Lecture 18:
Vision Transformers

Admin: Grading

- A3 grades Will be out today or tomorrow
- Midterm: Submit regrade requests by tonight on Piazza

Admin: PyTorch Tutorial

- A4 – A6 require deeper PyTorch knowledge than A1 – A3
- Instead of just PyTorch tensors, you also need to use autograd, modules, optimizers, learning rate schedules, etc
- We have prepared a PyTorch tutorial that walks through these concepts in the case of image classification: https://piazza.com/class/kxtai72amx34p0?cid=765

Admin: A4

Object Detection: FCOS, Faster R-CNN

Due Tuesday, 3/29/2022, 11:59pm ET

Updated A4 starter code out yesterday:

- Incorporates clarifications / documentation improvements from Piazza
- No functional code changes: you can copy-paste all your code from previous to current version and everything should still work
- Optional: if you are not confused, can keep going with original release

Admin: A4

- Autograder will be out (hopefully?) tomorrow
- We will give more autograder submissions (10/day)
- No tricky hidden test cases
- If you get good final AP, its very likely you are ok
- Autograding:
  - Very light
  - Make sure your code is vectorized
  - Make sure you didn't hardcode any image dimensions, feature dimensions, number of layers, etc

Admin: Project

Project details are available here:
https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/project.html

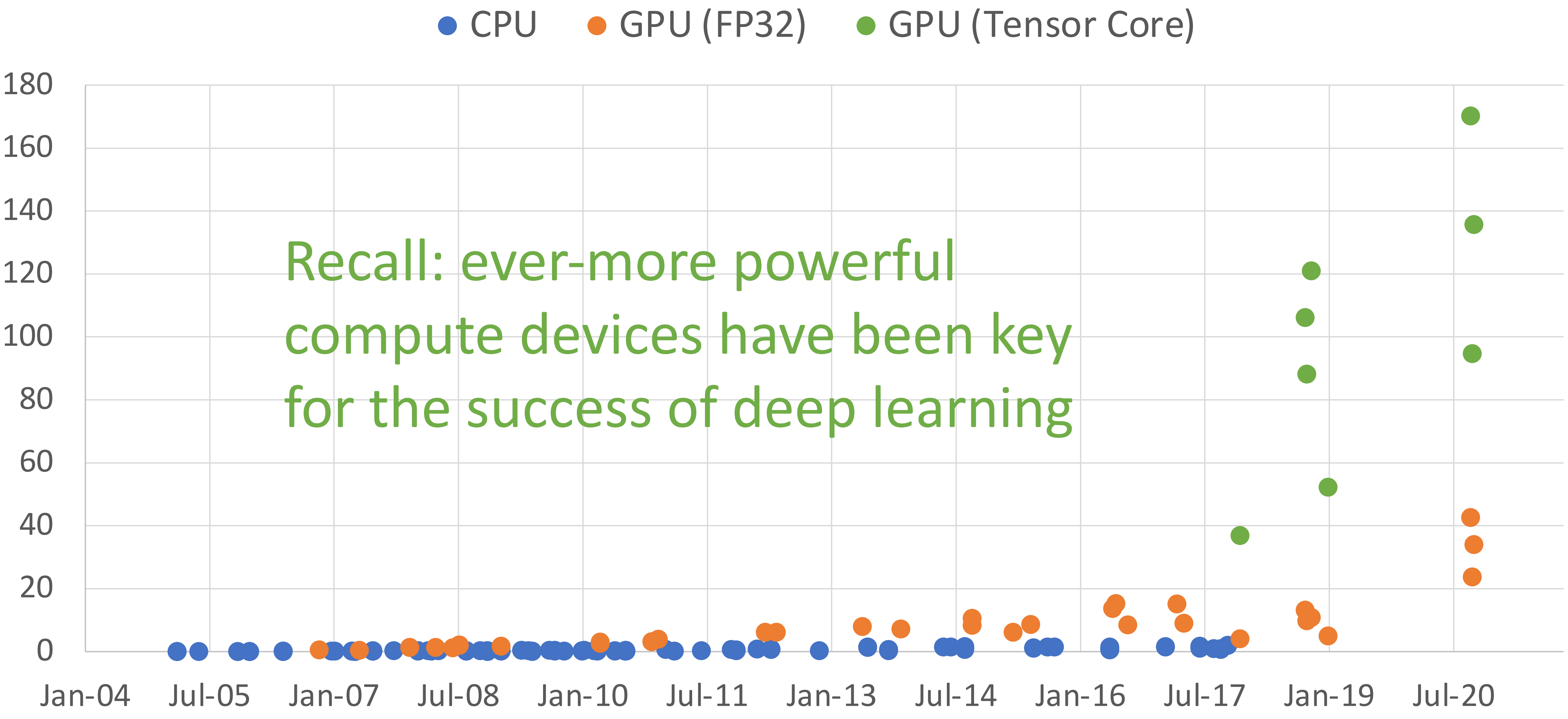
Project options:
- Image Classification
- Single-Image Super-Resolution
- Novel View Synthesis with NeRF
- Choose Your Own

For Choose Your Own project: need to submit a **project proposal** by Friday April 1, 11:59 ET. Make a private post on Piazza under tag "project-proposal". This is not graded, but we need to ok the project.

Today: Vision Transformers

But first…

GFLOP per Dollar



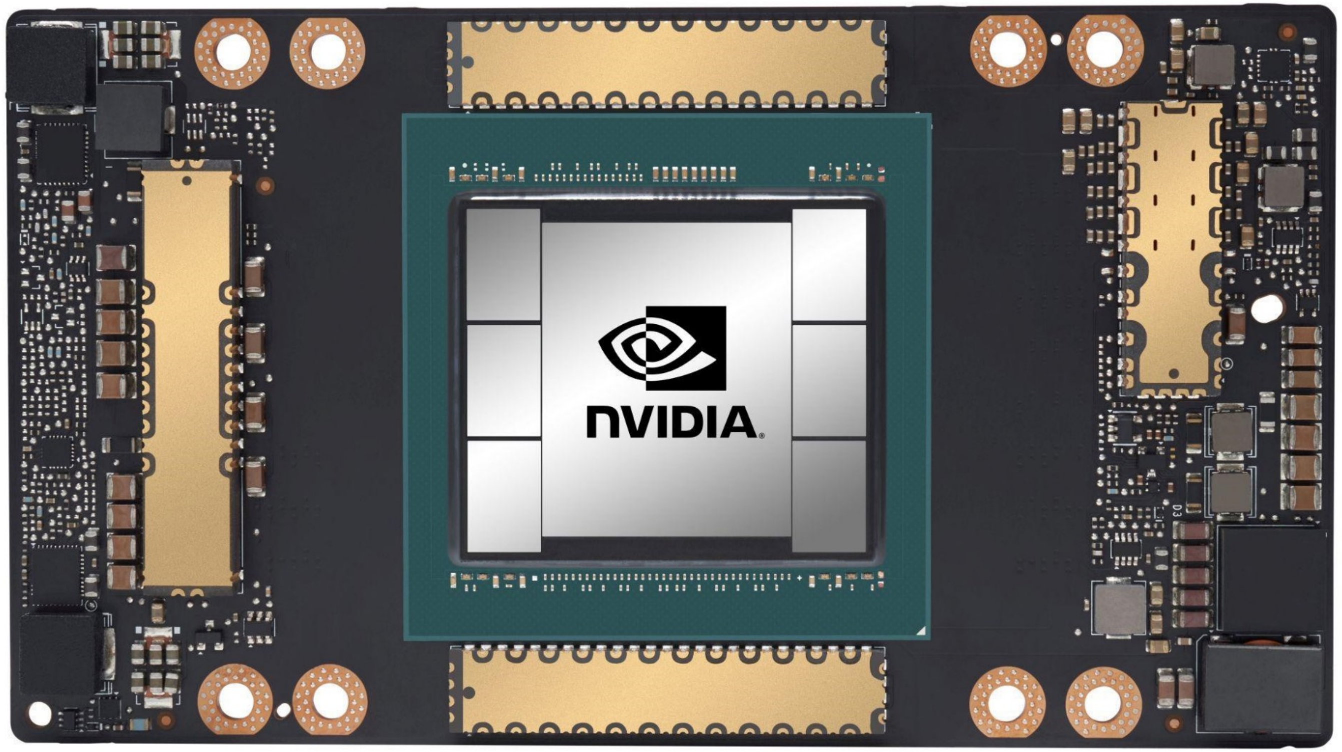
Recall: ever-more powerful compute devices have been key for the success of deep learning

Best GPU money can buy: NVIDA A100

Memory:
Capacity: 40/80 GB HBM2
Bandwidth: 1.5/2.0 TB/sec

Compute:
FP64: 9.7 TFLOP/sec
FP32: 19.5 TFLOP/sec
BF16: 39 TFLOP/sec
FP16: 78 TFLOP/sec

FLOP = “Floating Point Operation”; one addition, multiplication, etc
TFLOP = 1 trillion FLOPs (10^{12})



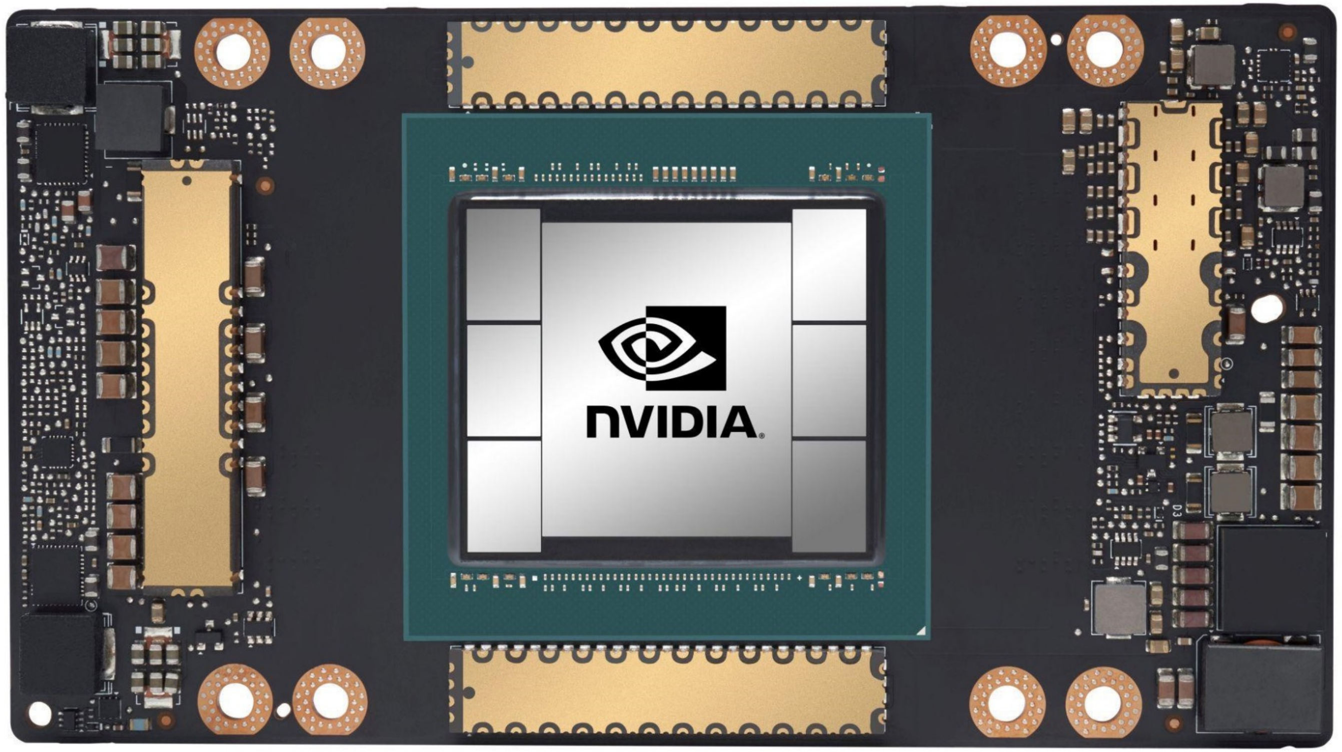
Best GPU money can buy: NVIDA A100

Memory:
Capacity: 40/80 GB HBM2
Bandwidth: 1.5/2.0 TB/sec

Compute:
FP64: 9.7 TFLOP/sec
FP32: 19.5 TFLOP/sec
BF16: 39 TFLOP/sec
FP16: 78 TFLOP/sec

Tensor Cores:
TF32: 156 TFLOP/sec
FP16/BF16: 312 TFLOP/sec

FLOP = “Floating Point Operation”; one addition, multiplication, etc
TFLOP = 1 trillion FLOPs (10^{12})



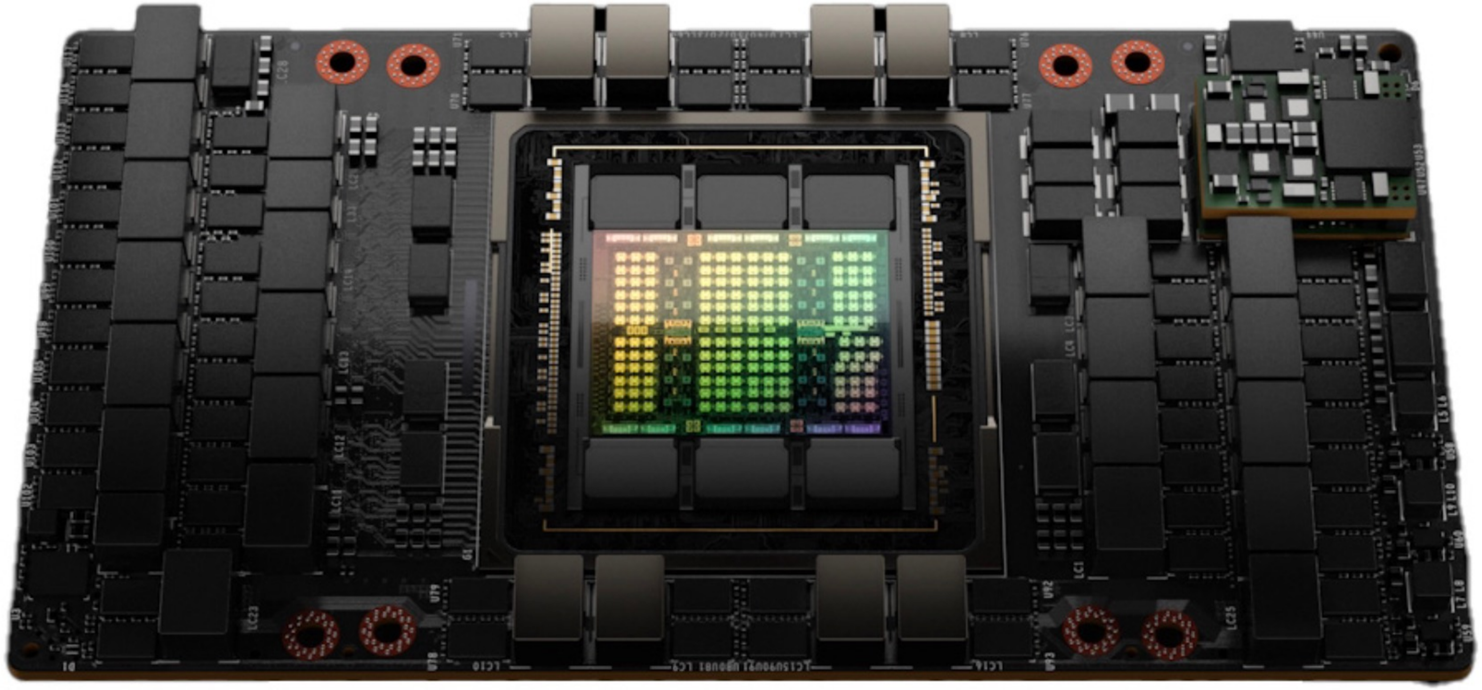
Yesterday: New NVIDIA H100 GPU

Memory:
Capacity: 40/80 GB HBM3
Bandwidth: 3.0 TB/sec **(1.5x better)**

FLOP = “Floating Point Operation”; one addition, multiplication, etc
TFLOP = 1 trillion FLOPs (10^{12})

Compute:
FP64: 30 TFLOP/sec (**3x better)**
FP32: 60 TFLOP/sec **(3x better)**
BF16: 120 TFLOP/sec (**3x better)**
FP16: 120 TFLOP/sec **(1.5x better)**

Tensor Cores:
TF32: 500 TFLOP/sec **(3.2x better)**
FP16/BF16: 1000 TFLOP/sec **(3.2x better)**



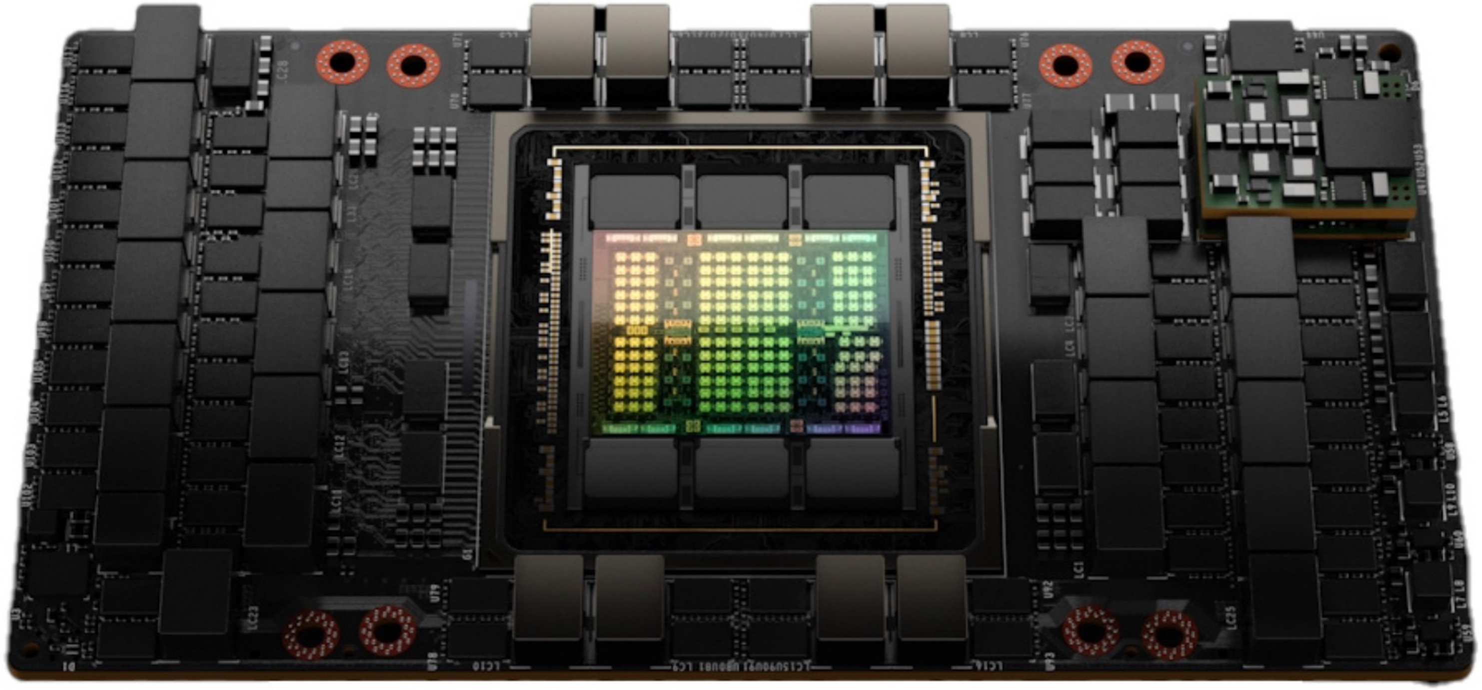
Yesterday: New NVIDIA H100 GPU

Memory:
Capacity: 40/80 GB HBM3
Bandwidth: 3.0 TB/sec **(1.5x better)**

FLOP = "Floating Point Operation"; one addition, multiplication, etc
TFLOP = 1 trillion FLOPs (10^{12})

What are these?

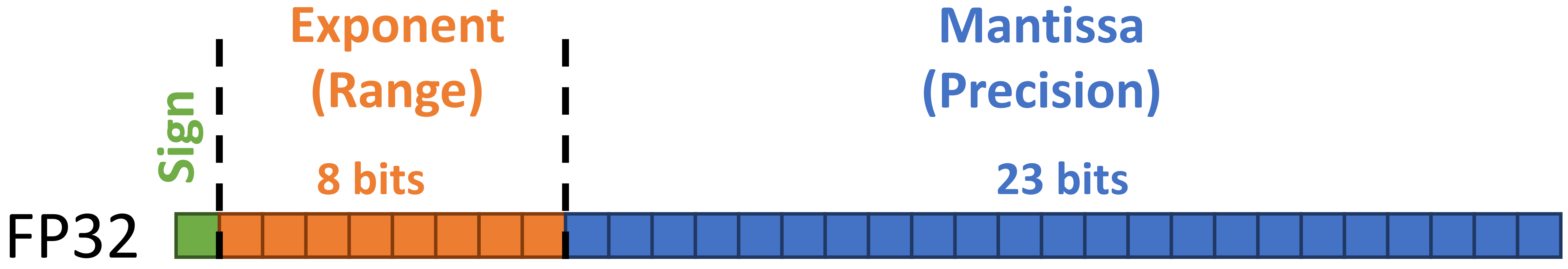
Compute:
FP64: 30 TFLOP/sec (**3x better)**
FP32: 60 TFLOP/sec **(3x better)**
BF16: 120 TFLOP/sec (**3x better)**
FP16: 120 TFLOP/sec **(1.5x better)**



Tensor Cores:
TF32: 500 TFLOP/sec **(3.2x better)**
FP16/BF16: 1000 TFLOP/sec **(3.2x better)**

Floating Point Formats $(-\mathbf{1})^{S}\left(\mathbf{2}^{\boldsymbol{E}+\boldsymbol{bias}}\right)\left(1+\frac{M}{2^{|M|}}\right)$

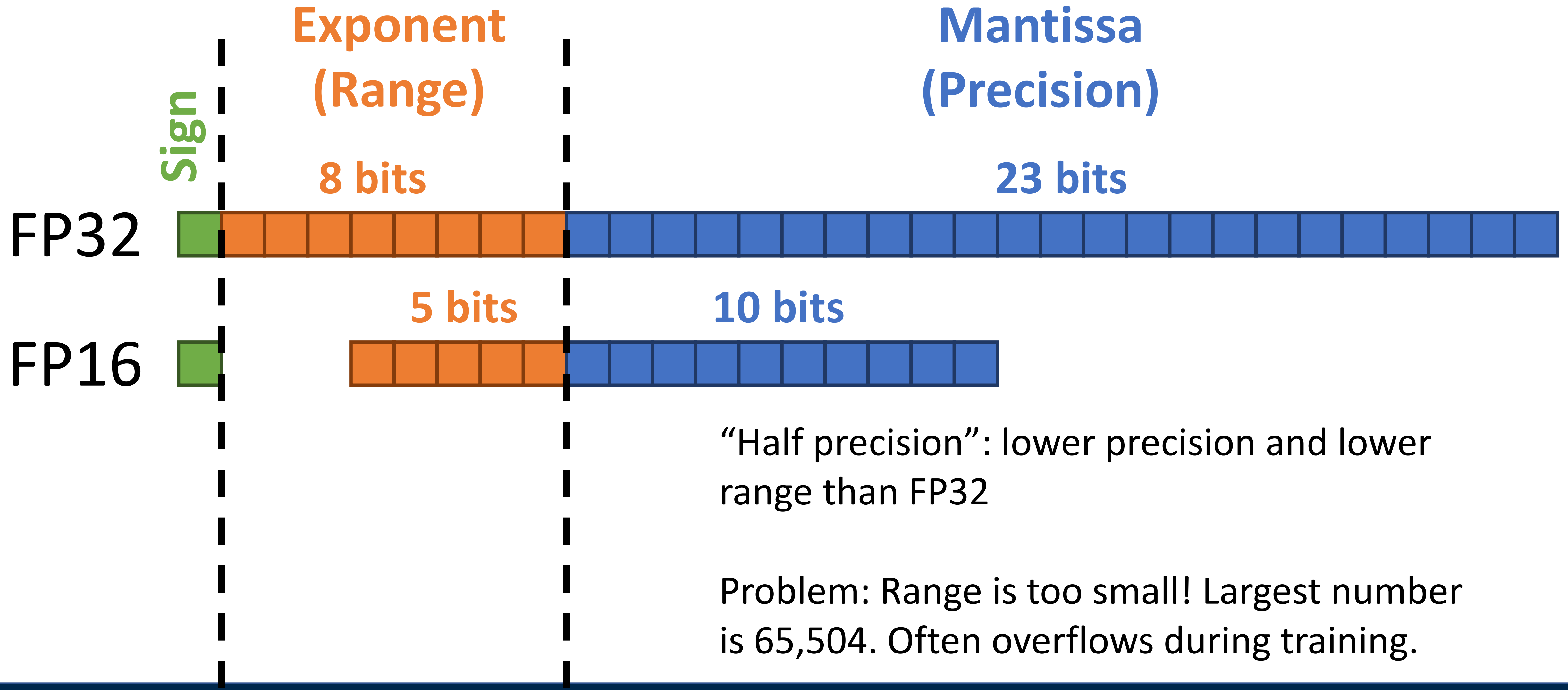
Floating Point Formats $(-1)^{S}\left(2^{E+bias}\right)\left(1+\frac{M}{2^{|M|}}\right)$



"Single precision": Standard datatype for deep learning

Bits are expensive: take memory, takes time to move them around, Multiplication is quadratic in #bits
Can we use fewer than 32 bits?

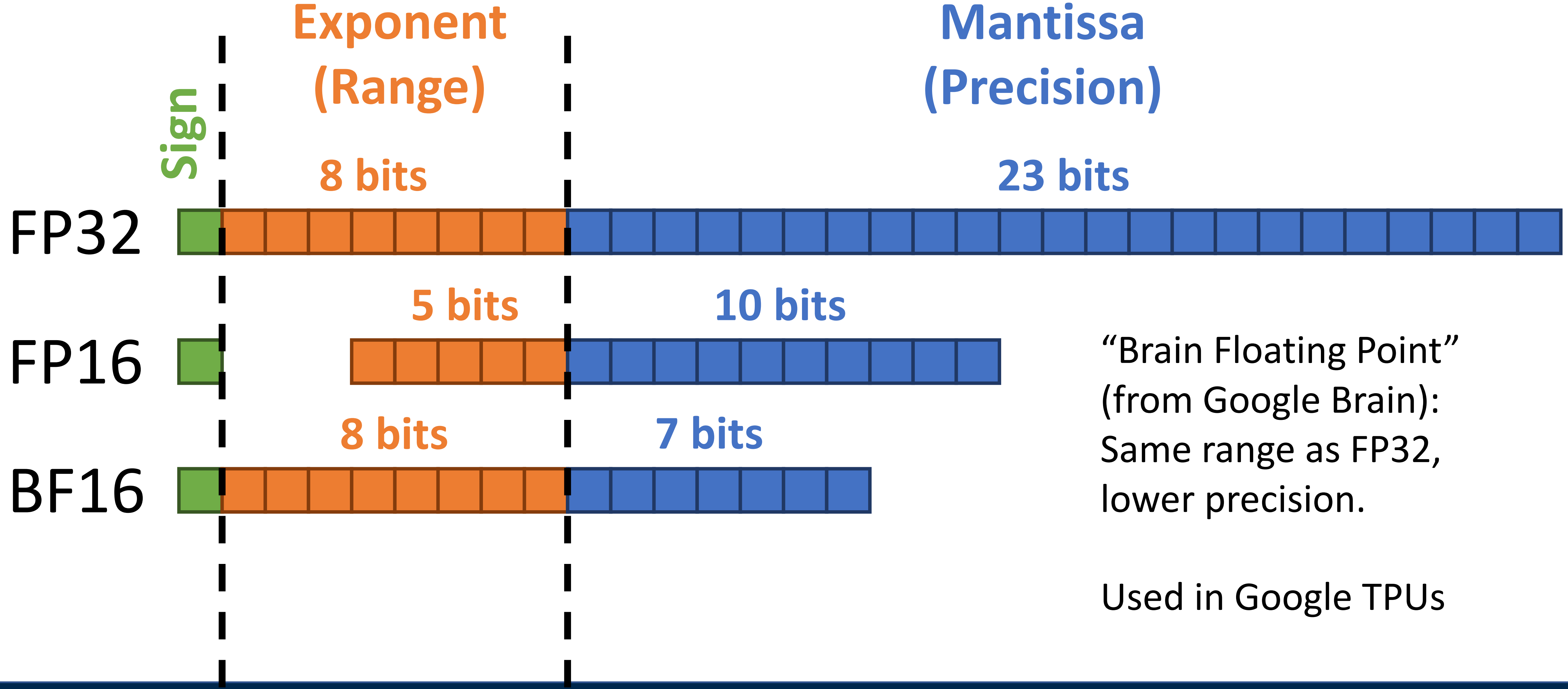
Floating Point Formats $(-1)^S(2^{E+bias})\left(1+\frac{M}{2^{|M|}}\right)$



"Half precision": lower precision and lower range than FP32

Problem: Range is too small! Largest number is 65,504. Often overflows during training.

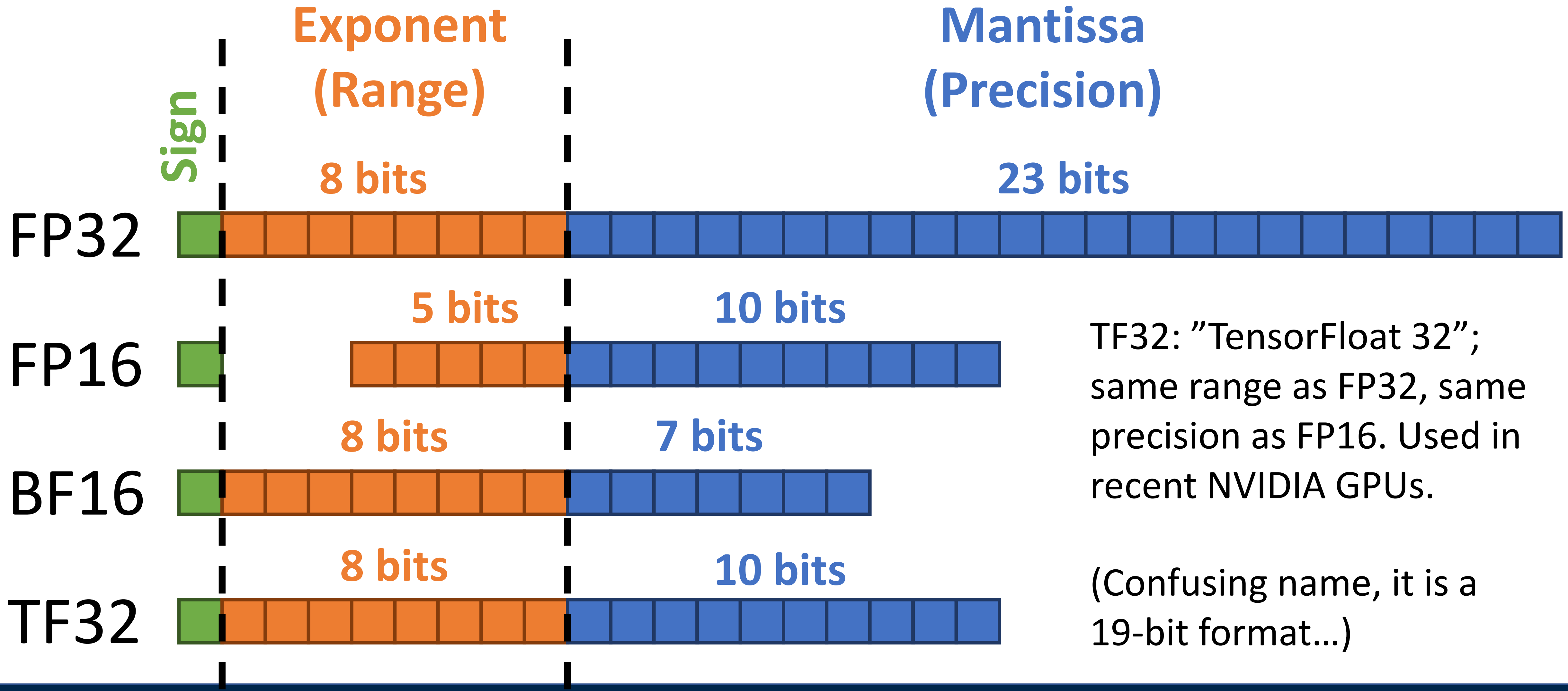
Floating Point Formats $(-1)^S\left(2^{E+bias}\right)\left(1+\frac{M}{2^{|M|}}\right)$



"Brain Floating Point" (from Google Brain): Same range as FP32, lower precision.

Used in Google TPUs

Floating Point Formats $(-1)^S\left(2^{E+bias}\right)\left(1+\frac{M}{2^{|M|}}\right)$



TF32: "TensorFloat 32"; same range as FP32, same precision as FP16. Used in recent NVIDIA GPUs.

