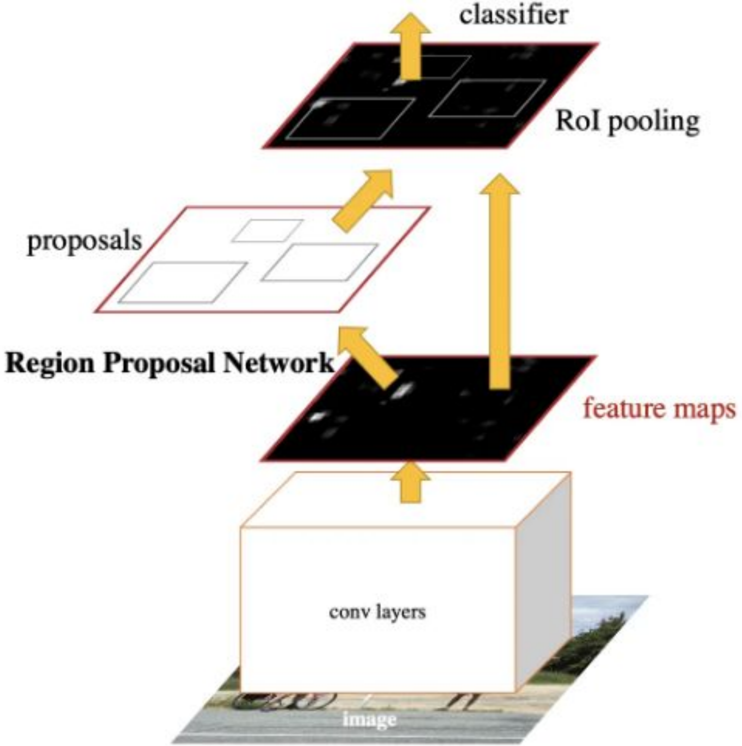
End-to-End Object Detection with Transformers

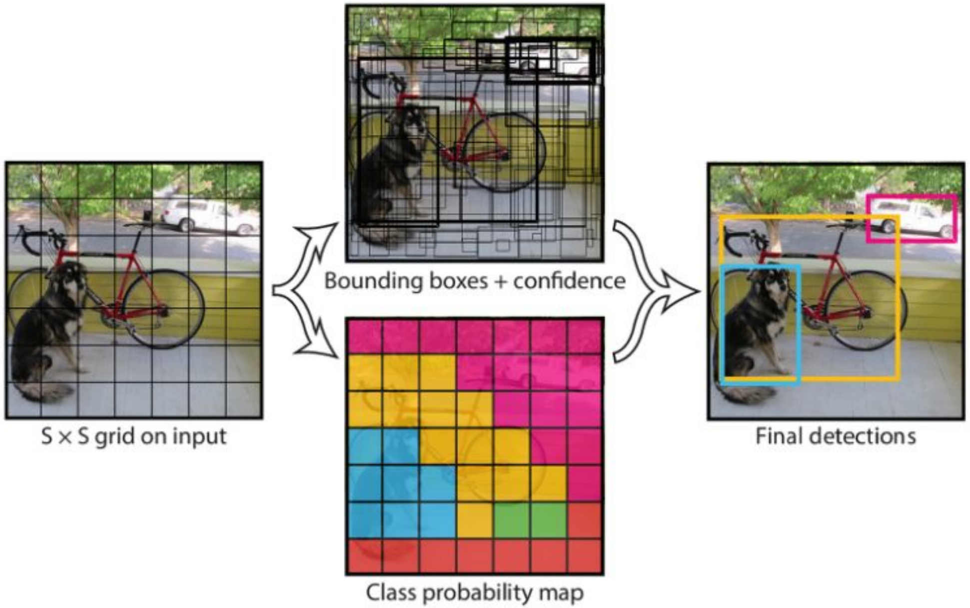
Nicolas Carion et al., ECCV 2020

Youngwoo Kim

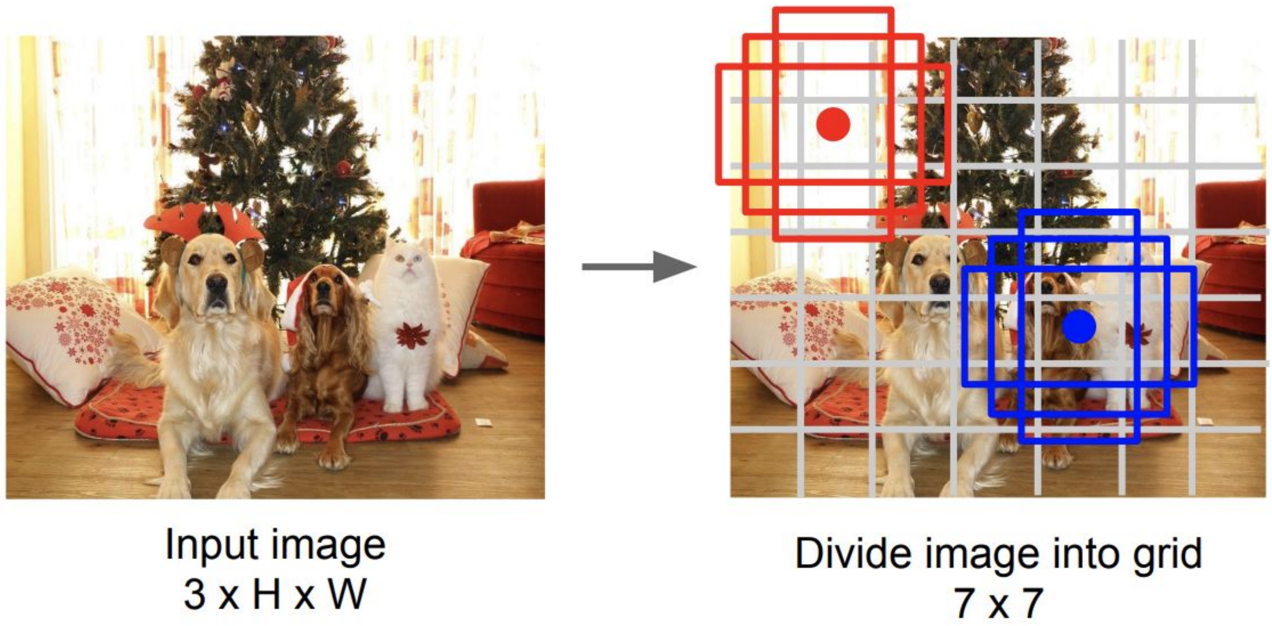
DETR are a **set-based** global loss that forces unique predictions via **bipartite matching** and a **transformer encoder-decoder architecture**.

