Woo Ahra
BOAZ
woo.ara00@gmail.com

1 Stage Detectors

2023.02.13

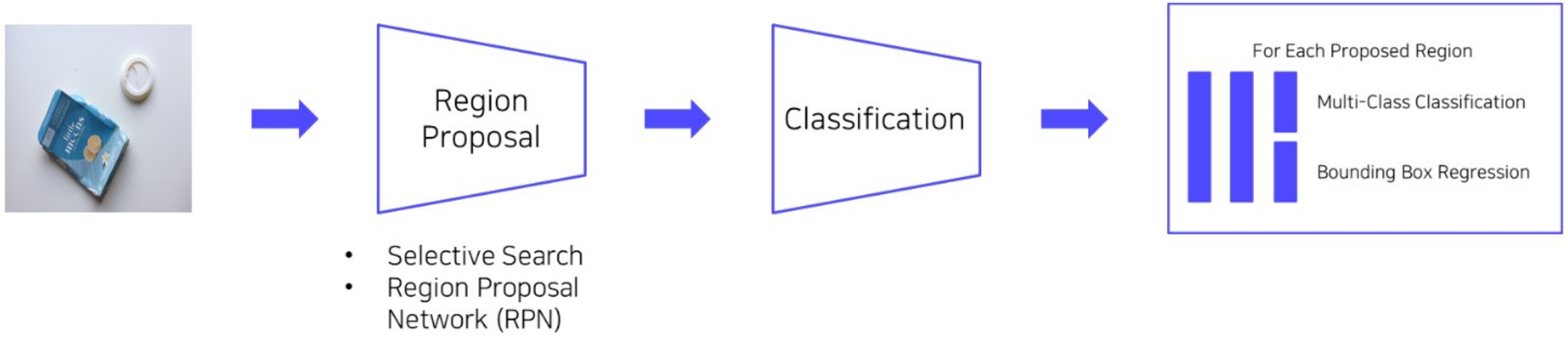
YOLO Family, SSD, RetinaNet

Table of Content

- 1. 1 Stage Detectors
- 2. YOLO
  - 2.1. Overview
  - 2.2. Pipeline
  - 2.3. Result
- 3. SSD
  - 3.1. Overview
  - 3.2. Pipeline
  - 3.3. Result
- 4. YOLO Follow-up
  - 4.1. YOLO v2
  - 4.2. YOLO v3
  - 4.3. YOLO v4
  - 4.4. YOLO v5,6,7,8
- 5. RetinaNet
  - 5.1. Overview
  - 5.2. Focal Loss

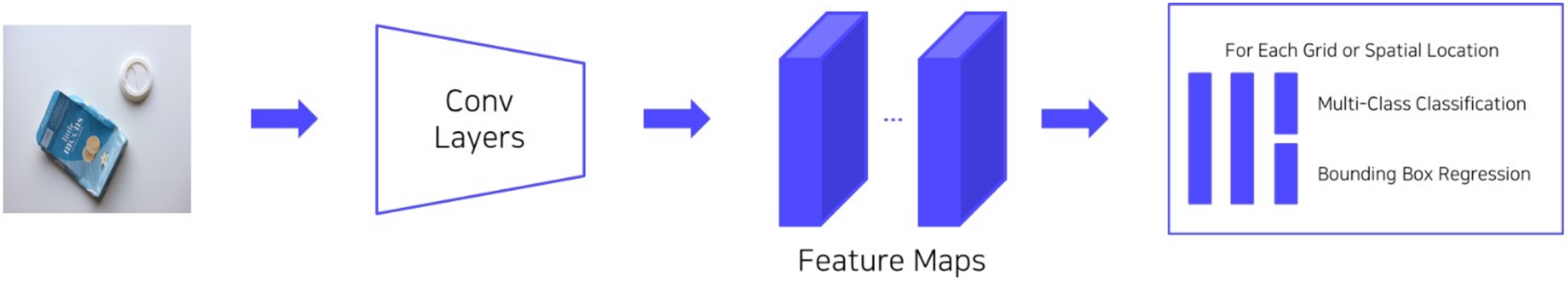
1. 1 Stage Detectors

- Background
  - 2 Stage Detectors
    - R-CNN 계열
    - 객체를 검출하는 정확도 측면에서 좋은 성능
    - 속도(FPS; Frame Per Second) 측면에서 매우 느림
    - 속도를 개선하기 위해 region proposal과 classification을 동시에 하는 1-stage detector가 제안



1. 1 Stage Detectors

- 1 Stage Detectors
  - Localization, Classification이 동시에 진행
  - 전체 이미지에 대해 특징 추출, 객체 검출이 이루어짐
  - 속도가 매우 빠름 (Real-time detection)
    - region proposal 방식 탈피 → grid 개념 도입
  - 영역을 추출하지 않고 전체 이미지를 보기 때문에 객체에 대한 맥락적 이해가 높음
    - Background error가 낮음