(Confusing name, it is a 19-bit format…)

Mixed Precision

We often need to compute dot products (for matrix multiply, convolution, etc):

$$y = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

Mixed Precision

We often need to compute dot products (for matrix multiply, convolution, etc):

$$y = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

Multiplication is more expensive than addition

Idea: Multiply in low precision, add in high precision

Mixed Precision

We often need to compute dot products (for matrix multiply, convolution, etc):

$$y = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

Multiplication is more expensive than addition
Idea: Multiply in low precision, add in high precision

Inputs: x_i, w_i in low precision (FP16, BF16, TF32)
Output: y in high precision (FP32)

$$y = FP32(x_1 w_1) + FP32(x_2 w_2) + \cdots + FP32(x_n w_n)$$

Mixed Precision

We often need to compute dot products (for matrix multiply, convolution, etc):

$$y = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

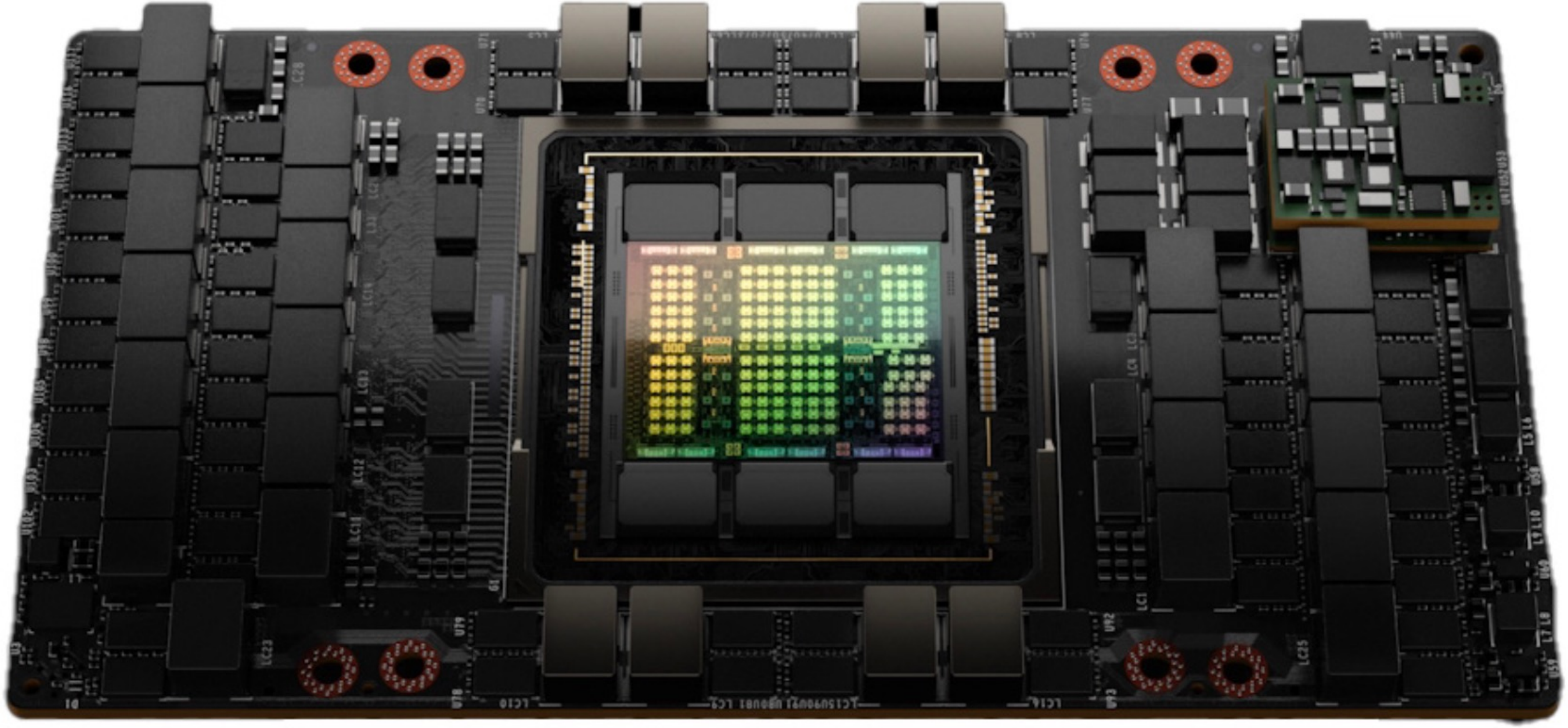
Multiplication is more expensive than addition
Idea: Multiply in low precision, add in high precision

Inputs: x_i, w_i in low precision (FP16, BF16, TF32)
Output: y in high precision (FP32)

$$y = FP32(x_1 w_1) + FP32(x_2 w_2) + \cdots + FP32(x_n w_n)$$

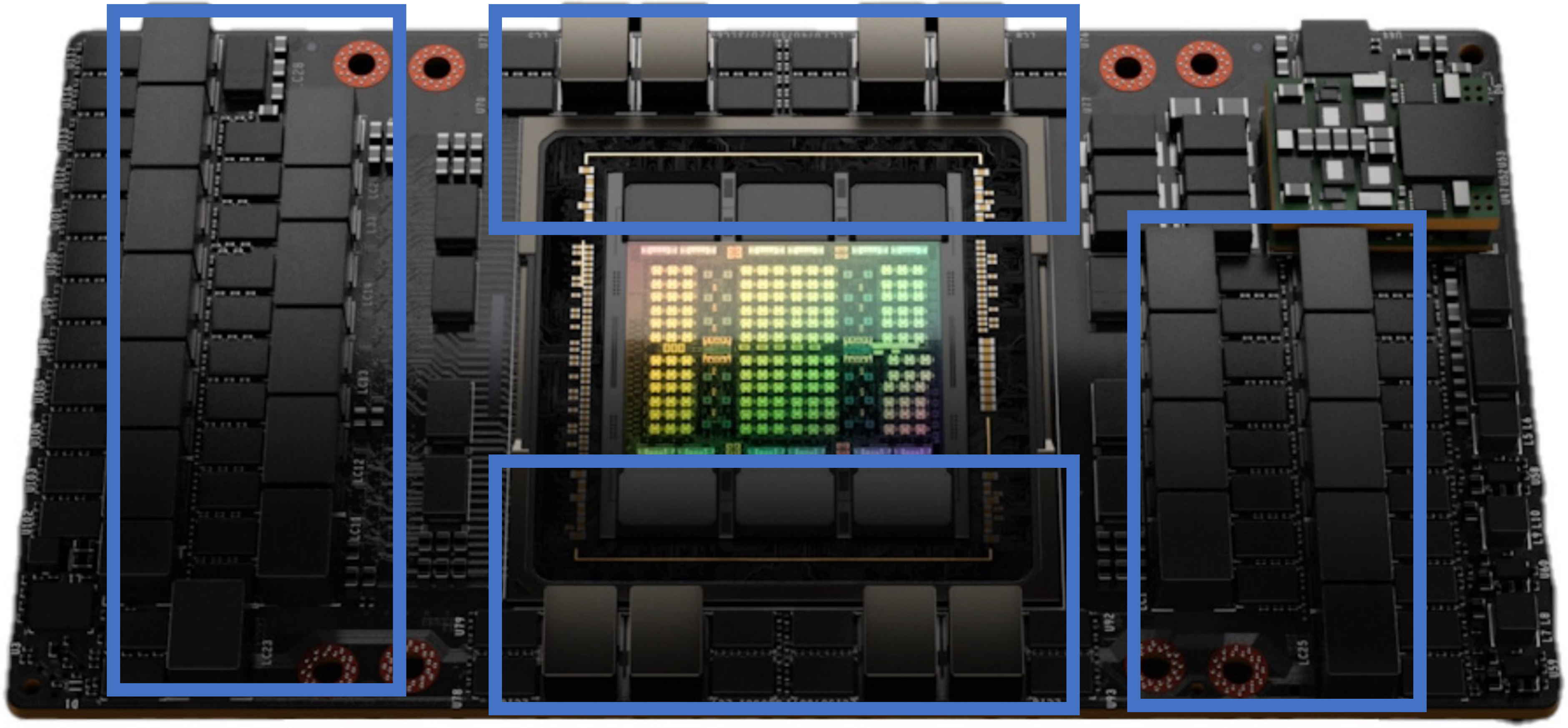
Tensor Cores in NVIDIA GPUs are special hardware for mixed-precision matrix multiplication with different low-precision formats (TF32, BF16 best for neural nets)

Yesterday: New NVIDIA H100 GPU

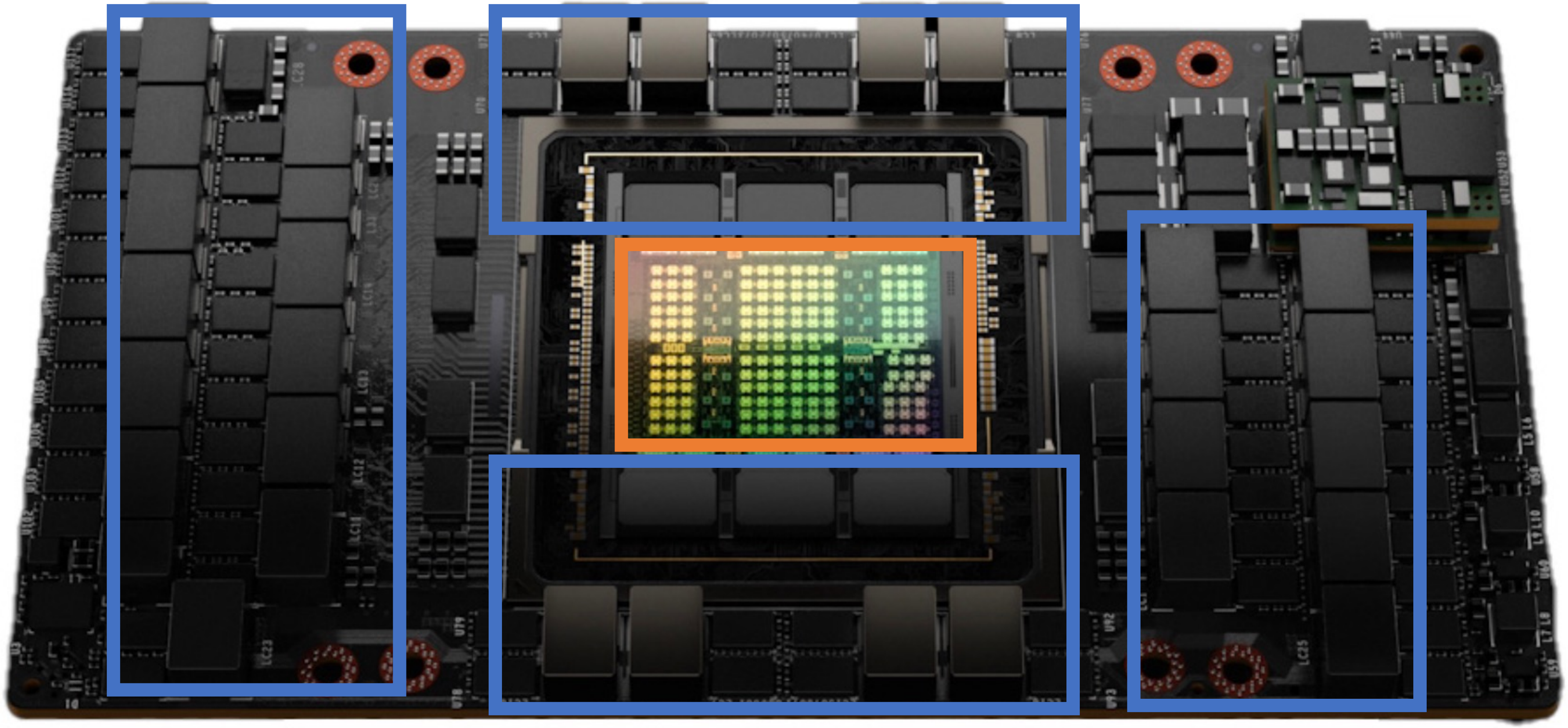


Yesterday: New NVIDIA H100 GPU

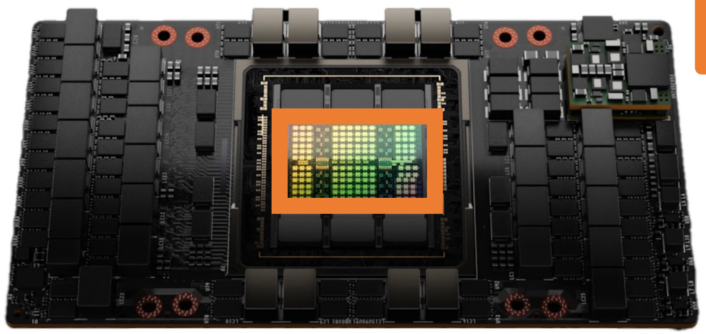
80 GB of HBM3 memory



Yesterday: New NVIDIA H100 GPU



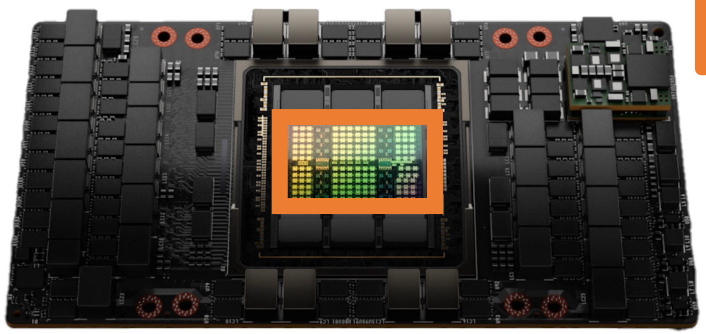
Yesterday: New NVIDIA H100 GPU



Yesterday: New NVIDIA H100 GPU

144 “Streaming Multiprocessors”: Independent multicore processors

(only 132/144 are enabled due to issues with yield)



H100 SM

Each SM has 4 subunits that can each simultaneously execute 32 threads (1 **warp**)

32 **FP32 cores** per subunit; each can compute $y = ax + b$ per clock cycle (1 **multiply-add** = 2 FLOPs)

H100 SM

Each SM has 4 subunits that can each simultaneously execute 32 threads (1 **warp**)

32 **FP32 cores** per subunit; each can compute y = ax + b per clock cycle (1 **multiply-add** = 2 FLOPs)

(132 SMs/GPU) * (128 cores/SM)
* (2 FLOPs/core/cycle) * (1.775 * 10^9 cycles/sec)
= 60 * 10^9 FLOPs/GPU/sec

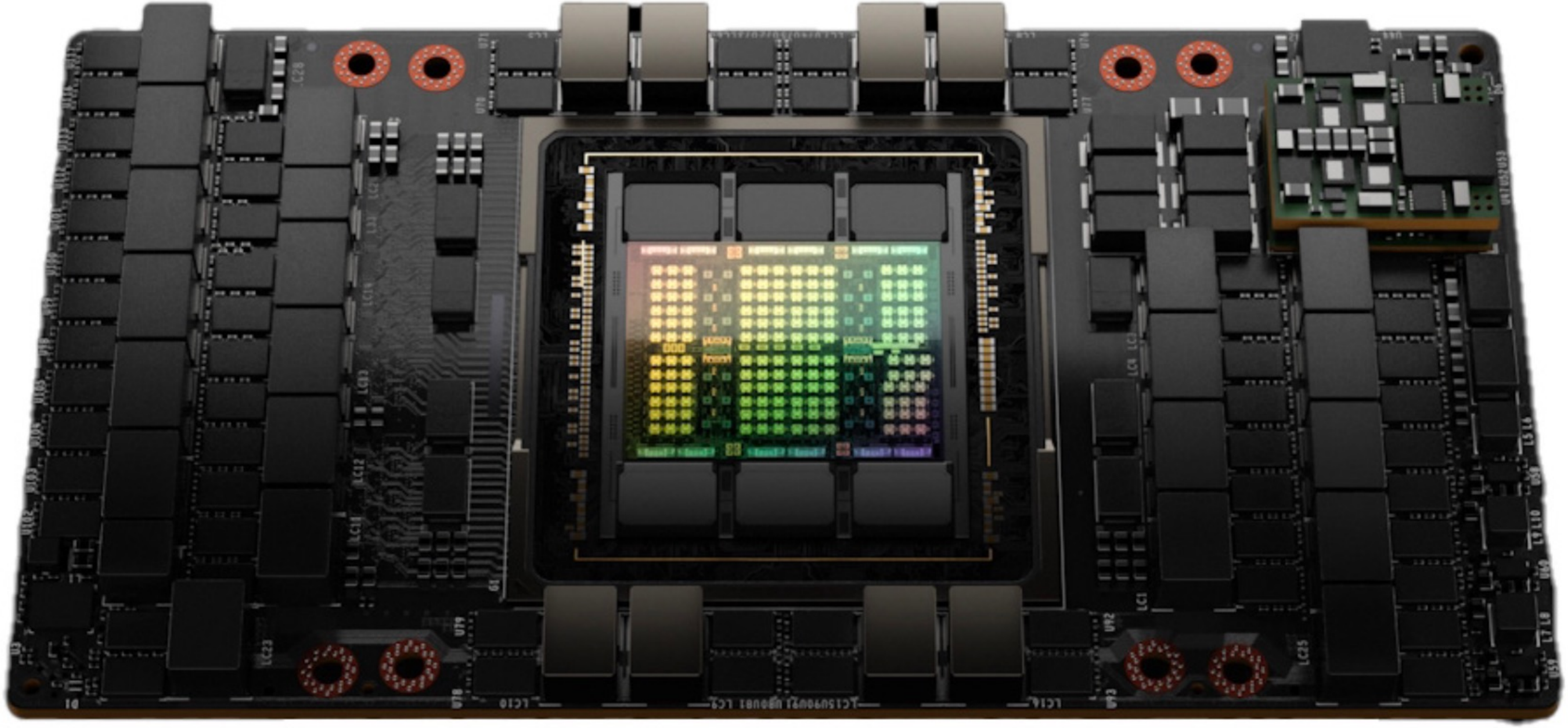
H100 SM

Each SM has 4 subunits that can each simultaneously execute 32 threads (1 **warp**)

32 **FP32 cores** per subunit; each can compute y = ax + b per clock cycle (1 **multiply-add** = 2 FLOPs)

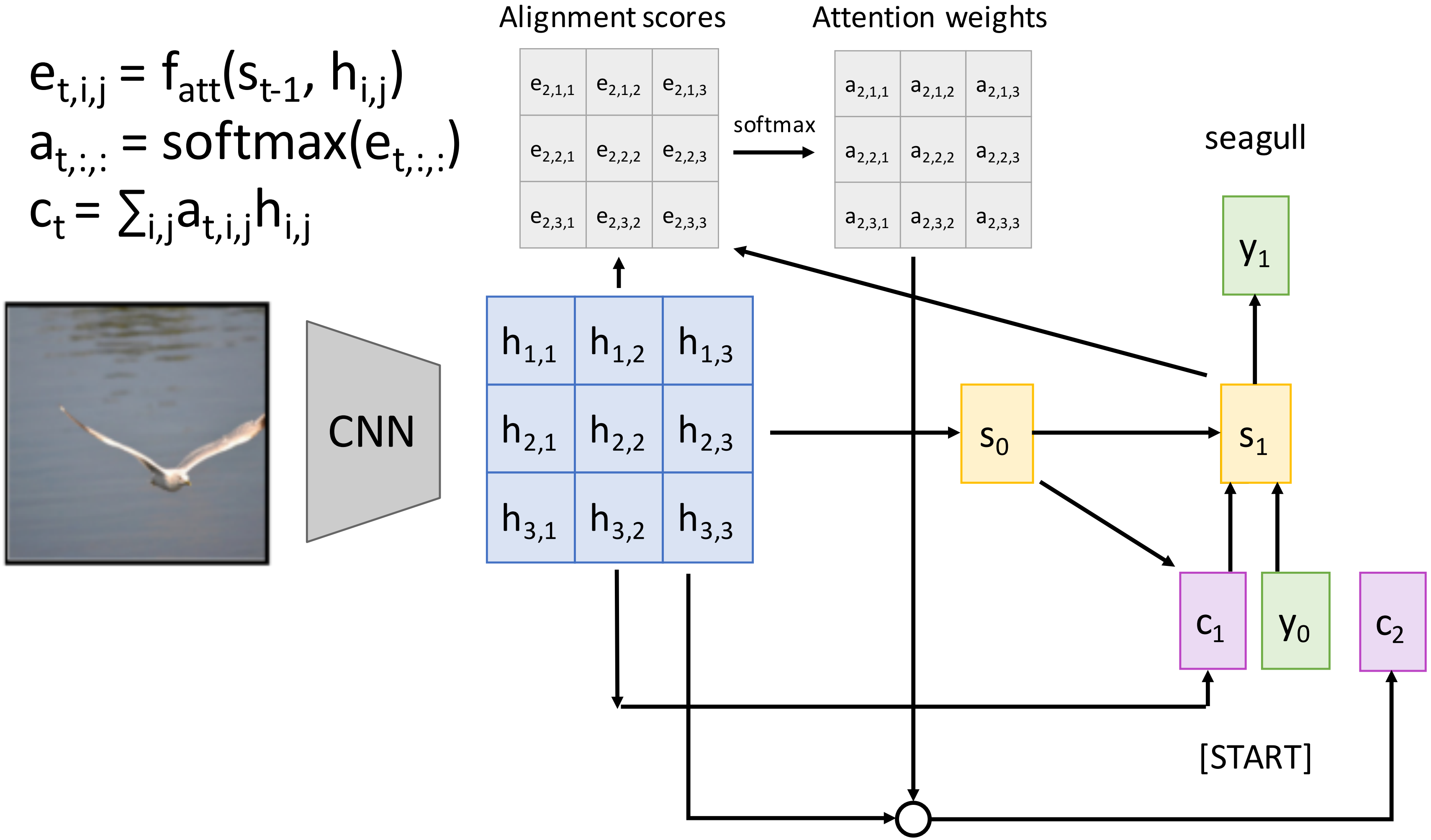
4 **Tensor cores** per subunit; each can do one tiny matrix multiply per clock: [4 x 16] * [16 x 8] = [4 x 8] (FP16/FP32, 4*8*16*2 FLOPs = 1024 FLOPs)

H100 GPU: Expect Bigger Models!

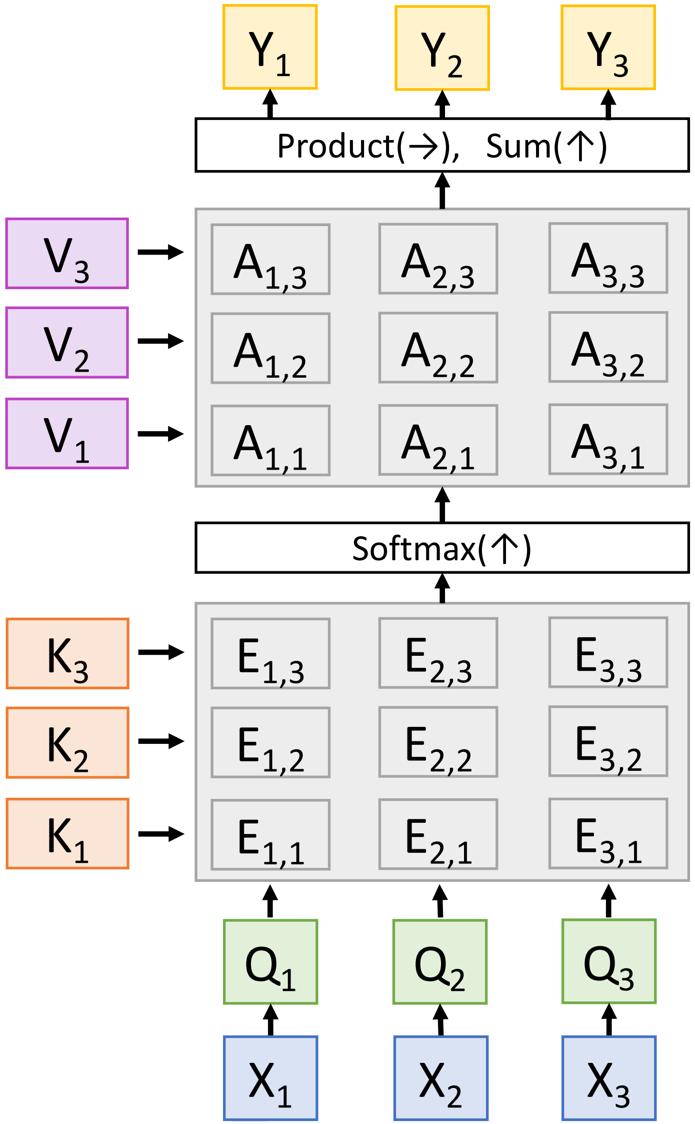


Last Time: Attention

$$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$$
$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



Last Time: Self-Attention Layer

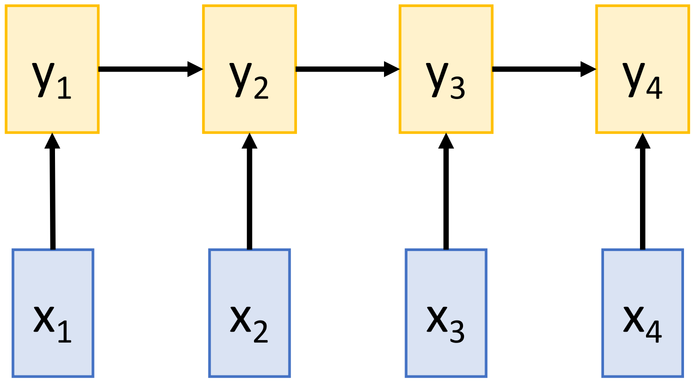


Inputs:
Input vectors: **X** (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)
Query matrix: $\mathbf{W_Q}$ (Shape: $D_X \times D_Q$)

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

Last Time: Three Ways of Processing Sequences

Recurrent Neural Network

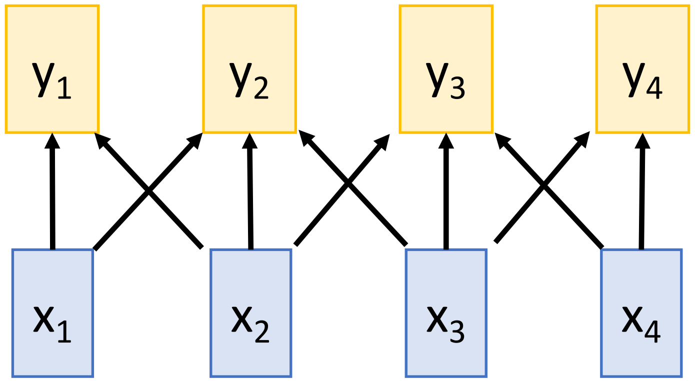


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

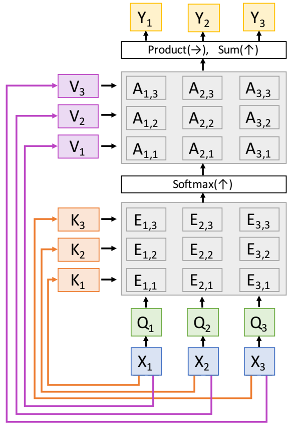


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention



Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

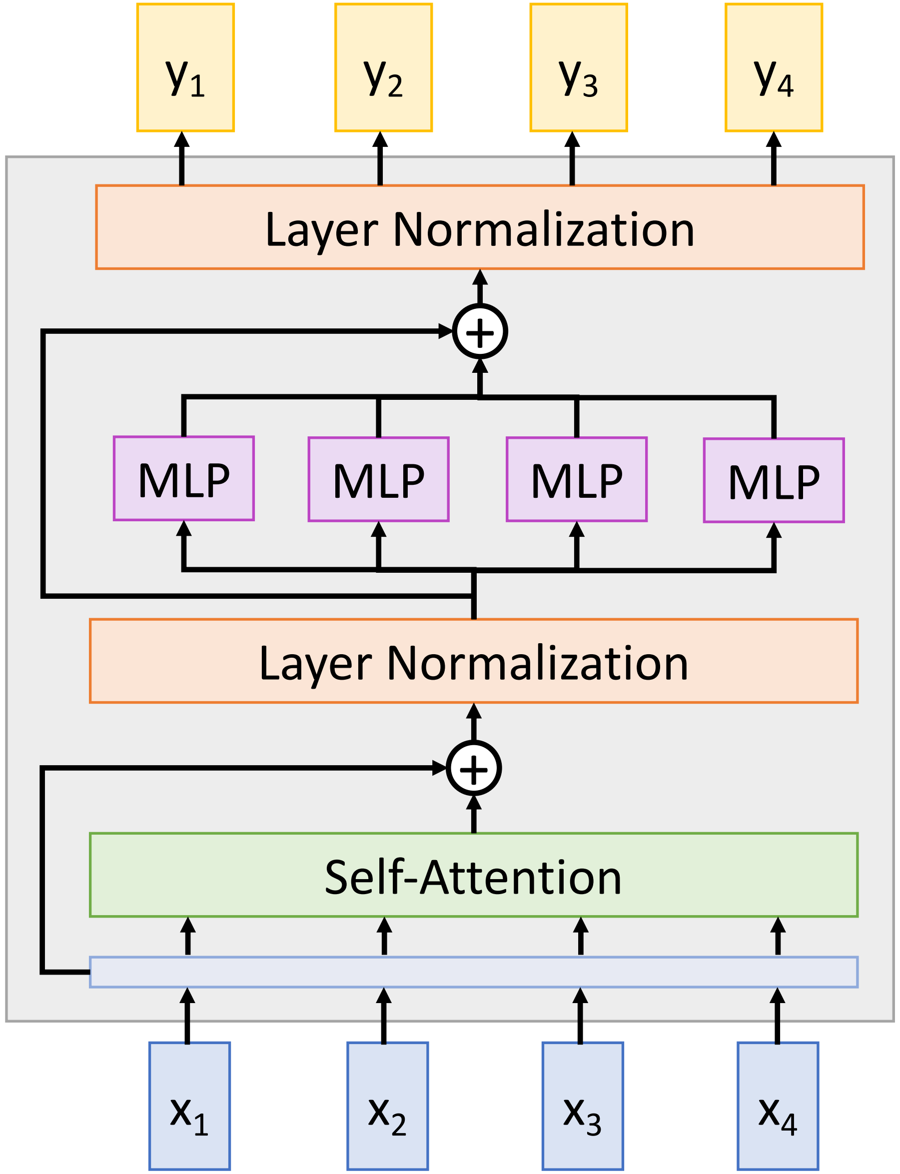
(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

Last Time: Transformer

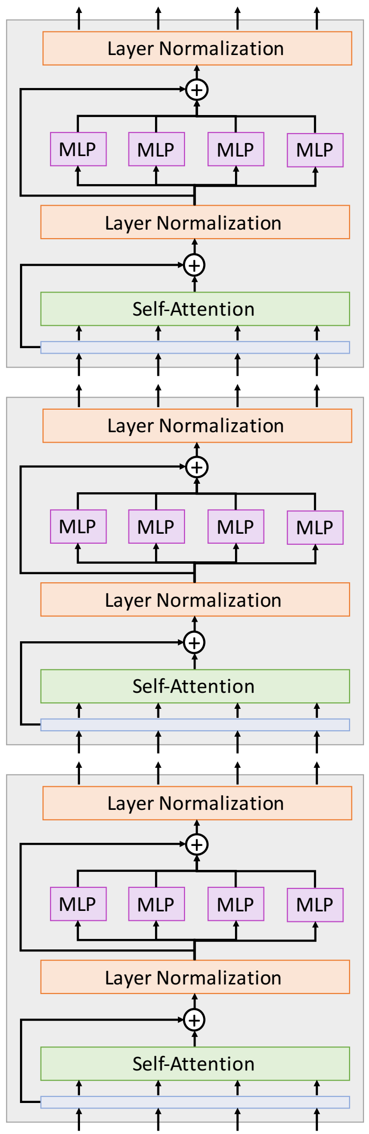
Transfomer block inputs a set of vectors, outputs a set of vectors.

