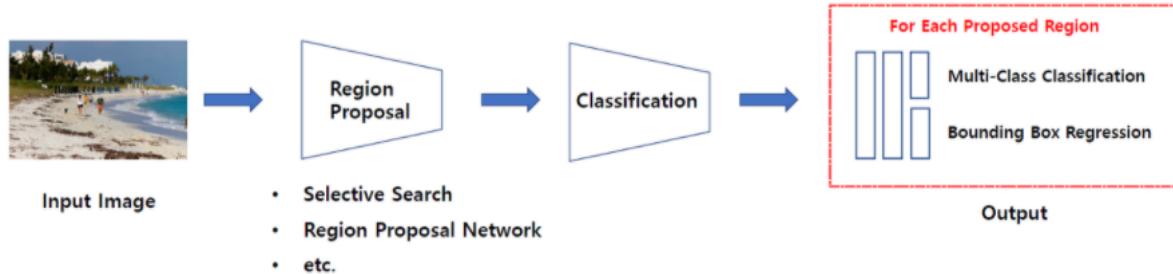


YOLO: You Only Look Once: Unified, Real-Time Object Detection

Object Detection의 두 가지 접근법

1. 2-stage Detector

- 2-stage Detector은 특징 추출과 객체 분류, 두 가지 과정을 거쳐 객체를 탐지함
- 특징 추출과 객체 분류라는 두 가지 문제를 순차적으로 해결함
- 역할을 분담하여 문제를 처리하므로 정확도는 높지만, 속도가 느리다는 단점이 있음
- 2-stage Detector에는 대표적으로 Fast R-CNN, OverFeat, DPM 등이 있음



- 논문에서 2-stage Detector 기법인 DPM과 R-CNN을 소개

(1) DPM

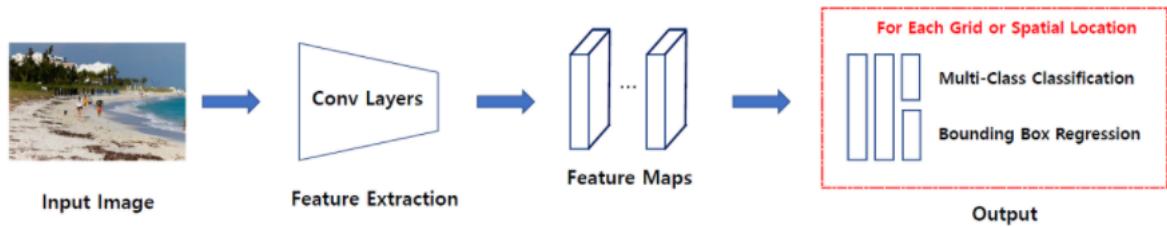
- DPM은 sliding window 기법을 이용
- 전체 이미지에 대하여 작은 구역에 일정한 간격(sliding window)을 두고 classifier를 실행

(2) R-CNN

- R-CNN은 region proposal(selective search를 이용)로 객체가 있을 법한 구역을 제안하고, 그 구역을 classifier로 분류
- 출력값으로 생성된 바운딩박스를 전처리과정(bounding box regression)을 거쳐 최종 바운딩박스를 선택
- 2-stage Detector은 속도가 느리고, 최적화하기 어려움
 - 그 이유는 각 요소를 개별적으로 학습시켜줘야 하기 때문

2. 1-stage Detector

- 1-stage Detector은 특징 추출과 객체 분류를 한 번에 처리하는 방법
- 표적으로 YOLO 계열과 SSD 계열이 있음



- 위 그림에서 보듯이 특징 추출, 객체 분류 두 문제를 단일 신경망으로 해결
- 논문에서는 YOLO가 object detection을 single regression problem으로 재구성

