

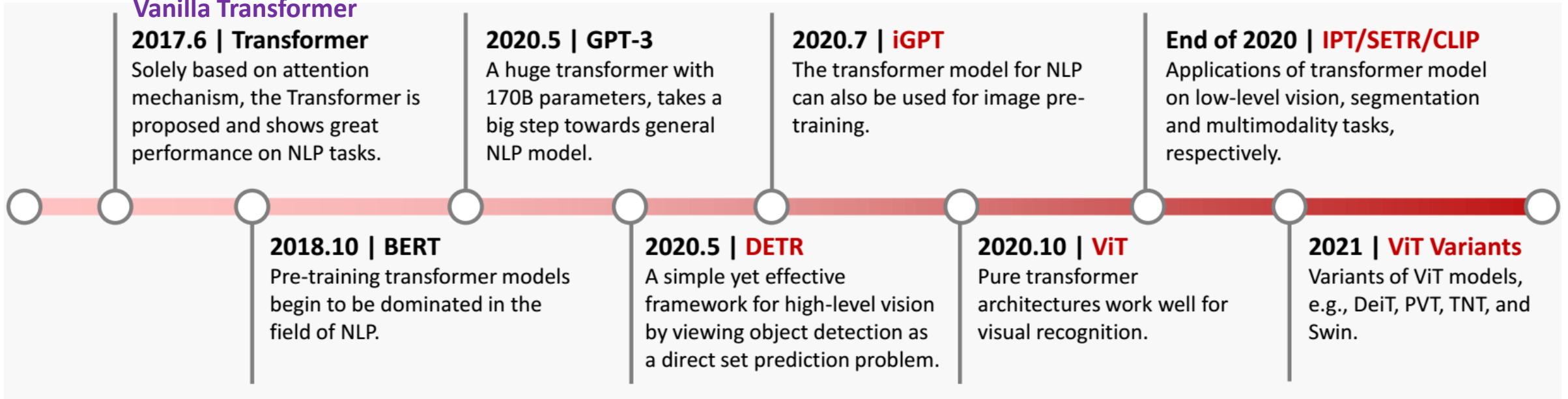
## 3.7 Architecture of Transformers

## 1.2.4 Transformers

A transformer is a deep learning model that generates significant weights to each part of the input data through using mechanisms of self-attention and cross-attention.

It is used primarily in the fields of natural language processing (NLP), computer vision (CV), and speech processing.

- TimeLine of Key milestones in the development of transformer \* Kai Han ....Feb2022

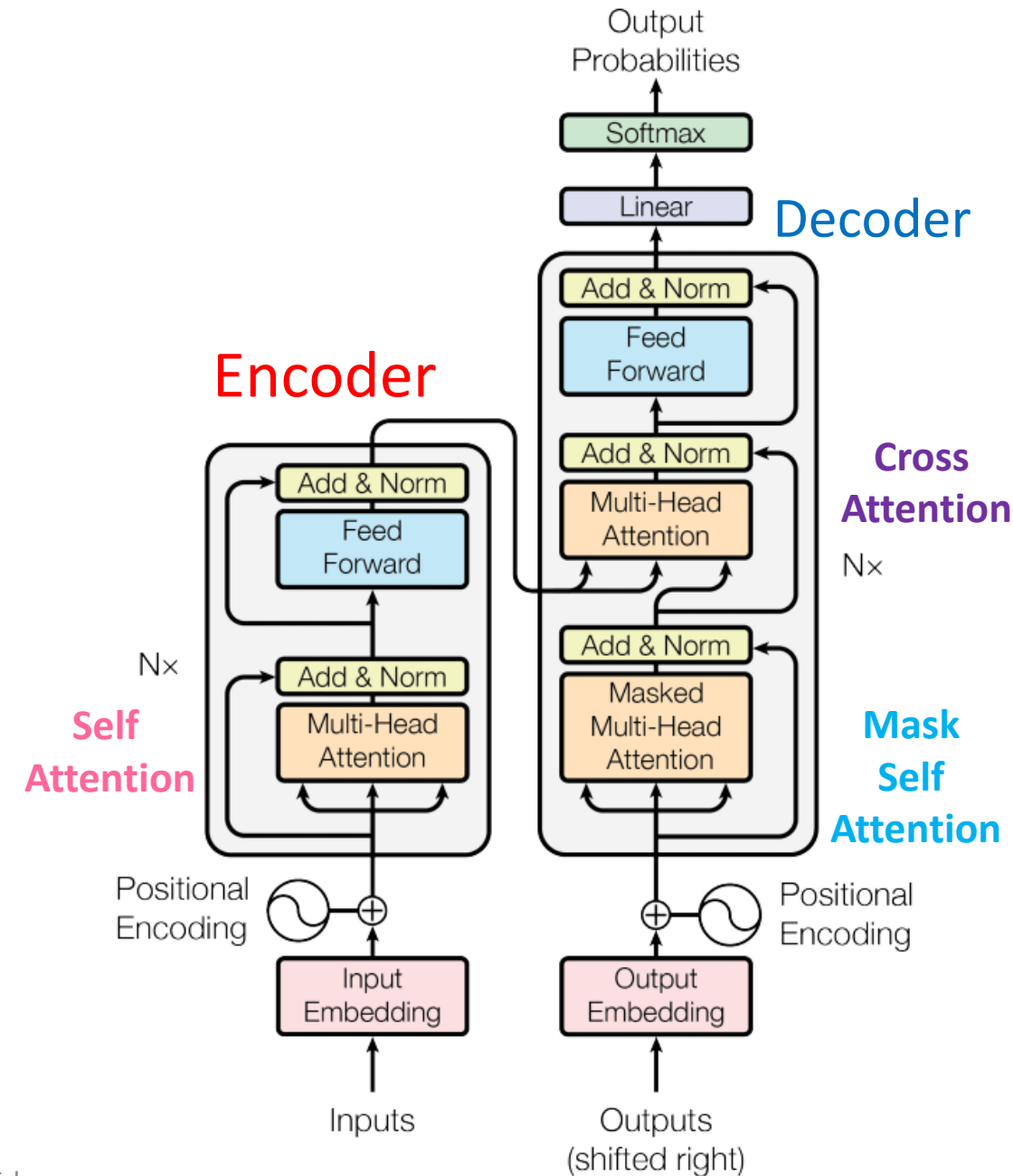


The vision transformer models are marked in red.

# Vanilla Transformer

\*Vaswani...2017 (Attention is all you need)

- a sequence-to-sequence model and consists of an encoder and a decoder, each of which is a stack of “N” identical blocks each *encoder block* is mainly composed of a multi-head self-attention module and a position-wise feed-forward network (FFN). In this work, the encoder is composed of a stack of N=6 identical layers.
- For building a deeper model, a residual connection is employed around each module, followed by Layer Normalization module.
- Compared to the encoder blocks, **decoder** blocks additionally insert **cross-attention** modules between the multi-head self-attention modules and the position-wise FFNs. In this work, the decoder is also composed of a stack of N=6 identical layers.
- Furthermore, the self-attention modules in the decoder are adapted to prevent each position from attending to subsequent positions.



## The key modules of the vanilla Transformer

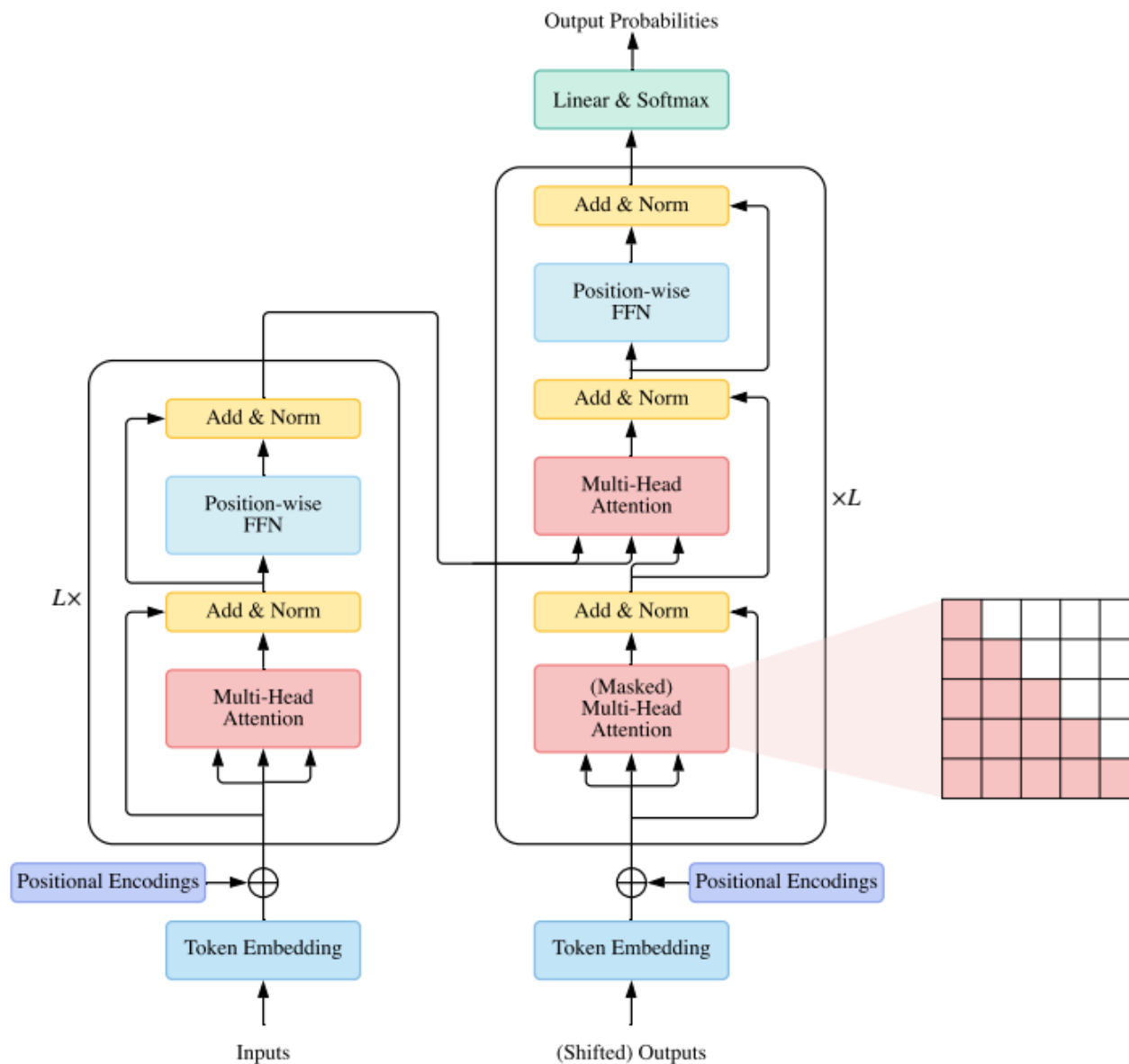
### 1. Attention Modules

- Single Attention
- Multi Head-Attention
  - Self-attention
  - Masked Self-attention
  - Cross-attention

### 2. Position-wise FFN (Feed Forward Network)

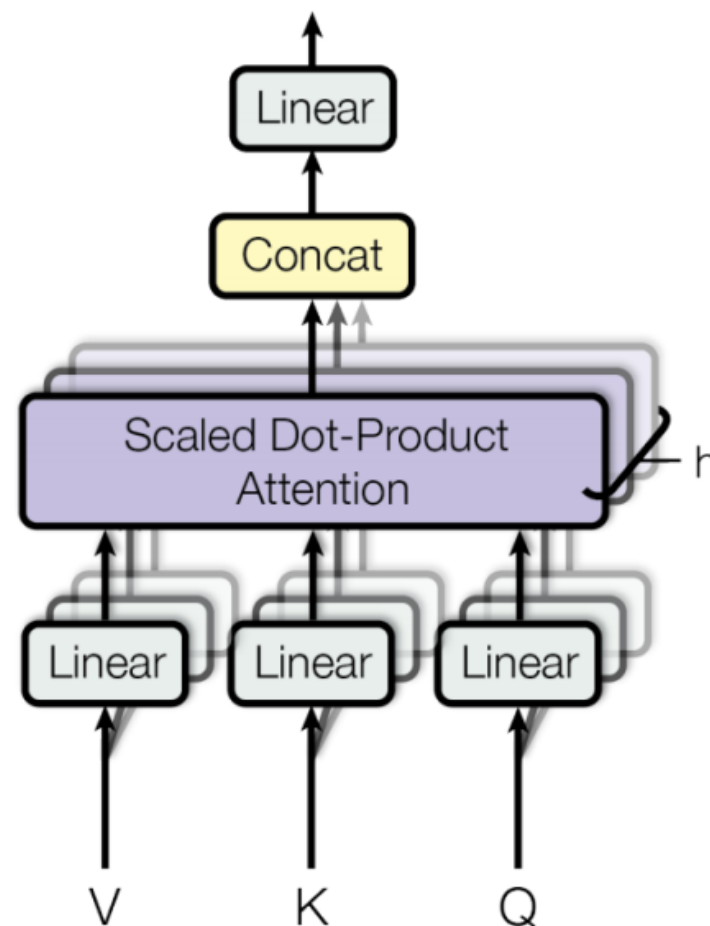
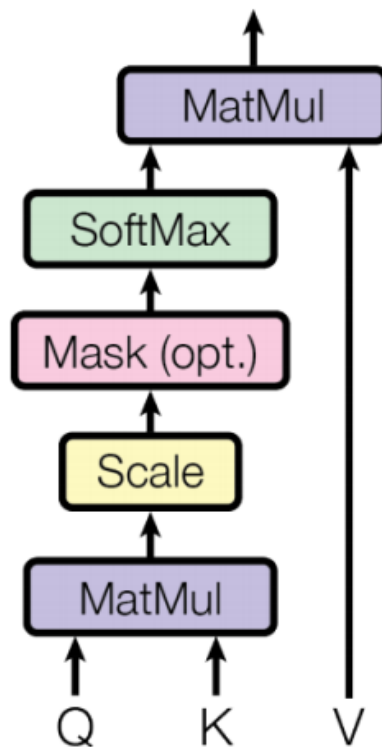
### 3. Residual Connection and Normalization.

### 4. Position Encodings.



Overview of vanilla Transformer architecture

In this work we  
employ  $h=8$   
parallel attention  
layers, or heads



Single-attention function    Multi-head attention function.

# 1. Attention Modules

- **Single attention function**

Transformer adopts attention mechanism with Query-Key-Value (QKV) model. Given the packed matrix representations of queries , keys , and values , the scaled dot-product attention used by Transformer is given by

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right), Q \in R^{N \times D_k}, K \in R^{M \times D_k}, V \in R^{M \times D_v}$$

N: length of Query, M: length of Keys(or values)-  $D_k$ : Dimension of query and key  $D_v$ : Dimension of values

- Softmax is applied in a row-wise manner.
- The dot-products of queries and keys are divided by  $\sqrt{D_k}$  to alleviate gradient vanishing problem of the softmax function.

- **Multi head attention function**

Instead of simply applying a single attention function, Transformer uses multi-head attention, where the original queries, keys and values are with  $H$  different sets of learned projections.

$$\text{MultiHeadAttn}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_H)W^O$$

$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad i = 1, 2, \dots, H$$

The model then concatenates all the outputs.

# Three types of attention (in terms of the source of queries and key-value pairs).

- *Self-attention*

In Transformer encoder, we set  $Q = K = V = X$ , where  $X$  is the outputs of the previous layer.

- *Masked Self-attention*

In the Transformer decoder, the self-attention is restricted such that queries at each position can only attend to all key-value pairs up to and including that position.

To enable parallel training, this is typically done by applying a mask function to the un-normalized attention matrix  $\hat{A} = \exp(\frac{QK^T}{\sqrt{D_k}})$  where the illegal positions are masked out by setting  $\hat{A}_{ij} = -\infty$  if  $i < j$ . This kind of self-attention is often referred to as autoregressive or causal attention.

- *Cross-attention*

The queries are projected from the outputs of the previous (decoder) layer, whereas the keys and values are projected using the outputs of the encoder.

## 2. Position-wise FFN.

The position-wise FFN is a fully connected feed-forward module that operates separately and identically on each position

$$\text{FFN}(H') = \text{ReLU}(H'W^1 + b^1)W^2 + b^2$$

where  $H'$  is the outputs of previous layer, and  $W^1 \in R^{D_m \times D_f}$ ,  $W^2 \in R^{D_f \times D_m}$ ,  $b^1 \in R^{D_f}$ ,  $b^2 \in R^{D_m}$  are trainable parameters. Typically the intermediate dimension  $D_f$  of the FFN is set to be larger than  $D_m$

## 3. Residual Connection and Normalization

In order to build a deep model, Transformer employs a residual connection [49] around each module, followed by Layer Normalization [4]. For instance, each Transformer encoder block may be written as

$$H' = \text{LayerNorm}(\text{SelfAttention}(X) + X)$$

$$H = \text{LayerNorm}(\text{FFN}(H') + H')$$

where  $\text{SelfAttention}(\cdot)$  denotes self attention module and  $\text{LayerNorm}(\cdot)$  denotes the layer normalization operation

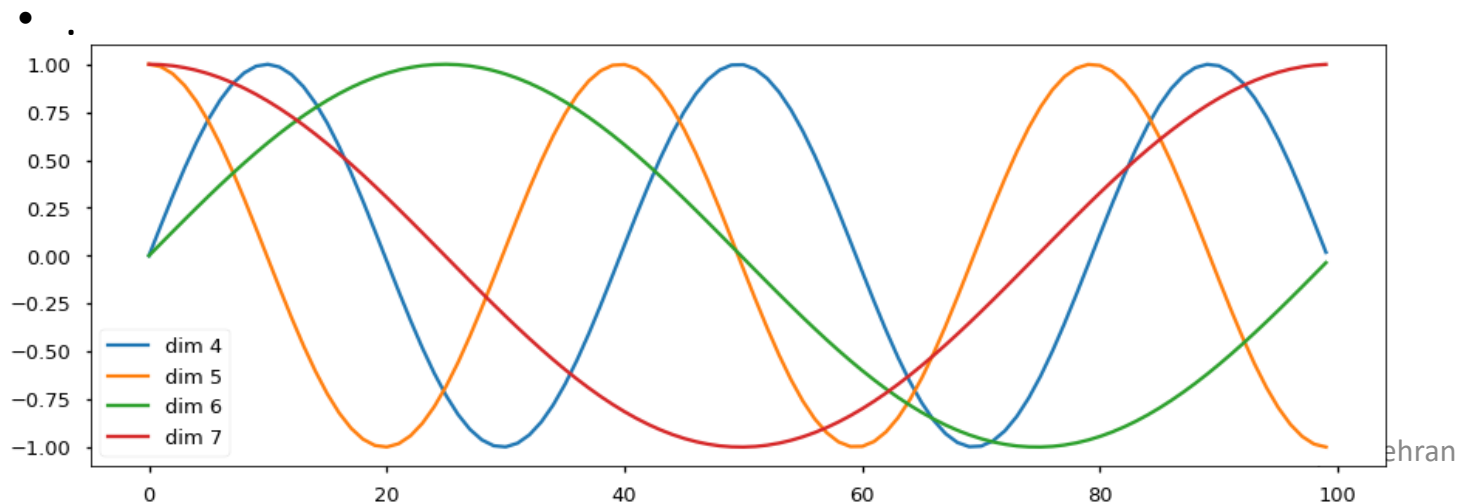
## 4. Position Encodings.

Since Transformer doesn't introduce recurrence or convolution, it is ignorant of positional information (especially for the encoder). Thus additional positional representation is needed to model the ordering of tokens



# Positional Encoding

- Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add “positional encodings” to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension  $d_{\text{model}}$
- as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed ([cite](#)).
- In this work, we use sine and cosine functions of different frequencies:  $\text{PE}(\text{pos}, 2i) = \sin(\text{pos}/10000^{2i/d_{\text{model}}})$
- $\text{PE}(\text{pos}, 2i+1) = \cos(\text{pos}/10000^{2i/d_{\text{model}}})$
- where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from  $2\pi$  to  $10000 \cdot 2\pi$ . We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k,  $\text{PE}_{\text{pos}+k}$  can be represented as a linear function of  $\text{PE}_{\text{pos}}$ .
- In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of  $\text{P}_{\text{drop}}=0.1$



# Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position **separately and identically**.

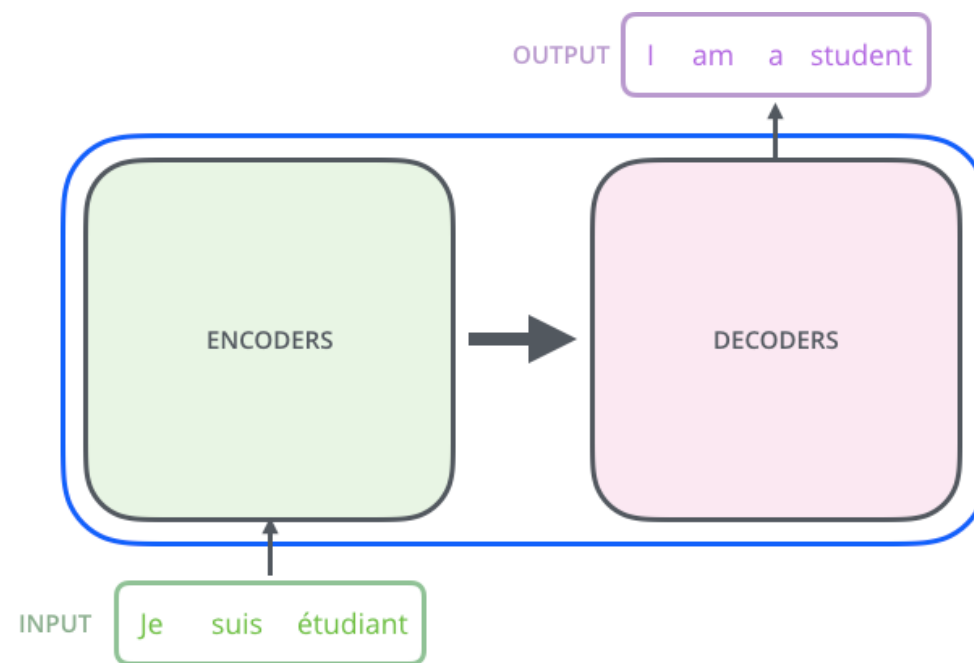
This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer.

**Another way of describing this is as two convolutions with kernel size 1.**

The dimensionality of input and output is  $d_{\text{model}}=512$ , and the inner-layer has dimensionality  $d_{\text{ff}}=2048$ .

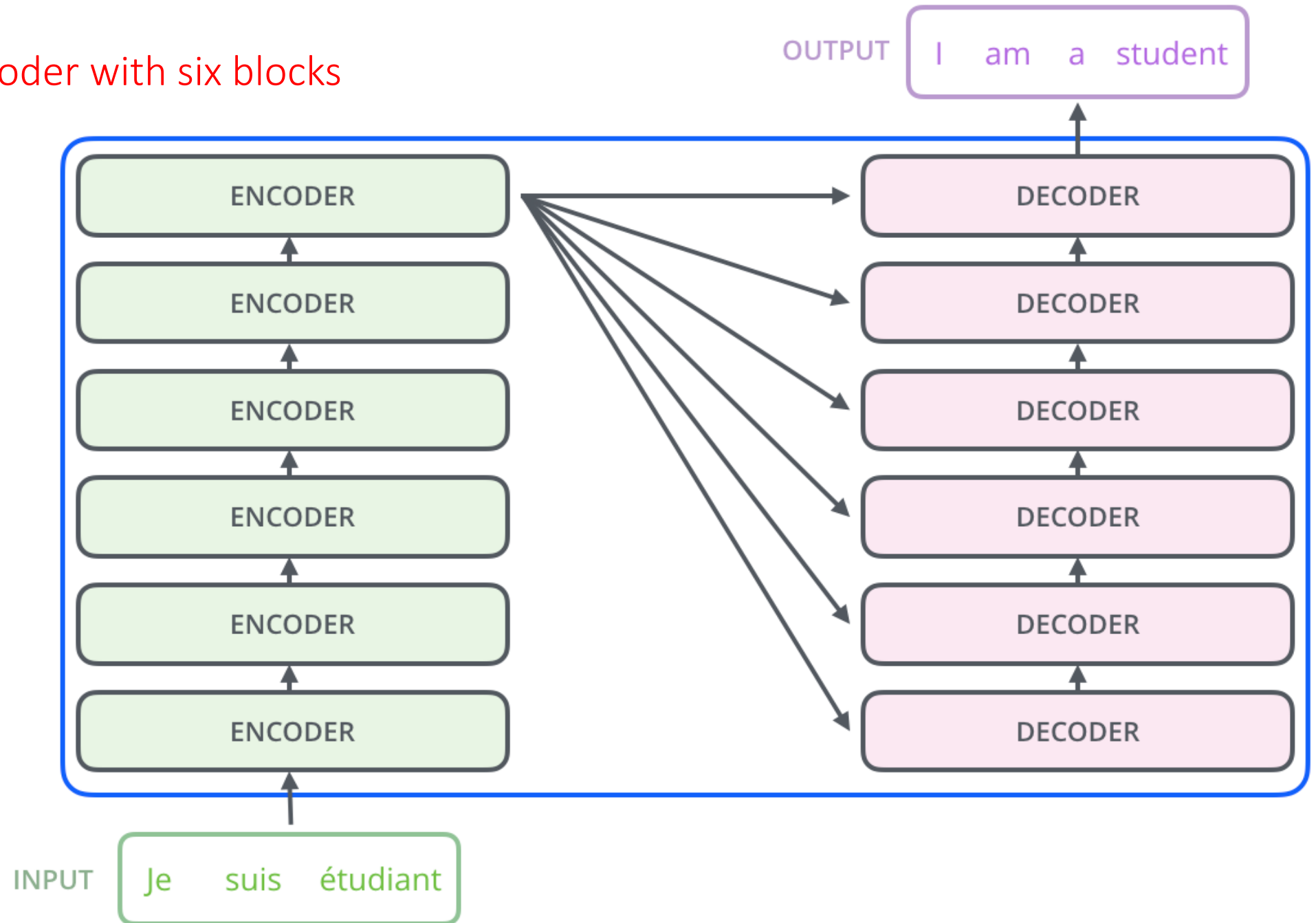


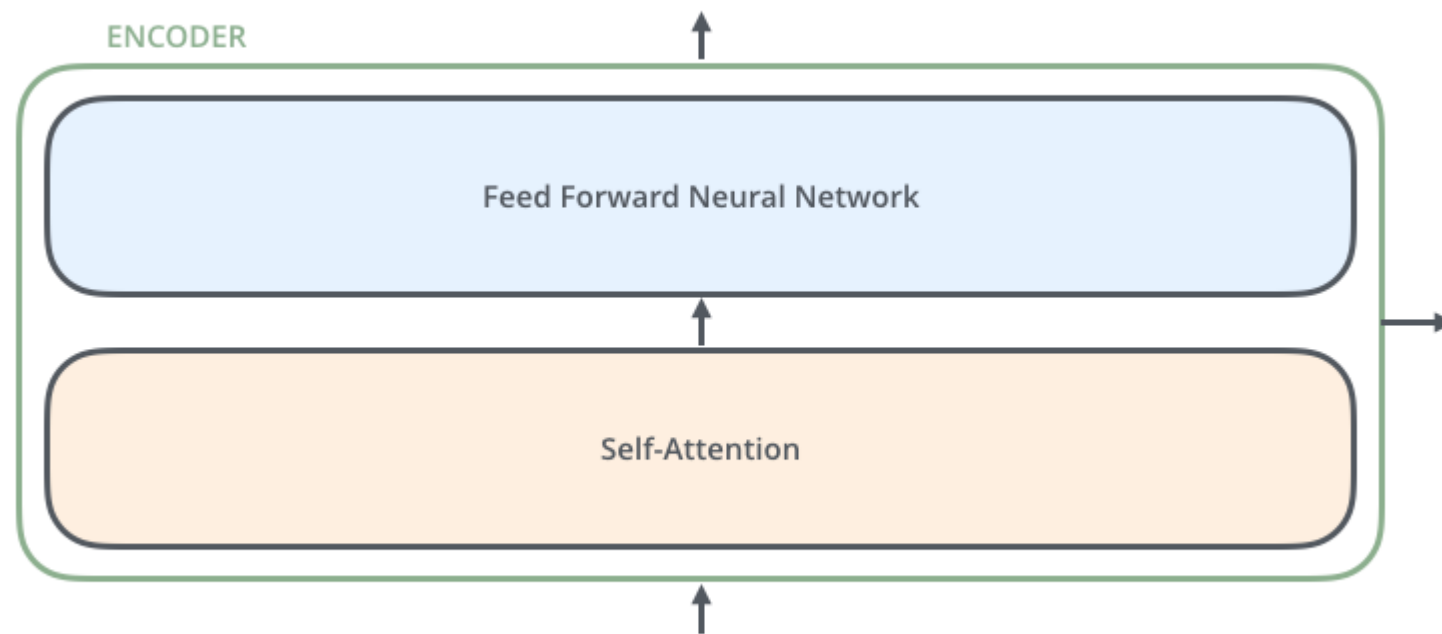
[Jay Alammar](#)

Visualizing machine learning one concept at a time.

