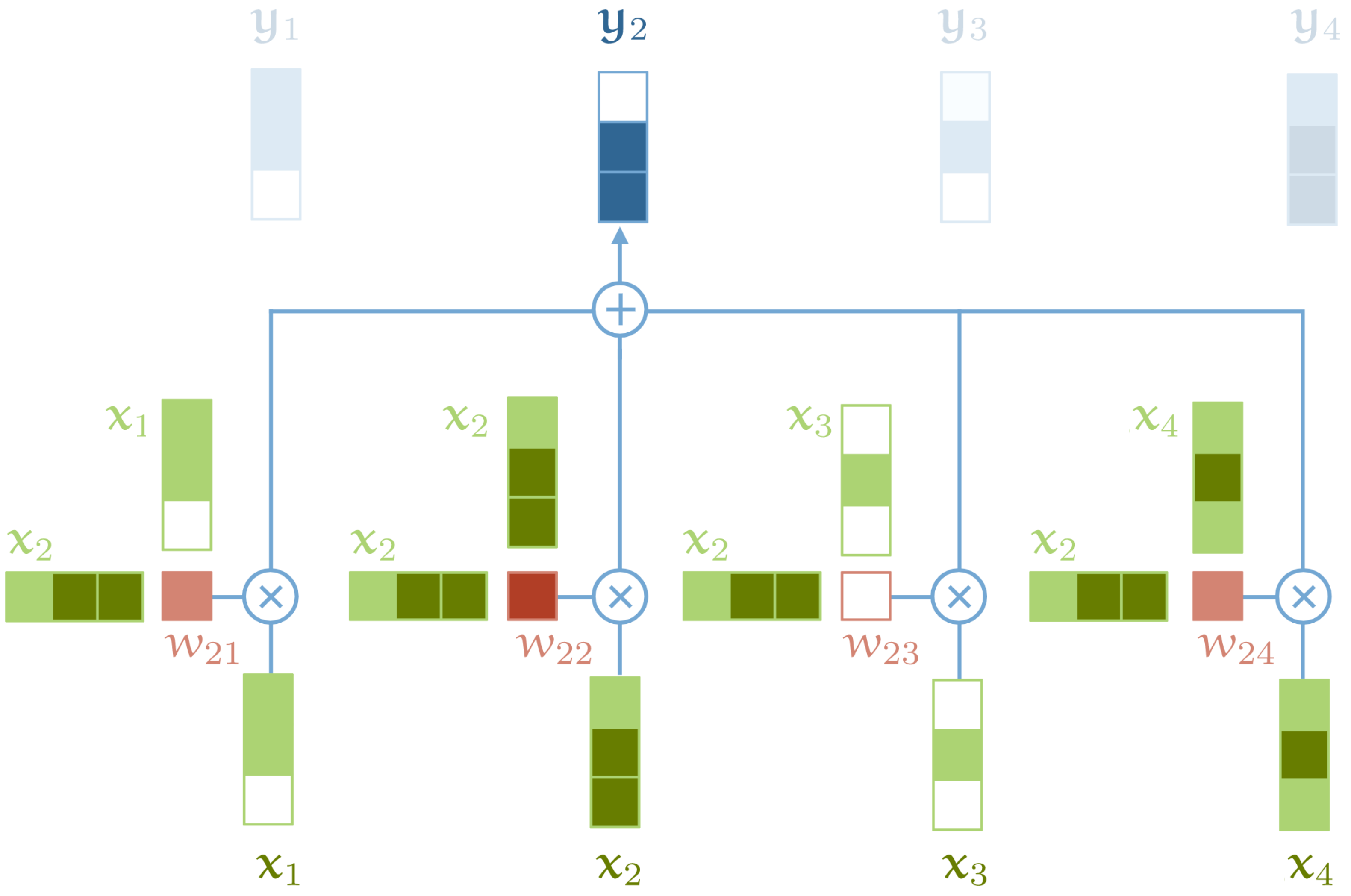
Vision Transformers

Saurabh Gupta

Overview

- Vision Transformers
- Finetuning Vision Transformers
- Multiscale Vision Transformers
- Transformers for Detection

Attention



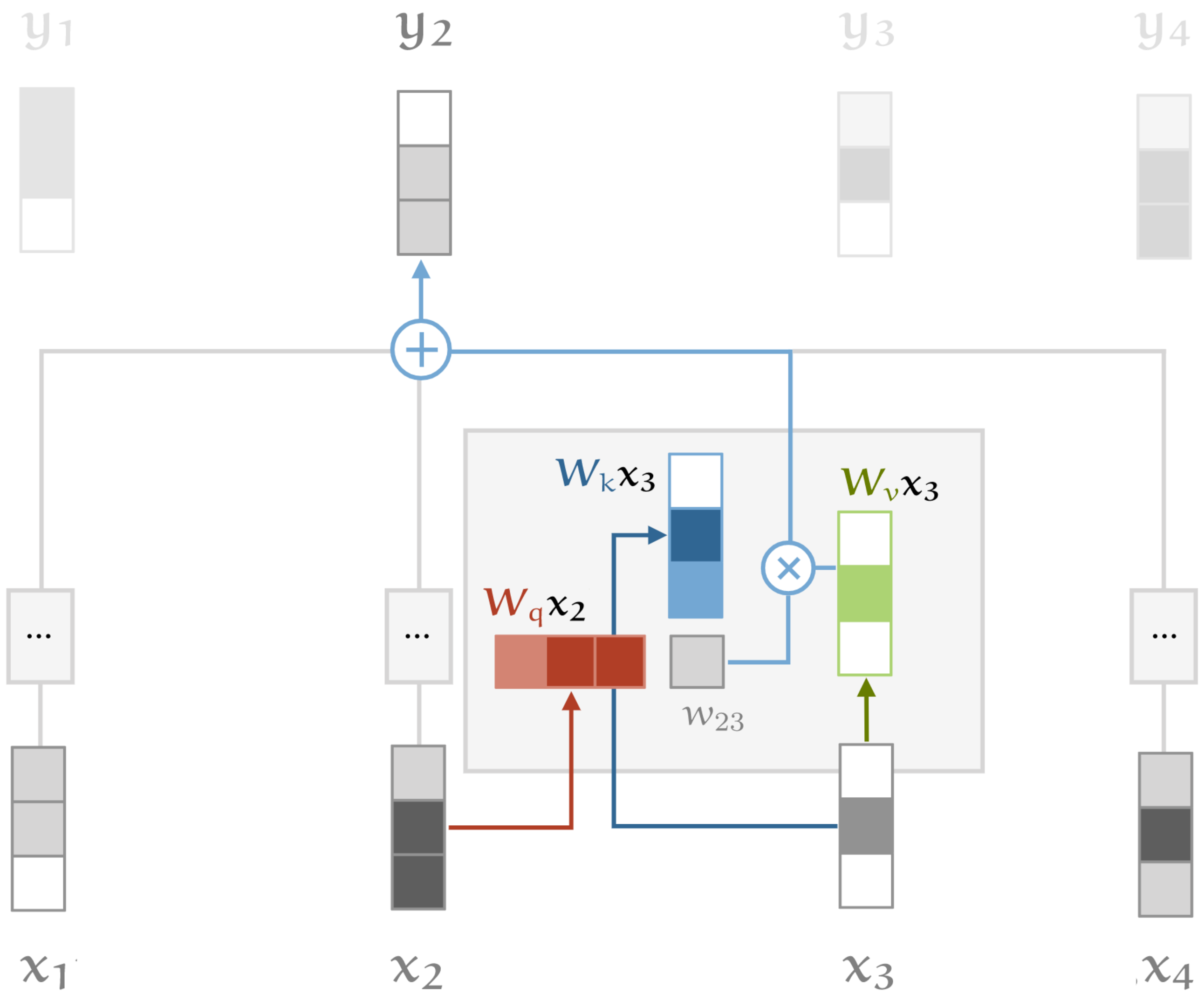
$$y_i = \sum_j w_{ij} x_{ij}$$

$$w_{ij} = softmax_j(x_i^T x_j / \sqrt{d_k})$$

$$w_{ij} = \frac{e^{x_i^T x_j}}{\sum_j e^{x_i^T x_j}}$$

Source: http://peterbloem.nl/blog/transformers See also: [Attention is all you need]()

