

# week 6

▼ 논문리뷰 \_ You Only Look Once : unified, real-time object detection

## 사전지식

▼ Object Detection



object classification : 이미지 내의 존재하는 물체의 class를 class probability로 표현

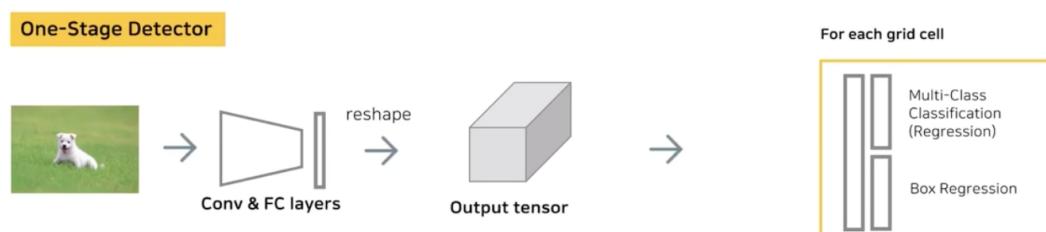
object localization : 이미지 내의 물체를 확인 뿐만아니라 물체의 위치를 bounding box로 표현

object detection: multiple objcet(다중 객체)를 감지

방식 )

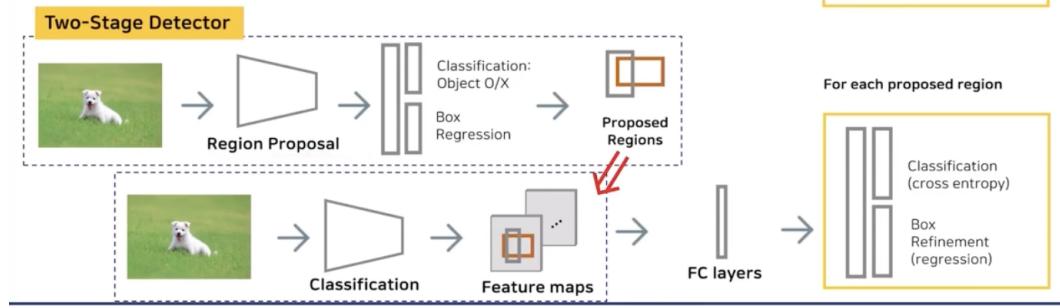
▼ one-stage detector \_ (YOLO)

: localization, classification을 동시에 수행해 결과를 얻음. 이미지 내 모든 위치를 존재할 수 있는 위치로 보고 각 후보영역에 대해 class를 예측함



▼ two-stage detector \_ (R-CNN)

: localization을 먼저 실행후, classification을 실행해 결과를 얻음. region proposal(후보 위치들을 제안) 후, 클래스를 예측함

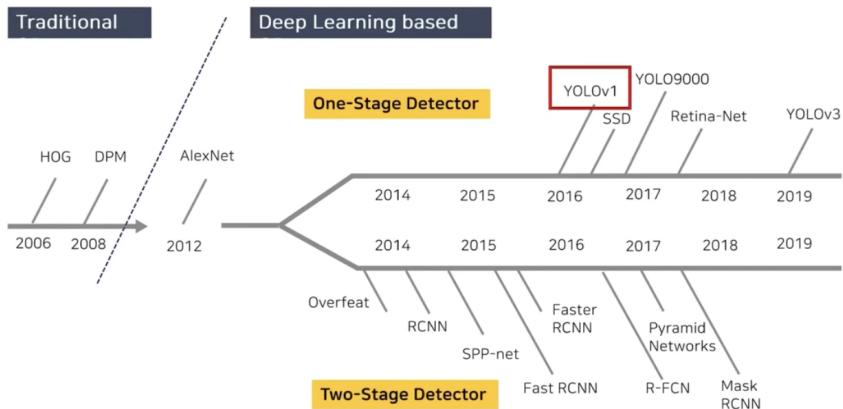


이미지 입력 → region proposal → proposed regions(가장 물체가 있을법한 위치)

이미지 입력 → classification → feature maps(물체의 특징을 뽑아냄)

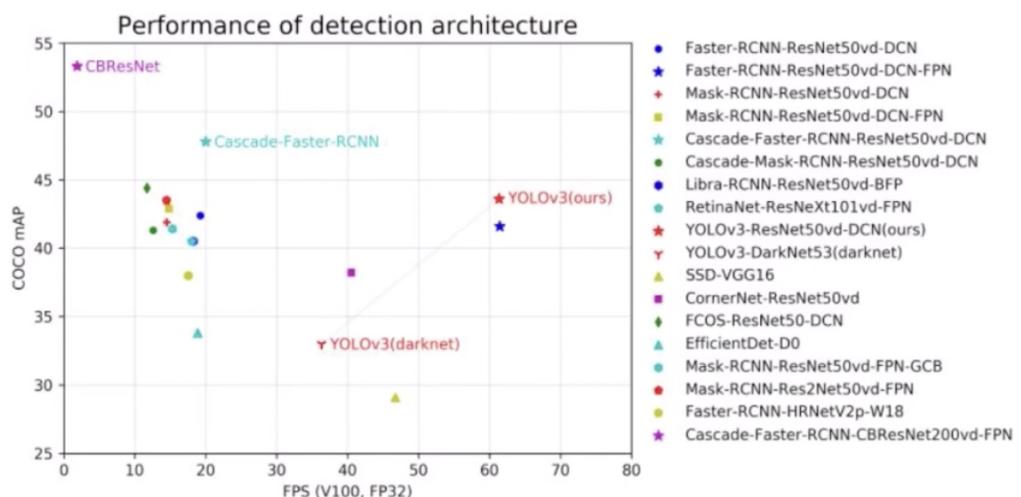
feature maps + proposed regions → fc layer → 클래스와 위치

### ▼ 역사 )



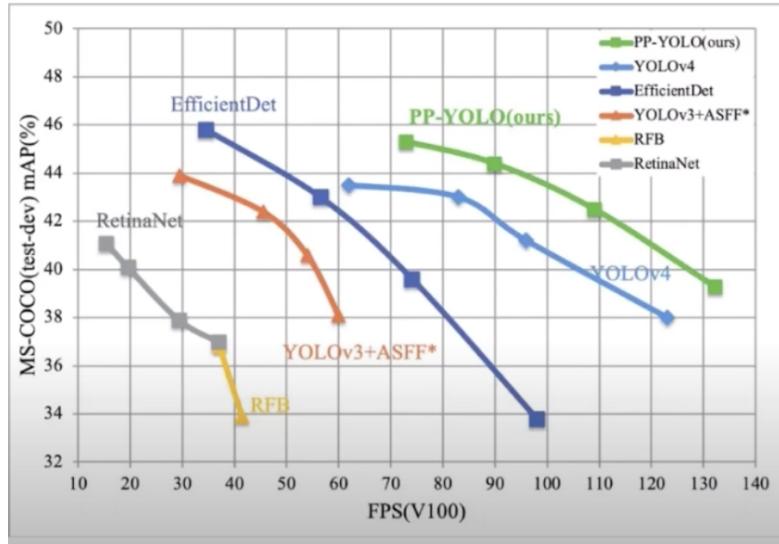
방식에 따라 나오는 모델이 다름

### ▼ 성능 )



→ YOLO : 속도는 빠르지만 성능은 좋지 않음 / RCNN : 속도는 느리지만 성능이 좋음

▼ YOLO 알고리즘 역사 )



