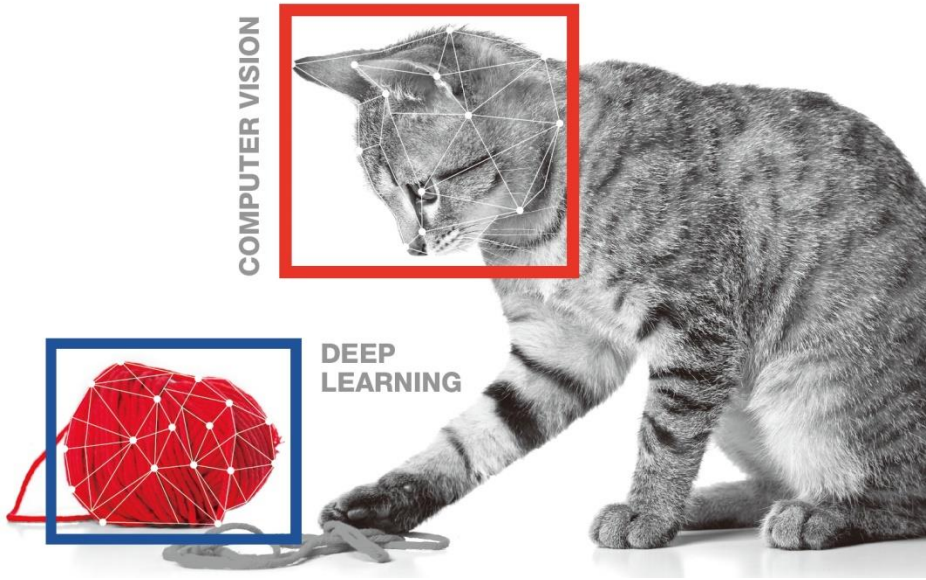
컴퓨터 비전과 딥러닝

[강의교안 이용 안내]



컴퓨터 비전과 딥러닝

Chapter 10 동적 비전

차례

Preview

■ 사람의 시각은 동적

- 아이는 연속 영상을 통해 물체가 취하는 무수히 많은 자세를 매 순간 학습 ← 사람의 비전이 심하게 변하는 환경에 강인한 이유

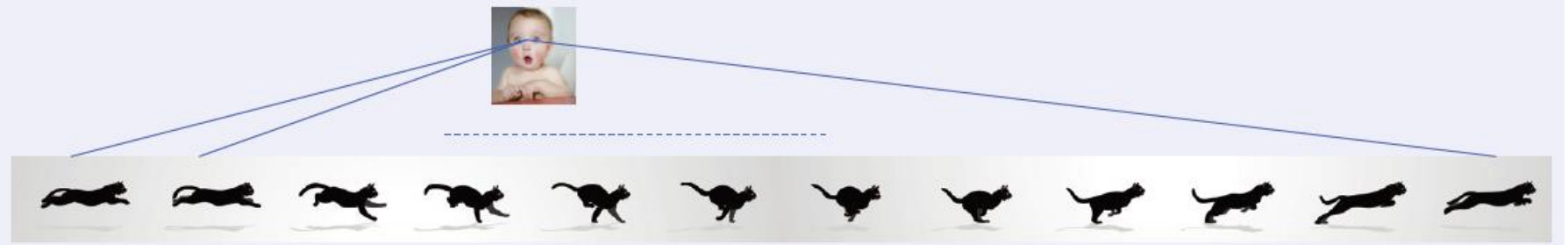


그림 10-1 사람의 동적 비전

■ 비디오는 일상

- 전세계에 설치된 10억 대의 감시 카메라
- 2019년 기준 분당 500시간 분량의 비디오가 유튜브에 업로드
- 비디오를 자동으로 인식할 수 있다면 무궁무진한 응용

10.1 모션 분석

■ 비디오(동영상)

- 시간 순서에 따라 정지 영상을 나열한 구조
- 보통 초당 30프레임. 초당 수백~수천 장의 빠른 비디오와 수분에 한 장 입력하는 느린 비디오
- 3차원 시공간$_{\text{spatio-temporal}}$ 형성
- 컬러 영상은 $m \times n \times 3$의 3차원 텐서, 비디오는 $m \times n \times 3 \times T$의 4차원 텐서

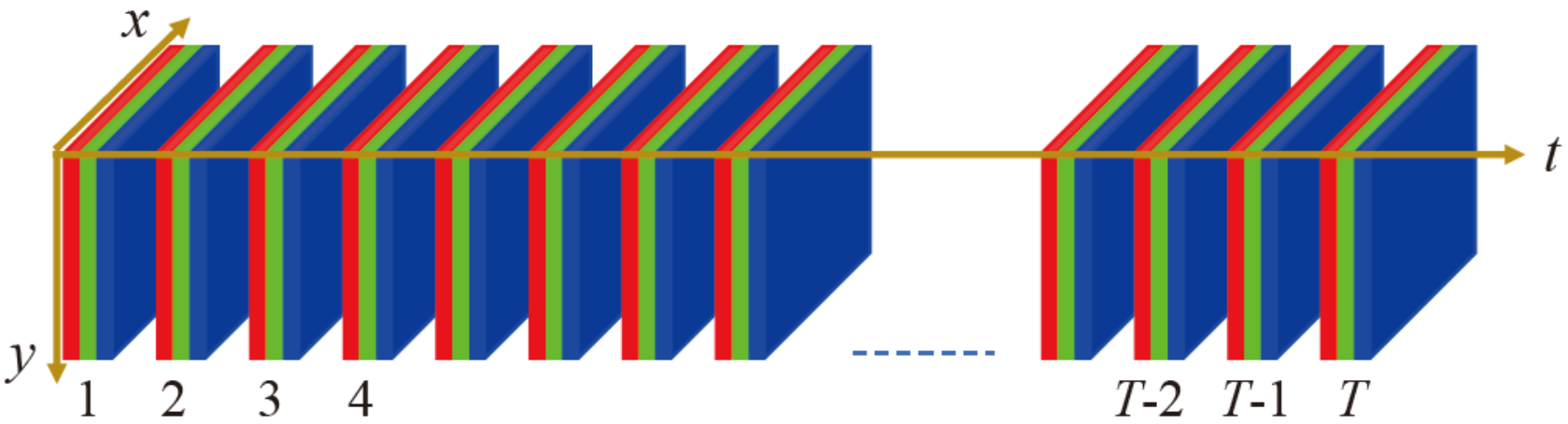


그림 10-2 3차원 공간에 표현되는 비디오

10.1 모션 분석

■ 초기 연구

- 카메라와 조명, 배경이 고정된 단순한 상황을 가정하고 차영상difference image 이용

$$d(j,i,t) = |f(j,i,0) - f(j,i,t)|, 0 \le j < m, 0 \le i < n, \ 1 \le t \le T \qquad (10.1)$$

기준 프레임(배경만 두고 촬영)

■ 이후

- 일반적인 비디오 처리로 발전. 초창기에는 주로 광류를 이용한 방법이 주류
- 이후 딥러닝 시대로 전환

10.1.1 모션 벡터와 광류

■ 광류optical flow

- 움직이는 물체는 명암 변화 일으킴
- 명암 변화를 분석하면 역으로 물체의 모션 정보 알아낼 수 있음
- 화소별로 모션 벡터를 추정해 기록한 맵을 광류라고 함

■ 모션 벡터 추정의 애매함

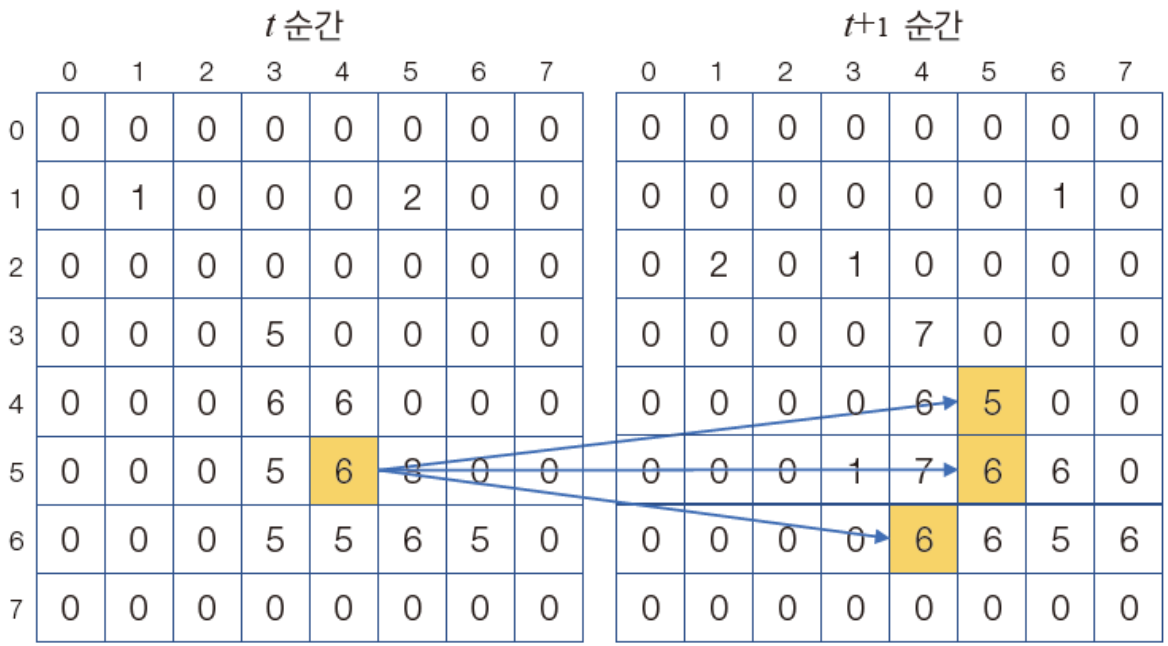


그림 10-3 연속 영상에서 모션 벡터를 추정할 때 발생하는 애매함

10.1.1 모션 벡터와 광류(optical flow)

■ 광류 추정 방법

- 밝기 항상성 가정(연속한 영상에서 같은 물체는 같은 명암으로 나타남)
- dt가 충분히 작다고 가정하면 테일러 급수에 따라 식 (10.2) 성립(비디오에서는 dt=1/30초로서 충분히 작음)

$$f(y+dy, x+dy, t+dt) = f(y,x,t) + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial t}dt + 2\text{차 이상의 합} \qquad (10.2)$$

- 2차 항 무시하고 밝기 항상성 적용하면

$$\frac{\partial f}{\partial y}\frac{dy}{dt} + \frac{\partial f}{\partial x}\frac{dy}{dt} + \frac{\partial f}{\partial t} = 0 \qquad (10.3)$$

- 모션 벡터 $\mathbf{v} = (v,u)$로 대치하면

$$\frac{\partial f}{\partial y}v + \frac{\partial f}{\partial x}u + \frac{\partial f}{\partial t} = 0 \qquad (10.4)$$ 광류 조건식(대부분 알고리즘이 사용하는 일반 공식)

10.1.1 모션 벡터와 광류

■ [예시 10-1] 광류 조건식

- 단순함을 위해 $\frac{\partial f}{\partial y}$와 $\frac{\partial f}{\partial x}$를 이웃 화소와 명암 차이로 계산하면

$$\frac{\partial f}{\partial y} = f(y+1,x,t) - f(y,x,t),\ \frac{\partial f}{\partial x} = f(y,x+1,t) - f(y,x,t),\ \frac{\partial f}{\partial t} = f(y,x,t+1) f(y,x,t)$$

- [그림 10-3]의 노란 표시된 화소에 위 식을 적용하면

$$\frac{\partial f}{\partial y} = f(6,4,t) - f(5,4,t) = -1,\ \frac{\partial f}{\partial x} = f(5,5,t) - f(5,4,t) = 2,\ \frac{\partial f}{\partial t} = f(5,4,t+1) - f(5,4,t) = 1$$

- 이들 값을 광류 조건식에 대입하면

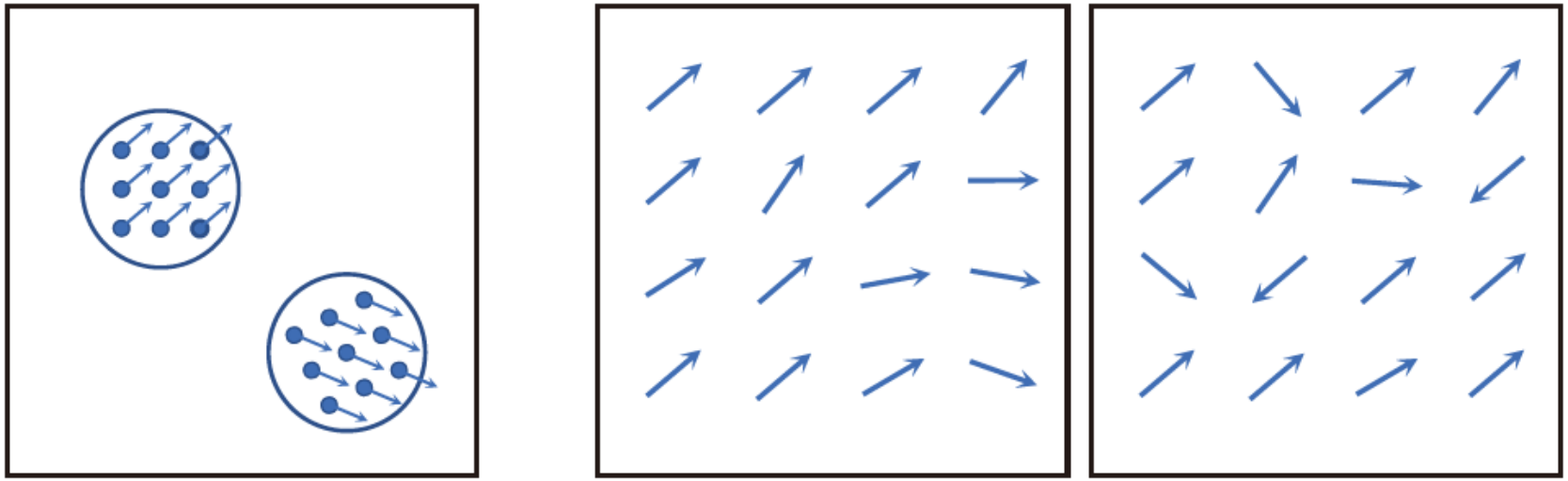
$$-v + 2u + 1 = 0$$

유일한 해를 확정하지 못함(단지 모션 벡터 $\mathbf{v} = (v,u)$가 만족해야 할 조건식 제공)

10.1.1 모션 벡터와 광류

■ 추가 가정을 통한 광류 계산

- Lucas-Kanade는 이웃 화소는 유사한 모션 벡터를 가져야 한다는 지역 조건 사용
- Horn-Schunck는 영상 전체에 걸쳐 모션 벡터는 천천히 변해야 한다는 광역 조건 사용



