PT

1. Introduction

2. Faster R-CNN

3. DETR

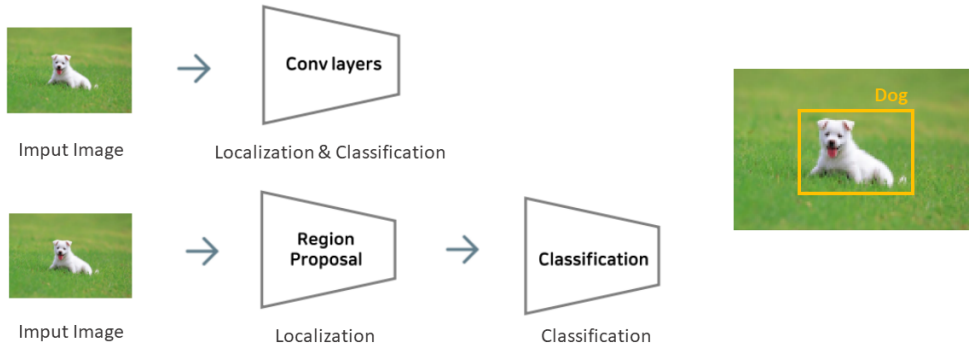
4. Experiments

DELAB 2

Object Detection에 대한 세미나를 준비하였다. DETR을 리뷰하기 전에 Faster R-CNN을 리뷰하는 이유는 DETR논문에서 Faster R-CNN 모델과 비교하는 내용이 있고, Faster R-CNN의 핵심인 RPN 모델이 객체 검출 task에서 큰 획을 그었기 때문에 한 번 알아보고 넘어가려고 한다. 이후 DETR 과 구조적인 비교를 하고 Experiments에서 성능 비교까지 해보겠다.

Introduction

- 객체 검출 (Object Detection)
  - 입력 영상에서 사용자가 원하는 객체를 찾고 바운딩 박스와 클래스를 구하는 것
  - one_stage_method: 바운딩 박스와 클래스를 한 번에 찾음 (속도는 빠르지만 정확도가 낮음)
  - two_stage_method: 바운딩 박스와 클래스를 순차적으로 찾음 (정확도는 높지만 속도가 느림)

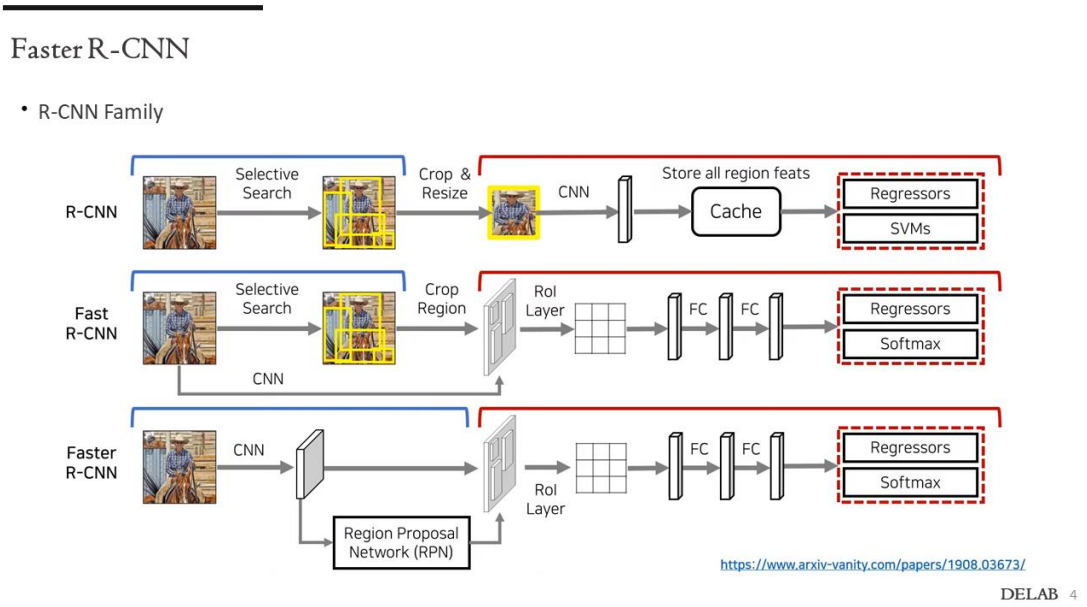


DELAB 3

먼저 객체 검출은 '입력 영상에서 사용자가 원하는 객체를 찾고 바운딩 박스와 클래스를 구하는 것'이다. 이 task는 크게 one stage method와 two stage method가 존재한다. one stage method는 CNN을 한 번만 통과시킴으로써 객체의 위치가 담긴 바운딩 박스를 찾아내고(localization) 해당 바운딩 박스의 클래스를 구한다.(classification) two stage method는 Region Proposal Network를 거

쳐서 바운딩 박스를 미리 찾고 classification layer를 거쳐서 바운딩 박스에 대한 클래스를 찾는다. 첫 번째 방법은 속도는 빠르지만 정확성이 떨어진다는 단점이 있고 두 번째 방법은 속도는 느리지만 정확성은 높다는 장점이 있다. 지금부터 소개할 R-CNN 패밀리는 Two Stage 방법이다.

p.s. classification에서는 후보 박스의 모든 class score가 0이라면(객체가 없다면) 해당 박스를 제거한다. 또한 2-stage-method의 localization을 진행할 때도 classification을 진행한다. (객체가 있는지 없는지 정도만 파악) 즉 1-stage-mehod는 후보 박스를 생성하고 제거하는 과정을 한 번 거치지만 2-stage-method는 후보 박스를 생성하고 제거하는 과정을 두 번 거치므로 더 정확한 박스가 나오는 것이다.



R-CNN은 이미지에 대해 CPU상에서 Selective Search를 진행한다. 그러면 물체가 존재할 법한 위치(박스) 약 2000개를 찾게 되고 그 2000개의 물체를 개별적으로 CNN에 넣어서 Feature를 추출한다. 이후 SVM으로 Classification을 하고 Regressor을 학습해서 박스 위치를 조정한다.

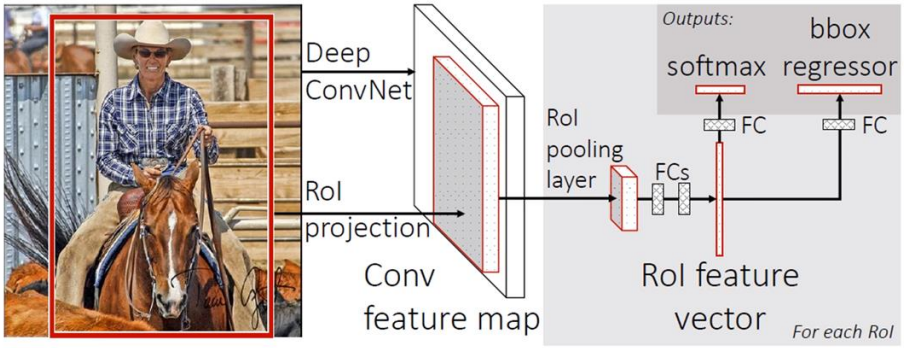
Fast R-CNN도 마찬가지로 Selective Search를 진행한다. 그러나 Feature Map을 뽑기 위해서 CNN을 한 번만 거친다. 즉 2000번 CNN을 통과시키는 것이 아니라 원본 이미지로 한 번만 통과시킨다. 사실 CNN 구조를 생각해보면 Feature Map은 Input Image에 대해서 각각의 위치에 대한 정보를 어느정도 보존하고 있기 때문에 이러한 작업이 가능하다. 이후 Region들을 하나씩 ROI Pooling에 넣어서 Max Pooling을 하고 Softmax를 통해 class에 대한 Probability를 구한다.

