DETR : End-to-End Object Detection with Transformers

Author : Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, Sergey Zagoruyko – Facebook AI

ECCV 2020
https://arxiv.org/abs/2005.12872

Yoo – Seok Oh in Software engineering Undergraduate course

Nov 11(Mon), 2024

목차

Chapter.1

Background

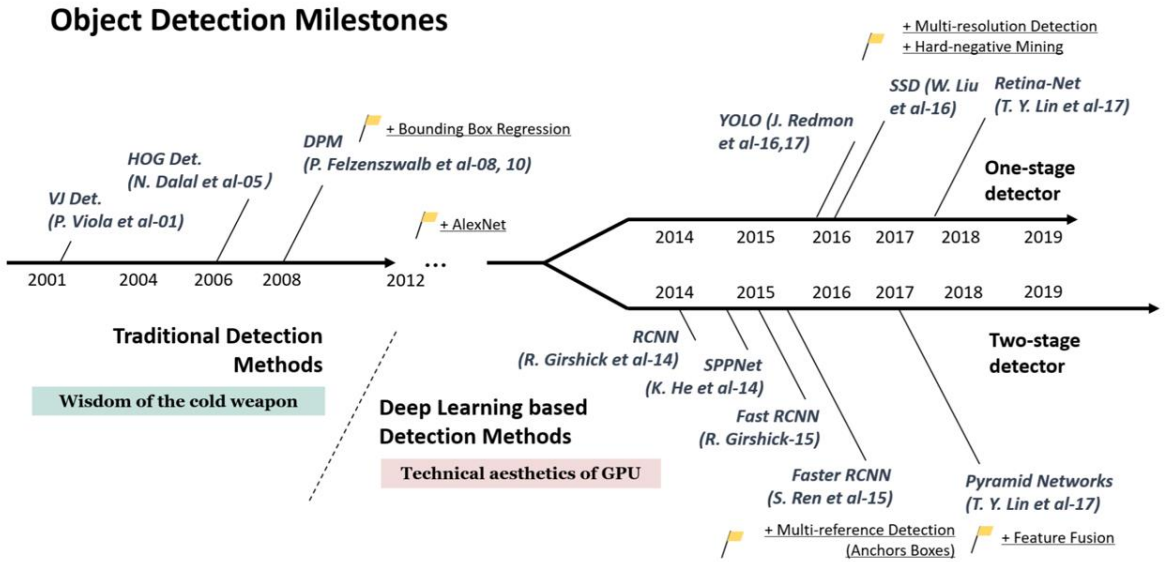
Object Detection

✓ 여러 객체의 위치(localization)를 찾고 class를 분류(classification)해주는 것

✓ Bounding Box로 여러 객체의 위치를 출력, Box 안에 있는 객체가 무엇인지 분류

✓ 1-stage detector와 2-stage detector로 나뉜다

이 둘의 차이는 detector 앞단에 후보 지역을 추천하는 RPN 존재 여부



Transformer

✓ 기존 RNN 기반의 encoder-decoder 구조에서 RNN 모듈을 제거하고 attention 만을 이용하여 번역등의 task를 수행하는 모델 구조

✓ 임베딩 된 input 시퀀스를 Encoder-Decoder 구조에 태워서 ouput을 출력

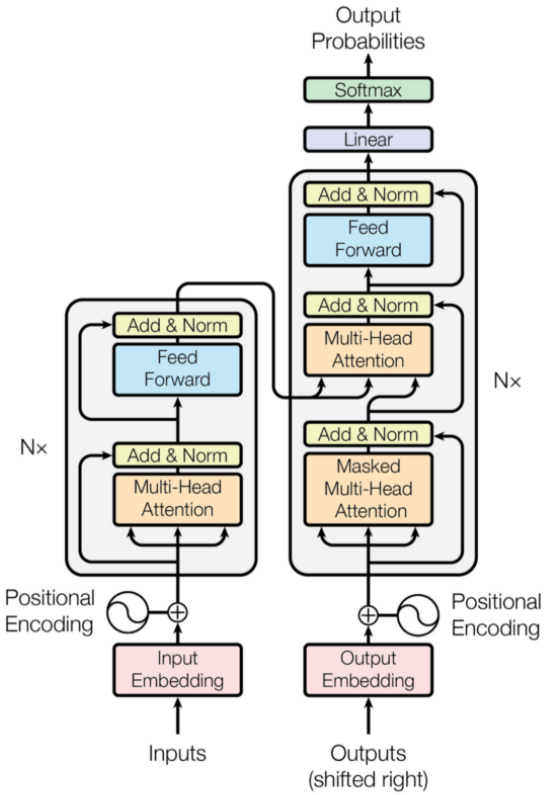
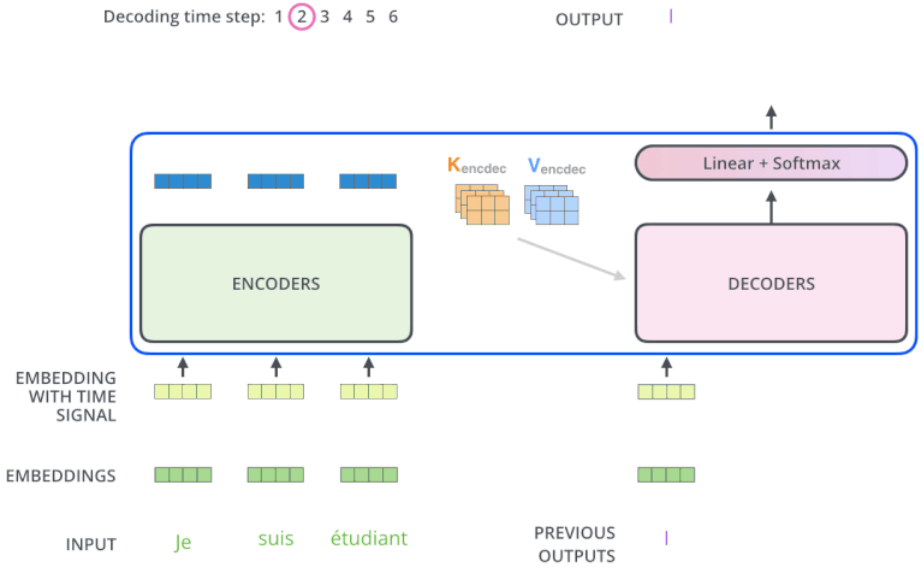


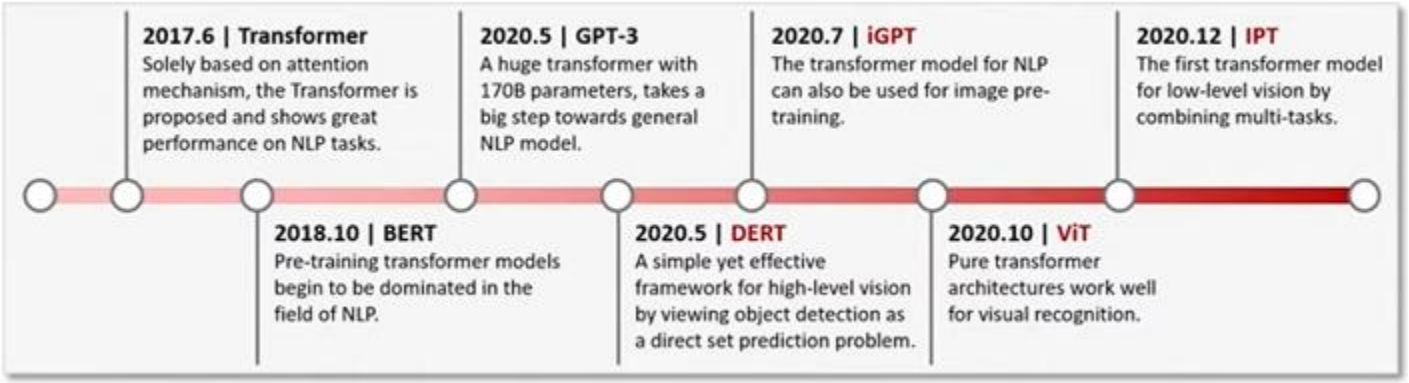
Figure 1: The Transformer - model architecture.



Transformer in Vision

현재 Transformer는 매우 핫하다!

* (computer Vision and Machine Learning) 에서 BERT, self-attention, transformer가 제목에 포함된 논문 갯수



* Vsion 분야에서 Transformer의 발전

Trend	Dataset	Best Model	Paper	Code	Compare
	COCO test-dev	Co-DETR			See all
	COCO minival	Co-DETR			See all
	COCO-O	EVA			See all
	PASCAL VOC 2007	Cascade Eff-B7 NAS-FPN (Copy Paste pre-training, single-scale)			See all
	COCO 2017 val	Relation-DETR (Swin-L 2x)			See all
	COCO 2017	MaxViT-B			See all
	CrowdHuman (full body)	InternImage-H			See all
	GRAZPEDWRI-DX	YOLOv8x			See all
	CPPE-5	TridentNet			See all
	Waymo 2D detection all_ns f0val	YOLOX-L			See all

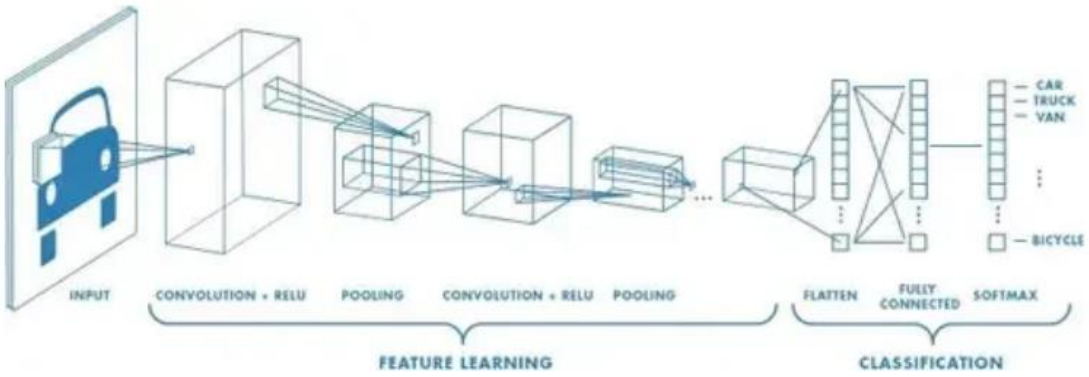
* Object Detection에서의 SOTA

ViT (Vision Transformer)

