동양미래대학교

파이썬을 활용한 컴퓨터 비전 입문

Chapter 13. 객체 검출

동양미래대학교
인공지능소프트웨어학과
권 범

본 강의자료는 길벗 출판사의『OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝』
교재 내용을 토대로 작성되었습니다.

목차

13.1 템플릿 매칭

13.2 캐스케이드 분류기와 얼굴 검출

13.3 HOG 알고리즘과 보행자 검출

13.4 QR 코드 검출

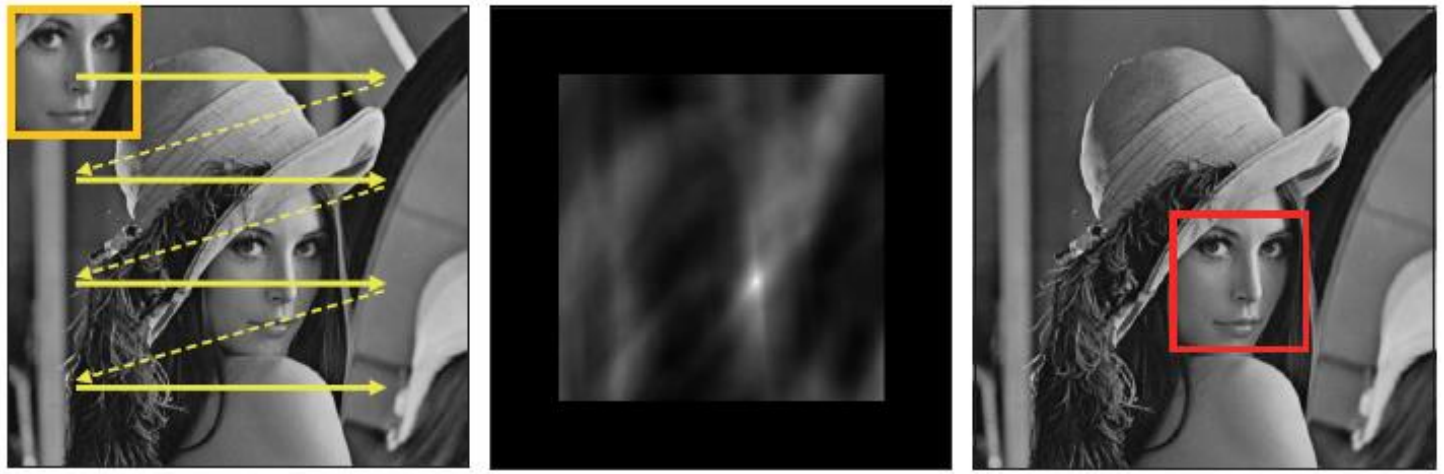
❖ 템플릿 매칭 (1/20)

- **입력 영상에서 작은 크기의 부분 영상 위치를 찾아내고 싶은 경우**에 주로 **템플릿 매칭**(**template matching**) 기법을 사용함
- **템플릿**(**template**)은 **찾고자 하는 대상**이 되는 작은 크기의 영상을 의미함
- 템플릿 매칭은 작은 크기의 템플릿 영상을 입력 영상 전체 영역에 대해 이동하면서 가장 비슷한 위치를 수치적으로 찾아내는 방식임

❖ 템플릿 매칭 (2/20)

- 그림 13-1은 레나 영상에서 레나 얼굴 영역 부분 영상을 템플릿으로 사용하여 템플릿 매칭을 수행하는 과정을 보여 줌
- 그림 13-1(a)와 같이 템플릿 영상을 입력 영상 전체 영역에 대해 이동하면서 템플릿 영상과 입력 영상 부분 영상과의 **유사도**(**similarity**) 또는 **비유사도**(**dissimilarity**)를 계산함
- 유사도를 계산할 경우에는 템플릿 영상과 비슷한 부분 영상 위치에서 값이 크게 나타남
- 반대로 비유사도를 계산할 경우에는 템플릿 영상과 비슷한 부분에서 값이 작게 나타남

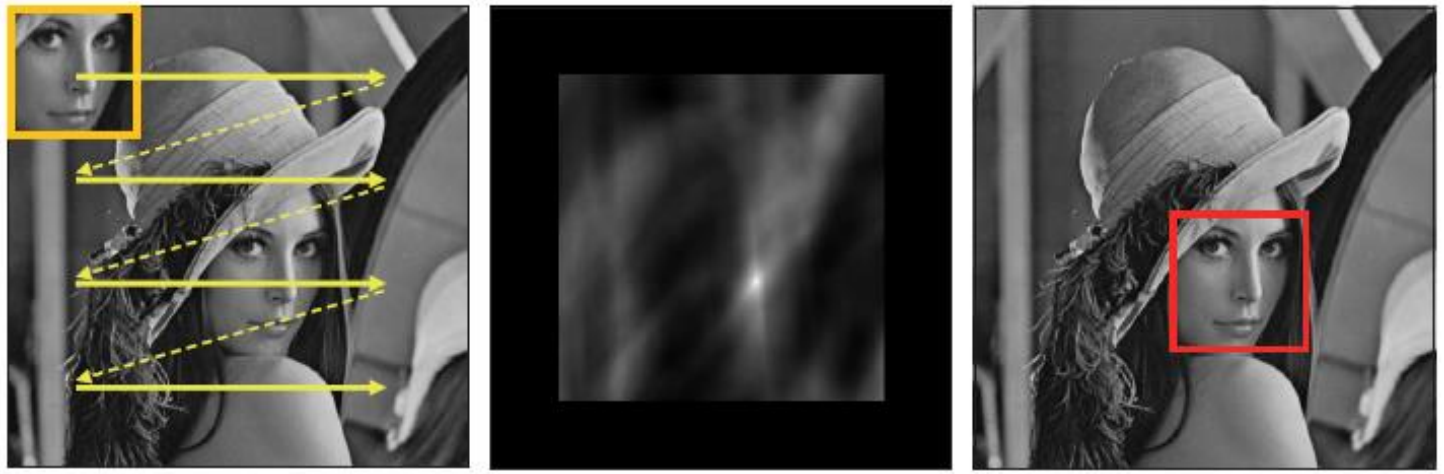
▼ **그림 13-1 템플릿 매칭 동작 원리**



❖ 템플릿 매칭 (3/20)

- 그림 13-1(b)는 입력 영상의 모든 위치에서 템플릿 영상과의 유사도를 계산함
- 그 결과를 그레이스케일 영상 형태로 나타냄
- 그림 13-1(b)에서 **가장 밝은 픽셀 위치가 템플릿 영상과 가장 유사한 부분임**
- 이 위치를 빨간색 사각형으로 표시한 결과를 그림 13-1(c)에 나타냄

▼ **그림 13-1 템플릿 매칭 동작 원리**



❖ 템플릿 매칭 (4/20)

- OpenCV에서는 `matchTemplate()` 함수를 사용하여 템플릿 매칭을 수행할 수 있음

```
result = cv2.matchTemplate(image, templ, method)
```

`image`	입력 영상. 8비트 또는 32비트 실수형
`templ`	템플릿 영상. 입력 영상 `image`보다 같거나 작아야 하며, `image`와 타입이 같아야 합니다.
`result`	(출력) 비교 결과를 저장할 행렬
`method`	템플릿 매칭 비교 방법. TemplateMatchModes 열거형 상수 중 하나를 지정합니다.

❖ 템플릿 매칭 (5/20)

- `matchTemplate()` 함수에서 템플릿 영상과 입력 영상 간의 비교 방식은 `method` 인자로 설정할 수 있음
- `method` 인자에는 TemplateMatchModes 열거형 상수 중 하나를 지정할 수 있음
- **TM_SQDIFF**는 제곱차(squared difference) 매칭 방법을 의미함
- 이 경우 두 영상이 완벽하게 일치하면 0이 되고 서로 유사하지 않으면 0보다 큰 양수를 갖음
- **TM_CCORR**은 상관관계(correlation) 매칭 방법을 의미함
- 이 경우 두 영상이 유사하면 큰 양수가 나오고 유사하지 않으면 작은 값이 나옴

❖ 템플릿 매칭 (6/20)

- **TM_CCOEFF**는 상관계수(correlation coefficient) 매칭 방법을 의미함
- 이는 비교할 두 영상을 미리 평균 밝기로 보정한 후 상관관계 매칭을 수행하는 방식임
- TM_CCOEFF 방법은 두 비교 영상이 유사하면 큰 양수가 나옴
- 유사하지 않으면 0에 가까운 양수 또는 음수가 나오게 됨

❖ 템플릿 매칭 (7/20)

