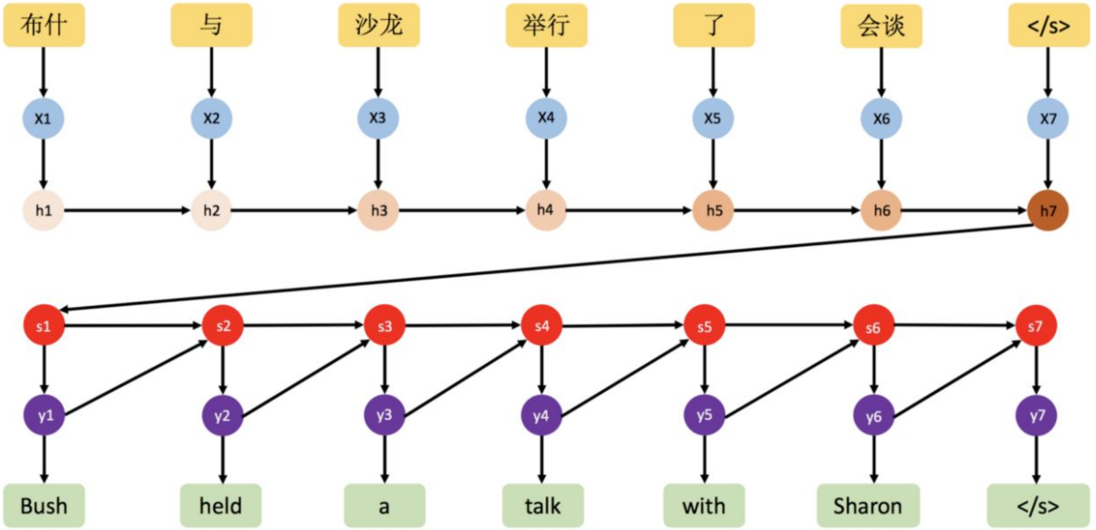
Transformers and their usage in Computer Vision

Deep Learning Revolution

	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~~ (Augmented)	Traditional ML/CV	CNNs, RNNs

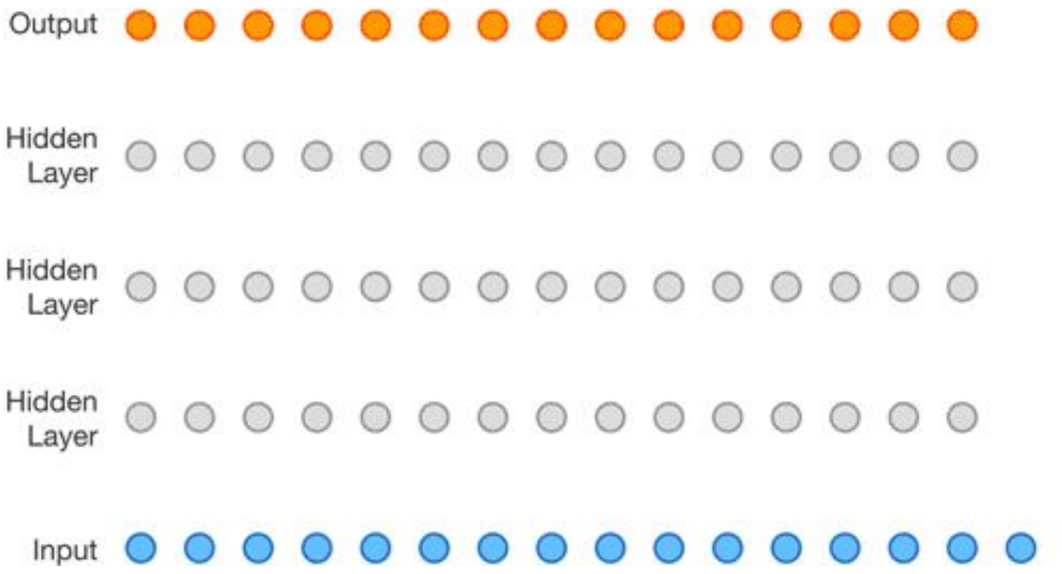
Problems with RNN



(Sutskever et al., 2014)

- Each word is dependent on the words coming before it (parametrized by the hidden states).
- Vanishing gradient problem.
- Long-short term memory dependencies are not that long.

Trying to solve it with convolution



Still the position of the words matters and it is structured. Why does word 5 should be before word 8 in machine translation?

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

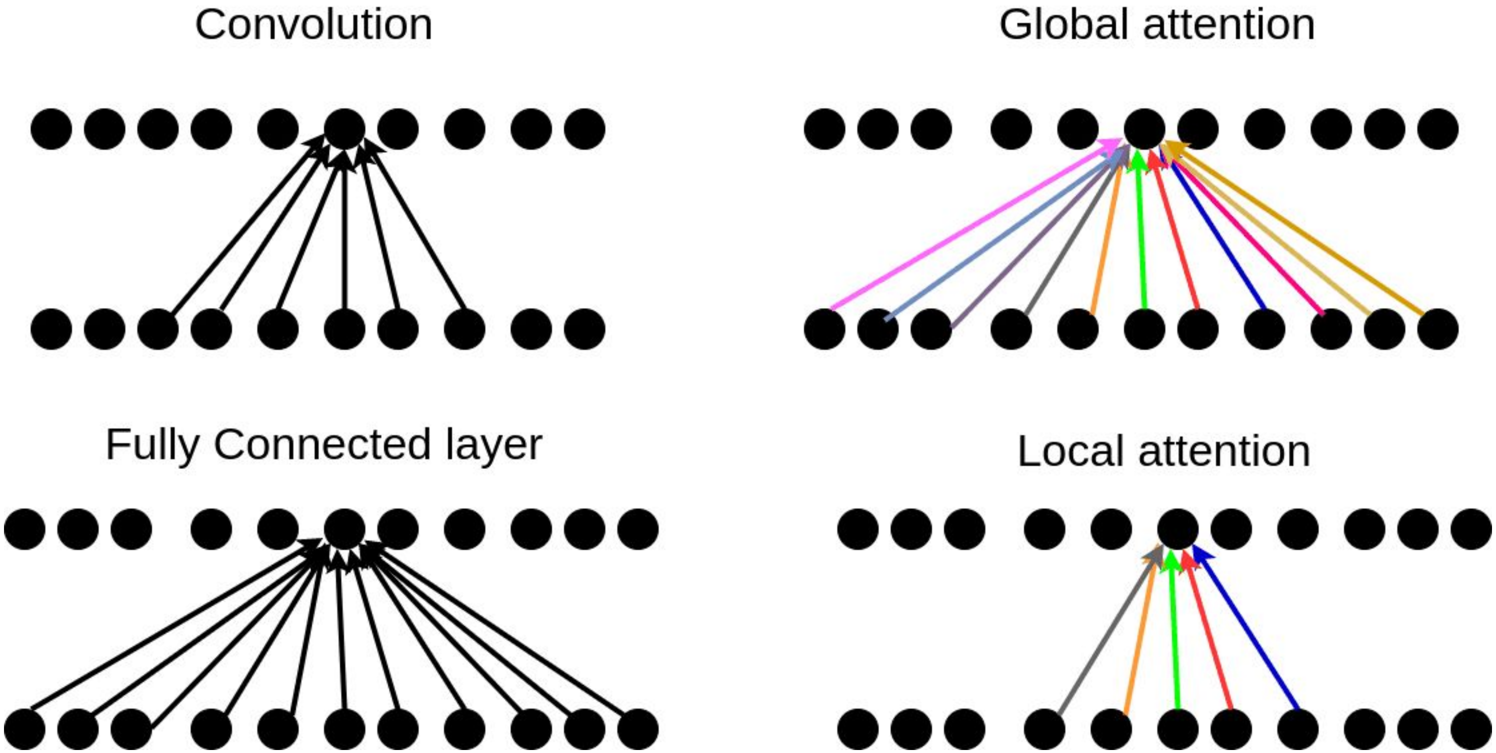
Aidan N. Gomez[*][†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*][‡]
illia.polosukhin@gmail.com

Over 20 thousand citations in less than 3.5 years!

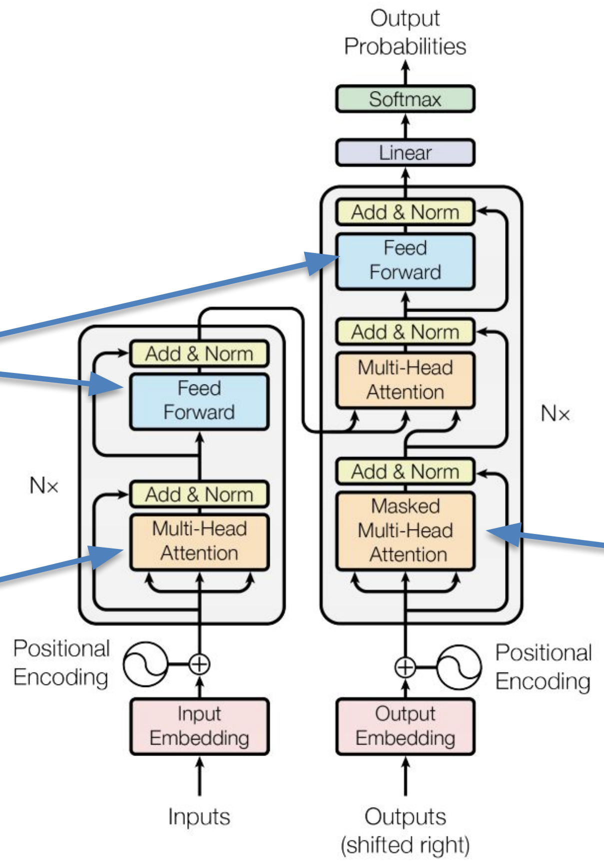
Attention vs convolution



Transformers

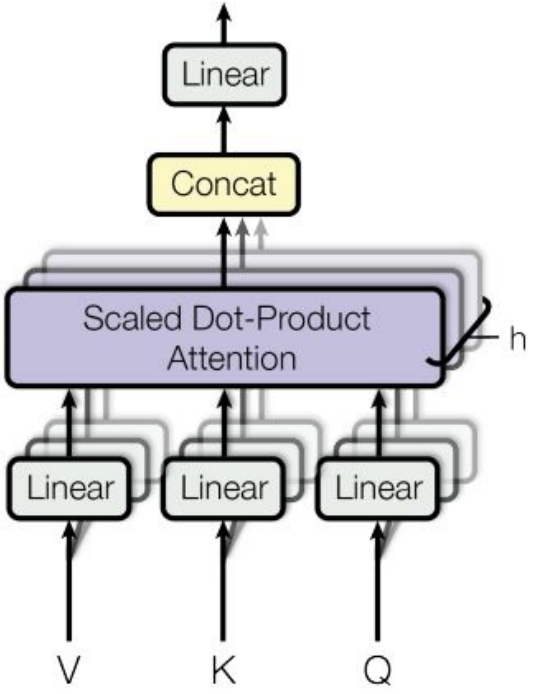
Fully connected layer

Multi-Head Attention on the "encoder"



Masked Multi-Head Attention on the "decoder"

Multi-Head Attention



Intuition: Take the query Q, find the most similar key K, and then find the value V that corresponds to the key.

