

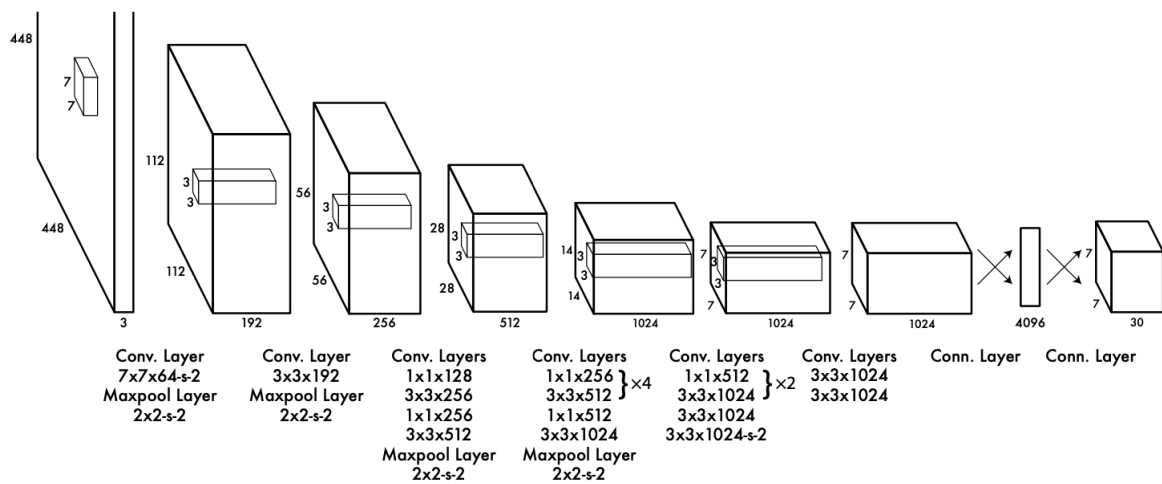


YOLO v1

1. YOLO는 v1부터 v3까지는 주 저자가 Joseph Redmon, Ali Farhadi
2. v4부터는 완전 다른 저자가 만들었다는 특이점이 있습니다. 허나 1-stage로 객체를 감지하는 모델, 처리 속도가 빠른 모델이라는 정신(?)은 공유하고 있습니다.

- 2015년에 나온 논문
- One-Stage 모델의 근본

전체적인 구조



- 448x448 크기의 이미지를 입력받아 여러층의 Layer를 거쳐 이미지에 있는 객체 위치와 객체의 정체를 알아내는 구조입니다.

- 한가지 모델로 object detection을 이뤄낸 것이죠. 그래서 1-Stage detector라고 하는 겁니다.

- Object Detection을 수행하는 모델은 크게 두가지 구조로 나뉩니다.

