

영상처리 실제

Image Processing Practice

신재혁

naezang@cbnu.ac.kr

9장 인식

Preview

■ 아이의 영상 해석 능력

- 개의 위치와 움직임, 표정을 분석하면서 같이 놀
- 심한 혼재_{clutter}와 가림_{occlusion}에 강인_{robust}하고 조명 변환에 불변



그림 9-1 뛰어난 인간의 비전

■ 인식_{recognition}은 넓은 의미의 단어

- 분류, 검출, 분류, 추적 등의 세부 문제를 포함
- 이 장은 세부 문제 별로 고전 알고리즘을 간략히 소개하고 딥러닝을 상세히 설명

인식

9.1 인식이란

■ 앞 장에서의 인식

- 5.5~5.6절에서는 RANSAC과 호모그래피 이용해 특정 물체를 인식
- 6.4절에서는 교통약자 표지판 비전 에이전트

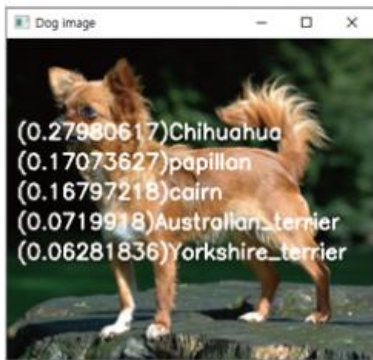
← 사례 분류에 국한

■ 이 장에서는 분류, 검출, 분할, 추적, 행동 분석 포함하는 인식으로 확장

9.1.1 인식의 세부 문제

■ 사람과 달리 컴퓨터 비전은 세부 문제로 구분

- 세부 문제는 별도의 데이터셋과 성능 기준을 가지고 알고리즘을 개발



(a) 분류



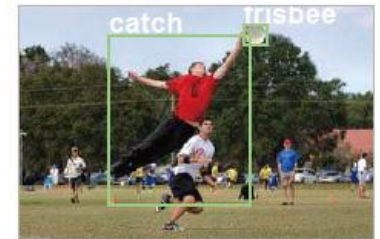
(b) 검출



(c) 분할



(d) 추적(0초, 3초, 6초 순간)



(e) 행동 분류

그림 9-2 인식을 여러 세부 문제로 나누는 컴퓨터 비전

9.1.1 인식의 세부 문제

■ 분류_{classification}

- 영상에 있는 물체의 부류를 알아내는 문제. 보통 부류 확률 벡터를 출력
 - 예) COCO 데이터셋은 사람, 자전거, 자동차 등 80부류
- 내 차, 어린이 교통표지판과 같은 특정 물체 알아내는 사례 분류_{instance classification}
- 고양이나 자전거처럼 물체 부류를 알아내는 범주 분류_{categorical classification}

■ 검출_{detection}

- 영상에서 물체를 찾아 직사각형(바운딩 박스)으로 위치 표현
- 보통 부류 확률도 같이 출력함. 보통 부류는 미리 정해져 있음

■ 분할_{segmentation}

- 물체가 점유하는 영역, 즉 화소 집합을 지정. 부류 확률도 같이 출력
- 의미 분할_{semantic segmentation}과 사례 분할_{instance segmentation}

9.1.1 인식의 세부 문제

■ 추적_{tracking}

- 비디오에 나타난 물체의 이동 궤적을 표시
- 시각 물체 추적_{VOT; Visual Object Tracking}과 다중 물체 추적_{MOT; Multiple Object Tracking}

■ 행동 분류

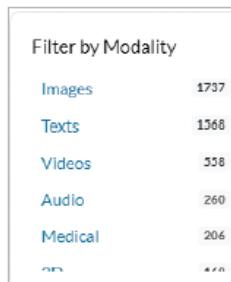
- 물체가 수행하는 행동의 종류를 알아내는 문제
- 현재는 사람의 행동을 몇 가지로 분류하는 수준

9.1.2 데이터셋과 방법론의 공진화

■ 컴퓨터 비전은 데이터셋의 발전에 힘입어 혁신을 이룸

- 예전에 MNIST와 CIFAR-10은 꽤 어려운 데이터셋이었는데 지금은 장난감 수준
- ImageNet은 딥러닝 발전에 결정적인 공헌

■ 데이터셋 소개 문헌



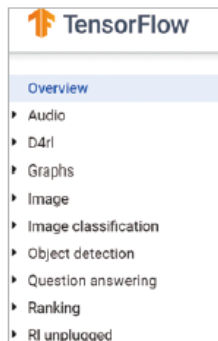
Filter by Modality	
Images	1737
Texts	1368
Videos	558
Audio	260
Medical	206
...	...

(a) PapersWithCode



Image data
1.1 Facial recognition
1.2 Action recognition
1.3 Object detection and recognition
1.4 Handwriting and character recognition
1.5 Aerial images
1.6 Other images

(b) 위키피디아



TensorFlow
Overview
▶ Audio
▶ D4rl
▶ Graphs
▶ Image
▶ Image classification
▶ Object detection
▶ Question answering
▶ Ranking
▶ RL unplugged

(c) 텐서플로



(d) AI 허브

그림 9-3 데이터셋을 소개하는 문헌

9.1.2 데이터셋과 방법론의 공진화

■ 유명 데이터셋과 대회

- 도전적인 데이터셋은 자연스럽게 대회로 이어짐. 연구팀은 대회를 통해 경쟁하면서 아이디어를 공유하며 동반 성장
- PASCAL VOC
 - 20부류 50만장 영상(처음 4부류에서 20부류로 확장)
- ImageNet
 - 21,841부류 1400만장 영상
 - ILSVRC대회는 1000부류를 뽑고 1백만장 훈련, 5만장 검증, 15만장 테스트 집합
- COCO
 - 80부류에 대한 33만 장 영상
 - 검출과 분할, 설명문 달기 데이터셋 제공
- OpenImages

9.1.2 데이터셋과 방법론의 공진화



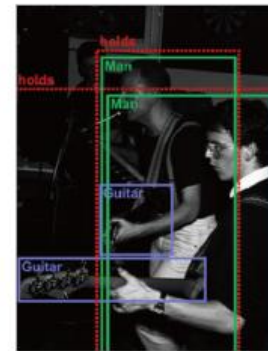
(a) Pascal VOC



(b) ImageNet



(c) COCO



(d) Open Images

그림 9-4 도전적인 데이터셋의 영상 샘플

9.1.2 데이터셋과 방법론의 공진화

■ 특정 분야 데이터셋

- DeepFashion: 50부류에 대한 80만장 영상
- Food-101: 101부류에 대한 10만장 가량 영상
- CheXpert: 65,240환자의 폐 엑스레이 사진 224,316장



(a) DeepFashion[Liu2016b]



(b) Food-101[Bossard2014]



(c) CheXpert[Irvin2019]

그림 9-5 특수 분야의 데이터셋

9.1.2 데이터셋과 방법론의 공진화

- 합리적이고 공정한 성능 평가 척도_{metric}가 중요
 - 세부 문제에 따라 다양한 척도
 - 대회는 척도와 척도를 계산하는 소프트웨어 모듈 제공하여 공정성 유지
 - 훈련 집합은 영상과 참값 제공하지만 테스트 집합은 영상만 제공
 - 참가자는 훈련 집합으로 모델 학습(별도의 검증 집합으로 하이퍼 매개변수 최적화)
 - 테스트 집합의 참값을 예측하여 결과를 서버로 전송
 - 서버는 자동 채점하여 리더보드에 공개

9.1.2 데이터셋과 방법론의 공진화

■ 영상 레이블링

- 많은 노동력 필요. 검출 레이블링은 분류 레이블링보다 어렵고, 분할 레이블링은 검출 레이블링보다 어려움
- 무료 레이블링 도구 여럿
- 대용량 데이터셋은 클라우딩을 활용하여 제작(예, mechanical Turk)

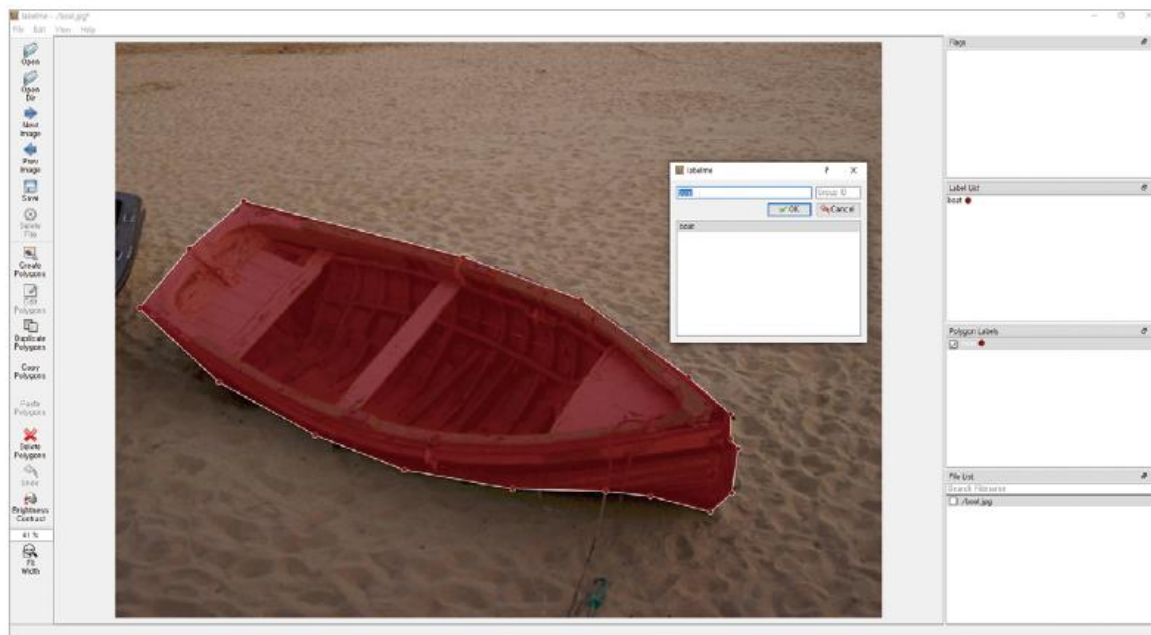


그림 9-6 데이터셋을 레이블링하는 도구인 LabelMe(<http://labelme.csail.mit.edu>)

TIP 위키피디아에서 'list of manual image annotation tools'를 검색하면 여러 도구를 살펴볼 수 있다.

9.1.3 사람과 컴퓨터 비전의 인식 과정 비교

■ 사람과 컴퓨터 비전은 인식하는 방식이 크게 다름

- 예) 사람은 검출을 따로 수행하지 않고 분할만 수행하는 반면에 컴퓨터 비전은 보다 쉬운 검출 문제를 중요하게 취급(검출은 자율주행에서 중요)

■ 사람의 인식

- 세부 문제를 별도 수행하지 않고 동시 수행. 지식 표현과 추론 등의 지능 요소를 통해 명확한 의도_{intention}를 지니며, 의도에 따라 선택적 주의집중_{selective attention} 발휘
- 사람의 의도는 거친 수준과 세밀한 수준을 자연스럽게 조절하고, 전환에 틈새 없고 노력이 들지 않음. 자신의 내부 상태와 외부 환경과 밀접하게 상호작용하며 수시로 변함
- 문맥 사용에 능숙
- 동영상 처리하여 상대의 행동을 인식하고 은밀한 의도까지 추론

저지 같은 행색

꺼저지 않는

9.1.3 사람과 컴퓨터 비전의 인식 과정 비교

■ 컴퓨터 비전의 인식

- 세부 문제별로 독립적으로 해결. 대상 물체 또는 응용 과업에 따라 더욱 세부적인 문제로 구분하여 해결(예, 사과 따는 로봇 제작에서는 사과 검출에 집중)
- 능동적인 의도가 없음
 - 과업이 달성할 목적이 의도를 대신하며, 목적을 데이터 레이블링에 반영되다 수준
- 사람의 의도에 해당하는 것을 고안하고 학습 과정에 반영하는 연구는 먼 미래의 일([그림 1-4]의 인공지능 기술과 협동은 필수)

9.2 분류

■ 분류 문제

- 7~8장에서 다루었고 프로그래밍 실습을 여러 번 수행
- 여기서는 좀더 체계적으로 소개
 - 9.2.1항에서는 분류 문제를 구분하고 고전 알고리즘을 간략히 소개
 - 9.2.2항은 딥러닝 기법을 간략히 보충

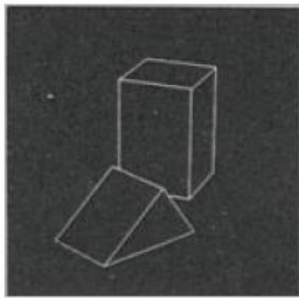
■ 사례 분류_{instance classification}와 범주 분류_{categorical classification}

- 사례 분류: 모양과 텍스처가 고정된 특정 물체를 분류(내 차나 교통표지판 같은 강체)
- 범주 분류: 코끼리나 자전거 같은 일반 부류의 물체를 분류(모양과 자세 변화가 심함)

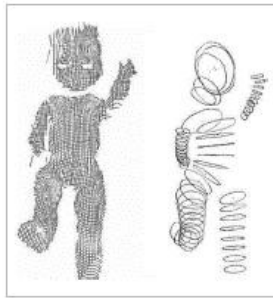
9.2.1 고전 방법론

■ 사례 분류

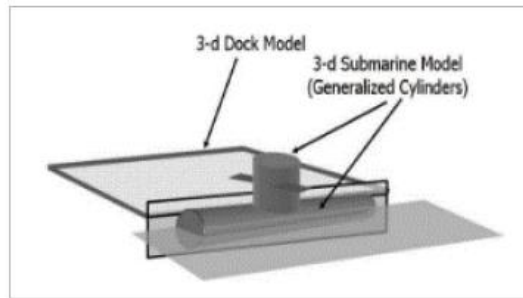
- 물체를 구성하는 직선과 곡선을 분석하는 기하학적 방법이 주류



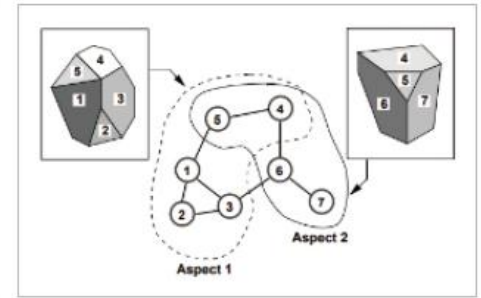
(a) 블록 세계



(b) 일반 실린더



(c) ACRONYM



(d) 양상 그래프

그림 9-8 사례 분류를 해결하는 고전 기법[Mundy2006]

