딥러닝 논문 요약 및 구현 스터디

발표자

이기환

You Only Look Once

Unified, Real-Time Object Detection

소개

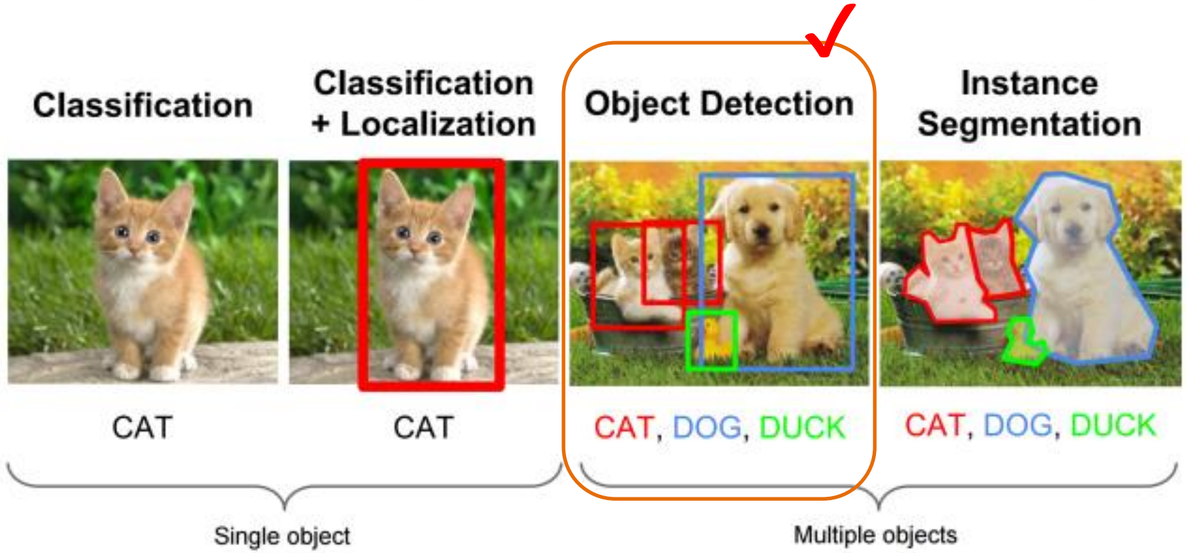
YOLO?

: You Only Look Once 로 이름 그대로 전체 이미지를 입력으로 받아 하나의 Convolutional Network를 이용한 모델

Object Detection에서 최초로 one-stage detection 방식을 제안

Object Detection
= Multi-Labeled **Classification**
+ Bounding Box Regression(**Localization**)

- **그렇다면 Object Detection이란 무엇인가?**



1) object classification

: 이미지 내 single object, object class

/ output : class probability

2) object localization

: 이미지 내 single object, object class, bounding box(물체의 위치)

/ output : (x,y,w,h) 좌표

3) Object detection (우측 그림의 세 번째)

: 이미지 내 multiple object, object class, bounding box,

/ output : class probabilities + (x,y,w,h) (* 또 다른 객체가 있다면 그 class probabilities+(x,y,w,h))

소개

One-stage Detection VS Two-stage Detection

one stage detection (Yolo)

: conv layers를 거쳐서 한 번에 multi-class classification(어떤 객체인지)와 bounding box regression(좌표값)을 찾아낸다.

1) conv & FC layers

2) reshape하고, output tensor를 만든다.

3) 어떠한 알고리즘(NMS)을 적용해서 우리가 원하는 클래스 정보와 좌표값을 찾아낸다.

two stage detection (Fast RCN)

: localization -> classification 순차적으로 수행하여 결과를 얻는다. 즉, 후보 object 위치 제안 후, object class를 예측한다.

1) 첫 번째 스테이지에서 region proposal과정을 통해 proposed regions(물체가 있음직한 곳)을 찾아낸다.

2) 두 번째 스테이지에서는 똑같은 원본을 classification layer에 넣어서 feature maps를 찾아낸다.

3) 이렇게 찾아낸 둘을 잘 연결해서 FC layers에 보내서 결과적으로 classfication과 Box refinement를 가능하게 한다.

소개

One-stage Detection VS Two-stage Detection

one stage detection (Yolo)

Conv & FC layers

Reshape

Output Tensor

For each grid cell

Multi-Class Classification (regression)

Box Regression

two stage detection (Fast RCN)

Region Proposal

Classification
: Object O or X
(단순 객체가 있나 없나)

Box Regression

Proposed Regions

Object Classification

Feature maps

FC layers

For each proposed region

Classification (cross entropy)

