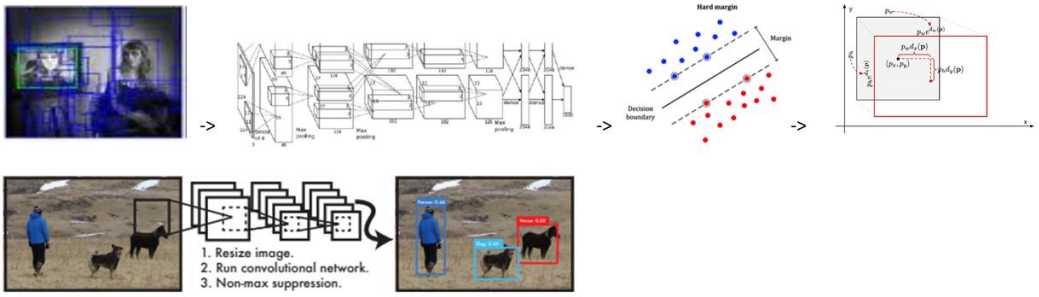
YOLO v1

YOLO

- YOLO (You Only Look Once)

1. 딱 한 번만 본다
2. 통합한다 (One-stage-method)
3. 빠르다



기존의 RCNN을 정리하면 이렇다.

(2000개의 ROI 생성 -> CNN 특징 추출 -> SVM 분류 -> Box Regressor로 박스 조정)

하지만 지금부터 설명할 YOLO는 이 과정이 아래처럼 정리된다.

(이미지 -> CNN을 통한 분류 및 박스 조정)

이 과정을 비교해보면 YOLO는 다음과 같은 특징을 지닌다.

1. 딱 한 번만 본다. (2000개의 영상을 보지 않고 1개의 원본 영상만 본다.)

2. 통합한다. (객체의 위치 및 바운딩 박스 찾기와 클래스 분류를 한 번에 한다.)

3. 빠르다. (1, 2과정을 통해 어떤 객체 검출 프로세스보다 빠르다.)

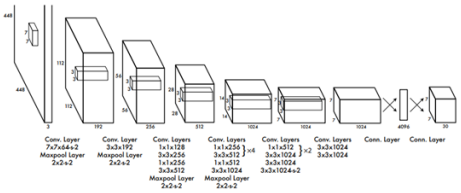
YOLO

- YOLO (You Only Look Once)

1. 이미지를 SxS 그리드로 분할한다. (논문에서 S=7)
2. 이미지 전체를 CNN에 넣고 특징 추출을 통해 Prediction Tensor (SxSx(B*5+C))생성
3. 바운딩 박스 및 클래스를 조정한다.

T. 인퍼런스 과정에서는 Non Maximum Suppression을 진행하고 결과를 보인다.


S × S grid on input




Final detections

YOLO는 크게 세 단계로 나타낼 수 있다. 인퍼런스 과정에서는 NMS 알고리즘이 추가로 진행된다.

각 단계에 대해 차근차근 알아 보겠다.

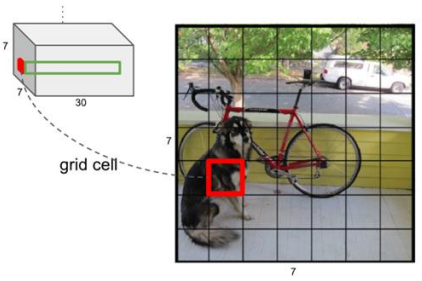
YOLO

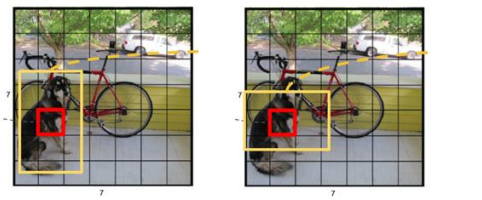
- YOLO (You Only Look Once)

1. 이미지를 SxS 그리드로 분할한다.
2. 이미지 전체를 CNN에 넣고 특징 추출을 통해 Prediction Tensor (SxSx(B*5+C))생성

S: 그리드 숫자, B: 그리드 별로 갖는 바운딩 박스의 개수, 5: 바운딩 박스의 정보(x,y,w,h,c_s), C: 클래스 개수만큼의 조건부 확률

Ex) S:7, B:2, C:20 -> (7x7x(2x5+20)) -> (7x7x30)




그리드 안에 바운딩 박스의 중앙점(x,y)이 위치해있음

YOLO는 영상을 SxS 그리드로 먼저 분할한다.

이후 이미지 전체를 CNN에 넣고 특징 추출을 통해 Prediction Tensor를 생성한다.

Prediction Tensor는 (B*5+C)개만큼의 정보가 셀 개수(S^2)만큼 있다.

이 때 B는 그리드 별로 갖는 바운딩 박스의 개수를 의미한다. 논문에서는 B를 2라고 설정했는데

그림 예시를 보면 두 바운딩 박스의 중앙점(x,y)이 한 그리드 안에 위치해있는 것을 확인할 수 있다.

5는 바운딩 박스의 정보를 나타낸다. 바운딩 박스의 정보는 중점x,y, width, height, confidence score를 의미한다.

c는 클래스 개수만큼의 조건부 확률을 의미한다. 논문에서 클래스 개수는 20으로 선정했다.

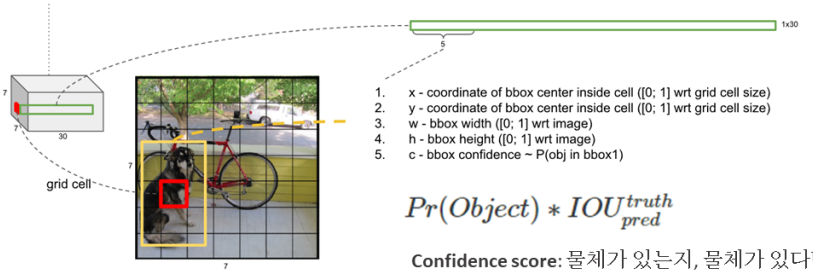
즉 한 그리드 당 2*5+20=30개의 (바운딩박스,클래스) 정보를 지닌다고 보면 된다.

YOLO

- YOLO (You Only Look Once)

  S: 그리드 숫자, B: 그리드 별로 갖는 바운딩 박스의 개수, 5: 바운딩 박스의 정보(x,y,w,h,c_s), C: 클래스 개수만큼의 조건부 확률

  Ex) S:7, B:2, C:20 -> (7x7x(2x5+20)) -> (7x7x30)



1. x - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
2. y - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
3. w - bbox width ([0; 1] wrt image)
4. h - bbox height ([0; 1] wrt image)
5. c - bbox confidence ~ P(obj in bbox1)

$$Pr(Object) * IOU_{pred}^{truth}$$

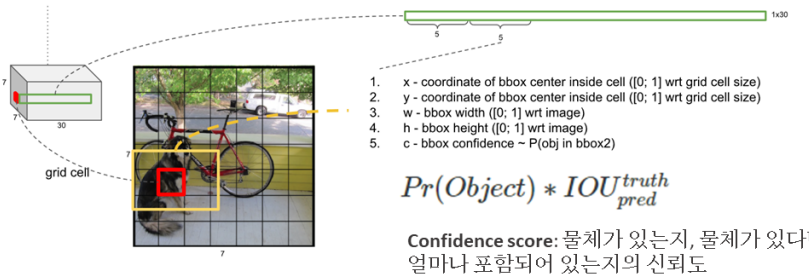
Confidence score: 물체가 있는지, 물체가 있다면 바운딩 박스에 얼마나 포함되어 있는지의 신뢰도

YOLO

- YOLO (You Only Look Once)

  S: 그리드 숫자, B: 그리드 별로 갖는 바운딩 박스의 개수, 5: 바운딩 박스의 정보(x,y,w,h,c_s), C: 클래스 개수만큼의 조건부 확률

  Ex) S:7, B:2, C:20 -> (7x7x(2x5+20)) -> (7x7x30)



1. x - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
2. y - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
3. w - bbox width ([0; 1] wrt image)
4. h - bbox height ([0; 1] wrt image)
5. c - bbox confidence ~ P(obj in bbox2)

$$Pr(Object) * IOU_{pred}^{truth}$$

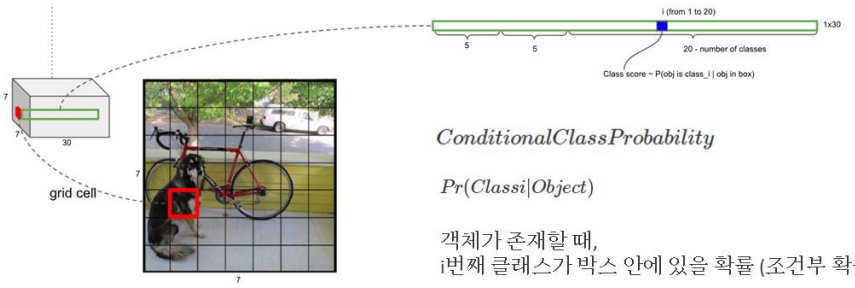
Confidence score: 물체가 있는지, 물체가 있다면 바운딩 박스에 얼마나 포함되어 있는지의 신뢰도

YOLO

- YOLO (You Only Look Once)

  S: 그리드 숫자, B: 그리드 별로 갖는 바운딩 박스의 개수, 5: 바운딩 박스의 정보(x,y,w,h,c_s), C: 클래스 개수만큼의 조건부 확률

  Ex) S:7, B:2, C:20 -> (7x7x(2x5+20)) -> (7x7x30)



$ConditionalClassProbability$

$Pr(Classi|Object)$

객체가 존재할 때,
i번째 클래스가 박스 안에 있을 확률 (조건부 확률)

한 그리드(셀)에 대해 30개의 정보가 무엇인지 나타내는 그림들이다. (B:2, C:20)

처음 5개는 첫 번째 바운딩 박스의 정보이다.