- 2000년대 초에 SIFT 특징 추출 알고리즘과 RANSAC 매칭 알고리즘에 힘입어 획기적으로 발전하고 실용 시스템 등장

9.2.1 고전 방법론

■ 범주 분류

- 부류 내 변화 intra-class variation가 아주 심함(뛰는 고양이, 누워있는 고양이, 뒷태만 보이는 고양이 등)
- 혼재 clutter와 가림 occlusion을 허용하므로 더욱 어려움
- PASCAL VOC 대회가 연구 촉발. 대표적인 고전 방법은 부품 기반 part-based 방법
 - 매년 성능 개선이 있지만 혁신은 없음

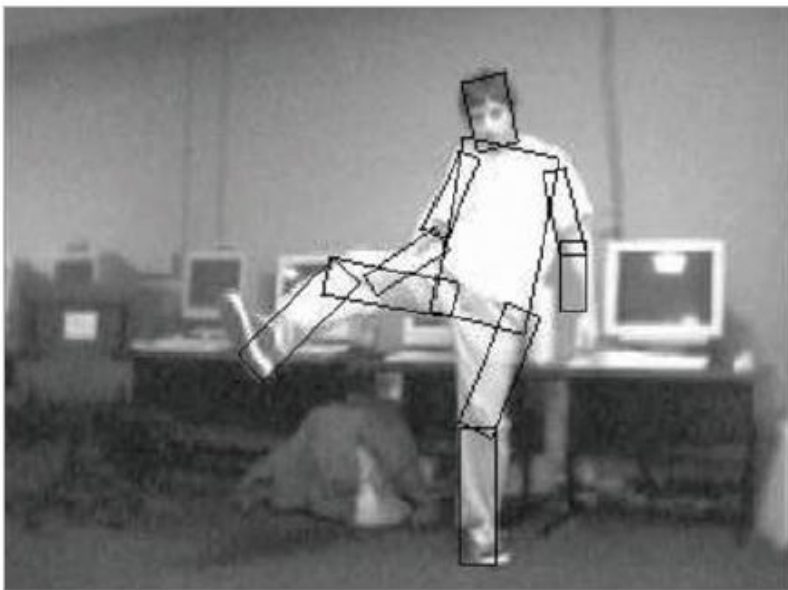


그림 9-9 부품 기반 방법[Felzenszwalb2005]

9.2.2 딥러닝 방법론

■ 딥러닝에서는 사례 분류는 풀렸다고 보고 범주 분류에 공을 들임

- ILSVRC 대회에서는 1~2회는 고전 방법이 우승하였으나, 3회(2012년)에 컨볼루션 신경망 기반의 AlexNet이 우승함에 따라 이후 딥러닝으로 패러다임 전환
- ImageNet의 성능 향상 추세([그림 9-10]은 5순위 정확률)

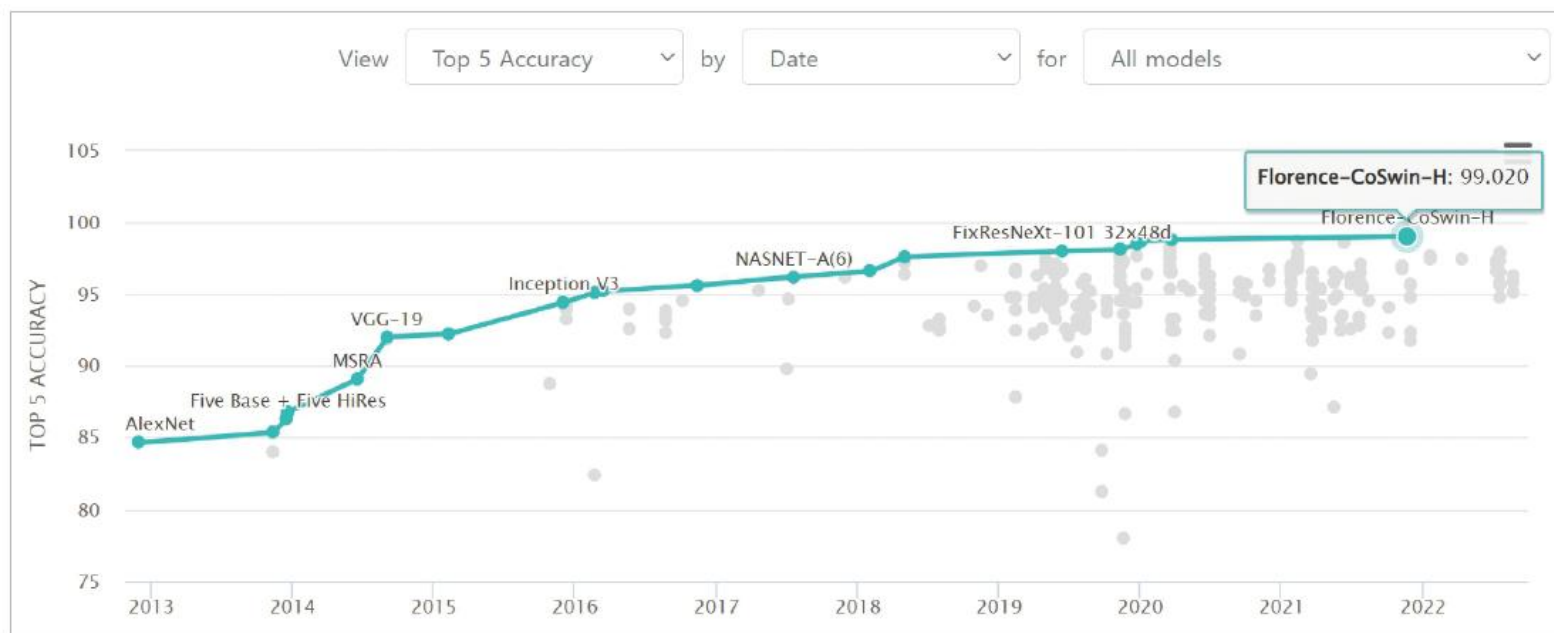


그림 9-10 ImageNet(ILSVRC 대회)의 5순위 정확률(<https://paperswithcode.com/sota/image-classification-on-imagenet>)

9.2.2 딥러닝 방법론

■ 미세 분류 fine-grained classification 문제

- 부류 내 변화가 크고, 부류 간 변화가 적어 아주 어려운 문제
- 여러 데이터셋: Stanford dogs, Stanford cars, CUB-200, iNat, Oxford 102 flowers 등



Laysan Albatross



Rusty Blackbird

Fish Crow

Brewer Blackbird

Shiny Cowbird

그림 9-11 부류 내 변화가 크고 부류 간 변화가 작은 미세 분류 문제(CUB-200 데이터셋)

9.2.2 딥러닝 방법론

■ 설명 가능

- 인간은 의사결정의 이유를 설명하는 능력이 뛰어나
- 딥러닝은 분류 결과에 대한 이유를 설명하지 못하는 치명적 한계
- 설명 가능_{explainability} 확보하려는 연구 활발: CAM과 GradCAM
- [그림 9-12]는 세 단계의 설명 수준을 예시



91.5% 확률로 붉은 날개
검은 새로 분류



하이라이트 영역을 보고
91.5% 확률로 붉은 날개
검은 새로 분류



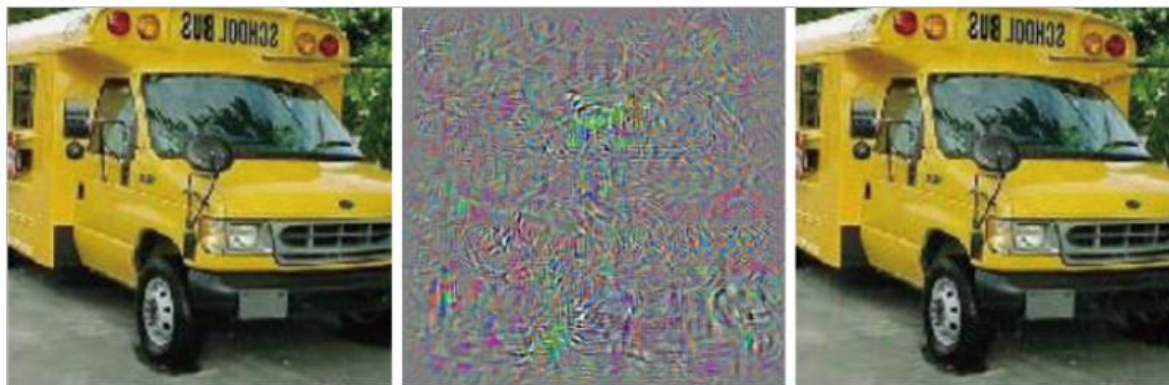
노란 띠와 붉은 반점을
보고 91.5% 확률로 붉은
날개 검은 새로 분류

그림 9-12 딥러닝의 설명 수준(왼쪽부터 설명 없음, 시각화 수준 설명, 문장 수준 설명)

9.2.2 딥러닝 방법론

■ 적대적 공격_{adversarial attack}과 방어 전략

- 딥러닝 모델을 속이려는 시도가 쉽게 통함(예, [그림 9-13(a)]는 버스 영상에 잡음을 섞으면 타조라고 분류하는 예시)
- 적대적 공격을 막는 연구 활발



(a) 자연 영상 교란[Szegedy2013]

그림 9-13 적대적 공격



(b) 교통 표지판에 스티커 부착

9.3 검출

■ 검출 문제

- 물체의 위치와 함께 부류 정보를 알아내야 하므로 분류보다 어려움
 - 검출 알고리즘은 물체 위치와 함께 부류 신뢰도_{confidence}를 나타내는 확률 벡터를 출력
- 딥러닝으로 전환하면서 획기적 발전 이룸(자율주행차는 대표적으로 성공적인 응용)

9.3.1 성능 척도

■ 성능 척도로는 COCO 대회 $mAP_{\text{mean Average Precision}}$ 를 주로 사용

- 물체 부류 각각에 대해 AP를 구하고 모든 부류에 대해 평균하면 mAP

■ 먼저 한 물체에 대한 $AP_{\text{Average Precision}}$ 계산 방법을 설명

- AP는 $IoU_{\text{Intersection over Union}}$ 를 기반으로 계산

$$IoU = \frac{area(B_{predict} \cap B_{GT})}{area(B_{predict} \cup B_{GT})} \quad (9.1)$$

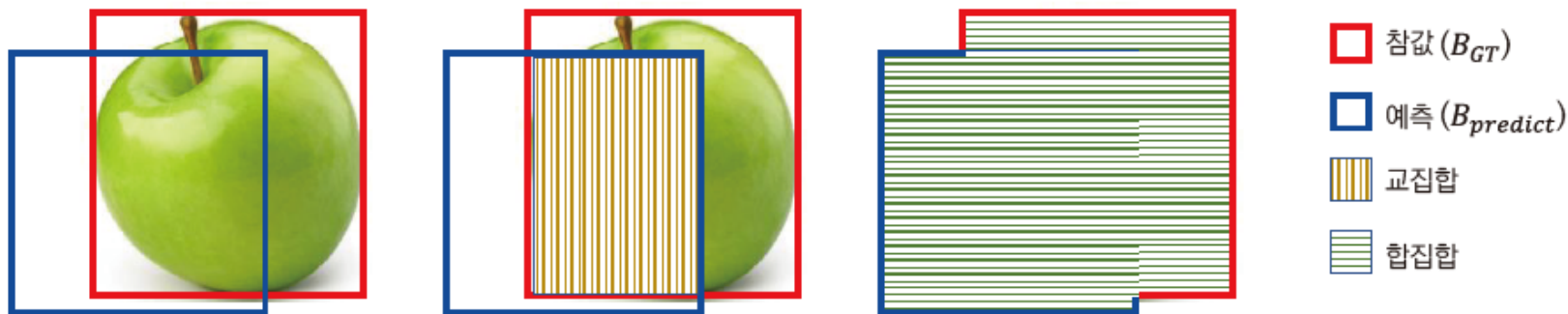
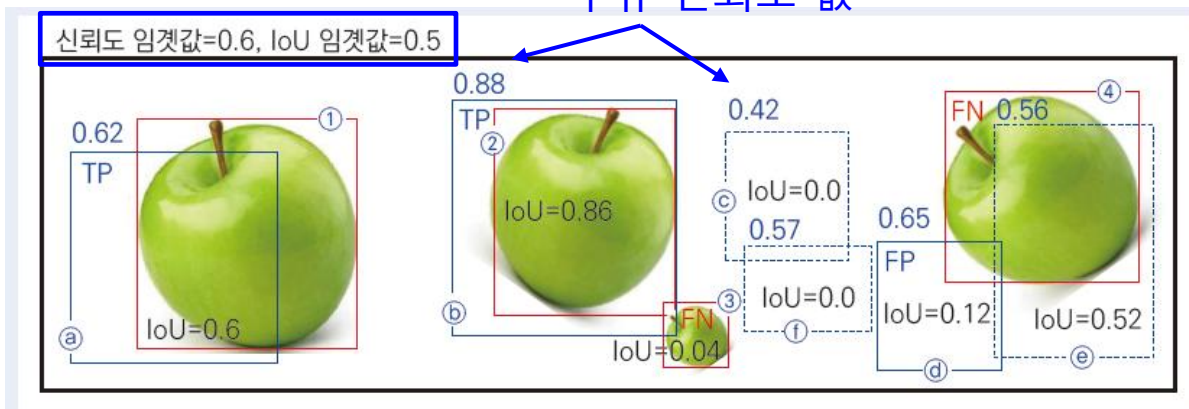


그림 9-14 IoU 계산

9.3.1 성능 척도

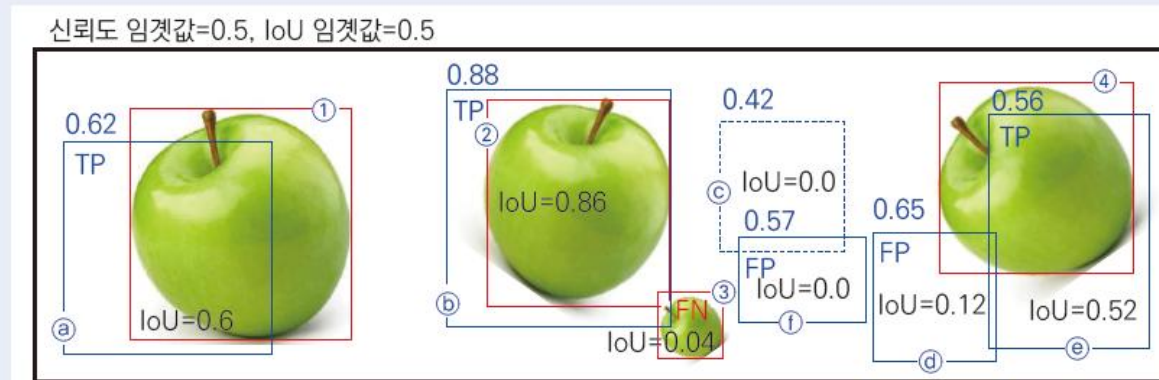
- 실제 상황에서는 [그림 9-15]와 같이 겹치는 박스가 여럿 나타남

부류 신뢰도 값



①~④는 참값 박스
a~f는 예측 박스

(a) 신뢰도 임계값을 0.6으로 설정



(b) 신뢰도 임계값을 0.5로 설정

그림 9-15 TP, FP, FN 세기(IoU 임계값=0.5)

