| 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

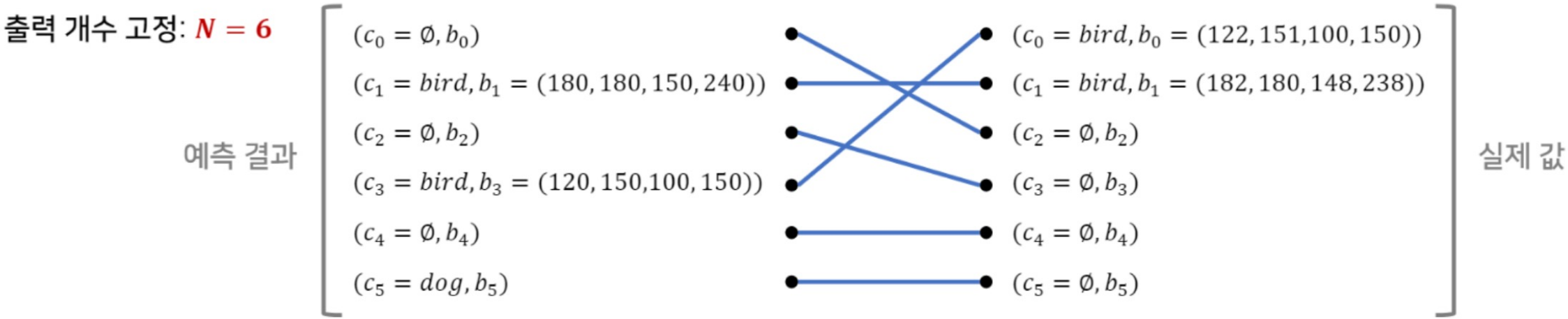
2. An architecture that predicts a set of object

   → DETR architecture

DETR : set prediction loss

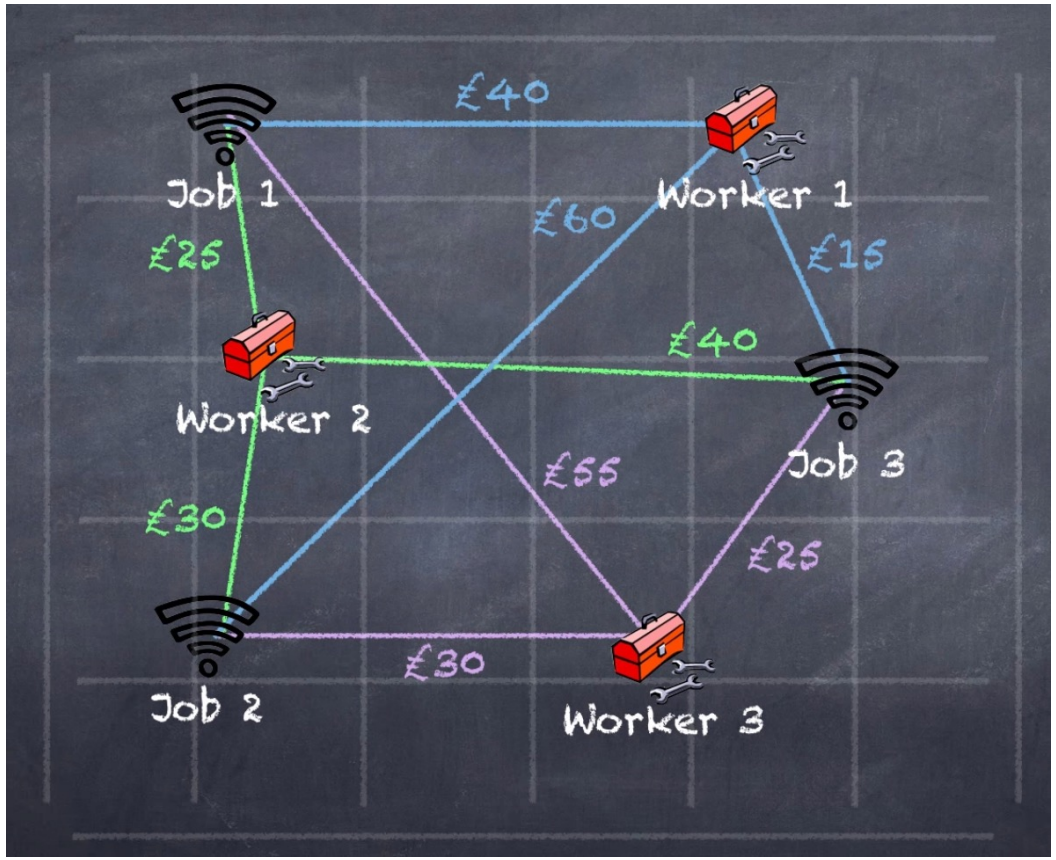
- **Bipartite matching :** training 과정에서 predicted와 GT box를 1:1 matching