YOLO v1 → YOLO9000 → YOLO v3 → YOLO v4 → YOLO v5 → PP-YOLO

## 주요 내용

▼ main contribution

- 제목의 의미

You only look once : 전체 이미지 보는 횟수가 1번

unified : classification, localization 단일화

real-time : 속도 개선

- 핵심 내용

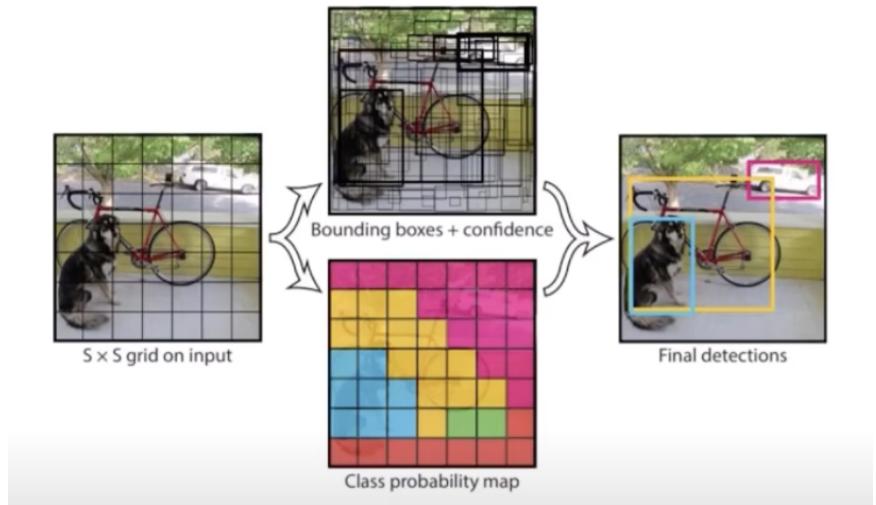
object detection을 regression problem 관점으로 전환

unified architecuture : 하나의 신경망으로 classification, localization 예측

DPM, RCNN모델보다 속도가 개선됨

여러 도메인에서 object detection 가능해짐

▼ Unified Detection



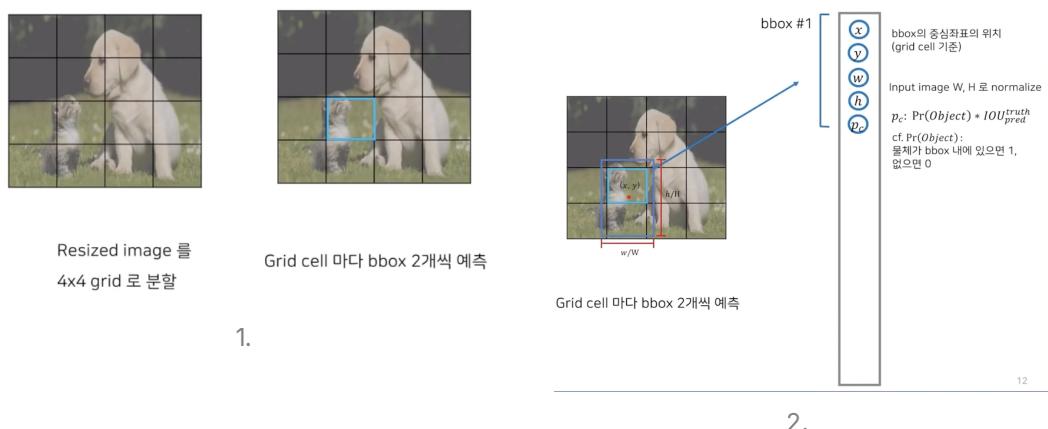
region proposal, feature extraction, classification, bbox regression 단계들을 one-stage detection으로 통합

이미지 전체로 얻은 feature map을 활용해 bbox 예측 및 모든 클래스에 대한 확률을 계산

$S \times S$  grid cell  $\rightarrow$  B bbox prediction + confidence / class probabilities  $\rightarrow$   
 $S \times S \times (B * 5 + 5)$

#### ▼ ex

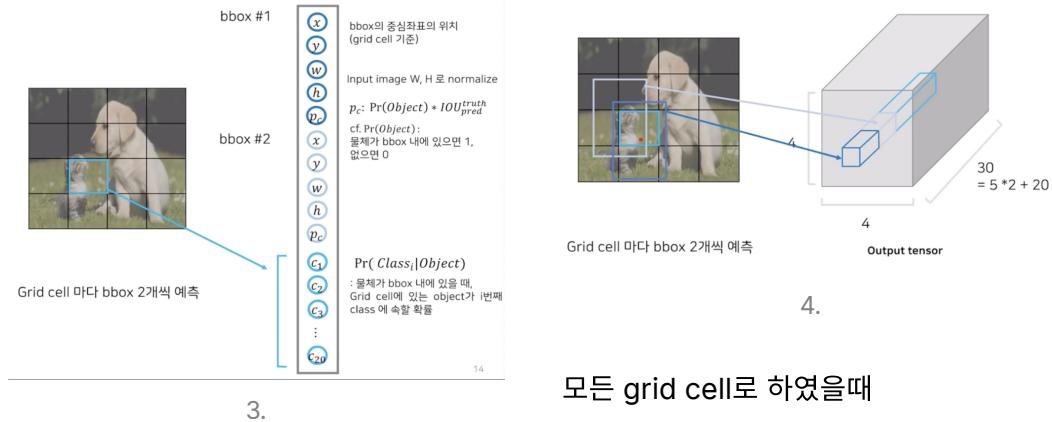
$$s = 4, B(\text{bounding box}) = 2, C(\text{class}) = 20$$



$(x, y)$  - bbox의 중심좌표

$(w, h)$  - 전체 입력이미지의 W, H로 normalize한 값 (0~1)

$P_c$  -  $Pr(\text{object})$  물체가 bbox내에 있으면 1, 없으면 0



$\Pr(\text{class} | \text{object})$  : 물체가 bbox에 있을 때, grid cell에 있는 object가 i번째 class에 속할 확률

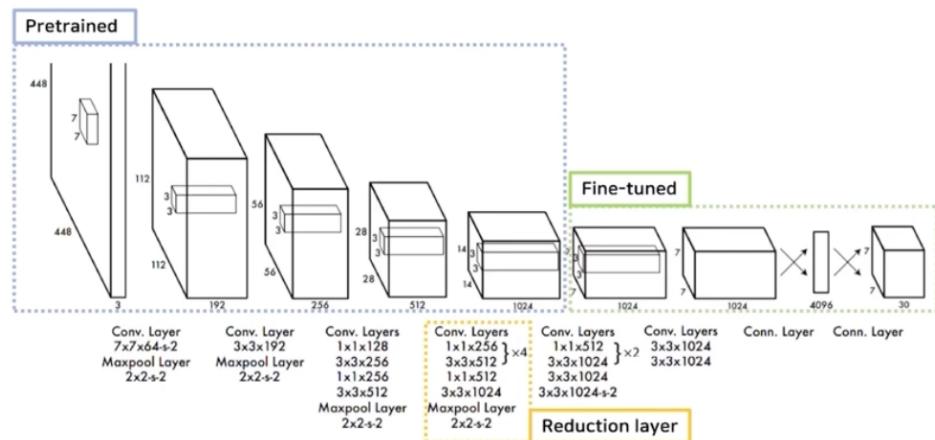
모든 grid cell로 하였을 때

4x4 - grid 분할

$$30 = 5(\text{output 수}) \times 2(\text{bbox}) + 20(\text{class})$$

## ▼ Network design

- GoogLeNet



GoogLeNet : 24 conv + 2 fc / Fast YOLO : 9 conv + 2 fc

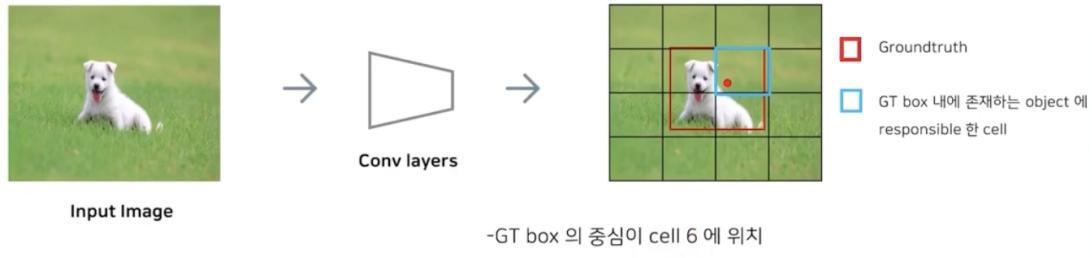
24 conv = 20 conv : 사전에 학습된 1000 class ImageNet (  $224 \times 224$  )