Faster R-CNN은 GPU상에서 Region Proposal을 할 수 있다. 이것은 RPN이라는 네트워크에서 진행한다. RPN은 Feature Map을 보고 어느 곳에 물체가 있을 법한지 예측을 하는 네트워크라고 생

각하 면 된다. 기존의 Selective Search보다 훨씬 시간적인 장점이 있다. 이후에는 Fast R-CNN의 구조를 따른다

Faster R-CNN

- Fast R-CNN
  - Feature Extraction, RoI Pooling, Classification, Regression 단계를 End-to-End로 묶어서 학습할 수 있는 모델



DELAB 5

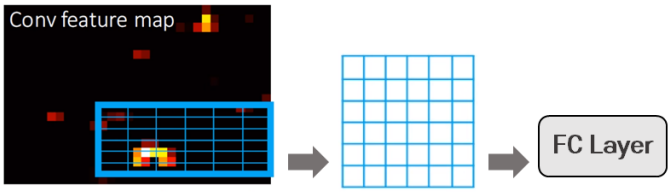
Fast R-CNN부터 좀 더 자세히 설명하겠다. 이전 모델인 R-CNN은 Feature Extractor인 AlexNet을 학습하고 박스 위치를 조정하는 Regressor도 학습하고 박스 분류를 하는 SVM도 학습을 하였다. 그러나 Fast R-CNN부터는 Selective Search이후의 모든 과정(특징 찾고 분류하고 위치 찾는 과정)을 End-to-End로 묶어서 학습할 수 있다.

또한 속도 향상을 시킨 주된 이유는 CNN을 한 번만 통과시킨다는 점인데, 이것이 가능한 이유는 Feature Map이 물체의 위치를 보존하고 있다보니 Selective Search를 통해 나온 RoI를 Feature Map에 projection시킬 수 있기 때문이다. 따라서 한 feature map에 모든 roi를 projection하고 작업하면 되니까 속도 향상이 일어난 것이다.

이후 RoI Pooling이라는 작업을 해서 나온 Feature들이 FC Layer를 통과해 클래스 분류 및 좌표 예측을 할 수 있다.

Faster R-CNN

- RoI Pooling
  - RoI 영역에서 Max Pooling을 진행함으로써 모든 RoI를 같은 크기의 Feature Map으로 변환하는 것



RoI in Conv feature map : 21x14 -> 3x2 max pooling with stride(3,2) -> output : 7x7

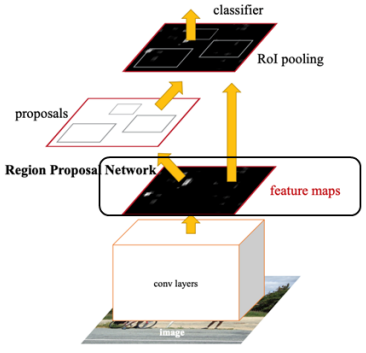
RoI in Conv feature map : 35x42 -> 5x6 max pooling with stride(5,6) -> output : 7x7

사실 Fast R-CNN에서 가장 중요한 파트는 RoI Pooling인데 이것은 쉽게 말하면 RoI에 대해서 Max Pooling을 하겠다는 것이다. RoI Pooling을 하는 이유는 다음에 나오는 fc layer의 입력과 크기를 맞춰주는 작업이 필요하기 때문이다.

만약 FC Layer의 입력이 49라고 하면 RoI Feature Map의 크기는 7x7이 돼야 한다. 따라서 Feature Map에 있는 RoI의 크기에 따라 필터와 stride를 바꿔가면서 Max Pooling을 하는 것이다.

Faster R-CNN

- Faster R-CNN (RPN + Fast R-CNN)
  - RPN을 제안하여 전체 프레임워크를 End-to-End로 학습할 수 있는 모델
  - RPN: Region Proposal Network (GPU 장치에서 수행함)



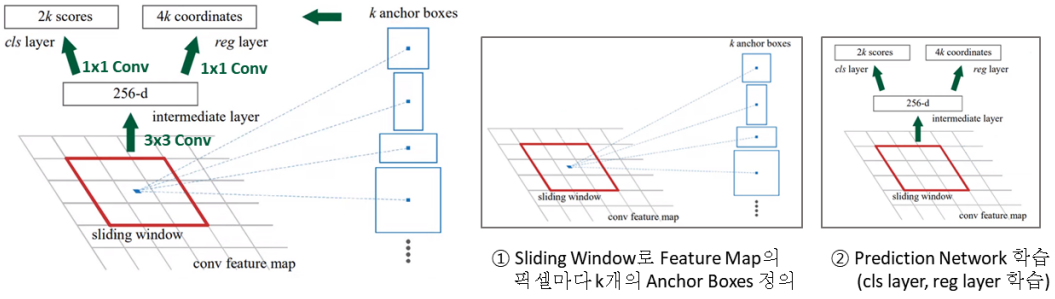
Fast R-CNN의 Conv feature map이
Region Proposal에도 쓰일 수 있다!

다음은 Faster R-CNN에 대한 소개이다. 이 모델은 RPN을 제안하여 전체 프레임워크를 End-to-End로 학습할 수 있는 모델이다. 이것이 가능한 이유는 Region을 GPU 연산으로 뽑기 때문에 가

능하다고 한다. 모델은 위와 같이 생겼는데, fast r-cnn의 feature map이 rpn 네트워크워크에서도 쓰이는 구조가 된다.

Faster R-CNN

- RPN
  - Fast R-CNN의 Feature Map과 k개의 Anchor Box들을 이용해서 RoI(Region of Interest)를 추출
  - ① Anchor를 정의하고 ② Prediction Network를 학습하는 역할



① Sliding Window로 Feature Map의 픽셀마다 k개의 Anchor Boxes 정의

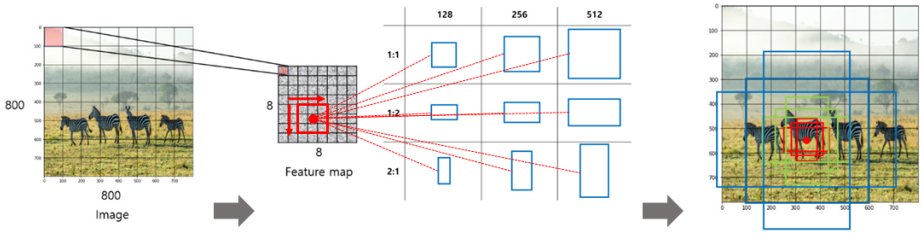
② Prediction Network 학습 (cls layer, reg layer 학습)

RPN은 Anchor Box라는 개념을 이용해서 RoI를 추출하는 역할을 한다. 왼쪽 그림을 보시면 3x3 필터로 feature map에서 슬라이딩 윈도우를 하게 되는데 이 과정에서 k개의 anchor box를 정의를 하는 개념이다.

