Università degli
Studi di Milano-Bicocca

Transformers for images

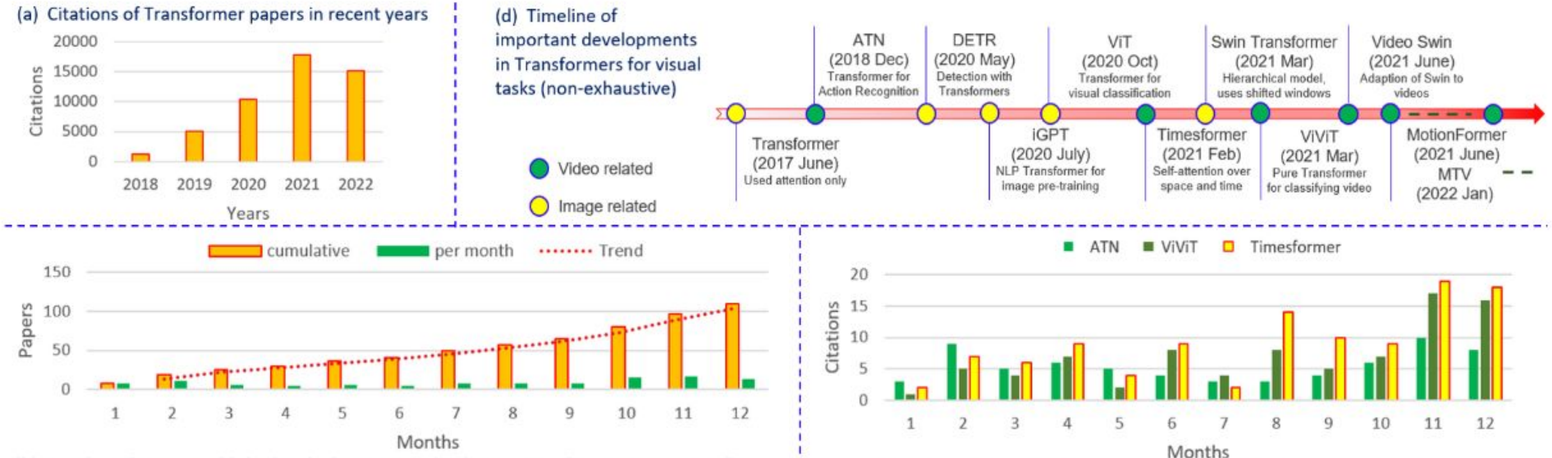
Prof. Flavio Piccoli - Dr. Mirko Paolo Barbato

What is a transformers?

Architecture based on the concept of **self-attention** to draw global dependencies between input and output, created for nlp and then extended to other fields including computer vision

The peak in the state-of-the-art for performance and publications

e.g. chatGPT is based on Transformers



(b) Number of papers published in the last 12 months that contain "Action Recognition" + ("Transformer" OR "Attention") in their titles

(c) Citations attracted by representative video Transformer publications in the last 12 months

How are transformers born?

In the Field of Natural Language Processing (NLP) to exploit the context information

Before transformers, the context was exploited using the architecture units :

- Recurrent Neural Network (RNN)
- Long short-term memory (LSTM)
- Gated Recurrent Unit (GRU)

Long-dependencies issues

Transformers use Self-Attention mechanisms to achieve Long-dependencies relationships:

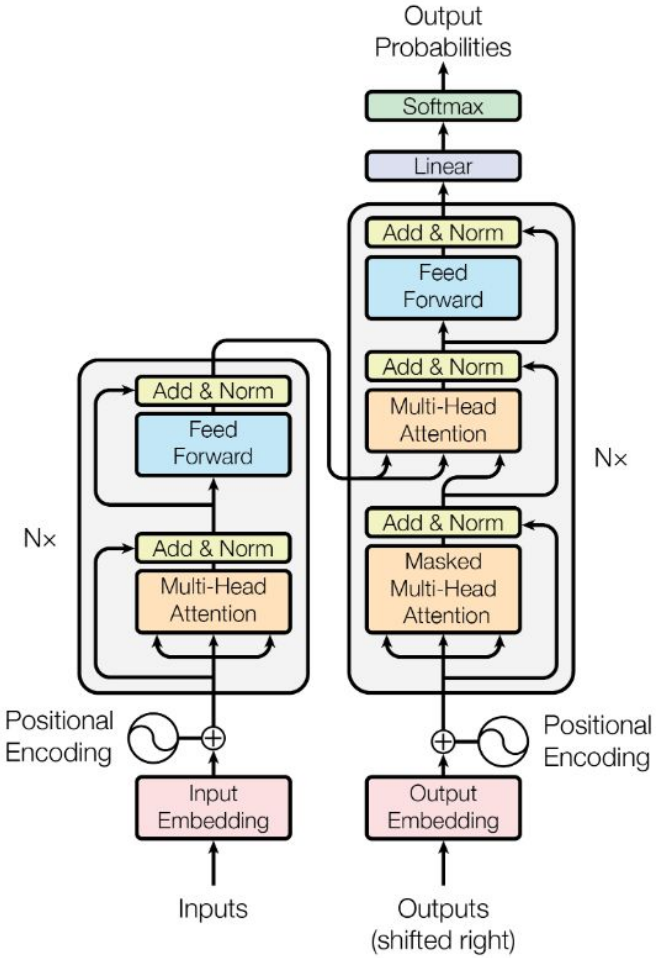
- **Self-Attention**: compute similarity scores between words in a sentence independently from their distances
- **Non-sequential**: sentences are ***not processed word by word as in RNN, LSTM, and Gated Recurrent*** creating a relationship between all the words of a sentence
- **Positional encoding**: encode information related to the specific position of a word in a sentence.

Attention is all you need?

The first Transformers consist of an Encoder-Decoder architecture for natural language translation

Encoder basic components of a Transformers:

- Input embedding
- Positional Encoding
- Transformer block:
    - Multi-Head Attention (self-attention mechanism)



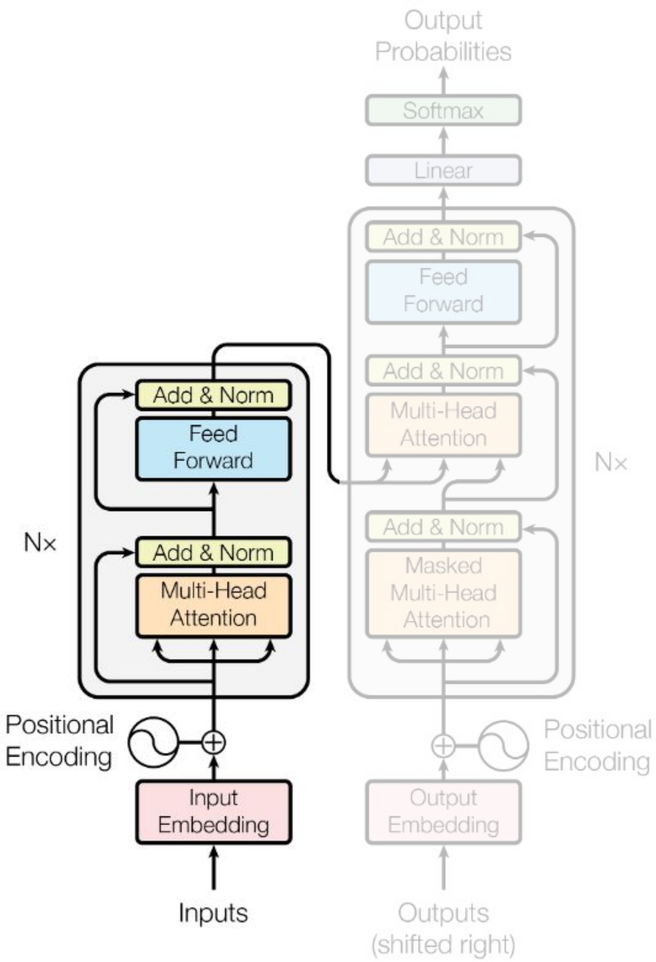
VASWANI, Ashish, et al. Attention is all you need. *Advances in neural information processing systems*, 2017, 30.