9.3.1 성능 척도

▪ AP 계산 알고리즘

- 신뢰도 임계값을 넘는 예측 박스만 예측 목록에 넣음
- 예측 목록에 있는 박스를 신뢰도가 큰 것부터 순서대로 처리
- 현재 처리하는 예측 박스가 IoU 임계값을 넘으면 참 긍정(TP), 그렇지 않으면 거짓 긍정(FP)으로 판정(2개 이상의 참값 박스와 겹치는 경우 IoU가 최대인 쌍 사용)
- 참 긍정이 발생하면 쌍이 된 참값 박스를 참값 목록에서 제거(이중으로 쌍 맺는 경우 방지)
- 예측 목록을 다 처리했는데 쌍을 맺지 못해 남아 있는 참값 박스는 거짓 부정(FN) 판정
- 이렇게 구한 TP, FP, FN을 가지고 식 (5.15)의 정밀도와 재현률 계산

9.3.1 성능 척도

■ [예시 9-1]

- [그림 9-15(a)] 신뢰도 임계값 0.6, IoU 임계값 0.5인 경우
 - 신뢰도 임계값을 넘긴 예측 박스 목록은 {a, b, d}임. 신뢰도로 정렬하면 $b \rightarrow d \rightarrow a$
 - b를 처리. b는 2, 3과 겹치는데 2의 IoU가 0.86으로 더 크므로 b-2 쌍을 맺고 TP로 판단. 2를 참값 목록에서 제거
 - d를 처리. d는 4와 IoU가 0.12인데 IoU 임계값을 넘지 못해 d를 FP로 판정
 - a를 처리. a와 1은 IoU 임계값을 넘기므로 a-1 쌍을 맺고 TP 판정
 - 참값 목록에 남아있는 3, 4를 FN 판정

신뢰도 임계값이 0.6일 때: TP=2, FP=1, FN=2

→ 정밀도=2/3=0.666, 재현율=2/4=0.5

- [그림 9-15(b)] 신뢰도 임계값 0.5, IoU 임계값 0.5인 경우
 - 신뢰도 임계값 0.5로 낮추고 같은 과정을 적용하면

신뢰도 임계값이 0.5일 때: TP=3, FP=2, FN=1

→ 정밀도=3/5=0.6, 재현율=3/4=0.75

9.3.1 성능 척도

- 신뢰도 임계값을 0.0, 0.1, 0.2, ..., 0.9, 1.0으로 증가시키며 재현률과 정밀도를 측정하면

표 9-1 신뢰도 임계값에 따른 [그림 9-15] 상황의 재현율과 정밀도

신뢰도 임계값	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
재현율	0.75	0.75	0.75	0.75	0.75	0.75	0.5	0.25	0.25	0.0	0.0
정밀도	0.5	0.5	0.5	0.5	0.5	0.6	0.666	1.0	1.0	—	—

9.3.1 성능 척도

- 가로축과 세로축에 재현률과 정밀도를 배치하고 그래프를 그리면 정밀도-재현률 그래프

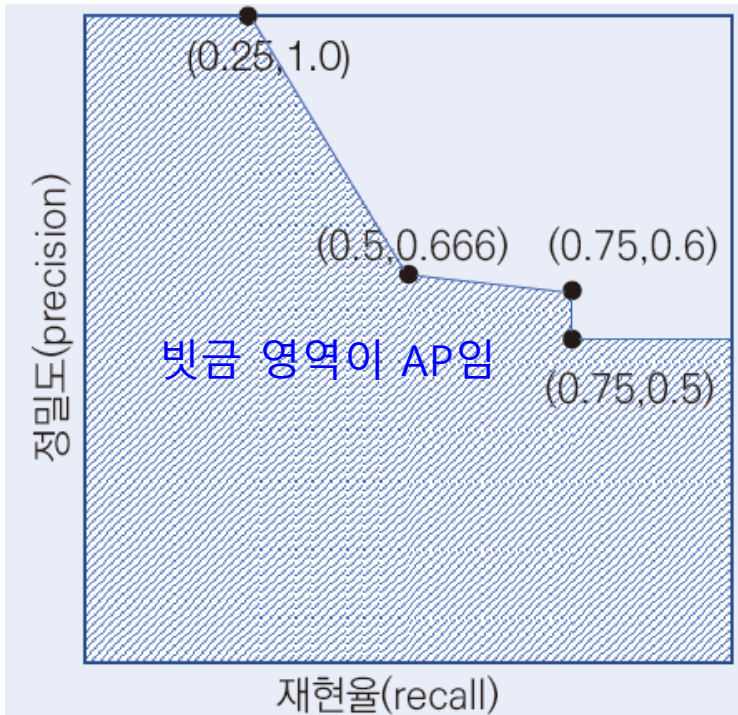


그림 9-16 정밀도-재현율 곡선

- 모든 부류에 대해 AP를 구한 다음 평균하면 mAP

9.3.2 고전 방법

■ 고전 방법의 초기 연구

- 주로 사람 또는 사람 얼굴에 국한해 진행
- PASCAL VOC 대회 이후 여러 부류로 확장
 - 2005년 1회 대회는 4개 부류(사람, 자전거, 자동차, 오토바이)
 - 2006년 2회 대회는 10개 부류로 확장
 - 2007년 3회 대회는 20부류로 확장
- Viola 알고리즘은 성공적인 얼굴 검출 방법의 대표
- HOG 특징을 사용하여 Viola 알고리즘을 사람 검출로 확장한 다음 PASCAL VOC의 20부류로 확장

9.3.2 고전 방법

■ 후보 영역에서 HOG_{Histogram Of Gradients} 특징 추출 과정

- 후보 영역을 128*64로 변환. 8*8 크기로 나누어 16*8개의 타일 생성
- 타일에서 특징 추출 과정
 - 64개 화소에서 그레이디언트 방향 계산하고 9개 구간으로 양자화
 - 히스토그램을 구해 9차원 특징 벡터 구성
- 4개 이웃 타일(노랑)을 이어 붙인 36차원 특징 벡터를 정규화(명암 변화에 둔감한 효과)

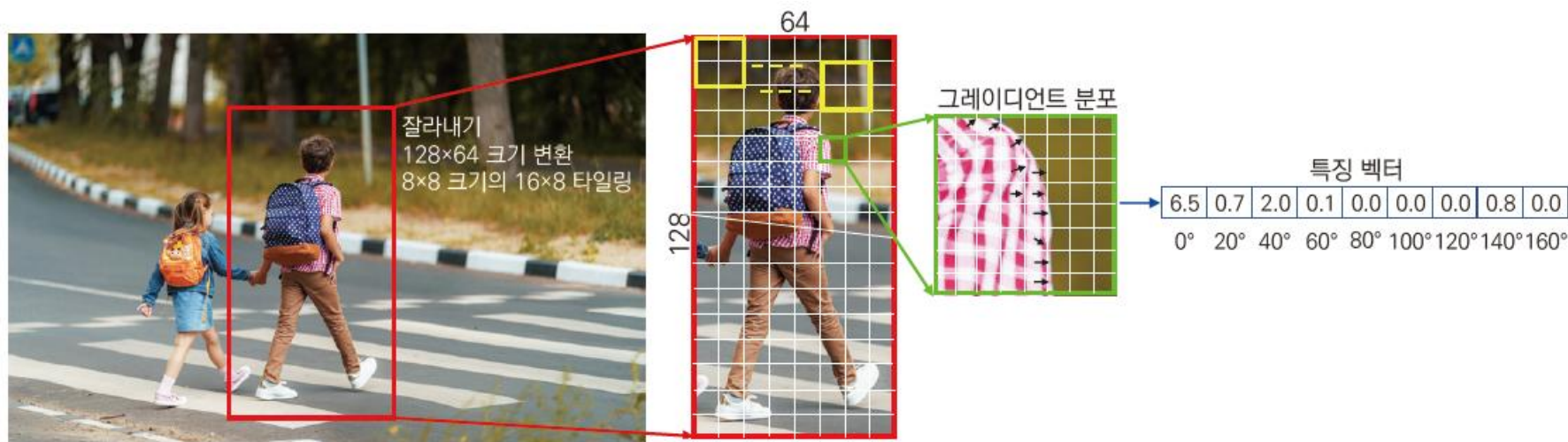


그림 9-18 HOG 특징 추출

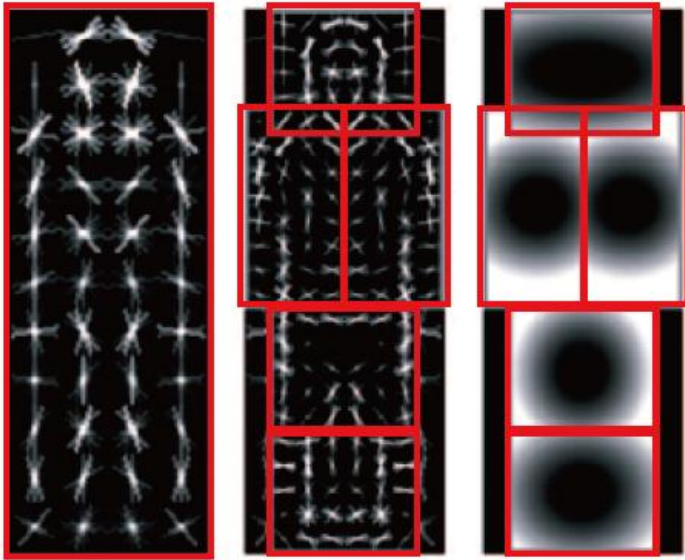
9.3.2 고전 방법

- 후보 영역의 사람 여부는 SVM 분류기 사용
- 후보 영역은 여러 해상도 영상에 슬라이딩 윈도우 적용하여 생성

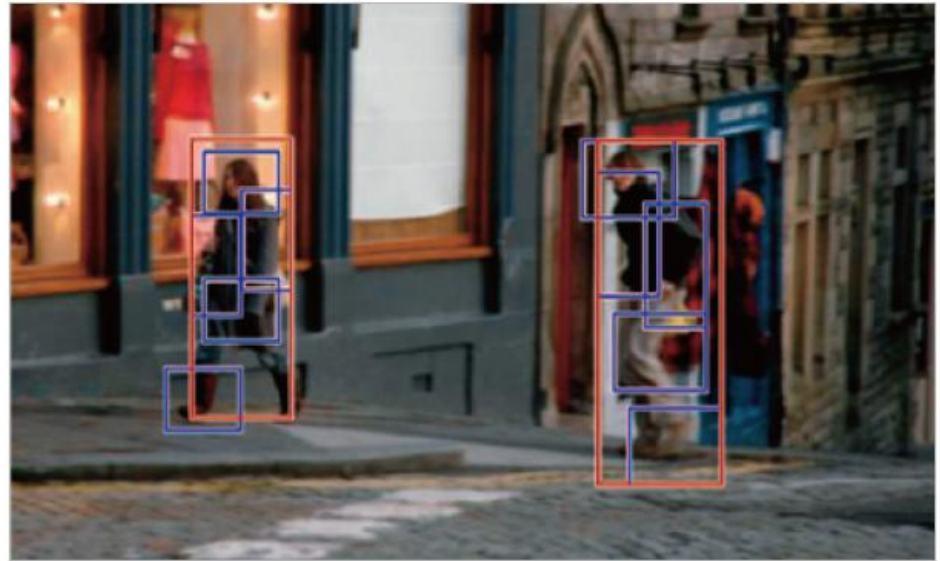
9.3.2 고전 방법

■ 이후 DPM_{Deformable Part Model} 등장

- 물체를 몇 개의 부품으로 모델링([그림 9-19(a)]는 사람을 5개 부품으로 모델링)
 - 왼쪽은 물체 전체, 중앙은 5개 부품, 오른쪽은 부품 발생 확률 분포(학습으로 알아냄)
 - HOG 특징 사용



(a) 사람의 DPM



(b) 사람을 검출한 사례

그림 9-19 DPM으로 사람 검출[Felzenszwalb2010]

9.3.3 컨볼루션 신경망: RCNN 계열

■ 딥러닝으로 전환

- 초기 ILSVRC 대회 검출 부문에서는 고전 방법론이 우승했는데, 2014년 컨볼루션 신경망을 사용한 RCNN이 우승하여 검출에서 컨볼루션 신경망 시대를 열다.