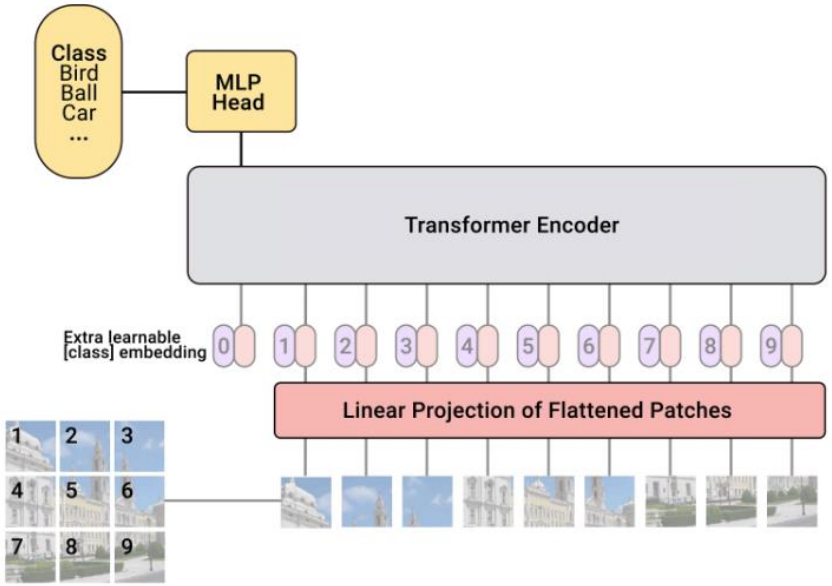
✓ 이미지를 여러 Patch로 나누어서 임베딩을 수행한 후, 각 패치를 하나의 token으로 이용하여 transformer 구조에 태운다
이 때, class token을 추가하여 학습 후 , 뒷단의 MLP Layer에 태워 이미지의 클래스를 예측

✓ CNN의 translation invariance, locality와 같은 inductive bias를 크게 줄여서 모델의 자유도를 얻음
단, ViT는 학습에 다량의 데이터가 필요

일반적인 CNN 기반 Classification



ViT 기반 Classification

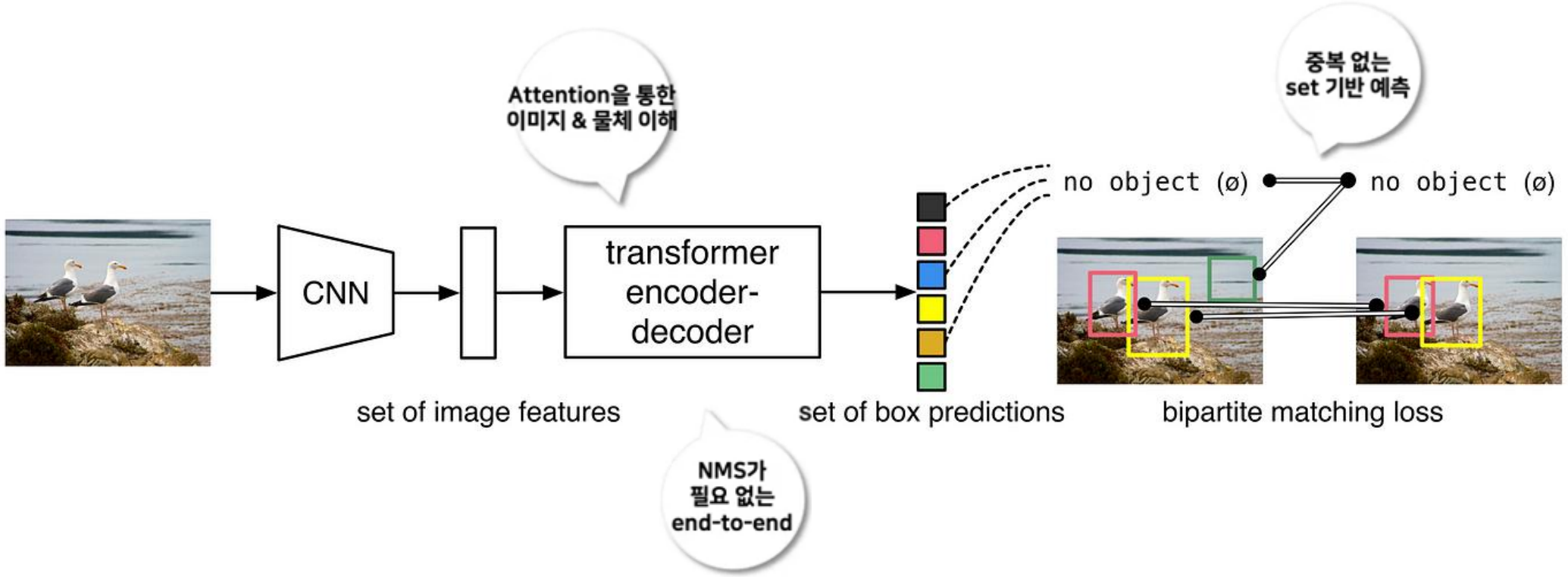


DETR (Detection with Transformers)

✓ Transformer와 이분 매칭(Bipartite-matching) 기반의 새로운 detection 구조

✓ Object detection을 direct set-predicition의 문제로 접근하여, end-to-end 모델로서 RPS와 NMS가 필요하지 않음

✓ 구조적으로 간결하고 다른 task에 확장성이 높으며, Attention 메커니즘을 사용하여 기존 Faster R-CNN에 비해 더 높은 성능을 보임



Object Detection VS DETR

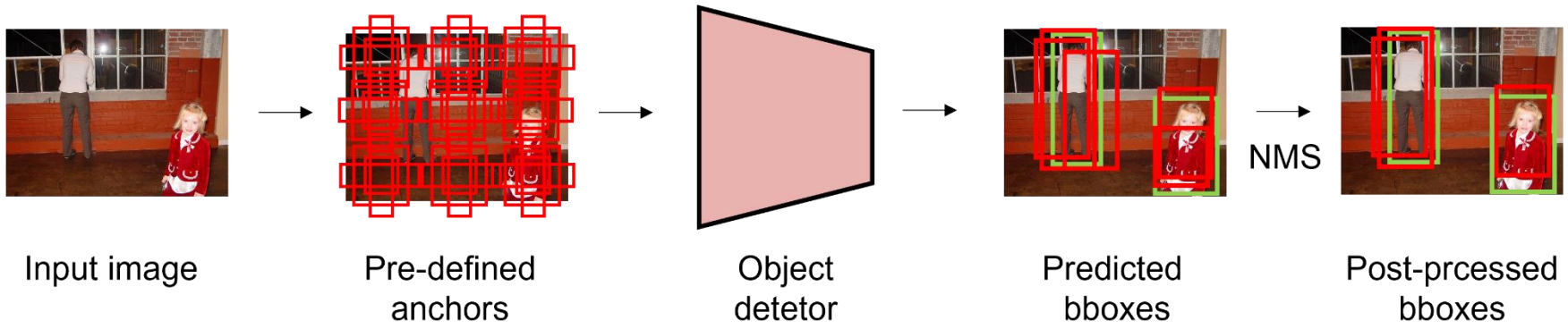


Fig 1. 기존의 object detector의 detection 과정

VS

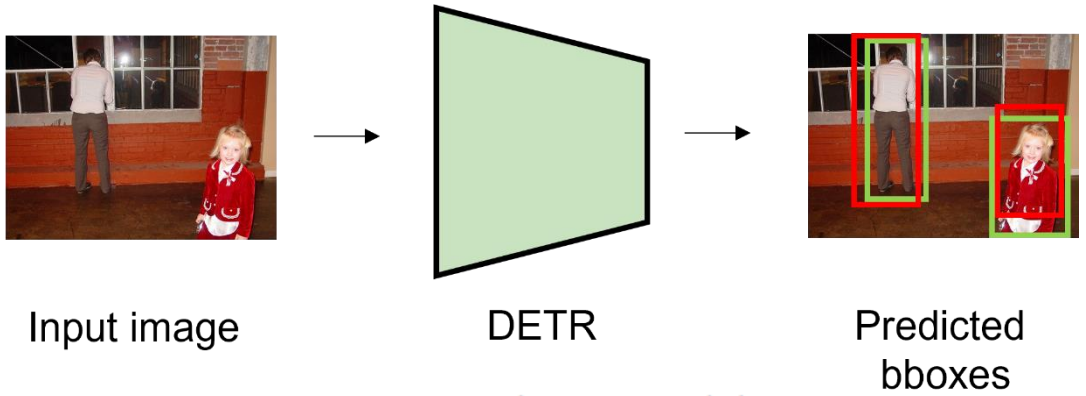
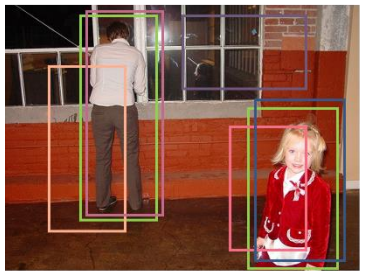


Fig 2. DETR의 detection 과정

Hungarian Algorithm

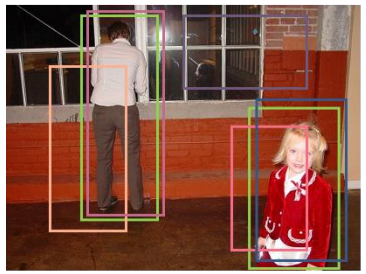
어떠한 집합 I와 matching 대상인 집합 J가 있으며, i∈I를 j∈J에 matching하는데 드는 비용을 c(i,j)라고 할 때, σ:I→J로의 일대일 대응 중에서 가장 적은 cost가 드는 matching에 대한 permutation σ을 찾는 것



