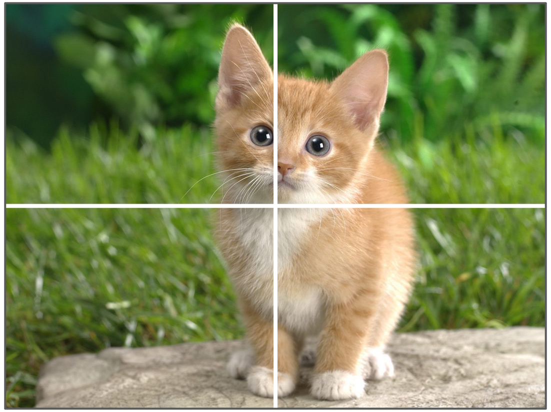
SOTA in Computer Vision

Part 3

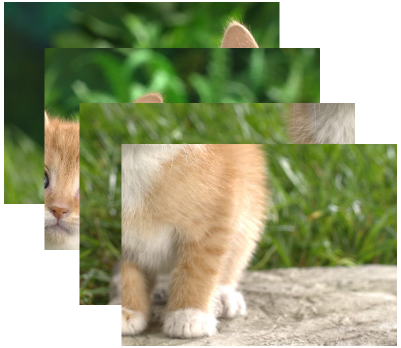
Core idea: Tokenized image

Split image to patches and consider them as tokens (just like words in a sentence) to make prediction

Image



Patches

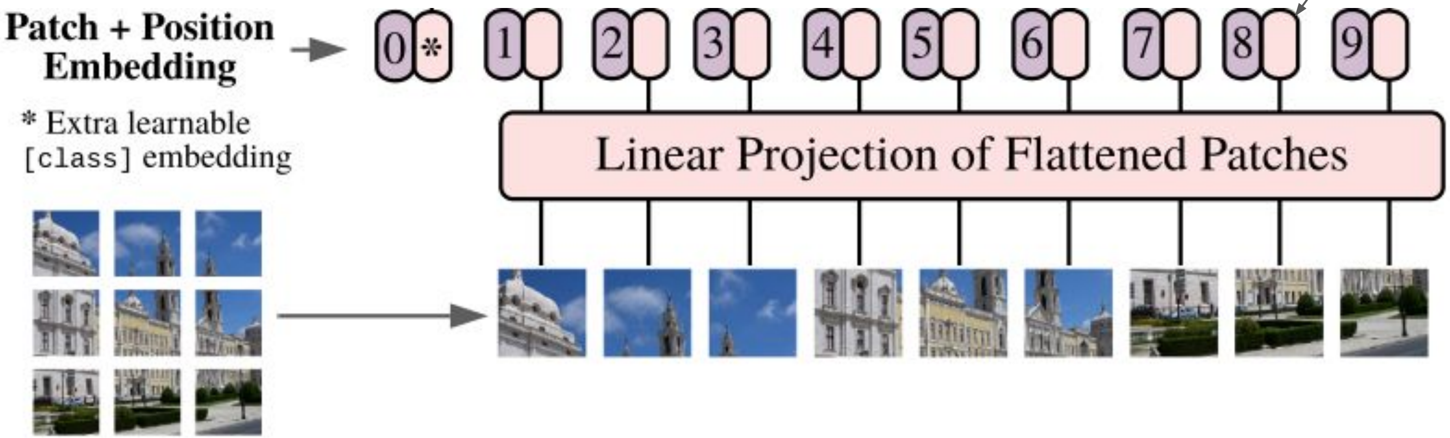


Tokens

Vision Transformer (ViT)

A special learnable [class] token is appended to represent the token of class prediction

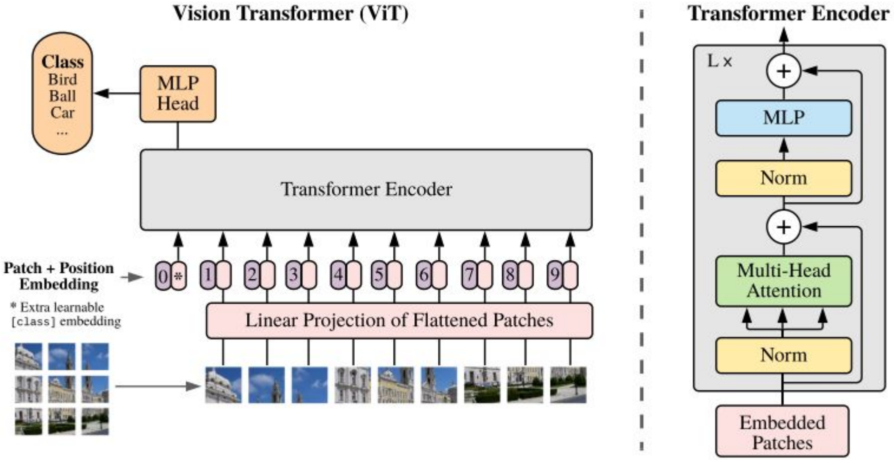
Image tokens get flatten and through an embedding



