
You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†

University of Washington*, Allen Institute for AI†, Facebook AI Research¶

YOLOv5, YOLOv8 AND YOLOv10: THE GO-TO DETECTORS FOR REAL-TIME VISION

Muhammad Hussain

Department of Computer Science, Huddersfield University, Queensgate, Huddersfield HD1 3DH, UK;

*Correspondence: M.Hussain@hud.ac.uk;

YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi

University of Washington

YOLO Paper Review

You Only Look Once: Unified, Real-time Object Detection - J. Redmon et al.

YOLOv3: An Incremental Improvement - J. Redmon, A. Farhadi

YOLOv5, YOLOv5 and YOLOv10: The Go-To Detectors For Real-Time Vision - M. Hussain

팀장 : 장우진

팀원 : 김선준 / 장재우 / 강희준 / 박은수

목차

① YOLO 이전의 Detector

- RCNN 계열

② YOLO 핵심 개념

- mAP
- NMS
- Anchor vs Anchor-free

③ Architecture

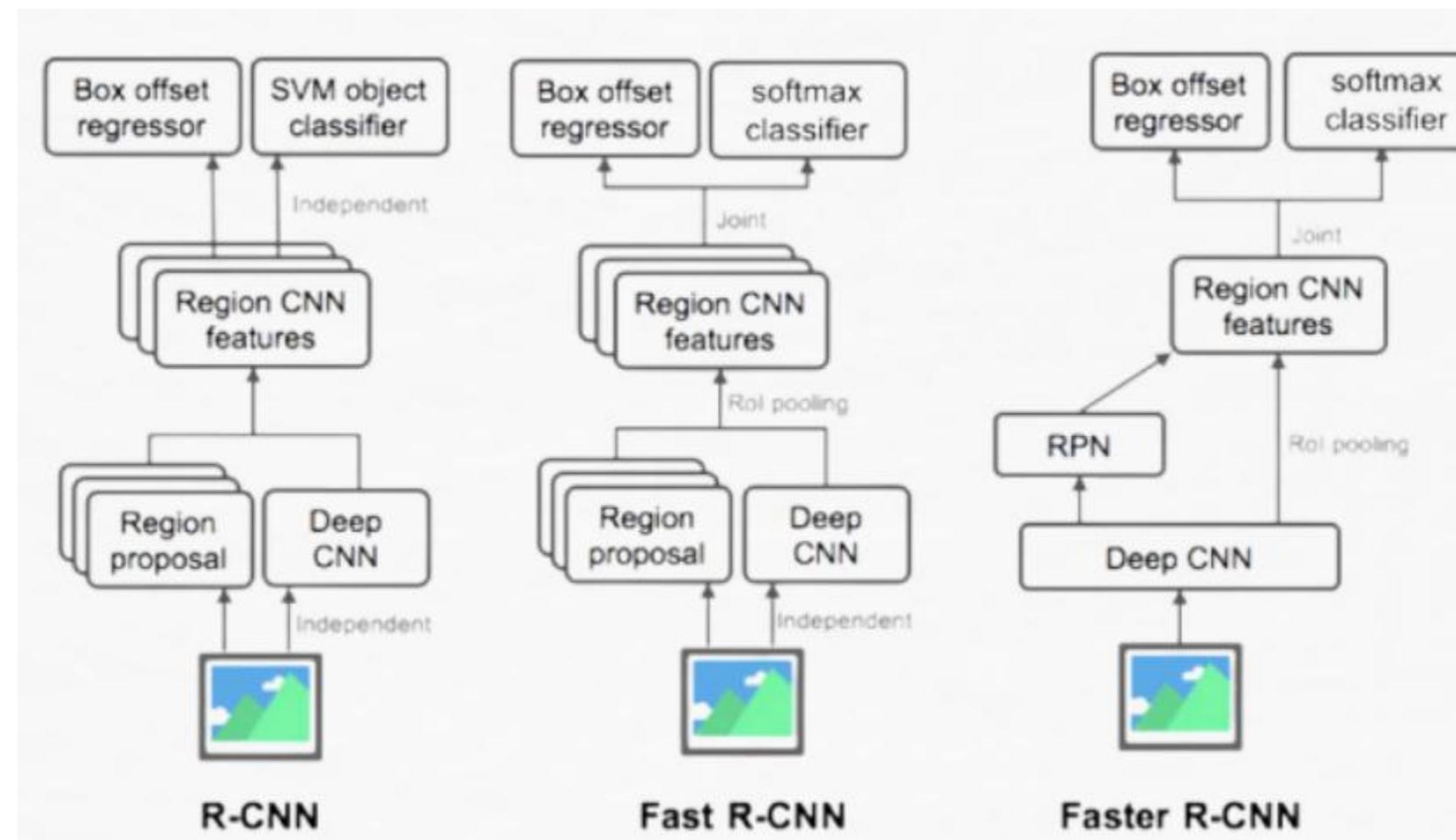
- YOLOv1 Architecture & Loss Function
- YOLOv3
- YOLOv5

④ YOLOv11 구현

- 재학습 코드
- 실제 결과 제시 및 분석

R-CNN 계열

YOLO 이전의 Detector



1. 제안 목적 및 개요

이미지 전체에서 높은 정확도(mAP)로 객체를 Detecting 하기위해 고안 Region Proposal을 통해 정밀한 객체 탐지 방식을 제안하였고 Mask-RCNN등으로 발전하면서 정밀한 Segmentation 까지 가능함

2. Region Proposal

Region Proposal은 기존의 CNN 방식이 Conv Layer를 통과하면서 각 객체의 위치를 정밀하게 탐지하지 못한다는 한계를 극복하기 위해 제안
ROI는 CNN 모델과 분리되어 따로 연산되며 여러 객체의 세밀한 위치를 찾을
수 있도록 Feature Map에서 합친 후 BBox와 Cls를 수행

3. 2 Stage Detector

전통 기법 대비: 딥 피처+제안영역(RP/RPN)으로 표현력과 정밀도가 월등하여
배경·조명·자세 변화, 소형·부분가림 상황에서 박스 정합(IoU)과 리콜이 높음
1 Stage 대비: ROI별 정제(분류·회귀) 구조로 작은/밀집/중첩 객체에서 정확도
우위, 샘플링·하드네거티브 컨트롤로 오탐 제어와 캘리브레이션이 유리

R-CNN 계열

YOLO 이전의 Detector

Region Proposal

이미지에서 수천 개 후보영역(Roi)을 생성하여 CNN과 함께 처리
최종 BBox와 Cls 성능은 IoU/NMS로 중복을 억제하여 고도화

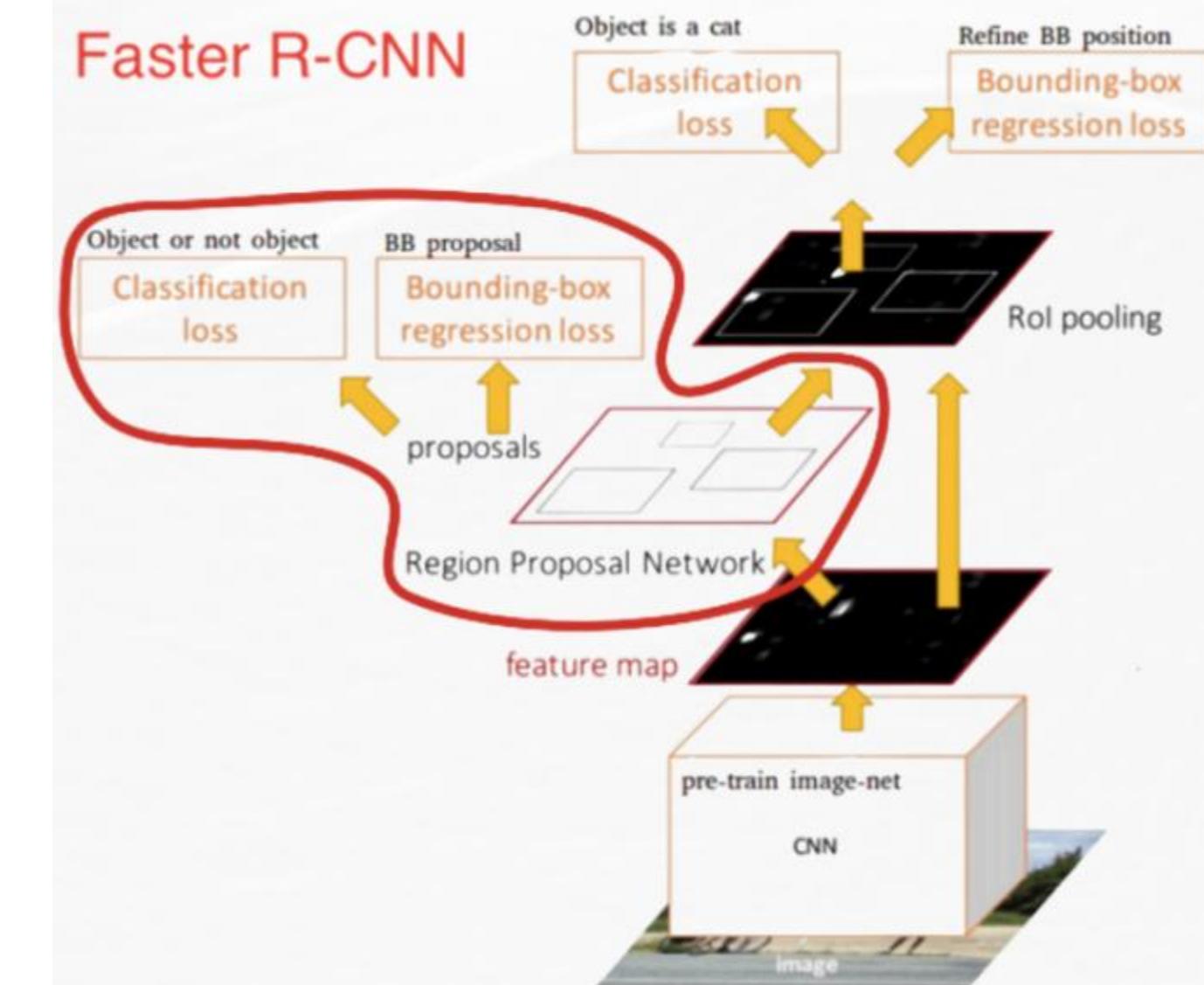
왜 Region Proposal을 사용할까?

Region proposal은 전체 이미지의 막대한 위치·크기 조합을 전부 보지 않고
물체일 가능성이 높은 후보 영역만 뽑아, 계산을 크게 줄이면서 네트워크가
배경보다 전경(물체) 학습에 집중할 수 있게 함

이렇게 추린 Roi를 정교하게 분류·박스회귀(및 RoIAlign 등)로 다듬으면, 특히
작은·겹친·복잡한 장면에서 정확도(정밀도·재현율)를 끌어올릴 수 있음

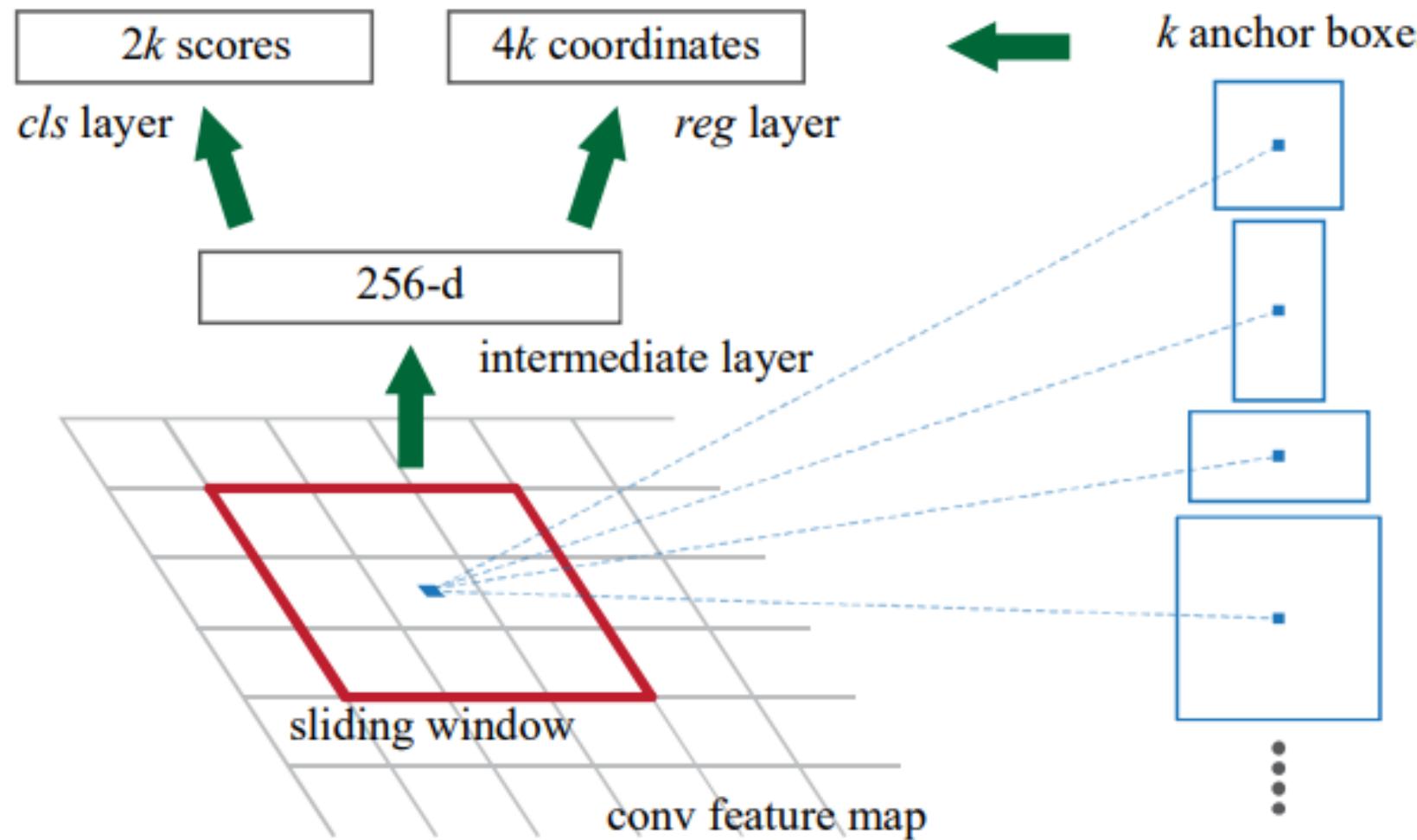
목적: 연산 시간을 획기적으로 줄일 수 있음

Faster R-CNN



R-CNN 계열

YOLO 이전의 Detector



한계점

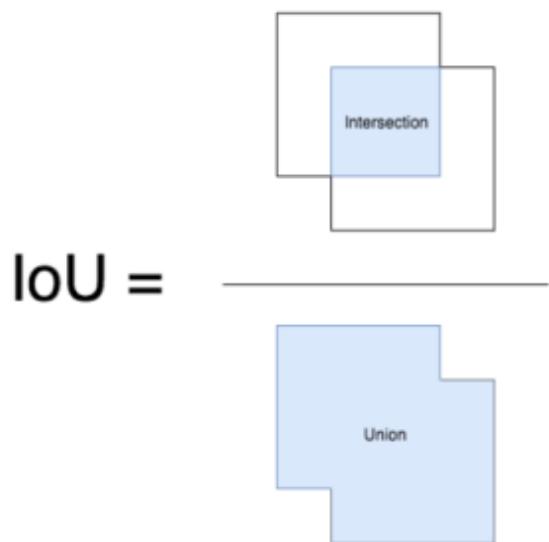
- 1) Region Proposal로 인한 연산량 증가와 그로 인한 속도 저하
- 2) ROI Box의 개수가 각 이미지의 해상도와 객체 수에 따라 가변적인 이유로 연산 처리 속도와 효율의 변동성이 큼
- 3) 멀티 스케일 이미지와 고해상도 이미지를 입력 받으면 ROI가 급증하여 처리 속도와 레이턴시가 흔들리는 문제가 발생

지금은 왜 안 쓰나?

