YOU ONLY LOOK ONCE: Unified, Real-Time Object Detection

2020. 03. 17 신한이

0. Abstract

" ***YOLO*** : You Only Look Once "

- **Object Detection** 에 대한 새로운 접근방식
- **A single NN**, 하나의 신경망으로 구성
- **quick** (*Fast YOLO : 1초에 155프레임 정보 처리)
- **한 번의 계산 과정**으로 bounding box, class probability 예측

1. Introduction

cf)
DPM
R-CNN

DPM: 이미지 전체를 sliding window

R-CNN: region proposal -> bounding box -> classifier -> 점수를 조정
(단점: 느리고, optimization이 힘들다.)

YOLO

YOLO:
하나의 CNN 안에서 통합되어 동작한다.
(성능이 좋고, 빠름)

1. Introduction

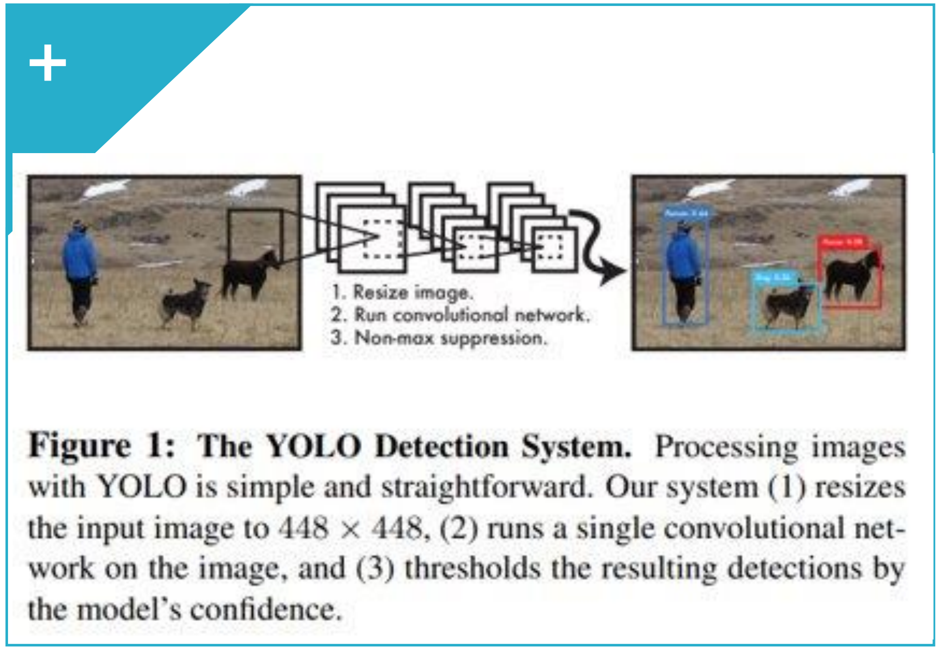


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO System

(1) resize to 448*448

(2) A single CNN

(3) thresholds the detections

System:

Bounding box의 위치

클래스 확률(class probabilities)

=> 모든 과정을 하나의 회귀(regression)문제로 정리

Feature:

1. 빠른 속도 , 단순한 구조

2. 이미지 전체를 사용 , background error 낮음

3. 물체의 일반적인 부분을 학습 , 새로운 이미지에 대해 검출 정확도 높다.

단, SOTA 모델에 비해 정확도가 떨어짐

2. Unified Detection

●○○○○ S x S grid로 이미지를 나눈다.

●●○○○ 각각의 셀(grid cell)에서 Bounding Box 에 대한 confidence score를 예측한다.

Confidence score

= Pr(Object) * IOU

●●●○○ 각 bounding box는 5개의 예측치로 구성 : x,y,w,h, confidence

이때 x,y,w,h는 box 내의 상대값으로 0-1 사이의 값을 가짐

●●●●○ 각 grid cell에서 C(conditional class probabilities)를 예측.