Vectors only communicate via (multiheaded) self-attention



Vaswani et al, "Attention is all you need", NeurIPS 2017

Last Time: Transformer



<u>Transformer Block</u>:
Input: Set of vectors x
Output: Set of vectors y

Hyperparameters:
- Number of blocks
- Number of heads per block
- Width (channels per head, FFN width)

Vaswani et al, "Attention is all you need", NeurIPS 2017

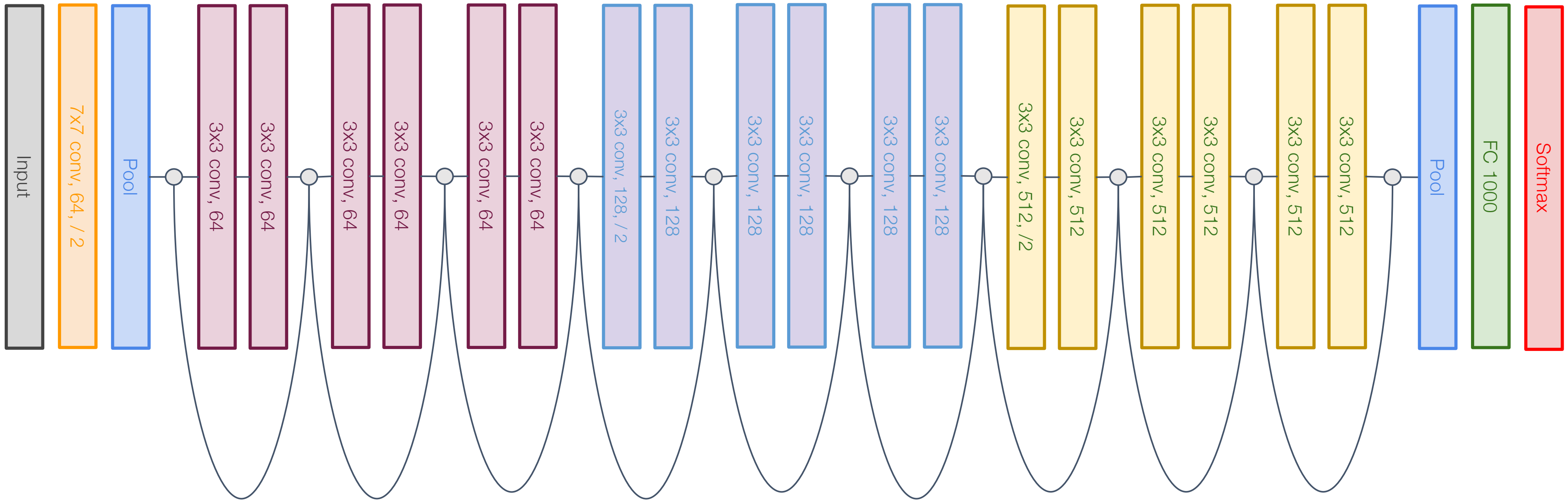
Last Time: Transformers in NLP

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)

Today: How to use Attention / Transformers for Vision?

Idea #1: Add attention to existing CNNs

Start from standard CNN architecture (e.g. ResNet)

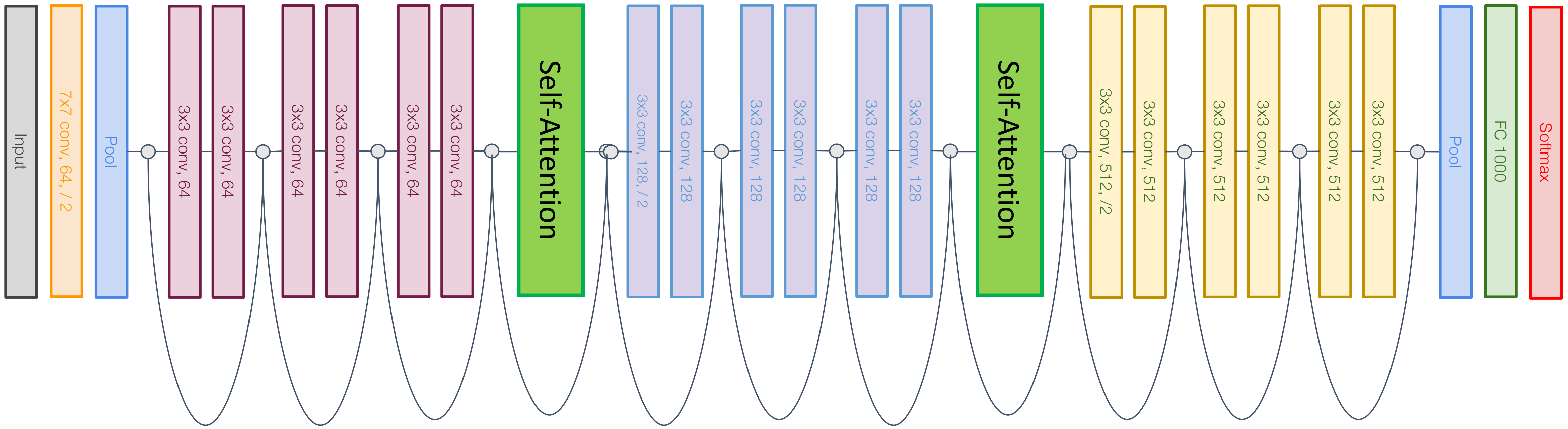


Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018
Wang et al, "Non-local Neural Networks", CVPR 2018

Idea #1: Add attention to existing CNNs

Start from standard CNN architecture (e.g. ResNet)

Add Self-Attention blocks between existing ResNet blocks



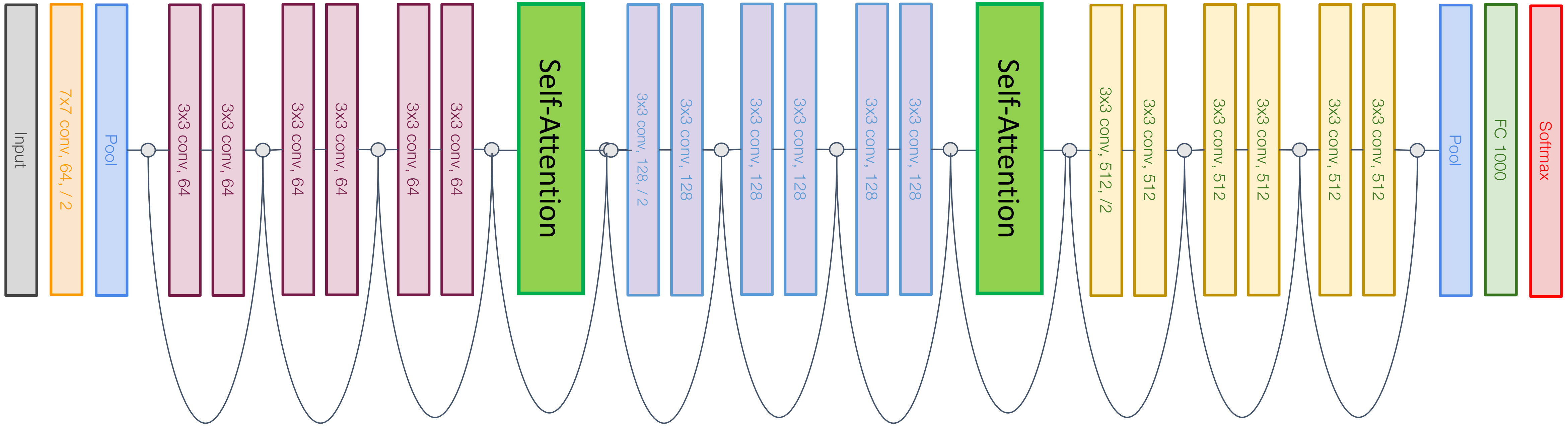
Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018
Wang et al, "Non-local Neural Networks", CVPR 2018

Idea #1: Add attention to existing CNNs

Model is still a CNN!
Can we replace convolution entirely?

Start from standard CNN architecture (e.g. ResNet)

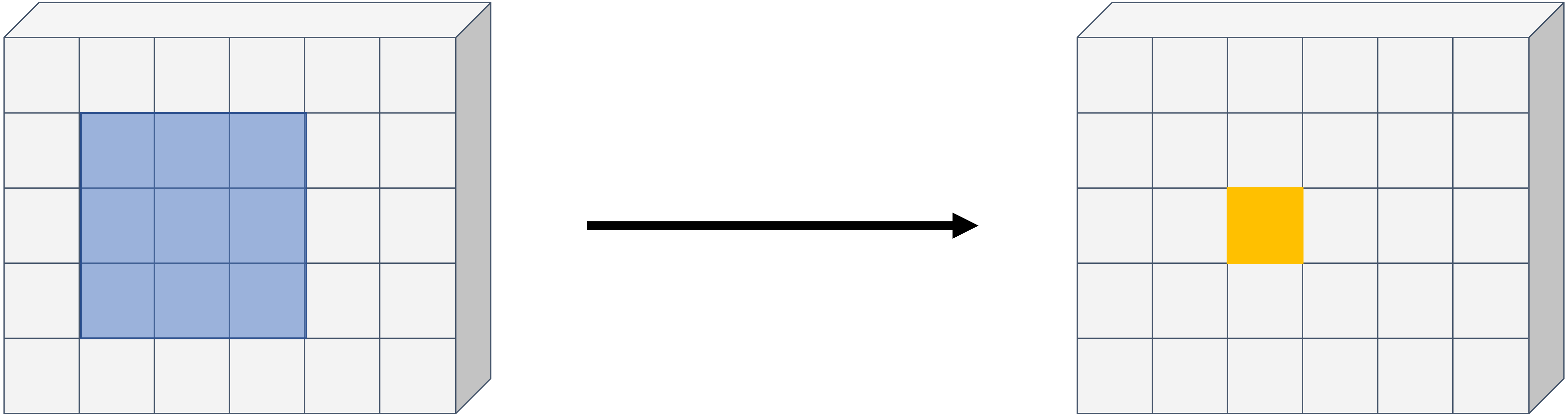
Add Self-Attention blocks between existing ResNet blocks



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018
Wang et al, "Non-local Neural Networks", CVPR 2018

Idea #2: Replace Convolution with "Local Attention"

Convolution: Output at each position is inner product of conv kernel with receptive field in input



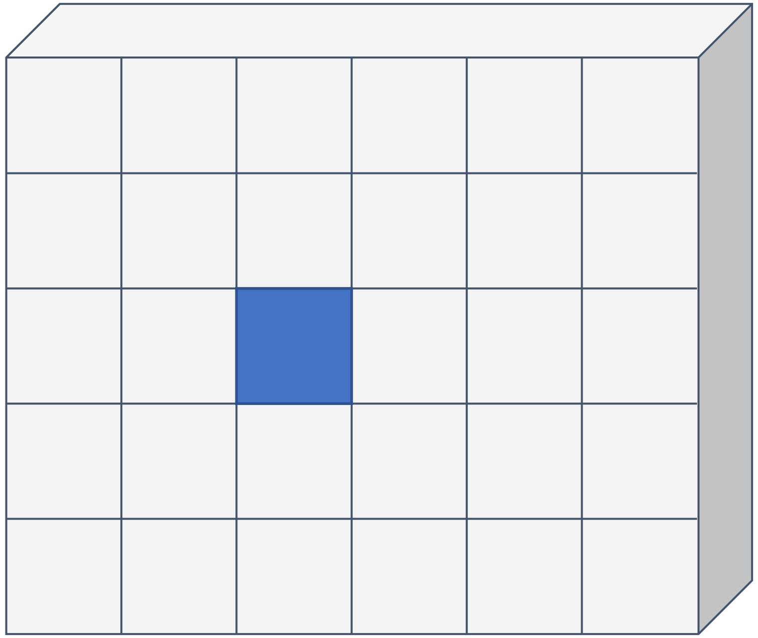
Input: C x H x W

Output: C' x H x W

Hu et al, "Local Relation Networks for Image Recognition", ICCV 2019; Ramachandran et al, "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

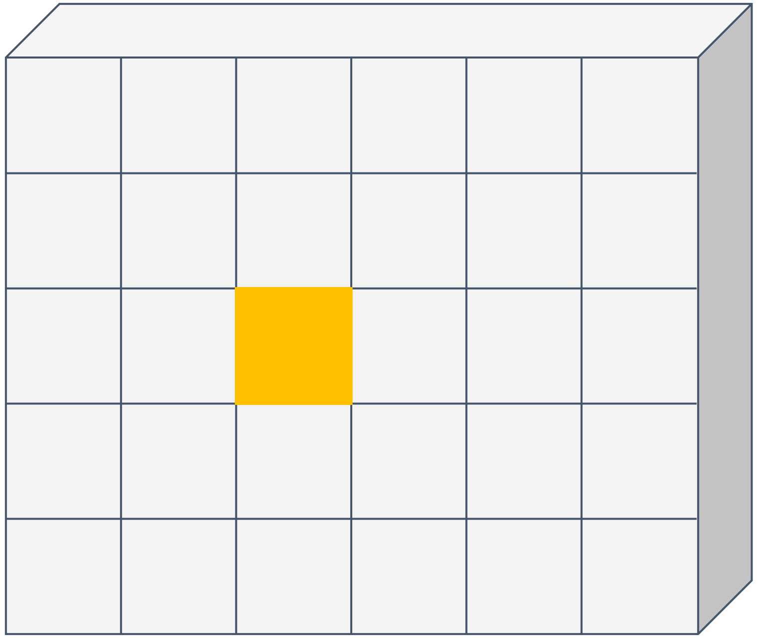
Idea #2: Replace Convolution with “Local Attention”

Map center of receptive field to **query**



Query: D_Q

Input: C x H x W



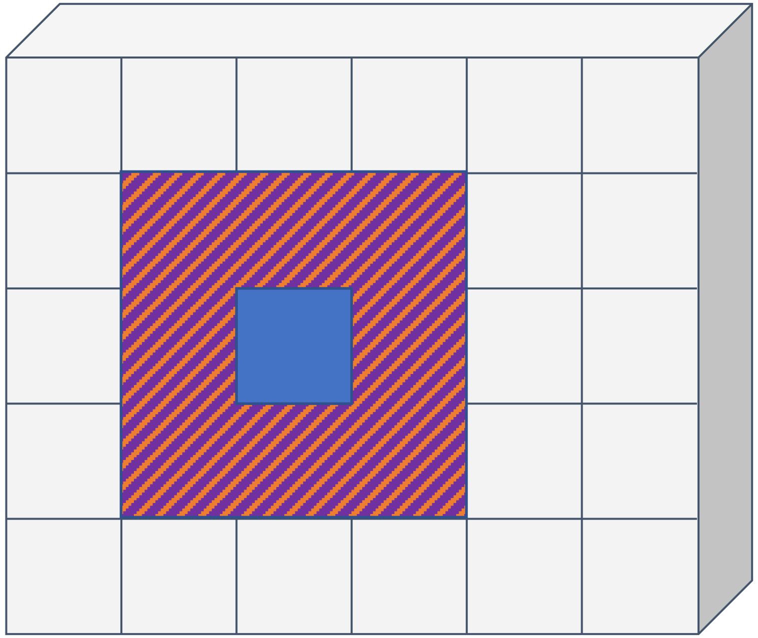
Output: C’ x H x W

Hu et al, “Local Relation Networks for Image Recognition”, ICCV 2019; Ramachandran et al, “Stand-Alone Self-Attention in Vision Models”, NeurIPS 2019

Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to **query**

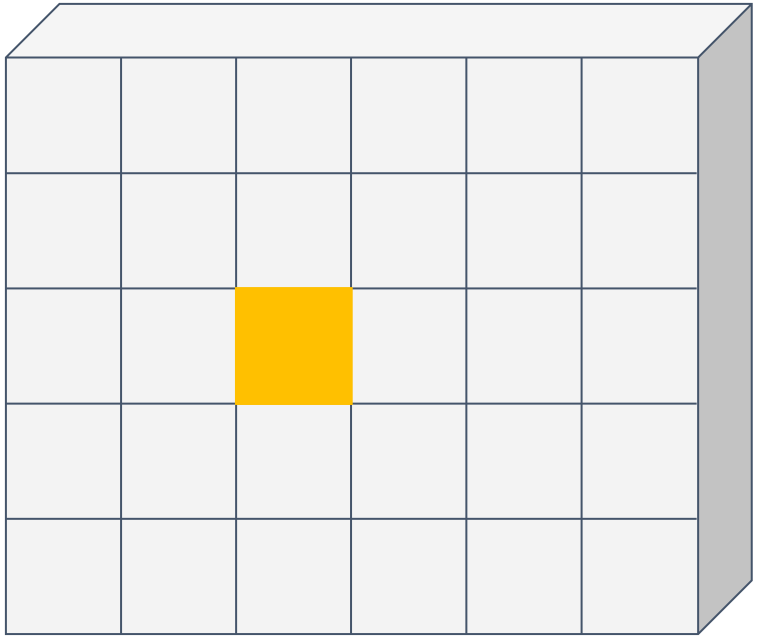
Map each element in receptive field to **key** and **value**



Query: D_Q

Keys: $R \times R \times D_Q$

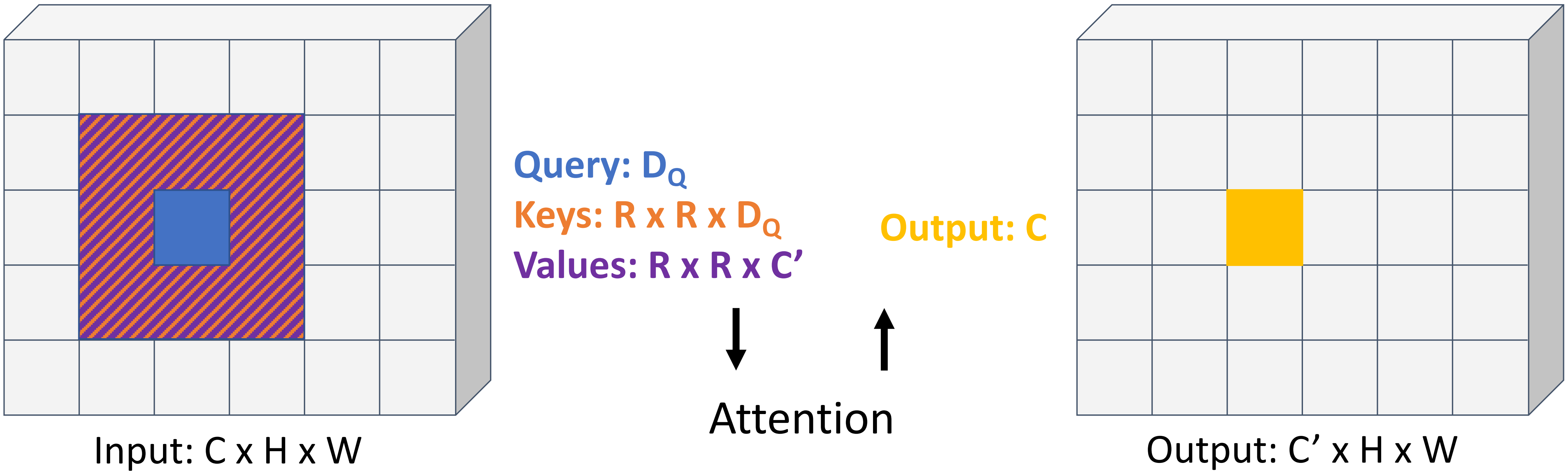
Values: $R \times R \times C'$



Hu et al, "Local Relation Networks for Image Recognition", ICCV 2019; Ramachandran et al, "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #2: Replace Convolution with “Local Attention”

Map center of receptive field to **query**
Map each element in receptive field to **key** and **value**
Compute **output** using attention



Hu et al, “Local Relation Networks for Image Recognition”, ICCV 2019; Ramachandran et al, “Stand-Alone Self-Attention in Vision Models”, NeurIPS 2019

Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to **query**
Map each element in receptive field to **key** and **value**
Compute **output** using attention
Replace all conv in ResNet with local attention

LR = "Local Relation"

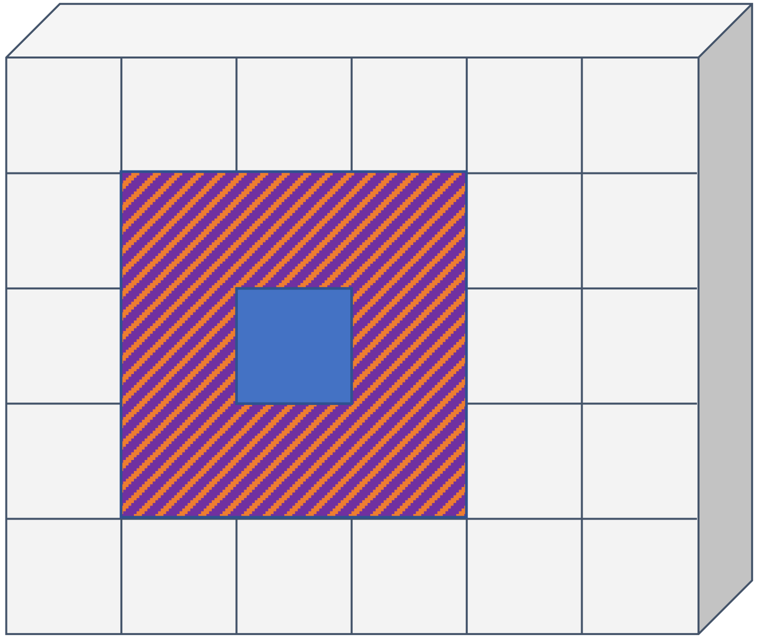
stage	output	ResNet-50	LR-Net-50 (7×7, m=8)
res1	112×112	7×7 conv, 64, stride 2	**1×1, 64** **7×7 LR, 64, stride 2**
res2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3 \text{ conv}, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 100 \\ \mathbf{7\times7\ LR, 100} \\ 1\times1, 256 \end{bmatrix} \times 3$
res3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3 \text{ conv}, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 200 \\ \mathbf{7\times7\ LR, 200} \\ 1\times1, 512 \end{bmatrix} \times 4$
res4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3 \text{ conv}, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 400 \\ \mathbf{7\times7\ LR, 400} \\ 1\times1, 1024 \end{bmatrix} \times 6$
res5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3 \text{ conv}, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 800 \\ \mathbf{7\times7\ LR, 800} \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params		**25.5**$\times 10^6$	**23.3**$\times 10^6$
FLOPs		**4.3**$\times 10^9$	**4.3**$\times 10^9$

Hu et al, "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al, "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to **query**
Map each element in receptive field to **key** and **value**
Compute **output** using attention
Replace all conv in ResNet with local attention

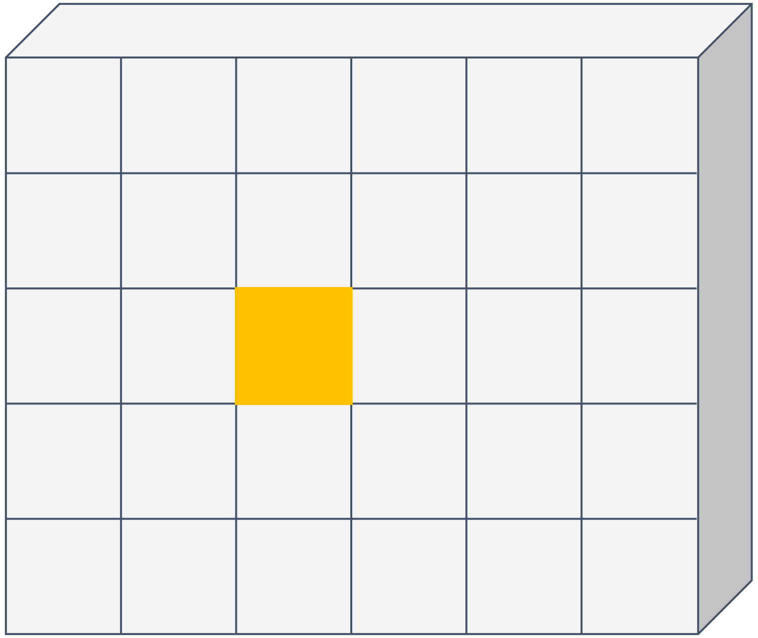
Lots of tricky details, hard to implement, only marginally better than ResNets



Query: D_Q
Keys: R x R x D_Q
Values: R x R x C'

Attention

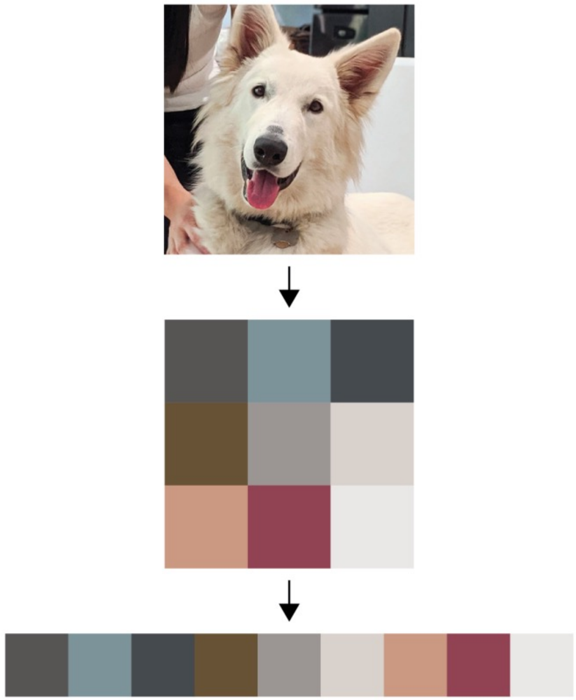
Output: C

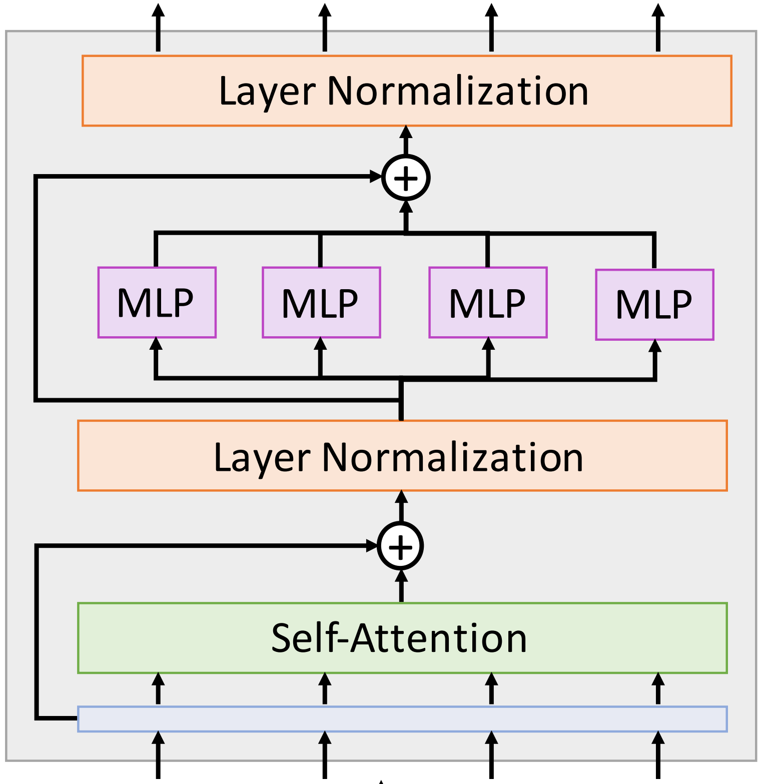


Hu et al, "Local Relation Networks for Image Recognition", ICCV 2019; Ramachandran et al, "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #3: Standard Transformer on Pixels

Treat an image as a set of pixel values



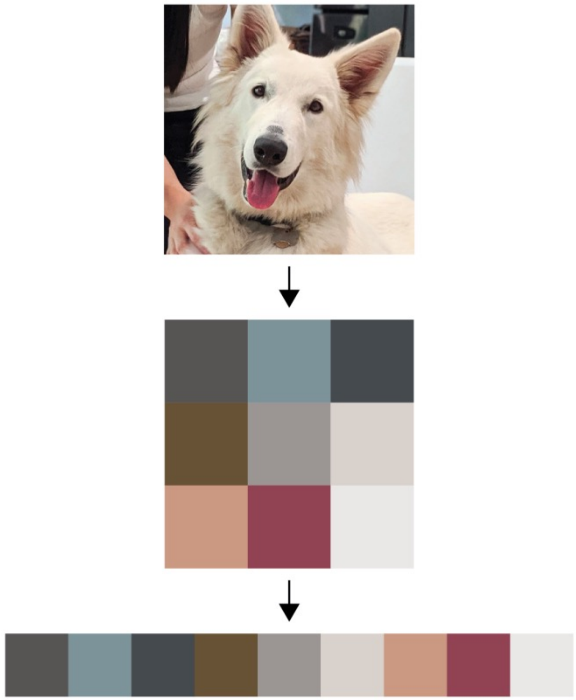


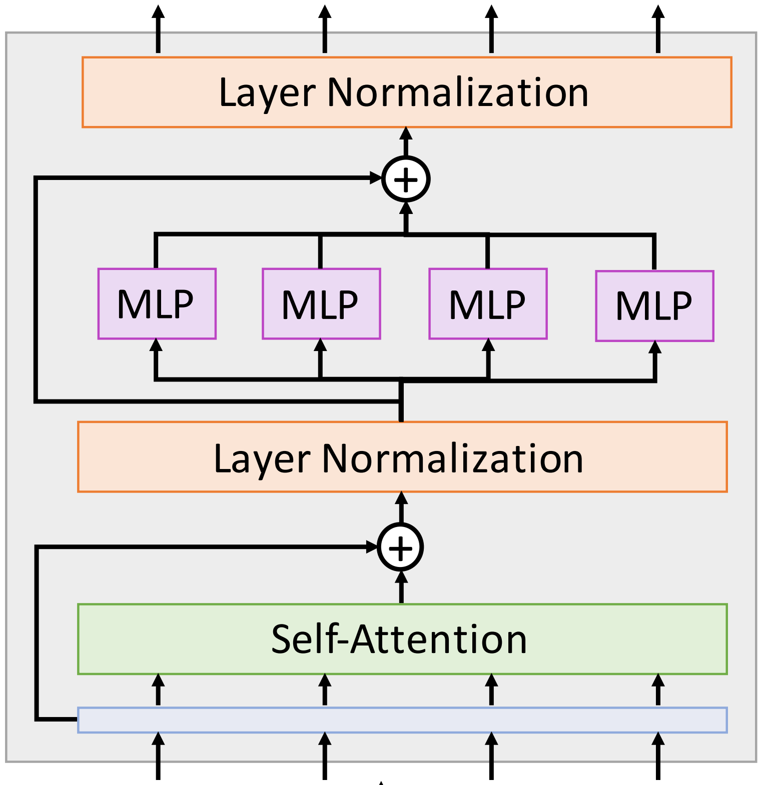
Feed as input to standard Transformer

Chen et al, "Generative Pretraining from Pixels", ICML 2020

Idea #3: Standard Transformer on Pixels

Treat an image as a set of pixel values





Feed as input to standard Transformer

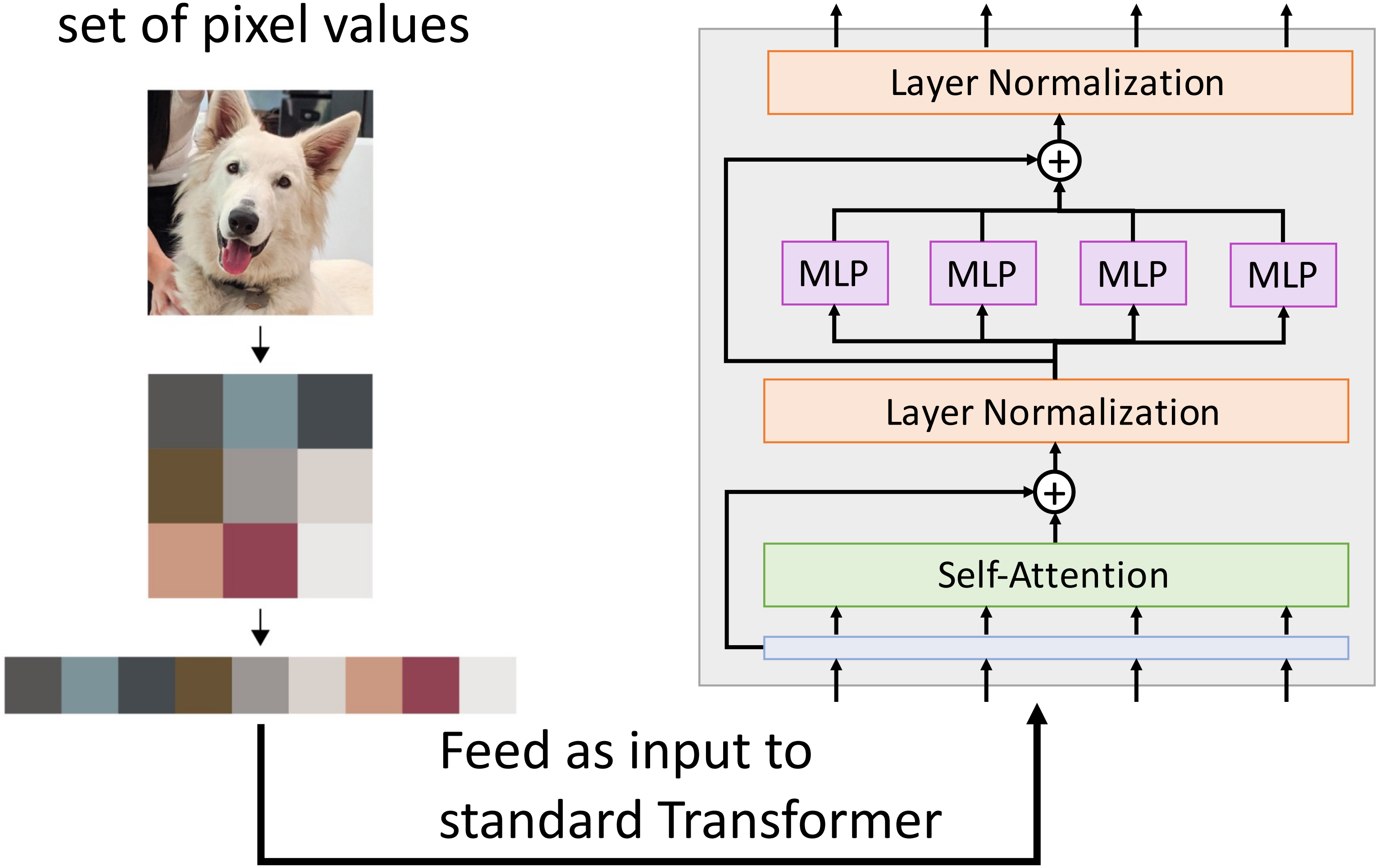
Problem: Memory use!