- 1) 속도와 지연: RPN → ROIAlign → Head 단계라 YOLO(단일 패스) 대비 실시간성 불리
- 2) 배치 효율/메모리: ROI의 다양성에 따라 GPU 벡터화 불가, 스루풋·메모리 효율이 저하
- 3) 배포·최적화 난이도: ROIAlign 같은 동적 연산으로 인한 TensorRT/양자화/모바일 포팅이 YOLO보다 까다로움
- 4) 스케일 비용: 입력 해상도·다중스케일에서 제안 수가 급증해 레이턴시 및 비용이 커지고, 소형·밀집 장면에서 처리량이 불안정해짐

mAP

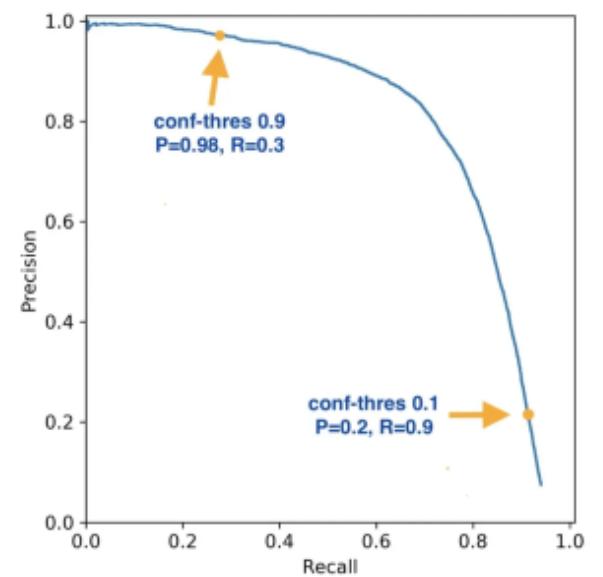
평가지표



$$\begin{aligned}\text{IoU}(A, B) &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|A \cap B|}{|A| + |B| - |A \cap B|}\end{aligned}$$

IoU (Intersection of Union)

- GT Box와 예측된 해당 클래스의 BBox의 겹침 정도를 수치화함
- 예측과 GT의 IoU가 임계치(파라미터) 이상이면 TP, 아니면 FP로 계산(AP/mAP 산출에 사용)
- 장점: 좌표 L2보다 평가 지표보다 정밀
<변형>
- DIoU: 중심 거리 패널티 포함 → 중심 정렬 유도
- CIoU: DIoU + 종횡비 일치 항 추가(박스 모양까지 맞춤)



$$\text{AP} = \int_0^1 P_{\text{interp}}(r) dr$$

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^C \text{AP}_c$$

mAP(mean Average Precision)

- AP(average precision) : AP@ τ 는 IoU 임계치 τ (예: 0.5)를 고정해 놓고, 점수 임계치(score threshold)를 전 범위로 스윕하여 얻은 PR 곡선 아래 면적(적분 면적)
- mAP: 모든 클래스 AP의 평균(macro average).
ex) τ 가 1에 가까우면, IoU 가 τ 보다 높은 BBox들의 클래스 점수만 평균 각 클래스 평균 점수를 다시 평균

NMS 원리

NMS 후처리

특징

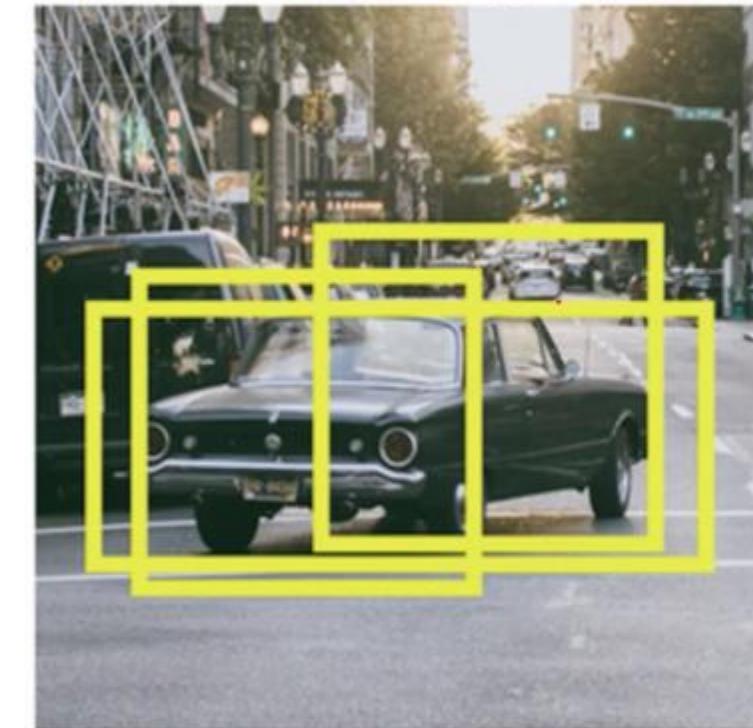
1. 개념

- 예측된 바운딩 박스를 두 가지 Confidence Score와 IoU(Intersection over Union) 임계값을 기반으로 의사 결정을 내리는 방식

2. 적용(Greedy NMS)

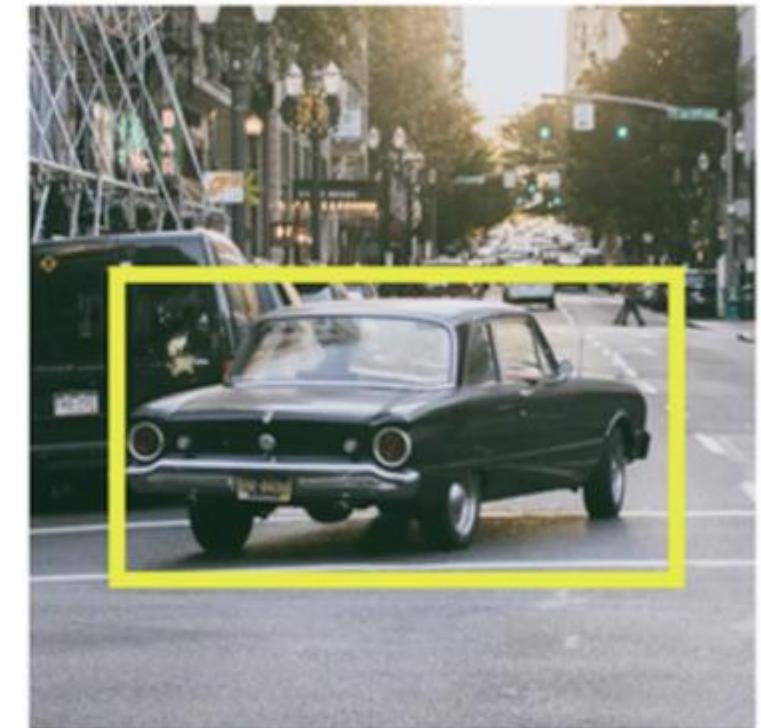
- 한 이미지에서, (보통 클래스별) 박스들을 conf score 내림차순으로 정렬
 $\text{conf} = \text{objectness} \times \text{class_prob}$
- 가장 높은 score의 박스 b 를 선택 및 보관.
- 나머지 박스들 중 선택된 b 와 $\text{IoU} > \theta$ (예: 0.5)인 것들을 제거(suppress).
- 남은 박스에 대해 2~3을 반복.
- 복잡도: 최악 $O(n^2)$ (정렬 $O(n \log n)$ + IoU 다대다 비교). 보통 Top-K 사전 필터링(예: 상위 1k~3k개)로 실용 속도를 확보

Before non-max suppression

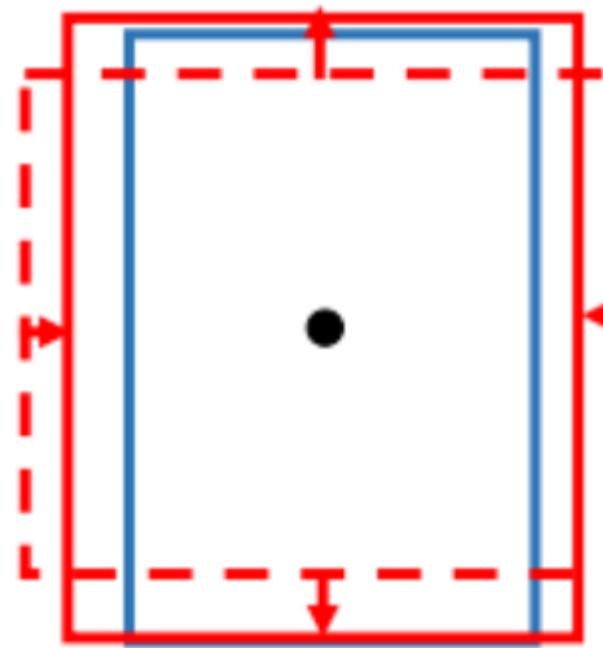


Non-Max
Suppression
→

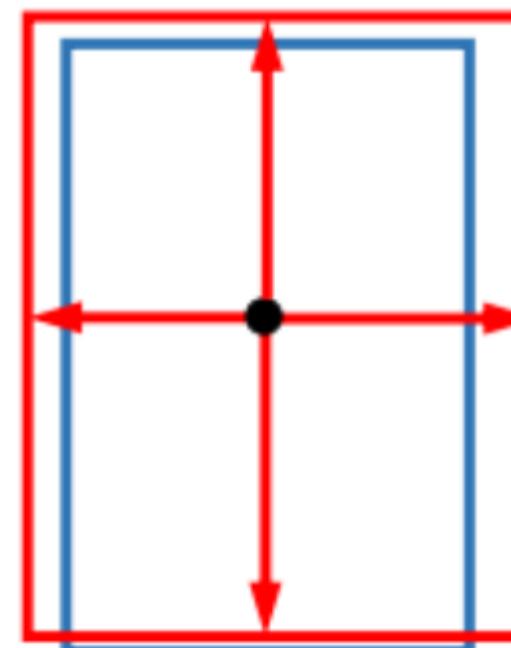
After non-max suppression



Anchor-based / Anchor-free



(a) Anchor-based approaches



(b) Anchor-free approaches

Anchor-based

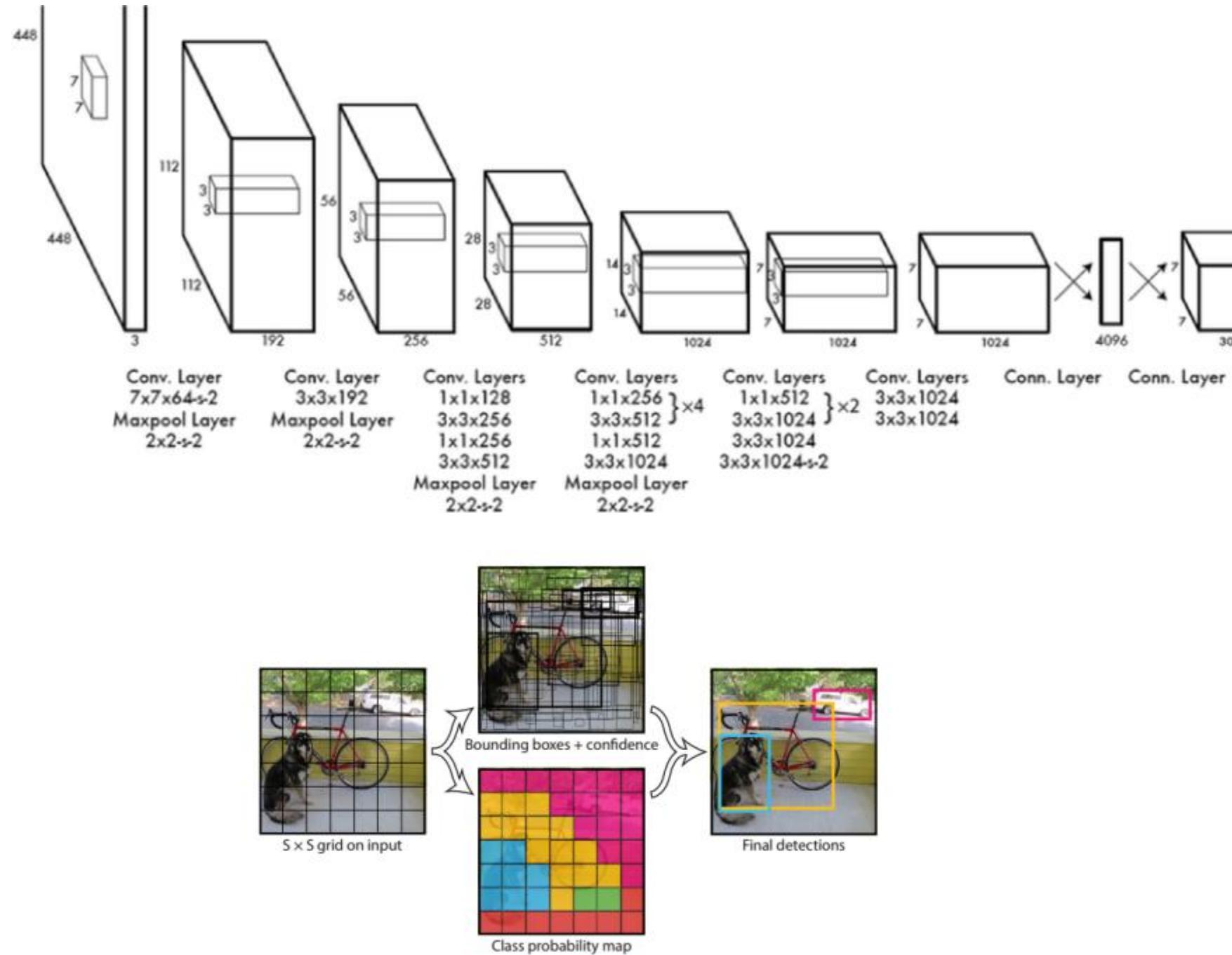
- 각 피처맵 위치(셀)마다 미리 정해 둔 여러 개의 앵커 박스(w, h 템플릿)를 깔아두고, 모델이 앵커 대비 오프셋(tx, ty, tw, th)과 클래스/오브젝트 점수를 예측
- 예측 후 NMS로 중복을 정리하죠
- YOLOv2 의 내장 함수에서부터 k-means dimension clustering 를 제공 <한계>
- 최초에 설정된 Anchor 크기에 BBox 크기가 고정

