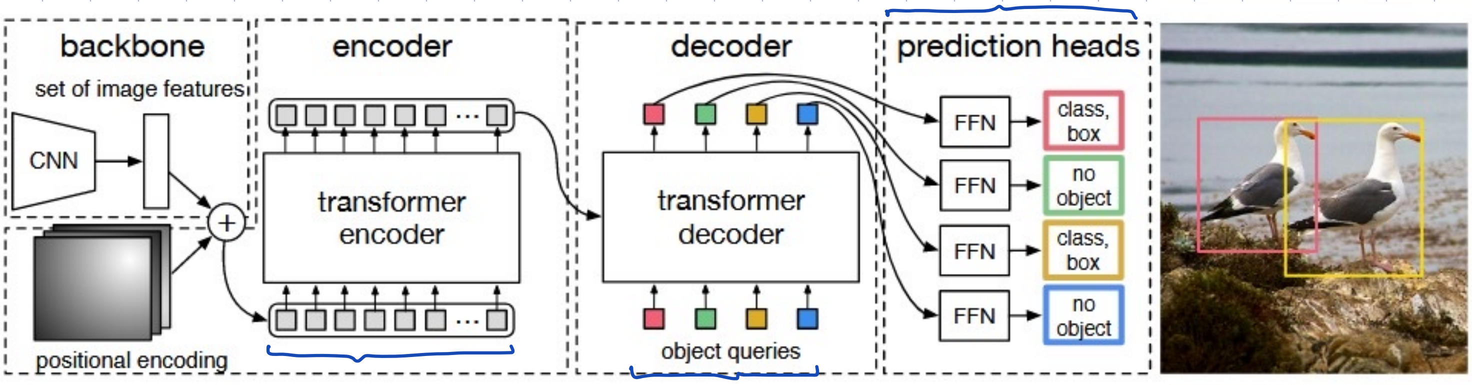
Detr:

