Chapter 6

Transformers and Visual Transformers

Robin Courant, Maika Edberg, Nicolas Dufour, and Vicky Kalogeiton

Abstract

Transformers were initially introduced for natural language processing (NLP) tasks, but fast they were adopted by most deep learning fields, including computer vision. They measure the relationships between pairs of input tokens (words in the case of text strings, parts of images for visual transformers), termed attention. The cost is exponential with the number of tokens. For image classification, the most common transformer architecture uses only the transformer encoder in order to transform the various input tokens. However, there are also numerous other applications in which the decoder part of the traditional transformer architecture is also used. Here, we first introduce the attention mechanism (Subheading 1) and then the basic transformer block including the vision transformer (Subheading 2). Next, we discuss some improvements of visual transformers to account for small datasets or less computation (Subheading 3). Finally, we introduce visual transformers applied to tasks other than image classification, such as detection, segmentation, generation, and training without labels (Subheading 4) and other domains, such as video or multimodality using text or audio data (Subheading 5).

Key words Attention, Transformers, Visual transformers, Multimodal attention

1 Attention

Attention is a technique in Computer Science that imitates the way in which the brain can focus on the relevant parts of the input. In this section, we introduce attention: its history (Subheading 1.1), its definition (Subheading 1.2), its types and variations (Subheadings 1.3 and 1.4), and its properties (Subheading 1.5).

To understand what attention is and why it is so useful, consider the following film review:

> While others claim the story is boring, I found it fascinating.

Is this film review positive or negative? The first part of the sentence is unrelated to the critic's opinion, while the second part suggests a positive sentiment with the word 'fascinating'. To a human, the answer is obvious; however, this type of analysis is not necessarily obvious to a computer.

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_6,

Typically, sequential data require *context* to be understood. In natural language, a word has a meaning because of its position in the sentence, with respect to the other words: its *context*. In our example, while "boring" alone suggests that the review is negative, its contextual relationship with other words allows the reader to reach the appropriate conclusion. In computer vision, in a task like object detection, the nature of a pixel alone cannot be identified: we need to account for its neighborhood, its *context*. So, how can we formalize the concept of *context* in sequential data?

1.1 The History of Attention

This notion of *context* is the motivation behind the introduction of the attention mechanism in 2015 [1]. Before this, language translation was mostly relying on encoder-decoder architectures: recurrent neural networks (RNNs) [2] and in particular long-short-term memory (LSTMs) networks were used to model the relationship among words [3]. Specifically, each word of an input sentence is processed by the encoder sequentially. At each step, the past and present information are summarized and encoded into a fixed-length vector. In the end, the encoder has processed every word and outputs a final fixed-length vector, which summarizes all input information. This final vector is then decoded and finally translates the input information into the target language.

However, the main issue of such structure is that all the information is compressed into one fixed-length vector. Given that the sizes of sentences vary and as the sentences get longer, a fixed-length vector is a real bottleneck: it gets increasingly difficult not to lose any information in the encoding process due to the vanishing gradient problem [1].

As a solution to this issue, Bahdanue et al. [1] proposed the attention module in 2015. The attention module allows the model to consider the parts of the sentence that are relevant to predicting the next word. Moreover, this facilitates the understanding of relationships among words that are further apart.

1.2 Definition of Attention

Given two lists of tokens, $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{N \times d_y}$, attention encodes information from $\boldsymbol{\Upsilon}$ into $\boldsymbol{X}$, where N is the length of inputs $\boldsymbol{X}$ and $\boldsymbol{\Upsilon}$ and d_x and d_y are their respective dimensions. For this, we first define three linear mappings, query mapping $\boldsymbol{W}^Q \in \mathbb{R}^{d_x \times d_q}$, key mapping $\boldsymbol{W}^K \in \mathbb{R}^{d_y \times d_k}$, and value mapping $\boldsymbol{W}^V \in \mathbb{R}^{d_y \times d_v}$, where d_q, d_k, and d_v are the embedding dimensions in which the query, key, and value are going to be computed, respectively.

Then, we define the query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$ [4] as:

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{X}\boldsymbol{W}^Q \\ \boldsymbol{K} &= \boldsymbol{\Upsilon}\boldsymbol{W}^K \\ \boldsymbol{V} &= \boldsymbol{\Upsilon}\boldsymbol{W}^V \end{aligned}$$

Next, the *attention matrix* is defined as:

$$A(\boldsymbol{Q}, \boldsymbol{K}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{d_k}}\right). \tag{1}$$

This is illustrated in the left part of Fig. 1. The nominator $\boldsymbol{Q}\boldsymbol{K}^T \in \mathbb{R}^{N \times N}$ represents how each part of the input in $\boldsymbol{X}$ attends to each part of the input in $\boldsymbol{\Upsilon}$.[1] This dot product is then put through the softmax function to normalize its values and get positive values that add to 1. However, for large values of d_k, this may result in the softmax to have incredibly small gradients, so it is scaled down by $\sqrt{d_k}$.

The resulting $N \times N$ matrix encodes the relationship between $\boldsymbol{X}$ with respect to $\boldsymbol{\Upsilon}$: it measures how important a token in $\boldsymbol{X}$ is with respect to another one in $\boldsymbol{\Upsilon}$.

Finally, the *attention output* is defined as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = A(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V}. \tag{2}$$

Figure 1 displays this. The attention output encodes the information of each token by taking into account the contextual information. Therefore, through the learnable parameters—queries, keys, and values—the attention layers learn a token embedding that takes into account their relationship.

Contextual Relationships How does Eq. 2 encode contextual relationships? To answer this question, let us reconsider analyzing the sentiment of film reviews. To encode contextual relationships into the word embedding, we first want a matrix representation of the relationship between all words. To do so, given a sentence of length N, we take each word vector and feed it to two different linear layers, calling one output "query" and the other output "key". We pack the queries into the matrix $\boldsymbol{Q}$ and the keys into the matrix $\boldsymbol{K}$, by taking their product ($\boldsymbol{Q}\boldsymbol{K}^T$). The result is a $N \times N$ matrix that explains how important the i-th word (row-wise) is to understand the j-th word (column-wise). This matrix is then scaled and normalized by the division and softmax. Next, we feed the word vectors into another linear layer, calling its output "value". We multiply these two matrices together. The results of their product are attention vectors that encode the meaning of each word, by including their contextual meaning as well. Given that each of these queries, keys, and values is learnable parameter, as the attention layer is trained, the model learns how relationships among words are encoded in the data.

[1] Note that in the literature, there are two main attention functions: additive attention [1] and dot-product attention (Eq. 1). In practice, the dot product is more efficient since it is implemented using highly optimized matrix multiplication, compared to the feed-forward network of the additive attention; hence, the dot product is the dominant one.

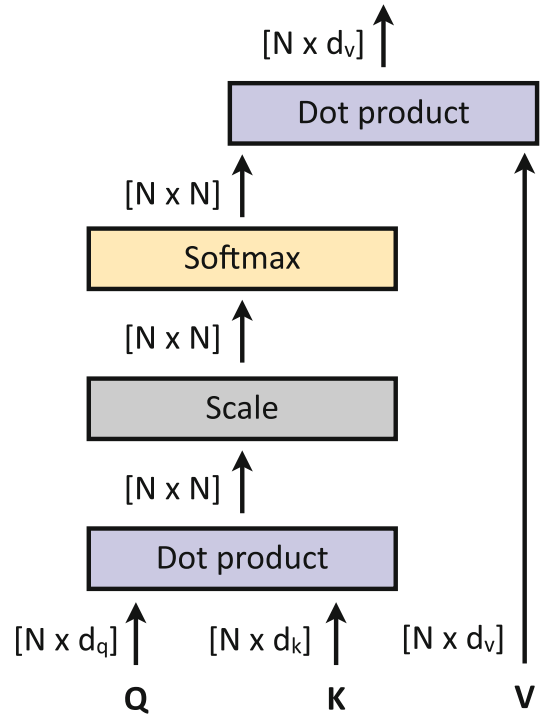

Fig. 1 Attention block. Next to each element, we denote its dimensionality. Figure inspired from [4]

1.3 Types of Attention

There exist two dominant types of attention mechanisms: *self-attention* and *cross attention* [4]. In *self-attention*, the queries, keys, and values come from the same input, i.e., $\boldsymbol{X} = \boldsymbol{\Upsilon}$; in *cross attention*, the queries come from a different input than the key and value vectors, i.e., $\boldsymbol{X} \neq \boldsymbol{\Upsilon}$. These are described below in Subheadings 1.3.1 and 1.3.2, respectively.

1.3.1 Self-Attention

In self-attention, the tokens of $\boldsymbol{X}$ attend to themselves ($\boldsymbol{X} = \boldsymbol{\Upsilon}$). Therefore, it is modeled as follows:

$$\mathrm{SA}(\boldsymbol{X}) = \mathrm{Attention}(\boldsymbol{X}\boldsymbol{W}^{Q}, \boldsymbol{X}\boldsymbol{W}^{K}, \boldsymbol{X}\boldsymbol{W}^{V}). \tag{3}$$

Self-attention formalizes the concept of context. It learns the patterns underlying how parts of the input correspond to each other. By gathering information from the same set, given a sequence of tokens, a token can attend to its neighboring tokens to compute its output.

1.3.2 Cross Attention

Most real-world data are multimodal—for instance, videos contain frames, audios, and subtitles, images come with captions, etc. Therefore, models that can deal with such types of multimodal information have become essential.

Cross attention is an attention mechanism designed to handle multimodal inputs. Unlike self-attention, it extracts queries from one input source and key-value pairs from another one ($\boldsymbol{X} \neq \boldsymbol{\Upsilon}$). It answers the following question: "Which parts of input $\boldsymbol{X}$ and input $\boldsymbol{\Upsilon}$ correspond to each other?" Cross attention (CA) is defined as:

$$\mathrm{CA}(\boldsymbol{X}, \boldsymbol{\Upsilon}) = \mathrm{Attention}(\boldsymbol{X}\boldsymbol{W}^{Q}, \boldsymbol{\Upsilon}\boldsymbol{W}^{K}, \boldsymbol{\Upsilon}\boldsymbol{W}^{V}). \tag{4}$$

1.4 Variation of Attention

Attention is typically employed in two ways: (1) multi-head self-attention (MSA, Subheading 1.4) and (2) masked multi-head attention (MMA, Subheading 1.4).

Attention Head We call attention head the mechanism presented in Subheading 1.2, i.e., query-key-value projection, followed by scaled dot product attention (Eqs. 1 and 2).

When employing an attention-based model, relying only on a single attention head can inhibit learning. Therefore, the multi-head attention block is introduced [4].

Multi-head Self-Attention (MSA) MSA is shown in Fig. 2 and is defined as:

$$\begin{aligned} \mathrm{MSA}(\boldsymbol{X}) &= \mathrm{Concat}\big(\mathrm{head}_1(\boldsymbol{X}), \ldots, \mathrm{head}_h(\boldsymbol{X})\big)\boldsymbol{W}^{O}, \\ \mathrm{head}_i(\boldsymbol{X}) &= \mathrm{SA}(\boldsymbol{X}) \ , \forall i \in \{1, h\}, \end{aligned} \tag{5}$$

where Concat is the concatenation of h attention heads and $\boldsymbol{W}^{O} \in \mathbb{R}^{hd_v \times d}$ is projection matrix. This means that the initial embedding dimension d_x is decomposed into $h \times d_v$ and the computation per head is carried out independently. The independent attention heads are usually concatenated and multiplied by a linear layer to match the desired output dimension. The output dimension is often the same as the input embedding dimension d. This allows an easier stacking of multiple blocks.

Multi-head Cross Attention (MCA) Similar to MSA, MCA is defined as:

$$\begin{aligned} \mathrm{MCA}(\boldsymbol{X}, \boldsymbol{\Upsilon}) &= \mathrm{Concat}(\mathrm{head}_1(\boldsymbol{X}, \boldsymbol{\Upsilon}), \ldots, \mathrm{head}_h(\boldsymbol{X}, \boldsymbol{\Upsilon}))\boldsymbol{W}^{O}, \\ \mathrm{head}_i(\boldsymbol{X}, \boldsymbol{\Upsilon}) &= \mathrm{CA}(\boldsymbol{X}, \boldsymbol{\Upsilon}) \ , \forall i \in \{1, h\}. \end{aligned} \tag{6}$$

Masked Multi-head Self-Attention (MMSA) The MMSA layer [4] is another variation of attention. It has the same structure as the multi-head self-attention block (Subheading 1.4), but all the later vectors in the target output are masked. When dealing with sequential data, this can help make training parallel.