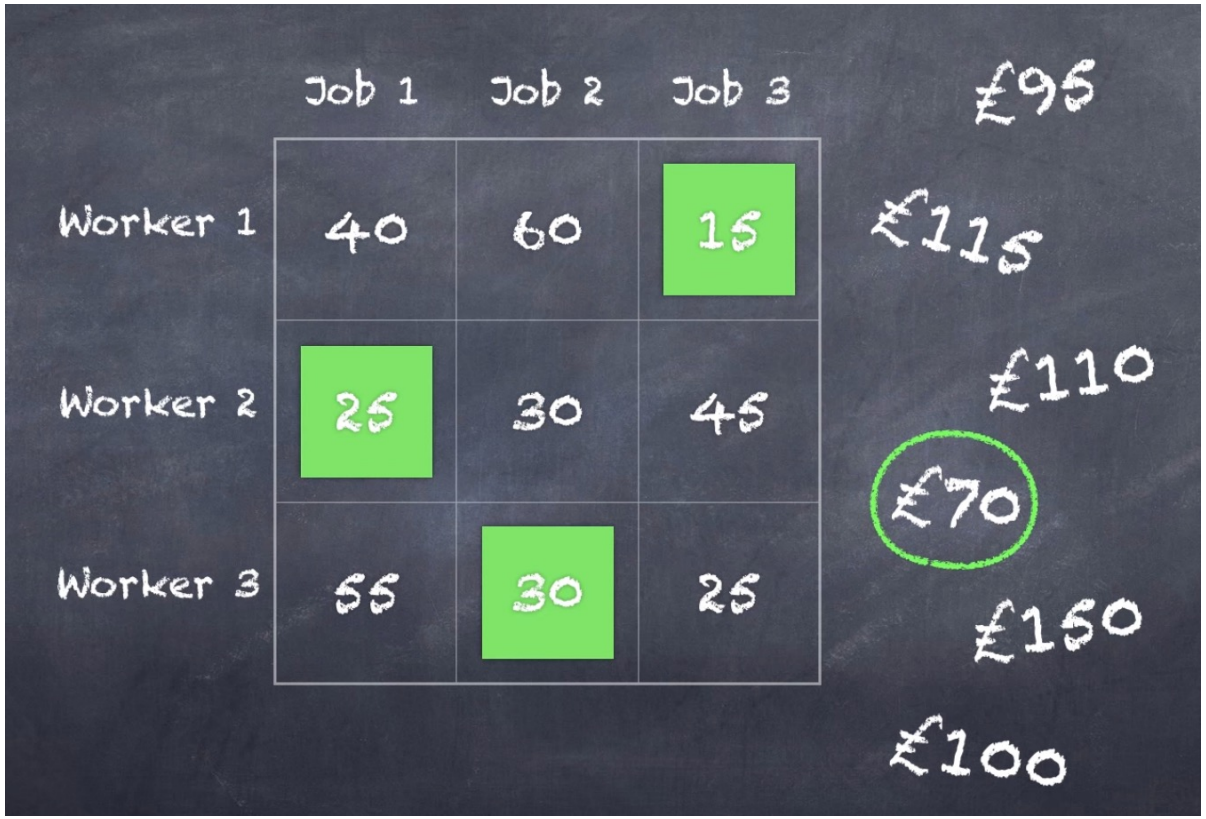
  → 중복 prediction이 없도록



DETR : set prediction loss

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





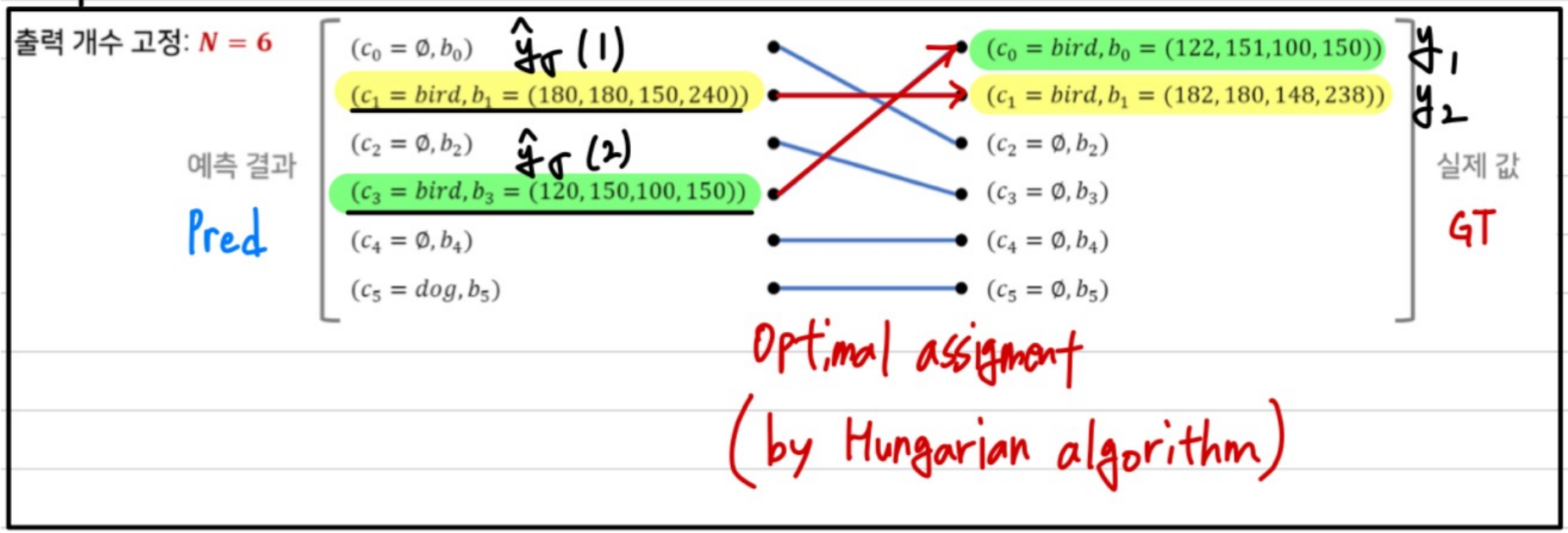
Optimal bipartite matching

DETR : set prediction loss

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

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

GT Pred

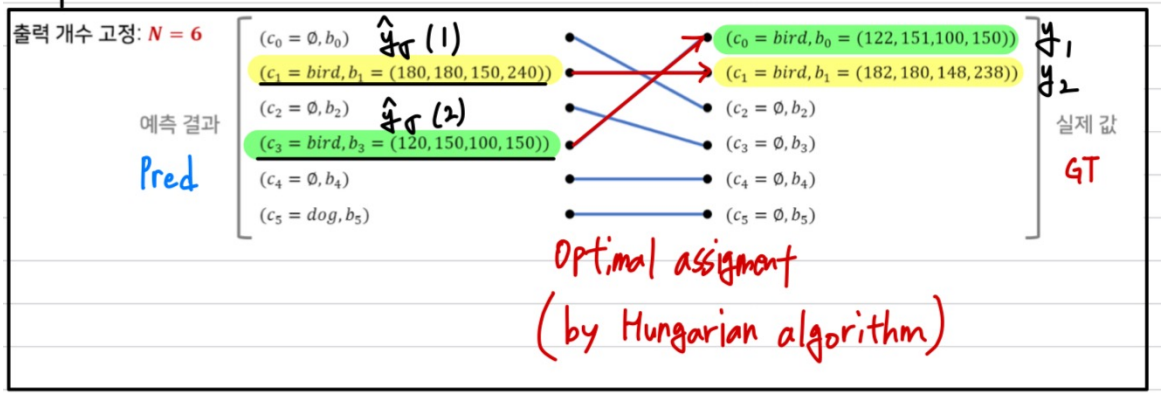


DETR : set prediction loss

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

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

GT Pred



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

DETR : set prediction loss

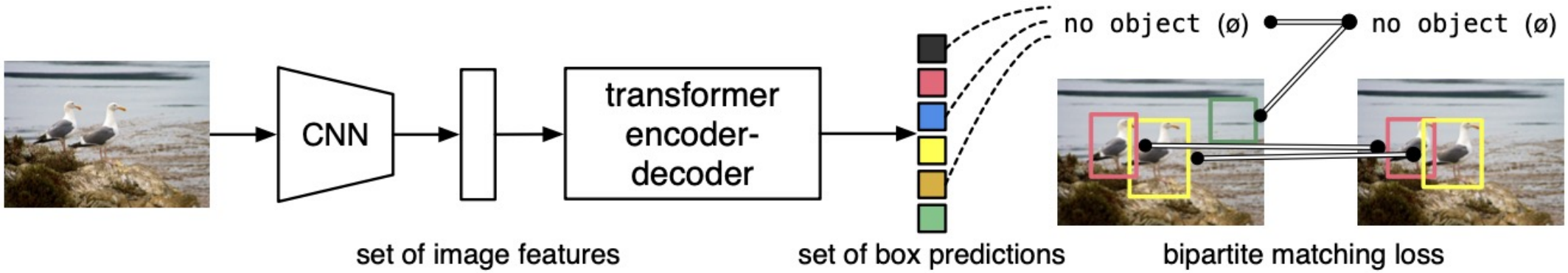


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

DETR : architecture

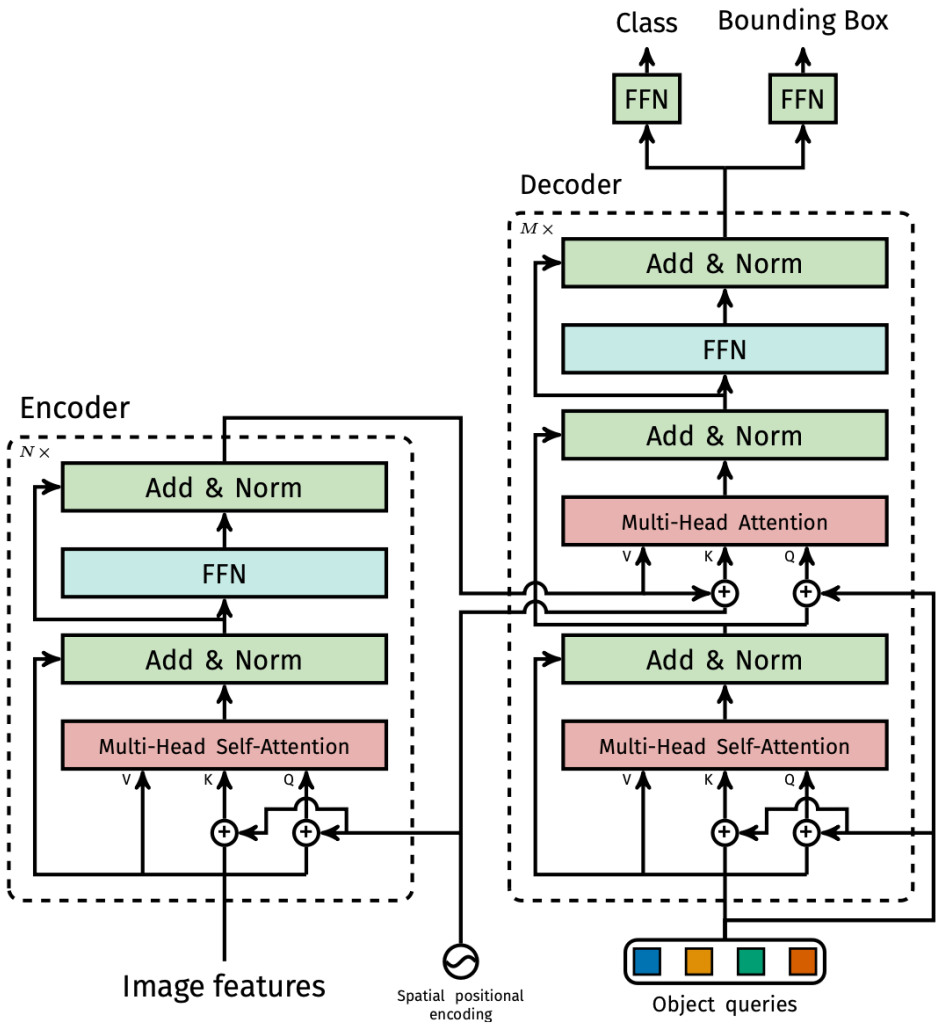
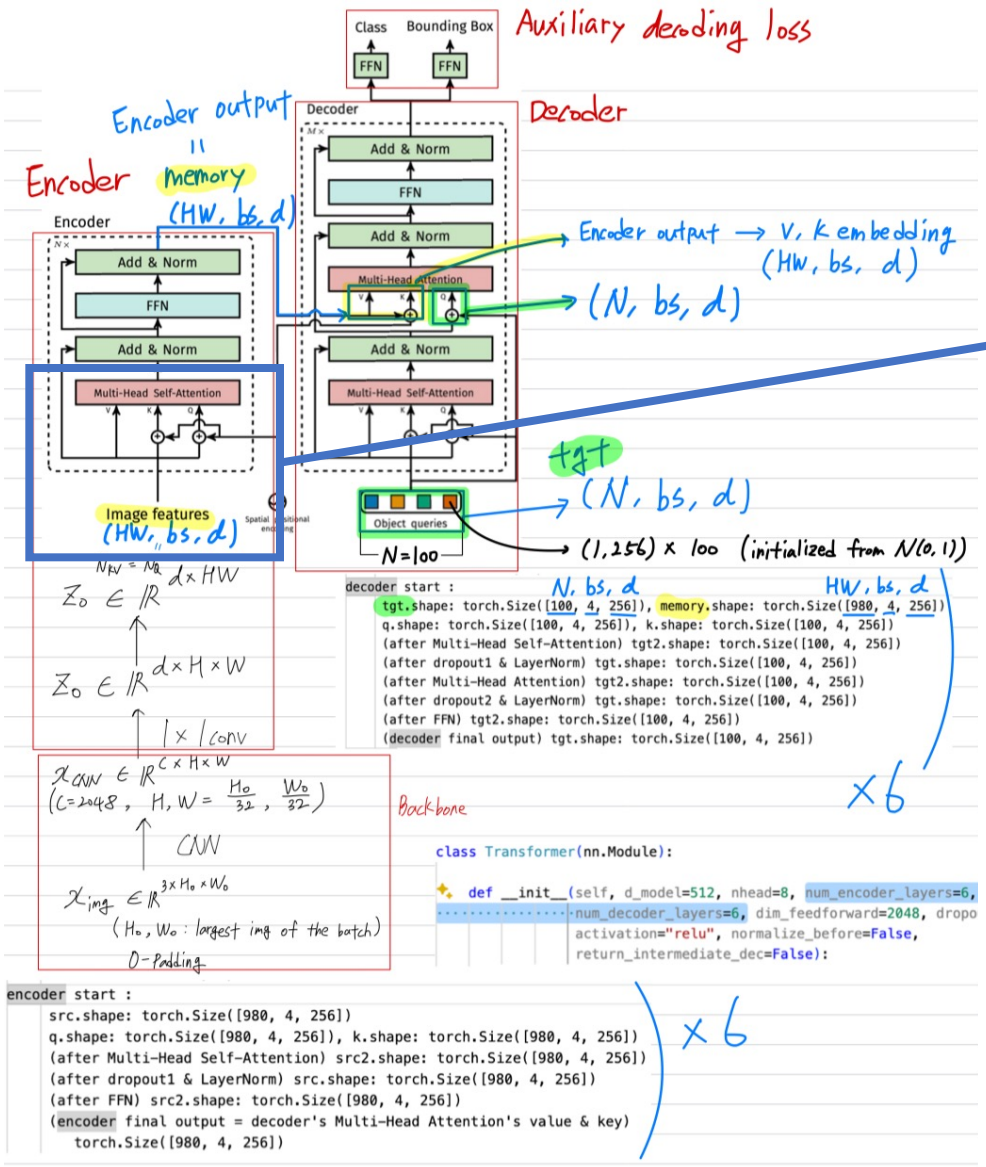
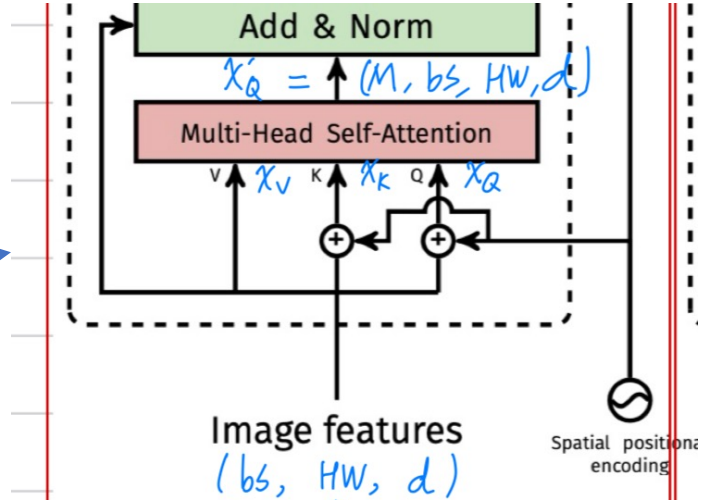


Fig. 10: Architecture of DETR's transformer. Please, see Section A.3 for details.