Attention is all you need?

Encoder-Decoder architecture for natural language translation

Encoder basic components of a Transformers:

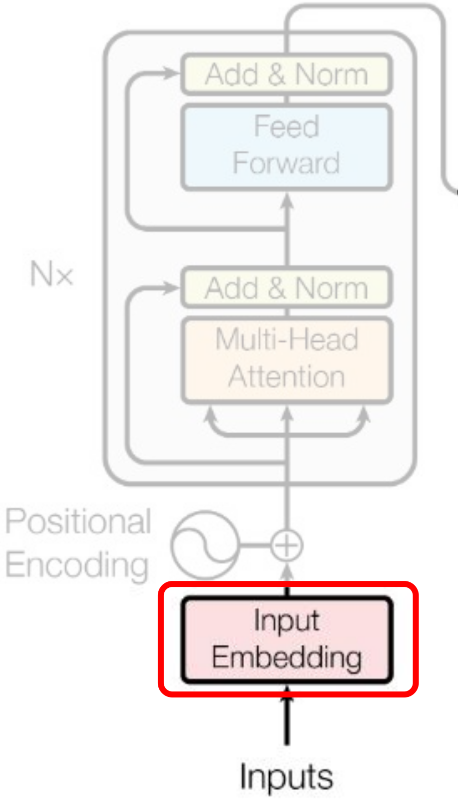
- Input embedding
- Positional Encoding
- Transformer block:
  - Multi-Head Attention (self-attention mechanism)



Transformer Encoder I

Components of a transformer encoder:

- **Input embedding**
  - The first module of a transformer encoder
  - It handles the inputs of the network and how they are initially represented
- Positional Encoding
- Transformer block:
  - Multi-Head Attention (self-attention mechanism)

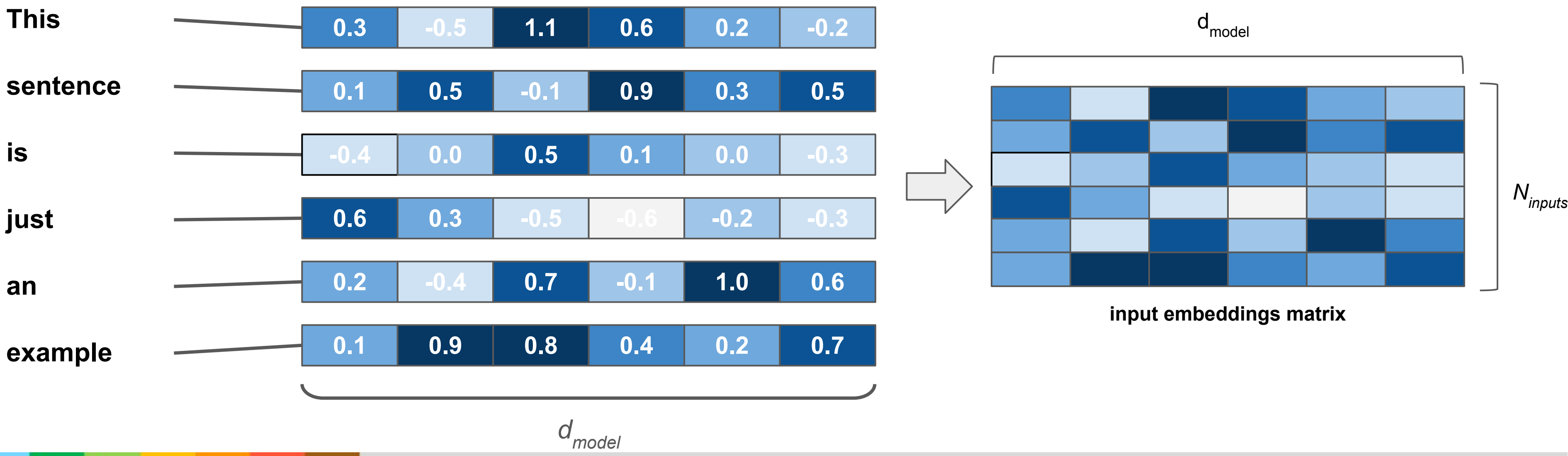


Input Embeddings

Conversion of the **inputs to vectors**:

the sequence of numbers of dimension $\boldsymbol{d_{model}}$ that represents the inputs in the space embedding.

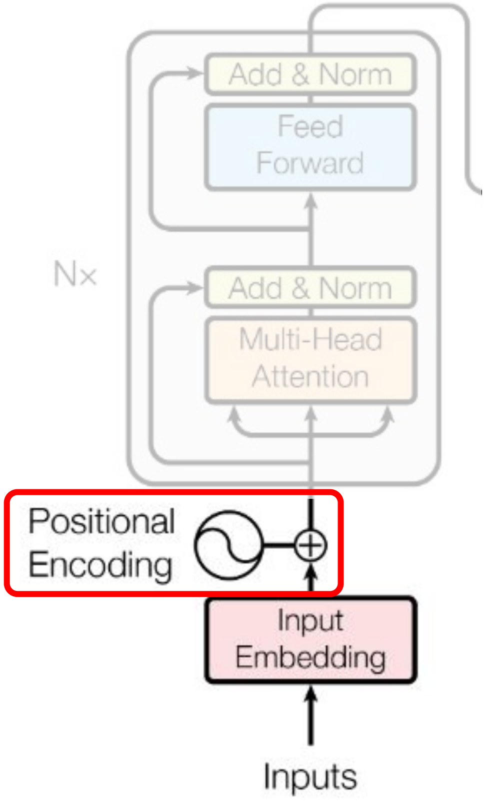
e.g.: This sentence is a just example



Transformer Encoder II

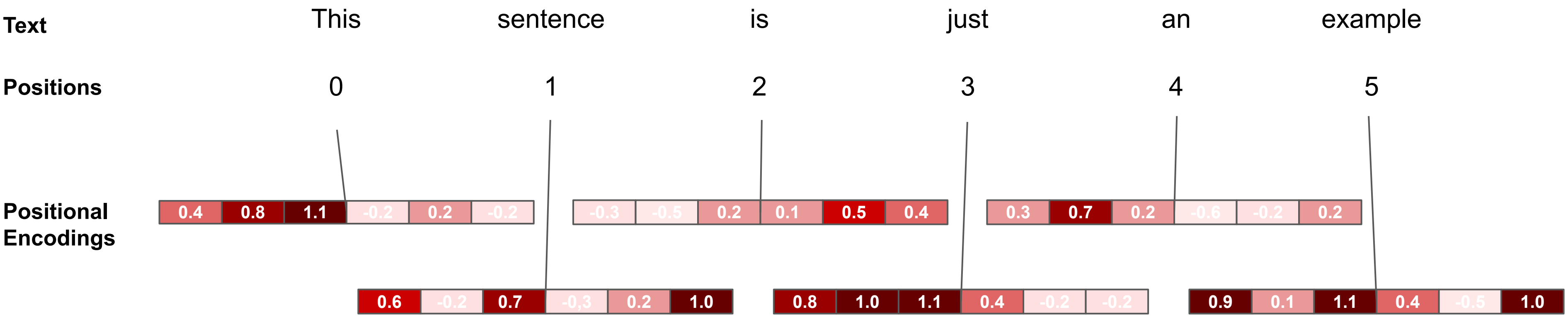
Components of a transformer encoder:

- Input embedding
- **Positional Encoding**
    - It complements the input embedding information by combining it with the information about the positions of each input
- Transformer block:
    - Multi-Head Attention (self-attention mechanism)



Positional Encoding

From **positions to vectors** of dimensions d_{model} (same as input embedding) that represents information about the **relative or absolute position**



Example: Positional Encoding

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

One way to extract the positional encoding of a word in NLP problems is to use sin and cosine to define respectively define encoding functions for even and odd values of the encoding:

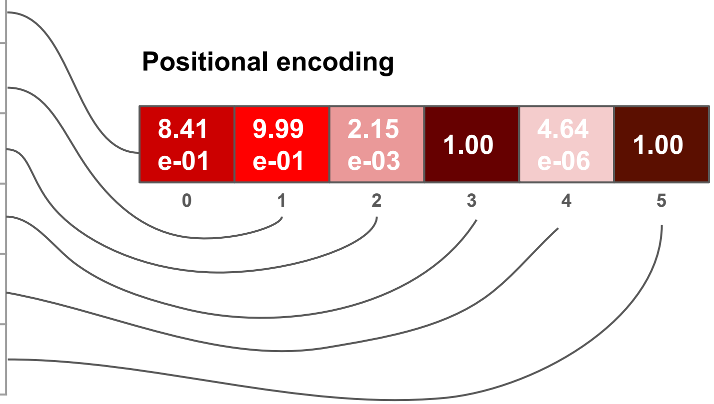
pos: index of the word

i: index in the final encoding

$\boldsymbol{d_{model}}$: dimension of the embeddings

e.g.: word “sentence” in *pos=1* with $d_{model} = 6$

Index i	PE value
0 [sin]	8.41e-01
1 [cos]	9.99e-01
2 [sin]	2.15e-03
3 [cos]	1.00
4 [sin]	4.64e-06
5 [cos]	1.00

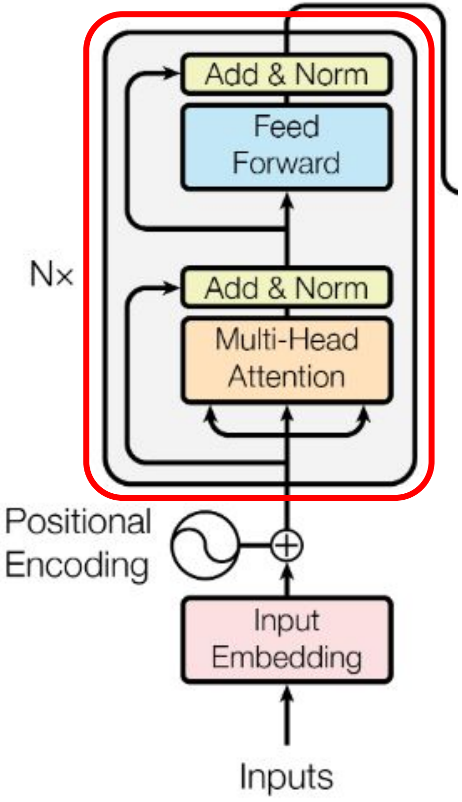


https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

Transformer Encoder III

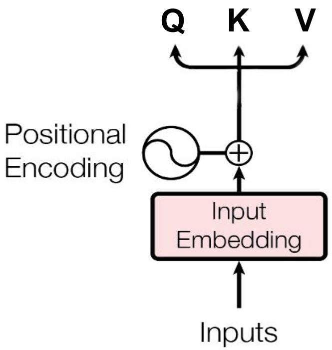
Components of a transformer encoder:

- Input embedding
- Positional Encoding
- **Transformer block**:
    - **Multi-Head Attention (self-attention mechanism)**
        - is the core of the transformers and is able to **represent long-range dependencies.**



Attention module - Inputs

The (multi-head) attention module computes the attention between a sequence of tokens and the sequence itself (self-attention)

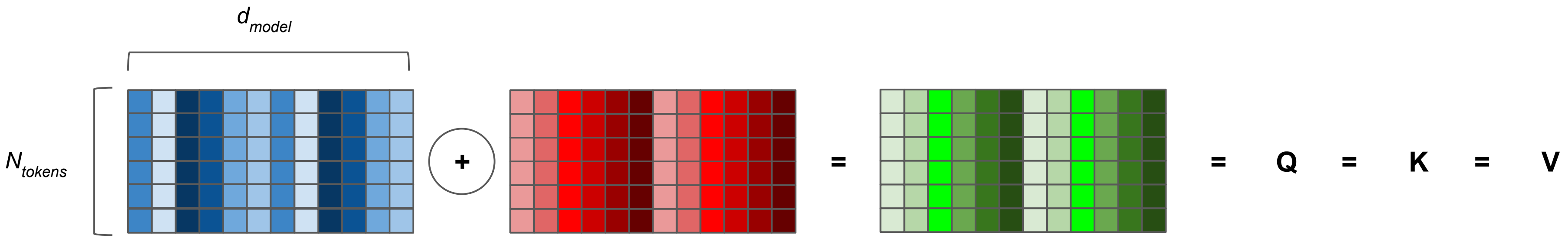


Inputs: Multi-Head attention block takes in input **3 matrices composed by tokens:**

- **Query (Q)** → (N_{tokens} , d_{model})
- **Key (K)** → (N_{tokens} , d_{model})
- **Value (V)** → (N_{tokens} , d_{model})

The **tokens** initially correspond to the **input embeddings** **added to the** **positional encodings**.

e.g.: In NLP, each token can represents a word in input. N_{tokens}= number of words

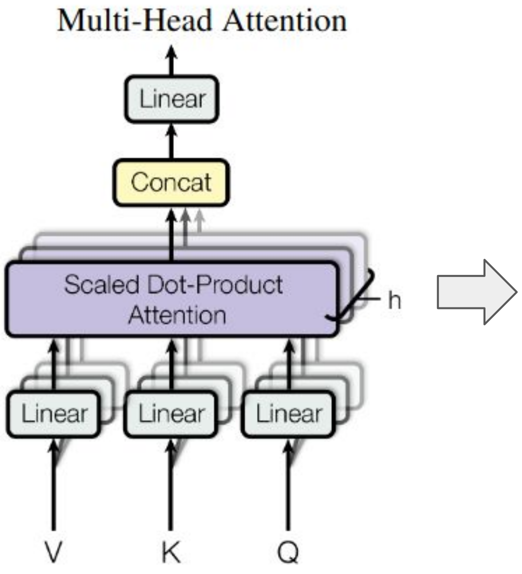


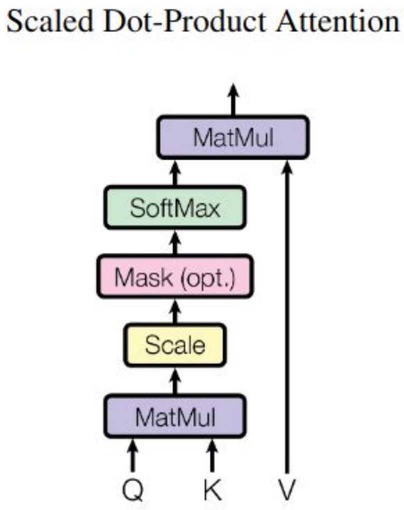
Input embeddings **Positional encodings** **Tokens**

Attention module - Scale Dot-Product I

The attention computation inside the multi-head attention block is based on the **Scaled Dot-Product module**.

Self-Attention between Q, K, and V exploits the relationship between each token with every other token to extract **new features for each token** that include the **global context information**.