YOLO

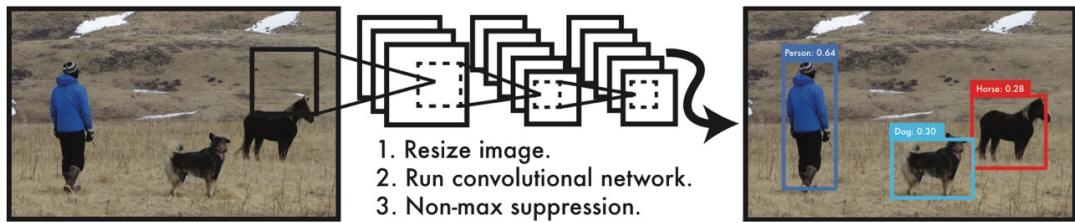
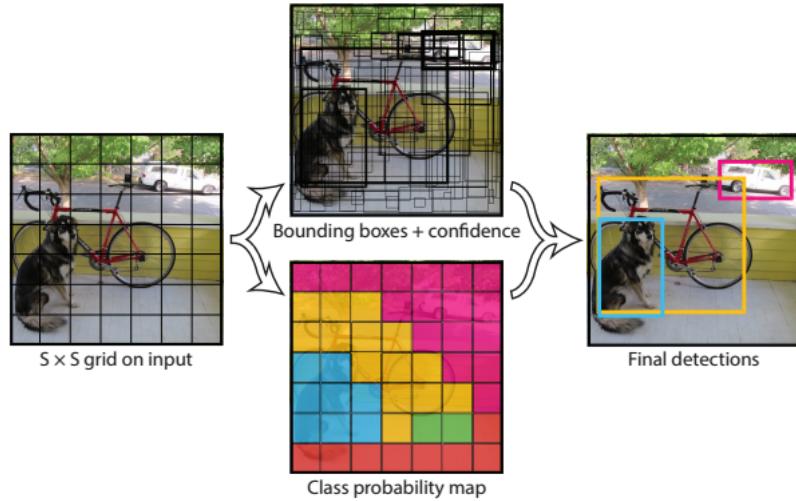


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

- YOLO v1은 localization과 classification을 하나의 문제로 정의하여 network가 동시에 두 task를 수행하도록 설계
- 이를 위해 이미지를 지정한 grid로 나누고, 각 grid cell이 한번에 bounding box와 class 정보라는 2가지 정답을 도출하도록 만듬
- 또한 각 grid cell에서 얻은 정보를 잘 feature map이 잘 encode할 수 있도록 독자적인 Convolutional Network인 DarkNet을 도입
- 이를 통해 얻은 feature map을 활용하여 자체적으로 정의한 regression loss를 통해 전체 모델을 학습
- 두 문제를 한꺼번에 처리를 하여 속도는 빠르지만 정확도가 떨어짐
- 이처럼 속도와 정확도에는 trade off 관계를 형성

Main Ideas

1. 1-stage detector



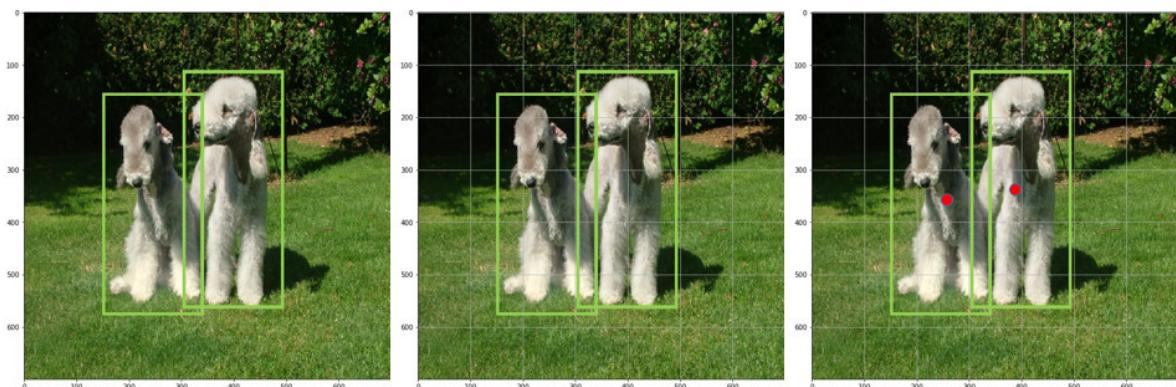
YOLO는 object detection 작동 순서 & 내용

(1) 입력 이미지를 $S \times S$ grid로 분할합니다.

