YOLO

1. Introduction

- 딥러닝 기반의 Object Detection System들은 Classification 모델을 Object Detection 시스템에 맞게 변형한 모델들이 주를 이루었음. Classification 모델을 변형한 시스템은 다양한 위치 및 크기에 대해서 학습하고 테스트를 진행
- 대부분의 object detextion 모델들은 sliding window 방법론을 사용하여 전체 이미지를 Classifier로 스캔하는 방식 형태의 방법론을 띄어왔음
- Yolo가 나오기 전의 가장 최근 모델로는 r-cnn이라는 모델이 있음
  R-CNN은 bounding box일 확률이 높은 곳을 제안하고 제안된 box 영역을 Classifier로 넣어 classification을 수행 -> 후처리를 통해서 bounding box를 개선(합치고, 제거하고)하는 방식을 사용
- 위와 같은 r-cnn 모델을은 구조가 매우 복잡함(mask r-cnn이 나오기전)
  region proposal / classification / box regression라는 3가지 단계를 거치는 과정을 가지며, 3가지 단계를 개별적으로 학습해야 하므로 복잡한 파이프라인을 갖게 되고 inference time 또한 많이 걸림
- You Only Look Once(이하 YOLO)는 기존의 방법론인 Classification모델을 변형한 방법론에서 벗어나, Object Detection 문제를 regression문제로 정의하는 것을 통해서 이미지에서 직접 bounding box 좌표와 각 클래스의 확률을 구하는 방법을 통해 기존 문제점을 해결

1. Introduction

- YOLO는 End-to-End 방식의 통합된 구조로 되어 있으며, 이미지를 convolutional neural network에 한 번 평가(inference)하는 것을 통해서 동시에 다수의 bounding box와 class 확률을 구함
- YOLO는 다음과 같은 장점을 가지게 됨
  1) 빠른 속도, 다른 실시간 시스템 대비 2배 이상의 Map
  2) sliding window 방식이 아닌 convolutional neural network를 사용해 전체 이미지를 보게끔 유도되어 (문맥 정보;contextual information)각 class에 대한 표현을 더 잘 학습
  3) 일반화된 Object의 표현을 학습함. 실험적으로 자연의 dataset을 학습시킨 이후에 학습시킨 네트워크에 artwork 이미지를 입력했을 때, DPM, R-CNN 대비 많은 격차로 좋은 Detection 성능을 보여줌

2. Unified Detection

- YOLO에서는 입력 이미지 전체를 네트워크에 통과시켜서 예측한다. 또한 특징 추출, 바운딩 박스 계산, 클래스 분류에 대한 예측이 동시에 이뤄진다.(Unified detection)
  => 좋은 성능을 내면서 end-to-end training 이 가능하게 됨

작동원리

1. input image를 S X S개의 grid(그리드)로 나눈다.
2. 각 grid에서는 B개의 Bounding Box와 Conditional Class Probability(확률)를 예측한다