Vision Transformer (ViT)

Then all tokens get through a Transformer encoder.

After that, an extra MLP is applied to the [class] token to get final prediction



Vision Transformer (ViT)

Variants

Model	Layers	Hidden size	MLP size	Heads	Params
ViT-Base (ViT-B)	12	768	3072	12	86M
ViT- Large (ViT-L)	24	1024	4096	16	307M
ViT-Huge (ViT-H)	32	1280	5120	16	632M

Vision Transformer (ViT)

Top-1 Accuracy on ImageNet dataset. ViT models are pre-trained with JFT-300M dataset before fine-tuning on ImageNet. Naming rule: ViT-<variant>/<patch_size>

ViT-H/14	**ViT-L/16**
88.55	87.76

Class-Attention in Image Transformers (CaiT)

Two improvements:

- **LayerScale:** Stable optimization for deeper vision transformers
- **Class Attention:** Specialize layers during optimization

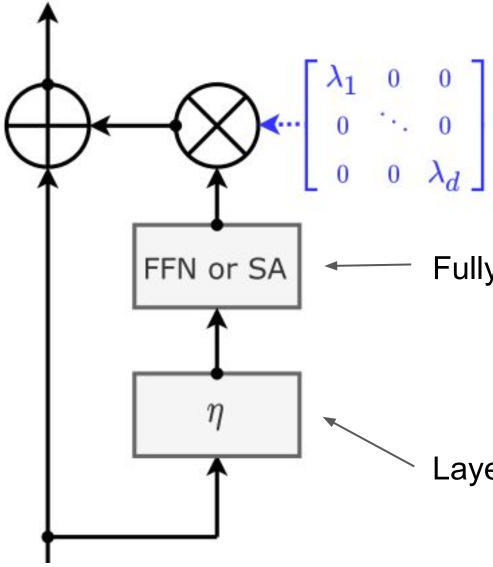
Class-Attention in Image Transformers (CaiT)

Why LayerScale?

By experiments, ViT does not perform well as depth increases.

Class-Attention in Image Transformers (CaiT)

LayerScale: Perform per-channel scaling on residual output. Scale parameters are learnable. Can think of it as a weight normalization method, hence stable the gradient and the training



Per-channel scaling, learnable

Fully-connected Network or Self Attention

LayerNorm previously used by ViT

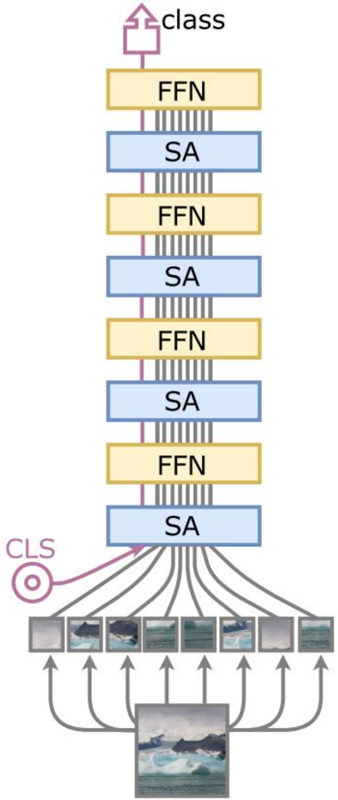
Class-Attention in Image Transformers (CaiT)

Why Class Attention?

As seen in ViT, the [class] token is appended at the start of the Transformer Encoders.