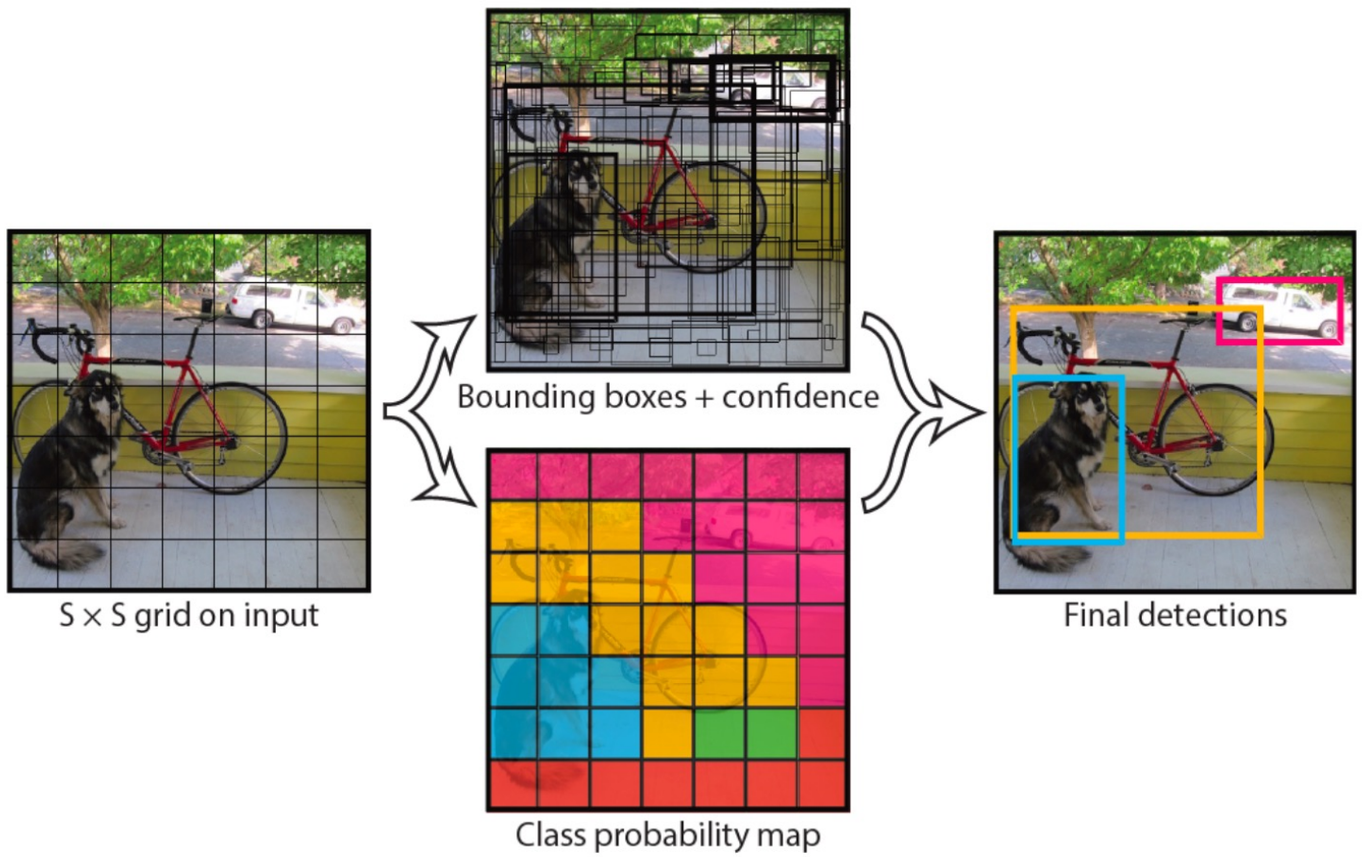


2. You Only Look Once History (YOLO)

- ❑ 2.1. Overview
- ❑ 2.2. Pipeline
- ❑ 2.3. Result

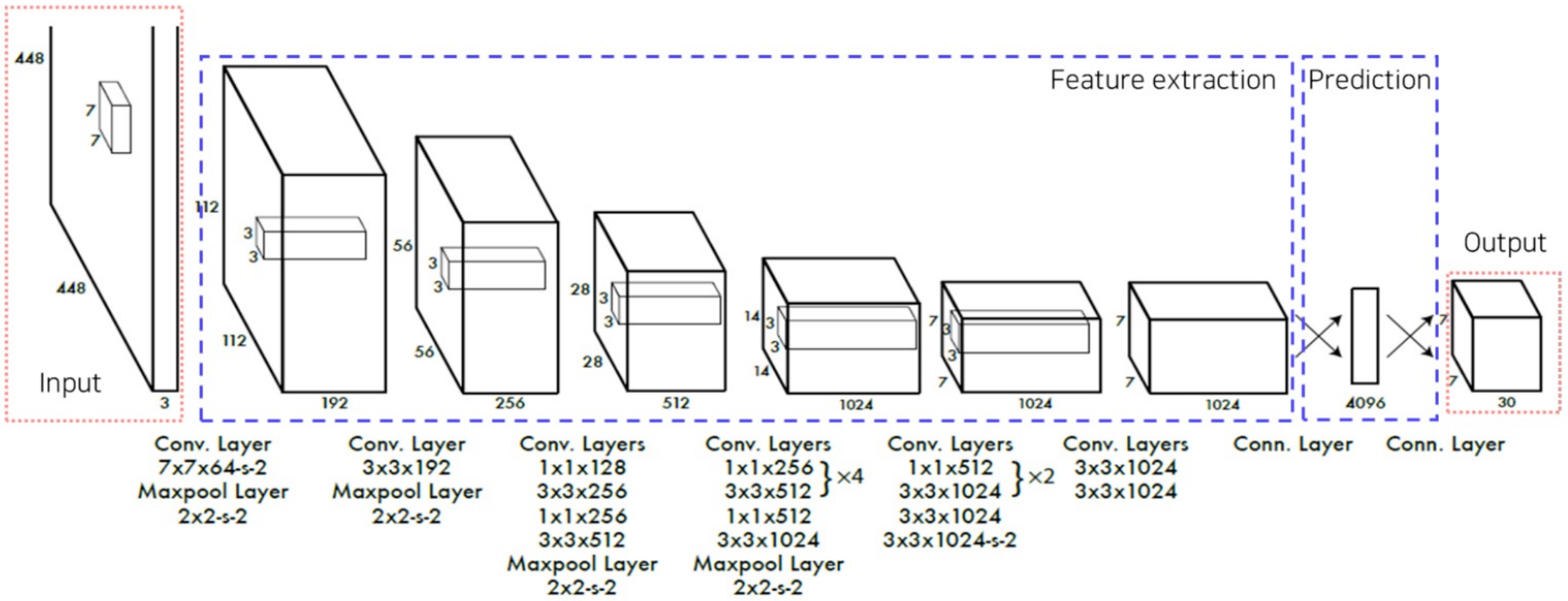
2.1. Overview

- YOLO 특징
  - Region proposal 단계 제거
  - 전체 이미지에서 bounding box예측과 클래스를 예측하는 일을 동시에 진행
  - 이미지를 전체적으로 관찰하여 Object detection을 수행하기 때문에 배경 오류가 적고 일반화 성능이 좋음
  - 성능이 낮음 (특히 small object를 잘 탐지하지 못함)