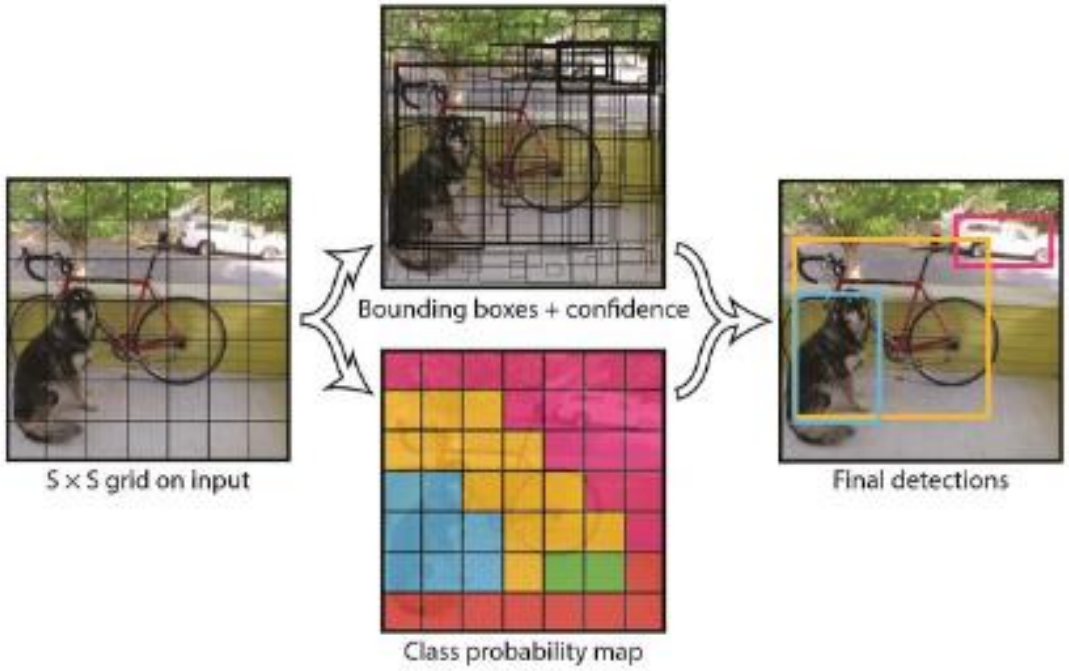
- 각 바운딩박스마다 *confidence scores를 예측함,

  * 바운딩박스내에 물체가 존재할 확률

- Confidence Score는 Bounding Box가 예측하는 5가지x, y, w, h, Confidence) 값들에 의해서 결정됨

  1) (x,y) : Bounding (box의 중심점을 의미하며, grid의 범위에 대한 상대값으로 표현됨

  2) (w,h) : w,h는 Bounding box의 너비와 높이로써 전체 이미지에 대해 상대값으로 표현됨



2. Unified Detection

Confidence score

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

Pr(Object) : Bounding Box 안에 물체가 있을 확률

IOU: 학습데이터의 Bounding Box와 predicted Box의 교집합 / 학습 데이터의 Bounding Box와 predicted Box의 합집합

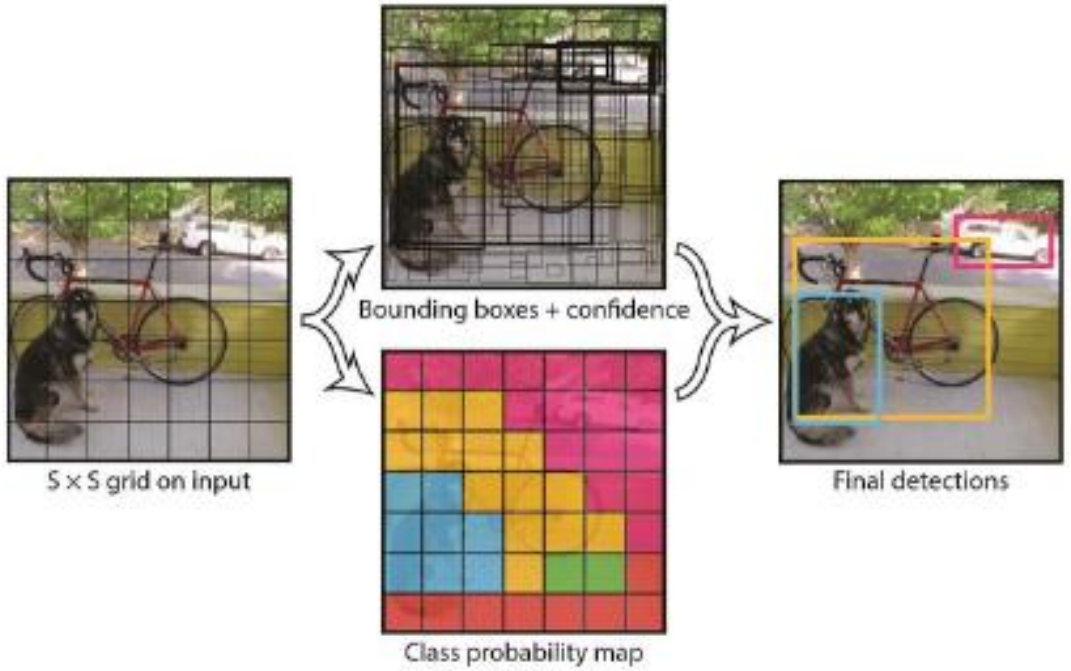
- Conditional Class Probability(확률) 예측 방법

