Introduction

The Attention mechanism [2] was proposed by Bengio's team in 2014 and has been widely used in various fields of deep learning in recent years, such as for capturing the receptive field on images in computer vision or used in NLP to locate key tokens or features. Transformer architectures have achieved state-of-the-art results in many natural language processing tasks. The BERT[3] algorithm recently proposed by the Google team for generating word vectors has achieved significant improvements in 11 NLP tasks, which is the most exciting news in the field of deep learning in 2018. The most important part of the BERT algorithm is the concept of Transformer. In the field of computer vision, CNN has been the dominant model for vision tasks since 2012. With the emergence of more and more efficient structures, computer vision and natural language processing are more and more convergent. Using Transformer to complete vision tasks has become a new research direction to reduce the complexity of the structure, explore scalability and improve training efficiency.

Here are a few of the well-known projects and papers about the application of Transformer structure in the computer vision:

- DETR (End-to-End Object Detection with Transformers): Transformers for object detection and segmentation. [6]
  Vision Transformer (An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale): Transformer for image classification. [8]
- Image GPT (Generative Pretraining from Pixels): Transformer for pixel-level image completion. [9]
- End-to-end Lane Shape Prediction with Transformers: Transformer for lane marking detection in autonomous driving. [10]
- Deformable DETR: Deformable Transformers for End-to-End Object Detection. [5]

In this blog, two famous papers about the Transformer and its applications in computer vision will be introduced. The first one is *Attention Is All You Need* [2] which is the most well-known introduction paper to the detailed structure of Transformer. The second one is *Deformable DETR: Deformable Transformers for End-to-End Object Detection* [5] which is the newest application of Transformer structure in object detection.

Paper 1: *Attention Is All You Need*

As the title of the paper says, the traditional CNN and RNN are abandoned in the Transformer, and the entire network structure is completely composed of the Attention mechanism. More precisely, the Transformer consists of and only consists of Self-Attention and Feed Forward Neural Network. A trainable neural network based on Transformer can be built by stacking Transformers. The author's experiment is to build an encoder-decoder with 6 layers of encoder and decoder, a total of 12 layers, and achieve a new high BLEU value in machine translation. [2]

The reason why the author uses the Attention mechanism is to consider that the calculation of RNN (or LSTM, GRU, etc.) is limited to sequential. RNN-related algorithms can only be calculated from left to right or from right to left. This mechanism brings up two problems:

1. The calculation of time slice ***t*** depends on the calculation result of time ***t - 1***, which limits the parallel ability of the model. [2]
2. Information will be lost in the process of sequential calculation. Although the structure of gate mechanisms such as LSTM alleviates the problem of long-term dependence to a certain extent, LSTM is still powerless for particularly long-term dependence phenomena. [2]

The proposal of Transformer solves the above two problems. First, it uses the Attention mechanism to reduce the distance between any two positions in the sequence to a constant. Second, it is not an RNN-like sequential structure, so it has better parallelizability and conforms to existing GPU frameworks. The definition of Transformer given in the paper is *"Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution"* [2]

Unfortunately, the author's paper is difficult to understand, especially the structural details and implementation of Transformer are not explained clearly. Especially query ***Q,*** key ***K,*** value ***V*** in the paper, the author does not explain what it means. By consulting the information, I found a very good technical blog explaining Transformer [4]. A lot of the figures in this article are also taken from the blog.

1. Transformer Architecture

The experiment for verifying Transformer in the paper is based on machine translation. Next, we will use machine translation as an example to analyze the structure of Transformer in detail. Transformer can be summarized as Figure 1

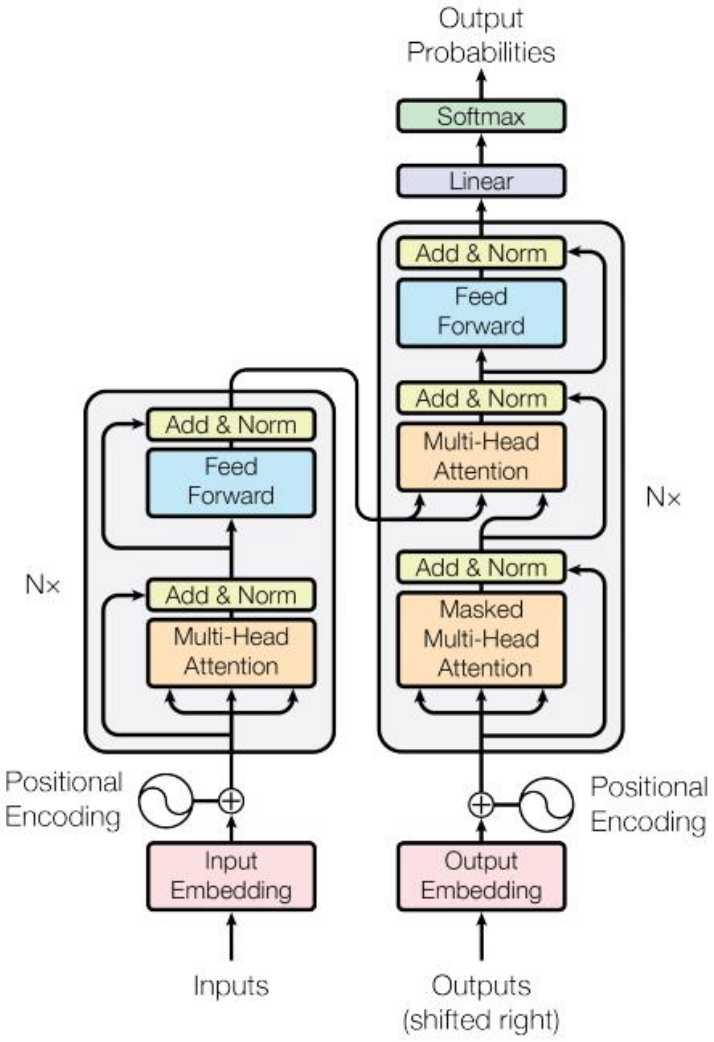

Figure 1: The overall structure of Transformer model [2]

Encoder: The input of the encoder is the embedding of the word plus the position encoding, and then entering a unified module, which can be repeated many times (N times, N layers). Each module can be divided into an Attention layer and a fully connected layer with some additional processing: Residual Connection and then a layer Normalization.

Decoder: The first input of decoder is the prefix information, and the next one is the embedding produced last time with position encoding, and then entering a module that can be repeated many times. This module can be divided into three parts. The first block is also the Attention layer, the second block is Cross-Attention, not Self-Attention, and the third block is the fully connected layer. Residual Connection and then a layer Normalization are also used.

Output: The final output must pass through the Linear layer (full connection layer), and then make predictions through Softmax.