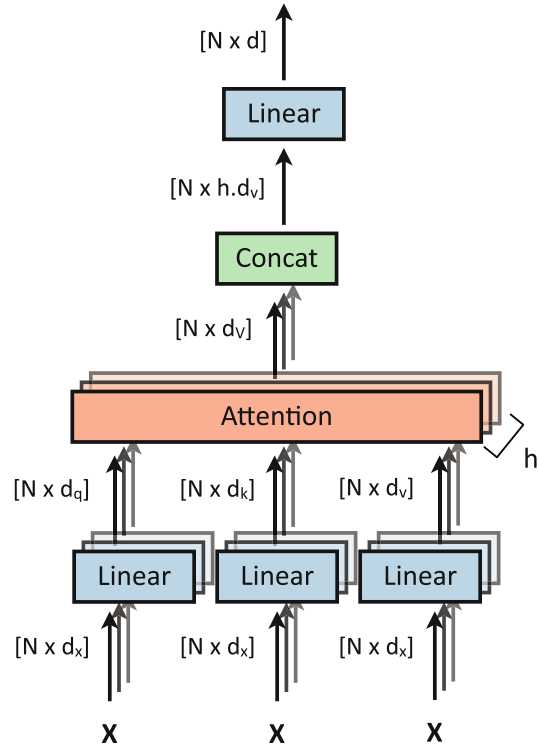

Fig. 2 Multi-head self-attention block (MSA). First, the input $\boldsymbol{X}$ is projected to queries, keys, and values and then passed through h attention blocks. The h resulting attention outputs are then concatenated together and finally projected to a d-dimensional output vector. Next to each element, we denote its dimensionality. Figure inspired from [4]

1.5 Properties of Attention

While attention encodes contextual relationships, it is *permutation equivalent*, as the mechanism does not account for the order of the input data. As shown in Eq. 2, the attention computations are all matrix multiplication and normalizations. Therefore, a permuted input results in a permuted output. In practice, however, this may not be an accurate representation of the information. For instance, consider the sentences "the monkey ate the banana" and "the banana ate the monkey." They have distinct meanings because of the order of the words. If the order of the input is important, various mechanisms, such as the positional encoding, discussed in Subheading 2.1.2, are used to capture this subtlety.

2 Visual Transformers

The transformer architecture was introduced in [4] and is the first architecture that relies purely on attention to draw connections between the inputs and outputs. Since its debut, it revolutionized deep learning, making breakthroughs in numerous fields, including

natural language processing, computer vision, chemistry, and biology, thus making its way to becoming the *default* architecture for learning representations. Recently, the standard transformer [4] has been adapted for vision tasks [5]. And again, visual transformer has become one of the central architectures in computer vision.

In this section, we first introduce the basic architecture of transformers (Subheading 2.1) and then present its advantages (Subheading 2.2). Finally, we describe the vision transformer (Subheading 2.3).

2.1 Basic Transformers

As shown in Fig. 3, the transformer architecture [4] is an encoder-decoder model. First, it embeds input tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ into a latent space, resulting in latent vectors $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N)$, which are fed to the decoder to output $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M)$. The encoder is a stack of L layers, with each one consisting of two sub-blocks: multi-head self-attention (MSA) layers and a multilayer perceptron (MLP). The decoder is also a stack of L layers, with each one consisting of three sub-blocks: masked multi-head self-attention (MMSA), multi-head cross attention (MCA), and MLP.

Overview Below, we describe the various parts of the transformer architecture, following Fig. 3. First, the input tokens are converted into the embedding tokens (Subheading 2.1.1). Then, the positional encoding adds a positional token to each embedding token to denote the order of tokens (Subheading 2.1.2). Then, the transformer encoder follows (Subheading 2.1.3). This consists of a stack of L multi-head attention, normalization, and MLP layers and encodes the input to a set of semantically meaningful features. After, the decoder follows (Subheading 2.1.4). This consists of a stack of L masked multi-head attention, multi-head attention, and MLP layers followed by normalizations and decodes the input features with respect to the output embedding tokens. Finally, the output is projected to linear and softmax layers.

2.1.1 Embedding

The first step of transformers consists in converting input tokens[2] into embedding tokens, i.e., vectors with meaningful features. To do so, following standard practice [6], each input is projected into an embedding space to obtain embedding tokens $\boldsymbol{Z}^e$. The embedding space is structured in a way that the distance between a pair of vectors is relative to the semantic similarity of their associated words. For the initial NLP case, this means that we get a vector of each word, such that the vectors that are closer together have similar meanings.

[2] Note the initial transformer architecture was proposed for natural language processing (NLP), and therefore the inputs were words.

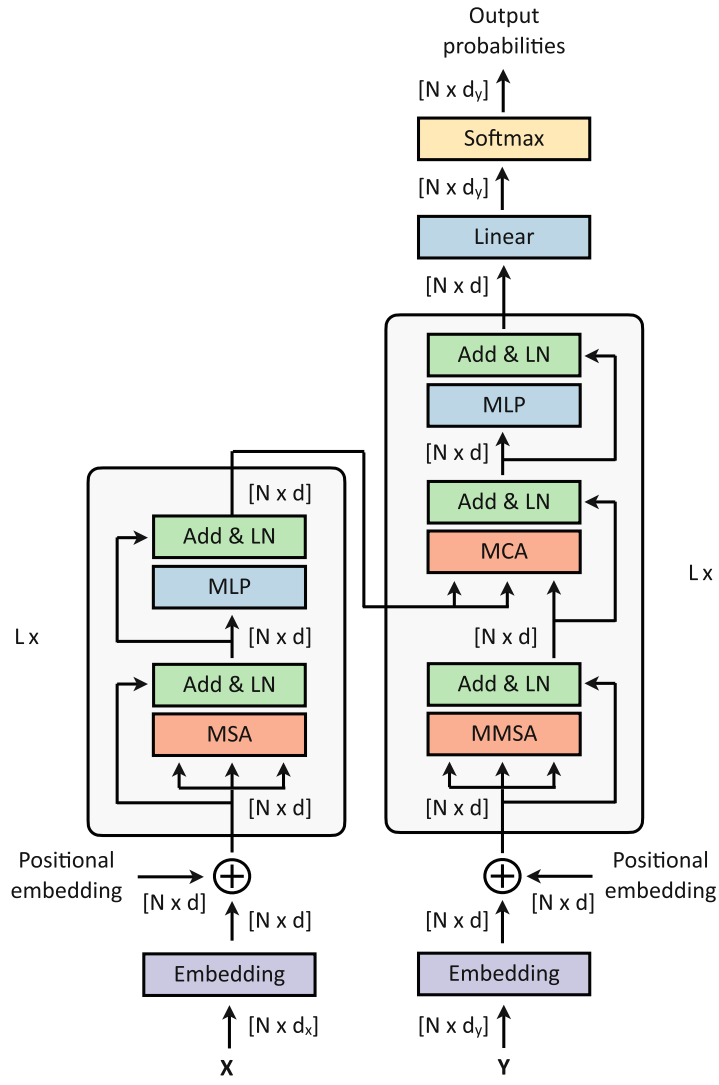

Fig. 3 The transformer architecture. It consists of an encoder (left) and a decoder (right) block, each one consisting from a series of attention blocks (multi-head and masked multi-head attention) and MLP layers. Next to each element, we denote its dimensionality. Figure inspired from [4]

2.1.2 *Positional Encoding*

As discussed in Subheading 1.5, the attention mechanism is positional agnostic, which means that it does not store the information on the position of each input. However, in most cases, the order of input tokens is relevant and should be taken into account, such as the order of words in a sentence matter as they may change its meaning. Therefore, [4] introduced the ***Positional Encoding*** $\mathbf{PE} \in \mathbb{R}^{N \times d_x}$, which adds a positional token to each embedding token $\boldsymbol{Z}^{\mathrm{e}} \in \mathbb{R}^{N \times d_x}$.

Sinusoidal Positional Encoding

The sinusoidal positional encoding [4] is the main positional encoding method, which encodes the position of each token with sinusoidal waves of multiple frequency. For an embedding token $\boldsymbol{Z}^{\mathrm{e}} \in \mathbb{R}^{N \times d_x}$, its positional encoding $\mathbf{PE} \in \mathbb{R}^{N \times d_x}$ is defined as:

$$\begin{aligned} \mathbf{PE}(i, 2j) &= \sin\left(\frac{i}{10000^{2j/d}}\right) \\ \mathbf{PE}(i, 2j+1) &= \cos\left(\frac{i}{10000^{2j/d}}\right), \forall i, j \in [|1, n|] \times [|1, d|]. \end{aligned} \tag{7}$$

Learnable Positional Encoding

An orthogonal approach is to let the model learn the positional encoding. In this case, $\mathbf{PE} \in \mathbb{R}^{N \times d_x}$ becomes a learnable parameter. This, however, increases the memory requirements, without necessarily bringing improvements over the sinusoidal encoding.

Positional Embedding

After its computation, either the positional encoding **PE** is added to the embedding tokens or they are concatenated as follows:

$$\begin{aligned} \boldsymbol{Z}^{\mathrm{pe}} &= \boldsymbol{Z}^{e} + \mathbf{PE}, \text{ or} \\ \boldsymbol{Z}^{\mathrm{pe}} &= \mathrm{Concat}(\boldsymbol{Z}^{e}, \mathbf{PE}), \end{aligned} \tag{8}$$

where Concat denotes vector concatenation. Note that the concatenation has the advantage of not altering the information contained in $\boldsymbol{Z}^{e}$, since the positional information is only added to the unused dimension. Nevertheless, it augments the input dimension, leading to higher memory requirements. Instead, the addition does preserve the same input dimension while altering the content of the embedding tokens. When the input dimension is high, this content altering is trivial, as most of the content is preserved. Therefore, in practice, for high dimension, summing positional encodings is preferred, whereas for low dimensions concatenating them prevails.

2.1.3 Encoder Block

The encoder block takes as input the embedding and positional tokens and outputs features of the input, to be decoded by the decoder block. It consists of a stack of L multi-head self-attention (MSA) layers and a multilayer perceptron (MLP). Specifically, the embedding and positional tokens, $\boldsymbol{Z}_x^{\mathrm{pe}} \in \mathbb{R}^{N \times d}$, go through a multi-head self-attention block. Then, a residual connection with layer normalization is deployed. In the transformer, this operation is performed after each sub-layer. Next, we feed its output to an MLP and a normalization layer. This operation is performed L times, and each time the output of each encoder block (of size $N \times d$) is the input of the subsequent block. In the $L-$th time, the output of the normalization is the input of the cross-attention block in the decoder (Subheading 2.1.4).

2.1.4 Decoder Block

The decoder has two inputs: first, an input that constitutes the queries $\boldsymbol{Q} \in \mathbb{R}^{N \times d}$ of the encoder, and, second, the output of the encoder that constitutes the key-value $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times d}$ pair. Similar to Subheadings 2.1.1 and 2.1.2, the first step constitutes encoding the output token to output embedding token and output positional token. These tokens are fed into the main part of the decoder, which consists of a stack of L masked multi-head self-attention (MMSA) layers, multi-head cross-attention (MCA) layers, and multilayer perceptron (MLP) followed by normalizations. Specifically, the embedding and positional tokens, $\boldsymbol{Z}_y^{\text{pe}} \in \mathbb{R}^{N \times d}$, go through a MMSA block. Then, a residual connection with layer normalization follows. Next, an MCA layer (followed by normalization) maps the queries to the encoded key values before forwarding the output to an MLP. Finally, we project the output of the L decoder blocks (of dimension $N \times d_y$) through a linear layer and get output probability through a softmax layer.

2.2 Advantages of Transformers

Since their introduction, the transformers have had a significant impact on deep learning approaches.