Anchor-free

- Anchor 템플릿 없이 각 위치가 객체의 중심부터 4개의 변까지의 거리(4방향) 등을 직접 회귀
- BBox 크기 예측도 anchor가 아닌 중심/영역 기반
- Ultralytics YOLOv8에서 공식적으로 anchor-free split head를 채택
- 같은 계열의 최신 모델들도 anchor-free model

YOLOv1 Architecture

모델 구조



1. Backbone

- 24 Conv Layer + 2 Fully Connected Layer 의 Customed CNN (GoogLeNet 느낌의 $1 \times 1 / 3 \times 3$ 조합, 잔차·FPN 없음)
- Conv($k=1 \times 1$) (Point-Wise Convolution):
 - 원리: 공간(H,W)은 건드리지 않고, 채널(C)만 섞는 연산
 - 목적: 1) 3×3 전에 채널을 줄여 FLOPs/파라미터를 크게 절약
 - 2) 스킵 연결 차원 맞추기
 - 3) 가볍고 빠름

2. Neck

없음

3. Head

그리드 셀 단위 Flatten - FC - reshape
Output shape: (S, S, B * 5 + C)

* 원 논문 (7, 7, 30)

YOLOv1 Architecture

모델 구조

특징

1 이미지를 $S * S$ 의 그리드로 분할

2. Backbone 과정 이후 Neck 없이 바로 Head로 연결됨

3. Head 구조

1) backbone output shape : (7, 7, 1024)

2) Flatten하여 Fully Connected -> ($S * S * (B * 5 + C)$,)

3) ($S, S, B * 5 + C$)로 Reshape (원 논문 Pascal Voc B=2)

- 각 그리드 셀 구조

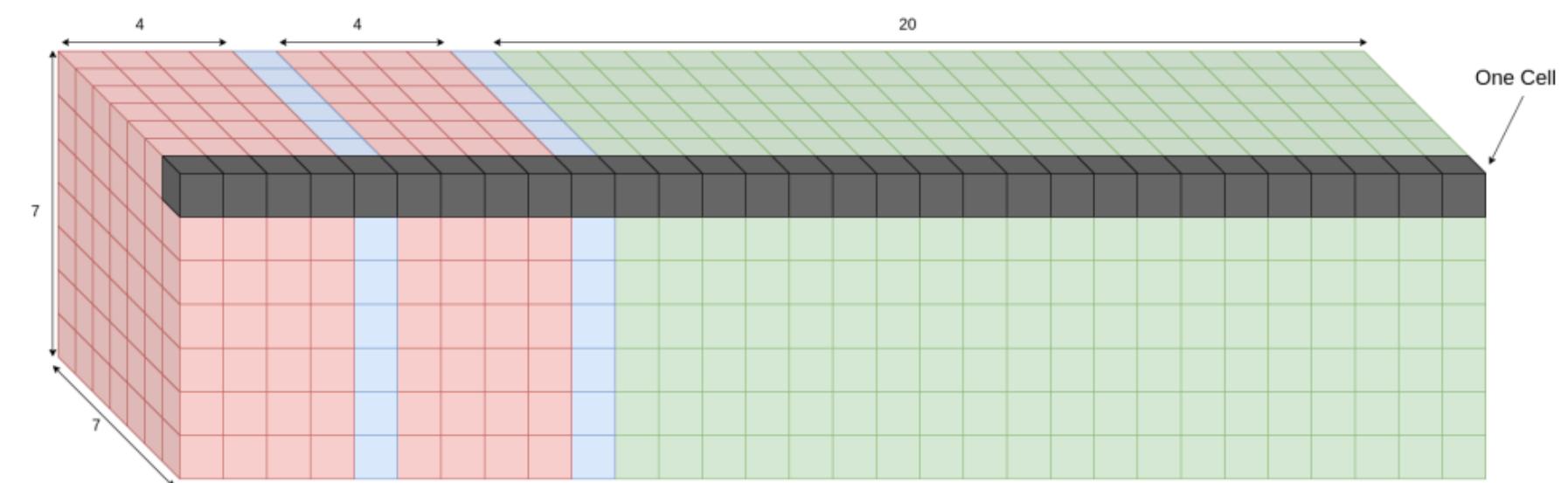
$(x, y, w, h, c) - (x, y, w, h, c) - (cp, cp, cp, \dots cp, cp)$

x, y = 중심 기준 좌표 / w, h = 객체 크기 / c = 객체 가능성(0, 1)

cp = 각 클래스마다 객체 가능성 (원 논문 Pascal Voc C= 20)

4) Bounding Box 중 NMS로 중복 박스 제거

5) Loss 계산



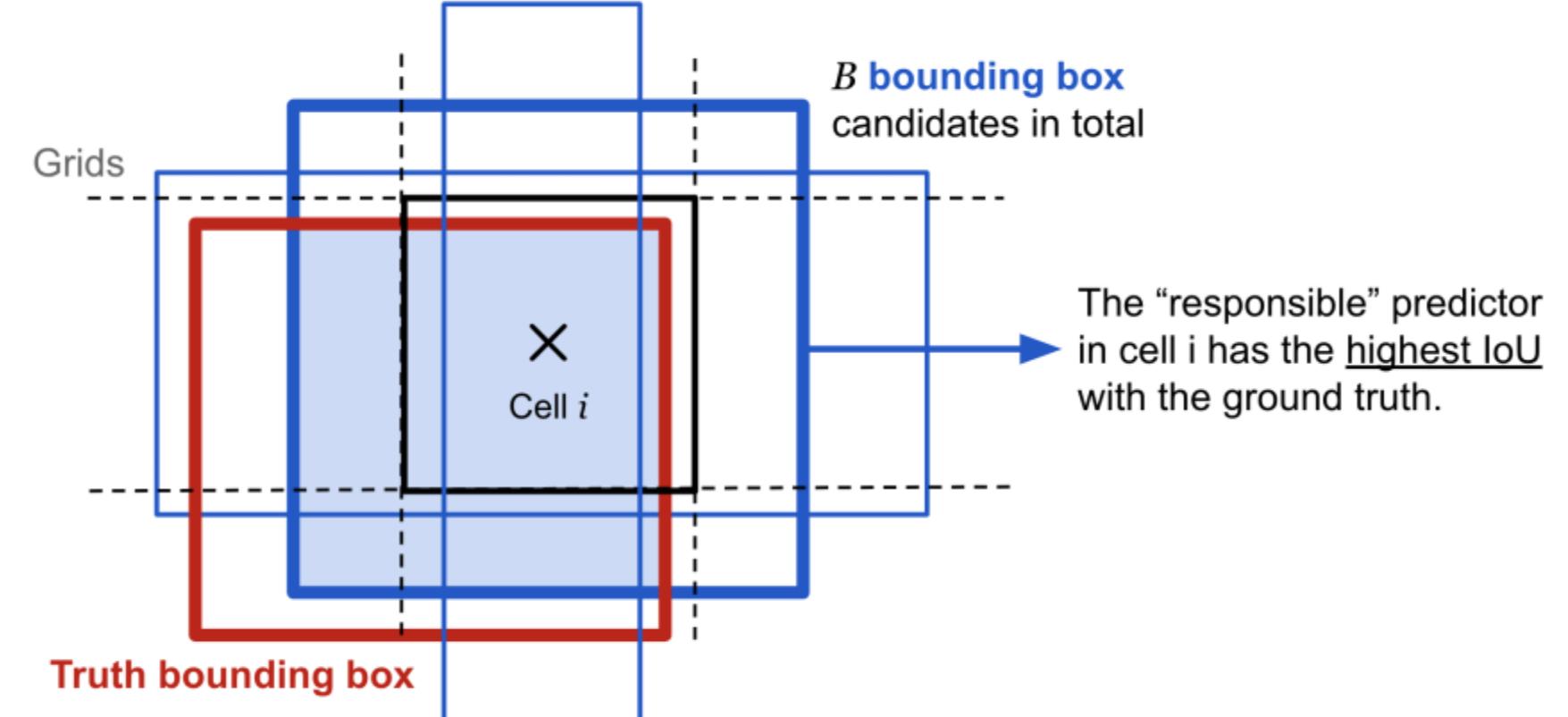
YOLOv1 Architecture

모델 구조

특징

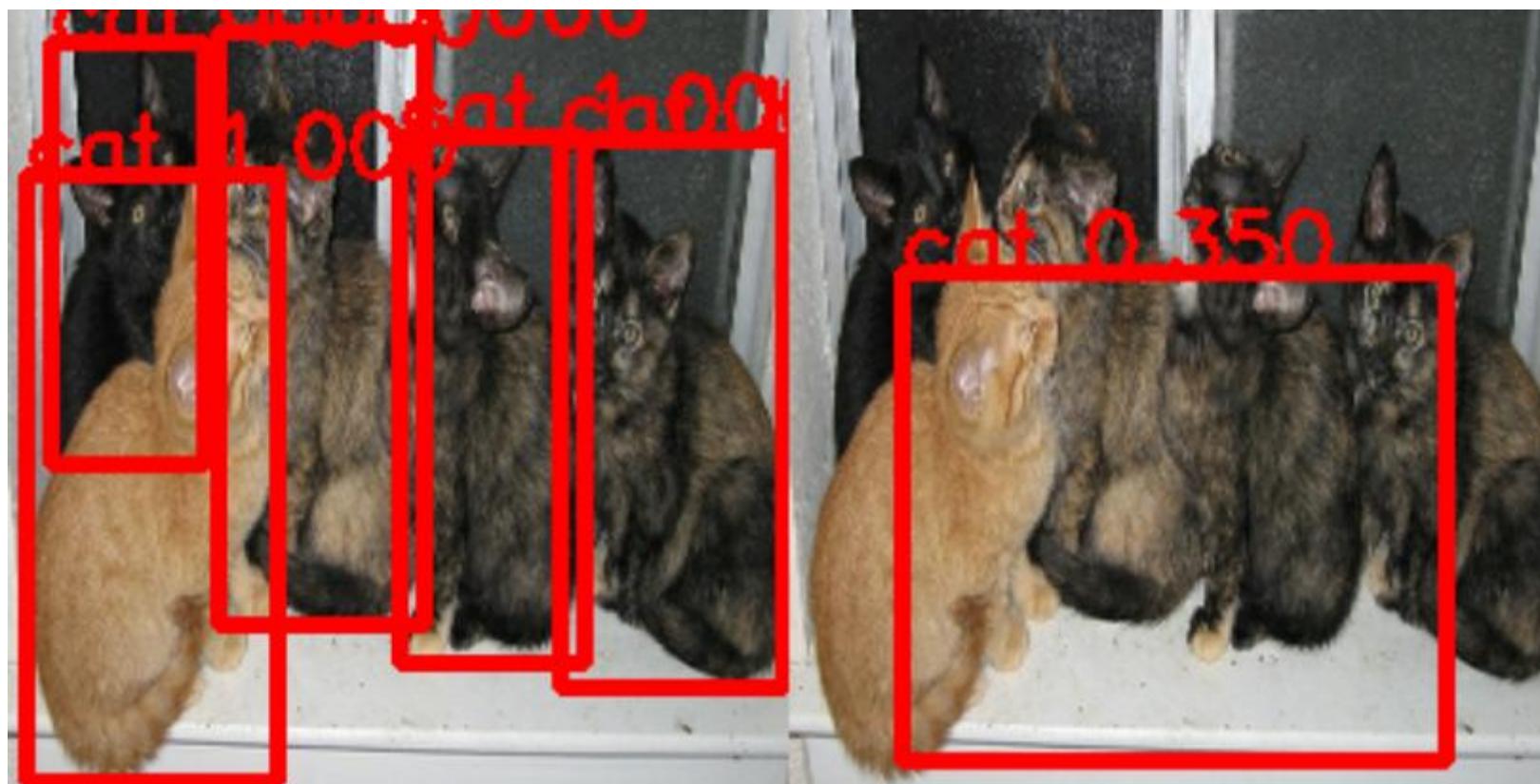
4. Responsible Predictor

- 한 그리드 셀 안에 물체가 있으면, 그 셀의 B 개 박스 예측기 중 IoU가 가장 큰 j 하나만 그 물체에 대해 "responsible predictor"로 지정 (IoU 기준 최고 점수를 가진 BBox)
- 하나의 물체를 두 박스가 중복해서 쪽는 것을 막아 안정적인 학습 유도
- 각 박스가 서로 다른 크기/비율로 자연스럽게 특화(specialize)되도록 함
- 한 셀에 물체 여러 개면?
→ YOLOv1은 Grid cell 당 객체 1개임, 다른 객체는 무시됨
- Loss 계산에서 Responsible Predictor 만 사용됨



