

UiO : Department of Informatics
University of Oslo

IN4310 Deep Learning for Image Analysis

Lecture 15: Vision Transformers

May 11, 2023

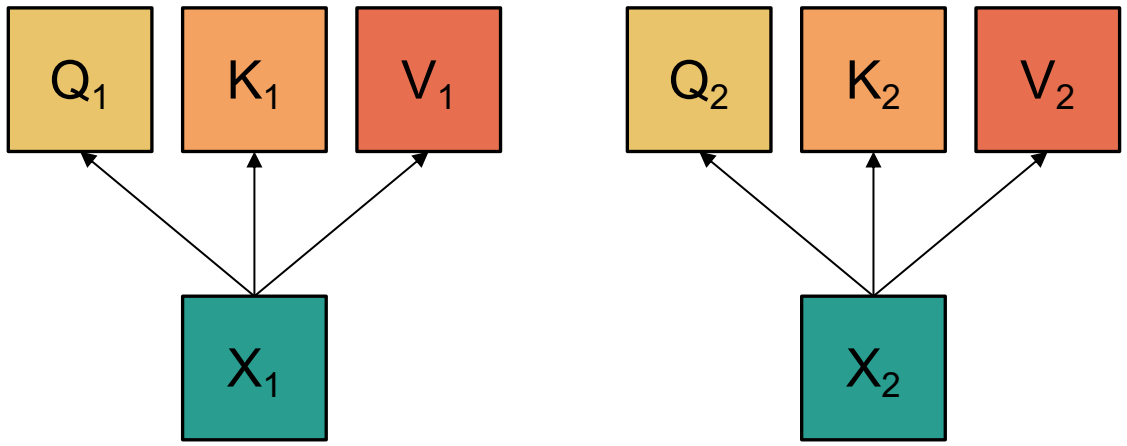
Dhananjay Tomar

Outline

- **Self-attention**
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- Transformer encoder block
- Transformer decoder block
- Vision transformer (ViT)
- Swin transformer
- DEtection TRansformer (DETR) for object detection
- Suggested resources

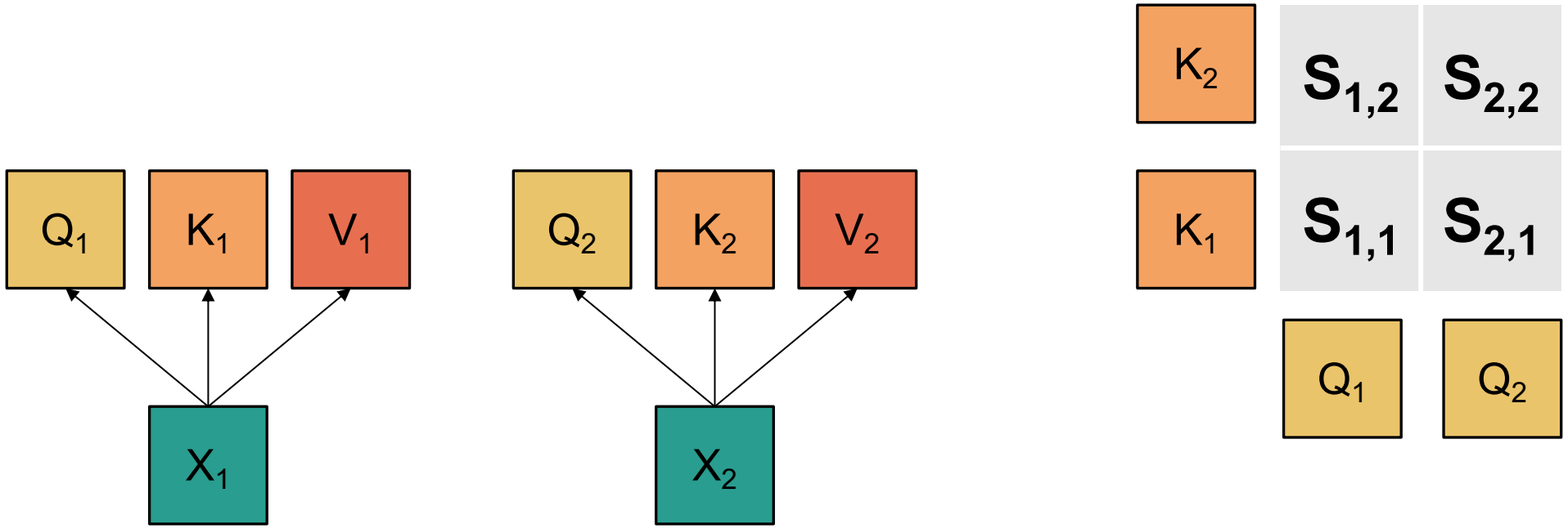
Self-Attention: Step 1: Compute Query, Key, Value vectors

- In step 1, each input vector X_i is multiplied with 3 different matrices to get query, key, and value vectors.
- You can imagine this as having 3 **`torch.nn.Linear`** layers:
  - L_Q = `torch.nn.Linear(size_of_x_vector, size_of_query_vector)`
  - L_K = `torch.nn.Linear(size_of_x_vector, size_of_key_vector)`
  - L_V = `torch.nn.Linear(size_of_x_vector, size_of_value_vector)`
  - $Q = L_Q(X)$ $K = L_K(X)$ $V = L_V(X)$
- Query, key, and value vectors typically have the same dimensions.



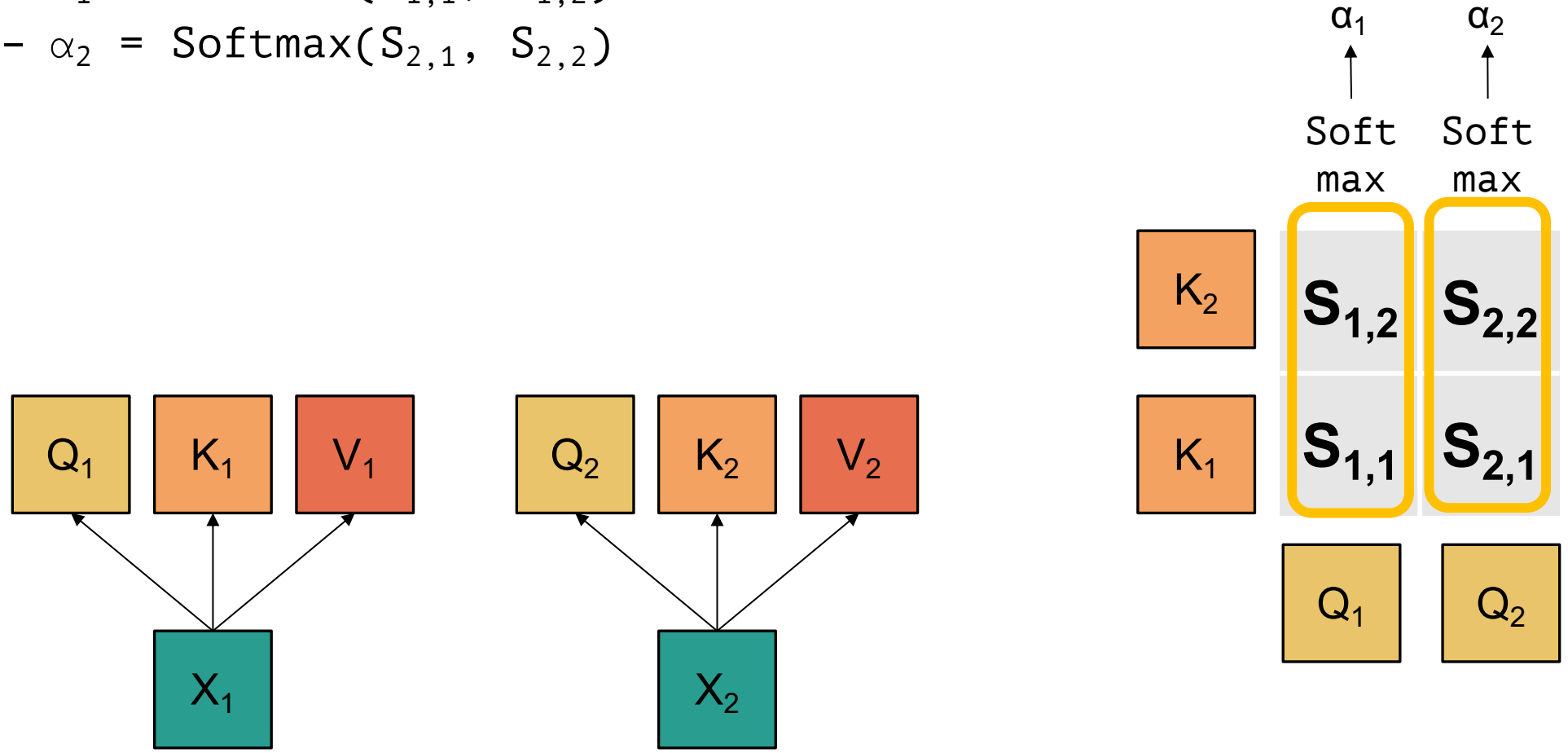
Self-Attention: Step 2a: Compute score

- Apply *scaled dot product attention* using query and key vectors to get scores for each pair.
- Each K_i, Q_i pair generates a score.
  - $S_{1,1}$ = dot_product(Q_1, K_1) $S_{1,2}$ = dot_product(Q_1, K_2) $S_{2,1}$ = dot_product(Q_2, K_1) $S_{2,2}$ = dot_product(Q_2, K_2)
- Each score is divided by the square root of the dimension of the key vector.
  - $S_{i,j}$ = $S_{i,j}$ / square_root(dimension of K_i vectors)
  - This is done to stabilise gradients during backpropagation.



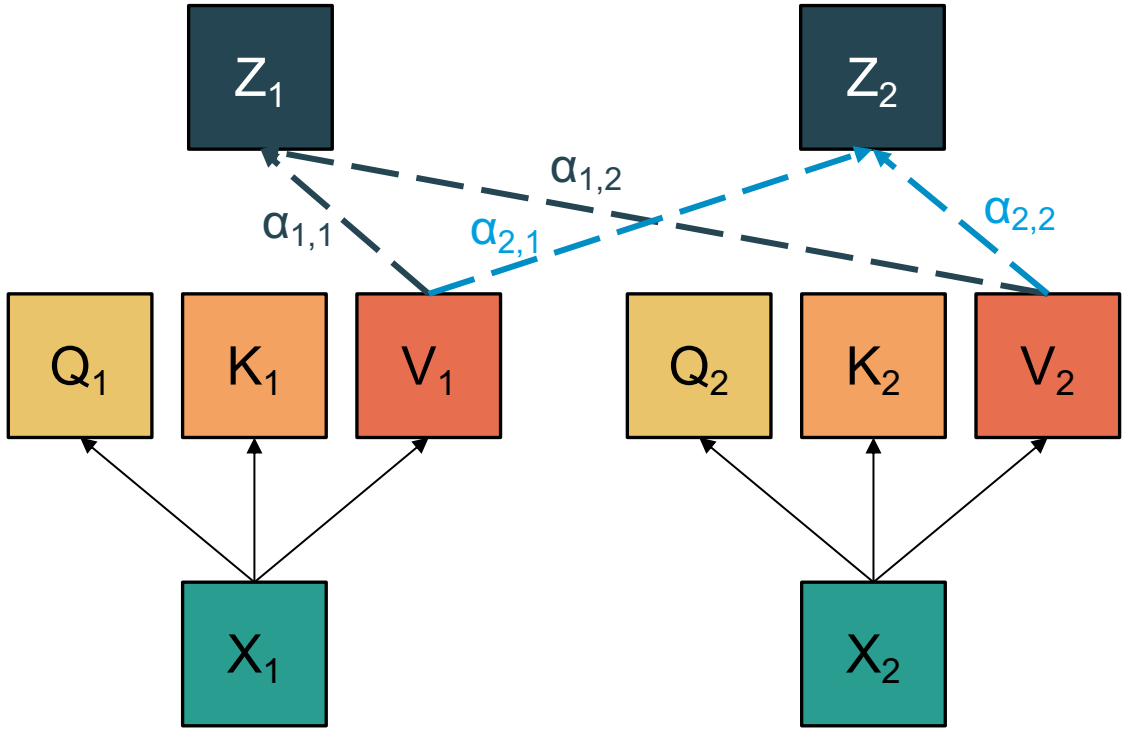
Self-Attention: Step 2b: Compute softmax score

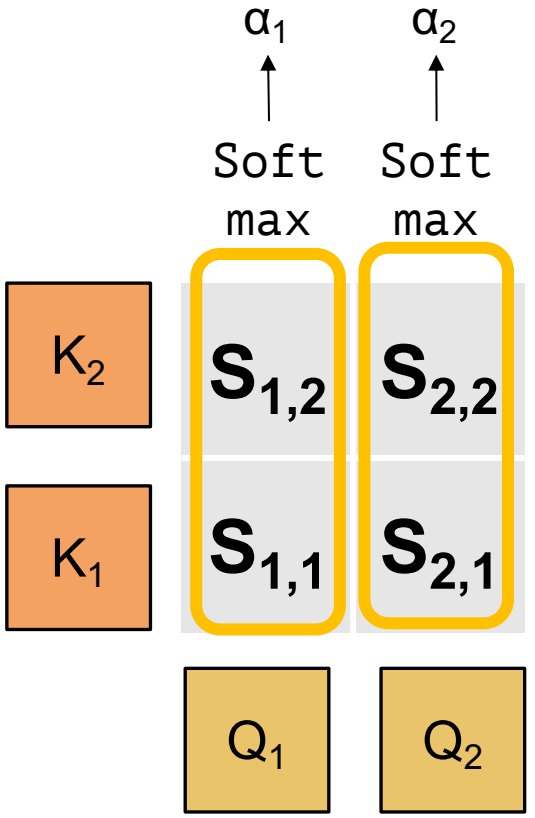
- Next, scores corresponding to each query vector are passed through a softmax layer to get softmax scores.
  - α_1 = Softmax($S_{1,1}$, $S_{1,2}$)
  - α_2 = Softmax($S_{2,1}$, $S_{2,2}$)



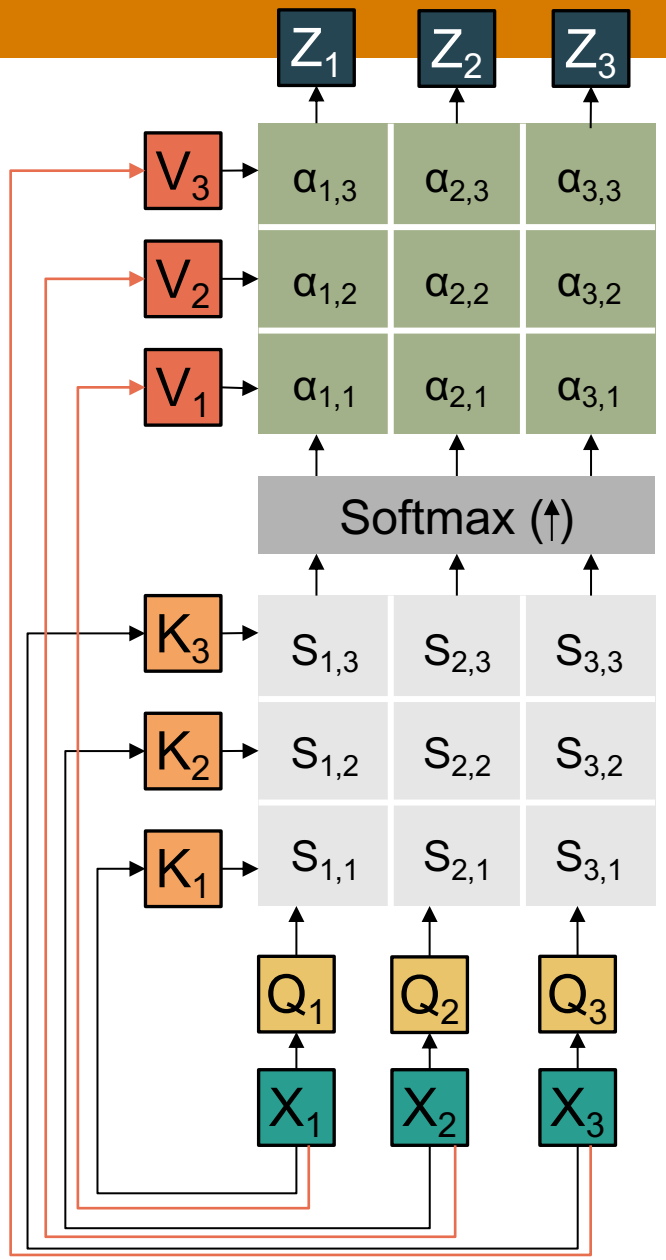
Self-Attention: Step 3: Weighted average of value vectors

- For every X_i, compute the weighted average of the value vectors using the softmax scores (corresponding to X_i) as weights.
  - $Z_1 = (\alpha_{1,1} * V_1) + (\alpha_{1,2} * V_2)$
  - $Z_2 = (\alpha_{2,1} * V_1) + (\alpha_{2,2} * V_2)$





Self-Attention: Overview



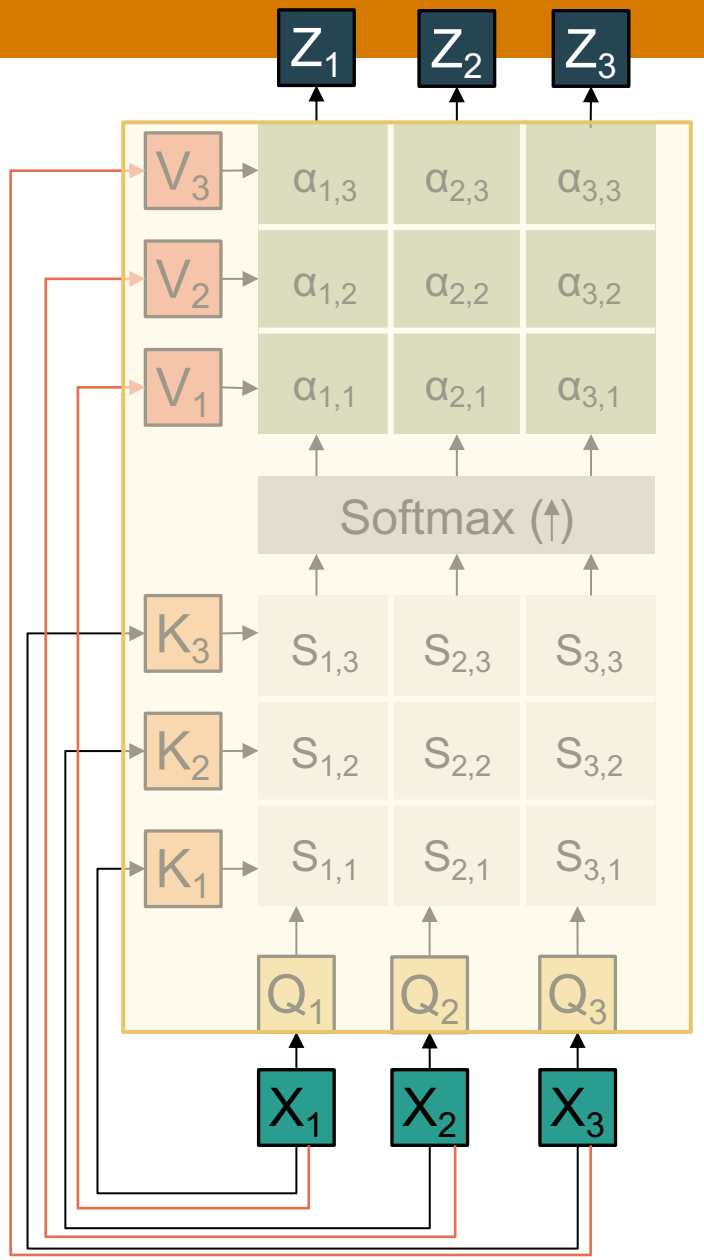
$$Z_1 = (\alpha_{1,1} * V_1) + (\alpha_{1,2} * V_2) + (\alpha_{1,3} * V_3)$$

$$Z_2 = (\alpha_{2,1} * V_1) + (\alpha_{2,2} * V_2) + (\alpha_{2,3} * V_3)$$

$$Z_3 = (\alpha_{3,1} * V_1) + (\alpha_{3,2} * V_2) + (\alpha_{3,3} * V_3)$$

Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$
Value vectors: $V = XW_V$
Scores: $S_{i,j} = K_i \bullet Q_j / \sqrt{D}$