또한 Prediction Network도 학습이 되는데 이 두 번째 과정을 함으로써 Predicted BBox에 물체가 있는지 없는지 확인하고 Predicted Box와 GT Bbox의 offset 차이를 최소화할 수 있다.

Faster R-CNN

- RPN
  - ① Sliding Window로 Feature Map의 픽셀마다 k개의 Anchor Boxes 정의
  - Anchor Box: scale([128, 256, 512]), aspect ratio([1:1, 1:2, 2:1]) -> k=9



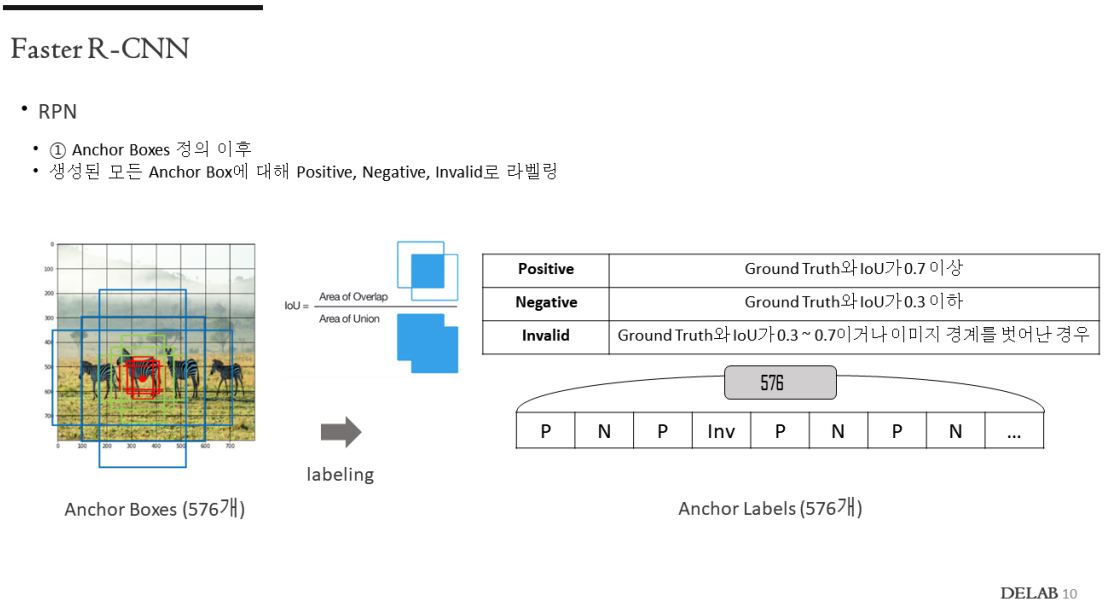
Feature Extraction
(800x800x3) -> (8x8x512)

Sliding Window
(8x8x9)개의 Anchor Box 정의

첫 번째 과정부터 그림으로 살펴보겠다. 예를 들어 input image가 800x800x3일 때, vggnet을 거쳐서 8x8x512 크기인 feature map이 나온다고 가정하자.

이 feature map은 슬라이딩 윈도우되면서 각 픽셀마다 k개의 anchor box가 정의된다.

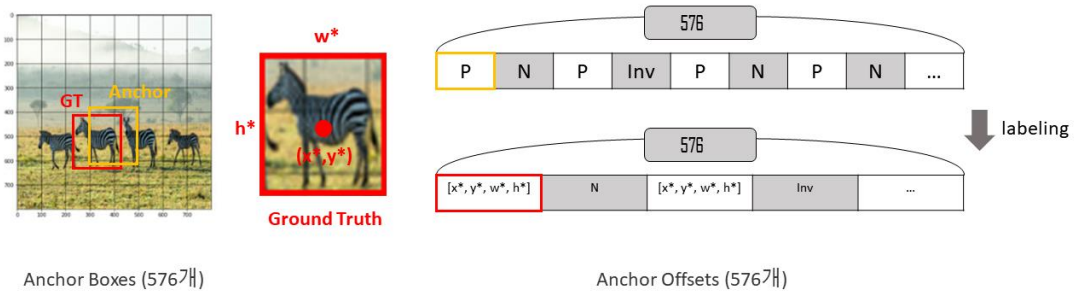
논문에서 Anchor Box의 scale은 128,256,512 세 개가 있고 aspect ratio는 1:1, 1:2. 2:1 세 개가 있으므로 총 9개가 된다. 따라서 슬라이딩 윈도우가 끝나면 8x8x9개 만큼의 Anchor Box가 생성된다.



Anchor box가 정의된 이후에는 생성된 모든 anchor box에 대해 라벨링을 진행한다. 이 때 IoU가 0.7 이상이면 Positive, 0.3이하이면 Negative, 그 외 나머진 경우는 Invalid라고 선정해서 크기가 576인 Anchor Label 벡터를 생성한다.

Faster R-CNN

- RPN
  - ① Anchor Labels 정의 이후
  - Positive Anchor Box에 한해, 해당하는 Ground Truth의 Bouding Box Offset (x*,y*,h*,w*)으로 라벨링

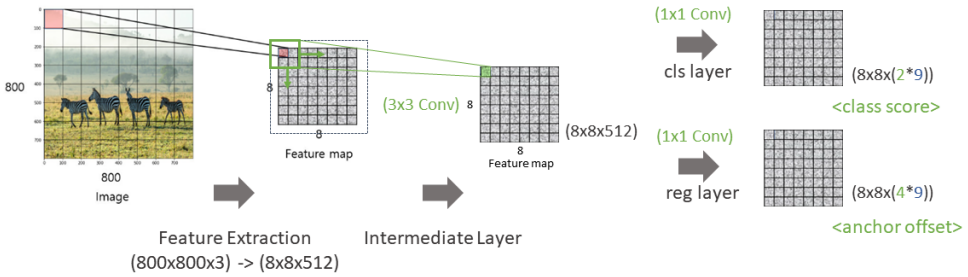


또한 Positive Anchor Box에 한해 해당하는 GT Bbox의 offset으로 라벨링을 진행해서 Anchor Offset 벡터를 생성한다.

이렇게 Anchor Labels와 Anchor Offset을 생성하면서 RPN의 Prediction Network를 학습할 때 필요한 데이터를 생성하는 것이다.

Faster R-CNN

- RPN
  - ② Prediction Network를 학습
  - 3x3 Conv를 거친 Feature Map에 1x1 Conv를 두 가지 목적(cls, reg)를 위해 적용



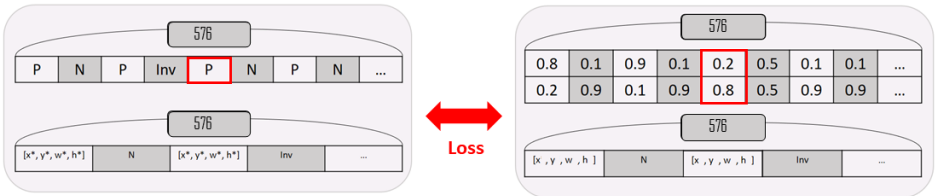
다음은 Prediction Network를 학습하는 내용이다.