Hence, the learned weights are asked to optimize two contradictory objectives:

1. Guiding the self-attention between patches
2. Summarizing the information useful to the linear classifier

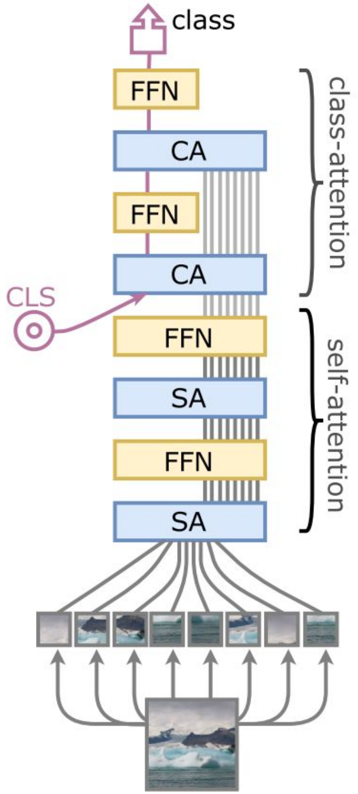


Class-Attention in Image Transformers (CaiT)

Class Attention

Two stages architecture:

- **1st stage:** Self-attention blocks without [class] token. Prepare the vectors for classifier
- **2nd stage:** [class] token appended. Apply self-attention but only update the [class] token. Focus on predicting



Class-Attention in Image Transformers (CaiT)

Variants

Table 3: CaiT models: The design parameters are depth and d. The mem columns correspond to the memory usage. All models are initially trained at resolution 224 during 400 epochs. We also fine-tune these models at resolution 384 (identified by ↑384) or train them with distillation (Υ). The FLOPs are reported for each resolution.

CAIT model	depth (SA+CA)	d	#params ($\times 10^6$)	FLOPs ($\times 10^9$) @224	FLOPs ($\times 10^9$) @384	Top-1 acc. (%): Imagenet1k-val @224	↑384	@224Υ	↑384Υ
XXS-24	24 + 2	192	12.0	2.5	9.5	77.6	80.4	78.4	80.9
XXS-36	36 + 2	192	17.3	3.8	14.2	79.1	81.8	79.7	82.2
XS-24	24 + 2	288	26.6	5.4	19.3	81.8	83.8	82.0	84.1
XS-36	36 + 2	288	38.6	8.1	28.8	82.6	84.3	82.9	84.8
S-24	24 + 2	384	46.9	9.4	32.2	82.7	84.3	83.5	85.1
S-36	36 + 2	384	68.2	13.9	48.0	83.3	85.0	84.0	85.4
S-48	48 + 2	384	89.5	18.6	63.8	83.5	85.1	83.9	85.3
M-24	24 + 2	768	185.9	36.0	116.1	83.4	84.5	84.7	85.8
M-36	36 + 2	768	270.9	53.7	173.3	83.8	84.9	85.1	86.1

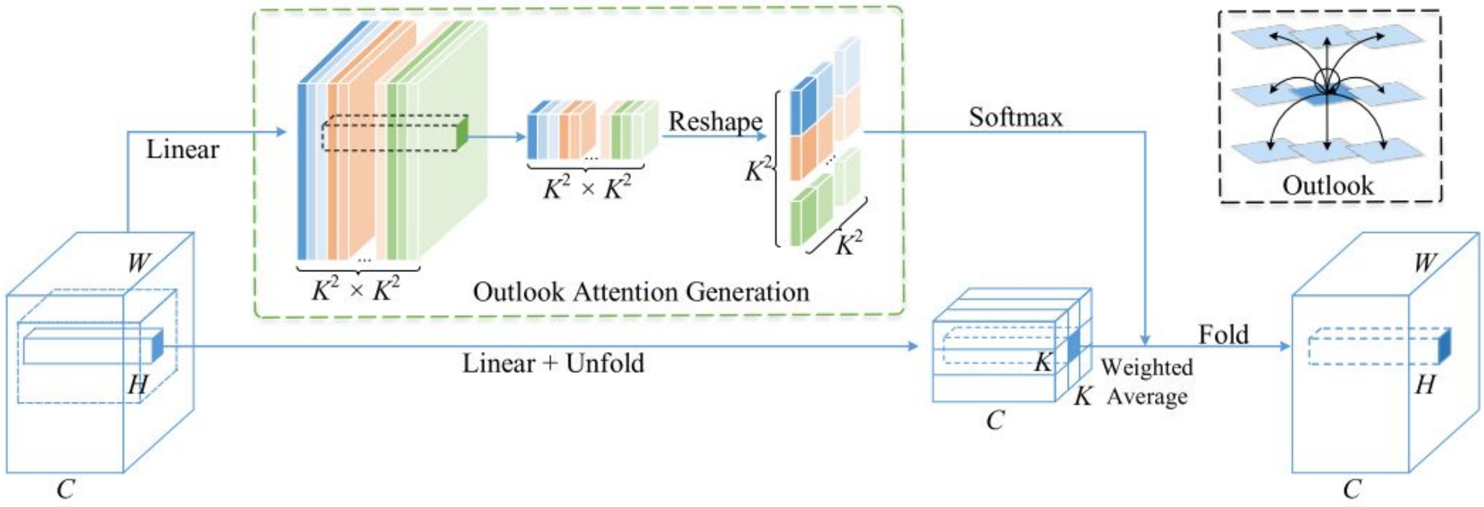
Vision Outlooker (VOLO)