	1	2	ø	ø	ø
1	12	11	1	1	1
2	4	2	8	6	9
3	1	3	5	7	8
4	2	5	6	7	9
5	2	1	9	10	6

Permutation = [1, 2, 3, 4, 5]
Matching score = 12 + 2 + 5 + 7 + 6 = 32

Fig 3. matching score가 높은 permutation



	1	2	ø	ø	ø
1	12	11	1	1	1
2	4	2	8	5	9
3	1	3	5	7	8
4	2	5	6	7	4
5	2	1	9	10	6

Permutation = [3, 4, 1, 5, 2]
Matching score = 1 + 5 + 1 + 4 + 1 = 12

Fig 4. matching score가 낮은 permutation

cost에 대한 행렬을 입력 받아, matching cost가 최소인 permutation을 출력

Bounding Box Loss

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

$$\mathcal{L}_{\text{iou}}(b_{\sigma(i)}, \hat{b}_i) = 1 - \left(\frac{|b_{\sigma(i)} \cap \hat{b}_i|}{|b_{\sigma(i)} \cup \hat{b}_i|} - \frac{|B(b_{\sigma(i)}, \hat{b}_i) \setminus b_{\sigma(i)} \cup \hat{b}_i|}{|B(b_{\sigma(i)}, \hat{b}_i)|}\right)$$

GIoU Loss

class	ℓ_1	GIoU	AP	Δ	AP_{50}	Δ	AP_S	AP_M	AP_L
✓	✓		35.8	-4.8	57.3	-4.4	13.7	39.8	57.9
✓		✓	39.9	-0.7	**61.6**	0	**19.9**	43.2	57.9
✓	✓	✓	**40.6**	-	**61.6**	-	**19.9**	**44.3**	**60.2**

* Box Loss에 대한 ablation study 결과

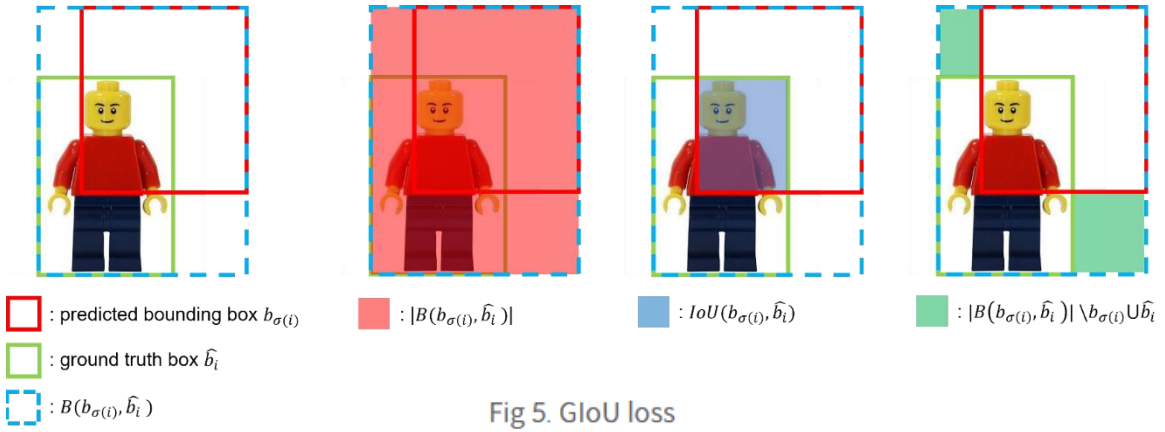


Fig 5. GIoU loss

$$GIoU = IoU(b_{\sigma(i)}, \hat{b}) - \frac{|B(b_{\sigma(i)}, \hat{b})| \setminus b_{\sigma(i)} \cup \hat{b}_i}{|B(b_{\sigma(i)}, \hat{b})|}$$

$$\mathcal{L}_{iou}(b_{\sigma(i)}, \hat{b}) = 1 - GIoU$$

✓ 일반적인 detector에서의 bounding box loss는 예측과 ground truth 간의 상대적인 좌표 및 크기를 이용하여 정의 (Anchor에서 정의된 box를 얼마나 움직이고, 얼마나 키워야 ground truth에 가까워지는지)

✓ 반면 DETR의 경우 절대적인 bounding box의 좌표 및 크기를 direct 하게 예측하므로 loss를 계산 시 일반적인 L1 loss 외에 scale 보정이 필요함 -> GIoU 사용

GIoU가 모델의 성능을 개선하는데 중요한 역할을 수행

Background (Contribution)

Contribution

1. 본 논문에서는 object detection을 direct set prediction으로 정의하여, transformer와 bipartite matching loss를 사용한 DETR(DEtection TRansformer)을 제안한다.

2. DETR은 COCO dataset에 대하여 Faster R-CNN과 비슷한 수준의 성능을 보인다.

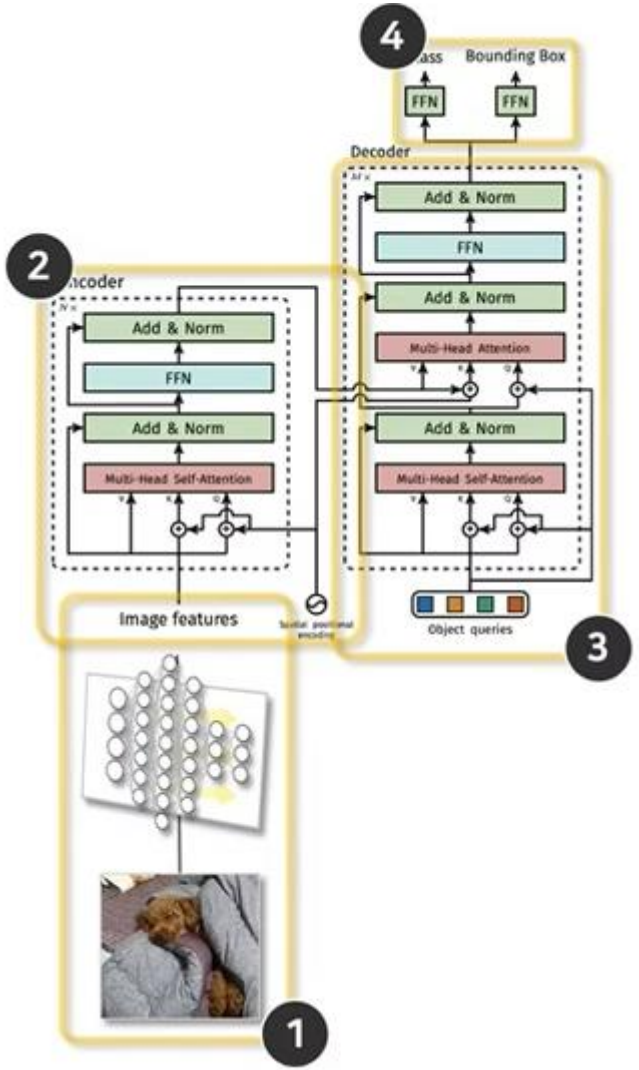
3. Self-attention을 통한 전역 정보를 활용함으로써 크기가 큰 객체를 Faster R-CNN보다 훨씬 더 잘 포착한다.