■ 두 단계 방법론과 한 단계 방법론

- RCNN이 대표하는 두 단계 방법: 후보 영역을 생성하고 분류를 통해 물체 여부 판정
- YOLO가 대표하는 한 단계 방법: 위치와 부류 정보를 묶어 참값을 만들고 신경



그림 9-20 물체 검출을 위한 컨볼루션 신경망의 발전(두 단계의 RCNN 계열과 한 단계의 YOLO 계열)

9.3.3 컨볼루션 신경망: RCNN 계열

■ RCNN의 처리 과정([그림 9-21])

- 영역 제안(후보 영역 생성) 단계
 - 물체가 있을 가능성이 높은 영역을 찾는 단계
 - 다양한 알고리즘 중에 선택적 탐색(selective search) 알고리즘 활용(슈퍼 화소 분할한 다음 군집화를 통해 후보 영역 생성)
- 영역 분류 단계
 - 227*227로 정규화하고 컨볼루션 신경망으로 4096차원 특징 추출
 - SVM으로 영역 분류

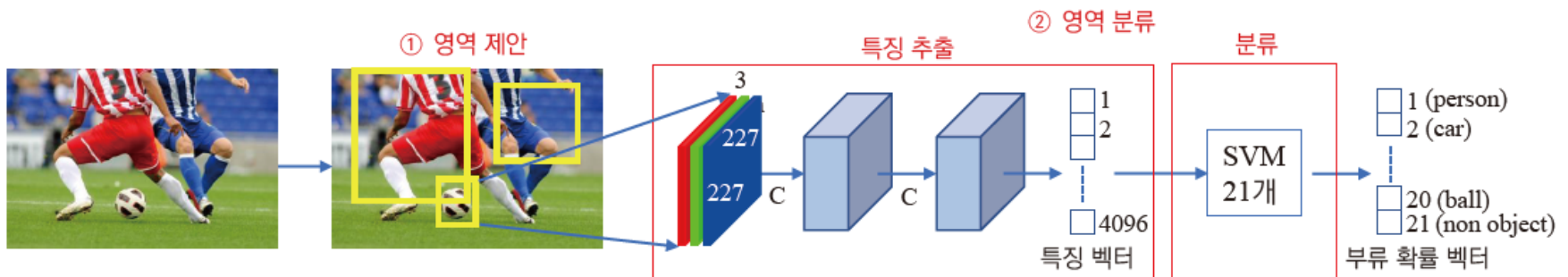


그림 9-21 RCNN의 처리 과정

9.3.3 컨볼루션 신경망: RCNN 계열

■ RCNN은 물체 검출에 컨볼루션 신경망을 도입한 점에서 혁신적

- 하지만 영역 제안에 고전 방법 사용하고 분류에 SVM 사용하는 한계
- 영역마다 독립적으로 분류 수행하므로 매우 느림

9.3.3 컨볼루션 신경망: RCNN 계열

■ fast RCNN의 처리 과정([그림 9-22])

- 영역 제안은 여전히 선택적 탐색 알고리즘 사용
- 영역 분류는 신경망 사용
 - 컨볼루션층으로 특징 추출
 - 완전연결층으로 분류(부류 확률 벡터)와 회귀(박스 위치)

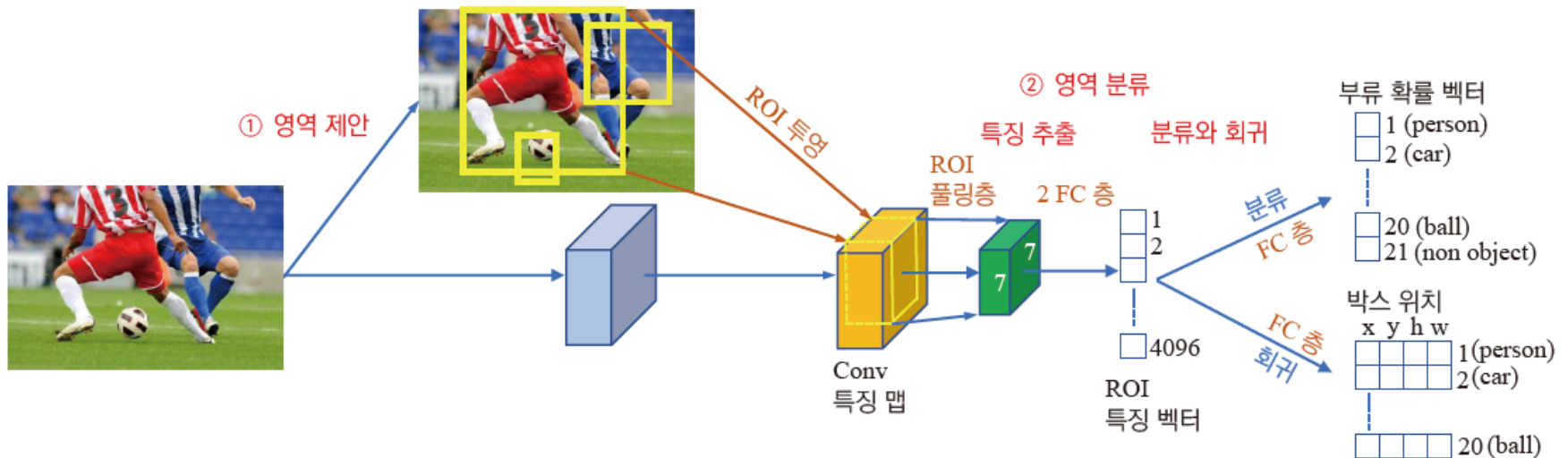


그림 9-22 fast RCNN의 구조

9.3.3 컨볼루션 신경망: RCNN 계열

■ 다중 과업 손실 함수와 전이 학습 사용

- 분류와 회귀를 한꺼번에 학습

$$J = J_{classification} + \lambda J_{regression}$$

- VGGNet이나 GoogLeNet을 미세 조정하여 전이 학습

■ 한계

- 선택적 탐색은 한 장 영상 처리하는데 CPU에서 2초 가량 소요
- 영역 제안 단계가 병목

9.3.3 컨볼루션 신경망: RCNN 계열

■ faster RCNN의 처리 과정([그림 9-23])

- 영역 제안을 RPN_{Region Proposal Network}으로 수행. 속도와 정확률을 획기적으로 개선
- 2015년 ILSVRC와 COCO 대회 우승하여 컨볼루션 신경망만으로 검출하는 시대 열다.

3*3 필터로 추출한 512차원 특징 벡터에
1*1 컨볼루션 적용하여 물체 여부와 박스 정보 추출
9번 적용하여 9개 앵커 생성

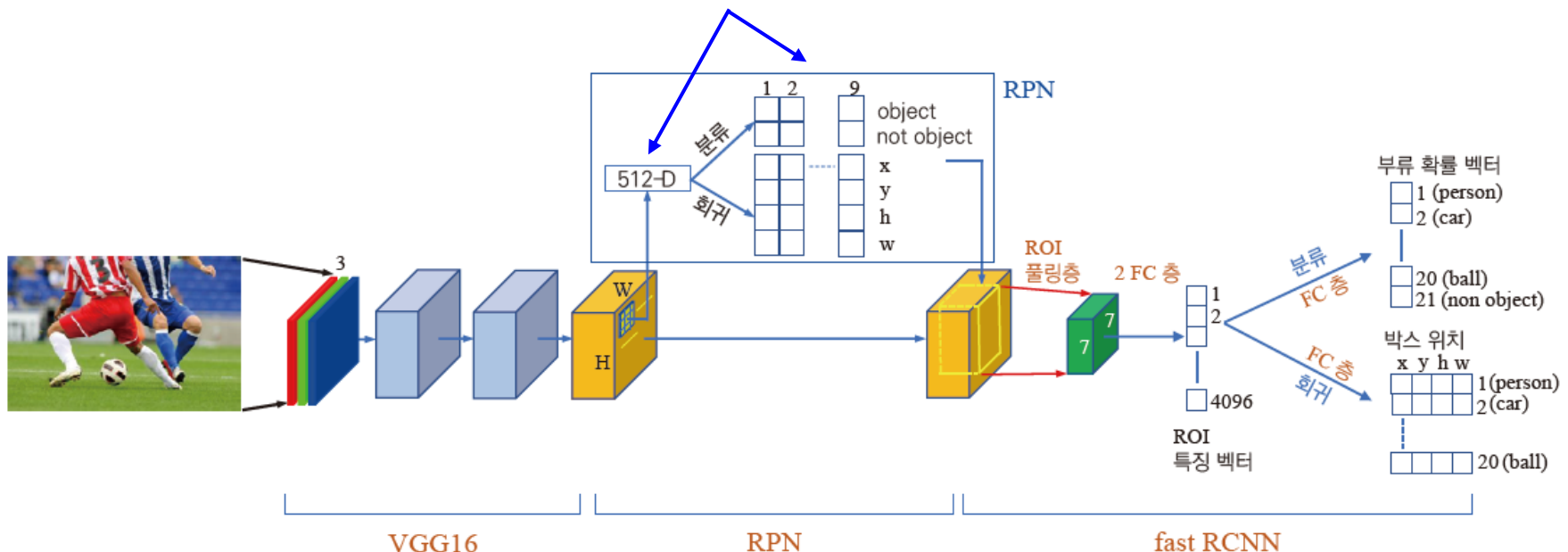


그림 9-23 faster RCNN의 구조

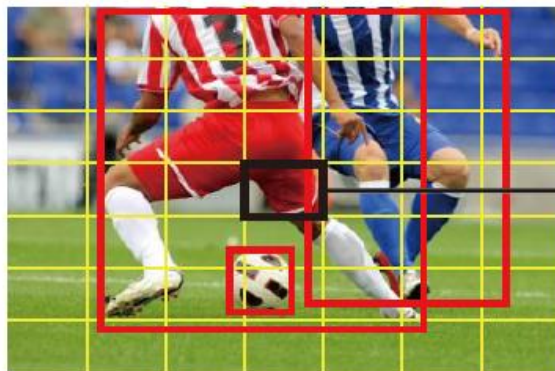
9.3.4 컨볼루션 신경망: YOLO 계열

■ YOLO_{You Only Look Once}는 RCNN보다 정확률 떨어지지만 월등히 빠름

- 물체 위치와 부류 정보를 회귀를 통해 한꺼번에 알아내는 한 단계 방법

■ YOLO의 처리 과정([그림 9-24])

- 영상을 $s*s$ 격자로 분할(실험에서는 $s=7$ 사용)
- 물체 중심에 해당하는 칸이 물체를 책임짐
 - 예) 빨간 유니폼 선수 영역의 중심은 검정 칸에 놓이므로 검정 칸이 책임을 짐
 - 박스의 위치(x,y,w,h)와 신뢰도(o) 표현. 한 칸은 물체를 두 개까지 책임질 수 있음
 - $p_1 \sim p_{80}$ 은 물체 부류 확률



$(x_1, y_1, w_1, h_1, o_1, x_2, y_2, w_2, h_2, o_2, p_1, p_2, p_3, \dots, p_{80})$
90차원 벡터

그림 9-24 YOLO가 물체의 위치와 부류를 표현하는 방식

9.3.4 컨볼루션 신경망: YOLO 계열

■ YOLO 신경망 구조

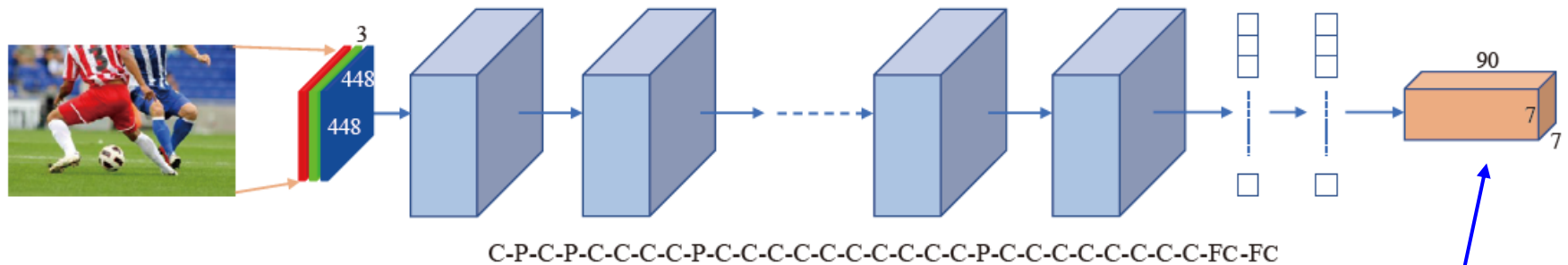


그림 9-25 YOLO가 사용하는 컨볼루션 신경망 구조

7*7=49개 칸
한 칸은 90차원 벡터로 표현

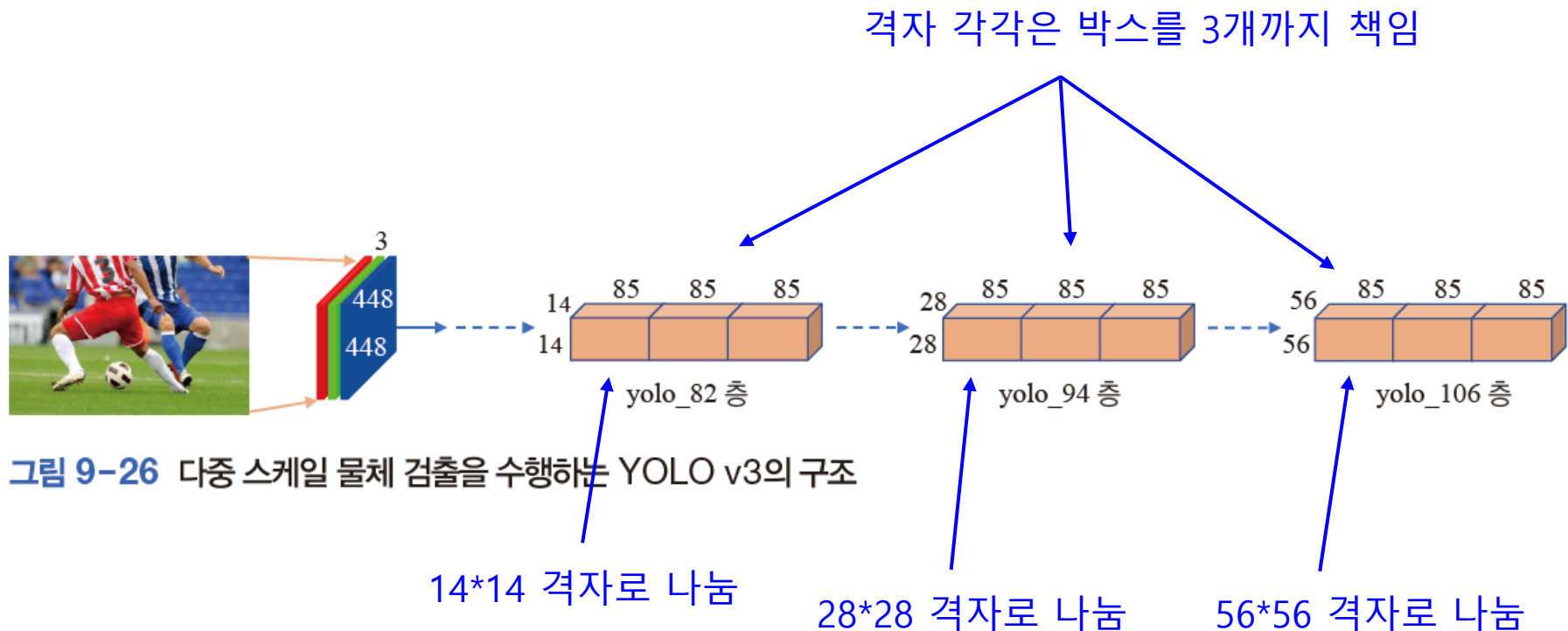
■ 학습

- 90차원 벡터는 박스 정보와 원핫 코드의 부류 정보가 섞여 있음
- 5개 항으로 구성된 손실 함수 사용하여 통째 학습

9.3.4 컨볼루션 신경망: YOLO 계열

■ YOLO의 버전업

- YOLO v3은 여러 스케일을 표현하여 다양한 크기의 물체 검출



9.3.4 컨볼루션 신경망: YOLO 계열


■ 정지 영상에서 물체 검출하는 프로그래밍

프로그램 9-1

YOLO v3으로 정지 영상에서 물체 검출하기

```
01 import numpy as np
02 import cv2 as cv
03 import sys
04
05 def construct_yolo_v3():
06     f=open('coco_names.txt', 'r')
07     class_names=[line.strip() for line in f.readlines()]
08
09     model=cv.dnn.readNet('yolov3.weights','yolov3.cfg')
10     layer_names=model.getLayerNames()
11     out_layers=[layer_names[i-1] for i in model.getUnconnectedOutLayers()]
12
13     return model,out_layers,class_names
14
```