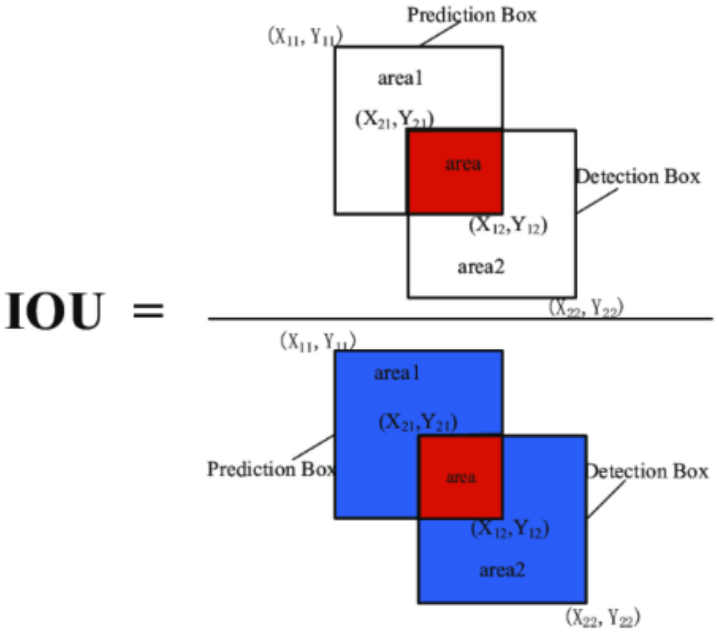
1) Conditional Class Probability(확률)는 분류한 클래스의 확률
2) **Pr(Class_{i}|Object)**로 표현하며 B가 배경이 아닌 객체를 포함하는 경우의 각 클래스의 조건부 확률이다.
3) B가 배경을 예측했다면 확률은 0이 됨. 최종적으로 클래스 조건부 확률 C와 각 바운딩 박스의 Confidence 예측 값을 곱하면 각 박스의 클래스별 Confidence Score 수치를 구할 수 있음

- class-specific confidence score

= confidence score* Conditional Class Probability

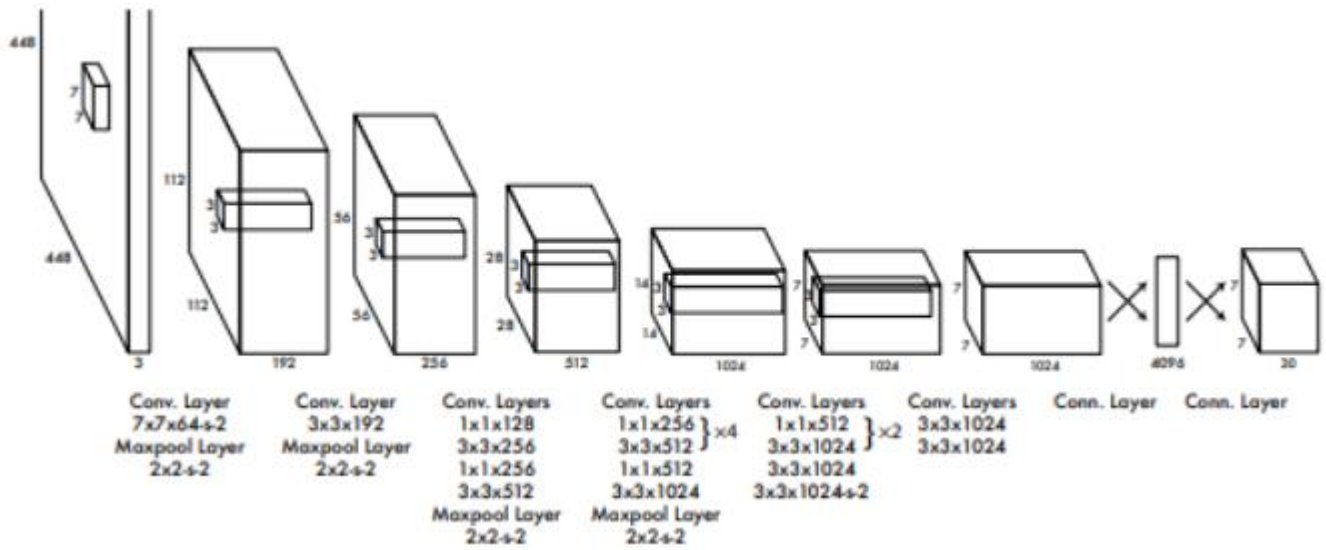
$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth}$$