- YOLO v1은 별도의 region proposals를 사용하지 않고 전체 이미지를 입력하여 사용

(2) 객체의 중심이 grid cell에 맞아 떨어지면 그 grid cell은 객체를 탐지했다고 표기합니다.

- 객체의 중심이 특정 grid cell에 위치한다면, 해당 grid cell은 그 객체를 detect하도록 할당(responsible for)



- 위의 그림을 보면 4행 3열의 grid cell이 왼쪽의 개를 예측하도록 할당되었고, 4행 4열의 grid cell이 오른쪽의 개를 예측하도록 할당
- 이는 곧 나머지 grid cell은 객체를 예측하는데 참여할 수 없음을 의미

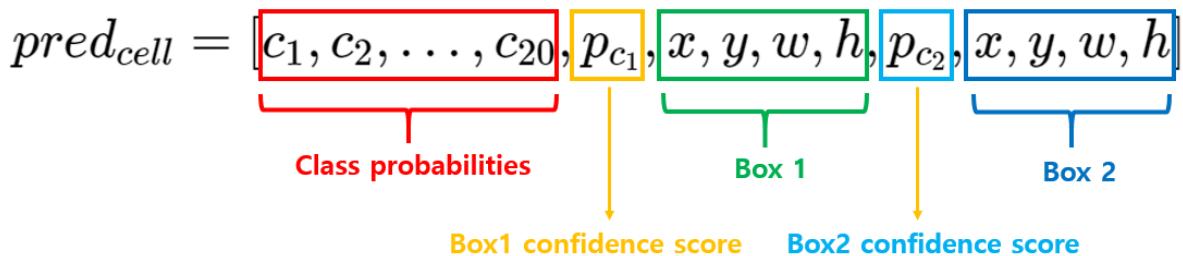
(3) 각 grid cell은 B개의 바운딩박스와 각 바운딩박스에 대한 confidence score를 예측합니다.

- **confidence score**는 해당 bounding box에 객체가 포함되어 있는지 여부와, box가 얼마나 정확하게 ground truth box를 예측했는지를 반영하는 수치

- confidence score는 $Pr(\text{Object}) * IoU(\text{truth}, \text{pred})$ 로 정의
- 만약 grid cell 내에 객체가 존재하지 않는다면 confidence score는 0이 될 것
- 반대로 grid cell 내에 객체가 존재한다면 confidence score는 IoU 값과 같아짐

(4) 각 바운딩 박스는 5개의 정보를 갖고 있음

- 각각의 bounding box는 box의 좌표 정보(x, y, w, h)와 confidence score라는 5개의 예측값을 가짐
 - 여기서 (x, y)는 grid cell의 경계에 비례한 box의 중심 좌표를 의미
 - 높이와 너비는 역시 마찬가지로 grid cell에 비례한 값
- 여기서 주의할 점은 x, y 는 grid cell 내에 위치하기에 0~1 사이의 값을 가지지만 객체의 크기가 grid cell의 크기보다 더 클 수 있기 때문에 width, height 값은 1 이상의 값을 가질 수 있음
- 마지막으로 confidence는 예측된 박스와 진짜 박스 사이의 IoU임
 - 하나의 bounding box는 하나의 객체만을 예측하며, 하나의 grid cell은 하나의 bounding box를 학습에 사용
 - 예를 들어 grid cell별로 B개의 bounding box를 예측한다고 할 때, confidence score가 가장 높은 1개의 bounding box만 학습에 사용



- 각 grid cell은 C개의 conditional class probabilities인 $Pr(\text{Class}_i | \text{Object})$ 를 예측
- 이는 특정 grid cell에 객체가 존재한다고 가정했을 때, 특정 class i 일 확률인 조건부 확률값
- bounding box 수와 상관없이 하나의 grid cell마다 하나의 조건부 확률을 예측
- 여기서 짚고 넘어갈 점은 bounding box별로 class probabilities를 예측하는 것이 아니라 grid cell별로 예측한다는 것

