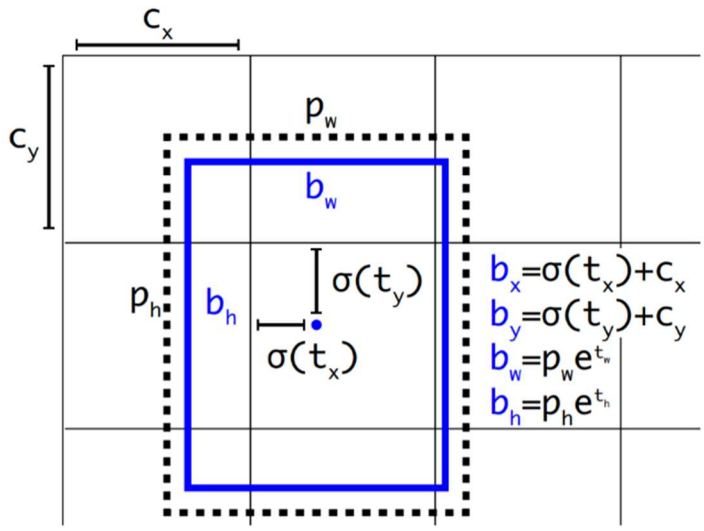
YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi
University of Washington

Introduction

작은 변화가 있을 것이라고 언급

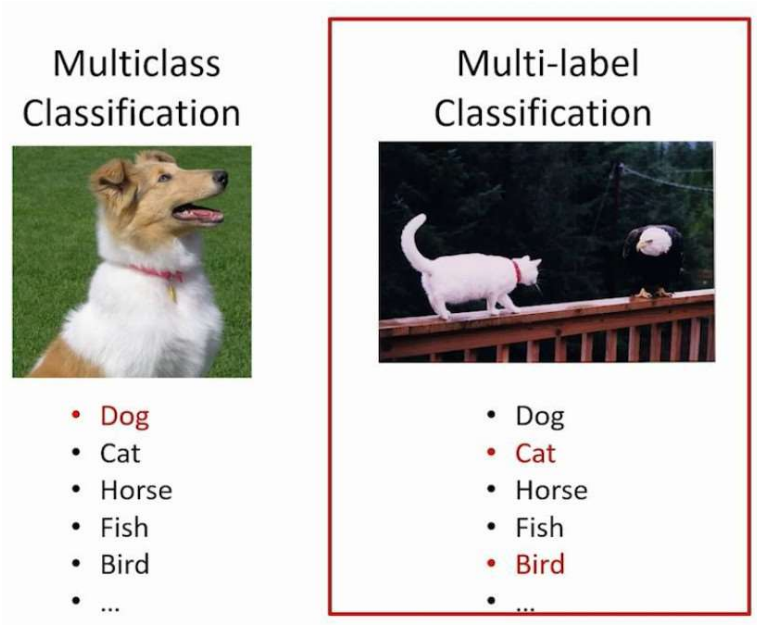
Bounding box Prediction



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

$$b_* = \sigma(t_*) + c_x$$
$$\sigma(t_*) = b_* - c_*$$
$$t_* = log(b_* - c_*)$$

Class Prediction



각각의 bounding box는 **multi-label classification**을 수행하는데 softmax 함수가 아닌 **Binary Cross-Entropy(BCE) 손실 함수**를 사용합니다.

Binary Cross-Entropy(BCE)란, 모델의 예측값과 실제 값 사이의 차이를 측정하여 예측이 정확할수록 손실이 적어지도록 하는 손실 함수입니다

$$L_{cls} = \sum_{i=1}^{N} \mathbf{1}_{obj}^{(i)} \sum_{c=1}^{C} [-y_{i,c} \log(\hat{p_{i,c}}) - (1 - y_{i,c}) \log(1 - \hat{p_{i,c}})]$$

. YOLOv3에서는 **Bounding Box가 객체를 포함하는지(Objectness Score)와 해당 객체가 어떤 클래스(Class Probability)에 속하는지를 예측하는 데 BCE를 사용합니다.**

.