Transformer is essentially an Encoder-Decoder structure, then Figure 1 can be represented as the structure of Figure 2:

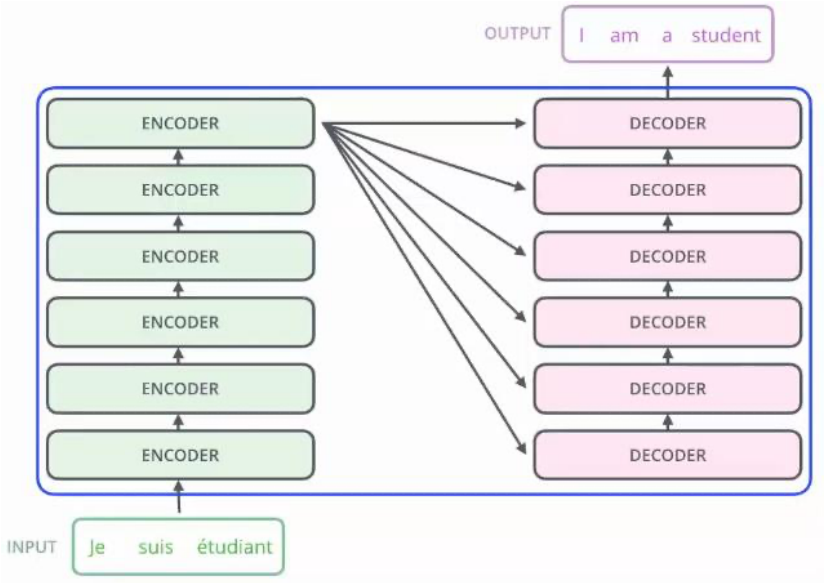

Figure 2: Transformer encoder-decoder [4]

1.1 Encoder

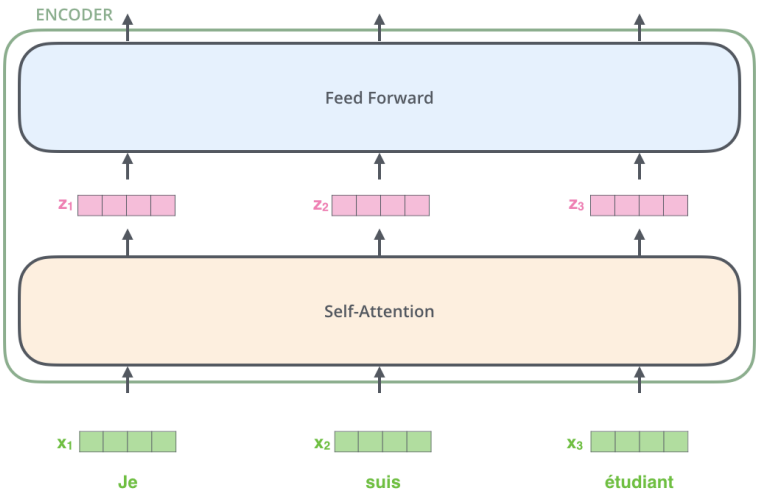

Figure 3: Introduction of encoder [4]

In the Transformer's encoder, shown in figure 3, the input data first passes through a module called “Self-Attention” to obtain a weighted feature vector *Z*. This *Z* is the result of equation 1:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

The first time to see this equation, you may be confused. In a later blog, we will reveal the actual meaning behind this equation.

After getting *Z*, it will be sent to the next module of the encoder, the Feed Forward Neural Network. This full connection layer has two layers. The first layer is ReLU activation function, and the second layer is a linear activation function. The Feed Forward Neural Network can be expressed as:

$$FFN(Z) = \max(0, ZW_1 + b_1)W_2 + b_2 \tag{2}$$

1.1.1 Self-Attention mechanism

Self-Attention is the core content of Transformer. Its core content is to learn a weight for each word of the input vector. For example, in the following example, we want to identify what "it" refers to. After using "Self-Attention", a weighting situation similar to Figure 4 can be obtained,

```
The animal didn't cross the street because it was too tired
```

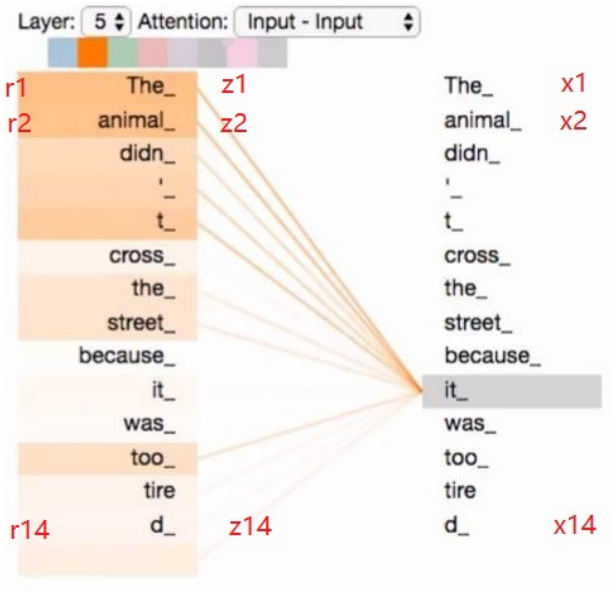

Figure 4: Example of using self-attention [4]

In self-attention, each word has 3 different vectors, which are Query vector $\boldsymbol{Q}$, Key vector $\boldsymbol{K}$ and Value vector $\boldsymbol{V}$, all of which are smae in length. They are obtained by multiplying the embedding vector X by three different weight matrixes $\boldsymbol{W^Q}$, $\boldsymbol{W^K}$, $\boldsymbol{W^V}$, by three different weight matrixes, where the dimensions of the three matrixes are also the same which is 512 x 64. [2]

Therefore, what does Query $\boldsymbol{Q}$, Key $\boldsymbol{K}$, Value $\boldsymbol{V}$ mean? What role do they play in Attention's computation? Let's first look at the calculation method of Attention. The whole process can be divided into 7 steps according to figure 5:

1. Convert input words into embedding vectors
2. According to the word embedding vector, three vectors $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are obtained
3. Calculate a score for each word embedding vector: $\boldsymbol{Score} = \boldsymbol{Q} * \boldsymbol{K}$
4. In order to stabilize the gradient, Transformer uses score normalization, which is divided by $\sqrt{\boldsymbol{d_k}}$ where $\boldsymbol{d_k}$ is the dimension of $\boldsymbol{K}$
5. Apply a softmax activation function to the score
6. The softmax score is multiplied (dot product) by the Value $\boldsymbol{V}$ to obtain the weighted score $\boldsymbol{V}$ of each input word vector
7. After the addition, the final output result $\boldsymbol{Z}$ is obtained: $\boldsymbol{Z} = \sum \boldsymbol{V}$