+ (4 conv + 2 fc) : 추가로 학습된 PASCAL VOC (  $448 \times 448$  )

중간에  $1 \times 1$  reduction layer로 연산량을 감소시킴

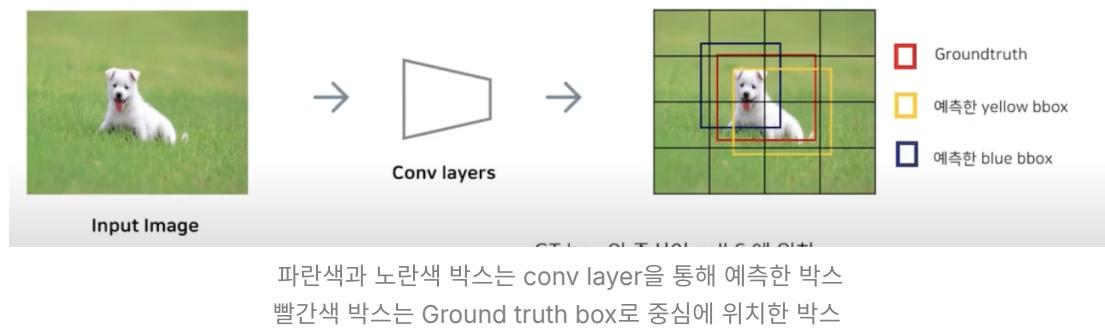
## ▼ Training

특정 object에 적합한 cell은 GT box의 중심에 위치하는 cell로 함 ⇒ 빨간색 박스



YOLO는 여러 bbox를 예측하지만, 학습단계에서는 1개의 bbox만 사용

→ 선정방법 : IOU(truth predict)값이 가장 높은 bounding box 하나만을 사용 (=GTbox와 가장 많이 겹치는 box 하나를 선정)



$$IOU_{yellow \ bbox}^{groundtruth} > IOU_{blue \ bbox}^{groundtruth}, 1_{6,yellow}^{obj} = 1$$

예제에서는 파란색 박스보다 노란색박스가 빨간색 박스와 더 많이 겹치므로 노란색 박스만을 학습에 사용

노란색 박스의 스칼라값만 1로 하고, 나머지는 0으로 하여 loss function에 반영

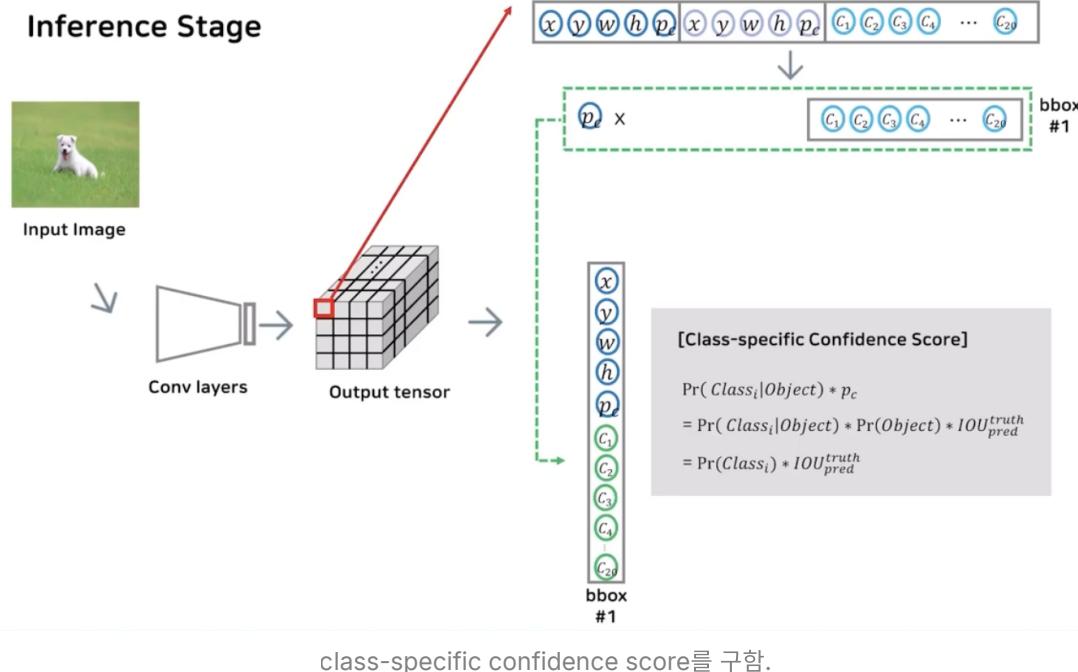
#### ▼ loss function : MSE(mean squared error)

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

- 모든 grid cell에서 예측한 “b개의 bbox의 좌표와 GTbox 좌표”의 MSE
- 모든 grid cell에서 예측한 “b개의 Pr(class | Object)와 GT 값”의 MSE
- 모든 grid cell의 “Pr(object) \* IOU예측값과 GT box 값”의 MSE

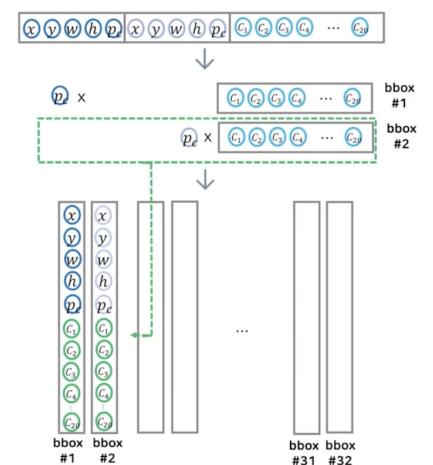
- $\mathbb{1}_{ij}^{obj}$  : if jth bbox predictor in cell i is responsible for prediction
- $\mathbb{1}_i^{obj}$  : object appears in cell i
- $\lambda_{coord} = 5$  : bbox coordinates loss 반영 ↑
- $\lambda_{noobj} = 0.5$  : no object 의 class probability loss 반영 ↓

grid cell에 object가 존재하는 경우의 오차와 predictor box로 선정된 경우의 오차만 학습



bonding score의 confidence score 와 classification을 곱해서 bonding box마다 class-specific confidence score를 구함

예측한 bonding box개수만큼 구함



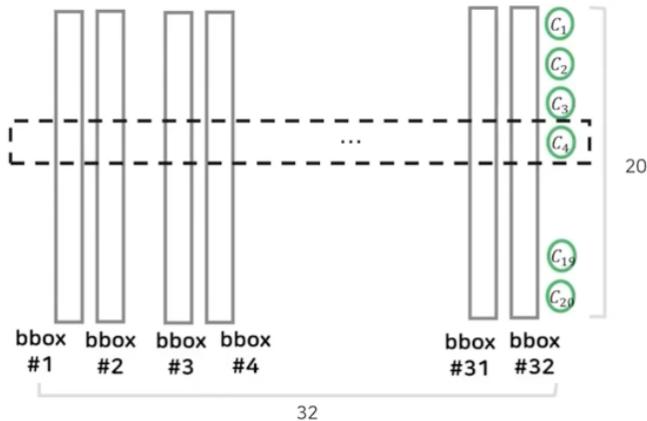
single network로 detection이 가능함.

