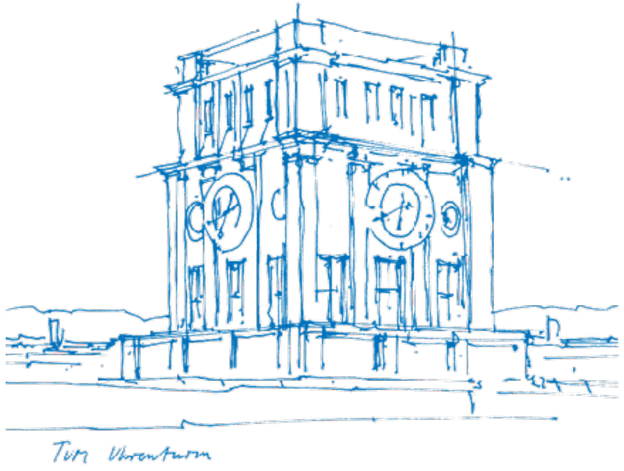
Computer Vision III:

Recent research: Transformers

Nikita Araslanov
17.01.2023

Adapted from:
Prof. Laura Leal-Taixé
https://dvl.in.tum.de



Course progress

1. Introduction
2. Object detection 1
3. Object detection 2
4. Multiple object tracking 1
5. Multiple object tracking 2
6. Semantic segmentation
7. Instance segmentation
8. Panoptic segmentation
9. Video object segmentation
10. Transformers (17.01) ← we are here
11. Unsupervised and semi-supervised scene understanding (24.01)
12. Q&A (February?)
13. Exam: 03.03

Deep Learning Evolution

	Deep Learning	Deep Learning 2.0
Main idea	Convolution	Attention
Field invented	Computer vision	NLP
Started	NeurIPS 2012	NeurIPS 2017
Paper	AlexNet	Transformers
Conquered vision	Around 2014-2015	Around 2020-2021
Replaced / Complement	Hand-crafted feature descriptors	CNNs, RNNs

CNNs: A few relevant facts

- CNNs encode local context information
- Receptive field increases monotonically with network depth.
  - This enables long-range context aggregation…
  - yet increased depth makes training more challenging.
- We may benefit from more explicit (i.e. in the same layer) non-local feature interactions.

Attention is all you need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Attention is all you need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Circa 45 thousand citations in 4.5 years!

Attention is all you need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

~~Circa 45 thousand citations in 4.5 years!~~

Circa 63 thousand citations in 5.5 years!

Attention is all you need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

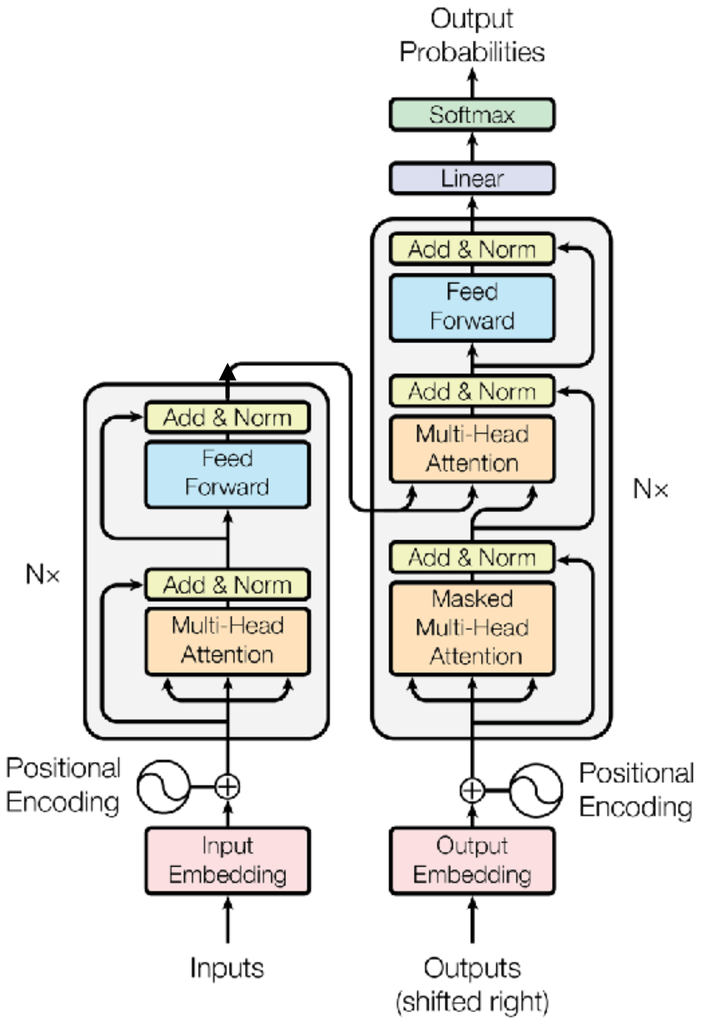
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com



So, what is so special about it?

Universality

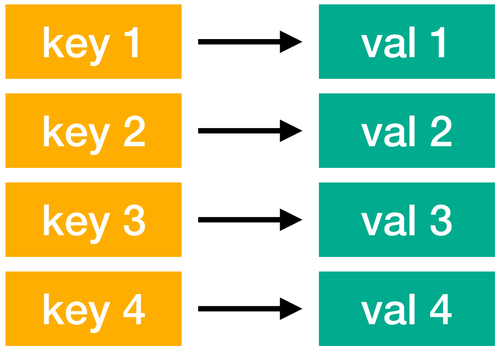
Universality: developing components that work across all possible settings.

- Consider other modalities (language, speech, etc.);
- Modern deep learning are only partially universal, even in the contexts we have seen so far (detection and segmentation);
- Transformers can be applied to language and images with excellent results in both domains!

[Adapted from: Vaswani et al., ICML '21]

Self-attention: A hash table analogy

Consider a hash table:

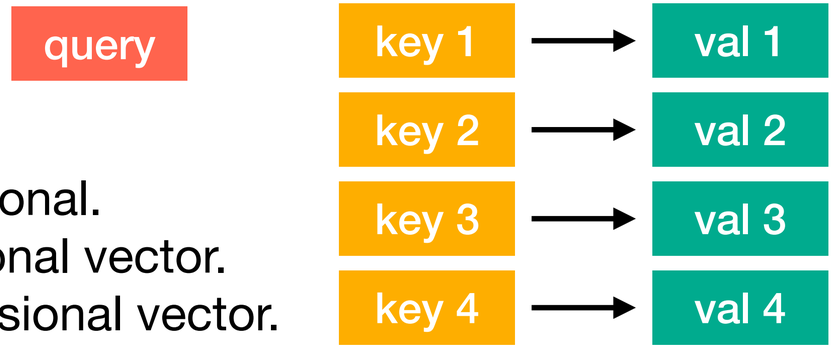


Each key is n-dimensional vector.
Each value is m-dimensional vector.

- A hash table is not a differentiable function (of values w.r.t. keys); it's not even continuous.
  - We can access any value if we know the corresponding key exactly.
- How can we make it differentiable? – "differentiable soft look-up"

Self-attention: A hash table analogy

Let's say, we have a query vector:



The query is n-dimensional.
Each key is n-dimensional vector.
Each value is m-dimensional vector.

- We can obtain an (approximate) value corresponding to the query:

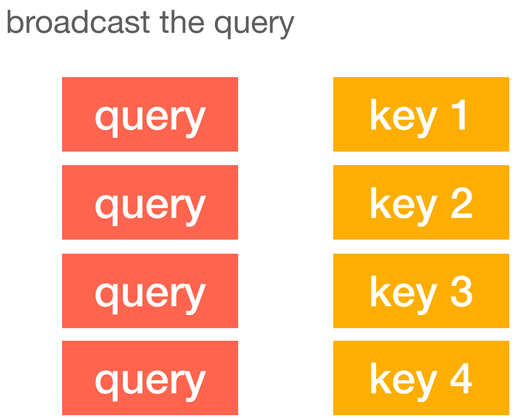
$$v'(q) = \sum_i w(q, k_i) v_i$$

- $w(q, k_i)$ encodes normalised similarity of the query with each key:

$$\sum_i w(q, k_i) = 1$$

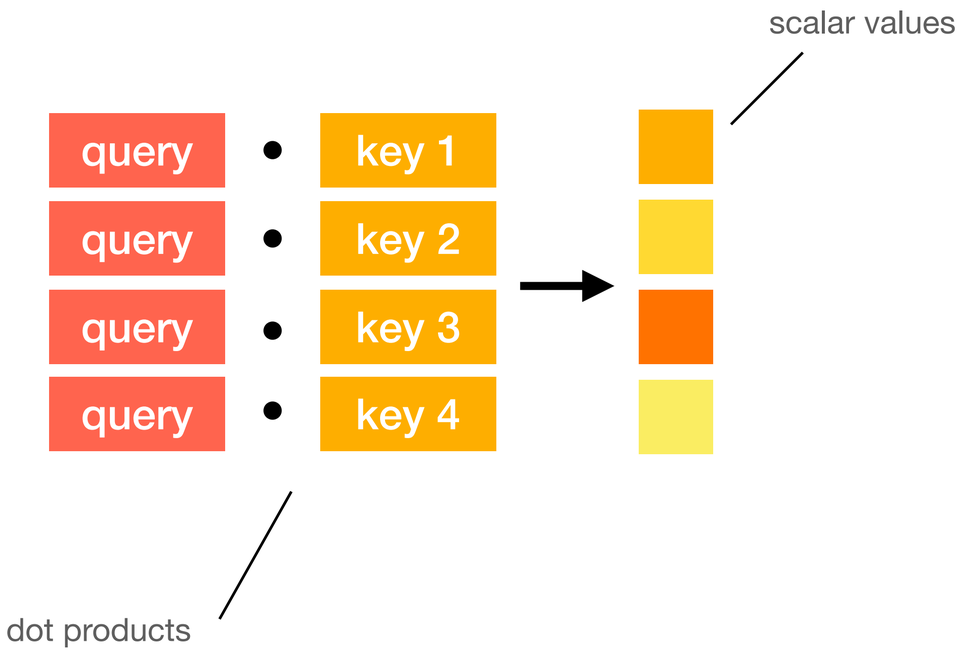
Self-attention: A hash table analogy

We can use standard operators in our DL arsenal: dot product and softmax



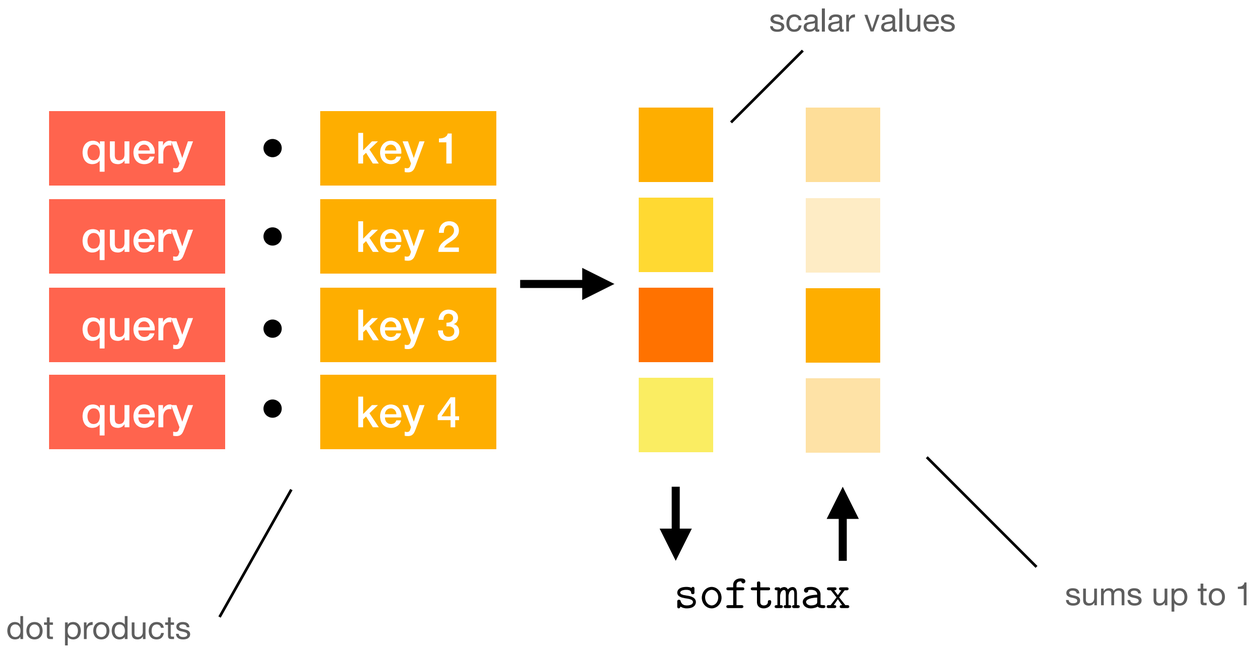
Self-attention: A hash table analogy

We can use standard operators in our DL arsenal: dot product and softmax



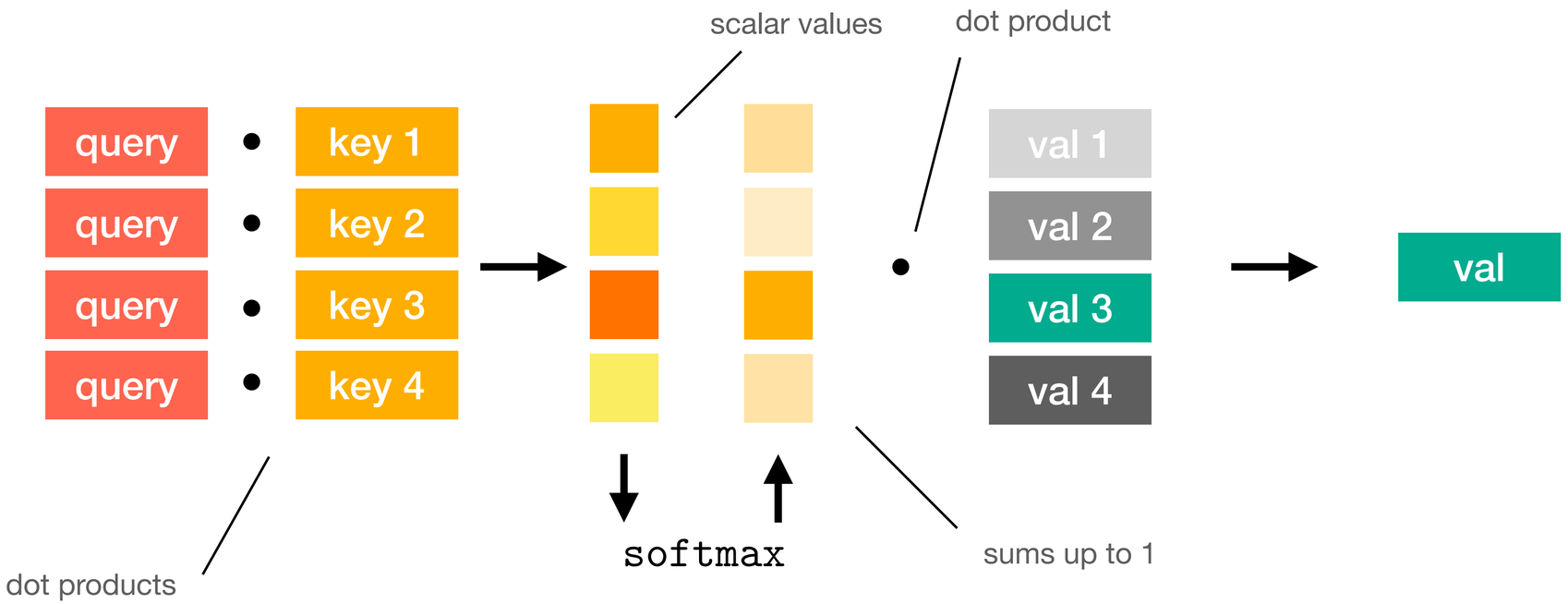
Self-attention: A hash table analogy

We can use standard operators in our DL arsenal: dot product and softmax



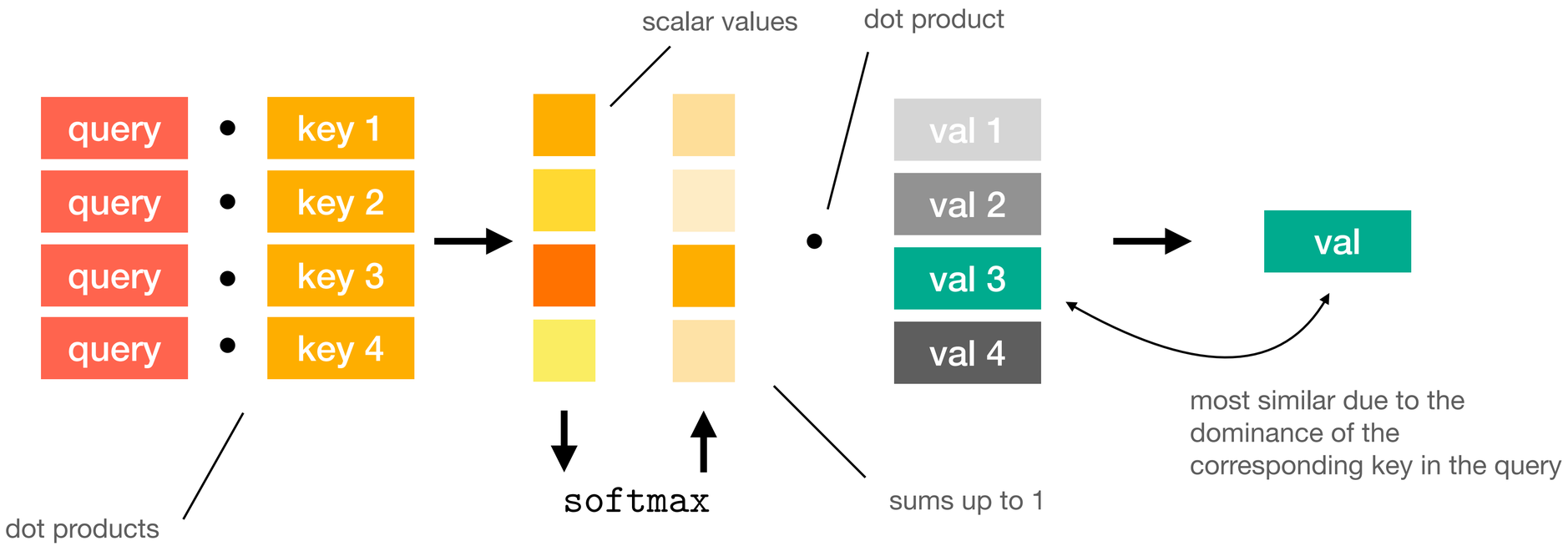
Self-attention: A hash table analogy

We can use standard operators in our DL arsenal: dot product and softmax



Self-attention: A hash table analogy

We can use standard operators in our DL arsenal: dot product and softmax



Self-attention: Summary

Putting it all together. Define:

- a query vector $Q \in \mathbb{R}^{S \times n}$ We have S queries
- a key matrix $K \in \mathbb{R}^{T \times n}$ Our hash table has T (key, value) pairs
- a value matrix $V \in \mathbb{R}^{T \times m}$

Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

Self-attention: Scaling factor

Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

scaling factor

$Q \in \mathbb{R}^{S \times n}$

$K \in \mathbb{R}^{T \times n}$

$V \in \mathbb{R}^{T \times m}$

- factor $\sqrt{n}$ eases optimisation especially for large n (QUIZ: Ideas why?)
  - Large n increases the magnitude of the sum (we have n terms);
  - Values with large magnitude inside softmax lead to vanishing gradient.

Self-attention: Linear projection

Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

scalar factors

$Q \in \mathbb{R}^{S \times n}$

$K \in \mathbb{R}^{T \times n}$

$V \in \mathbb{R}^{T \times m}$

- What is the output dimension? (QUIZ)

Self-attention: Linear projection

Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

scalar factors

$Q \in \mathbb{R}^{S \times n}$

$K \in \mathbb{R}^{T \times n}$

$V \in \mathbb{R}^{T \times m}$

- What is the output dimension? (QUIZ)

$\mathbb{R}^{S \times m}$

“for every query we fetch the corresponding value”

- We can make the input and the output representation homogeneous.
  - This is important if we want to stack this operation and build deep networks.

Self-attention: Linear projection

Input: $X \in \mathbb{R}^{T \times d}$ "T **tokens** of size d"

Define 4 linear mappings: $W^Q, W^K \in \mathbb{R}^{d \times n}$ $W^V \in \mathbb{R}^{m \times d}$ $W^O \in \mathbb{R}^{m \times d}$

These are model parameters

Compute Q, K, V:

$$K := XW^K \quad Q := XW^Q \quad V := XW^V$$

[T x n] [T x n] [T x m]

Output: $Y := \text{Attention}(Q, K, V)W^O$

[T x d]

We started with [T x d] and produced [T x d] output

Self-attention: Complexity

Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

scalar factors

$Q \in \mathbb{R}^{S \times n}$

$K \in \mathbb{R}^{T \times n}$

$V \in \mathbb{R}^{T \times m}$

- Memory complexity ($T \gg n, m$): $O(T^2)$

$$\text{softmax}(QK^T/\sqrt{n})V$$

[T x T]