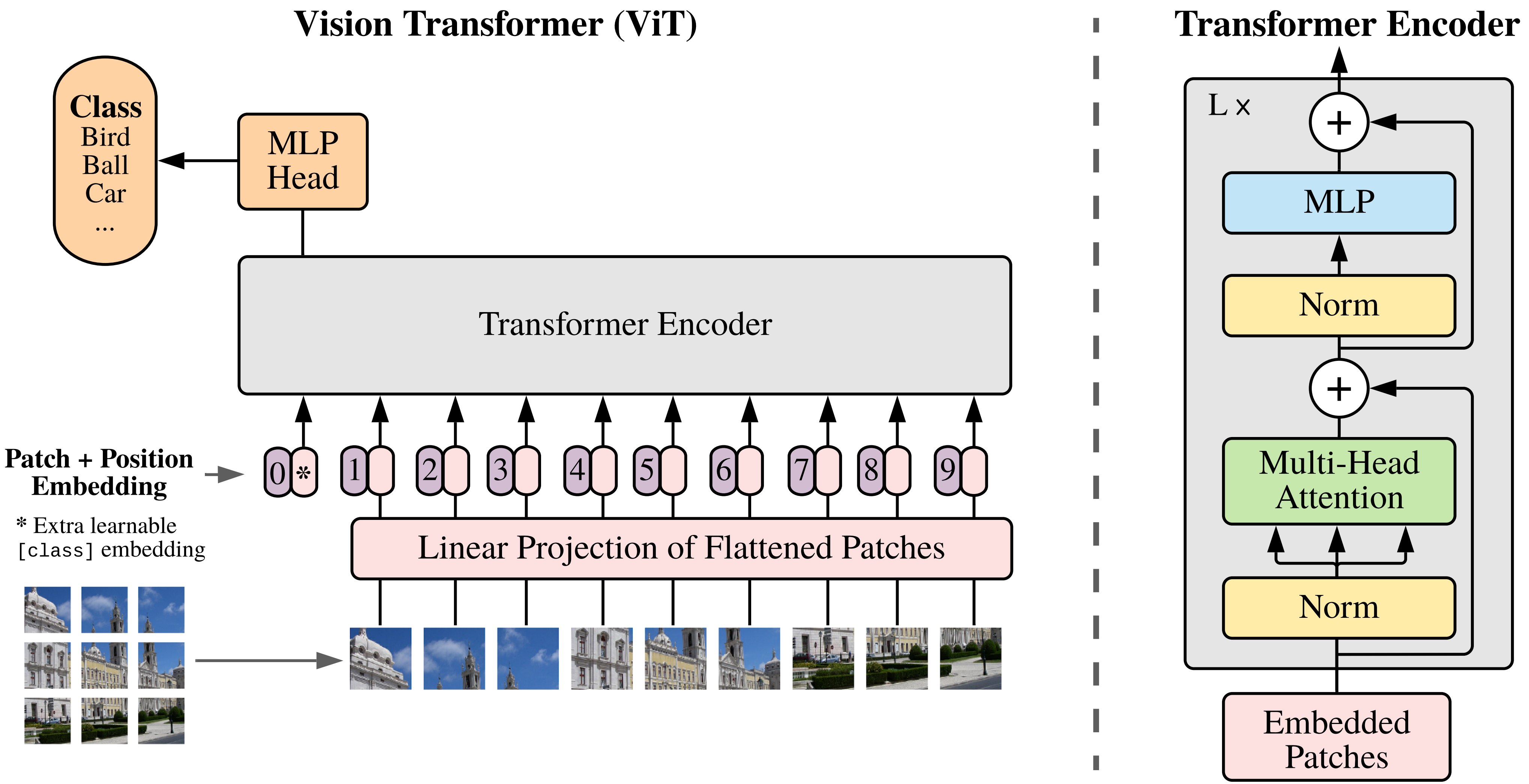
Attention (with key, query and value)



$$\boldsymbol{y_i} = \sum_j w_{ij} \boldsymbol{W_v x_{ij}}$$

$$w_{ij} = softmax_j((\boldsymbol{W_q x_i})^{\boldsymbol{T}} \boldsymbol{W_k x_j} / \sqrt{d_k})$$

Source: http://peterbloem.nl/blog/transformers See also: [Attention is all you need]()



$$
\begin{aligned}
\mathbf{z}_0 &= [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\, \cdots;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, && \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D},\, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D} \\
\mathbf{z'}_\ell &= \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, && \ell = 1 \ldots L \\
\mathbf{z}_\ell &= \text{MLP}(\text{LN}(\mathbf{z'}_\ell)) + \mathbf{z'}_\ell, && \ell = 1 \ldots L \\
\mathbf{y} &= \text{LN}(\mathbf{z}_L^0)
\end{aligned}
$$

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

Multihead Self-Attention

A MULTIHEAD SELF-ATTENTION

Standard **qkv** self-attention (SA, Vaswani et al. (2017)) is a popular building block for neural architectures. For each element in an input sequence $\mathbf{z} \in \mathbb{R}^{N \times D}$, we compute a weighted sum over all values $\mathbf{v}$ in the sequence. The attention weights A_{ij} are based on the pairwise similarity between two elements of the sequence and their respective query $\mathbf{q}^i$ and key $\mathbf{k}^j$ representations.

$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U}_{qkv} \qquad \mathbf{U}_{qkv} \in \mathbb{R}^{D \times 3D_h}, \tag{5}$$

$$A = \mathrm{softmax}\left(\mathbf{q}\mathbf{k}^\top / \sqrt{D_h}\right) \qquad A \in \mathbb{R}^{N \times N}, \tag{6}$$

$$\mathrm{SA}(\mathbf{z}) = A\mathbf{v}\,. \tag{7}$$

Multihead self-attention (MSA) is an extension of SA in which we run k self-attention operations, called “heads”, in parallel, and project their concatenated outputs. To keep compute and number of parameters constant when changing k, D_h (Eq. 5) is typically set to D/k.

$$\mathrm{MSA}(\mathbf{z}) = [\mathrm{SA}_1(z); \mathrm{SA}_2(z); \cdots; \mathrm{SA}_k(z)]\,\mathbf{U}_{msa} \qquad \mathbf{U}_{msa} \in \mathbb{R}^{k \cdot D_h \times D} \tag{8}$$

Other Details

- **MLP:** 1 hidden layer with GELU non-linearity
- **Layer Norm:** Normalize representation for each token to be normalized to zero mean, unit variance
- Learn a **Positional Embedding** for patch locations: $W_{pos} l_{onehot}$

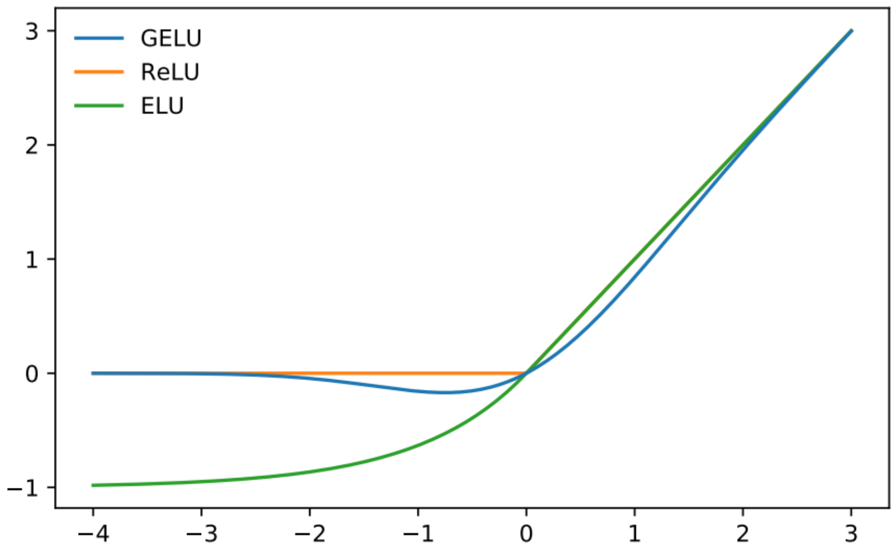


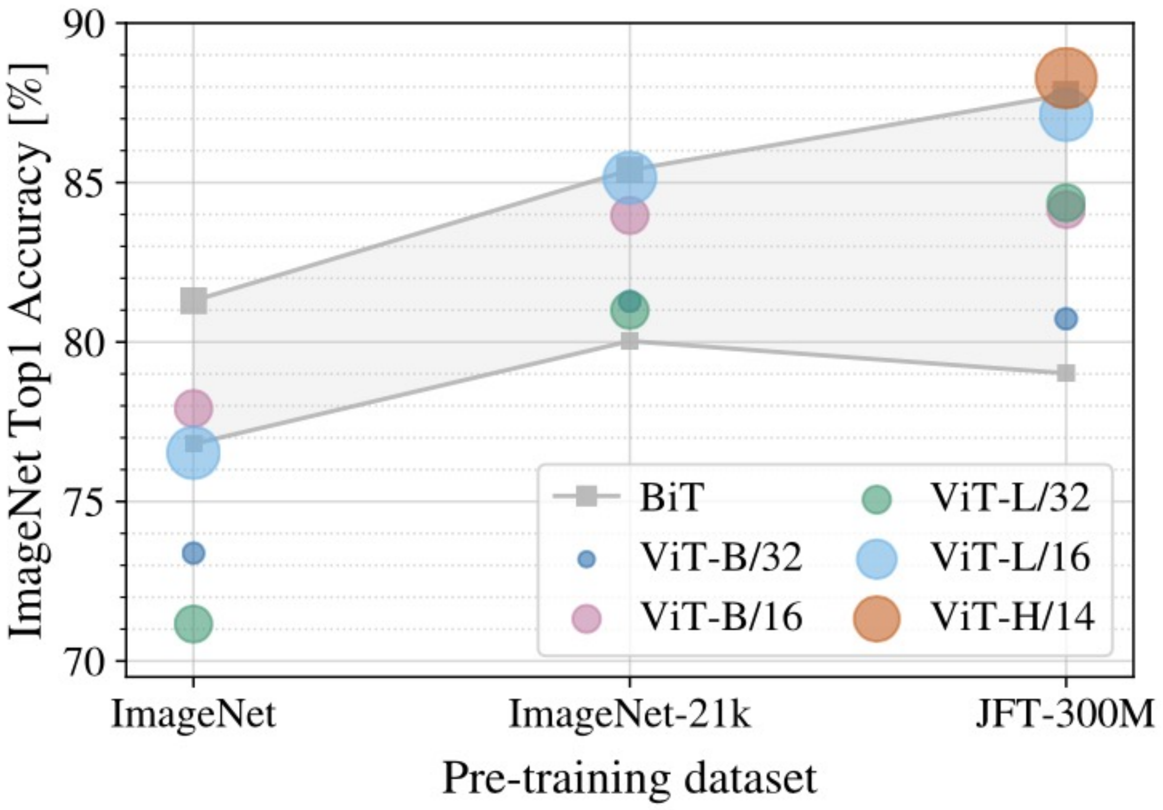
Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

