Ajou University

2024.10.30

Enhancing Vision Transformer Inference Speed

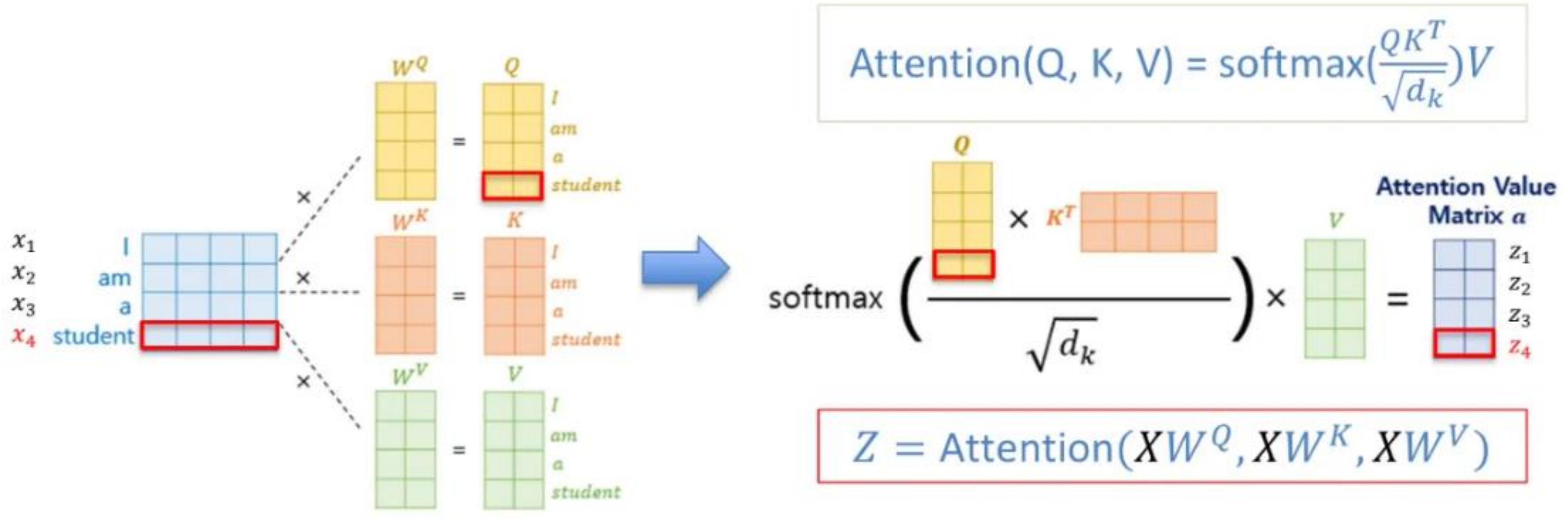
Integrating Hybrid Architectures and Novel Similarity Measures

남현원, 정용훈, 한동엽, 강전찬

Table of contents

What is ViT

Transformer model : Advanced feedforward network based on attention

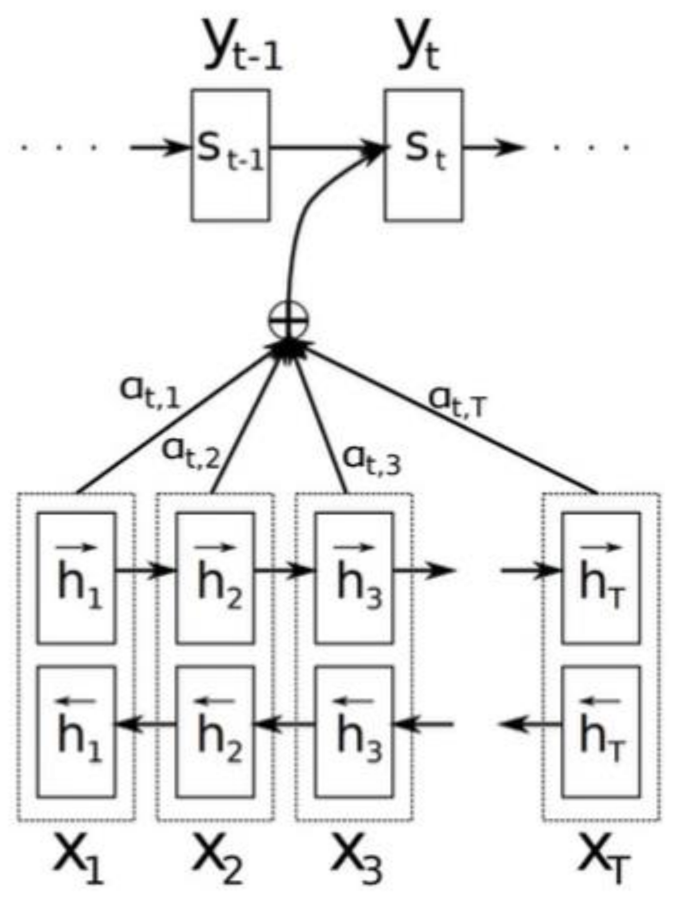

W^Q
Q
W^K
K
W^V
V
x_1 I
x_2 am
x_3 a
x_4 student
I
am
a
student
$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$
softmax
K^T
$\sqrt{d_k}$
Attention Value Matrix a
z_1
z_2
z_3
z_4
$Z = \text{Attention}(XW^Q, XW^K, XW^V)$

Focuses on Important information.

- By assigning higher weight to crucial part

3 step of Attention

1. Similarity Measurement ($Q * K^T$)
2. Weight Calculation (Softmax)
3. Weighted Sum Calculation (* V)



Decoder

Encoder

We can use result of Attention (Weighted sum) to input of decoder with previous output.

We can think of it as a method that tells you what's important in a probabilistic way.

What is ViT

Transformer model Architecture

Multi head Attention

- Self-attention occurs multiple times in parallel across the entire input
- model learns various relationships between words

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Encoder self - attention

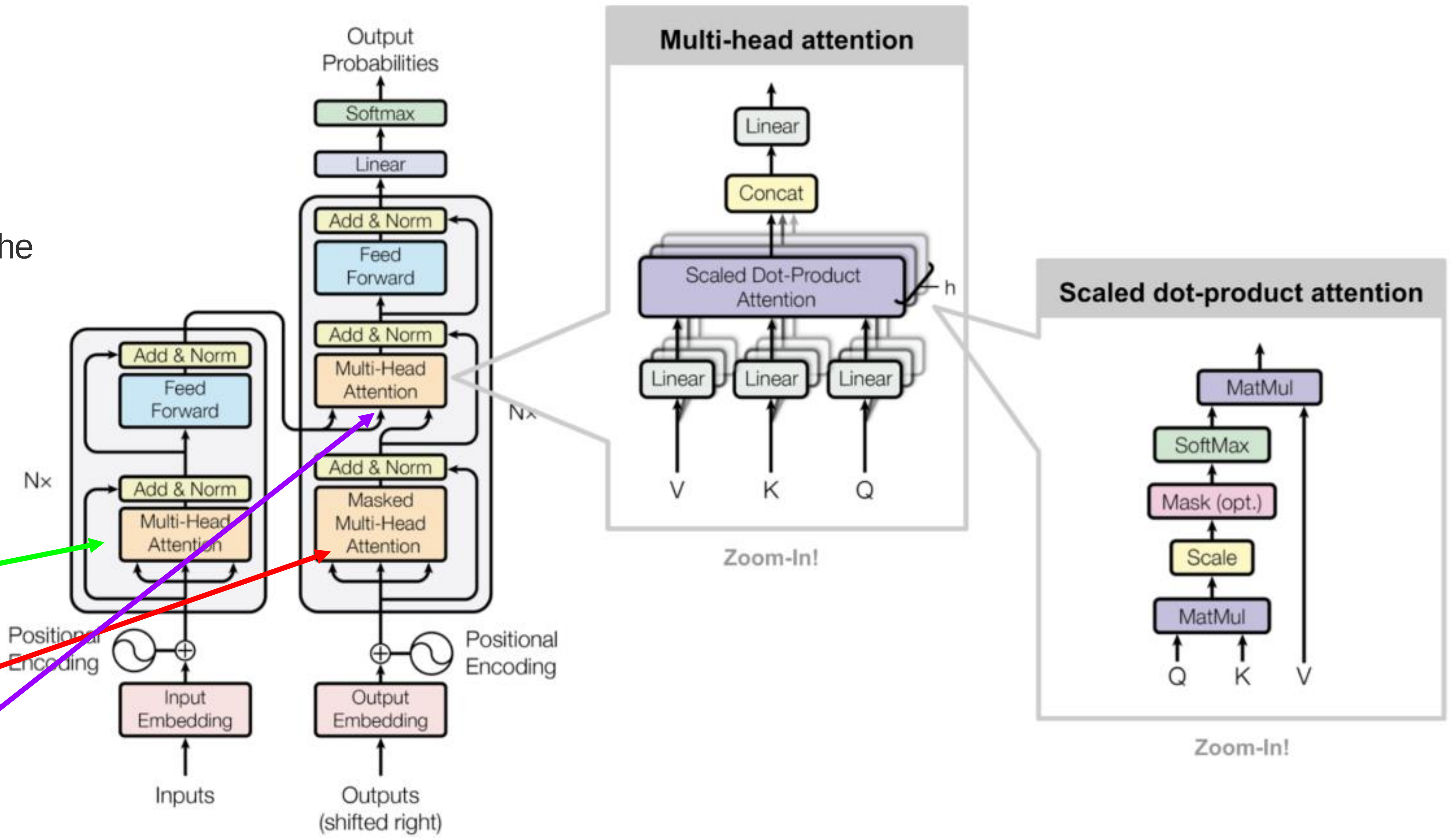
- Token look at each other

Decoder self - attention (masked)

- Tokens look at the previous state

Decoder - encoder attention

- Target token looks at the source
- Key, Value (Encoder), Query (Decoder)



Simple Transformer Model with Multi head Attention and Scaled Dot Product Attention

Why Transformer model?