Self-Attention: Overview

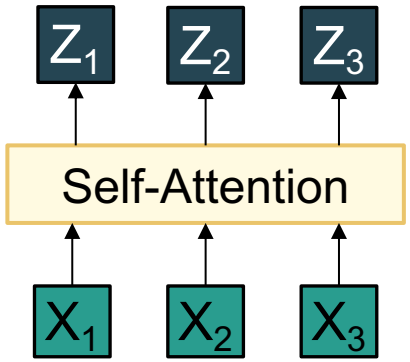


$$Z_1 = (\alpha_{1,1} * V_1) + (\alpha_{1,2} * V_2) + (\alpha_{1,3} * V_3)$$

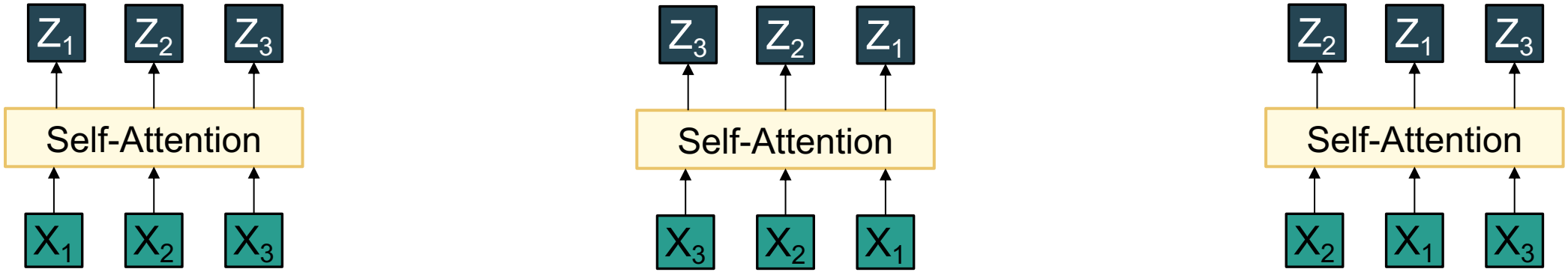
$$Z_2 = (\alpha_{2,1} * V_1) + (\alpha_{2,2} * V_2) + (\alpha_{2,3} * V_3)$$

$$Z_3 = (\alpha_{3,1} * V_1) + (\alpha_{3,2} * V_2) + (\alpha_{3,3} * V_3)$$

Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$
Value vectors: $V = XW_V$
Scores: $S_{i,j} = K_i \bullet Q_j / \sqrt{D}$



Self-Attention: Permutation equivariance

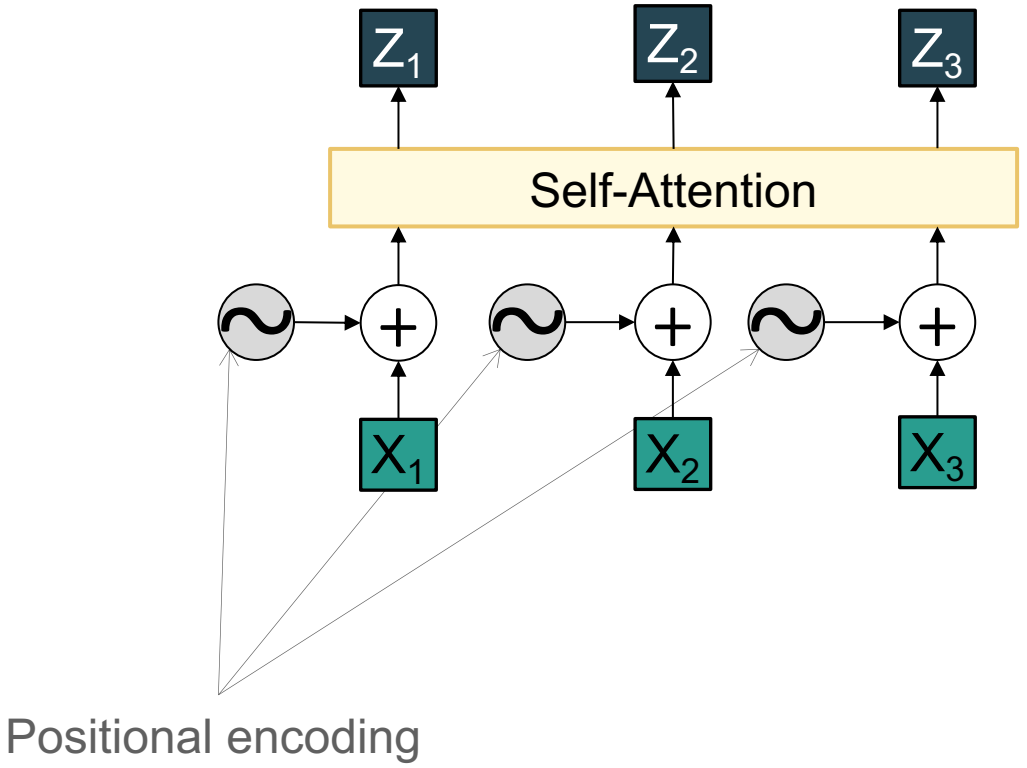


- Self-attention layer is permutation equivariant.
- Self-attention layer doesn't care about the order of the input sequence.

Problem: How to encode the order of input sequence when it matters like in language or images?

Positional encoding

- There are multiple ways of encoding position.
  - Absolute/fixed positional encoding.
  - Relative positional encoding.
  - A hybrid of relative and absolute positional encoding.
  - Learned positional encoding.
  - etc.
- Positional encodings are vectors with the same dimension as the input vectors.
- Positional encodings are usually added to the input vectors.



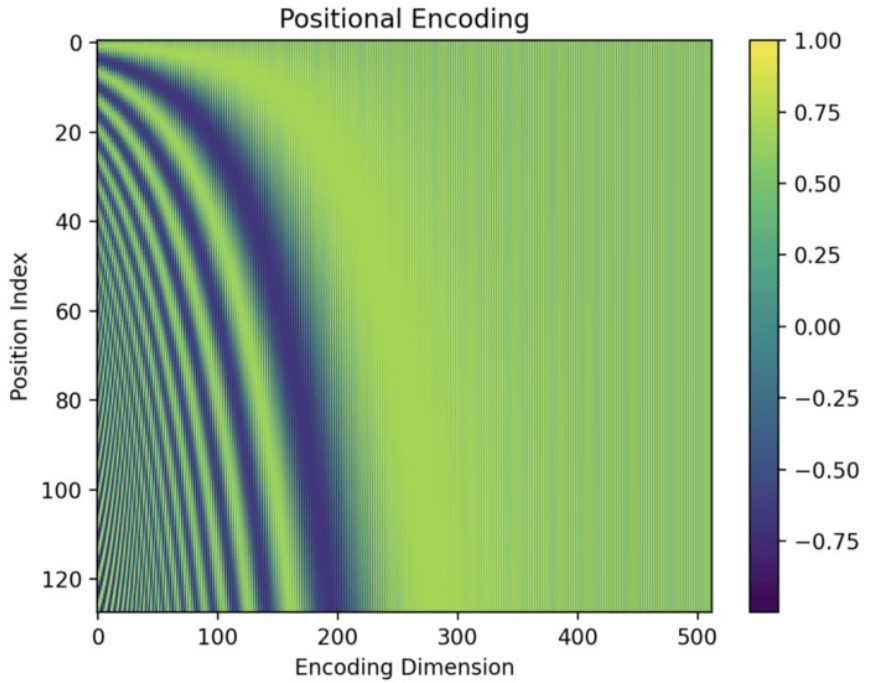
Positional encoding: Absolute positional encoding

- Absolute positional encoding at position ***pos*** for dimension **i = 0...(d/2 - 1)**:

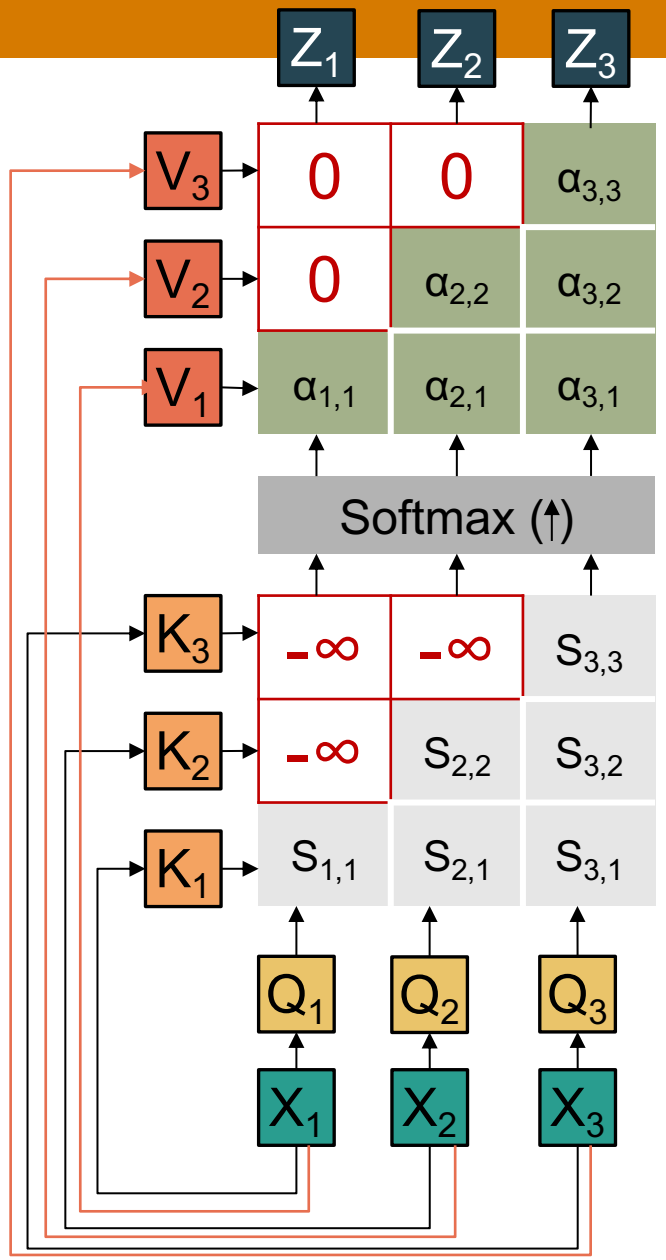
$$PE(pos, 2i) = \sin(pos/10000^{2i/d})$$
$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d})$$

- A 512-dimensional encoding for ***i*** in the sequence:

$\sin(i/10000^{0/512})$	$cos(i/10000^{0/512})$	$\sin(i/10000^{2/512})$	$\cos(i/10000^{2/512})$	…	$\sin(i/10000^{510/512})$	$\cos(i/10000^{510/512})$



Masked self-attention



$Z_1 = (\alpha_{1,1} * V_1)$

$Z_2 = (\alpha_{2,1} * V_1) + (\alpha_{2,2} * V_2)$

$Z_3 = (\alpha_{3,1} * V_1) + (\alpha_{3,2} * V_2) + (\alpha_{3,3} * V_3)$

Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$
Value vectors: $V = XW_V$
Scores: $S_{i,j} = K_i \bullet Q_j / \sqrt{D}$

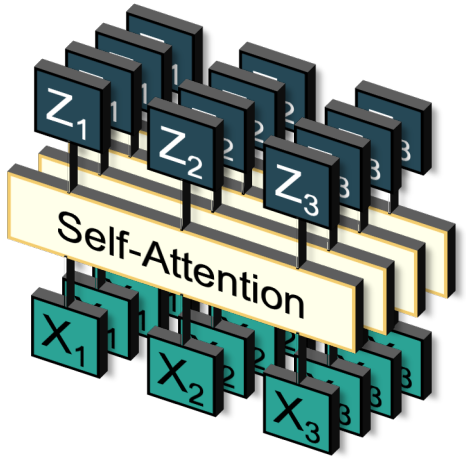
- Don't let vectors look ahead in the sequence at future vectors.
- Manually set scores to -infinity.
- Specially relevant for Language Modelling

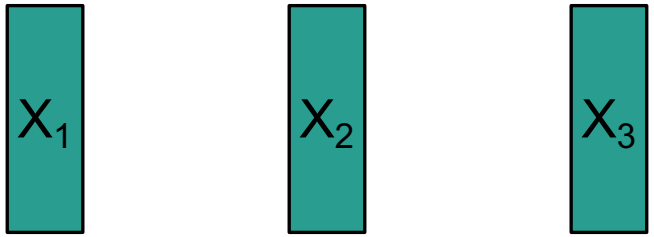
Outline

- Self-attention
  - Positional encoding
  - Masked self-attention
- **Multi-head self-attention**
- Transformer encoder block
- Transformer decoder block
- Vision transformer (ViT)
- Swin transformer
- DEtection TRansformer (DETR) for object detection
- Suggested resources

Multi-head Self-Attention

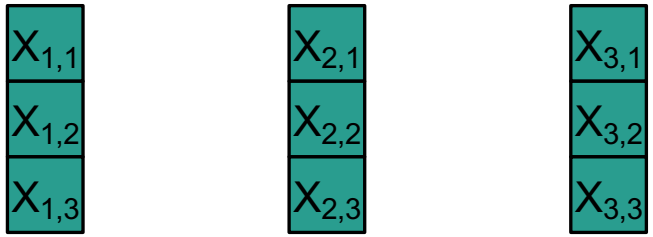
- Use H independent “attention heads” in parallel.
  - Attention heads do NOT share parameters.
- Each Input embedding is split into H parts.
- Each part is then processed by an attention head.