Conventional Object Detection

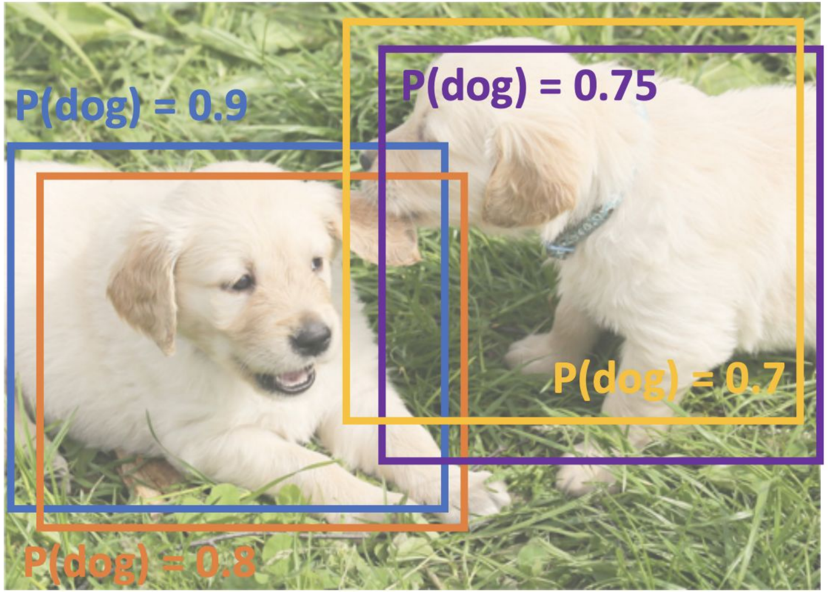


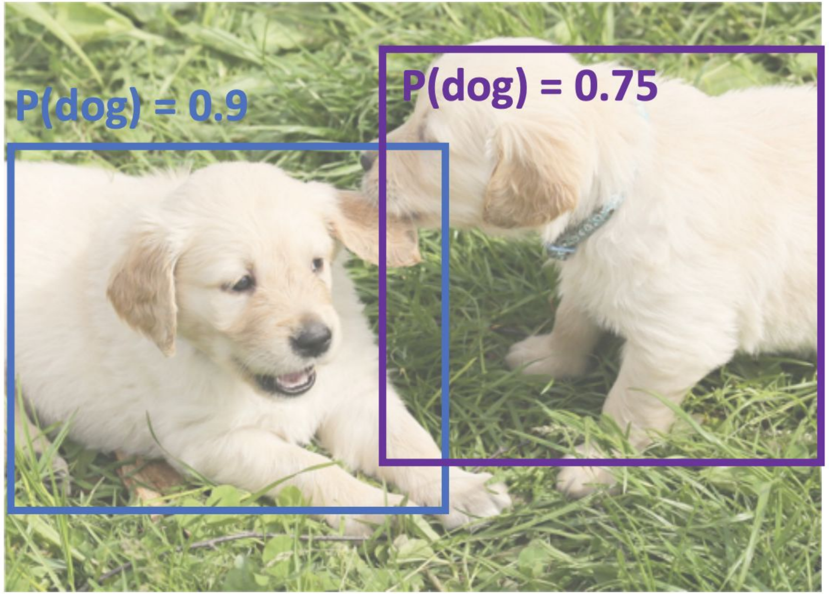


Anchor Box

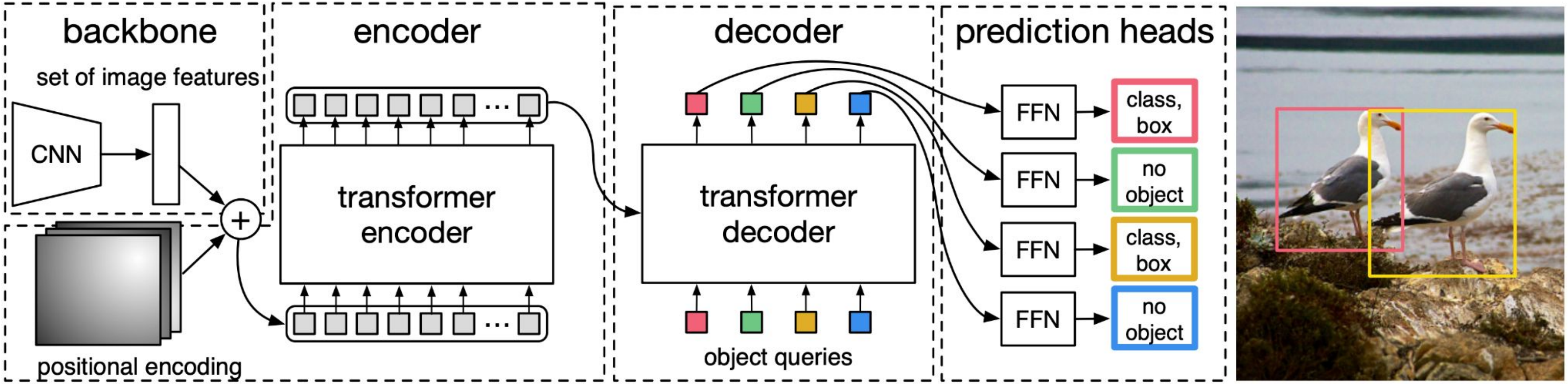


Non Maximum Suppression





DETR



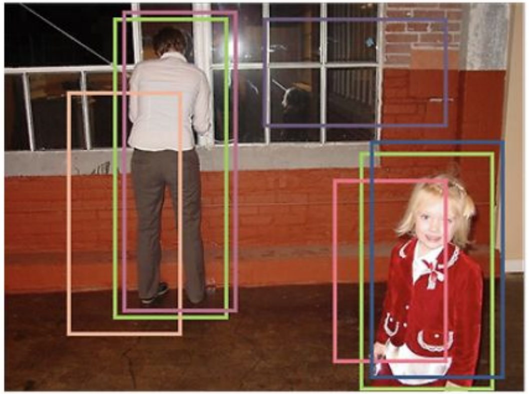
1. Bipartite matching
2. Transformer encoder-decoder