- Runtime complexity: $O(T^2 n)$

Multi-head attention

Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

scalar factors

$Q \in \mathbb{R}^{S \times n}$

$K \in \mathbb{R}^{T \times n}$

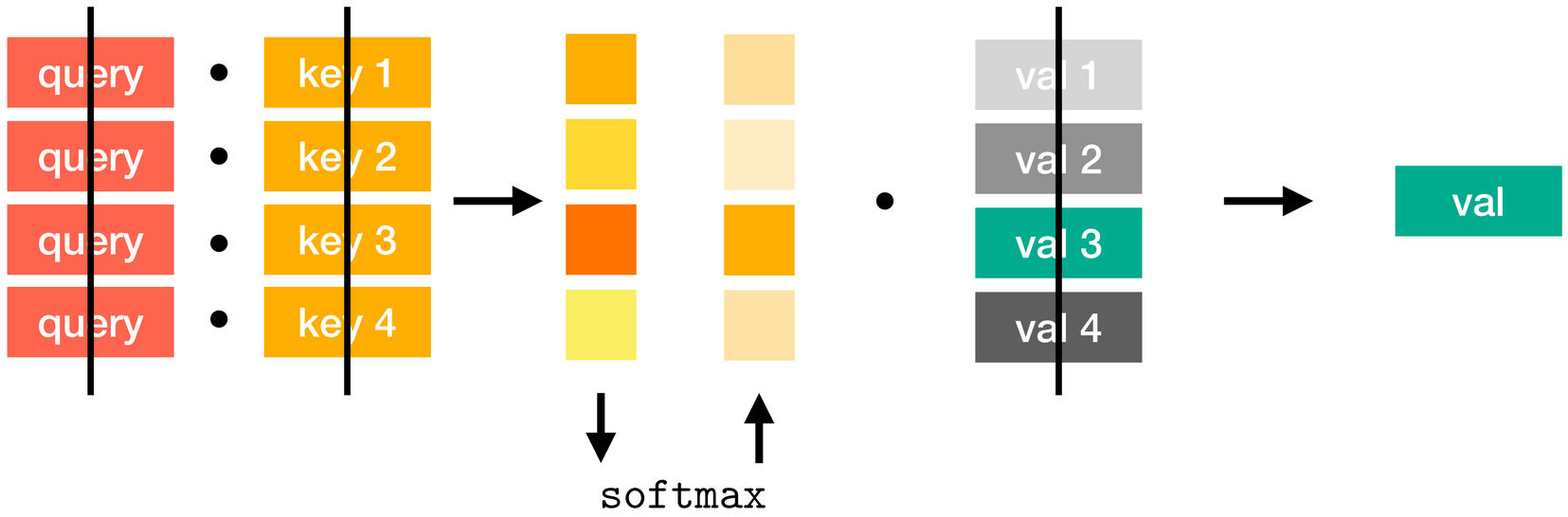
$V \in \mathbb{R}^{T \times m}$

- One softmax – one look-up operation;
- We can extend the same operation to multiple look-ups without increasing the complexity;
- This is called **multi-head** attention.

Multi-head attention

Self-attention: $\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(QK^T/\sqrt{n})V$

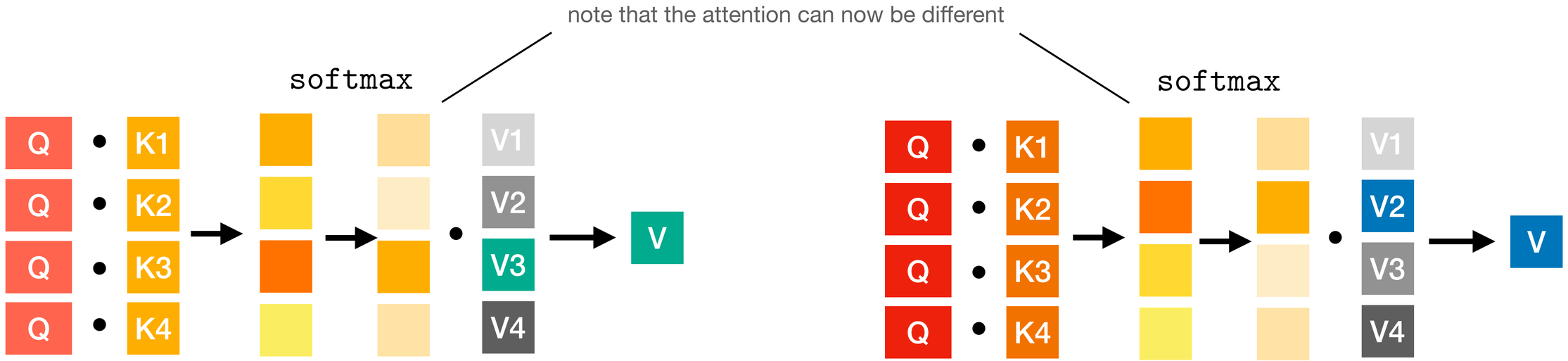
Main idea: chunk Q,K,V vectors



For example, split it in two (e.g. split 128d vectors into 2 64d vectors)

Multi-head attention

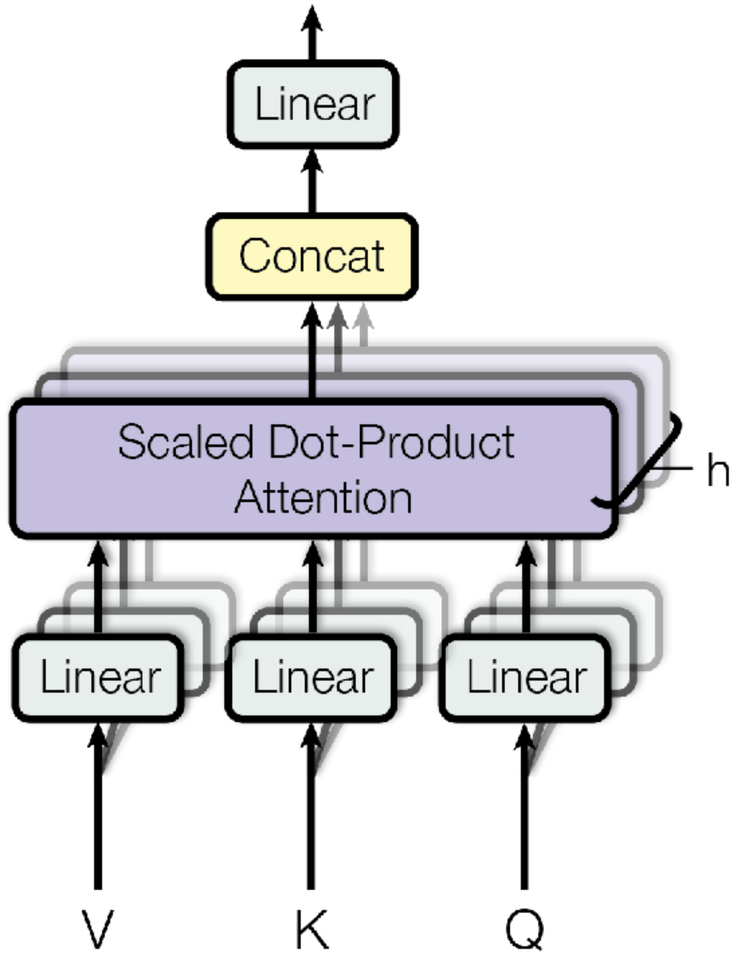
- Now, we have two (Q,K,V) triples, where feature dimension is reduced by a factor of 2.
- Repeat the same process:



- Concatenate V V to obtain the original resolution.

Multi-head attention

Intuition: Given a single query, we can fetch multiple values corresponding to different keys.



[Vaswani et al., 2017]

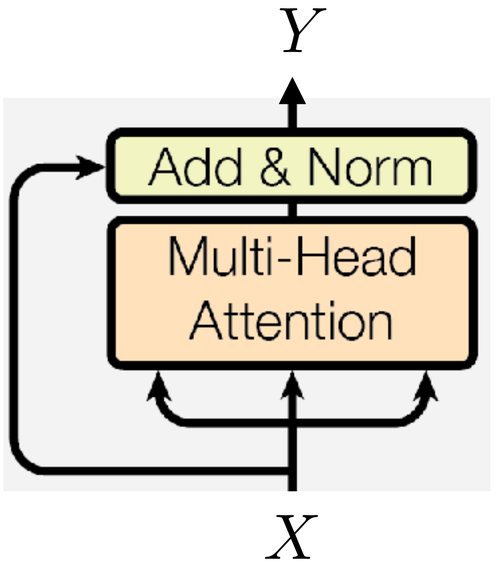
Multi-head attention

- We can design self-attention with arbitrary number of heads
  - condition: feature dimensionality of Q,K and V should be divisible by it.
- With Z heads, a single query can fetch up to Z different values
  - we can model more complex interactions between the tokens
- No increase in runtime complexity (some increase in wall-time).

Self-attention: Normalisation

Further improvements:

- Layer normalisation (Ba et al., 2016)
  - normalise each token feature w.r.t. its own mean and standard deviation.
- Residual connection:
  - add the original token feature before the normalisation.



$$Y := \mathrm{LayerNorm}(X + \mathrm{MHA}(X))$$

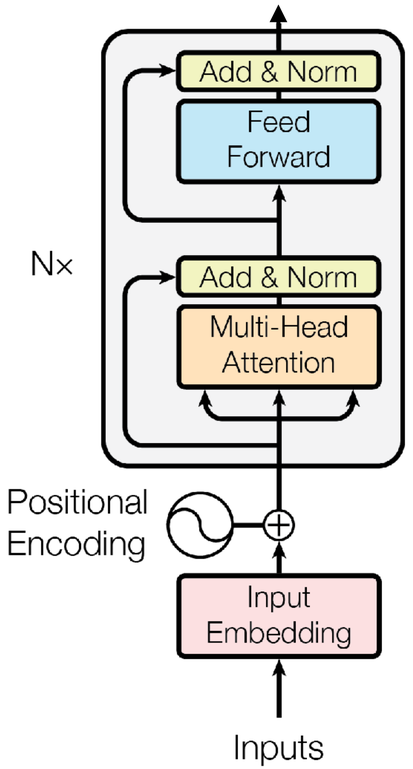
[Vaswani et al., 2017]

Self-attention: Recap

- Self-attention is versatile:
  - arbitrary number of tokens;
  - design choices: the number of heads, feature dimensionality.
- Self-attention computation scales quadratically w.r.t. tokens,
  - since we need to compute the pairwise similarity matrix.
- Self-attention is permutation-invariant (w.r.t. the tokens):
  - it does matter if query A is at index i in Q; we will always fetch the same value.
  - Is it actually always useful?

Transformers

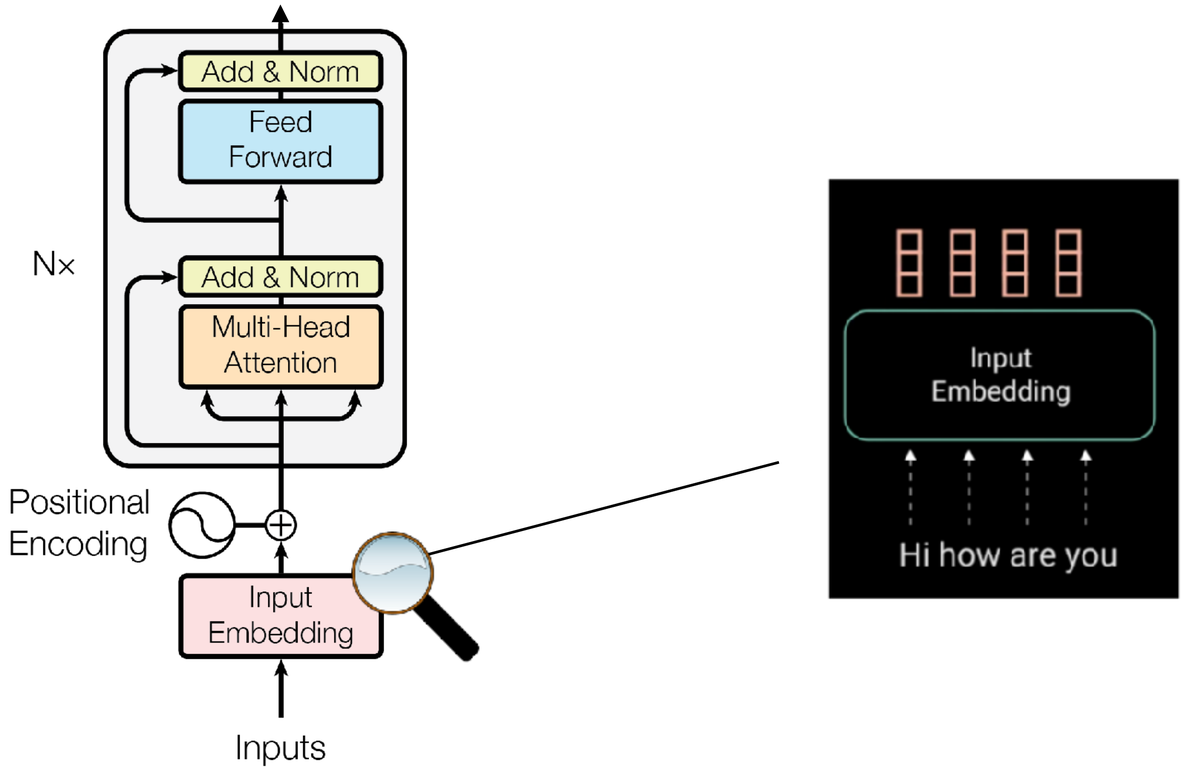
- Transformer-based encoder:



[Vaswani et al., 2017]

Transformers

- Transformer-based encoder:

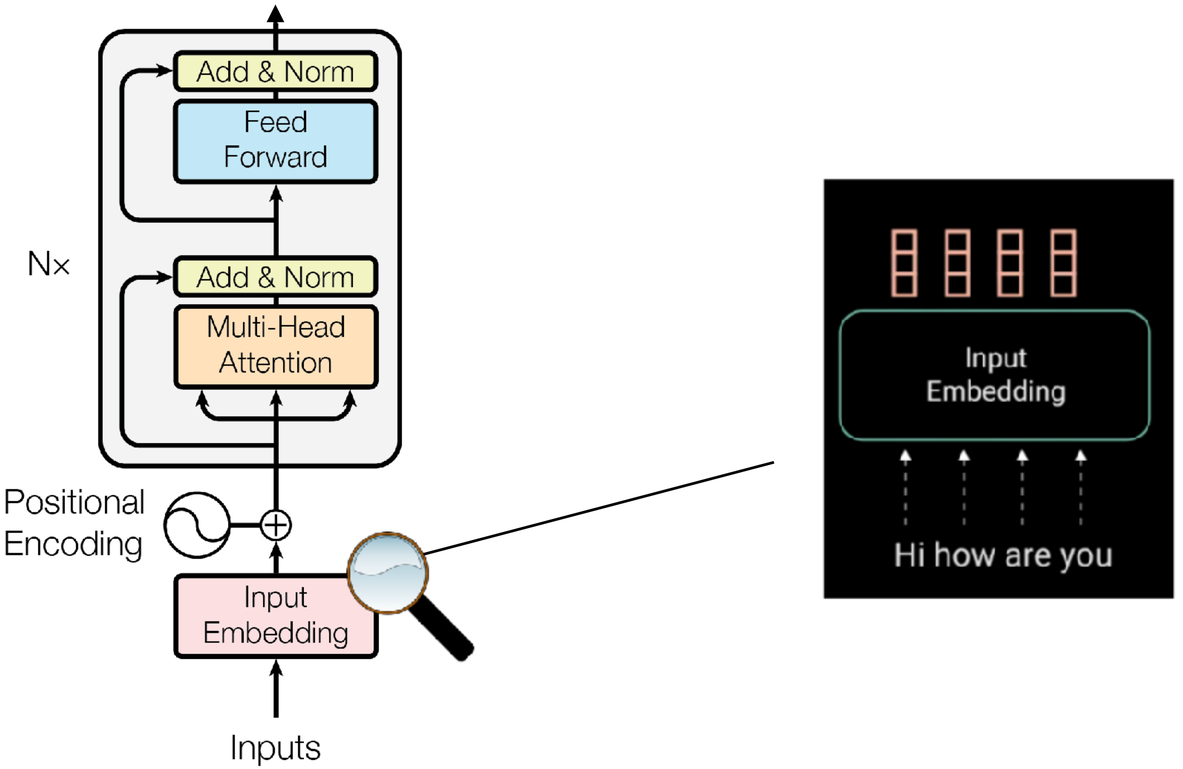


[Vaswani et al., 2017]

Transformers

- Transformer-based encoder:

Breaks permutation invariance.

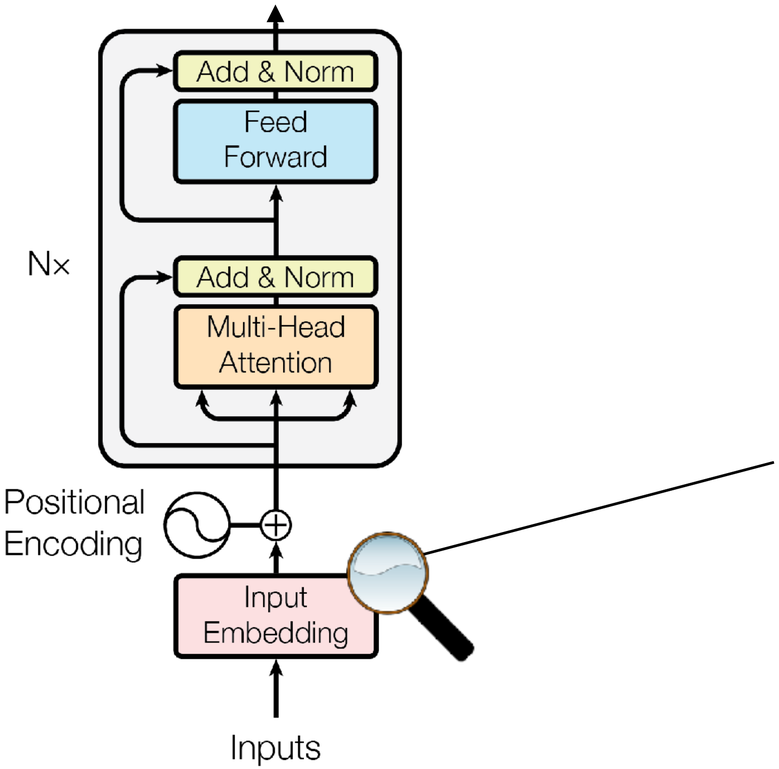


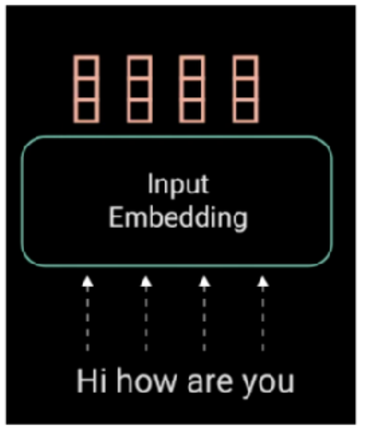
[Vaswani et al., 2017]

Transformers

- Transformer-based encoder:

Breaks permutation invariance.





- Positional encoding is a matrix of size distinct values corresponding to each row

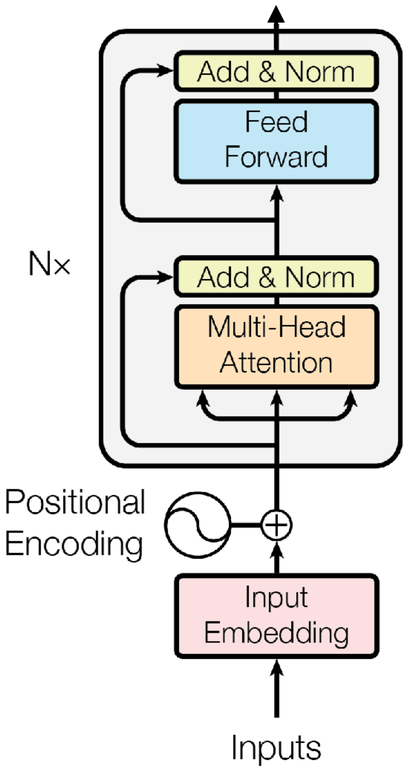
[Vaswani et al., 2017]

Positional encoding

- Positional encoding stores a unique value (vector) corresponding to a token index.
  - It can be a learnable model parameter or fixed, for example:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

  - It introduces the notion of spatial affinity (e.g. distance between two words in sentence; two patches in an image, etc.)
  - It breaks permutation invariance.



[Vaswani et al., 2017]

Self-attention: Summary

- Self-attention is versatile:
  - arbitrary number of tokens;
  - design choices: the number of heads, feature dimensionality.
- Self-attention computation scales quadratically w.r.t. tokens,
  - since we need to compute the pairwise similarity matrix.
- Self-attention is permutation-invariant (w.r.t. the tokens):
  - it does matter if query A is at index i in Q; we will always fetch the same value.
  - **unless we add positional encoding.**

Transformers in computer vision

Vision Transformer (ViT)

- An all-round transformer architecture
  - Competitive with CNNs (classification accuracy)*
  - Main idea: Train on a sequences of image patches; only minor changes to the original (NLP) Transformer model.
  - Can be complemented with a CNN model.

(Dosovitskiy et al., 2020)

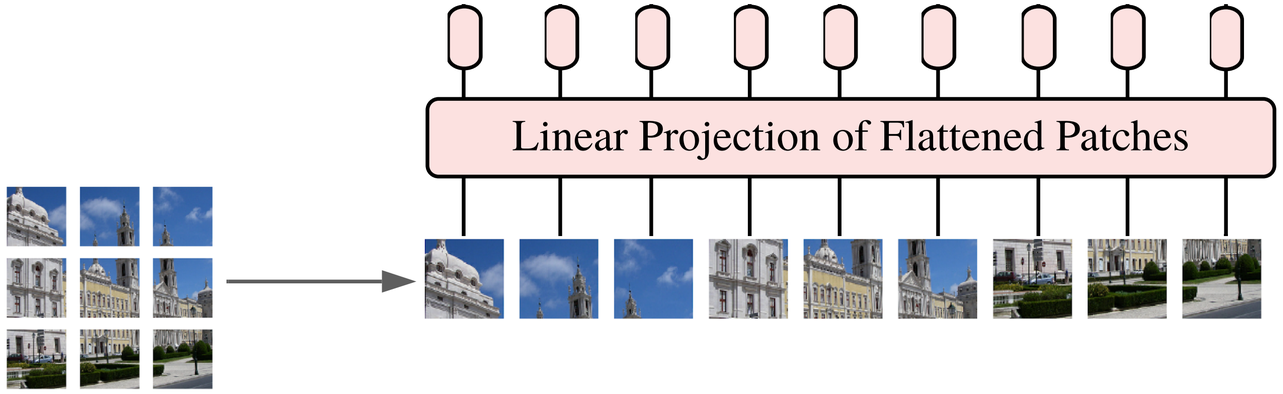
Vision Transformer

1. Split an image into fixed-sized patches.

(Dosovitskiy et al., 2020)

Vision Transformer

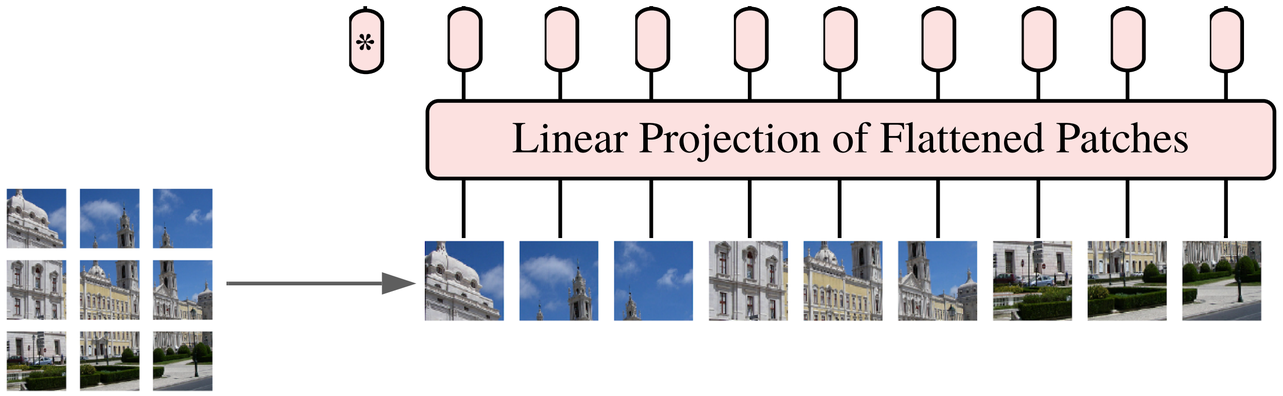
1. Split an image into fixed-sized patches.
2. Linearly embed each patch (a fully connected layer).