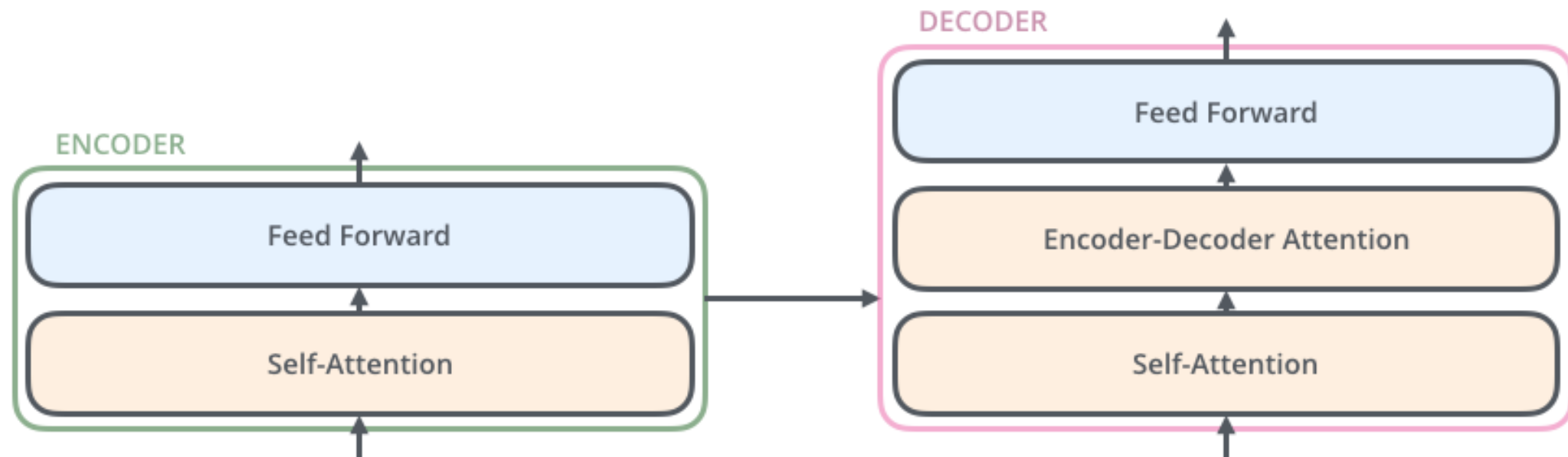
[@JayAlammar](#) on Twitter. [YouTube Channel](#)

## Encoder and Decoder with six blocks

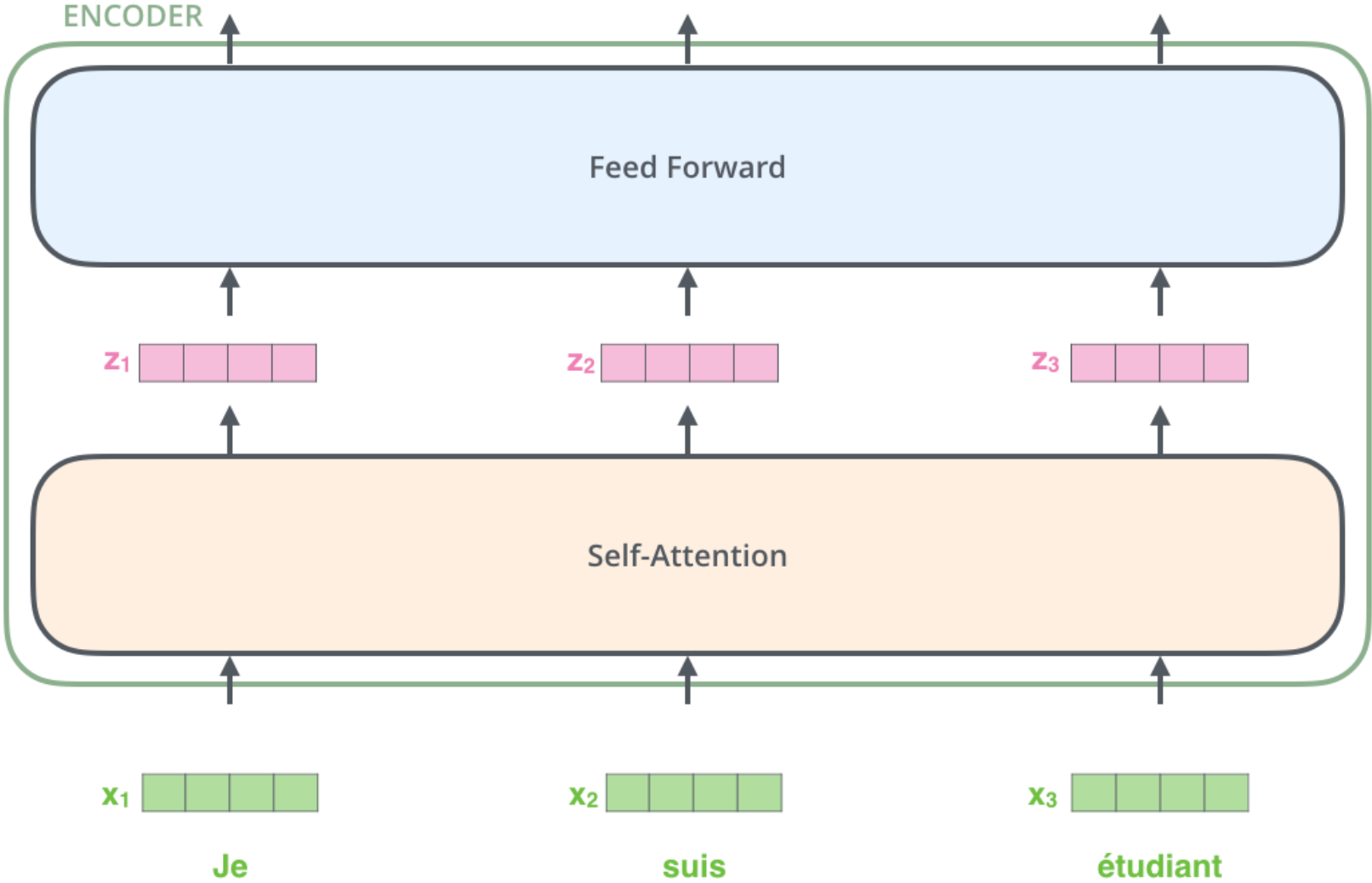




## Encoder and Decoder with one block

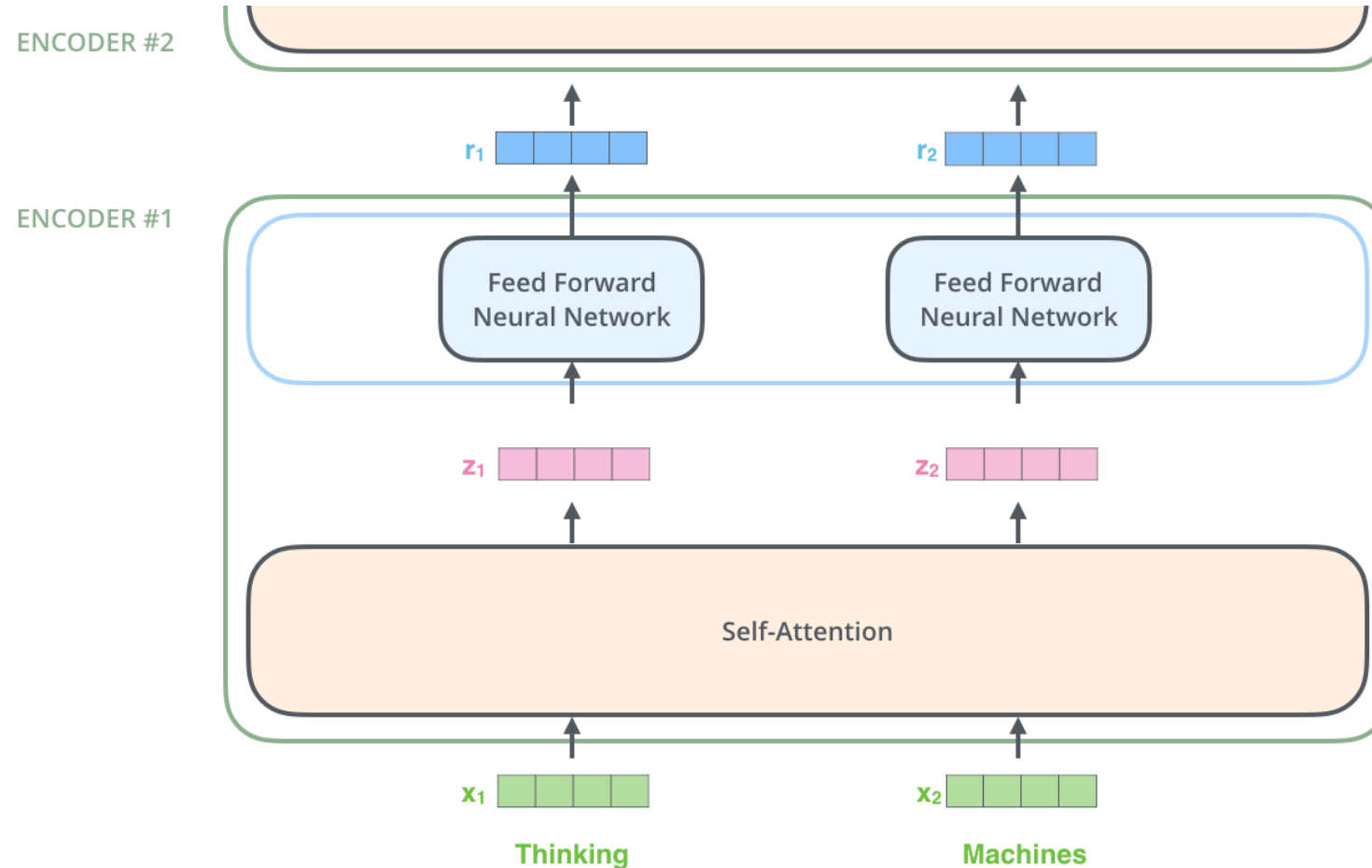


Bringing The Tensors Into The Picture



# Now We're Encoding!

The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.





# Self-Attention in Detail

Multiplying  $x_1$  by the  $W^Q$  weight matrix produces  $q_1$ , the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

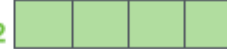
Input

Thinking

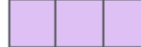
Machines

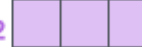
Embedding

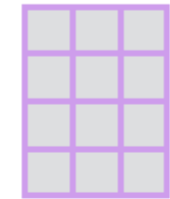
$x_1$  

$x_2$  

Queries

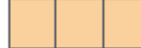
$q_1$  

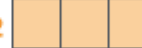
$q_2$  

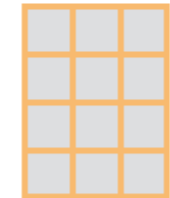


$W^Q$

Keys

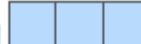
$k_1$  

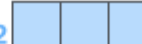
$k_2$  

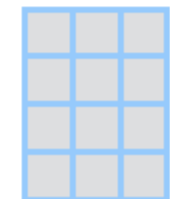


$W^K$

Values

$v_1$  

$v_2$  



$W^V$

Input

Embedding

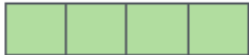
Queries

Keys

Values

Score

Thinking

$x_1$  

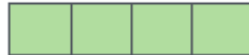
$q_1$  

$k_1$  

$v_1$  

$$q_1 \cdot k_1 = 112$$

Machines

$x_2$  

$q_2$  

$k_2$  

$v_2$  

$$q_1 \cdot k_2 = 96$$

Input

Thinking

Machines

Embedding

$x_1$  

$x_2$  

Queries

$q_1$  

$q_2$  

Keys

$k_1$  

$k_2$  

Values

$v_1$  

$v_2$  

Score

$$q_1 \cdot k_1 = 112$$

$$q_1 \cdot k_2 = 96$$

Divide by 8 (  $\sqrt{d_k}$  )

14

12

Softmax

0.88

0.12

Input

Embedding

Queries

Keys

Values

Score

Divide by 8 (  $\sqrt{d_k}$  )

Softmax

Softmax  
X  
Value

Sum

Thinking

$x_1$

$q_1$

$k_1$

$v_1$

$q_1 \cdot k_1 = 112$

14

0.88

$v_1$

$z_1$

Machines

$x_2$

$q_2$

$k_2$

$v_2$

$q_2 \cdot k_2 = 96$

12

0.12

$v_2$

$z_2$

# Matrix Calculation of Self-Attention

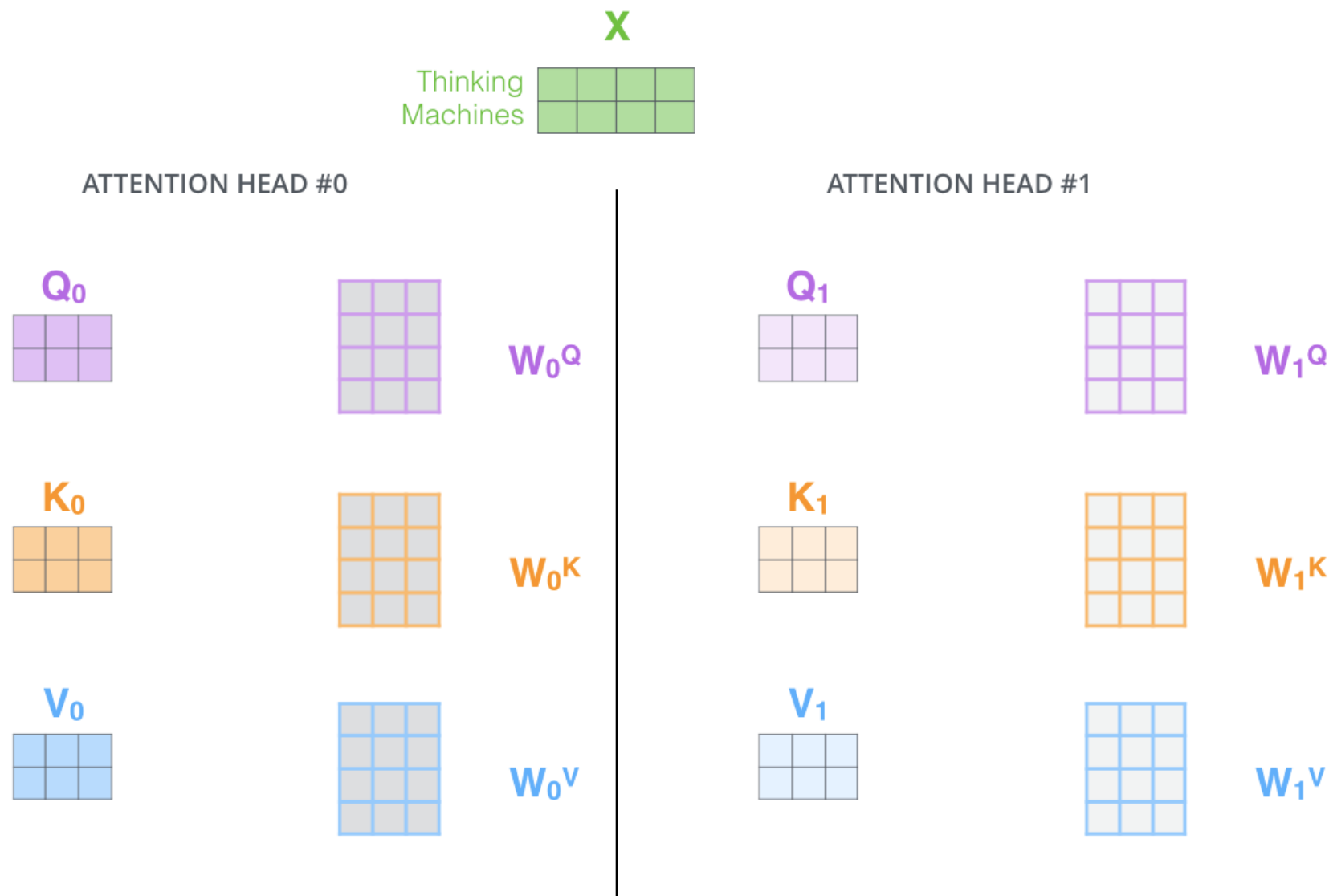
Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)



# The self-attention calculation in matrix form

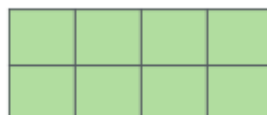
$$\text{softmax}\left(\frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \end{matrix}}{\sqrt{d_k}}\right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$
$$= \begin{matrix} \text{Z} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

# The Beast With Many Heads



X

Thinking  
Machines

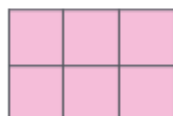


Calculating attention separately in  
eight different attention heads



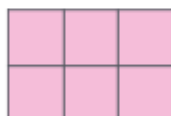
ATTENTION  
HEAD #0

$Z_0$



ATTENTION  
HEAD #1

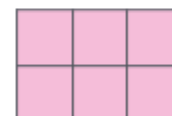
$Z_1$



...

ATTENTION  
HEAD #7

$Z_7$





$Z_0$	$Z_1$	$Z_2$	$Z_3$	$Z_4$	$Z_5$	$Z_6$	$Z_7$

X

$$= \begin{matrix} & Z \\ \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \end{matrix}$$

wo

1) This is our input sentence\*

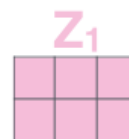
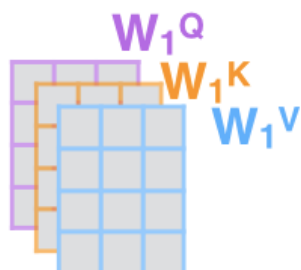
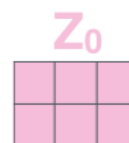
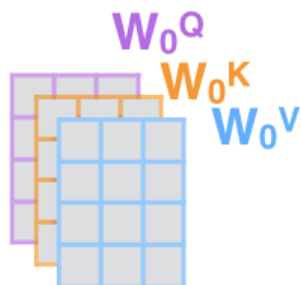
2) We embed each word\*

3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices

4) Calculate attention using the resulting  $Q/K/V$  matrices

5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer

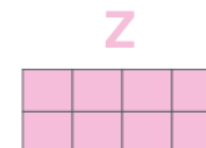
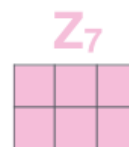
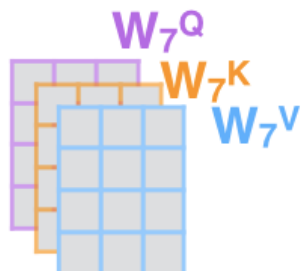
Thinking  
Machines



...

...

...

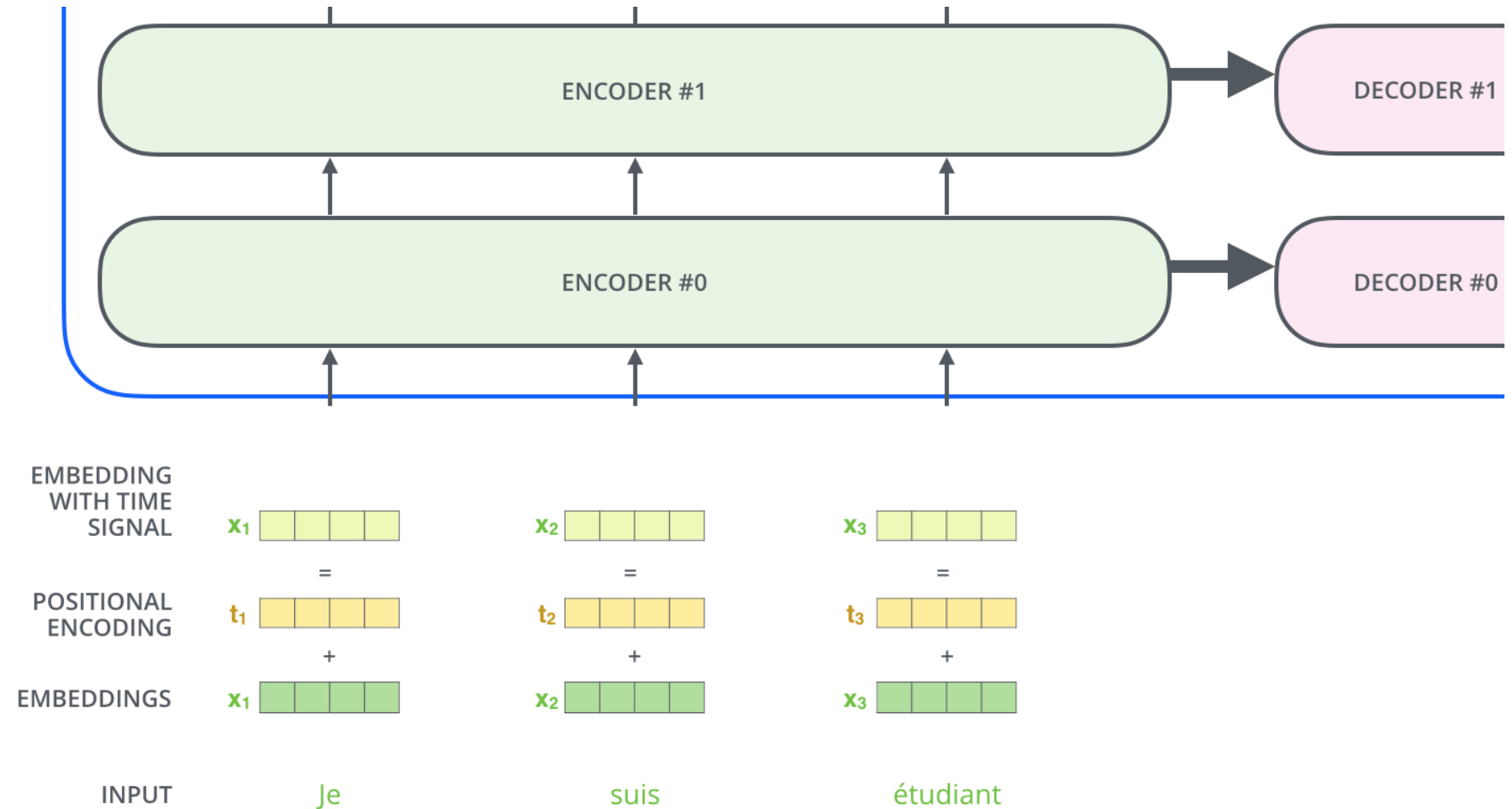


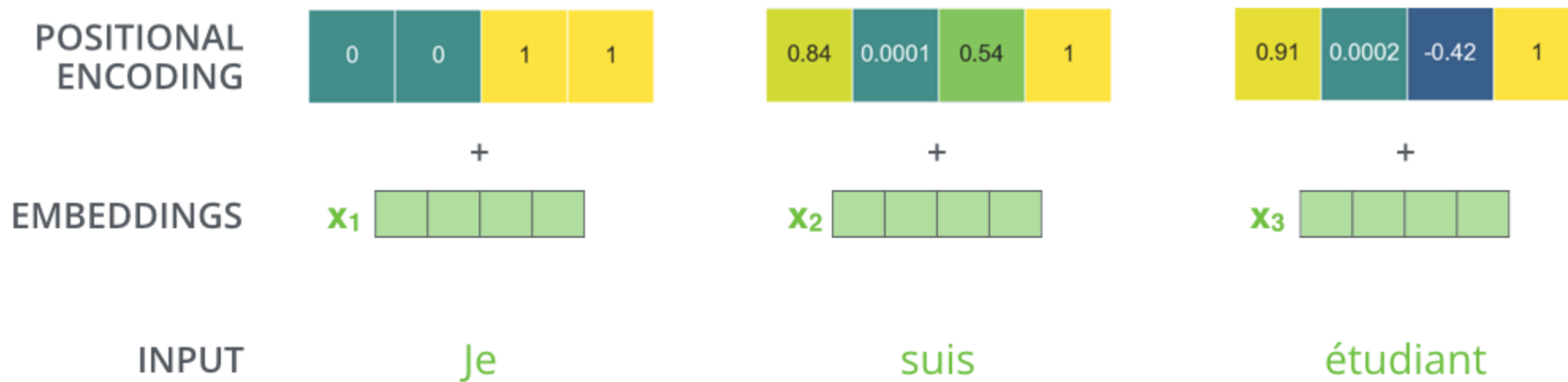
\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



# Representing The Order of The Sequence Using Positional Encoding

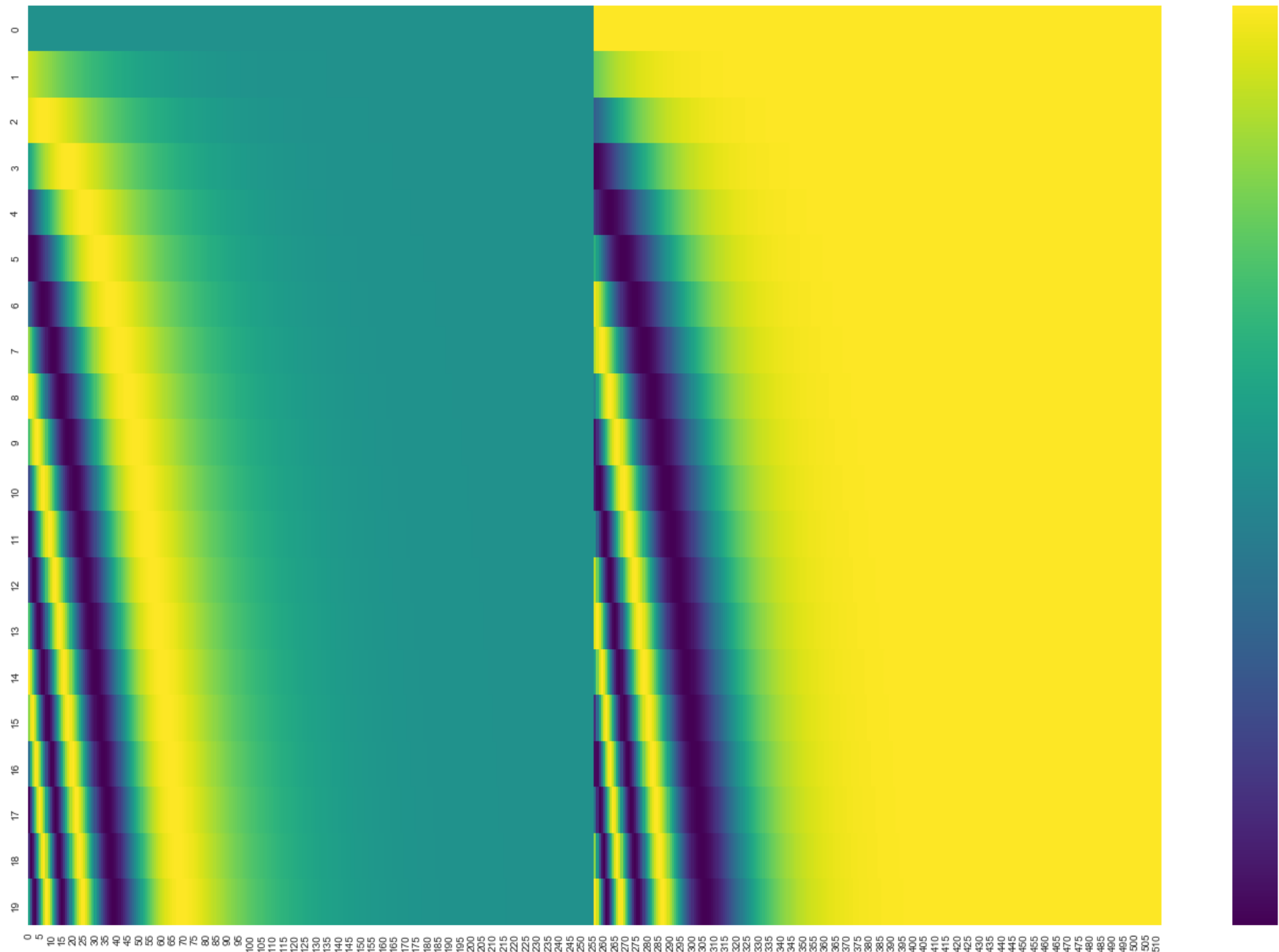
To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.



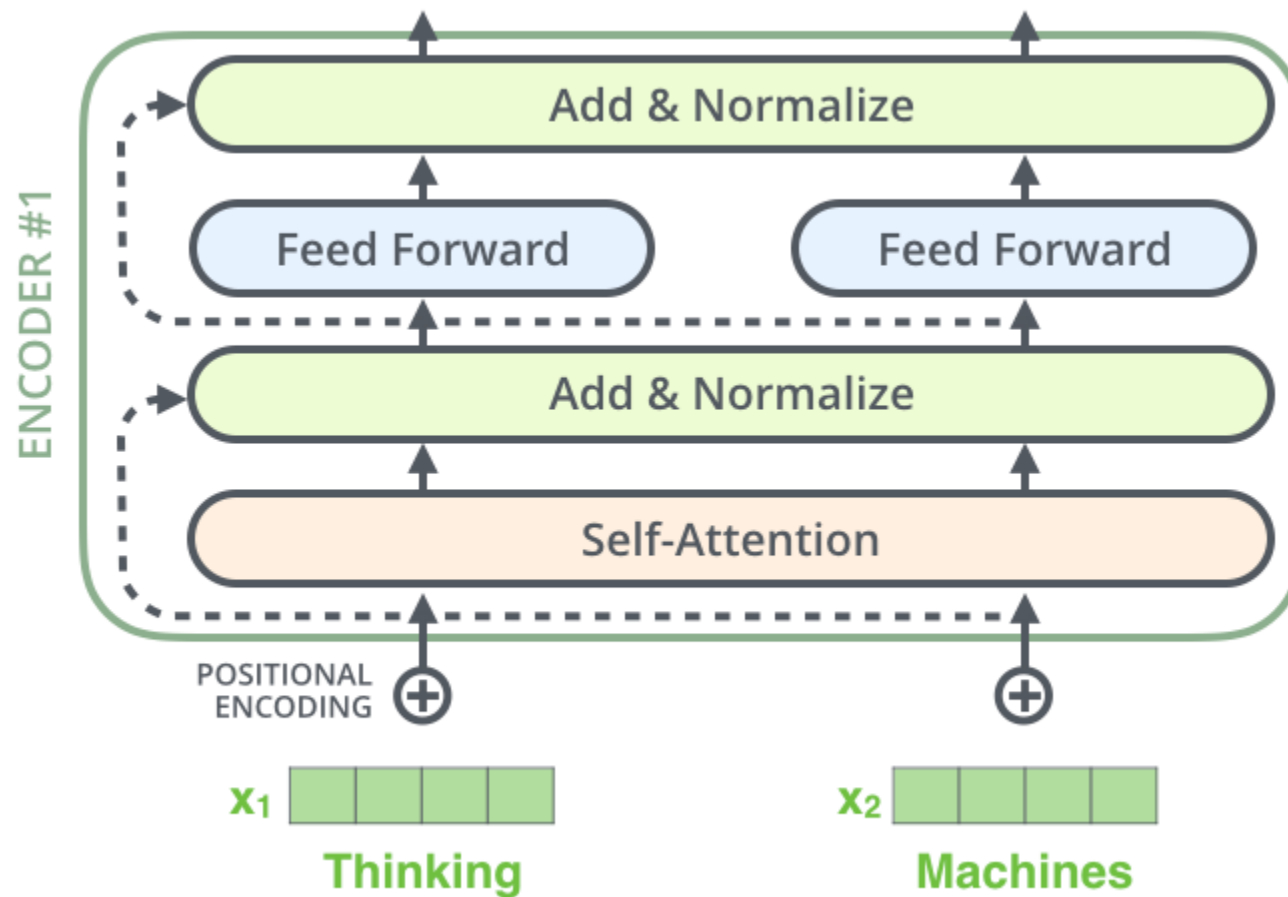


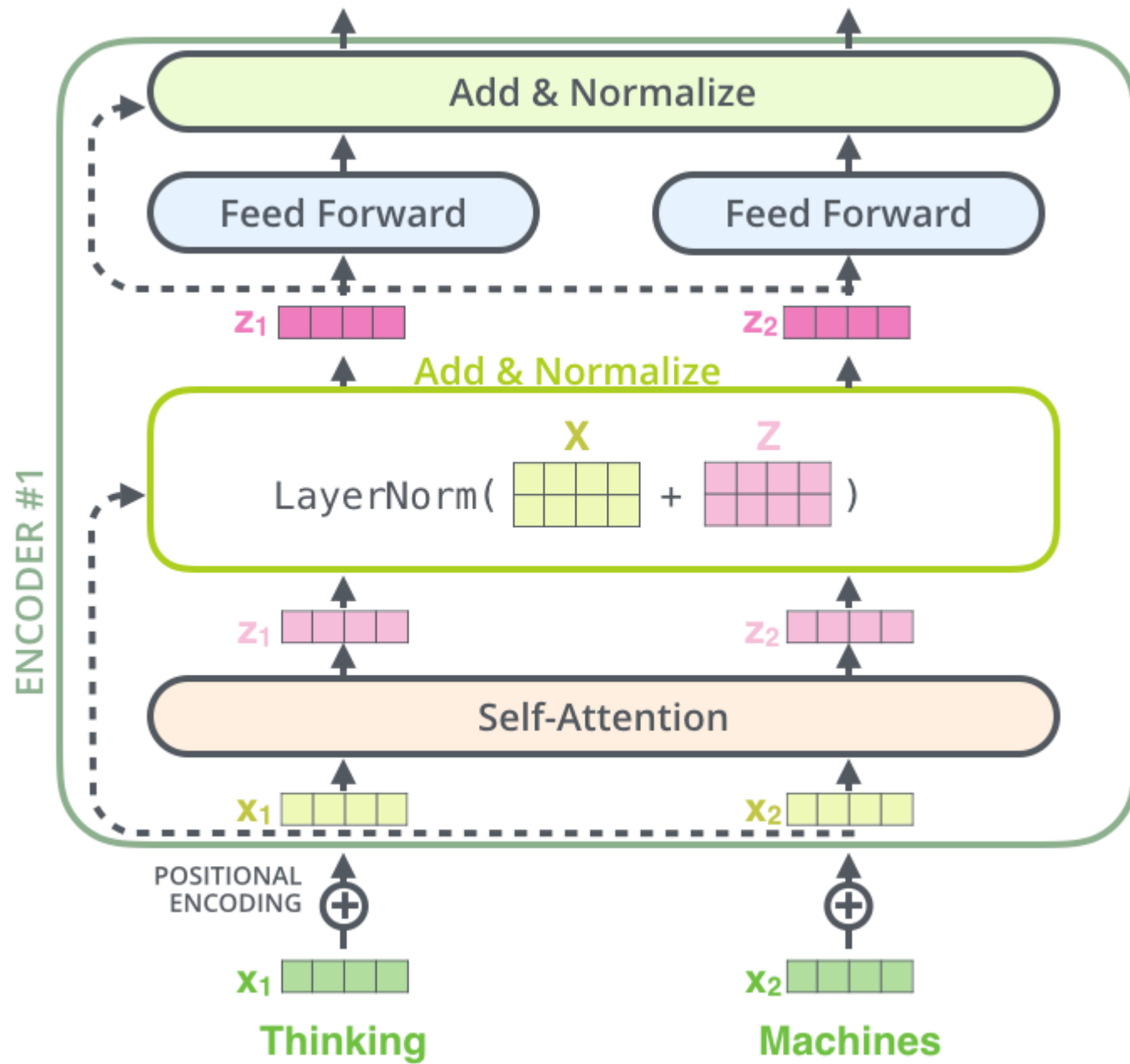
A real example of positional encoding with a toy embedding size of 4

A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.