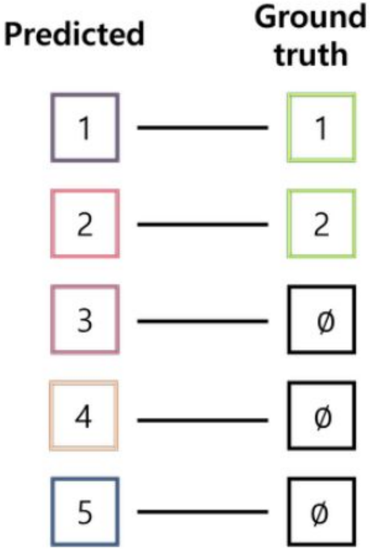
```python
import torch
from torch import nn
from torchvision.models import resnet50

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)
```

Find Optimal Matching

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

Hungarian Algorithm

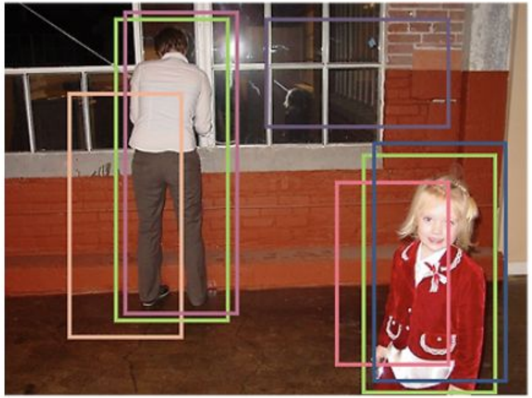


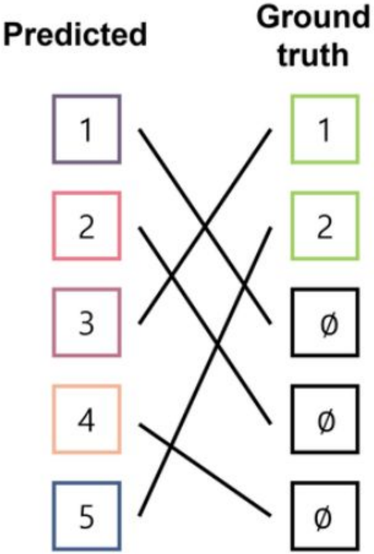


	1	2	∅	∅	∅
1	12	11	1	1	1
2	4	2	8	6	9
3	1	3	5	7	8
4	2	5	6	7	9
5	2	1	9	10	6

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

Hungarian Algorithm





	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

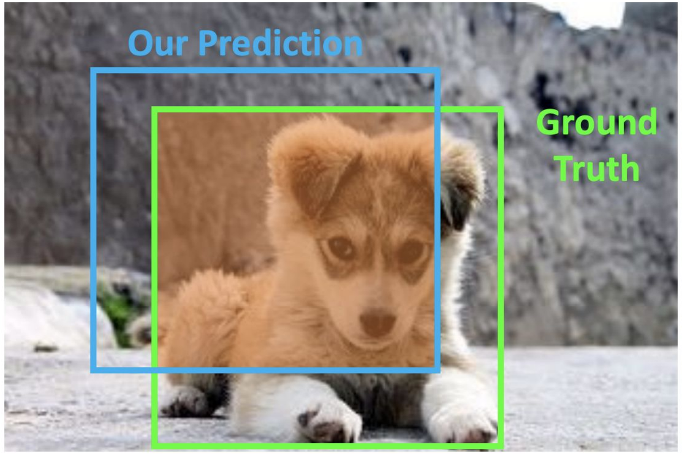
Bounding Box Loss

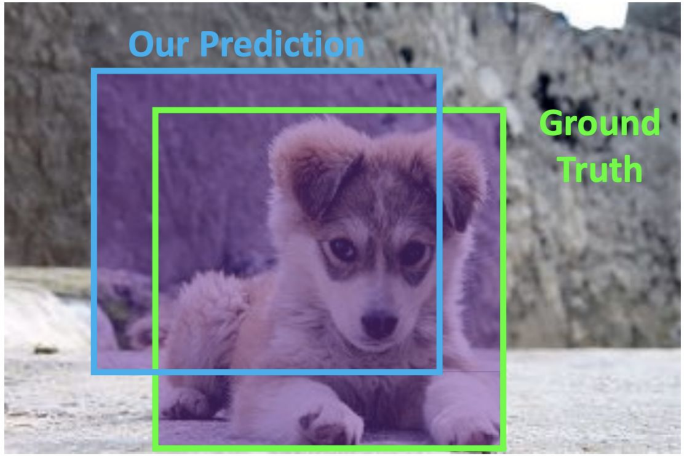
$$\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}(i)}) = \lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}}||b_i - \hat{b}_{\sigma(i)}||_1$$

Intersection over Union

$$\frac{\text{Area of Intersection}}{\text{Area of Union}}$$

IoU > 0.5 is “decent”,
IoU > 0.7 is “pretty good”,
IoU > 0.9 is “almost perfect”





