo: <https://github.com/onnx/models>
  - 모델 파일: [https://github.com/onnx/models/tree/master/vision/classification/inception\\_and\\_googlenet/googlenet](https://github.com/onnx/models/tree/master/vision/classification/inception_and_googlenet/googlenet)
- 클래스 이름파일:
  - 1~1000번 클래스에 대한 설명을 저장한 텍스트 파일
  - [https://github.com/opencv/opencv/blob/4.1.0/samples/data/dnn/classification\\_classes\\_ILSVRC2012.txt](https://github.com/opencv/opencv/blob/4.1.0/samples/data/dnn/classification_classes_ILSVRC2012.txt)

# GOOGLENET 영상 인식

실습: classify.py

## ■ 구글넷 영상 인식 예제

```
# Load input image
filename = 'space_shuttle.jpg'
if len(sys.argv) > 1:
    filename = sys.argv[1]
    명령행 인자 지원.

img = cv2.imread(filename)

# Load network
model = 'googlenet/bvlc_googlenet.caffemodel'
config = 'googlenet/deploy.prototxt'
#model = 'googlenet/inception-v1-9.onnx'
#config = ''

net = cv2.dnn.readNet(model, config)
    네트워크 불러오기

# Load class names
classNames = None
with open('classification_classes_ILSVRC2012.txt', 'rt') as f:
    classNames = f.read().rstrip('\n').split('\n')
```

# GOOGLENET 영상 인식

## ■ 구글넷 영상 인식 예제 (Con't)

```
# Inference
```

```
blob = cv2.dnn.blobFromImage(img, 1, (224, 224), (104, 117, 123))
net.setInput(blob, 'data')
prob = net.forward()
```

```
# Check results & Display
```

blob: shape=(1, 3, 224, 224), dtype=float32  
prob: shape=(1, 1000), dtype=float32.

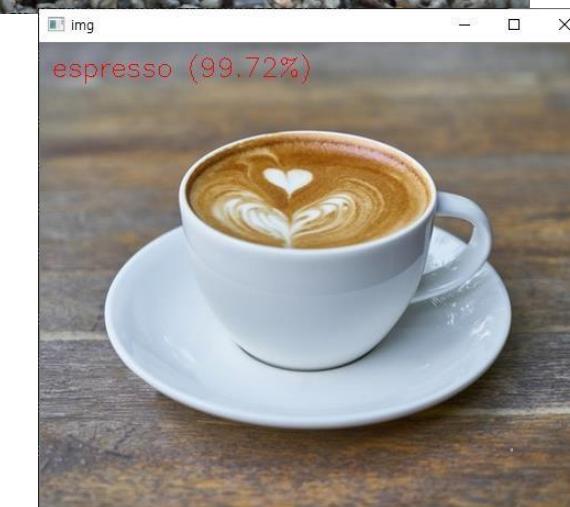
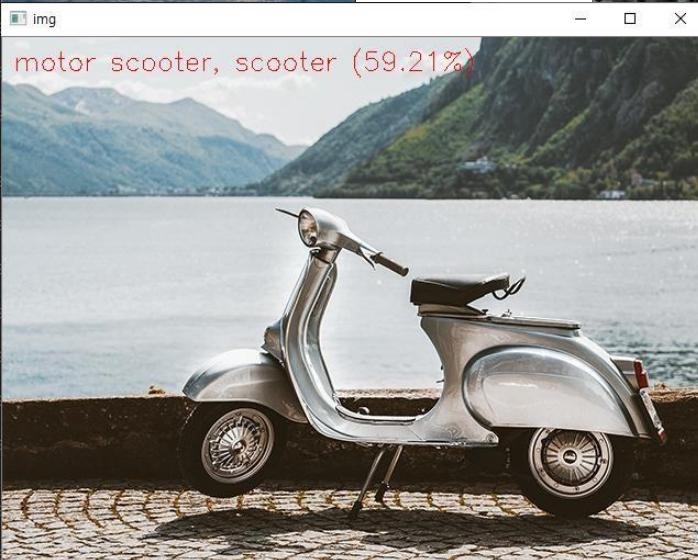
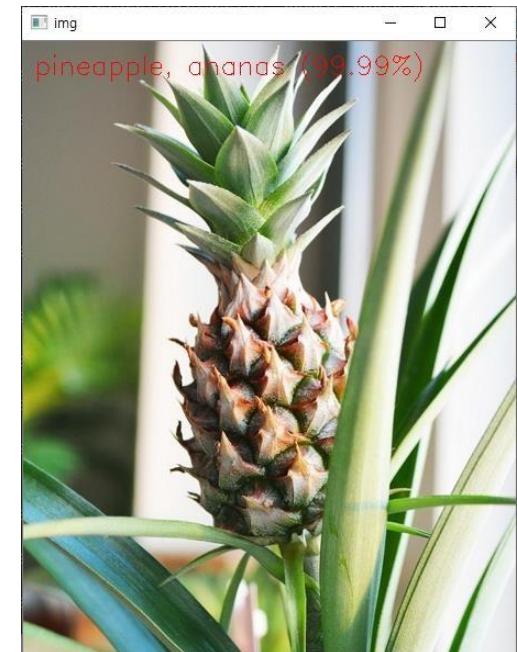
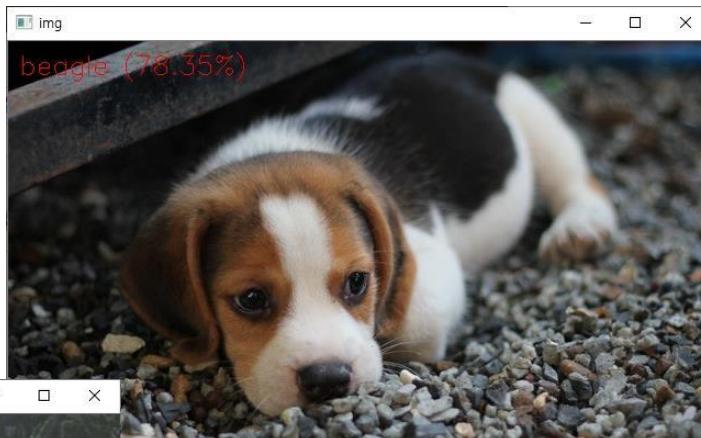
```
out = prob.flatten()
classId = np.argmax(out)
confidence = out[classId]
```

```
text = f'{classNames[classId]} ({confidence * 100:.2f}%)'
cv2.putText(img, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255),
           1, cv2.LINE_AA)
```