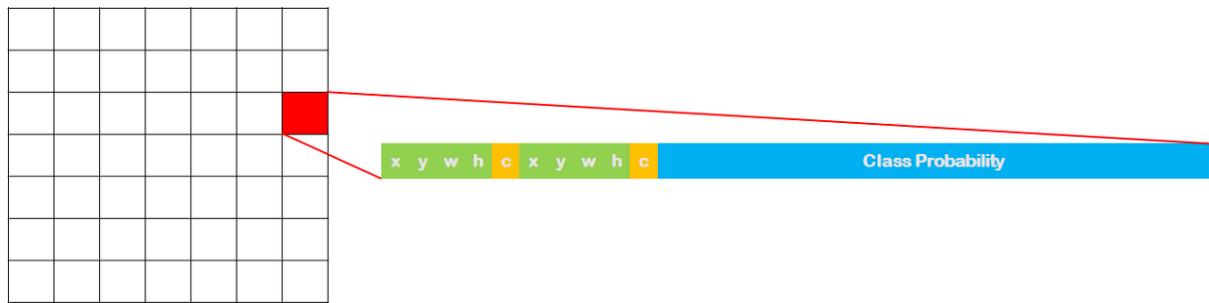
(5) 각 grid cell은 바운딩박스 이외에도 class 확률을 예측

- 최종적으로 예측값은 $(S \times S \times (B * 5 + C))$ 크기의 tensor를 갖습니다. 논문에서는 $S = 7, B = 2, C = 20$ 을 사용하여 $7 \times 7 \times 30$ tensor를 갖습니다.

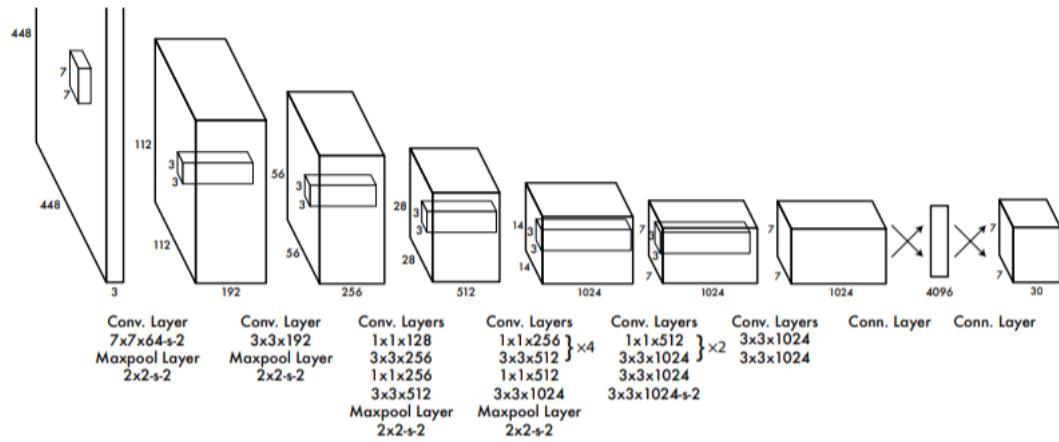
- 이미지를 7×7 grid로 나누고 각 grid cell은 2개의 bounding box와 해당 box의 confidence score, 그리고 C개의 class probabilities를 예측
- 이와 같은 과정을 통해 bounding box의 위치와 크기, 그리고 class에 대한 정보를 동시에 예측하는 것이 가능

(6) non-max suppression을 거쳐서 최종 바운딩박스를 선정





2. DarkNet



- YOLO v1 모델은 앞서 살펴본 최종 예측값의 크기인 $7 \times 7 \times 30$ 에 맞는 feature map을 생성하기 위해 **DarkNet**이라는 독자적인 Convolutional Network을 설계
- YOLO는 24개의 convolutional layer와 2개의 fully connected layer로 이루어져 있음
- GoogLeNet에서 영향을 받아 1×1 차원 감소 layer 뒤에 3×3 convolutional layer를 이용