(PASCAL VOC data기준) 이미지 1개당 32개 bbox생성. 각 클래스에대한 예측값 (grid cell :  $4 \times 4$ )

⇒ object 당 bbox 개수가 많으므로 NMS 적용 필요

▼ NMS(Non-Maximum Suppression)

: 각 object에 대해 예측한 여러 bbox 중에 가장 예측력이 좋은 bbox만 남김



각각의 클래스마다 가장 예측력이 좋은 bbox를 남겨야함

▼ object 1개인 경우

예시)

0.9	0.88	0.75	0.6	...	0.0	0.0
bbox #12	bbox #13	bbox #16	bbox #17		bbox #2	bbox #1

강아지

Red box highlights the first row of the table. A red arrow points from the first cell (0.9) to a small image of a dog with a red bounding box around it. A blue circle labeled "강아지" (dog) is positioned next to the last cell (0.0).

- class: 강아지
- confidence: 0.9
- bbox: #12

class-specific confidence score값이 높은 순서대로 정렬함

→ 특정 값을 넘지 못하는 bbox는 제거함 (bbox1, 2)

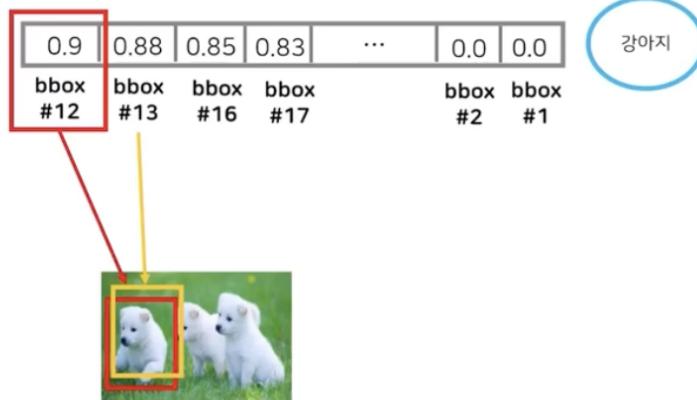
→ 가장 높은 값의 bbox 12를 선택함

→ 그 옆에 있는 bbox 13와의 IOU값을 구함 - 높은 IOU값을 같게됨 (물체가 하나밖에 없는 경우이기에..)

→ 나머지 bbox도 bbox 12와의 IOU값이 높기에 NMS에 의해 모두 제거됨

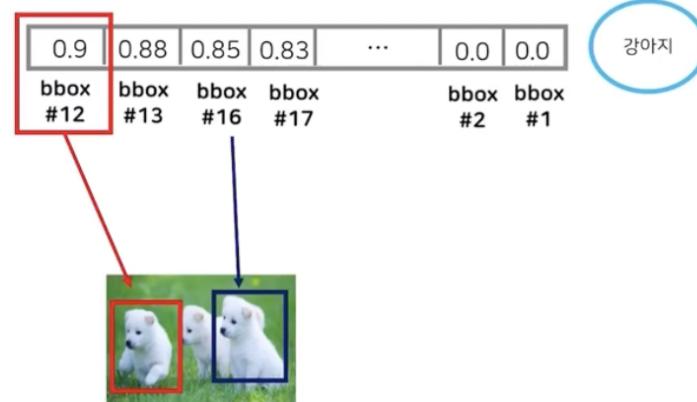
▼ 같은 class에 속하는 object 2개인 경우

예시)



bbox 12와 bbox 13의 IOU값이 높으므로 NMS에 의해 제거됨 (= 같은 객체를 가리키고 있다는 의미)

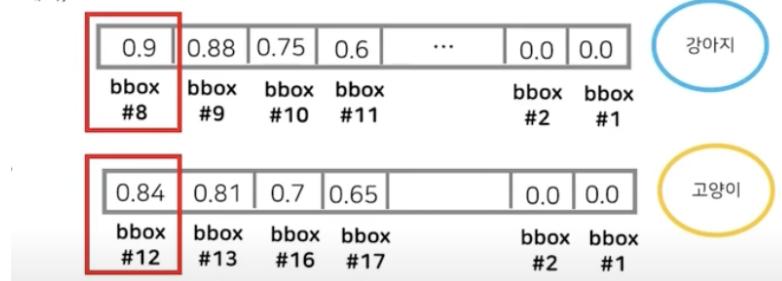
예시)



bbox 12와 bbox 16의 IOU값은 0 즉, 낮으므로 제거되지않음 (= 다른 객체를 가리키고 있음)

▼ 다른 class에 속하는 object 여려개인 경우 - 실제 가장 흔함

예시)





클래스별로 수행하기에 강아지와 고양이 따로 수행됨

강아지 ) bbox 8과의 IOU값이 높은 bbox는 제거함

고양이 ) bbox 12와의 IOU값이 높은 bbox는 제거함

⇒ 최종적으로, 가장 예측력이 좋은 bbox 8 과 bbox 12만 남게됨

#### ▼ pseudo code

```

for i in range(len(class)):
    if bbox['p_c'] < θ:
        bbox_list.remove(bbox)

    while (not processed bbox exists):
        selected_bbox = bbox with the highest p_c
        bbox_list.remove(other boxes which has high IOU with selected bbox)
    
```

#### ▼ Experiment

##### 1. 속도 및 성능 비교 실험

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	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
<hr/>			
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

- 속도

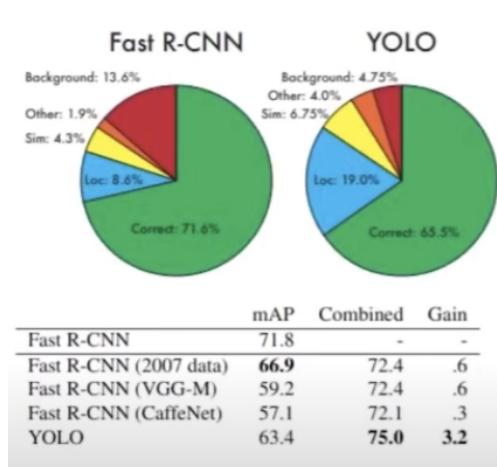
Fast YOLO > YOLO > DPM,  
RCNN

- 성능

Faster-RCNN > Fast-RCNN  
> YOLO > DPM

+ ) 작은물체, 특정 카테고리의  
object에서 mAP가 RCNN보다 낮  
음

##### 2. 오류율 분석



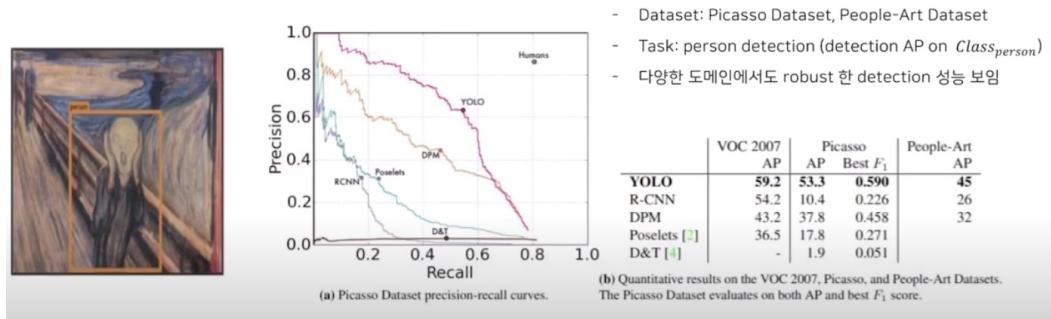
- Fast RCNN보다 YOLO의 오류율은 감소

→ false positive를 감소시킴  
(background에 없는데 있다고하는 경우 들을 없앰)