2.2. Pipeline

- GoogLeNet 변형
  - 24개의 convolution layer : 특징 추출
  - 2개의 fully connected layer : box의 좌표값 및 확률 계산

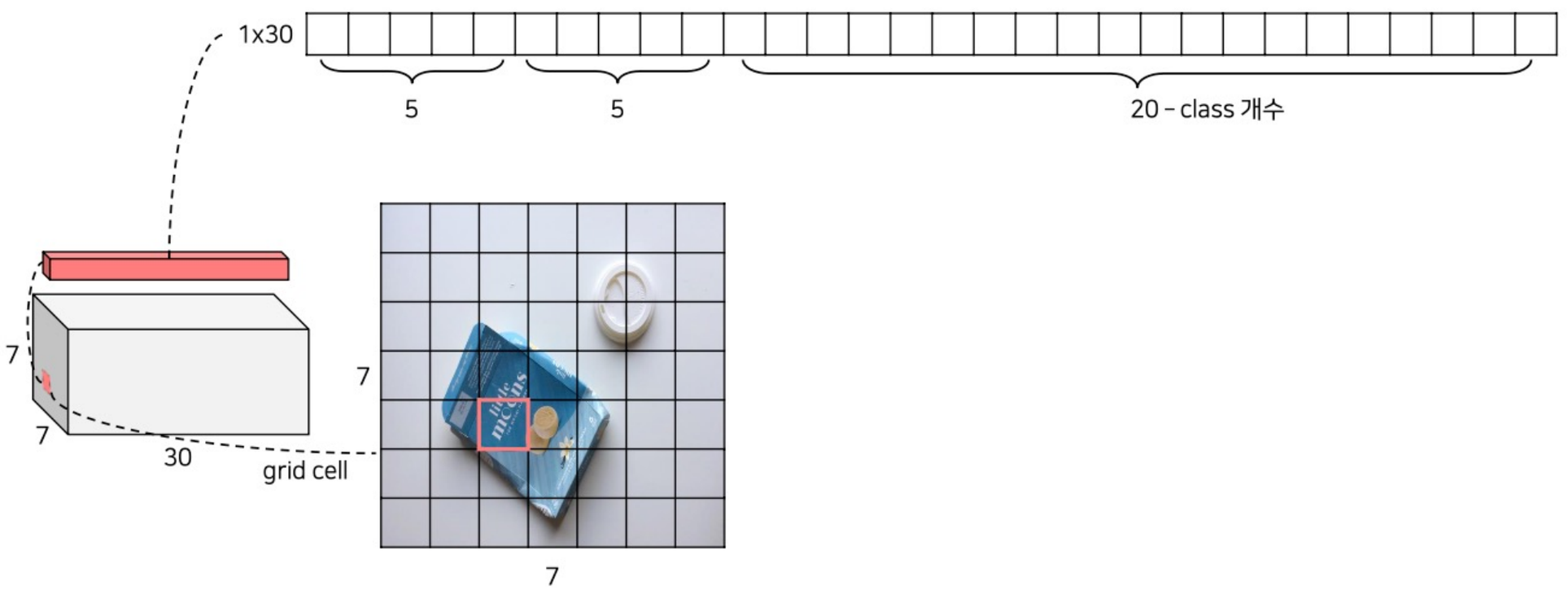


2.2. Pipeline

- 입력 이미지를 S x S 그리드 영역으로 나눔 (S=7)
- 각 그리드 영역마다 B개의 Bounding box와 Confidence score 계산 (B=2)
  - Confidence score : 해당 bbox 안에 물체가 있는지 없는지 확률로 나타낸 값
- 각 그리드 영역마다 C개의 class에 대한 해당 클래스일 확률 계산 (C=20)