(Dosovitskiy et al., 2020)

Vision Transformer

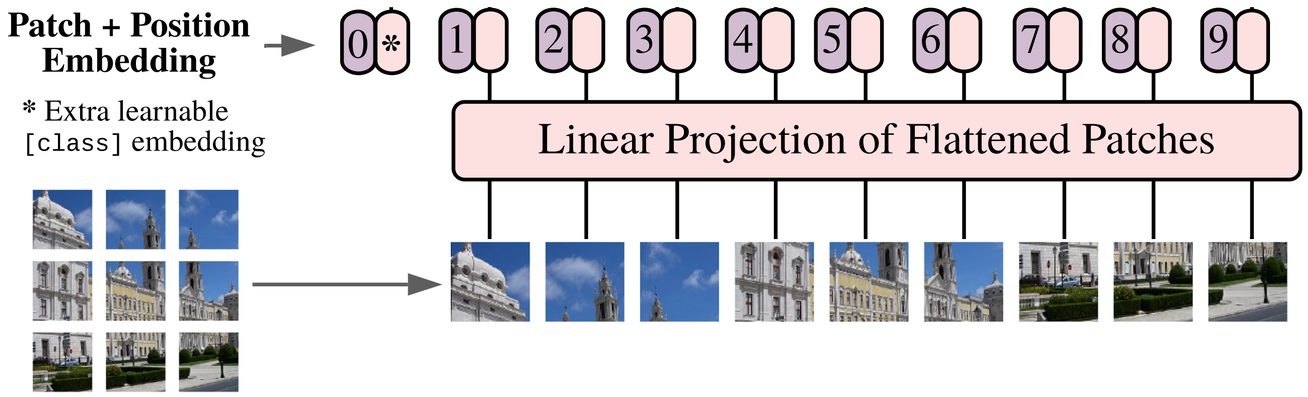
1. Split an image into fixed-sized patches.
2. Linearly embed each patch (a fully connected layer).
3. Attach an extra “[class]” embedding (learnable).



(Dosovitskiy et al., 2020)

Vision Transformer

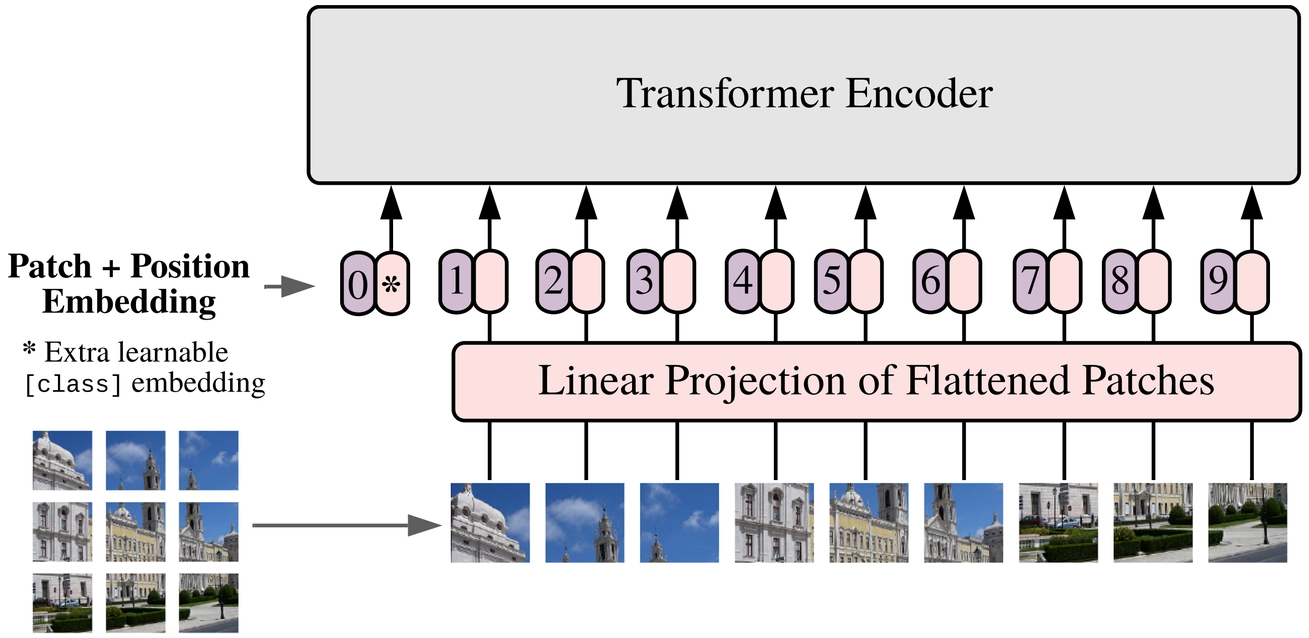
1. Split an image into fixed-sized patches.
2. Linearly embed each patch (a fully connected layer).
3. Attach an extra “[class]” embedding (learnable).
4. Add positional embeddings.



(Dosovitskiy et al., 2020)

Vision Transformer

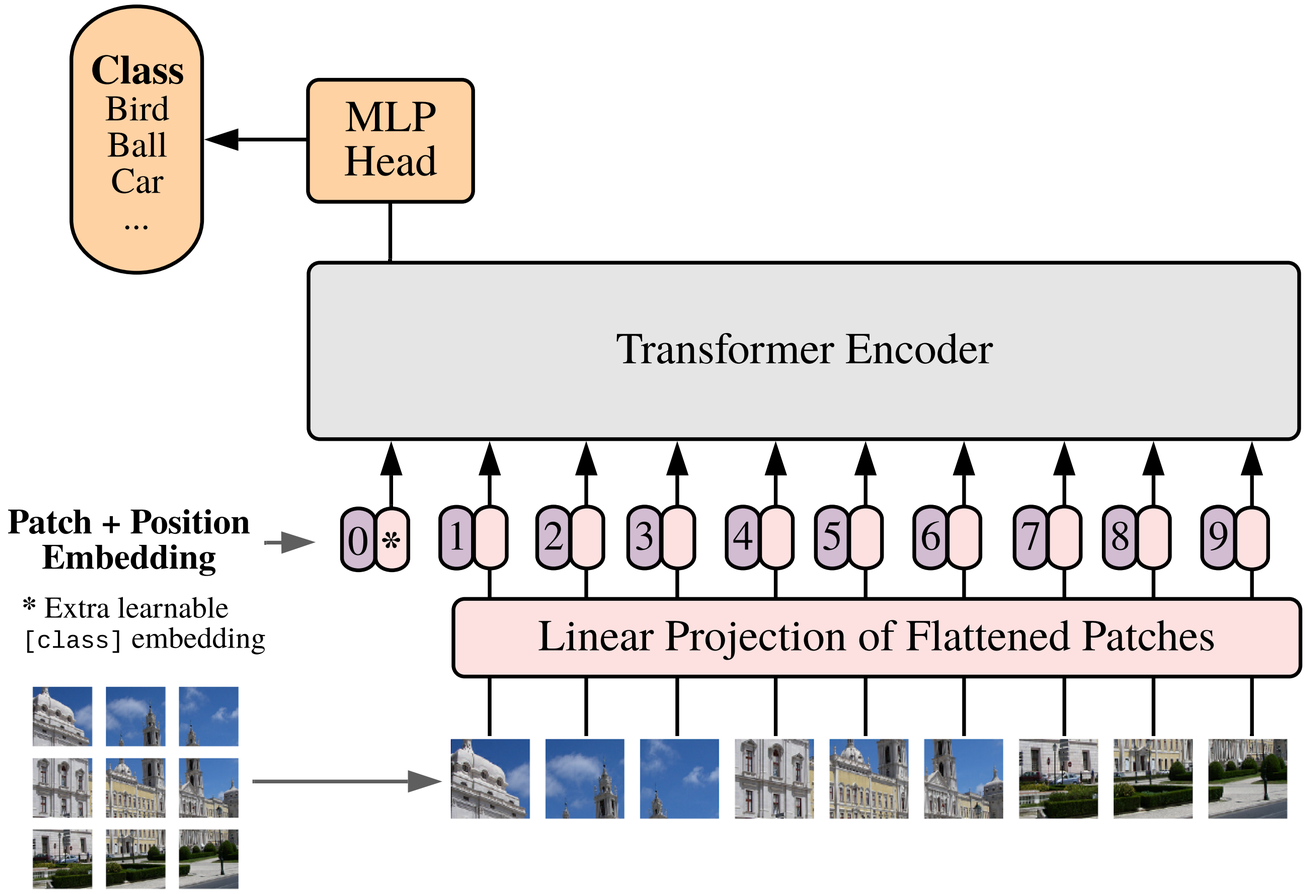
1. Split an image into fixed-sized patches.
2. Linearly embed each patch (a fully connected layer).
3. Attach an extra “[class]” embedding (learnable).
4. Add positional embeddings.
5. Feed the sequence to the standard Transformer encoder.