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

# GOOGLENET 영상 인식

## ■ 구글넷 영상 인식 예제 실행 결과



# 13. 딥러닝 활용: 객체 검출

1) OpenCV DNN 얼굴 검출

# OPENCV DNN 얼굴 검출

## ■ OpenCV DNN 얼굴 검출 예제

- OpenCV 예제에서 DNN 모듈을 사용한 얼굴 검출 기능을 지원
  - SSD(Single Shot MultiBox Detector) 기반 얼굴 검출 네트워크
  - [https://github.com/opencv/opencv/tree/master/samples/dnn/face\\_detector](https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector)
- 기존의 Haar-Cascade 방법보다 속도 & 정확도 면에서 더 좋은 성능을 나타냄

	Haar Cascade	DL
Size on disk	528KB	10MB (fp32), 5MB (fp16)
Efficiency @ 300x300**	30 ms	9.34 ms
Performance AP @ IoU = 0.5*	0.609 (FDDB) 0.149 (WIDER FACE, val.)	0.797 (FDDB) 0.173 (WIDER FACE, val.)

\*PASCAL VOC metric using COCO evaluation tool, <http://cocodataset.org/#detections-eval>

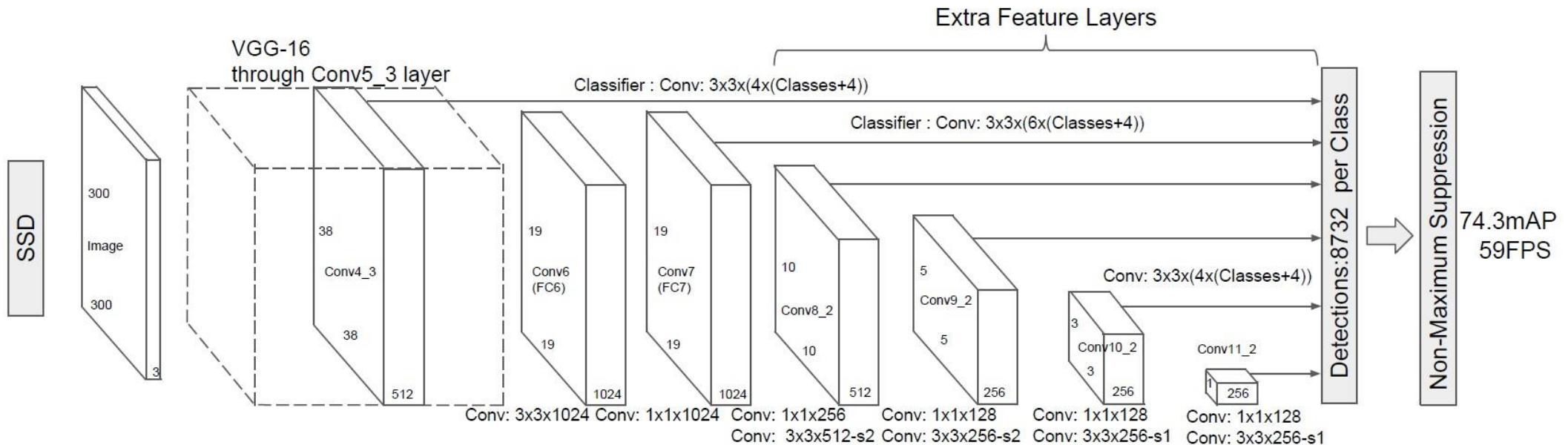
\*\*Intel® Core™ i5-4460 CPU @ 3.20GHz x 4

Table from [http://dl.opencv.org/present/cvpr\\_opencv.pdf](http://dl.opencv.org/present/cvpr_opencv.pdf)

# OPENCV DNN 얼굴 검출

## ■ SSD(Single Shot MultiBox Detector) (W. Liu, et. al, 2016)

- 동시대 다른 객체 검출 알고리즘과 비교하여 성능과 속도 두 가지를 모두 만족시킨 알고리즘
    - Faster R-CNN: 73.2 mAP, 7 FPS
    - YOLOv1: 63.4 mAP, 45 FPS
- SSD: 74.3 mAP, 59 FPS



<https://arxiv.org/pdf/1512.02325.pdf>

# OPENCV DNN 얼굴 검출

## ■ OpenCV Face 검출 모델 & 설정 파일 다운로드

- 모델 파일
  - OpenCV 제공 스크립트 사용 방법
    - [https://github.com/opencv/opencv/tree/master/samples/dnn/face\\_detector](https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector)에서 파일 다운로드 후 `download_weights.py` 파일 실행
  - 모델 파일 직접 다운로드
    - Caffe(FP16): [https://raw.githubusercontent.com/opencv/opencv\\_3rdparty/dnn\\_samples\\_face\\_detector\\_20180205\\_fp16/res10\\_300x300\\_ssd\\_iter\\_140000\\_fp16.caffemodel](https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20180205_fp16/res10_300x300_ssd_iter_140000_fp16.caffemodel)  
Not exist
    - TensorFlow(uint8): [https://raw.githubusercontent.com/opencv/opencv\\_3rdparty/dnn\\_samples\\_face\\_detector\\_20180220\\_uint8/opencv\\_face\\_detector\\_uint8.pb](https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20180220_uint8/opencv_face_detector_uint8.pb)  
Only this file works
- 구성 파일 다운로드
  - [https://github.com/opencv/opencv/tree/master/samples/dnn/face\\_detector](https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector)에서 `deploy.prototxt`, `opencv_face_detector.pbtxt` 파일 다운로드
- 다운로드 받은 파일을 `c:\coding\python\opencv\ch13\opencv_face_detector\` 폴더에 저장