In other words, learn V, K, Q where:
V – here is a bunch of interesting things.
K – here is how we can index some things.
Q – I would like to know this interesting thing.

Loosely connected to Neural Turing Machines (Graves et al.).

Multi-Head Attention

Index the values via a differentiable operator.

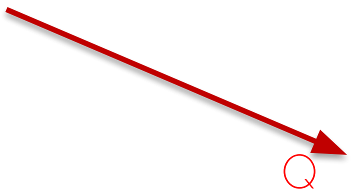
Multiply queries with keys

Get the values

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

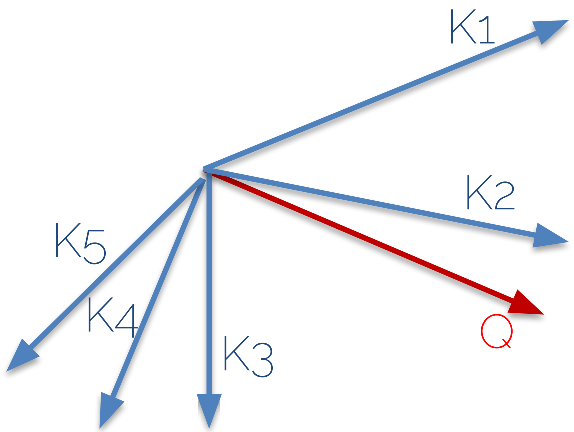
To train them well, divide by. $\sqrt{d_k}$, "probably" because for large values of the key's dimension, the dot product grows large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

Multi-Head Attention

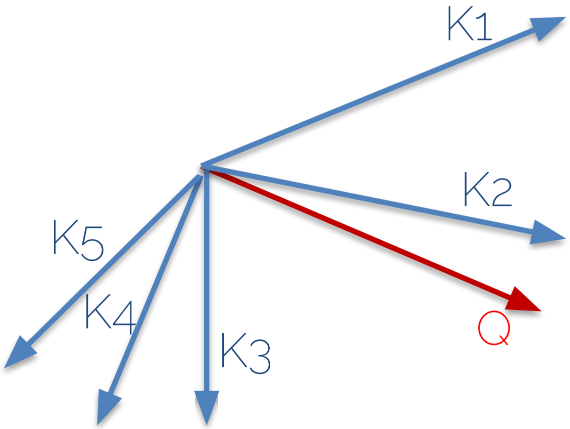


Adapted from Y. Kilcher

Multi-Head Attention

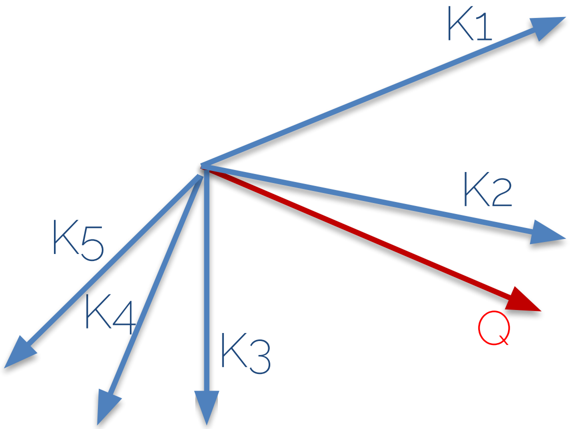


Multi-Head Attention



Values
V1
V2
V3
V4
V5

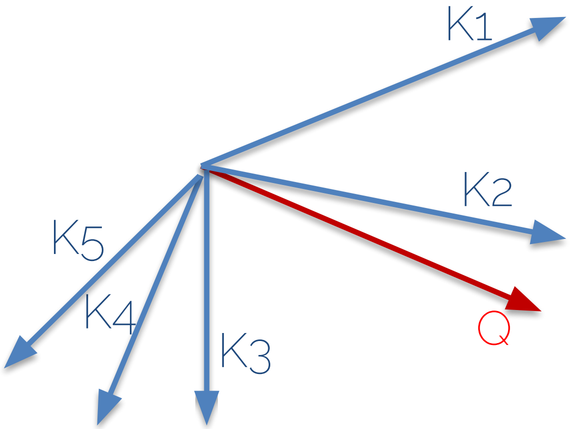
Multi-Head Attention



Values
V1
V2
V3
V4
V5

QK^T Essentially, dot product between sum(<Q,K1>), sum(<Q,K2>), sum(<Q,K3>), sum(<Q,K4>), sum(<Q,K5>).

Multi-Head Attention

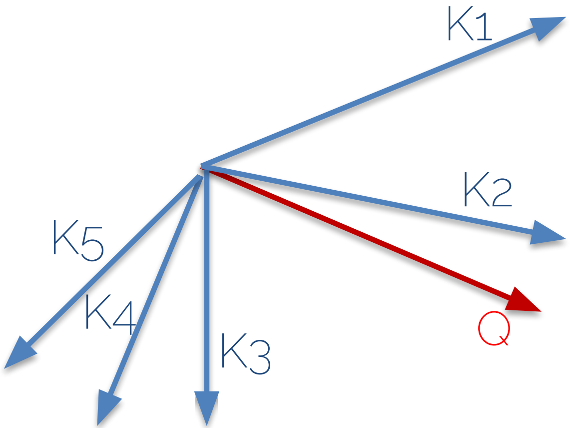


Values
V1
V2
V3
V4
V5

$$\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Is simply inducing a distribution over the values.
The larger a value is, the higher is its softmax value.
Can be interpreted as a differentiable soft indexing.

Multi-Head Attention

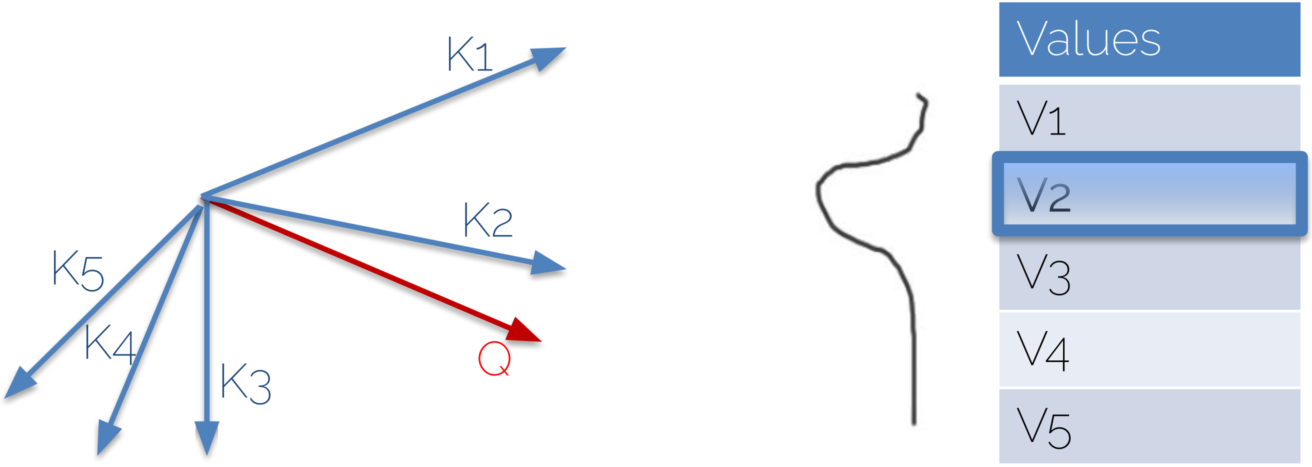


Values
V1
V2
V3
V4
V5

$$\mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})$$

Is simply inducing a distribution over the values.
The larger a value is, the higher is its softmax value.
Can be interpreted as a differentiable soft indexing.

Multi-Head Attention

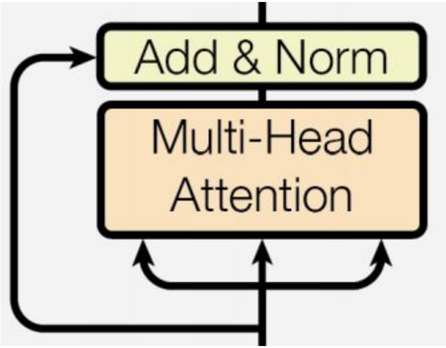


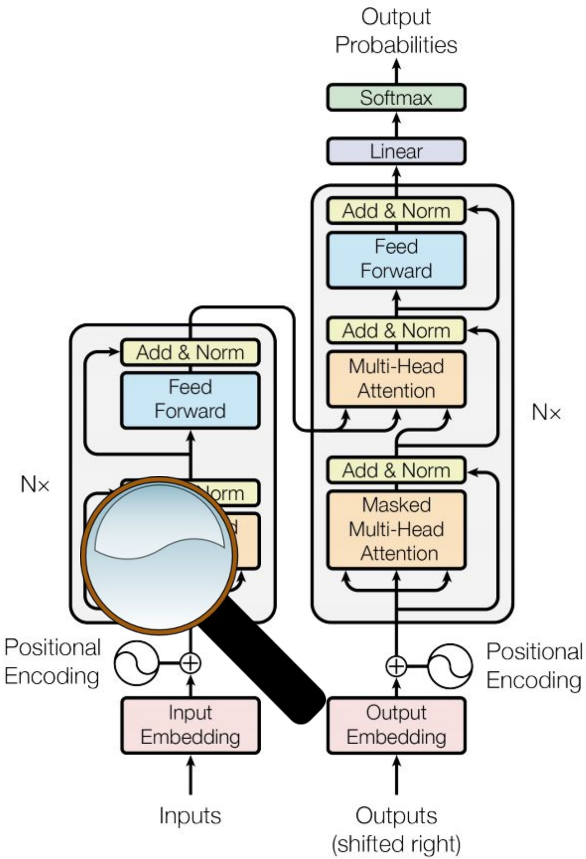
$$\mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Selecting the value V where the network needs to attend..

