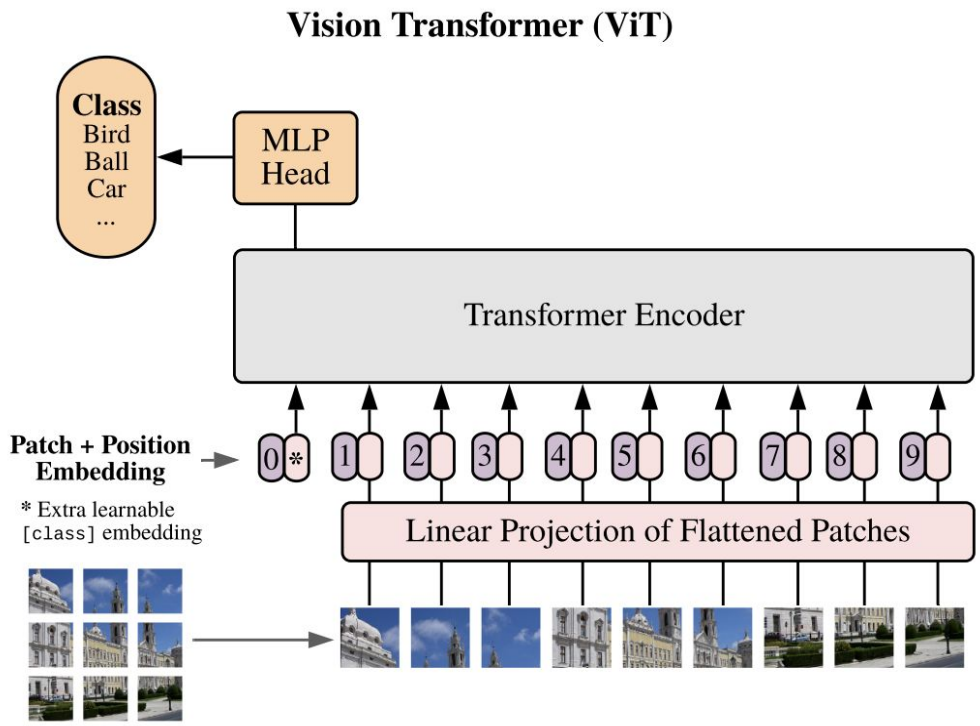
Announcements

- Challenge released.
- No laptops during class.
- Special access code to TorchStack: QYHBEGH
  - https://torchstack.dev/

Last week

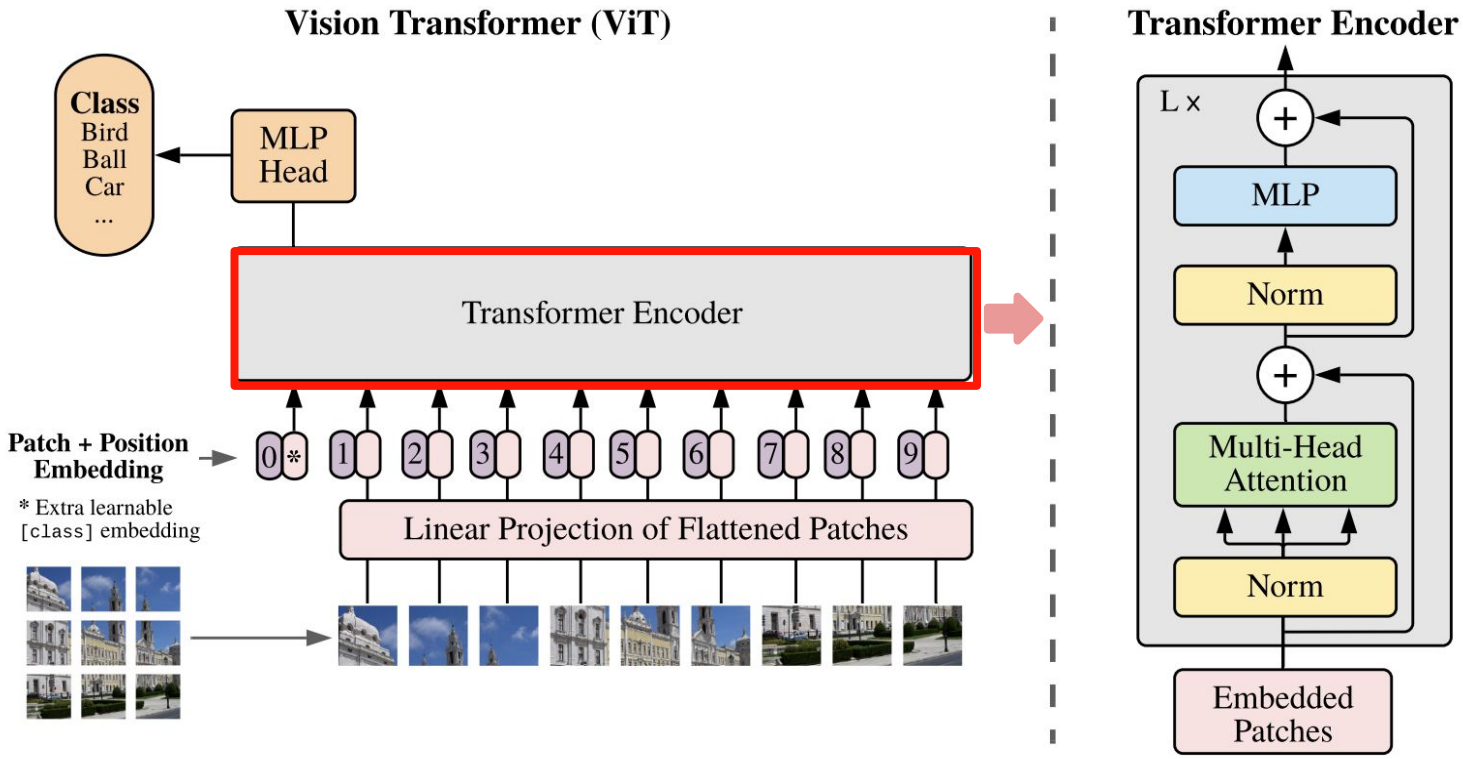
- Scaling up of transformers
- Extending transformers to computer vision

Recall: Transformers for vision



AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE, ICLR 21

Recall: Transformers for vision



AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE, ICLR 21

Today

- Positional embeddings in transformers
- Model interpretability

Today

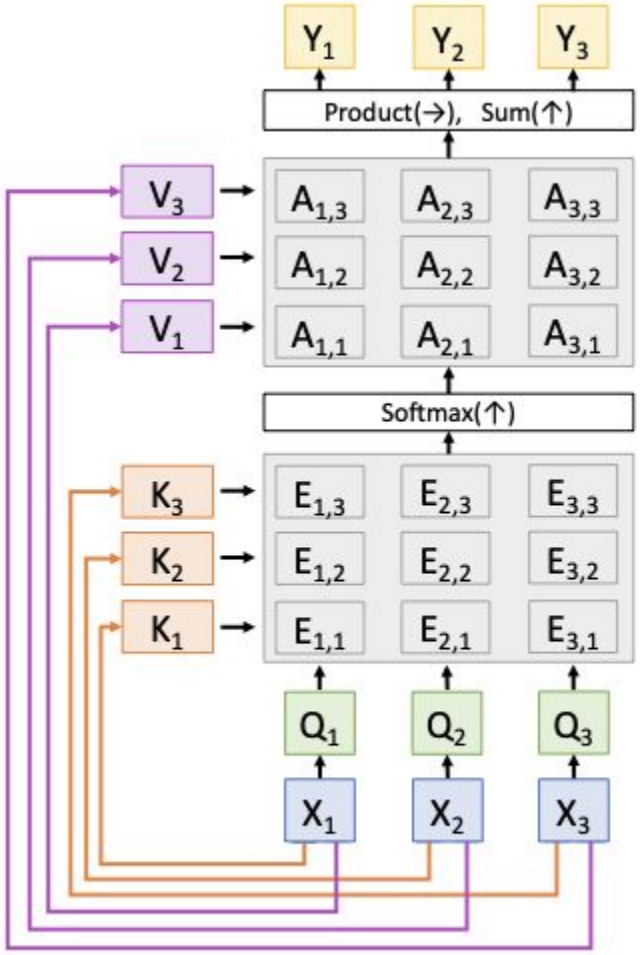
- **Positional embeddings in transformers**
- Model interpretability

Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = AV (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

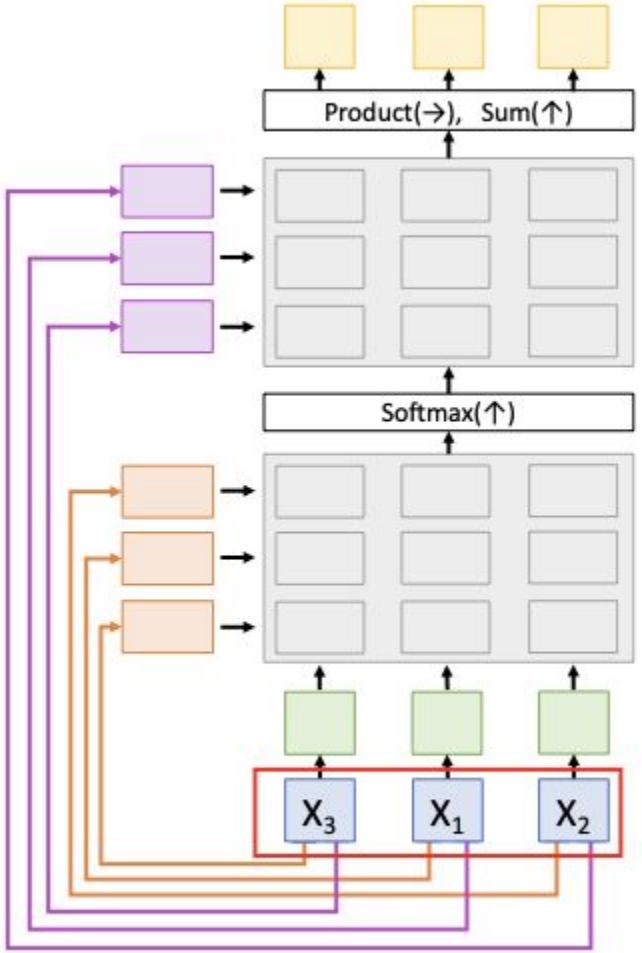


Self-Attention Layer

Consider **permuting** the input vectors:

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



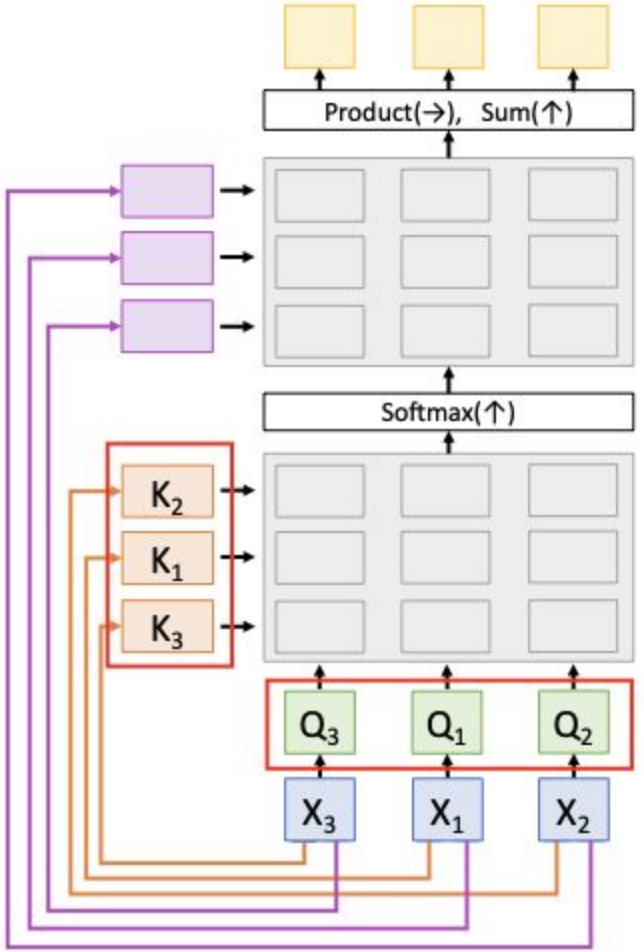
Self-Attention Layer

Consider **permuting** the input vectors:

Queries and Keys will be the same, but permuted

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



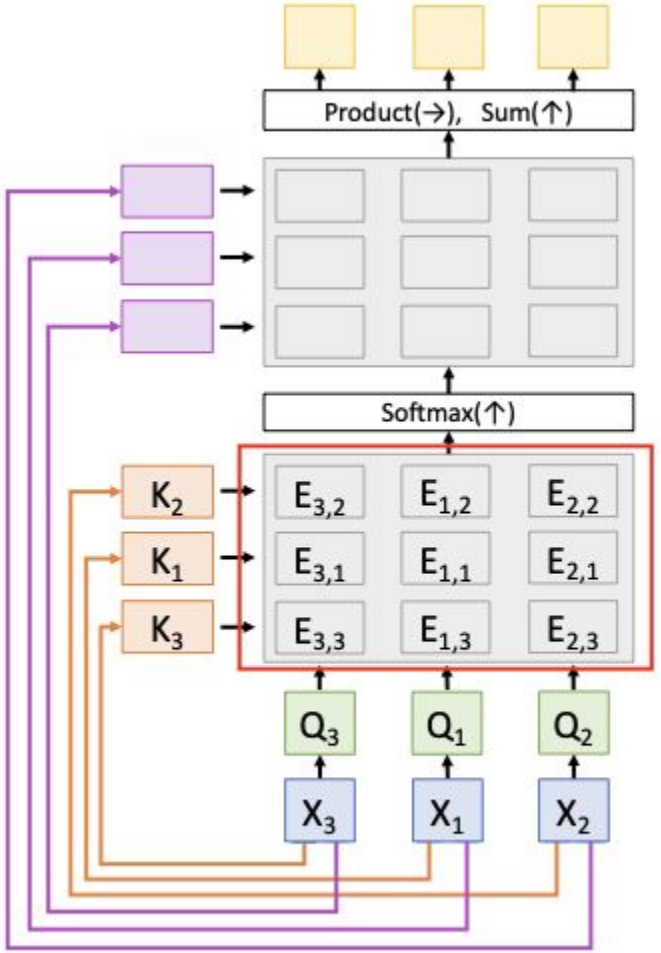
Self-Attention Layer

Consider **permuting** the input vectors:

Similarities will be the same, but permuted

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



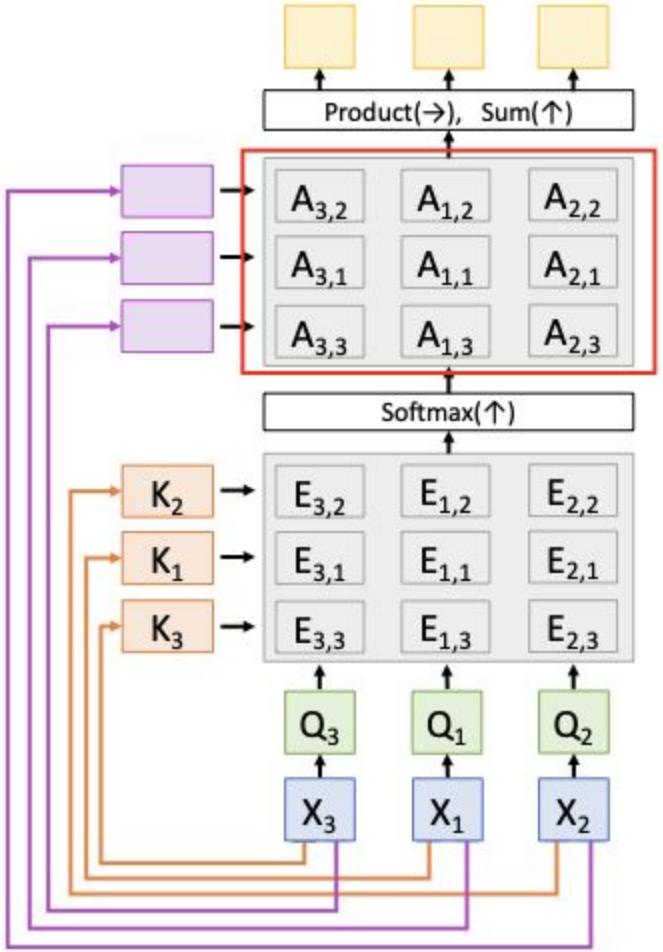
Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



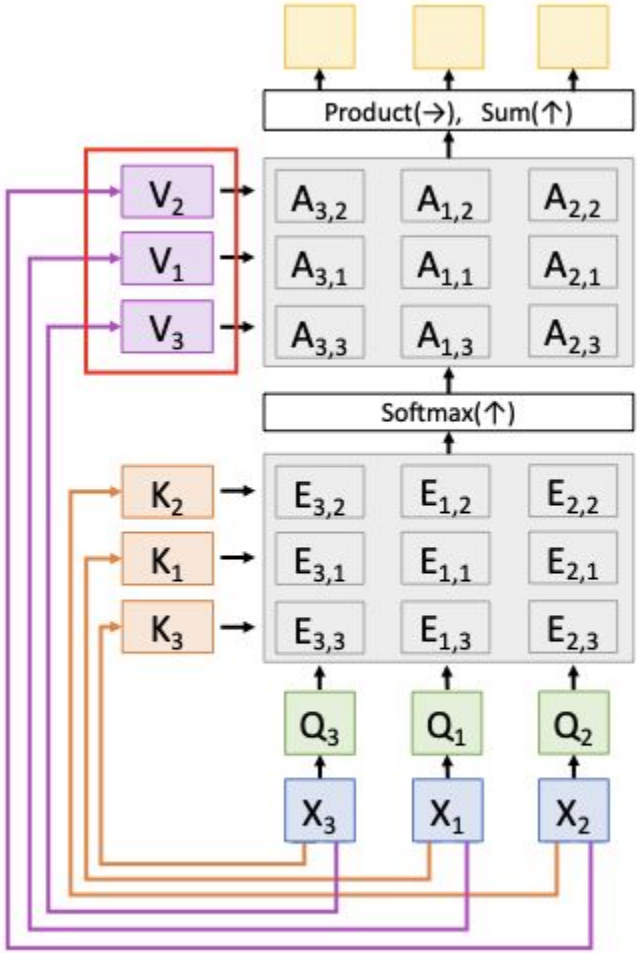
Self-Attention Layer

Consider **permuting** the input vectors:

Values will be the same, but permuted

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



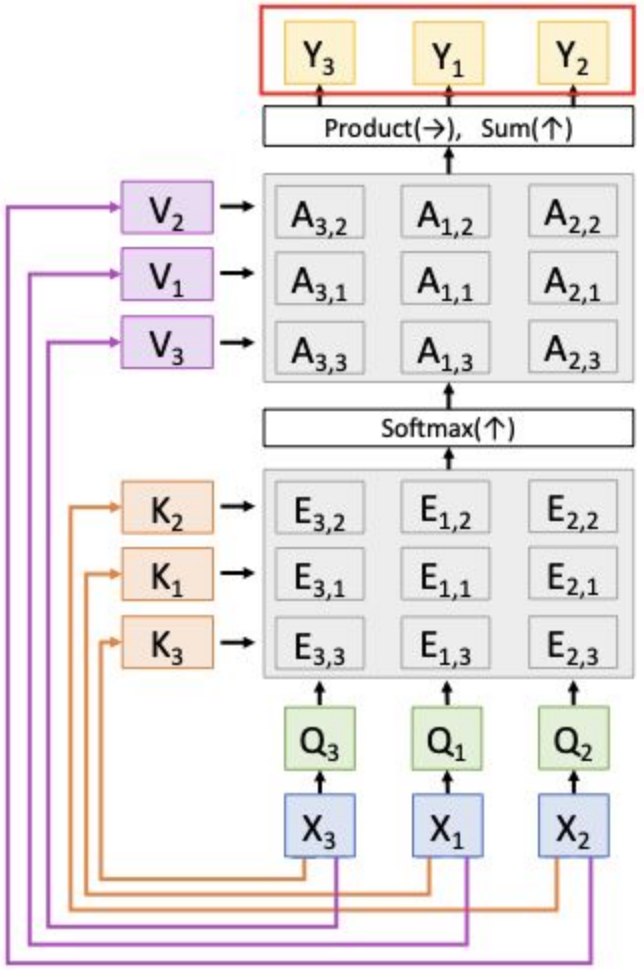
Self-Attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

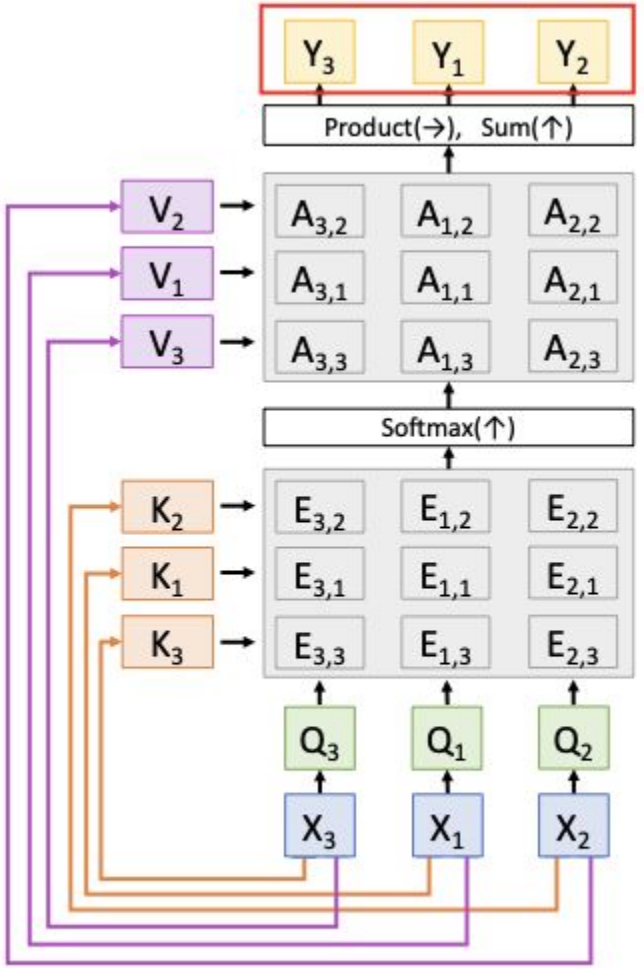
Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant**
$f(s(x)) = s(f(x))$

Self-Attention layer works on **sets** of vectors

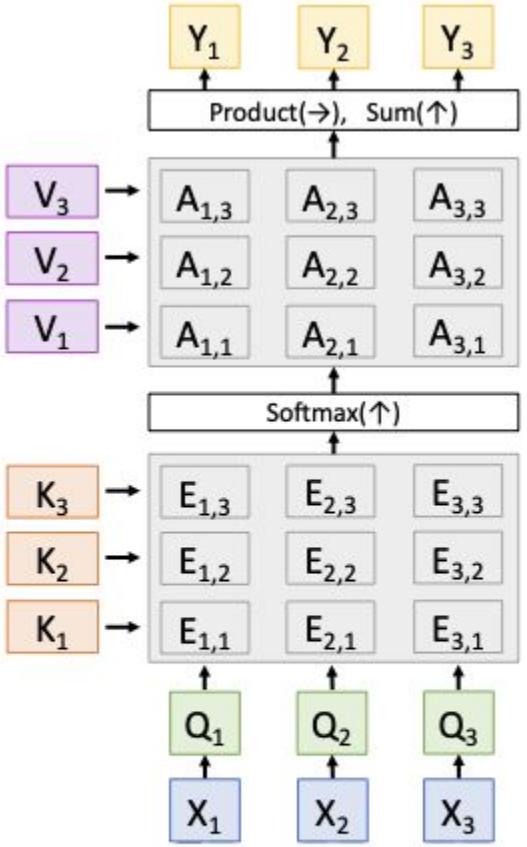


Self-Attention Layer

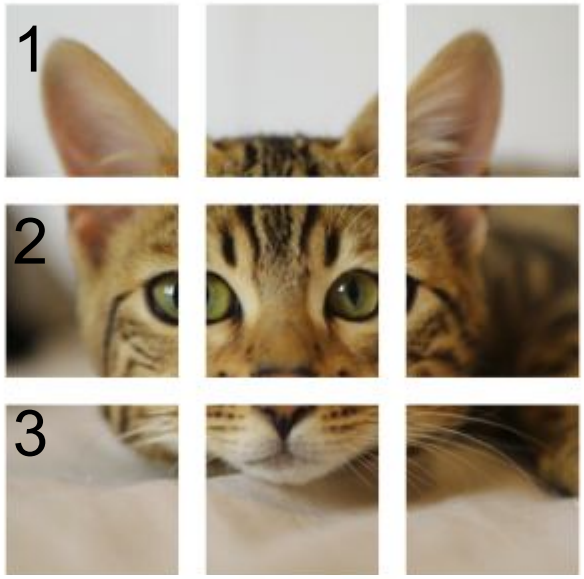
Self attention doesn't "know" the order of the vectors it is processing!

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = AV (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Images…

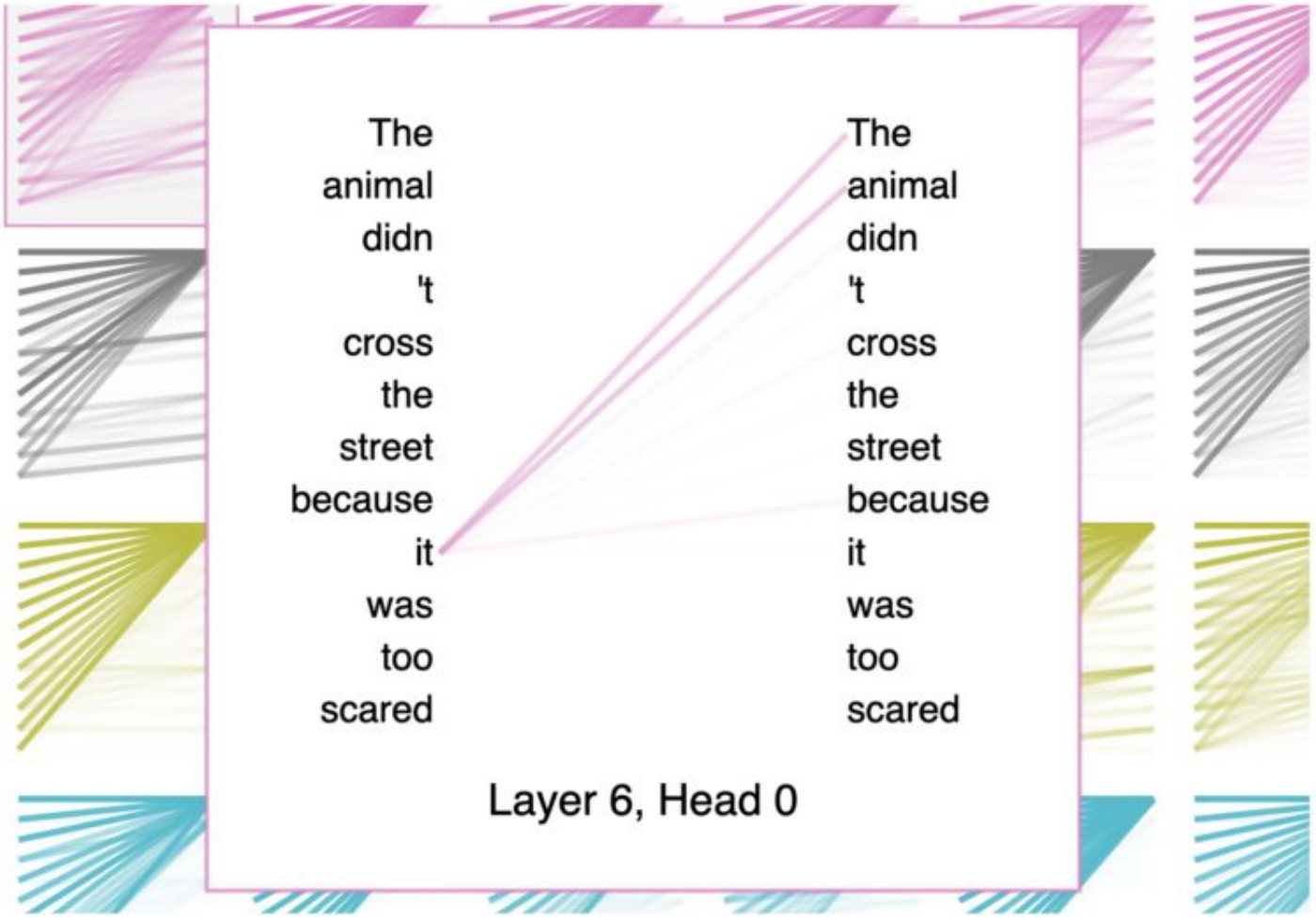


One ordering

Shuffled

- Would the model know that the input is shuffled? No

Should the model know the order of the input tokens? Why or why not?



