

# 딥러닝을 활용한 컴퓨터 비전

# 목차

1. 오브젝트 디텍션

2. 세그멘테이션

# 1. 오브젝트 디텍션

---

## 1.1 오브젝트 디텍션 개요

1.2 OpenCV 개요

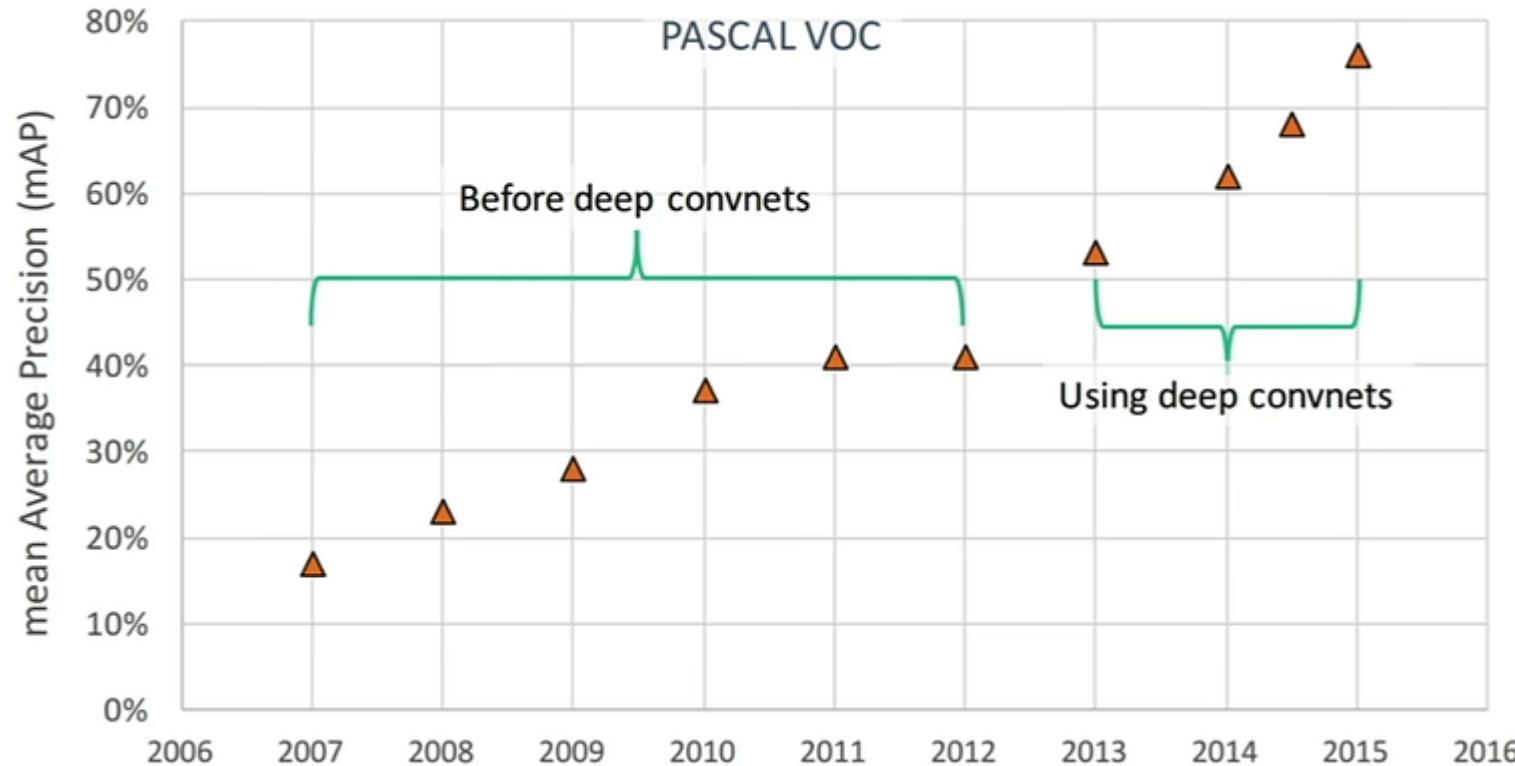
1.3 R-CNN, Fast R-CNN, Faster R-CNN

1.4 SSD (Single Shot MultiBox Detector)

1.5 YOLO (You Only Look Once) 시리즈

1.6 RetinaNet

---



Object(s)의 위치를 찾아내는 것

Classification



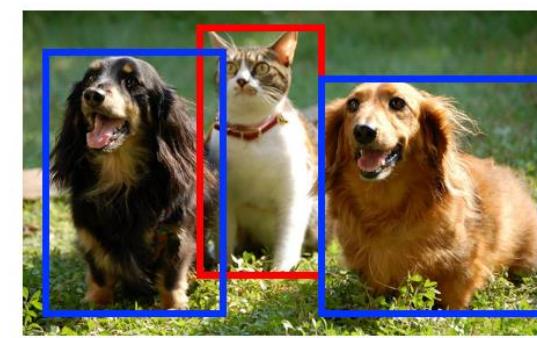
CAT

Classification  
+ Localization



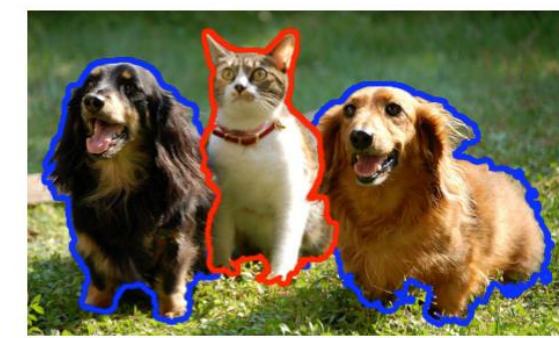
CAT

Object Detection



CAT, DOG

Instance Segmentation



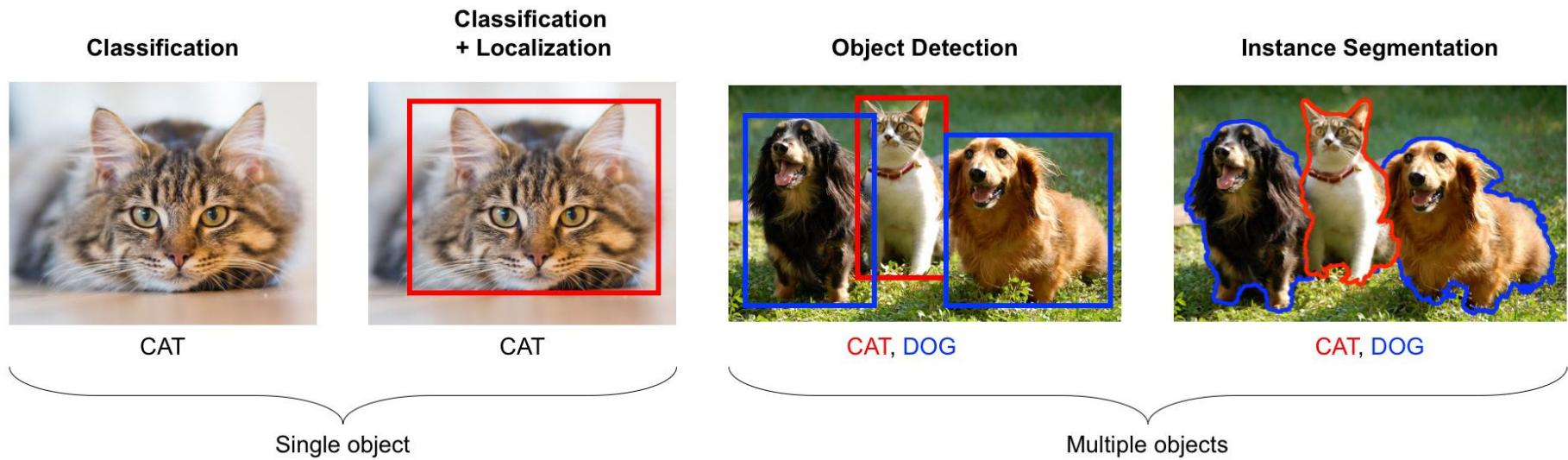
CAT, DOG

Single object

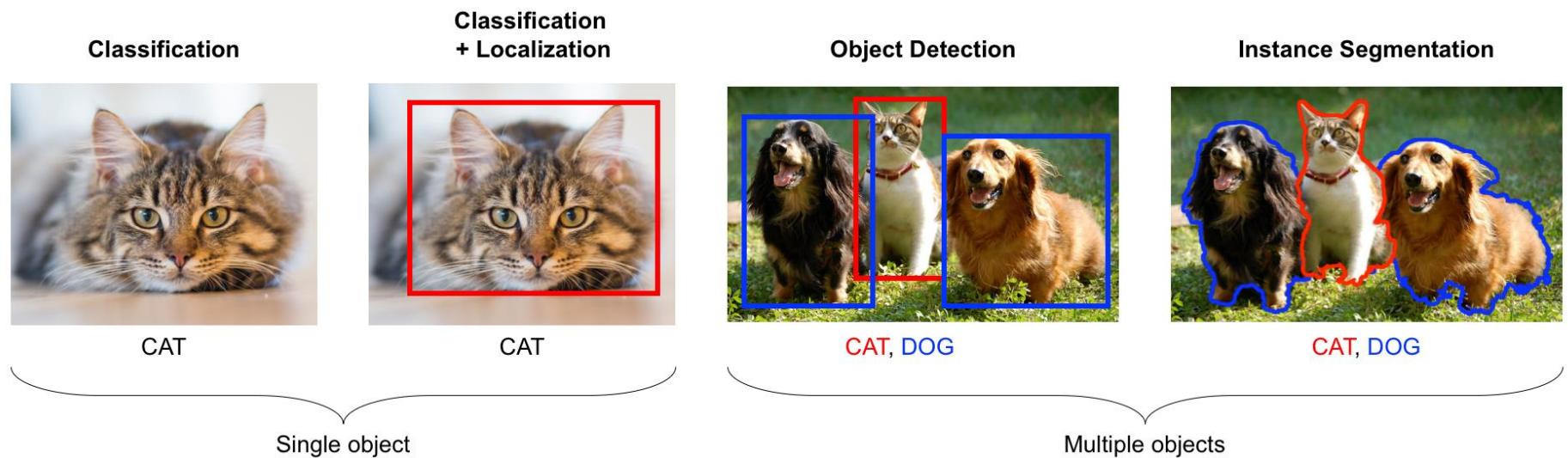
Multiple objects

# Localization/Detection/Segmentation

## 1.1 오브젝트 디텍션 개요



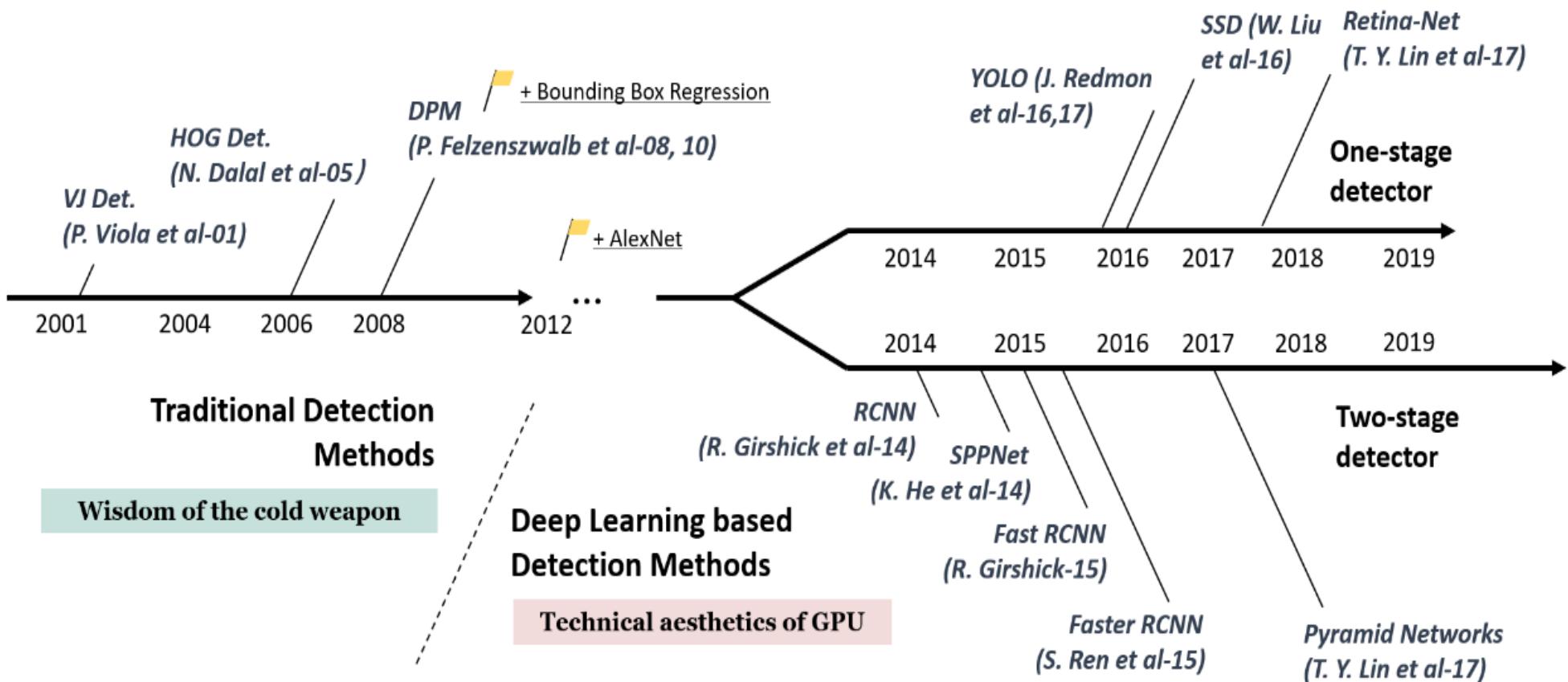
- Localization: 단 하나의 Object 위치를 Bounding box로 지정하여 찾음
- Object Detection: 여러 개의 Object들에 대한 위치를 Bonding box로 지정하여 찾음
- Segmentation: Detection보다 더 발전된 형태로 Pixel레벨 Detection수행



- Localization/Detection은 해당 Object의 위치를 Bounding box로 찾고, Bounding Box내의 오브젝트를 판별
- Localization/Detection은 Bounding box regression(box의 좌표값을 예측)과 Classification 두 개의 문제가 합쳐져 있음
- Localization에 비해 Detection은 두 개 이상의 Object를 이미지의 임의 위치에서 찾아야 하므로 상대적으로 Localization보다 여러가지 어려운 문제에 봉착하게 됨

# Object Detection History

## 1.1 오브젝트 디텍션 개요



# Object Detection의 주요 구성 요소

## 1.1 오브젝트 디텍션 개요

영역추정

Region Proposal

Detection을 위한 Deep  
Learing네트워크 구성

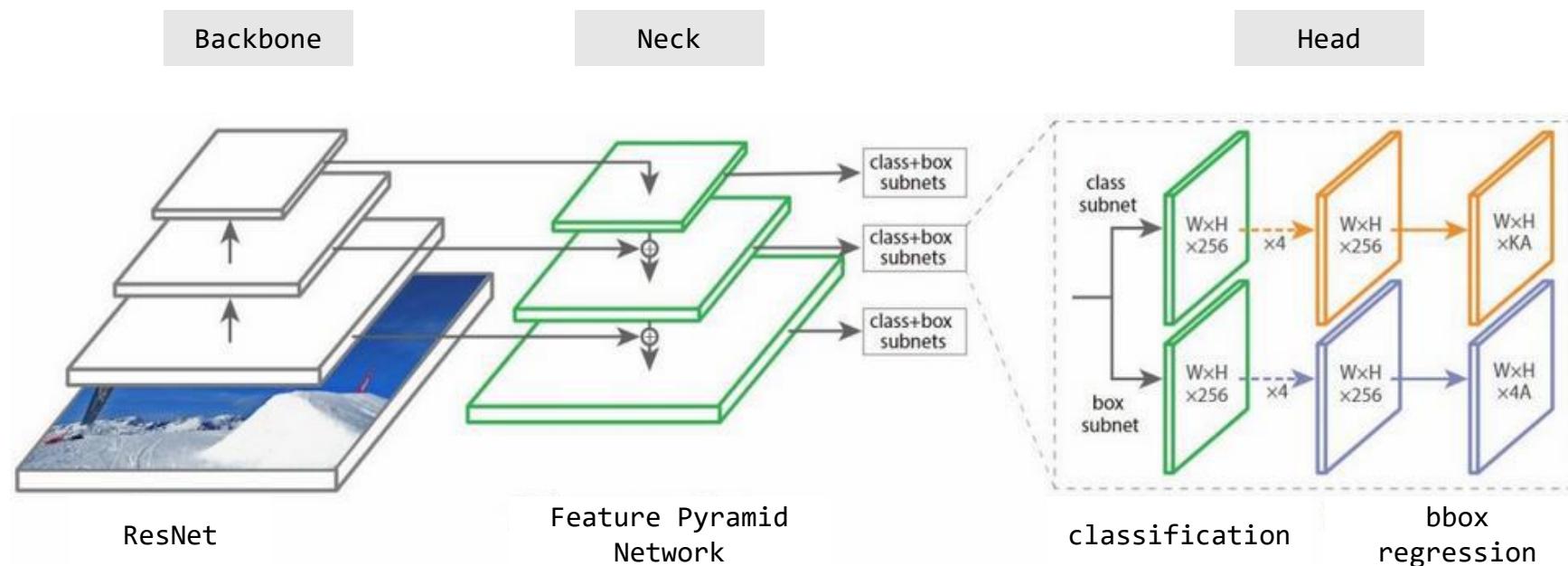
Feature Extraction  
&  
FPN  
&  
Network Prediction

Detection을 구성하는  
기타요소

IOU,  
NMS,  
mAP,  
Anchor box

# 일반적인 Object Detection 모델

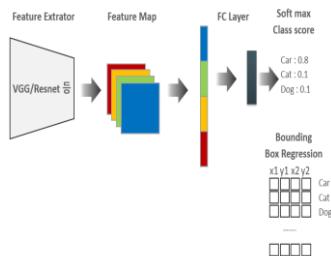
## 1.1 오브젝트 디텍션 개요



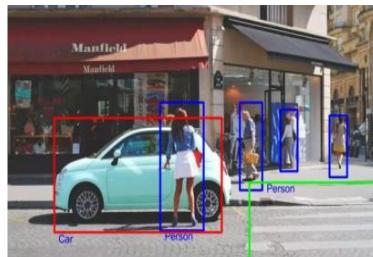
# Object Detection의 난제

## 1.1 오브젝트 디텍션 개요

Classification+  
Regression을 동시에



다양한 크기와 유형의  
오브젝트가 섞여 있음



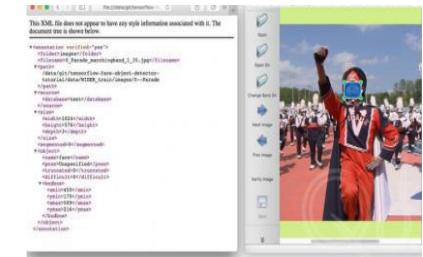
중요한 Detect시간



명확하지 않은  
이미지



데이터 세트의  
부족



이미지에서 여러  
개의 물체를  
classification함과  
동시에 위치를 찾  
아야 함

크기가 서로 다르고,  
생김새가 다양한  
오브젝트가 섞여 있는  
이미지에서 이들을  
Detect해야 함

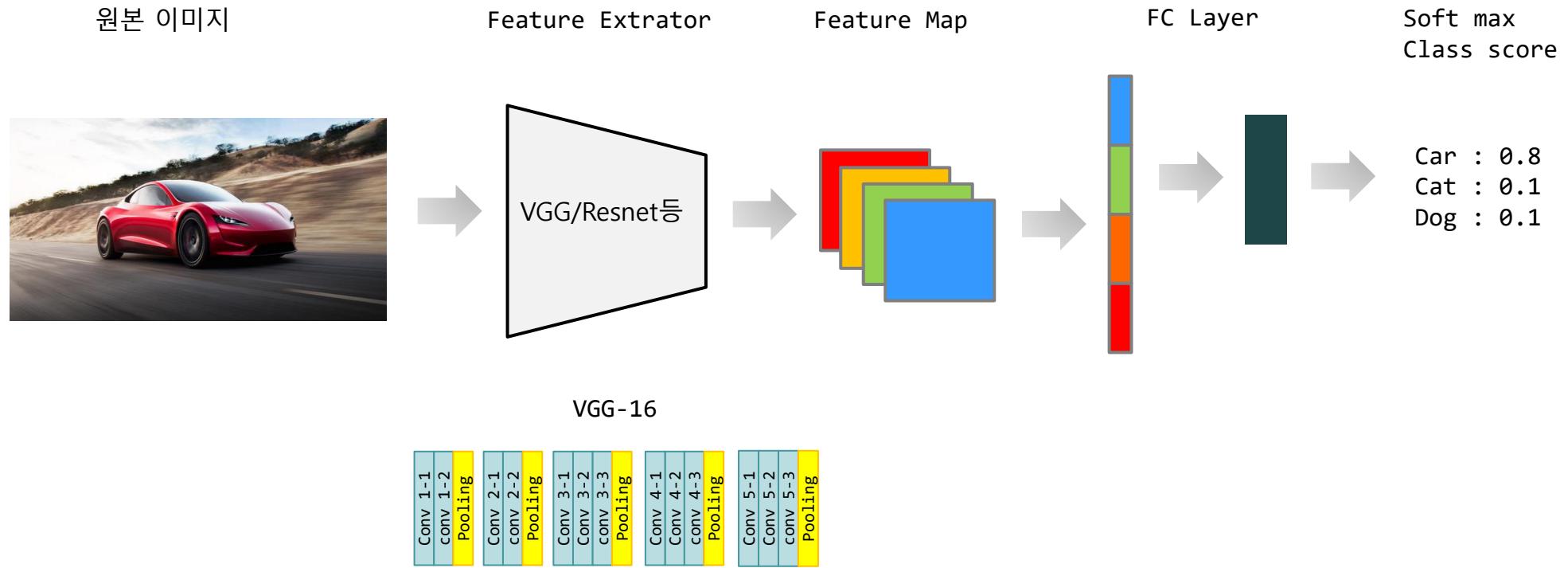
Detect시간이 중요한  
실시간 영상 기반에서  
Detect해야 하는  
요구사항 증대

오브젝트 이미지가 명  
확하지 않은 경우가  
많음. 또한, 전체 이미  
지에서 Detect할 오브  
젝트가 차지하는 비중  
이 높지 않음  
(배경이 대부분을 차  
지하는 경우가 많음)

훈련 가능한 데이터 세트가  
부족(MS Cocodataset 80  
개, Google Open Image  
500개)하며 annotation을  
만들어야 하므로 훈련 데이  
터 세트를 생성하기가 상대  
적으로 어려움

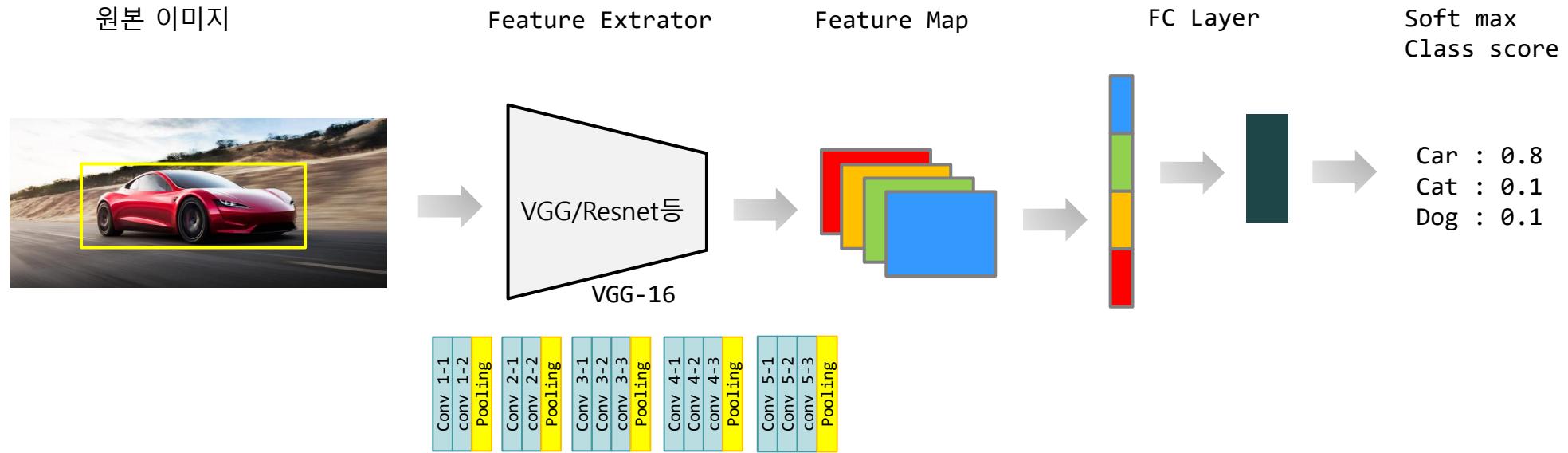
# Object Localization 개요

## 1.1 오브젝트 디텍션 개요



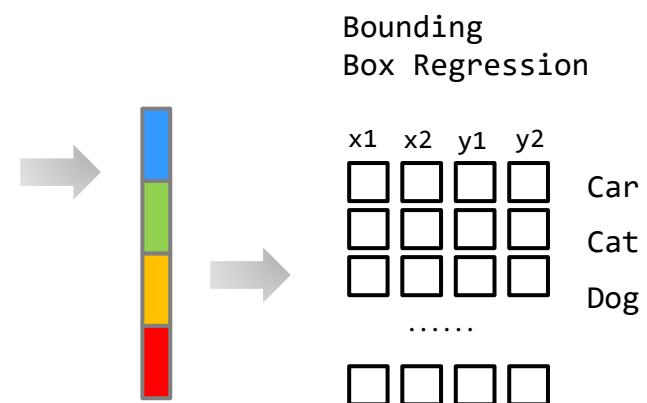
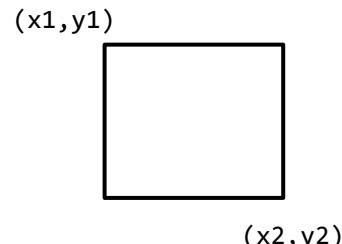
# Object Localization 개요

## 1.1 오브젝트 디텍션 개요



### Annotation 파일

```
<annotation>
  <folder>VOC2012</folder>
  <filename>2007_000032.jpg</filename>
  <source>
    <database>The VOC2007 Database</database>
    <annotation>PASCAL VOC2007</annotation>
    <image>flickr</image>
  </source>
  <size>
    <width>500</width>
    <height>381</height>
    <depth>3</depth>
  </size>
  <segmented>1</segmented>
  <object>
    <name>aeroplane</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>104</xmin>
      <ymin>78</ymin>
      <xmax>375</xmax>
      <ymax>183</ymax>
    </bndbox>
  </object>
</annotation>
```



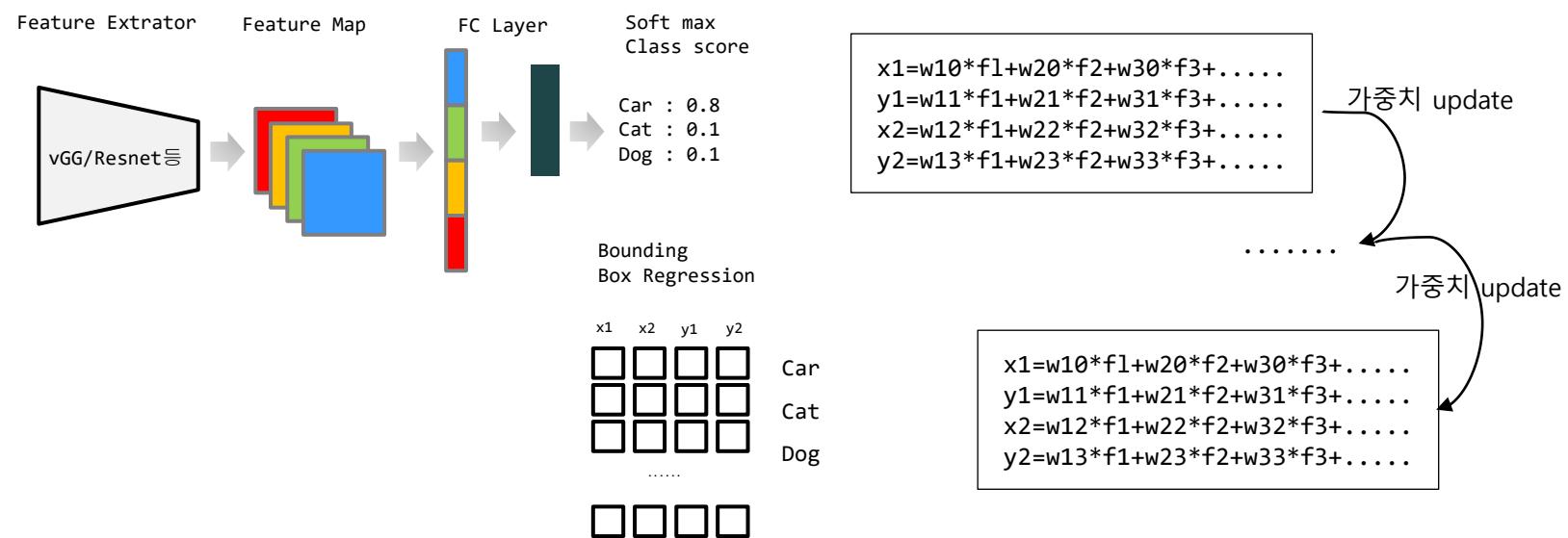
# Object Localization - Bounding box 학습

## 1.1 오브젝트 디텍션 개요

원본 이미지의 Bounding box



학습된 Object의 Bounding box



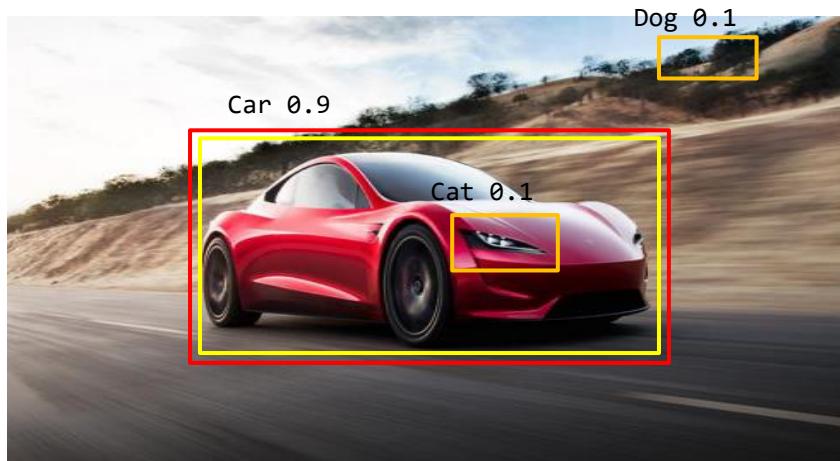
# 여러 이미지와 Bounding box 좌표로 학습

## 1.1 오브젝트 디텍션 개요



# Object Localization 예측 결과

## 1.1 오브젝트 디텍션 개요



Class	Confidence Score	X1	Y1	X2	Y2
Car	0.9	50	60	220	150
Dog	0.1	210	10	240	30
Cat	0.1	160	100	180	120

이미지의 어느 위치에서 Object를 찾아야 하는가?

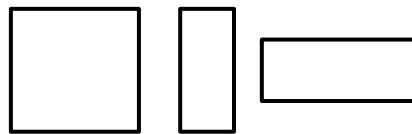


# Sliding Window 방식

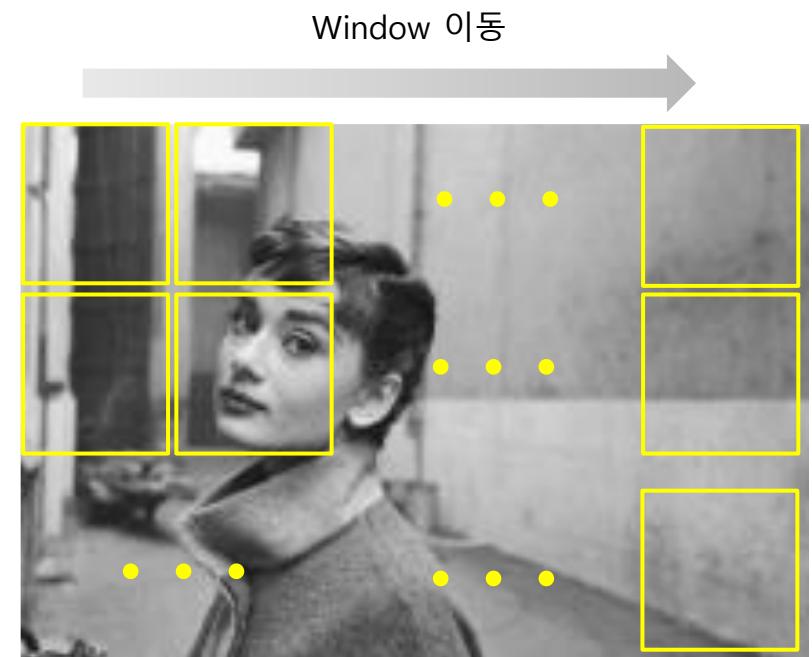
## 1.1 오브젝트 디텍션 개요

Window를 왼쪽 상단에서 부터 오른쪽 하단으로 이동시키면서 Object를 Detection하는 방식

다양한 형태의 Window를 각각 sliding시키는 방식



Window Scale은 고정하고 Scale을 변경한 여러 이미지를 사용하는 방식



- Object Detection의 초기 기법으로 활용
- 오브젝트 없는 영역도 무조건 슬라이딩 하여야 하며 여러 형태의 Window와 여러 Scale을 가진 이미지를 스캔해서 검출해야 하므로 수행 시간이 오래 걸리고 검출 성능이 상대적으로 낮음
- Region Proposal(영역추정)기법의 등장으로 활용도는 떨어졌지만 Object Detection 발전을 위한 기술적 토대 제공

# 이미지 Scale 조정에 따른 여러 크기의 Object Detection

## 1.1 오브젝트 디텍션 개요

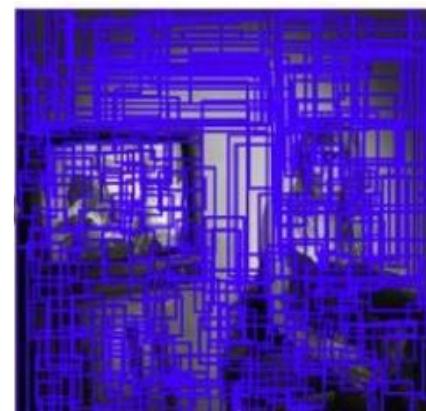
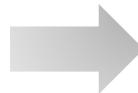


"Object가 있을 만한 후보 영역을 찾자"



원본 이미지

Select



후보 Bounding Box선택

최종 후보 도출



최종 Object Detection

# Selective Search - Region Proposal의 대표 방법

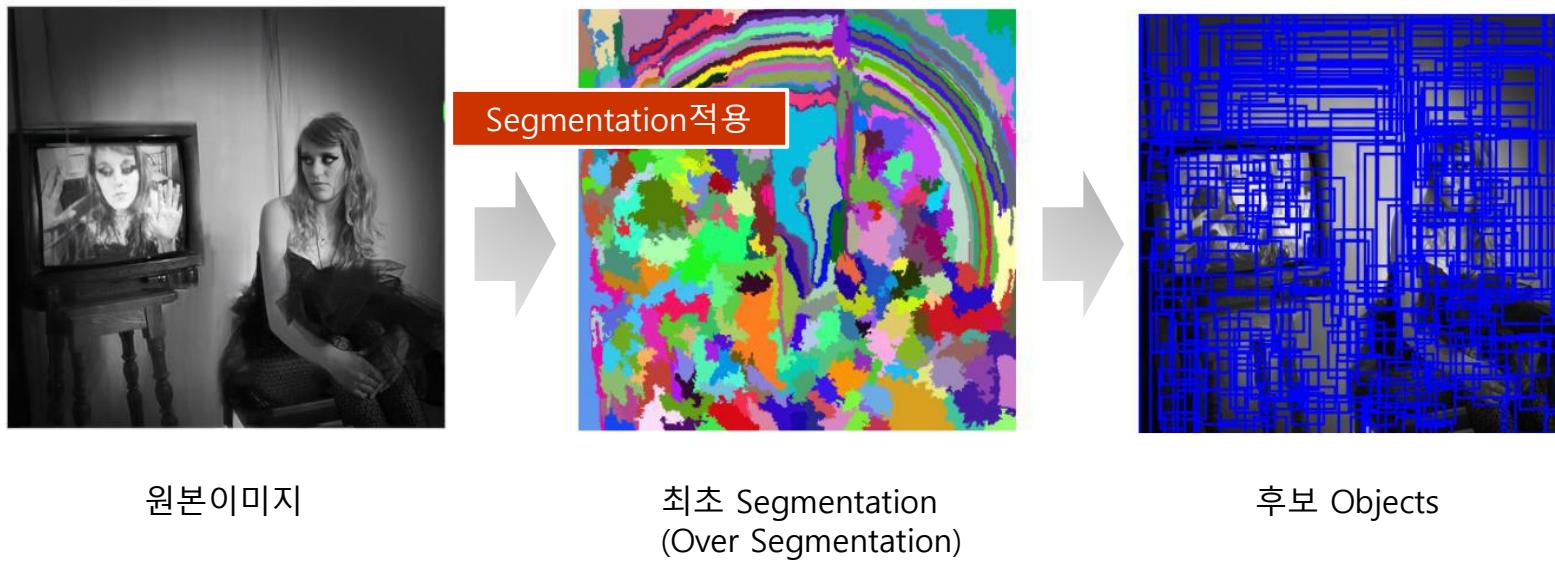
## 1.1 오브젝트 디텍션 개요

빠른 Detection과 높은 Recall 예측 성능을 동시에 만족하는 알고리즘

컬러, 무늬(Texture), 크기(Size), 형태(Shape)에 따라 유사한 Region을 계층적 그룹핑 방법으로 계산

Selective Search는 최초에는 Pixel Intensity기반한 graph-based segment 기법에 따라 Over Segmentation을 수행

(각각의 objec들이 1개의 개별 영역에 담길 수 있도록 많은 초기 영역을 생성 by Felzenszwalb and Huttenlocher 2004)



# Selective Search의 수행 프로세스

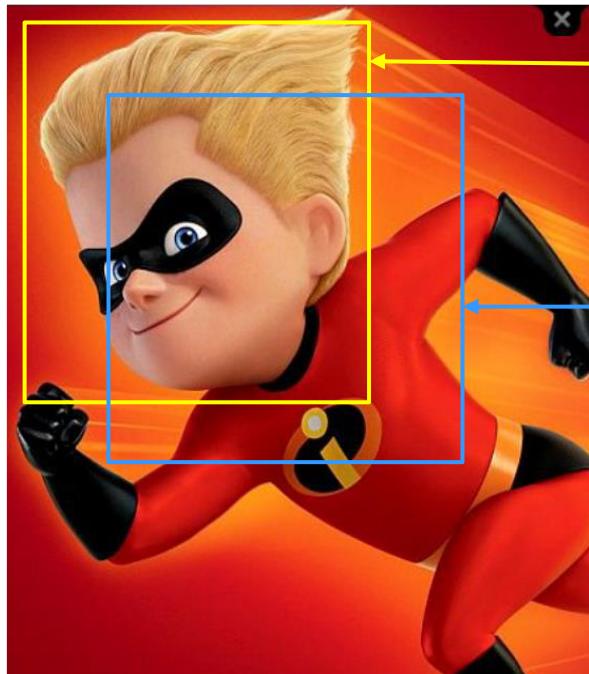
## 1.1 오브젝트 디텍션 개요

1. 개별 Segment된 모든 부분들을 Bounding box로 만들어서 Region Proposal 리스트로 추가
2. 컬러, 무늬(Texture), 크기(Size), 형태(Shape)에 따라 유사도가 비슷한 Segment들을 그룹핑함
3. 다시 1번 Step Region Proposal 리스트 추가, 2번 Step 유사도가 비슷한 Segment들 그룹핑을 계속 반복하면서 Region Proposal을 수행



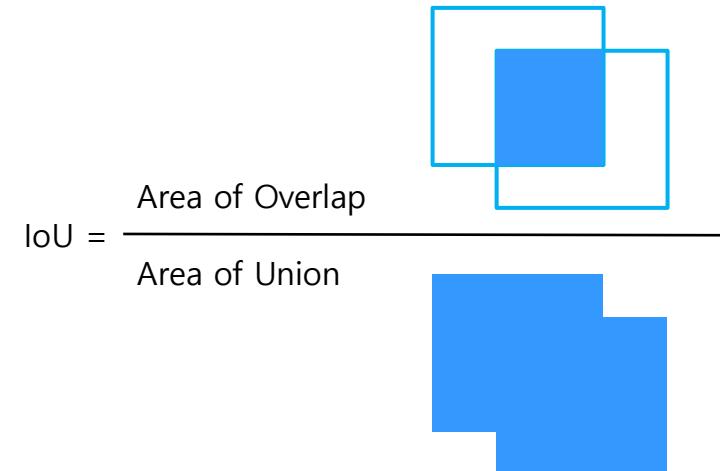
### IoU: Intersection over Union

모델이 예측한 결과와 실측(Ground Truth) Box가 얼마나 정확하게 겹치는지를 나타내는 지표



실측(Ground Truth)  
Bounding box

예측(Predicted)  
Bounding box

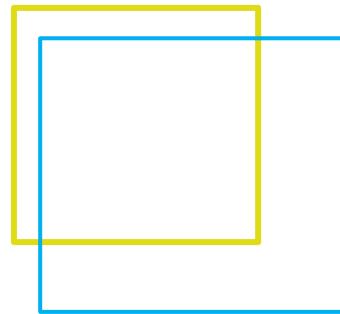


IoU는 개별 Box가 서로 겹치는 영역/전체 Box의 합집합 영역

# IoU에 따른 Detection 성능

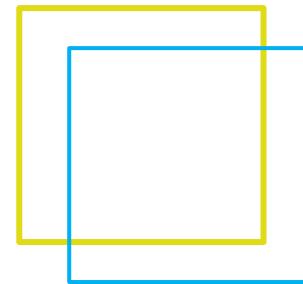
## 1.1 오브젝트 디텍션 개요

IoU: 0.403



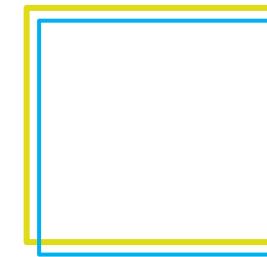
Poor

IoU: 0.740



Good

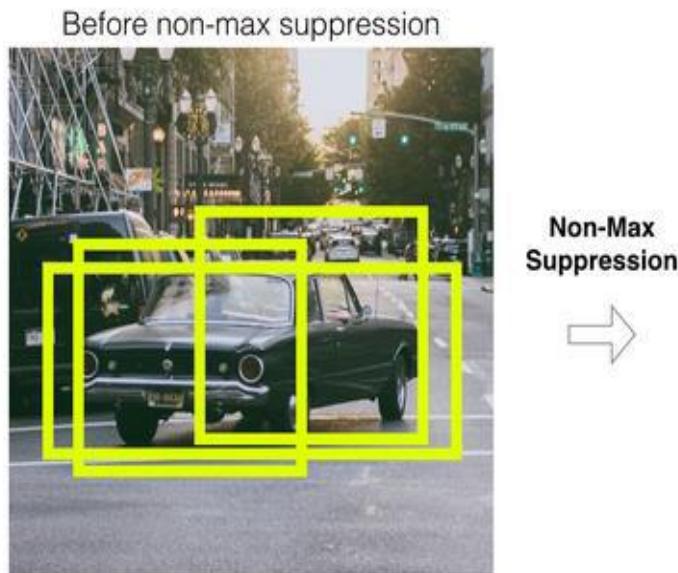
IoU: 0.936



Excellent

# NMS(Non Max Suppression)

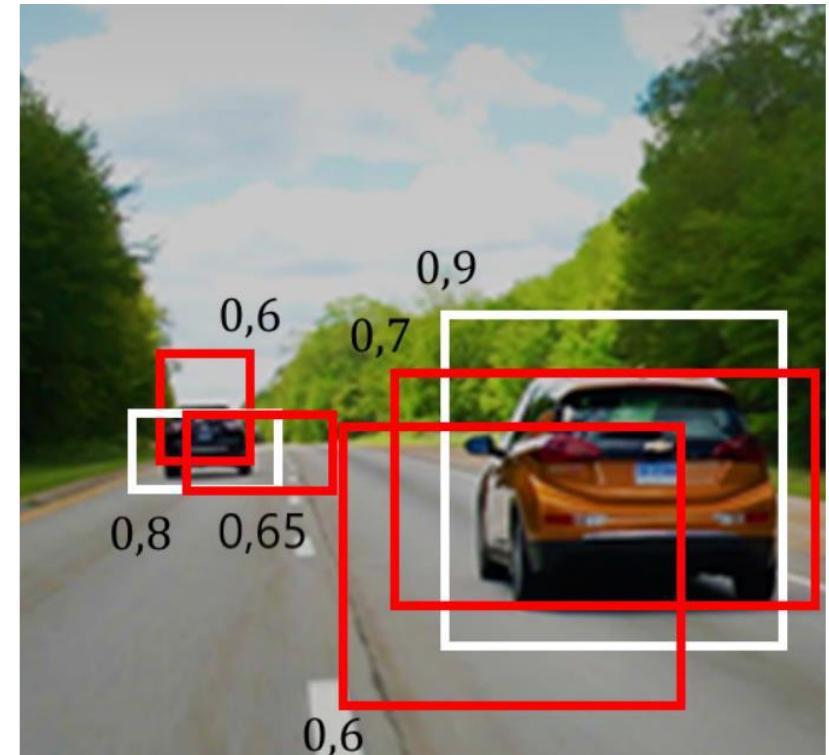
## 1.1 오브젝트 디텍션 개요



- Object Detection 알고리즘은 Object가 있을 만한 위치에 많은 Detection을 수행하는 경향이 강함
- NMS는 Detected된 Object의 Bounding box중에 비슷한 위치에 있는 box를 제거하고 가장 적합한 box를 선택하는 기법

1. Detected된 bounding box별로 특정 Confidence threshold이하 bounding box는 먼저 제거(confidence score < 0.5)
2. 가장 높은 confidence score를 가진 box순으로 내림차순 정렬하고 아래 로직을 모든 box에 순차적으로 적용
  - 높은 confidence score를 가진 box와 겹치는 다른 box를 모두 조사하여 IOU가 특정 threshold 이상인 box를 모두 제거(예:IOU Threshold > 0.4)
3. 남아있는 box만 선택

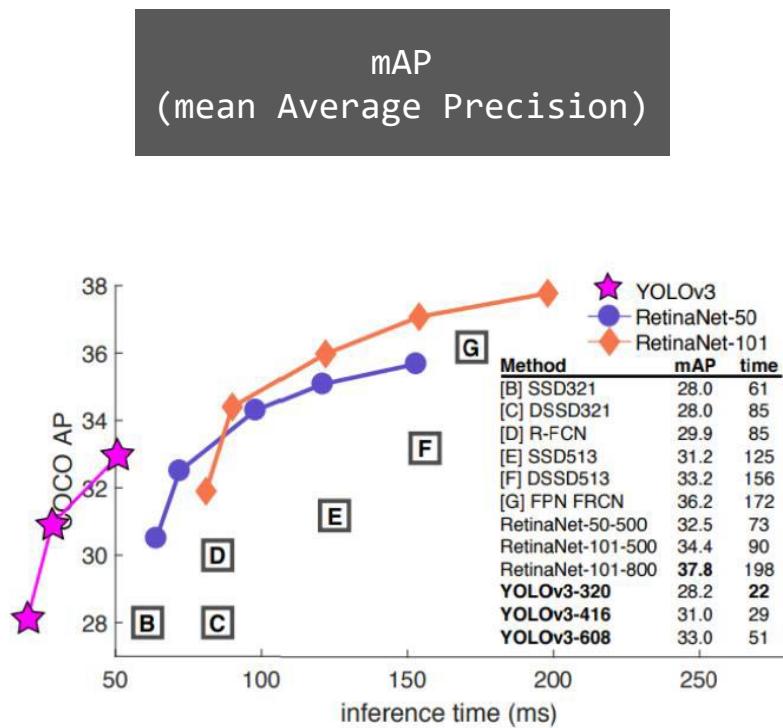
Confidence score가 **높을수록**,  
IOU Threshold가 **낮을수록** 많은 Box가 제거됨



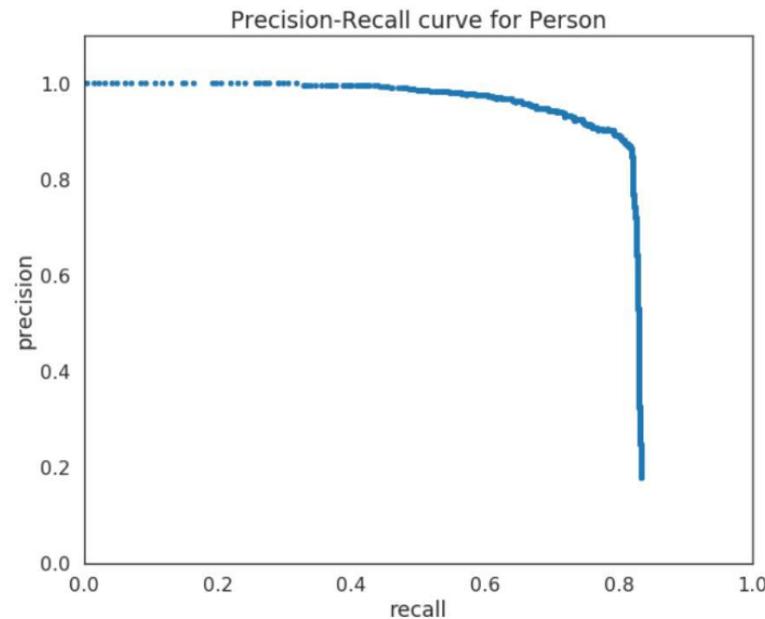
# Object Detection 성능 평가 Metric - mAP

## 1.1 오브젝트 디텍션 개요

실제 Object가 Detected된 재현율(Recall)의 변화에 따른 정밀도(Precision)의 값을 평균한 성능 수치



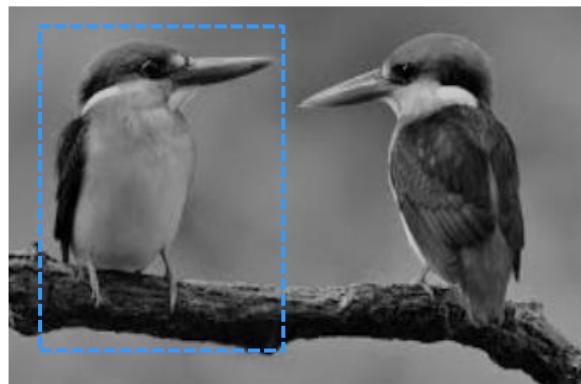
- IOU
- Precision-Recall Curve, Average Precision
- Confidence threshold



정밀도(Precision)과 재현율(Recall)은 주로 이진 분류(Binary Classification)에서 사용되는 성능 지표입니다.

- 정밀도(Precision)는 예측을 Positive로 한 대상 중에 예측과 실제값이 Positive로 일치한 데이터의 비율을 뜻합니다. Object Detection에서는 검출 알고리즘이 검출 예측한 결과가 실제 Object들과 얼마나 일치하는지를 나타내는 지표입니다.
- 재현율(Recall)은 실제 값이 Positive인 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율을 뜻합니다. Object Detection에서는 검출 알고리즘이 실제 Object들을 빠드리지 않고 얼마나 정확히 검출 예측하는지를 나타내는 지표입니다.

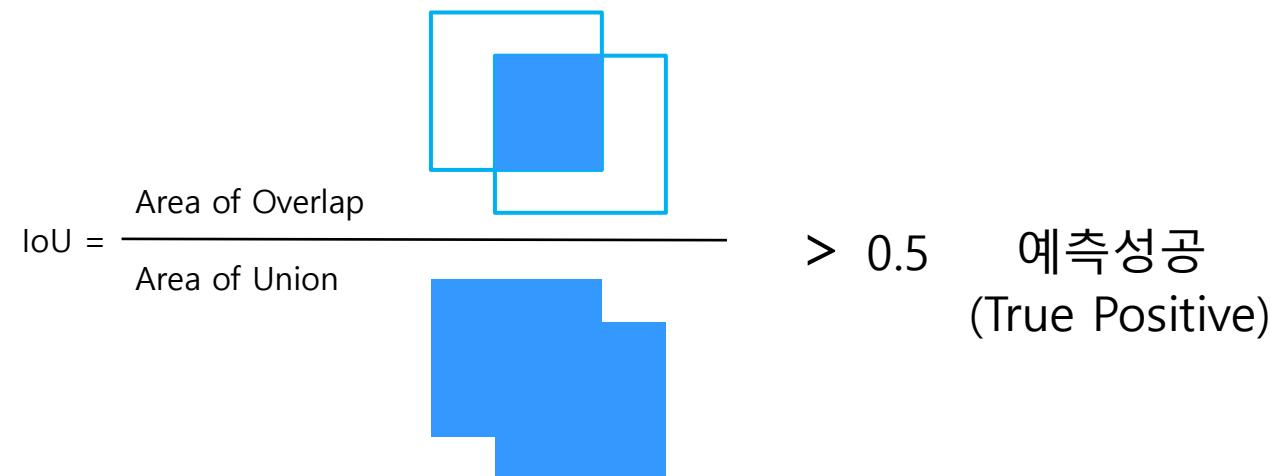
검출 예측을 정확히 Bird로 함  
Precision=100%



무슨 소리?  
오른쪽의 새는 아예 예측하지 못함  
Recall=50%

Object Detection에서 개별 Object에 대한 검출(Detection)예측이 성공하였는지의 여부를 IOU로 결정

일반적으로 PASCAL VOC Challenge에서 사용된 기준을 적용하여 IOU가 0.5이상이면 예측 성공으로 인정  
(그러나 COCO challenge에서는 여러 개의 IOC 기준을 변경해 예측 성공을 적용)



# 오차행렬 (Confusion Matrix)

## 1.1 오브젝트 디텍션 개요

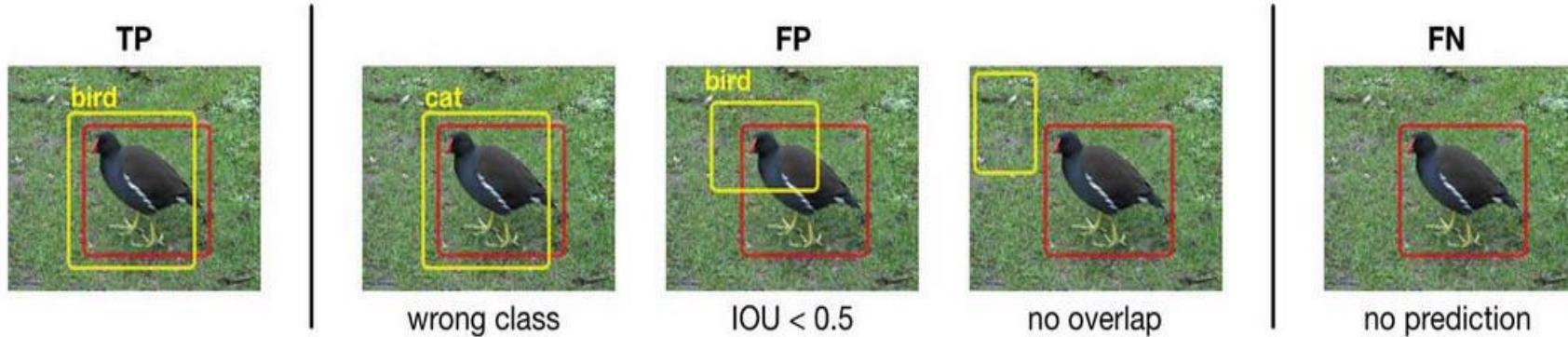
오차행렬은 이진 분류의 예측 오류가 얼마인지와 더불어 어떠한 유형의 예측 오류가 발생하고 있는지를 함께 나타내는 지표입니다.

		Predicted 0	Predicted 1
Actual 0	TN	FP	
	FN	TP	

# Object Detection에서 TP, FP, FN에 따른 정밀도와 재현율

## 1.1 오브젝트 디텍션 개요

Object Detection에서 TP, FP, FN



- 정밀도 =  $TP / (FP+TP)$
- 재현율 =  $TP / (FN+TP)$

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

- 재현율이 상대적으로 더 중요한 지표인 경우는 실제 Positive 양성인 데이터 예측을 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 : 암 진단, 금융사기 판별
- 정밀도가 상대적으로 더 중요한 지표인 경우는 실제 Negative 음성인 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 : 스팸메일

### 정밀도를 100%로 만드는 법

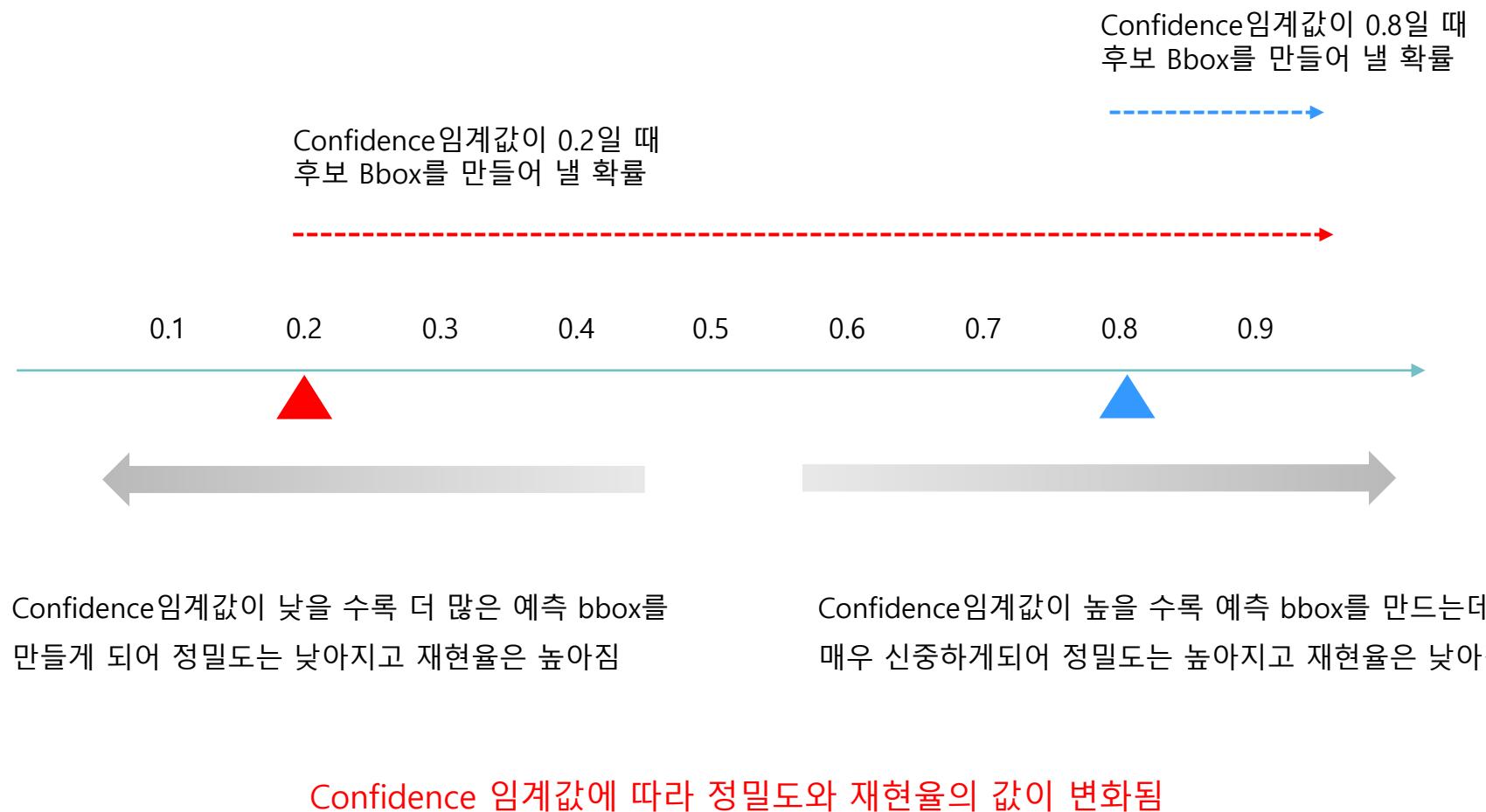
- 확실한 기준이 되는 경우만 Positive로 예측하고 나머지는 모두 Negative로 예측한다. 정밀도= $TP/(TP+FP)$ 이다. 전체 환자 1000명 중 확실한 Positive 징후만 가진 환자는 단 1명이라고 하면 이 한 명만 Positive로 예측하고 나머지는 모두 Negative로 예측하더라도 FP는 0, TP는 1이 되므로 정밀도는  $1/(1+0)$ 으로 100%가 된다.

### 재현율을 100%로 만드는 법

- 모든 환자를 Positive로 예측하면 된다. 재현율= $TP/(TP+FN)$ 이므로 전체 환자 1000명을 다 Positive로 예측하면 됨. 이 중 실제 양성인 사람이 30명 정도라도 TN이 수치에 포함되지 않고 FN은 아예 0이므로  $30/(30+0)$ 으로 100%가 된다.

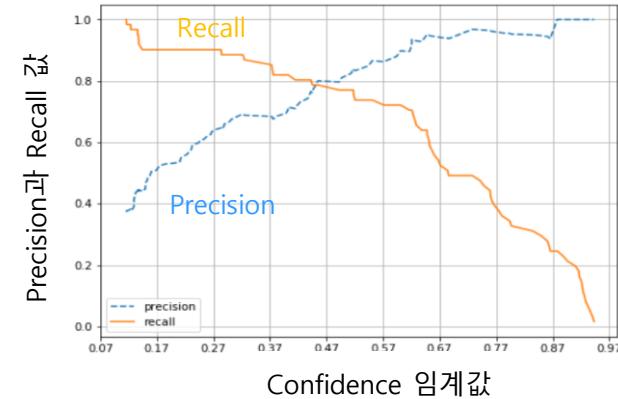
# Confidence임계값에 따른 정밀도- 재현율 변화

## 1.1 오브젝트 디텍션 개요



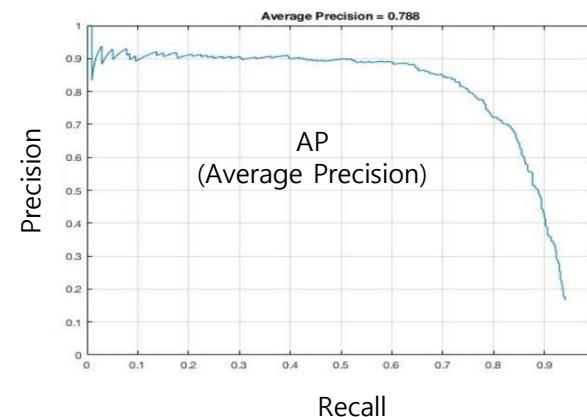
### 정밀도 재현율 트레이드 오프 (Precision Recall Trade-off)

- Confidence 임곗값(Threshold)을 조정하면 정밀도 또는 재현율의 수치를 높일 수 있습니다. 하지만 정밀도와 재현율은 상호보완적인 평가 지표이기 때문에 어느 한 쪽을 강제로 높이면 다른 하나의 수치는 떨어지기 쉽습니다. 이를 정밀도/재현율의 트레이드 오프(Trade-off)라고 부릅니다.



### 정밀도 재현율 곡선 (Precision-Recall Curve)

- Recall값의 변화에 따른 (confidence값을 조정하면서 얻어진) Precision값을 나타낸 곡선을 정밀도 재현율 곡선이라고 합니다. 그리고 이렇게 얻어진 Precision값의 평균을 AP라고 하며, 일반적으로 정밀도 재현율 곡선의 면적 값으로 계산됩니다.



# Confidence에 따른 Precision과 Recall의 변화

## 1.1 오브젝트 디텍션 개요

Confidence:0.9

confidence	True/ False	Precision	Recall
0.9	True	1.0	0.2

Confidence:0.8

confidence	True/ False	Precision	Recall
0.9	True	1.0	0.2
0.8	True	1.0	0.4

Confidence:0.7

confidence	True/ False	Precision	Recall
0.9	True	1.0	0.2
0.8	True	1.0	0.4
0.7	False	0.67	0.4

Confidence:0.6

confidence	True/ False	Precision	Recall
0.9	True	1.0	0.2
0.8	True	1.0	0.4
0.7	False	0.67	0.4
0.6	False	0.5	0.4

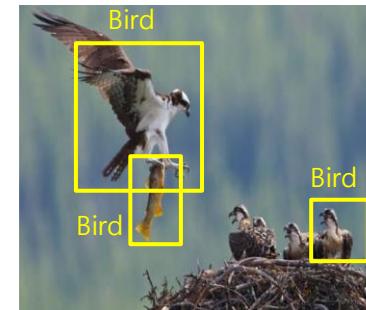
Precision=1/1, Recall=1/5



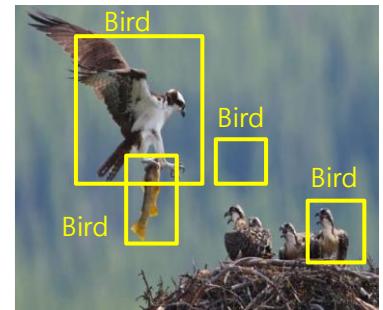
Precision=2/2, Recall=2/5



Precision=2/3, Recall=2/5



Precision=2/4, Recall=2/5

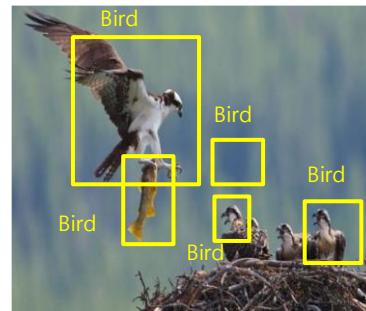


# Confidence에 따른 Precision과 Recall의 변화

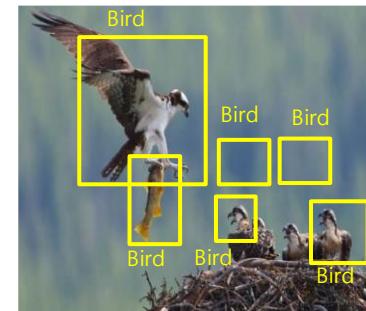
## 1.1 오브젝트 디텍션 개요

confidence	True/F alse	Precision	Recall
0.9	True	1.0	0.2
0.8	True	1.0	0.4
0.7	False	0.67	0.4
0.6	False	0.5	0.4
0.5	True	0.6	0.6
0.4	False	0.5	0.6
0.3	True	0.57	0.8
0.2	False	0.5	0.8
0.1	False	0.44	0.6
0.05	True	0.5	1.0

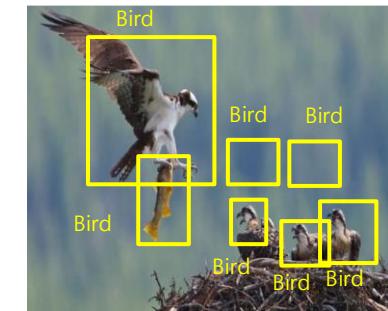
Confidence : 0.5  
Precision=3/5, Recall=3/5



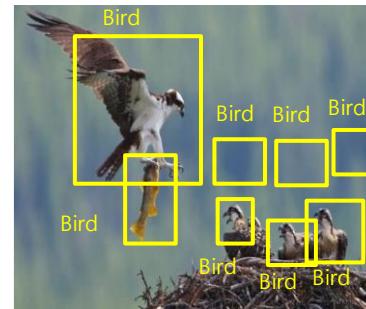
Confidence : 0.4  
Precision=3/6, Recall=3/5



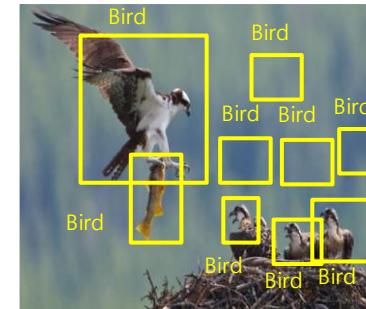
Confidence : 0.3  
Precision=2/4, Recall=2/5



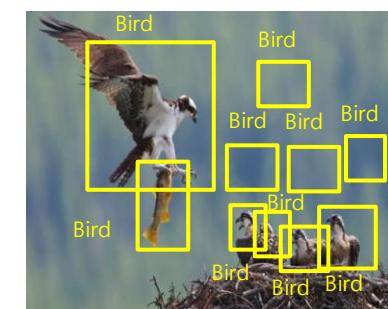
Confidence : 0.2  
Precision=4/8, Recall=4/5



Confidence : 0.1  
Precision=4/9, Recall=4/5

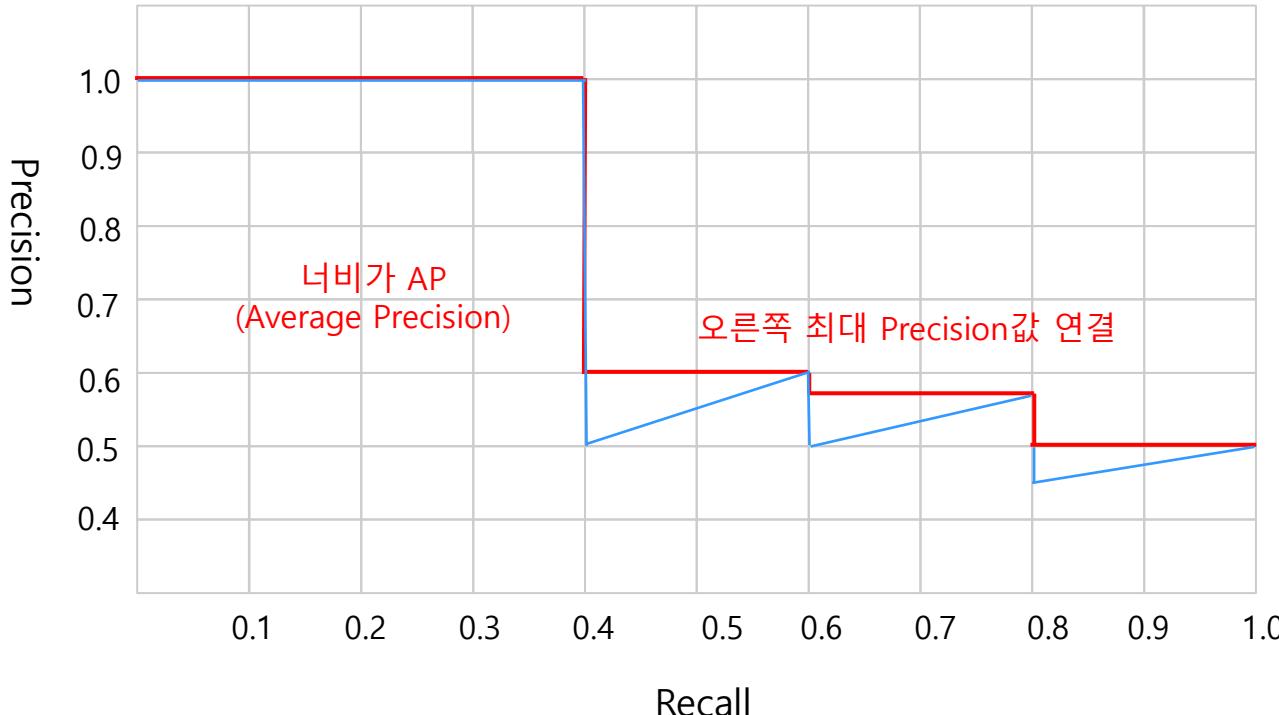


Confidence : 0.05  
Precision=5/10, Recall=5/5



# AP(Average Precision)계산하기

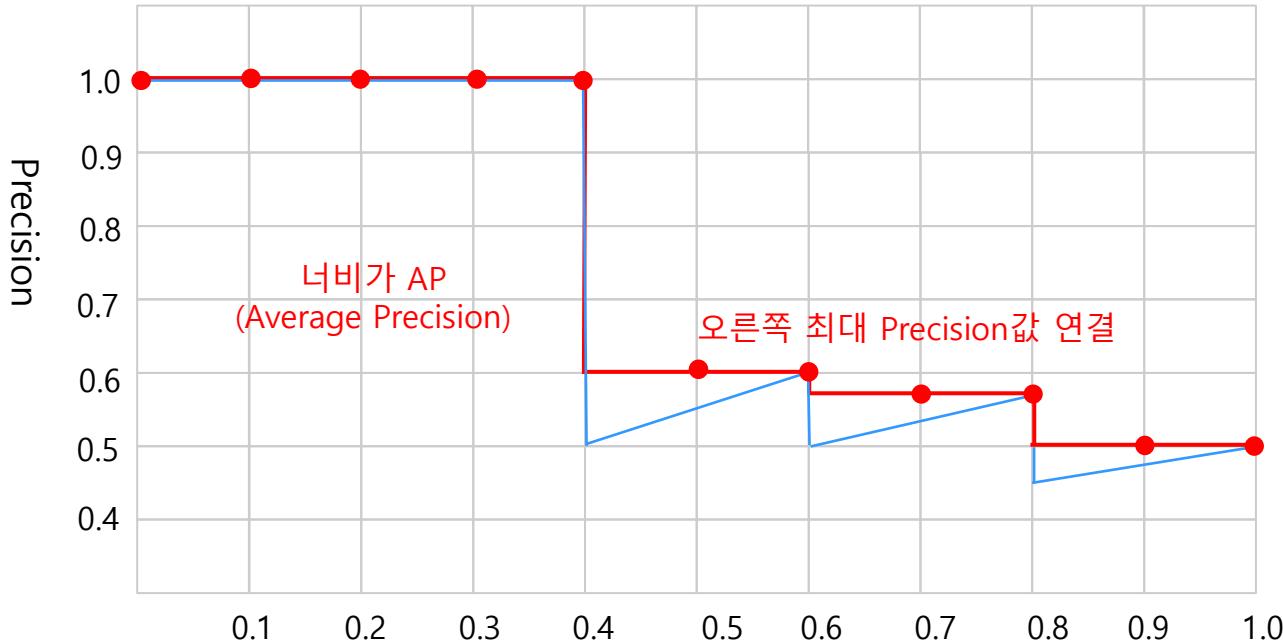
## 1.1 오브젝트 디텍션 개요



confidence	True/False	Precision	Recall
0.9	True	1.0	0.2
0.8	True	1.0	0.4
0.7	False	0.67	0.4
0.6	False	0.5	0.4
0.5	True	0.6	0.6
0.4	False	0.5	0.6
0.3	True	0.57	0.8
0.2	False	0.5	0.8
0.1	False	0.44	0.6
0.05	True	0.5	1.0

# AP(Average Precision)계산하기

## 1.1 오브젝트 디텍션 개요



개별 11개 (0.0~1.0까지) Recall포인트별로  
최대 Precision의 평균 값을 구함

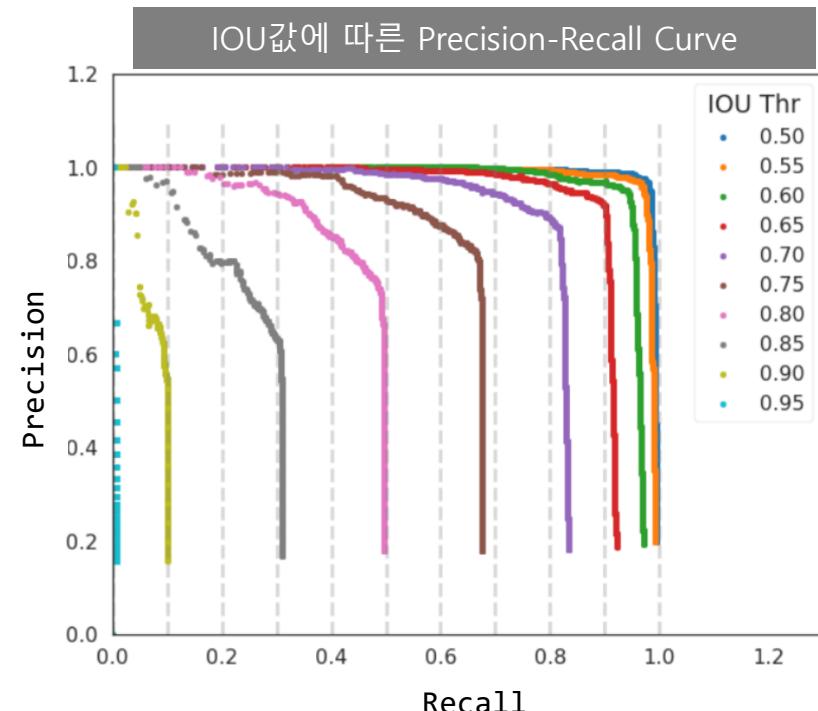
$$AP = \frac{1}{11} \times (mp(r=0) + mp(r=0.1) + \dots + mp(r=1))$$

$$= \frac{1}{11} \times (1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 0.6 + 0.6 + 0.57 + 0.57 + 0.5)$$

$$= \frac{1}{11} \times (5 \times 1.0 + 0.6 \times 2 + 0.57 \times 2 + 0.5 \times 2)$$
$$= 0.758$$

- AP는 한 개 오브젝트에 대한 성능 수치
- mAP(mean Average Precision)은 여러 오브젝트들의 AP를 평균한 값

- 예측 성공을 위한 IOU를 0.5 이상으로 고정한 PASCAL VOC와 달리 COCO Challenge는 IOU를 다양한 범위로 설정하여 예측 성공 기준을 정함
- IOU 0.5 부터 0.05 씩 값을 증가시켜 0.95 까지 해당하는 IOU 별로 mAP를 계산([AP@\[.50 : .05 : .95\]](#)는 시작 IOU 기준 : 0.5, 증가 Step : 0.05, 최종 IOU : 0.95에 따른 AP값을 의미)
- 또한 크기의 유형(대/중/소)에 따른 mAP도 측정



# 데이터 세트와 알고리즘 별 mAP 수치 예시

## 1.1 오브젝트 디텍션 개요

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

### VOC 2007 YOLO V2

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	<b>52.1</b>
<i>One-stage methods</i>							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	<b>40.8</b>	<b>61.1</b>	<b>44.1</b>	<b>24.1</b>	<b>44.2</b>	51.2

### COCO YOLO V2

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	<b>52.1</b>
<i>One-stage methods</i>							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	<b>40.8</b>	<b>61.1</b>	<b>44.1</b>	<b>24.1</b>	<b>44.2</b>	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

### COCO YOLO V3

많은 Detection과 Segmentation 딥러닝 패키지가 아래 Dataset들을 기반으로 Pretrained되어 배포

PASCAL VOC



MS COCO



Google Open Images



XML Format  
20개의 오브젝트 카테고리

json Format  
80개의 오브젝트 카테고리

csv Format  
600개의 오브젝트 카테고리

<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/>



Visual Object Classes Challenge 2012 (VOC2012)



[click on an image to see the annotation]

### Classification /Detection

20 classes



### Segmentation

Image



Objects



Class



### Action Classification

10 action classes + "other"



### Person Layout

Image



Person Layout

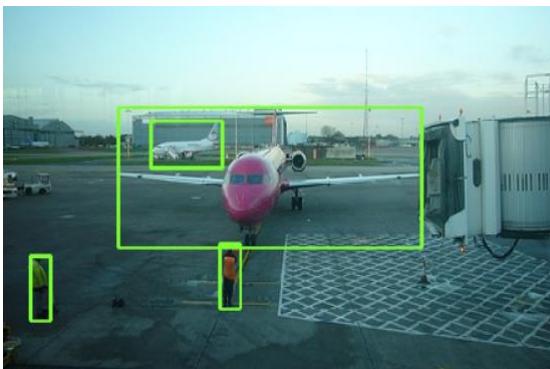


# Annotation 이란?

## 1.1 오브젝트 디텍션 개요

이미지의 Detection 정보를 별도의 설명 파일로 제공되는 것을 일반적으로 Annotation이라고 함  
Annotation은 Object의 Bounding Box 위치나 Object 이름 등을 특정 포맷으로 제공함

원본 이미지에 Bounding Box 시각화



원본 이미지



```
<annotation>
  <folder>VOC2012</folder>
  <filename>2007_000032.jpg</filename>
  <source>
    <database>The VOC2007 Database</database>
    <annotation>PASCAL VOC2007</annotation>
    <image>flickr</image>
  </source>
  <size>
    <width>500</width>
    <height>281</height>
    <depth>3</depth>
  </size>
  <segmented>1</segmented>
  <object>
    <name>aeroplane</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>104</xmin>
      <ymin>78</ymin>
      <xmax>375</xmax>
      <ymax>183</ymax>
    </bndbox>
  </object>
  <object>
    <name>aeroplane</name>
    <pose>Left</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>133</xmin>
      <ymin>88</ymin>
      <xmax>197</xmax>
      <ymax>123</ymax>
    </bndbox>
  </object>
</annotation>
```

VOC 2012기준

VOCdevkit

└ VOC2012

Annotations	Xml포맷이며, 개별 xml파일은 한 개 image에 대한 Annotation정보를 가지고 있음. 확장자 xml을 제외한 파일명은 imgae 파일명(확장자 jpg를 제외한)과 동일하게 맵핑
ImageSet	어떤 이미지를 train, test, val에 사용할 것인지에 대한 맵핑 정보를 개별 오브젝트별로 파일을 가지고 있음
JPEGImages	Detection과 Segmentation에 사용될 원본 이미지
SegmentationClass	Segmentic Segmentation에서 사용될 masking 이미지
SegmentationObject	Instance Segmentation에 사용될 masking 이미지

# Annotation 파일 예시

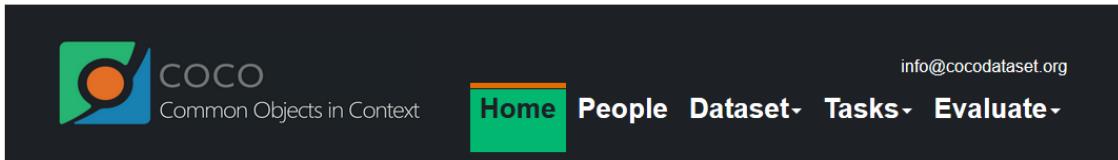
## 1.1 오브젝트 디텍션 개요

Annotation 파일 2007\_000032.xml 파일 일부



2007\_000032.jpg 파일에 대한  
Annotation 정보를 가지고 있음

```
<annotation>
    <folder>VOC2012</folder>
    <filename>2007_000032.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
    </source>
    <size>
        <width>500</width>
        <height>281</height>
        <depth>3</depth>
    </size>
    <segmented>1</segmented>
    <object> ←개별 Object 정보
        <name>aeroplane</name>
        <pose>Frontal</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>104</xmin>
            <ymin>78</ymin>
            <xmax>375</xmax>
            <ymax>183</ymax>
        </bndbox>
    </object>
    <object> ←개별 Object 정보
        <name>aeroplane</name>
        <pose>Left</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>133</xmin>
            <ymin>88</ymin>
            <xmax>197</xmax>
            <ymax>123</ymax>
        </bndbox>
    </object>
```



### News

- We are pleased to announce the COCO 2019 Detection, Keypoint, Panoptic, and DensePose Challenges.
- The new rules and awards for this year challenges encourage innovative methods.
- Results to be announced at the [Joint COCO and Mapillary Recognition ICCV workshop](#).