먼저 Feature Map에 1by1 패딩을 주고 3x3 conv를 하면서 feature를 강화한다. 이후 1x1 conv를 통해 채널을 압축시킴으로써 class score가 담긴 3차원 텐서와 anchor offset이 담긴 3차원 텐서를

생성한다. 전자의 경우, 576개의 anchor box에 대해 object가 있을 확률과 없을 확률이 저장되고 후자의 경우, 576개의 anchor box의 중심 좌표 x,y와 width, height값이 저장된다.

Faster R-CNN

- RPN
  - ② Prediction Network를 학습
  - Anchor Box로 생성한 벡터를 이용해 RPN 학습



Anchor Label Vector
Anchor Offset Vector

Predicted Label Vector
Predicted Offset Vector

DELAB 13

앞에서 언급한 3차원 텐서들은 오른쪽 그림과 같이 flatten하게 펼칠 수 있는데, 1번 과정에서 생성한 정답 데이터와 Loss를 구하면서 RPN이 학습된다고 생각하면 된다.

Faster R-CNN

- RPN Loss
  - Classification Loss와 Regression Loss로 구성됨
  - Regression loss는 p_i^*이 1일 때만 동작 -> anchor box에 object가 없으면 regressor는 학습되지 않음
  - Normalization과 λ는 별로 중요하지 않음 (비율만 비슷하게 맞추기)

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

- i : mini-batch 내의 anchor의 index
- p_i : anchor i에 객체가 포함되어 있을 예측 확률
- p_i^* : anchor i에 객체가 있으면 1, 없으면 0
- t_i : 예측 bounding box의 좌표
- t_i^* : GT bounding box의 좌표
- L_{cls} : Log loss
- L_{reg} : Smooth L1 loss
- N_{cls} : mini-batch의 크기 (256)
- N_{reg} : anchor location의 수
- λ : balancing parameter (10)

DELAB 14

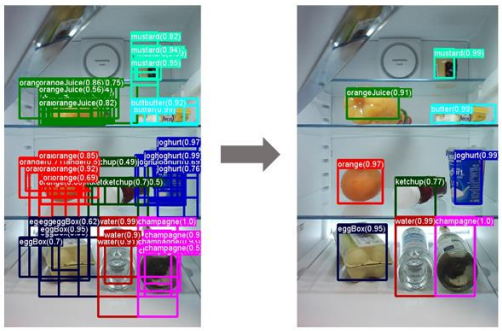
학습에 사용되는 Loss는 기본적인 classification loss와 regression loss로 구성이 된다.

Classification loss는 log loss가 사용이 되었다고 하고 regression loss는 smooth l1 loss가 사용이 되었다고 한다. 이 때 p_i와 p_i^*이 사용이 되는데, 예를 들어 앞장을 보면 빨간 색 박스친 부분

에서 p_i^*은 1이 되고 p_i는 0.2가 되므로 Loss가 크다. 따라서 학습을 하면서 0.8, 0.9 따위가 될 것이다. regression loss는 p_i^*이 1일 때만 동작하는 특징이 있어 anchor box가 object가 존재해야 학습이 된다. 또한 normalization term이 각 항에 붙었는 데 두 loss의 비율만 비슷하면 큰 상관이 없다고 한다.

Faster R-CNN

- NMS
  - class score에 따라 상위 N개의 Region Proposal만을 추출함 (576 -> N)
  - Non Maximum Suppression을 적용하여 최적의 Region Proposal만을 Fast R-CNN에 전달함 (N->N')
  - NMS : 여러 개의 bounding box가 겹쳐 있는 경우에 겹친 box들 일부를 제거하는 방법



DELAB 15

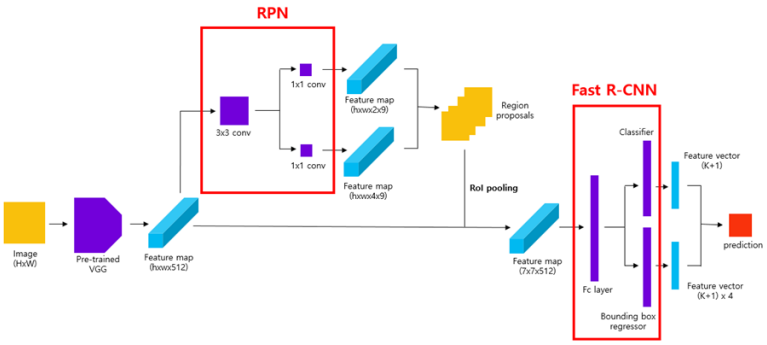
RPN이 Region을 뽑은 이후에는 class score에 따라 상위 n개의 proposal만 추출한다.

이후 nms알고리즘을 적용해서 최적의 proposal만을 다시 추출해서 fast rcnn 파트로 넘긴다고 한다. Nms는 지난 yolo 세미나에서도 소개했지만 여러 개의 bounding box가 겹쳐있는 경우에 겹친 box를 일부 제거하는 방법이다.

Faster R-CNN

- Training

1. VGGNet과 RPN을 Pre-trained 모델로부터 학습한다
2. VGGNet과 Fast R-CNN 모델을 학습한다 (Region Proposals 이용)
3. VGGNet은 고정하고 RPN의 학습만 진행한다
4. VGGNet은 고정하고 Fast R-CNN의 학습만 진행한다.

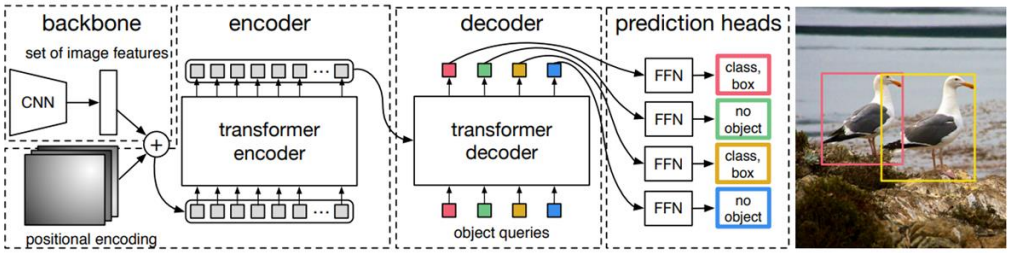


이후 Fast R-CNN과 함께 학습을 진행할 때는 4 step alternative training을 함으로써 rpn과 fast rcnn이 둘 다 학습이 되도록 만들었다고 한다. 과정은 설명(1~4)와 같다.