Single-Head instructions:

$$Q = Linear(Q)$$
$$K = Linear(K)$$
$$V = Linear(V)$$

d_k is a scaling factor to keep softmax under control (description in the next slide)

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

Multi-Head Attention allows the network to learn **different types of features** and is highly parallelable to reduce computation time

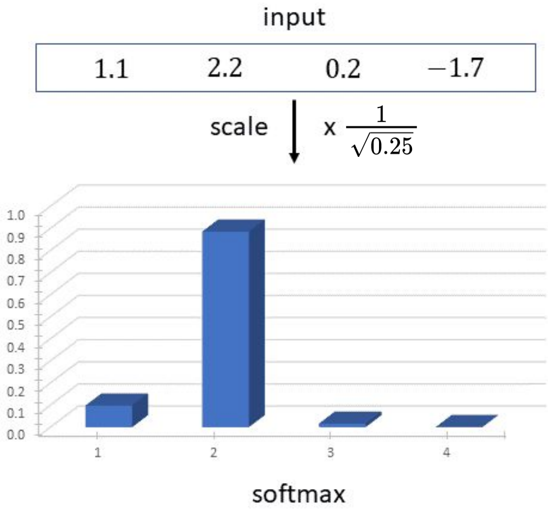
The features from each head are concatenated

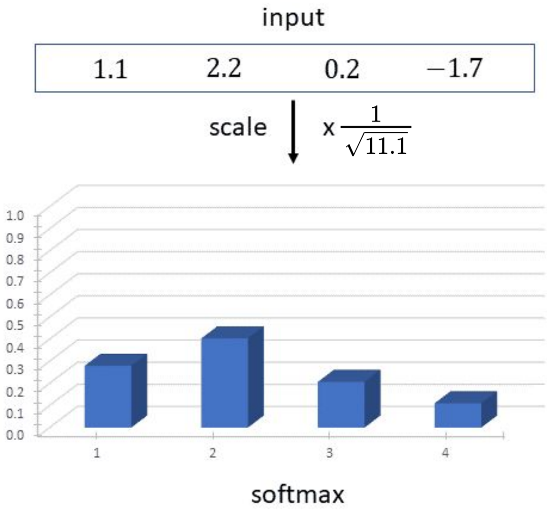
Scaling factor

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

With **large values of d_k**, the softmax tends to **flatten the weights** making the entire attention mechanism useless because every token in V would receive the same amount of attention

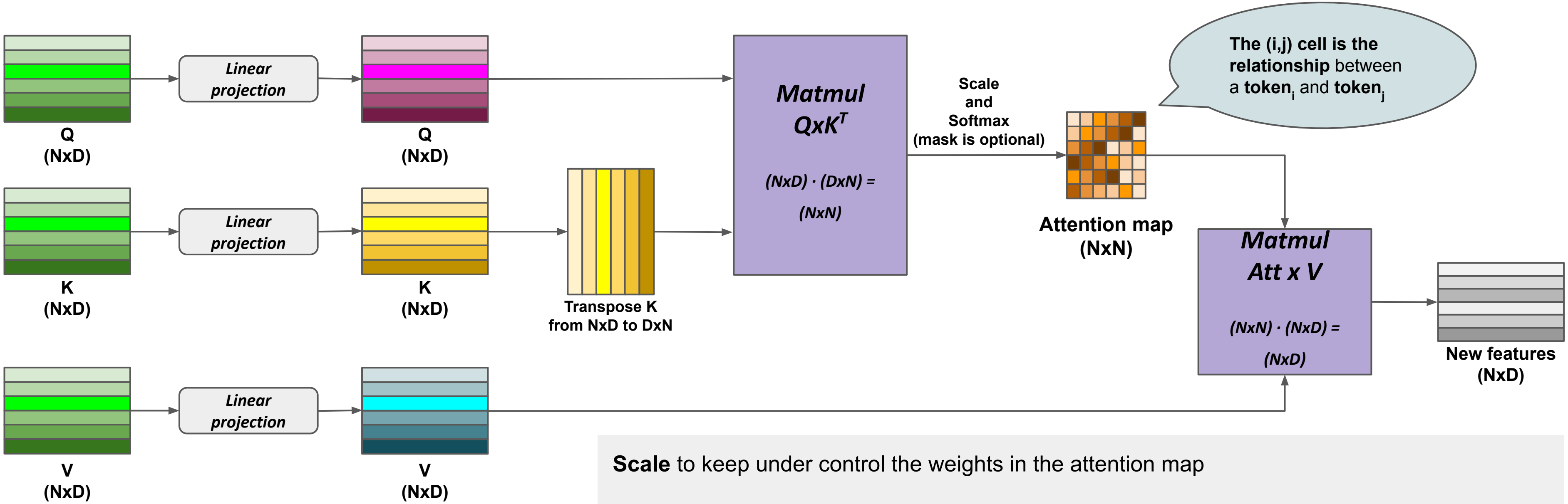
Small values of d_k can lead to **reduced performance**





Attention module - Scale Dot-Product II

N = N_{tokens}
D = d_{model}



Scale to keep under control the weights in the attention map

Softmax to convert the relationship between tokens as weights that sum 1

Mask of attention matrix is optional and it is used to eventually remove the weight of values in that we don't want to consider.
e.g.: diagonal of the attention

Visualization of attention - Example

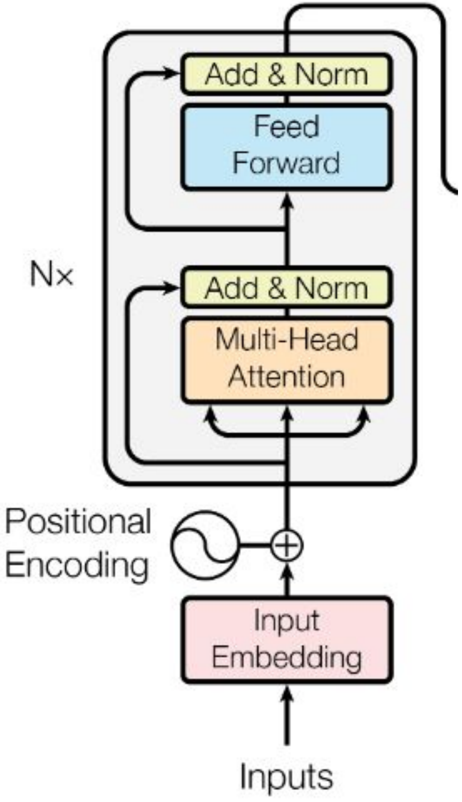
Figure 1

Figure 2

- Figures 1 and 2 represent the self-attention of two different heads from the same layer of a transformer encoder.
- the visualization shows that the two heads learn different representations and relationships between the tokens
- the attentions learn the language structures inside the sentences:
  - e.g.: relationships between subject and verbs
- the attentions learn long-range dependencies between words

Overview component of transformer block

1. Input Embedding + Positional Embedding
2. Multi-Head Attention (Q, K, V inputs)
3. Add & norm
4. Feed forward (MLP) [same d_{model}]
5. Add & norm again
6. Repeat from 2 for how many blocks you want



Vision Transformer

How to adapt transformer to images?

Problem:

How to transform an image into a sequence of tokens?

How to adapt transformer to images?

Problem:

How to transform an image into a sequence of tokens?

- Using pixels as the words of a sentence?
    - The **computational complexity would be too high** for training a model
        - if and image has n x m dimensions, the computational complexity would be n x m

How to adapt transformer to images?

Problem:

How to transform an image into a sequence of tokens?

- Using pixels as the words of a sentence?
  - The **computational complexity would be too high** for training a model
    - if and image has n x m dimensions, the computational complexity would be n x m

Solution:

- Divide the image in **patches**!!!