- Fast RCNN + YOLO ⇒ mAP 3.2% 향상

( 다른 모델을 양상불해서 얻은 효과가 아니다! )

### 3. 다양한 도메인



사람만을 찾는 것이기에 mAP가 아닌 AP지표를 사용

이미지 내의 사람을 찾아내는 실험을 진행

YOLO는 VOC, picasso AP, people-art AP 모두 비슷한 성능을 도출해냄

반면, R-CNN, DPM 등은 VOC에서의 성능에비해 picasso AP, people-art AP가 낮게 나옴

#### ▼ limitation

##### 1. 작은 물체에 대해 탐지 성능이 낮음

이유 : object가 크면 bbox간의 IOU값이 커져 적절한 predictor를 선택할 수 있음

하지만, object가 작으면 bbox간의 IOU값의 차이가 작아서 근소한 차이로 predictor가 결정됨

##### 2. 일반화 된 지식이랑 다르게 object 비율이 달라지면 detection 성능이 낮아짐

→ 대부분 detection 모델의 한계점

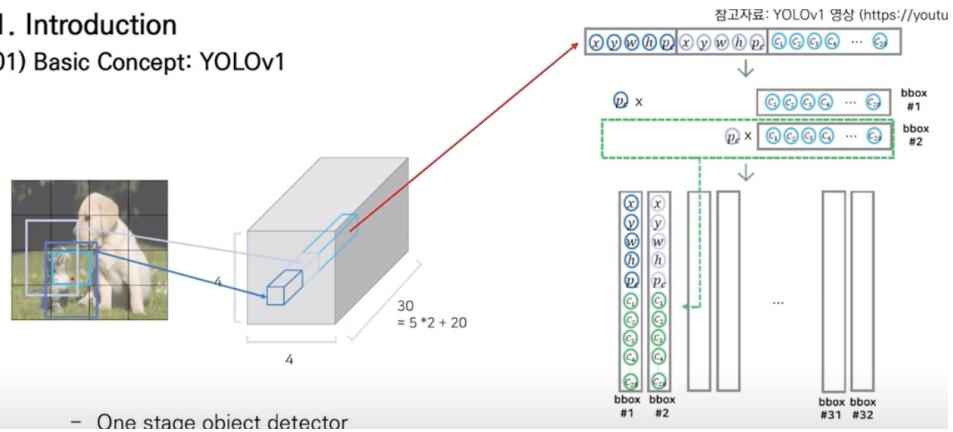
#### ▼ 논문리뷰 \_ YOLO9000 : Better, Faster, Stronger

## 사전지식

## ▼ YOLO v1

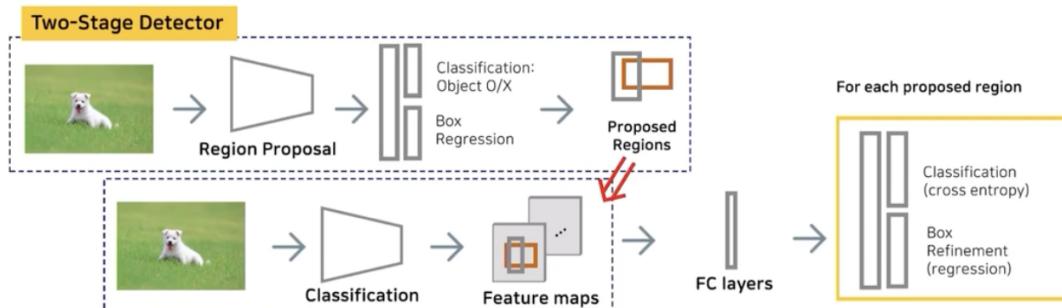
### 1. Introduction

#### 01) Basic Concept: YOLOv1



- One stage object detector
- 각 grid cell마다 b개의 bounding box를 예측 ( $x, y, w, h, p_c$ )
- class probability는 동일한 grid cell에서 예측된 bbox일 경우에 같은 값을 지님
- localization error로 인해 성능이 타 OD방법론에 비해 낮지만 real-time에 비해 빠름  
YOLO v1 : 45 fps / mAP : 63.4 fps / real-time : 30 fps

## ▼ Faster R-CNN



- Two stage object detector
    - region proposal network ) sliding window 마다 9개의 anchor box 생성 후 물체유무 및 물체의 위치 후보군을 선택해둠
    - fast R-CNN ) CNN 통해 얻은 feature map에서 proposed region 기반한 ROI로 classification, bbox 좌표
- ⇒ One stage detector에 비해 성능은 높지만 속도가 느림

## 주요 내용

### ▼ main contribution

- 제목의 의미

YOLOv2			YOLO9000						
Better	Faster	Stronger							
<ul style="list-style-type: none"> <li>- Batch Normalization</li> <li>- High Resolution Classifier</li> <li>- Convolutional with Anchor Boxes</li> <li>- Dimension Clusters</li> <li>- Direct Location Prediction</li> <li>- Fine-Grained Features</li> <li>- Multi-Scale Training</li> </ul>	<ul style="list-style-type: none"> <li>- Darknet-19</li> </ul>	<ul style="list-style-type: none"> <li>- Dataset combination with WordTree</li> <li>- Joing Classification and Detection</li> </ul>	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

YOLO v2 ⇒ better, faster

YOLO 9000 ⇒ stronger

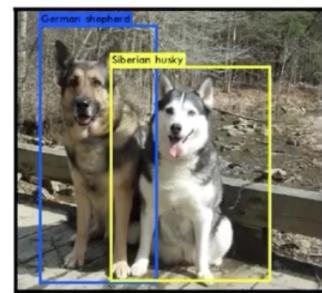
- 핵심 내용

- YOLO v2 : v1의 단점을 개선해 연산을 빠르고, 정확도를 높인 YOLO v2를 제안
- YOLO 9000 : detection dataset의 적은 class 개수로 인한 예측가능한 class 개수의 증가  
→ classification dataset과의 join training을 통해 존재하지 않는 object class에 대한 예측도 가능해짐
- Darknet - 19 : 새로운 classification network인 Darknet-19를 통해 성능을 향상

Method	data	mAP	Fps
YOLOv1	07++12	57.9	45
YOLOv2	07++12	78.6	40

[YOLOv2 성능개선]

mAP : 성능 / Fps : 속도



[YOLO9000 결과]

성능이 개선됨 ( mAP 21향상, Fps 감소  
하긴하였으나 성능이 향상된것에 비하면  
괜찮음! )

더 정확한 개의 이름으로 예측하게됨

## ▼ YOLO v2

### better

- ▼ batch normalization

: Covariate shift (이전 layer의 파라미터 변화로 인해 현재 layer의 입력 분포가 바뀌는 현상) 방지. 학습과정에서 scale(y), shift(b)를 batch별로 구하고, 값을 저장해 test시 활용

- YOLO v2에서의 BN(batch normalization)

1. mini batch 사용하여 학습시, 빠른 수렴이 가능
2. 정규화 효과 (overfitting 발생하지 않음)  $\Rightarrow$  dropout layer 대신 batch normalization을 추가해 mAP 2% 증가

Train	Inference
<p><b>Input:</b> Values of <math>x</math> over a mini-batch: <math>\mathcal{B} = \{x_1 \dots m\}</math>;  <b>Parameters to be learned:</b> <math>\gamma, \beta</math></p> <p><b>Output:</b> <math>\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}</math></p> $\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$ $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$ $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$ $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$	<p><b>for</b> <math>k = 1 \dots K</math> <b>do</b></p> <p>// For clarity, <math>x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}</math>, etc.</p> <p>Process multiple training mini-batches <math>\mathcal{B}</math>, each of size <math>m</math>, and average over them:</p> $\begin{aligned} E[x] &\leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}] \\ \text{Var}[x] &\leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2] \end{aligned}$ <p>In <math>N_{\text{BN}}^{\text{inf}}</math>, replace the transform <math>y = \text{BN}_{\gamma, \beta}(x)</math> with</p> $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$