### What is COCO?



COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints

### Collaborators

Tsung-Yi Lin Google Brain  
Genevieve Patterson MSR, Trash TV

Matteo R. Ronchi Caltech  
Yin Cui Google  
Michael Maire TTI-Chicago  
Serge Belongie Cornell Tech

Lubomir Bourdev WaveOne, Inc.  
Ross Girshick FAIR  
James Hays Georgia Tech  
Pietro Perona Caltech  
Deva Ramanan CMU  
Larry Zitnick FAIR  
Piotr Dollár FAIR

### Sponsors



- 80개 Object Category
- 300K의 Image들과 1.5Million 개의 object들
- (하나의 image에 평균 5개의 Object들로 구성)
- Tensorflow Object Detection API 및 많은 오픈 소스 계열의 주요 패키지들은 COCO Dataset으로 Pretrained된 모델을 제공함

# MS-COCO Dataset 오브젝트 카테고리

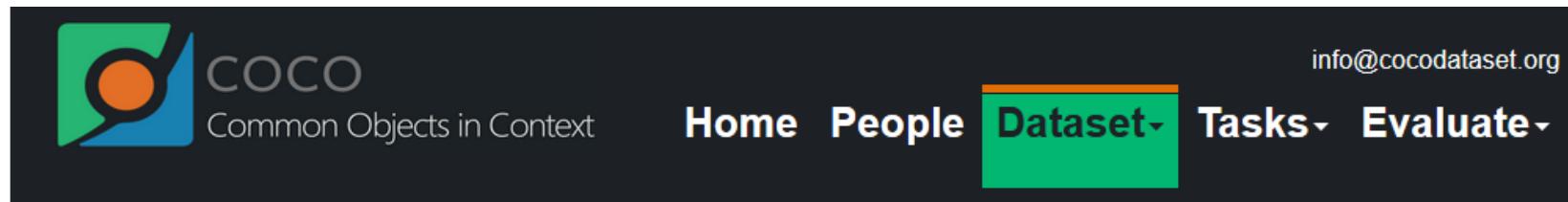
## 1.1 오브젝트 디텍션 개요

ID	COCO Paper	COCO 2017	Group
1	person	person	person
2	bicycle	bicycle	vehicle
3	car	car	vehicle
4	motorcycle	motorcycle	vehicle
5	airplane	airplane	vehicle
6	bus	bus	vehicle
7	train	train	vehicle
8	truck	truck	vehicle
9	boat	boat	vehicle
10	traffic light	traffic light	vehicle
11	fire hydrant	fire hydrant	outdoor
12	street sign	—	outdoor
13	stop sign	stop sign	outdoor
14	parking meter	parking meter	outdoor
15	bench	bench	outdoor

.....  
80개의 오브젝트 카테고리

ID	COCO Paper	COCO 2017	Group
77	cell phone	cell phone	electronic
78	microwave	microwave	appliance
79	oven	oven	appliance
80	toaster	toaster	appliance
81	sink	sink	appliance
82	refrigerator	refrigerator	appliance
83	blender	—	appliance
84	book	book	indoor
85	clock	clock	indoor
86	vase	trvaset	indoor
87	scissors	scissors	indoor
88	teddy bear	teddy bear	indoor
89	hair drier	hair drier	indoor
90	toothbrush	toothbrush	indoor
91	hair brush	—	indoor

<http://cocodataset.org/#download>



### Tools

[COCO API](#)

### Images

[2014 Train images \[83K/13GB\]](#)  
[2014 Val images \[41K/6GB\]](#)  
[2014 Test images \[41K/6GB\]](#)  
[2015 Test images \[81K/12GB\]](#)  
[2017 Train images \[118K/18GB\]](#)  
[2017 Val images \[5K/1GB\]](#)  
[2017 Test images \[41K/6GB\]](#)  
[2017 Unlabeled images \[123K/19GB\]](#)

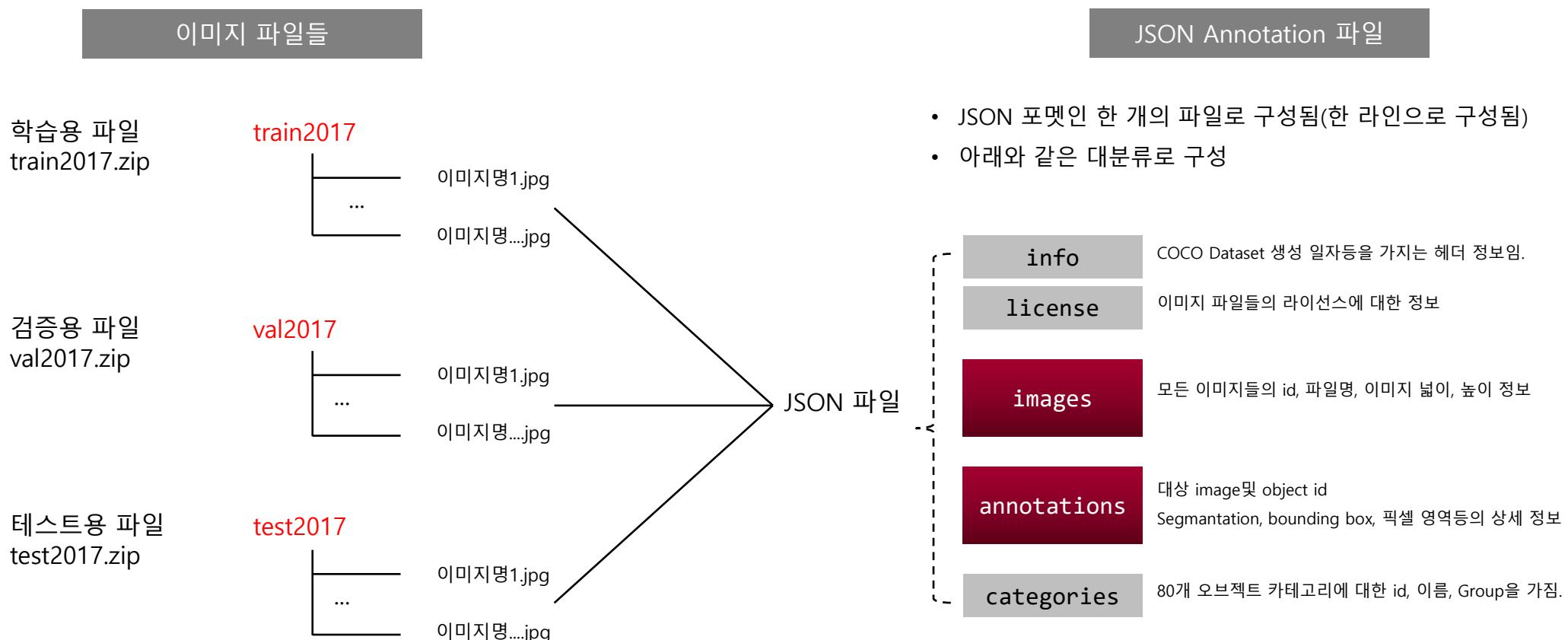
### Annotations

[2014 Train/Val annotations \[241MB\]](#)  
[2014 Testing Image info \[1MB\]](#)  
[2015 Testing Image info \[2MB\]](#)  
[2017 Train/Val annotations \[241MB\]](#)  
[2017 Stuff Train/Val annotations \[1.1GB\]](#)  
[2017 Panoptic Train/Val annotations \[821MB\]](#)  
[2017 Testing Image info \[1MB\]](#)  
[2017 Unlabeled Image info \[4MB\]](#)

# MS-COCO Dataset 구성

## 1.1 오브젝트 디텍션 개요

COCO 2017 데이터 세트 기준



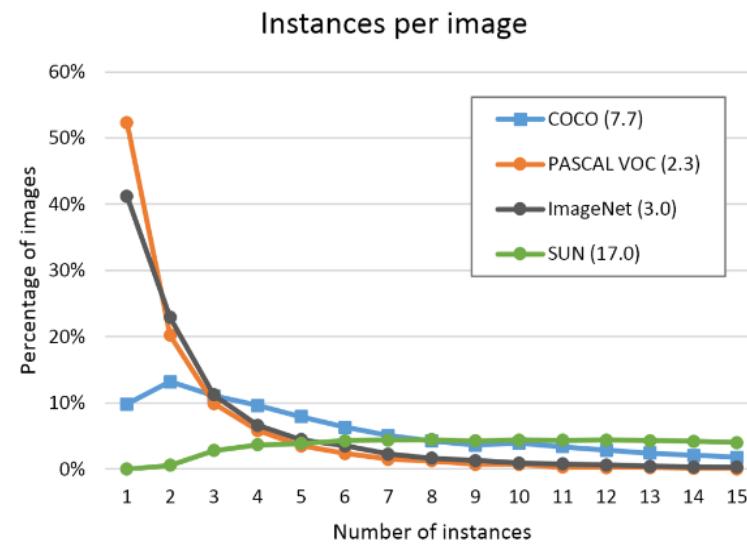
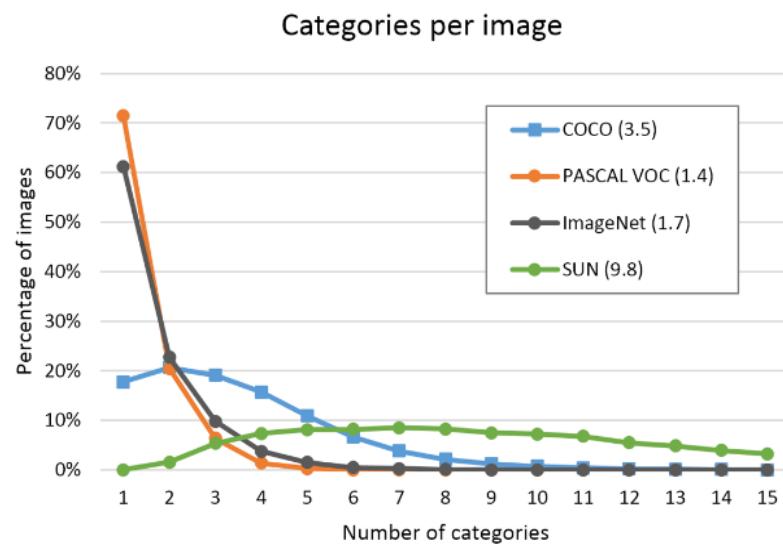
### JSON 파일의 images 대분류 예시

```
"images": [  
    {  
        "license": 1,  
        "file_name": "00000037777.jpg",  
        "coco_url": "http://images.cocodataset.org/val2017/00000037777.jpg",  
        "height": 230,  
        "width": 352,  
        "date_captured": "2013-11-14 20:55:31",  
        "flickr_url": "http://farm9.staticflickr.com/8429/7839199426_f6d48aa585_z.jpg",  
        "id": 37777  
    },  
    ...  
]
```

### JSON 파일의 annotations 대분류 예시

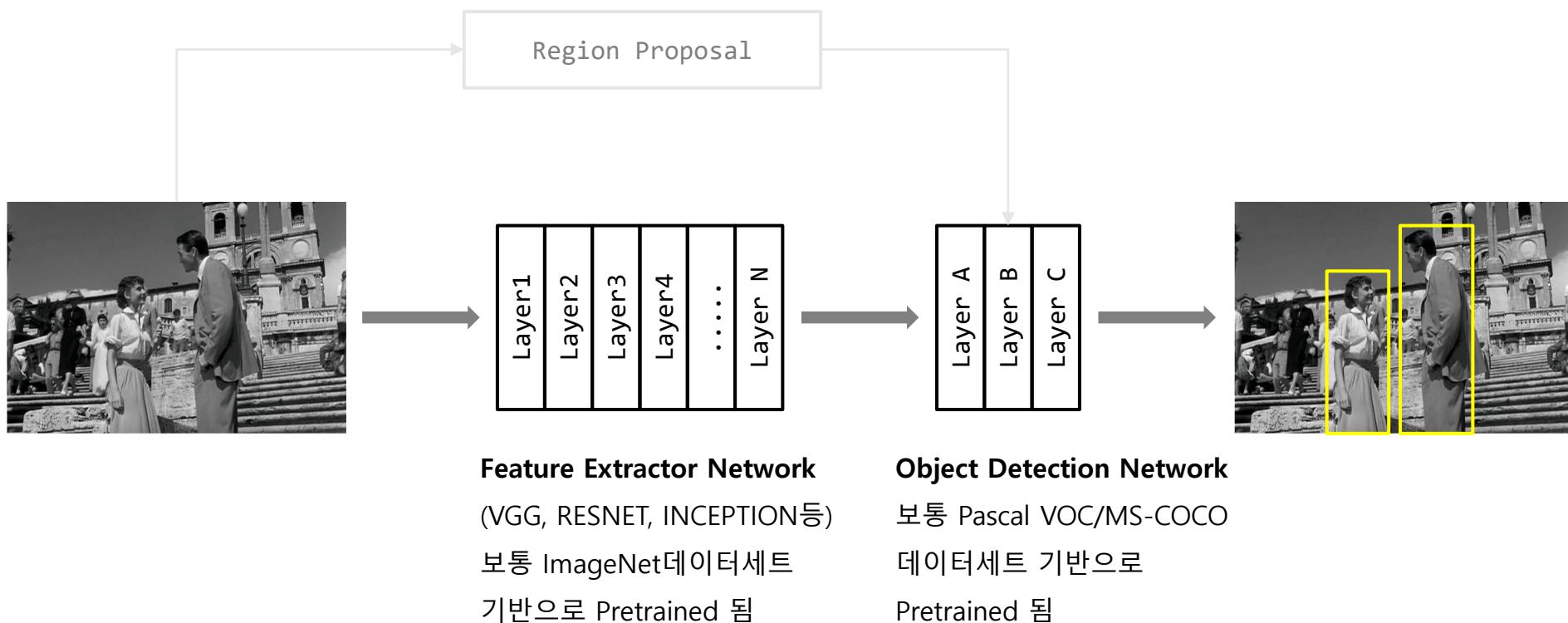
```
"annotations": [  
    {  
        "segmentation": [[510.66,423.01,511.72,420.03,...,510.45,423.01]],  
        "area": 702.1057499999998,  
        "iscrowd": 0,  
        "image_id": 289343,  
        "bbox": [473.07,395.93,38.65,28.67],  
        "category_id": 18,  
        "id": 1768  
    },  
    ...  
]
```

COCO Dataset은 이미지 한 개에 여러 오브젝트들을 가지고 있으며 타 데이터 세트에 비해 난이도가 높은 데이터를 제공



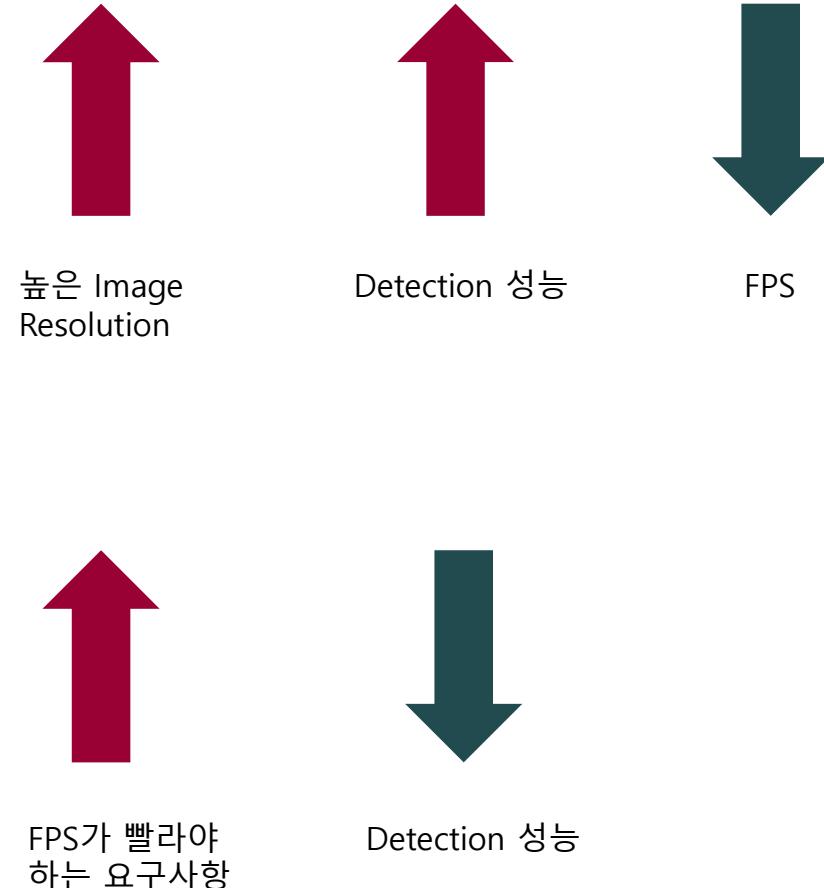
# Object Detection 네트워크 개요

## 1.1 오브젝트 디텍션 개요



# Image Resolution, FPS, Detection 성능 상관 관계

## 1.1 오브젝트 디텍션 개요



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

VOC 2007 for YOLOv2

# 1. 오브젝트 디텍션

---

1.1 오브젝트 디텍션 개요

## 1.2 OpenCV 개요

1.3 R-CNN, Fast R-CNN, Faster R-CNN

1.4 SSD (Single Shot MultiBox Detector)

1.5 YOLO (You Only Look Once) 시리즈

1.6 RetinaNet

---

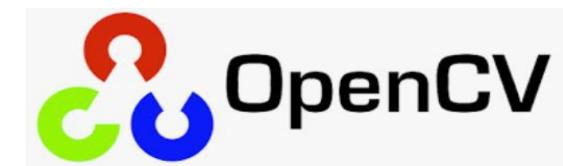
PIL  
(Python Image Library)



Scikit Image



OpenCV



- 주로 이미지 처리만을 위해 사용
- 처리 성능이 상대적으로 느림

- 파이썬 기반의 전반적인 컴퓨터 비전 기능 제공
- 사이파이(Scipy)기반

- 오픈소스 기반의 최고 인기 컴퓨터 비전 라이브러리
- 컴퓨터 비전 기능 일반화에 크게 기여 (어려운 기능도 API 몇 줄로 간단하게 구현)
- C++기반이나 Python도 지원 (Java C# 등 다양한 언어 지원)



- 인텔이 초기 개발 주도
- Window, 리눅스, Mac OS X, 안드로이드, i-OS 등 다양한 플랫폼에서 사용 가능
- 방대한 컴퓨터 비전 관련 라이브러리와 손쉬운 인터페이스 제공

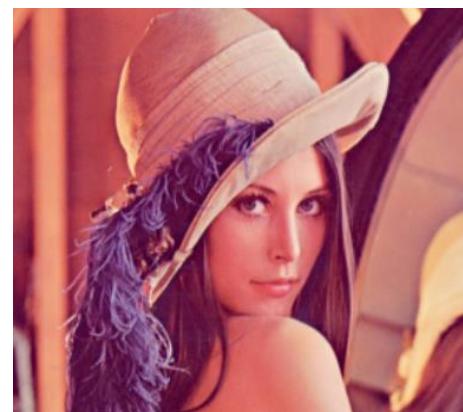
### imread()를 이용한 이미지 로딩

OpenCV에서 이미지 로딩은 `imread('파일명')`을 이용. `imread('파일명')`은 파일을 읽어 넘파이 array로 변환.

OpenCV에서 `imread()`를 이용하여 이미지 로딩 시 가장 주의해야 할 점은 OpenCV가 이미지를 RGB 형태가 아닌 BGR 형태로 로딩하기 때문에 색감이 원본 이미지와 다르게 나타난다는 것임.

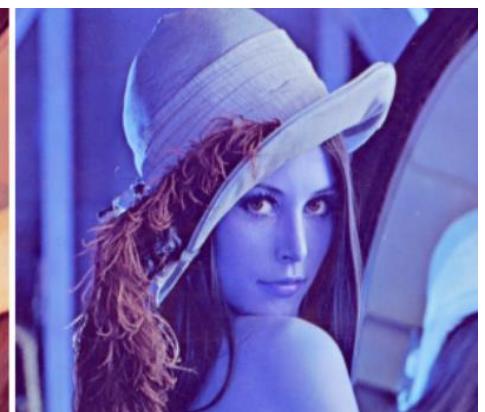
```
import cv2  
  
import matplotlib.pyplot as plt  
  
img_array = cv2.imread('파일명')  
  
plt.imshow(img_array)
```

RGB



원본 이미지

BGR



OpenCV.imread()로  
읽은 이미지

cvtColor()를 이용하여 BGR을 RGB로 변환

OpenCV에서 imread('파일명')을 이용하여 로딩된 이미지 배열을 BGR 형태의 배열이므로 이를 RGB 형태의 배열로 변환하려면 cvtColor(이미지 배열, cv2.COLOR\_BGR2RGB)를 이용함.

```
import cv2  
import matplotlib.pyplot as plt  
  
bgr_img_array = cv2.imread('파일명')  
rgb_img_array = cv2.cvtColor(bgr_img_array, cv2.COLOR_BGR2RGB)  
plt.imshow(rgb_img_array)
```



원본 이미지

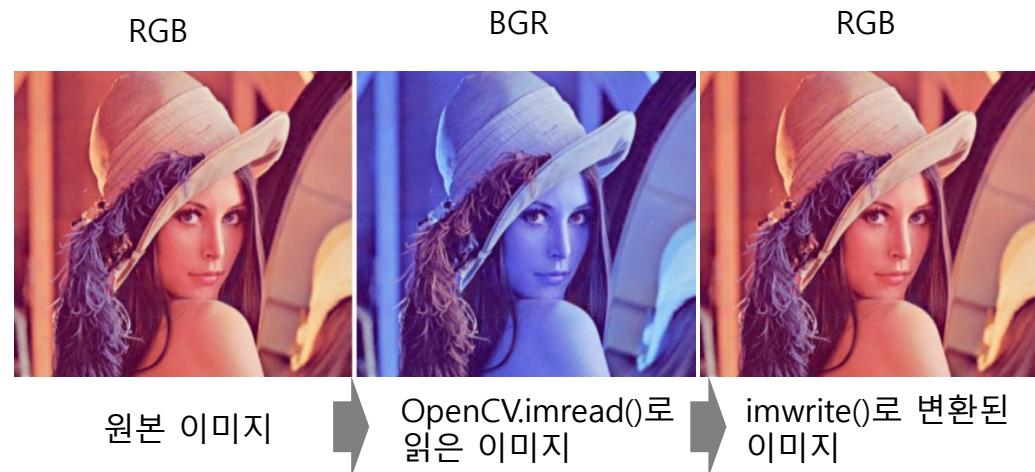
OpenCV.imread()로  
읽은 이미지

cvtColor()로  
변환된  
이미지

### imwrite()를 이용하여 파일에 쓰기

OpenCV에서 메모리에 있는 이미지 배열을 다시 파일에 저장하려면 `imwrite('출력파일명', 이미지배열)`을 이용함. 이때 `imread()`로 인해 BGR 형태로 되어 있는 이미지 배열을 다시 RGB 형태로 변환하여 저장함. 따라서 `imread()`로 읽고, `imwrite()`로 출력한 이미지 파일은 다시 RGB 형태의 파일이 됨.

```
import cv2  
import matplotlib.pyplot as plt  
  
img_array = cv2.imread('파일명')  
cv2.imwrite('출력파일명', img_array)
```



### OpenCV Windows Frame 인터페이스

OpenCV는 OS의 Window Frame과 바로 인터페이스하여 사용할 수 있는 여러 기능을 제공. 하지만 이들 기능을 사용하려면 Window Frame 생성이 가능한 GUI 개발 환경에서만 가능 (Window GUI, Linux X-windows 등). 주피터 노트북 기반에서는 사용 시 오류 발생

**cv2.imshow(이미지 array)**는 이미지 배열을 window frame에 보여줌

**cv2.waitKey()** : 키보드 입력이 있을 때까지 무한 대기

**cv2.destroyAllWindows()** : 화면의 윈도우 프레임 모두 종료

주피터 노트북 기반에서는 이미지 배열의 시각화에 matplotlib을 사용

OpenCV의 VideoCapture클래스는 동영상을 개별 Frame으로 하나씩 읽어(Read)들이는 기능을 제공

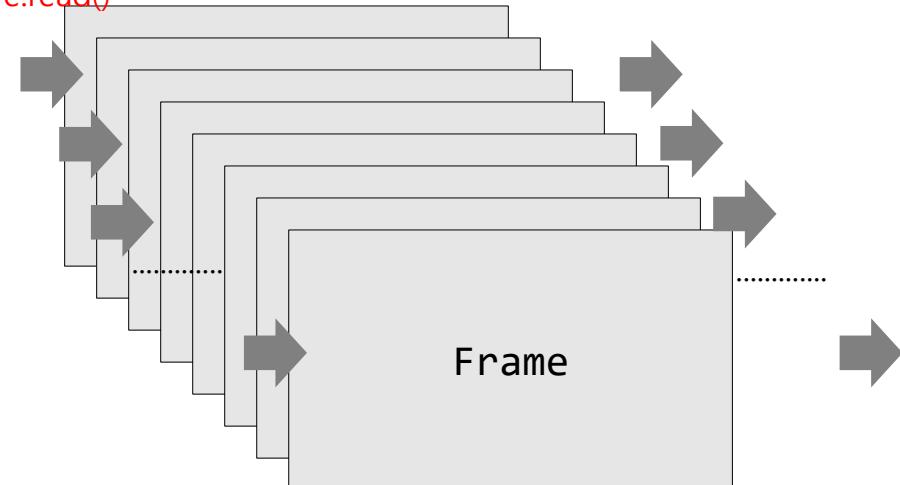
VideoWriter는 VideoCapture로 읽어들인 개별 Frame을 동영상 파일로 Write 수행.

```
cap = cv2.VideoCapture(video_input_path)
.....
.....
vid_write = cv2.VideoWrite(video_output_path,...)

while True:
    hasFrame, img_frame = cap.read()
    if not hasFrame:
        print('더 이상 처리할 frame이 없습니다.')
        break
    vid_write.write(img_frame)
```

videoCapture.read()

videoWrite.write()



VideoCapture 객체는 생성 인자로 입력 video 파일 위치를 받아 생성

```
cap = cv2.VideoCapture(video_input_put)
```

VideoCapture 객체는 입력 video 파일의 다양한 속성을 가져 올 수 있음

영상 Frame 너비 : cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH)

영상 Frame 높이 : cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT)

영상 FPS : cap.get(cv2.CAP\_PROP\_FPS)

VideoCapture 객체의 read()는 마지막 Frame까지 차례로 Frame을 읽음

```
while True:
```

```
    hasFrame, img_frame = cap.read()
```

```
    if not hasFrame:
```

```
        print('더 이상 처리할 frame이 없습니다.')
```

```
        break
```

- VideoWriter 객체는 write할 동영상 파일 위치, Encoding코덱 유형, write fps 수치, frame 크기를 생성자로 입력 받아 이들 값에 따른 동영상 Write 수행
- VideoWriter는 write시 특정 포맷으로 동영상을 Encoding 할 수 있음(DIVX, MJPG, X264, WMV1, WMV2)

```
cap = cv2.VideoCapture(video_input_path)
```

```
codec = cv2.VideoWrite_fourcc(*'XVID')
```

```
vid_size = (round(cap.get(cv2.CAP_PROP_FRAME_WIDTH))), round(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
```

```
vid_fps = cap.get(cv2.CAP_PROP_FPS)
```

```
vid_write = cv2.VideoWriter(video_output_path, codec, vid_fps, vid_size)
```

# 1. 오브젝트 디텍션

---

1.1 오브젝트 디텍션 개요

1.2 OpenCV 개요

## **1.3 R-CNN, Fast R-CNN, Faster R-CNN**

1.4 SSD (Single Shot MultiBox Detector)

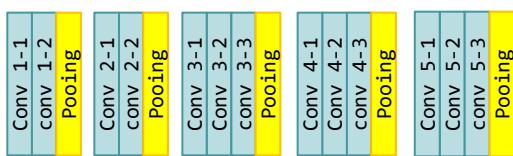
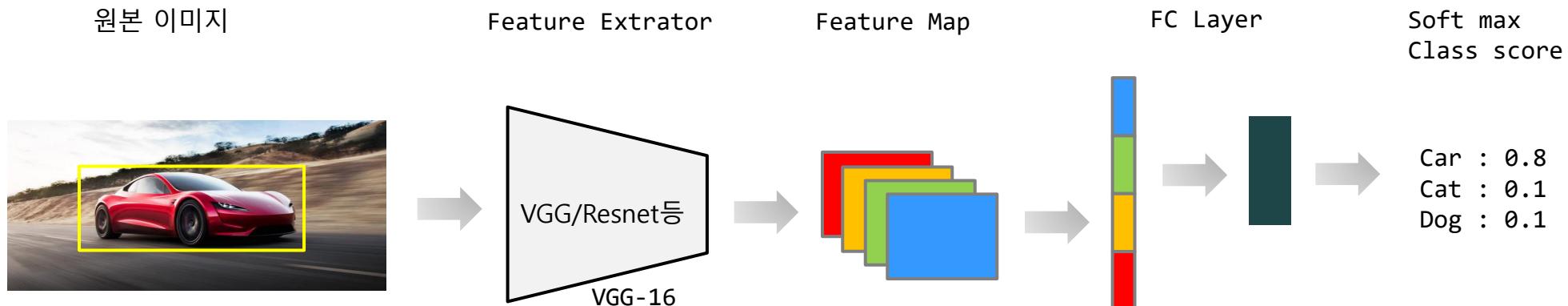
1.5 YOLO (You Only Look Once) 시리즈

1.6 RetinaNet

---

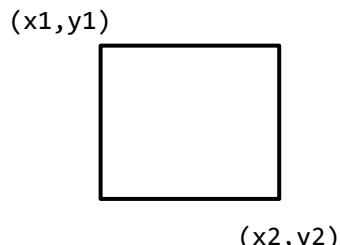
# Object Localization /Detection 개요

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



### Annotation 파일

```
<annotation>
  <folder>VOC2012</folder>
  <filename>2007_000032.jpg</filename>
  <source>
    <database>The VOC2007 Database</database>
    <annotation>PASCAL VOC2007</annotation>
    <image>flickr</image>
  </source>
  <size>
    <width>500</width>
    <height>381</height>
    <depth>3</depth>
  </size>
  <segmented>1</segmented>
  <object>
    <name>aeroplane</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>104</xmin>
      <ymin>78</ymin>
      <xmax>375</xmax>
      <ymax>183</ymax>
    </bndbox>
  </object>
</annotation>
```



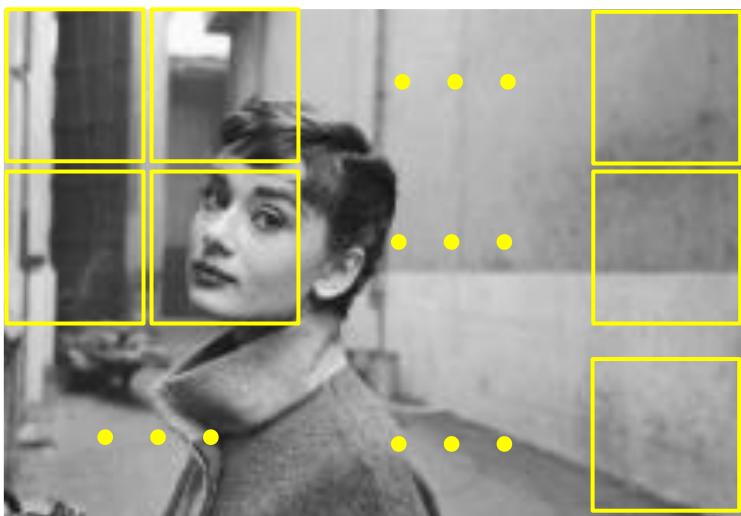
### Bounding Box Regression

	$x_1$	$x_2$	$y_1$	$y_2$	
Car	□	□	□	□	
Cat	□	□	□	□	
Dog	□	□	□	□	
.....	□	□	□	□	

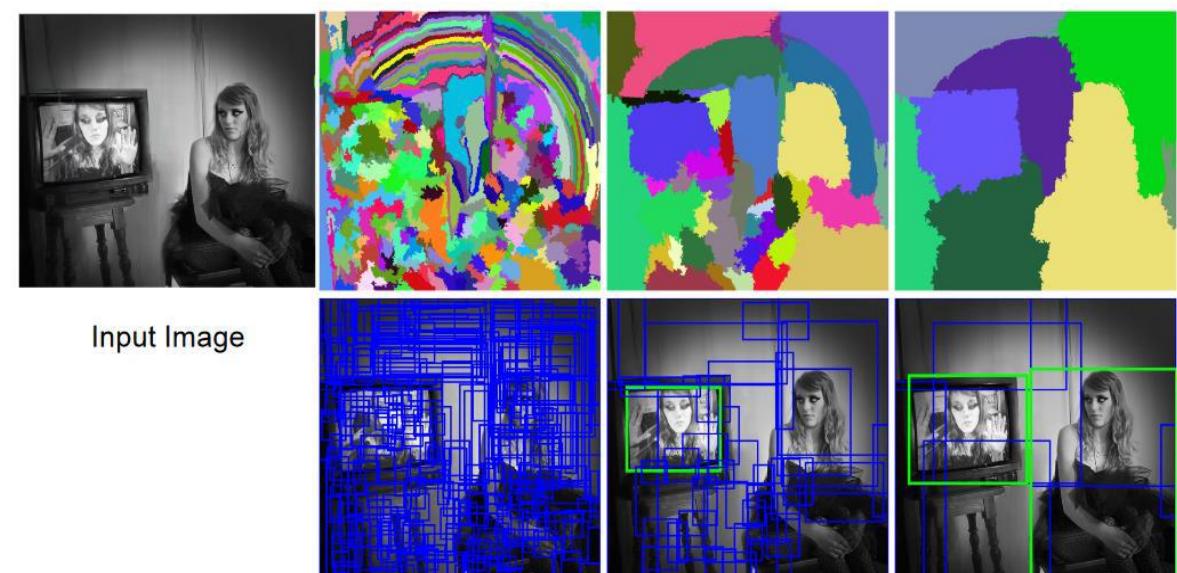
# Sliding Window 방식과 Region Proposal 방식

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

Sliding Window 방식

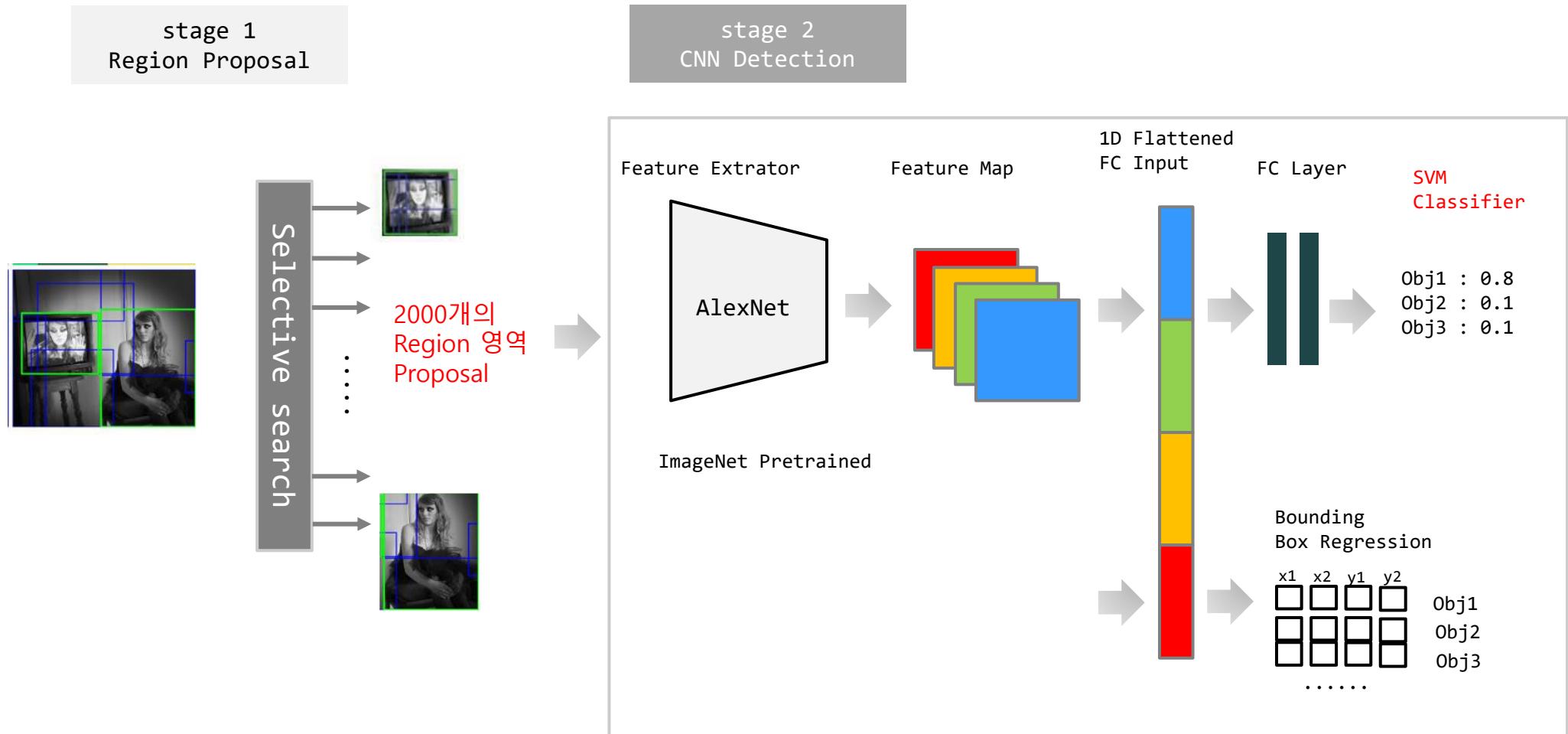


Region Proposal 방식



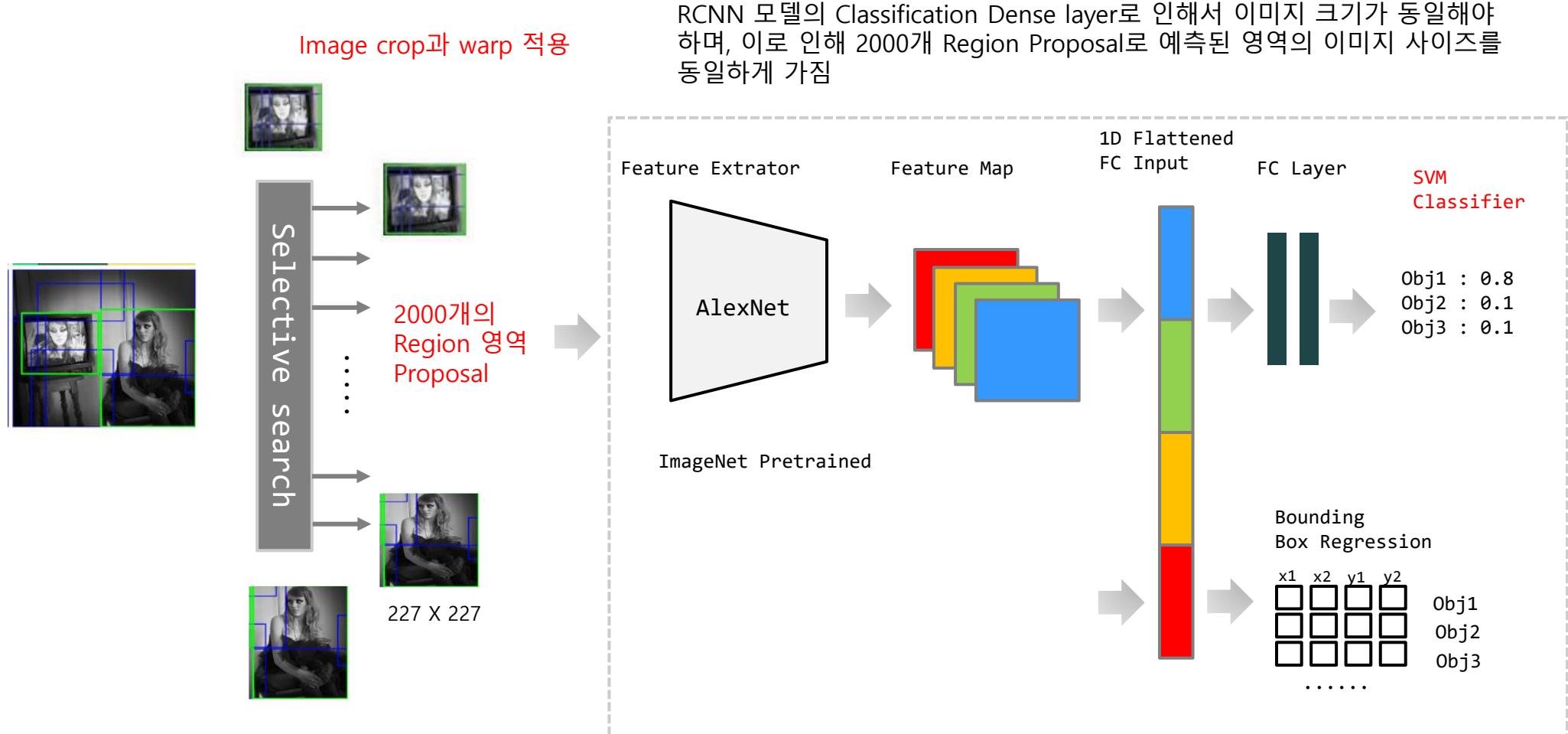
# Region Proposal 방식에 기반한 Object Detection-RCNN

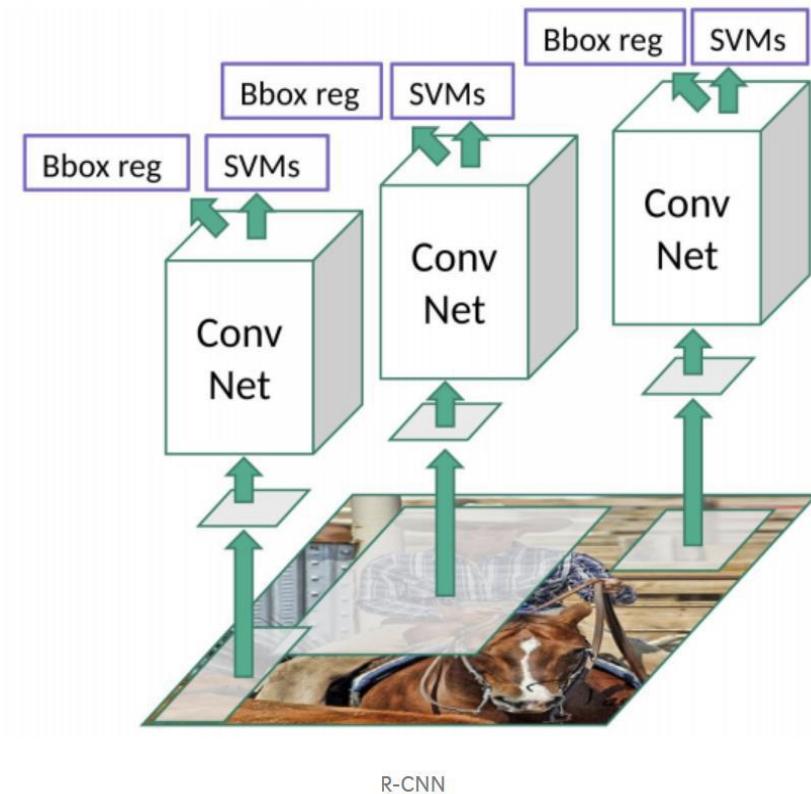
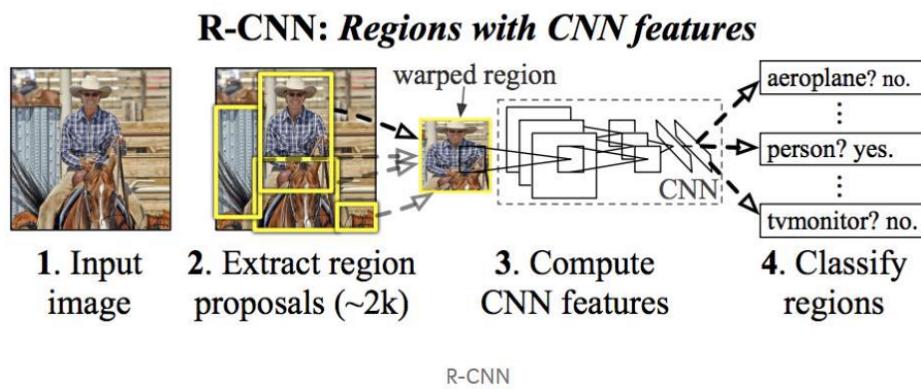
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



# Region Proposal 방식에 기반한 Object Detection-RCNN

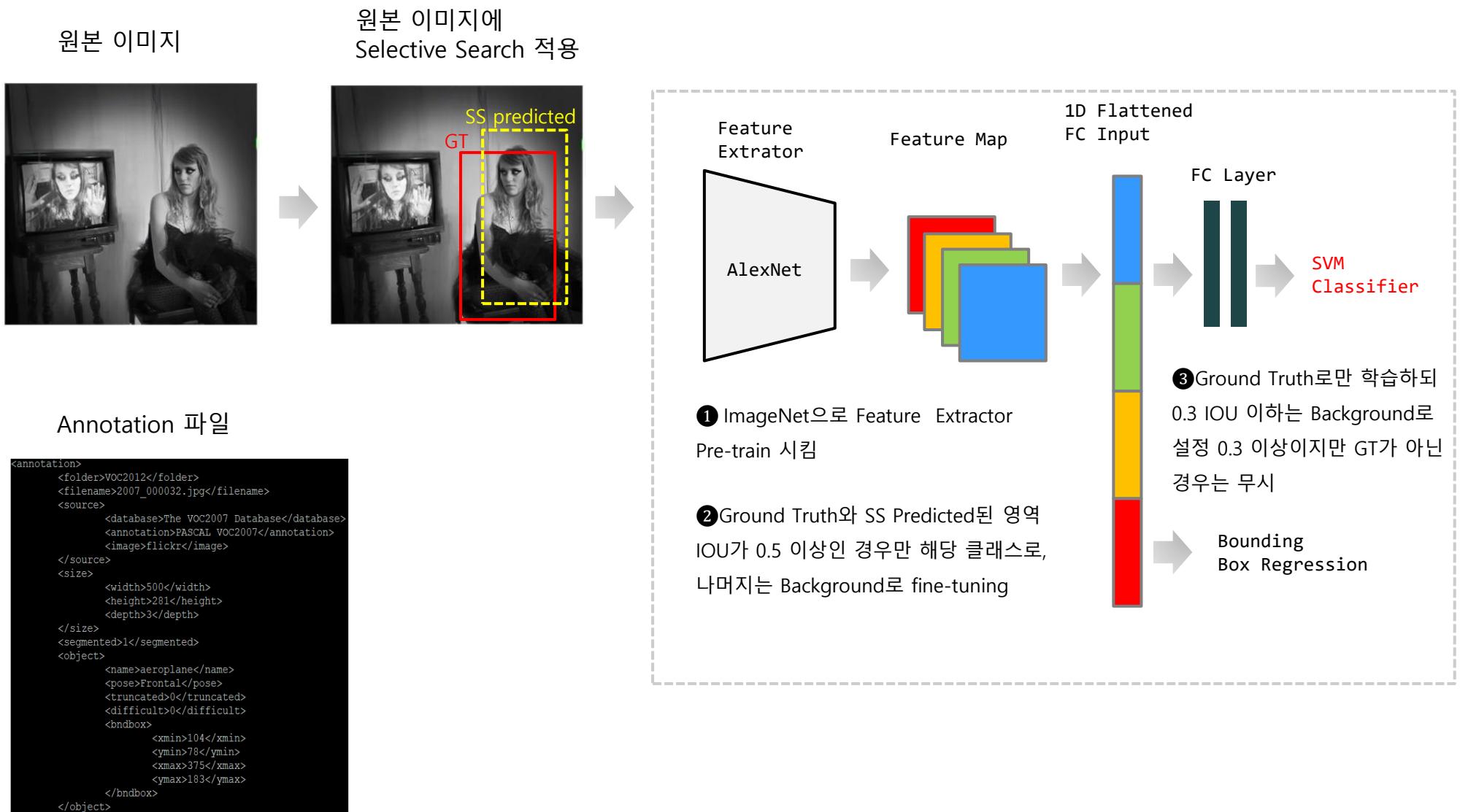
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN





# RCNN Training - Classification

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



# Bounding Box Regression

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

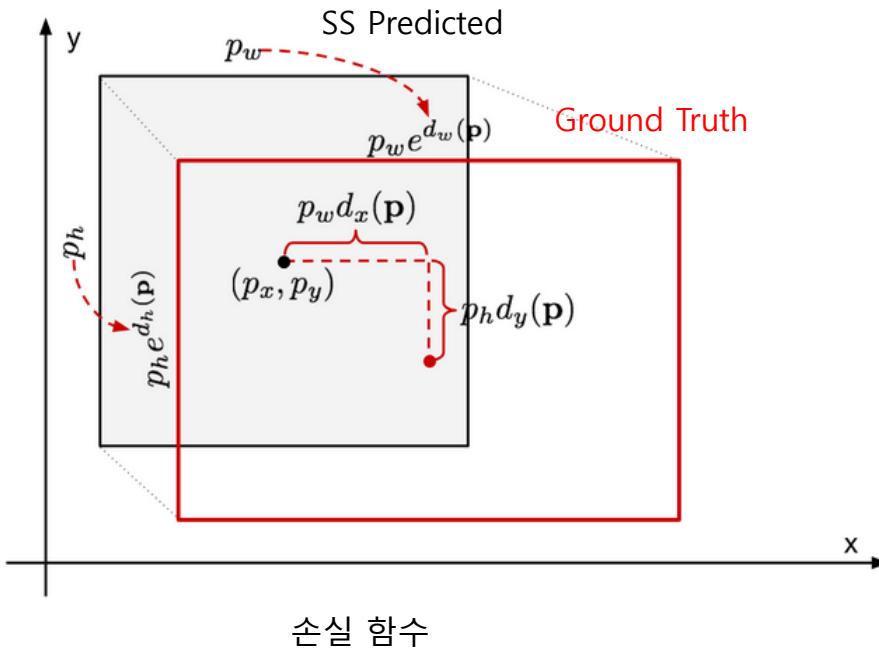
수정 예측 값

$$\hat{g}_x = p_w d_x(p) + p_x$$

$$\hat{g}_y = p_h d_y(p) + p_y$$

$$\hat{g}_w = p_w \exp(d_w(p))$$

$$\hat{g}_h = p_h \exp(d_h(p))$$



Target

$$t_x = (g_x - p_x)/p_w$$

$$t_y = (g_y - p_y)/p_h$$

$$t_w = \log(g_w/p_w)$$

$$t_h = \log(g_h/h)$$

$$\mathcal{L}_{reg} = \sum_{i \in \{x,y,w,h\}} (t_i - d_i(p))^2 + \lambda \| w \|^2$$

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [20] <sup>†</sup>	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [41]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [18] <sup>†</sup>	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	71.8	65.8	53.0	36.8	35.9	59.7	60.0	69.9	27.9	50.6	41.4	70.0	62.0	69.0	58.1	29.5	59.4	39.3	61.2	52.4	53.7

### 높은 Detection 정확도

동시대의 다른 알고리즘 대비 매우 높은 Detection 정확도

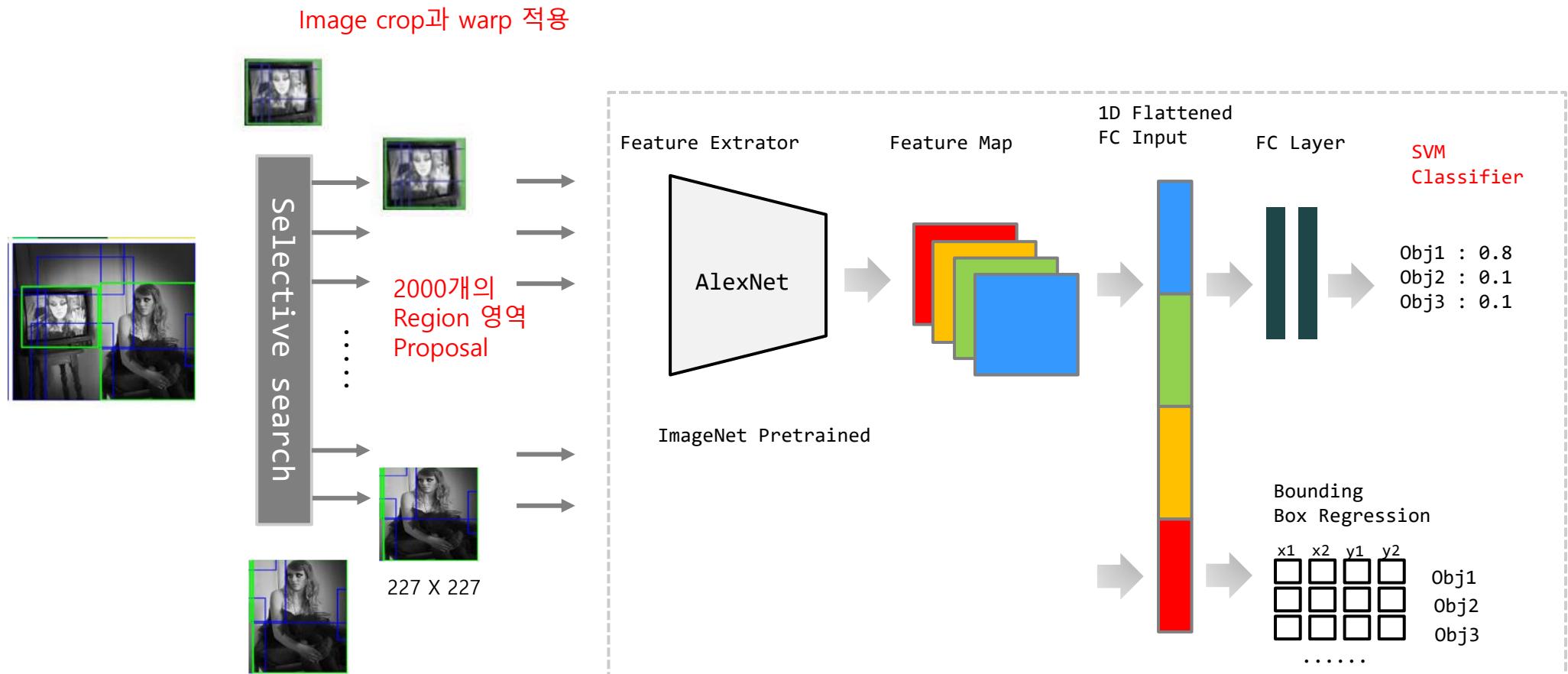
**But**

너무 느린 Detection 시간과 복잡한 아키텍처 및 학습 프로세스

- 하나의 이미지마다 selective search를 수행하여 2000개의 region 영역 이미지를 도출
- 개별 이미지별로 2000개씩 생성된 region 이미지를 CNN Feature map 생성
- 각기 따로 노는 구성 요소들. Selective search, CNN Feature Extractor, SVM과 Bounding box regressor로 구성되어 복잡한 프로세스를 거쳐서 학습 및 Object Detection이 되어야 함.

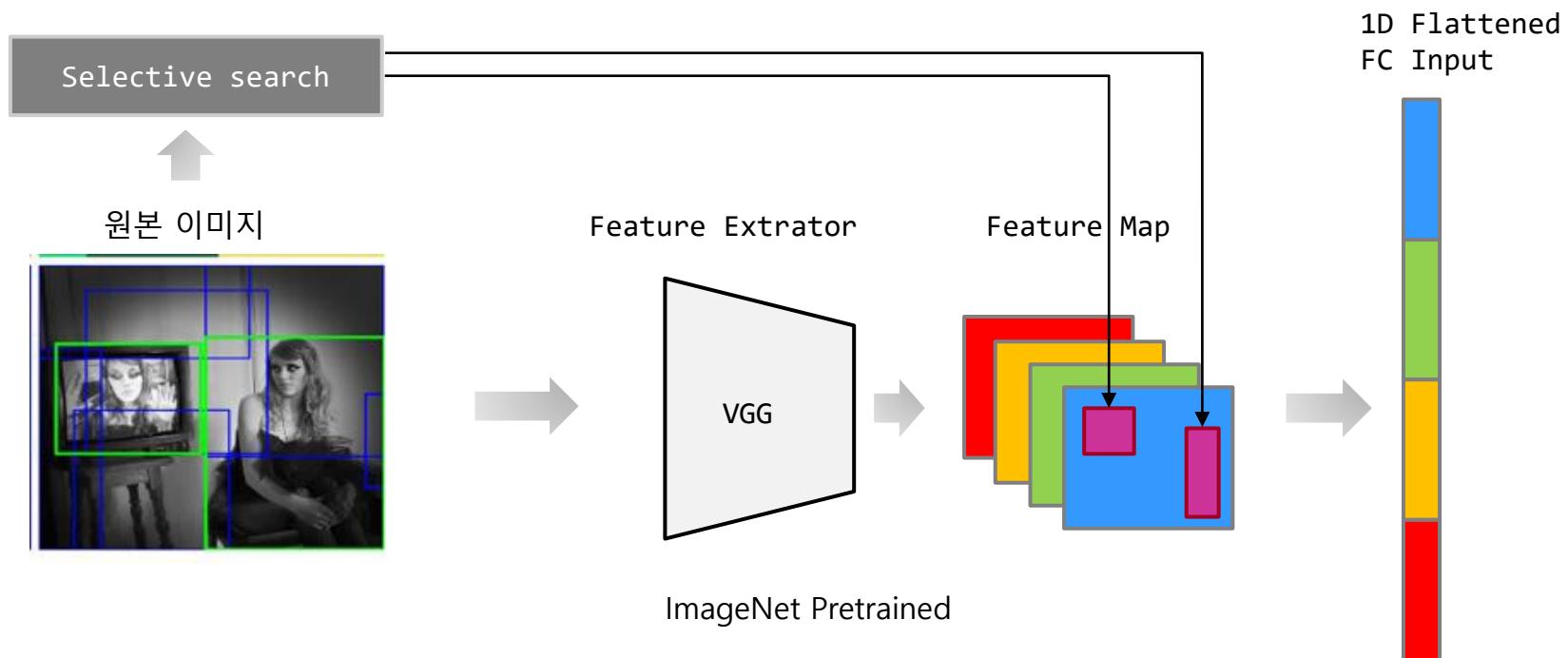
1장의 이미지를 Object Detection 하는데 약 50초 소요

- Deep Learning 기반 Object Detection 성능 입증
- Region Proposal 기반 성능 입증
- Detection 수행 시간을 줄이고 복잡하게 분리된 개별 아키텍처를 통합 할 수 있는  
방안 연구 매진

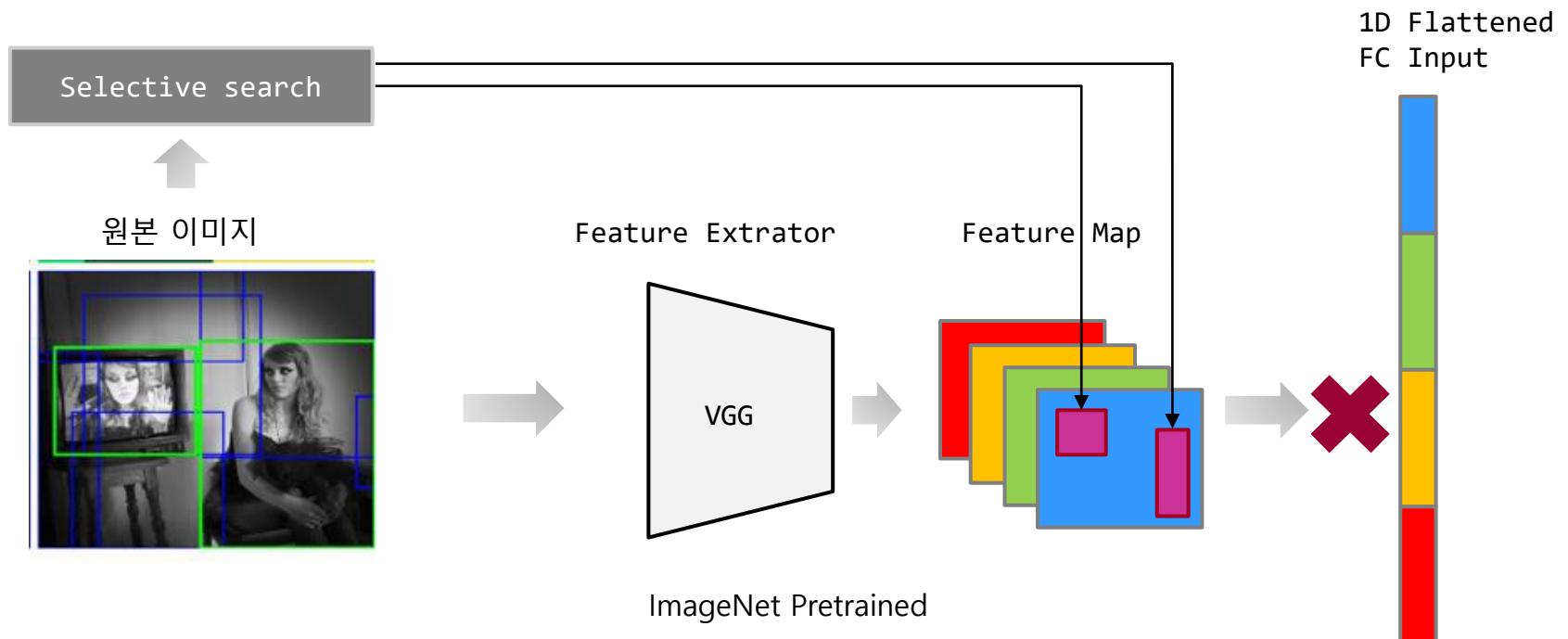


- 2000개의 Region 영역 이미지가 CNN으로 입력되면서 Object Detection 수행시간이 오래 걸림
- Region 영역 이미지가 Crop/Warp 되어야 함.

2000개의 Region Proposal 이미지를 CNN으로 Feature Extraction 하지 않고 원본 이미지만 CNN으로 Feature Map 생성 뒤에 원본 이미지의 Selective search로 추천된 영역의 이미지만 Feature Map으로 매핑하여 별도 추출



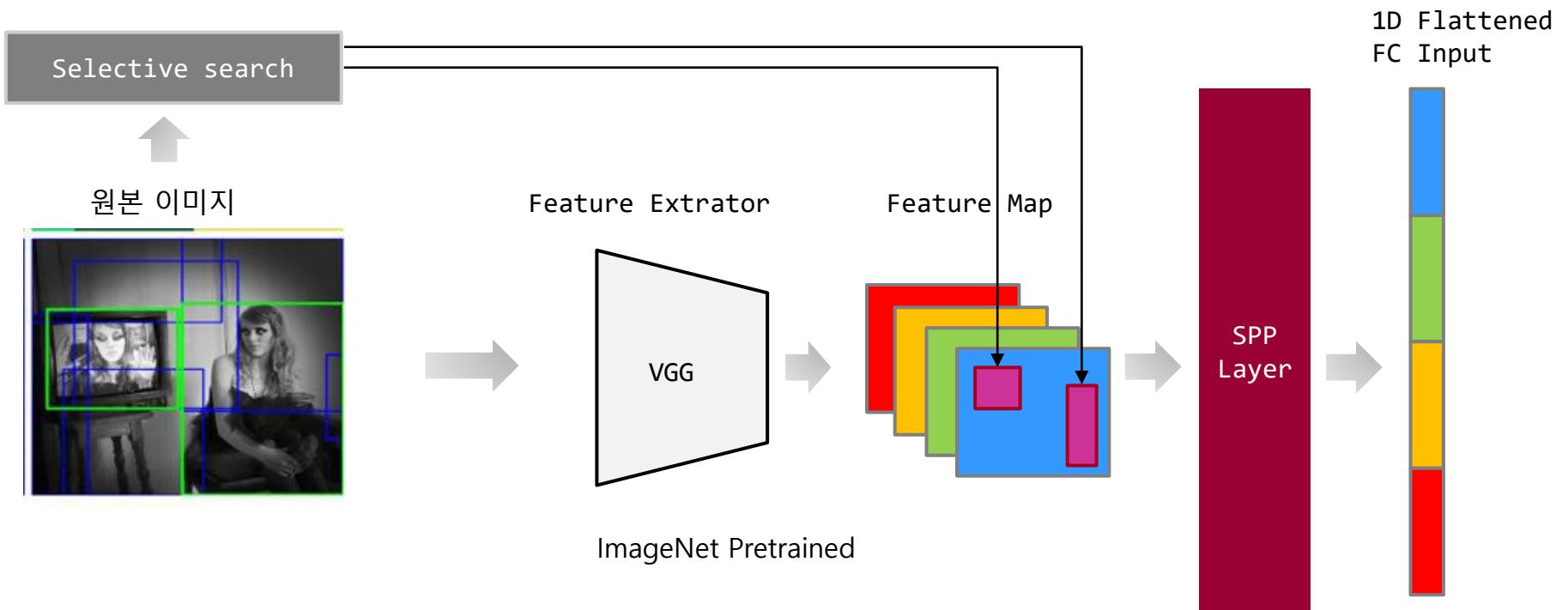
CNN은 서로 다른 사이즈의 Image를 수용하지 않는데, 가장 큰 이유는 Flatten Fully Connection Input의 크기가 고정이 되어야 하기 때문임.



# 서로 다른 크기의 Region Proposal 이미지 개선 방안

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

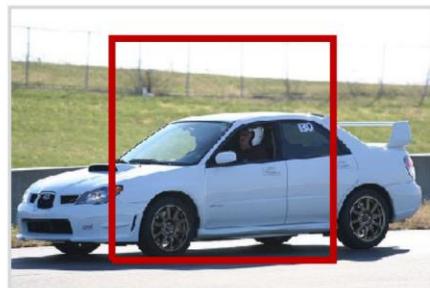
Feature Map으로 투영된 서로 다른 크기를 가진 Region Proposal 이미지를 SPP Net의 고정된 크기 Vector로 변환하여 FC에 1D Flattened 된 input 제공



# SPP(Spatial pyramid Pooling)

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

SPP는 CNN Image classification에서 서로 다른 이미지의 크기를 고정된 크기로 변환하는 기법으로 소개



crop



warp



SPP는 오래전부터 컴퓨터 비전 영역에서 활용된 Spatial Pyramid Matching 기법에 근간을 둠

Bag of Visual words

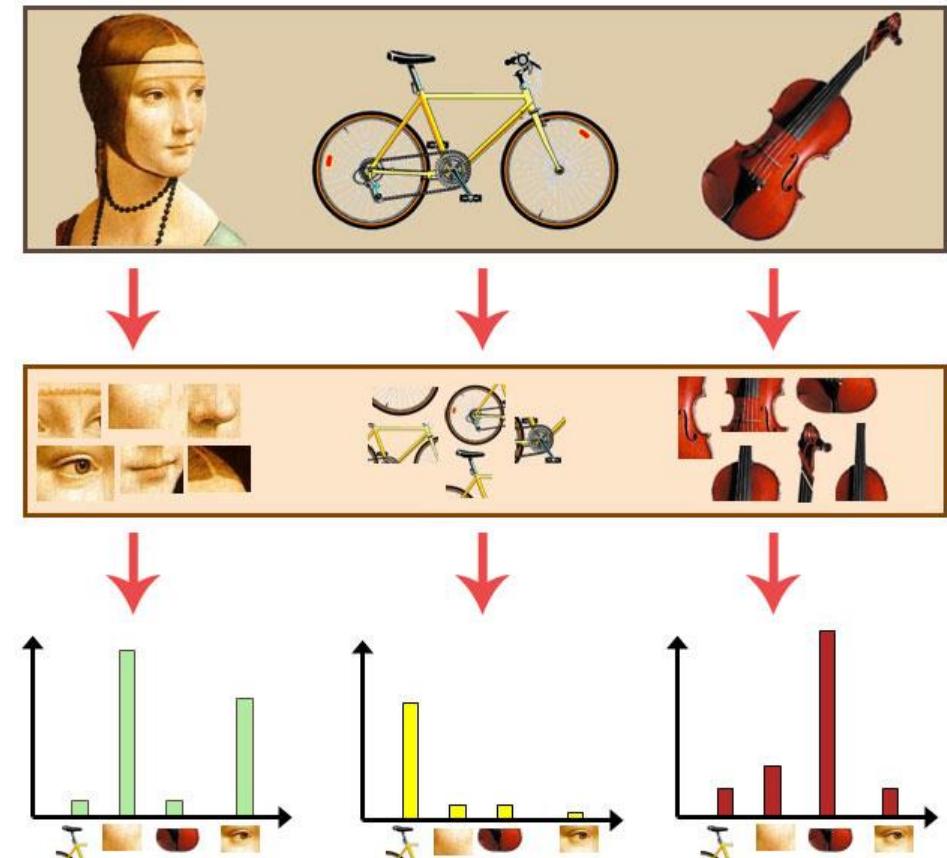
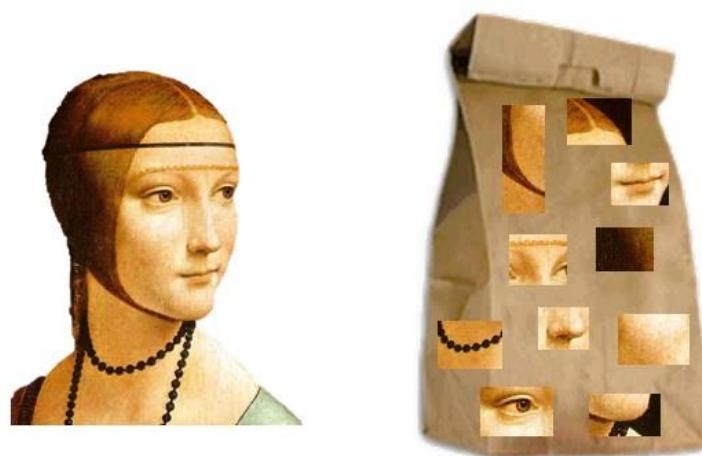


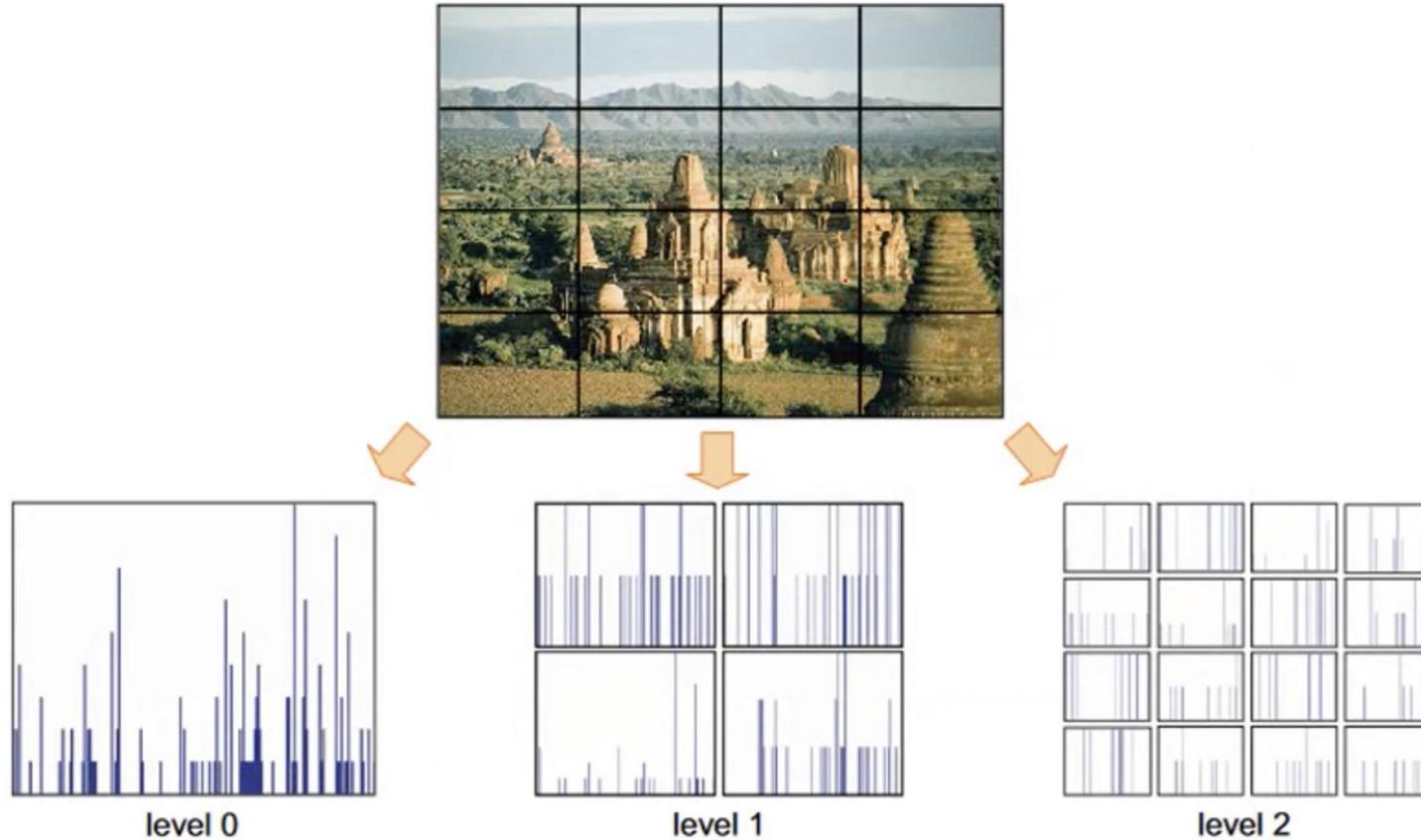
Spatial Pyramid Matching

# Bag of Visual words

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

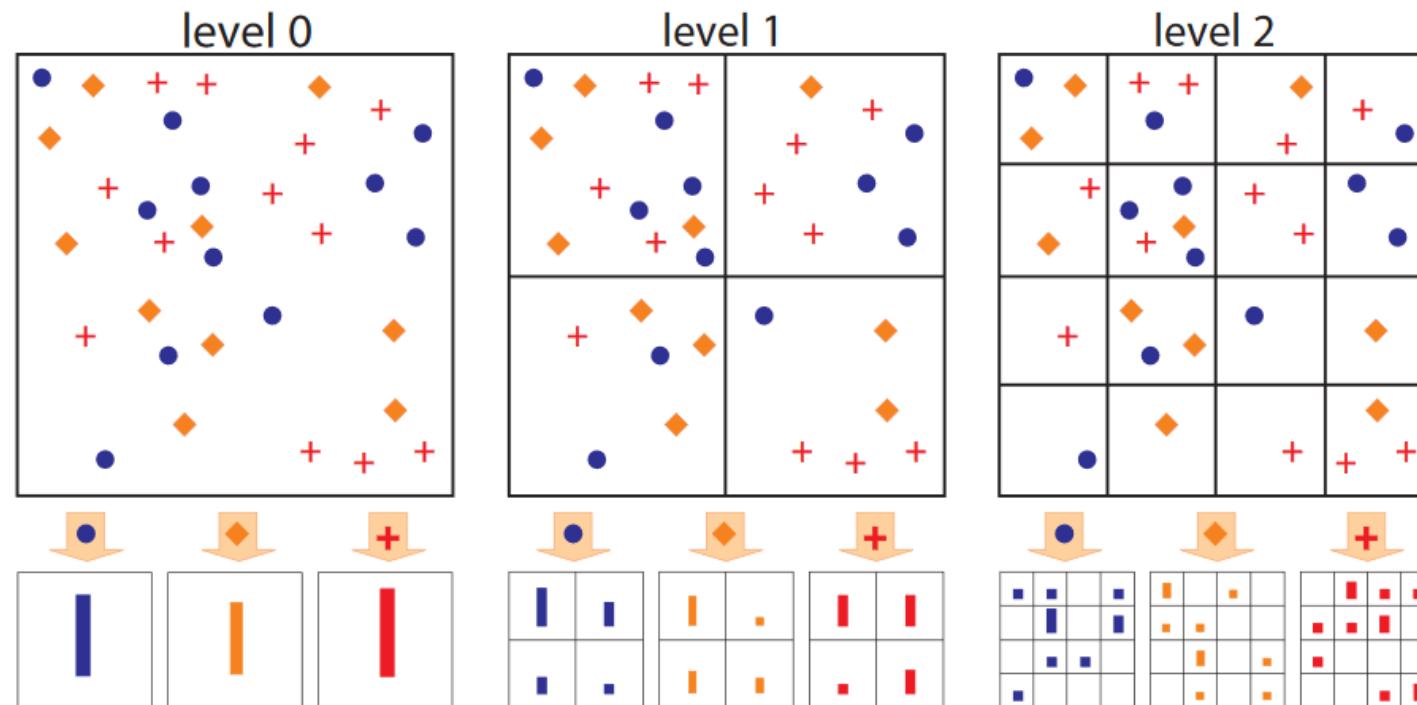
Object → Bag of 'words'



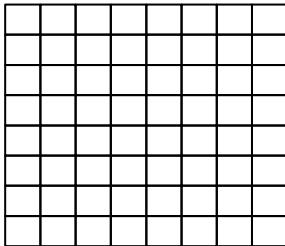


# SPM(Spatial Pyramid Matching) 개요

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

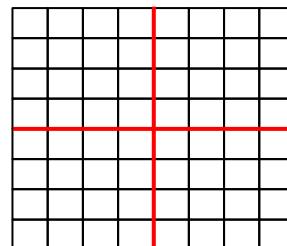


8 X 8 Feature Map

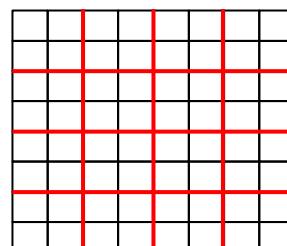


$$\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

$3 \times 1 = 3$



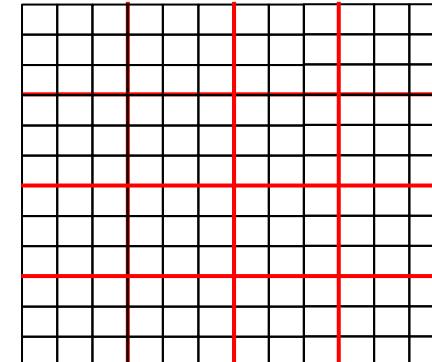
$$3 \times 4 = 12$$



$$3 \times 16 = 48$$

$3+12+48=63$ 개 원소의 vector 값으로 표현 가능

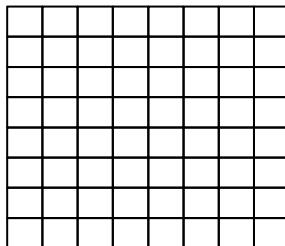
12 X 12 Feature Map



# Spatial Pyramid Pooling

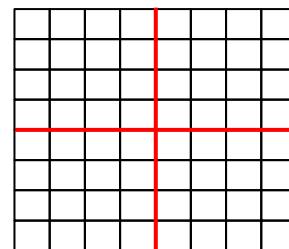
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

8 X 8 Feature Map



$$\begin{array}{|c|} \hline \square \\ \hline \end{array}$$

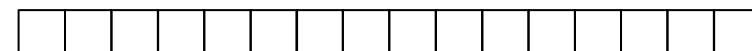
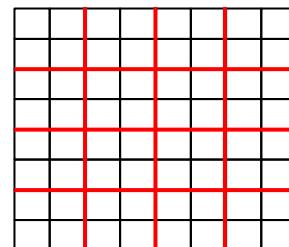
$1 \times 1 = 1$



$$4 \times 1 = 4$$

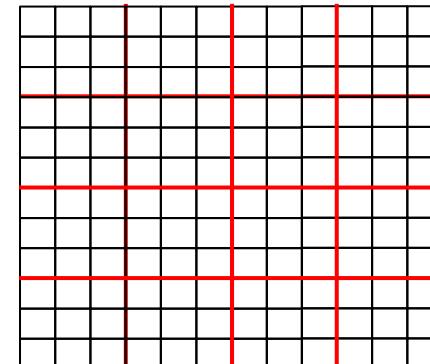
Max Pooling만 적용

$1+4+16=21$ 개 벡터 값으로 표현

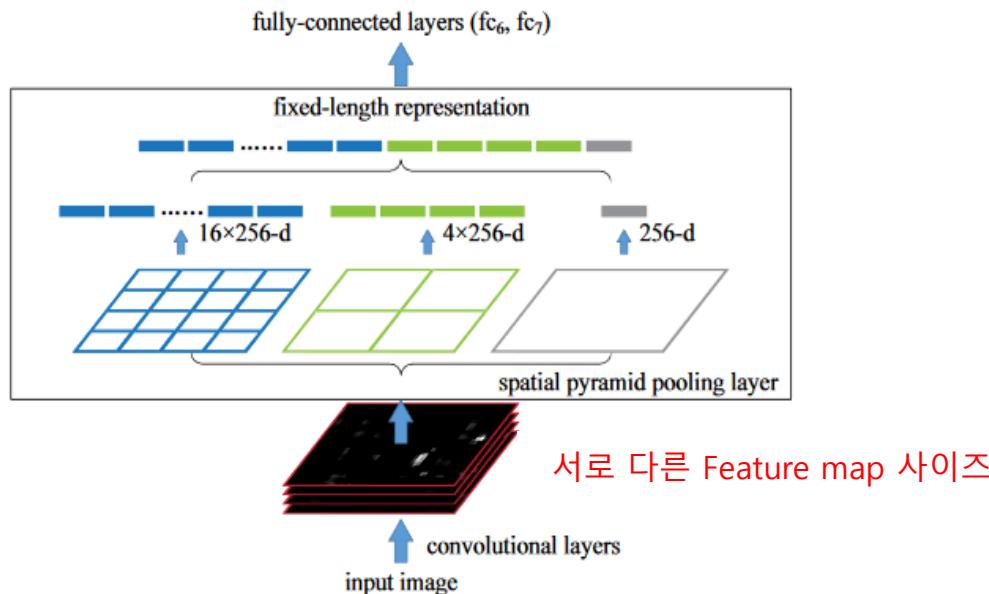


$$16 \times 1 = 16$$

12 X 12 Feature Map



### 이미지 Classification에서의 SPP-Net 구조

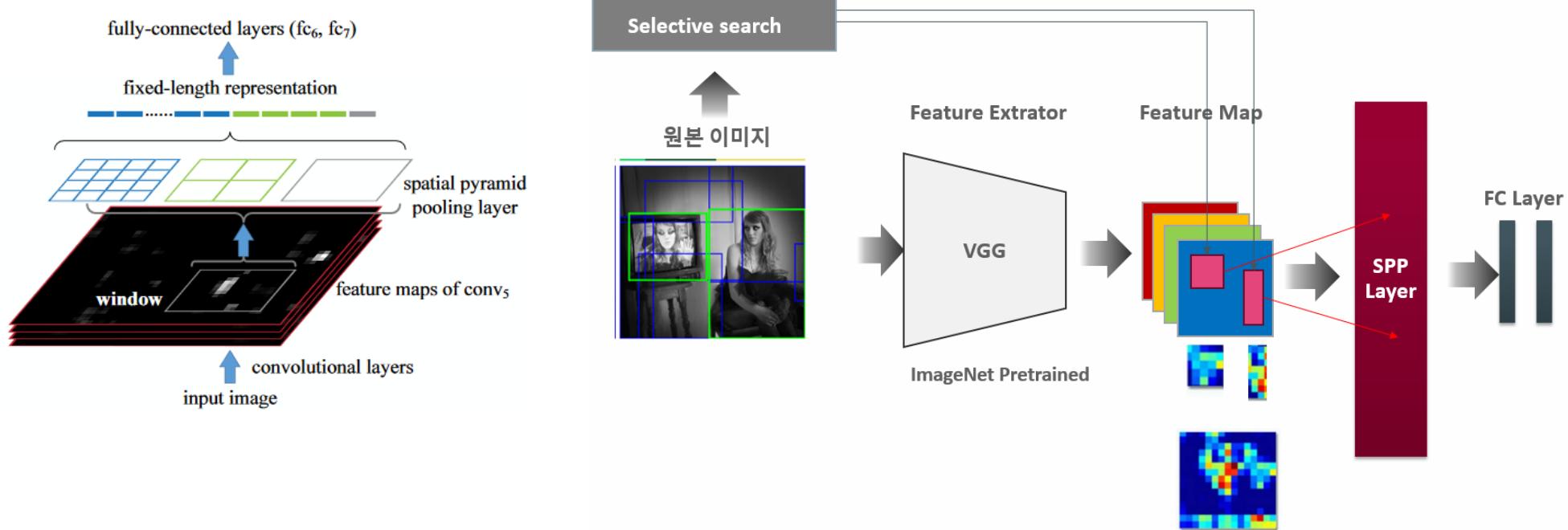


### 2014 ILSVRC Classification 결과

rank	team	top-5 test
1	GoogLeNet [32]	6.66
2	VGG [33]	7.32
3	<u>ours</u>	<u>8.06</u>
4	Howard	8.11
5	DeeperVision	9.50
6	NUS-BST	9.79
7	TTIC_ECP	10.22