사전 학습 모델을 읽어 YOLO 구성



9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

```
15 def yolo_detect(img,yolo_model,out_layers): ← YOLO 모델로 물체를 검출
16     height,width=img.shape[0],img.shape[1]
17     test_img=cv.dnn.blobFromImage(img,1.0/256,(448,448),(0,0,0),swapRB=True)
18
19     yolo_model.setInput(test_img)
20     output3=yolo_model.forward(out_layers)
21
22     box,conf,id=[],[],[]                # 박스, 신뢰도, 부류 번호
23     for output in output3:
24         for vec85 in output:
25             scores=vec85[5:]
26             class_id=np.argmax(scores)
27             confidence=scores[class_id]
28             if confidence>0.5:           # 신뢰도가 50% 이상인 경우만 취함
29                 centerx,centery=int(vec85[0]*width),int(vec85[1]*height)
30                 w,h=int(vec85[2]*width),int(vec85[3]*height)
31                 x,y=int(centerx-w/2),int(centery-h/2)
32                 box.append([x,y,x+w,y+h])
33                 conf.append(float(confidence))
34                 id.append(class_id)
35
36     ind=cv.dnn.NMSBoxes(box,conf,0.5,0.4)
37     objects=[box[i]+[conf[i]]+[id[i]] for i in range(len(box)) if i in ind]
38     return objects
```

9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

```
39
40 model,out_layers,class_names=construct_yolo_v3()           # YOLO 모델 생성
41 colors=np.random.uniform(0,255,size=(len(class_names),3)) # 부류마다 색깔
42
43 img=cv.imread('soccer.jpg')
44 if img is None: sys.exit('파일이 없습니다.')
45
46 res=yolo_detect(img,model,out_layers)                       # YOLO 모델로 물체 검출
47
48 for i in range(len(res)):                                   # 검출된 물체를 영상에 표시
49     x1,y1,x2,y2,confidence,id=res[i]
50     text=str(class_names[id])+'.%.3f'%confidence
51     cv.rectangle(img,(x1,y1),(x2,y2),colors[id],2)
52     cv.putText(img,text,(x1,y1+30),cv.FONT_HERSHEY_PLAIN,1.5,colors[id],2)
53
54 cv.imshow("Object detection by YOLO v.3",img)
55
56 cv.waitKey()
57 cv.destroyAllWindows()
```

9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

1.00 신뢰도로 person 부류

0.941 신뢰도로 person 부류



0.999 신뢰도로 sports ball 부류

9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

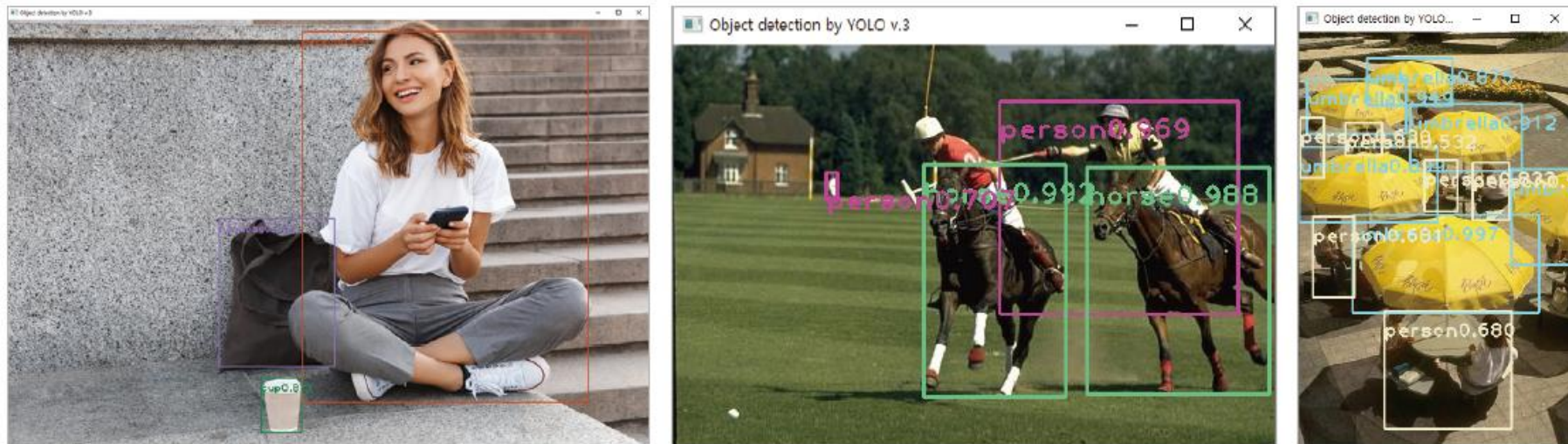


그림 9-27 YOLO v3으로 물체를 검출하는 여러 사례

9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

■ 비디오에서 물체 검출하는 프로그래밍

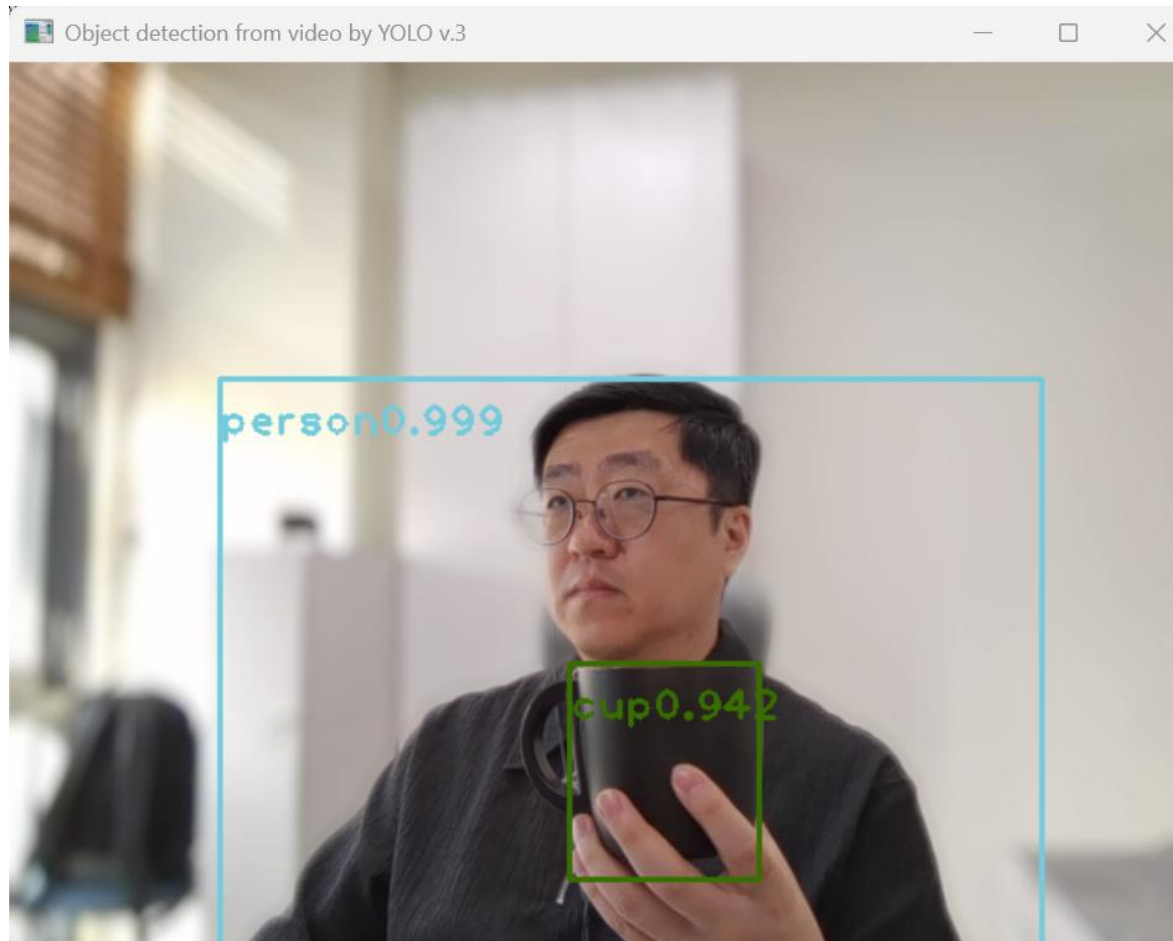
프로그램 9-2

YOLO v3으로 비디오에서 물체 검출하기

1~41행은 [프로그램 9-1]과 같음

```
43 cap=cv.VideoCapture(0,cv.CAP_DSHOW)
44 if not cap.isOpened(): sys.exit('카메라 연결 실패')
45
46 while True:
47     ret,frame=cap.read()
48     if not ret: sys.exit('프레임 획득에 실패하여 루프를 나갑니다.')
49
50     res=yolo_detect(frame,model,out_layers)
51
52     for i in range(len(res)):
53         x1,y1,x2,y2,confidence,id=res[i]
54         text=str(class_names[id])+'.3f'%confidence
55         cv.rectangle(frame,(x1,y1),(x2,y2),colors[id],2)
56         cv.putText(frame,text,(x1,y1+30),cv.FONT_HERSHEY_PLAIN,1.5,colors[id],2)
57
58     cv.imshow("Object detection from video by YOLO v.3",frame)
59
60     key=cv.waitKey(1)
61     if key==ord('q'): break
62
63 cap.release()      # 카메라와 연결을 끊음
64 cv.destroyAllWindows()
```

9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출



9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

■ 비디오 처리량 측정하기

프로그램 9-3

YOLO v3의 비디오 처리량 측정하기

1~41행은 [프로그램 9-1]과 같음

```
43 cap=cv.VideoCapture(0,cv.CAP_DSHOW)
44 if not cap.isOpened(): sys.exit('카메라 연결 실패')
45
46 import time
47
48 start=time.time()
49 n_frame=0
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
56     for i in range(len(res)):
57         x1,y1,x2,y2,confidence,id=res[i]
58         text=str(class_names[id])+'.3f'%confidence
59         cv.rectangle(frame,(x1,y1),(x2,y2),colors[id],2)
60         cv.putText(frame,text,(x1,y1+30),cv.FONT_HERSHEY_PLAIN,1.5,colors[id],2)
61
62     cv.imshow("Object detection from video by YOLO v.3",frame)
63     n_frame+=1
64
65     key=cv.waitKey(1)
66     if key==ord('q'): break
```

9.3.5 프로그래밍 실습: YOLO v3으로 물체 검출

```
67
68 end=time.time()
69 print('처리한 프레임 수=',n_frame,', 경과 시간=',end-start,'\n초당 프레임 수=',n_frame/
      (end-start))
70
71 cap.release()          # 카메라와 연결을 끊음
72 cv.destroyAllWindows()
```

처리한 프레임 수= 82, 경과 시간= 29.883725881576538
초당 프레임 수= 2.7439684169554437

 초당 2.74 프레임 처리

9.4 분할

■ 고전 알고리즘

- 4.4절(정규화 절단 등)과 4.5절(GrabCut 등)
- 6장 [프로그램 6-3]의 오림 비전 에이전트: 관심 물체 잘라내기
- 분할된 영역이 어떤 물체인지 알아내지 못하는, 즉 의미를 모르는 분할 알고리즘
- 딥러닝이 등장하면서 경쟁력 상실

9.4 분할

■ 딥러닝 분할

- 셀 수 있는 thing 물체(자동차, 사람 등), 셀 수 없는 stuff 물체(하늘, 도로 등)
- 의미 분할_{semantic segmentation}: 모든 화소에 부류 할당(thing과 stuff 모두 분할), 같은 부류는 구분하지 않음
- 사례 분할_{instance segmentation}: thing만 분할, 같은 부류 물체 여럿이면 번호 붙여 구분
- 총괄 분할_{panoptic segmentation}: 모든 화소에 부류 할당(thing과 stuff 모두 분할), 같은 부류 물체가 여럿이면 번호 붙여 구분



(a) 예제 영상



(b) 의미 분할



(c) 사례 분할



(d) 총괄 분할

그림 9-28 세 종류의 영상 분할 문제

9.4.1 성능 척도와 데이터셋

■ 분할을 밀집 분류 문제로 간주하고 분류 척도 사용

- 분할은 화소마다 부류를 지정하는 문제로 볼 수 있음
- 화소 정확률, 즉 PA(pixel accuracy)로 성능 측정 가능

$$PA = \frac{\sum_{i=0, C} p_{ii}}{\sum_{i=0, C} \sum_{j=0, C} p_{ij}} \quad (9.2)$$

- 부류별 PA의 평균이 평균 화소 정확률, 즉 MPA(mean pixel accuracy)

$$MPA = \frac{1}{C+1} \sum_{i=0, C} \frac{p_{ii}}{\sum_{j=0, C} p_{ij}} \quad (9.3)$$

9.4.1 성능 척도와 데이터셋

■ IoU에 기반한 척도: AP와 mPA(9.3.1의 검출 척도와 같음), Dice 계수

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \quad (9.4) \quad \text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \quad (9.5)$$

■ [예시 9-2]

참값						예측값						$A \cup B$						$A \cap B$					
0	0	0	0	0	0	0	0	0	0	0	0												
0	2	2	2	0	0	0	2	2	2	2	0		2	2	2	2			2	2	2		
2	2	2	2	1	0	0	2	2	1	2	0	2	2	2	2	2			2	2			
2	2	2	1	1	0	0	2	2	1	1	0	2	2	2					2	2			
0	2	2	1	1	0	0	2	2	1	1	0		2	2					2	2			
0	2	2	1	1	0	0	0	2	2	1	0		2	2	2					2			

(a) 참값과 예측값

(b) 영역 2의 IoU와 Dice 계수 계산 과정

그림 9-29 분할 성능을 측정하는 예시

$$\text{PA} = \frac{29}{36} = 0.8056, \text{MPA} = \frac{1}{3} \left(\frac{14}{15} + \frac{5}{7} + \frac{10}{14} \right) = 0.7873$$

$$\text{IoU} = \frac{10}{17} = 0.5882, \text{Dice} = \frac{2 \times 10}{14 + 13} = 0.7407$$

9.4.1 성능 척도와 데이터셋

■ 분할 데이터셋

- [그림 9-4]가 소개한 PASCAL VOC, ImageNet, COCO, OpenImages는 분할 레이블 제공
- 특수 목적 또는 확장된 데이터셋
 - 도심 도로 장면 분할 Cityscapes
 - 유튜브 비디오 분할 Youtube-Objects
 - 자율주행 KITTI
 - RGB-D 분할 NYU-Depth V2
 - COCO 확장한 LVIS



(a) Cityscapes(취리히)



