

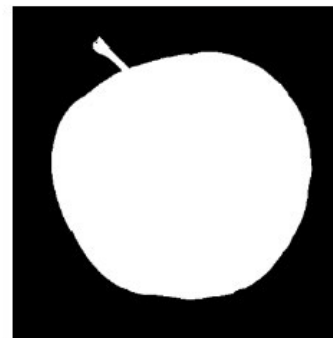
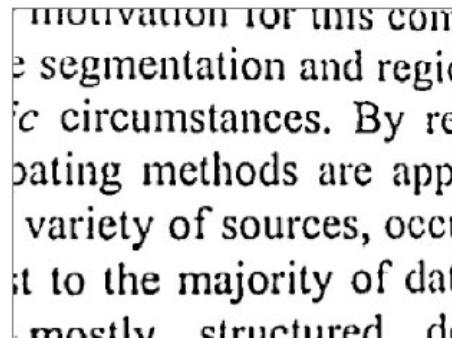
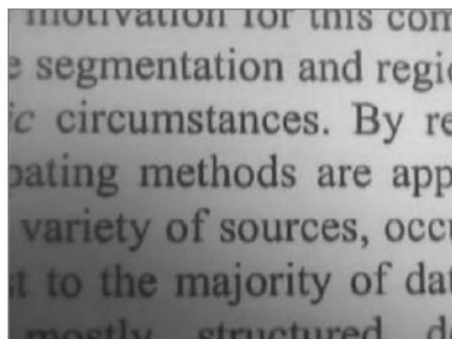
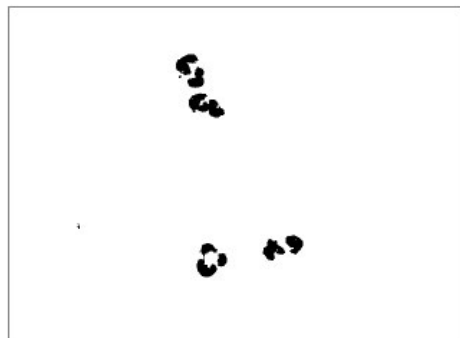
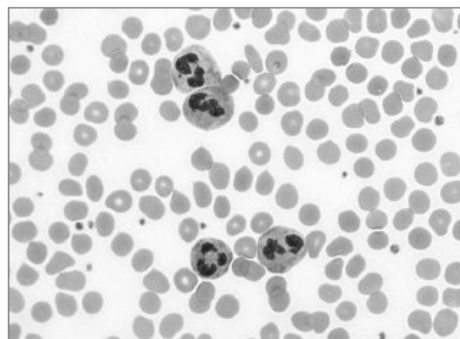
OpenCV-Python

영상 분할과 객체 검출

멀티미디어

영상의 이진화

- 영상의 이진화(Binarization)이란?
 - 영상(이미지)의 픽셀 값을 0 또는 255로 만드는 연산
 - 배경 vs 객체
 - 관심영역 vs 비관심 영역

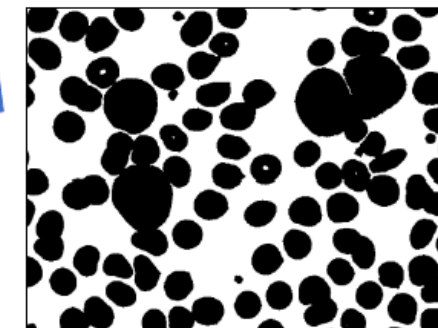
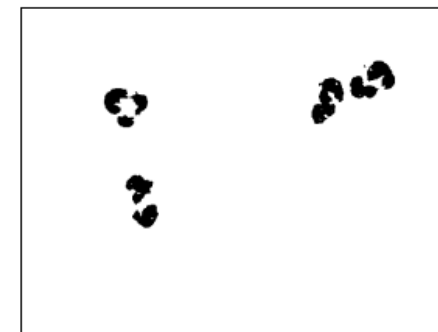
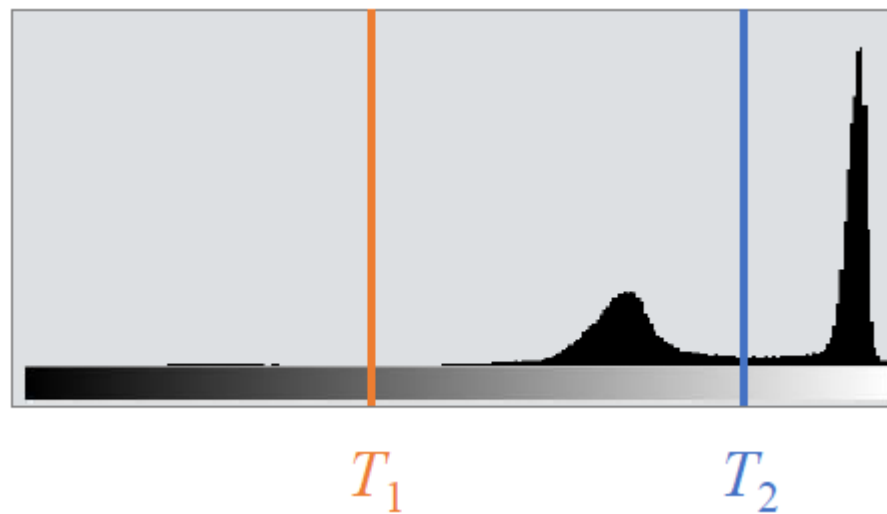
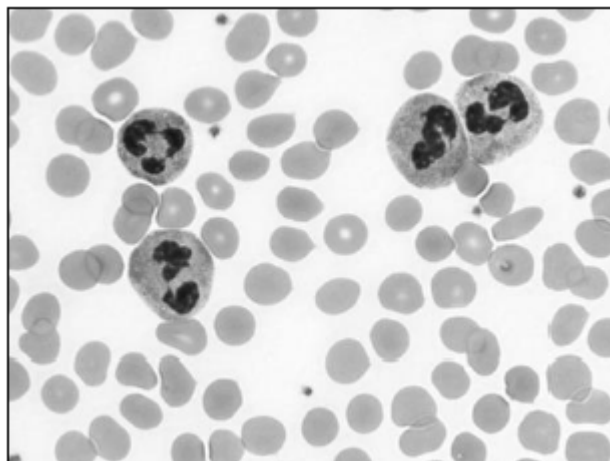


영상의 이진화

- 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

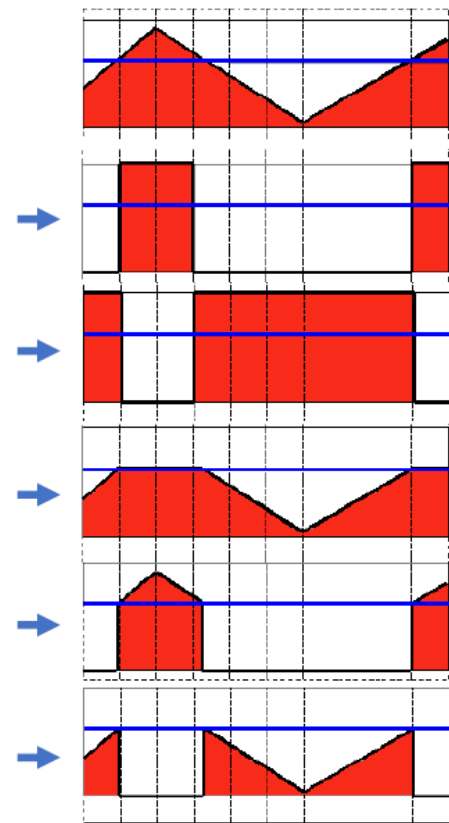
```
cv2.threshold(src, thresh, maxval, type, dst=None) -> retval, dst
```

- Src: 입력 이미지, 다채널, 8비트 또는 32비트 실수형
- Thresh : 사용자 지정 임계값
- Maxval : cv2.THRESH_BINARY 또는 cv2.THRESH_BINARY_INV 최댓값, 보통 255로 지정
- Type: 함수동작 타입, cv2.THRESHY_로 시작하는 플래그
- Retval : 사용된 임계값
- Dft : 출력 이미지. Src와 동일한 크기, 동일 타입, 같은 채널수.