R x R image needs R^4 elements per attention matrix

Chen et al, “Generative Pretraining from Pixels”, ICML 2020

Idea #3: Standard Transformer on Pixels

Treat an image as a set of pixel values



Problem: Memory use!

R x R image needs R^4 elements per attention matrix

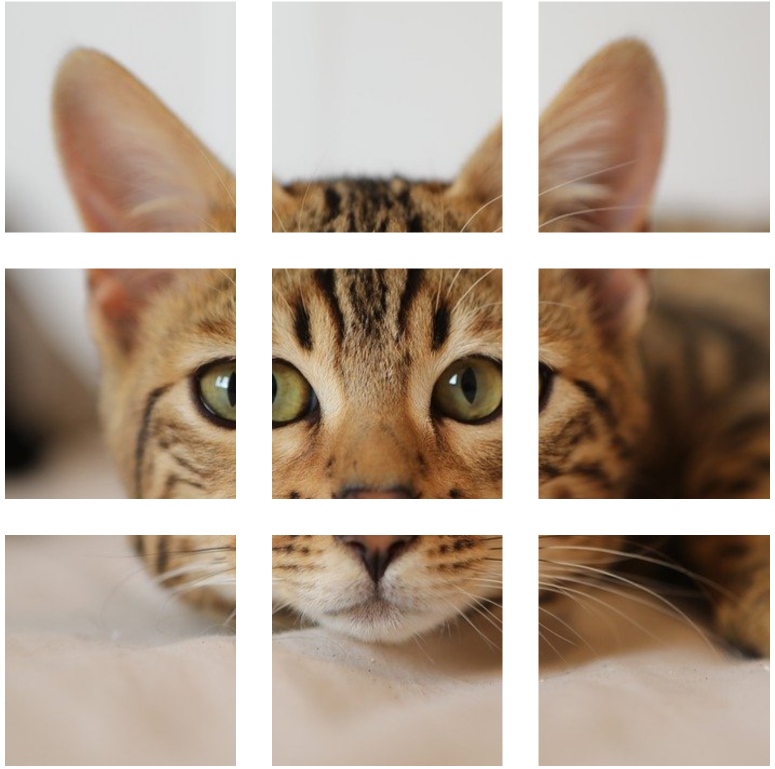
R=128, 48 layers, 16 heads per layer takes 768GB of memory for attention matrices for a single example…

Chen et al, "Generative Pretraining from Pixels", ICML 2020

Idea #4: Standard Transformer on Patches

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

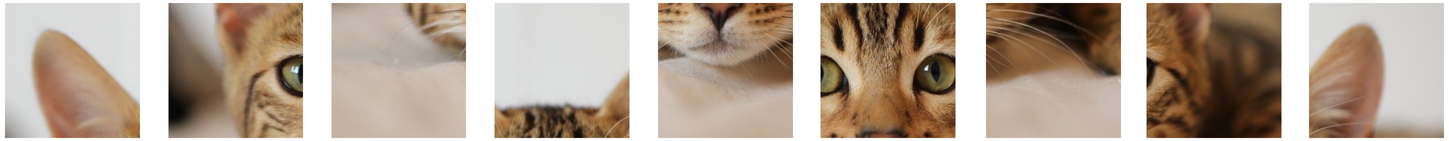
Idea #4: Standard Transformer on Patches



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

Idea #4: Standard Transformer on Patches

N input patches, each of shape 3x16x16

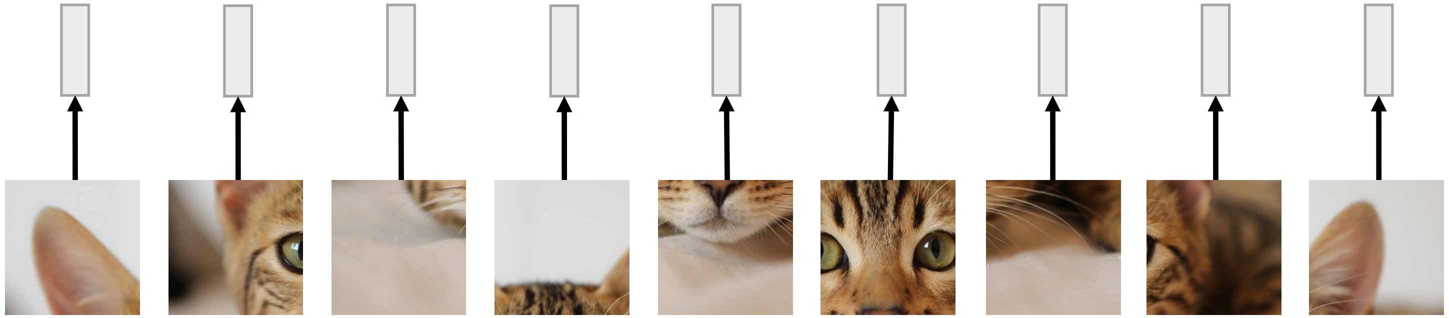


Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Idea #4: Standard Transformer on Patches

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16



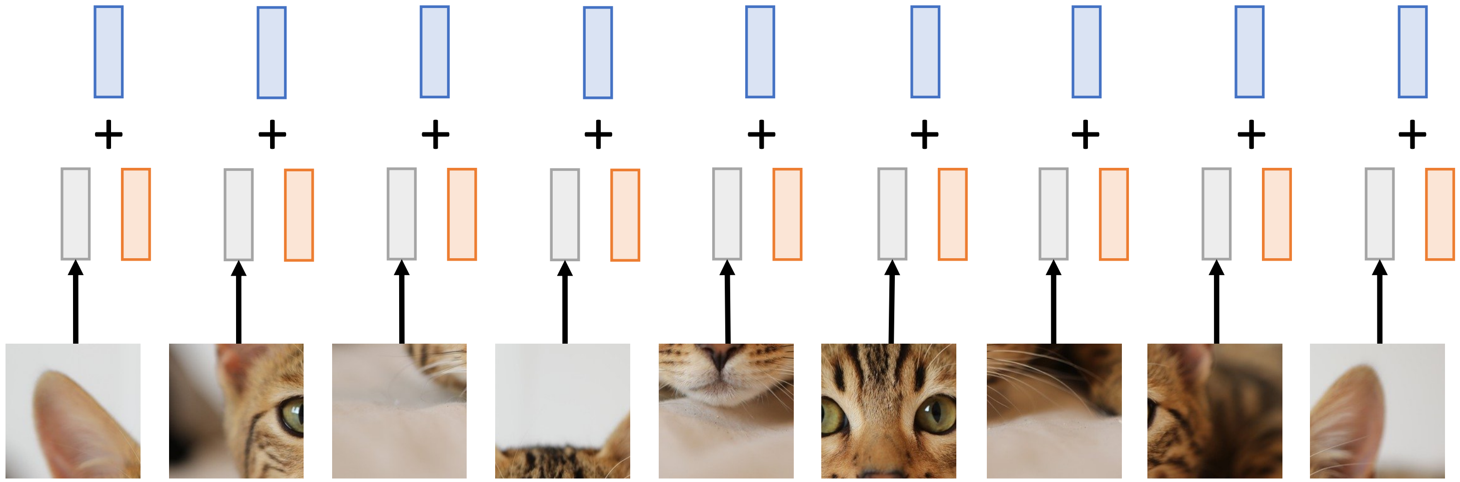
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches

Add positional embedding: learned D-dim vector per position

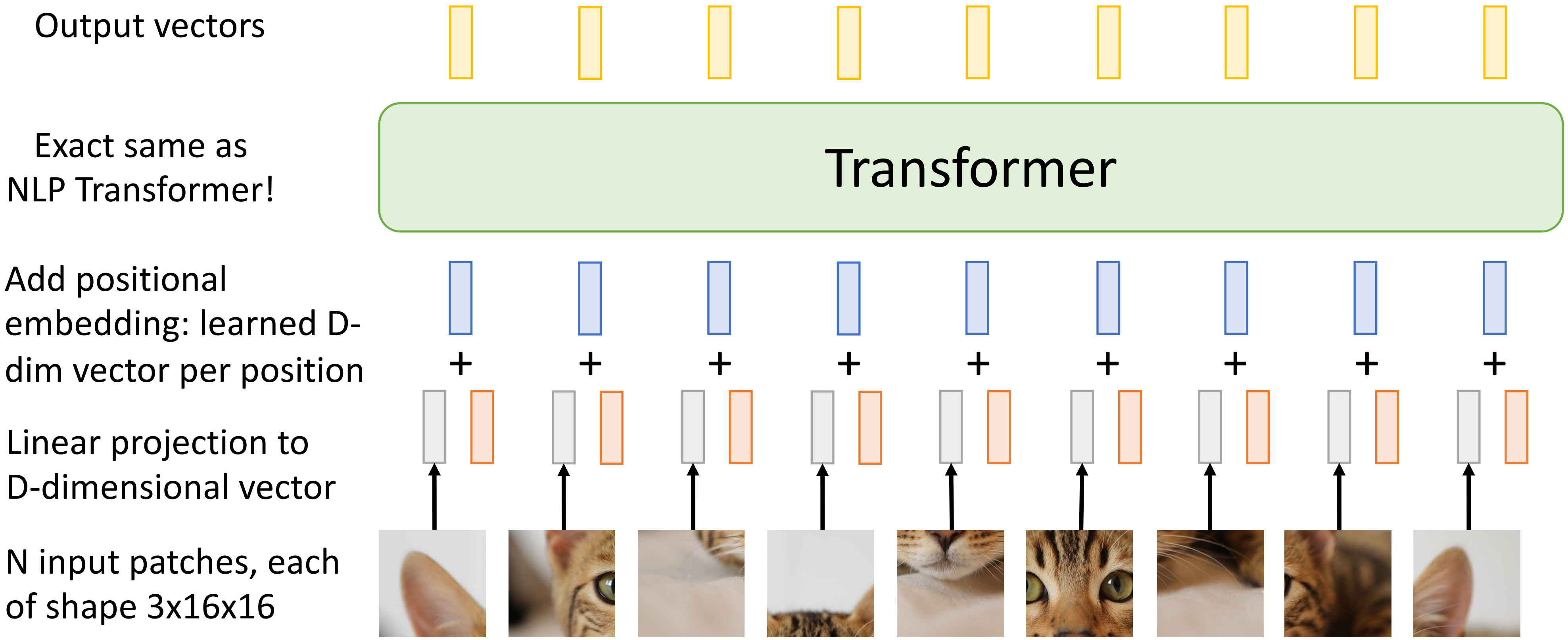
Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16



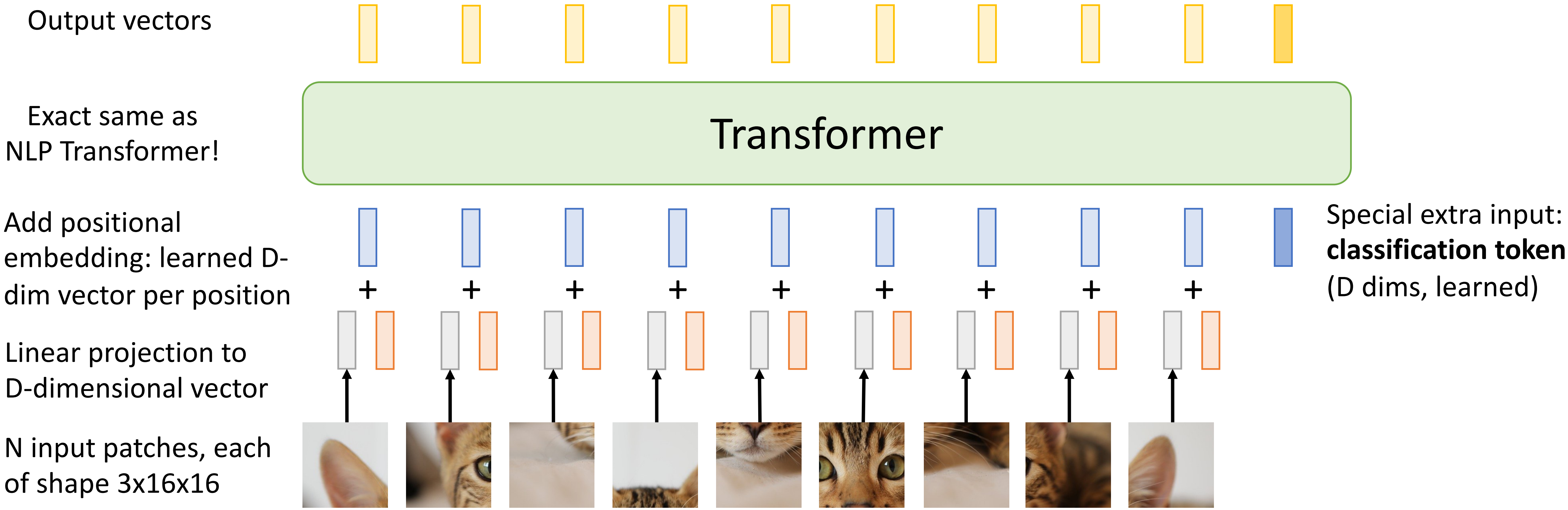
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Idea #4: Standard Transformer on Patches



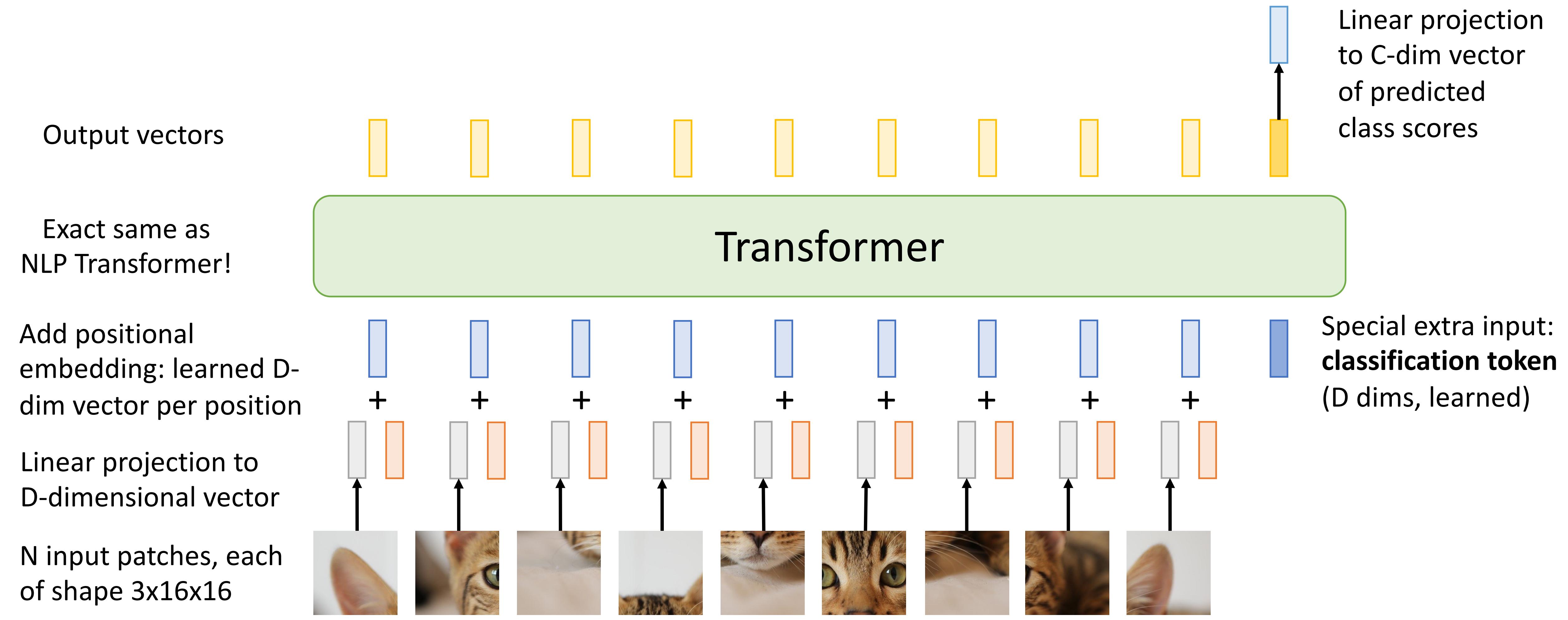
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Idea #4: Standard Transformer on Patches



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

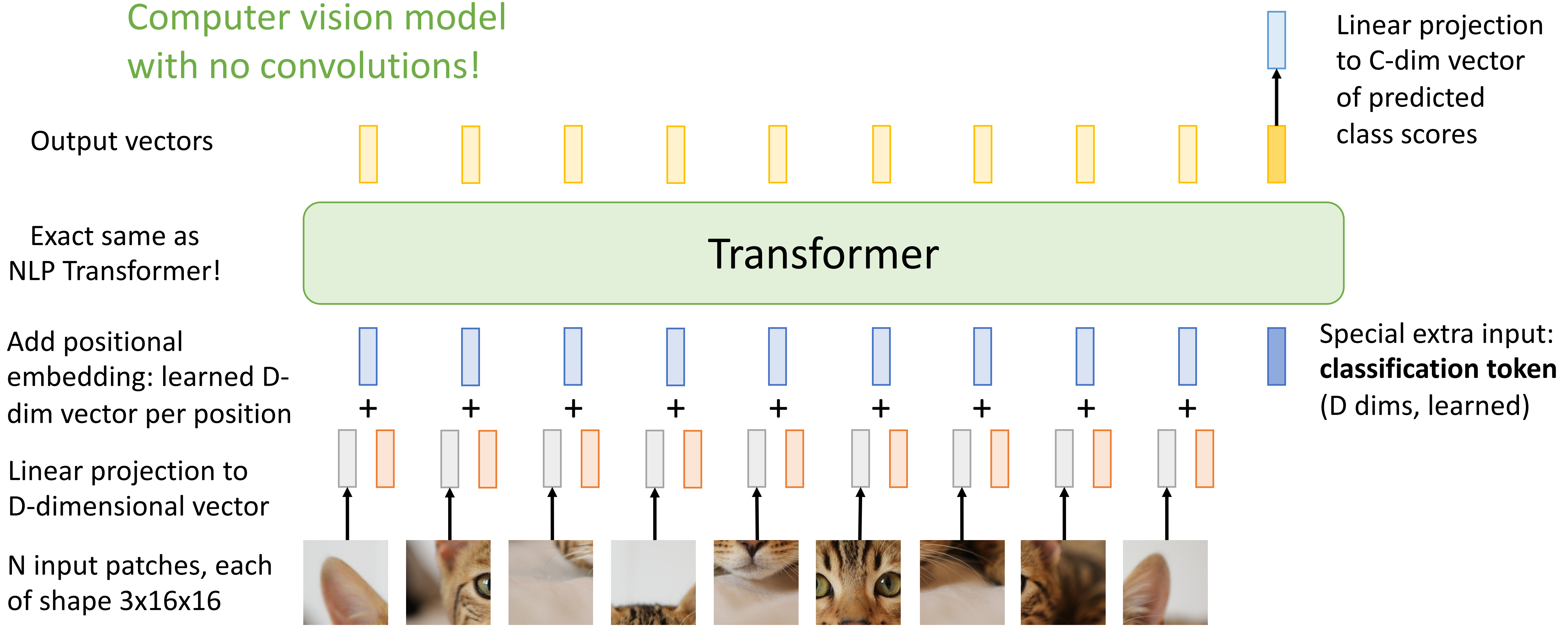
Idea #4: Standard Transformer on Patches



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

Vision Transformer (ViT)

Computer vision model with no convolutions!

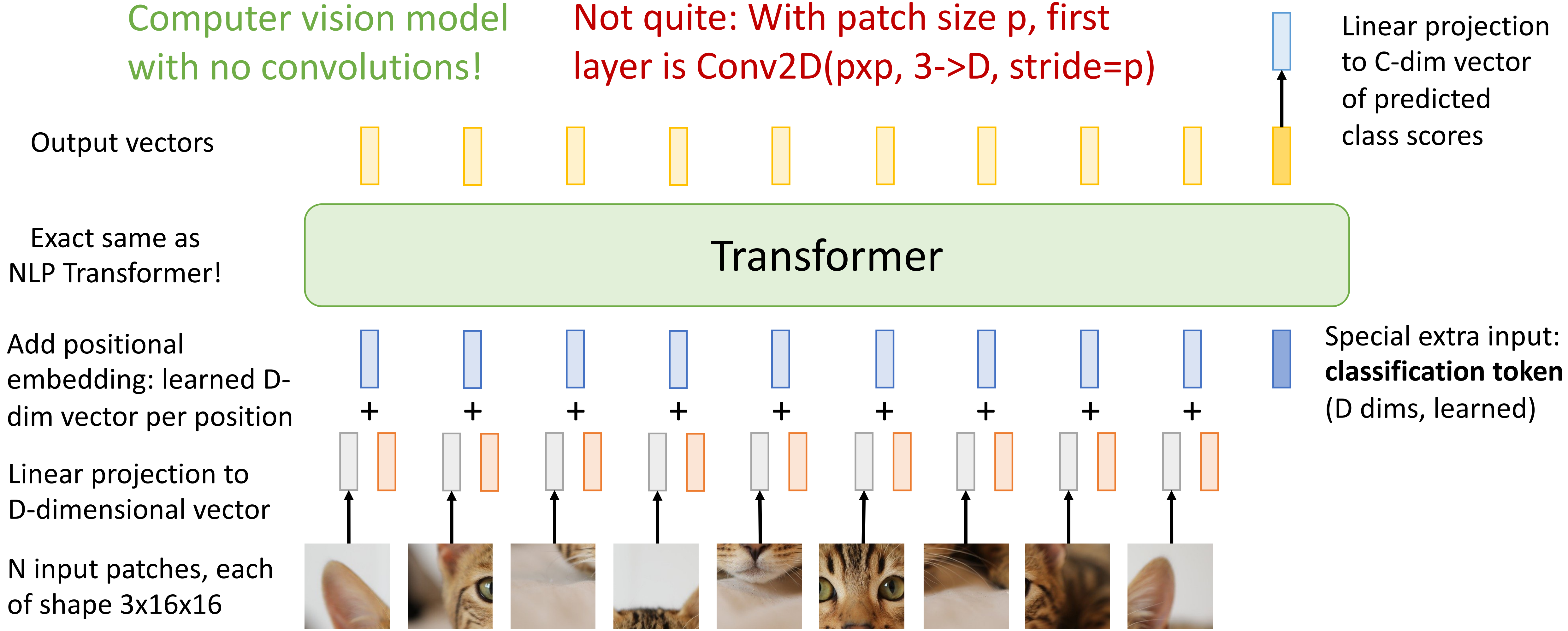


Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Vision Transformer (ViT)

Computer vision model with no convolutions!

Not quite: With patch size p, first layer is Conv2D(pxp, 3->D, stride=p)



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

Vision Transformer (ViT)

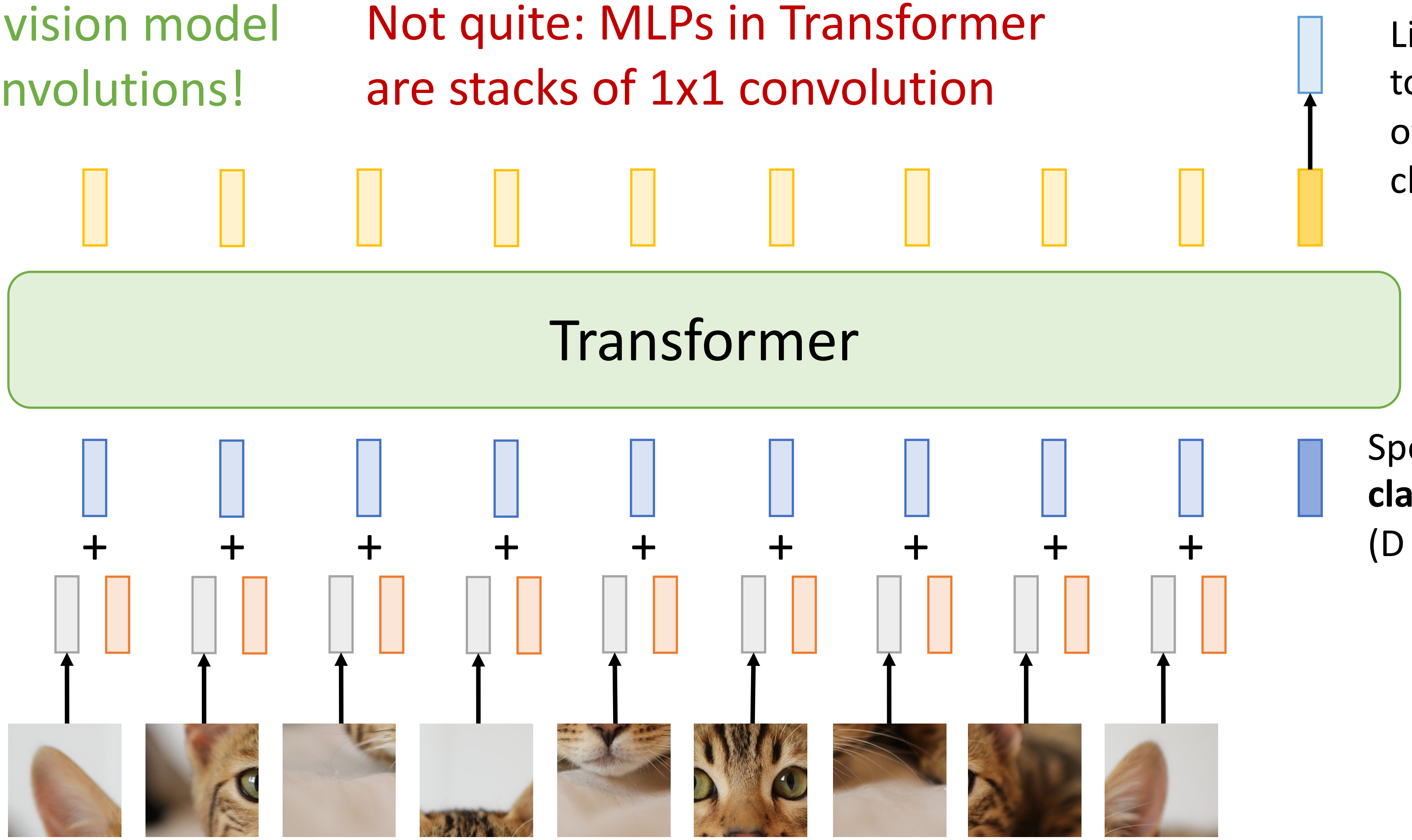
Computer vision model with no convolutions!

Not quite: MLPs in Transformer are stacks of 1x1 convolution

Linear projection to C-dim vector of predicted class scores

Output vectors

Exact same as NLP Transformer!



Special extra input: **classification token** (D dims, learned)

Add positional embedding: learned D-dim vector per position

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16

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

Vision Transformer (ViT)

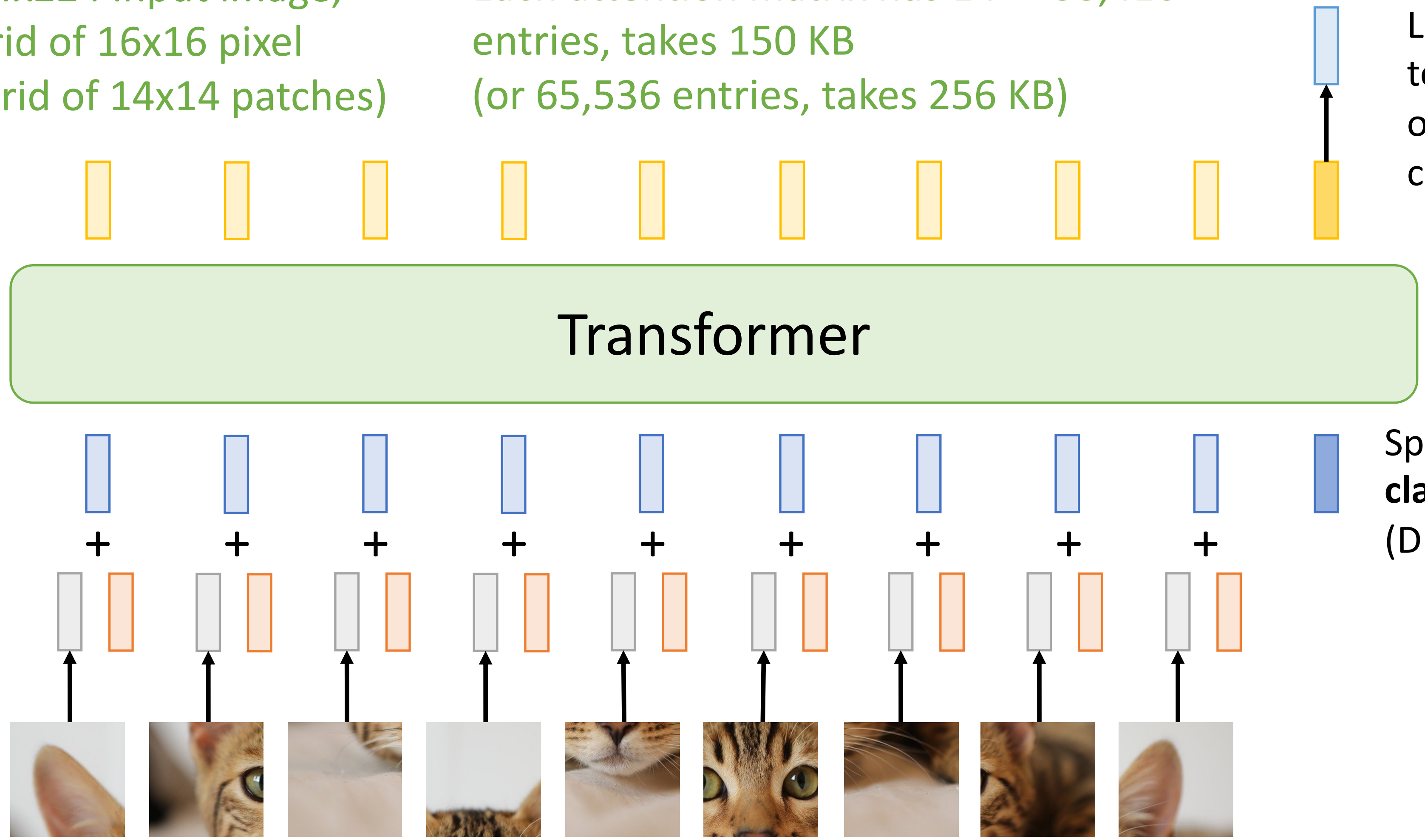
In practice: take 224x224 input image, divide into 14x14 grid of 16x16 pixel patches (or 16x16 grid of 14x14 patches)

Each attention matrix has $14^4 = 38{,}416$ entries, takes 150 KB (or 65,536 entries, takes 256 KB)

Linear projection to C-dim vector of predicted class scores

Output vectors

Exact same as NLP Transformer!