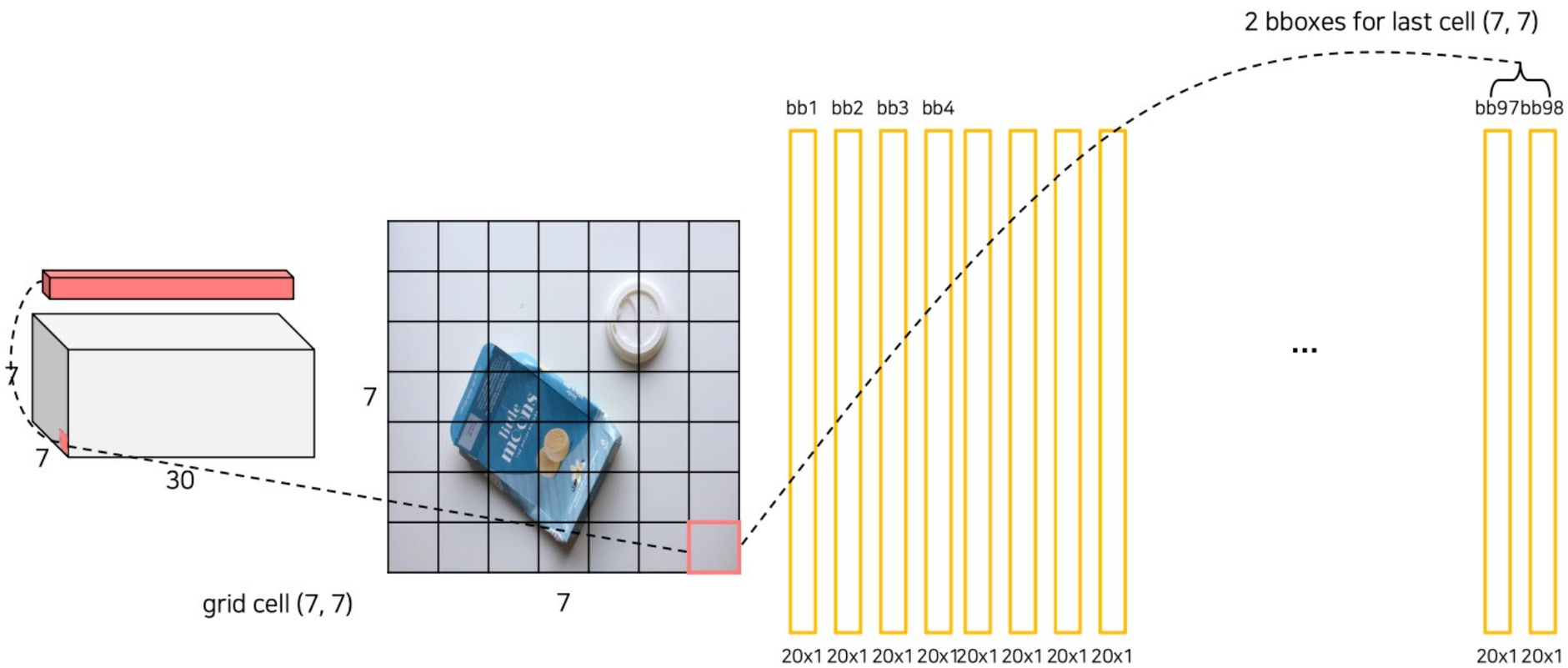
2.2. Pipeline

- output = S x S x {(c, x, y, w, h) * B + C}



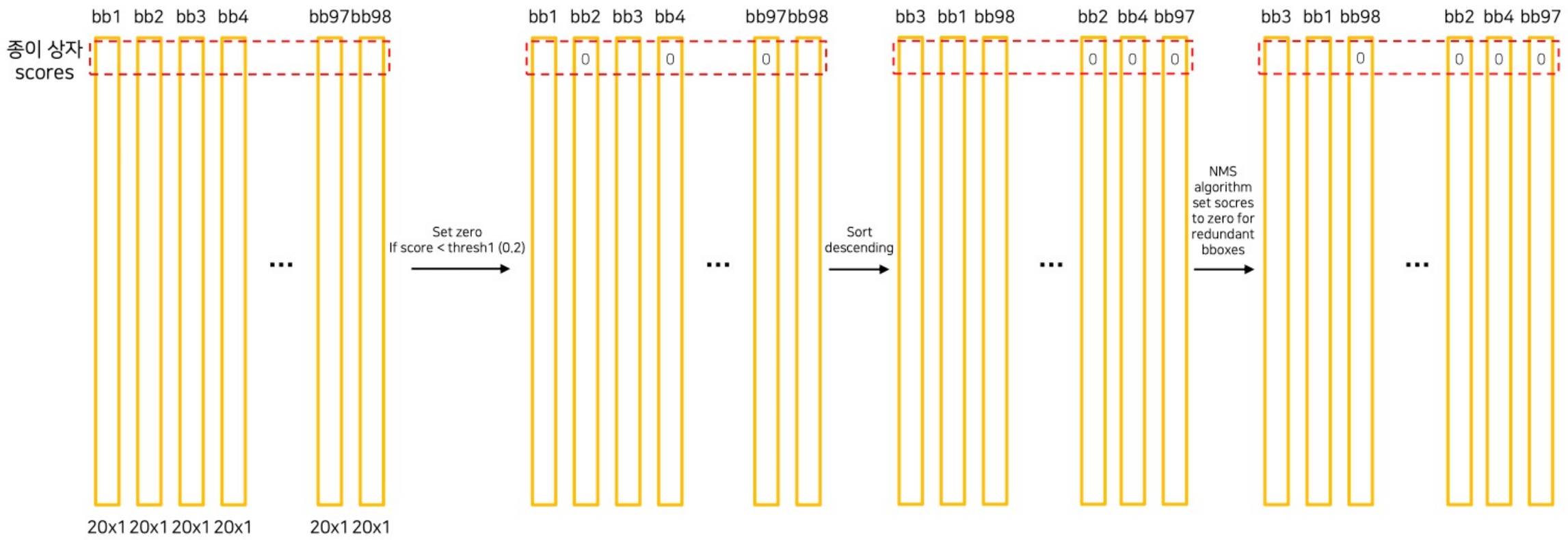
2.2. Pipeline

- output = S x S x {(c, x, y, w, h) * B + C}
- bbox 개수 : 7 x 7 x 2 = 98



2.2. Pipeline

- output = S x S x {(c, x, y, w, h) * B + C}
- bbox 개수 : 7 x 7 x 2 = 98



+) NMS : 여러 개의 bbox가 겹쳐 있는 경우 가장 신뢰도가 높은 하나의 bbox만 남기는 후처리 과정

2.2. Pipeline

- Loss

Localization Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence Loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification Loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