DETR : architecture





- $\text{Attention}(X_Q, X_K, X_V)$

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

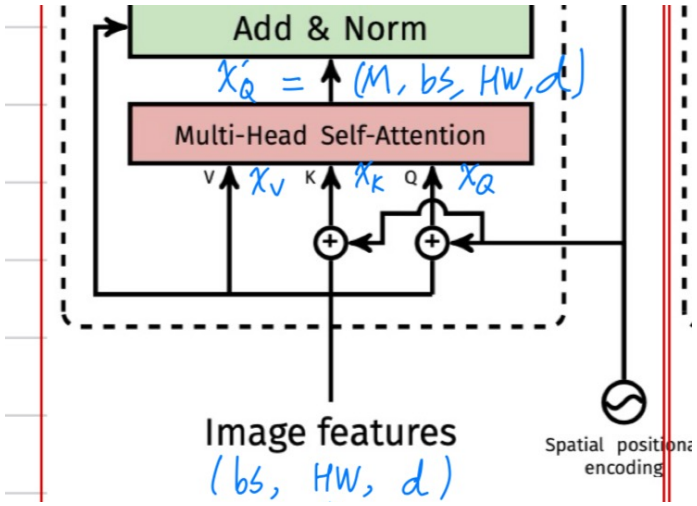
$(bs, N_Q, d)(bs, d, N_{KV})(bs, N_{KV}, d)$

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

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

$(HW, HW) \cdot (HW, d)$

DETR : 문제점



- Attention (X_Q, X_K, X_V)

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

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

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

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

$(HW, HW) \cdot (HW, d)$

- **문제점 1 :** Attention module은 image feature map의 모든 pixel에 균일한 weight가 적용됨

  → 긴 training epoch 소요

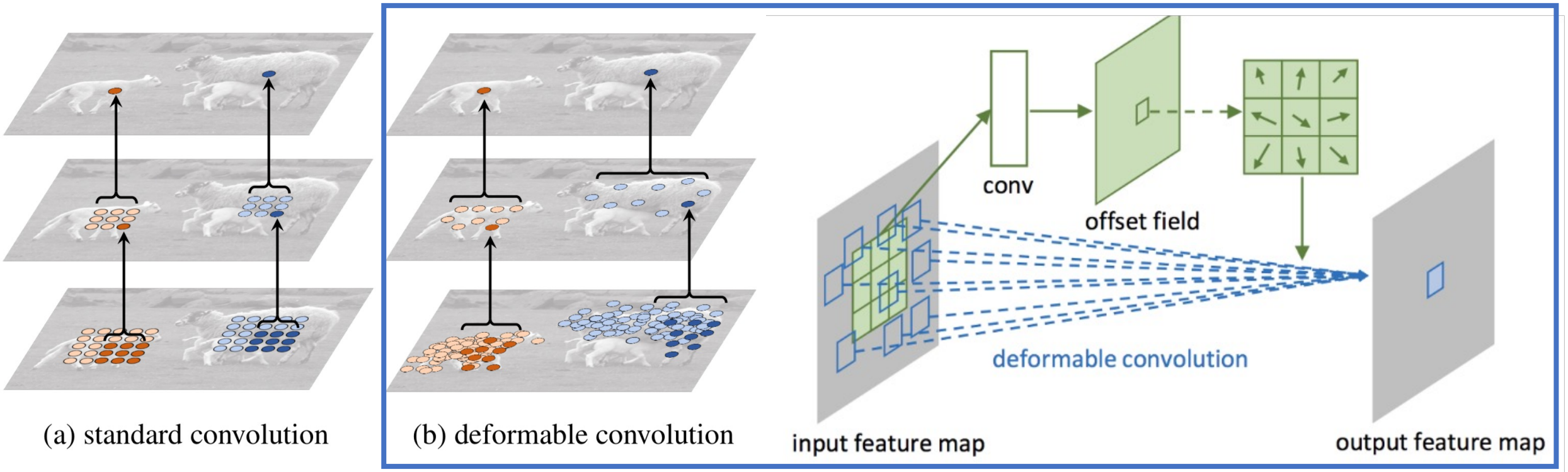
- **문제점 2 :** low performance at detection small objects.

  → 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



Deformable DETR : Multi scale deformable attention module

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

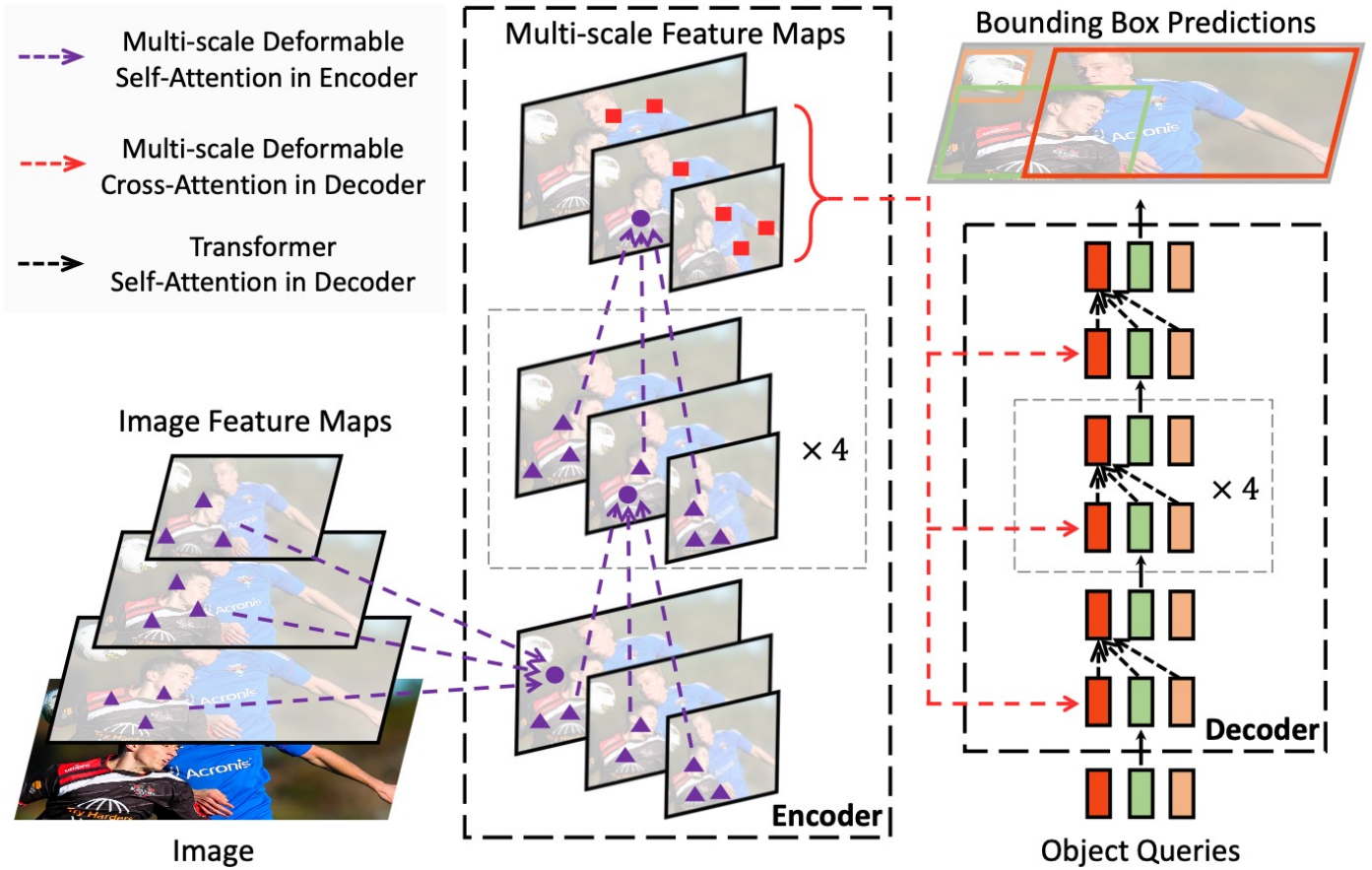


Figure 1: Illustration of the proposed Deformable DETR object detector.

Deformable DETR : Multi scale deformable attention module

- $\boldsymbol{X}$: Backbone output feature map = Encoder input feature map

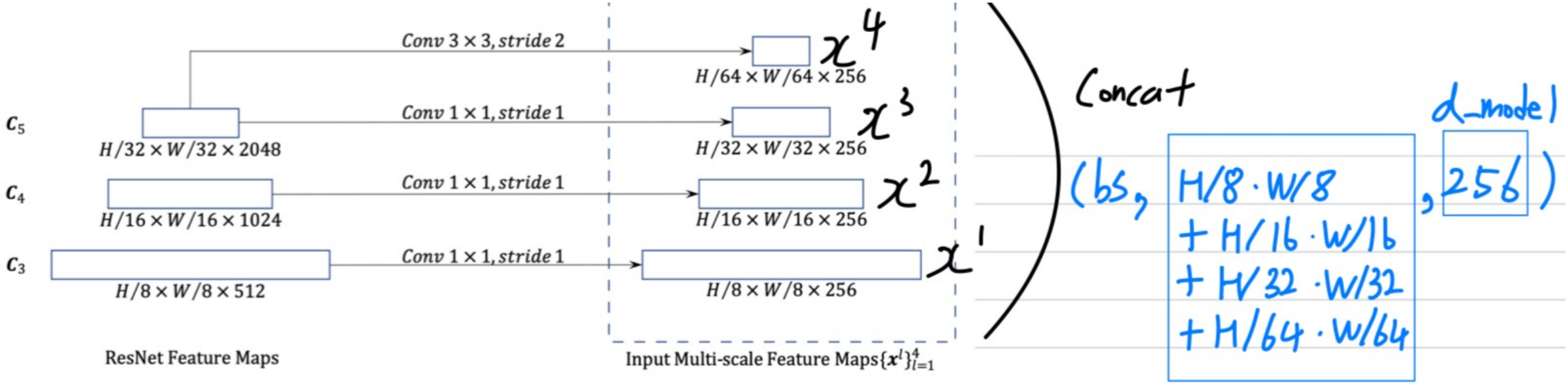
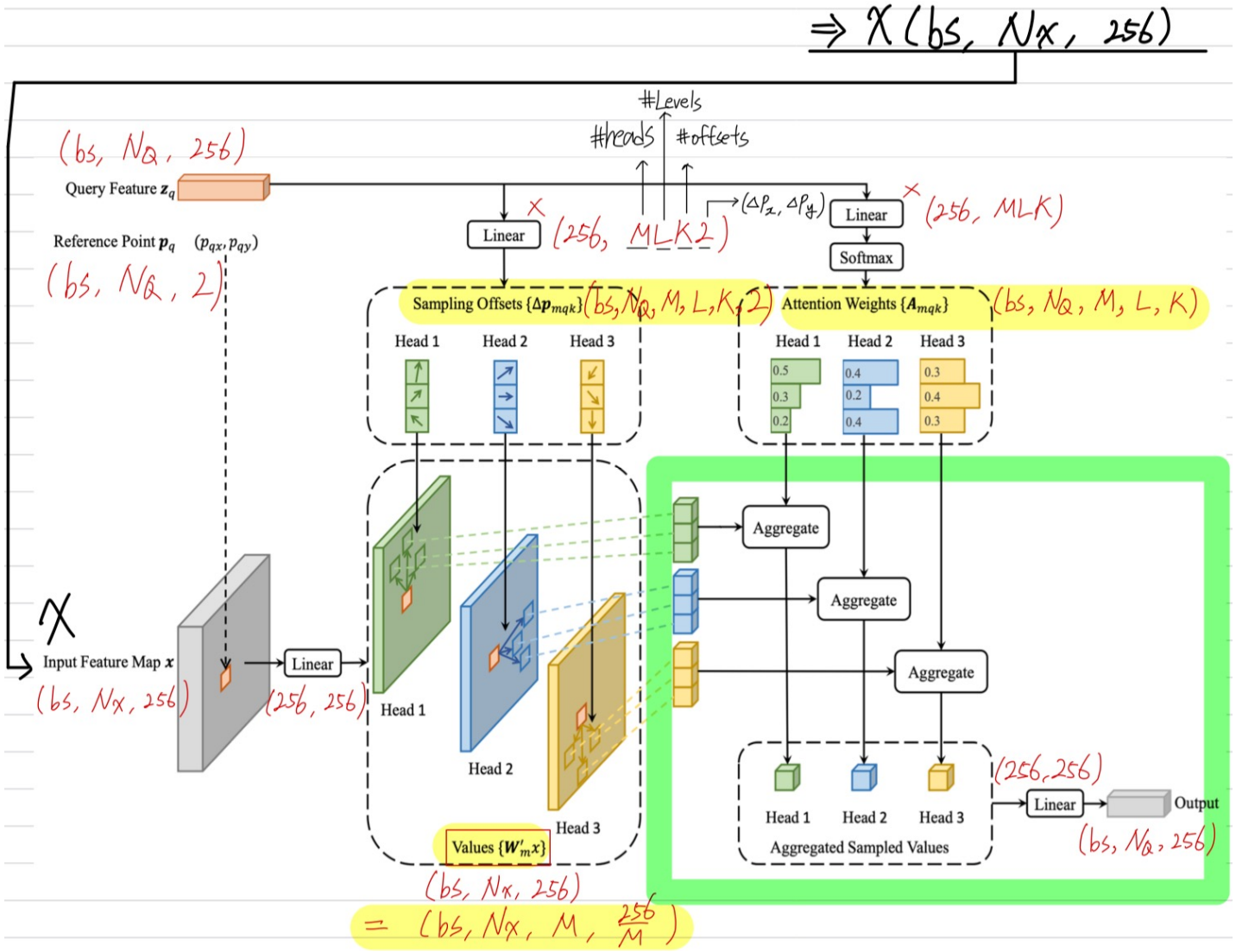


Figure 4: Constructing mult-scale feature maps for Deformable DETR.

