You only look once: Unified, real-time object detection (YOLO) 2015

Tags	
Author	Redmon, Joseph, et al.
Category	**Object Detection**
pdf	[pdf]

*"객체의 경계 상자(bounding box)탐지와 클래스 분류를 **한 신경망 내에서 수행하도록** 하여 모델의 복잡성을 낮추고 속도를 높인 객체 탐지 모델 YOLO를 선보이는 논문이다."*

Introduction

- object detection
- 기존의 object detection 모델들:
    - DPM: sliding window
        - 윈도우 크기로 분할된 이미지에 대해서 윈도우마다 분류를 수행
    - R-CNN: region proposal-based
        - 이미지에서 객체 영역을 우선 도출하고 (localize) 각 객체 영역에 대해서 분류를 수행

→ 복잡한 파이프라인에 대해 최적화를 달성하기 어렵다. 즉 잘 학습시키기 어렵다.

- 반면, YOLO는 객체 분류 문제를 **회귀**로 접근하여 **하나의 신경망으로 통합된 모델**을 학습시키고 추론 시 테스트 이미지에 대해 학습된 신경망을 바로 적용하여 한 번에 객체 영역과 각각에 대한 분류 라벨을 도출할 수 있다.

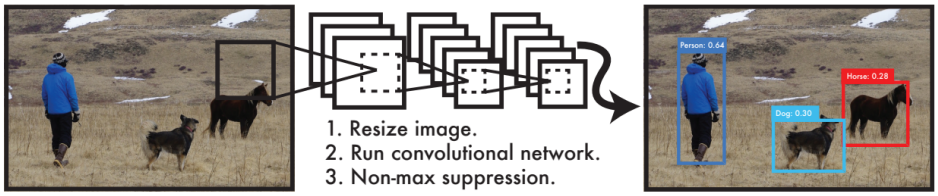


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

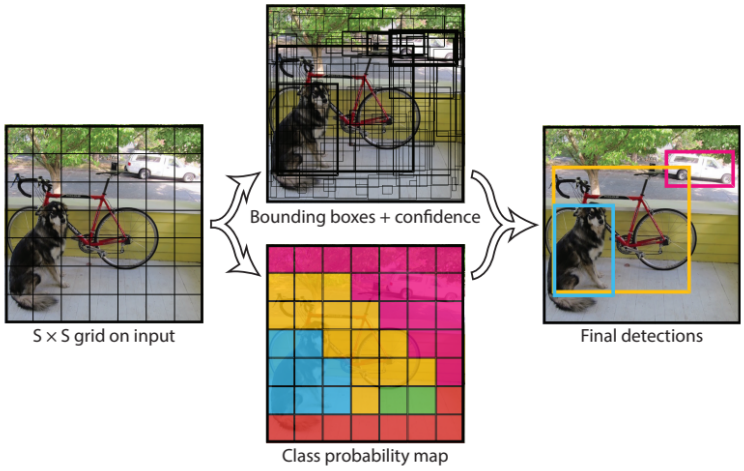
→ 파이프라인이 단순하기 때문에 추론 속도가 빠르다

→ 전체 이미지를 보기 때문에 문맥 정보를 반영하여 배경 오류(background error)를 줄였다.

→ 일반화 성능이 뛰어나다. (자연 이미지에 대해 학습시킨 모델을 예술 이미지에 대해 적용했을 때도 좋은 성능을 보였다.)

→ 한편, 위치 추정(localization) 성능은 떨어졌다.