Input sentence: The animal didn't cross the street because it was too scared.

Shuffled input: didn't street it it too The cross animal the because scared too

Summary so far

- *Ordering of the sequence captures inherent semantic information in text, audio, image, and video.*

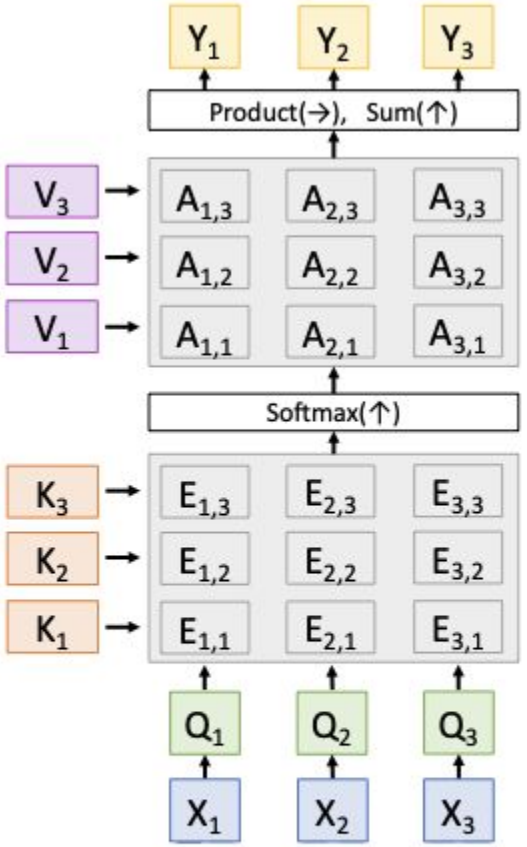
- *It is crucial to preserve the ordering.*

Self-Attention Layer

Self attention doesn't "know" the order of the vectors it is processing!

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

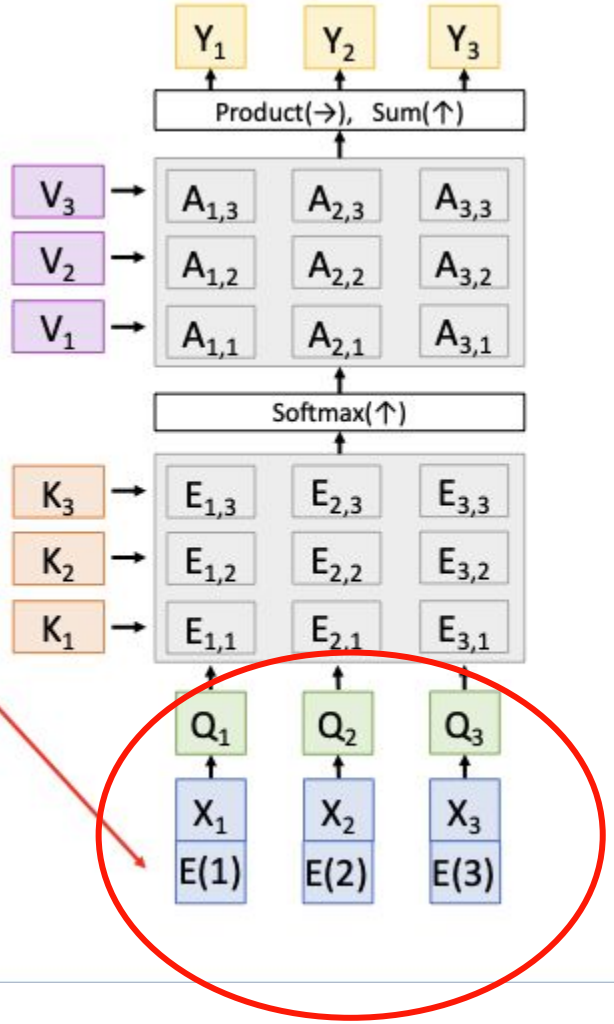
Self attention doesn't "know" the order of the vectors it is processing!

In order to make processing position-aware, concatenate or add **positional encoding** to the input

E can be learned lookup table, or fixed function

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = AV (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

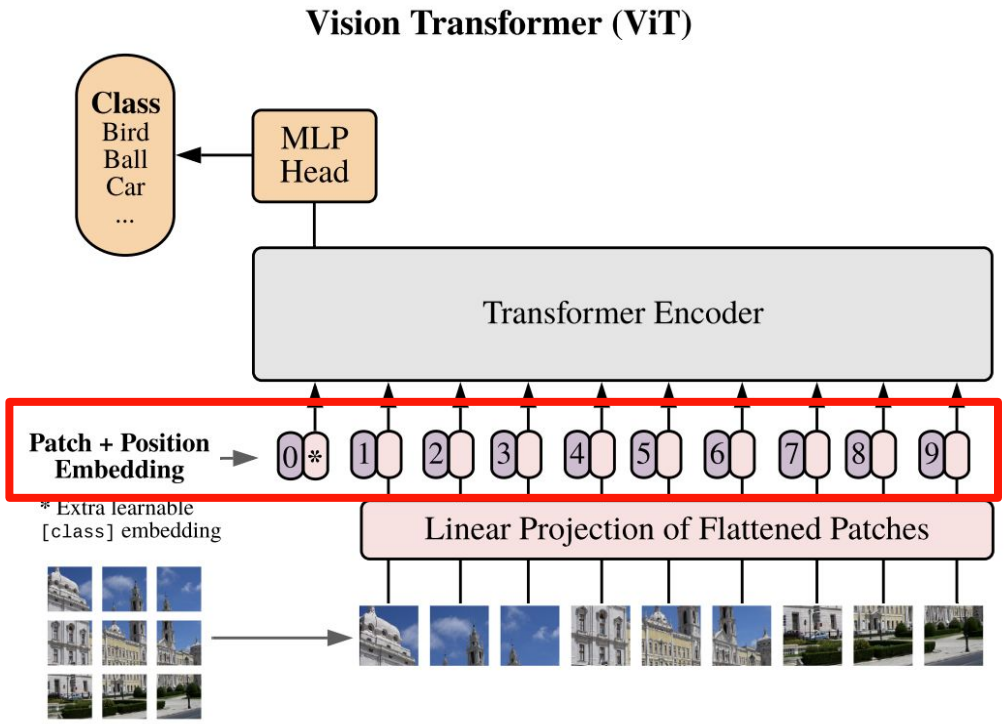


Positional encodings == identifiers that preserve the ordering.

Summary so far

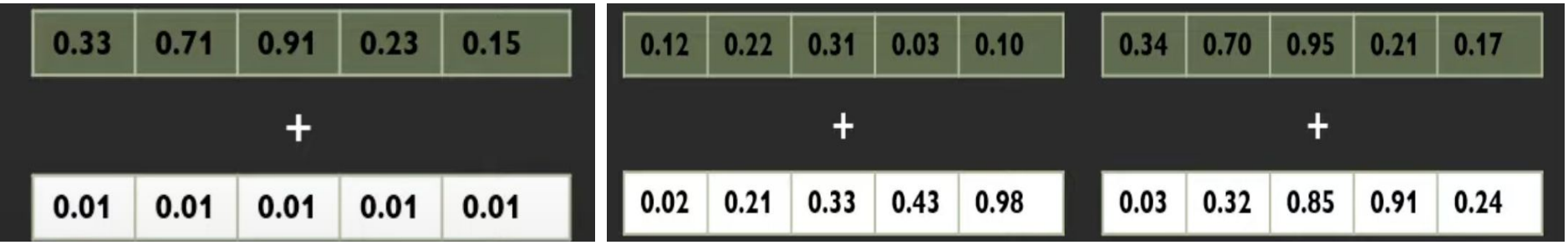
- *Ordering of the sequence captures inherent semantic information in text, audio, image, and video.*
- *It is crucial to preserve the ordering.*
- ***Positional embeddings:*** *Help remember the order of the input.*

How to incorporate positional embedding?



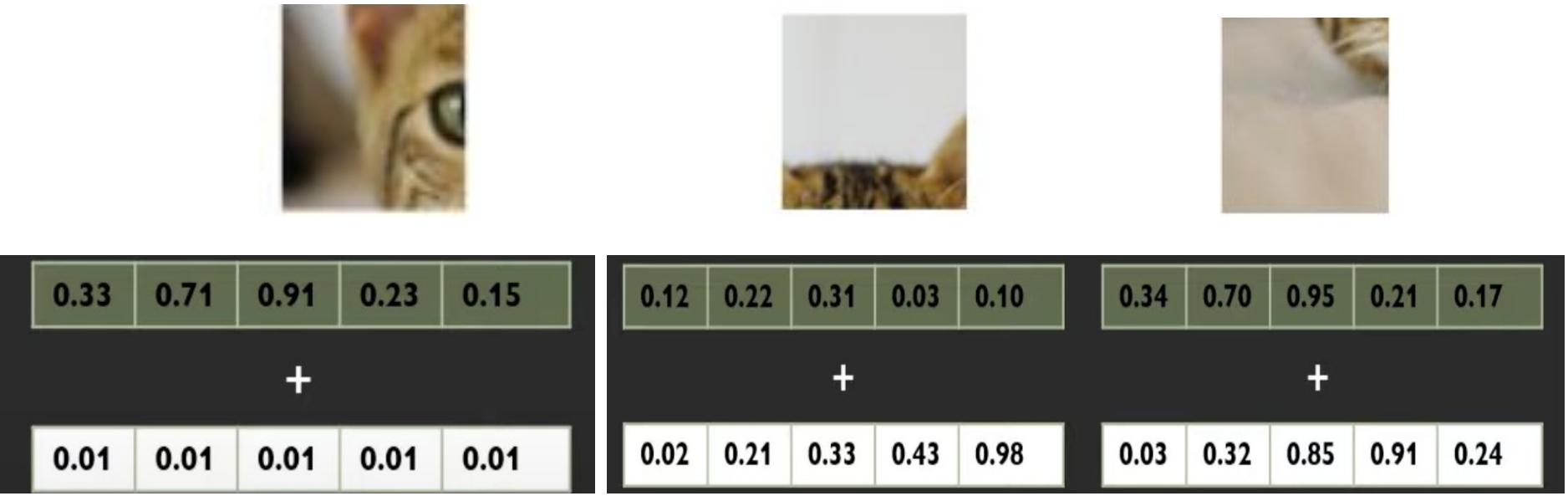
AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE, ICLR 21

Add to the input token embeddings

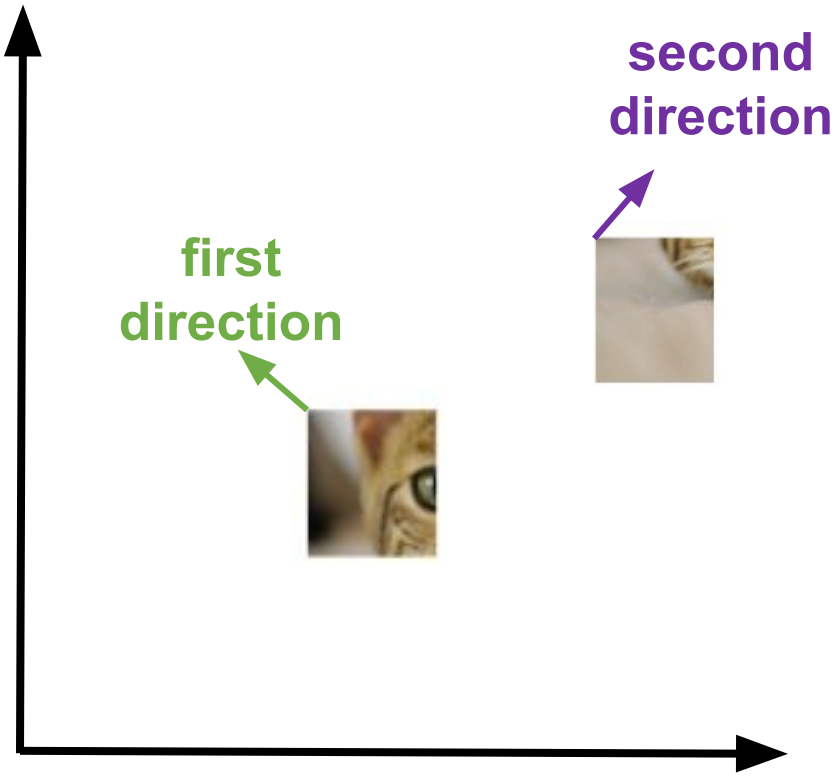


Add to the input token embeddings

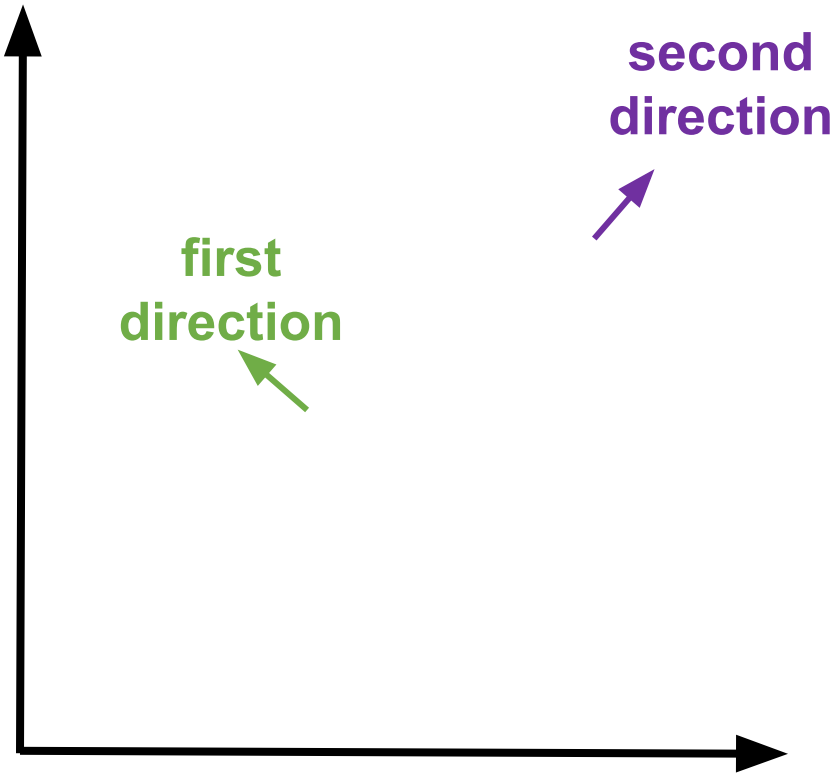
✅ Retain the feature dimensionality.