3. Loss Function

Localization loss

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

Confidence loss

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

Classification loss

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

- 기존 R-CNN 계열의 모델이 classification, localization task에 맞게 서로 다른 loss function을 사용했던 것과 달리 YOLO v1 모델은 regression 시 주로 사용되는 **SSE(Sum of Squared Error)**를 사용
- Localization loss, Confidence loss, Classification loss의 합으로 구성

Localization loss

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

- λ_{coord} :
 - 많은 grid cell은 객체를 포함하지 않아 confidence score가 0이 되어 객체를 포함하는 grid cell의 gradient를 압도하여, 모델이 불안정해질 수 있음
 - λ_{coord} 는 이러한 문제를 해결하기 위해 객체를 포함하는 cell에 가중치를 두는 파라미터
 - $\lambda_{\text{coord}} = 5$ 로 설정

- S^2 :

- grid cell의 수($=7 \times 7 = 49$)

- B :

- grid cell별 bounding box의 수($=2$)

- $1_{i,j}^{obj}$:
 - i 번째 grid cell의 j 번째 bounding box가 객체를 예측하도록 할당(responsible for) 받았을 때 1, 그렇지 않을 경우 0인 index parameter
 - grid cell에서는 B개의 bounding box를 예측하지만 그 중 confidence score가 높은 오직 1개의 bounding box만을 학습에 사용
- x_i, y_i, w_i, h_i :
 - ground truth box의 x,y 좌표와 width, height
 - 크기가 큰 bounding box의 작은 오류가 크기가 작은 bounding box의 오류보다 훨씬 중요하다는 것을 반영하기 위해 w_i, h_i 값에 루트를 씌어주게 됨
- $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$:
 - 예측 bounding box의 x, y 좌표, width, height

→ 해석

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

- x,y는 바운딩 박스 좌표입니다. 정답 좌표랑 예측 좌표 차이를 제곱하여 error를 계산
- $1_{i,j}^{obj}$ 는 obj는 객체가 탐지된 바운딩 박스를 말함(가장 높은 IOU를 갖고 있는 바운딩박스)
- 이 수식은 i번째 grid에서 j번째 객체를 포함한 바운딩박스를 의미
- λ_{coord} 는 localization error의 페널티를 키우기 위한 파라미터로 5로 설정
- 즉, 객체를 포함한 바운딩박스의 좌표에는 5배의 페널티를 부과

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

- 바운딩 박스의 너비와 높이에 대한 error
- 너비와 높이에도 5배의 페널티를 부여하는 것을 확인할 수 있음
- 큰 박스에서 작은 변화와 작은 박스에서 작은 변화는 차이가 있음
- 이를 보정해주기 위해 루트를 씌어준 다음 제곱을 해줌

Confidence loss

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

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

- λ_{noobj} :
 - 앞서 언급한 객체를 포함하지 않는 grid cell에 곱해주는 가중치 파라미터
 - $\lambda_{\text{noobj}} = 0.5$ 로 설정
 - $\lambda_{\text{coord}} = 5$ 로 설정한 것에 비해 상당히 작게 설정하여 객체를 포함하지 않은 grid cell의 영향력을 줄임
- $\mathbb{1}_{i,j}^{\text{noobj}}$:
 - i 번째 grid cell의 j 번째 bounding box가 객체를 예측하도록 할당(responsible)받지 않았을 때 1, 그렇지 않을 경우 0인 index parameter
- C_i : 객체가 포함되어 있을 경우 1, 그렇지 않을 경우 0
- \hat{C}_i : 예측한 bounding box의 confidence score

→ 해석

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

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

- 객체를 포함한 바운딩 박스에 대한 confidence error

Classification loss

$$+ \sum_{i=0}^S \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (P_i(c) - \hat{P}_i(c))^2$$

- $P_i(c)$:
 - 실제 class probabilities

- $\hat{P}_i(c)$:
 - 예측한 class probabilities

→ 해석

- $p(c)$ 클래스 확률에 대한 error
- 객체를 포함한 바운딩박스에 대해서만 계산
- classification error에 대한 부분으로 생각할 수 있음

Experiment

1) 논문에서는 YOLO를 다른 실시간 detection systems을 PASCAL VOC 2007에서 비교

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
<hr/>			
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

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.