Model: Unified Detection



1. 전체 이미지를 S×S 의 격자(grid)로 분할한다.
2. 경계 상자 (bounding box) B를 예측한다.
    - 각 B에 대해 계산하는 값들: 신뢰도 점수(confidence score), (x, y), (w, h)

- **신뢰도 점수(confidence score)**:

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

- (x, y): B의 중심 위치
- (w, h): 전체 이미지에서 경계 상자 B가 차지하는 너비(weight)와 높이(height)

3. 각 격자 영역(grid cell)에 대해 해당 영역이 객체를 포함할 때 **특정 분류 클래스에 해당할 조건부 확률**을 계산한다.

$$Pr(Class_i|Object)$$

4. 위에서 구한 값들을 바탕으로 **클래스별 신뢰도 점수(class-specific confidence score)**를 구한다.
    - 1) 실제 객체 영역 박스와 경계 상자 영역(B) 이 일치하는 정도와 2) 경계 상자 영역에 해당 클래스가 존재할 확률 모두를 반영하는 값이다.

$$Pr(Class_i|Object) * Pr(Obejct) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

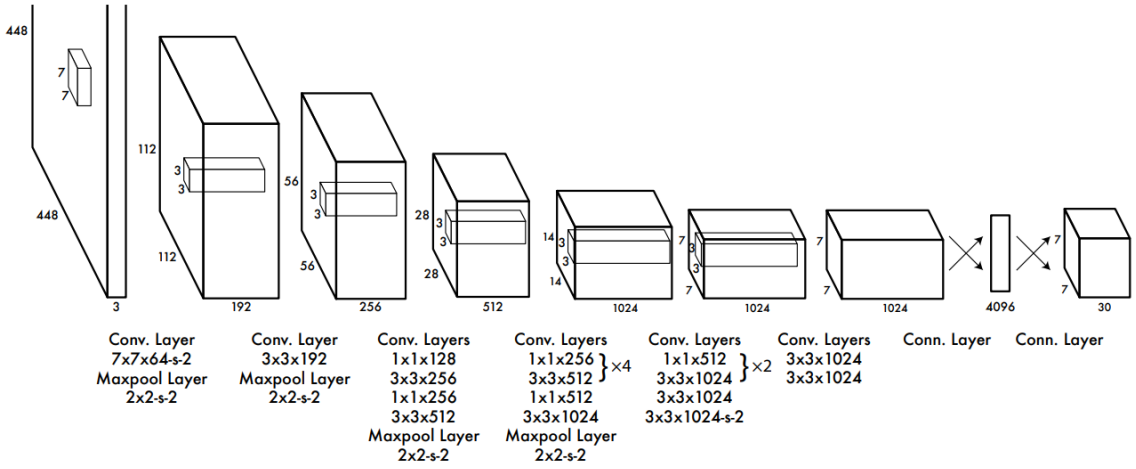
5. 모델 예측 Tensor: S×S×(B*5+C)

B*5? B 하나 당 x, y, w, h, confidence 5개의 값 도출하기 때문.

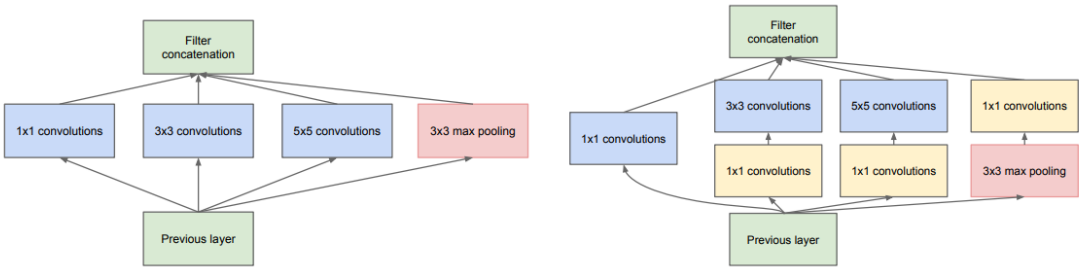
> 결국 학습 데이터셋의 정답 박스 (groundtruth box)는 사람이 일일히 그려줘야 하는 것이다. ~~(PASCAL VOC)~~ → 지도 학습(학습 데이터 수집 비용이 천문학적일 듯)
> IOU는 정답 박스와 예측 박스(B)가 겹치는 정도를 의미한다.