영상의 이진화

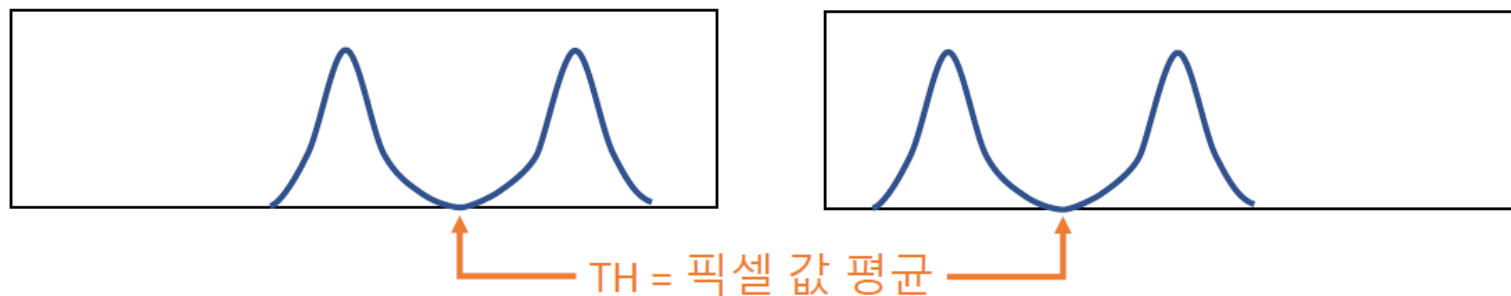
- Cv2.threshold() 함수 동작 타입

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

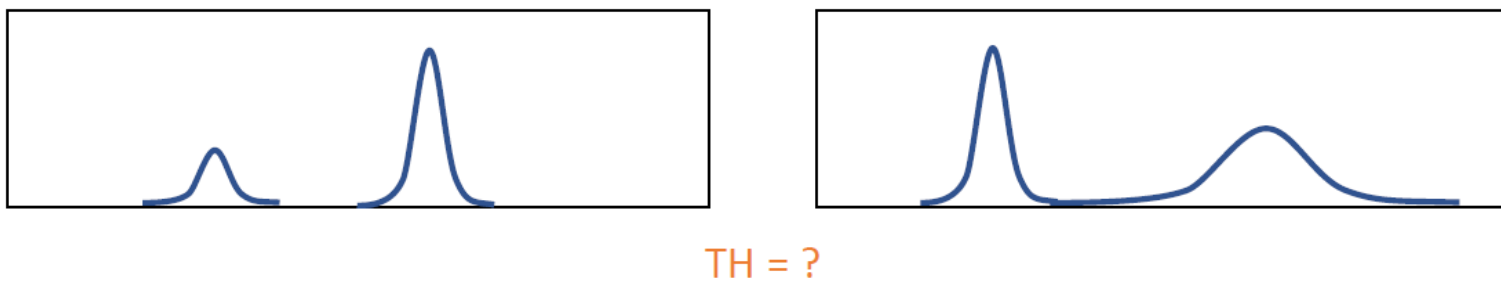


자동 이진화

- 임계값 자동 결정 방법
 - 영상 이미지의 히스토그램의 전경, 배경 픽셀 분포가 비슷하다면?

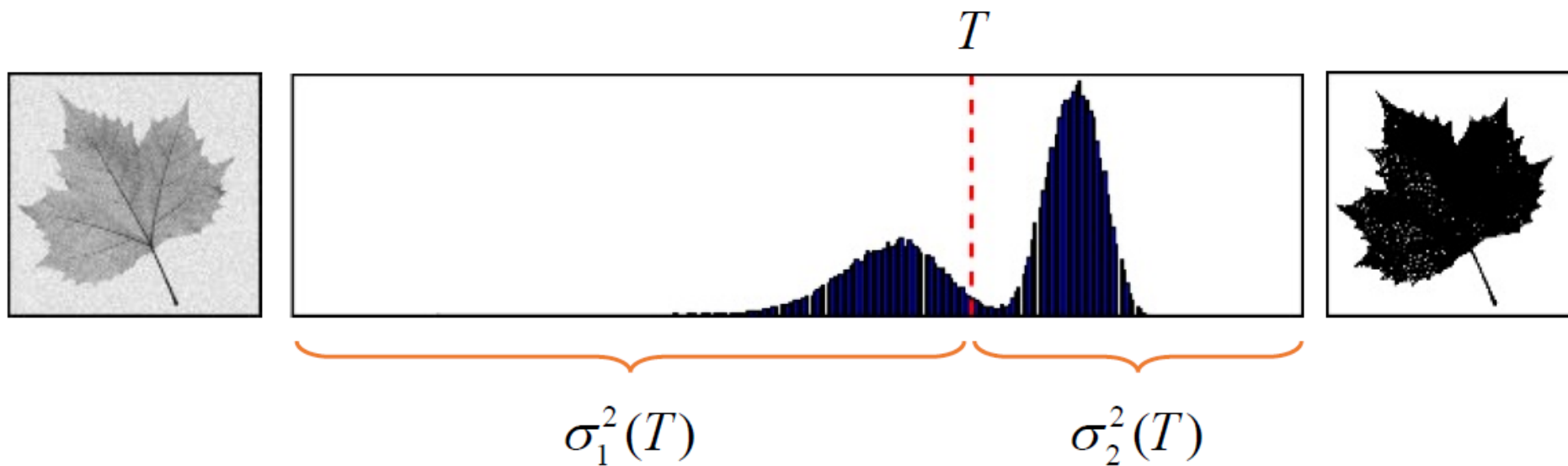


- 전경 배경 픽셀 분포가 크게 다르다면?



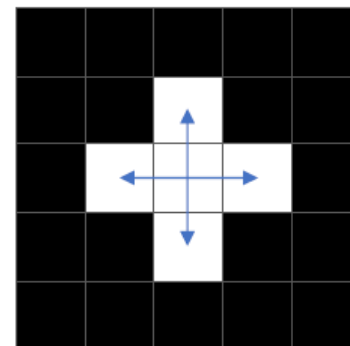
자동 이진화

- Otsu 이진화 방법
 - 입력 이미지가 배경과 객체 구대로 구성되어있다고 가정 -> Bimodal histogram
 - 임의의 임계값 T 에 의해 나뉘지는 두 픽셀 분포 그룹의 분산이 최소가 되는 T 를 선택
 - 일종의 최적화 알고리즘

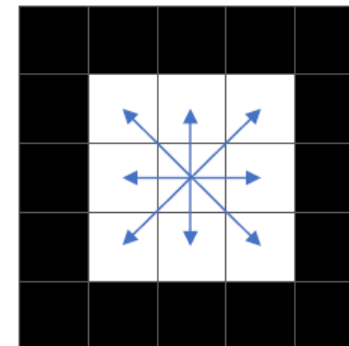


객체 단위 분석

- 객체 단위 분석
 - 객체를 분할하여 특징을 분석
 - 객체 위치 및 크기 정보 분석
- 레이블링이란?
 - 동일 객체에 속한 모든 픽셀에 고유한 번호를 매기는 작업
 - 일반적으로 이진 영상에서 수행
 - Opencv는 3.x. 버전부터 레이블링 알고리즘 함수를 제공
- 픽셀의 연결 관계
 - 4-이웃 연결 관계
 - 8-이웃 연결 관계