Vision Transformer (ViT)

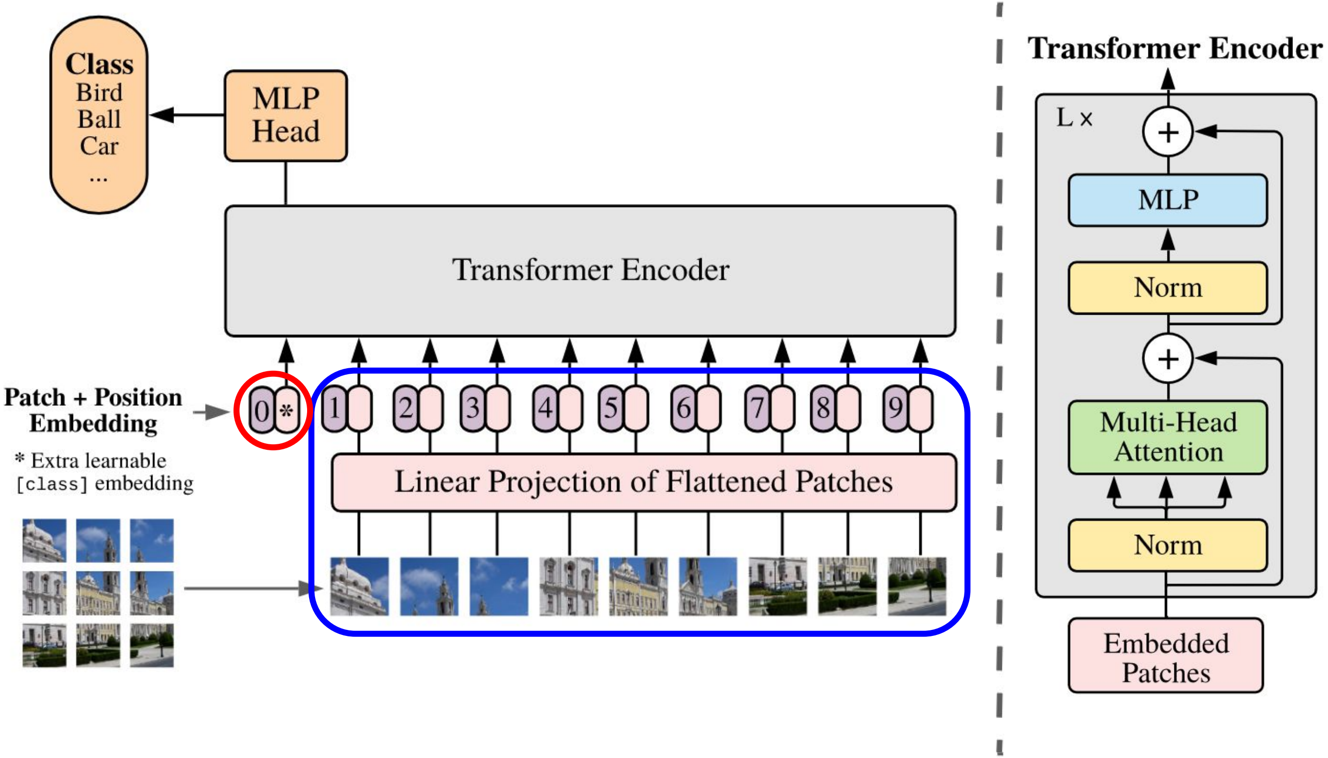
Transformer architecture for **image classification**

New elements in ViT**:**

- **Patch embedding**
- **Class token**

The **Transformer Encoder** is similar to the standard transformer encoder used

Multi-Head attention module works exactly as the original one



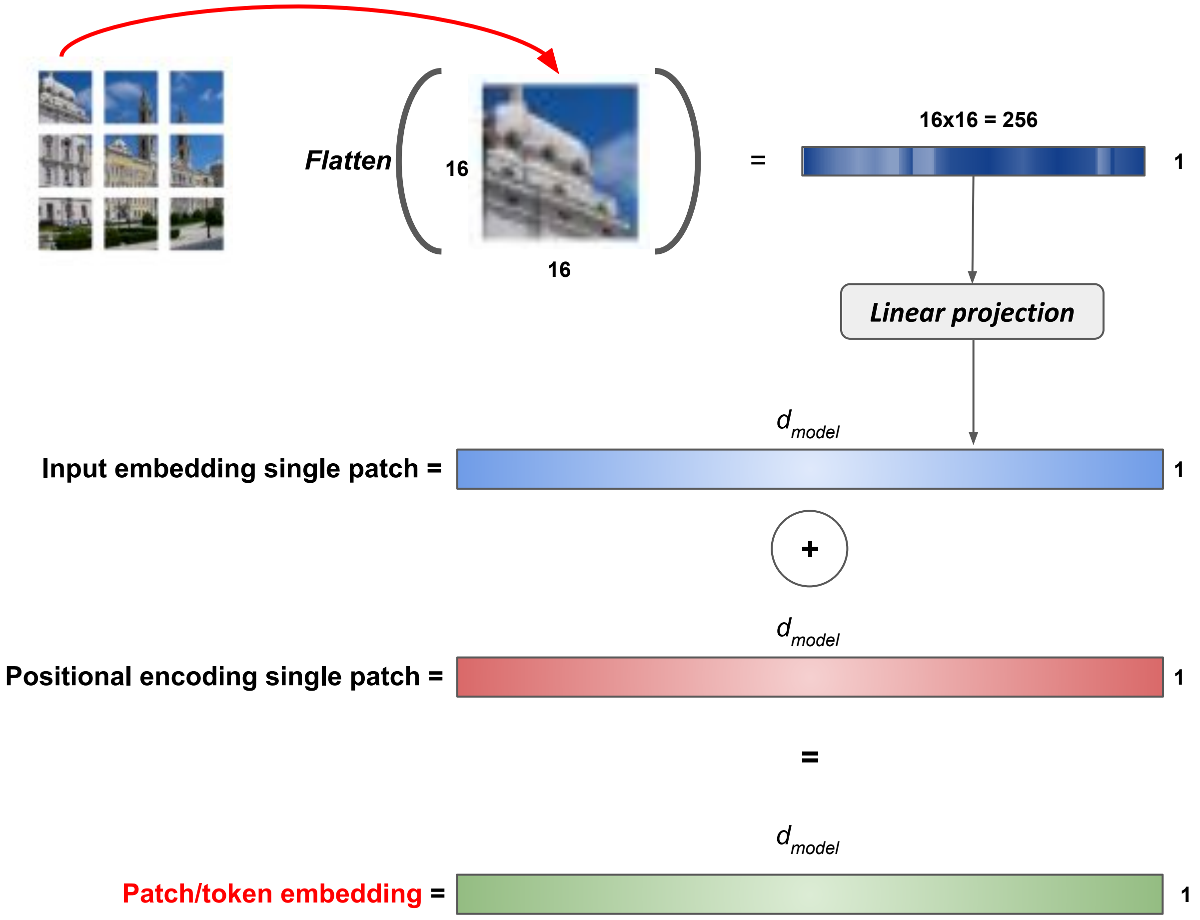
DOSOVITSKIY, Alexey, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Patch embeddings

Input embedding: linear projections to vectors of d_{model} elements of the flattened patches