Why Vision Outlooker?

Authors argue that ViT cannot out-performed CNN due to low efficacy in encoding fine-level features and context in token representation

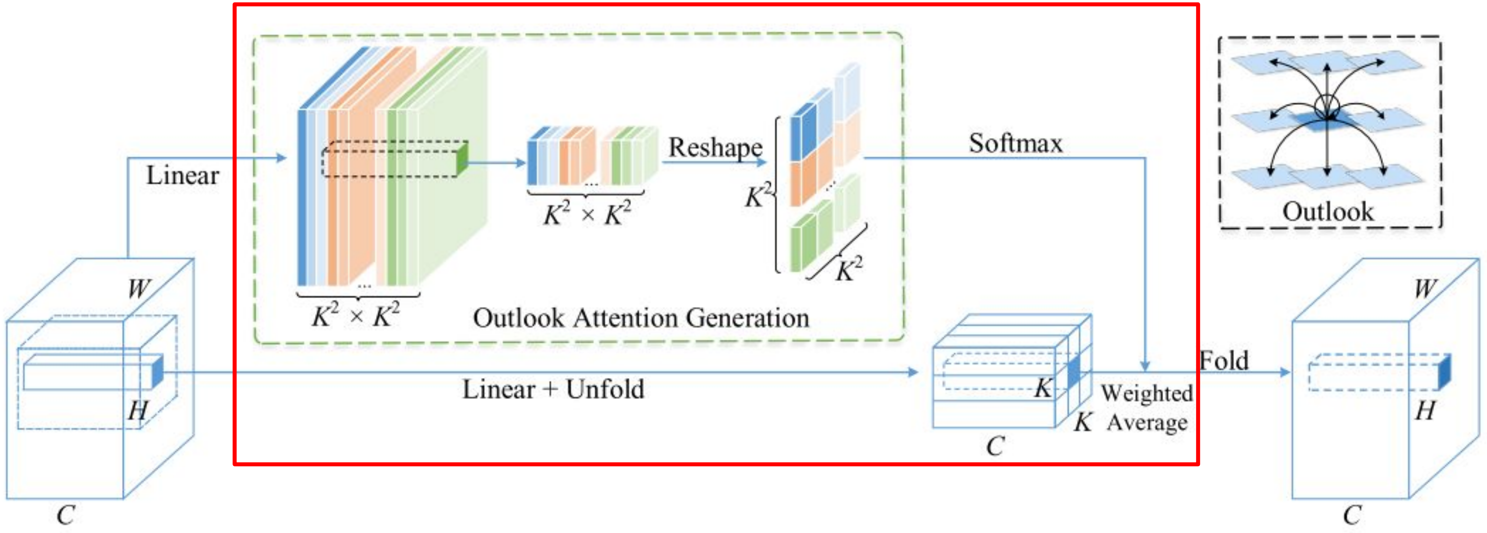
Vision Outlooker (VOLO)