▼ high resolution classifier

저해상도가 아닌 고해상도 이미지를 바로 사용하기에.. fine-tuning 단계를 하나 더 추가 함

① Pretraining Image Classification Dataset



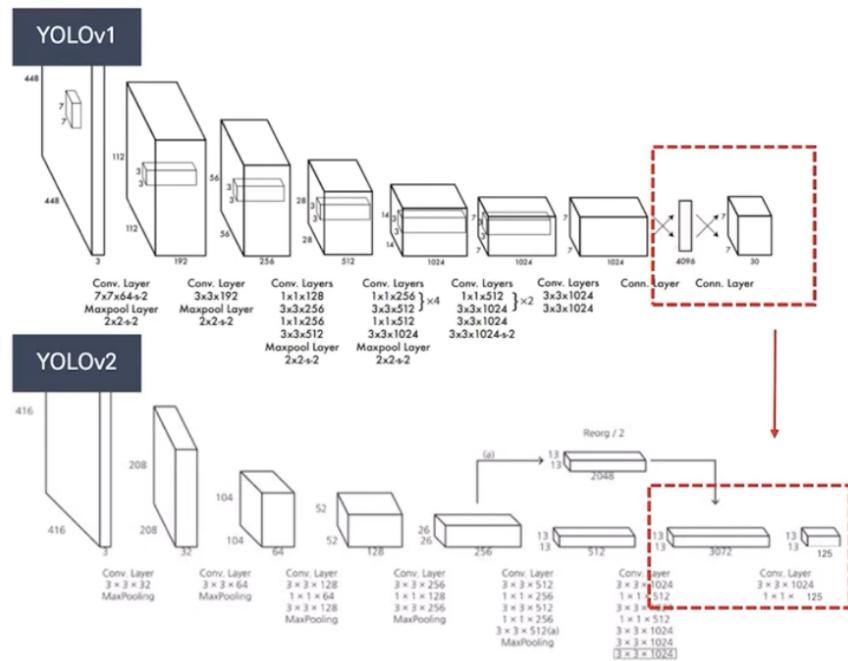
참고) ③에서의 입력이미지 사이즈 416x416 인 이유

- 최종 feature map 크기를 홀수로 만들기 위함
- Large object는 이미지 중앙에 위치하는 경우 많음
- 중심에 위치하는 grid cell이 1개로 만들기 위함

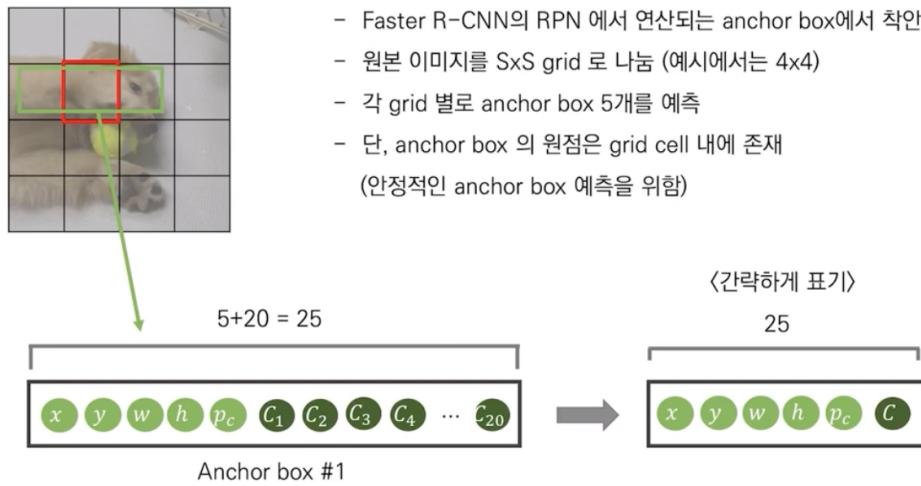
2에서 448x448 고해상도 이미지를 넣게됨

3에서 416x416인 이유 : 최종 feature map 크기를 홀수로 만들기위함(중심에 위치한 grid cell을 1개로 하기위해서)

▼ convolutional with Anchor Boxes

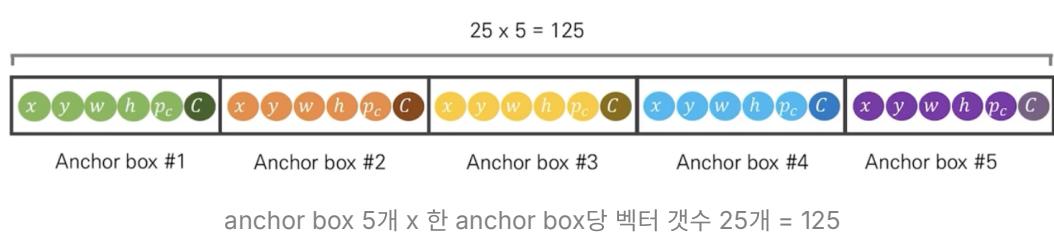


- 네트워크 변화 : fc layer 2개를 모두 제거함, convolution network로 prediction
- Anchor box 사용 : grid cell마다 5개 anchor box로 예측함
- 적용 효과 : mAP ) 69.5 → 69.2 / Recall ) 81% → 88% ( $\Rightarrow$  anchor box로 예측한 것 중 잘 맞춘 box의 개수가 증가했다는 의미 )



하나의 grid cell을 위한 anchor box로,

중심 위치를 의미하는  $x, y$  / 너비와 높이를 의미하는  $w, h$  / confidence score를 의미하는  $P_c$  / 클래스를 의미하는  $C \Rightarrow 5(x, y, w, h, p_c) + 20(class) = 25$



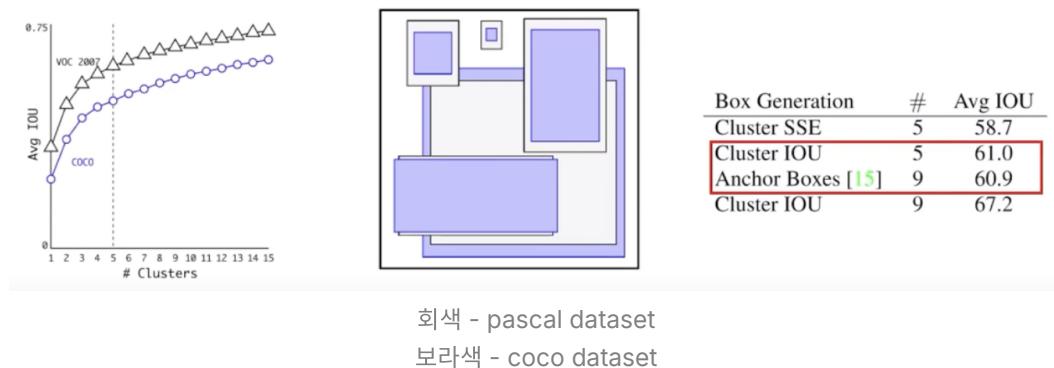
grid cell마다 object 존재할 것 같은 5개의 anchor box를 찾음

YOLO v1과 달리, anchor box별로  $p_c$  계산

output tensor :  $13 \times 13$  (featuremap 크기)  $\times 125$  (anchorbox 및 벡터 크기)

#### ▼ dimension clusters

anchor box를 어떻게 설정하나?