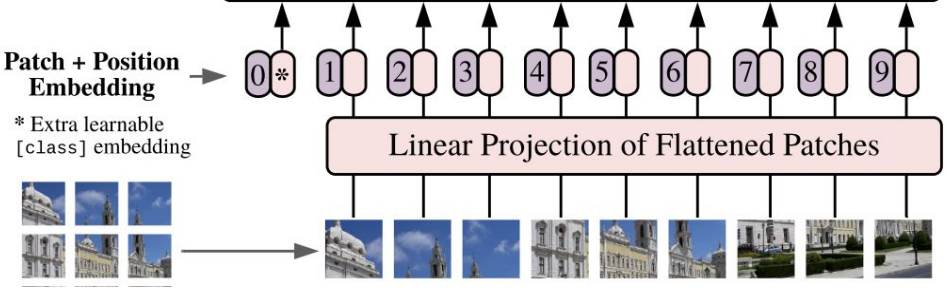


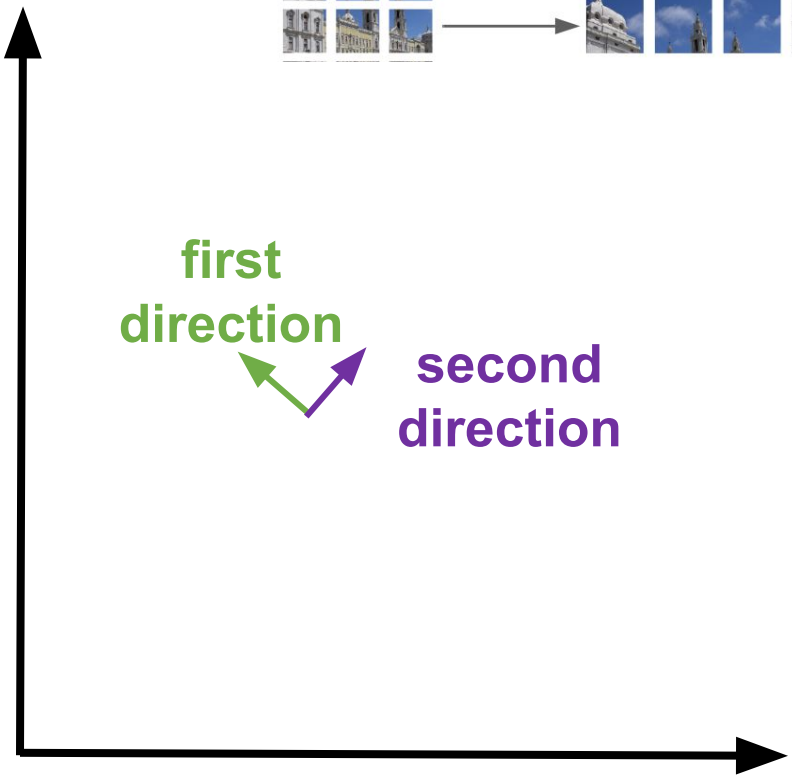
Intuition behind addition of positional encodings



Positional embeddings



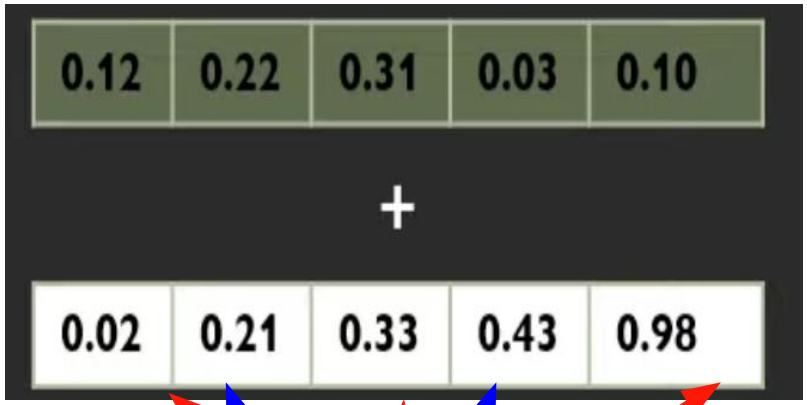




✅ Retain the dimensionality of input token.
✅ Not too high or too low of a magnitude.

Commonly used positional encodings

- Sinusoidal functions
- Cosine functions

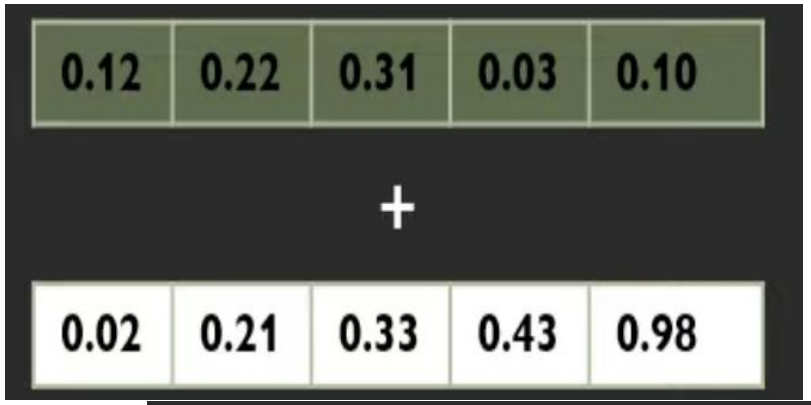


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

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

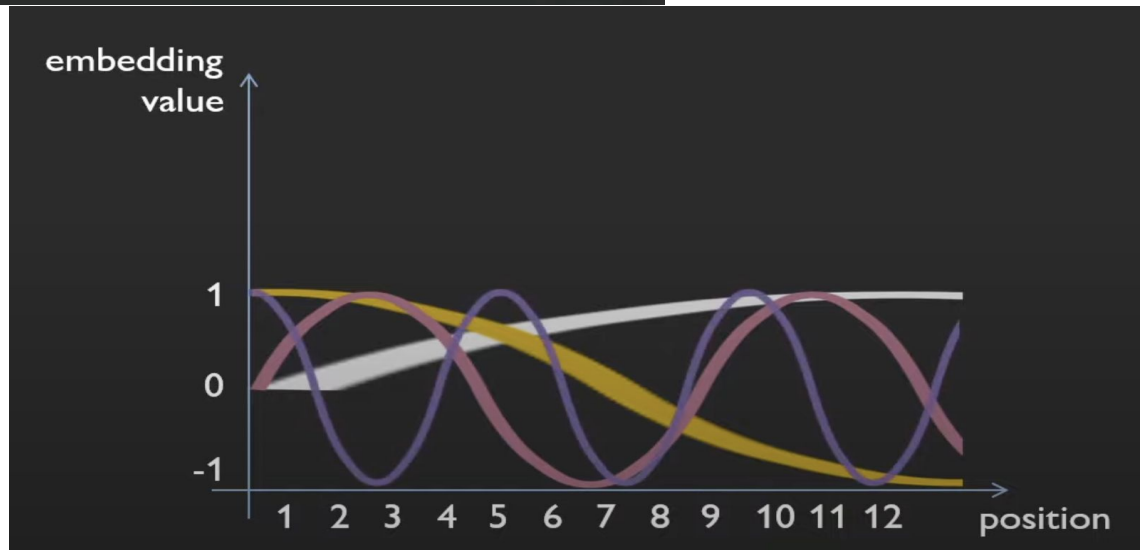
Commonly used positional encodings

- Sinusoidal functions
- Cosine functions



$$PE_{(pos,2i)}$$

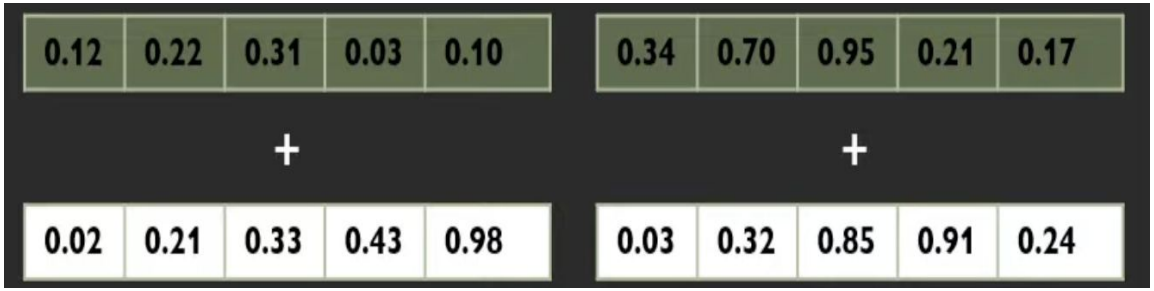
$$PE_{(pos,2i+1)}$$



What are the upsides of using sinusoidal / cosine functions as positional embeddings?

Are there any downsides of using sine and cosine functions as positional embeddings?

Downsides of adding positional embeddings

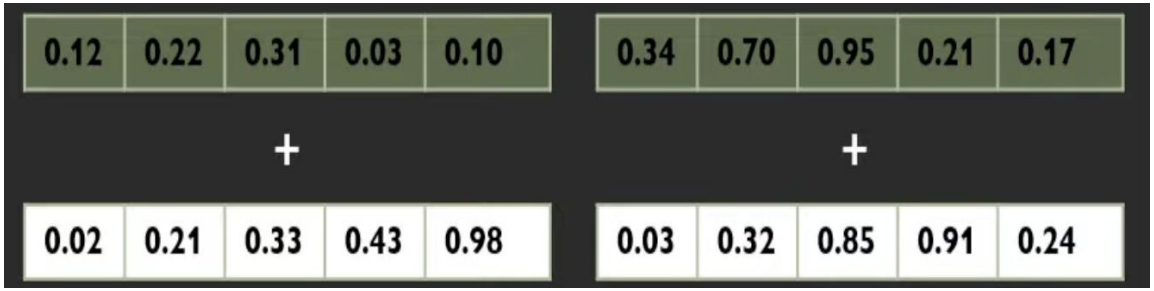


✅ Retain the dimensionality of input token.
✅ Not too high or too low of a magnitude.
❌ Mixing up semantic information with positional information

Summary so far

- ***Positional embeddings:*** *Help remember the order of the input.*
  - ***Addition:***
    - Retains the feature dim
    - Mixes up semantic information with positional information

Downsides of adding positional embeddings



✅ Retain the dimensionality of input token.
✅ Not too high or too low of a magnitude.
❌ Mixing up semantic information with positional information

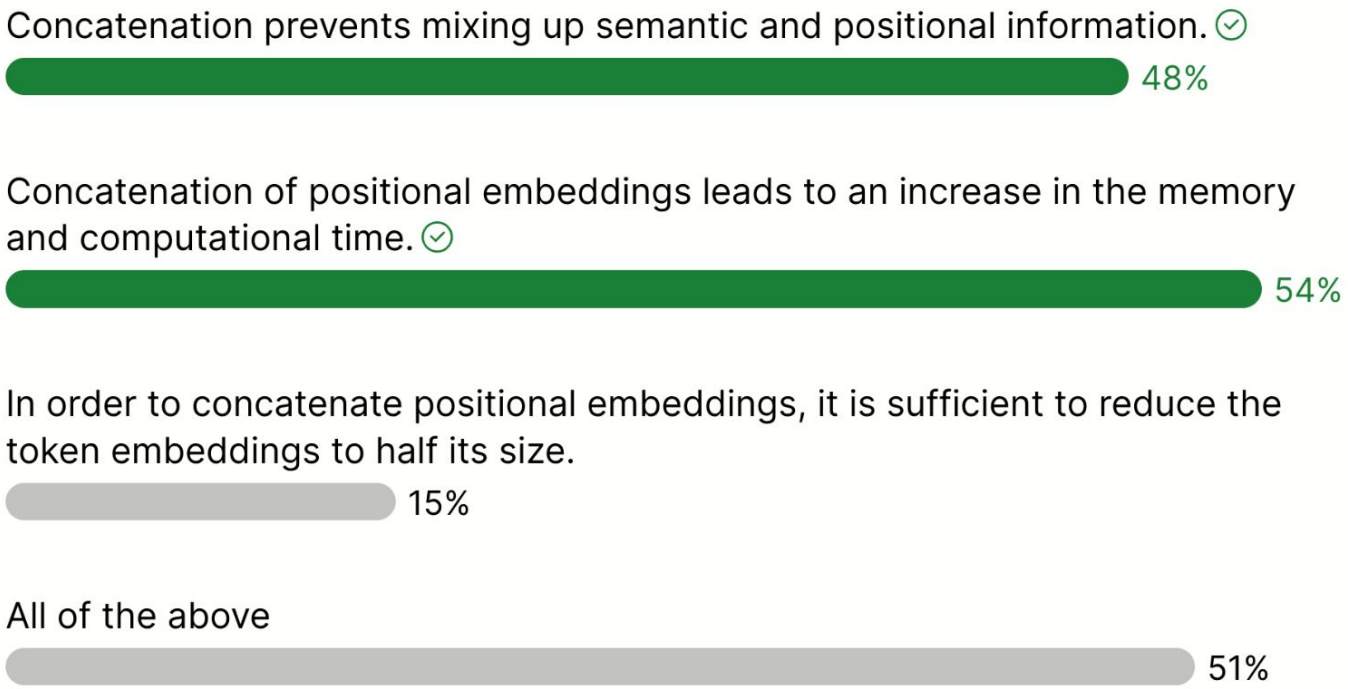
💡 How about concatenation?

Concatenation of positional embeddings

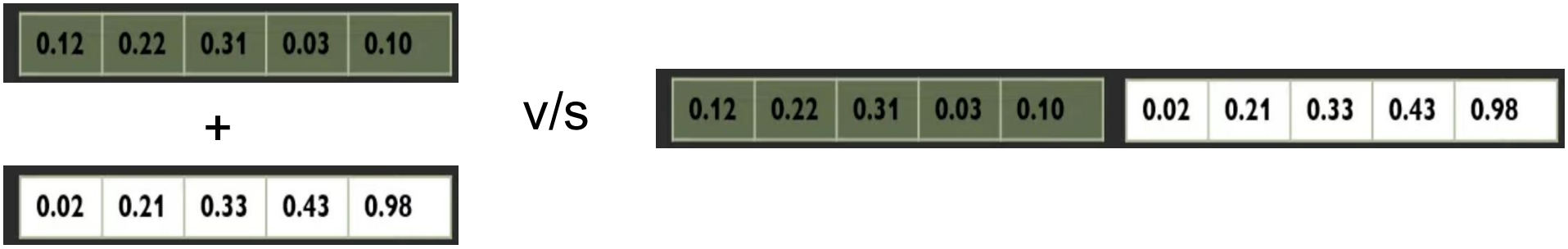
Which of the following is true about concatenating positional embeddings? Select all that apply

Join at
slido.com
#2812 878

Which of the following is true about concatenating positional embeddings? Select all that apply



Concatenation of positional embeddings



Studied heavily in computer vision

Pros:

- Add an additional independent direction.
- Does not mix up semantic and positional information.