# OPENCV DNN 얼굴 검출

## ■ OpenCV DNN 얼굴 검출(SSD) 입력

- Size: (300, 300)
- Scale: 1 (0 ~ 255)
- Mean: (104, 177, 123)
- RGB: false

## ■ OpenCV DNN 얼굴 검출(SSD) 입력

- `out.shape=(1, 1, 200, 7)`
- `detect = out[0, 0, :, :]`



0	1	c	x1	y1	x2	y2
0	1	c	x1	y1	x2	y2
0	1	c	x1	y1	x2	y2
:	:	:	:	:	:	:

# OPENCV DNN 얼굴 검출

실습: face\_detect.py

## ■ OpenCV DNN 얼굴 검출 예제

```
model = 'opencv_face_detector/res10_300x300_ssd_iter_140000_fp16.caffemodel'
config = 'opencv_face_detector/deploy.prototxt'
#model = 'opencv_face_detector/opencv_face_detector_uint8.pb'
#config = 'opencv_face_detector/opencv_face_detector.pbtxt'

cap = cv2.VideoCapture(0)
net = cv2.dnn.readNet(model, config)

while True:
    _, frame = cap.read()

    if frame is None:
        break

    blob = cv2.dnn.blobFromImage(frame, 1, (300, 300), (104, 177, 123))
    net.setInput(blob)

    detect = net.forward()
    detect = detect[0, 0, :, :]
```

detect.shape=(1, 1, N, 7)이고, 이중 뒤쪽 두 개의 차원에 검출  
정보가 저장됨. 그러므로 편의상 2차원 행렬로 변환하여 사용.

# OPENCV DNN 얼굴 검출

## ■ OpenCV DNN 얼굴 검출 예제

```
(h, w) = frame.shape[:2]

for i in range(detect.shape[0]):
    confidence = detect[i, 2]
    if confidence < 0.5:
        break

    x1 = int(detect[i, 3] * w)
    y1 = int(detect[i, 4] * h)
    x2 = int(detect[i, 5] * w)
    y2 = int(detect[i, 6] * h)

    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0))
    label = f'Face: {confidence:.2f}'
    cv2.putText(frame, label, (x1, y1 - 10),
               cv2.FONT_HERSHEY_SIMPLEX,
               0.8, (0, 255, 0), 1, cv2.LINE_AA)

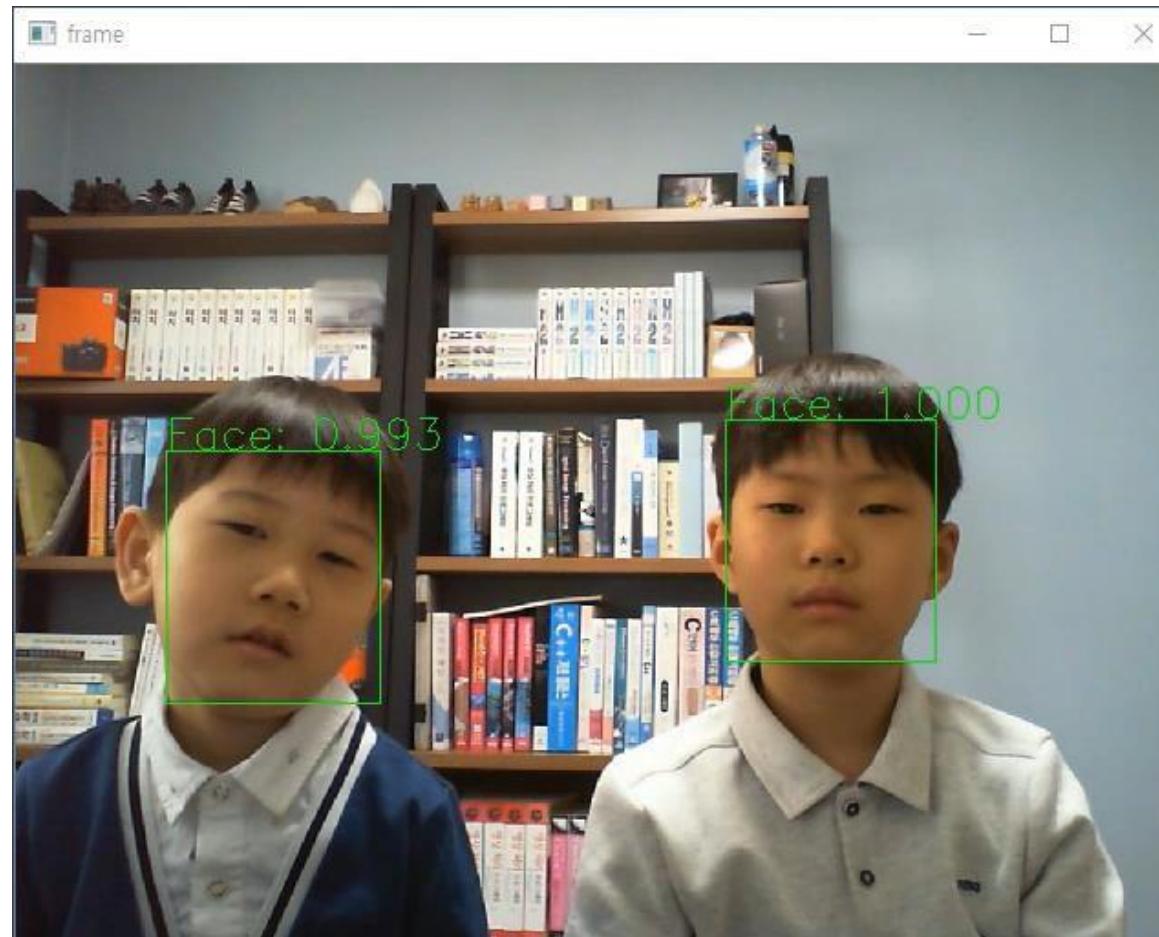
cv2.imshow('frame', frame)
if cv2.waitKey(1) == 27:
    break
```