Network Design

- 합성곱 신경망 (Convolutional Neural Network) - GoogLeNet의 아키텍처를 차용
- 24 CONV - 2 FC



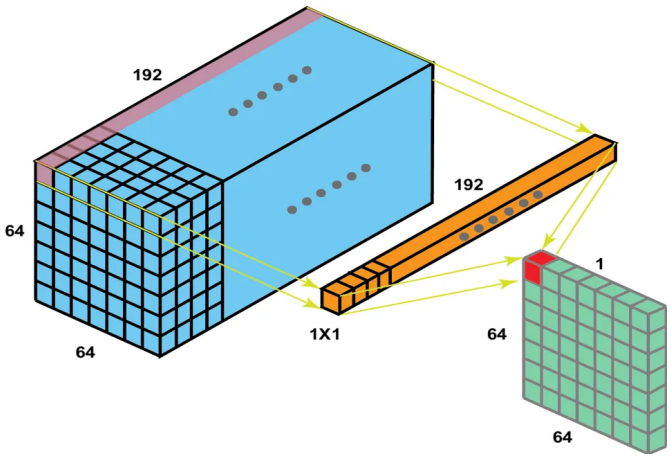
- Inception Modules(from GoogLeNet) 대신 1×1 reduction layers를 사용
  - Inception module
    - 서로 다른 크기의 필터들을 동시에 적용하여 다양한 특징들을 학습시키려고 하였다.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

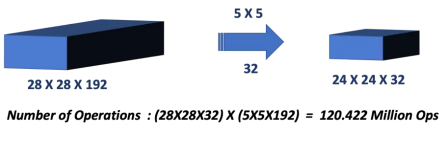
- 1×1 convolutional layers
  - 필터(커널)의 크기가 1인 신경망 층을 의미 (→ 아웃풋 크기는 인풋 크기와 같을 것, 다만 채널이 축소된다.)

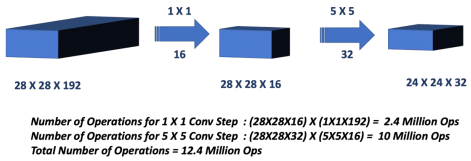


  - 기대효과:
    - 차원 축소 (Dimensionality Reduction), 파라미터 수 축소

      → 연산의 효율성 개선하면서도 모델에 비선형성(non-linearity)를 추가할 수 있다.

      연산량 비교: 120.422M vs 12.4M





- A Fast version of YOLO: 합성곱 층의 개수를 축소한(24 → 9) 버전이다.

Training

- 처음 20개의 합성곱 층(CONV)를 이미지 분류 데이터셋(ImageNet)으로 사전 학습시키고(pre-train)
- 이후 4개의 합성곱 층(CONV)과 두 개의 완전 연결층(FC)를 추가하여 객체 탐지 데이터셋에 대해(어떤 것?) 학습시켰다.
- 예측 결과로 `각 클래스에 대한 확률값` 과 `경계 상자 좌표` 가 도출되었다.

[Detail]

- 객체 탐지(object detection) 학습에 맞게 해상도(resolution)를 증가시켰다.(224×224 → 448×448)
- 경계 상자의 너비와 높이 (w, h)를 전체 이미지에 대한 비율로 정규화시켰다. → [0, 1]
- 경계 상자의 중심 좌표(x, y)를 해당 중심 좌표를 포함하는 격자 영역(grid cell) 내에서의 위치로 매개변수화했다. → [0, 1]
- 활성화 함수
  - 마지막 층에 대해서는 선형 활성화 함수 적용 (아마 `y=x` ?)
  - 나머지 층에 대해서는 leaky ReLU 적용

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

- Loss 계산: Sum-Squared Error을 변형

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

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

i번째 grid cell에 있는 j번째 bounding box가 IOU값이 가장 높아 예측자로 선택됨 → 예측자로 선택되지 않은 bounding box는 0

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

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

i번째 grid cell에 객체가 있는지→ 존재하면 1

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

- 기본 SSE

$$SSE = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

  - 한계:
    - 위치 추정 에러를 이미지 분류 에러의 가중치와 같게 설정한다.
    - 한 이미지에서 대부분의 격자 영역은 객체를 포함하지 않는데, 이 영역들의 신뢰도 점수(confidence score)를 0으로 끌어내려서 객체를 포함하는 영역의 기울기(gradient)를 압도해버린다.

