👀

You Only Look Once: Unified, Real-Time Object Detection

Abstract & Introduction

Introduction

- 기존의 object detection 방식은 분류기를 재활용하여 detection을 수행

→ 기존 방식의 문제점은, 단계별로 작동되는 pipeline이기 때문에, (ex. R-CNN은 먼저 proposal 된 박스에 대해 CNN 분류기를 적용) 시간과 비용 문제. 그리고 각 구성 요소가 별도로 학습되어야 하기 때문에 최적화가 어렵다는 문제가 있음

- 본 논문에서는 **object detection를 regression 문제로 재정의**하여 하나의 neural network로 전체 이미지에서 bouding box + class predction probability를 동시에 예측 **(따라서 한번에 모든 객체의 위치와 종류를 판단하는 것!)** → pipeline 전체가 하나의 네트워크로 구성되어있기 때문에 detection에 직접적으로 최적화되도록 end-to-end 학습이 가능해짐 + 이미지 전체를 한번에 보기 때문에 전반적인 문맥을 고려한 탐지가 가능해짐
- 기존의 SOTA와 비교해볼 때, YOLO는 위치에 대한 오차는 많지만, 배경에 대한 false-positive ratio가 적음. YOLO은 object의 일반적인 패턴을 학습하여 R-CNN 등의 방법들에 비해 새로운 도메인 (ex. art) 에서도 성능이 뛰어나다는 장점이 있음

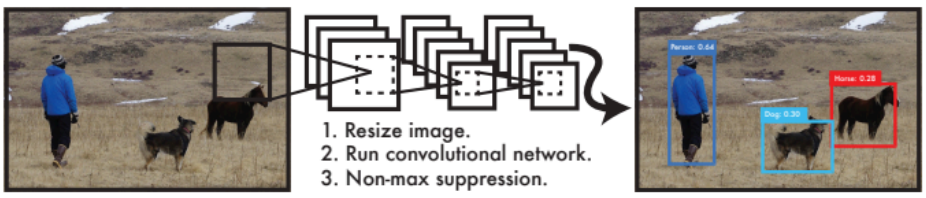


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

- YOLO은 regression 문제로 치환하므로 복잡한 pipeline이 필요가 없고, test 에선 이미지 네트워크를 한번만 통과시키면 detection이 완료되기 때문에 한번만 통과시키면 탐지가 완료됨
- Fast R-CNN은 문맥을 보지 못하기 때문에 배경을 객체로 오탐지 하는 문제 → YOLO는 전역적 이미지 이해가 가능하기에 Fast R-CNN 보다 배경 오류가 절반으로 줄어듦

Method

Unified Detection

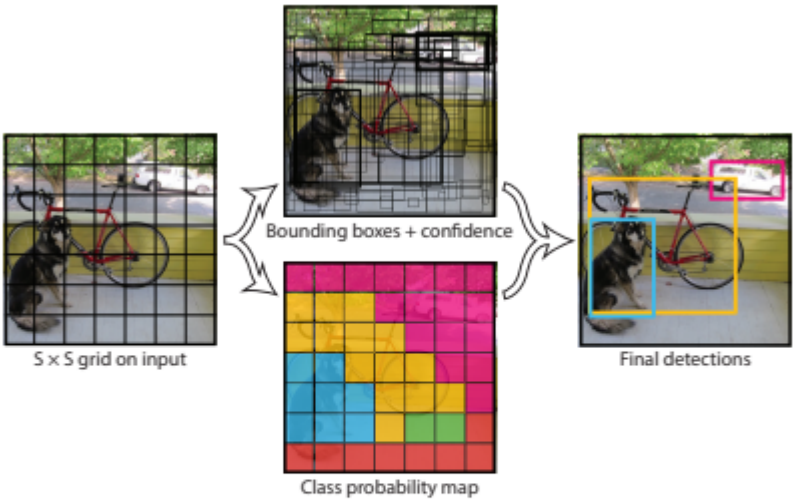


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

- YOLO는 그리드 기반이기에 입력 이미지를 **S×S 그리드**로 나눔 → 어떤 객체의 중심점이 포함된 그리드 셀이 그 객체를 탐지해야 함
  - B개의 bounding box와 해당 신뢰도를 예측 → 이때 신뢰도는 이 박스에 객체가 존재할 확률 × 예측된 박스와 실제 객체 박스 간의 IOU (객체가 없으면 신뢰도=0, 객체가 있다면 신뢰도는 **IOU 값**과 같아야 함)
  - 따라서 bounding box 는 총 5개 예측 : `x, y` : 그리드 셀 내에서의 중심 좌표, `w, h` : 전체 이미지 기준 너비와 높이, `confidence` : 예측 박스와 실제 박스 간의 IOU
  - 또한 각 그리드 셀은 C개의 조건부 클래스에 대한 확률도 예측함. 이때 한 셀에 있는 객체 수와 상관없이 class prob은 1 set만 예측함