Outlooker use information in K x K window to generate its representation



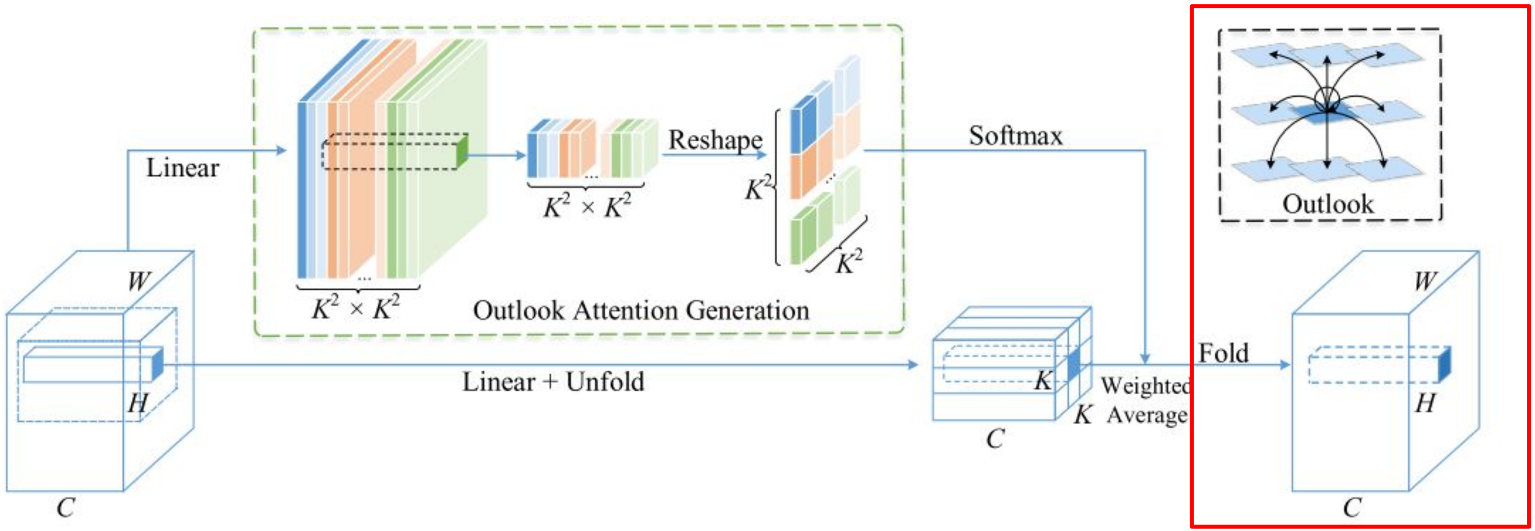
Vision Outlooker (VOLO)

For each position (token), generate a K^2 x K^2 attention map, meaning an K x K attention map for each neighbor. Then calculate weighted average value. Finally, we got K x K x C tensor



Vision Outlooker (VOLO)

But each neighbor also has its own KxKxC tensors, these tensors also have the weighted average value generated by the neighbor. So we have to sum all the values from the neighbors according to the relative position. This is called **folding**



Vision Outlooker (VOLO)

Network Architecture:

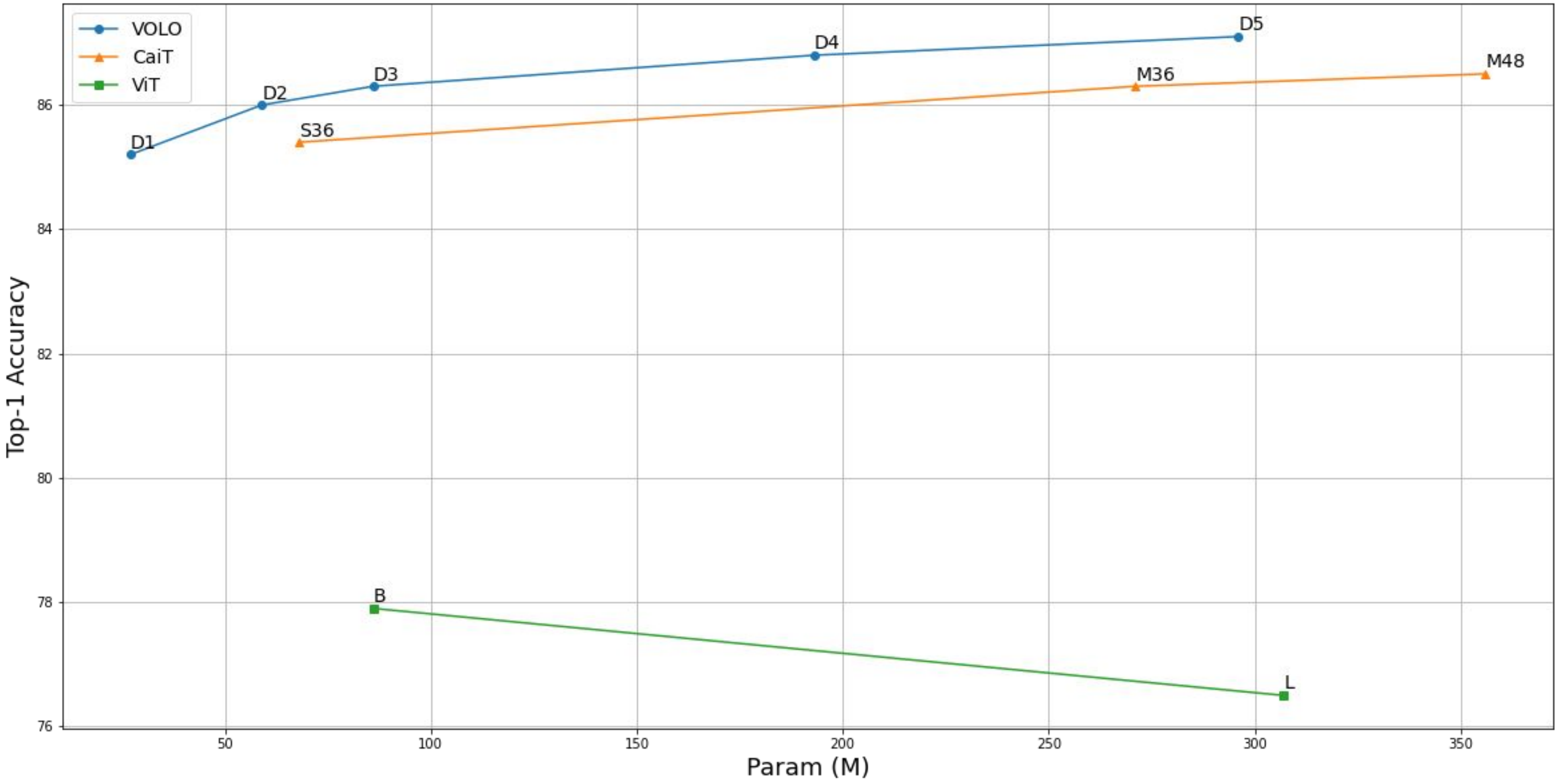
- 1st stage: Stack of Outlook Attentions for patch tokens
- 2nd stage: Stack of Self-Attention