Transformers – a closer look

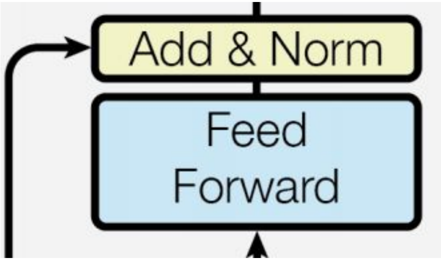
K parallel attention heads.

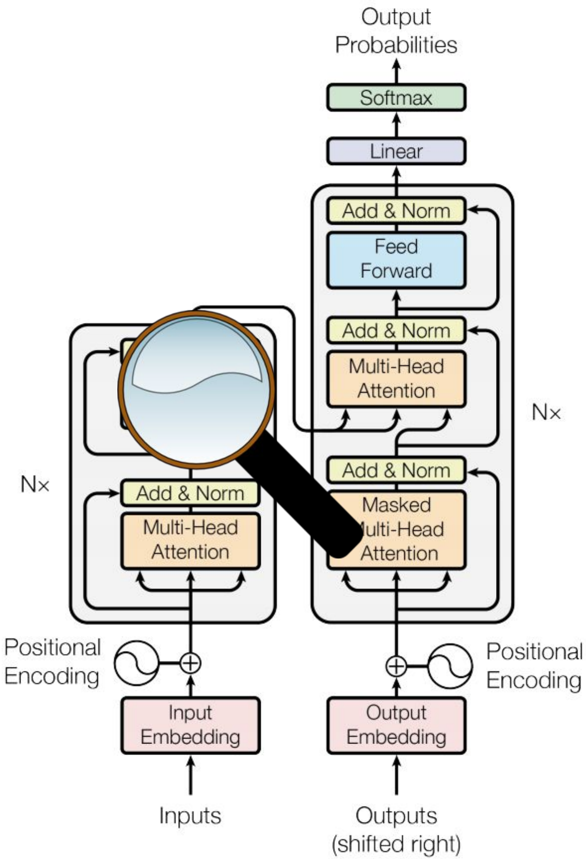




Transformers – a closer look

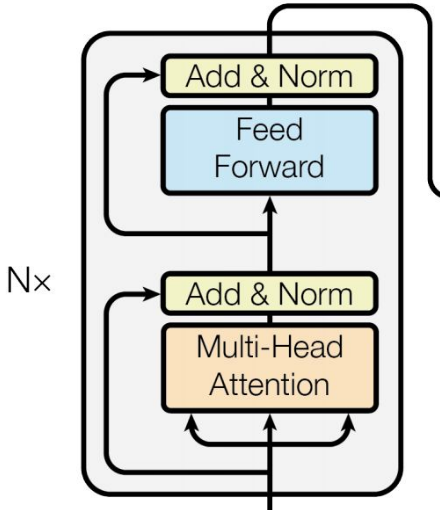
Good old fully-connected layers.

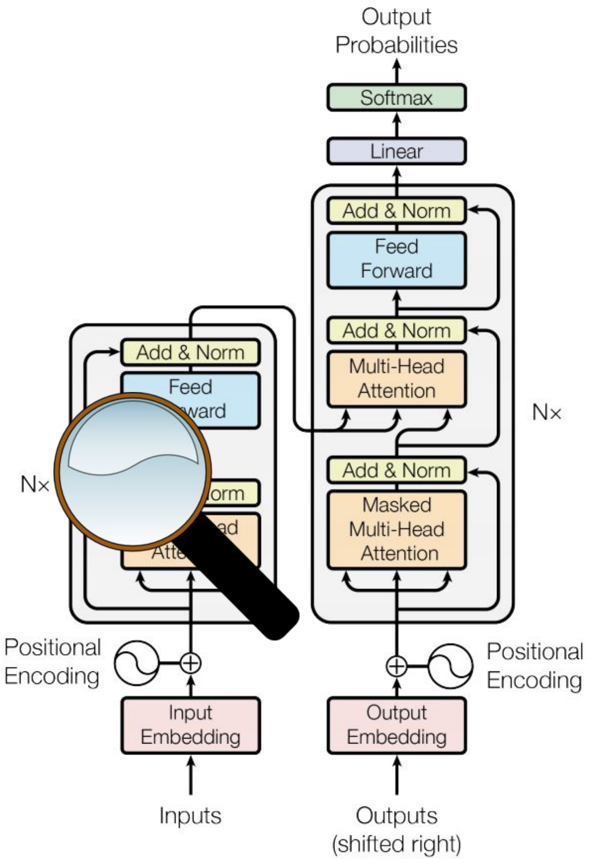




Transformers – a closer look

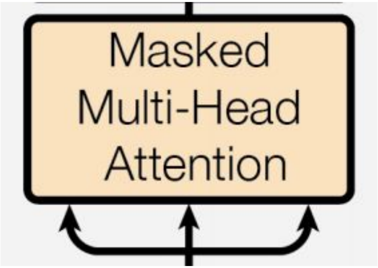
N layers of attention followed by FC

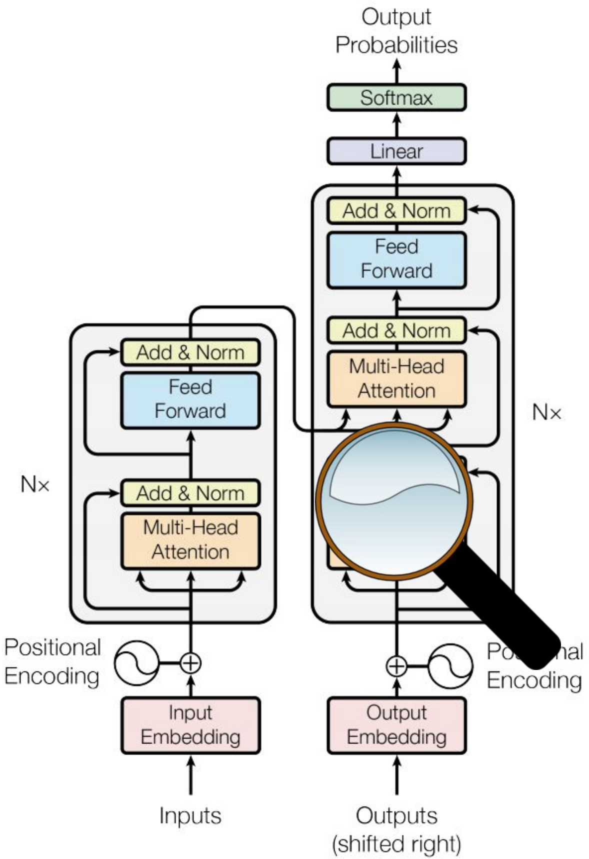




Transformers – a closer look

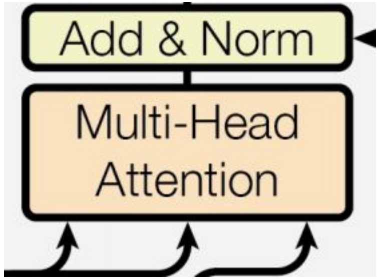
Same as multi-head attention, but masked. Ensures that the predictions for position i can depend only on the known outputs at positions less than i.

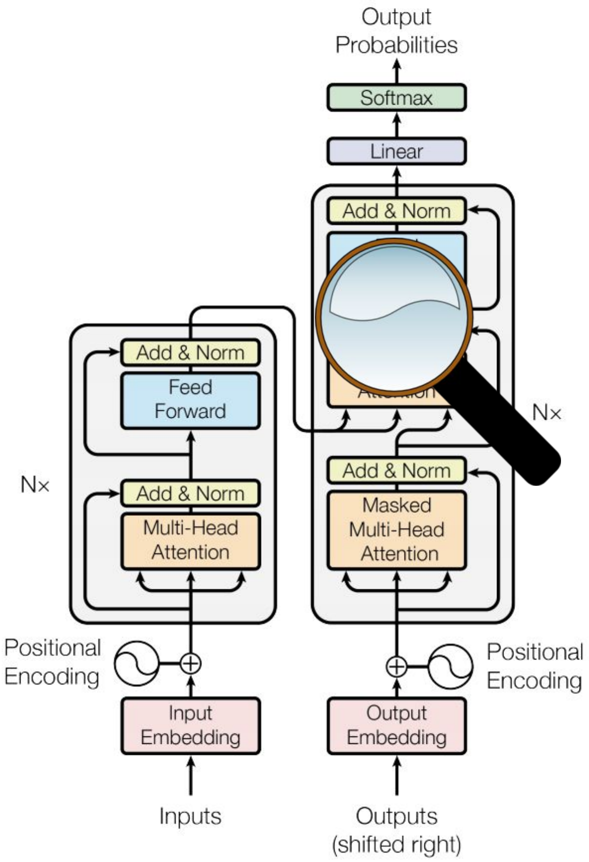




Transformers – a closer look

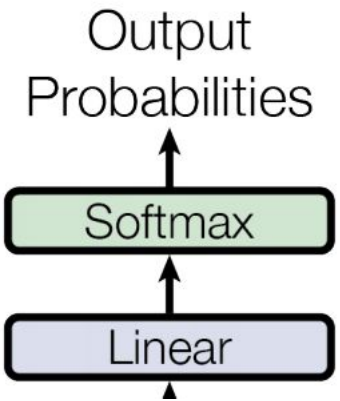
Multi-headed attention between encoder and the decoder.