Deformable DETR : Multi scale deformable attention module



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

(bs, NQ, 256)

Deformable DETR : Multi scale deformable attention module

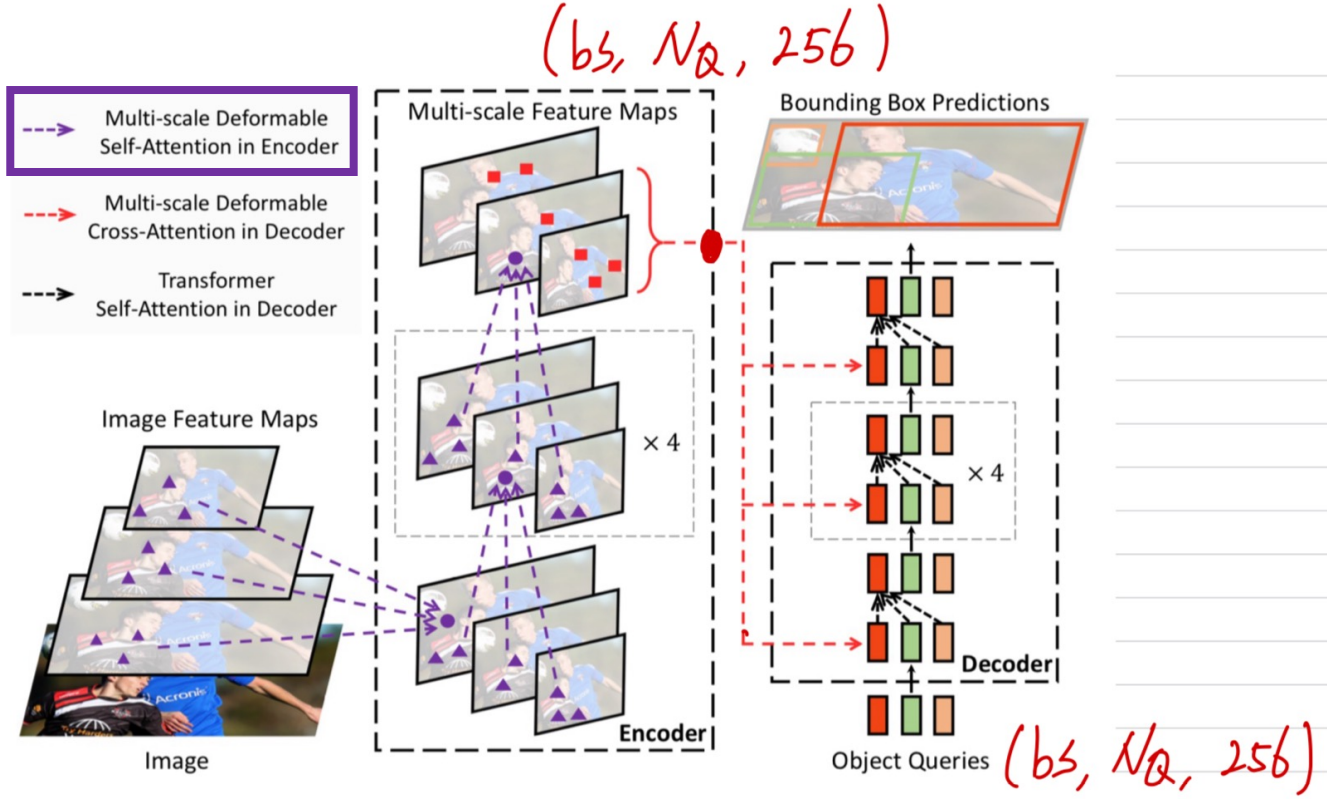
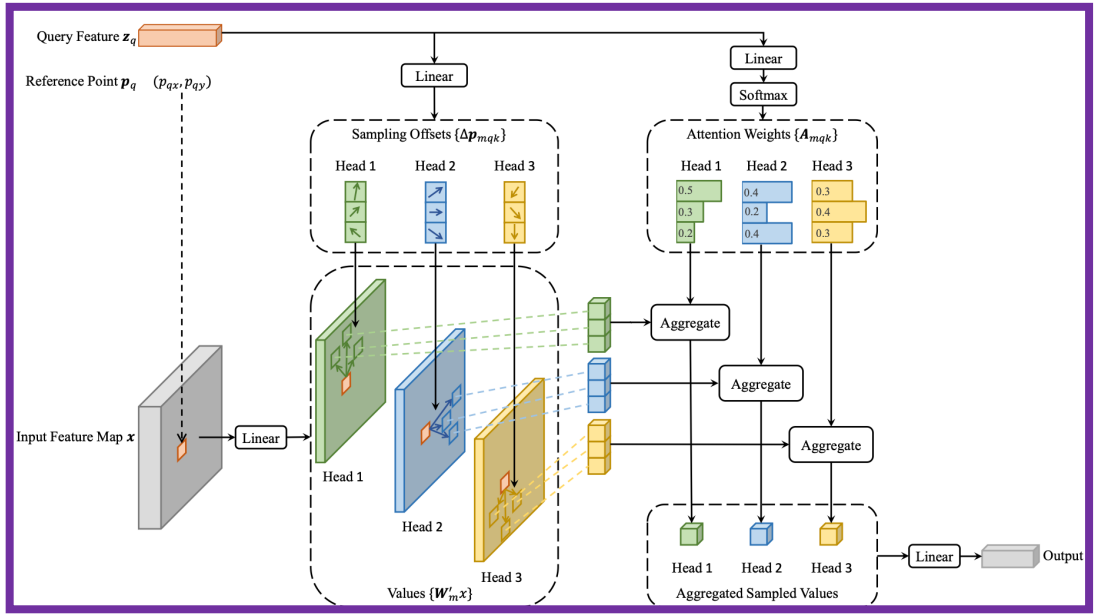


Figure 1: Illustration of the proposed Deformable DETR object detector.



RT DETR : Efficient Hybrid Encoder

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

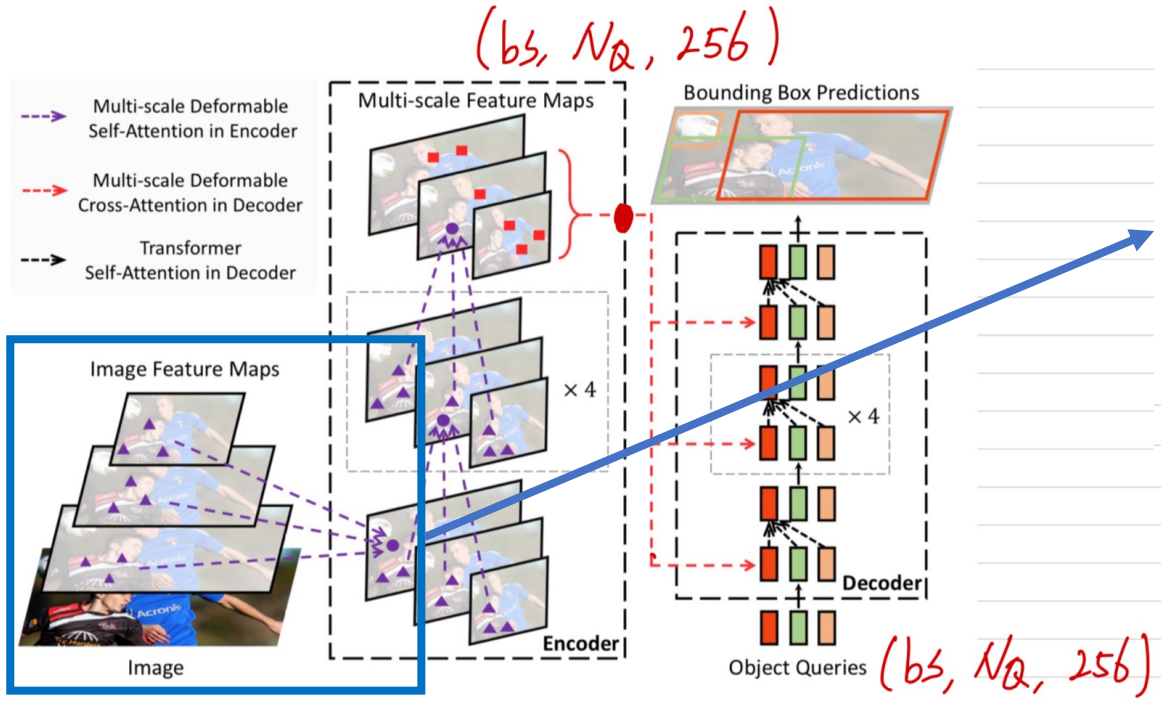


Figure 1: Illustration of the proposed Deformable DETR object detector.

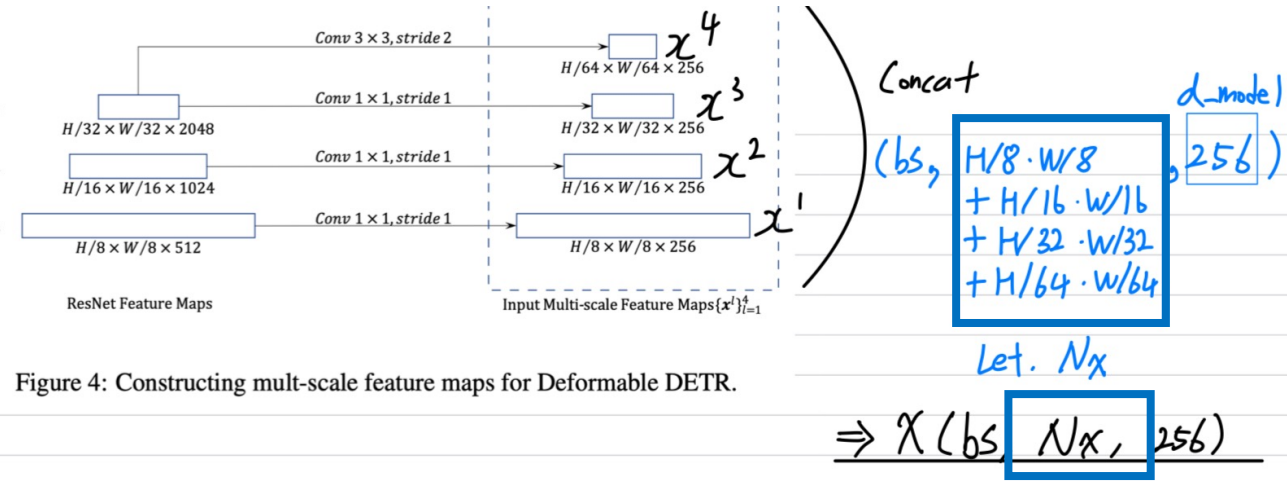


Figure 4: Constructing mult-scale feature maps for Deformable DETR.