이 때 다섯번째로 Confidence Score가 저장되는데, 이것은 바운딩 박스에 물체가 있는지, 물체가 있다면 바운딩 박스에 얼마나 포함되어 있는지를 나타내는 신뢰도이다.

그 다음 5개는 두 번째 바운딩 박스의 정보이다.

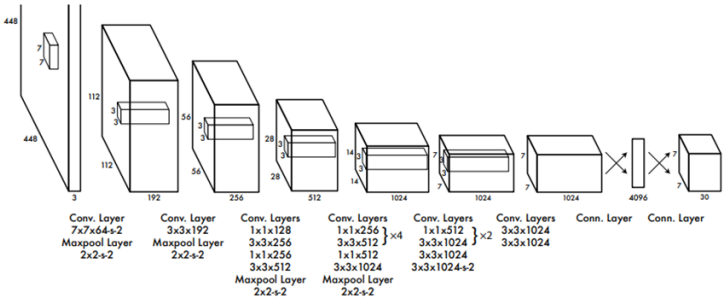
그 다음 20개는 각 클래스의 조건부 확률 정보이다.

Pr(Classi|Object) (i=1~20)가 저장이 되는데 이것의 의미는

객체가 존재할 때 i번 클래스가 박스에 있을 확률을 나타낸다.

YOLO

- YOLO (You Only Look Once)



GooLeNet for Image Classification 모델을 기반으로 함 (24 Convolutional Layers + 2 Fully Connected Layers)
1x1 Convolution Layer를 번갈아 사용하면서 특징 공간이 줄어드는 것이 특징

이번에는 YOLO에서 사용하는 CNN 구조를 알아 보겠다.

YOLO는 GooLeNet을 기본 네트워크로 사용한다. 즉 학습한 GooLeNet을 파인튜닝해서 사용하겠다는 의미이다. GoogLeNet의 inception을 가져와 1x1 conv를 활용하여 연산량을 줄이려고 하는 것을 볼 수 있다.

사실 GoogLeNet과 완전히 동일하지는 않고 살짝 변형된 네트워크인데, 24개의 Convolution Layer와 2개의 Fully Connected Layer 구조를 지녔다는 점이 특징이다.

앞의 Layer는 pre-trained된 것을 그대로 쓰고 나머지 2개 레이어로 Detection을 한다는 의미

YOLO

- YOLO (You Only Look Once)

loss function:

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3)$$: class에 대한 loss

λ_{coord} λ_{noobj} : 바운딩 박스의 변수에 대한 값을 더 반영, 객체가 없는 영역에 대한 값을 덜 반영하기 위한 하이퍼 파라미터 (객체가 없는 영역이 훨씬 많기 때문)

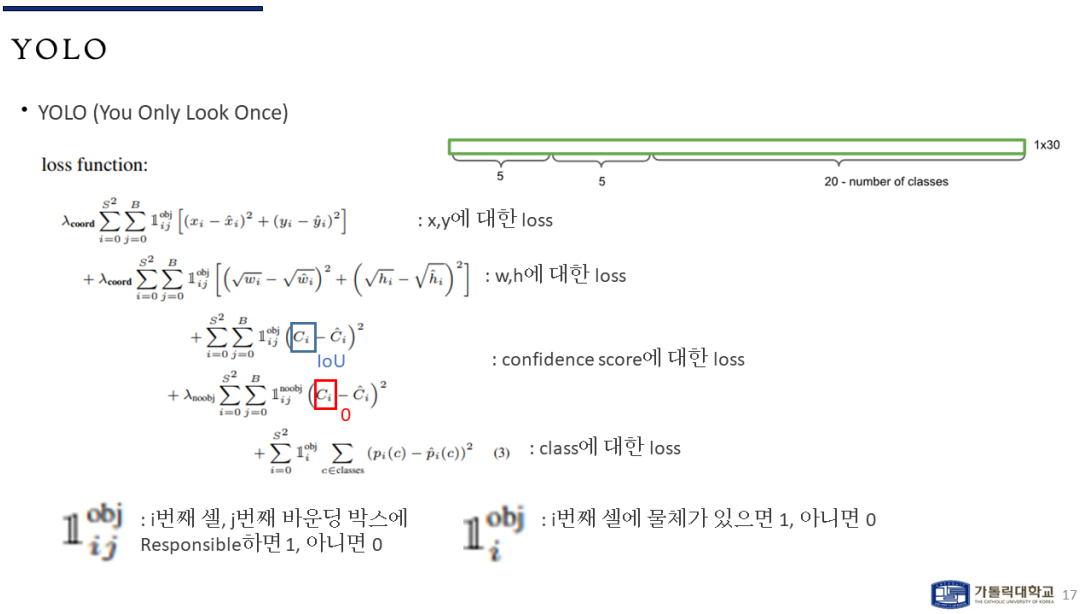
YOLO의 loss function은 5개의 loss가 합쳐진 구조라고 볼 수 있다. 첫 번째는 바운딩 박스의 중점(x,y)을 얼마나 잘 학습했는지 평가하는 loss, 두 번째는 w,h에 대한 loss, 세 번째와 네 번째는 confidence score에 대한 loss인데 객체가 있는 케이스와 객체가 없는 케이스를 나눠서 평가한 것이다. 마지막 class에 대한 loss이다.

Loss는 많이 쓰이는 SSE(오차 제곱합)를 채택해서 얼마나 적게 틀렸는지를 평가하게 된다. 참고로 w(너비)와 h(높이)는 root가 씌워 졌는데 이것은 큰 box의 오차가 작은 box의 오차보다 큰 가중치를 받지 않게 하기 위함이다. (root없으면 큰 box의 오차가 훨씬 커서 큰 box기준으로 학습이 더 많이 됨)

주의 깊게 바라볼 점은 **Lamda coord와 Lambda noobj**이다. 이것은 사용자가 직접 설정하는 하이퍼 파라미터인데, 바운딩 박스의 변수 값을 더 반영하고 객체가 없는 바운딩 박스의 변수 값은 덜 반영하는 것을 권장하고 있다. 따라서 논문에서 전자는 5, 후자는 0.5로 지정한다.

이렇게 하는 이유는 객체가 없는 바운딩 박스가 훨씬 많기 때문에 객체가 없는 바운딩 박스에 대해서만 많이 학습할 가능성이 커지기 때문이다.

즉 소수에 해당하는 객체가 있는 바운딩 박스는 잘 학습이 되지 않아서 객체 검출 성능이 현저히 낮게 된다. 이러한 데이터 불균형 문제를 하이퍼 파라미터로 해결할 필요성이 있다.



두 번째로 주의 깊게 바라볼 점은 **1 함수들**이다. 1함수들은 (1 obj ij)와 (1 obj i)로 나눠지게 된다. i는 셀에 대한 인덱스, j는 바운딩 박스에 대한 인덱스에 해당한다.

두 함수로 나누는 이유는 바운딩 박스를 고려할 필요가 없는 loss도 존재하기 때문이다. 바로

class에 대한 loss이다. 이 loss는 각 셀(그리드)에 대한 Pr(c)에만 관심이 있다.

1 함수는 기본적으로 바운딩 박스에 객체가 있으면 1, 없으면 0을 주면서 객체가 있는 경우에만 학습이 되도록 설정하는 역할을 한다.

물론 (1 noobj ij)같은 경우 객체가 없으면 1, 있으면 0이 된다.

다만 여기서 responsible하다는 의미를 알아야 한다. 여기서 responsible하다는 것은 i번째 셀이 객체가 있는 것과 더불어 j번째 바운딩 박스가 가장 IoU가 높았을 때 responsible하다고 표현한다. 즉 B가 2이기 때문에 두 바운딩 박스 중 한 박스만 responsible하다는 의미이다. 결론적으로 학습을 할 때는 한 박스만을 이용해서 학습을 하겠다고 생각하면 된다.