Generalized IoU

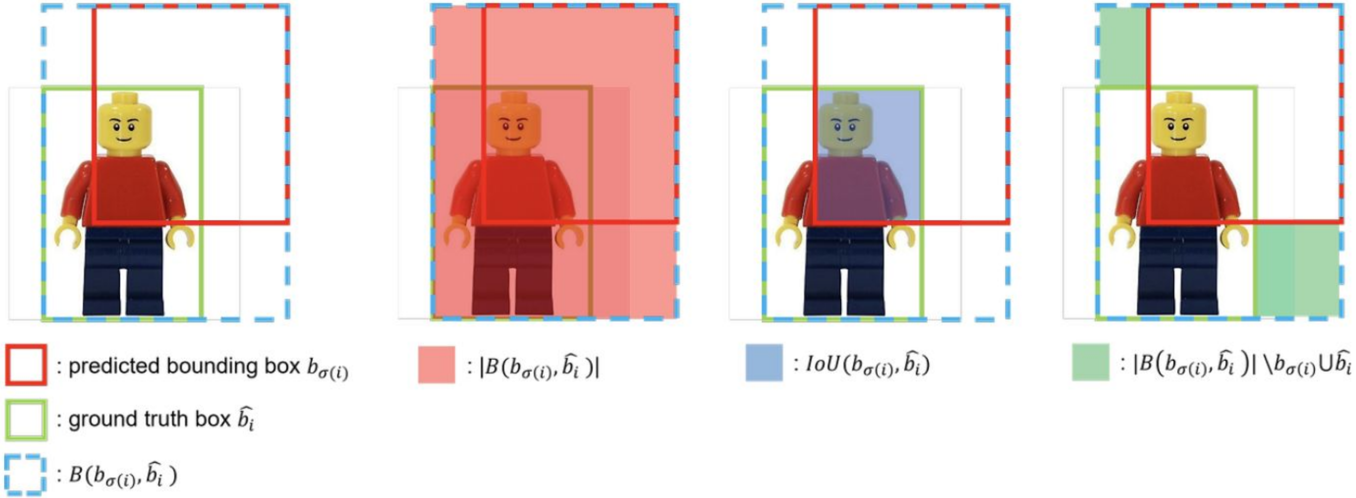


Fig 5. GIoU loss

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

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

Compute Hungarian Loss

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

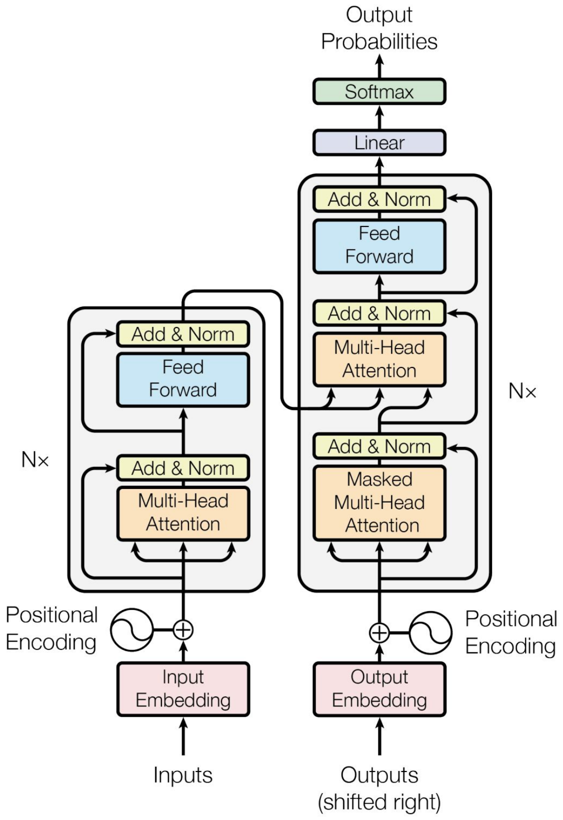
Backbone (Conventional CNN)

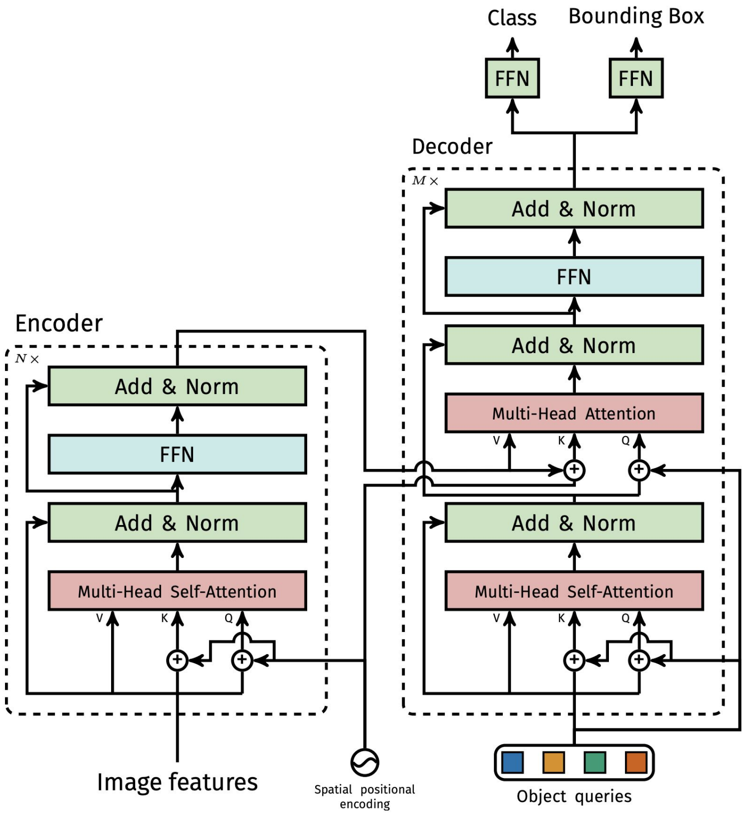
$$x_{\text{img}} \in \mathbb{R}^{3 \times H_0 \times W_0}$$