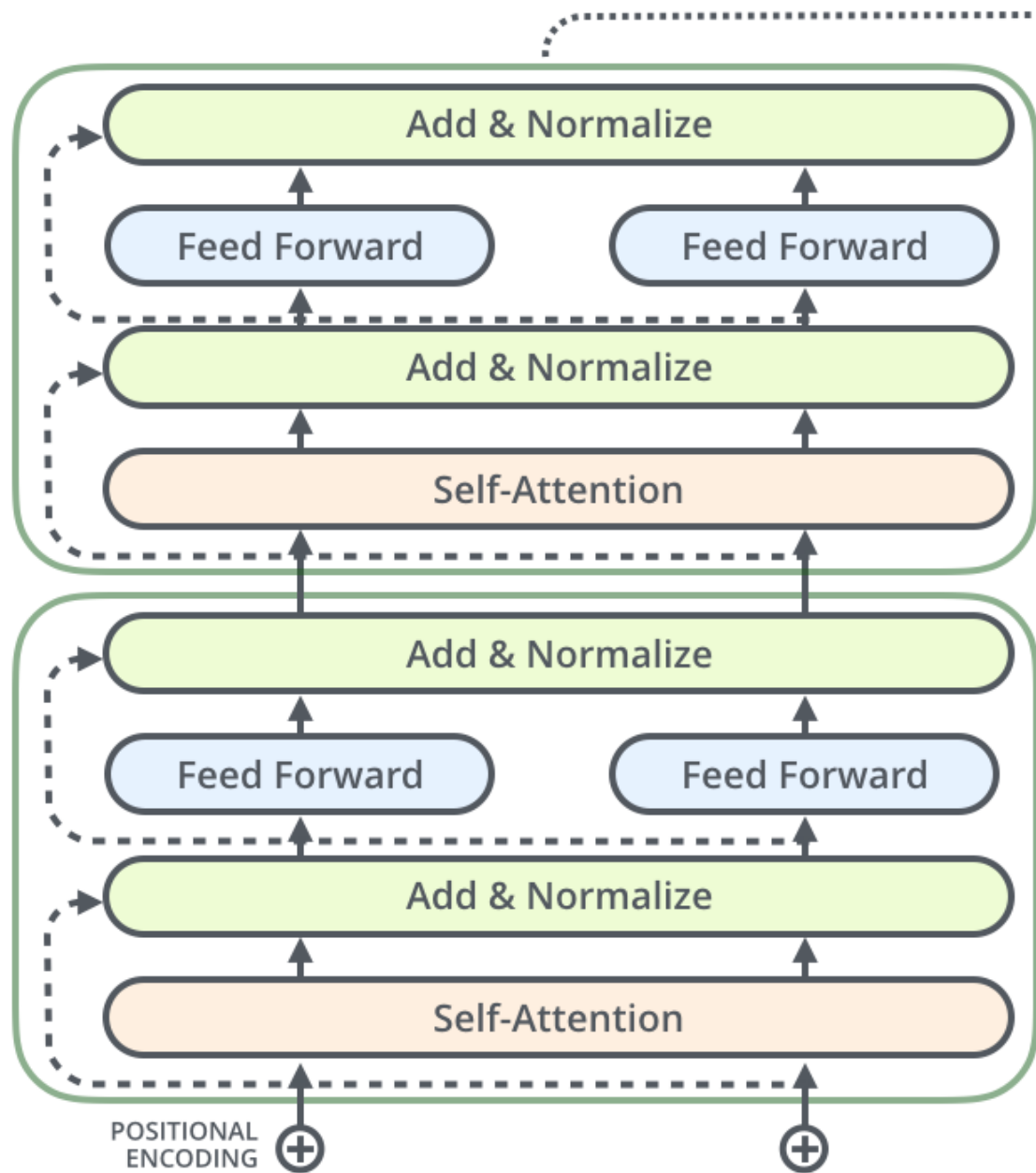
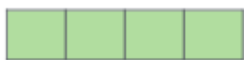
# The Residuals



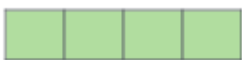


ENCODER #2

ENCODER #1

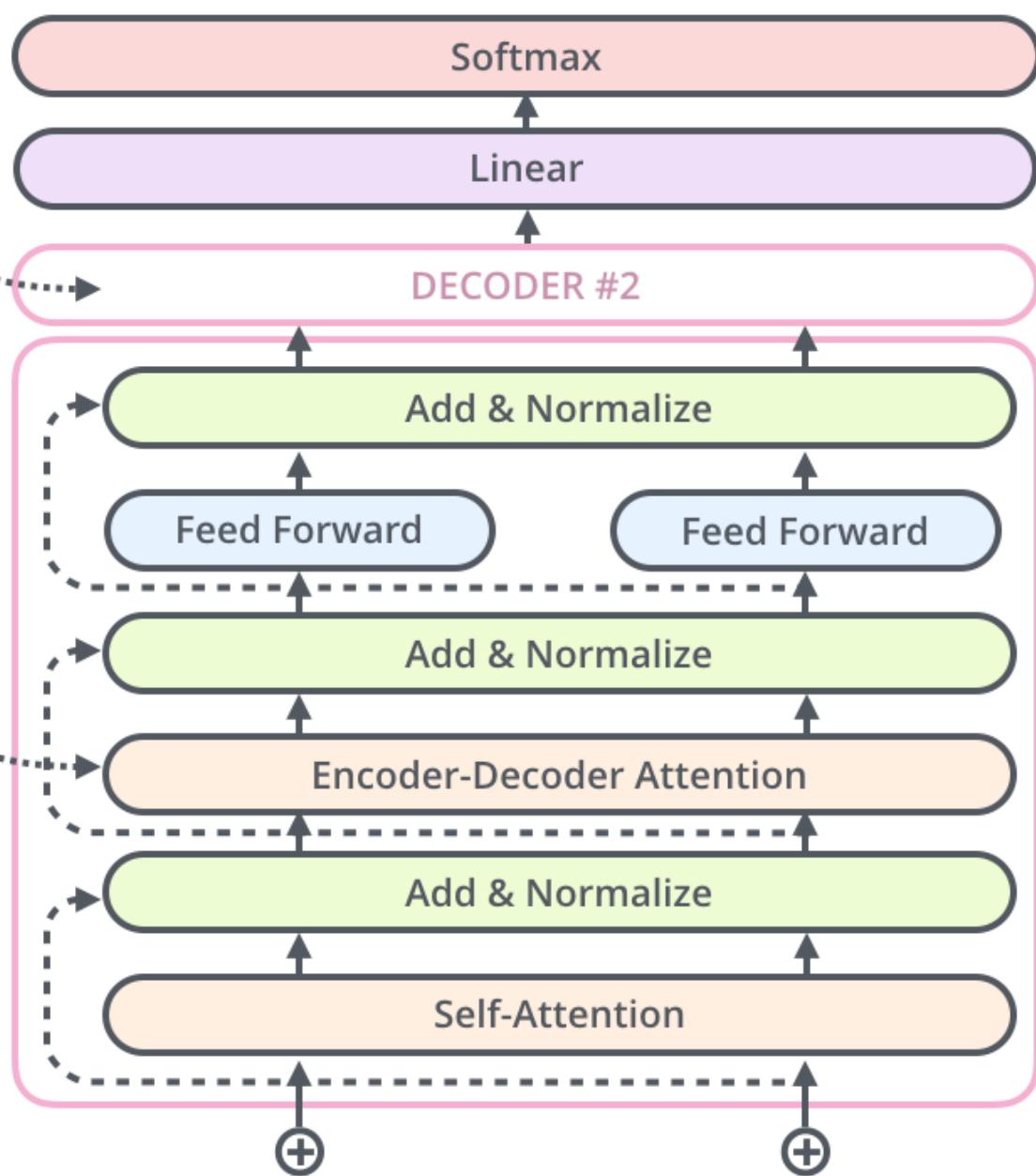
POSITIONAL  
ENCODING $x_1$ 

Thinking

 $x_2$ 

Machines

DECODER #1



Softmax

Linear

DECODER #2

Add &amp; Normalize

Feed Forward

Feed Forward

Add &amp; Normalize

Encoder-Decoder Attention

Add &amp; Normalize

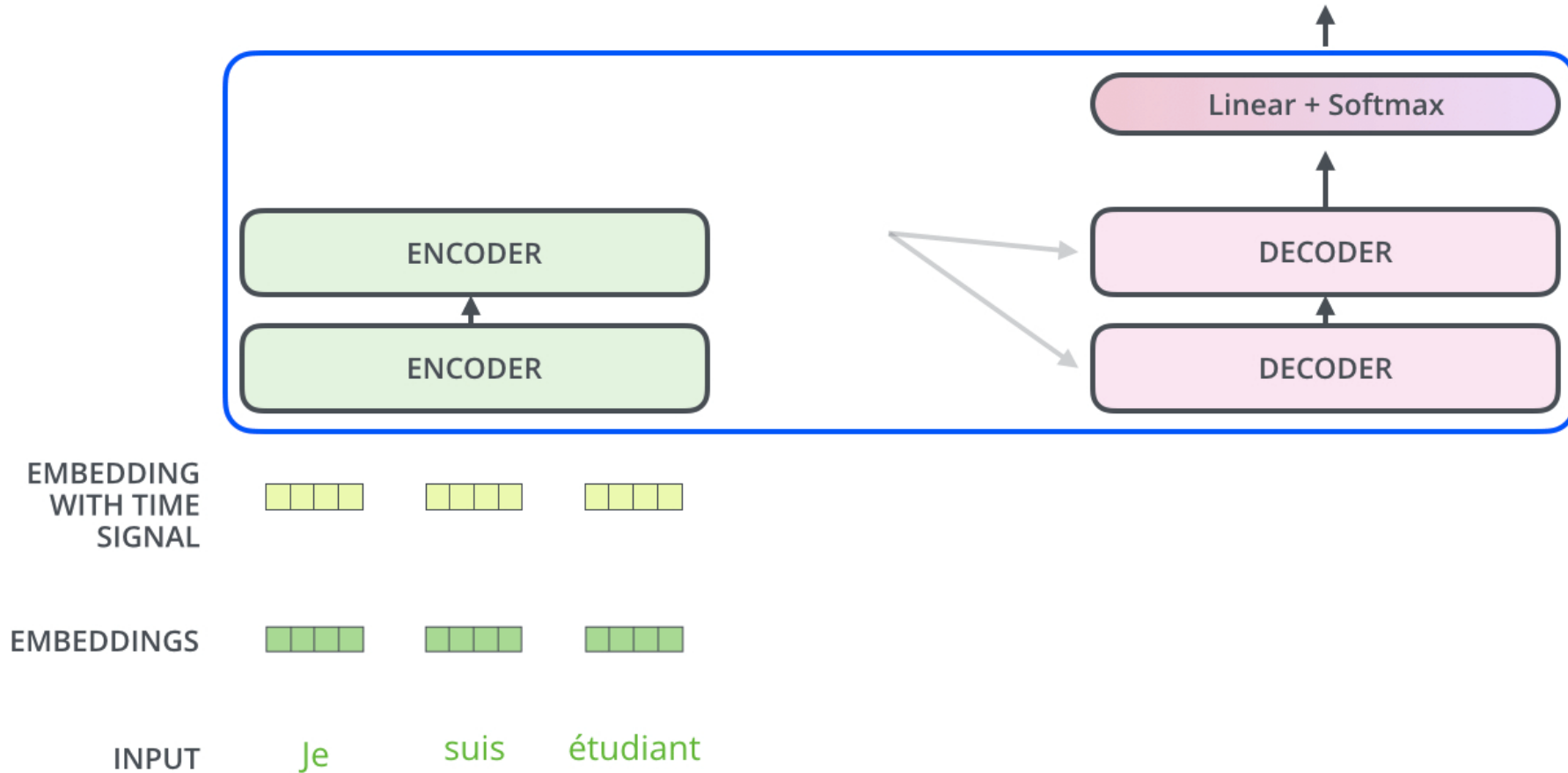
Self-Attention



Decoding time step: 1 2 3 4 5 6

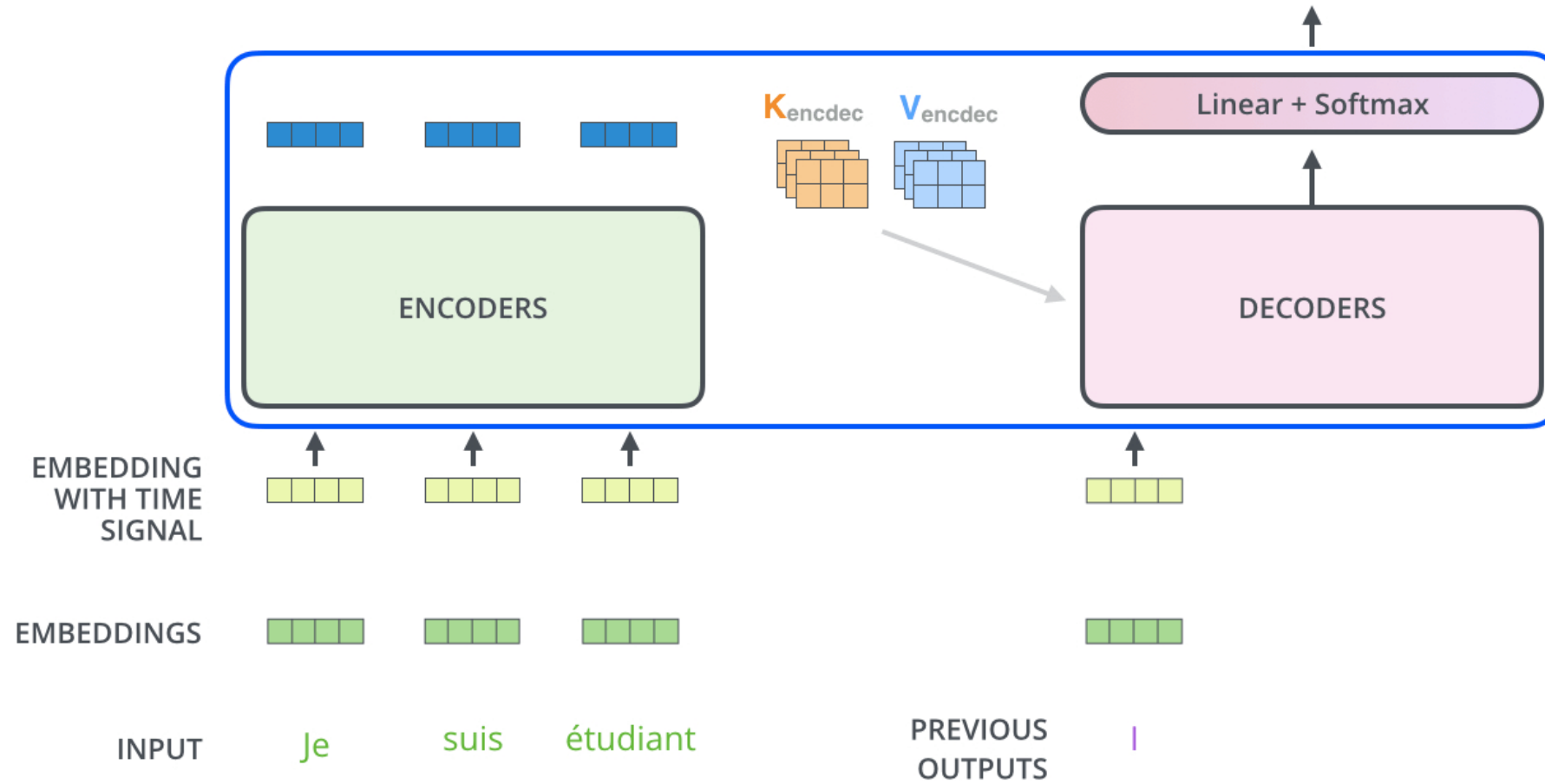
OUTPUT

After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).



Decoding time step: 1 2 3 4 5 6

OUTPUT |



## The Final Linear and Softmax Layer

Which word in our vocabulary is associated with this index?

Get the index of the cell with the highest value (**argmax**)

log\_probs



am

5

logits



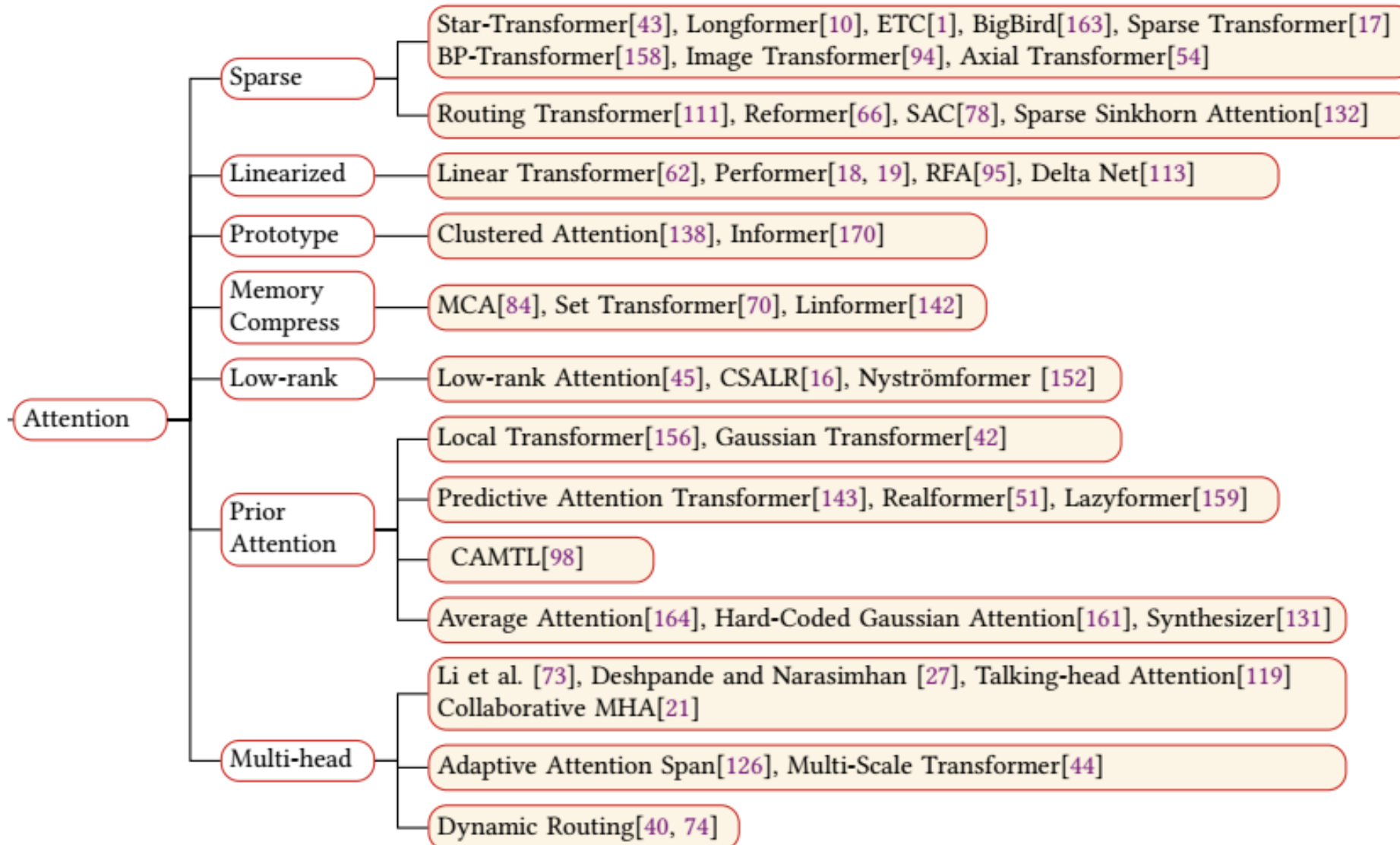
Softmax

Linear

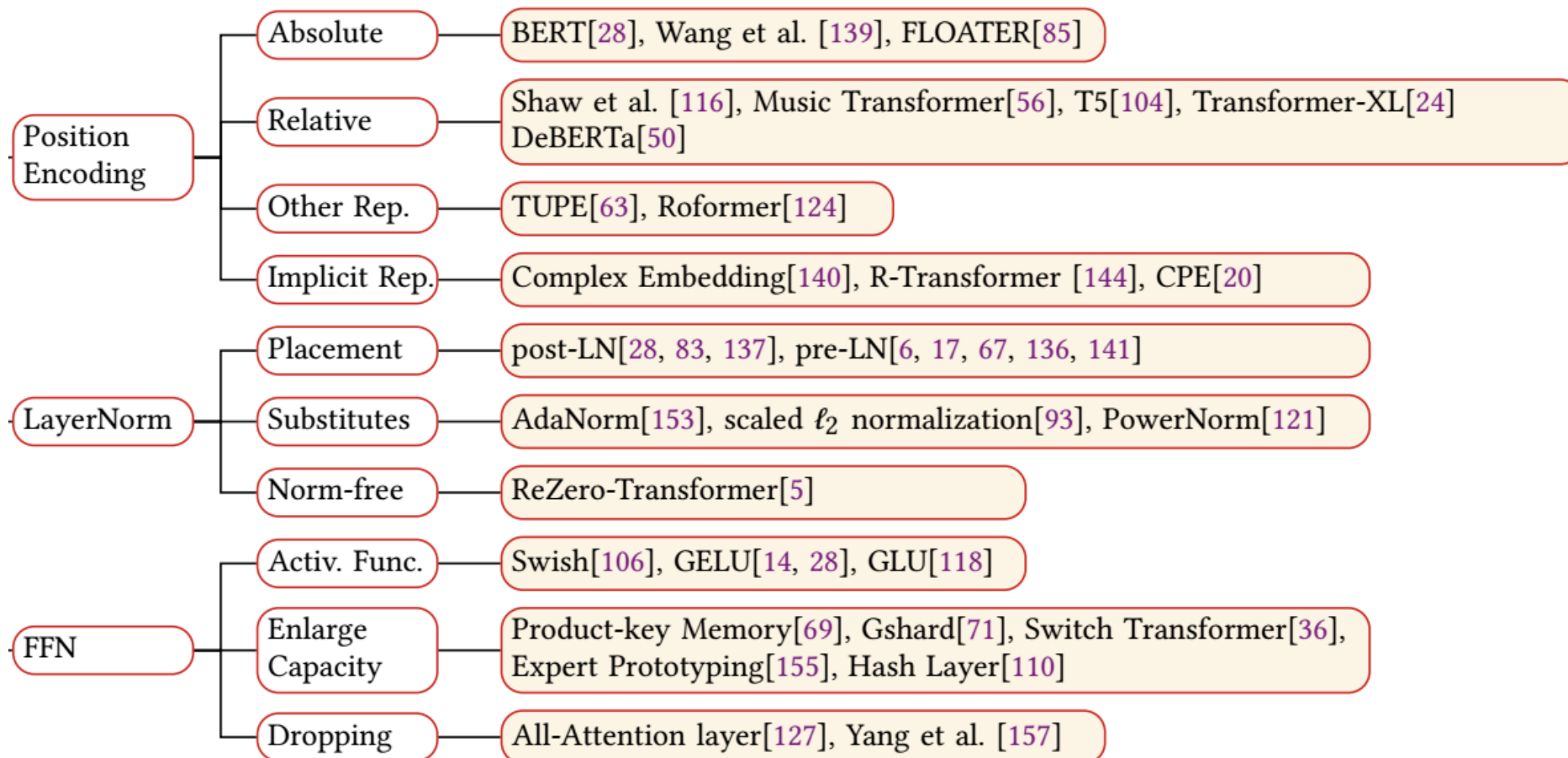
Decoder stack output



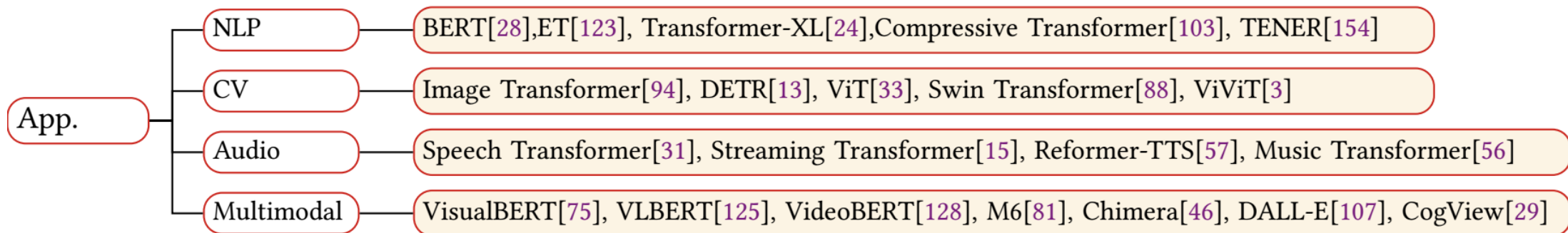
## Taxonomy of Transformers in Module Level (Attention)



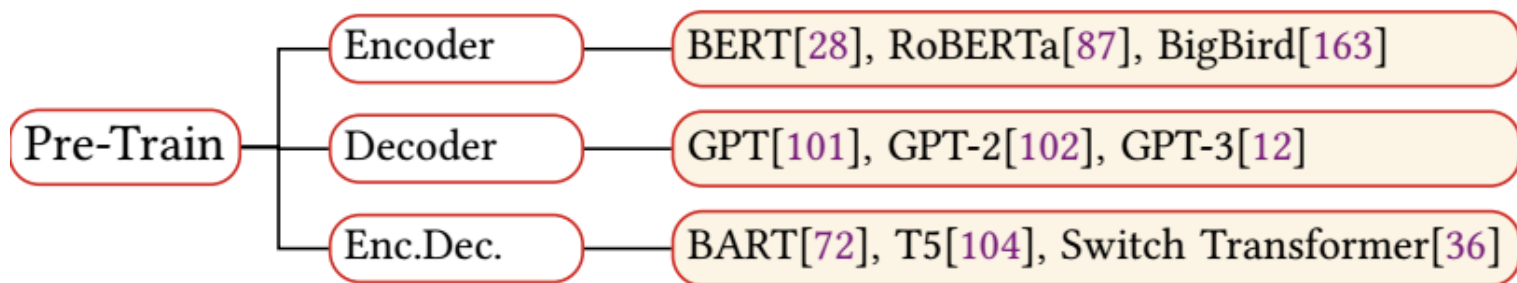
## Taxonomy of Transformers in Module Level (Position-Layer Norm-FFN)