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 $S5(=x^3)$ feature만 사용해도 될 것이다"

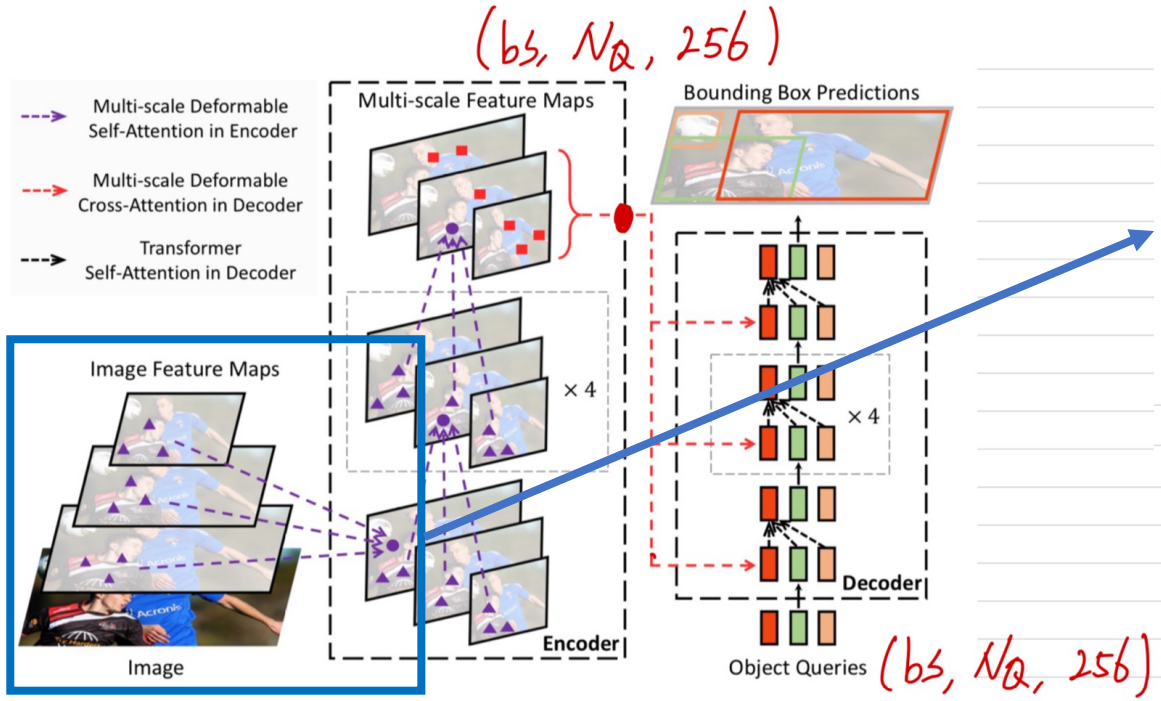


Figure 1: Illustration of the proposed Deformable DETR object detector.

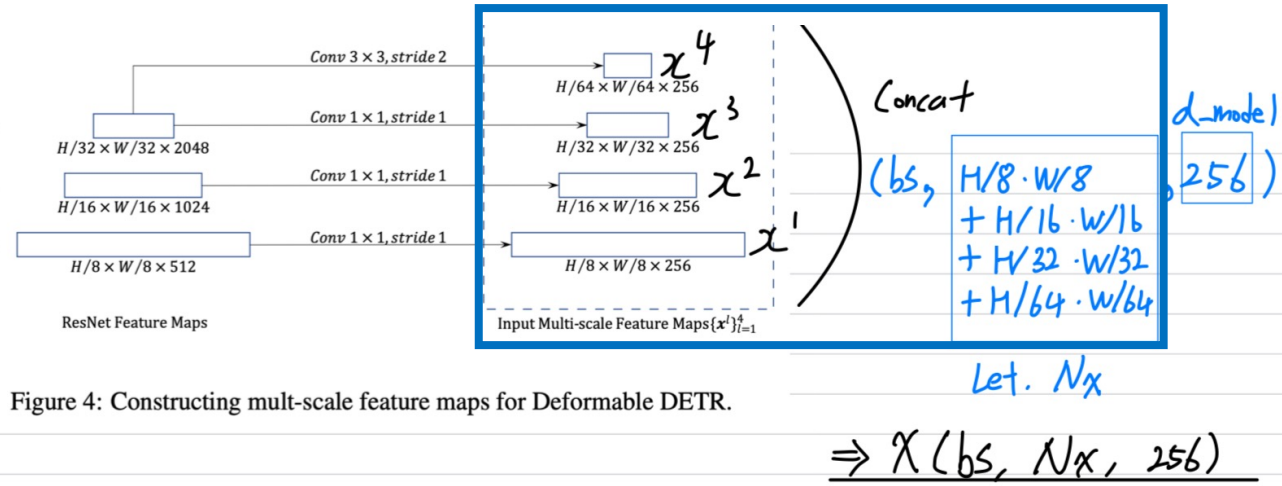


Figure 4: Constructing mult-scale feature maps for Deformable DETR.

RT DETR : Efficient Hybrid Encoder

- **문제점 1 :** "encoder input으로 high-level feature인 $S5(=x^3)$ feature만 사용해도 될 것이다"

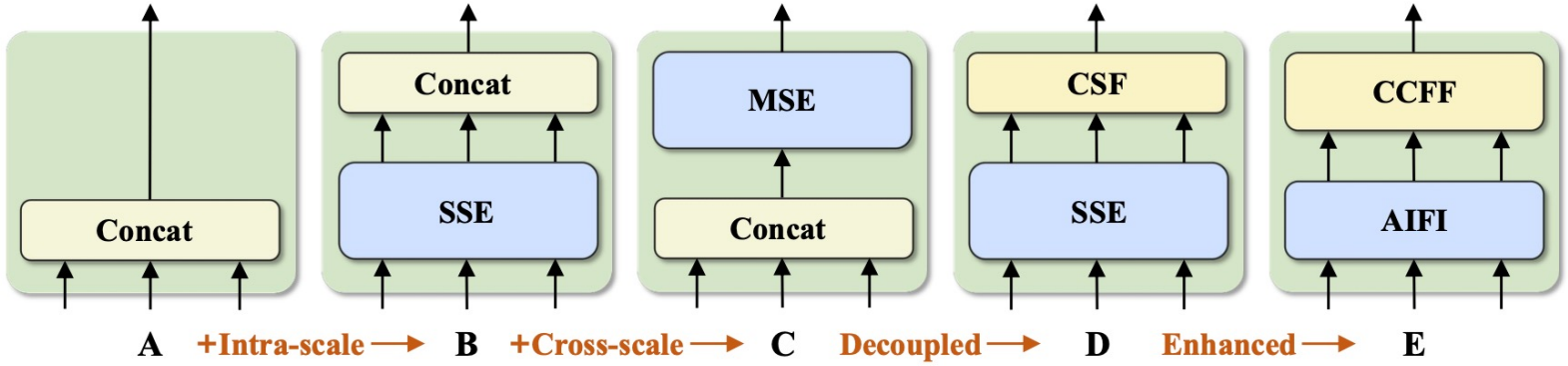


Figure 3. The encoder structure for each variant. **SSE** represents the single-scale Transformer encoder, **MSE** represents the multi-scale Transformer encoder, and **CSF** represents cross-scale fusion. **AIFI** and **CCFF** are the two modules designed into our hybrid encoder.

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

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

C : multi-scale features들을 Concat 후 encoder

D_{S_5} : S_5 single-scale encoder + cross-scale fusion(PANet)

E : D_{S_5} + 발전된 efficient hybrid encoder (CCFF + AIFI)

RT DETR : Efficient Hybrid Encoder

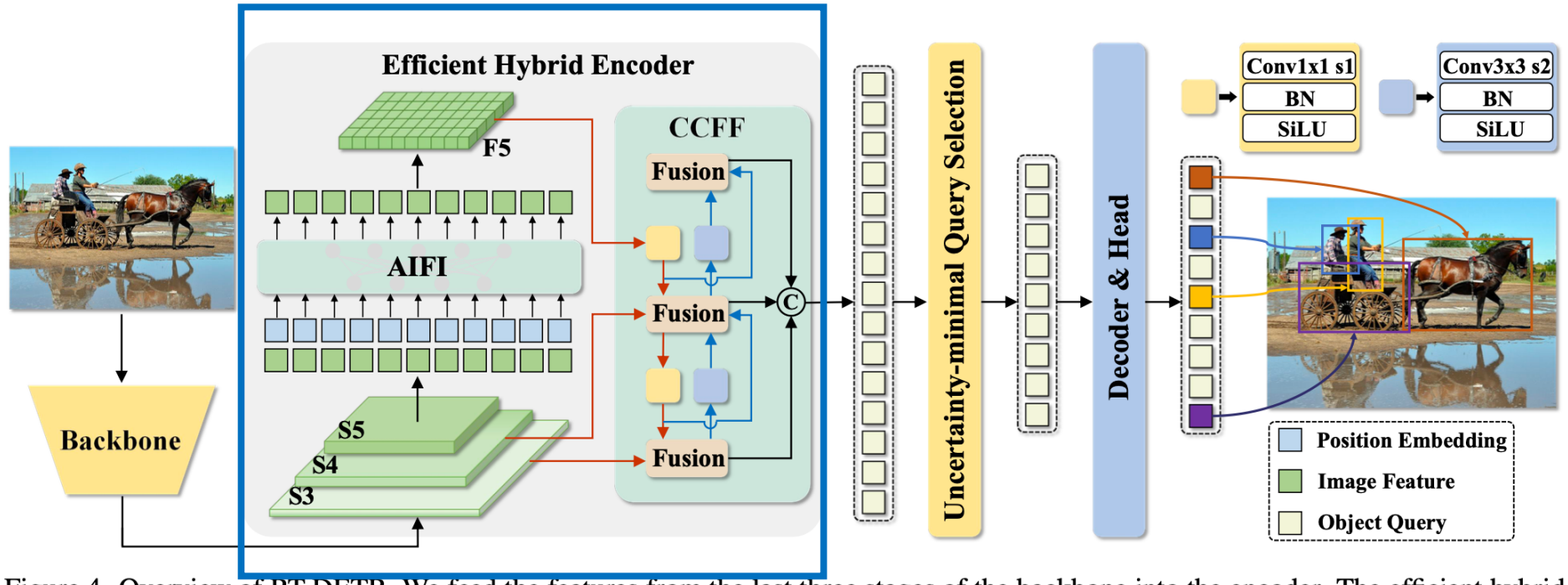
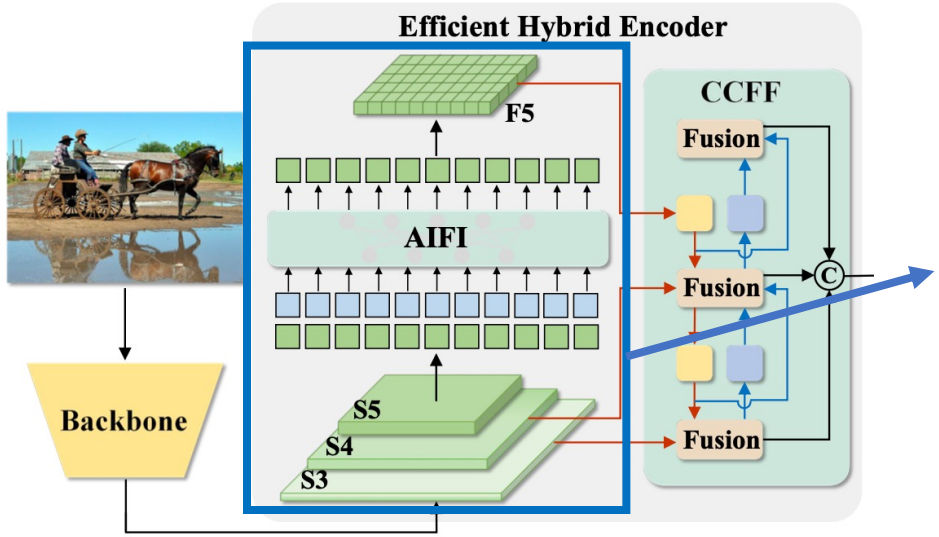


Figure 4. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale 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.