Cons:

- Increases compute / memory footprint.

Summary so far

- ***Positional embeddings:*** *Help remember the order of the input.*
  - ***Addition:***
    - Mixes up semantic information with positional information
  - **Concatenation:**
    - Does not mix up the signals
    - Increases compute / memory footprint.

Other options

- Learn your positional embeddings along with the task.
- Relative positional embeddings
  - based on semantic distance between the tokens
- RoPE
  - embedding + rotated variant of the embedding.

  .

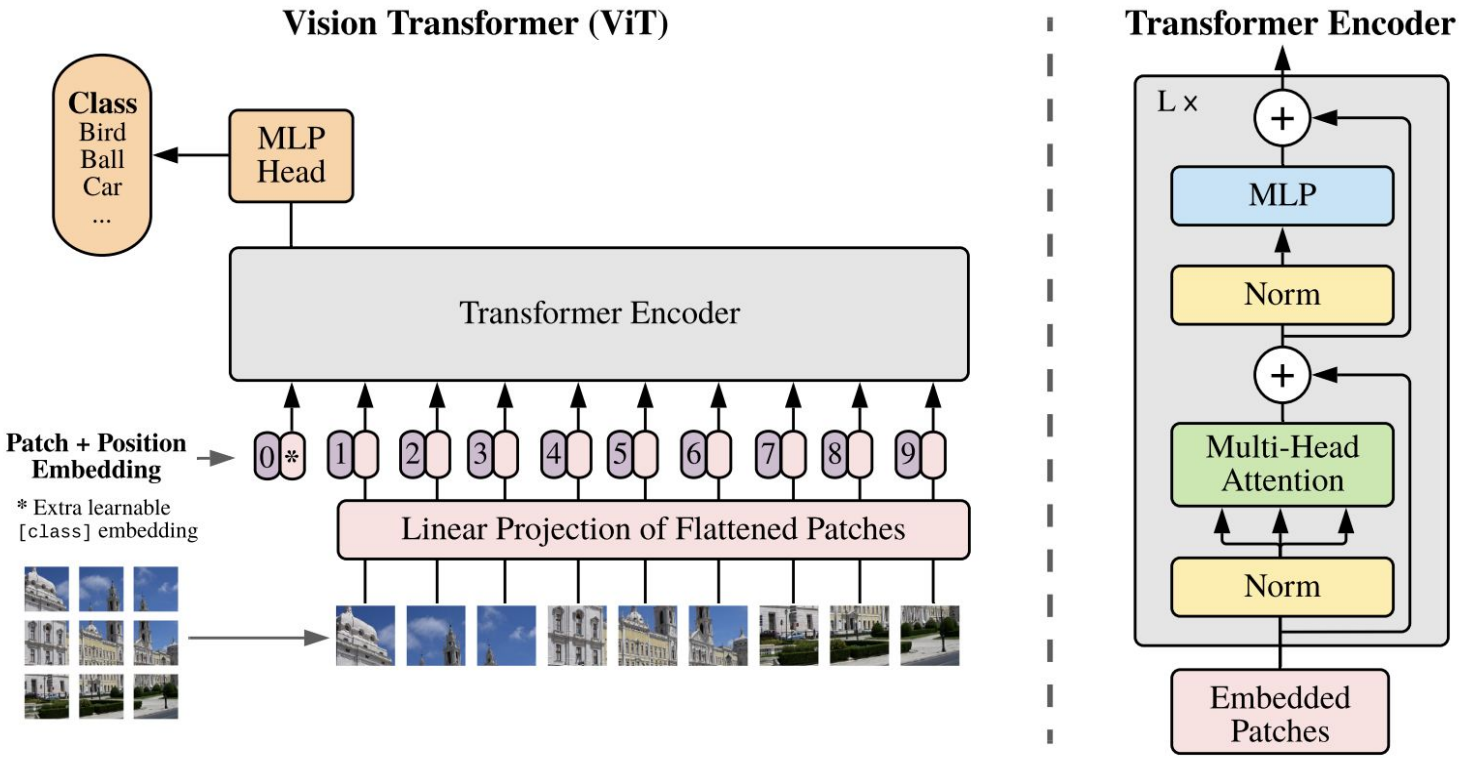
  .

  .

  .

- A very active area of research!

Summary



✅ Very powerful architecture.

⚠️ Very data hungry - will not recommend unless there is sufficient data even for fine tuning tasks.

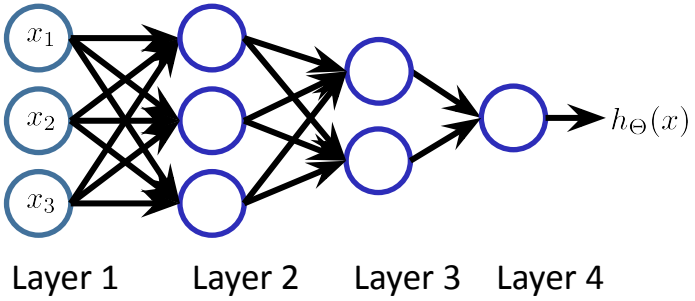
Today

- Positional embeddings in transformers
- **Model interpretability**

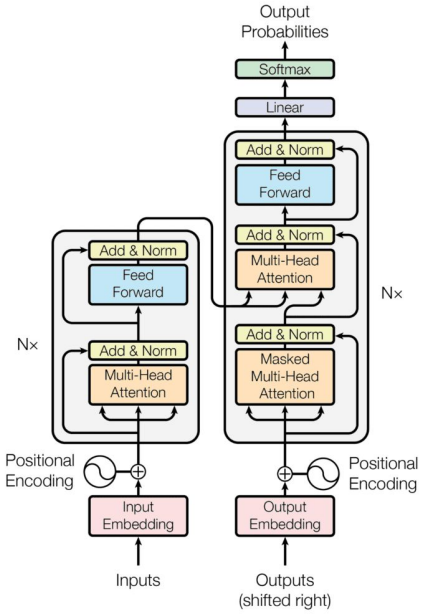
So far: Network architectures

Feed-forward

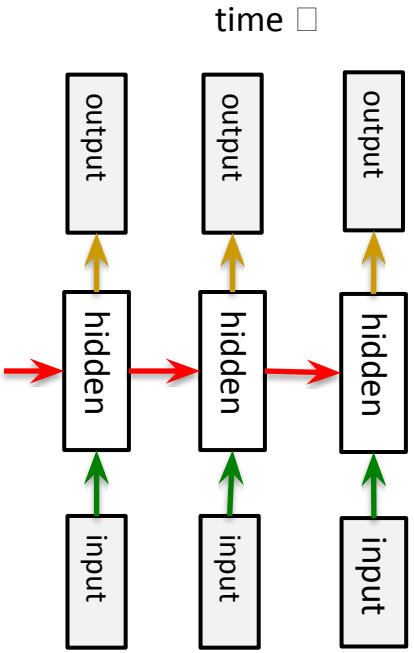
Fully connected



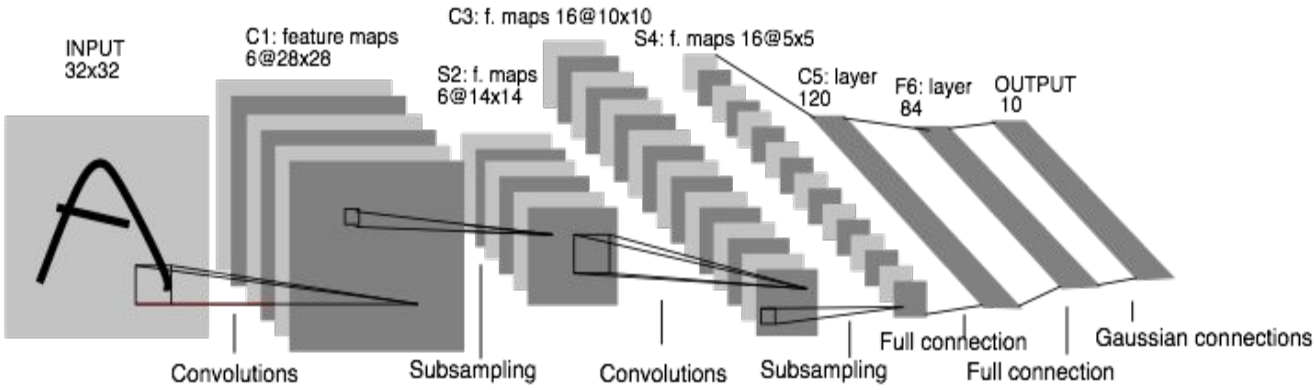
Transformer



Recurrent



Convolutional

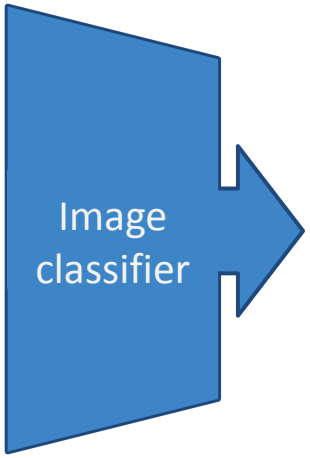


Why do we need model understanding?

❌	✅
	
Azure: food, cheese **Clarifai:** food, wood **Google:** food, dish	**Azure:** toilet, design **Clarifai:** people, faucet **Google:** product, liquid

- Model debugging
- Establish user trust

Why do we need model understanding?



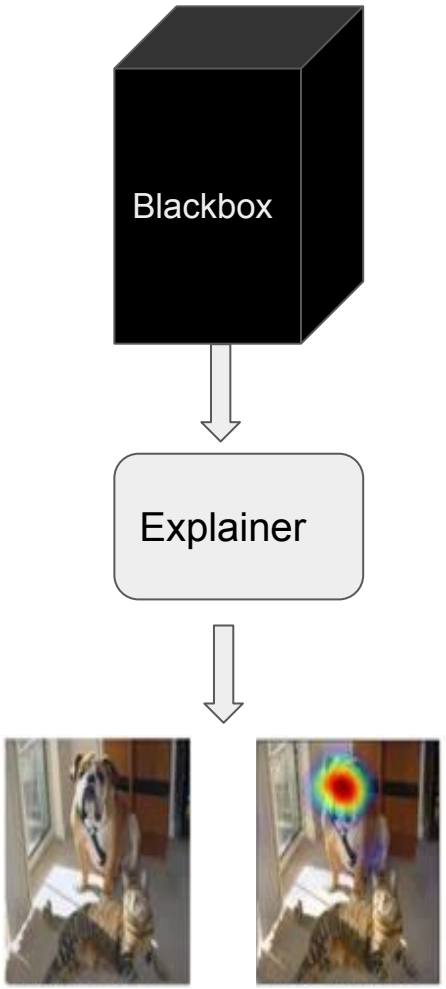
"lion": 94%

- *What makes this image look like a "lion" to the model?*
- *Does the lion cub contribute as much?*
- *What about the reflections?*

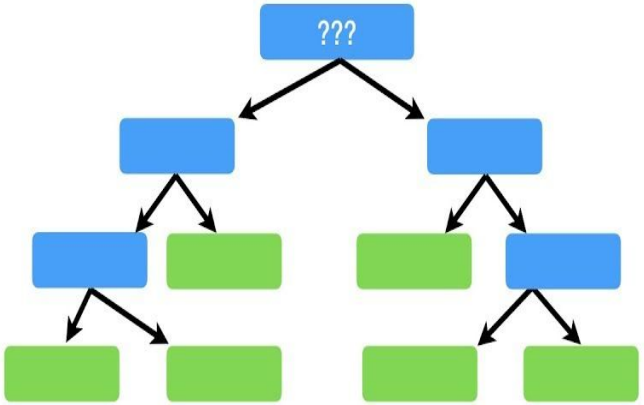
Build a deeper understanding of the model

How to achieve model understanding?

