

# | Paper Review |

## **DETRs Beat YOLOs on Real-time Object Detection**

Wenyu Lv, Yian Zhao, Shangliang Xu, Jinman Wei, Guanzhong Wang, Cheng Cui, Yuning Du, Qingqing Dang, Yi Liu

Baidu Inc.

CVPR 2024

Hyungseop Lee

Department of Embedded Systems Engineering, INU

# Outline

---

## 1. DETR

- 1) Direct set prediction
- 2) 문제점

## 2. Deformable DETR

- 1) Multi Scale Deformable attention module
- 2) 문제점

## 3. RT DETR

- 1) Hybrid Encoder (AIFI, CCFF)
- 2) Uncertainty-minimal Query Selection

# DETR이란?

---

- Object detection? object의 (bounding box, category) set을 prediction하는 것
- 이전의 object detector들의 성능은 post-processing(NMS), anchor set 설계 등 heuristic에 의해 크게 영향을 받음  
→ direct X Indirect O
- 위 문제를 해결하기 위해 DETR은 Transformer encoder-decoder architecture를 이용하여 direct set prediction approach를 제안하여 training pipeline을 간소화  
→ 즉, anchor or NMS와 같은 hand-designed 요소들을 제거

# DETR : Direct set prediction을 위한 주요 구성

---

1. Prediction과 Ground Truth box를 중복 없이 1:1 matching할 수 있도록 하는 loss  
→ Object detection set prediction loss
  
2. An architecture that predicts a set of object  
→ DETR architecture

# DETR : set prediction loss

- **Bipartite matching** : training 과정에서 predicted와 GT box를 1:1 matching  
→ 중복 prediction이 없도록

출력 개수 고정: **N = 6**

예측 결과

$(c_0 = \emptyset, b_0)$

$(c_1 = bird, b_1 = (180, 180, 150, 240))$

$(c_2 = \emptyset, b_2)$

$(c_3 = bird, b_3 = (120, 150, 100, 150))$

$(c_4 = \emptyset, b_4)$

$(c_5 = dog, b_5)$

$(c_0 = bird, b_0 = (122, 151, 100, 150))$

$(c_1 = bird, b_1 = (182, 180, 148, 238))$

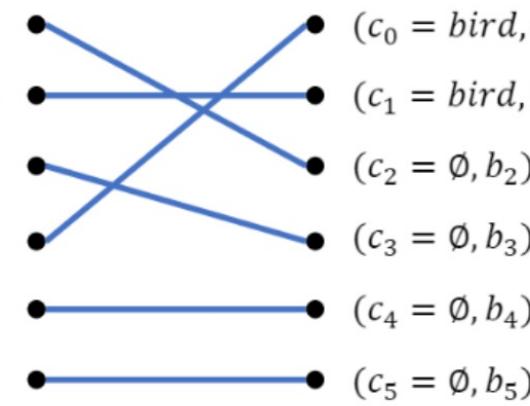
$(c_2 = \emptyset, b_2)$

$(c_3 = \emptyset, b_3)$

$(c_4 = \emptyset, b_4)$

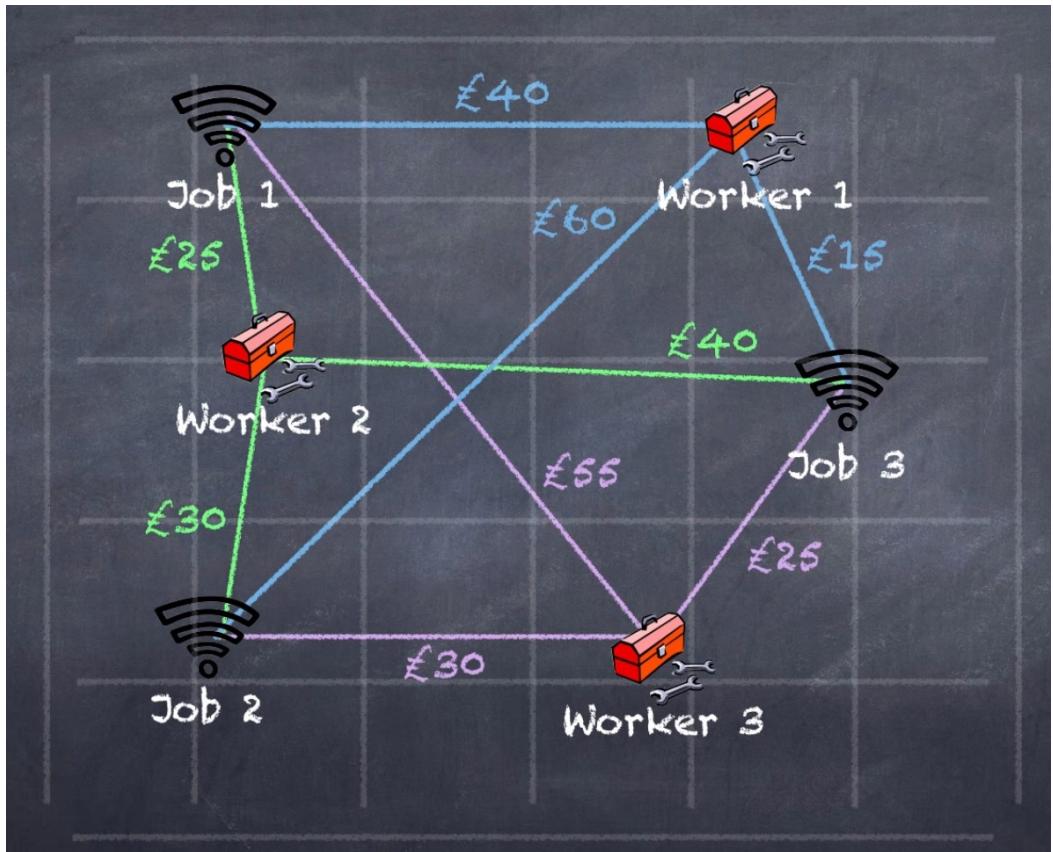
$(c_5 = \emptyset, b_5)$

실제 값



# DETR : set prediction loss

- Hungarian algorithm for Bipartite matching : optimal bipartite matching을 찾기 위해



	Job 1	Job 2	Job 3
Worker 1	40	60	15
Worker 2	25	30	45
Worker 3	55	30	25

Handwritten annotations on the right side of the matrix indicate total costs:

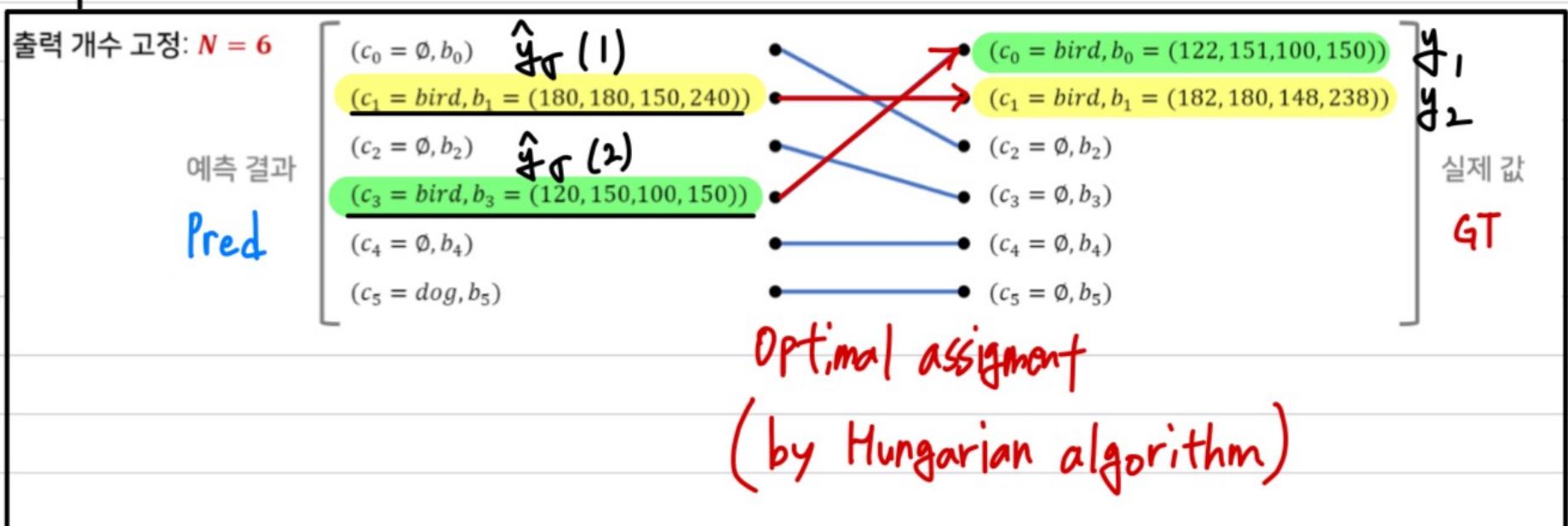
- £95 (top right)
- £115 (above the 15 cell)
- £110 (above the 45 cell)
- £70 (circled in green)
- £150 (below the 25 cell)
- £100 (below the 25 cell)

Optimal bipartite matching

# DETR : set prediction loss

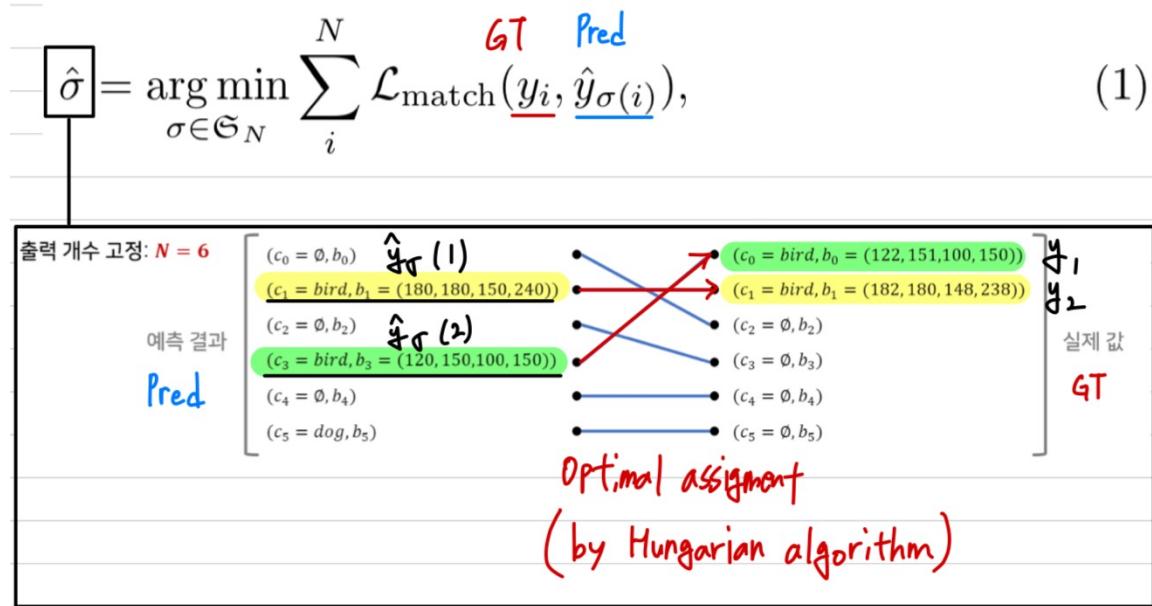
- $\hat{\sigma}$  : Optimal bipartite matching

$$\hat{\sigma} = \arg \min_{\sigma \in \mathfrak{S}_N} \sum_i^N \mathcal{L}_{\text{match}}(\underline{y_i}, \underline{\hat{y}_{\sigma(i)}}), \quad (1)$$



# DETR : set prediction loss

- $\mathcal{L}_{\text{Hungarian}}(y, \hat{y})$  : 학습에 사용될 loss (optimal bipartite matching에 대한 loss)



$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^N \left[ -\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}(i)}) \right], \quad (2)$$

# DETR : set prediction loss

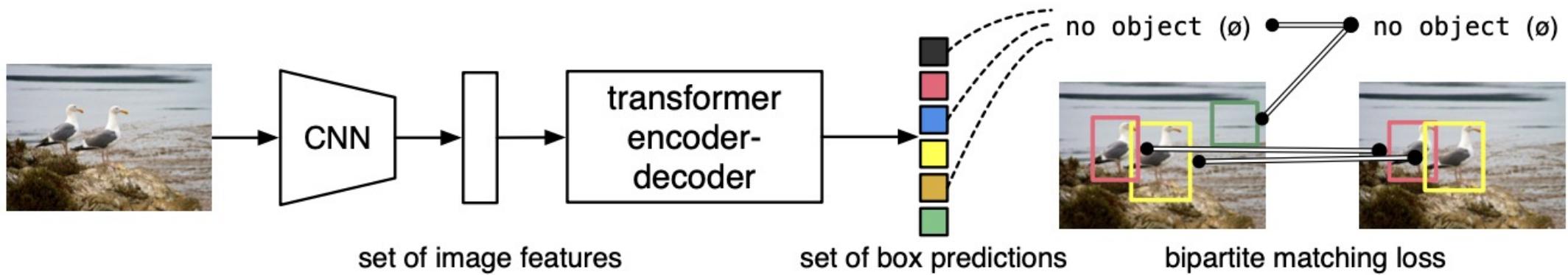


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a “no object” ( $\emptyset$ ) class prediction.