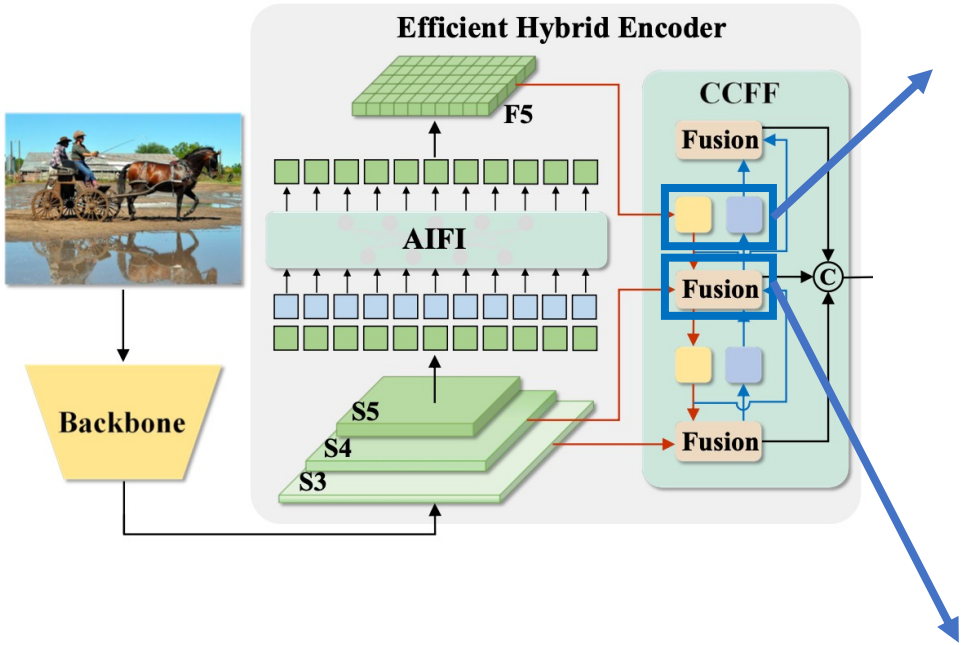
RT DETR : Efficient Hybrid Encoder



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

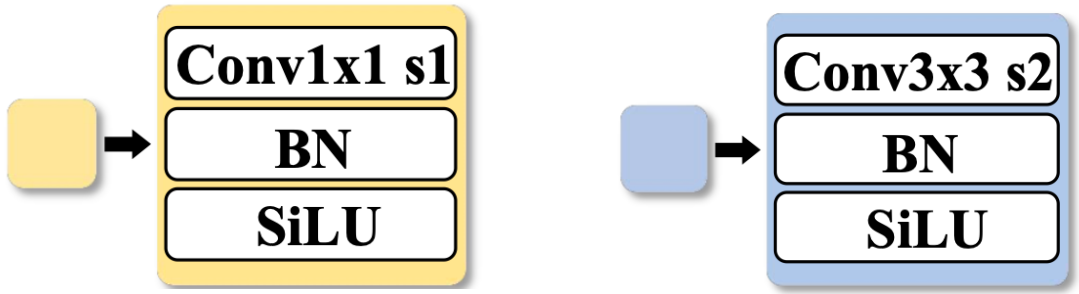
  : $S5$에 대해서만 self-attention 적용 → F5

RT DETR : Efficient Hybrid Encoder

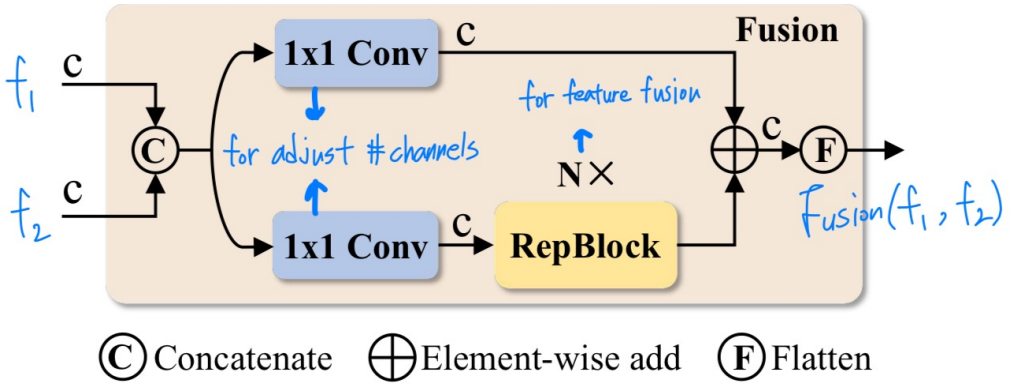


- **CCFF(Convolution-based Cross Feature Fusion)**
  - 두 feature의 height, width를 맞춰주기 위해

    upsampling　　downsampling



  - 서로 다른 두 feature를 fusion



RT DETR : Efficient Hybrid Encoder

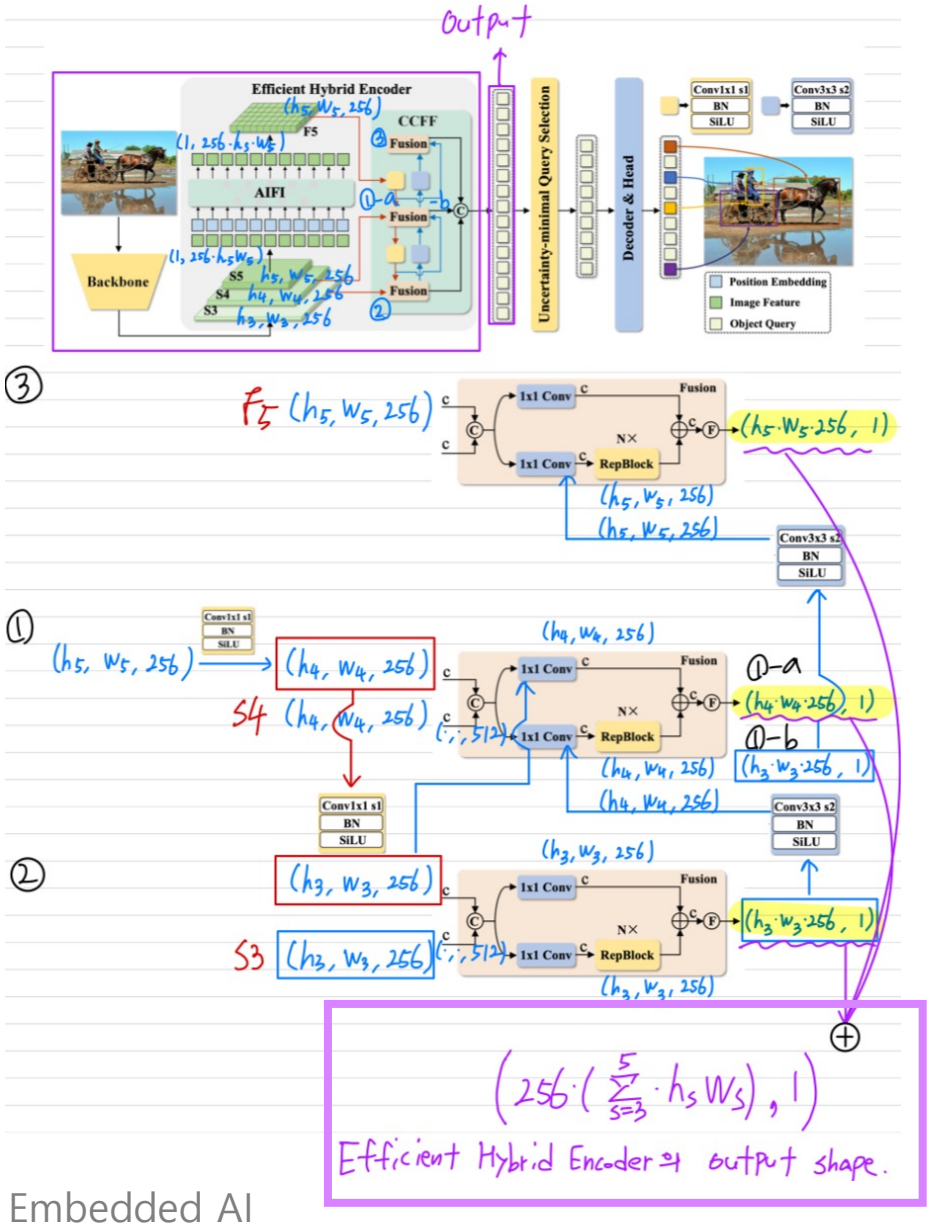


Figure 4: Constructing mult-scale feature maps for Deformable DETR.

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

- Deformable DETR query length : $\Sigma_{S=3}^{5} H_S W_S$
- RT DETR query length : $H_5 W_5$

→ Encoder의 computational bottleneck을 완화

DETR : 문제점

- **문제점 2 :** object queries initialization with Uncertainty

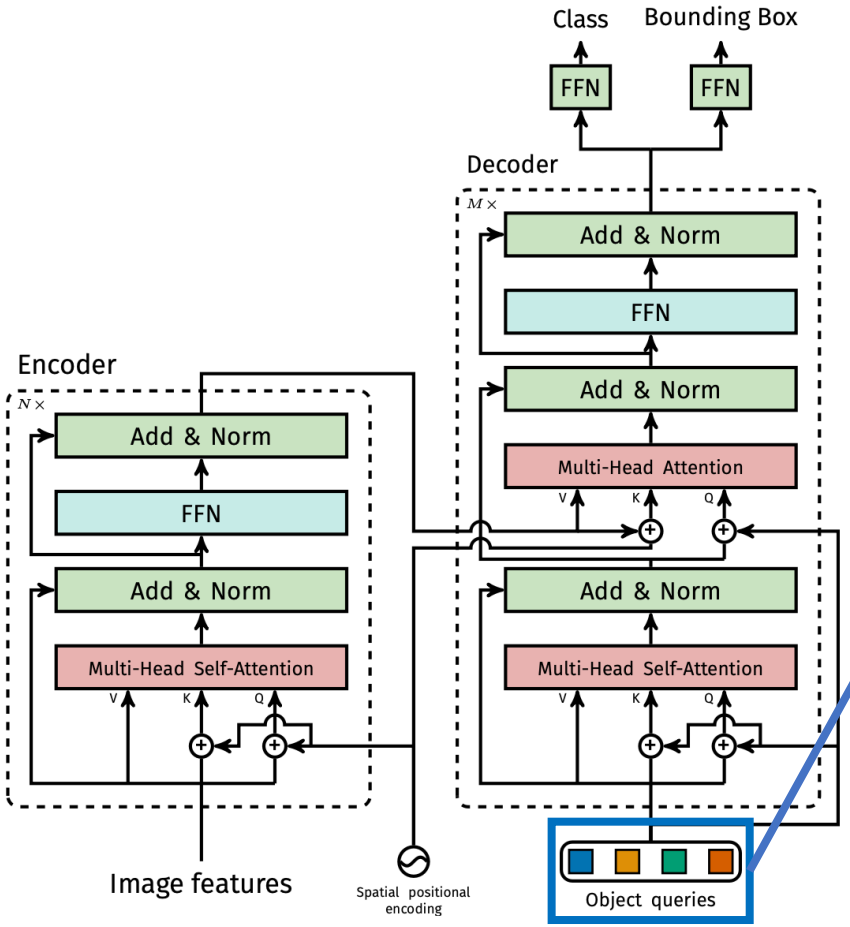


Fig. 10: Architecture of DETR's transformer. Please, see Section A.3 for details.

