




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

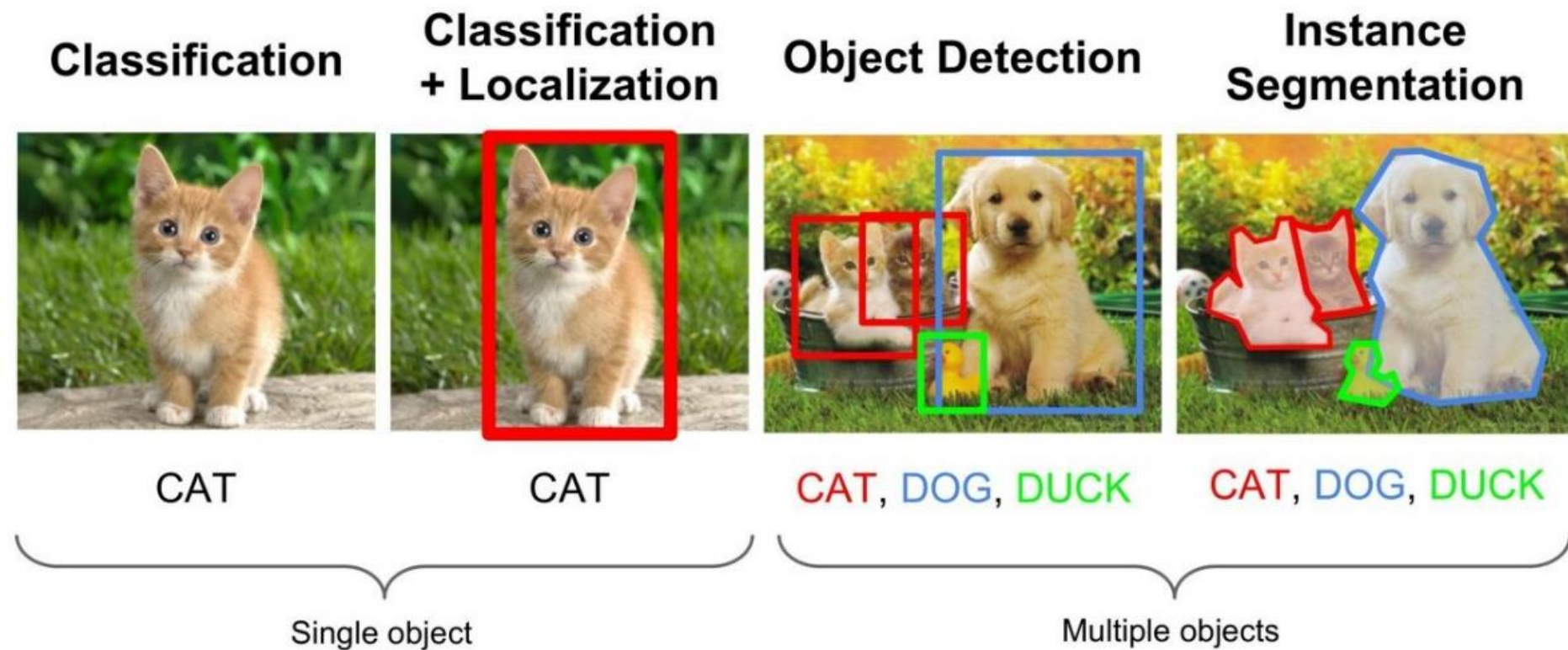
(Remon, Divvala, Girshick, & Farhadi, 2016, CVPR)




AI빅데이터융합경영학과  
20212548

김지은

# Object detection?



- Object classification: 이미지 내에 object 한 개 존재, 그것이 개인지 고양이인지 판단
- Object classification + localization: 이미지 내에 object 한 개 존재, 위치+객체 판단
- Object detection: 이미지 내에 객체가 한 개 이상 존재, 서로 다른 object의 위치 + 객체 판단
- Instance segmentation: 이미지 내에 객체가 한 개 이상 존재, 픽셀단위의 위치 + 객체 판단