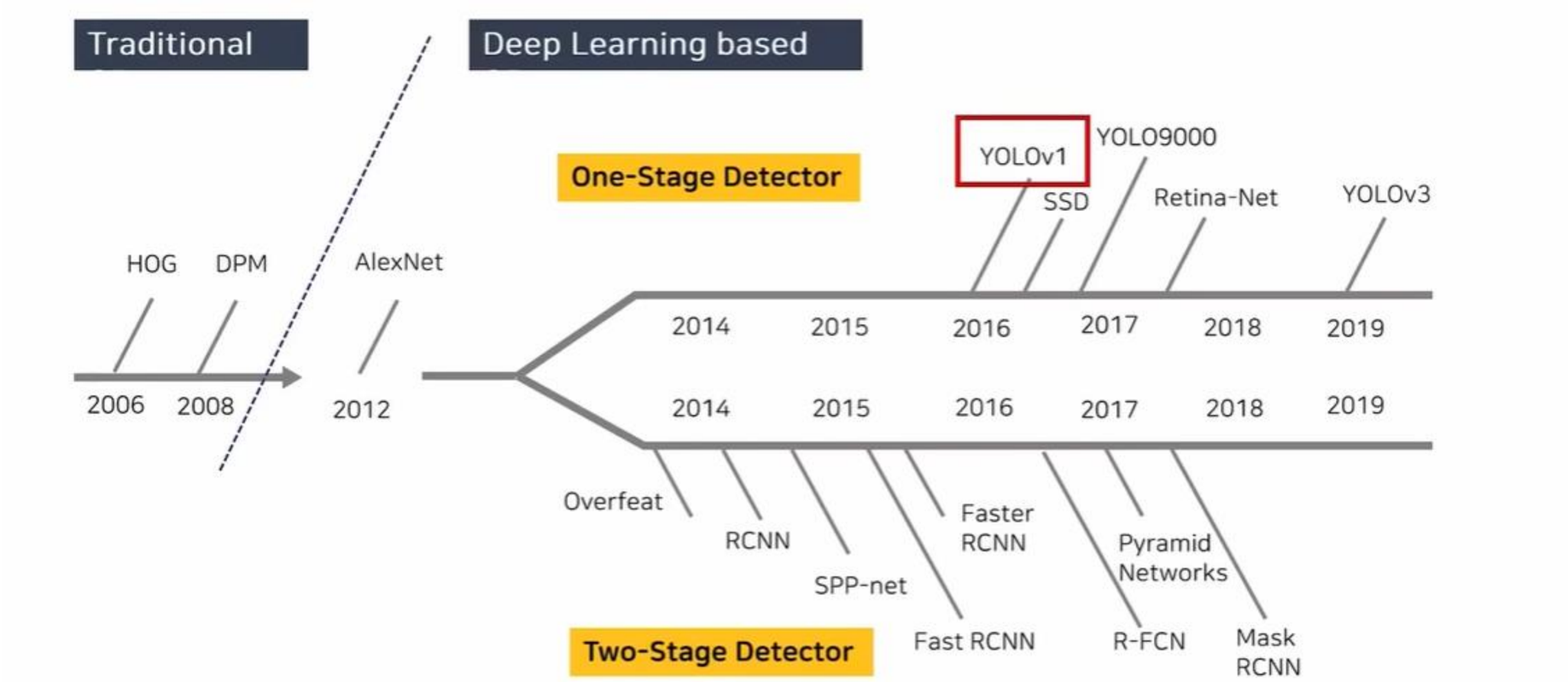
Box Refinement (regression)

소개

Object Detection MileStones

: 과거의 HOG, DPM과 달리 AlexNet등장 이후 딥러닝에 기반한 모델들이 계속해서 등장하고 있다.

여기서 one stage(yolov123, SSD 등)와 two stage(RCNN계열과 SPP-net 등)로 나뉘어 발전하고 있다.



Abstract

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

- **YOLO**
  - 이전의 classifier를 이용한 detection과 달리

    bounding box와 class probability에 대한 regression 문제로 접근
  - 단일 신경망을 사용하여 한 번의 평가로 전체 이미지에서 직접 bbox와 클래스의 확률을 예측

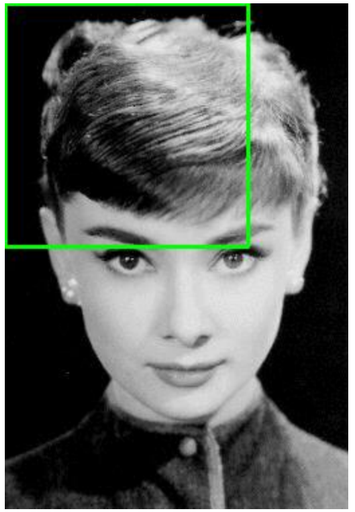
  ⇒ 전체 detection pipeline이 단일 네트워크이기에 엔드 투 엔드 최적화가 가능하여 효율적임

- **모델 성능**
  - YOLO의 기본 모델은 초당 45프레임으로 이미지를 실시간 처리

    Fast YOLO는 초당 155프레임 처리로 다른 실시간 detector들에 비해 약 두 배 가량의 mAP 성능 향상을 볼 수 있음
  - Localization erorro는 더 발생하지만, background에서 오탐지를 예측할 가능성은 더 낮음

I. Introduction

사람은 이미지를 보고 객체들이 어디 있고, 어떤 객체인지, 어떤 상호작용을 하고 있는지 즉시 파악이 가능 ex) 운전과 같은 복잡한 수행도 수월하게 가능



[Sliding window]

- **이렇게 한번에 가능하게 할 수 있을까?**

이전 대부분의 모델들은 Deformable Parts Models(DPM)이 sliding window 방법론을 사용하여 전체 이미지를 Classifier로 스캔하는 방식 형태의 방법론을 사용했음

예를 들어 YOLO 이전의 가장 최근 모델인 R-CNN 모델은 region proposal / classification / box regression라는 3가지 단계를 거치는 과정을 가지며 복잡한 파이프라인을 갖고 있음