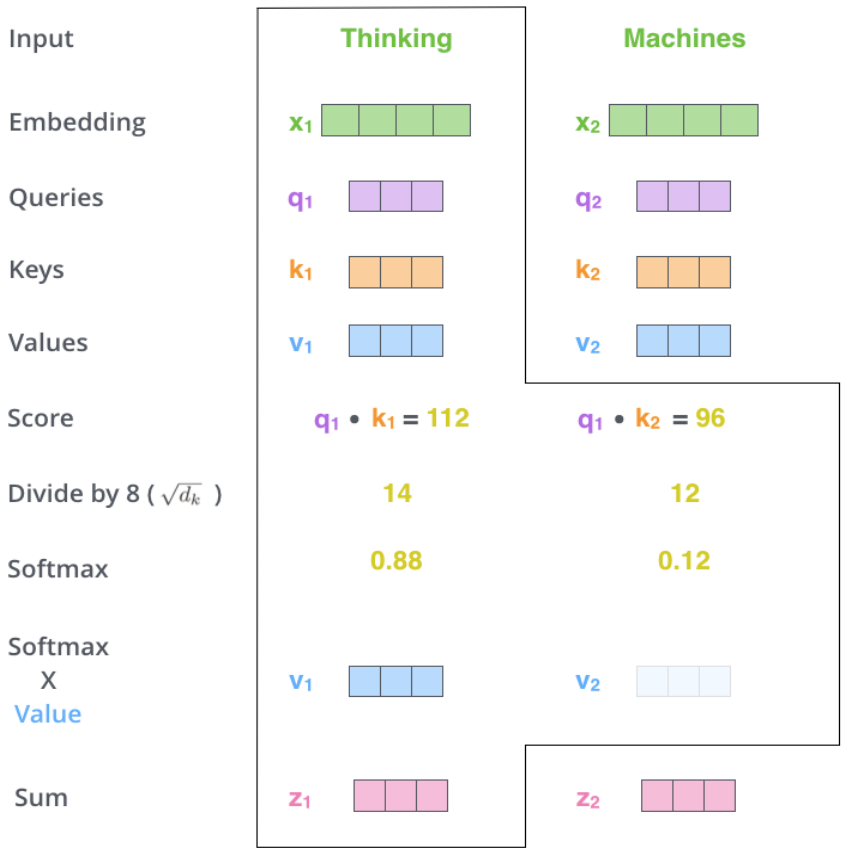

Figure 5: The step-by-step computing of self-attention [4]

From my perspective, the concepts of Query, Key, and Value are taken from the information retrieval system. For a simple example, when you search for a product: “a red jacket worn by young ladies in winter” on an e-commerce platform, the content you enter on the search engine is Query, and then the search engine matches you with a Key such as the type, color, description of product based on the Query and then get the matching score which is the Value according to the similarity between Query and Key.

$\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ in self-attention also play a similar role. In matrix calculation, dot product is one of the methods to calculate the similarity between two matrices. Therefore, in equation 1, we use $\boldsymbol{QK^T}$ to calculate the similarity. The next step is to match the output according to the similarity. Here, the weighted matching method is used, and the weight is the similarity between the query and the key.

1.1.2 multi-headed Attention

If we use different $\boldsymbol{W^Q}$, $\boldsymbol{W^K}$, $\boldsymbol{W^V}$, we can get different $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$. Multi-headed Attention refers to the use of many different $\boldsymbol{W^Q}$, $\boldsymbol{W^K}$, $\boldsymbol{W^V}$ to get different $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$. So what are the benefits of this? Firstly, it expands the model’s ability to focus on different positions. If there are multiple $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$, Attention can be viewed from multiple different aspect. Secondly, it also gives the attention layer multiple “representation subspaces” because each of these sets of $\boldsymbol{W^Q}$, $\boldsymbol{W^K}$, $\boldsymbol{W^V}$ is randomly initialized. [2]

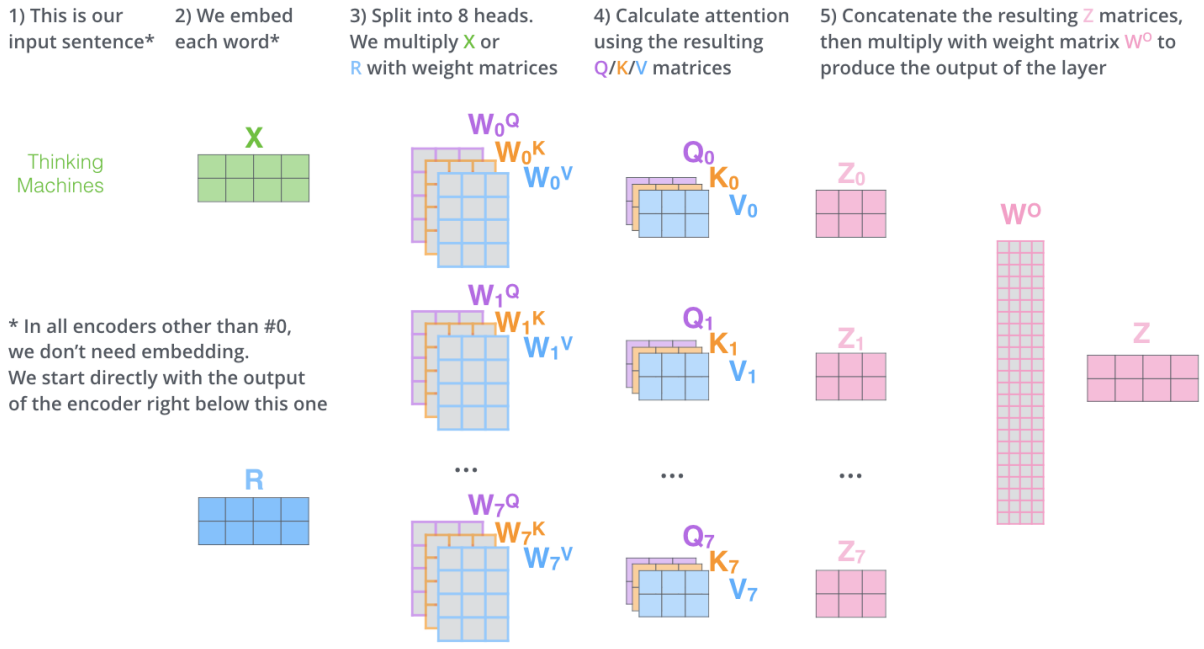

Figure 6: multi-headed Attention explanation [4]

Now that there are multiple versions $\boldsymbol{Z}$ of one input $\boldsymbol{X}$, how to combine them into a vector? we concat multiple versions of $\mathbf{Z}$ into a long vector, and a fully connected network is used to obtain a short vector. The multi-headed Attention is expressed by the equation (3):

$$Attention(Q,K,V) = Concat(head_1, head_2, head_3, \ldots\ldots head_h)W^0 \quad (3)$$
$$where\ head_i = Attention(QW_I^Q, KW_I^K, VW_I^V)$$

1.1.3 Embedding Inputs

The encoder input is the embedding of the word, and there are usually two options:

1. Use Pre-trained embeddings and solidify, which is actually a Lookup Table
2. Initialize embeddings randomly (of course you can also choose the result of Pre-trained), but set it to trainable parameter. In this way, the embeddings are continuously improved during the training process.