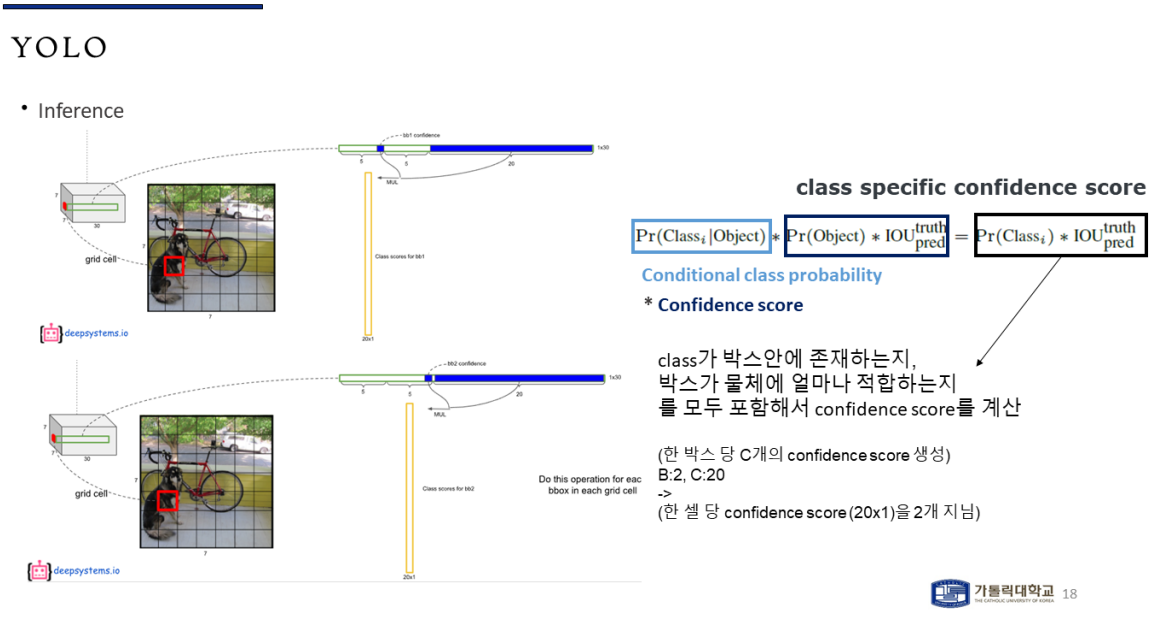
세 번째로 주의 깊게 바라볼 점은 **confidence score의 정답**이다.

객체가 있을 때의 confidence score는 IoU랑 같아야 할 것이다.

객체가 없을 때의 confidence score는 0이랑 같아야 할 것이다.

왜냐하면 confidence score= Pr(Object)*IoU인데 학습이 잘 됐다면 객체가 있을 때의 Pr(Object)는 1이 돼야 하고 객체가 없을 때의 Pr(Object)는 0이 돼야 하기 때문이다.

따라서 Pr(Object)가 1혹은 0이 되도록 학습이 진행되어야만 한다.



이번에는 YOLO의 인퍼런스 과정에 대해 살펴보겠다.

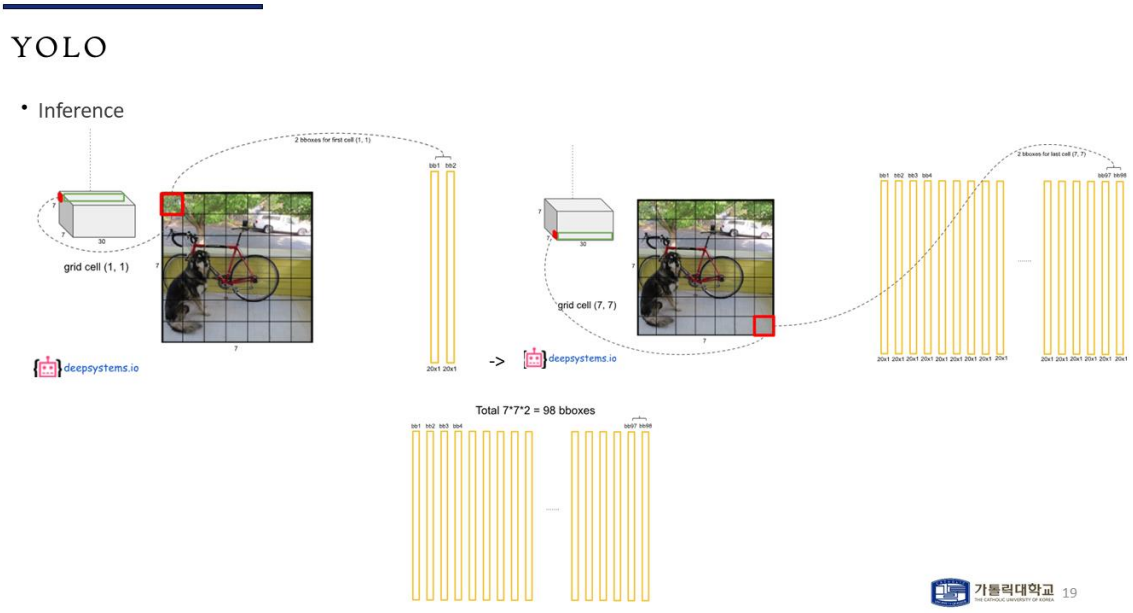
인퍼런스를 할 때는 새로운 confidence score가 제안된다.

바로 기존의 클래스 조건부확률과 confidence score를 곱한 것인데 이 의미는

(class가 박스안에 존재하는지, 박스가 물체에 얼마나 적합한지를 모두 포함한 점수)라고 생각하면 된다.

한 셀에 대해서 confidence score는 b개(바운딩 박스 개수)만큼 있고 클래스 조건부 확률은 c(개(클래스 개수)있으니 이 둘을 곱하면 한 셀에 대해서 (c*1)*b개의 confidence score를 가진다고 보면 된다.

논문에서 c=20, b=2니까 한 셀당 confidence score(20*1)을 2개 가진다고 생각하면 된다.



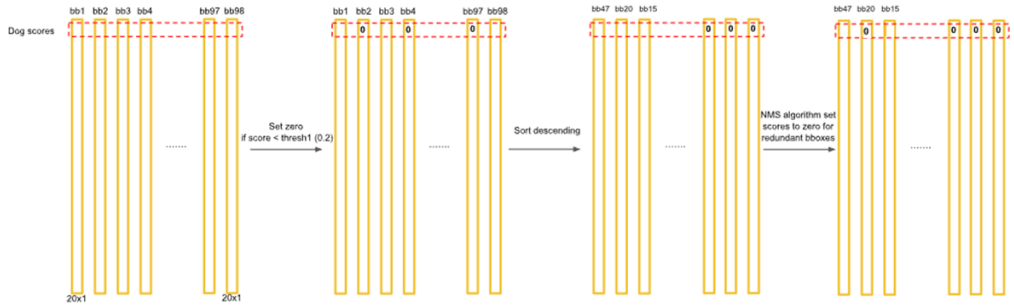
즉 49개의 셀에 대해서 confidence score를 모두 계산하면

confidence score(20*1)을 98개 가진다고 생각하면 된다.

다른 말로 98개의 bbox에 대한 confidence score(20*1)를 가진다는 의미이다.

YOLO

- Inference



1. Threshold를 넘지 않는 score -> 0 처리
2. 내림차순 정렬
3. NMS를 통해 0 Score 추가 -> 논문에서는 98개의 박스 중 2개만 남게 됨 (Dog Box)

   Non Maximum Suppression: IoU를 기반으로 일정 수준 겹쳐있는 박스들을 모두 제거

confidence score을 구하는 것은 필요 없는 박스를 제거하는 것에 의의가 있다.

같은 행은 같은 class를 나타내게 되는데 예를 들어 첫 행은 dog에 대한 confidence scores이다.

각각의 행에 대한 점수들로 필요 없는 박스를 지워 나가는 것이다.

첫 행인 Dog Scores에 대해 살펴 보겠다.

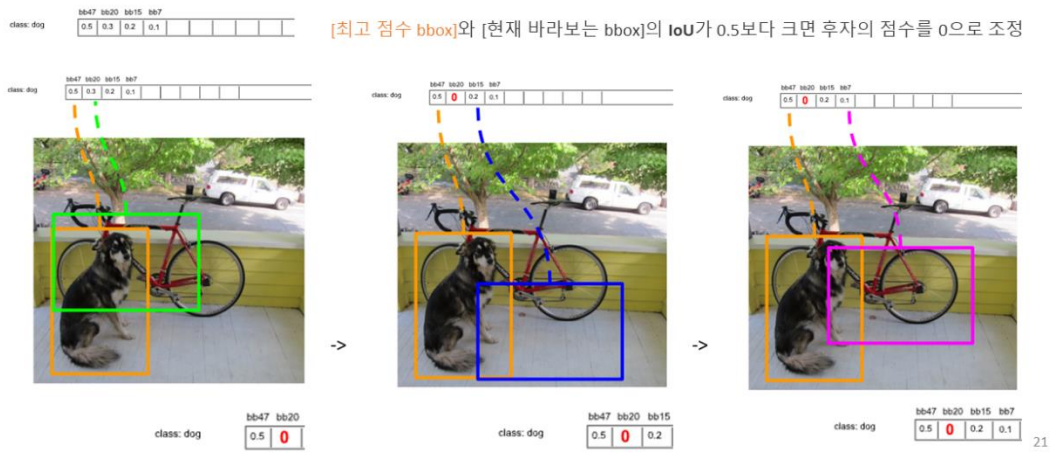
첫 번째로 일정 수준의 임계치를 넘지 않는 박스에 대해서는 score를 0처리한다. score를 0처리한다는 것은 해당 dog 관련 box에서 제거하겠다는 의미이다.

두 번째로 내림차순을 통해 점수들을 확인한다.

세 번째로 NMS 알고리즘을 통해 겹쳐있는 박스들을 제거한다. 이 때도 물론 Dog 관련 box에서 제외하겠다는 의미이다.