- TM_SQDIFF, TM_CCORR, TM_CCOEFF 방법에 대해 각각 영상의 **밝기 차이 영향을 줄여 주는 정규화 수식이 추가**된 **TM_SQDIFF_NORMED**, **TM_CCORR_NORMED**, **TM_CCOEFF_NORMED** 방법도 제공함
- TM_CCORR_NORMED 방법은 매칭 결괏값이 0에서 1 사이의 실수로 나타남
- TM_CCOEFF_NORMED 방법은 매칭 결괏값이 -1에서 1 사이의 실수로 나타남
- 두 방법 모두 결괏값이 1에 가까울수록 매칭이 잘 되었음을 의미함

❖ 템플릿 매칭 (8/20)

▼ 표 13-1 TemplateMatchModes 열거형 상수

TemplateMatchModes 열거형 상수	설명
TM_SQDIFF	제곱차 매칭 방법 $R(x,y)=\sum_{x',y'}\left(T(x',y')-I(x+x',y+y')\right)^2$
TM_SQDIFF_NORMED	정규화된 제곱차 매칭 방법 $R(x,y)=\frac{\sum_{x',y'}\left(T(x',y')-I(x+x',y+y')\right)^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$
TM_CCORR	상관관계 매칭 방법 $R(x,y)=\sum_{x',y'}T(x',y')\cdot I(x+x',y+y')$
TM_CCORR_NORMED	정규화된 상관관계 매칭 방법 $R(x,y)=\frac{\sum_{x',y'}T(x',y')\cdot I(x+x',y+y')}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$

❖ 템플릿 매칭 (9/20)

▼ 표 13-1 TemplateMatchModes 열거형 상수

TemplateMatchModes 열거형 상수	설명
TM_CCOEFF	상관계수 매칭 방법 $R(x,y)=\sum_{x',y'}T'(x',y')\cdot I'(x+x',y+y')$ $T'(x',y')=T(x',y')-1/(w\cdot h)\cdot\sum_{x'',y''}T'(x'',y'')$ $I'(x+x',y+y')=I(x+x',y+y')-1/(w\cdot h)\cdot\sum_{x'',y''}I(x+x'',y+y'')$
TM_CCOEFF_NORMED	정규화된 상관계수 매칭 방법 $R(x,y)=\frac{\sum_{x',y'}T'(x',y')\cdot I'(x+x',y+y')}{\sqrt{\sum_{x',y'}T'(x',y')^2\cdot\sum_{x',y'}I'(x+x',y+y')^2}}$

❖ 템플릿 매칭 (10/20)

- 여러 매칭 방법 중에서 **정규화된 상관계수 매칭 방법이 좋은 결과를 제공하는 것으로 알려져 있음**
- 계산 수식이 가장 복잡하기 때문에 실제 동작 시 연산량이 많을 수 있다는 점을 고려해야 함
- 제곱차 매칭 방법을 사용할 경우, result 결과 행렬에서 최솟값 위치를 가장 매칭이 잘 된 위치로 선택해야 함
- 반면에 상관관계 또는 상관계수 매칭 방법을 사용할 경우에는 result 결과 행렬에서 최댓값 위치가 가장 매칭이 잘 된 위치임
- 참고로 result 행렬에서 최솟값 위치 또는 최댓값 위치는 OpenCV의 `minMaxLoc()` 함수를 이용하여 쉽게 알아낼 수 있음

❖ 템플릿 매칭 (11/20)

- 코드 13-1에 나타난 `template_matching()` 함수는 `imread()` 함수로 두 장의 영상을 불러와서 템플릿 매칭을 수행함
- 유사도 계산 결과와 템플릿 매칭 결과를 화면에 출력함

코드 13-1 템플릿 매칭 예제 (template.py)

```
1   import sys
2   import numpy as np
3   import cv2
4
5
6   img = cv2.imread('circuit.bmp', cv2.IMREAD_COLOR)
7   templ = cv2.imread('crystal.bmp', cv2.IMREAD_COLOR)
8
9   if img is None or templ is None:
10      print('Image load failed!')
11      sys.exit()
12
```

❖ 템플릿 매칭 (12/20)

코드 13-1 템플릿 매칭 예제 (template.py)

```
13  img = img + (50, 50, 50)
14
15  noise = np.zeros(img.shape, np.int32)
16  cv2.randn(noise, 0, 10)
17  img = cv2.add(img, noise, dtype=cv2.CV_8UC3)
18
19  res = cv2.matchTemplate(img, templ, cv2.TM_CCOEFF_NORMED)
20  res_norm = cv2.normalize(res, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
21
22  _, maxv, _, maxloc = cv2.minMaxLoc(res)
23  print('maxv:', maxv)
24
```

❖ 템플릿 매칭 (13/20)

코드 13-1 템플릿 매칭 예제 (template.py)

```
25  (th, tw) = templ.shape[:2]
26  cv2.rectangle(img, maxloc, (maxloc[0] + tw, maxloc[1] + th), (0, 0, 255), 2)
27
28  cv2.imshow('templ', templ)
29  cv2.imshow('res_norm', res_norm)
30  cv2.imshow('img', img)
31  cv2.waitKey()
32  cv2.destroyAllWindows()
```

❖ 템플릿 매칭 (14/20)

- template.py 소스 코드 설명
  - 6행 circuit.bmp 파일을 입력 영상 `img`로 사용합니다.
  - 7행 crystal.bmp 파일을 템플릿 영상 `templ`로 사용합니다.
  - 13행 입력 영상 밝기를 50만큼 증가시킵니다.
  - 15~17행 입력 영상에 표준 편차가 10인 가우시안 잡음을 추가합니다.
  - 19행 정규화된 상관계수 매칭 방법을 사용하여 템플릿 매칭을 수행합니다.
  - 20행 템플릿 매칭 결과 행렬 `res`의 모든 원소 값을 0~255 사이로 정규화하고, 타입을 `CV_8U`로 변환하여 `res_norm` 영상에 저장합니다.

❖ 템플릿 매칭 (15/20)

- template.py 소스 코드 설명

  - 22행 `res` 행렬에서 최댓값 위치를 찾아 `maxloc`에 저장합니다.
    이 위치에서의 최댓값 `maxv`는 템플릿 매칭이 잘 되었는지를 가늠하는 척도로 사용할 수 있습니다.

  - 23행 `res` 행렬의 최댓값을 콘솔 창에 출력합니다.

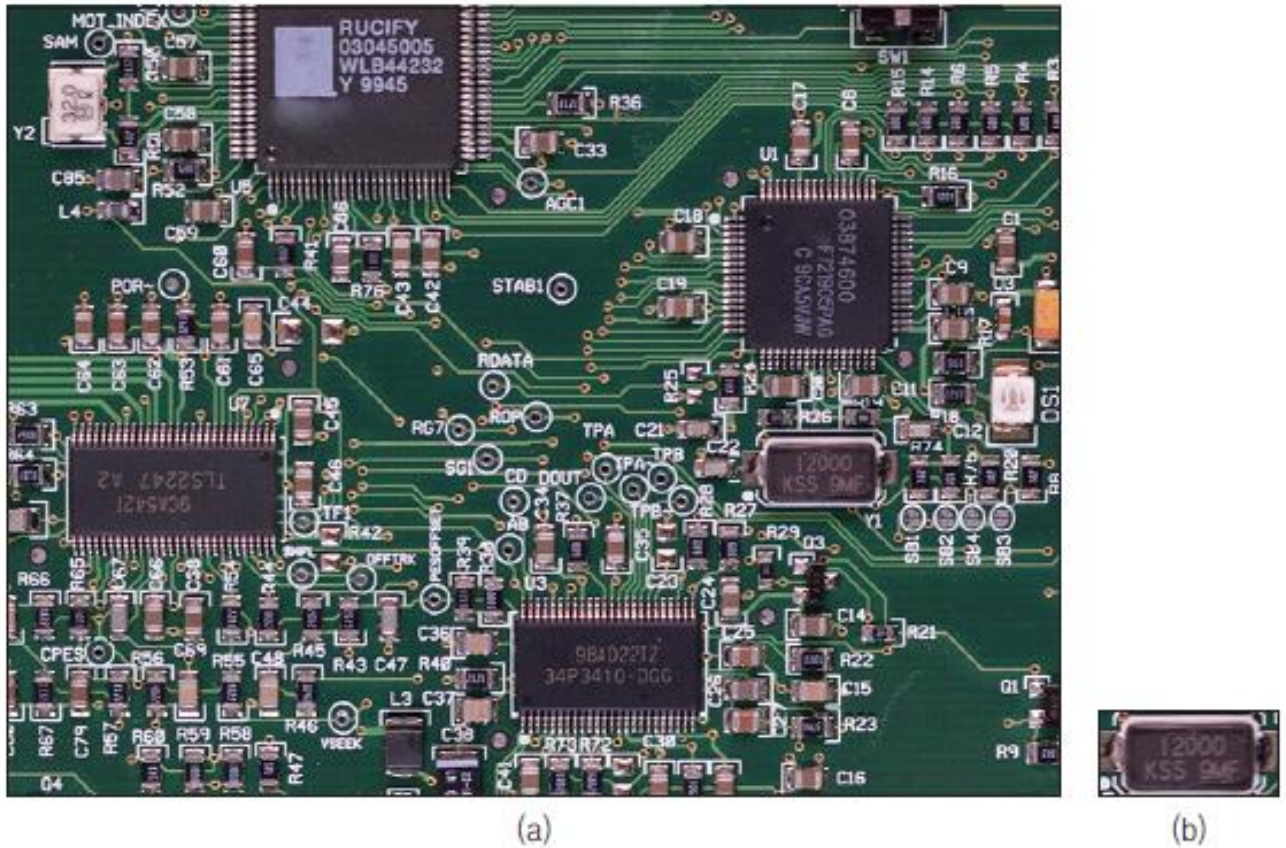
  - 13행 `img` 영상에 템플릿 매칭으로 찾은 위치를 빨간색 사각형으로 표시합니다.