Add positional embedding: learned D-dim vector per position

Special extra input: **classification token** (D dims, learned)

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16

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

Vision Transformer (ViT)

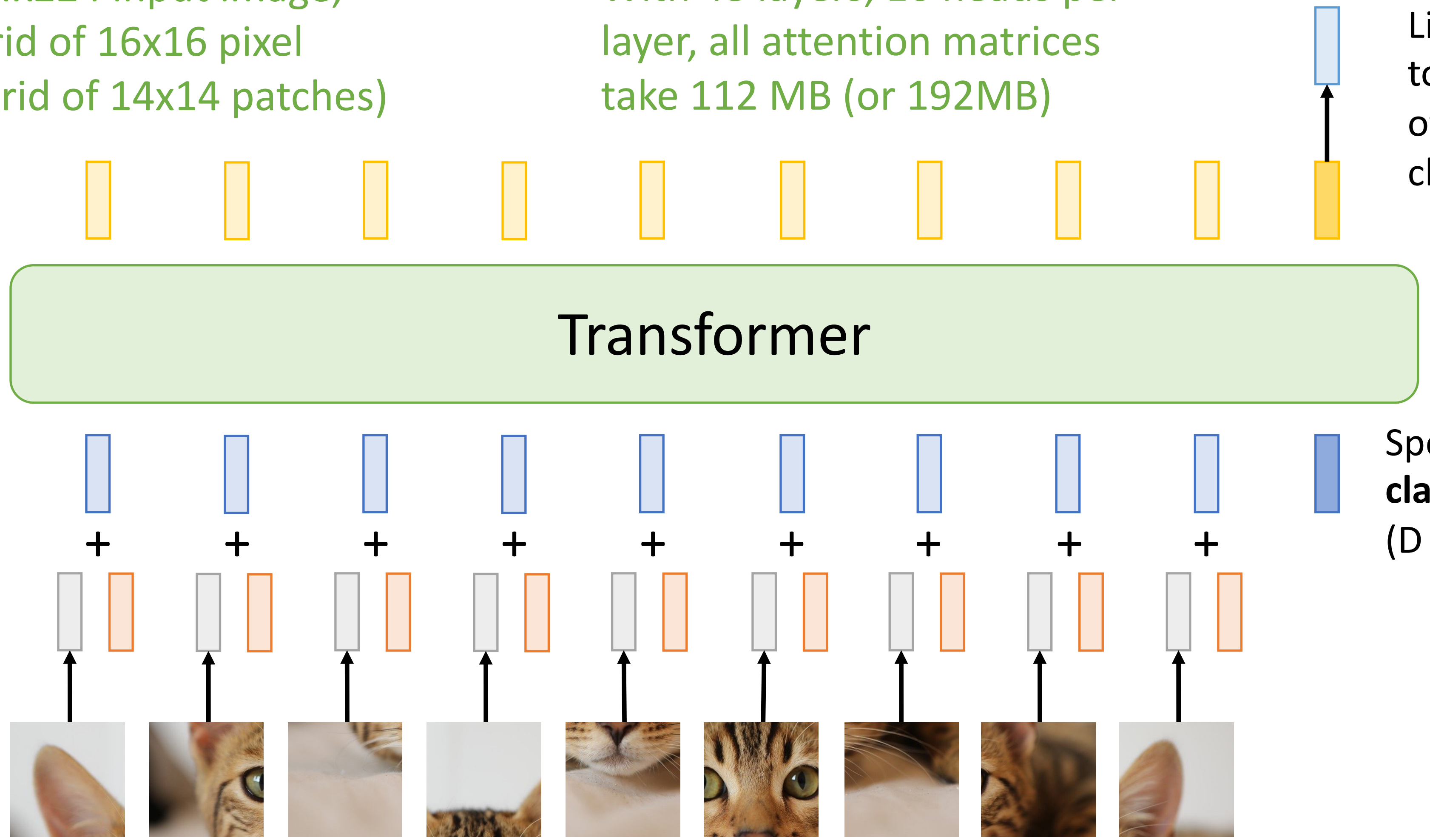
In practice: take 224x224 input image, divide into 14x14 grid of 16x16 pixel patches (or 16x16 grid of 14x14 patches)

With 48 layers, 16 heads per layer, all attention matrices take 112 MB (or 192MB)

Linear projection to C-dim vector of predicted class scores

Output vectors

Exact same as NLP Transformer!



Add positional embedding: learned D-dim vector per position

Special extra input: **classification token** (D dims, learned)

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16

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

Vision Transformer (ViT)

In practice: take 224x224 input image, divide into 14x14 grid of 16x16 pixel patches (or 16x16 grid of 14x14 patches)

With 48 layers, 16 heads per layer, all attention matrices take 112 MB (or 192MB)

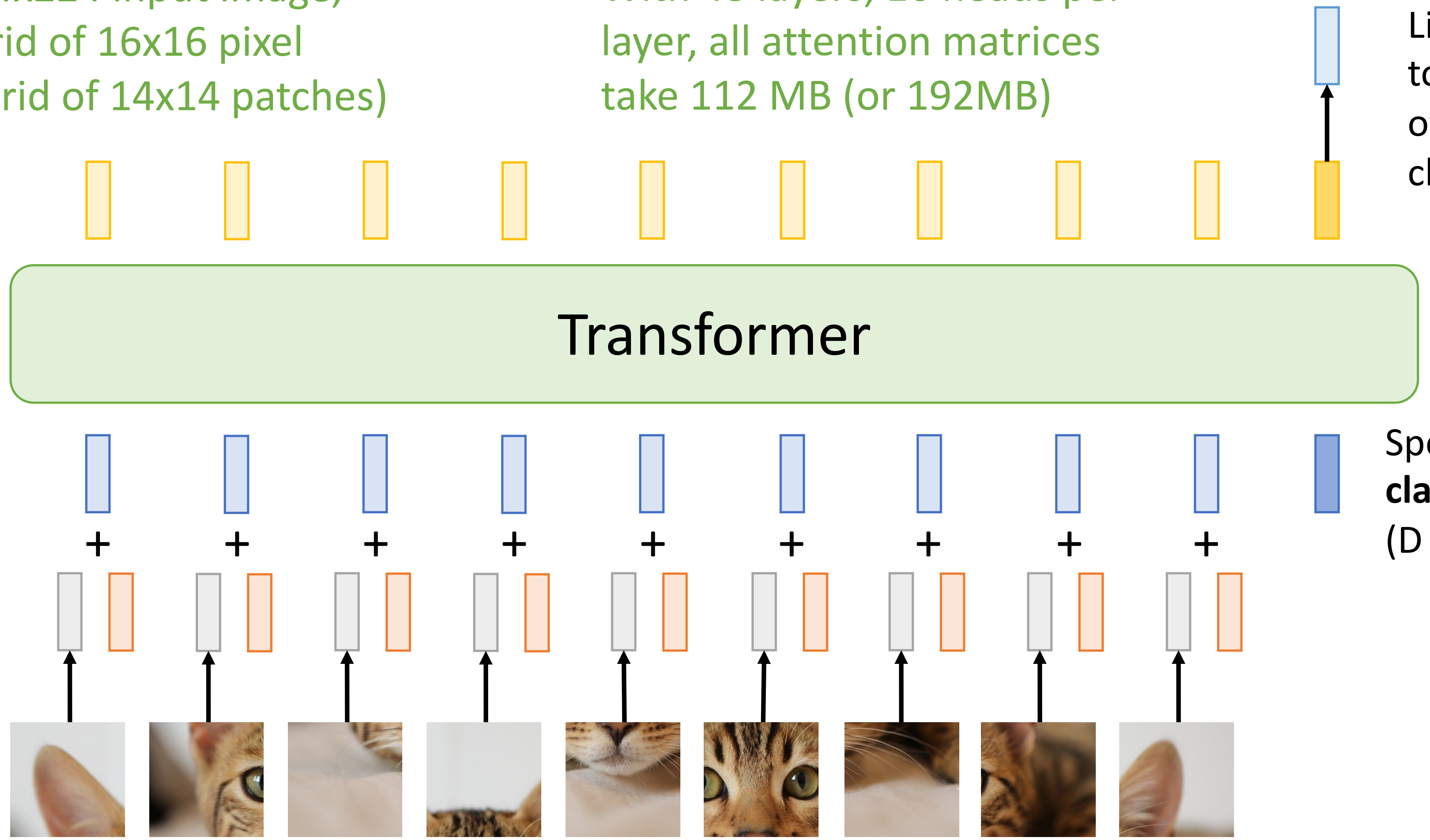
Output vectors

Exact same as NLP Transformer!

Add positional embedding: learned D-dim vector per position

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16

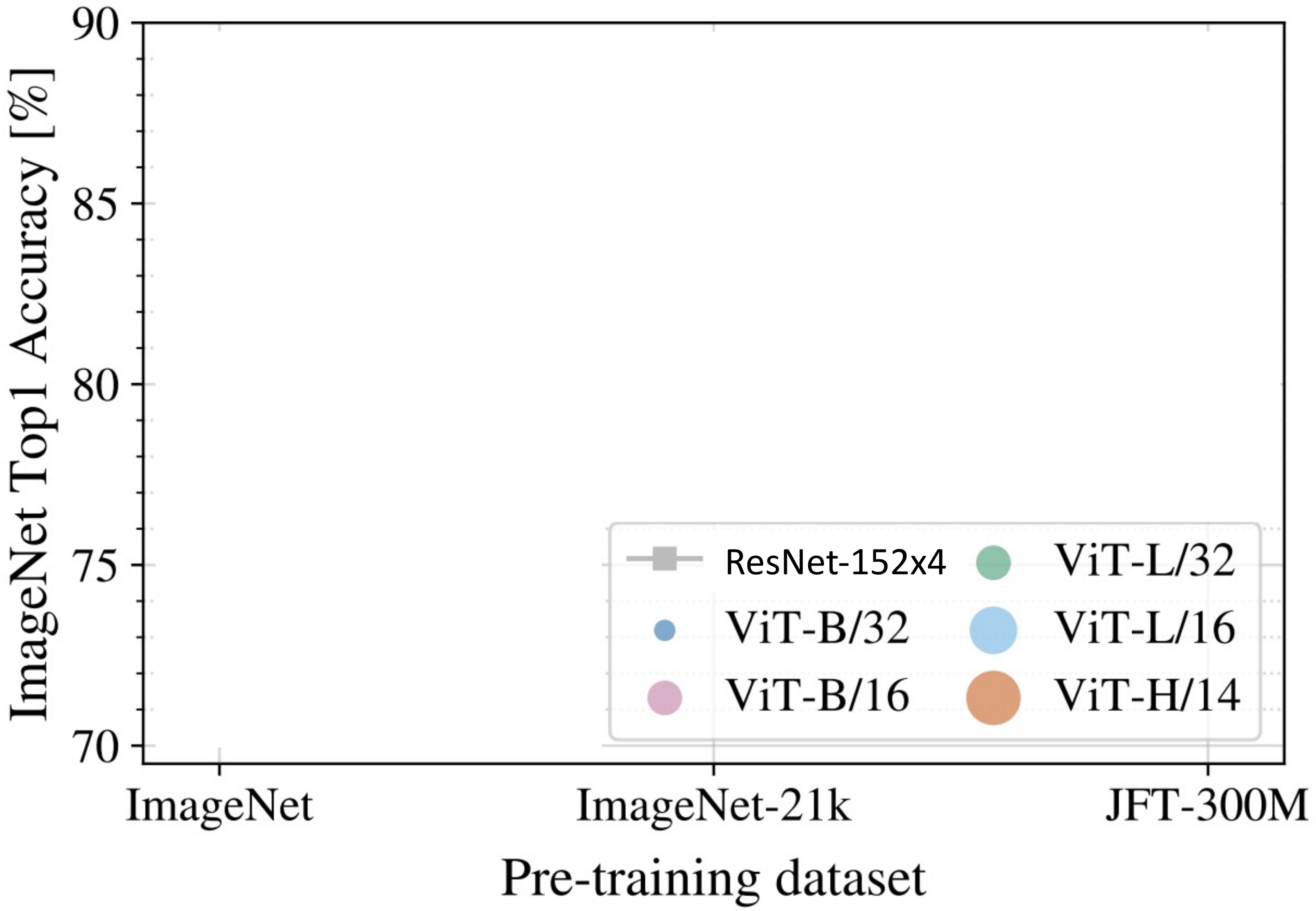


Linear projection to C-dim vector of predicted class scores

Special extra input: **classification token** (D dims, learned)

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

Vision Transformer (ViT) vs ResNets



B = Base
L = Large
H = Huge

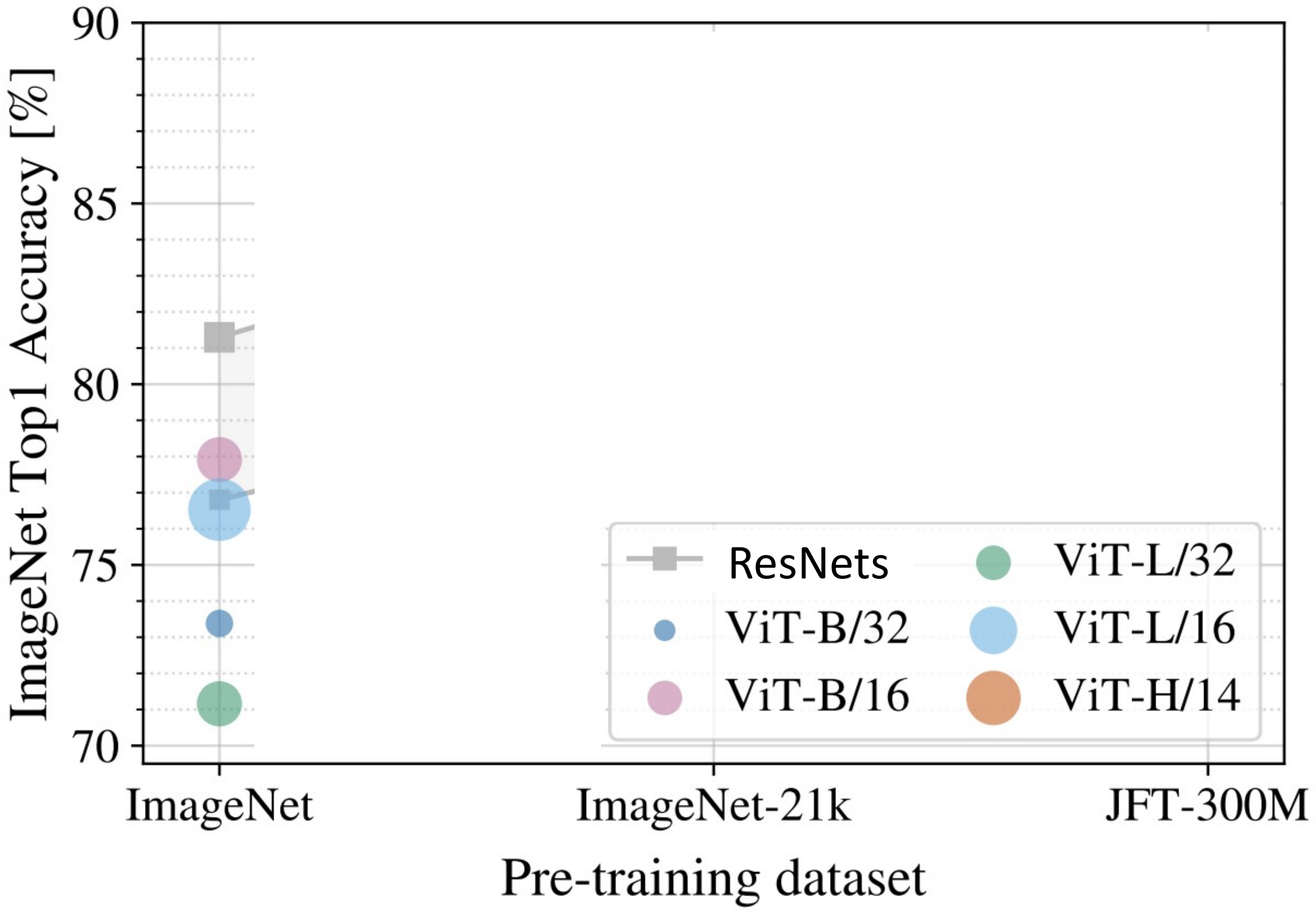
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

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

Vision Transformer (ViT) vs ResNets

Recall: ImageNet dataset has 1k categories, 1.2M images

When trained on ImageNet, ViT models perform worse than ResNets



B = Base
L = Large
H = Huge

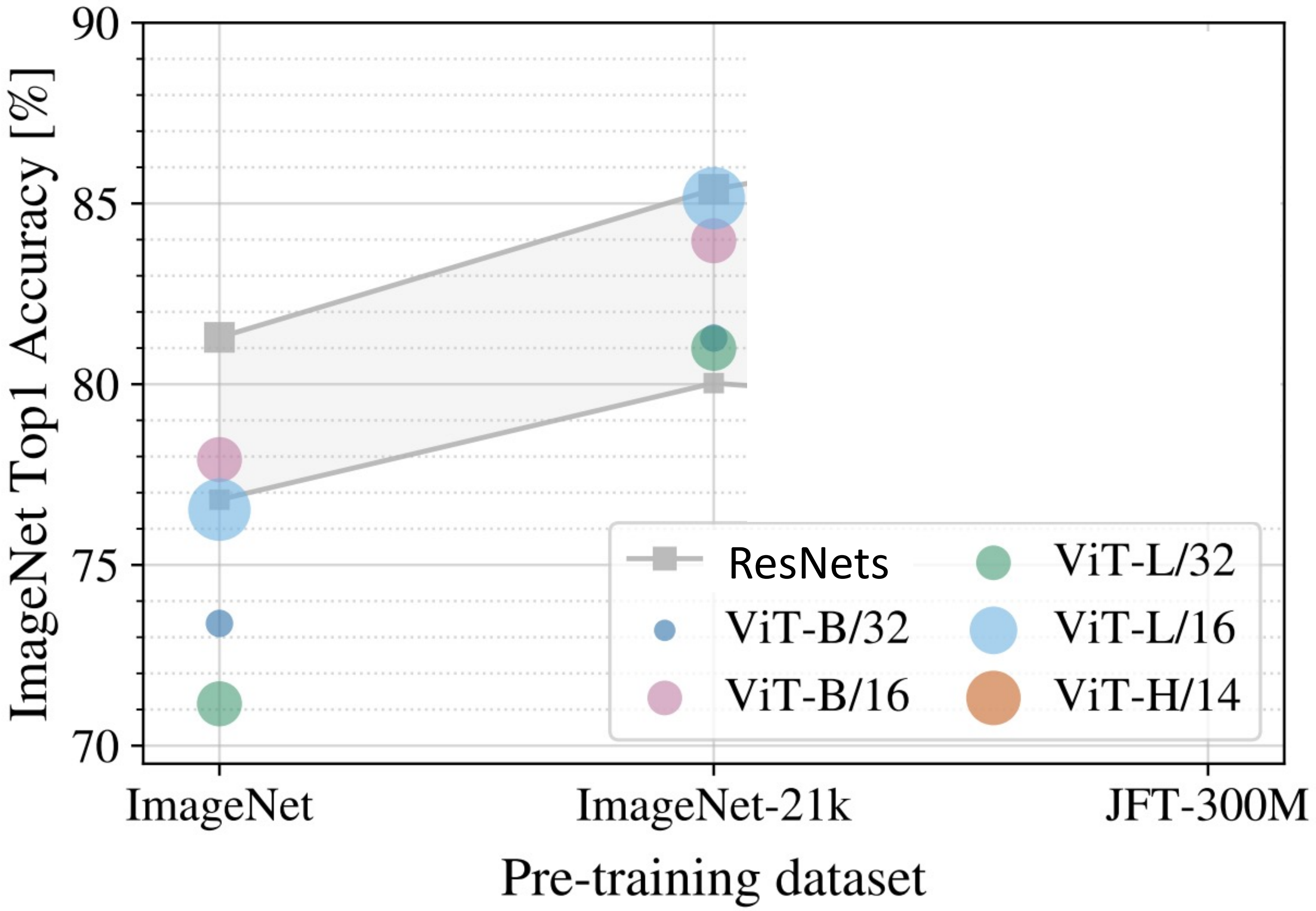
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

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

Vision Transformer (ViT) vs ResNets

ImageNet-21k has 14M images with 21k categories

If you pretrain on ImageNet-21k and fine-tune on ImageNet, ViT does better: big ViTs match big ResNets



B = Base
L = Large
H = Huge

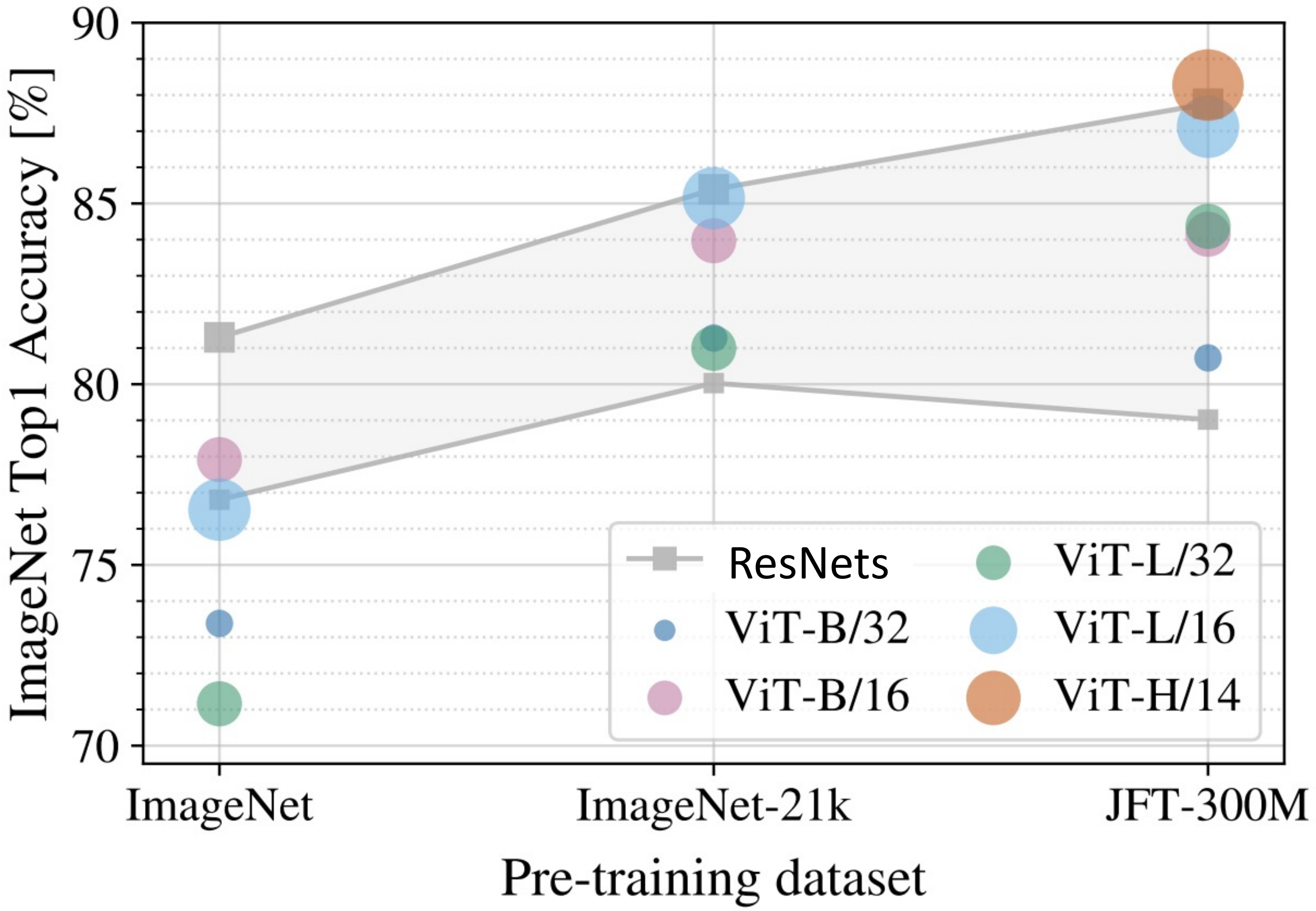
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

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

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



B = Base
L = Large
H = Huge

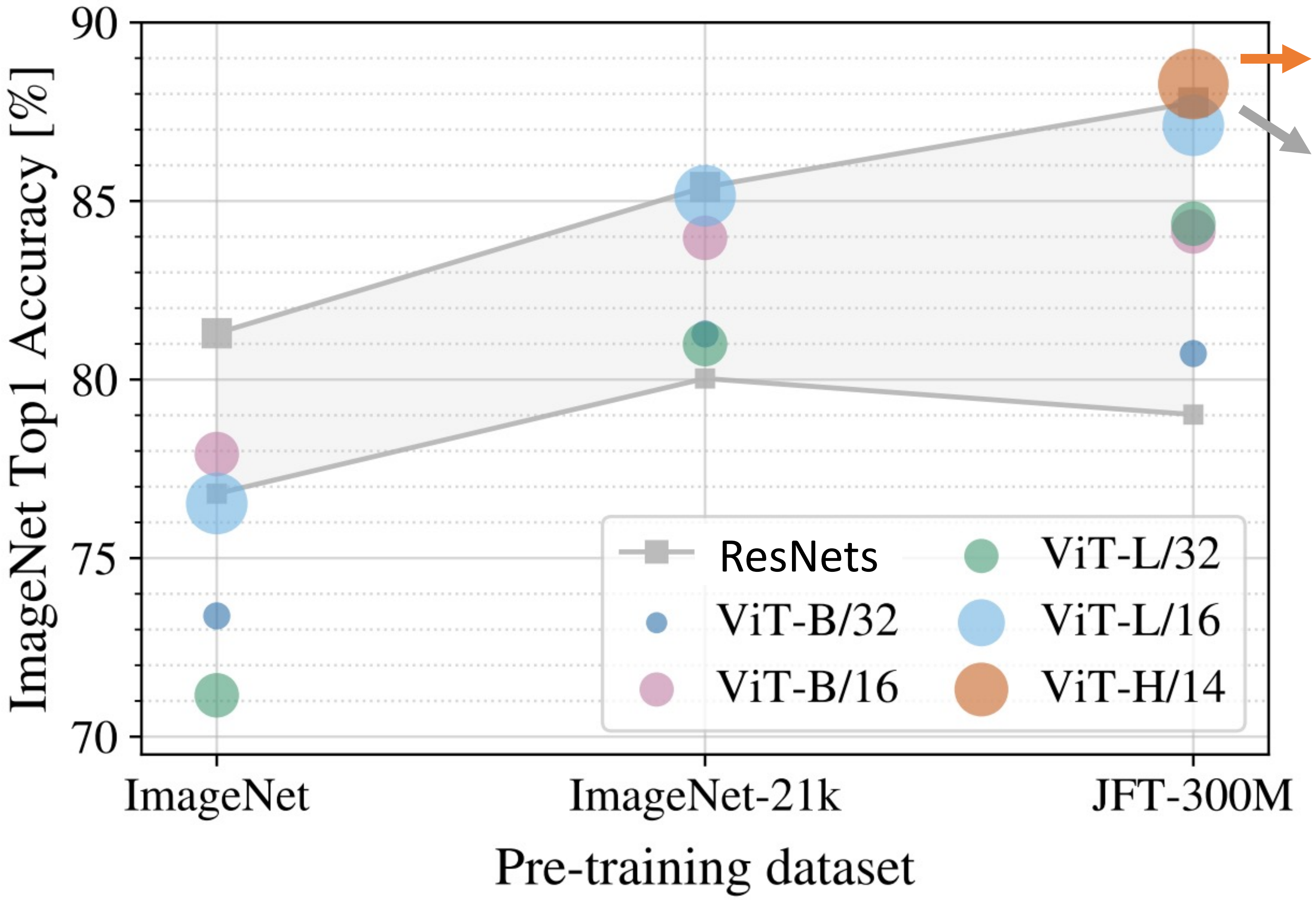
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

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

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



ViT: 2.5k TPU-v3 core days of training

ResNet: 9.9k TPU-v3 core days of training

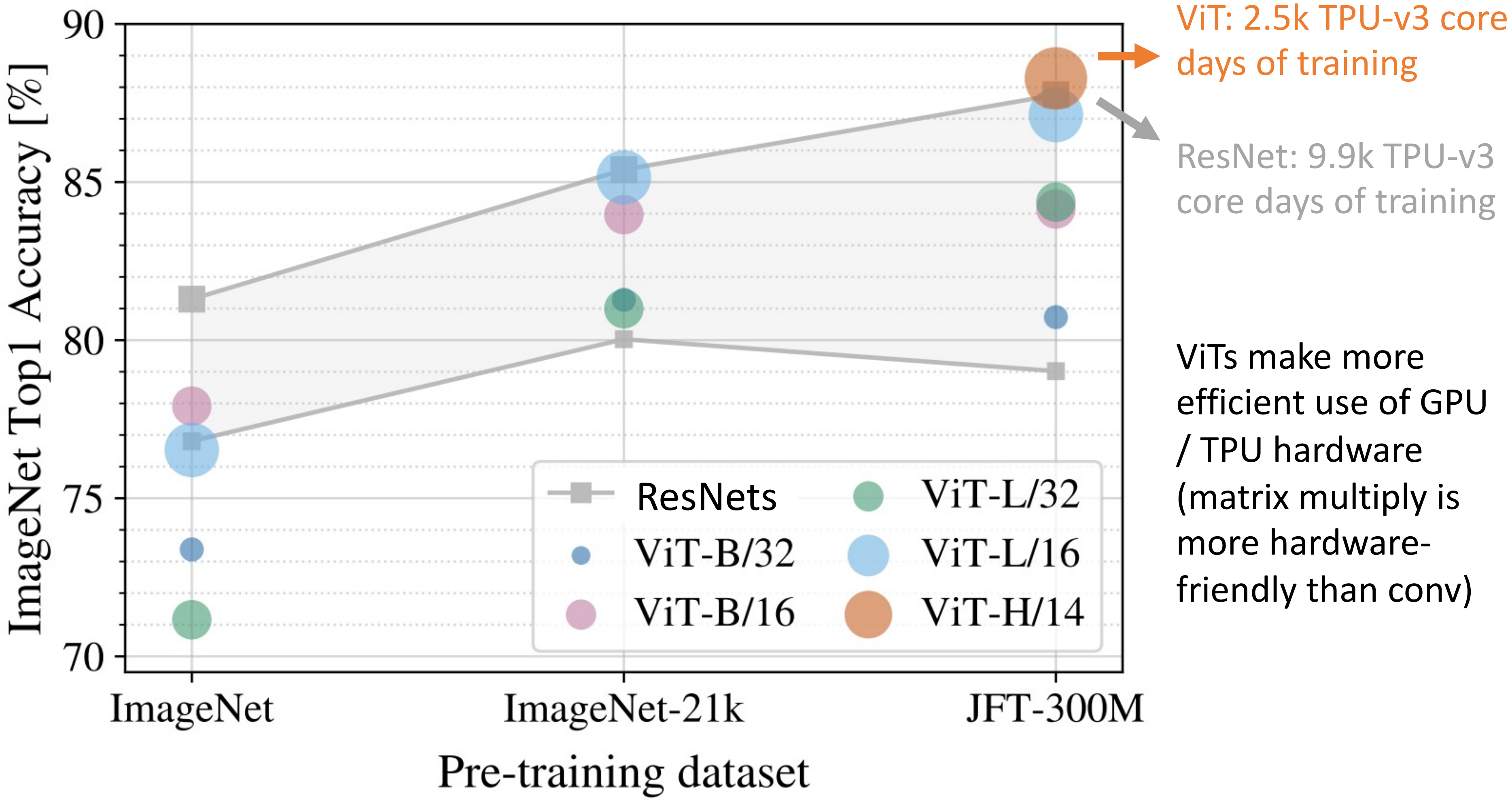
ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

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

Vision Transformer (ViT) vs ResNets

Claim: ViT models have “less inductive bias” than ResNets, so need more pretraining data to learn good features

(Not sure I buy this explanation: “inductive bias” is not a well-defined concept we can measure!)

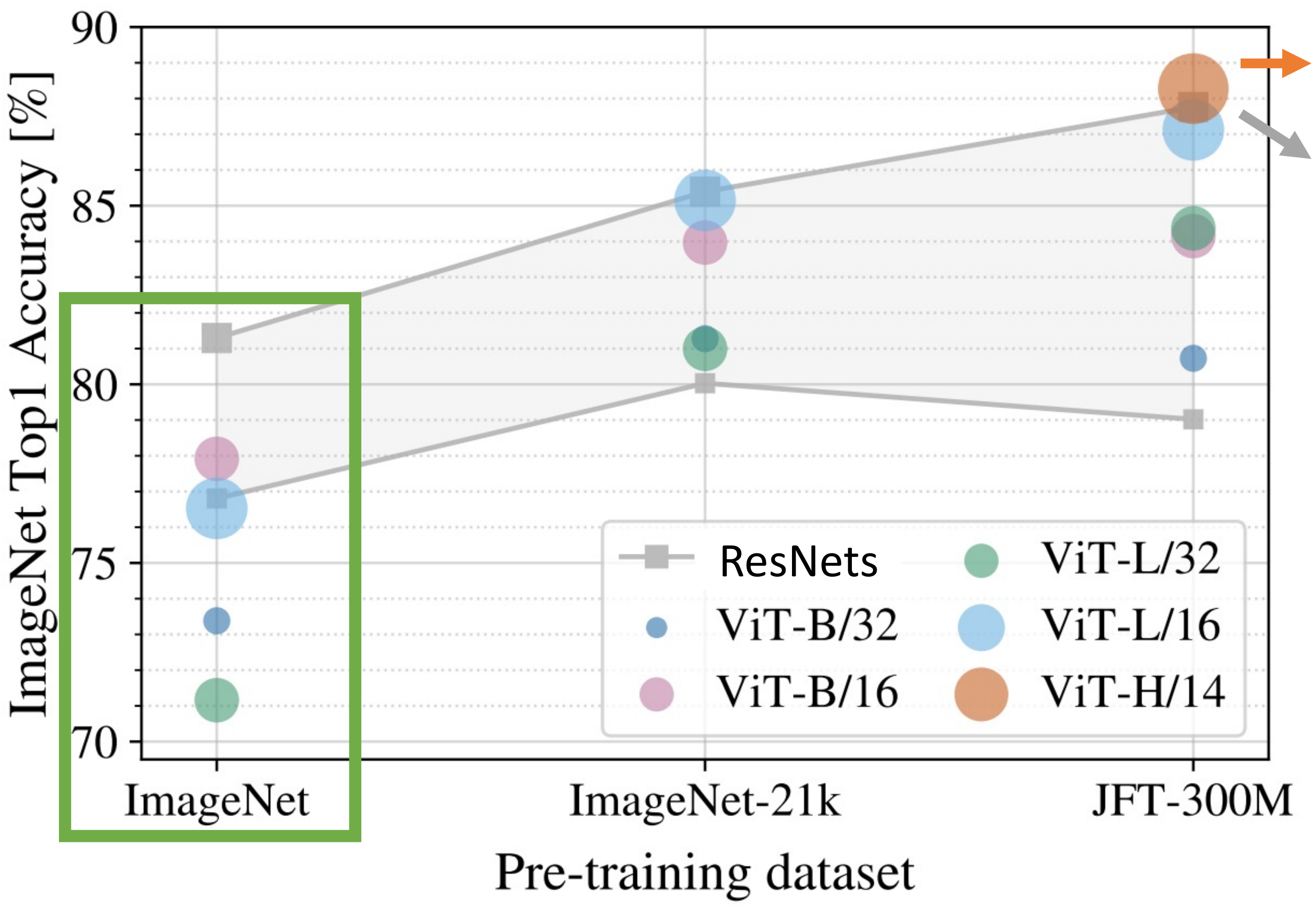


ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

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

Vision Transformer (ViT) vs ResNets

How can we improve the performance of ViT models on ImageNet?