C = Pr(calss|Object)

●●●●● Class-specific confidence score 계산

Score

= C *confidence score

= Pr(calss)*IOU

2. Unified Detection 2.1. Network Design

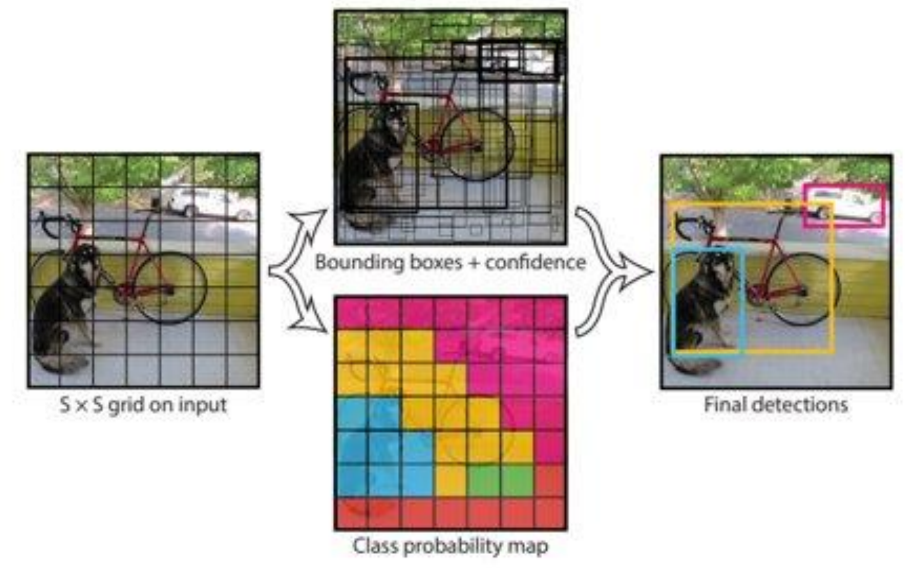


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

파스칼 VOC 데이터셋 사용

(S=7, B=2 세팅, C=20)

S=7 -> Inpit Img: 7 x 7 grid cells

B=2 -> 1 grid cell, 2 bounding box 예측

=> S x S x (B*5 + C) tensor 생성.

Final tensor dimension = (7 x 7 x 30)

최종 예측 tensor dimension :