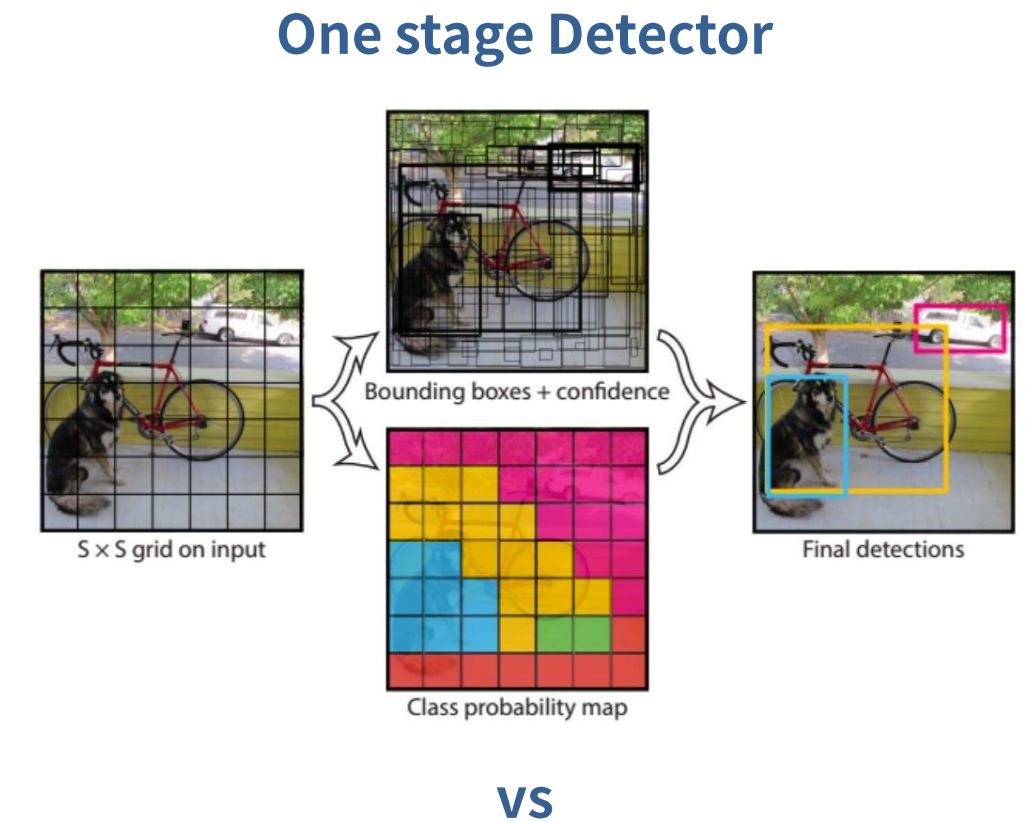
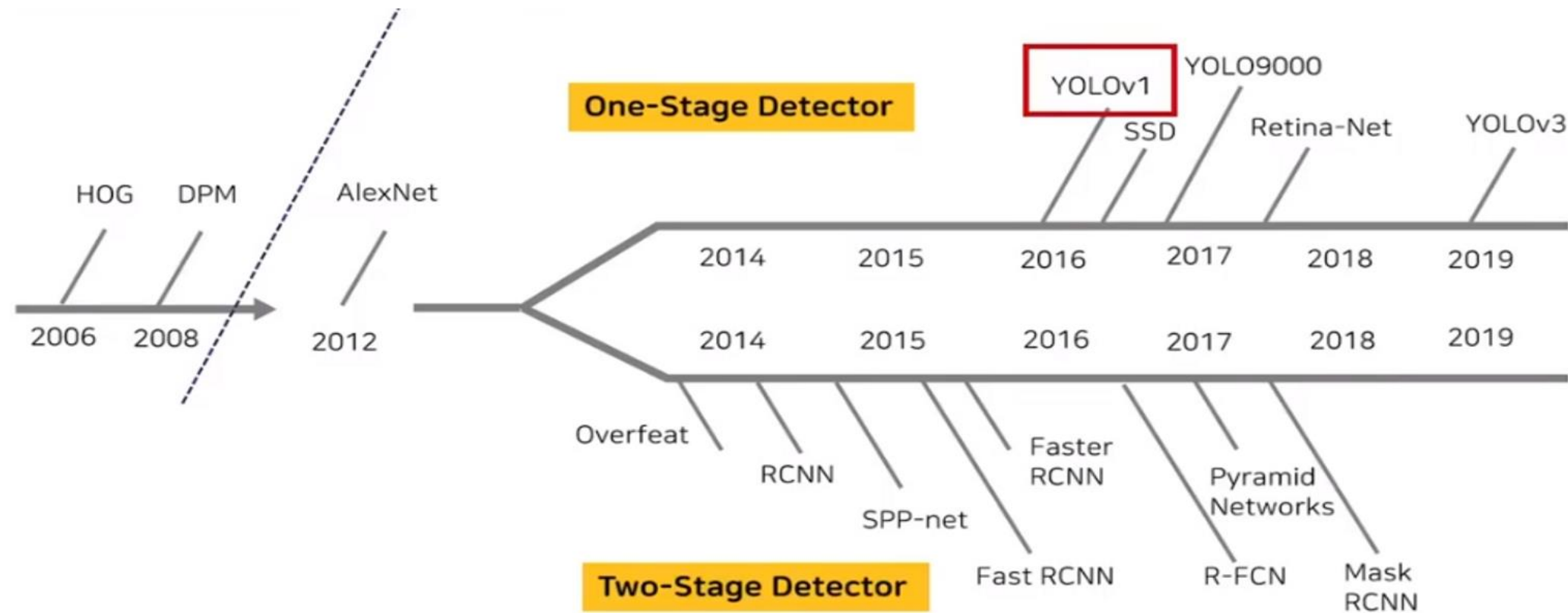