(a) Lucas-Kanade 알고리즘의 지역 조건 (b) Horn-Schunck 알고리즘의 광역 조건

그림 10-4 광류 추정 알고리즘이 사용하는 가정

10.1.1 모션 벡터와 광류

■ Farneback 알고리즘으로 광류 추정

- OpenCV는 Lucas-Kanade와 Horn-Schunck 알고리즘을 개선한 Farneback 알고리즘을 구현한 calcOpticalFlowFarneback 함수 제공

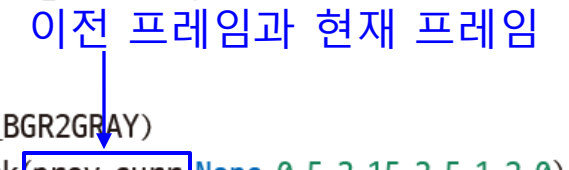
프로그램 10-1 Farneback 알고리즘으로 광류 추정하기

```
01  import numpy as np
02  import cv2 as cv
03  import sys
04
05  def draw_OpticalFlow(img,flow,step=16):
06      for y in range(step//2,frame.shape[0],step):
07          for x in range(step//2,frame.shape[1],step):
08              dx,dy=flow[y,x].astype(np.int)
09              if(dx*dx+dy*dy)>1:
10                  cv.line(img,(x,y),(x+dx,y+dy),(0,0,255),2) # 큰 모션 있는 곳은 빨간색
11              else:
12                  cv.line(img,(x,y),(x+dx,y+dy),(0,255,0),2)
13
14  cap=cv.VideoCapture(0,cv.CAP_DSHOW)       # 카메라와 연결 시도
15  if not cap.isOpened(): sys.exit('카메라 연결 실패')
```

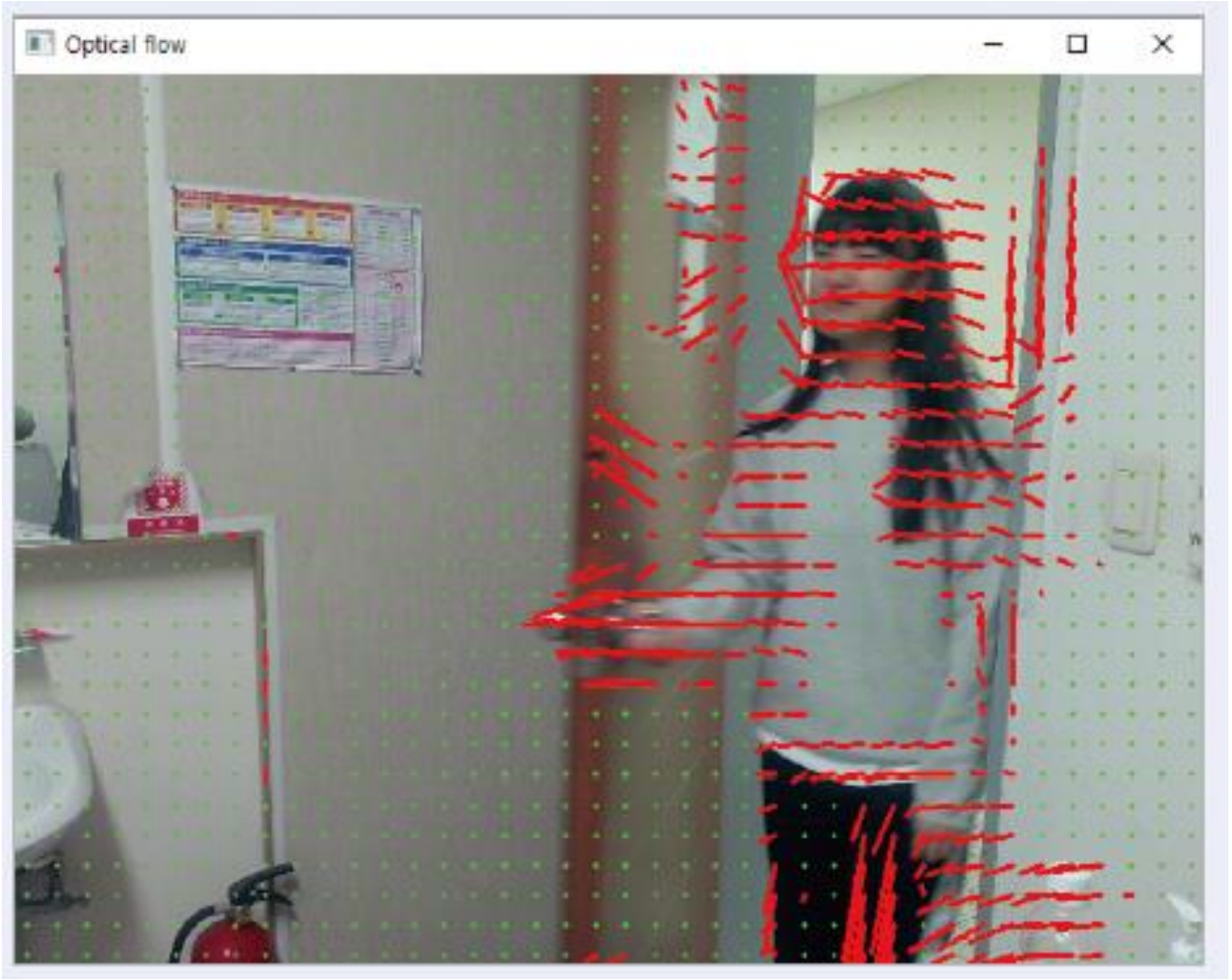
추정한 광류를 영상에 표시하는 함수

10.1.1 모션 벡터와 광류

```
16
17  prev=None
18
19  while(1):
20      ret,frame=cap.read()                    # 비디오를 구성하는 프레임 획득
21      if not ret: sys('프레임 획득에 실패하여 루프를 나갑니다.')
22
23      if prev is None:                        # 첫 프레임이면 광류 계산 없이 prev만 설정
24          prev=cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
25          continue
26
27      curr=cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
28      flow=cv.calcOpticalFlowFarneback(prev,curr,None,0.5,3,15,3,5,1.2,0)
29
30      draw_OpticalFlow(frame,flow)
31      cv.imshow('Optical flow',frame)
32
33      prev=curr
34
35      key=cv.waitKey(1)                       # 1밀리초 동안 키보드 입력 기다림
36      if key==ord('q'):                       # 'q' 키가 들어오면 루프를 빠져나감
37          break
38
39  cap.release()                               # 카메라와 연결을 끊음
40  cv.destroyAllWindows()
```



10.1.1 모션 벡터와 광류



10.1.1 모션 벡터와 광류

■ 희소 광류 추정을 이용한 KLT 추적

- KLT 추적은 고전 추적 알고리즘 중에 가장 유명
- 광류 정보를 이용하여 지역 특징을 추적하는 방식(지역 특징은 4장 알고리즘을 추적에 유리하도록 개조)
- 지역 특징에서만 모션 벡터를 추정하기 때문에 희소 광류라 부름
- 큰 이동을 처리하기 위해 피라미드 영상 활용
- OpenCV는 지역 특징을 추출하는 goodFeaturesToTrack 함수와 광류와 추적 정보를 계산하는 calcOpticalFlowPyrLK 함수 제공

10.1.1 모션 벡터와 광류

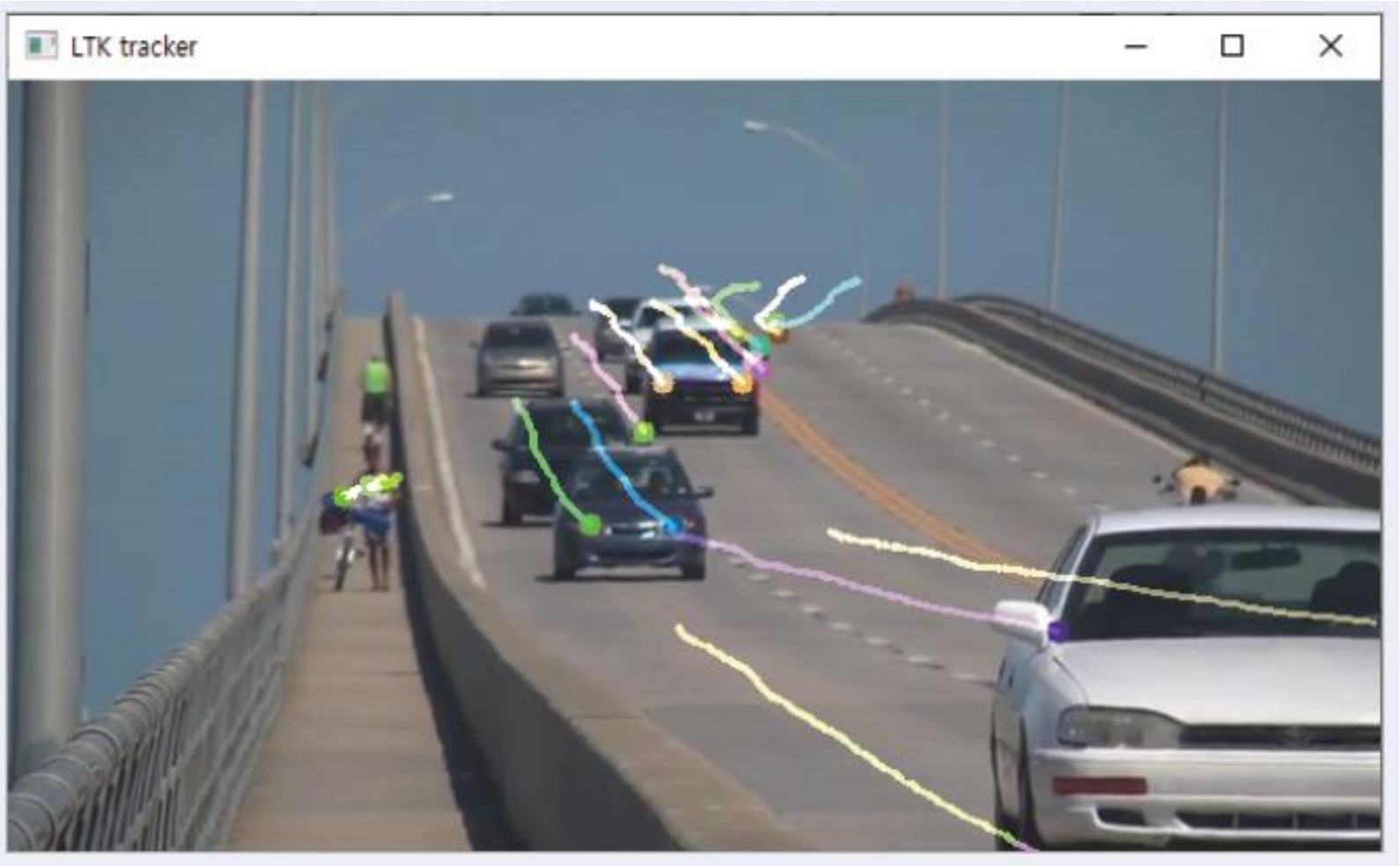
프로그램 10-2 KLT 추적 알고리즘으로 물체 추적하기

```
01  import numpy as np
02  import cv2 as cv
03
04  cap=cv.VideoCapture('slow_traffic_small.mp4')
05
06  feature_params=dict(maxCorners=100,qualityLevel=0.3,minDistance=7,blockSize=7)
07  lk_params=dict(winSize=(15,15),maxLevel=2,criteria=(cv.TERM_CRITERIA_EPS|cv.
    TERM_CRITERIA_COUNT,10,0.03))
08
09  color=np.random.randint(0,255,(100,3))
10
11  ret,old_frame=cap.read()                    # 첫 프레임
12  old_gray=cv.cvtColor(old_frame,cv.COLOR_BGR2GRAY)
13  p0=cv.goodFeaturesToTrack(old_gray,mask=None,**feature_params)
14
15  mask=np.zeros_like(old_frame)               # 물체의 이동 궤적을 그릴 영상
16
```

10.1.1 모션 벡터와 광류

```
while(1):
    ret,frame=cap.read()
    if not ret: break

    new_gray=cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
    p1,match,err=cv.calcOpticalFlowPyrLK(old_gray,new_gray,p0,None,**lk_
    params)                                    # 광류 계산

    if p1 is not None:                         # 양호한 쌍 선택
        good_new=p1[match==1]
        good_old=p0[match==1]

    for i in range(len(good_new)):             # 이동 궤적 그리기
        a,b=int(good_new[i][0]),int(good_new[i][1])
        c,d=int(good_old[i][0]),int(good_old[i][1])
        mask=cv.line(mask,(a,b),(c,d),color[i].tolist(),2)
        frame=cv.circle(frame,(a,b),5,color[i].tolist(),-1)

    img=cv.add(frame,mask)
    cv.imshow('LTK tracker',img)
    cv.waitKey(30)

    old_gray=new_gray.copy()                   # 이번 것이 이전 것이 됨
    p0=good_new.reshape(-1,1,2)

cv.destroyAllWindows()
```

10.1.1 모션 벡터와 광류



10.1.2 딥러닝 기반 광류 추정

■ 데이터셋

- 광류는 매 화소마다 모션 벡터를 지정해야 하므로 참값 레이블링이 어려움
- KITTI 데이터셋은 차량에 설치된 장치를 통해 수집(자율주행을 위해 제작)
- Sintel 데이터셋은 컴퓨터그래픽으로 제작한 애니메이션 대상으로 참값 레이블링
- Flying chairs 데이터셋은 실제 영상에 그래픽 의자를 인위적으로 추가해 참값 레이블링

(a) KITTI 데이터셋

(b) Sintel 데이터셋

(c) Flying chairs 데이터셋

그림 10-5 광류 데이터셋[Dosovitskiy2015]

10.1.2 딥러닝 기반 광류 추정

■ 빠르게 딥러닝 시대로 전환

- FlowNet은 컨볼루션 신경망을 광류 추정에 적용한 최초 논문
  - [그림 9-35]의 DeConvNet같은 분할용 신경망을 사용
  - 연속된 2장의 영상을 이어 붙인 384*512*6 텐서를 입력 또는 두 개의 신경망에 따로 입력하고 2개의 특징 맵을 결합해 사용
- 분할 알고리즘의 출력을 활용하는 아이디어. 예) Bai 기법