2.3. Result

❑ 성능

- Faster R-CNN에 비해 속도가 빠름
- 다른 real-time detector에 비해 정확도가 높음

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

❑ 한계

- 그리드보다 작은 크기의 물체 검출 불가능
- 신경망을 통과하며 마지막 feature만 사용 → 정확도 하락

3. Single Shot MultiBox Detector (SSD)

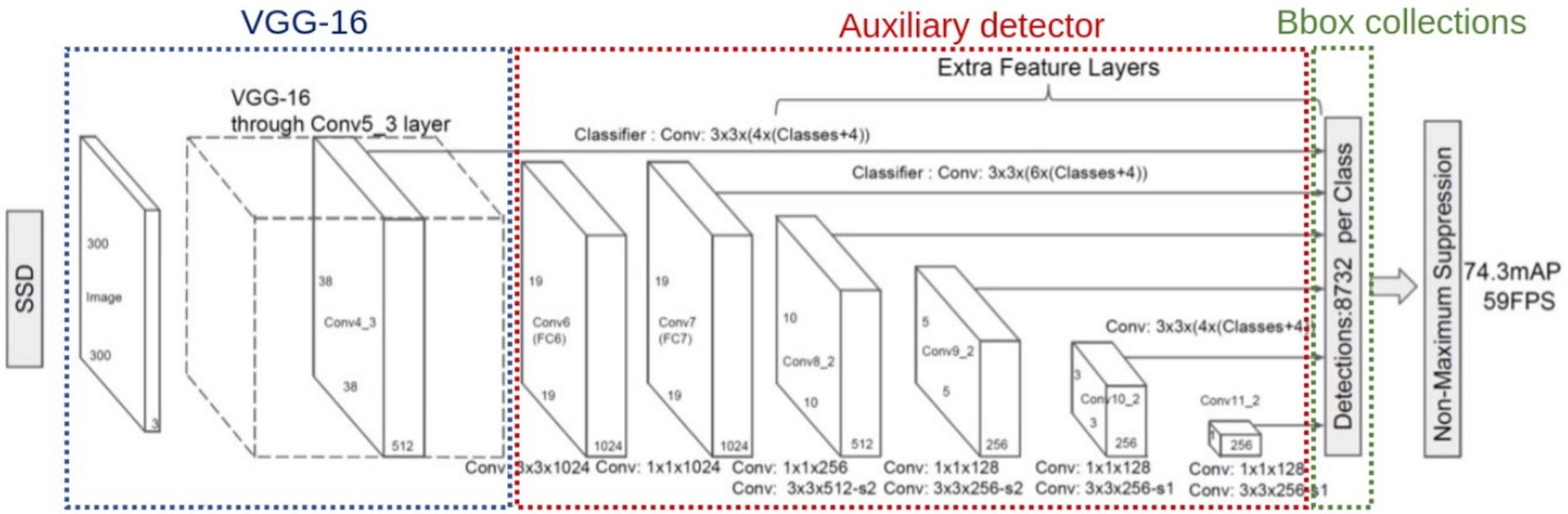
- ❑ 3.1. Overview
- ❑ 3.2. Pipeline
- ❑ 3.3. Result

3.1. Overview

- SSD 특징
  - backbone : VGG-16
  - Extra convolution layers에 나온 feature map들 모두 detection 수행
  - Fully connected layer 대신 convolution layer 사용하여 속도 향상
  - Default box 사용 (anchor box)
    - 서로 다른 scale과 비율을 가진 미리 계산된 box 사용