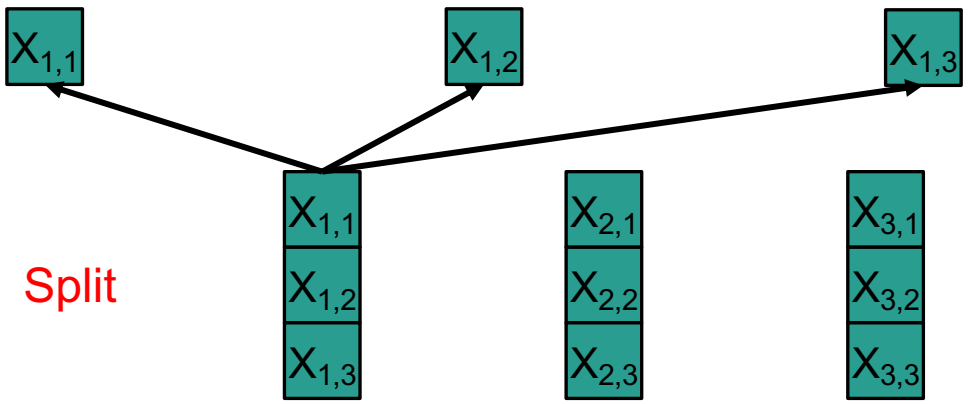


Multi-head Self-Attention

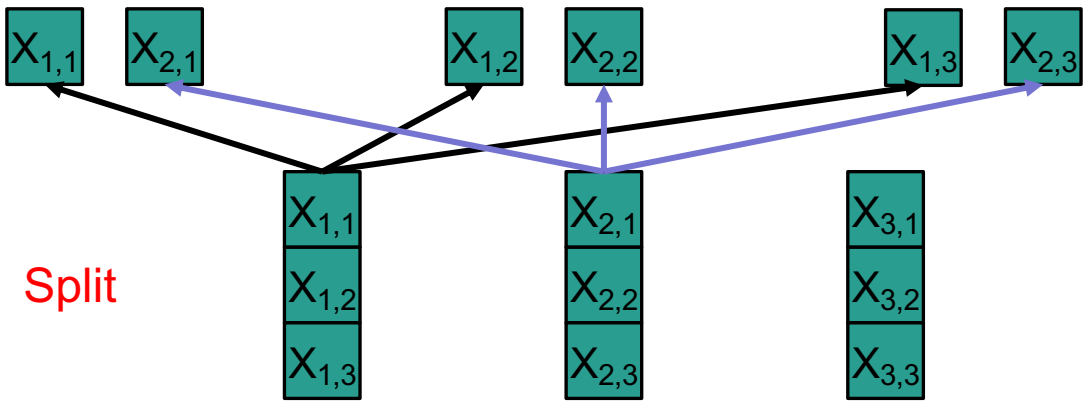
Split



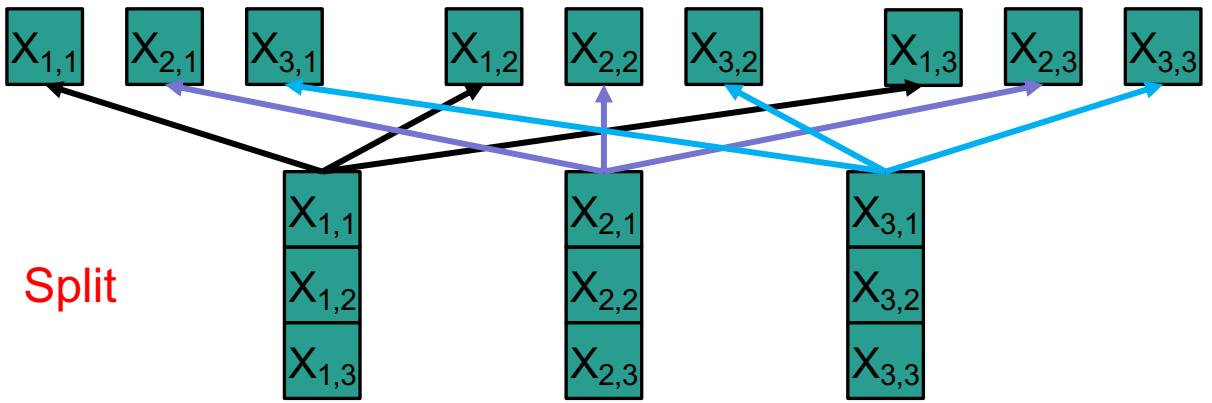
Multi-head Self-Attention



Multi-head Self-Attention

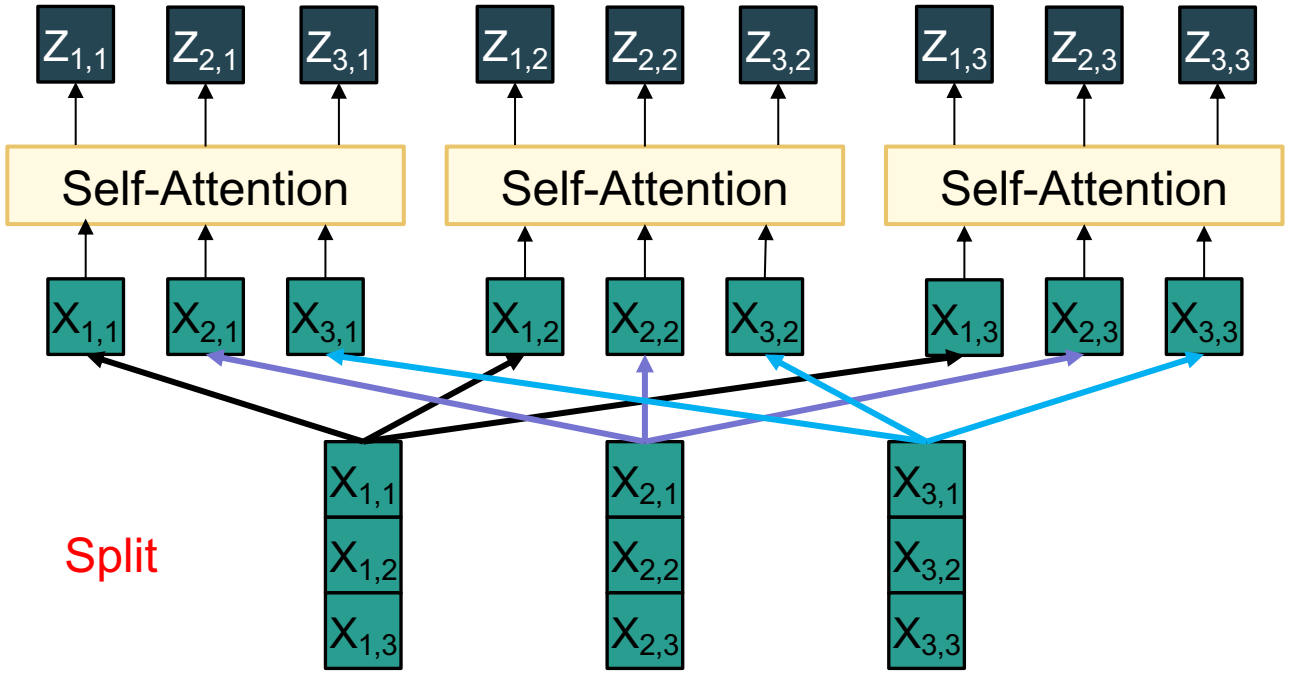


Multi-head Self-Attention

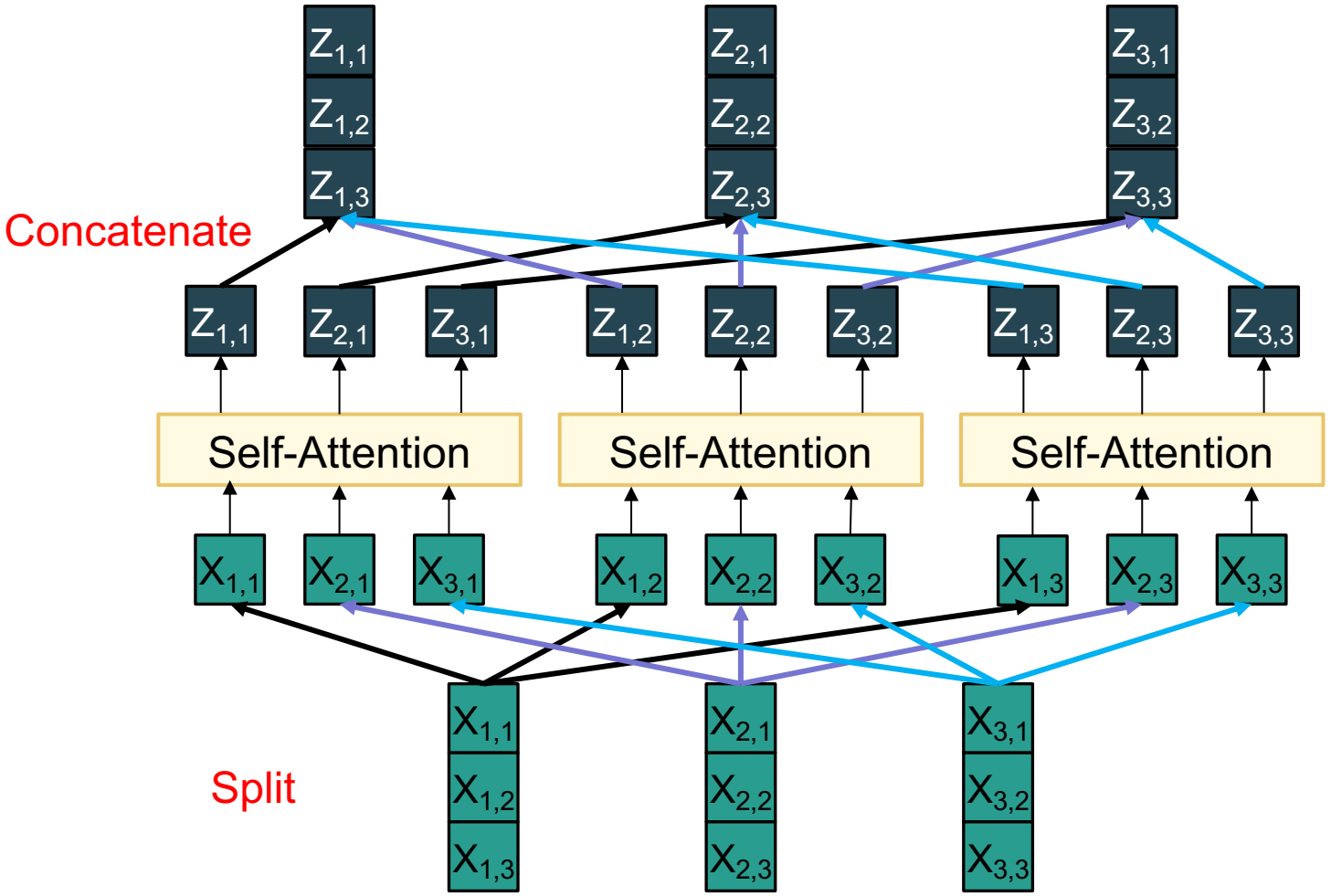


Multi-head Self-Attention

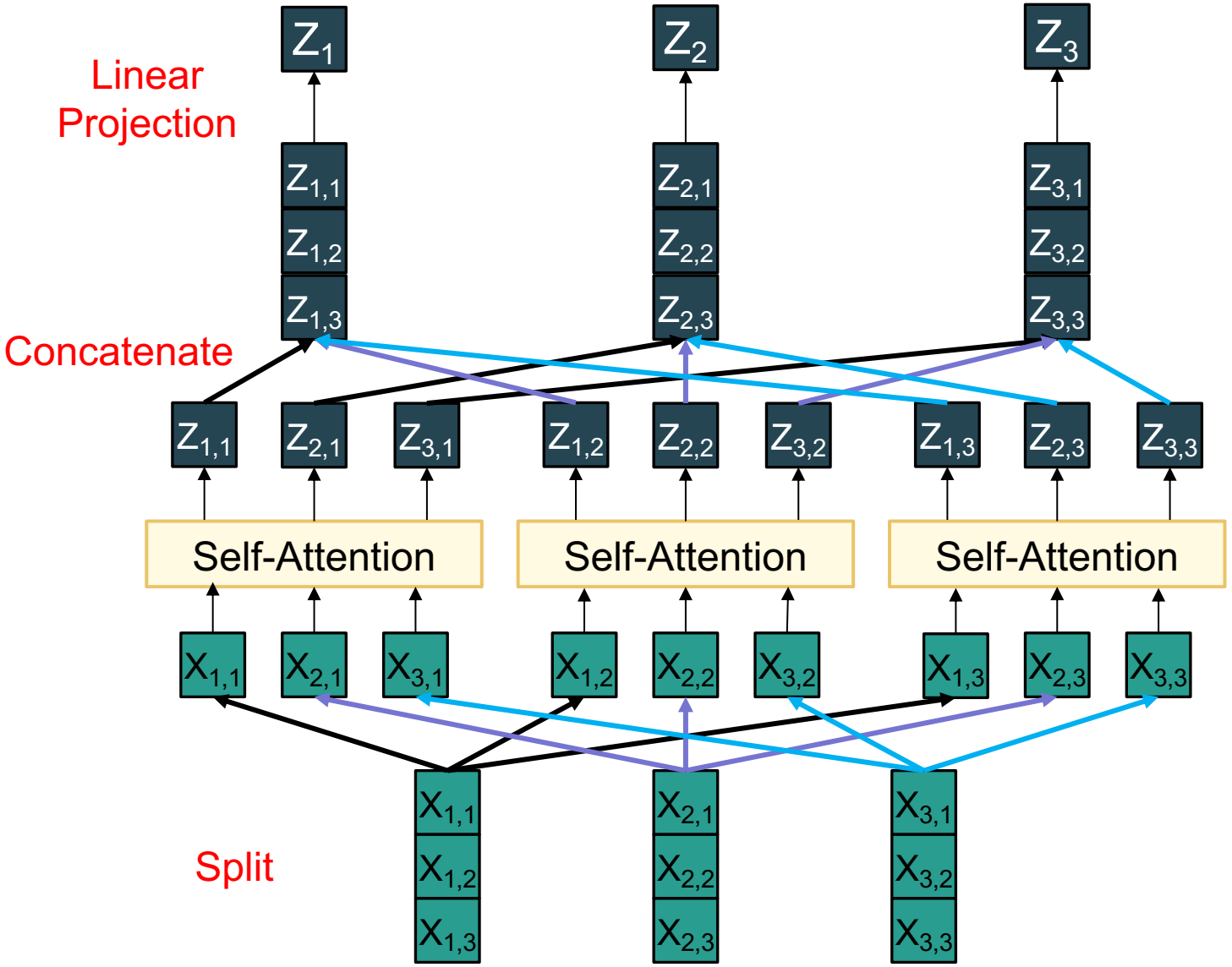
Run self-attention in parallel on each set of input vectors using a different set of parameters for each self-attention head.



Multi-head Self-Attention



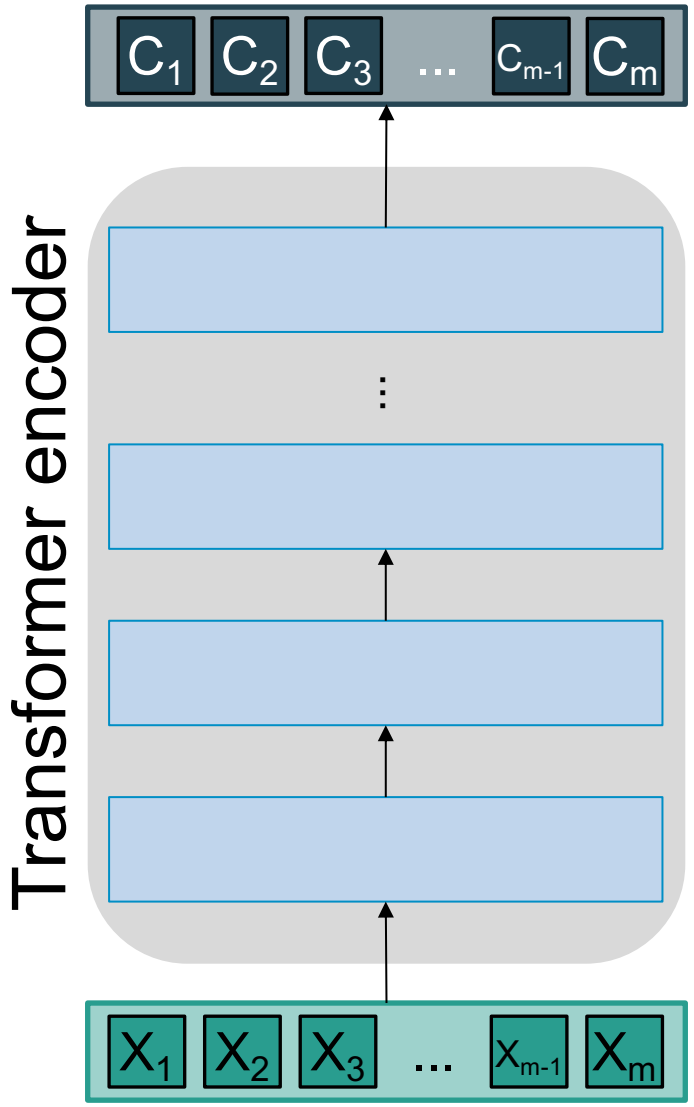
Multi-head Self-Attention



Outline

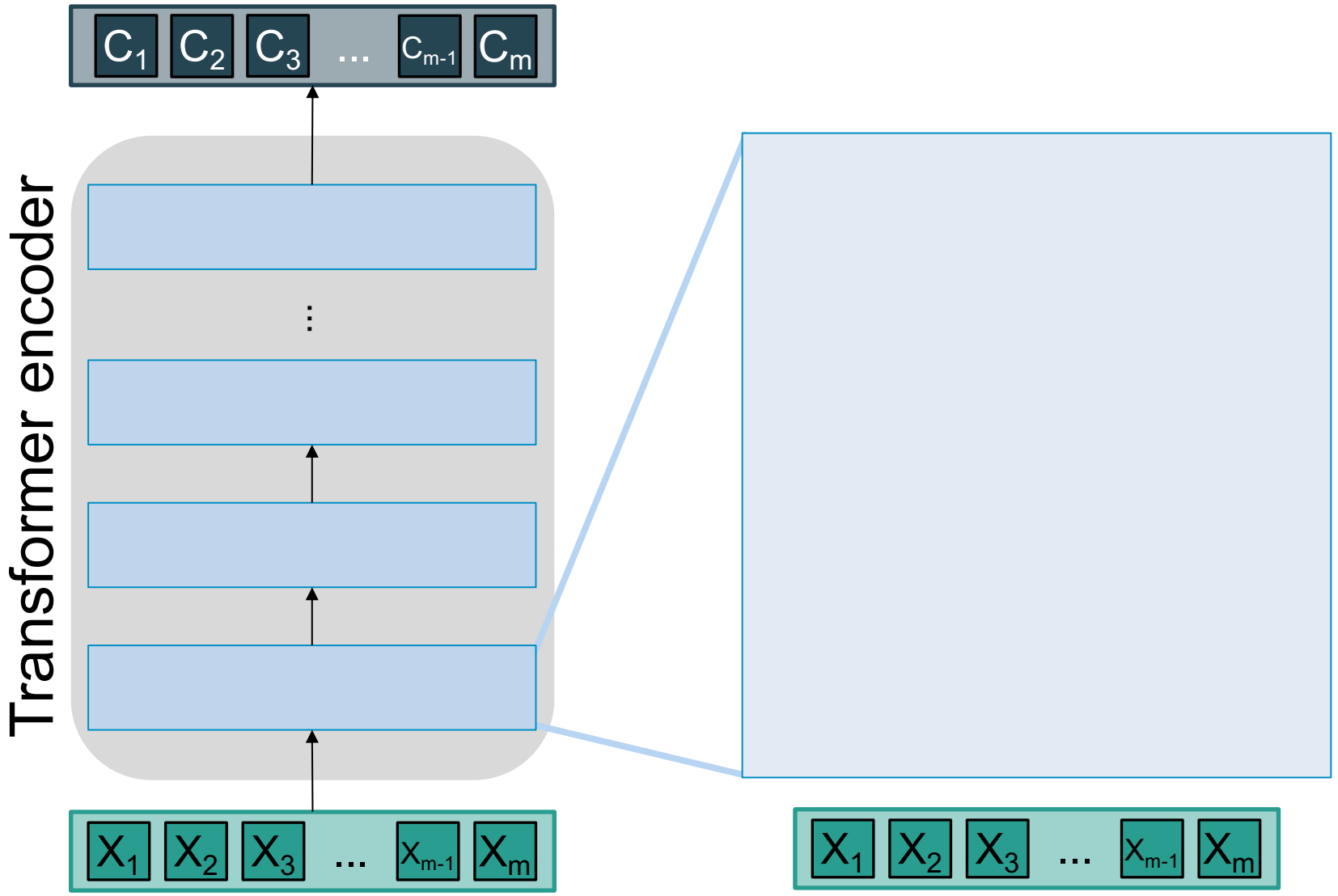
- Self-attention
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- **Transformer encoder block**
- Transformer decoder block
- Vision transformer (ViT)
- Swin transformer
- DEtection TRansformer (DETR) for object detection
- Suggested resources

Transformer encoder block



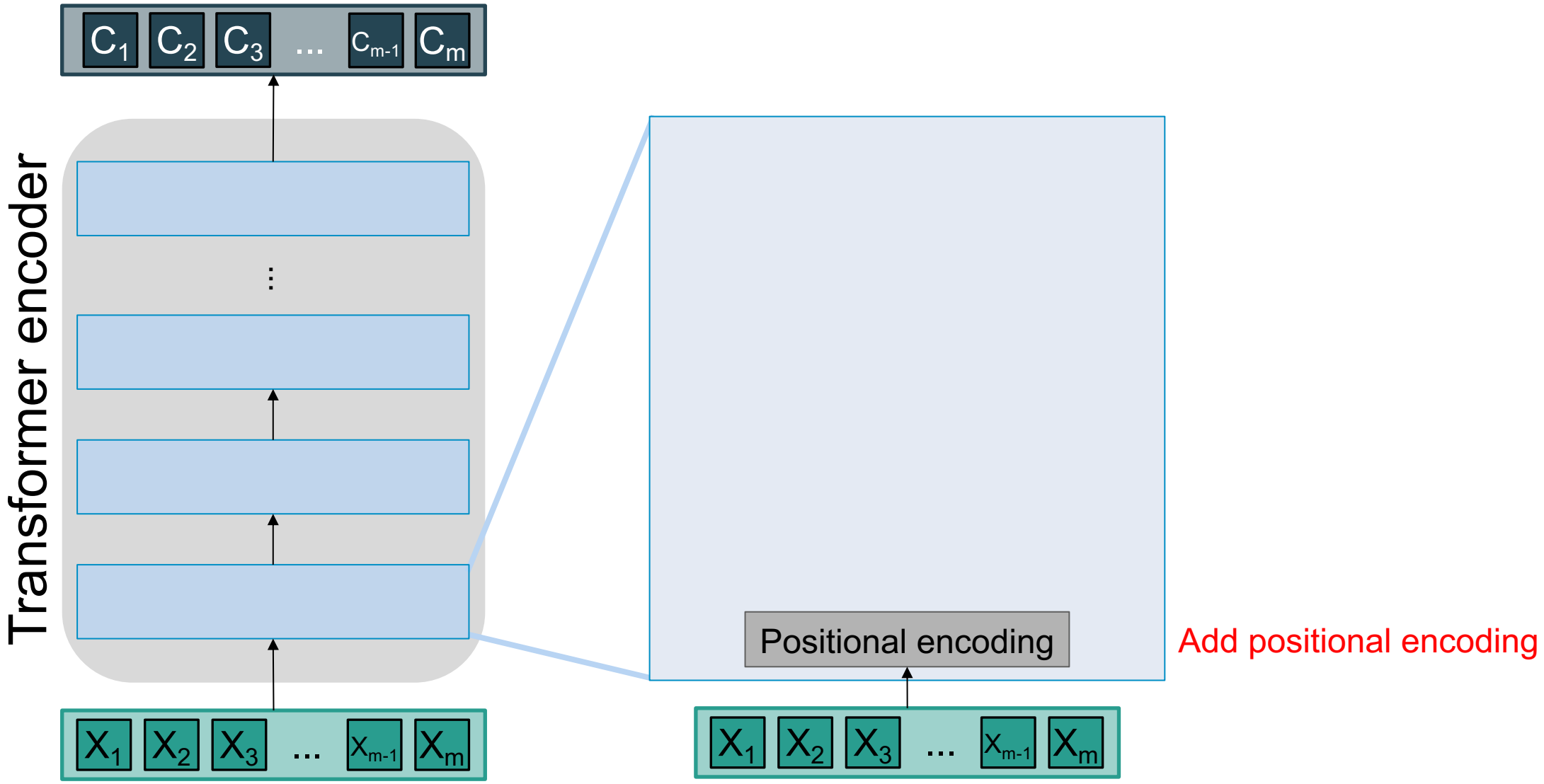
Transformer encoder is made up of a series of N encoder blocks.
In the original model (Vaswani et al.) N=6

Transformer encoder block

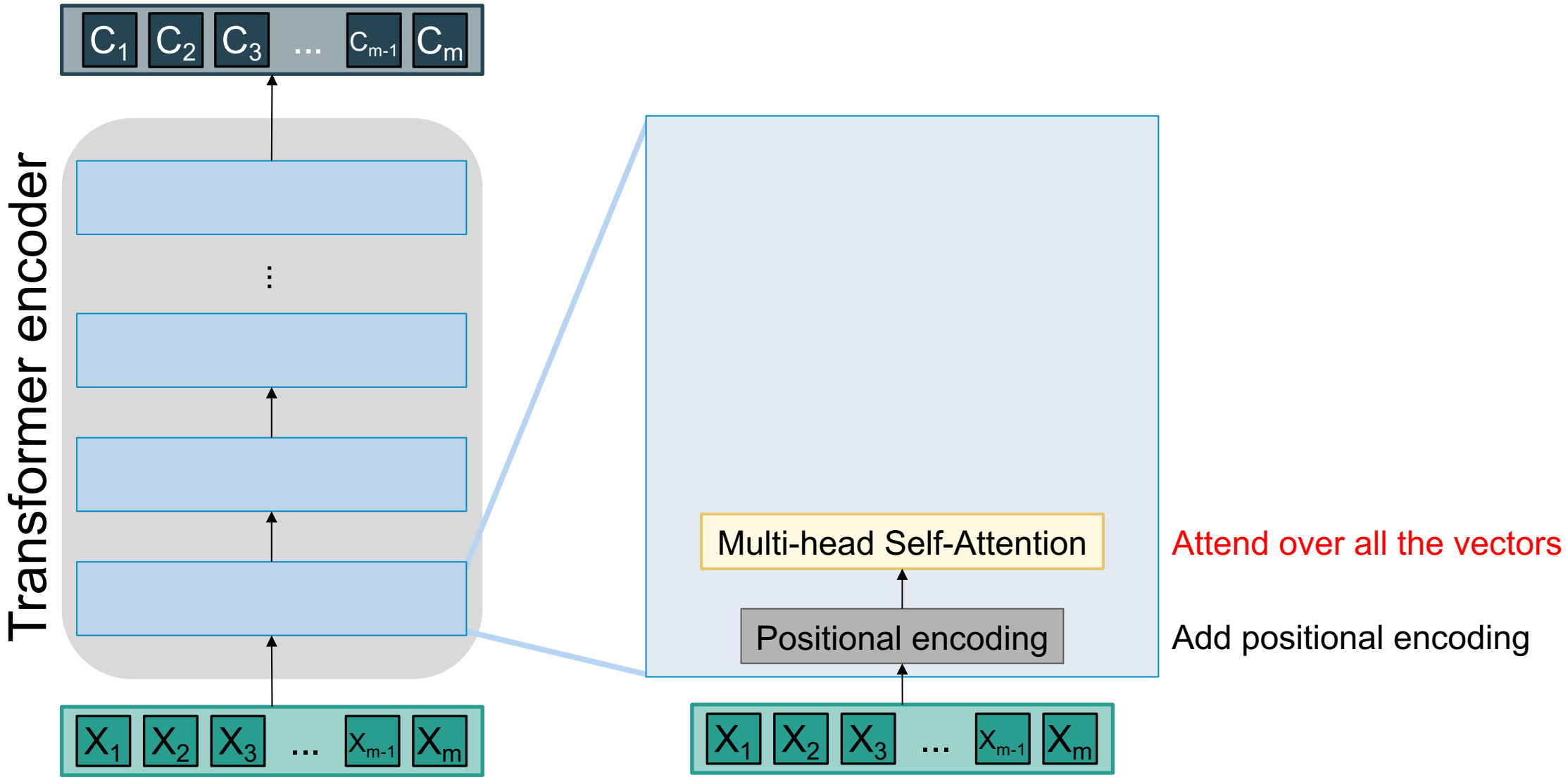


Let's look at the architecture of an encoder block.