↳ loss with last decoder layer
↳ minimize it with Grd truth

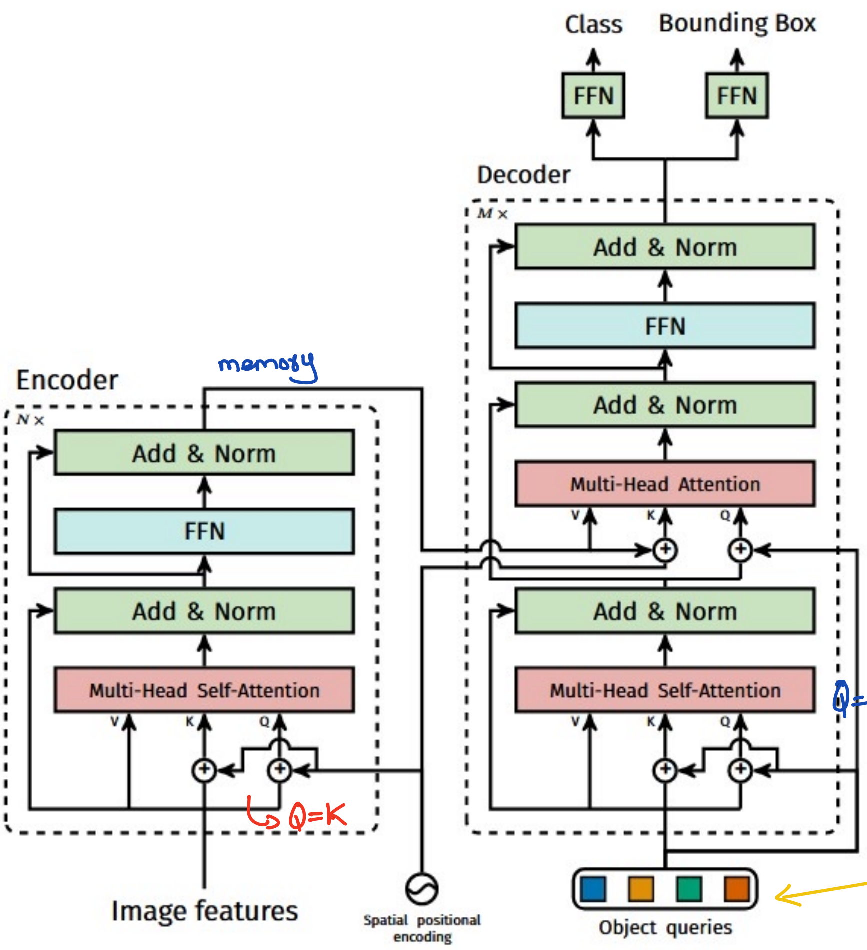
↳ Distinct bi-partite matching



↳ flatten image with positional encoding

query_embed → learnable
tgt → set to 0

↳ Actual loss that's used for backprop is from each decoder layer's output



memory

↳ Q=K

evolving state (what has been found so far)

set to 0

Q=K= query_embed + tgt

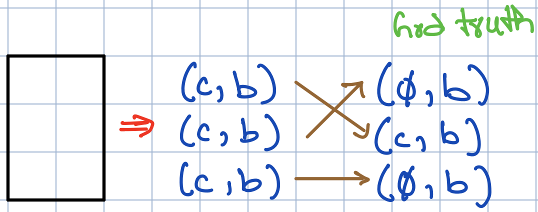
nn.embedding
↳ learnable
(what to look for)

Serve as anchor points to be added at each decoder layer.

Why do this?