Chapter.2

Model Architecture

Model Architecture

Overview



간단!

```python
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):

    def __init__(self, num_classes, hidden_dim, nheads,
                 num_encoder_layers, num_decoder_layers):
        super().__init__()
        # We take only convolutional layers from ResNet-50 model
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
                                          num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))

    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([
            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
        ], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
                             self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```

Model Architecture

CNN Backbone

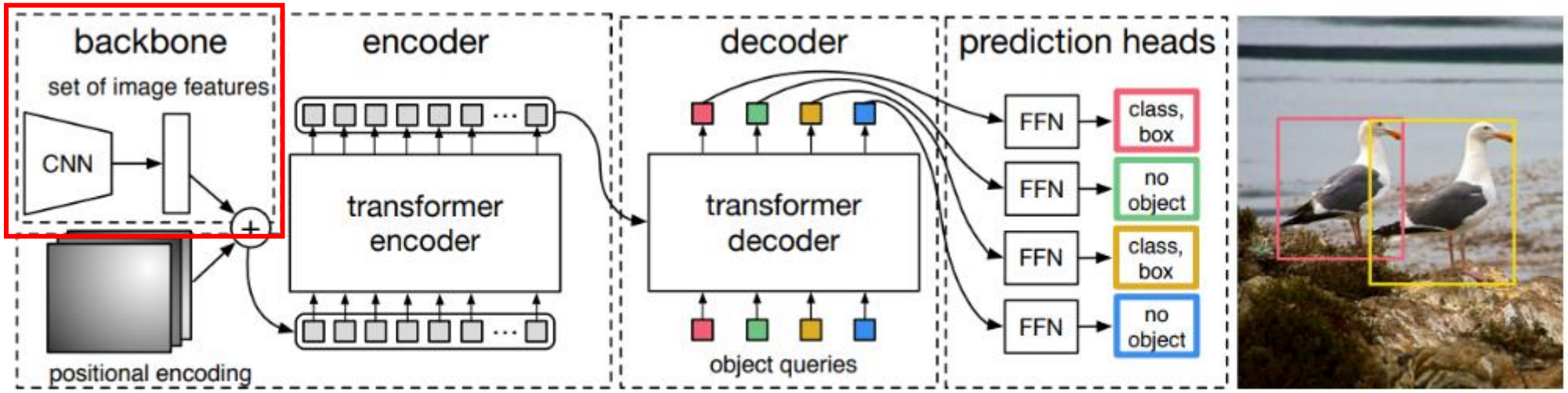


Fig 9. DETR architecture

먼저 입력 이미지 $x_{img} \in \mathbb{R}^{3\times H_0\times W_0}$를 CNN backbone network에 입력하여, **feature map** $f \in \mathbb{R}^{C\times H\times W}$**를 생성**

이 때 $C = 2048$이며, $H, W = \frac{H_0}{32}, \frac{W_0}{32}$

차원수를 높여 모델이 더 복잡하고 다양한 특징을 포착할 수 있도록 하고
해상도를 32배 축소시켜 이미지의 중요한 정보만을 요약하여 Transformer구조에 들어가기에 적합하게 한다

Model Architecture

Encoder

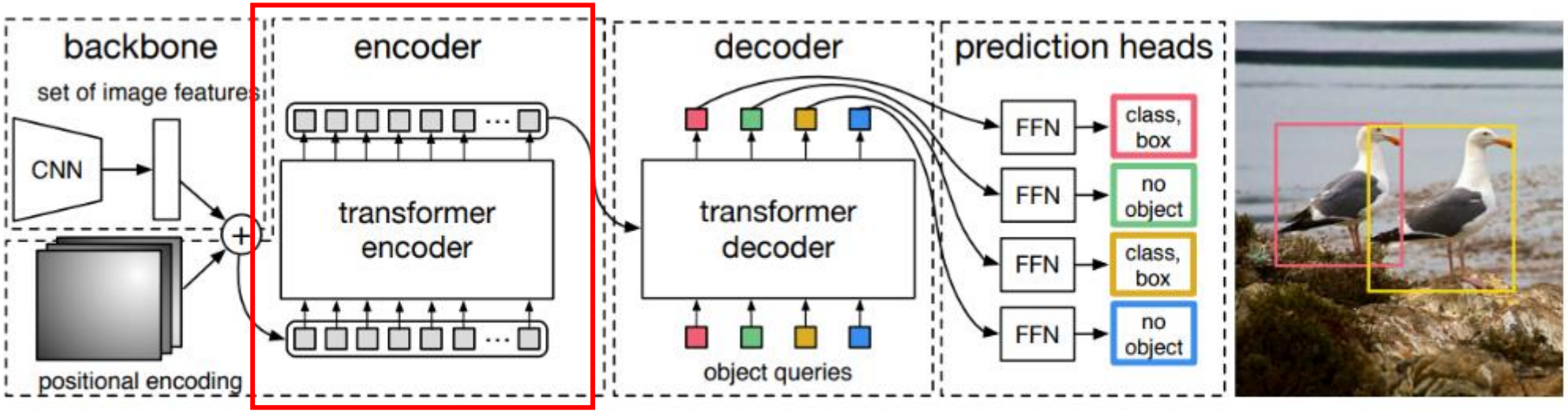


Fig 9. DETR architecture

1. 이미지를 Transformer 입력 형태로 변환
2. Positional Encoding

   Transformer 아키텍처와 동일하게 Positional Encoding을 더하여 입력
3. Self-Attention

   입력된 각 픽셀들 간 어떠한 관계가 있는지를 global 하게 학습
4. FFN