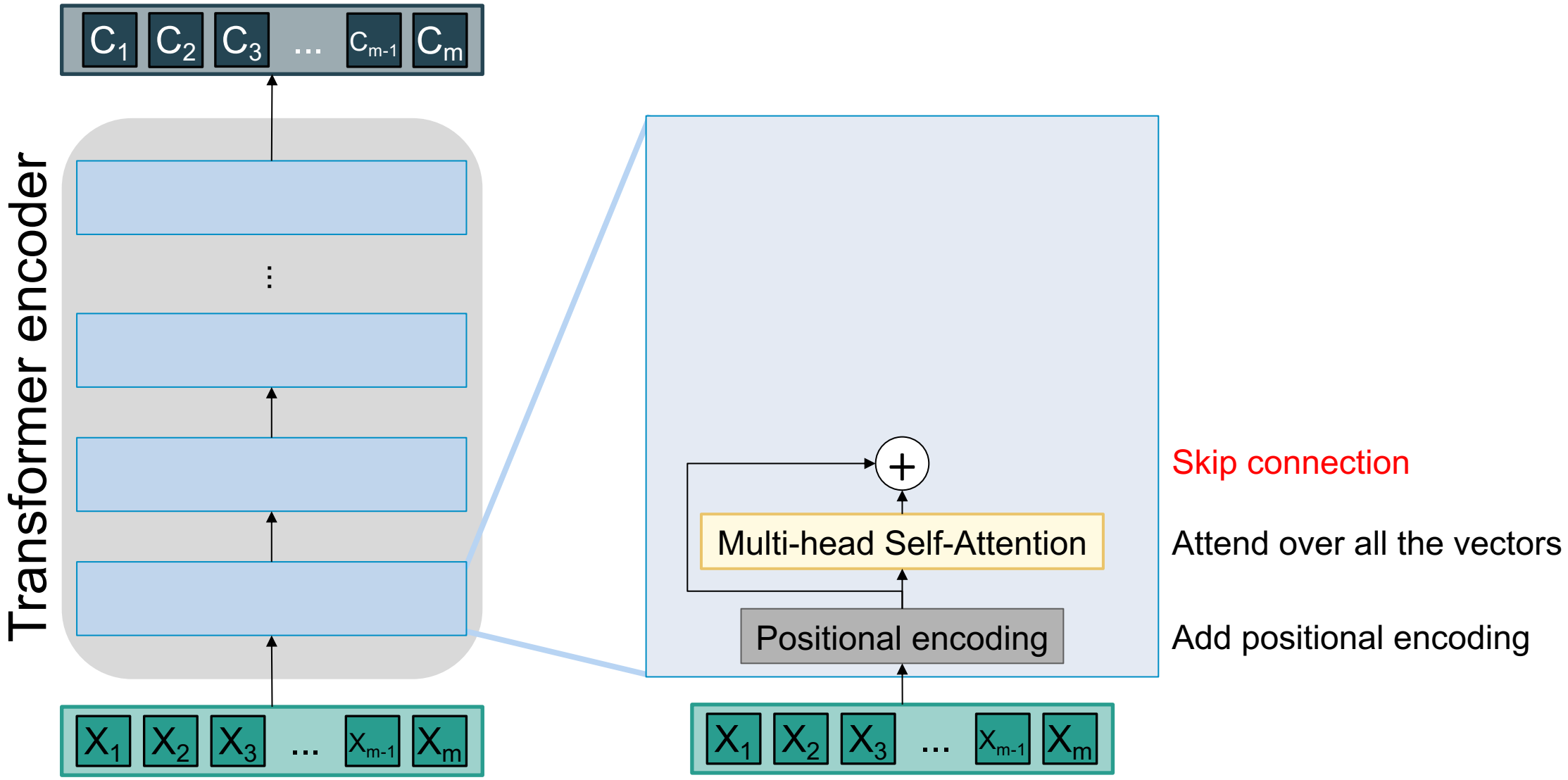
Transformer encoder block



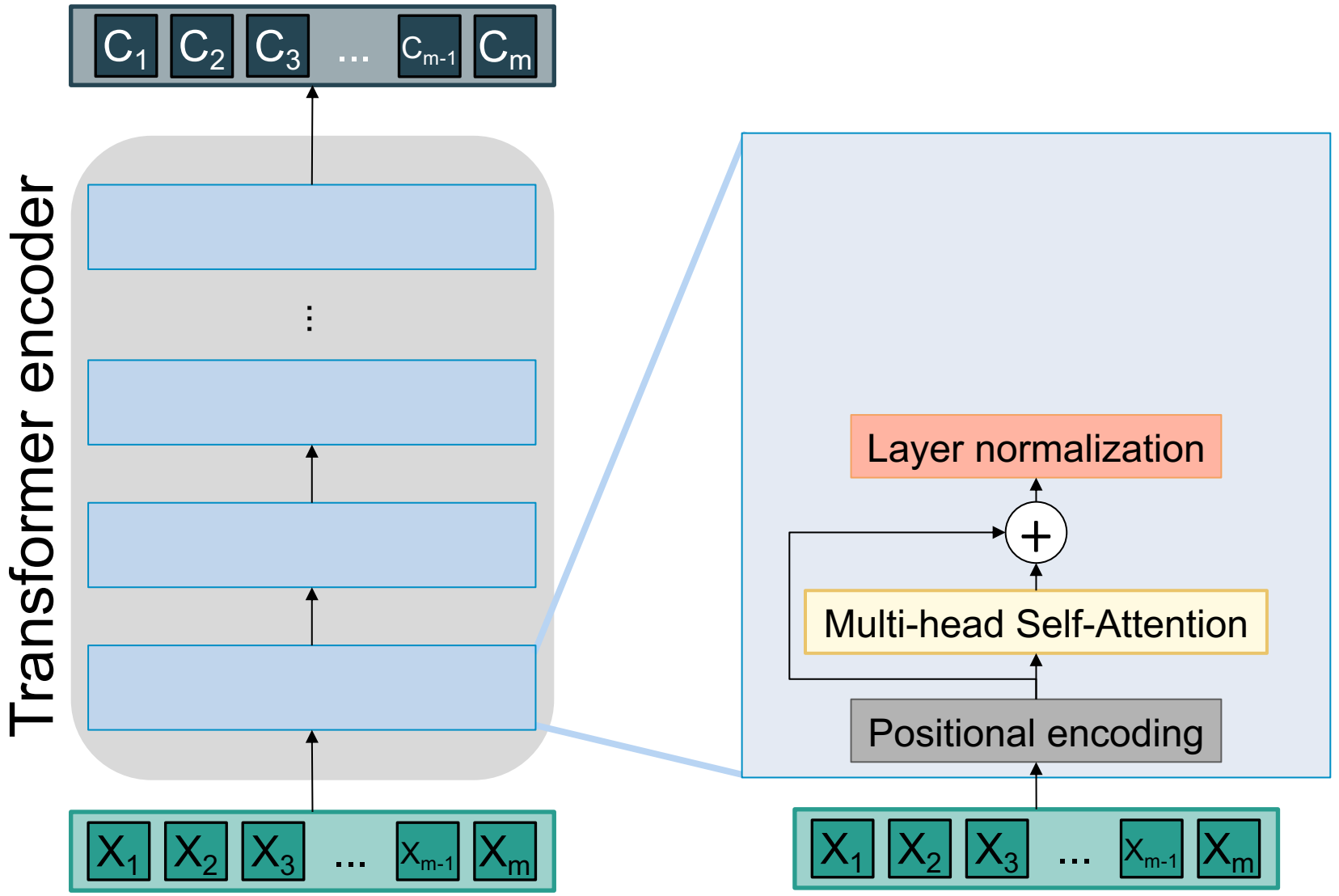
Transformer encoder block



Transformer encoder block



Transformer encoder block



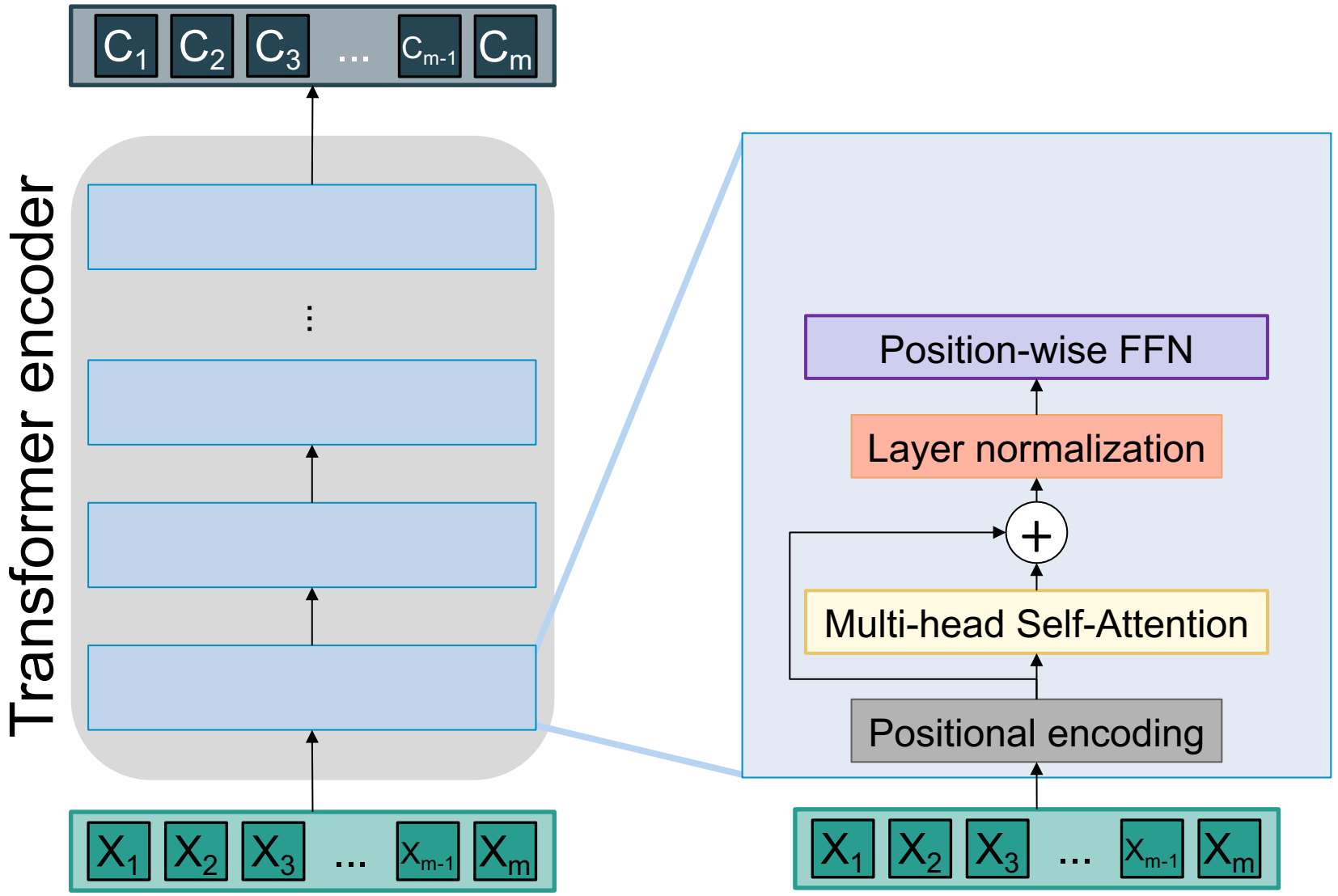
LayerNorm over each vector individually

Skip connection

Attend over all the vectors

Add positional encoding

Transformer encoder block



2-layer Feed Forward Network over each vector individually (shared W)

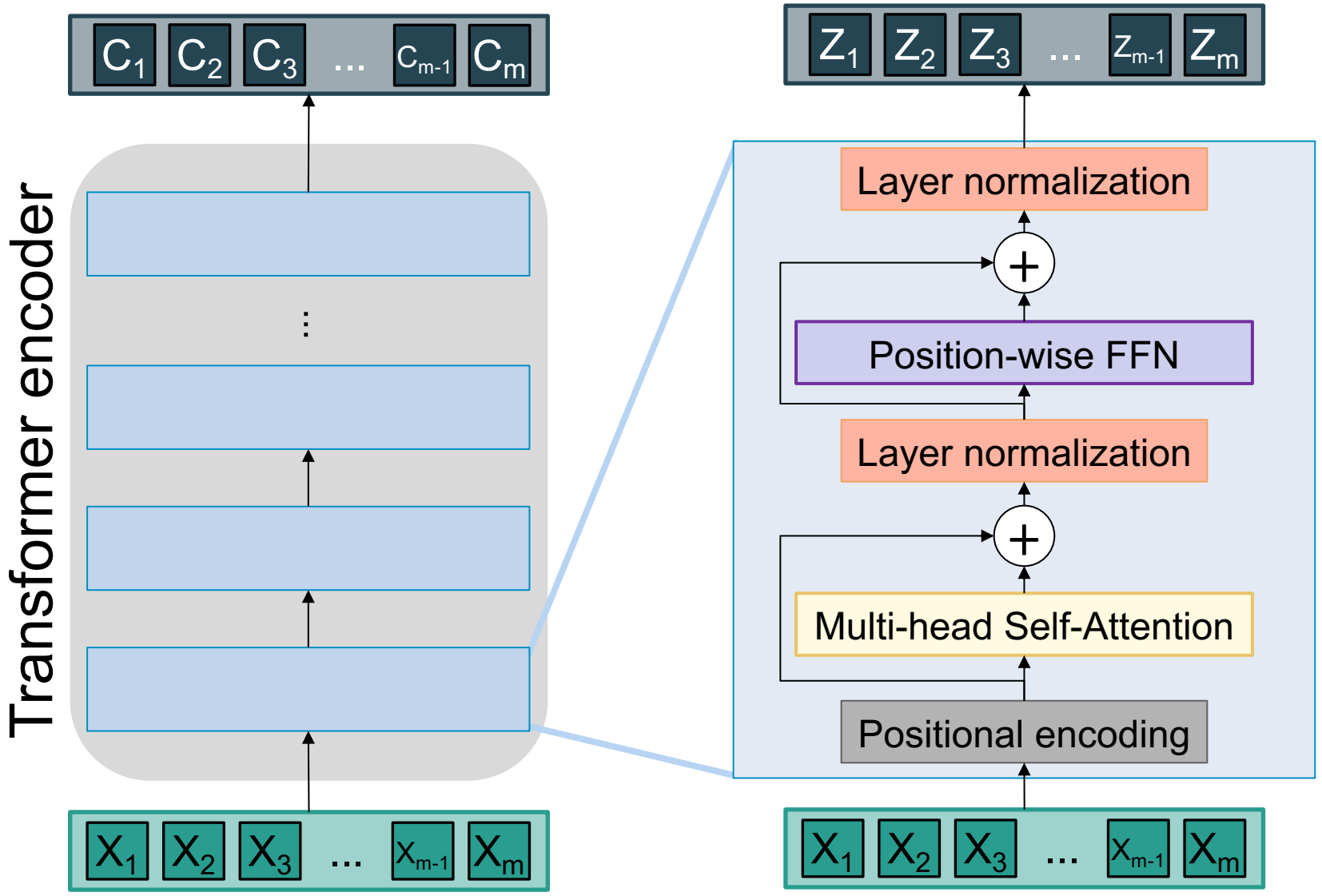
LayerNorm over each vector individually

Skip connection

Attend over all the vectors

Add positional encoding

Transformer encoder block



2-layer Feed Forward Network over each vector individually (shared W)

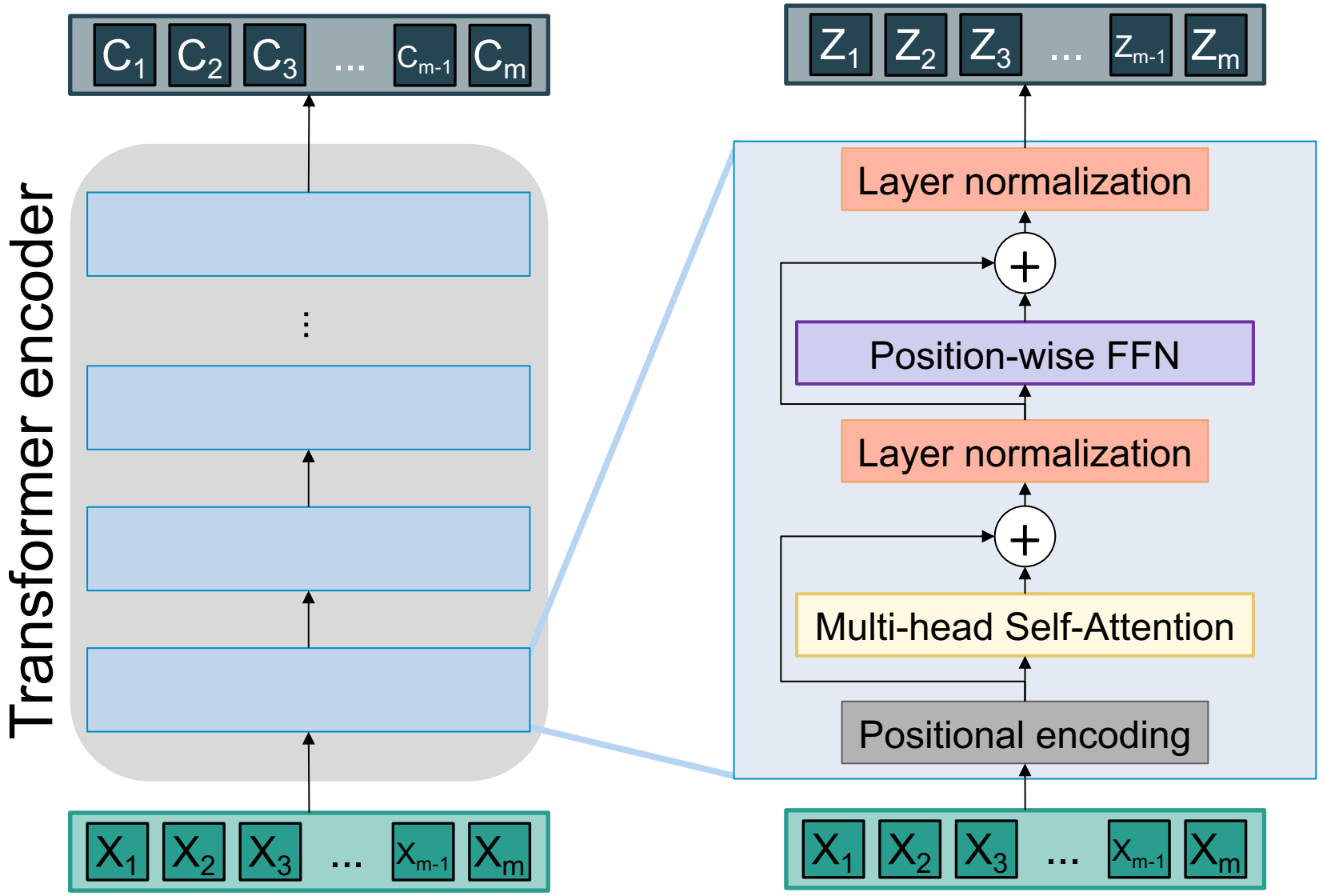
LayerNorm over each vector individually

Skip connection

Attend over all the vectors

Add positional encoding

Transformer encoder block



Transformer Encoder Block:

Input: Set of vectors **X**
Output: Set of vectors **Z**

Self-attention is the only interaction between vectors.

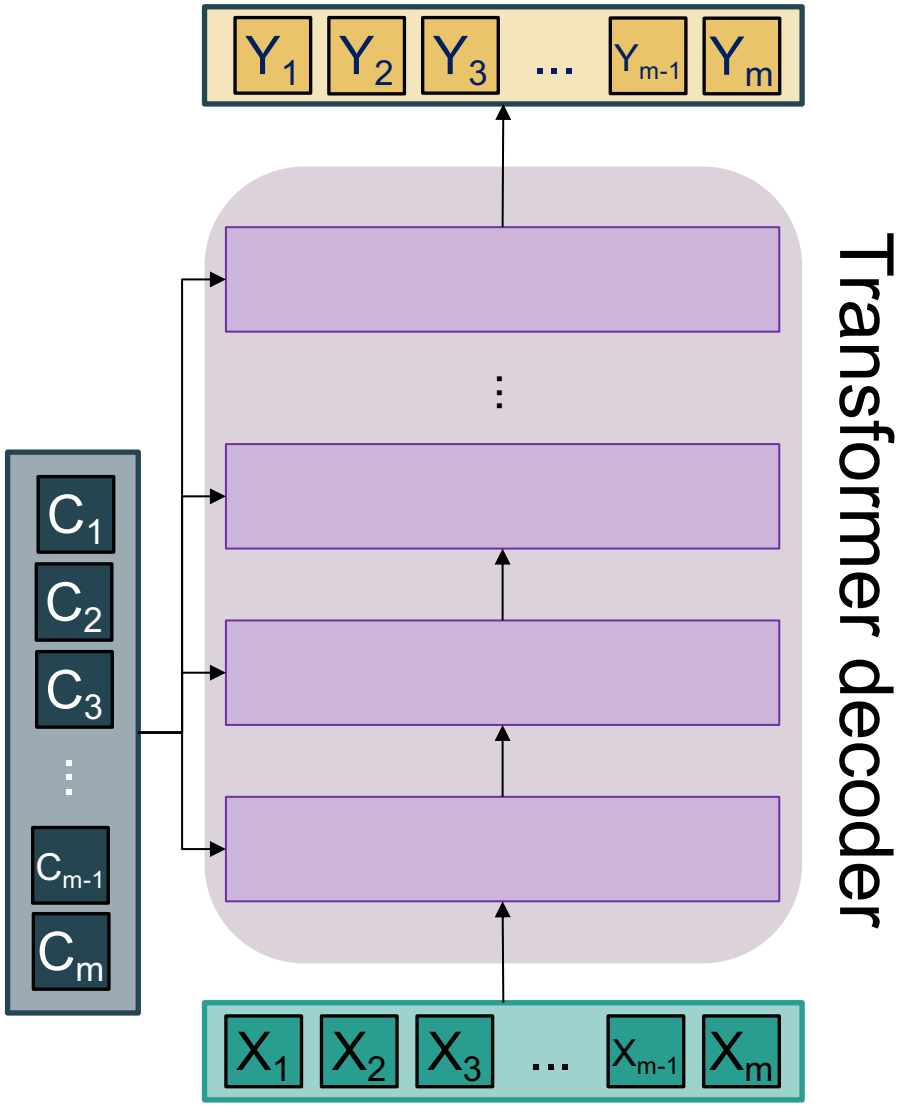
LayerNorm and FFN operate independently per vector.