❖ 템플릿 매칭 (16/20)

- 그림 13-2(a)는 입력 영상인 circuit.bmp 파일이고, 그림 13-2(b)는 템플릿으로 사용한 crystal.bmp 영상임
- `template_matching()` 함수는 그림 13-2(a)의 회로 기판 영상에서 그림 13-2(b)의 수정 발진기 부품 위치를 찾아 표시함
- `template_matching()` 함수에서는 실제 영상 획득 과정에서 발생할 수 있는 잡음과 조명의 영향을 시뮬레이션하기 위해 입력 영상의 밝기를 50만큼 증가시킴
- 표준 편차가 10인 가우시안 잡음을 추가한 후 템플릿 매칭을 수행함

❖ 템플릿 매칭 (17/20)

▼ 그림 13-2 템플릿 매칭 예제에서 사용한 입력 영상과 템플릿 영상



❖ 템플릿 매칭 (18/20)

- 그림 13-3(a)는 템플릿으로 사용한 crystal.bmp 영상임
- 그림 13-3(b)는 템플릿 매칭으로 계산된 유사도 행렬을 그레이스케일 형식 영상으로 나타낸 `res_norm` 영상임
- `template_matching()` 함수에서 `TM_CCOEFF_NORMED` 방식으로 템플릿 매칭을 수행했으므로 템플릿 매칭 결과 행렬 `res`는 -1부터 1 사이의 실수임
- 이를 0부터 255 사이의 정수 범위로 정규화한 결과가 `res_norm`임

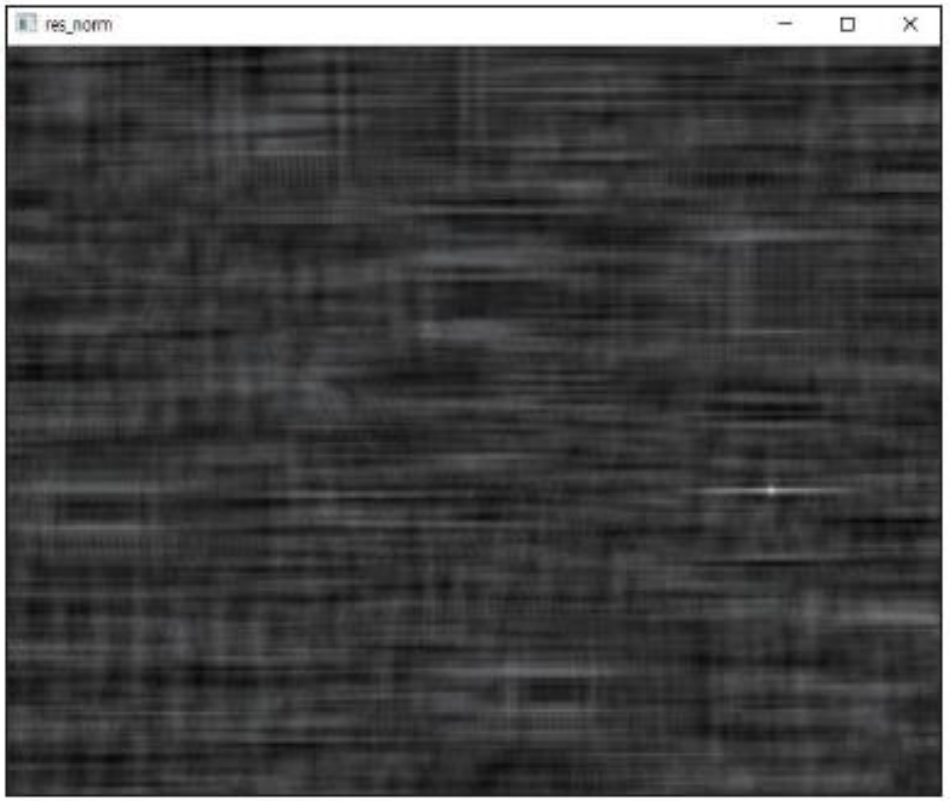
❖ 템플릿 매칭 (19/20)

- `res_norm` 영상에서 가장 밝게 나타나는 위치가 템플릿 영상과 가장 유사한 부분임
- 그림 13-3(c)에 나타난 `img` 영상은 원본 circuit.bmp 영상보다 밝아졌고 잡음이 추가되어 지저분하게 변경되었지만, 수정 발진기 부품 위치가 정확하게 검출됨
- `template_matching()` 함수가 실행되면 콘솔창에는 "maxv: 0.976276" 문자열이 출력됨
- 이는 템플릿 매칭으로 검출된 위치에서 정규화된 상관계수 값을 나타냄
- 이 값이 1에 가까운 실수이므로 매칭이 잘 되었다고 가늠할 수 있음

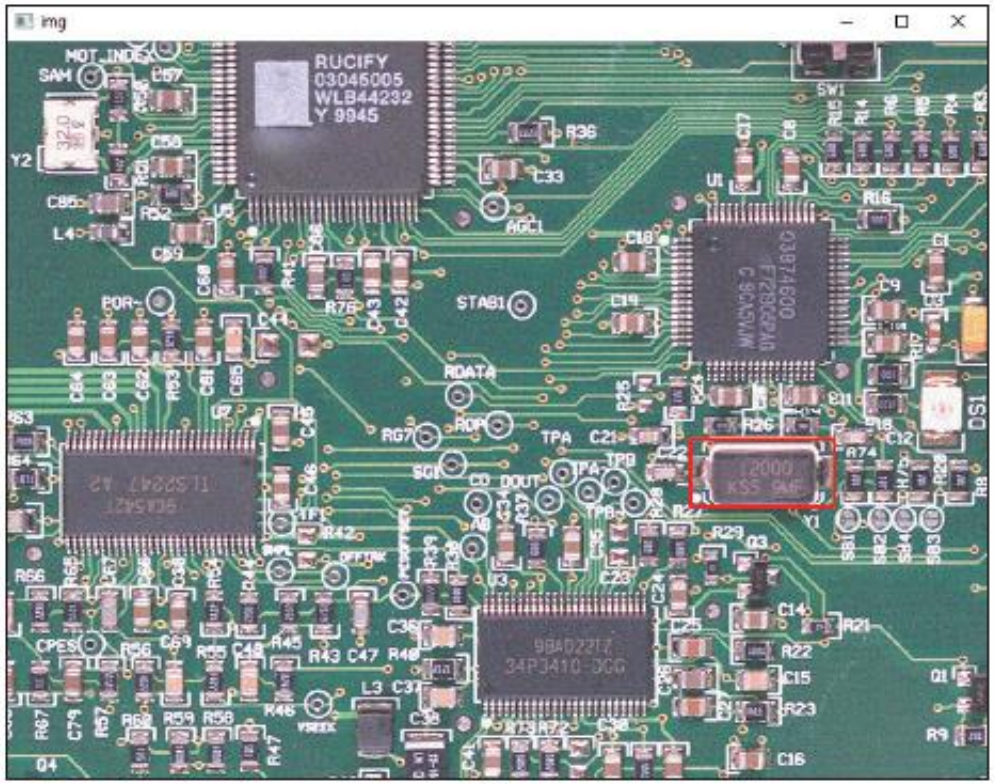
❖ 템플릿 매칭 (20/20)

▼ 그림 13-3 템플릿 매칭 예제 실행 결과

(a)



(b)



(c)

13.2 캐스케이드 분류기와 얼굴 검출

13.2 캐스케이드 분류기와 얼굴 검출

❖ 캐스케이드 분류기와 얼굴 검출 (1/10)