      → 변형 1. 손실 가중치 조정

      - 좌표 예측에 대한 손실 가중치를 증가시키고

        → $\lambda_{coord} = 5$
      - 객체가 없는 셀의 신뢰도 예측에 대한 손실 가중치를 감소시킨다.

        → $\lambda_{noobj} = 0.5$

        - 같은 손실이라도 작은 경계 상자에 대한 오차가 큰 경계 상자에 대한 오차보다 영향력이 큰데, SSE는 이를 반영하지 못하고 오차에 대한 가중치를 같게 설정한다.

          → 변형 2. 예측한 경계 상자의 너비와 높이(w, h)에 대해 제곱근을 구해서 사용한다.

    - YOLO는 하나의 격자 영역에 대해 여러 경계 상자를 예측하는데, 하나의 객체 당 하나의 경계 상자만을 필요로 하므로 IOU값이 가장 높은 경계 상자를 예측값으로 선정했다.

Inference

- 신경망 하나만 사용해서 추론(One network evaluation) → 빠른 속도
- 비-최대 억제(Non-maximal suppresion)를 사용해서 공간적 다양성을 추가한 격자 디자인 완성
    - 크기가 큰 객체는 여러 격자 영역에 걸쳐 나타날 수 있다.
    - Non-maximal suppresion
        - 여러 bounding box 중에 타당한 box만 선택하는 방법

> 1. 하나의 클래스에 대한 bounding boxes 목록에서 가장 높은 점수를 가지고 있는 box를 final box에 추가하고 목록에서 제거한다.
> 2. 목록에 남아있는 boxes 중에서 가장 높은 점수를 가지고 있는 box만을 final box에 추가하고 bounding boxes 목록에서 제거한다.
> 3. IOU의 임계값(threshold)를 설정하고, bounding boxes 목록 중에서 IOU가 임계값보다 낮은 것들만 남긴다.
> 4. 2의 과정부터 bounding boxes에 남아 있는 box가 없을 때까지 반복한다.

    - mAP를 23% 향상시켰다.

Limitations of YOLO

- 공간적 제약 (Spatial constaints)
    - 각 격자 영역마다 2개의 경계 상자만을 예측한다. (B=2로 설정)
    - 예측할 수 있는 주변 객체 수에 대한 제약으로 여러 객체가 뭉쳐 있는 경우 (ex. 세 때) 객체 예측을 잘 하지 못했다.
- 형태적 제약 (poor to New and unusual configuarations)
    - 객체의 비율이나 구성이 평소와 다른 경우 예측하지 못한다.
- 낮은 해상도 특성에 기반한 예측 (coarse features)
    - 모델 아키텍처가 입력 이미지로부터 여러 다운샘플링을 거치기 때문에 해상도가 낮은 특성, coarse features 기반으로 경계 상자를 예측한다.

    > **coarse**: 해상도가 낮음 (계산 효율성 증가하지만 세부적인 특성을 파악하기 어려움)
    > vs
    >
    > **fine**: 해상도가 높음 (계산 효율성 감소하지만 세부적인 특성 파악 가능)

- 경계 상자 크기에 따른 오차 영향력 차이의 간과
    - 작은 경계 상자와 큰 경계 상자의 오차를 동일하게 취급하였다.

Experiments

1. VOC 2007에 대해 실시간으로 추론한 결과, YOLO는 기존의 SOTA detectors에 비해 빠르고 성능도 좋았다.

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

2. 오류 유형을 다음과 같이 5개로 구분하여 분석한 결과, YOLO는 Fast R-CNN보다 backgruond error는 적었지만 localization 성능이 저조함을 알 수 있었다.