YOLOv1 Architecture

모델 구조



한계

1. 한개의 그리드 셀 당 단 1개의 클래스만 허용하기 때문에 한 셀에 여러 개의 클래스가 한 셀에 있으면 탐지하지 못함
 - 클래스 벡터가 셀당 최대 1개
2. $S * S$ 그리드 셀로 인해 군집 이미지 처리 불리
 - 성긴 그물처럼 한 셀에 여러 객체가 있더라도 특성을 지닌 객체만 탐지
3. Anchor 의 부재로 다양한 크기의 객체를 탐지하기 어려움
 - 폭(w) / 높이(h)에 대한 예측을 직접 회귀로(\sqrt{w} , \sqrt{h}) 예측하기 때문에
오탐 비율이 높음 (w, h는 이미지 전체에 대한 비율의 형태로 소수점으로
추출) ($\sqrt{\cdot}$ 형태로 예측하는 이유: 큰 BBox가 Loss 를 지배하지 못하게
하고, 작은 박스의 오차에 더 민감해짐)
4. Fully Connected Head 사용으로 인해 입력 해상도의 유연성이 부족하고,
공간적 일반화 능력이 떨어짐
 - Fully Convolution Network 에 비해 2차원 공간에 대한 이해도가
부족함

YOLOv1 Loss Function

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

Object가 존재하는 grid cell i

object가 존재하는 grid cell i의 predictor bounding box j
object가 존재하면 1, 존재하지 않으면 0

object가 존재하는 grid cell i의 predictor bounding box j에 대한 bounding box 중심의 제곱오차

object가 존재하는 grid cell i의 predictor bounding box j에 대한 bounding box 크기의 오차

object가 존재하는 grid cell i의 predictor bounding box j에 대한 confidence score 오차

object가 존재하지 않는 grid cell i의 bounding box j에 대한 confidence score 오차

object가 존재하지 않는 grid cell i의 bounding box j
object가 존재하면 0, 존재하지 않으면 1

Object가 존재하는 grid cell i에 대해 conditional class probability 오차

MSE 형태 Loss의 한계

1. x,y,w,h에 대한 L2 거리를 줄여도, IoU가 개선되지 않음
2. Classification을 MSE로 계산할 경우 CE/BCE 대비 과신/과소신이 심해지고 혼동 클래스 구분이 약해짐

i = Grid Cell 번호(left-top=1)

x, y = GT Box의 중심 좌표

x^* , y^* = 예측된 BBox의 중심 좌표

w, h = GT Box의 가로, 세로 길이

w^* , h^* = 예측된 BBox의 가로, 세로 길이

특징

default : $\lambda_{\text{coord}}=5$ / $\lambda_{\text{noobj}}=0.5$

1. Localization Loss(좌표 손실)

1) $\lambda_{\text{coord}}=5$

2) Grid cell에 어떤 객체가 존재하고, 예측된 클래스가 동일하고,
예측된 BBox가 존재하면 1
Grid cell에 어떤 객체가 존재하지만, 예측된 클래스가 다르면 0
예측된 BBox가 없다면 0

3) Grid cell의 GT Box의 중심 좌표와

예측된 BBox의 중심 좌표 사이의 MSE 형태

2. Localization Loss(크기 손실)

1) $\lambda_{\text{coord}}=5$

2) Grid cell에 어떤 객체가 존재하고, 예측된 클래스가 동일하고,
예측된 BBox가 존재하면 1
Grid cell에 어떤 객체가 존재하지만, 예측된 클래스가 다르면 0
예측된 BBox가 없다면 0

3) Grid cell의 GT Box의 가로(\sqrt{w}) 세로(\sqrt{h}) 길이와

BBox의 가로 세로 길이 사이의 MSE 형태

3. Confidence Loss

1) $\lambda_{\text{noobj}}=0.5$

2) $C^* = \text{Pr}(\text{object}) \times \text{IoU}(\text{pred}, \text{GT})$

Responsible Predictor에는 $\text{Pr}(\text{object})=10$ 으로 타깃=IoU

3) Grid cell에 GT box가 없을 경우가 더 많기 때문에 $\lambda_{\text{noobj}}=0.5$ 로 약화

4) 각 Confidence의 MSE 형태

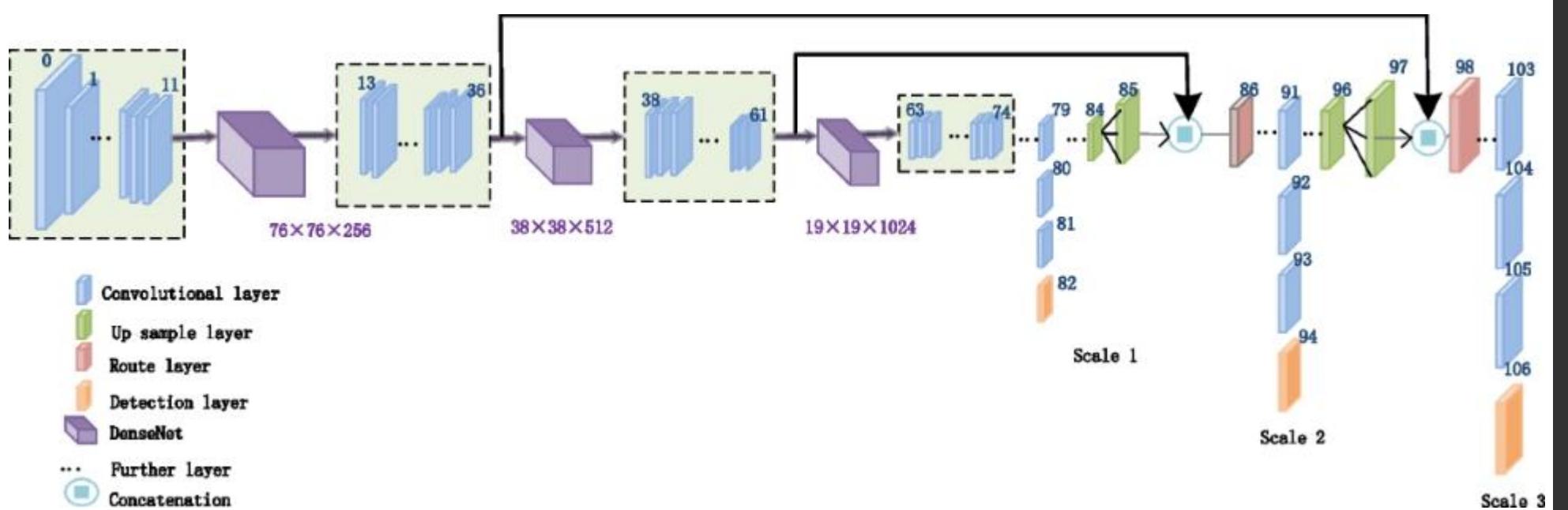
4. Classification Loss

1) $p(c) = c(\text{Class})$ 의 $p(\text{Probability})$

2) 객체 Class 확률의 MSE 형태

YOLOv3 Architecture

모델 구조



특징

1. Backbone(Darknet-53)

- 잔차 블럭(Residual Block) + Conv–BN–Leaky ReLU 스택
- 1-2-8-8-4 의 잔차 블럭

2. Neck(FPN)

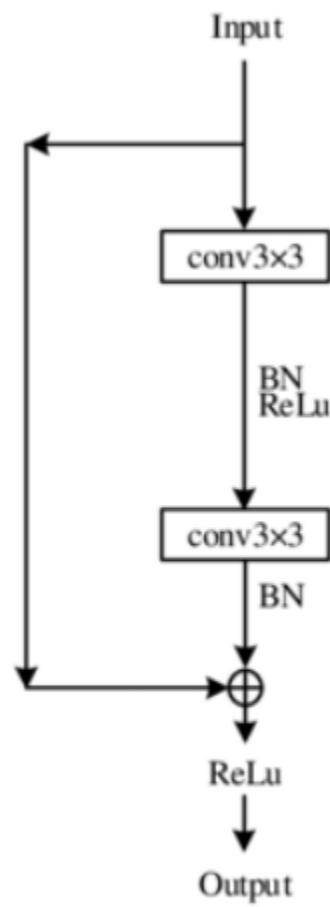
- 이전 YOLO 모델의 문제점
 - 1) 작은 물체를 탐지하기 위해서 고해상도 피처 필요 but 의미가 부족
 - 2) 큰/복잡한 물체를 탐지하기 위해서 깊은 레이어 필요 but 해상도 부족
- 해결책
 - 1) 깊은 의미 + 얕은 해상도를 양방향으로 섞어 모든 해상도에서 의미 있는 피처를 구성
(Top-down 업샘플(저해상도/강의미를 하위에 전달))
 - 2) 3x3 Smoothing(주변 문맥 블렌딩, 업샘플 anti-aliasing))

3. Head

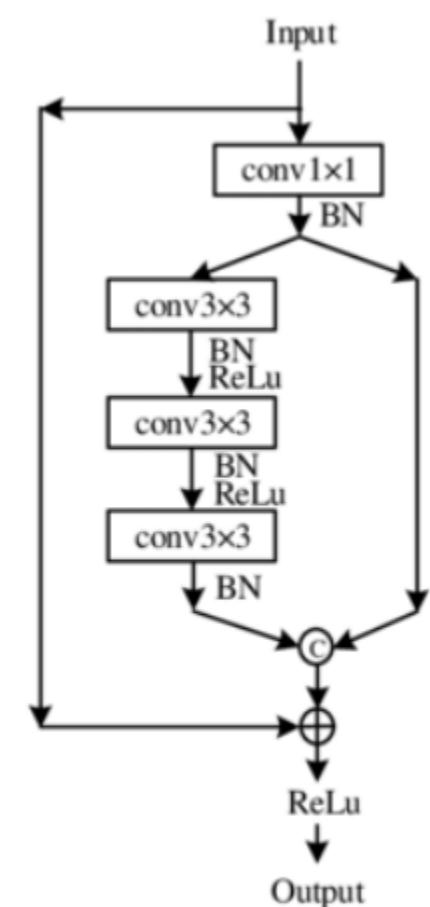
- 앵커 매칭
 - 1) GT 중심 기준, 크기/비율이 가장 잘 맞는 앵커를 positive로 할당

YOLOv3 Architecture

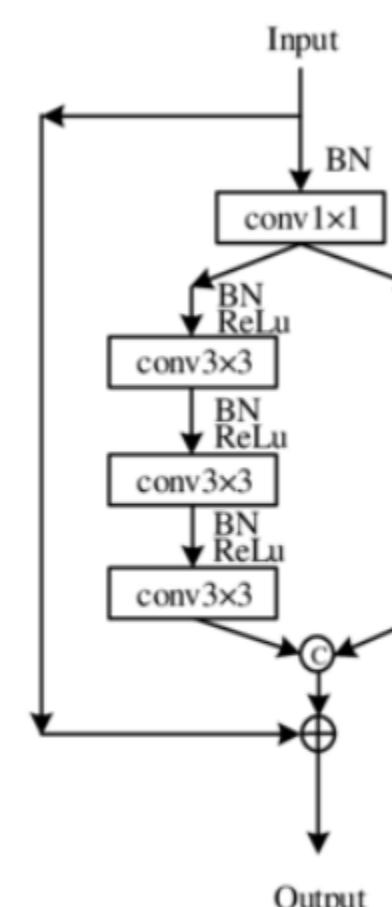
모델 구조



(a) Basic Residual Block



(b) Feature Reuse Residual block



(c) Feature Reuse Pre-activation Residual block

잔차 연결(Residual Connection)

YOLOv3의 residual connection은 Darknet-53 백본 안에서 쓰는 skip connection.

입력 x 를 두 개의 Conv($1 \times 1 \rightarrow 3 \times 3$)로 변환한 결과 $F(x)$ 에 그대로 더해서 $y = x + F(x)$ 로 반환

효과: 깊은 구조에서도 복제된 입력이 직접 경로로 흘러서 기울기 소실을 줄이고 학습이 쉬워짐

$$y = x + F(x; W), \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \cdot \left(I + \frac{\partial F}{\partial x} \right)$$

YOLOv3 Architecture

모델 구조

특징

4. k-means로 anchor boxes 추출

- k-means 클러스터링

- 1) 임의 수 k 개를 입력하여 임의의 위치 k 개를 지정

- 2) GT를 샘플링하여 입력 해상도로 변환 $\rightarrow k$ 개의 묶음으로 나눔

- 3) IoU를 측정하여 anchor box 후보의 w, h 를 갱신

- 4) 반복

예시)

GT (w, h): (20,30), (22,32), (80,40), (78,42), (210,190), (200,180)