# DETR : architecture

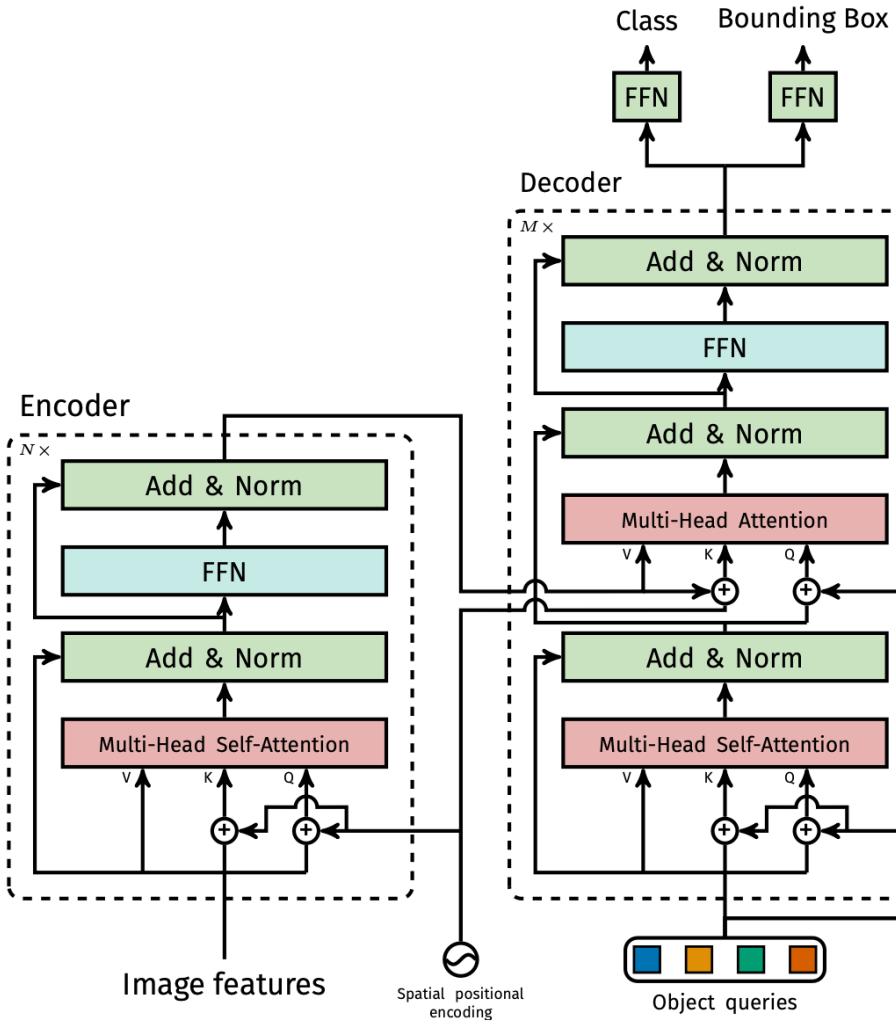
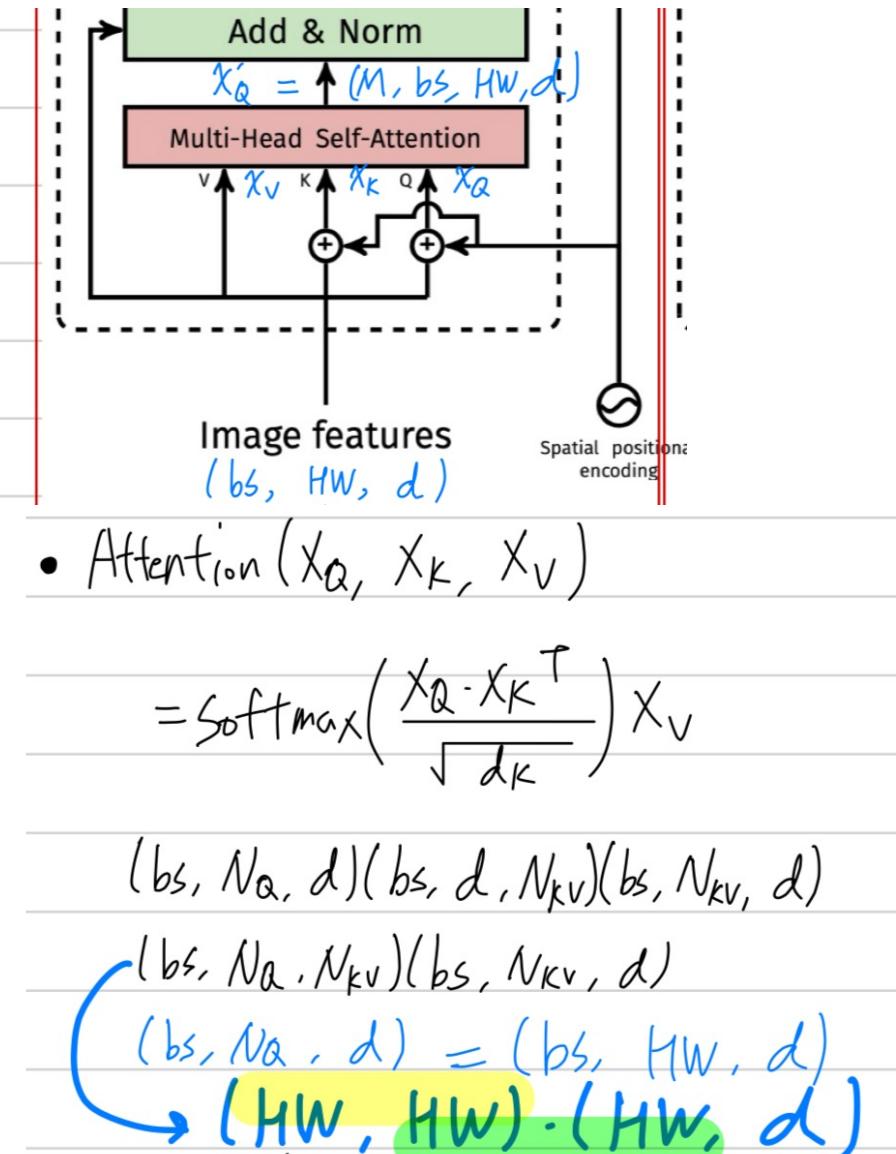
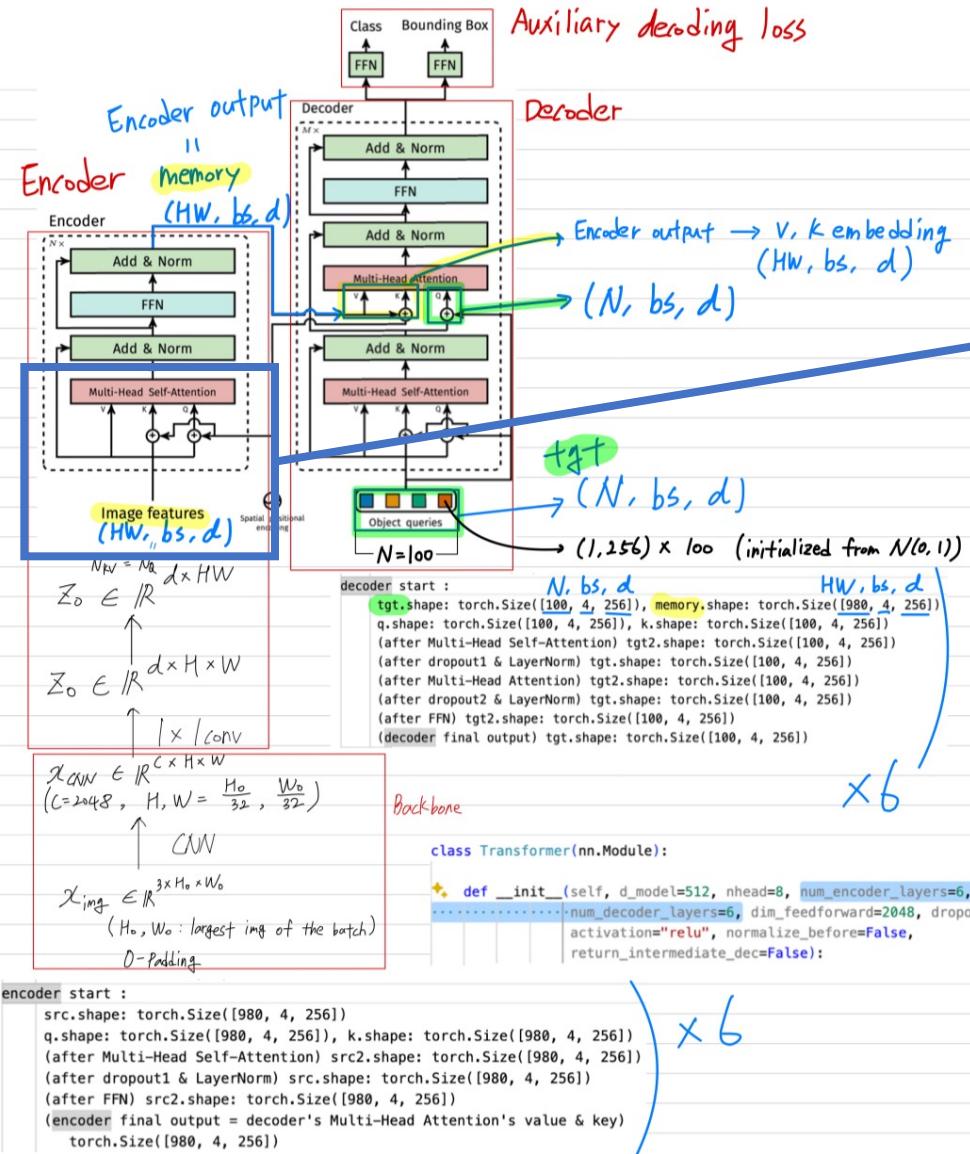
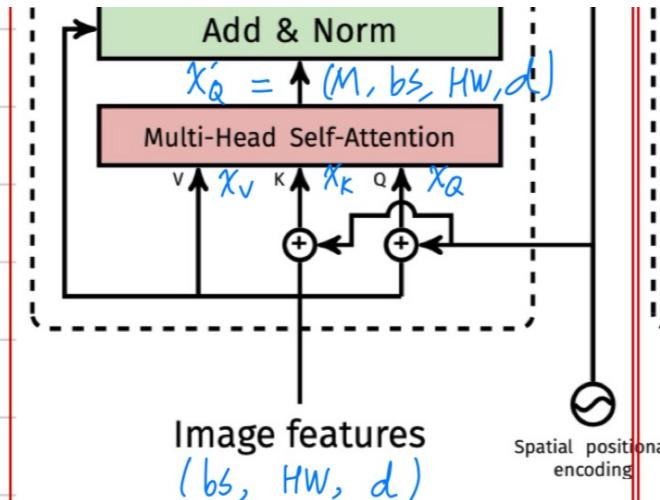


Fig. 10: Architecture of DETR's transformer. Please, see Section A.3 for details.

# DETR : architecture



# DETR : 문제점



- Attention ( $X_Q, X_K, X_V$ )

$$= \text{Softmax}\left(\frac{X_Q \cdot X_K^T}{\sqrt{d_K}}\right) X_V$$

$$(bs, N_Q, d)(bs, d, N_{KV})(bs, N_{KV}, d)$$

$$(bs, N_A, N_{KV})(bs, N_{KV}, d)$$

$$(bs, N_A, d) = (bs, HW, d)$$

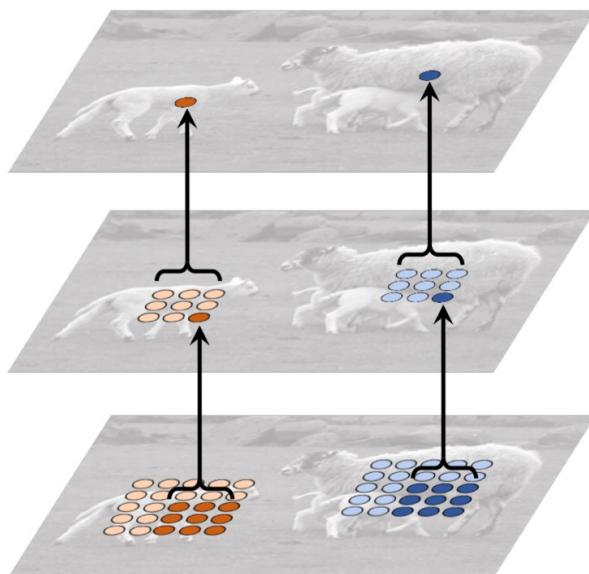
(HW, HW) · (HW, d)

- **문제점 1** : Attention module은 image feature map의 모든 pixel에 균일한 weight가 적용됨  
→ 긴 training epoch 소요

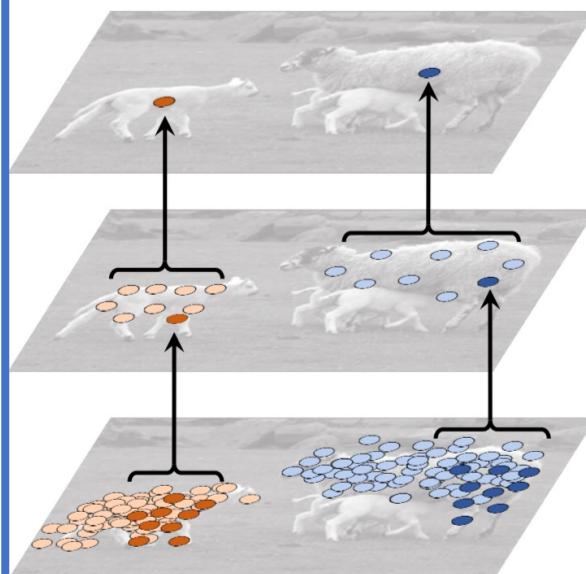
- **문제점 2** : low performance at detection small objects.  
→ 보통 high-resolution feature map으로부터 small object를 detection하기 위해 multi-scale features를 사용.  
→ 하지만 Attention computation에서 high-resolution feature maps을 처리하는 것은 매우 큰 **computational & memory complexities**

# Deformable DETR : deformable convolution

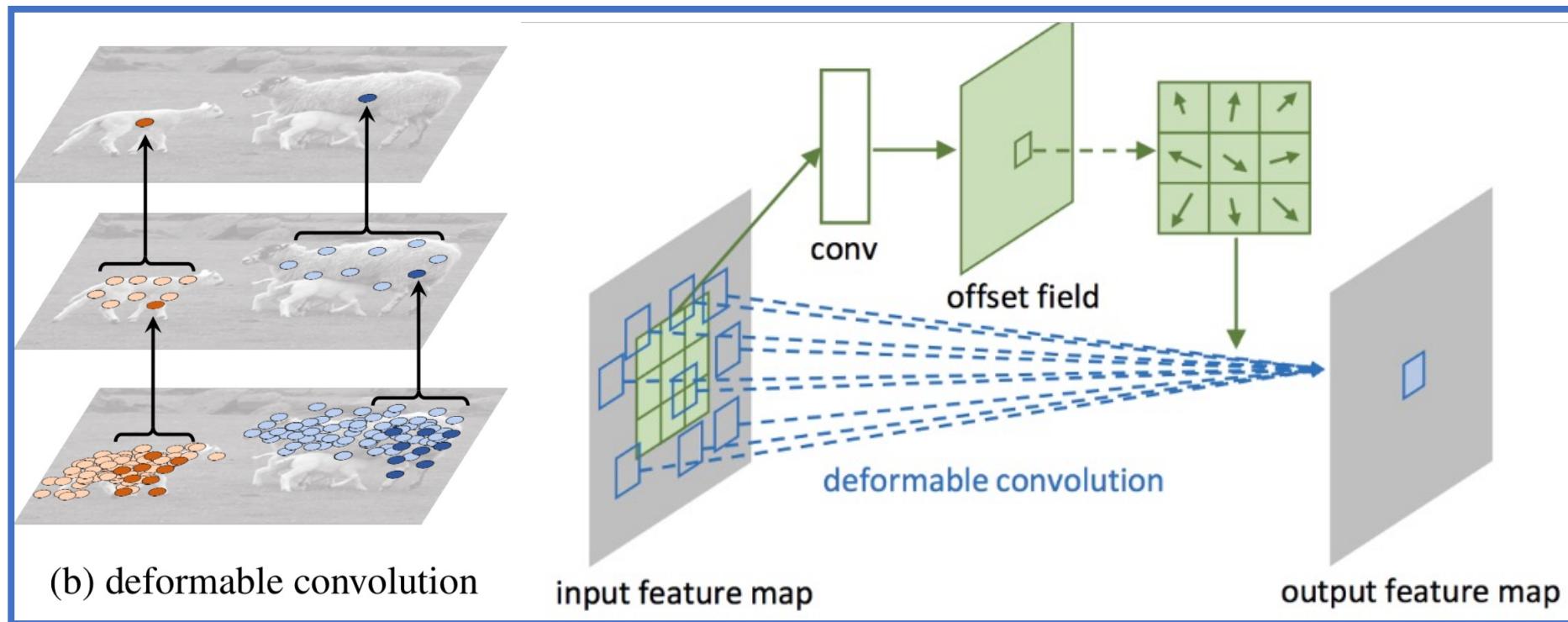
- **Deformable convolution** : sparse spatial location에 집중할 수 있는 효율적인 convolution mechanism