7*7*30

2. Unified Detection
2.1. Network Design

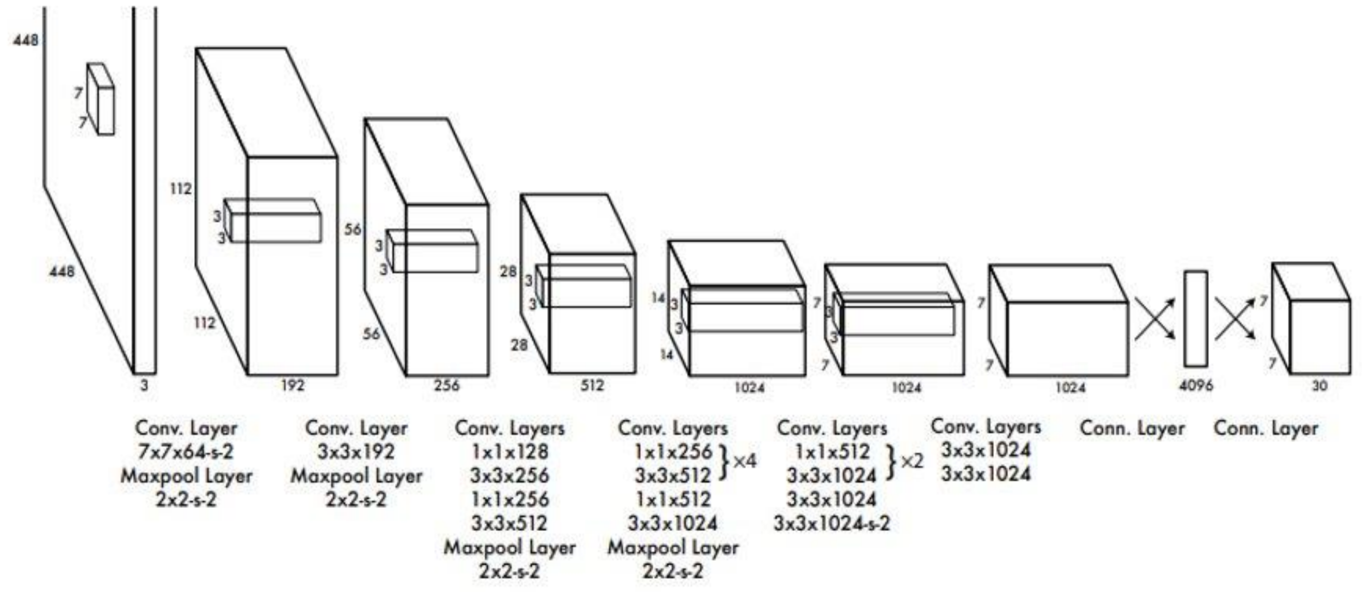


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224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

YOLO는 총 24개의 컨볼루션 계층(convolutional layers)과 2개의 전결합 계층(fully connected layers)으로 구성

1 x 1 축소 계층(reduction layer)과 3 x 3 컨볼루션 계층의 결합이 인셉션 구조를 대신.

=> 7*7*30 output

2. Unified Detection

2.2. training

ImageNet dataset
첫 20개의 계층 사용
+4개의 계층과 2개의 전결합
계층 추가 => 성능 향상

Darknet 프레임워크 사용

해상도 증가: 224^->448^

Pretraining

Output:
1. class probabilities
2. box의 위치 정보(x,y,w,h)

⇒ Loss도 2가지: localization loss classification loss

Output 값

NN의 마지막 계층
: linear activation func.

나머지 모든 계층
: leaky ReLU

Loss Func.

2. Unified Detection

2.2. training

● **구조상 문제:**

SSE를 기반으로 할 때의 구조적 문제 발생

SSE의 최적화 / box의 크기에 상관없이 동일하게 loss를 계산했을 때, 오차율 커짐

● **해결:**

1) localization loss의 가중치를 증가

2) 객체가 있는 grid cell의 confidence loss의 가중치를 증가

3) bounding box의 w,h에 square root를 취해준 값을 loss function으로 사용

- 과적합(overfitting)을 방지: 드롭아웃(dropout)과 data augmentation을 적용

2. Unified Detection

2.2. training

loss function:

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}
$$

2. Unified Detection

2.3. Inference

+

파스칼 VOC dataset / YOLO는 한 이미지 당 98개의 bounding box 예측
-> box마다 클래스 확률 계산.
-> 굉장히 빠름.

+

하나의 object를 여러 grid cell이 검출하는 경우
Bounding box 여러 개가 생성될 수 있음. (다중 검출 문제)
⇒ **비 최대 억제(non-maximal suppression) 방법으로 개선**

2. Unified Detection

2.4. Limitations of YOLO

Limit 1. 하나의 grid cell에서 두 개의 bounding box만을 예측, 하나의 객체(Object)만 검출

⇒ 공간적 제약(Spatial Constraints) 문제

(Ex. 물체가 겹쳐있는 경우 객체를 놓침.)

Limit 2. Training 때 학습하지 못한 가로세로비율을 접함

⇒ 혼란 야기

Limit 3. Bounding Box에서 큰 박스와 작은 박스에 동일한 가중치를 적용하여

작은 박스는 위치에 따라 큰 IOU 변화

⇒ 위치 문제(Localization)

3. Comparison to Other Detection Systems

DPM

YOLO와 비교점:

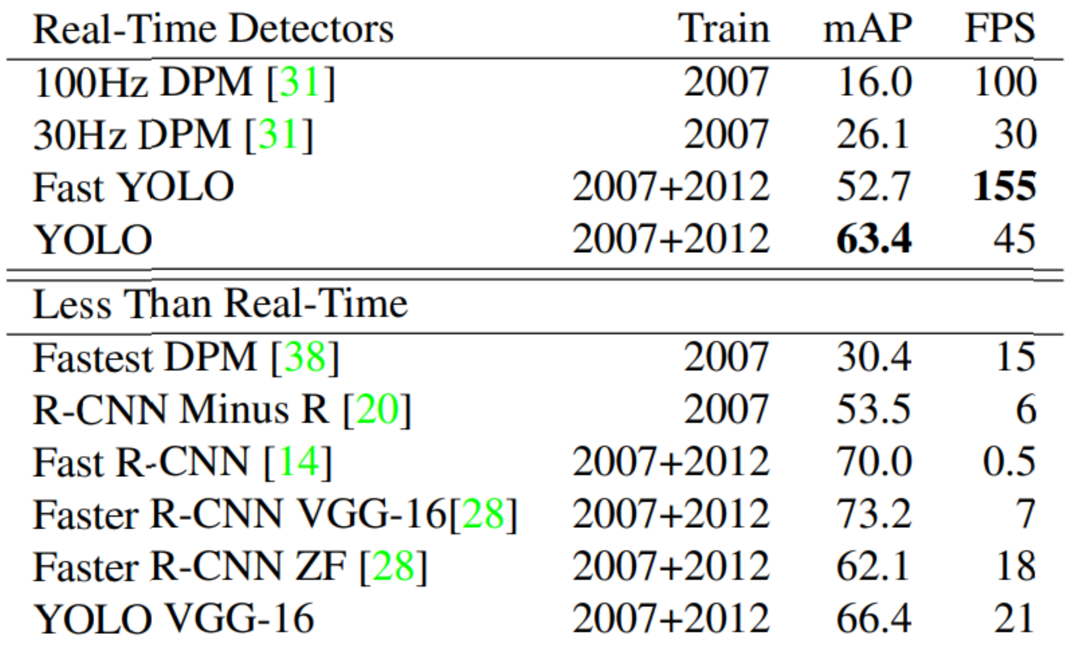
- **Sliding window 적용**
- **서로 분리된 파이프라인**(특징 추출, 위치파악, box 파악 등)
- **YOLO가 더 빠르고 정확.**

R-CNN

YOLO와 비교점:

- **Region proposal 방식, 복잡한 파이프라인**
- **속도가 느림**
- **각 cell에서 bounding box를 예측하고 score를 매긴다는 공통점 있음.**