   Output에 대한 normalization

Model Architecture

Decoder

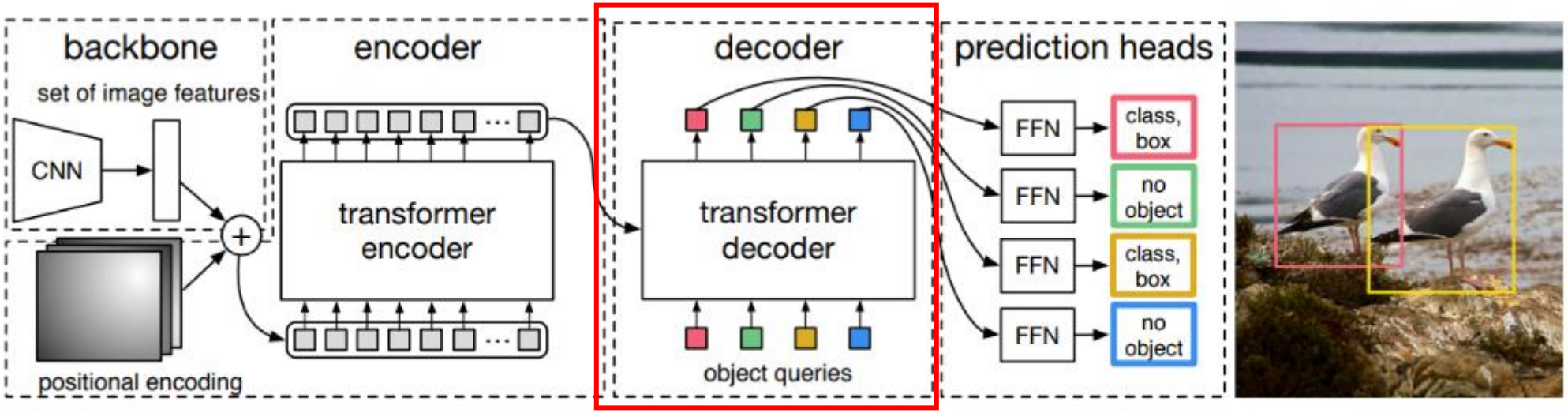


Fig 9. DETR architecture

1. **Object query 입력**
2. **Decoder Self-Attention**

   Self Attention을 통한 query slot 간 관계 학습

3. **Encoder-Decoder**

   Encoder의 결과물과 query slot 간 attention을 통해 어느 query가 어떤 위치에서 object를 찾을 수 있을지 학습

4. **FFN**

   Output에 대한 normalization

Model Architecture

FFN

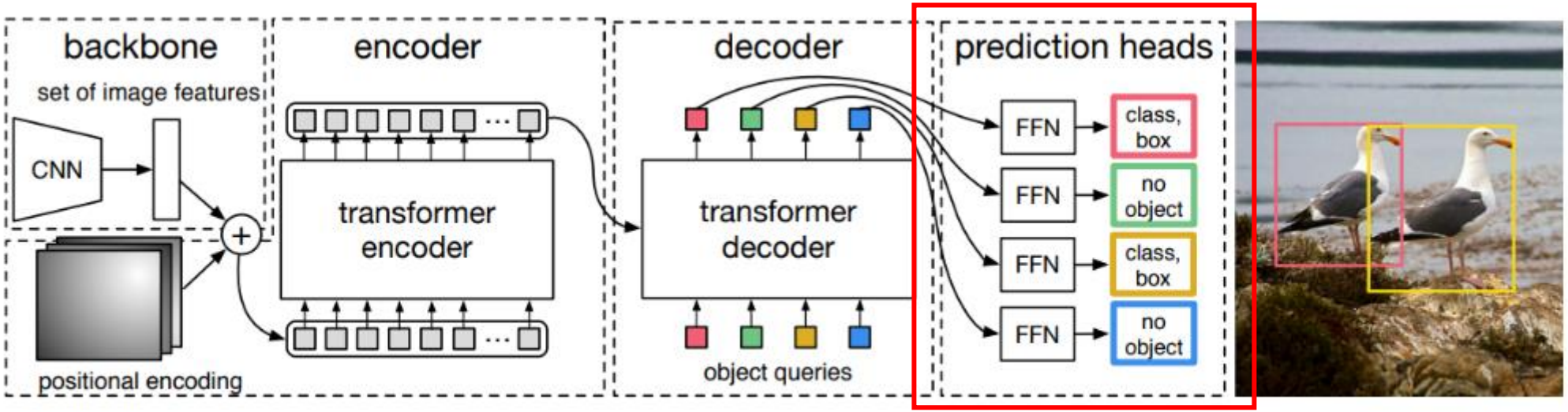


Fig 9. DETR architecture

1. 3개의 linear layer와 ReLU 활성화 함수로 이루어짐

2. 이미지에 대한 class label과 bounding box에 좌표(normalized center coordinate, width, height)를 예측

Transformer for NLP task vs DETR Transformer

Overview

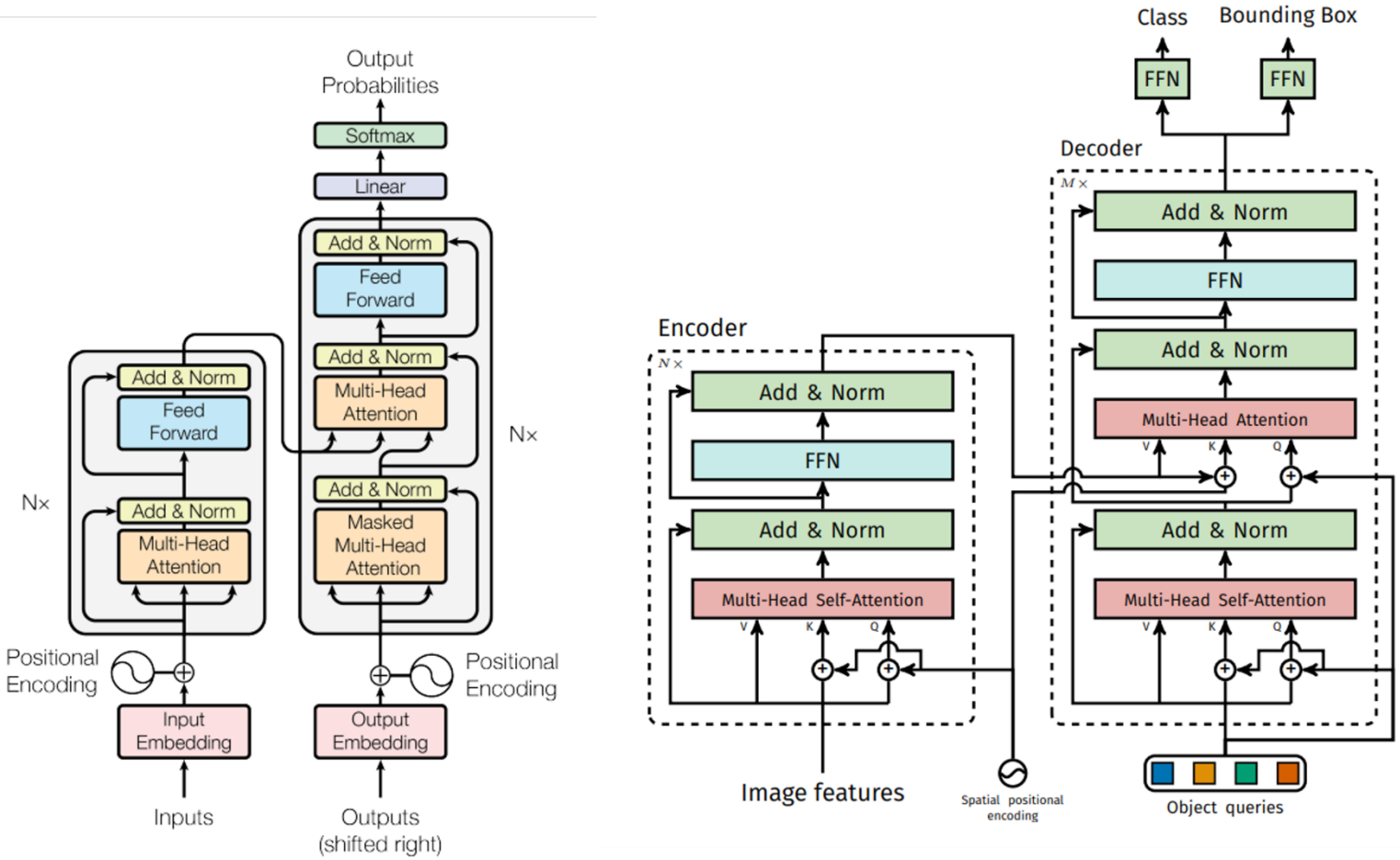
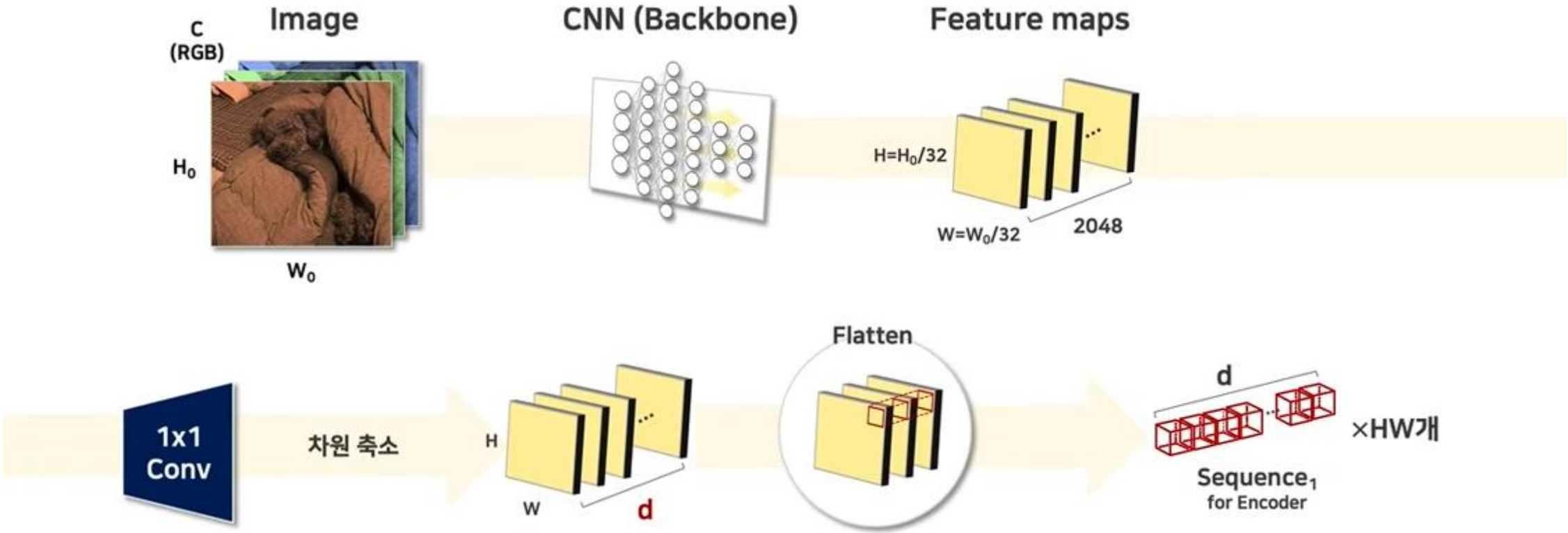


Fig 6. Vanilla Transformer와 DETR의 Transformer

Transformer for NLP task vs DETR Transformer

Input Image

1. Backbone CNN에 이미지를 태워 feature map을 얻는다
2. 얻은 feature map을 1x1 convolution layer에 태워, 미리 설정한 토큰 임베딩 차원(d=256)으로 축소
3. 최종적으로 dxHW로 flatten 하여 transformer에 입력으로 사용할 수 있는 시퀀스를 얻음



Transformer for NLP task vs DETR Transformer

Positional Encoding

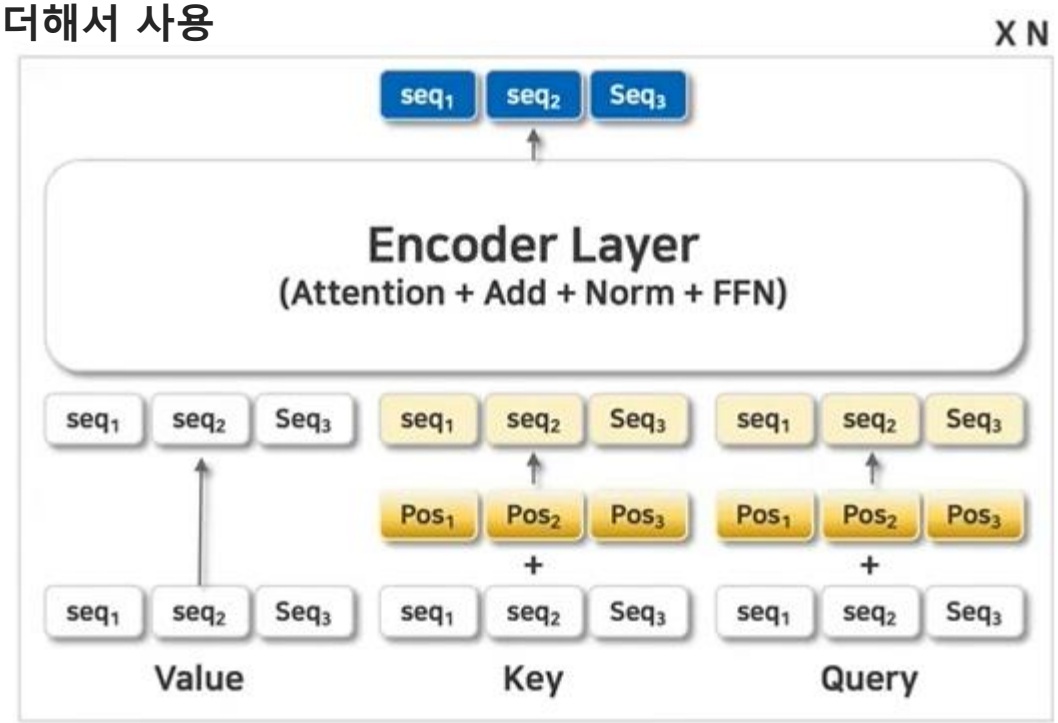
1. Transformer는 한번에 모든 시퀀스를 입력하므로 순서 정보를 가지지 않으므로 positional encoding이 필요함
2. DETR에서는 일반적인 Transformer의 positional encoding과 다르게 2D positional encoding을 사용함
3. 모든 encoder layer마다 positional encoding을 query, key에 더하여 입력으로 사용
4. Decoder에서도 object query에 positional encoding을 더해서 사용

1D positional encoding (Standard Transformer [1])