1. Long large dependencies (relationship between distant tokens)
2. Self-attention is lower computational cost than RNN
3. Sequntial operation is small (RNN Receives input sequentially)

What is ViT

Image Processing Model based on Transformers

Apply the standard transformer used in NLP directly to images.

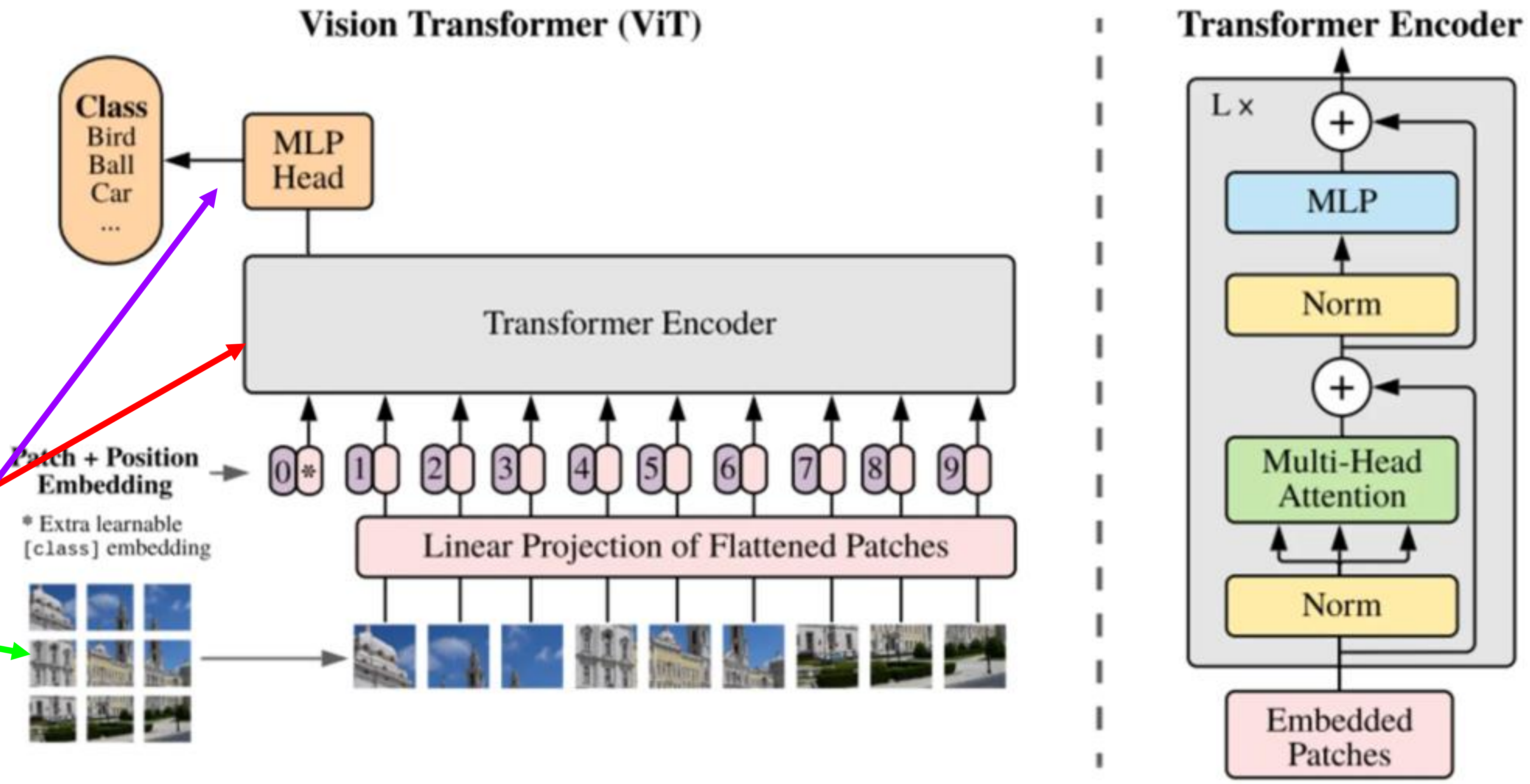
Input image → Divide the image into patches.

→ each patch linear embedding → input in order

Split the image into patches and input them as a sequence.
Same way words are input in NLP

Application of self-Attention mechanism consisting of multi-head

Multi Layer Perceptron to Classification Image



ViT architecture

Why Vit?

1. Global Information Gathering through Self-Attention
2. Train High-resolution images (large scale) better

But …

Problem of ViT

Time Complexity

We first looked at the time complexity of the transformer.

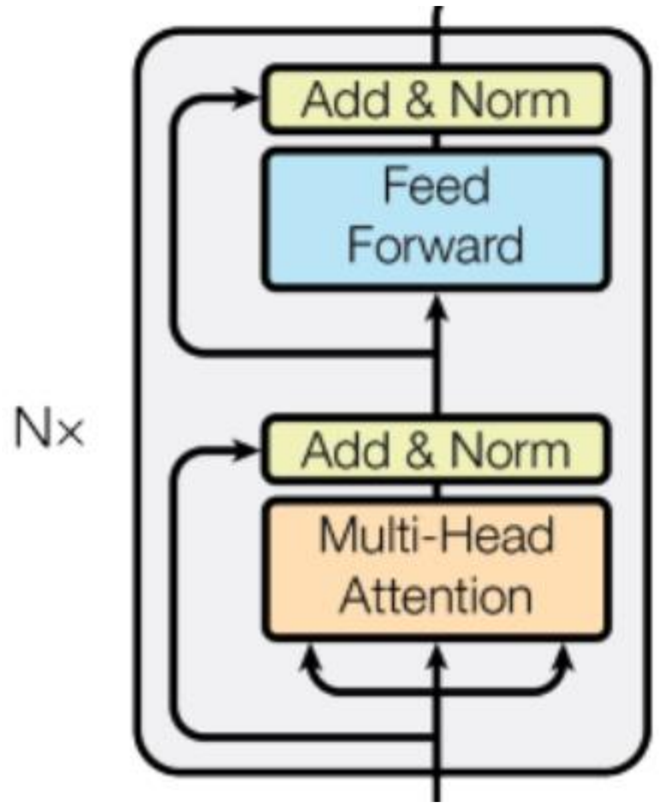
FFN and MHA are the main operations for transformer blocks.

Self-attention is calculated $O(n^2 d)$ as as shown in the following 'Attention is All You Need'

and for each input patch, it takes $O(nd^2)$ to generate Query, Key, and Value.

So, So the total MHA is $O(h \cdot (n\, d^2 + n^2 d))$

But, FFN only takes $O(n \cdot d \cdot d_{ff})$ for input dimension d and output dimension d_{ff}.