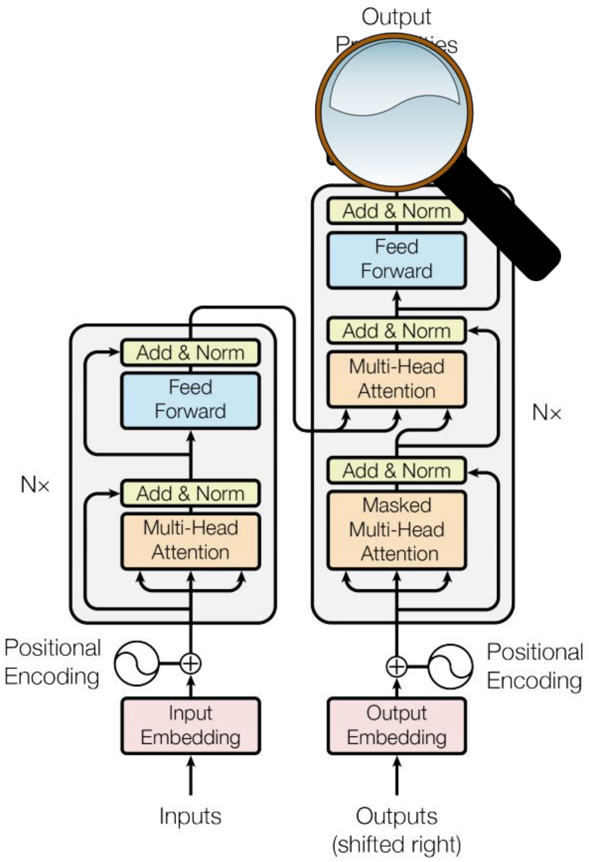




Transformers – a closer look

Projection and prediction.



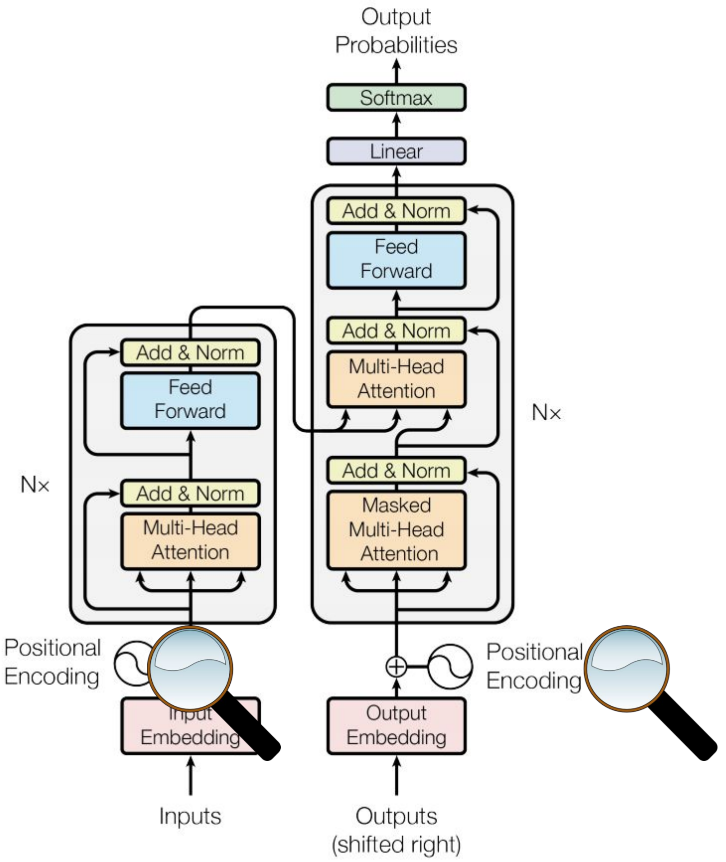


What is missing from self-attention?

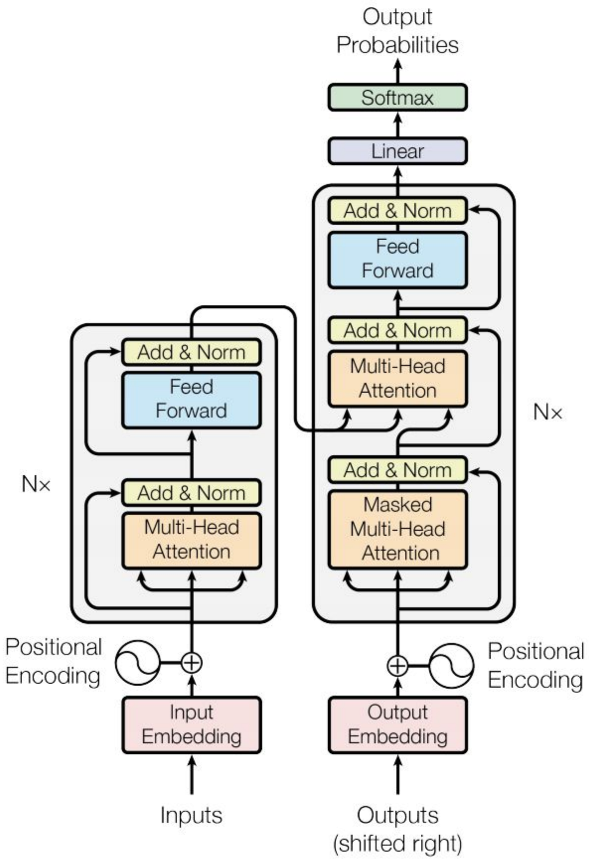
- Convolution: a different linear transformation for each relative position. Allows you to distinguish what information came from where.
- Self-attention: a weighted average.

Transformers – a closer look

Uses fixed positional encoding based on trigonometric series, in order for the model to make use of the order of the sequence



Transformers – a final look



Self-attention: complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

where n is the sequence length, d is the representation dimension and k is the convolutional kernel size.

Self-attention: complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

where n is the sequence length, d is the representation dimension and
k is the convolutional kernel size.

Considering that most sentences have a smaller dimension than the representation dimension (in the paper, it is 512), self-attention is very efficient.

Transformers – training tricks

- ADAM optimizer with proportional learning rate:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

- Residual dropout.
- Label smoothing.
- Checkpoint averaging.

Transformers - results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Transformers - summary

- Significantly improved SOTA in Machine Translation.

Transformers - summary

- Significantly improved SOTA in Machine Translation.
- Launched a new deep-learning revolution in NLP.

Transformers - summary

- Significantly improved SOTA in Machine Translation.
- Launched a new deep-learning revolution in MLP.
- Building block of NLP models like BERT (Google) or GPT-3 (OpenAI).

Transformers - summary

- Significantly improved SOTA in Machine Translation.
- Launched a new deep-learning revolution in MLP.
- Building block of NLP models like BERT (Google) or GPT-3 (OpenAI).
- BERT has been heavily used in Google Search.

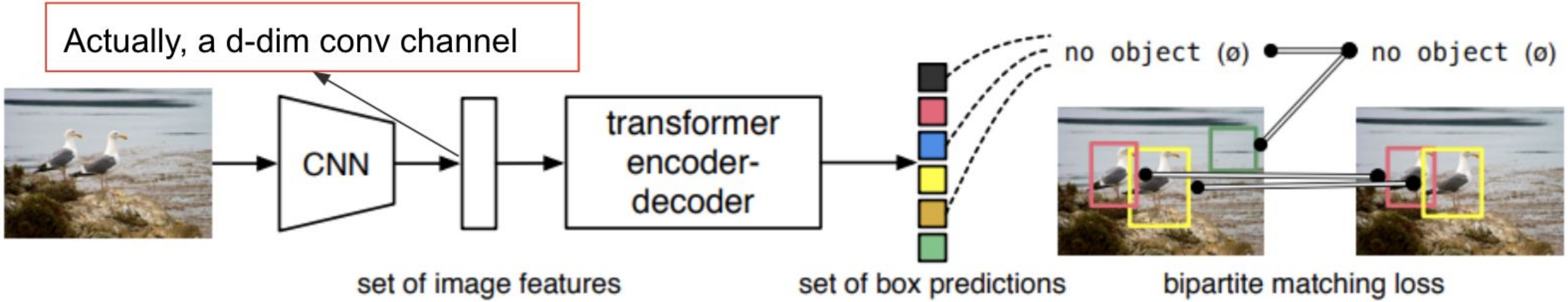
Transformers - summary

- Significantly improved SOTA in Machine Translation.
- Launched a new deep-learning revolution in MLP.
- Building block of NLP models like BERT (Google) or GPT-3 (OpenAI).
- BERT has been heavily used in Google Search.

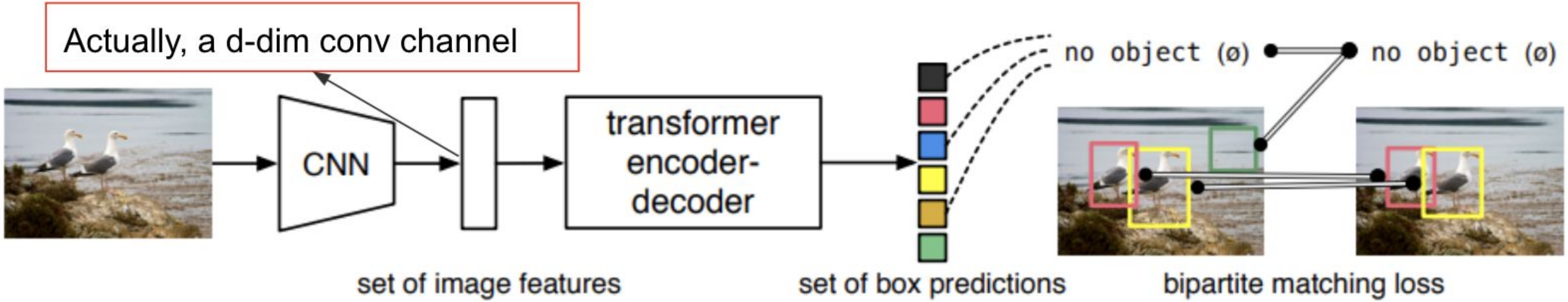
- And eventually made its way to computer vision (and other related fields).