Transformer chooses the latter. [2]

1.1.4 Position Embedding

The Transformer models we have introduced so far do not have the ability to capture sequential information, which means that the Transformer will get similar results no matter what the sentence sequences is. In other words, Transformer is just a more powerful bag of words model.

In order to solve this problem, the Position Embedding is introduced in the paper when encoding the word vector. Specifically, the Position Embedding adds the positional information of the word to the word vector, so that the Transformer can distinguish words in different positions.

To know the distance and position of words, you need to know the coordinates representation of the words. There are many different ways to measure distance, In Transformer, the sin and cos function of different frequencies is used according to equation 4.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (4)$$

1.1.5 Skip Connection and Layer Normalization

The “Add & Norm” module is connected to the back of each sub-module of the Encoder side and the Decoder side, where “Add” represents residual connection, and “Norm” represents Layer Normalization.

The skip connection, first mentioned in ResNet of computer vision, mainly solve the problem that, when the network is very deep, the backward propagation of gradient will become weaker and weaker, and training will be very difficult. If a residual connection is generated, the gradient will be passed back more clearly. The residual connection are not necessary, but in Transformer, the author recommends using residual connection after Self-Attention and Normalized Feed Forward Network. [2]

At the same time, rather than Batch Normalization, Layer Normalization, a regularization strategy to avoid network overfitting, is also used.

1.2 Decoder

The Decoder in the paper is also a stack of N=6 layers. Divided into 3 SubLayer, Encoder and Decoder have three two differences which is shown in figure 7:

1. Decoder SubLayer-1 uses the "Masked" Multi-Headed Attention mechanism to prevent the model from seeing the data to be predicted and prevent leakage.
2. SubLayer-2 is an Encoder-Decoder Multi-head Attention.

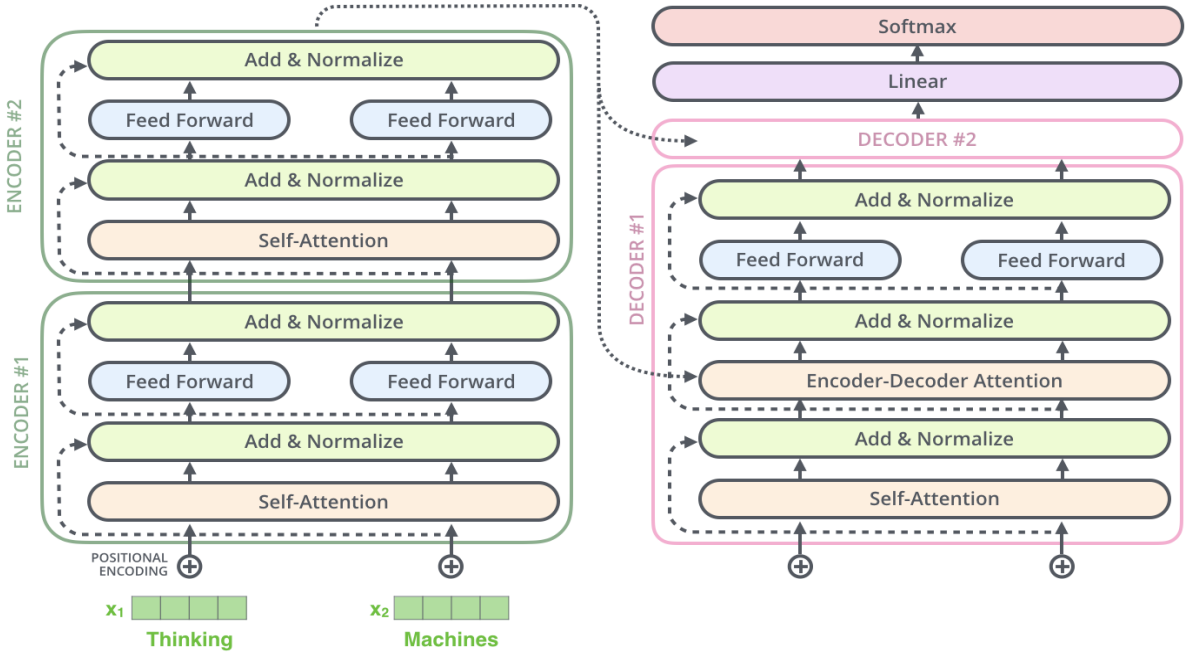

Figure 7: Detailed encoder-decoder structure comparison [4]

1.2.1 Masked Multi-Headed Attention

In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. The sequence mask is to prevent the Decoder from seeing future information. That is, for a sequence, when time step is *t*, the decoding output should only depend on the output before *t*, but not on the output after *t*. Therefore, we need to think of a way to hide the information after time step *t*. It is also very simple: generate an upper triangular matrix, and the values of the upper triangle are all 0. By applying this matrix to each sequence, we can achieve our purpose by masking future positions. [2]

1.2.2 Encoder-Decoder Multi-head Attention

The prediction process of Attention is the same as the normal Encoder-Decoder mode, except that the RNN is replaced with Self-Attention. In SubLayer-2 (the second attention layer in the Decoder), the input not only has the outputs of the previous layer, but also the outputs from the Encoder (otherwise, the output of the Encoder will be useless), and then use the vector generated by the Encoder as the key and use the previous outputs of decoder as the query, and then perform Self-Attention. It is equivalent to the fact that the Encoder tells us what the key and value are, and what decoder is doing now is to generate the query. [2]

1.3 The Final Linear and Softmax Layer

The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders into a larger vector called a logits vector. The softmax layer then turns those vectors into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