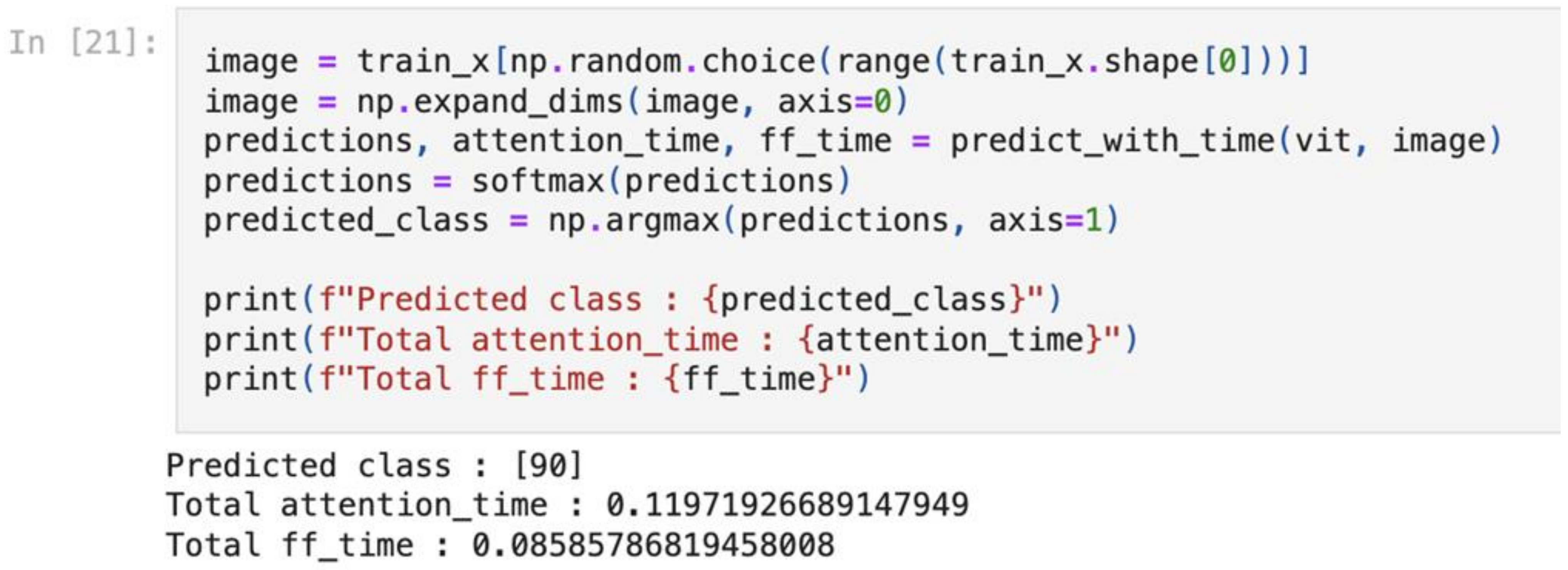


Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Attention is All You Need

Problem of ViT

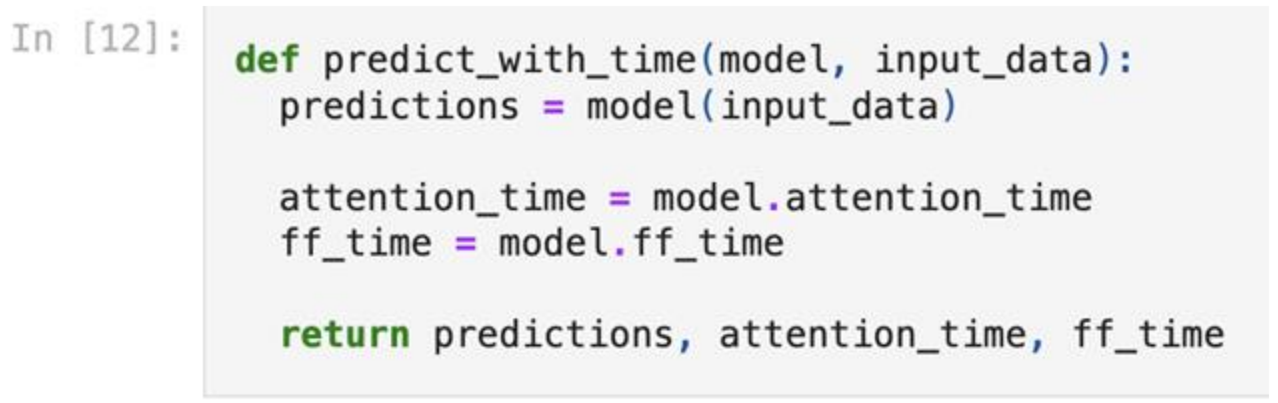
Time Complexity Check Code



```
In [21]: image = train_x[np.random.choice(range(train_x.shape[0]))]
         image = np.expand_dims(image, axis=0)
         predictions, attention_time, ff_time = predict_with_time(vit, image)
         predictions = softmax(predictions)
         predicted_class = np.argmax(predictions, axis=1)

         print(f"Predicted class : {predicted_class}")
         print(f"Total attention_time : {attention_time}")
         print(f"Total ff_time : {ff_time}")

         Predicted class : [90]
         Total attention_time : 0.11971926689147949
         Total ff_time : 0.08585786819458008
```



```
In [12]: def predict_with_time(model, input_data):
           predictions = model(input_data)

           attention_time = model.attention_time
           ff_time = model.ff_time

           return predictions, attention_time, ff_time
```

In fact, even when the time of FFN and Attention was measured, it was confirmed that Attention took longer.

In the Transformer structure, the attention was judged to be bottleneck, and other studies that tried to improve it were decided to be examined.

Our Github Repo : https://github.com/hun9008/AI_TeamProject_24FW-

Our Goal

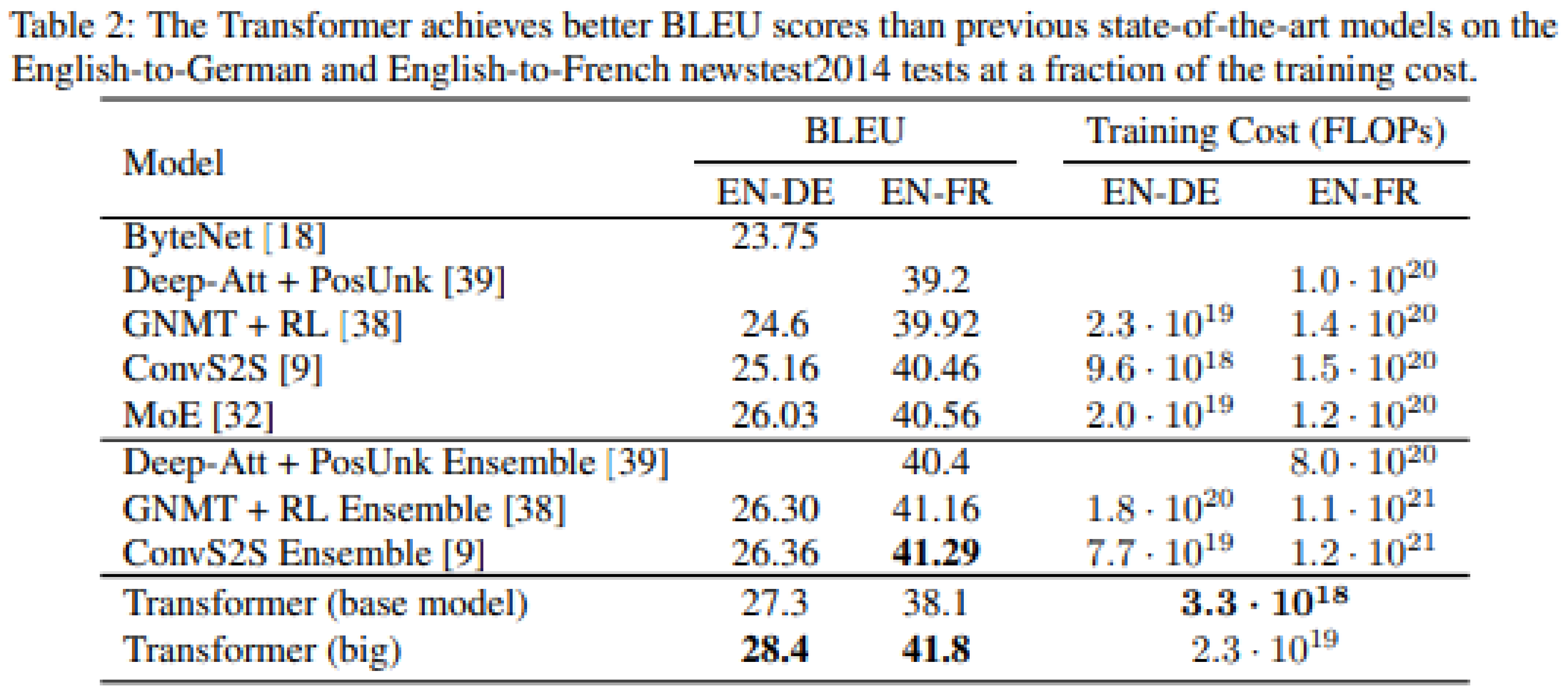


Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Looking at this table, we can see that the Transformer shows good performance compared to other models.