예시) $k=3$

소 Anchor box \rightarrow 중심 $\approx (21,31)$

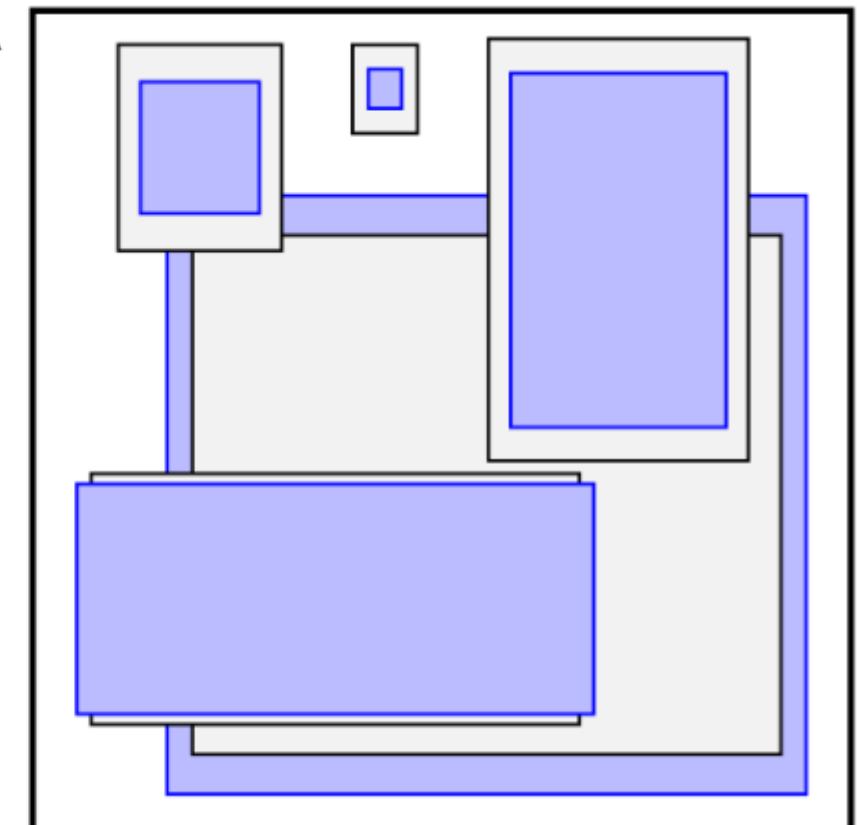
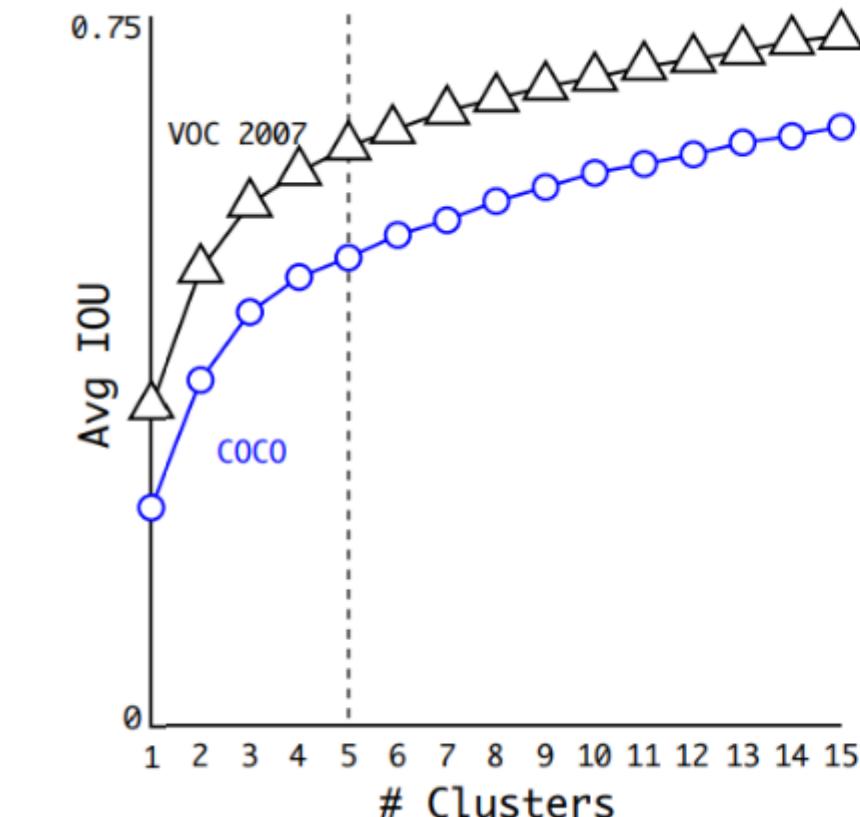
중 Anchor box \rightarrow 중심 $\approx (79,41)$

대 Anchor box \rightarrow 중심 $\approx (205,185)$

\rightarrow 이 3개가 앵커 박스. YOLOv3에서는 $k=30$ 이 사용됨.

\rightarrow 스케일당 하나씩 배치

\rightarrow 실사용에서 주로 $k=9$ 로 사용(C5/C4/C3 각 3개씩 배치)

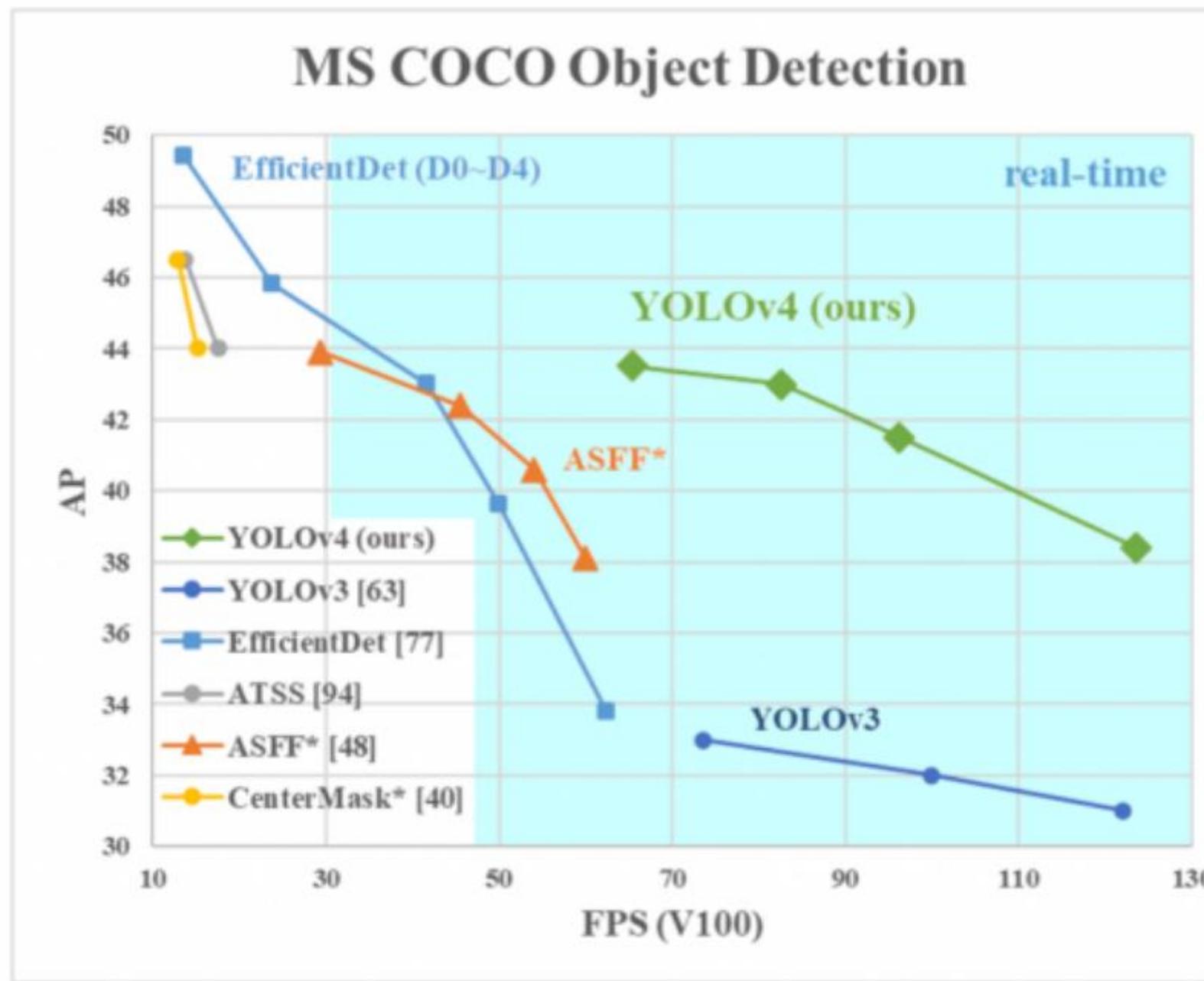


We still use k-means clustering to determine our bounding box priors. We just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales.

출처: YOLOv3: An Incremental Improvement - J.Redmond et al.

YOLOv3 Architecture

모델 구조



한계

- 무거운 Backbone
 - Darknet-53은 정확도 대비 효율이 나쁘진 않지만, 최신 CSP/Rep/ConvNeXt 대비 연산·메모리 효율이 떨어짐
- 앵커 기반 의존성
 - 앵커 크기·비율 설정에 성능이 민감
 - 데이터가 바뀌면 재추출(k-means 등)·재학습이 필요
(극단적 종횡비·크기 분포에 취약)
- NMS 의존성
 - 전통 NMS/Soft-NMS 의존
→ 중첩 객체(특히 작은 물체 밀집)에서 오탐·미탐 발생
- FPN(Top down) 사용의 한계
 - 상향식 정보 회수(PANet 의 bottom-up 경로) 부재
저해상도 강의미 ↔ 고해상도 국소정보의 양방향 융합 부족
- Scale 수 부족: 극소 물체(수~10 px) 탐지 불리

YOLOv5 Architecture

모델 구조

1. Backbone

CSPDarknet + SPPF

- CSP(Cross Stage Partial) : 피처를 두 갈래로 나눠, 한 갈래만 여러 Bottleneck(Residual-like)을 통과시킨 뒤 concat하여 합침
- SPPF (Spatial Pyramid Pooling – Fast) : $k=5$ MaxPool을 연속 3번 적용해서 사실상 SPP(Spatial Pyramid Pooling)을 근사 ($k=5, s=1, p=(k-1)/2$)으로 원 해상도 유지)

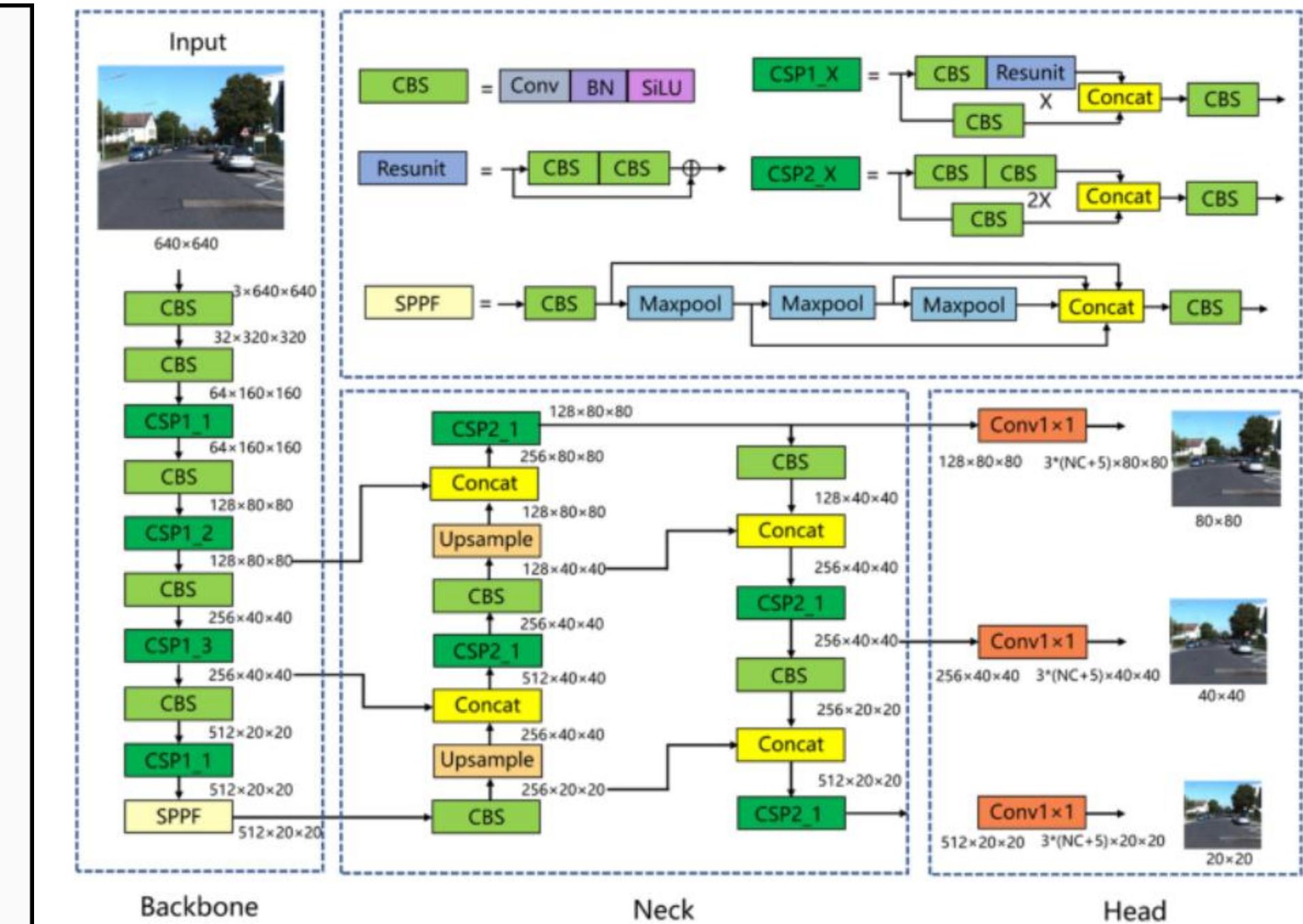
2. Neck

FPN + PANet

- PANet(Path Aggregation Network) : Bottom Up (양방향 피라미드)