## Object detection

**Classification + localization**  
(객체 분류 + bounding box를 통한 위치 정보 파악)

\* Bounding box: 하나의 객체를 포함하는 가장 작은 직사각형

# One-stage vs Two-stage Detector



- **One stage: localization + classification을 동시에 수행**

Ex) conv를 통과한 후, 각 grid cell마다 classification 결과와 bounding box regression을 통해 결과 도출

- **Two stage: localization -> classification 순차적 진행 Ex) DPM\*, R-CNN\***

Ex) region proposal을 통해 먼저 후보 box를 추출, classification 결과와 bounding box regression을 통해 결과 도출

\* DPM(deformable parts models): 이미지 전체를 거쳐 슬라이딩 윈도우 방식으로 객체 검출

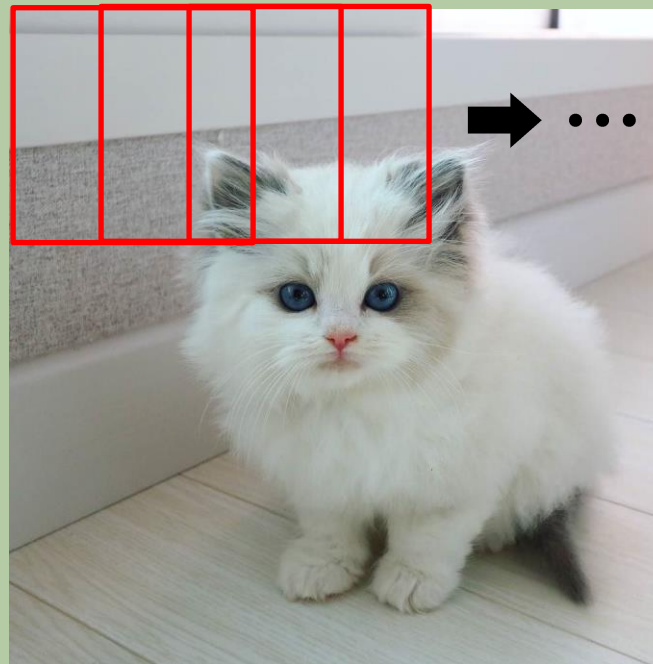
\* R-CNN: bounding box 생성 후, classification & regression -> 중복제거(NMS) 진행

➡ 느낌



# ROI (Region proposal)

## Sliding window



기존의 경우, sliding window를 이용한 방식으로  
이미지의 모든 영역을 window로 탐색  
➡ 굳이 객체가 없는 곳까지 탐색하게 됨  
➡ 비효율적임

물체가 있을 만한 영역만을 찾아내자!  
➡ Region Proposal  
Ex) Selective search

## Selective Search



1. 색상, 질감 등을 활용해 Segmentation을 먼저 실시한 후, 후보영역들을 선정
2. 많은 후보들을 적절하게 통합함
3. 적절한 box 후보들 생성