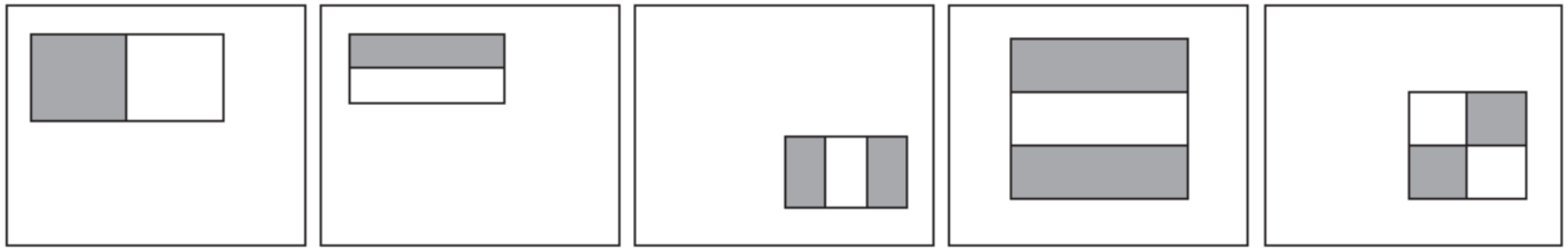
- OpenCV에서 제공하는 **얼굴 검출 기능**은 2001년에 비올라(P. Viola)와 존스(M. Jones)가 발표한 부스팅(boosting) 기반의 캐스케이드 분류기(cascade classifier) 알고리즘을 기반으로 만듦[Viola01]
- 비올라와 존스가 개발한 객체 검출 알고리즘은 기본적으로 다양한 객체를 검출할 수 있지만, 특히 얼굴 검출에 적용되어 속도와 정확도를 인정받은 기술임
- 비올라-존스 얼굴 검출 알고리즘은 기본적으로 영상을 24×24 크기로 정규화함
- 유사-하르 필터(Haar-like filter) 집합으로부터 특징 정보를 추출하여 얼굴 여부를 판별함

[Viola01] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Dec. 2001, pp. 511-518.

❖ 캐스케이드 분류기와 얼굴 검출 (2/10)

- 유사-하르 필터란 흑백 사각형이 서로 붙어 있는 형태로 구성된 필터임
- 24×24 영상에서 만들 수 있는 유사-하르 필터의 예를 그림 13-4에 나타냄
- 유사-하르 필터 형태에서 흰색 영역 픽셀 값은 모두 더함
- 검은색 영역 픽셀 값은 모두 빼서 하나의 특징 값을 얻을 수 있음
- 사람의 정면 얼굴 형태가 전형적으로 밝은 영역(이마, 미간, 볼 등)과 어두운 영역(눈썹, 입술 등)이 정해져 있음
- 유사-하르 필터로 구한 특징 값은 얼굴을 판별하는 용도로 사용할 수 있음

▼ **그림 13-4 유사-하르 필터의 예**



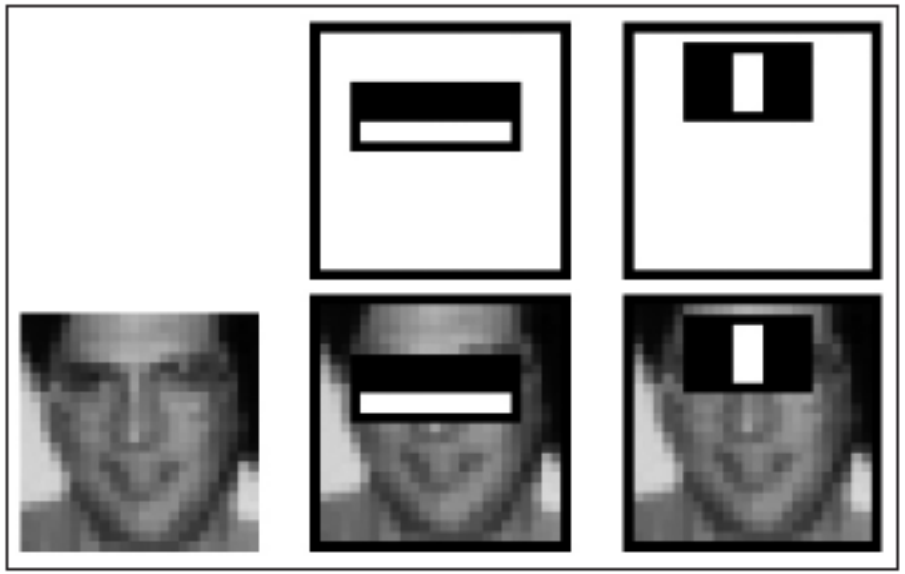
❖ 캐스케이드 분류기와 얼굴 검출 (3/10)

- 24×24 크기에서 다양한 크기의 유사-하르 필터를 대략 18만 개 생성할 수 있음
- 픽셀 값의 합과 차를 계산하는 것이 복잡하지는 않지만 시간이 오래 걸린다는 점이 문제 됨
- 다행히 비올라와 존스는 **에이다부스트**(**adaboost**) **알고리즘**과 **적분 영상**(**integral image**)을 이용하여 이 문제를 해결함

❖ 캐스케이드 분류기와 얼굴 검출 (4/10)

- **에이다부스트 알고리즘**은 수많은 유사-하르 필터 중에서 **얼굴 검출에 효과적인 필터를 선별하는 역할을 수행함**
- 실제 논문에서는 약 6000개의 유사-하르 필터를 선별함
- 이 중 얼굴 검출에 가장 유용하다고 판별된 유사-하르 필터 일부를 그림 13-5에 나타냄

▼ **그림 13-5 얼굴 검출에 유용한 유사-하르 필터의 예**



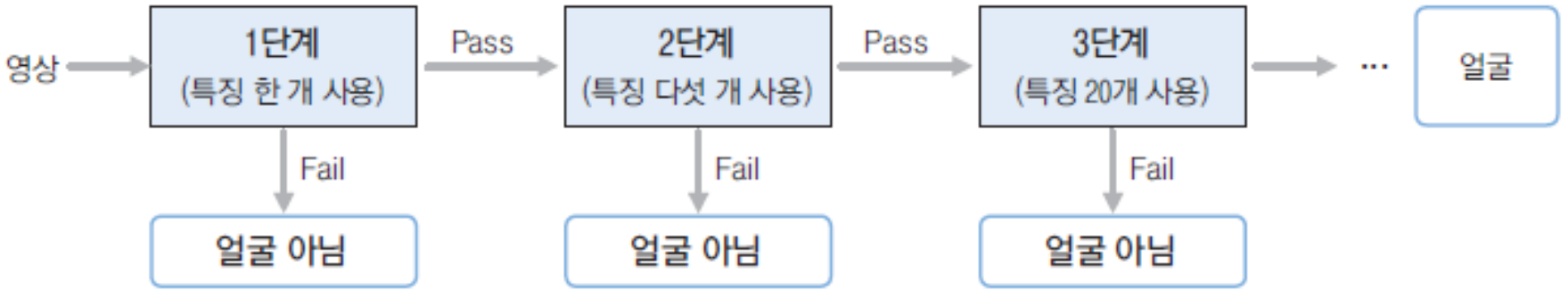
❖ 캐스케이드 분류기와 얼굴 검출 (5/10)

- 에이다부스트 알고리즘에 의해 24×24 부분 영상에서 검사할 특징 개수가 약 6000개로 감소함
- 입력 영상 전체에서 부분 영상을 추출하여 검사해야 하기 때문에 여전히 연산량이 부담될 수 있음
- 나타날 수 있는 얼굴 크기가 다양하기 때문에 보통 입력 영상의 크기를 줄여 가면서 전체 영역에 대한 검사를 다시 수행해야 함
- 비올라와 존스는 대부분의 **영상에 얼굴이 한두 개 있을 뿐이고 나머지 대부분의 영역은 얼굴이 아니라는 점에 주목함**
- 비올라-존스 알고리즘에서는 **캐스케이드**(cascade) **구조라는 새로운 방식을 도입하여 얼굴이 아닌 영역을 빠르게 걸러 내는 방식을 사용함**

❖ 캐스케이드 분류기와 얼굴 검출 (6/10)

- 그림 13-6은 얼굴이 아닌 영역을 걸러 내는 캐스케이드 구조임
- 캐스케이드 구조 1단계에서는 얼굴 검출에 가장 유용한 유사-하르 필터 하나를 사용하여, 얼굴이 아니라고 판단되면 이후의 유사-하르 필터 계산은 수행하지 않음
- 1단계를 통과하면 2단계에서 유사-하르 필터 다섯 개를 사용하여 얼굴이 아닌지를 검사함
- 얼굴이 아니라고 판단되면 이후 단계의 검사는 수행하지 않음
- 이러한 방식으로 **얼굴이 아닌 영역을 빠르게 제거함으로써** 비올라-존스 얼굴 검출 알고리즘은 동시대의 **다른 얼굴 검출 방식보다 약 15배 빠르게 동작하는 성능을 보여줌**

▼ **그림 13-6 캐스케이드 분류기**



❖ 캐스케이드 분류기와 얼굴 검출 (7/10)