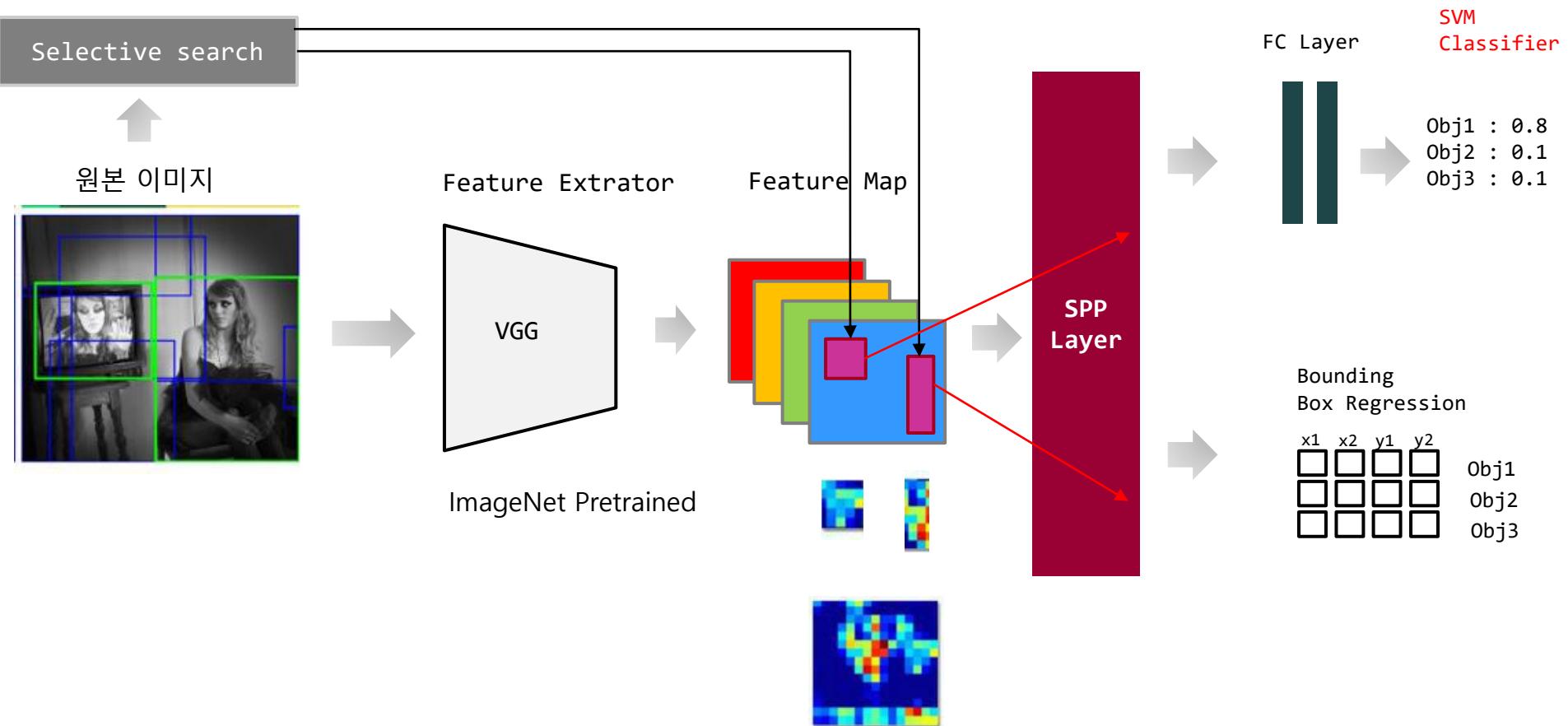
# SPP-Net-Object Detection

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

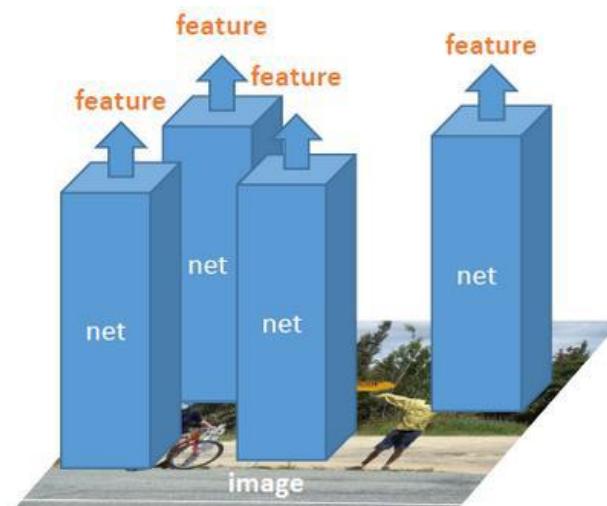


# SPP-Net을 RCNN에 적용

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

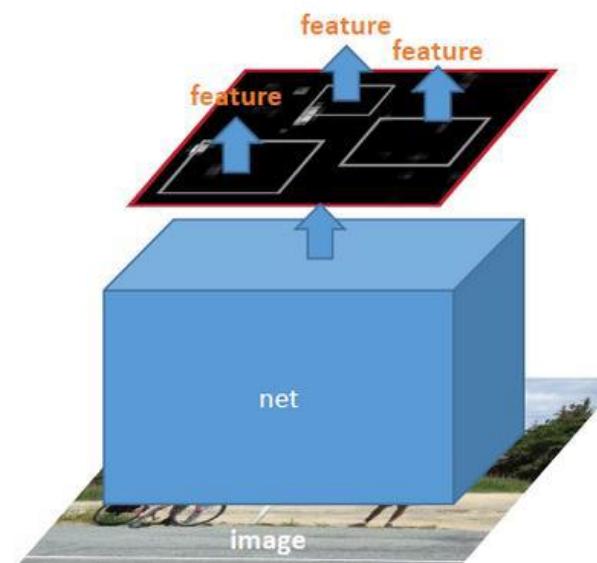


이미지 한 개에 2000번 CNN을 통과해야 함



R-CNN

이미지 한 개는 한번만 CNN



SPP-Net

# SPP-Net 성능 결과

1.3 R-CNN, Fast R-CNN, Faster R-CNN

## PASCAL VOC 2007 데이터 세트 적용 결과

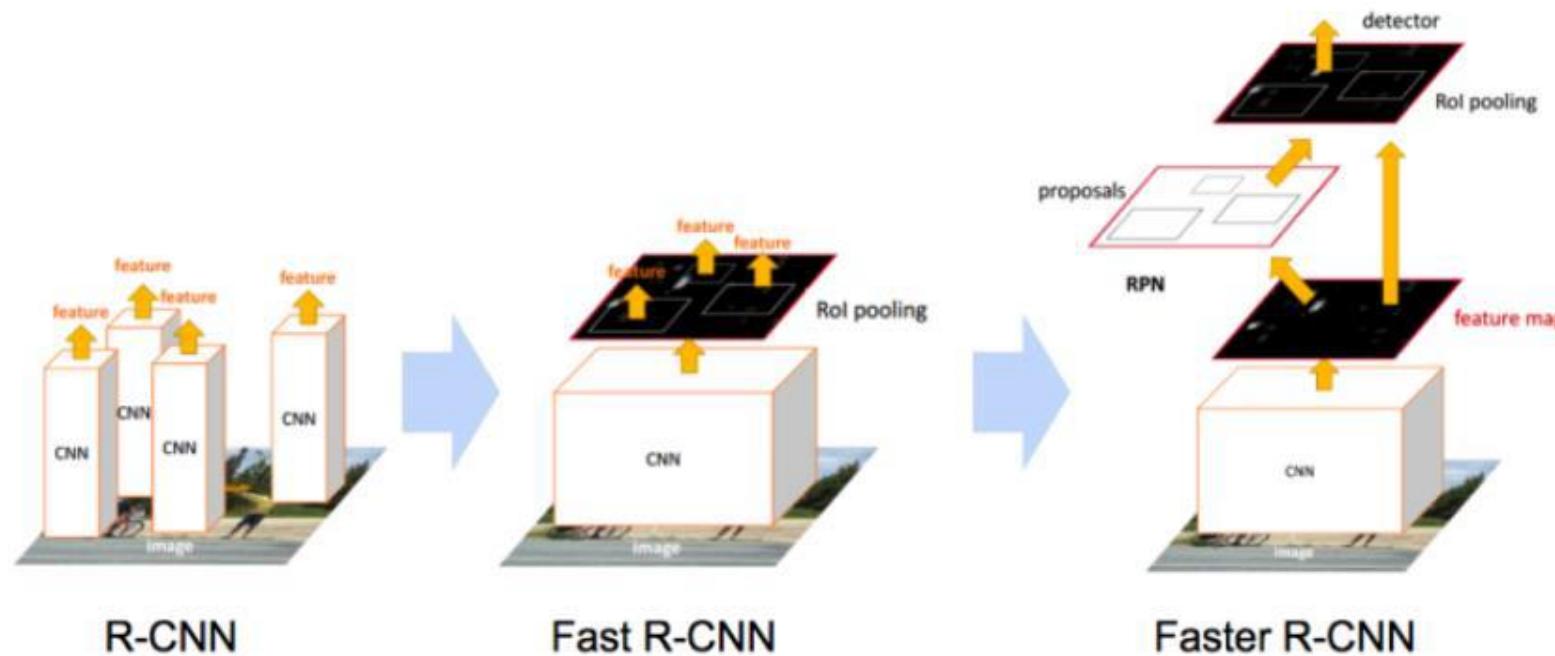
method	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
DPM [23]	33.7	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5
SS [20]	33.8	43.5	46.5	10.4	12.0	9.3	49.4	53.7	39.4	12.5	36.9	42.2	26.4	47.0	52.4	23.5	12.1	29.9	36.3	42.2	48.8
Regionlet [39]	41.7	54.2	52.0	20.3	24.0	20.1	55.5	68.7	42.6	19.2	44.2	49.1	26.6	57.0	54.5	43.4	16.4	36.6	37.7	59.4	52.3
DetNet [40]	30.5	29.2	35.2	19.4	16.7	3.7	53.2	50.2	27.2	10.2	34.8	30.2	28.2	46.6	41.7	26.2	10.3	32.8	26.8	39.8	47.0
RCNN ftfc7 (A5)	54.2	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7
RCNN ftfc7 (ZF5)	55.1	64.8	68.4	47.0	39.5	30.9	59.8	70.5	65.3	33.5	62.5	50.3	59.5	61.6	67.9	54.1	33.4	57.3	52.9	60.2	62.9
SPP ftfc7 (ZF5)	55.2	65.5	65.9	51.7	38.4	32.7	62.6	68.6	69.7	33.1	66.6	53.1	58.2	63.6	68.8	50.4	27.4	53.7	48.2	61.7	64.7
RCNN bb (A5)	58.5	68.1	72.8	56.8	43.0	36.8	66.3	74.2	67.6	34.4	63.5	54.5	61.2	69.1	68.6	58.7	33.4	62.9	51.1	62.5	64.8
RCNN bb (ZF5)	59.2	68.4	74.0	54.0	40.9	35.2	64.1	74.4	69.8	35.5	66.9	53.8	64.2	69.9	69.6	58.9	36.8	63.4	56.0	62.8	64.9
SPP bb (ZF5)	59.2	68.6	69.7	57.1	41.2	40.5	66.3	71.3	72.5	34.4	67.3	61.7	63.1	71.0	69.8	57.6	29.7	59.0	50.2	65.2	68.0

## ILSVRC 2014 Object Detection 결과

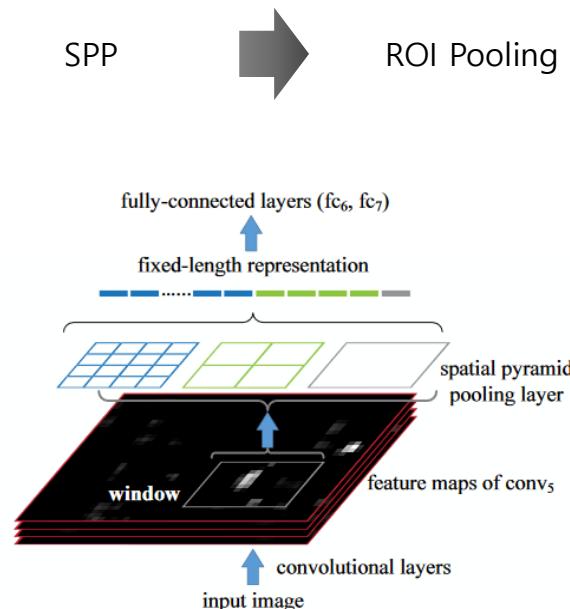
rank	team	mAp
1	NUS	37.21
2	<u>ours</u>	<u>35.11</u>
3	UvA	32.02
-	(our single-model)	(31.84)
4	Southeast-CASIA	30.47
5	1-HKUST	28.86
6	CASIA_Cripac_2	28.61

## PASCAL VOC 2007 데이터 세트 기반 수행 시간

	SPP-net 1-scale	SPP-net 5-scale	RCNN
mAP	58.0	<b>59.2</b>	58.5
GPU time/img	<b>0.14s</b>	0.38s	9s
speed-up	<b>64x</b>	<b>24x</b>	-



SPP Layer를 ROI Pooling Layer로



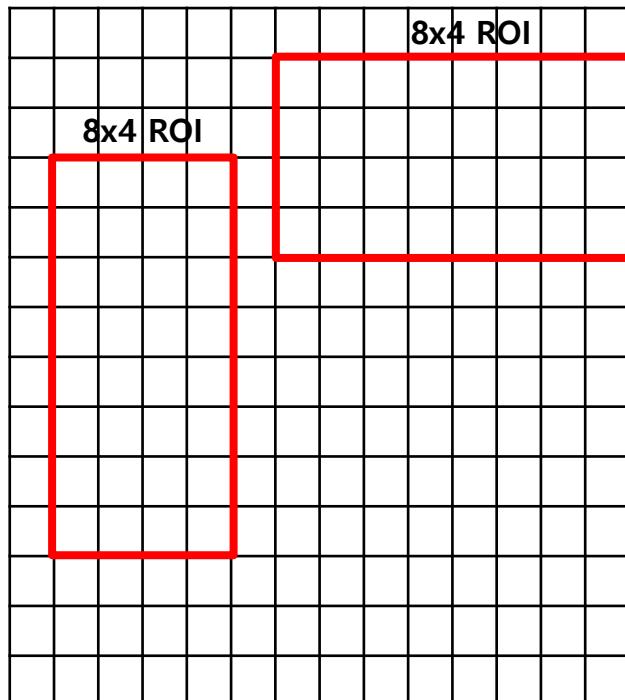
End-to-End Network Learning  
(ROI Proposal은 제외)

- SVM을 Softmax로 변환
- Multi-task loss 함수로 Classification과 Regression을 함께 최적화

# ROI Pooling

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

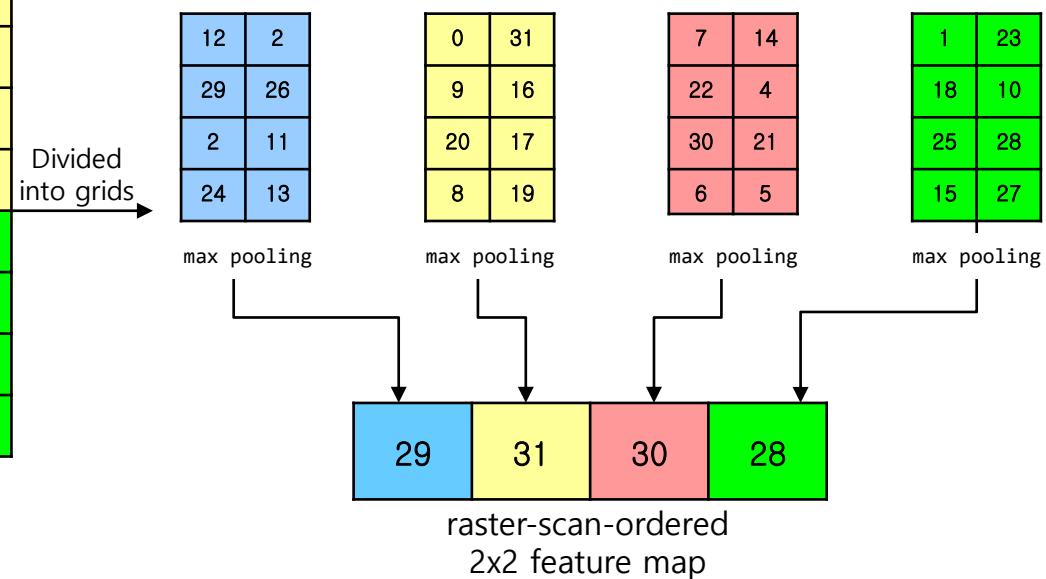
14x14x Feature Map



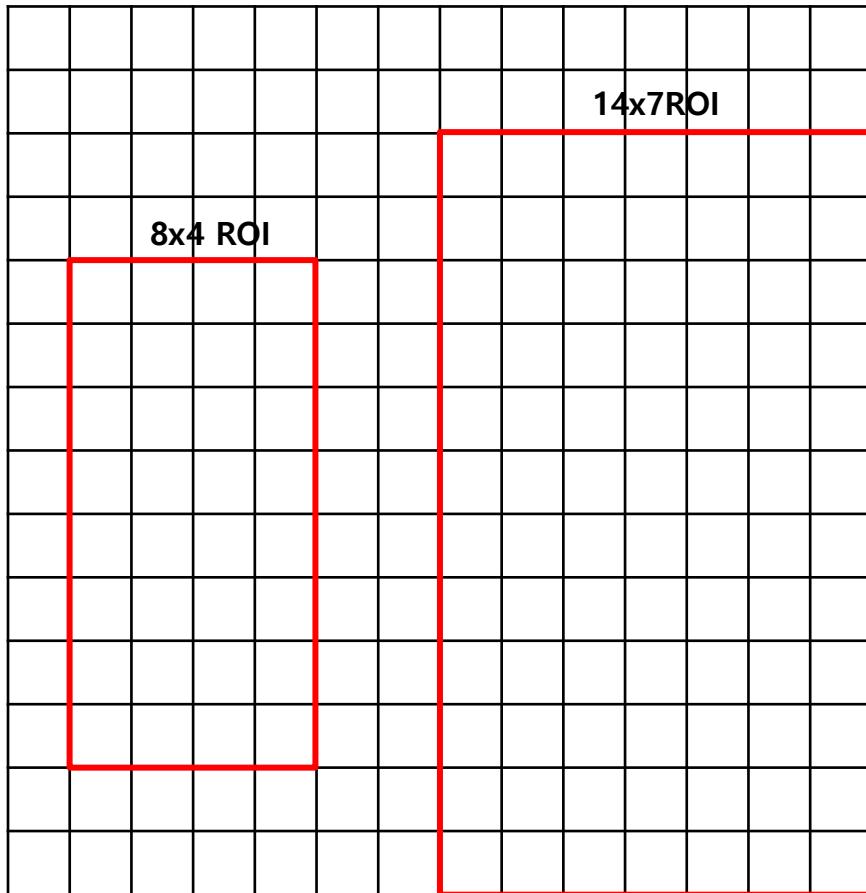
12	2	0	31
29	26	9	16
2	11	20	17
24	13	8	19
7	14	1	23
22	4	18	10
30	21	25	28
6	5	15	27

a channel of  
ROI feature map

- Feature Map 상의 임의의 ROI를 고정 크기의 Pooling 영역으로 매핑
- 매핑 시 일반적으로 Max Pooling 적용

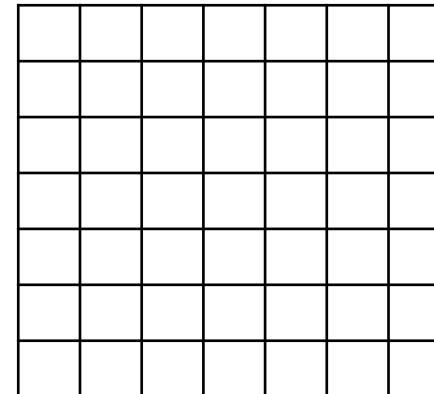


14x14 Feature Map



- Feature Map 상의 임의의 ROI를 고정 크기의 Pooling 영역으로 매핑
- 매핑 시 일반적으로 Max Pooling 적용

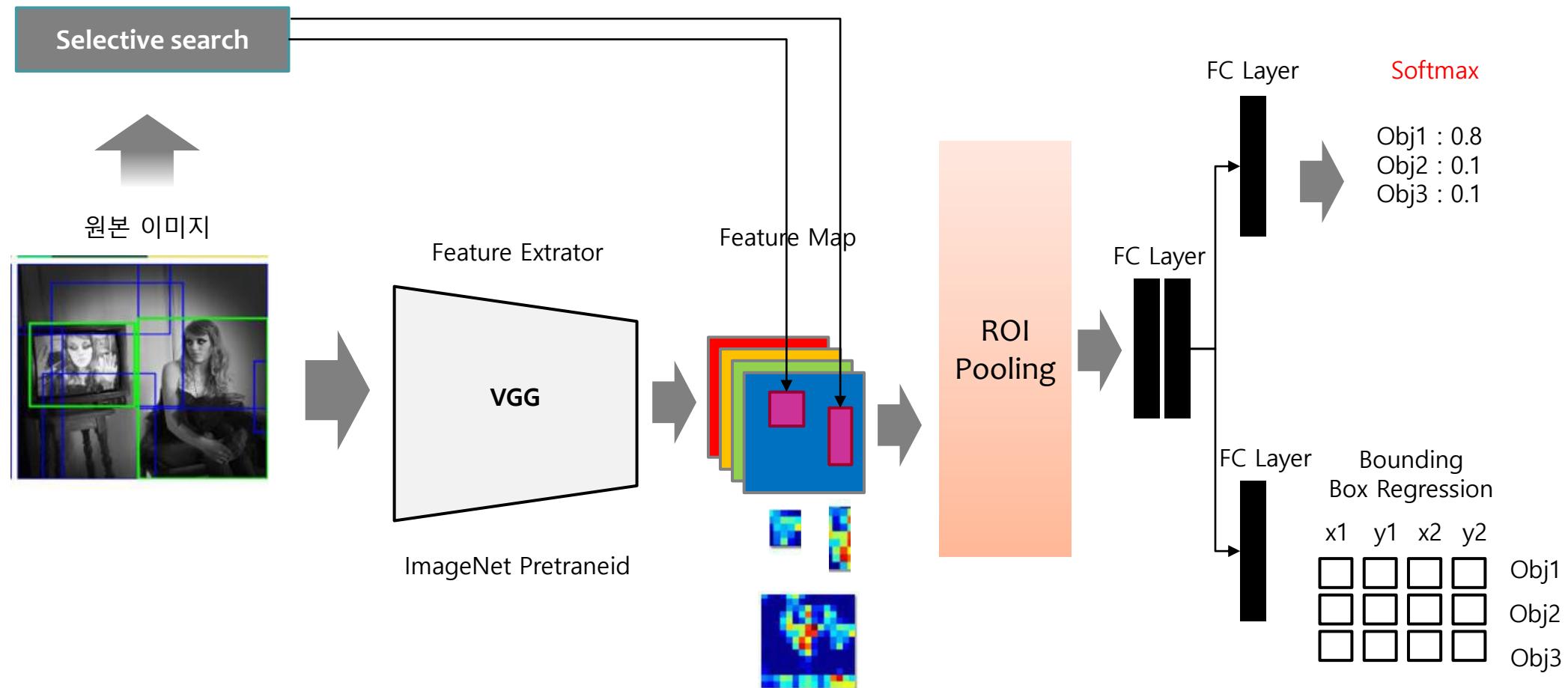
7x7 Pool



Pool 크기를 보통 7x7로 설정

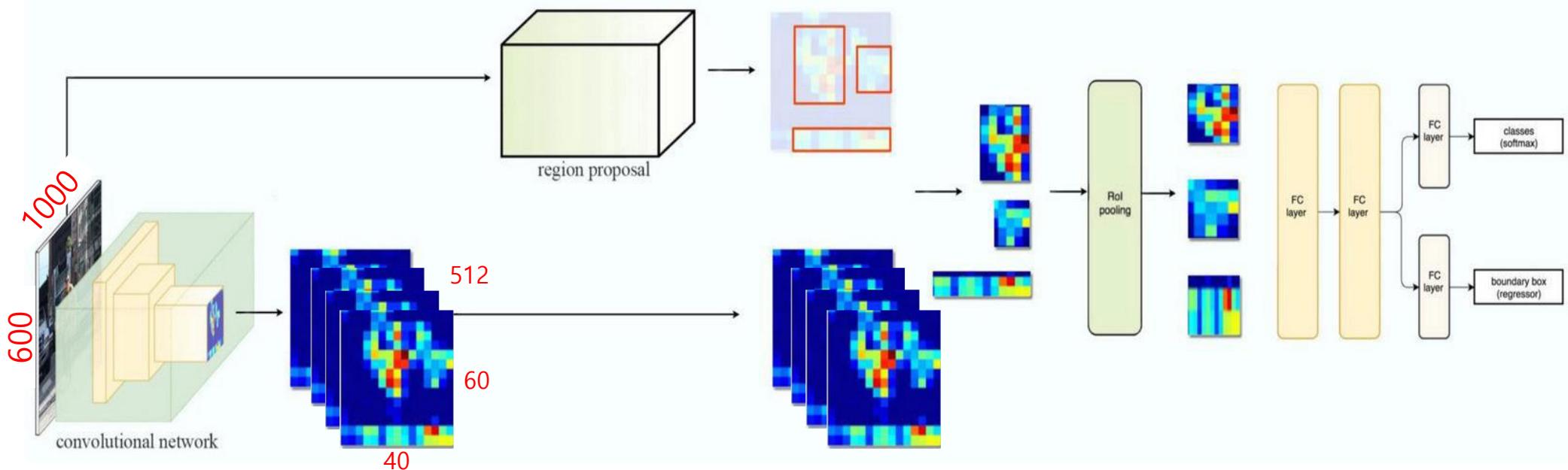
# ROI Pooling을 적용한 Fast RCNN 구조

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



# Fast RCCN 구조

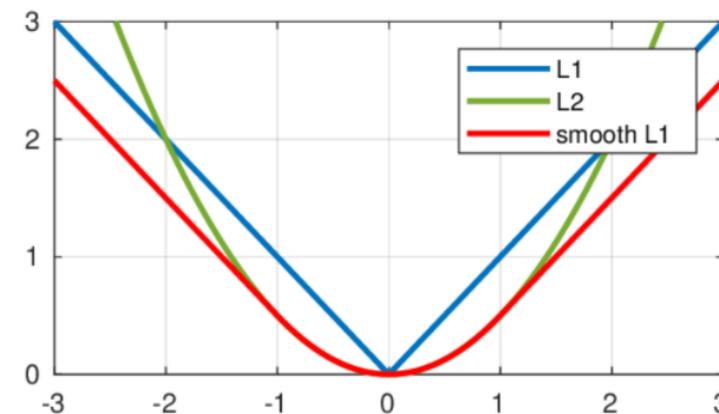
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



### Classification과 Regression를 함께 반영한 Loss 함수

Multi-task Loss 함수:  $L(p, u, t^u, v) = \underbrace{L_{cls}(p, u)}_{\text{Classification Loss}} + \lambda[u \geq 1] \underbrace{L_{loc}(t^u, v)}_{\text{Regression Loss}},$

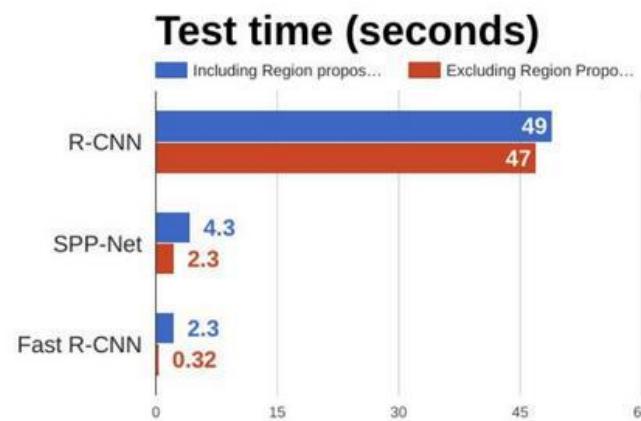
Regression Loss 함수:  $L_{loc}(t^u, v) = \sum_{i \in \{x, y, w, h\}} smooth_{L_1}(t_i^u - v_i), \quad smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$



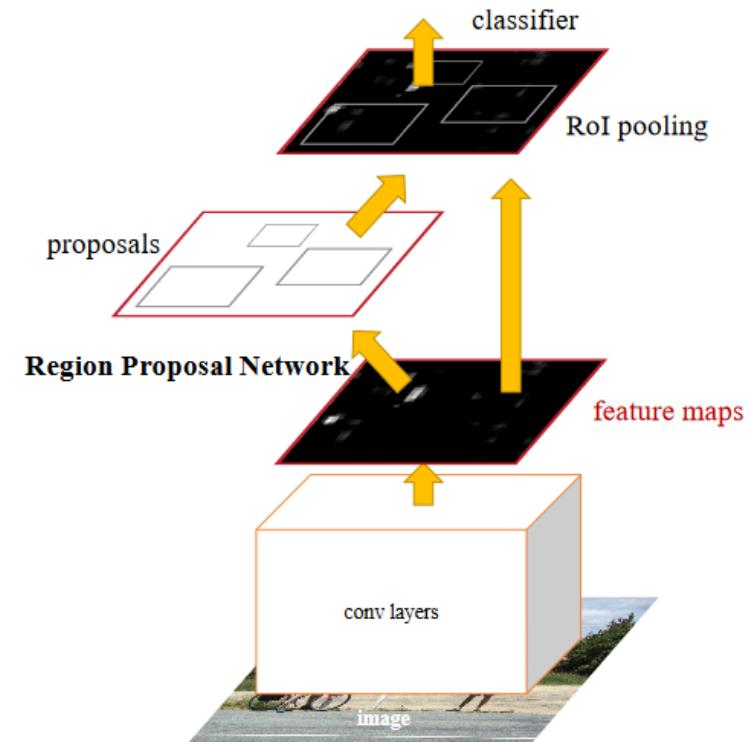
PASCAL VOC 2012 데이터 세트 적용 결과

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2	68.4

수행 시간 비교

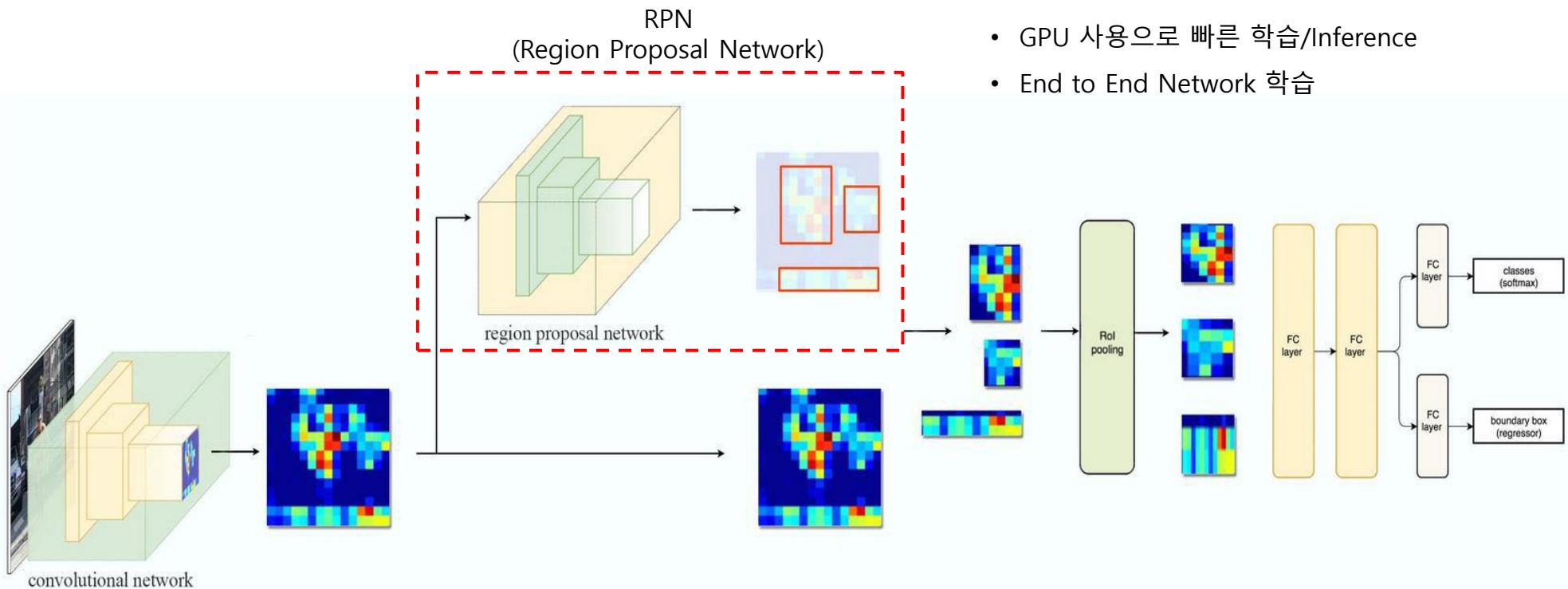


**Faster RCNN = RPN+Fast RCNN**



Faster RCNN

### Selective Search를 Neural Network 구조로 변경

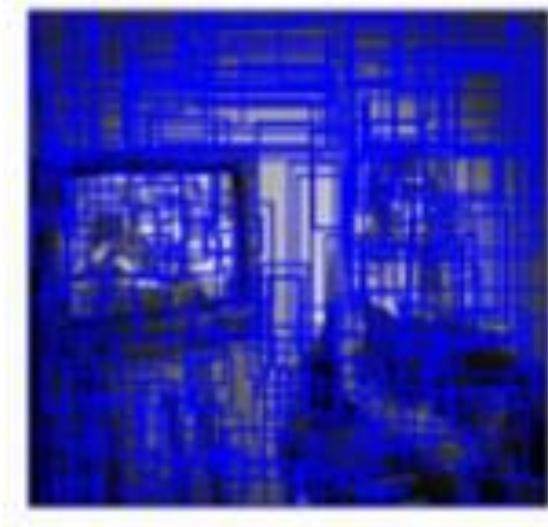


### Seletive Search를 대체하기 위한 Region Proposal Network 구현 이슈

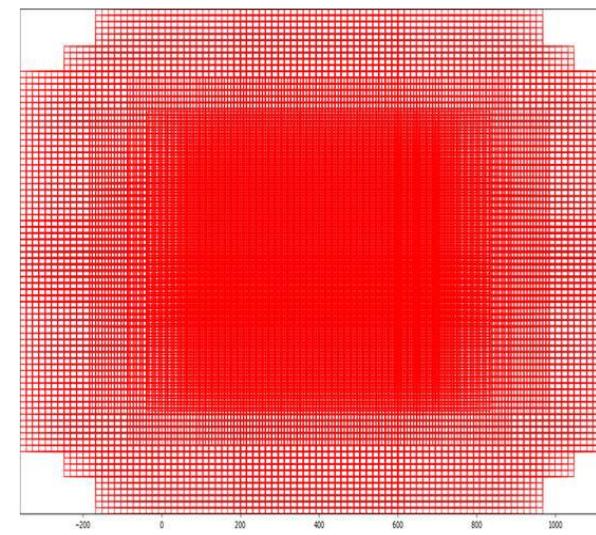
- 데이터로 주어질 피처는 pixel 값, Target은 Ground Truth Bounding Box인데 이를 이용해 어떻게 Selective 수준의 Region Proposal을 할 수 있을 것인가?

(reference) Anchor Box  
Object가 있는지 없는지의 후보 Box

Selecttive Search ROI

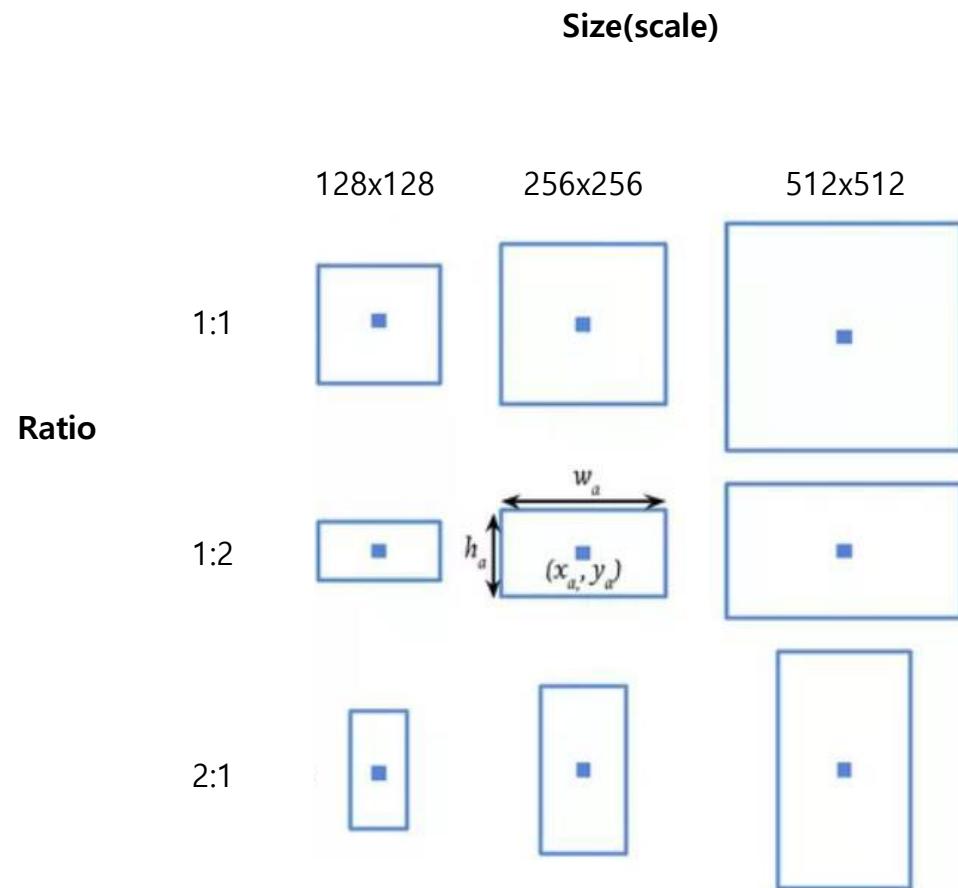


Anchor Box ROI

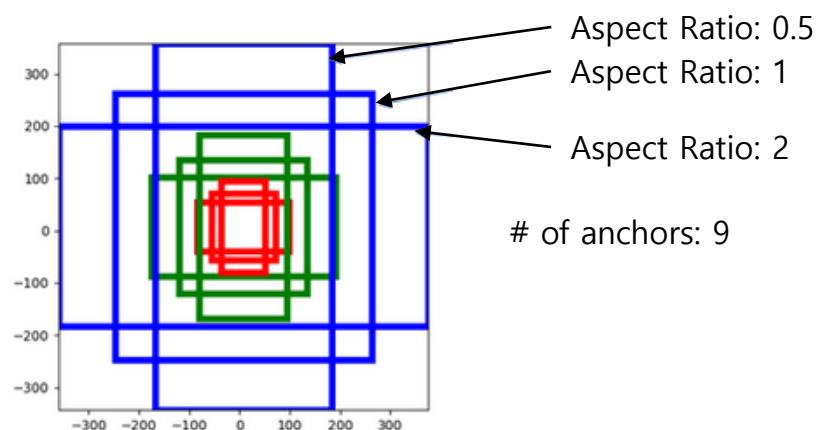


# Anchor Box 구성

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



총 9개의 Anchor box, 3개의 서로 다른 크기, 3개의 서로 다른 ratio로 구성

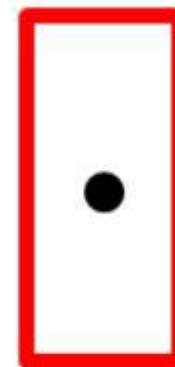


# Anchor Box 특징

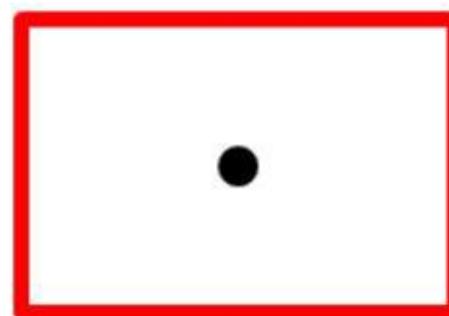
1.3 R-CNN, Fast R-CNN, Faster R-CNN



Anchor box 1:

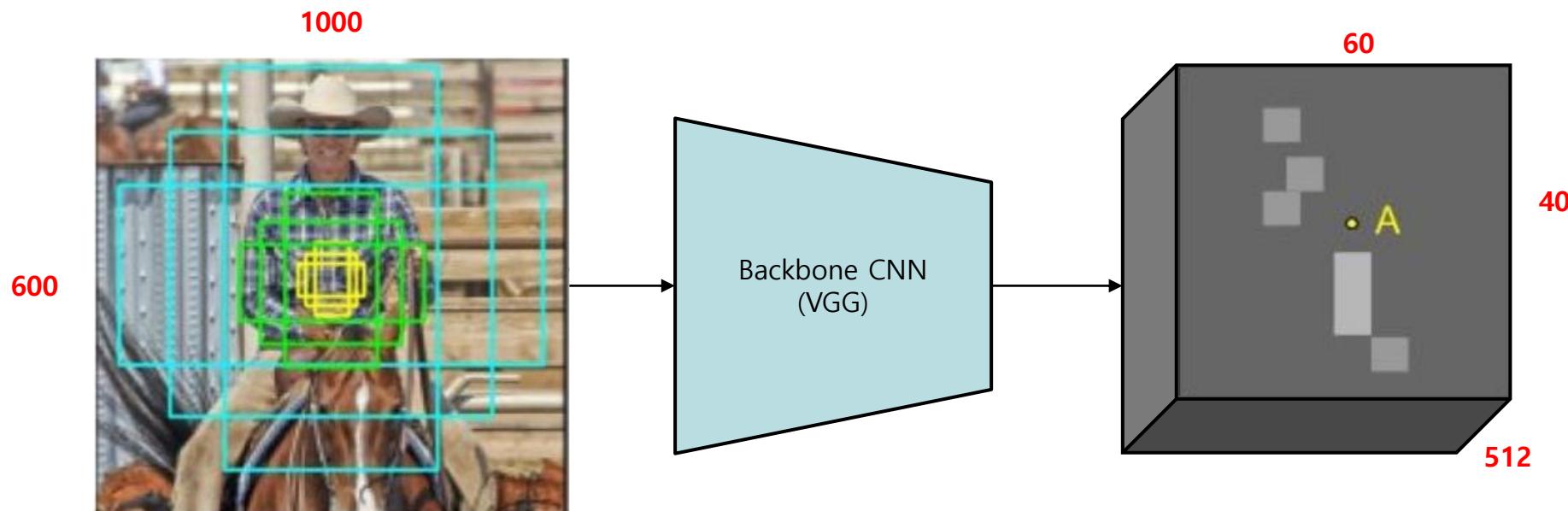


Anchor box 2:



# 이미지와 Feature Map에서 Anchor Box 매핑

1.3 R-CNN, Fast R-CNN, Faster R-CNN



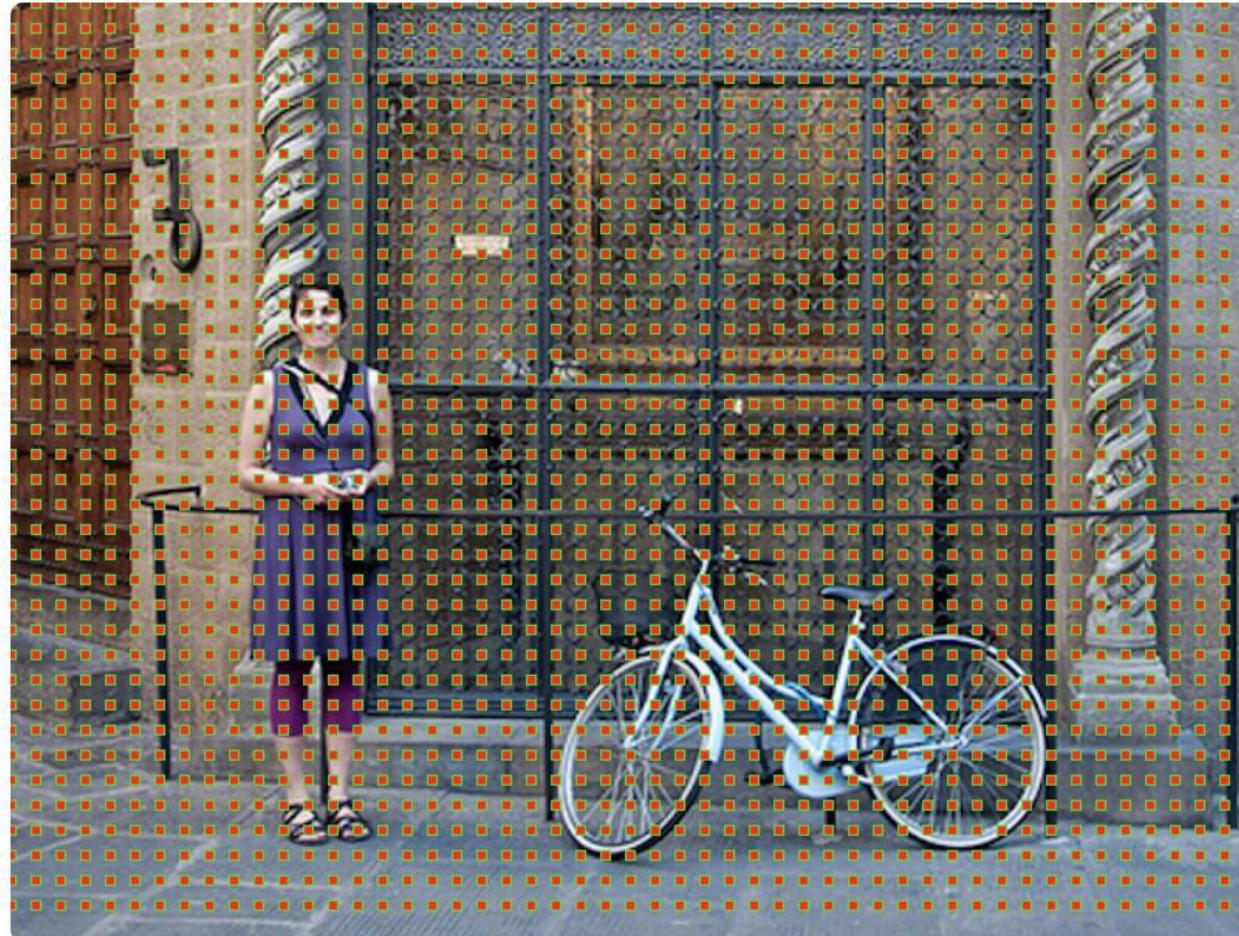
# 이미지와 Feature Map에서 Anchor Box 매핑

1.3 R-CNN, Fast R-CNN, Faster R-CNN

원본 이미지가 1/6 크기의 Feature Map으로 Down Sampling 될 때

Image Height 600  
(Feature Map Height 38)

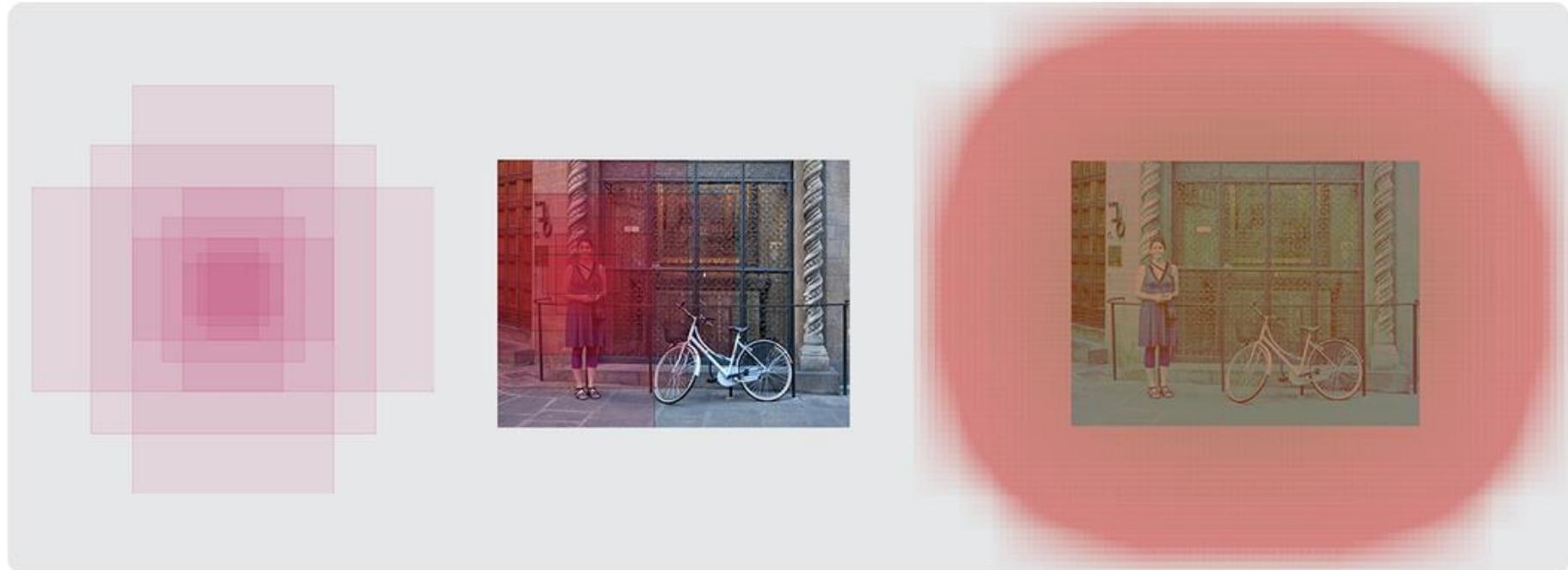
Image Width 800(Feature Map Width 50)



<https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>

# 이미지와 Feature Map 에서 Anchor Box 매핑

1.3 R-CNN, Fast R-CNN, Faster R-CNN



*Left: Anchors, Center: Anchor for a single point, Right: All anchors*

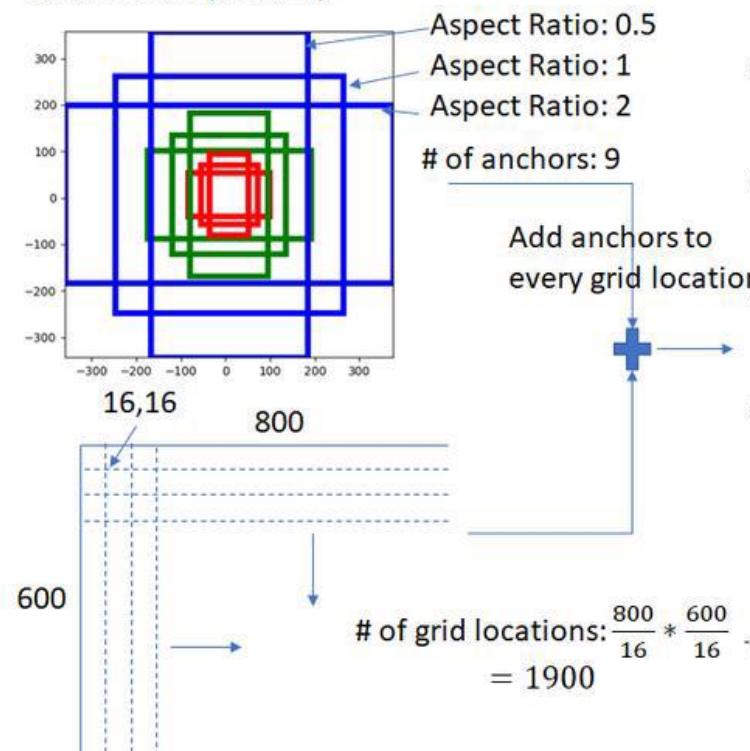
# 이미지와 Feature Map에서 Anchor Box 매핑

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

### Generate Anchors

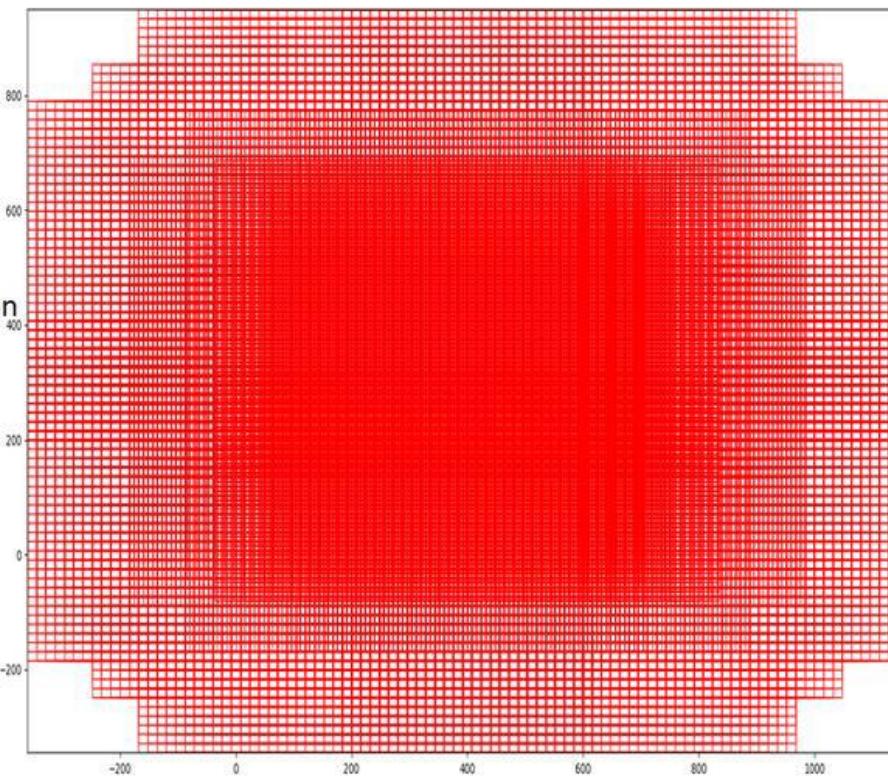
Given:

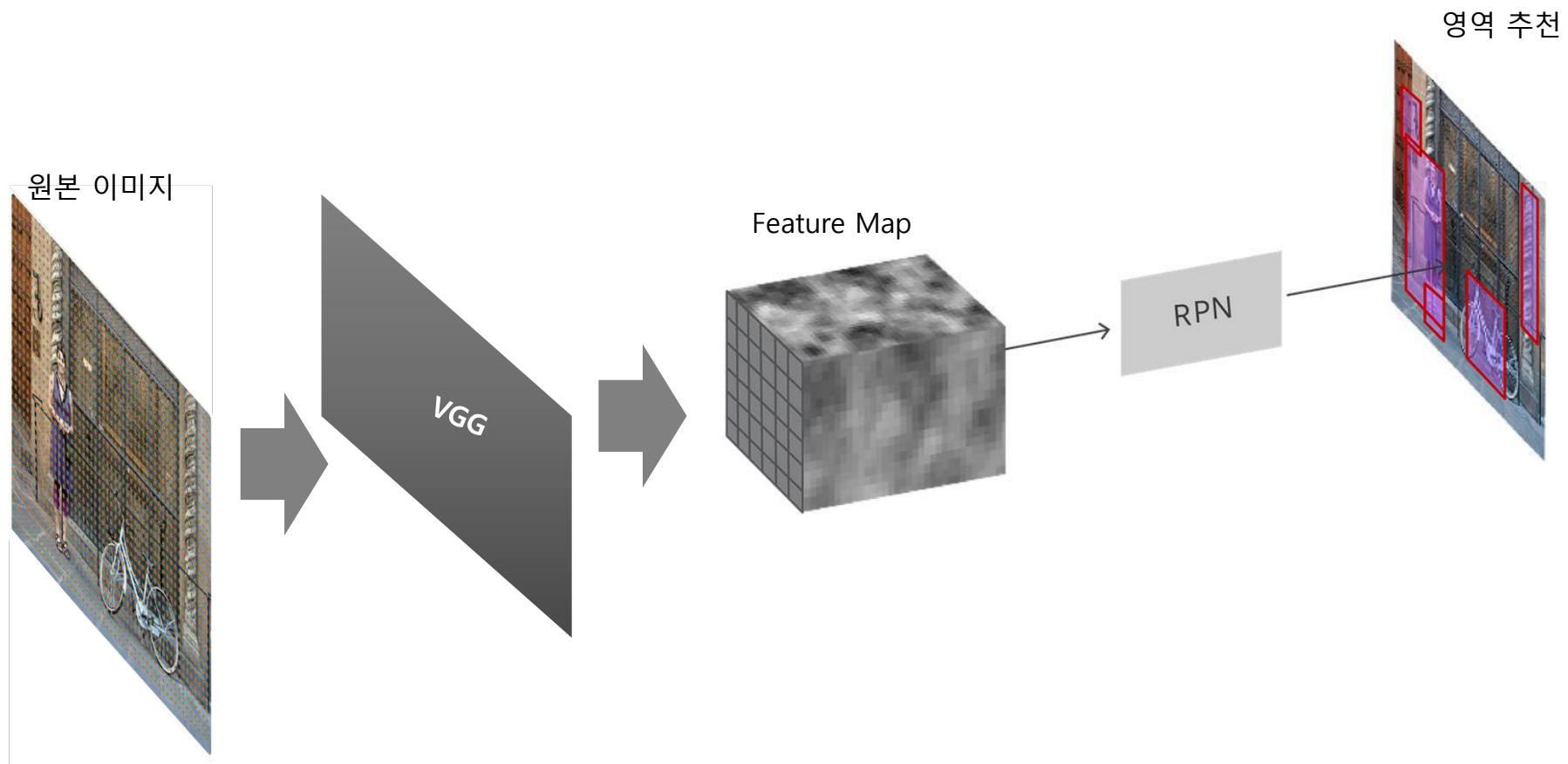
- Set of aspect ratios (0.5, 1, 2)
- Stride length (downscaling performed by resnet head: 16)
- Anchor Scales (8, 16, 32)

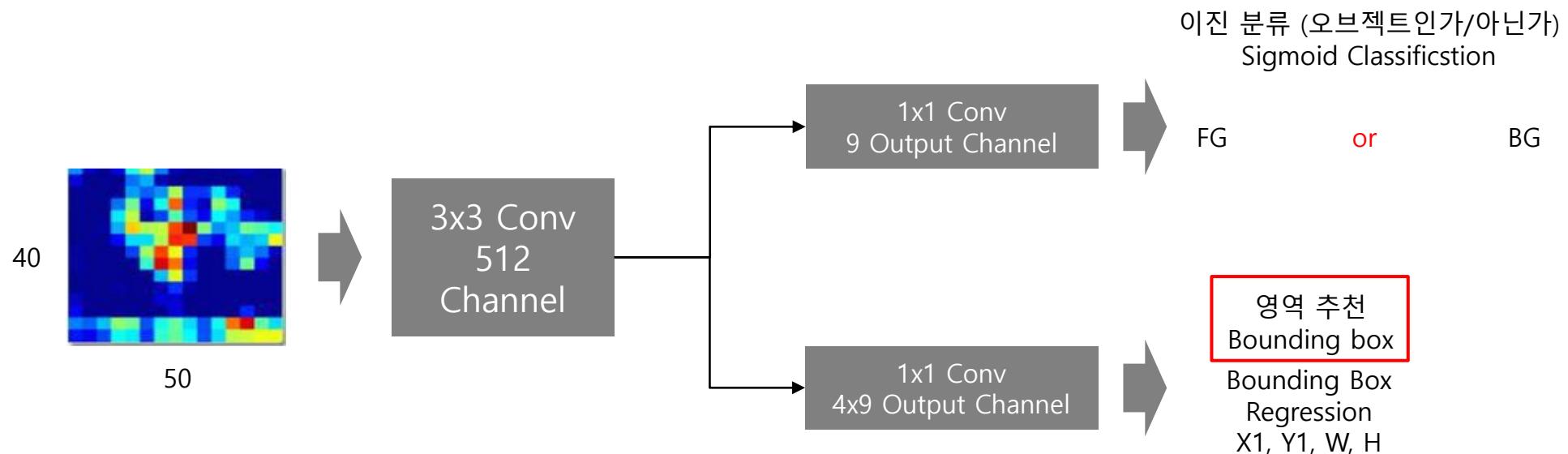


Create uniformly spaced grid with  
spacing = stride length

Total number of anchors:  $1900 * 9 = 17100$   
Some boxes lie outside the image  
boundary







```

def rpn(base_layers, num_anchors):

    x = Convolution2D(512, (3, 3), padding='same', activation='relu', kernel_initializer='normal', name='rpn_conv1')(base_layers)

    x_class = Convolution2D(num_anchors, (1, 1), activation='sigmoid', kernel_initializer='uniform', name='rpn_out_class')(x)
    x_regr = Convolution2D(num_anchors * 4, (1, 1), activation='linear', kernel_initializer='zero', name='rpn_out_regress')(x)

    return [x_class, x_regr, base_layers]

```

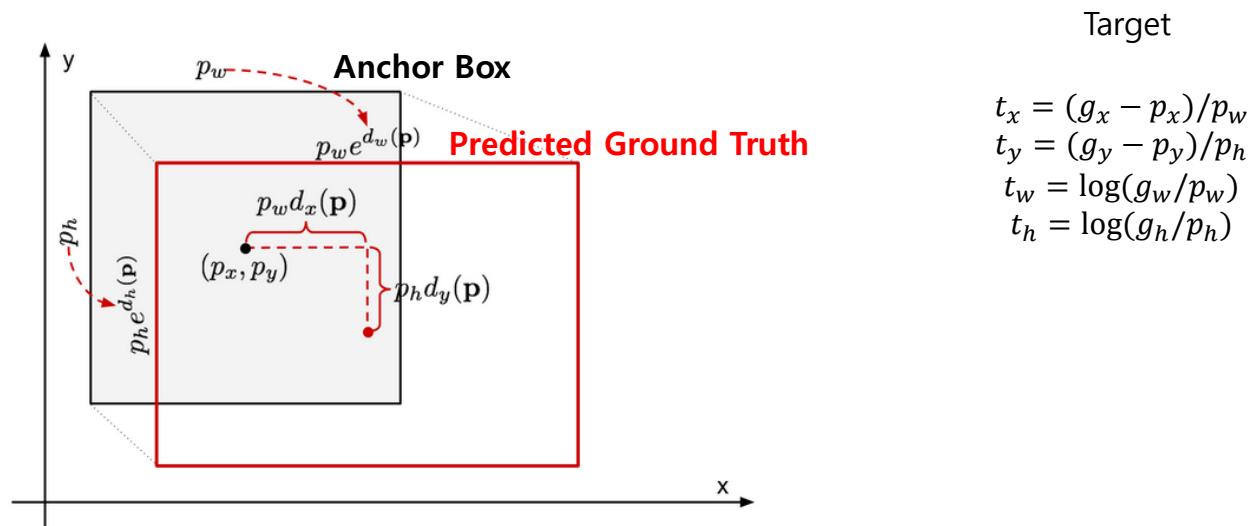
# RPN Bounding Box Regression

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

RPN Bounding Box Regression은 Anchor box를 Reference로 이용하여 Ground truth와 예측 Bbox의 중심좌표x, y 그리고 w, h의 차이가 Anchor box와 Ground Truth 간의 중심좌표x,y, w, h의 차이와 최대한 동일하게 예측될 수 있어야 함.

수정 예측값

$$\begin{aligned}\hat{g}_x &= p_w d_x(p) = p_y \\ \hat{g}_y &= p_h d_y(p) = p_y \\ \hat{g}_w &= p_w \exp(d_w(p)) \\ \hat{g}_h &= p_h \exp(d_h(p))\end{aligned}$$



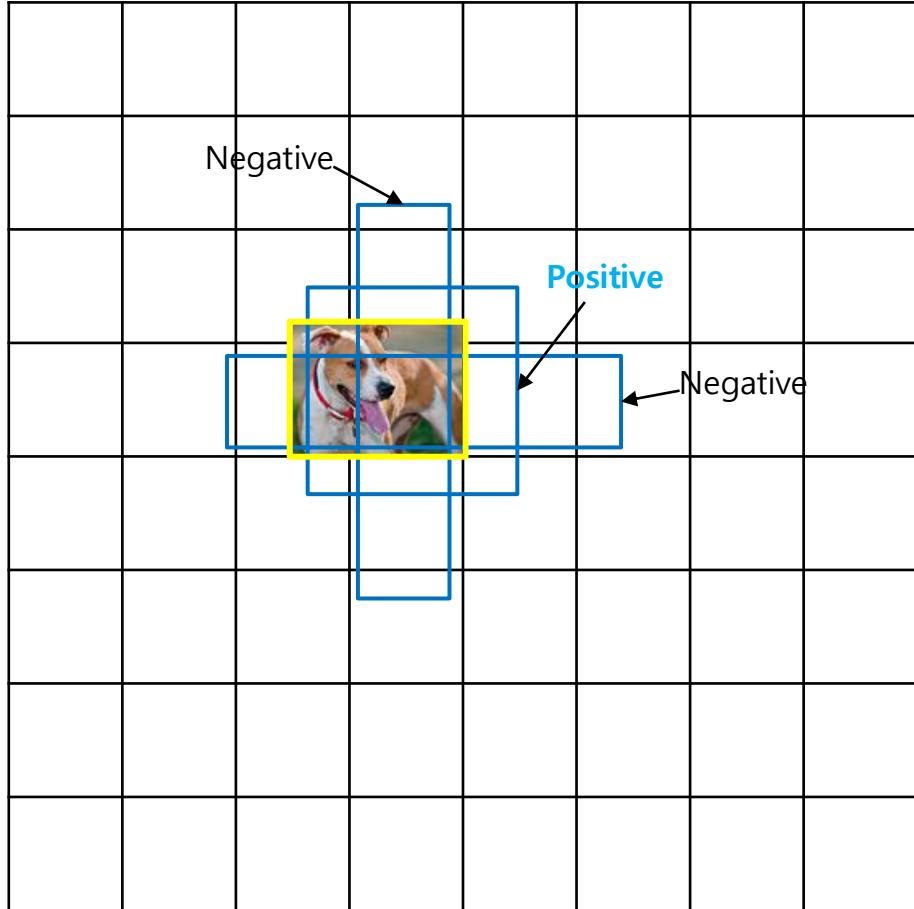
Faster RCNN RPN Box  
Regression 논문 표기

$$\begin{aligned}t_x &= (x - x_a)/w_a, t_y = (y - y_a)/h_a, \\ t_w &= \log(w/w_a), t_h = \log(h/h_a), \\ t_x^* &= (x^* - x_a)/w_a, t_y^* = (y^* - y_a)/h_a, \\ t_w^* &= \log(w^*/w_a), t_h^* = \log(h^*/h_a),\end{aligned}$$

$x_a$ : anchor x 좌표  
 $x^*$ : Ground truth x 좌표  
 $x$ : 예측 x 좌표

# Positive Anchor Box, Negative Anchor Box

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

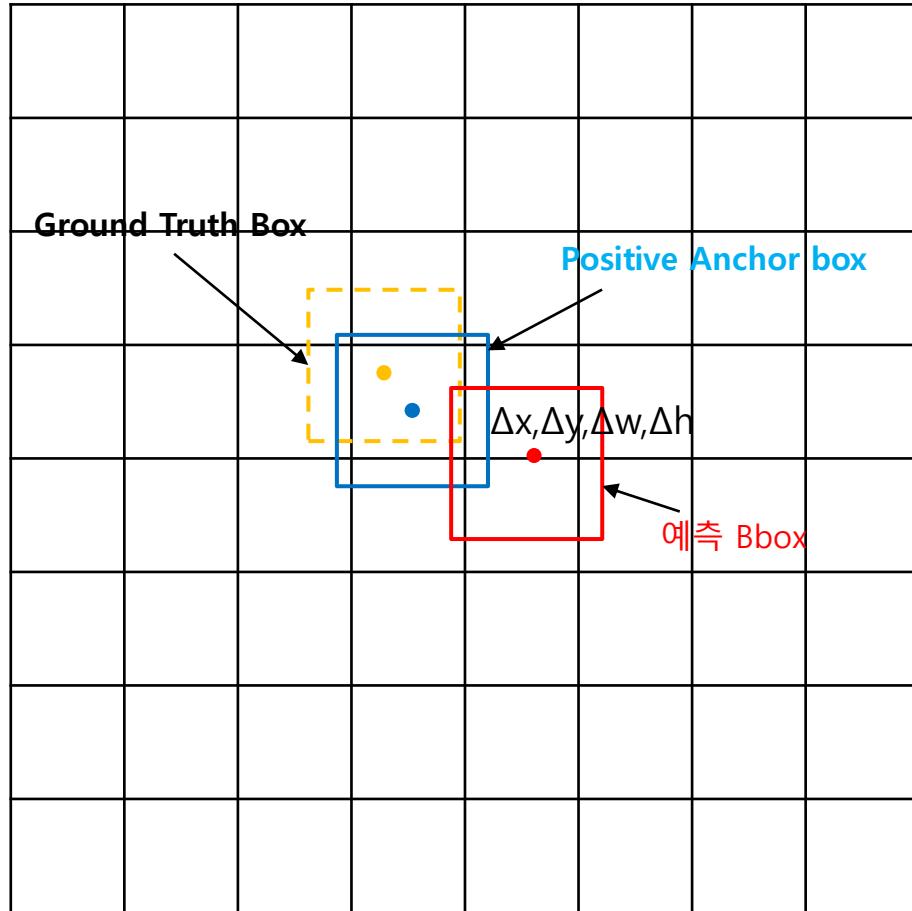


Ground Truth BB 겹치는 IOU 값에 따라 Anchor Box 를 Positive Anchor Box, Negative Anchor box 로 분류

- IOU 가 가장 높은 Anchor 는 Positive
- IOU 가 0.7 이상이면 Positive
- IOU 가 0.3 보다 낮으면 Negative

# Anchor box를 Reference로 한 Bounding Box Regression

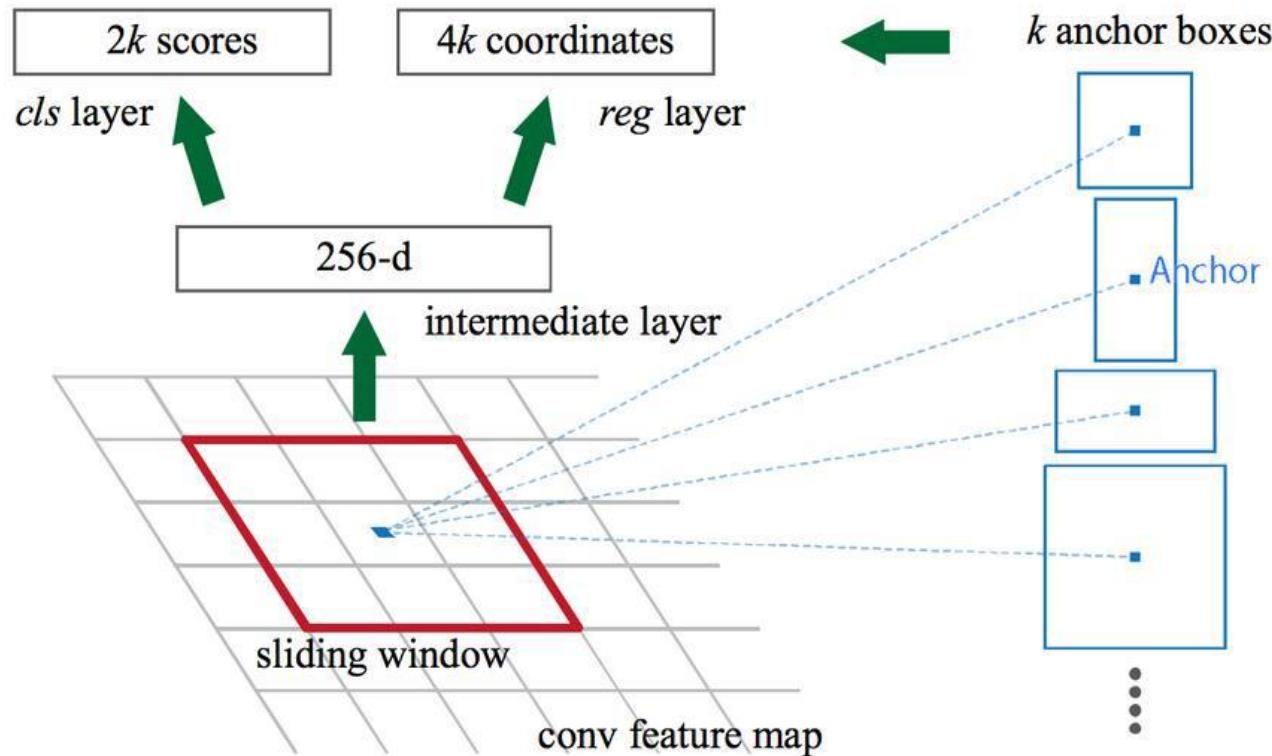
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



예측 bounding box 와 Positive anchor box 와의 좌표 차이는 Ground Truth 와 Positive anchor box 와의 좌표 차이와 최대한 동일하게 될 수 있도록 regression 학습

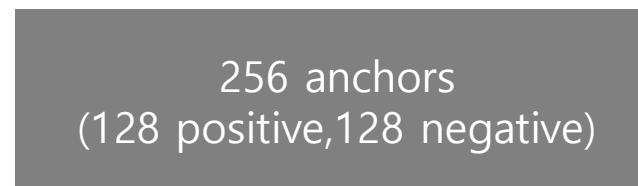
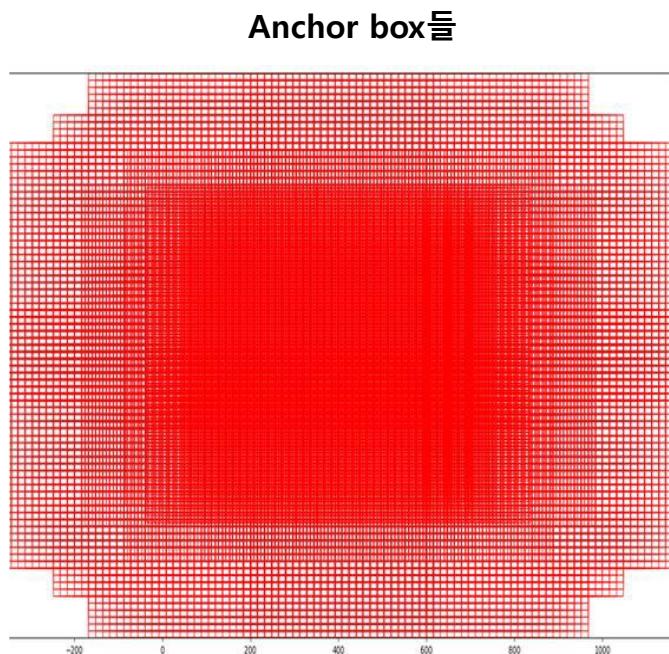
# Anchor box에 따른 RPN Output

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



$$\mathcal{L}(\{p_i\}, \{t_i\}) = 1/N_{cls} \sum_i \mathcal{L}_{cls}(p_i, p_i^*) + \lambda/N_{box} \sum_i p_i \cdot L_1^{smooth}(t_i - t_i^*)$$

Symbol	Explanation
$p_i$	Anchor $i$ 가 오브젝트일 예측 확률
$p_i^*$	Anchor $i$ 의 Ground truth Object 여부 (Positive 1, Negative 0)
$t_i$	Anchor $i$ 와 예측 좌표 차이 ( $x, y, w, h$ )
$t_i^*$	Anchor $i$ 와 Ground Truth 좌표 차이( $x, y, w, h$ )
$N_{cls}$	미니 배치에 따른 정규화 값(256)
$N_{box}$	박스 개수 정규화 값(최대 2400)
$\lambda$	밸런싱 값: 10

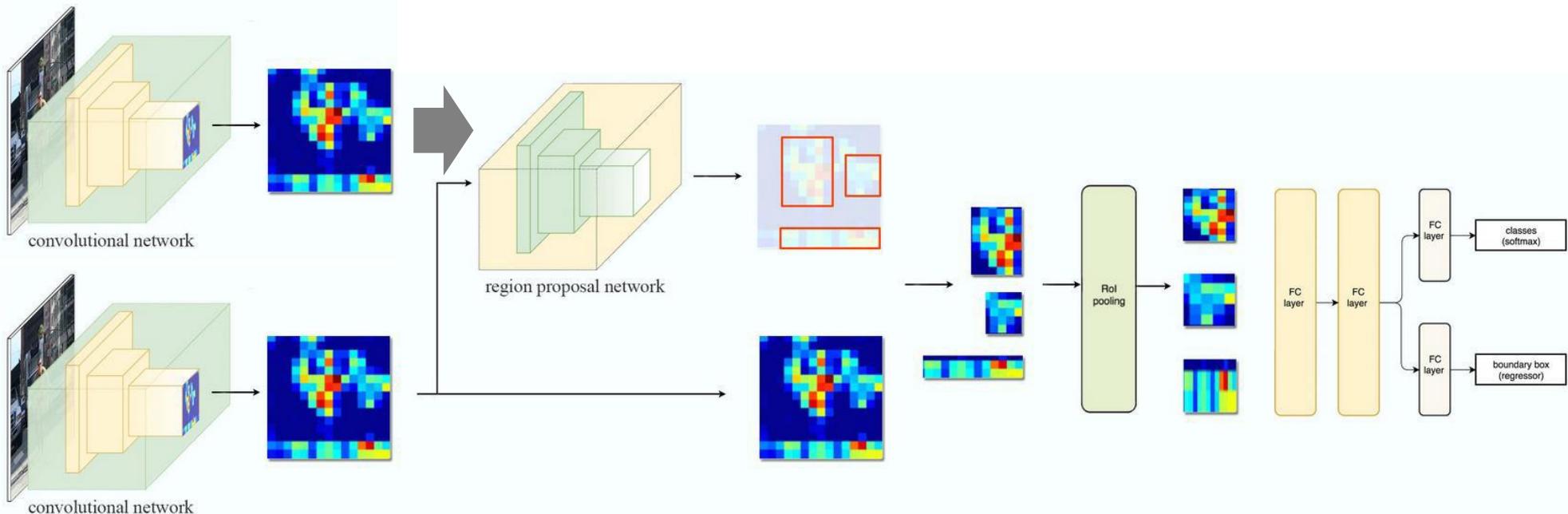


학습



1. RPN을 먼저 학습
2. Fast RCNN Classification/Regression 학습
3. RPN을 Fine Tuning
4. Fast RCNN Fine Tuning

Alternating training



# Faster RCNN Detection 성능 비교

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

### PASCAL VOC 데이터 세트에서 Detection 성능 비교

method	# box	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SS	2000	12	65.7	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7
SS	2000	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
RPN	300	12	67.0	82.3	76.4	71.0	48.4	45.2	72.1	72.3	87.3	42.2	73.7	50.0	86.8	78.7	78.4	77.4	34.5	70.1	57.1	77.1	58.9
RPN	300	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
RPN	300	COCO+07++12	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2

### Coco 데이터 세트에서 Detection 성능 비교

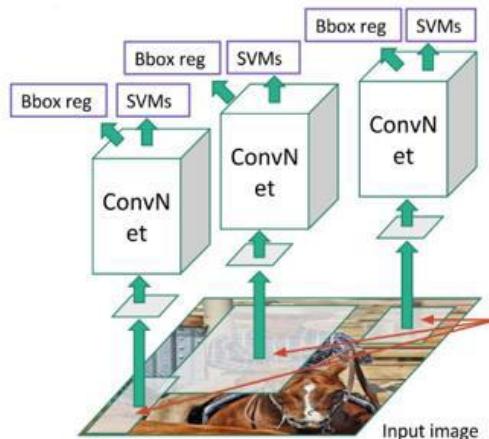
method	proposals	training data	COCO val		COCO test-dev	
			mAP@.5	mAP@[.5, .95]	mAP@.5	mAP@[.5, .95]
Fast R-CNN [2]	SS, 2000	COCO train	-	-	35.9	19.7
Fast R-CNN [impl. in this paper]	SS, 2000	COCO train	38.6	18.9	39.3	19.3
Faster R-CNN	RPN, 300	COCO train	41.5	21.2	42.1	21.5
Faster R-CNN	RPN, 300	COCO trainval	-	-	42.7	21.9

# Faster RCCN 수행 시간 비교

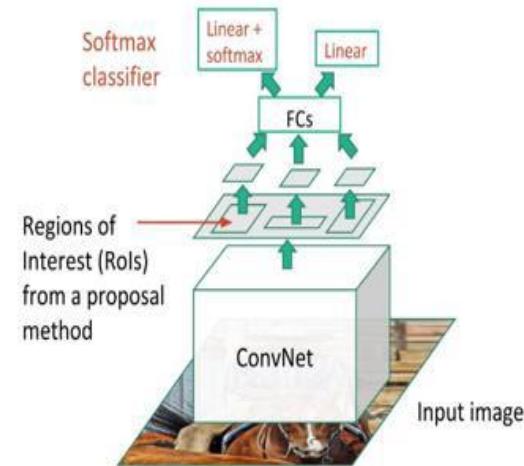
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

model	system	conv	proposal	region-wise	total	rate
VGG	SS+Fast R-CNN	146	1510	174	1830	0.5fps
VGG	RPN+Fast R-CNN	141	10	47	198	5fps
ZF	RPN+Fast R-CNN	31	3	25	59	17fps

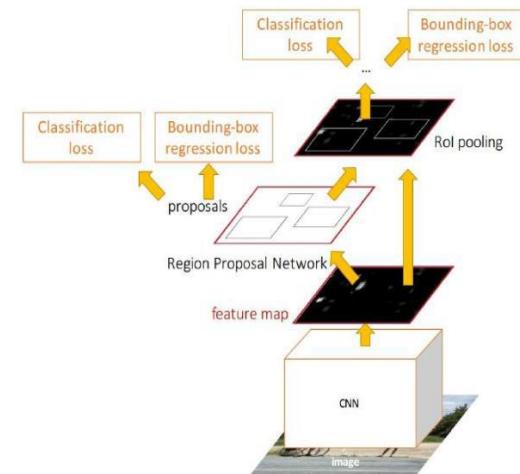
RCNN



Fast RCNN



Faster RCNN





### 장점

- 딥러닝 개발 프레임 워크 없이 쉽게 Inference 구현 가능
- OpenCV 에서 지원하는 다양한 Computer vision 처리 API 와 Deep learning 을 쉽게 결합

### 단점

- GPU 지원 기능이 약함 .
- DNN 모듈은 과거에 NVIDIA GPU 지원 안됨 . 2019년 10월에 Google에서 NVIDIA GPU 지원 발표함 . 아직 환경 구성/설치가 어려움 . 점차 개선이 예상됨
- OpenCV는 모델을 학습할 수 있는 방법을 제공하지 않으며 오직 Inference 만 가능
- CPU 기반에서 Inference 속도가 개선되었으나 , GPU( 지원 여건이 안정적이지 않음