ViT: 2.5k TPU-v3 core days of training

ResNet: 9.9k TPU-v3 core days of training

ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

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

Improving ViT: Augmentation and Regularization

Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

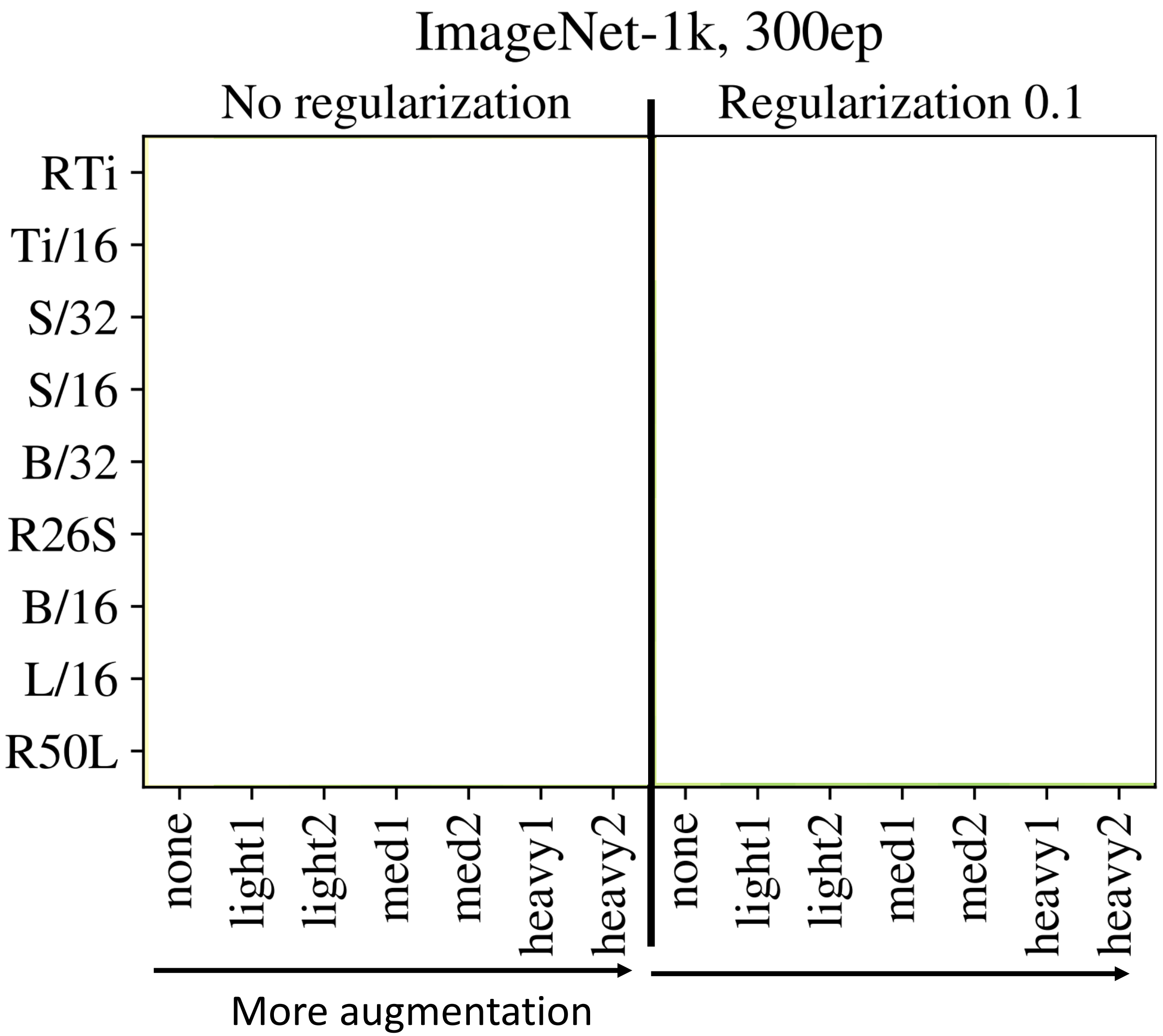
Data Augmentation for ViT models:
- MixUp
- RandAugment

Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Augmentation and Regularization

Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment



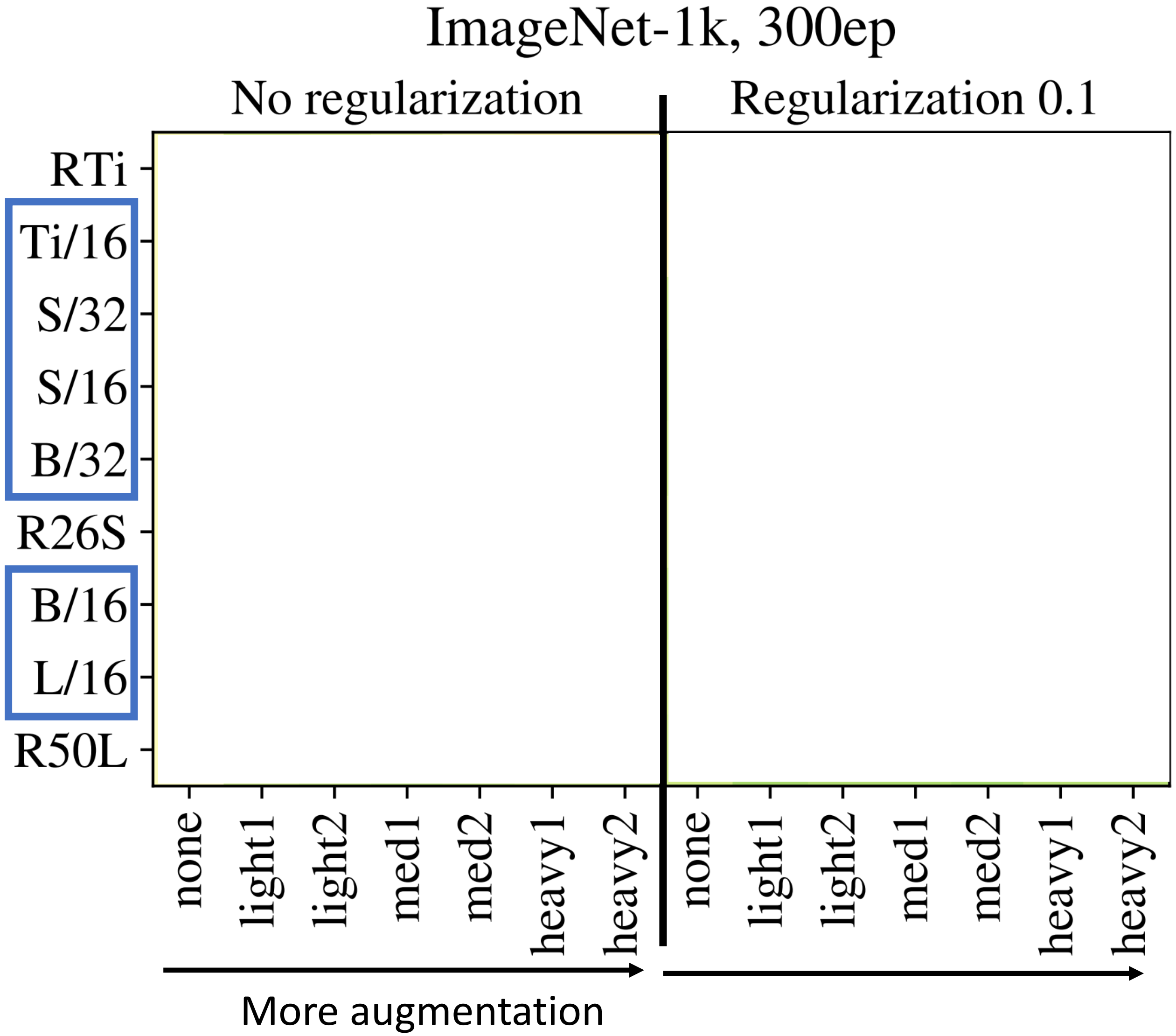
Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Augmentation and Regularization

Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment

ViT models:
Ti = Tiny
S = Small
B = Base
L = Large



Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

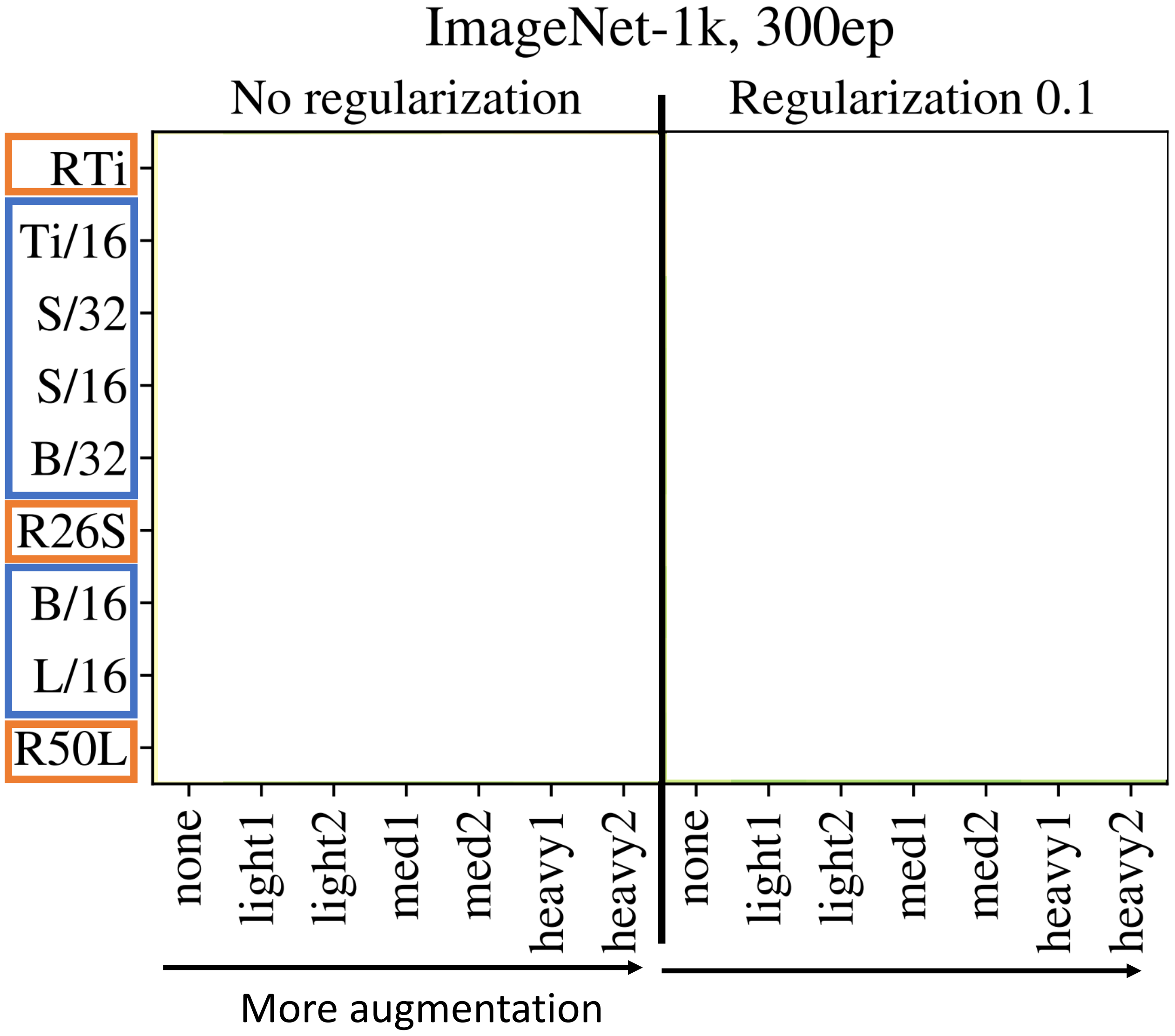
Improving ViT: Augmentation and Regularization

Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment

Hybrid models:
ResNet blocks,
then ViT blocks

ViT models:
Ti = Tiny
S = Small
B = Base
L = Large



Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Augmentation and Regularization

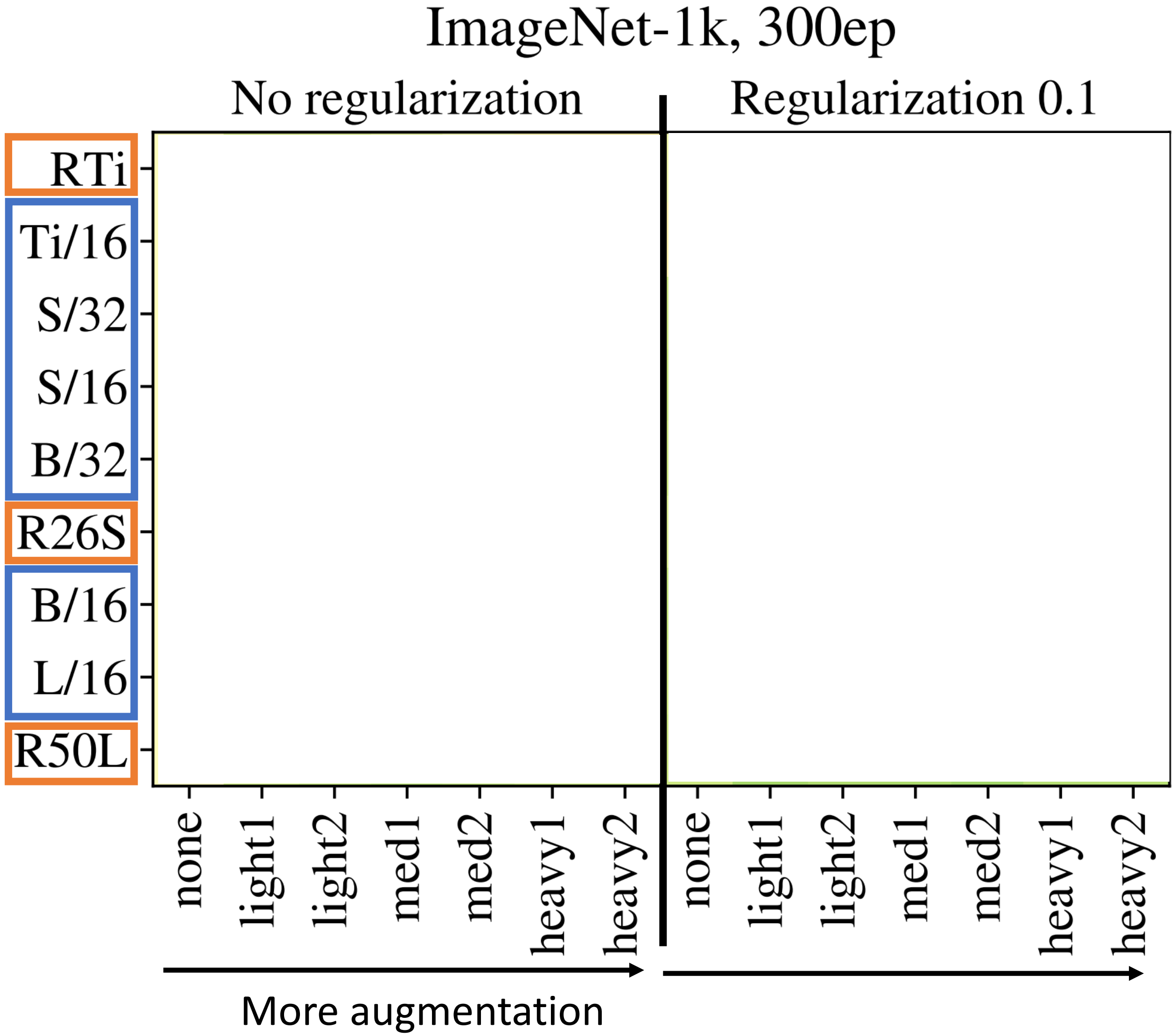
Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment

Hybrid models: ResNet blocks, then ViT blocks

ViT models:
Ti = Tiny
S = Small
B = Base
L = Large

Original Paper:
77.9
76.53



Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Augmentation and Regularization

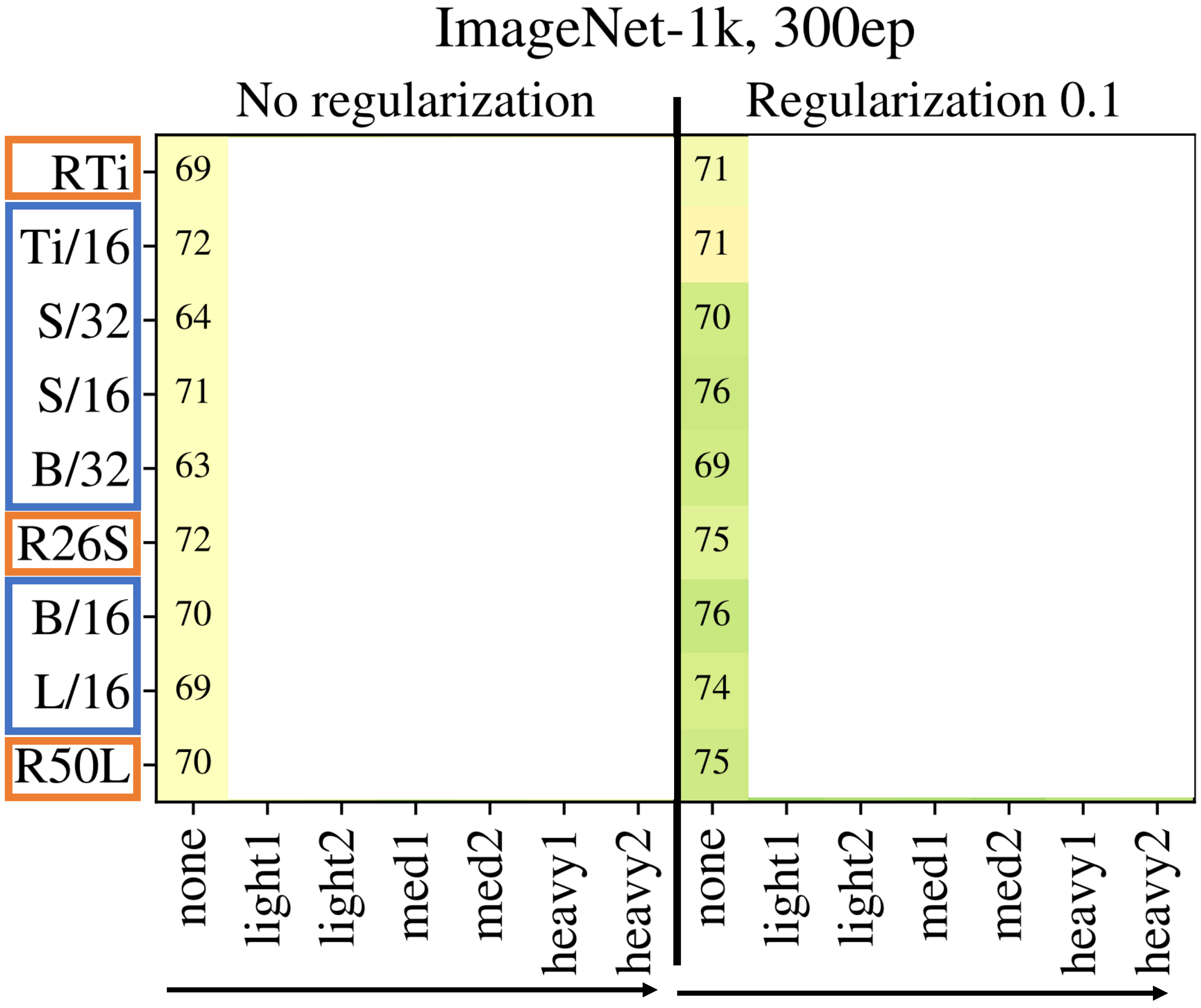
Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment

Hybrid models: ResNet blocks, then ViT blocks

ViT models:
Ti = Tiny
S = Small
B = Base
L = Large

Original Paper:
77.9
76.53



Adding regularization is (almost) always helpful

Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Augmentation and Regularization

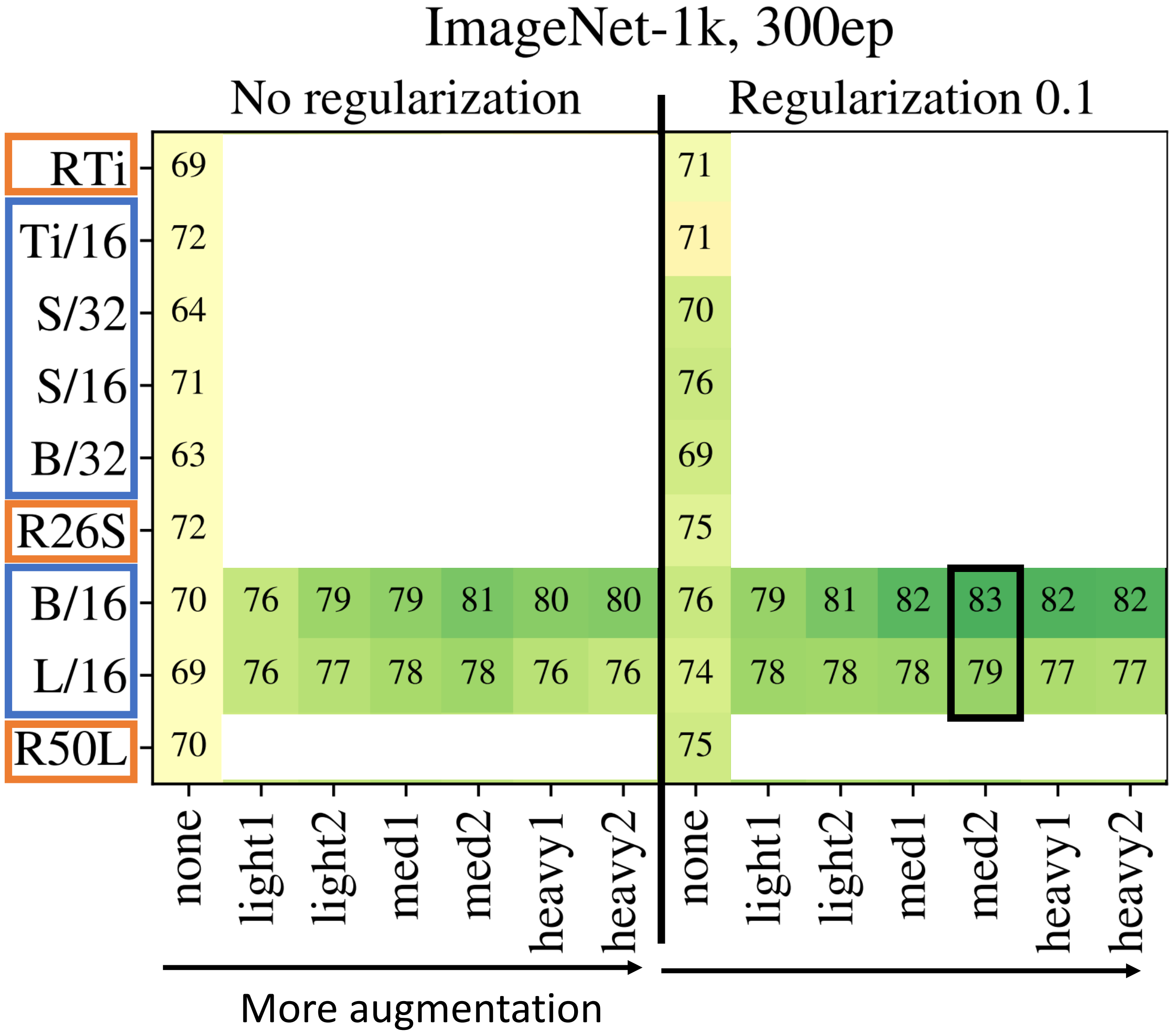
Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment

Hybrid models: ResNet blocks, then ViT blocks

ViT models:
Ti = Tiny
S = Small
B = Base
L = Large

Original Paper:
77.9
76.53



Regularization + Augmentation gives big improvements over original results

Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Augmentation and Regularization

Regularization for ViT models:
- Weight Decay
- Stochastic Depth
- Dropout (in FFN layers of Transformer)

Data Augmentation for ViT models:
- MixUp
- RandAugment

Hybrid models: ResNet blocks, then ViT blocks

ViT models:
Ti = Tiny
S = Small
B = Base
L = Large

Original Paper:
77.9
76.53

Lots of other patterns in full results

ImageNet-1k, 300ep

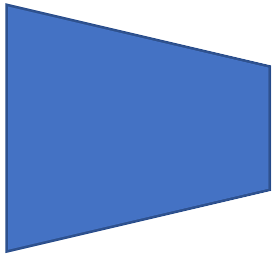
	No regularization							Regularization 0.1						
	none	light1	light2	med1	med2	heavy1	heavy2	none	light1	light2	med1	med2	heavy1	heavy2
RTi	69	73	73	72	70	69	68	71	70	67	65	63	62	61
Ti/16	72	76	75	75	74	72	71	71	72	68	65	63	63	62
S/32	64	71	76	76	76	74	74	70	72	72	71	71	69	68
S/16	71	77	79	81	82	80	80	76	79	80	79	79	77	77
B/32	63	70	73	75	76	75	76	69	74	77	77	78	77	77
R26S	72	76	78	79	80	80	80	75	78	81	82	82	81	81
B/16	70	76	79	79	81	80	80	76	79	81	82	83	82	82
L/16	69	76	77	78	78	76	76	74	78	78	78	79	77	77
R50L	70	75	76	77	77	76	76	75	78	78	78	79	77	77

More augmentation

Steiner et al, "How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers", arXiv 2021

Improving ViT: Distillation

Step 1: Train a **teacher model** on images and ground-truth labels



P(cat) = 0.9
P(dog) = 0.1

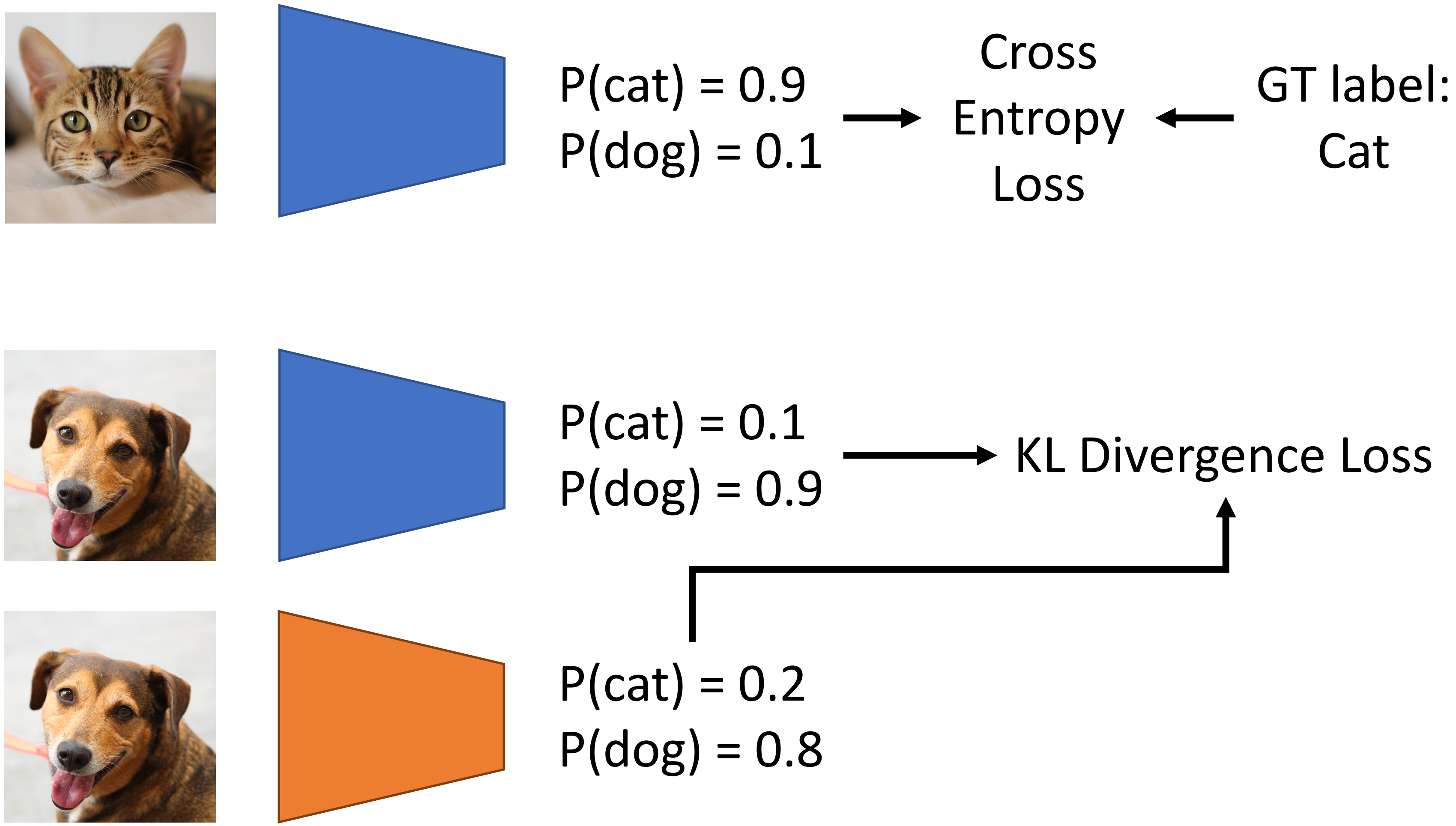
→ Cross Entropy Loss ← GT label: Cat

Hinton et al, "Distilling the knowledge in a neural network", NeurIPS Deep Learning and Representation Learning Workshop, 2015

Improving ViT: Distillation

Step 1: Train a **teacher model** on images and ground-truth labels

Step 2: Train a **student model** to match predictions from the **teacher**

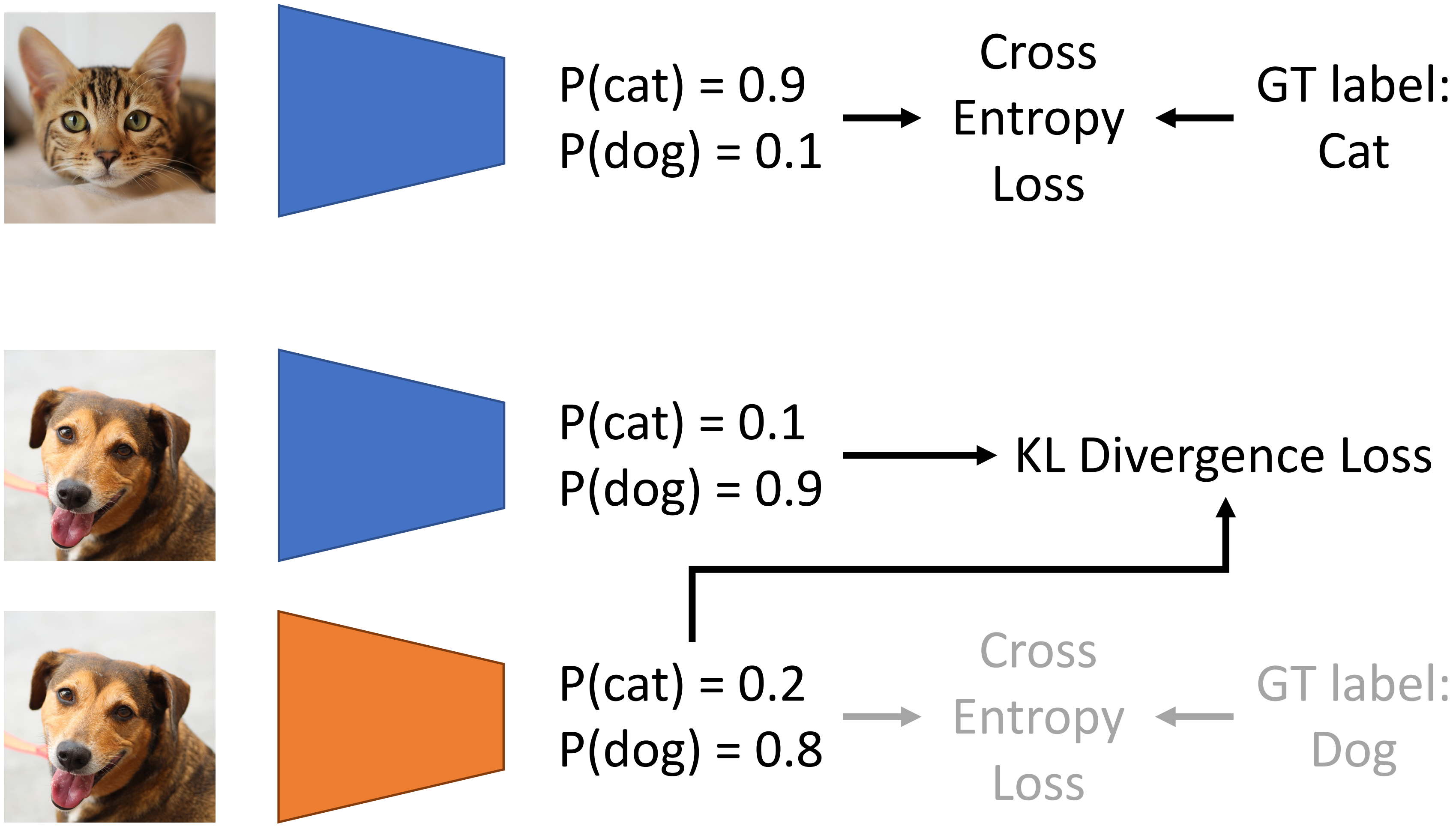


Hinton et al, "Distilling the knowledge in a neural network", NeurIPS Deep Learning and Representation Learning Workshop, 2015