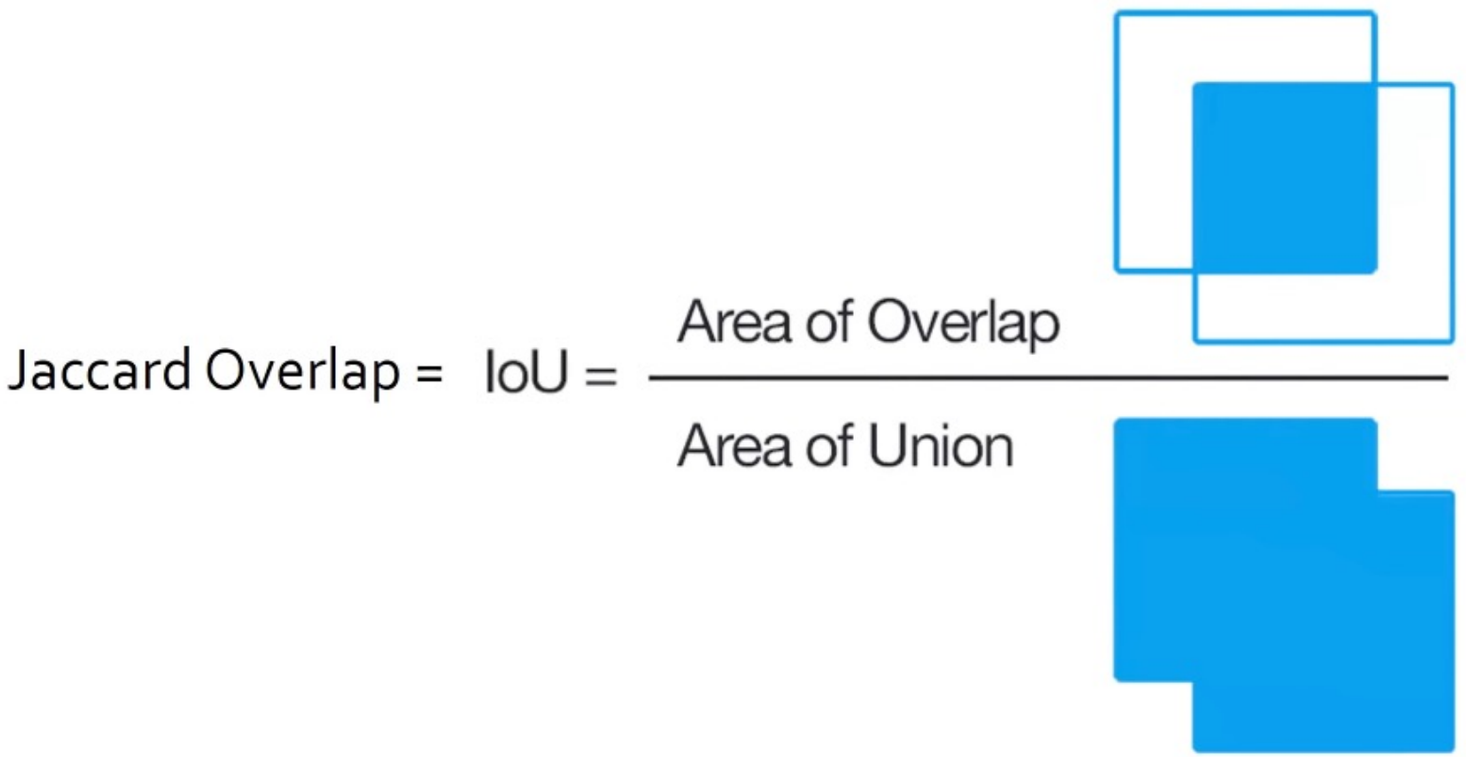
3.2. Pipeline

- SSD Network
  - VGG-16(Backbone) + Extra Convolution Layers
  - output : n x n x [default box x {(num classes + background) + (cx, cy, w, h)}]
    = n x n x {default box x (Classes + 4)}



3.2. Pipeline

- Training – Matching Strategy
  - IoU가 0.5이상 되는 bounding box를 찾음



3.2. Pipeline

- Training - Loss
  - anchor box에서 ground truth box로 얼마나 가야 하는지 학습

Loss Function

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

Localization loss (Smooth L1)

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

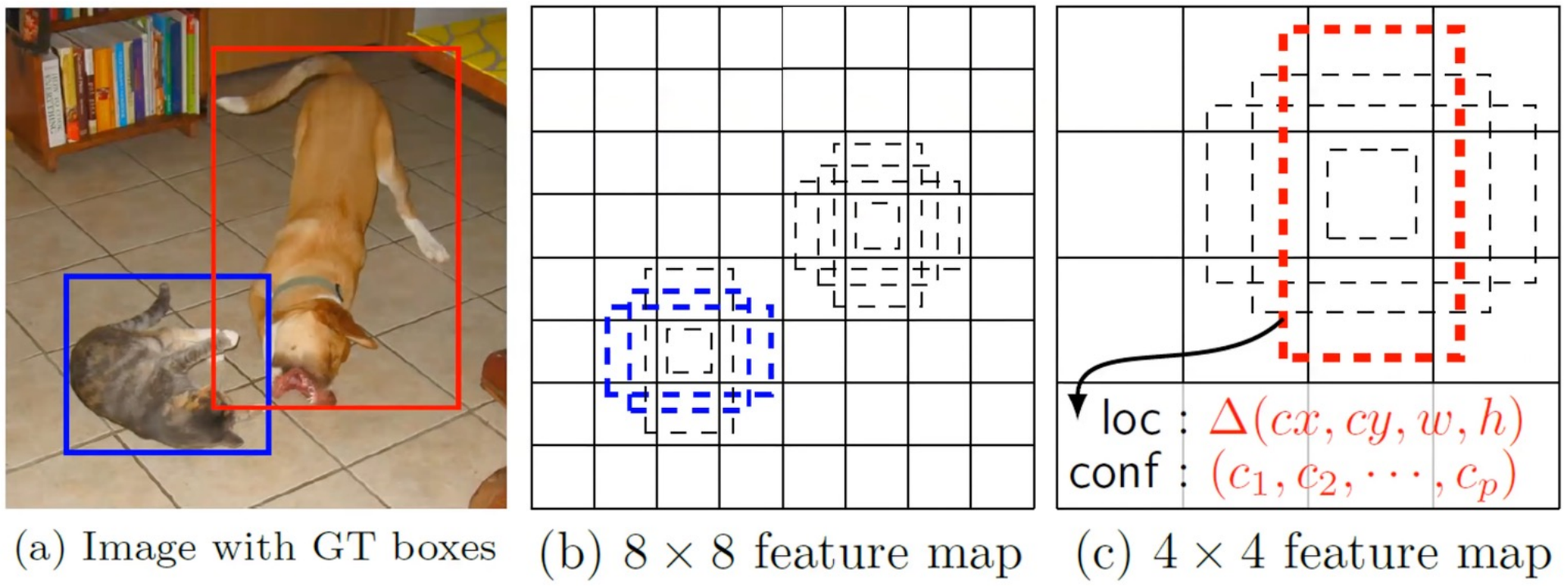
$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