4-이웃 연결 관계

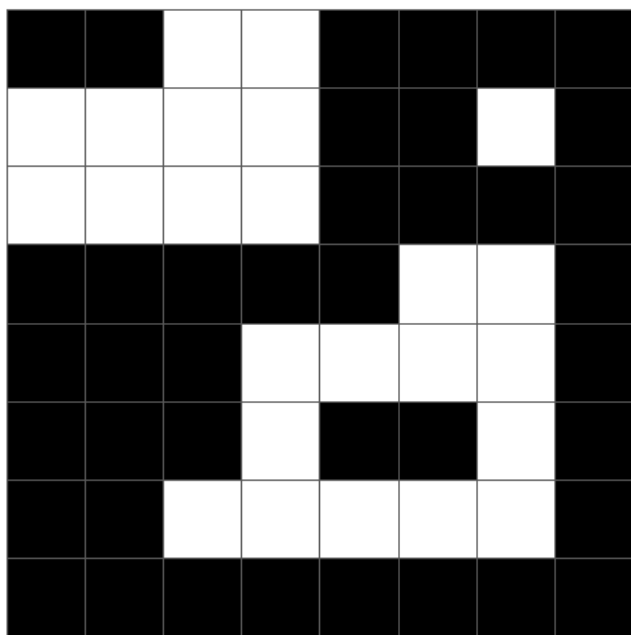


8-이웃 연결 관계

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

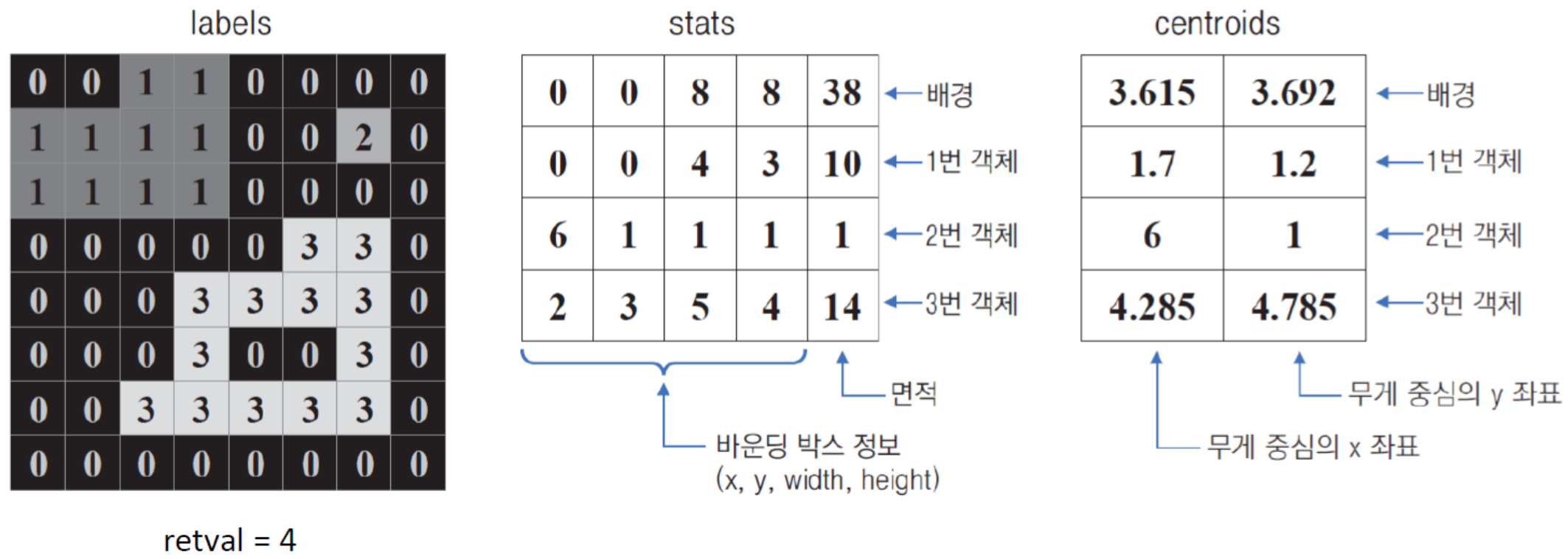
- 객체 정보를 함께 반환하는 레이블링 함수

```
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.
- stats : 각 객체의 바운딩 박스, 픽셀 개수 정보를 담은 행렬. Numpy.ndarray.shape=(N,5)
- Centroids : 각 객체의 무게 중심 위치 정보를 담은 행렬
- Connectivity : 4 또는 8. 기본은 8
- Ltype : labels 행렬 타입
- Retval : 객체의 수

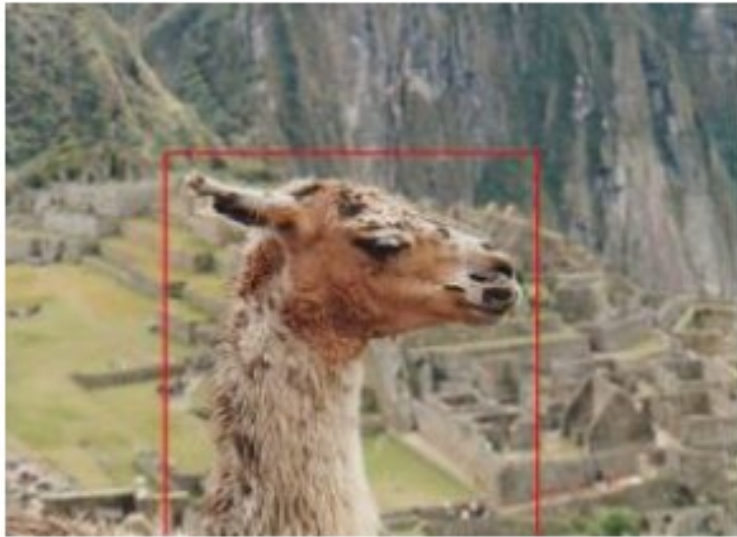
레이블링

- 객체 정보를 함께 반환하는 레이블링 함수



그랩컷

- 그랩컷(grab cut)
- 이미지 픽셀을 두 개의 그룹으로 나누는 최적의 컷을 찾는 방식



그랩컷

- 그랩컷(grab cut) 함수

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