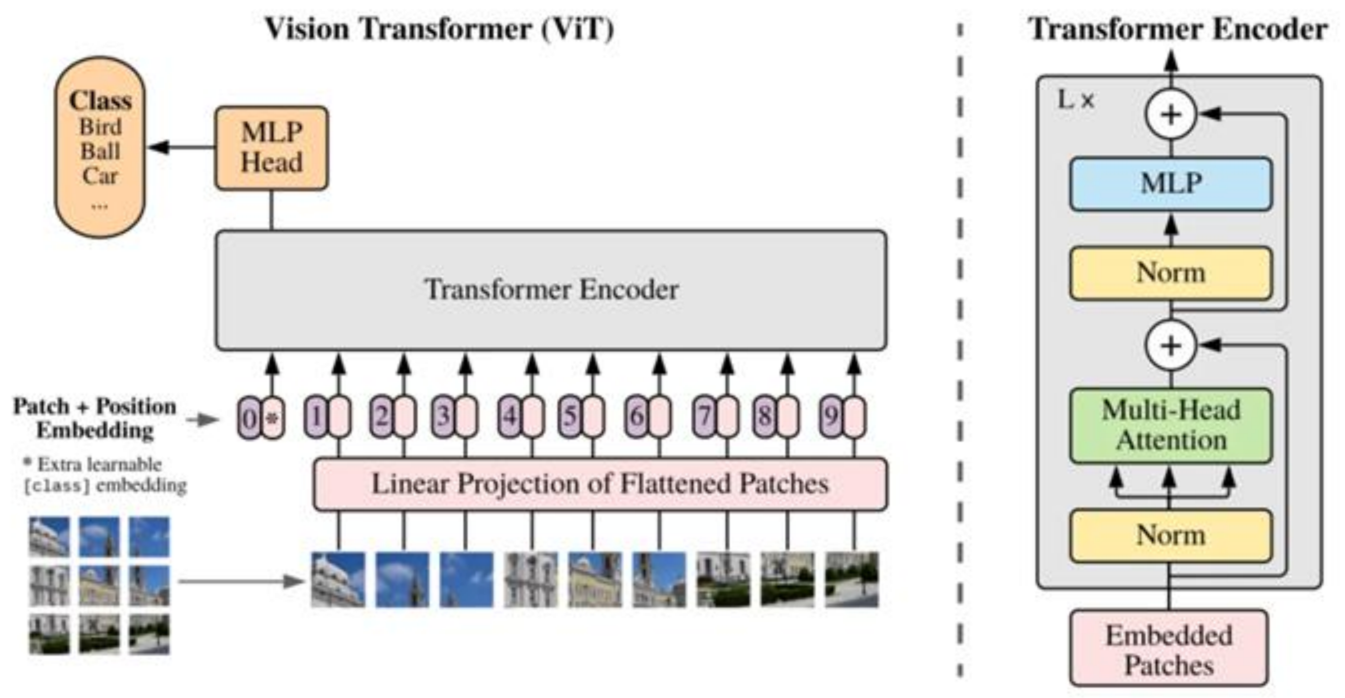
But it still requires substantial computational power when the number of patches increases, which limits its practical use in industry.

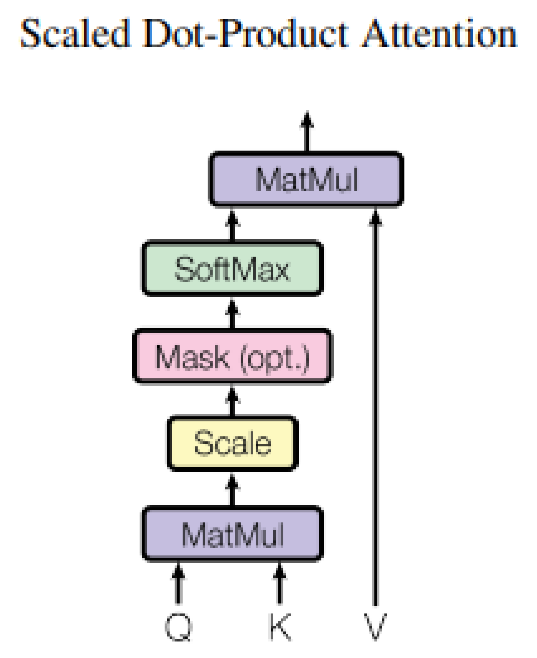
We identified this issue and initiated this project to improve computational efficiency while maintaining the same level of performance.

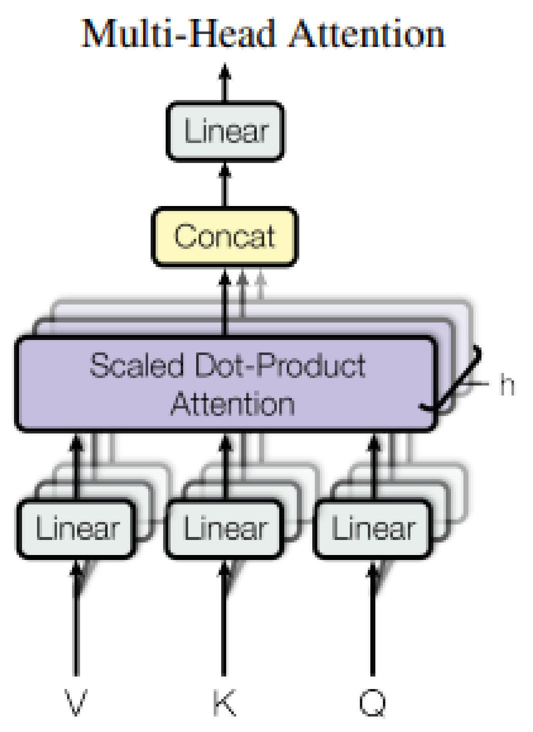
Our Goal

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Attention is All You Need







As we saw previous slides, when calculating self attention, It still has a **high time complexity**. Through this project, we aim to explore methods to **reduce this time complexity**.

Related Works

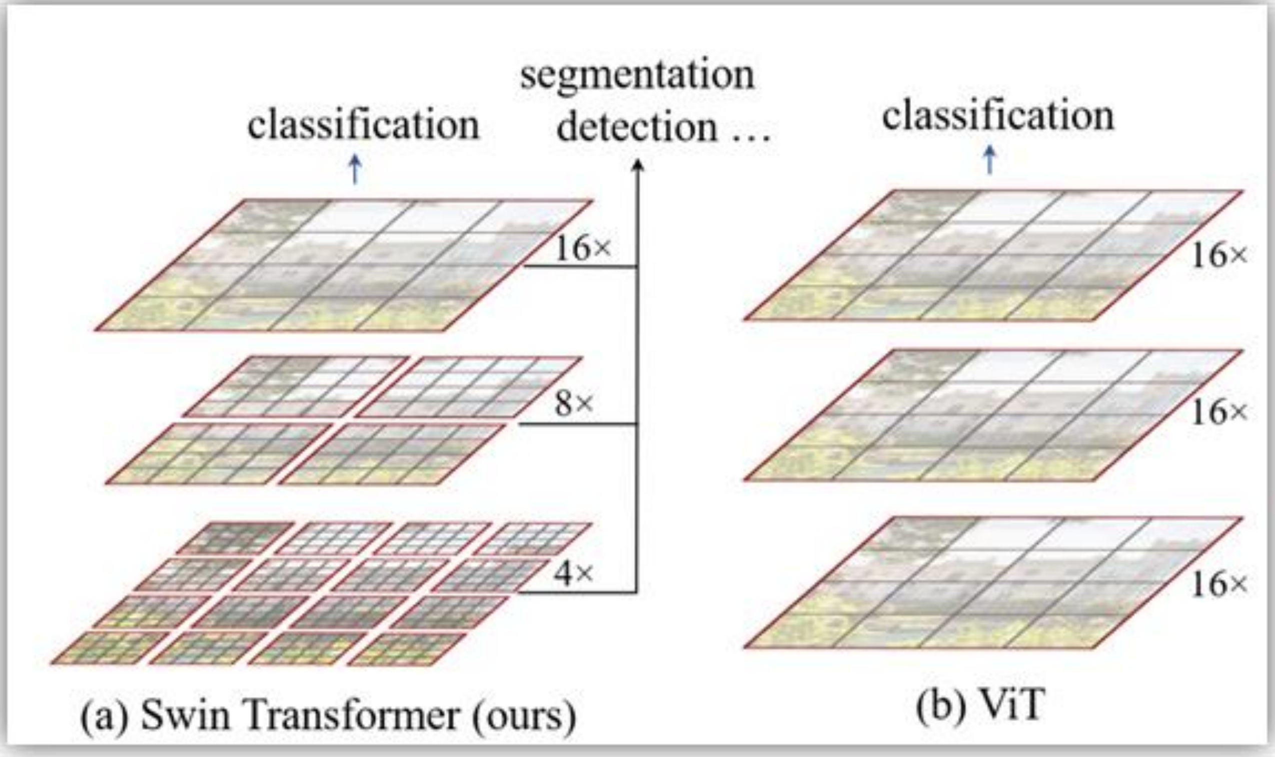
Swin Transformer: Hierarchical Vision Transformer using Shifted Windows (ICCV 2021)

Vision tasks with attention are affected on two factors:

1. Unlike NLP tasks, vision task is more relevant to **scale**
   - **fixed-size tokens are inappropriate** for vision tasks

1. Attention computation is directly affected by image resolution
   - image embeddings are more **dense** than language tokens

-> Hierarchical feature maps can attend to various scale of inputs.