- Correct: correct class and IOU $> .5$
- Localization: correct class, $.1 <$ IOU $< .5$
- Similar: class is similar, IOU $> .1$

- Other: class is wrong, IOU $> .1$
- Background: IOU $< .1$ for any object

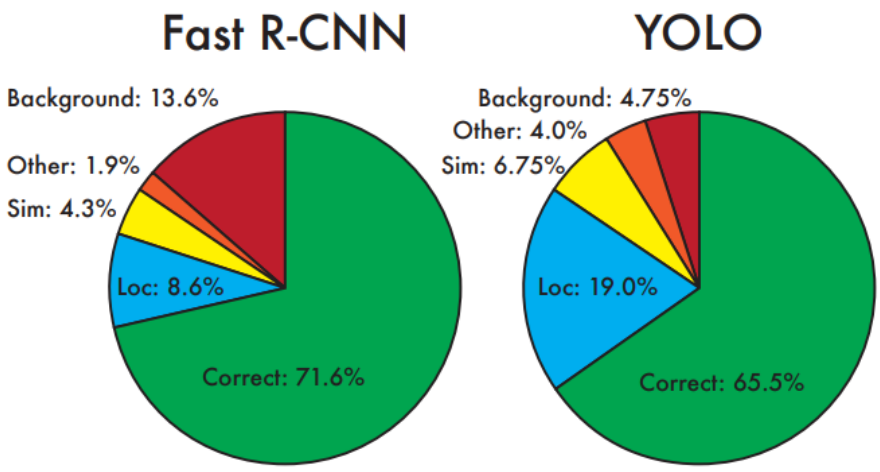


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

3. Fast R-CNN과 YOLO을 결합한 결과, Fast R-CNN을 다른 모델과 결합한 것보다 YOLO와 결합한 것이 성능이 더 좋았다.
    - 결합 방법
    1. R-CNN과 YOLO 각각 경계 상자를 생성한다.
    2. R-CNN에 의해 예측된 경계 상자에 대해 YOLO가 유사한 경계 상자를 예측하는지 보고 만약 일치하는 경우 'boost'를 제공한다. (아마 R-CNN 기준으로 R-CNN 예측값에 대한 점수를 조정해서 R-CNN 모델을 업데이트 하는 것 같다.)

> **boost를 제공한다?**
> → 해당 예측에 대한 점수(확률/신뢰도 점수)를 증가시킨다.
> → YOLO가 예측한 확률과 두 경계 상자(from R-CNN, YOLO)의 겹치는 정도를 기반으로 설정한다.

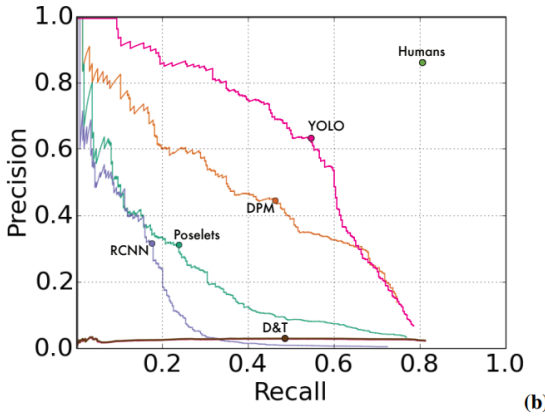
	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	**66.9**	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	**75.0**	**3.2**

- 단순 앙상블로 인한 성능 향상이라기 보다, YOLO는 R-CNN(background)과 다른 종류의 오류(localization)를 발생시키는 경향이 있으므로 이 두 모델이 서로 보완되는 효과가 있는 것으로 보인다.

4. VOC 2012에 대해 추론시킨 결과

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 [27]	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 [28]	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 [32]	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

5. 일반화 성능: 다른 도메인(예술 작품)에서의 인간 탐지하기



(a) Picasso Dataset precision-recall curves.

	VOC 2007 AP	Picasso AP	Picasso Best F_1	People-Art 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.

6. YOLO를 웹캠에 연결해서 실시간 성능을 측정한 결과, 마치 추적 장치처럼 물체가 이동하고 형태가 변형됨에도 일관되게 객체를 탐지해냈다.