Confidence loss (Softmax)

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

3.2. Pipeline

- Choosing Scales and Aspect Ratios for Default Boxes
  - 큰 feature map (early stage feature map) → 작은 물체 탐지
  - 작은 feature map (late stage feature map) → 큰 물체 탐지



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

3.3. Result

- Inference Time

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

4. YOLO Follow-up

- ❑ 4.1. YOLO v2
- ❑ 4.2. YOLO v3
- ❑ 4.3. YOLO v4
- ❑ 4.4. YOLO v5,6,7,8

4.1. YOLO v2

- Concepts
  - Better : 정확도 향상
  - Faster : 속도 향상
  - Stronger : 더 많은 class 예측 (80 → 9000)

4.1. YOLO v2

- Better
  - Batch Normalization을 모든 layer에 적용
    - regularization 효과를 얻어 drop out 제거
  - High Resolution Classifier
    - YOLO v1은 detection task의 절반의 해상도로 classifier을 학습하는데 반해, YOLO v2는 같은 해상도로 학습
  - Convolutional Anchor Boxes
    - FC layer 삭제, fully convolution 구조 사용
    - anchor box를 활용해 경계 상자 예측
  - Dimension Clusters
    - 사람이 미리 정의한 anchor box를 사용하지 않고 데이터에 맞는 anchor box 사용
    - 데이터에 존재하는 ground truth box를 이용하여 clustering을 실시
  - Direct Location Prediction
    - (x,y)를 특정 grid cell 안으로 한정하여 학습 초반 random initialization으로 인한 학습 불안정성 예방
  - Fine-Grained Features
    - 작은 물체를 잘 탐지하기 위해 더 높은 해상도를 가진 이전 단계 layer을 가져와 detection을 위한 output feature map에 concatenate하여 사용
  - Multi-Scale Training
    - 매 10 batch마다 input image 크기를 바꿔가며 모델 학습

4.1. YOLO v2

- Faster
  - Dark Net
    - VGG16 기반 detection frameworks는 과도하게 복잡함
    - Dark Net 이라는 새로운 네트워크 구조 제안

4.1. YOLO v2

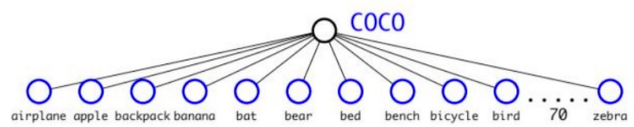
- Result

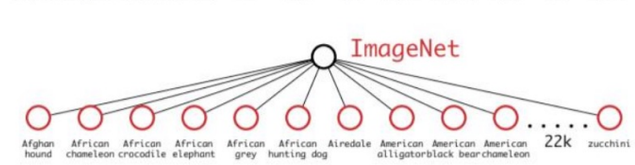
	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

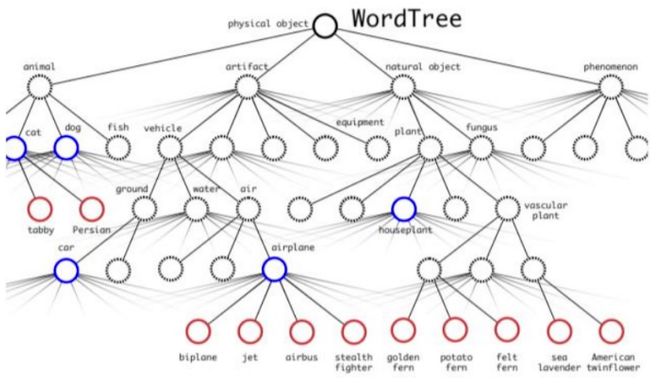
4.1. YOLO v2

- Stronger
  - Classification 데이터셋(ImageNet), detection 데이터셋(Coco) 함께 사용
    - Detection 데이터셋 : 일반적인 객체 class로 분류
    - Classification 데이터셋 : 세부적인 객체 class로 분류

  - WordTree 구성







- ImageNet 데이터셋 : Coco 데이터셋 = 4:1
  - Detection 이미지 : classification loss는 특정 범주에 대해서만 loss 계산
  - Classification 이미지 : classification loss만 역전파 수행 (IoU)