(Dosovitskiy et al., 2020)

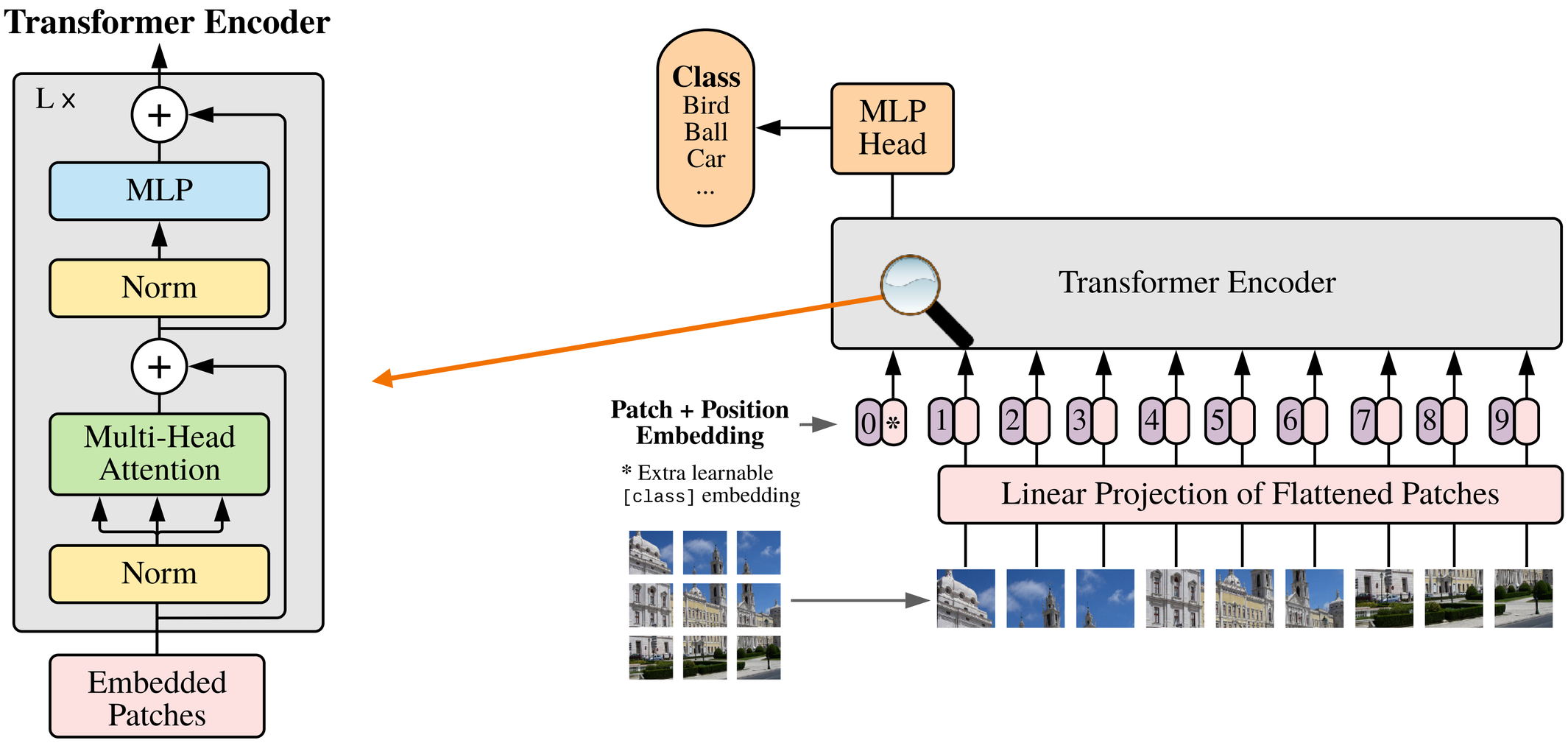
Vision Transformer

1. Split an image into fixed-sized patches.
2. Linearly embed each patch (a fully connected layer).
3. Attach an extra “[class]” embedding (learnable).
4. Add positional embeddings.
5. Feed the sequence to the standard Transformer encoder.
6. Predict the class with an MLP using the output [class] token.



(Dosovitskiy et al., 2020)

ViT Encoder



(Dosovitskiy et al., 2020)

Experiments with ViT

- ViT performs well only when pre-trained on large JFT dataset (300M images). (QUIZ: Why?)

Experiments with ViT

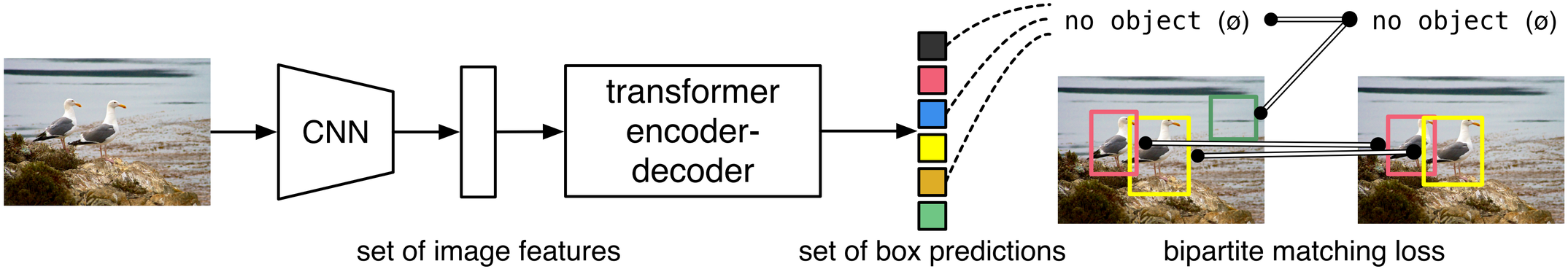
- ViT performs well only when pre-trained on large JFT dataset (300M images). (QUIZ: Why?)
- ViT does not have the inductive biases of CNNs:
  - locality (self-attention is global);
  - 2D neighbourhood structure (positional embedding is inferred from data);
  - translation equivariance.

Experiments with ViT

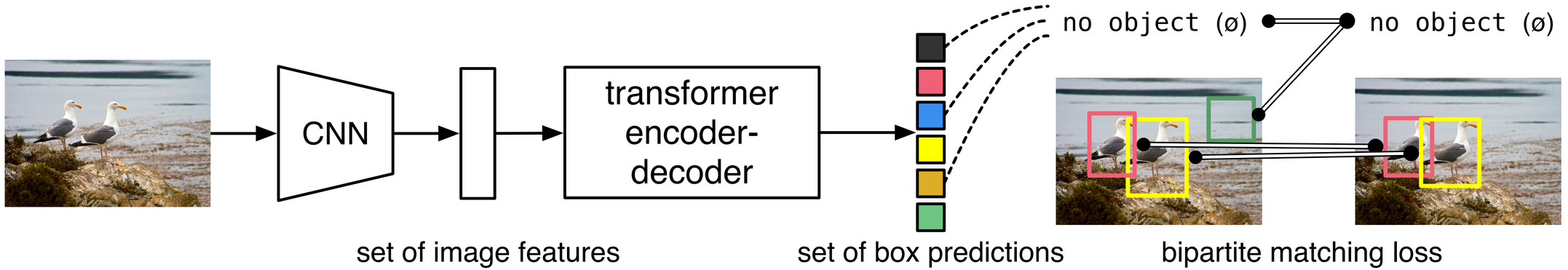
- ViT performs well only when pre-trained on large JFT dataset (300M images). (QUIZ: Why?)
- ViT does not have the inductive biases of CNNs:
  - locality (self-attention is global);
  - 2D neighbourhood structure (positional embedding is inferred from data);
  - translation equivariance.
- But still: now we use the same computational framework for two distinct modalities: language and vision!

Transformer for detection?

- Would it make sense to adapt the Transformer to object detection?
- Recall that object detection is a set prediction problem
  - i.e. we do not care about the ordering of the bounding boxes.
- Transformers are well-suited for processing sets.
- Directly formulate the task as set prediction!

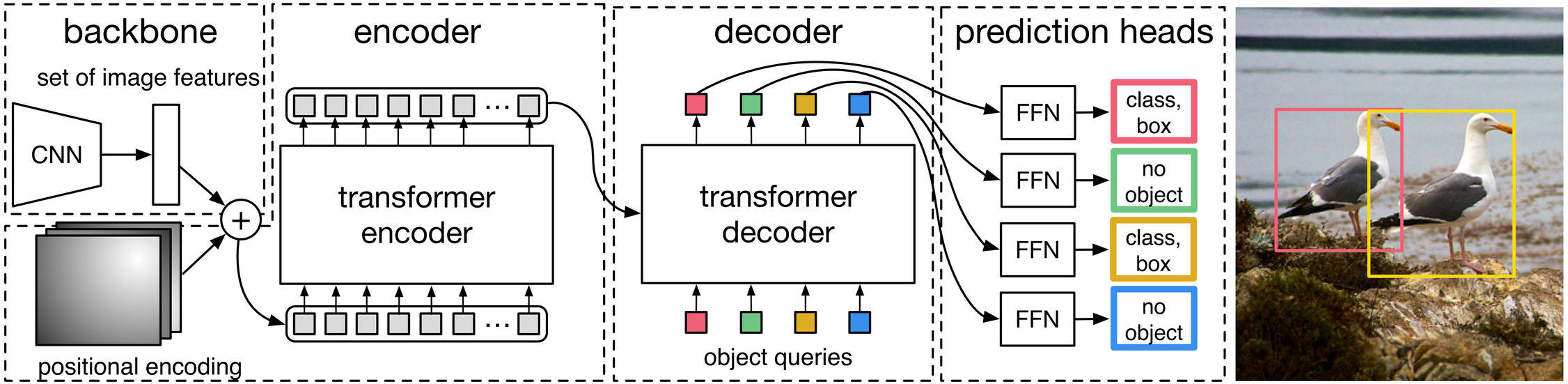


Detection: DETR

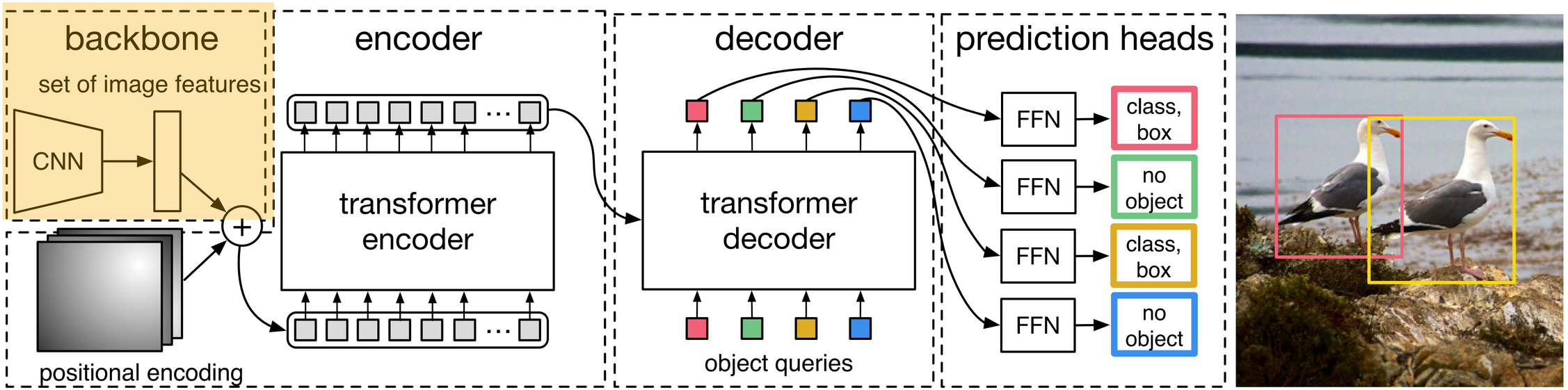


- The CNN predicts local feature embeddings.
- The Transformer predicts the bounding boxes in parallel.
- During training, we uniquely assigns predictions to ground truth boxes with Hungarian matching.
- No need for non-maximum supression.