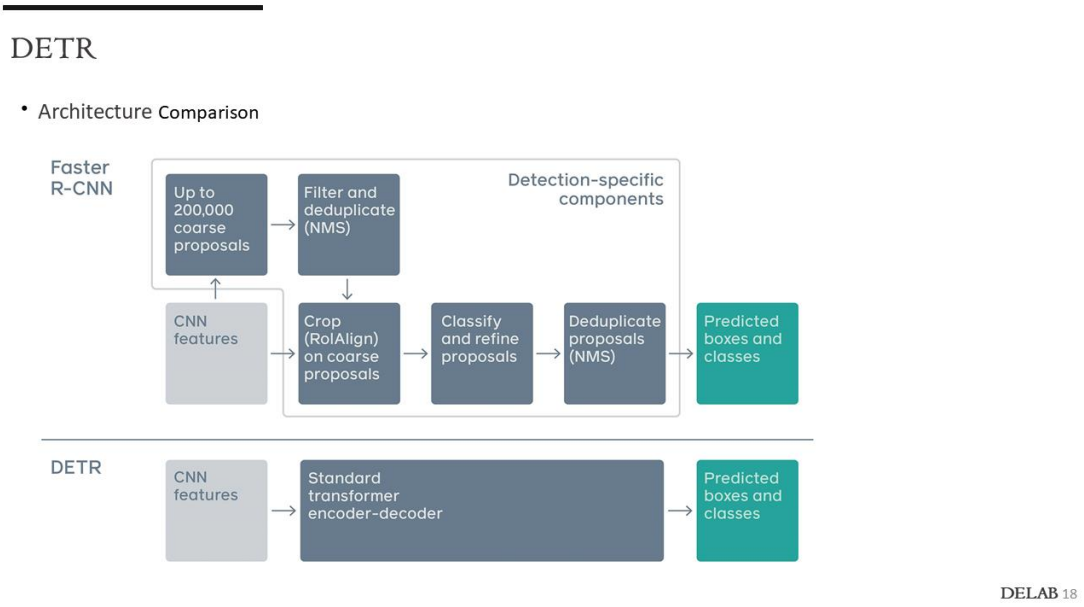
p.s. (page.7)에서 faster r-cnn은 end-to-end모델이라고 했는데 왜 나눠서 학습하는지 의아할 수 있다. end-to-end란 하나의 loss function을 가지고 전체 모델을 한 번에 학습(backpropagation)하는 것을 말한다. 사실 faster r-cnn은 위처럼 두 모델(rpn, fast r-cnn)을 나눠서 학습할 수도 있지만 두 loss(rpn loss, fast r-cnn loss)를 joint해서 같이 학습할 수도 있다. 따라서 end-to-end학습이 가능하다. 그러나 그렇게 하면 나눠서 학습했을 때보다 더 성능이 안 나왔다고 한다.

DETR

- DEtection TRansformer
  - Transformer 구조를 이용한 객체 검출 모델
  - ① Detection용 Transformer 제안 ② 이분 매칭 손실 함수 제안
  - RPN 모델(Region Proposal)과 NMS 알고리즘(Post-Processing) 없이 Faster R-CNN 이상의 성능

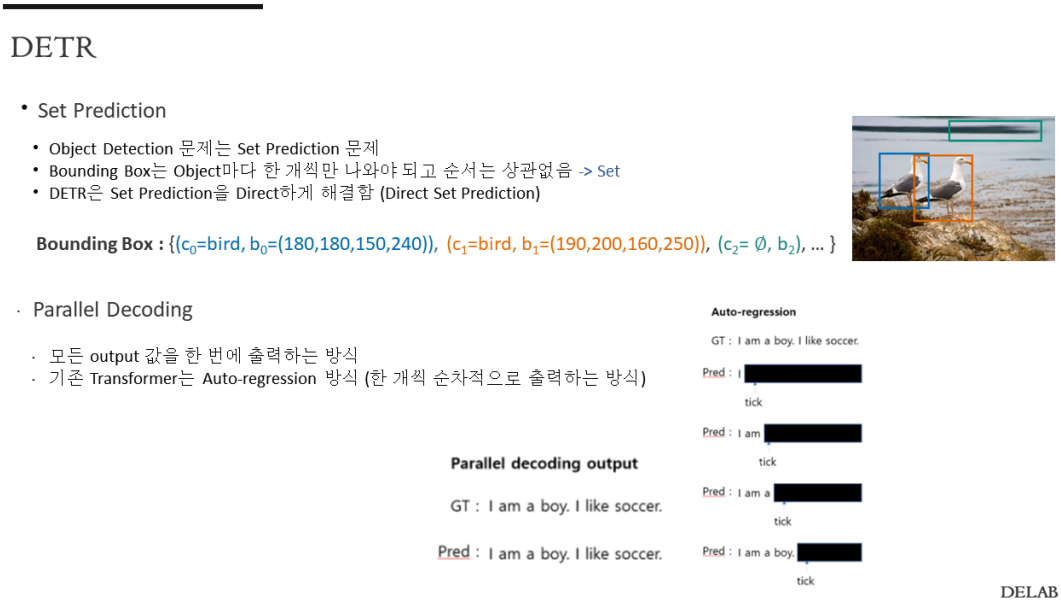


다음으로 소개할 모델은 Detection Transformer이다. 이름 그대로 Transformer 구조를 이용한 객체 검출 모델이다. 이 논문에서는 크게 Transformer와 이분매칭 손실함수를 소개하였다. 사실 이 논문이 뜨게 된 이유는 Detection Task에 자주 쓰이는 RPN과 NMS도 없는데 준수한 성능을 보였기 때문이다. 아주 간단한 구조의 Detection 모델이라고 생각할 수 있다.



얼마나 간단한지 Faster R-CNN과 아키텍쳐를 비교해보겠다. Faster RCNN은 RPN을 통해 최대 20만개의 region을 추출하고 nms를 통해 region을 선별한다. 이후 fast rcnn구조를 통해 box가 한번 더 선정된다. 마지막으로 다시 nms를 통해 오브젝트와 겹치는 박스를 제거하는 과정이 전체 구조이다.

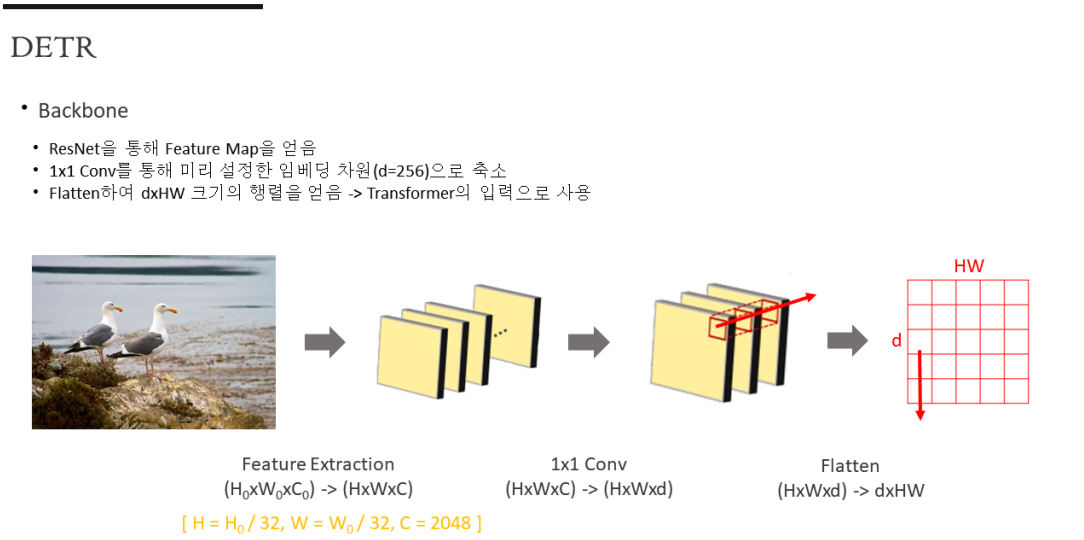
그런데 DETR은 Transformer 구조를 통과하면 바운딩 박스와 클래스 정보가 나타난다.