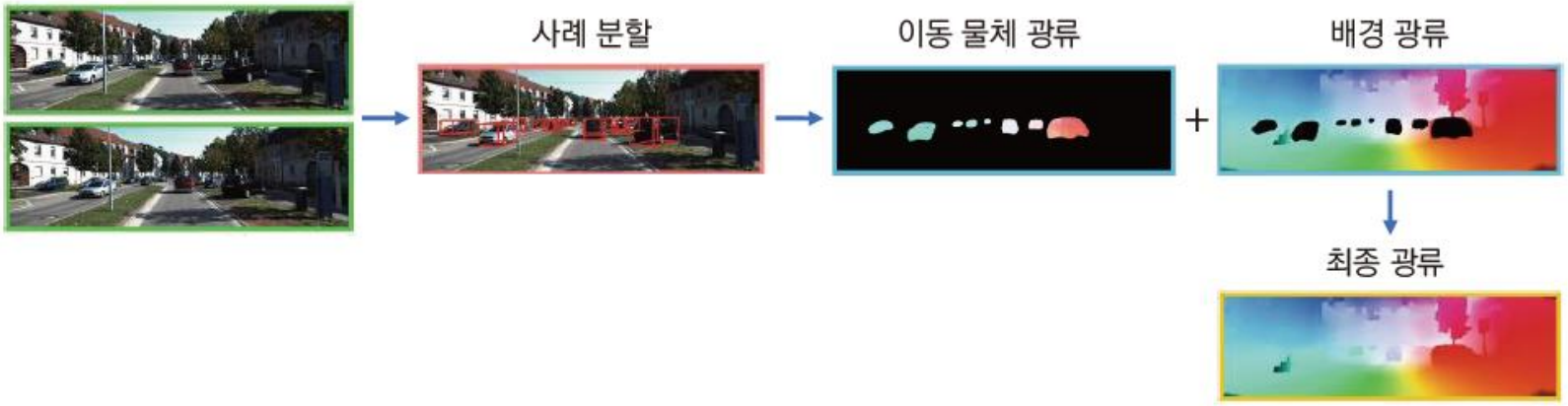
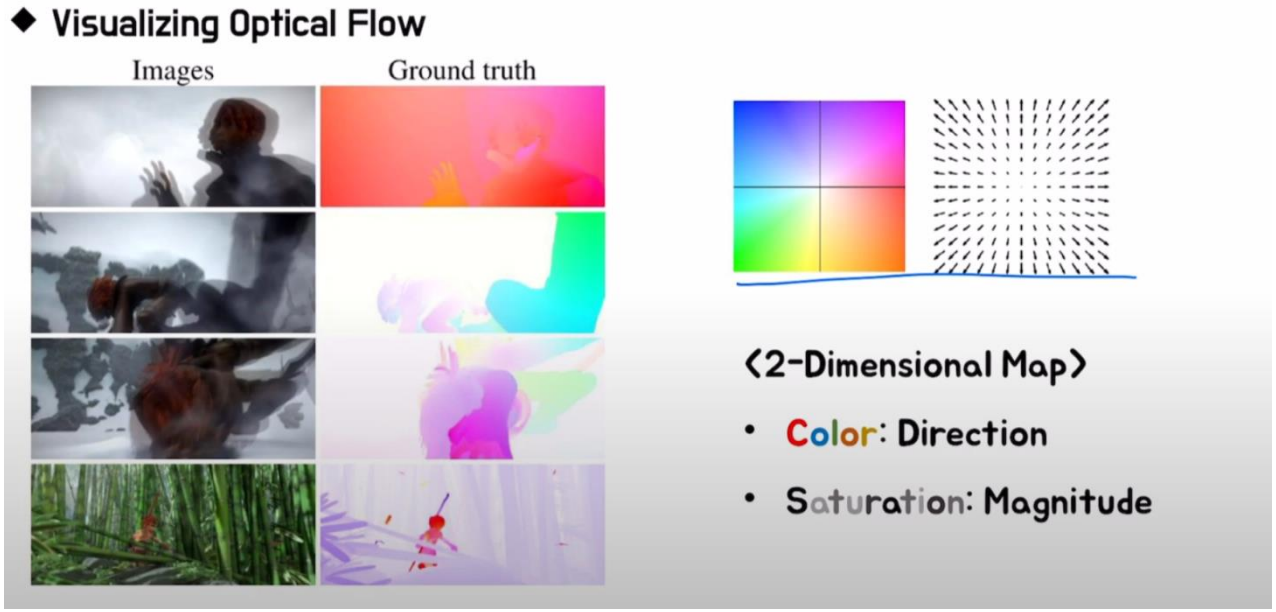


그림 10-6 사례 분할을 활용한 광류 추정[Bai2016]

- 비용 볼륨을 활용하는 RAFT
- RAFT와 트랜스포머를 결합한 FlowFormer

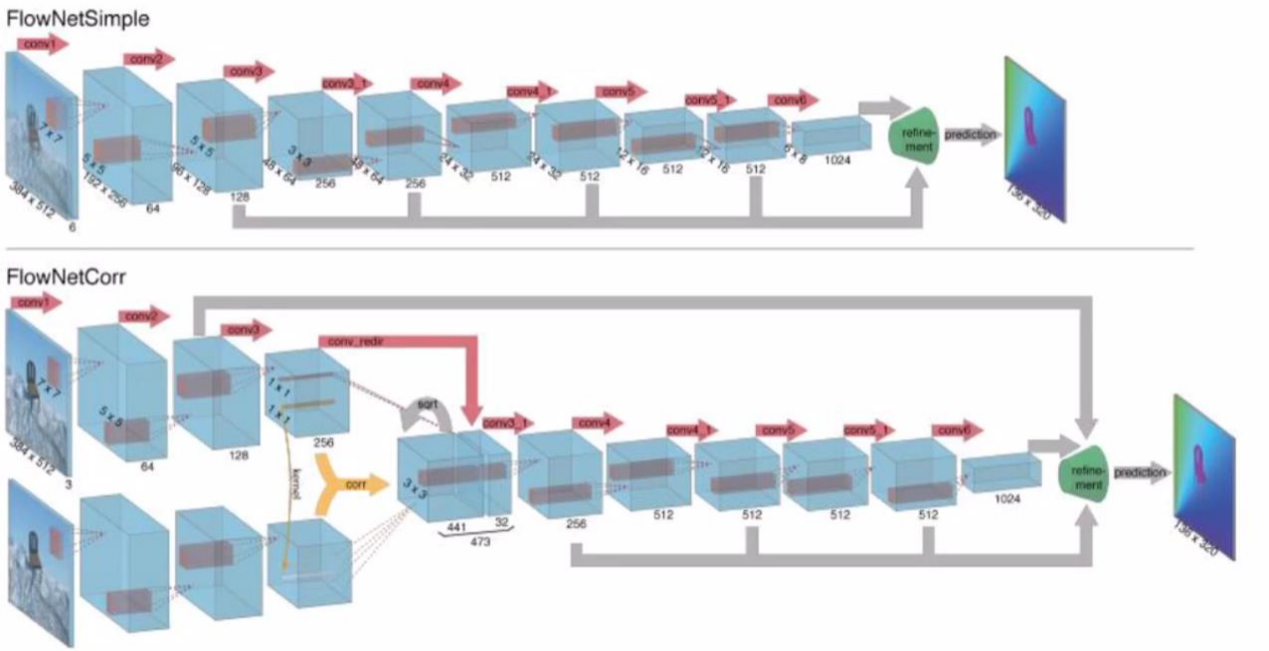
10.1.2 딥러닝 기반 광류 추정

- Optical Flow는 주로 Colormap으로 표현하는데 RGB 값이 아닌 HSV 값으로 표현
  - 여기서 HSV는 H는 색상, S가 채도, V가 명도
  - Optical Flow에선 H(색상)을 가지고 Direction(방향)을 표현하고 S(채도)를 가지고 Magnitude(크기)를 나타냄
  - 간단한 예시로 색상이 파란색이면 Direction이 왼쪽 윗 방향으로 움직인 것이고 빨간색이면 오른쪽 아래로 움직인 것.



10.1.2 딥러닝 기반 광류 추정

- FlowNetSimple은 이미지 2장을 한꺼번에 concat해서 네트워크에 넣어서 optical flow를 얻는 구조
- FlowNetCorr는 각 이미지에 대해서 feautre extraction을 진행한 다음에 두 feature를 **corrleation layer**라는 곳에 넣는다. 그리고 FlowNetSimple과 동일하게 네트워크에 넣어서 Optical Flow를 얻어내는 구조이다.



10.2 추적

■ 고전 알고리즘의 대표는 KLT 추적 알고리즘([프로그램 10-2])

- 이후 뚜렷한 개선 없음
- KLT는 지역 특징 추적하기 때문에 특징점이 뚜렷하지 않은 물체는 추적 불가능
- 추적 대상이 어떤 부류의 물체인지에 대한 정보 전혀 없음
- 추적하는 점이 같은 물체인지에 대한 정보조차 없음
- 응용 한계

■ 딥러닝은 새로운 길을 엶

- 검출과 분할을 추적으로 확장
  - 검출 예) YOLO v3은 비디오에서 물체를 안정적으로 검출하고 부류 정보를 출력함([프로그램 9-2])
  - 분할 예) mask RCNN은 겹쳐 나타나는 사람 개개인을 구별해주는 사례 분할([프로그램 9-8])

10.2.1 문제의 이해

■ 응용 상황에 따라 다양한 세부 문제로 구분

- 얼핏 프레임마다 물체를 검출하고 같은 물체를 찾아 이어주면 추적 문제가 해결된다는 단순한 생각. 하지만 그렇게 단순하지 않음
- 재식별이 필요한 응용과 필요 없는 응용
  - 재식별re-identification이란 물체가 사라졌다 다시 나타나는 경우 끊긴 추적을 이어주는 과정(카메라 시야를 벗어나거나 강한 조명으로 잠시 안보이다 다시 나타나는 상황 등)

10.2.1 문제의 이해

■ VOT와 MOT

- VOT$_{\text{Visual Object Tracking}}$는 첫 프레임에서 지정한 물체 위치를 이후 프레임에서 추적
- MOT$_{\text{Multiple Object Tracking}}$는 물체 부류를 지정하면 프레임에 나타나는 물체 여러 개 추적

(a) VOT(iceskater2 비디오)　　(b) MOT(MOT20-03 비디오의 457번째 프레임)

그림 10-7 VOT 대회와 MOT 대회가 제공하는 데이터셋

10.2.1 문제의 이해

■ 배치 방식과 온라인 방식

- 배치 방식에서는 t 순간을 처리할 때 미래의 t+1, t+2, … 프레임을 사용할 수 있음 (예, 녹화된 경기를 분석해 선수의 장단점 파악)
- 온라인 방식은 지난 순간의 프레임만 사용 가능 (예, 실시간 감시 시스템)

■ 다중 카메라 추적

- VOT와 MOT는 단일 카메라를 가정
- 다중 카메라에서는 수초 내지 수분이 지난 후 다른 카메라에 나타나는 동일 물체를 이어주는 장기 재식별이 중요
- 매장처럼 소규모와 도시 도로망처럼 대규모로 구분(AI City 대회는 도시 규모의 비디오 데이터 처리)

■ 박스 추적과 영역 추적

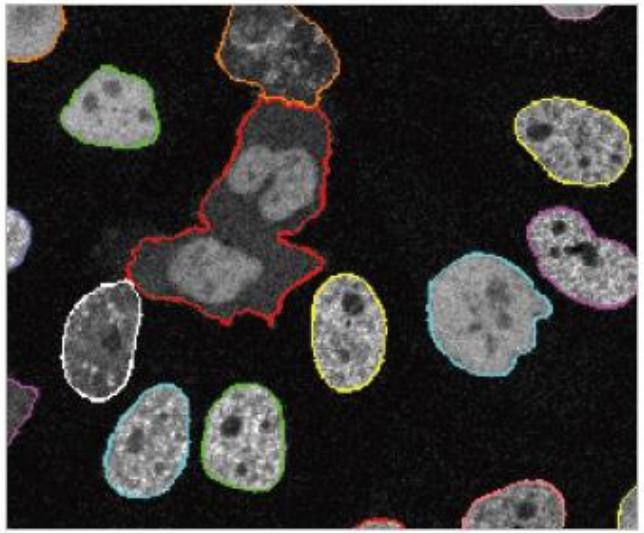
- 예) VOT는 2020년부터 박스 추적을 영역 추적으로 확장([그림 10-7(a)])

10.2.1 문제의 이해

■ 추적 물체

- 주로 사람이나 자동차를 대상으로 하는데 다른 종류 물체를 다루는 응용 많음
- 예) 돼지 추적과 세포 추적

(a) 돼지 추적(국립축산과학원 제공)



(b) 세포 추적

그림 10-8 동물과 세포 추적

10.2.2 MOT의 성능 측정

- **세부 문제에 따라 척도 다름**
- **여기서는 MOT 성능 척도를 소개**
  - 프레임 간의 연관성까지 고려해야 하므로 검출이나 분할보다 복잡. 또한 중간에 다른 물체로 교환되는 오류까지 고려해야함
  - 여러 척도 있는데 주로 MOTA$_{\text{MOT Accuracy}}$를 사용
    - GT_t와 FN_t, FP_t, $IDSW_t$는 t 순간 참값과 거짓 부정, 거짓 긍정, 번호 교환 오류 개수

$$\text{MOTA} = 1 - \frac{\sum_{t=1,T}\left(FN_t + FP_t + IDSW_t\right)}{\sum_{t=1,T} GT_t} \qquad (10.5)$$

    - IoU에 따라 매칭 여부 결정하고 그에 따라 FN, FP, $IDSW$를 셈
    - 이전 프레임의 박스를 현재 프레임의 박스에 할당하는데 헝가리안 알고리즘 사용
  - MOTA 단점을 개선한 HOTA$_{\text{Higher Order Tracking Accuracy}}$
    - MOTA는 쌍 맺기 성공률이 낮더라도 검출 성공률이 높으면 좋은 점수 부여하는 단점

10.2.2 MOT의 성능 측정

■ [예시 10-2] MOTA 계산

- 순간 1: IoU 0.5를 넘지못해 매칭 없음. FN과 FP가 하나씩 발생
- 순간 2: 매칭이 일어나 TP 하나 발생
- 순간 3: 빨강은 TP, 파랑은 FP
- 순간 4: 빨강은 TP, 파랑은 FP
- 순간 5: 빨강은 FP, 파랑은 TP. IDSW 하나 발생. 노랑은 TP
- 순간 6: TP 2개 발생
- 순간 7: TP 하나 발생

$$\mathrm{MOTA} = 1 - \frac{(1+1+0)+(0+0+0)+(0+1+0)+(0+1+0)+(0+1+1)+(0+0+0)+(0+0+0)}{(1+1+1+1+2+2+1)} = 1 - \frac{6}{9} = 0.3333$$