Transformers usage in Computer Vision

DETR – end-to-end object detection with Transformers

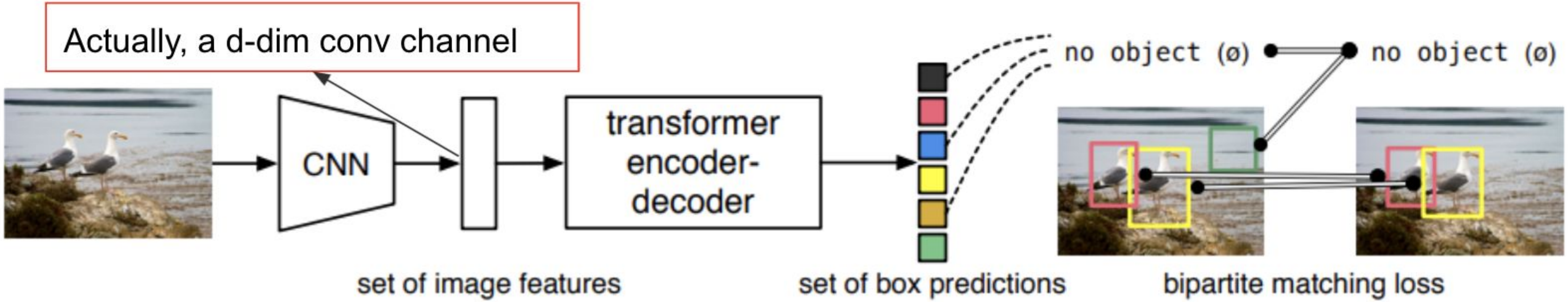


DETR – end-to-end object detection with Transformers



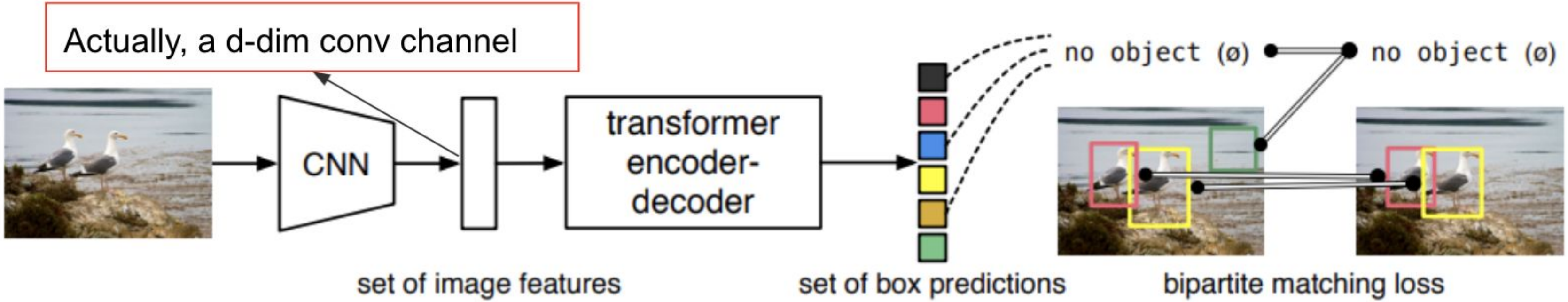
- DETR Predicts in parallel the final set of detections.

DETR – end-to-end object detection with Transformers



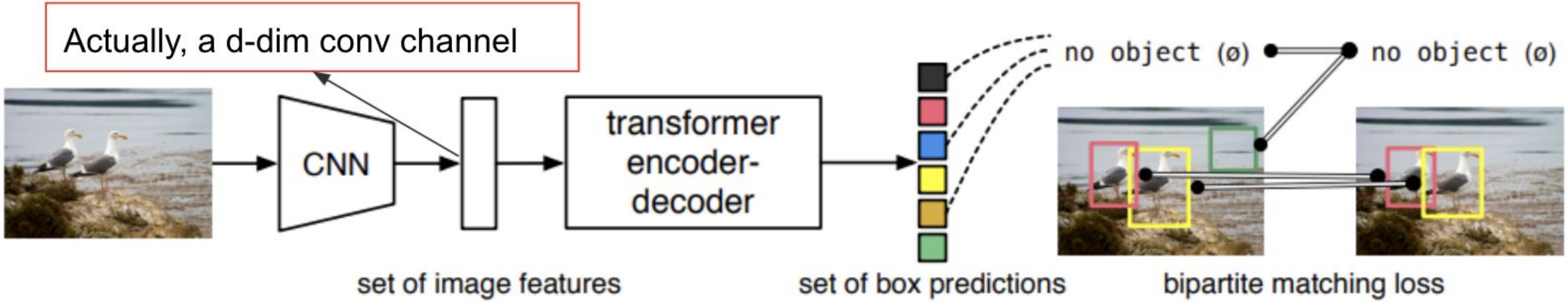
- DETR Predicts in parallel the final set of detections.
- The CNN is used for feature learning, the Transformer is used to make predictions.

DETR – end-to-end object detection with Transformers



- DETR Predicts in parallel the final set of detections.
- The CNN is used for feature learning, the Transformer is used to make predictions.
- During training, bipartite matching uniquely assigns predictions with ground truth boxes.

DETR – end-to-end object detection with Transformers



- DETR Predicts in parallel the final set of detections.
- The CNN is used for feature learning, the Transformer is used to make predictions.
- During training, bipartite matching uniquely assigns predictions with ground truth boxes.
- No need for the annoying NMS ☺.

DETR – the loss function

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

It also contain empty ground-truth

DETR – the loss function

It also contain empty ground-truth

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

Loss for the class

Loss for the bounding-box

$$-\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\mathrm{box}}(b_i, \hat{b}_{\sigma(i)}).$$

DETR – the loss function

It also contain empty ground-truth

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

Loss for the class

Loss for the bounding-box

$$-\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}).$$

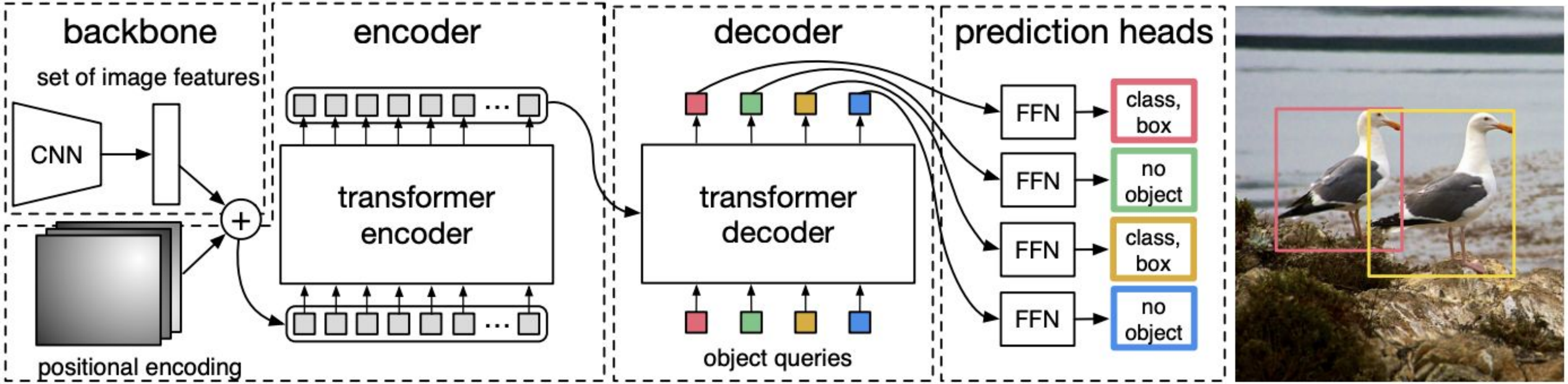
$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

Optimal assignment computed in the first step

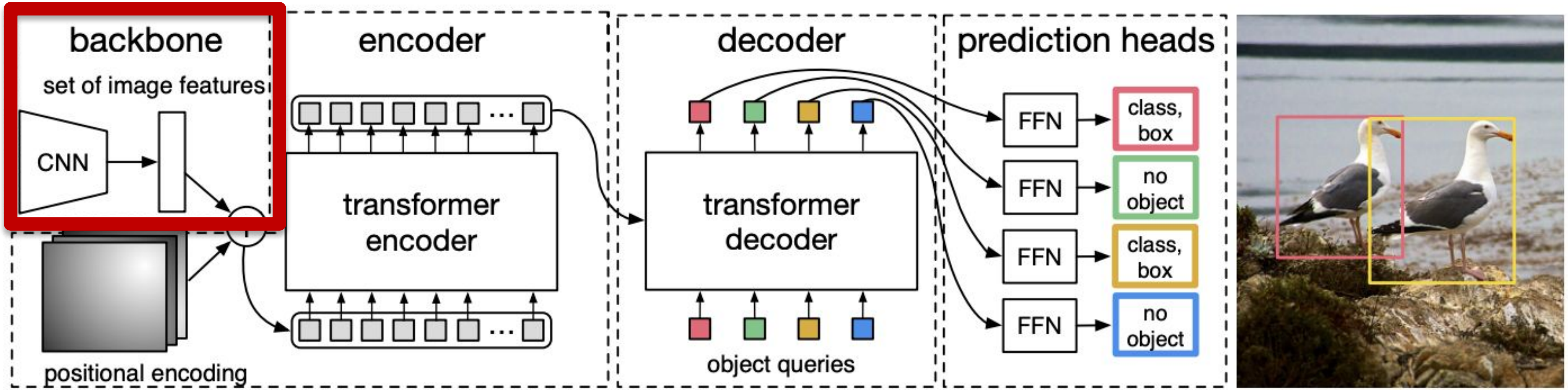
DETR – Bounding Box loss

Bounding box loss. The second part of the matching cost and the Hungarian loss is $\mathcal{L}_{\text{box}}(\cdot)$ that scores the bounding boxes. Unlike many detectors that do box predictions as a Δ w.r.t. some initial guesses, we make box predictions directly. While such approach simplify the implementation it poses an issue with relative scaling of the loss. The most commonly-used ℓ_1 loss will have different scales for small and large boxes even if their relative errors are similar. To mitigate this issue we use a linear combination of the ℓ_1 loss and the generalized IoU loss [38] $\mathcal{L}_{\text{iou}}(\cdot,\cdot)$ that is scale-invariant. Overall, our box loss is $\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$ defined as $\lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}}||b_i - \hat{b}_{\sigma(i)}||_1$ where $\lambda_{\text{iou}}, \lambda_{\text{L1}} \in \mathbb{R}$ are hyperparameters. These two losses are normalized by the number of objects inside the batch.