Highly parallelisable but has high memory usage.

Outline

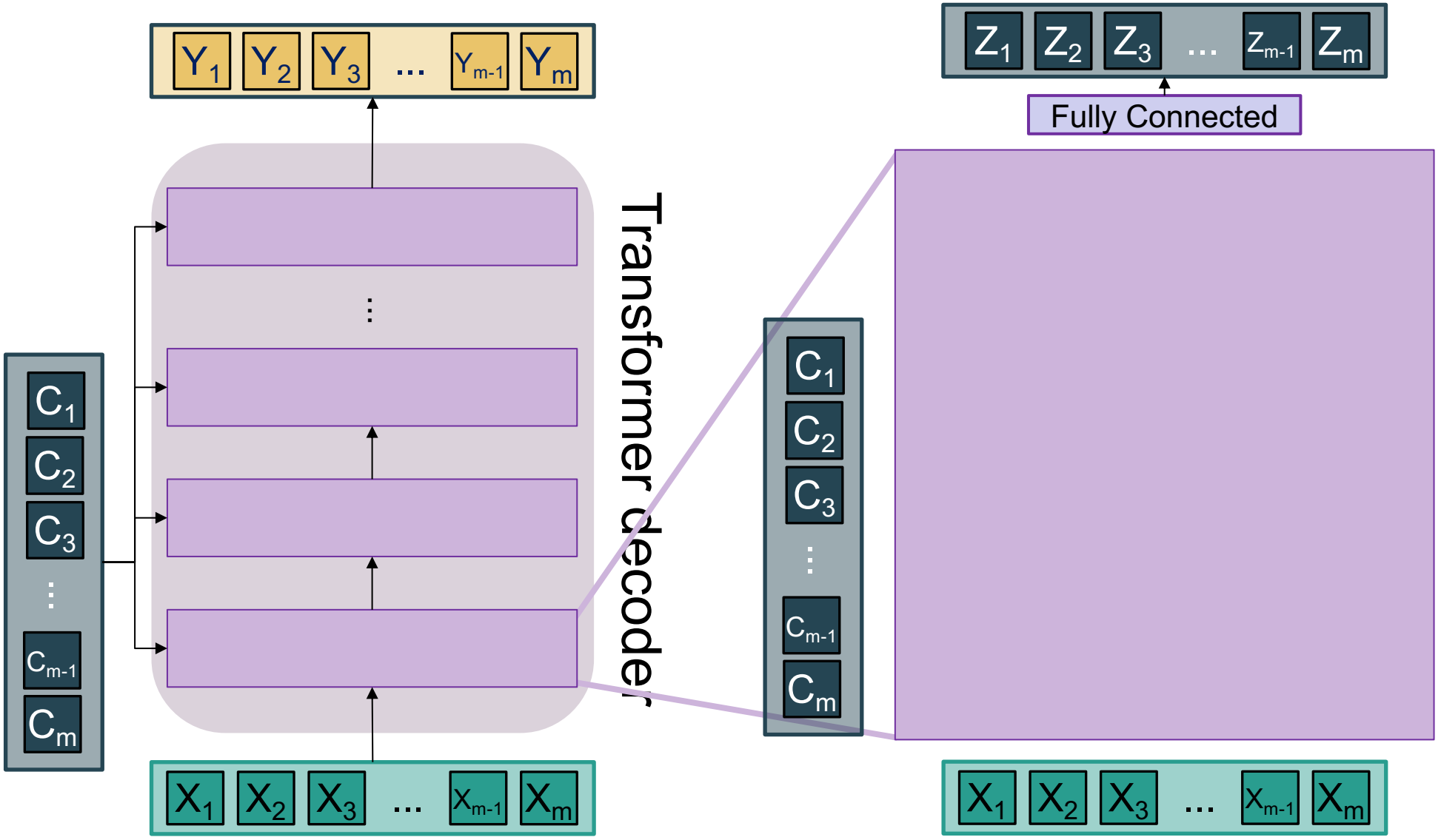
- Self-attention
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- Transformer encoder block
- **Transformer decoder block**
- Vision transformer (ViT)
- Swin transformer
- DEtection TRansformer (DETR) for object detection
- Suggested resources

Transformer decoder block



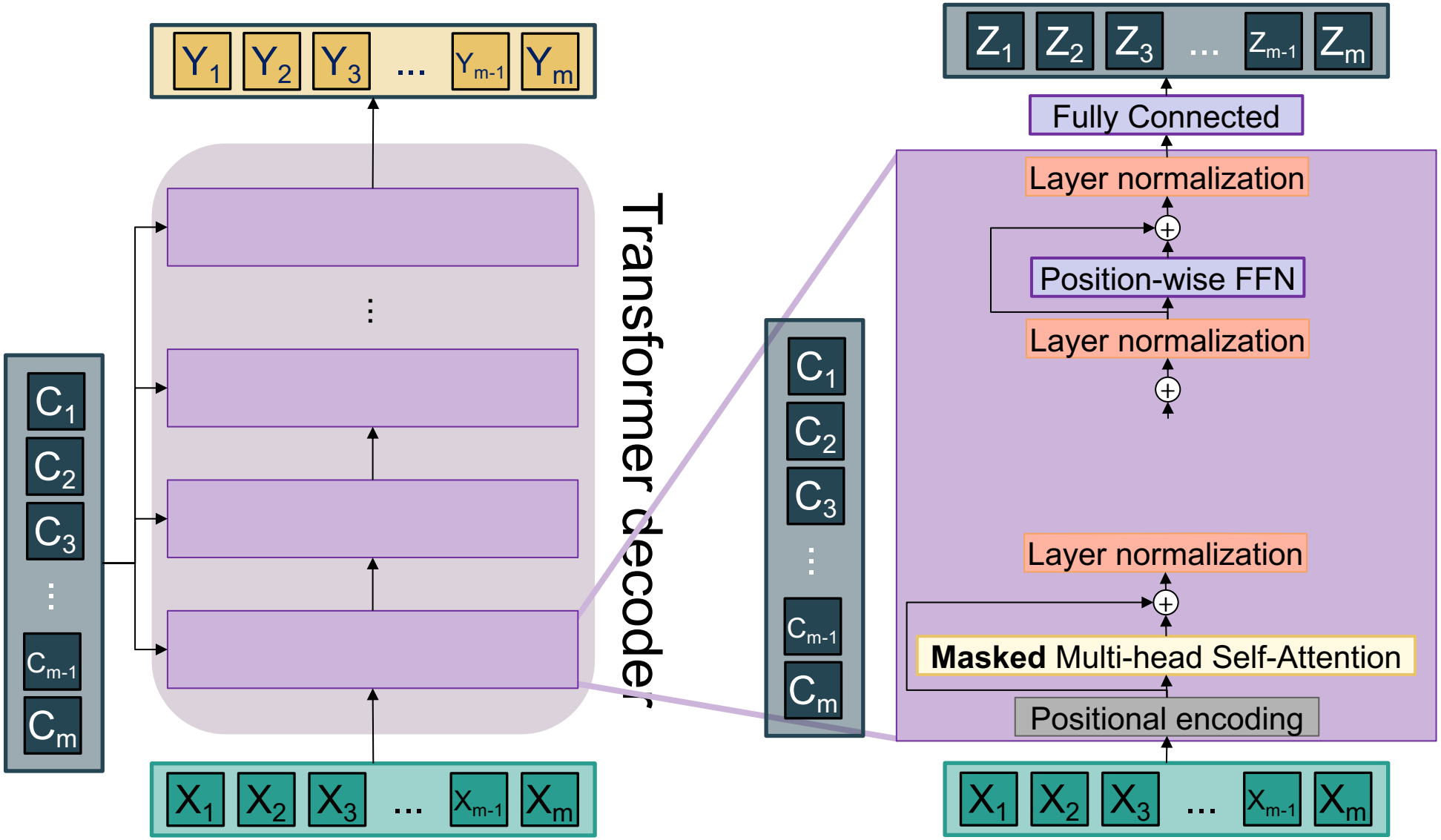
Transformer decoder is made up of a series of N decoder blocks.
In the original model (Vaswani et al.) N=6

Transformer decoder block



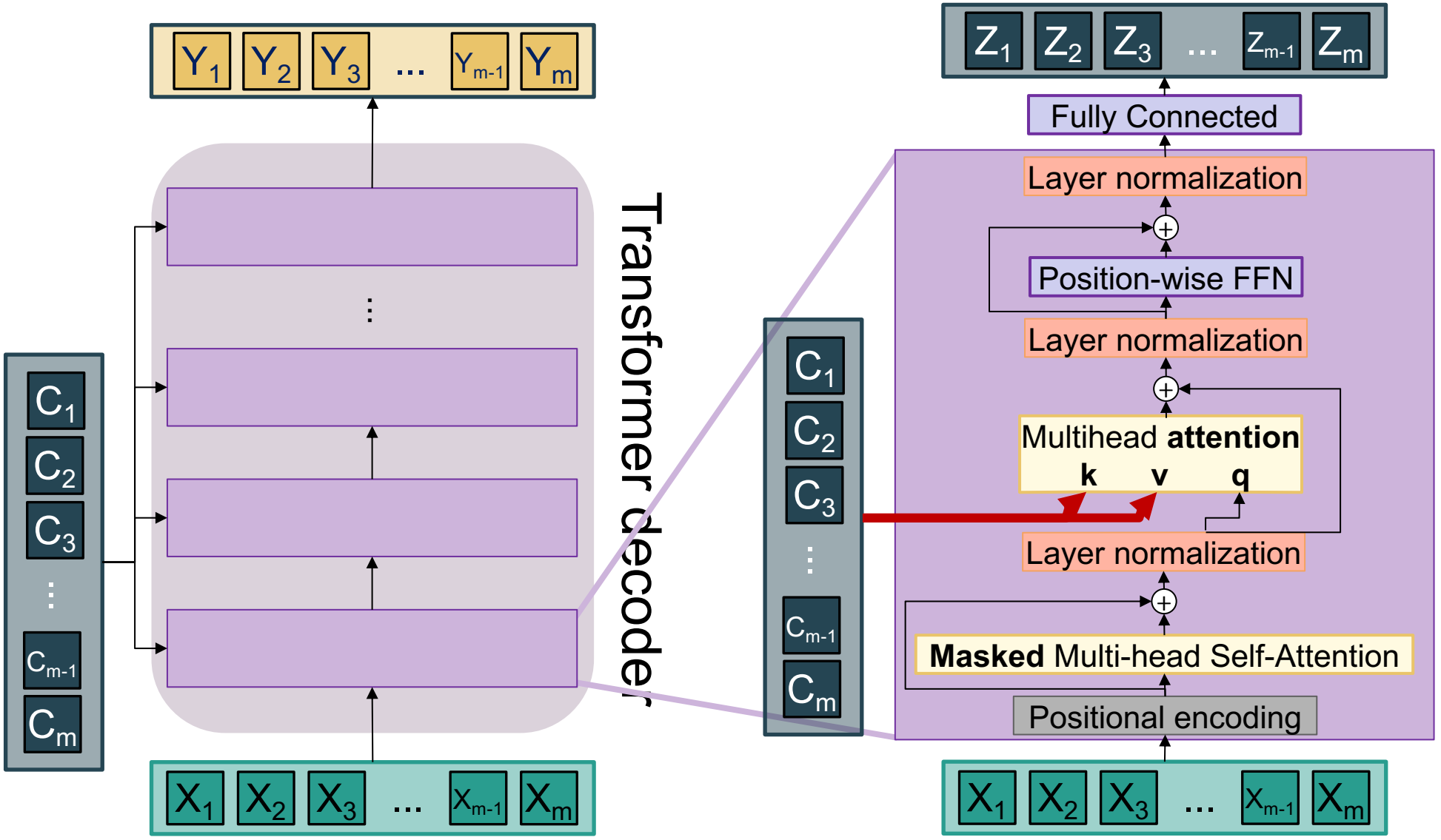
Let's look at the architecture of a decoder block.

Transformer decoder block



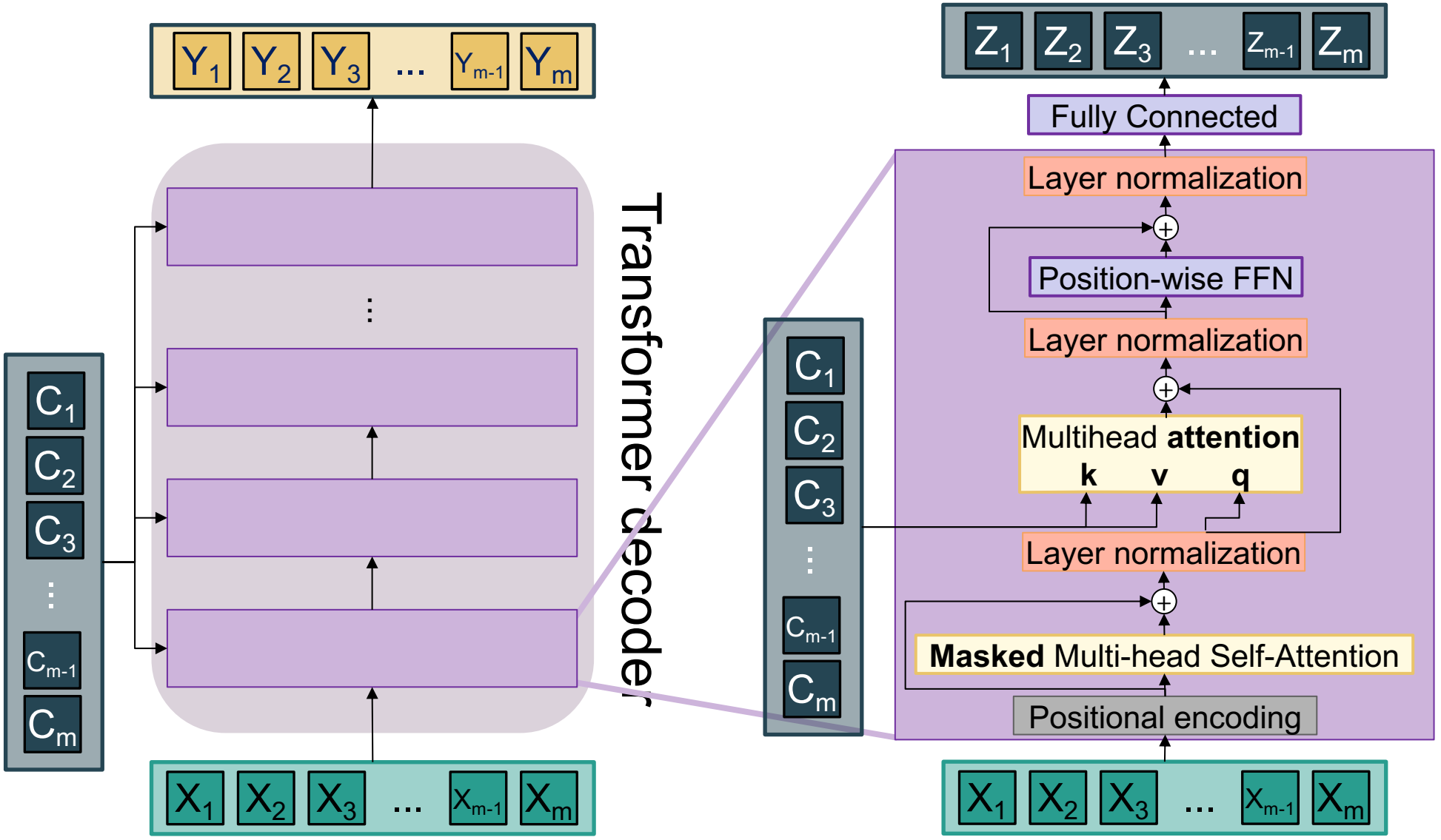
Most of the block is same as the encoder block.

Transformer decoder block



Multi-head attention block attends over encoder's outputs.

Transformer decoder block



Transformer Decoder Block:

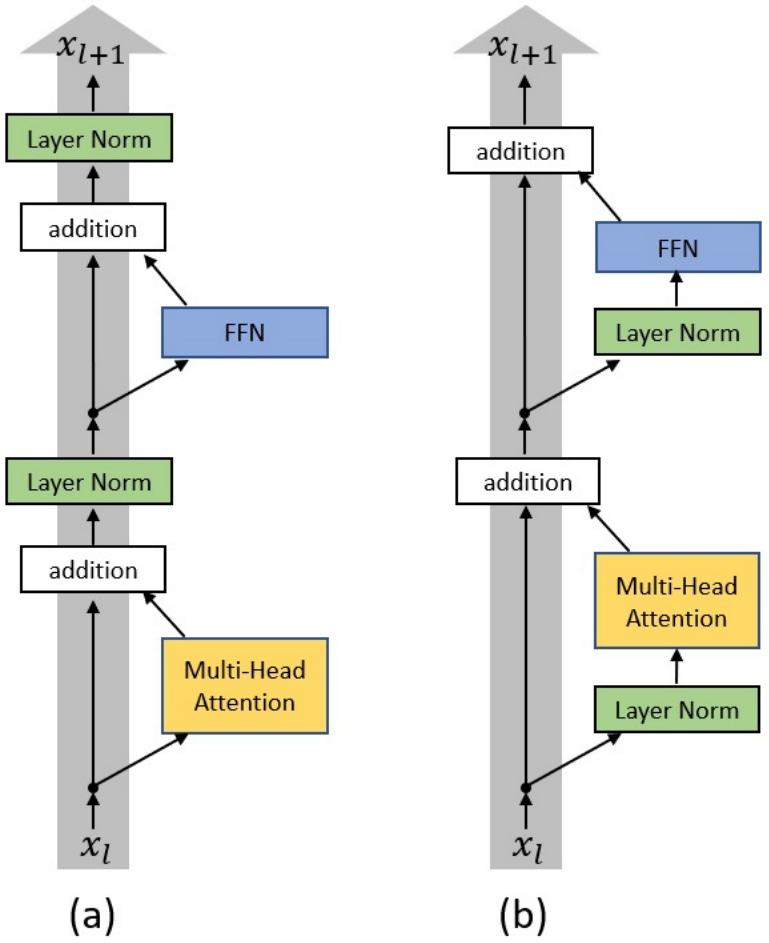
Input: Set of vectors **X** and set of context vectors **C**
Output: Set of vectors **Z**

Masked Self-attention interacts only with past inputs.

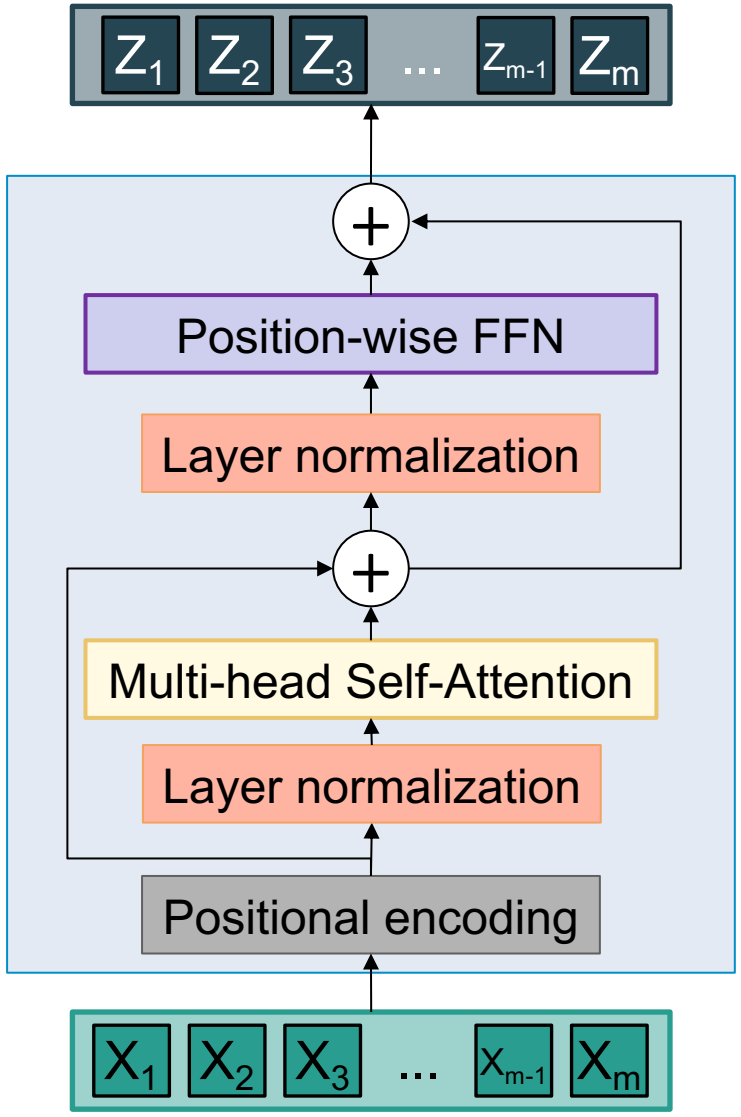
Multi-head attention is **NOT** self-attention. It attends over encoder's outputs.