- OpenCV 는 자체적으로 딥러닝 가중치 모델을 생성하지 않고 타 Framework 에서 생성된 모델을 변환하여 로딩함.
- Dnn 패키지는 파일로 생성된 타 프레임워크 모델을 로딩할 수 있도록 readNetFromXXX (가중치 모델파일 , 환경 파일 ) API 제공
- 가중치 모델파일은 타 프레임워크 모델 파일 , 환경 파일은 타 프레임워크 모델 파일의 환경(Config ) 파일을 DNN 패키지에서 다시 변환한 환경 파일
- ONNX 모델 지원도 가능하지만 , 아직은 지원이 충분하지 않은 상황

Framework	타 Framework 모델 로딩	특징
Tensorflow	cvNet = Cv2.dnn.readNetFromTensorflow (가중치 모델 파일 , 환경 파일)	가장 많은 유형의 Detection/Segmentation 모델 제공 (가장 다양한 Base Network 지원)
Darknet	cvNet = Cv2.dnn.readNetFromDarknet(가중치 모델 파일 , 환경 파일)	오직 YOLO 모델로만 로딩 가능
Torch	cvNet = Cv2.dnn.readNetFromTorch(가중치 모델 파일 , 환경 파일)	
Caffe	cvNet = Cv2.dnn.readNetFromCaffe(가중치 모델 파일 , 환경 파일)	

<https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API>

### OpenCV지원 Tensorflow 모델 리스트

Model	Version	weights	config
MobileNet-SSD v1	2017_11_17	<a href="#">weights</a>	<a href="#">config</a>
MobileNet-SSD v1 PPN	2018_07_03	<a href="#">weights</a>	<a href="#">config</a>
MobileNet-SSD v2	2018_03_29	<a href="#">weights</a>	<a href="#">config</a>
Inception-SSD v2	2017_11_17	<a href="#">weights</a>	<a href="#">config</a>
Faster-RCNN Inception v2	2018_01_28	<a href="#">weights</a>	<a href="#">config</a>
Faster-RCNN ResNet-50	2018_01_28	<a href="#">weights</a>	<a href="#">config</a>
Mask-RCNN Inception v2	2018_01_28	<a href="#">weights</a>	<a href="#">config</a>

Tensorflow API로 Pretrained 된 모델 (Frozen graph)를 다운로드

Pretrained된 모델 (Frozen graph)를 구동을 위한 Config 다운로드

Inference 수행 시간 위주

Inference 정확도 위주

Tensorflow 1.x로 만들어진 모델을 지원

Tensorflow 2.x지원은 아직 미정

### ① 가중치 모델 파일과 환경 설정 파일을 로드하여 Inference 네트워크 모델 생성

```
cvNet=cv2.dnn.readNetFromTensorflow('frozen_inference_graph.pb','graph.pbtx')
```

```
img=cv2.imread('img.jpg')  
rows, cols, channels=image.shape
```

### ② 입력 이미지를 Preprocessing하여 네트워크에 입력

```
cvNet.setInput(cv2.dnn.blobFromImage(img,size=(300,300),swapRB=True,crop=False))
```

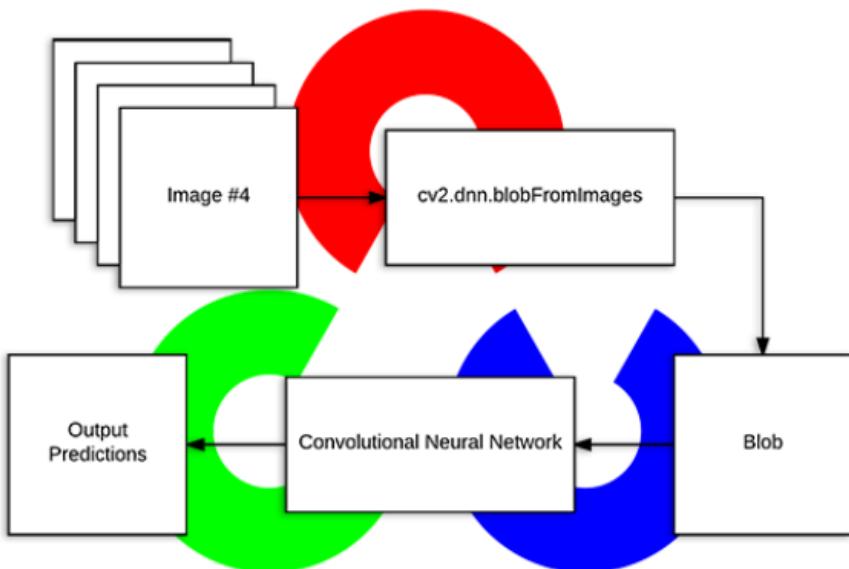
### ③ Inference 네트워크에서 Output추출

```
networkOutput=cvNet.forward()
```

### ④ 추출된 output에서 detect 정보를 기반으로 원본 image 위에 object detection 시각화

```
for detention in networkOutput[0,0]:
```

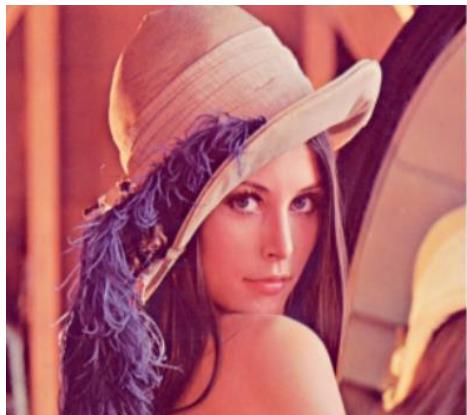
object detected된 결과, bounding box 좌표, 예측 레이블들을 원본 image 위에 시각화 로직



Image를 Preprocessing수행하여 네트웍에 입력할 수 있게 제공

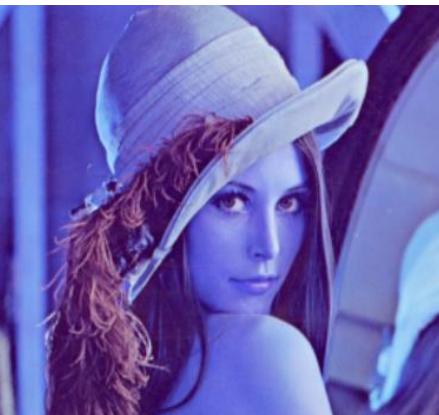
1. 이미지 사이즈 고정
2. 이미지 값 스케일링
3. BGR을 RGB로 변경, 이미지를 Cropt할 수 있는 옵션 제공

RGB



원본 이미지

BGR



OpenCV.imread()로  
읽은 이미지

OpenCV는 RGB 이미지를 BGR 형태로 저장하므로 원본 Image그대로 저장하려면 cvtColor()를 이용하여 다시 RGB로 변환 필요.

```
img_bgr=cv2.imread('원본image')
img_rgb=cv2.cvtColor(img_bgr,cv2.COLOR_BGR2RGB)
```

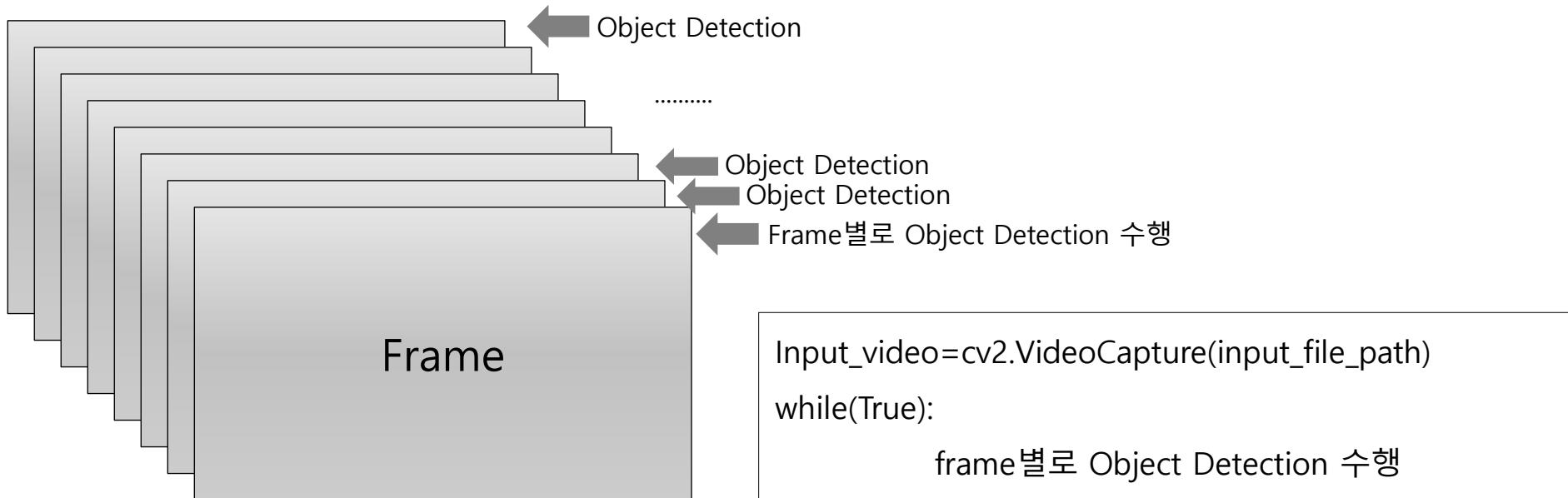
blobFromImage()의 swapRB=True는 cv2.imread()로 로딩된 BGR 형태의 데이터를 RGB로 변환하여 네트워크로 입력함.

```
img_bgr=cv2.imread('원본image')
cvNet.setInput(cv2.dnn.blobFromImage(
    img_bgr, size=(300,300), swapRB=True, crop=False))
```

# OpenCV Video Stream Capture를 이용한 Video Object Detection

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

OpenCV의 videoCapture() API를 이용하여 Video Stream을 Frame by Frame 별로 Capture한 Image에 Object Detection을 수행하는 방식



ID	COCO Paper	COCO 2017	Group		ID	COCO Paper	COCO 2017	Group
1	person	person	person		77	cell phone	cell phone	electronic
2	bicycle	bicycle	vehicle		78	microwave	microwave	appliance
3	car	car	vehicle		79	oven	oven	appliance
4	motorcycle	motorcycle	vehicle		80	toaster	toaster	appliance
5	airplane	airplane	vehicle		81	sink	sink	appliance
6	bus	bus	vehicle		82	refrigerator	refrigerator	appliance
7	train	train	vehicle	80개의 오브젝트 카테고리	83	blender	-	appliance
8	truk	truk	vehicle		84	book	book	indoor
9	boat	boat	vehicle		85	clock	clock	indoor
10	traffic light	traffic light	vehicle		86	vase	trvaset	indoor
11	fire hydrant	fire hydrant	outdoor		87	scissors	scissors	indoor
12	street sign	-	outdoor		88	teddy bear	teddy bear	indoor
13	stop sign	stop sign	outdoor		89	hair drier	hair drier	indoor
14	parking meter	parking meter	outdoor		90	toothbrush	toothbrush	indoor
15	bench	bench	outdoor		91	hair brush	-	indoor

# OpenCV와 Tensorflow의 Coco 클래스 id와 name 매핑

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

Class id가 0부터 ~90

```
0:'person',1:'bicycle',2:'car',3:'motorcycle',4:'airplane',5:'bus',6:'train',7:'truck',8:'boat',9:'traffic light',
10:'fire hydrant',11:'street sign',12:'stop sign',13:'parking meter',14:'bench',15:'bird',16:'cat',17:'dog',18:'horse',19:'sheep',
20:'cow',21:'elephant',22:'bear',23:'zebra',24:'giraffe',25:'hat',26:'backpack',27:'umbrella',28:'shoe',29:'eye glasses',
30:'handbag',31:'tie',32:'suitcase',33:'frisbee',34:'skis',35:'snowboard',36:'sports ball',37:'kite',38:'baseball bat',39:'baseball glove',
40:'skateboard',41:'surfboard',42:'tennis racket',43:'bottle',44:'plate',45:'wine glass',46:'cup',47:'fork',48:'knife',49:'spoon',
50:'bowl',51:'banana',52:'apple',53:'sandwich',54:'orange',55:'broccoli',56:'carrot',57:'hot dog',58:'pizza',59:'donut',
60:'cake',61:'chair',62:'couch',63:'potted plant',64:'bed',65:'mirror',66:'dining table',67:'window',68:'desk',69:'toilet',
70:'door',71:'tv',72:'laptop',73:'mouse',74:'remote',75:'keyboard',76:'cell phone',77:'microwave',78:'oven',79:'toaster',
80:'sink',81:'refrigerator',82:'blender',83:'book',84:'clock',85:'vase',86:'scissors',87:'teddy bear',88:'hair drier',89:'toothbrush',
90:'hair brush'
```

Class id가 1부터 ~91

```
1:'person',2:'bicycle',3:'car',4:'motorcycle',5:'airplane',6:'bus',7:'train',8:'truck',9:'boat',10:'traffic light',
11:'fire hydrant',12:'street sign',13:'stop sign',14:'parking meter',15:'bench',16:'bird',17:'cat',18:'dog',19:'horse',20:'sheep',
21:'cow',22:'elephant',23:'bear',24:'zebra',25:'giraffe',26:'hat',27:'backpack',28:'umbrella',29:'shoe',30:'eye glasses',
31:'handbag',32:'tie',33:'suitcase',34:'frisbee',35:'skis',36:'snowboard',37:'sports ball',38:'kite',39:'baseball bat',40:'baseball glove',
41:'skateboard',42:'surfboard',43:'tennis racket',44:'bottle',45:'plate',46:'wine glass',47:'cup',48:'fork',49:'knife',50:'spoon',
51:'bowl',52:'banana',53:'apple',54:'sandwich',55:'orange',56:'broccoli',57:'carrot',58:'hot dog',59:'pizza',60:'donut',
61:'cake',62:'chair',63:'couch',64:'potted plant',65:'bed',66:'mirror',67:'dining table',68:'window',69:'desk',70:'toilet',
71:'door',72:'tv',73:'laptop',74:'mouse',75:'remote',76:'keyboard',77:'cell phone',78:'microwave',79:'oven',80:'toaster',
81:'sink',82:'refrigerator',83:'blender',84:'book',85:'clock',86:'vase',87:'scissors',88:'teddy bear',89:'hair drier',90:'toothbrush',
91:'hair brush'
```

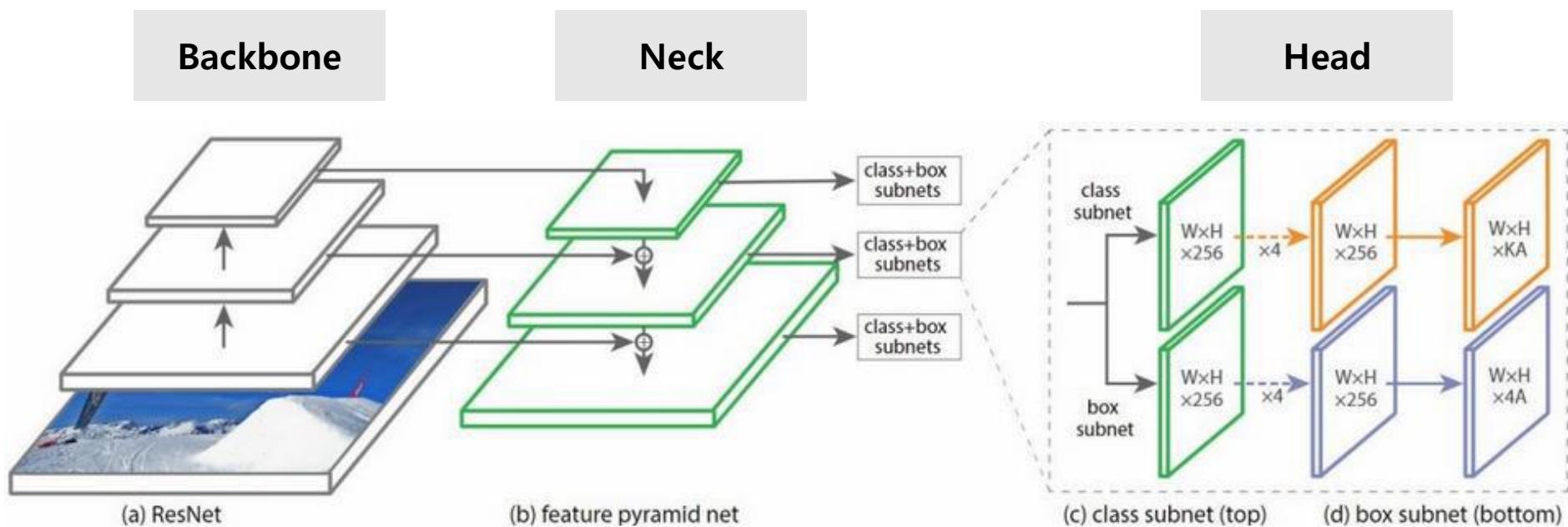
Class id가 0부터 ~79

```
0:'person',1:'bicycle',2:'car',3:'motorbike',4:'aeroplane',5:'bus',6:'train',7:'truck',8:'boat',9:'traffic light',10:'fire hydrant',
11:'stop sign',12:'parking meter',13:'bench',14:'bird',15:'cat',16:'dog',17:'horse',18:'sheep',19:'cow',20:'elephant',
21:'bear',22:'zebra',23:'giraffe',24:'backpack',25:'umbrella',26:'handbag',27:'tie',28:'suitcase',29:'frisbee',30:'skis',
31:'snowboard',32:'sports ball',33:'kite',34:'baseball bat',35:'baseball glove',36:'skateboard',37:'surfboard',38:'tennis racket',39:'cup',
40:'fork',41:'knife',42:'spoon',43:'bowl',44:'banana',45:'apple',46:'sandwich',47:'orange',48:'broccoli',
49:'carrot',50:'hot dog',51:'pizza',52:'donut',53:'cake',54:'chair',55:'sofa',56:'pottedplant',57:'bed',58:'diningtable',
59:'toilet',60:'tvmonitor',61:'laptop',62:'mouse',63:'remote',64:'keyboard',65:'cell phone',66:'microwave',67:'oven',68:'toaster',
69:'sink',70:'refrigerator',71:'blender',72:'book',73:'clock',74:'vase',75:'scissors',76:'teddy bear',77:'hair drier',78:'toothbrush'
```

# OpenCV DNN 모델별 Coco 클래스 id와 name 매핑

1.3 R-CNN, Fast R-CNN, Faster R-CNN

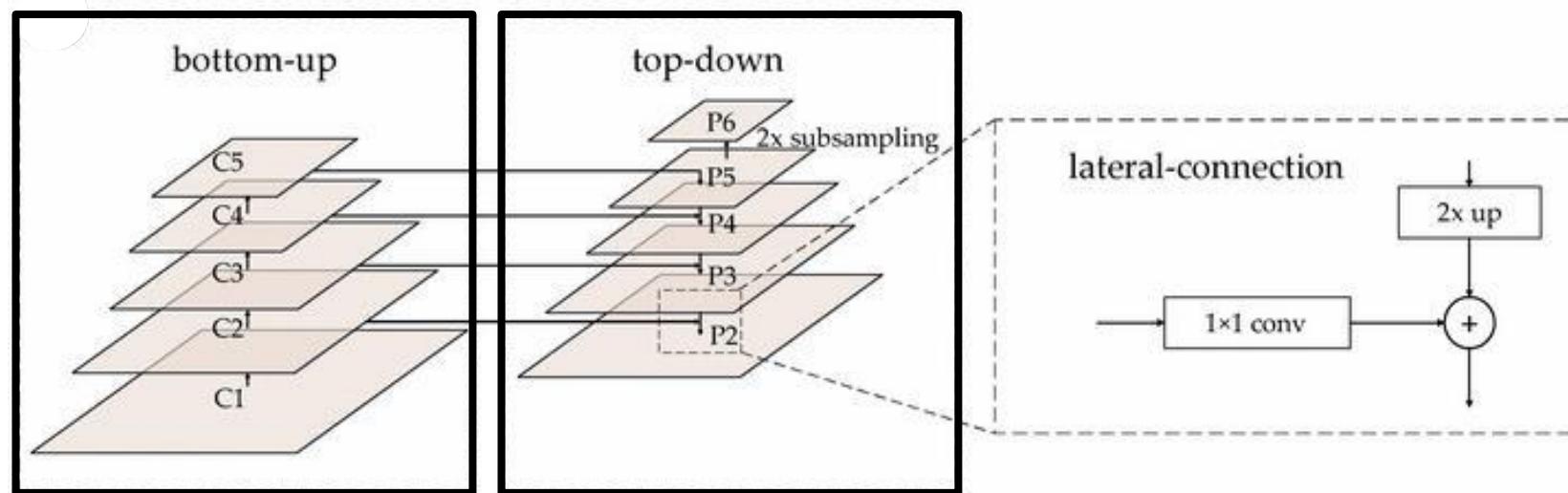
OpenCV DNN	Tensorflow Faster RCNN 모델 로드	0~90
	Tensorflow SSD 모델 로드	1~91
	Tensorflow Mask RCNN 모델 로드	0~90
	Darknet YOLO 모델 로드	0~79



# Feature Pyramid Network

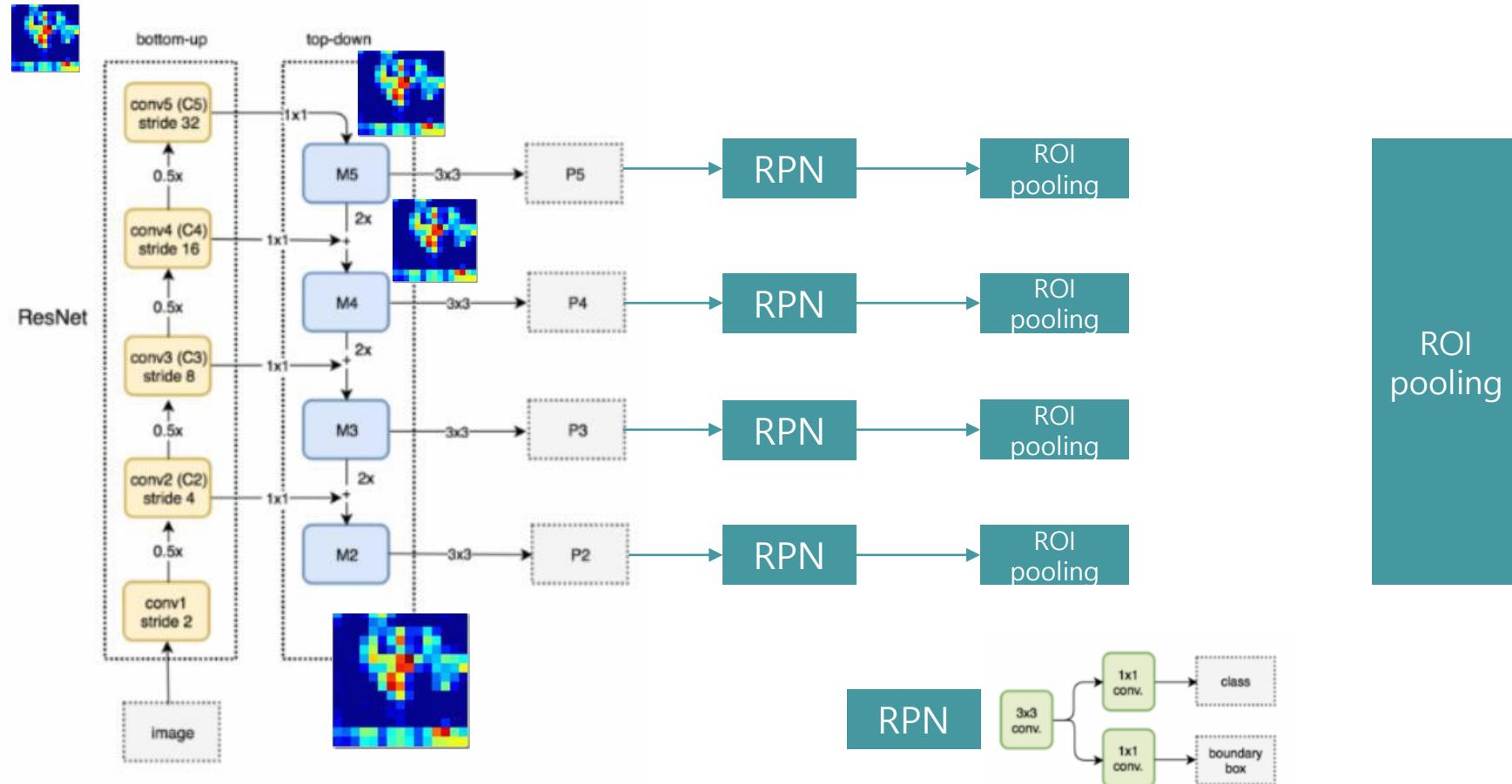
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

- 작은 Object들을 보다 잘 Detect하기 위해서 다양한 Feature Map을 활용
- 상위 Feature map의 추상화 된 정보와 하위 Feature map의 정보를 효과적으로 결합

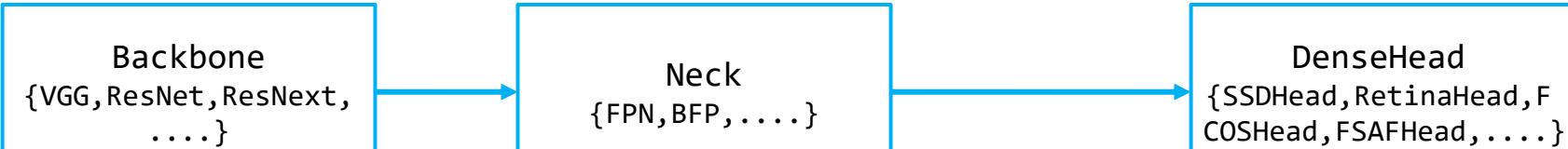


# FPN을 활용한 Faster RCNN RPN

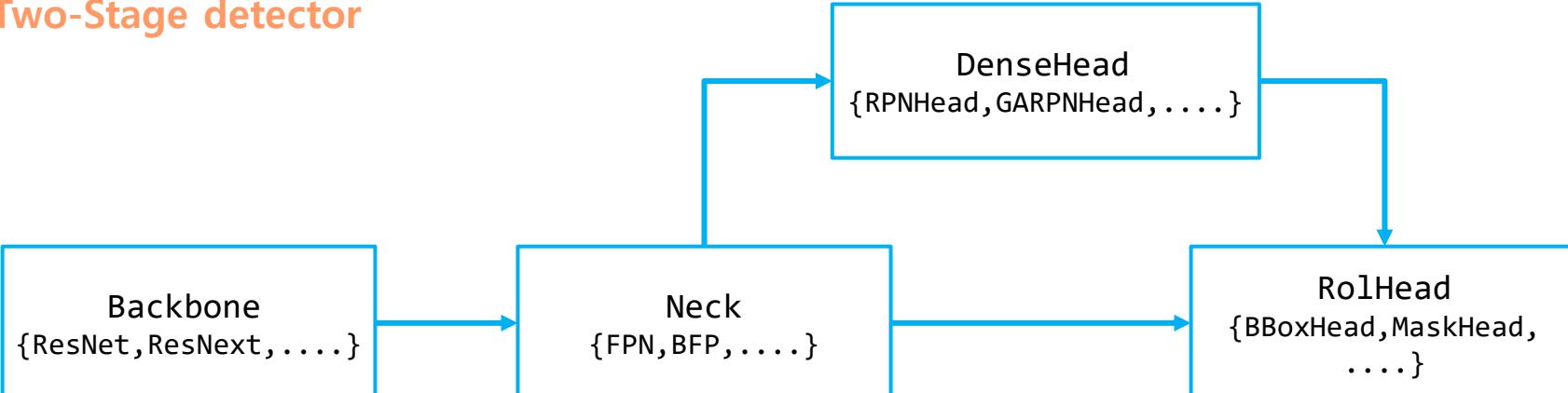
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



### Single-Stage detector

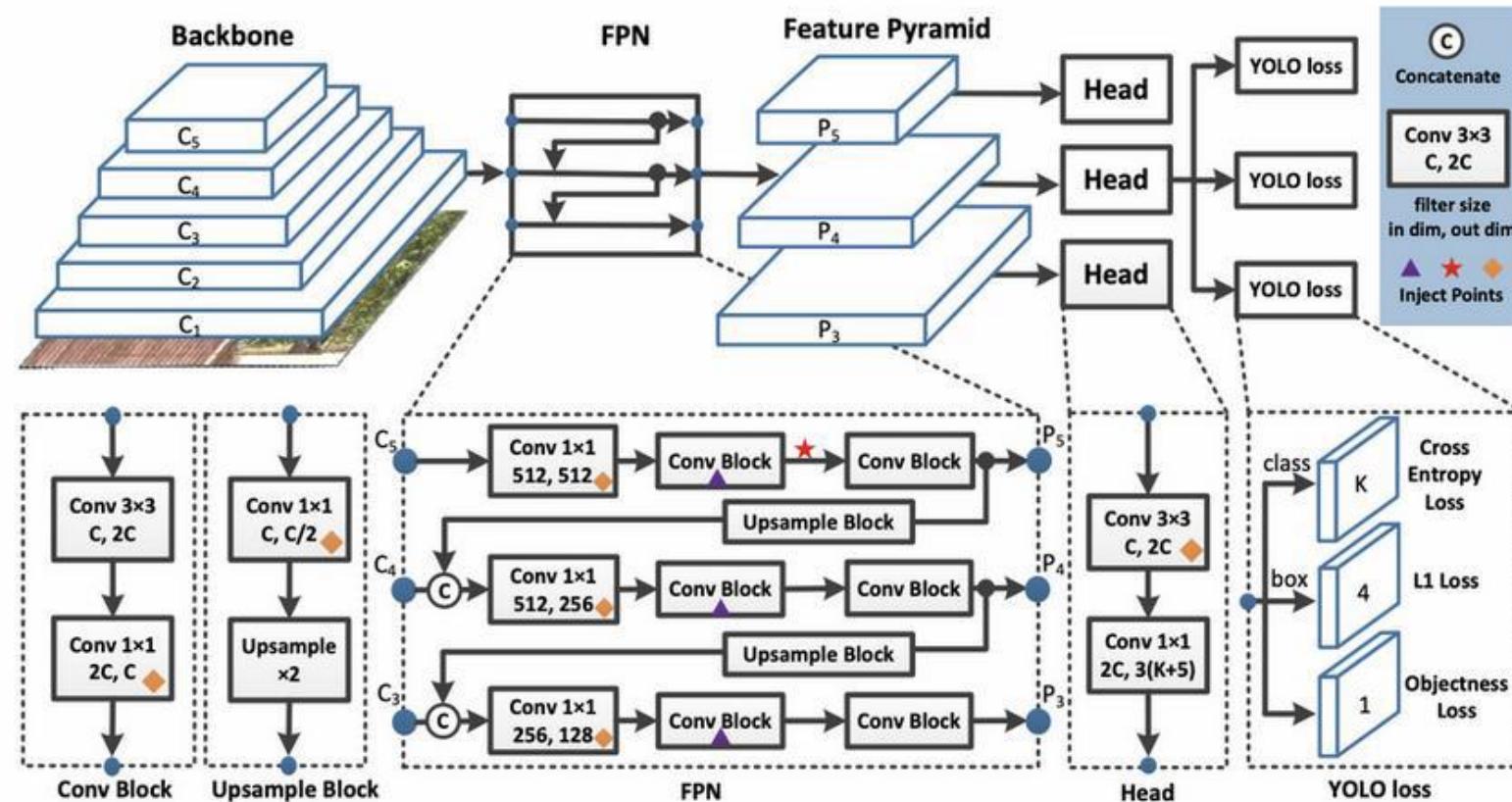


### Two-Stage detector



# YOLO V3 모델 아키텍처

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



# Pytorch 기반의 주요 Object Detection Segmentation 패키지

1.3 R-CNN, Fast R-CNN, Faster R-CNN

여러 개의 Object Detection/Segmentation 알고리즘을 패키지화



**Code** 기반

지원 알고리즘이 많지 않음



**Config** 기반

Facebook Research에서 주도



**Config** 기반

중국 칭화대학 중심의 OpenMMLab 주도

### MMDetection:

Open MMLab Detection Toolbox and Benchmark(<https://arxiv.org/pdf/1906.0755.pdf>)



- 칭화대학(중국, 베이징시)의 주도로 만들어진 Computer vision Open Source project인 OpenMMLab에서 출발
- 2018년 MS-COCO Challenge에서 우승 후 모듈을 확장하여 다수의 알고리즘 수용
- 최신의 다양한 Object Detection, Segmentation 알고리즘을 Package로 구현제공
- 뛰어난 구현 성능, 효율적인 모듈 설계, Config기반으로 데이터부터 모델 학습/평가 까지 이어지는 간편한 파이프라인 적용
- Pytorch 기반으로 구현

## 백본(backbone) 지원

- ResNet (CVPR'2016)
- ResNeXt (CVPR'2017)
- VGG (ICLR'2015)
- HRNet (CVPR'2019)
- RegNet (CVPR'2020)
- Res2Net (TPAMI'2020)
- ResNeSt (ArXiv'2020)

- RPN (NeurIPS'2015)
- Fast R-CNN (ICCV'2015)
- Faster R-CNN (NeurIPS'2015)
- Mask R-CNN (ICCV'2017)
- Cascade R-CNN (CVPR'2018)
- Cascade Mask R-CNN (CVPR'2018)
- SSD (ECCV'2016)
- RetinaNet (ICCV'2017)
- GHM (AAAI'2019)
- Mask Scoring R-CNN (CVPR'2019)
- Double-Head R-CNN (CVPR'2020)
- Hybrid Task Cascade (CVPR'2019)
- Libra R-CNN (CVPR'2019)
- Guided Anchoring (CVPR'2019)
- FCOS (ICCV'2019)
- RepPoints (ICCV'2019)
- Foveabox (TIP'2020)
- FreeAnchor (NeurIPS'2019)
- NAS-FPN (CVPR'2019)
- ATSS (CVPR'2020)
- FSAF (CVPR'2019)

## Object Detection과 Segmentation 지원

- PAFPN (CVPR'2018)
- Dynamic R-CNN (ECCV'2020)
- PointRend (CVPR'2020)
- CARAFE (ICCV'2019)
- DCNv2 (CVPR'2019)
- Group Normalization (ECCV'2018)
- Weight Standardization (ArXiv'2019)
- OHEM (CVPR'2016)
- Soft-NMS (ICCV'2017)
- Generalized Attention (ICCV'2019)
- GCNet (ICCVW'2019)
- Mixed Precision (FP16) Training (ArXiv'2017)
- InstaBoost (ICCV'2019)
- GRoIE (ICPR'2020)
- DetectoRS (ArXiv'2020)
- Generalized Focal Loss (NeurIPS'2020)
- CornerNet (ECCV'2018)
- Side-Aware Boundary Localization (ECCV'2020)
- YOLOv3 (ArXiv'2018)
- PAA (ECCV'2020)
- YOLACT (ICCV'2019)
- CentripetalNet (CVPR'2020)
- VFNet (ArXiv'2020)
- DETR (ECCV'2020)
- Deformable DETR (ICLR'2021)
- CascadeRPN (NeurIPS'2019)
- SCNet (AAAI'2021)
- AutoAssign (ArXiv'2020)
- YOLOF (CVPR'2021)
- Seasaw Loss (CVPR'2021)

### 매우 많은 Prebuilt 모델

- Fast RCNN
- Faster RCNN
- RetinaNet
- .....
- .....

### 일반화된 모델 아키텍처

### MMdetection 모델 아키텍처

- Backbone
- Neck
- DenseHead (AnchorHead/AnchorFreeHead)
- RoIExtractor
- RoIHead (BBoxHead/MaskHead)

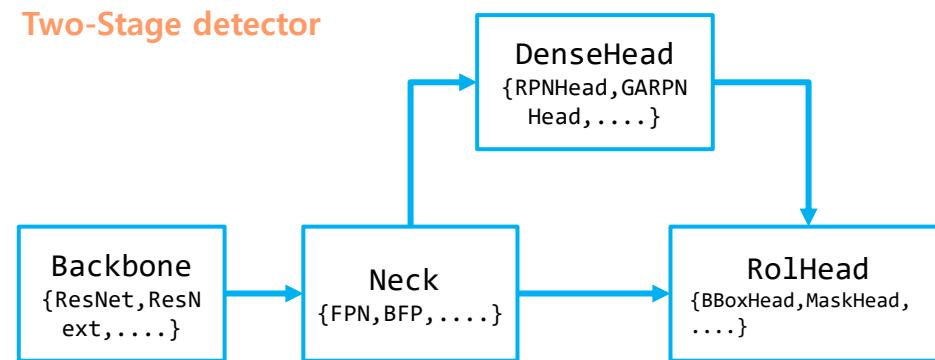
### MMdetection 모델 아키텍처

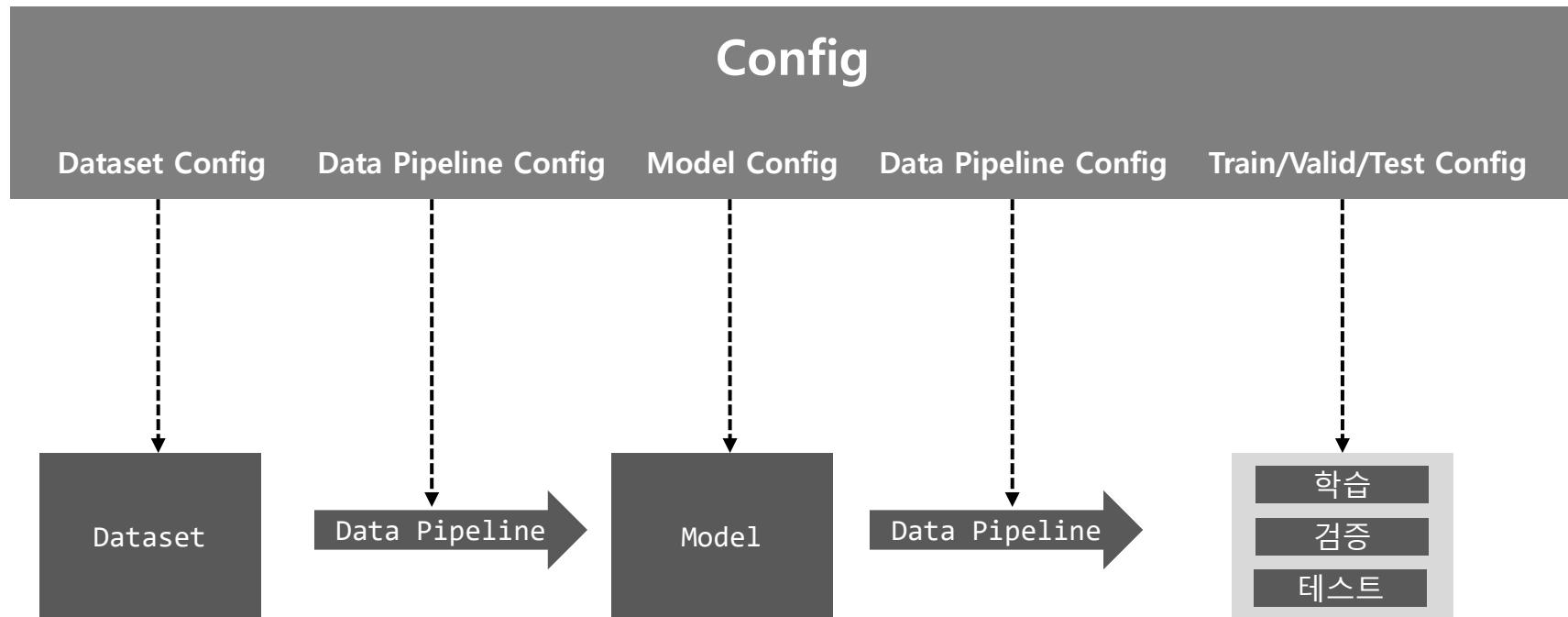
- Backbone: Feature Extractor(이미지-> Feature Map)
- Neck: Backbone과 Heads를 연결하면서 heads가 feature mapn 의 특성을 보다 잘 해석하고 처리할 수 있도록 정제 작업수행
- DenseHead: Feature Map에서 Object의 위치와 Classification을 처리하는 부분
- ROIExtractor: Feature Map에서 ROI 정보를 뽑아내는 부분
- ROIHead (BBoxHead/MaskHead): ROI정보를 기반으로 Object위치와 Classification을 수행하는 부분

#### Single-Stage detector



#### Two-Stage detector





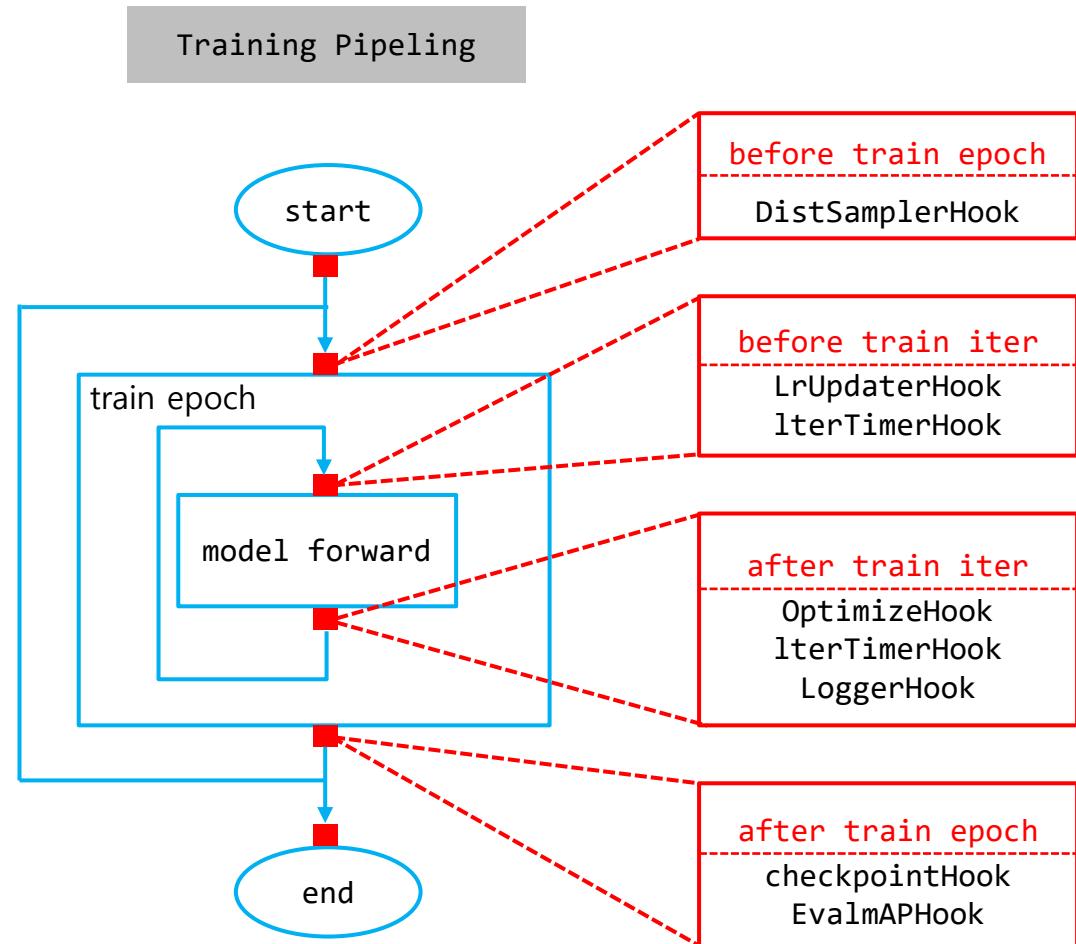
# MMDetection Config 예시

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

```
Config:  
model = dict(  
    type='FasterRCNN',  
    pretrained='open-mmlab://detectron2/resnet50_caffe',  
    backbone=dict(  
        type='ResNet',  
        depth=50,  
        num_stages=4,  
        out_indices=(0, 1, 2, 3),  
        frozen_stages=1,  
        norm_cfg=dict(type='BN', requires_grad=False),  
        norm_eval=True,  
        style='caffe'),  
    neck=dict(  
        type='FPN',  
        in_channels=[256, 512, 1024, 2048],  
        out_channels=256,  
        num_outs=5),  
    rpn_head=dict(  
        type='RPNHead',  
        in_channels=256,  
        feat_channels=256,  
        anchor_generator=dict(  
            type='AnchorGenerator',  
            scales=[8],  
            ratios=[0.5, 1.0, 2.0],  
            strides=[4, 8, 16, 32, 64]),  
        bbox_coder=dict(  
            type='DeltaXYWHBoxCoder',  
            target_means=[0.0, 0.0, 0.0, 0.0],  
            target_stds=[1.0, 1.0, 1.0, 1.0]),  
        loss_cls=dict(  
            type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0),  
        loss_bbox=dict(type='L1Loss', loss_weight=1.0)),  
    ...)
```

```
data = dict(  
    samples_per_gpu=2,  
    workers_per_gpu=2,  
    train=dict(  
        type='RaccoonDataset',  
        ann_file='train.txt',  
        img_prefix='images',  
        pipeline=[  
            dict(type='LoadImageFromFile'),  
            dict(type='LoadAnnotations', with_bbox=True),  
            dict(  
                type='Resize',  
                img_scale=[(1333, 640), (1333, 672), (1333, 704), (1333, 736),  
                          (1333, 768), (1333, 800)],  
                multiscale_mode='value',  
                keep_ratio=True),  
            dict(type='RandomFlip', flip_ratio=0.5),  
            dict(  
                type='Normalize',  
                mean=[103.53, 116.28, 123.675],  
                std=[1.0, 1.0, 1.0],  
                to_rgb=False),  
            dict(type='Pad', size_divisor=32),  
            dict(type='DefaultFormatBundle'),  
            dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])  
        ],  
        data_root='/content/raccoon/'),  
    val=dict(  
        type='RaccoonDataset',  
        ann_file='val.txt',  
        img_prefix='images',  
        pipeline=[  
            dict(type='LoadImageFromFile'),  
            dict(  
                type='MultiScaleFlipAug',
```

- Hook(Callback)을 통해 학습에 필요한 여러 설정들을 Customization 가능
- 대부분 Configuration에서 이를 설정함



- MMDetection의 성공을 발판으로 Computer Vision 전반에 걸친 여러 package로 확산

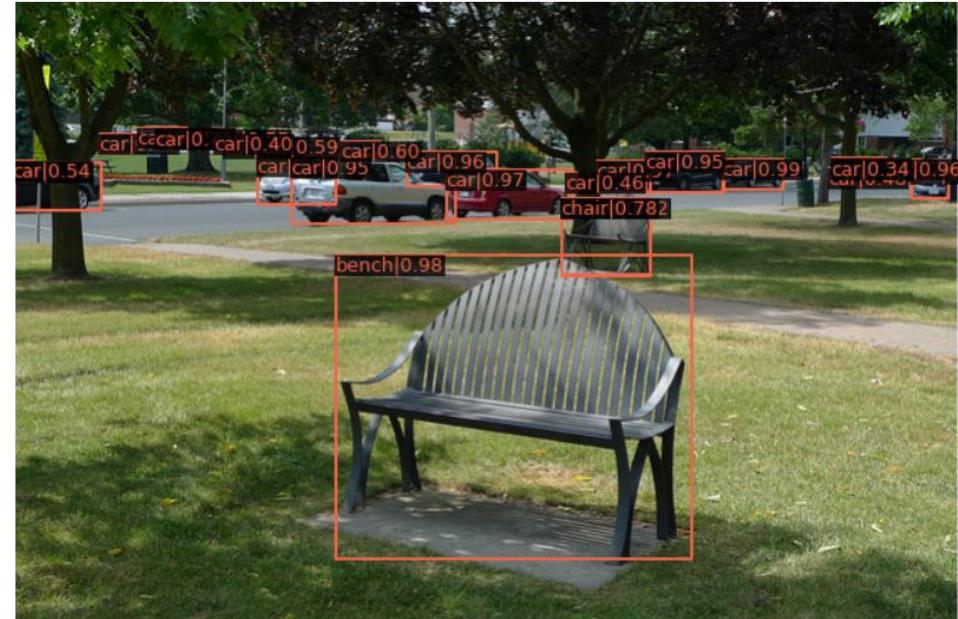
<b>MMCV</b> A foundational python library for computer vision <a href="#">Learn more</a> Stars 2.5k	<b>MMDetection</b> Object detection toolbox and benchmark <a href="#">Learn more</a> Stars 15k	<b>MMDetection3D</b> The next generation toolbox for general 3D detection <a href="#">Learn more</a> Stars 1.1k
<b>MMEditing</b> Image and video editing toolbox for editing tasks <a href="#">Learn more</a> Stars 2.1k	<b>MMAction2</b> Next generation toolbox for action understanding <a href="#">Learn more</a> Stars 848	<b>MMSegmentation</b> Comprehensive semantic segmentation toolbox <a href="#">Learn more</a> Stars 1.8k
<b>MMClassification</b> Image classification toolbox based on PyTorch <a href="#">Learn more</a> Stars 629	<b>MPose</b> Toolbox for pose estimation based on PyTorch <a href="#">Learn more</a> Stars 841	<b>MMTracking</b> Video perception toolbox based on PyTorch <a href="#">Learn more</a> Stars 1.3k
<b>MMOCR</b> Toolbox for text detection, recognition and understanding. <a href="#">Learn more</a> Stars 1.2k	<b>MMGeneration</b> OpenMMLab image and video generative models toolbox. <a href="#">Learn more</a> Stars 342	

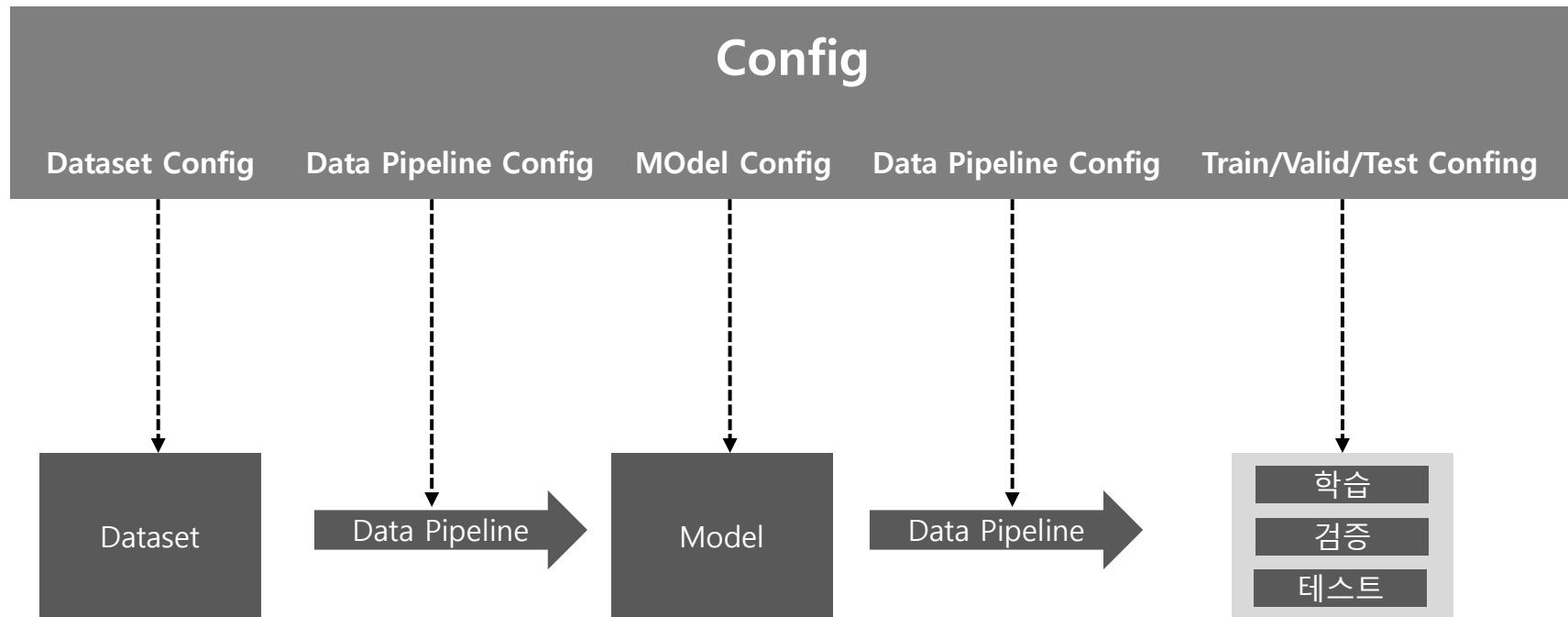
# MS-COCO 학습된 모델의 inference 반환결과

1.3 R-CNN, Fast R-CNN, Faster R-CNN

```
results = inference_detector(model, img)
```

```
x_min   y_min   x_max   y_max   class_confidence score  
class_id 0 [ array([[375.2, 119.7, 381.9, 134.4, 0.13],  
                   [532.3, 109.2, 540.4, 125.3, 0.08]],  
                 dtype=float32)  
  
class_id 1 array([ ], shape=(0, 5), dtype=float32)  
  
class_id 2 array([[609.6, 113.8, 634.5, 136.9, 0.98],  
                   [481.7, 110.4, 522.4, 130.2, 0.98],  
                   [101.8, 112.1, 504.3, 142.2, 0.92],  
                   ....], dtype=float32)  
  
class_id 3 array([ ], shape=(0, 5), dtype=float32)  
  
class_id... ....  
  
class_id 79 array([ ], shape=(0, 5), dtype=32)  
]  
  
class_id 0:'person', 1:'bicycle', 2:'car'
```





- MMDetection은 다양한 유형의 Dataset을 변환 클래스를 통해 지원
- CustomDataset과 CocoDataset이 가장 쉽게 적용 될 수 있는 Dataset

<https://github.com/open-mmlab/mmdection/tree/master/mmdet/datasets>

CustomDataset

CocoDataset

VOCDataset

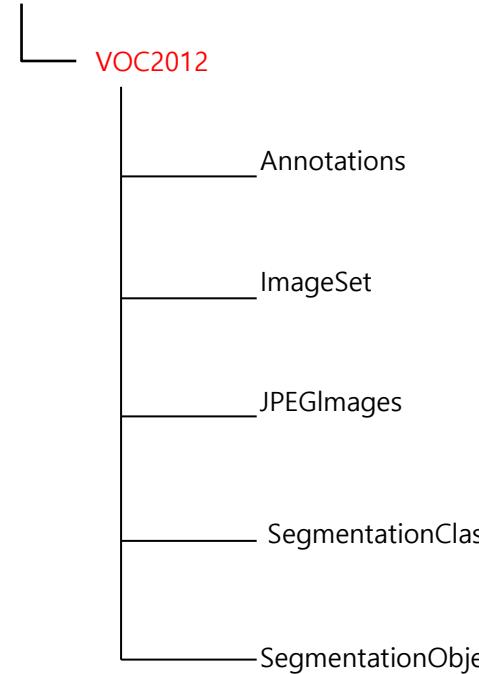
WIDERFaceDataset

XMLDataset

# PASCAL VOC 데이터 세트 특징

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

VOCdevkit

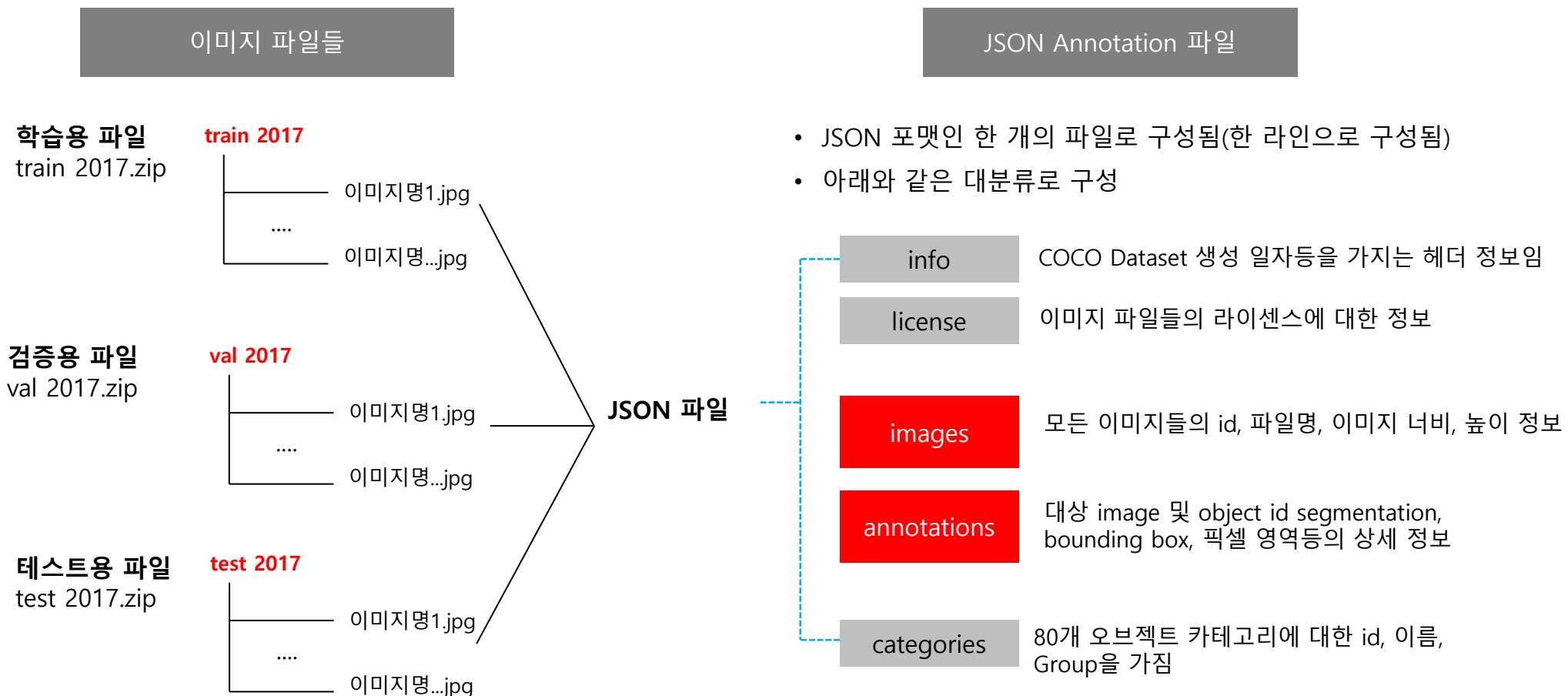


이미지 1개당 1개의  
annotation파일을 가짐



```
<annotation>
    <folder>VOC2012</folder>
    <filename>2007_000032.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
    </source>
    <size>
        <width>500</width>
        <height>281</height>
        <depth>3</depth>
    </size>
    <segmented>1</segmented>
    <object> ← 개별 Object정보
        <name>aeroplane</name>
        <pose>Frontal</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>104</xmin>
            <ymin>78</ymin>
            <xmax>375</xmax>
            <ymax>183</ymax>
        </bndbox>
    </object>
    <object> ← 개별 Object정보
        <name>aeroplane</name>
        <pose>Left</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>133</xmin>
            <ymin>88</ymin>
            <xmax>197</xmax>
            <ymax>123</ymax>
        </bndbox>
    </object>
</annotation>
```

모든 이미지들에 대해서 단 1개의 Annotation 파일을 가짐



- 모든 이미지들에 대한 annotation 정보들을 list 객체로 가짐
- list내의 개별 원소는 dict로 구성되면 개별 dict 는 1개 이미지에 대한 annotation 정보를 가짐
- 1개 이미지는 여러 개의 Object bbox와 labels annotation 정보들을 개별 dict로 가짐
- 1개 이미지의 Object bbox와 2차원 array로, Object label은 1차원 array로 구성

```
[  
  {  
    'filename' : 'a.jpg',  
    'width' : 1280,  
    'height' : 720,  
    'ann' : {  
      'bboxes' : <np.ndarray, float32> (n, 4),  
      'labels' : <np.ndarray, int64> (n, ),  
      'bboxes_ignore' : <np.ndarray, float32> (k, 4),  
      'labels_ignore' : <np.ndarray, int64> (k, ) (optional field)  
    }  
  },  
  ....  
]
```

# CustomDataset 구조 개요

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

```
@DATASETS.register_module() ← Dataset 객체를 Config에 등록
class CustomDataset(Dataset):
    CLASSES=None
    ↓
    Config로 부터 객체 생성시 인자가 입력됨
    def_init_(self, ann_file, pipeline, classes=None, data_root=None,
               img_prefix=" ", ,):
        self.ann_file = ann_file
        self.data_root = data_root
        self.img_prefix = img_prefix
        self.CLASSES = self.get_classes(classes)

        self.ann_file = osp.join(self.data_root, self.ann_file)
        self.img_prefix = osp.join(self.data_root, self.img_prefix)
        .....
        self.pipeline = Compose(pipeline)

def load_annotations(self, ann_file): ← Middle Format으로 변환하는 부분
    직접 Customize코드를 작성해야 함
    return mmcv.load(ann_file)
```

# CustomDataset을 상속받아 Dataset 생성

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

```
from mmdet.datasets.builder import DATASETS
from mmdet.datasets.custom import CustomDataset

# 반드시 아래 Decorator 설정 할것 @DATASETS.register_module() 설정 시 force=True를 입력하지 않으면 |
@DATASETS.register_module(force=True)      CustomDataset을 상속한 클래스 생성
class KittyTinyDataset(CustomDataset):
    CLASSES = ('Car', 'Truck', 'Pedestrian', 'Cyclist')

    # annotation에 대한 모든 파일명을 가지고 있는 텍스트 파일을 __init__(self, ann_file)로 입력 받고,
    def load_annotations(self, ann_file):      load_annotations()을 재정의하여 middle format으로 원본 소스를 변환
        cat2label = {k:i for i, k in enumerate(self.CLASSES)}
        image_list = mmcv.list_from_file(self.ann_file)
        # 포맷 종류 데이터를 담을 list 객체
        data_infos = []

        for image_id in image_list:

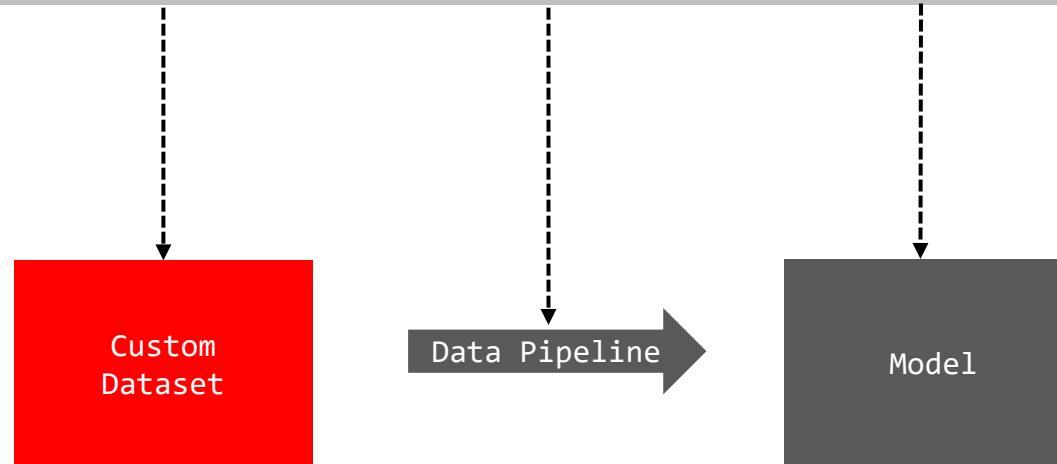
            filename = '{0}/{1}.jpeg'.format(self.img_prefix, image_id)
            # 원본 이미지의 너비, 높이를 image를 직접 로드하여 구함.
            image = cv2.imread(filename)
            height, width = image.shape[:2]
            # 개별 image의 annotation 정보 저장용 Dict 생성. key값 filename에는 image의 파일명만 들어감(
            data_info = {'filename': str(image_id) + '.jpeg',
                        'width': width, 'height': height}
            # 개별 annotation이 있는 서브 딕토리의 prefix 변환.
            label_prefix = self.img_prefix.replace('image_2', 'label_2')
            # 개별 annotation 파일을 1개 line 씩 읽어서 list 로드
            lines = mmcv.list_from_file(osp.join(label_prefix, str(image_id)+'.txt'))
```

# MMDetection Custom Dataset 생성 주요 로직

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

0. Dataset을 위한 Config설정(data\_root, ann\_file, img\_prefix)

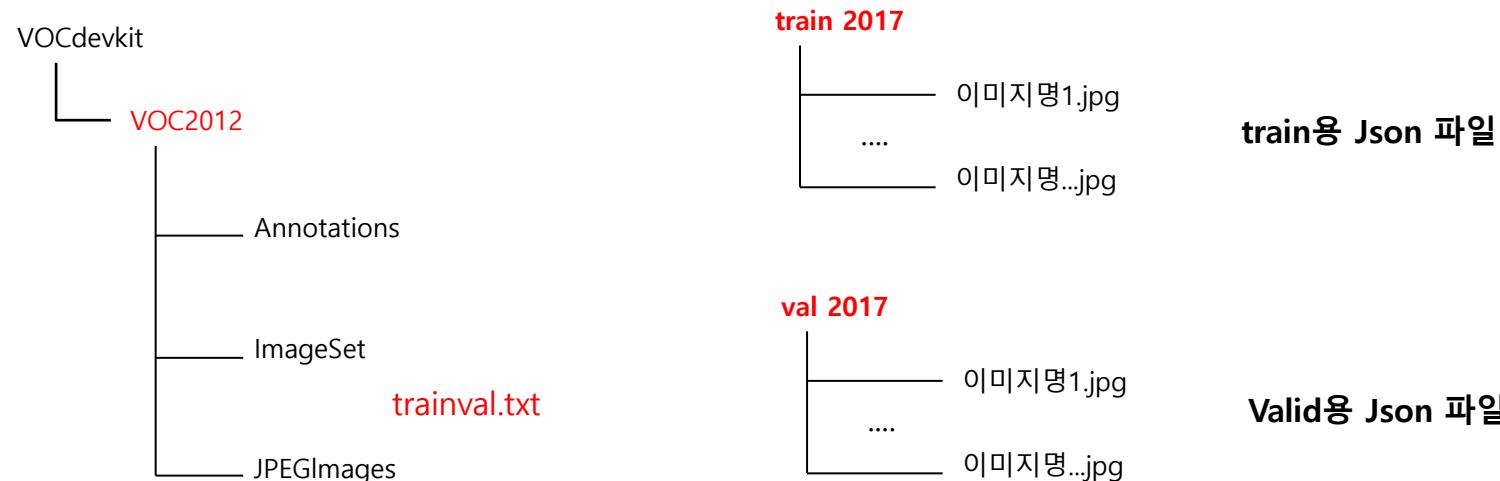
1. CustomDataset 객체를 MMDetection Framework에 등록
2. Config에 설정된 주요 값으로 CustomDataset 객체 생성



Dataset은 학습용, 검증용, 테스트용으로 각각 만들어 질 수 있어야 한다.

### 소스 데이터들의 학습용, 검증용, 테스트용 분리 유형

- image들과 annotation 파일이 학습용, 검증용, 테스트용 디렉토리에 별도로 분리
- 별도의 메타 파일에서 학습용, 검증용, 테스트용 image들과 annotation 파일을 지정
- image들은 학습용, 검증용, 테스트용 디렉토리 별로 분리, annotation은 학습용, 검증용, 테스트용으로 하나만 가짐



# **data\_root, ann\_file, img\_prefix의 활용**

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

학습용 Dataset 생성



학습 Data Config 설정

```
train.data_root=/content/kitti_tiny  
train.ann_file='train.txt'  
train.img_prefix='training/image_2'
```

검증용 Dataset 생성



검증 Data Config 설정

```
val.data_root=/content/kitti_tiny  
val.ann_file='val.txt'  
val.img_prefix='training/image_2'
```

```
train_dataset =
```

```
    kittiTinyDataset (ann_file=train.ann_file,  
                      data_root=train.data_root,  
                      img_prefix=train.img_prefix)
```

```
val_dataset =
```

```
    kittiTinyDataset (ann_file=val.ann_file,  
                      data_root=val.data_root,  
                      img_prefix=val.img_prefix)
```

☞ img\_prefix는 여러 개의 image들을 포함할 수 있는 디렉토리 형태로 지정되지만, ann\_file은 **단 하나만**(일반적으로 file) 지정 할 수 있음

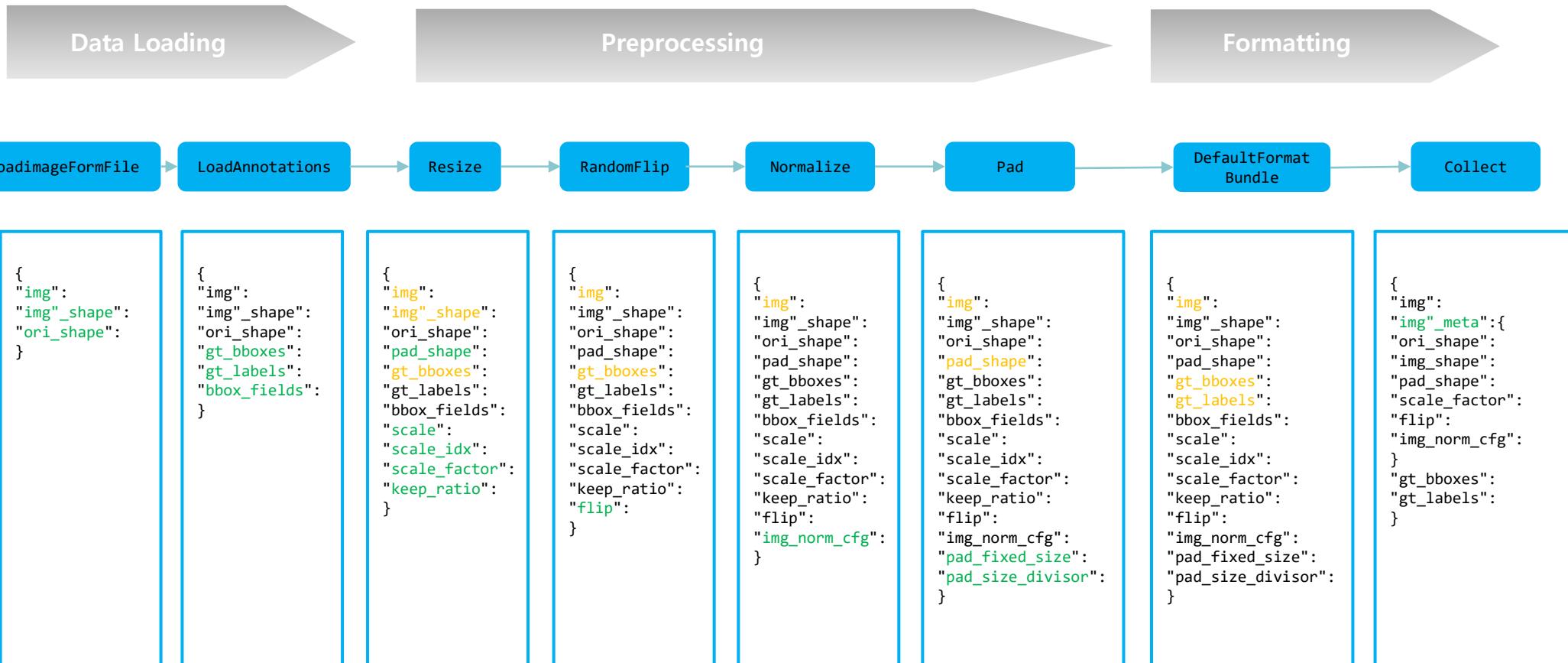
# Config 대분류 및 주요 설정 내역

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN

Config 대분류	설명
dataset	dataset의 type(CustomDataset, CocoDataset 등), train/val/test Dataset 유형, data_root, train/val/test Dataset의 주요 파라미터 설정(type, ann_file, img_prefix, Pipeline 등)
model	Object Detection Model의 backbone, neck, dense head, roi extractor, roi head 주요 영역별로 세부 설정
schedule	optimizer 유형 설정(SGD, Adam, RMSprop 등), 최초 learning 설정 학습 중 동적 Learning rate 적용 정책 설정(step, cyclic, CosineAnnealing 등) train 시 epochs 횟수
run time	주로 hook(callback) 관련 설정 학습 중 checkpoint 파일, log 파일 생성을 위한 interval epochs 수

# Data Pipeline 구성

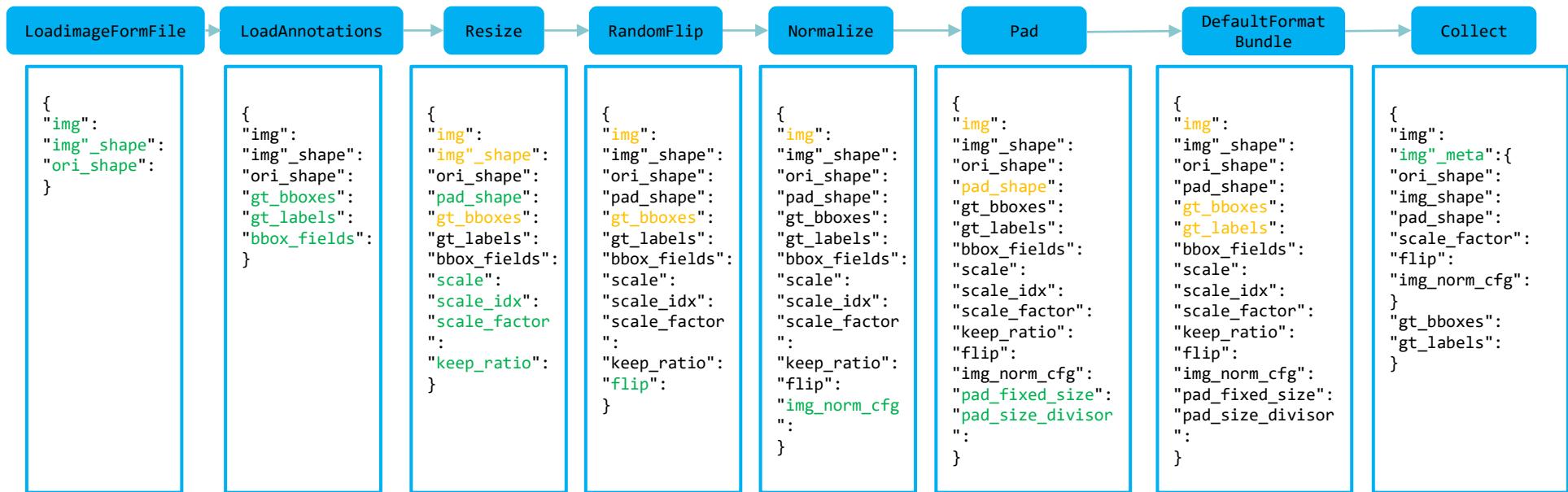
## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



순차적으로 파이프라인이 진행 될 때마다 개별 작업은 **새로운 key값을 추가하거나**(녹색으로 표시), **기존 key값을 update 적용**(주황색으로 표시)

# Data Pipeling 구성

## 1.3 R-CNN, Fast R-CNN, Faster R-CNN



```

img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normaliza', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels']),
]
  
```

```

test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFliAug',
        img_scale=(1333, 800),
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normaliza', **img_norm_cfg),
            dict(type='Pad', size_divisor=32),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
  ]
  
```

# 1. 오브젝트 디텍션

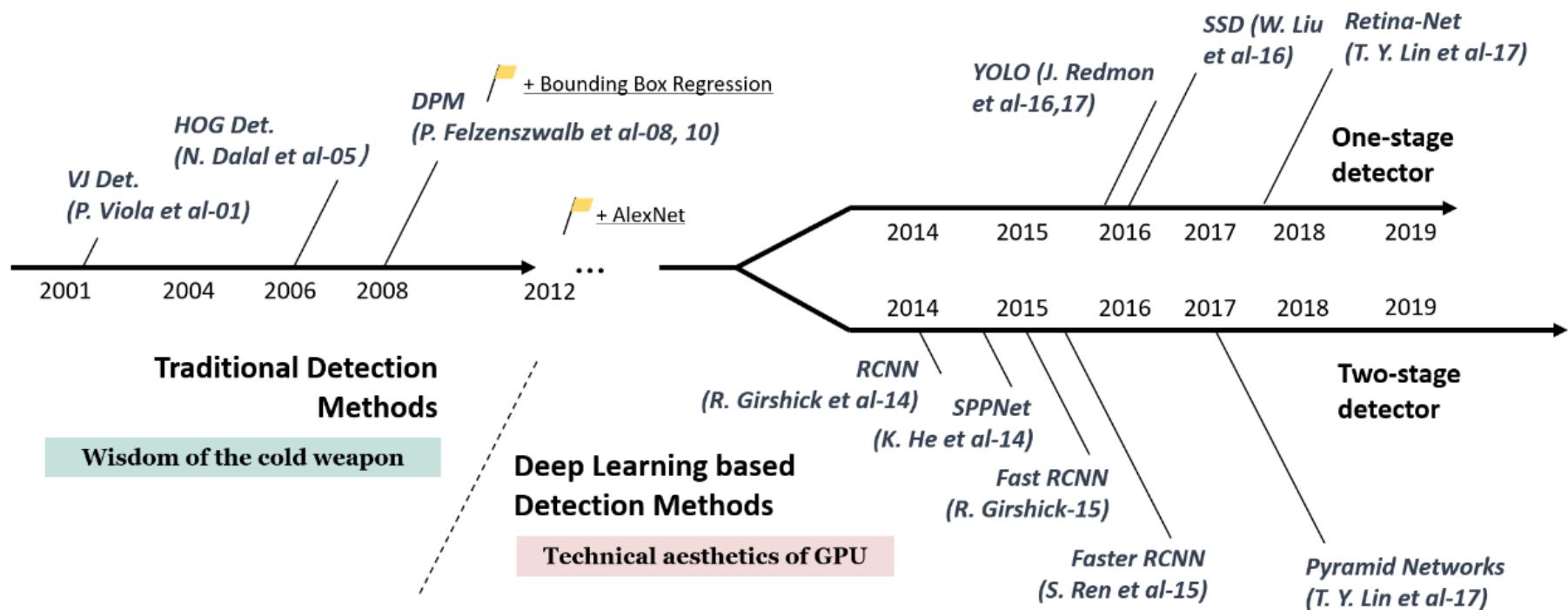
---

- 1.1 오브젝트 디텍션 개요
  - 1.2 OpenCV 개요
  - 1.3 R-CNN, Fast R-CNN, Faster R-CNN
  - 1.4 SSD (Single Shot MultiBox Detector)**
  - 1.5 YOLO (You Only Look Once) 시리즈
  - 1.6 RetinaNet
-

# Object Detection History

1.4 SSD (Single Shot MultiBox Detector)

<https://arxiv.org/pdf/1905.05055.pdf>



# SSD Detection 성능 및 수행 시간 비교

## 1.4 SSD (Single Shot MultiBox Detector)

- 2016년 11월 논문 출시(<https://arxiv.org/pdf/1512.02325.pdf>)

### SSD 성능 비교

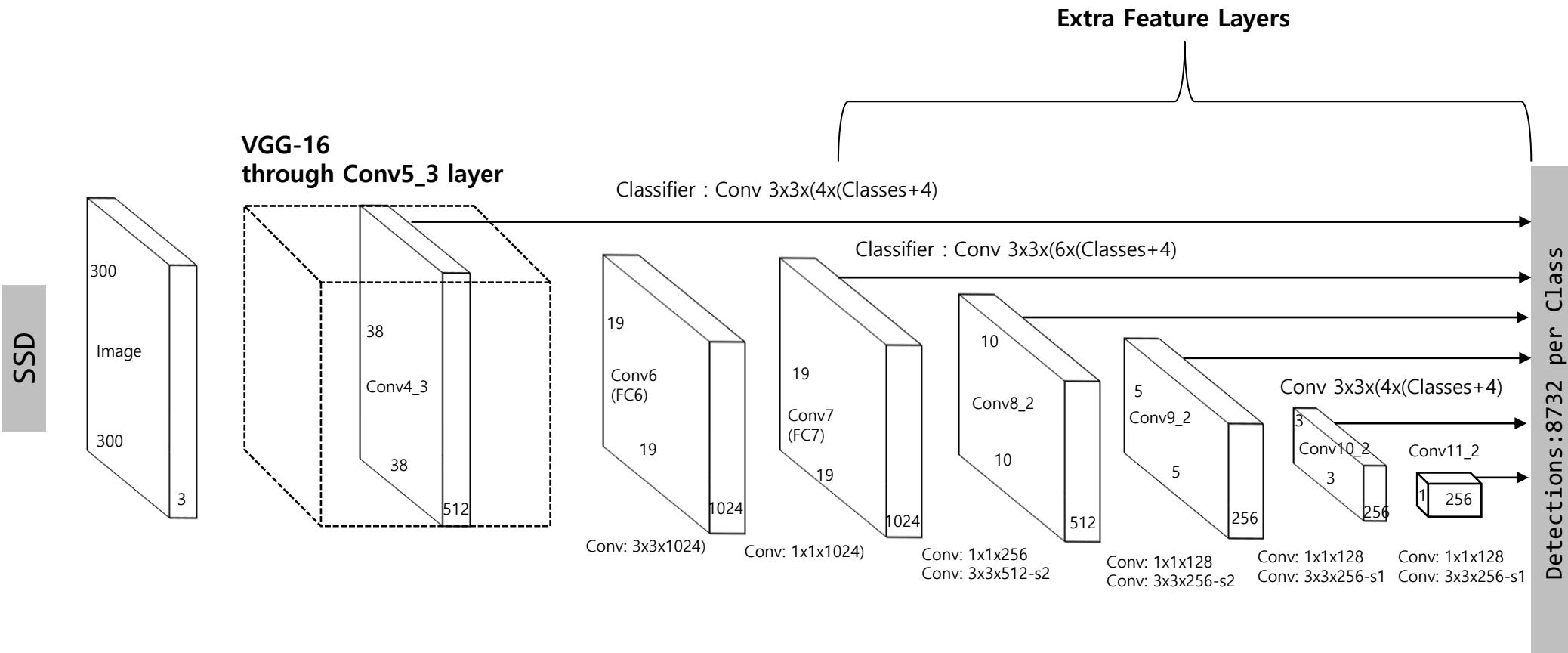
Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster [2]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
Faster [2]	07++12+COCO	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2
YOLO [5]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD300	07++12+COCO	77.5	90.2	83.3	76.3	63.0	53.6	83.8	82.8	92.0	59.7	82.7	63.5	89.3	87.6	85.9	84.3	52.6	82.5	74.1	88.4	74.2
SSD512	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
SSD512	07++12+COCO	80.0	90.7	86.8	80.5	67.8	60.8	86.3	85.5	93.5	63.2	85.7	64.4	90.9	89.0	88.9	86.8	57.2	85.1	72.8	88.4	75.9