4. 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

Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

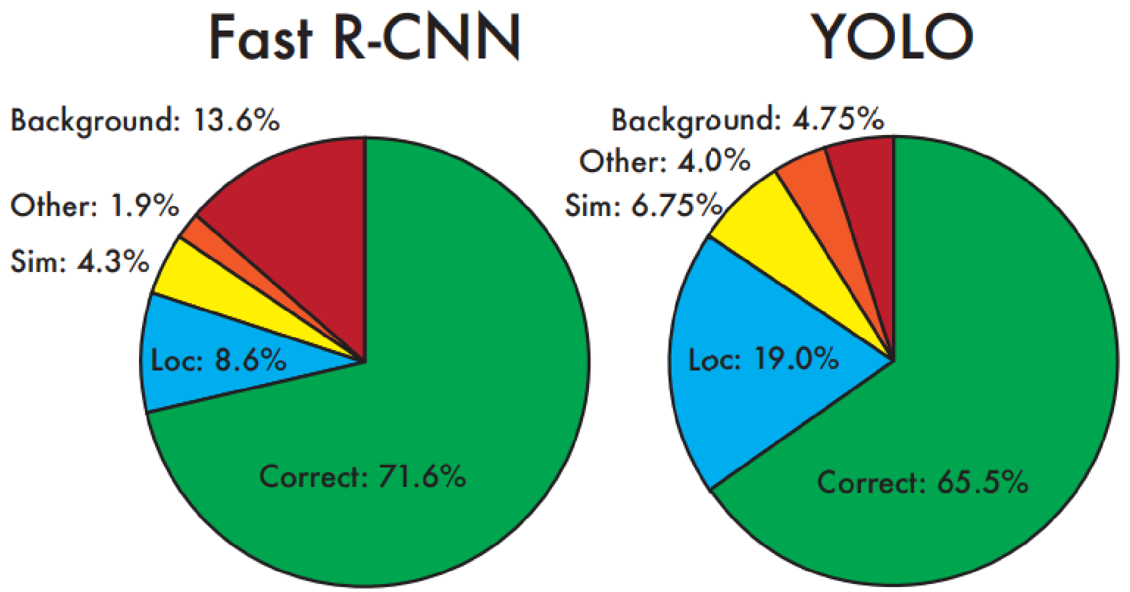
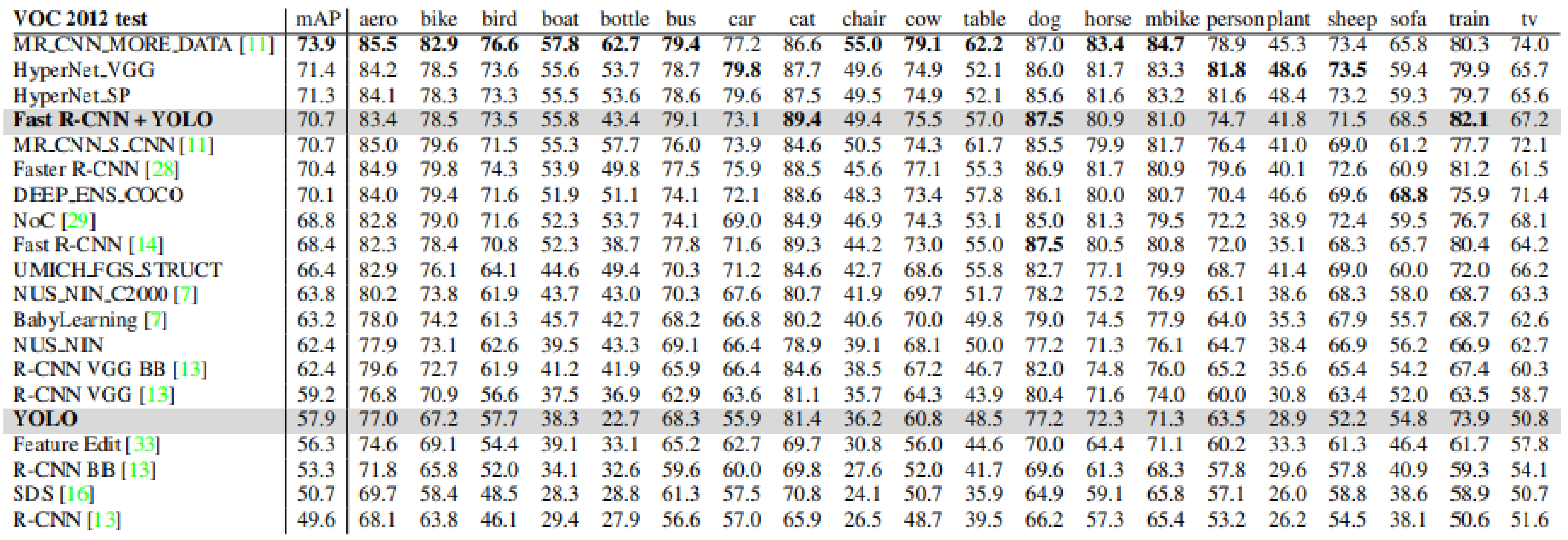