In natural language processing (NLP), before transformers, most architectures used to rely on recurrent modules, such as RNNs [2] and in particular LSTMs [3]. However, recurrent models process the input sequentially, meaning that, to compute the current state, they require the output of the previous state. This makes them tremendously inefficient, as they are impossible to parallelize. On the contrary, in transformers, each input is processed independent of the others, and the multi-head attention can perform multiple attention computations at once. This makes transformers highly efficient, as they are highly parallelizable.

This results in not only exceptional scalability, both in the complexity of the model and the size of datasets, but also relatively fast training. Notably, the recent switch transformers [7] was pretrained on 34 billion tokens from the C4 dataset [8], scaling the model to over 1 trillion parameters.

This scalability [7] is the principal reason for the power of the transformer. While it was originally introduced for translation, it refrains from introducing many inductive biases, i.e., the set of assumptions that the user makes about the structure of the model input. In doing so, the transformer relies on data to learn how they are structured. Compared to its counterparts with more biases, the transformer requires much more data to produce comparable results [5]. However, if a sufficient amount of data is available, the lack of inductive bias becomes a strength. By learning the structure of the data from the data, the transformer is able to learn better without human assumptions hindering [9].

In most tasks involving transformers, the model is first pretrained on a large dataset and then fine-tuned for the task at hand on a smaller dataset. The pretraining phase is essential for

transformers to learn the global structure of the specific input modality. For fine-tuning, typically fewer data suffice as the model is already rich. For instance, in natural language processing, BERT [10], a state-of-the-art language model, is pretrained on a Wikipedia-based dataset [11], with over 6 million articles and Book Corpus [12] with over 10,000 books. Then, this model can be fine-tuned on much more specific tasks. In computer vision, the vision transformer (ViT) is pretrained on the JFT-300M dataset, containing over 1 billion labels for 300 million images [5]. Hence, with a sufficient amount of data, transformers achieve results that were never possible before in various areas of machine learning.

2.3 Vision Transformer

Transformers offer an alternative to CNNs that have long held a stranglehold on computer vision. Before 2020, most attempts to use transformers for vision tasks were still highly reliant on CNNs, either by using self-attention jointly with convolutions [13, 14] or by keeping the general structure of CNNs while using self-attention [15, 16].

The reason for this is rooted in the two main weaknesses of the transformers. First, the complexity of the attention operation is high. As attention is a quadratic operation, the number of parameters skyrockets quickly when dealing with visual data, i.e., images—and even more so with videos. For instance, in the case of ImageNet [17], inputting a single image with $256 \times 256 = 65,536$ pixels in an attention layer would be too heavy computationally. Second, transformers suffer from lack of inductive biases. Since CNNs were specifically created for vision tasks, their architecture includes spatial inductive biases, like translation equivariance and locality. Therefore, the transformers have to be pretrained on a significantly large dataset to achieve similar performances.

The vision transformer (ViT) [5] is the first systematic approach that uses directly transformers for vision tasks by addressing both aforementioned issues. It rids the concept of convolutions altogether, using purely a transformer-based architecture. In doing so, it achieves the state of the art on image recognition on various datasets, including ImageNet [17] and CIFAR-100 [18].

Figure 4 illustrates the ViT architecture. The input image is first split into 16×16 patches, flattened, and mapped to the expected dimension through a learnable linear projection. Since the image size is reduced to 16×16, the complexity of the attention mechanism is no longer a bottleneck. Then, ViT encodes the positional information and attaches a learnable embedding to the front of the sequence, similarly to BERT's classification token [10]. The output of this token represents the entirety of the input—it encodes the information from each part of the input. Then, this sequence is fed into an encoder block, with the same structure as in the standard transformers [4]. The output of the classification token is then fed into an MLP that outputs class probabilities.

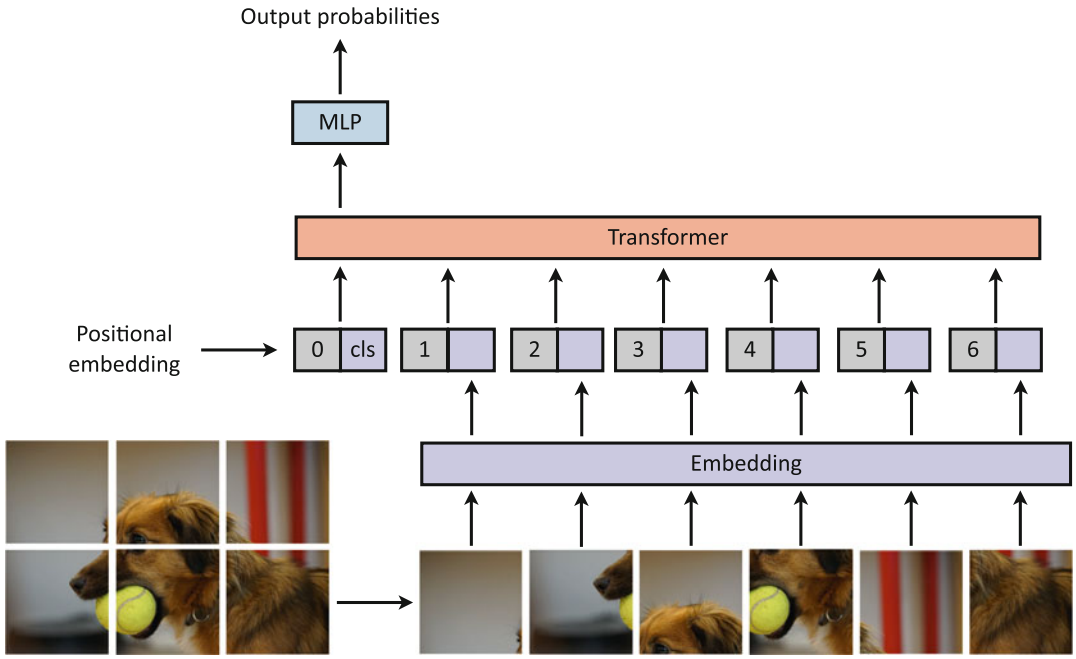

Fig. 4 The vision transformer architecture (ViT). First, the input image is split into patches (bottom), which are linearly projected (embedding), and then concatenated with positional embedding tokens. The resulting tokens are fed into a transformer, and finally the resulting classification token is passed through an MLP to compute output probabilities. Figure inspired from [5]

Due to the lack of inductive biases, when ViT is trained only on mid-sized datasets such as ImageNet, it scores some percentage points lower than the state of the art. Therefore, the proposed model is first pretrained on the JFT-300M dataset [19] and then fine-tuned on smaller datasets, thereby increasing its accuracy by 13%.

For a complete overview of visual transformers and follow-up works, we invite the readers to study [9, 20].

3 Improvements over the Vision Transformer

In this section, we present transformer-based methods that improve over the original vision transformer (Subheading 2.3) in two main ways. First, we introduce approaches that are trained on smaller datasets, unlike ViT [5] that requires pretraining on 300 million labeled images (Subheading 3.1). Second, we present extensions over ViT that are more computational-efficient than ViT, given that training a ViT is directly correlated to the image resolution and the number of patches (Subheading 3.2).

3.1 Data Efficiency

As discussed in Subheading 2.3, the vision transformer (ViT) [5] is pretrained on a massive proprietary dataset (JFT-300M) which contains 300 million labeled images. This need arises with transformers because we remove the inductive biases from the architecture compared to convolutional-based networks. Indeed, convolutions contain some translation equivariance. ViT does not benefit from this property and thus has to learn such biases, requiring more data. JFT-300M is an enormous dataset, and to make ViT work in practice, better data-efficiency is needed. Indeed, collecting that amount of data is costly and can be infeasible for most tasks.

Data-Efficient Image Transformers (DeiT) [21] The first work to achieve an improved data efficiency is DeiT [21] . The main idea of DeiT is to distil the inductive biases from a CNN into a transformer (Fig. 5). DeiT adds another token that works similarly to the class token. When training, ground truth labels are used to train the network according to the class token output with a cross-entropy (CE) loss. However, for the distillation network, the output labels are compared to the labels provided from a teacher network with a

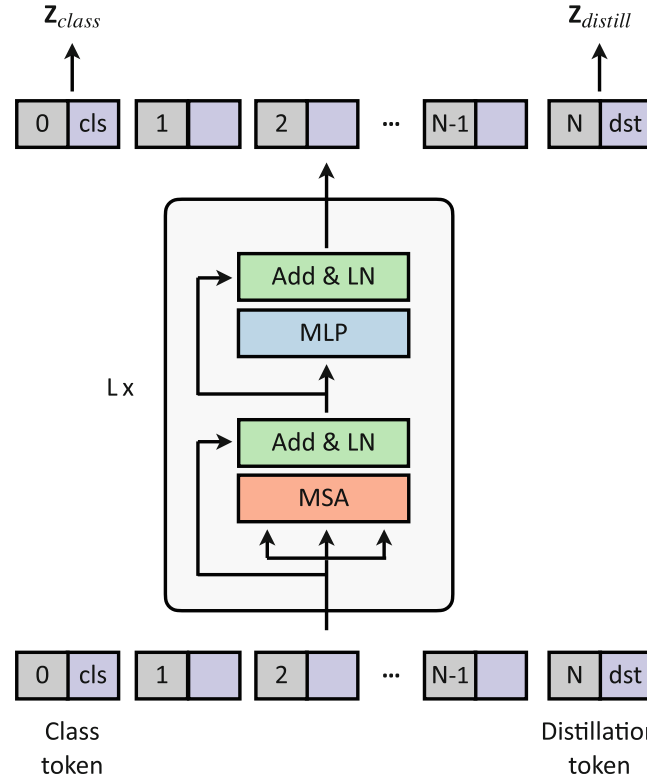

Fig. 5 The DeiT architecture. The architecture features an extra token, the distillation token. This token is used similarly to the class token. Figure inspired from [21]

cross-entropy loss. The final loss for a N-categorical classification task is defined as follows:

$$\begin{aligned} \mathcal{L}_{\text{global}}^{\text{hardDistill}} &= \frac{1}{2}\left(\mathcal{L}_{\text{CE}}(\Psi(\boldsymbol{Z}_{\text{class}}), \boldsymbol{y}) + \mathcal{L}_{\text{CE}}(\Psi(\boldsymbol{Z}_{\text{distill}}), \boldsymbol{y}_T)\right), \\ \mathcal{L}_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right] \end{aligned} \tag{9}$$

with Ψ the softmax function, $\boldsymbol{Z}_{\text{class}}$ the class token output, $\boldsymbol{Z}_{\text{distill}}$ the class token output, $\boldsymbol{y}$ the ground truth label, and $\boldsymbol{y}_T$ the teacher label prediction.

The teacher network is a Convolutional Neural Network (CNN). The main idea is that the distillation head will provide the inductive biases needed to improve the data efficiency of the architecture. By doing this, DeiT achieves remarkable performance on the ImageNet dataset, by training "only" on ImageNet-1K [17], which contains 1.3 million images.

Convit [22] The main disadvantage of DeiT [21] is that it requires a pretrained CNN, which is not ideal, and it would be more convenient to not have this requirement. The CNN has a hard inductive bias constraint that can be a major limitation. Indeed, if enough data is available, learning the biases from the data can result in better representations.

Convit [22] overpasses this issue by including the inductive bias of CNNs into a transformer in a soft way. Specifically, if the inductive bias is limiting the training, the transformer can discard it. The main idea is to include the inductive bias into the ViT initialization. Therefore, before beginning training, the ViT is equivalent to a CNN. Then, the network can progressively learn the needed biases and diverge from the CNN initialization.

Compact Convolutional Transformer [23], DeiT [21], and Convit [22] successfully achieve data efficiency at the ImageNet scale. However, ImageNet is a big dataset with 1.3 million images, whereas most datasets are significantly smaller.