```
sinusoid_table[:, 0::2] = np.sin(sinusoid_table[:, 0::2])  # dim 2i
sinusoid_table[:, 1::2] = np.cos(sinusoid_table[:, 1::2])  # dim 2i+1
return torch.FloatTensor(sinusoid_table).unsqueeze(0)
```

2D positional encoding (DETR [2])

```
pos_x = x_embed[:, :, :, None] / dim_t
pos_y = y_embed[:, :, :, None] / dim_t
pos_x = torch.stack((pos_x[:, :, :, 0::2].sin(), pos_x[:, :, :, 1::2].cos()), dim=4).flatten(3)
pos_y = torch.stack((pos_y[:, :, :, 0::2].sin(), pos_y[:, :, :, 1::2].cos()), dim=4).flatten(3)
pos = torch.cat((pos_y, pos_x), dim=3).permute(0, 3, 1, 2)
return pos
```



Transformer for NLP task vs DETR Transformer

Object queries

- ✓ 정보를 담기위한 일종의 그릇(slot)
- ✓ Transformer Decoder에서 각 객체의 위치와 클래스를 예측하기 위해 사용하는 학습 가능한 벡터
- ✓ Decoder의 Encoder-Decoder attention을 통해 이미지의 어느 부분을 위주로 봐야할지 (어느 위치가 확률이 높은지) 학습
- ✓ Decoder의 Self-attention을 통해 자신들의 역할을 어떻게 분배하여 최적의 1대1 매칭을 수행할 수 있을 지를 학습

보물지도 (이미지)

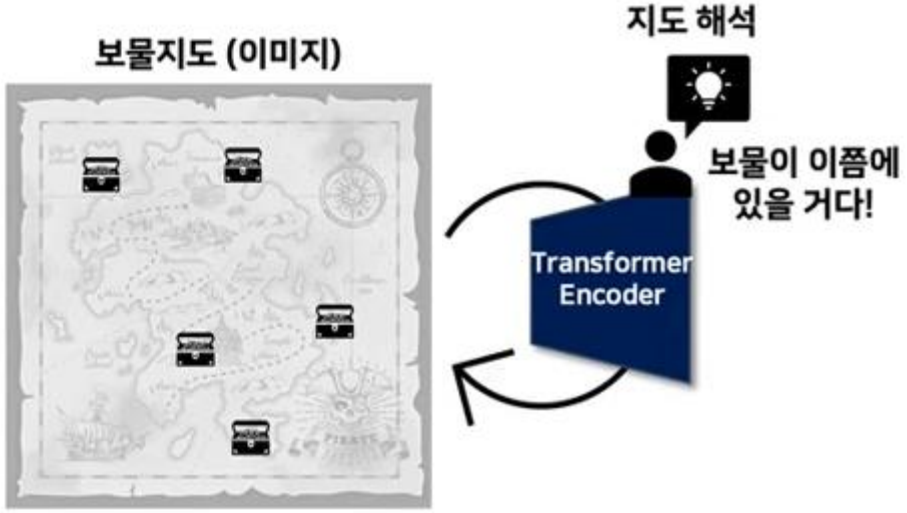
보물 사냥꾼 (Object Queries)

"정보가 없으니 일단 아무 데나 가보자"

Transformer for NLP task vs DETR Transformer

Object queries

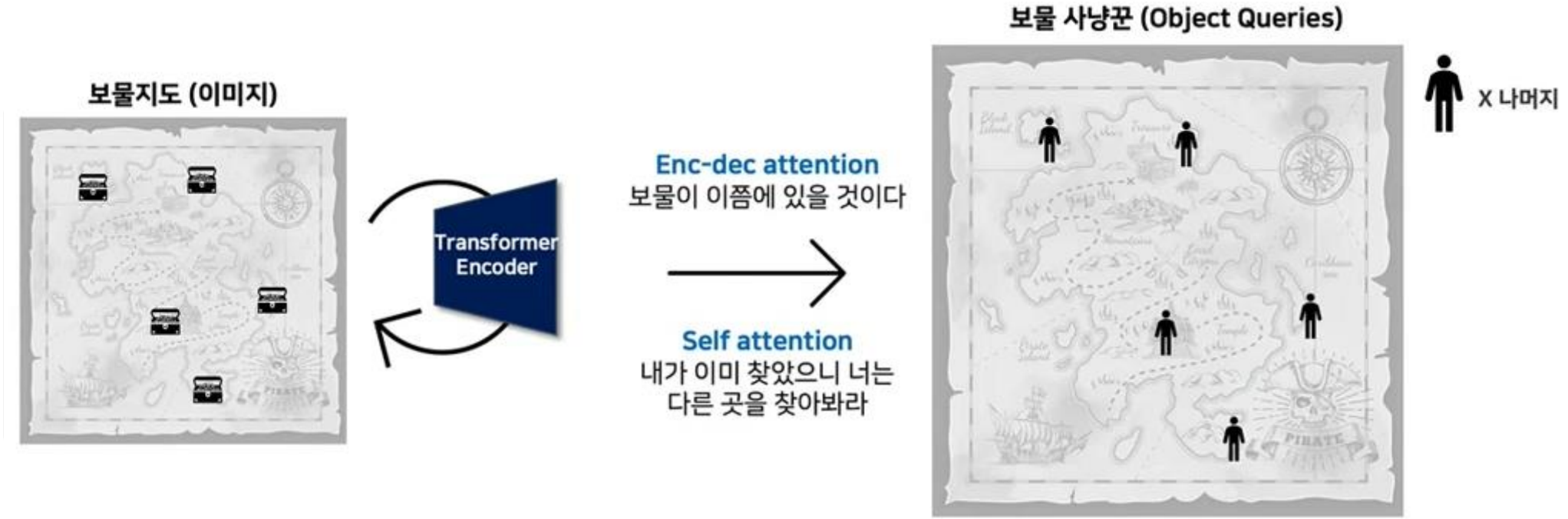
- ✓ 정보를 담기위한 일종의 그릇(slot)
- ✓ Transformer Decoder에서 각 객체의 위치와 클래스를 예측하기 위해 사용하는 학습 가능한 벡터
- ✓ Decoder의 Encoder-Decoder attention을 통해 이미지의 어느 부분을 위주로 봐야할지 (어느 위치가 확률이 높은지) 학습
- ✓ Decoder의 Self-attention을 통해 자신들의 역할을 어떻게 분배하여 최적의 1대1 매칭을 수행할 수 있을 지를 학습



Transformer for NLP task vs DETR Transformer

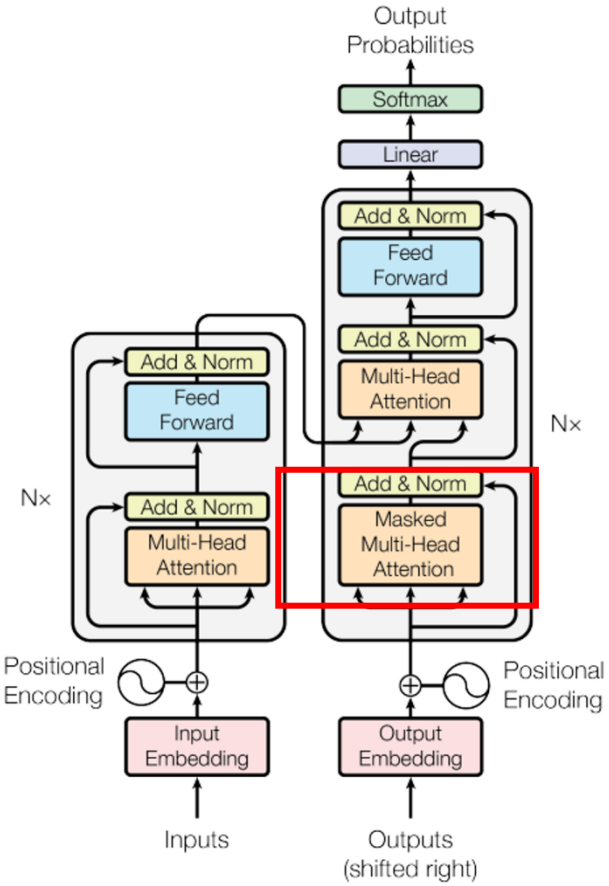
Object queries

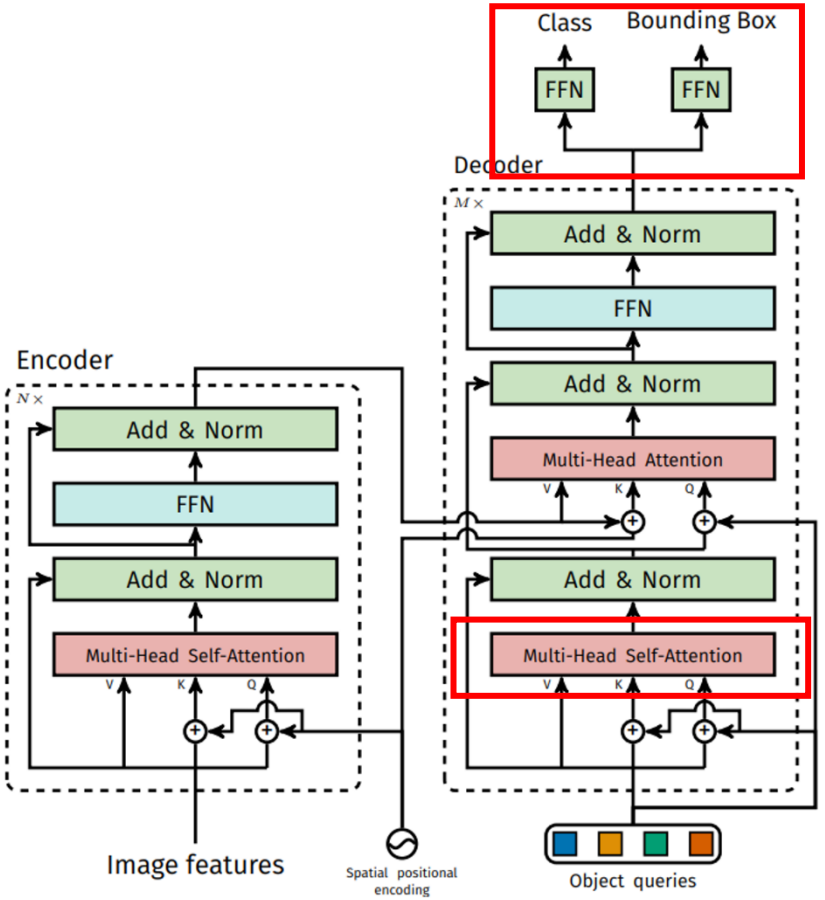
- ✓ 정보를 담기위한 일종의 그릇(slot)
- ✓ Transformer Decoder에서 각 객체의 위치와 클래스를 예측하기 위해 사용하는 학습 가능한 벡터
- ✓ Decoder의 Encoder-Decoder attention을 통해 이미지의 어느 부분을 위주로 봐야할지 (어느 위치가 확률이 높은지) 학습
- ✓ Decoder의 Self-attention을 통해 자신들의 역할을 어떻게 분배하여 최적의 1대1 매칭을 수행할 수 있을 지를 학습



Transformer for NLP task vs DETR Transformer

Not masked Multi head self-attention, Two head





Chapter.3

Training Process

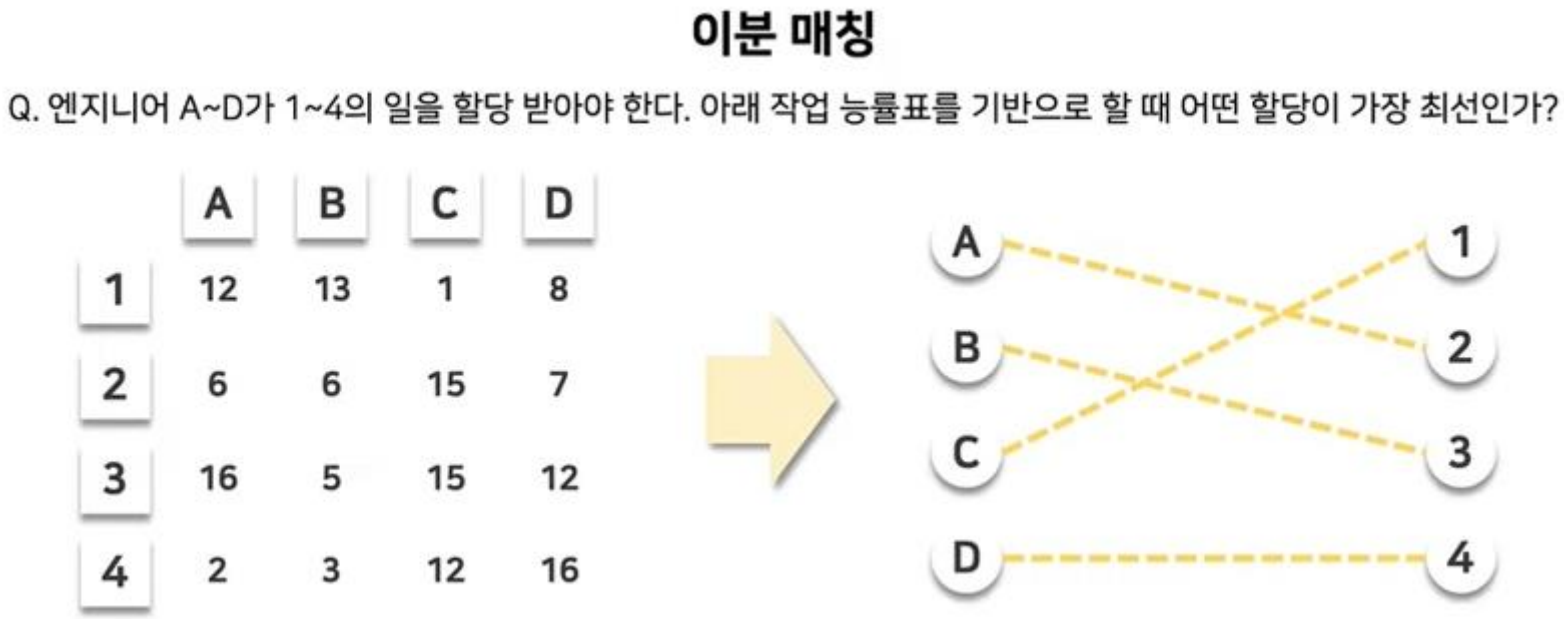
Training Process

Bipartite Matching (feat. Hungarian Algorithm)

✓ Direct set prediction에서 중요한 두 가지 요소는