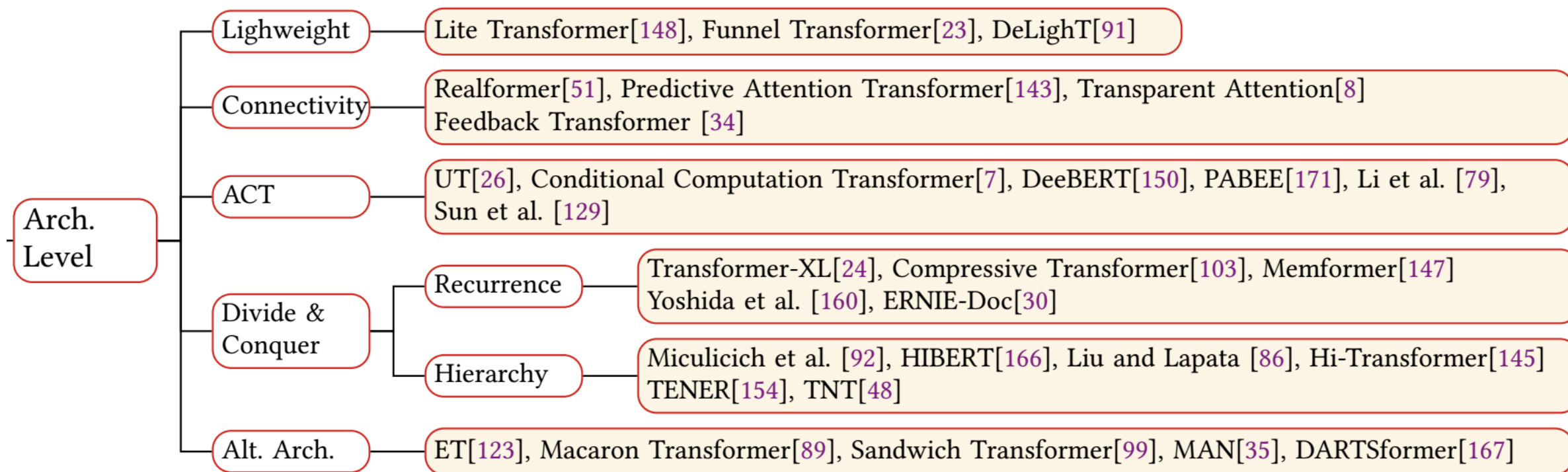
## Taxonomy of Transformers in App



## Taxonomy of Transformers in Pre-Train



# Taxonomy of Transformers in Arch. Level



ACT: Adaptive Computation Time

# Model Usage

Generally, the Transformer architecture can be used in three different ways:

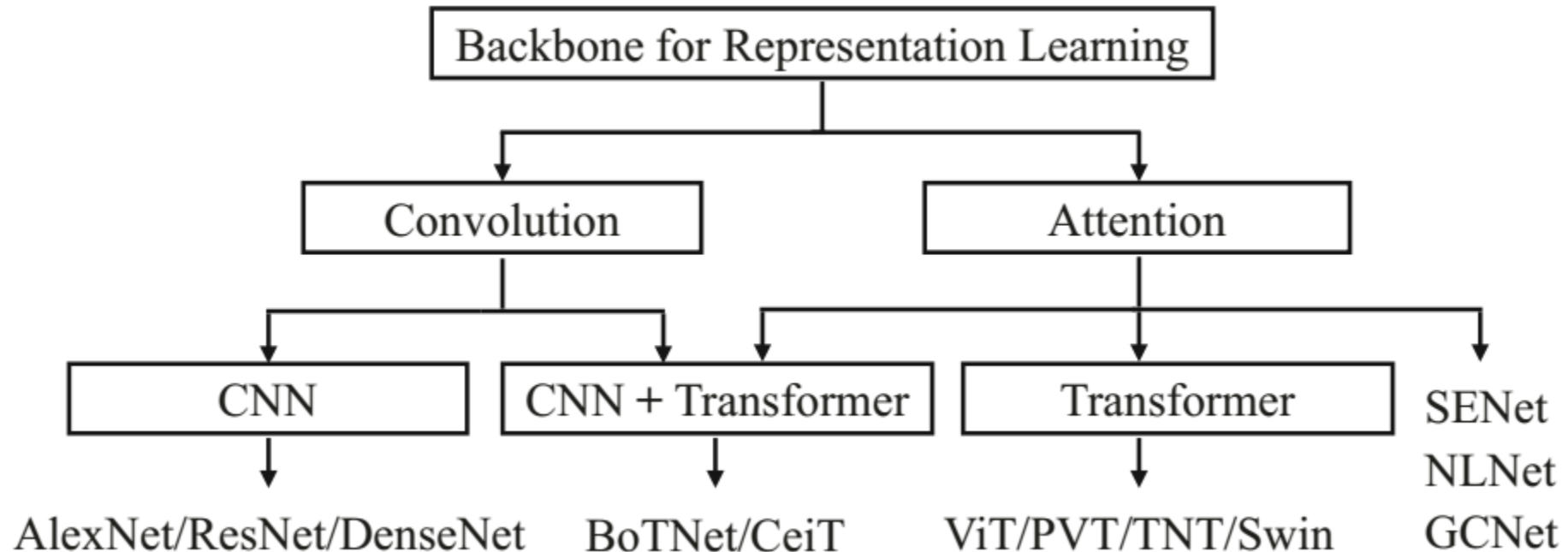
- *Encoder-Decoder*. The full Transformer architecture as introduced is used. This is typically used in sequence-to-sequence modeling (e.g., neural machine translation).
- *Encoder only*. Only the encoder is used and the outputs of the encoder are utilized as are presentation for the input sequence. This is usually used for classification or sequence labeling problems.
- *Decoder only*. Only the decoder is used, where the encoder-decoder cross-attention module is also removed. This is typically used for sequence generation, such as language modeling.



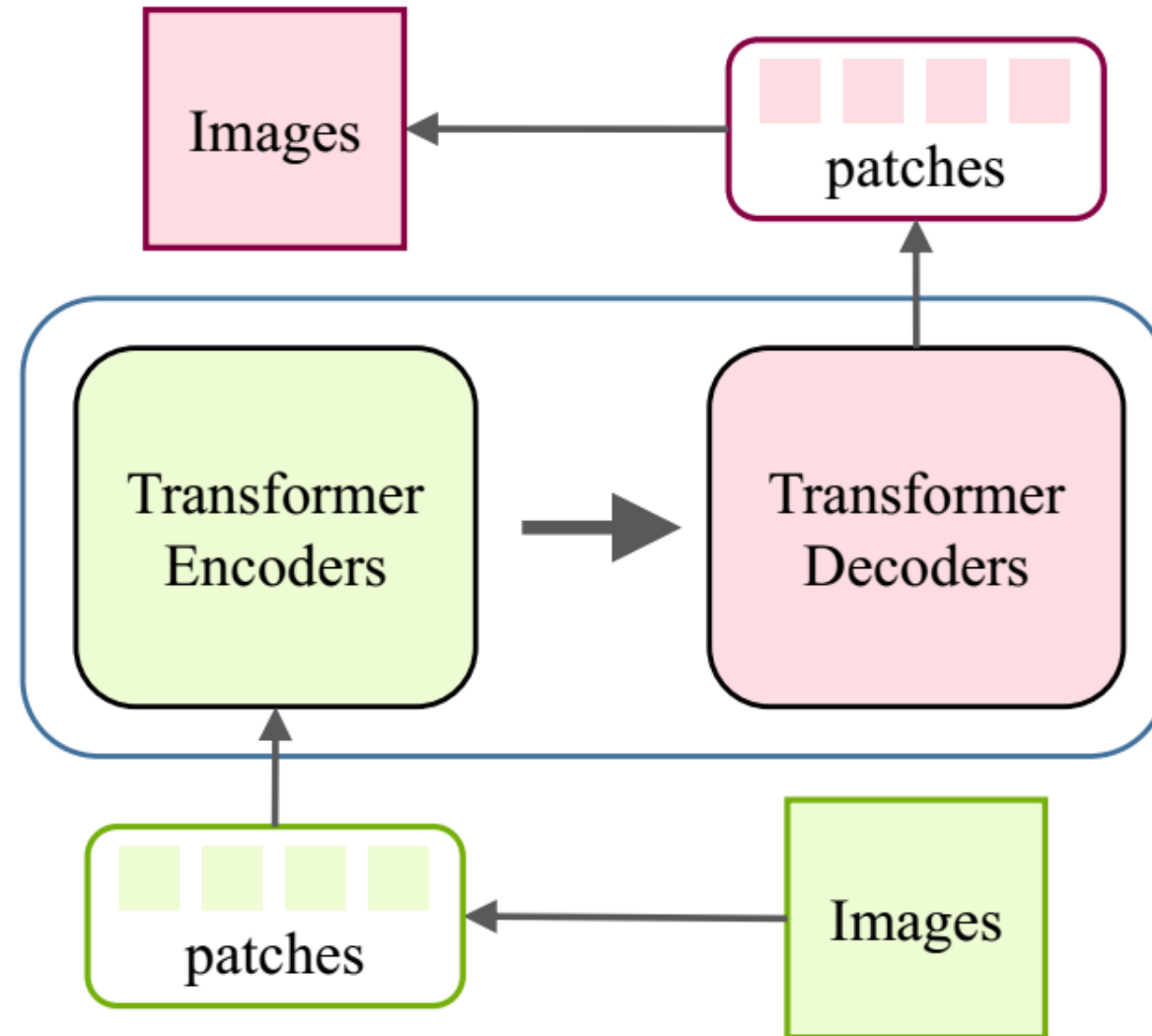
# Vision Transformer (ViT)

The applications of transformer based models in computer vision, including image classification, high/mid-level vision, low-level vision and video processing.

Convolution and Attention in Computer Vision Problems:



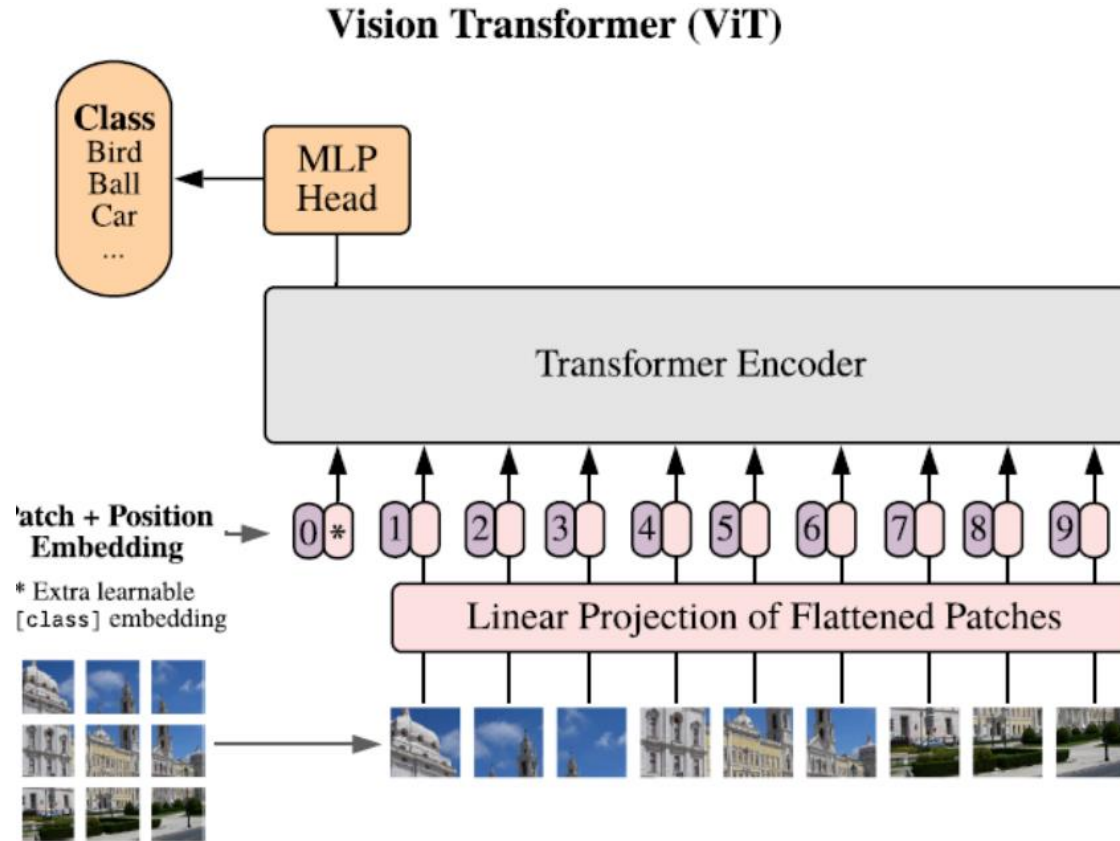
# A generic framework for using transformer in image processing



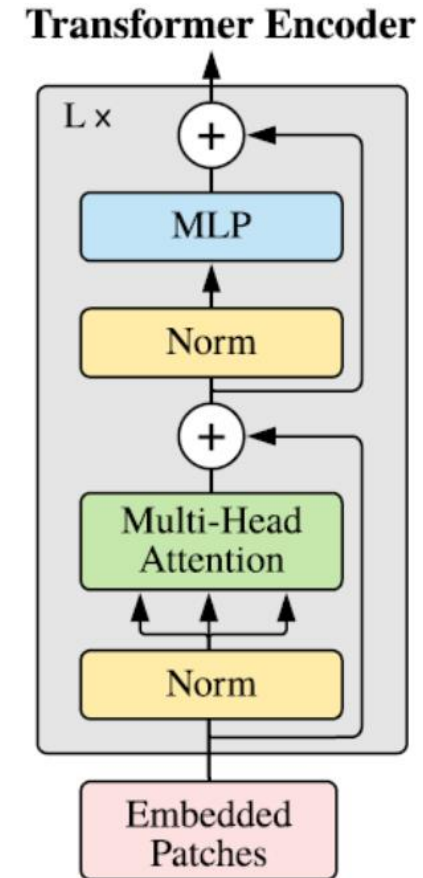
# Vision Transformer (ViT)

Dosovitskiy...2021

Vision Transformer (ViT) is a pure transformer directly applies to the sequences of image patches for image classification task. It follows transformer's original design as much as possible.



The framework of ViT



# How the Vision Transformer works

\* [Nikolas Adaloglou](#)

The total architecture is called Vision Transformer (ViT in short). Let's examine it step by step.

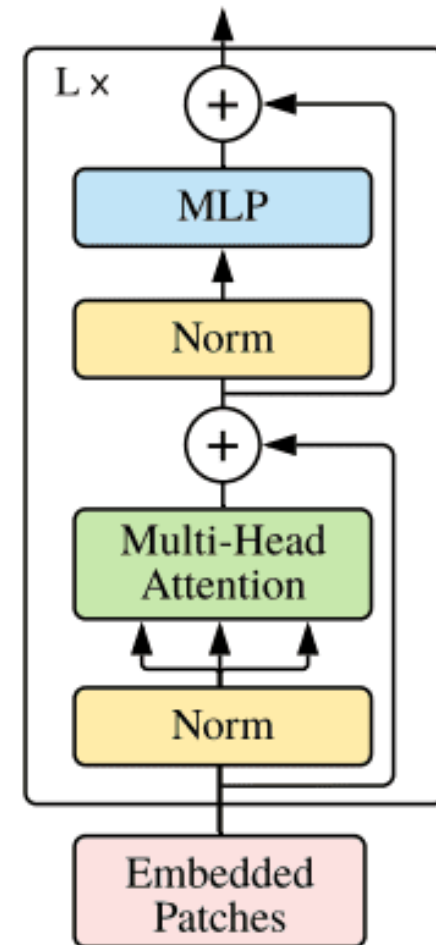
1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embedding from the flattened patches
4. Add positional embedding (they model positional embeddings with trainable linear layers)
5. Feed the sequence as an input to a standard transformer encoder
6. Pretrain the model with image labels (fully supervised on a huge dataset)
7. Fine tune on the downstream dataset for image classification

Image patches are basically the sequence tokens (like words).

In fact, the encoder block is identical to the original transformer proposed by Vaswani et al. (2017)

The only thing that changes is the number of those blocks.

**Transformer Encoder**



To this end, and to further prove that with more data they can train larger ViT variants, 3 models were proposed:

Model	Layers	Hidden size $D$	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Heads refer to [multi-head attention](#), while the MLP size refers to the blue module in the figure.

MLP stands for multi-layer perceptron but it's actually a bunch of linear transformation layers.

**Hidden size  $D$  is the embedding size**, which is kept fixed throughout the layers. Why keep it fixed? So that we can use short residual [skip connections](#).

**But is this enough?**

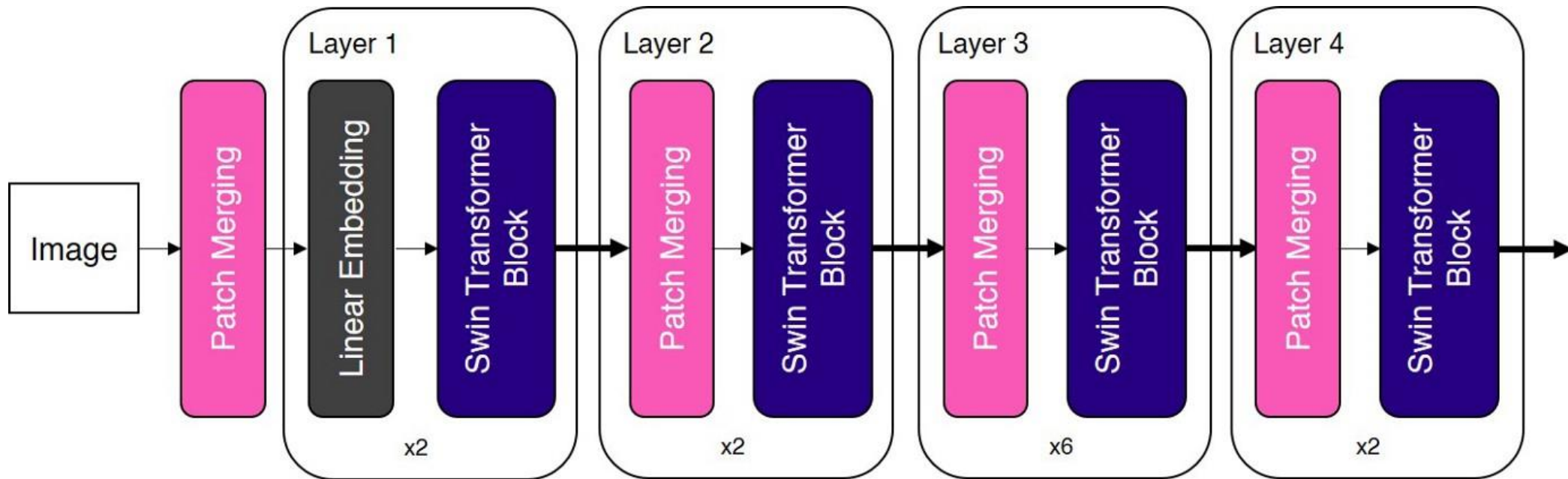
Yes and no. Actually, we need a massive amount of data and as a result computational resources.

# Swin Transformer

- Swin Transformer (Liu et al., 2021) is a **transformer-based deep learning model with state-of-the-art performance in vision tasks**. Unlike the Vision Transformer (ViT) (Dosovitskiy et al., 2020) which precedes it, Swin Transformer is highly efficient and has greater accuracy.
- In 2020, the Vision Transformer (ViT) garnered much attention from the AI community, notable for its pure transformer architecture with promising results in vision tasks. Despite its promise,
- ViTs suffer from several shortcomings:
  1. Most notably, ViTs struggle with high resolution images as its computational complexity is *quadratic* to the image size.
  2. Furthermore, the fixed scale tokens in ViTs are unsuitable in vision tasks where the visual elements are of variable scale.
- A flurry of research work followed ViT, and most of them made enhancements to the standard transformer architecture in order to address the above-mentioned shortcomings.
- In 2021, Microsoft researchers published the Swin Transformer ([Liu et al., 2021](#)), arguably one of the most exciting piece of research following up from the original ViT

# Swin Transformers

The Swin Transformer introduced two key concepts to address the issues faced by the original ViT — **hierarchical feature maps** and **shifted window attention**. In fact, the name of Swin Transformer comes from “**Shifted window Transformer**”. The overall architecture of the Swin Transformer is shown below.



As we can see, the ‘Patch Merging’ block and the ‘Swin Transformer Block’ are the two key building blocks in Swin Transformer.



*The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection.*

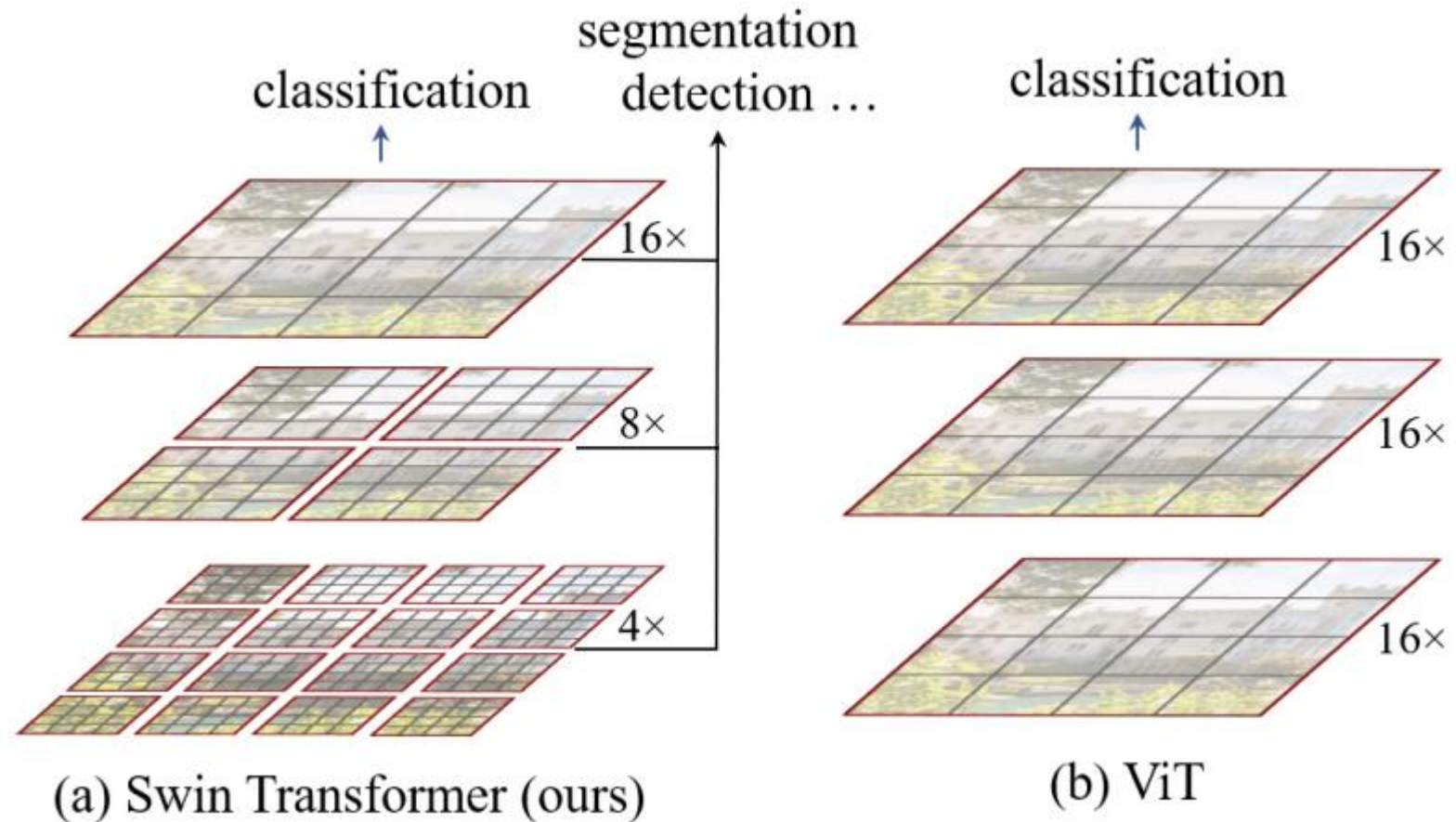


TABLE 1: Representative works of vision transformers.

Category	Sub-category	Method	Highlights	Publication
Backbone	Supervised pretraining	ViT [15] TNT [29] Swin [30]	Image patches, standard transformer Transformer in transformer, local attention Shifted window, window-based self-attention	ICLR 2021 NeurIPS 2021 ICCV 2021
	Self-supervised pretraining	iGPT [14] MoCo v3 [31]	Pixel prediction self-supervised learning, GPT model Contrastive self-supervised learning, ViT	ICML 2020 ICCV 2021
High/Mid-level vision	Object detection	DETR [16] Deformable DETR [17] UP-DETR [32]	Set-based prediction, bipartite matching, transformer DETR, deformable attention module Unsupervised pre-training, random query patch detection	ECCV 2020 ICLR 2021 CVPR 2021
		Max-DeepLab [25] VisTR [33] SETR [18]	PQ-style bipartite matching, dual-path transformer Instance sequence matching and segmentation Sequence-to-sequence prediction, standard transformer	CVPR 2021 CVPR 2021 CVPR 2021
	Pose Estimation	Hand-Transformer [34] HOT-Net [35] METRO [36]	Non-autoregressive transformer, 3D point set Structured-reference extractor Progressive dimensionality reduction	ECCV 2020 MM 2020 CVPR 2021
Low-level vision	Image generation	Image Transformer [27] Taming transformer [37] TransGAN [38]	Pixel generation using transformer VQ-GAN, auto-regressive transformer GAN using pure transformer architecture	ICML 2018 CVPR 2021 NeurIPS 2021
	Image enhancement	IPT [19] TTSR [39]	Multi-task, ImageNet pre-training, transformer model Texture transformer, RefSR	CVPR 2021 CVPR 2020
Video processing	Video inpainting	STTN [28]	Spatial-temporal adversarial loss	ECCV 2020
	Video captioning	Masked Transformer [20]	Masking network, event proposal	CVPR 2018
Multimodality	Classification	CLIP [40]	NLP supervision for images, zero-shot transfer	arXiv 2021
	Image generation	DALL-E [41] Cogview [42]	Zero-shot text-to image generation VQ-VAE, Chinese input	ICML 2021 NeurIPS 2021
	Multi-task	UniT [43]	Different NLP & CV tasks, shared model parameters	ICCV 2021
Efficient transformer	Decomposition	ASH [44]	Number of heads, importance estimation	NeurIPS 2019
	Distillation	TinyBert [45]	Various losses for different modules	EMNLP Findings 2020
	Quantization	FullyQT [46]	Fully quantized transformer	EMNLP Findings 2020
	Architecture design	ConvBert [47]	Local dependence, dynamic convolution	NeurIPS 2020

\*High-level vision deals with the interpretation and use of what is seen in the image (events)

\*Mid-level vision deals with how this information is organized as objects and their attributes (classification, detection, segmentation)

\*Low-level vision deals with how this information is organized as edges, blobs, and... (In image generation and enhancement)

# ImageNet result comparison of representative CNN and vision transformer models

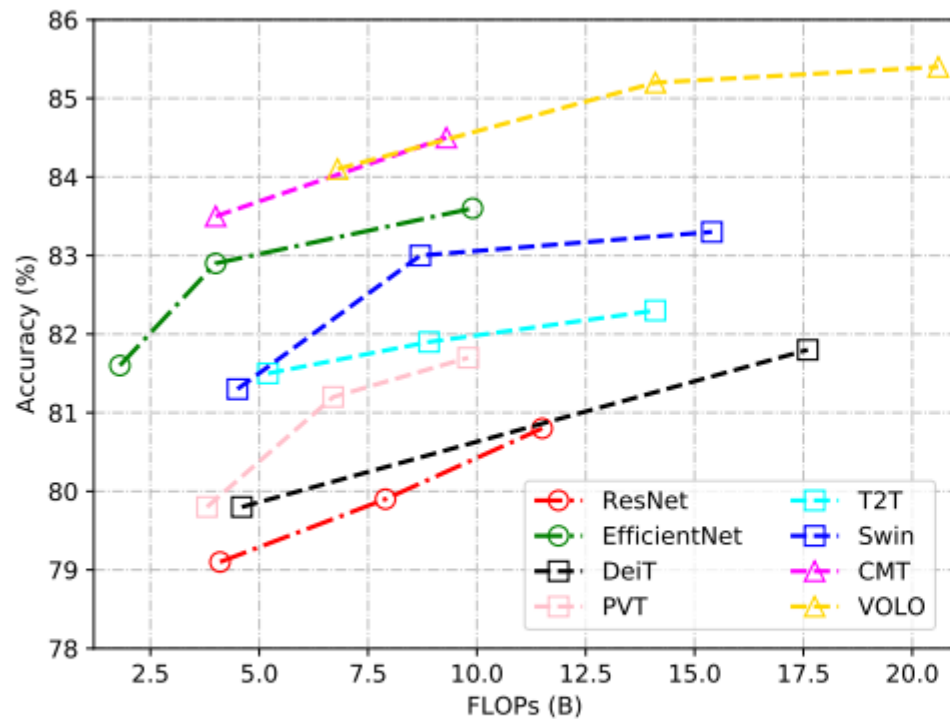
**Pure transformer** means only using a few convolutions in the stem stage.