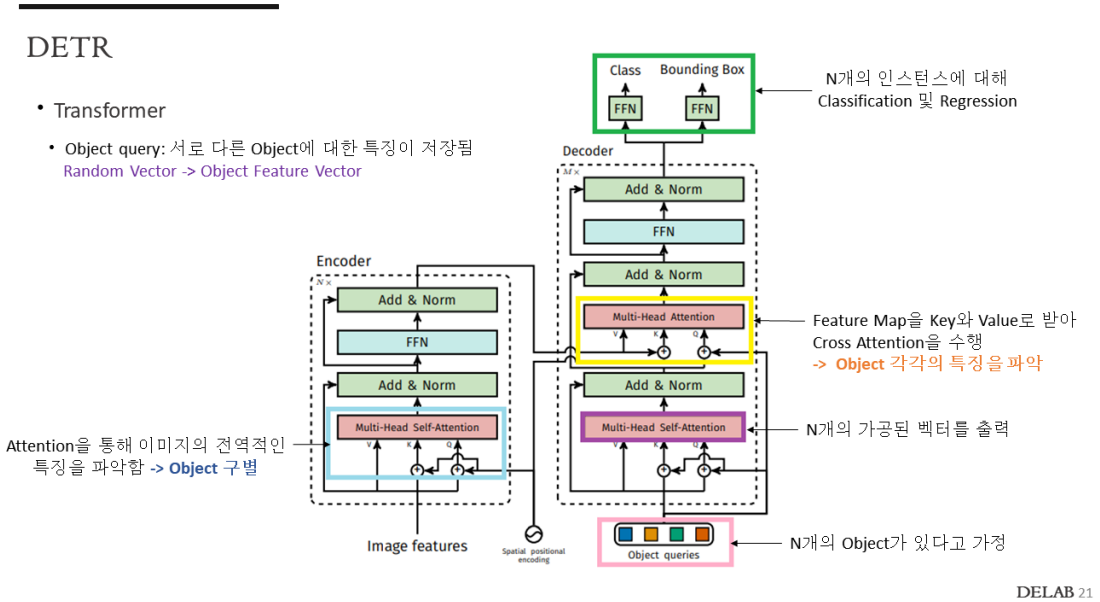
이 논문의 related work에는 set prediction과 parallel decoding에 대해 소개한다. 먼저 이 논문의 저자는 object detection 문제는 set prediction 문제와 다를 바가 없다고 말한다. 왜냐하면 최종적인 bounding box는 object마다 한 개씩만 나와야 되고 순서는 상관이 없으니 set의 성질을 따르기 때문이다. 그런데 object마다 한 개의 box만 나오려면 nms나 window penalty같은 post processing이 필요하였는데 detr은 후처리없이 할 수 있다고 언급한다.

두 번째는 parallel decoding이다. Attention is all you need 논문에서 제안한 트랜스포머 구조는 auto-regression 방식을 따른다. 이건 output을 한 개씩 순차적으로 출력하는 방식을 의미하는데, Transformer를 상기시켜보면, GT가 Decoder의 입력이라고 할 때 masked attention을 통해 뒷 부분의 시퀀스를 가리면서 바로 뒤에 나오는 시퀀스가 무엇인지 찾는 역할을 하였다.

하지만 detr은 parallel decoding방식이다. 즉 모든 output값이 한 번에 출력된다고 생각하면 된다.



지금부터는 아키텍쳐에 대해 소개하겠다. 먼저 백본 네트워크이다. 이미지가 입력으로 주어지면 resnet을 통해 [h/32 x w/32 x 2048]인 feature map을 얻어낸다. 이후 1x1 conv를 통해 미리 설정한 임베딩 차원으로 축소를 한다. 여기서 임베딩이라고 하는 이유는 트랜스포머 인코더의 입력으로 사용될 때는 임베딩 과정을 거치기 때문이다. (detr transformer은 임베딩하는 모델이 없으므로 여기서 진행한다) 이후 flatten을 해서 dxHW인 행렬을 만들고 이 행렬이 트랜스포머의 입력으로 들어간다.



다음은 DETR의 Transformer 구조이다. 제일 특이한 점은 object query라는 입력이 decoder에 사용이 되었다는 점이다. 이것은 서로 다른 object에 대한 특징이 저장된 벡터라고 생각하면 된다.

예를 들어 Decoder에 4개의 obect query가 있으면 네 오브젝트에 대한 특징이 출력된다.
트랜스포머 구조를 살펴보면, 먼저 dxHW 행렬이 Encoder에 들어가서 Attention 연산을 한다. 이때 나오는 self attention map은 각 object에 대한 특징이 담겨 있는데, 결국 Encoder는 Object를 구별하면서 특징을 찾았다는 것이다. 이러한 특징은 디코더의 사이드 입력으로 들어간다.

디코더는 N개의 Object가 있다고 가정해서 n개의 object query가 들어가는데, 처음에는 그냥 n개의 가공된 벡터를 출력한다. 그러나 두 번째 Attention부터는 Feature Map을 key와 value로 쓰는 cross attention을 수행하면서 object 각각에 대한 특징을 파악할 수 있게 된다.

이렇게 Encoder에서 Object를 구별하고 Decoder에서 n개의 object에 대해 추가적으로 특징을 파악하면 초록색 영역에서 바운딩 박스를 찾게 될 때 많은 도움이 된다고 한다.

그리고 Positional Encoding을 할 때는 key와 query에만 해주었다고 하는데, 이렇게 했을 때 더 성능이 잘 나왔기 때문이라고 한다.

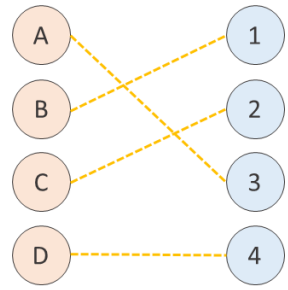
DETR

- Bipartite Matching
  - 예측 BBOX와 GT BBOX 사이에 중복이 없는 일대일 매칭을 함
  - DETR은 헝가리안 알고리즘을 활용한 이분 매칭을 진행

Q. 네 명의 사람(A~D)이 일(1~4)을 할당 받아야 한다. 표의 수치는 일의 능률이라고 할 때 어떤 할당이 가장 최선인가?

	A	B	C	D
1	12	13	1	8
2	6	6	15	7
3	16	5	15	12
4	2	3	12	16

Hungarian Algorithm



[능률의 합이 최대화되는 방향으로 매칭] <-> [BBOX 정보의 차이가 최소화되는 방향으로 매칭]

DELAB 22

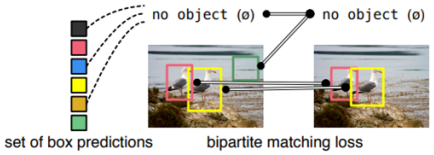
다음 아키텍쳐를 소개하기 전에 Bipartite Matching 개념을 소개하겠다.

알고리즘 수업에서도 나오는 내용인데 위 그림과 같은 대표적인 문제가 있다. (네 명의 사람이 있고 네 개의 일이 있을 때 각각 능률이 테이블에 작성되어있다. 모든 사람이 하나씩 일을 할당받는 경우 어떻게 할당하는 것이 최선인가?)라는 문제이다.
이 문제는 헝가리안 알고리즘을 사용해서 이분 매칭을 하면 해결이 된다.