(a) standard convolution



(b) deformable convolution



# Deformable DETR : Multi scale deformable attention module

- **Deformable attention module** : Deformable DETR encoder의 하나의 layer

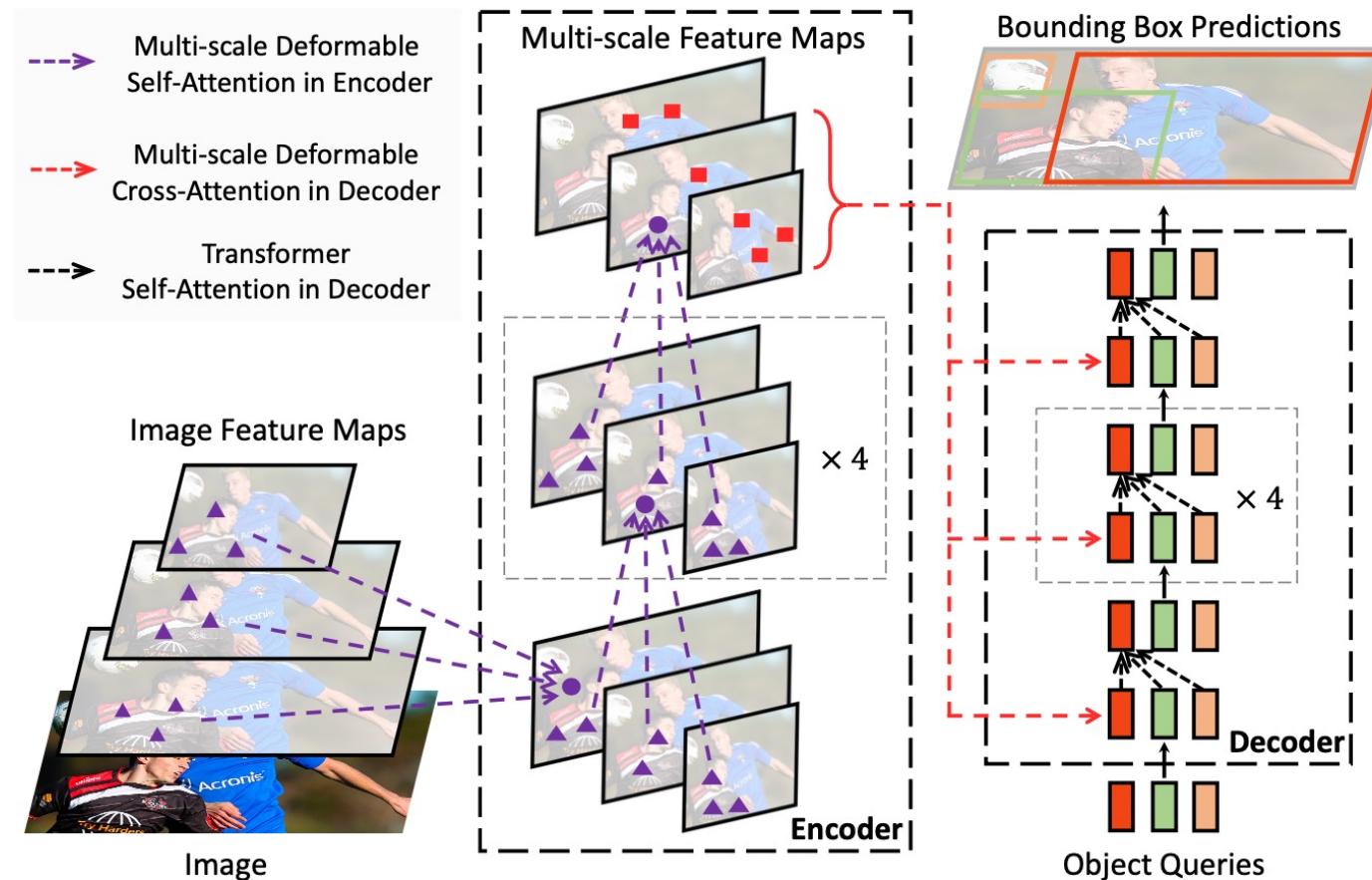


Figure 1: Illustration of the proposed Deformable DETR object detector.

# Deformable DETR : Multi scale deformable attention module

- $X$  : Backbone output feature map = Encoder input feature map

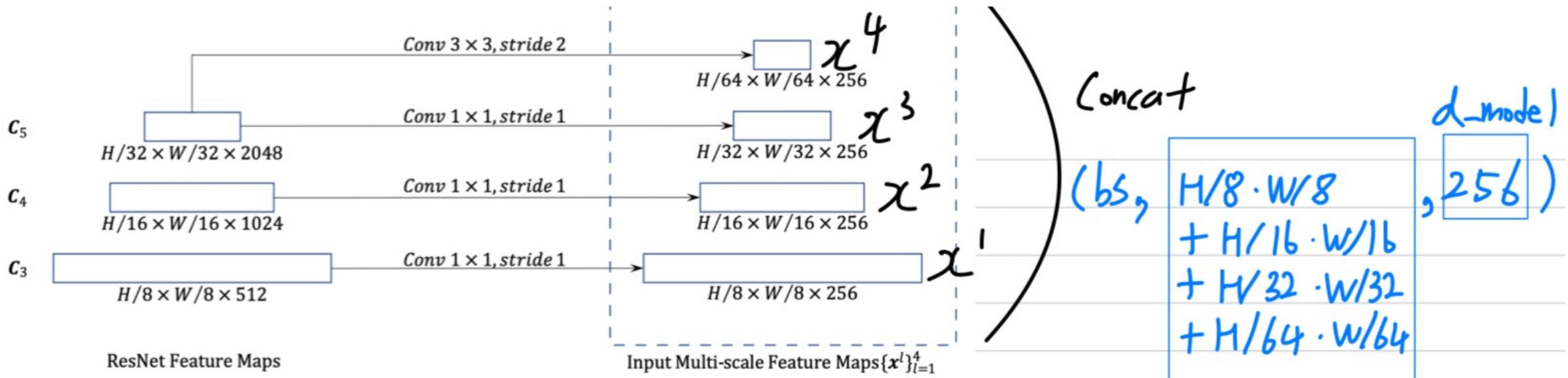
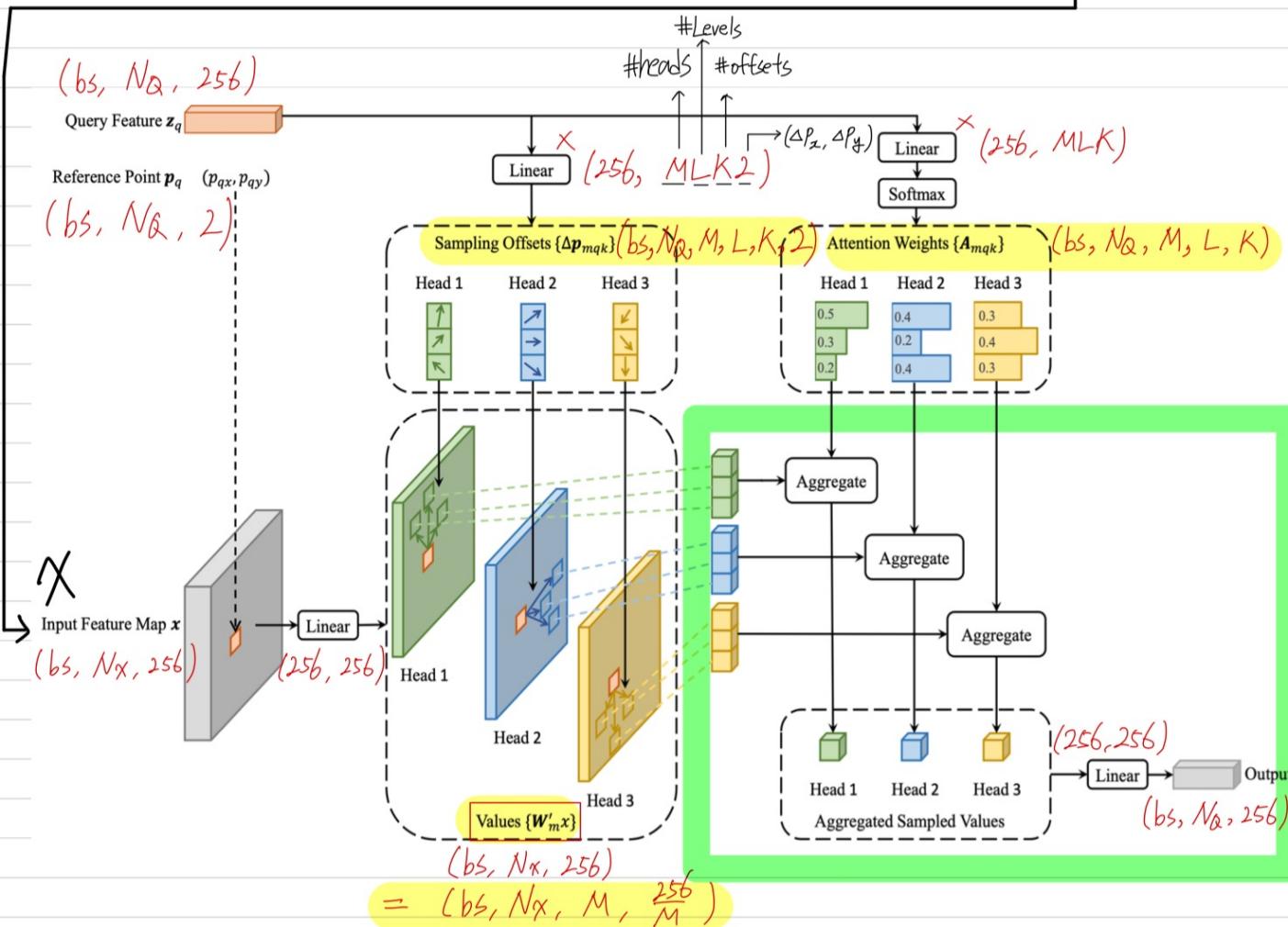


Figure 4: Constructing multi-scale feature maps for Deformable DETR.

# Deformable DETR : Multi scale deformable attention module

$\Rightarrow X(bs, Nx, 256)$



$$\text{MSDeformAttn}(z_q, \hat{p}_q, \{x^l\}_{l=1}^L) = \sum_{m=1}^M W_m \left[ \sum_{l=1}^L \sum_{k=1}^K A_{mlqk} \cdot W'_m x^l (\phi_l(\hat{p}_q) + \Delta p_{mlqk}) \right],$$

$(bs, NQ, 256)$

# Deformable DETR : Multi scale deformable attention module

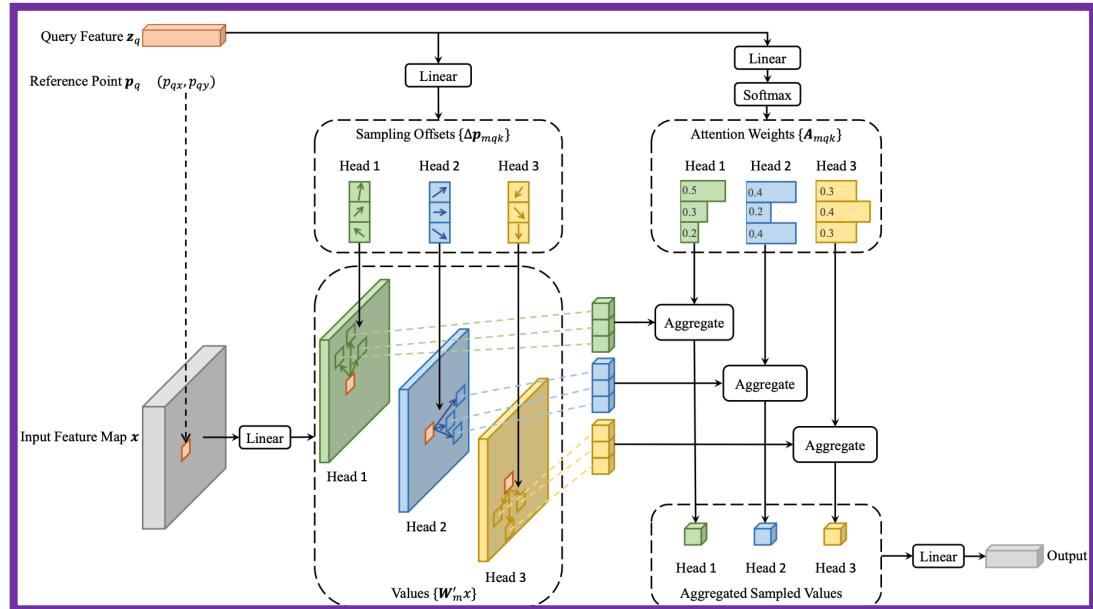
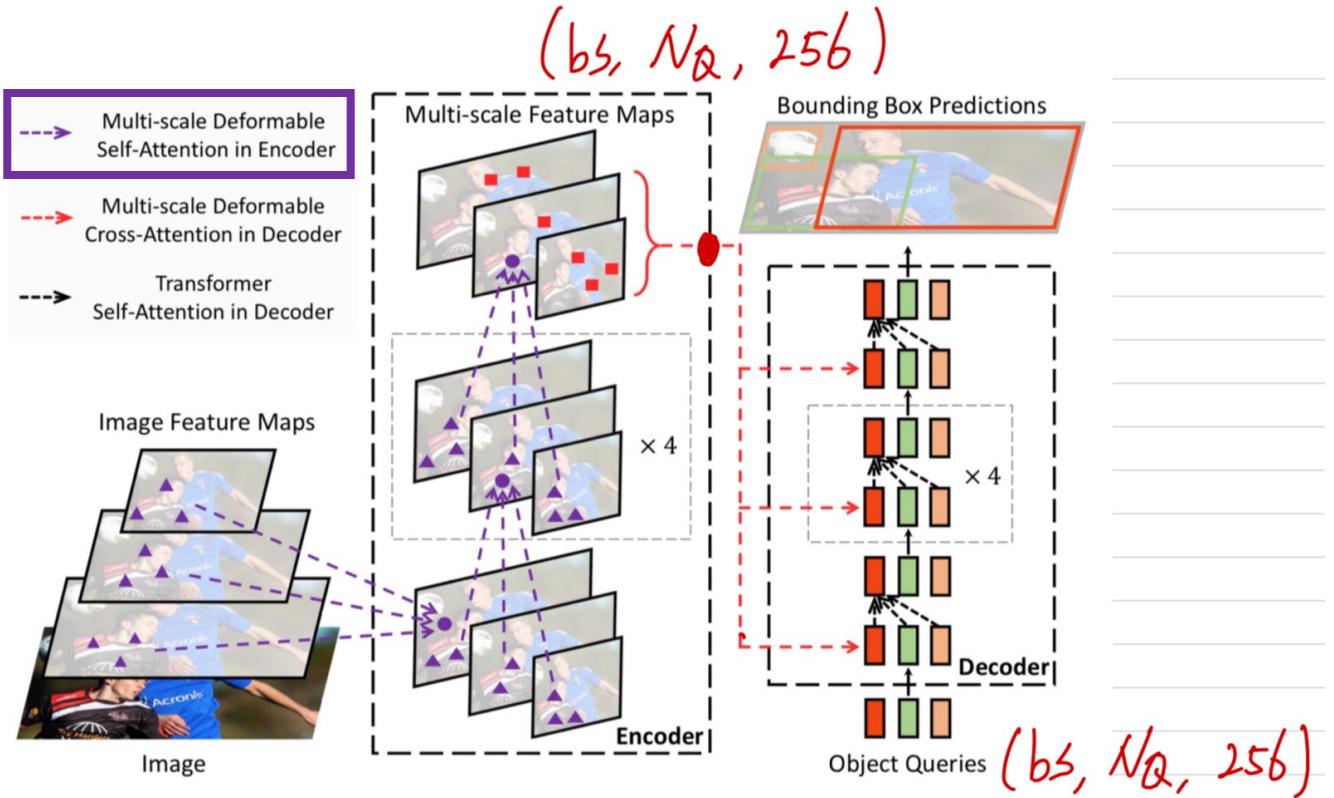


Figure 1: Illustration of the proposed Deformable DETR object detector.