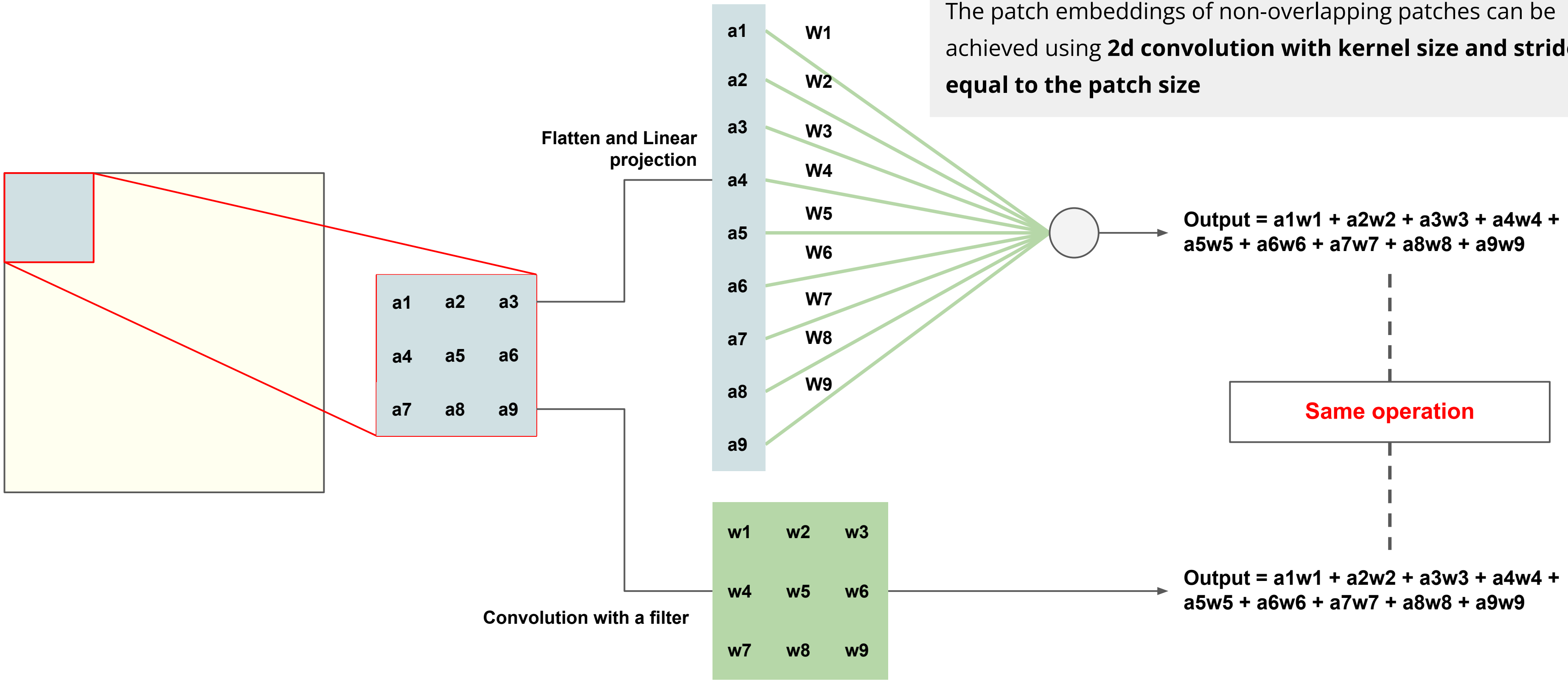
e.g.: the original version uses patches of 16x16 pixels

Positional encoding: d_{model} parameters **learned by the network** during the training for each of the patch



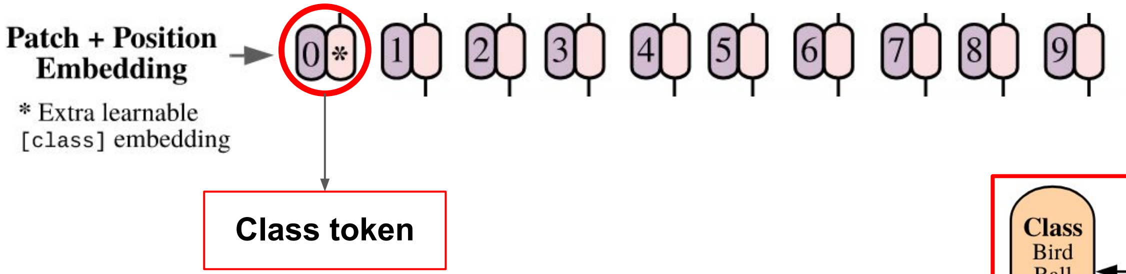
How to simply compute Patch Embedding?

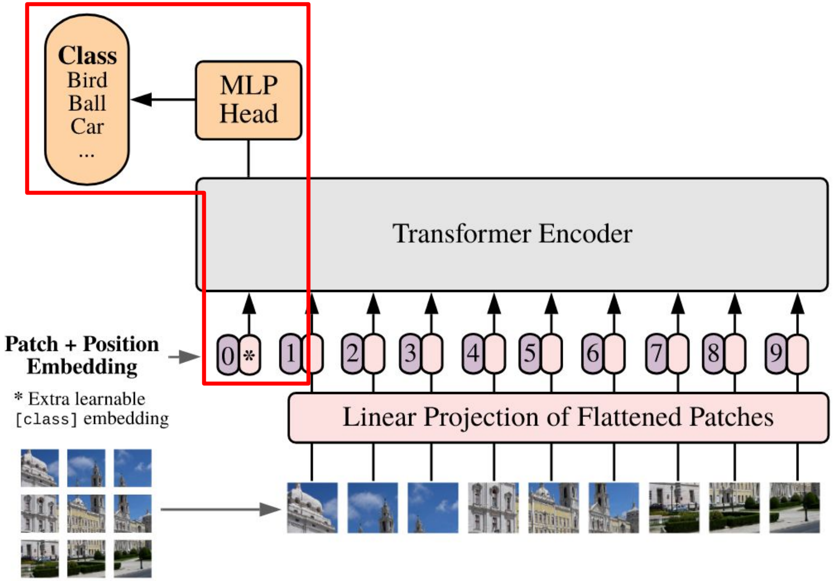
The patch embeddings of non-overlapping patches can be achieved using **2d convolution with kernel size and stride equal to the patch size**



Class token

A **learnable embedding** of dimension d_{model} added as input together with the sequence of embedded patches





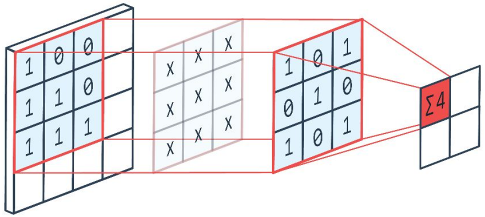
The **scope** of the class token is to become a **representation of all the content of the image** after the transformer Encoder exploits the relationship between the class token and all the other patches

Only the class token feature extracted from the transformer encoder **is used for the classification**

Transformer vs CNN (global vs local)

CNN

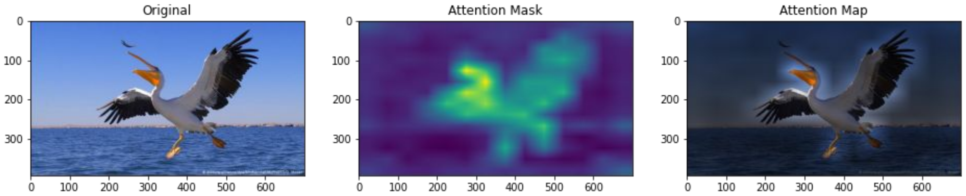
The convolutional layers use **limited receptive fields** thus taking into consideration only the **local information**.



To include information about all the images, it is necessary to make the architectures deeper and deeper till the receptive field is big enough to see all the images.

Transformer

The Transformer Encoders inputs are all the patches of an image as tokens, and the encoders compute the relationship between all of them, thus taking in consideration the **global information**.