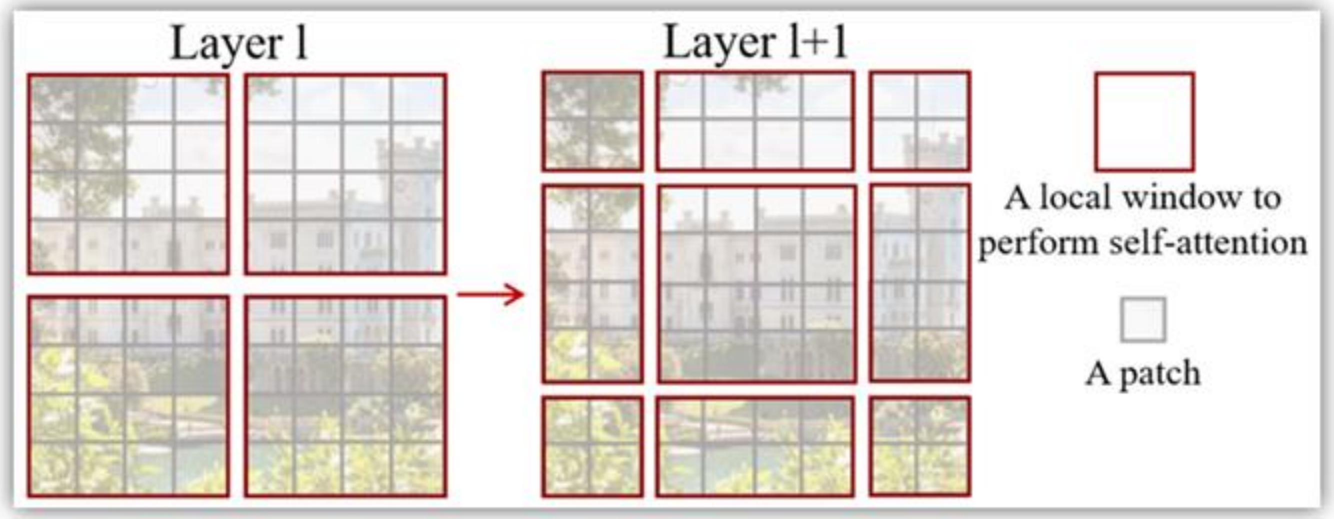


$$\Omega(\mathrm{MSA}) = 4hwC^2 + 2\boxed{(hw)^2}C$$

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2\boxed{M^2hw}C$$

Complexity becomes linear!

h and w is the feature map dimension.
M is the window size of Swin Transformer.



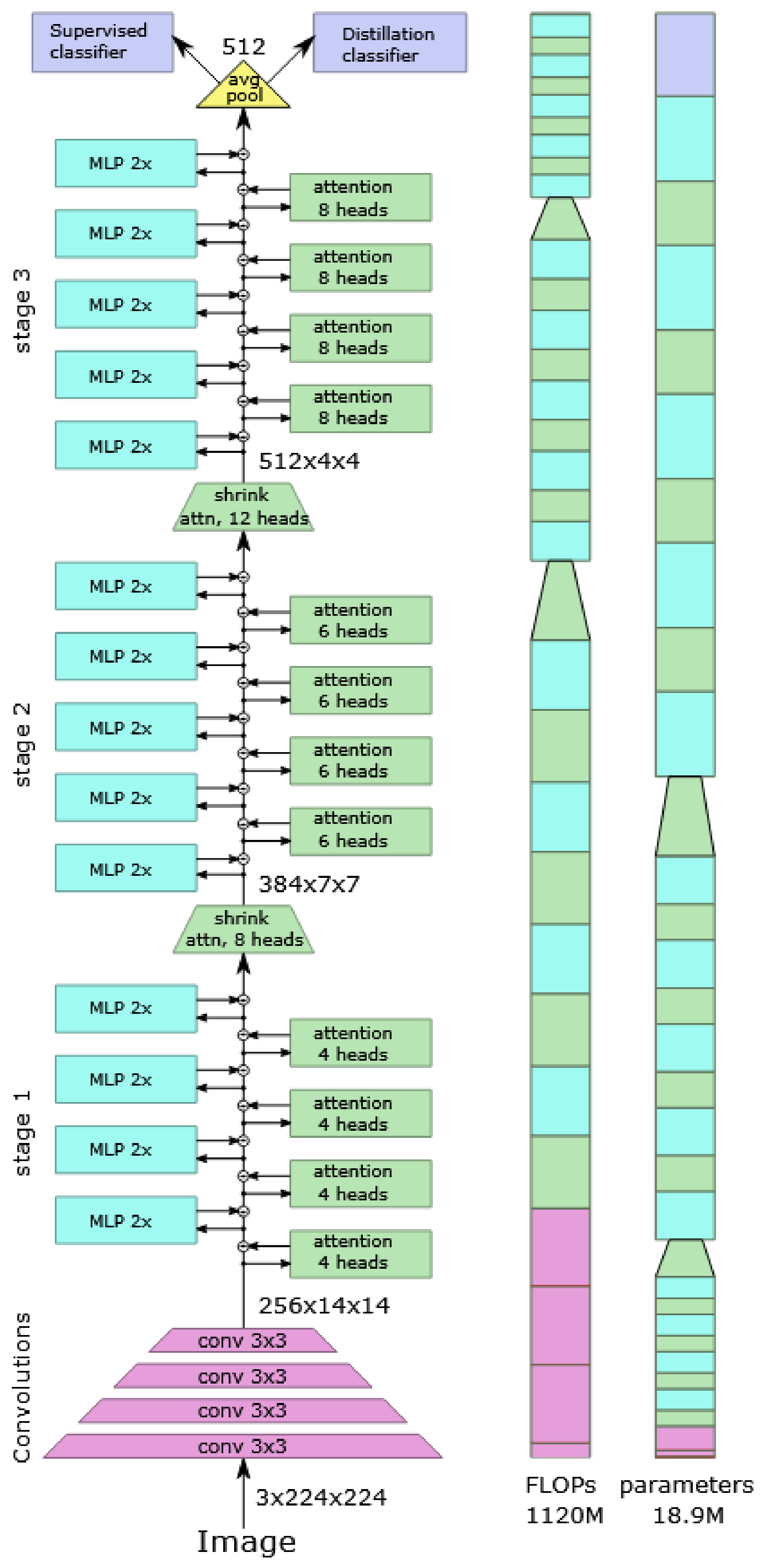
Related Works

LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference (ICCV 2021)

A hybrid approach which uses both convolutional and attention architectures, aimed to **optimize computation cost.**

This model has outperformed other approaches in trade-off between accuracy and inference time.

Convnets are highly dependent on complex data accesses and IOs, but transformers are computed with large-scale matrix multiplications, which can be used to optimize speed/accuracy tradeoff.