$$f \in \mathbb{R}^{C \times H \times W}$$

$$C = 2048 \quad H, W = \frac{H_0}{32}, \frac{W_0}{32}$$

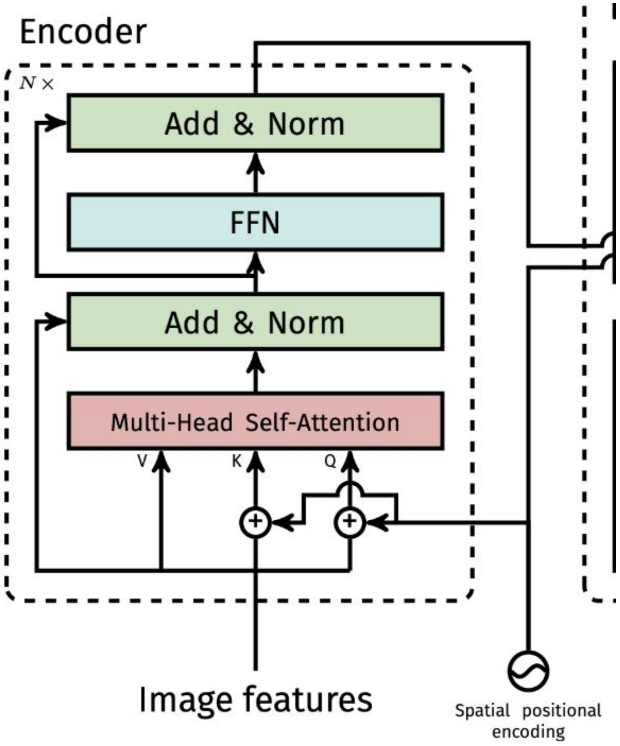
Transformer





Transformer Encoder

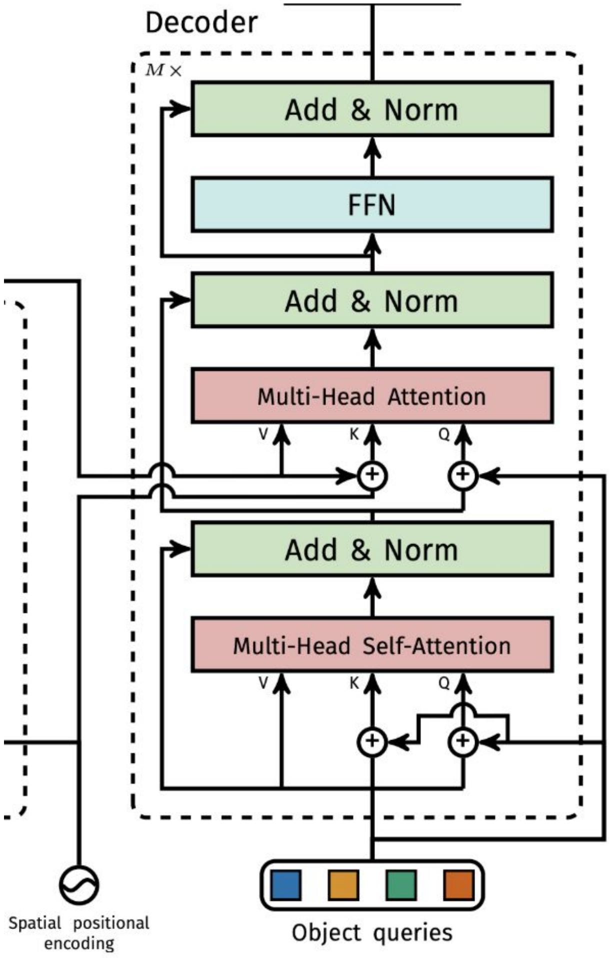
- Reduces the channel dimension of the feature map from C to a smaller dimension d = 256
- Flatten feature map to d x HW feature map for transformer
- Fixed positional encodings are added to the input of each attention layer



Transformer Decoder

- decodes the N objects in parallel at each decoder layer, instead of using an autoregressive model, which predicts the output sequence one element at a time.

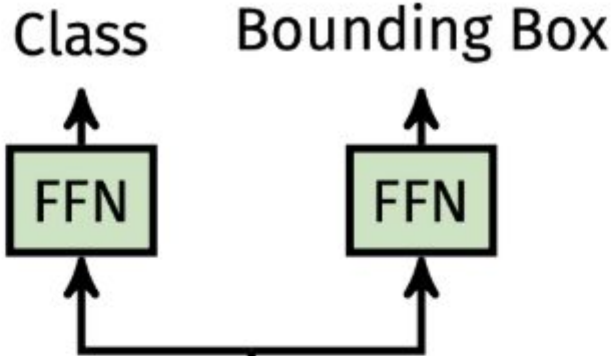
- Input embeddings, also known as object queries, are added to each attention layer



Prediction Feed Forward Networks

- Independently decodes object queries into bounding box coordinates and class labels