DETR: A closer look

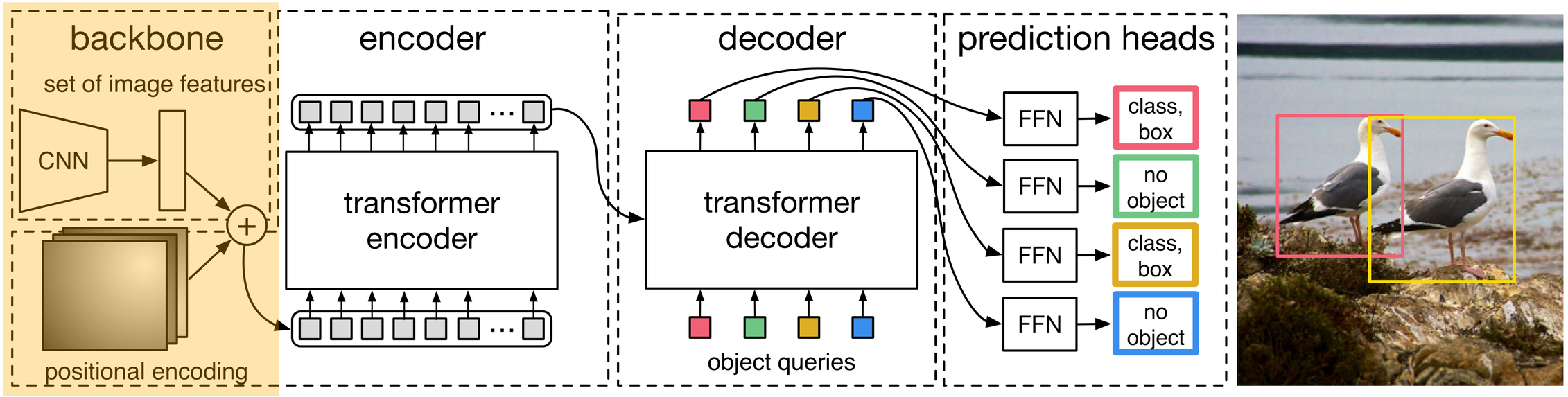


DETR: A closer look



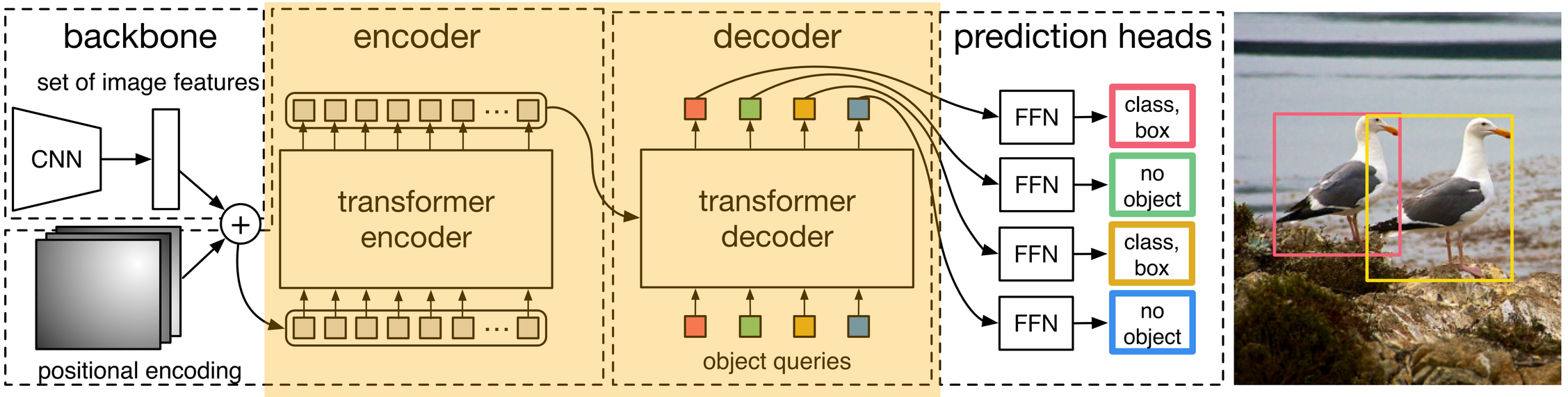
- DETR uses a conventional CNN backbone to learn a 2D representation of an input image.

DETR: A closer look



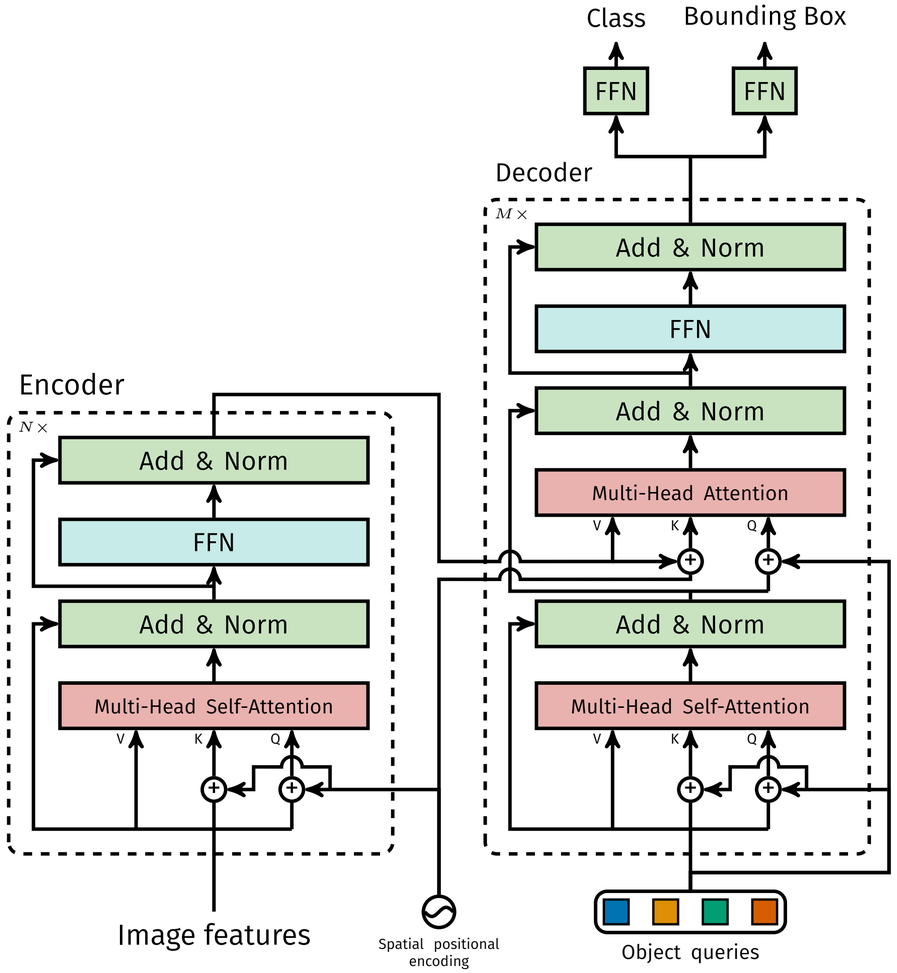
- The model flattens it and supplements it with a positional encoding before passing it to the Transformer.

DETR: A closer look



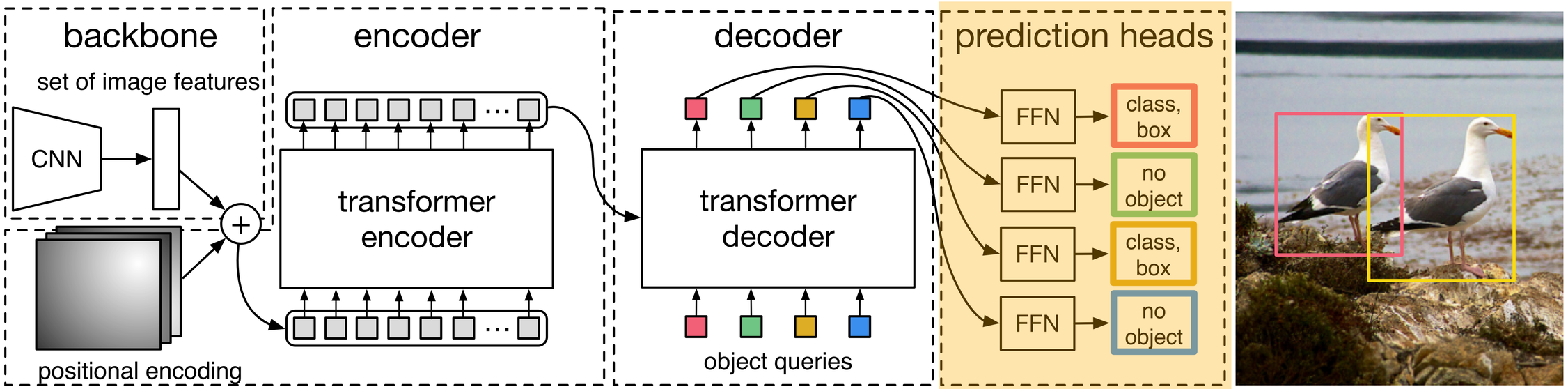
- The Transformer encodes the input sequence.
- We supply **object queries** (learnable positional encoding) to the decoder, which jointly attends to the encoder output.

DETR: Transformer architecture



- The Transfomer is very similar to [Vaswani et al., 2017].
- Output (for each object query):
  - the class logits;
  - the bounding box (normalised coordinates).

DETR: A closer look



- Each output embedding of the decoder is passed to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.

DETR: The loss function

- Hungarian matching between the predictions (queries) and the ground truth.

DETR: The loss function

- Hungarian matching between the predictions (queries) and the ground truth.

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

Optimal assignment computed in the first step

DETR: The loss function

- Hungarian matching between the predictions (queries) and the ground truth.

It also contain empty ground-truth

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

Loss for the class

DETR: The loss function

- Hungarian matching between the predictions (queries) and the ground truth.