Improving ViT: Distillation

Step 1: Train a **teacher model** on images and ground-truth labels

Step 2: Train a **student model** to match predictions from the **teacher** (sometimes also to match GT labels)



Hinton et al, "Distilling the knowledge in a neural network", NeurIPS Deep Learning and Representation Learning Workshop, 2015

Improving ViT: Distillation

Often works better than training student from scratch (especially if teacher is bigger than student)

Step 1: Train a **teacher model** on images and ground-truth labels

P(cat) = 0.9
P(dog) = 0.1

Cross Entropy Loss

GT label: Cat

Step 2: Train a **student model** to match predictions from the **teacher** (sometimes also to match GT labels)

P(cat) = 0.1
P(dog) = 0.9

KL Divergence Loss

P(cat) = 0.2
P(dog) = 0.8

Cross Entropy Loss

GT label: Dog

Hinton et al, "Distilling the knowledge in a neural network", NeurIPS Deep Learning and Representation Learning Workshop, 2015

Improving ViT: Distillation

Can also train student on **unlabeled** data! (Semi-supervised learning)

Step 1: Train a **teacher model** on images and ground-truth labels

P(cat) = 0.9
P(dog) = 0.1

Cross Entropy Loss

GT label: Cat

Step 2: Train a **student model** to match predictions from the **teacher** (sometimes also to match GT labels)

P(cat) = 0.1
P(dog) = 0.9

KL Divergence Loss

P(cat) = 0.2
P(dog) = 0.8

Cross Entropy Loss

GT label: Dog

Hinton et al, "Distilling the knowledge in a neural network", NeurIPS Deep Learning and Representation Learning Workshop, 2015

Improving ViT: Distillation

Step 1: Train a **teacher CNN** on ImageNet

P(cat) = 0.9
P(dog) = 0.1 → Cross Entropy Loss ← GT label: Cat

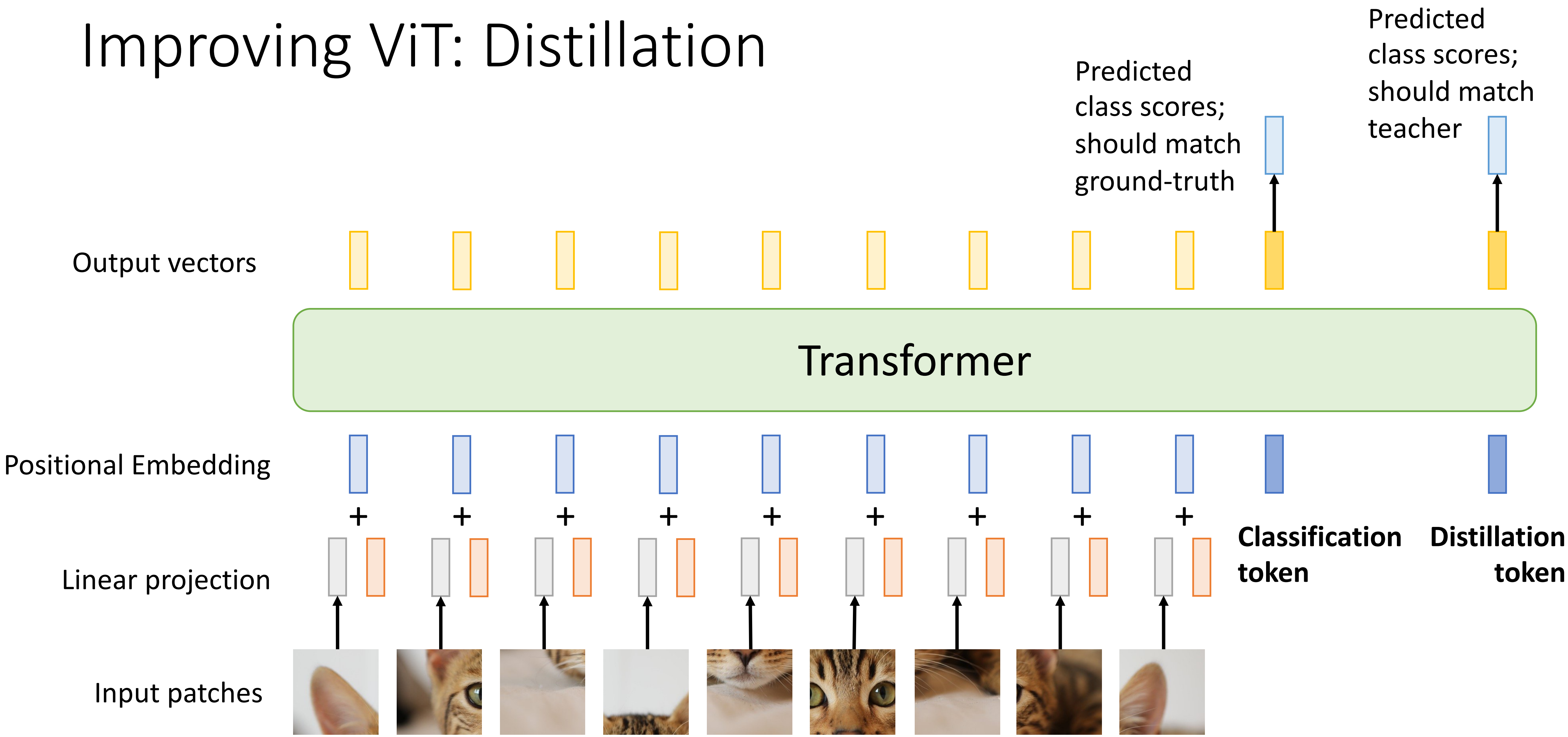
Step 2: Train a **student ViT** to match ImageNet predictions from the **teacher CNN**
(and match GT labels)

P(cat) = 0.1
P(dog) = 0.9 → KL Divergence Loss

P(cat) = 0.2
P(dog) = 0.8 → Cross Entropy Loss ← GT label: Dog

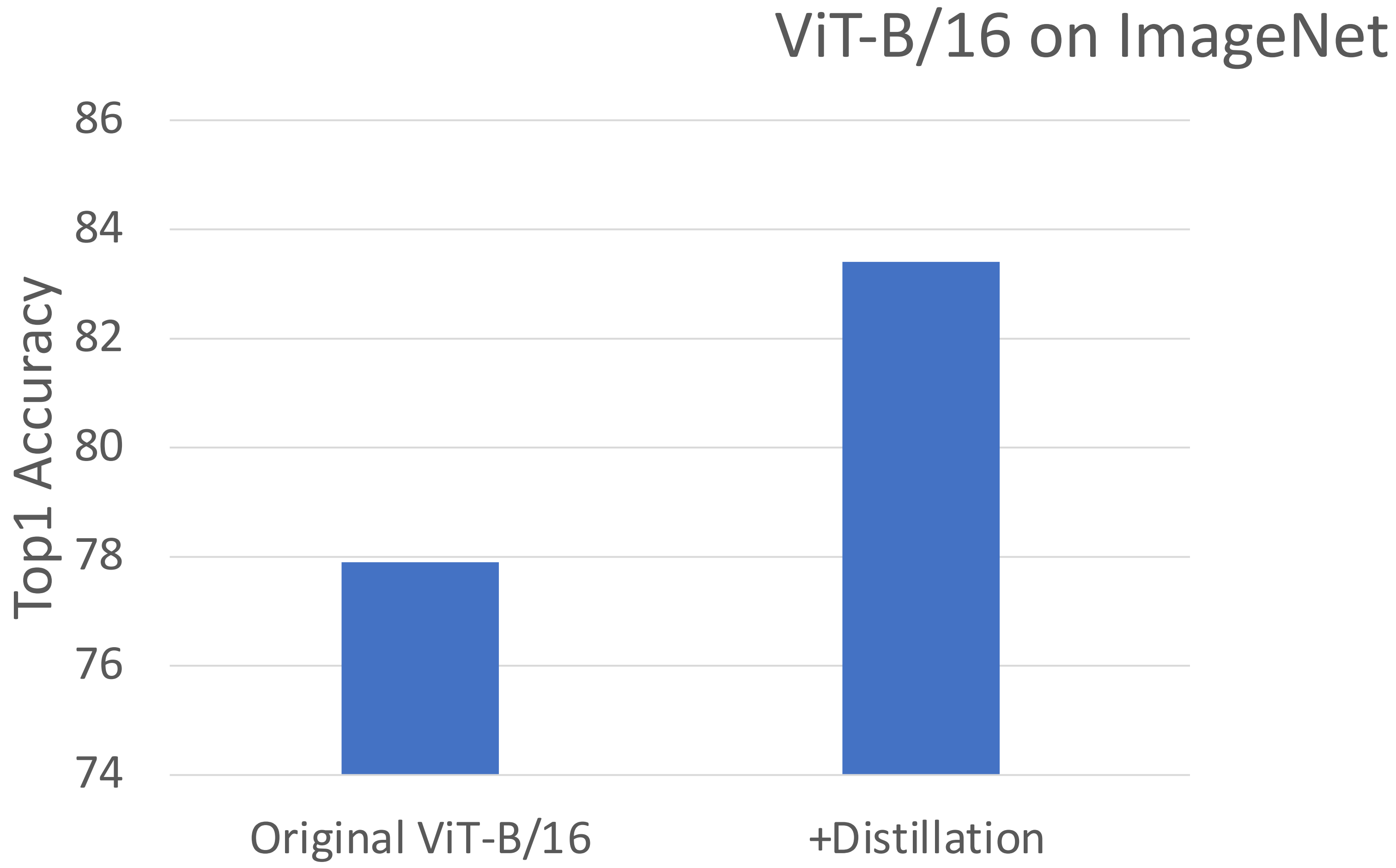
Touvrom et al, "Training data-efficient image transformers & distillation through attention", ICML 2021

Improving ViT: Distillation



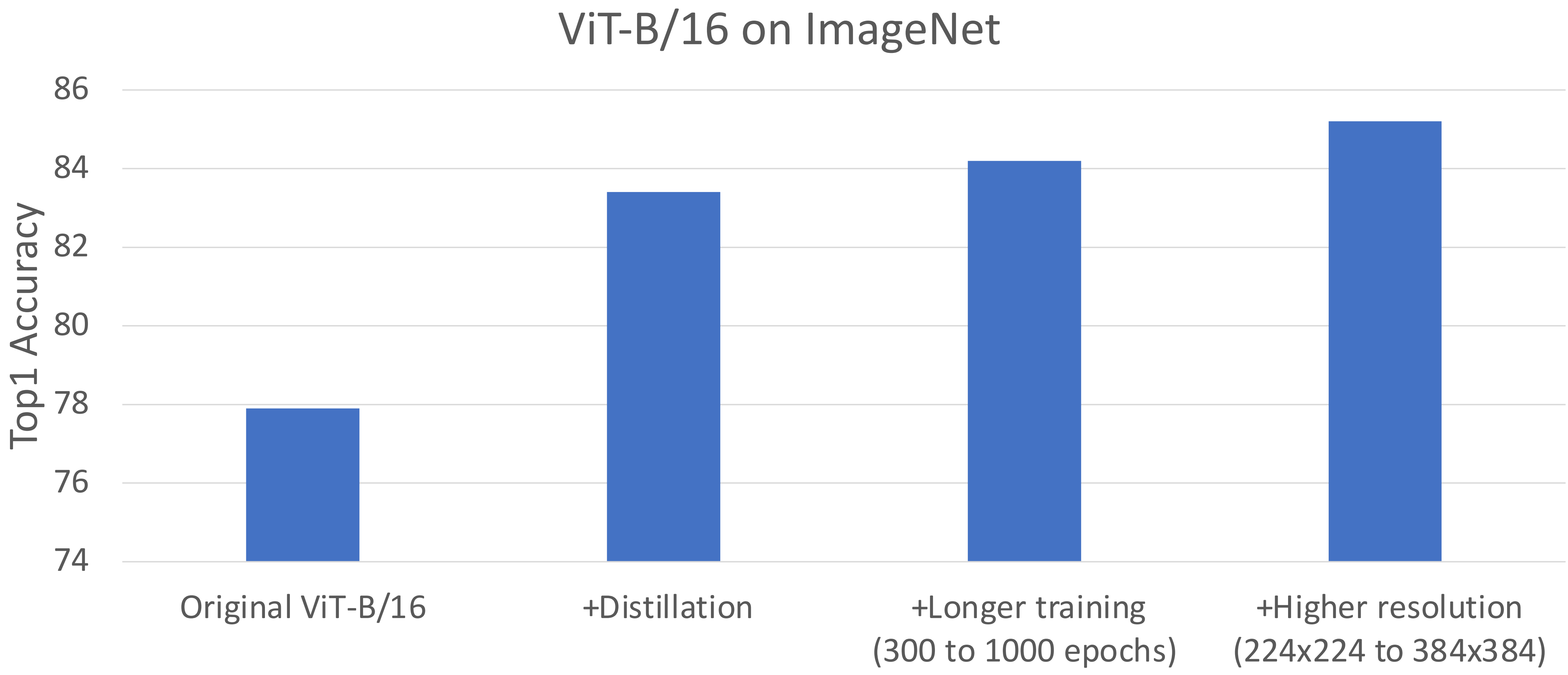
Touvrom et al, “Training data-efficient image transformers & distillation through attention”, ICML 2021

Improving ViT: Distillation



Touvrom et al, “Training data-efficient image transformers & distillation through attention”, ICML 2021

Improving ViT: Distillation



Touvrom et al, "Training data-efficient image transformers & distillation through attention", ICML 2021

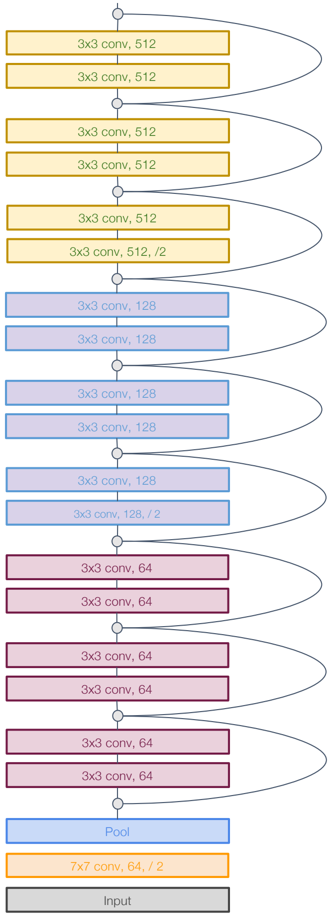
ViT vs CNN

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224



In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

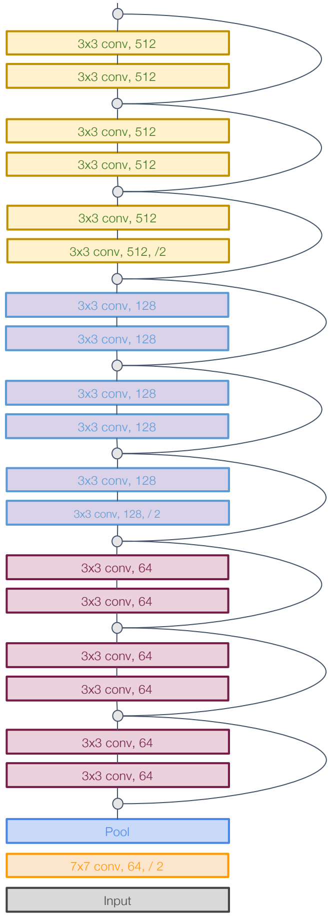
ViT vs CNN

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

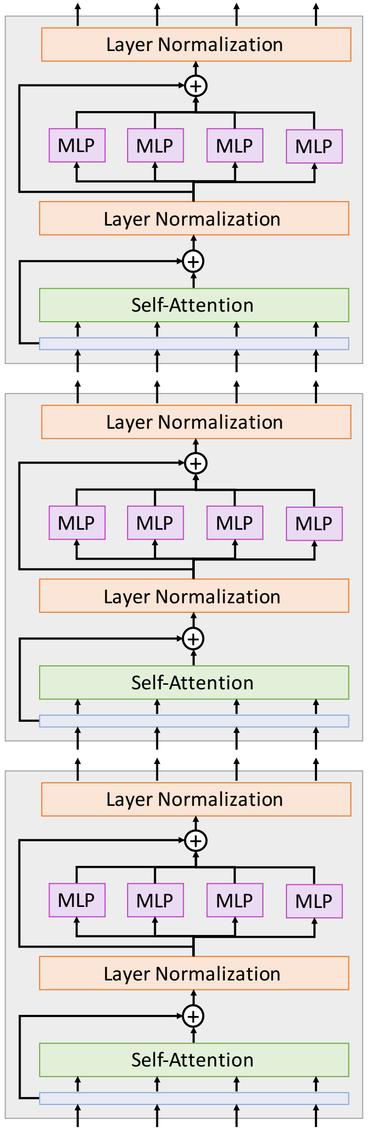
Input:
3 x 224 x 224



In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)



3rd block:
768 x 14 x 14

2nd block:
768 x 14 x 14

1st block:
768 x 14 x 14

Input:
3 x 224 x 224

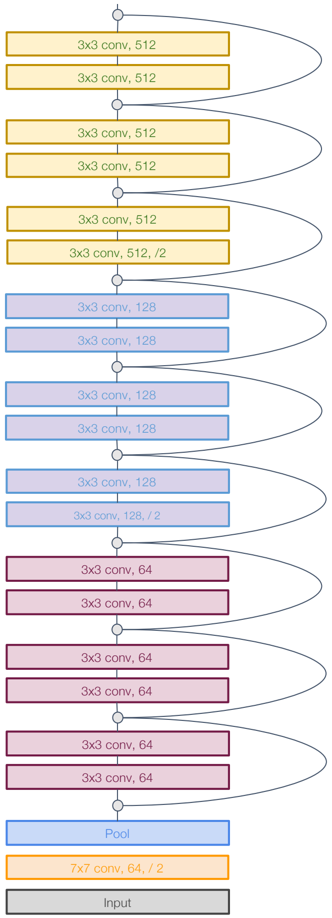
ViT vs CNN

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224

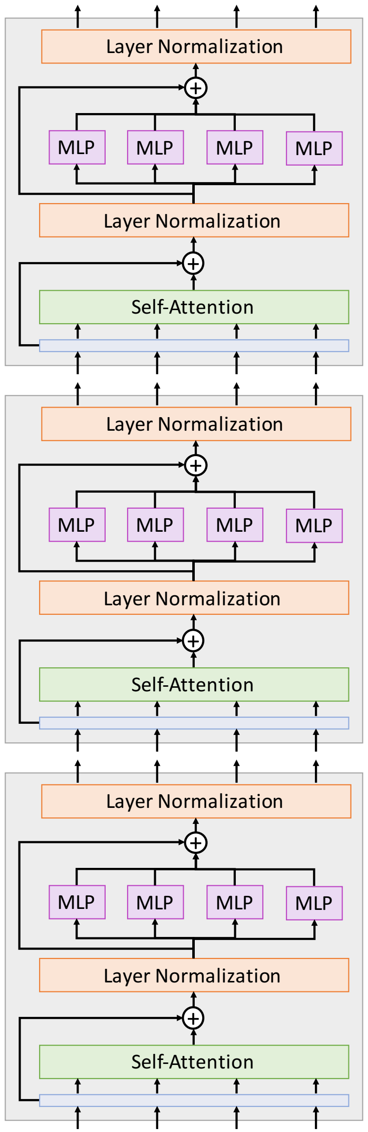


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)

Can we build a **hierarchical** ViT model?



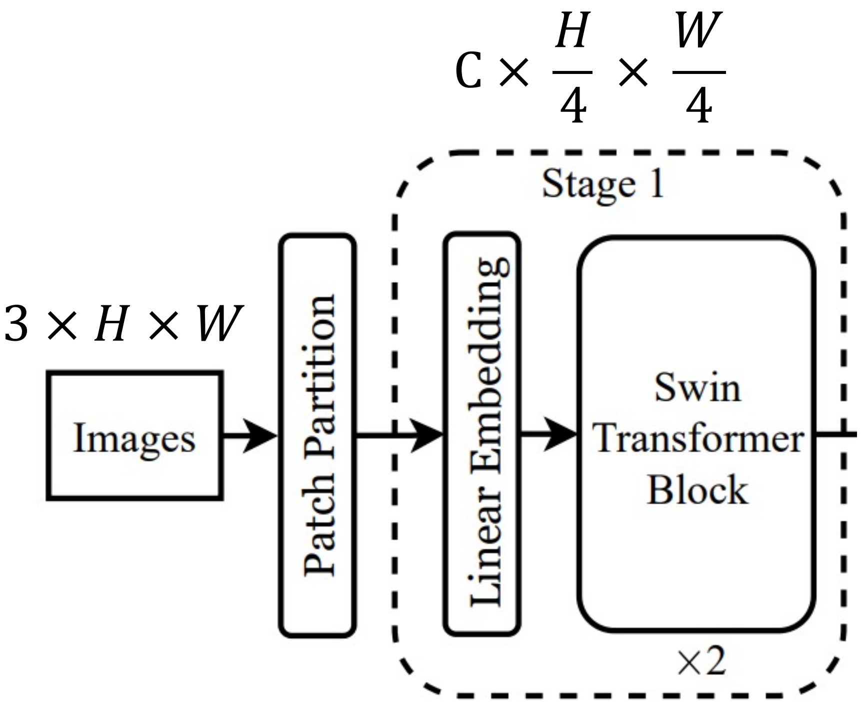
3rd block:
768 x 14 x 14

2nd block:
768 x 14 x 14

1st block:
768 x 14 x 14

Input:
3 x 224 x 224

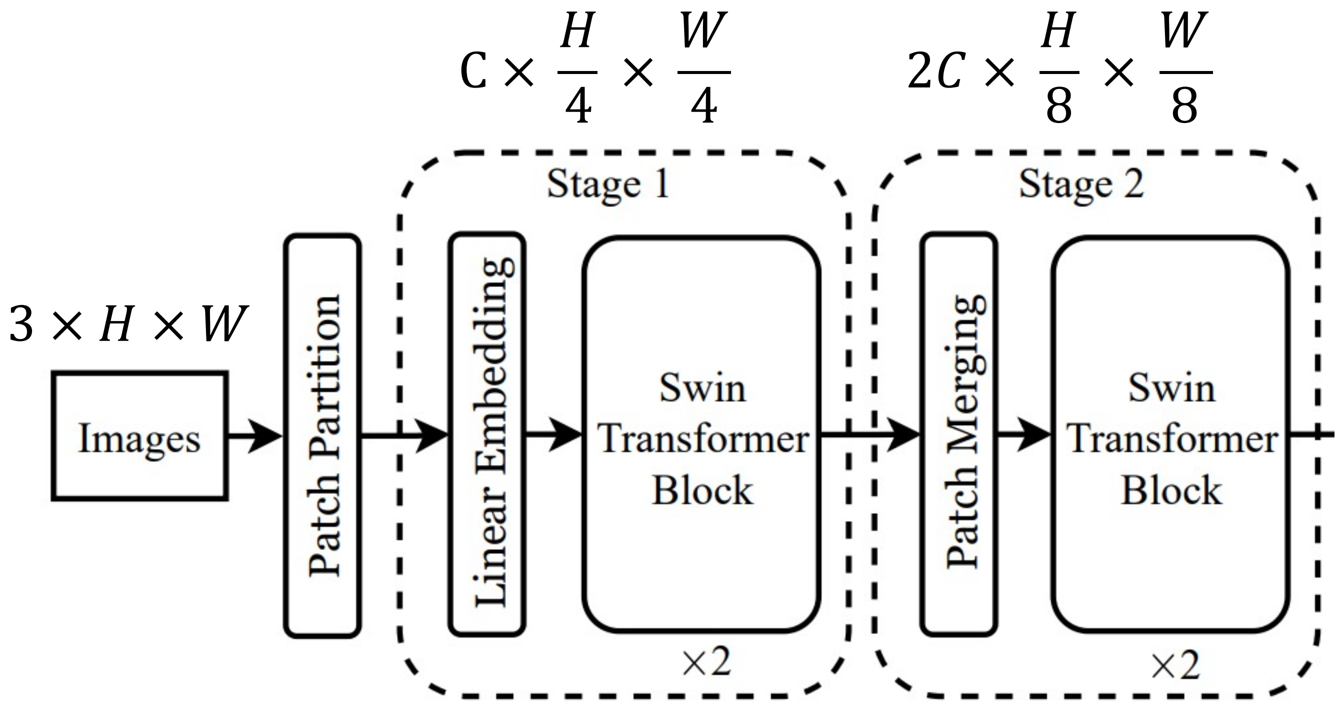
Hierarchical ViT: Swin Transformer



Divide image into 4x4
patches and project
to C dimensions

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

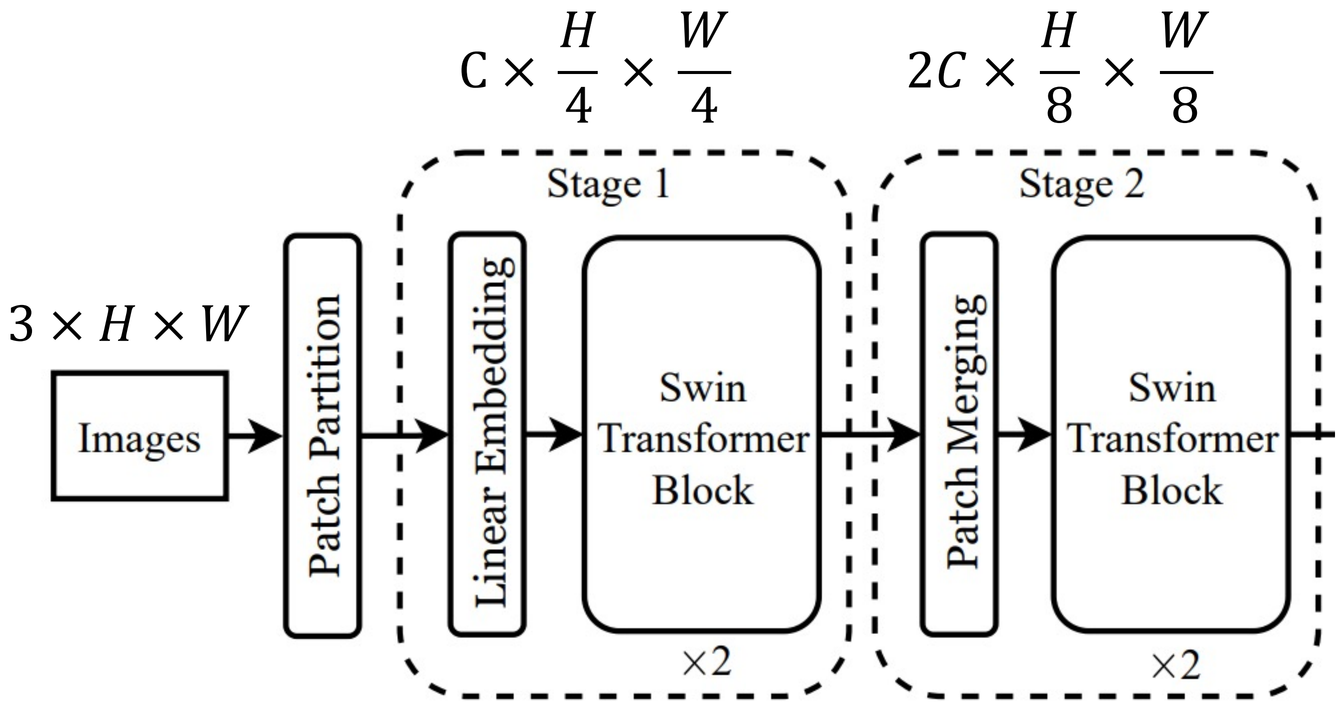


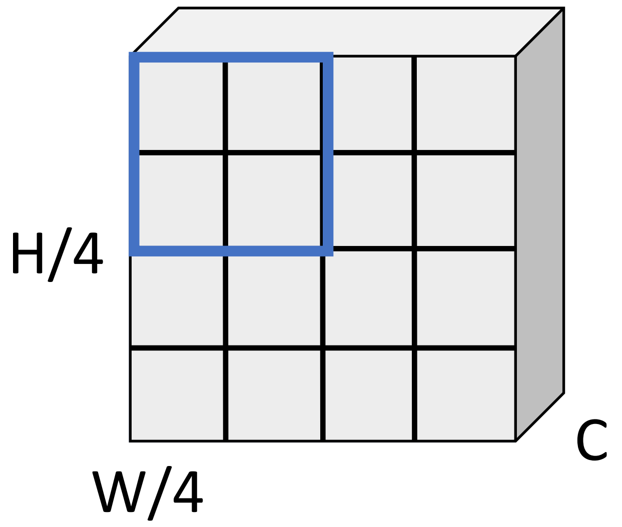
Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



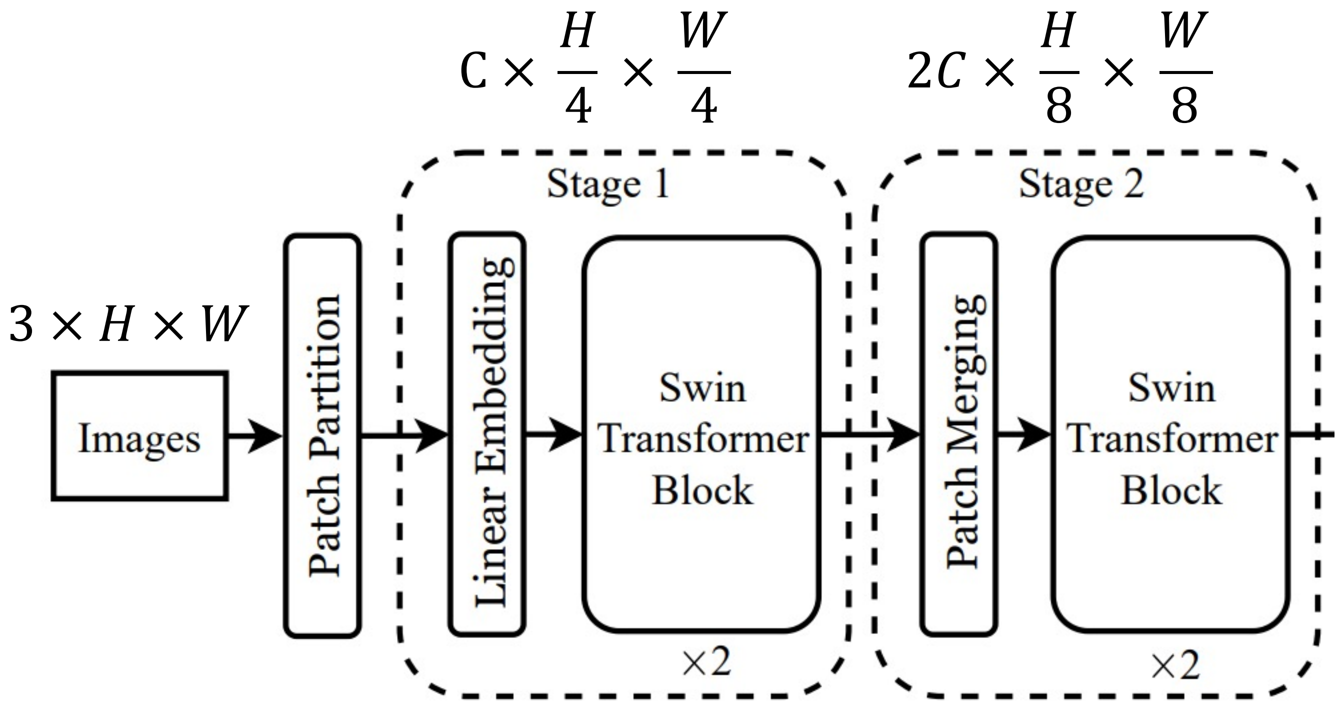


Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

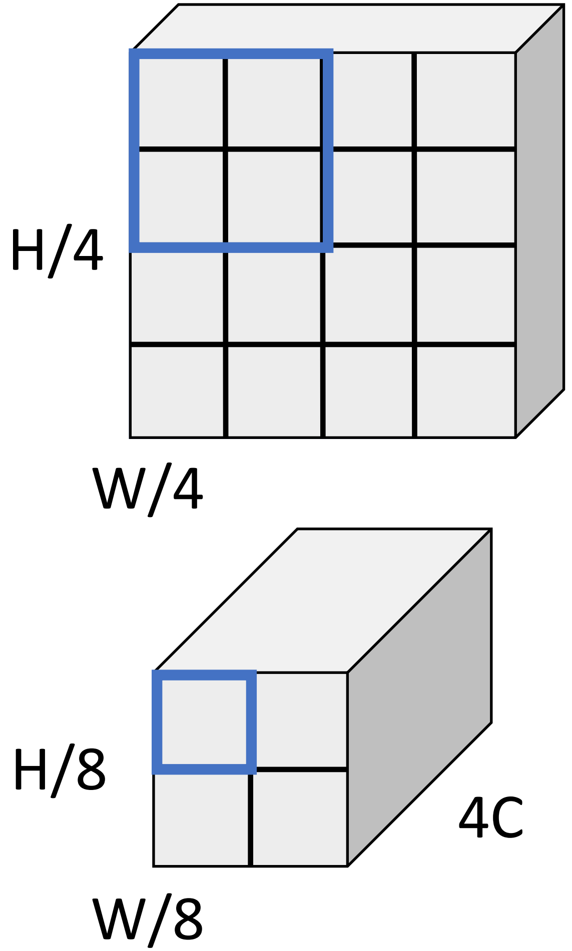
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