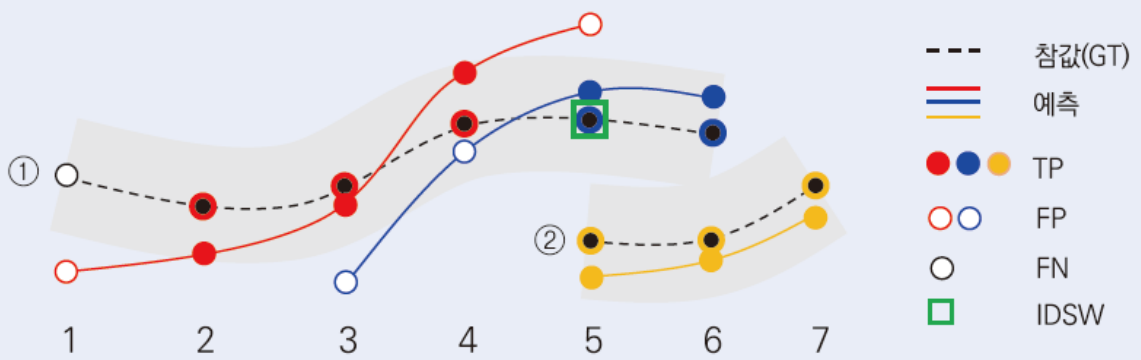


그림 10-9 MOTA 계산

10.2.3 딥러닝 기반 추적

■ 딥러닝 기반 추적 알고리즘의 전략

- 딥러닝 모델로 물체를 검출한 다음 이웃한 프레임에서 검출된 물체 집합을 매칭하여 쌍을 맺는 방식

■ 보통 4단계 처리 절차

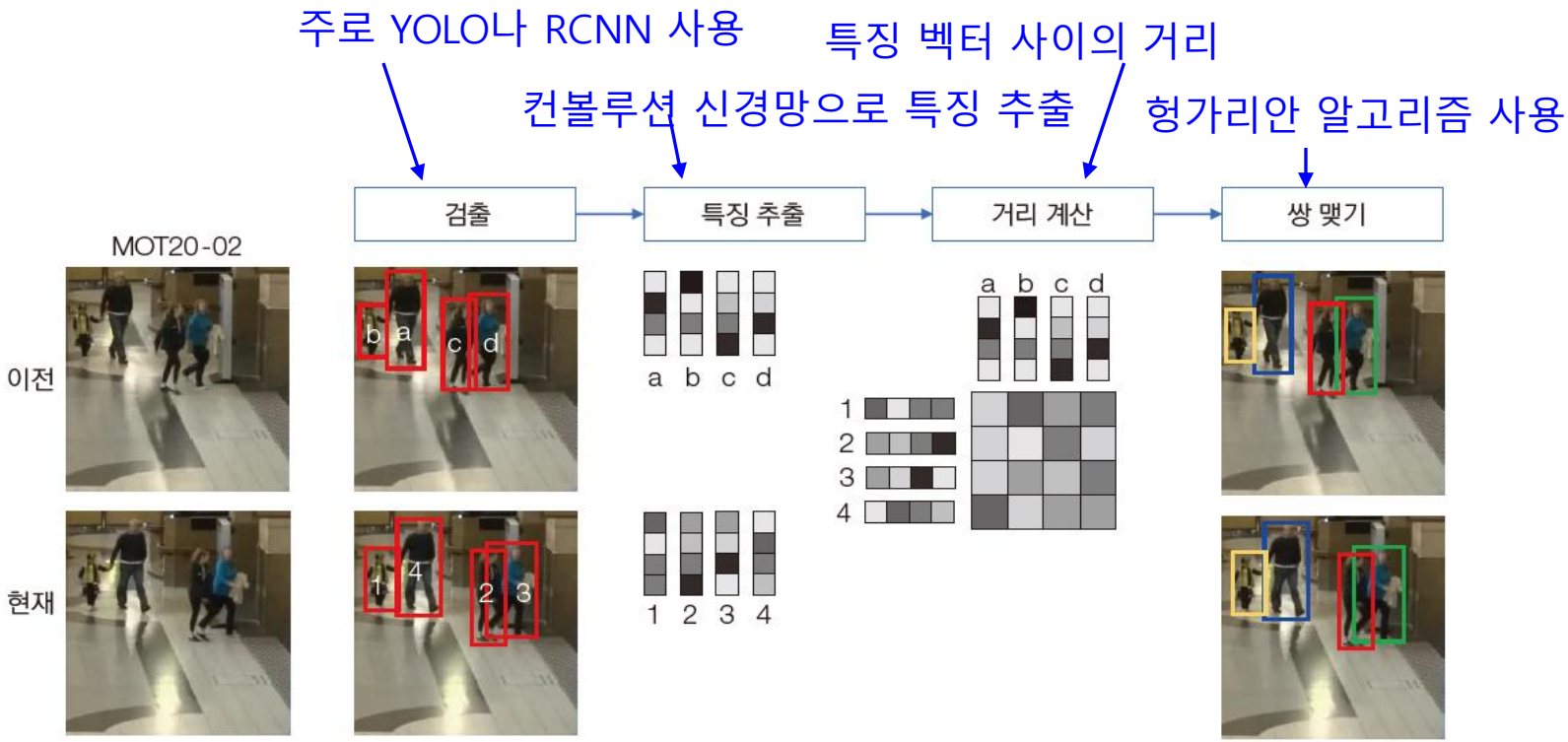


그림 10-10 물체 추적의 네 단계 처리 과정

10.2.3 딥러닝 기반 추적

■ 실제 구현에서는 다양한 변형 시도됨

- [그림 10-10]은 개념 설명할 용도
- 예) 이전 추적에서 쌓은 이력 정보 활용해 성능 향상. 물체 재식별을 추가로 다룸

■ SORT와 DeepSORT

- 단순하면서 효과적인 추적 알고리즘. 4단계 모두 단순한 알고리즘으로 처리
- 1단계: faster RCNN으로 구한 박스를 $B_{detection}$에 저장(사람 추적이라면 사람만 남기고 모두 제거)
- 2단계: t-1 순간에 결정해 놓은 위치 정보와 이동 정보(식 (10.6))를 t 순간으로 예

$$\mathbf{b} = \left(x, y, s, r, \dot{x}, \dot{y}, \dot{s}\right) \quad (10.6)$$

- 3단계: $B_{detection}$ 박스와 $B_{prediction}$ 박스의 IoU를 계산하고 1-IoU로 거리 행렬 만듦
- 4단계: 헝가리안 알고리즘으로 매칭 쌍을 찾음([예시 10-3])

10.2.3 딥러닝 기반 추적

■ [예시 10-3] 헝가리안 알고리즘으로 최적 매칭 쌍 구하기

- 헝가리안 알고리즘은 최소 비용이 되도록 작업자에게 과업을 할당하는 최적화 알고리즘
- [그림 10-11(a)]는 1~3 작업자에게 a~c 작업을 할당하는 예인데 1-c, 2-a, 3-b가 최적
- SORT에서는 $B_{detection}$을 행, $B_{prediction}$을 열에 배치. [그림 10-11(b)] 사례에서는 4열에 가상의 박스 d 배치. 1-b, 3-a, 4-c 쌍이 최적

	a	b	c
1	2	5	1
2	1	3	4
3	2	3	6

(a) 작업자에게 과업을 할당하는 문제

	a	b	c	d
1	0.9	0.2	0.7	1.0
2	0.7	0.4	0.8	1.0
3	0.1	0.3	1.0	1.0
4	1.0	0.7	0.3	1.0

(b) $B_{predict}$ 박스를 $B_{detection}$ 박스에 할당하는 문제

그림 10-11 헝가리안 알고리즘

TIP 헝가리안 알고리즘의 구체적인 동작 원리는 다음 문서를 참고한다.

https://web.archive.org/web/20120105112913/http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf

10.2.3 딥러닝 기반 추적

■ 다음 프레임으로 넘어가지 전에 후처리 수행

- 가장 중요한 일은 $B_{prediction}$에 있는 목표물의 식 (10.6)을 갱신
- 매칭된 목표물과 매칭 없는 목표물 구별해 처리
  - 매칭된 목표물은 쌍이 된 박스 정보로 x, y, s, r을 대치. 이동 이력 정보에 해당하는 $\dot{x}$, $\dot{y}$, $\dot{s}$는 칼만 필터$_{Kalman\ filter}$로 갱신
  - 매칭 실패한 목표물은 $\dot{x}$, $\dot{y}$, $\dot{s}$를 x, y, s에 더해 x, y, s를 갱신
- $B_{detection}$ 박스 중에 매칭에 실패한 것은 새로 등장한 목표물로 간주하여 식 (10.6) 정보를 생성하여 $B_{prediction}$에 추가

■ SORT를 개선한 DeepSORT

- SORT는 IDSW가 많은 편. 2단계에서 박스의 IoU만 특징으로 사용한 탓
- DeepSORT는 2단계(특징 추출)에서 IoU와 컨볼루션 신경망으로 구한 특징을 같이 사용

10.2.4 프로그래밍 실습: SORT로 사람 추적

■ [프로그램 10-3] SORT를 구현

- 물체 검출은 YOLO v3을 사용
- 추적은 SORT 논문이 제공하는 소스코드 sort.py 활용

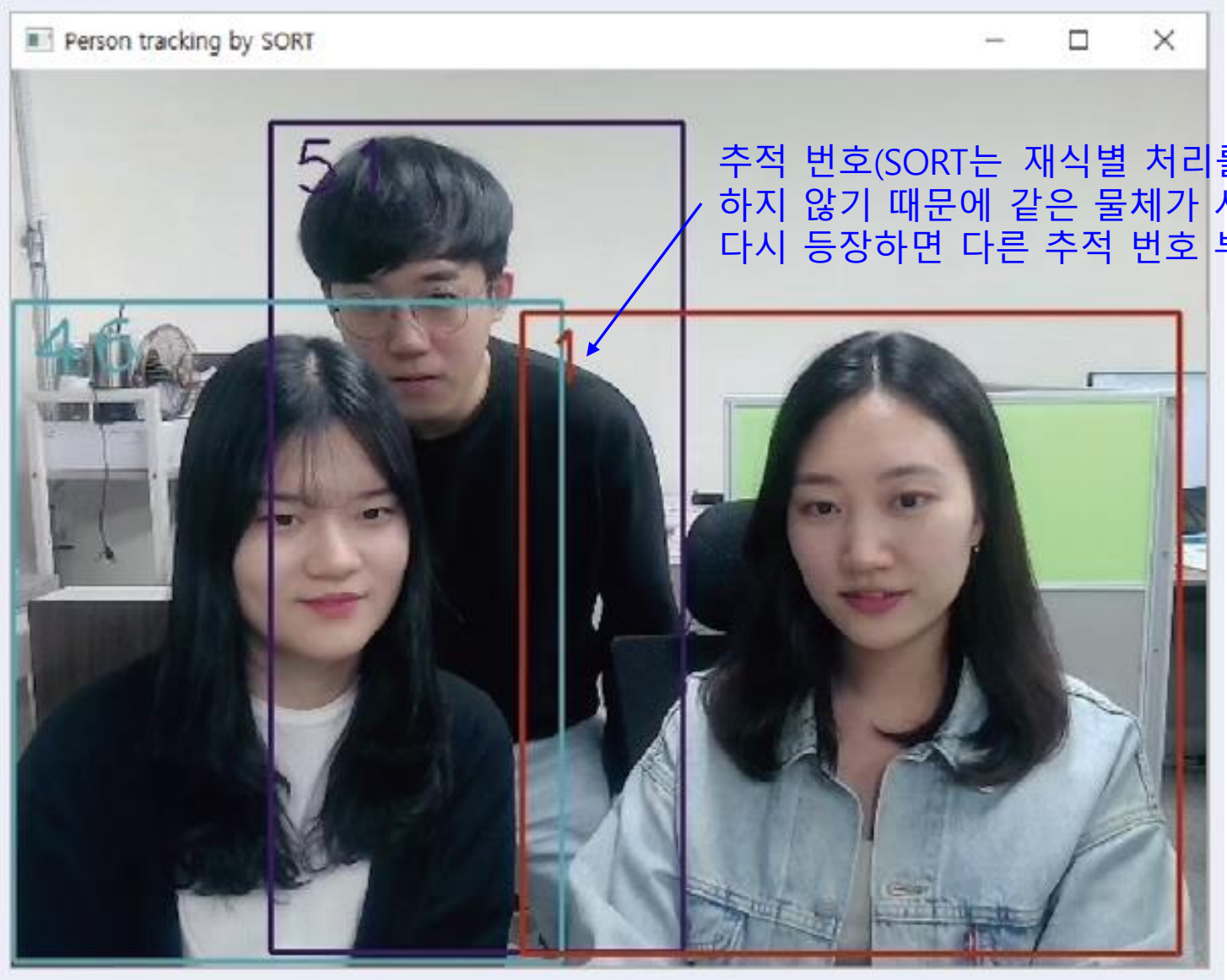
프로그램 10-3 SORT로 사람 추적하기

```
    1~38행은 [프로그램 9-1]과 같음 (YOLO v3으로 물체를 검출하는 yolo_detect 함수 등)

40  model,out_layers,class_names=construct_yolo_v3()        # YOLO 모델 생성
41  colors=np.random.uniform(0,255,size=(100,3))            # 100개 색으로 트랙 구분
42
43  from sort import Sort ←— sort.py 모듈 불러옴
44
45  sort=Sort() ←— Sort 클래스로 sort 객체 생성
46
47  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
48  if not cap.isOpened(): sys.exit('카메라 연결 실패')
49
```

10.2.4 프로그래밍 실습: SORT로 사람 추적

```
50  while True:
51      ret,frame=cap.read()
52      if not ret: sys.exit('프레임 획득에 실패하여 루프를 나갑니다.')
53
54      res=yolo_detect(frame,model,out_layers)
55      persons=[res[i] for i in range(len(res)) if res[i][5]==0] # 부류 0은 사람
56
57      if len(persons)==0:
58          tracks=sort.update()
59      else:
60          tracks=sort.update(np.array(persons))
61
62      for i in range(len(tracks)):
63          x1,y1,x2,y2,track_id=tracks[i].astype(int)
64          cv.rectangle(frame,(x1,y1),(x2,y2),colors[track_id],2)
65          cv.putText(frame,str(track_id),(x1+10,y1+40),cv.FONT_HERSHEY_
            PLAIN,3,colors[track_id],2)
66
67      cv.imshow('Person tracking by SORT',frame)
68
69      key=cv.waitKey(1)
70      if key==ord('q'): break
71
72  cap.release()                                    # 카메라와 연결을 끊음
73  cv.destroyAllWindows()
```