- Img : 입력영상 이미지
- Mask : 입출력 마스크 cv2.GC_BGD(0), cv2.GC_FGD(1), cv2.GC_PR_BGD(2), cv2.GC_PR_FGD(3) 네개의값으로구성됨. cv2.GC_INIT_WITH_RECT 모드로초기화.(BGD:background, FGD:Foreground)
- Rect : 관심(ROI) 영역
- bgdModel : 배경 모델 행렬
- fgdModel : 전경 모델 행렬
- iterCount : 결과 생성을 위한 반복 횟수
- Mode : cv2.GC로 시작하는 모드. 보통 cv2.GC_INIT_WITH_RECT 모드로 초기화

템플릿 매칭(1): 이해하기

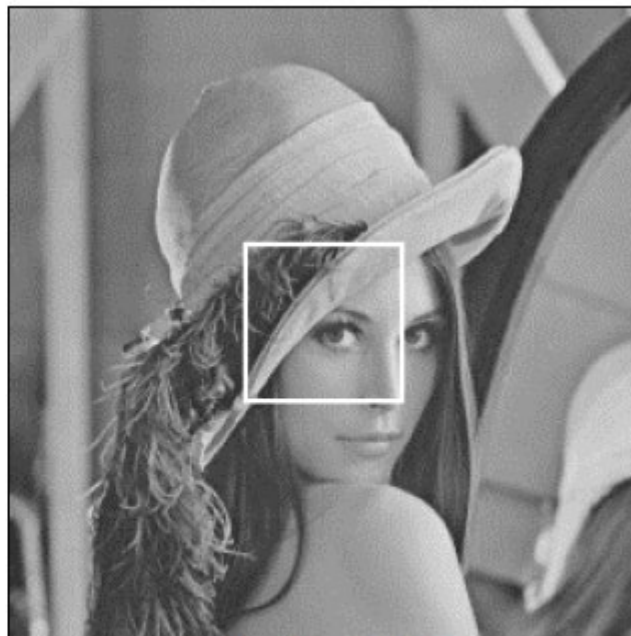
- 템플릿 매칭이란?
- 입력 이미지에서 작은 크기의 템플릿 이미지와 일치하는 부분을 찾는 기법
- 템플릿: 찾을 대상이 되는 작은 이미지



입력 영상

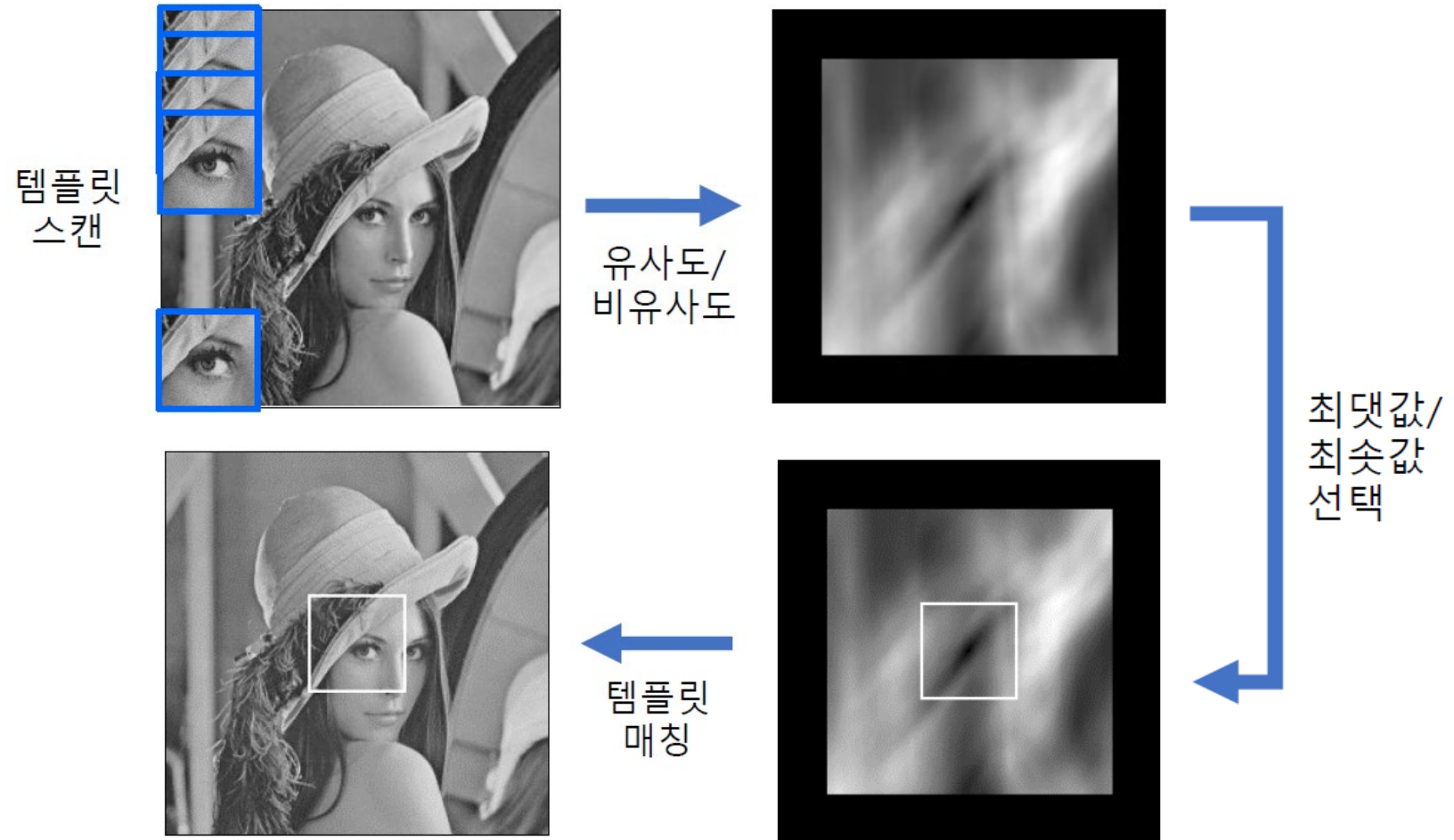


템플릿



검출 결과

템플릿 매칭(1): 이해하기



템플릿 매칭(1): 이해하기

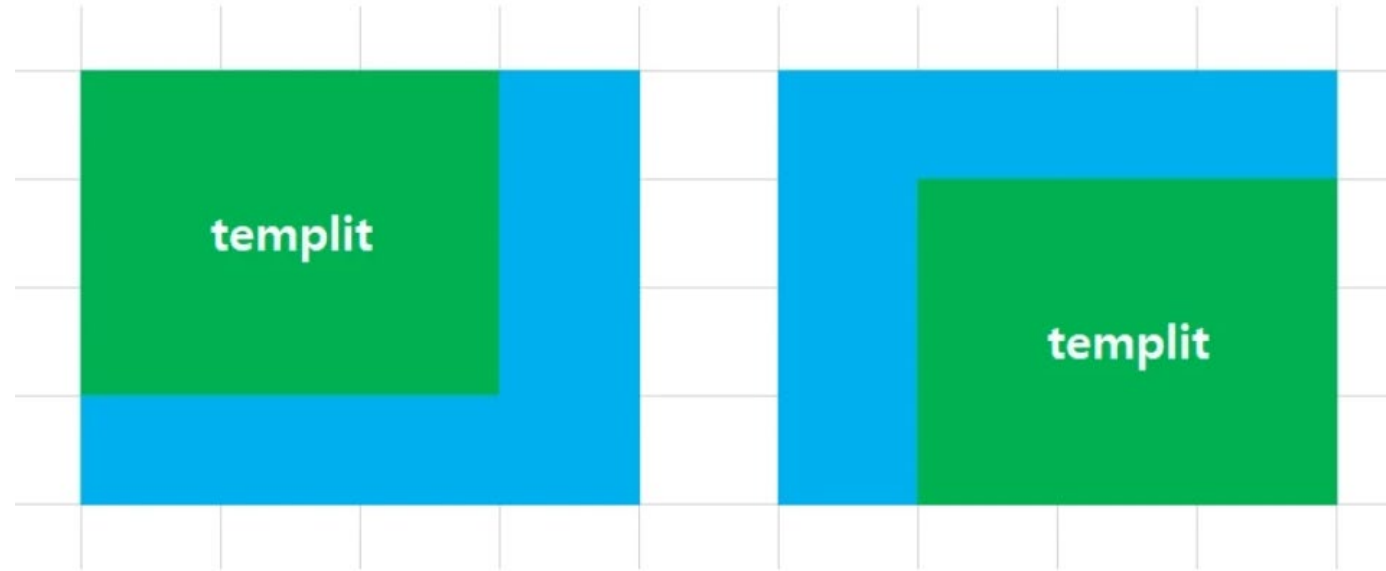
- 템플릿 매칭 함수