Vision Outlooker (VOLO)

Variants:

Specification	VOLO-D1	VOLO-D2	VOLO-D3	VOLO-D4	VOLO-D5
Patch Embedding	8×8	8×8	8×8	8×8	8×8
Stage 1 (28×28)	$\begin{bmatrix} \text{head: 6, stride: 2} \\ \text{kernel: } 3 \times 3 \\ \text{mlp: 3, dim: 192} \end{bmatrix} \times 4$	$\begin{bmatrix} \text{head: 8, stride: 2} \\ \text{kernel: } 3 \times 3 \\ \text{mlp: 3, dim: 256} \end{bmatrix} \times 6$	$\begin{bmatrix} \text{head: 8, stride: 2} \\ \text{kernel: } 3 \times 3 \\ \text{mlp: 3, dim: 256} \end{bmatrix} \times 8$	$\begin{bmatrix} \text{head: 12, stride: 2} \\ \text{kernel: } 3 \times 3 \\ \text{mlp: 3, dim: 384} \end{bmatrix} \times 8$	$\begin{bmatrix} \text{head: 12, stride: 2} \\ \text{kernel: } 3 \times 3 \\ \text{mlp: 4, dim: 384} \end{bmatrix} \times 12$
Patch Embedding	2×2	2×2	2×2	2×2	2×2
Stage 2 (14×14)	$\begin{bmatrix} \text{\#heads: 12} \\ \text{mlp: 3, dim: 384} \end{bmatrix} \times 14$	$\begin{bmatrix} \text{\#heads: 16} \\ \text{mlp: 3, dim: 512} \end{bmatrix} \times 18$	$\begin{bmatrix} \text{\#heads: 16} \\ \text{mlp: 3, dim: 512} \end{bmatrix} \times 28$	$\begin{bmatrix} \text{\#heads: 16} \\ \text{mlp: 3, dim: 768} \end{bmatrix} \times 28$	$\begin{bmatrix} \text{\#heads: 16} \\ \text{mlp: 4, dim: 768} \end{bmatrix} \times 36$
Total Layers	18	24	36	36	48
Parameters	26.6M	58.7M	86.3M	193M	296M

Comparison



VOLO
CaiT
ViT
D1
D2
D3
D4
D5
S36
M36
M48
B
L
86
84
82
80
78
76
Top-1 Accuracy
50
100
150
200
250
300
350
Param (M)