3. Head

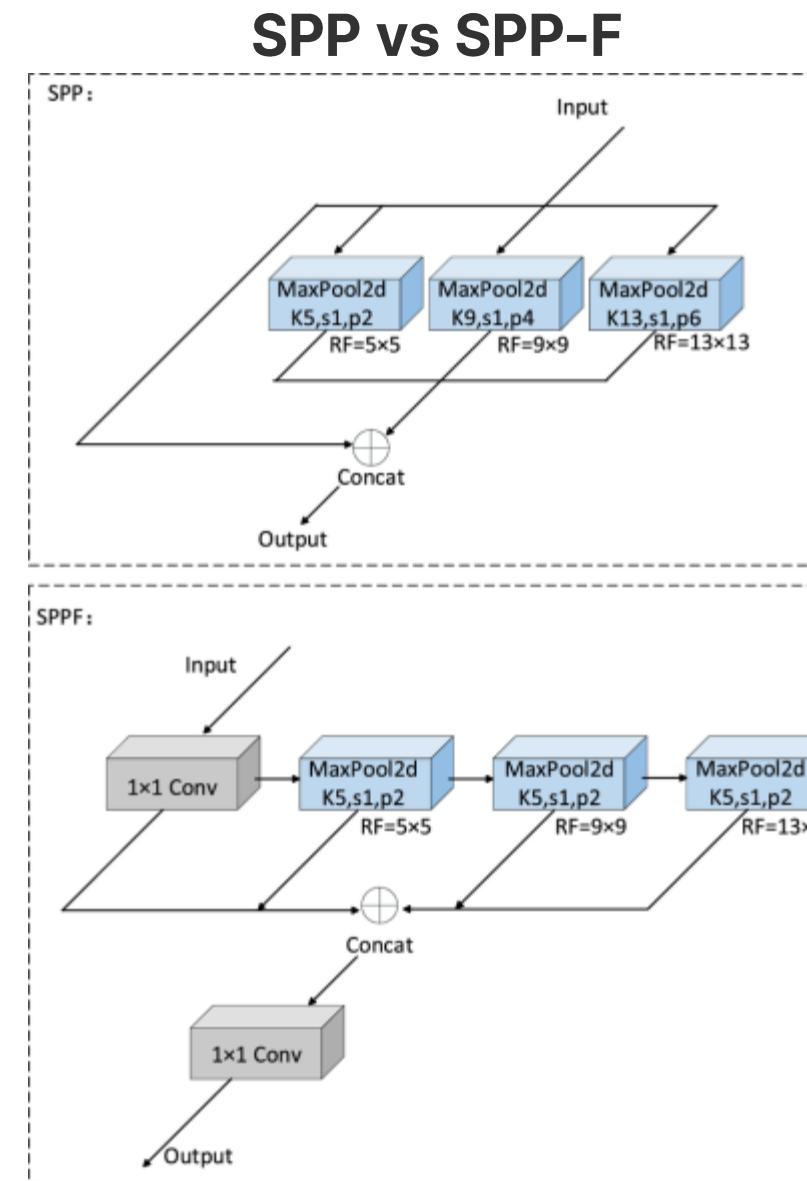
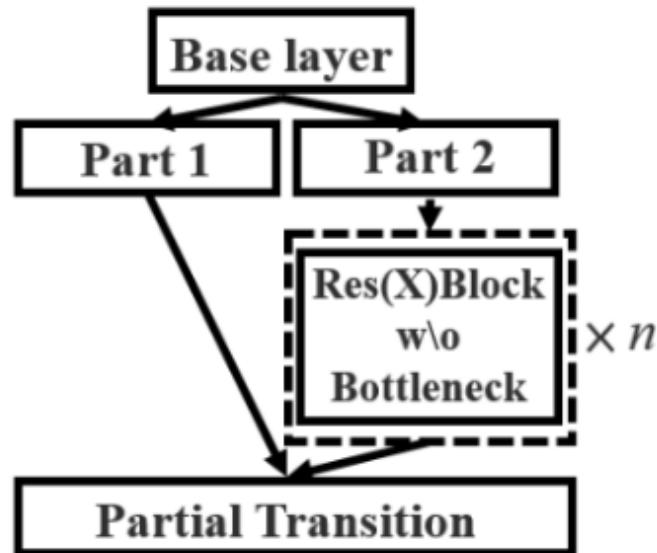
- Conv(1×1)로 $na * (nc + 5)$ 출력 ($na = \text{number of anchors} | n = \text{number of classes}$)
- Coupled Head: cls/box/obj를 Conv(1×1)로 한번에 추출 ($\text{conf}[c] = p_{\text{obj}} * p_{\text{cls}}[c]$)



YOLOv5 Architecture

모델 구조

CSP(Cross-Stage Partial)



특징

1. Backbone (CSPDarknet)
 - 1) CSP(Cross-Stage Partial) : 입력 X를 채널 기준 $[x_1 | x_2]$ 로 나누고 x_1 은 레이어를 바로 통과(bypass) | x_2 는 bottleneck을 통과 다시 x_1 과 x_2 를 concat
 - 2) SPPF(Spatial-Pyramid Pooling - Fast) : 기존 SPP의 변형 형태 기존의 SPP는 MaxPool($k=5, 9, 13$)으로 각 부분의 값을 특징적으로 추출 대신 SPPF는 MaxPool($k=5, s=1, p=2$)로 원형을 유지하며 연산량 제한 → 연산량 감소로 속도 상승
2. Neck (FPN + PANet)
 <기존 문제점>
 - 1) YOLOv3 까지 사용된 FPN 단일은 Top-Down 단일 방식 → 저해상도에서의 의미를 고해상도로 전달을 할 수 있으나 고해상도에서의 의미를 저해상도로 전달하지 못함
 - 2) PANet은 Bottom-Up 융합 방식 → backbone으로부터 C3,C4,C5가 추출 → FPN에서 P3,P4,P5를 추출 → PANet에서 N3,N4,N5를 추출

YOLOv5 Architecture

모델 구조

특징

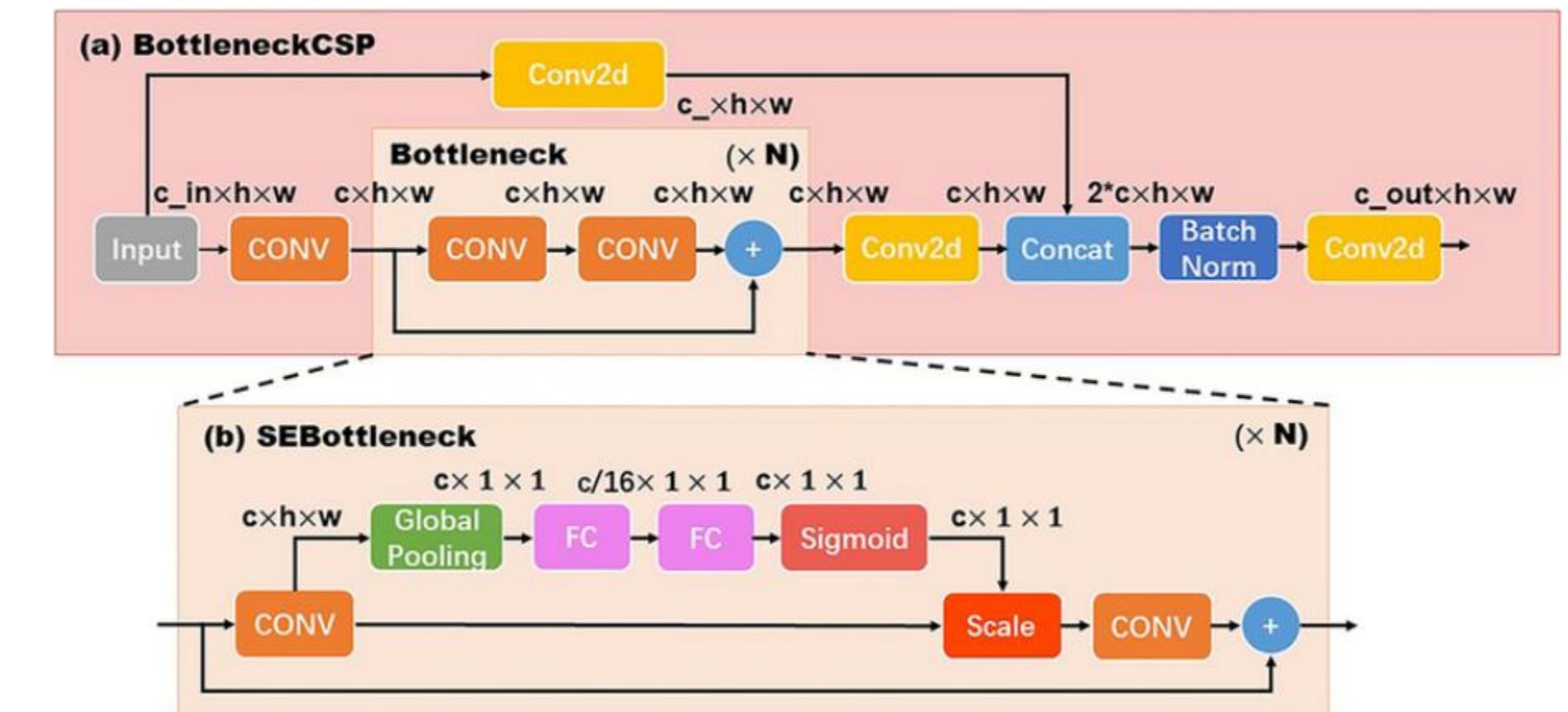
3. BottleneckCSP

- 1) input x 를 $x_1 | x_2$ 로 채널(클래스)기준 분리
→ 하나를 bypass, 하나를 SEBottleneck 에 투입
→ SEBottleneck input을 다시 채널 기준 분리
→ Add -> concat -> BatchNorm. → (다음 레이어)
ex) input(n, n, n, n) → (Bottleneck)
→ $x_1(n, n, n, n/2) | x_2(n, n, n/2)$ → x_1 은 bypass
→ $x_2(n, n, n/2)$ → (SEBottleneck)
→ $x_2_1(n, n, n, n/4) | x_2_2(n, n, n, n/4)$ -> x_2_1 은 bypass
-> x_2_2 를 Global Average Pooling -> FC → FC -> sigmoid
→ $x_2_1-x_2_2$ (add) -> x_1-x_2 (concat)

4. Head

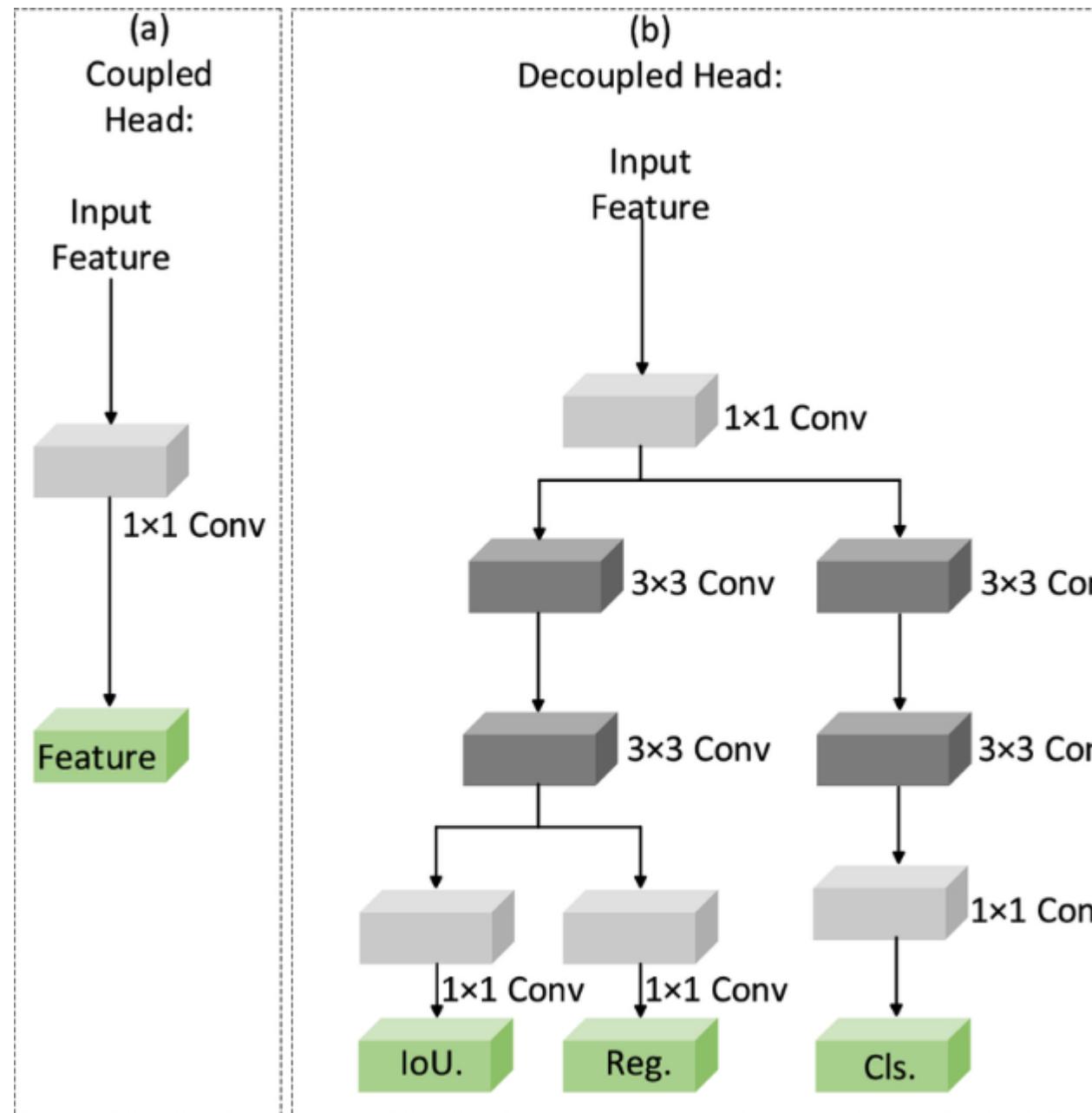
- 1) 출력 $[B, na*(nc+5), H, W]$ → reshape → $[B, na, (nc+5), H, W]$
- 2) Coupled Head : box, obj, class를 하나의 Conv로 동시에 예측

FLOP(Floating point Operations): 부동 소수점 연산 1회
FLOPs(Floating point Operations Per Second) : 총 연산 횟수



YOLOv5 Architecture

모델 구조



한계

1. Coupled head의 한계
 - box, obj, class 를 한번의 Conv로 추출
 - 그래디언트 간섭(회귀·분류 신호 충돌) 발생 가능
(군중/밀집 장면의 Detecting 성능 저하 / 클래스 불균형에 정밀도 저하)
2. Anchor-based 제약
 - 데이터 분포와 Anchor Template이 안 맞으면 미스 매치·수렴 지연
 - 다른 데이터로 이식할 때 Anchor 재계산 필요
3. NMS 의존성
 - YOLO 이전 버전들과 동일한 NMS 의존성 문제
4. 전역 문맥/장거리 상호작용 부족
 - 얇은 Convolution Layer로 인해 장거리 문맥을 파악이 불리

YOLO 전체 버전 비교

YOLOv1 (출시 연도: 2016)

<아키텍쳐>

- 단일 네트워크가 Grid Cell 단위로 BBox / Classification 동시 회귀

<Loss Function>

- 좌표·크기·객체성·분류를 MSE 형태로 최적화

2016

2017

YOLOv2 (출시 연도: 2017)

<아키텍쳐>

- 앵커 박스·패스스루/멀티스케일 예측
- Head : 앵커 기반 좌표 파라미터화
- k-means 차원 클러스터로 수렴성과 mAP를 개선

YOLOv3 (출시 연도: 2018)

<아키텍쳐>

- Darknet-53 Backbone과 FPN 도입으로 정확도/속도 균형을 개선

<Loss Function>

- objectness·클래스에 sigmoid +BCE, 멀티라벨 분류로 확률 학습을 개선

2018

2019

2020

2021

2022

YOLOv4 (출시 연도: 2020)

<아키텍쳐>

- CSPDarknet53 백본 + SPP + PANet로 피처 융합을 강화하고 단일 GPU 학습을 가능케 함

YOLOv5 (출시 연도: 2020)

<아키텍쳐>

- PyTorch 구현로 전환하고 CSP(C3)·SPPF·PAN-FPN 등 모듈을 정제해 실용 배포성/효율을 향상
- 앵커 기반·coupled head였고(이후 v5u에서 앵커프리 분할 헤드 옵션이 추가됨)

YOLOv6 (출시 연도: 2022)

<아키텍쳐>

- RepConv/효율 설계와 경량화로 실시간 성능 강화

<헤드>

- decoupled head로 BBox/Class를 분리해 정확도와 추론 효율을 개선

YOLO 전체 버전 비교

YOLOv7 (출시 연도: 2022)

- <아키텍쳐>
- E-ELAN으로 그라디언트 경로와 채널 카드inality를 설계해 큰 모델에서도 안정적 학습
 - Trainable Bag-of-Freebies와 보조 헤드 등 훈련 기법

YOLOv9 (출시 연도: 2024)

- <아키텍쳐>
- GELAN(Generalized Efficient Layer Aggregation Network)
- <모델 개선>
- PGI(Programmable Gradient Information)로 Loss 계산에 완전한 입력 정보를 주입해 그라디언트 품질을 향상.