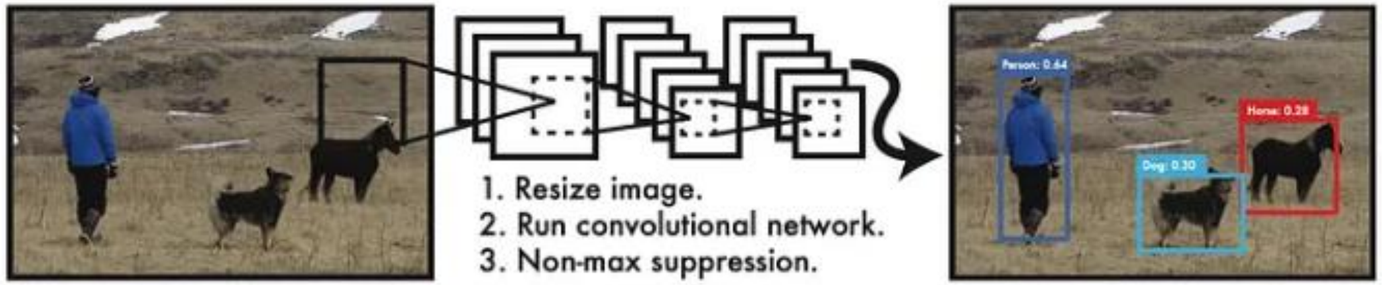
그렇기에 최적화가 어렵고, 큰 inference time을 갖는다는 단점이 존재

YOLO는 기존의 Classification모델을 변형한 방법론에서 벗어나, Object Detection 문제를 single regression problem으로 정의하는 것을 통해서 하나의 회귀 문제로 재정의

즉, 하나의 파이프 라인으로 빠르게 bounding box와 클래스 확률을 동시에 계산

I. Introduction

- **YOLO의 장점**



1) YOLO is extremely fast : 빠른 성능

복잡한 객체 검출 프로세스를 하나의 회귀문제로 변형함을 통해 동영상 또한 실시간 처리 가능

2) YOLO reasons globally about the image : 예측할 때 이미지 전체를 확인

Sliding window나 region proposal과 달리 전체 이미지를 봄으로써 background error를 줄일 수 있음

3) YOLO learns generalizable representations : 일반적인 부분을 학습

일반적인 부분을 학습함으로 여러 domain에서 Object Detection이 가능하게 됨 – 실험 파트에서 소개

II.Unified Detection

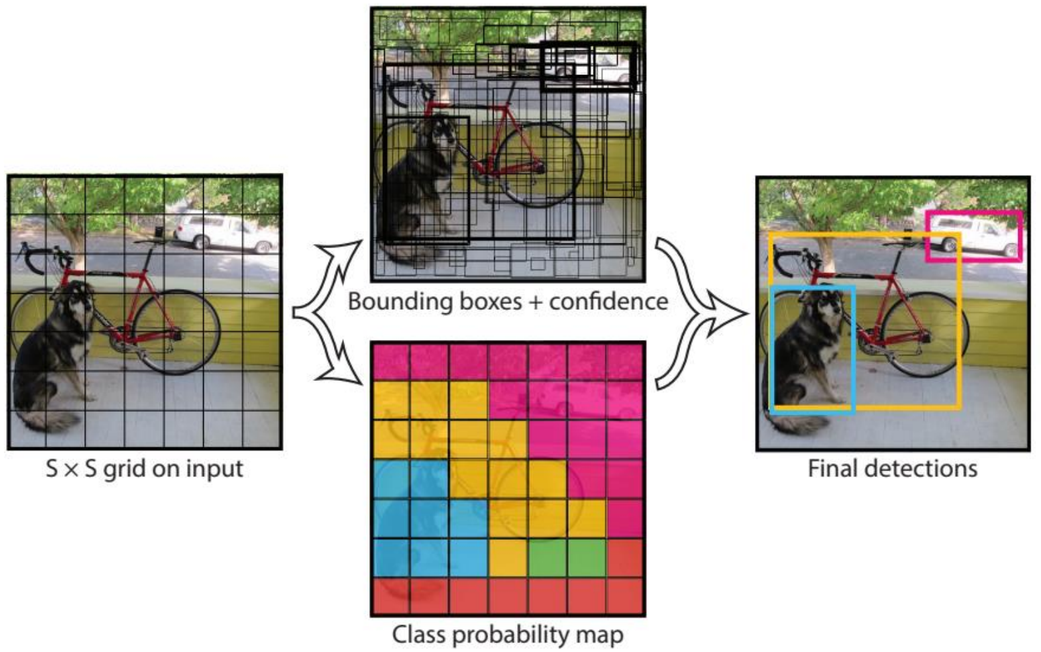
- **Unified Detection 0**

region proposal, feature extraction, classification, bbox regression

=> one-stage detection으로 통합

: 이미지 전체로 얻은 feature map을 활용해서 bbox 예측& 모든 클래스에 대한 확률계산

그림을 보면 bbox찾기와
class probability가
병렬적으로 이루어지고 있음



II.Unified Detection

- **Unified Detection 1**

1) 입력 이미지를 SxS grid로 나눔
2) 만약 어떤 객체의 중심이 특정 grid cell에 존재한다면, 그 cell이 해당 객체를 검출해야 함
3) 각각의 grid cell은 B개의 bbox와 그 bbox에 대한 confidence score를 예측함

: confidence score? 물체가 있는지에 대한 확률(Pr(Object) : 1or0)과,
물체가 있다는 것을 얼마나 신뢰할 만한지(IOU) 알려주는 값

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

* IOU = (실제 bounding box와 예측 bounding box의 교집합) / (실제 bounding box와 예측 bounding box의 합집합)

⇒ Confidence score와 IOU값이 같을 때 가장 이상적인 score (Pr(Object)=1)