tgt = torch.zeros_like(bs, N_Q, 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')
```

RT DETR : Uncertainty-minimal Query Selection

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

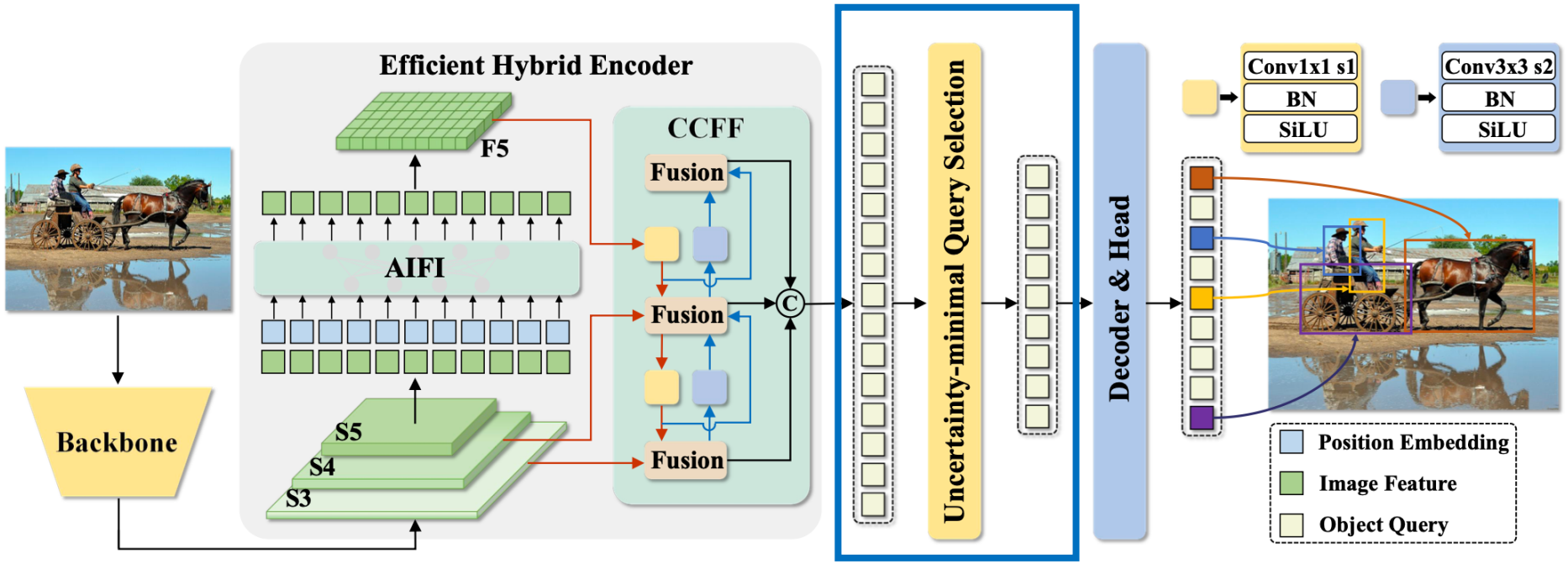


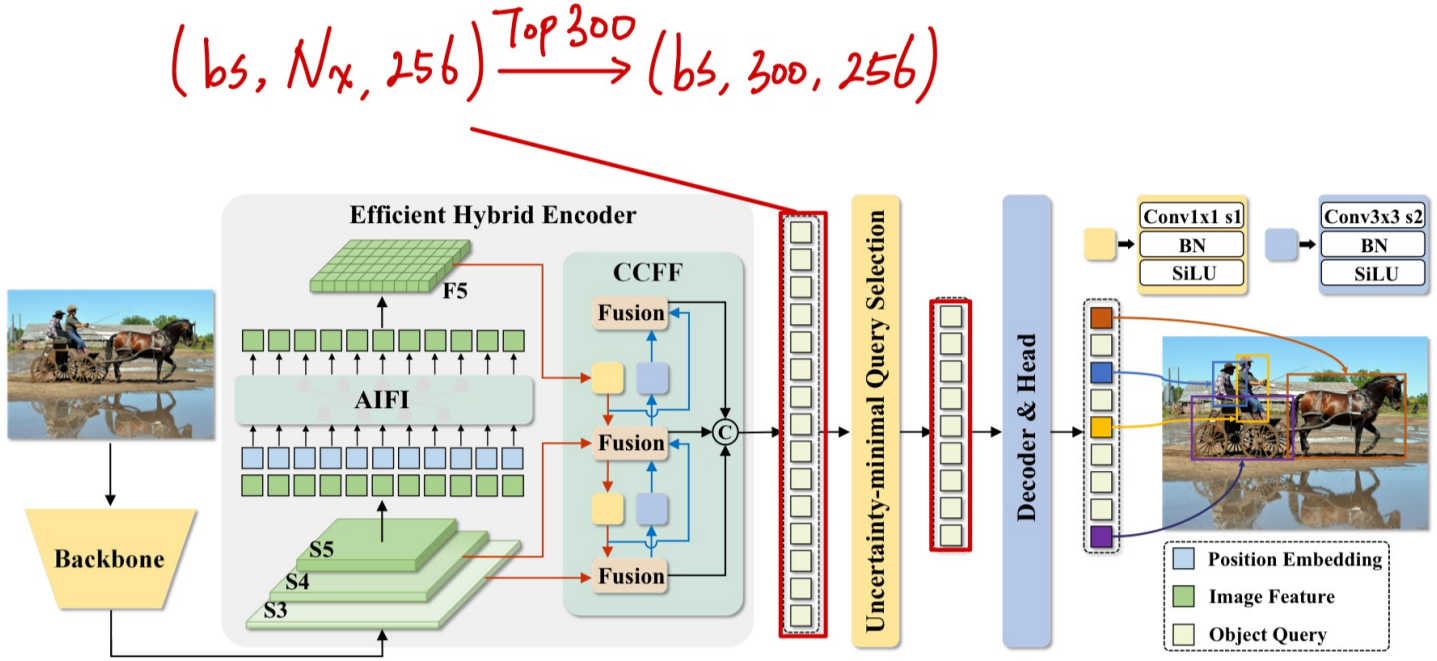
Figure 4. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale 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)



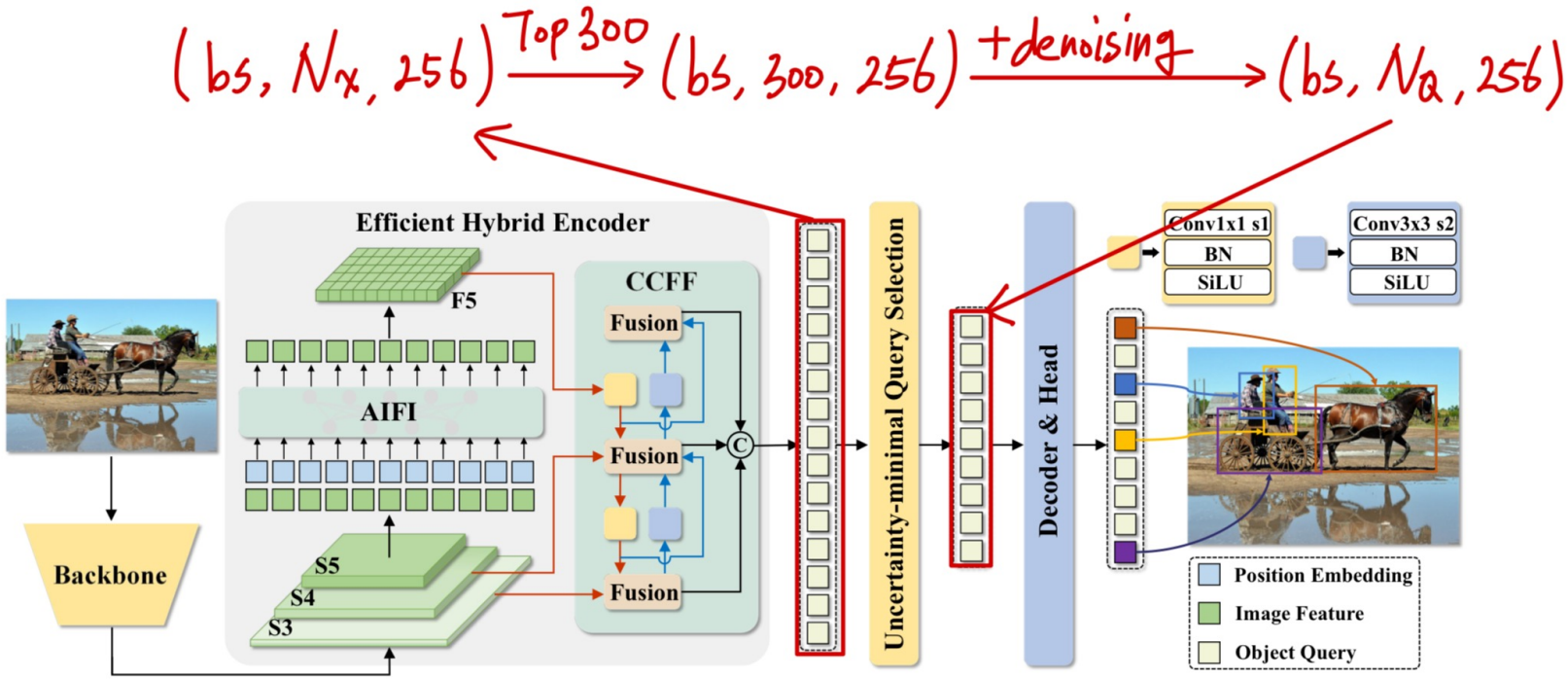

```python
def _get_decoder_input(self,      You, 2주 전 • RT-DETR setup
                       memory,
                       spatial_shapes, # [[4H, 4W], [2H, 2W], [H, W]]
                       denoising_class=None,
                       denoising_bbox_unact=None):


    # memory : encoder ouptut feature map
    bs, _, _ = memory.shape
    '''
        N_X = 4H*4W + 2H*2W + H*W
        memory : encoder ouptut
            (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

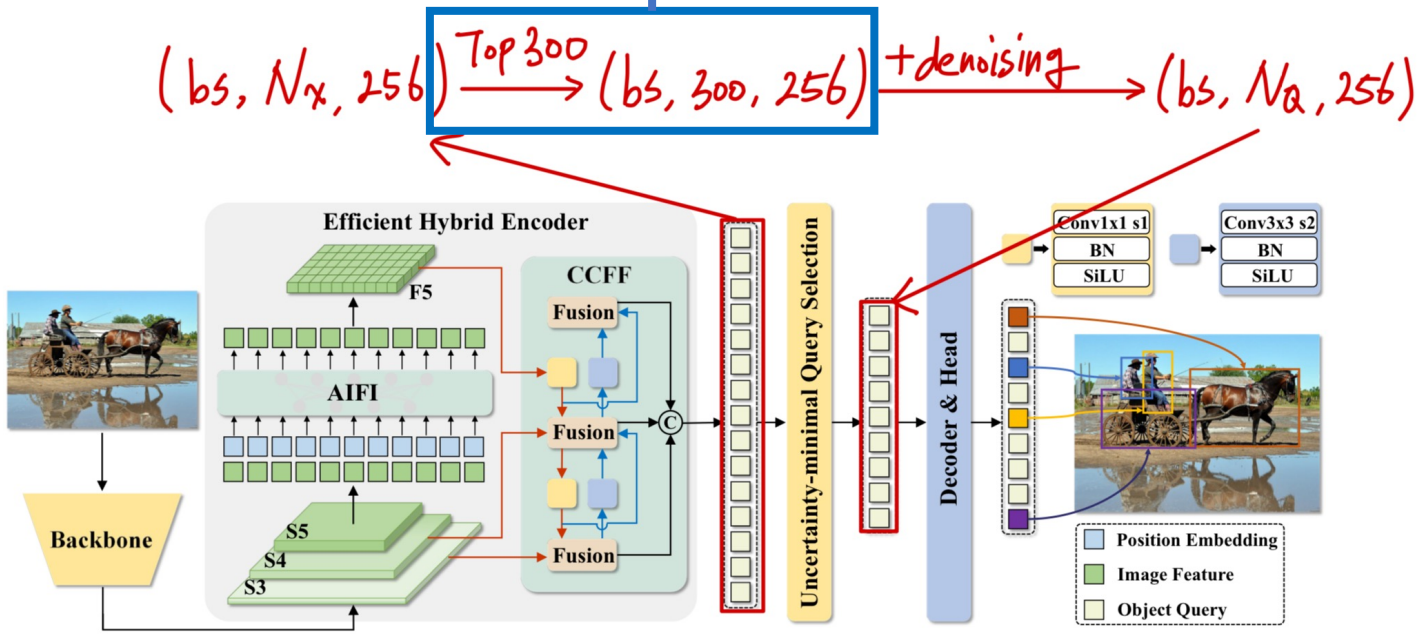
- Encoder의 output feature map의 uncertainty를 최소화할 수 있도록 topK에 대한 loss를 추가

feature uncertainty localization classification Distribution

$$\mathcal{U}(\hat{\boldsymbol{x}}) = \|\mathcal{P}(\hat{\boldsymbol{x}}) - \mathcal{C}(\hat{\boldsymbol{x}})\|, \hat{\boldsymbol{x}} \in \mathbb{R}^D \quad (2)$$