```
cv2.matchTemplate(image, templ, method, result=None, mask=None) -> result
```

- Image : 입력 이미지
- Templ : 템플릿 이미지
- Method : 비교방법
- Result : 비교결과 행렬(image의 크기가 $W \times H$ 이고, templ의 크기가 $w \times h$ 이면 result 크기는 $(W-w+1) \times (H-h+1)$)

템플릿 매칭(1): 이해하기

- Result



- $(W-w+1, H-h+1) = (4-3+1, 4-3+1) = (2, 2)$
- 2*2크기를 가짐

템플릿 매칭(1): 이해하기

- Method

cv2.TM_SQDIFF

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

- 템플릿 이미지의 픽셀값과 부분 이미지의 픽셀값을 뺀 값을 제공하여 더하겠다.(완전히 같으면 0, 다르면 값이 커짐)

cv2.TM_CCORR

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

- 템플릭 이미지의 픽셀값과 부분 이미지의 픽셀 값을 곱하고 다 더함(완전히 같으면 값이 제일 크고, 다르면 값이 작아짐)

cv2.TM_CCOEFF

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

- 밝기값 보정 후 CCORR을 실행

템플릿 매칭(1): 이해하기

cv2.TM_SQDIFF



cv2.TM_CCORR



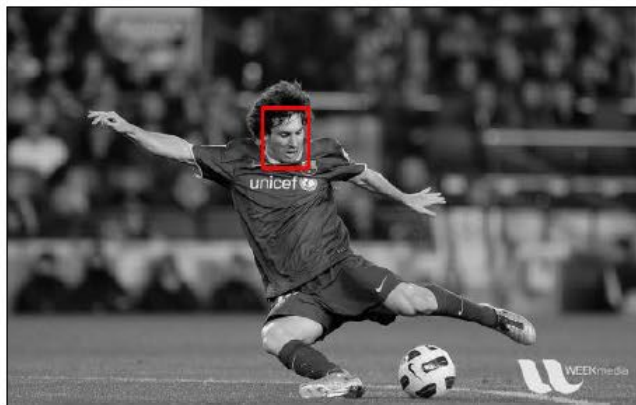
cv2.TM_CCOEFF



cv2.TM_SQDIFF_NORMED



cv2.TM_CCORR_NORMED



cv2.TM_CCOEFF_NORMED



템플릿 매칭(1): 이해하기

cv2.TM_SQDIFF



cv2.TM_CCORR



cv2.TM_CCOEFF



cv2.TM_SQDIFF_NORMED



cv2.TM_CCORR_NORMED

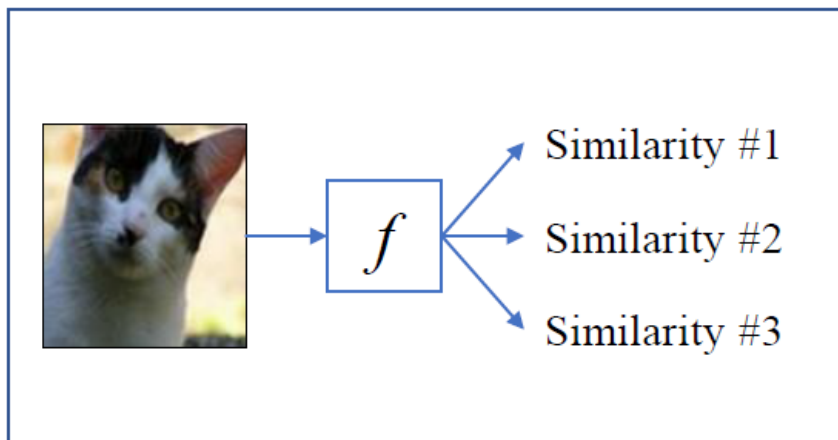





cv2.TM_CCOEFF_NORMED