DETR – a closer look

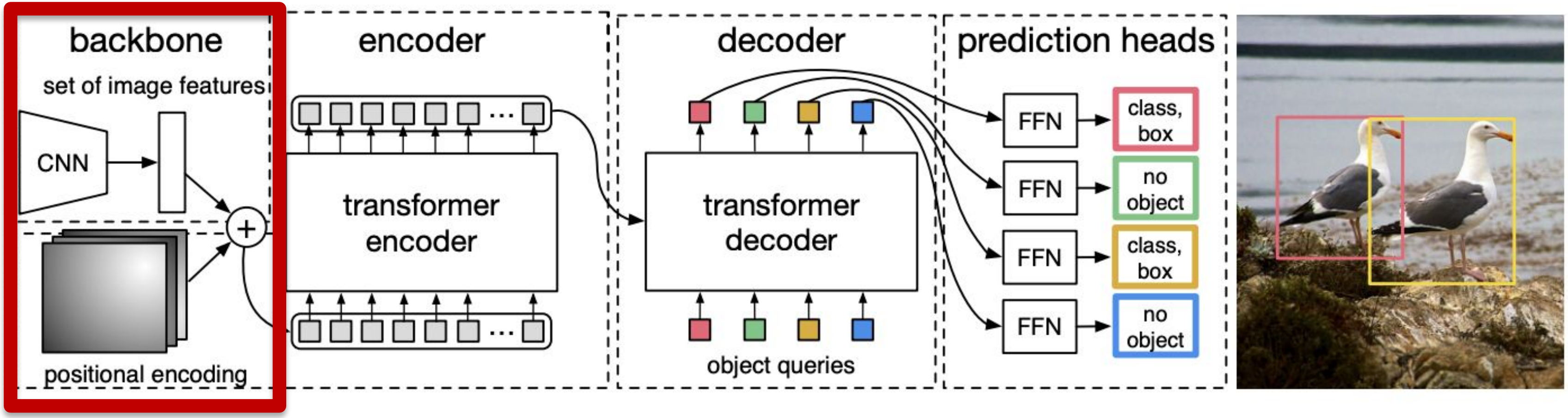


DETR – a closer look



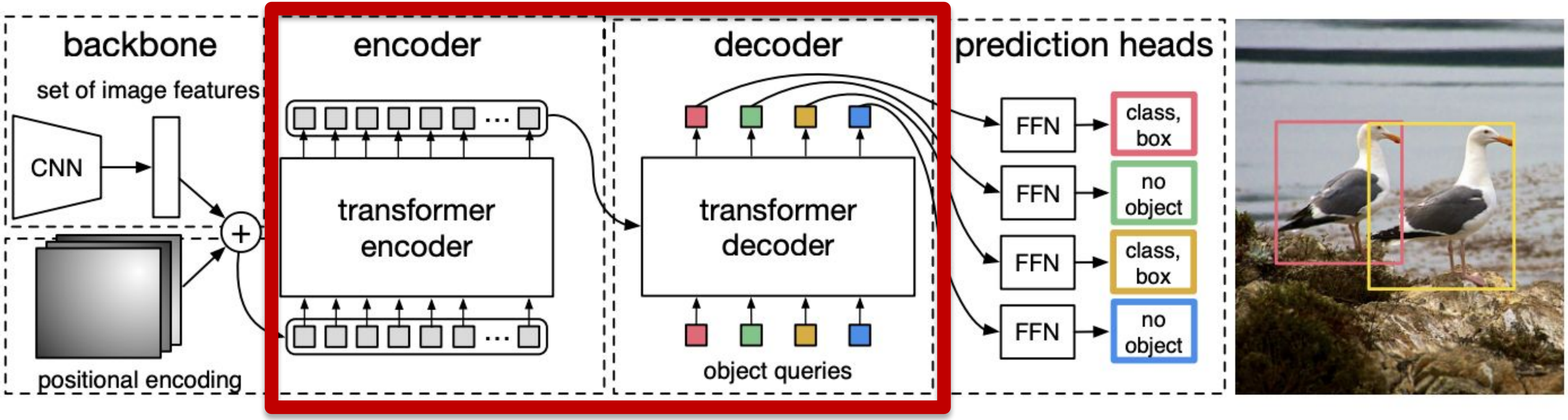
- DETR uses a conventional CNN backbone to learn a 2D representation of an input image.

DETR – a closer look



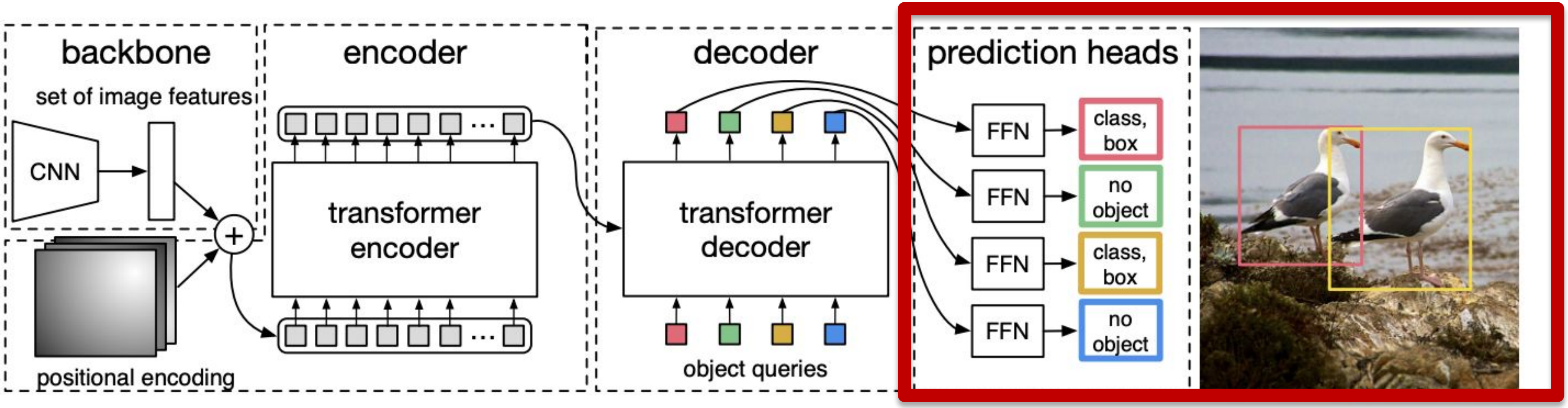
- The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder.

DETR – a closer look



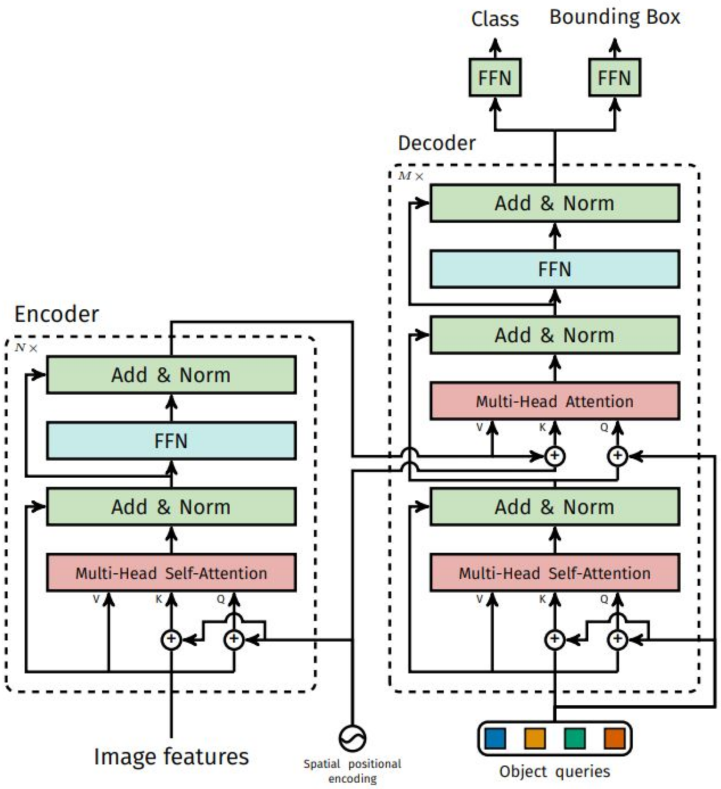
- A transformer first encodes the given input, and then the decoder takes as input a small fixed number of learned positional embeddings, which we call object queries, and additionally attends to the encoder output.