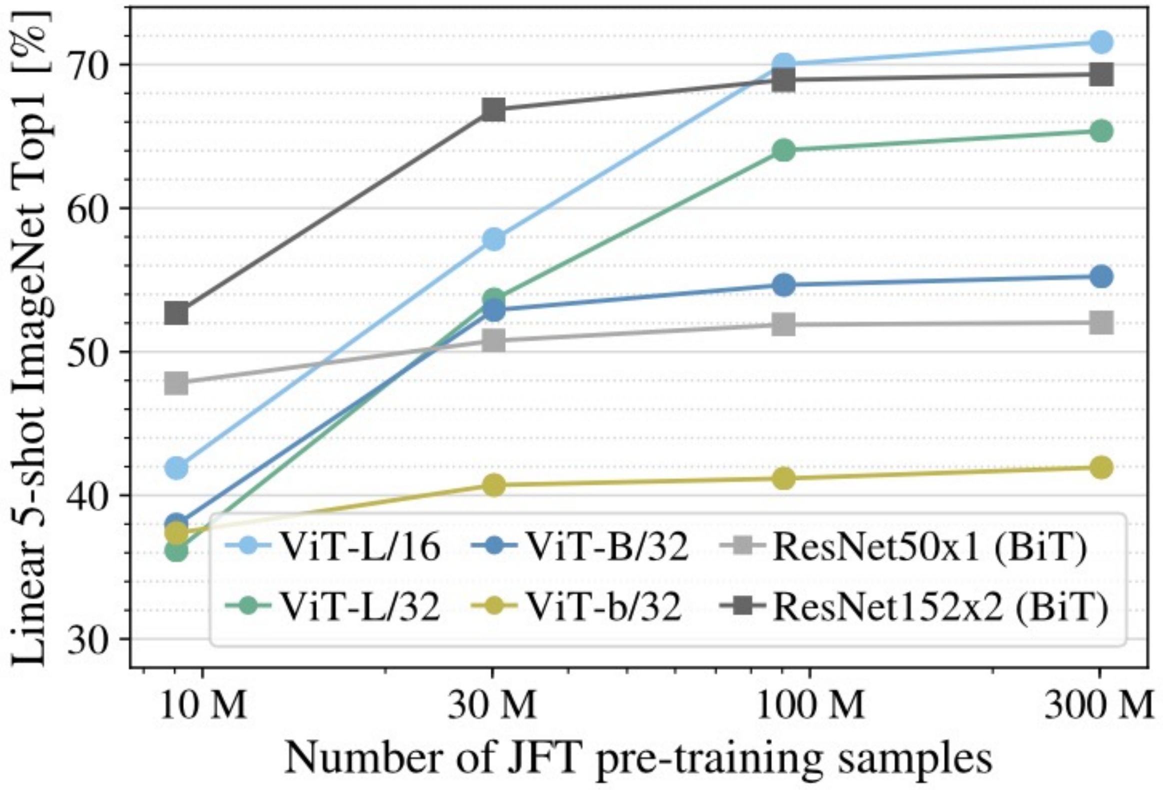
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Results

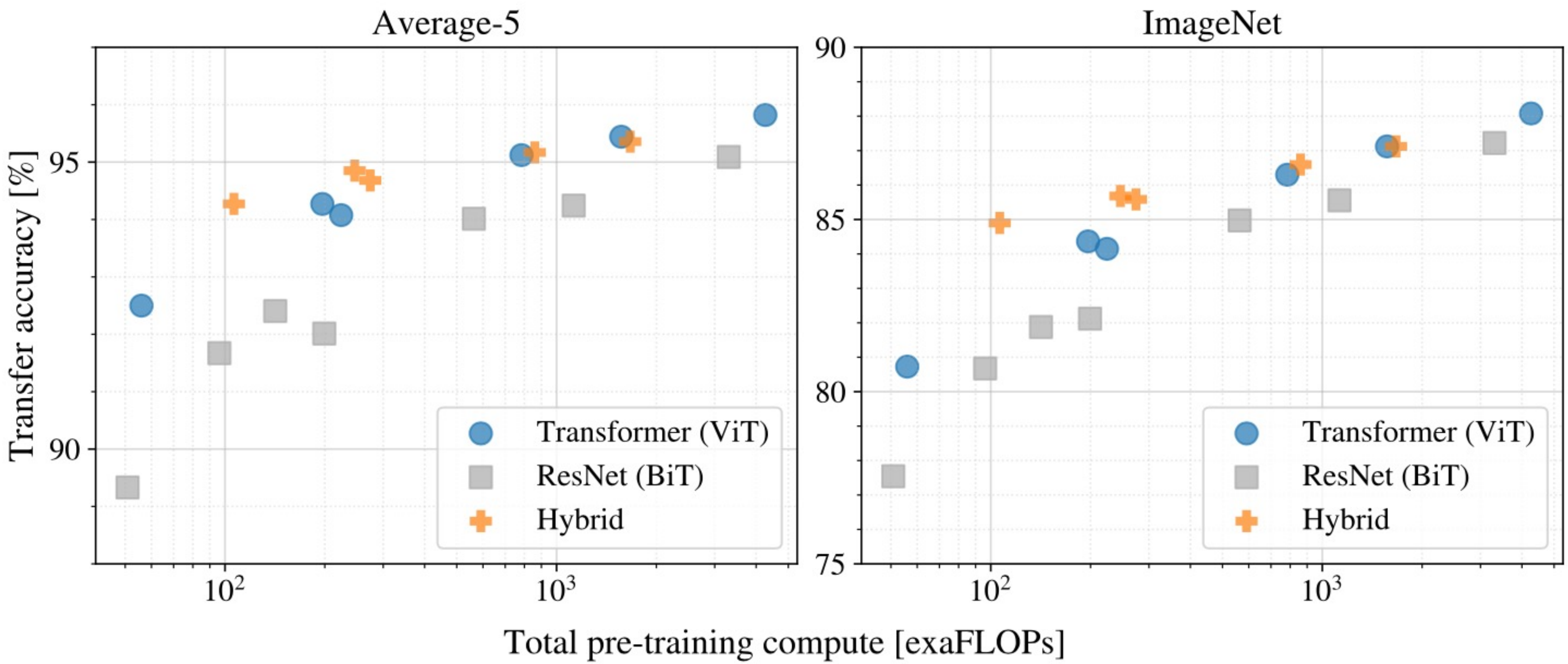
	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** $\pm$ 0.04	87.76 $\pm$ 0.03	85.30 $\pm$ 0.02	87.54 $\pm$ 0.02	88.4/88.5*
ImageNet ReaL	**90.72** $\pm$ 0.05	90.54 $\pm$ 0.03	88.62 $\pm$ 0.05	90.54	90.55
CIFAR-10	**99.50** $\pm$ 0.06	99.42 $\pm$ 0.03	99.15 $\pm$ 0.03	99.37 $\pm$ 0.06	—
CIFAR-100	**94.55** $\pm$ 0.04	93.90 $\pm$ 0.05	93.25 $\pm$ 0.05	93.51 $\pm$ 0.08	—
Oxford-IIIT Pets	**97.56** $\pm$ 0.03	97.32 $\pm$ 0.11	94.67 $\pm$ 0.15	96.62 $\pm$ 0.23	—
Oxford Flowers-102	99.68 $\pm$ 0.02	**99.74** $\pm$ 0.00	99.61 $\pm$ 0.02	99.63 $\pm$ 0.03	—
VTAB (19 tasks)	**77.63** $\pm$ 0.23	76.28 $\pm$ 0.46	72.72 $\pm$ 0.21	76.29 $\pm$ 1.70	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Scale Better with More Data