템플릿 매칭(2): 인쇄체 숫자 인식

- 인식(Recognition)이란?
 - 여러 개의 클래스 중에 가장 유사한 클래스를 선택



			
cat	3.45	0.51	1.42
car	2.90	6.04	4.64
flog	0.09	2.31	3.65
	Good!	Good!	NG!

템플릿 매칭(2): 인쇄체 숫자 인식

- 숫자 템플릿 이미지
 - Consolas 폰트로 쓰여진 숫자 이미지를 digit0.bmp~digit9.bmp로 저장
 - 각 숫자 이미지의 크기는 100*150 크기로 정규화



digit0.bmp



digit1.bmp



digit2.bmp



digit3.bmp



digit4.bmp



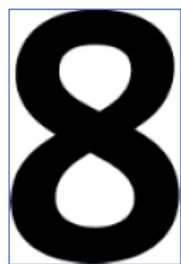
digit5.bmp



digit6.bmp



digit7.bmp



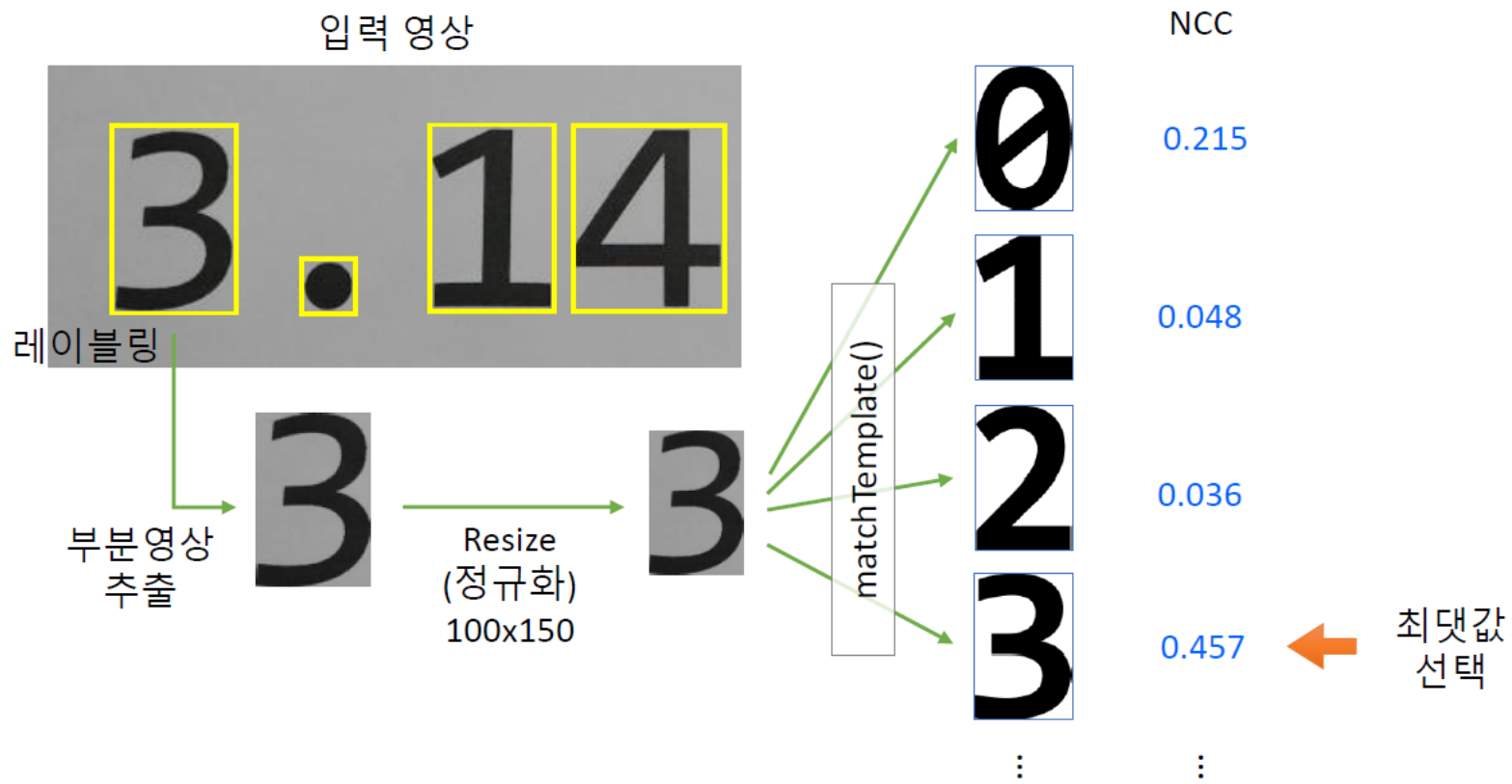
digit8.bmp



digit9.bmp

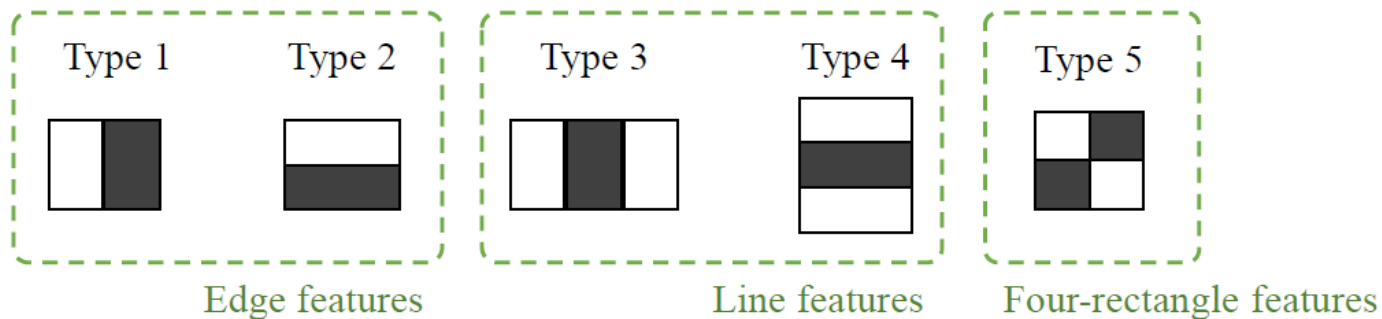
템플릿 매칭(2): 인쇄체 숫자 인식

- 숫자 인식 방법



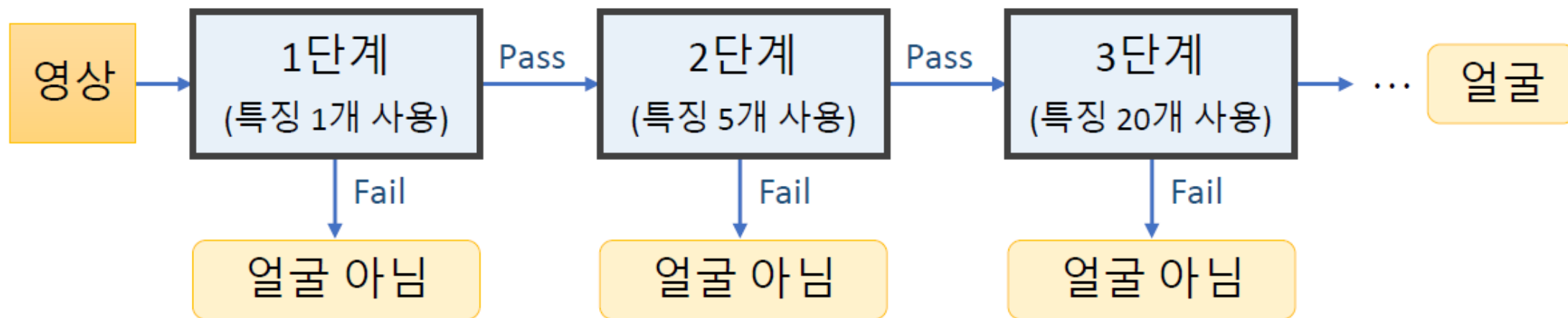
캐스케이드 분류기 : 얼굴 검출

- Viola – Jones 얼굴 검출기
 - Positive 이미지(얼굴 영상 이미지)와 negative 이미지(얼굴이 아닌 영상 이미지)를 훈련하여 빠르고 정확하게 영역을 검출
- 유사 하르 특징(Haar- like features)
 - 사각형 형태의 필터 집합을 사용
 - 흰색 사각형 영역 픽셀 값의 합에서 검정색 사각형 영역 픽셀 값을 뺀 결과 값을 추출(얼굴 특징 추출)



캐스케이드 분류기 : 얼굴 검출

- 캐스케이드 분류기(Cascade classifier)
- 일반적인 이미지에는 얼굴이 한 두개 있을 뿐 나머지 영역은 대부분 non-face영역
- Non-face 영역을 빠르게 skip 하도록 다단계 검사 수행



캐스케이드 분류기 : 얼굴 검출



캐스케이드 분류기 : 얼굴 검출

- Cv2.CascadeClassifier 객체 생성 및 학습 데이터 불러오기