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

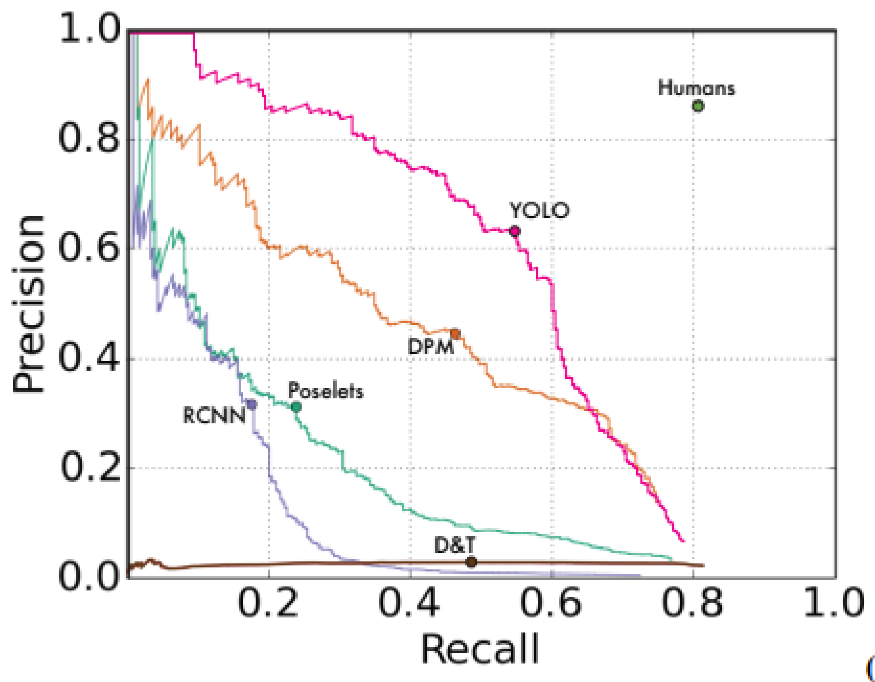
4. Experiments



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.

4. Experiments



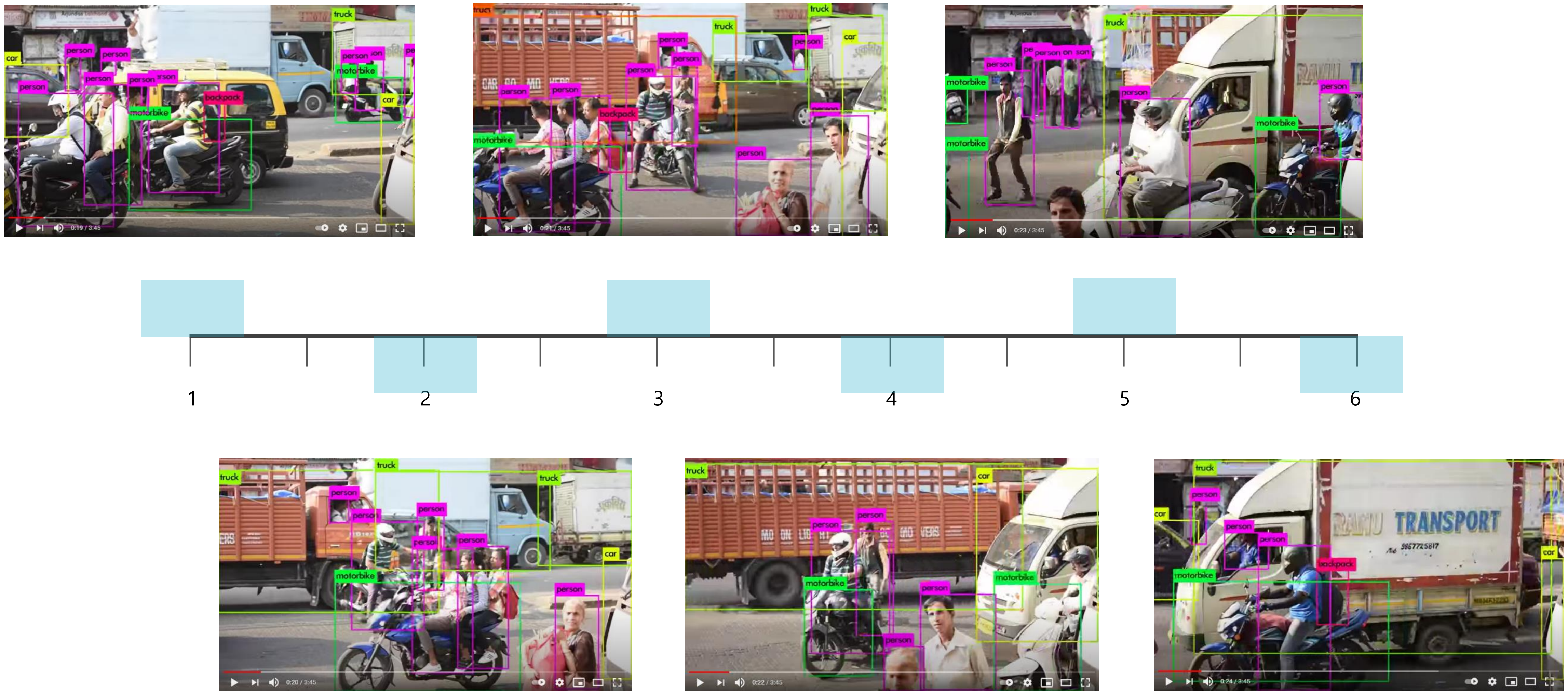
(a) Picasso Dataset precision-recall curves.