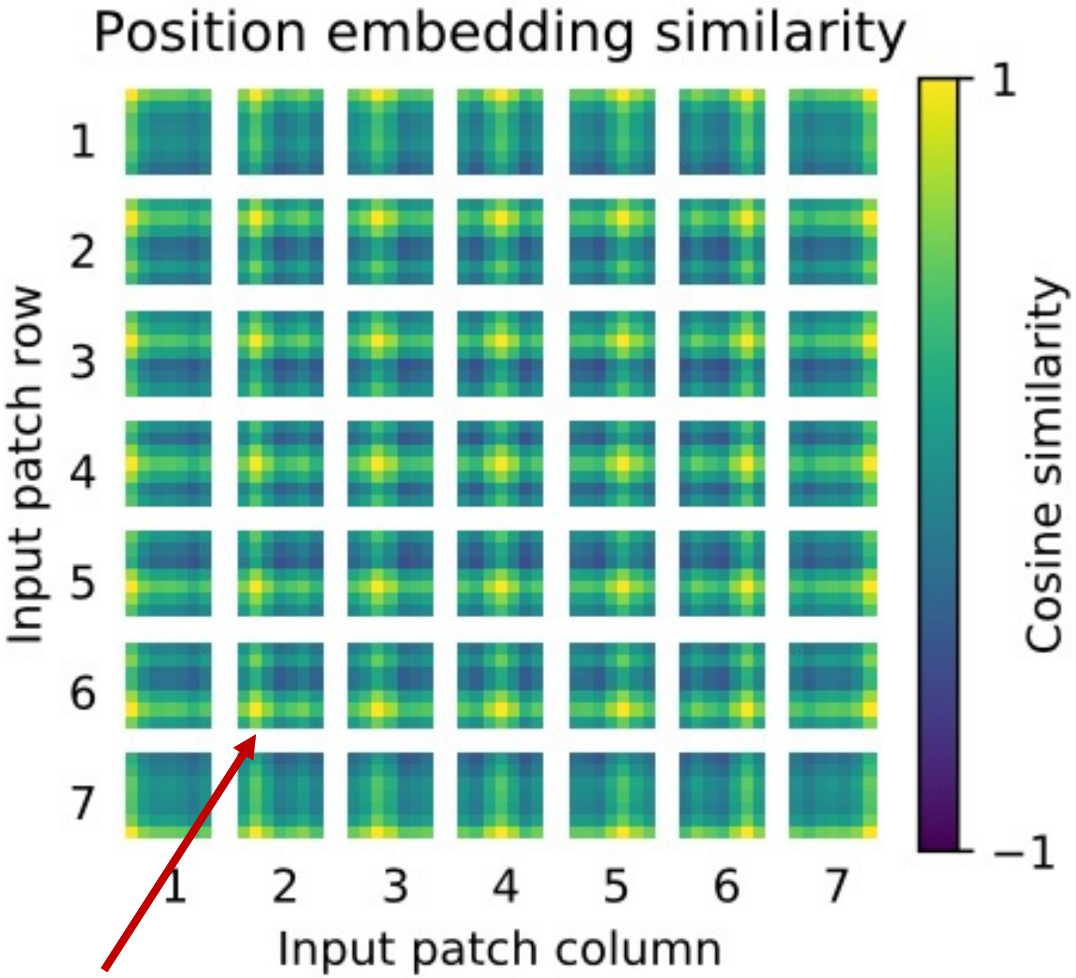




Surprisingly, faster than ResNets to train

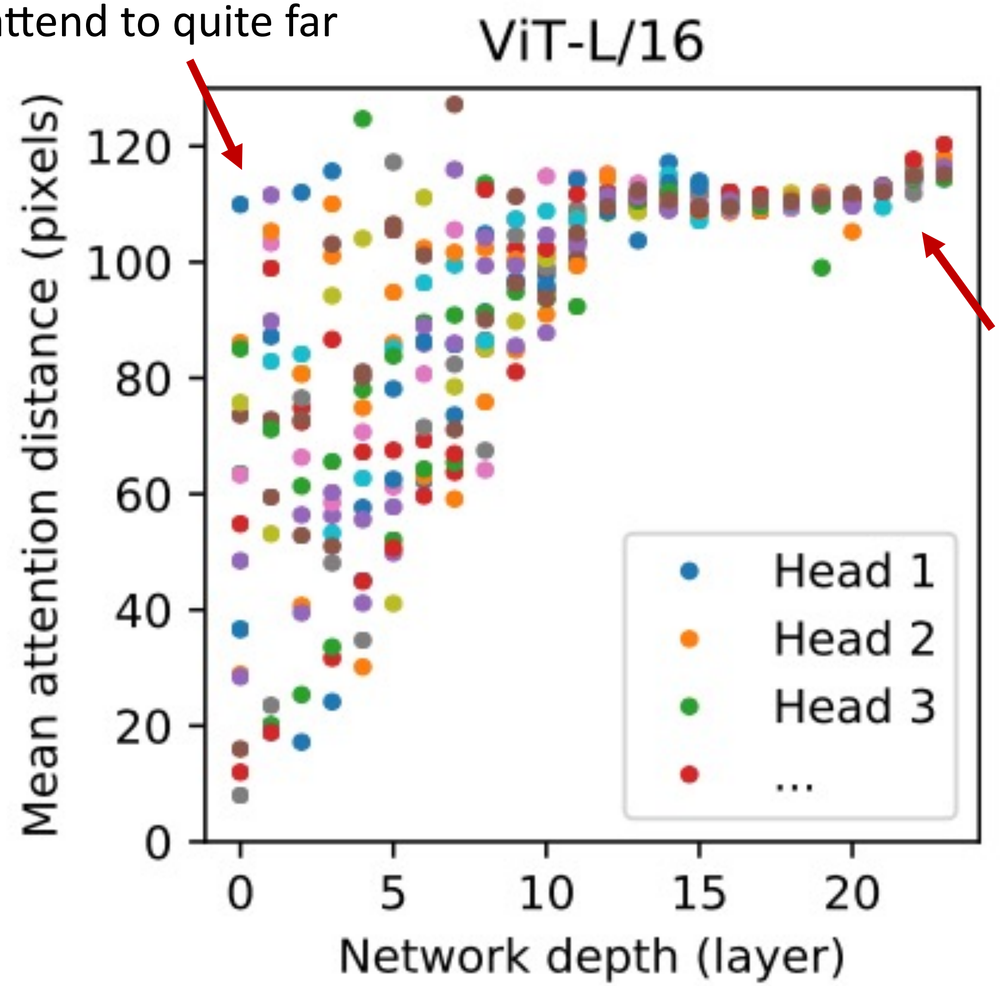


Visualization



Position encoding automatically learn spatial proximity

Some early layers attend locally, others attend to quite far



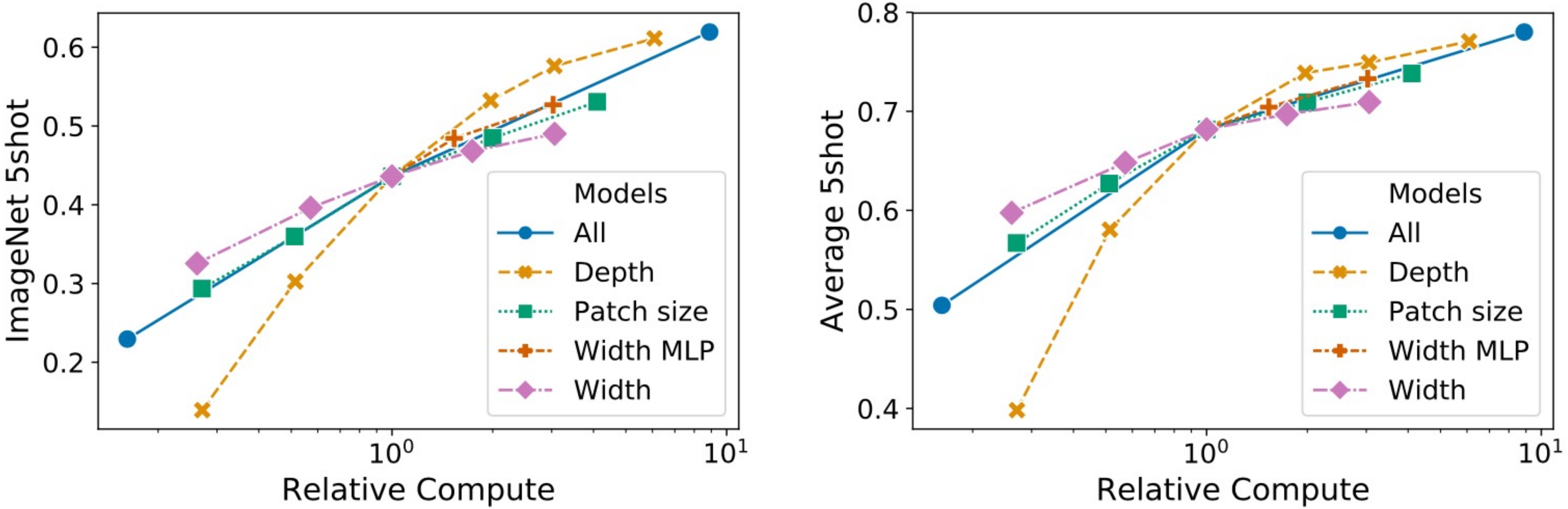
Later layers attend to quite far!

Positional Embedding

- Bag of word is insufficient
- Many encodings work well

Pos. Emb.	Default/Stem	Every Layer	Every Layer-Shared
No Pos. Emb.	0.61382	N/A	N/A
1-D Pos. Emb.	0.64206	0.63964	0.64292
2-D Pos. Emb.	0.64001	0.64046	0.64022
Rel. Pos. Emb.	0.64032	N/A	N/A

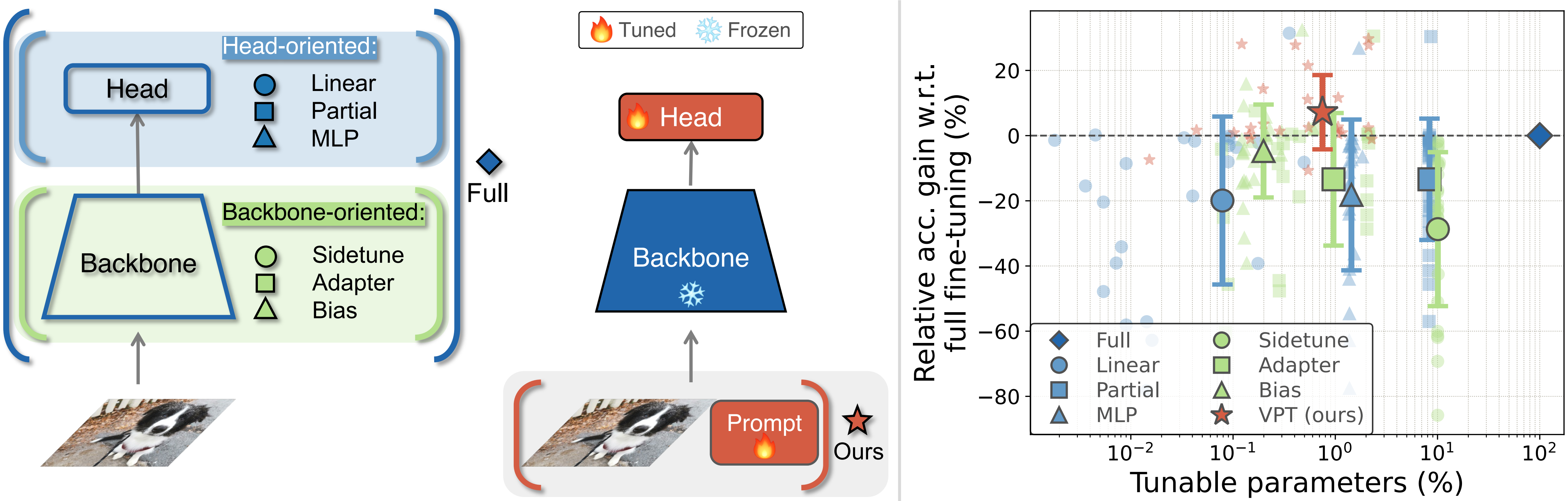
Scaling depth is most effective at current operating point



Finetuning at Higher-Resolution

- Often beneficial to fine-tune at higher
- Keep patch-size same, increase number of patches
- ViT can in-principle handle longer sequence lengths
- Except, positional encodings need to be interpolated.