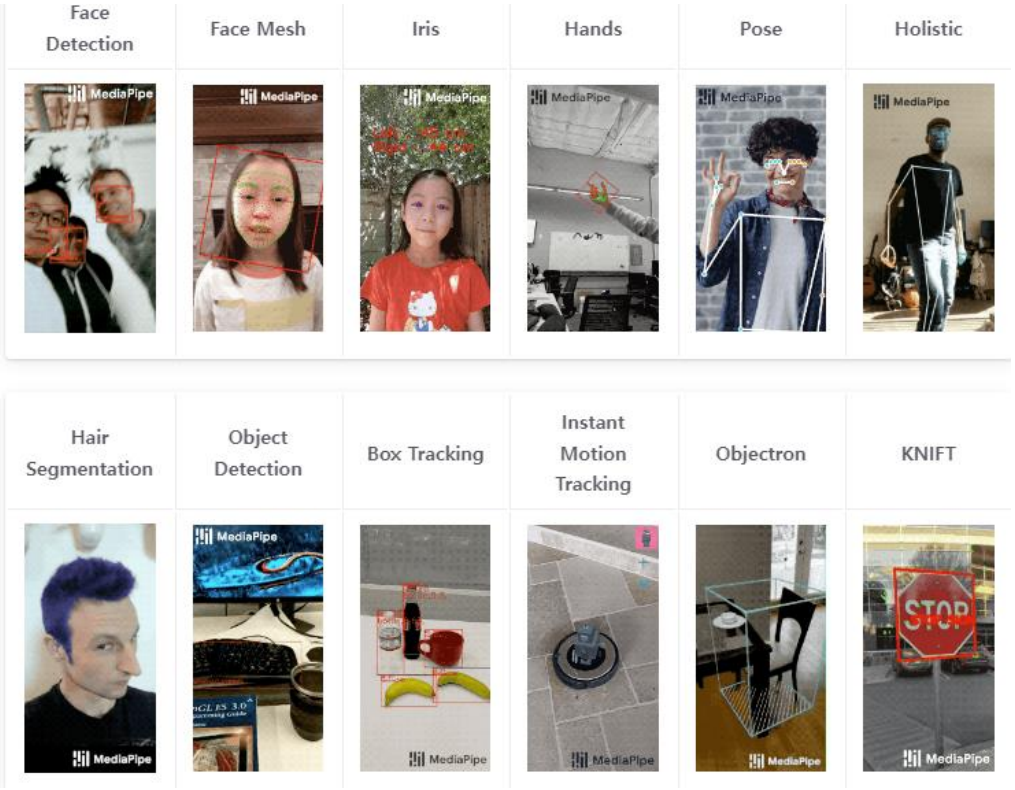
10.2.4 프로그래밍 실습: SORT로 사람 추적



10.3 MediaPipe를 이용해 비디오에서 사람 인식

■ MediaPipe

- 비디오에서 얼굴 검출, 얼굴 그물망 검출, 손 검출, 자세 추정 등을 수행하는 구글이 개발하여 공개한 솔루션
- C++로 개발하고 모바일 장치에도 실시간 실행할 수 있게 빠른 알고리즘 설계 구현
- 총 16개의 솔루션(https://google.github.io/mediapipe/)



10.3.1 얼굴 검출

■ BlazeFace 원리

- SSD(한 단계 물체 검출 모델)을 얼굴 검출에 적합하게 개조하여 사용
  - 입력 영상을 16*16, 8*8까지 줄인 맵을 만드는데, 매 화소에 컨볼루션을 적용하여 2~6개의 박스를 예측함
  - 비최대 억제(신뢰도가 최대인 박스 선택) 대신, 여러 박스의 가중 평균을 취함
  - 얼굴 박스뿐 아니라 6개의 랜드마크(눈 중심, 귀 구슬점, 입 중심, 코끝)까지 추출
  - 200~1000FPS 보장

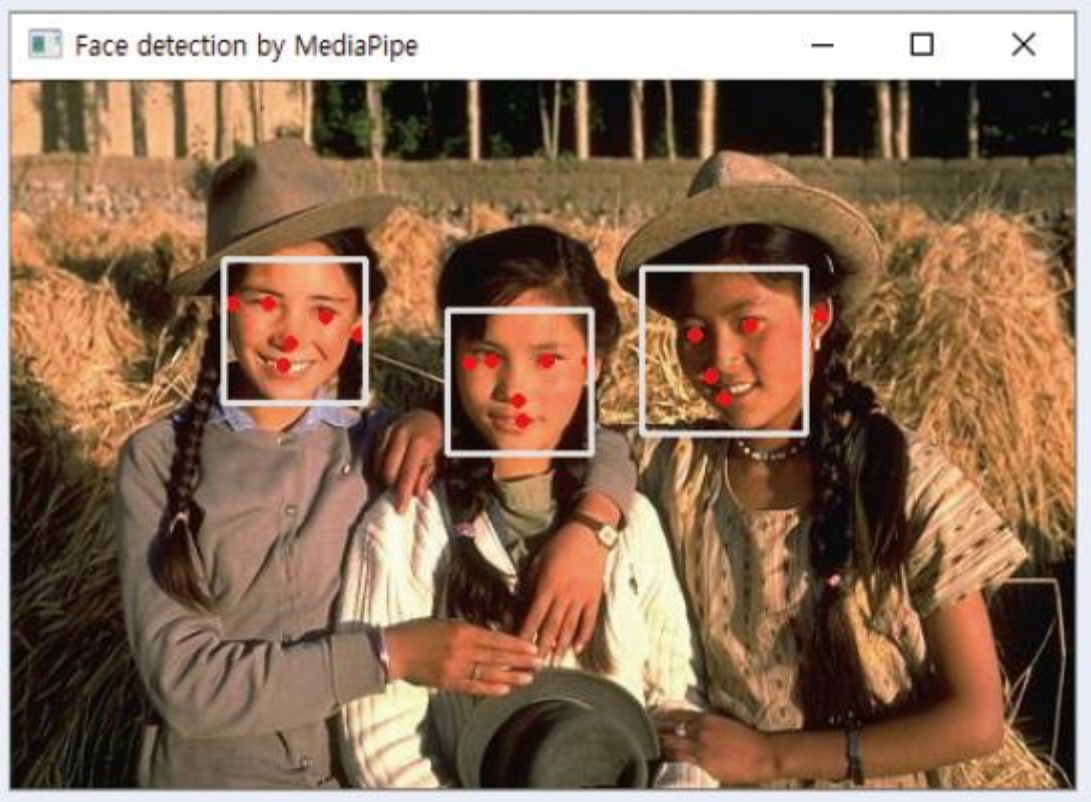
10.3.1 얼굴 검출

■ 프로그래밍 실습: 정지 영상에서 얼굴 검출

프로그램 10-4 BlazeFace로 얼굴 검출하기

```
01  import cv2 as cv
02  import mediapipe as mp
03
04  img=cv.imread('BSDS_376001.jpg')
05
06  mp_face_detection=mp.solutions.face_detection
07  mp_drawing=mp.solutions.drawing_utils
08
09  face_detection=mp_face_detection.FaceDetection(model_selection=1,min_
    detection_confidence=0.5)
10  res=face_detection.process(cv.cvtColor(img,cv.COLOR_BGR2RGB))
11
12  if not res.detections:
13      print('얼굴 검출에 실패했습니다. 다시 시도하세요.')
14  else:
15      for detection in res.detections:
16          mp_drawing.draw_detection(img,detection)
17      cv.imshow('Face detection by MediaPipe',img)
18
19  cv.waitKey()
20  cv.destroyAllWindows()
```

10.3.1 얼굴 검출



10.3.1 얼굴 검출

검출 결과를 담은 리스트 내용을 확인

```
In [1]: print(res.detections)
   [label_id: 0
     score: 0.8932861089706421
     location_data {
       format: RELATIVE_BOUNDING_BOX
       relative_bounding_box { xmin: 0.1996, ymin: 0.2552, width: 0.1337, height: 0.2004}
       relative_keypoints { x: 0.2414, y: 0.3165}
       relative_keypoints { x: 0.2967, y: 0.3340}
       … …
   label_id: 0
     score: 0.8440857529640198
       … …
   label_id: 0
     score: 0.813656747341156
       … …]
```

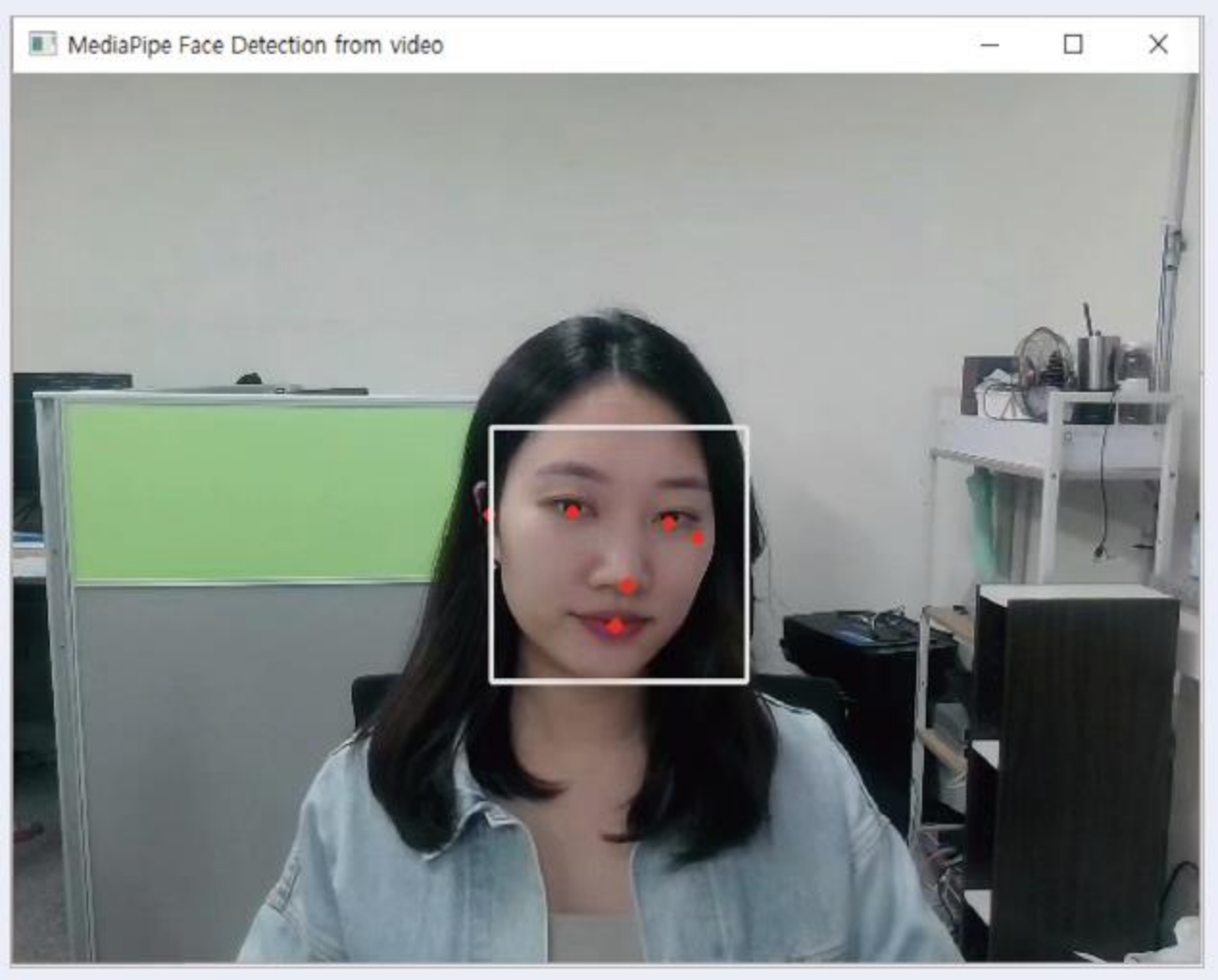
10.3.1 얼굴 검출

■ 프로그래밍 실습: 비디오에서 얼굴 검출

프로그램 10-5 비디오에서 얼굴 검출하기

```
01  import cv2 as cv
02  import mediapipe as mp
03
04  mp_face_detection=mp.solutions.face_detection
05  mp_drawing=mp.solutions.drawing_utils
06
07  face_detection=mp_face_detection.FaceDetection(model_selection=1,min_
    detection_confidence=0.5)
08
09  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
10
11  while True:
12      ret,frame=cap.read()
13      if not ret:
14          print('프레임 획득에 실패하여 루프를 나갑니다.')
15          break
16
17      res=face_detection.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
18
19      if res.detections:
20          for detection in res.detections:
21              mp_drawing.draw_detection(frame,detection)
22
23      cv.imshow('MediaPipe Face Detection from video',cv.flip(frame,1))
24      if cv.waitKey(5)==ord('q'):
25          break
26
27  cap.release()
28  cv.destroyAllWindows()
```

10.3.1 얼굴 검출



10.3.1 얼굴 검출

■ 프로그래밍 실습: 얼굴을 장식하는 증강 현실

프로그램 10-6 얼굴을 장식하는 증강 현실 구현하기

```
01  import cv2 as cv
02  import mediapipe as mp
03
04  dice=cv.imread('dice.png',cv.IMREAD_UNCHANGED) # 증강 현실에 쓸 장신구
05  dice=cv.resize(dice,dsize=(0,0),fx=0.1,fy=0.1)
06  w,h=dice.shape[1],dice.shape[0]
07
08  mp_face_detection=mp.solutions.face_detection
09  mp_drawing=mp.solutions.drawing_utils
10
11  face_detection=mp_face_detection.FaceDetection(model_selection=1,min_
    detection_confidence=0.5)
12
13  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
14
```