- OpenCV는 비올라-존스 알고리즘을 구현하여 객체를 분류할 수 있는 CascadeClassifier 클래스를 제공함
- CascadeClassifier 클래스는 미리 학습된 객체 검출 분류기 XML 파일을 불러오는 기능과 주어진 영상에서 객체를 검출하는 기능으로 이루어져 있음

❖ 캐스케이드 분류기와 얼굴 검출 (8/10)

- CascadeClassifier 클래스를 이용하여 객체를 검출하려면 먼저 CascadeClassifier 객체를 생성해야 함
- CascadeClassifier 객체를 생성한 후에는 미리 학습된 분류기 정보를 불러올 수 있음
- **분류기 정보는 XML 파일 형식으로 저장되어 있음**

❖ 캐스케이드 분류기와 얼굴 검출 (9/10)

- OpenCV는 미리 학습된 얼굴 검출, 눈 검출 등을 위한 분류기 XML 파일을 Github를 통해 제공함

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

- 현재 제공되고 있는 XML 파일 이름과 검출 대상에 대한 설명을 표 13-2에 정리함
- 하나의 검출 대상에 대해 서로 다른 방법으로 학습된 여러 개의 XML 파일이 제공됨

▼ **표 13-2 OpenCV에서 제공하는 유사-하르 기반 분류기 XML 파일**

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

❖ 캐스케이드 분류기와 얼굴 검출 (10/10)

▼ 표 13-2 OpenCV에서 제공하는 유사-하르 기반 분류기 XML

XML 파일 이름	검출 대상
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 분류기를 만드는 절차 (1/3)

- 정면 얼굴 검출을 위한 XML 파일을 준비함
- XML 파일을 이용하여 CascadeClassifier 객체를 생성함
- 다음과 같이 코드를 작성하면 됨

```
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

- 참고로 위 예제 코드에서 사용된 haarcascade_frontalface_default.xml 파일은 프로그램 실행 시 프로그램과 같은 폴더에 있어야 함

❖ 캐스케이드 분류기를 만드는 절차 (2/3)

- XML 파일을 불러오는 코드를 수행한 후에는 XML 분류기 파일이 정상적으로 불러졌는지를 확인하는 것이 좋음
- 이때 사용할 수 있는 메서드가 `empty()` 메서드임
- 이 메서드는 분류기 파일을 정상적으로 불러왔는지를 확인함

`classifier.empty()`	
반환값	분류기 파일을 정상적으로 불러왔으면 `False`, 그렇지 않으면 `True`를 반환합니다.

❖ 캐스케이드 분류기를 만드는 절차 (3/3)

- XML 파일을 정상적으로 불러왔다면 이제 detectMultiScale() 메서드를 이용하여 객체 검출을 실행할 수 있음

classifier.detectMultiScale(image, scaleFactor, minNeighbors)

image	입력 영상
반환값	(출력) 검출된 객체의 사각형 좌표 정보
scaleFactor	검색 윈도우 확대 비율. 1보다 커야 합니다.
minNeighbors	검출 영역으로 선택하기 위한 최소 검출 횟수
minSize	검출할 객체의 최소 크기
maxSize	검출할 객체의 최대 크기

❖ 얼굴 검출 예제 프로그램 (1/5)

- 코드 13-3에 나타난 `detect_face()` 함수는 OpenCV에서 제공하는 haarcascade_frontalface_default.xml 파일을 이용하여 kids.png 영상에서 얼굴을 검출함
- 검출된 얼굴 영역을 화면에 표시함

코드 13-3 얼굴 검출 예제 프로그램 (cascade.py)

```
1   import cv2
2
3
4   def detect_face():
5       src = cv2.imread('kids.png')
6
7       if src is None:
8           print('Image load failed!')
9           return
10
11      classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
12
```

❖ 얼굴 검출 예제 프로그램 (2/5)

코드 13-3 얼굴 검출 예제 프로그램 (cascade.py)

```
13      if classifier.empty():
14          print('XML load failed!')
15          return
16
17      faces = classifier.detectMultiScale(src)
18
19      for (x, y, w, h) in faces:
20          cv2.rectangle(src, (x, y), (x + w, y + h), (255, 0, 255), 2)
21
22      cv2.imshow('src', src)
23      cv2.waitKey()
24      cv2.destroyAllWindows()
25
26  if __name__ == '__main__':
27      detect_face()
```

❖ 얼굴 검출 예제 프로그램 (3/5)

- cascade.py 소스 코드 설명

  - 11행 CascadeClassifier 객체를 생성함과 동시에 haarcascade_frontalface_default.xml 파일을 불러옵니다.
  - 13~15행 분류기를 정상적으로 불러왔는지 확인합니다.
    분류기를 정상적으로 불러오지 못했으면 에러 메시지를 출력하고 함수를 종료합니다.
  - 17행 src 영상에서 얼굴을 검출하여 검출된 사각형 정보를 faces에 저장합니다.
  - 19~20행 검출된 얼굴 영역 사각형을 src 영상에 보라색으로 그립니다.

❖ 얼굴 검출 예제 프로그램 (4/5)

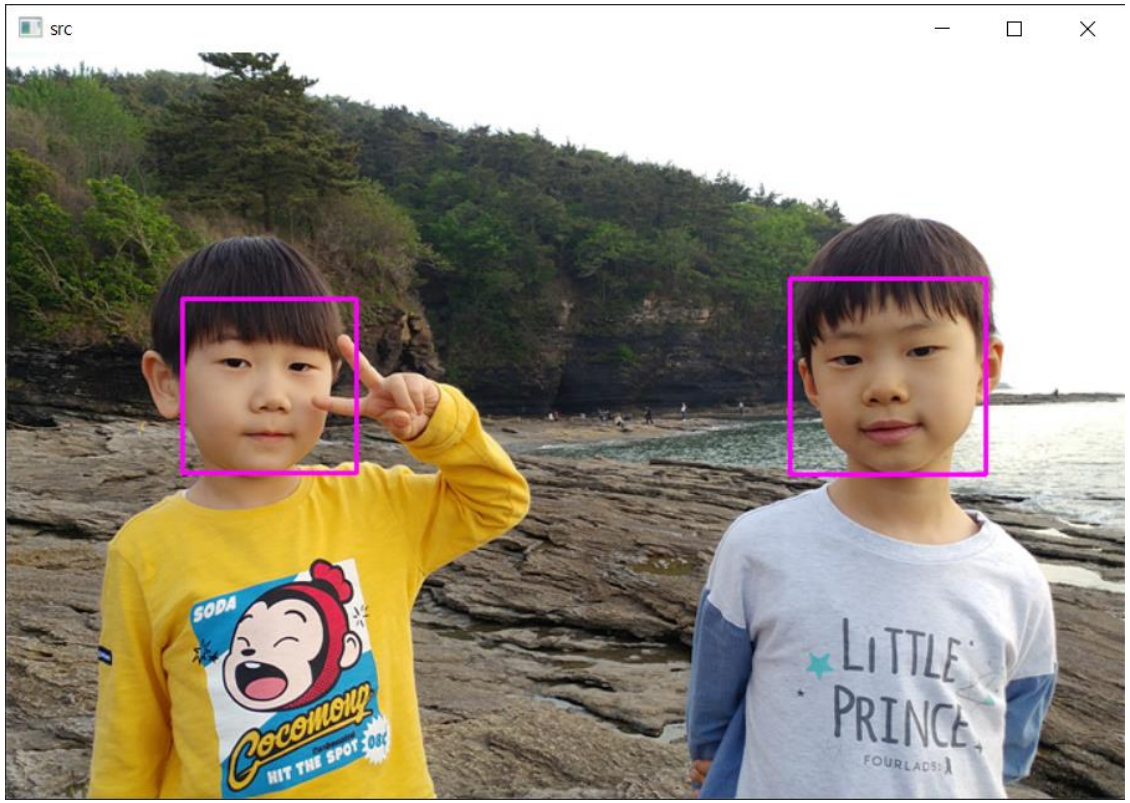
- 코드 13-3의 `detect_face()` 함수는 kids.png 파일과 haarcascade_frontalface_default.xml 파일을 필요로 함
- kids.png 파일과 haarcascade_frontalface_default.xml 파일을 미리 프로젝트 폴더에 복사해두어야 함
- haarcascade_frontalface_default.xml 파일은 OpenCV에서 제공하는 정면 얼굴 검출 분류기 XML 파일임

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

❖ 얼굴 검출 예제 프로그램 (5/5)

- detect_face() 함수를 실행한 결과 화면을 그림 13-7에 나타냄
- 입력 영상으로 사용한 kids.png 파일은 두 명의 아이 얼굴이 들어 있는 영상임

▼ **그림 13-7 얼굴 검출 예제 프로그램 실행 화면**



❖ 눈 검출 예제 프로그램 (1/5)

- 눈을 검출하기 위해서는 먼저 얼굴을 검출하고, 얼굴 영역 안에서만 눈을 검출하는 것이 효율적임
- 눈 검출을 위해 OpenCV가 제공하는 XML 파일 중 haarcascade_eye.xml 파일을 사용할 것이며, 이 파일을 미리 프로젝트 폴더에 복사해야 함
- 입력 영상에서 얼굴을 찾은 후, 눈 위치까지 찾는 예제 프로그램 소스 코드를 코드 13-4에 나타냄

❖ 눈 검출 예제 프로그램 (2/5)

코드 13-4 눈 검출 예제 프로그램 (cascade.py)