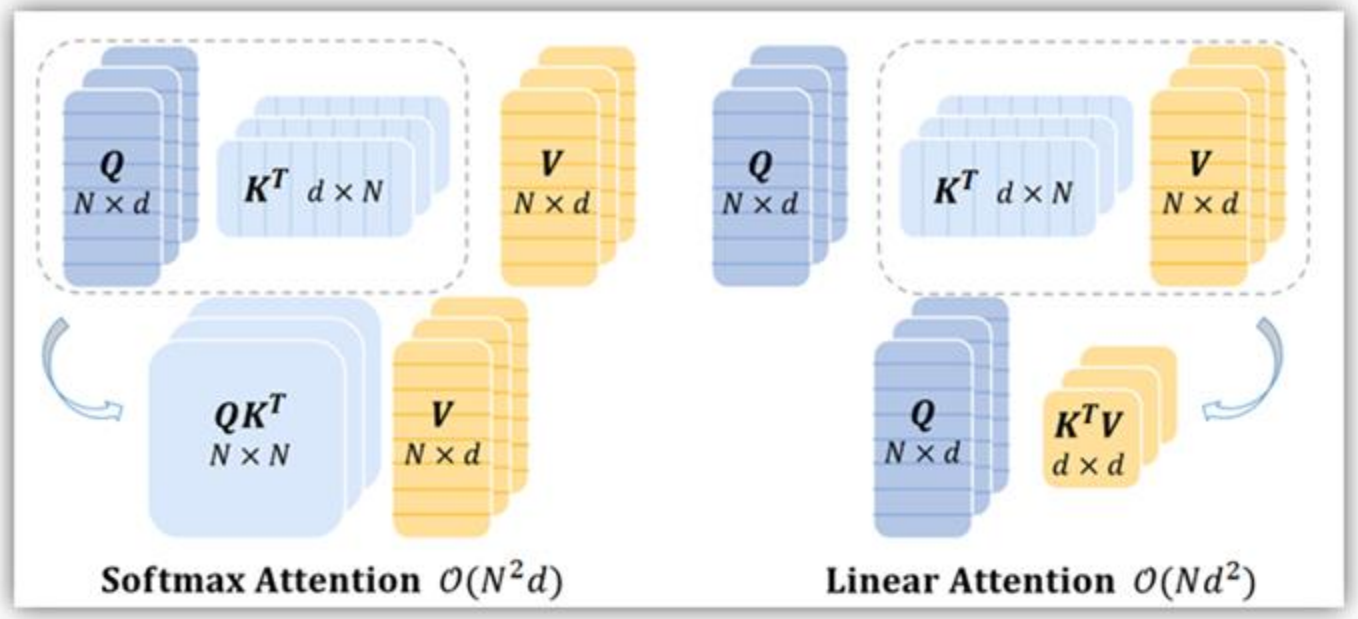
Related Works

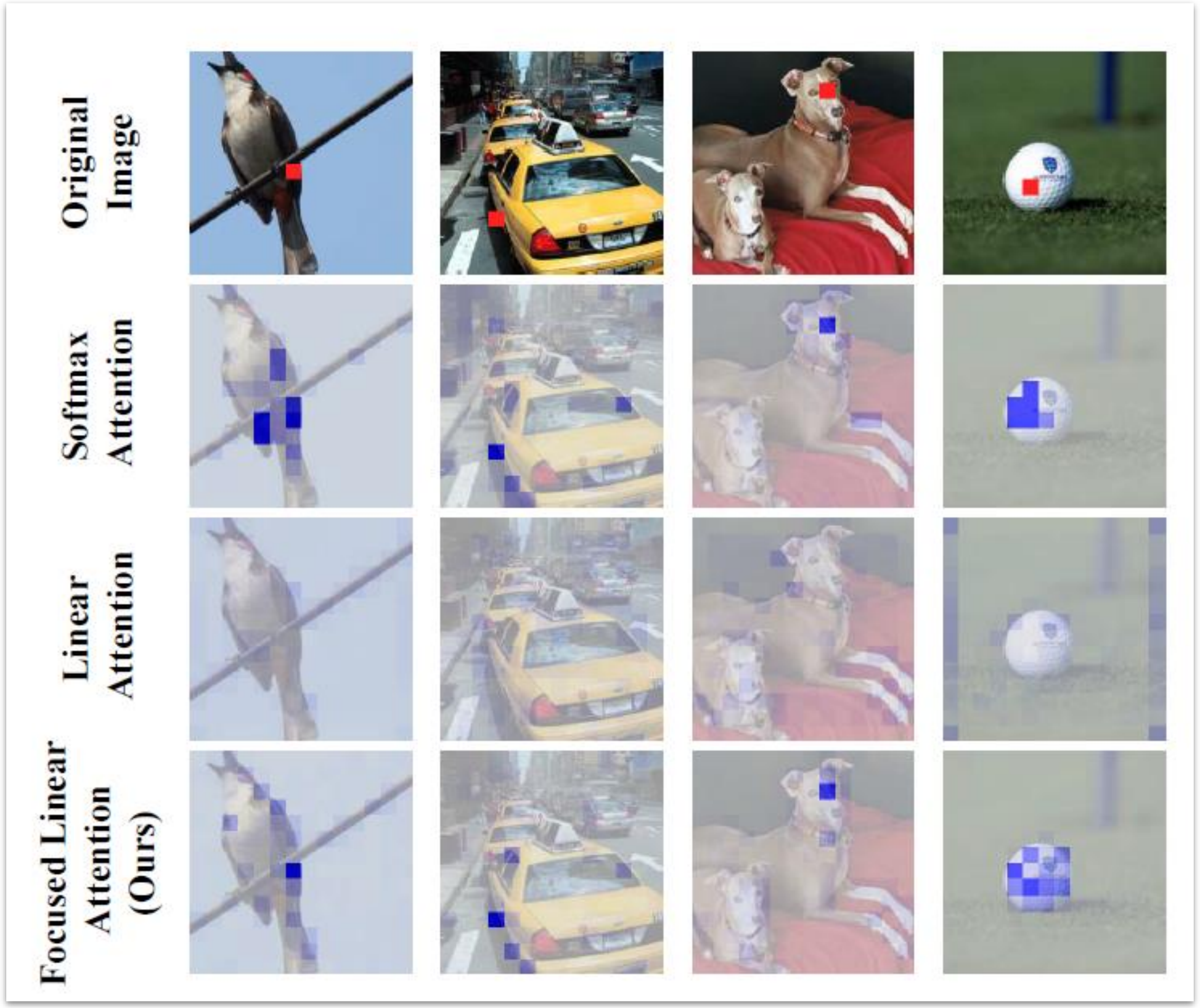
FLatten Transformer: Vision Transformer using Focused Linear Attention (ICCV 2023)

Regarding the O(N^2) complexity of attention computation,
linear attention* was proposed with **complexity of O(N)**.

But due to the performance drop when applied to vision tasks,
there were approaches with additional computations or mapping functions.

Here authors propose **focused** linear attention,
which achieves same focus and feature diversity as the softmax attention.





*Efficient Attention: Attention with Linear Complexities (WACV 2021)

Related Works

FLatten Transformer: Vision Transformer using Focused Linear Attention (ICCV 2023)

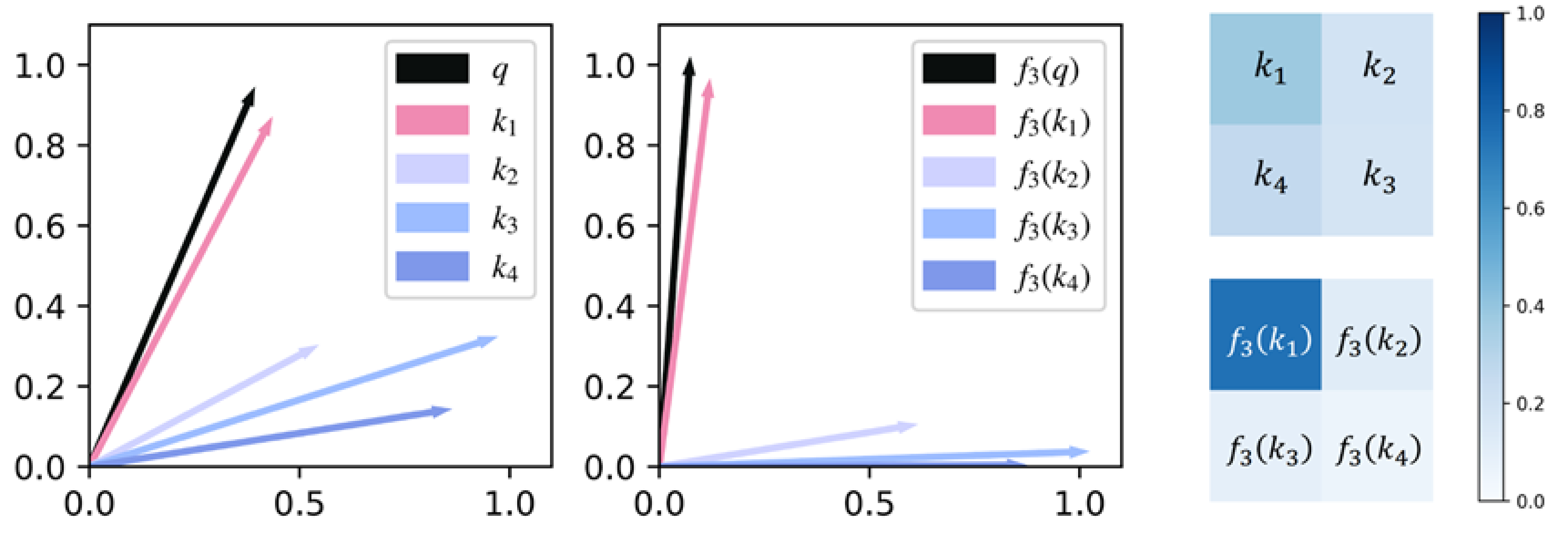
Authors proposed the **focused function:**

$$\mathrm{Sim}\left(Q_i, K_j\right) = \phi_p\left(Q_i\right) \phi_p\left(K_j\right)^T,$$

$$\text{where } \phi_p(x) = f_p\left(\mathrm{ReLU}(x)\right), \ f_p(x) = \frac{\|x\|}{\|x^{**p}\|} x^{**p},$$

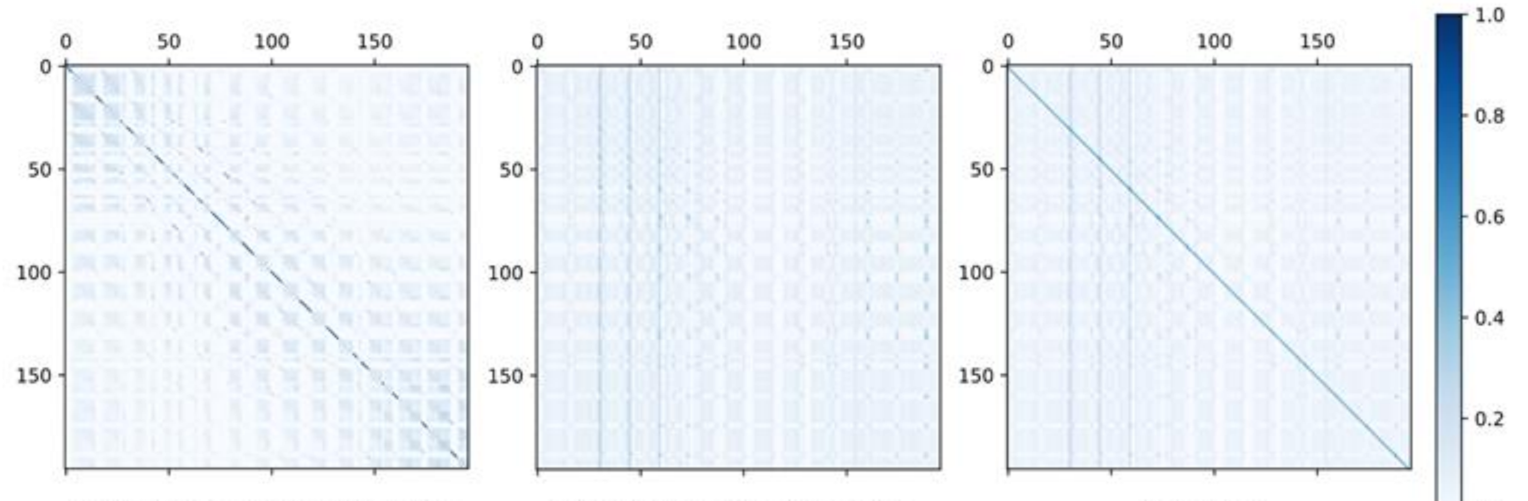
which makes similar query-key pairs closer,
and pushes dissimilar query-key pairs away from each other.