- The value of NNN is typically much larger than the actual number of objects



Faster R-CNN vs DETR

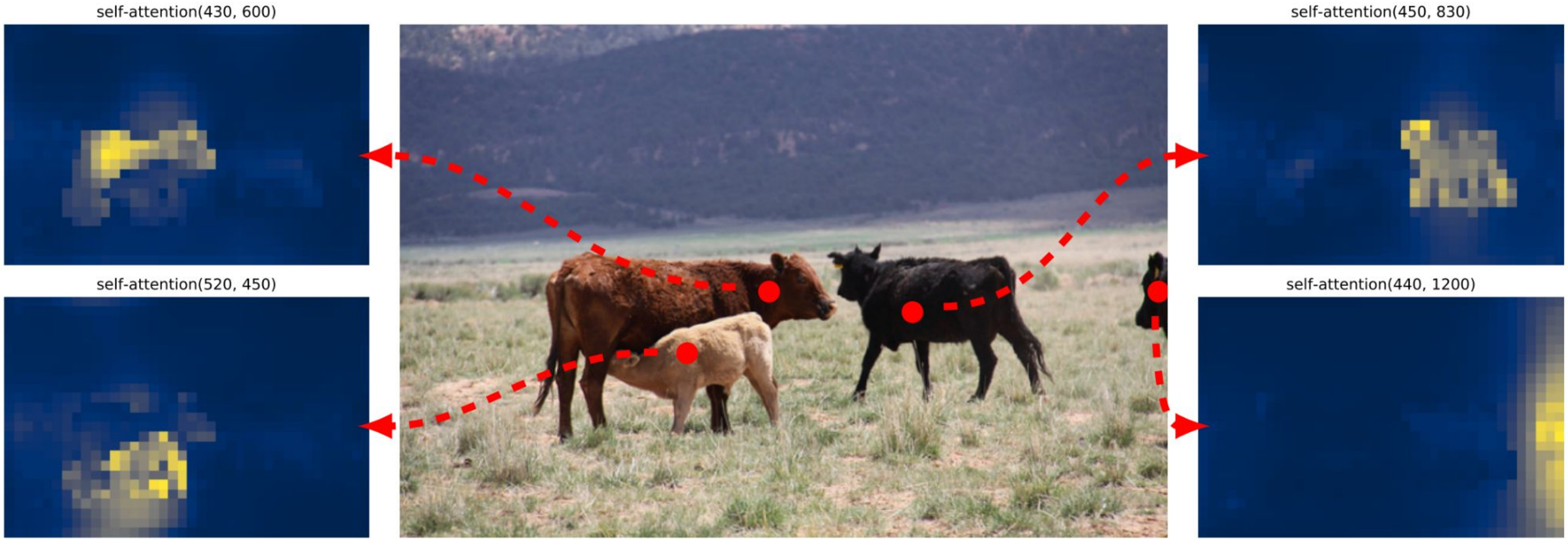
Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster 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**

Number of encoder layers

- Without encoder layers, overall AP drops by 3.9
- AP on large object drops by 6.0

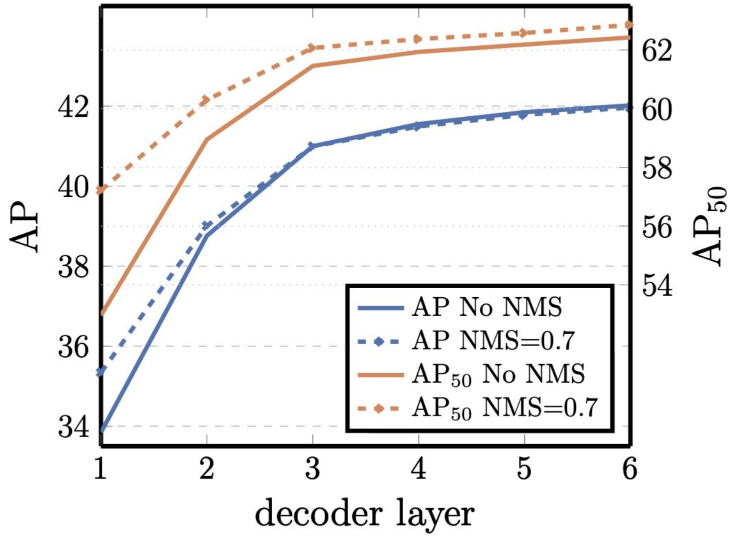
#layers	GFLOPS/FPS	#params	AP	AP_{50}	AP_S	AP_M	AP_L
0	76/28	33.4M	36.7	57.4	16.8	39.6	54.2
3	81/25	37.4M	40.1	60.6	18.5	43.8	58.6
6	86/23	41.3M	40.6	61.6	19.9	44.3	60.2
12	95/20	49.2M	41.6	62.1	19.8	44.9	61.9

Visualizing encoder attention

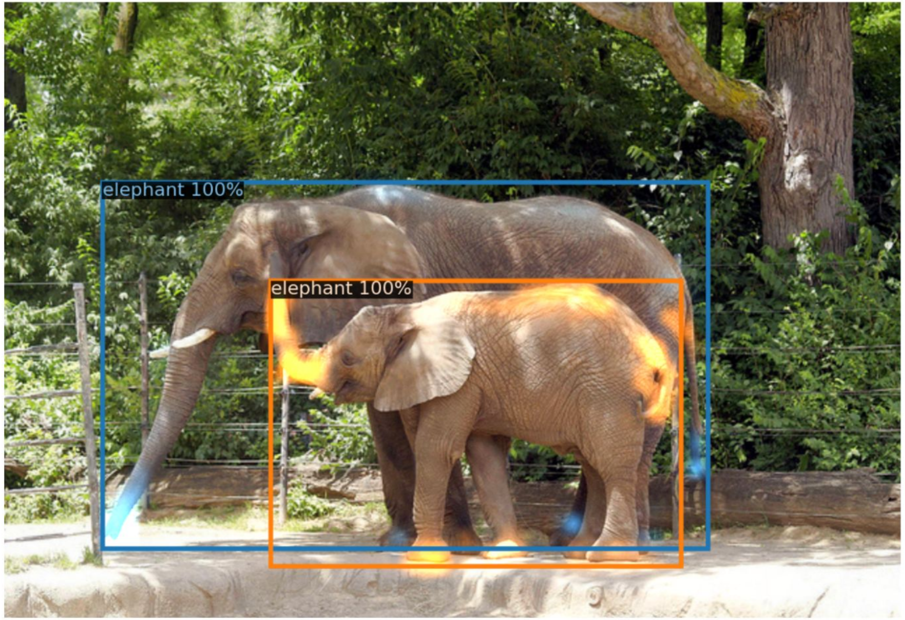


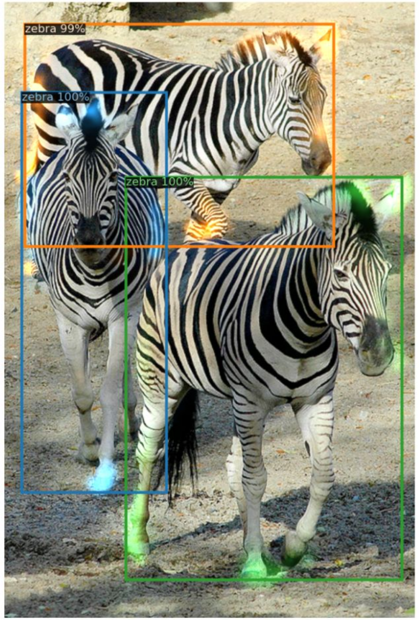
Number of decoder layers

- As decoder layers increase, DETR performs good enough without NMS
- Both improved totalling into +8.2/9.5 AP between the first and the last layer