- **Unified Detection 2**

각각의 bbox는 5개의 예측치로 구성 : x, y, w, h, confidence score

: (x,y)는 bbox의 중심 값으로 셀의 중앙에 위치한다면 (0.5,0.5)

w, h도 마찬가지로 0과 1사이의 값을 가짐

II.Unified Detection

- **Unified Detection 3**

$$C(conditional\ class\ probabilities) = \Pr(Class_i|Object)$$

각각의 그리드 셀은 conditional class probabilities(C)를 예측함

이는 그리드 셀 안에 객체가 있다는 조건 하에 그 객체가 어떤 클래스인지에 대한 조건부 확률로, 그리드 셀에 몇 개의 bbox가 있는지와 무관하게 하나의 그리드 셀에는 오직 하나의 클래스에 대한 확률값만을 구함

다시 말해 하나의 grid cell은 B개의 bbox를 예측한다고 했는데, B의 개수와는 무관하게 하나의 그리드 셀에서는 클래스 하나만을 예측하는 것

- **Unified Detection 4**

이렇게 구한 conditional class probability(C)를 앞에서 구한 개별 bbox의 confidence score를 곱해주는데,

이 값을 각 bbox에 대한 class-specific confidence score라고 부름

이 score는 bbox에 특정 클래스 객체가 나타날 확률(Pr(class_i))과 예측된 bbox가 그 클래스 객체에 얼마나 잘 들어맞는지(fits the object=IOU_pred^truth)를 나타냄

$$\begin{aligned} &class\ specific\ confidence\ score \\ &= \Pr(Class_i|Object) * \Pr(Object) * IOU_{pred}^{truth} \\ &= \Pr(Class_i) * IOU_{pred}^{truth} \end{aligned}$$

II.Unified Detection

- **Unified Detection 5**

S : 4x4 / B : 각 그리드 셀 마다 예측하는 Bbox개수 2개

/ C : 전체 고려하는 클래스

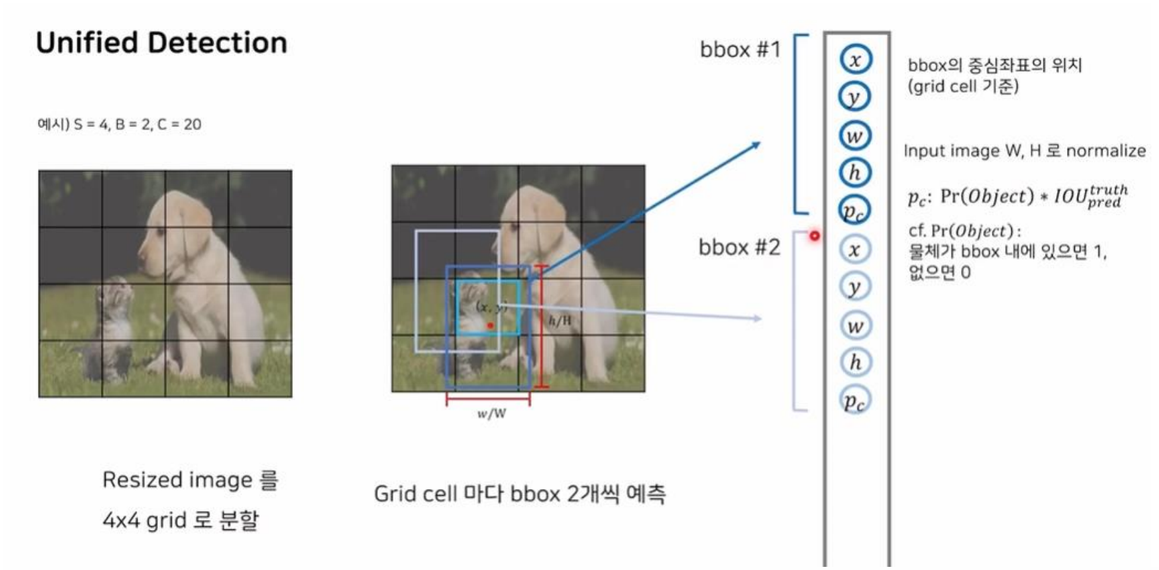
/ P_c: confidence score.