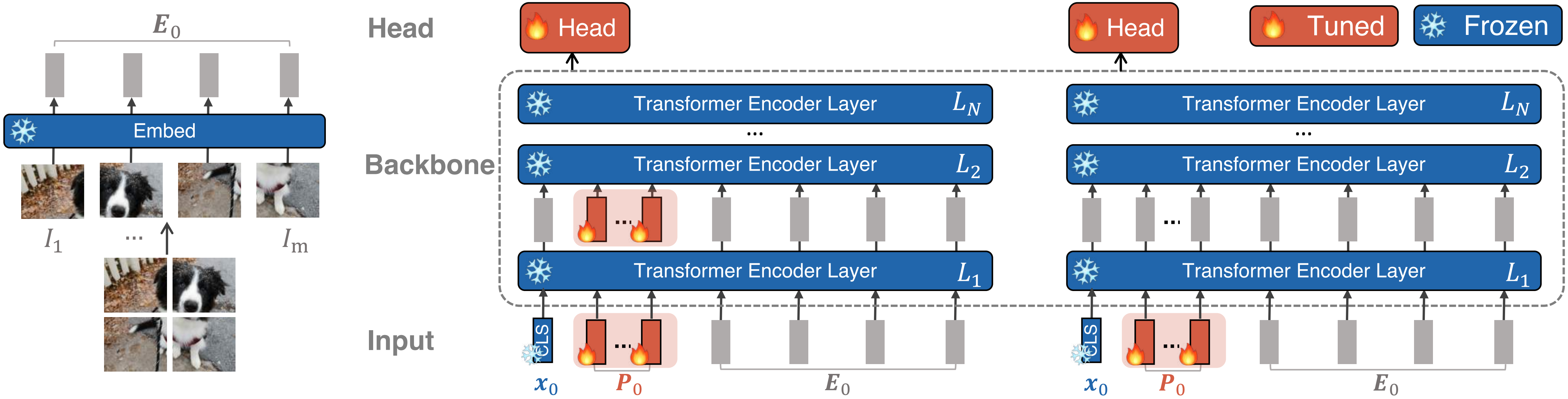
How to finetune Transformer architectures?



(a) Existing tuning protocols (b) Visual-Prompt Tuning (VPT) (c) Results on visual classification tasks

[Visual Prompt Tuning](), Jia et al. ECCV 2022

How to finetune Transformer architectures?

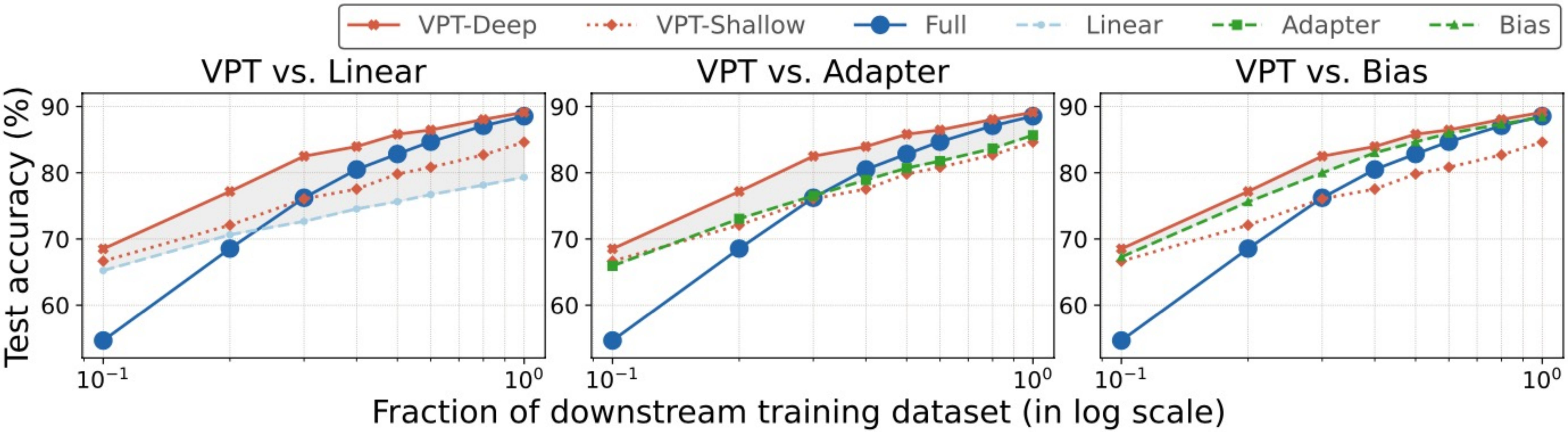


(a) Visual-Prompt Tuning: Deep (b) Visual-Prompt Tuning: Shallow

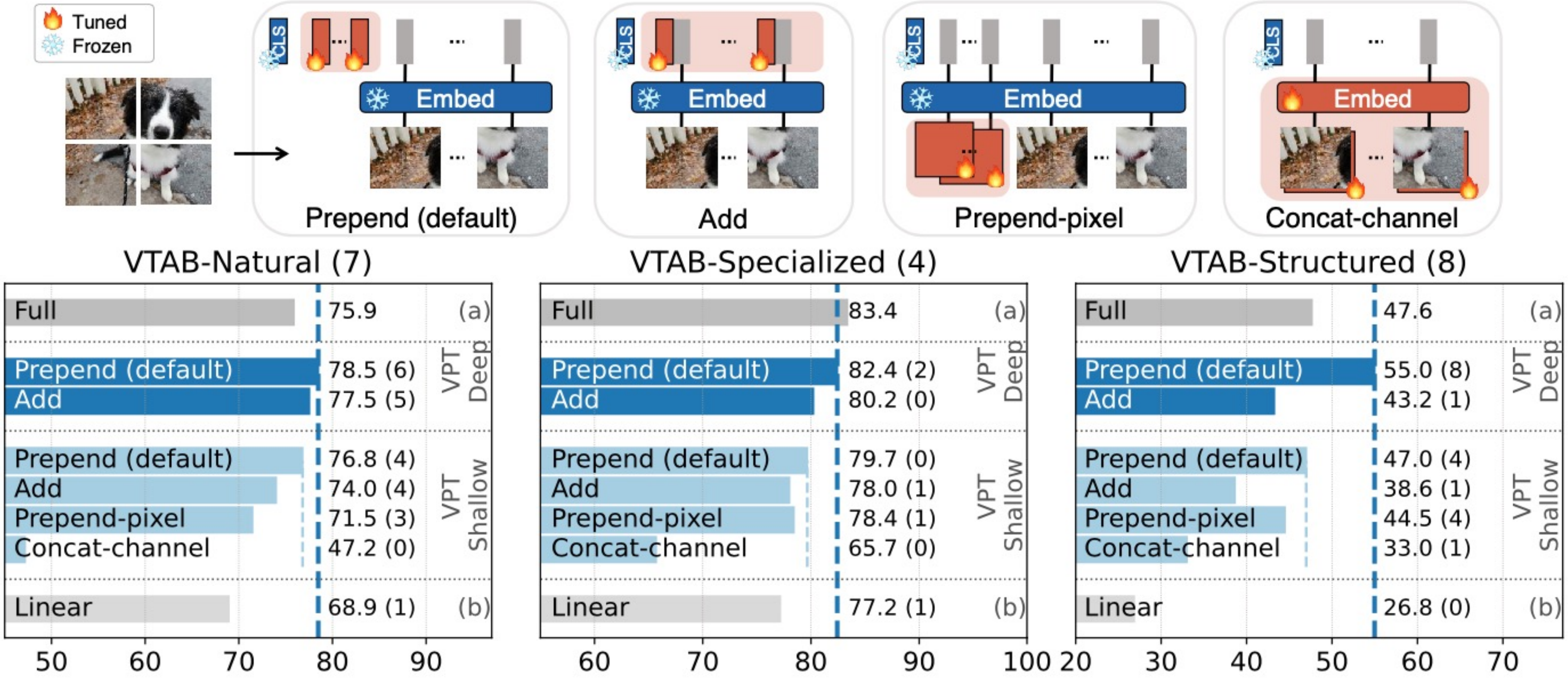
Datasets

- VTAB Dataset
  - Collection of 19 diverse visual classification tasks from 3 groups:
    - **Natural:** natural images captured using standard cameras
    - **Specialized:** such as medical and satellite imagery
    - **Structured:** geometric comprehension like object counting.
  - Each task of VTAB contains 1000 training examples.
- FGVC Dataset
  - 5 benchmarked Fine-Grained Visual Classification tasks
    - CUB-200-2011 (birds), NABirds, Oxford Flowers, Stanford Dogs, Stanford Cars