	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	

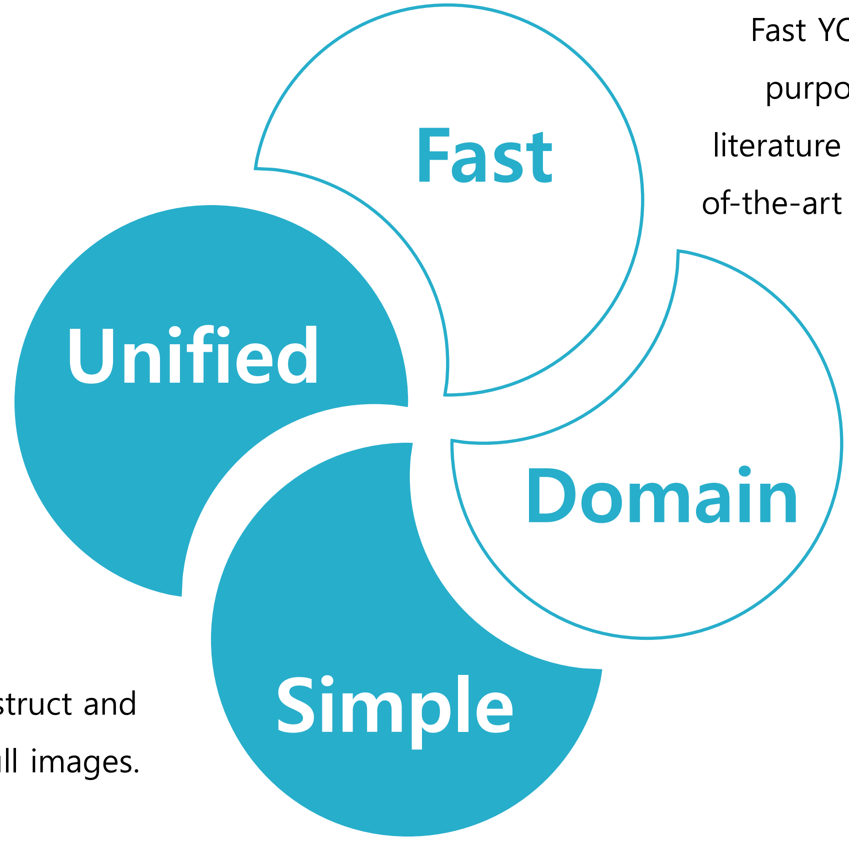
(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 5: Generalization results on Picasso and People-Art datasets.

5. Real-Time Detection In the Wild



6. Conclusion



Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection.

We introduce YOLO, a unified model for object detection.

YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

Our model is simple to construct and can be trained directly on full images.

감사하게도 도움을 받은 참고 문서들

https://bkshin.tistory.com/entry/%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-YOLOYou-Only-Look-Once

https://pjreddie.com/darknet/yolo/

https://curt-park.github.io/2017-03-26/yolo/

https://arclab.tistory.com/167

https://yeomko.tistory.com/19