To reach higher data efficiency, the compact convolutional transformer [23] uses a CNN operation to extract the patches and then uses these patches in a transformer network (Fig. 6). The compact convolutional transformer comes with some modifications that lead to major improvements. First, by having a more complex encoding of patches, the system relies on the convolutional inductive biases at the lower scales and then uses a transformer network to remove the locality constraint of the CNN. Second, the authors show that discarding the "class" token results in higher efficiency. Specifically, instead of the class token, the compact convolutional transformer pools together all the patches token and classifies on top of this pooled token. These two modifications enable using

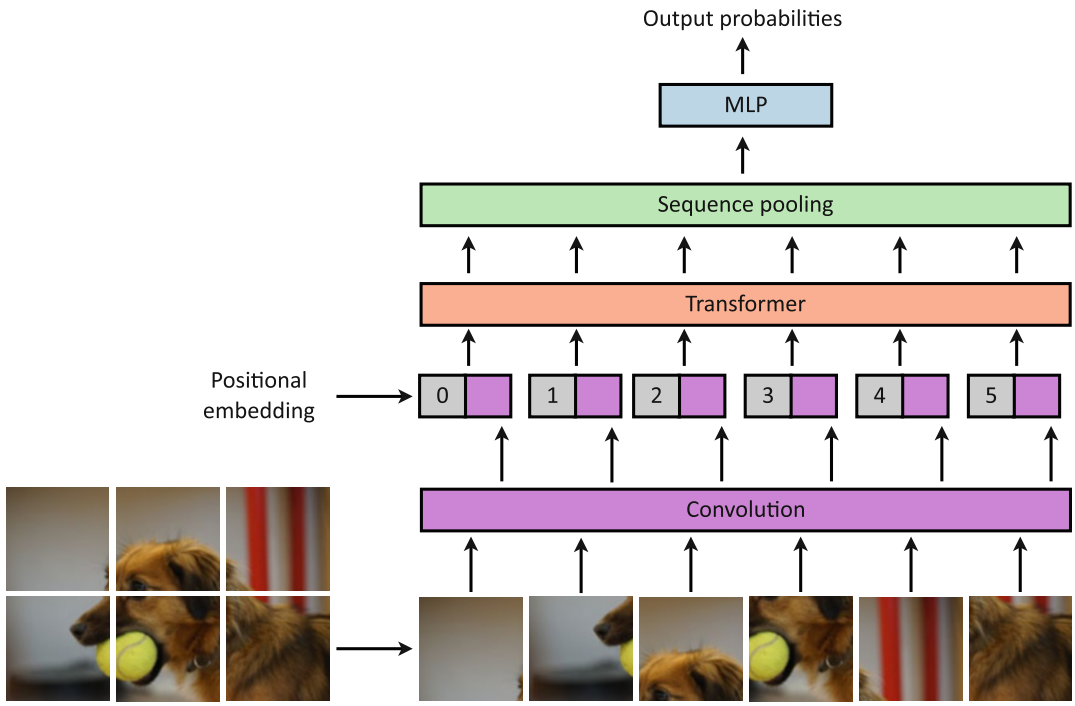

Fig. 6 Compact convolutional transformers. This architecture features a convolutional-based patch extraction to leverage a smaller transformer network, leading to higher data efficiency. Figure inspired from [23]

smaller transformers while improving both the data efficiency and the computational efficiency. Therefore, these improvements allow the compact convolutional transformer to be successfully trained on smaller datasets, such as CIFAR or MNIST.

3.2 Computational Efficiency

The vision transformer architecture (Subheading 2.3) suffers from a $\mathcal{O}(n^2)$ complexity with respect to the number of tokens. When considering small resolution images or big patch size, this is not a limitation; for instance, for an image of 224×224 resolution with 16×16 patches, this amounts to 196 tokens. However, when needing to process larger images (for instance, 3D images in medical imaging) or when considering smaller patches, using and training such models becomes *prohibitive*. For instance, in tasks such as segmentation or image generation, it is needed to have more granular representations than 16×16 patches; hence, it is crucial to solve this issue to enable more applications of vision transformer.

Swin Transformer [24] One idea to make transformers more computation-efficient is the Swin transformer [24]. Instead of attending every patch in the image, the Swin transformer proposes to add a locality constraint. Specifically, the patches can only attend other patches that are limited to a vicinity window K. This restores the local inductive bias of CNNs. To allow communication across

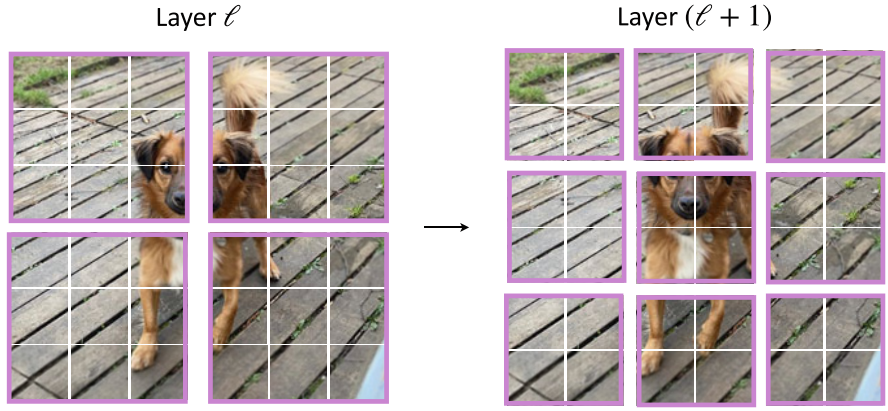

Fig. 7 Shifting operation in the Swin transformer [24]. Between each attention operation, the attention window is shifted so that each patch can communicate with different patches than before. This allows the network to gain more global knowledge with the network's depth. Figure inspired from [24]

patches throughout the network, the Swin transformer shifts the attention windows from one operation to another (Fig. 7). Therefore, the Swin transformer is quadratic with regard to the size of the window K but linear with respect to the number of tokens n with complexity $\mathcal{O}(nK^2)$. In practice, however, K is small, and this solves the quadratic complexity problem of attention.

Perceiver [25, 26] Another idea for more computation-efficient visual transformers is to make a more drastic change to the architecture. If instead of using self-attention the model uses cross attention, the problem of the quadratic complexity with regard to the number of tokens can be solved. Indeed, computing the cross attention between two sets of length m and n, respectively, has complexity $\mathcal{O}(mn)$. This idea is introduced in the perceiver [25, 26]. The key idea is to have a smaller set of latent variables that will be used as queries and that will retrieve information in the image token set (Fig. 8). Since this solves the quadratic complexity issue, it also removes the need of using patches; hence, in the case of transformers, each pixel is mapped to a single token.

4 Vision Transformers for Tasks Other than Classification

Subheadings 1–3 introduce visual transformers for one main application: classification. Nevertheless, transformers can be used for numerous other tasks than classification.

In this section, we present some fundamental vision tasks where transformers have had a major impact: object detection in images (Subheading 4.1), image segmentation (Subheading 4.2), training

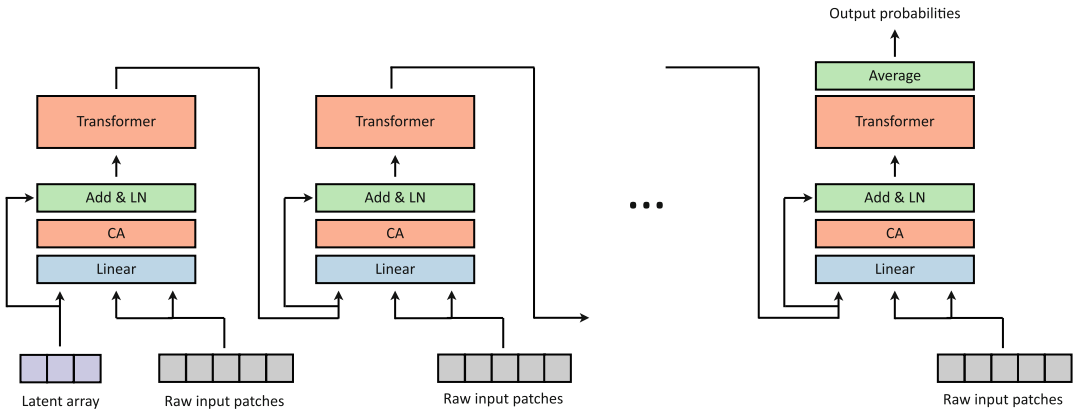

Fig. 8 The perceiver architecture [25, 26]. A set of latent tokens retrieve information from the image through cross attention. Self-attention is performed between the tokens to refine the learned representation. These operations are linear with respect to the number of image tokens. Figure inspired from [25, 26]

visual transformers without labels (Subheading 4.3), and image generation using generative adversarial networks (GANs) (Subheading 4.4).

4.1 Object Detection with Transformers

Detection is one of the early tasks that have seen improvements thanks to transformers. Detection is a combined recognition and localization problem; this means that a successful detection system should both recognize whether an object is present in an image and localize it spatially in the image. Carion et al. [14] is the first approach that uses transformers for detection.

DEtection TRansformer (DETR) [14] DETR first extracts visual representations with a convolutional network (Fig. 9).[3] Then, the encodings are processed by a transformer network. Finally, the processed tokens are provided to a transformer decoder. The decoder uses cross attention between a set of learned tokens and the image tokens encoded by the encoder and outputs a set of tokens. Each output token is then passed through a feed-forward network that predicts if an object is present in an image or not; if the object is indeed present, the network also predicts the class and spatial location of the object, i.e., coordinates within the image.

4.2 Image Segmentation with Transformers

The goal of image segmentation is to assign to each pixel of an image the label of the object it belongs to. The segmenter [27] is a purely ViT approach addressing image segmentation. The idea is to first use ViT to encode the image. Then, the segmenter learns a token per

[3] Note that, in DETR, the transformer is not directly used to extract the visual representation. Instead, it focuses on refining the visual representation to extract the object information.

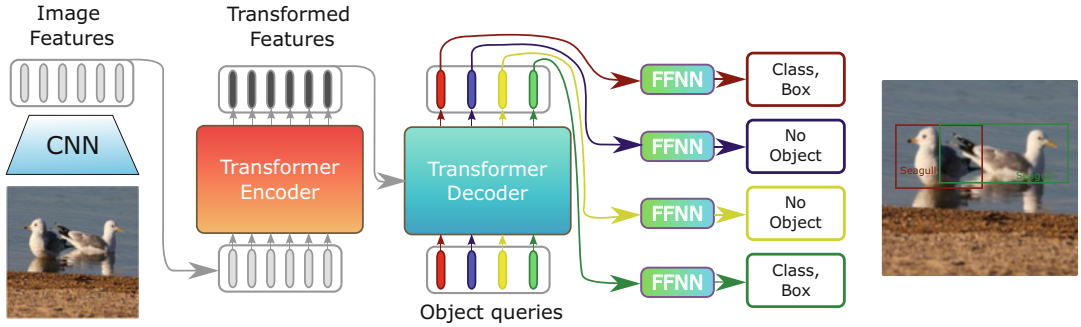

Fig. 9 The DETR architecture. It refines a CNN visual representation to extract object localization and classes. Figure inspired from [14]

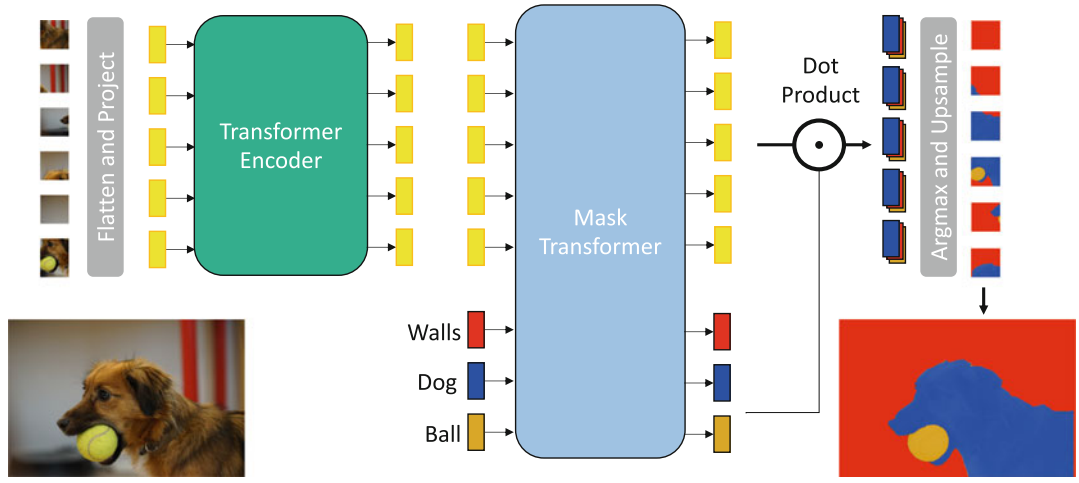

Fig. 10 The segmenter architecture. It is a purely ViT-based approach to perform semantic segmentation. Figure inspired from [27]

semantic label. The encoded patch tokens and the semantic tokens are then fed to a second transformer. Finally, by computing the scalar product between the semantic tokens and the image tokens, the network assigns a label to each patch. Figure 10 displays this.