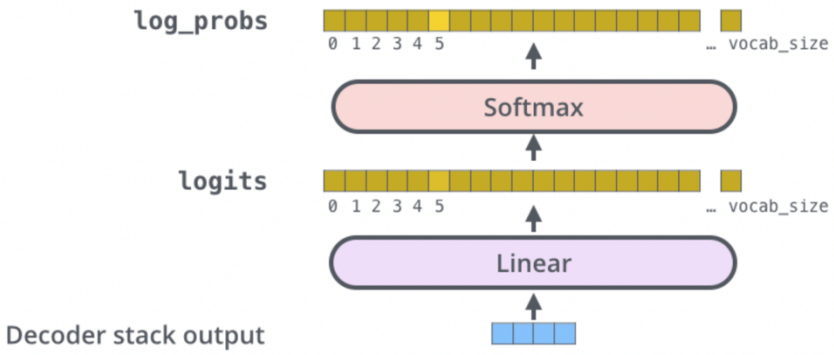

Figure 8: The Final Linear and Softmax Layer [4]

2. Why Self-Attention

Self-Attention layers and recurrent/convolutional layers are compared in figure 9 to illustrate the benefits of Self-Attention. Suppose an input sequence (x1, x2, x3……, xn) uses Self-Attention Layer, Recurrent Layer and Convolutional Layer separately to map to a sequence (z1, z2, z3……, zn) with the same length of the input. [2]

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Figure 9: Computation efficiency of Self-Attention layers and recurrent/convolutional layers [4]

2.1 Parallel Computing

The Self-Attention layer connects all positions with a constant-level $O(1)$ sequential operation, while recurrent layers needs a $O(n)$ sequential operation.

2.2 Complexity Per Layer

In terms of computational complexity, the self-attention layer is faster than the recurrent layer when the sequence length ***n*** is less than the representation dimension ***d***, which is a most common situation in machine translations. [2]

3. Training and Results

3.1 Training data and batch processing

The paper is trained on the standard WMT 2014 English-German dataset, which consists of approximately 4.5 million sentence pairs. Sentences are encoded using a byte-pair encoding that has a shared source target vocabulary of approximately 37,000 tokens. For English-French, the paper uses the larger WMT 2014 English-French dataset, which contains 36 million sentences and splits the tokens into a 32,000-word vocabulary. Sentence pairs are grouped together in batches of similar sequence length. Each training batch contains a set of sentence pairs containing approximately 25,000 source tokens and 25,000 target tokens. [2]

3.2 Hardware and Schedule

The paper trains the model on a machine with 8 NVIDIA P100 GPUs. For the base model using the hyperparameters described in this paper, each training step takes about 0.4 seconds. Paper trains the base model for a total of 100,000 steps or 12 hours. [2]

3.3 Optimizer

The paper uses the Adam optimizer with $\beta_1 = 0.9,\ \beta_2 = 0.98\ and\ \varepsilon = 10^{-9}$. During training, the paper changes the learning rate according to the following equation 5:

$$lrate = d_{model}^{-0.5} * \min\left(step_{num}{}^{-0.5}, step_{num} * warm\_steps^{-1.5}\right) \quad (5)$$

3.4 Regularization

The paper uses Residual Dropout with a Dropout probability $P_{drop} = 0.1$ and Label Smoothing with a label smoothing of value $\epsilon_{ls} = 0.1$. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score. [2]

3.5 Results

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

Figure 10: The performance of Transformer [2]

On the WMT 2014 English to German translation task, the Transformer model (big) outperforms the best previously reported models by more than 2.0 in BLEU score, achieving a BLEU of 28, training on 8 P100 GPUs takes 3.5 days. On the WMT 2014 English-French translation task, the Transformer model (big) achieves a BLEU score of 41.0, which is better than all previously released models, and the training cost is less than 1/4 of the previous models. [2]

4. Conclusion

The Advantages of Transformer structure can be summarized as following points:

- The computational complexity of Transformer is lower than that of RNN and CNN.
- The structure of Transformer is more suitable for GPU because its ability of Parallel computing.
- From the point of view of the path length to be passed to calculate a sequence length of n, CNN needs to increase the number of convolution layers to expand the receptive field, RNN needs to be calculated one by one from 1 to n, but Self-attention only needs one step of matrix calculation.
- Self-Attention can solve long-term dependencies problems better than RNN. If the amount of calculation is too large, such as the case where the sequence length n is greater than the sequence dimension d, the sliding window can also be used to limit the calculation amount of Self-Attention.

Paper 2: Deformable DETR: Deformable Transformers for End-to-End Object Detection

Deformable DETR [5] is one of the most famous DETR [6] variants in the past two years. Here we analyze the essence of Deformable DETR in detail from the paper. In the past two years, from Attention to NAS, the computer vision industry has been pulled to a new height by Transformer. As a method that has attracted much attention in Transformer in the past two years, DETR has also attracted many people to change and improve it. One of the more famous is Deformable DETR. DETR proposes to remove artificial components in object detection methods, which can also maintain excellent performance. However, since the Transformer attention module can only process image feature maps limitedly, its convergence speed is relatively slow and the feature map resolution is limited. [5] To alleviate these problems, the author proposes Deformable DETR, whose attention module will only focus on a small number of key sampling points around the target box. Deformable DETR can achieve better performance than DETR (especially on small objects). Extensive experiments on the COCO benchmark demonstrate the effectiveness of the Deformable DETR. [5]

1. Why Deformable DETR

Method	Epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	Training GPU hours	Inference FPS
Faster R-CNN + FPN	109	42.0	62.1	45.5	26.6	45.4	53.4	42M	180G	380	26
DETR	500	42.0	62.4	44.2	20.5	45.8	61.1	41M	86G	2000	28

Figure 11: Comparison between DETR and Faster R-CNN + FPN [5]

DETR has several shortcomings which have been shown in the Comparison between DETR and Faster R-CNN + FPN in figure 11:

1. Extremely long training time: Compared with existing detectors, DETR requires longer training to achieve convergence (500 epochs), which is 10-20 times slower than Faster R-CNN. This is because, when the Transformer is initialized, the attention weights assigned to all feature pixels are almost equal. Long training epochs is necessary for the attention weights to be learned to focus on sparse meaningful locations. [5]

2. DETR has poor performance in small object detection. Existing object detectors usually have multi-scale feature maps. Small objects are usually detected on high-resolution feature maps. However, DETR does not use multi-scale feature maps to detect, mainly because high-resolution feature maps will increase the unacceptable computational complexity of DETR. Transformer is accompanied by high computational complexity and space complexity when calculating attention weights. Especially in the encoder part, it has a quadratic relationship with the number of feature pixels, so it is difficult to deal with high-resolution features. This is the reason why DETR has poor performance in detecting small objects. [5]