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Class}_i) * \text{IOU}^{\text{truth}}_{\text{pred}} \quad (1)$$

test에서 계산 방식은 다음과 같음 → class 별 신뢰도 점수임

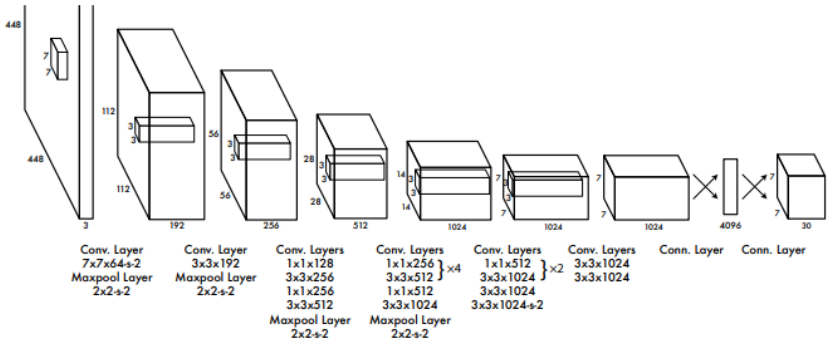


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224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

- 네트워크 디자인은, CNN으로 구현됨. GoogLeNet의 inception 모듈 대신, **1×1 reduction layer + 3×3 convolutional layer**을 사용 → 따라서 전체 네트워크는 24개 convolutional layer + 2개의 fully connected layer
- S=7, bounding box 개수 B=6, class 수 C=20 → 모든 예측은 **S × S × (B×5 + C) tensor로 표현됨**

Training

- 사전학습으로 ImageNet 1000-클래스 데이터셋 사용함. 이때 프레임워크는 Darknet으로 → top-5 정확도 88%
- 여기에 합성곱층 4개, 완전연결층 2개를 추가해서 무작위로 초기화하여 detection 용 네트워크로 전환함 + 이미지를 2배 업스케일 (더 섬세한 시각 정보가 필요하므로) + 최종 output layer는 바운딩 박스의 너비와 높이를 이미지 크기로 정규화고 x,y 좌표를 그리드 셀 기준 오프셋을 표현하여 (모두 0-1사이 값을 가지게 됨) 최종 layer를 linear activation ftn, 나머지는 Leaky ReLU를 사용함

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \quad (2)$$

leaky relu formula

- 이때 손실함수는 출력에 대한 summation of squared error를 사용. cls/reg(위치에 대한) loss에게 같은 가중치를 부여하면 작은 박스에서와 큰 박스에서의 오차를 똑같이

취급하기에 부적절함. 또한 이미지에는 대부분 객체가 없는 grid cell 이 많기 때문에 confidence=0인 경우가 많고 이는 학습이 불안정해지는 원인이 됨

loss function:

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

- 따라서 bounding box에 대한 loss 증가시키고, 객체가 없는 박스에 대한 confidence loss를 줄임 → 두개의 lambda parameter를 사용해서
- 또한 작은 박스는 약간의 오차도 크게 영향을주기 때문에, 너비와 높이 대신 루트를 씌운 너비와 높이를 예측하여 작은 박스일 수록 오차를 더 민감하게 반영하도록

  → YOLO는 한 셀에서 여러 개의 box를 예측하지만, **학습 시에는 각 객체당 하나의 box predictor가 책임을 짐 (이때 책임자는 현재 예측된 박스 중 ground truth와 가장 IOU가 높은 박스 선택)**

  → 각 박스 predictor의 전문화 성능 향상 + recall 향상
- training dataset : PASCAL VOC 2007 + 2012 학습/검증셋 → 총 약 135 epoch (VOC 2012 테스트 시에는 VOC 2007 테스트 데이터도 훈련에 포함)
- hyperparams set: batch size = 64, momentum = 0.9, learning rate decay = 0.0005
- overfitting을 방지하기 위해 dropout prob=0.5, data augmentation (무작위 스케일링+이동을 20%, 노출과 채도 조절은 1.5배 정도)

Inference

- 학습 때와 마찬가지로, 테스트 시에도 한 번의 네트워크 실행만으로 탐지 가능
- PASCAL VOC 기준으로 한 이미지당 98개의 바운딩 박스와 클래스 확률을 예측
- 일반적으로 한 객체에 대해 한 셀만 예측하지만 큰 객체나 셀 경계에 객체는 여러 셀에서 예측이 가능 **by NMS 적용**

Limitation

- YOLO 는 공간적 제약이 강함. 각 셀은 2개의 박스, 하나의 클래스만 예측이 가능하기 때문에 근처에 있는 여러 객체를 구별하기 어렵다는 문제 + 작은 객체가 무리지어 있을 때 성능이 저하됨
- 또한 박스를 예측할 때, 아키텍처가 다운샘플링을 여러번 적용하기 때문에 정밀한 위치를 예측하는 것이 어려움
- loss 가 작은/큰 박스에서 loss를 동등하게 다루기 때문에 localization error 발