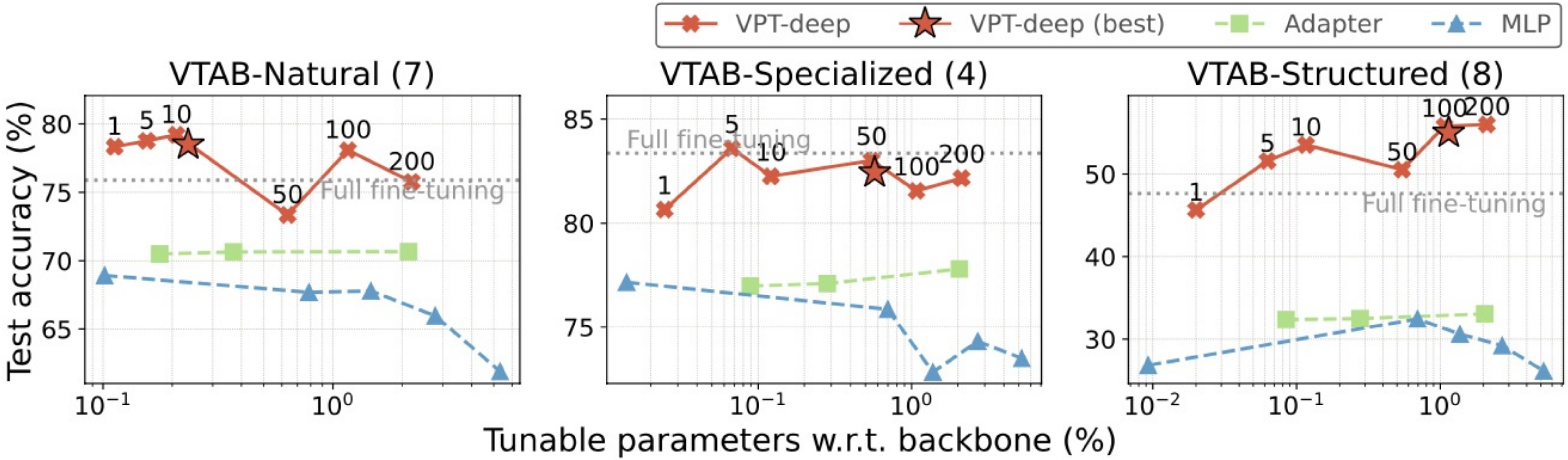
Outperforms CNN finetuning methods, deep better than shallow



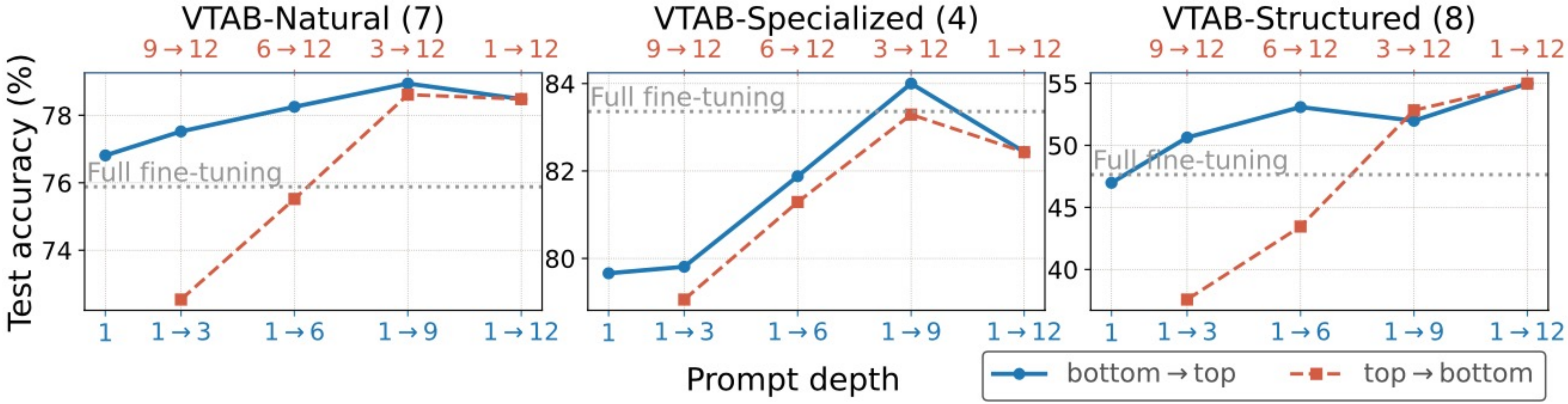
Appending tokens is best



May need to add many tokens



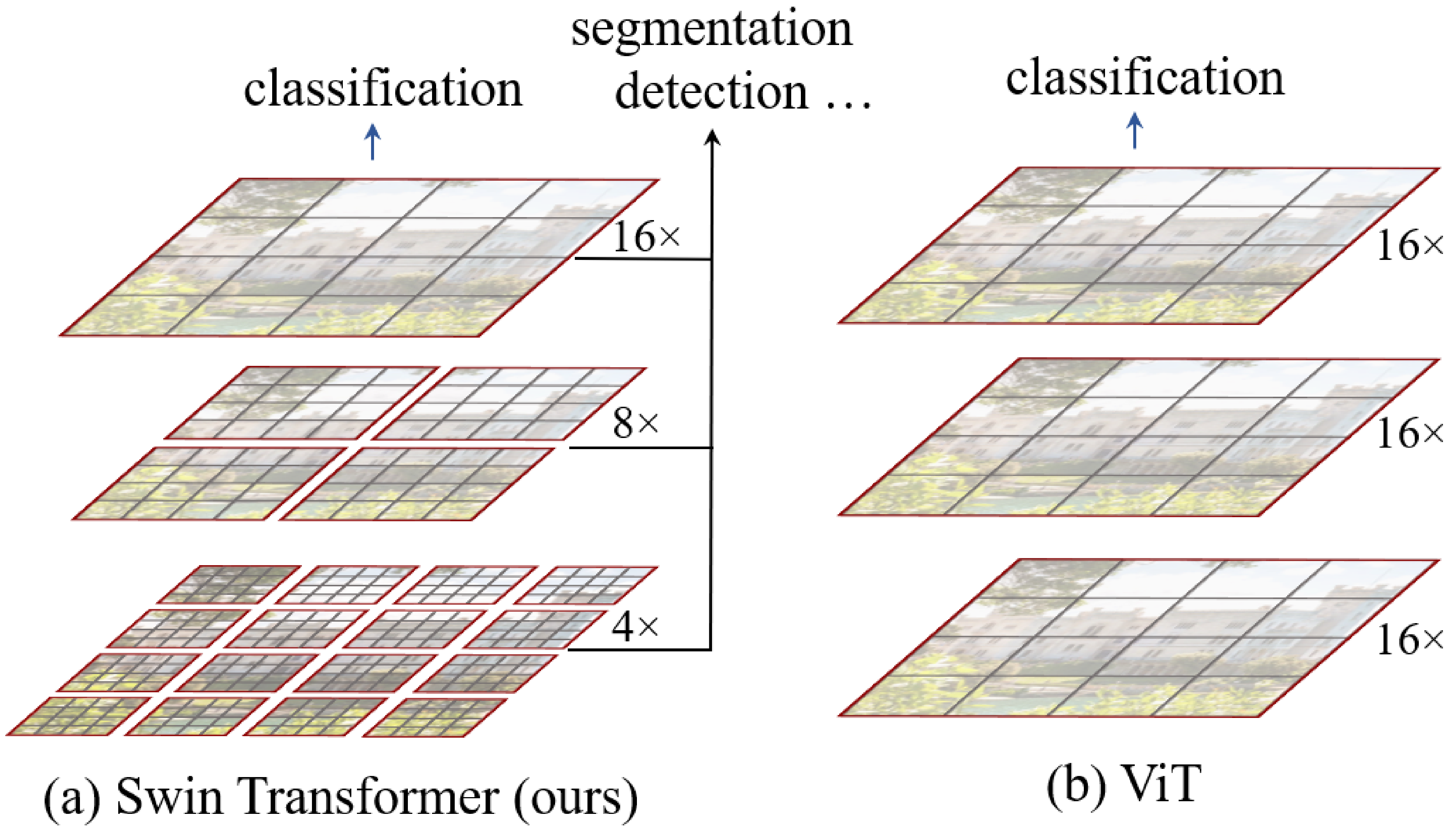
Tokens in early layers is better than later layers



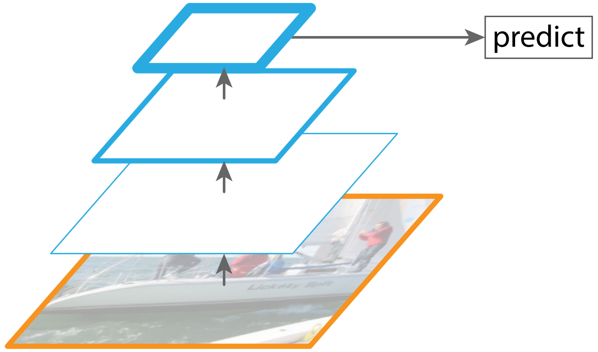
Also applicable to CNNs!

		ConvNeXt-Base (87.6M)				ResNet-50 (23.5M)			
		Total params	VTAB-1k Natural	VTAB-1k Specialized	VTAB-1k Structured	Total params	VTAB-1k Natural	VTAB-1k Specialized	VTAB-1k Structured
	Total # of tasks		7	4	8		7	4	8
(a)	Full	19.01×	77.97	83.71	60.41	19.08×	59.72	76.66	54.08
(b)	Linear	1.01×	74.48 (5)	81.50 (0)	34.76 (**1**)	1.08×	63.75 (**6**)	77.60 (**3**)	30.96 (0)
	Partial-1	2.84×	73.76 (4)	81.64 (0)	39.55 (0)	4.69×	64.34 (**6**)	**78.64** (2)	**45.78** (**1**)
	Mlp-3	1.47×	73.78 (5)	81.36 (**1**)	35.68 (**1**)	7.87×	61.79 (**6**)	70.77 (1)	33.97 (0)
(c)	Bias	1.04×	69.07 (2)	72.81 (0)	25.29 (0)	1.10×	63.51 (**6**)	77.22 (2)	33.39 (0)
(ours)	Visual-Prompt Tuning	1.02×	**78.48** (**6**)	**83.00** (**1**)	**44.64** (**1**)	1.09×	**66.25** (**6**)	77.32 (2)	37.52 (0)