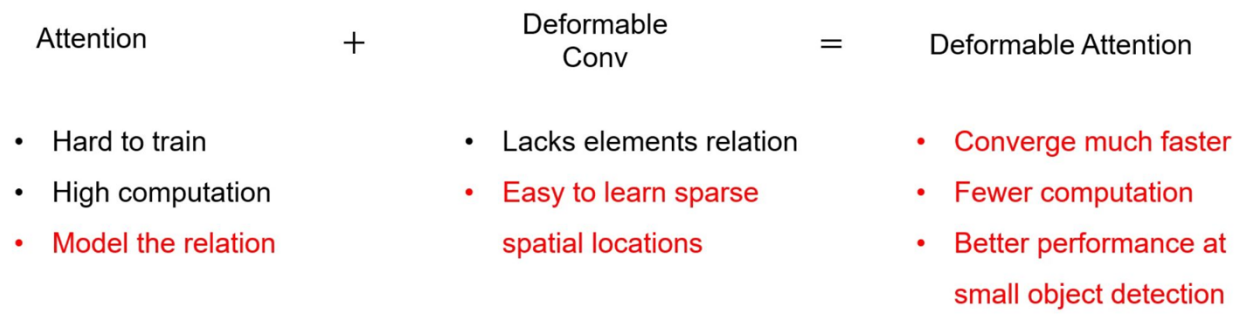

Figure 12: The advantage of the combination of Attention and Deformable Conv [7]

Based on this, a Deformable DETR model is proposed. Deformable DETR combines the spatial sparse sampling advantage of deformable conv with the ability of transformer to model relationships and attentions between elements. The high computational complexity of DETR comes from the attention calculation of the transformer structure in the global context. The authors note that, although this attention is calculated in the global context, at the end, a certain visual element will only establish a strong connection with a small number of other visual elements through weights. [5]

Therefore, deformable DETR no longer uses the global attention calculation, and only calculates the attention of a small number of points around the reference point instead of calculating the global one, which can reduce the amount of calculation and speed up the convergence.

2. Deformable DETR Architecture

2.1 Deformable Attention Module

The core problem of applying Transformer attention to image feature maps is that it will look at all possible locations and increase computing complexity. To address this problem, the paper proposes a deformable attention module. Inspired by deformable convolution, the deformable attention module only cares about a small set of key sampling points around the reference point, regardless of the global size of feature maps. Convergence issues and high computing complexity caused by high feature map resolution can be alleviated by using deformable attention module by assigning a small, fixed number of keys to each query.

$$MutiHeadAttn(Z_q, x) = \sum_{m=1}^{M} W_m \left[\sum_{k \in \Omega_k} A_{mqk} * W'_m x_k \right] \quad (6)$$

The equation above is the original Multi-head attention used in DETR model [6], where $\boldsymbol{Z_q}$ is the input feature of $\boldsymbol{q^{th}}$ query, $\mathbf{M}$ is the set of Multi-head attention, $\boldsymbol{W_m}$ is the output projection matrix at $\boldsymbol{m^{th}}$ head, $\boldsymbol{\Omega_k}$ is the set of query, $\boldsymbol{A_{mqk}}$ is the attention weight of $\boldsymbol{q^{th}}$ query to $\boldsymbol{k^{th}}$ key at $\boldsymbol{m^{th}}$ head, and $\boldsymbol{W'_m}$ is the input value projection matrix at $\boldsymbol{m^{th}}$ head. From the equation 6, we can find that, because the DETR doesn't sample keys from the input sequence, $\boldsymbol{\Omega_k = H * W}$

where $\boldsymbol{H}$ *and* $\boldsymbol{W}$ is the height and width of the input image, causing extremely high compute complexity.

The problem of applying Transformer attention on image feature maps is that it traverses all possible keys in all the locations. To address this issue, paper propose a deformable attention module. Inspired by deformable convolution, the deformable attention module only focuses on a small set of key sampling points around the reference point, regardless of the spatial size of the feature map, as shown in Figure 13.

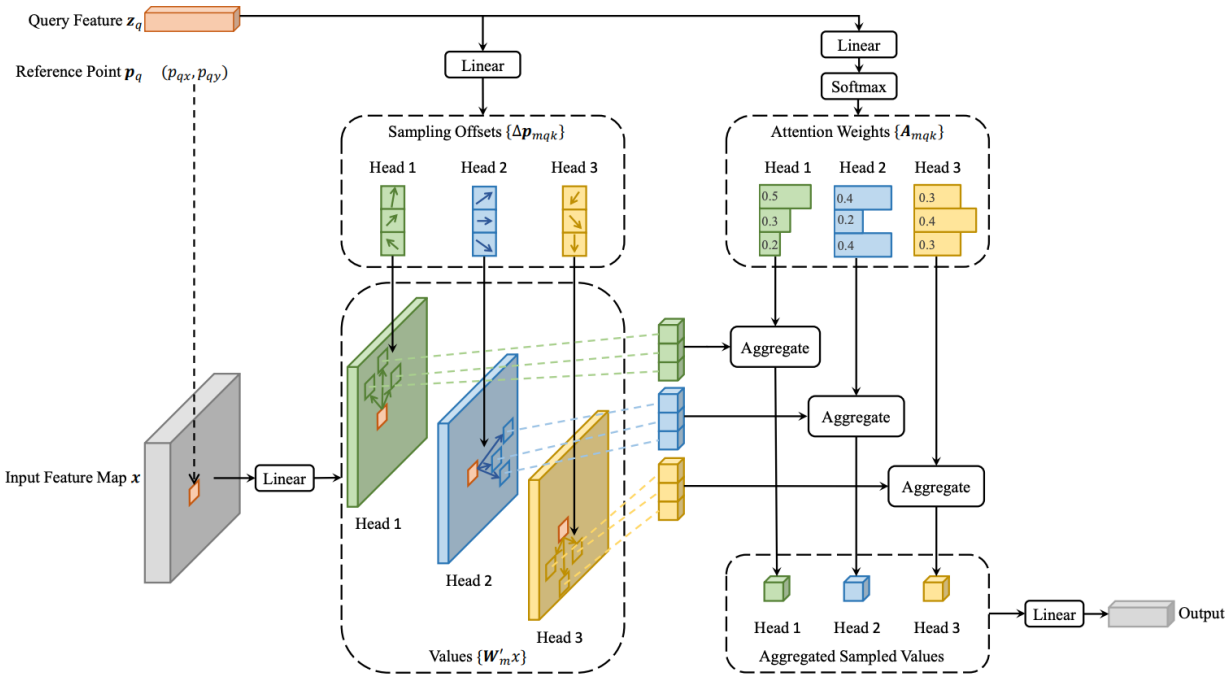

Figure 13: Deformable attention module explanation [5]

$$DeformAttn(Z_q,\ p_q\ ,x) = \sum_{m=1}^{M} W_m \left[\sum_{k=1}^{K} A_{mqk} * W'_m x_k(p_q + \Delta_{p_{mqk}})\right] \tag{7}$$