**CNN + Transformer** means using convolutions in the intermediate layers.

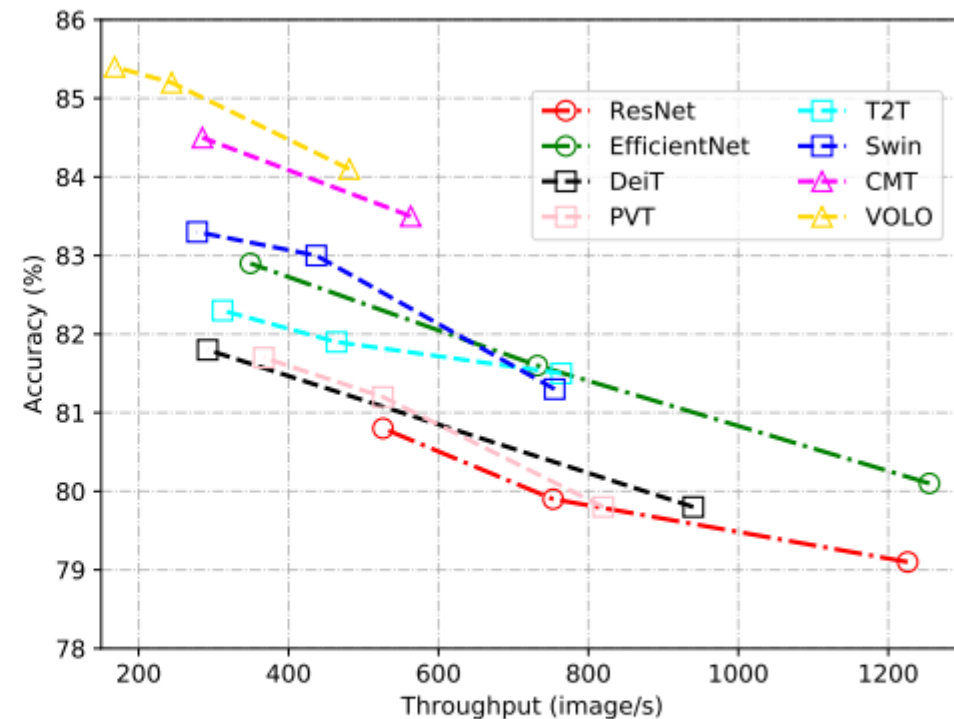
Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
<b>CNN</b>				
ResNet-50 [12], [67]	25.6	4.1	1226	79.1
ResNet-101 [12], [67]	44.7	7.9	753	79.9
ResNet-152 [12], [67]	60.2	11.5	526	80.8
EfficientNet-B0 [93]	5.3	0.39	2694	77.1
EfficientNet-B1 [93]	7.8	0.70	1662	79.1
EfficientNet-B2 [93]	9.2	1.0	1255	80.1
EfficientNet-B3 [93]	12	1.8	732	81.6
EfficientNet-B4 [93]	19	4.2	349	82.9

Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
<b>Pure Transformer</b>				
DeiT-Ti [15], [59]	5	1.3	2536	72.2
DeiT-S [15], [59]	22	4.6	940	79.8
DeiT-B [15], [59]	86	17.6	292	81.8
T2T-ViT-14 [67]	21.5	5.2	764	81.5
T2T-ViT-19 [67]	39.2	8.9	464	81.9
T2T-ViT-24 [67]	64.1	14.1	312	82.3
PVT-Small [72]	24.5	3.8	820	79.8
PVT-Medium [72]	44.2	6.7	526	81.2
PVT-Large [72]	61.4	9.8	367	81.7
TNT-S [29]	23.8	5.2	428	81.5
TNT-B [29]	65.6	14.1	246	82.9
CPVT-S [85]	23	4.6	930	80.5
CPVT-B [85]	88	17.6	285	82.3
Swin-T [60]	29	4.5	755	81.3
Swin-S [60]	50	8.7	437	83.0
Swin-B [60]	88	15.4	278	83.3
<b>CNN + Transformer</b>				
Twins-SVT-S [62]	24	2.9	1059	81.7
Twins-SVT-B [62]	56	8.6	469	83.2
Twins-SVT-L [62]	99.2	15.1	288	83.7
Shuffle-T [65]	29	4.6	791	82.5
Shuffle-S [65]	50	8.9	450	83.5
Shuffle-B [65]	88	15.6	279	84.0
CMT-S [94]	25.1	4.0	563	83.5
CMT-B [94]	45.7	9.3	285	84.5
VOLO-D1 [95]	27	6.8	481	84.2
VOLO-D2 [95]	59	14.1	244	85.2
VOLO-D3 [95]	86	20.6	168	85.4
VOLO-D4 [95]	193	43.8	100	85.7
VOLO-D5 [95]	296	69.0	64	86.1

# FLOPs and throughput comparison of representative CNN and vision transformer models (ImageNet Top-1 (%)).



(a) Acc v.s. FLOPs.














(b) Acc v.s. throughput.

**Throughput** is how much information actually gets delivered in a certain amount of time.

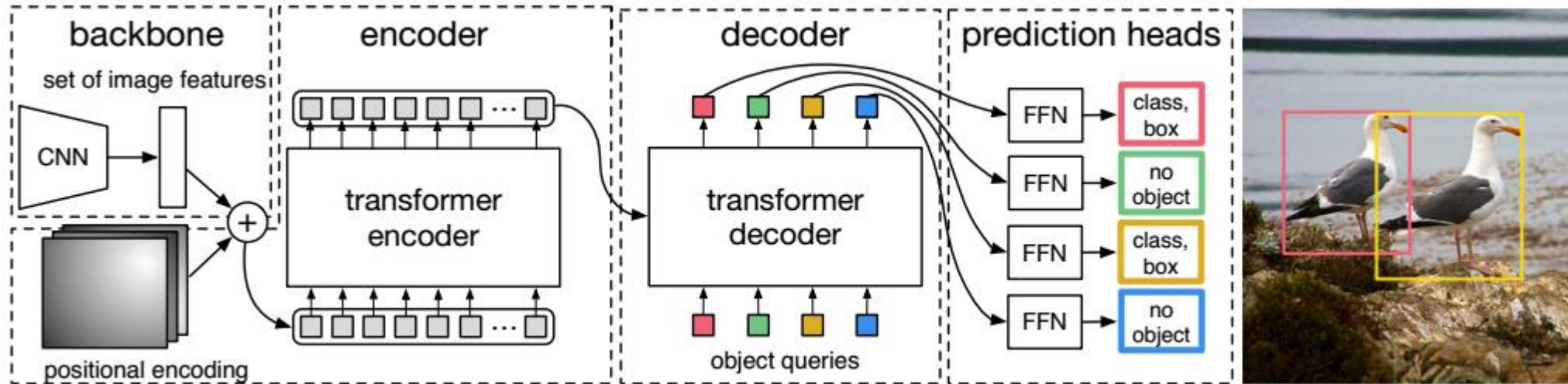
**FLOP**: Floating point operations per second



# ImageNet Benchmark (Image Classification) | Papers With Code

Rank	Model	Top 1 Accuracy	↑ Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags 
1	<b>Model soups</b> (ViT-G/14)	90.94%		1843M	✓	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	<a href="#">Transformer</a> <a href="#">JFT-3B</a>
2	<b>CoAtNet-7</b>	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	<a href="#">Conv+Transformer</a> <a href="#">JFT-3B</a>
3	<b>ViT-G/14</b>	90.45%		1843M	✓	Scaling Vision Transformers			2021	<a href="#">Transformer</a> <a href="#">JFT-3B</a>
4	<b>CoAtNet-6</b>	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	<a href="#">Conv+Transformer</a> <a href="#">JFT-3B</a>
5	<b>V-MoE-15B</b> (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	<a href="#">Transformer</a>
6	<b>Meta Pseudo Labels</b> (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels			2021	<a href="#">EfficientNet</a> <a href="#">JFT-300M</a>

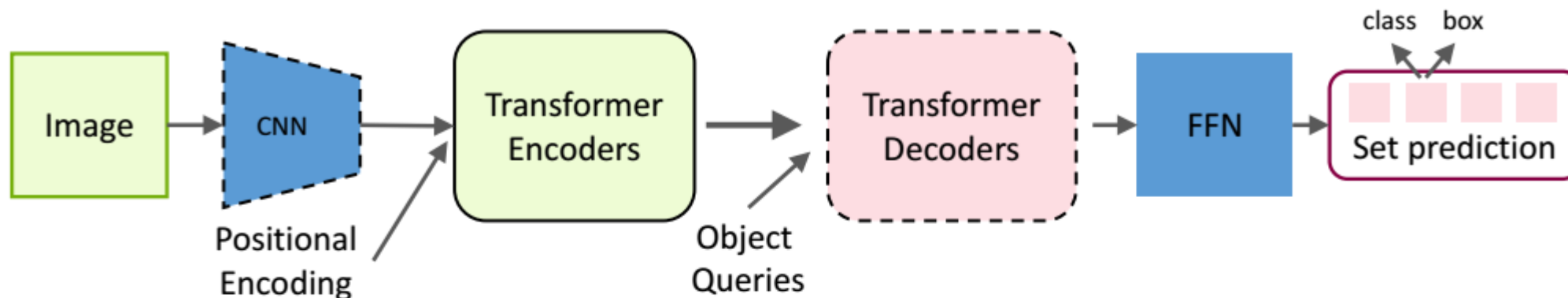
# Transformers in Detection



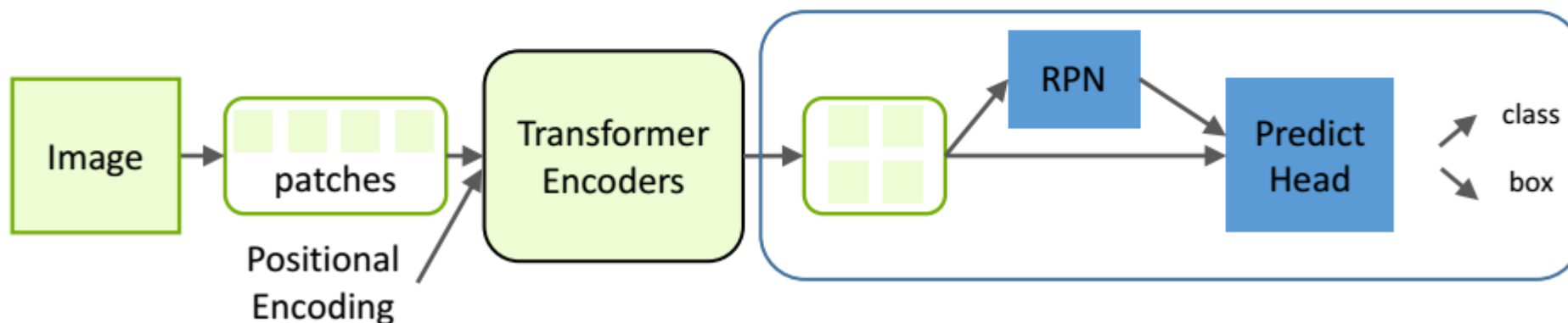
## The overall architecture of DETR

\*N. Carion et al. End-to-end object detection with transformers. In *ECCV*, 2020

# General framework of transformer-based object detection



(a) Transformer-based set prediction for detection



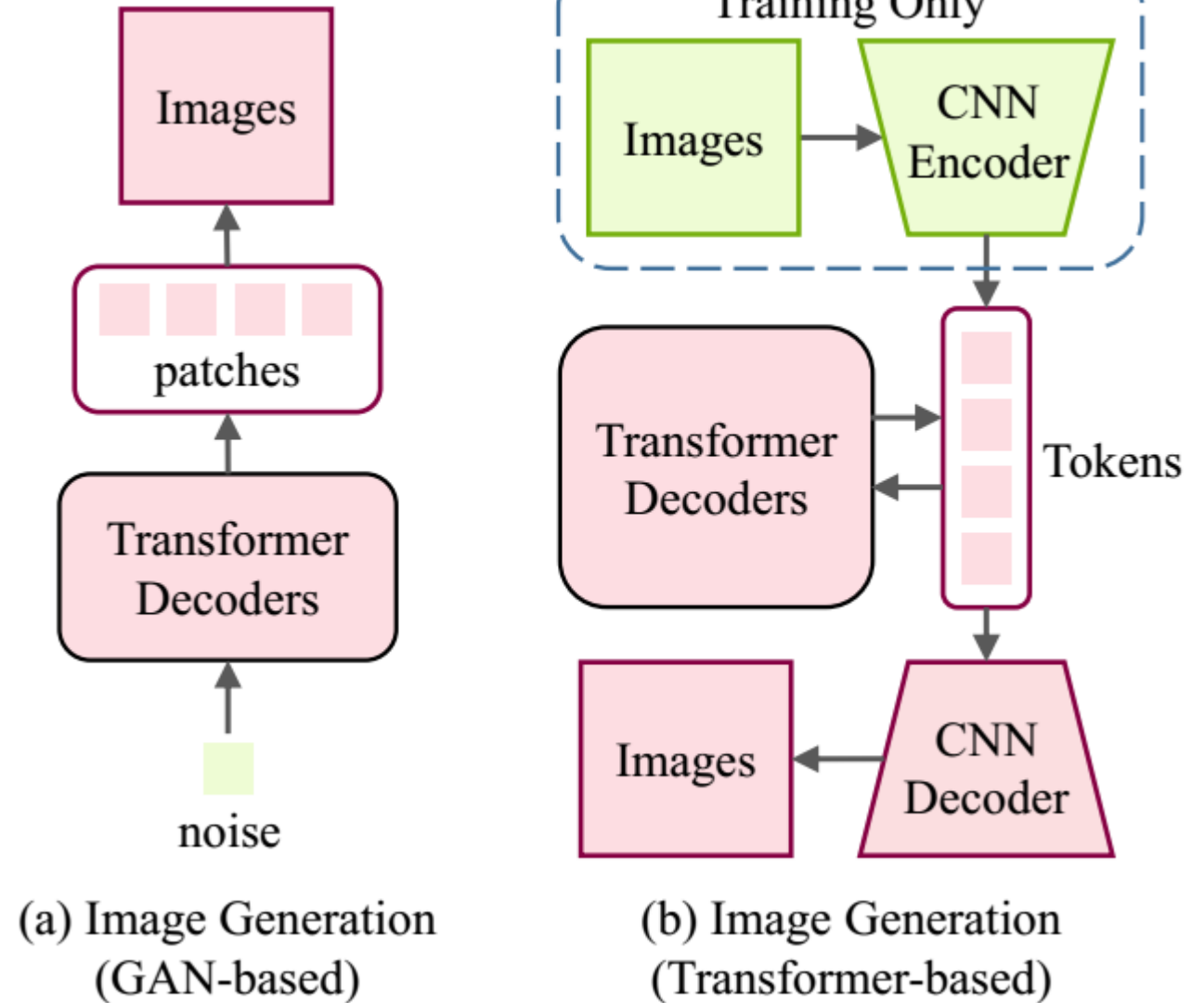
(b) Transformer-based backbone for detection

TABLE 3: Comparison of different transformer-based object detectors on COCO 2017 val set. Running speed (FPS) is evaluated on an NVIDIA Tesla V100 GPU as reported in [17]. <sup>†</sup>Estimated speed according to the reported number in the paper. <sup>‡</sup>ViT backbone is pre-trained on ImageNet-21k. \*ViT backbone is pre-trained on an private dataset with 1.3 billion images.

Method	Epochs	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>	#Params (M)	GFLOPs	FPS
<i>CNN based</i>										
FCOS [125]	36	41.0	59.8	44.1	26.2	44.6	52.2	-	177	23 <sup>†</sup>
Faster R-CNN + FPN [13]	109	42.0	62.1	45.5	26.6	45.4	53.4	42	180	26
<i>CNN Backbone + Transformer Head</i>										
DETR [16]	500	42.0	62.4	44.2	20.5	45.8	61.1	41	86	28
DETR-DC5 [16]	500	43.3	63.1	45.9	22.5	47.3	61.1	41	187	12
Deformable DETR [17]	50	46.2	65.2	50.0	28.8	49.2	61.7	40	173	19
TSP-FCOS [120]	36	43.1	62.3	47.0	26.6	46.8	55.9	-	189	20 <sup>†</sup>
TSP-RCNN [120]	96	45.0	64.5	49.6	29.7	47.7	58.0	-	188	15 <sup>†</sup>
ACT+MKKD (L=32) [121]	-	43.1	-	-	61.4	47.1	22.2	-	169	14 <sup>†</sup>
SMCA [123]	108	45.6	65.5	49.1	25.9	49.3	62.6	-	-	-
Efficient DETR [124]	36	45.1	63.1	49.1	28.3	48.4	59.0	35	210	-
UP-DETR [32]	150	40.5	60.8	42.6	19.0	44.4	60.0	41	-	-
UP-DETR [32]	300	42.8	63.0	45.3	20.8	47.1	61.7	41	-	-
<i>Transformer Backbone + CNN Head</i>										
ViT-B/16-FRCNN <sup>‡</sup> [113]	21	36.6	56.3	39.3	17.4	40.0	55.5	-	-	-
ViT-B/16-FRCNN* [113]	21	37.8	57.4	40.1	17.8	41.4	57.3	-	-	-
PVT-Small+RetinaNet [72]	12	40.4	61.3	43.0	25.0	42.9	55.7	34.2	118	-
Twins-SVT-S+RetinaNet [62]	12	43.0	64.2	46.3	28.0	46.4	57.5	34.3	104	-
Swin-T+RetinaNet [60]	12	41.5	62.1	44.2	25.1	44.9	55.5	38.5	118	-
Swin-T+ATSS [60]	36	47.2	66.5	51.3	-	-	-	36	215	-
<i>Pure Transformer based</i>										
PVT-Small+DETR [72]	50	34.7	55.7	35.4	12.0	36.4	56.7	40	-	-
TNT-S+DETR [29]	50	38.2	58.9	39.4	15.5	41.1	58.8	39	-	-
YOLOS-Ti [126]	300	30.0	-	-	-	-	-	6.5	21	-
YOLOS-S [126]	150	37.6	57.6	39.2	15.9	40.2	57.3	28	179	-
YOLOS-B [126]	150	42.0	62.2	44.5	19.5	45.3	62.1	127	537	-



# Transformers in Image Generation



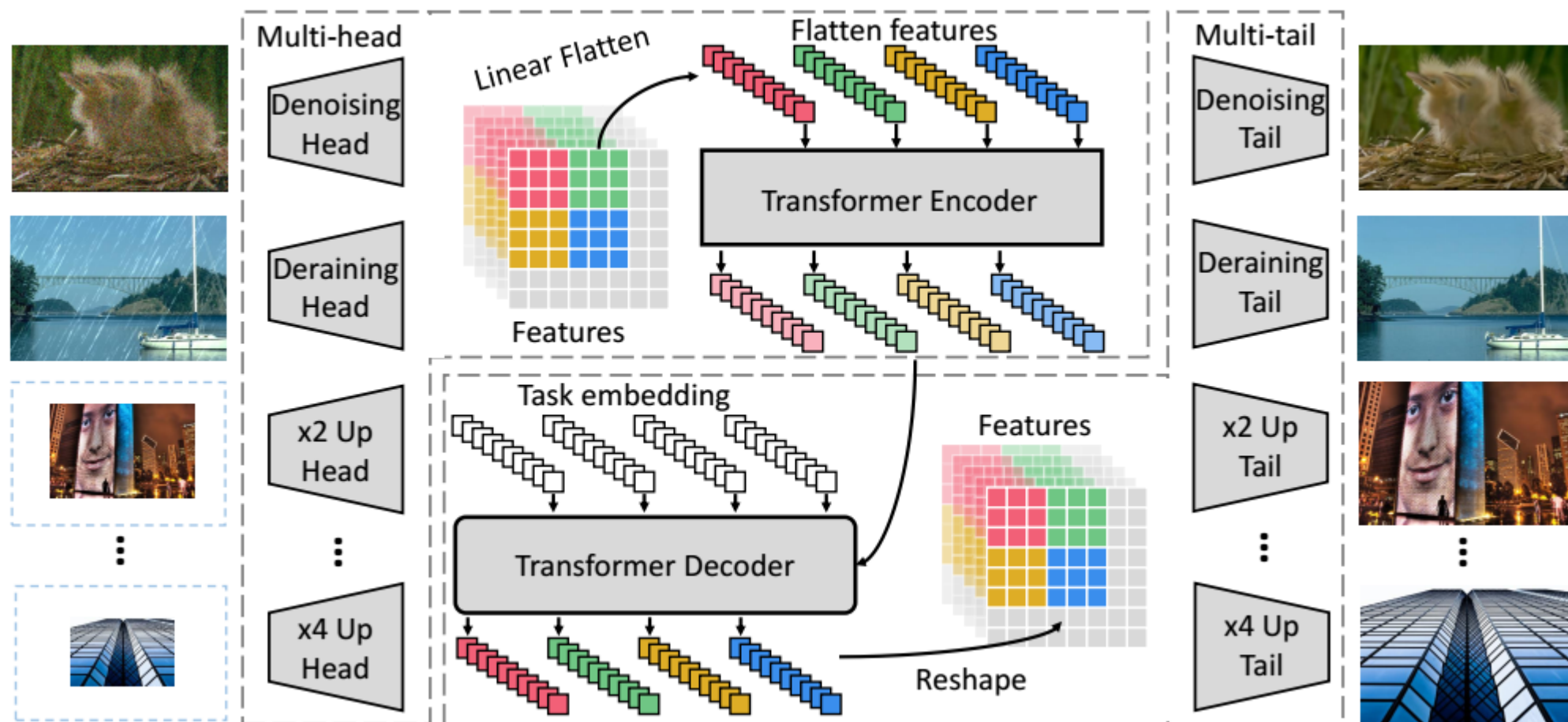
A generic framework for transformer in image generation

# Transformers in Image Enhancement

**IPT**

Image Processing Transformer

[19] H. Chen et al. Pre-trained image processing transformer. In *CVPR*, 2021

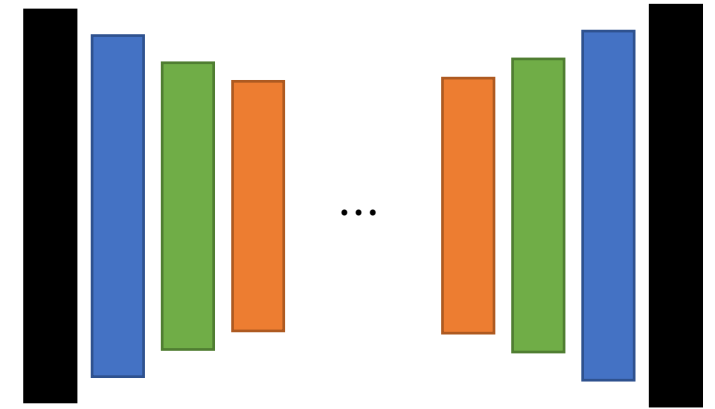


IPT fully utilizes the advantages of transformers by using large pre-training datasets. It achieves state-of-the-art performance in several image processing tasks, including super-resolution, denoising, and deraining.

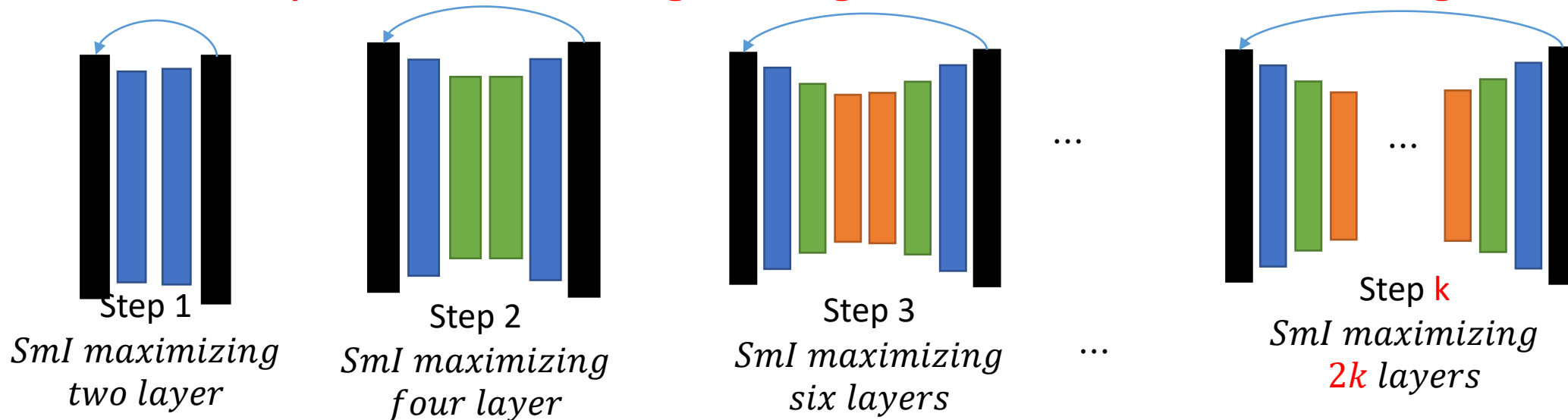
### 3.3.8 Layer-wise Forward Learning Part 3

# Layer Wise Forward Learning of an Auto-Encoder

- Smoothness index can be used to train an auto-encoder step by step in a forward manner.
- At the first step two hidden layers are trained and then by each further step two new layers are added and trained to the AE.
- At each step, the training procedure is based on “Sml” maximizing.
- In fact, the smoothness index is approximated by a triplet loss and then the loss is minimized.
- Each example at the output layer decreases(increases) its distance to one of its K nearest neighbors that has nearest (furthest) distance at the target(input) space (reconstruction)



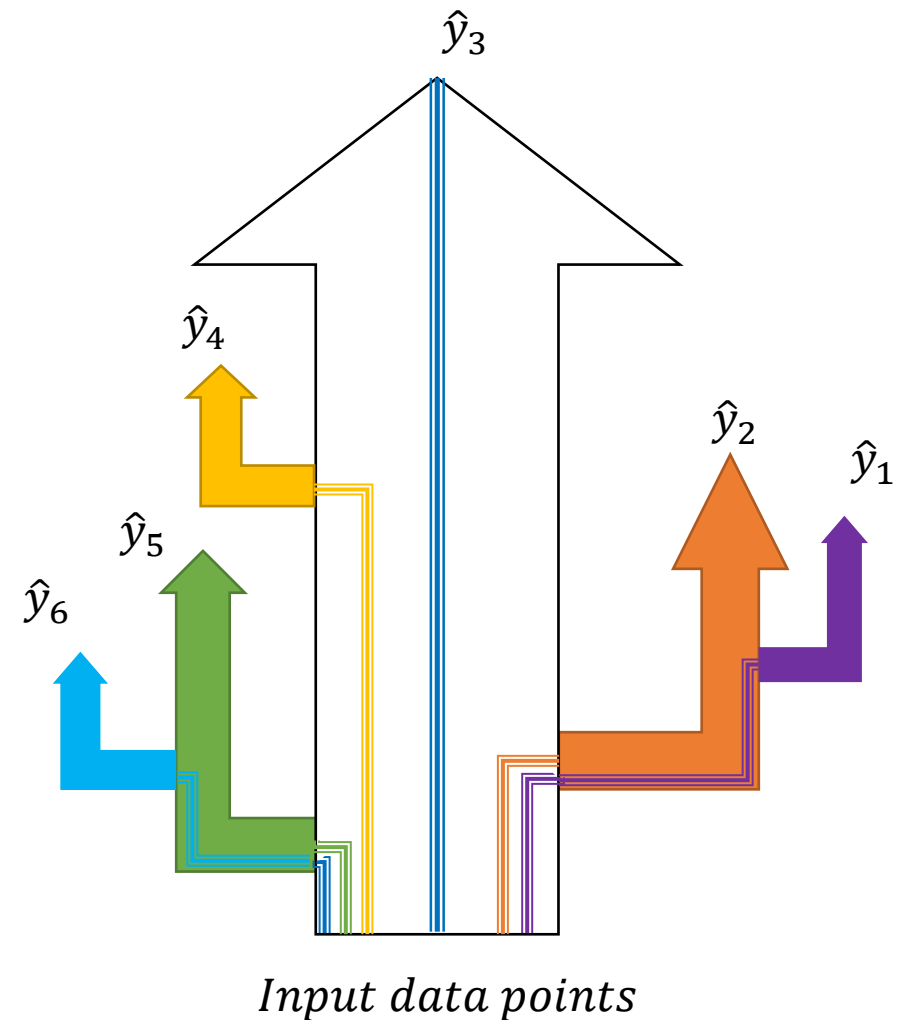
## Layer-wise Learning through Smoothness Maximizing



### 3.3.4 Layer-wise branching

# Layer-wise branching

- In a classification (regression) problem, the required features may be extracted earlier or later through layers of a DNN.
- If we design a new architecture of DNN which can conduct earlier features in some independent paths, more compact and more generalized DNNs can be learned.
- In such an architecture, there are a main body and some branches. The main body forms the main flow of data but each branch process a part of data-flow.
- We can use complexity measures like SI(Sml) to distinguish early features from other features and form a new branch for it.
- Each branch like the main body can define some sub-branches, too.
- In such a DNN, the amount of data-flow decreases through the main body just after each branching.



## Some notes

- Branch in point where an increasing jump on SI(Sml) is occurred.
- Find the appropriate part of data (class/output/examples) to be branched.
- Branching is done as a forward design operation

# Utilizing SI for Branching

Image-based and Partially Categorical Annotating  
Approach for Pedestrian Attribute Recognition

Hossein Bodaghi

DukeMTMC-reID: 34183 images

pretrained network:

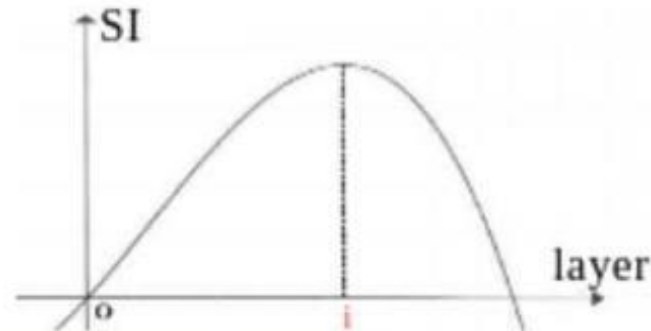
person re identification

ID,

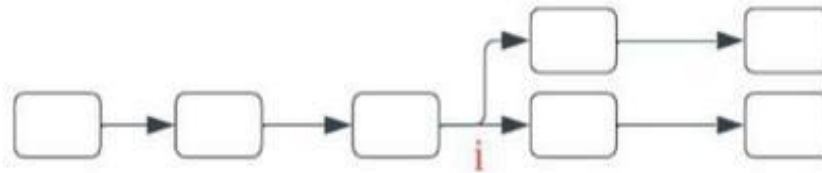


Task1: person reID

1. Check the information flow through  
network for attribute recognition



2. generating branches



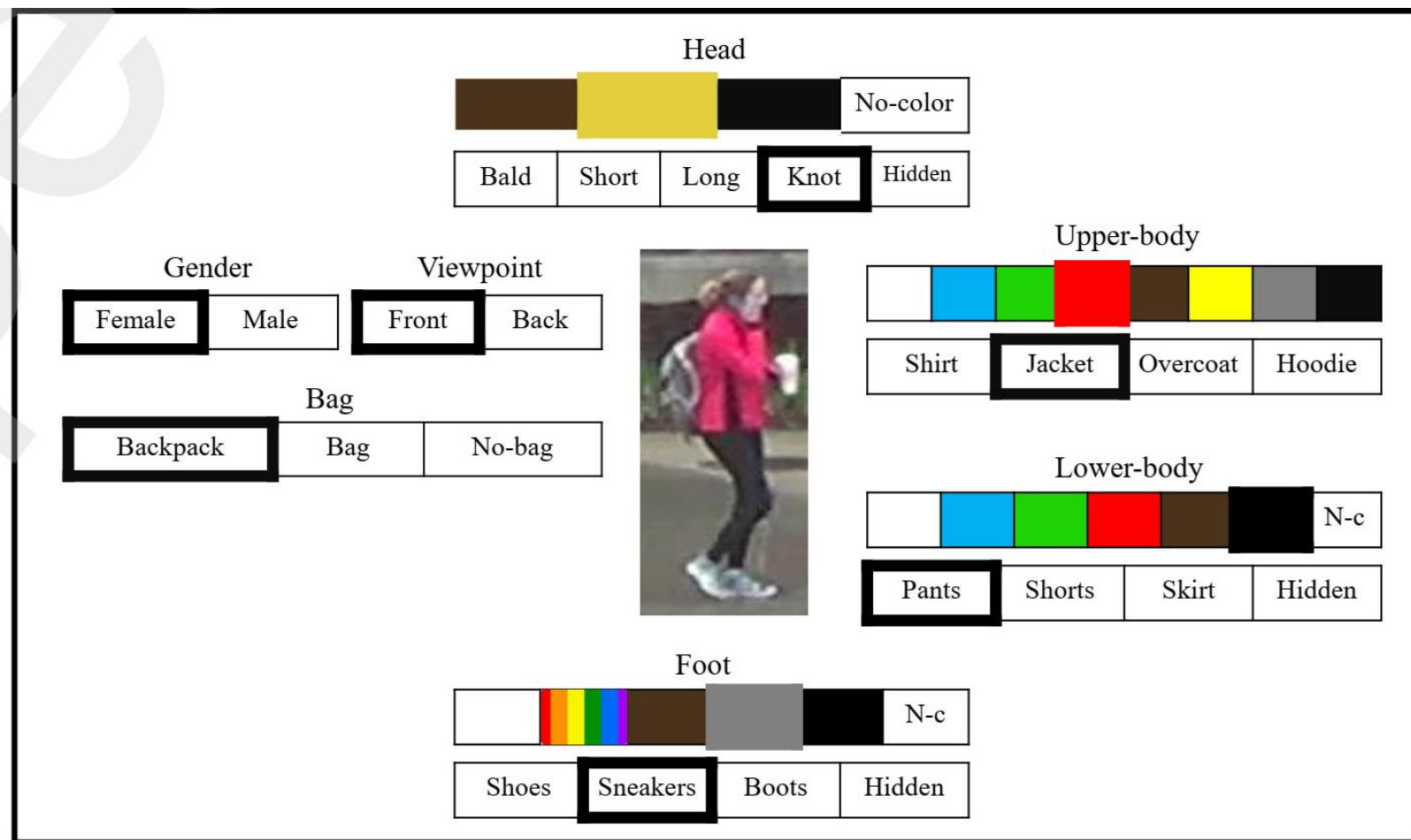
3. Train the branch for  
attribute recognition

attributes,



Task2: attribute recognition

In order to examine the impact of data on the PAR result, this research suggests an image-based partially categorical attribute dataset (CA-Duke) including 36,411 images of the DukeMTMC-reID dataset for 74 pedestrian attributes.