2022

2023

2024

2025

YOLOv8 (출시 연도: 2023)

- <아키텍쳐>
- Anchor-free · decoupled head(분할 Ultralytics head)
 - C2f 블록과 경량화된 네트워크로 파라미터 효율을 개선

YOLOv10 (출시 연도: 2024)

- <후처리>
- CDA(Consistent Dual Assignments)로 NMS-free 학습을 구현해 지연을 줄이고 일관성을 확보
 - 효율-정확 동시 최적화 설계로 레이턴시와 파라미터를 절감

YOLOv11 (출시 연도: 2024)

- <아키텍쳐>
- 적은 파라미터/빠른 CPU 추론으로 mAP 향상
 - 전반적으로 v8 대비 정확도↑·FLOPs/속도 효율↑

YOLOv12 (출시 연도: 2025)

- <아키텍쳐>
- Attention-centric 설계로 CNN 중심 설계에서 한 단계 진화
 - 실시간성을 유지하면서 Attention 기반 전역 문맥을 적극 활용

YOLOv1 구현 코드

coco dataset

```

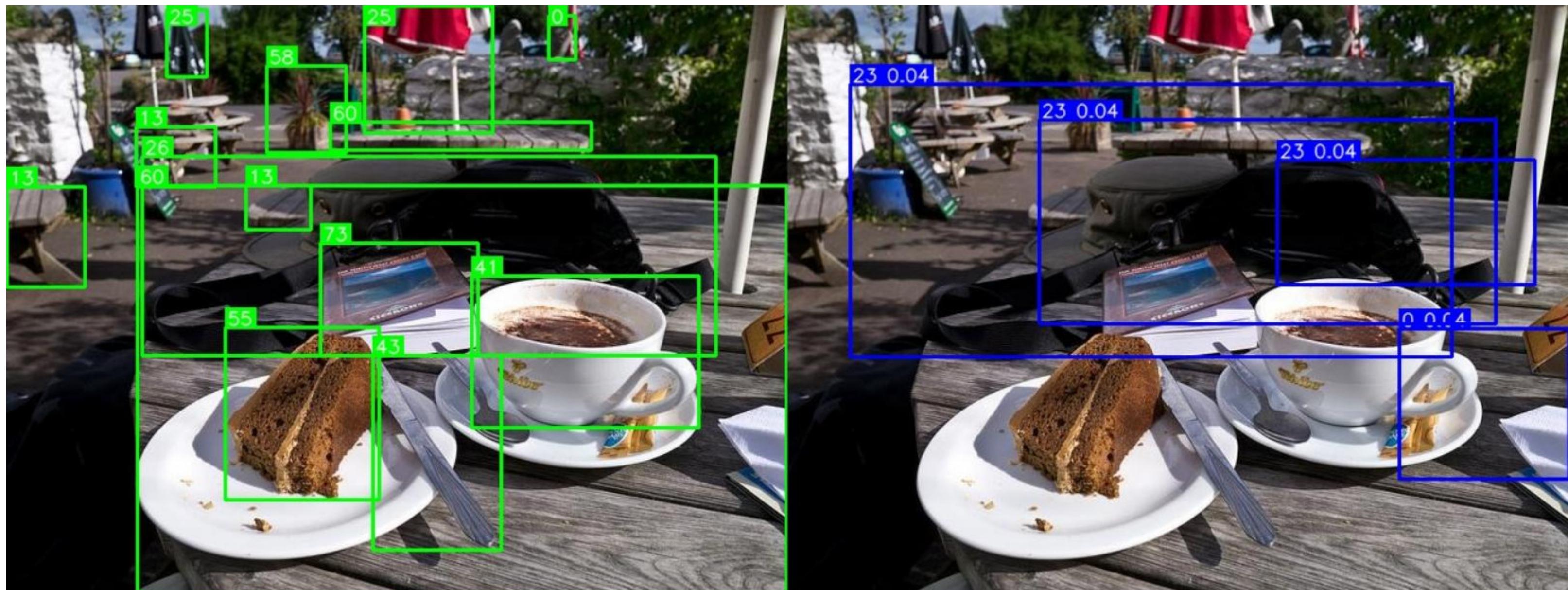
class YOLOv1(nn.Module):
    def __init__(self, S=S, B=B, C=C):
        super().__init__()
        self.S, self.B, self.C = S, B, C
        self.backbone = nn.Sequential(
            ConvBNL(3, 64, 7, 2, 3),      # 224
            nn.MaxPool2d(2,2),           # 112
            ConvBNL(64,128,3,1,1),
            nn.MaxPool2d(2,2),           # 56
            ConvBNL(128,256,3,1,1),
            nn.MaxPool2d(2,2),           # 28
            ConvBNL(256,512,3,1,1),
            nn.MaxPool2d(2,2),           # 14
            ConvBNL(512,1024,3,1,1),
        )
        self.neck = nn.Sequential(
            nn.AdaptiveAvgPool2d((7,7)),   # 7x7
            ConvBNL(1024,512,3,1,1),
        )
        self.head = nn.Conv2d(512, B*5 + C, 1, 1, 0) # 7x7x(10+C)

    def forward(self, x):
        x = self.backbone(x)
        x = self.neck(x)
        x = self.head(x)                  # [N, 10+C, 7,7]
        x = x.permute(0,2,3,1).contiguous() # [N,7,7,10+C]
        return x

```

YOLOv1 구현 결과

coco dataset



YOLOv3 구현 코드

coco dataset

```

def main():
    set_seed(SEED)
    download_coco128()
    write_coco128_yaml()

    from ultralytics import YOLO
    model = YOLO("yolov3u.pt")

    ultra_device = 0 if torch.cuda.is_available() else "cpu"

    # ---- 학습 ----
    model.train(
        data=str(DATA_YAML),
        epochs=EPOCHS,
        imgsz=IMG_SIZE,
        batch=BATCH_SIZE,
        seed=SEED,
        device=ultra_device,
        project=str(RUNS_DIR),
        name="train_coco128",
        exist_ok=True,
        verbose=True
    )

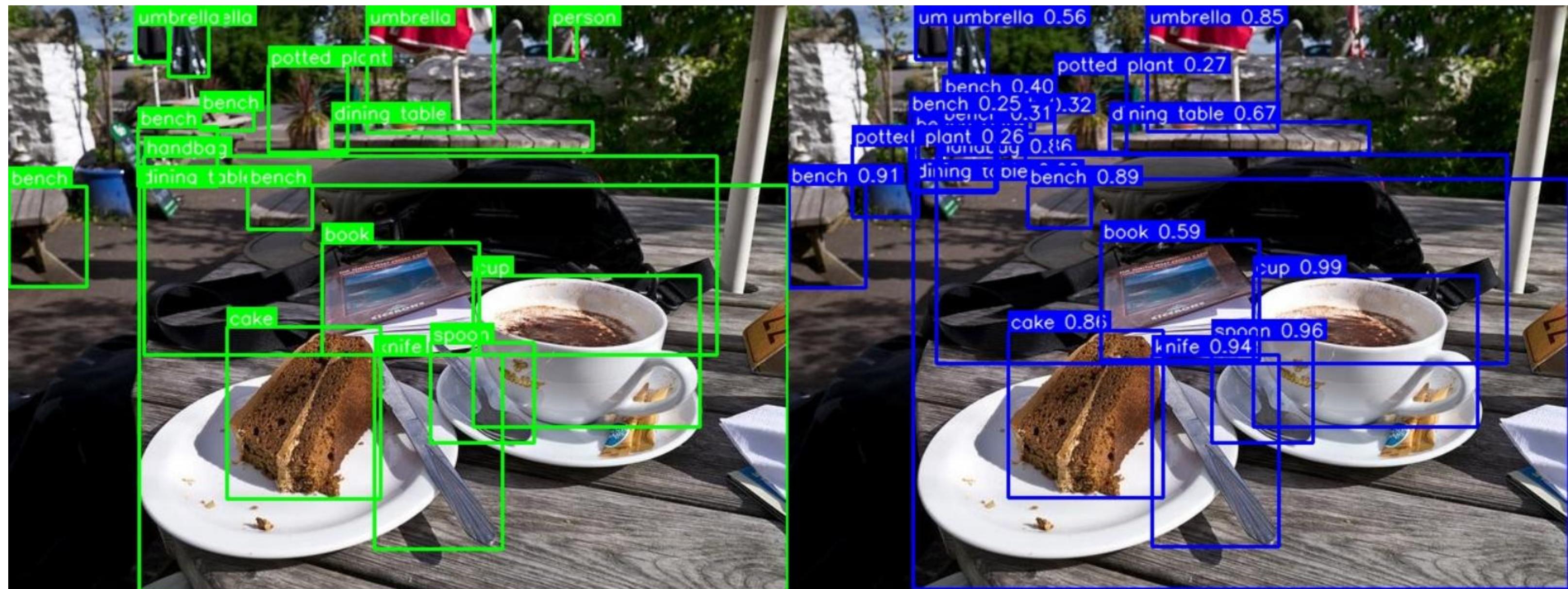
    # ---- 검증(mAP) ----
    val_res = model.val(
        data=str(DATA_YAML), imgsz=IMG_SIZE, conf=0.001, iou=IOU_THRES,
        split="val", device=ultra_device, project=str(RUNS_DIR),
        name="val_coco128", exist_ok=True, verbose=False
    )
    try:
        mAP_50_95 = float(val_res.box.map); mAP_50 = float(val_res.box.map50)
    except Exception:
        P,R,mAP_50,mAP_50_95 = val_res.mean_results()
    print(f"[YOLOv3] mAP@50-95: {mAP_50_95:.4f}")
    print(f"[YOLOv3] mAP@50 : {mAP_50:.4f}")

    # ---- 예측 저장(ultralytics 기본 저장) ----
    pred_results = model.predict(
        source=str(IMAGES_DIR), imgsz=IMG_SIZE, conf=CONF_THRES, iou=IOU_THRES,
        device=ultra_device, save=True, save_txt=True, save_conf=True,
        project=str(OUT_DIR), name="pred_images", exist_ok=True, verbose=False
    )

```

YOLOv3 구현 결과

coco dataset



YOLOv5 구현 코드

coco dataset

```

def main():
    set_seed(SEED)
    download_coco128()
    write_coco128_yaml()

    from ultralytics import YOLO
    try:
        model = YOLO("yolov5su.pt")
    except Exception:
        model = YOLO("yolov5s.pt")

    ultra_device = 0 if torch.cuda.is_available() else "cpu"

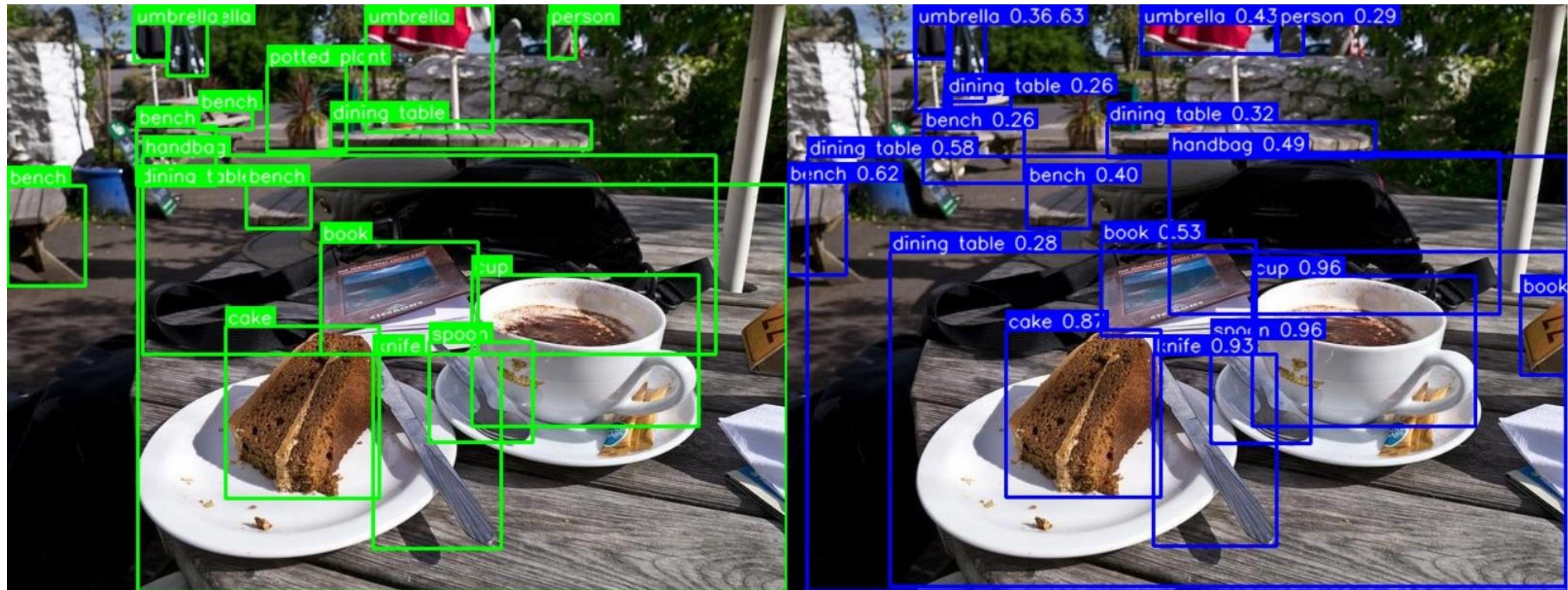
    # ---- 학습 ----
    model.train(
        data=str(DATA_YAML),
        epochs=EPOCHS,
        imgsz=IMG_SIZE,
        batch=BATCH_SIZE,
        seed=SEED,
        device=ultra_device,
        project=str(RUNS_DIR),
        name="train_coco128_v5",
        exist_ok=True,
        verbose=True
    )

    # ---- 검증(mAP) ----
    val_res = model.val(
        data=str(DATA_YAML), imgsz=IMG_SIZE, conf=0.001, iou=IOU_THRES,
        split="val", device=ultra_device, project=str(RUNS_DIR),
        name="val_coco128_v5", exist_ok=True, verbose=False
    )
    try:
        mAP_50_95 = float(val_res.box.map); mAP_50 = float(val_res.box.map50)
    except Exception:
        P,R,mAP_50,mAP_50_95 = val_res.mean_results()
    print(f"[YOLOv5] mAP@50-95: {mAP_50_95:.4f}")
    print(f"[YOLOv5] mAP@50     : {mAP_50:.4f}")