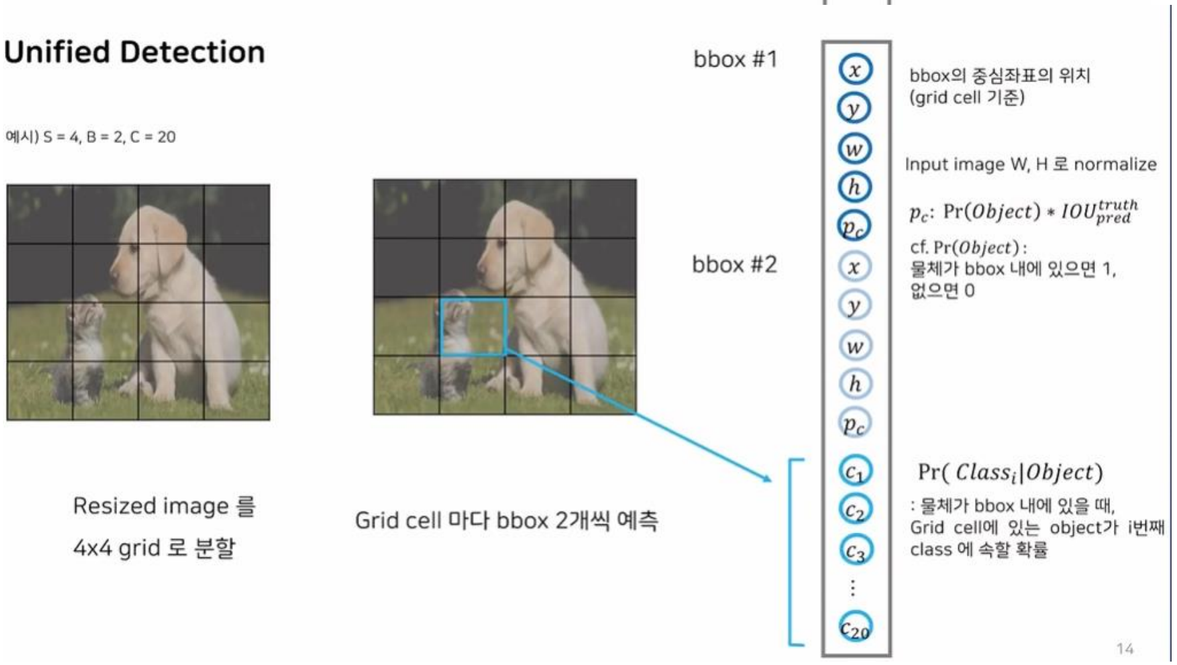
Pr(Class_i | Object) : 하나의 그리드 셀에 대해 물체가 bbox 내에 있을 때, 내부 object가 i번째 클래스에 속할 확률

4x4에서 하나의 grid cell에서 하나의 바운딩박스에 대해 나오는 output값이 5개

-> 하나의 그리드 셀에서 2개의 bbox를 그리기에 5*2

-> 총 클래스 개수가 20이기에 : 최종 예측 텐서 dim = 4x4x(5*2+20)





II.Unified Detection

Unified Detection 요약:

- YOLO는 객체 검출의 개별 요소를 단일 신경망(single neural network)으로 통합한 모델
- 입력 이미지(input images)를 S x S 그리드(S x S grid)로 나눔
- 각각의 그리드 셀(grid cell)은 B개의 bounding box와 그 bounding box에 대한 confidence score를 예측
- class-specific confidence score는 bounding box에 특정 클래스(class) 객체가 나타날 확률과 예측된 bounding box가 그 클래스(class) 객체에 얼마나 잘 들어맞는지를 나타냄
- 최종 예측 텐서의 dimensio은 (7 x 7 x 30)

II.i.Network Design

- YOLO : 하나의 CNN 구조 _ Network Design

이미지 분류를 위한 GoogleNet에서 영감을 받았으며, 24 conv layer(이미지 특징 추출) & 2 FC layer(클래스 확률과 bbox좌표값 예측) 로 구성

(* Fast Yolo는 9 conv layer & 필터수가 더 적은 신경망 사용 :훈련 및 테스트 시 사용하는 나머지 파라미터는 YOLO와 모두 동일)

GoogleNet과의 차이는 GoogleNet의 인셉션 구조 대신 1x1 reduction layer와 3x3 conv layer 계층의 결합을 사용

: 최종 output은 7 x 7 x 7 prediction tensor

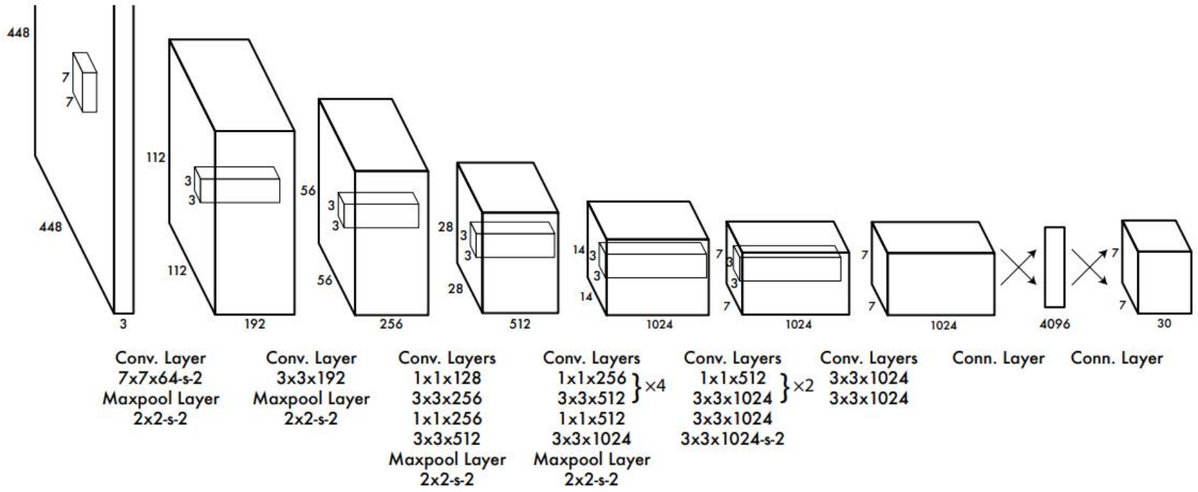


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

II.ii. Training

- YOLO : 하나의 CNN 구조 _ Training

1000개의 클래스를 갖는 ImageNet데이터셋으로 사전훈련(20 conv layer pretrain, 그 후 4 conv layer & 2 FC layer)

최종 output값은 class probabillity와 bbox좌표(x, y, w, h 모두 0과 1 사이의 값으로 정규화)

마지막 계층에만 linear activation ft 적용, 나머지 모든 계층에는 leaky Relu

YOLO의 loss
- localization loss : bbox의 위치를 얼마나 잘 예측했는가?
- classification loss : 클래스를 얼마나 잘 예측했는가?