This changes the **distribution** of attentions,
which makes more "focused" attention map as the figure.



(a) Query and Keys (b) Attention Map

Also we can see that the attention matrix rank is preserved,
contrast to the original linear attention map.



(a) Softmax Attention Map
Matrix Rank: 196

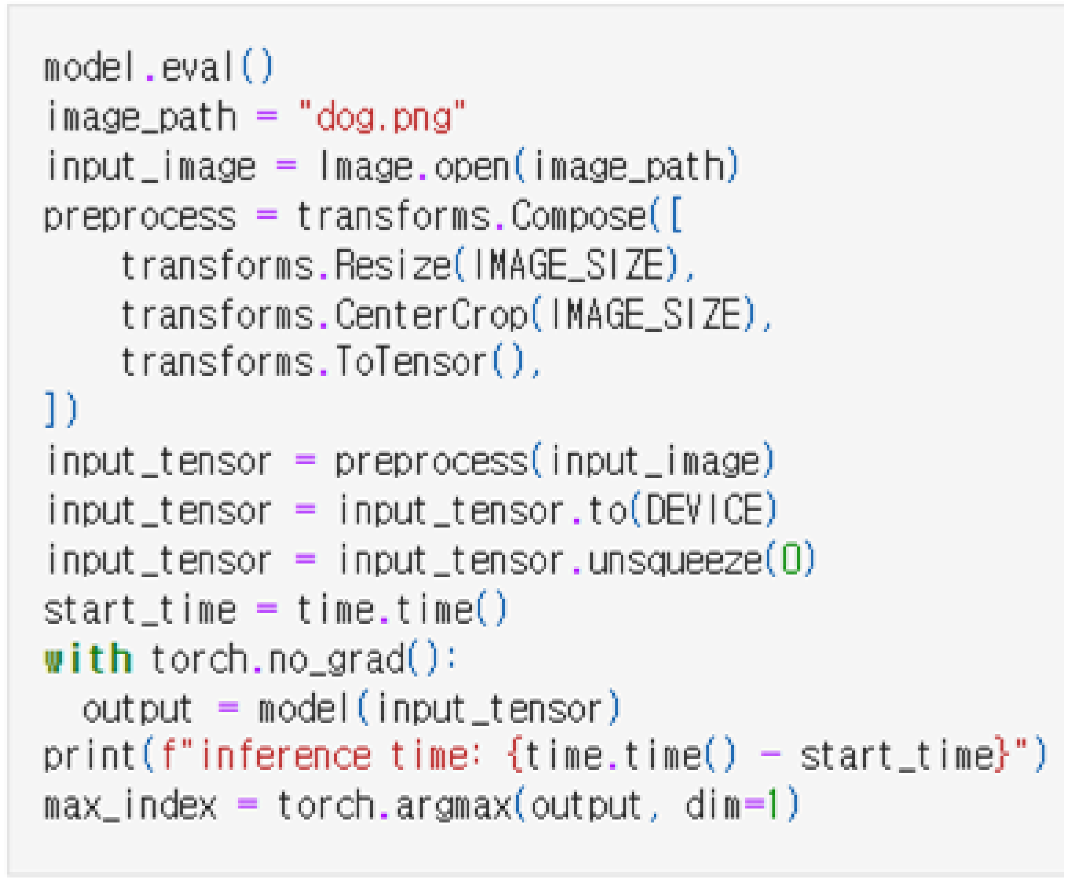
(b) Linear Attention Map
Matrix Rank: 54

(c) Ours
Matrix Rank: 196

Proposed method

Comparison of inference times across different sets by related work with various similarity functions.

Check inference time code



```
model.eval()
image_path = "dog.png"
input_image = Image.open(image_path)
preprocess = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.CenterCrop(IMAGE_SIZE),
    transforms.ToTensor(),
])
input_tensor = preprocess(input_image)
input_tensor = input_tensor.to(DEVICE)
input_tensor = input_tensor.unsqueeze(0)
start_time = time.time()
with torch.no_grad():
  output = model(input_tensor)
print(f"inference time: {time.time() - start_time}")
max_index = torch.argmax(output, dim=1)
```

Comparing inference time

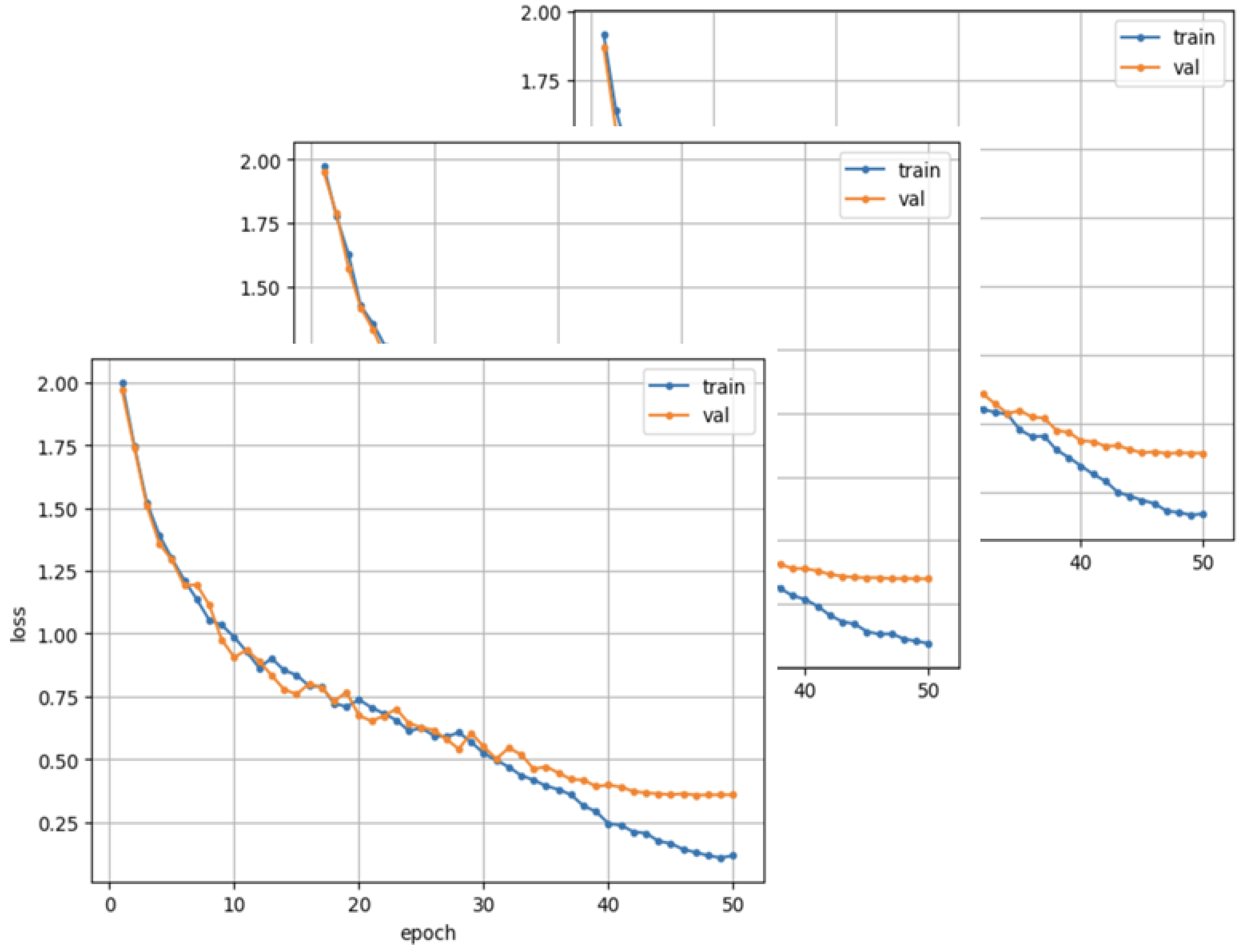
Possible Set	Inference time	
Swin Transformer (Base model)	The product of the query and key	0.0091863 s
Swin + FLatten	We tried but there is overfitting problem. We will try again.	
Swin + Different similarity functions (Jaccard)	$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} = \frac{\|A \cap B\|}{\|A\| + \|B\| - \|A \cap B\|}$	0.0096664 s
Swin + Different similarity functions (Euclidian)	$d(x,y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$	0.0087957 s
Swin + Different similarity functions (Cosine)	$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\\|\mathbf{A}\\|\\|\mathbf{B}\\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$	Using inner products similar to the base model