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 – Transformer architecture

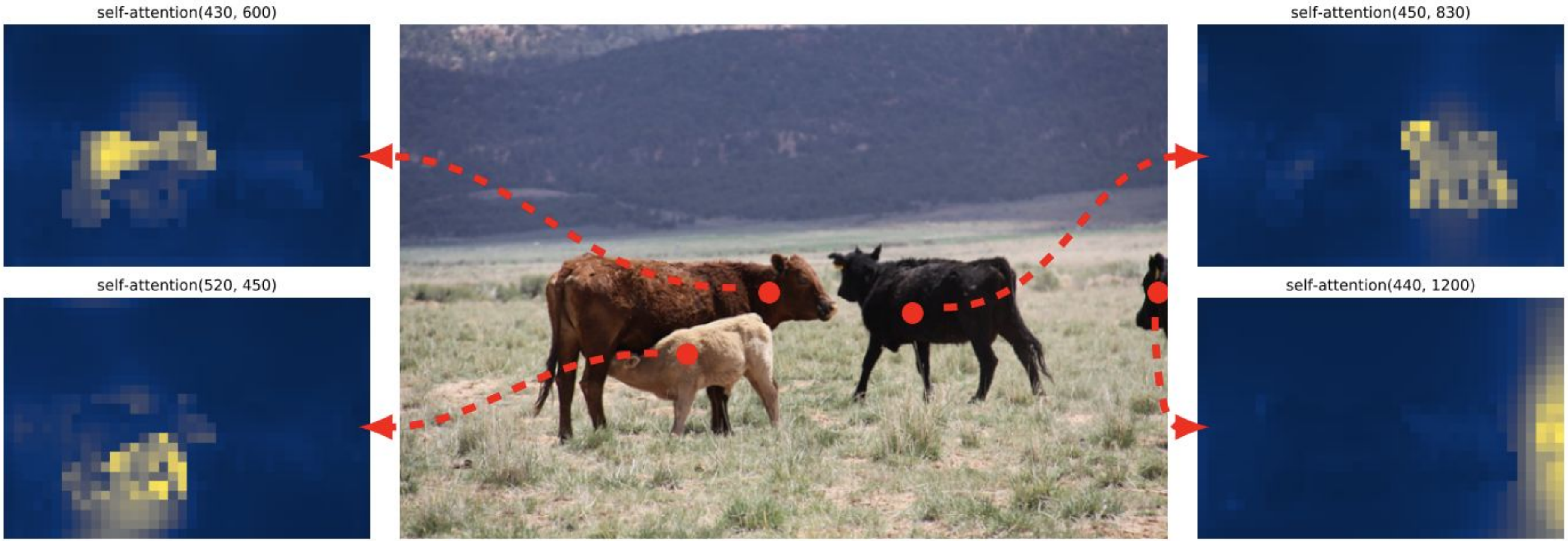


- Very similar to Attention is All you need architecture, with just a few addition made to work for this particular problem.

DETR – Results

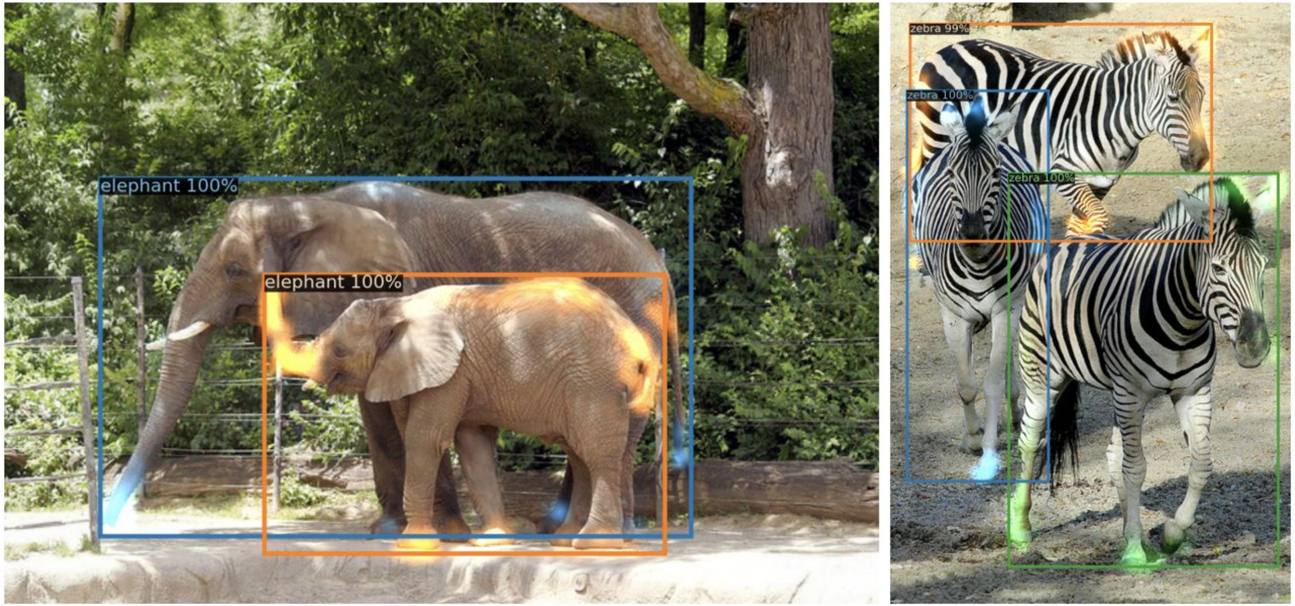
Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster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

DETR – Quantitative results



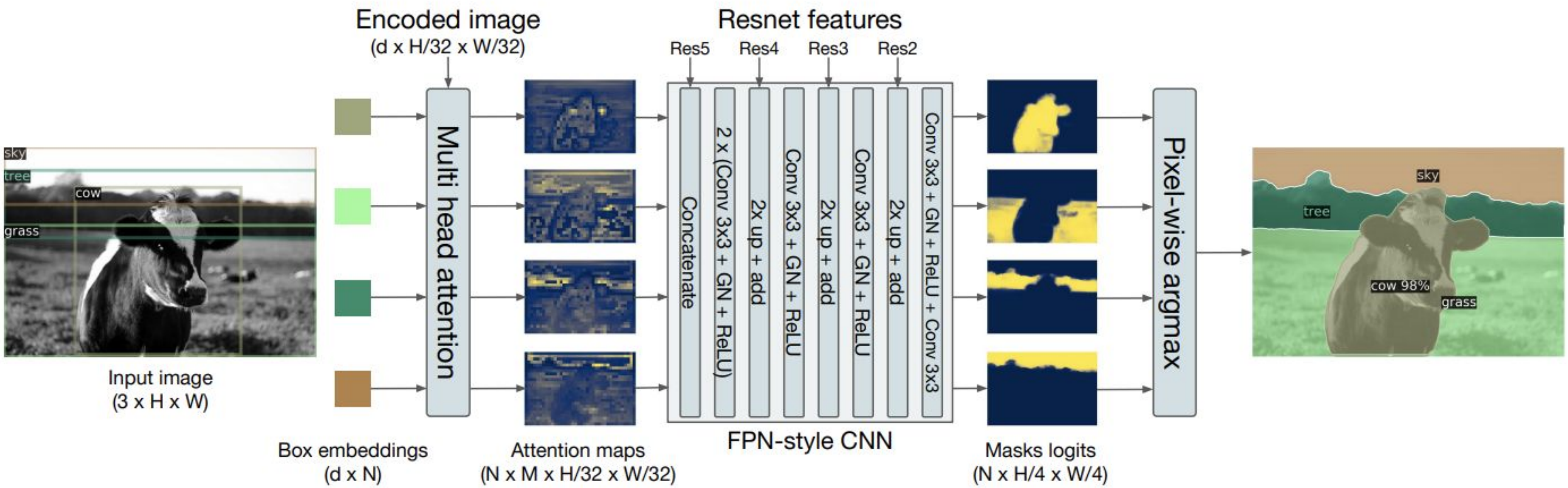
- The encoder is able to separate individual instances. Predictions are made with baseline DETR model on a validation set image.

DETR – Quantitative results



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

DETR used for panoptic semantic segmentation



- A binary mask is generated in parallel for each detected object, then the masks are merged 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

Using DETR in practice

```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)
```

Using DETR in practice

```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)
```

However, training takes 3 days using 16 V100 GPUs.

Recently, there are more efficient modifications.

DETR - summary

- Reaches good results in object detection without any bells and whistles.

DETR - summary

- Reaches good results in object detection without any bells and whistles.
- Gets rid of NMS.

DETR - summary

- Reaches good results in object detection without any bells and whistles.
- Gets rid of NMS.
- With a minor modification can be used for panoptic segmentation.

DETR - summary

- Reaches good results in object detection without any bells and whistles.
- Gets rid of NMS.
- With a minor modification can be used for panoptic segmentation.
- Shows that Transformers can be used in Vision.

Brace yourself – the Transformers are coming

- An image is worth 16x16 words: Transformers for image recognition at scale
- Transgan: Two transformers can make one strong gan
- Tokens-to-token vit: Training vision transformers from scratch on imagenet
- Unsupervised learning of visual features by contrasting cluster assignments

and many many more.

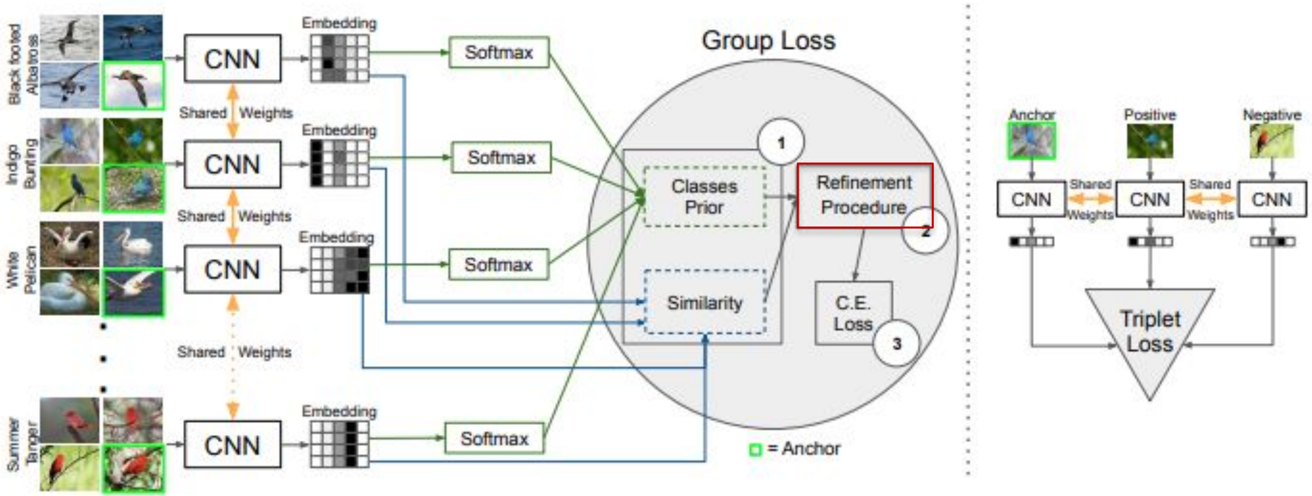
And in our group…

Seidenschwarz, Elezi, Leal-Taixe, Learning Intra-Batch Connections for Deep Metric Learning, ICML 2021