문제점 1) Loss

Yolo의 loss는 SSE를 기반으로 함(SSE가 최적화하기에 쉽기 때문) : 최종 아웃풋의 목적은 SSE를 최적화하는 것

But SSE를 최적화하는 것이 YOLO의 최종 목표인 mAP(평균 정확도)를 높이는 것과는 완벽히 일치하지 않음

문제점 2) grid cell

대부분의 이미지에서 grid cell에는 객체가 존재하지 않음

-> 즉, confidence score가 0이며, 이렇게 학습시키면 객체가 작은 경우 0이 압도적으로 많아 모델의 불균형을 초래할 수 있음

-> 이를 개선하기 위해 객체가 존재하는 bbox 좌표에 대한 loss의 가중치를 증가시키고, 객체가 존재하지 않는 bbox의 confidence loss에 대한 가중치는 감소

=>이는 localization loss와 classification loss의 가중치를 증가시키고, 객체가 없는 grid cell의 confidence loss 보다 객체가 있는 grid cell의 condidence loss의 가중치를 증가시켜 문제를 해결함 * 이 때 사용한 파라미터가 λ_coord(=5)와 λ_noobj(=0.5)

λ_coord: coordinates(x, y, w, h)에 대한 loss와 다른 loss들과의 균형을 위한 balancing parameter.
λ_noobj: 객체가 있는 box와 없는 box 간에 균형을 위한 balancing parameter. (일반적으로 image내에는 객체가 있는 그리드 셀보다는 없는 셀이 훨씬 많으므로)

II.ii. Training

- YOLO : 하나의 CNN 구조 _ Training

문제점 3) SSE

SSE는 큰 bounding box와 작은 boudning box에 대해 모두 동일한 가중치로 loss를 계산

-> But 작은 bounding box가 큰 bounding box보다 작은 위치 변화에 더 민감

-> 이를 개선하기 위해 bounding box의 너비(widht)와 높이(hegith)에 square root를 취함으로 너비와 높이가 커짐에 따라 그 증가율이 감소해 loss에 대한 가중치를 감소시키는 효과가 발휘

YOLO의 bounding box

YOLO는 하나의 grid cell 당 Conv layer를 통해 예측했던 여러 개의 bbox를 예측하는데, 이 여러 box 중 ground-truth bounding box와 IOU가 가장 높은 bbox 하나만을 선택함 (predictor box)

$\mathbb{1}_{ij}^{\text{obj}}$: cell i에서 responsible한 j번째 값을 표시하여 loss function에 반영

여기서 선택된 하나의 값만 1로 주고, 선택되지 않은 나머지 값은 0으로 둠으로써 loss function에 전파가 되지 않도록 함

II.ii. Training

- YOLO : 하나의 CNN 구조 _ Training

훈련 단계에서 사용하는 loss function

: grid cell i 의 j 번째 bbox가 사용되는지 여부 (1or0)

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

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

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

: grid cell i 의 j 번째 bbox가 사용 안 되는지 여부 (1or0)

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

: grid cell i 안에 객체 존재 여부 (1or0)

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

(1) Object가 존재하는 grid cell i의 bounding box predictor j에 대해, x와 y의 loss를 계산

(2) Object가 존재하는 그리드 셀 i의 bounding box predictor j에 대해, w와 h의 loss를 계산
큰 box에 대해서는 작은 분산(small deviation)을 반영하기 위해 제곱근을 취한 후, sum-squared error를 구함

(3) Object가 존재하는 그리드 셀 i의 bounding box predictor j에 대해, confidence score의 loss를 계산 (Ci=1)

(4) Object가 존재하지 않는 그리드 셀 i의 bounding box predictor j에 대해, confidence score의 loss를 계산. (Ci = 0)

(5) Object가 존재하는 그리드 셀 i에 대해, conditional class probability의 loss를 계산
(p_i(c)=1 if class c is correct, otherwise: p_i(c)=0)

II.iii. Inference

- YOLO : 하나의 CNN 구조 _ Inference

훈련 단계와 마찬가지로, 추론 단계에서도 테스트 이미지로부터 객체를 검출하는 데에는 하나의 신경망만 계산 (그렇기에 속도가 매우 빠름)

YOLO의 grid design에 한 가지 문제가 있다면, 객체의 크기가 너무 크거나 중심이 grid cell 경계에 위치해 있는다면, 한 객체에 대해 여러 개 bbox가 생길 수 있음
-> 이는 다중 검출 문제(multiple detections)라 하는데, NMS(non-maximal suppression) 방법을 통해 개선

- Non-Maximum Suppression

각 객체에 대해 예측한 여러 bbox중에서 가장 예측력이 좋은 bbox만 남기는 알고리즘
: 우선 output tensor에서 나오는 하나의 grid cell 값에 대해 (x,y,w,h,p_c | x,y,w,h,p_c | c1,c2,c3,…,c20) 각각의 bbox에 대해서 confidence score와 class probability값의 곱을 통해 class-specific confidence score를 만들고, 이를 다시 기존 bbox좌표값 및 confidence score와 결합하여 bbox를 만들어 냄
이렇게 차례대로 만들어진 bbox들에 대하여 각 클래스 별로 순차적으로 NMS적용
: 0) 하나의 클래스에 대해서 class-specific confidence score를 sorting 1) 임계값을 넘지 못 하는 bbox 우선 제거
- 객체 1개 : 가장 높은 값을 가지는 bbox 선택
- 같은 객체 2개 이상 : 가장 높은 값을 갖는 bbox와 순차적으로 IOU 계산을 통해 높은 값들은 제거(같은 객체) / 낮은 값은 다른 객체를 가리키고 있는 것이기에 유지
- 다른 객체 : 위에서 말한 바와 같이 클래스별 class-specific confidence score를 각각 sorting / 그 후 위와 동일하게 순차적 IOU 계산