This is the equation of deformable attention in the paper. [5] The difference from multi-head attention is that:

- Self-attention uses all features as the set of key value $\boldsymbol{\Omega_k}$, deformable attention module is to learn to sample **K** key values ($\mathbf{K} << \boldsymbol{\Omega_k}$) independently near each reference point
- $\boldsymbol{p_q}$ represents 2-d coordinate of reference point for $\boldsymbol{q^{th}}$ query
- $\Delta_{\boldsymbol{p_{mqk}}}$ represents sampling offset of $\boldsymbol{q^{th}}$ query to $\boldsymbol{k^{th}}$ key at $\boldsymbol{m^{th}}$ head
- Because $\boldsymbol{x_k}\left(\boldsymbol{p_q} + \Delta_{\boldsymbol{p_{mqk}}}\right)$is fractional, bilinear interpolation is applied.
- According to figure 13, the query feature $\boldsymbol{Z_q}$ is fed to a linear projection operator of **3MK** channels, where the first **2MK** channels encode the sampling offsets $\Delta_{\boldsymbol{p_{mqk}}}$, and the

remaining **MK** channels are fed to a softmax operator to obtain the attention weights $\boldsymbol{A_{mqk}}$. [5]

2.2 Multi-scale Deformable Attention Module

Most existing object detection frameworks benefit from multi-scale feature maps. The deformable attention module designed can be naturally extended to multi-scale feature maps according to figure 14

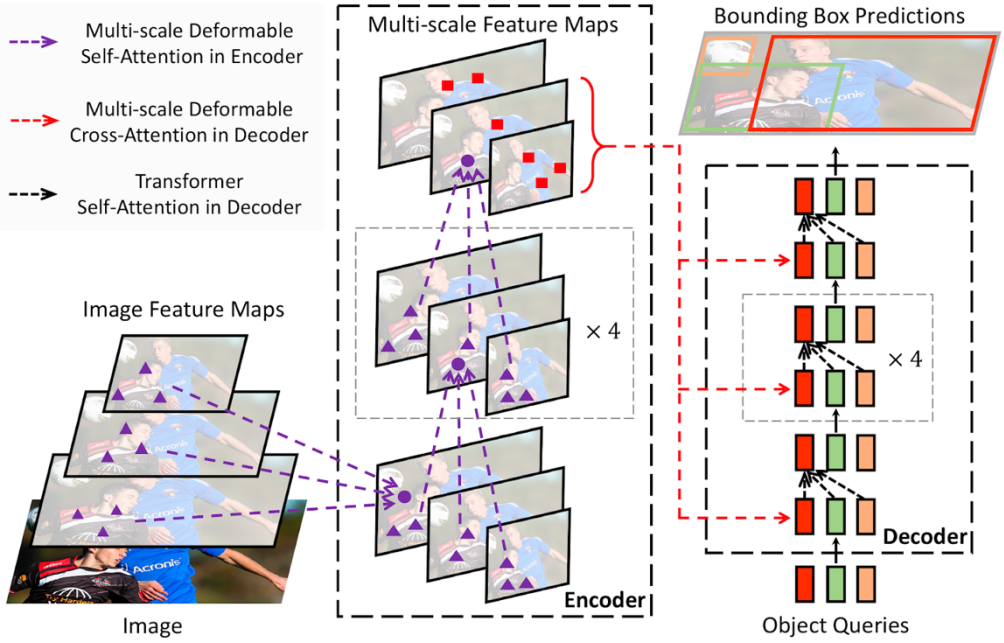

Figure 14: Multi-scale Deformable Attention Module [5]

When expanding to multi-scale features map, the process is the same as deformable attention, except that each head needs to process sampling points from four feature layers L. The equation 8 is the equation of deformable attention under multi-scale conditions. [5]

$$MSDeformAttn\left(Z_q, \hat{p}_q, \{x^l\}_{l=1}^{L}\right) = \sum_{m=1}^{M} W_m \left[\sum_{l=1}^{L}\sum_{k=1}^{K} A_{mlqk} * W'_m x^l\left(\emptyset_l(\hat{p}_q) + \Delta_{p_{mlqk}}\right)\right] \quad (8)$$

figure 15 shows the relationship between Transformer Attention, Deformable Convolution, Deformable Attention Module, Multi-scale Deformable Attention Module.

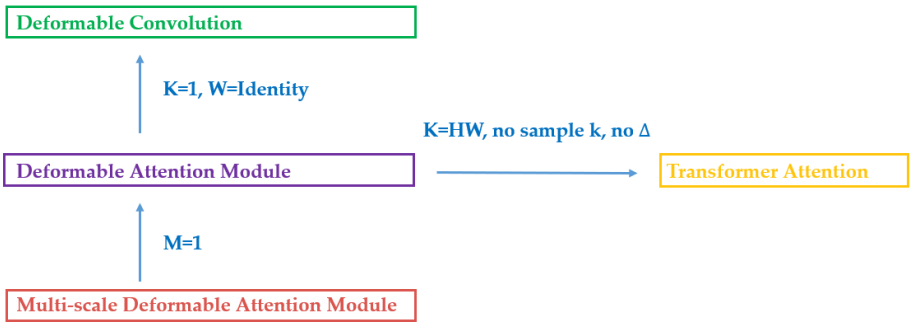

Figure 15: Relationship between Transformer Attention, Deformable Convolution, Deformable Attention Module, Multi-scale Deformable Attention Module. [7]

2.3 Encoder

The paper replaces the Transformer attention module which processing feature maps in DETR with the proposed multi-scale deformable attention module. Both the input and output of the encoder are multi-scale feature maps with the same resolution. In the encoder, shown in figure 16, multi-scale feature maps $\{x^l\}_{l=1}^{L}$ $(L = 4)$ are extracted through the output feature maps from stage C3 to stage C5 in ResNet (transformed by a 1 × 1 convolution) [5]