faster R-CNN에서 anchor box ratio, size를 미리 정하는 부분에 대해 문제제기

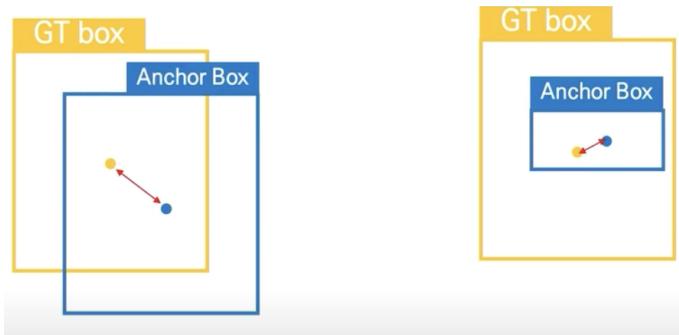
⇒ YOLO v2에서 k-means clustering을 토해 GT와 가장 유사한 optimal anchor box 탐색

k-means clustering 기준 :  $\text{IOU}(\text{box}, \text{centroid})$  를 사용

object detection dataset 사용해 생성한 anchor box들로 clustering을 수행

실험 결과 )  $k=5$  일 때 가장 적합했음, dataset 별로 anchor box 사이즈 및 크기에 차이 존재

cluster IOU(YOLO v2)가 anchor boxes(faster R-CNN)보다 적은 anchor box를 사용했음에도 IOU 값이 더 좋음



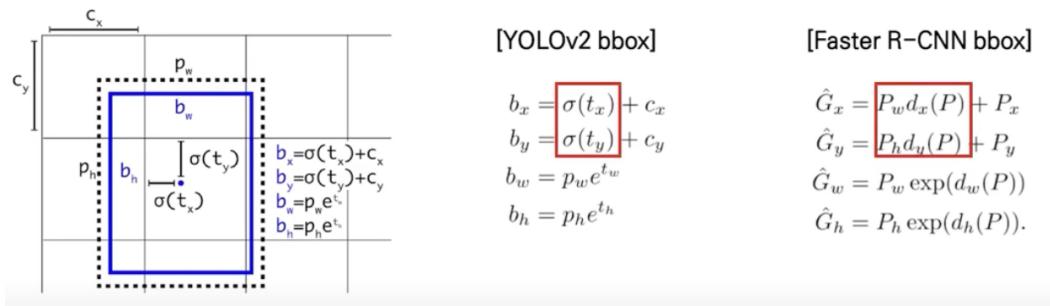
왼쪽이 겹치는 면적이 더 넓지만 점 간의 거리가 오른쪽에 비해 좋지 못함

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

euclidean distance를 사용하지 않고, IOU를 사용

→ anchor box 중심점 간 거리로 계산 시, 실제로는 유사하지 않음에도 같은 cluster에 속하게 될 우려가 존재하기에..

#### ▼ direct location prediction

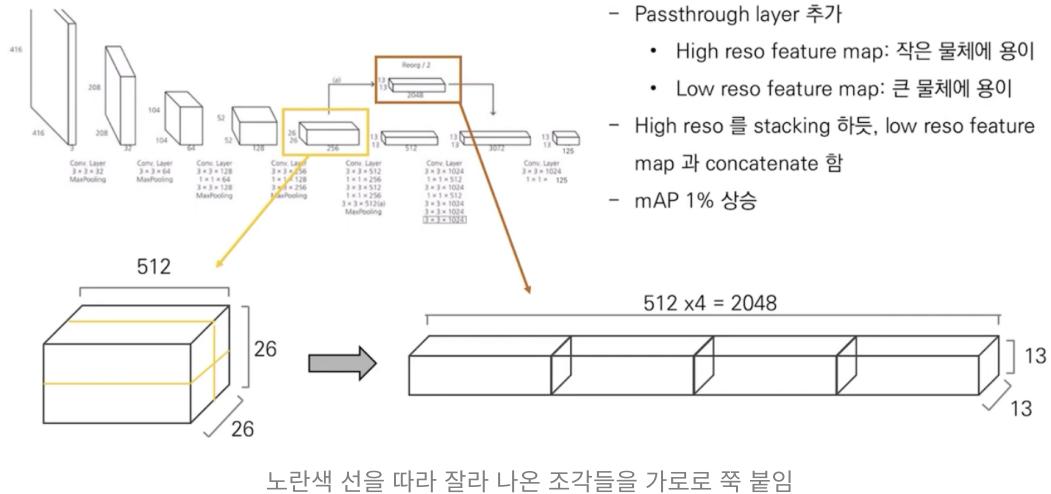


anchor box의 각 좌표에 **sigmoid function**을 적용해 grid cell 내 중심에 위치하도록 함

sigmoid function을 적용하면 0~1 사이의 값을 갖게됨

적용 이유 ) faster R-CNN처럼 함수 d에 아무 제약이 없으면 cell을 벗어난 anchor가 생성됨

#### ▼ fine-grained features



- passthrough layer 추가

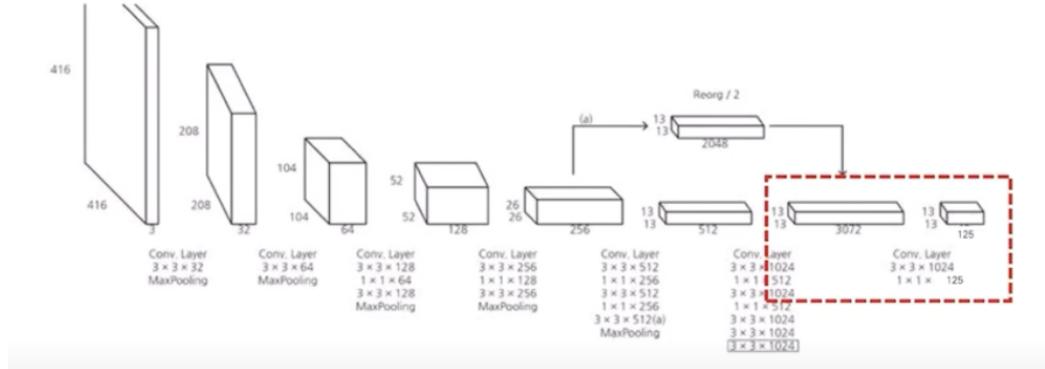
high reso feature map - 작은 물체에 용이함

low reso feature map - 큰 물체에 용이함

high reso를 stacking하듯이, low reso feature map과 concatenate함

⇒ mAP 1% 상승

### ▼ multi-scale training



맨 마지막 부분에 fully connected layer 대신에 convolutional layer를 사용함

→ input size 변화 가능해짐

적용 이유 ) 여러 size의 이미지에도 좋은 성능을 내기위함

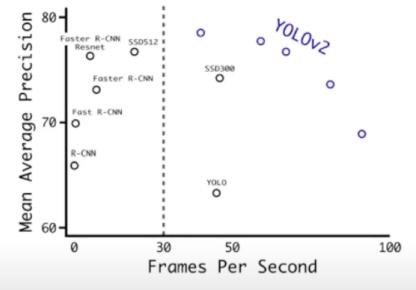
모든 batch 10개마다 random하게 input size를 변경함

단, 범위는 {320, 352....} 32씩 증가

## ▼ Experiment

- #### • YOLO v2 성능 비교

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	<b>78.6</b>	40



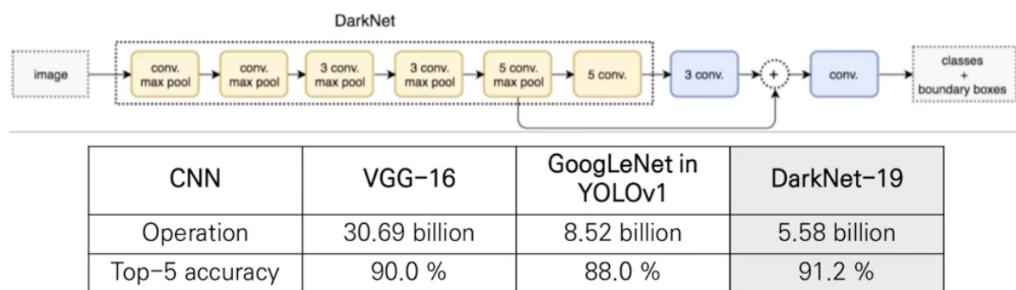
pascal voc 데이터 ) mAP 상승, fps 개선

여전히 mAP와 fps 간의 trade off 존재 (accuracy-speed trade off 관계)