Loss for the bounding-box

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

DETR: Bounding Box loss

- The bounding box loss $\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i))$:

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

hyperparameters

DETR: Bounding Box loss

- The bounding box loss $\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i))$:

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

Generalised IoU

DETR: Bounding Box loss

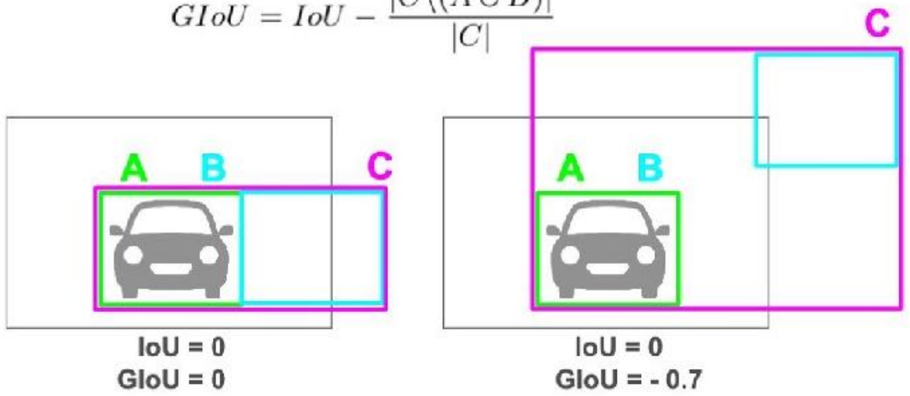
- The bounding box loss $\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i))$:

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

Generalised IoU

[Credit: Helena Gva]

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$



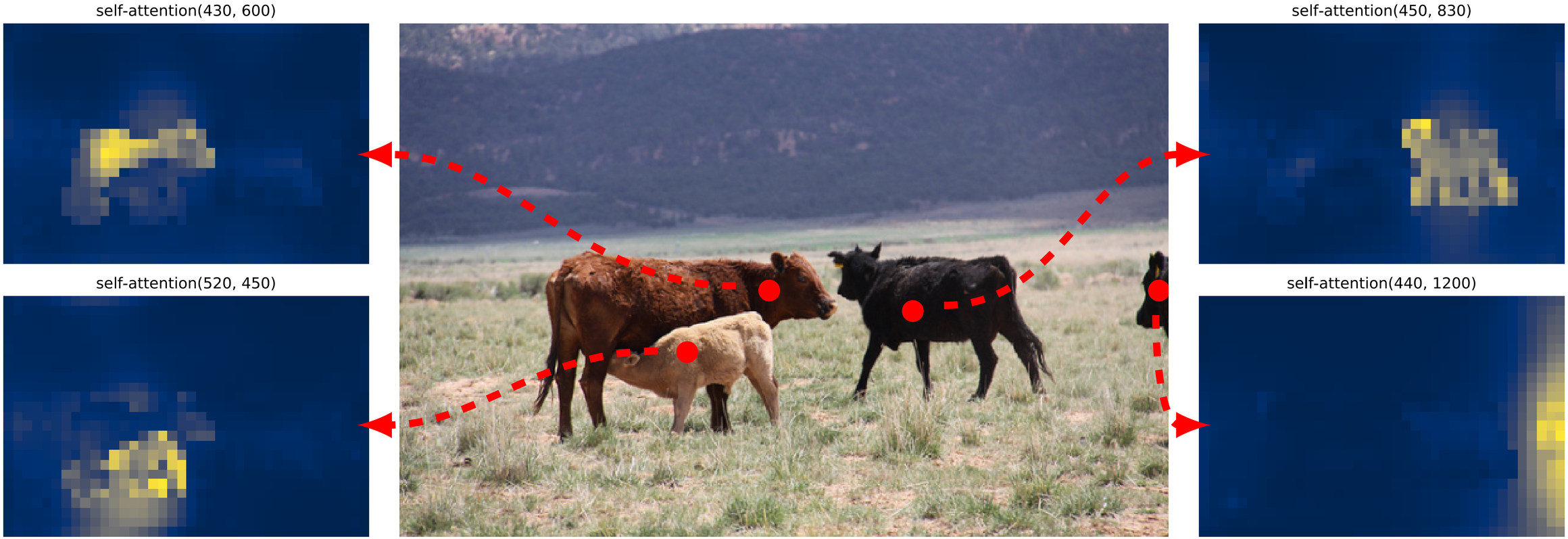
Idea: prevent vanishing gradients when IoU is zero.

DETR: Results

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**

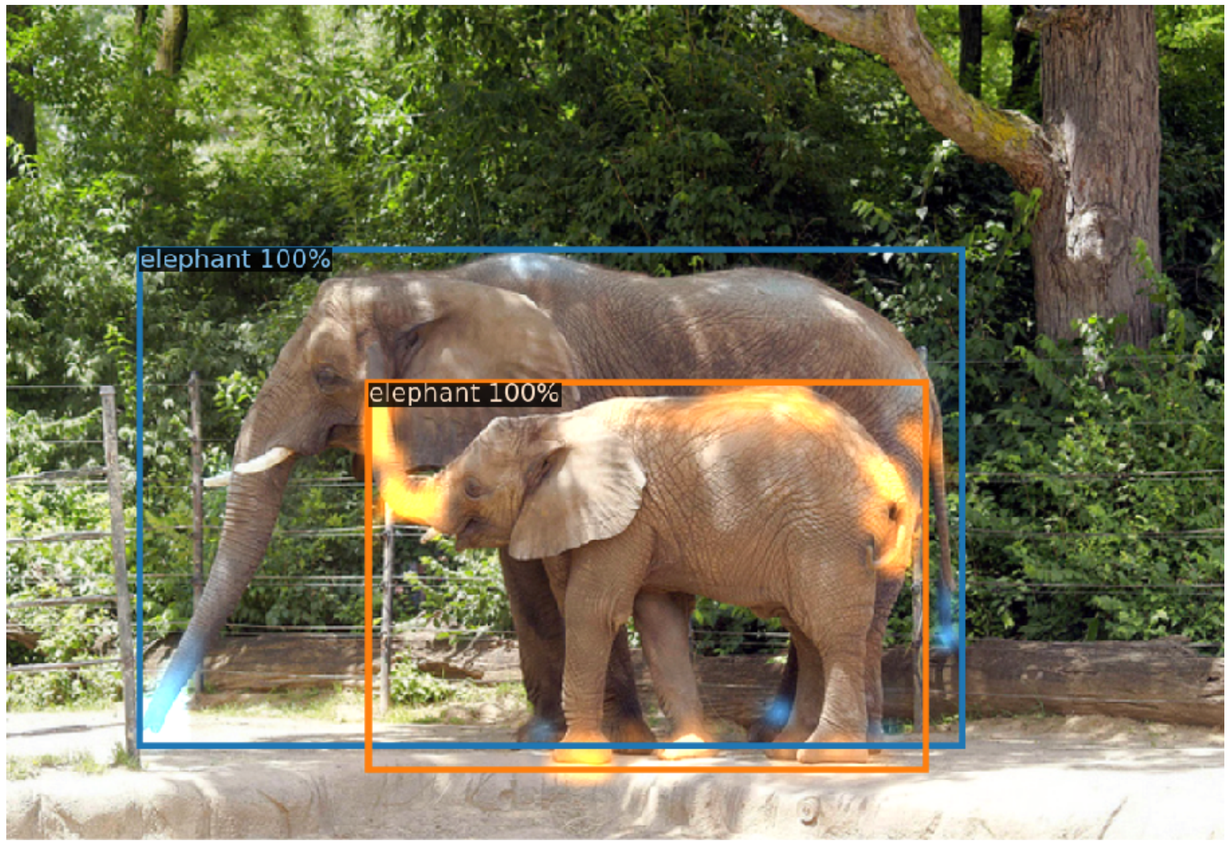
* x2 higher feature resolution (“Dilated C5 stage”)

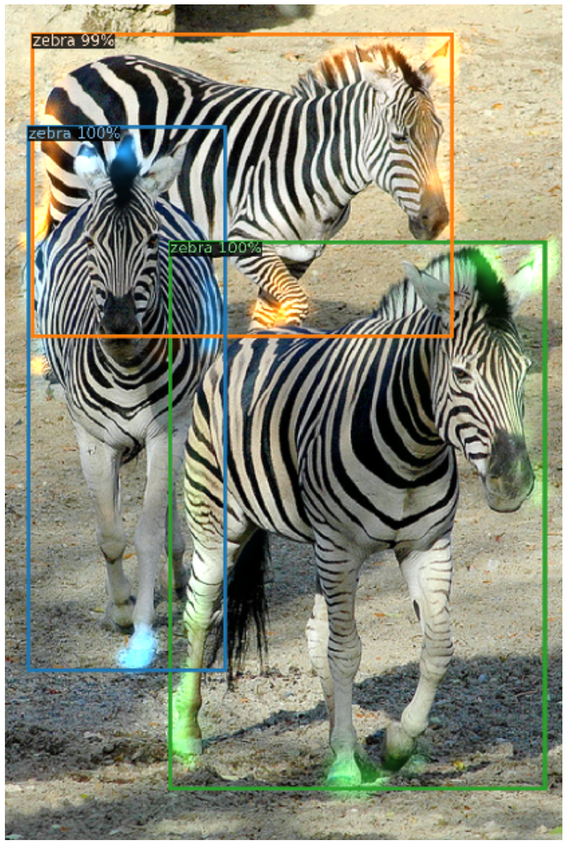
DETR: Qualitative results



The encoder is able to separate individual instances. Predictions are made with baseline DETR model on a validation set image.

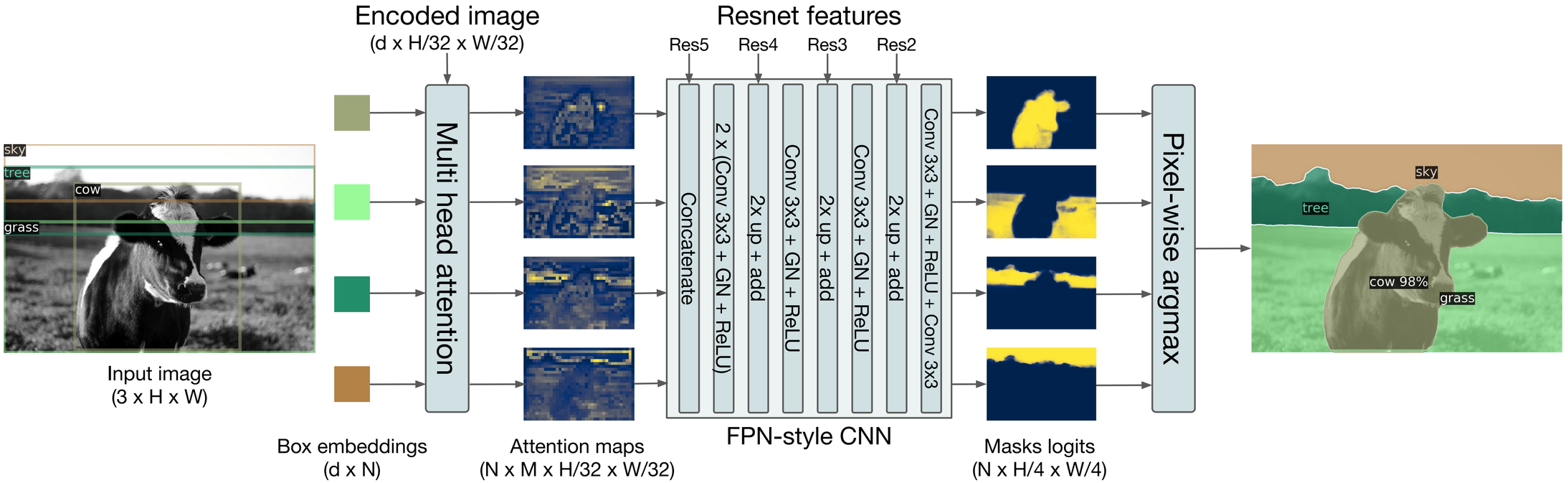
DETR: Qualitative results





Decoder attention for every predicted object. Attention scores are coded with different colours for different objects. Decoder typically attends to object extremities, such as legs and heads.

DETR for panoptic segmentation



We can generate a binary mask (both for things and stuff) and then merge the masks using pixel-wise argmax.

DETR: Panoptic Segmentation results

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

DETR: Summary

- Accurate detection with a (relatively) simple architecture.
- No need for non-maximum suppression
  - We can simply disregard bounding boxes with “empty” class, or with low confidence.
- Accurate panoptic segmentation with a minor extension.

Issues:

- High computational and memory requirements (especially the memory).
- Slow convergence / long training.

See: Zhu et al., “Deformable DETR: Deformable Transformers for End-to-End Object Detection” (ICLR 2021).

Computer Vision III:

Recent research: Transformers

Nikita Araslanov
17.01.2023

Adapted from:
Prof. Laura Leal-Taixé
https://dvl.in.tum.de