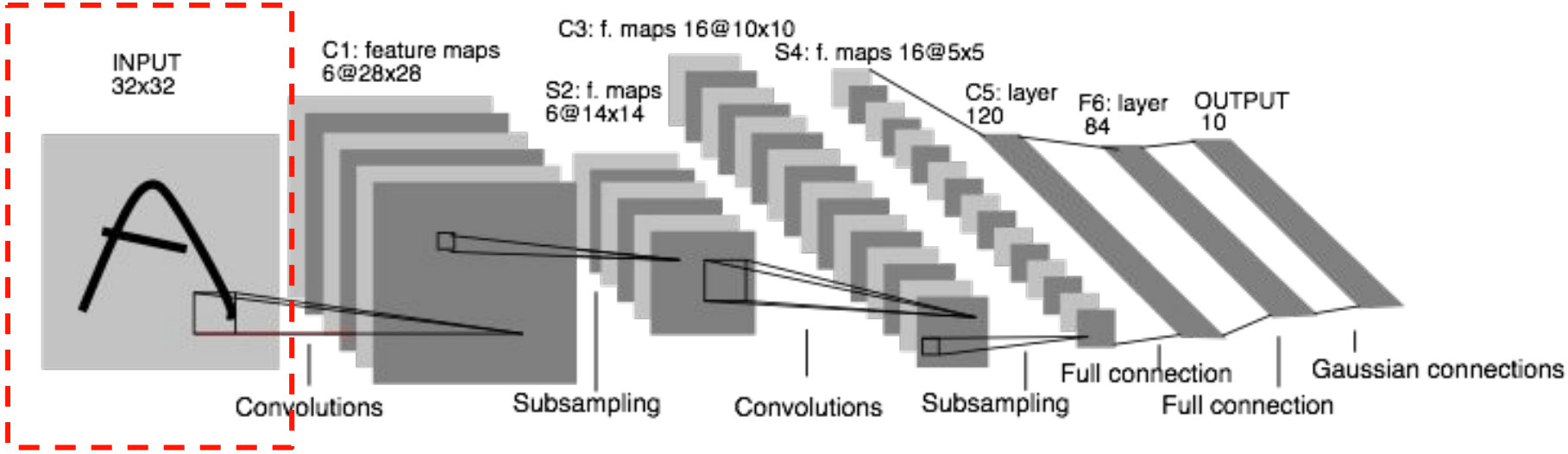
Posthoc explanations



Inherently interpretable models



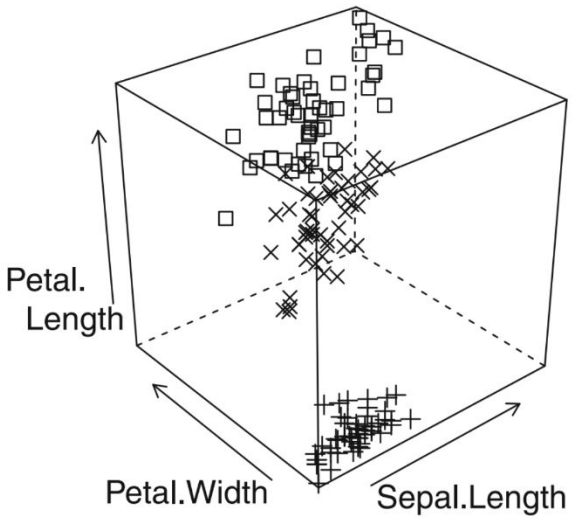
End to end pipeline

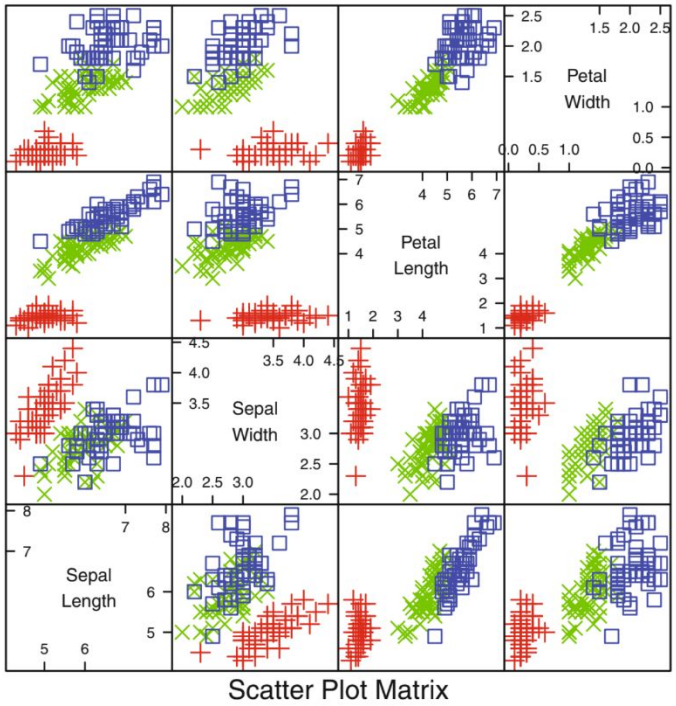


Recall: Visualizing features

Iris dataset-

- 4 features
  - Sepal length/width
  - Petal length/width
- 3 classes





Visualizing high dimensional data is hard!

Recall: Visualizing features

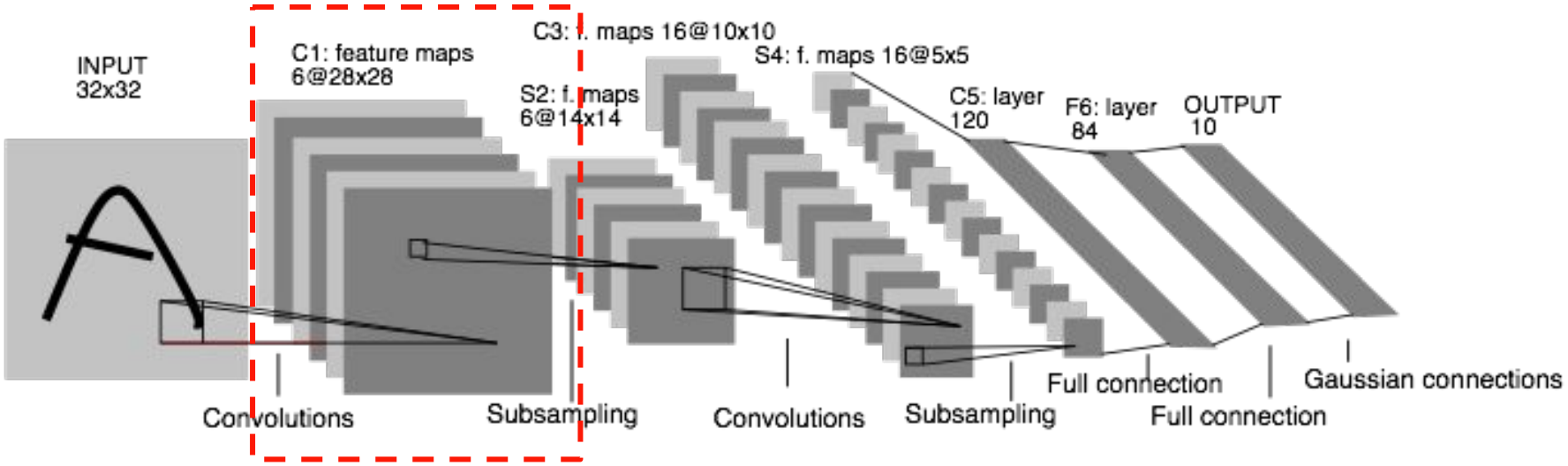
Goal: Projecting data to fewer dimensions

- Retain relative information about the feature higher dimensions
- Many project down to 2D

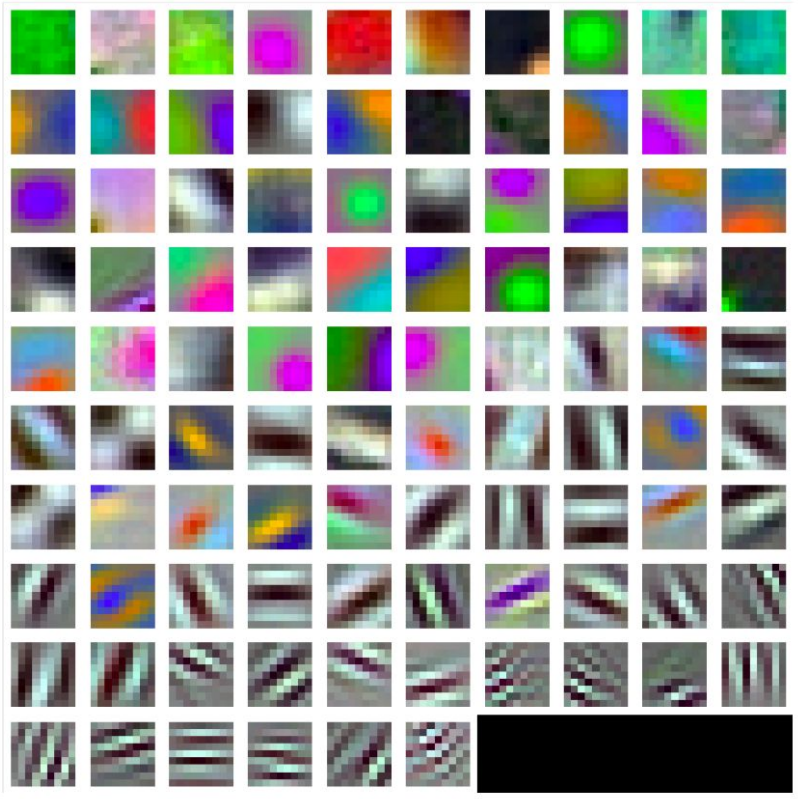
Potential Solutions:

- PCA
- Covert into a (sort-of) classification problem

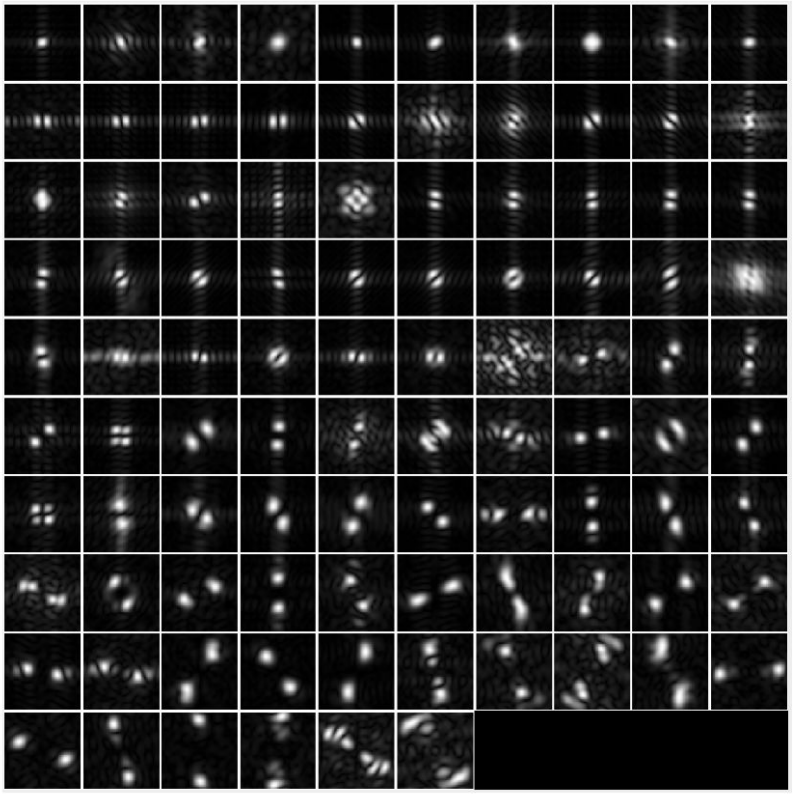
End to end pipeline

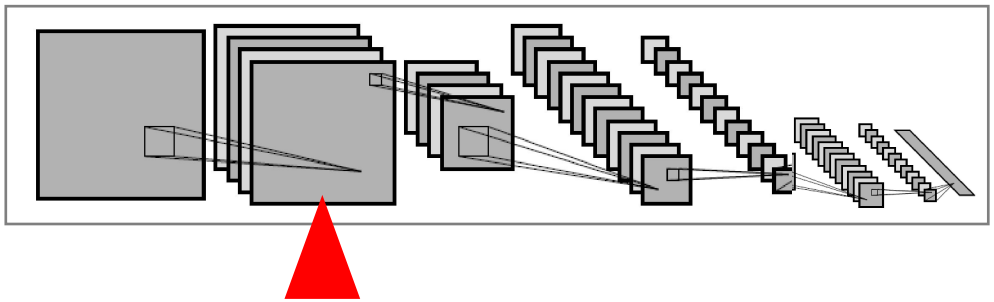


Convolutional filters



96 Units in conv1



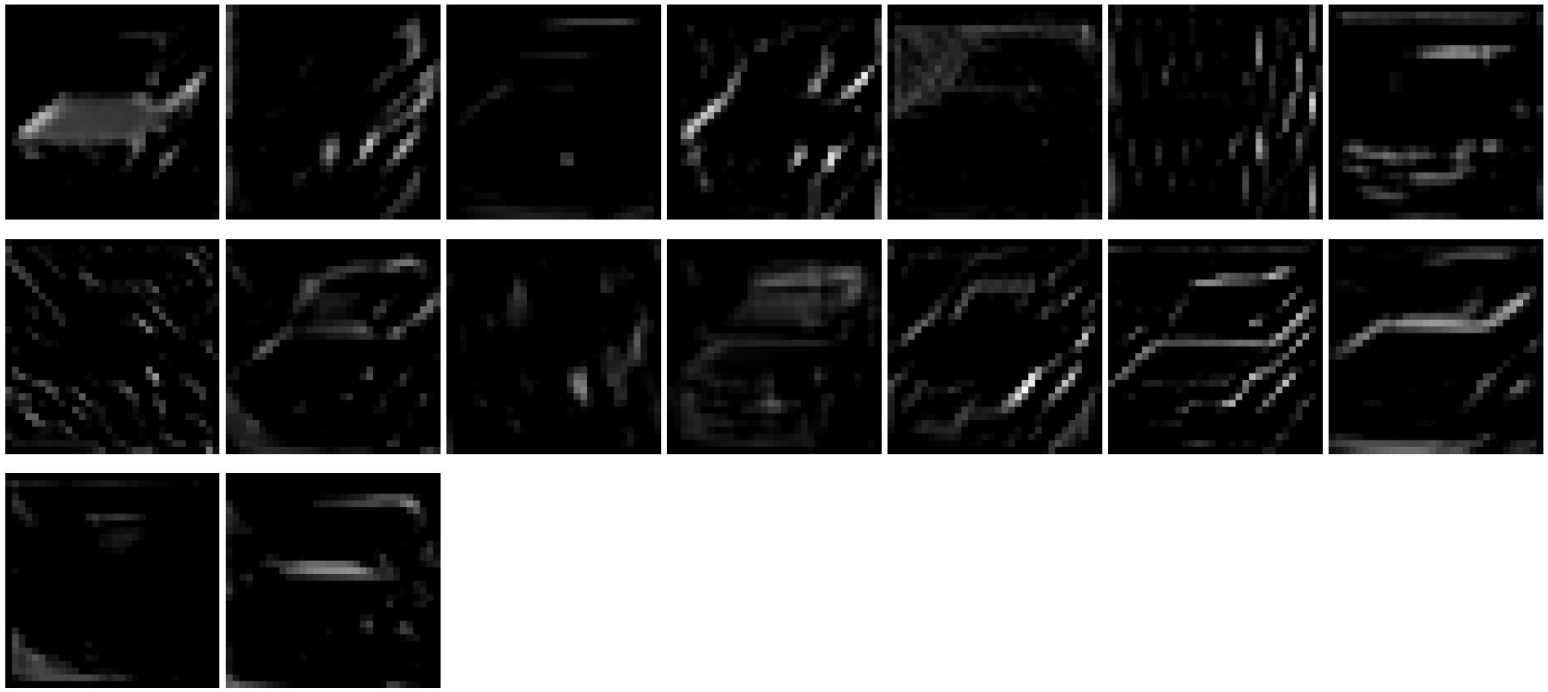


filter size 5x5x3, stride 1

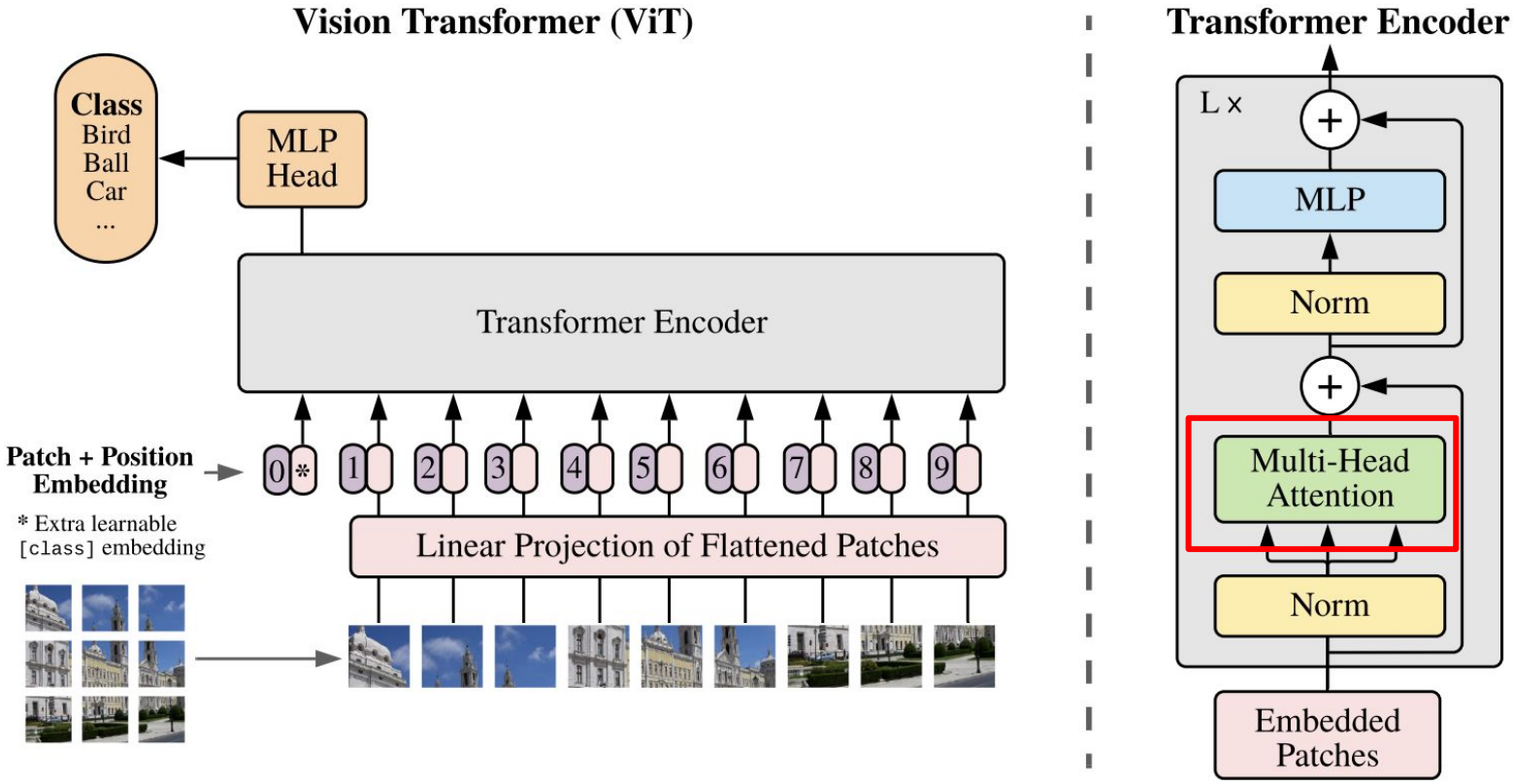
input (32x32x3)