```

YOLOv5 구현 결과

coco dataset



YOLOv12 구현 코드

coco dataset

```

def main():
    set_seed(SEED)
    download_coco128()
    write_coco128_yaml()

    from ultralytics import YOLO
    try:
        model = YOLO("yolov12n.pt")
    except Exception:
        model = YOLO("yolov12s.pt")

    ultra_device = 0 if torch.cuda.is_available() else "cpu"

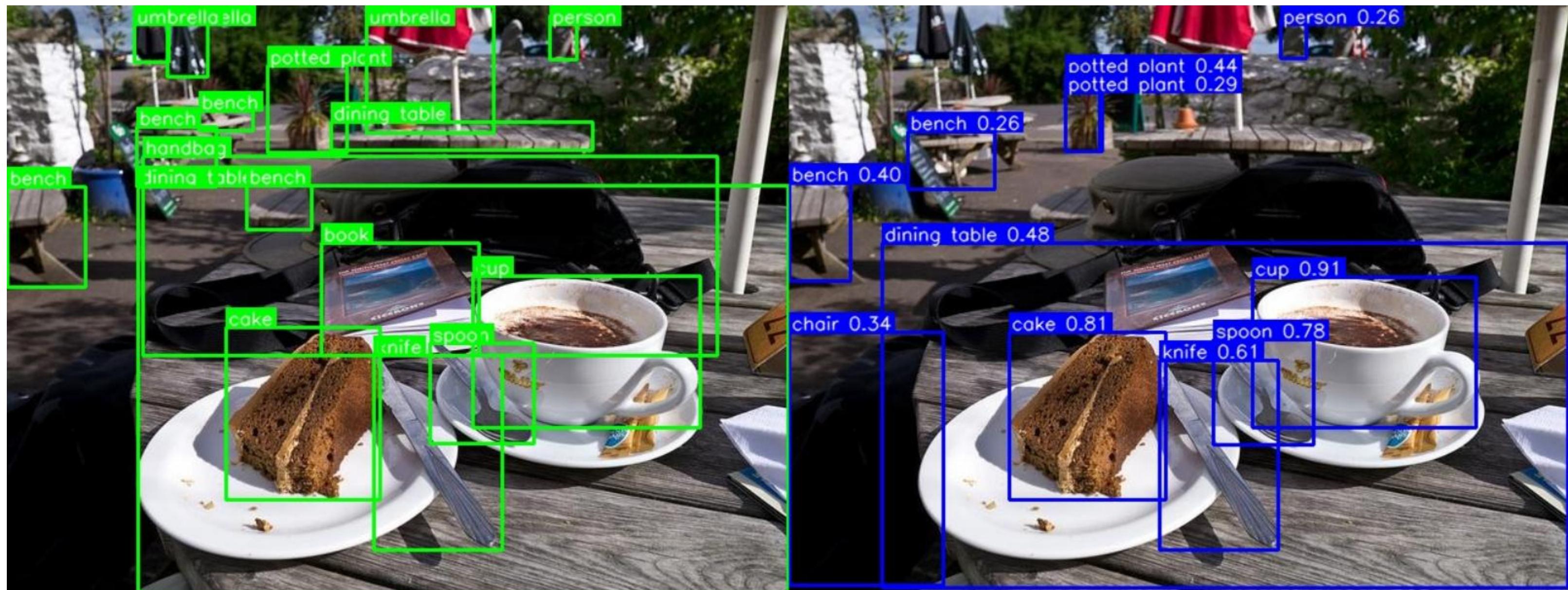
    # ---- 학습 ----
    model.train(
        data=str(DATA_YAML),
        epochs=EPOCHS,
        imgsz=IMG_SIZE,
        batch=BATCH_SIZE,
        seed=SEED,
        device=ultra_device,
        project=str(RUNS_DIR),
        name="train_coco128_v12",
        exist_ok=True,
        verbose=True
    )

    # ---- 검증(mAP) ----
    val_res = model.val(
        data=str(DATA_YAML), imgsz=IMG_SIZE, conf=0.001, iou=IOU_THRES,
        split="val", device=ultra_device, project=str(RUNS_DIR),
        name="val_coco128_v12", exist_ok=True, verbose=False
    )
    try:
        mAP_50_95 = float(val_res.box.map); mAP_50 = float(val_res.box.map50)
    except Exception:
        P,R,mAP_50,mAP_50_95 = val_res.mean_results()
    print(f"[YOLOv12] mAP@50-95: {mAP_50_95:.4f}")
    print(f"[YOLOv12] mAP@50     : {mAP_50:.4f}")

```

YOLOv12 구현 결과

coco dataset



레모네이드 Detection

coco dataset

```

# 1) Split
print("[STEP] Build classification dataset from faces_crops")
classes = make_split_from_crops(crops_dir, data_dir, VAL_RATIO, seed=SEED)
print(f"[INFO] classes ({len(classes)}): {classes}")

# 2) Augment (train만)
print("[STEP] Augment & Balance train set")
train_root = data_dir / "train"
augment_and_balance(train_root, img_size=IMG_SIZE, mode=BALANCE_MODE,
                     target_per_class=TARGET_PER_CLASS, max_aug_per_src=MAX_AUG_PER_SRC, seed=SEED)

# 3) Train YOLOv11-CLS
print("[STEP] Train YOLOv11-CLS")
weights = pick_weights()
model = YOLO(weights)
model.train(
    data=str(data_dir),
    epochs=EPOCHS, imgsz=IMG_SIZE,
    batch=BATCH, device=device,
    workers=0,                      # Windows 안전
    auto augment="randaugment",     # 분류 전용 강한 증강 추가
    erasing=0.25,                  # Random Erasing
    lr0=0.001, patience=20, verbose=True,
    label_smoothing=0.1
)

run_dir = latest_run_dir(Path.cwd())
if run_dir is None:
    raise RuntimeError("학습 결과(run dir)를 찾지 못했습니다.")
best = run_dir / "weights" / "best.pt"
print(f"[INFO] best weights: {best}")
cls_model = YOLO(str(best))

```

레모네이드 Detection

coco dataset

```

# 4) Test: MTCNN 검출 -> 분류 -> 시각화
print("[STEP] Detect & Classify faces on test images")
mtcnn = MTCNN(image_size=160, margin=10, keep_all=True, post_process=True, device=device if device!="cpu" else None)
test_imgs = [p for p in test_dir.iterdir() if p.is_file() and p.suffix.lower() in [".jpg",".jpeg",".png",".bmp",".webp"]]
test_imgs = sorted(test_imgs, key=lambda p: ("test" not in p.name.lower(), p.name.lower()))

total_faces = 0
for img_path in tqdm(test_imgs):
    try:
        img = Image.open(img_path).convert("RGB")
    except:
        continue
    boxes, probs = mtcnn.detect(img)
    if boxes is None or len(boxes)==0:
        continue
    keep = [(b,p) for b,p in zip(boxes, probs) if p is None or p >= CONF_FACE]
    if not keep:
        continue
    keep = sorted(keep, key=lambda x: ((x[0][2]-x[0][0])*(x[0][3]-x[0][1])), reverse=True)
    if KEEP_TOPK is not None:
        keep = keep[:KEEP_TOPK]

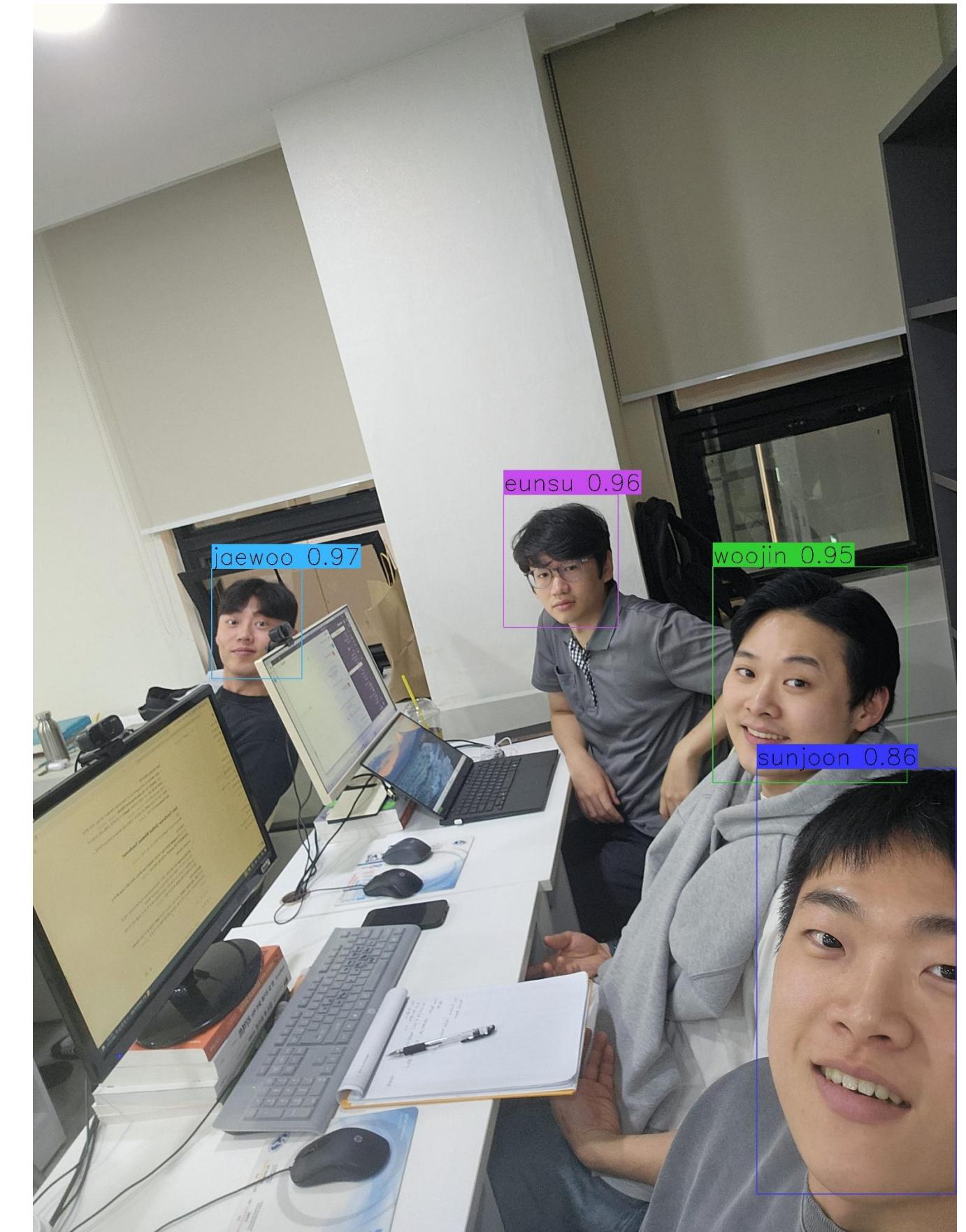
    keep_boxes, keep_labels = [], []
    for b, p in keep:
        x1,y1,x2,y2 = [int(v) for v in b]
        crop = img.crop((max(0,x1), max(0,y1), x2, y2)).resize((IMG_SIZE, IMG_SIZE))
        res = cls_model.predict(source=crop, imgsz=IMG_SIZE, verbose=False)[0]
        top1 = int(res.probs.top1)
        names = res.names if hasattr(res, "names") else cls_model.names
        name = names[top1] if names and top1 in range(len(names)) else str(top1)
        conf = float(res.probs.top1conf)
        keep_boxes.append(np.array([x1,y1,x2,y2], dtype=float))
        keep_labels.append(f"{name} ({conf:.2f})")
        total_faces += 1

    if keep_labels:
        vis = draw_boxes_and_labels(img, np.stack(keep_boxes,0), keep_labels)
        vis.save(vis_dir / img_path.name)

print("\n[SUMMARY]")
print(f"- Classes: {classes}")
print(f"- Best weights: {best}")
print(f"- Test visualizations: {vis_dir}")
print(f"- Total faces annotated: {total_faces}")

```

레모네이드 Detection with YOLO11



References

- Girshick, Ross. 2015. "Fast R-CNN." Proceedings of the IEEE International Conference on Computer Vision (ICCV).
- Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You Only Look Once: Unified, Real-Time Object Detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- Redmon, Joseph, and Ali Farhadi. 2018. "YOLOv3: An Incremental Improvement." arXiv 1804.02767.
- Terven, Juan, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. 2023. "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS." Machine Learning and Knowledge Extraction 5 (4): 1680–1716.
- Neubeck, Alexander, and Luc Van Gool. 2006. "Efficient Non-Maximum Suppression." In Proceedings of the 18th International Conference on Pattern Recognition (ICPR).
- Hussain, Muhammad. "YOLOv5, YOLOv8 and YOLOv10: The Go-To Detectors for Real-time Vision." arXiv, July 3, 2024.

Q&A