4.2. YOLO v3

❑ Concept

- ■ 조금 느리지만 성능은 더 높은 모델을 만들고자 함

❑ 특징

- ■ Bounding Box & Class Prediction
  - ▪ 각 class에 대해 독립적으로 logistic regression을 적용
  - ▪ 상호 연관된 class에 대해 더 잘 예측하게 되고, 하나의 object에 대해 여러 class를 예측

- ■ Predictions Across Scales
  - ▪ 3개의 다른 크기의 feature에 대해 각각 prediction을 진행
  - ▪ upsampling과 이전 단계의 해상도를 가져와 concatnation

- ■ DarkNet-53
  - ▪ DarkNet-19를 기반으로 크기를 늘려 사용
  - ▪ DarkNet-19 보다는 무겁지만 ResNet-101 or 152 보다는 가벼움

❑ Result

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

4.3. YOLO v4

❑ YOLO v4 : Optimal Speed and Accuracy of Object Detection

❑ 최신 딥러닝 기법 적용
- WRC (Weighted-Residual-Connections)
- CSP (Cross-Stage-Partial-Connections)
- CmBN (Cross mini-Batch Normalizations)
- SAT (Self-Adversarial-Training)
- Mish Activation
- Mosaic Data Agumentation
- Drop Block Regularization
- CIOU Loss

❑ 목적
- 일반적인 학습 환경에서도 높은 정확도와 빠른 학습
- detector를 학습하는 동안, 최신 BOF, BOS 기법이 성능에 미치는 영향 증명
- CBN, PAN, SAM을 포함한 기법을 활용하여 single GPU training에 효과적임

4.4. YOLO v5,6,7,8

- YOLO v5 : https://github.com/ultralytics/yolov5
  - Focus() → Conv Layer
  - SPP() → SPPF()
  - BottleNeckCSP() → C3()
  - 정확도를 유지시키면서 연산량을 줄여 성능을 높임

- YOLO v6 : https://arxiv.org/abs/2301.05586
  - Head 변화 : 3개 Scale Detection → 4개 Scale Detection

- YOLO v7 : https://arxiv.org/abs/2207.02696
  - trainable bag-of-freebies : inference cost를 증가시키지 않고 정확도를 향상시킬 수 있는 방법 제안
  - 5~160 FPS 범위의 속도와 정확도 측면에서 현재까지 나온 모든 Object Detector의 성능을 능가

- YOLO v8 : https://github.com/ultralytics/ultralytics

5. RetinaNet

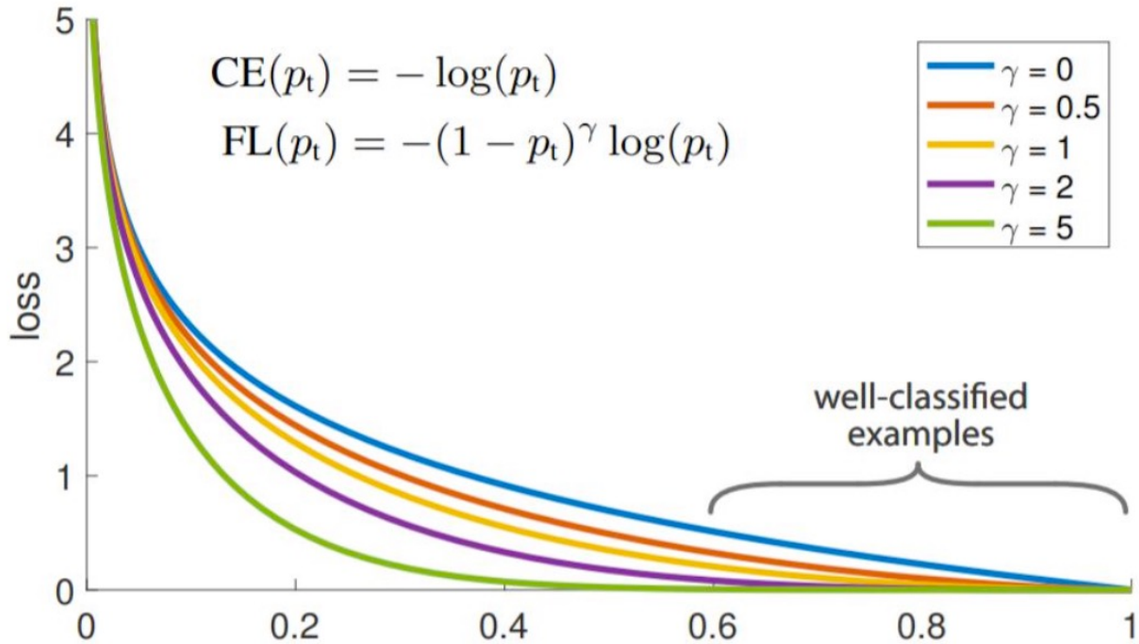
- ❑ 5.1. Overview
- ❑ 5.2. Focal Loss

5.1. Overview

- 1 Stage Detector Problems
  - Class imbalance
  - Anchor Box 대부분 Negative Samples (background)

5.2. Focal Loss

- Concept
  - 새로운 function 제시 : cross entropy loss + scaling factor
  - 쉬운 예제에 작은 가중치, 어려운 예제에 큰 가중치



- Result
  - One-stage methods의 단점이었던 성능 면에서 큰 향상을 이룸
  - Object Detection에서 background와의 class imbalance 조정

Thank you