detect:

0	1	c	x1	y1	x2	y2
0	1	c	x1	y1	x2	y2
0	1	c	x1	y1	x2	y2
:	:	:	:	:	:	:

# OPENCV DNN 얼굴 검출

- OpenCV DNN 얼굴 검출 예제 실행 결과



# 13. 딥러닝 활용: 객체 검출

2) Mask-RCNN 영역 분할

# MASK-RCNN 영역 분할

## ■ 영역 분할이란?

- 객체의 바운딩 박스뿐만 아니라 픽셀 단위 클래스 분류까지 ⑦ 객체의 윤곽 구분
- Semantic segmentation: 하나의 클래스는 모두 같은 레이블
- Instance segmentation: 객체 단위 다른 레이블

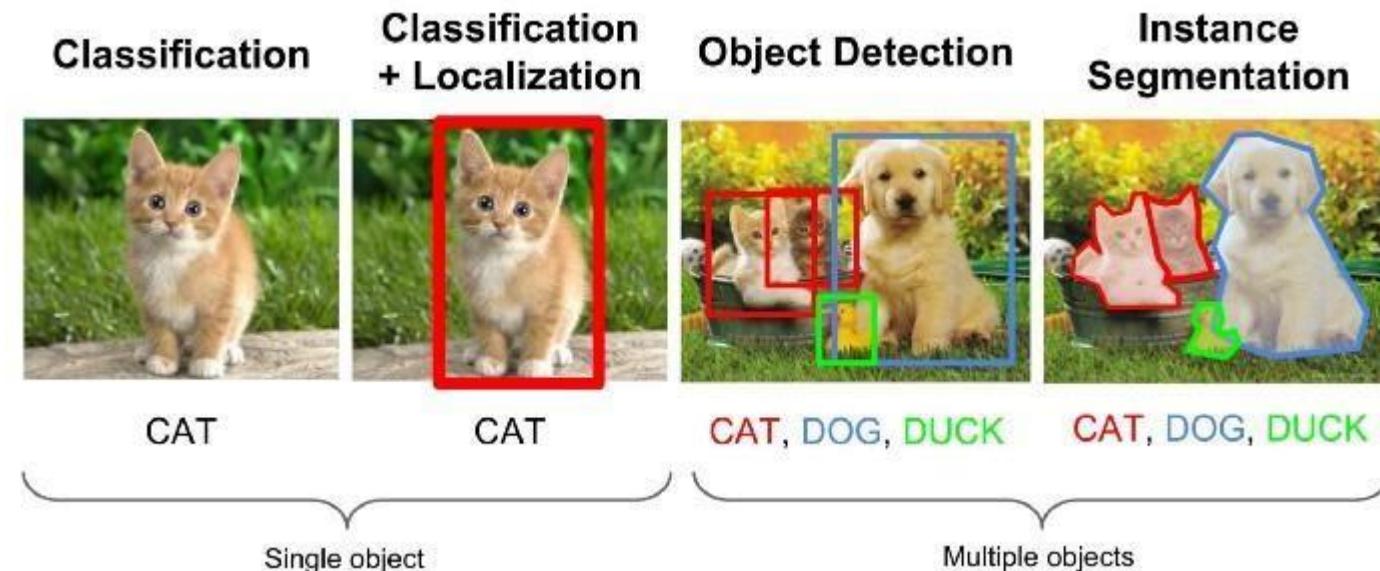


Image from <https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>

# MASK-RCNN 영역 분할

## ■ Mask-RCNN이란?

- 대표적인 객체 영역 분할 딥러닝 알고리즘 (He et. al., 2017)
- Faster R-CNN (object detection) + FCN (semantic segmentation)