2.1 Network Design

- YOLO는 24개의 Convolutional Layer(Conv Layer)와 2개의 Fully-Connected Layer(FC Layer)로 연결된 구조를 사용
- 각 layer마다 leaky ReLU를 사용



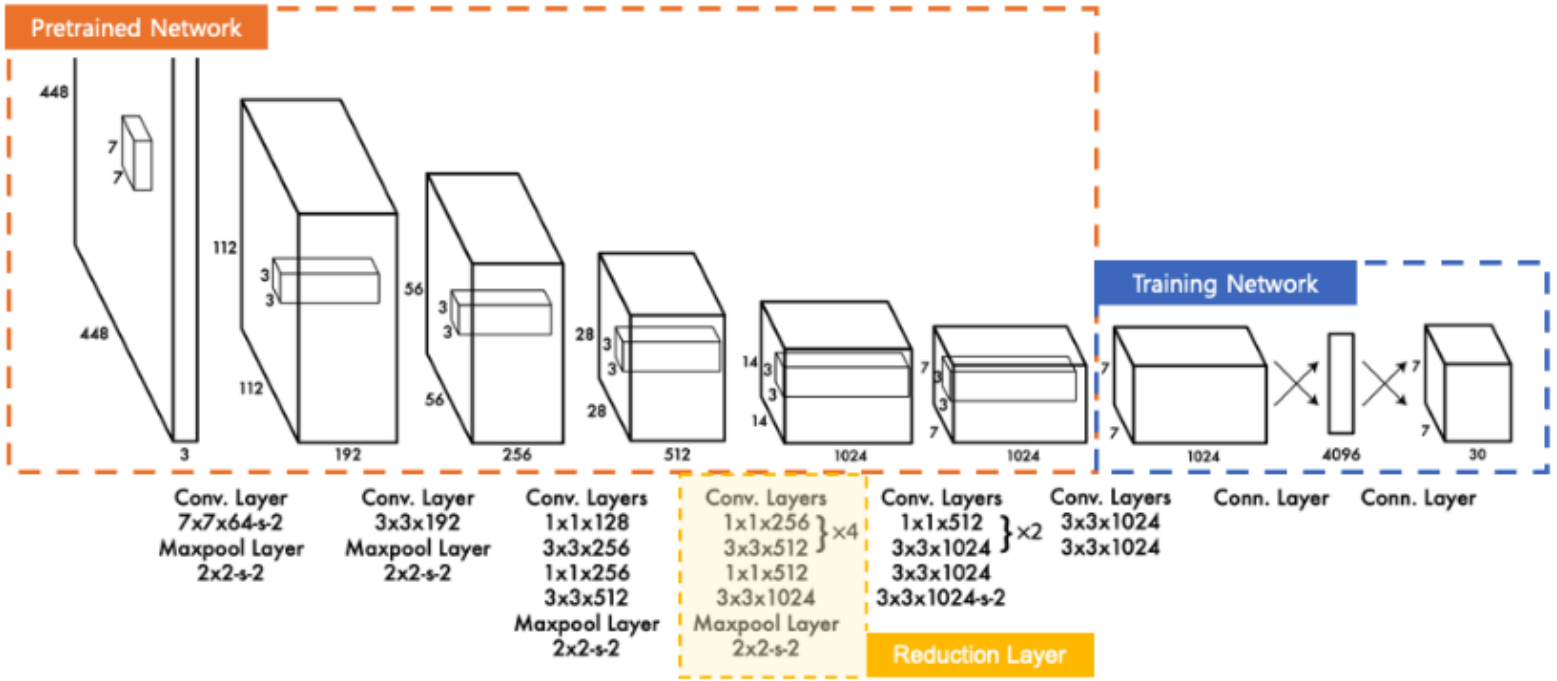
2.02. Training

Pre-trained Network

- 주황색 테두리로 표현한 부분(20개의 convolutional layers)
- GoogLeNet을 이용하여 ImageNet 1000-class dataset을 사전에 학습한 결과를 Fine-Tuning한 네트워크

Training Network

- 파란색 영역
- Pre-trained Network에서 학습한 feature를 이용하여 Class probability와 Bounding box를 학습하고 예측하는 네트워크
- YOLO의 예측 모델은 B(Bounding Box)는 2로 주고 C(분류)는 20로 설정하였기 때문에 7 x 7 x 30 OUTPUT 이 도출됨