### SSD 수행 시간 비교

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

# SSD Network 구조

## 1.4 SSD (Single Shot MultiBox Detector)



# SSD 주요 구성 요소

## 1.4 SSD (Single Shot MultiBox Detector)

Multi Scale Feature Layer

Default(Anchor) Box

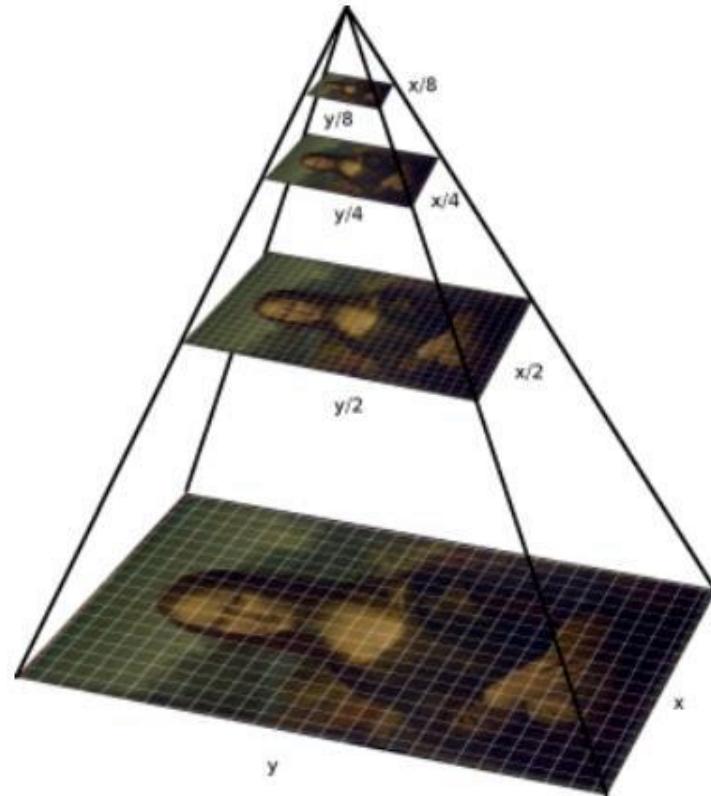
# 이미지 Scale 조정에 따른 여러 크기의 Object Detection

1.4 SSD (Single Shot MultiBox Detector)



# 이미지 Scale 조정에 따른 여러 크기의 Object Detection

## 1.4 SSD (Single Shot MultiBox Detector)

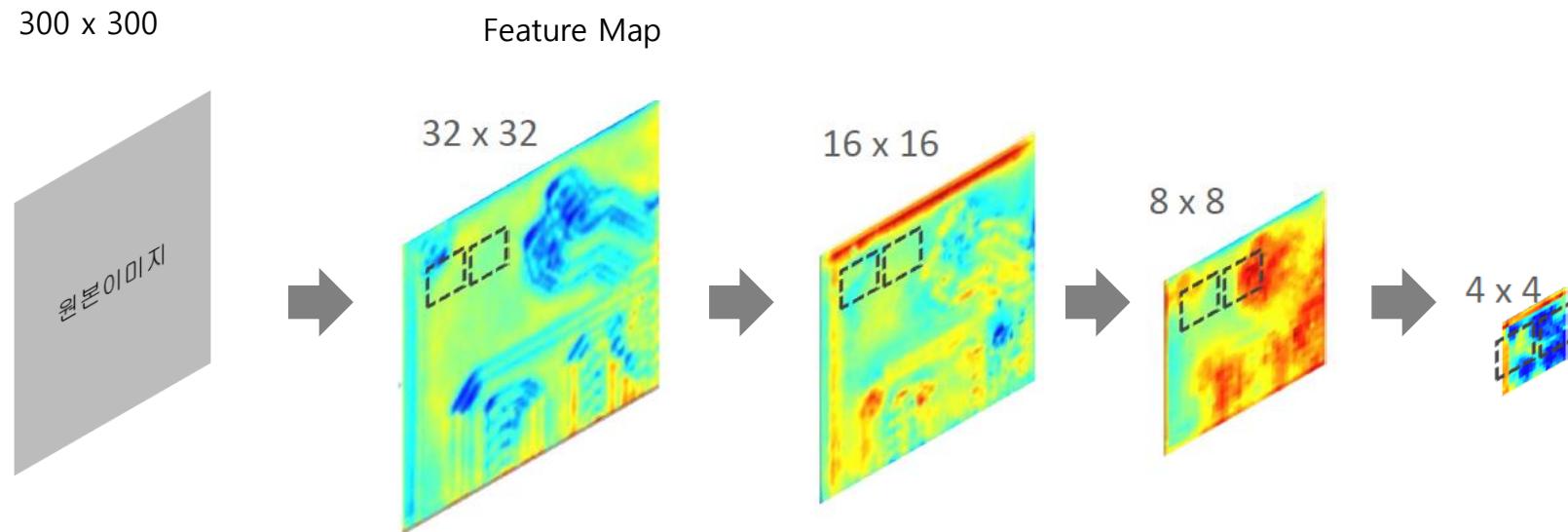


이미지 피라미드

# 다른 크기의 Feature Map을 이용한 Object Detection

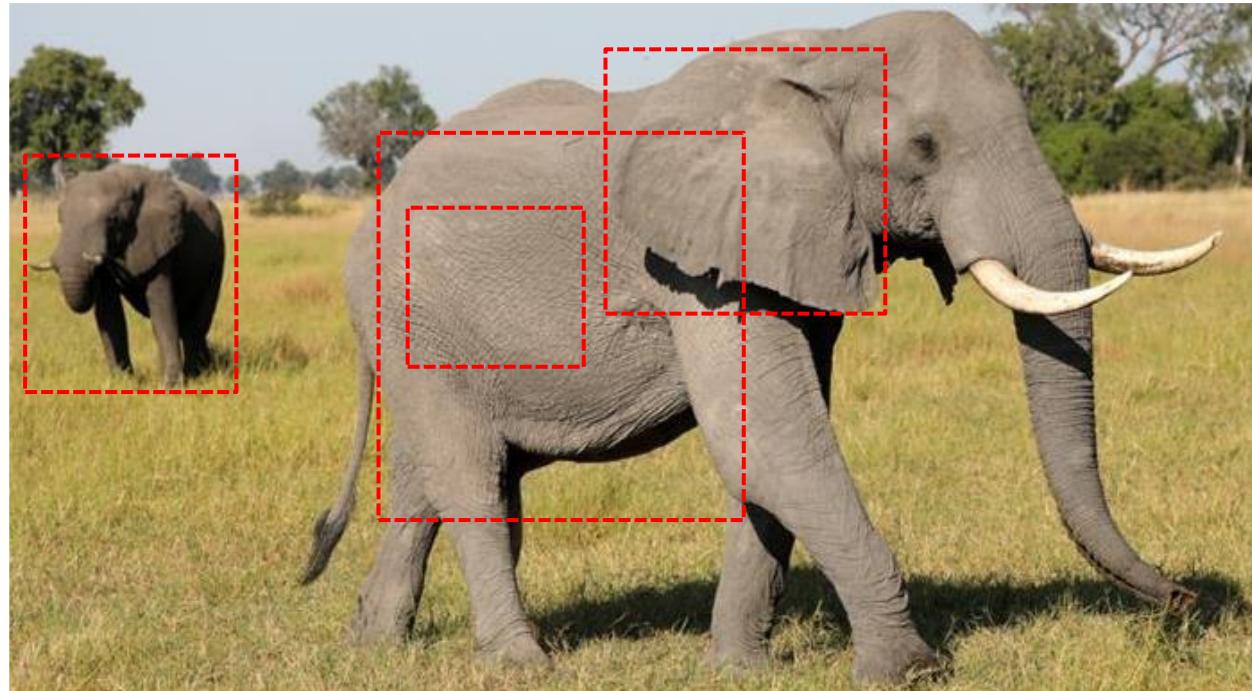
1.4 SSD (Single Shot MultiBox Detector)

서로 다른 크기의 Feature Map들을 기반으로 Object Detection 수행



# Feature Map 기반의 Multi-Scale Feature Layer

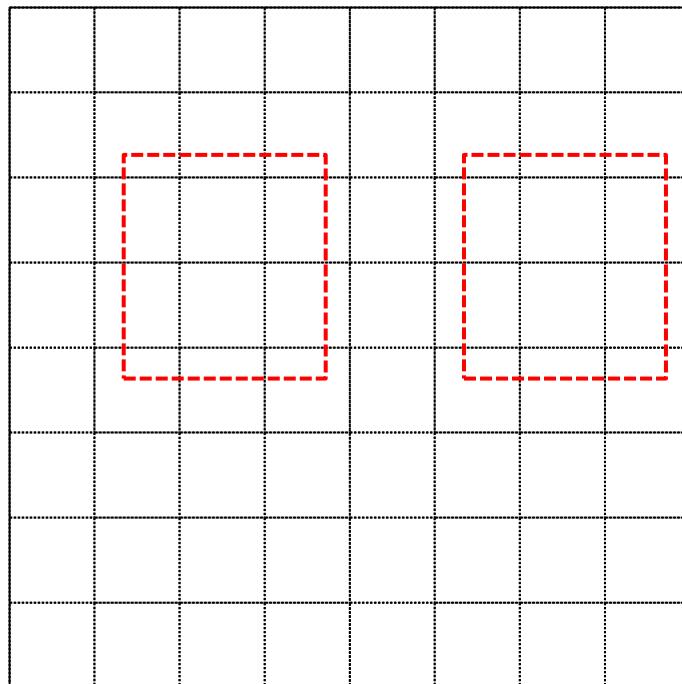
1.4 SSD (Single Shot MultiBox Detector)



# Feature Map 기반의 Multi-Scale Feature Layer

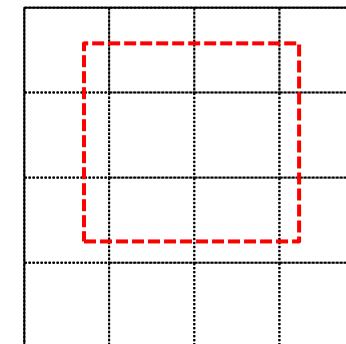
1.4 SSD (Single Shot MultiBox Detector)

**8 x 8 Feature Map**



Feature Map의 크기가 작을수록 더 큰 이미지를 찾을 수 있음

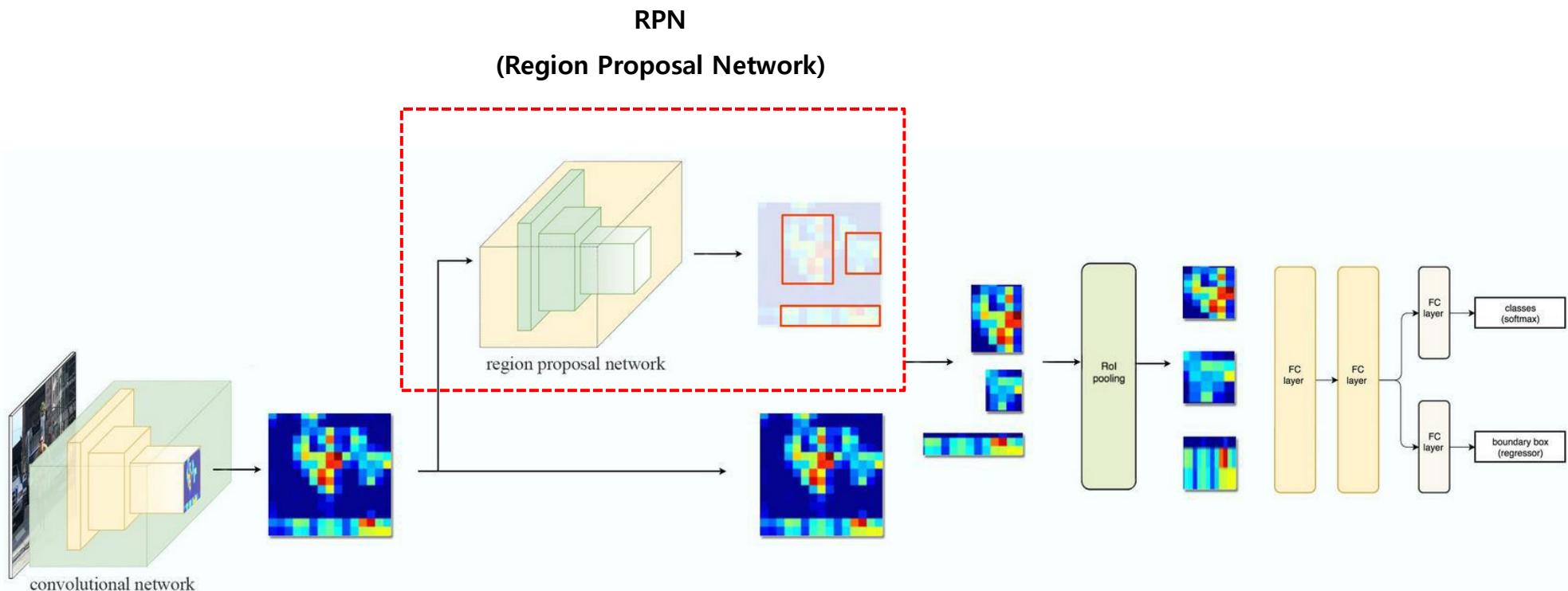
**4 x 4 Feature Map**



# Anchor Box 기반의 Object Detection 모델

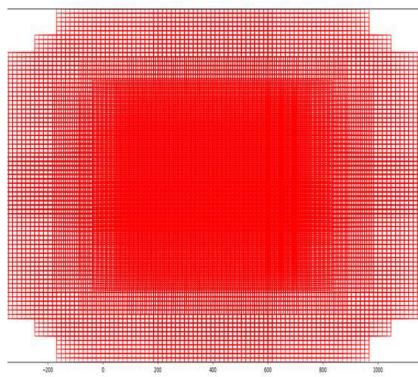
- Faster RCNN

1.4 SSD (Single Shot MultiBox Detector)

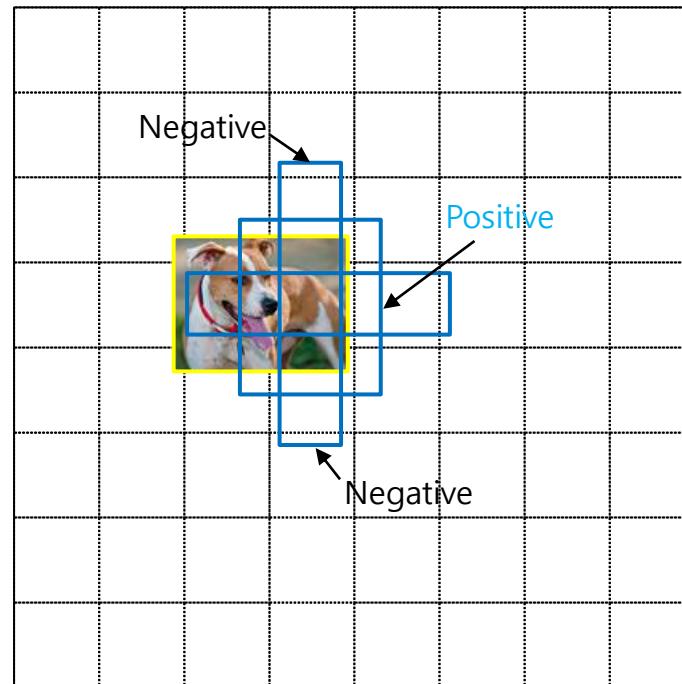


# RPN에서의 Anchor Box의 활용

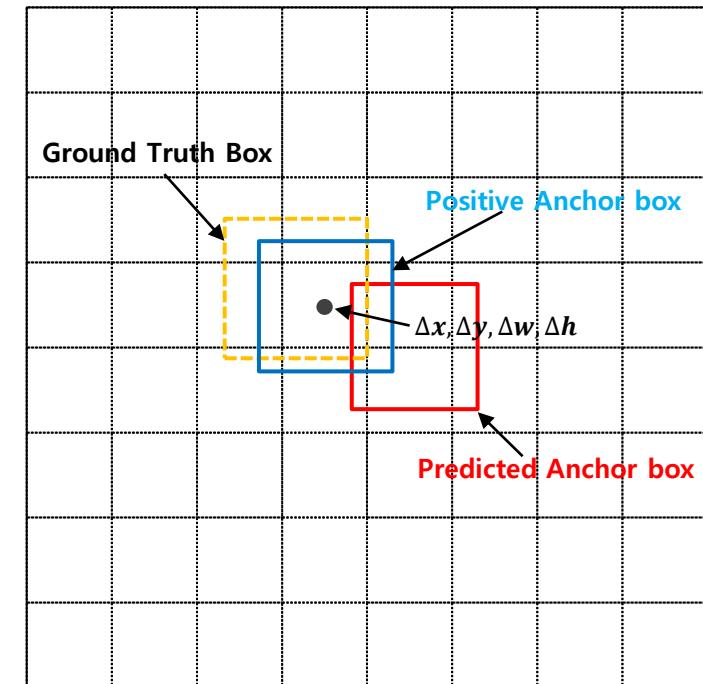
## 1.4 SSD (Single Shot MultiBox Detector)



### Classification



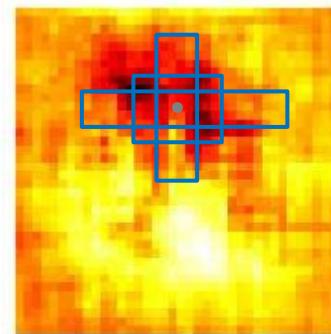
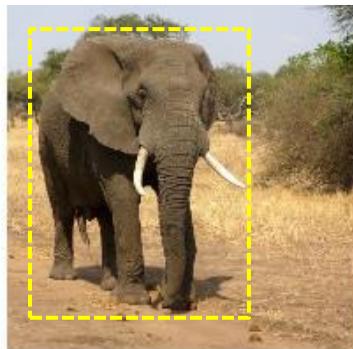
### Bounding Box Regression



# Anchor box를 활용한 Object Detection

## 1.4 SSD (Single Shot MultiBox Detector)

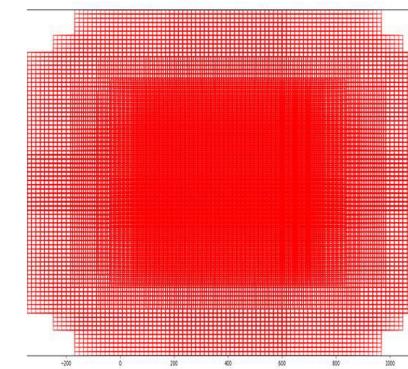
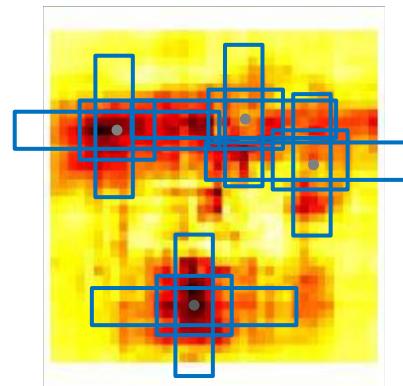
코끼리



개별 anchor박스가 다음 정보를 가질 수 있도록 학습

- Anchor box와 겹치는 Feature 맵 영역의 Object클래스 분류
- GT Box위치를 예측할 수 있도록 수정된 좌표

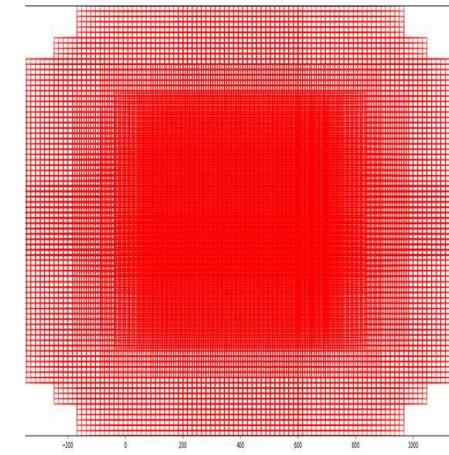
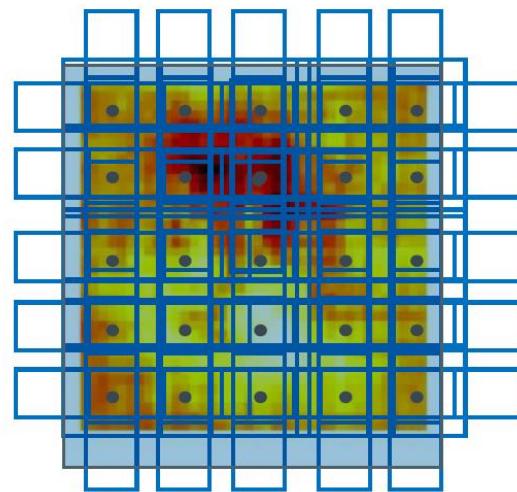
고-카트



# Anchor box를 활용한 Object Detection

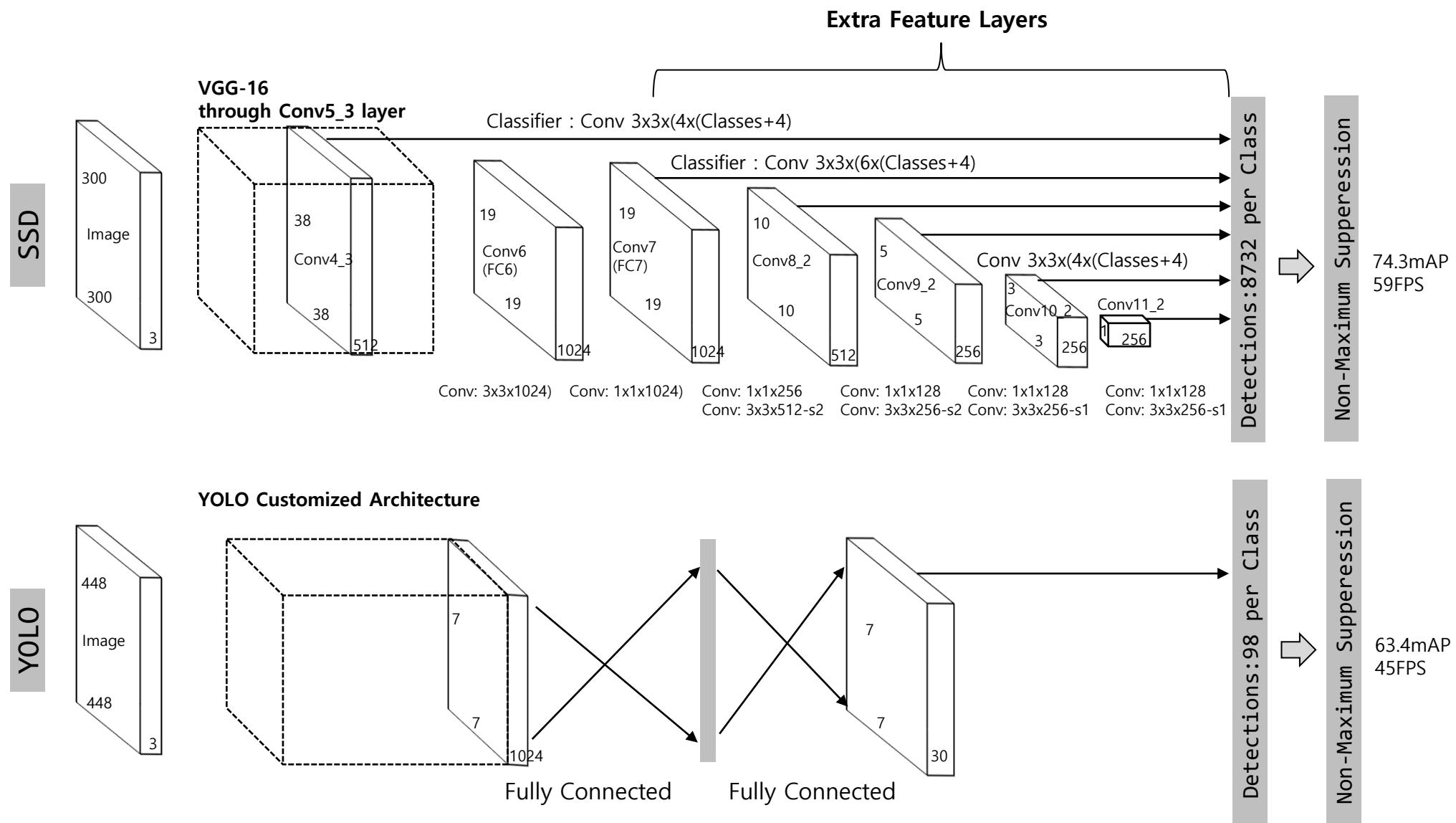
## 1.4 SSD (Single Shot MultiBox Detector)

개별 Anchor box별로 Detection하려는 Object  
유형 Softmax값, 수정 좌표값



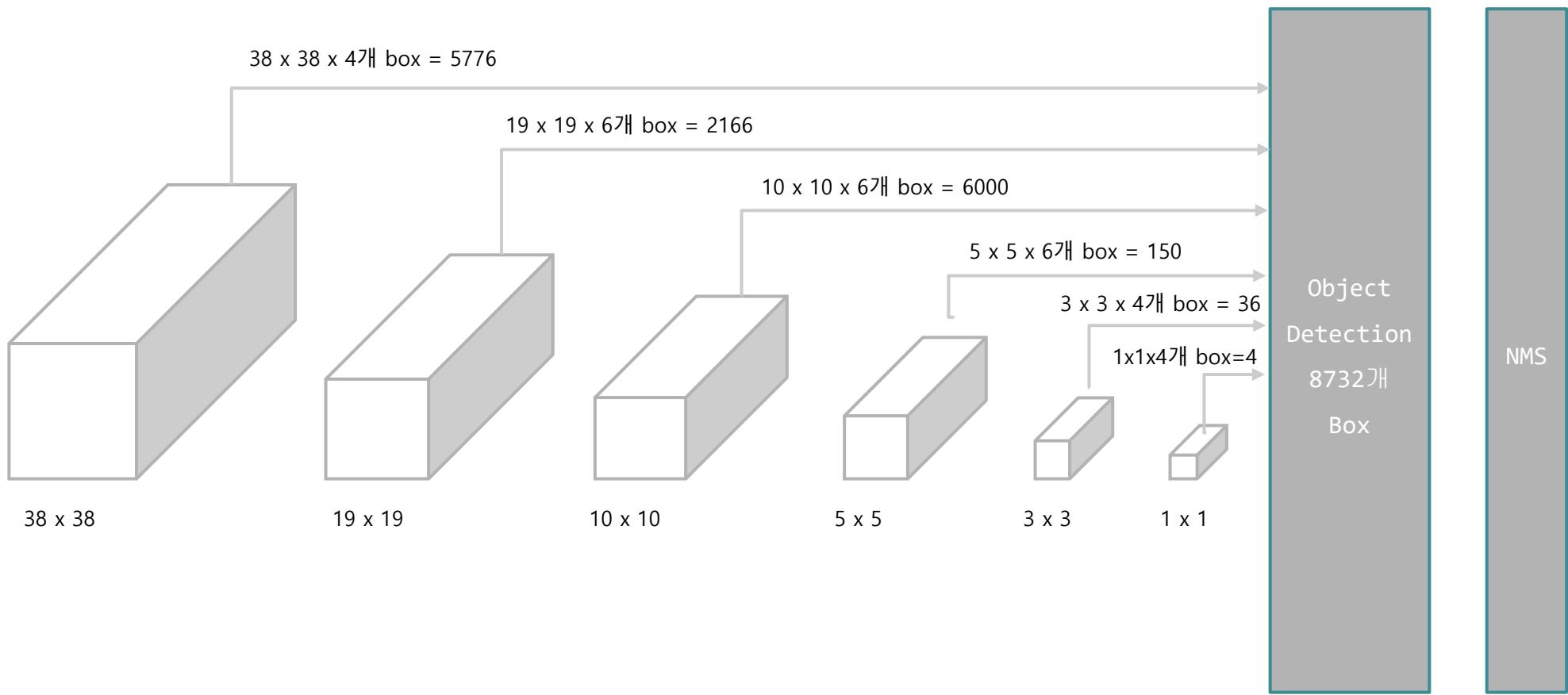
# SSD Network 구성

## 1.4 SSD (Single Shot MultiBox Detector)



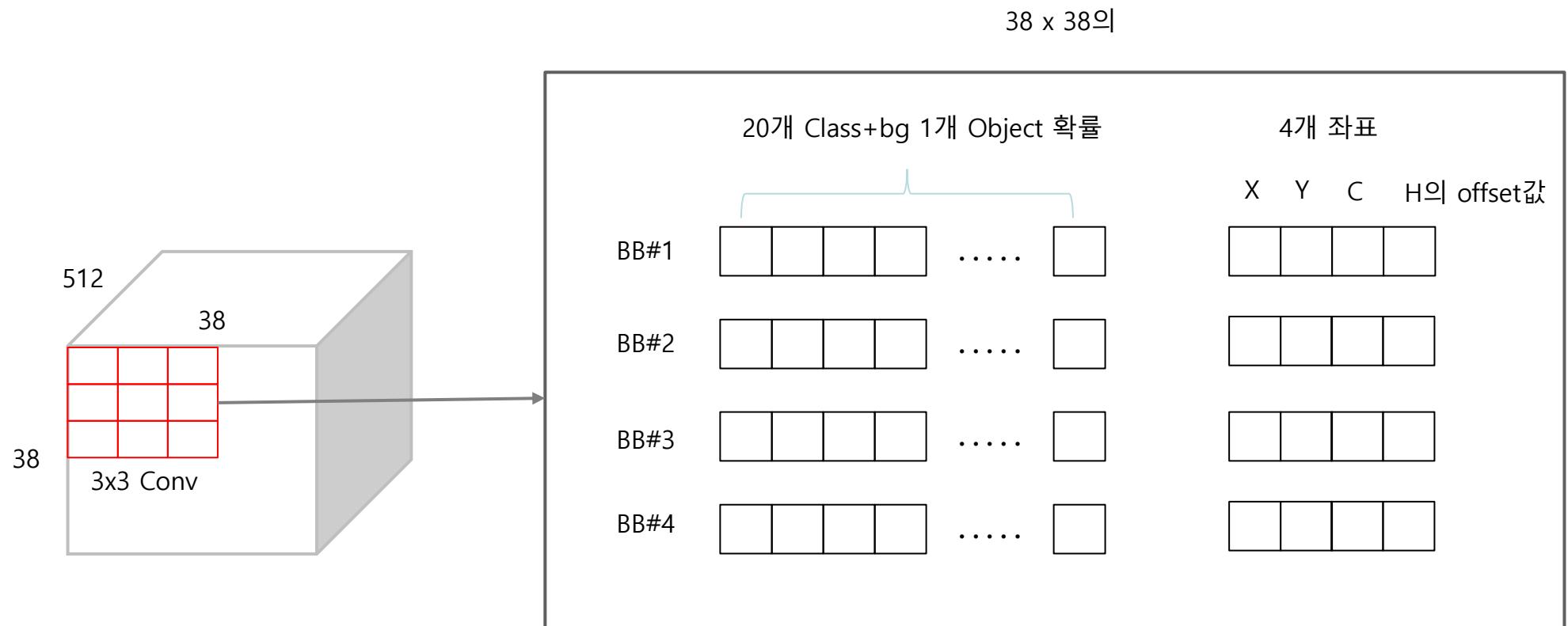
# SSD Network 구성

## 1.4 SSD (Single Shot MultiBox Detector)



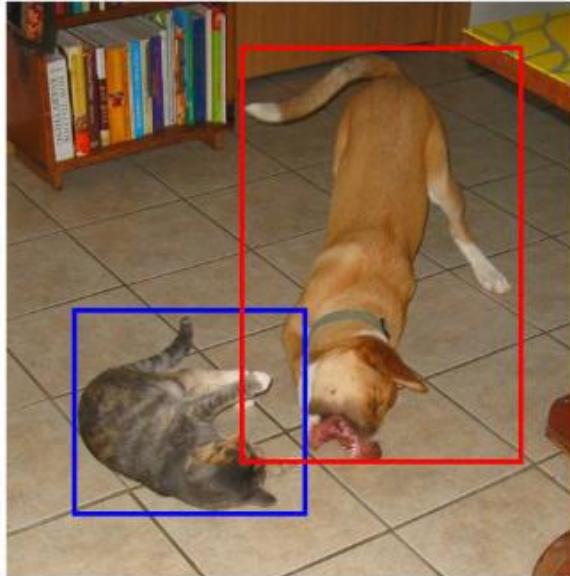
# Anchor 박스를 활용한 Convolution Predictors for detection

## 1.4 SSD (Single Shot MultiBox Detector)

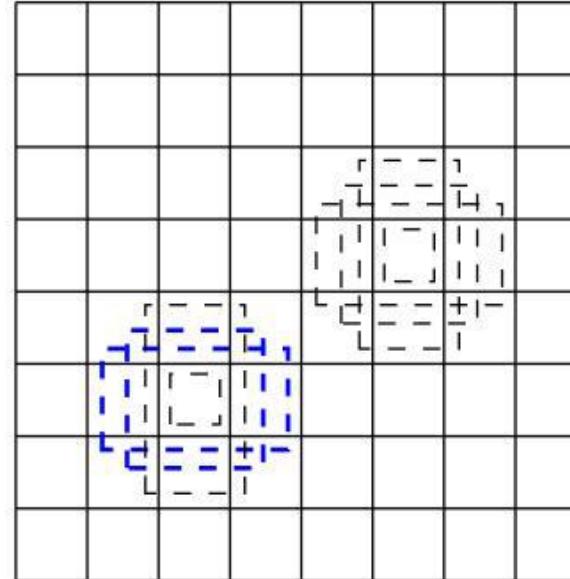


# SSD의 Multi Scale Feature Map과 Anchor Box 적용

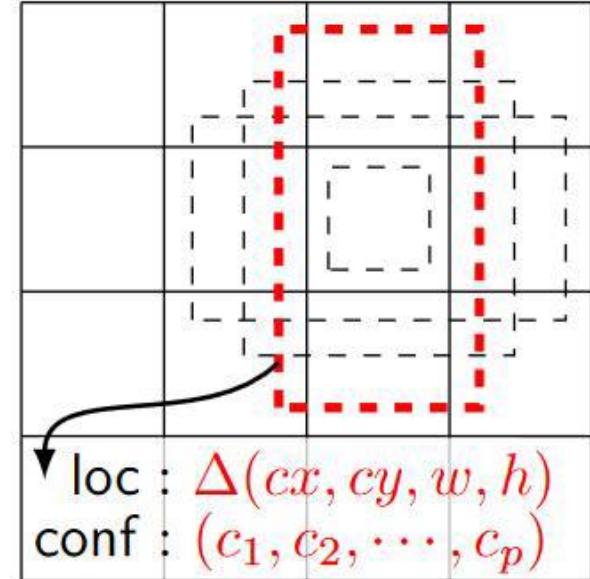
## 1.4 SSD (Single Shot MultiBox Detector)



(a) Image with GT boxes



(b)  $8 \times 8$  feature map



(c)  $4 \times 4$  feature map

### Matching 전략

Bounding box와 겹치는 IOU가 0.5이상인 Default(Anchor) Box들의 Classification과 Bounding box Regression을 최적화 학습 수행

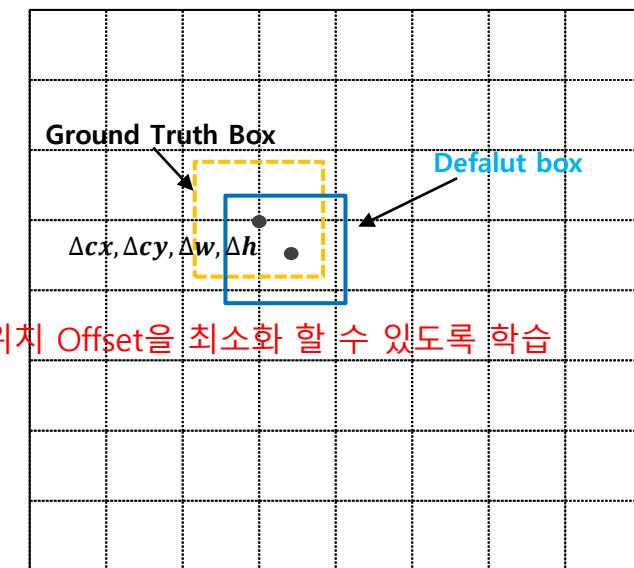
Loss 함수

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

N은 Matched된 Default box개수    Classification    Bounding Box regression

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

### Bounding Box Regression



# Design choice 별 Performance

1.4 SSD (Single Shot MultiBox Detector)

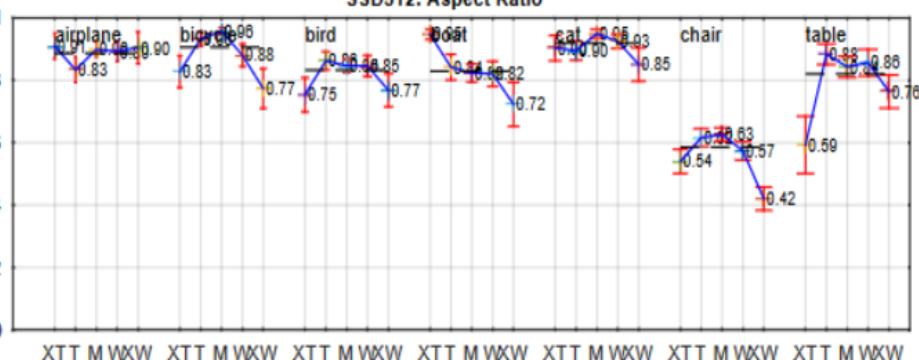
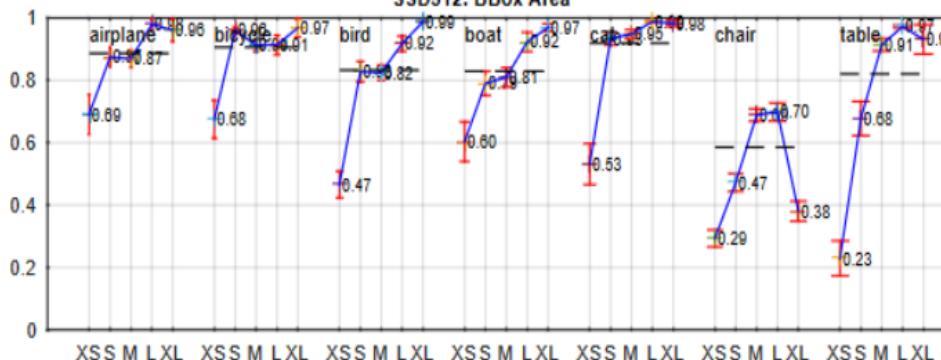
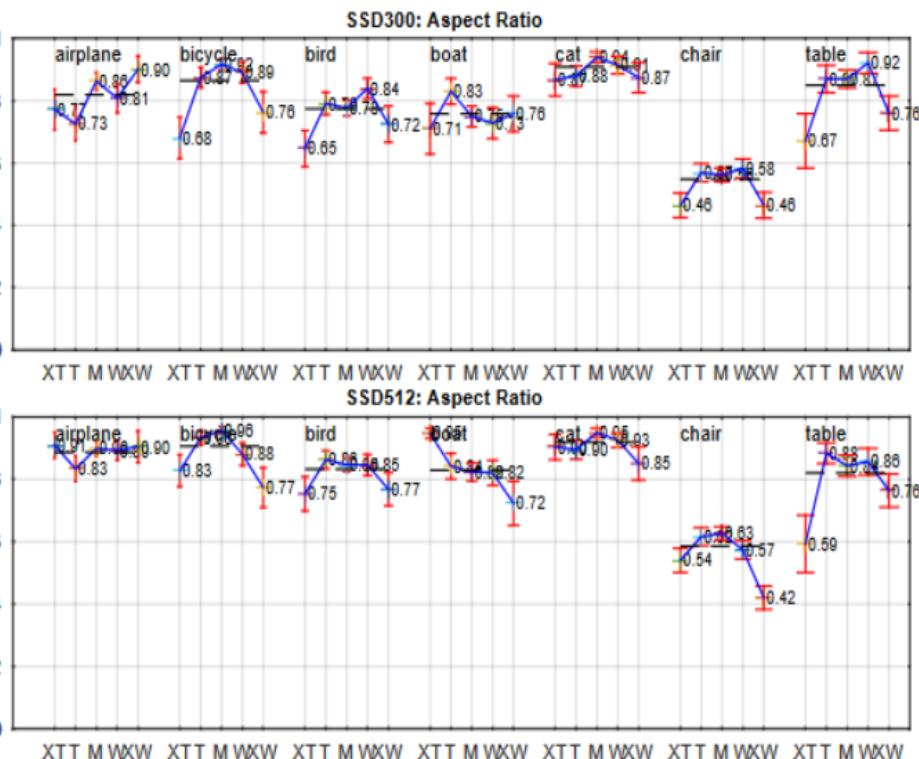
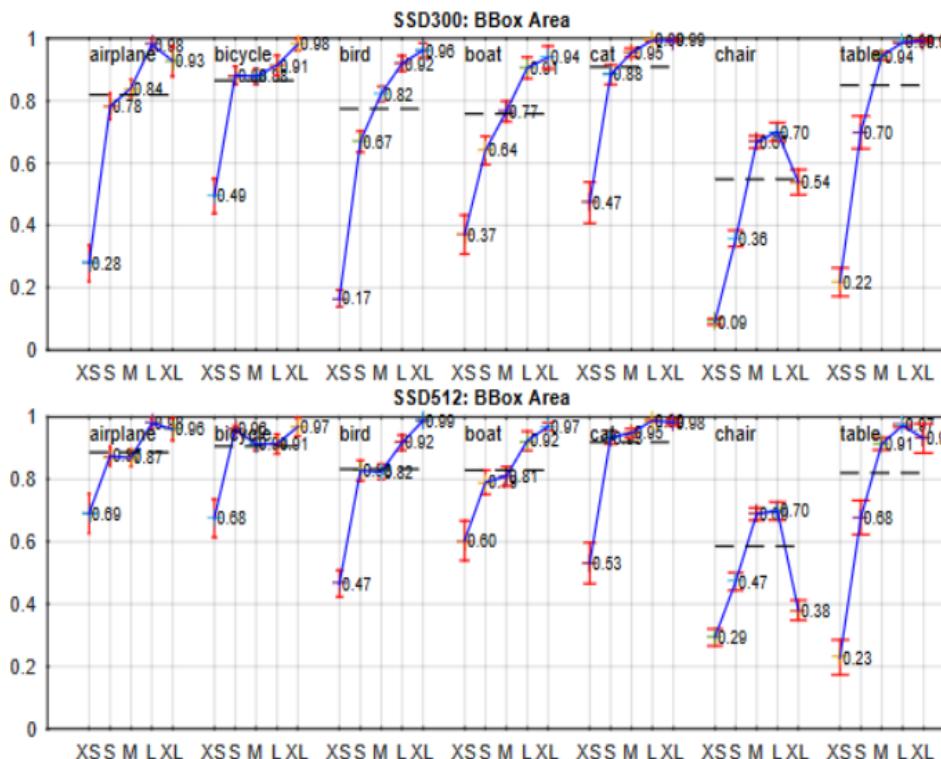
	SSD3000				
more data augmentation?		✓	✓	✓	✓
include $\{\frac{1}{2}, 2\}$ box?	✓		✓	✓	✓
include $\{\frac{1}{3}, 3\}$ box?	✓			✓	✓
use atrous?	✓	✓	✓		✓
VOC2007 test mAP	65.5	71.6	73.7	74.2	74.3

## Data Augmentation

- GT Object와 IOU가 0.1, 0.3, 0.5, 0.7, 0.9 가 될 수 있도록 특정 Object들의 Image를 잘라냄
- 잘라낸 이미지를 random하게 sampling
- 잘라낸 sample 이미지를 0.1~1 사이로, aspect ratio는 1/2~2 사이로 크기를 맞춤
- 개별 sample 이미지를 다시 300x300으로 고정. 그리고 그 중 50%는 horizontal flip

# 작은 Object 의 Detection 성능

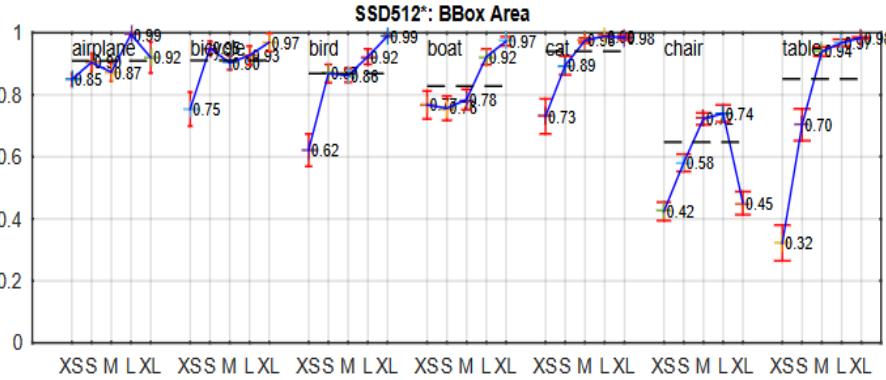
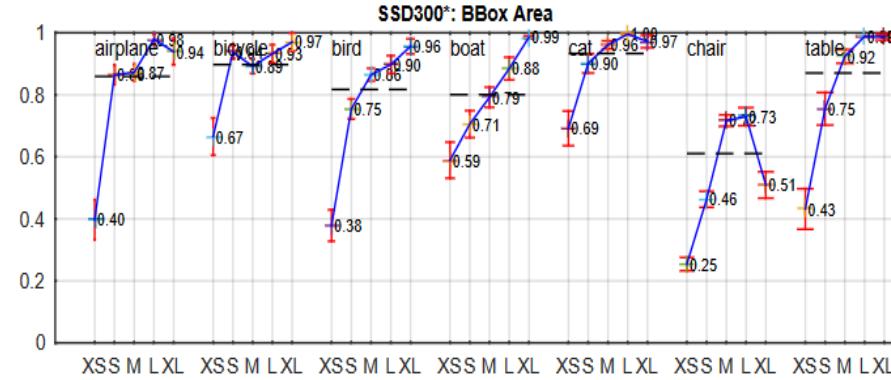
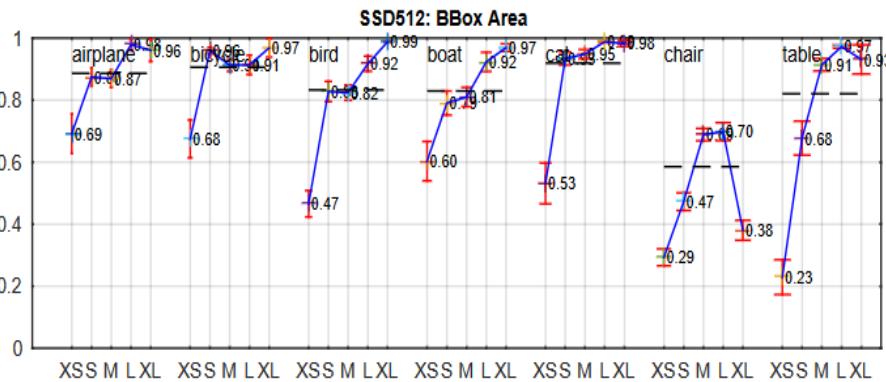
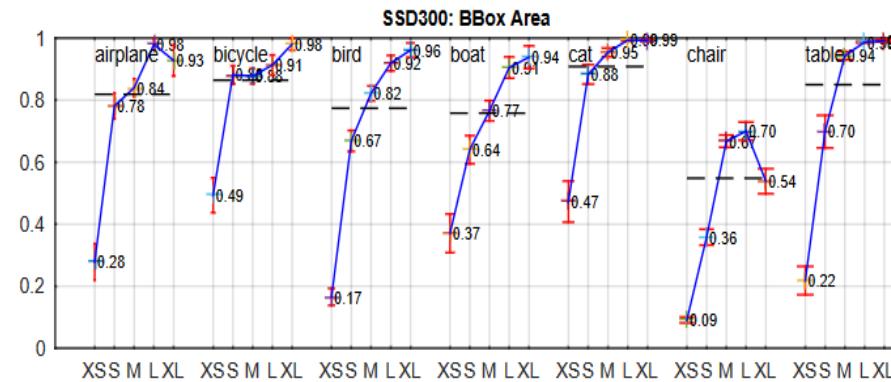
## 1.4 SSD (Single Shot MultiBox Detector)



# Data Augmentation 후

## 작은 Object의 Detection 성능 향상

1.4 SSD (Single Shot MultiBox Detector)



# SSD Detection 성능 및 수행 시간 비교

## 1.4 SSD (Single Shot MultiBox Detector)

- 2016년 11월 논문 출시(<https://arxiv.org/pdf/1512.02325.pdf>)

### SSD 성능 비교

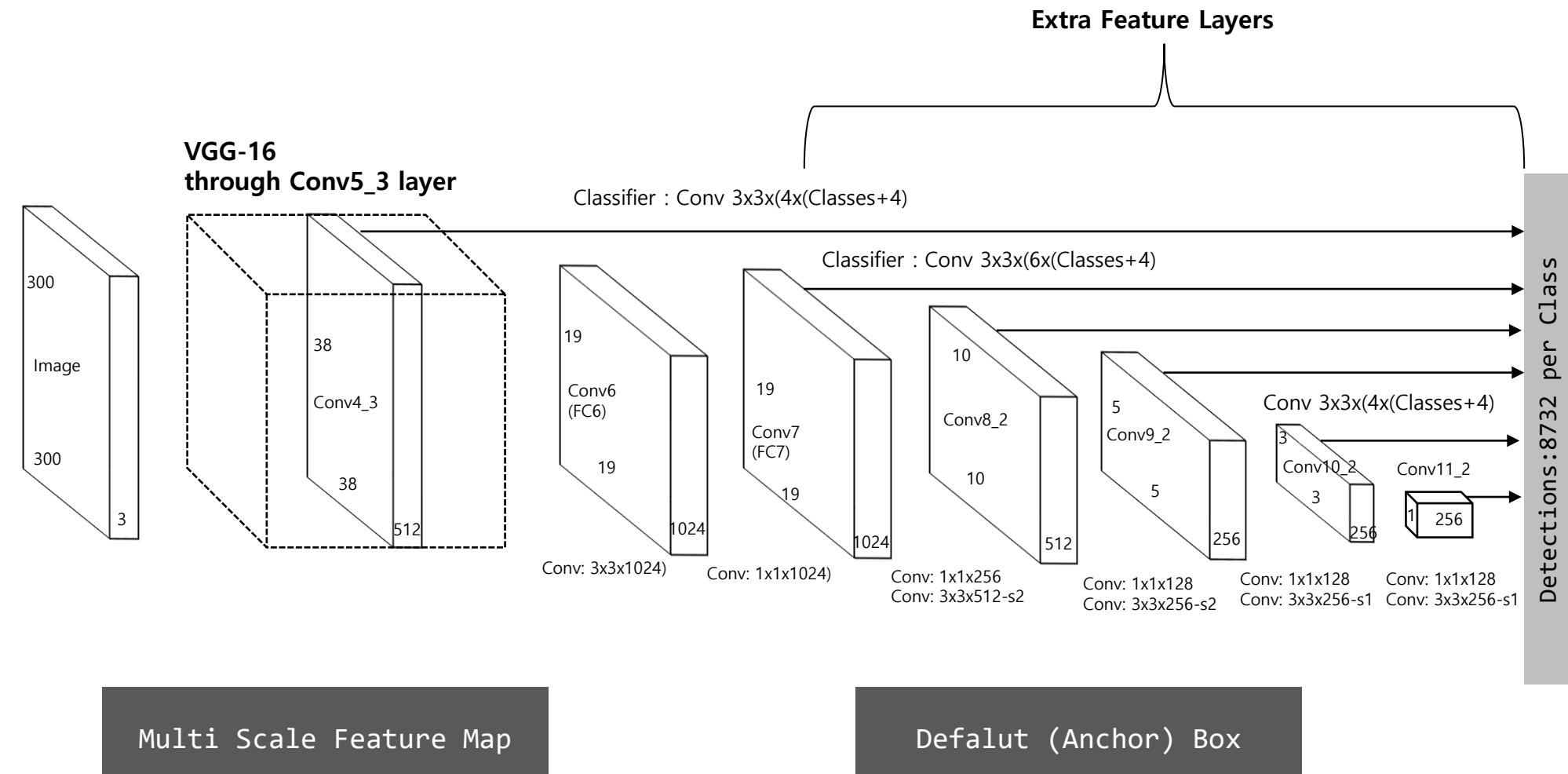
Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster [2]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
Faster [2]	07++12+COCO	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2
YOLO [5]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD300	07++12+COCO	77.5	90.2	83.3	76.3	63.0	53.6	83.8	82.8	92.0	59.7	82.7	63.5	89.3	87.6	85.9	84.3	52.6	82.5	74.1	88.4	74.2
SSD512	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
SSD512	07++12+COCO	80.0	90.7	86.8	80.5	67.8	60.8	86.3	85.5	93.5	63.2	85.7	64.4	90.9	89.0	88.9	86.8	57.2	85.1	72.8	88.4	75.9

### SSD 수행 시간 비교

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

# Summary

## 1.4 SSD (Single Shot MultiBox Detector)



# 1. 오브젝트 디텍션

---

- 1.1 오브젝트 디텍션 개요
  - 1.2 OpenCV 개요
  - 1.3 R-CNN, Fast R-CNN, Faster R-CNN
  - 1.4 SSD (Single Shot MultiBox Detector)
  - 1.5 YOLO (You Only Look Once) 시리즈**
  - 1.6 RetinaNet
-



# One Stage Detector

1.5 YOLO (You Only Look Once) 시리즈

2015. 06

Yolo v1

2015. 12

SSD

2016. 12

Yolo v2

2017. 08

Retinanet

2018. 04

Yolo v3

2019. 11

SSD

2020. 04

Yolo v4

Feature  
Pyramid  
Network

Yolo V1 논문 저자  
Joseph Redmon , Santosh Divvala , Ross Girshick , Ali Farhadi



Darknet  
C로 작성된 Deep Learning Framework



# YOLO Version

## 1.5 YOLO (You Only Look Once) 시리즈

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

V1

빠른 Detection 시간  
그러나 낮은 정확도

V2

수행 시간과 성능 모두 개선  
SSD에 비해 작은 Object 성능 저하

V3

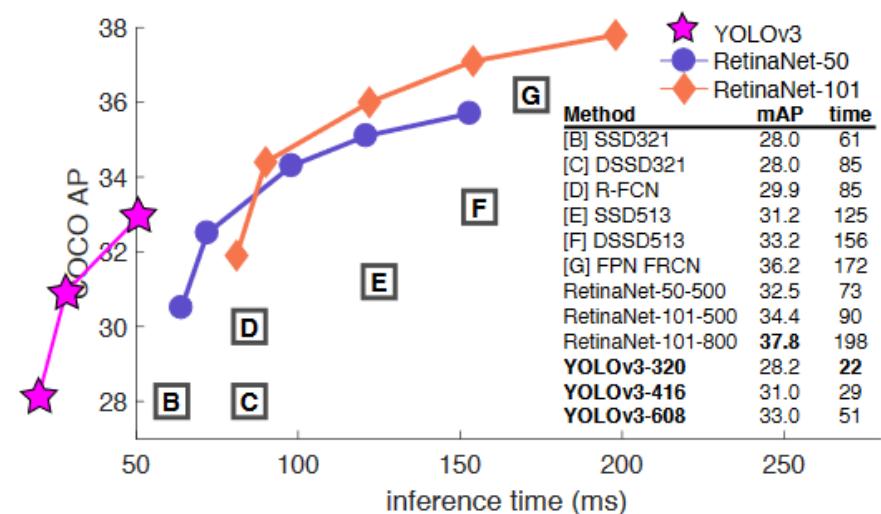
V2대비 수행 시간은 조금 느림  
성능 대폭 개선

V4

V3대비 수행시간 약간 향상  
성능 대폭 개선

## Abstract

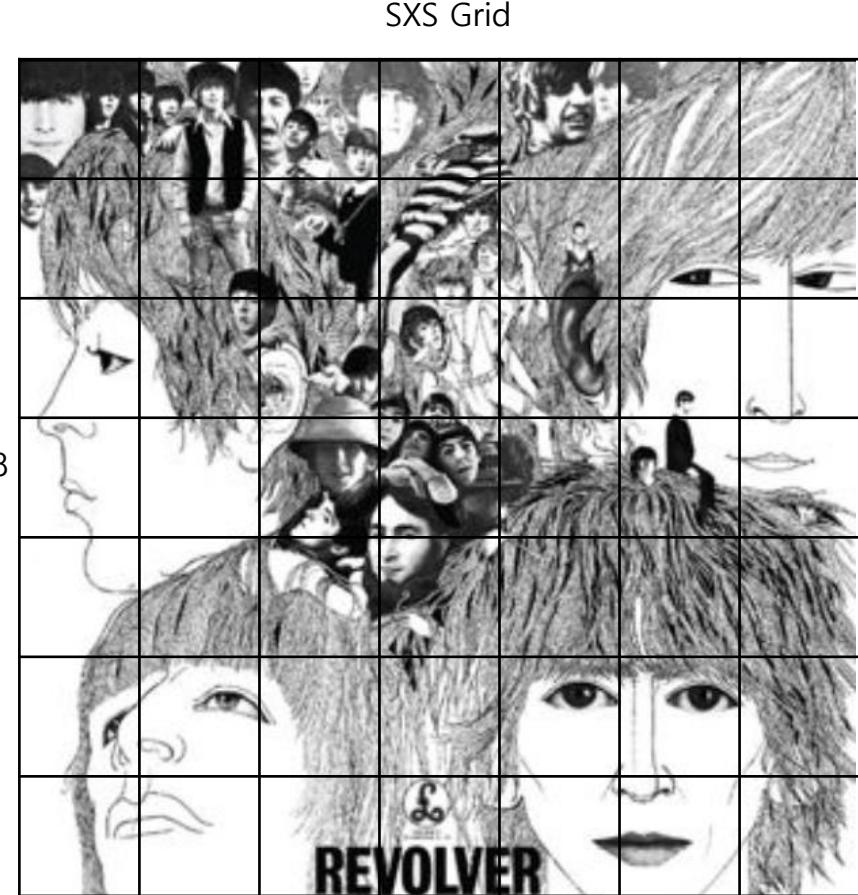
We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At  $320 \times 320$  YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP<sub>50</sub> in 51 ms on a Titan X, compared to 57.5 AP<sub>50</sub> in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at <https://pjreddie.com/yolo/>.

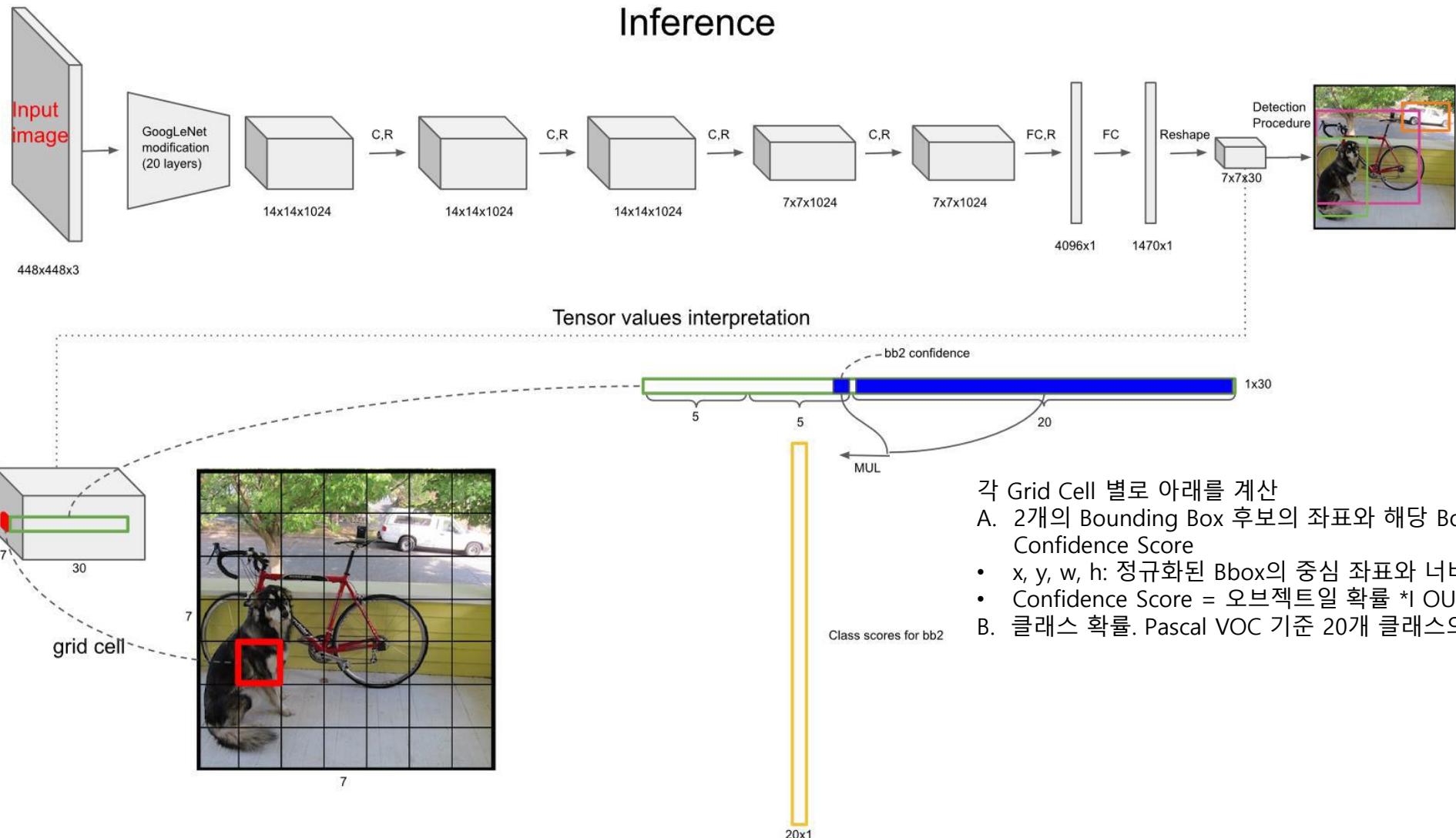


## Speed + Accuracy

- Yolo v1은 입력 이미지를 SXS Grid로 나누고 각 Grid의 Cell이 하나의 Object에 대한 Detection 수행
- 각 Grid Cell이 2개의 Bounding Box 후보를 기반으로 Object의 Bounding Box를 예측

448x448  
Image





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

BBox 중심 x, y좌표Loss

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} (x - \hat{x}_i)^2 + (y - \hat{y}_i)^2$$

예측 좌표  $x, y$  값과 Ground Truth  $x, y$  값의 오차 제곱을 기반  
 모든 Cell의 2개의 Bbox(89개 Bbox)중에 예측 Bbox를 책임지는 Bbox만 Loss 계산  
 $\mathbb{I}_{ij}^{obj}$ 는 98개의 Bbox 중 오브젝트 예측을 책임지는 Bbox만 1, 나머지는 0

BBox 너비 w, 높이 h Loss

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

예측 너비, 높이 값과 Ground Truth 너비, 높이 값의 오차 제곱을 기반으로 하되,  
 크기가 큰 오브젝트의 경우 오류가 상대적으로 커짐을 제약하기 위해서 제곱근을 취함

## Object Confidence Loss

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

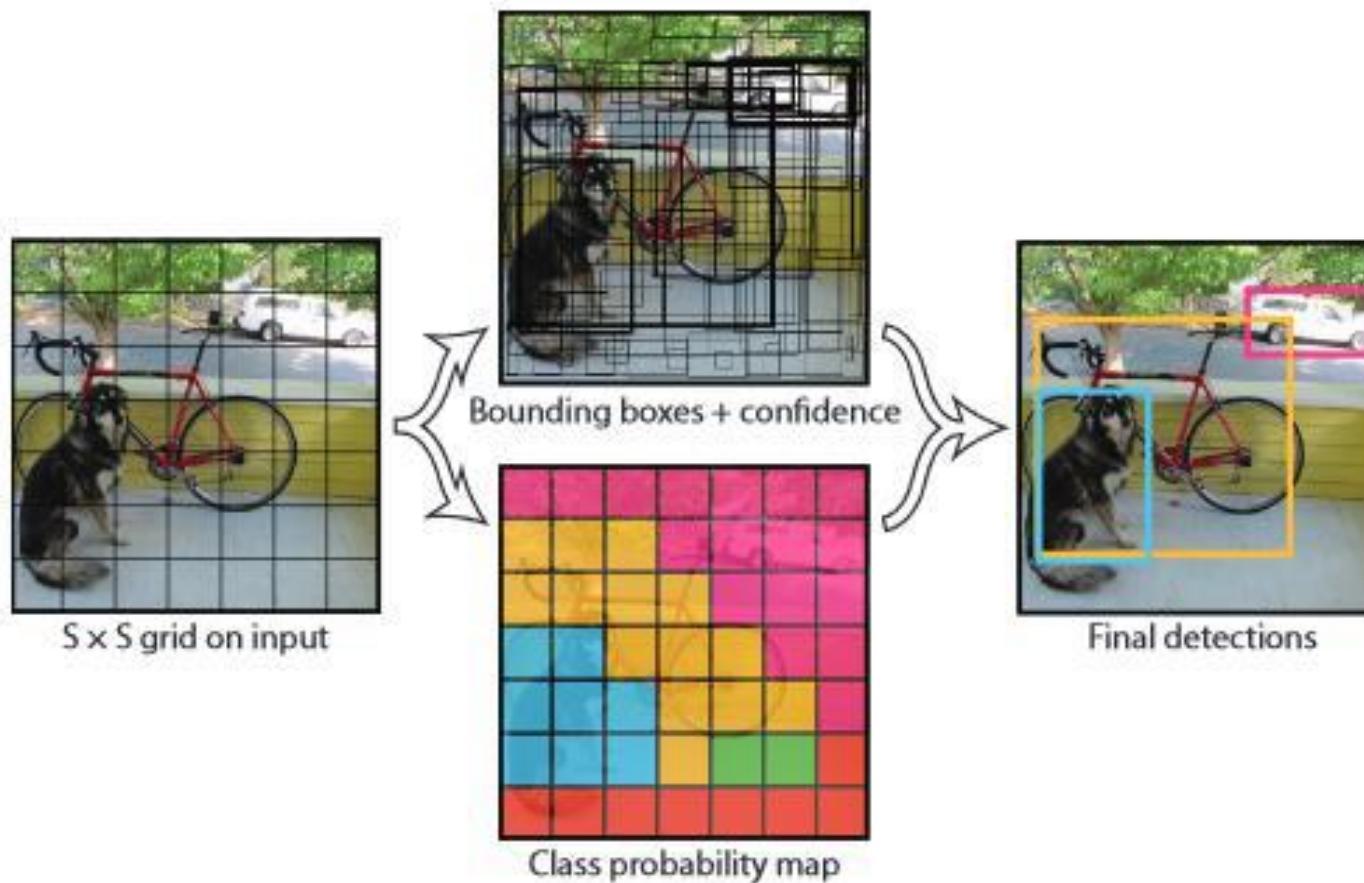
$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

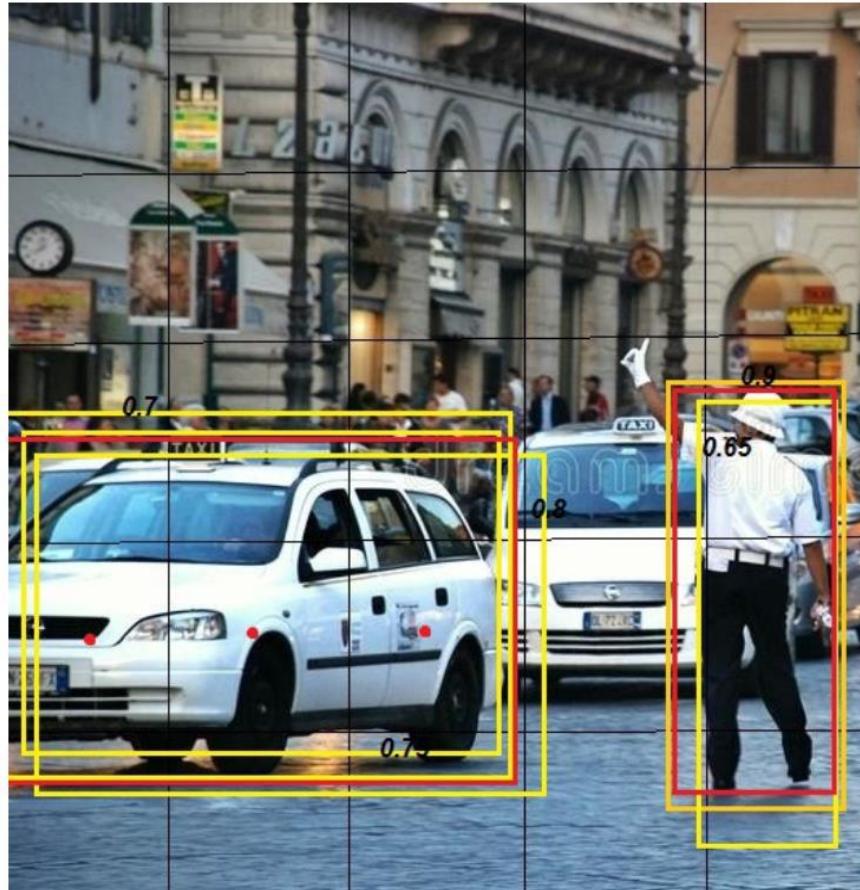
예측된 Object Confidence Score 와 Ground Truth 의 IOU 의 예측 오차를 기반  
Object를 책임지는 Bbox confidence loss + Object 가 없어야 하는 Bbox 의  
confidence loss

## Classification Loss

$$\sum_{i=0}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

예측classification 확률 오차의 제곱, Object 를 책임지는 Bbox 만 대상





### 개별 Class 별 NMS 수행

1. 특정 Confidence 값 이하는 모두 제거
2. 가장 높은 Confidence 값을 가진 순으로 Bbox 정렬
3. 가장 높은 Confidence 를 가진 Bbox 와 IOU 와 겹치는 부분이 IOU Threshold 보다 큰 Bbox 는 모두 제거
4. 남아 있는 Bbox 에 대해 3 번 Step 을 반복

### Object Confidence와 IOU Threshold로 Filtering 조절

Detection시간은 빠르나 Detection 성능이 떨어짐, 특히 작은 Object에 대한 성능이 나쁨

# YOLO-V1, V2, V3 비교

## 1.5 YOLO (You Only Look Once) 시리즈

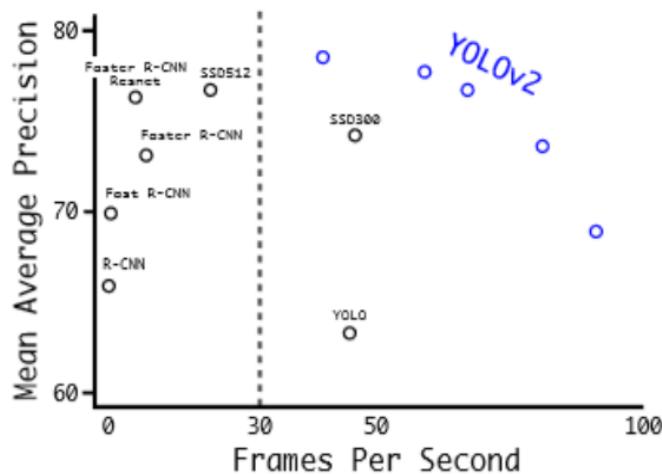
anchor box 기반의 모델과 더 뛰어난 Backbone 구성, 다양한 성능 향상 테크닉을 적용하면서 발전됨.

항목	V1	V2	V3
원본 이미지 크기	446X446	416X416	416X416
Feature Extractor	Inception 변형	Darknet 19	Darknet 53
Grid당 Anchor Box 수	2개 (anchor box 는 아님)	5개	Output Feature Map 당 3 개 서로 다른 크기와 스케일로 총 9 개
Anchor box 결정 방법		K-Means Clustering	K-Means Clustering
Output Feature Map 크기 (Depth 제외)	7X7	13X13	13X13, 26X26, 52X52 3개의 Feature Map 사용
Feature Map Scaling 기법			FPN(Feature Pyramid Network)

# YOLO-V2 Detection 시간 및 성능

1.5 YOLO (You Only Look Once) 시리즈

PASCAL VOC 2007 Detection 시간



MS-COCO 기준 Detection 성능

		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k	<b>26.8</b>	<b>46.5</b>	<b>27.8</b>	<b>9.0</b>	<b>28.9</b>	<b>41.9</b>	<b>24.8</b>	<b>37.5</b>	<b>39.8</b>	<b>14.0</b>	<b>43.5</b>	<b>59.0</b>
YOLOv2 [11]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

- Batch Normalization
- High Resolution Classifier : 네트워크의 Classifier 단을 보다 높은 resolution(448x448)로 fine tuning
- 13 x 13 feature map 기반에서 개별 Grid cell 별 5 개의 Anchor box에서 Object Detection
  - anchor box의 크기와 ratio는 K Means Clustering으로 설정
- 예측 Bbox의 x, y 좌표가 중심 Cell 내에서 벗어나지 않도록 Direct Location Prediction 적용
- Darknet 19 Classification 모델 채택
- Classification layer를 fully Connected layer에서 Fully Convolution으로 변경하고 서로 다른 크기의 image들로 네트워크 학습

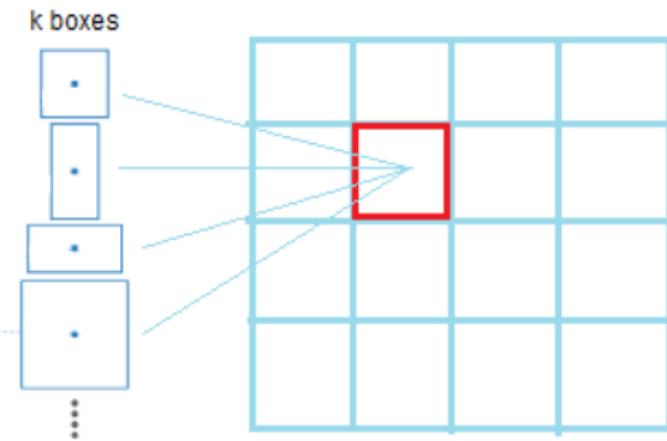
# Yolo v2 Anchor Box로 1 Cell에서 여러 개 Object Detection

1.5 YOLO (You Only Look Once) 시리즈

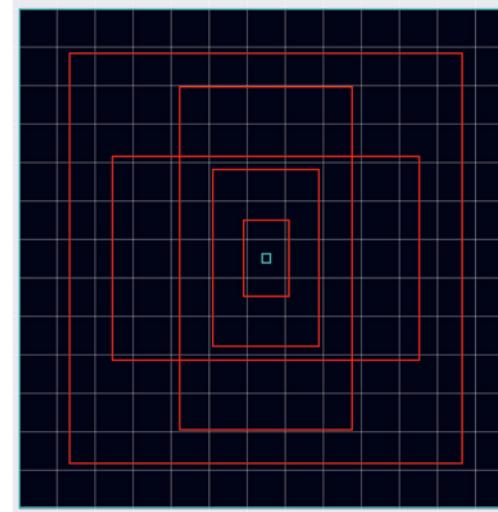
SSD와 마찬가지로 1 개의 Cell에서 여러 개의 Anchor 를 통해 개별 Cell에서 여러 개 Object Detection 가능

K-Means Clustering 을 통해 데이터 세트의 이미지 크기와 Shape Ratio 따른 5 개의 군집화 분류를 하여 Anchor Box 를 계산

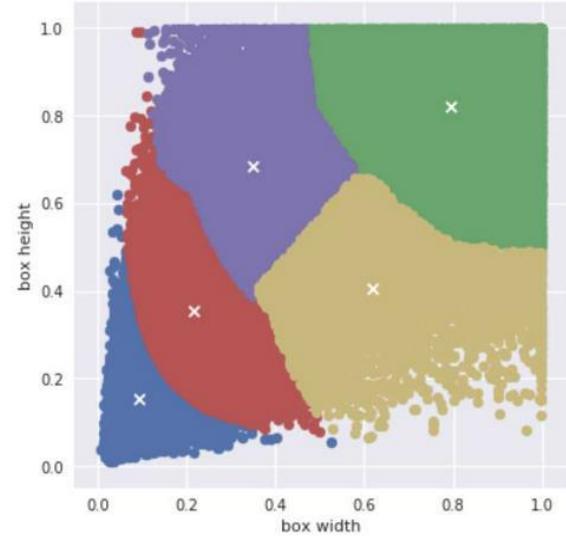
Convolutional With Anchor Boxes



5개 Anchor Box

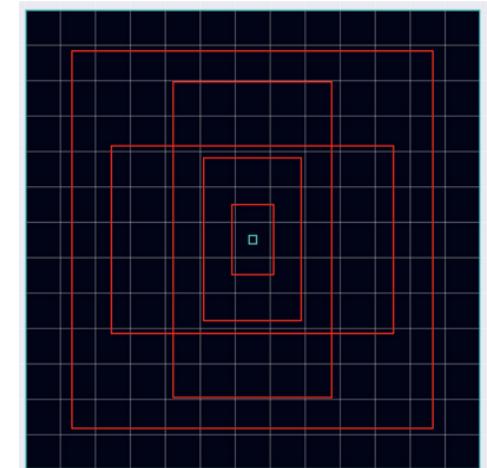
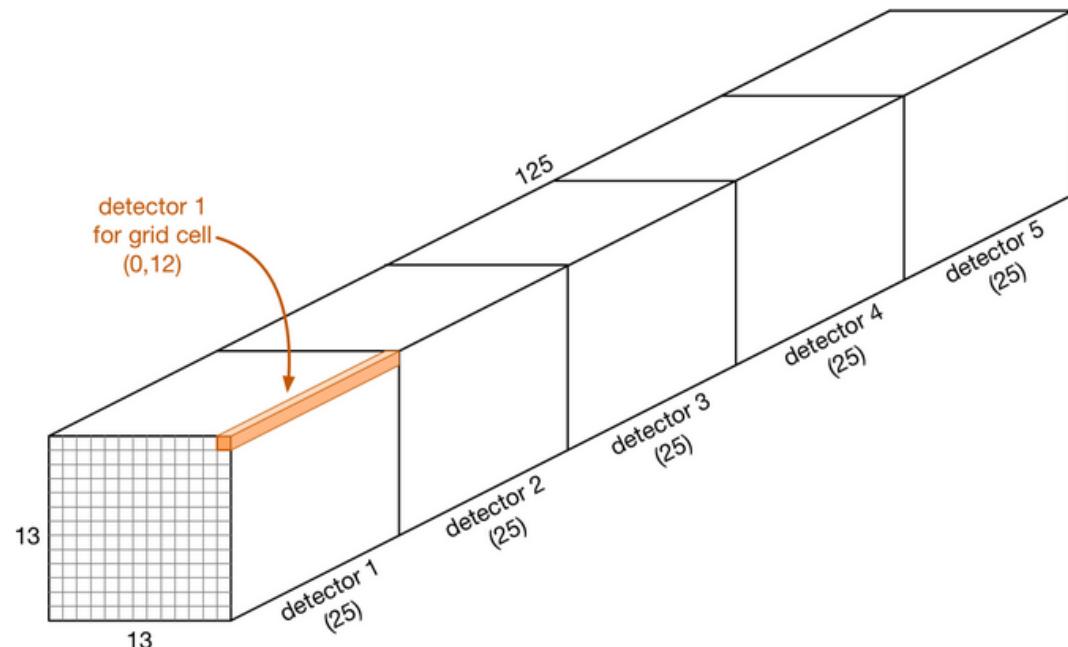


K-Means Clustering



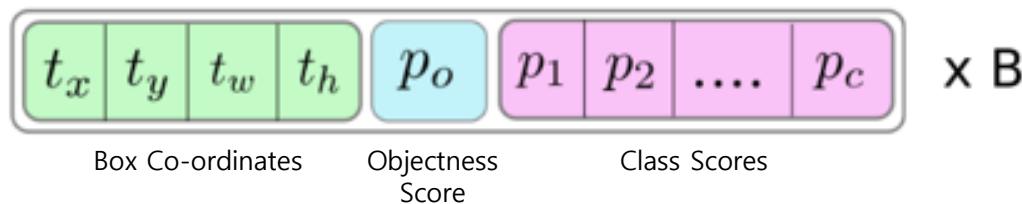
# YOLO v2 Output Feature Map

1.5 YOLO (You Only Look Once) 시리즈



Bounding Box별 25 개 정보

Attributes of a bounding box



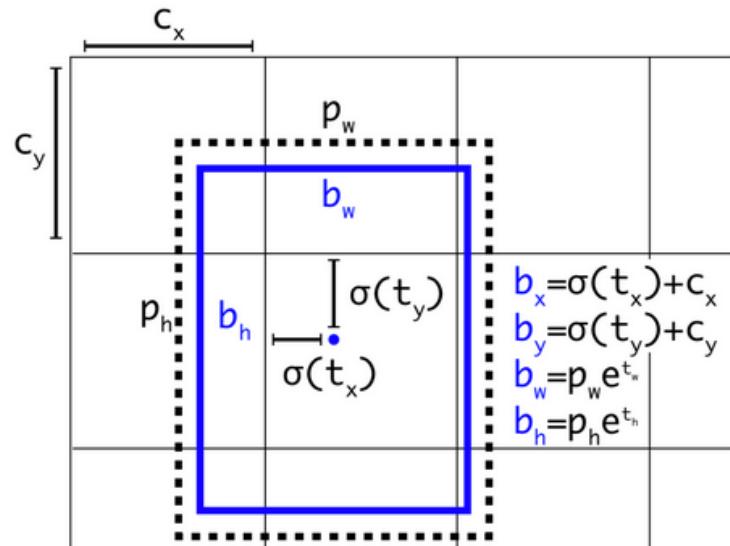
$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

$$\Pr(\text{object}) \cdot \text{IoU}(b, \text{object}) = \sigma(t_o)$$



(pw,ph): anchor box size

(tx,ty,tw,th): 모델 예측 offset 값

(bx,by ), (bw, bh): 예측 Bounding box 중심 좌표와 Size

Center 좌표가 Cell 중심을 너무 벗어나지 못하도록  
0 ~ 1사이의 시그모이드 값으로 조절

- 논문에는 YOLO V2 Loss 에 대한 별도 언급이 없음.
- YOLO V1 Loss 와 유사한 Loss 식

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

**BBox 중심 x, y 좌표 Loss**

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

**BBox 넓이 w, 높이 h Loss**

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

**Object Confidence Loss**

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

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

**Classification Loss**

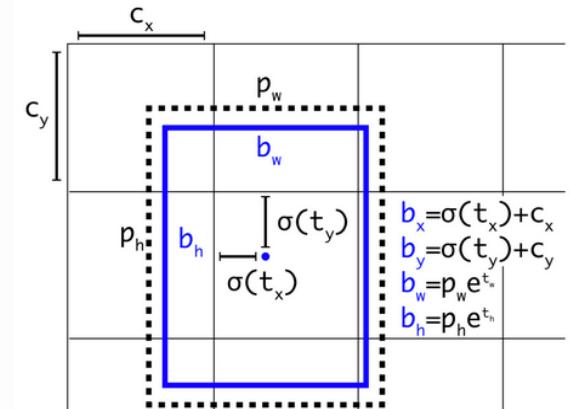
$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

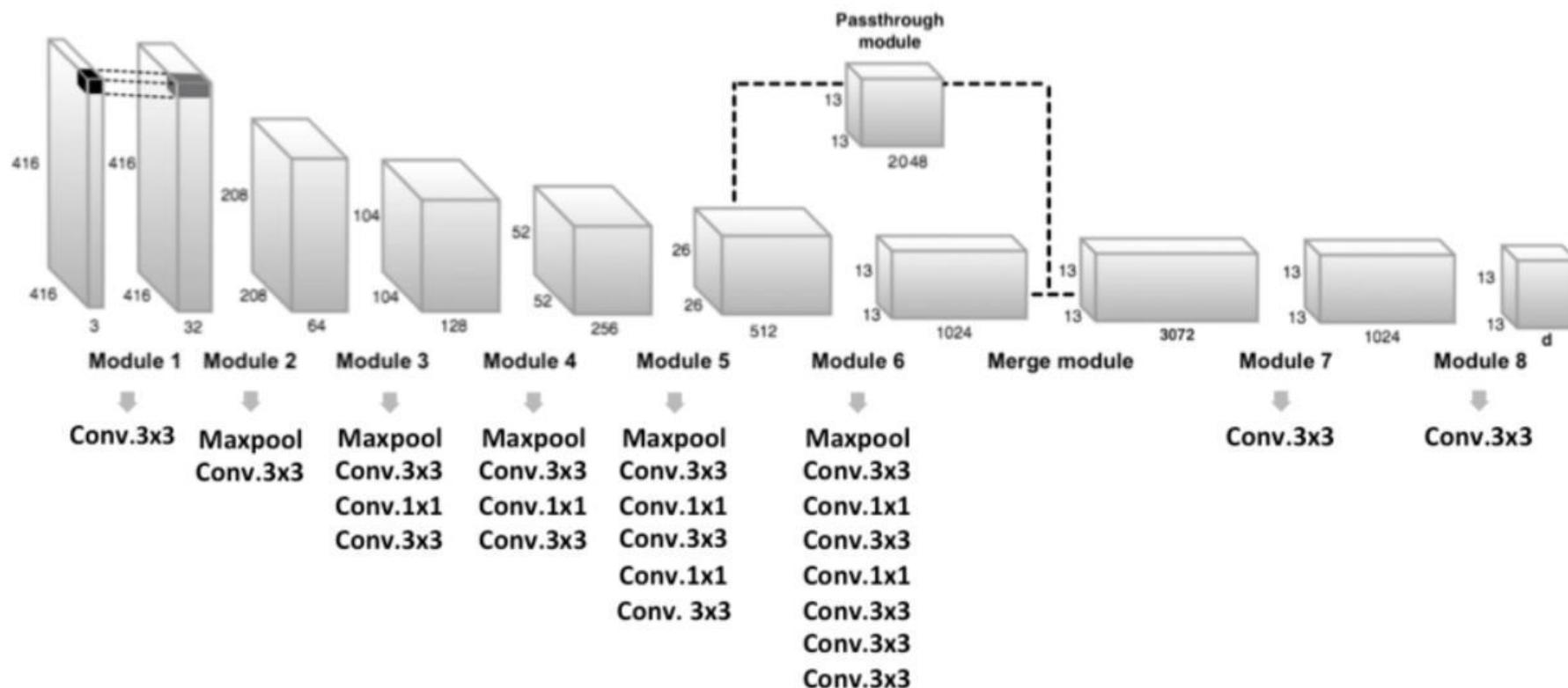
$$\Pr(\text{object}) \cdot \text{IoU}(b, \text{object}) = \sigma(t_o)$$



# Passthrough module 을 통한 fine grained feature

1.5 YOLO (You Only Look Once) 시리즈

좀 더 작은 오브젝트를 Detect하기 위해서  $26 \times 26 \times 512$  feature map의 특징을 유지한 채  $13 \times 13 \times 2048$ 로 reshape한 뒤  $13 \times 13 \times 1024$ 에 추가하여 feature map 생성



Classification layer가 Convolution layer로 생성하여 동적으로 입력 이미지 크기 변경 가능  
학습 시 10회 배치 시마다 입력 이미지 크기를 모델에서 320부터 608까지  
동적으로 변경 (32의 배수로 설정)

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

# Darknet 19 Backbone

1.5 YOLO (You Only Look Once) 시리즈

Darknet 19

Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			

VGG-16: 30.69 BFLOPS, Top 5 Accuracy: 90%

Yolo v1: 8.52 BFLOPS, Top 5 Accuracy: 88%

Darknet 19: 5.58 BFLOPS, Top 5 Accuracy: 91.2%

Classification layer에 Fully Connected layer를 제거하고 Conv layer를 적용

	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	<b>78.6</b>	

# One Stage Detector

1.5 YOLO (You Only Look Once) 시리즈

2015. 06

Yolo v1

2015. 12

SSD

2016. 12

Yolo v2

2017. 08

Retinanet

2018. 04

Yolo v3

2019. 11

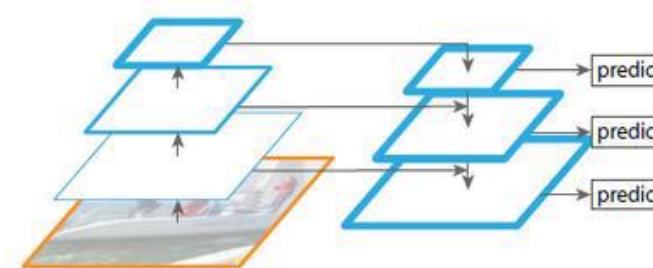
SSD

2020. 04

Yolo v4

Feature  
Pyramid  
Network

Feature Pyramid Network



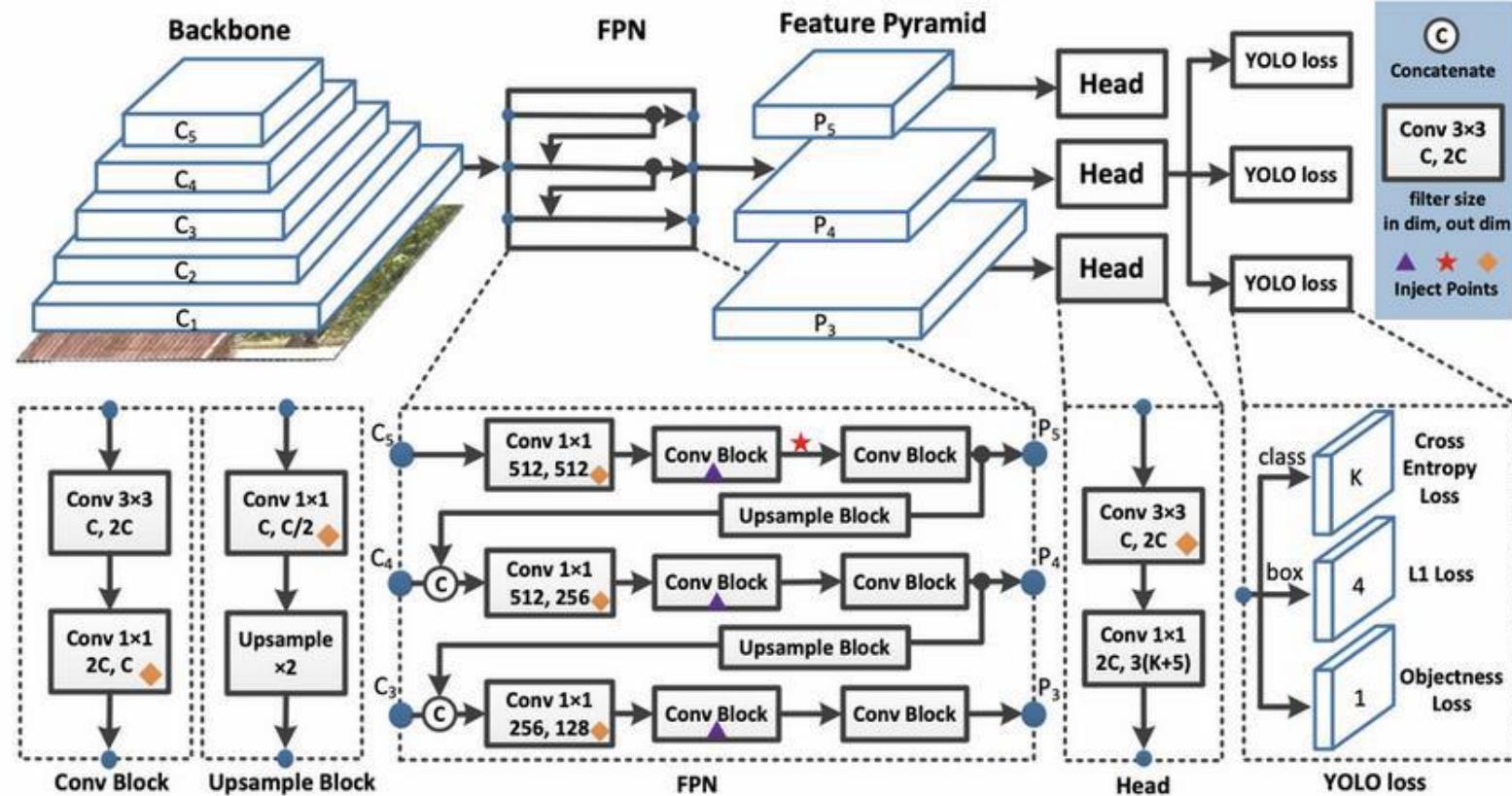
# YOLO-V1, V2, V3 비교

1.5 YOLO (You Only Look Once) 시리즈

anchor box 기반의 모델과 더 뛰어난 Backbone 구성 , 다양한 성능 향상 테크닉을 적용하면서 발전됨 .