II.iv. Limitations of YOLO

- **YOLO : 하나의 CNN 구조 _ Limitations**

1) 작은 객체들이 몰려있는 경우 검출을 잘하지 못 함

: 객체가 크면 bbox간의 IOU값의 차이가 커져서 적절한 predictor box를 선택할 수 있지만, 객체가 작으면 bbox도 작아지고 bbox간 IOU값 차이가 작아져 근소한 차이로 predictor box가 선정됨

2) 일반화된 지식과 다르게 객체 비율이 달라지면(aspect ratio) detection 성능이 낮아짐

(대부분 detection 모델들이 갖고 있는 한계점)

IV. Experiment

- YOLO : 하나의 CNN 구조 _ Experiment

4.1 Comprision to Other Real-Time Systems

- 객체 검출에 대해 많은 연구진들은 표준화된 객체 검출 파이프라인을 빠르게 만드는데 초점을 두고 있음
- Fast YOLO는 파스칼셋 데이터 기준으로 가장 빠르며, DPM 대비 2배 높은 정확도를 가짐
- 표에서 FPS가 30정도 이상이 되어야 실시간 detect를 할 수 있다고 봄(1초에 30프레임)
- Two-stage 계열이 성능은 좋지만 속도가 느림을 볼 수 있음

: YOLO는 기존 모델들에 비해 속도는 월등히 빠르고, 정확도도 꽤 높은 수준임을 확인 가능

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	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[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

IV. Experiment

- **YOLO : 하나의 CNN 구조 _ Experiment**

4.2,3 Fast-RCNN과 YOLO의 비교 / 둘의 결합

- 이 표는 두 모델의 어느 부분에서 에러율이 높은지 분석한 것
- FastRCNN보다 background error는 감소 : False positive를 감소시킴
  (background에 아무것도 없는데 있다고 할 확률을 매우 줄였다.)
- Fast RCNN + YOLO를 합쳤을 때 MAP가 향상되었다. (하단의 표)
  : 이는 단순히 앙상블했기에 높아진 것이 아니다.
  왜냐하면 다른 모델들과 앙상블시 성능향상은 없었기에

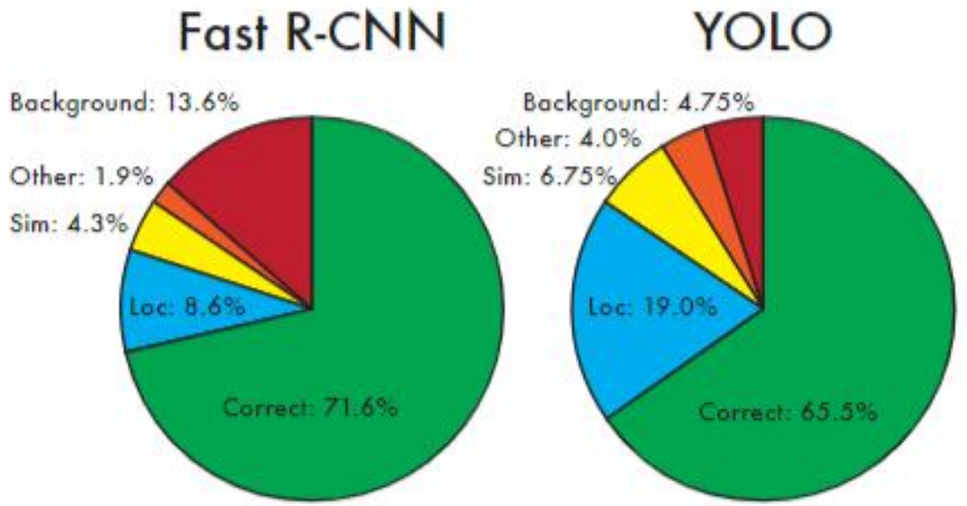


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

	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**

IV. Experiment

- YOLO : 하나의 CNN 구조 _ Experiment

4.4 VOC 2012 Results

속도 측면에서는 YOLO가 빠르고, 정확도 측면에서는 Fast R-CNN과 YOLO를 결합한 모델이 가장 우수함

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	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
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	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
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	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
BabyLearning [7]	63.2	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
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	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
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	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
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Table 3: PASCAL VOC 2012 Leaderboard. YOLO compared with the full `comp4` (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

IV. Experiment

- YOLO : 하나의 CNN 구조 _ Experiment

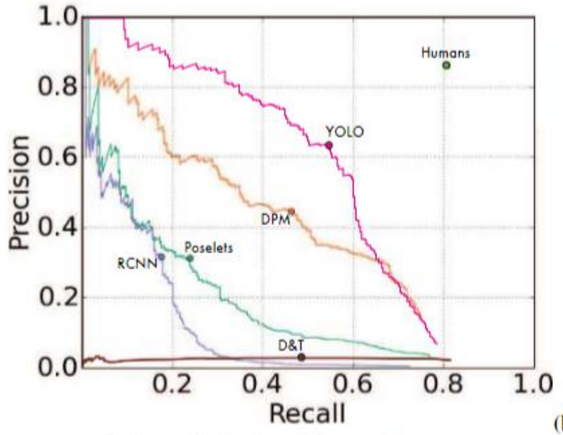
4.5 YOLO는 다양한 domain

dataset : picasso dataset, people-art dataset

task : person detection (detection AP on Class_person) : 이미지 내의 사람만 찾아내는 실험. 그렇기에 MAP가 아닌 AP로 표시