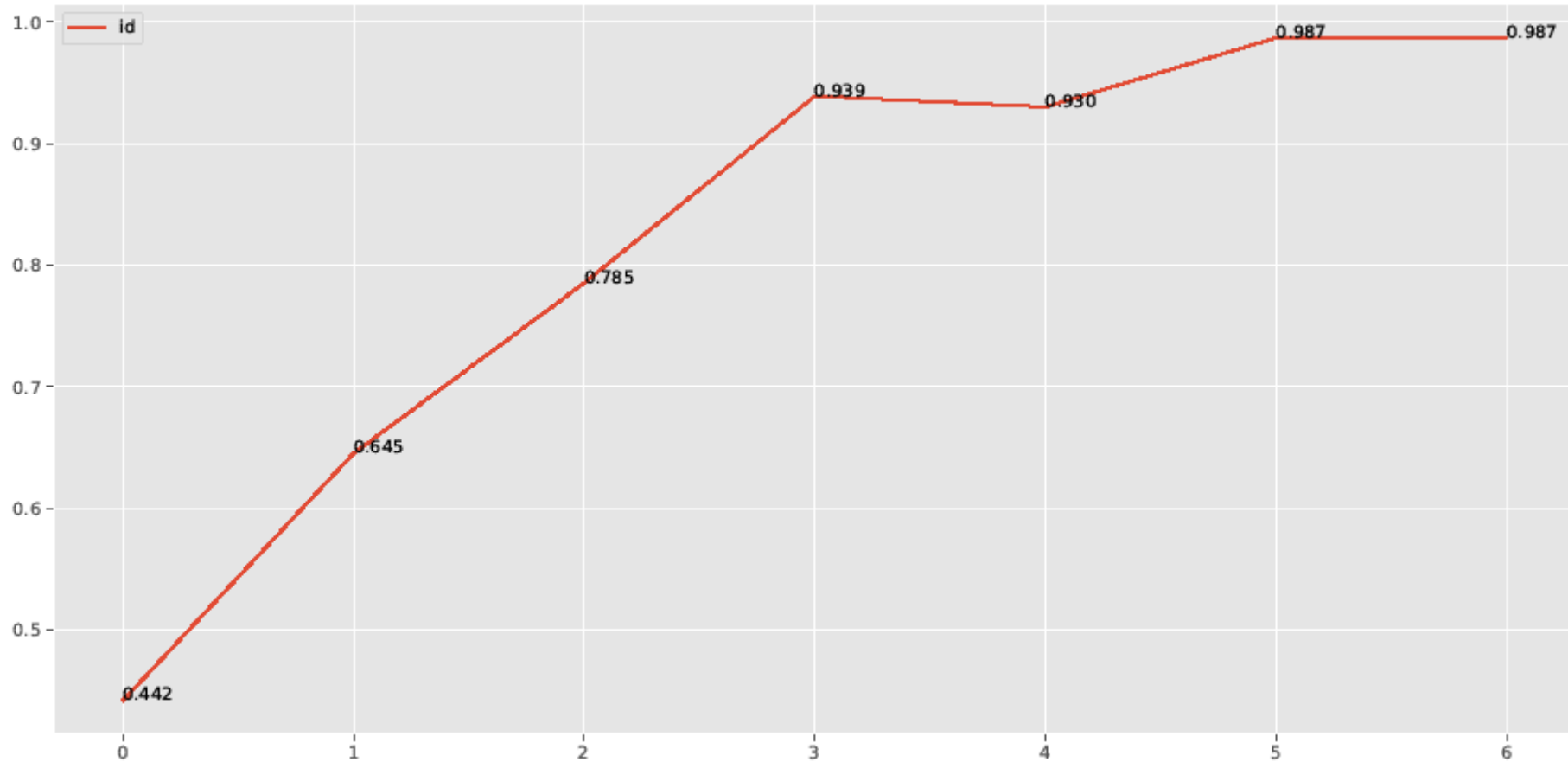


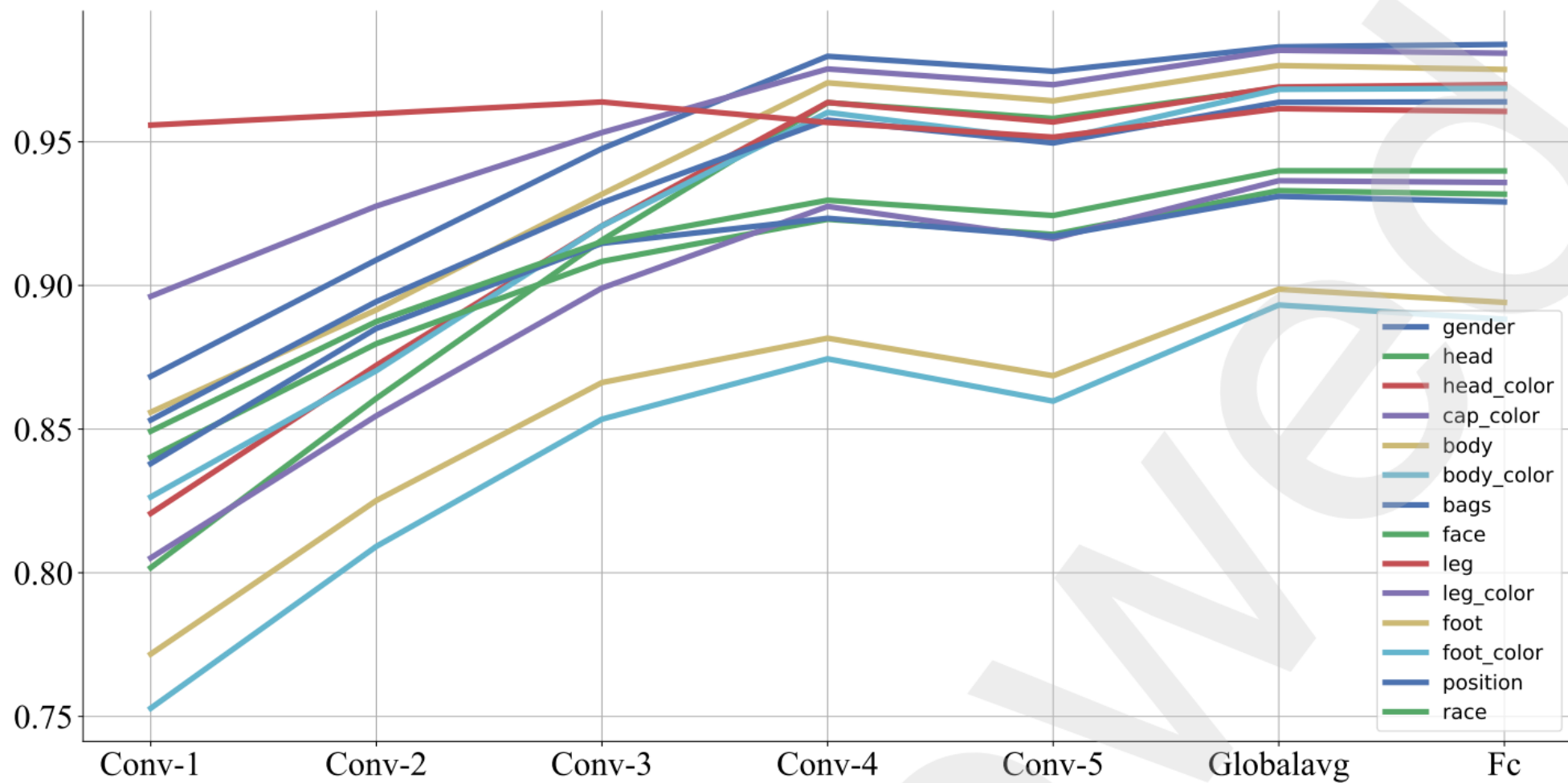
**Figure 1:** Partial attribute annotating for proposed dataset that includes the categories of each part and their colors.



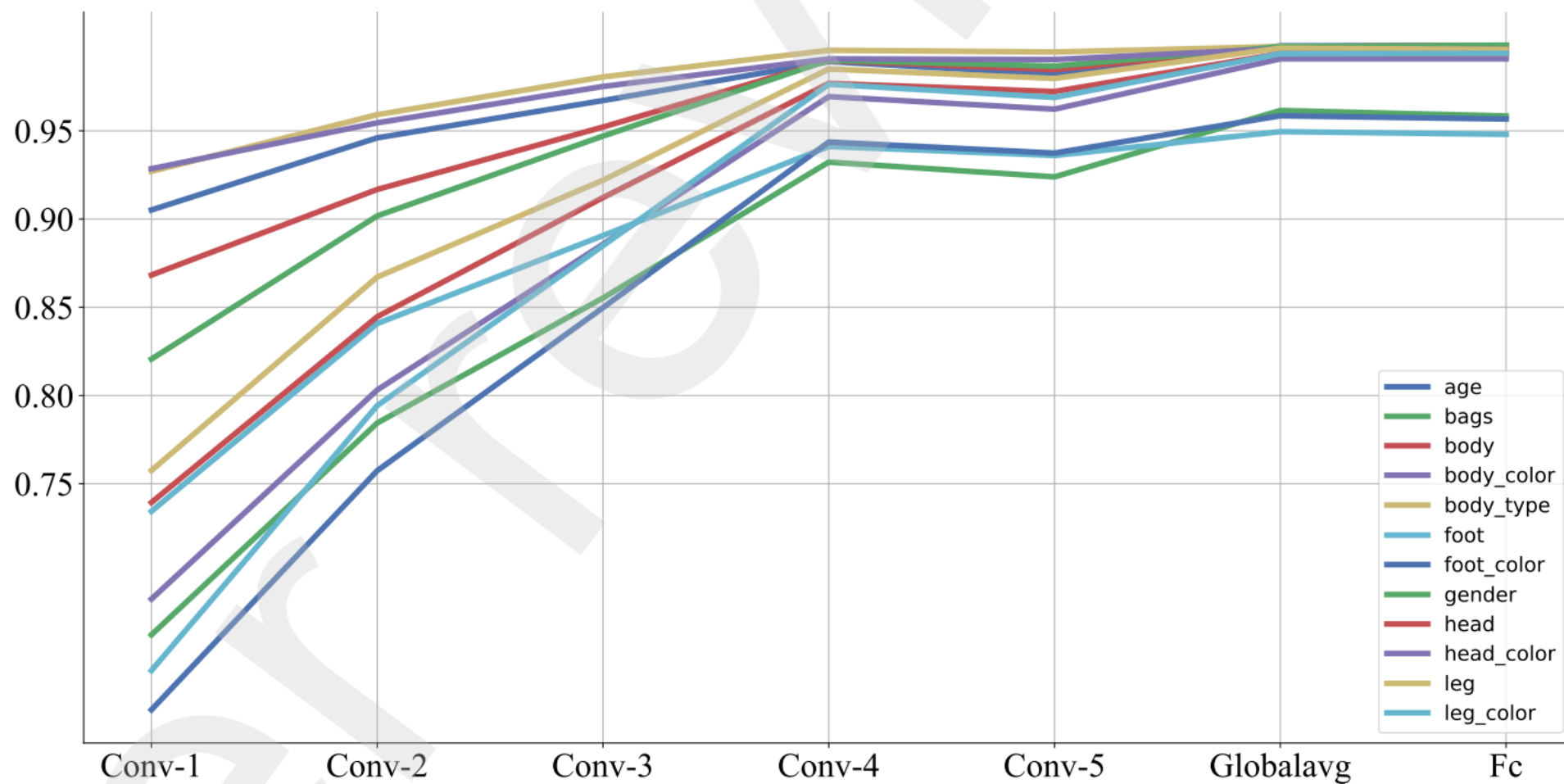
# Separation Index for person Reid

(a classification problem)- OS-Net (Omni-Scale Feature Learning Network)

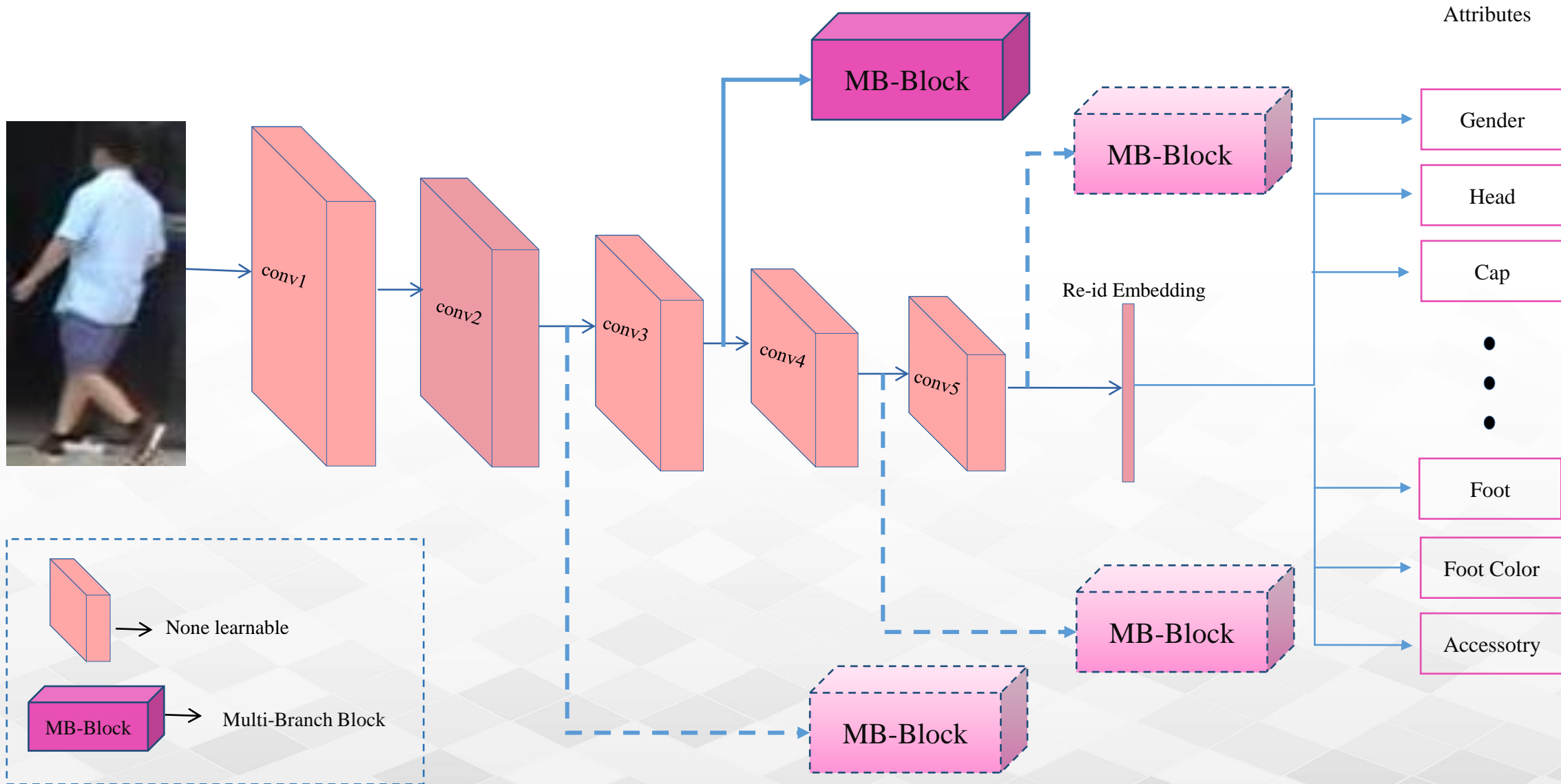




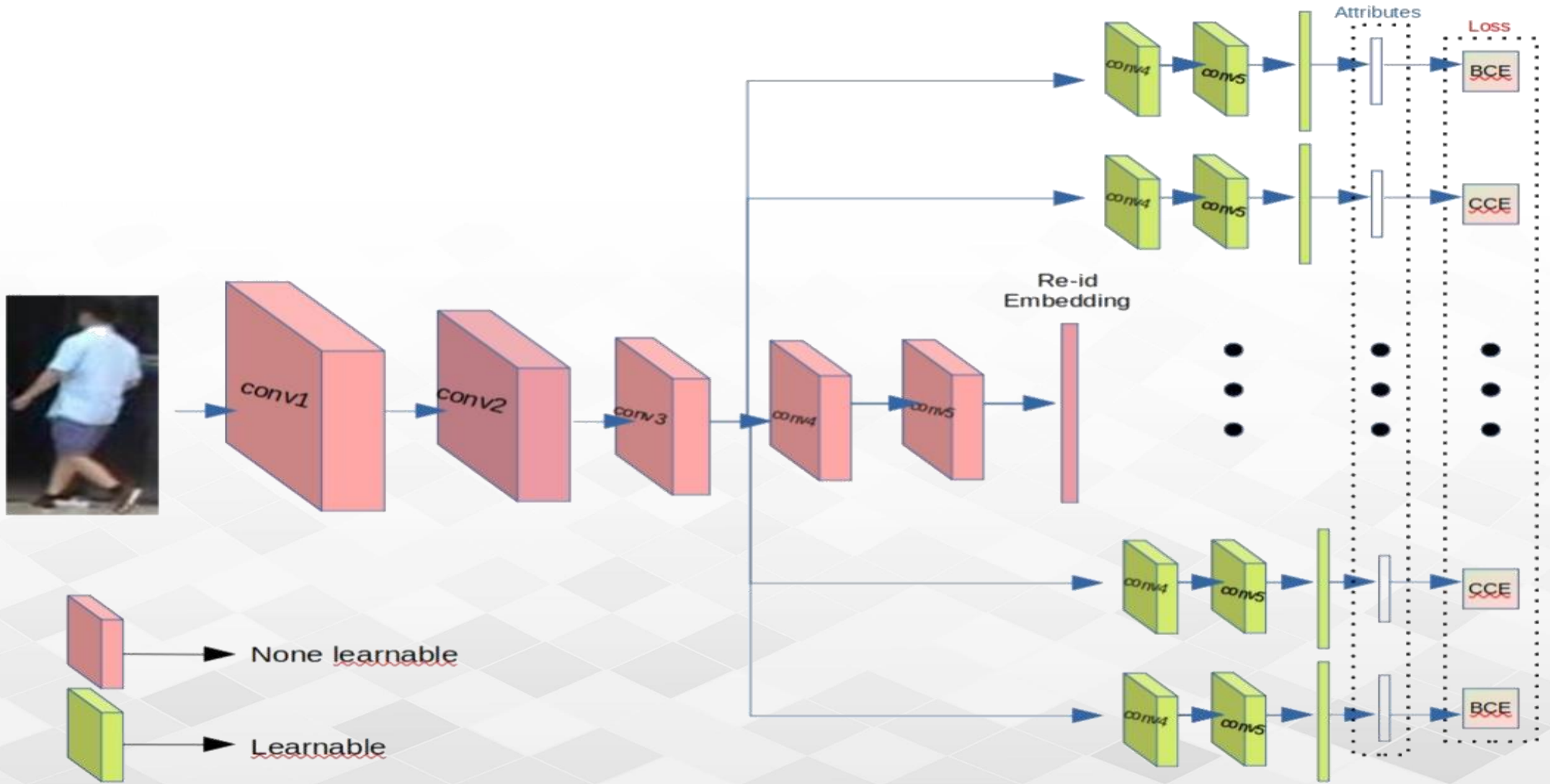
**Figure 6:** The SI value on the train set of the CA-Duke dataset in the middle layers of the pre-trained OSNet.



**Figure 7:** The SI value on the train set of the CA-Market dataset in the middle layers of the pre-trained OSNet.



# Multi-Branch Network



**Table 11**

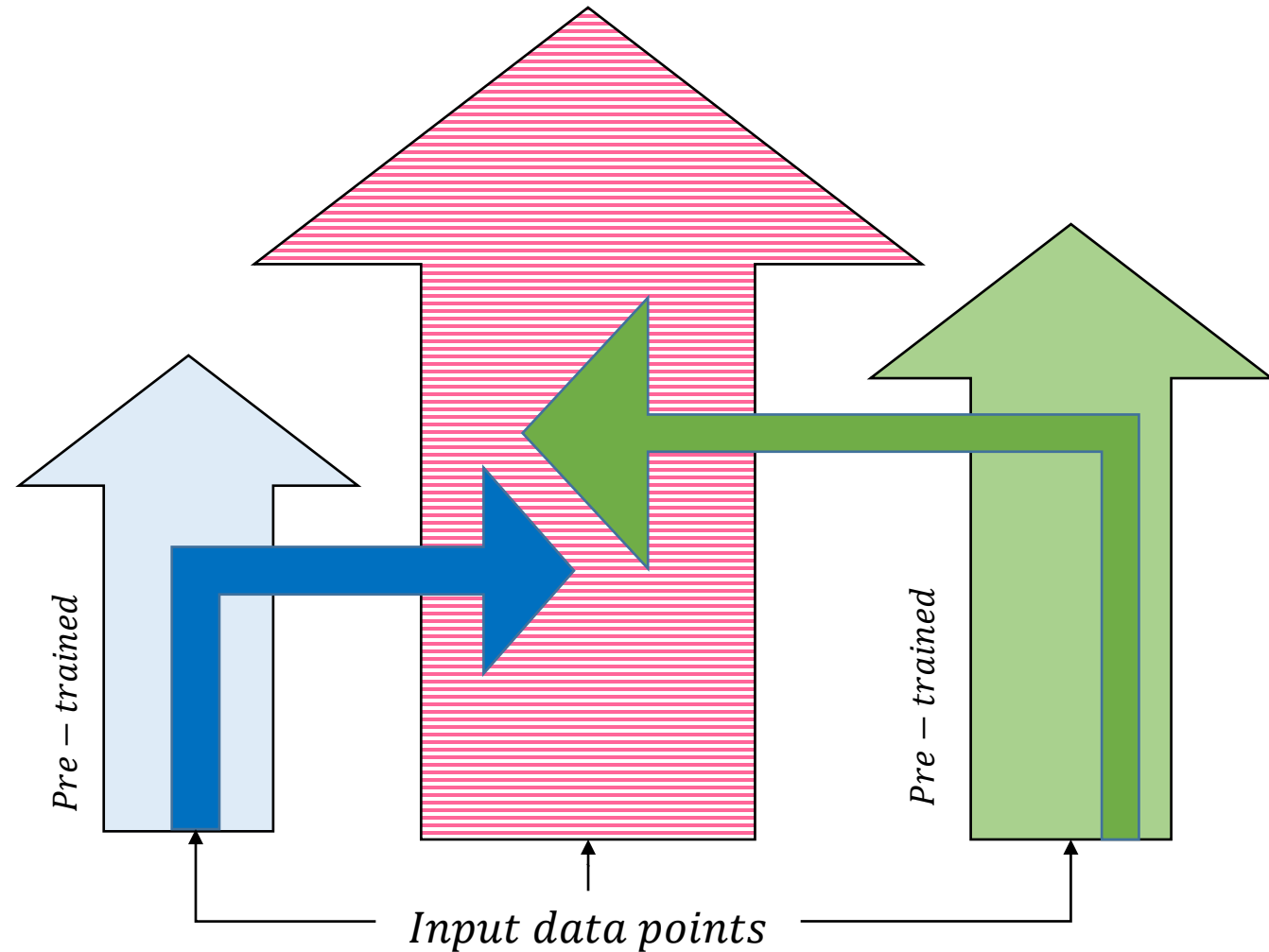
Comparing the results of the proposed multi-branch network with OSNet\_x1\_0 as the baseline, for person re-identification task with the state-of-the-art models.

Models	Market-1501				DukeMTMC-reID			
	rank-1	rank-5	rank-10	mAP	rank-1	rank-5	rank-10	mAP
APDR [10]	93.10	97.20	98.20	80.1	84.30	92.40	94.70	69.70
DARelD [42]	94.60	-	-	87.00	88.90	-	-	78.40
AICNet [53]	93.90	-	-	81.80	84.20	-	-	70.40
GPS [54]	95.20	<b>98.40</b>	<b>99.10</b>	87.80	88.20	95.20	96.70	78.70
Pose-Embedded Feature Branch [55]	92.70	-	-	81.30	86.20	-	-	72.60
RANGEv2 + K-reciprocal [56]	95.10	-	-	91.30	88.70	-	-	84.20
DADAA [57]	92.60	-	-	88.80	83.10	-	-	77.80
ASNet [58]	<b>96.30</b>	-	-	90.50	90.60	-	-	82.70
AL-APR [12]	89.00	95.60	97.27	74.38	80.52	-	-	63.67
AFFNet [54]	93.90	-	-	81.70	84.60	-	-	70.70
PAAN(w SB) [44]	95.53	-	-	84.26	82.59	-	-	65.53
JVIA [59]	94.70	97.00	97.90	88.40	85.60	-	-	85.80
AAB [11]	96.10	-	-	88.60	89.90	-	-	80.40
ARN [60]	82.36	-	-	72.37	73.83	-	-	62.82
AANet-152 + RR [8]	95.10	-	97.94	92.38	90.36	-	-	<b>86.87</b>
OSNet_X1_0 [47]	94.80	-	-	84.9	88.60	-	-	73.50
<b>Ours (Vectorized)</b>	87.54	88.32	91.02	84.99	86.52	87.89	89.24	75.17
<b>Ours (Baseline + Re-ranking)</b>	94.77	97.33	98.07	<b>92.64</b>	<b>94.39</b>	<b>96.45</b>	<b>96.90</b>	83.58

### 3.3.6 Layer-wise Fusion

# Layer-wise Fusion

- In a classification (regression) problem, the required features may be extracted by fusing some prepared features from some pre-trained models.
- If we design a new architecture of DNN which can combine desired features from some pre-trained DNNs, more generalized DNNs can be learned.
- In such an architecture, there are a main body which is integrated by some branches from other DNNs at different points. The main body forms the main flow of data but each branch insert information-flow.
- We can use complexity measures like SI(Sml) to find some desired features from other pre-trained DDNs.



## Some notes

- Fuse at any point where SI(Sml) may increase significantly.
- Fusion is done in as a forward design operation.

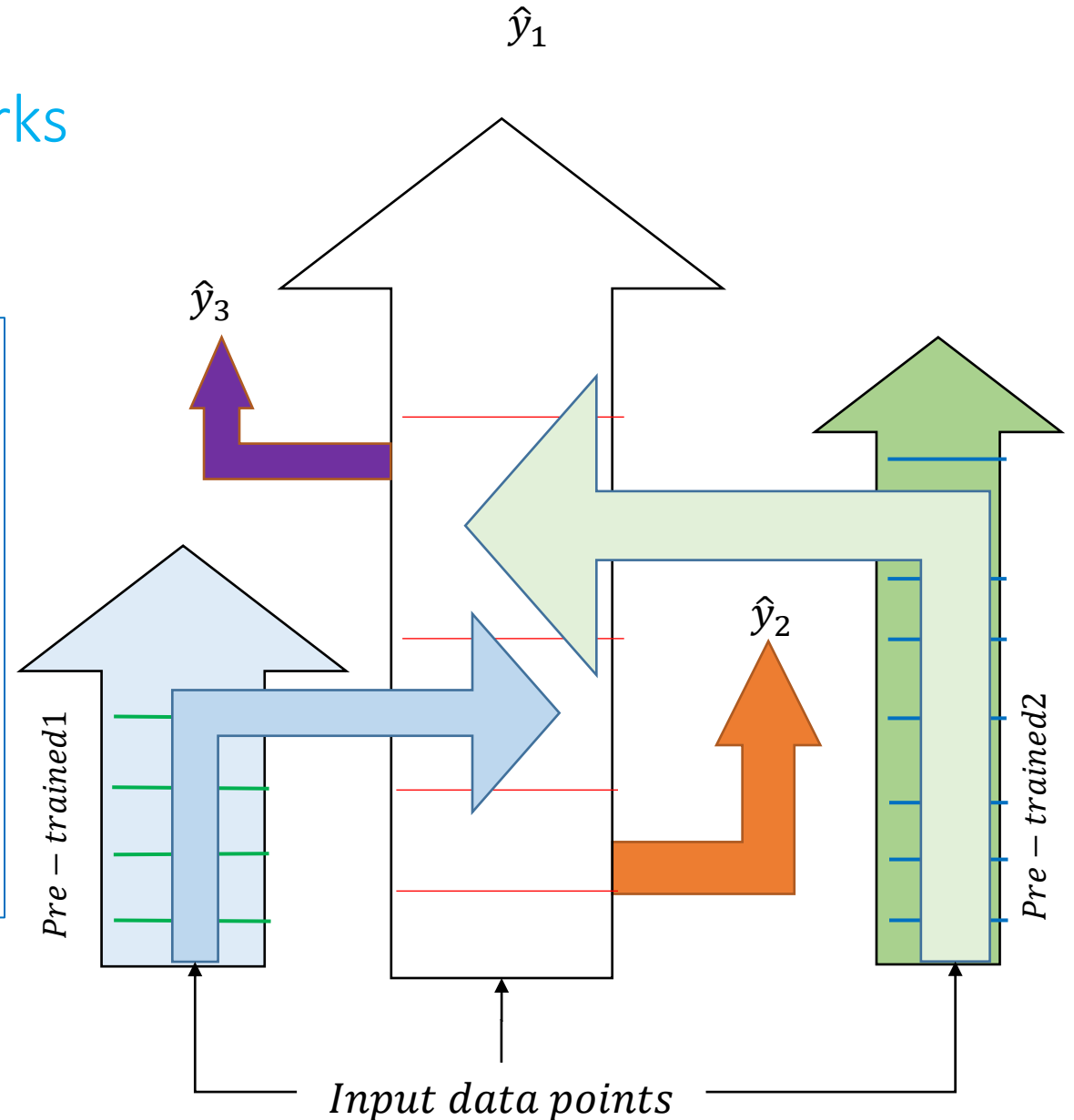


### 3.3.7 Forward Design

# Forward Design of Deep Neural Networks

In forward design of deep neural network, flowing tasks are done in a way that SI(Sml) maximizes in a classification(regression) problem.