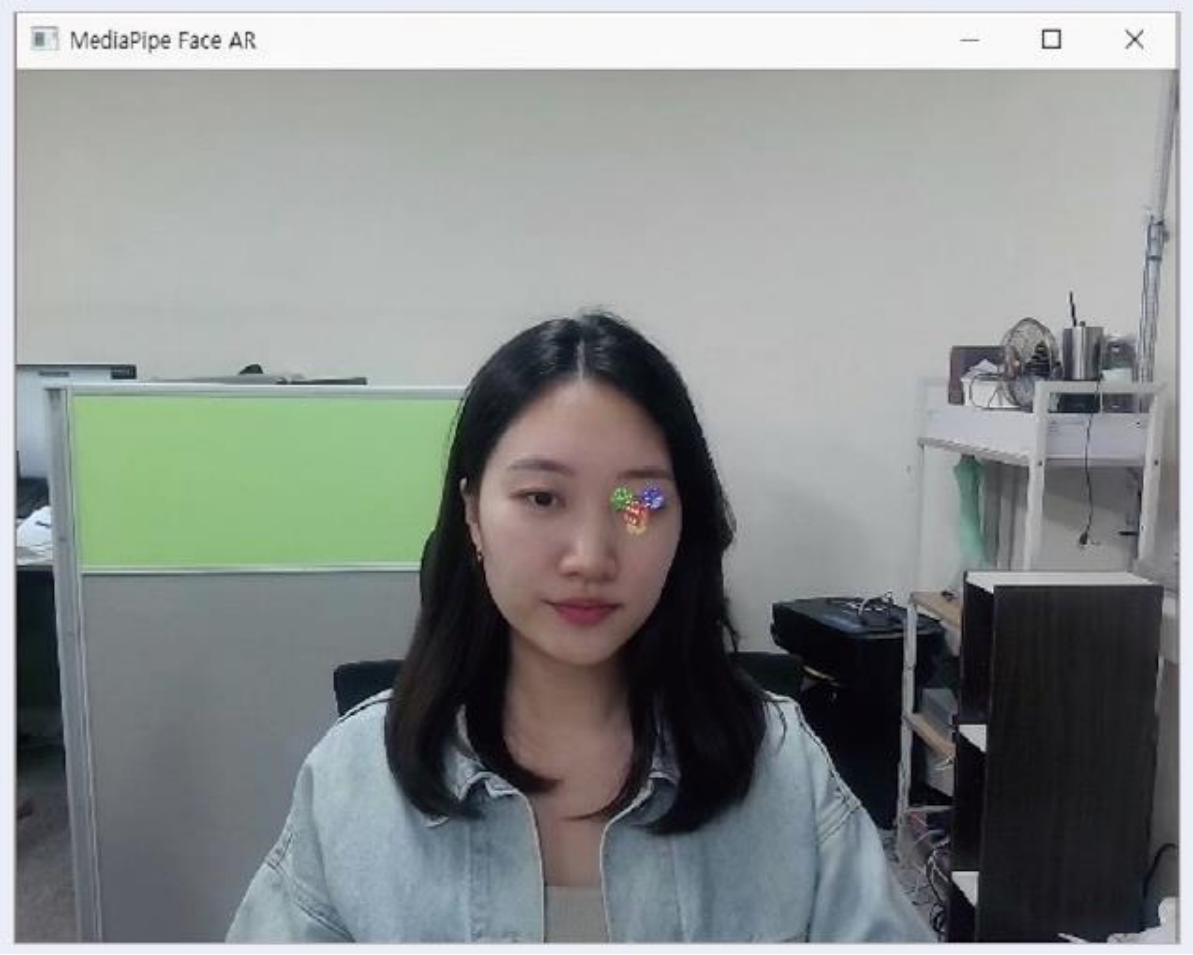
알파 채널로 투명도 조절이 가능한 png 파일

10.3.1 얼굴 검출

```
15  while True:
16      ret,frame=cap.read()
17      if not ret:
18          print('프레임 획득에 실패하여 루프를 나갑니다.')
19          break
20
21      res=face_detection.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
22
23      if res.detections:
24          for det in res.detections:
25              p=mp_face_detection.get_key_point(det,mp_face_detection.
                FaceKeyPoint.RIGHT_EYE)
26              x1,x2=int(p.x*frame.shape[1]-w//2),int(p.x*frame.shape[1]+w//2)
27              y1,y2=int(p.y*frame.shape[0]-h//2),int(p.y*frame.shape[0]+h//2)
28              if x1>0 and y1>0 and x2<frame.shape[1] and y2<frame.shape[0]:
29                  alpha=dice[:,:,3:]/255 # 투명도를 나타내는 알파값
30                  frame[y1:y2,x1:x2]=frame[y1:y2,x1:x2]*(1-alpha)+dice[:,:,:3]*alpha
31
32      cv.imshow('MediaPipe Face AR›,cv.flip(frame,1))
33      if cv.waitKey(5)==ord('q'):
34          break
35
36  cap.release()
37  cv.destroyAllWindows()
```

MediaPipe는 속도가 빨라
실시간 증강 현실 구현에 적합

10.3.1 얼굴 검출




```
In [7]: dir(mp_face_detection.FaceKeyPoint)   랜드마크 이름을 알려면
   ['LEFT_EAR_TRAGION', 'LEFT_EYE', 'MOUTH_CENTER', 'NOSE_TIP', 'RIGHT_EAR_TRAGION',
   'RIGHT_EYE', ...]
```

10.3.2 얼굴 그물망 검출

■ FaceMesh 원리

- 468개의 랜드마크 검출

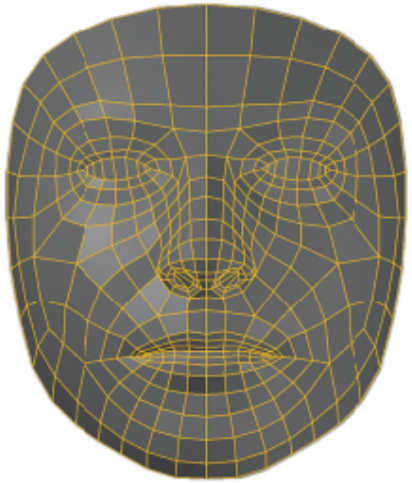


그림 10-12 FaceMesh가 사용하는 468개의 얼굴 랜드마크[Kartynnik2019]

- 첫 프레임에서 BlazeFace로 얼굴 검출. 이후에는 추적만으로 처리 시간을 획기적 단축
- 신뢰도가 임곗값보다 낮으면 BlazeFace를 다시 적용하여 새로 얼굴 검출

10.3.2 얼굴 그물망 검출

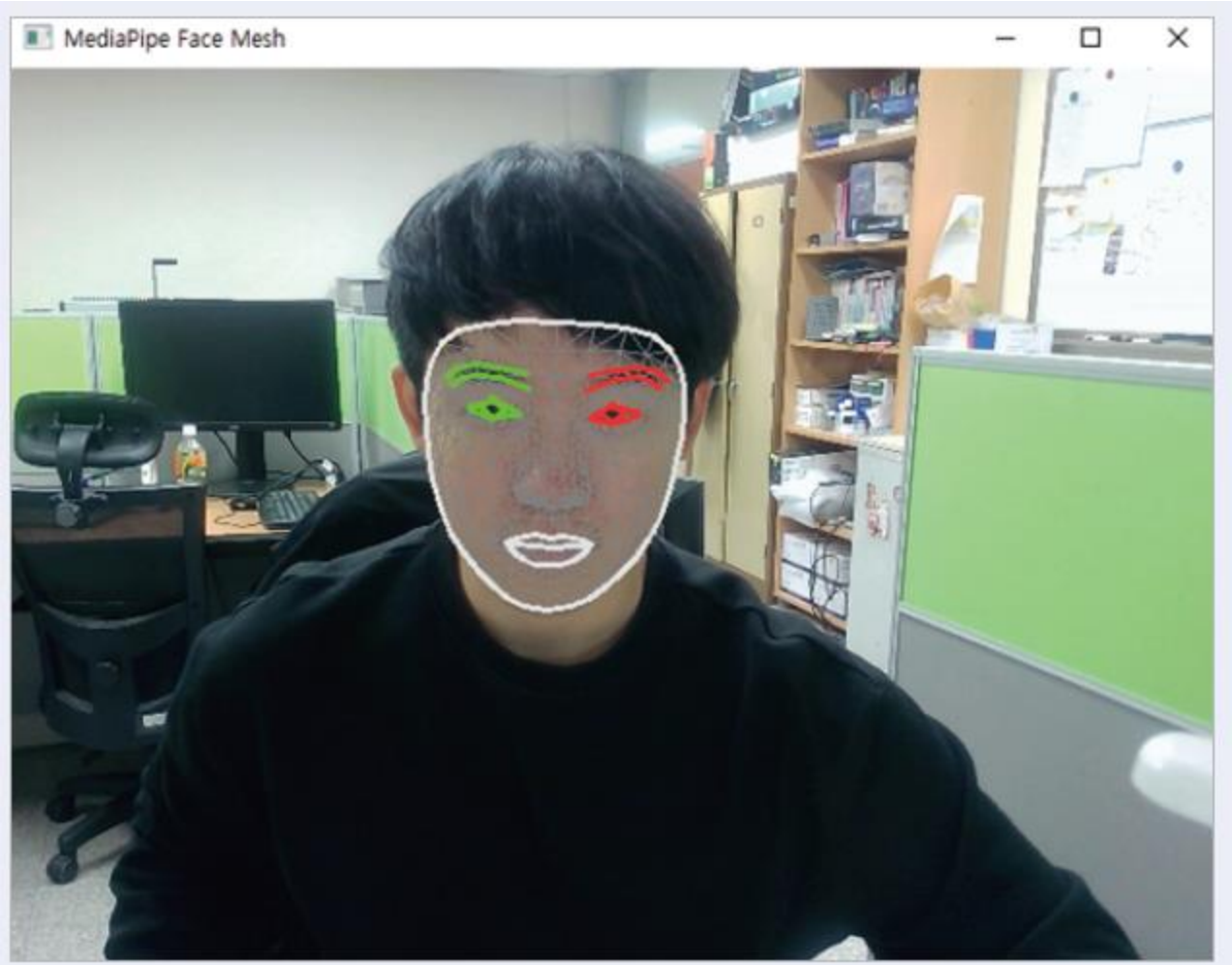
프로그램 10-7 FaceMesh로 얼굴 그물망 검출하기

```
01  import cv2 as cv
02  import mediapipe as mp
03
04  mp_mesh=mp.solutions.face_mesh
05  mp_drawing=mp.solutions.drawing_utils
06  mp_styles=mp.solutions.drawing_styles
07
08  mesh=mp_mesh.FaceMesh(max_num_faces=2,refine_landmarks=True,min_detection_
    confidence=0.5,min_tracking_confidence=0.5)
09
10  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
11
```

10.3.2 얼굴 그물망 검출

```
12  while True:
13      ret,frame=cap.read()
14      if not ret:
15        print('프레임 획득에 실패하여 루프를 나갑니다.')
16        break
17
18      res=mesh.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
19
20      if res.multi_face_landmarks:
21          for landmarks in res.multi_face_landmarks:
22              mp_drawing.draw_landmarks(image=frame,landmark_
                list=landmarks,connections=mp_mesh.FACEMESH_TESSELATION,landmark_
                drawing_spec=None,connection_drawing_spec=mp_styles.get_default_
                face_mesh_tesselation_style())
23              mp_drawing.draw_landmarks(image=frame,landmark_
                list=landmarks,connections=mp_mesh.FACEMESH_CONTOURS,landmark_
                drawing_spec=None,connection_drawing_spec=mp_styles.get_default_
                face_mesh_contours_style())
24              mp_drawing.draw_landmarks(image=frame,landmark_
                list=landmarks,connections=mp_mesh.FACEMESH_IRISES,landmark_
                drawing_spec=None,connection_drawing_spec=mp_styles.get_default_
                face_mesh_iris_connections_style())
25
26      cv.imshow('MediaPipe Face Mesh›,cv.flip(frame,1))      # 좌우 반전
27      if cv.waitKey(5)==ord('q'):
28        break
29
30  cap.release()
31  cv.destroyAllWindows()
```

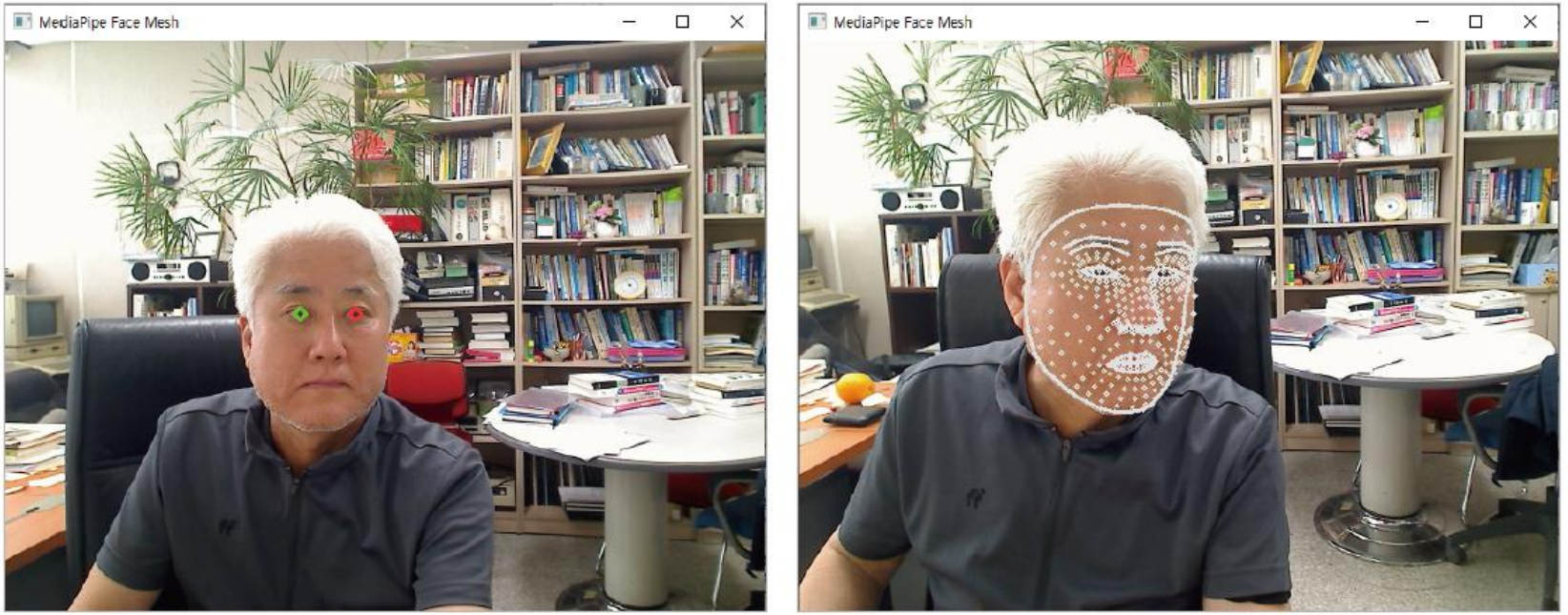
10.3.2 얼굴 그물망 검출



10.3.2 얼굴 그물망 검출

```
mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.
FACEMESH_CONTOURS,landmark_drawing_spec=mp_drawing.DrawingSpec(thickness=1,circle_
radius=1))
```

다양한 방식으로 그리기 가능



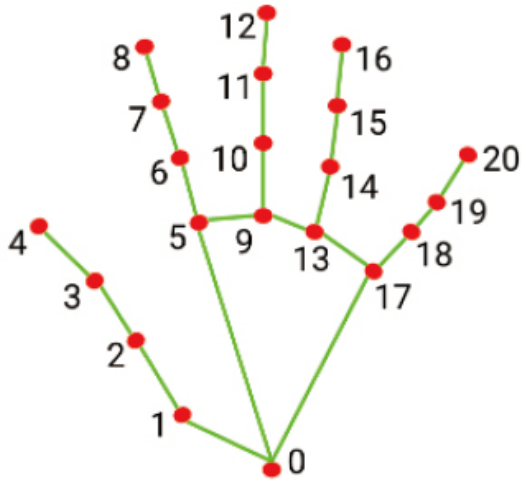
(a) 22~24행에서 24행만 남기고 실행한 결과 (b) 23행만 남긴 뒤 수정 후 실행한 결과