Prediction across scales

YOLOv3는 서로 다른 3개의 scale을 이용하여 최종 결과를 예측합니다. 여기서 **multi-scale feature map**을 얻는 방법은 먼저 416x416 크기의 이미지를 네트워크에 입력하여 feature map이 크기가 52x52, 26x26, 13x13이 되는 layer에서 feature map을 추출합니다.

순서대로 작은 객체, 중간 크기 객체, 큰 객체를 탐지하는데 필요 합니다. 이제 해상도가 낮은 feature map(13x13)부터 탐지를 시작. 그 후13x13 feature map을 2배 크기로 업샘플링하여 26x26크기로 변환한 후 기존의 26x26크기의 feature map과 병합하여 중간 크기의 객체를 탐지하도록 함. 52x52 또한 같은 방식으로 수행하여 높은 해상도의 feature map을 통해 작은 객체도 잘 탐지할 수 있게 됨.

YOLOv3는 최종적으로 **각 Feature Map의 Output Channel을 255로 조정**합니다. 그 이유는 **각 Grid Cell이 3개의 Anchor Box에 대해 예측을 수행**해야 하기 때문

Output channels = 3x(4+1+80) = 255

출력되는 feature map의 크기는 52x52x255, 26x26x255, 13x13x255. 이 말은 각각의 feature map이 255개의 채널을 가진다는 의미.

이로 인해 더 높은 level의 feature map으로부터 **fine-grained 정보**를 얻을 수 있으며, 더 낮은 level의 feature map으로부터 더 유용한 **semantic 정보**를 얻을 수 있음

Feature Extractor

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

Table 1. **Darknet-53.**

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

Table 2. **Comparison of backbones.** Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

YOLOv3은 53개의 layer를 가지는 Darknet-53을 backbone network로 사용.

Darkenet-53은 ResNet-101보다 1.5배 빠르며, ResNet-152와 비슷한 성능을 보이지만 2배 이상 빠릅니다. 또한 당시 초당 가장 높은 floating point operation 속도를 보여주었으며, 이는 GPU를 효율적으로 사용함을 의미

Training

먼저 이미지를 입력하여 지정한 layer에서 52x52, 26x26, 13x13 크기의 feature map을 추출 (여기서 이미지 size는 416x416)

그 후 앞에서 얻은 3개의 서로 다른 scale을 가진 feature map을 1x1, 3x3 conv로 구성된 Fully Convolutional Network(FCN)에 입력하여 feature pyramid를 설계. 위에서 언급한 순서에 따라 진행되어 **52x52(x255), 26x26(x255), 13x13(x255)** 크기의 feature map을 얻음.

이제 앞에서 얻은 multi-scale feature maps를 loss function을 통해 학습.

YOLO v3의 loss function은 4개의 항으로 구성

1) bounding **box offset의 MSE(Mean Squared Error)**

$$L_{box} = \sum_{i=1}^{N} \mathbf{1}_{obj}^{(i)} \left[(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2 + (\hat{w}_i - w_i)^2 + (\hat{h}_i - h_i)^2 \right]$$

- 예측한 바운딩 박스(Bounding Box)와 실제 정답(Ground Truth) 바운딩 박스의 차이를 계산

- 오프셋(Offset) 값에 대한 평균 제곱 오차(Mean Squared Error, MSE)를 사용

2) 객체를 예측하도록 할당된(responsible for) bounding box의 **objectness score의 BCE(Binary Cross Entropy)**

$$L_{obj} = \sum_{i=1}^{N} \mathbf{1}_{obj}^{(i)} \left[-\log(\hat{p}_i) \right]$$

- Objectness Score는 예측한 바운딩 박스가 실제 객체를 포함하는지 나타내는 확률 값(0~1). 바운딩 박스가 객체를 포함하면 1 그렇지 않으면 0

3) 객체를 예측하도록 할당되지 않은 bounding box의 **no objectness score의 BCE**

$$L_{noobj} = \sum_{i=1}^{N} \mathbf{1}_{noobj}^{(i)} \left[-\log(1 - \hat{p}_i) \right]$$

- 모델이 객체가 없는 영역에서도 바운딩 박스를 생성하면 불필요한 계산이 증가할 수 있음. 이를 방지하기 위해 위 방법을 이용하여 불필요한 바운딩 박스 생성을 억제 함.