4.3 Training Transformers Without Labels

Visual transformers have initially been trained for classification tasks. However, this tasks requires having access to massive amounts of labeled data, which can be hard to obtain (as discussed in Subheading 3.1). Subheadings 3.1 and 3.2 present ways to train ViT more efficiently. However, it would also be interesting to be able to train this type of networks with "cheaper" data. Therefore, the goal of this part is to introduce unsupervised learning with transformers, i.e., training transformers without any labels.

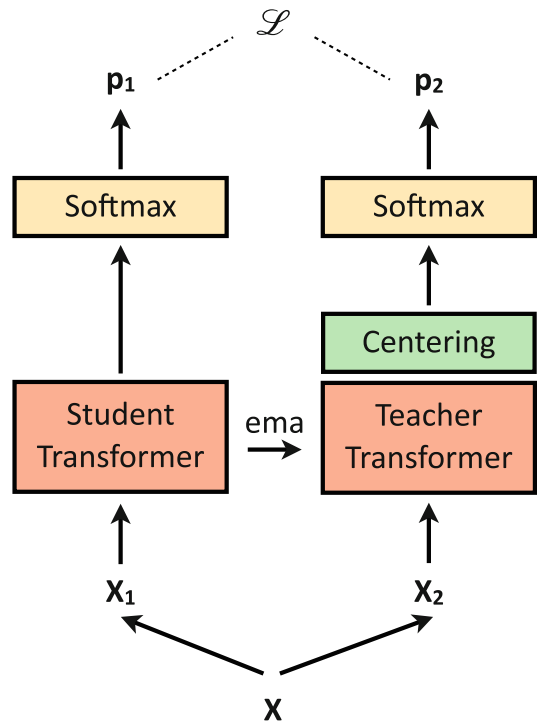

Fig. 11 The DINO training procedure. It consists in matching the outputs between two networks ($\boldsymbol{p}_1$ and $\boldsymbol{p}_2$) having two different augmentations ($\boldsymbol{X}_1$ and $\boldsymbol{X}_2$) of the same image as input ($\boldsymbol{X}$). The parameters of the teacher model are updated with an exponential moving average (ema) of the student parameters. Figure inspired from [28]

Self-DIstillation with NO labels (DINO) [28] DINO is one of the first works that trains a ViT with self-supervised learning (Fig. 11). The main idea is to have two ViT models following the teacher-student paradigm: the first model is updated through gradient descent, and the second is an exponential moving average of the first one. Then, the whole two-stream DINO network is trained using two augmentations of the same image, which are each passed to one of the two networks. The goal of the training is to match the output between the two networks, i.e., no matter the augmentation in the input data, both networks should produce the same result. The main finding of DINO is that the ViT is capable of learning a semantic understanding of the image, as the attention matrices display some semantic information. Figure 12 visualizes the attention matrix of the various ViT heads trained with DINO.

Masked Autoencoders (MAE) [29] Another way to train a ViT without supervision is by using an autoencoder architecture. Masked autoencoders (MAE) [29] perform a random masking of the input token and give the task to reconstruct the original image to a decoder. The encoder learns a representation that performs

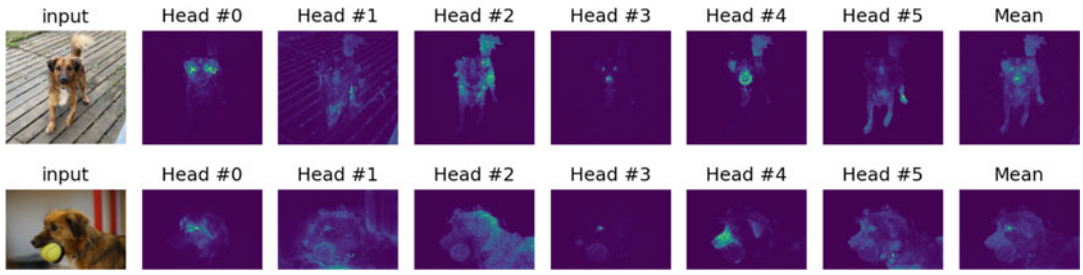

Fig. 12 DINO samples. Visualization of the attention matrix of ViT heads trained with DINO. The ViT discovers the semantic structure of an image in an unsupervised way

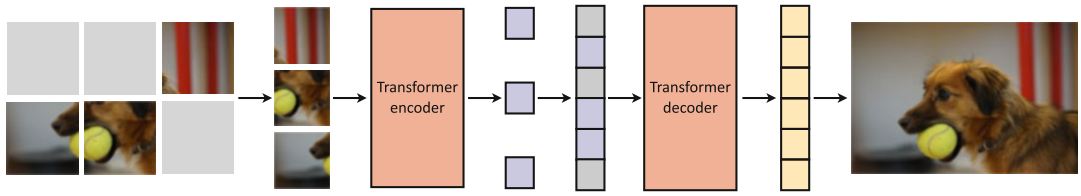

Fig. 13 The MAE training procedure. After masking some tokens of an image, the remaining tokens are fed to an encoder. Then a decoder tries to reconstruct the original image from this representation. Figure inspired from [29]

well in a given downstream task. This is illustrated in Fig. 13. One of the key observations of the MAE work [29] is that the decoder does not need to be very good for the encoder to achieve good performance: by using only a small decoder, MAE successfully trains a ViT in an autoencoder fashion.

4.4 Image Generation with Transformers and Attention

Attention and vision transformers have also helped in developing fresh ideas and creating new architectures for generative models and in particular for generative adversarial networks (GANs).

GANsformers [30] GANsformers are the most representative work of GANs with transformers, as they are a hybrid architecture using both attention and CNNs. The GANsformer architecture is illustrated in Fig. 14. The model first splits the latent vector of a GAN into multiple tokens. Then, a cross-attention mechanism is used to improve the generated feature maps, and at the same time, the GANsformer architecture retrieves information from the generated feature map to enrich the tokens. This mechanism allows the GAN to have better and richer semantic knowledge, which is showed to be useful for generating multimodal images.

StyleSwin [31] Another approach for generative modeling is to purely use a ViT architecture like StyleSwin [31]. StyleSwin is a GAN that leverages a similar type of attention as the Swin transformer [24]. This allows to generate high-definition images without having to deal with the quadratic cost problem.

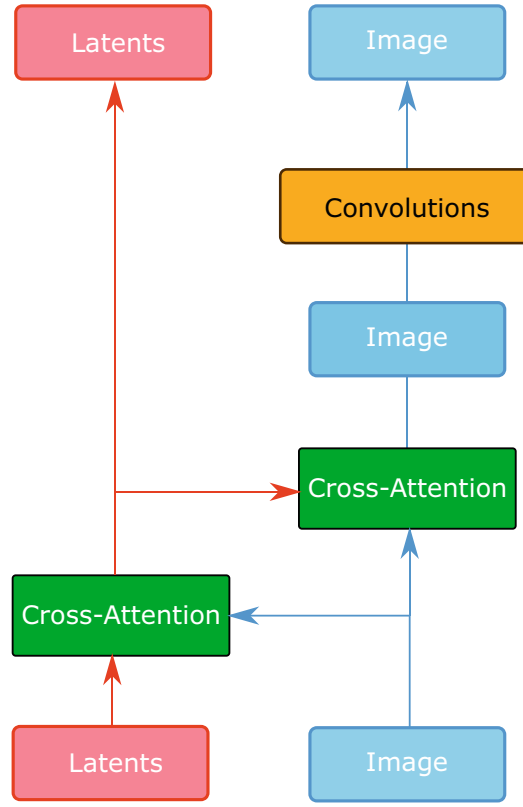

Fig. 14 GANsformer architecture. A set of latents contribute to bring information to a CNN feature map. Figure inspired from [30]

5 Vision Transformers for Other Domains

In this section, we present applications of visual transformers to other domains. First, we describe multimodal transformers operating with vision and language (Subheading 5.1), then we describe video-level attention and video transformers (Subheadings 5.2 and 5.3), and finally we present multimodal video transformers operating with vision, language, and audio (Subheading 5.4).

5.1 Multimodal Transformers: Vision and Language

As transformers have found tremendous success in both natural language processing and computer vision, their use in vision-language tasks is also of interest. In this section, we describe some representative multimodal methods for vision and language: ViLBERT (Subheading 5.1.1), DALL-E (Subheading 5.1.3), and CLIP (Subheading 5.1.2).

5.1.1 ViLBERT

Vision-and-language BERT (VilBERT) [32] is an example of architecture that fuses two modalities. It consists of two parallel streams, each one working with one modality. The vision stream extracts

bounding boxes from images via an object detection network, by encoding their position. The language stream embeds word vectors and extracts feature vectors using the basic transformer encoder block [4] (Fig. 3 left). These two resulting feature vectors are then fused together by a cross-attention layer (Subheading 1.3.2). This follows the standard architecture of the transformer encoder block, where the keys and values of one modality are passed onto the MCA block of the other modality. The output of the cross-attention layer is passed into another transformer encoder block, and these two layers are stacked multiple times.

The language stream is initialized with BERT trained on Book Corpus [12] and Wikipedia [11], while the visual stream is initialized with Faster R-CNN [33]. On top of the pretraining of each stream, the whole architecture is pretrained on the Conceptual Captions dataset [34] on two pretext tasks.

ViLBERT has been proven powerful for a variety of multimodal tasks. In the original paper, ViLBERT was fined-tuned to a variety of tasks, including visual question answering, visual commonsense reasoning, referring expressions, and caption-based image retrieval.

5.1.2 CLIP

Connecting Text and Images (CLIP) [35] is designed to address two major issues of deep learning models: costly datasets and inflexibility. While most deep learning models are trained on labeled datasets, CLIP is trained on 400 million text-image pairs that are scraped from the Internet. This reduces the labor of having to manually label millions of images that are required to train powerful deep learning models. When models are trained on one specific dataset, they also tend to be difficult to extend to other applications. For instance, the accuracy of a model trained on ImageNet is generally limited to its own dataset and cannot be applied to real-world problems. To optimize training, CLIP models learn to perform a wide variety of tasks during pretraining, and this task allows for zero-shot transfer to many existing datasets. While there are still several potential improvements, this approach is competitive to supervised models that are trained on specific datasets.

CLIP Architecture and Training

CLIP is used to measure the similarity between the text input and the image generated from a latent vector. At the core of the approach is the idea of learning perception from supervision contained in natural language. Methods which work on natural language can learn passively from the supervision contained in the vast amount of text on the Internet.

Given a batch of N (image, text) pairs, CLIP is trained to predict which of the $N \times N$ possible (image, text) pairings across a batch actually occurred. To do this, CLIP learns a multimodal embedding space by jointly training an image encoder and a text encoder to maximize the cosine similarity of the image and text

embeddings of the N real pairs in the batch while minimizing the cosine similarity of the embeddings of the $N^2 - N$ incorrect pairings. A symmetric cross-entropy loss over these similarity scores is optimized.

Two different architectures were considered for the image encoder. For the first, ResNet-50 [36] is used as the base architecture for the image encoder due to its widespread adoption and proven performance. Several modifications were made to the original version of ResNet. For the second architecture, ViT is used with some minor modifications: first, adding an extra layer normalization to the combined patch and position embeddings before the transformer and, second, using a slightly different initialization scheme.

The text encoder is a standard transformer [4] (Subheading 2.1) with the architecture modifications described in [35]. As a base size, CLIP uses a 63M-parameter 12-layer 512-wide model with eight attention heads. The transformer operates on a lowercased byte pair encoding (BPE) representation of the text with a 49,152 vocab size [37]. The max sequence length is capped at 76. The text sequence is bracketed with [SOS] and [EOS] tokens,[4] and the activations of the highest layer of the transformer at the [EOS] token are treated as the feature representation of the text which is layer normalized and then linearly projected into the multimodal embedding space.