# One-stage vs Two-stage detector

## Two stage

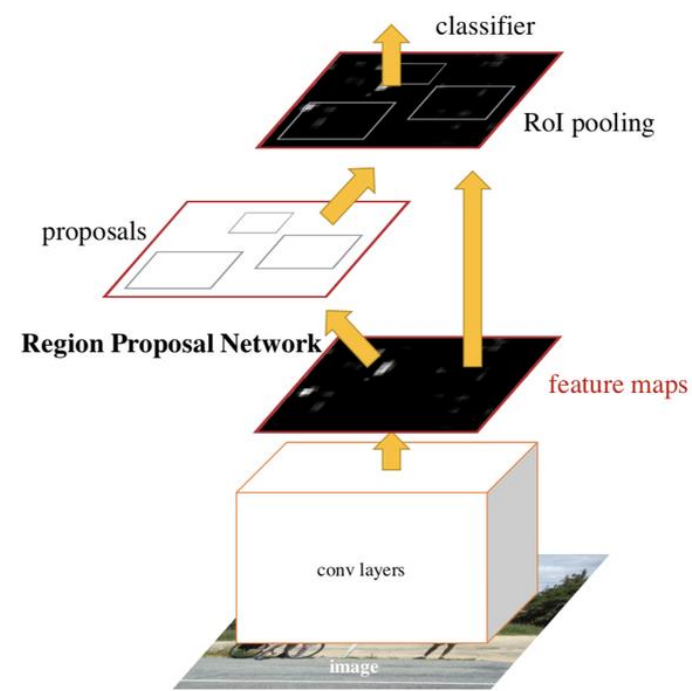


Image → Region proposal → feature extractor → classification, regression

후보 object 위치 제안 후, object class 예측  
→ 느리지만 정확도는 높음 ex) R-CNN 계열  
\*느린 또다른 이유: bbox 크기를 다 동일하게 만들어줘야 함



Figure 2: Warped training samples from VOC 2007 train.