1. 예측과 GT 사이에 중복이 없는 일대일 매칭을 가능케 해야 하며,
2. 한 번의 추론에서 object set을 예측하고 그들의 관계를 모델링 할 수 있어야 한다.

✓ 이를 만족시키는 object detector를 학습시키기 위해 DETR은 헝가리안 알고리즘을 활용한 이분 매칭을 통해 loss를 정의함

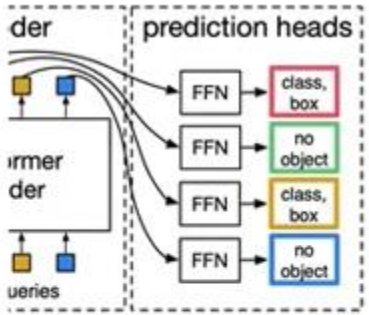


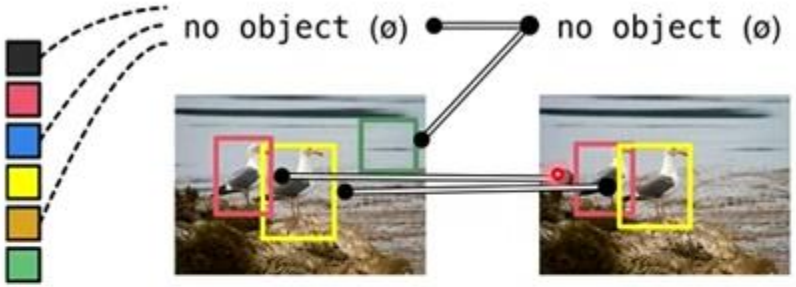
Bipartite Matching (feat. Hungarian Algorithm)

✓ Object query 마다 예측된 결과물과 ground truth set 간 Hungarian Algorithm 기반 매칭 수행

✓ 이 때 매칭의 기준으로 $\mathcal{L}_{match}$ 를 활용하고, $\mathcal{L}_{match}$ 를 최소화하는 최적의 순열 $\hat{\sigma}$ 을 찾는다

✓ Anchor는 동일한 ground truth object와의 중복 예측을 허용하는 반면,
DETR은 set-prediction과 ground-truth 간 일대일 매칭을 수행하여 중복을 배제





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

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

클래스 예측 Cost　　박스 좌표 예측 Cost

Training Process

Bipartite Matching (feat. Hungarian Algorithm)

✓ Object query 마다 예측된 결과물과 ground truth set 간 Hungarian Algorithm 기반 매칭 수행

✓ 이 때 매칭의 기준으로 $\mathcal{L}_{match}$ 를 활용하고, $\mathcal{L}_{match}$ 를 최소화하는 최적의 순열 $\hat{\sigma}$ 을 찾는다

✓ Anchor는 동일한 ground truth object와의 중복 예측을 허용하는 반면,
DETR은 set-prediction과 ground-truth 간 일대일 매칭을 수행하여 중복을 배제

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

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

클래스 예측 Cost

박스 좌표 예측 Cost

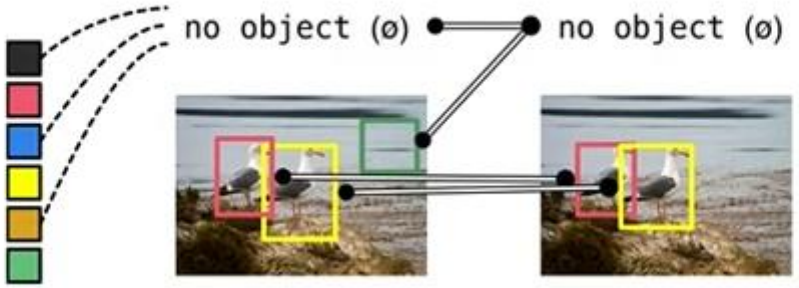
$\hat{p}_{\sigma(i)}(c_i)$ 순열 σ의 i번째 요소가 해당하는 GT의 클래스를 예측한 확률

$\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$ 순열 σ의 i번째 요소의 예측 박스 좌표와 해당하는 GT의 박스 좌표 간 Loss

Hungarian Loss

✓ 앞선 Hungarian algorithm으로 찾은 최적의 순열 $\hat{\sigma}$ 을 이용해 학습을 위한 loss를 계산

✓ 일반적인 object detection task에서의 loss와 유사하게 정의하지만,
bounding box loss에 GIoU를 추가하여 box의 스케일에 둔감한 loss를 정의



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

Box Loss와의 균형을 위해
이분 매칭과 달리 Log term 사용

Chapter.4

Experiments

DETR vs Faster R-CNN

✓ Faster R-CNN과의 합리적이 비교를 위하여 Faster R-CNN에 다음과 같은 요소를 추가

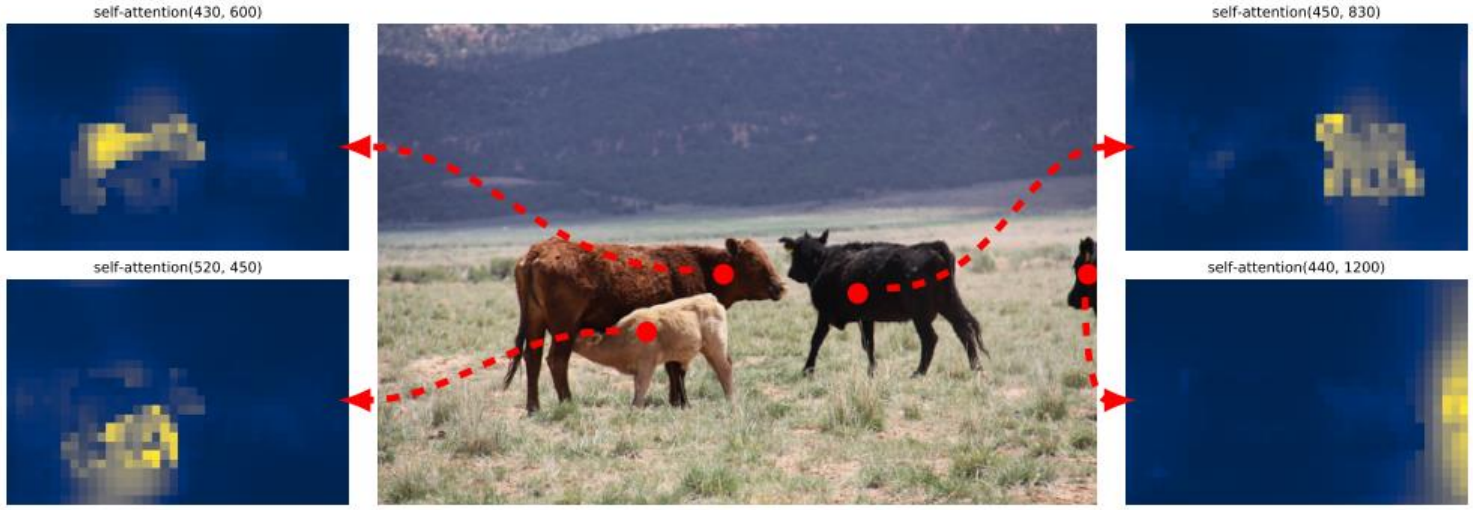
1. Bounding box loss function에 g-IoU Loss를 추가
2. DETR 학습 시와 동일한 crop augmentation을 추가
3. 더욱 긴 training schedule을 이용하여 학습