즉 이렇게 사람과 일을 일대일로 매칭하듯이 예측한 Bounding Box와 Ground Truth Bounding Box 간의 일대일 매칭을 하겠다는 것이다. 이렇게 일대일 매칭을 하면 겹치는 바운딩 박스가 생길 수가 없으므로 NMS같은 후처리가 필요없는 것이다.

DETR

- Bipartite Matching
  - 앞 예제 기준, 사람-> set of box predictions, 일 -> ground truth으로생각하고 매칭
  - L_{match}를 최소화하는 순열(σ^)을 헝가리안 알고리즘으로 찾음



$$\lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_i,\hat{b}_{\sigma(i)}) + \lambda_{\text{L1}}||b_i - \hat{b}_{\sigma(i)}||_1$$

L_{box} : GIoU와 L1 Loss를 사용하므로 BBox의 offset이 비슷할 때 낮은 값을 지님

$$-\mathbb{1}_{\{c_i\neq\varnothing\}}\hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i\neq\varnothing\}}\mathcal{L}_{\text{box}}(b_i,\hat{b}_{\sigma(i)})$$

L_{match} : GT BBox와 예측 BBOX의 정보가 잘 매칭되었을 때 낮은 값을 지님

$b^{\wedge}_{\sigma(i)}$: 순열 σ의 i번째 요소의 예측 박스 좌표

$p^{\wedge}_{\sigma(i)}$: 순열 σ의 i번째 요소에 해당하는 GT의 클래스를 예측한 확률

$$\hat{\sigma} = \underset{\sigma\in\mathfrak{S}_N}{\arg\min}\sum_{i}^{N}\mathcal{L}_{\text{match}}(y_i,\hat{y}_{\sigma(i)}),$$

σ^ : L_{match}를 최소화하도록 이분 매칭한 순열

DELAB 23

실제로 이분 매칭이 어떻게 쓰였는지 소개하겠다.

그 전에 Object query가 Decoder와 FFN을 거쳐서 set of box predictions가 된다. 이 set of box predictions는 예측된 박스에 대한 좌표와 class score를 가진다.

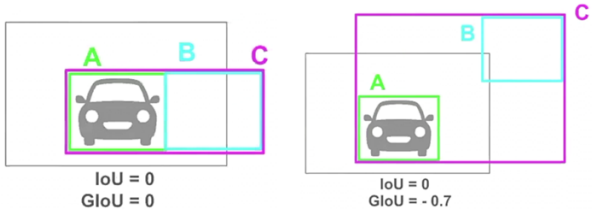
오른쪽에 보이는 그림은 Ground Truth Box로써 정답에 대한 Annotation을 갖게 된다.
이전 ppt의 문제를 기준으로 설명하면 set of box predictions는 사람(a,b,c,d,..), ground trurh는 일(1,2,3,4,...)이라고 생각할 수 있다. 그림만 봤을 때는 ground truth에서 object가 두 개밖에 없는데 이런 경우 1대1 매칭을 가능하게 하기 위해 no object를 네 개를 추가한다.

또한 앞 에제에서는 능률의 합이 최대화되는 방향으로 매칭하였다면, 여기서는 BBOX의 정보의 차이가 최소화되는 방향으로 매칭을 한다.

따라서 L_{match}라는 Loss Function을 제안하였다. 이 L_{match}가 최소화되도록 순열(순서)을 구하는 것이다. L_{match}에 대해 소개하기 전 L_{box}부터 소개하겠다. L_{box}는 GIoU Loss와 L1 Loss를 합한 Loss로써 BBOX의 offset값이 비슷한 경우 낮은 값을 지닌다.

DETR

- GIoU Loss
  - IoU Loss를 이용하면 Predicted Bounding Box가 멀리 있는 경우까지 고려하지 못 함
  - **C \ (A ∪ B)**는 C박스의 영역에서 A와 B박스의 합집합을 뺀 영역



Algorithm 1: Generalized Intersection over Union

input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$
output: $GIoU$

1 For A and B, find the smallest enclosing convex object C, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$
2 $IoU = \frac{|A \cap B|}{|A \cup B|}$
3 $GIoU = IoU - \frac{|C \setminus (A \cup B)|}{|C|}$

A: Ground Truth Bounding Box
B: Predicted Bounding Box
C: A와 B를 포괄하는 박스

L_{IoU} = 1- IoU (0~1)
L_{GIoU} = 1- GIoU (-1~1)

GIoU라는 것은 IoU의 상위 개념이다. 그림에서 A를 GT, B를 예측 bbox, C를 A와 B를 포괄하는 박스라고 하자. 이 때 두 그림 중 B를 더 많이 학습해야 되는 건 오른쪽 그림이다. 왜냐하면 GT BBOX와 더 멀리 떨어져 있기 때문이다. 그러나 IoU의 개념 자체는 A합집합B 중에서 A교집합B가 얼만큼 있는지이므로 멀리 떨어진 경우는 고려할 수 없다. 따라서 IoU를 Loss로 사용한다면 두 그림의 B같은 경우 덜 학습이 된다.

이러한 상황을 막기 위해 나온 것이 GIoU이다. GIoU는 C박스의 영역에서 A와 B박스의 합집합을 뺀 영역까지 고려한다.

따라서 C를 1이라고 할 때, 두 번째 그림의 GIoU는 0-(0.7/1)이 되므로 -0.7이 나온다.

반면 첫 번째 그림은 분모와 분자가 같으므로 0이 나오게 된다. 이러한 GIoU를 Loss로 계산할 때는 1을 빼서 진행함으로써 값이 낮을수록 두 박스가 비슷하다고 표현할 수 있다.

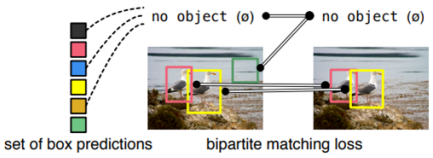
(다시 23page)

즉 이러한 GIoU Loss를 사용함으로써 더 정확하게 두 BBox와의 차이를 계산한다.
(i번째 predicted bbox, i번째 predicted bbox와 매칭된 gt bbox)
다시 L_{match}식을 확인해보자. 좌변은 클래스에 대한 예측 확률이고 우변은 L_{box}이다. 확률은 높을수록 좋지만 Loss 낮을수록 좋으므로 앞에 negative를 붙여줬다. L_{match} 계산을 할 때는 예측 bbox의 class score가 공집합이 아닐 때, 즉 object가 있을 때만 진행한다.

이렇게 계산한 L_{match}가 최소화되도록 이분매칭을 진행하는 것이다.

DETR

- Hungarian Loss
  - Bipartite Matching은 학습과는 무관함, 실제 역전파를 진행할 때는 $L_{hungarian}$을 사용함
  - L_{match}와 유사한 목적 함수
  - Box Loss와 균형을 위해 클래스 예측 확률에 Log term을 사용



$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

그러나 이분매칭은 인퍼런스를 위해 필요한 작업일 뿐 실제 학습과는 무관하다. 실제 역전파를 진행할 때는 $L_{hungarian}$이라는 목적 함수를 사용한다.
이는 L_{match}와 유사한 목적 함수이므로 자세한 설명은 생략하겠다.
차이점은 Box Loss와 균형을 위해 클래스 예측 확률에 Log Term을 사용했다는 것이다.