coco 데이터 ) SSD512 > Faster R-CNN > YOLO v2 순서로 성능이 좋음

## faster

### ▼ network ( Darknet-19 )



연산량과 정확도

- DarkNet-19 : conv layer 19 & maxpooling layer 5

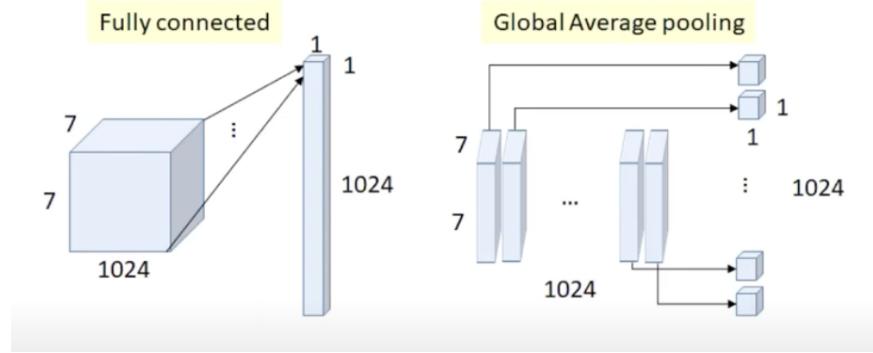
VGG-16과 유사하게 3x3 filter를 사용 → 더 깊은 네트워크를 만듬

network in network 구조 이후에, global average pooling을 사용 → params 수 감소 효과

1×1 filter로 feature representation을 더욱 압축

⇒ VGG-16만큼 정확하며, GoogLeNet처럼 빠른 CNN인 DarkNet-19

### ▼ fully connected layer vs global average pooling



global average pooling : 채널려로 average pooling을 하여 feature을 1차원 벡터로 만듬

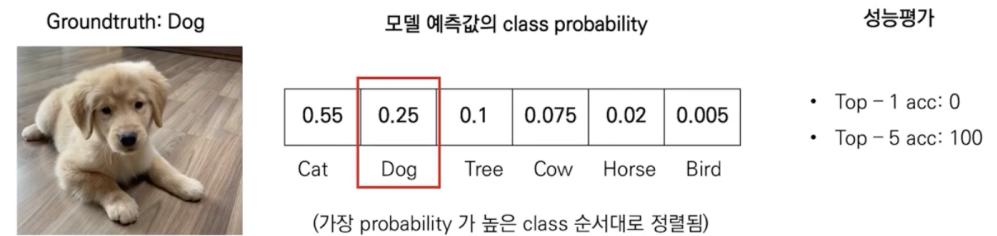
→ fc layer 사용할때보다 params 개수가 크게 감소됨

#### ▼ Top-1 accuracy vs Top-5 accuracy

Top-5 accuracy : 예측확률이 가장 높은 클래스 5개와 groundtruth가 일치하는지

Top-1 accuracy : 예측확률이 가장 높은 클래스 1개와 groundtruth가 일치하는지

ex\_ 강아지



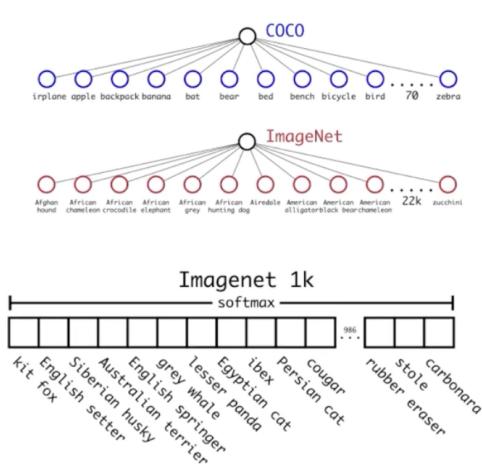
top-1 acc는 가장 높은 확률의 클래스가 cat이므로 성능은 0이되고,

top-5 acc는 가장 높은 확률의 클래스 5개 중 dog가 포함되므로 성능이 100이 됨

#### ▼ YOLO 9000

### stronger

#### ▼ dataset combination with WordTree



- hierarchical classification

imageNet label은 wordnet기반으로 구성됨

- wordnet : directed graph 형 태(트리기반 아님!)

→ hierarchical classification학습을 통해 word tree 생성

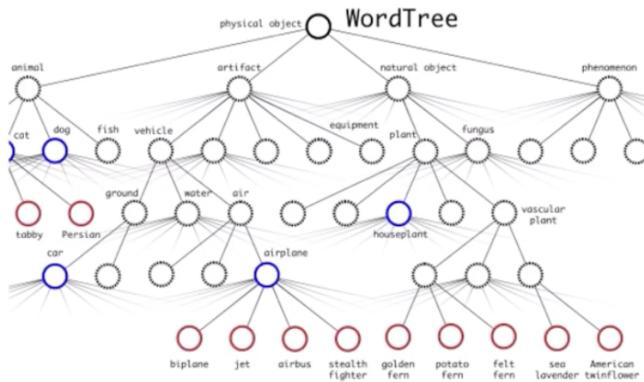
→ wordnet 구조로 공통 root를 갖는 Label를 묶는 작업 시행 (강아지 - 골든리트리버, 말티즈..)

- dataset combination with WordTree

imageNet + coco데이터셋 + imagenet detection을 합침

⇒ 9000개 class label 생성

▼ joint classification and detection



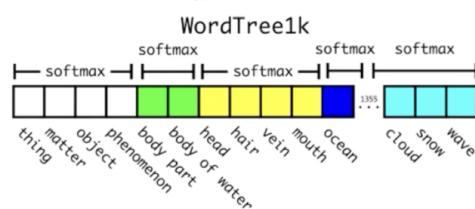
detection dataset(coco data)와 classification dataset(image net)의 개수 차이가 크므로, oversampling으로 개수를 맞추어줌

- detection dataset(coco data) : classification + bbox regression의 loss를 propagate
- classification dataset(image net) : only classification의 loss만을 propagate

⇒ coco데이터로 detection 학습 + 9000개 object에대한 구분 가능

▼ Experiment

onger [실험1] Classification



- Classification (ft.1000-class imagenet)

wordtree를 darknet-19로 분류성능 검증 진행

- wordnet → wordtree 과정에서 369개의 노드가 생성
- Top1 acc : 71.9%, Top5 acc : 90.4%

특징 ) 기존에는 하나로 계산하였는데... → 같은 부모노드로부터 나온 노드끼리 softmax를 계산해 probability 계산

[실험2] Object Detection

diaper	0.0
horizontal bar	0.0
rubber eraser	0.0
sunglasses	0.0
swimming trunks	0.0
...	
red panda	50.7
fox	52.1
koala bear	54.3
tiger	61.0
armadillo	61.7

(위)잘안된 예시, (아래)잘된 예시

- Detection (ft. imagenet detection, coco dataset)

9000개 class로 wordtree 구축 후에 YOLO v2학습함

- detection 성능 : 16.0 ~ 19.7 mAP
- 동물 class 성능 좋음 / clothing, equipment 성능 낮음
- train시 등장했던 class 44개에 대해서는 성능이 좋았으나 나머지 class에서는 좋지 못함