- YOLO는 R-CNN의 background false positives의 수를 줄여 큰 성능 향상

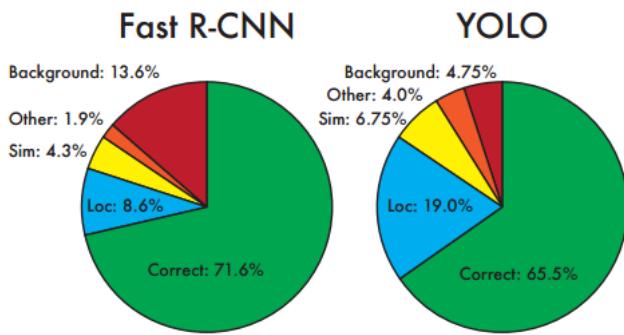


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories ($N = \#$ objects in that category).

- 빨간부분의 background부분에서 YOLO가 더 error가 적음을 보여줌

	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.

- 모든 R-CNN이 예측한 bounding box에서 YOLO또한 비슷하게 예측을 하고 있는지 체크하고, 만약 그렇다면 YOLO가 예측한 확률과 두 박스간의 겹치는 면적을 기반으로 boost를 주는 방식
- 위에서 볼 수 있듯이 합친 버전은 원래의 Fast R-CNN보다 mAP를 3.2% 증가시켰음을 알 수 있음

2) VOC 2012에 대한 결과도 보여주었으며 당시 SOTA 시스템과 mAP를 통해 비교

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

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.

- 기본 YOLO모델은 다른 SOTA 시스템보다 특정 카테고리에서 낮게 나왔음을 확인할 수 있음
- 이는 이전에 언급한 YOLO의 한계점 때문인데, 이와달리 YOLO를 Fast R-CNN와 합친 버전은 우수한 성능을 보여줌

3) 마지막으로, YOLO가 새로운 도메인에 대해서 다른 모델들보다 우수함을 확인

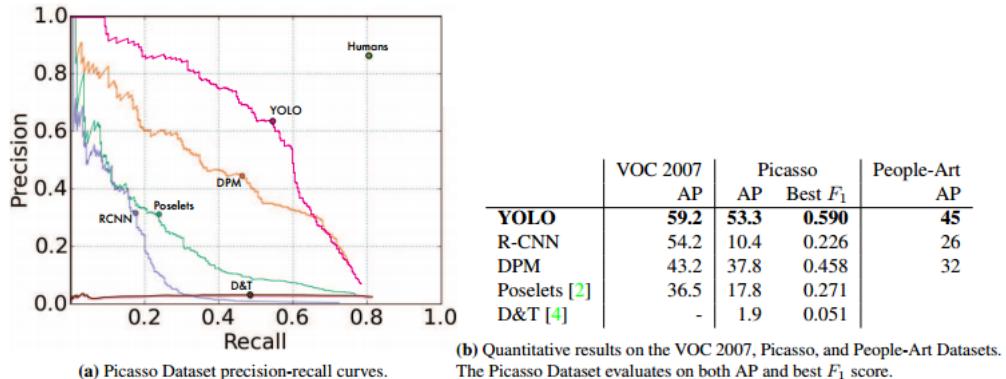


Figure 5: Generalization results on Picasso and People-Art datasets.