# RT DETR : Efficient Hybrid Encoder

- 문제점 1 : 급격히 증가한 sequence length로 인해 encoder가 computational bottleneck이 되는 문제가 여전히 존재

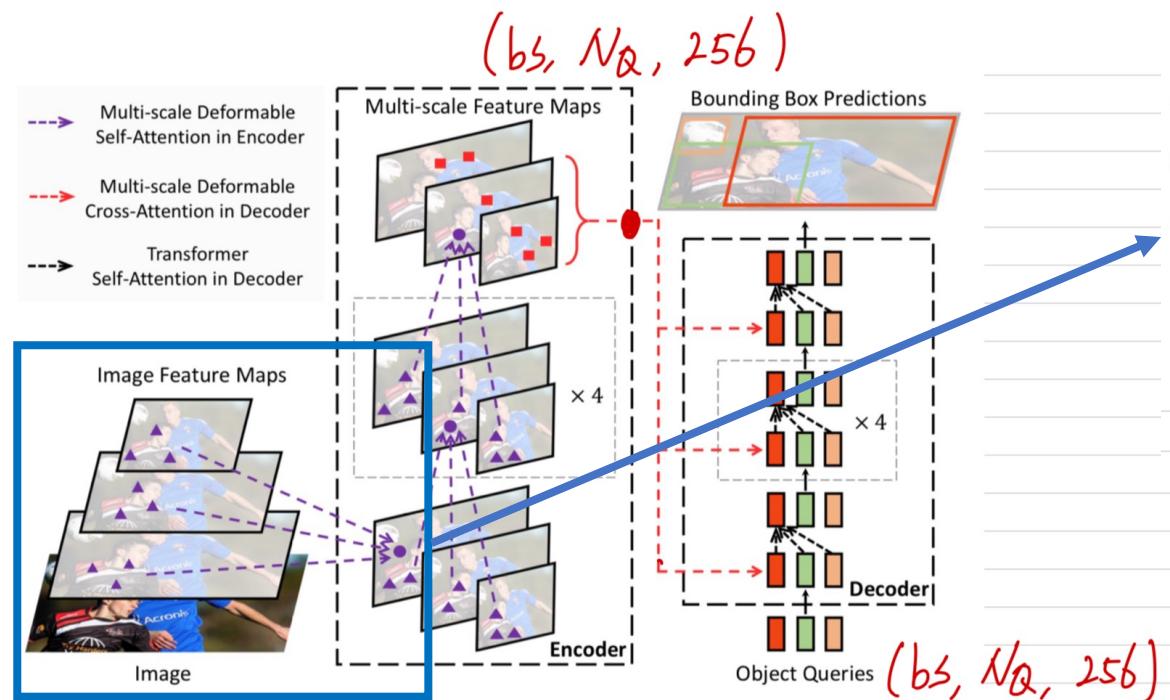


Figure 1: Illustration of the proposed Deformable DETR object detector.

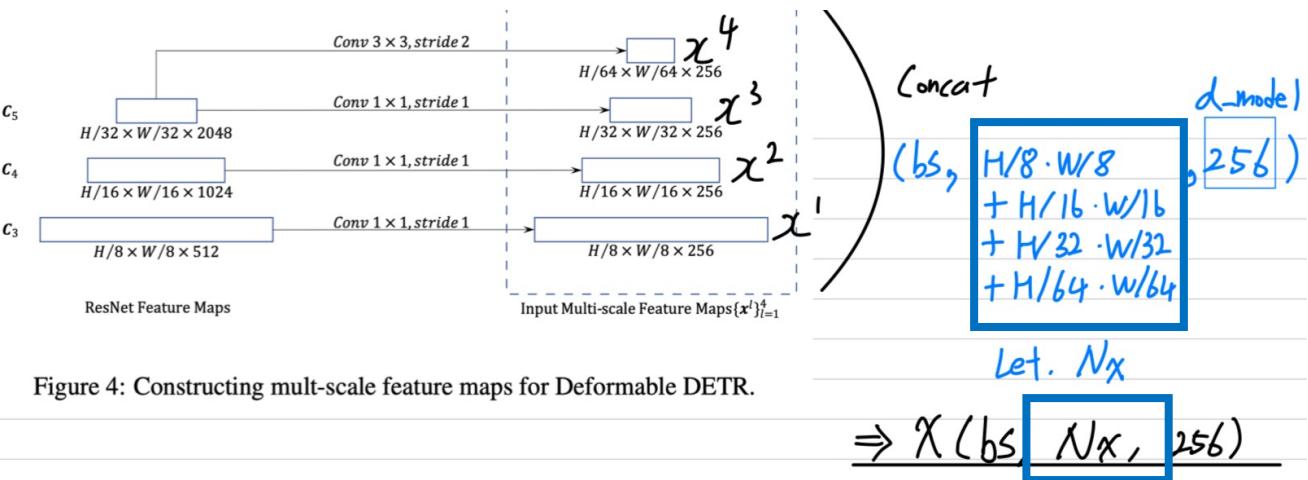


Figure 4: Constructing multi-scale feature maps for Deformable DETR.

# RT DETR : Efficient Hybrid Encoder

- 문제점 1 : object에 대한 풍부한 의미 정보를 포함하는 high-level features는 low-level features에서 추출되기 때문에, concat된 multi-scale features들에 대한 feature interaction은 redundant를 만듦  
→ “encoder input으로 high-level feature인  $x^3 (= x^3)$  feature만 사용해도 될 것이다”

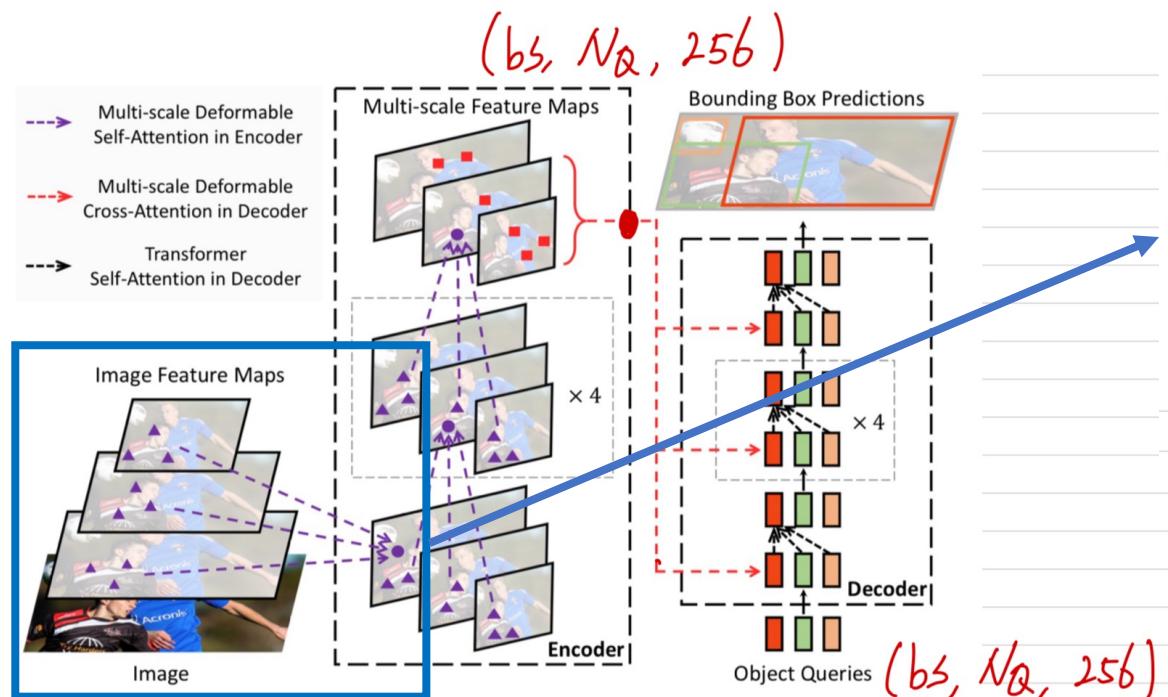


Figure 1: Illustration of the proposed Deformable DETR object detector.

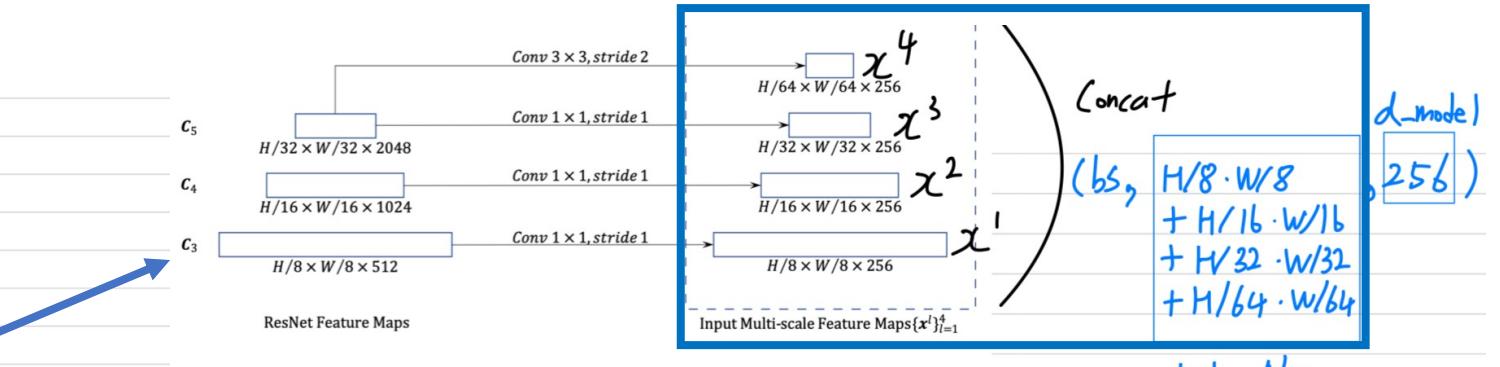


Figure 4: Constructing multi-scale feature maps for Deformable DETR.

$\Rightarrow x (bs, Nx, 256)$

# RT DETR : Efficient Hybrid Encoder

- 문제점 1 : “encoder input으로 high-level feature인  $S_5 (= x^3)$  feature만 사용해도 될 것이다”

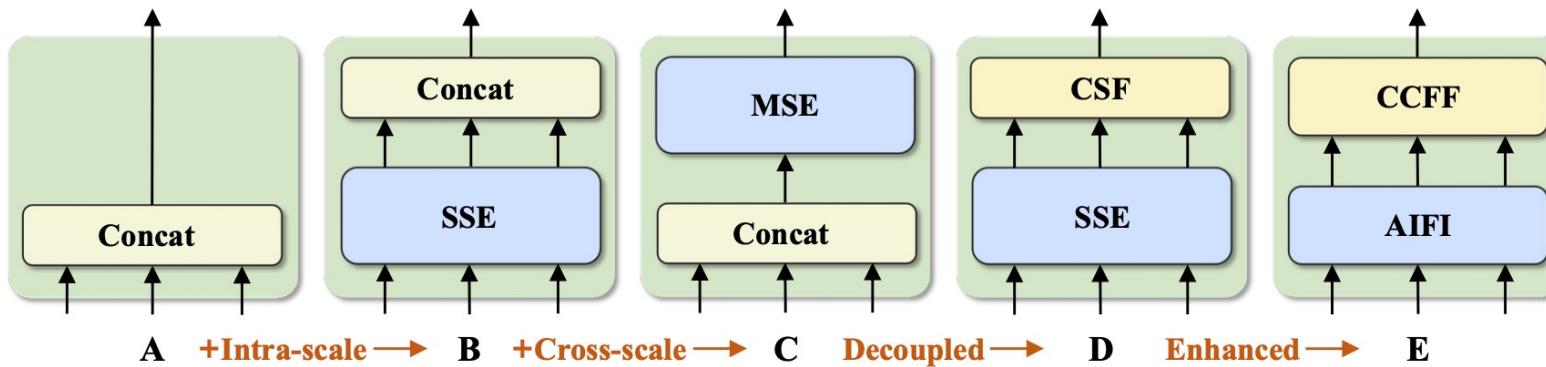


Figure 3. The encoder structure for each variant. SSE represents the single-scale Transformer encoder, MSE represents the multi-scale Transformer encoder, and CSF represents cross-scale fusion. AIFI and CCFF are the two modules designed into our hybrid encoder.

Variant	AP (%)	#Params (M)	Latency (ms)
A	43.0	31	7.2
B	44.9	32	11.1
C	45.6	32	13.3
D	46.4	35	12.2
$D_{S_5}$	46.8	35	7.9
E	47.9	42	9.3

*C* : multi-scale features들을 Concat 후 encoder  
*D<sub>S<sub>5</sub></sub>* :  $S_5$  single-scale encoder + cross-scale fusion(PANet)  
*E* :  $D_{S_5}$  + 발전된 efficient hybrid encoder (CCFF + AIFI)

Table 3. The indicators of the set of variants illustrated in Figure 3.