(b) LVIS



(c) NYU-Depth V2

그림 9-30 분할에 쓰는 다양한 데이터셋

9.4.2 의미 분할을 위한 FCN

■ FCN Fully Convolutional Networks

- FCN은 분할을 처음 시도한 성공적인 컨볼루션 신경망
- 이후 다양한 아이디어로 개선된 뛰어난 성능의 모델이 여럿 등장
- 이름이 뜻하듯이 컨볼루션층만으로 구성됨

9.4.2 의미 분할을 위한 FCN

■ 구조와 동작

- 입력은 $m \times n \times 3$ 컬러 영상, 출력은 $m \times m \times (C+1)$ 맵
- 출력 맵의 각 화소는 $C+1$ 차원의 부류 확률 벡터(C 개 부류+배경)
 - 출력층은 화소별로 softmax 적용하여 확률 벡터 출력
 - 참값은 화소별로 원핫 코드 표현

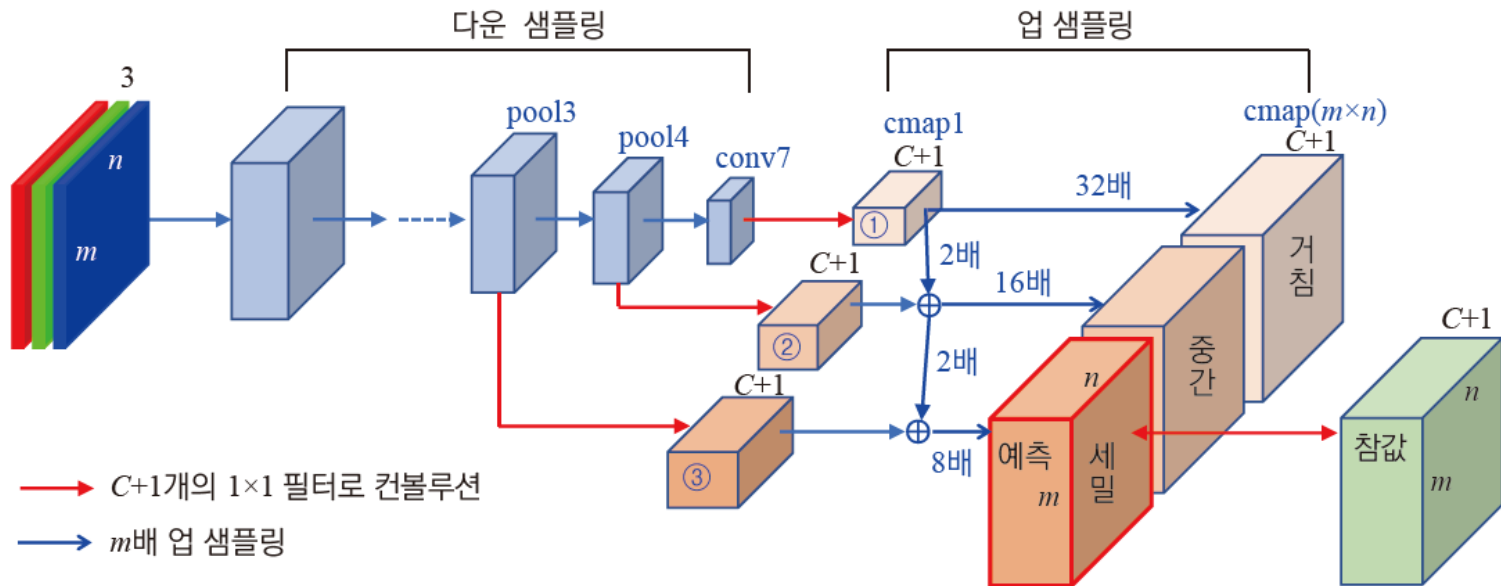


그림 9-31 FCN의 구조

9.4.2 의미 분할을 위한 FCN

■ [예시 9-3] FCN의 참값 텐서와 예측 텐서

[그림 9-29(a)]를 FCN에 맞추어 표현

0	1	2
1 1 1 1 1 1	0 0 0 0 0 0	0 0 0 0 0 0
1 0 0 0 1 1	0 0 0 0 0 0	0 1 1 1 0 0
0 0 0 0 0 1	0 0 0 0 1 0	1 1 1 1 0 0
0 0 0 0 0 1	0 0 0 1 1 0	1 1 1 0 0 0
1 0 0 0 0 1	0 0 0 1 1 0	0 1 1 0 0 0
1 0 0 0 0 1	0 0 0 1 1 0	0 1 1 0 0 0

(a) 참값 텐서

0 배경	1 부류1	2 부류2
.8 .6 .7 .9 .9 .8	.1 .1 .1 .1 .1 0	.1 .3 .2 0 0 .2
1 .1 .1 0 0 .8	0 0 .1 0 .1 .1	0 .9 .8 1 .9 .1
1 .1 0 0 0 .5	0 0 0 .9 .1 .3	0 .9 1 .1 .9 .2
.8 .1 0 0 .1 .7	0 0 0 1 .9 .2	.2 .9 1 0 0 .1
.9 .1 0 0 0 .9	.1 .1 0 1 1 0	0 .8 1 0 0 .1
1 1 .1 0 .1 .9	0 0 0 0 .9 0	0 0 .9 1 0 .1

(b) 예측 텐서

더하면 1이고
0번 맵이 최대이므로
배경으로 해석

그림 9-32 FCN의 참값 텐서와 예측 텐서

9.4.2 의미 분할을 위한 FCN

■ 다운 샘플링과 업 샘플링

- FCN은 다운 샘플링으로 맵 크기를 줄인 다음 업 샘플링으로 원래 크기 복원
- 업 샘플링을 어떻게 달성할 것인가?
 - 양선형 보간법([그림 3-22])을 채택하면 다운 샘플링은 학습으로 수행하고 업 샘플링은 고전 방법을 사용하는 꼴
 - FCN은 전치 컨볼루션으로 업 샘플링 달성

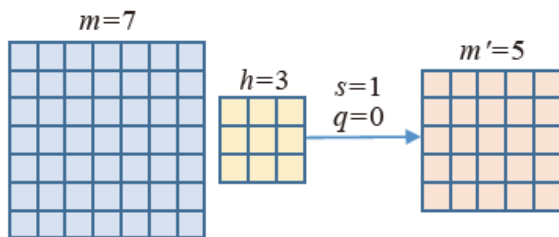
TIP 어떤 문헌은 전치 컨볼루션을 디컨볼루션(deconvolution)이라고 부르는데, 접두어 de는 떼어낸다는 뜻을 가지므로 적당한 용어가 아니다. 이 책에서는 디컨볼루션이라는 용어를 사용하지 않는다.

9.4.2 의미 분할을 위한 FCN

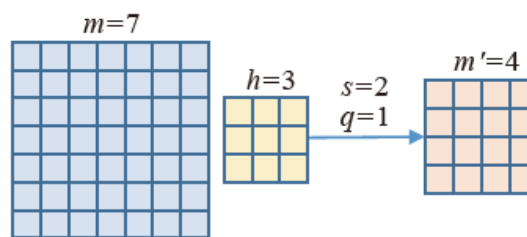
■ 컨볼루션에 따른 특징 맵의 크기 변화

- [그림 9-33]은 $m \times m$ 특징 맵을 $h \times h$ 필터로 컨볼루션하여 $m' \times m'$ 맵을 출력
- 보폭을 s , 덧대기 크기를 q 라 하면 식 (9.6)에 따라 m' 가 정해짐

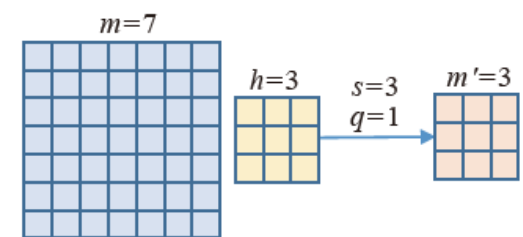
$$m' = \left\lfloor \frac{m + 2q - h}{s} \right\rfloor + 1 \quad (9.6)$$



(a) 보폭 1, 패딩 없음



(b) 보폭 2, 패딩 1줄



(c) 보폭 3, 패딩 1줄

그림 9-33 컨볼루션에 따른 특징 맵의 크기 변화

9.4.2 의미 분할을 위한 FCN

■ 전치 컨볼루션 transposed convolution

- $m' \times m'$ 특징 맵을 전치 컨볼루션하여 $m \times m$ 맵으로 업 샘플링

- 아래 세 단계

1. 행 사이에 $s'-1$ 개의 0 행을 삽입하고 열 사이에 $s'-1$ 개의 0 열을 삽입한다. 새로운 영상은 $(s(m'-1)+1) \times (s(m'-1)+1)$ 가 된다.
2. 새로운 영상에 $q'=h-q-1$ 만큼 0을 덧대기한다. 새로운 영상은 $(s(m'-1)+2h-2q-1) \times (s(m'-1)+2h-2q-1)$ 이 된다.
3. 새로운 영상에 보폭이 1인 컨볼루션을 적용한다.

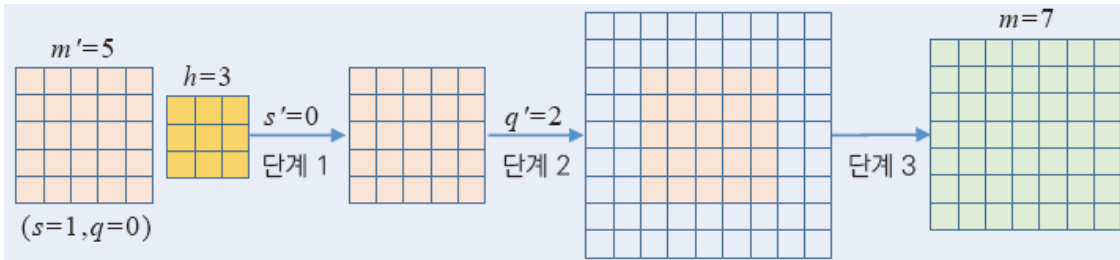
- 세 단계를 적용하면 식 (9.7)에 따라 m 이 정해짐

$$m = (m' - 1) \times s + h - 2q \quad (9.7)$$

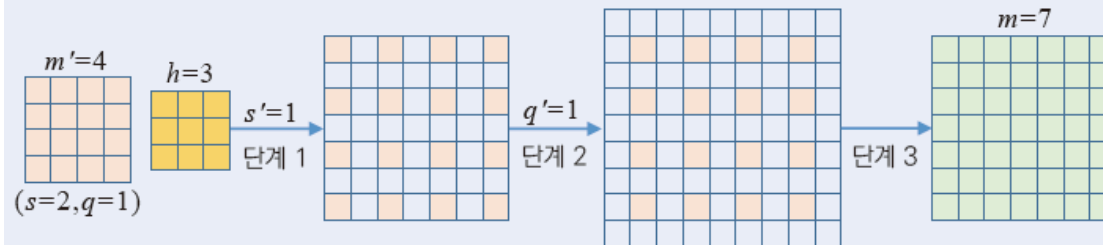
- 맵의 크기를 복원하지만, 값을 복원하는 것은 아님. 전치 컨볼루션을 위한 필터를 학습으로 알아내기 때문에 값을 복원할 필요 없음