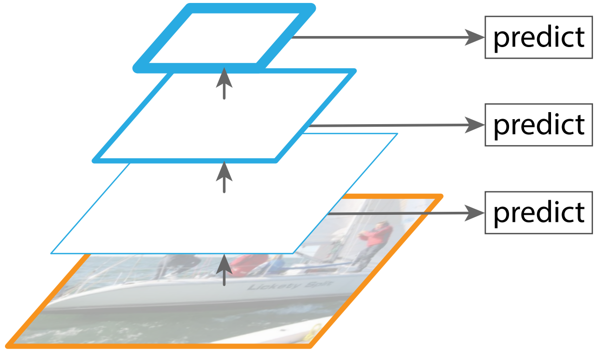
All representations are at the same scale



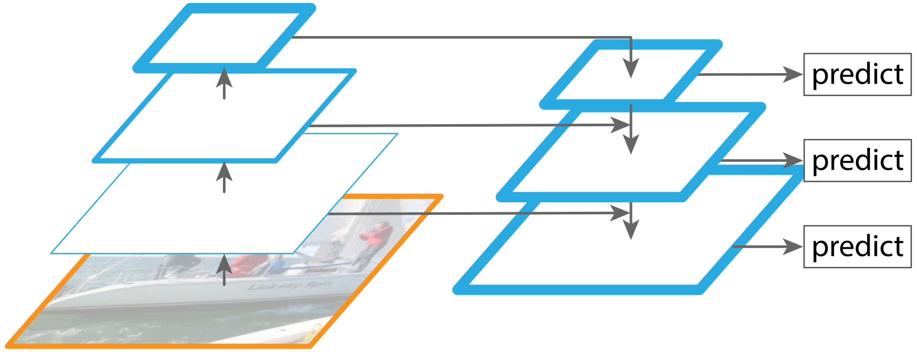
FPNs for CNN Object Detectors



(b) Single feature map



(c) Pyramidal feature hierarchy



(d) Feature Pyramid Network

Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(*) baseline from He *et al.* [16][†]	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	*2fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

SWin Transformer (Patch Merging)

- Merge 2x2 neighboring patches
- Apply linear layer on the 4C-dimensional concatenated features (no pooling?)
- Reduces number of tokens by a factor of 4
- Output channels is set to 2C

SWin Transformer

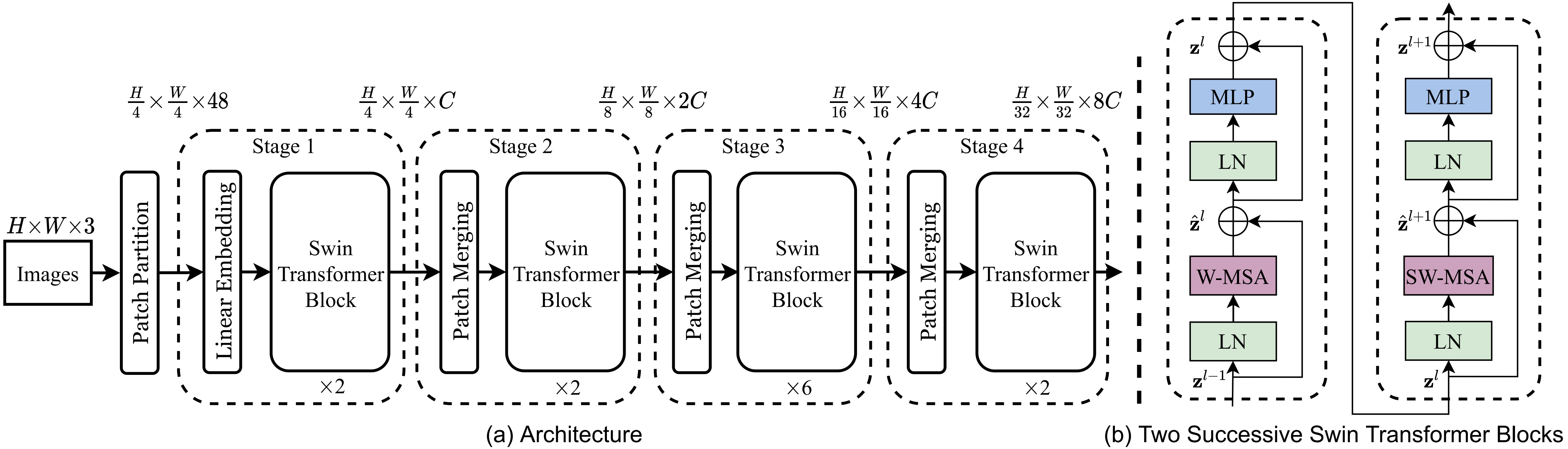


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

SWin Transformer

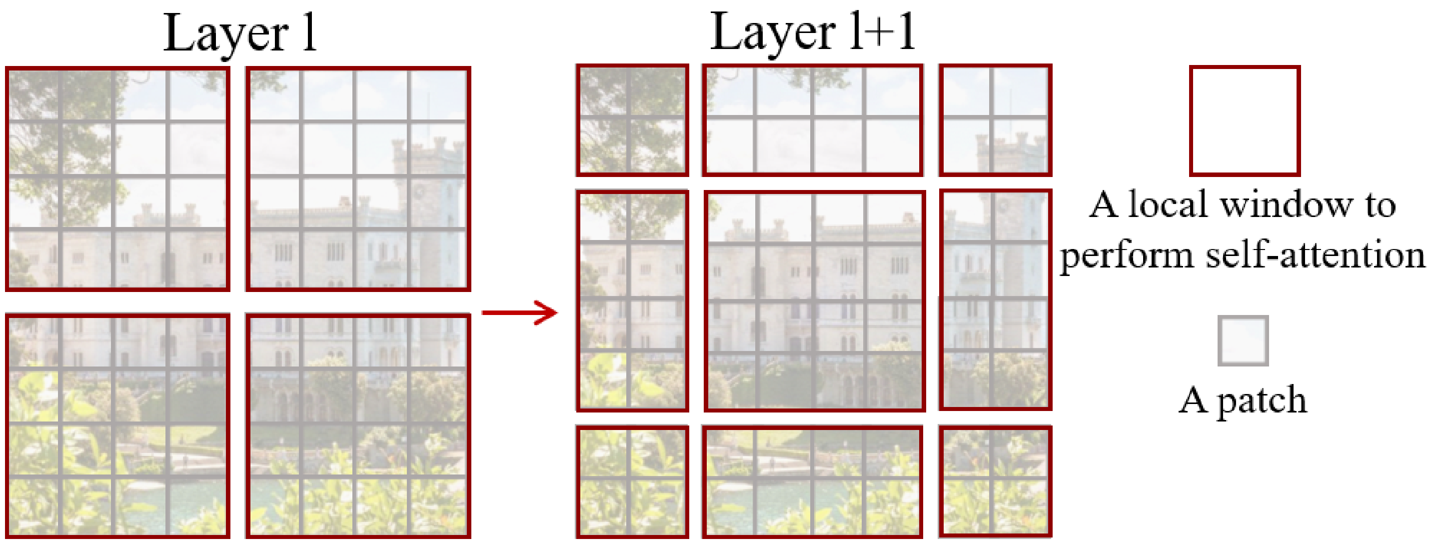


Figure 2. An illustration of the *shifted window* approach for computing self-attention in the proposed Swin Transformer architecture. In layer l (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer $l + 1$ (right), the window partitioning is shifted, resulting in new windows. The self-attention computation in the new windows crosses the boundaries of the previous windows in layer l, providing connections among them.

SWin Transformer

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
(b) ImageNet-22K pre-trained models					
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3

Table 1. Comparison of different backbones on ImageNet-1K classification. Throughput is measured using the GitHub repository of [68] and a V100 GPU, following [63].

(a) Various frameworks

Method	Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	#param.	FLOPs	FPS
Cascade	R-50	46.3	64.3	50.5	82M	739G	18.0
Mask R-CNN	Swin-T	**50.5**	**69.3**	**54.9**	86M	745G	15.3
ATSS	R-50	43.5	61.9	47.0	32M	205G	28.3
	Swin-T	**47.2**	**66.5**	**51.3**	36M	215G	22.3
RepPointsV2	R-50	46.5	64.6	50.3	42M	274G	13.6
	Swin-T	**50.0**	**68.5**	**54.2**	45M	283G	12.0
Sparse	R-50	44.5	63.4	48.2	106M	166G	21.0
R-CNN	Swin-T	**47.9**	**67.3**	**52.3**	110M	172G	18.4

(b) Various backbones w. Cascade Mask R-CNN

	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6

Plain ViT Backbones

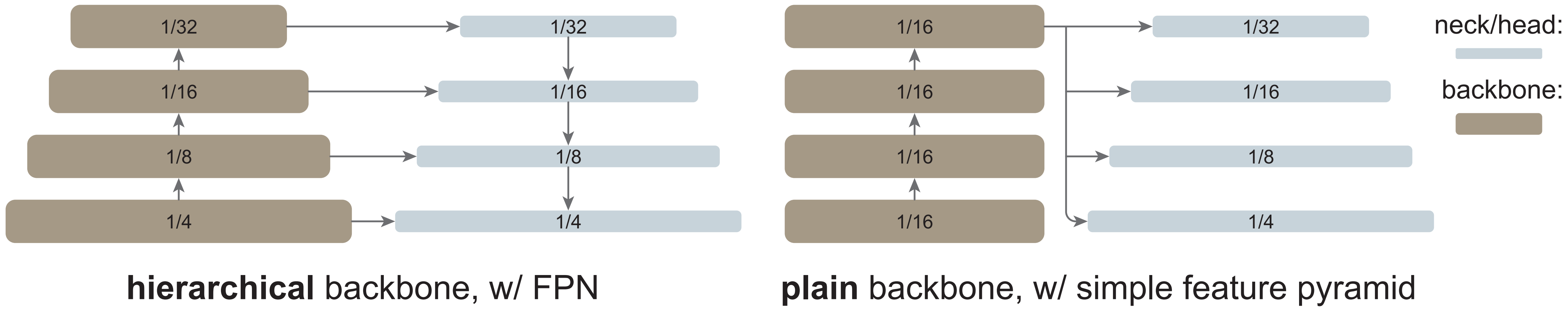
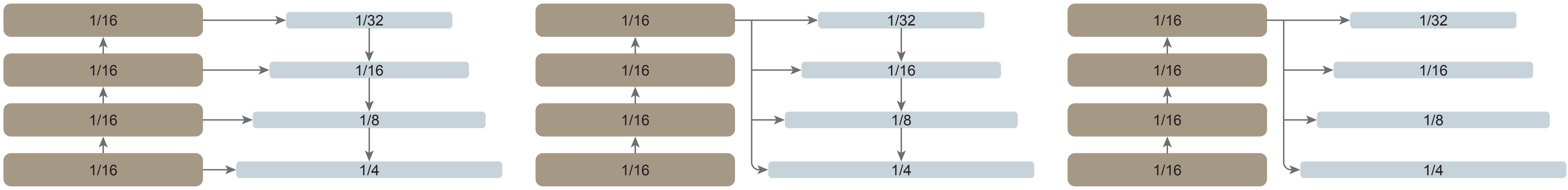


Figure 1: A typical hierarchical-backbone detector (left) *vs*. our plain-backbone detector (right). Traditional hierarchical backbones can be naturally adapted for multi-scale detection, *e.g.*, using FPN. Instead, we explore building a simple pyramid from only the last, large-stride (16) feature map of a plain backbone.