그림 10-13 다양한 얼굴 그물망을 적용하기 위해 [프로그램 10-7]의 22~24행을 변형한 경우

10.3.3 손 랜드마크 검출

■ BlazeHand 원리

- 손을 검출하는 BlazePalm 모듈과 랜드마크를 검출하고 추적하는 모듈로 구성
- BlazePalm은 SSD를 개조하여 사용. 계산 시간 절약을 위해 첫 프레임에서 BlazePalm을 적용. 이후에는 이전 프레임의 랜드마크를 현재 프레임에서 예측하는 방식 사용



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

그림 10-14 손을 위한 21개의 랜드마크[Zhang2020]

10.3.3 손 랜드마크 검출

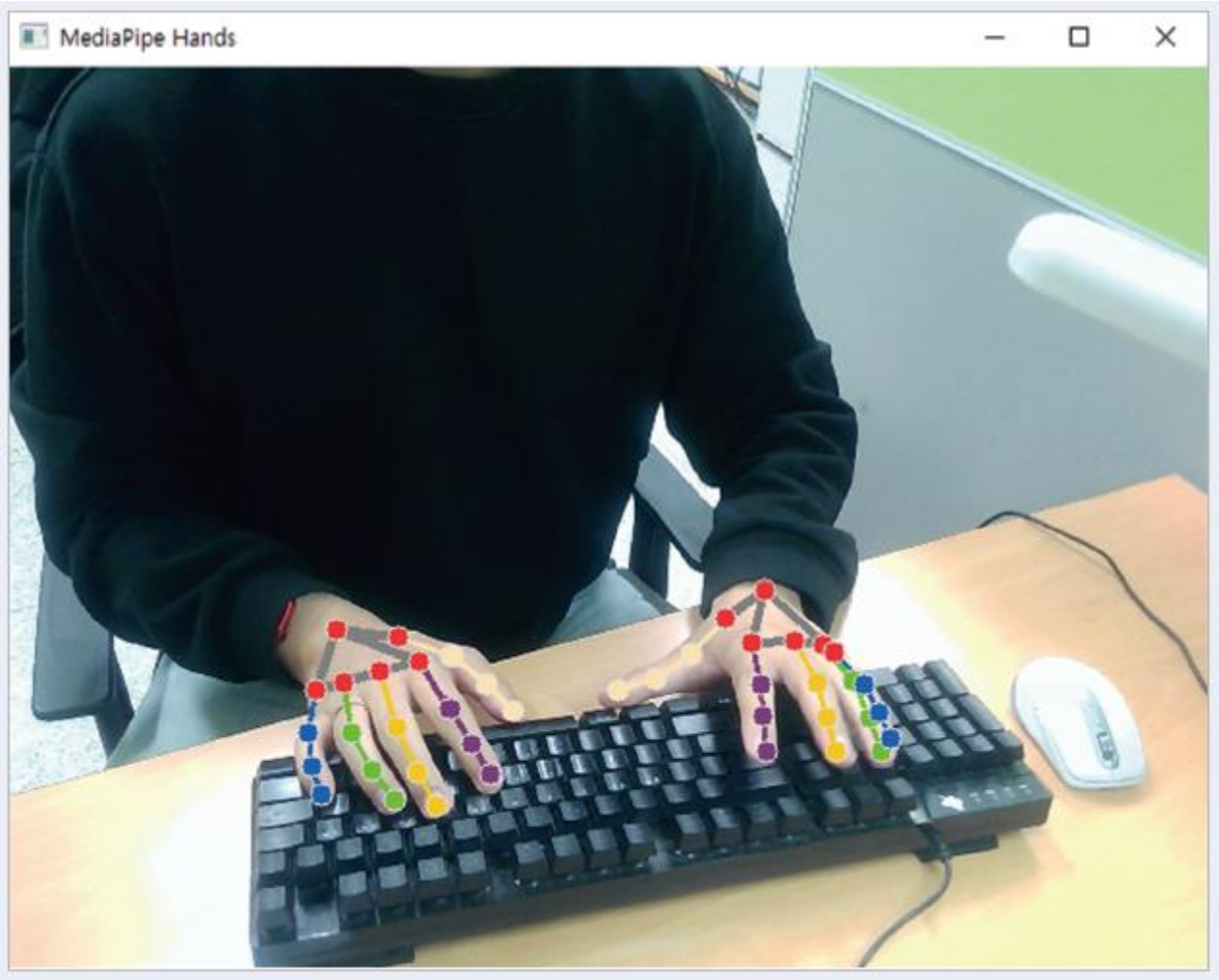
프로그램 10-8 손 랜드마크 검출하기

```
01  import cv2 as cv
02  import mediapipe as mp
03
04  mp_hand=mp.solutions.hands
05  mp_drawing=mp.solutions.drawing_utils
06  mp_styles=mp.solutions.drawing_styles
07
08  hand=mp_hand.Hands(max_num_hands=2,static_image_mode=False,min_detection_
    confidence=0.5,min_tracking_confidence=0.5)
09
10  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
11
```

10.3.3 손 랜드마크 검출

```
12  while True:
13      ret,frame=cap.read()
14      if not ret:
15        print('프레임 획득에 실패하여 루프를 나갑니다.')
16        break
17
18      res=hand.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
19
20      if res.multi_hand_landmarks:
21          for landmarks in res.multi_hand_landmarks:
22              mp_drawing.draw_landmarks(frame,landmarks,mp_hand.HAND_
                CONNECTIONS,mp_styles.get_default_hand_landmarks_style(),mp_styles.
                get_default_hand_connections_style())
23
24      cv.imshow('MediaPipe Hands›,cv.flip(frame,1))    # 좌우반전
25      if cv.waitKey(5)==ord('q'):
26        break
27
28  cap.release()
29  cv.destroyAllWindows()
```

10.3.3 손 랜드마크 검출



10.4 자세 추정과 행동 분류

■ 전신을 분석하는 기능

- 많은 응용
  - 예) 보행자에 대한 자세 추정과 행동 분류는 자율주행차에 필수
  - 예) CCTV에 나타난 사람을 분석하여 범죄 예방에 활용
- 보통 자세 추정한 결과를 가지고 행동 분류. 10.4.1항은 자세 추정, 10.4.3항은 행동 분류를 소개

10.4.1 자세 추정

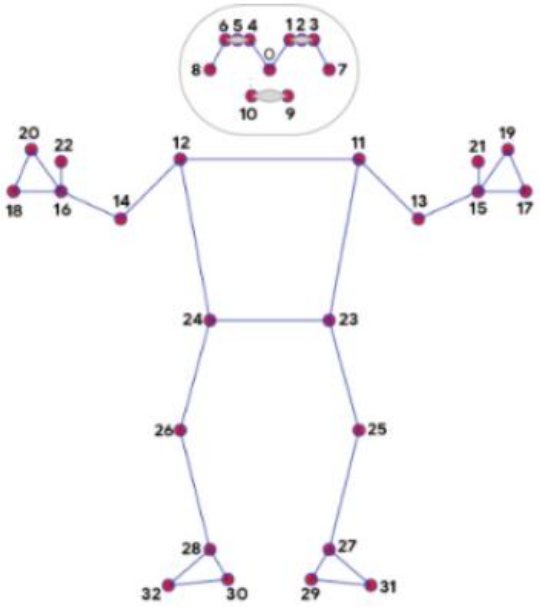
■ 자세 추정

- 정지 영상 또는 비디오를 분석해 전신에 있는 관절 위치를 알아내는 일
- 관절을 랜드마크 또는 키포인트라 부름
- 아주 많은 응용
  - 행동 분류와 행동 예측
  - 선수의 자세 교정
  - 환자의 재활 도우미 또는 낙상 방지
  - 증강 현실이나 가상 현실
  - 애니메이션 또는 게임 제작 등

10.4.1 자세 추정

■ 인체 모델

- 골격 표현법과 부피 표현법
- 이 책은 골격 표현법에 국한



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_inde›

(a) BlazePose의 골격 표현[Bazarevsky2020]

(b) SMPL 부피 표현[Loper2015]

그림 10-15 인체 모델

10.4.1 자세 추정

■ 정지 영상에서 자세 추정

- 고전 컴퓨터 비전
  - 주로 HOG 특징 사용([그림 9-18])
  - 부품 모델로 인체 표현([그림 9-19])
  - 깨끗한 배경을 가정한 상황에서나 동작하는 초보적 수준
- 딥러닝으로 전환하여 획기적 발전

10.4.1 자세 추정

■ 좌표 회귀와 열지도 회귀

- DeepPose는 자세 추정에 딥러닝을 처음 적용한 모델
  - 220*220*3 영상을 받아 5개 컨볼루션층과 2개 완전연결층을 거쳐 2k개의 실수를 출력
  - k는 랜드마크 개수로서 (x,y) 좌표를 회귀하는 모델
- 좌표를 직접 회귀하는 대신 열지도 회귀$_{\text{heatmap regression}}$하는 방법
  - 랜드마크 (x,y) 위치에 가우시안을 씌운 맵을 예측. 따라서 k개의 2차원 맵을 출력
  - SHG$_{\text{Stacked HourGlass}}$는 열지도 회귀를 채택한 대표적 모델

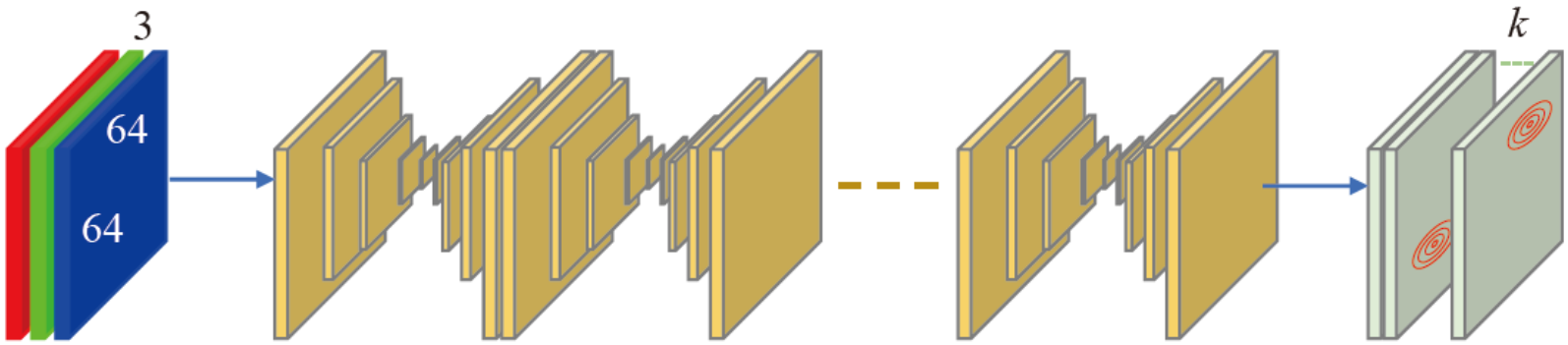


그림 10-16 열지도 회귀 기법을 사용하는 SHG 모델

10.4.1 자세 추정

■ 여러 사람의 자세 추정

- 하향식$_{\text{top-down}}$ 모델
  - faster RCNN과 같은 모델로 사람 부분을 잘라내 SHG같은 모델을 적용하여 해결
  - AlphaPose와 CrowdPose 모델이 대표적
- 상향식$_{\text{bottom-up}}$ 모델
  - 랜드마크를 모두 검출한 다음 랜드마크를 결합하여 사람별로 자세 추정
  - OpenPose 모델이 대표적

(a) AlphaPose[Fang2017]의 실험 결과

(b) OpenPose[Cao2017]의 실험 결과

그림 10-17 다수 사람의 자세 추정

10.4.1 자세 추정

■ 비디오에서 자세 추정

- 가장 단순한 방법은 프레임별로 독립적으로 자세 추정 적용
  - 추정한 자세가 오류로 인해 프레임 간에 심하게 흔들리는 현상
- 동작은 매끄럽게 변한다는 사실을 잘 이용하여 일관성 있는 자세 추정 가능
- 가림이나 흐릿한 경우 이웃 프레임 정보를 이용하여 안 보이는 랜드마크 추정 가능

10.4.1 자세 추정

■ 이웃 프레임을 고려하는 접근 방법

- 광류를 사용하는 방법
  - Jain은 RGB 영상과 광류 맵을 결합한 텐서를 컨볼루션 신경망에 입력함
  - Pfister는 여러 프레임에서 랜드마크 열지도를 예측한 다음 광류로 현재 프레임에 맞게 변환하고 변환된 열지도를 결합하여 성능 향상
  - Song은 광류로 t-1, t, t+1 순간의 랜드마크를 추출한 다음에 이들로부터 t 순간의 자세를 정제
- 순환 신경망을 사용하는 방법
  - 시계열 데이터를 모델링하는 LSTM 사용
  - Luo는 t 순간 프레임에서 추출한 특징 맵을 t-1 순간의 정보와 결합하여 LSTM층에 입력. 이 특징 맵은 LSTM층과 컨볼루션층을 통과하여 t 순간의 자세를 예측

10.4.1 자세 추정

■ 자세 추적pose tracking

- 하나의 프레임에서 여러 명을 구분하고 개개인의 자세를 추정한 다음 이후 프레임에서 자세 단위로 사람을 추적
- 박스 단위로 사람을 표시했던 [그림 10-7(b)]의 MOT 문제가 골격 단위로 표시한 사람을 추적하는 문제로 확장

그림 10-18 자세 추적을 위한 PoseTrack 데이터셋의 예시 비디오(세 프레임 건너 표시)

10.4.1 자세 추정

■ 데이터셋

- FLIC: 영화에서 5,003장의 정지 영상을 추출하고 상반신에 10개 랜드마크를 레이블링
- LSP: 운동 장면을 찍은 정지 영상 2,000장에 대해 14개 랜드마크를 레이블링
- MPII: 유튜브 비디오에서 수집한 정지 영상을 16개 랜드마크로 레이블링. 25,000장 가량의 영상에서 4만명 넘는 사람을 레이블링
- COCO: 20만장 이상의 영상에서 25만명 이상을 17개 랜드마크로 레이블링
- CrowdPose: 군중 영상을 대상으로 레이블링
- PoseTrack: 비디오에서 자세 추적([그림 10-18])

10.4.1 자세 추정

■ 성능 척도

- 일부 랜드마크만 보이거나 다른 사람과 엉킨 경우 많아 성능 측정은 꽤 까다로움
- 초창기에는 주로 PCP$_{\text{Percentage of Correct Parts}}$ 사용
  - 관절과 관절을 잇는 구성품을 제대로 찾은 비율
  - [그림 10-19]에서 예측한 랜드마크가 점선 원 안에 있으면 제대로 찾았다고 판정
    - 반지름은 구성품 길이의 0.5이므로 PCP@0.5라고 표기
    - 원의 반지름을 바꾸면서 PCP@0.1, PCP@0.2, ...를 측정하여 그래프를 그리면 PCP 곡선

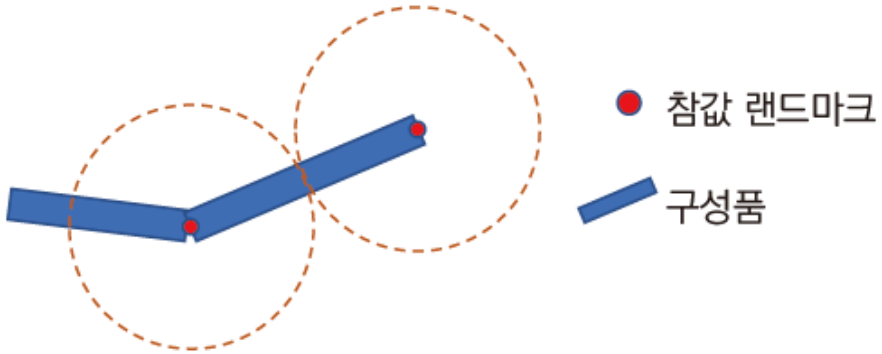


그림 10-19 자세 추정의 성능 척도로 사용되는 PCP@0.5

10.4.1 자세 추정

- 구성품이 영상에 짧게 나타나는 경우 PCP는 너무 큰 벌점을 주는 경향이 있어 현재는 잘 사용하지 않음
- 현재는 랜드마크별로 제대로 찾았는지 측정하는 척도를 주로 사용
  - PCKh@0.5: 랜드마크가 머리 구성품 길이의 0.5 이내에 있으면 맞다고 판정
  - PCK@0.2: 몸통 직경의 0.2 이내에 있으면 맞다고 판정
- 검출이 사용하는 AP와 mAP를 척도로 사용
  - 이때 IoU 대신 $OKS_{Object\ \ Keypoint\ \ Similarity}$를 사용

TIP OKS에 대한 자세한 내용은 COCO 성능 평가 사이트(https://cocodataset.org/#keypoints-eval)를 참조한다.