4) bounding box의 **multi-class BCE**

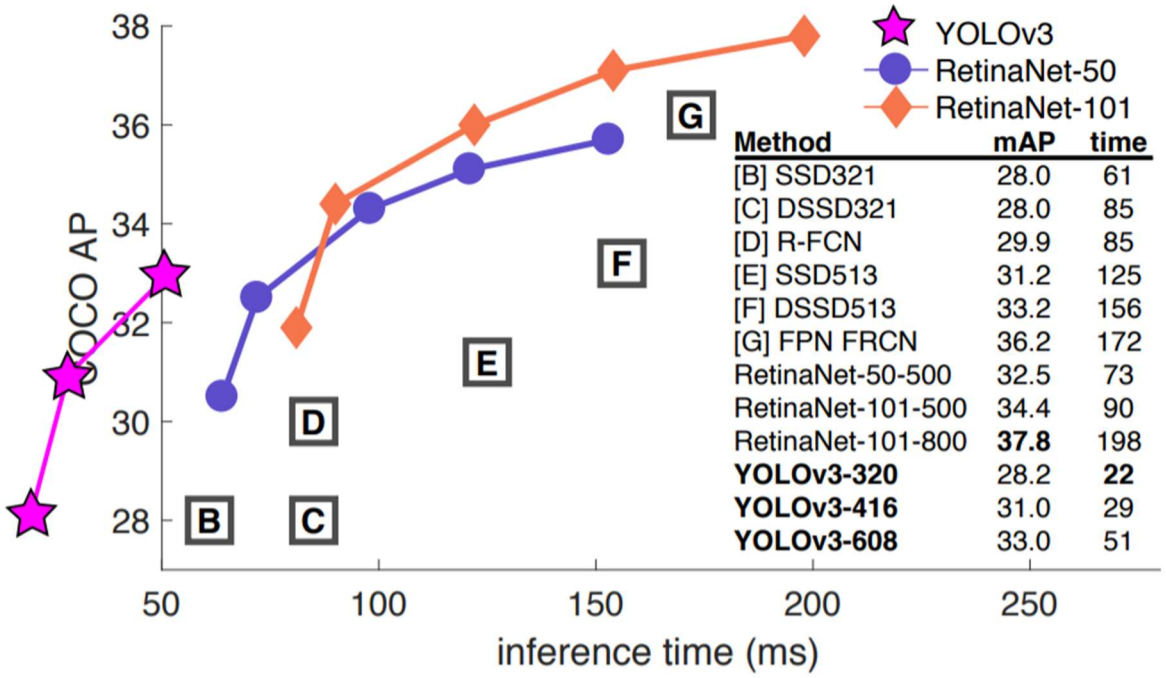
$$L_{cls} = \sum_{i=1}^{N} \mathbf{1}_{obj}^{(i)} \sum_{c=1}^{C} \left[-y_{i,c} \log(\hat{p_{i,c}}) - (1 - y_{i,c}) \log(1 - \hat{p_{i,c}}) \right]$$

- YOLOv3는 한 개의 바운딩 박스가 여러 개의 클래스 중 하나에 속할 확률을 예측하기 때문에 다중 클래스 분류 손실(**multi-class BCE**)을 사용

최종 loss function

$$L_{total} = L_{box} + L_{obj} + L_{noobj} + L_{cls}$$

Inference



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Inference시 마지막 예측 결과에 **NMS(Non Maximum Suppression)**을 적용함.

속도적인 측면에서 매우 혁신적인 성과. BUT RetinaNet보다 mAp가 낮음.