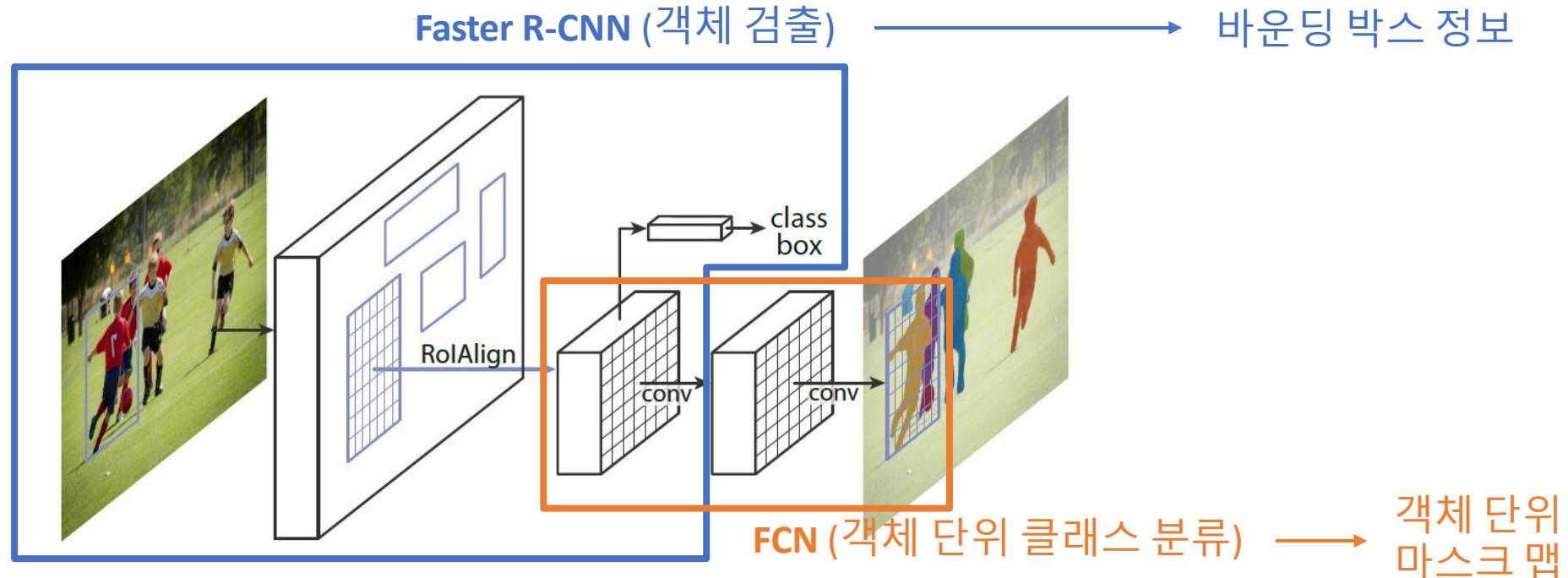


Image from <https://arxiv.org/pdf/1703.06870>

# MASK-RCNN 영역 분할

## ■ Mask-RCNN 모델 & 설정 파일 다운로드

- 모델 파일

- [http://download.tensorflow.org/models/object\\_detection/mask\\_rcnn\\_inception\\_v2\\_coco\\_2018\\_01\\_28.tar.gz](http://download.tensorflow.org/models/object_detection/mask_rcnn_inception_v2_coco_2018_01_28.tar.gz)

- 설정 파일

- [https://github.com/opencv/opencv\\_extra/blob/master/testdata/dnn/mask\\_rcnn\\_inception\\_v2\\_coco\\_2018\\_01\\_28.pbtxt](https://github.com/opencv/opencv_extra/blob/master/testdata/dnn/mask_rcnn_inception_v2_coco_2018_01_28.pbtxt)

- 클래스 이름파일

- c:\coding\python\opencv\ch13\mask\_rcnn\coco\_90.names

- frozen\_inference\_graph.pb, mask\_rcnn\_inception\_v2\_coco\_2018\_01\_28.pbtxt 파일을 c:\coding\python\opencv\ch13\mask\_rcnn\ 폴더에 저장

# MASK-RCNN 영역 분할

## ■ Mask-RCNN 입력

- Size: 임의의 크기 (auto resize)
- Scale: 1 (1 ~ 255)
- Mean: [0, 0, 0]
- RGB: true

## ■ Mask-RCNN 출력

- 2개의 출력 레이어 사용
  - 'detection\_out\_final' ⑦ boxes: boxes.shape=(1, 1, 100, 7)
  - 'detection\_masks' ⑦ masks: masks.shape=(100, 90, 15, 1)

# MASK-RCNN 영역 분할

실습: mask\_rcnn.py

## ■ Mask-RCNN 영역 분할 예제

```
# 모델 & 설정 파일
model = 'mask_rcnn/frozen_inference_graph.pb'
config = 'mask_rcnn/mask_rcnn_inception_v2_coco_2018_01_28.pbtxt'
class_labels = 'mask_rcnn/coco_90.names'
confThreshold = 0.6
maskThreshold = 0.3

# 네트워크 생성
net = cv2.dnn.readNet(model, config)

img = cv2.imread(f)

# 블롭 생성 & 추론
blob = cv2.dnn.blobFromImage(img, swapRB=True)
net.setInput(blob)
boxes, masks = net.forward(['detection_out_final', 'detection_masks'])

# boxes.shape=(1, 1, 100, 7)
# masks.shape=(100, 90, 15, 15)
```

# MASK-RCNN 영역 분할

## ■ Mask-RCNN 영역 분할 예제

```
numClasses = masks.shape[1] # 90
numDetections = boxes.shape[2] # 100

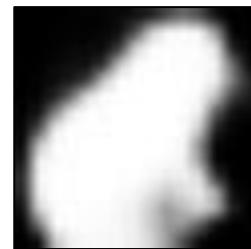
boxesToDraw = []
for i in range(numDetections):
    box = boxes[0, 0, i] # box.shape=(7,)
    mask = masks[i] # mask.shape=(90, 15, 15)
    score = box[2]
    if score > confThreshold:
        classId = int(box[1])
        classMask = mask[classId]

    # 바운딩 박스 그리기 & 마스크 영역 그리기
```