# RT DETR : Efficient Hybrid Encoder

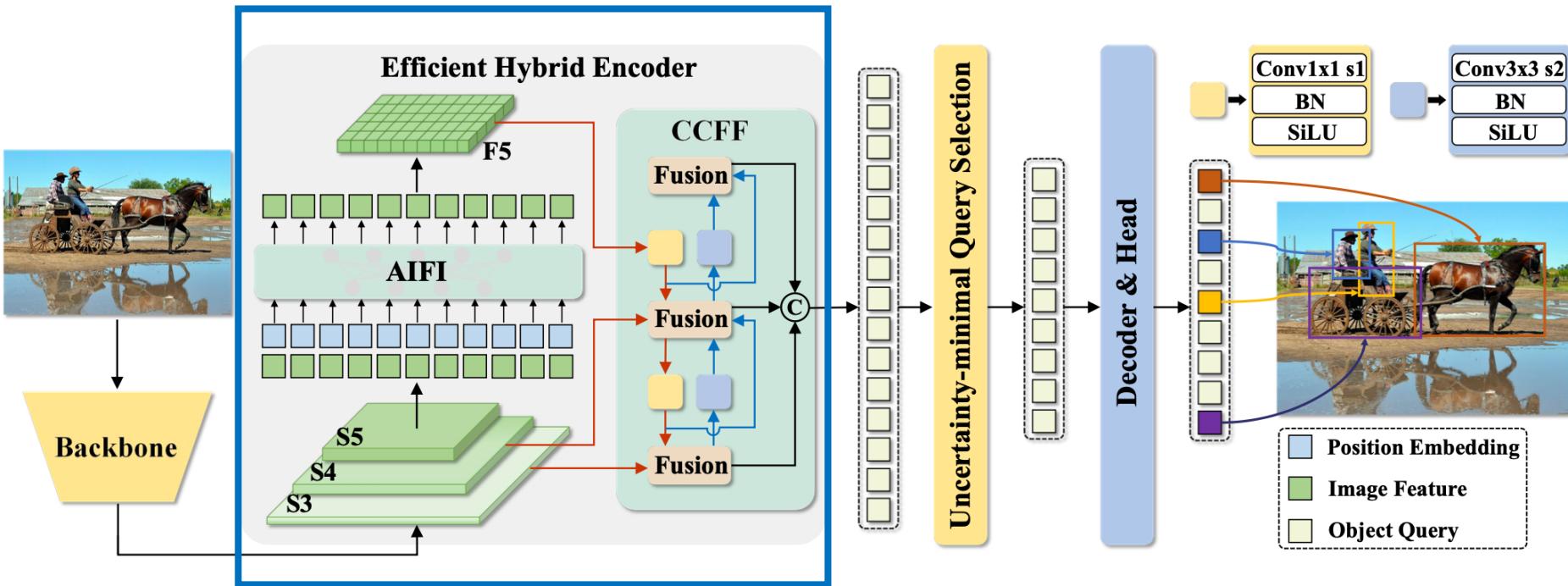
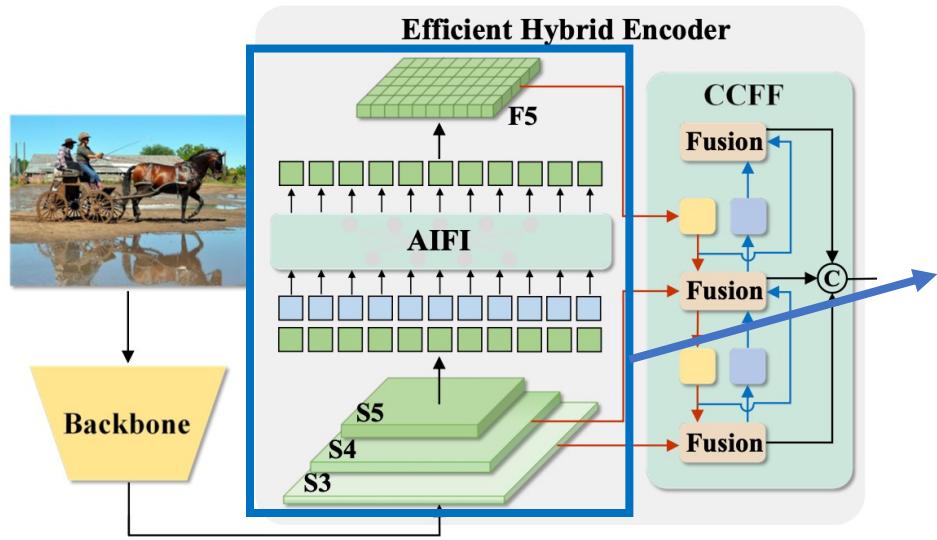


Figure 4. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale Feature Fusion (CCFF). Then, the uncertainty-minimal query selection selects a fixed number of encoder features to serve as initial object queries for the decoder. Finally, the decoder with auxiliary prediction heads iteratively optimizes object queries to generate categories and boxes.

# RT DETR : Efficient Hybrid Encoder

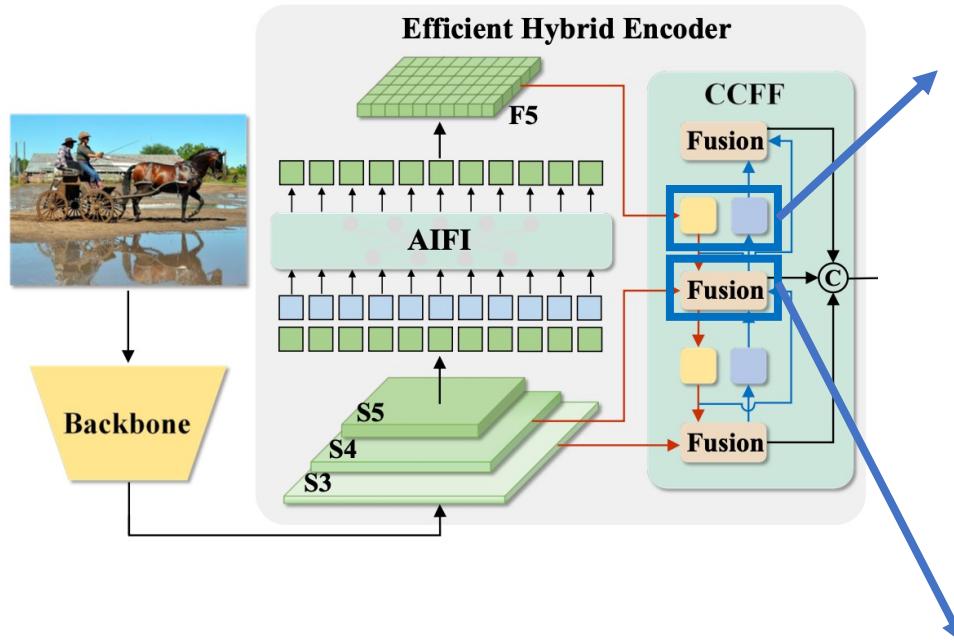


- **AIFI(Attention-based Intra-scale Feature Interaction)**

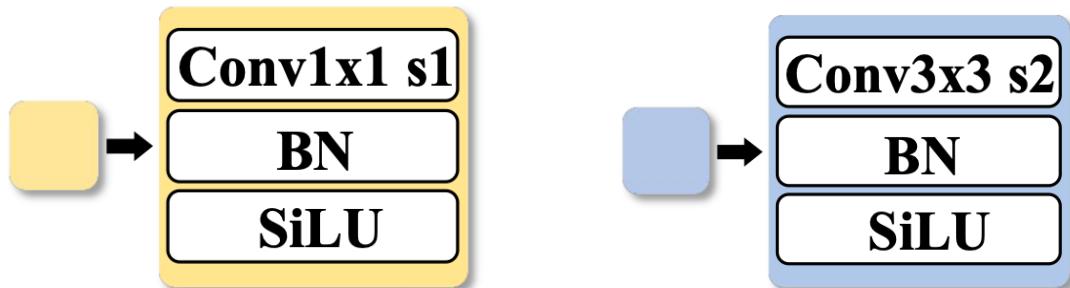
: S5에 대해서만 self-attention 적용 → F5

# RT DETR : Efficient Hybrid Encoder

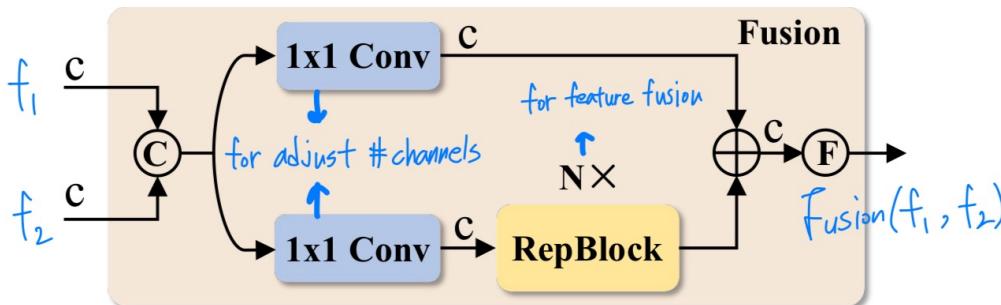
- CCFF(Convolution-based Cross Feature Fusion)



- 두 feature의 height, width를 맞춰주기 위해  
upsampling      downsampling

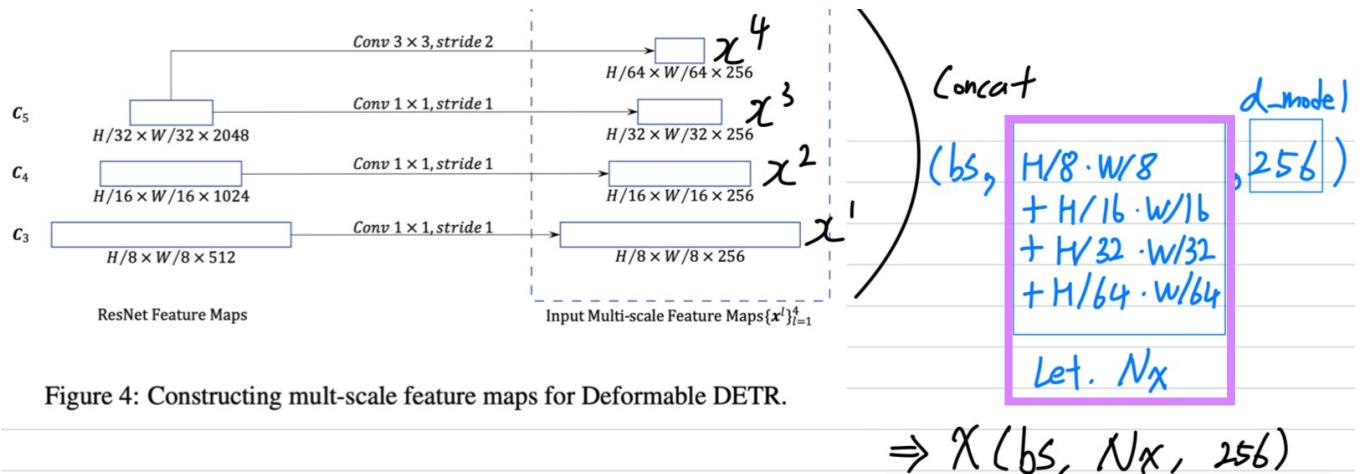
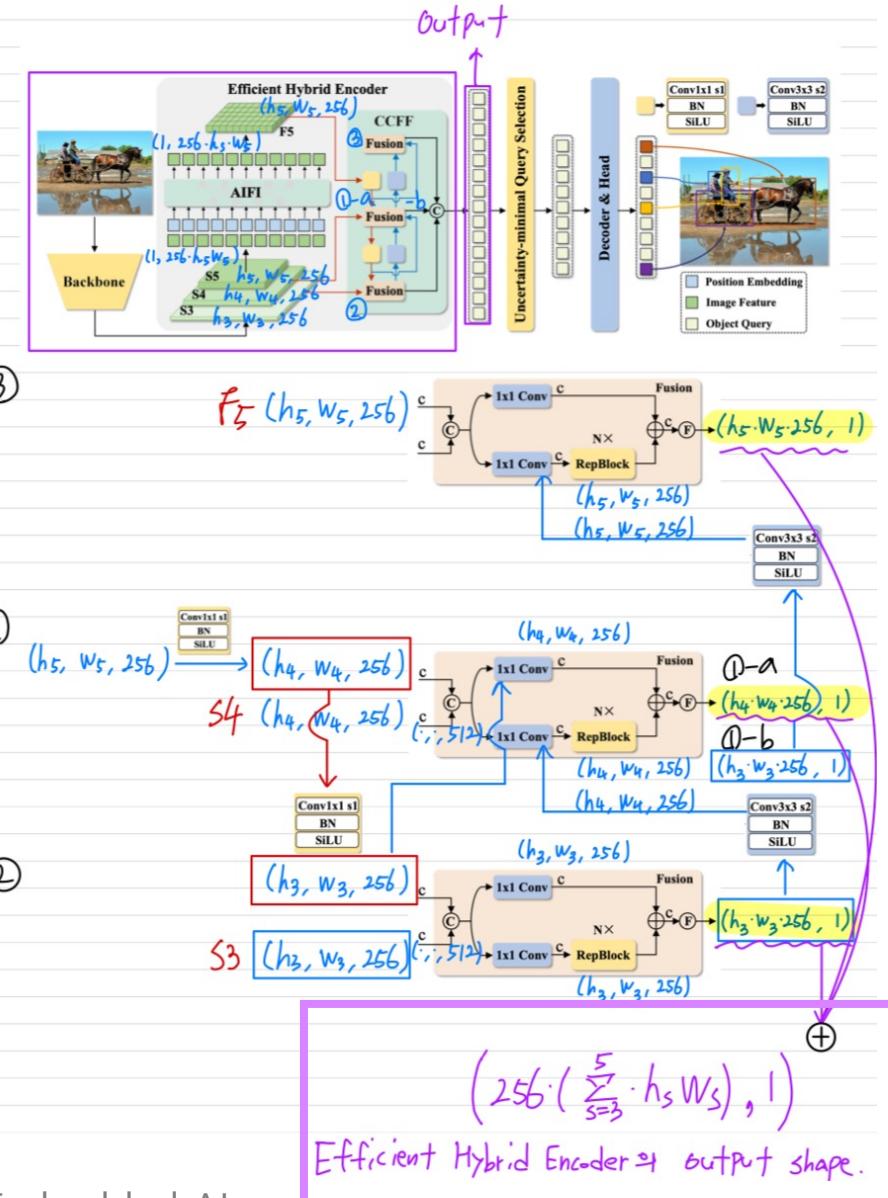


- 서로 다른 두 feature를 fusion



© Concatenate   ⊕ Element-wise add   F Flatten

# RT DETR : Efficient Hybrid Encoder

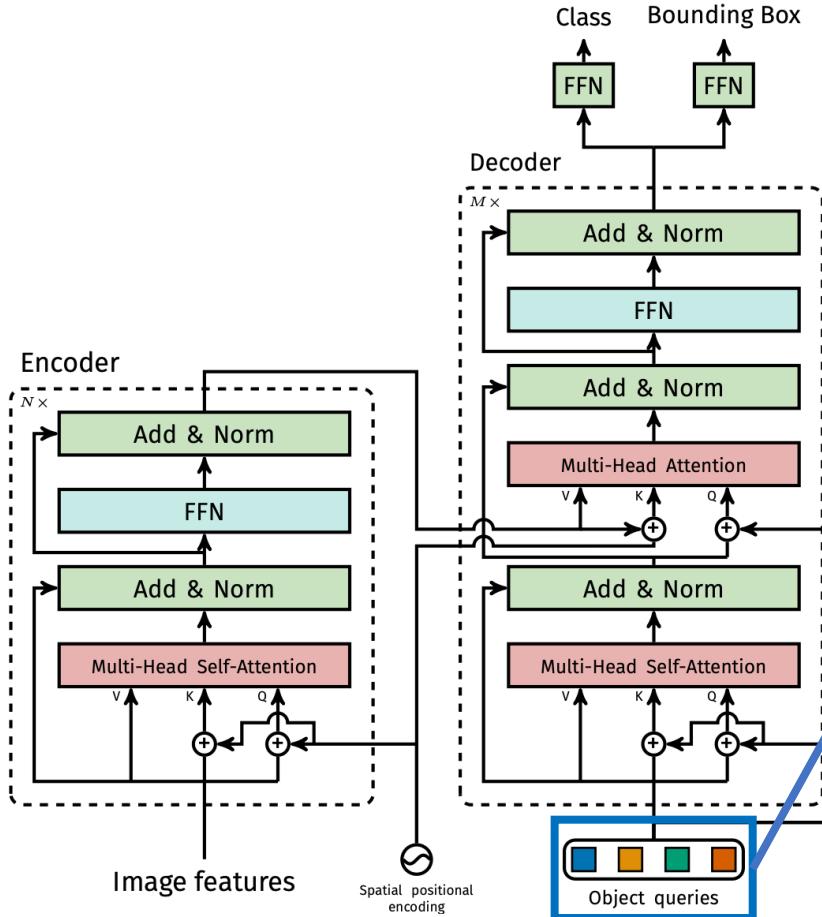


- Deformable DETR query length :  $\sum_{s=3}^5 H_s W_s$
  - RT DETR query length :  $H_5 W_5$

→ Encoder의 computational bottleneck을 완화

# DETR : 문제점

- 문제점 2 : object queries initialization with Uncertainty



```
tgt = torch.zeros_like(bs, NQ, 256)
```

```
tgt = torch.zeros_like(query_embed) You, 4주 전 • 04_DETR clone 1
memory = self.encoder(src, src_key_padding_mask=mask, pos=pos_embed)
print(f"tgt : {tgt}, tgt.shape: {tgt.shape}")
hs = self.decoder(tgt, memory, memory_key_padding_mask=mask,
| | | | | pos=pos_embed, query_pos=query_embed)

decoder start :
tgt: tensor([[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],

[[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],

[[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.]],

...,

[[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],

[[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],

[[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.]], device='cuda:0')
```

Fig. 10: Architecture of DETR’s transformer. Please, see Section A.3 for details.