## One stage

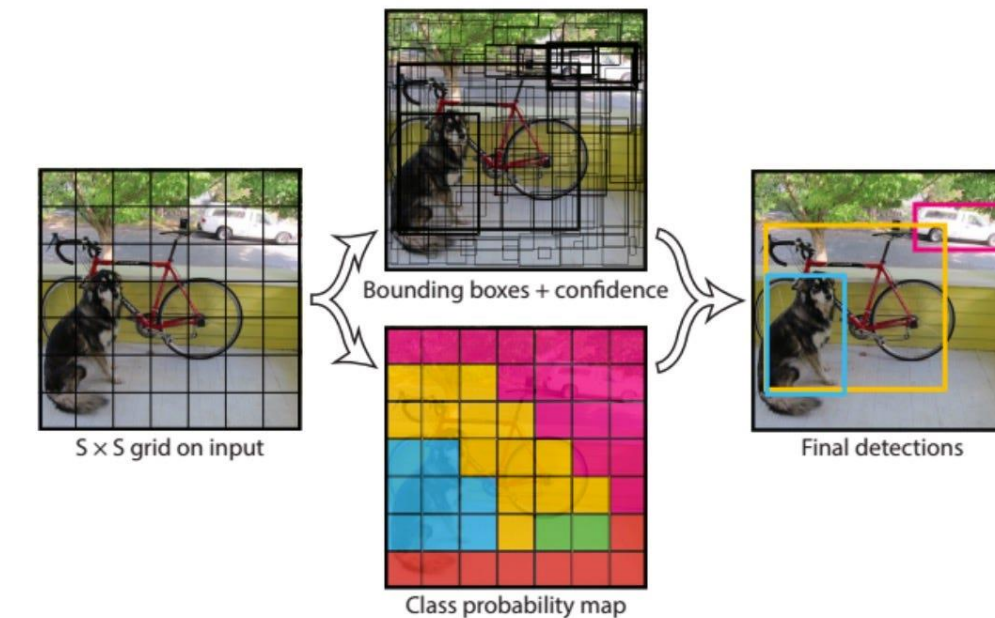


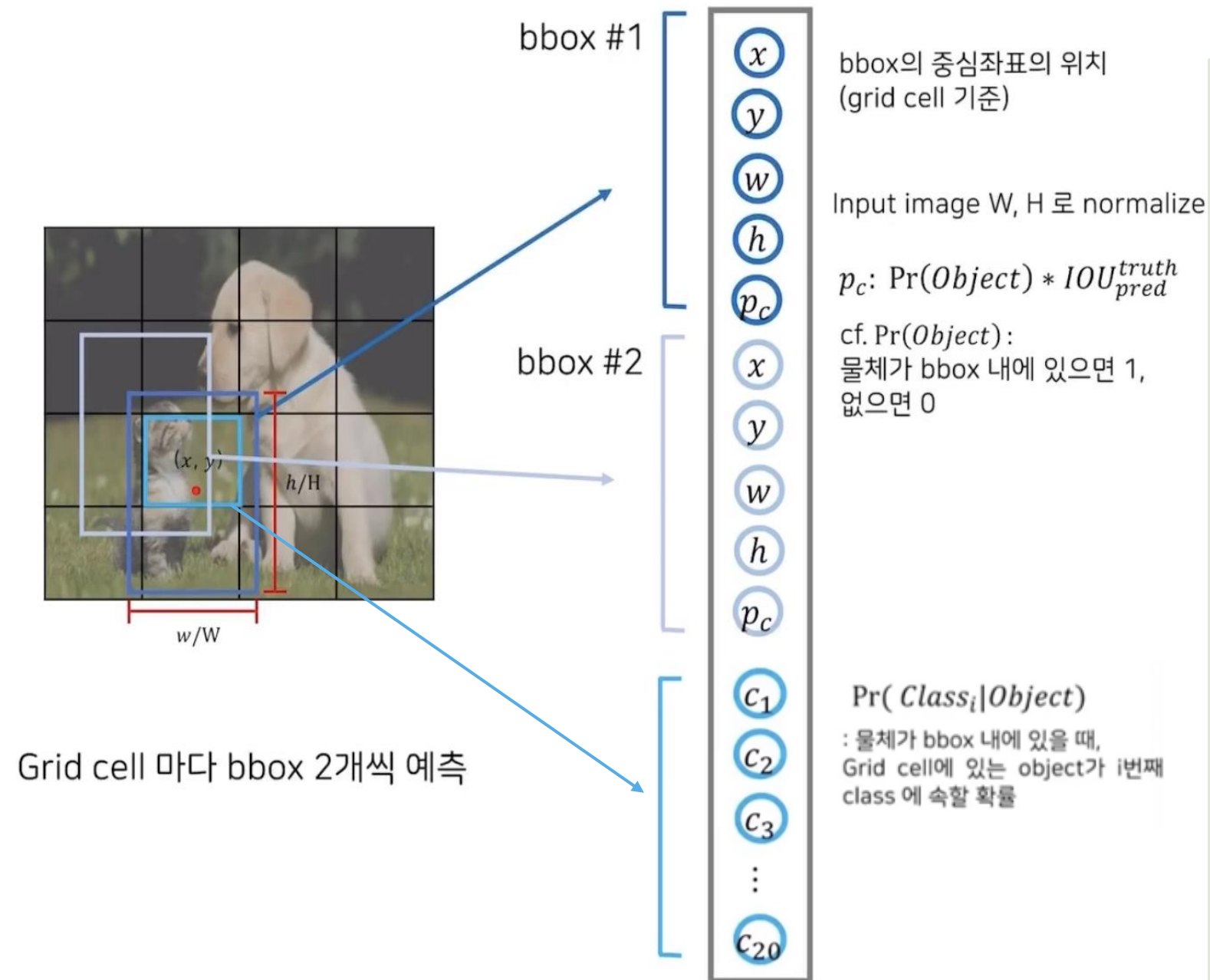
Image → feature extractor → classification, regression

ROI 영역을 추출하지 않고, 전체 이미지를 봄(각 grid cell마다 2개의 bbox 산출)  
→ 이미지를 한 번 보는 것만으로도 object의 위치와 종류를 추측할 수 있음!  
→ 빠르지만 정확도는 비교적 낮음 ex) YOLO