- 다른 domain에서의 detection이 다른 system보다 월등히 좋음

Conclusion

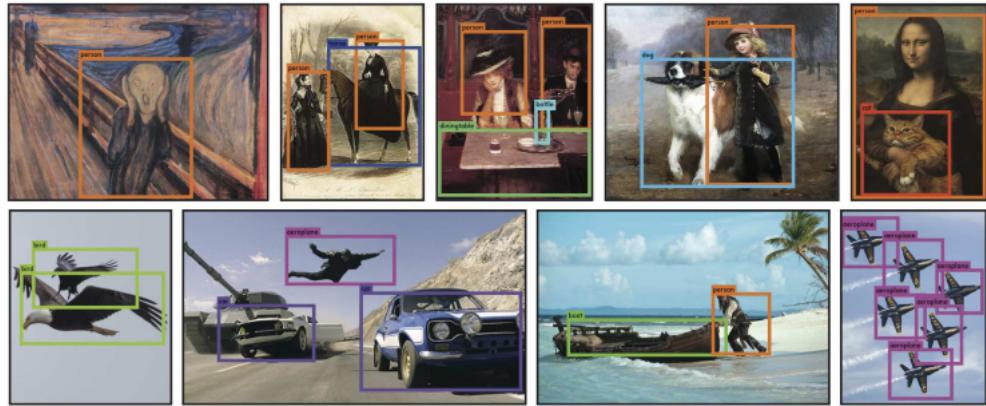


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

Yolo V1 장점

1. 매우 빠름

- YOLO는 복잡한 pipeline이 필요하지 않음
- 이미지에 neural network를 실행하기만 하면 됨
- 추가적으로 YOLO는 실시간 객체 탐지 방법보다 2배 이상의 mAP(mean average precision)을 얻었다고 논문에서 말하고 있음
- YOLO base 모델은 Titan X GPU에서 batch processing 없이 1초당 45개의 프레임을 처리 가능
- 그리고 base보다 빠른 버전인 YOLO fast는 150fps 이상 빠르다고 함
- 이는 streaming video 또한 실시간으로 25ms 이하의 latency로 처리 가능함을 의미

2. Fast R-CNN 보다 background error가 두 배이상 적음

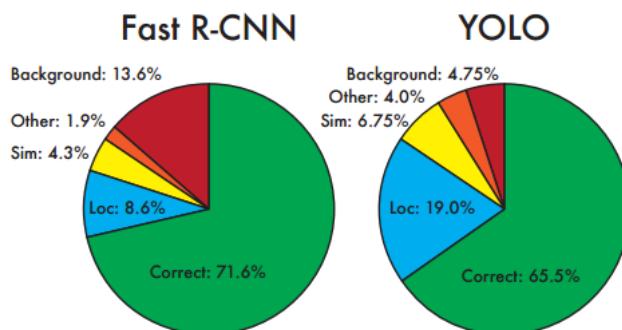


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

- background error는 배경 이미지에 객체가 있다고 탐지한 것을 의미
- YOLO는 예측할 때, 이미지 전체를 이용
- 이 덕분에 class와 객체 출현에 대한 contextual information까지 이용할 수 있음
- 반면에 Fast R-CNN은 selective search가 제안한 영역만을 이용하여 예측하기 때문에 larger context를 이용하지 못함
- 따라서 R-CNN은 배경을 객체로 탐지하는 실수를 하게 됨

3. 객체의 일반화된 representations를 학습하여 다른 도메인에서 좋은 성능

- YOLO는 새로운 이미지나 새로운 도메인에 적용할 때, DPM, R-CNN 같은 detection 방법을 크게 능가함
- 전체 이미지에 대하여 학습을 하기 때문인
- 이러한 특성 덕분에 새로운 도메인에 적용하여 fine-tunning에 이점이 있음

Yolo V1의 한계점

1. 각 grid cell은 하나의 클래스만을 예측, object가 겹쳐있으면 제대로 예측하지 못함
2. 바운딩 박스 형태가 data를 통하여 학습되므로, 새로운/독특한 형태의 바운딩박스의 경우 정확히 예측하지 못함
3. 작은 바운딩 박스의 loss가 IoU에 더 민감하게 영향을 줌, localization에 안좋은 영향을 미침

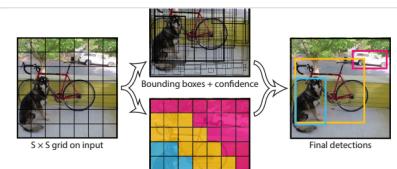
참고자료

- 블로그

[논문 리뷰] YOLO v1 (2016) 리뷰

이번에 리뷰할 논문은 'You Only Look Once: Unified, Real-Time Object Detection'입니다. Deep Learning을 이용한 object detection 접근법은 크게 두 가지로 나눌 수 있습니다. 1. 2-stage Detector 2-stage Detector은 특정 추출과 객체 분류, 두 가지 과정을 거쳐 객체를 탐지합니다. 특징 추출과 객체 분류라는 두 가지 문제를 순차적으로 해