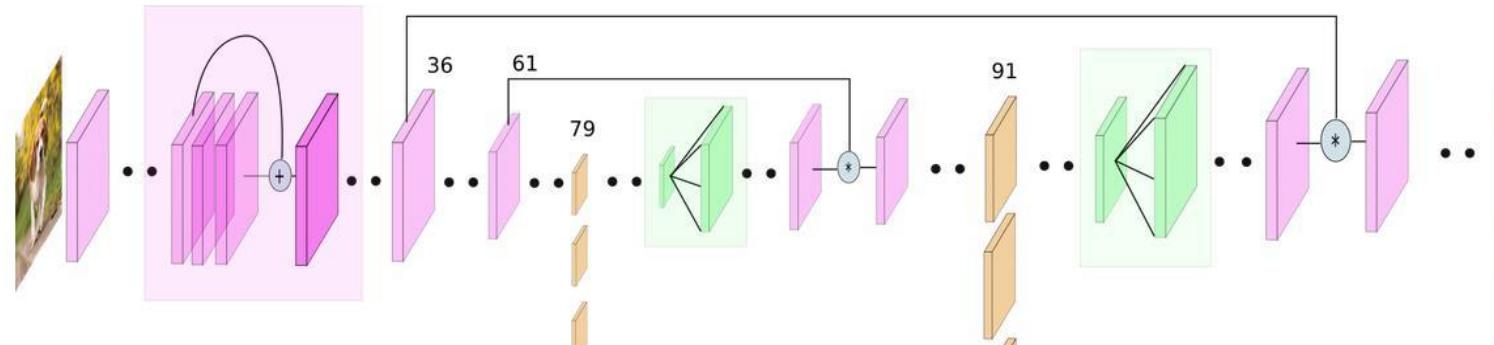
항목	V1	V2	V3
원본 이미지 크기	446X446	416X416	416X416
Feature Extractor	Inception 변형	Darknet 19	Darknet 53
Grid당 Anchor Box 수	2개	5개	Output Feature Map 당 3 개 서로 다른 크기와 스케일로 총 9 개
Anchor box 결정 방법		K-Means Clustering	K-Means Clustering
Output Feature Map 크기 (Depth 제외)	7X7	13X13	13X13, 26X26, 52X52 3개의 Feature Map 사용
Feature Map Scaling 기법			FPN(Feature Pyramid Network)

- Feature Pyramid Network 유사한 기법을 적용하여 3개의 Feature Map Output에서 각각 3 개의 서로 다른 크기와 scale을 가진 anchor box로 Detection
- Backbone 성능 향상 Darknet 53
- Multi Labels 예측 : Softmax가 아닌 Sigmoid 기반의 logistic classifier로 개별 Object의 Multi labels 예측



# YOLO-V3 Network 구조

1.5 YOLO (You Only Look Once) 시리즈

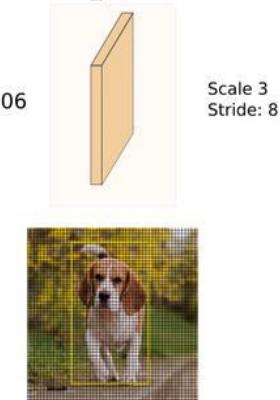


Feature Pyramid Network



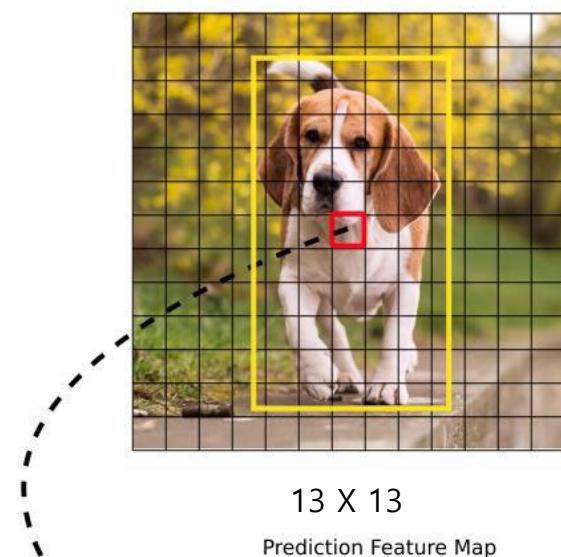
참조: what's new in YOLO v3

<https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>

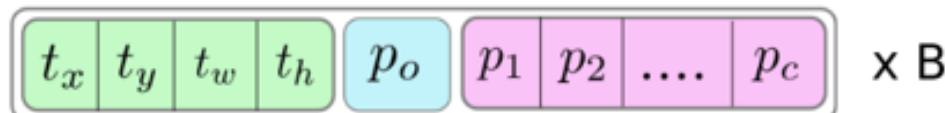


# YOLO-V3 Output Feature Map

1.5 YOLO (You Only Look Once) 시리즈



Attributes of a bounding box



Box Co-ordinates

Objectness  
Score

Class Scores

<https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/>

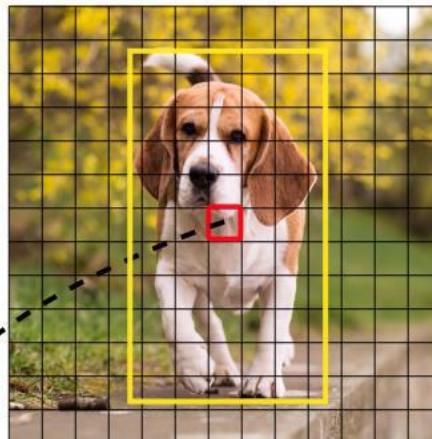
# Darknet-53 특성

1.5 YOLO (You Only Look Once) 시리즈

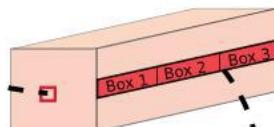
Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	<b>77.6</b>	<b>93.8</b>	29.4	1090	37
Darknet-53	77.2	<b>93.8</b>	18.7	<b>1457</b>	78

Type	Filters	Size	Output
Convolutional	32	$3 \times 3$	$256 \times 256$
Convolutional	64	$3 \times 3 / 2$	$128 \times 128$
1x	32	$1 \times 1$	
	64	$3 \times 3$	
	Residual		$128 \times 128$
Convolutional	128	$3 \times 3 / 2$	$64 \times 64$
2x	64	$1 \times 1$	
	128	$3 \times 3$	
	Residual		$64 \times 64$
Convolutional	256	$3 \times 3 / 2$	$32 \times 32$
8x	128	$1 \times 1$	
	256	$3 \times 3$	
	Residual		$32 \times 32$
Convolutional	512	$3 \times 3 / 2$	$16 \times 16$
8x	256	$1 \times 1$	
	512	$3 \times 3$	
	Residual		$16 \times 16$
Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$
4x	512	$1 \times 1$	
	1024	$3 \times 3$	
	Residual		$8 \times 8$
Avgpool		Global	
Connected		1000	
Softmax			

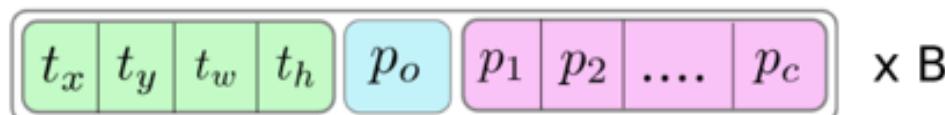
```
<annotation>
  <folder>VOC2007</folder>
  <filename>003585.jpg</filename>
  <size>
    <width>333</width>
    <height>500</height>
    <depth>3</depth>
  </size>
  <object>
    <name>person</name>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>138</xmin>
      <ymin>183</ymin>
      <xmax>259</xmax>
      <ymax>411</ymax>
    </bndbox>
  </object>
  <object>
    <name>motorbike</name>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>89</xmin>
      <ymin>244</ymin>
      <xmax>291</xmax>
      <ymax>425</ymax>
    </bndbox>
  </object>
  ...
</annotation>
```



13 X 13  
Prediction Feature Map



Attributes of a bounding box

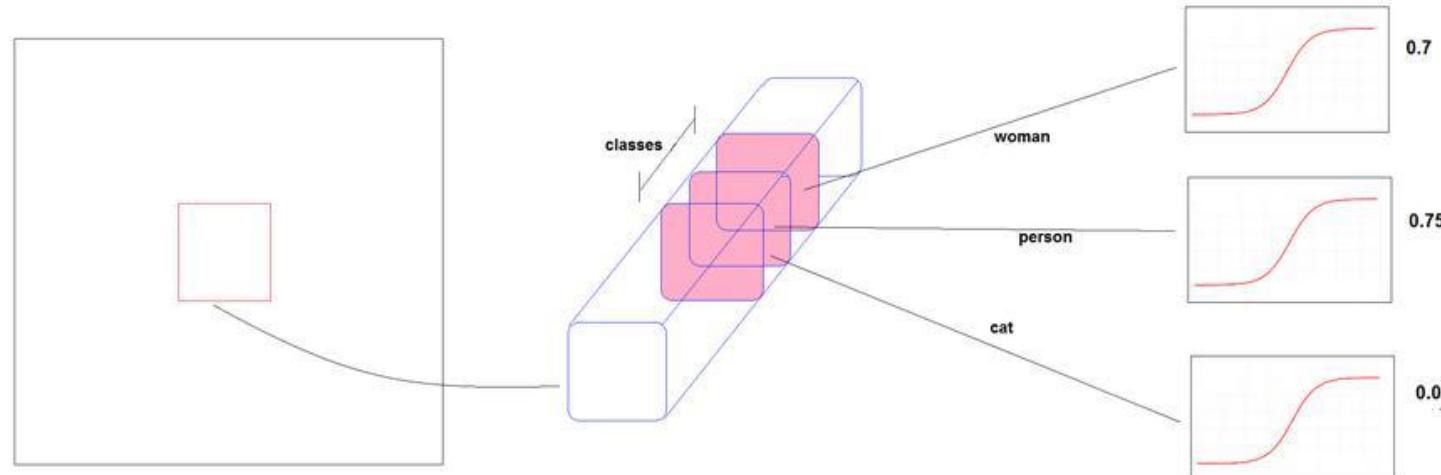


Box Co-ordinates

Objectness  
Score

Class Scores

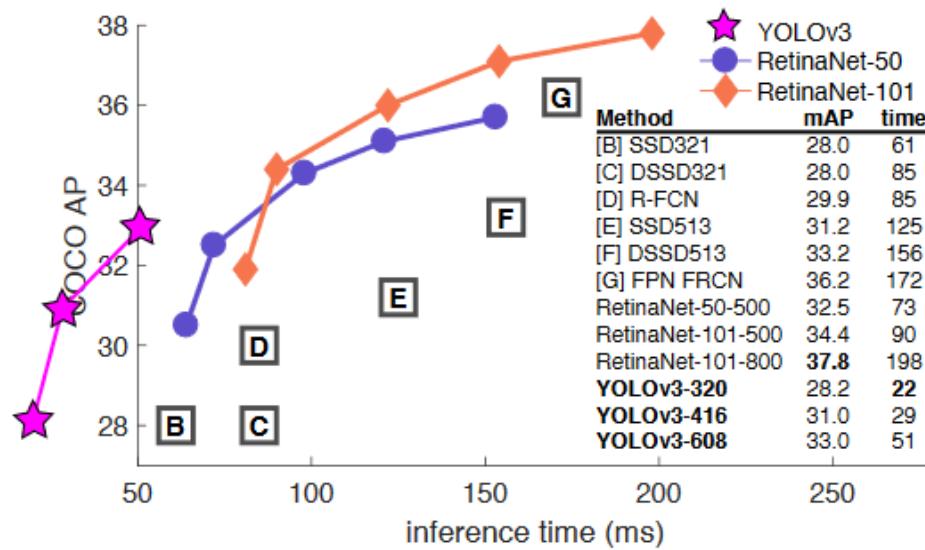
여러 개의 독립적인 Logistic Classifier 사용



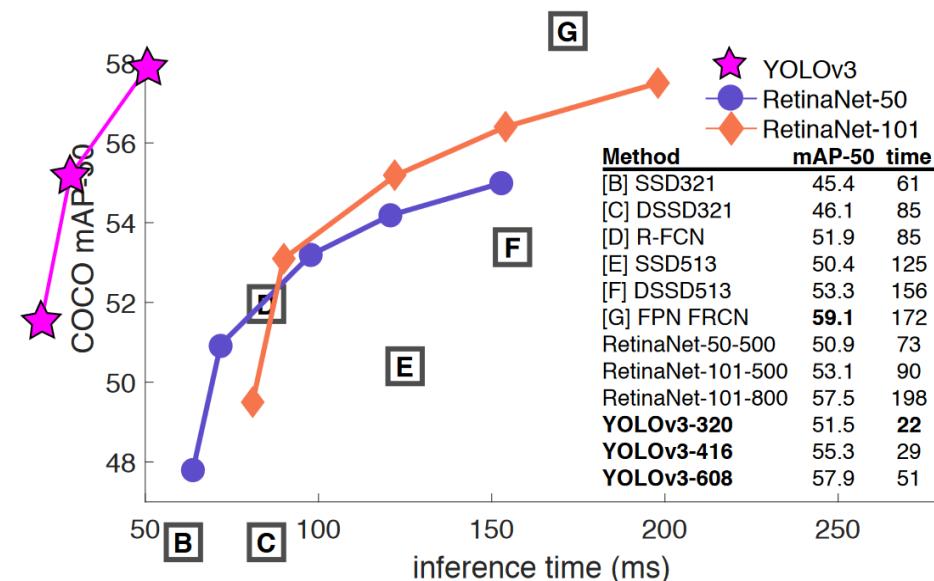
# YOLO-V3 성능 비교

1.5 YOLO (You Only Look Once) 시리즈

COCO(IOU 0.5 ~ 0.95 기준)



COCO(IOU 0.5 기준)

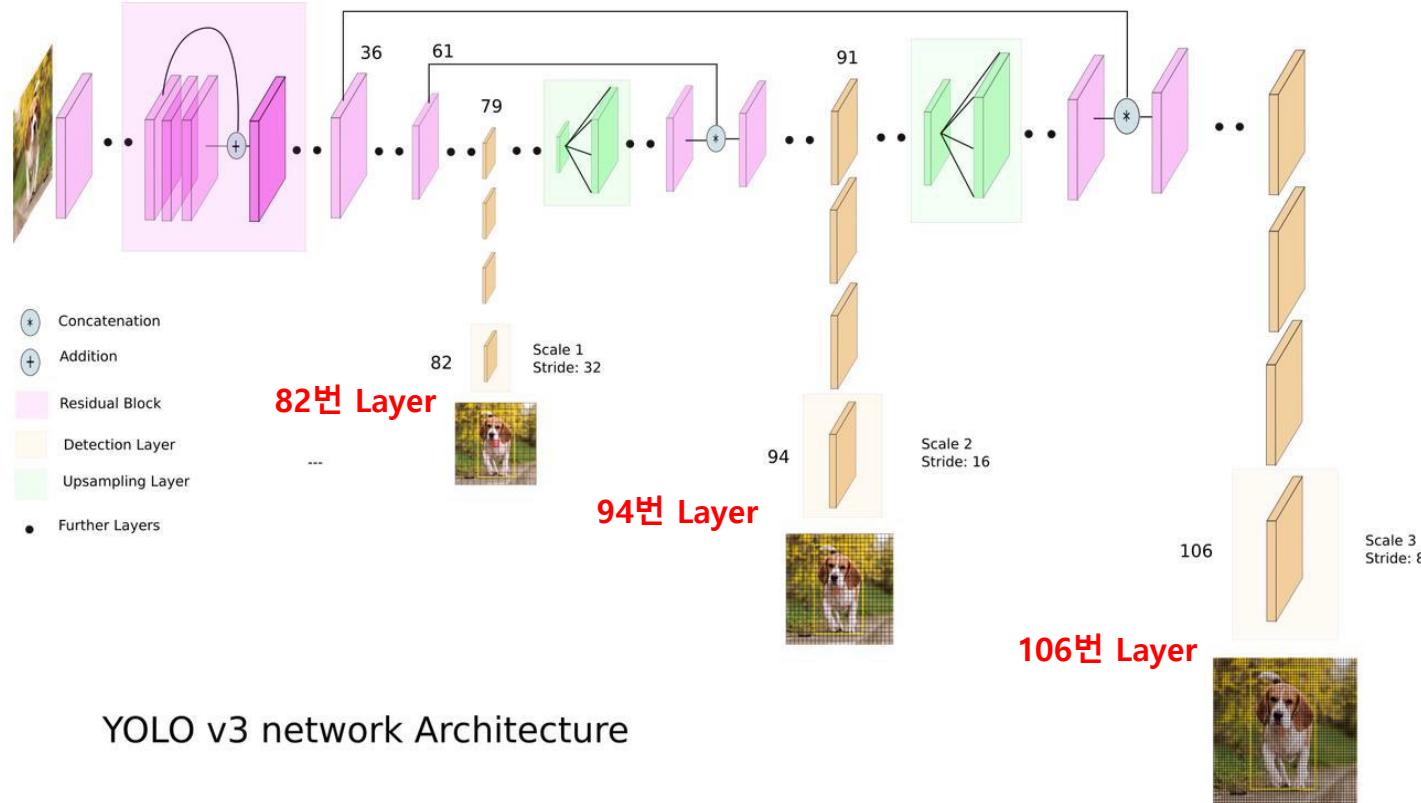


- OpenCV Yolo inference 코드는 기존 OpenCV inference 코드와 다름
- 3 개의 Output Feature Map에서 직접 Object Detection 정보 추출

- Weight 모델 파일과 config 파일은 Darknet 사이트에 Download 가능(<https://pjreddie.com/darknet/yolo/>)
- cv2.dnn.readNetFromDarknet( config 파일 , weight 모델 파일)으로 pretrained 된 inference 모델 로딩
- readNetFromDarket (**config 파일** , weight 모델 파일)에서 config 파일 인자가 weight 모델 파일 인자보다 먼저 위치함에 유의

# 3개의 scale Output Layer 에서 직접 Detection 결과 추출

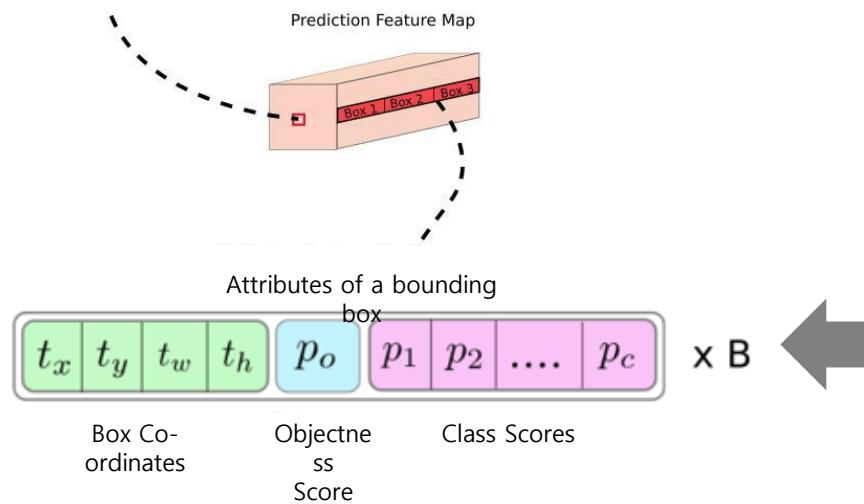
1.5 YOLO (You Only Look Once) 시리즈



사용자가 직접 3개의 다른Scale별로 구성된 output layer에서 Object Detect 결과를 추출해야 함.



사용자가 직접 NMS(Non Maximum Suppressing)로 최종 결과 필터링 해야 함.



Coco데이터 세트로 Pretrained 된 모델에서  
bounding box 정보 추출하기

- Bounding Box 정보를 4개의 Box좌표, 1 개의 Object Score, 그리고 80개의 Class score(Coco 는 80 개의 Object Category 임)로 구성된 총 85 개의 정보 구성에서 정보 추출 필요.
- Class id와 class score는 이 80개 vector에서 가장 높은 값을 가지는 위치 인덱스와 그 값임.

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

- OpenCV Yolo로 추출한 좌표는 Detected 된 Object의 center와 width, height 값이므로 이를 좌상단, 우하단 좌표로 변경 필요.

1

cv2.dnn.readNetFromDarknet(config파일, weight 모델 파일)으로 pretrained된 inference 모델 로딩

- Config 파일 위치 주의

2

사용자가 3개의 다른 Scale별로 구성된 output layer에서 Object Detect 결과 추출

- Detected된 Object 당 85개의 vector를 가짐. 4개의 위치 정보는 center점의 x, y 좌표와 width, height이므로 이를 좌상단, 우하단 좌표로 변경 필요.
- Class id 와 class score 는 이 80 개 vector 에서 가장 높은 값을 가지는 위치 인덱스와 그 값 .

3

사용자가 직접 NMS(Non Maximum Suppressing)로 최종 결과 필터링 해야 함

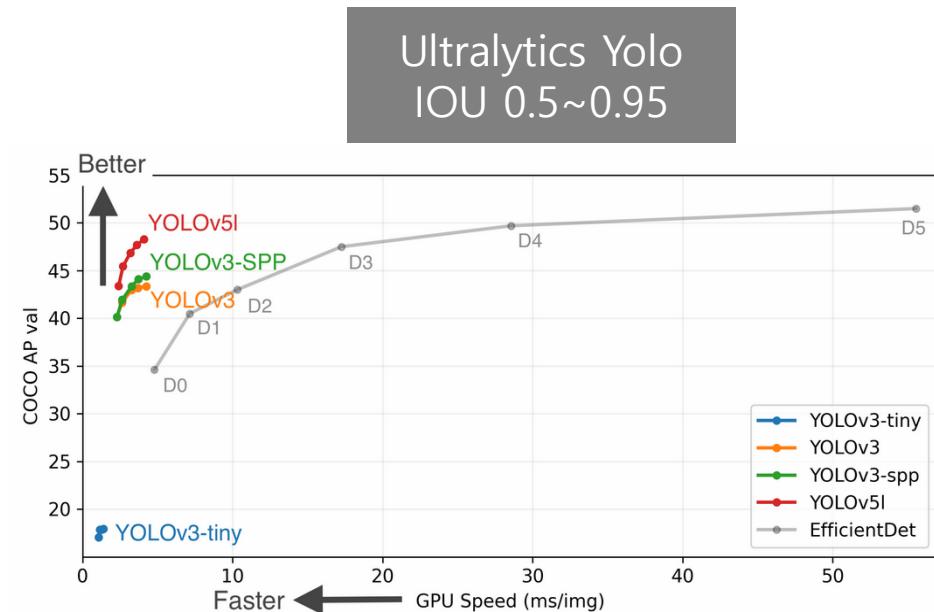
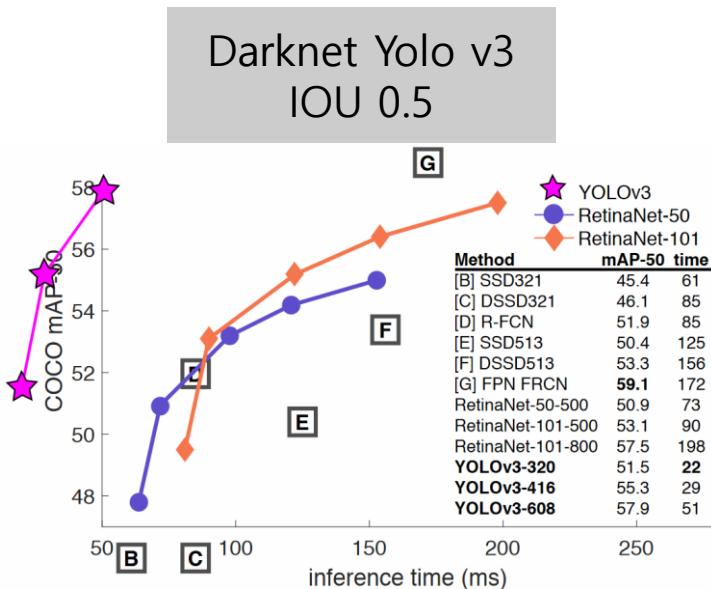
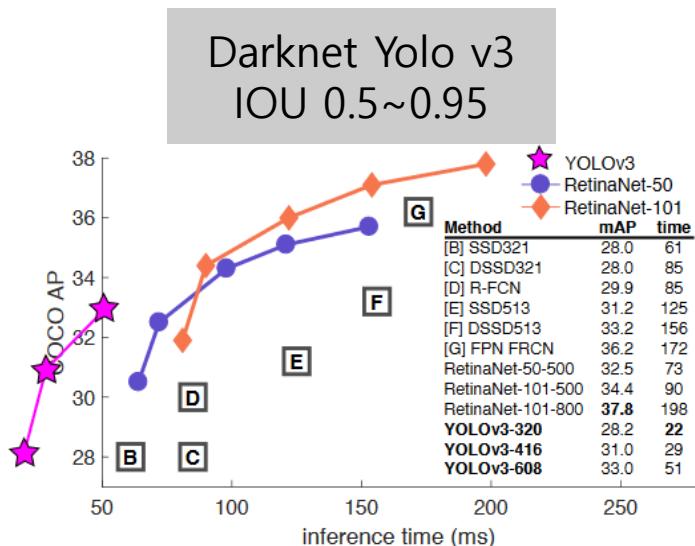
- OpenCV에서 제공하는 NMS 함수로 NMS 최종 필터링 된 Object 시각화

## Ultralytics YOLO v3



# Ultralytics Yolo Performance

1.5 YOLO (You Only Look Once) 시리즈



초기의 Yolo v3에서 Yolo v4 등에 적용된 일부 성능 개선 효과들을 구현하면서 성능이 향상됨.

# Ultralytics Yolo Performance

1.5 YOLO (You Only Look Once) 시리즈

예측 성능 뿐만 아니라 예측 시간이 크게 단축

Model	size (pixels)	mAP <sup>val</sup> 0.5:0.95	mAP <sup>test</sup> 0.5:0.95	mAP <sup>val</sup> 0.5	Speed V100 (ms)		params (M)	FLOPS 640 (B)
YOLOv3-tiny	640	17.6	17.6	34.8	1.2		8.8	13.2
YOLOv3	640	43.3	43.3	63.0	4.1		61.9	156.3
YOLOv3-SPP	640	44.3	44.3	64.6	4.1		63.0	157.1
YOLOv5l	640	48.2	48.2	66.9	3.7		47.0	115.4

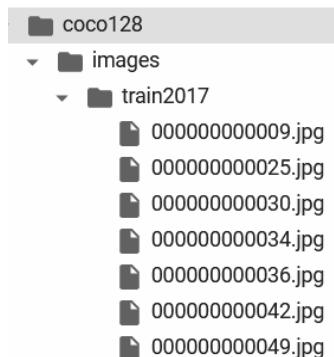
- 견고한 구현 (Enterprise 솔루션 지향)
- 다양한 편의 기능
  - 학습 시 loss, weigh 등에 대한 시각적인 util 기능 제공
  - 편리한 evaluation 결과 도출 및 시각화

1. Ultralytics 학습용 데이터 포맷으로 annotation 생성하고 , 이미지와 annotation 디렉토리를 적절하게 구성
2. Dataset 과 관련된 config yaml 설정
3. Yolo v3, Tiny Yolo v3, Yolo v3 SPP 모델 지정 Pretrained 모델을 선택하거나 model config 를 지정
4. train.py 에 batch size, 이미지 크기 , epochs 횟수 등을 인자로 입력하여 학습 수행

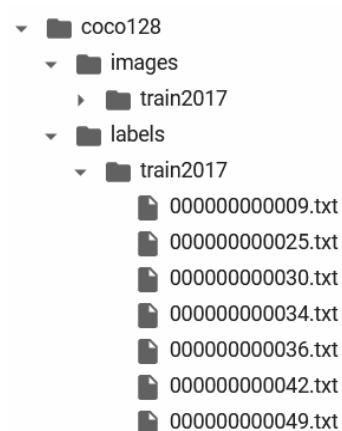
```
python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml --weights yolov3.pt --nosave
```

Pascal VOC와 유사하게 1 개의 이미지별로 1 개의 Annotation 을 가짐  
annotation파일의 확장자는 .txt 임

이미지 Directory



Annotation Directory



coco128.yaml

```
# download command/URL (optional)
download: https://github.com/ultralytics/yolov5/releases/

# train and val data as 1) directory: path/images/, 2) file pattern: path/images/img*.jpg
train: ./coco128/images/train2017/ # 128 images
val: ./coco128/images/train2017/ # 128 images

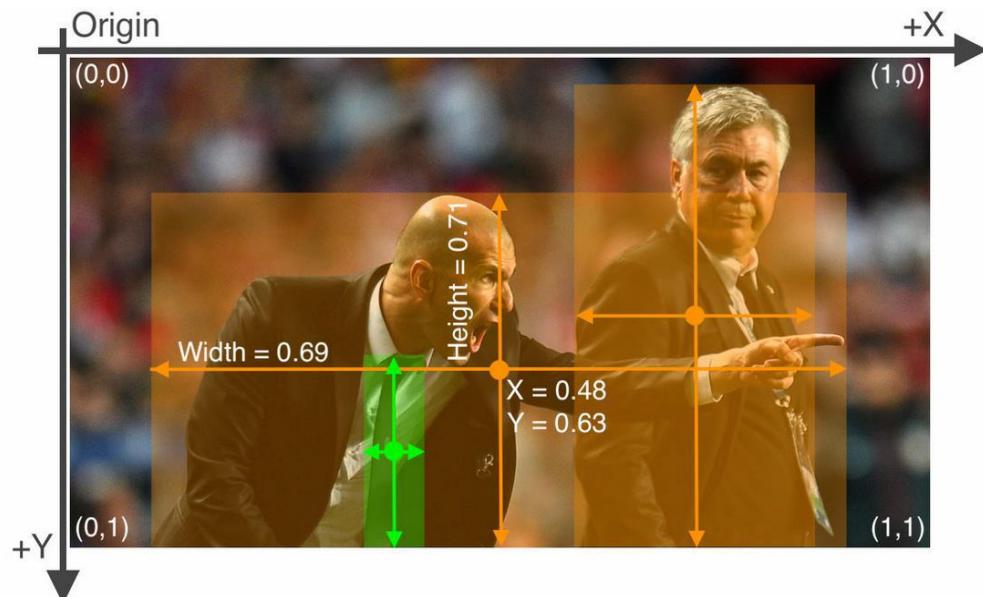
# number of classes
nc: 80
```

- ✓ label 디렉토리는 지정하지 않아도 image 디렉토리 이름에서 마지막에 있는 'images' 를 'labels' 로 변환한 디렉토리로 자동 지정됨 .
- ✓ 이미지 디렉토리는 반드시 'images' 문자열을 포함해야 하고 , label 디렉토리는 반드시 'labels' 문자열을 포함해야 함 .

# Ultralytics Yolo annotation 포맷

1.5 YOLO (You Only Look Once) 시리즈

5개의 값이 공백으로 분리되며 ,첫 번째 값은 0 부터 시작하는 Class ID, 두번째 부터 다섯 번째는 각각 Bounding box 의 중심 x 좌표 , 중심 y 좌표 , 너비 (Width), 높이(Height)이며 , 각 위치 값은 이미지의 너비 , 높이 대비 값으로 Normalization 된 0 ~ 1 사이 값



class id	x center	y center	width	height
0	0.481719	0.634028	0.690625	0.713278
0	0.741094	0.524306	0.314750	0.933389
27	0.364844	0.795833	0.078125	0.400000

# 중심 좌표 값 Normalization 의 장점

1.5 YOLO (You Only Look Once) 시리즈

- Yolo 는 내부 알고리즘이 Loss 및 Predict 계산 시 Bounding Box 의 중심 좌표 (Center X, Center Y) 그리고 너비 , 높이를 기반으로 적용
- 좌표 값을 이미지 크기 대비 0 ~ 1 사이로 Normalization 하게 되면 서로 다른 이미지 크기들의 Bounding box 좌표 값을 다시 재조정할 필요 없음

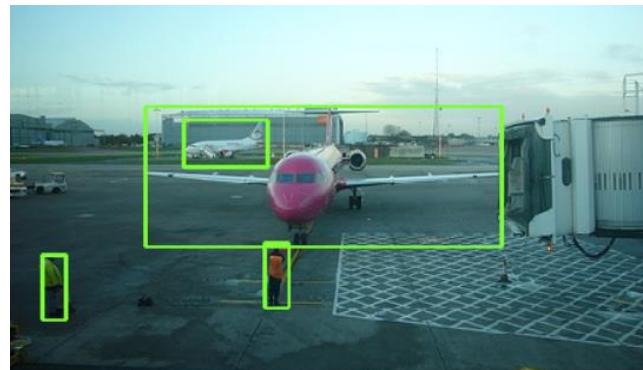
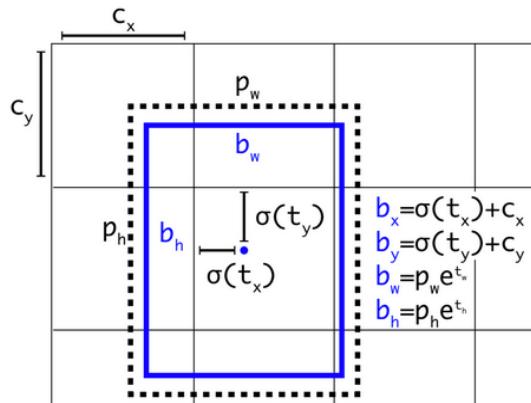
$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

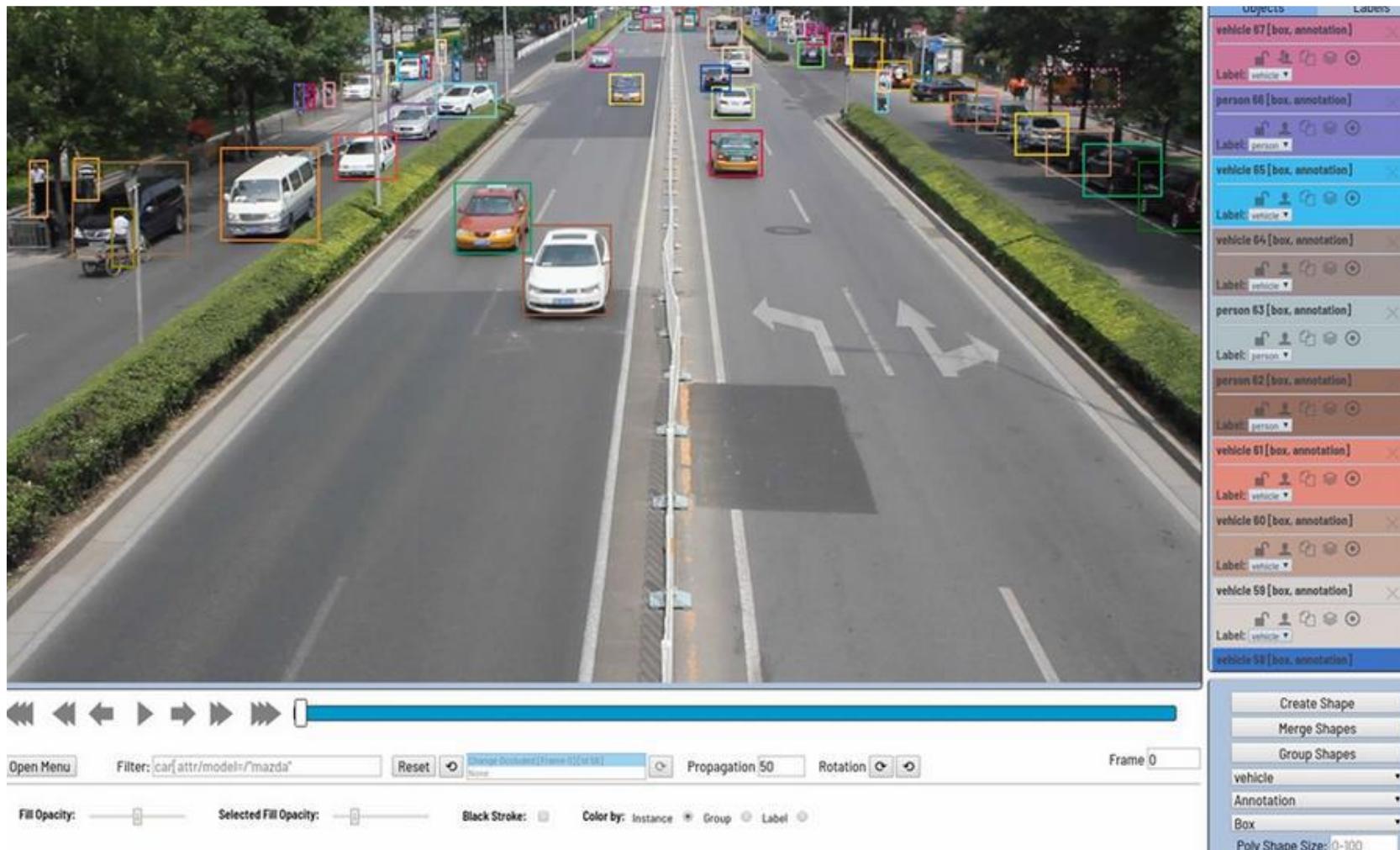
$$\text{Pr(object)} \cdot \text{IoU}(b, \text{object}) = \sigma(t_o)$$



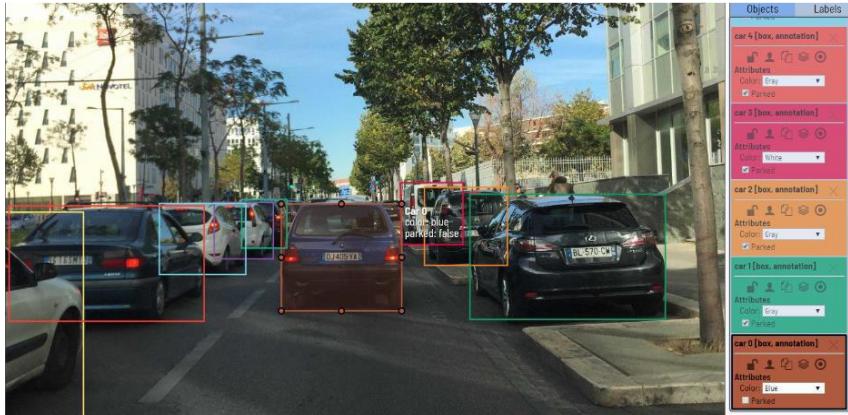
- 데이터 세트 위치 , 클래스명/개수 : dataset.yaml
- 학습율 , loss, augmentation 등의 하이퍼 파라미터 : hyp.scratch.xml
- batch 크기 , epochs, image 크기 : train.py 의 인자로 입력
- model config : yolov3.yaml, yolov3 tiny.yaml, yolov3-spp.yaml

# CVAT(Computer Vision Annotation Tool) 소개

1.5 YOLO (You Only Look Once) 시리즈



### Object Detection



입력 소스

JPEG

출력 지원 포맷

Pascal VOC

PNG

COCO

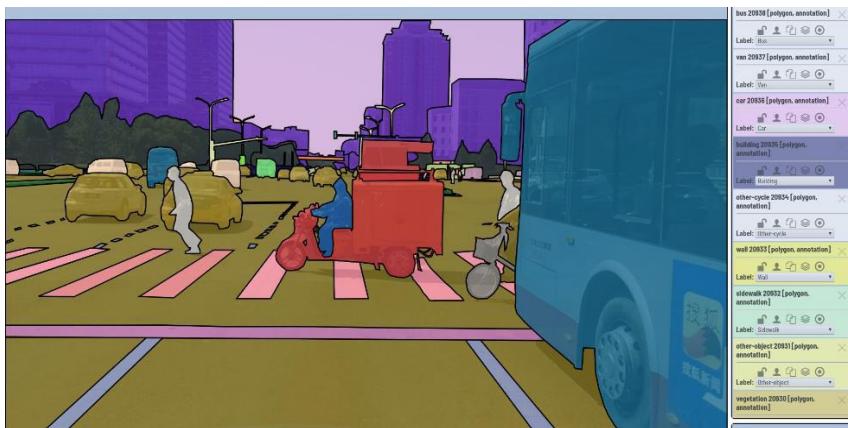
TIFF

YOLO

Video

.....

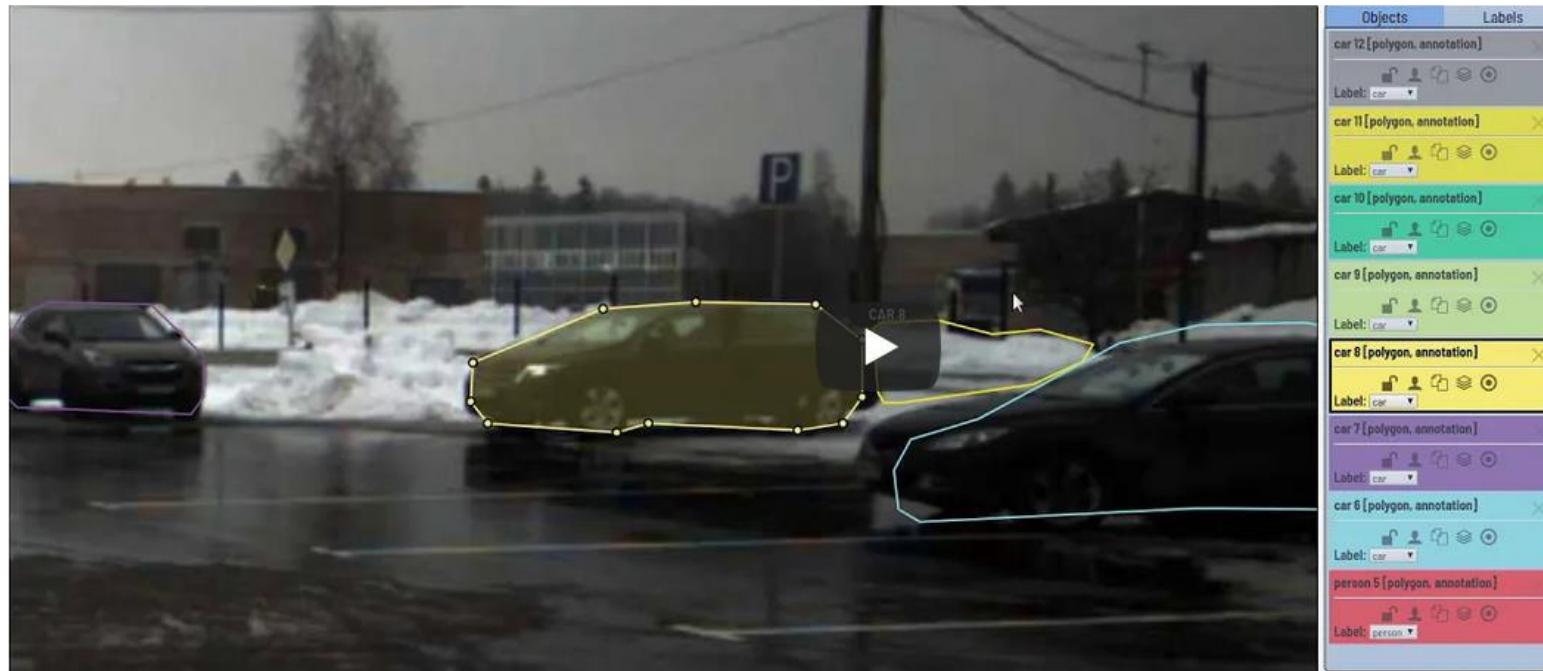
### Segmentation



.....

# Semi-Auto Segmentation 기능 지원

1.5 YOLO (You Only Look Once) 시리즈



### cvat.org 서버 기반 작업

- 500MB 의 Image Upload 제한

### Local Install 기반 작업

- 로컬에 웹서버를 구성하여 작업

# 1. 오브젝트 디텍션

---

- 1.1 오브젝트 디텍션 개요
- 1.2 OpenCV 개요
- 1.3 R-CNN, Fast R-CNN, Faster R-CNN
- 1.4 SSD (Single Shot MultiBox Detector)
- 1.5 YOLO (You Only Look Once) 시리즈

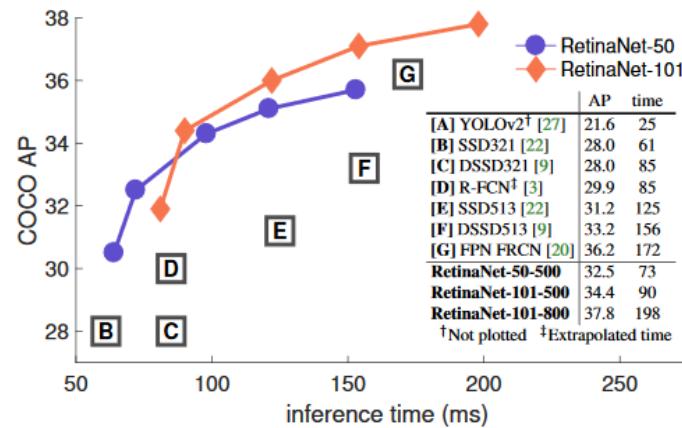
## 1.6 RetinaNet

---

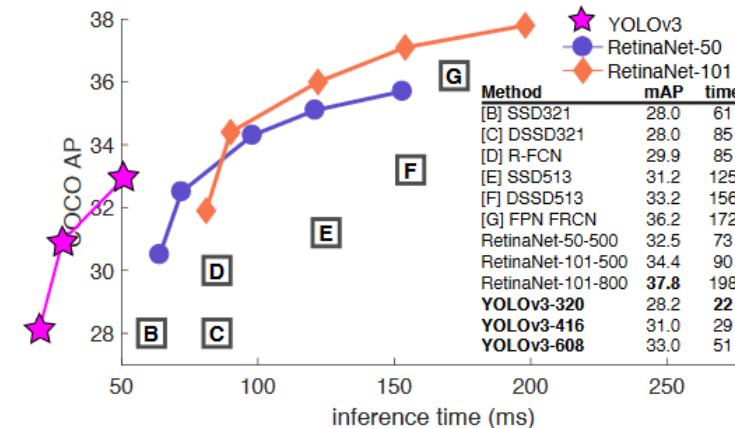
- One Stage Detector의 빠른 Detection 시간의 장점을 가지면서 One Stage Detector의 Detection 성능 저하 문제 개선
- 수행시간은 YOLO 나 SSD 보다 느리지만 Faster RCNN 보다 빠름
- 수행 성능은 타 Detection 모델 보다 뛰어남. 특히 타 One Stage Detectior 보다 작은 오브젝트에 대한 Detection 능력이 뛰어남

Focal Loss + Feature Pyramid Network

RetinaNet 논문에서 발췌

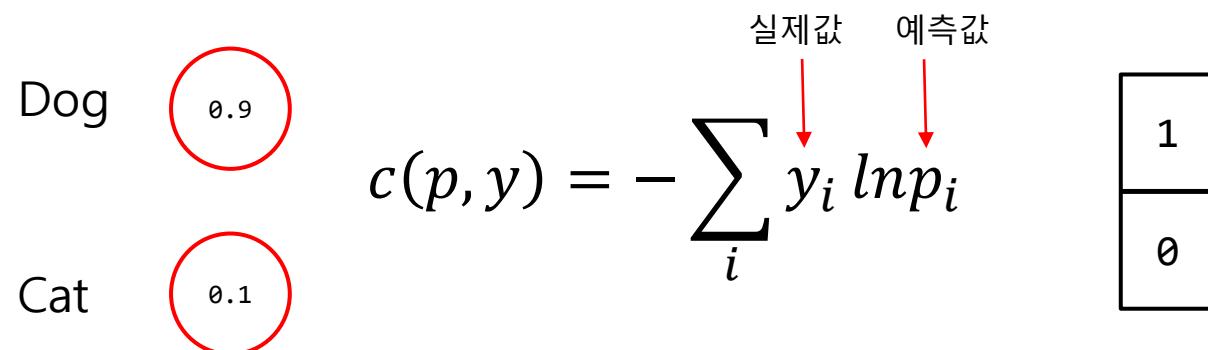


YOLO v3 논문에서 발췌



# Focal Loss의 필요성 - Cross Entropy

1.6 RetinaNet



# Focal Loss의 필요성 - Cross Entropy

1.6 RetinaNet

NN1

Row	Dog^	Cat^	Dog	Cat
#1	0.9	0.1	1	0
#2	0.1	0.9	0	1
#3	0.4	0.6	1	0

NN2

Row	Dog^	Cat^	Dog	Cat
#1	0.6	0.4	1	0
#2	0.3	0.7	0	1
#3	0.1	0.9	1	0

Classification Error

$$1/3 = 0.33$$

$$1/3 = 0.33$$

Mean Squared Error

$$0.25$$

$$0.71$$

Cross-Entropy

$$0.38$$

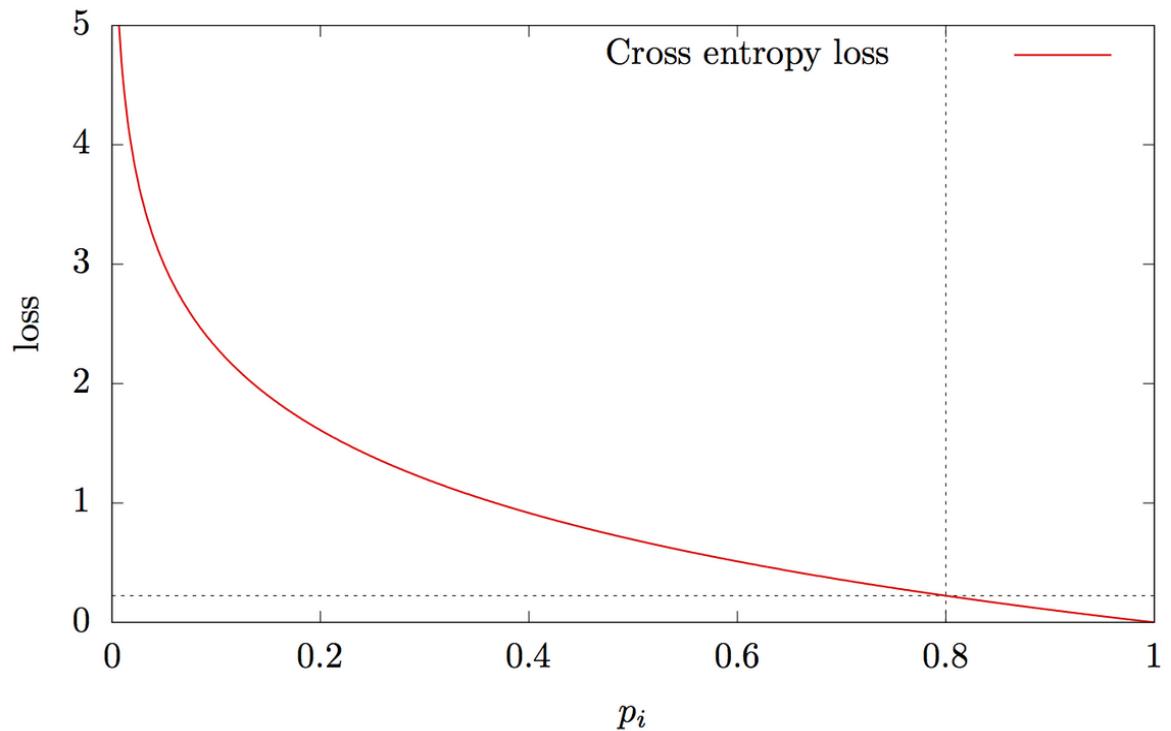
$$1.06$$

# Cross Entropy Loss 곡선

1.6 RetinaNet

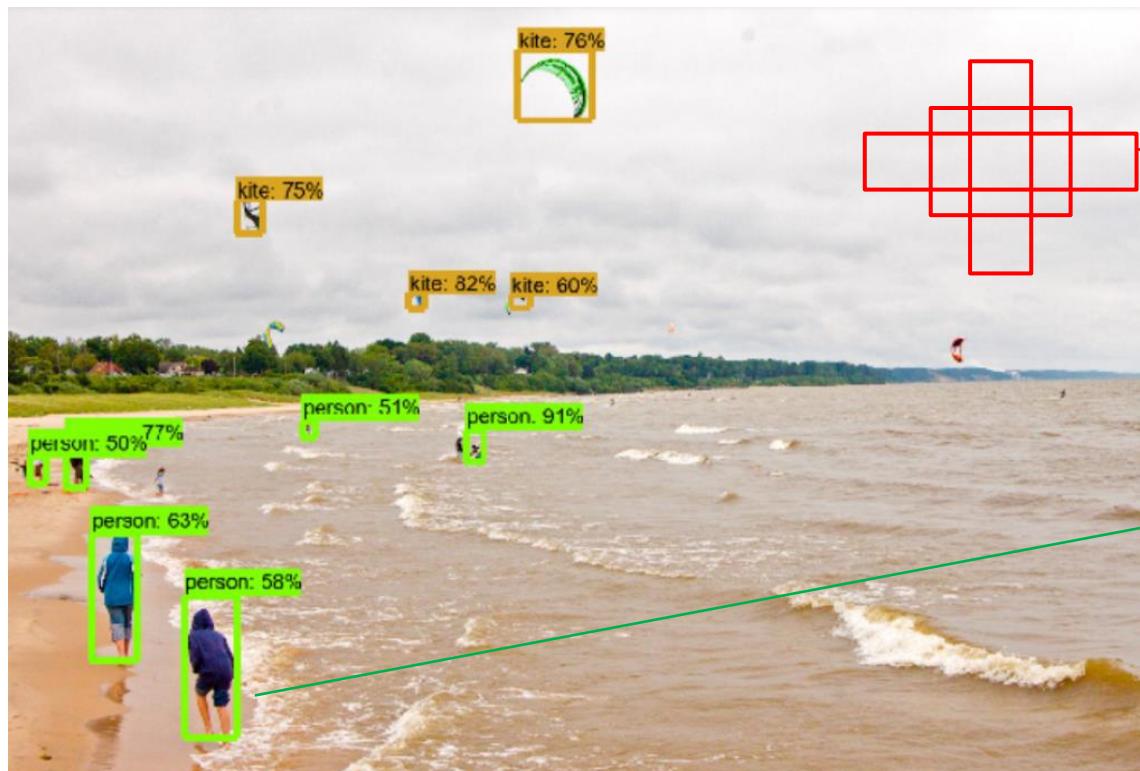
Cross Entropy Loss

$$C(p, y) = - \sum_i y_i \ln p_i$$



# Object Detection의 Class imbalance 이슈

## 1.6 RetinaNet



- 많은 background(negative) exempl들
- 대부분의 ancho 들이 background에 치중

- 매우 적은 foreground(positive) exempl들
- 매우 유용한 정보를 모델에 제공

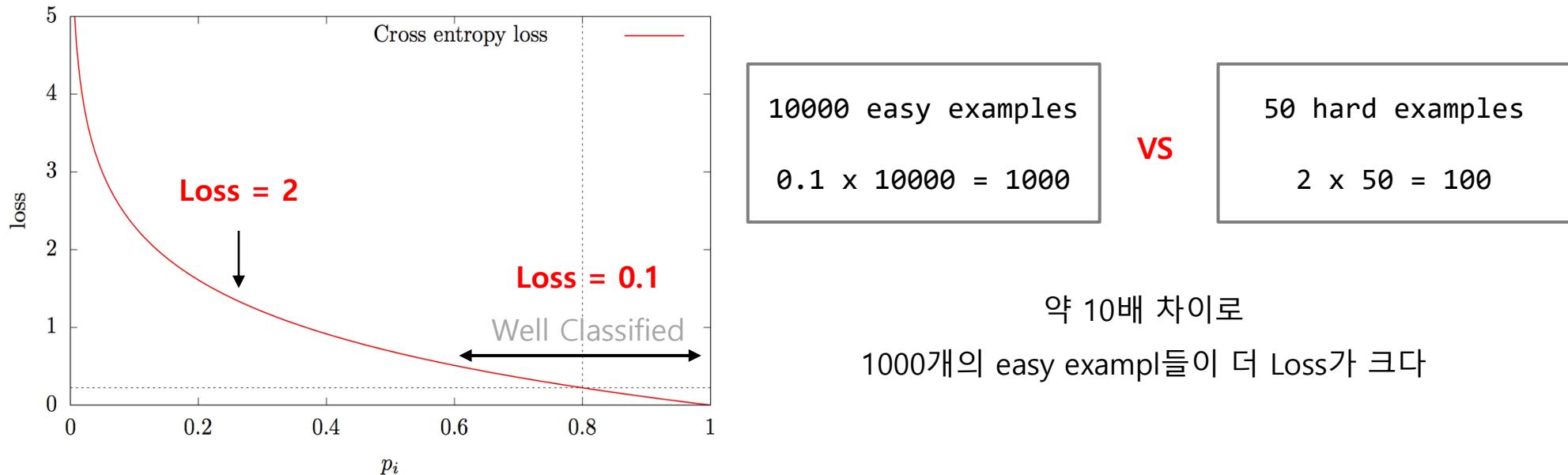
- Easy Example : 찾기 쉬운 대상들. Background나 크고 선명한 대상 오브젝트. 이미 높은 예측 확률을 가지고 있음
- Hard Example : 찾기 어려운 대상들. 작고 형태가 불분명하여 낮은 예측 확률을 가지고 있음

### One Stage Object Detector의 Class imbalance 문제

- Easy Example이 많고 Hard Example이 적은 Class imbalance라는 Object Detection이 안고 있는 고유 문제
- Two Stage Detector의 경우는 Region Proposal Network에서 오브젝트가 있을만한 높은 확률 순으로 필터링을 먼저 수행할 수 있음
- 하지만 One Stage는 Region Proposal과 Detection을 같이 수행하므로 매우 많은 오브젝트 후보들에 대해서 Detection을 수행해야 하므로 Class imbalance로 인한 성능 저하 영향이 큼

# Class imbalance 상태에서 Cross entropy의 문제점

1.6 RetinaNet



예측이 불확실한 Object들을 개선하려는 방향성으로 학습이 진행되는 것이 아니라 이미 높은 예측 확률을 가진 easy example, 즉 Background나 확실한 Object들에 대해서 더 정확한 예측을 하기 위해 학습이 진행됨

기존 One Stage Object Detector의  
Class Imbalance 해결 방안

RetinaNet의  
Class Imbalance 해결 방안

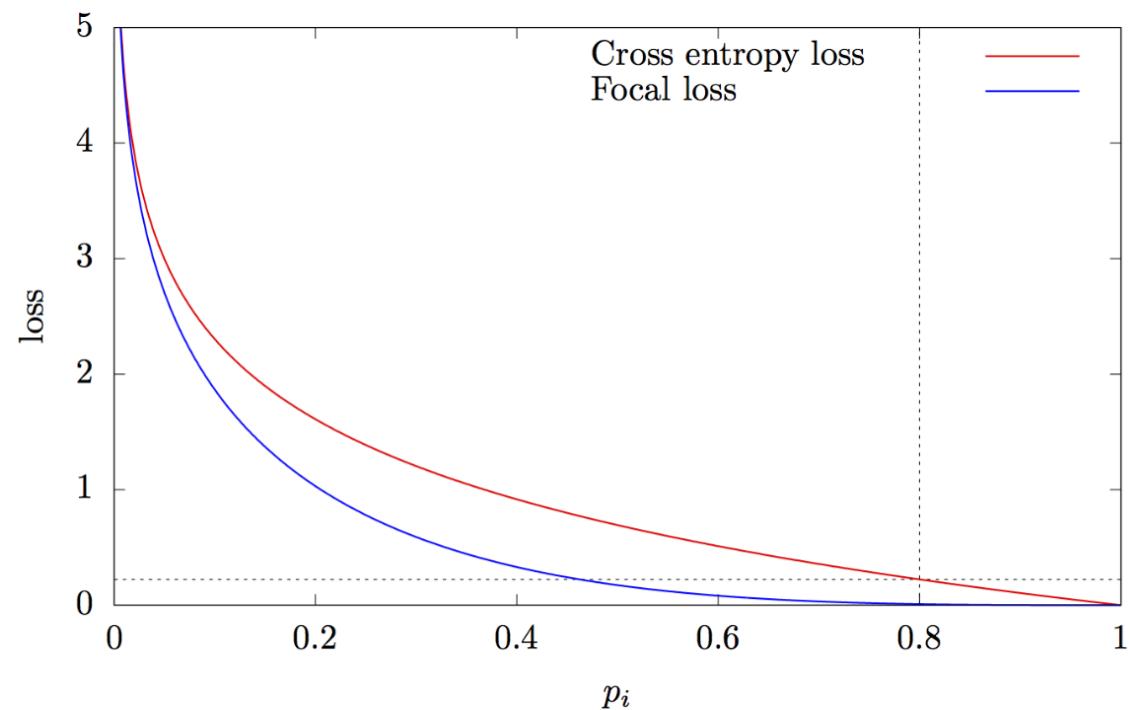
- 학습 시 경험 치에 기반한 샘플링이나  
data augmentation에 집중
- 동적으로 Cross entropy를 조절할 방법을 찾자

# Focal Loss - Cross Entropy에 가중치를 부여

1.6 RetinaNet

Focal Loss

$$C(p, y) = - \sum_i y_i (1 - p_i)^\gamma \ln p_i$$



# Focal Loss는 확실한 Object들에 매우 작은 loss를 부여

## 1.6 RetinaNet

쉽게 판별

P = 0.9

$$\text{CE}(\text{foreground}) = -\log(0.9) = 0.1053$$

alpha는 0.25, gamma=2

$$\text{CE}(\text{background}) = -\log(1-0.1) = 0.1053$$

$$\text{FL}(\text{foreground}) = -1 \times 0.25 \times (1-0.9)^{\gamma} \log(0.9) = 0.00026$$

$$\text{FL}(\text{background}) = -1 \times 0.25 \times (1-(1-0.9))^{\gamma} \log(1-0.9) = 0.00026$$

$$C(p, y) = - \sum_i y_i (1-p_i)^{\gamma} \ln p_i$$

잘못 판별

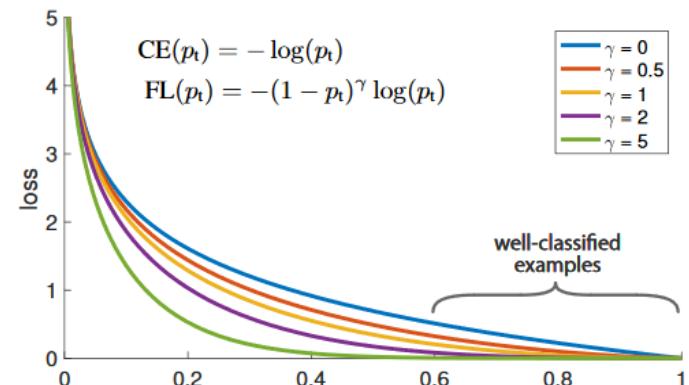
P = 0.1

$$\text{CE}(\text{foreground}) = -\log(0.1) = 2.3025$$

$$\text{CE}(\text{background}) = -\log(1-0.1) = 2.3025$$

$$\text{FL}(\text{foreground}) = -1 \times 0.25 \times (1-0.1)^{\gamma} \log(0.1) = 0.4667$$

$$\text{FL}(\text{background}) = -1 \times 0.25 \times (1-(1-0.1))^{\gamma} \log(1-0.1) = 0.4667$$



매우 쉽게 판별

P = 0.99

$$\text{CE}(\text{foreground}) = -\log(0.99) = 0.01$$

$$\text{CE}(\text{background}) = -\log(1-0.99) = 0.01$$

$$\text{FL}(\text{foreground}) = -1 \times 0.25 \times (1-0.99)^{\gamma} \log(0.99) = 0.00000025$$

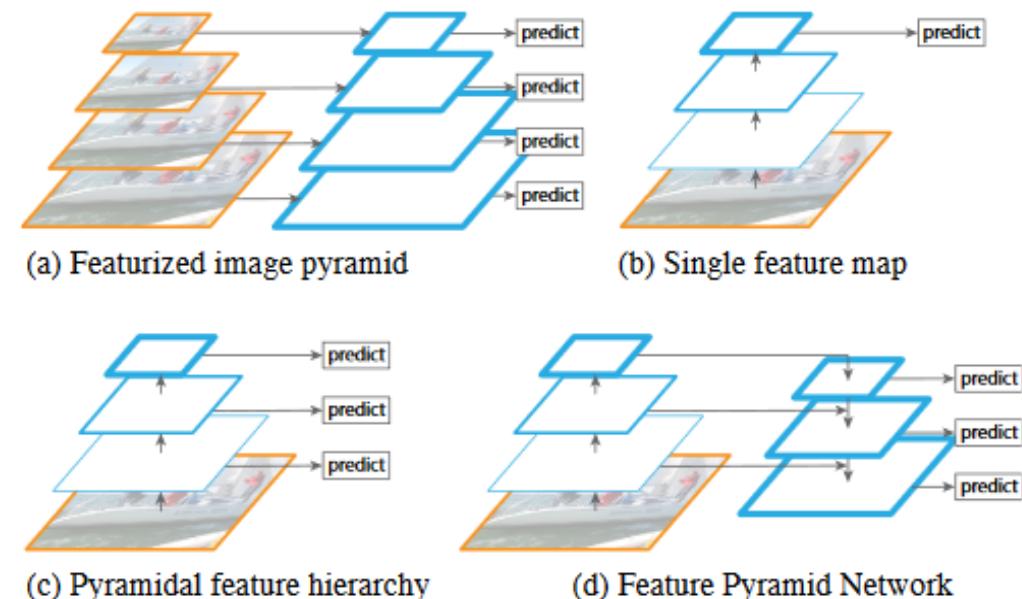
$$\text{FL}(\text{background}) = -1 \times 0.25 \times (1-(1-0.99))^{\gamma} \log(1-0.99) = 0.00000025$$

쉽게 판별:  $0.1/0.00026 = \text{CE} \text{ 대비 } 384\text{배 작은 loss}$

잘못 판별:  $2.3/0.4667 = \text{CE} \text{ 대비 } 5\text{배 작은 loss}$

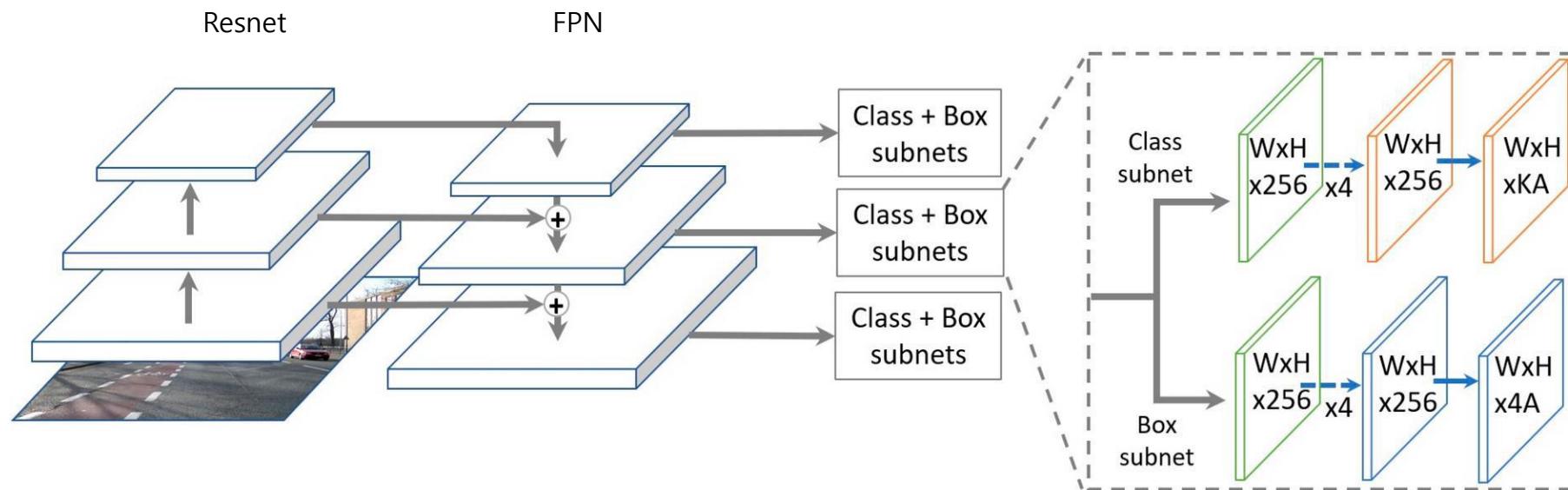
매우 쉽게 판별:  $0.01/0.00000025 = \text{CE} \text{ 대비 } 40,000\text{배 작은 loss}$

서로 다른 크기를 가지는 Object들을 효과적으로 Detection하기 위하여 bottom up과 top down방식을 추출된 feature map들을 lateral connection으로 연결하는 방식



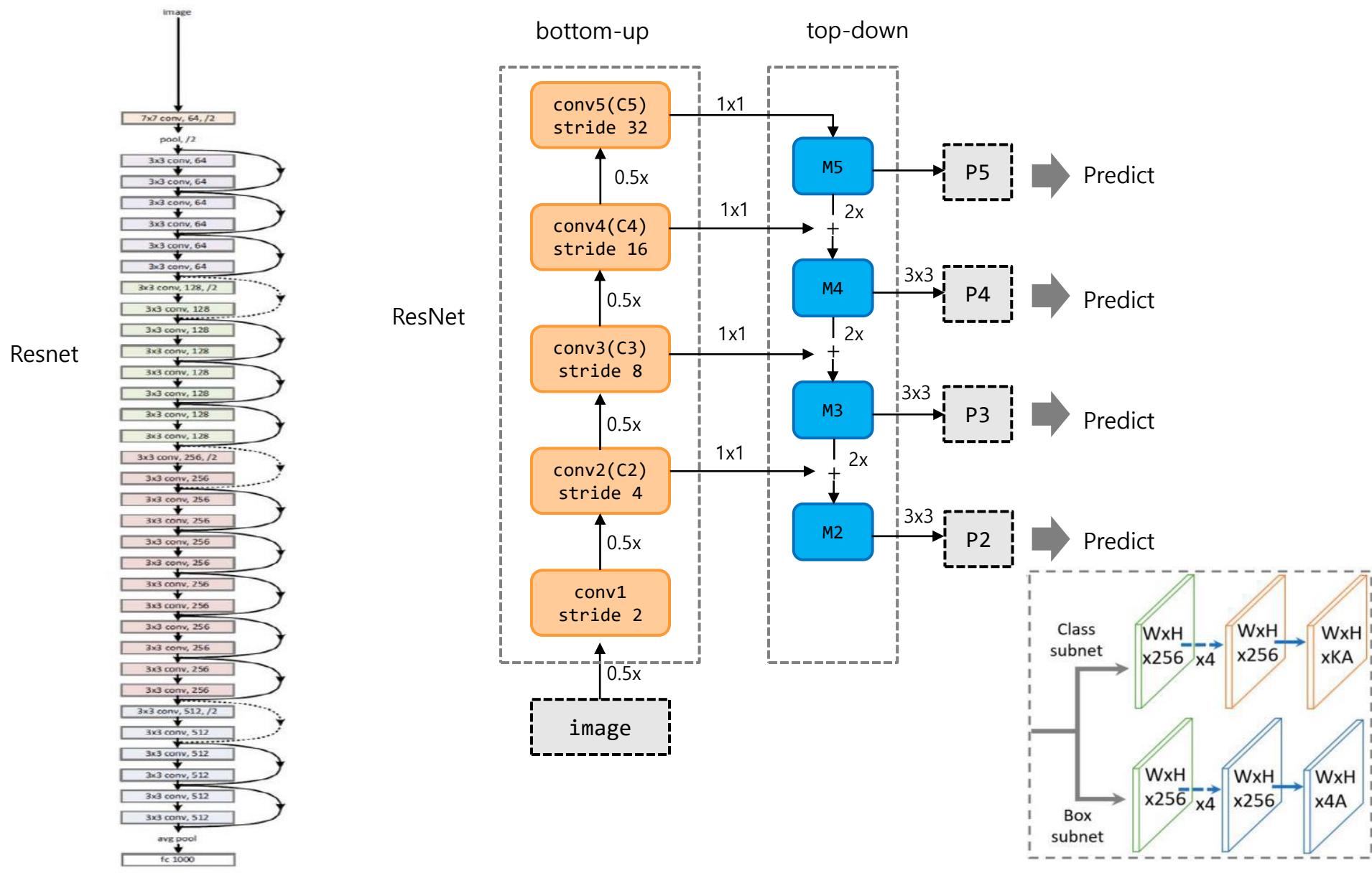
# Feature Pyramid Network 구조

1.6 RetinaNet



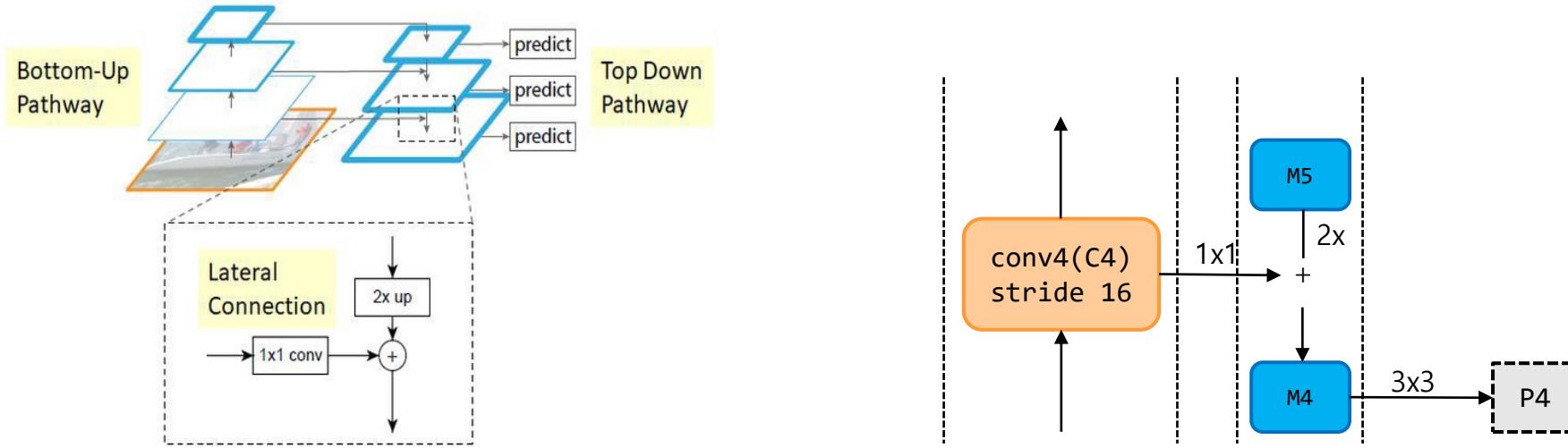
# Feature Pyramid Network 구조

## 1.6 RetinaNet



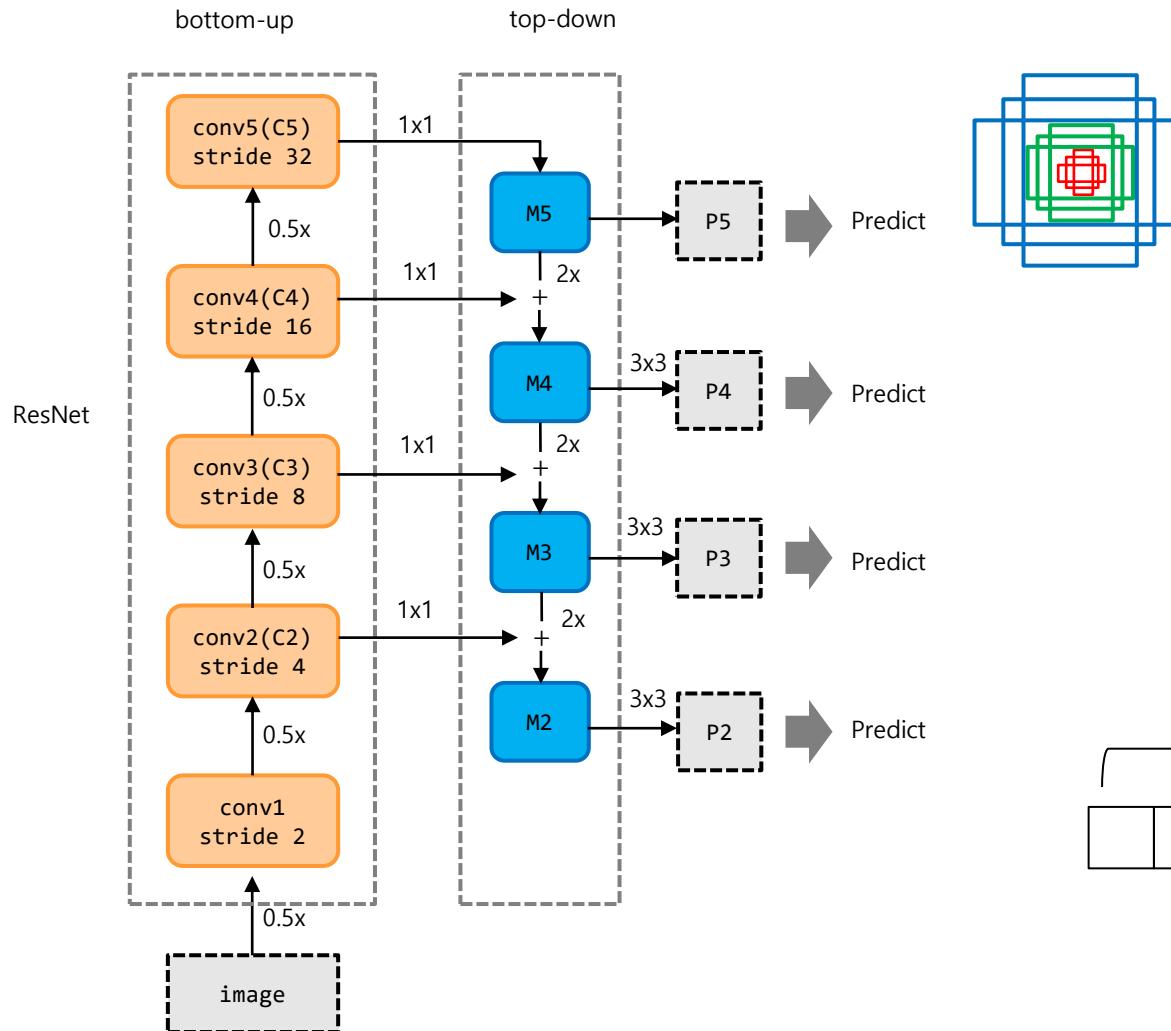
# Lateral Connection과 Top Down Merge

1.6 RetinaNet

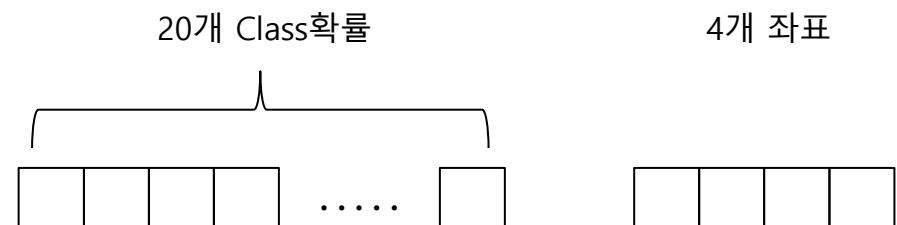


# RetinaNet FPN Anchor box

## 1.6 RetinaNet



- 9개의 anchor box가 P2~P5의 개별 Layer의 개별 grid에 할당
- 3개의 서로 다른 크기와 3개의 서로 다른 스케일을 가짐
- 약 100K의 anchor box들
- 개별 anchor box는 Classification을 위한 k개의 클래스 확률값과 Bounding box regression을 위한 4개 좌표 값을 가짐