YOLO

- Non Maximum Suppression



NMS는 겹치는 박스를 제거하기 위한 알고리즘이다.

위 사진은 Dog Class를 기준으로 박스를 제거하는 예시이다.

박스들은 점수를 기준으로 내림차순되어 정렬돼 있다.

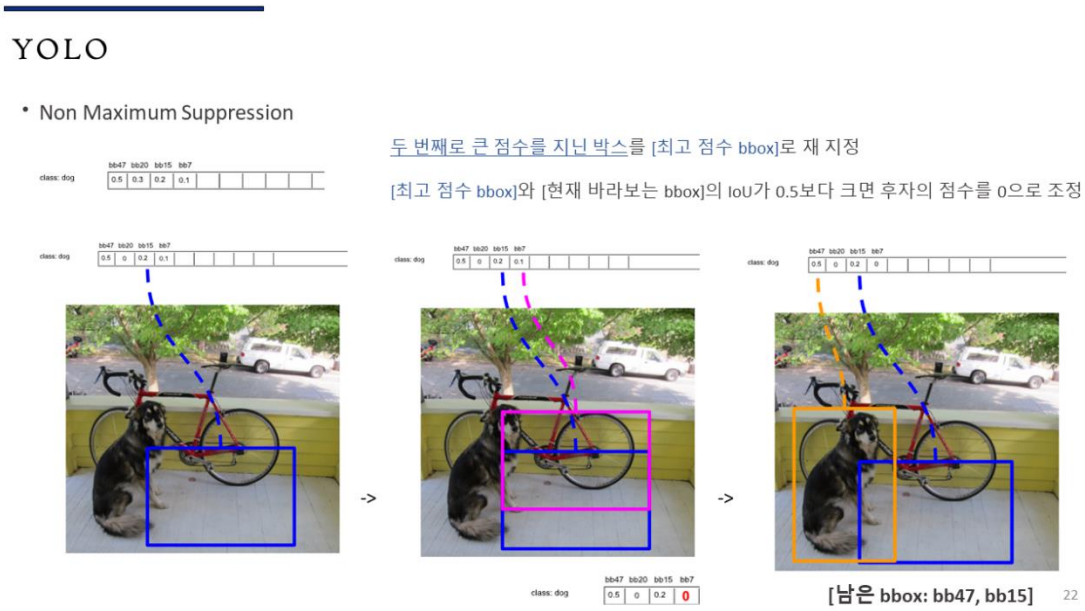
Dog Class에 대해 가장 높은 점수를 지닌 박스는 주황색 박스이다. 이 박스를 [최고 점수 bbox]로 지정한다. 그리고 이후의 박스들을 차례차례 [현재 바라보는 bbox]로 지정하고 두 bbox간 IoU가 0.5면 [현재 바라보는 bbox]를 삭제하는 알고리즘이 된다.

첫 번째 비교는 IoU가 0.5이상이라 Score가 0이 되었다.

두 번째 비교와 세 번째 비교는 IoU가 0.5미만이라 Score는 유지된다.

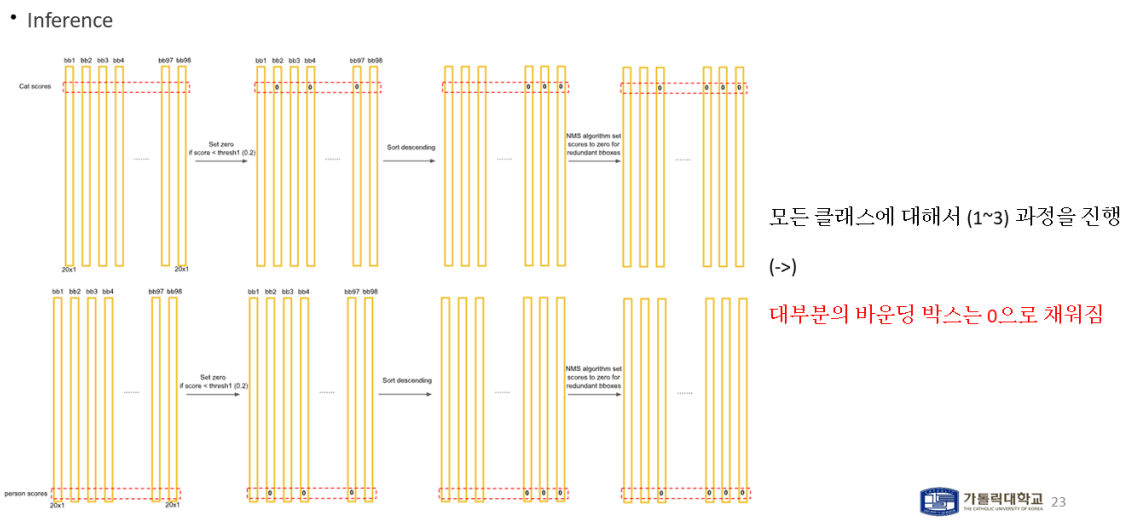
다섯번째 박스부터는 score가 처음부터 0이므로 비교가 무의미하다.

(아예 개와 관련없는 박스)



모든 박스에 대해 조사가 끝나면 두 번째로 큰 점수를 지닌 박스를 [최고 점수 bbox]로 지정한다. 그리고 똑같은 알고리즘을 진행한다.

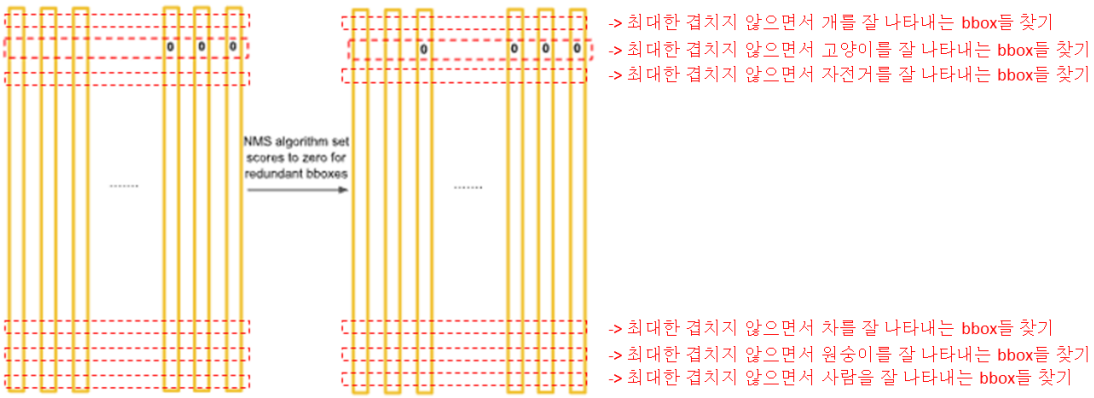
Dog Class에 대해 NMS가 끝나고 남은 박스는 두 개였다고 한다. (주황, 파랑)



이 과정을 모든 클래스에 대해서 진행한다. 그럼 대부분의 바운딩 박스는 0으로 채워지게 된다. 각 행마다 Threshold 및 NMS처리가 되면서 매우 적은 양의 bbox들만 0보다 큰 score를 가지게 될 것이다. 그리고 클래스가 20개라고는 하나 영상에서 20 종류의 객체가 나오지 않았으므로 어떤 행들은 모두 0이 될 것이다.

YOLO

- Inference

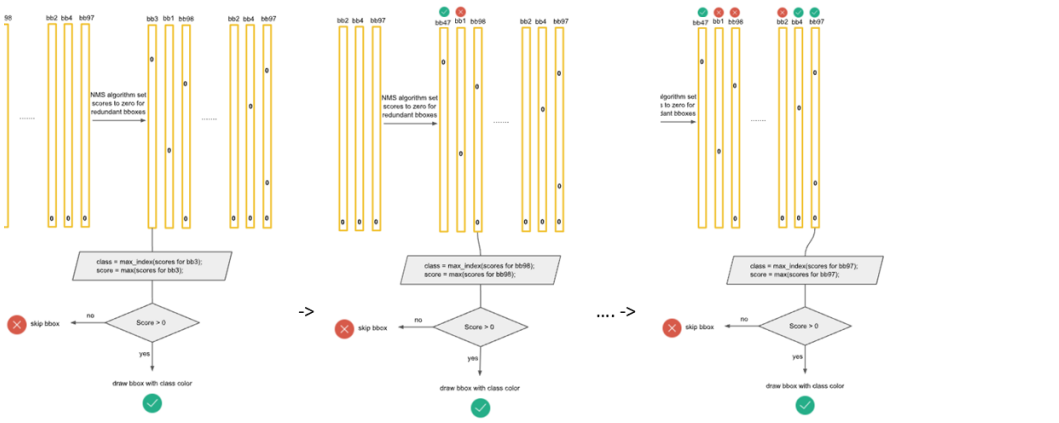


전체에 대해서 조사한다는 것은 (최대한 겹치지 않으면서 클래스를 잘 나타내는 bbox)를 찾는다는 의미와 같다. 근데 만약 한 bbox가 (최대한 겹치지 않으면서 개를 잘 나타내는 bbox)임과 동시에 (최대한 겹치지 않으면서 고양이를 잘 나타내는 bbox)로 선정되면 무슨 일이 일어날까? 해당 bbox는 (개,고양이)로 분류되는 것일까? 이런 문제를 막기 위해 (최종 결정) 작업이 추가된다.