Input example image

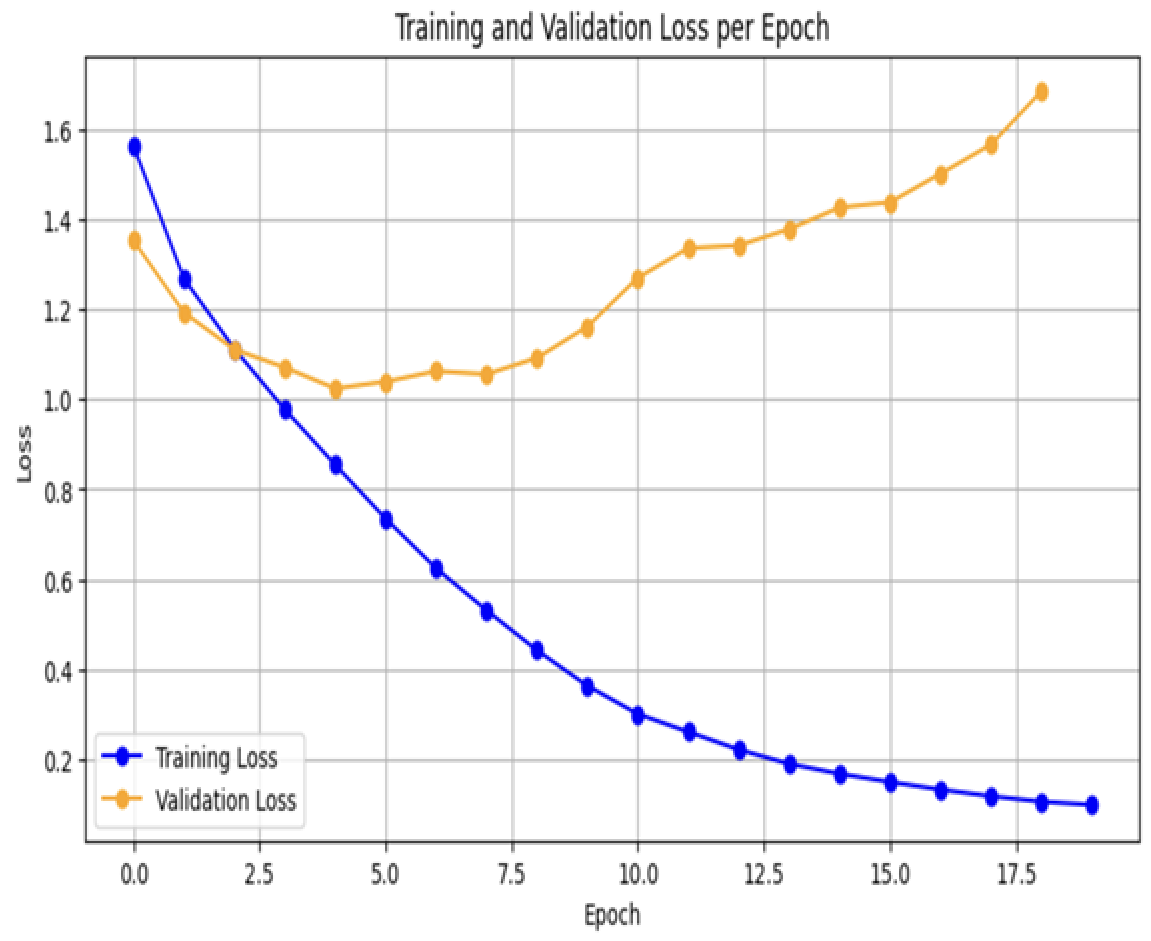
After proposal…
We will compare possible set by using more complex multiple image

Proposed method

Comparison of inference times across different sets by related work with various similarity functions.



Swin, Jaccard, Euclidian

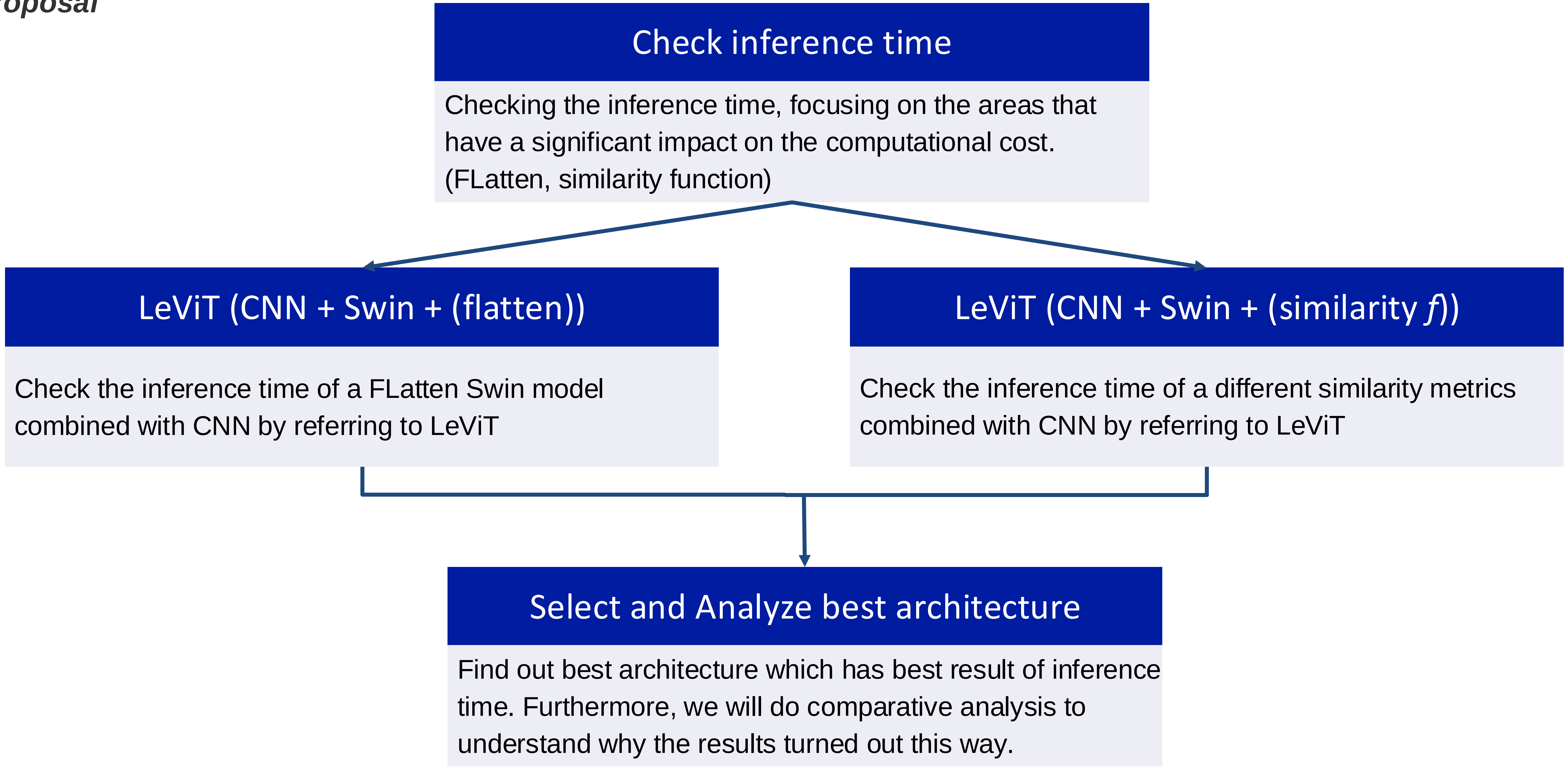


Swin + FLatten

Proposed method

Test more possible set, analyze the reasons and find the one with the best inference time.

After proposal



Check inference time

Checking the inference time, focusing on the areas that have a significant impact on the computational cost. (FLatten, similarity function)

LeViT (CNN + Swin + (flatten))

Check the inference time of a FLatten Swin model combined with CNN by referring to LeViT

LeViT (CNN + Swin + (similarity *f*))

Check the inference time of a different similarity metrics combined with CNN by referring to LeViT

Select and Analyze best architecture

Find out best architecture which has best result of inference time. Furthermore, we will do comparative analysis to understand why the results turned out this way.