- Backbone

Backbone은 입력받은 이미지의 특성을 추출하는 역할

- Head

head에서는 특성이 추출된 특성 맵을 받아 object detection 역할을 수행

Backbone

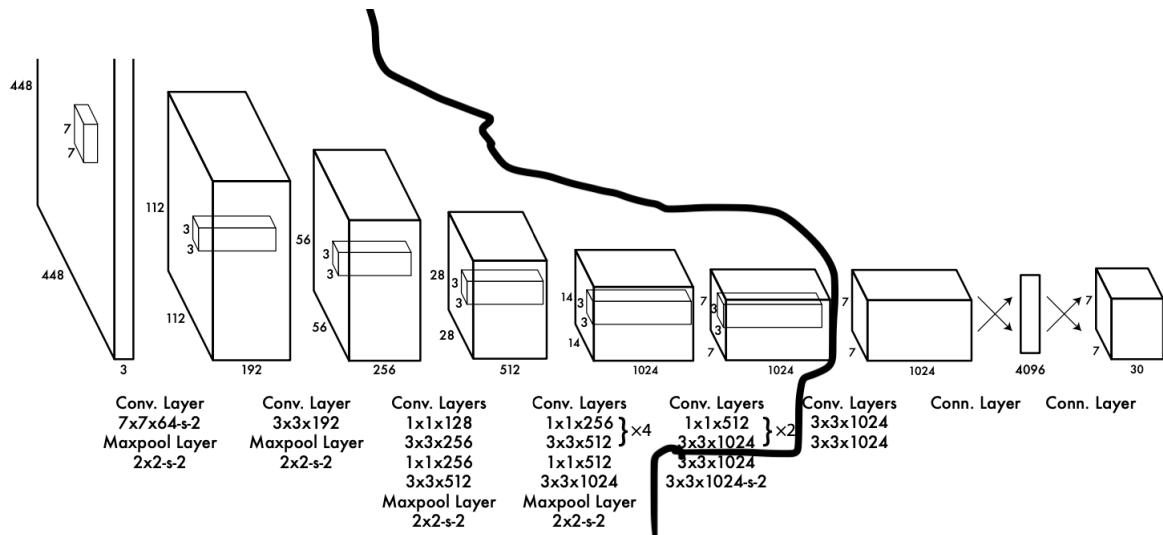
보통 Backbone은 '특성 추출'이 목적이기 때문에 특성 추출에 최적화된 모델, 다시 말해 classification을 목적으로 만들어진 모델을 사용합니다. 왜냐하면 특성된 추출을 가지고 어떤 객체인지 알아내기 때문에 특성을 추출하는데 특화되었기 때문이죠. YOLO가 만들어진 당시 가장 성능이 좋은 classification 모델은 VGG였습니다. 보통 VGG16을 썼죠.

허나 YOLO의 저자들은 VGG16을 쓰지 않고 자신들만의 Backbone을 만들었습니다. 바로 DarkNet이라는 모델입니다.

DarkNet의 특성은 입력받는 이미지의 해상도가 448x448로 VGG가 받던 224x224보다 4배나 더 크다는 것, 그리고 처리 속도가 빠르단 것이었습니다.

입력 해상도가 높은 이유는 Detection이 고해상도 이미지를 종종 요구하기 때문이었다고 논문에 나와있었고 처리 속도는 논문의 Experiment에 실험 결과가 나와있습니다.

참고로, 위 그림에서 DarkNet의 영역만 따로 추려내보면



위와 같습니다. 왼쪽은 DarkNet, 오른쪽은 head입니다. head는 따로 이름이 지어지진 않았 습니다.

그리고 추가로 설명드리자면 DarkNet은 CNN Layer 20층으로 구성되었고 Head는 CNN Layer 4층, FCN 2층으로 구성되었습니다.

Head

Head에 더 설명해 드리도록 하겠습니다. 여기서 주목할 부분은 마지막 출력값의 양식입니 다. 448x448 해상도를 가진 이미지 한 장을 입력 받았을 때 7,7,30 사이즈의 3차원 텐서를 출력값으로 내놓습니다.

즉, 출력값의 셀 하나가 원본 이미지의 64x64영역을 대표하고 이 영역에서 검출된 30개의 데이터가 담겨있다는 뜻입니다. ($448 / 7 = 64$)

30개의 데이터는 다음과 같습니다.

1. 해당 영역을 중심으로 하는 객체의 Bounding Box 2개(x, y, w, h, confidence)
2. 해당 영역을 중심으로 하는 객체의 class score 20개

한 셀에서 2개의 bounding box를 검출하니까 총 검출되는 박스는 $7 \times 7 \times 2 = 98$ 개입니다. 이 98개의 박스는 각각 confidence를 가지고 있는데요, 말 그대로 신뢰도입니다. 이 bounding box를 얼마나 신뢰할 수 있느냐? 를 나타낸 점수라 보시면 됩니다. 수학적으로 나타내면 $\text{Pr}(\text{Object}) * \text{IoU}$ 입니다.

그리고 x, y는 해당 셀에 대해 normalize된 값이고 w,h는 전체 이미지에 대해 normalize된 값입니다. 예를 들어 (0,0)셀에서 나온 bounding box의 [x,y,w,h]가 [0.5, 0.5, 0.2, 0.2]면 변 환했을 때 $x = 31, y = 31, w = 448 \times 0.2 = 96, h = 96$ 이란 것이죠. (0,0) 셀은 원본 이미지의 (0,0)<->(63, 63)인 사각형을 대표하기 때문입니다.