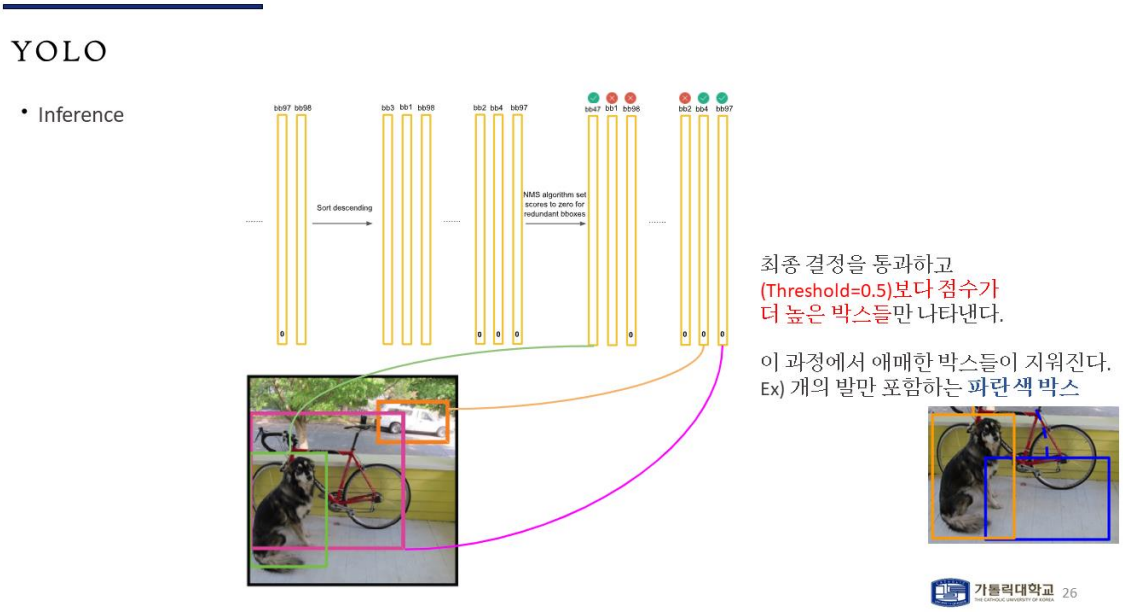
YOLO

- Inference

박스 중 가장 Score가 큰 값이 0보다 크면 check, 아니면 pass



최종 결정 작업은 각 bbox마다 가장 큰 score를 기준으로 해당 score가 0보다 크면 check, 작으면 pass하겠다는 것이다. 즉 각 bbox마다 가장 확률이 높은 한 클래스를 기준으로 분류하겠다는 의미이다.

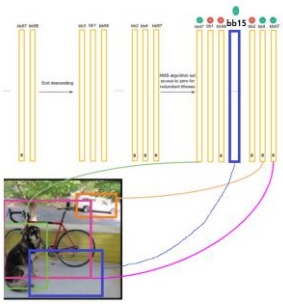


이 최종 결정을 통과하고 (Threshold=0.5)보다 점수가 더 높은 박스들만 나타내면 인퍼런스가 끝난다.

사실 이 Threshold는 사용자가 최종적으로 정하기 나름이다.

다만 Threshold를 너무 낮게 잡으면 애매한 박스들도 보일 수 있다.

예시이기는 하나, NMS를 설명하는 과정에서 Dog Class와 관련된 bbox들을 제거할 때 파란 색 박스(bb15)는 제거되지 않았다. 만약 bb98과 bb2사이에 위 bb15가 최종 결정되었고, 이 때의 클래스가 dog였다고 가정해보자.



컴퓨터 입장에서는 어느 정도 개를 나타내기도 하면서 겹치지 않았기 때문에 제거하지 않은 것이다. 하지만 인간이 보기에는 애매한 박스임이 분명하다.

이런 애매한 박스들을 마지막 Thresholding을 통해 제거하는 것이다.

Experiments

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

pascal voc 데이터를 이용해서 real-time을 테스트해 본 결과 YOLO가 가장 성능이 좋았다.

일반 yolo보다 적은 conv layer를 사용하는 fast yolo같은 경우, Map는 떨어지지만 FPS가 상당히 높은 것을 확인할 수 있다.

반면 R-CNN 시리즈는 속도를 높이기 위한 RPN같은 알고리즘을 적용했지만 FPS가 많이 낮다고 말한다.

또한 속도를 높이려면 Detection pipeline 각각에 대해 최적화를 진행하는 것보다 파이프라인 자체를 버리고 속도를 위한 설계를 하는 것이 더 좋다고 말한다.

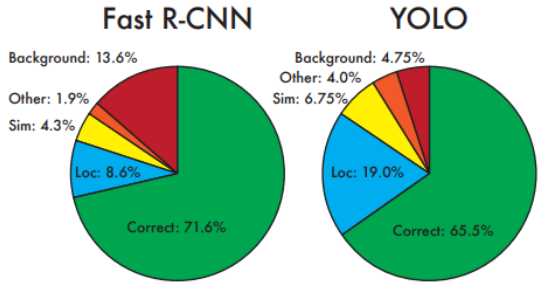


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

YOLO랑 Fast R-CNN을 비교하면 background에 대해서 YOLO가 훨씬 잘 찾는 것을 볼 수 있다. 이것은 각각의 셀에 대해서 바운딩 박스를 결정하기 때문에 (배경에 대한 바운딩 박스가 많으므로) 당연한 결과라고 생각한다.

이렇게 배경을 잘 찾는 것을 장점으로 생각해서 Fast R-CNN이랑 combine을 하면 상당한 성능을 보인다고 한다. 그리고 이렇게 합쳤을 때 일반 Fast R-CNN이랑 비교해서 연산 시간이 크게 늘어

나지 않았다고 한다. 성능을 잠깐 확인해보자면 Fast R-CNN에서 가장 Map가 좋았을 때가 66.9였는데 YOLO랑 합쳐져서 배경을 더 잘 구분하고 나니까 70.7까지 올랐다고 한다.

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	**66.9**	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	**75.0**	**3.2**

VOC 2012 test	mAP
MR_CNN_MORE_DATA [11]	**73.9**
HyperNet_VGG	71.4
HyperNet_SP	71.3
Fast R-CNN + YOLO	70.7
MR_CNN_S_CNN [11]	70.7
Faster R-CNN [28]	70.4
DEEP_ENS_COCO	70.1
NoC [29]	68.8
Fast R-CNN [14]	68.4
UMICH_FGS_STRUCT	66.4
NUS_NIN_C2000 [7]	63.8
BabyLearning [7]	63.2
NUS_NIN	62.4
R-CNN VGG BB [13]	62.4
R-CNN VGG [13]	59.2
YOLO	57.9

YOLOv2 – YOLO9000

yolo v2로 detection을 9000개 클래스에 대해 하겠다.

introduction을 보면 많은 detection method들이 적은 object set에 제한돼있다고 말하면서 classification할 때처럼 많은 클래스에 대해서 detection을 할 줄 알아야 한다고 말한다.

그러나 클래스를 많이 늘리면 라벨링 문제에서 많은 비용이 들어간다는 문제점이 있다.

따라서 어떻게 detectioin data set을 확장할 것인가, classification data와 detection data를 섞어서 joint training하는 방법에 대해서 소개를 할 것이다.

또한 첫 번째로 YOLO를 v2로 업데이트하고 그 모델에 9000개 클래스에 대한 학습을 진행할 것이라고 합니다.

YOLO v1의 단점은

다른 알고리즘에 비해서 (1)성능(AP)이 떨어지고 (2)recall이 낮다는 점이다. 즉 object를 잘 못 찾았다고 합니다. (recall은 실제 정답 중 기계가 낸 정답의 비율인데, recall이 낮다는 것은 실제 정답을 많이 못 찾았다고 해석하면 됨)

Better

1. Batch Normalization

conv레이어에 bn을 사용해서 성능이 2% 향상됨(vanishing 문제 해결), 오버피팅 문제도 개선

2. High Resolution Classifier

yolo는 pre-trained된 cnn(classifier)을 사용하는데 224x224크기의 ImageNet데이터로 학습된 모델을 사용한다. 그런데 detection을 하기 위해 448x448크기의 영상이 사용되므로 강제로 cnn입력을 바꾸게 된다. 따라서 네트워크가 갑자기 변한 input resolution에 대해 잘 적응을 하지 못 했다고 생각하였다. 그래서 해당 cnn 네트워크를 448x448 영상들로 10epoch학습을 해서 4% 성능 향상을 이루어 냈다고 한다.

3. Convolution with Anchor Boxes

- faster r-cnn이 anchor box들로 bbox를 미리 결정하는 것처럼 yolo도 conv layer상에서 anchor boxes를 사용한다고 한다.

- 입력 영상을 yolo1처럼 448x448을 쓰지 않고 416x416을 사용하였는데, cnn이 영상을 1/32로

downscale하는데 이 때 feature map의 가로,세로 사이즈가 홀수가 되도록 만드는 것이다. 이건 feature map의 가운데에서 뽑아낸 bbox가 large object를 detect하는 경우가 많은데, 이 때 가운데가 하나의 셀이면 더 좋다고 한다. 따라서 강제로 홀수 사이즈가 되도록 한 것이다.