DETR

- DETR Video (ECCV 2020, Nicolas Carion)

지금까지 설명한 내용을 비디오로 확인하겠다. 논문 저자가 직접 유튜브에 올린 영상을 가져왔다.

먼저 이미지가 cnn을 거쳐서 transformer의 입력 행렬로 변환된다. 여기에 positional encoding을 통해 이미지의 위치가 추가된다. 이후 Encoder에 들어가면 이미지에서 Object를 구분시킨 특징이

나온다. 다음은 n개의 object query가 Decoder에 들어가면 n개의 object에 대한 특징이 나오게 된다. 이 각각의 특징들을 FFN에 넣어서 바운딩 박스를 찾고 classification을 한다. 마지막으로 이분 매칭을 하면서 인퍼런스가 종료된다.

Experiments

- Results
  - Faster RCNN의 업그레이드된 버전과 비교함
  - 작은 물체에 대해서는 성능이 떨어짐 (Backbone의 마지막 Layer를 사용 -> 위치 정보 소실 -> 작은 물체에 대한 정보 소실)
  - 큰 물체에 대해서는 성능이 증가됨 (Attention을 사용 -> 전역적으로 특징 파악 -> 큰 물체에 대한 특징을 잘 찾음)

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
RetinaNet	205/18	38M	38.7	58.0	41.5	23.3	42.3	50.3
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
RetinaNet+	205/18	38M	41.1	60.4	43.7	25.6	44.8	53.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

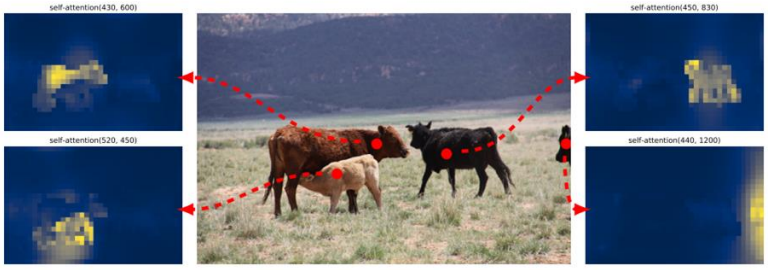
실험 결과를 살펴보겠다.

Faster R-CNN과 비교하였을 때, 작은 물체에 대해서는 성능이 더 떨어졌으나 큰 물체에 대해서는 성능이 더 올랐다고 한다. 성능이 떨어진 이유를 추측해보자면, 트랜스포머가 Feature Map을 입력으로 받을 때 Backbone의 마지막 Layer를 사용하기 때문에 정교한 정보와 위치 정보가 많이 사라진 상태이다. 따라서 작은 물체를 잘 찾지 못한다.

그러나 Attention이라는 것이 멀리 떨어진 픽셀과도 유사도를 구하는 방식이므로 큰 물체에 대한 특징은 잘 찾게 된다.

Experiments

- Encoder Attention
  - Encoder의 마지막 Layer에서 Self Attention Map들을 시각화한 그림
  - Object들을 적절히 분리하는 것을 확인할 수 있음
  - **Encoder:** 서로 다른 Object를 구분함으로써 Detection을 용이하게 하는 역할



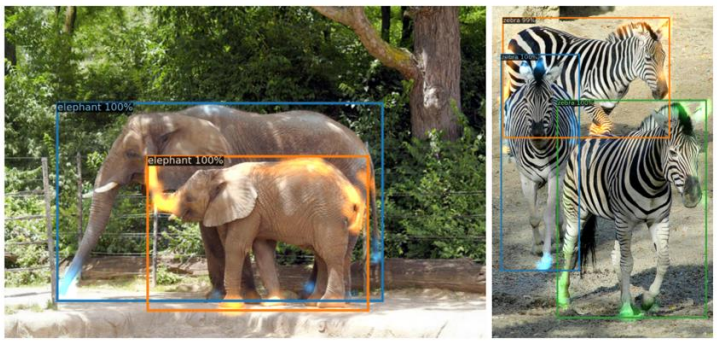
DELAB 28

다음은 Encoder에서 Attention을 확인한 사진이다.

Encoder의 마지막 Layer에서 Self Attention Map을 시각화한 사진들인데, 각 사진을 보면 Object들을 적절히 분리하는 것을 볼 수 있다. 심지어 소가 겹쳐있는 경우에도 특정 소만 잘 찾게 된다. 이러한 결과를 보면 Encoder는 서로 다른 Object를 잘 구분하도록 학습이 되었고 결과적으로 Detection을 더 용이하게 하는 것을 알 수 있다.

Experiments

- Decoder Attention
  - Decoder의 마지막 Layer에서 Cross Attention Map들을 시각화한 그림
  - Object의 말단(Extremities) 부분을 Attention하는 경향이 있음
  - **Decoder:** 서로 다른 Object에 대한 Extremities Feature를 파악함으로써 Detection을 용이하게 하는 역할



DELAB 29

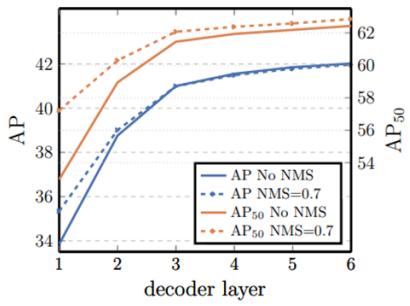
다음은 Decoder에서 Attention을 확인한 사진이다.

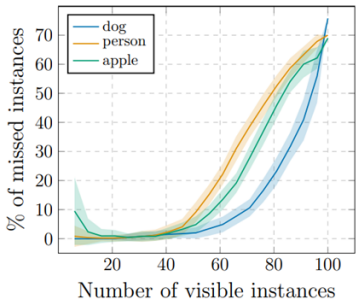
Decoder의 마지막 Layer에서 Cross Attention Map을 시각화한 사진들인데, 각 사진을 보면 Object

의 말단 부분에 대한 특징을 찾는 것을 확인할 수 있다. 따라서 Encoder에서는 object의 전역적인 특징(전체적인 특징)을 찾았다면, Decoder에서는 object의 지역적인 특징(코너,테두리)을 찾은 것이다. 근데 obejct의 말단 부분을 기준으로 bounding box가 쳐지기 때문에 decoder도 결국 detection을 더욱 용이하게 하는 역할임을 알 수 있다.

Experiments

- The Others
  - Decoder Layer가 깊어질수록 NMS 알고리즘이 필요가 없음 (Decoder Layer를 한 개 사용하면 한 개의 Object에 여러 BBOX가 나옴)
  - 50개의 인스턴스에 대해서는 미싱율이 낮지만 이후부터는 미싱율이 높음 (학습 데이터에 인스턴스가 50개 이상인 이미지는 적음)





DELAB 30

마지막으로 다른 실험 결과들을 알아 보겠다.

첫 번째는 Decoder Layer가 깊어질수록 한 개의 Object에 하나의 BBOX가 나왔다는 사실이다. 이분 매칭을 하지 않고도 Decoder Layer만으로도 겹치는 BBOX문제를 해결한 것을 볼 수 있다.

두 번째는 50개 이하의 인스턴스에 대해서는 미싱율이 낮지만 이후부터는 미싱율이 높았다는 것이다. 이건 학습 데이터에 인스턴스가 50개 이상인 이미지가 적으므로 오버피팅이 나왔다고 표현할 수 있다.