```
cv2.CascadeClassifier( ) -> <CascadeClassifier object>  
cv2.CascadeClassifier(filename) -> <CascadeClassifier object>
```

```
cv2.CascadeClassifier.load(filename) -> retval
```

- Filename: XML 파일이름
- Retval : 성공하면 True, 실패하면 False.

캐스케이드 분류기 : 얼굴 검출

- 미리 학습된 XML 파일 다운로드

<https://github.com/opencv/opencv/tree/master/data/haarcascades>

XML 파일 이름	검출 대상
haarcascade_frontalface_default.xml haarcascade_frontalface_alt.xml haarcascade_frontalface_alt2.xml haarcascade_frontalface_alt_tree.xml	정면 얼굴 검출
haarcascade_profileface.xml	측면 얼굴 검출
haarcascade_smile.xml	웃음 검출
haarcascade_eye.xml haarcascade_eye_tree_eyeglasses.xml haarcascade_lefteye_2splits.xml haarcascade_righteye_2splits.xml	눈 검출
haarcascade_frontalcatface.xml haarcascade_frontalcatface_extended.xml	고양이 얼굴 검출
haarcascade_fullbody.xml	사람의 전신 검출
haarcascade_upperbody.xml	사람의 상반신 검출
haarcascade_lowerbody.xml	사람의 하반신 검출
haarcascade_russian_plate_number.xml haarcascade_licence_plate_rus_16stages.xml	러시아 자동차 번호판 검출

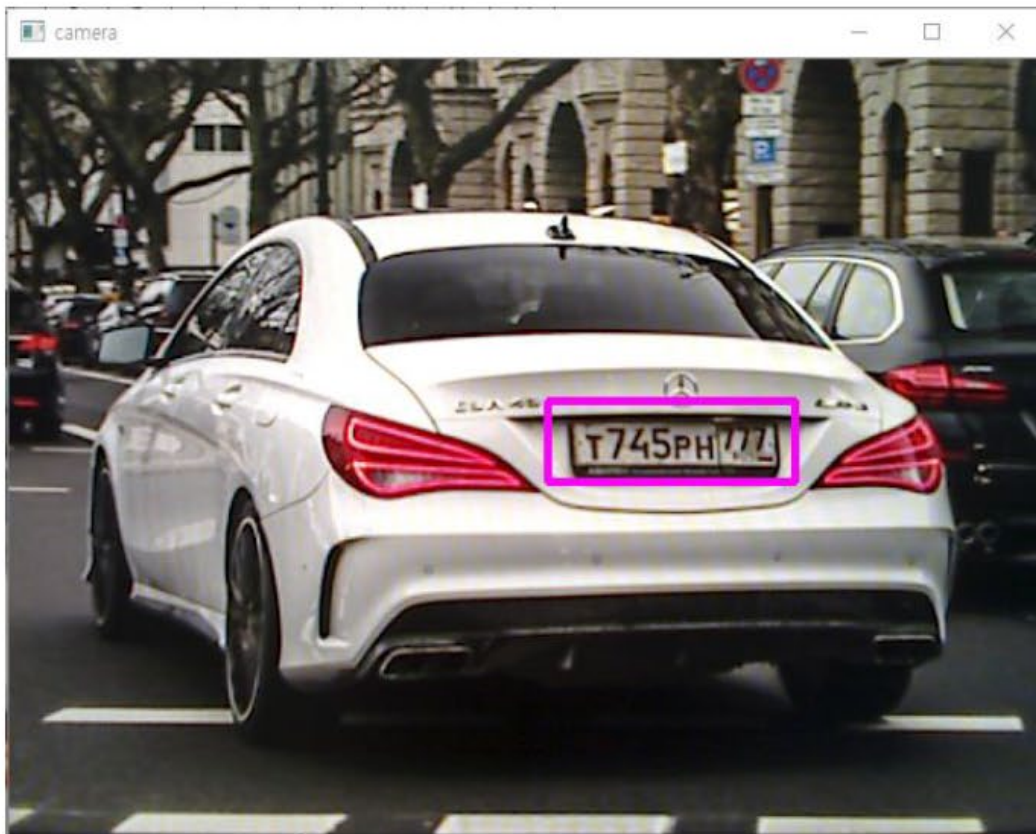
캐스케이드 분류기 : 얼굴 검출

- CascadeClassifier 멀티스케일 객체 검출 함수

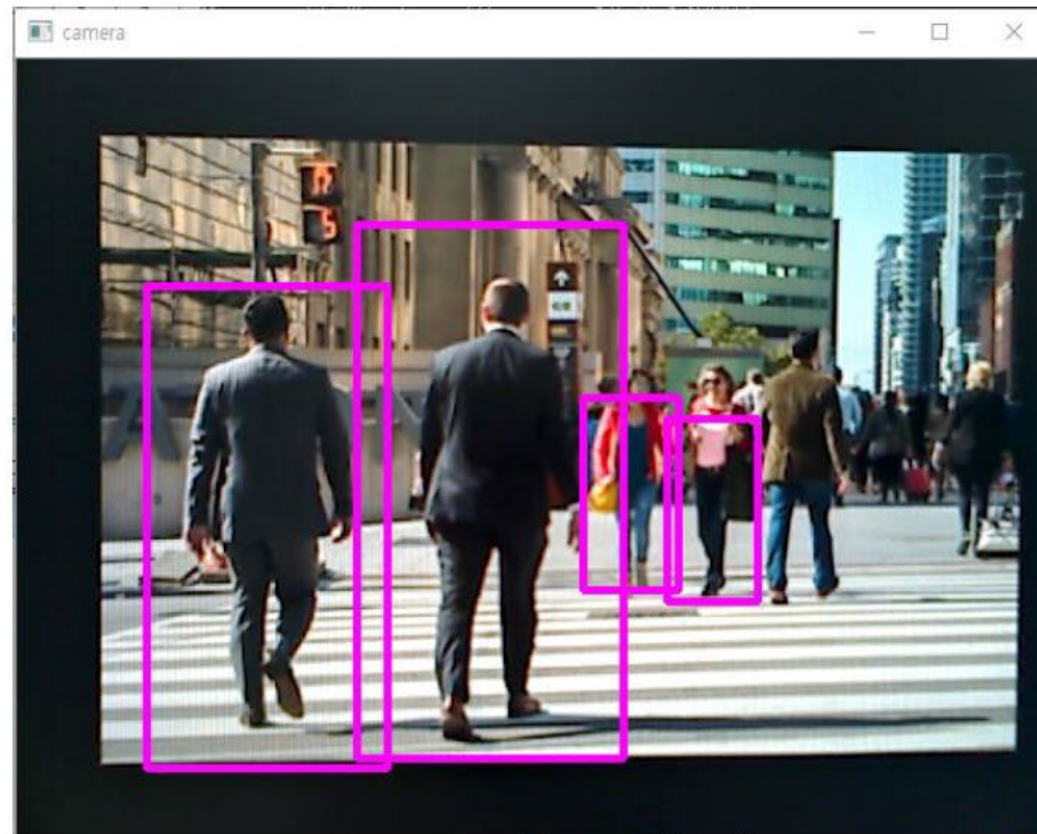
```
cv2.CascadeClassifier.detectMultiScale(image, scaleFactor=None,  
    minNeighbors=None, flags=None, minSize=None, maxSize=None) -> result
```

- Image : 입력이미지
- scaleFactor : 영상 축소 비율. 기본값은 1.1 #내부 사각형의 크기를 키워줌 크기가 커질수록 속도 빨라짐
- minNeighbors : 얼마나 많은 이웃 사각형이 검출되어야 최종 검출 영역으로 설정할지를 지정. 기본값은 3