Meinhardt, Kirillov, Leal-Taixe, Feichtenhofer, TrackFormer: Multi-Object Tracking with Transformers, arXiv: 2101.02702

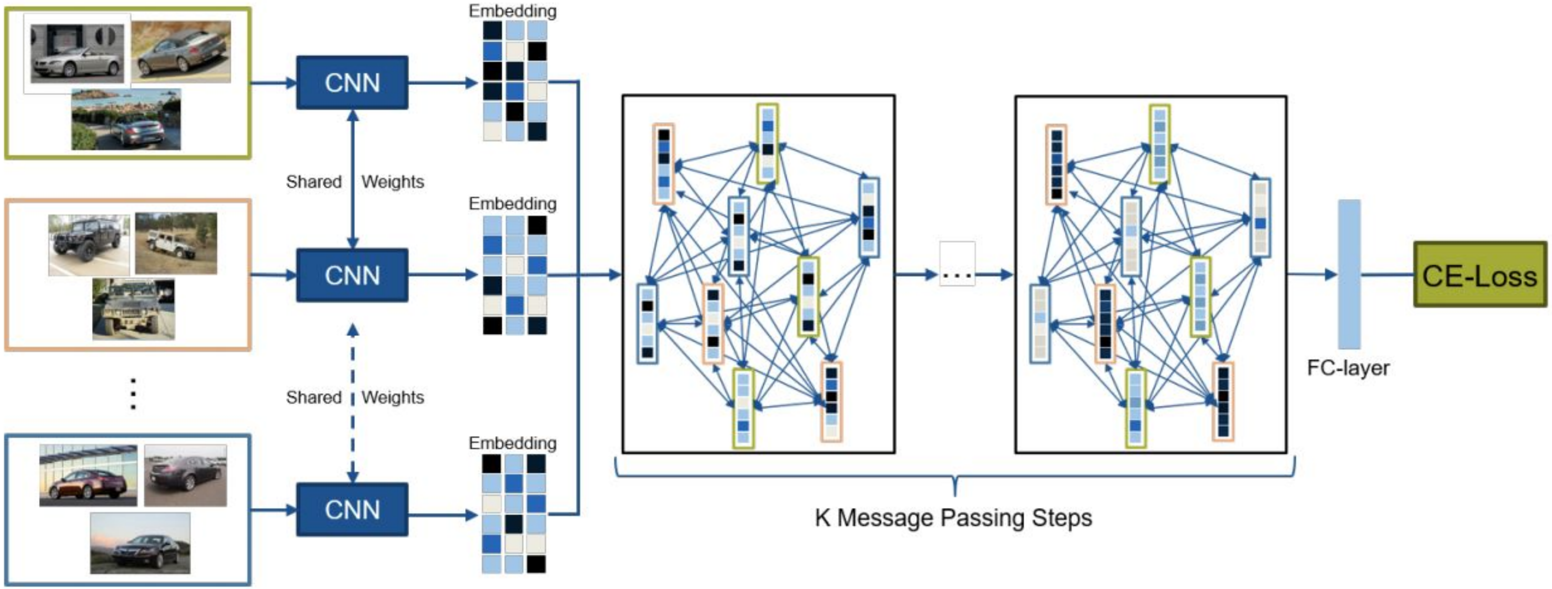
and many more coming soon ☺️

One minute teaser in Learning Intra-Batch Connections



- Remember Group Loss? It had a refinement procedure based on a rule (replicator dynamics)? Why not "learn the rule" instead via a dynamic graph modelled by Transformers?

One min-teaser in Learning Intra-Batch Connections



- A CNN is used to extract features from a batch of images, We model contextual relations via a MPN (implemented by Transformers). In this way, we learn which images should affect the other images more (or less).

SOTA in object retrieval

	CUB-200-2011					CARS196					Stanford Online Products			
Method	**R@1**	**R@2**	**R@4**	**R@8**	**NMI**	**R@1**	**R@2**	**R@4**	**R@8**	**NMI**	**R@1**	**R@10**	**R@100**	**NMI**
Triplet^{64} (Schroff et al., 2015) *CVPR15*	42.5	55	66.4	77.2	55.3	51.5	63.8	73.5	82.4	53.4	66.7	82.4	91.9	89.5
Npairs^{64} (Sohn, 2016) *NeurIPS16*	51.9	64.3	74.9	83.2	60.2	68.9	78.9	85.8	90.9	62.7	66.4	82.9	92.1	87.9
Deep Spectral^{512} (Law et al., 2017) *ICML17*	53.2	66.1	76.7	85.2	59.2	73.1	82.2	89.0	93.0	64.3	67.6	83.7	93.3	89.4
Angular Loss^{512} (Wang et al., 2017) *ICCV17*	54.7	66.3	76	83.9	61.1	71.4	81.4	87.5	92.1	63.2	70.9	85.0	93.5	88.6
Proxy-NCA^{64} (Movshovitz-Attias et al., 2017) *ICCV17*	49.2	61.9	67.9	72.4	59.5	73.2	82.4	86.4	88.7	64.9	73.7	-	-	90.6
Margin Loss^{128} (Manmatha et al., 2017) *ICCV17*	63.6	74.4	83.1	90.0	69.0	79.6	86.5	91.9	95.1	69.1	72.7	86.2	93.8	90.7
Hierarchical triplet^{512} (Ge et al., 2018) *ECCV18*	57.1	68.8	78.7	86.5	-	81.4	88.0	92.7	95.7	-	74.8	88.3	94.8	-
ABE^{512} (Kim et al., 2018) *ECCV18*	60.6	71.5	79.8	87.4	-	85.2	90.5	94.0	96.1	-	76.3	88.4	94.8	-
Normalized Softmax^{512} (Zhai & Wu, 2019) *BMVC19*	61.3	73.9	83.5	90.0	69.7	84.2	90.4	94.4	96.9	**74.0**	78.2	90.6	96.2	91.0
RLL-H^{512} (Wang et al., 2019b) *CVPR19*	57.4	69.7	79.2	86.9	63.6	74	83.6	90.1	94.1	65.4	76.1	89.1	95.4	89.7
Multi-similarity^{512} (Wang et al., 2019a) *CVPR19*	65.7	77.0	86.3	91.2	-	84.1	90.4	94.0	96.5	-	78.2	90.5	96.0	-
Relational Knowledge^{512} (Park et al., 2019a) *CVPR19*	61.4	73.0	81.9	89.0	-	82.3	89.8	94.2	96.6	-	75.1	88.3	95.2	-
Divide and Conquer^{1028} (Sanakoyeu et al., 2019) *CVPR19*	65.9	76.6	84.4	90.6	69.6	84.6	90.7	94.1	96.5	70.3	75.9	88.4	94.9	90.2
SoftTriple Loss^{512} (Qian et al., 2019a) *ICCV19*	65.4	76.4	84.5	90.4	69.3	84.5	90.7	94.5	96.9	70.1	78.3	90.3	95.9	**92.0**
HORDE^{512} (Jacob et al., 2019) *ICCV19*	66.3	76.7	84.7	90.6	-	83.9	90.3	94.1	96.3	-	**80.1**	**91.3**	96.2	-
MIC^{128} (Brattoli et al., 2019) *ICCV19*	66.1	76.8	85.6	-	69.7	82.6	89.1 93.2	-	68.4	77.2	89.4	95.6	90.0	
Easy triplet mining^{512} (Xuan et al., 2020b) *WACV20*	64.9	75.3	83.5	-	-	82.7	89.3	93.0	-	-	78.3	90.7	**96.3**	-
Group Loss^{1024} (Elezi et al., 2020) *ECCV20*	65.5	77.0	85.0	**91.3**	69.0	**85.6**	91.2	94.9	**97.0**	**72.7**	75.1	87.5	94.2	**90.8**
Proxy NCA++^{512} (Teh et al., 2020) *ECCV20*	66.3	**77.8**	**87.7**	**91.3**	**71.3**	84.9	90.6	94.9	**97.2**	71.5	**79.8**	**91.4**	**96.4**	-
Proxy Anchor^{512} (Kim et al., 2020) *CVPR20*	**68.4**	**79.2**	**86.8**	**91.6**	-	**86.1**	**91.7**	**95.0**	**97.3**	-	79.1	**90.8**	**96.2**	-
Proxy Few^{512} (Zhu et al., 2020) *NeurIPS20*	**66.6**	77.6	86.4	-	**69.8**	85.5	**91.8**	**95.3**	-	72.4	78.0	90.6	**96.2**	90.2
Ours^{512}	**70.3**	**80.3**	**87.6**	**92.7**	**74.0**	**88.1**	**93.3**	**96.2**	**98.2**	**74.8**	**81.4**	**91.3**	95.9	**92.6**

Conclusions

- Transformers have revolutionzed the field of NLP, achieving incredible results.

Conclusions

- Transformers have revolutionzed the field of NLP, achieving incredible results.
- Starting from DETR, in the last year, they have massively impacted the field of computer vision.

Conclusions

- Transformers have revolutionzed the field of NLP, achieving incredible results.
- Starting from DETR, in the last year, they have massively impacted the field of computer vision.
- Complementing CNNs, they have reached SOTA (or near SOTA) results in object retrieval, person re-ID, multi-object tracking, image generation.

Conclusions

- Transformers have revolutionized the field of NLP, achieving incredible results.
- Starting from DETR, in the last year, they have massively impacted the field of computer vision.
- Complementing CNNs, they have reached SOTA (or near SOTA) results in object retrieval, person re-ID, multi-object tracking, image generation.
- And recently, they have started replacing CNNs (An image is 16x16 words, TransGAN etc).