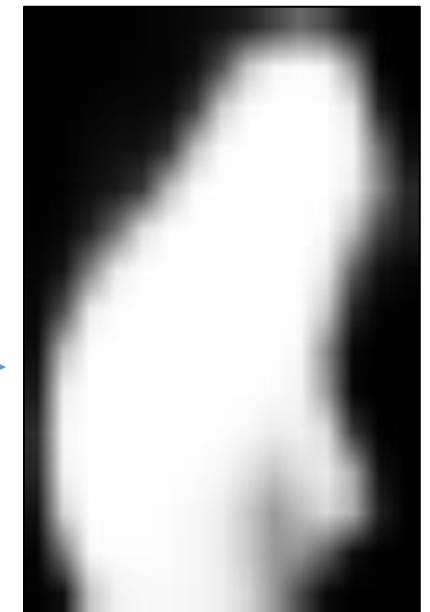
box

0	classId	conf	x1	y1	x2	y2
---	---------	------	----	----	----	----

classMask



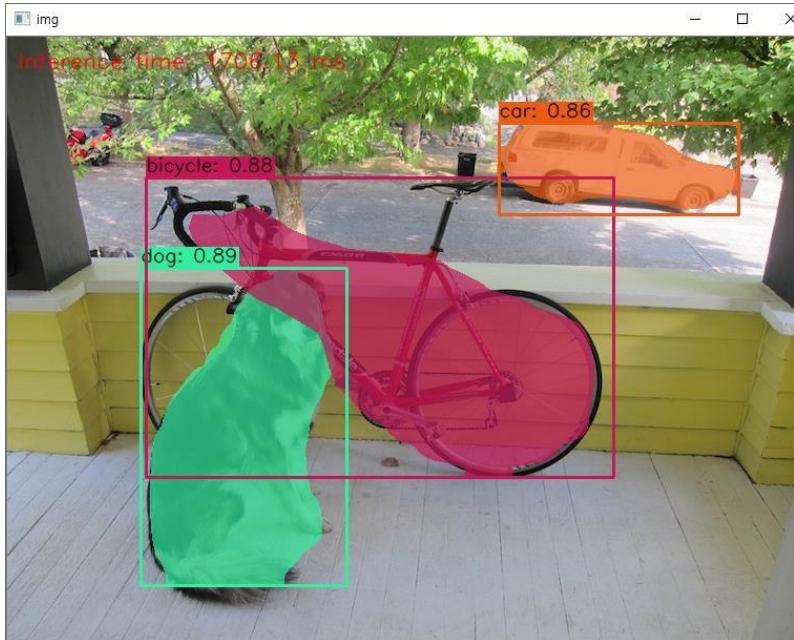
15x15



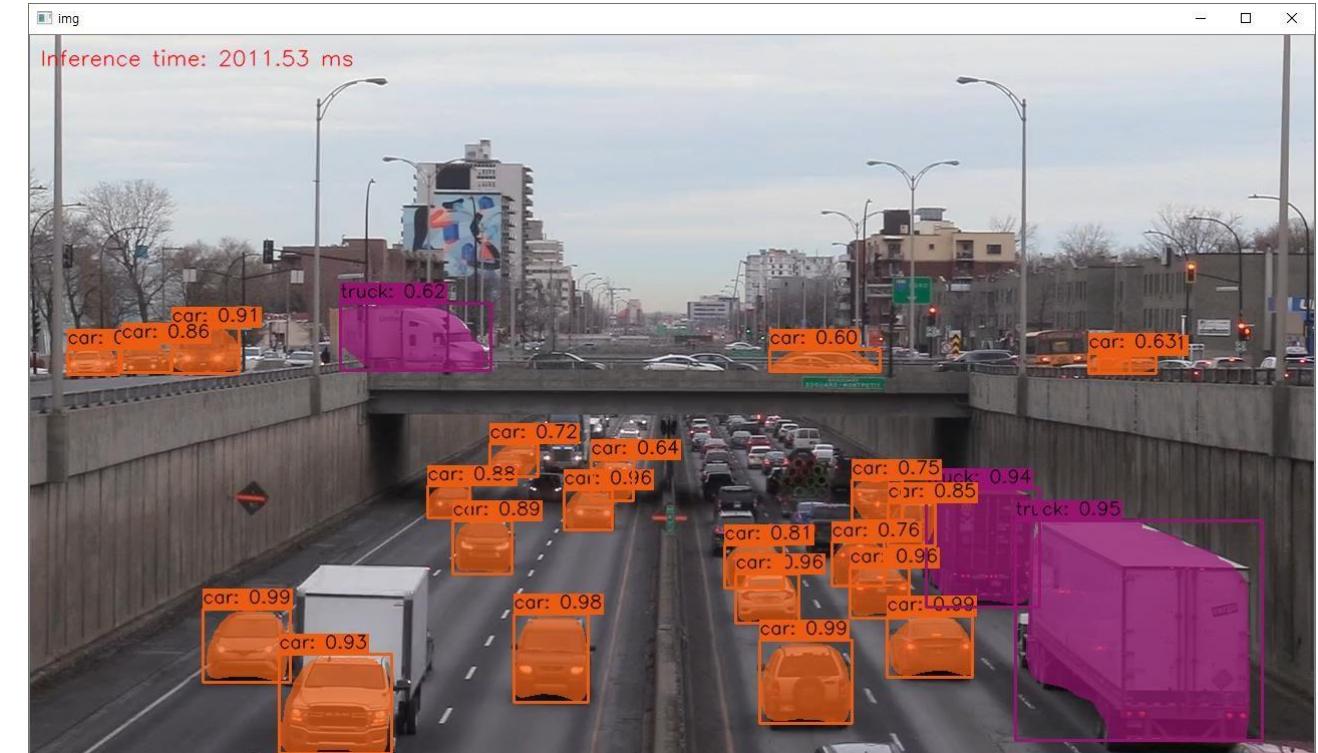
197x303

# MASK-RCNN 영역 분할

## ■ Mask-RCNN 영역 분할 예제 실행 결과



17 dog 0.8920537  
1 bicycle 0.8750928  
2 car 0.8585277



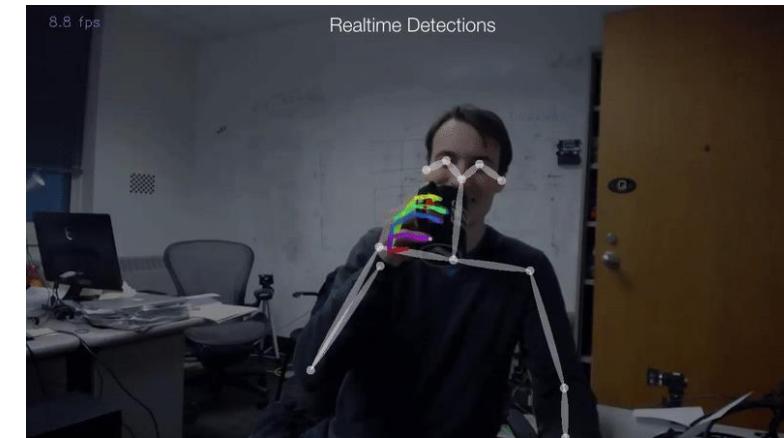
# 13. 딥러닝 활용: 객체 검출

3) OpenPose: 포즈 인식

# OPENPOSE: 포즈 인식

## ■ OpenPose란?

- 카네기 멜론 대학에서 만든 딥러닝 기반 동작 인식 라이브러리(CVPR, 2017)