- Adding the query embedding at each layer creates a form residual connection.

These help with gradient flow. Without this, the network might forget the intial query information.



match left to right where L is minimum.

$$\hat{\sigma} = \operatorname{argmin} \sum_{i}^{N} \mathcal{L}_{match}\left(y_i, \hat{y}_{\sigma(i)}\right)$$

↳ ith assignment

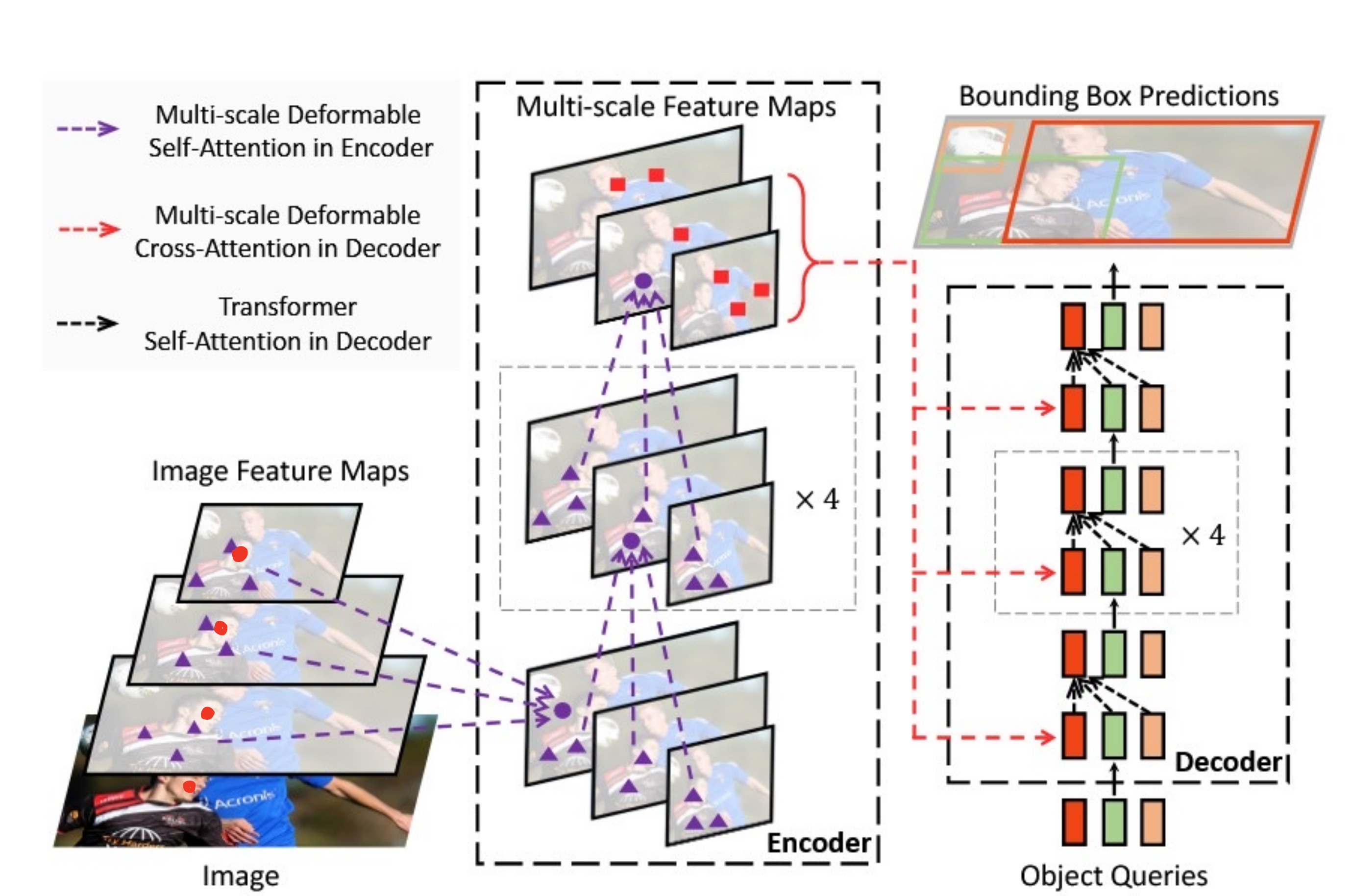
$$\mathcal{L}_{match} = -\mathbb{1}_{\{c_i \neq \emptyset\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}\left(b_i, \hat{b}_{\hat{\sigma}(i)}\right)$$