```
1  import cv2
2
3
4  def detect_eyes():
5      src = cv2.imread('kids.png')
6
7      if src is None:
8          print('Image load failed!')
9          return
10
11     face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
12     eye_classifier = cv2.CascadeClassifier('haarcascade_eye.xml')
13
14     if face_classifier.empty() or eye_classifier.empty():
15         print('XML load failed!')
16         return
17
```

❖ 눈 검출 예제 프로그램 (3/5)

코드 13-4 눈 검출 예제 프로그램 (cascade.py)

```
18      faces = face_classifier.detectMultiScale(src)
19
20      for (x1, y1, w1, h1) in faces:
21          cv2.rectangle(src, (x1, y1), (x1 + w1, y1 + h1), (255, 0, 255), 2)
22
23          faceROI = src[y1:y1 + h1, x1:x1 + w1]
24          eyes = eye_classifier.detectMultiScale(faceROI)
25
26          for (x2, y2, w2, h2) in eyes:
27              center = (int(x2 + w2 / 2), int(y2 + h2 / 2))
28              cv2.circle(faceROI, center, int(w2 / 2), (255, 0, 0), 2, cv2.LINE_AA)
29
30      cv2.imshow('src', src)
31      cv2.waitKey()
32      cv2.destroyAllWindows()
33
34  if __name__ == '__main__':
35      detect_eyes()
```

❖ 눈 검출 예제 프로그램 (4/5)

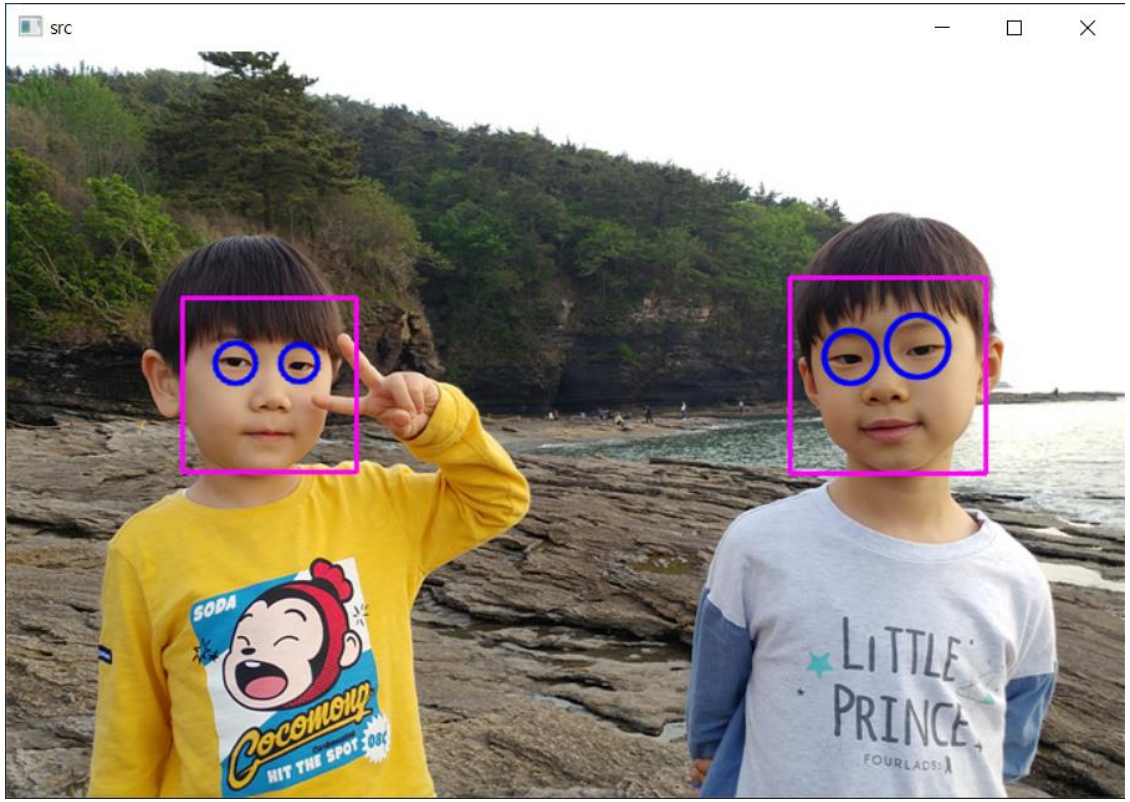
- cascade.py 소스 코드 설명

  - 12행 눈 검출을 위해 haarcasecade_eye.xml 파일을 사용하는 CascadeClassifier 객체를 생성합니다.
  - 23행 입력 영상에서 검출한 사각형 얼굴 영역의 부분 영상을 추출하여 `faceROI`에 저장합니다.
  - 24행 `faceROI` 영상에서 눈을 검출합니다.
  - 26~28행 검출한 눈의 중앙에 파란색 원을 그립니다.
    `faceROI` 영상은 `src` 영상의 부분 영상을 참조하므로 `faceROI`에 원을 그리면 `src`영상에도 원이 그려집니다.

❖ 눈 검출 예제 프로그램 (5/5)

- 앞서 detect_face() 함수에서 검출한 얼굴 영역 안에서 눈을 검출하여 파란색 원으로 나타낸 것을 확인할 수 있음

▼ **그림 13-8 눈 검출 예제 프로그램 실행 결과**



13.3 HOG 알고리즘과 보행자 검출

13.3 HOG 알고리즘과 보행자 검출

❖ HOG 알고리즘과 보행자 검출 (1/14)

- **HOG**(**Histograms of Oriented Gradients**)는 그래디언트 방향 히스토그램을 의미함
- 다랄과 트릭스는 사람이 서 있는 영상에서 그래디언트를 구함[Dalal05]
- 그래디언트의 크기와 방향 성분을 이용하여 사람이 서 있는 형태에 대한 특징 벡터를 정의함
- 머신 러닝의 일종인 서포트 벡터 머신(SVM, Support Vector Machine) 알고리즘을 이용하여 입력 영상에서 보행자 위치를 검출하는 방법을 제안함

**[Viola01] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection,"
in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2005, pp. 886-893.**

❖ HOG 알고리즘과 보행자 검출 (2/14)

- 보행자 검출을 위한 HOG는 기본적으로 64×128 크기의 영상에서 계산함
- 그림 13-9(a)는 64×128 크기의 입력 영상을 확대하여 나타낸 그림임
- HOG 알고리즘은 먼저 입력 영상으로부터 그래디언트를 계산함
- 그래디언트는 크기와 방향 성분으로 계산하며, 방향 성분은 0°부터 180°까지로 설정함
- 그 다음은 입력 영상을 8×8 크기 단위로 분할하는데, 각각의 8×8 부분 영상을 셀(cell)이라고 부름
- 64×128 영상에서 셀은 가로 방향으로 여덟 개, 세로 방향으로 16개 생성됨
- 각각의 셀로부터 그래디언트 방향 성분에 대한 히스토그램을 구함
- 이때 방향 성분을 20° 단위로 구분하면 총 아홉 개의 빈으로 구성된 방향 히스토그램이 만들어짐
- 인접한 네 개의 셀을 합쳐서 블록(block)이라고 정의함

❖ HOG 알고리즘과 보행자 검출 (3/14)

- 그림 13-9(b)에서 노란색 실선은 셀을 구분하는 선이고, 빨간색 사각형은 블록 하나를 나타냄
- 하나의 블록에는 네 개의 셀이 있고 각 셀에는 아홉 개의 빈으로 구성된 히스토그램 정보가 있음
- 블록 하나에서는 총 36개의 실수 값으로 이루어진 방향 히스토그램 정보가 추출됨
- 블록은 가로와 세로 방향으로 각각 한 개의 셀만큼 이동하면서 정의함
- 64×128 영상에서 블록은 가로 방향으로 일곱 개, 세로 방향으로 15개 정의할 수 있음
- 결국 64×128 영상에서 105개의 블록이 추출될 수 있고, 전체 블록에서 추출되는 방향 히스토그램 실수 값 개수는 105×36=3780이 됨
- 이 3780개의 실수 값이 64×128 영상을 표현하는 HOG 특징 벡터 역할을 함
- 그림 13-9(c)는 각 셀에서 계산된 그래디언트 방향 히스토그램을 비주얼하게 표현한 결과임

❖ HOG 알고리즘과 보행자 검출 (4/14)

▼ 그림 13-9 HOG 알고리즘

❖ HOG 알고리즘과 보행자 검출 (5/14)

- 다랄과 트릭스는 수천 장의 보행자 영상과 보행자가 아닌 영상에서 HOG 특징 벡터를 추출함
- 이 두 특징 벡터를 구분하기 위해 SVM 알고리즘을 사용함
- SVM은 두 개의 클래스를 효과적으로 분리하는 능력을 가진 머신 러닝 알고리즘임
- 다랄과 트릭스는 수천 개의 보행자 특징 벡터와 보행자가 아닌 특징 벡터를 이용하여 SVM을 학습시킴
- 효과적인 보행자 검출 방법을 완성시킴

❖ HOG 알고리즘과 보행자 검출 (6/14)

- HOG와 SVM을 이용한 객체 검출 기술은 이후 보행자 검출뿐만 아니라 다양한 형태의 객체 검출에서도 응용됨
- **OpenCV는 HOG 알고리즘을 구현한 HOGDescriptor 클래스를 제공함**
- HOGDescriptor 클래스를 이용하면 특정 객체의 HOG 기술자를 쉽게 구할 수 있음
- HOGDescriptor 클래스는 보행자 검출을 위한 용도로 미리 계산된 HOG 기술자 정보를 제공함

❖ HOG 알고리즘과 보행자 검출 (7/14)

- HOGDescriptor 클래스를 이용하려면 먼저 HOGDescriptor 객체를 생성해야 함
- 보행자 검출이 목적이라면 HOGDescriptor 클래스의 기본 생성자를 이용하여 객체를 생성하면 됨
- HOGDescriptor 클래스의 기본 생성자는 검색 윈도우 크기를 64×128로 설정함
- 셀 크기는 8×8, 블록 크기는 16×16, 그래디언트 방향 히스토그램 빈 개수는 9로 설정함
- 기본 생성자에 의해 만들어지는 HOG 기술자 하나는 3780개의 float 실수로 구성됨
- 다음은 보행자 검출을 목적으로 HOGDescriptor 클래스 객체 hog를 선언하는 예제 코드임