- OpenPose represents the first real-time multi-person system to jointly detect **human body, hand, facial, and foot keypoints** (in total 135 keypoints) on single images.
- Caffe, OpenCV, C++
- <https://github.com/CMU-Perceptual-Computing-Lab/openpose>



# OPENPOSE: 포즈 인식

## ■ OpenPose 네트워크 구조

- **F**: Feature map
- **L**: Set of 2D vector fields of part affinity (limb)
- **S**: Set of 2D confidence maps of body part locations (elbow, knee, etc.)

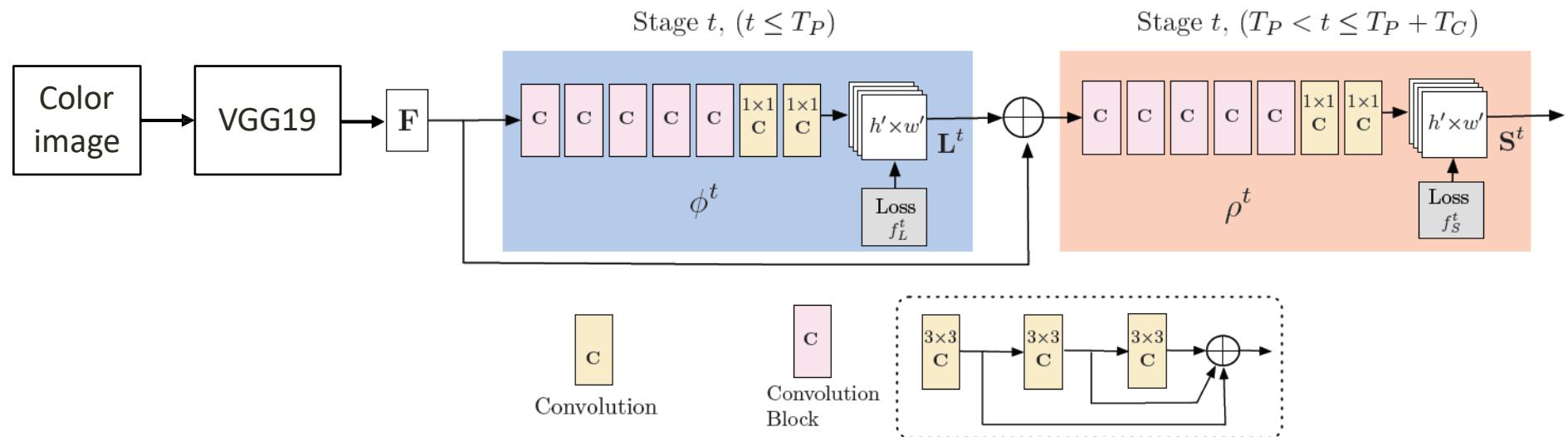
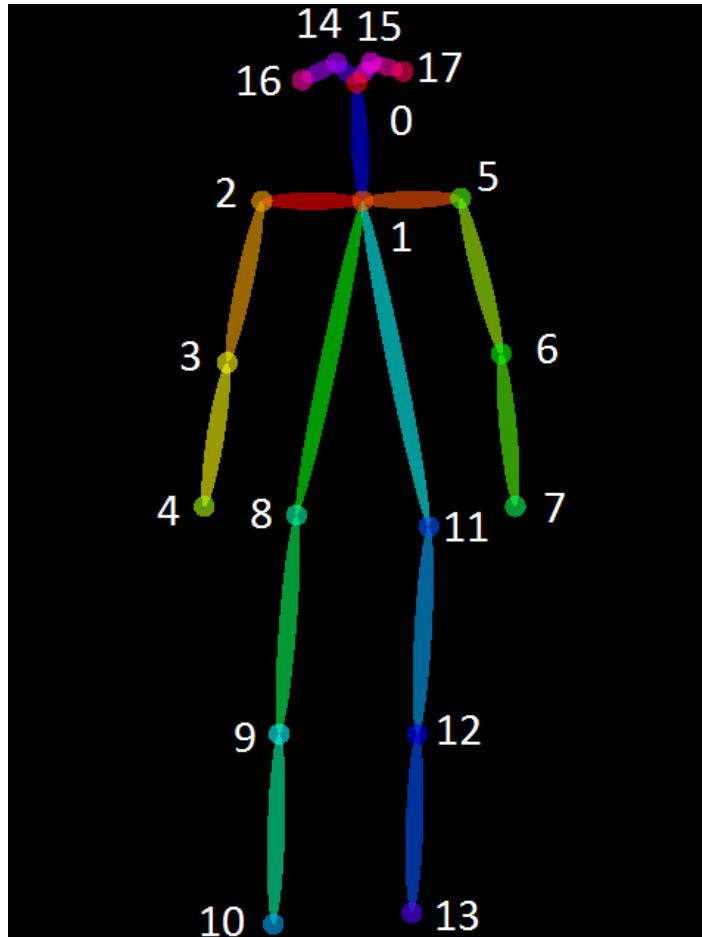