Actual loss for training:

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

We do this for each decoder layer.

Deformable Detr:



- flatten the four level image into a sequence.
- Add level & positional encoding to it → pos = level + positional encoding

→ src

A reference point
- Find it at all levels
- Pass this (src + pos) through a Linear layer to get 4 offsets & Attention weights.
- Do this for each level
- Get one output from each level
- Sum them & pass through a linear layer to get final output
- Repeat for each attention head

$$\mathrm{MSDeformAttn}(\boldsymbol{z}_q, \hat{\boldsymbol{p}}_q, \{\boldsymbol{x}^l\}_{l=1}^{L}) = \sum_{m=1}^{M} \boldsymbol{W}_m \Big[\sum_{l=1}^{L} \sum_{k=1}^{K} A_{mlqk} \cdot \boldsymbol{W}'_m \boldsymbol{x}^l (\phi_l(\hat{\boldsymbol{p}}_q) + \Delta \boldsymbol{p}_{mlqk}) \Big], \quad (3)$$

For query,

↳ Same process is done

↳ Except the reference point is on the image & found through a linear layer.

For a single scale:

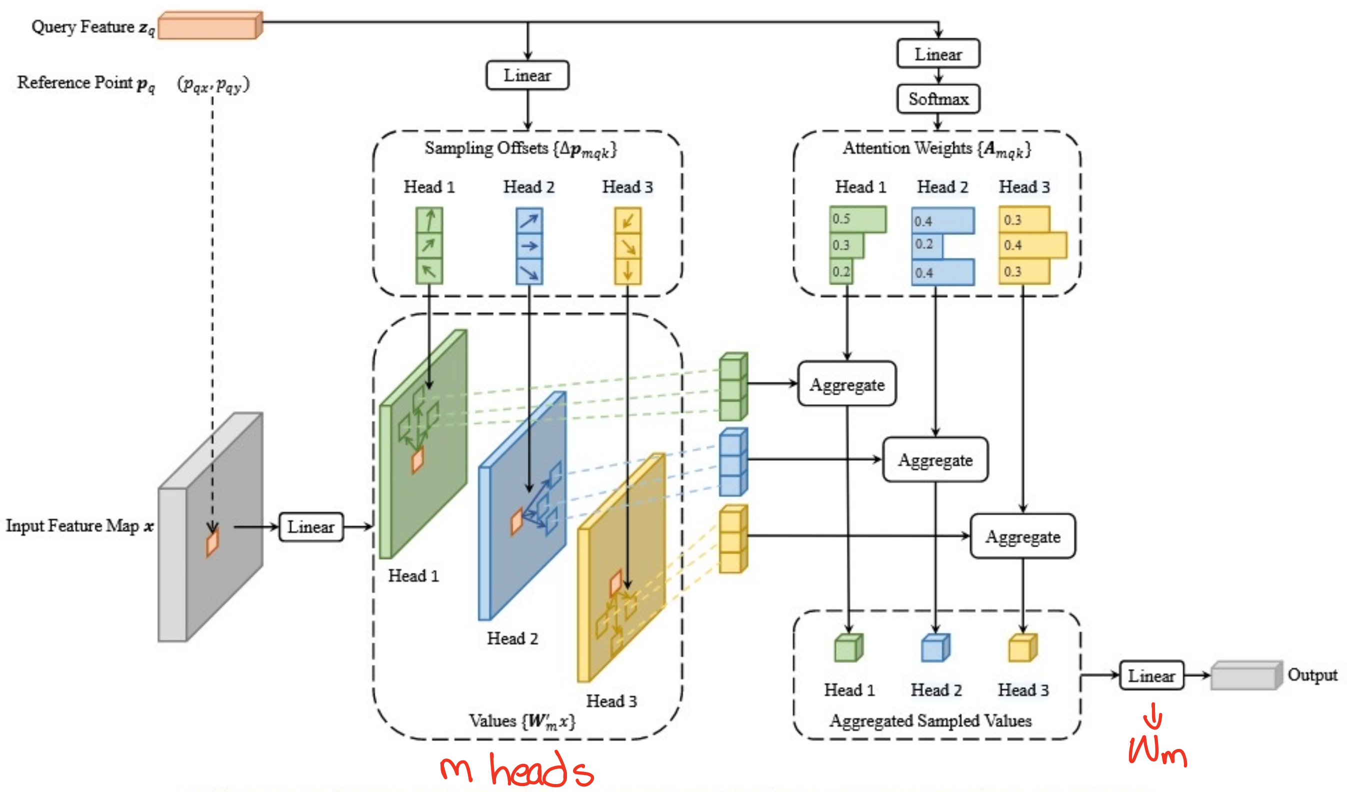


Figure 2: Illustration of the proposed deformable attention module.

Dab - Detr:

↳ The authors recognized the object queries as slowing convergence.

↳ From previous work, they know that cross-attn module is the one slowing the convergence

↳ Since later steps are same in encoder & de-coder, they conclude the issue is likely from queries.

↳ To find the reason, they have two hypothesis

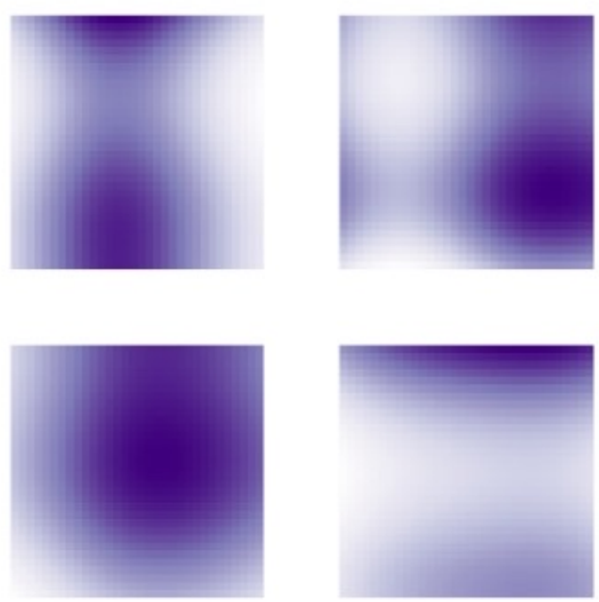
1. The Queries are hard to learn

↳ So, they fix learned Queries & train other modules & find that the improvement is minimal. So, this can't be it

2. The positional information in the queries is not encoded the same way as sinusoidal positional encoding used for image features.

↳ While sinusoidal encoding provides a structured & consistent way to represent positions for the image features from the start, the learnable queries have to gradually discover useful positional representations during training, which might slow down convergence.