# RT DETR : Uncertainty-minimal Query Selection

- “Decoder에게 high-quality queries를 제공한다면, training convergence 속도가 빨라질 것이다”

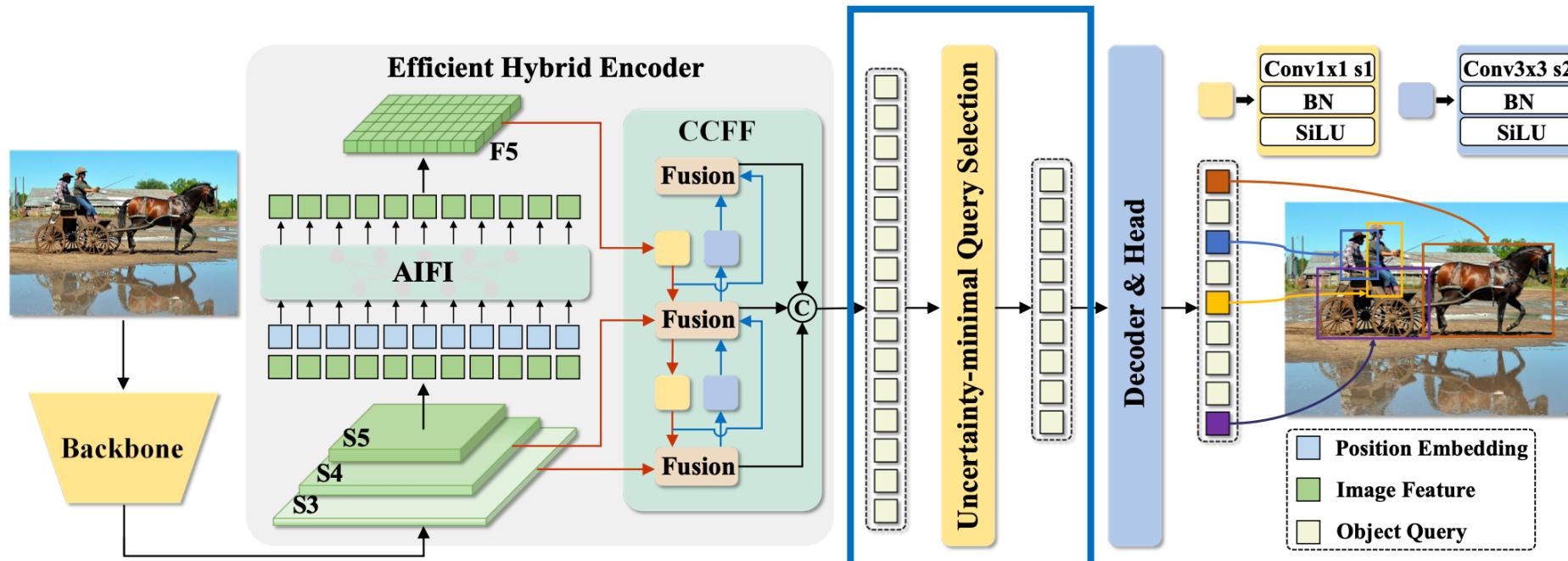


Figure 4. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale Feature Fusion (CCFF). Then, the uncertainty-minimal query selection selects a fixed number of encoder features to serve as initial object queries for the decoder. Finally, the decoder with auxiliary prediction heads iteratively optimizes object queries to generate categories and boxes.

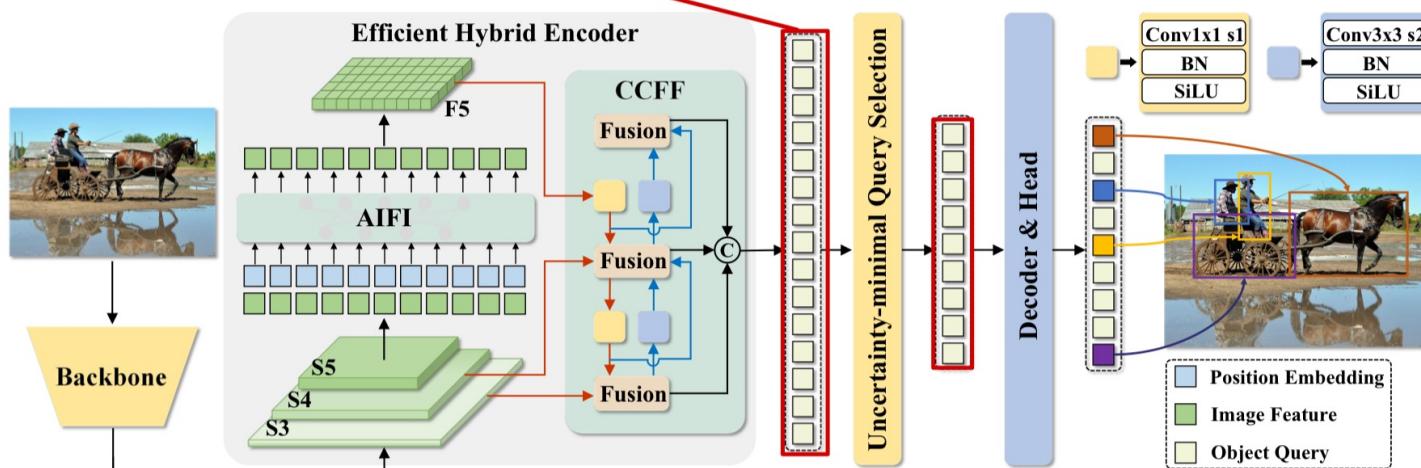
# RT DETR : Uncertainty-minimal Query Selection

- Encoder의 output feature map ( $bs, N_X, 256$ )

→ confidence score top 300개의 index 추출

→  $(bs, 300, 256)$

$$(bs, N_X, 256) \xrightarrow{\text{Top 300}} (bs, 300, 256)$$



```
def _get_decoder_input(self,
                      memory,
                      spatial_shapes, # [[4H, 4W], [2H, 2W], [H, W]]
                      denoising_class=None,
                      denoising_bbox_unact=None):
    ...
    # memory : encoder output feature map
    bs, _, _ = memory.shape
    ...
    N_X = 4H*4W + 2H*2W + H*W
    memory : encoder output
    (bs, N_X, 256)
    ...

    # prepare input for decoder
    if self.training or self.eval_spatial_size is None:
        anchors, valid_mask = self._generate_anchors(spatial_shapes, device=memory.device)
    ...
        anchors : (1, N_X, 4)
        valid_mask : (1, N_X, 1) -> False or True
    ...
    else:
        anchors, valid_mask = self.anchors.to(memory.device), self.valid_mask.to(memory.device)

    # memory = torch.where(valid_mask, memory, 0)
    memory = valid_mask.to(memory.dtype) * memory # TODO fix type error for onnx export

    output_memory = self.enc_output(memory)
    ...
    output_memory : (bs, N_X, 256) * (256, 256)
    (bs, N_X, 256)
    ...

    enc_outputs_class = self.enc_score_head(output_memory)
    enc_outputs_coord_unact = self.enc_bbox_head(output_memory) + anchors
    ...
    enc_outputs_class : (bs, N_X, 80)
    enc_outputs_coord_unact : (bs, N_X, 4)
    ...

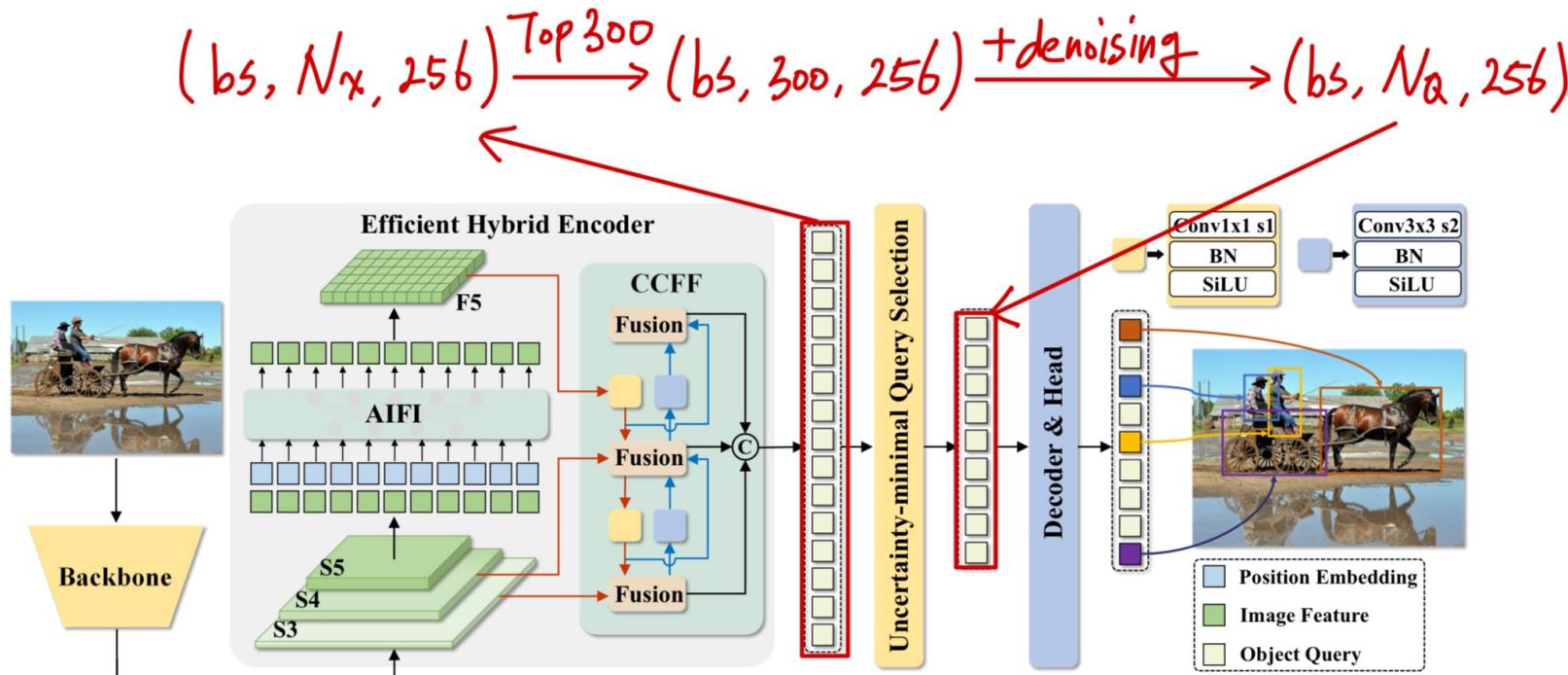
    , topk_ind = torch.topk(enc_outputs_class.max(-1).values, self.num_queries, dim=1)
    ...
    topk_ind : (bs, num_queries=300)
    ...
```

# RT DETR : Uncertainty-minimal Query Selection

- Model이 noise에 잘 대응할 수 있도록 denoising query  $N_{denoising}$  추가

→  $(bs, 300 + N_{denoising}, 256)$

→  $(bs, N_Q, 256)$

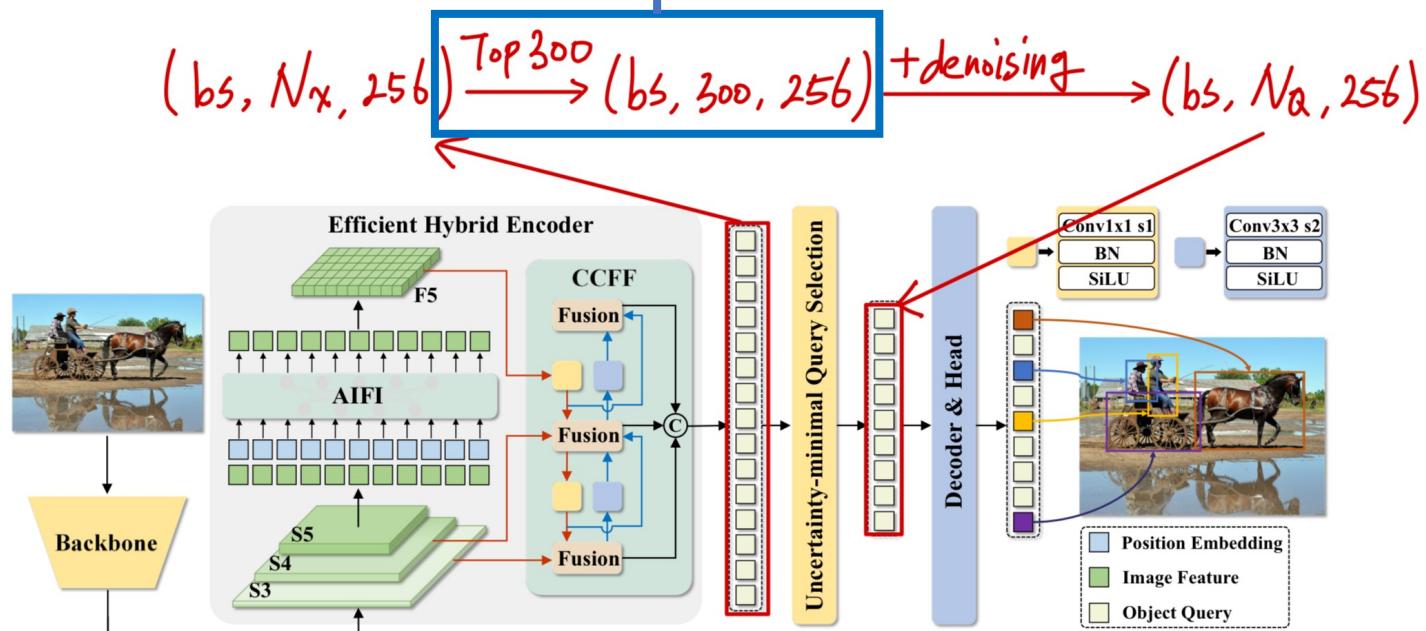


# RT DETR : Uncertainty-minimal Query Selection

- Encoder의 output feature map의 uncertainty를 최소화할 수 있도록 topK에 대한 loss를 추가

$$\mathcal{U}(\hat{\mathcal{X}}) = \underbrace{\text{feature uncertainty}}_{\mathcal{U}(\hat{\mathcal{X}})} + \underbrace{\text{localization}}_{\|\mathcal{P}(\hat{\mathcal{X}}) - \mathcal{C}(\hat{\mathcal{X}})\|}, \hat{\mathcal{X}} \in \mathbb{R}^D \quad (2)$$

$$\mathcal{L}(\hat{\mathcal{X}}, \hat{\mathcal{Y}}, \mathcal{Y}) = \mathcal{L}_{box}(\hat{\mathbf{b}}, \mathbf{b}) + \mathcal{L}_{cls}(\mathcal{U}(\hat{\mathcal{X}}), \hat{\mathbf{c}}, \mathbf{c}) \quad (3)$$