# FPN 성능 비교 (Faster RCNN On FPN)

1.6 RetinaNet

method	backbone	competition	image pyramid	test-dev					test-std				
				AP@.5	AP	AP <sub>s</sub>	AP <sub>m</sub>	AP <sub>l</sub>	AP@.5	AP	AP <sub>s</sub>	AP <sub>m</sub>	AP <sub>l</sub>
ours, Faster R-CNN on FPN	ResNet-101	-		59.1	36.2	18.2	39.0	48.2	58.5	35.8	17.5	38.7	47.8
<i>Competition-winning single-model results follow:</i>													
G-RMI <sup>†</sup>	Inception-ResNet	2016	-	34.7	-	-	-	-	-	-	-	-	-
AttractioNet <sup>‡</sup> [10]	VGG16 + Wide ResNet <sup>§</sup>	2016	✓	53.4	35.7	15.6	38.0	52.7	52.9	35.3	14.7	37.6	51.9
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015	-	49.6	31.5	-	-	-	-	-	-	-	-
ION <sup>‡</sup> [2]	VGG-16	2015	-	53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

# EfficientDet - Scalable and Efficient Object Detection

1.6 RetinaNet



arXiv.org > cs > arXiv:1911.09070

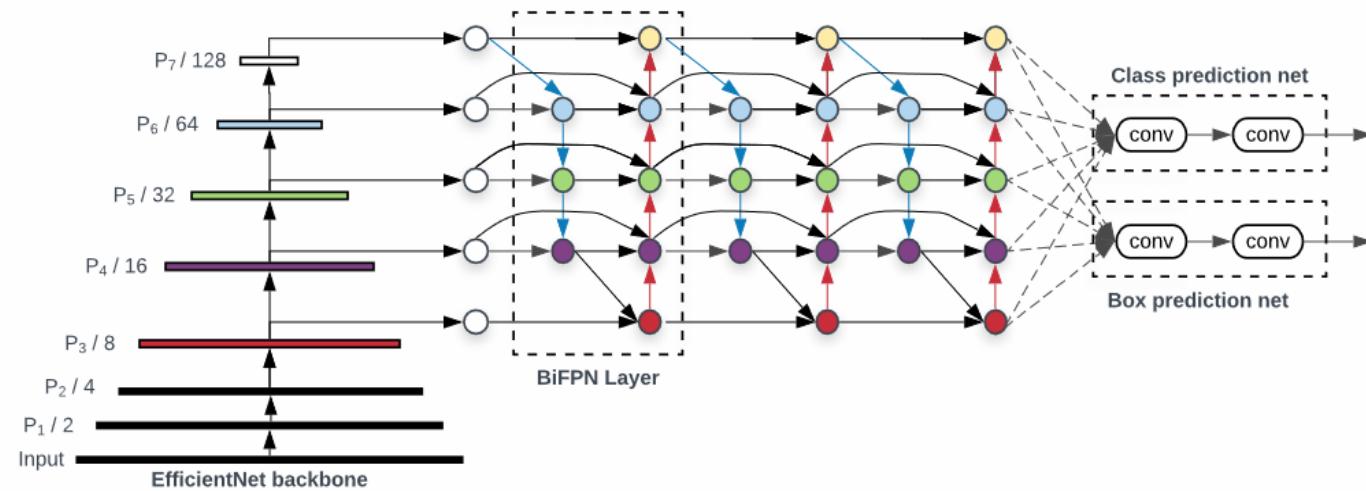
Computer Science > Computer Vision and Pattern Recognition

[Submitted on 20 Nov 2019 (v1), last revised 27 Jul 2020 (this version, v7)]

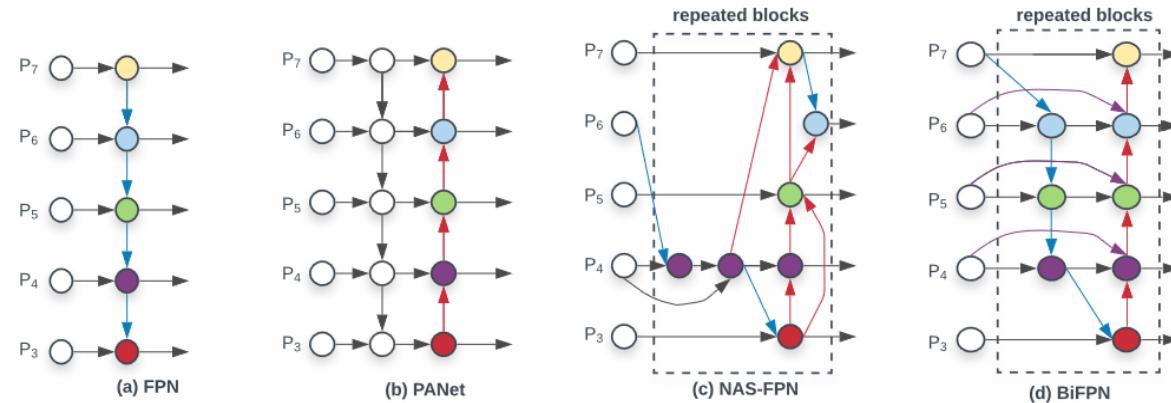
## EfficientDet: Scalable and Efficient Object Detection

Mingxing Tan, Ruoming Pang, Quoc V. Le

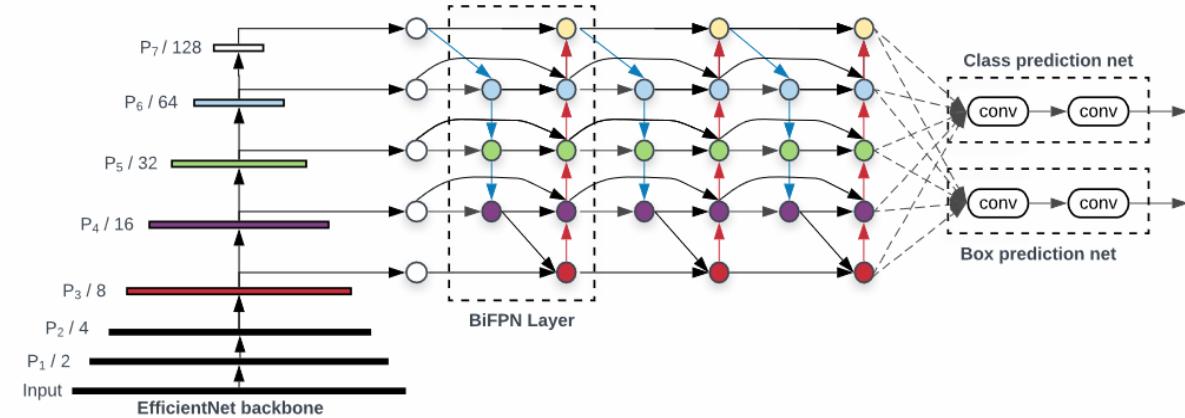
<https://arxiv.org/abs/1911.09070>

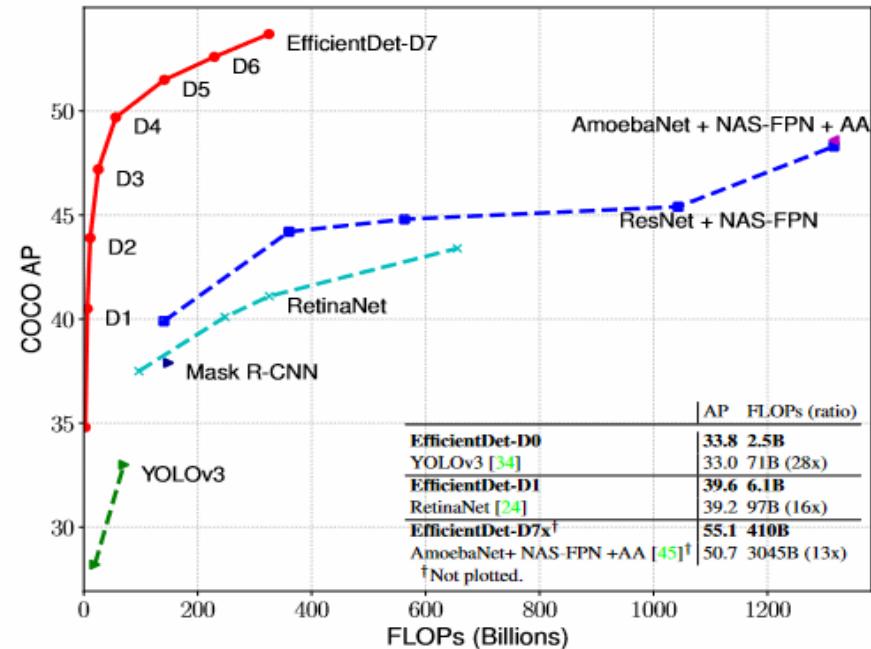


### BiFPN (bi-direction FPN)



Compound Scaling  
Backbone, BiFPN,  
Box/Classification Network

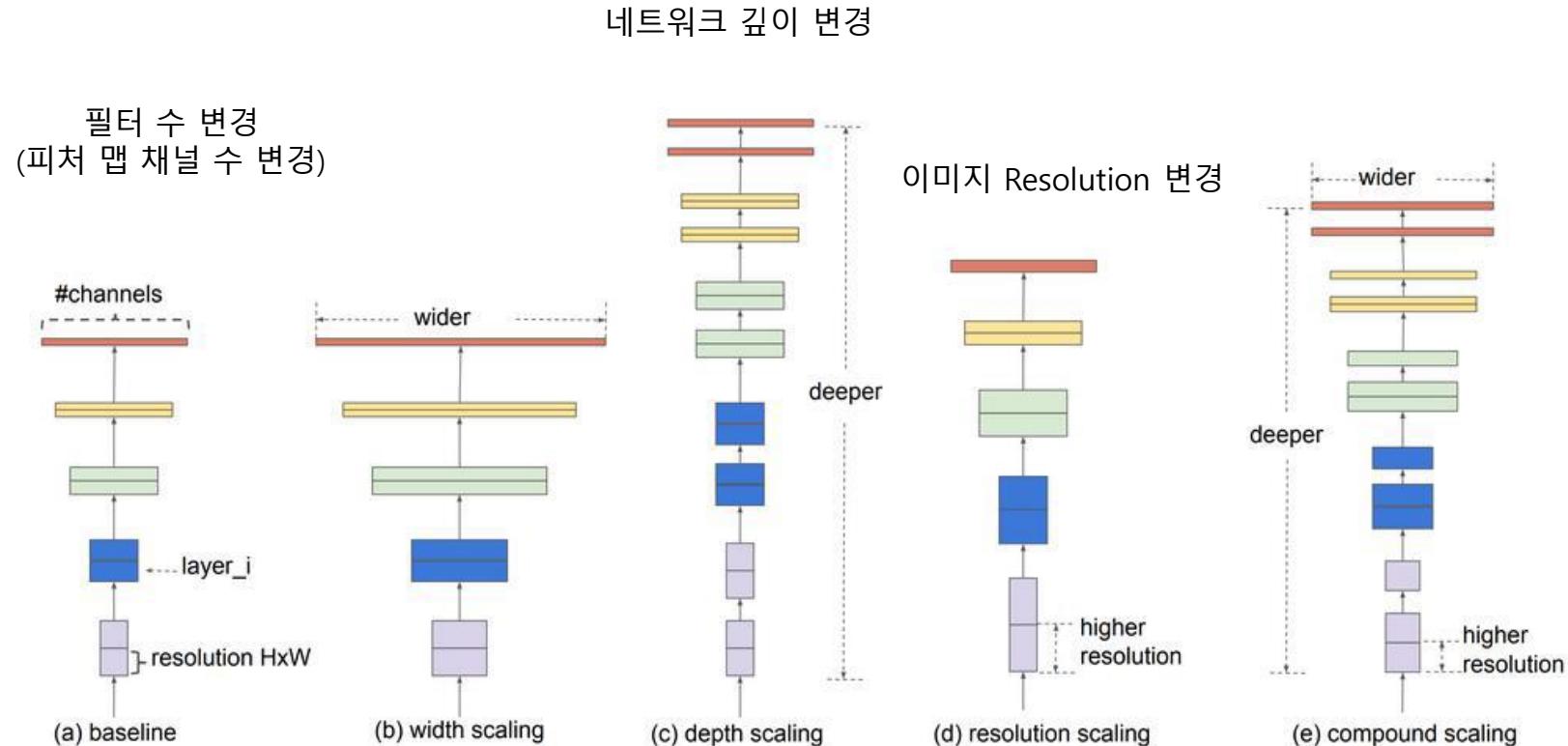




작은 연산 수, 적은 파라미터 수에 비해 상대적으로  
타 모델보다 높은 모델 예측 성능을 나타냄



네트워크의 깊이(Depth), 필터 수(Width), 이미지 Resolution 크기  
를 함께 최적으로 조합하여 모델 성능 극대화



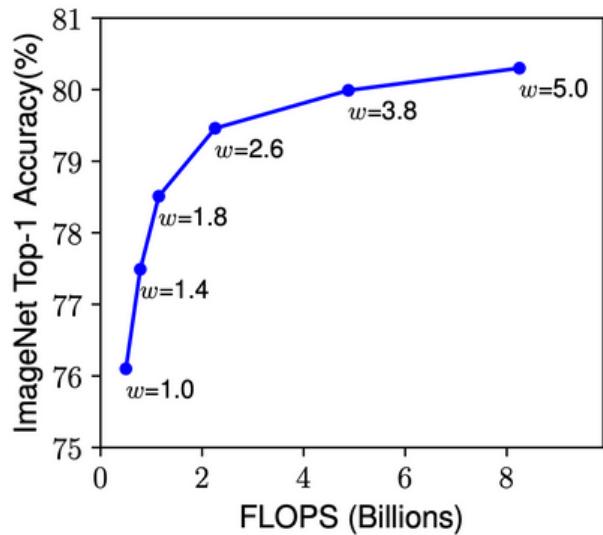
# EfficientNet - 개별 Scaling 요소에 따른 성능 향상 테스트

1.6 RetinaNet

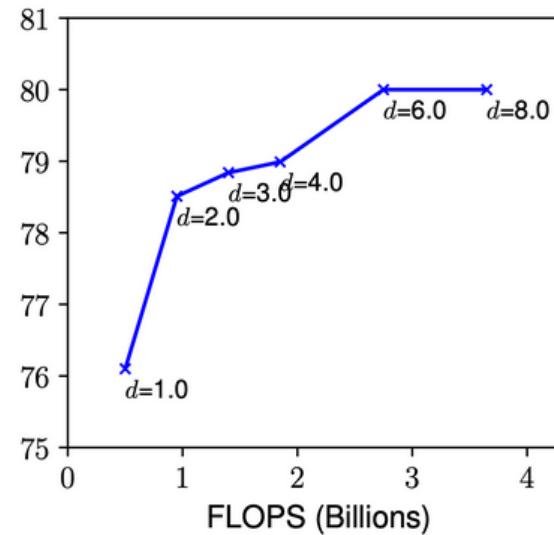
필터 수, 네트워크 깊이를 일정 수준 이상 늘려도 성능 향상 미비, 단 Resolution이 경우 어느 정도 약간 씩 성능 향상이 지속.

ImageNet 데이터세트 기준 80% 정확도에서 개별 Scaling 요소를 증가 시키더라도 성능 향상이 어려움

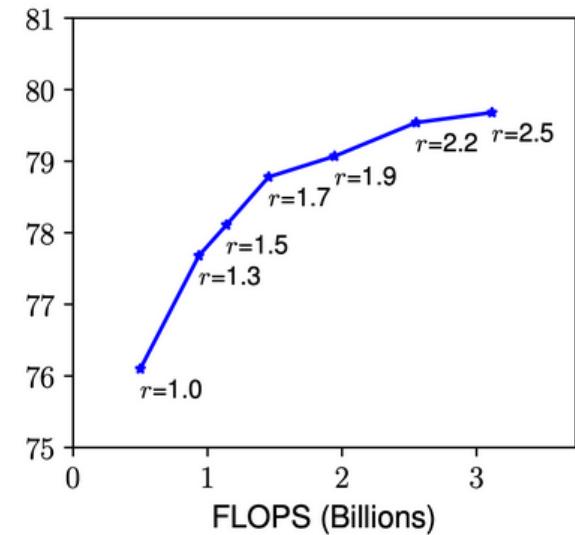
필터 수 변경



깊이 변경



이미지 Resolution 변경



### 최적 Scaling 도출 기반 식

$$\text{depth} : d = \alpha^\phi$$

$$\text{width} : w = \beta^\phi$$

$$\text{resolution} : r = \gamma^\phi$$

$$\alpha \cdot \beta^2 \cdot \gamma^2 \simeq 2 \quad (\alpha \geq 1, \beta \geq 1, \gamma \geq 1)$$

- 3가지 Scaling Factor를 동시 고려하는 Compound Scaling 적용
- 최초에는  $\varphi$  를 1로 고정하고 grid search 기반으로  $\alpha, \beta, \gamma$ 의 최적 값을 찾아냄. EfficientNetB0의 경우  $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$
- 다음으로  $\alpha, \beta, \gamma$  을 고정하고  $\varphi$ 를 증가 시키면서 EfficientB1~B7까지 Scale up 구성

depth, width, resolution에 따른 FLOPS 변화를

기반으로 최적 식 도출

width, resolution은 2배가 되면 FLOPS는 4배가

됨(그래서 제곱을 곱함)

Base model	resolution
EfficientNetB0	224
EfficientNetB1	240
EfficientNetB2	260
EfficientNetB3	300
EfficientNetB4	380
EfficientNetB5	456
EfficientNetB6	528
EfficientNetB7	600

EfficientNet   

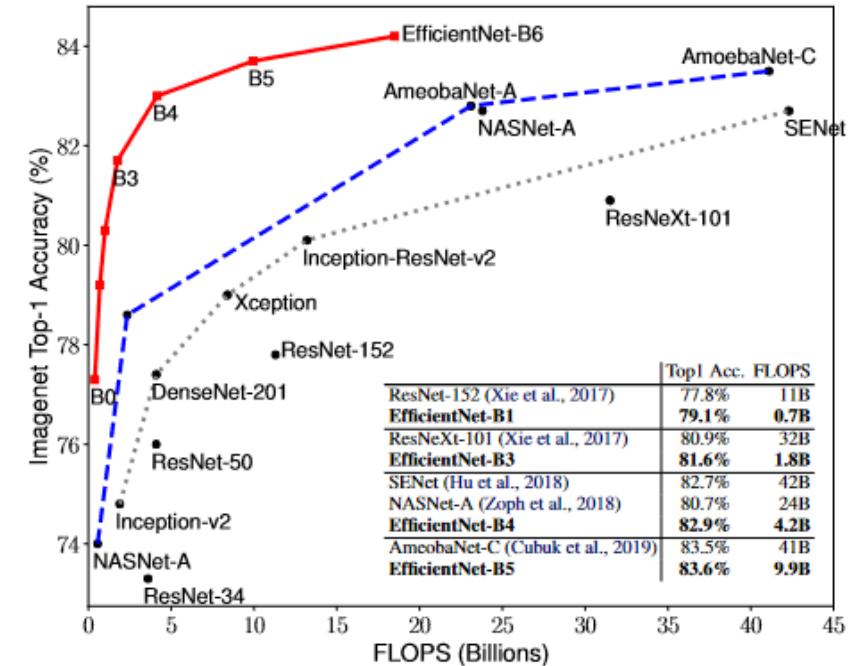
```
name: (channel_multiplier, depth_multiplier, resolution, dropout_rate)
'efficientnet-b0': (1.0, 1.0, 224, 0.2),
'efficientnet-b1': (1.0, 1.1, 240, 0.2),
'efficientnet-b2': (1.1, 1.2, 260, 0.3),
'efficientnet-b3': (1.2, 1.4, 300, 0.3),
'efficientnet-b4': (1.4, 1.8, 380, 0.4),
'efficientnet-b5': (1.6, 2.2, 456, 0.4),
'efficientnet-b6': (1.8, 2.6, 528, 0.5),
'efficientnet-b7': (2.0, 3.1, 600, 0.5),
'efficientnet-b8': (2.2, 3.6, 672, 0.5),
'efficientnet-l2': (4.3, 5.3, 800, 0.5),
```

# EfficientNet - 더 빠른 Inference 속도와 더 높은 예측 성능

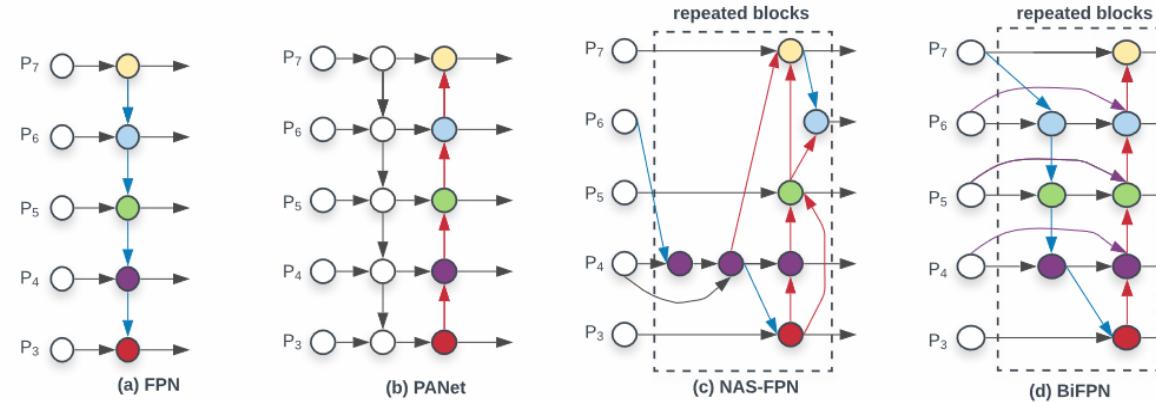
## 1.6 RetinaNet

Latency는 Intel Xeon CPU E5-2690 1 Core로 측정

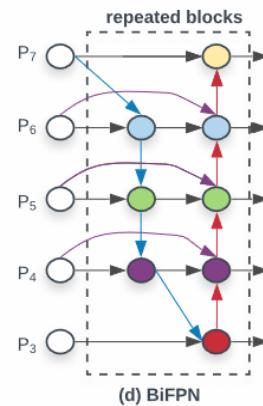
Acc. @Latency		Acc. @Latency	
ResNet-152	77.8% @ 0.554s	GPipe	84.3% @ 19.0s
EfficientNet-B1	78.8% @ 0.098s	EfficientNet-B7	84.4% @ 3.1s
<b>Speedup</b>	<b>5.7x</b>	<b>Speedup</b>	<b>6.1x</b>



### Cross Scale Connections

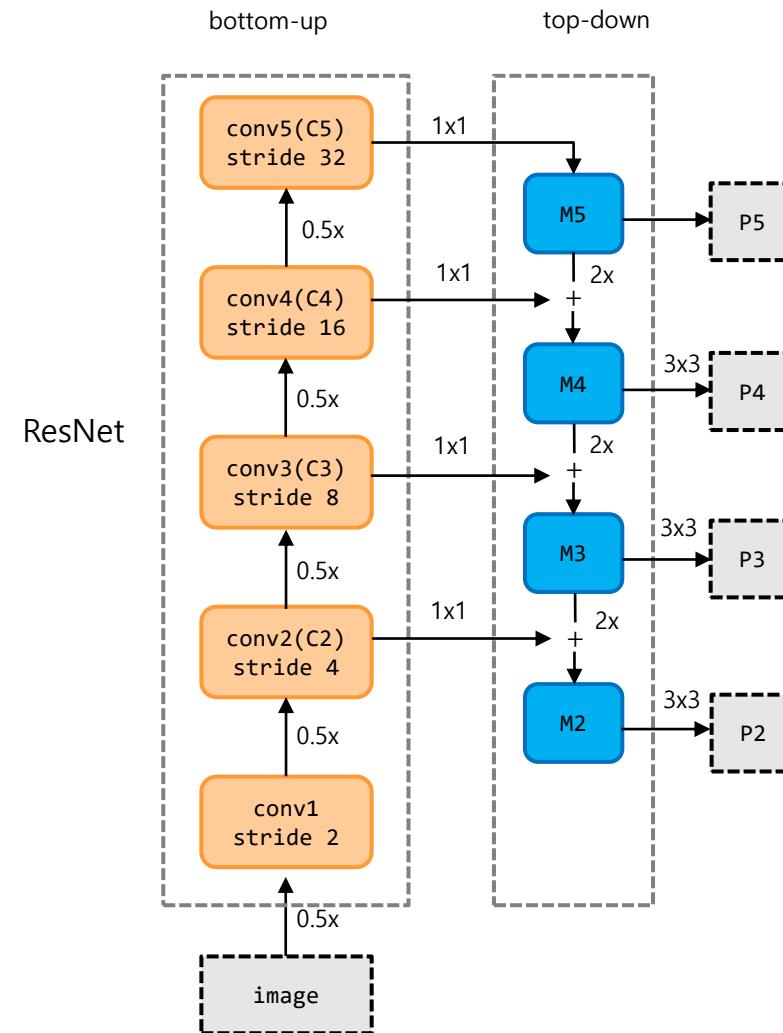
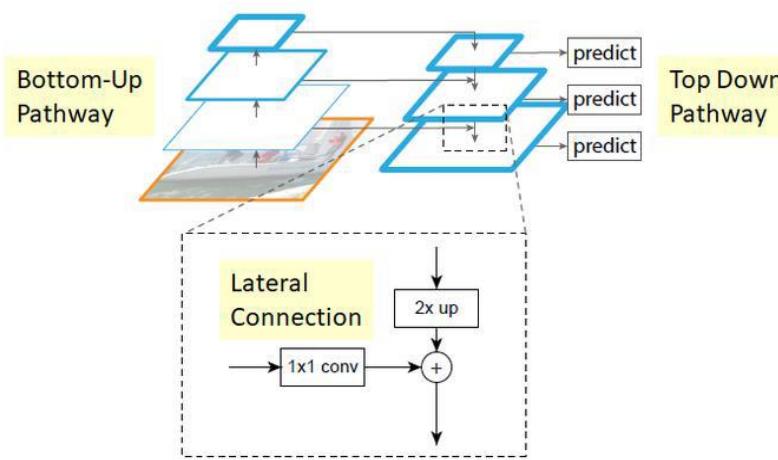


### Weighted Feature Fusion



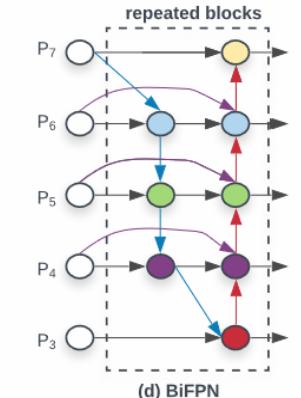
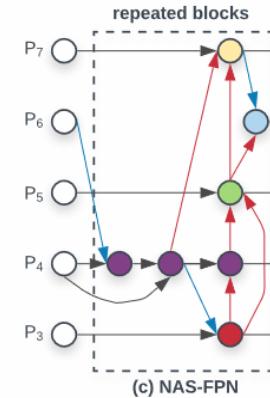
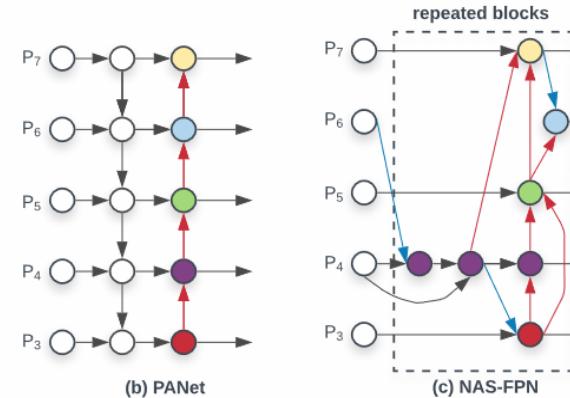
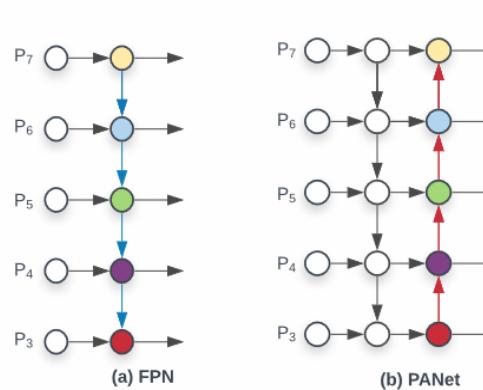
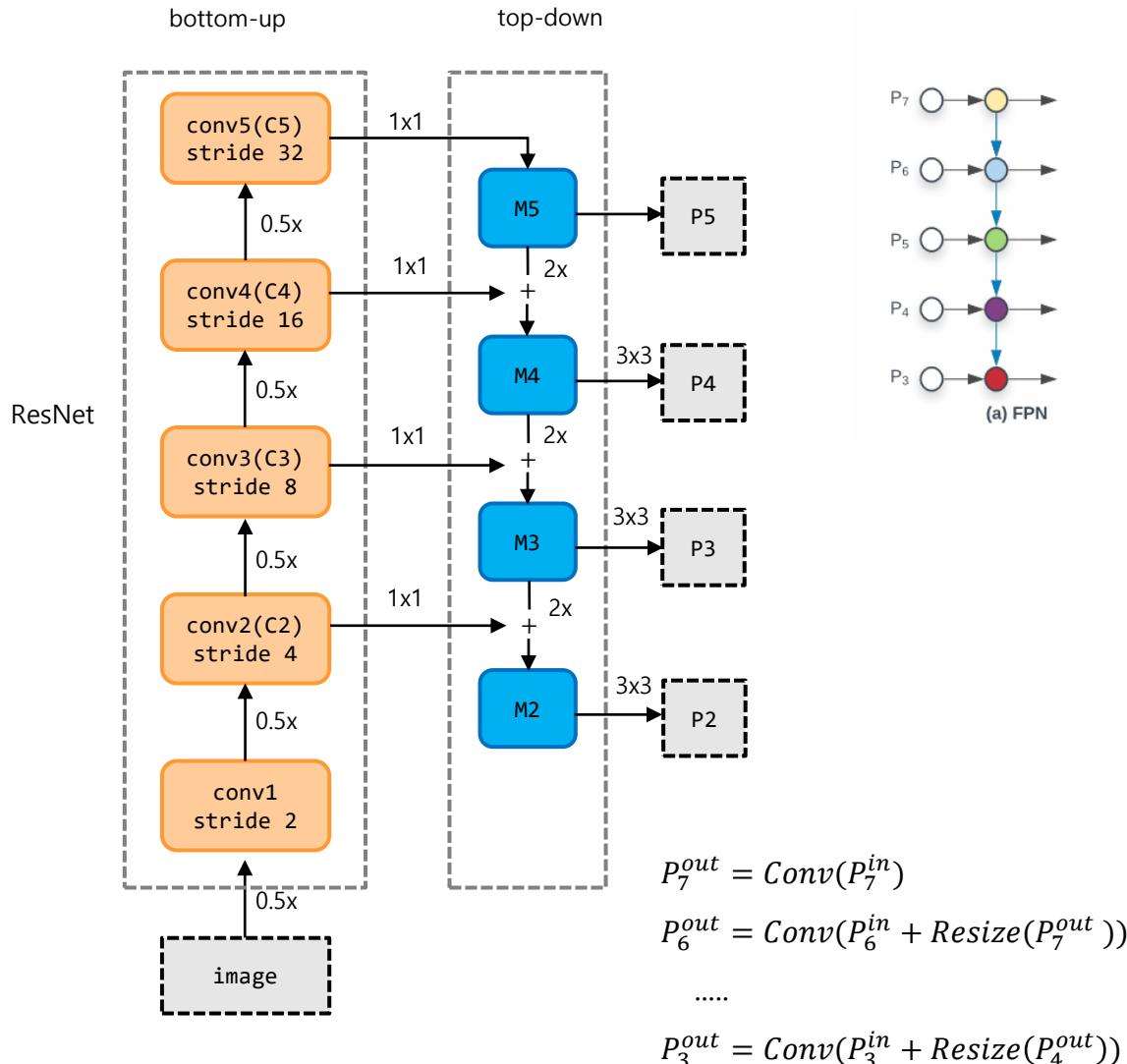
$$P_6^{td} = \text{Conv} \left( \frac{w_1 \cdot P_6^{in} + w_2 \cdot \text{Resize}(P_7^{in})}{w_1 + w_2 + \epsilon} \right)$$

$$P_6^{out} = \text{Conv} \left( \frac{w'_1 \cdot P_6^{in} + w'_2 \cdot P_6^{td} + w'_3 \cdot \text{Resize}(P_5^{out})}{w'_1 + w'_2 + w'_3 + \epsilon} \right)$$

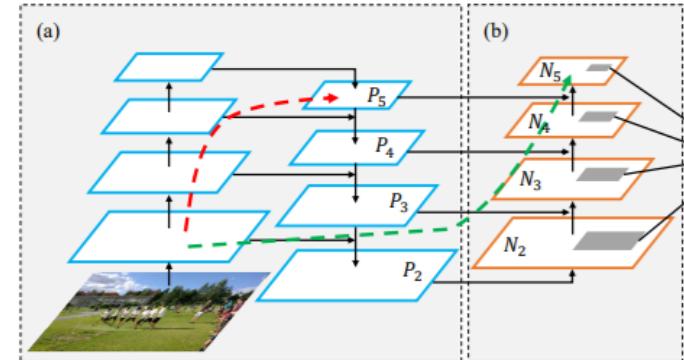


# BiFPN - Cross - Scale Connection

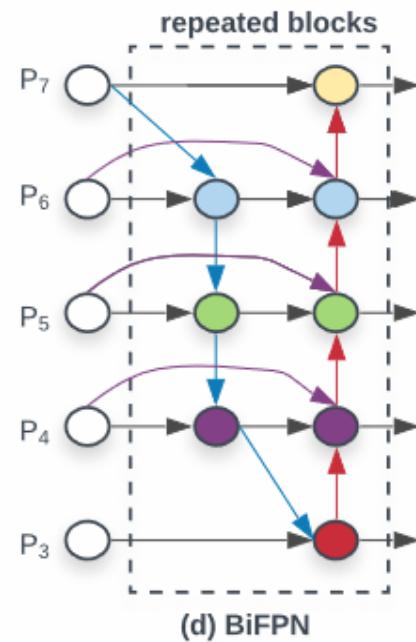
1.6 RetinaNet



PANet(Path Aggregation Network)for Instance Segmentation



서로 다른 resolution(feature map size)를 가지는 input feature map들은 Output feature map을 생성하는 **기여도**가 다르기 때문에 서로 다른 **가중치**를 부여하여 합쳐질 수 있어야 한다.



Fast normalized fusion:

$$P_6^{td} = \text{Conv}\left(\frac{w_1 \cdot P_6^{in} + w_2 \cdot \text{Resize}(P_7^{in})}{w_1 + w_2 + \epsilon}\right)$$

$$P_6^{out} = \text{Conv}\left(\frac{w'_1 \cdot P_6^{in} + w'_2 \cdot P_6^{td} + w'_3 \cdot \text{Resize}(P_5^{out})}{w'_1 + w'_2 + w'_3 + \epsilon}\right)$$

Unbounded fusion:  $0 = \sum_i w_i \cdot I_i$

입력 feature map에 가중치를 곱해서 출력 feature map 생성.

여기서 \$w\_i\$는 정해진 값이 아니라 **학습시켜서 도출된** weigh임.

Softmax-based fusion:  $0 = \sum_i \frac{e^{wi}}{\sum_j e^{wj}} \cdot I_i$

Fast normalized fusion:  $0 = \sum_i \frac{w_i}{\epsilon + \sum_j w_{-j}} \cdot I_i$

	Ap	Parameters	FLOPs
ResNet50+FPN	37.0	34M	97B
<b>EfficientNet-B3+FPN</b>	40.3	21M	75B
<b>EfficientNet-B3+BiFPN</b>	44.4	12M	24B

연산량 감소를 위해 BiFPN Network 구현시  
Separable Convolution을 적용

# EfficientDet Compound Scaling

1.6 RetinaNet

## backbone network

EfficientNet B0~B6로 Scaling 그대로 적용

## BiFPN network

Depth는 BiFPN 기본 반복 block을 3개 설정하고 Scaling 적용

$$D_{bifpn} = 3 + \emptyset$$

width(채널수)는 {1.2, 1.25, 1.3, 1.35, 1.4, 1.45}중 Grid Search를 통해서 1.35로 Scaling 계수를 선택하고 이를 기반으로 Scaling 적용

$$W_{bifpn} = 64 \cdot (1.35^\emptyset)$$

## Prediction Network

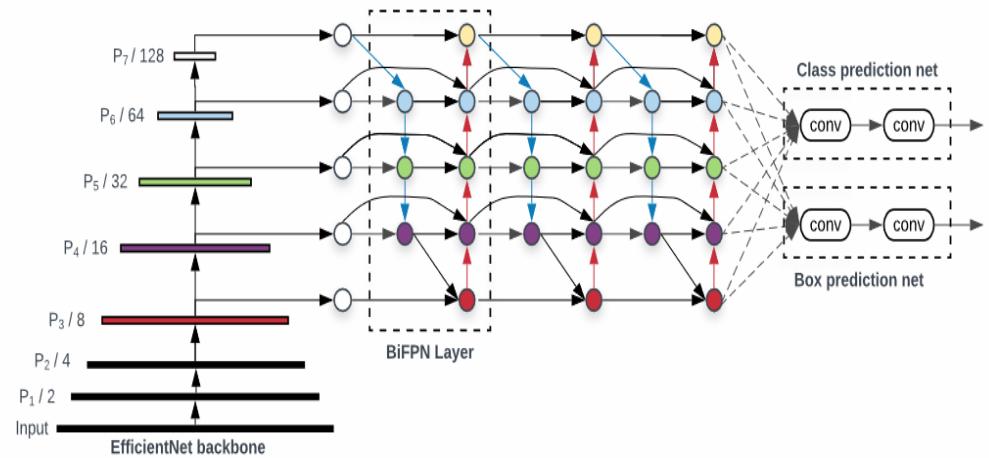
width(채널수)는 BiFPN 채널수와 동일

Depth는 아래식을 적용

$$D_{box} = D_{class} = 3 + [\emptyset/3]$$

## 입력 이미지 크기

$$R_{input} = 512 + \emptyset \cdot 128$$



	Input size $R_{input}$	Backbone Network	BiFPN #channels $W_{bifpn}$	BiFPN #layers $D_{bifpn}$	Box/class #layers $D_{class}$
D0 ( $\phi = 0$ )	512	B0	64	3	3
D1 ( $\phi = 1$ )	640	B1	88	4	3
D2 ( $\phi = 2$ )	768	B2	112	5	3
D3 ( $\phi = 3$ )	896	B3	160	6	4
D4 ( $\phi = 4$ )	1024	B4	224	7	4
D5 ( $\phi = 5$ )	1280	B5	288	7	4
D6 ( $\phi = 6$ )	1280	B6	384	8	5
D7 ( $\phi = 7$ )	1536	B6	384	8	5
D7x	1536	B7	384	8	5

### 기타 적용 요소

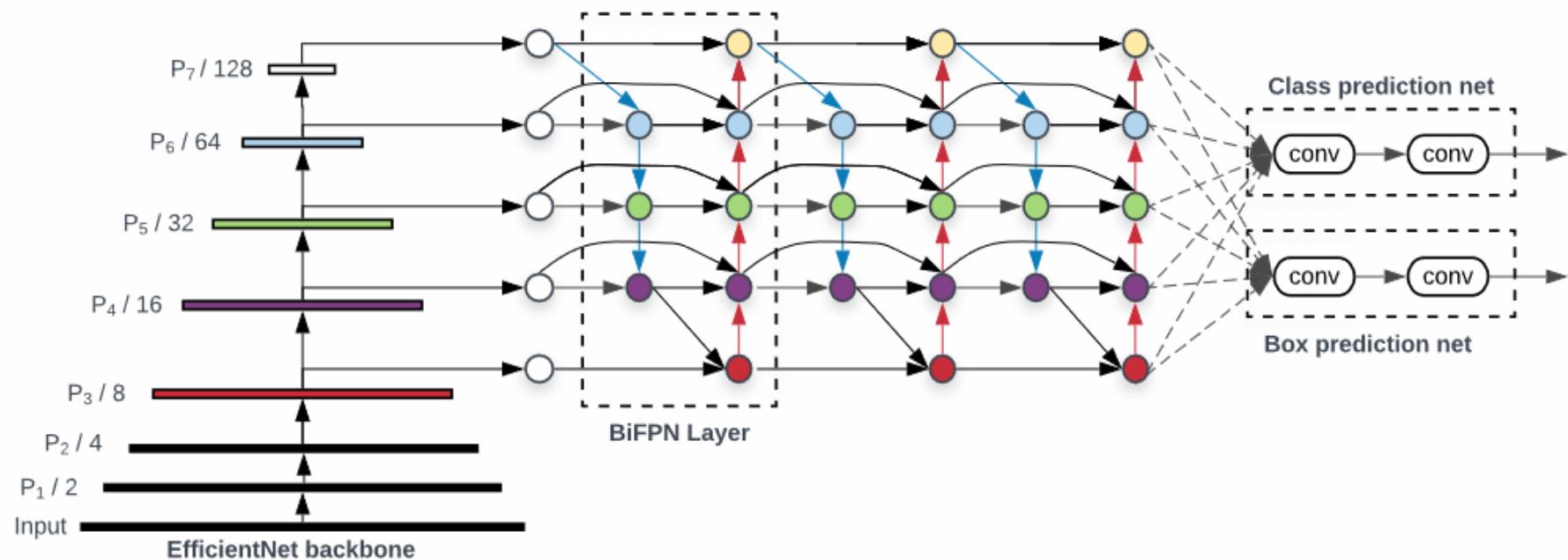
- activation : SiLU(SWISH)
- Loss : Focal Loss
- Augmentation : horizontal flip과 Scale jittering
- NMS : Soft NMS

Model	test-dev			val AP	Params	Ratio	FLOPs	Ratio	Latency (ms)	
	AP	AP <sub>50</sub>	AP <sub>75</sub>						TitianV	V100
<b>EfficientDet-D0 (512)</b>	<b>34.6</b>	<b>53.0</b>	<b>37.1</b>	<b>34.3</b>	<b>3.9M</b>	<b>1x</b>	<b>2.5B</b>	<b>1x</b>	<b>12</b>	<b>10.2</b>
YOLOv3 [34]	33.0	57.9	34.4	-	-	-	71B	28x	-	-
<b>EfficientDet-D1 (640)</b>	<b>40.5</b>	<b>59.1</b>	<b>43.7</b>	<b>40.2</b>	<b>6.6M</b>	<b>1x</b>	<b>6.1B</b>	<b>1x</b>	<b>16</b>	<b>13.5</b>
RetinaNet-R50 (640) [24]	39.2	58.0	42.3	39.2	34M	6.7x	97B	16x	25	-
RetinaNet-R101 (640)[24]	39.9	58.5	43.0	39.8	53M	8.0x	127B	21x	32	-
<b>EfficientDet-D2 (768)</b>	<b>43.9</b>	<b>62.7</b>	<b>47.6</b>	<b>43.5</b>	<b>8.1M</b>	<b>1x</b>	<b>11B</b>	<b>1x</b>	<b>23</b>	<b>17.7</b>
Detectron2 Mask R-CNN R101-FPN [1]	-	-	-	42.9	63M	7.7x	164B	15x	-	56 <sup>‡</sup>
Detectron2 Mask R-CNN X101-FPN [1]	-	-	-	44.3	107M	13x	277B	25x	-	103 <sup>‡</sup>
<b>EfficientDet-D3 (896)</b>	<b>47.2</b>	<b>65.9</b>	<b>51.2</b>	<b>46.8</b>	<b>12M</b>	<b>1x</b>	<b>25B</b>	<b>1x</b>	<b>37</b>	<b>29.0</b>
ResNet-50 + NAS-FPN (1024) [10]	44.2	-	-	-	60M	5.1x	360B	15x	64	-
ResNet-50 + NAS-FPN (1280) [10]	44.8	-	-	-	60M	5.1x	563B	23x	99	-
ResNet-50 + NAS-FPN (1280@384)[10]	45.4	-	-	-	104M	8.7x	1043B	42x	150	-
<b>EfficientDet-D4 (1024)</b>	<b>49.7</b>	<b>68.4</b>	<b>53.9</b>	<b>49.3</b>	<b>21M</b>	<b>1x</b>	<b>55B</b>	<b>1x</b>	<b>65</b>	<b>42.8</b>
AmoebaNet+ NAS-FPN +AA(1280)[45]	-	-	-	48.6	185M	8.8x	1317B	24x	246	-
<b>EfficientDet-D5 (1280)</b>	<b>51.5</b>	<b>70.5</b>	<b>56.1</b>	<b>51.3</b>	<b>34M</b>	<b>1x</b>	<b>135B</b>	<b>1x</b>	<b>128</b>	<b>72.5</b>
Detectron2 Mask R-CNN X152 [1]	-	-	-	50.2	-	-	-	-	-	234 <sup>‡</sup>
<b>EfficientDet-D6 (1280)</b>	<b>52.6</b>	<b>71.5</b>	<b>57.2</b>	<b>52.2</b>	<b>52M</b>	<b>1x</b>	<b>226B</b>	<b>1x</b>	<b>169</b>	<b>92.8</b>
AmoebaNet+ NAS-FPN +AA(1536)[45]	-	-	-	50.7	209M	4.0x	3045B	13x	489	-
<b>EfficientDet-D7 (1536)</b>	<b>53.7</b>	<b>72.4</b>	<b>58.4</b>	<b>53.4</b>	<b>52M</b>		<b>325B</b>		<b>232</b>	<b>122</b>
<b>EfficientDet-D7x (1536)</b>	<b>55.1</b>	<b>74.3</b>	<b>59.9</b>	<b>54.4</b>	<b>77M</b>		<b>410B</b>		<b>285</b>	<b>153</b>

너무 거대한 Backbone, 여러 겹의 FPN, 큰 입력 이미지의 크기 등의 개별적인 부분들에 집중하는 것은 비효율적.

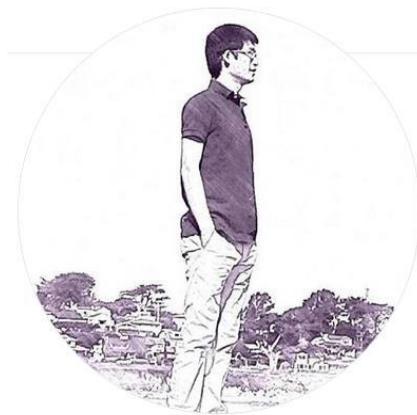
EfficientNet에서 개별 요소들을 함께 Scaling하면서 최적 결합을 통한 성능 향상을 보여줌.

EfficientDet에서도 Backbone, BiFPN, Prediction layer 그리고 입력이미지 크기를 Scaling 기반을 최적 결합하여 D0~D7 모델 구성



# AutoML EfficientDet 패키지 소개

1.6 RetinaNet



google / automl

Code Issues 84 Pull requests 6 Actions Projects 1 Wiki Security Insights

master ▾ 5 branches 3 tags Go to file Add file ▾ Code ▾

**mingxingtan** Update gpu figures. ✓ affe19b 7 days ago 652 commits

.github/workflows	Create python-app.yml (#854)	8 months ago
efficientdet	Sync.	15 days ago
efficientnetv2	Update gpu figures.	7 days ago
.gitignore	Gradient checkpointing libraries (#716)	9 months ago
.pylintrc	Add the example image and pylintrc.	14 months ago
CONTRIBUTING.md	Revert "Internal change"	15 months ago
LICENSE	Fix issues introduced in #55	15 months ago
README.md	Revert "Internal change"	15 months ago

- Efficientdet 구현 모델 중 가장 예측 성능이 뛰어난 모델 보유
- GPU와 TPU 모두에서 구동 가능
- Tensorflow 2의 Native Tensorflow와 tf.keras 모두로 구현
- 약간(?)의 버그(소스 코드 자체의 문제라기 보다는 tensorflow 자체의 문제로 보임)
- TFRECORD 형태로 입력 데이터 세트를 만들어야 하는 복잡함(?)



EfficientDet D0~D7

Tensorflow Object Detection API로 구현됨

EfficientDet - lite 0~4

AutoML EfficientDet으로 구현됨

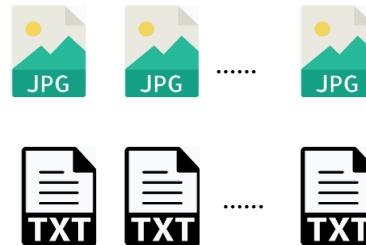
- Automl EfficientDet은 EfficientDet모델 위에 최신의 성능 향상 기법이 도입된 잘 짜여진 모듈로 구성
- config기반으로 다양한 환경 설정이 필요하지만 이를 위한 tutorial/document가 부족
- TFRECORD기반으로 학습/검증 데이터를 만들어야 함

- 매우 많은 갯수의 이미지, 오디오, 비디오, 텍스트들의 (주로 비정형)데이터를 보다 빠르게 Access하기 위해 만들어진 데이터 포맷
- 파일 시스템에 있는 여러 개의 데이터 파일들을 하나의 큰 **protobuf(protocol buffers)**포맷을 가지는 파일로 변경하여 Random IO acces를 줄여 Access성능을 향상 시킴
- Tensorflow에서는 CPU에서 GPU로의 데이터 전송 속도를 향상시켜 학습 속도를 빠르게 하기 위해서 도입.  
(하지만 병렬 I/O처리로도 충분히 GPU데이터 전송 속도를 향상시켜서 학습 속도를 빠르게 할 수 있음)
- 직관적이지 않은 인터페이스 API, 어려운 Debugging등의 문제를 가지고 있음

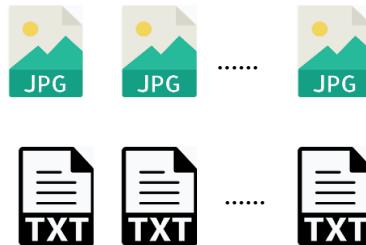
# TFRecord의 구성

## 1.6 RetinaNet

250개의 이미지 파일과  
Annotation파일



250개의 이미지 파일과  
Annotation파일



250개의 이미지 파일과  
Annotation파일



TFRecord #1

Example #1

```
encoded: 000111000000....1
format: jpg
x_min: [100, 10, 20, 30, 40, 50]
y_min: [30, 40, 100, 200, 250]
```

.....

Example #250

```
encoded: 011100011100000
format: jpg
x_min: [200, 100, 20]
y_min: [30, 20, 220]
```

TFRecord #2

Example #1

```
encoded: 000111000000....1
format: jpg
x_min: [100, 10, 20, 30, 40, 50]
y_min: [30, 40, 100, 200, 250]
```

.....

Example #250

```
encoded: 011100011100000
format: jpg
x_min: [200, 100, 20]
y_min: [30, 20, 220]
```

TFRecord #N

Example #2

```
encoded: 000111000000....1
format: jpg
x_min: [100, 10, 20, 30, 40, 50]
y_min: [30, 40, 100, 200, 250]
```

Example #250

```
encoded: 011100011100000
format: jpg
x_min: [200, 100, 20]
y_min: [30, 20, 220]
```

json, xml, avro 와 같이 데이터 전송과 저장(Serialization)을 위한 스키마 정보를 포함한 데이터 포맷



```
message Meal {
    required string name = 1;
    optional string alternate_name = 2;

    enum FoodType {
        INGREDIENT = 0;
        FILLING = 1;
        TOPPING = 2;
    }

    ** enum DrinkType {
        BUBBLY = 0;
        ALCOHOLIC = 1;
    }**

    message MealItems {
        required string item_name = 1;
        optional FoodType type = 2 [default = INGREDIENT];
    }

    ** message DrinkItems {
        required string drink_name = 1;
        optional DrinkType type = 2 ;
    }**

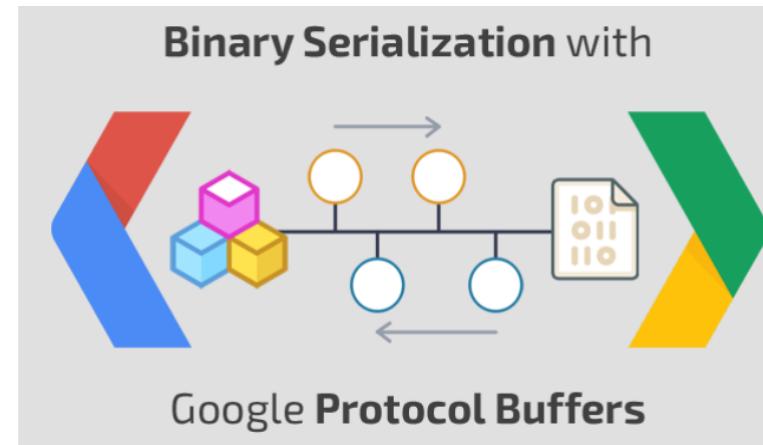
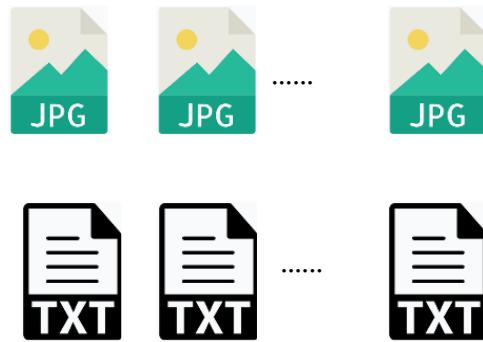
    repeated MealItems item = 4;
    ** repeated DrinkItems drink = 5;
}**}' < meals-2.txt
```

# Protobuf 기반으로 TFRecord 생성

1.6 RetinaNet

250개의 이미지 파일과

Annotation파일



TFRecord #1

Example #1

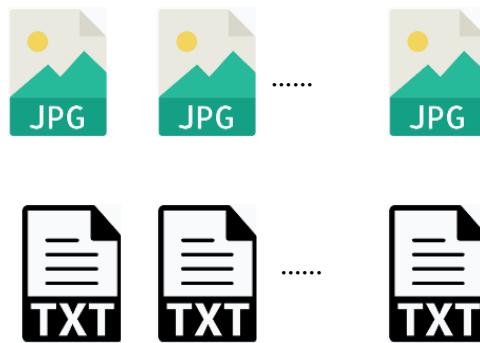
```
encoded: 000111000000....1  
format: jpg  
x_min: [100, 10, 20, 30, 40, 50]  
y_min: [30, 40, 100, 200, 250]
```

.....

Example #250

```
encoded: 011100011100000  
format: jpg  
x_min: [200, 100, 20]  
y_min: [30, 20, 220]
```

250개의 이미지 파일과  
Annotation 파일



250개의 이미지별로 개별 Example 을 encoding 지정하여 총 250개의 Example 을 하나의 TFRecord 로 생성.

```
example=tf.train.Example(  
    features=tf.train.Features(  
        feature={  
            'image/encoded':tfrecord_util.bytes_feature(example_dict['encoded']),  
            'image/format':tfrecord_util.bytes_feature('png'.encode('utf8')),  
            'image/object/bbox/x_min':tfrecord_util.float_list_feature(example_dict['xmin']),  
            'image/object/bbox/y_min':tfrecord_util.float_list_feature(example_dict['ymin'])  
        })
```

```
tf.io.TFRecordWriter('train-001.tfrecord').write(example.SerializeToString())
```

TFRecord #1

Example #1

```
encoded: 000111000000...1  
format: jpg  
x_min: [100, 10, 20, 30, 40, 50]  
y_min: [30, 40, 100, 200, 250]
```

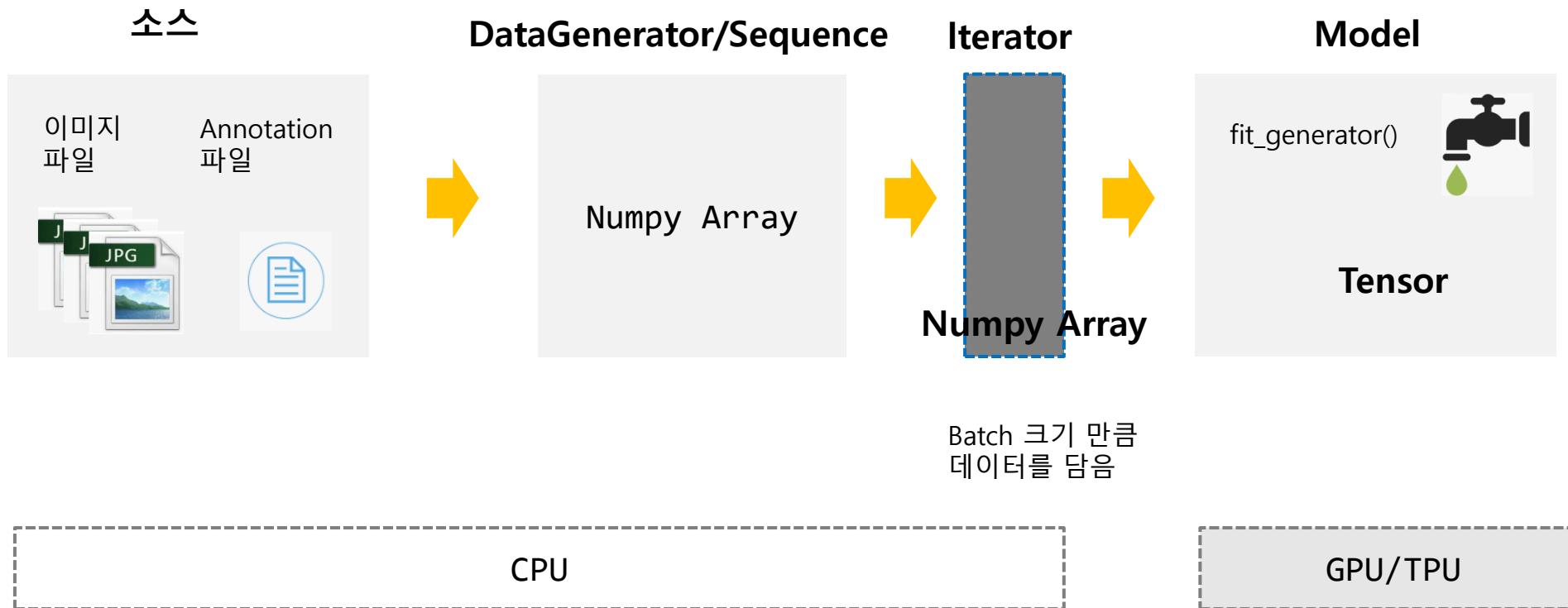
.....

Example #250

```
encoded: 01100011100000  
format: jpg  
x_min: [200, 100, 20]  
y_min: [30, 20, 220]
```

# 일반 파일 기반의 딥러닝 모델 Data Pipeline

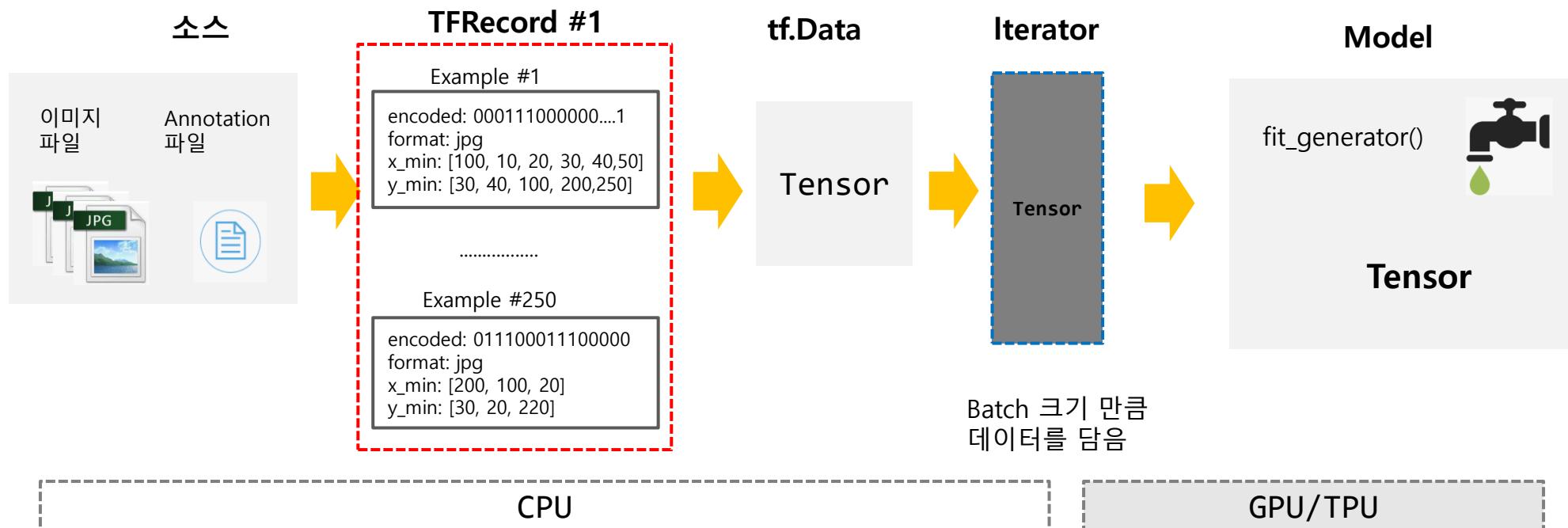
1.6 RetinaNet



CPU에서 GPU로 학습 데이터 전달 속도가 원활하지 않아서 GPU Utilization이 떨어지고, 학습시간이 오래 걸리는 현상이 발생할 수 있음.

# TFRecord기반의 딥러닝 모델 Data Pipeline

1.6 RetinaNet



- TFRecord를 이용하여 파일 I/O 시간을 많이 줄여줄 수 있으므로 더 많은 데이터를 GPU로 전달하여 GPU Utilization을 높이고 학습 수행속도 향상
- 일반 파일기반에서도 병렬로 I/O를 CPU로 로딩 후 GPU로 전달하면 GPU Utilization이 많이 향상되어 TFRecord기반의 학습시간과 별 차이가 없음.
- 하지만 **TPU**의 경우는 보다 높은 Bandwidth로 데이터 처리가 가능하므로 TFRecord를 이용하여 효율적인 학습시간 단축이 가능

## 2. 세그멘테이션

---

### 2.1 세그멘테이션 개요

2.2 FCN

2.3 Mask RCNN

2.4 SAM(Segment Anything Model)

---

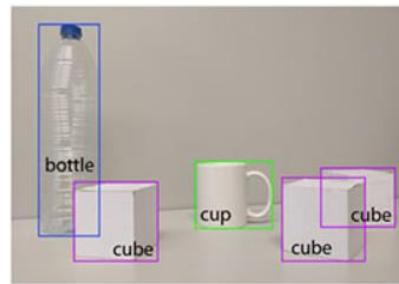
# Object Detection과 Segmentation

## 2.1 세그멘테이션 개요

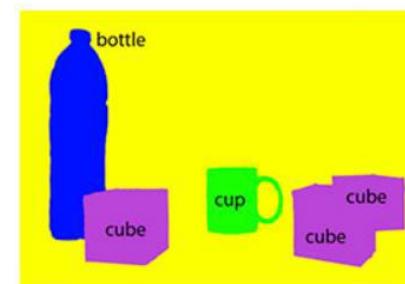
Image Classification



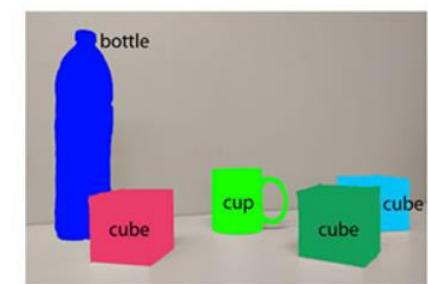
Object Detection



Semantic Segmentation



Instance Segmentation





# 이미지의 개별 pixel별 Classification

## 2.1 세그멘테이션 개요



Input

segmented →

- 1:Person
- 2:Purse
- 3:Plants/Grass
- 4:sidewalk
- 5:Building/Structures

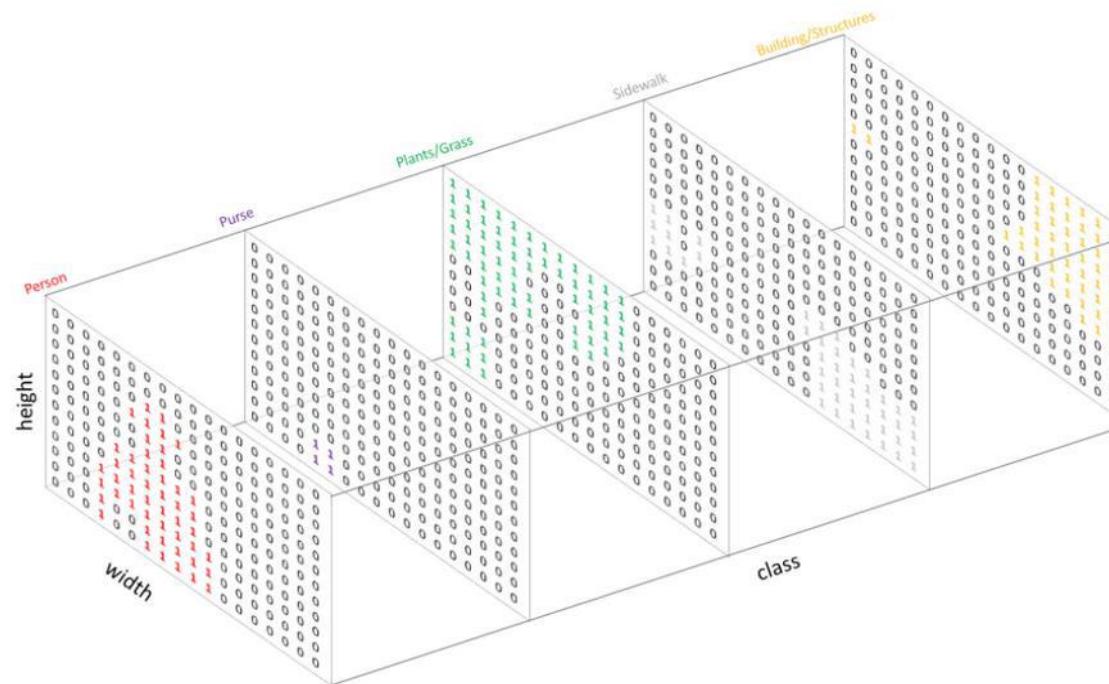
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	3	3	1	1	1	3	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	3	1	1	1	1	3	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	3	3	1	1	1	3	3	3	3	3	3	5	5	5	5	5
5	5	3	3	3	3	3	1	1	1	3	3	3	3	3	3	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	1	4	4	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	1	4	4	4	4	4	4	5	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4	4	4

Semantic Labels

참조 <https://medium.com/hyunjulie/1편-semantic-segmentation-첫걸음4180367ec9cb>

# 이미지의 개별 Pixel 별 Classification

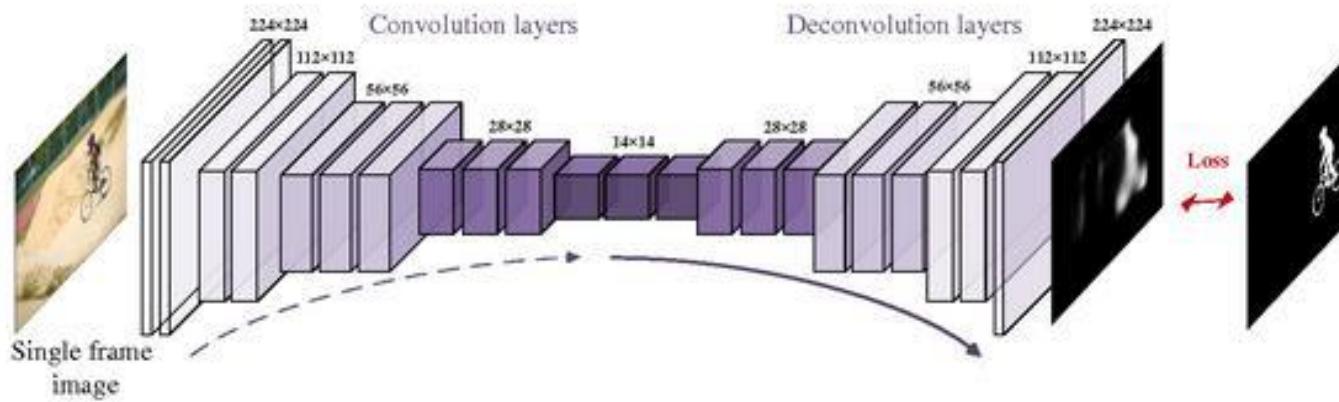
## 2.1 세그멘테이션 개요



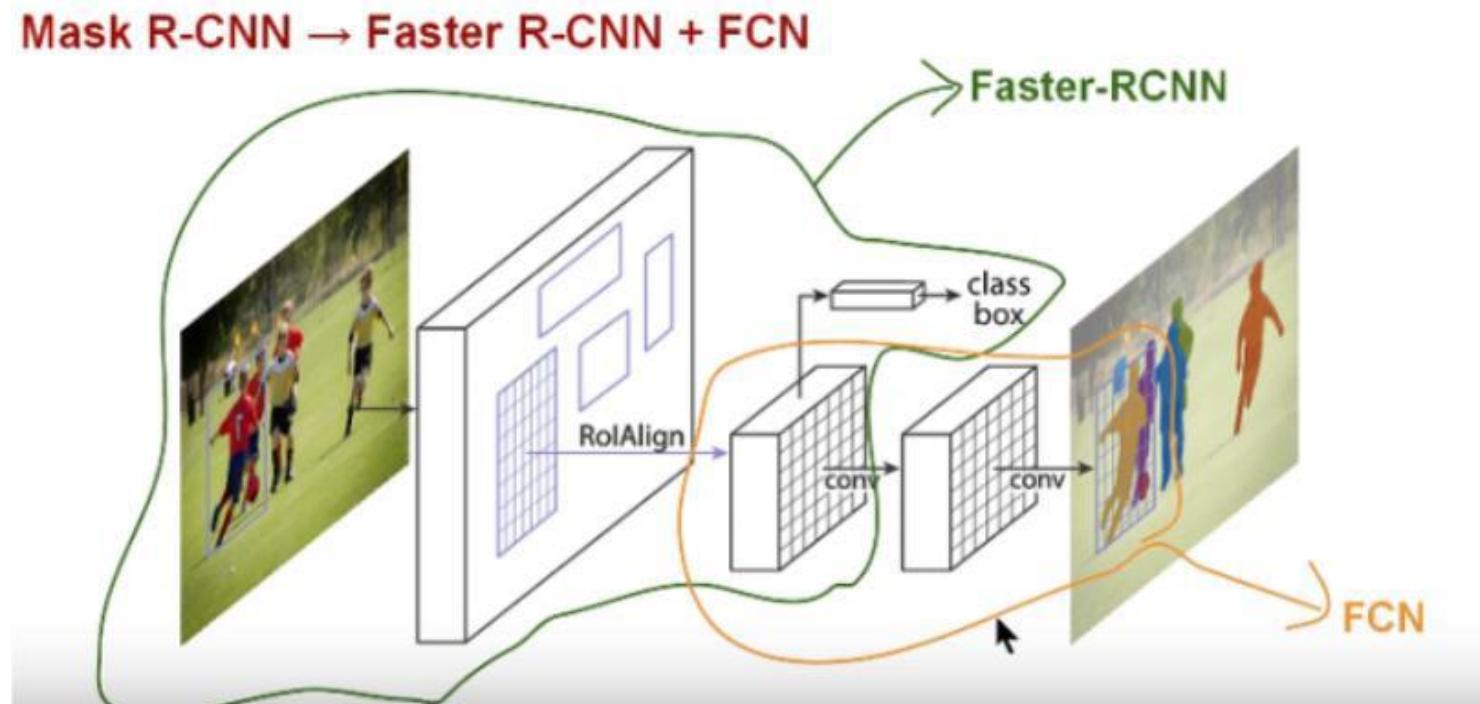
참조 <https://medium.com/hyunjulie/1편-semantic-segmentation-첫걸음4180367ec9cb>

# Semantic Segmentation Encoder-Decoder Model

## 2.1 세그멘테이션 개요



FCN, Segnet, U-NET, Dilated FCN



## 2. 세그멘테이션

---

2.1 세그멘테이션 개요

### 2.2 FCN

2.3 Mask RCNN

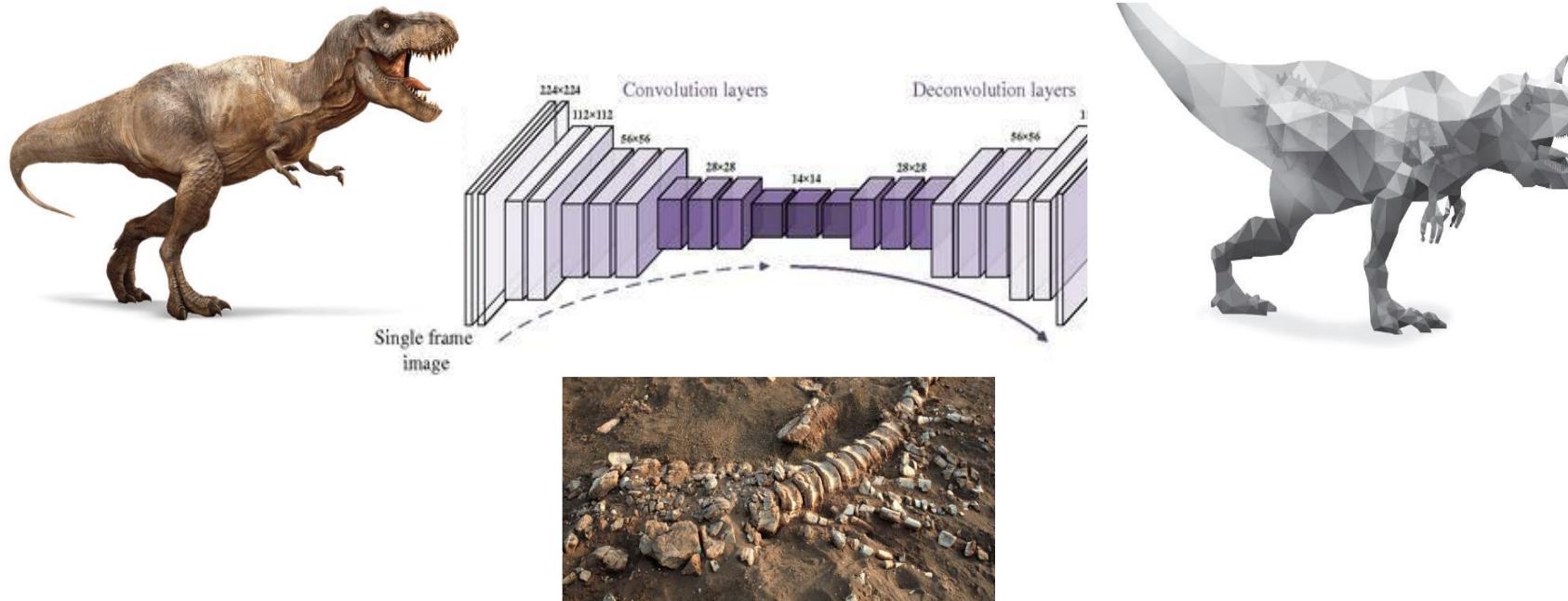
2.4 SAM(Segment Anything Model)

---

# Semantic Segmentation Encoder-Decoder Model

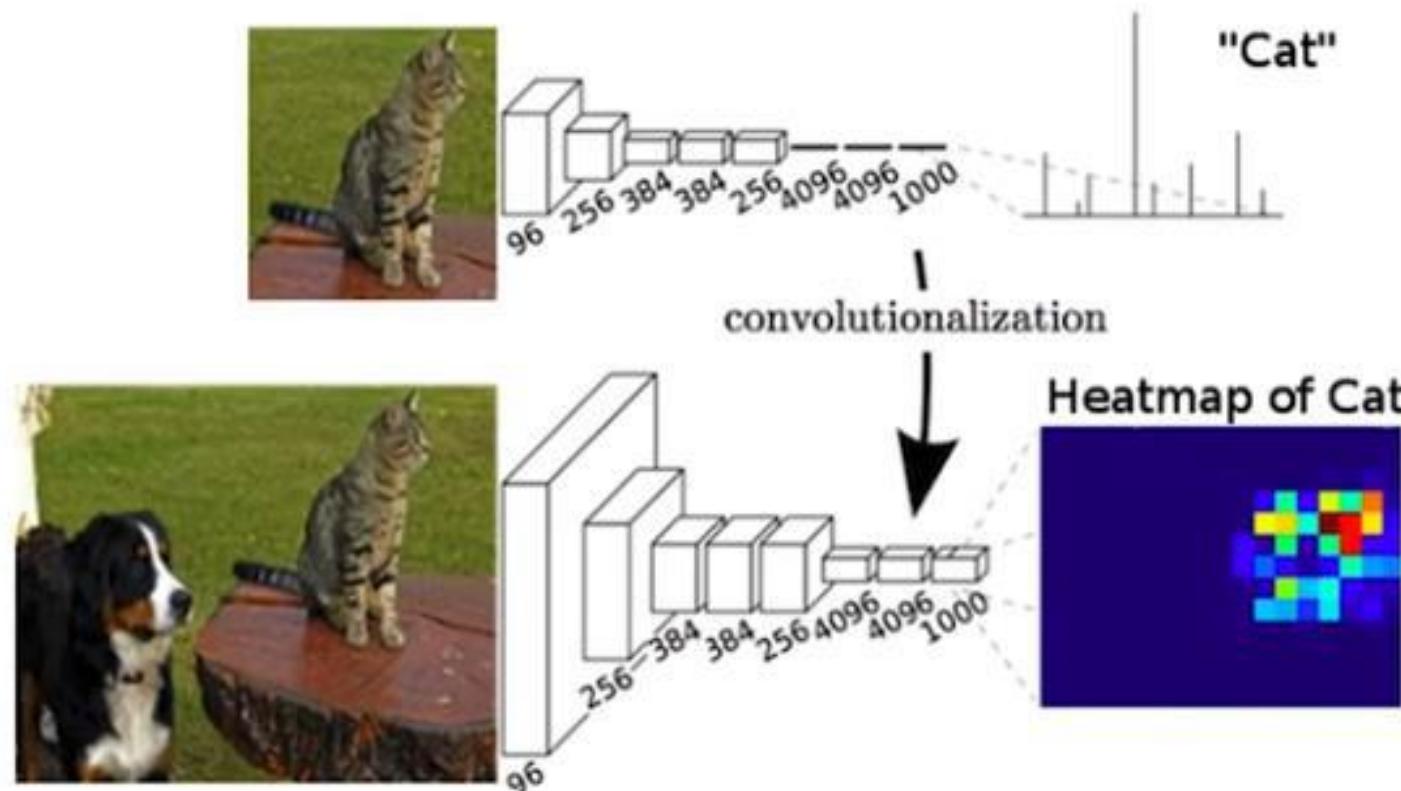
## 2.2 FCN(Fully Convolution Networks)

- 원본이미지를 Convolution으로 차원축소(Dimension Reduction)하여 응축된 정보를 가지고, 이를 다시 복원하면서 필요한 정보를 학습
- 이렇게 학습된 정보를 기반으로 Segmentation 수행