Visualizing decoder attention





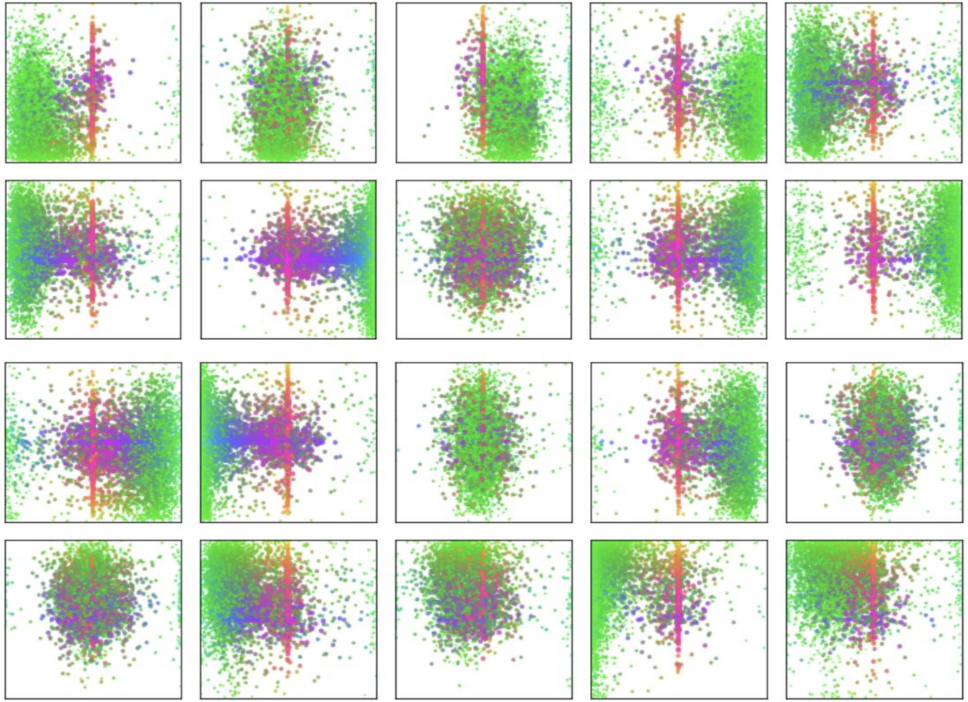
Ablation - Loss

Table 4: Effect of loss components on AP. We train two models turning off ℓ_1 loss, and GIoU loss, and observe that ℓ_1 gives poor results on its own, but when combined with GIoU improves AP_M and AP_L. Our baseline (last row) combines both losses.