Comparison to Other Detection Systems

- 기존의 detection model은 1) 이미지에서 강력한 feature를 추출, 2) Sliding window 또는 영역제안으로 후보 영역을 설정, 3) 분류기로 객체를 판단
- **vs DPM (Deformable Parts Model)**: DPM은 sliding window 방식을 사용하고 각 pipeline 의 구성요소가 별도로 동작함. 따라서 YOLO가 더 빠르고 정확한 성능을 제공
- **vs R-CNN:** Selective Search로 박스 후보를 생성하여 CNN으로 특징을 추출하고 SVM으로 분류 후 linear model로 박스를 조정하는 방식임. 따라서 각 단계가 별도로 최적화가 필요하고 매우 느림. YOLO는 그리드 셀이 박스를 예측하고 CNN이 점수를 부여한다는 점에선 비슷하지만. 훨씬 더 적은 박스를 예측하고 하나의 통합된 모델임
- **vs DeepMultiBox**: R-CNN과 달리 CNN이 ROI를 예측하는 방식임. 이는 단일 객체 탐지는 가능하지만, 일반적인 객체 탐지는 불가능함
- **vs Overfeat:** CNN으로 위치를 학습한 후 이를 탐지로 전환하는 방식임. 이는 sliding window 방식이기에 전체적인 맥락을 고려하지 못함
- **vs MultiGrasp:** YOLO의 그리드 구조와 유사한 grasp detection 시스템이지만 단일 객체만 예측하므로 YOLO에비해 단순함

→ YOLO는 다중 객체 다중 클래스에 모두 대응이 가능한 detector

Experiments & Conclusion

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.

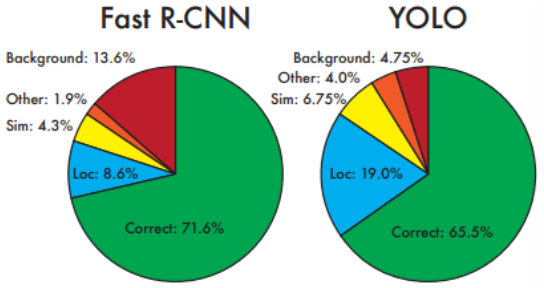


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

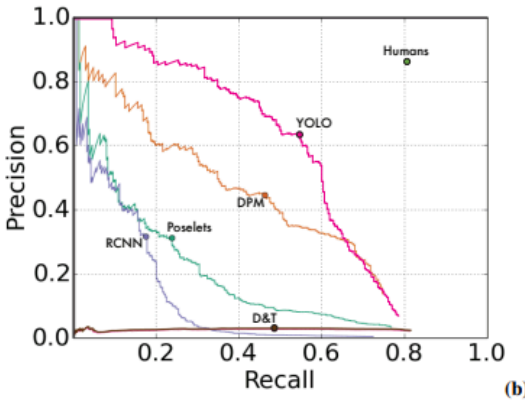
- Other: class is wrong, IOU $>$ .1
- Background: IOU $<$ .1 for any object

	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**

Table 2: Model combination experiments on VOC 2007. We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

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.



(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.

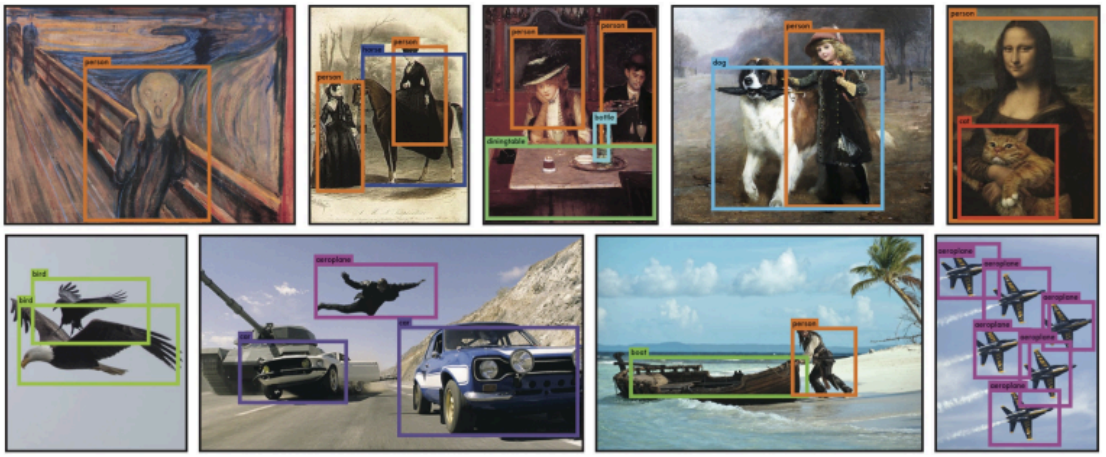


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

- 본 논문의 모델은 **구성이 간단**하며, **전체 이미지를 대상으로 직접 학습**이 가능
- 기존의 classifier 기반 접근 방식과 달리, YOLO는 detection 성능과 직접적으로 연관된 loss function을 사용하여 학습이 되고, 모델 전체가 공동으로 학습이 됨
- 실시간 객체 탐지 분야에서 SOTA 달성 + 새로운 도메인에 대한 일반화 성능이 뛰어나기에 빠르고 안정적인 object detection에 이상적