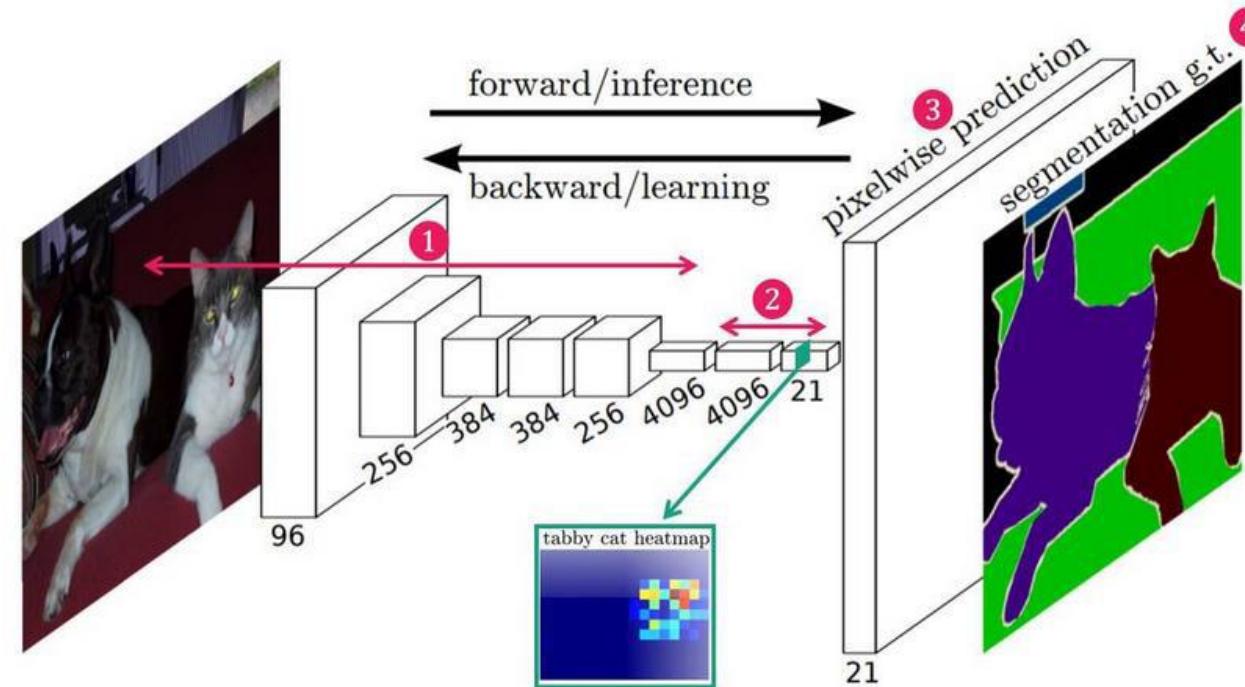
# Fully Connected Layer vs Fully Convolutional layer

## 2.2 FCN(Fully Convolution Networks)



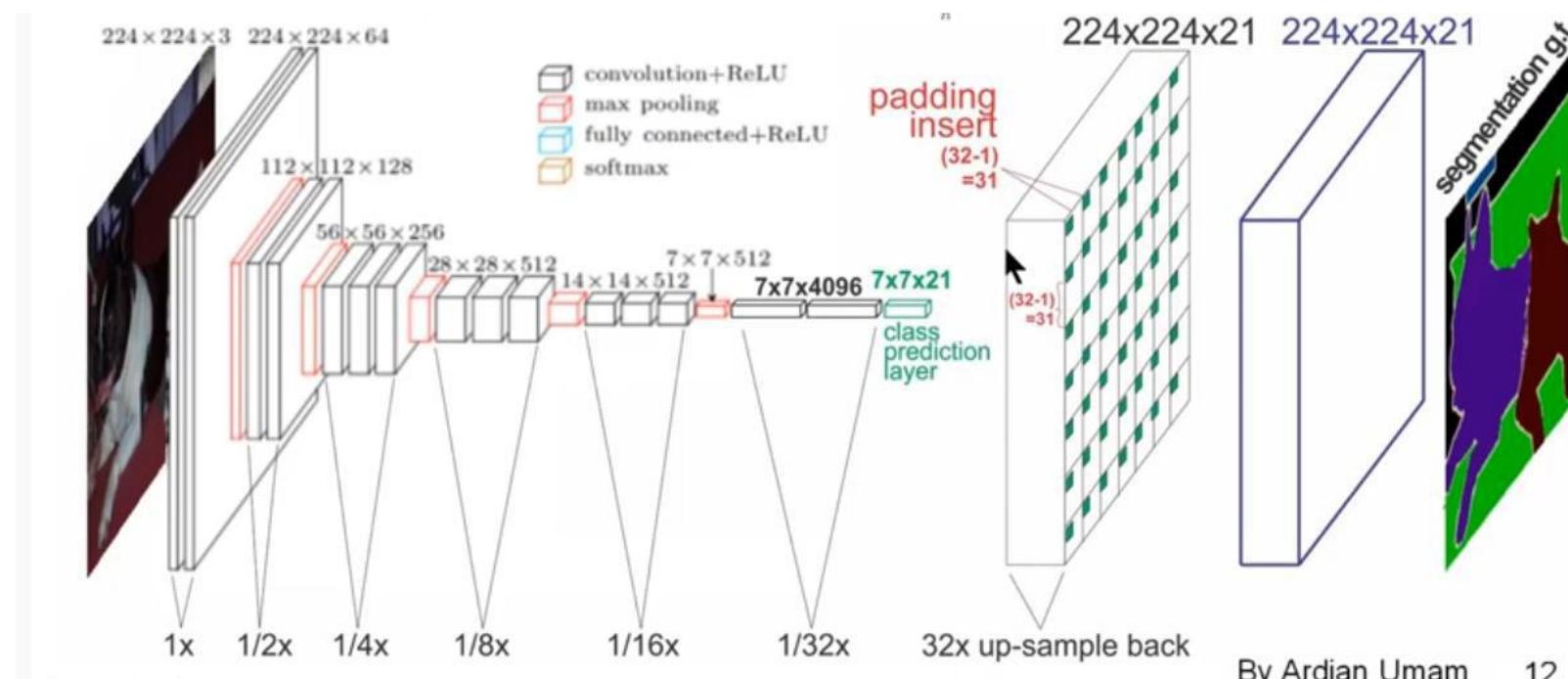
# FCN(Fully Convolution Network for Semantic Segmentation)

## 2.2 FCN(Fully Convolution Networks)



# FCN Down sampling과 Upsampling

## 2.2 FCN(Fully Convolution Networks)

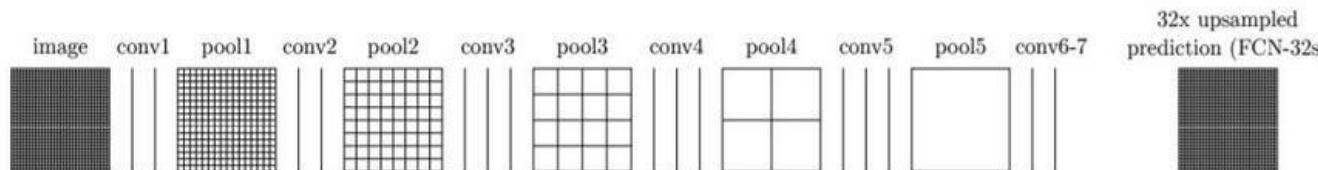


Bv Ardian Umam

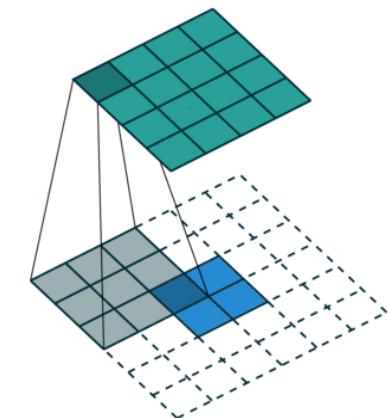
12

# 32x Upsampling을 통한 Pixel wise prediction

## 2.2 FCN(Fully Convolution Networks)

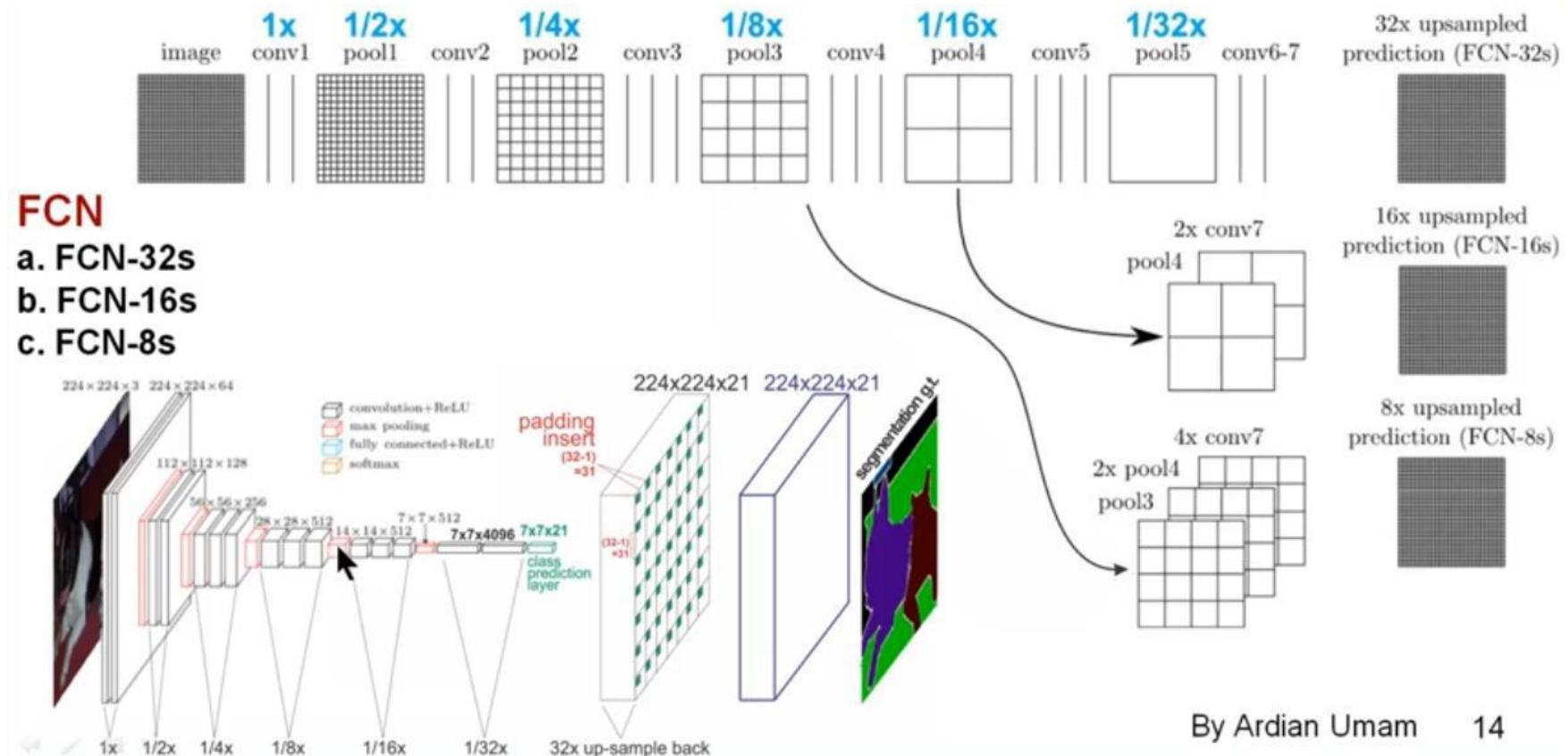


De-Convolution을 통한  
Upsampling



# Feature Map을 혼합하여 Pixel Wise Prediction

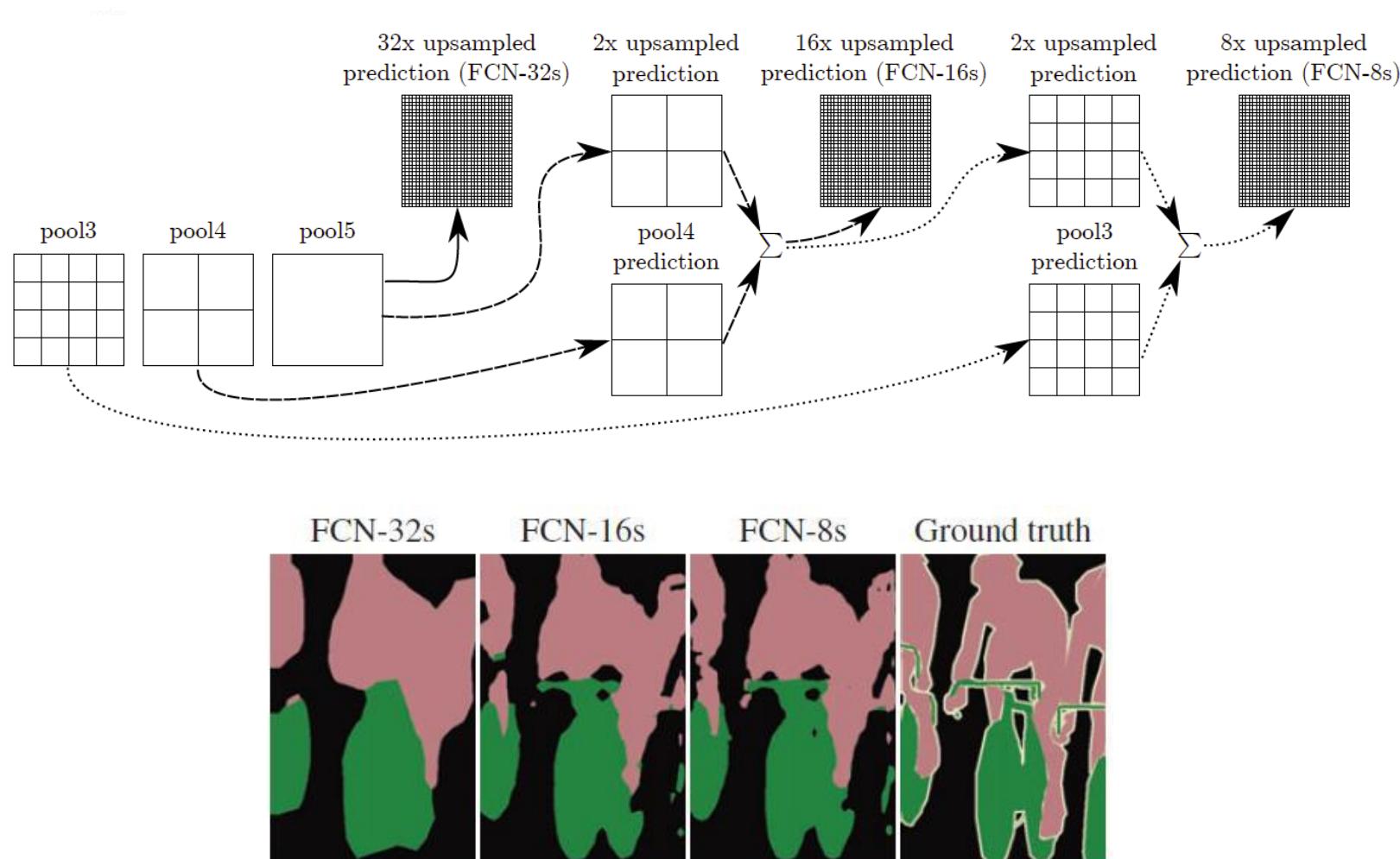
## 2.2 FCN(Fully Convolution Networks)

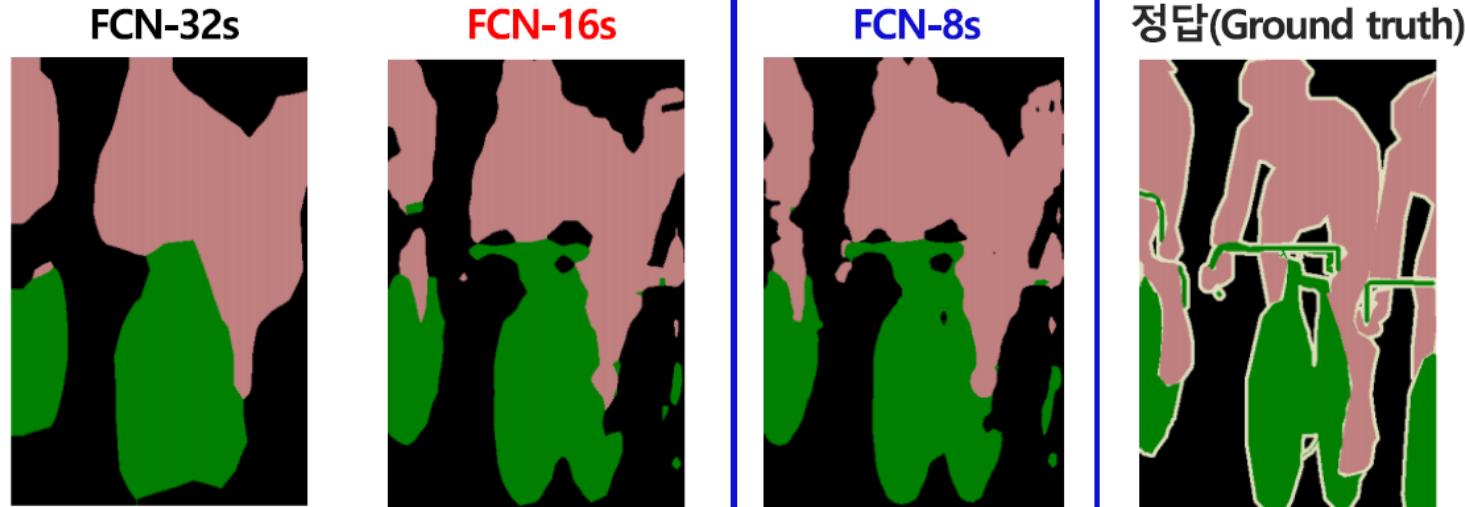


By Ardian Umam 14

# Feature Map을 혼합하여 Pixel Wise Prediction

## 2.2 FCN(Fully Convolution Networks)





	FCN-32s	FCN-16s	FCN-8s
IoU (Intersection over Union)	59.4(%)	62.4(%)	<b>62.7(%)</b>

## 2. 세그멘테이션

---

2.1 세그멘테이션 개요

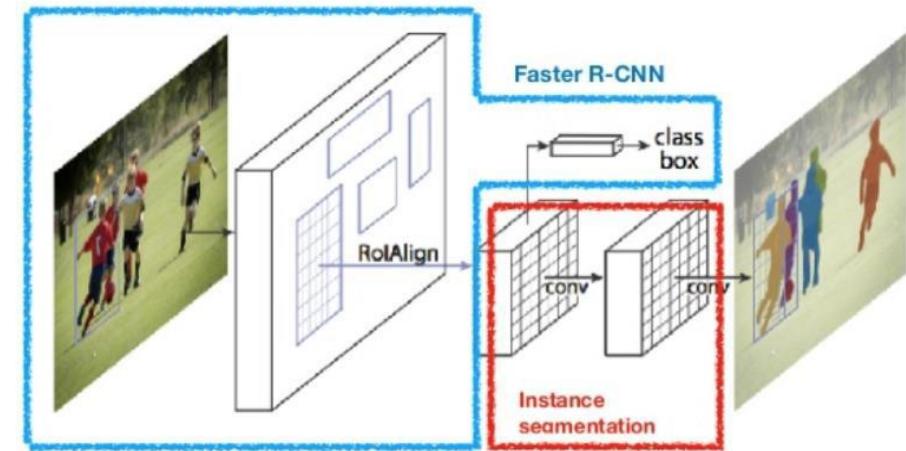
2.2 FCN

**2.3 Mask RCNN**

2.4 SAM(Segment Anything Model)

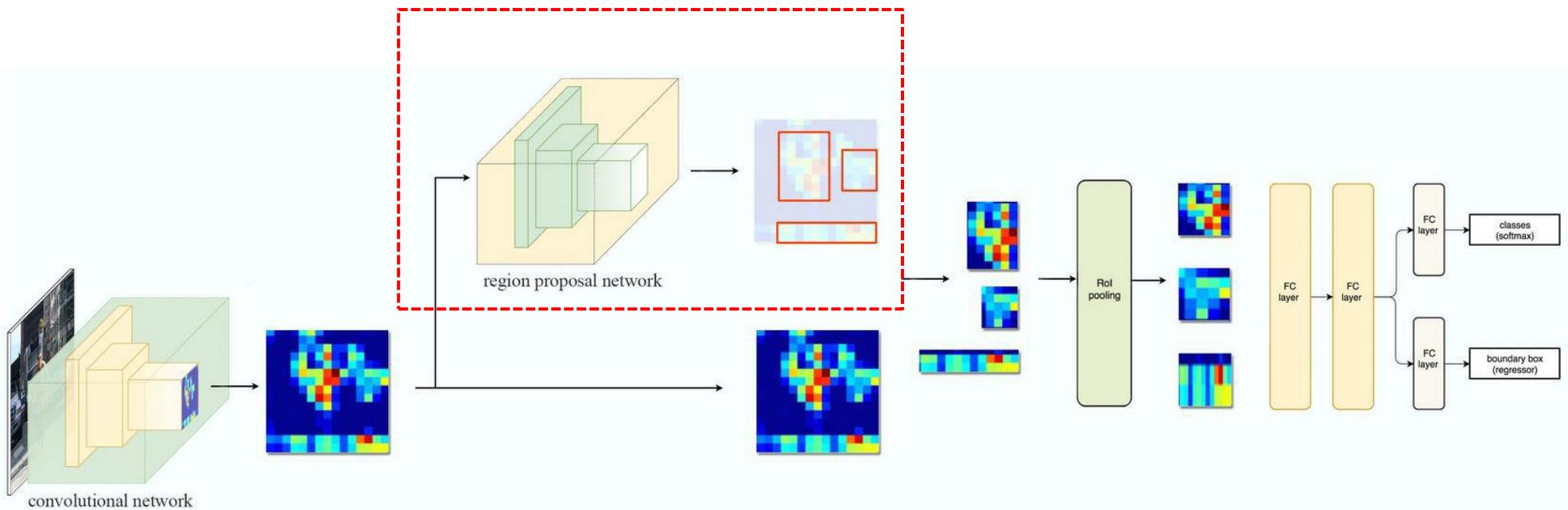
---

- Faster RCNN 과 FCN 기법 개선 및 결합
  - ✓ ROI-Align
  - ✓ 기존 Bounding box regression과 Classification에 Binary Mask Prediction 추가
- 비교적 빠른 Detection 시간과 높은 정확도
- 직관적이고 상대적으로 쉬운 구현



RPN

(Region Proposal Network)

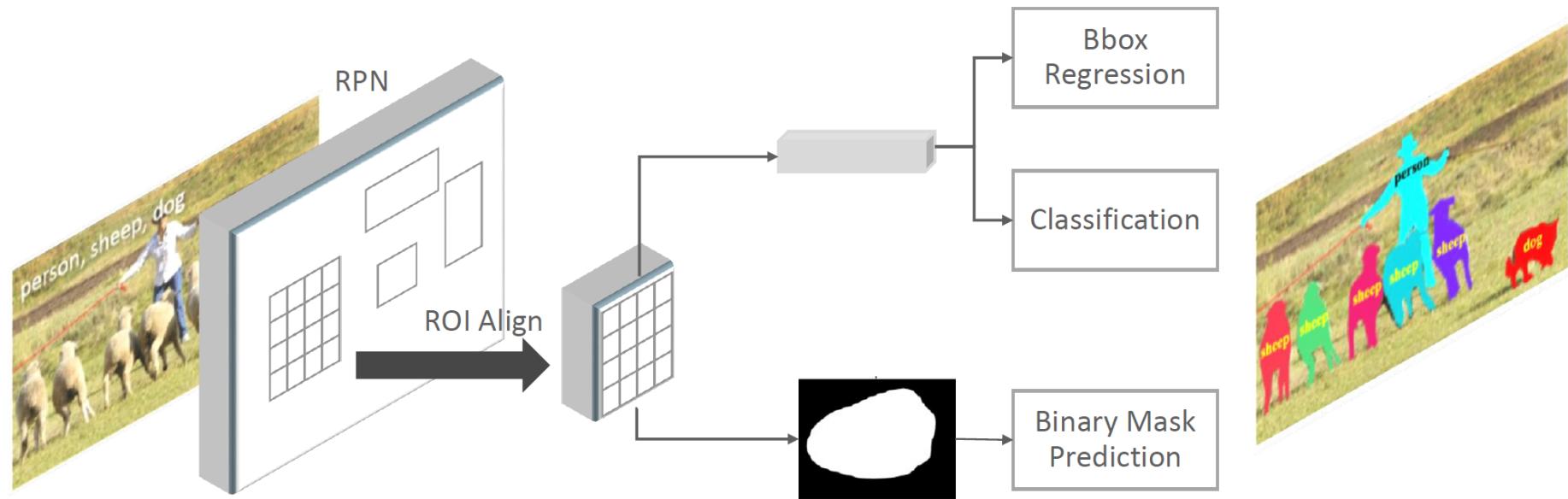


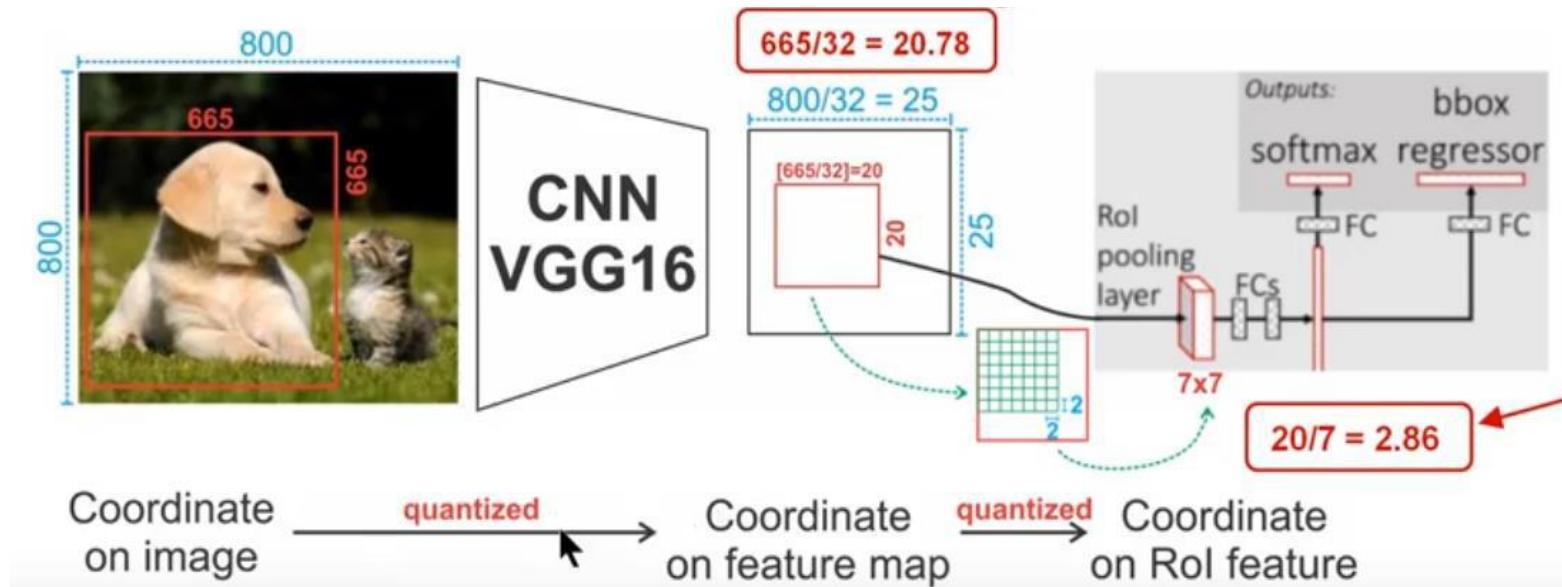
Selective Search를 Neural Network 구조로 변경

- GPU 사용으로 빠른 학습/Inference
- End to End Network 학습

# Mask RCNN 구조

## 2.3 Mask RCNN





출처: Mask RCNN Arc.(Part3) -How ROI Pooling, ROI Warping & ROI Align Work

<https://www.youtube.com/watch?v=XGi-Mz3do2s&list=PLkRkKTC6HZMxZrxnHUDYSLiPZxiUUFD2C&index=4>

# Segmentation에서 ROI Pooling 문제점

## 2.3 Mask RCNN

Feature Map

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

ROI 영역

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

2x2

2x2 ROI (Max) Pooling

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

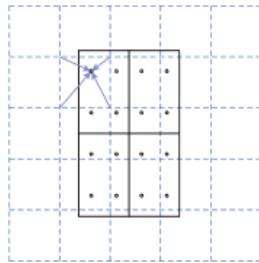


0.9	0.7
0.9	0.1

# Bilinear Interpolation을 이용한 ROI-Align

## 2.3 Mask RCNN

### ROI-Align



- ROI를 소수점 그대로 매핑하고 ROI의 개별 Grid에 4개의 point를 균등하게 배열
- 개별 point에서 가장 가까운 feature map Groid를 고려하면서 포인트를 weighted Sum으로 계산
- 계산된 포인트를 기반으로 Max Pooling 수행

### Bilinear Interpolation

확대 전 (입력)

0	1	2
1	4	6
2	7	9
⋮		
3	4	6
4		
5		★
6	7	9
⋮		
7		12

3배 확대 후 (출력)

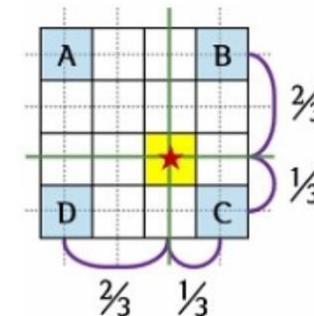
0	1	2	3	4	5	6
1						
2						
3	4		6		10	
4						
5						
6	7		9		12	
⋮						

(5,5)의 값=?

A			B
D			C

=

6			10
9			12

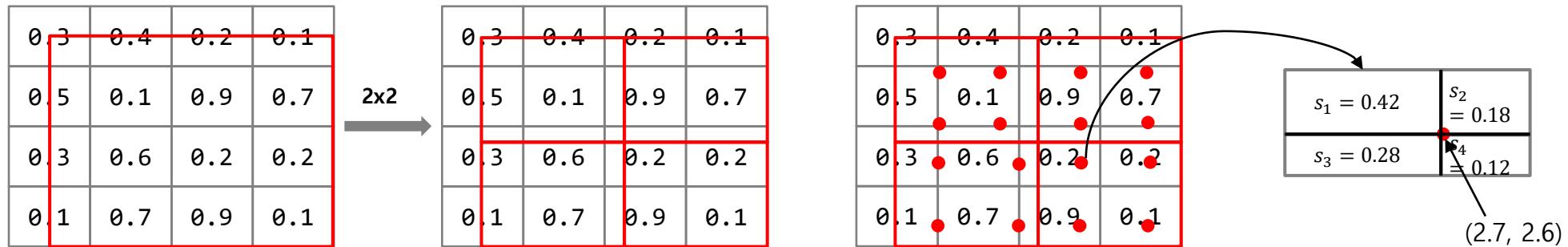


6	7.33	8.67	10
7	8.22	9.44	10.67
8	9.11	10.22	11.33
9	10	11	12

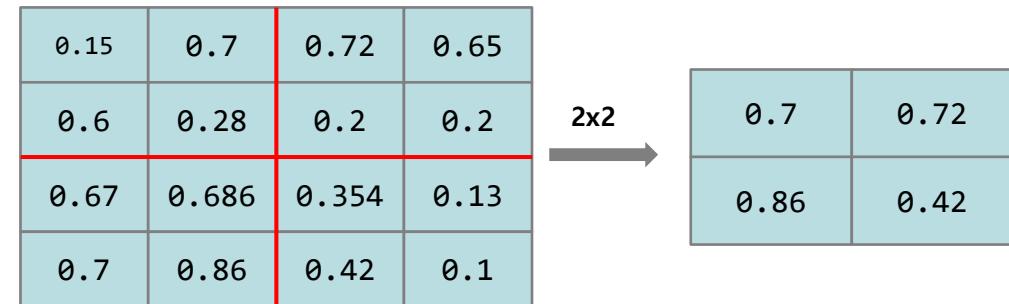
출처: 네이버블로그-영상처리bilinear interpolation 예제  
<https://blog.naver.com/dic1224/220882679460>

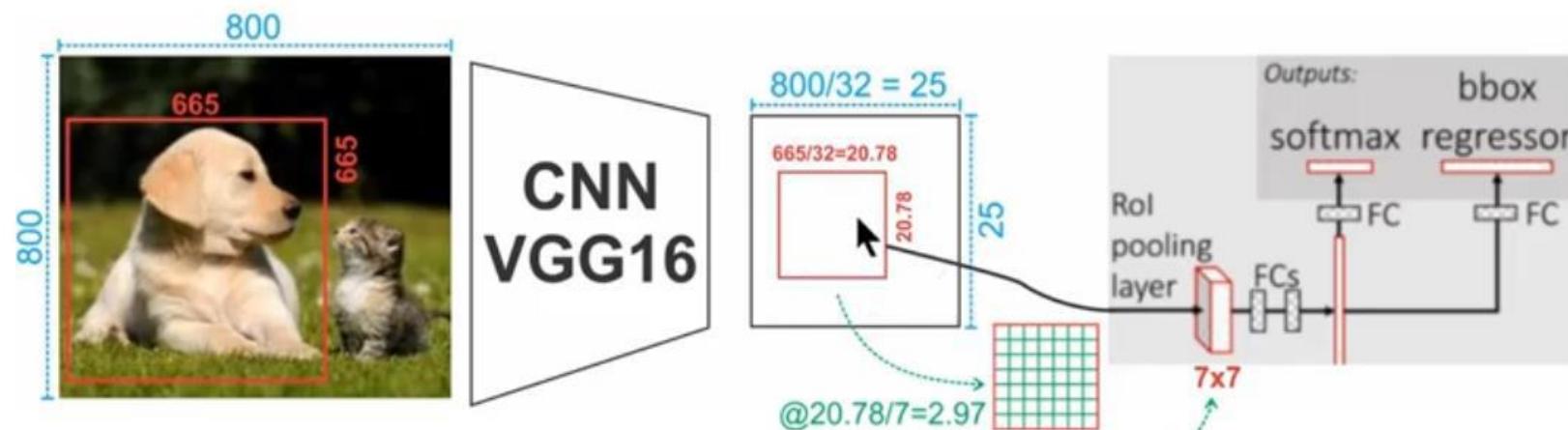
# Bilinear Interpolation을 이용한 ROI-Align

## 2.3 Mask RCNN



$$\begin{aligned}\text{InterpolatedValue} &= S_1 * \text{Pixel}(3,3) + S_2 * \text{Pixel}(2,3) + \\ &S_3 * \text{Pixel}(3,2) + S_4 * \text{Pixel}(2,2) \\ &= 0.42 * 0.1 + 0.18 * 0.2 + 0.28 * 0.9 + 0.12 * 0.2 = \mathbf{0.354}\end{aligned}$$





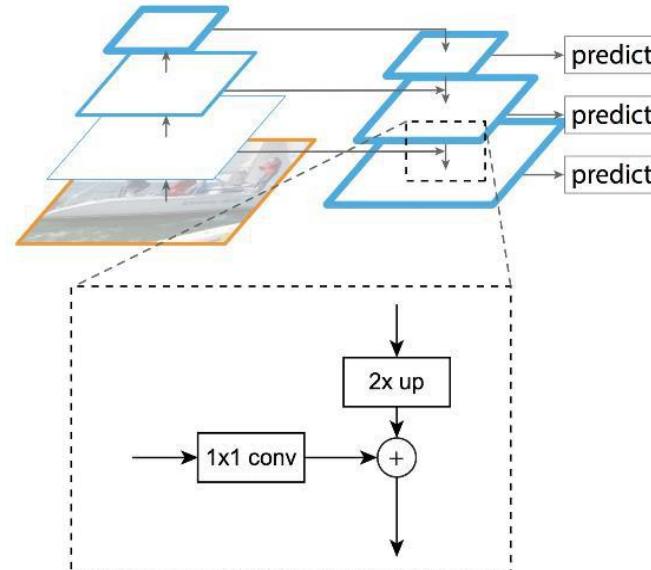
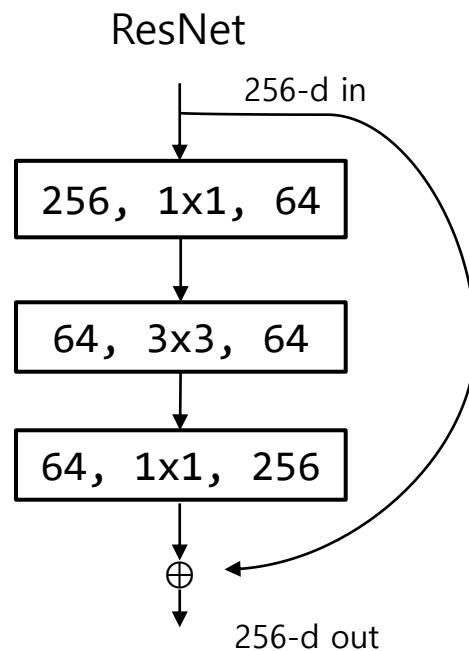
# ROI-Align 적용 후 성능 향상

## 2.3 Mask RCNN

	align?	bilinear?	agg	AP	$AP_{50}$	$AP_{75}$
ROIPool [12]			max	26.9	48.8	26.4
ROIWarp [10]		✓ ✓	max ave	27.2 27.1	49.2 48.9	27.1 27.1
ROIAlign	✓ ✓	✓ ✓	max ave	30.2 30.3	51.0 51.2	31.8 31.5

ROI Pooling 대비 약 3~4% AP 성능 향상

### Resnet + Feature Pyramid Network



- **Mask RCNN Loss**

$$L = L_{cls} + L_{bbox} + L_{mask}$$

- **Classification Loss**

Multiclass cross-entropy loss

- **Bounding box Loss**

$$L_{loc}(t^u, v) = \sum_{i \in \{x, y, w, h\}} smooth_{L_1}(t_i^u - v_i),$$

Smooth L1 loss

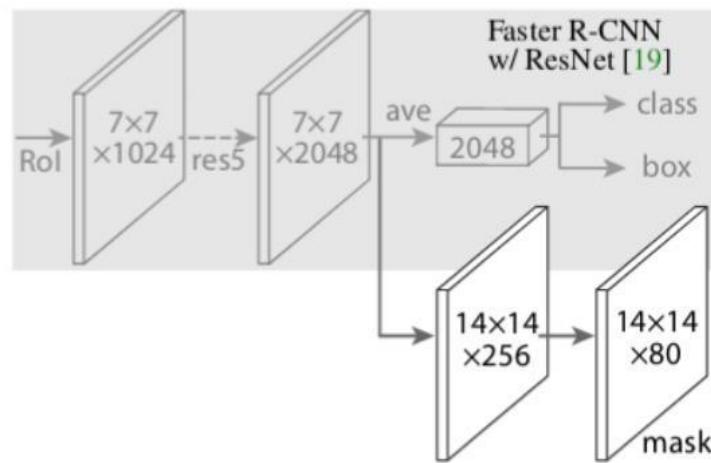
$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

- **Mask Loss**

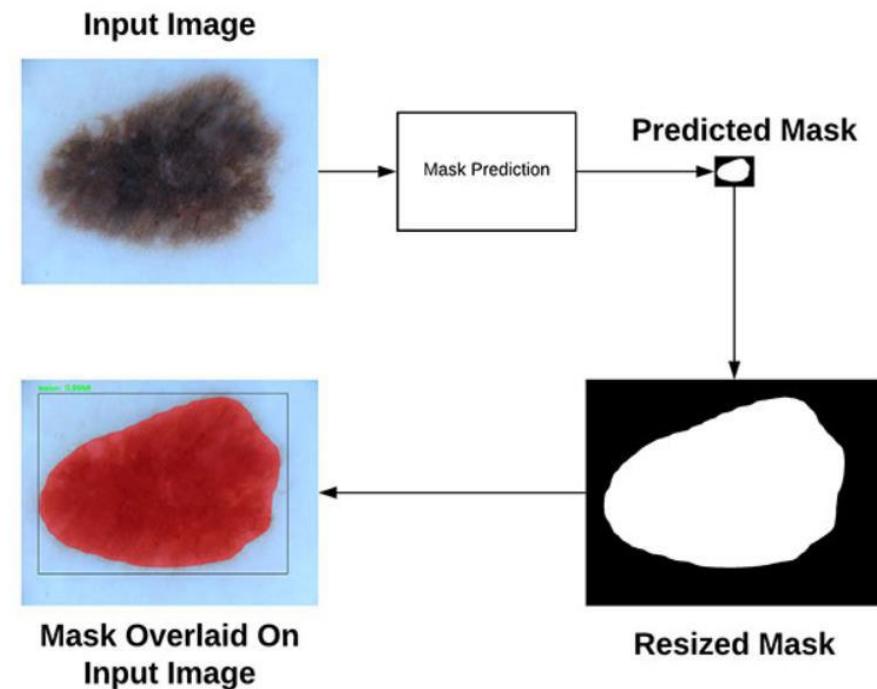
K개의 정해진 Class에 대해서 그 class에 pixel이 속하는지 그렇지 않는지 sigmoid로 결정

Binary cross-entropy loss

Mask RCNN에서 Mask Prediction



Mask Prediction 후 Resize하여 원본 이미지에 적용

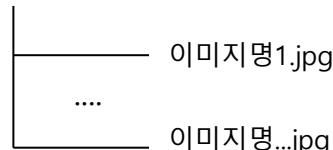


MS-COCO Dataset 기준

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
MNC [10]	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
FCIS [26] +OHEM	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
FCIS+++ [26] +OHEM	ResNet-101-C5-dilated	33.6	54.5	-	-	-	-
Mask R-CNN	ResNet-101-C4	33.1	54.9	34.8	12.1	35.6	51.1
Mask R-CNN	ResNet-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask R-CNN	ResNeXt-101-FPN	<b>37.1</b>	<b>60.0</b>	<b>39.4</b>	<b>16.9</b>	<b>39.9</b>	<b>53.5</b>

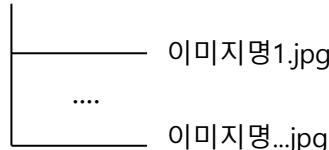
### MS COCO 데이터 세트 구조

train 2017



train용 Json 파일

val 2017



Valid용 Json 파일

### JSON Annotation 파일

- JSON 포맷인 한 개의 파일로 구성됨(한 라인으로 구성됨)
- 아래와 같은 대분류로 구성

info

COCO Dataset 생성 일자 등을 가지는 헤더 정보임

license

이미지 파일들의 라이센스에 대한 정보

images

모든 이미지들의 id, 파일명, 이미지 너비, 높이 정보

annotations

대상 image 및 object id segmentation, bounding box, 픽셀 영역등의 상세 정보

categories

80개 오브젝트 카테고리에 대한 id, 이름, Group을 가짐

```
{  
    "info" : { . . . },  
    "license" : [ . . . ],  
    "images" : [ . . . ],      ← 개별 image들의 정보  
    "annotation" : [ . . . ],  ← Bounding box, segmentation, annotation과  
    "categories" : [ . . . ],  
    "segment_info" : [ . . . ]  
}
```

```
"images": [  
  {  
    "license": 4,  
    "file_name": "000000397133.jpg",  
    "coco_url": "http://images.cocodataset.org/val2017/000000397133.jpg",  
    "height": 427,  
    "width": 640,  
    "date_captured": "2013-11-14 17:02:52",  
    "flickr_url": "http://farm7.staticflickr.com/6116/6255196340_da26cf2c9e_z.jpg",  
    "id": 397133  
  },  
  {  
    "license": 1,  
    "file_name": "000000037777.jpg",  
    "coco_url": "http://images.cocodataset.org/val2017/000000037777.jpg",  
    "height": 230,  
    "width": 352,  
    "date_captured": "2013-11-14 20:55:31",  
    "flickr_url": "http://farm9.staticflickr.com/8429/7839199426_f6d48aa585_z.jpg",  
    "id": 37777  
  },  
]
```

개별 image 정보를 Dictionary로 가지는 list 형태

```

"annotations": [
  {
    "segmentation": [
      [
        510.66,
        423.01,
        511.72,
        420.03,
        510.45,
        416,
      ]
    ],
    "area": 702.1057499999998,
    "iscrowd": 0,
    "image_id": 289343,
    "bbox": [
      [
        473.07,
        395.93,
        38.65,
        28.67
      ],
      "category_id": 18,
      "id": 1768
    ],
  }
]
  {
    "segmentation": [
      [
        289.74,
        443.39,
        302.29,
        445.32,
      ],
      "area": 27718.476299999995,
      "iscrowd": 0,
      "image_id": 61471,
      "bbox": [
        [
          272.1,
          200.23,
          151.97,
          279.77
        ],
        "category_id": 18,
        "id": 1773
      ],
    ]
  }
]

```

개별 Object의 annotation을 Dictionary로 가지는 list 형태

- 개별 Object의 Segmentation 정보를 Polygon 형태로 가짐
- bbox 정보와 개별 object의 class id(category\_id)정보를 가짐
- 개별 object의 id와 해당 object가 어떤 이미지에 소속되는지 (image\_id)매핑 정보를 가짐

```
"categories": [  
    {  
        "supercategory": "person",  
        "id": 1,  
        "name": "person"  
    },  
    {  
        "supercategory": "vehicle",  
        "id": 2,  
        "name": "bicycle"  
    },  
]
```

개별 Object의 annotation을 Dictionary로 가지는 list 형태

- 개별 category id와 category명, 그리고 부모 category명을 가짐

coco 데이터 세트내 이미지, Annotations등의 정보들을 쉽게 access 할 수 있도록 지원해 주는 API

### *pycocotools*

```
class COCO:  
    def __init__(self, annotation_file=None):  
  
    def getAnnIds(self, imgIds=[], catIds=[], areaRng=[], iscrowd=None):  
  
    def getCatIds(self, catNms=[], supNMs=[], catIds[]):  
  
    def getImgIds(self, imgIds=[], catIds[]):  
  
    def loadAnns(self, ids=[]):
```

# Polygon과 Mask

## 2.3 Mask RCNN

원본 이미지에 Polygon 적용



Mask



원본 이미지에 Mask 적용



# Mask RCNN inference 시 반환 결과 분석 - bbox정보

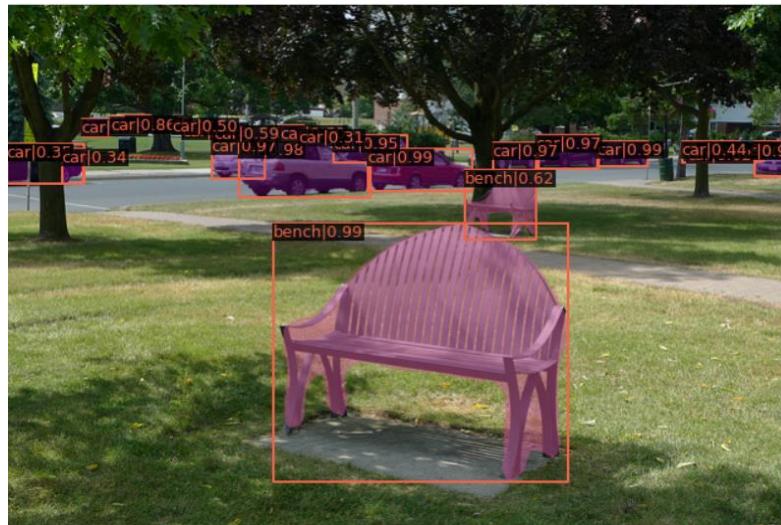
## 2.3 Mask RCNN

```
results = inference_detector(model, img_arr)
```

results[0]은 bbox prediction(class, bbox좌표) 결과

results[1]은 mask prediction 결과

results[1]은 list 객체로서 내부 원소가 2차원 array로 구성



coco class id=0 bbox정보  
(detec된 object 1개)

500	100	540	125	0.99	shape=(1,5)
-----	-----	-----	-----	------	-------------

coco class id=1 bbox정보  
(detec된 object 없음)

.....	.....	.....	.....	.....	shape=(0,5)
-------	-------	-------	-------	-------	-------------

coco class id=2 bbox정보  
(detec된 object 3개)

420	200	440	225	0.90	shape=(3,5)
....	....	....	....	....	
....	....	....	....	....	

]

# Mask RCNN inference 시 반환 결과 분석 - mask정보

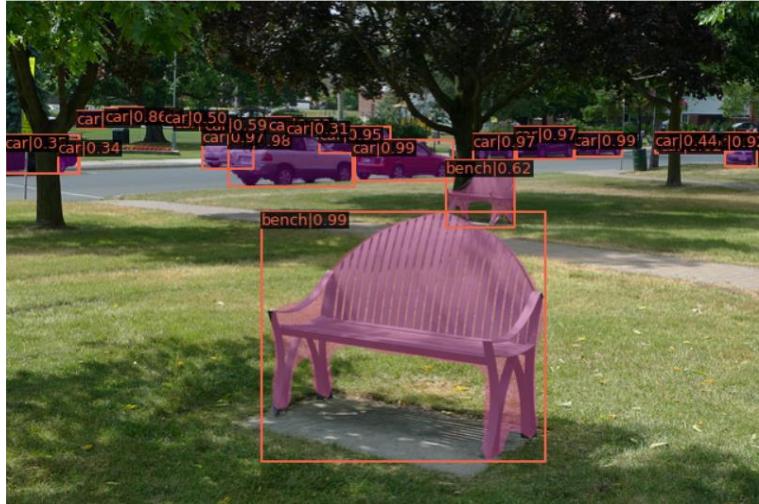
## 2.3 Mask RCNN

```
results = inference_detector(model, img_arr)
```

results[0]은 bbox prediction(class, bbox좌표) 결과

results[1]은 mask prediction 결과

results[1]은 list 객체로서 내부 원소가 class 갯수만큼의 list로 구성. 해당 내부 원소 list는 object 별 mask 정보를 가지며, 개별 mask정보는 image size만큼의 array로 구성



coco class id=0 mask정보  
(detec된 object 1개)

coco class id=1 mask정보  
(detec된 object 없음)

coco class id=2 bbox정보  
(detec된 object 3개)

image width					
image height	False	False	....	False	False
....	....	....	....	....	....
False	False	....	False	False	False

image width					
image height	False	False	....	False	False
....	....	....	....	....	....
False	False	....	False	False	False

image width					
image height	False	False	....	False	False
....	....	....	....	....	....
False	False	....	False	False	False

image width					
image height	False	False	....	False	False
....	....	....	....	....	....
False	False	....	False	False	False

## 2. 세그멘테이션

---

2.1 세그멘테이션 개요

2.2 FCN

2.3 Mask RCNN

**2.4 SAM(Segment Anything Model)**

---

최근 대규모의 언어 모델들은 놀라운 Zero-shot / Few-shot Generalization 성능을 보이고 있다. 이러한 Foundation Model들은 종종 Prompt Engineering을 통해 여러가지 Task에 대한 적절한 텍스트 응답을 생성해주는 능력을 보여주기도 한다.

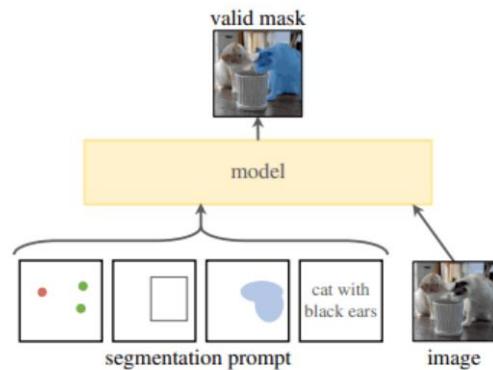
이 연구에서는 Image Segmentation에 대한 Foundation Model을 만드는 것을 목표로 한다. 또한 강력한 Generalization을 바탕으로 Prompt Engineering을 통해 여러 종류의 Image Segmentation 하위 문제들을 풀어내고자 한다.

이것을 가능하게 하기 위해서는 다음과 같은 세 가지 질문에서 출발할 필요가 있다.

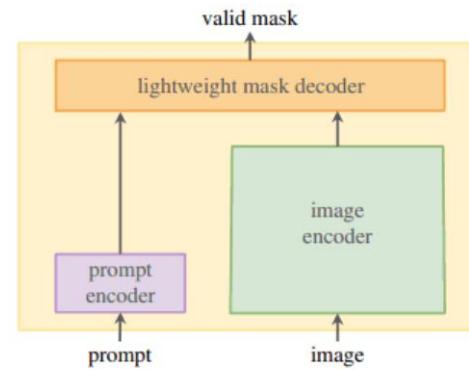
Zero-shot Generalization을 가능하게 하는 Task는 무엇인가? (Task)

적절한 Model 구조는 무엇인가? (Model)

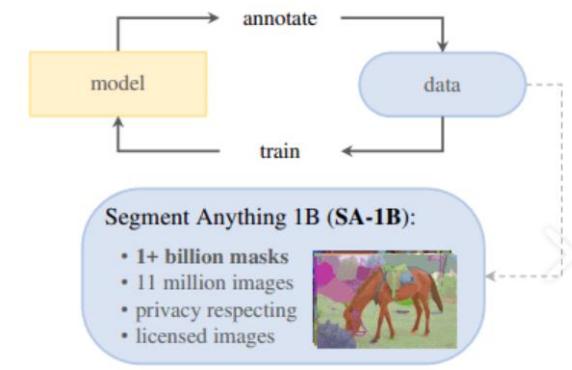
1의 Task와 2의 Model에 적절한 Data는 무엇인가? (Data)



(a) Task: promptable segmentation



(b) Model: Segment Anything Model (SAM)



(c) Data: data engine (top) & dataset (bottom)

NLP (Natural Language Process)의 경우, Foundation Model을 학습하기 위해 Next Token Prediction과 같은 Task에 대한 Pre-training을 수행한다. 마찬가지로 Segmentation에서의 Foundation Model을 도출하기 위해 이와 유사한 효과를 기대할 수 있는 task를 정의하는 것을 목표로 한다.

저자들은 Foreground / Background Points, Box / Masks / Free-form Text 등을 이미지에 대한 Prompt로 정의한다. (2023년 4월 6일 기준 Free-form Text 입력은 미공개) 일명 Promptable Segmentation Task는 그 어떤 모호한 정보가 Prompt로 입력되더라도 유효한 마스크를 반환하는 것을 목표로 한다.

Training의 경우 Points, Boxes, Masks와 같은 Prompt Sequence를 시뮬레이트해서 마스크를 취득한 뒤 Ground-Truth Mask와 비교하는 방식을 사용한다(\*)。

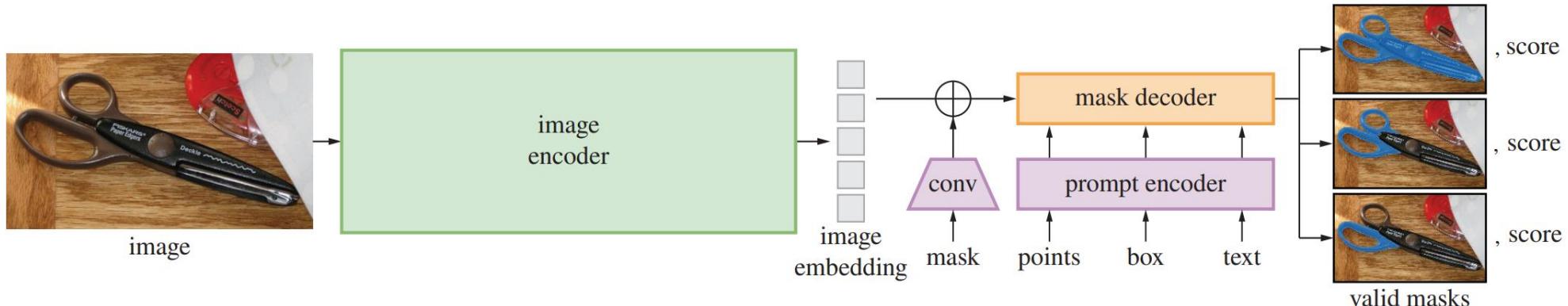
이렇게 얻어낸 Pre-trained Model은 Prompt Engineering을 통해 여러가지 Task에 유연하게 적용될 수 있다. 좀 더 자세히 살펴보자.

Training Process를 위해 Interactive Segmentation을 시뮬레이션한다.

1. Target Mask의 Foreground Point 또는 Bounding Box를 랜덤하게 선택한다. Point는 Target Mask에서 랜덤하게 샘플링하며, Box는 Ground Truth Mask의 Bounding Box를 사용한다. 박스의 좌표에는 박스 Sidelength의 10%의 STD로 생성한 랜덤 노이즈를 추가한다 (최대 20픽셀). - 유저가 Loose Box를 그리는 경우에 대비하기 위함.
2. 첫 번째 Prompt로 Prediction을 한 뒤, 그 다음 Point는 Previous Mask Prediction과 Ground Truth Mask 사이의 Error Region에서 Uniformly Random으로 선택한다. 만약 Error Region이 False Negative면 Point는 Forground가 되고, False Positive면 Background가 된다. 또한 Previous Mask를 모델에 추가적인 Prompt로 입력한다. 이때 Mask는 정보량을 최대화하기 위해 {0,1}로 Thresholding되지 않은 값을 사용한다. 만약 여러개의 Mask가 반환되었을 경우에는 Predicted IoU가 가장 높은 것을 사용한다.
3. 마스크 정보를 제공하는 것에서 이득을 얻기 위해 랜덤하게 2개 이상의 Iteration에서 Point 정보를 입력하지 않는다. (모델은 이전 마스크 정보를 Refine하게 됨)

# Segment Anything Model

## 2.4 SAM(Segment Anything Model)



SAM은 세 개의 컴포넌트로 구성되어 있다: Image Encoder, Flexible Prompt Encoder, Fast Mask Decoder.

- **Image Encoder:**

고해상도의 이미지를 처리하기 위해 Masked Autoencoder로 Pre-training을 한 Vision Transformer (ViT) 기반의 구조를 사용한다(\*). Image encoder의 출력은 입력 이미지 크기를 기준으로 16배 다운스케일된 Embedding이다. 참고로, Image Encoding은 Prompt처럼 실시간성이 요구되지 않기 때문에 높은 스케일로 사용하는 것이 가능하다. (프롬프트와 달리 이미지 당 최초 한번만 수행)

1024x1024의 이미지 입력을 사용했으며 좌우/위아래 중 짧은 부분에는 padding을 한다. 즉, Embedding의 크기는  $1024 / 16$ 에 해당하는  $64 \times 64$ 가 된다. 또한  $1 \times 1$  Convolution을 사용해서 256 채널의 출력을 얻으며 이어서 256채널의  $3 \times 3$  Convolution을 적용한다. 각 Convolution 연산 뒤에는 Layer Normalization을 적용한다.

- **Flexible Prompt Encoder:**

Point, Box, Text와 같은 Sparse Prompt는 256 차원의 Embedding으로 매핑된다. Point는 Point Location과 학습 가능한 Foreground / Background Embedding 중 하나와의 합으로 표현한다. Box는 Top-left Corner를 나타내는 Learned Embedding과 Bottom-right Corner를 나타내는 Learned Embedding의 합으로 표현한다. 끝으로 Free-form Text Encoder는 CLIP의 것을 차용한다.

Dense Prompt의 경우 이미지와 공간적으로 대응되는 정보다 (e.g. 마스크). 마스크는 입력 이미지보다 1/4의 크기로 입력한다. 여기에 2x2, stride 2의 Convolution을 적용하여 추가적으로 크기를 4배 더 줄인다. 마지막으로 1x1 Convolution을 사용하여 채널을 256으로 변환한다. 각 Layer는 GELU activation과 Layer Normalization을 적용한다. 마스크와 Image Embedding은 Element-wise로 더해진다. 만약에 Mask 입력이 없는 경우에는 "no mask"를 나타내는 embedding을 사용한다.

- **Fast Mask Decoder:**

Prompt Embedding과 Image Embedding을 빠르게 Output Mask에 매핑하는 역할을 한다. 두 개의 입력을 합치는 방법으로는 Transformer Segmentation Models를 참고했으며 (참고1, 참고2), Standard Transformer Decoder를 약간 수정한 구조를 사용했다.

Decoder 적용 이전 단계에서는 Prompt Embedding에 Learned Output Token Embedding을 삽입한다 (Vision Transformer (ViT) 의 Class Token과 유사).

각 Decoder Layer는 다음 4가지 Step으로 동작한다.

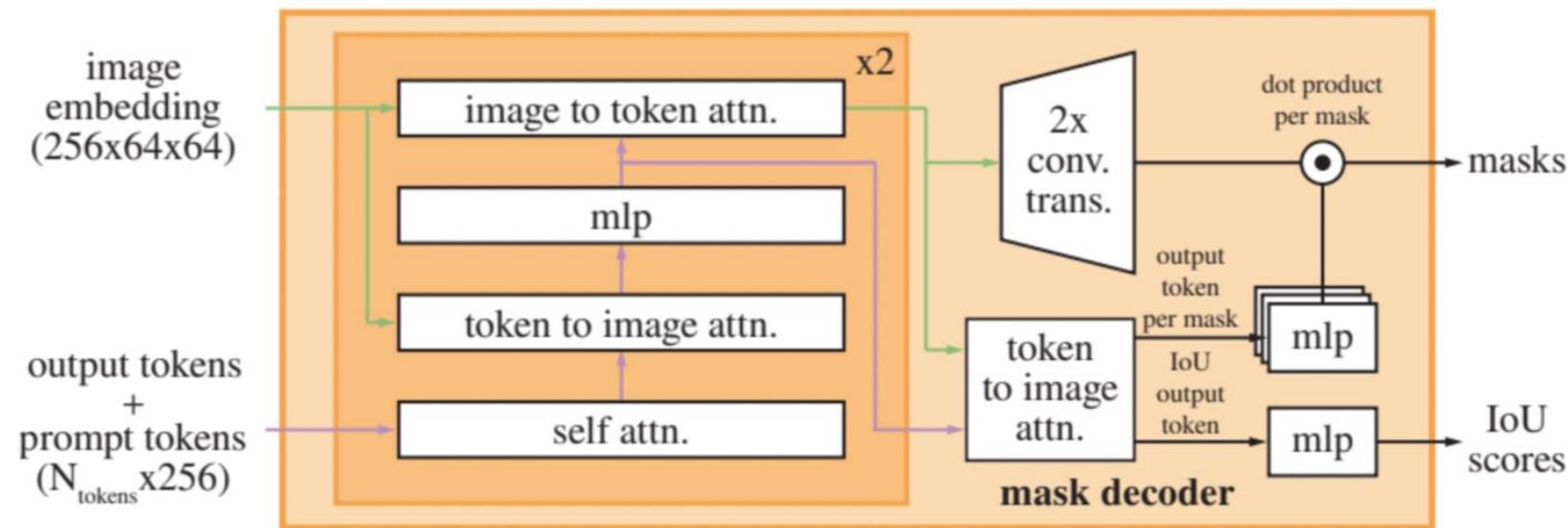
1. Token에 대한 Self-attention
2. Token을 Query로 하여 Image Embedding에 Cross-Attention
3. Point-wise MLP로 각 Token을 업데이트
4. Image Embedding을 Query로 하여 Token에 Cross-Attention  
(Cross-Attention 과정에서 Image Embedding은 64x64의 256 차원 벡터)

각 Self/Cross-attention과 MLP에는 Residual Connection, Layer Normalization, Dropout(0.1)을 적용한다. 다음 Decoder Layer에서는 이전 레이어에서 업데이트된 Image Embedding과 Token을 사용한다.  
(Decoder는 2층으로 구성)

또한 중요한 Geometric 정보를 잊지 않기 위해 Attention Layer에 입력할 때마다 Image Embedding에 Positional Encoding을 합산하며, 마찬가지로 Attention Layer에 입력할 때마다 Updated Token에 Original Prompt Token 전체를 합산한다.

Decoder를 수행한 뒤에는 2개의 Transposed Convolutional Layer를 적용하여 Image Embedding의 크기를 4배 키운다. 한편, Token을 Query로 하여 한번 더 Image Embedding에 Attention을 한뒤, Output Token Embedding에 3개의 MLP 를 적용하여 Channel-wise로 Upscaled Image Embedding에 적용한다. 다음은 각 컴포넌트에 대한 추가적인 정보다.

- Transformer는 Embedding Dimension으로 256을 사용.
- Transformer MLP block은 2048의 Internal Dimension 사용. (Prompt Token에만 적용)
- Cross-attention Layer에서는 64x64 Image Embedding을 사용했으며, Query / Key / Value의 채널은 계산 효율을 위해 128로 두배 줄임.
- 모든 Attention Layer는 8개의 Head를 사용.
- Output Image Embedding에 적용하는 Transposed Convolution은 2x2, stride 2이며 Output Channel은 64와 32 (GELU activation, Layer Normalization 사용).



- **Loss:**

Mask의 경우 Focal Loss와 Dice Loss를 20:1 비율로 Linear Combination하여 사용. IoU Prediction Head의 경우 IoU Prediction과 Ground-truth 사이의 Mean-square-error로 Loss를 계산. IoU Loss와 Mask Loss는 1:1로 합산.

텍스트 정보와 달리 Segmentation Mask는 인터넷에서 손쉽게 구하기 어려운 데이터다. Meta는 대규모 (1.1B)의 마스크를 취득하기 위해 자체적인 Data Engine을 구현했다. Data Engine은 다음 세 가지 Stage로 구성된다.

### 1. A model-assisted manual annotation stage:

첫 번째 단계는 전통적인 interactive segmentation 과정과 유사하다. 전문 annotator들이 Segment Anything Model (이하 SAM)의 도움을 받아 웹 기반의 인터페이스를 통해 foreground / background object를 작업한다. 이때 마스크는 "stuff" 또는 "things"로 어노테이터가 자유롭게 레이블링하도록 한다. 이 단계에서 사용하는 SAM은 공개된 segmentation dataset을 이용해 학습한다. 그리고 어노테이션 작업으로 추가 Data를 충분히 취득했을 때 새로 취득한 데이터만을 사용해서 다시 학습을 수행한다. 이 과정에서 Image Encoder를 ViT-B에서부터 ViT-H까지 점진적으로 증가시키며, 총 6번의 추가 학습 과정을 수행하게 된다. 결과적으로 이 단계에서 120k 이미지로부터 4.3M개의 마스크를 취득한다.

### 2. A semi-automatic stage with a mix of automatically predicted masks and model-assisted annotation:

이 단계에서는 모델의 Segment Anything 능력을 향상시키기 위해 마스크의 종류를 다양화하는 것을 목표로 한다. 우선 모델을 이용해 Confident Mask를 잡아준 뒤에 이를 작업화면 상에 표시하면, 어노테이터들은 그 외 영역의 Object들을 작업하도록 한다. Confident 마스크를 파악하기 위해서는 첫 번째 단계에서 수집한 모든 마스크에 대한 Bounding Box Detector를 학습시킨다 ("Object" 카테고리). 이 단계를 거치며 180k 이미지로부터 5.9M 마스크를 취득한다. 첫 번째 단계에서처럼 새로 취득한 데이터를 이용해 5번의 점진적인 모델 학습을 거친다.

### 3. A fully automatic stage (without annotator input):

마지막 단계는 완전 자동화된 어노테이션 단계다.

- 모델에  $32 \times 32$ 의 Regular Grid Point를 입력(prompted). 모델은 각 포인트 아래의 Object에 해당하는 마스크를 반환(Predicted).
- 모델은 Object의 Subpart, Part, Whole Object를 모두 반환.
- 모델의 IoU Prediction Module를 사용해 Confident Mask를 선택.
- 선택한 마스크 중에서 Stable한 마스크를 고름. (Probability Map을  $0.5 - \delta$ 와  $0.5 + \delta$ 로 Thresholding 한 뒤에도 마스크의 형태가 유사하면 Stable하다고 판단)
- 앞서 선택한 Stable & Confident Mask들에 Non-maximal Suppression (NMS)를 적용하여 중복된 것들을 제거.
- 작은 마스크 품질을 높이기 위해 Multiple Overlapping Zoomed-in Image Crops를 이용.

위와 같은 방법을 적용해서 11M 이미지로부터 1.1B의 고품질 마스크를 추출해냈다.

# Segment Anything Dataset

## 2.4 SAM(Segment Anything Model)

SA-1B 데이터셋은 11M의 다양하고, 높은 화질의 이미지를 포함하며, 또한 Data Engine으로부터 취득한 1.1B개의 고품질 마스크를 포함한다.

- **Image:**

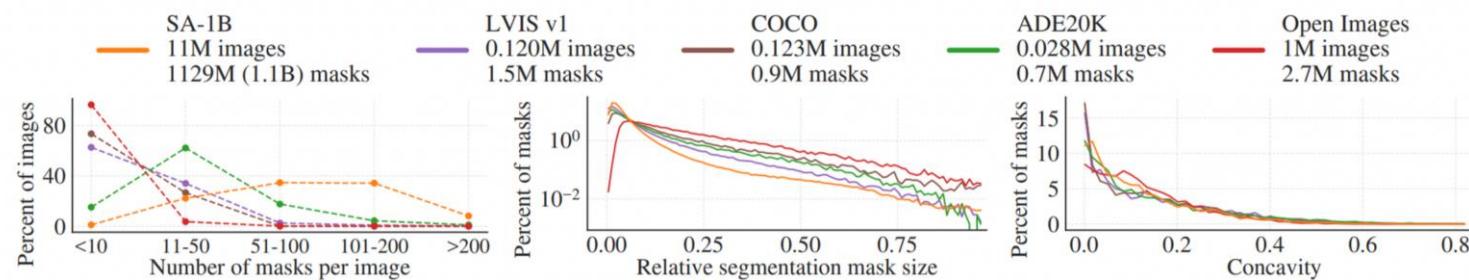
평균  $3300 \times 4950$  pixels 이미지를 취득. 이미지 크기가 너무 커지는 것을 고려하여 Shortest Side를 기준으로 1500 pixel로 다운샘플. 사람, 차량 번호판 등은 블러 처리.

- **Masks:**

Data Engine을 통해 1.1B의 마스크를 만들었으며, 이 중 99.1%는 완전 자동으로 취득. 즉, 자동 마스킹 기능의 성능이 마스크 품질에 있어 매우 중요. 이 부분을 확인하기 위해 전문가 어노테이터와 생성된 마스크 사이의 품질을 측정하였고, 생성된 마스크의 품질이 충분히 좋다고 결론. 이에 SA-1B에는 자동으로 취득한 마스크만 포함 (!).

- **Mask Quality:**

마스크의 품질을 평가하기 위해 500개의 이미지를 랜덤으로 샘플링 (약 50k의 마스크 포함). 전문 어노테이터들은 샘플링한 데이터를 수정하는 작업을 수행. 이 방법으로 자동 생성 마스크와 이에 대한 Ground Truth Mask 취득했으며, 이 둘의 IoU를 측정. 결과적으로 85~91%의 IoU를 기록. 또한 실험을 통해 Human Rating으로 여러 데이터셋에 대해 생성된 마스크가 충분히 좋은 품질을 가지고 있음을 확인 (아래 Experimental Results의 Fig9-(b) 참고).



# Zero-Shot Single Point Valid Mask Evaluation

## 2.4 SAM(Segment Anything Model)

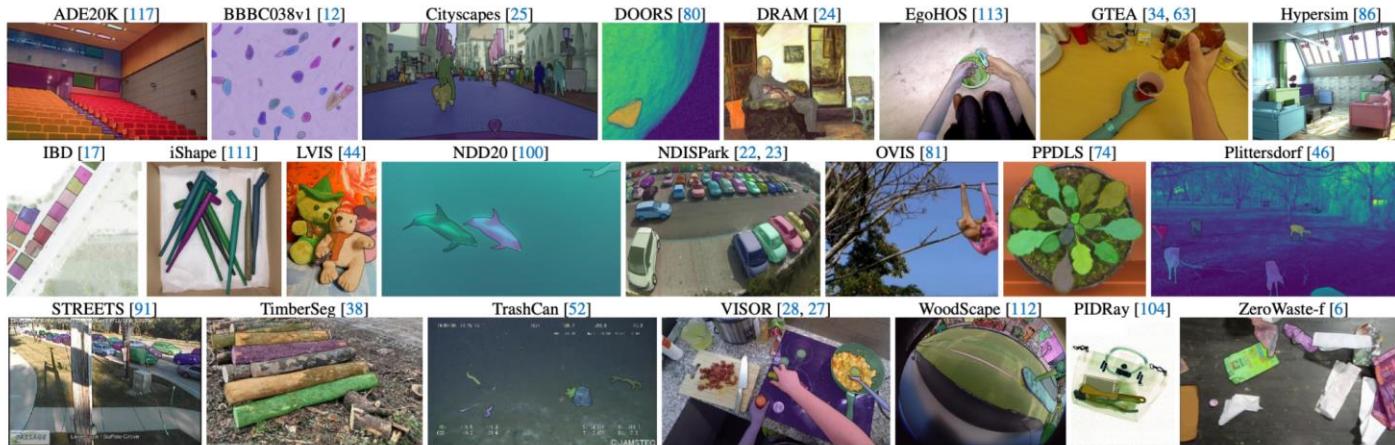
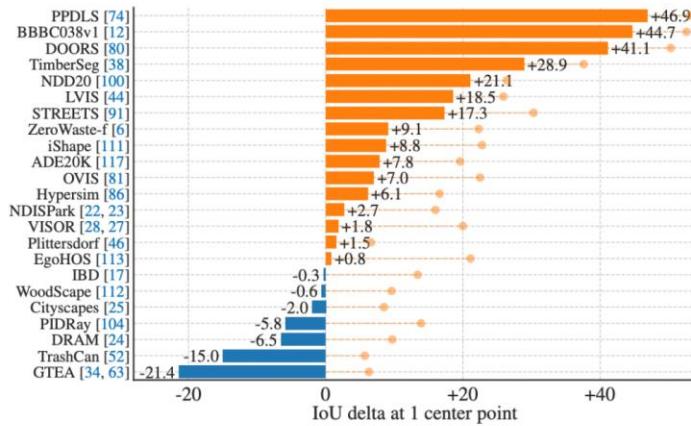
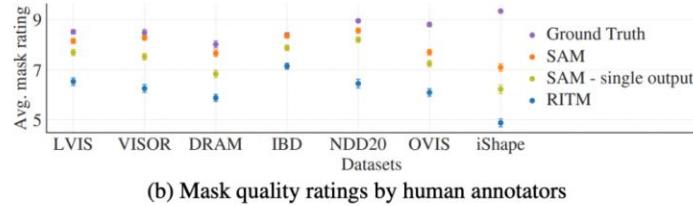


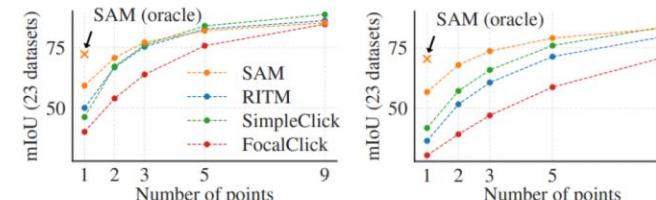
Figure 8: Samples from the 23 diverse segmentation datasets used to evaluate SAM’s zero-shot transfer capabilities.



(a) SAM vs. RITM [92] on 23 datasets



(b) Mask quality ratings by human annotators



(c) Center points (default)

(d) Random points

Single Foreground Point로 Object에 대한 Segmenting을 수행하는 것이다. 다양한 이미지 분포에서 실험을 진행하기 위해 23개의 데이터셋을 사용했다. 결과적으로 23개 데이터셋 중 16개 데이터셋에서 RITM을 상회하는 성적을 보였다.



BSDS500 데이터셋을 이용해 Edge Detection을 수행해봤다. Edge는 Unthresholded Mask Probability Map에 Sobel Filtering을 적용해서 도출했다. Edge Detection을 학습하지 않았음에도 정성적으로 매우 그럴듯한 Edge Map을 보여준다. 또한 SAM은 HED 같은 딥러닝 기반의 Edge Detector 보다도 더 좋은 정량적 성과를 보인다.

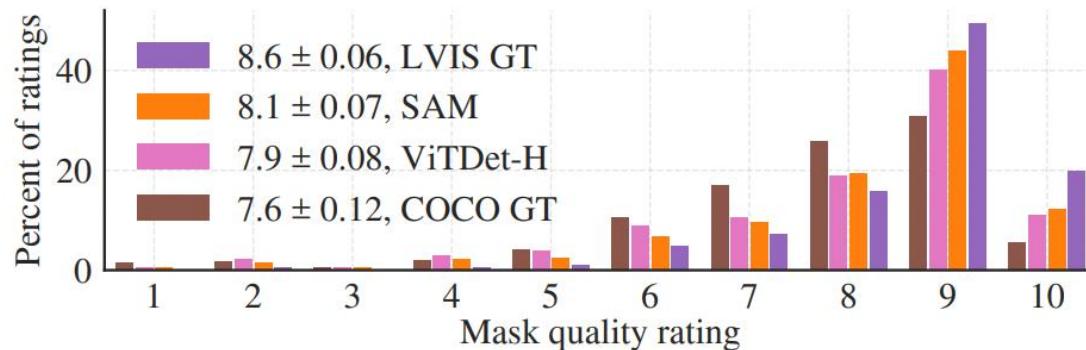
method	all	mask AR@1000					
		small	med.	large	freq.	com.	rare
ViTDet-H [62]	63.0	51.7	80.8	87.0	63.1	63.3	58.3
<i>zero-shot transfer methods:</i>							
SAM – single out.	54.9	42.8	76.7	74.4	54.7	59.8	62.0
SAM	59.3	45.5	81.6	86.9	59.1	63.9	65.8

Table 4: Object proposal generation on LVIS v1. SAM is applied zero-shot, *i.e.* it was not trained for object proposal generation nor did it access LVIS images or annotations.

Object Detection에서 매우 중요한 Task인 Object Proposal Generation의 벤치마킹 결과다. ViTDet-H가 전반적으로 가장 좋은 결과를 보였지만, 일부 Metric에서는 SAM이 약간은 앞서는 모습을 보이기도 한다.

method	COCO [66]				LVIS v1 [44]			
	AP	AP <sup>S</sup>	AP <sup>M</sup>	AP <sup>L</sup>	AP	AP <sup>S</sup>	AP <sup>M</sup>	AP <sup>L</sup>
ViTDet-H [62]	51.0	32.0	54.3	68.9	46.6	35.0	58.0	66.3
<i>zero-shot transfer methods (segmentation module only):</i>								
SAM	46.5	30.8	51.0	61.7	44.7	32.5	57.6	65.5

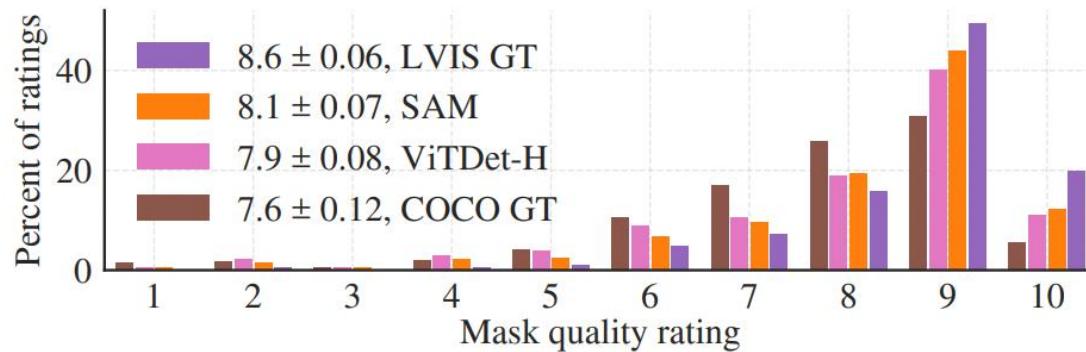
Table 5: Instance segmentation results. SAM is prompted with ViTDet boxes to do zero-shot segmentation. The fully-supervised ViTDet outperforms SAM, but the gap shrinks on the higher-quality LVIS masks. Interestingly, SAM outperforms ViTDet according to human ratings (see Fig. 11).



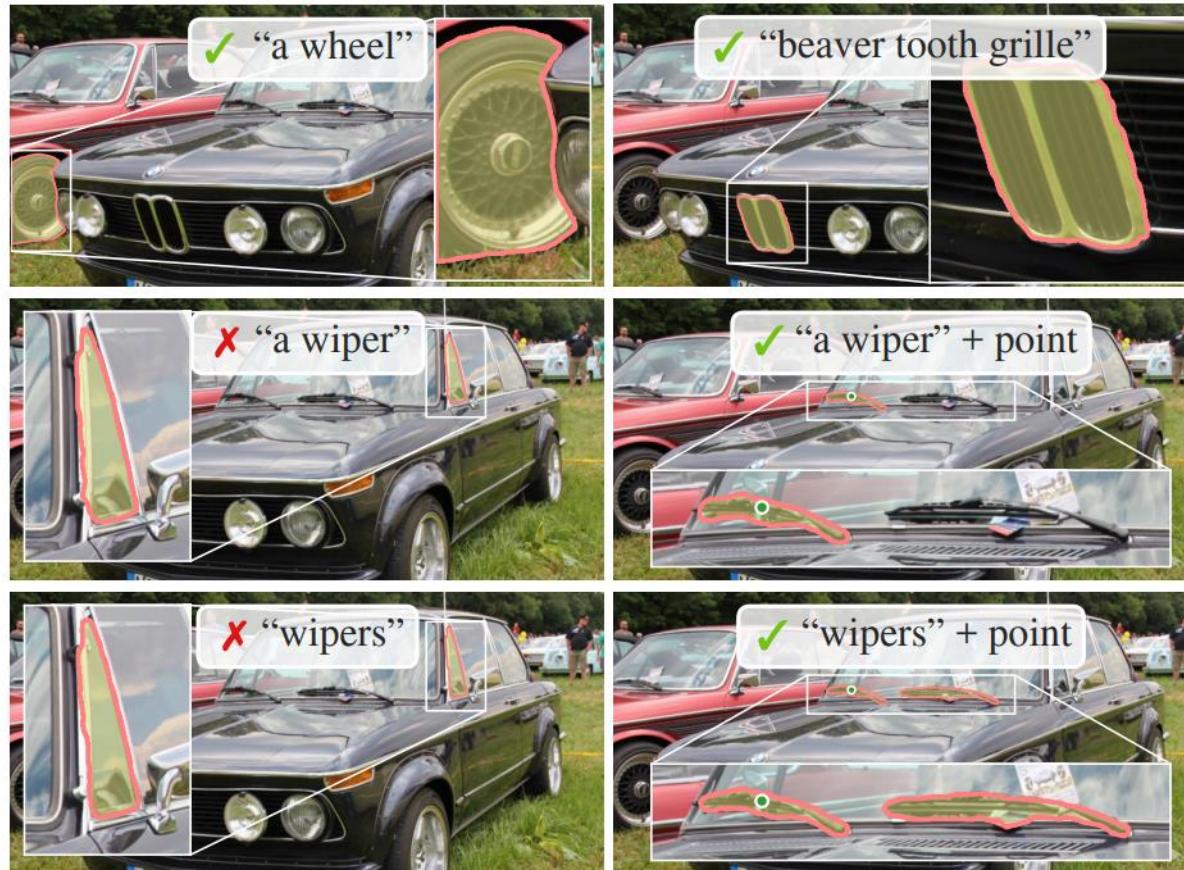
COCO와 LVIS 데이터셋에서 ViTDet과 SAM의 성능을 비교해봤다. 정량평가로는 ViTDet에 다소 뒤쳐지지만, 흥미롭게도 Human Rating에서는 ViTDet보다 더 좋은 결과를 얻었다.

method	COCO [66]				LVIS v1 [44]			
	AP	AP <sup>S</sup>	AP <sup>M</sup>	AP <sup>L</sup>	AP	AP <sup>S</sup>	AP <sup>M</sup>	AP <sup>L</sup>
ViTDet-H [62]	51.0	32.0	54.3	68.9	46.6	35.0	58.0	66.3
<i>zero-shot transfer methods (segmentation module only):</i>								
SAM	46.5	30.8	51.0	61.7	44.7	32.5	57.6	65.5

Table 5: Instance segmentation results. SAM is prompted with ViTDet boxes to do zero-shot segmentation. The fully-supervised ViTDet outperforms SAM, but the gap shrinks on the higher-quality LVIS masks. Interestingly, SAM outperforms ViTDet according to human ratings (see Fig. 11).



COCO와 LVIS 데이터셋에서 ViTDet과 SAM의 성능을 비교해봤다. 정량평가로는 ViTDet에 다소 뒤쳐지지만, 흥미롭게도 Human Rating에서는 ViTDet보다 더 좋은 결과를 얻었다.



Free-fromed Text로부터 Segmentation을 수행하는 Task다. 이 실험의 경우 SAM의 능력치를 실험하기 위한 일종의 PoC (Proof of Concept)이다. Text 입력만으로는 정확한 출력이 나오지 않을 때는, PhraseClick 처럼 클릭 정보와 함께 보조적인 정보로 활용될 수 있을 것이다.