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**

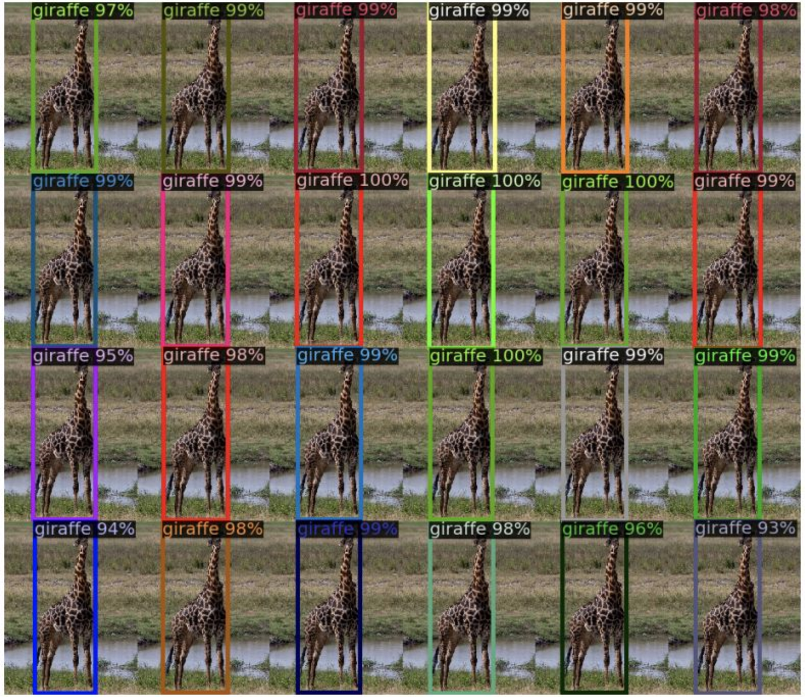
Decoder output slot analysis

- DETR learns different specialization for each query slot

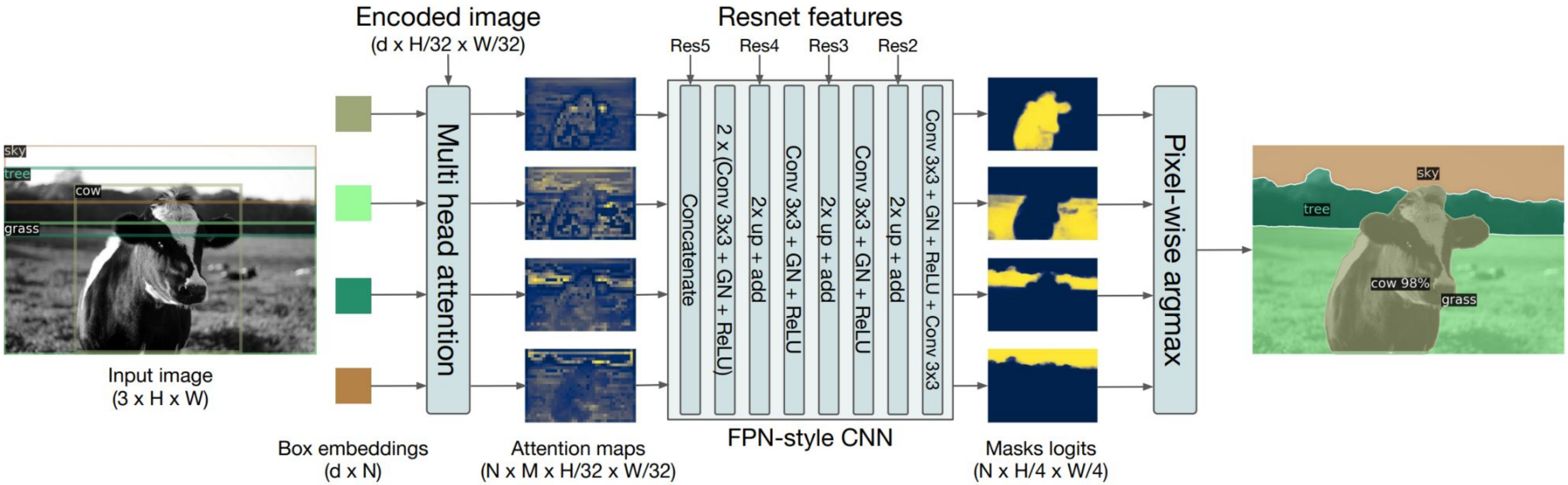


Generalization to unseen numbers of instances

- No image with more than 13 giraffes in the training set
- Able to detect all 24 giraffes
- Confirms that there is no strong class-specialization in each object query



DETR for panoptic segmentation



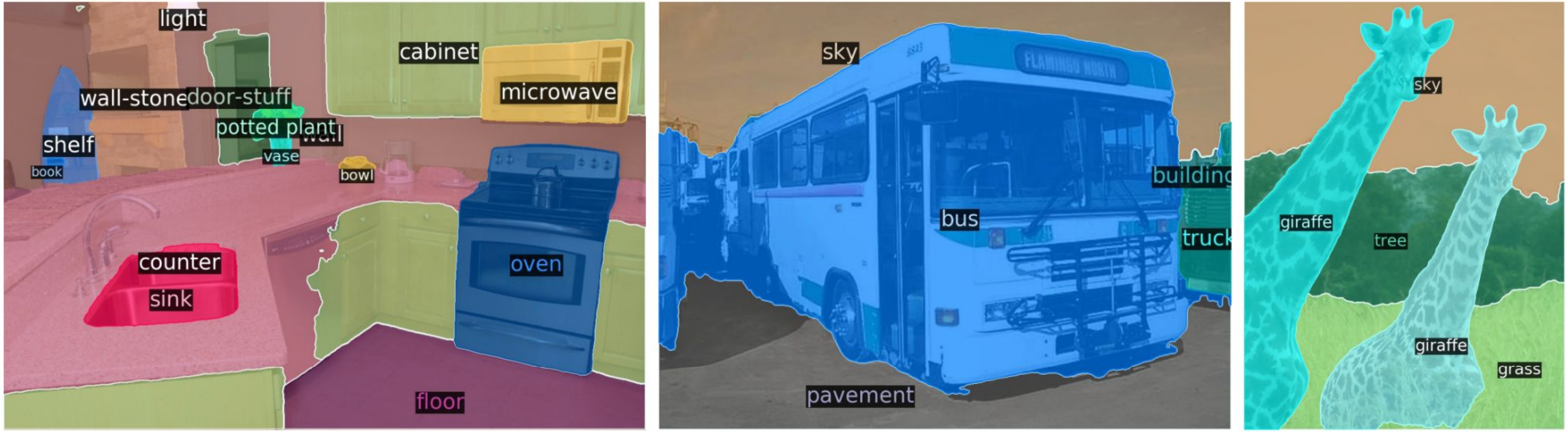

light
cabinet
wall-stone
door-stuff
microwave
potted plant
wall
shelf
vase
book
bowl
counter
oven
sink
floor
sky
bus
buildin
truck
pavement
sky
giraffe
tree
giraffe
grass

Table 5: Comparison with the state-of-the-art methods UPSNet [51] and Panoptic FPN [18] on the COCO `val` dataset We retrained PanopticFPN with the same data-augmentation as DETR, on a 18x schedule for fair comparison. UPSNet uses the `1x` schedule, UPSNet-M is the version with multiscale test-time augmentations.

Model	Backbone	PQ	SQ	RQ	PQ^{th}	SQ^{th}	RQ^{th}	PQ^{st}	SQ^{st}	RQ^{st}	AP
PanopticFPN++	R50	42.4	79.3	51.6	49.2	82.4	58.8	32.3	74.8	40.6	37.7
UPSnet	R50	42.5	78.0	52.5	48.6	79.4	59.6	33.4	75.9	41.7	34.3
UPSnet-M	R50	43.0	79.1	52.8	48.9	79.7	59.7	34.1	78.2	42.3	34.3
PanopticFPN++	R101	44.1	79.5	53.3	**51.0**	**83.2**	60.6	33.6	74.0	42.1	**39.7**
DETR	R50	43.4	79.3	53.8	48.2	79.8	59.5	36.3	78.5	45.3	31.1
DETR-DC5	R50	44.6	79.8	55.0	49.4	80.5	60.6	**37.3**	**78.7**	**46.5**	31.9
DETR-R101	R101	**45.1**	**79.9**	**55.5**	50.5	80.9	**61.7**	37.0	78.5	46.0	33.0

Conclusion

- DETR is a new design for object detection systems based on transformers and bipartite matching loss for direct set prediction
- Despite some challenges, DETR successfully offers an end-to-end object detection system that does not require post-processing steps.
- Achieves comparable results to an optimized Faster R-CNN baseline on COCO dataset