5.1.3 DALL-E and DALL-E 2

DALL-E [38] is another example of the application of transformers in vision. It generates images from a natural language prompt—some examples include "an armchair in the shape of an avocado" and "a penguin made of watermelon." It uses a decoder-only model, which is similar to GPT-3 [39]. DALL-E uses 12 billion parameters and is pretrained on Conceptual Captions [34] with over 3.3 million text-image pairs. DALL-E 2 [40] is the upgraded version of DALL-E, based on diffusion models and CLIP (Subheading 5.1.2), and allows better performances with more realistic and accurate generated images. In addition to producing more realistic results with a better resolution than DALL-E, DALL-E 2 is also able to edit the outputs. Indeed, with DALL-E 2, one can add or remove realistically an element in the output and can also generate different variations of the same output. These two models clearly demonstrate the powerful nature and scalability of transformers that are capable of efficiently processing a web-scale amount of data.

[4] [SOS], start of sequence; [EOS], end of sequence

5.1.4 Flamingo

Flamingo [41] is a visual language model (VLM) tackling a wide range of multimodal tasks based on few-shot learning. This is an adaptation of large language models (LLMs) handling an extra visual modality with 80B parameters.

Flamingo consists of three main components: a vision encoder, a perceiver resampler, and a language model. First, to encode images or videos, a vision convolutional encoder [42] is pretrained in a contrastive way, using image and text pairs.[5] Then, inspired by the perceiver architecture [25] (detailed in Subheading 1.3.2), the perceiver resampler takes a variable number of encoded visual features and outputs a fixed-length latent code. Finally, this visual latent code conditions the language model by querying language tokens through cross-attention blocks. Those cross-attention blocks are interleaved with pretrained and frozen language model blocks.

The whole model is trained using three different kinds of datasets without annotations (text with image content from webpages [41], text and image pairs [41, 43], and text and video pairs [41]). Once the model is trained, it is fine-tuned using few-shot learning techniques to tackle specific tasks.

5.2 Video Attention

Video understanding is a long-standing problem, and despite incredible computer vision advances, obtaining the best video representation is still an active research area. Videos require employing effective spatiotemporal processing of RGB and time streams to capture long-range interactions [44, 45] while focusing on important video parts [46] with minimum computational resources [47].

Typically, video understanding benefits from 2D computer vision, by adapting 2D image processing methods to 3D spatiotemporal methods [48]. And through the Video Vision Transformer (ViViT) [49], history repeats itself. Indeed, with the rise of transformers [4] and the recent advances in image classification [5], video transformers appear as logical successors of CNNs.

However, in addition to the computationally expensive video processing, transformers also require a lot of computational resources. Thus, developing efficient spatiotemporal attention mechanisms is essential [25, 49, 50].

In this section, we first describe the general principle of video transformers (Subheading 5.2.1), and then, we detail three different attention mechanisms used for video representation (Subheadings 5.2.2, 5.2.3, and 5.2.4).

[5] The text is encoded using a pretrained BERT model [10].

5.2.1 General Principle

Generally, inputs of video transformers are RGB video clips $\mathsf{X} \in \mathbb{R}^{F \times H \times W \times 3}$, with F frames of size $H \times W$.

To begin with, video transformers split the input video clip X into ST tokens $\boldsymbol{x}_i \in \mathbb{R}^K$, where S and T are, respectively, the number of tokens along the spatial and temporal dimension and K is the size of a token.

To do so, the simplest method extracts nonoverlapping 2D patches of size $P \times P$ from each frame [5], as used in TimeSformer [50]. This results in $S = HW/P^2$, $T = F$, and $K = P^2$.

However, there exist more elegant and efficient token extraction methods for videos. For instance, in ViViT [49], the authors propose to extract 3D volumes from videos (involving $T \neq F$) to capture spatiotemporal information within tokens. In TokenLearner [47], they propose a learnable token extractor to select the most important parts of the video.

Once raw tokens $\boldsymbol{x}_i$ are extracted, transformer architectures aim to map them into d-dimensional embedding vectors $\boldsymbol{Z} \in \mathbb{R}^{ST \times d}$ using a linear embedding $\boldsymbol{E} \in \mathbb{R}^{d \times K}$:

$$\boldsymbol{Z} = [\boldsymbol{z}_{cls}, E\boldsymbol{x}_1, E\boldsymbol{x}_2, \ldots, E\boldsymbol{x}_{ST}] + \mathbf{PE}, \tag{10}$$

where $\boldsymbol{z}_{cls} \in \mathbb{R}^d$ is a classification token that encodes information from all tokens of a single sample [10] and $\mathbf{PE} \in \mathbb{R}^{ST \times d}$ is a positional embedding that encodes the spatiotemporal position of tokens, since the subsequent attention blocks are permutation invariant [4].

In the end, embedding vectors $\boldsymbol{Z}$ pass through a sequence of L transformer layers. A transformer layer ℓ is composed of a series of multi-head self-attention (MSA) [4], layer normalization (LN) [51], and MLP blocks:

$$\begin{aligned} \boldsymbol{Y}^\ell &= \mathrm{MSA}(\mathrm{LN}(\boldsymbol{Z}^\ell)) + \boldsymbol{Z}^\ell, \\ \boldsymbol{Z}^{\ell+1} &= \mathrm{MLP}(\mathrm{LN}(\boldsymbol{Y}^\ell)) + \boldsymbol{Y}^\ell. \end{aligned} \tag{11}$$

In this way, as shown in Fig. 2, we denote four different components in a video transformer layer: the query-key-value (QKV) projection, the MSA block, the MSA projection, and the MLP. For a layer with h heads, the complexity of each component is [4]:

- *QKV projection*: $\mathcal{O}(h.(2STdd_k + STdd_v)$
- *MSA*: $\mathcal{O}(hS^2T^2.(d_k + d_v))$
- *MSA projection*: $\mathcal{O}(STHd_vd)$
- *MLP*: $\mathcal{O}(STd^2)$

We note that the MSA complexity is the most impacting component, with a quadratic complexity with respect to the number of tokens. Hence, for comprehension and clarity purposes, in the rest of the section, we consider the global complexity of a video transformer with L layers to equal to $\mathcal{O}(LS^2T^2)$.

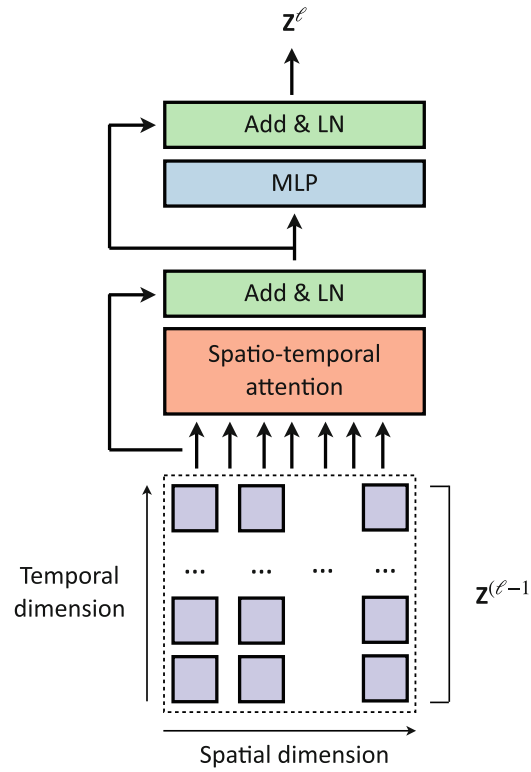

Fig. 15 Full space-time attention mechanism. Embedding tokens at layer $\ell - 1$, $\boldsymbol{Z}^{(\ell-1)}$ are all fed simultaneously through a unique spatiotemporal attention block. Finally, the spatiotemporal embedding is passed through an MLP and normalized to output embedding tokens of the next layer, $\boldsymbol{Z}^{\ell}$. Figure inspired from [50]

5.2.2 Full Space-Time Attention

As described in [49, 50], *full space-time attention* mechanism is the most basic and direct spatiotemporal attention mechanism. As shown in Fig. 15, it consists in computing self-attention across all pairs of extracted tokens.

This method results in a heavy complexity of $\mathcal{O}(LS^2T^2)$ [49, 50]. This quadratic complexity can fast be memory-consuming, in which it is especially true when considering videos. Therefore, using full space-time attention mechanism is impractical [50].

5.2.3 Divided Space-Time Attention

A smarter and more efficient way to compute spatiotemporal attention is the *divided space-time attention* mechanism, first described in [50].

As shown in Fig. 16, it relies on computing spatial and temporal attention separately in each transformer layer. Indeed, we first compute the spatial attention, i.e., self-attention within each temporal index, and then the temporal attention, i.e., self-attention across all temporal indices.

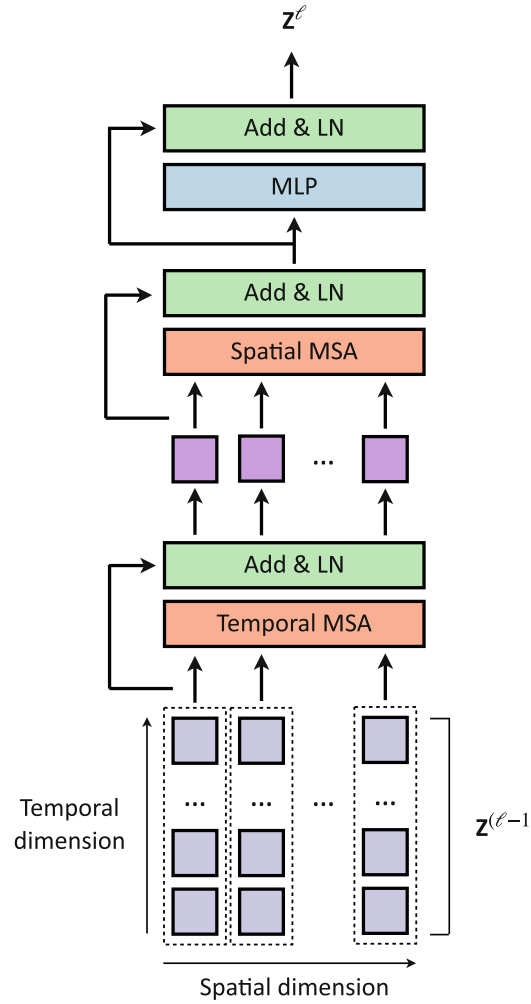

Fig. 16 Divided space-time attention mechanism. Embedding tokens at layer $\ell - 1$, $\boldsymbol{Z}^{(\ell-1)}$ are first processed along the temporal dimension through a first MSA block, and the resulting tokens are processed along the spatial dimension. Finally, the spatiotemporal embedding is passed through an MLP and normalized to output embedding tokens of the next layer, $\boldsymbol{Z}^{\ell}$. Figure inspired from [50]

The complexity of this attention mechanism is $\mathcal{O}(LST.(S + T))$ [50]. By separating the calculation of the self-attention over the different dimensions, one tames the quadratic complexity of the MSA module. This mechanism highly reduces the complexity of a model with respect to the full space-time complexity. Therefore, it is reasonable to use it to process videos [50].

5.2.4 Cross-Attention Bottlenecks

An even more refined way to reduce the computational cost of attention calculation consists of using cross attention as a bottleneck. For instance, as shown in Fig. 17 and mentioned in Subheading 3.2, the perceiver [25] projects the extracted tokens $\boldsymbol{x}_i$ into a

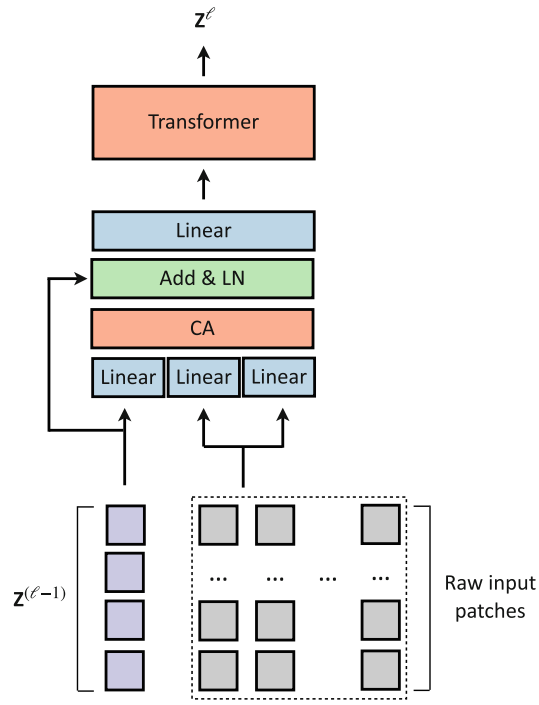

Fig. 17 Attention bottleneck mechanism. Raw input patches and embedding tokens at layer $\ell - 1$, $\boldsymbol{Z}^{(\ell-1)}$ are fed to a cross-attention block (CA) and then normalized and projected. Finally, the resulting embedding is passed through a transformer to output embedding tokens of the next layer, $\boldsymbol{Z}^{\ell}$. Figure inspired from [25]

very low-dimensional embedding through a cross-attention block placed before the transformer layers.