Exactly as for NLP, the transformer for images learns the “long-distance relationship” **independently from the distance between patches.**

Swin Transformer

Shifted Windows Transformer (Swin)

ViT problems:

- **not suitable for dense vision tasks**
  - low-resolution feature maps ("big" patches)
  - e.g.: object detection and segmentation segmentation
- **unfeasible** when the input image **resolution is high**
  - higher resolutions need more patches
  - **computational complexity is $O(N^2)$** where N is the number of patches (dot production between tokens)

Swin is a Hierarchical Vision Transformer that uses a particular mechanism of **Shifted Windows**:

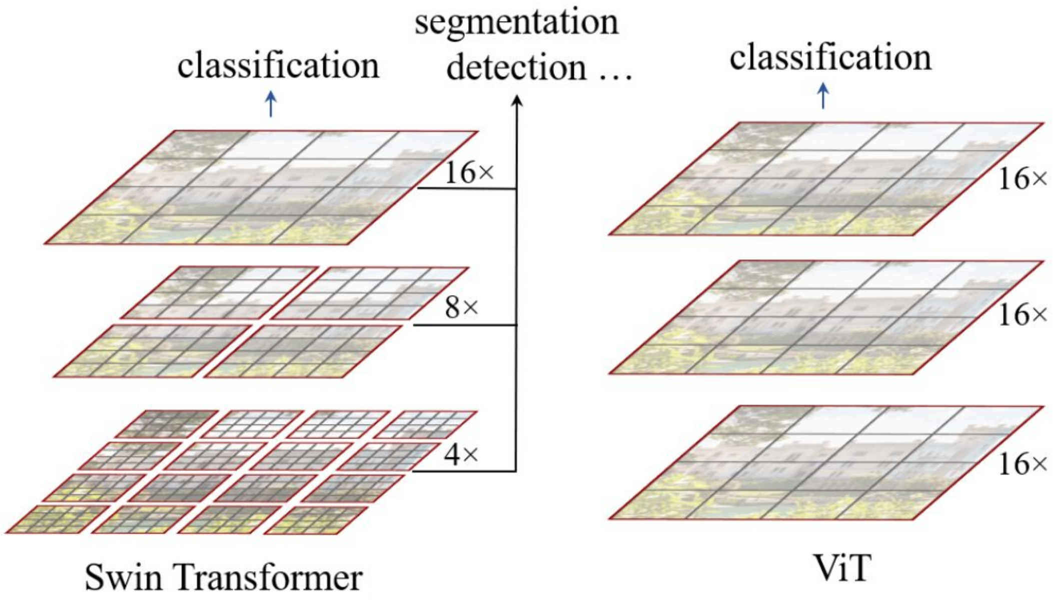
- **suitable for any vision tasks** including dense tasks
- linear computational complexity → **$O(N)$** where N is the number of patches

Hierarchical Architecture with Windows

ViTs typically use patches of 16x16 pixels on a regular grid, keeping them constant inside the network and computing the self-attention considering all the patches together.

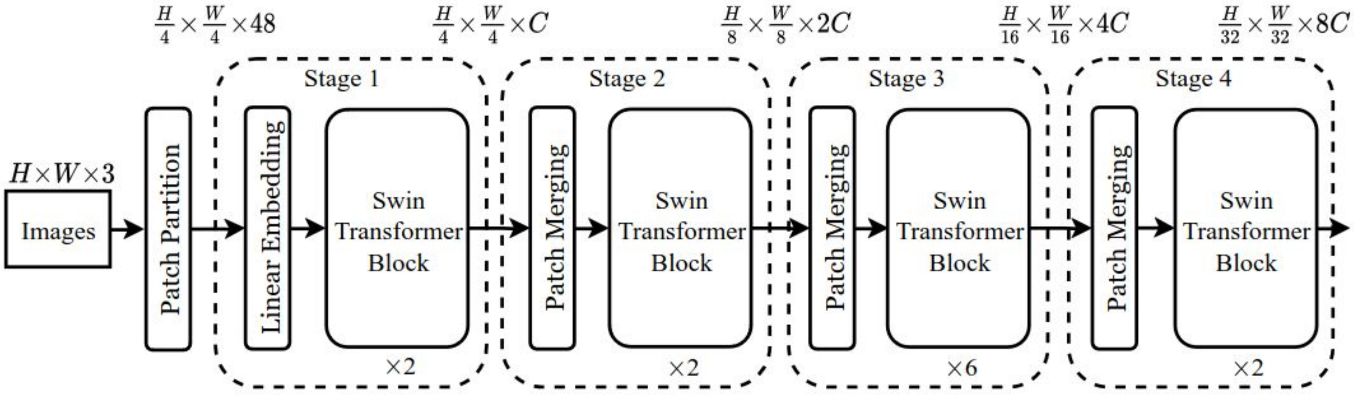
Swin starts with patches of 4x4 pixels and increases the dimension of the patches inside the architecture. → **low-resolution problem.**

To maintain the **complexity linear**, **windows** (group patches together) are used to compute attention locally only between the patches inside the same window. Layer after layer **the windows are merged to achieve the knowledge on a global scale**.



LIU, Ze, et al. Swin transformer: Hierarchical vision transformer using shifted windows. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021. p. 10012-10022.

General Architecture



Architecture

The architecture is composed of **4 stages** where:

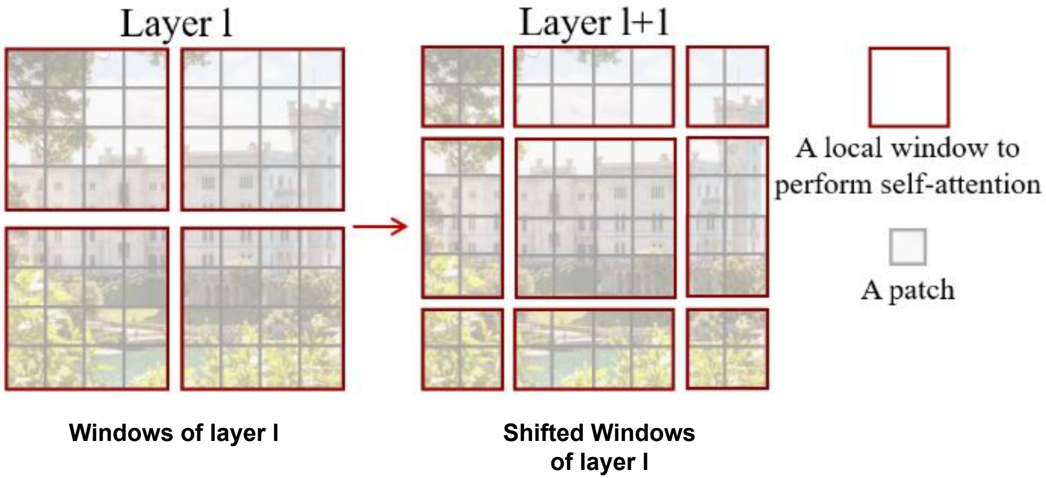
- Stage 1
  - starts from flattened patches of 4x4 pixels (48 because it considers rgb images → 16 pixels per 3 channels)
  - creates the tokens using Linear embedding and applies Swin Transformer Blocks to compute self-attention inside the windows
- Stages 2, 3, and 4 consist of a **Patch Merging** module that increases the dimension of the patches, **followed by Swin Transformer Blocks** to compute self-attention inside the windows

Shifted Window

There are **two types of Swin Transformer Blocks** that differ in what kind of windows they use.

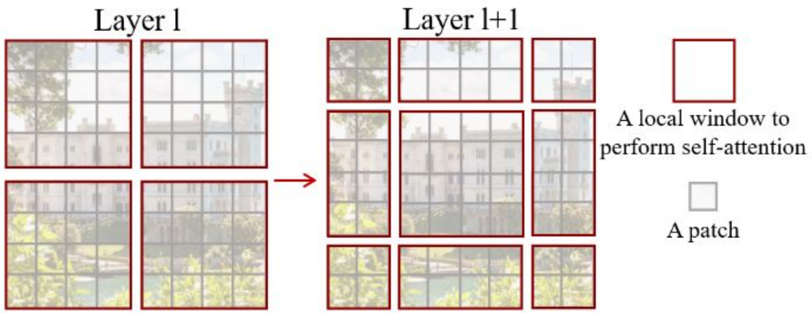
The first type of windows structure is a **simple grid**.