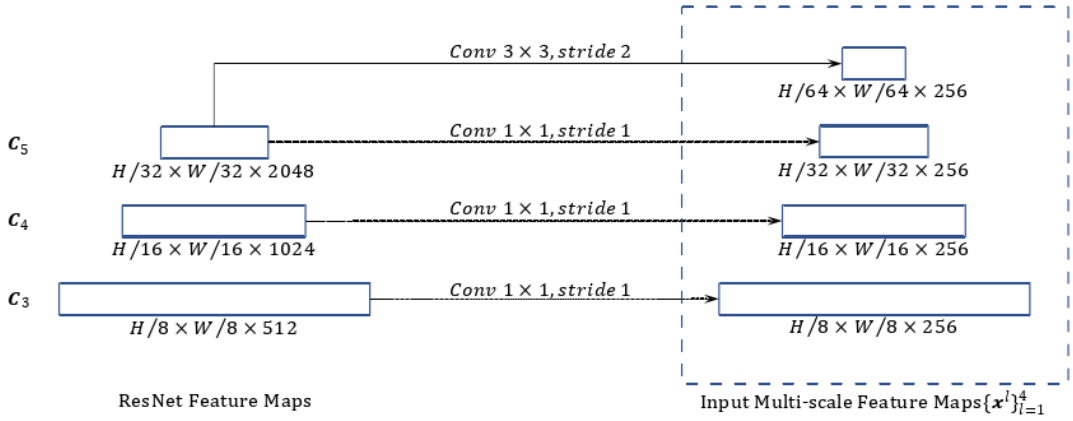

Figure 16: Multi-scale feature maps extraction in Encoder

2.4 Decoder

In decoder, there are two kinds of attention modules: cross-attention and self-attention modules, and the query elements of both types of attention modules are object queries.

In the cross-attention module, the object query extracts feature from the feature maps, where the key element is the feature maps output by the encoder. In the self-attention module, the object query and keys are from previous decoder outputs. Since the proposed deformable attention module is designed to handle convolutional feature maps as key elements, we only replace each

cross-attention module with a multi-scale deformable attention module, while the self-attention module remains unchanged. [5] For each query, the 2D normalized coordinates of the reference point are predicted from its object query embedding via a learnable linear projection followed by a sigmoid function.

3. Training and Results

3. 1 Training data and batch processing

ImageNet pre-trained ResNet-50 is utilized as the backbone for ablations. Multi-scale feature maps are extracted without FPN. M = 8 and K = 4 are set for deformable attentions by default. Parameters of the deformable Transformer encoder are shared among different feature levels. Other hyper-parameter setting, and training strategy mainly follow DETR (Carion et al., 2020), except that Focal Loss with loss weight of 2 is used for bounding box classification, and the number of object queries is increased from 100 to 300. By default, models are trained for 50 epochs and the learning rate is decayed at the 40-th epoch by a factor of 0.1. The paper train models using Adam optimizer with base learning rate of $2 * 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay of 10^{-4}. Learning rates of the linear projections, used for predicting object query reference points and sampling offsets, are multiplied by a factor of 0.1. Run time is evaluated on NVIDIA Tesla V100 GPU. [5]

3.2 Results

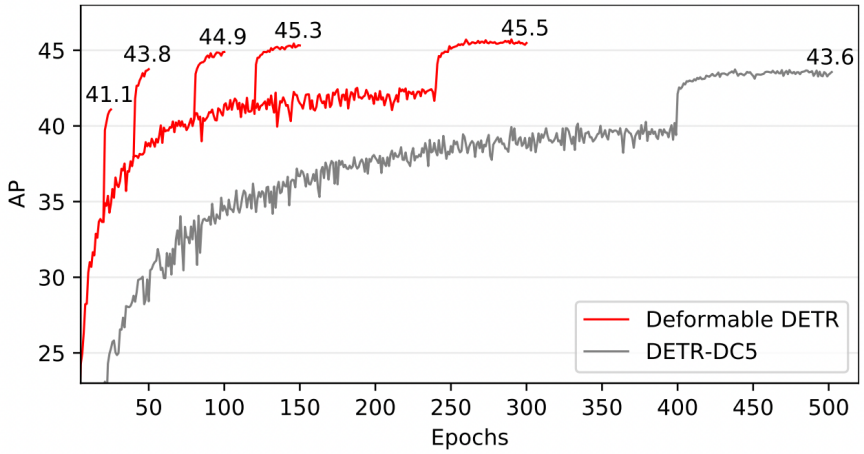

Figure 17: Convergence curves of Deformable DETR and DETR-DC5 on COCO 2017 val set [5]

Compared with Faster R-CNN+FPN, DETR requires more training epochs to converge and has lower performance in detecting small objects. Compared to DETR, Deformable DETR achieves better performance (especially on small objects) with 10 × fewer training epochs. The detailed convergence curve is shown in figure 17. The Deformable DETR converge to the optimal with around 250 epoch, while DETR need 400 epoch to converge to the optimal. [5]

Method	Epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	Training GPU hours	Inference FPS
Faster R-CNN + FPN	109	42.0	62.1	45.5	26.6	45.4	53.4	42M	180G	380	26
DETR	500	42.0	62.4	44.2	20.5	45.8	61.1	41M	86G	2000	28
DETR-DC5	500	43.3	63.1	45.9	22.5	47.3	61.1	41M	187G	7000	12
DETR-DC5	50	35.3	55.7	36.8	15.2	37.5	53.6	41M	187G	700	12
DETR-DC5$^+$	50	36.2	57.0	37.4	16.3	39.2	53.9	41M	187G	700	12
Deformable DETR	50	43.8	62.6	47.7	26.4	47.1	58.0	40M	173G	325	19
+ iterative bounding box refinement	50	45.4	64.7	49.0	26.8	48.3	61.7	40M	173G	325	19
++ two-stage Deformable DETR	50	46.2	65.2	50.0	28.8	49.2	61.7	40M	173G	340	19

Figure 18: Comparision of Deformable DETR with DETR on COCO 2017 val set [5]

For the overall performance, The Deformable DETR converge to the optimal with around 50 epochs and 325 Training GPU hours, while DETR need 500 epochs with 2000 Training GPU hours to converge to the optimal. Moreover, Deformable DETR has a similar FLOPs score with Faster R-CNN and DETR. But the runtime speed is much faster (1.6×) than DETR, and is just 25% slower than Faster R-CNN The speed issue of deformable det and DETR is mainly due to the large amount of memory access in Transformer attention. [5]

Reference

[1] https://github.com/tensorflow/tensor2tensor

[2] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

[3] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[4] Alammar, J. (n.d.). *The illustrated transformer*. The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. Retrieved October 27, 2022, from http://jalammar.github.io/illustrated-transformer/

[5] Zhu, Xizhou, et al. "Deformable detr: Deformable transformers for end-to-end object detection." *arXiv preprint arXiv:2010.04159* (2020).

[6] Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

[7] *彼岸的客人*. 开发者的网上家园. (n.d.). Retrieved October 27, 2022, from https://www.cnblogs.com/biandekeren-blog/p/16085350.html

[8] Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

[9] Chen, Mark, et al. "Generative pretraining from pixels." *International conference on machine learning*. PMLR, 2020.

[10] Liu, Ruijin, et al. "End-to-end lane shape prediction with transformers." *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021.