10.4.2 BlazePose를 이용한 자세 추정

■ BlazePose

- 33개 랜드마크로 자세 표현([그림 10-15(a)])
  - COCO의 17개보다 많아 응용 범위 넓음
- 깊이 정보를 얼추 추정하여 랜드마크를 3차원 좌표로 표현
- 좌표 회귀와 열지도 회귀를 둘 다 사용하여 성능 향상
  - 학습할 때는 열지도 출력에 대해 손실 함수를 계산하여 학습 성능을 높임
  - 추론(예측) 과정에서는 열지도 출력 부분을 떼내고 33*3 텐서 출력을 취함(33개 랜드마크에 대해 (x,y) 좌표와 보이는지 여부 표시)
- 몸 전체 ROI를 직접 찾는 일이 어려워 BlazeFace로 얼굴을 찾은 다음 어깨 중심 두 개와 엉덩이 중심 두 개를 잇는 선을 기준으로 전신 ROI 검출
- 빠른 계산을 위해 ROI 검출은 첫 프레임에만 적용하고 이후에는 랜드마크로 예측

10.4.2 BlazePose를 이용한 자세 추정

프로그램 10-9 BlazePose를 이용한 자세 추정하기

```
01  import cv2 as cv
02  import mediapipe as mp
03
04  mp_pose=mp.solutions.pose
05  mp_drawing=mp.solutions.drawing_utils
06  mp_styles=mp.solutions.drawing_styles
07
08  pose=mp_pose.Pose(static_image_mode=False,enable_segmentation=True,min_
    detection_confidence=0.5,min_tracking_confidence=0.5)
09
10  cap=cv.VideoCapture(0,cv.CAP_DSHOW)
11
```

비디오로 간주하라는 지시

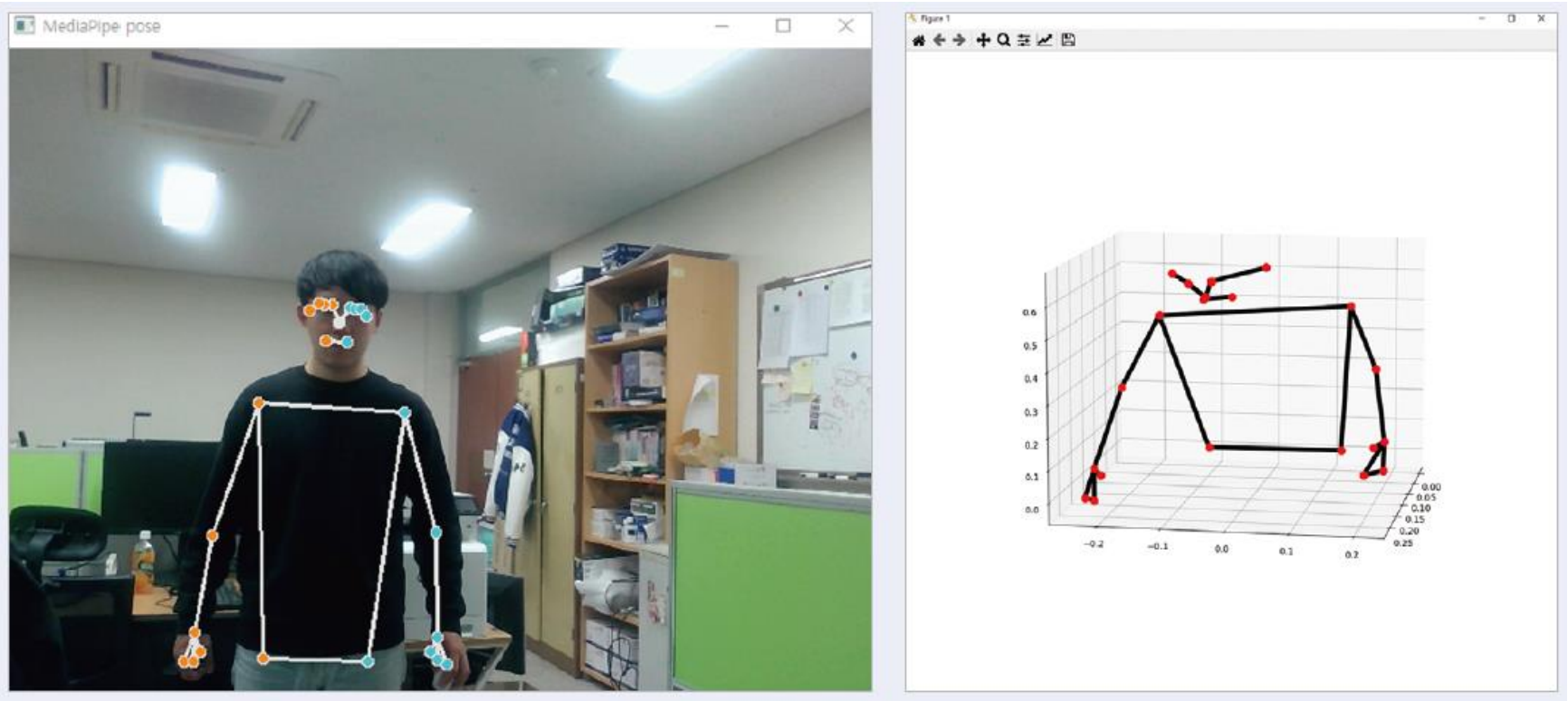
전경과 배경을 분할하라는 지시

10.4.2 BlazePose를 이용한 자세 추정

```
12  while True:
13      ret,frame=cap.read()
14      if not ret:
15        print('프레임 획득에 실패하여 루프를 나갑니다.')
16        break
17
18      res=pose.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
19
20      mp_drawing.draw_landmarks(frame,res.pose_landmarks,mp_pose.POSE_
        CONNECTIONS,landmark_drawing_spec=mp_styles.get_default_pose_landmarks_
        style())
21
22      cv.imshow('MediaPipe pose›,cv.flip(frame,1)) # 좌우 반전
23      if cv.waitKey(5)==ord('q'):
24        mp_drawing.plot_landmarks(res.pose_world_landmarks,mp_pose.POSE_
          CONNECTIONS)
25        break
26
27  cap.release()
28  cv.destroyAllWindows()
```

(x,y,z) 좌표를 가짐

10.4.2 BlazePose를 이용한 자세 추정



10.4.3 행동 분류

■ 사람의 인식 능력

- 다른 사람의 행동을 정확히 인식하고 이전에 취득한 지식과 현재 상황 결합하여 상대 의도를 능숙하게 추론
- 행동 분류를 넘어 행동 이해action understanding까지 능숙

■ 컴퓨터 비전의 한계

- 행동 이해로 나아가지 못함
- 정지 영상 또는 비디오를 분석해 미리 정해진 몇 가지 행동으로 분류하는 정도

10.4.3 행동 분류

■ 데이터셋

- Kinetics: 700부류로 레이블링된 비디오가 65만개(비디오는 약 10초 분량)
- HAA500: 500부류로 레이블링된 비디오가 59만개. 원자 수준의 행동 부류 레이블링

(a) Kinetics 데이터셋

(b) HAA500 데이터셋

그림 10-20 행동 분류를 위한 데이터셋

10.4.3 행동 분류

■ 행동 분류action classification 모델

- 3차원 컨볼루션
  - 비디오는 3차원 공간, 즉 spatio-temporal 공간을 형성하므로 3차원 컨볼루션은 자연스런 선택

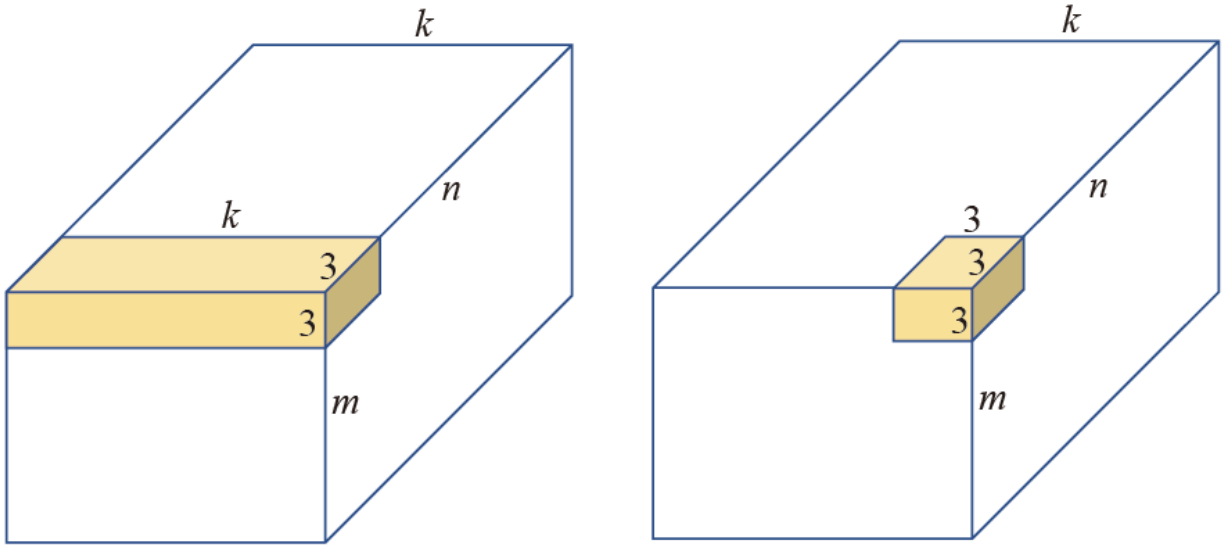
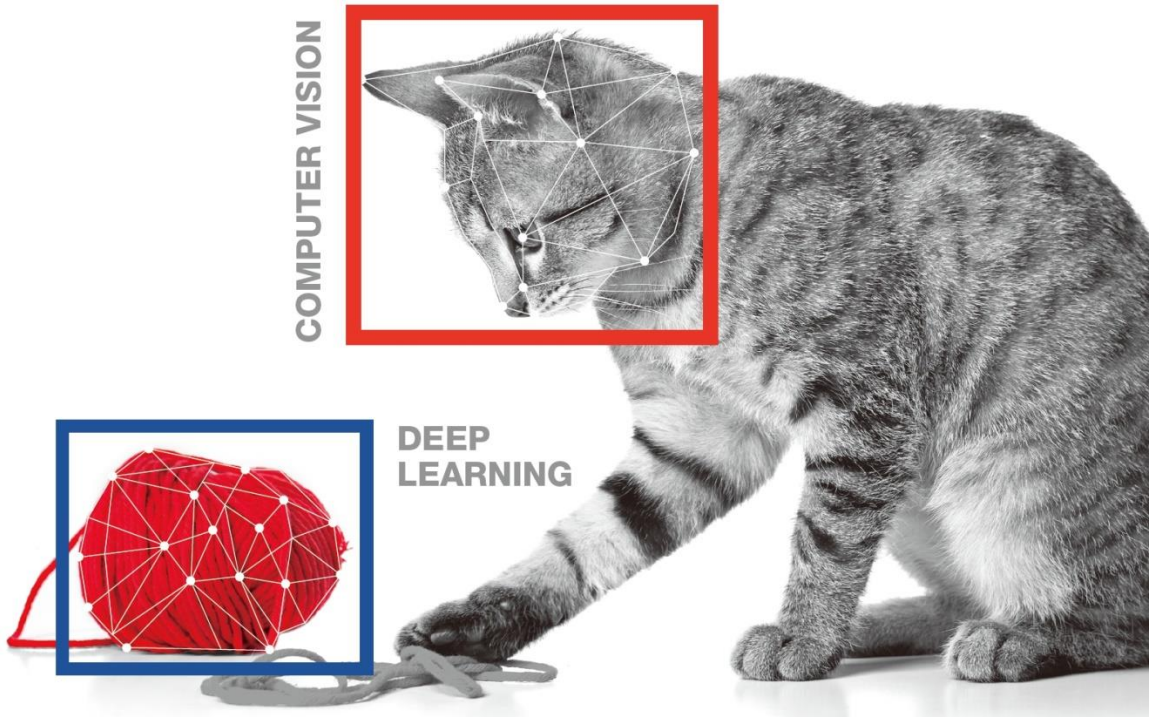


그림 10-21 2D 컨볼루션(왼쪽)과 3D 컨볼루션(오른쪽)

- 실제로는 2차원 컨볼루션을 더 많이 사용
  - 컨볼루션 신경망으로 프레임별 특징을 추출하고 LSTM으로 시계열 처리하는 모델
  - 프레임에서 추출한 특징과 광류를 결합하는 방법

10.4.3 행동 분류

- 행동 분류는 아주 어려운 문제로 아직 사람 성능에 미치지 못함
- 행동 예측action prediction은 행동 분류보다 어려움 ← 중요한 미래 연구



컴퓨터 비전과 딥러닝

감사합니다.