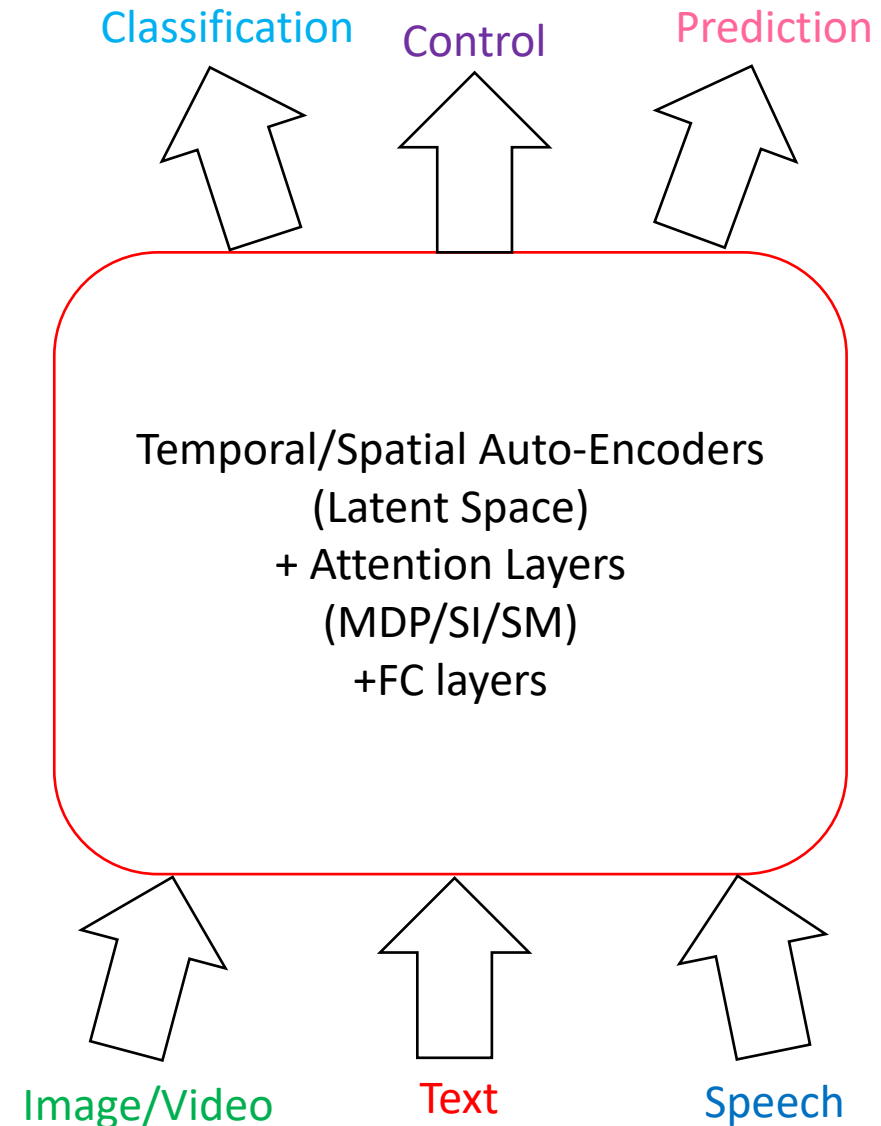
1. Organization of different layers
2. Adjusting the resolution, width, and depth of the network
3. Adding new branches on some layers where SI(Sml) increases significantly.
4. Fusing at some nodes from some other pre-trained networks through which SI(Sml) increases significantly.



### 3.3.8 Forward Multi-Task Design

A neural network which receives multi-modal data and it is trained to do multi-tasks in a supervised method. Such network works as a generalist AI agent. The key idea is that by appropriate temporal/spatial encoders, different data streams are encoded as the latent space; then the required features for each task are chosen from the latent space by using Sml(SI).

- **Language Tasks**
  - Speech recognition
  - Speech generating
  - ...
- **Vision Tasks**
  - Image recognition
  - Action Recognition
  - Text recognition
  - ...
- **Physical world tasks**
  - Motion tasks
  - Grasping tasks
  - ...



Narrow AI → General and wide AI

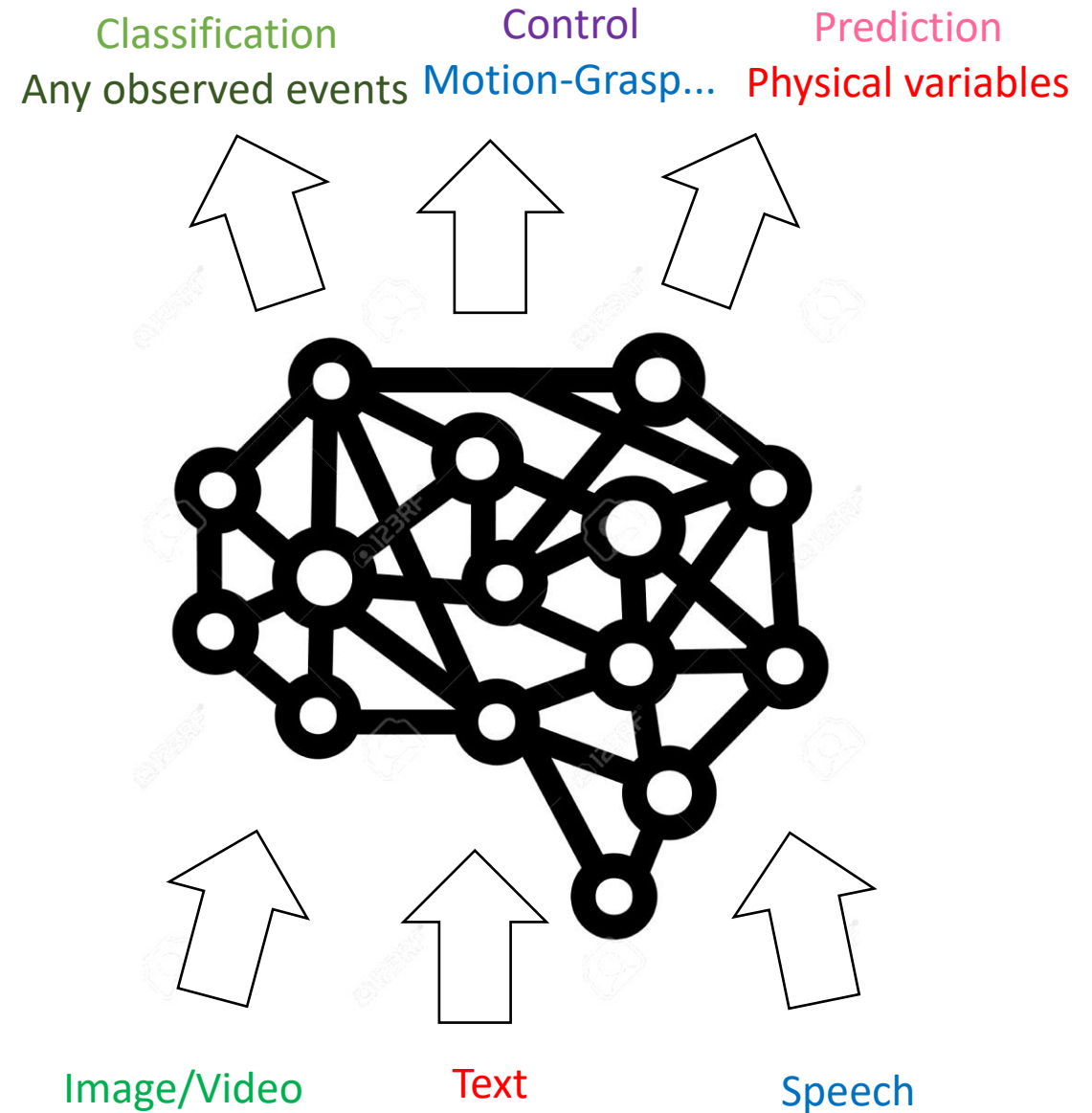
## Artificial General Intelligent

**A neural network which receives multi-modal data and it is trained to do multi-tasks in an un-supervised method.**

Such network works as a generalist AI agent which must learn without using labels and targets.

Some Steps

1. Receive all measuring signals: Visual, inertial, force,.....
2. Encode them as latent space by using appropriate spatial, temporal, and recurrent encoders
3. Using some attention layers and mechanisms to reveals some useful features which maximizes the gathered information from the environment
4. Learning to motion in the environment and to label objects .
5. Learning to do the required physical tasks.

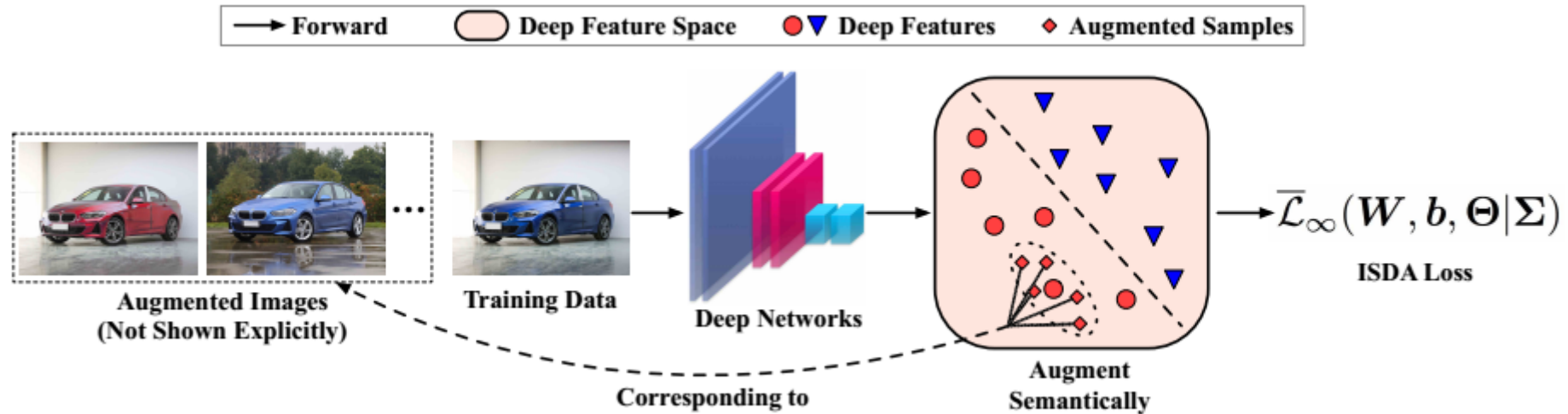


### 3.9 Related works in local Layer-wise learning

- To introduce a learning approach that is more plausible with the learning in the brain
- To have better learning and design in accuracy, compactness, speed,...
- To increase the generalization and robustness of DNNs

# (1) Implicit Semantic Data Augmentation for Deep Networks

Yulin Wang et al. 2020, Department of Automation, Tsinghua University, Beijing, China



The proposed implicit semantic data augmentation (ISDA) has two important components, i.e., (1) online estimation of class-conditional covariance matrices and (2) optimization with a robust loss function.

The first component aims to find a distribution from which we can sample meaningful semantic transformation directions for data augmentation, while the second saves us from explicitly generating a large amount of extra training data, leading to remarkable efficiency compared to existing data augmentation techniques.

## Comparison Tables

*Image net* →  
*Cifar10 + Cifar100* ↓

Table 1: Single crop error rates (%) of different deep networks on the validation set of ImageNet. We report the results of our implementation with and without ISDA. The better results are **bold-faced**, while the numbers in brackets denote the performance improvement achieved by ISDA. We also report the theoretical computational overhead and the additional training time introduced by ISDA in the last two columns, which is obtained with 8 Tesla V100 GPUs.

Networks	Params	Top-1 / Top-5 Error Rates (%)		Additional Cost (Theoretical)	Additional Cost (Wall Time)
		Baselines	ISDA		
ResNet-50 [4]	25.6M	23.0 / 6.8	<b>21.9</b> <sub>(1.1)</sub> / <b>6.3</b>	0.25%	7.6%
ResNet-101 [4]	44.6M	21.7 / 6.1	<b>20.8</b> <sub>(0.9)</sub> / <b>5.7</b>	0.13%	7.4%
ResNet-152 [4]	60.3M	21.3 / 5.8	<b>20.3</b> <sub>(1.0)</sub> / <b>5.5</b>	0.09%	5.4%
DenseNet-BC-121 [5]	8.0M	23.7 / 6.8	<b>23.2</b> <sub>(0.5)</sub> / <b>6.6</b>	0.20%	5.6%
DenseNet-BC-265 [5]	33.3M	21.9 / 6.1	<b>21.2</b> <sub>(0.7)</sub> / <b>6.0</b>	0.24%	5.4%
ResNeXt-50, 32x4d [33]	25.0M	22.5 / 6.4	<b>21.3</b> <sub>(1.2)</sub> / <b>5.9</b>	0.24%	6.6%
ResNeXt-101, 32x8d [33]	88.8M	21.1 / 5.9	<b>20.1</b> <sub>(1.0)</sub> / <b>5.4</b>	0.06%	7.9%

Table 2: Evaluation of ISDA on CIFAR with different models. The average test error over the last 10 epochs is calculated in each experiment, and we report mean values and standard deviations in three independent experiments. The best results are **bold-faced**.

Method	Params	CIFAR-10	CIFAR-100
ResNet-32 [4]	0.5M	7.39 ± 0.10%	31.20 ± 0.41%
ResNet-32 + ISDA	0.5M	<b>7.09 ± 0.12%</b>	<b>30.27 ± 0.34%</b>
ResNet-110 [4]	1.7M	6.76 ± 0.34%	28.67 ± 0.44%
ResNet-110 + ISDA	1.7M	<b>6.33 ± 0.19%</b>	<b>27.57 ± 0.46%</b>
SE-ResNet-110 [34]	1.7M	6.14 ± 0.17%	27.30 ± 0.03%
SE-ResNet-110 + ISDA	1.7M	<b>5.96 ± 0.21%</b>	<b>26.63 ± 0.21%</b>
Wide-ResNet-16-8 [35]	11.0M	4.25 ± 0.18%	20.24 ± 0.27%
Wide-ResNet-16-8 + ISDA	11.0M	<b>4.04 ± 0.29%</b>	<b>19.91 ± 0.21%</b>
Wide-ResNet-28-10 [35]	36.5M	3.82 ± 0.15%	18.53 ± 0.07%
Wide-ResNet-28-10 + ISDA	36.5M	<b>3.58 ± 0.15%</b>	<b>17.98 ± 0.15%</b>
ResNeXt-29, 8x64d [33]	34.4M	3.86 ± 0.14%	18.16 ± 0.13%
ResNeXt-29, 8x64d + ISDA	34.4M	<b>3.67 ± 0.12%</b>	<b>17.43 ± 0.25%</b>
DenseNet-BC-100-12 [5]	0.8M	4.90 ± 0.08%	22.61 ± 0.10%
DenseNet-BC-100-12 + ISDA	0.8M	<b>4.54 ± 0.07%</b>	<b>22.10 ± 0.34%</b>
DenseNet-BC-190-40 [5]	25.6M	3.52%	17.74%
DenseNet-BC-190-40 + ISDA	25.6M	<b>3.24%</b>	<b>17.42%</b>

### Algorithm 1 The ISDA Algorithm.

```

1: Input:  $\mathcal{D}$ ,  $\lambda_0$ 
2: Randomly initialize  $\mathbf{W}$ ,  $\mathbf{b}$  and  $\Theta$ 
3: for  $t = 0$  to  $T$  do
4:   Sample a mini-batch  $\{\mathbf{x}_i, y_i\}_{i=1}^B$  from  $\mathcal{D}$ 
5:   Compute  $\mathbf{a}_i = G(\mathbf{x}_i, \Theta)$ 
6:   Estimate the covariance matrices  $\Sigma_1, \Sigma_2, \dots, \Sigma_C$ 
7:   Compute  $\bar{\mathcal{L}}_\infty$  according to Eq. (4)
8:   Update  $\mathbf{W}$ ,  $\mathbf{b}$ ,  $\Theta$  with SGD
9: end for
10: Output:  $\mathbf{W}$ ,  $\mathbf{b}$  and  $\Theta$ 

```

$$\mathcal{L}_\infty(\mathbf{W}, \mathbf{b}, \Theta | \Sigma) \leq \frac{1}{N} \sum_{i=1}^N -\log\left(\frac{e^{\mathbf{w}_{y_i}^T \mathbf{a}_i + \mathbf{b}_{y_i}}}{\sum_{j=1}^C e^{\mathbf{w}_j^T \mathbf{a}_i + \mathbf{b}_j + \frac{\lambda}{2}(\mathbf{w}_j^T - \mathbf{w}_{y_i}^T)\Sigma_{y_i}(\mathbf{w}_j - \mathbf{w}_{y_i})}}\right) \triangleq \bar{\mathcal{L}}_\infty.$$

$\lambda$  is a positive coefficient to control the strength of semantic data augmentation. The strength at initial epochs is chosen low but linearly it increases:  $\lambda = (t/T) \times \lambda_0$

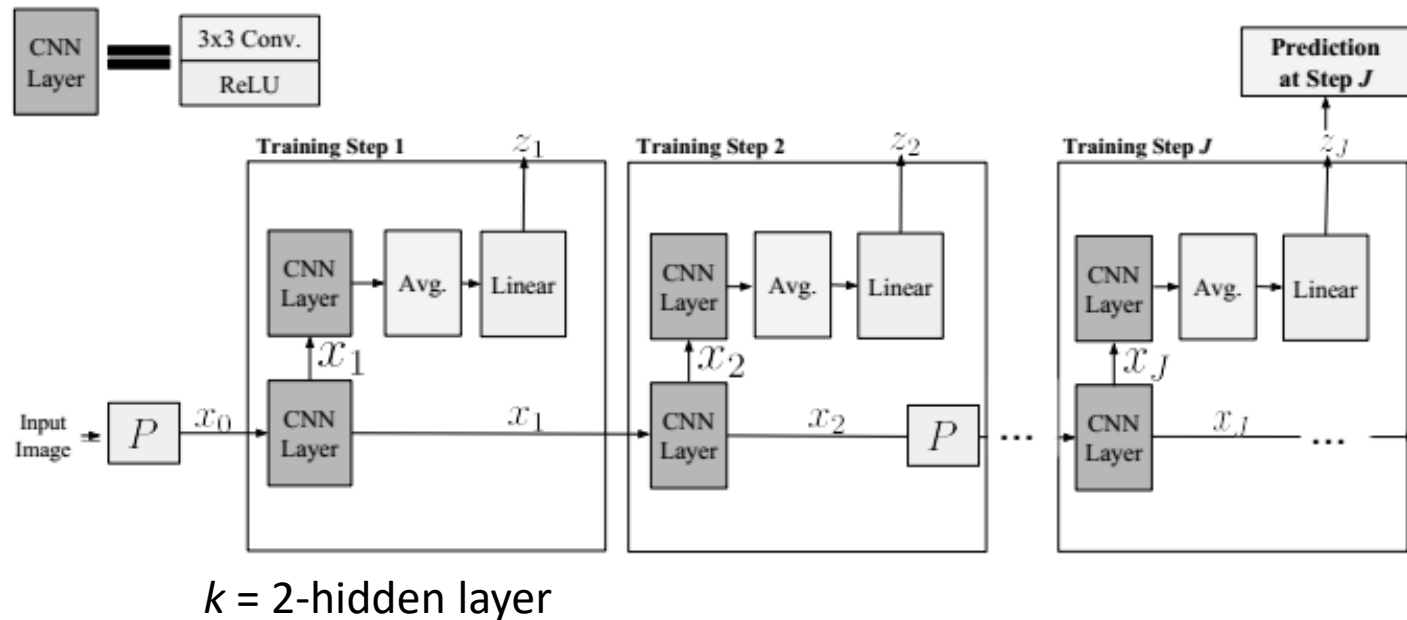


## (2) Greedy Layerwise Learning Can Scale to ImageNet

Belilovsky et al. (University of Montral)-*Proceedings of the 36 th International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019.

We find that solving sequential 1-hidden-layer auxiliary problems lead to a CNN that exceeds AlexNet performance on ImageNet.

Extending this training methodology to construct individual layers by solving 2-and-3-hidden layer auxiliary problems, we obtain an 11-layer network that exceeds several members of the VGG model family on ImageNet, and can train a VGG-11 model to the same accuracy as end-to-end learning.



---

### Algorithm 1 Layer Wise CNN

---

**Input:** Training samples  $\{x_0^n, y^n\}_{n \leq N}$   
**for**  $j \in 0..J - 1$  **do**  
    Compute  $\{x_j^n\}_{n \leq N}$  (via Eq.(1))  
     $(\theta_j^*, \gamma_j^*) = \arg \min_{\theta_j, \gamma_j} \hat{\mathcal{R}}(z_{j+1}; \theta_j, \gamma_j)$   
**end for**

---

# Comparison tables

Layer-wise Trained	Acc. (Ens.)
SimCNN ( $k = 1$ train )	88.3 (88.4)
SimCNN ( $k = 2$ train)	90.4 (90.7)
SimCNN( $k = 3$ train)	91.7 ( <b>92.8</b> )
BoostResnet (Huang et al., 2017)	82.1
ProvableNN (Malach et al., 2018)	73.4
(Mosca et al., 2017)	81.6
<b>Reference e2e</b>	
AlexNet	89
VGG <sup>1</sup>	92.5
WRN 28-10 (Zagoruyko et al. 2016)	<b>96.0</b>
<b>Alternatives</b>	[Ref.]
Scattering + Linear	82.3
FeedbackAlign (Bartunov et al., 2018)	62.6 [67.6]

Table 2. Results on CIFAR-10. Compared to the few existing methods using *only* layerwise training schemes we report much more competitive results to well known benchmark models that like ours do not use skip connections. In brackets e2e trained version of the model is shown when available.

$k = 1$  : the auxiliary classifier consists of only the linear  $A$  and  $L$  operators (1-hidden layer NN)  
 $K > 1$  :  $K$  hidden layers for auxiliary classifiers

	Top-1 (Ens.)	Top-5 (Ens.)
SimCNN ( $k = 1$ train)	58.1 (59.3)	79.7 (80.8)
SimCNN ( $k = 2$ train)	65.7 (67.1)	86.3 (87.0)
SimCNN ( $k = 3$ train)	69.7 (71.6)	88.7 (89.8)
VGG-11 ( $k = 3$ train)	67.6 (70.1)	88.0 (89.2)
VGG-11 (e2e train)	67.9	88.0
<b>Alternative</b>	[Ref.]	[Ref.]
DTargetProp (Bartunov et al., 2018)	1.6 [28.6]	5.4 [51.0]
FeedbackAlign (Xiao et al., 2019)	6.6 [50.9]	16.7 [75.0]
Scat. + Linear (Oyallon et al., 2018)	17.4	N/A
Random CNN	12.9	N/A
FV + Linear (Sánchez et al., 2013)	54.3	74.3
<b>Reference e2e CNN</b>		
AlexNet	56.5	79.1
VGG-13	69.9	89.3
VGG-19	72.9	90.9
Resnet-152	78.3	94.1

Table 3. Single crop validation acc. on ImageNet. Our SimCNN models use  $J = 8$ . In parentheses see the ensemble prediction.

# (3) Hebbian Semi-Supervised Learning in a Sample Efficiency Setting?

Gabriele Lagani, 2021, *Computer Science Department, University of Pisa, Pisa, Italy*

## Highlights

1. A semi supervised training strategy that combines Hebbian learning with gradient descent  
**Hebbian for internal layers + SGD for the final fully connected layer**
2. All internal layers (both convolutional and fully connected) are pre-trained using an unsupervised approach based on Hebbian learning
3. The last fully connected layer (the classification layer) is trained using Stochastic Gradient Descent (SGD)
4. The learning approach has been applied to CIFAR10, CIFAR100, tiny ImageNet
5. In regimes when the number of available labeled samples is low the proposed approach outperforms the other approaches in almost all the cases

# Hebians

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \Delta \mathbf{W}$$

$$\Delta \mathbf{W} = \eta \mathbf{y} \mathbf{x}$$

“neurons that fire together wire together.”

$$\Delta \mathbf{W} = \eta \mathbf{y} \mathbf{x} - \lambda \mathbf{W}$$

Adding Stable Term

In particular, the term  $\lambda$  can be chosen in the form  $\lambda = \eta y$

$$\Delta \mathbf{W} = \eta \mathbf{y} \mathbf{x} - \eta \mathbf{y} \mathbf{W} = \eta \mathbf{y} (\mathbf{x} - \mathbf{W})$$

In a complex neural network, we need a strategy to prevent neurons from learning redundant information.

The Hebbian PCA learning rule is obtained by minimizing the representation error loss function, defined as:

$$L(\mathbf{w}_i) = E[(\mathbf{x} - \sum_{j=1}^i y_j \mathbf{w}_j)^2]$$

The classification layer is placed on top of the various internal layers (L1; :::; L5)

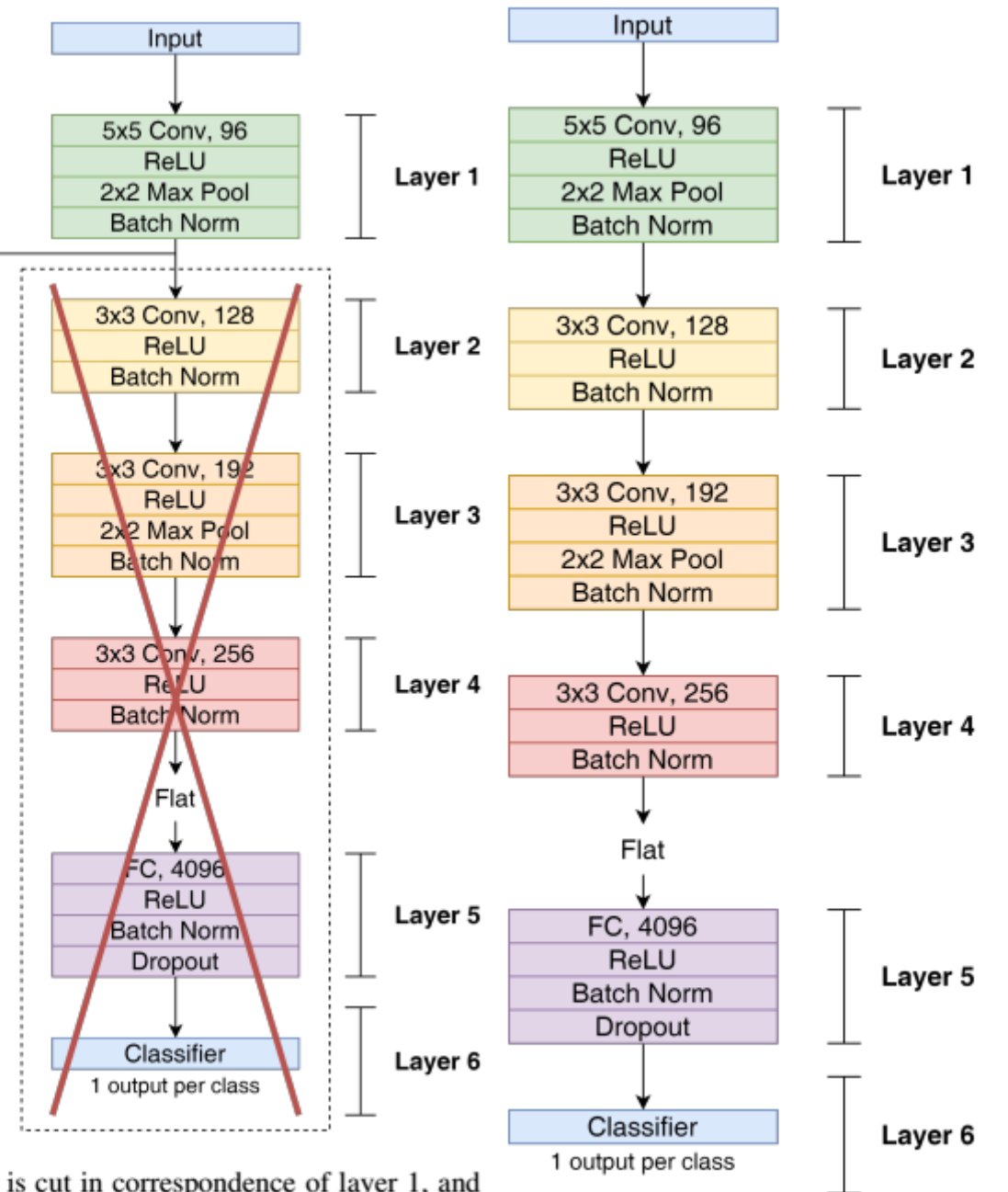
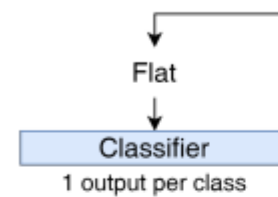


Figure 4: A neural network is cut in correspondence of layer 1, and a linear classifier is placed on top of the features extracted from that layer, in order to evaluate their quality in classification tasks.

: 1: The neural network used for the experiments.

# Comparison Table(CIFAR10)

Table 1: CIFAR10 accuracy (top-1) and 95% confidence intervals, obtained with a linear classifier on top of various layers, for the various sample efficiency regimes. Results obtained with supervised backprop (BP), VAE-based semi-supervised approach(VAE), Hebbian PCA (HPCA), and HPCA plus Fine Tuning (HPCA+FT) are compared. It is possible to observe that, in regimes where the number of available samples is low (roughly between 1% and 5% of the total available samples), HPCA performs better than BP and VAE approaches in almost all the cases, leading to an improvement up to almost 5% (on layer 3, in the 1% regime) w.r.t. non-Hebbian approaches. HPCA+FT helps to further boost accuracy.