우측 아래 표 : VOC 2007 (AP=59.2) -> Picasso (AP=53.3), People-Art (AP=45) 로 다른 도메인으로 이동해도 모두 우수함

하지만 RCNN 등은 성능이 떨어지며, 떨어지는 폭의 차이가 큰 현상을 확인해 볼 수 있음



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

VI. Conclusion

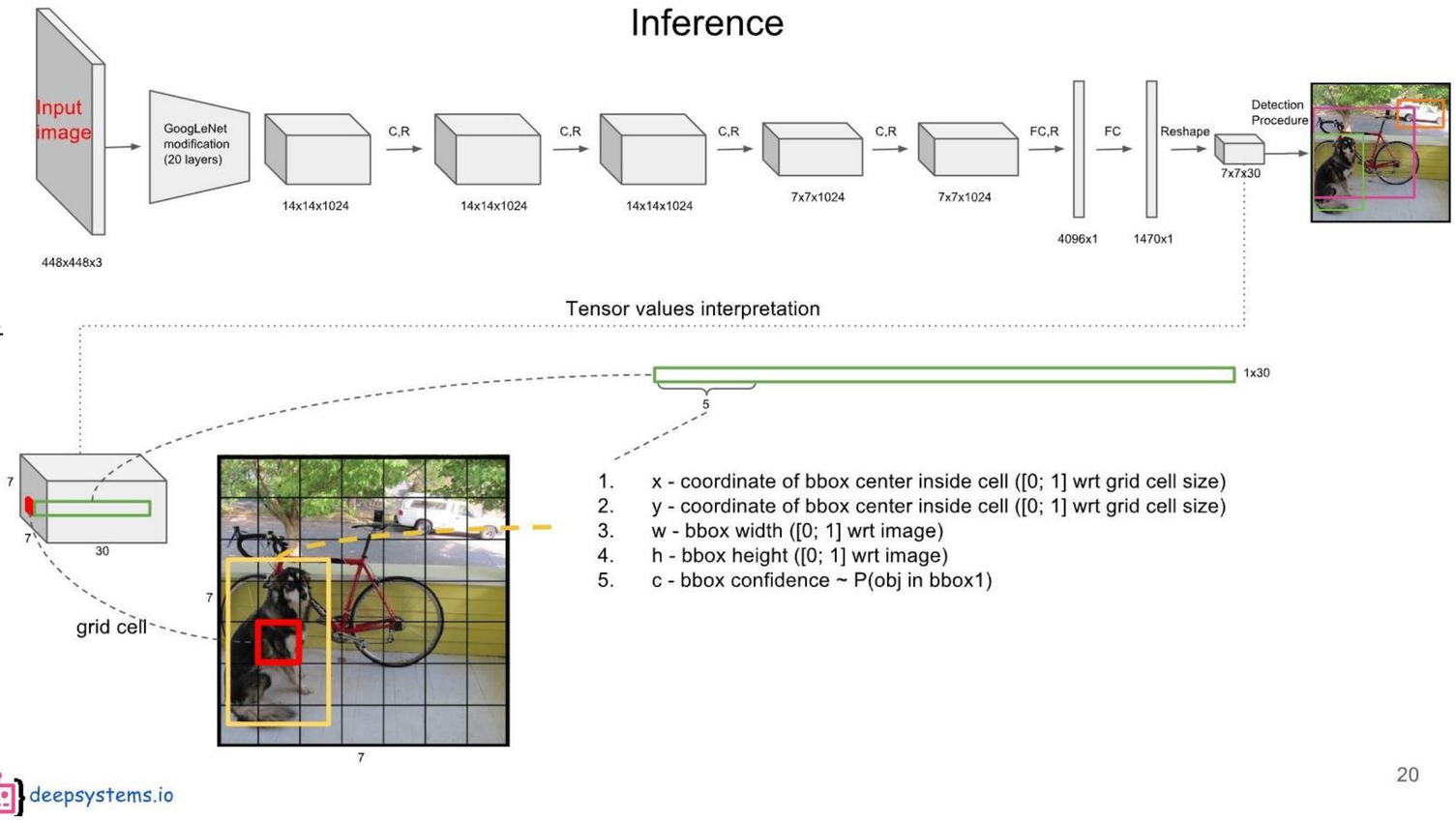
- **YOLO : 하나의 CNN 구조 _ Conclusion**

- YOLO는 구성이 간단하고 전체 이미지에 대해 직접 훈련할 수 있다는 장점이 있음

- Classifier기반 접근 방식과 달리 YOLO는 검출 성능에 직접적으로 대응하는 손실 함수에 대해 학습하고 전체 모델을 공동으로 학습

- Fast YOLO 또한 우수한 속도와 성능을 보임

- YOLO는 실시간 객체 감지 분야에서 실시간 객체 감지 뿐 아니라 새로운 도메인으로도 잘 일반화되어 많은 곳에서 충분히 활용이 가능함



deepsystems.io