2.02. Training

Loss Function

(1) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, x와 y의 loss를 계산
(2) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, w와 h의 loss를 계산
- 큰 box에 대해서는 small deviation을 반영하기 위해 제곱근을 취한 후, sum-squared error를 한다
- 같은 error라도 larger box의 경우 상대적으로 IOU에 영향을 적게 준다
(3) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, confidence score의 loss를 계산
(4) Object가 존재하지 않는 grid cell i의 bounding box j에 대해, confidence score의 loss를 계산
(5) Object가 존재하는 grid cell i에 대해, conditional class probability의 loss 계산

* λ_{coord} : coordinates(x,y,w,h)에 대한 loss와 다른 loss들과의 균형을 위한 balancing parameter
* λ_{noobj} : obj가 있는 box와 없는 box간에 균형을 위한 balancing parameter

- 1^{obj}_{ij} : Object가 존재하는 grid cell i의 predictor bounding box j
- 1^{noobj}_{ij} : Object가 존재하지 않는 grid cell i의 bounding box j
- 1^{obj}_{i} : Object가 존재하는 grid cell I

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

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

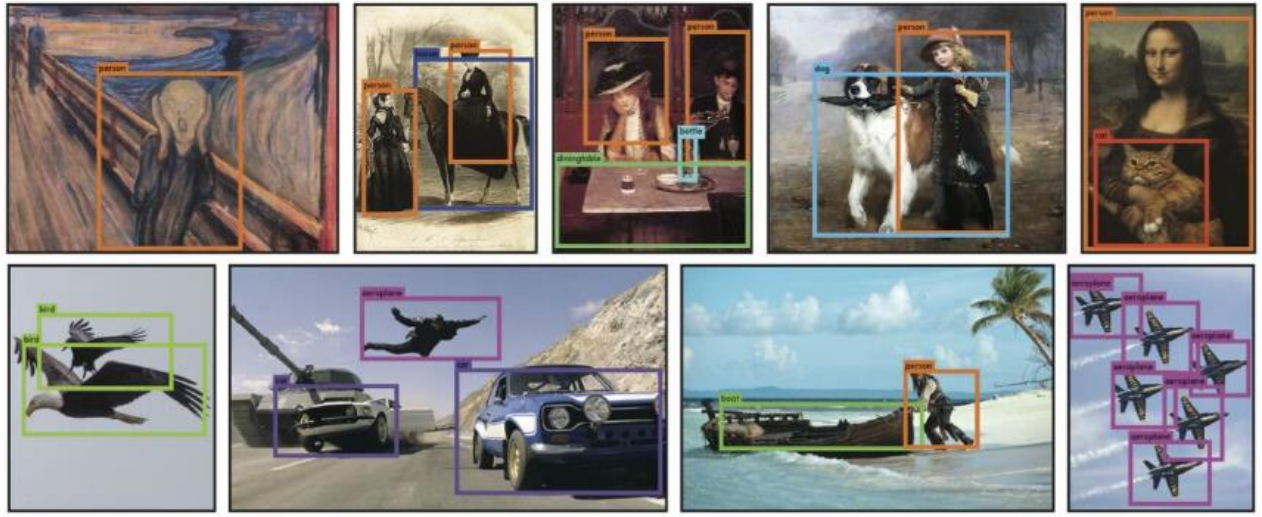
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \quad (3)$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \quad (4)$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

3. Experiment

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



결과

- one stage detector로서 엄청 빠르다. (Real Time Object Detection)
- 다른 도메인에서도 빠르고 나름 괜찮은 성능을 보인다

한계점

- YOLO는 1개의 grid cell당 1개의 class만 취급하기 때문에 2개 이상의 물체들의 중심이 한 grid cell에 모여있더라도 한가지의 class만 예측할 수 있다. 그렇다는 것은 새 떼와 같은 작은 물체들이 모여있을때 감지를 하지 못하게 된다.
- 일정한 비율의 bbox로만 예측을 하다보니 색다른 비율을가진 물체에 대한 예측이 좋지 못하다. -> 일반화가 어려움
- 작은 bbox의 loss와 큰 bbox의 loss를 동일하게 처리한다. -> 큰 상자의 작은 움직임에 비해 작은 상자의 작은 움직임은 훨씬 더 큰 형향을 끼치기 때문