- Flags : 현재 사용하지 않음
- minSize : 최소 객체 크기
- maxSize : 최대 객체 크기
- Result : 검출된 객체의 사각형 정보(x,y,w,h)

캐스케이드 분류기 : 얼굴 검출



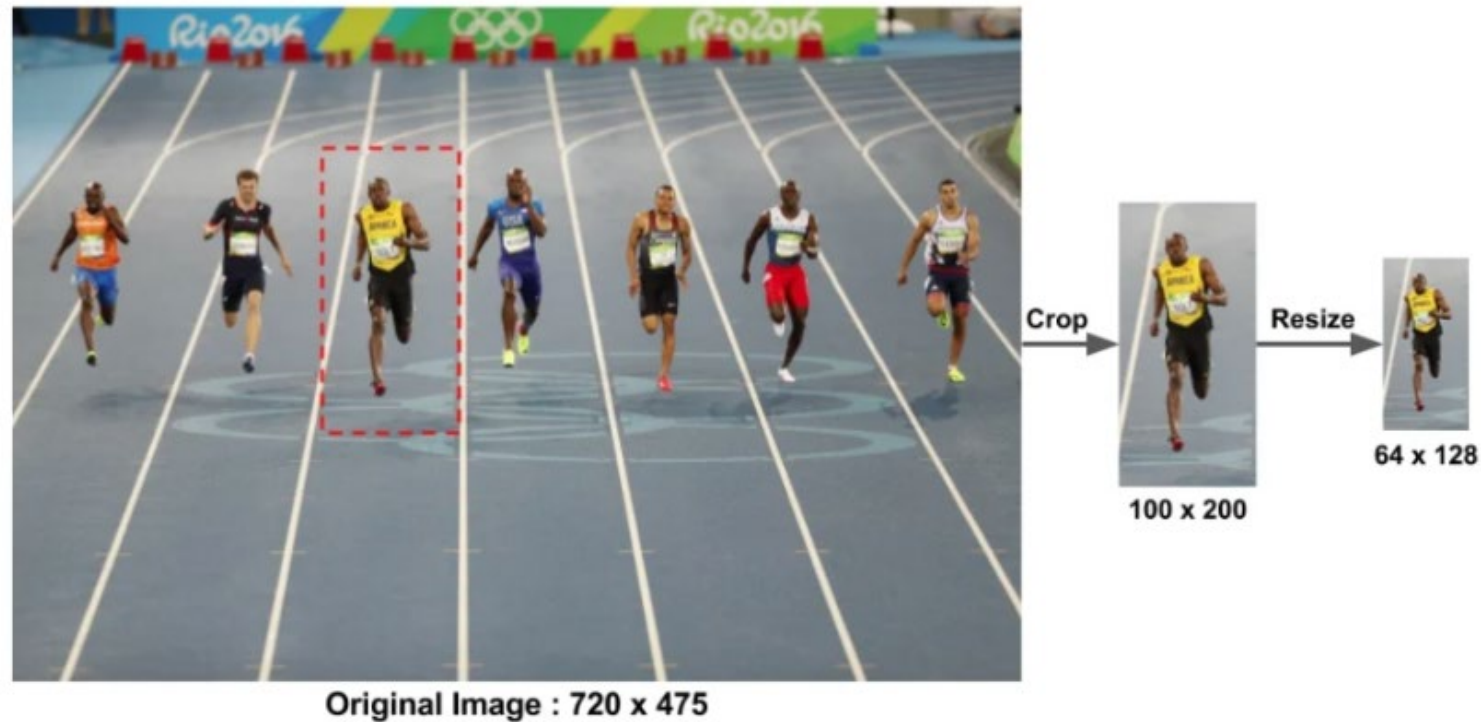
러시아 자동차 번호판 검출
(haarcascade_russian_plate_number.xml)



사람 전신 검출
(haarcascade_fullbody.xml)

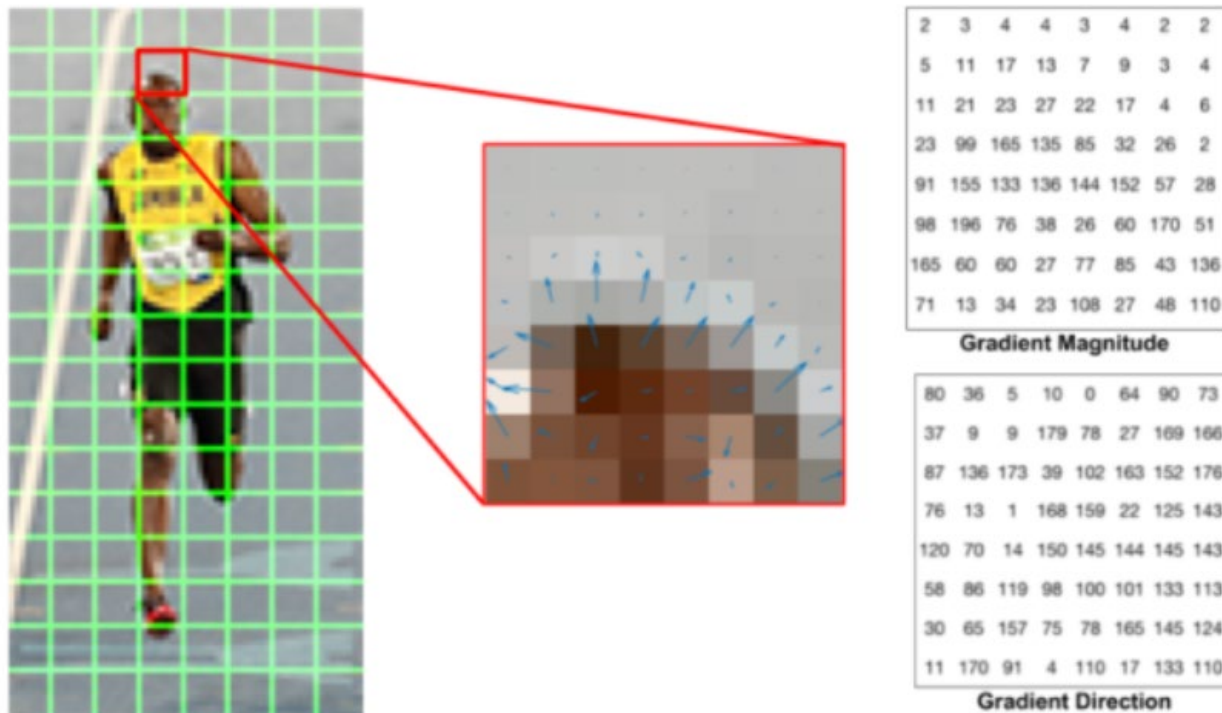
HOG 보행자 검출

- HOG(Histogram of Oriented Gradients)란?
 - 이미지의 지역적 그래디언트 방향 정보를 특징 벡터로 사용
 - 머신러닝과 결합하여 다양한 객체 인식에 활용됨

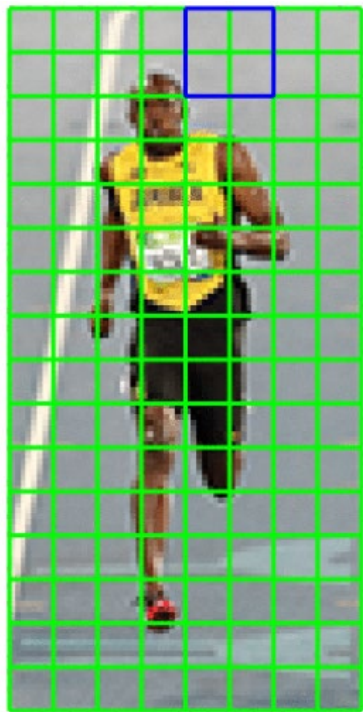


HOG 보행자 검출

- 64*128 이미지를 8*8크기의 셀(cell)로 분할
- 각 셀마다 방향과 크기 성분을 이용하여 방향 히스토그램을 생성(180도를 20도씩 9가지 방향) : 방향 히스토그램 빈(bin)개수는 9개



HOG 보행자 검출



- 8*8 셀 4개를 하나의 블록으로 지정
-> 하나의 블록의 크기는 16*16
-> 8픽셀 단위로 이동 : stride = 8
-> 각 블록의 히스토그램 빈(bin) 개수는 4*9=36개
- 하나의 부분 이미지 패치에서의 특징 벡터 크기
-> $7 * 15 * 36 = 3780$

HOG 보행자 검출

- HOG 객체 생성 을 위한 학습된 분류기 계수 불러오기

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

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

- Retval : 미리 훈련된 특징 벡터. Numpy.ndarray.shape = (3781,1)
- SVM 분류기 계수 등록하기 -> 기계학습 분야 중 하나로 패턴 인식, 자료 분석을 위한 지도학습 모델

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

HOG 보행자 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 이미지
- hitThreshold : 특징 벡터와 svm 분류 평면까지의 거리에 대한 임계값
- winStride : 이동 크기
- Padding : 패딩 크기
- Scale : 검색 크기 비율
- FinalThreshold : 검출 결정을 위한 임계값
- foundLocations: 검출된 사각형 영역 정보
- foundWeights : 검출된 사각형 영역에 대한 신뢰도