Highly parallelisable but has high memory usage.

Transformer block: Pre-Norm Transformer



(a) Post-norm transformer (b) Pre-norm transformer
Image Source: https://arxiv.org/pdf/2002.04745.pdf



Pre-Norm Transformer:

Layer normalization is inside residual (skip) connections.

Gives more stable training.

Outline

- Self-attention
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- Transformer encoder block
- Transformer decoder block
- **Vision transformer (ViT)**
- Swin transformer
- DEtection TRansformer (DETR) for object detection
- Suggested resources

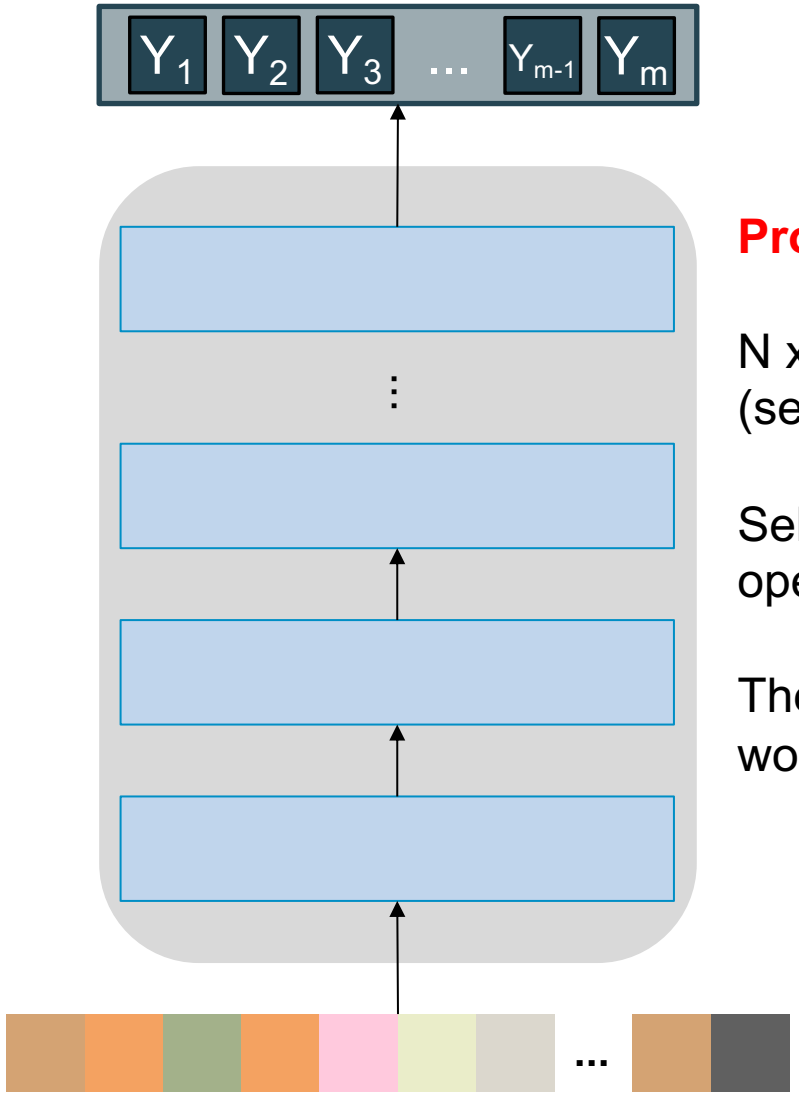
How to use transformers for images?

Idea: Treat an image as a sequence of pixels and use the standard transformer



Photo by Josh Hild on Unsplash

Flatten and feed as input



Problem: Memory usage!

N x N image has N^2 pixels (sequence length).

Self-attention is an $O(N^2)$ operation.

Therefore, self-attention would need N^4 elements!

Idea: Use standard transformer on patches



Photo by Josh Hild on Unsplash

Idea: Use standard transformer on patches



Photo by Josh Hild on Unsplash

Vision Transformer (ViT)

N input patches, each of shape 3x16x16



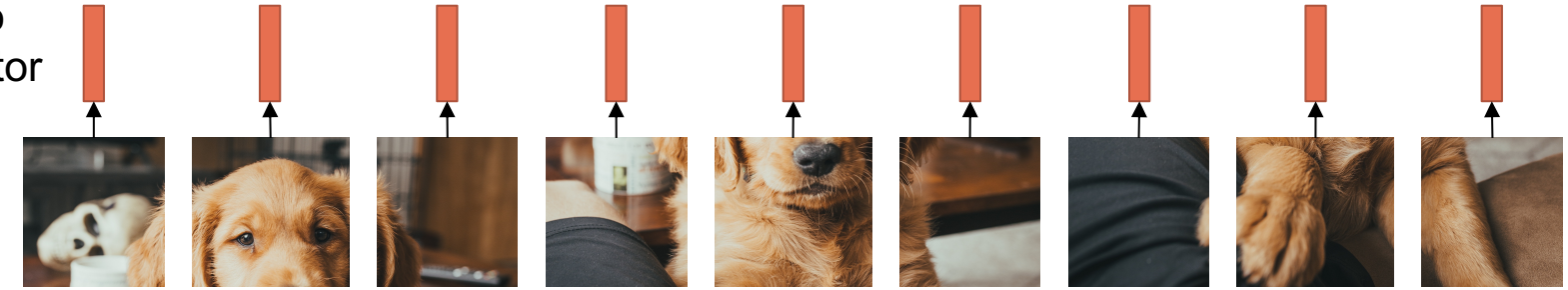
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Photo by Josh Hild on Unsplash

Vision Transformer (ViT)

Linear projection to D-dimensional vector

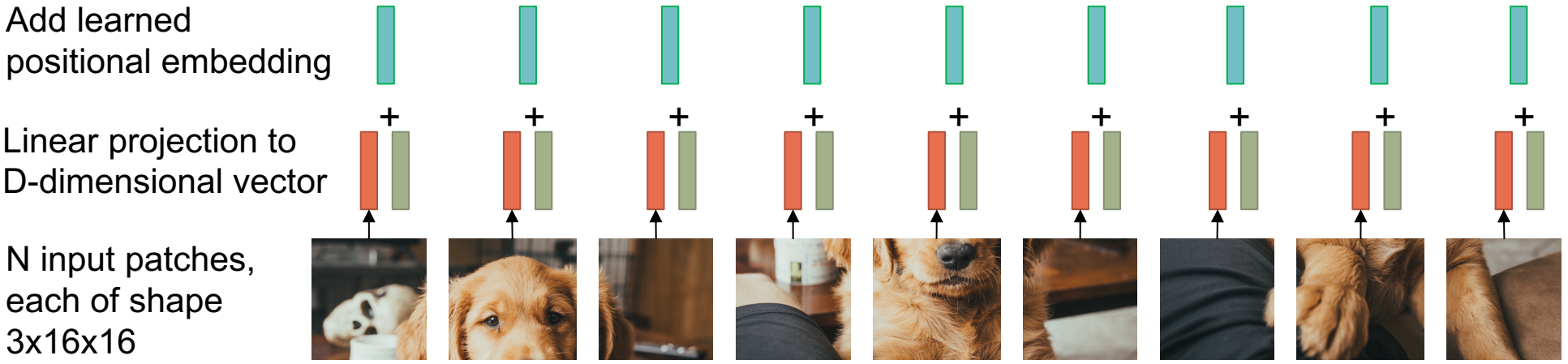
N input patches, each of shape 3x16x16



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Photo by Josh Hild on Unsplash

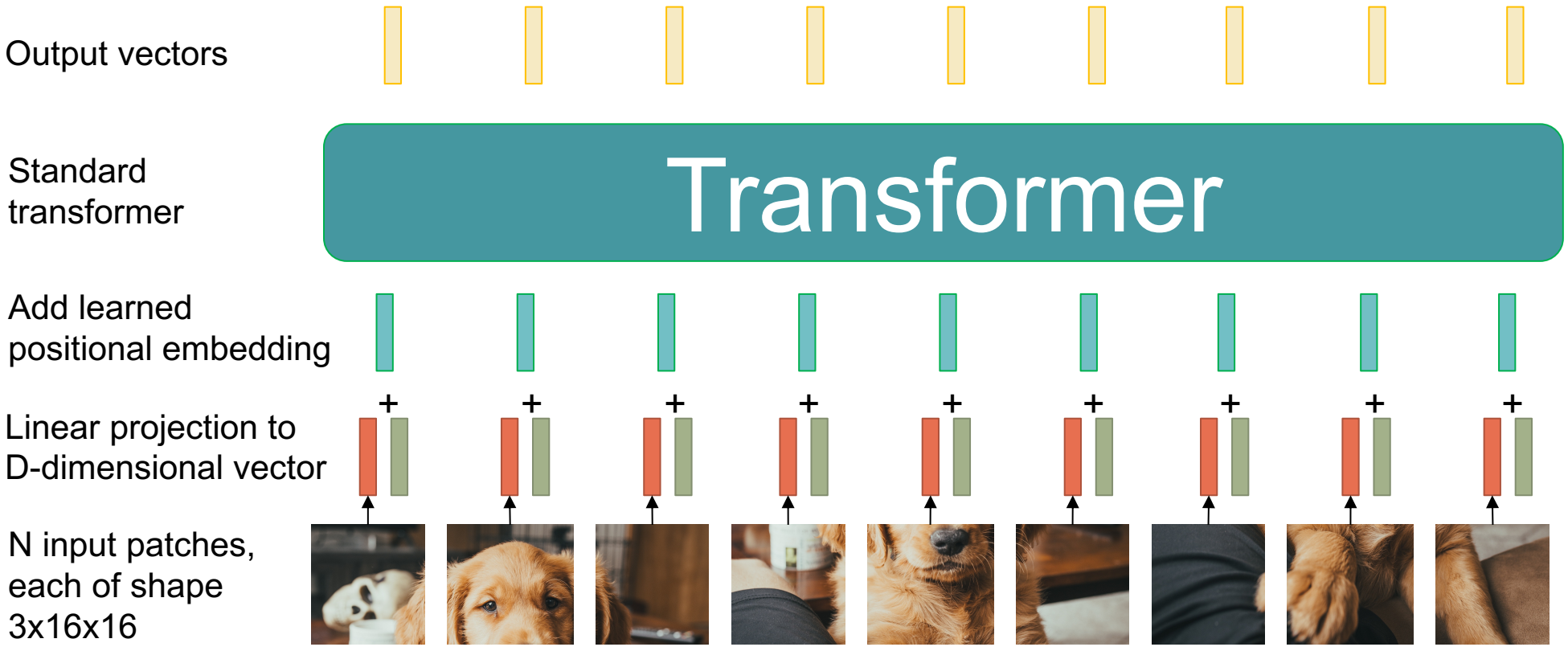
Vision Transformer (ViT)



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Photo by Josh Hild on Unsplash

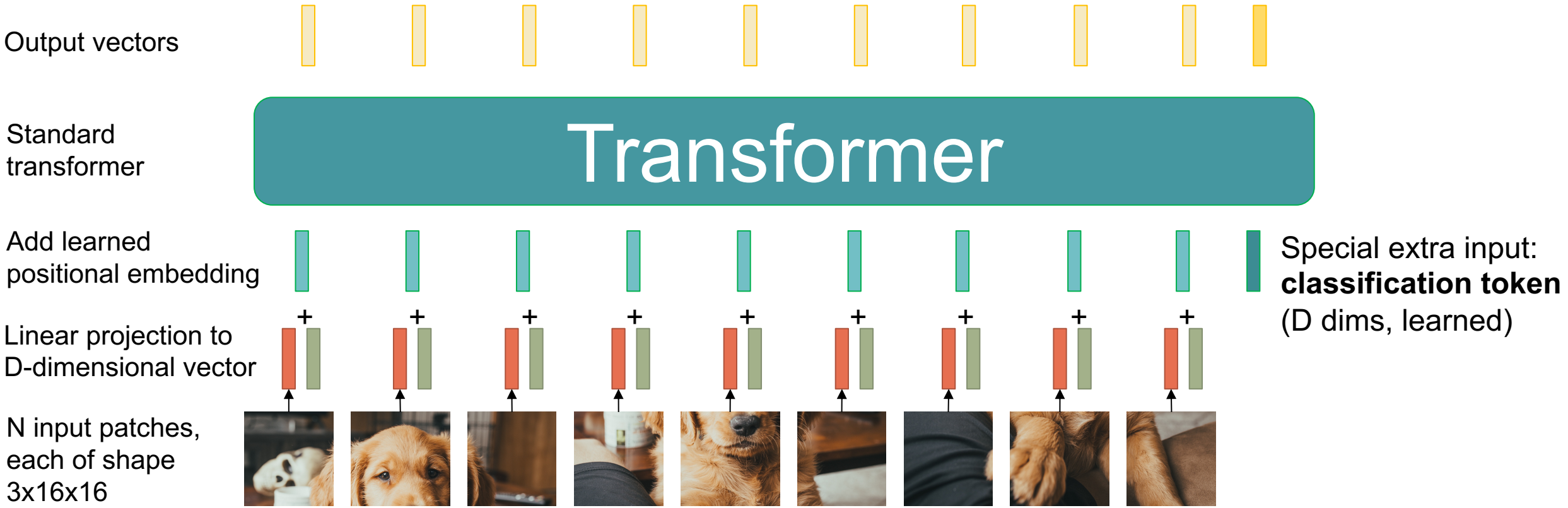
Vision Transformer (ViT)



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Photo by Josh Hild on Unsplash

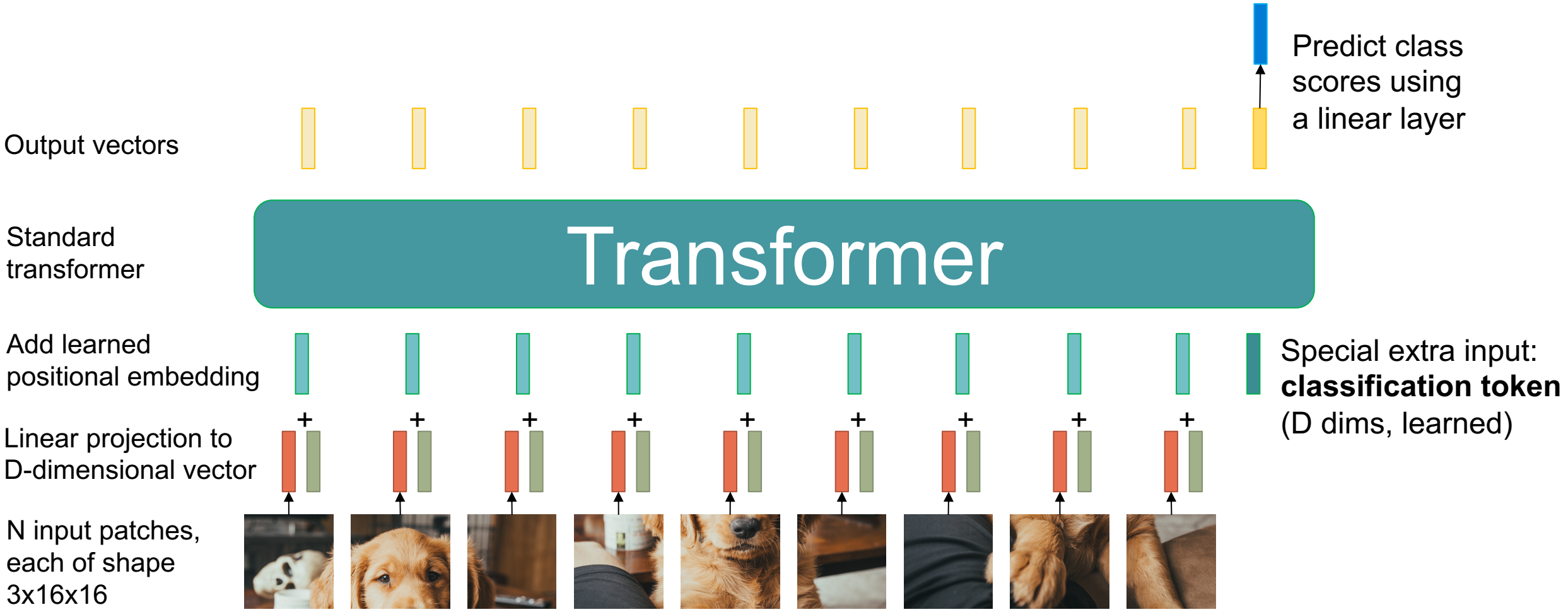
Vision Transformer (ViT)



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Photo by Josh Hild on Unsplash

Vision Transformer (ViT)



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Photo by Josh Hild on Unsplash

Outline

- Self-attention
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- Transformer encoder block
- Transformer decoder block
- Vision transformer (ViT)
- **Swin transformer**
- DEtection TRansformer (DETR) for object detection
- Suggested resources

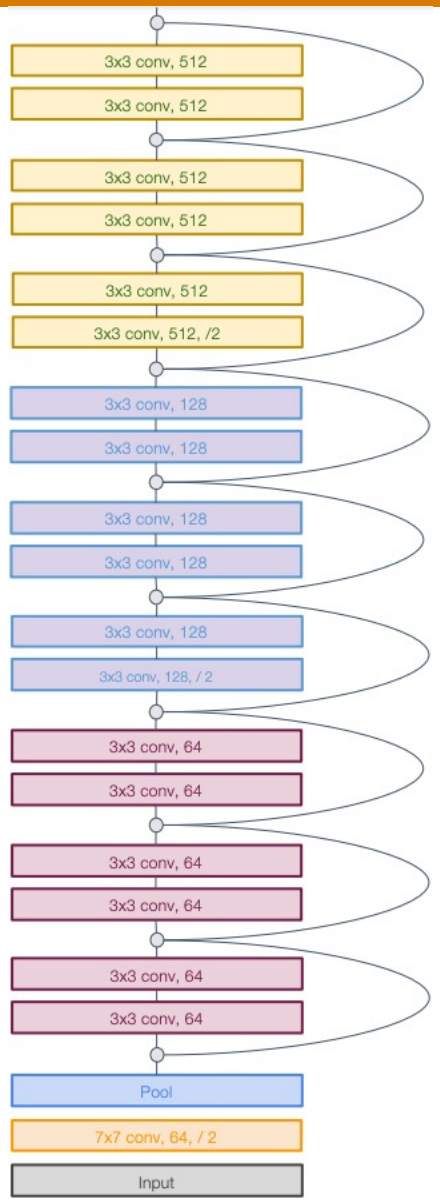
ViT vs ConvNet

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224



In most ConvNets, as we go deeper, resolution **decreases** and #channels **increase**. (Hierarchical architecture)

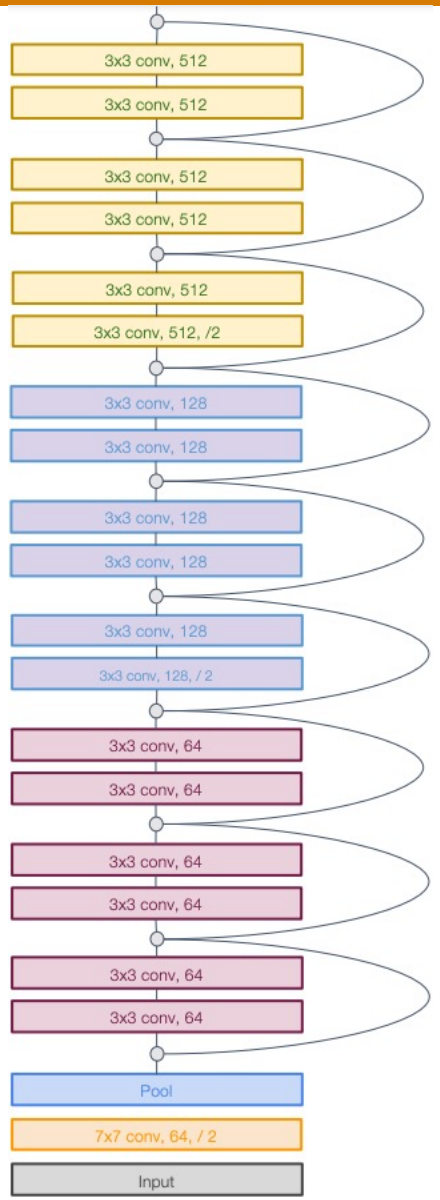
ViT vs ConvNet

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

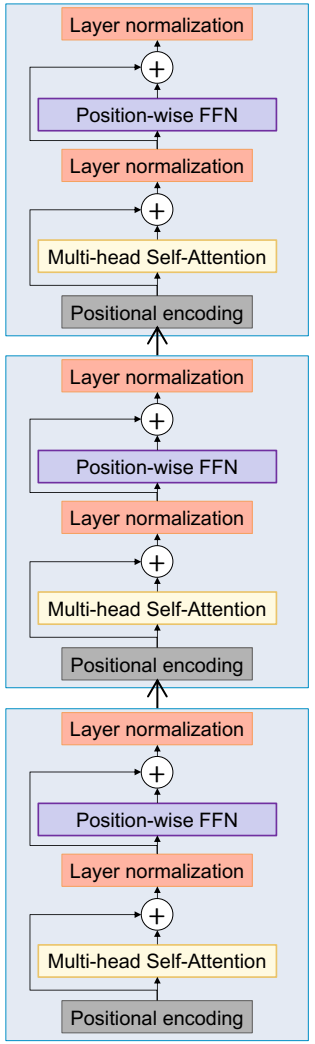
Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224