Pred GT

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



RT DETR : Uncertainty-minimal Query Selection

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

```python
class RTDETRTransformer(nn.Module):
    def forward(self, feats, targets=None):
        target, init_ref_points_unact, enc_topk_bboxes, enc_topk_logits = \      You, 2주 전 • RT-DETR setu
            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
                #                                      6번째 decoder layer는 aux_outputs이 아니라 최종 output임
            out['aux_outputs'].extend(self._set_aux_loss([enc_topk_logits], [enc_topk_bboxes]))
                # -> 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
```

```python
# In case of auxiliary losses, we repeat this process with the 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)
```

```python
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

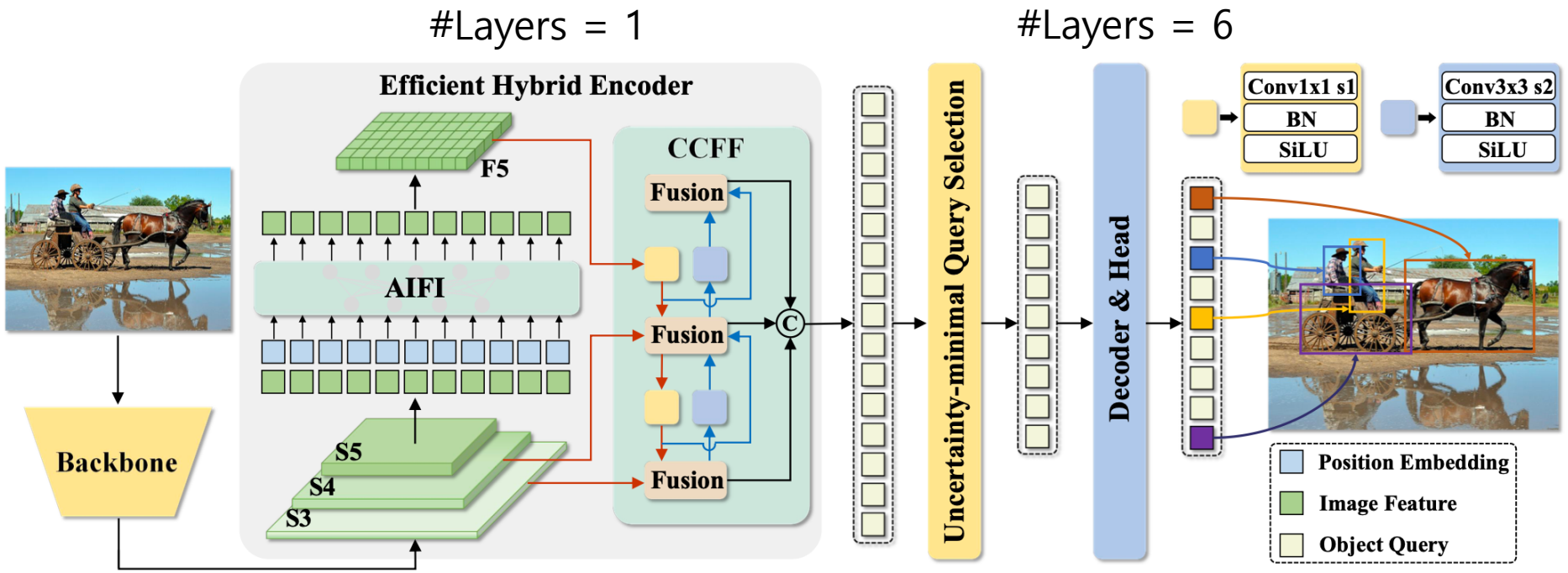
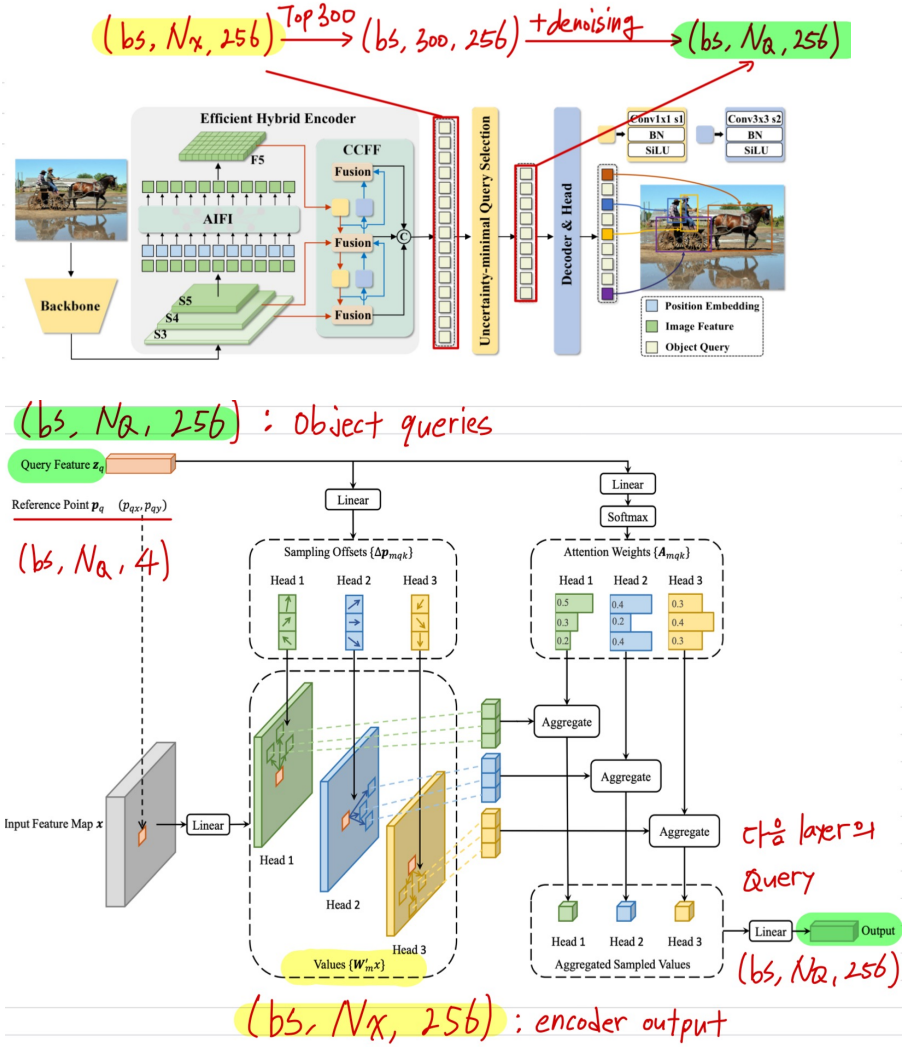


Figure 4. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale 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    (self_atten)
5457      (Query = object queries) tgt : torch.Size([4, 480, 256])   (bs, NQ, 256)
5458      (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5459      (Value = memory) memory : torch.Size([4, 11109, 256])      (bs, NX, 256)
5460      (self_atten output) tgt : torch.Size([4, 480, 256])
5461
5462    (cross-attention) MSDeformableAttention
5463      (Query = object queries) tgt : torch.Size([4, 480, 256])
5464      (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5465      (Value = memory) memory : torch.Size([4, 11109, 256])
5466      (cross_atten output = next layer Query) tgt :  torch.Size([4, 480, 256])
```

```
5606  (decoder layer 5) :
5607
5608    (self_atten)
5609      (Query = object queries) tgt : torch.Size([4, 480, 256])
5610      (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5611      (Value = memory) memory : torch.Size([4, 11109, 256])
5612      (self_atten output) tgt : torch.Size([4, 480, 256])
5613
5614    (cross-attention) MSDeformableAttention
5615      (Query = object queries) tgt : torch.Size([4, 480, 256])
5616      (ref_points_input) ref_points_input : torch.Size([4, 480, 1, 4])
5617      (Value = memory) memory : torch.Size([4, 11109, 256])
5618      (cross_atten output = next layer Query) tgt :  torch.Size([4, 480, 256])
```

RT DETR : Experiments

- **Comparison with SOTA**

anchor – based < anchor – free
faster, more accurate

Model	Backbone	#Epochs	#Params (M)	GFLOPs	$FPS_{bs=1}$	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
Real-time Object Detectors											
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
End-to-end Object Detectors											
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
Real-time End-to-end Object Detector (ours)											
RT-DETR	R50	72	42	136	**108**	**53.1**	**71.3**	**57.7**	34.8	58.0	70.0
RT-DETR	R101	72	76	259	**74**	**54.3**	**72.7**	58.6	36.0	58.8	**72.1**

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**

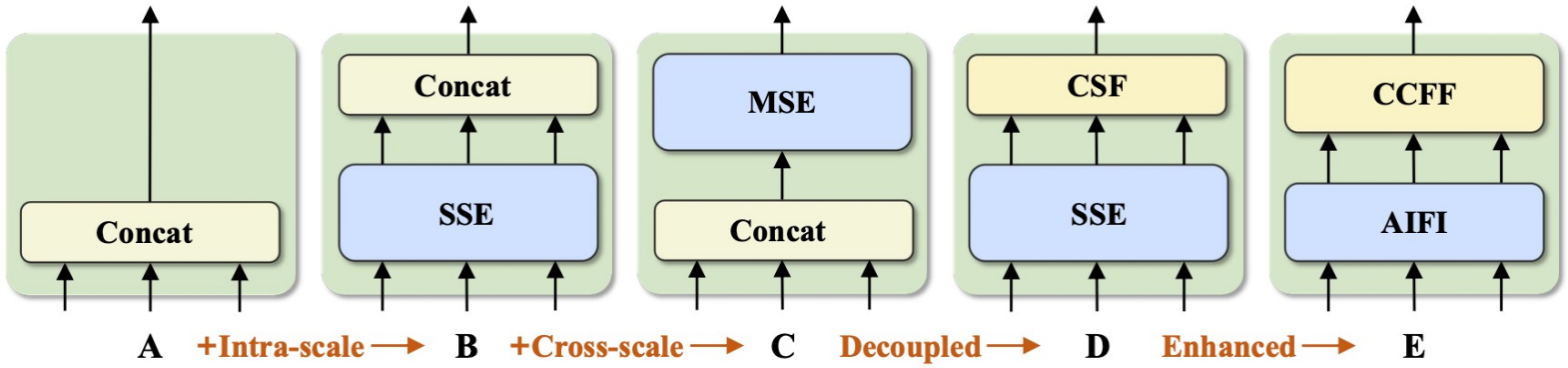


Figure 3. The encoder structure for each variant. **SSE** represents the single-scale Transformer encoder, **MSE** represents the multi-scale Transformer encoder, and **CSF** represents cross-scale fusion. **AIFI** and **CCFF** are the two modules designed into our hybrid encoder.

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

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

C : multi-scale features들을 Concat 후 encoder

D_{S_5} : S_5 single-scale encoder + cross-scale fusion(PANet)

E : D_{S_5} + 발전된 efficient hybrid encoder (CCFF + AIFI)

RT DETR : Experiments

- **Ablation Study on Decoder**

distillation

ID	AP(%)				Latency (ms)
	Det^4	Det^5	Det^6	Det^7	
7	-	-	-	52.6	9.6
6	-	-	**53.1**	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. $\textbf{Det}^k$ 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