# RT DETR : Uncertainty-minimal Query Selection

- Encoder의 output feature map의 uncertainty를 최소화할 수 있도록 topK에 대한 loss를 추가

```
class RTDETRTransformer(nn.Module):
    def forward(self, feats, targets=None):
        target, init_ref_points_unact, enc_topk_bboxes, enc_topk_logits = \
            self._get_decoder_input(memory, spatial_shapes, denoising_class, denoising_bbox_unact)

        # decoder
        out_bboxes, out_logits = self.decoder(
            target,
            init_ref_points_unact,
            memory,
            spatial_shapes,
            level_start_index,
            self.dec_bbox_head,
            self.dec_score_head,
            self.query_pos_head,
            attn_mask=attn_mask)

        if self.training and dn_meta is not None:
            dn_out_bboxes, out_bboxes = torch.split(out_bboxes, dn_meta['dn_num_split'], dim=2)
            dn_out_logits, out_logits = torch.split(out_logits, dn_meta['dn_num_split'], dim=2)

        # 6번째 decoder layer의 output을 최종 output으로 사용
        out = {'pred_logits': out_logits[-1], 'pred_boxes': out_bboxes[-1]}

        if self.training and self.aux_loss:
            out['aux_outputs'] = self._set_aux_loss(out_logits[:-1], out_bboxes[:-1])
            # -> aux_outputs[0] ~ aux_outputs[4] : 5개의 decoder layer에 대한 aux loss
            # -> aux_outputs[5] : encoder output feature map에 대한 aux loss
            # enc_topk_logits -> loss_map['vfl'] -> Varifocal loss
            # enc_topk_bboxes -> loss_map['boxes'] -> L1 loss + GIoU loss

            if self.training and dn_meta is not None:
                out['dn_aux_outputs'] = self._set_aux_loss(dn_out_logits, dn_out_bboxes)
                out['dn_meta'] = dn_meta

        return out
```

```
    # In case of cardinality losses, we expect that process them output of each intermediate layer
    if 'aux_outputs' in outputs:      You, 2분 전 • Uncommitted changes
        for i, aux_outputs in enumerate(outputs['aux_outputs']):
            indices = self.matcher(aux_outputs, targets)
            for loss in self.losses:
                if loss == 'masks':
                    # Intermediate masks losses are too costly to compute, we ignore them.
                    continue
                kwargs = {}
                if loss == 'labels':
                    # Logging is enabled only for the last layer
                    kwargs = {'log': False}

                l_dict = self.get_loss(loss, aux_outputs, targets, indices, num_boxes, **kwargs)
                l_dict = {k: l_dict[k] * self.weight_dict[k] for k in l_dict if k in self.weight_dict}
                l_dict = {k + f'_aux_{i}': v for k, v in l_dict.items()}
                losses.update(l_dict)

    def get_loss(self, loss, outputs, targets, indices, num_boxes, **kwargs):
        loss_map = {
            'labels': self.loss_labels,
            'cardinality': self.loss_cardinality,
            'boxes': self.loss_boxes,
            'masks': self.loss_masks,

            'bce': self.loss_labels_bce,
            'focal': self.loss_labels_focal,
            'vfl': self.loss_labels_vfl,
        }
        assert loss in loss_map, f'do you really want to compute {loss} loss?'
        return loss_map[loss](outputs, targets, indices, num_boxes, **kwargs)
```

# RT DETR : Architecture

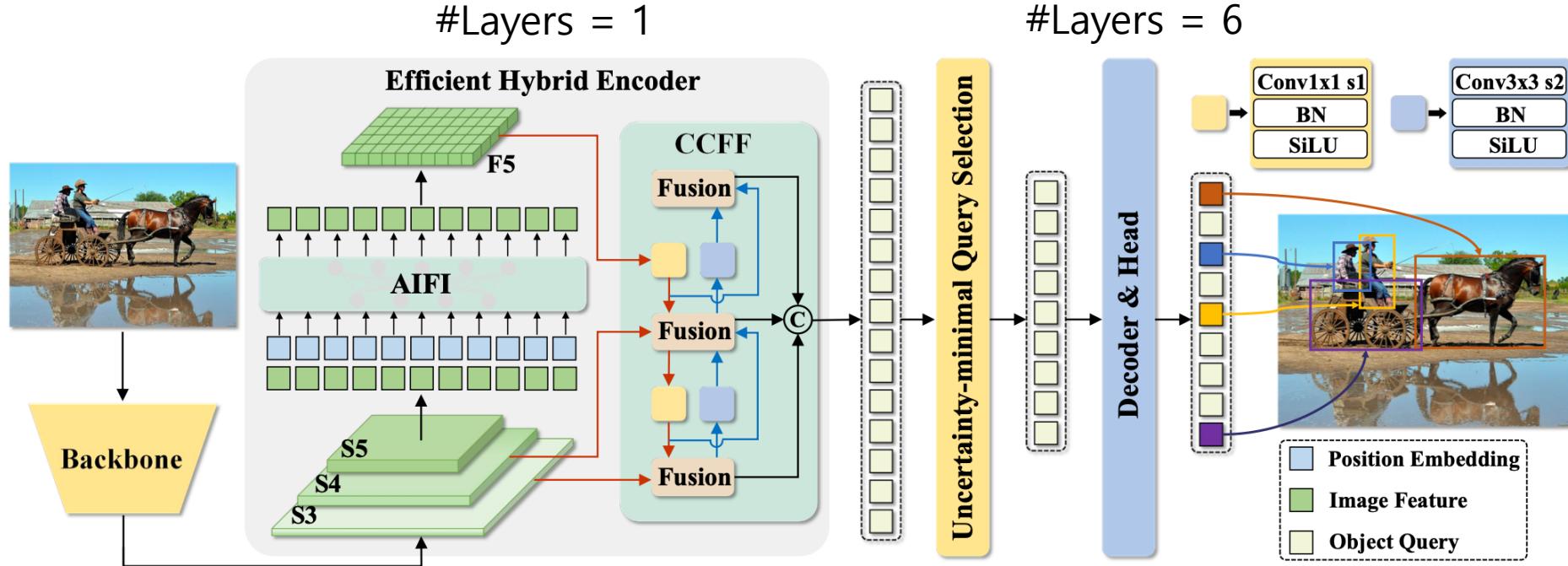
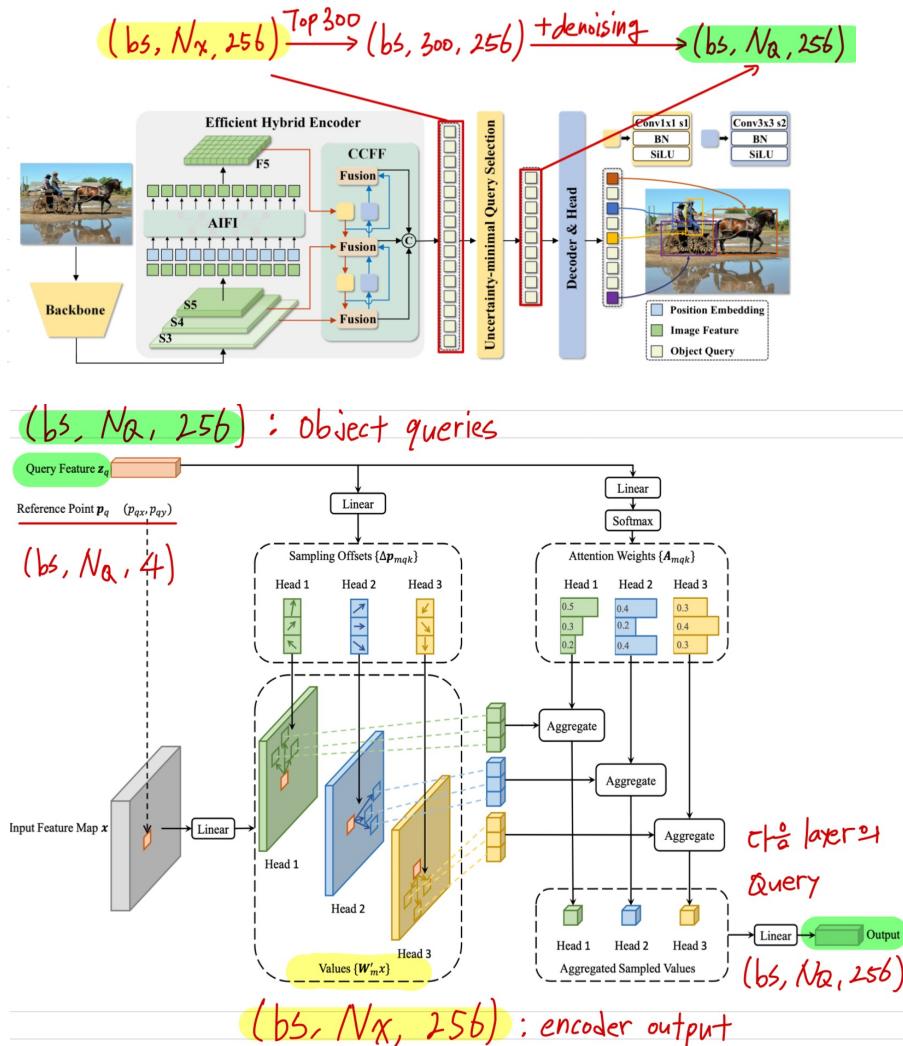


Figure 4. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale Feature Fusion (CCFF). Then, the uncertainty-minimal query selection selects a fixed number of encoder features to serve as initial object queries for the decoder. Finally, the decoder with auxiliary prediction heads iteratively optimizes object queries to generate categories and boxes.