Here, the cross-attention block placed before the L transformer layers reduce the input dimension from ST to N, where $N \ll ST$,[6] thus resulting in a complexity of $\mathcal{O}(STN)$. Hence, the total complexity of this attention block is $\mathcal{O}(STN + LN^2)$. It reduces again the complexity of a model with respect to the *divided space-time attention* mechanism. We note that it enables to design deep architectures, as in the perceiver [25], and then it enables the extraction of higher-level features.

5.2.5 Factorized Encoder

Lastly, the *factorized encoder* [49] architecture is the most efficient with respect to the complexity/performance trade-off.

As in *divided space-time attention*, the *factorized encoder* aims to compute spatial and temporal attention separately. Nevertheless, as shown in Fig. 18, instead of mixing spatiotemporal tokens in each transformer layer, here, there exist two separate encoders:

[6] In practice, $N \leq 512$ for perceiver [25], against $ST = 16 \times 16 \times (32/2) = 4096$ for ViViT-L [49]

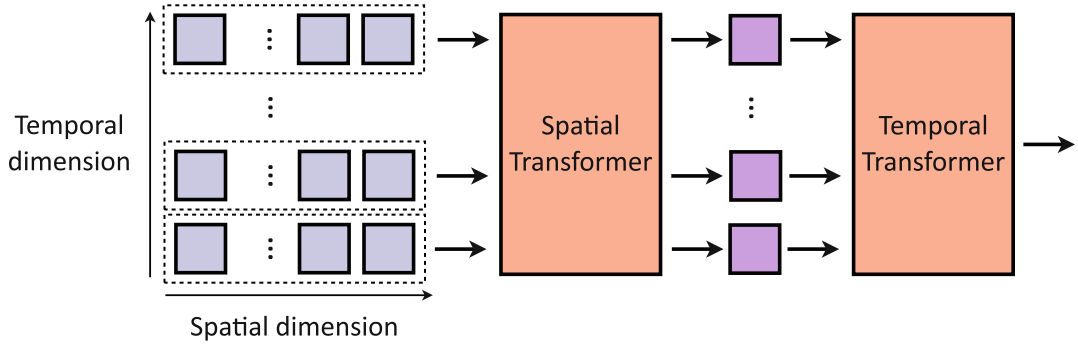

Fig. 18 Factorized encoder mechanism. First, a spatial transformer processes input tokens along the spatial dimension. Then, a temporal transformer processes the resulting spatial embedding along the temporal dimension. Figure inspired from [25]

First, a representation of each temporal index is obtained, thanks to a spatial encoder with L_s layers. Second, these tokens are passed through a temporal encoder with L_t layers (i.e., $L = L_s + L_t$).

Hence, the complexity of a such architecture has two main components: the spatial encoder complexity of $\mathcal{O}(L_s S^2)$ and the temporal encoder complexity of $\mathcal{O}(L_t T^2)$. It results in a global complexity of $\mathcal{O}(L_s S^2 + L_t T^2)$. Thus, it leads to very lightweight models. However, as it first extracts per-frame features and then aggregates them to a final representation, it corresponds to a late-fusion mechanism, which can sometimes be a drawback as it does not mix spatial and temporal information simultaneously [52].

5.3 Video Transformers

In this section, we present two modern transformer-based architectures for video classification. We start by introducing the TimeSformer architecture in Subheading 5.3.1 and then the ViViT architecture in Subheading 5.3.2.

5.3.1 TimeSformer

TimeSformer [50] is one of the first architectures with space-time attention that impacted the video classification field. It follows the same structure and principle described in Subheading 5.2.1.

First, it takes as input an RGB video clip sampled at a rate of 1/32 and decomposed into 2D 16×16 patches.

As shown in Fig. 19, the TimeSformer architecture is based on the ViT architecture (Subheading 2.3), with 12 12-headed MSA layers. However, the added value compared to the ViT is that TimeSfomer uses the *divided space-time attention* mechanism (Subheading 5.2.3). Such attention mechanism enables to capture high-level spatiotemporal features while taming the complexity of the model. Moreover, the authors introduce three variants of the architecture: (i) TimeSformer, the standard version of the model, that operates on 8 frames of 224×224; (ii) TimeSformer-L, a configuration with high spatial resolution, that operates on 16 frames of 448×448; and (iii) TimeSformer-HR, a long temporal range setup, that operates on 96 frames of 224×224.

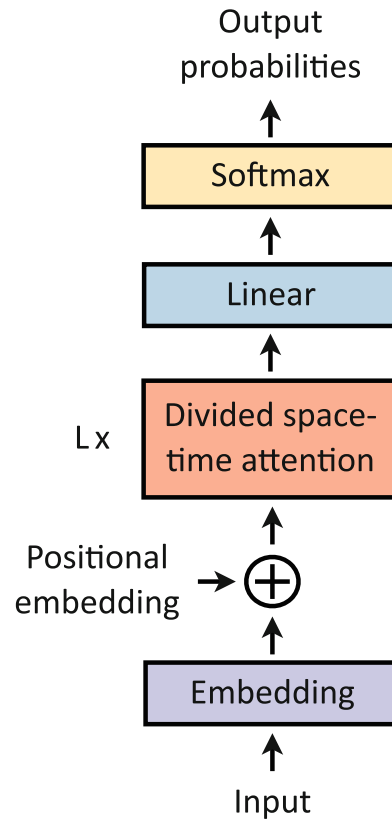

Fig. 19 TimeSformer architecture. The TimeSformer first projects input to embedding tokens, which are summed to positional embedding tokens. The resulting tokens are then passed through *L* divided space-time attention blocks and then linearly projected to obtain output probabilities

Finally, the terminal classification token embedding is passed through an MLP to output a probability for all video classes. During inference, the final prediction is obtained by averaging the output probabilities from three different spatial crops of the input video clip (top left, center, and bottom right).

TimeSformer achieves similar state-of-the-art performances as the 3D CNNs [53, 54] on various video classification datasets, such as Kinetics-400 and Kinetics-600 [55]. Note the TimeSformer is much faster to train (416 training hours against 3840 hours [50] for a SlowFast architecture [54]) and, also, more efficient (0.59 TFLOPs against 1.97 TFLOPs [50] for a SlowFast architecture [53]).

5.3.2 ViViT

ViViT [49] is the main extension of the ViT [5] architecture (Subheading 2.3) for video classification.

First, the authors use a 16 tubelet embedding instead of a 2D patch embedding, as mentioned in Subheading 5.2.1. This alternate embedding method aims to capture the spatiotemporal

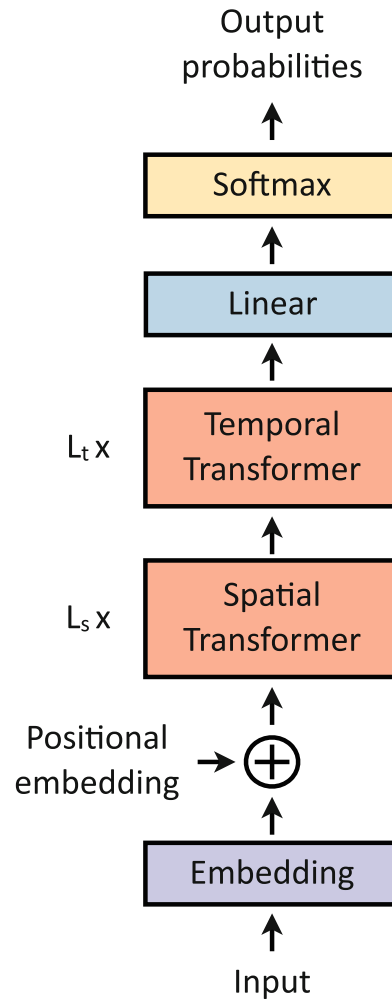

Fig. 20 ViViT architecture. The ViViT first projects input to embedding tokens, which are summed to positional embedding tokens. The resulting tokens are first passed through L_s spatial attention blocks and then through L_t temporal attention blocks. The resulting output is linearly projected to obtain output probabilities

information from the tokenization step, unlike standard architectures that fuse spatiotemporal information from the first attention block.

As shown in Fig. 20, the ViViT architecture is based on *factorized encoder* architecture (Subheading 5.2.5) and consists of one spatial and one temporal encoder operating on input clips with 32 frames of 224×224. The spatial encoder uses one of the three ViT variants as backbone.[7] For the temporal encoder, the number

[7] ViT-B: 12 12-headed MSA layers; ViT-L: 24 16-headed MSA layers; and ViT-H: 32 16-headed MSA layers.

of layers does not impact much the performance, so that, according to the performance/complexity trade-off, the number MSA layers is fixed at 4. The authors show that such architecture reaches high performances while reducing drastically the complexity.

Finally, as in TimeSformer (Subheading 5.3.1), ViViT outputs probabilities for all video classes through the last classification token embedding and averages the obtained probabilities across three crops of each input clip (top left, center, and bottom right).

ViViT outperforms both 3D CNNs [53, 54] and TimeSformer [50] on the Kinetics-400 and Kinetics-600 datasets [55]. Note the complexity of this architecture is highly reduced in comparison to other state-of-the-art models. For instance, the number of FLOPs for a ViViT-L/$16 \times 16 \times 2$ is 3.89×10^{12} against 7.14×10^{12} for a TimeSformer-L [50] and 7.14×10^{12} for a SlowFast [53] architecture.

5.4 Multimodal Video Transformers

Nowadays, one of the main gaps between artificial and human intelligence is the ability for us to process multimodal signals and to enrich the analysis by mixing the different modalities. Moreover, until recently, deep learning models have been focusing mostly on very specific visual tasks, typically based on a single modality, such as image classification [5, 17, 18, 56, 57], audio classification [25, 52, 58, 59], and machine translation [10, 60–63]. These two factors combined have pushed researchers to take up multimodal challenges.

The *default* solution for multimodal tasks consists in first creating an individual model (or network) per modality and then in fusing the resulting single-modal features together [64, 65]. Yet, this approach fails to model interactions or correlations among different modalities. However, the recent rise of attention [4, 5, 49] is promising for multimodal applications, since attention performs very well at combining multiple inputs [25, 52, 66, 67].

Here, we present two main ways of dealing with several modalities:

1. **Concatenating tokens from different modalities into one vector [25, 66].** The multimodal video transformer (MM-ViT) [66] combines raw RGB frames, motion features, and audio spectrogram for video action recognition. To do so, the authors fuse tokens from all different modalities into a single-input embedding and pass it through transformer layers. However, a drawback of this method is that it fails to distinguish well one modality to another. To overcome this issue, the authors of the perceiver [25] propose to learn a modality embedding in addition to the positional embedding (*see* Subheadings 3.2 and 5.2.1). This allows associating each token

with its modality. Nevertheless, given that (i) the complexity of a transformer layer is quadratic with respect to the number of tokens (Subheading 5.2.1) and (ii), with this method, the number of tokens is multiplied by the number of modalities, it may lead to skyrocketing computational cost [66].

2. **Exploiting cross attention [52, 67, 68].** Several modern approaches exploit cross attention to mix multiple modalities, such as [52] for audio and video, [67] for text and video, and [68] for audio, text, and video. The commonality among all these methods is that they exploit the intrinsic properties of cross attention by querying one modality with a key-value pair from the other one [52, 67]. This idea can be easily generalized to more than two modalities by computing cross attention across each combination of modalities [68].