Regime: r%- where the percentage of labeled samples is r%

Regimes	Method	L1	L2	L3	L4	L5
1%	BP	33.27 $\pm$ 0.44	34.56 $\pm$ 0.34	36.80 $\pm$ 0.52	35.47 $\pm$ 0.58	35.18 $\pm$ 0.57
	VAE	33.54 $\pm$ 0.27	34.41 $\pm$ 0.84	29.92 $\pm$ 1.25	24.91 $\pm$ 0.66	22.54 $\pm$ 0.60
	HPCA	36.78 $\pm$ 0.46	37.26 $\pm$ 0.14	41.31 $\pm$ 0.57	39.33 $\pm$ 0.72	38.46 $\pm$ 0.44
	HPCA+FT	<b>37.01</b> $\pm$ 0.42	<b>37.65</b> $\pm$ 0.19	<b>41.88</b> $\pm$ 0.53	<b>40.06</b> $\pm$ 0.65	<b>39.75</b> $\pm$ 0.50
2%	BP	37.33 $\pm$ 0.25	39.01 $\pm$ 0.19	42.34 $\pm$ 0.51	41.50 $\pm$ 0.32	41.10 $\pm$ 0.39
	VAE	37.65 $\pm$ 0.35	39.13 $\pm$ 0.40	36.52 $\pm$ 0.47	29.39 $\pm$ 0.32	26.78 $\pm$ 0.72
	HPCA	41.13 $\pm$ 0.30	<b>41.63</b> $\pm$ 0.18	45.76 $\pm$ 0.41	44.70 $\pm$ 0.45	43.15 $\pm$ 0.45
	HPCA+FT	<b>41.60</b> $\pm$ 0.28	42.12 $\pm$ 0.24	<b>46.56</b> $\pm$ 0.38	<b>45.61</b> $\pm$ 0.19	<b>45.51</b> $\pm$ 0.43
3%	BP	40.49 $\pm$ 0.26	41.90 $\pm$ 0.40	45.13 $\pm$ 0.53	45.26 $\pm$ 0.22	44.52 $\pm$ 0.24
	VAE	41.22 $\pm$ 0.27	43.16 $\pm$ 0.44	42.60 $\pm$ 0.87	31.91 $\pm$ 0.44	29.00 $\pm$ 0.33
	HPCA	44.16 $\pm$ 0.42	44.84 $\pm$ 0.08	48.92 $\pm$ 0.17	47.70 $\pm$ 0.57	45.60 $\pm$ 0.27
	HPCA+FT	<b>44.74</b> $\pm$ 0.08	<b>45.61</b> $\pm$ 0.28	<b>49.75</b> $\pm$ 0.41	<b>48.94</b> $\pm$ 0.45	<b>48.80</b> $\pm$ 0.27
4%	BP	43.38 $\pm$ 0.22	45.43 $\pm$ 0.18	49.51 $\pm$ 0.49	48.96 $\pm$ 0.48	48.80 $\pm$ 0.24
	VAE	44.39 $\pm$ 0.30	45.88 $\pm$ 0.39	46.01 $\pm$ 0.40	34.26 $\pm$ 0.21	31.15 $\pm$ 0.35
	HPCA	46.37 $\pm$ 0.16	47.16 $\pm$ 0.28	50.70 $\pm$ 0.26	49.45 $\pm$ 0.15	47.75 $\pm$ 0.54
	HPCA+FT	<b>47.10</b> $\pm$ 0.25	<b>48.26</b> $\pm$ 0.09	<b>52.00</b> $\pm$ 0.16	<b>51.05</b> $\pm$ 0.29	<b>51.28</b> $\pm$ 0.28
5%	BP	45.11 $\pm$ 0.21	47.57 $\pm$ 0.29	50.61 $\pm$ 0.32	50.54 $\pm$ 0.23	50.42 $\pm$ 0.14
	VAE	46.31 $\pm$ 0.39	48.21 $\pm$ 0.21	48.98 $\pm$ 0.34	36.32 $\pm$ 0.35	32.75 $\pm$ 0.32
	HPCA	47.51 $\pm$ 0.65	48.69 $\pm$ 0.37	51.69 $\pm$ 0.56	50.44 $\pm$ 0.43	48.51 $\pm$ 0.32
	HPCA+FT	<b>48.49</b> $\pm$ 0.44	<b>50.14</b> $\pm$ 0.46	<b>53.33</b> $\pm$ 0.52	<b>52.49</b> $\pm$ 0.16	<b>52.20</b> $\pm$ 0.37
10%	BP	51.60 $\pm$ 0.40	54.60 $\pm$ 0.31	57.97 $\pm$ 0.28	<b>57.63</b> $\pm$ 0.23	57.30 $\pm$ 0.22
	VAE	53.83 $\pm$ 0.26	<b>56.33</b> $\pm$ 0.22	57.85 $\pm$ 0.22	52.26 $\pm$ 1.08	45.67 $\pm$ 1.15
	HPCA	52.57 $\pm$ 0.29	53.29 $\pm$ 0.25	56.09 $\pm$ 0.38	54.24 $\pm$ 0.28	52.68 $\pm$ 0.36
	HPCA+FT	<b>54.36</b> $\pm$ 0.32	56.08 $\pm$ 0.28	<b>58.46</b> $\pm$ 0.15	56.54 $\pm$ 0.23	<b>57.35</b> $\pm$ 0.18
25%	BP	60.43 $\pm$ 0.26	64.96 $\pm$ 0.18	66.63 $\pm$ 0.17	68.04 $\pm$ 0.05	68.04 $\pm$ 0.20
	VAE	<b>62.51</b> $\pm$ 0.24	<b>67.26</b> $\pm$ 0.32	<b>68.48</b> $\pm$ 0.21	<b>68.79</b> $\pm$ 0.29	<b>68.70</b> $\pm$ 0.15
	HPCA	58.30 $\pm$ 0.28	59.20 $\pm$ 0.24	59.98 $\pm$ 0.20	57.54 $\pm$ 0.20	56.46 $\pm$ 0.18
	HPCA+FT	61.45 $\pm$ 0.26	65.25 $\pm$ 0.16	64.71 $\pm$ 0.17	62.43 $\pm$ 0.13	64.77 $\pm$ 0.22
100%	BP	61.59 $\pm$ 0.08	67.67 $\pm$ 0.11	73.87 $\pm$ 0.15	83.88 $\pm$ 0.04	84.71 $\pm$ 0.02
	VAE	<b>67.53</b> $\pm$ 0.22	<b>75.83</b> $\pm$ 0.31	<b>80.78</b> $\pm$ 0.28	<b>84.27</b> $\pm$ 0.35	<b>85.23</b> $\pm$ 0.26
	HPCA	64.69 $\pm$ 0.29	65.92 $\pm$ 0.14	64.43 $\pm$ 0.21	61.24 $\pm$ 0.22	61.16 $\pm$ 0.33
	HPCA+FT	66.76 $\pm$ 0.13	75.16 $\pm$ 0.20	79.90 $\pm$ 0.18	83.55 $\pm$ 0.33	84.38 $\pm$ 0.22

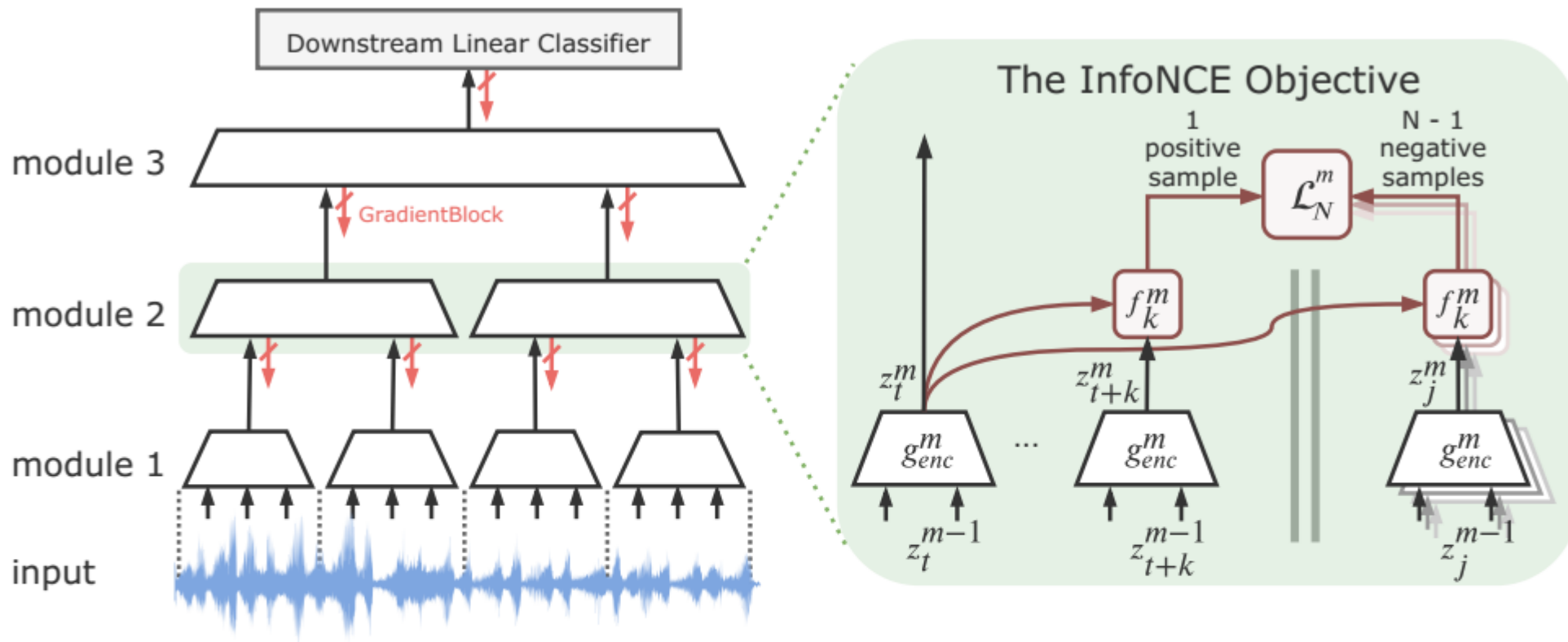
## (4) Greedy InfoMax for Self-Supervised Representation Learning(GIM)

Sindy Lowe et al. University of Amsterdam

### Highlights

1. we propose a novel deep learning method for **local self-supervised representation learning** that does not require labels nor end-to-end backpropagation but exploits the natural order in data instead.
2. Inspired by the observation that biological neural networks appear to learn without backpropagating a global error signal, **we split a deep neural network into a stack of gradient-isolated modules.**
3. Each module is trained to maximize the mutual information between its consecutive outputs using the InfoNCE bound from Oord et al.(2018).





*Figure 1. The Greedy InfoMax Learning Approach. (Left)* For the self-supervised learning of representations, we stack a number of modules through which the input is forward-propagated in the usual way, but gradients do not propagate backwards. Instead, every module is trained greedily using a local loss. *(Right)* Every encoding module maps its inputs  $z_t^{m-1}$  at time-step  $t$  to  $g_{enc}^m(\text{GradientBlock}(z_t^{m-1})) = z_t^m$ , which is used as the input for the following module. The InfoNCE objective is used for its greedy optimization. This loss is calculated by contrasting the predictions of a module for its future representations  $z_{t+k}^m$  against negative samples  $z_j^m$ , which enforces each module to maximally preserve the information of its inputs. We employ an additional autoregressive module  $g_{ar}$ , which is not depicted here.

# Architecture & Comparison

*Table 3. General outline of our architecture*

Layer	Output Size (Sequence Length $\times$ Channels)	Parameters		
		Kernel	Stride	Padding
Input	$20480 \times 1$			
Conv1	$4095^2 \times 512$	10	5	2
Conv2	$1023^2 \times 512$	8	4	2
Conv3	$512^2 \times 512$	4	2	2
Conv4	$257^2 \times 512$	4	2	2
Conv5	$128 \times 512$	1	2	1
GRU	$128 \times 256$	-	-	-

<sup>2</sup>For applying the InfoNCE objective on these layers, we randomly sample a time-window of size 128 to decrease the dimensionality.

*Table 1. Results for classifying speaker identity and phone labels in the LibriSpeech dataset. All models use the same audio input sizes and the same architecture. GIM creates representations that are useful for audio classification tasks despite its greedy training and lack of a global objective.*

Method	Phone	Speaker
	Classification Accuracy	Classification Accuracy
MFCC features	39.7%	17.6%
Randomly initialized	27.6%	1.9%
Supervised	74.6%	98.5%
Greedy Supervised	71.1%	84.5%
CPC (Oord et al., 2018) <sup>a</sup>	64.6%	97.4%
Greedy InfoMax (GIM)	60.0%	97.5%

<sup>a</sup>In our reimplementation, we achieved 62.2% for the phone and 98.8% for the speaker classification task.

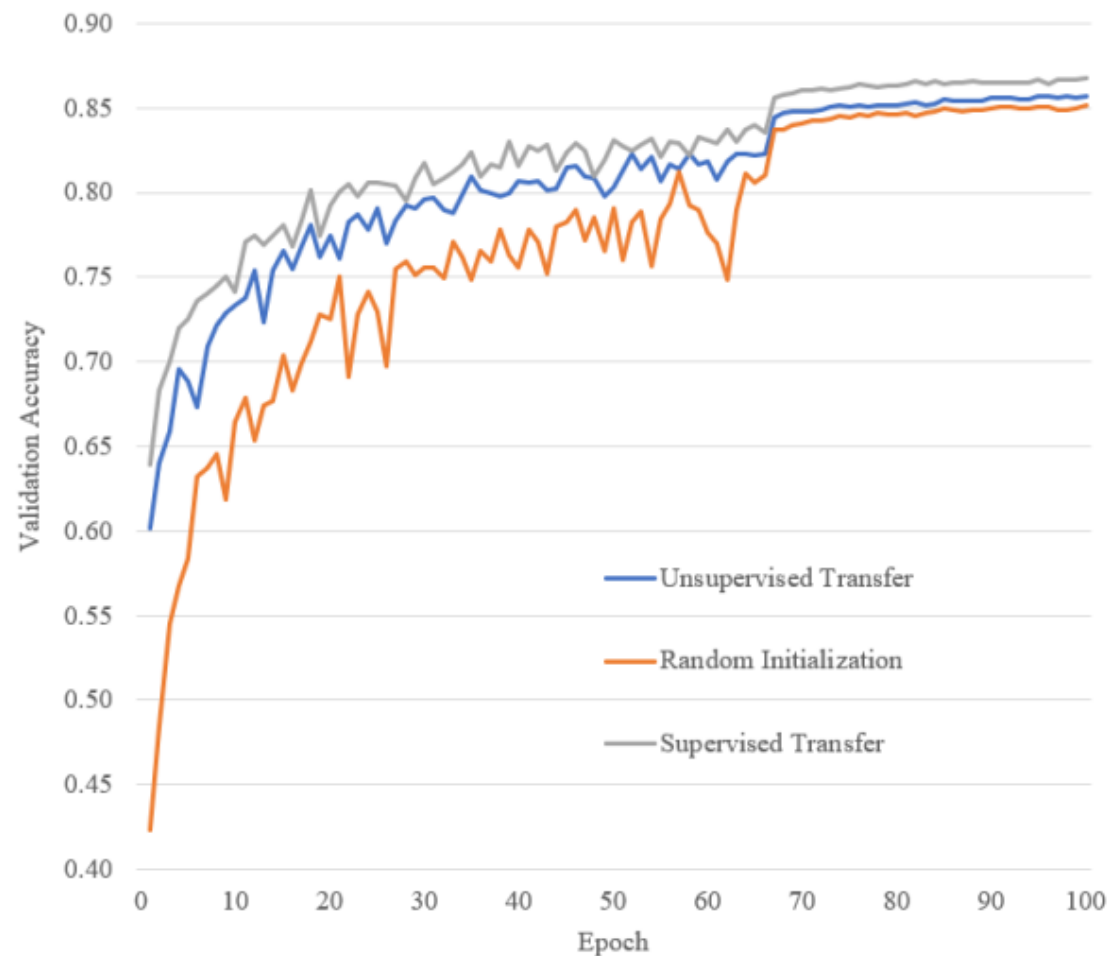


# (5)Layer-Wise Contrastive Unsupervised Representation Learning

Stephen Zhao, 2019

- In this work, we focus on the set of unsupervised learning approaches that Arora et al. (2019) call “contrastive unsupervised representation learning”.
- We use a layer-wise adaptation, starting within the context of images.
- We train feature representations of semantically similar images (i.e. nearby patches within the same image) to be closer than that of unrelated images (e.g. random patches), to learn convolutional neural network filters from unlabeled data.

Figure 3: Performance on CIFAR-10. The learning rate is 0.001 for the first two-thirds of the training, and is 0.0001 afterwards. Results are averaged over 3 independent runs for each line (keeping transferred filters constant across runs, but allowing for fine-tuning).



# (6) LoCo: Local Contrastive Representation Learning

Yuwen Xiong et al.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

- In this work, we discover that by overlapping local blocks stacking on top of each other, we effectively increase the decoder depth and allow upper blocks to implicitly send feedbacks to lower blocks.
- This simple design closes the performance gap between local learning and end-to-end contrastive learning algorithms for the first time.
- Aside from standard ImageNet experiments, we also show results on complex downstream tasks such as object detection and instance segmentation directly using readout features

# LoCo Architecture

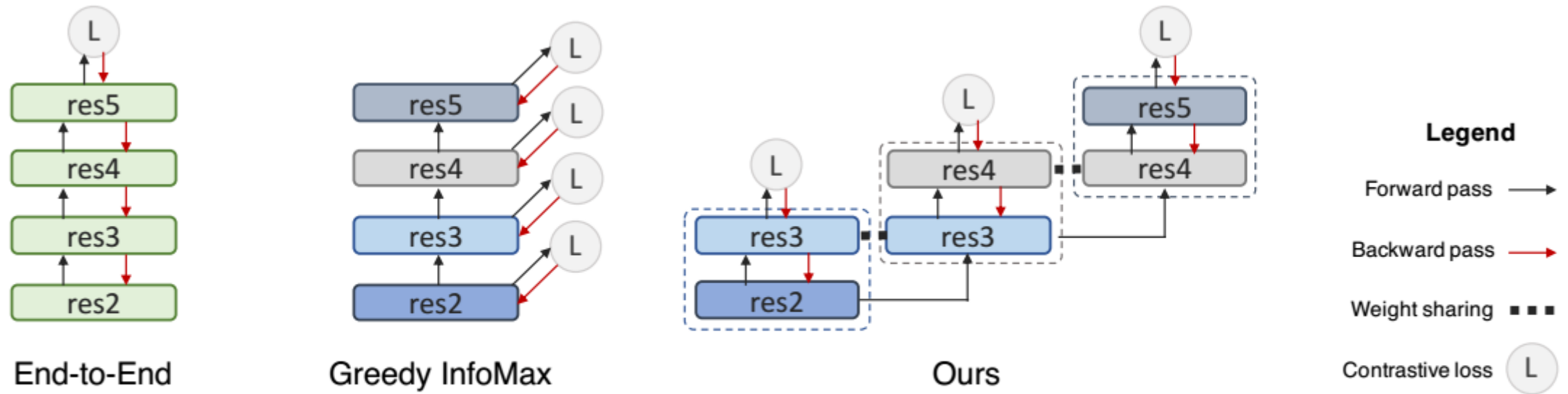


Figure 1: Comparison between End-to-End, Greedy InfoMax (GIM) and LoCo

In this paper, we used SimCLR as our main baseline, since it has superior performance on ImageNet, and we apply the changes proposed in GIM(Greedy Infomax) on top of SimCLR as our local learning baseline.

In our experiments, we find that simply applying GIM on SimCLR results in a significant loss in performance and in the next section we will explain our techniques to bridge the performance gap.

# Comparison tables

Method	Architecture	Acc.	Local
Local Agg. [62]	ResNet-50	60.2	
MoCo [22]	ResNet-50	60.6	
PIRL [45]	ResNet-50	63.6	
CPC v2 [56]	ResNet-50	63.8	
SimCLR* [11]	ResNet-50	69.3	
SimCLR [11]	ResNet-50	<b>69.8</b>	
GIM [39]	ResNet-50	64.7	✓
LoCo (Ours)	ResNet-50	69.5	✓
SimCLR [11]	ShuffleNet v2-50	69.1	
GIM [39]	ShuffleNet v2-50	63.5	✓
LoCo (Ours)	ShuffleNet v2-50	<b>69.3</b>	✓

Table 1: ImageNet accuracies of linear classifiers trained on representations learned with different unsupervised methods, SimCLR\* is the result from the SimCLR paper with 1000 training epochs.

Greedy InfoMax (GIM)

SimCLR: T. Chen... A simple framework for contrastive learning of visual representations, 2020.

Method	Arch	COCO		Cityscapes	
		AP <sup>bb</sup>	AP	AP <sup>bb</sup>	AP
Supervised	R-50	33.9	31.3	33.2	27.1
Backbone weights with 100 Epochs					
SimCLR	R-50	32.2	29.9	33.2	28.6
GIM	R-50	27.7 (-4.5)	25.7 (-4.2)	30.0 (-3.2)	24.6 (-4.0)
Ours	R-50	32.6 (+0.4)	30.1 (+0.2)	33.2 (+0.0)	28.4 (-0.2)
SimCLR	Sh-50	32.5	30.1	33.3	28.0
GIM	Sh-50	27.3 (-5.2)	25.4 (-4.7)	29.1 (-4.2)	23.9 (-4.1)
Ours	Sh-50	31.8 (-0.7)	29.4 (-0.7)	33.1 (-0.2)	27.7 (-0.3)
Backbone weights with 800 Epochs					
SimCLR	R-50	34.8	32.2	34.8	30.1
GIM	R-50	29.3 (-5.5)	27.0 (-5.2)	30.7 (-4.1)	26.0 (-4.1)
Ours	R-50	34.5 (-0.3)	32.0 (-0.2)	34.2 (-0.6)	29.5 (-0.6)
SimCLR	Sh-50	33.4	30.9	33.9	28.7
GIM	Sh-50	28.9 (-4.5)	26.9 (-4.0)	29.6 (-4.3)	23.9 (-4.8)
Ours	Sh-50	33.6 (+0.2)	31.2 (+0.3)	33.0 (-0.9)	28.1 (-0.6)

Table 2: Mask R-CNN results on COCO and Cityscapes. Backbone networks are frozen. “R-50” denotes ResNet-50 and “Sh-50” denotes ShuffleNet v2-50.

Average Precision (AP)

Average Precision bounding box (AP<sup>bb</sup>)

# (7) Progressive Stage-wise Learning for Unsupervised Feature Representation Enhancement

Zefan Li et al., Shanghai Jiao Tong University

In this work, we explore new dimensions of unsupervised learning by proposing the **Progressive Stage-wise Learning (PSL)** framework.

For a given unsupervised task, we design multilevel tasks and define different learning stages for the deep network. (Each new task has overlap with the former one)

Early learning stages are forced to focus on low-level tasks while late stages are guided to extract deeper information through harder tasks.

We discover that by progressive stage-wise learning, unsupervised feature representation can be effectively enhanced.

Our extensive experiments show that PSL consistently improves results for the leading unsupervised learning methods.

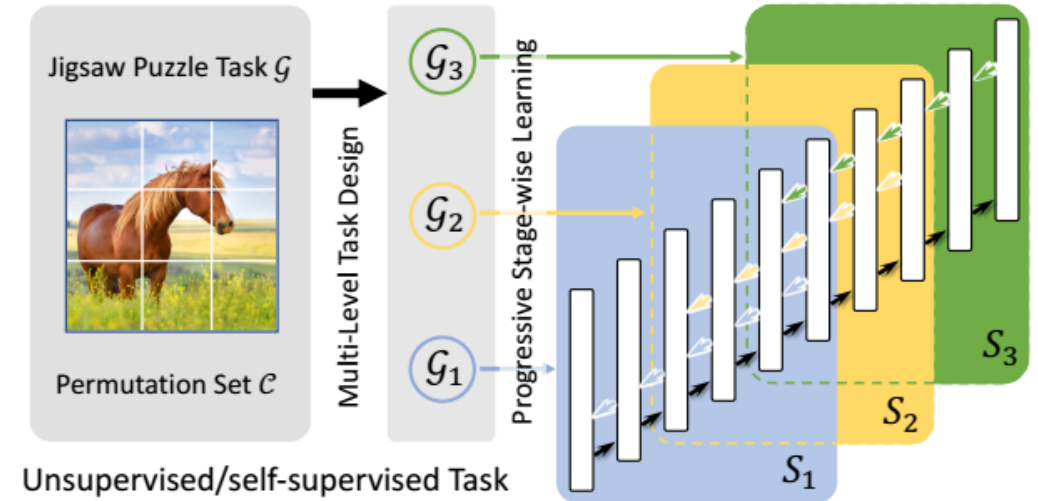


Figure 1. We present the framework of the proposed Progressive Stage-wise Learning (PSL) algorithm, aiming for improving unsupervised/self-supervised task. We take the jigsaw puzzle task  $\mathcal{G}$  for example. We first do multi-level task partition  $\mathcal{G} \rightarrow \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$  with an increased task complexity and perform progressive stage-wise training for different learning stages of the network. The black arrow denotes forward pass while colored arrow represents the backward pass of each learning stage (i.e.,  $S_1$ ,  $S_2$ , and  $S_3$ ).

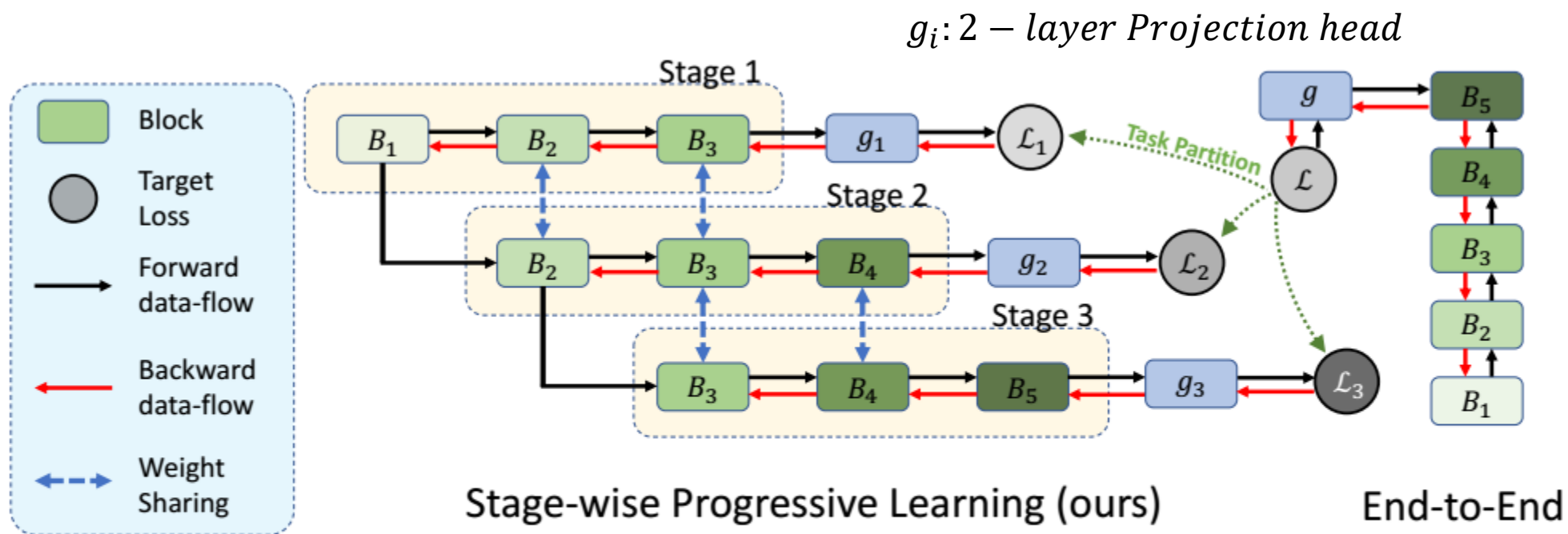


Figure 2. We present the detail of the proposed Stage-wise Progressive Learning framework. In the right is the end-to-end learning scheme while we present PSL in the middle.  $g$  and  $\{g_i\}_{i=1}^3$  are projection heads, mapping the intermediate representation to the target feature space. After the training is completed, we throw away the projection heads and use the backbone network for downstream tasks.



# Comparison Tables

Task	1% labels	10% labels
Supervised	48.4	80.4
Jigsaw [38]	45.4	79.6
Jigsaw+PSL	<b>48.7</b>	<b>83.5</b>
Rotation [21]	52.1	82.5
Rotation+PSL	<b>54.8</b>	<b>83.7</b>

Table 7. **Semi-supervised learning on ImageNet.** We use ResNet-50 as our backbone networks and report single-crop top-5 accuracy on the ImageNet validation set. All models are self-supervised trained on ImageNet and finetuned on 1% and 10% of the ImageNet training data, following [46, 37]. The supervised results are presented for reference.

Method	Arch	# Param(M)	Top 1
Colorization [49]	R50	24	39.6
BigBiGAN [15]	R50	24	56.6
LA [51]	R50	24	58.8
NPID++ [37]	R50	24	59.0
MoCo [26]	R50	24	60.6
PIRL [37]	R50	24	63.6
CPC v2 [28]	R50	24	63.8
PCL [34]	R50	24	65.9
SwAV [9]	R50	24	75.3
Jigsaw [38]	R50	24	45.7
<b>Jigsaw + PSL</b>	R50	24	<b>50.9</b>
Rotation [21]	Rv50w4×	86	47.3
Rotation*	Rv50	24	48.6
<b>Rotation+PSL</b>	Rv50	24	<b>53.3</b>
SimCLR [10]	R50	24	61.9
<b>SimCLR+PSL</b>	R50	24	<b>64.3</b>
MoCov2 [11]	R50	24	67.5
<b>MoCov2+PSL</b>	R50	24	<b>68.1</b>

Table 4. ImageNet accuracy of linear classifiers trained on self-supervised learned representations. All are reported as unsupervised pre-training on ImageNet, followed by supervised linear classification and evaluated on the ImageNet validation set. Note that Rotation [21] uses  $\mathcal{R}_2$  as the transformation set while Rotation\* uses  $\mathcal{R}_3$  as the transformation set. SimCLR results are obtained by 200 training epochs with batchsize 256.

# End of Chapter 3