```
hog = cv2.HOGDescriptor()
```

❖ HOG 알고리즘과 보행자 검출 (8/14)

- HOGDescriptor 클래스는 미리 계산된 보행자 검출을 위한 HOG 기술자 정보를 반환하는 `HOGDescriptor_getDefaultPeopleDetector()` 메서드를 제공함

cv2.HOGDescriptor_getDefaultPeopleDetector()	
반환값	보행자 검출을 위해 학습된 분류기 계수

❖ HOG 알고리즘과 보행자 검출 (9/14)

- HOGDescriptor 클래스를 이용하여 원하는 객체를 검출하려면 먼저 검출할 객체에 대해 학습된 SVM 분류기 계수를 `hog.setSVMDetector()` 메서드에 등록해야 함

`hog.setSVMDetector(svmdetector)`	
`svmdetector`	선형 SVM 분류기를 위한 계수

❖ HOG 알고리즘과 보행자 검출 (10/14)

- HOG 기술자를 이용하여 실제 입력 영상에서 객체 영역을 검출하려면 `hog.detectMultiScale()` 메서드를 사용함

```
detected, _ = hog.detectMultiScale(frame)
```

`frame`	입력 영상. CV_8UC1 또는 CV_8UC3
`detected`	(출력) 검출된 사각형 영역 정보

❖ HOG 알고리즘과 보행자 검출 (11/14)

- 코드 13-5에 나타난 보행자 검출 예제 프로그램은 HOGDescriptor 클래스가 제공하는 보행자 검출 HOG 정보를 이용하여 동영상 매 프레임에서 보행자를 검출하고, 그 결과를 화면에 표시함

코드 13-5 보행자 검출 예제 프로그램 (hog.py)

```
1   import sys
2   import numpy as np
3   import cv2
4   import random
5
6
7   cap = cv2.VideoCapture('vtest.avi')
8
9   if not cap.isOpened():
10      print('Video open failed!’)
11      sys.exit()
12
13  hog = cv2.HOGDescriptor()
14  hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
15
```

❖ HOG 알고리즘과 보행자 검출 (12/14)

코드 13-5 보행자 검출 예제 프로그램 (hog.py)

```
16  while True:
17      ret, frame = cap.read()
18
19      if not ret:
20          break
21
22      detected, _ = hog.detectMultiScale(frame)
23
24      for (x, y, w, h) in detected:
25          c = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
26          cv2.rectangle(frame, (x, y), (x + w, y + h), c, 3)
27
28      cv2.imshow('frame', frame)
29      if cv2.waitKey(10) == 27:    // 10진수 10은 ESC Key를 의미함
30          break
31
32  cv2.destroyAllWindows()
```

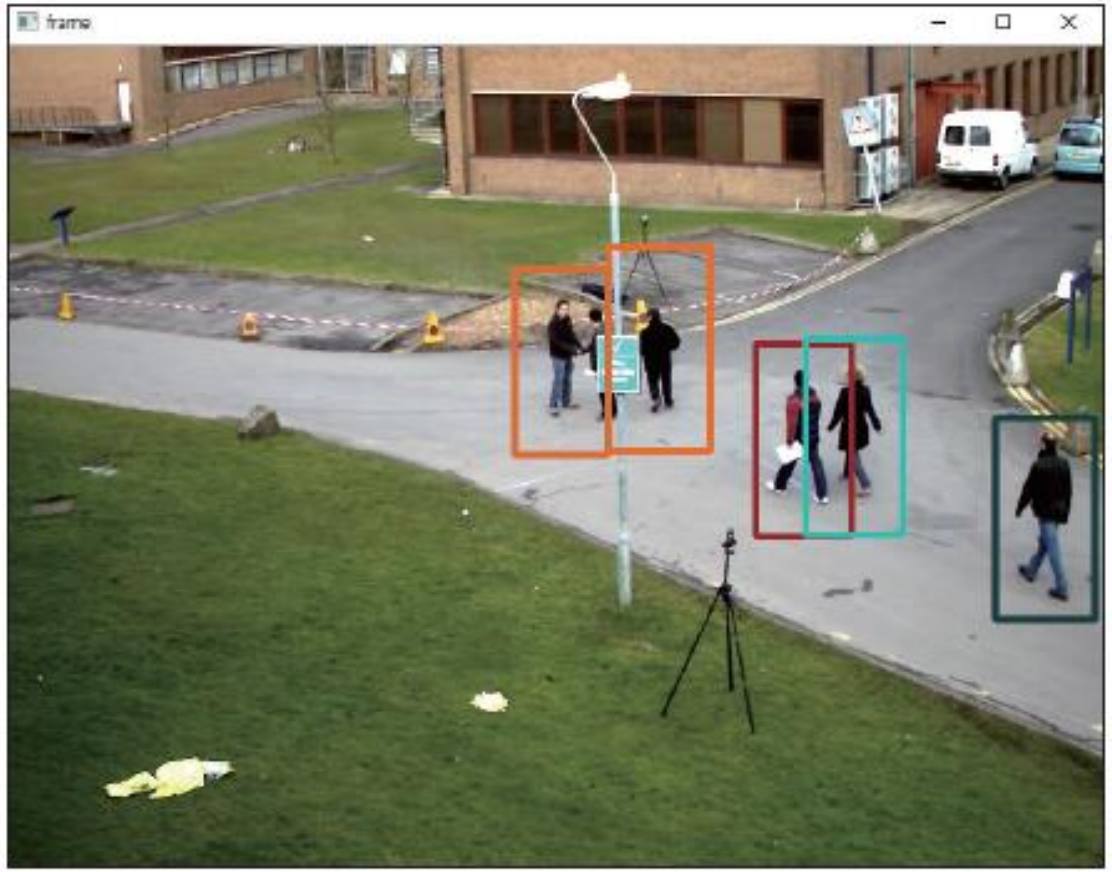
❖ HOG 알고리즘과 보행자 검출 (13/14)

- hog.py 소스 코드 설명

  - 7행 현재 폴더에서 vtest.avi 파일을 불어옵니다.
    프로그램 시작 전에 vtest.avi 파일을 미리 현재 폴더로 복사해야 합니다.
  - 13행 HOGDescriptor 객체 hog를 선언합니다.
  - 14행 보행자 검출을 위한 용도로 학습된 SVM 분류기 계수를 등록합니다.
  - 22행 동영상 매 프레임마다 보행자 검출을 수행합니다.
    검출된 사각형 정보는 `detected` 변수에 저장됩니다.
  - 24~26행 검출된 사각형 정보를 이용하여 임의의 색상으로 3픽셀 두께의 사각형을 그립니다.

❖ HOG 알고리즘과 보행자 검출 (14/14)

▼ 그림 13-10 보행자 검출 예제 프로그램 실행 화면



13.4 QR 코드 검출

13.4 QR 코드 검출

❖ QR 코드 검출 (1/8)

- QR 코드는 흑백 격자 무늬 모양의 2차원 바코드 일종으로 숫자, 영문자, 8비트 문자, 한자 등의 정보를 저장할 수 있음
- 최근에는 명함이나 광고 전단 등에 웹 사이트 URL 문자열을 포함한 QR 코드를 프린트하여 사용자가 스마트폰의 QR 코드 앱을 통해 해당 웹 사이트에 쉽게 접속할 수 있도록 하는 서비스가 늘어나고 있음

▼ 그림 13-11 QR 코드의 예

❖ QR 코드 검출 (2/8)

- 입력 영상에서 QR 코드를 인식하려면 먼저 QR 코드 세 모서리에 포함된 흑백 정사각형 패턴을 찾아 QR 코드 전체 영역 위치를 알아내야 함
- 검출된 QR 코드를 정사각형 형태로 투시 변환함
- QR 코드 내부에 포함된 흑백 격자 무늬를 해석하여 문자열을 추출해야 함
- 일련의 연산은 매우 복잡하고 정교한 영상 처리를 필요로 함
- **다행히 OpenCV는 4.0.0 버전부터 QR 코드를 검출하고 QR 코드에 포함된 문자열을 해석하는 기능을 제공함**

❖ QR 코드 검출 (3/8)

- OpenCV에서 QR 코드를 검출하고 해석하는 기능은 QRCodeDetector 클래스에 구현되어 있음
- QRCodeDetector 클래스를 이용하여 영상에서 QR 코드를 검출하거나 해석하려면 먼저 QRCodeDetector 객체를 생성해야 함
- QRCodeDetector 객체는 단순히 QRCodeDetector 클래스 타입의 변수를 하나 선언하는 방식으로 생성할 수 있음
- 다음은 QRCodeDetector 타입의 변수 detector를 선언하는 예제 코드임