↳ furthermore, the model learns bad queries still



(a) DETR

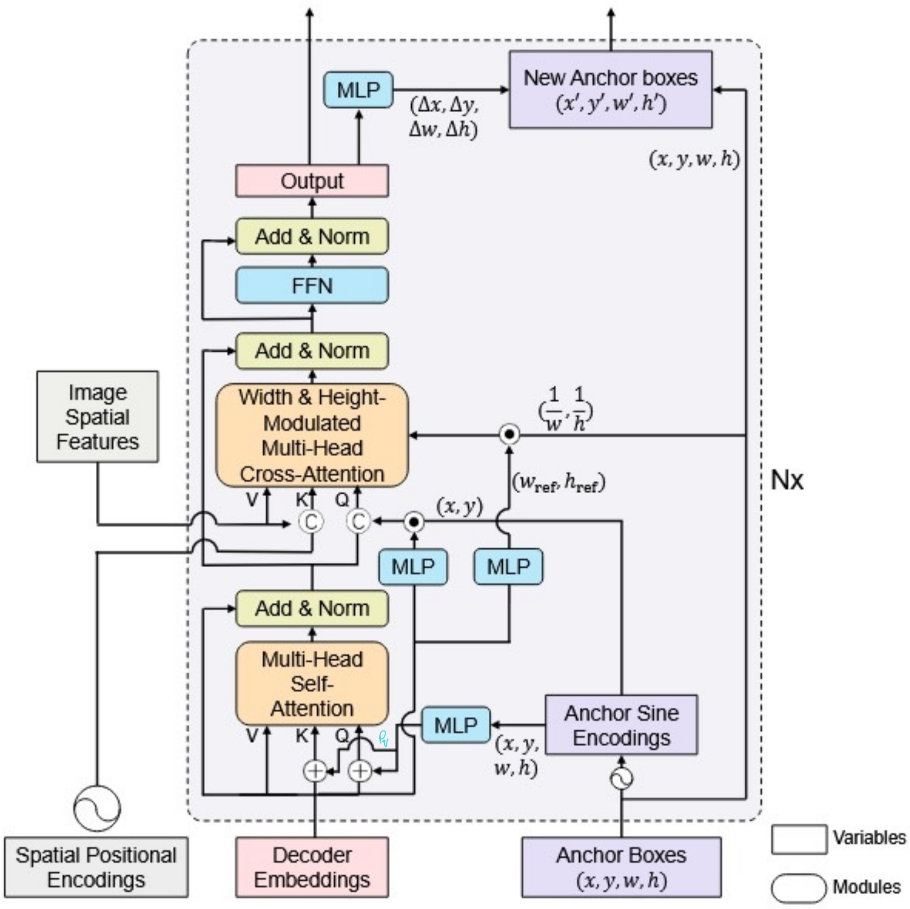
↳ Computed as learned queries $\cdot$ Positional encoding of image

↳ All over the place

↳ No sense of scale

↳ No sense of importance of certain points more than others (Uniform)

↳ Query embeddings are needed to compute attention & they are NOT doing a good job at it.



↳ The Anchor boxes are added to output prediction of each decoder layer.

↳ This way anchor boxes are also learned

↳ They serve as priors for the position, size, shape of bbox of obj.

Content queries (Decoder Embeddings)
&
Positional queries (Anchor Boxes)
are fed to Decoder as dual queries.

↳ The Anchor boxes are learned directly & derive positional queries from these anchors.

Self Attn: $Q_q = C_q + P_q$, $K = C_q + P_q$, $V_q = C_q$

Cross-Attn: $Q_q = \mathrm{Cat}\left[C_q, PE(x_q, y_q)\right] \cdot MLP^{(csq)}(C_q)$

$K_{x,y} = \mathrm{Cat}\left(F_{x,y}, PE(x,y)\right)$, $V_{x,y} = F_{x,y}$

image feature at position (x,y)

In DETR models, when computing attention between queries and image features, both content and position information get mixed together. This makes it hard to know how much each aspect contributes to the similarity scores.

What they're doing is:

1. Separating content and position information by concatenating them rather than adding them
2. This lets the model learn to pay attention to each aspect separately

The MLP(csq) part is addressing a scaling issue. Position information and content information might naturally have different magnitudes or importance. They're using the content information to produce a custom scaling factor for the position information.

Think of it like this: Some objects might need precise position information (like small objects), while others might rely more on visual appearance. This approach lets the model adjust how much it cares about position versus content for each query, making it more flexible