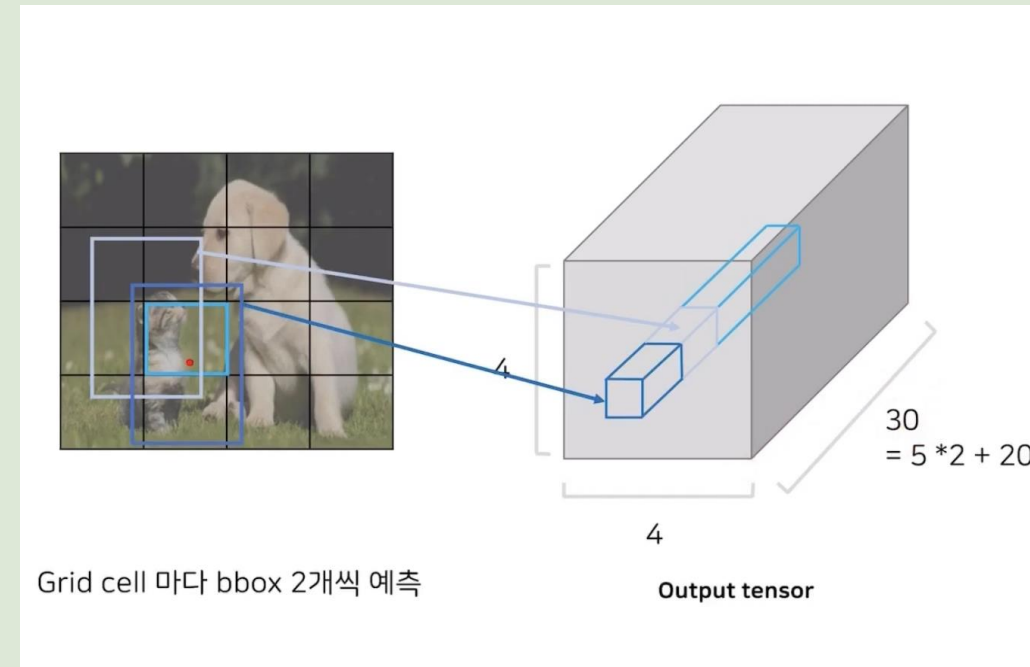


# Yolo – Unified Detection

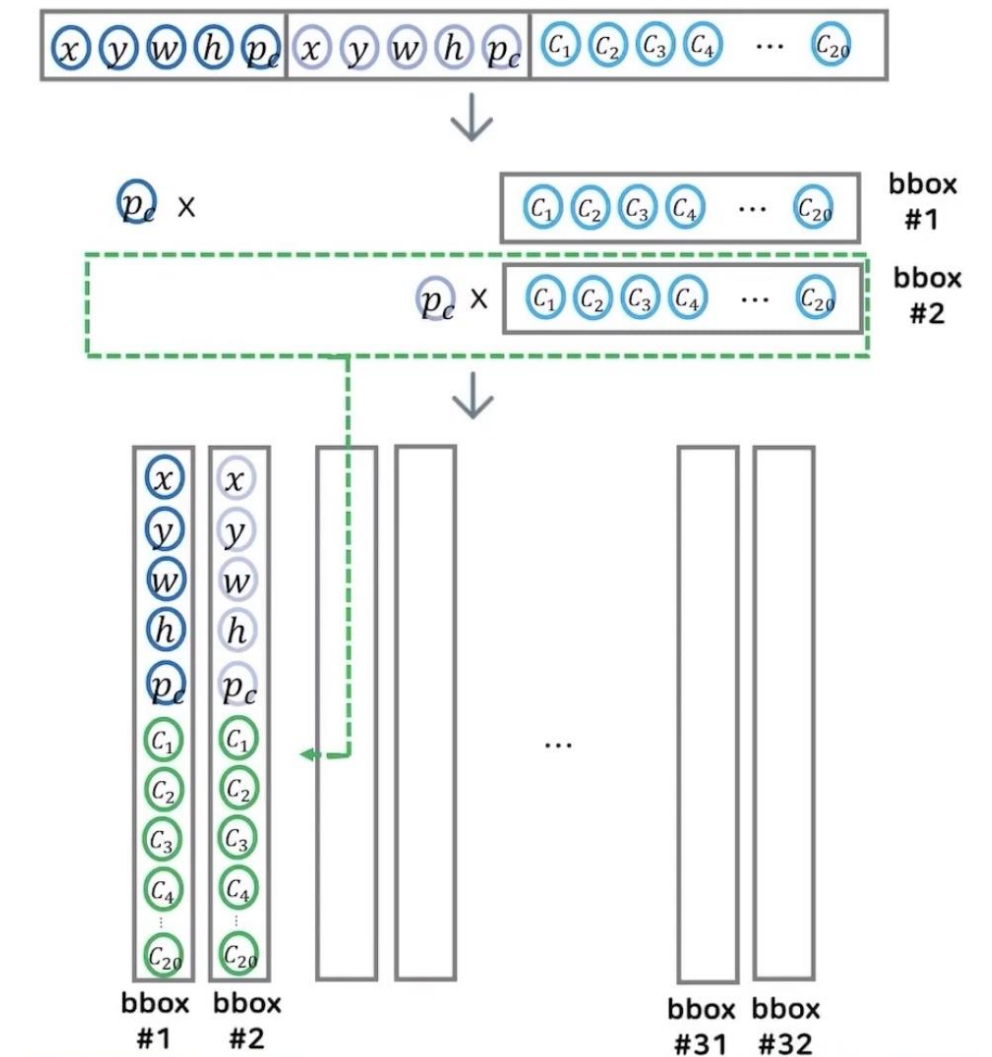
## 1. 49개의 bbox 검출 및 class 확률 예측



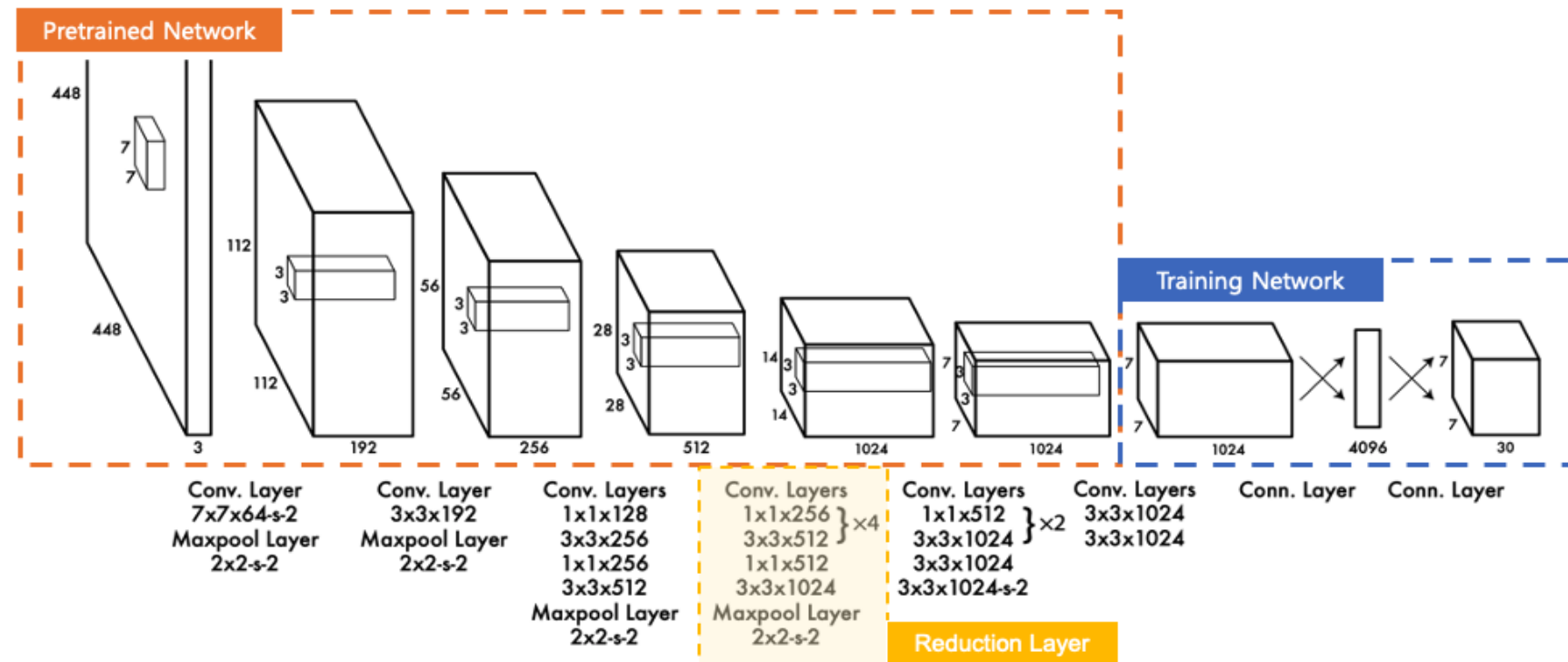
## 2. 7x7x30



## 3. class-specific confidence score 계산 + 임계값 기준 제거 + NMS



# Yolo – Network design



## <Pre-trained model>

- GoogleNet의 변형
- 24conv + 2fc layer
- 20 conv: pretrained된 layer 사용
- 4 conv + 2fc ➡ fine tuning 진행 (PASCAL VOC)
- 1x1 layer 연산량 감소
- 224\*224 ➡ 448\*448\*3 (input)