✓ 크기가 큰 물체에 대해서는 Faster R-CNN 대비 높은 성능을 보였지만, 크기가 작은 물체에 대해서는 낮은 성능을 기록

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
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
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**

Ablation : Encoder

#layers	GFLOPS/FPS	#params	AP	AP_{50}	AP_S	AP_M	AP_L
0	76/28	33.4M	36.7	57.4	16.8	39.6	54.2
3	81/25	37.4M	40.1	60.6	18.5	43.8	58.6
6	86/23	41.3M	40.6	61.6	19.9	44.3	60.2
12	95/20	49.2M	41.6	62.1	19.8	44.9	61.9



* 인코더 마지막 layer 특정 픽셀에서의 self-attention 시각화 (attention map)

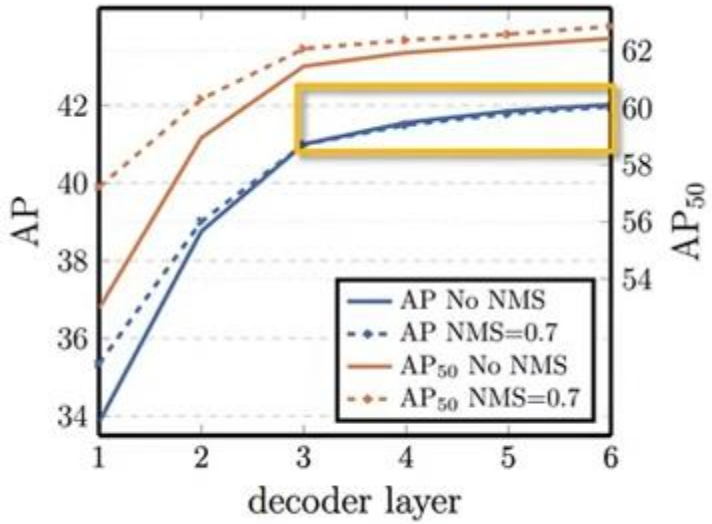
✓ Encoder를 사용하지 않는 경우 약 4~5 AP 감소가 있으며, 특히 큰 물체에 대한 detection 성능이 크게 저하됨

✓ DETR의 attention mechanism이 이미지 내 물체를 분리하는 데에 굉장히 중요함

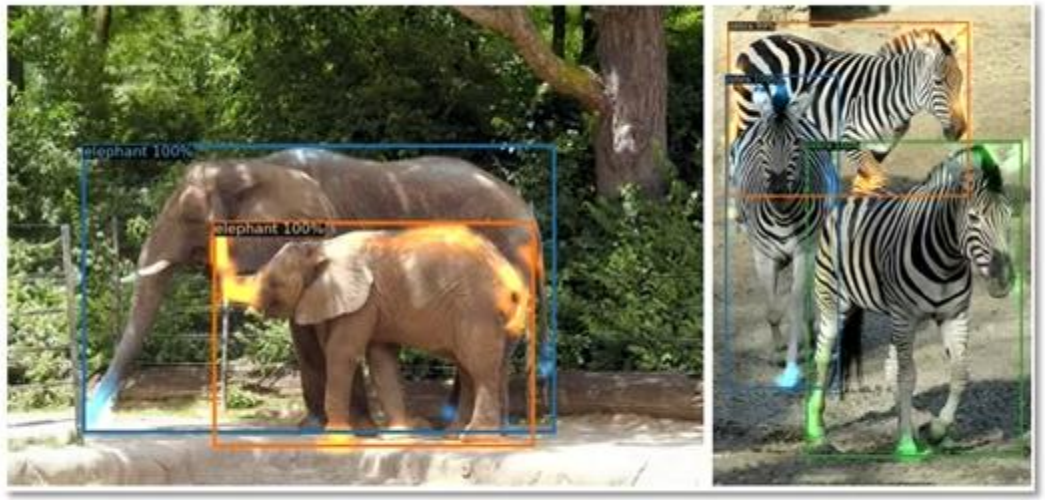
✓ Attention map을 통하여 Encoder에서 이미 물체를 어느 정도 구분하고 있음을 알 수 있고,
따라서 학습된 임베딩을 key와 value로 사용하는 Decoder가 detection 하는 과정을 좀 더 쉽게 만들어 줌

Experiments

Ablation : Decoder



* Decoder layer별 AP 및 NMS의 효과



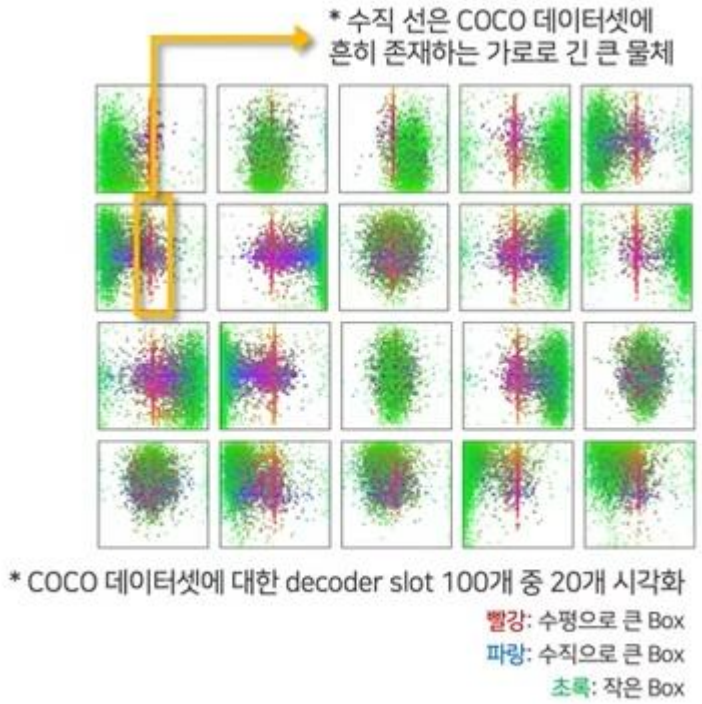
* 디코더 어텐션 시각화

✓ Decoder의 깊이가 깊어질수록 예측 정확도가 높아지며, NMS를 추가하여도 효과가 없음

✓ Decoder가 물체를 탐지하는 학습 과정을 통해 각 object들의 말단에 어텐션을 크게 주도록 학습되며, object가 겹치는 경우 잘못된 어텐션을 주지 않도록 학습됨

✓ Encoder가 global attention을 통해 이미지 내의 물체를 잘 나눈 후, decoder는 클래스 및 물체의 바운더리를 잘 추출하도록 attention을 준다

Decoder’s Object Queries



* COCO 데이터셋에 대한 decoder slot 100개 중 20개 시각화
빨강: 수평으로 큰 Box
파랑: 수직으로 큰 Box
초록: 작은 Box

* Object query의 평균적인 위치 시각화
(COCO val dataset)

Active object query

✓ 각 object query 마다 이미지의 특정한 부분 및 박스 크기에 대한 예측을 주로 수행

Chapter.5

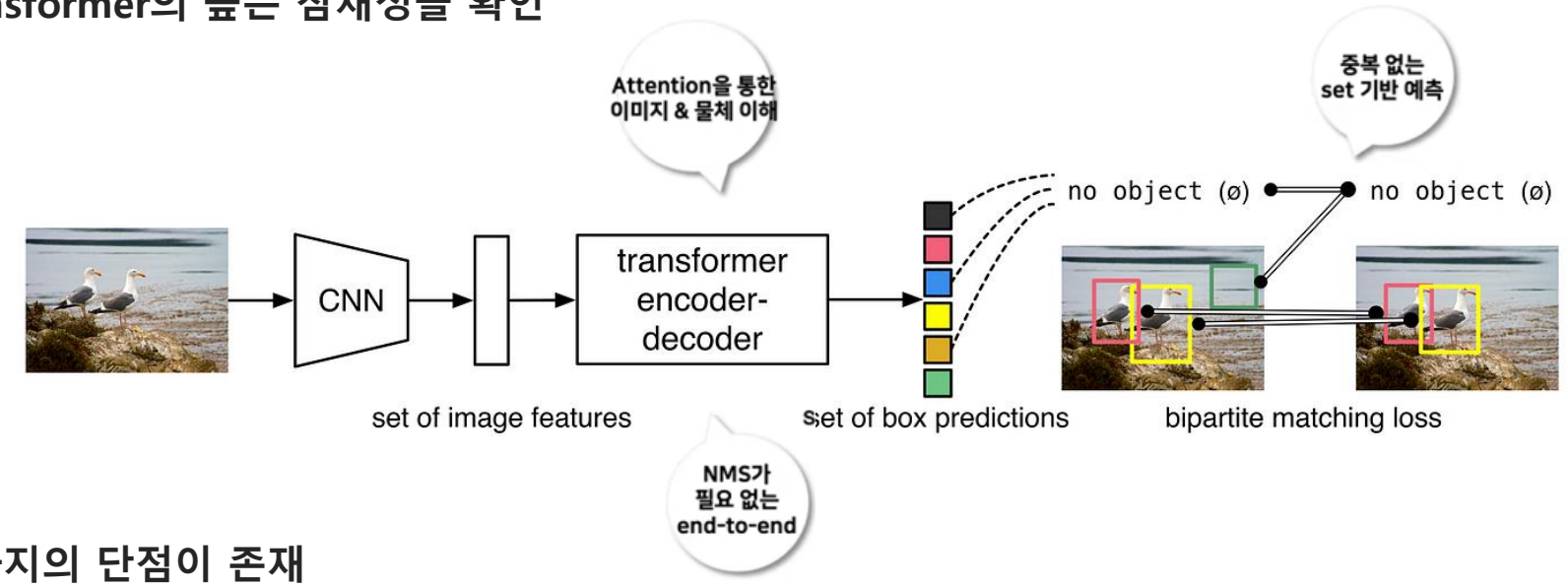
Conclusion

Conclusion

✓ Object detection을 direct set-prediction의 관점에서 새롭게 접근

✓ 트랜스포머의 구조와 더불어, RPN, NMS를 없앰으로써 굉장히 간결한 pipeline을 구축함

✓ 확장성이 뛰어나고 충분히 competitive한 성능을 얻어내어 object detection 및 vision 분야에서 transformer의 높은 잠재성을 확인



✓ 두 가지의 단점이 존재

1. 여러 크기의 anchor를 사용하지 않기 때문에 다양한 크기, 형태의 객체를 포착하지 못한다
2. 하나의 예측 bounding box를 ground truth에 matching 하기 때문에 converge 하는데 훨씬 긴 학습시간을 필요로 한다

하지만, DETR은 Faster R-CNN과 비슷한 성능을 보이면서도 post-processing을 필요로 하지 않는 점에서 큰 의의를 가진다!

QnA