- grid마다 클래스를 예측하는 것이 아니라 모든 Anchor Box마다 클래스를 예측한다.

-> anchor box를 사용했을 때 AP는 좀 낮아졌지만 Recall이 많이 좋아졌다고 한다. 물체를 찾은 것 중에서 정답을 맞추는 것(Precision이 높은 것)도 중요하지만 물체를 잘 찾는 것(Recall이 높은 것)도 중요하기 때문에 Anchor box를 사용한다.

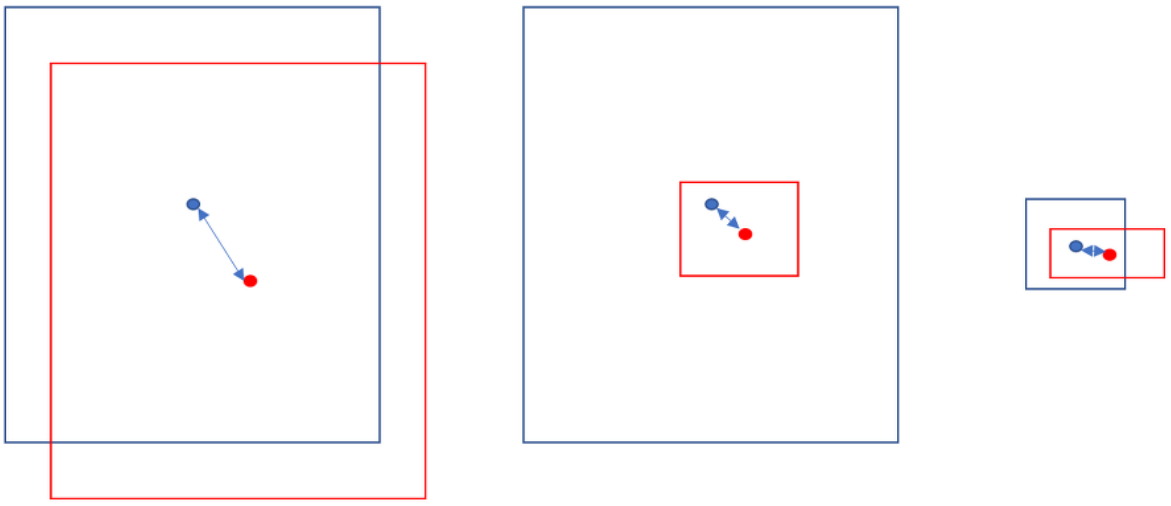
< 69.5 mAP -> 69.2 mAP / 81% -> 88% >

Anchor Box 정하기

미리 정의된 anchor box를 쓰는 것이 문제라고 생각함. 데이터에 잘 맞는 anchor box를 잘 찾는 것이 중요하다고 생각함 -> k means clustering을 사용해서 anchor box들을 결정 -> k값을 바꾸면서 AVG IoU를 조사한 결과 k=5일 때 가장 좋았다고 함

(내 생각에는 많은 오브젝트 센터 중에서 랜덤으로 k개 찍고 각 오브젝트 센터와 어울리는 anchor box를 1개씩 선정하는 것 같음. 그래서 k가 원래 구역의 개수인데 anchor box의 개수가 되는 것임)

k means clustering을 할 때는 단순히 오브젝터 센터와 anchor box의 중심간의 거리가 낮은 것을 선택한 것이 아니라 IOU가 높으면 (d가 낮아져서) 선택하는 방식을 채택했다고 합니다. 그렇지 않고 그냥 Euclidian공식을 사용하면 아래 그림의 (중간, 오른쪽 anchor box)처럼 IOU가 낮은 anchor box들이 grouping될 확률이 높아집니다.



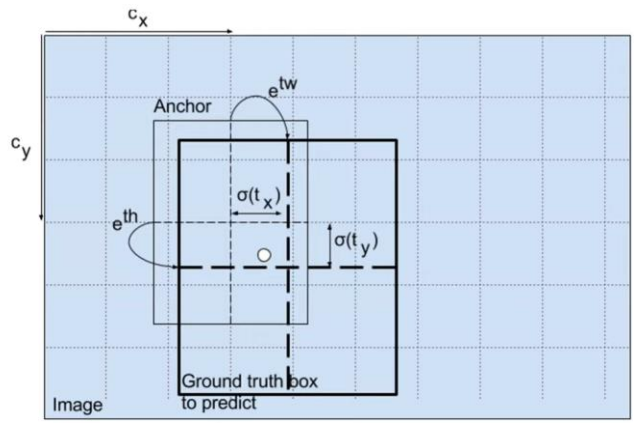
Faster R-CNN의 RPN과 비교했을 때 Avg IOU도 더 좋았다고 한다.

5. Direct Location Prediction

RCNN에서 제안한 box regressor은 매핑 함수 dx(P)의 리턴 값이 어떤 값이든 될 수 있으므로 G^ 이 초기에는 Ground Truth의 위치와 많이 다를 수 있다고 이야기한다.

따라서 시그모이드를 이용한 새로운 수식을 이용해서 Ground Truth와 관련된 일정 범위 내에서만 움직이도록 하는 것을 제안했다.

아래 그림을 보면 알겠지만 예측 Bbox 중심점(C_x,C_y)에 이동량인 sigmoid(t_x)를 더하면 시그모이드 함수에 의해 중심이 한 칸을 움직이지 못 한다.



$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

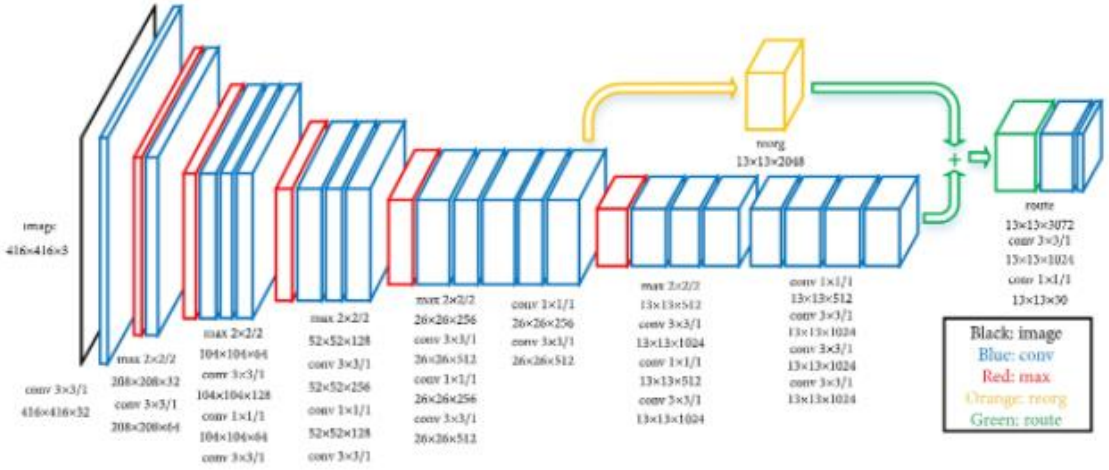
앞에서 정한 Anchor Box로 Direct Location Prediction을 해서 올바른 위치로 박스가 움직이도록 한다.

➔ 일반 Anchor Box 대비 5% 성능 향상

6. Fine-Grained Features

기존 yolo는 마지막 layer의 feature map만 사용해서 abstract한 정보만을 지닌다. 따라서 상위 레이어의 feature map을 하위 feature map에 합쳐주는 passthrough layer를 도입했다. 예를 들어 26x26x512인 feature map을 13x13x2048로 리스케일하고 이것을 13x13x1024 feature map에 추가하여 13x13x3072크기의 feature map을 만드는 것이다. 그러면 abstract하면서도 detail한 특징들이 모두 담긴다. -> 작은 객체도 잘 검출했고 성능도 1% 오름

(input과 가까운 feature들은 작고 디테일한 특징들이 있고 output과 가까운 feature들은 크고 추상적인 특징들이 있다. 따라서 앞쪽 feature map에 작은 object에 대한 특징들이 있을 것이고 뒤쪽 feature map에 큰 object에 대한 특징들이 있다. 따라서 두 object를 모두 잘 검출하기 위해서는 앞뒤의 feature들을 합칠 필요성이 있다.)



7. Multi-Scale Training

fc layer를 없애서 다양한 input 사이즈를 받을 수 있다. 학습할 때 배치 10번 돌 때마다 랜덤하게 {320,352,...608} 크기의 영상 중 하나를 input으로 집어 넣어서 다양한 스케일의 영상에도 적응이 가능하도록 만들었다.