6 Conclusion

Attention is an intuitive and efficient technique that enables handling local and global cues.

On this basis, the first pure attention architecture, the transformer [4], has been designed for NLP purposes. Quickly, the computer vision field has adapted the transformer architecture for image classification, by designing the first visual transformer model: the vision transformer (ViT) [5].

However, even if transformers naturally lead to high performances, the raw attention mechanism is a computationally greedy and heavy technique. For this reason, several enhanced and refined derivatives of attention mechanisms have been proposed [21–26].

Then, rapidly, a wide variety of other tasks have been conquered by transformer-based architectures, such as object detection [14], image segmentation [27], self-supervised learning [28, 29], and image generation [30, 31]. In addition, transformer-based architectures are particularly well suited to handle multidimensional tasks. This is because multimodal signals are easily combined through attention blocks, in particular vision and language cues [32, 35, 38] and spatiotemporal signals are also easily tamed, as in [25, 49, 50].

For these reasons, transformer-based architectures enabled many fields to make tremendous progresses in the last few years. In the future, transformers will need to become more and more computationally efficient, e.g., to be usable on cellphones, and will play a huge role to tackle multimodal challenges and bridge together most AI fields.

References

1. Bahdanau D, Cho K, Bengio Y (2015) Neural machine translation by jointly learning to align and translate. In: International conference on learning representations
2. Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Empirical methods in natural language processing, association for computational linguistics, pp 1724–1734
3. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, vol 27
4. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, vol 30
5. Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby N (2021) An image is worth 16×16 words: transformers for image recognition at scale. In: International conference on learning representations
6. Press O, Wolf L (2017) Using the output embedding to improve language models. In: Proceedings of the 15th conference of the European chapter of the association for computational linguistics: volume 2, short papers, association for computational linguistics, pp 157–163
7. Fedus W, Zoph B, Shazeer N (2021) Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. arXiv preprint arXiv:210103961
8. Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu PJ (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. J Mach Learn Res 21(140):1–67
9. Khan S, Naseer M, Hayat M, Zamir SW, Khan FS, Shah M (2021) Transformers in vision: a survey. ACM Comput Surv 24:200
10. Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), association for computational linguistics, pp 4171–4186
11. Wikimedia Foundation (2019) Wikimedia downloads. https://dumps.wikimedia.org
12. Zhu Y, Kiros R, Zemel R, Salakhutdinov R, Urtasun R, Torralba A, Fidler S (2015) Aligning books and movies: towards story-like visual explanations by watching movies and reading books. In: Proceedings of the international conference on computer vision, pp 19–27
13. Wang X, Girshick R, Gupta A, He K (2018) Non-local neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7794–7803
14. Carion N, Massa F, Synnaeve G, Usunier N, Kirillov A, Zagoruyko S (2020) End-to-end object detection with transformers. In: Proceedings of the European conference on computer vision. Springer, Berlin, pp 213–229
15. Ramachandran P, Parmar N, Vaswani A, Bello I, Levskaya A, Shlens J (2019) Stand-alone self-attention in vision models. In: Advances in neural information processing systems, vol 32
16. Wang H, Zhu Y, Green B, Adam H, Yuille A, Chen LC (2020) Axial-deeplab: stand-alone axial-attention for panoptic segmentation. In: Proceedings of the European conference on computer vision. Springer, Berlin, pp 108–126
17. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 248–255
18. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Tech rep University of Toronto, Toronto, ON
19. Sun C, Shrivastava A, Singh S, Gupta A (2017) Revisiting unreasonable effectiveness of data in deep learning era. In: Proceedings of the international conference on computer vision, pp 843–852
20. Selva J, Johansen AS, Escalera S, Nasrollahi K, Moeslund TB, Clapés A (2022) Video transformers: a survey. arXiv preprint arXiv:220105991
21. Touvron H, Cord M, Douze M, Massa F, Sablayrolles A, Jégou H (2021) Training data-efficient image transformers & distillation through attention. In: International conference on machine learning. PMLR, pp 10347–10357

22. d'Ascoli S, Touvron H, Leavitt ML, Morcos AS, Biroli G, Sagun L (2021) Convit: improving vision transformers with soft convolutional inductive biases. In: International conference on machine learning. PMLR, pp 2286–2296
23. Hassani A, Walton S, Shah N, Abuduweili A, Li J, Shi H (2021) Escaping the big data paradigm with compact transformers. arXiv preprint arXiv:210405704
24. Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S, Guo B (2021) Swin transformer: hierarchical vision transformer using shifted windows. In: Proceedings of the international conference on computer vision
25. Jaegle A, Gimeno F, Brock A, Vinyals O, Zisserman A, Carreira J (2021) Perceiver: general perception with iterative attention. In: International conference on machine learning. PMLR, pp 4651–4664
26. Jaegle A, Borgeaud S, Alayrac JB, Doersch C, Ionescu C, Ding D, Koppula S, Zoran D, Brock A, Shelhamer E et al (2021) Perceiver IO: a general architecture for structured inputs & outputs. arXiv preprint arXiv:210714795
27. Strudel R, Garcia R, Laptev I, Schmid C (2021) Segmenter: transformer for semantic segmentation. In: Proceedings of the international conference on computer vision, pp 7262–7272
28. Caron M, Touvron H, Misra I, Jégou H, Mairal J, Bojanowski P, Joulin A (2021) Emerging properties in self-supervised vision transformers. In: Proceedings of the international conference on computer vision
29. He K, Chen X, Xie S, Li Y, Dollár P, Girshick R (2021) Masked autoencoders are scalable vision learners. arXiv preprint arXiv:211106377
30. Hudson DA, Zitnick L (2021) Generative adversarial transformers. In: International conference on machine learning. PMLR, pp 4487–4499
31. Zhang B, Gu S, Zhang B, Bao J, Chen D, Wen F, Wang Y, Guo B (2022) Styleswin: transformer-based GAN for high-resolution image generation. In: Proceedings of the IEEE conference on computer vision and pattern recognition
32. Lu J, Batra D, Parikh D, Lee S (2019) Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In: Advances in neural information processing systems, vol 32
33. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, vol 28
34. Sharma P, Ding N, Goodman S, Soricut R (2018) Conceptual captions: a cleaned, hypernymed, image alt-text dataset for automatic image captioning. In: Proceedings of ACL
35. Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J et al (2021) Learning transferable visual models from natural language supervision. In: International conference on machine learning. PMLR, pp 8748–8763
36. He X, Peng Y (2017) Fine-grained image classification via combining vision and language. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5994–6002
37. Sennrich R, Haddow B, Birch A (2016) Neural machine translation of rare words with subword units. In: Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers), association for computational linguistics, pp 1715–1725
38. Ramesh A, Pavlov M, Goh G, Gray S, Voss C, Radford A, Chen M, Sutskever I (2021) Zero-shot text-to-image generation. In: International conference on machine learning. PMLR, pp 8821–8831
39. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, et al (2020) Language models are few-shot learners. In: Advances in neural information processing systems, vol 33, pp 1877–1901
40. Ramesh A, Dhariwal P, Nichol A, Chu C, Chen M (2022) Hierarchical text-conditional image generation with clip latents. arXiv preprint arXiv:220406125
41. Alayrac JB, Donahue J, Luc P, Miech A, Barr I, Hasson Y, Lenc K, Mensch A, Millican K, Reynolds M et al (2022) Flamingo: a visual language model for few-shot learning. arXiv preprint arXiv:220414198
42. Brock A, De S, Smith SL, Simonyan K (2021) High-performance large-scale image recognition without normalization. In: International conference on machine learning. PMLR, pp 1059–1071
43. Jia C, Yang Y, Xia Y, Chen YT, Parekh Z, Pham H, Le Q, Sung YH, Li Z, Duerig T (2021) Scaling up visual and vision-language

representation learning with noisy text supervision. In: International conference on machine learning. PMLR, pp 4904–4916

44. Epstein D, Vondrick C (2021) Learning goals from failure. In: Proceedings of the IEEE conference on computer vision and pattern recognition
45. Marin-Jimenez MJ, Kalogeiton V, Medina-Suarez P, Zisserman A (2019) Laeo-net: revisiting people looking at each other in videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition
46. Nagrani A, Yang S, Arnab A, Jansen A, Schmid C, Sun C (2021) Attention bottlenecks for multimodal fusion. In: Advances in neural information processing systems
47. Ryoo M, Piergiovanni A, Arnab A, Dehghani M, Angelova A (2021) Tokenlearner: Adaptive space-time tokenization for videos. In: Advances in neural information processing systems, vol 34
48. Hara K, Kataoka H, Satoh Y (2018) Can spatiotemporal 3d CNNs retrace the history of 2d CNNs and imagenet? In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6546–6555
49. Arnab A, Dehghani M, Heigold G, Sun C, Lučić M, Schmid C (2021) Vivit: a video vision transformer. In: Proceedings of the international conference on computer vision, pp 6836–6846
50. Bertasius G, Wang H, Torresani L (2021) Is space-time attention all you need for video understanding? In: International conference on machine learning
51. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv preprint arXiv:160706450
52. Nagrani A, Yang S, Arnab A, Jansen A, Schmid C, Sun C (2021) Attention bottlenecks for multimodal fusion. In: Advances in neural information processing systems, vol 34
53. Feichtenhofer C, Fan H, Malik J, He K (2019) Slowfast networks for video recognition. In: Proceedings of the international conference on computer vision, pp 6202–6211
54. Carreira J, Zisserman A (2017) Quo vadis, action recognition? A new model and the kinetics dataset. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6299–6308
55. Kay W, Carreira J, Simonyan K, Zhang B, Hillier C, Vijayanarasimhan S, Viola F, Green T, Back T, Natsev P et al (2017) The kinetics human action video dataset. arXiv preprint arXiv:170506950
56. Wu H, Xiao B, Codella N, Liu M, Dai X, Yuan L, Zhang L (2021) CvT: introducing convolutions to vision transformers. In: Proceedings of the international conference on computer vision, pp 22–31
57. Touvron H, Cord M, Sablayrolles A, Synnaeve G, Jégou H (2021) Going deeper with image transformers. In: Proceedings of the international conference on computer vision, pp 32–42
58. Gemmeke JF, Ellis DP, Freedman D, Jansen A, Lawrence W, Moore RC, Plakal M, Ritter M (2017) Audio set: an ontology and human-labeled dataset for audio events. In: IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 776–780
59. Gong Y, Chung YA, Glass J (2021) AST: audio spectrogram transformer. In: Proceedings of interspeech 2021, pp 571–575
60. Bojar O, Buck C, Federmann C, Haddow B, Koehn P, Leveling J, Monz C, Pecina P, Post M, Saint-Amand H, et al (2014) Findings of the 2014 workshop on statistical machine translation. In: Proceedings of the 9th workshop on statistical machine translation, pp 12–58
61. Liu X, Duh K, Liu L, Gao J (2020) Very deep transformers for neural machine translation. arXiv preprint arXiv:200807772
62. Edunov S, Ott M, Auli M, Grangier D (2018) Understanding back-translation at scale. In: Empirical methods in natural language processing, association for computational linguistics, pp 489–500
63. Lin Z, Pan X, Wang M, Qiu X, Feng J, Zhou H, Li L (2020) Pre-training multilingual neural machine translation by leveraging alignment information. In: Empirical methods in natural language processing, association for computational linguistics, pp 2649–2663
64. Owens A, Efros AA (2018) Audio-visual scene analysis with self-supervised multisensory features. In: Proceedings of the European conference on computer cision, pp 631–648
65. Ramanishka V, Das A, Park DH, Venugopalan S, Hendricks LA, Rohrbach M, Saenko K (2016) Multimodal video description. In: Proceedings of the ACM international conference on multimedia, pp 1092–1096

66. Chen J, Ho CM (2022) Mm-vit: Multi-modal video transformer for compressed video action recognition. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision, pp 1910–1921
67. Narasimhan M, Rohrbach A, Darrell T (2021) Clip-it! language-guided video summarization. In: Advances in neural information processing systems, vol 34
68. Liu ZS, Courant R, Kalogeiton V (2022) FunnyNet: Audiovisual Learning of Funny Moments in Videos. In: Proceedings of the Asian conference on computer vision. Springer, Macau, China, pp 3308–3325