# RT DETR : Decoder

- Decoder layer : self-attention 이후에 MSDeformableAttention



```

5454 (decoder layer 0) :
5455
5456
5457 (self_atten)
5458   (Query = object queries) tgt : torch.Size([4, 480, 256]) (bs, N_Q, 256)
5459   (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5460   (Value = memory) memory : torch.Size([4, 11109, 256]) (bs, N_X, 256)
5461   (self_atten output) tgt : torch.Size([4, 480, 256])
5462
5463 (cross-attention) MSDeformableAttention
5464   (Query = object queries) tgt : torch.Size([4, 480, 256])
5465   (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5466   (Value = memory) memory : torch.Size([4, 11109, 256])
   (cross_atten output = next layer Query) tgt : torch.Size([4, 480, 256])
5467
5468
5469
5470
5471
5472
5473
5474
5475
5476
5477
5478
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528
5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5599
5599
5600
5601
5602
5603
5604
5605
5606 (decoder layer 5) :
5607
5608
5609 (self_atten)
5610   (Query = object queries) tgt : torch.Size([4, 480, 256])
5611   (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5612   (Value = memory) memory : torch.Size([4, 11109, 256])
5613   (self_atten output) tgt : torch.Size([4, 480, 256])
5614
5615 (cross-attention) MSDeformableAttention
5616   (Query = object queries) tgt : torch.Size([4, 480, 256])
5617   (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5618   (Value = memory) memory : torch.Size([4, 11109, 256])
   (cross_atten output = next layer Query) tgt : torch.Size([4, 480, 256])
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5888
5889
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5898
5899
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5978
5979
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5998
5999
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6029
6030
6031
6032
6033
6034
6035
6036
6037
6038
6039
6039
6040
6041
6042
6043
6044
6045
6046
6047
6048
6049
6049
6050
6051
6052
6053
6054
6055
6056
6057
6058
6059
6059
6060
6061
6062
6063
6064
6065
6066
6067
6068
6069
6069
6070
6071
6072
6073
6074
6075
6076
6077
6078
6078
6079
6079
6080
6081
6082
6083
6084
6085
6086
6087
6088
6089
6089
6090
6091
6092
6093
6094
6095
6096
6097
6098
6098
6099
6099
6100
6101
6102
6103
6104
6105
6106
6107
6108
6109
6109
6110
6111
6112
6113
6114
6115
6116
6117
6118
6118
6119
6119
6120
6121
6122
6123
6124
6125
6126
6127
6128
6129
6129
6130
6131
6132
6133
6134
6135
6136
6137
6138
6139
6139
6140
6141
6142
6143
6144
6145
6146
6147
6148
6149
6149
6150
6151
6152
6153
6154
6155
6156
6157
6158
6159
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6178
6179
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6198
6199
6199
6200
6201
6202
6203
6204
6205
6206
6207
6208
6209
6209
6210
6211
6212
6213
6214
6215
6216
6217
6218
6218
6219
6219
6220
6221
6222
6223
6224
6225
6226
6227
6228
6229
6229
6230
6231
6232
6233
6234
6235
6236
6237
6238
6239
6239
6240
6241
6242
6243
6244
6245
6246
6247
6248
6249
6249
6250
6251
6252
6253
6254
6255
6256
6257
6258
6259
6259
6260
6261
6262
6263
6264
6265
6266
6267
6268
6269
6269
6270
6271
6272
6273
6274
6275
6276
6277
6278
6278
6279
6279
6280
6281
6282
6283
6284
6285
6286
6287
6288
6289
6289
6290
6291
6292
6293
6294
6295
6296
6297
6298
6298
6299
6299
6300
6301
6302
6303
6304
6305
6306
6307
6308
6309
6309
6310
6311
6312
6313
6314
6315
6316
6317
6318
6318
6319
6319
6320
6321
6322
6323
6324
6325
6326
6327
6328
6329
6329
6330
6331
6332
6333
6334
6335
6336
6337
6338
6339
6339
6340
6341
6342
6343
6344
6345
6346
6347
6348
6349
6349
6350
6351
6352
6353
6354
6355
6356
6357
6358
6359
6359
6360
6361
6362
6363
6364
6365
6366
6367
6368
6369
6369
6370
6371
6372
6373
6374
6375
6376
6377
6378
6378
6379
6379
6380
6381
6382
6383
6384
6385
6386
6387
6388
6389
6389
6390
6391
6392
6393
6394
6395
6396
6397
6398
6398
6399
6399
6400
6401
6402
6403
6404
6405
6406
6407
6408
6409
6409
6410
6411
6412
6413
6414
6415
6416
6417
6418
6418
6419
6419
6420
6421
6422
6423
6424
6425
6426
6427
6428
6429
6429
6430
6431
6432
6433
6434
6435
6436
6437
6438
6439
6439
6440
6441
6442
6443
6444
6445
6446
6447
6448
6449
6449
6450
6451
6452
6453
6454
6455
6456
6457
6458
6459
6459
6460
6461
6462
6463
6464
6465
6466
6467
6468
6469
6469
6470
6471
6472
6473
6474
6475
6476
6477
6478
6478
6479
6479
6480
6481
6482
6483
6484
6485
6486
6487
6488
6489
6489
6490
6491
6492
6493
6494
6495
6496
6497
6497
6498
6498
6499
6499
6500
6501
6502
6503
6504
6505
6506
6507
6508
6509
6509
6510
6511
6512
6513
6514
6515
6516
6517
6518
6518
6519
6519
6520
6521
6522
6523
6524
6525
6526
6527
6528
6529
6529
6530
6531
6532
6533
6534
6535
6536
6537
6538
6539
6539
6540
6541
6542
6543
6544
6545
6546
6547
6548
6548
6549
6549
6550
6551
6552
6553
6554
6555
6556
6557
6558
6559
6559
6560
6561
6562
6563
6564
6565
6566
6567
6568
6569
6569
6570
6571
6572
6573
6574
6575
6576
6577
6578
6578
6579
6579
6580
6581
6582
6583
6584
6585
6586
6587
6588
6589
6589
6590
6591
6592
6593
6594
6595
6596
6597
6597
6598
6598
6599
6599
6600
6601
6602
6603
6604
6605
6606
6607
6608
6609
6609
6610
6611
6612
6613
6614
6615
6616
6617
6618
6618
6619
6619
6620
6621
6622
6623
6624
6625
6626
6627
6628
6629
6629
6630
6631
6632
6633
6634
6635
6636
6637
6638
6639
6639
6640
6641
6642
6643
6644
6645
6646
6647
6648
6649
6649
6650
6651
6652
6653
6654
6655
6656
6657
6658
6659
6659
6660
6661
6662
6663
6664
6665
6666
6667
6668
6669
6669
6670
6671
6672
6673
6674
6675
6676
6677
6678
6678
6679
6679
6680
6681
6682
6683
6684
6685
6686
6687
6688
6689
6689
6690
6691
6692
6693
6694
6695
6696
6697
6697
6698
6698
6699
6699
6700
6701
6702
6703
6704
6705
6706
6707
6708
6709
6709
6710
6711
6712
6713
6714
6715
6716
6717
6718
6718
6719
6719
6720
6721
6722
6723
6724
6725
6726
6727
6728
6729
6729
6730
6731
6732
6733
6734
6735
6736
6737
6738
6739
6739
6740
6741
6742
6743
6744
6745
6746
6747
6748
6749
6749
6750
6751
6752
6753
6754
6755
6756
6757
6758
6759
6759
6760
6761
6762
6763
6764
6765
6766
6767
6768
6769
6769
6770
6771
6772
6773
6774
6775
6776
6777
6778
6778
6779
6779
6780
6781
6782
6783
6784
6785
6786
6787
6788
6789
6789
6790
6791
6792
6793
6794
6795
6796
6797
6798
6798
6799
6799
6800
6801
6802
6803
6804
6805
6806
6807
6808
6809
6809
6810
6811
6812
6813
6814
6815
6816
6817
6818
6818
6819
6819
6820
6821
6822
6823
6824
6825
6826
6827
6828
6829
6829
6830
6831
6832
6833
6834
6835
6836
6837
6838
6839
6839
6840
6841
6842
6843
6844
6845
6846
6847
6848
6849
6849
6850
6851
6852
6853
6854
6855
6856
6857
6858
6859
6859
6860
6861
6862
6863
6864
6865
6866
6867
6868
6869
6869
6870
6871
6872
6873
6874
6875
6876
6877
6878
6878
6879
6879
6880
6881
6882
6883
6884
6885
6886
6887
6888
6889
6889
6890
6891
6892
6893
6894
6895
6896
6897
6898
6898
6899
6899
6900
6901
6902
6903
6904
6905
6906
6907
6908
6909
6909
6910
6911
6912
6913
6914
6915
6916
6917
6918
6919
6919
6920
6921
6922
6923
6924
6925
6926
6927
6928
6929
6929
6930
6931
6932
6933
6934
6935
6936
6937
6938
6939
6939
6940
6941
6942
6943
6944
6945
6946
6947
6948
6949
6949
6950
6951
6952
6953
6954
6955
6956
6957
6958
6959
6959
6960
6961
6962
6963
6964
6965
6966
6967
6968
6969
6969
6970
6971
6972
6973
6974
6975
6976
6977
6978
6978
6979
6979
6980
6981
6982
6983
6984
6985
6986
6987
6988
6989
6989
6990
6991
6992
6993
6994
6995
6996
6997
6998
6998
6999
6999
7000
7001
7002
7003
7004
7005
7006
7007
7008
7009
7009
7010
7011
7012
7013
7014
7015
7016
7017
7018
7019
7019
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7029
7030
7031
7032
7033
7034
7035
7036
7037
7038
7039
7039
7040
7041
7042
7043
7044
7045
7046
7047
7048
7049
7049
7050
7051
7052
7053
7054
7055
7056
7057
7058
7059
7059
7060
7061
7062
7063
7064
7065
7066
7067
7068
7069
7069
7070
7071
7072
7073
7074
7075
7076
7077
7078
7078
7079
7079
7080
7081
7082
7083
7084
7085
7086
7087
7088
7089
7089
7090
7091
7092
7093
7094
7095
7096
7097
7098
7098
7099
7099
7100
7101
7102
7103
7104
7105
7106
7107
7108
7109
7109
7110
7111
7112
7113
7114
7115
7116
7117
7118
7119
7119
7120
7121
7122
7123
7124
7125
7126
7127
7128
7129
7129
7130
7131
7132
7133
7134
7135
7136
7137
7138
7139
7139
7140
7141
7142
7143
7144
7145
7146
7147
7148
7149
7149
7150
7151
7152
7153
7154
7155
7156
7157
7158
7159
7159
7160
7161
7162
7163
7164
7165
7166
7167
7168
7169
7169
7170
7171
7172
7173
7174
7175
7176
7177
7178
7178
7179
7179
7180
7181
7182
7183
7184
7185
7186
7187
7188
7189
7189
7190
7191
7192
7193
7194
7195
7196
7197
7198
7198
7199
7199
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7229
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239
7239
7240
7241
7242
7243
7244
7245
7246
7247
7248
7249
7249
7250
7251
7252
7253
7254
7255
7256
7257
7258
7259
7259
7260
7261
7262
7263
7264
7265
7266
7267
7268
7269
7269
7270
7271
7272
7273
7274
7275
7276
7277
7278
7279
7279
7280
7281
7282
7283
7284
7285
7286
7287
7288
7289
7289
7290
7291
7292
7293
7294
7295
7296
7297
7298
7298
7299
7299
73
```

# RT DETR : Experiments

- Comparison with SOTA

anchor - based < anchor - free

faster, more accurate

Model	Backbone	#Epochs	#Params (M)	GFLOPs	FPS <sub>bs=1</sub>	AP <sup>val</sup>	AP <sub>50</sub> <sup>val</sup>	AP <sub>75</sub> <sup>val</sup>	AP <sub>S</sub> <sup>val</sup>	AP <sub>M</sub> <sup>val</sup>	AP <sub>L</sub> <sup>val</sup>
<i>Real-time Object Detectors</i>											
YOLOv5-L [11]	-	300	46	109	54	49.0	67.3	-	-	-	-
YOLOv5-X [11]	-	300	86	205	43	50.7	68.9	-	-	-	-
PPYOLOE-L [40]	-	300	52	110	94	51.4	68.9	55.6	31.4	55.3	66.1
PPYOLOE-X [40]	-	300	98	206	60	52.3	69.9	56.5	33.3	56.3	66.4
YOLOv6-L [16]	-	300	59	150	99	52.8	70.3	57.7	34.4	58.1	70.1
YOLOv7-L [38]	-	300	36	104	55	51.2	69.7	55.5	35.2	55.9	66.7
YOLOv7-X [38]	-	300	71	189	45	52.9	71.1	57.4	36.9	57.7	68.6
YOLOv8-L [12]	-	-	43	165	71	52.9	69.8	57.5	35.3	58.3	69.8
YOLOv8-X [12]	-	-	68	257	50	53.9	71.0	58.7	35.7	59.3	70.7
<i>End-to-end Object Detectors</i>											
DETR-DC5 [4]	R50	500	41	187	-	43.3	63.1	45.9	22.5	47.3	61.1
DETR-DC5 [4]	R101	500	60	253	-	44.9	64.7	47.7	23.7	49.5	62.3
Anchor-DETR-DC5 [39]	R50	50	39	172	-	44.2	64.7	47.5	24.7	48.2	60.6
Anchor-DETR-DC5 [39]	R101	50	-	-	-	45.1	65.7	48.8	25.8	49.4	61.6
Conditional-DETR-DC5 [27]	R50	108	44	195	-	45.1	65.4	48.5	25.3	49.0	62.2
Conditional-DETR-DC5 [27]	R101	108	63	262	-	45.9	66.8	49.5	27.2	50.3	63.3
Efficient-DETR [42]	R50	36	35	210	-	45.1	63.1	49.1	28.3	48.4	59.0
Efficient-DETR [42]	R101	36	54	289	-	45.7	64.1	49.5	28.2	49.1	60.2
SMCA-DETR [9]	R50	108	40	152	-	45.6	65.5	49.1	25.9	49.3	62.6
SMCA-DETR [9]	R101	108	58	218	-	46.3	66.6	50.2	27.2	50.5	63.2
Deformable-DETR [45]	R50	50	40	173	-	46.2	65.2	50.0	28.8	49.2	61.7
DAB-Deformable-DETR [23]	R50	50	48	195	-	46.9	66.0	50.8	30.1	50.4	62.5
DAB-Deformable-DETR++ [23]	R50	50	47	-	-	48.7	67.2	53.0	31.4	51.6	63.9
DN-Deformable-DETR [17]	R50	50	48	195	-	48.6	67.4	52.7	31.0	52.0	63.7
DN-Deformable-DETR++ [17]	R50	50	47	-	-	49.5	67.6	53.8	31.3	52.6	65.4
DINO-Deformable-DETR [44]	R50	36	47	279	5	50.9	69.0	55.3	34.6	54.1	64.6
<i>Real-time End-to-end Object Detector (ours)</i>											
RT-DETR	R50	72	42	136	<b>108</b>	<b>53.1</b>	<b>71.3</b>	<b>57.7</b>	34.8	58.0	70.0
RT-DETR	R101	72	76	259	<b>74</b>	<b>54.3</b>	<b>72.7</b>	58.6	36.0	58.8	<b>72.1</b>

Table 2. Comparison with SOTA (only L and X models of YOLO detectors, see Appendix for the comparison with S and M models). We do not test the speed of other DETRs, except for DINO-Deformable-DETR [44] for comparison, as they are not real-time detectors. Our RT-DETR outperforms the state-of-the-art YOLO detectors and DETRs in both speed and accuracy.

# RT DETR : Experiments

- Ablation Study on Hybrid Encoder

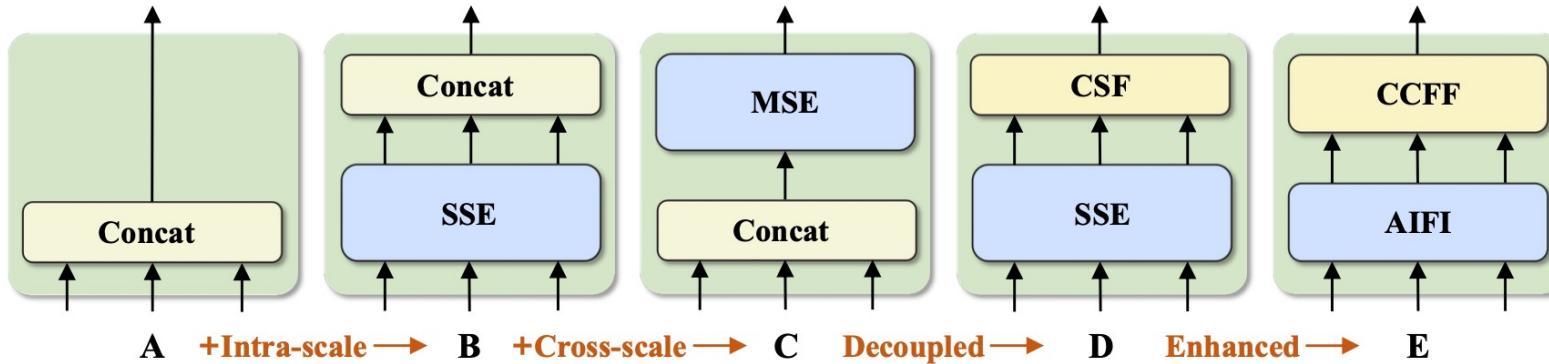


Figure 3. The encoder structure for each variant. SSE represents the single-scale Transformer encoder, MSE represents the multi-scale Transformer encoder, and CSF represents cross-scale fusion. AIFI and CCFF are the two modules designed into our hybrid encoder.

Variant	AP (%)	#Params (M)	Latency (ms)
A	43.0	31	7.2
B	44.9	32	11.1
C	45.6	32	13.3
D	46.4	35	12.2
$D_{S_5}$	46.8	35	7.9
E	47.9	42	9.3

*C* : multi-scale features들을 Concat 후 encoder  
*D<sub>S<sub>5</sub></sub>* :  $S_5$  single-scale encoder + cross-scale fusion(PANet)  
*E* :  $D_{S_5}$  + 발전된 efficient hybrid encoder (CCFF + AIFI)

Table 3. The indicators of the set of variants illustrated in Figure 3.

# RT DETR : Experiments

- Ablation Study on Decoder



ID	Det <sup>4</sup>	Det <sup>5</sup>	Det <sup>6</sup>	Det <sup>7</sup>	Latency (ms)
7	-	-	-	52.6	9.6
6	-	-	<b>53.1</b>	52.6	9.3
5	-	52.9	53.0	52.5	8.8
4	52.7	52.7	52.7	52.1	8.3
3	52.4	52.3	52.4	51.5	7.9
2	51.6	51.3	51.3	50.6	7.5
1	49.6	48.8	49.1	48.3	7.0

Table 5. Results of the ablation study on decoder. **ID** indicates decoder layer index. **Det<sup>k</sup>** represents detector with  $k$  decoder layers. All results are reported on RT-DETR-R50 with  $6\times$  configuration.

## RT DETR ADN

= Backbone(ResNet) ADN  
+ Decoder Layer ADN