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	**78.6**	40

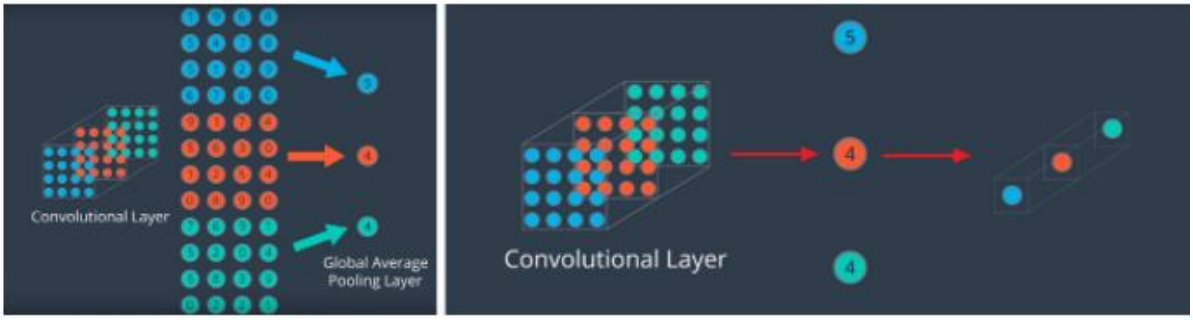
위의 표에서 확인할 수 있듯이 입력 이미지의 크기가 작은 경우 더 높은 FPS를 가지며, 입력 이미지의 크기가 큰 경우 더 높은 mAP 값을 가지게 된다.

전의 detector보다는 성능 향상이 되었지만 여전히 trade-off관계를 보인다. (fps랑 mAP)

Faster

Darknet-19라는 네트워크를 디자인해서 사용함 (VGGNet이랑 비슷함 – 대다수 Conv Filter를 3x3으로 설정)

마지막 layer에 **global average pooling**을 사용하는데 global average pooling은 (height,width,channel)형태의 feature를 (channel,)형태로 간단하게 만들어 버린다. GAP 연산 결과가 1차원 벡터가 되기 때문에 최종 출력에 FC Layer대신 사용할 수 있고 이 점에서 많은 파라미터 수를 감소시켜 속도를 향상시켰다.



파라미터 수는 줄어도 정확도는 top 5 accuracy가 91.2%라고 한다. (ImageNet데이터셋 기준)

Training for classification

ImageNet데이터를 가지고 160 epochs학습함 (Initial training: 224x224) /

resizing한 데이터를 가지고 10 epochs 파인튜닝함 (448x448)

➔ higher resolution에 대한 top-5 accuray: 93.2%

Traing for detection

GAP은 지우고, 3x3x1024 conv layer와 1x1 conv layer 추가

detection을 할 때는 5 box with (5개의 coordinates(x,y,w,h,c_s)와 20개의 class probability)를 예측함 -> 1x1 conv layer에서 필터 수를 125개로 지정 (위의 125개의 정보를 파악해야 됨)

Stronger

이 부분부터는 어떻게 detection data와 classificiation data를 통합하고 이를 어떻게 이용하는지 나타낸다.

detection dataset은 좀 일반적인 라벨을 다루는 반면 classification dataset은 좀 더 깊은 범위의 라벨까지 다룬다. 예를 들어 detection dataset의 라벨이 '개'일 때, classification dataset의 라벨은 '요크셔 테리어'이다. 우리는 두 데이터 셋을 모두 이용하고 싶기 때문에 일관된 dataset이 필요하다. 그냥 단순하게 coco와 imagenet을 결합하면 문제가 발생한다. 예를 들어 '요크셔 테리어'와 '개'가 완전 독단적인 클래스로 학습을 하게 된다는 의미이다.

따라서 일단 ImageNet dataset으로 word tree라는 것을 만들었다. 이 word tree는 각 범주 간 계층적인 구조를 나타낸다. 예를 들어 '동물-포유류-사냥개-테리어' 범주(노드)를 거쳐서 '요크셔 테리어'에 도달할 수 있다.

이러한 트리에서 특정 범주에 속할 확률은 루트 노드로부터 해당 범주의 노드까지의 조건부 확률의 곱으로 표현할 수 있다. (아래 사진 참고)

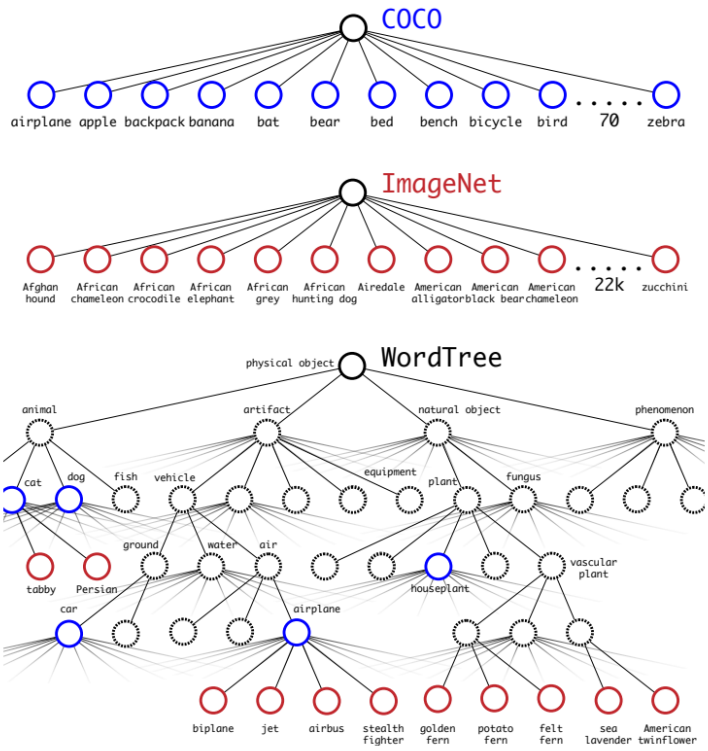
$$\begin{aligned} Pr(Yorkshireterrier) = Pr(Yorkshierterrier|terrier) \\ * Pr(terrier|huntingdog) \\ *...* \\ * Pr(mammal|animal) \\ * Pr(animal|physicalobject) \end{aligned}$$

Joint classification and detection

이렇게 word tree를 구축하면 두 dataset을 combination을 할 수 있다.

이 때, 두 dataset의 개수 차이가 크므로 oversampling으로 detection을 4정도 뽑고 classification을 1정도 뽑아서 개수를 맞춘다.

combination을 하면 detection dataset의 데이터들은 트리에서 높은 노드에 해당되고 classification dataset의 데이터들은 좀 더 낮은 위치의 노드에 해당된다.



이렇게 combination된 dataset을 이용하면 coco에 있는 데이터로 detection학습도 가능하고 word tree 덕분에 classification 시 더 많은 클래스를 분류할 수 있다

- training때 detection data가 들어오면 원래 loss function을 활용해 계산
- training때 classification data가 들어오면 원래 classification loss만 활용해 계산
- classification loss 계산 시 ground truth label의 그 계층이나 상위 계층만 역전파 /아래 계층에 내려가지 못하도록 아래 계층에 대한 예측할 시 error 부과

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

AP 결과를 확인해보면 동물은 성능이 잘 나오지만 선글라스와 같은 기타 사물은 성능이 잘 안

나온다. 이 이유는 Detection Dataset에 해당 사물에 대한 데이터가 없기 때문이다. (관련된 bounding box 정답이 아예 없음) 예를 들어 선글라스에 대해서는 선글라스의 상위 범주인 안경 따위도 detection data에 존재하지 않으므로 classification을 한다고 하더라도 bbox가 IoU Threshold를 못 넘어서 detection될 일이 없다.

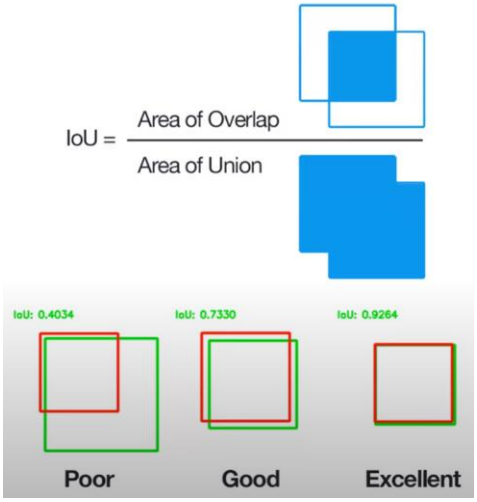
YOLOv3

An Incremental Improvement

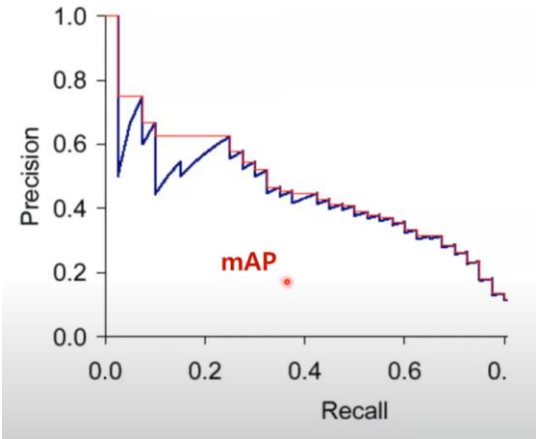
IoU: Area of Overlap / Area of Union

두 바운딩 박스의 교집합의 면적 / 두 바운딩 박스의 합집합의 면적

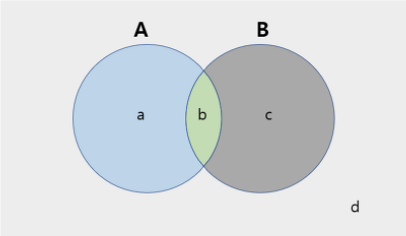
[두 바운딩 박스: 예측 박스, 정답 박스]



mAP: Precision과 Recall의 그래프는 trade-off 관계를 갖게 된다. Precision과 Recall에 대한 그래프 아래 면적을 AP라고 하고 이것을 클래스 별로 따로 따로 구해서 평균을 낸 것



(precision-recall-treade_off에 대해 이해를 도운 그림 설명)



벤다이어그램으로 두 관계를 생각해본다면
A : 실제 날씨가 맑은 날
B : 모델에서 예측한 날씨가 맑은 날
a : FN
b : TP
c : FP
d : TN

$$(Precision) = \frac{b}{b+c}$$
$$(Recall) = \frac{b}{a+b}$$

1. Bounding Box Prediction

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

YOLOv2에서 제안한 bounding box prediction을 그대로 이용한다. 그래서 이동량에 대한 tx,ty,tw,th를 예측하는 regression문제이기 때문에 squared error공식을 이용한다. yolo는 bounding box마다 objectness score를 가지고 있다. logistic regression으로 0에서 1사이의 값을 갖도록 예측한다. 다른 바운딩 박스보다 월등하게 ground truth와 overlap한다면 score는 1을 준다. 그리고 ground truth object에 대해 1개의 바운딩 박스만 가지는 것도 특징이다.