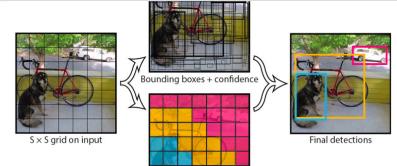
*: <https://deep-learning-study.tistory.com/430?category=968059>



YOLO v1 논문(You Only Look Once:Unified, Real-Time Object Detection) 리뷰

이번 포스팅에서는 YOLO v1 논문(You Only Look Once:Unified, Real-Time Object Detection) 논문을 리뷰해보도록 하겠습니다. 2-stage detector는 localization과 classification을 수행하는 network 혹은 커포넌트가 분리되어 있습니다. 이는 각 task가 순차적으로 진행되는 것을 의미하며, 이러한 과정에서 병목현상이 발생하여 detection 속도

*: <https://herbwood.tistory.com/13>



[논문 리뷰] You Only Look Once: Unified, Real-Time Object Detection(2016)

오늘은 정말 유명한 Object Detection 모델 Yolo에 대해 리뷰해보도록 하겠습니다. 기존의 object detection 모델들은 detection을 수행하기 위해 classifiers들을 제안했었다. 하지만 이 논문에서 제안하는 yolo는 object detection을 separated bounding boxes와 class 확률에 대한 regression problem으로 정의한다.

*: <https://simonezz.tistory.com/80>



Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it

<https://velog.io/@skhim520/YOLO-v1-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%BO-%EB%BO%8F-%EC%BD%94%EB%93%9C-%EA%B5%AC%ED%98%84>

YOLO의 loss function에 대해

뭔가 이상합니다 | 들어가기 앞서. - 이 글은 먼저 YOLO 및 YOLO v2까지 논문을 보시고 나서 읽어야 합니다. - YOLO 논문에서는 어떻게(How) 작동하는가에 대해서는 열심히 설명하고 있습니다. Grid로 나누어 object의 center 가 떨어지는 cell이 object를 detect하는 방식으로 한다. 이런 것들 말이죠. 그런데 왜(Why) grid로 나누는
↳ <https://brunch.co.kr/@kmbmjn95/35>

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

Joseph Redmon
University of Washington
pjreddie@cs.washington.edu

Santosh Divvala
Allen Institute for Artificial Intelligence
santoshd@allenai.org

Ross Girshick
Facebook AI Research
rbg@fb.com

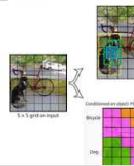
Ali Farhadi
University of Washington
ali1@cs.washington.edu

• 유튜브

YOLO 사람처럼 한번에 파악! | AI 인공지능 이미지 객체탐지 'You Only Look Once: Unified, Real-Time Object Detection' 논문 리뷰

YOLO, 사람처럼 한번에 보고 파악하자! YOLO라고 널리 알려진 'You Only Look Once: Unified, Real-Time Object Detection' 논문 리뷰입니다. 논문 링크: <https://arxiv.org/pdf/1506.02640.pdf> 으로 모델은 처음으로...

▶ https://www.youtube.com/watch?v=lxycUfn_p4Q



십분딥러닝_14_YOLO(You Only Look Once)

YOLO 네트워크 모델에 대한 설명입니다. (계속 10분을 넘기는 불상사가 발생하네요...ㅠㅠㅋㅋ)[PPT]
<https://www.slideshare.net/HyunKyuJeon3/14yoloyou-only-look-once>

▶ <https://www.youtube.com/watch?v=8DjlJc7xH5U>

10-MIN
DEEP-LEARNING

10분 동안 십분(十分) 즐기는 딥러닝

• 깃허브

https://github.com/motokimura/yolo_v1_pytorch