```
detector = cv2.QRCodeDetector()
```

❖ QR 코드 검출 (4/8)

- QRCodeDetector 객체를 생성한 후에는 QRCodeDetector 클래스 메서드를 이용하여 QR 코드를 검출하거나 문자열을 해석할 수 있음
- 먼저 입력 영상에서 QR 코드 영역을 검출하고 검출된 QR 코드 영역에서 QR 코드에 저장된 문자열을 추출하기 위해서 `detectAndDecode()` 메서드를 사용함

입력 영상에서 QR 코드 검출과 해석을 한꺼번에 수행

```
info, points, _ = detectAndDecode(frame)
```

`frame`	입력 영상. CV_8U 또는 CV_8UC3
`info`	QR 코드에 포함된 문자열
`points`	QR 코드를 감싸는 사각형의 네 꼭지점 좌표

❖ QR 코드 검출 (5/8)

- 코드 13-6에서는 컴퓨터에 연결된 카메라로부터 들어오는 매 프레임마다 QR 코드를 검출함
- 검출된 QR 코드 사각형 영역과 QR 코드에 포함된 문자열을 화면에 함께 표시함

코드 13-6 QR 코드 검출 및 해석 예제 프로그램 (QRCode.py)

```
1   import sys
2   import numpy as np
3   import cv2
4
5
6   cap = cv2.VideoCapture(0)
7
8   if not cap.isOpened():
9       print('Video open failed!')
10      sys.exit()
11
12  detector = cv2.QRCodeDetector()
13
```

❖ QR 코드 검출 (6/8)

코드 13-6 QR 코드 검출 및 해석 예제 프로그램 (QRCode.py)

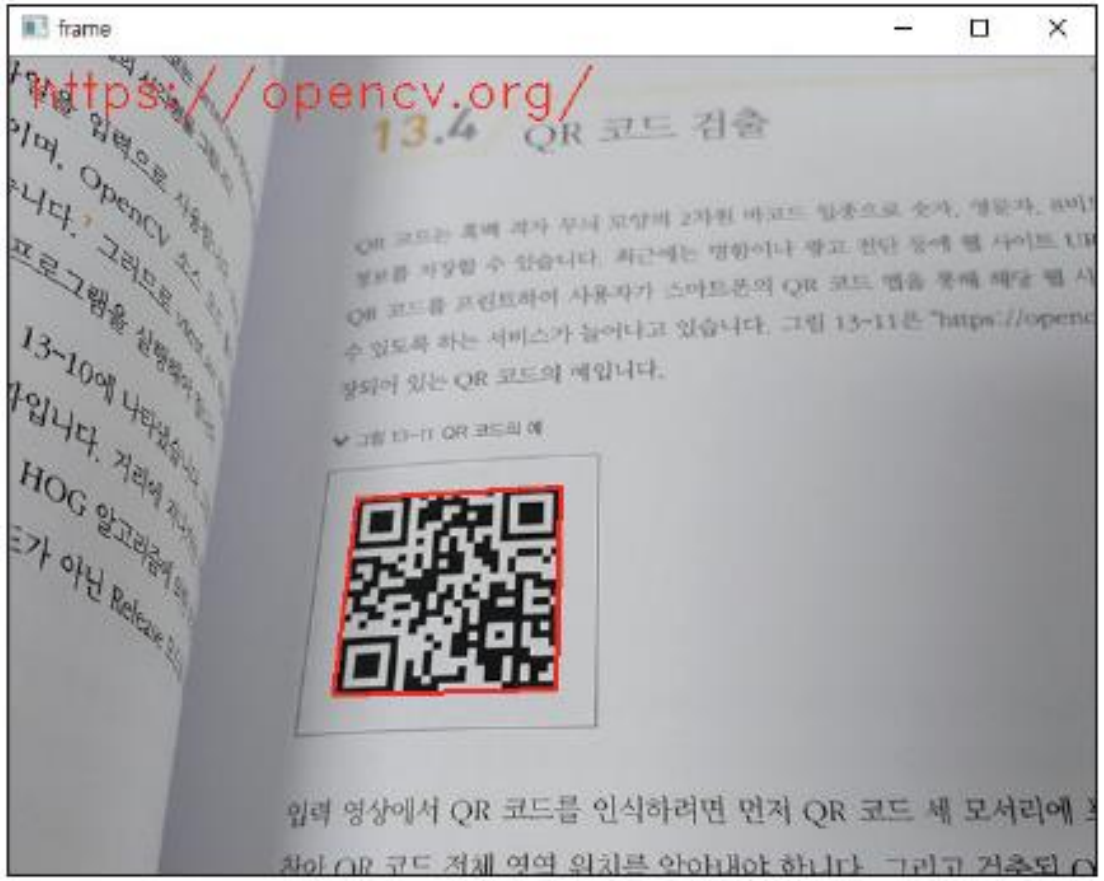
```
14  while True:
15      ret, frame = cap.read()
16      if not ret:
17          print('Frame load failed!’)
18          break
19
20      info, points, _ = detector.detectAndDecode(frame)
21      if points is not None:
22          points = np.array(points, dtype=np.int32).reshape(4, 2)
23          cv2.polylines(frame, [points], True, (0, 0, 255), 2)
24
25      if len(info) > 0:
26          cv2.putText(frame, info, (10, 30), cv2.FONT_HERSHEY_DUPLEX, 1, (0, 0, 255), lineType=cv2.LINE_AA)
27
28      cv2.imshow('frame', frame)
29      if cv2.waitKey(1) == 27:
30          break
31
32  cv2.destroyAllWindows()
```

❖ QR 코드 검출 (7/8)

- QRCode.py 소스 코드 설명
  - 6행 컴퓨터에 연결된 기본 카메라를 이용하여 VideoCapture 객체 `cap`을 생성합니다.
  - 12행 QRCodeDetector 객체 `detector` 변수를 선언합니다.
  - 20행 카메라 매 프레임마다 QR 코드 검출 및 해석을 수행합니다.
  - 21~23행 만약 QR 코드를 검출하고 QR 코드에 빨간색 사각형을 그립니다.
  - 25~26행 QR 코드 문자열이 `info` 변수에 저장되었다면, 해석된 문자열을 화면 좌측 상단에 빨간색 글자로 출력합니다.
  - 29~30행 ESC Key를 누르면 `while` 반복문을 빠져나오고 프로그램이 종료됩니다.

❖ QR 코드 검출 (8/8)

▼ 그림 13-12 QR 코드 검출 및 해석 예제 프로그램 실행 결과



THANK YOU!
Q & A

- Name: 권범
- Office: 동양미래대학교 2호관 704호 (02-2610-5238)
- E-mail: bkwon@dongyang.ac.kr
- Homepage: https://sites.google.com/view/beomkwon/home