9.4.2 의미 분할을 위한 FCN

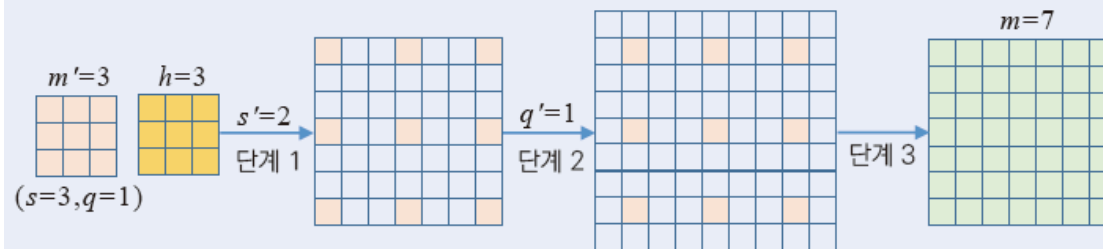
■ [예시 9-4] 전치 컨볼루션



(a) [그림 9-33(a)] 업 샘플링



(b) [그림 9-33(b)] 업 샘플링



(c) [그림 9-33(c)] 업 샘플링

그림 9-34 전치 컨볼루션에 의한 업 샘플링

[그림 9-33(a)]의 결과에
전치 컨볼루션 적용

[그림 9-33(b)]의 결과에
전치 컨볼루션 적용

[그림 9-33(c)]의 결과에
전치 컨볼루션 적용

9.4.2 의미 분할을 위한 FCN

■ FCN의 동작

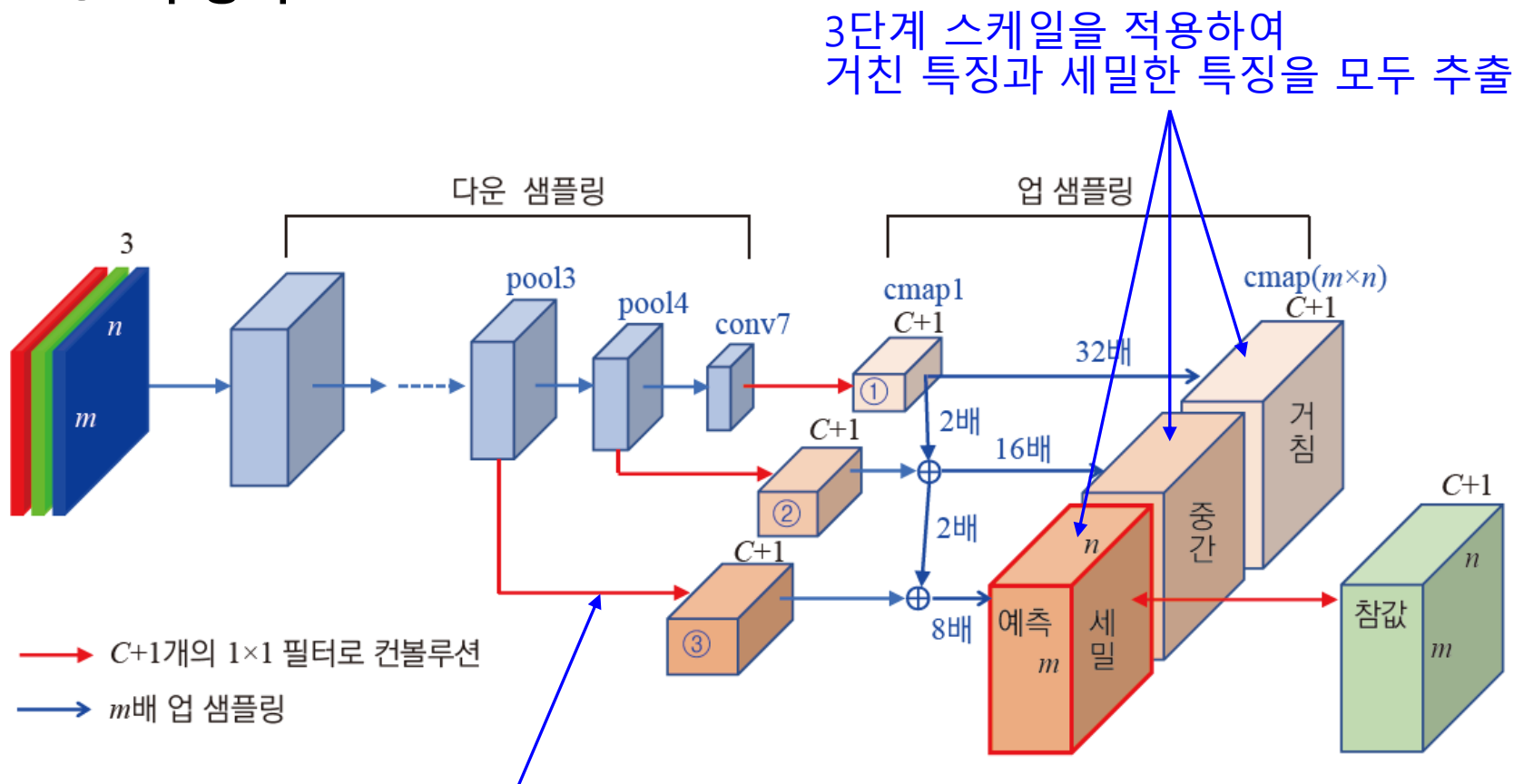


그림 9-31 FCN의 구조

C+1개의 1*1 필터를 적용하여
부류별로 분할 맵을 가진 텐서로 변환

9.4.3 FCN을 개선한 신경망: DeConvNet, U-net, DeepLabv3+

■ FCN의 성공 이후 다양한 변형 신경망 등장

■ DeConvNet

- 오토인코더와 FCN을 결합한 형태(인코더는 다운 샘플링, 디코더는 업 샘플링 담당)
- FCN에 비해 학습이 훨씬 세련되고 성능도 우월

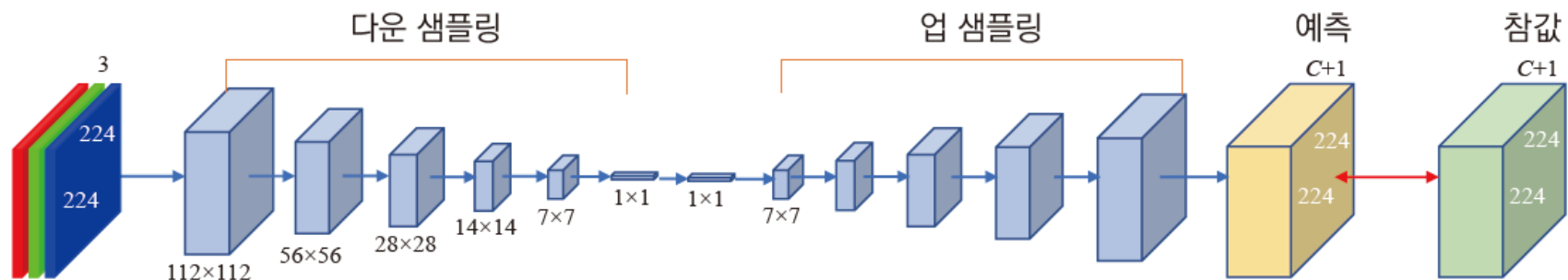


그림 9-35 DeConvNet의 구조

9.4.3 FCN을 개선한 신경망: DeConvNet, U-net, DeepLabv3+

■ U-net

- 의료 영상 분할을 목적으로 개발되어 지금은 자연영상에까지 널리 사용됨
- 다른 신경망처럼 다운 샘플링과 업 샘플링을 수행함
- 가장 특이한 점은 지름길 연결(특징 맵을 이어 붙이는 지름길 연결)

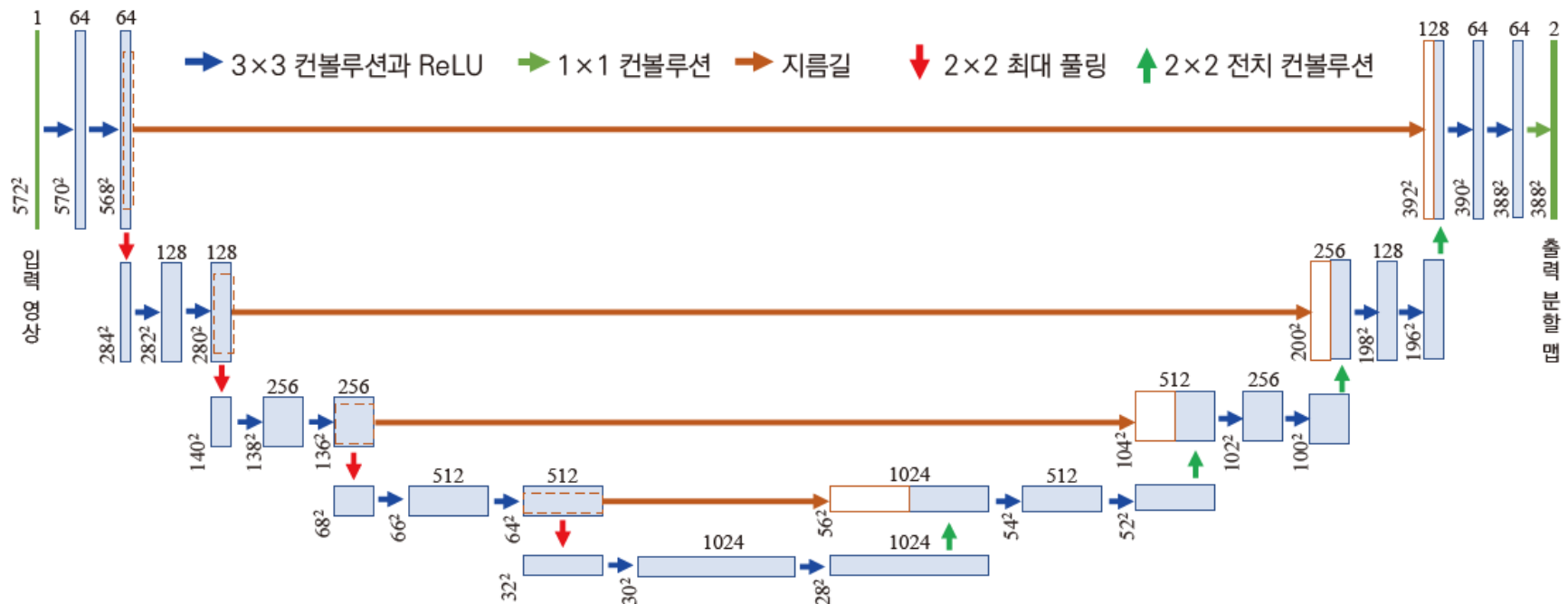
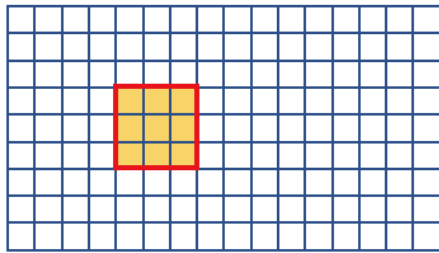


그림 9-36 U-net의 구조

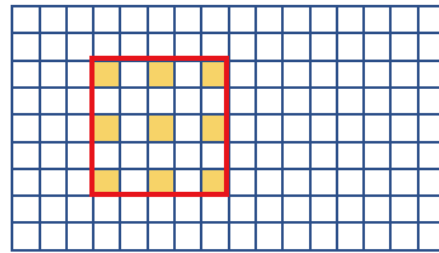
9.4.3 FCN을 개선한 신경망: DeConvNet, U-net, DeepLabv3+

■ DeepLabv3+

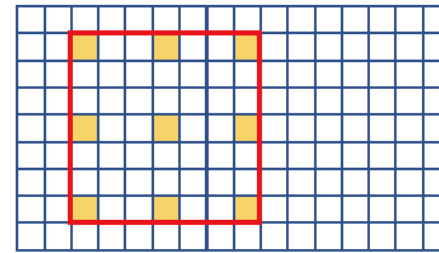
- 인코더와 디코더에 기반한 신경망은 영상을 상당히 작게 축소했다가 복원하기 때문에 상세 내용 잃을 가능성 높음
- DeepLabv3+는 팽창 컨볼루션을 이용하여 이런 단점을 누그러뜨림



(a) 팽창 계수 $r=1$

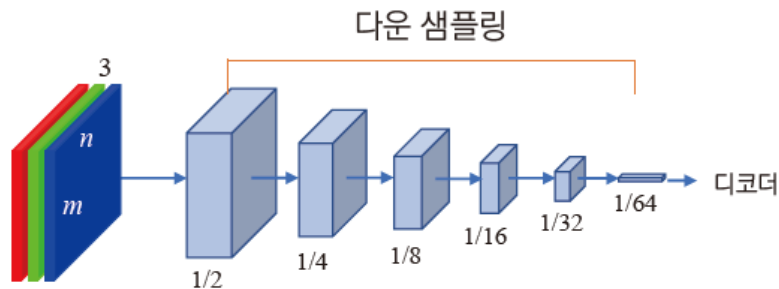


(b) 팽창 계수 $r=2$

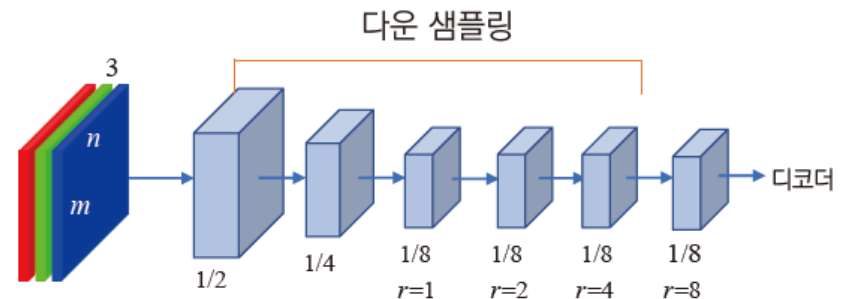


(c) 팽창 계수 $r=3$

그림 9-37 팽창 컨볼루션(3×3 필터)



(a) 팽창 컨볼루션을 적용하지 않음



(b) 팽창 컨볼루션을 적용함(DeepLabv3+)

그림 9-38 팽창 컨볼루션 적용 여부에 따른 신경망 구조

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

■ Oxford IIIT Pet 데이터셋 활용

- 개와 고양이 품종 37개 포함. 품종마다 200여장. 총 7,349장의 영상
- 영상마다 머리 검출을 위한 박스와 분할을 위한 레이블(여기서는 분할 레이블 사용)
- 절차에 따라 폴더에 설치해야 실습 가능

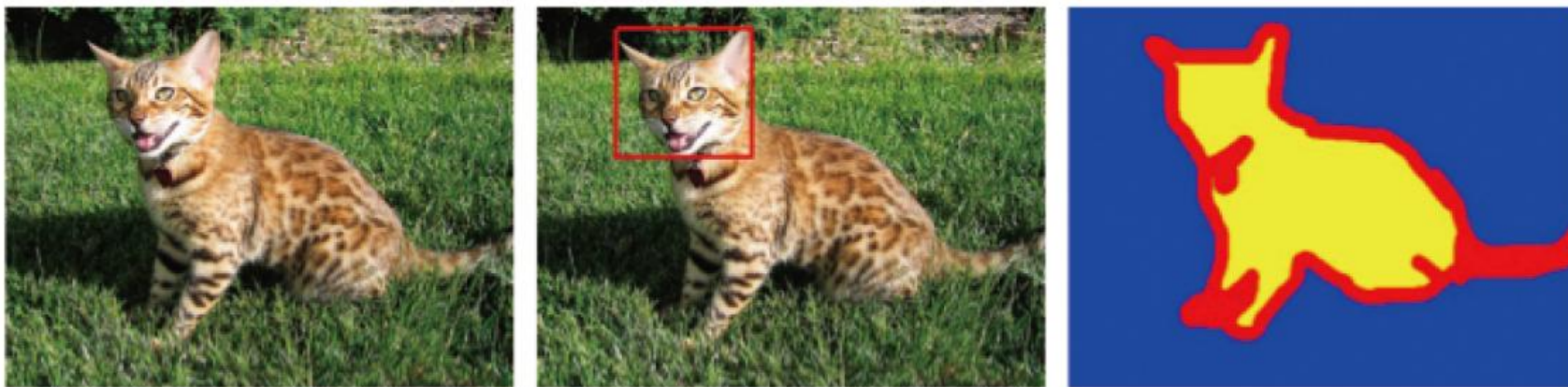


그림 9-39 Oxford IIIT Pet 데이터셋의 예제 영상과 레이블

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

프로그램 9-4

Oxford pets 데이터셋으로 U-net 학습하기

```
01  from tensorflow import keras
02  import numpy as np
03  from tensorflow.keras.preprocessing.image import load_img
04  from tensorflow.keras import layers
05  import os
06  import random
07  import cv2 as cv
08
09  input_dir='./datasets/oxford_pets/images/images/'
10  target_dir='./datasets/oxford_pets/annotations/annotations/trimaps/'
11  img_siz=(160,160)           # 모델에 입력되는 영상 크기
12  n_class=3                   # 분할 레이블 (1:물체, 2:배경, 3:경계)
13  batch_siz=32                # 미니 배치 크기
14
15  img_paths=sorted([os.path.join(input_dir,f) for f in os.listdir(input_
    dir) if f.endswith('.jpg')])
16  label_paths=sorted([os.path.join(target_dir,f) for f in os.listdir(target_
    dir) if f.endswith('.png') and not f.startswith('.')])
17
```

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

```
18 class OxfordPets(keras.utils.Sequence): ← 데이터셋을 읽어오는 클래스
19     def __init__(self, batch_size, img_size, img_paths, label_paths):
20         self.batch_size=batch_size
21         self.img_size=img_size
22         self.img_paths=img_paths
23         self.label_paths=label_paths
24
25     def __len__(self):
26         return len(self.label_paths)//self.batch_size
27
28     def __getitem__(self, idx):
29         i=idx*self.batch_size
30         batch_img_paths=self.img_paths[i:i+self.batch_size]
31         batch_label_paths=self.label_paths[i:i+self.batch_size]
32         x=np.zeros((self.batch_size,)+self.img_size+(3,), dtype="float32")
33         for j, path in enumerate(batch_img_paths):
34             img=load_img(path, target_size=self.img_size)
35             x[j]=img
36         y=np.zeros((self.batch_size,)+self.img_size+(1,), dtype="uint8")
37         for j, path in enumerate(batch_label_paths):
38             img=load_img(path, target_size=self.img_size, color_mode="grayscale")
39             y[j]=np.expand_dims(img, 2)
40             y[j]=1 # 부류 번호를 1,2,3에서 0,1,2로 변환
41         return x,y
42
```