2. Class Prediction

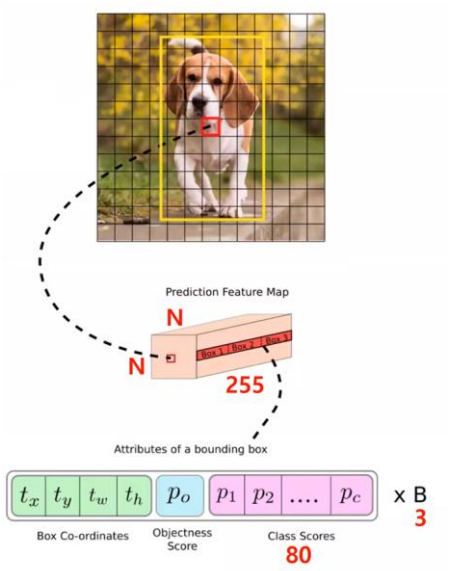
multilabel classification을 할 때 softmax를 이용하지 않는다. 대신에 training할 때 각 클래스에 대해서 binary cross-entropy loss를 사용한다. softmax를 사용하면 각 box는 정확히 한 클래스로만

분류를 하는 데 woman, personr같이 overlap이 필요한 box도 존재한다. 따라서 softmax를 사용하지 않는다.

3. Predictions Across Scales

yolo는 3가지 스케일에 대해 3가지 바운딩 박스를 사용한다. YOLO는 base feature extreactor에 몇몇의 conv layer를 추가하였는데 이 네트워크는 3차원 tensor를 예측한다.

예를 들어 아래 스케일에 대해서 3가지 바운딩 박스를 사용할 때 feature map을 추출하면 아래와 같다.



tensor는 NxNx[3*(4+1+80)]인데 4는 box coordinate, 1은 object score, 80은 코코 데이터셋 클래스 수를 의미한다. 이러한 box feature를 B개만큼 구하는 것이다.

바운딩 박스를 결정할 때는 k-means 클러스터링을 사용한다. 코코 데이터 셋에 대한 9개의 클러스터는 (10×13),(16×30),(33×23),(30×61),(62×45),(59× 119),(116 × 90),(156 × 198),(373 × 326)와 같이 되는데 이 중에서 앞에 세 개는 작은 스케일을 탐지하는데 사용되고 중간 세 개는 중간 스케일, 마지막 세 개는 큰 스케일을 탐지하는데 사용한다.

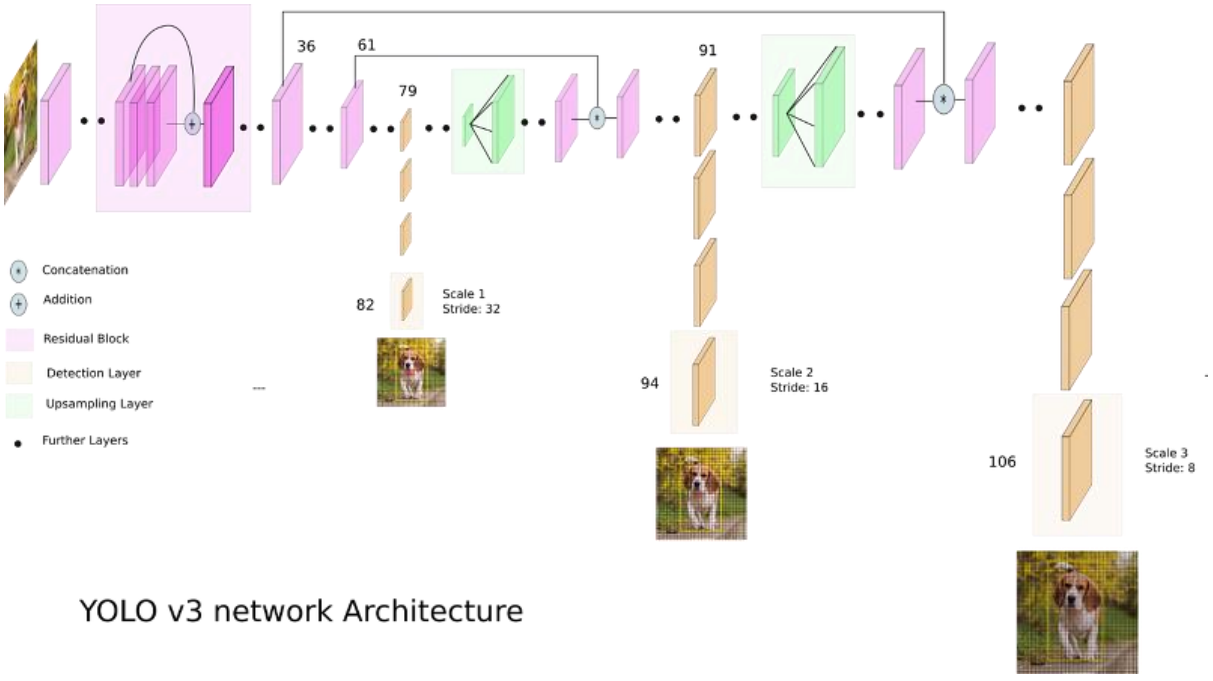
4. Feature Extractor

feature extractor는 새로운 네트워크를 사용하였다. Darknet 53이라는 것을 사용하였는데 FPS는 78로 빠르면서 성능은 ResNet-152와 비슷하다고 한다. 특히 1초에 할 수 있는 연산량이 다른 네트워크보다 1.5배 많다. 이 의미는 이 네트워크가 GPU를 더 잘 활용할 수 있다. ResNet같은 경우 성능은 좋지만 효율적으로 GPU를 활용하지 못해서 FPS가 낮은 것이라고 언급한다.

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

이 Darknet-53은 YOLOv2의 backbone인 Darknet-19에 residual network를 추가하였고 bottle neck 구조와 short-cut을 도입하였고 한다. 총 53개의 conv layer가 사용된다.

Network Architecture

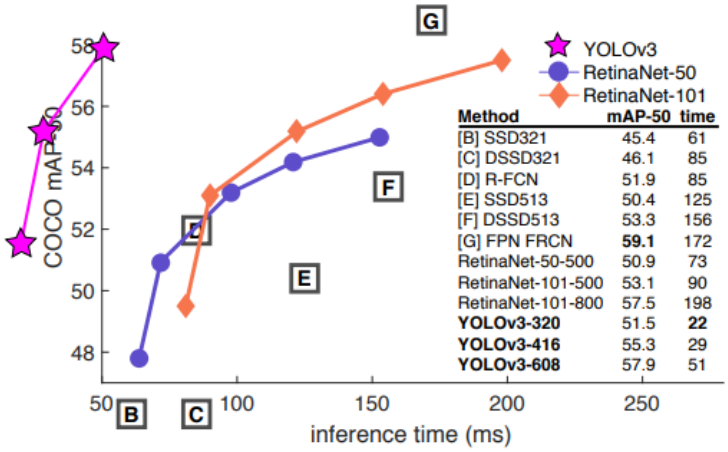


YOLO v3 network Architecture

YOLO V3 네트워크는 FPN과 비슷한 구조를 따른다.

일단 Darknet53을 통해 Feature를 파악하고 Detection Layer로 넘어가는데 초기에는 resolution이 제일 작으니(많은 conv연산 이루어졌으니) 큰 물체를 찾아낸다. 이후에는 이 feature map을 업샘플링한 것에 이전 feature map을 concatenation하면서 좋은 feature들도 가지면서 finer-grained한 feature도 가지게 된다. 따라서 좀 더 작은 물체를 찾아낼 수 있게 된다. 이 작업을 한 번 더하면서 총 3개의 스케일에 대해 detection을 진행할 수 있다.

Results



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

retinaNet이랑 mAP는 비슷해도 inference time은 훨씬 짧다는 결과를 만들었다. (mAP-50으로 측정)

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

여러 detection 알고리즘과 비교를 했을 때도 Two-stage-method인 FASTER R-CNN정도의 성능이 나왔다. 그리고 SSD보다는 확실히 성능이 좋다.