20개의 class score는 해당 영역에서 검출된 객체가 어떤 클래스의 객체일 확률을 클래스 별로 나타낸 겁니다. 20은 이제 YOLO를 훈련시킬 때 사용할 PASCAL VOC 2007 dataset에 있는 클래스가 20종류라 20을 사용한 겁니다. 만약 다른 데이터셋을 사용한다면 20 말고 다른 숫자를 사용할 수 있겠죠.

활성화 함수

저자는 활성화 함수도 다른걸 사용합니다. 저자는 Linear Activation function과

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

위와 같은 식을 가진 Leaky ReLU를 사용했습니다. Linear Activation function는 맨 마지막 Layer에 사용했는데요, 찾아보니 받은 값을 그대로 출력하는 함수였습니다. 음...그냥 사용하지 않았다! 와 동의어로 보시면 되겠습니다.

Leaky ReLU는 마지막 Layer를 제외한 모든 레이어에서 사용했습니다.

Loss function

손실 함수는 multi-task loss를 사용합니다.

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

이렇게 생겼는데요, 위에 두줄은 bbox의 위치에 관한 손실(localization loss), 중간 3, 4 번째 줄은 confidence score에 관한 손실(confidence loss), 마지막 한 줄은 class score에 관한 에러입니다.(classification loss)

하나씩 뜯어보면 간단한 식들입니다. 여기 시그마에서 S^2 는 전체 cell의 개수 = 49고 B는 각 셀에서 출력하는 bounding box의 개수 = 2입니다.

즉, localization loss, confidence loss는 **해당 셀에 실제 객체의 중점이 있을 때 해당 셀에서 출력한 2개의 bounding box 중 Ground Truth Box와 IoU가 더높은 bounding box와 Ground Truth Box와의 loss를 계산한 것들입니다.**

그리고 classification loss는 **해당 셀에 실제 객체의 중점이 있을 때 해당 셀에서 얻은 class score와 label data 사이의 loss를 나타낸 값**이죠.

후에 보여드릴 구현 코드를 보시면 더 쉽게 이해하실 수 있을겁니다.

Experiment : Backbone에 따른 성능 변화

저자는 Experiment 부분에서 Backbone에 따른 성능 변화를 나타냈습니다.

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

위 그림은 Detector들의 성능을 나타낸 겁니다. 여기서 YOLO가 성능, 처리속도를 같이 고려했을 때 가장 성능이 좋은 model로 나타난걸 알 수 있죠.

이 중 YOLO와 YOLO VGG-16이 보이실 겁니다. VGG-16은 backbone을 DarkNet대신 VGG-16을 사용한 모델인데요, 처리속도는 많이 떨어졌지만 정확도(mAP)는 좋아진 걸 확인하실 수 있습니다.

저는 원래 DarkNet이 있는 original YOLO를 구현하려고 했습니다. 그런데 텐서플로우2.x로 DarkNet이 구현 되어있는 걸 찾을 수가 없어 포기하고 VGG16으로 구현했습니다.

훈련


훈련 방법 등은 다음과 같습니다.

1. Backbone : ImageNet 2012 dataset으로 1주일간 훈련
2. Head : Weight decay = 0.0005, momentum = 0.9, batch size = 65, epoch = 135로 설정. 처음에는 learning rate를 0.001로 맞춘 뒤 epoch = 75까지 0.01로 조금씩 상승시킴. 그리고 이후 30회는 0.001로 훈련시키고 마지막 30회는 0.0001로 훈련시킴.
3. 데이터 증강(data augmentation) : 전체 이미지 사이즈의 20%만큼 random scaling을 합니다. 그리고 translation도 하며 원본 이미지의 1.5배만큼 HSV를 증가시킵니다.

출저:

[논문리뷰] YOLO v1 리뷰 + 코드 구현(TensorFlow2)

안녕하세요. 밍기뉴와제제입니다. 오늘은 YOLO v1에 관한 리뷰를 하고 코드 구현한걸 설명해 드리고자 합니다. YOLO는 1-stage detector를 대표하는 모델 중 하나입니다. 현재 v5까지 나온걸로 압니

 <https://velog.io/@minkyu4506/YOLO-v1-%EB%A6%AC%E B%B7%B0-%EC%BD%94%EB%93%9C-%EA%B5%AC%ED% 98%84tensorflow2#%EA%B5%AC%ED%98%84-%ED%9B%8 4%EA%B8%B0>