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

```
43 def make_model(img_size,num_classes): ← U-net 모델을 만드는 클래스
44     inputs=keras.Input(shape=img_size+(3,))
45
46     # U-net의 다운 샘플링(축소 경로)
47     x=layers.Conv2D(32,3,strides=2,padding='same')(inputs)
48     x=layers.BatchNormalization()(x)
49     x=layers.Activation('relu')(x)
50     previous_block_activation=x      # 지름길 연결을 위해
51
52     for filters in [64,128,256]:
53         x=layers.Activation('relu')(x)
54         x=layers.SeparableConv2D(filters,3,padding='same')(x)
55         x=layers.BatchNormalization()(x)
56         x=layers.Activation('relu')(x)
57         x=layers.SeparableConv2D(filters,3,padding='same')(x)
58         x=layers.BatchNormalization()(x)
59         x=layers.MaxPooling2D(3,strides=2,padding='same')(x)
60         residual=layers.Conv2D(filters,1,strides=2,padding='same')(previous_
            block_activation)
61         x=layers.add([x,residual])      # 지름길 연결
62         previous_block_activation=x      # 지름길 연결을 위해
```


9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

```
63
64     # U-net의 업 샘플링(확대 경로)
65     for filters in [256, 128, 64, 32]:
66         x=layers.Activation('relu')(x)
67         x=layers.Conv2DTranspose(filters,3,padding='same')(x)
68         x=layers.BatchNormalization()(x)
69         x=layers.Activation('relu')(x)
70         x=layers.Conv2DTranspose(filters,3,padding='same')(x)
71         x=layers.BatchNormalization()(x)
72         x=layers.UpSampling2D(2)(x)
73         residual=layers.UpSampling2D(2)(previous_block_activation)
74         residual=layers.Conv2D(filters,1,padding='same')(residual)
75         x=layers.add([x,residual])           # 지름길 연결
76         previous_block_activation=x         # 지름길 연결을 위해
77
78     outputs=layers.Conv2D(num_classes,3,activation='softmax',padding='same')(x)
79     model=keras.Model(inputs, outputs)      # 모델 생성
80     return model
81
```

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

```
82 model=make_model(img_siz,n_class)           # 모델 생성
83
84 random.Random(1).shuffle(img_paths)
85 random.Random(1).shuffle(label_paths)
86 test_samples=int(len(img_paths)*0.1)         # 10%를 테스트 집합으로 사용
87 train_img_paths=img_paths[:-test_samples]
88 train_label_paths=label_paths[:-test_samples]
89 test_img_paths=img_paths[-test_samples:]
90 test_label_paths=label_paths[-test_samples:]
91
92 train_gen=OxfordPets(batch_siz,img_siz,train_img_paths,train_label_paths)
                                     # 훈련 집합
93 test_gen=OxfordPets(batch_siz,img_siz,test_img_paths,test_label_paths)
                                     # 검증 집합
94
95 model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
96               metrics=['accuracy'])
97 cb=[keras.callbacks.ModelCheckpoint('oxford_seg.h5',save_best_only=True)]
                                     # 학습 결과 자동 저장
98
99 model.fit(train_gen,epochs=30,validation_data=test_gen,callbacks=cb)
100
101 preds=model.predict(test_gen)         # 예측
102
103 cv.imshow('Sample image',cv.imread(test_img_paths[0])) # 0번 영상 디스플레이
104 cv.imshow('Segmentation label',cv.imread(test_label_paths[0])*64)
105 cv.imshow('Segmentation prediction',preds[0])         # 0번 영상 예측 결과 디스플레이
106
107 cv.waitKey()
108 cv.destroyAllWindows()
```

체크포인트 기능을 이용하여
학습 도중에 모델을 저장

학습 도중에 발생한
가장 높은 성능의 모델만 기록

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

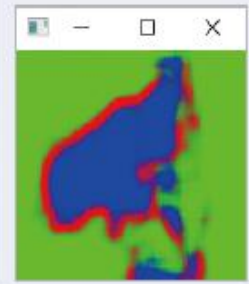
입력 영상



참값



예측값



9.5 사람 인식

■ 사람 인식은 가장 많은 관심과 연구 진행

- 인증, 보안, 고객 분석, 스포츠 등 응용이 무궁무진
- 스마트 기기 로그인 등 상용화 시스템 등장

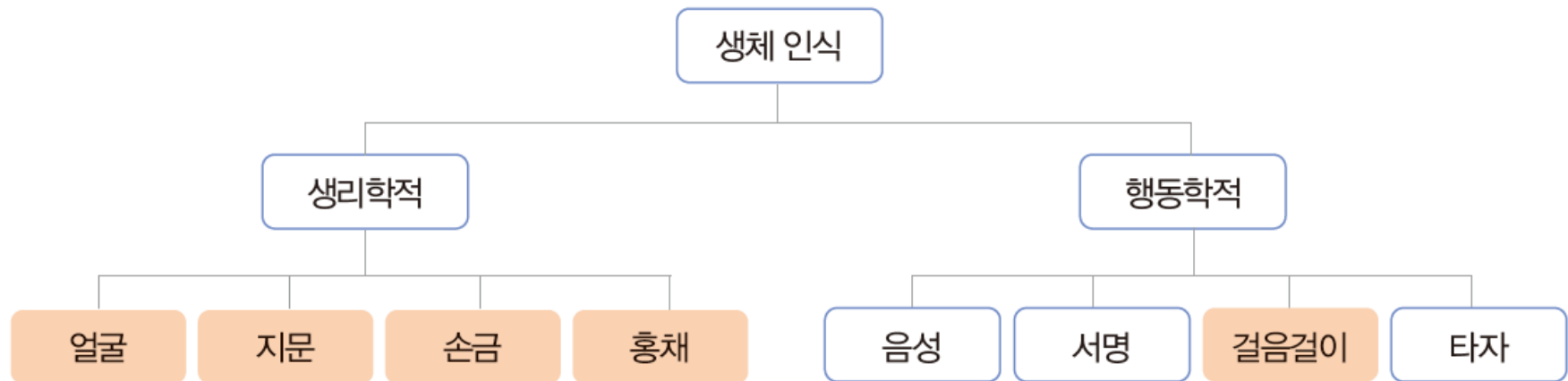


그림 9-40 생체 인식(주황색 표시는 컴퓨터 비전 기술을 사용해야 하는 특성)

9.5 사람 인식

■ 얼굴 인식은 사람 인식 중에서 가장 많이 연구된 분야

- 고전 방법에서 가장 성공적인 고유 얼굴_{eigen face} 기법
 - 얼굴 영상에 주성분 분석을 적용하여 차원을 줄인 다음 매칭 알고리즘으로 인식
 - 당시에는 획기적 진전
 - 정면 얼굴에 적용할 수 있는 한계(LFW 데이터셋에서 정확률 95%, 그 이상 돌파 못함)
- 컨볼루션 신경망 시대
 - 2014년 AlexNet을 백본으로 사용한 DeepFace는 LFW에서 97.35% 달성하여 사람 수준에 근접
 - 이후 VGGNet을 백본으로 사용한 VGGFace, GoogLeNet을 사용한 FaceNet, Resnet을 사용한 SphereFace 등이 99.8% 달성

9.5 사람 인식

■ 얼굴 확인_{face verification}과 얼굴 식별_{face identification}

- 얼굴 확인은 두 장의 얼굴이 동일인인지 확인하는 문제(예, 공항 여권 검사)
- 얼굴 식별은 입력 영상을 등록된 영상과 매칭하여 누구인지 알아내는 문제

■ 얼굴 인식은 미세 분류 문제의 일종

- 손실 함수를 잘 설계하여 성능 향상 모색
- 같은 부류 영상은 유사도 높게 유지하고 다른 부류는 낮게 유지하는 손실 함수
 - Contrastive 손실 함수, triplet 손실 함수, center 손실 함수 등이 개발됨

9.5 사람 인식

■ 현대적인 얼굴 인식은 제약을 두지 않는 상황에서 인식

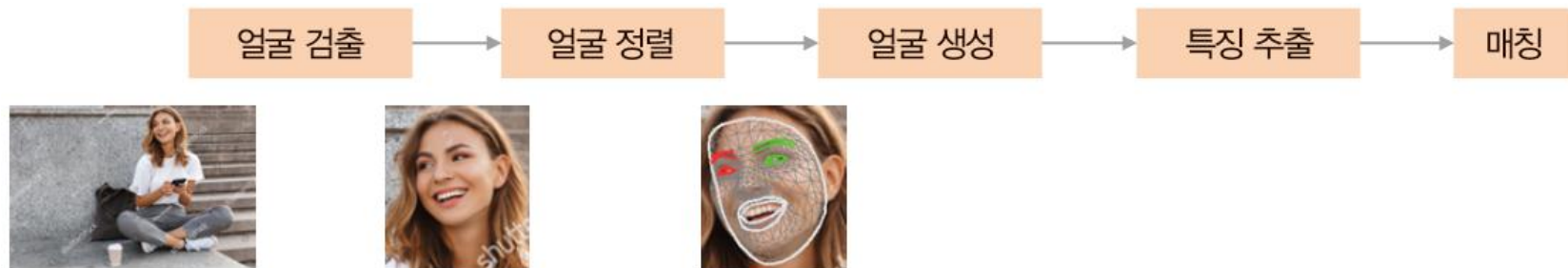
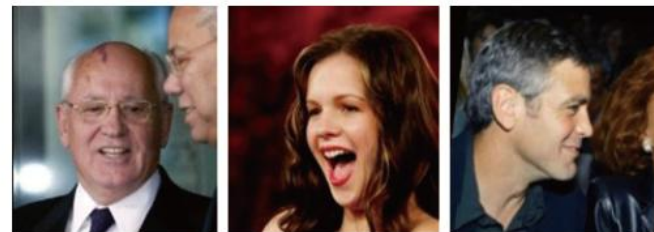


그림 9-42 현대적인 얼굴 인식의 처리 과정

■ 얼굴 데이터셋



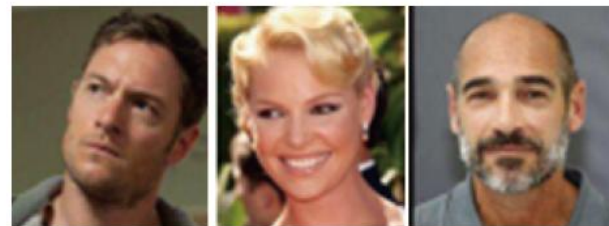
(a) Yale face



(b) LFW



(c) MegaFace



(d) VGG face

그림 9-41 얼굴 데이터셋

9.5 사람 인식

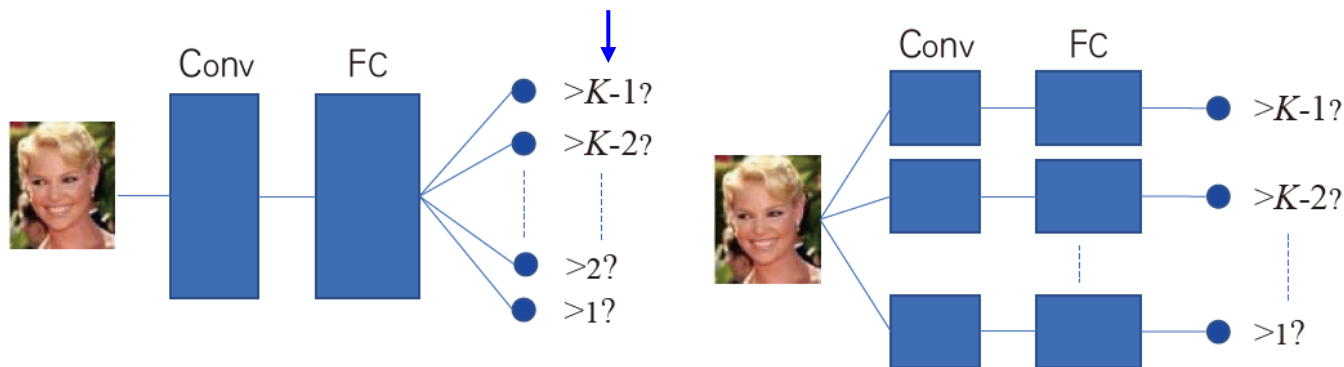
■ 성별과 나이 추정 데이터셋

- MORPH II: 16~77세인 얼굴 영상 55,134장. 인종과 성별 레이블도 있음
- IMDB-WIKI: 연예인의 성별과 나이를 레이블링한 50만 장 이상의 영상
- AFAD_{Asian Face Age Dataset}: 성별과 나이 레이블을 가진 아시아인 얼굴 영상 16만장
- UTKface: 0~116세에 대해 나이, 성별, 인종을 레이블링한 23,708장의 얼굴 영상

9.5 사람 인식

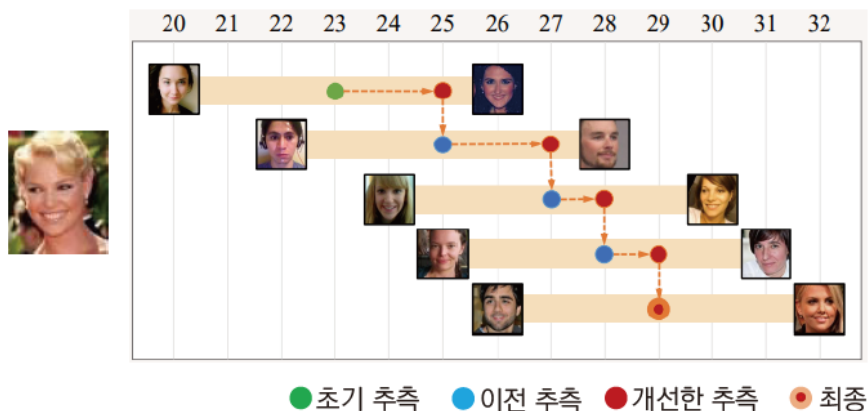
■ 나이 추정은 주로 순서형 회귀 ordinal regression 문제로 취급

K-1개의 출력층 각각은 나이가 k 이상인가라는 질문을 책임짐
예라고 답한 출력층의 개수를 예측 나이로 출력



(a) 다중 출력층 사용[Niu2016]

(b) 여러 신경망 사용[Chen2017b]



(c) 이동 윈도우 회귀[Shin2022]