Convolution

conv (32x32x16) params: 16x5x5x3+16 = 1216

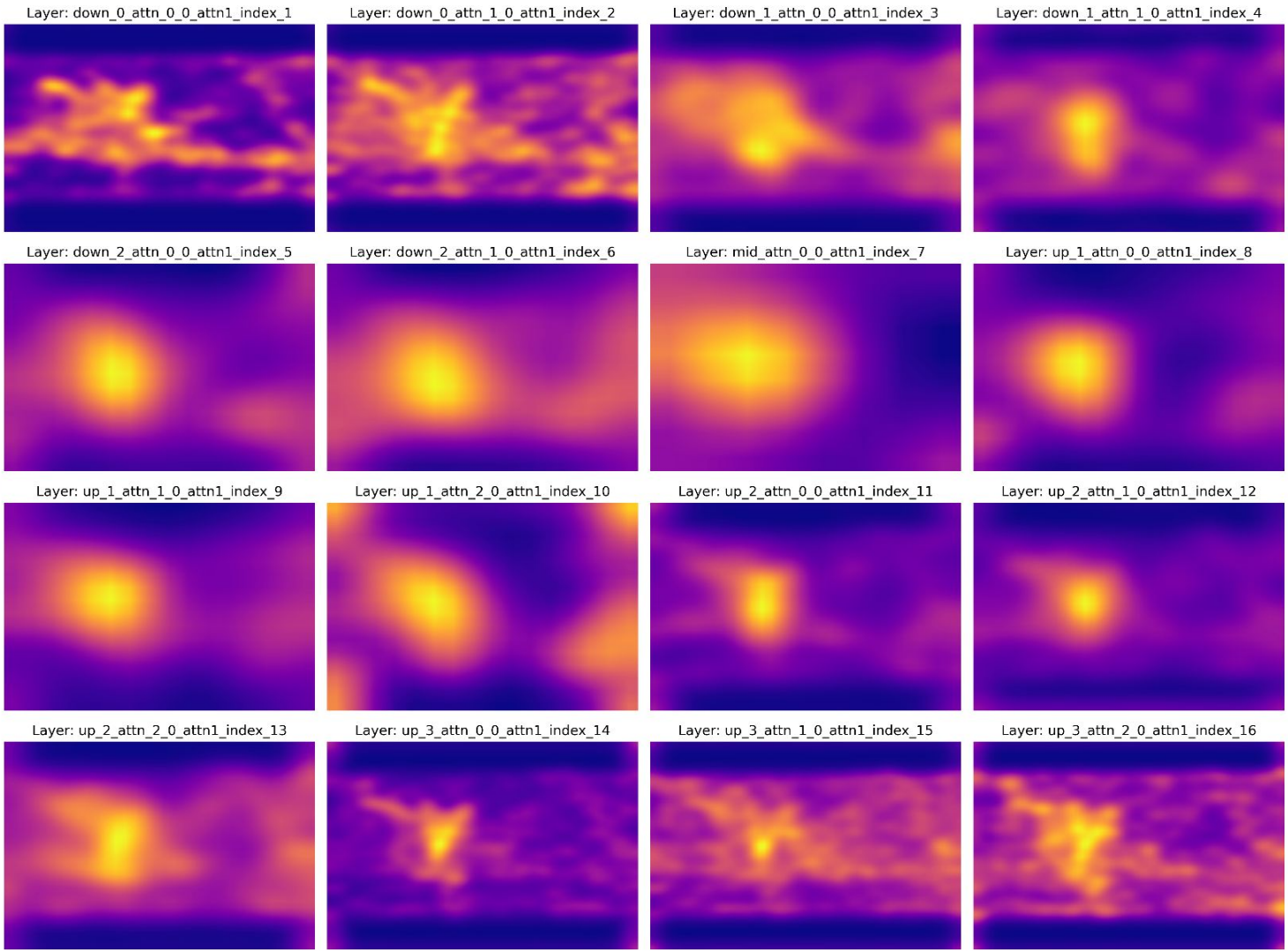


Recall: Attention in transformers



Self-Attention Map visualizations

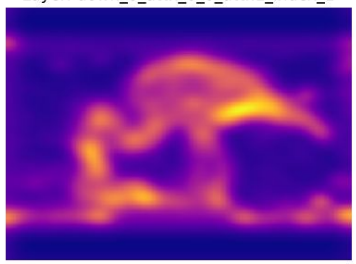
Original Image



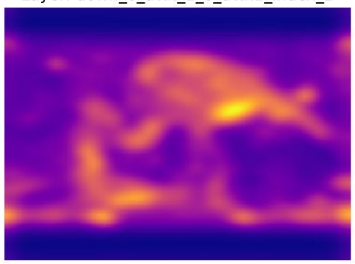
Self-Attention Map visualizations

Original Image

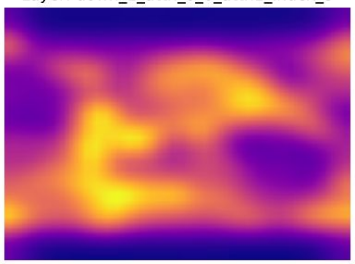
Layer: down_0_attn_0_0_attn1_index_1



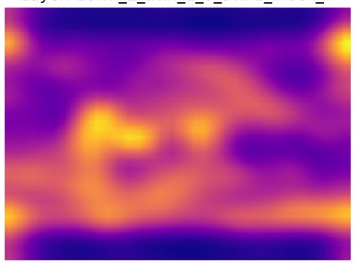
Layer: down_0_attn_1_0_attn1_index_2



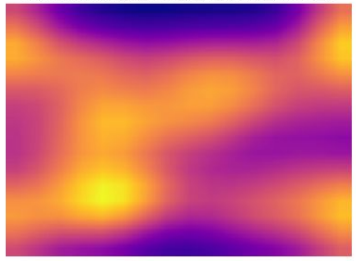
Layer: down_1_attn_0_0_attn1_index_3



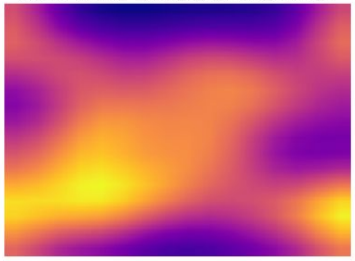
Layer: down_1_attn_1_0_attn1_index_4



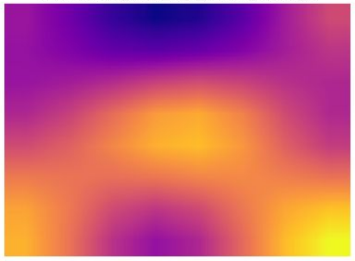
Layer: down_2_attn_0_0_attn1_index_5



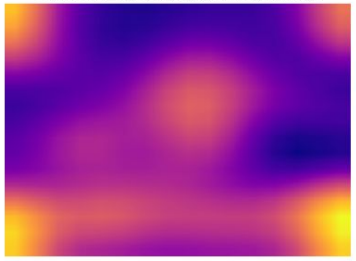
Layer: down_2_attn_1_0_attn1_index_6



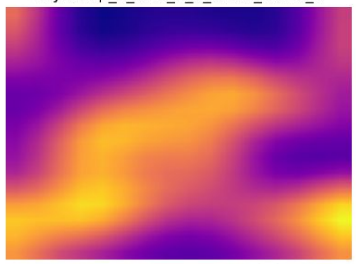
Layer: mid_attn_0_0_attn1_index_7



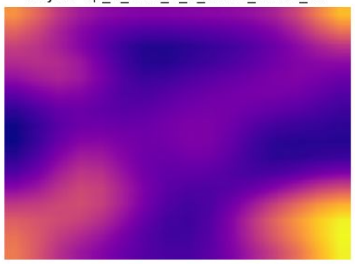
Layer: up_1_attn_0_0_attn1_index_8



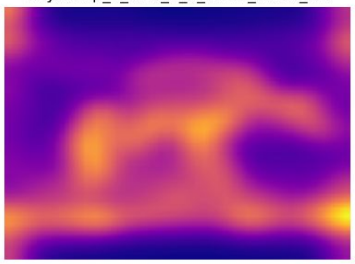
Layer: up_1_attn_1_0_attn1_index_9



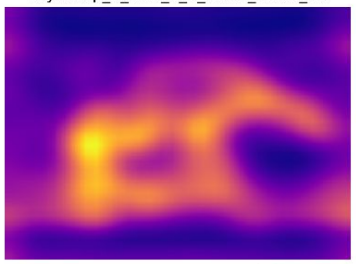
Layer: up_1_attn_2_0_attn1_index_10



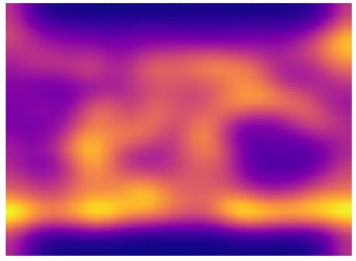
Layer: up_2_attn_0_0_attn1_index_11



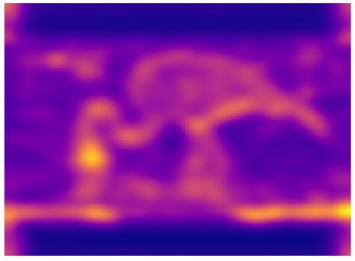
Layer: up_2_attn_1_0_attn1_index_12



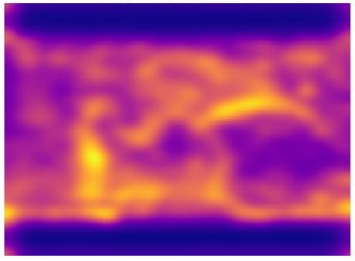
Layer: up_2_attn_2_0_attn1_index_13



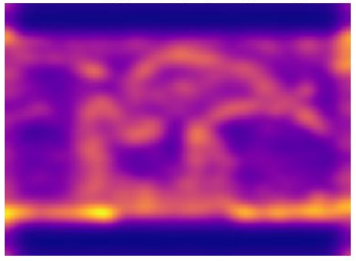
Layer: up_3_attn_0_0_attn1_index_14



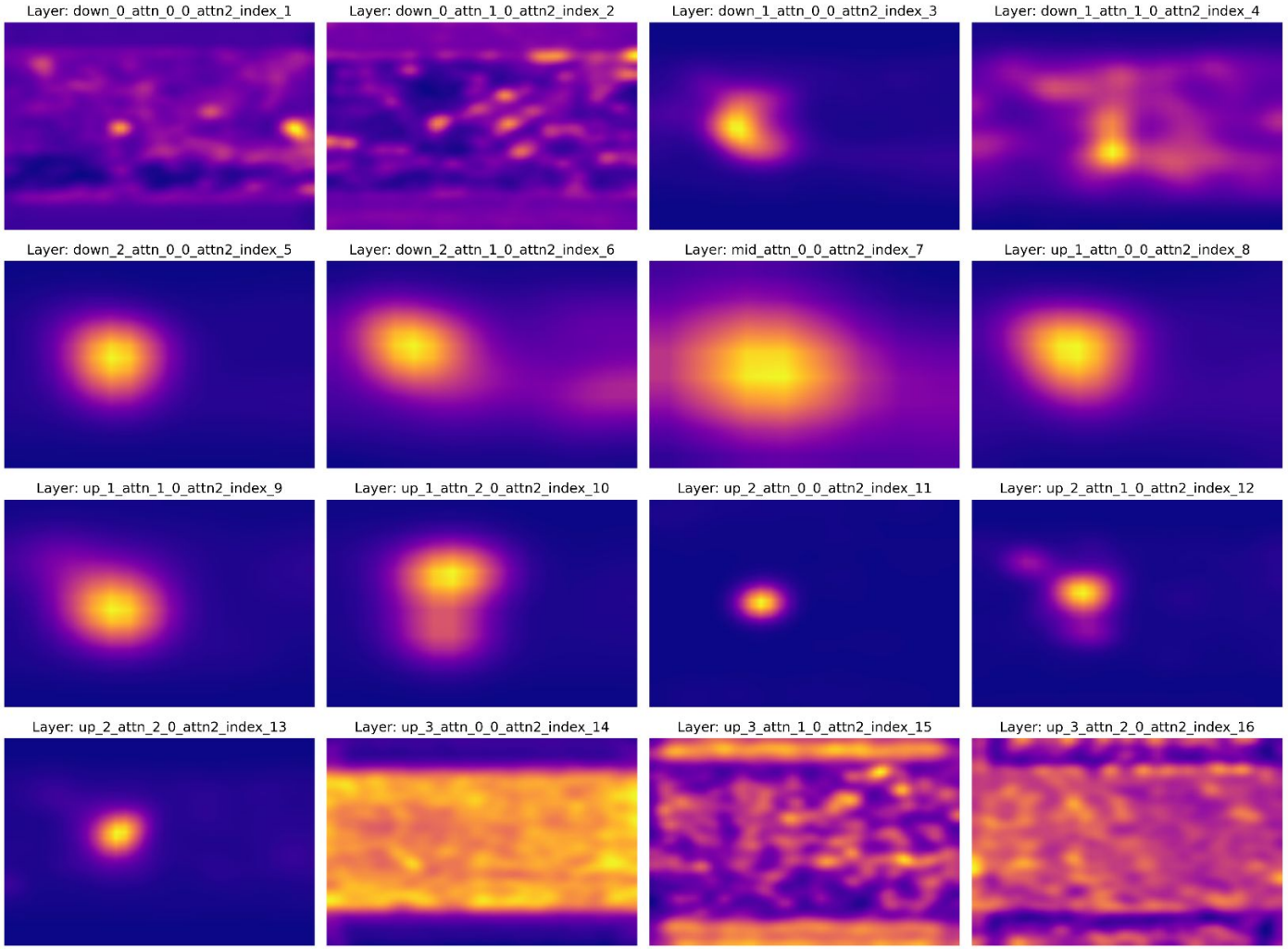
Layer: up_3_attn_1_0_attn1_index_15



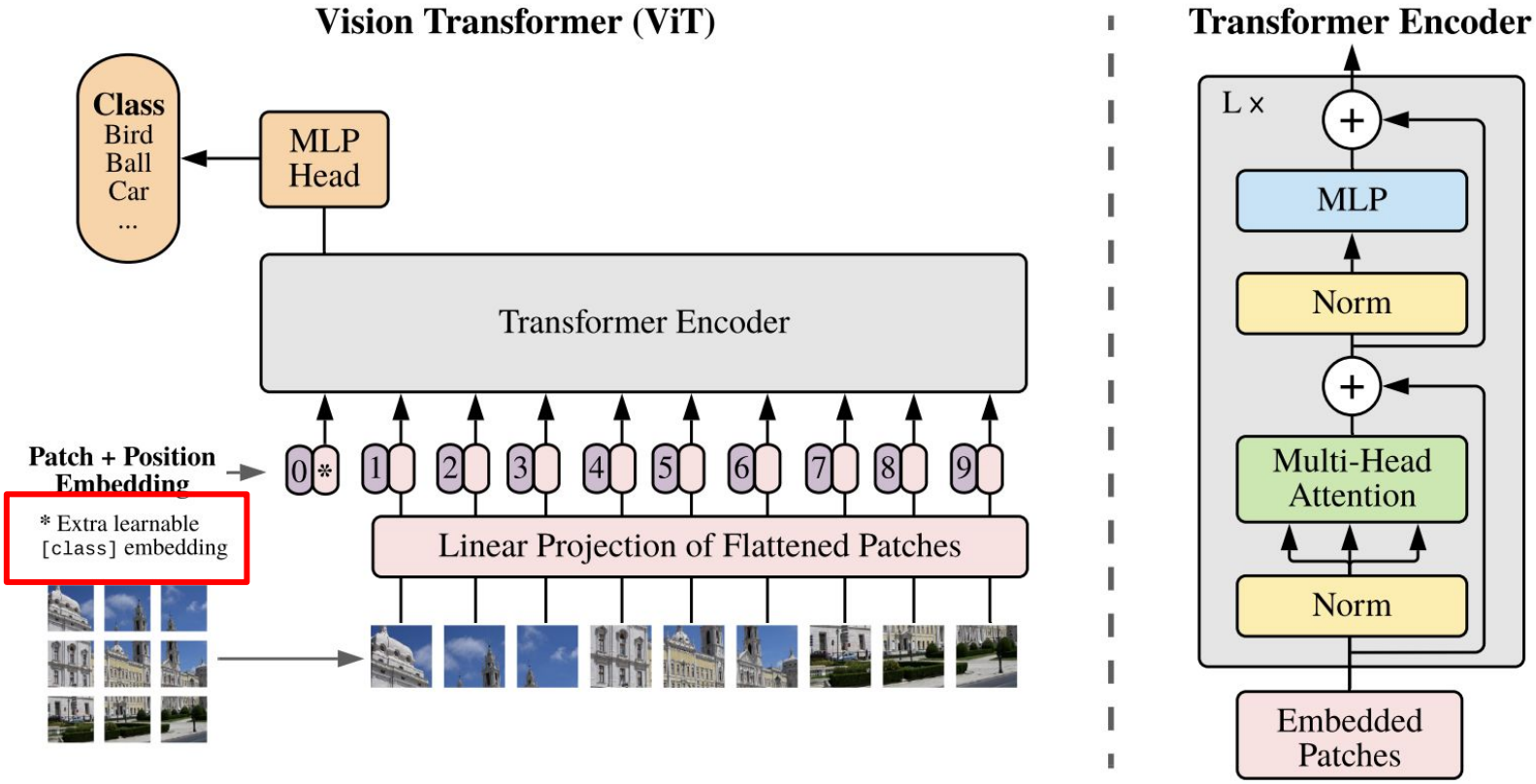
Layer: up_3_attn_2_0_attn1_index_16



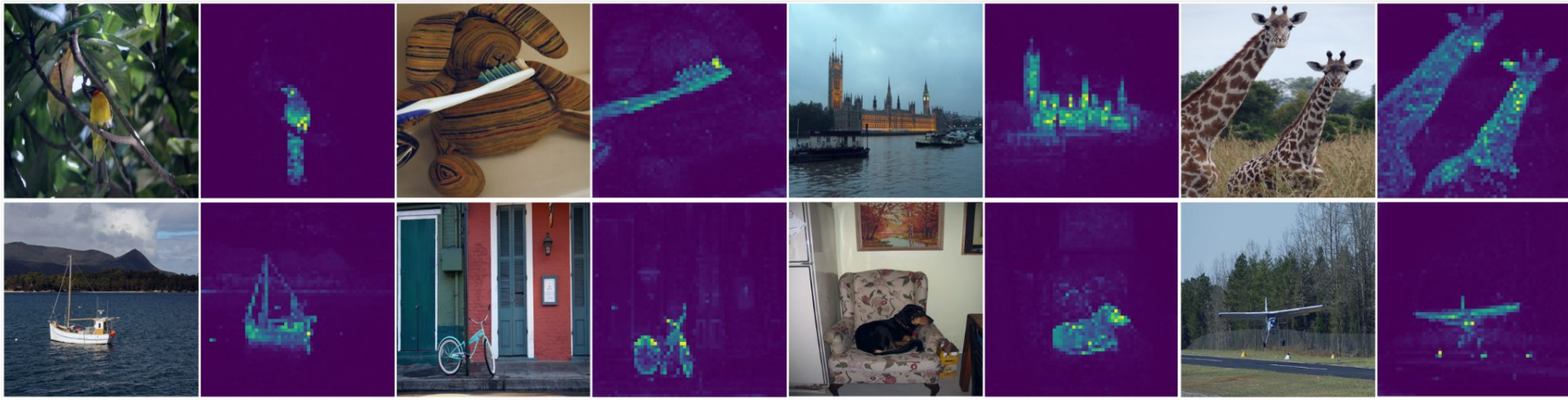
Cross attention visualization, when value = “pet”



Recall: “CLS” token in transformers

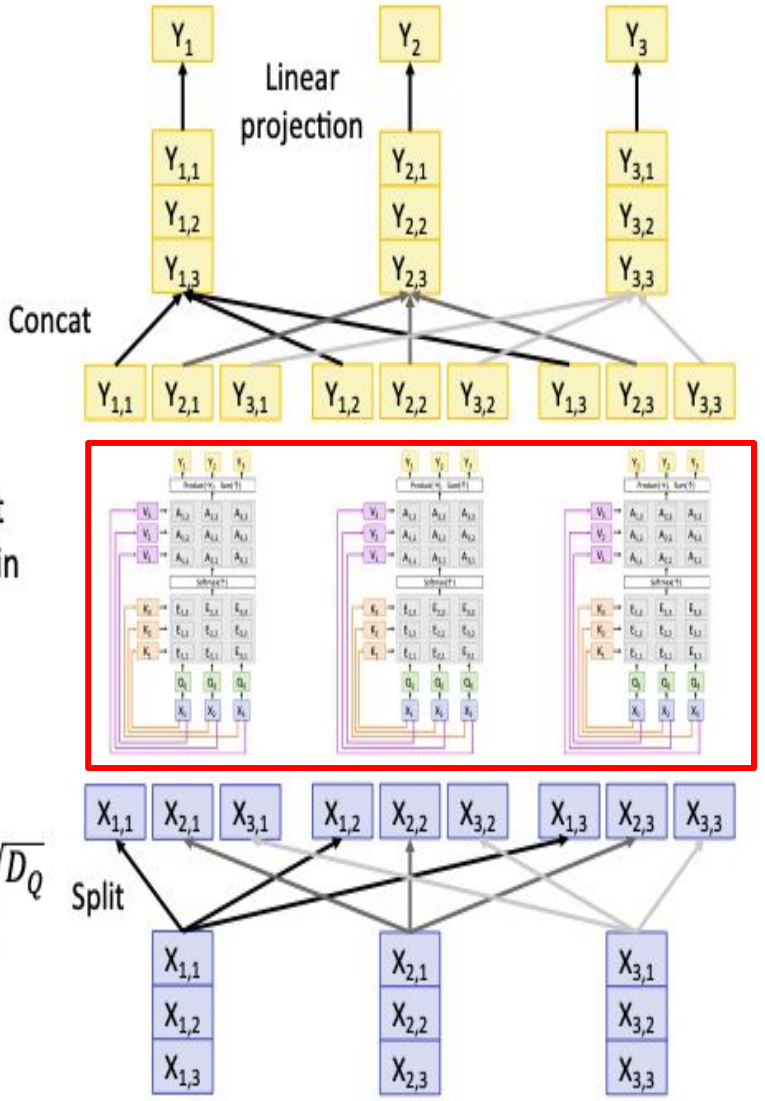


What is captured in CLS token?



Caron et.al., Emerging Properties in Self-Supervised Vision Transformers

Multihead Self-Attention



Inputs:

Input vectors: X (Shape: $N_X \times D_X$)

Key matrix: W_K (Shape: $D_X \times D_Q$)

Value matrix: W_V (Shape: $D_X \times D_V$)

Query matrix: W_Q (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

Computation:

Query vectors: $Q = XW_Q$

Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)

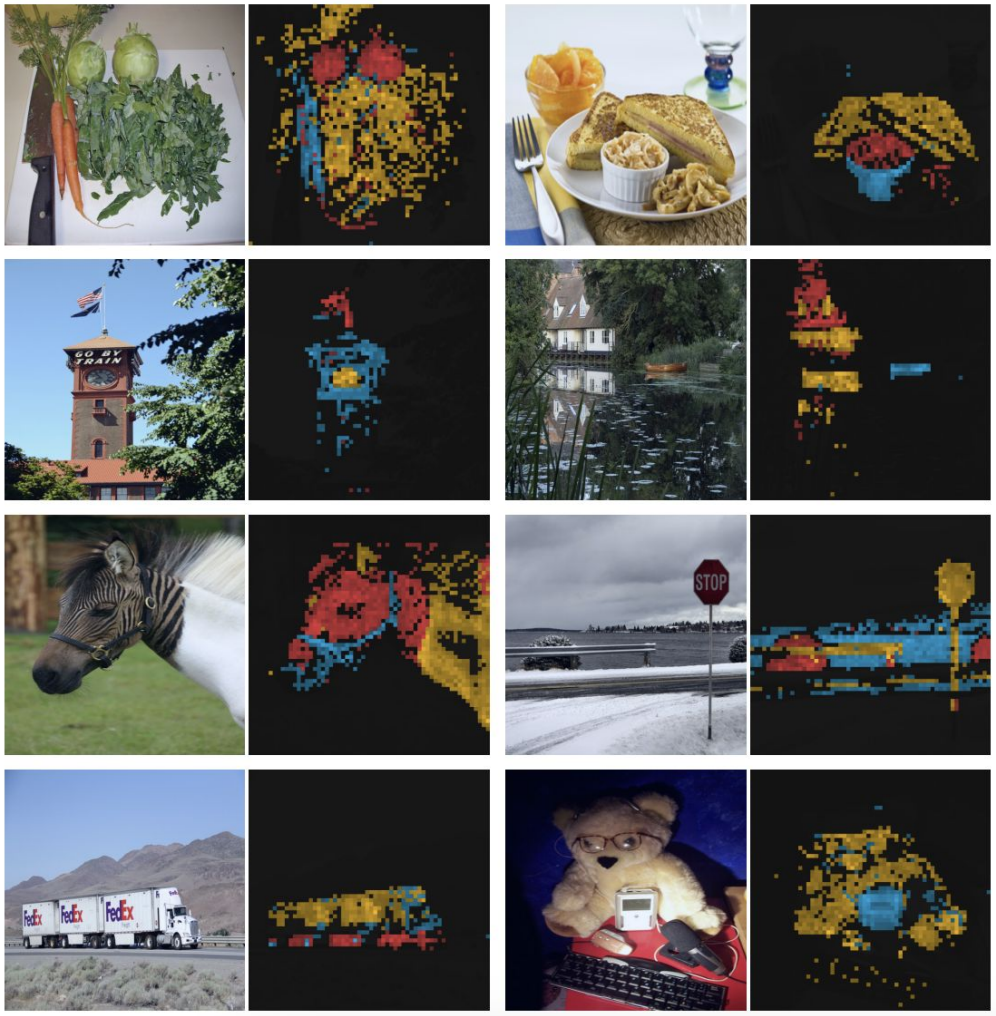
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)

Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)

Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

What is captured in CLS token in the intermediate attention layers?



*Each color corresponds to different attention head.