Image from <https://arxiv.org/pdf/1812.08008>

# OPENPOSE: 포즈 인식

- OpenPose 학습 데이터셋: COCO (18 parts)



- |            |            |
|------------|------------|
| 0: 코       | 11: 왼쪽 엉덩이 |
| 1: 목       | 12: 왼쪽 무릎  |
| 2: 오른쪽 어깨  | 13: 왼쪽 발목  |
| 3: 오른쪽 팔꿈치 | 14: 오른쪽 눈  |
| 4: 오른쪽 손목  | 15: 왼쪽 눈   |
| 5: 왼쪽 어깨   | 16: 오른쪽 귀  |
| 6: 왼쪽 팔꿈치  | 17: 왼쪽 귀   |
| 7: 왼쪽 손목   | 18: 배경     |
| 8: 오른쪽 엉덩이 |            |
| 9: 오른쪽 무릎  |            |
| 10: 오른쪽 발목 |            |

# OPENPOSE: 포즈 인식

## ■ OpenPose (COCO, 18 parts) 모델 & 설정 파일 다운로드

- 모델 파일: [http://posefs1.perception.cs.cmu.edu/OpenPose/models/pose/coco/  
pose\\_iter\\_440000.caffemodel](http://posefs1.perception.cs.cmu.edu/OpenPose/models/pose/coco/pose_iter_440000.caffemodel)
- 설정 파일: [https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/  
models/pose/coco/pose\\_deploy\\_linevec.prototxt](https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/models/pose/coco/pose_deploy_linevec.prototxt)
- 다운로드 받은 파일을 c:\coding\python\opencv\ch13\openpose\ 폴더에 복사
- MPI, Face, Hand 등의 모델은 <https://github.com/CMU-Perceptual-Computing-Lab/openpose>에서 getModels.bat 파일 참고

# OPENPOSE: 포즈 인식

## ■ OpenPose 입력

- Size: (368, 368)
- Scale: 0.00392 (1/255.)
- Mean: [0, 0, 0]
- RGB: false



[Keypoint confidence Maps ]

## ■ OpenPose 출력 (COCO)

- `out.shape=(1, 57, 46, 46)`
- $57 = 18 \text{ keypoint confidence Maps}$ 
  - + 1 background
  - +  $19 * 2$  Part Affinity Maps



[Part Affinity Maps]

Image from <https://www.learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/>

# OPENPOSE: 포즈 인식

실습: openpose.py

## ■ OpenPose 포즈 검출 예제

```
# 모델 & 설정 파일
model = 'open_pose/pose_iter_440000.caffemodel'
config =
'open_pose/pose_deploy_linevec.prototxt'

# 포즈 점 개수, 점 연결 개수, 연결 점 번호 쌍
nparts = 18
pose_pairs = [(1, 2), (2, 3), (3, 4),
(1, 5), (5, 6), (6, 7), # 왼팔
(1, 8), (8, 9), (9, 10), # 오른팔
# 왼쪽다리
(1, 11), (11, 12), (12, 13), # 오른쪽다리
(1, 0), (0, 14), (14, 16), (0, 15), (15, 17)] # 얼굴

img = cv2.imread('pose1.jpg')
net = cv2.dnn.readNet(model, config)
blob = cv2.dnn.blobFromImage(img, 1/255., (368, 368))
net.setInput(blob)
out = net.forward() # out.shape=(1, 19, 46, 46)
```

blob.shape=(1, 3, 368, 368)  
blob.dtype=float32  
out.shape=(1, 57, 46, 46)  
out.dtype=float32

# OPENPOSE: 포즈 인식

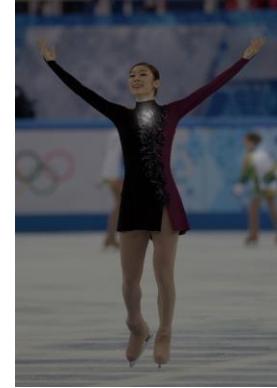
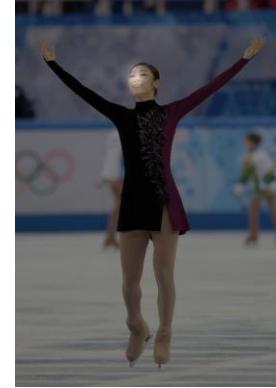
```
# 검출된 점 추출
points = []
for i in range(nparts):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv2.minMaxLoc(heatMap)
    x = int(w * point[0]) // out.shape[3]
    y = int(h * point[1]) // out.shape[2]

    points.append((x, y) if conf > 0.1 else None)

# 검출 결과 영상 만들기
for pair in pose_pairs:
    p1 = points[pair[0]]
    p2 = points[pair[1]]

    if p1 is None or p2 is None:
        continue

    cv2.line(img, p1, p2, (0, 255, 0), 3, cv2.LINE_AA)
    cv2.circle(img, p1, 4, (0, 0, 255), -1,
               cv2.LINE_AA)
    cv2.circle(img, p2, 4, (0, 0, 255), -1,
               cv2.LINE_AA)
```



0번 heatmap  
(코)

1번 heatmap  
(목)

points에 18개 점 좌표가 저장됨  
(검출되지 않은 점은 None)

# OPENPOSE: 포즈 인식

## ■ OpenPose 포즈 검출 예제 실행 결과