Plain ViT Backbones



(a) FPN, 4-stages (b) FPN, last map (c) simple feature pyramid

pyramid design	ViT-B AP^{box}	ViT-B AP^{mask}	ViT-L AP^{box}	ViT-L AP^{mask}
no feature pyramid	47.8	42.5	51.2	45.4
(a) FPN, 4-stage	50.3 (+2.5)	44.9 (+2.4)	54.4 (+3.2)	48.4 (+3.0)
(b) FPN, last-map	50.9 (+3.1)	45.3 (+2.8)	**54.6** (+3.4)	48.5 (+3.1)
(c) simple feature pyramid	**51.2** (+3.4)	**45.5** (+3.0)	**54.6** (+3.4)	**48.6** (+3.2)

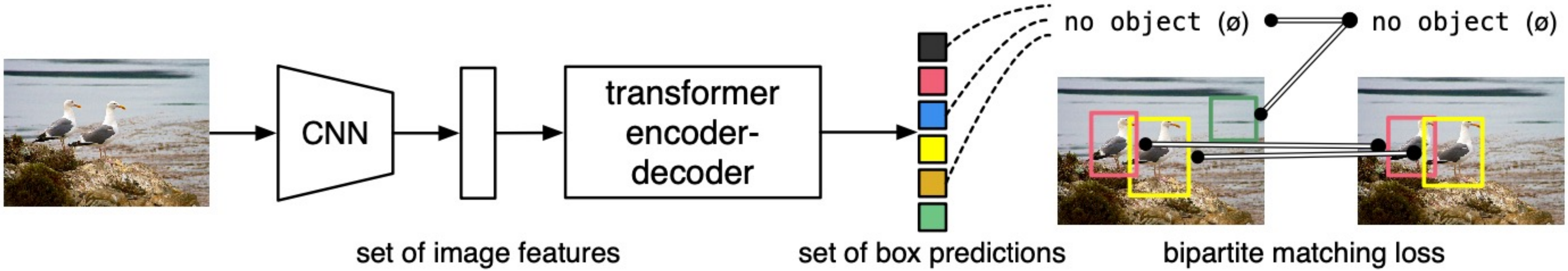
Plain ViT Backbones

backbone	pre-train	Mask R-CNN AP^{box}	Mask R-CNN AP^{mask}	Cascade Mask R-CNN AP^{box}	Cascade Mask R-CNN AP^{mask}
hierarchical-backbone detectors:					
Swin-B	21K, sup	51.4	45.4	54.0	46.5
Swin-L	21K, sup	52.4	46.2	54.8	47.3
MViTv2-B	21K, sup	53.1	47.4	55.6	48.1
MViTv2-L	21K, sup	53.6	47.5	55.7	48.3
MViTv2-H	21K, sup	54.1	47.7	55.8	48.3
our plain-backbone detectors:					
ViT-B	1K, MAE	51.6	45.9	54.0	46.7
ViT-L	1K, MAE	55.6	49.2	57.6	49.8
ViT-H	1K, MAE	**56.7**	**50.1**	**58.7**	**50.9**

Table 5: **Comparisons of plain *vs.* hierarchical backbones** using Mask R-CNN [25] and Cascade Mask R-CNN [4] on COCO. Tradeoffs are plotted in Figure 3. All entries are implemented and run by us to align low-level details.

End-to-End Object Detection with Transformers

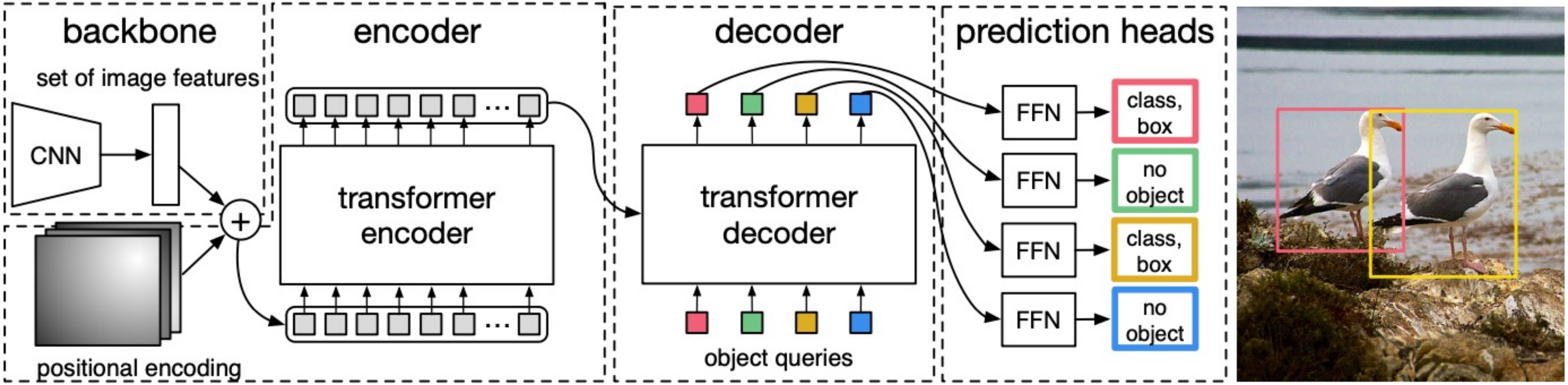
- Do we need specialized machinery (i.e. Faster RCNN) for object detection?
- Cast detection as a set prediction problem



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

End-to-End Object Detection with Transformers

- Architecture



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

End-to-End Object Detection with Transformers

- Set matching loss function

- $$\hat{\sigma} = \underset{\sigma \in \mathfrak{S}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{\mathrm{match}}(y_i, \hat{y}_{\sigma(i)})$$
  $$-\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\mathrm{box}}(b_i, \hat{b}_{\sigma(i)})$$

- $$\mathcal{L}_{\mathrm{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\mathrm{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

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

Comparison against Faster RCNN

Table 1: Comparison with Faster R-CNN with a ResNet-50 and ResNet-101 backbones on the COCO validation set. The top section shows results for Faster R-CNN models in Detectron2 [50], the middle section shows results for Faster R-CNN models with GIoU [38], random crops train-time augmentation, and the long `9x` training schedule. DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP_S but greatly improved AP_L. We use torchscript Faster R-CNN and DETR models to measure FLOPS and FPS. Results without R101 in the name correspond to ResNet-50.

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

What do object queries learn?

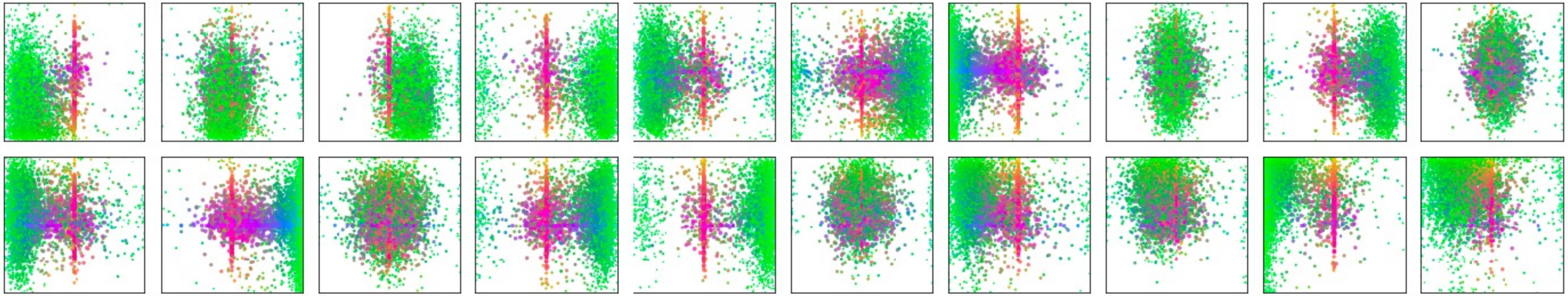


Fig. 7: Visualization of all box predictions on all images from COCO 2017 val set for 20 out of total $N = 100$ prediction slots in DETR decoder. Each box prediction is represented as a point with the coordinates of its center in the 1-by-1 square normalized by each image size. The points are color-coded so that green color corresponds to small boxes, red to large horizontal boxes and blue to large vertical boxes. We observe that each slot learns to specialize on certain areas and box sizes with several operating modes. We note that almost all slots have a mode of predicting large image-wide boxes that are common in COCO dataset.