The second type of windows structure (**shifted windows**) is used to explicit the relationship between patches on the boundaries of the windows that cannot be coupled in the first configuration.



Efficient Computation of the Shifted Window

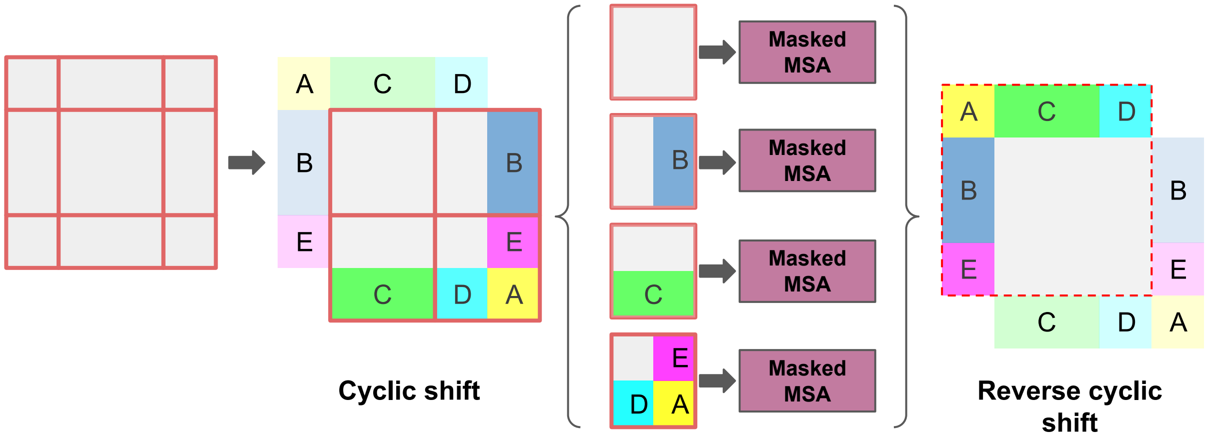
The **shifted-windows configuration is composed of more batches of patches**, one more for the rows and one more for the columns → **computation is more complex**



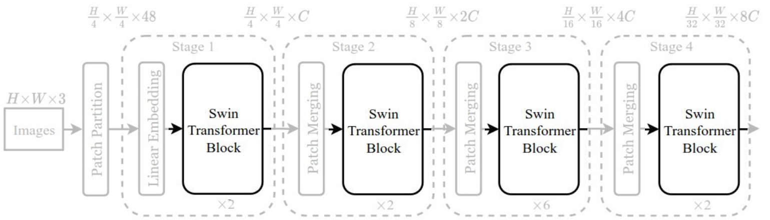
4 batches **9 batches**

How to make it computationally efficient? Cyclic-shifting loop

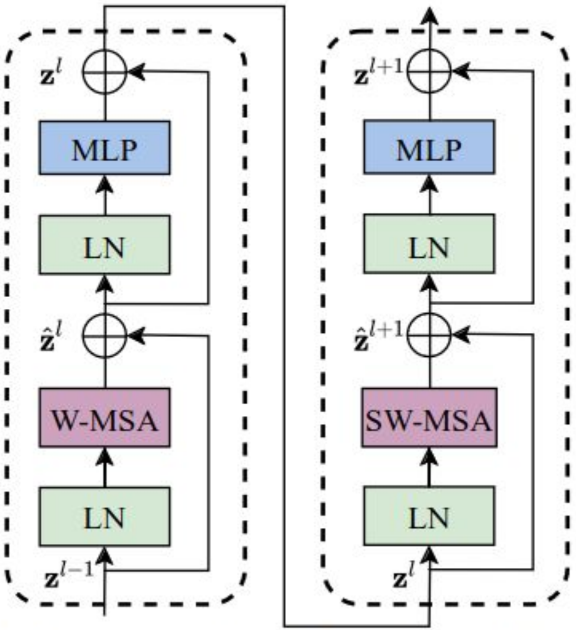
1. **shift the top-left batches** (A, B, and C) **to the bottom-right direction**, making it possible to divide the image into **4 batches** (like grid division)
2. **mask attention** for the batches that include non-adjacent patches to not mix them



Swin Transformer Block



Architecture



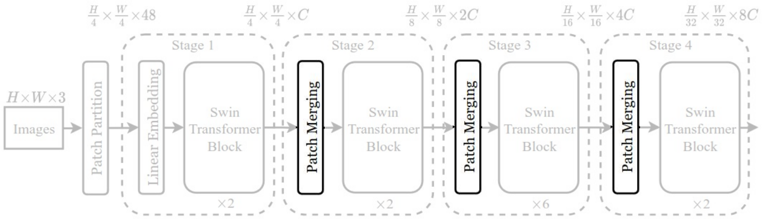
Two Successive Swin Transformer Blocks

Swin Transformer Blocks are always composed of **two consequential Swin Transformers Blocks** where the two windows schemes are alternated:

1. grid scheme
2. shifted windows scheme

All the other components of a Swin Transformer Block are the same as ViT Encoder.

Patch Merging



Architecture

The Patch Merging layers **reduce the number of tokens** at the beginning of Stages 2, 3, and 4.

The Patch Merging layer consists of:

1. **concatenation of the features** of each group of 2 × 2 neighboring patches (4 patches of embedding dimension C)
2. application of a **linear layer** on the 4C-dimensional concatenated features with an output dimension of 2C.

This reduces the number of tokens by a multiple of 2 × 2 = 4 (2× downsampling of resolution) at each stage.

Considering an image of HxW pixels and that Swin starts with patches of 4x4 pixels, at the end of each stage the number of tokens is:

Stage 1		Stage 2		Stage 3		Stage 4
$\frac{H}{4} \times \frac{W}{4} \times C$	$\Rightarrow$	$\frac{H}{8} \times \frac{W}{8} \times 2C$	$\Rightarrow$	$\frac{H}{16} \times \frac{W}{16} \times 4C$	$\Rightarrow$	$\frac{H}{32} \times \frac{W}{32} \times 8C$

Comparison with other models

Classification

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
(a) Regular ImageNet-1K trained models					
RegNetY-4G [48]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224^2	84M	16.0G	334.7	82.9
EffNet-B3 [58]	300^2	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380^2	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456^2	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528^2	43M	19.0G	96.9	84.0
EffNet-B7 [58]	600^2	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384^2	307M	190.7G	27.3	76.5
DeiT-S [63]	224^2	22M	4.6G	940.4	79.8
DeiT-B [63]	224^2	86M	17.5G	292.3	81.8
DeiT-B [63]	384^2	86M	55.4G	85.9	83.1
Swin-T	224^2	29M	4.5G	755.2	81.3
Swin-S	224^2	50M	8.7G	436.9	83.0
Swin-B	224^2	88M	15.4G	278.1	83.5
Swin-B	384^2	88M	47.0G	84.7	84.5

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
(b) ImageNet-22K pre-trained models					
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3

ADE20K Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-		
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large‡	50.3	61.7	308M	-	-
UperNet	DeiT-S†	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B‡	51.6	-	121M	1841G	8.7
UperNet	Swin-L‡	**53.5**	**62.8**	234M	3230G	6.2

Segmentation

It **outperforms ViT for classification** tasks on ImageNet using **fewer parameters**

It **outperforms other models in the segmentation** tasks, being **more flexible than ViT**

Thank you!
Any Questions?