# Yolo – Loss Function

Loss

Localization Loss

Confidence Loss

Classification Loss

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

- $\mathbb{1}_{ij}^{\text{obj}}$  : Object가 존재하는 grid cell i의 predictor bounding box j
- $\mathbb{1}_{ij}^{\text{noobj}}$  : Object가 존재하지 않는 grid cell i의 bounding box j
- $\mathbb{1}_i^{\text{obj}}$  : Object가 존재하는 grid cell i

## [ Mean Squared Error ]

- (1) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, x와 y의 loss를 계산
- (2) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, w와 h의 loss를 계산
- (3) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, confidence score의 loss를 계산
- (4) Object가 존재하지 않는 grid cell i의 bounding box j에 대해, confidence score의 loss를 계산
- (5) Object가 존재하는 grid cell i에 대해, classification의 loss 계산
  - $\lambda_{\text{coord}}$  : coordinates(x,y,w,h)에 대한 loss와 다른 loss들과의 균형을 위한 balancing parameter
  - $\lambda_{\text{noobj}}$  : obj가 있는 box와 없는 box간에 균형을 위한 balancing parameter
  - 어떤 loss를 더 많이 반영할 것인가(가중치의 개념)



# 발전 모델

## YOLO v1

2016년에 발표된 최초 버전으로, 실시간 객체 검출을 위한 딥러닝 기반의 네트워크

## YOLO v3

2018년에 발표된 세 번째 버전으로, 네트워크 구조와 학습 방법을 개선하여 객체 검출의 정확도와 속도를 모두 개선

## YOLO v5

2020년 6월에 발표된 버전으로 YOLOv4와 비교하여 객체 검출 정확도에서 10% 이상 향상되었으며, 더 빠른 속도와 더 작은 모델 크기를 가짐

## YOLO v6

2022년 9월 발표된 버전으로, 여러 방법을 이용하여 알고리즘의 효율을 높이고, 특히 시스템에 탑재하기 위한 Quantization과 distillation 방식도 일부 도입하여 성능 향상

2017년에 발표된 두 번째 버전으로, 성능을 개선하고 속도를 높인 것이 특징

## YOLO v2

2020년 4월에 발표된 네 번째 버전으로, SPP와 PAN 등의 기술이 적용되어 더욱 정확한 객체 검출과 더 높은 속도를 제공

## YOLO v4

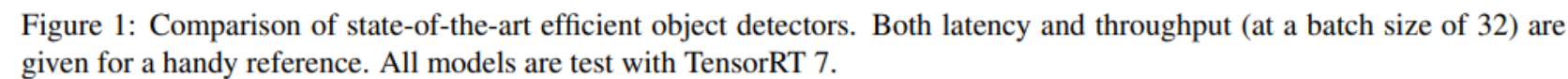
2022년 7월 발표된 버전으로, 훈련 과정의 최적화에 집중하여 훈련 cost를 강화하는 최적화된 모듈과 최적 기법인 trainable bag-of-freebies를 제안

## YOLO v7

2023년 1월 발표된 버전으로, YOLO 모델을 위한 완전히 새로운 리포지토리를 출시하여 개체 감지, 인스턴스 세분화 및 이미지 분류 모델을 train하기 위한 통합 프레임워크 로 구축됨 ➡ 공식적 논문 미출시

## YOLO v8

100%



[ Experiments/Summary ]

- YOLO v6 SOTA 달성
- APval(정확도), FPS(속도) 측면에서 YOLO v6가 훨씬 좋은 결과를 보임
- 실시간으로 쓰레기를 탐지해 수거해가는 것이 중요하기에 속도는 빠르지만 정확도 또한 높은 YOLO model을 사용하는 것이 적합하다고 판단