In most ConvNets, as we go deeper, resolution **decreases** and #channels **increase**. (Hierarchical architecture)

In ViT, all blocks have the same resolution and number of channels



3rd Block:
768 x 14 x 14

2nd Block:
768 x 14 x 14

1st Block:
768 x 14 x 14

Input:
3 x 224 x 224

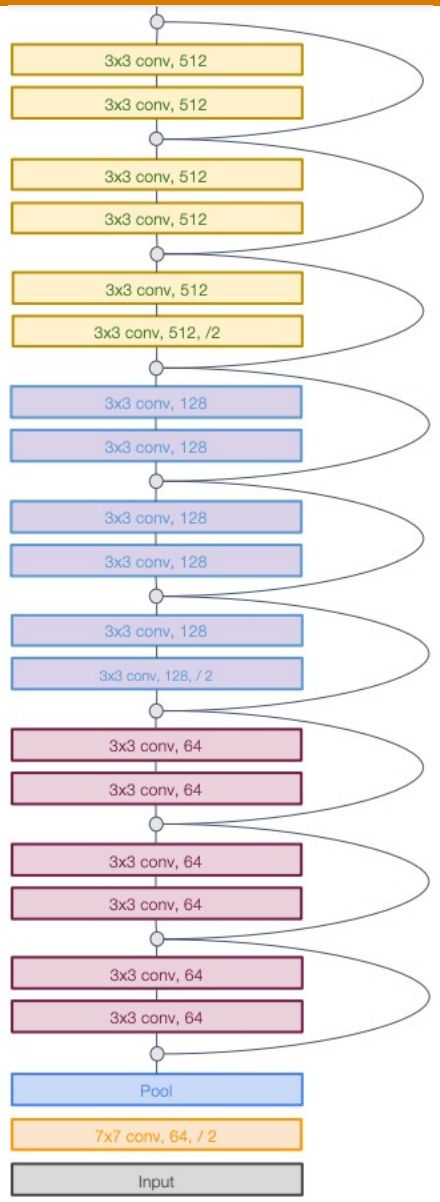
ViT vs ConvNet

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

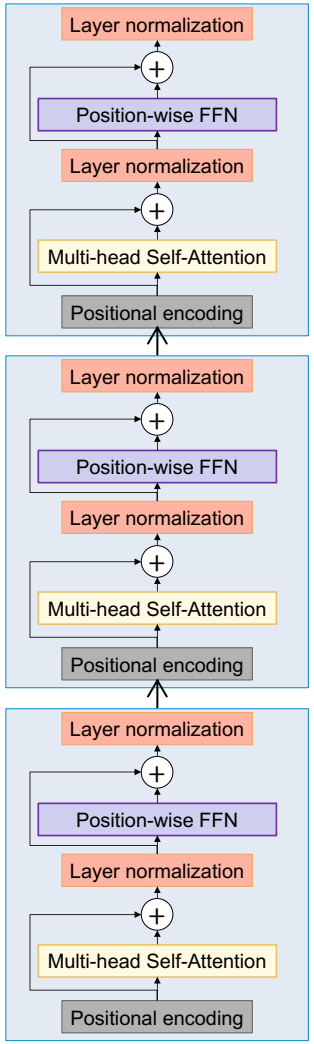
Input:
3 x 224 x 224



In most ConvNets, as we go deeper, resolution **decreases** and #channels **increase**. (Hierarchical architecture)

In ViT, all blocks have the same resolution and number of channels

Can we build a hierarchical ViT model?



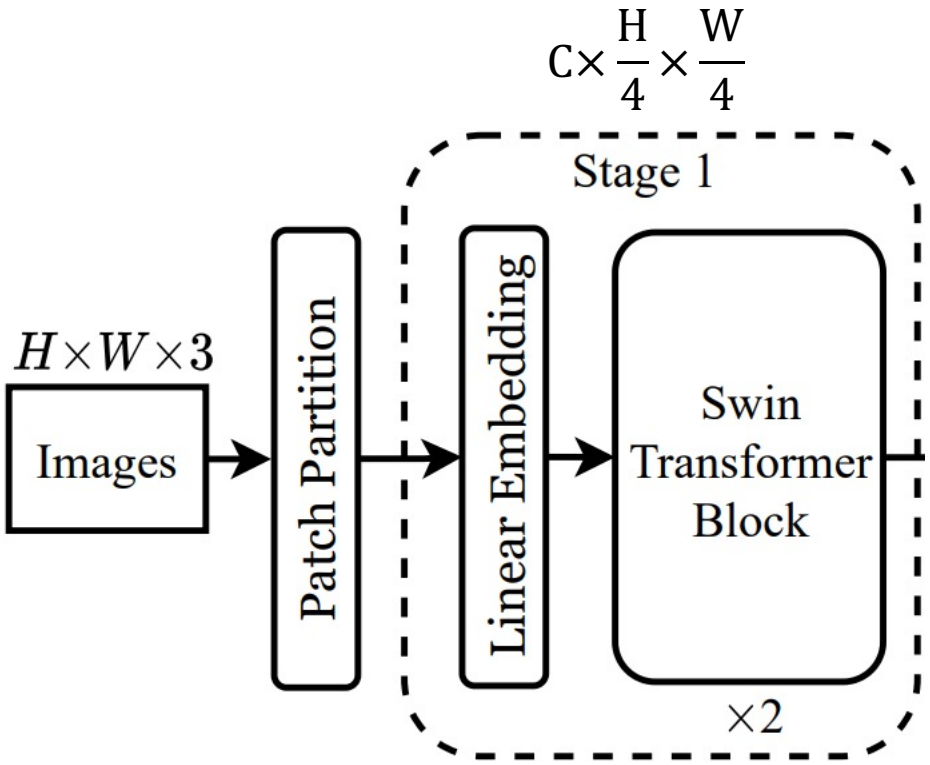
3rd Block:
768 x 14 x 14

2nd Block:
768 x 14 x 14

1st Block:
768 x 14 x 14

Input:
3 x 224 x 224

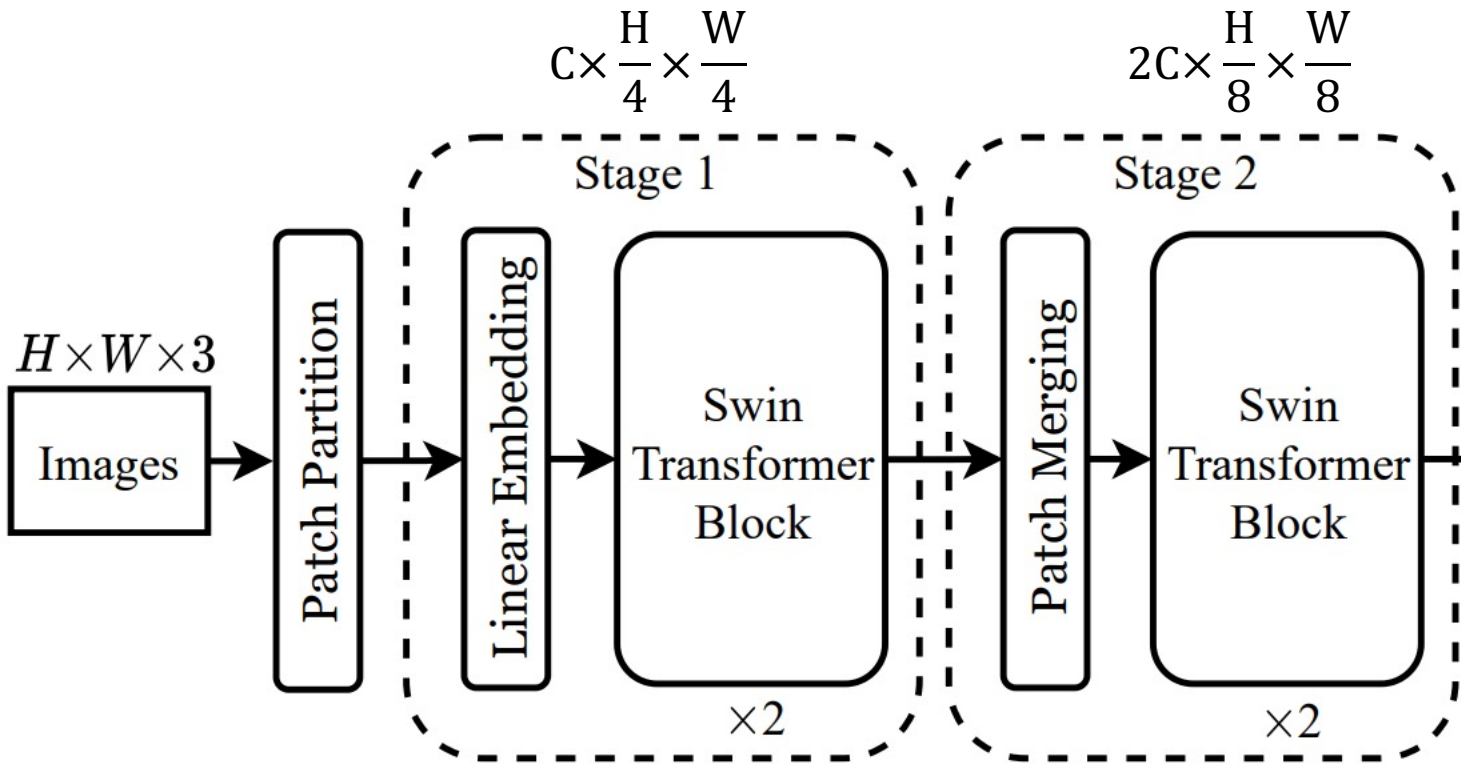
Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighbourhoods. Now patches are (effectively) 8x8

Patch Merging:

1. For every group of 2x2 patches, concatenate the corresponding 4 vectors.
2. Pass the concatenated vectors (of size 4C) through a linear layer to get vectors of size 2C.

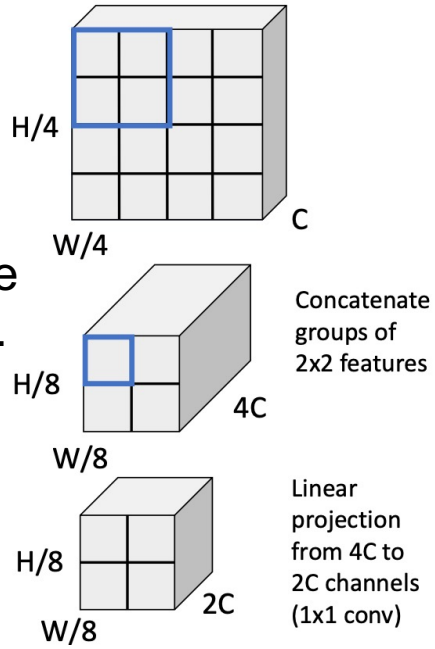
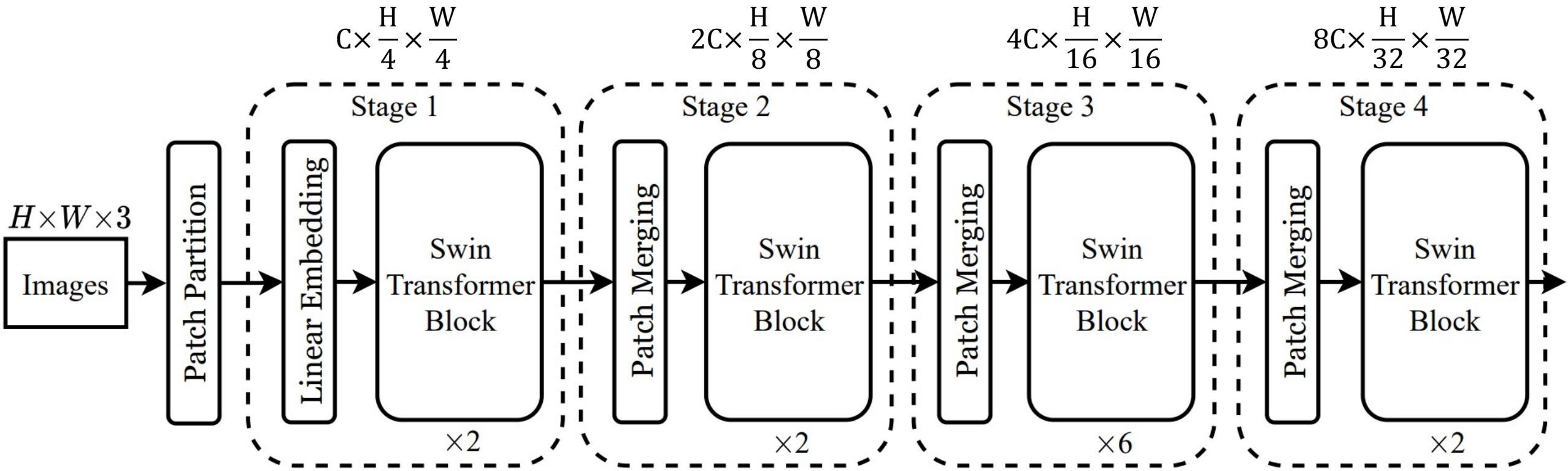


Image source

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighbourhoods. Now patches are (effectively) 8x8

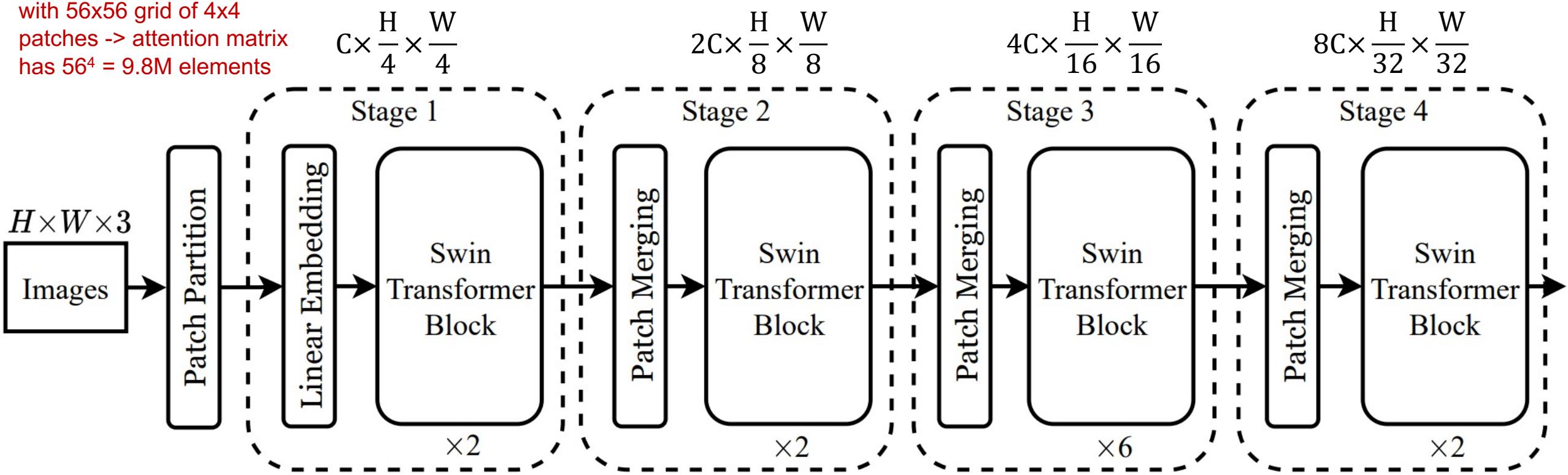
Merge 2x2 neighbourhoods. Now patches are (effectively) 16x16

Merge 2x2 neighbourhoods. Now patches are (effectively) 32x32

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

Problem: 224x224 image with 56x56 grid of 4x4 patches -> attention matrix has 56^4 = 9.8M elements



Solution: Limit the attention of a patch to its "window".

Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighbourhoods. Now patches are (effectively) 8x8

Merge 2x2 neighbourhoods. Now patches are (effectively) 16x16

Merge 2x2 neighbourhoods. Now patches are (effectively) 32x32

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention



Instead of allowing each token to attend to all other tokens, divide image into M x M **windows** (here M=4).

Compute attention only within each window.

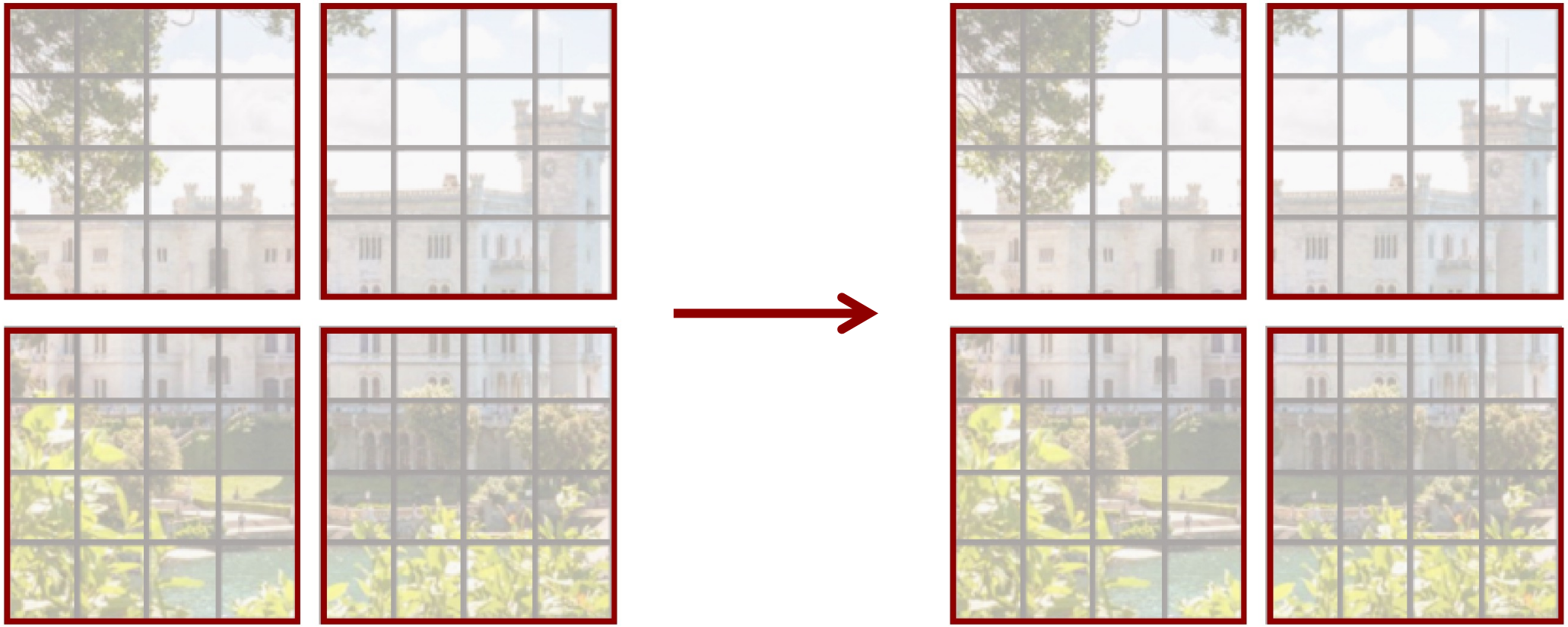
A local window to perform self-attention

A patch

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

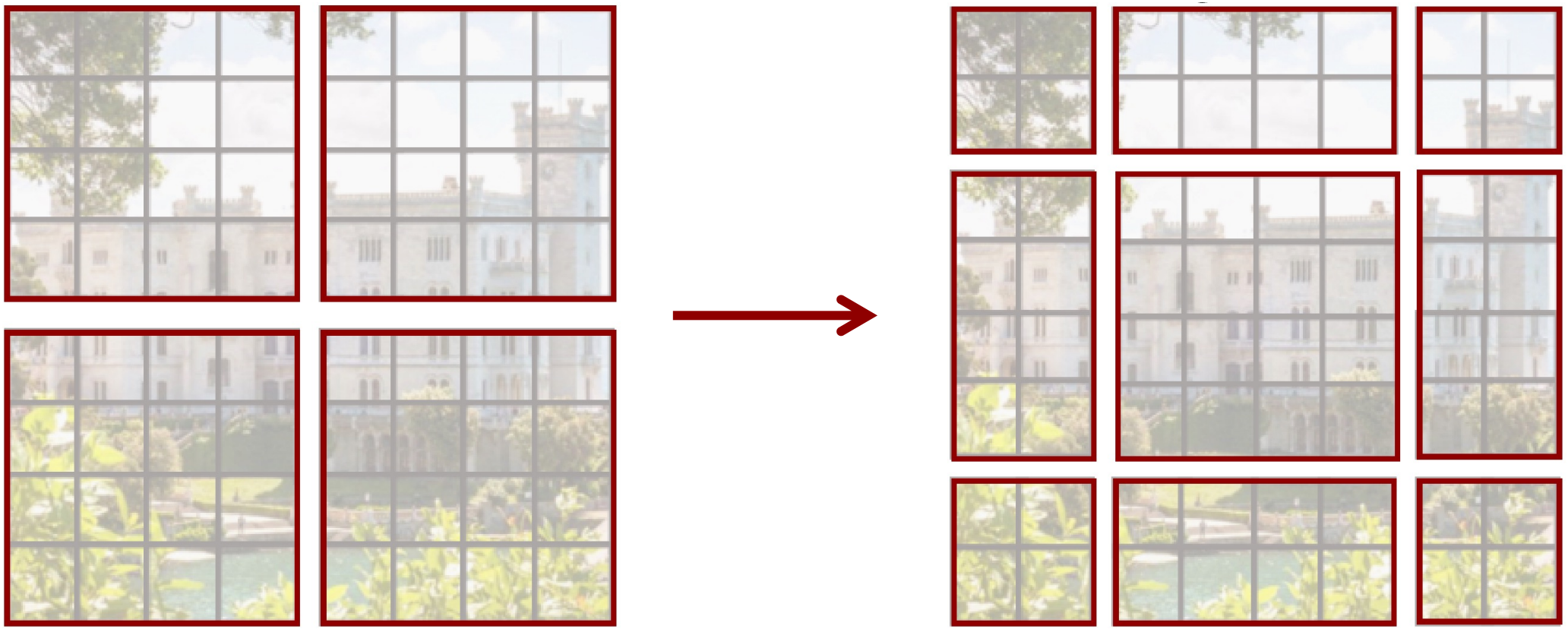
Problem: No communication across windows. Global context?



Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks.

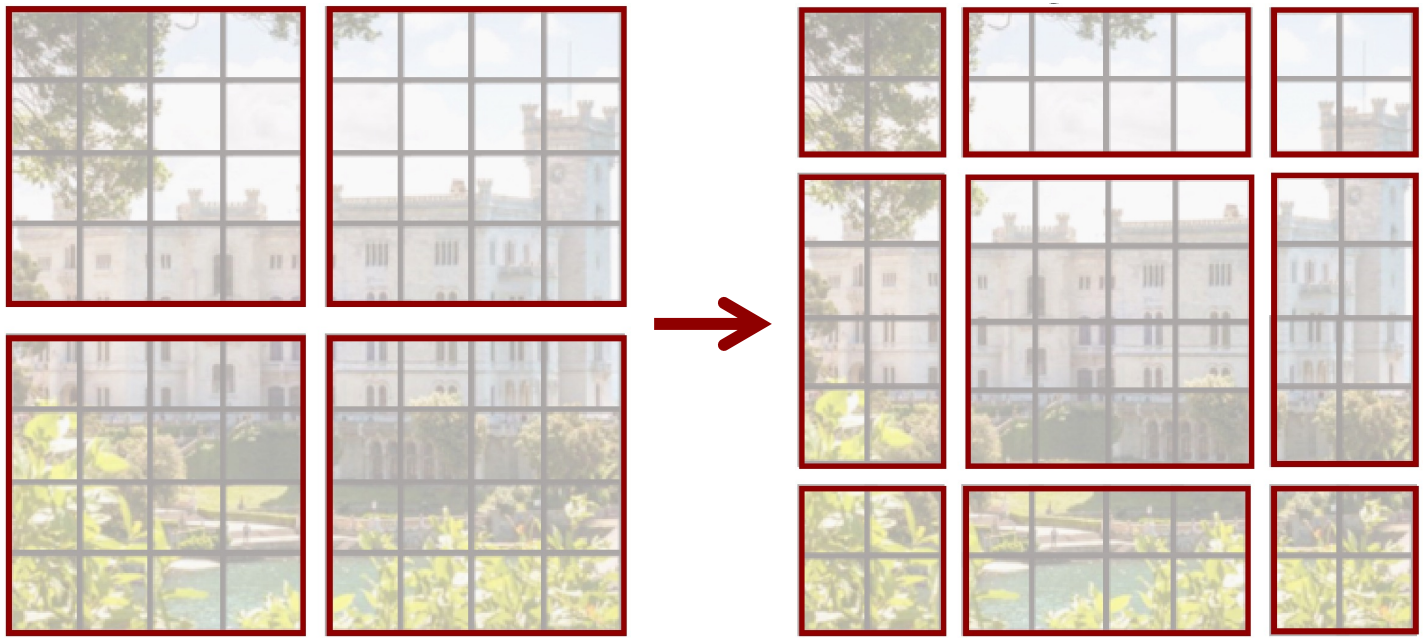


Block L: Normal windows

Block L+1: Shifted windows

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks.



Block L: Normal windows

Block L+1: Shifted windows

Block L

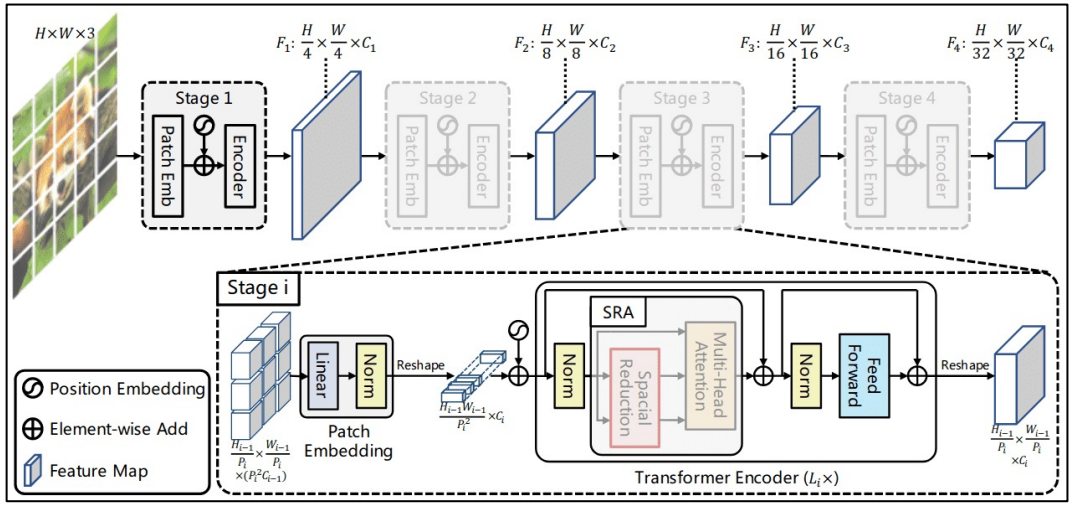
$$\hat{\mathbf{z}}^l = \text{W-MSA}\left(\text{LN}\left(\mathbf{z}^{l-1}\right)\right) + \mathbf{z}^{l-1},$$
$$\mathbf{z}^l = \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^l\right)\right) + \hat{\mathbf{z}}^l,$$

$$\hat{\mathbf{z}}^{l+1} = \text{SW-MSA}\left(\text{LN}\left(\mathbf{z}^l\right)\right) + \mathbf{z}^l,$$
$$\mathbf{z}^{l+1} = \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l+1}\right)\right) + \hat{\mathbf{z}}^{l+1},$$

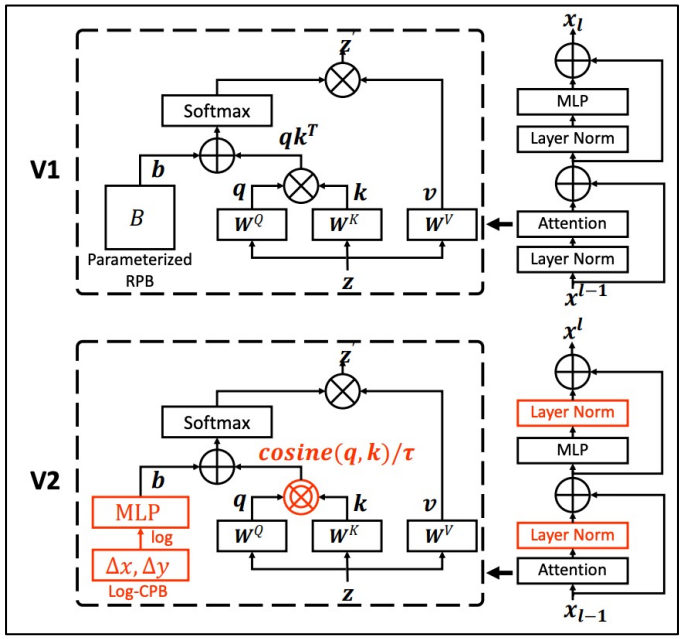
Block L+1

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

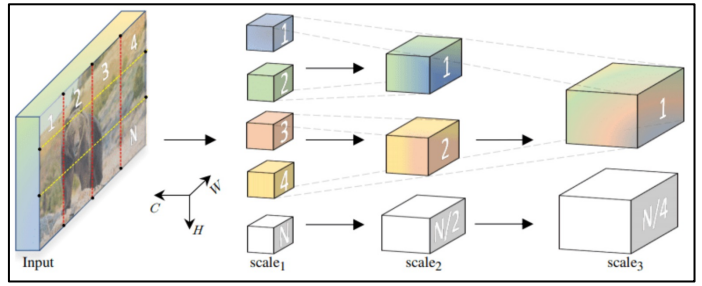
Other hierarchical vision transformers



Wang, Wenhai, et al. "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions.", ICCV 2021



Liu, Ze, et al. "Swin transformer v2: Scaling up capacity and resolution.", arXiv 2021, CVPR 2022



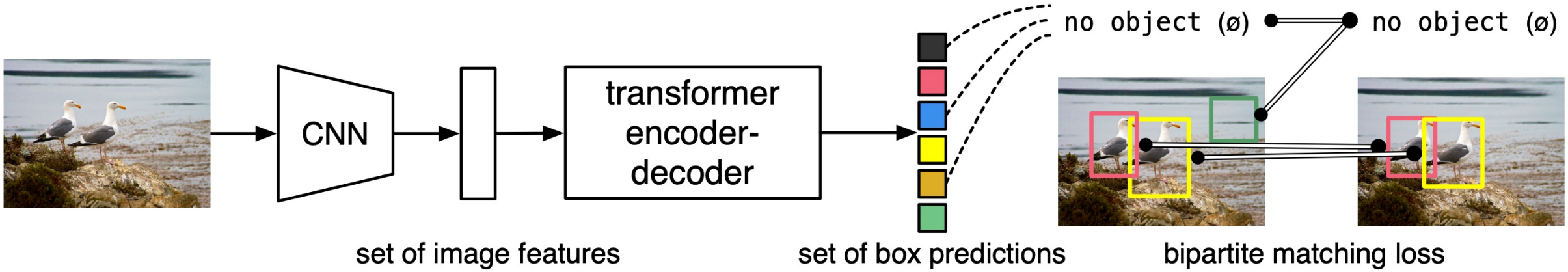
Fan, Haoqi, et al. "Multiscale vision transformers.", ICCV 2021

Outline

- Self-attention
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- Transformer encoder block
- Transformer decoder block
- Vision transformer (ViT)
- Swin transformer
- **DEtection TRansformer (DETR) for object detection**
- Suggested resources

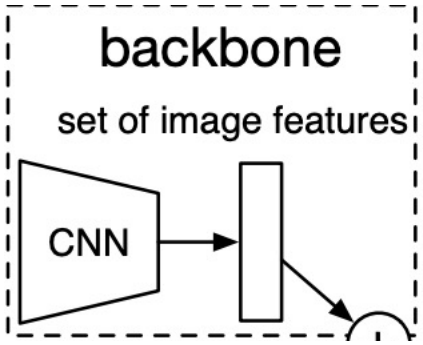
Object detection with transformers: DETR

- Simple object detection pipeline: directly output a set of boxes from a Transformer.
- Anchor-free approach, i.e., no pre-defined anchors are used.
- No Non-Maximum Suppression (NMS) used. It is learned by the transformer automatically.
- Match predicted boxes to ground truth boxes with bipartite matching.



Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object detection with transformers: DETR

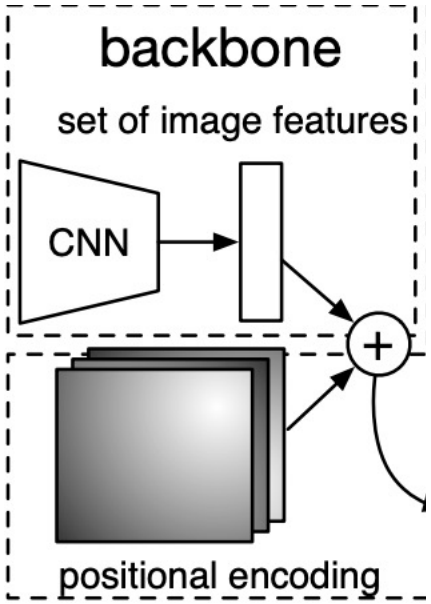


A backbone ConvNet (like ResNet50) is used to extract features from the input.

1x1 conv layer is used to reduce the number of channels.

Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object detection with transformers: DETR



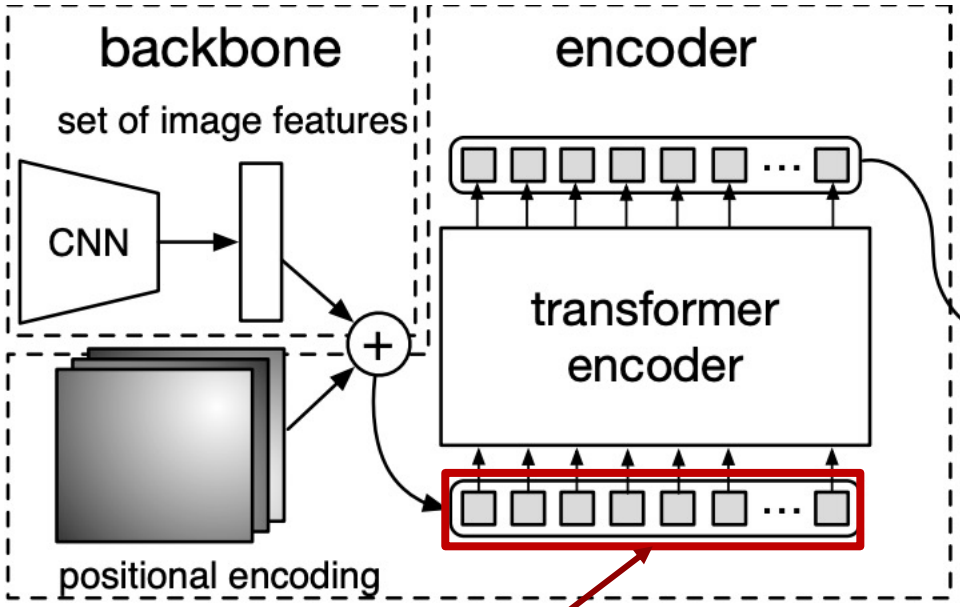
A backbone ConvNet (like ResNet50) is used to extract features from the input.

1x1 conv layer is used to reduce the number of channels.

Fixed positional encodings are added to every encoder block's input

Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object detection with transformers: DETR

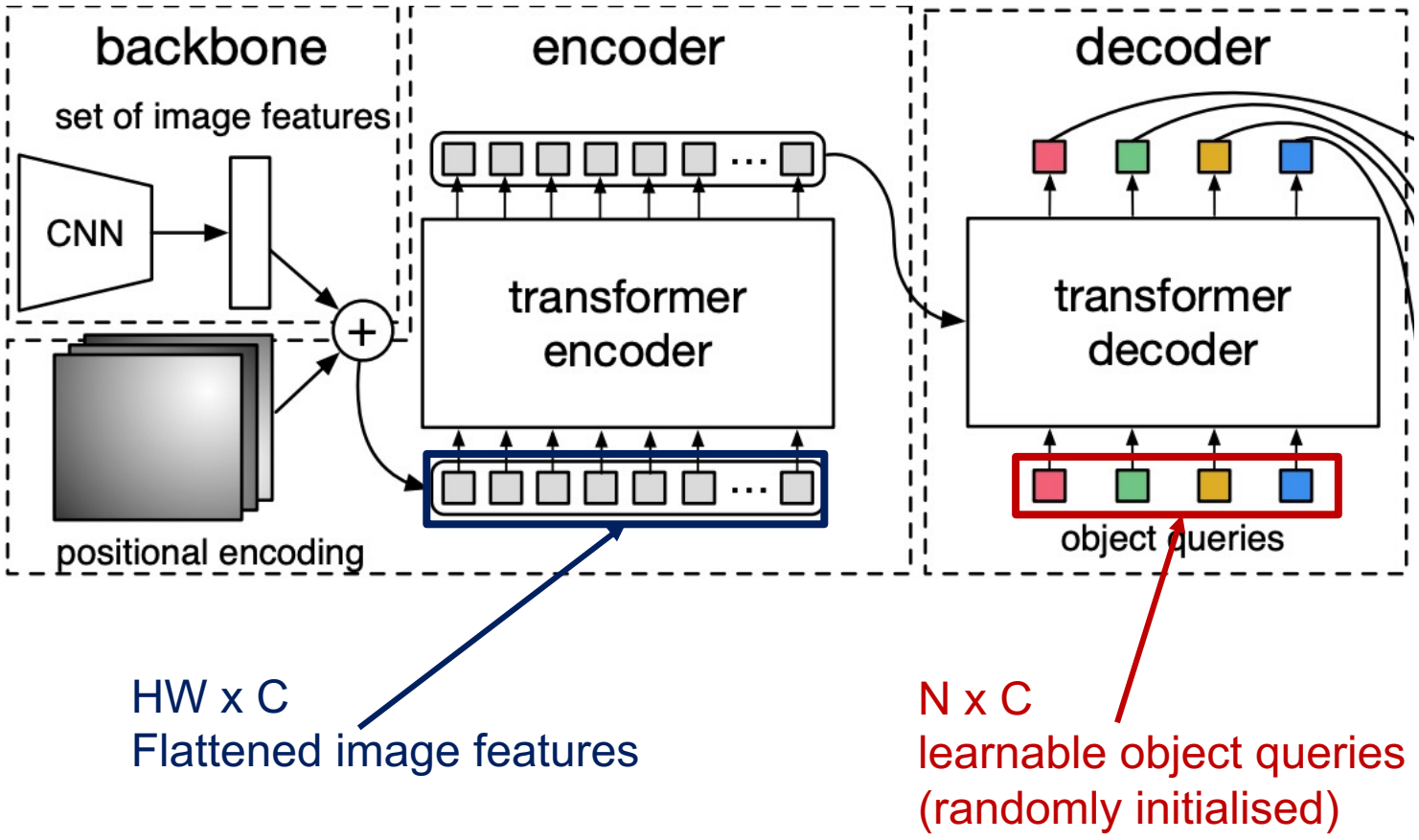


Standard transformer encoder is used to process the flattened image features.

HW x C
Flattened image features

Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object detection with transformers: DETR

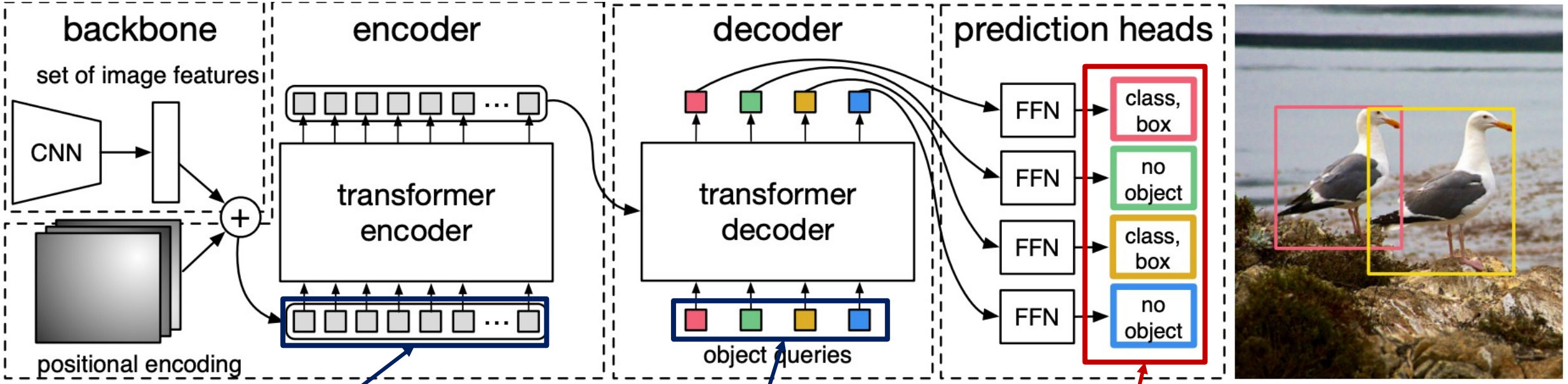


A standard transformer decoder is used to generate outputs.

Input: N object queries, encoder's output

Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object detection with transformers: DETR



HW x C
Flattened image features

N x C
learnable object queries
(randomly initialised)

Each object query predicts one bounding box

Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Is self-attention magical and absolutely necessary?

- A lot of works have explored replacing self-attention with **X** and some of them achieve competitive results.
  - Wu et al replace self-attention with convolution.
  - Tay et al replace alignment scores in self-attention with random projection.
  - Tolstikhin et al replace self-attention with an MLP-mixer making the architecture exclusively based on MLPs.
  - Lee-Thorp et al replace self-attention with Fourier Transforms.
  - Tatsunami and Taki replace self-attention with LSTM.

- But, self-attention continues to be the standard choice.

Outline

- Self-attention
  - Positional encoding
  - Masked self-attention
- Multi-head self-attention
- Transformer encoder block
- Transformer decoder block
- Vision transformer (ViT)
- Swin transformer
- DEtection TRansformer (DETR) for object detection
- **Suggested resources**

Suggested resources

- The Illustrated Transformer: A famous blog post covering the basics of Transformers.

- The Annotated Transformer: A blog post that implements Transformer line by line with explanation.