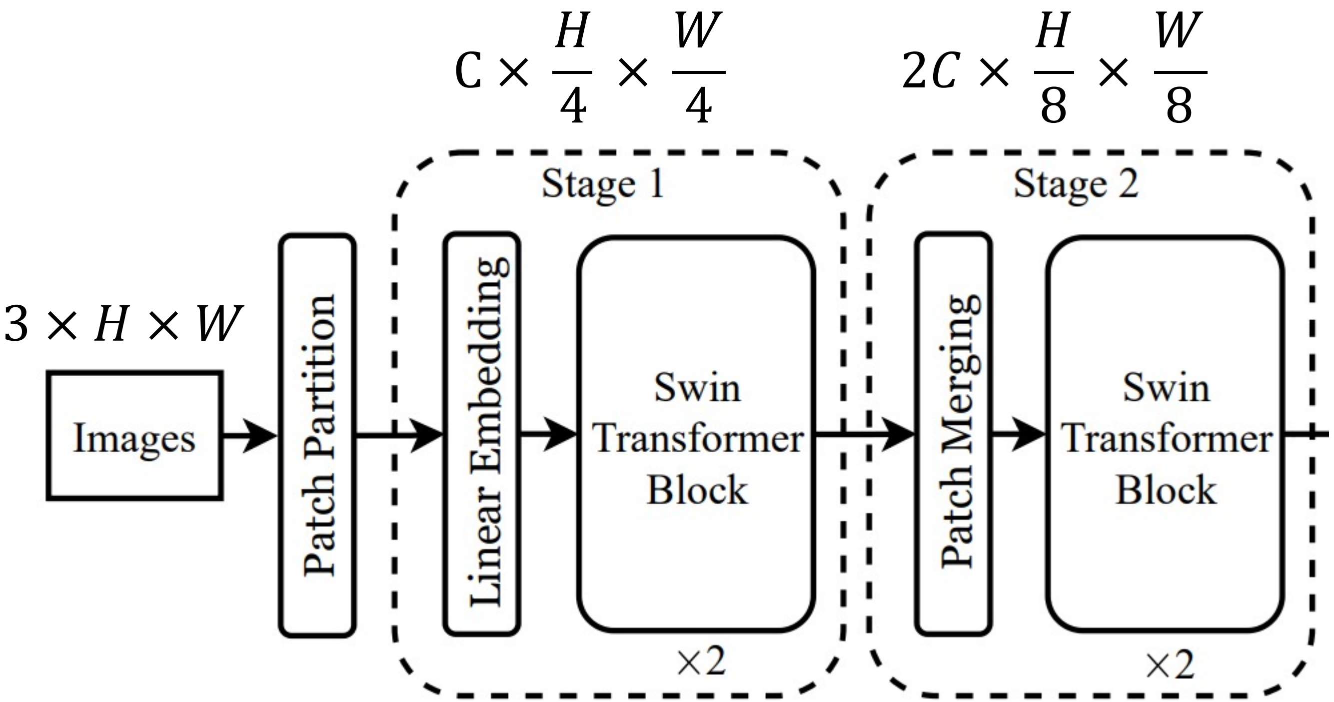
Merge 2x2 neighborhoods; now patches are (effectively) 8x8



Concatenate groups of 2x2 features

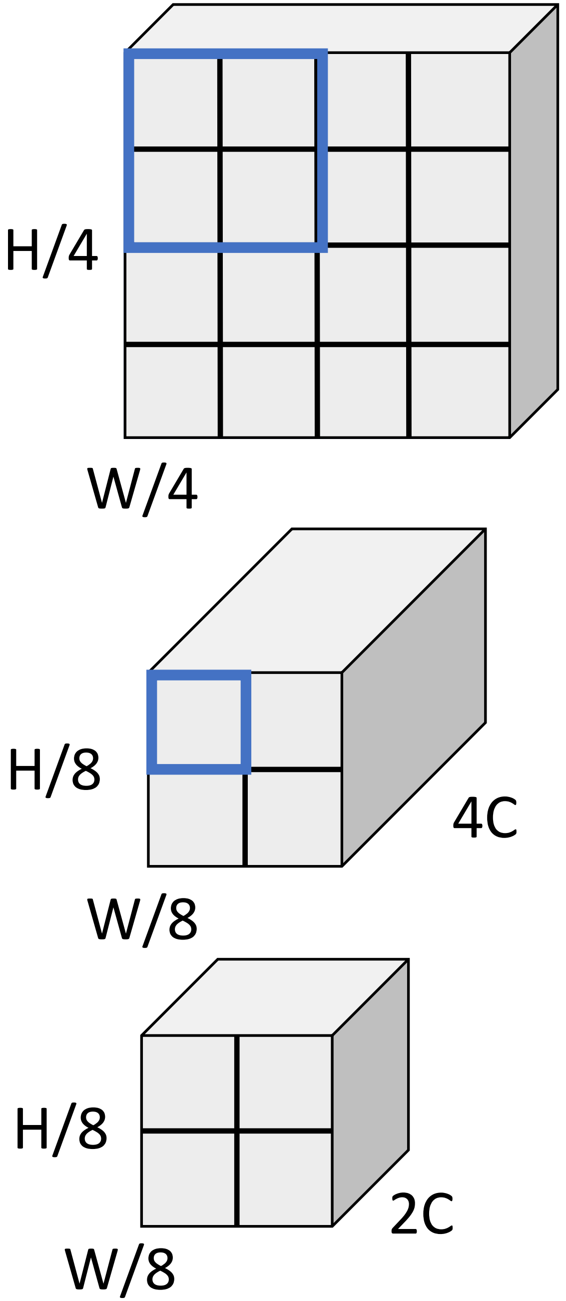
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

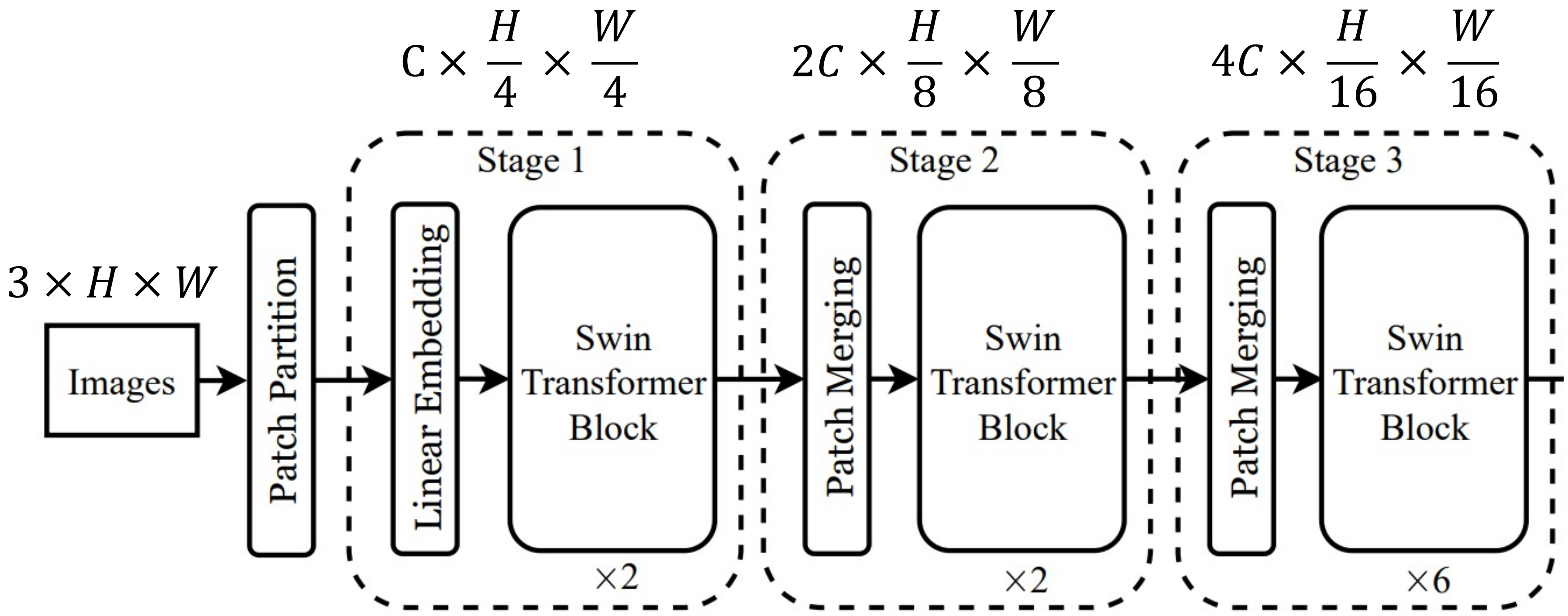


Concatenate groups of 2x2 features

Linear projection from 4C to 2C channels (1x1 conv)

Liu et al, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”, CVPR 2021

Hierarchical ViT: Swin Transformer



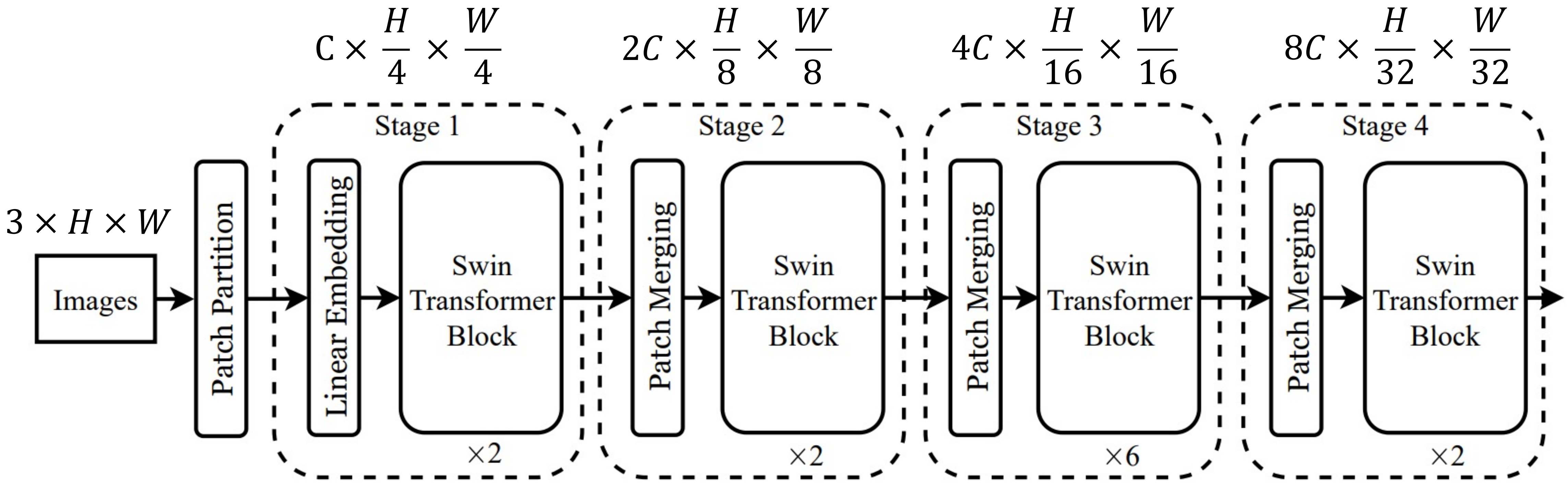
Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

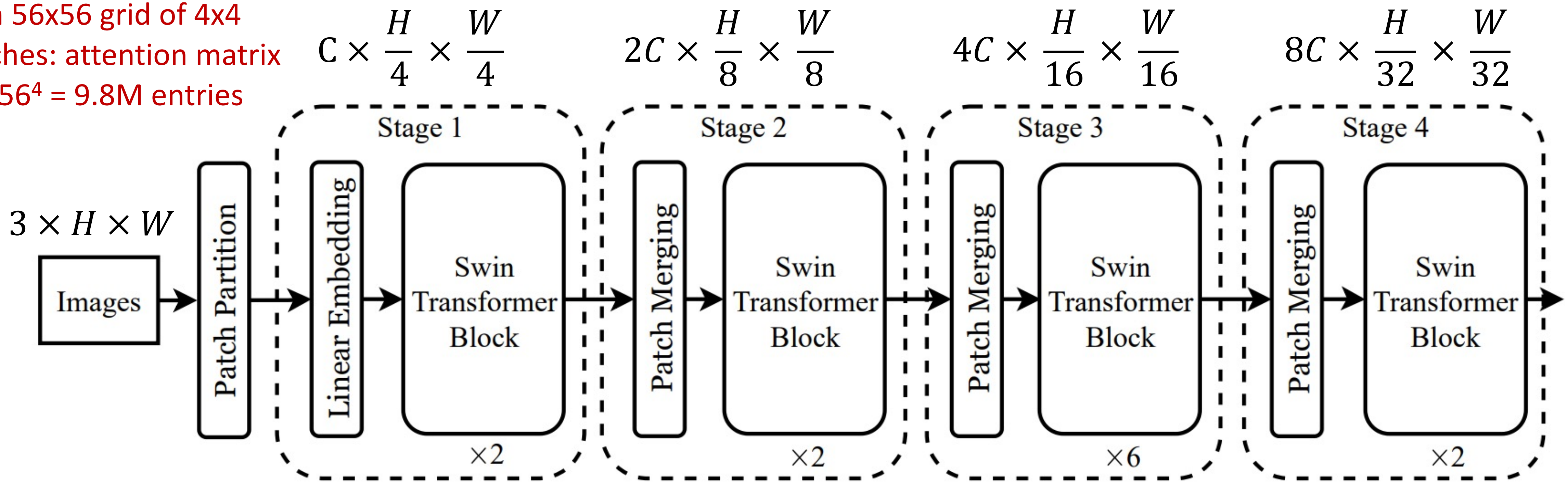
Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

Problem: 224x224 image with 56x56 grid of 4x4 patches: attention matrix has 56^4 = 9.8M entries



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

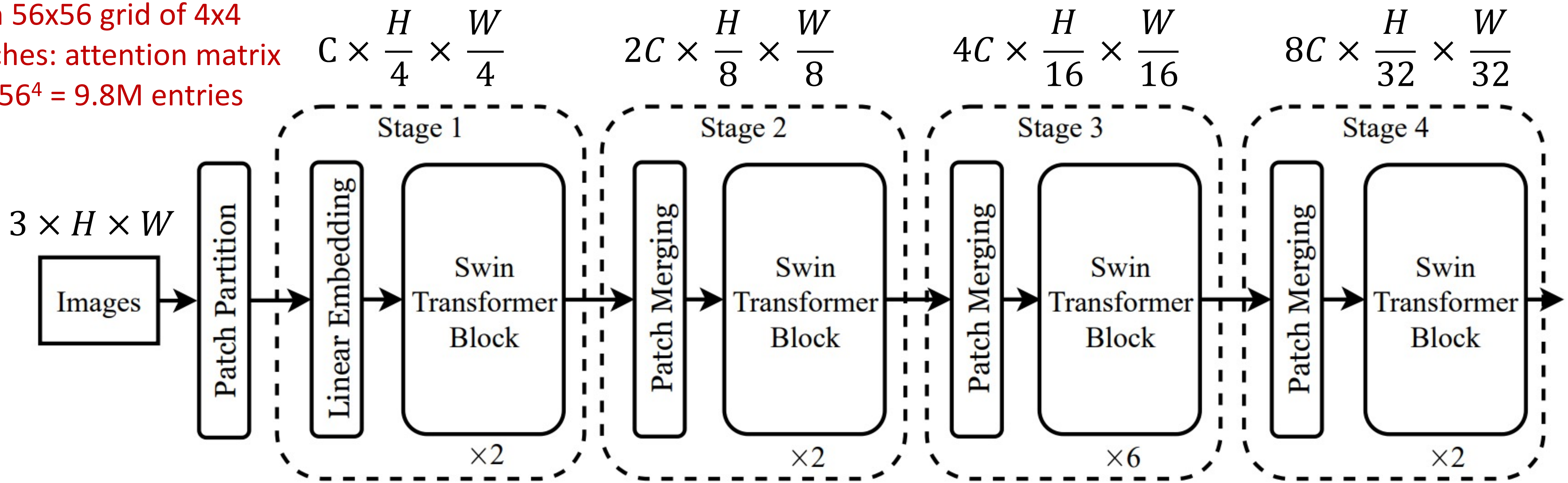
Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

Problem: 224x224 image with 56x56 grid of 4x4 patches: attention matrix has 56^4 = 9.8M entries



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Solution: don't use full attention, instead use attention over patches

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

With H x W grid of **tokens**, each attention matrix is H^2W^2 – **quadratic** in image size

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

With H x W grid of **tokens**, each attention matrix is H^2W^2 – **quadratic** in image size

Rather than allowing each **token** to attend to all other tokens, instead divide into **windows** of M x M tokens (here M=4); only compute attention within each window

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

With H x W grid of **tokens**, each attention matrix is H^2W^2 – **quadratic** in image size

Rather than allowing each **token** to attend to all other tokens, instead divide into **windows** of M x M tokens (here M=4); only compute attention within each window

Total size of all attention matrices is now: $M^4(H/M)(W/M) = M^2HW$

Linear in image size for fixed M!
Swin uses M=7 throughout the network

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

Problem: tokens only interact with other tokens within the same window; no communication across windows

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Ugly detail: Non-square windows at edges and corners

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Block L: Normal windows

Block L+1: Shifted Windows

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

Standard Attention:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D}}\right)V$$

$Q, K, V: M^2 \times D$ (Query, Key, Value)

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

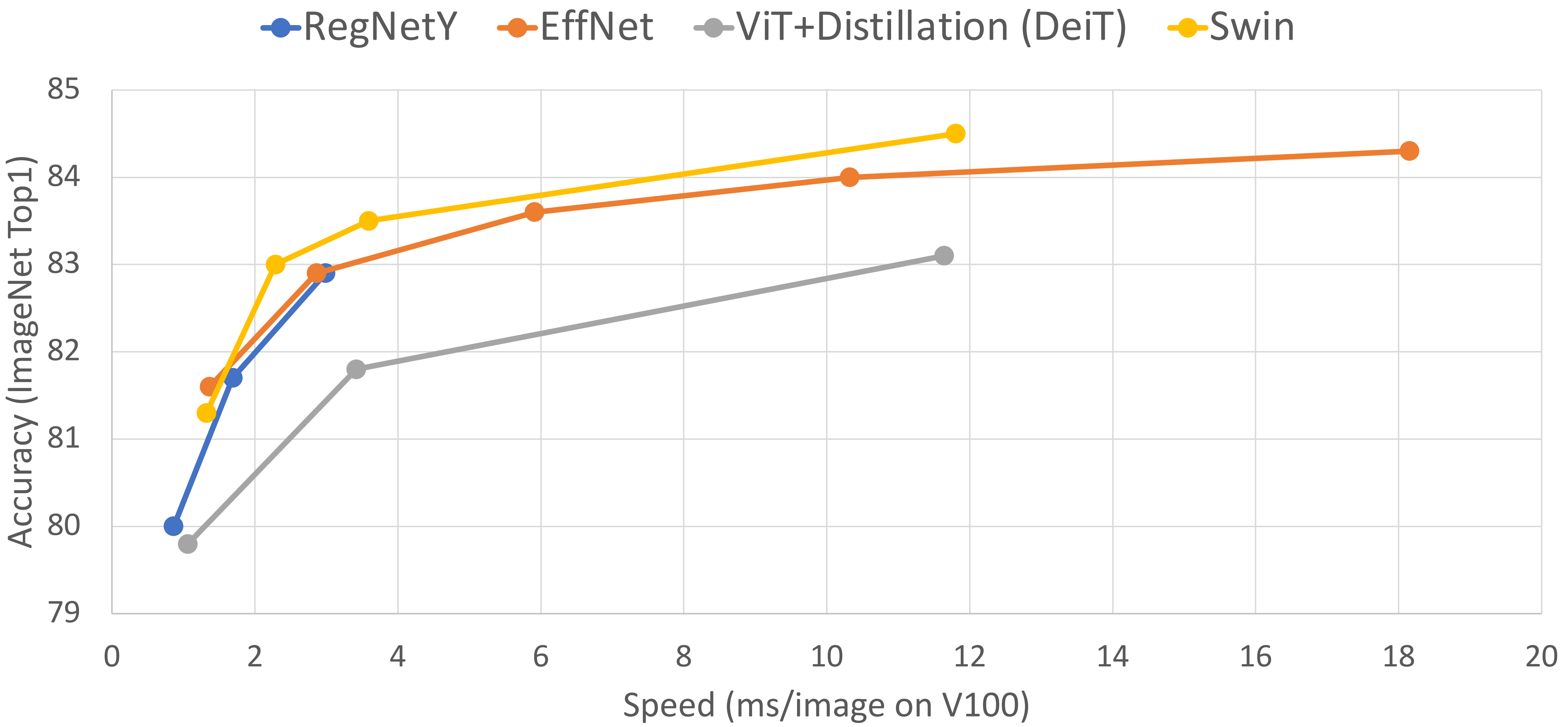
Attention with relative bias:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D}} + B\right)V$$

$Q, K, V: M^2 \times D$ (Query, Key, Value)
$B: M^2 \times M^2$ (learned biases)

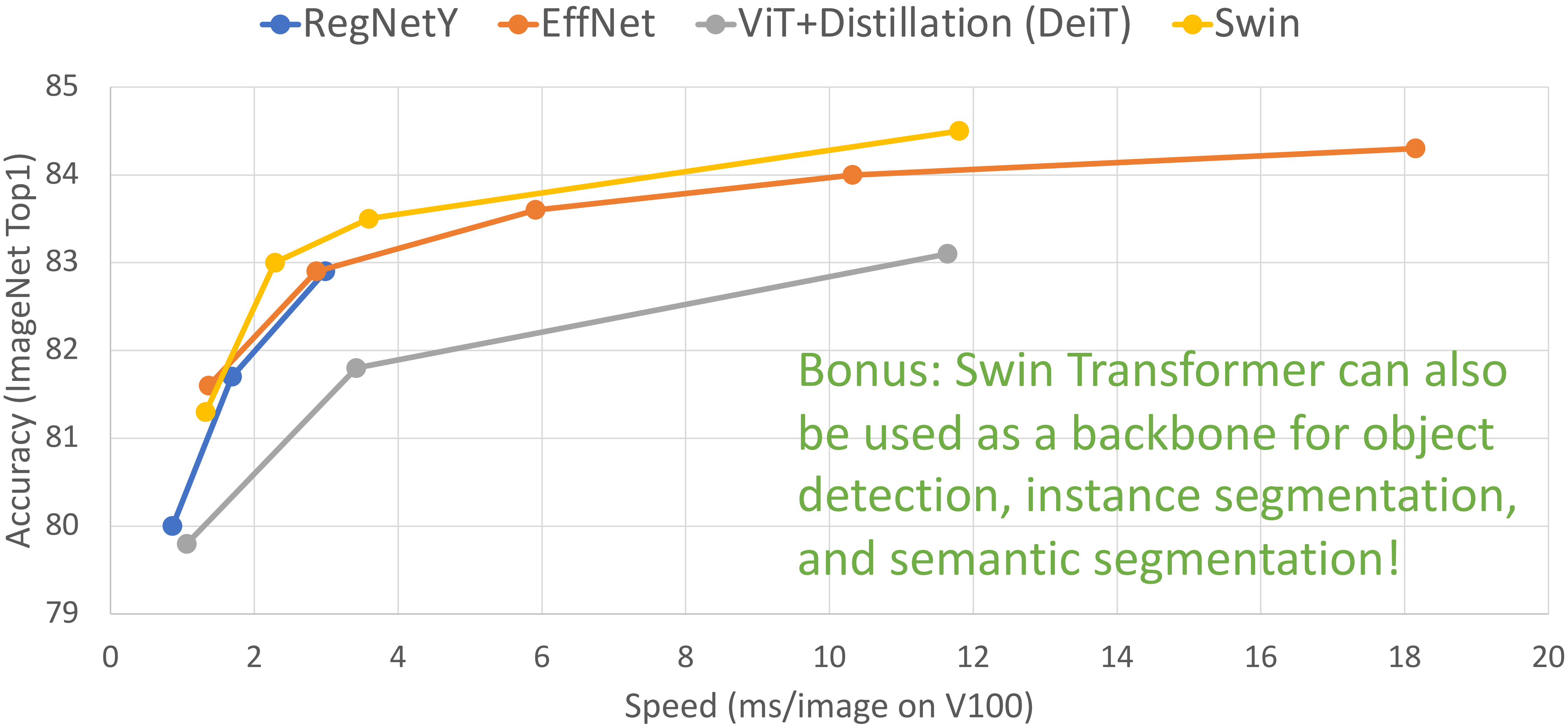
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Speed vs Accuracy



Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Speed vs Accuracy

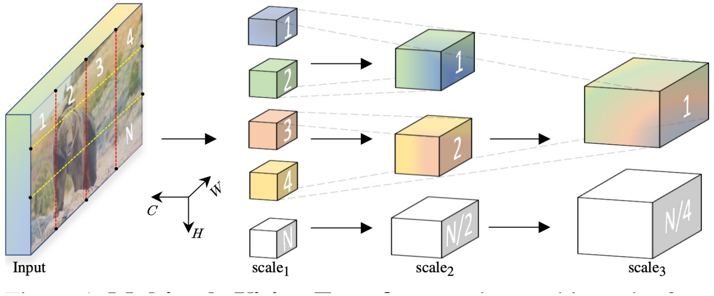


Bonus: Swin Transformer can also be used as a backbone for object detection, instance segmentation, and semantic segmentation!

Liu et al, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”, CVPR 2021

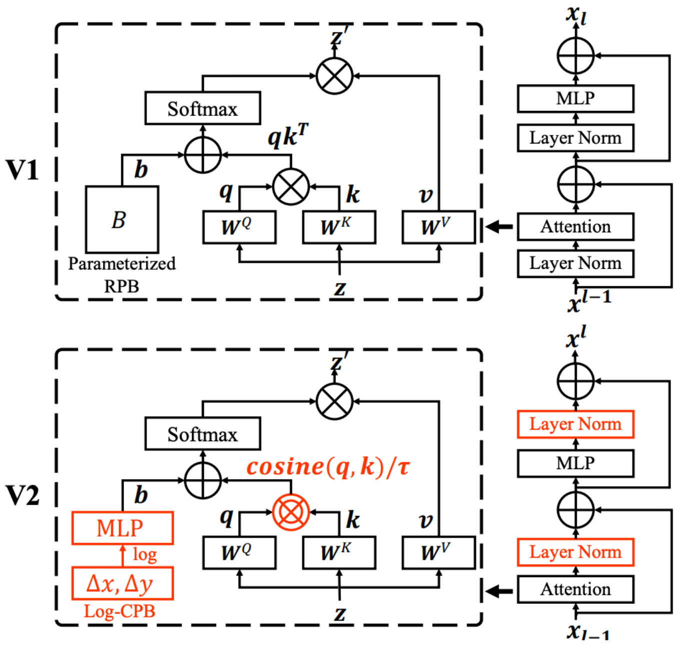
Other Hierarchical Vision Transformers

MViT



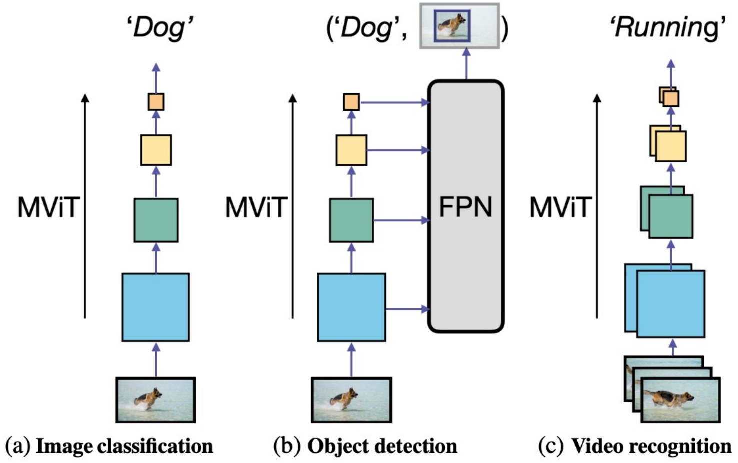
Fan et al, “Multiscale Vision Transformers”, ICCV 2021

Swin-V2



Liu et al, “Swin Transformer V2: Scaling up Capacity and Resolution”, CVPR 2022

Improved MViT

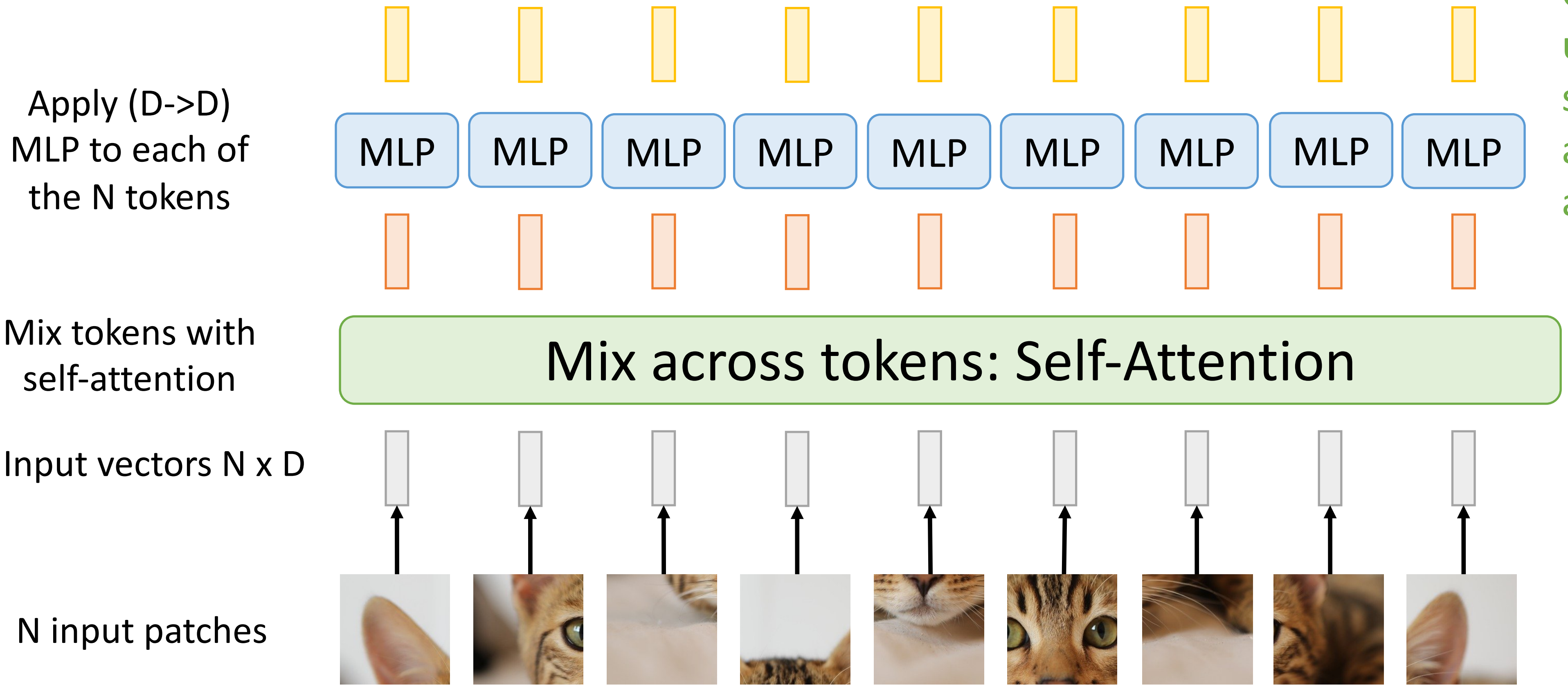


Li et al, “Improved Multiscale Vision Transformers for Classification and Detection”, arXiv 2021

Vision Transformer: Another Look

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

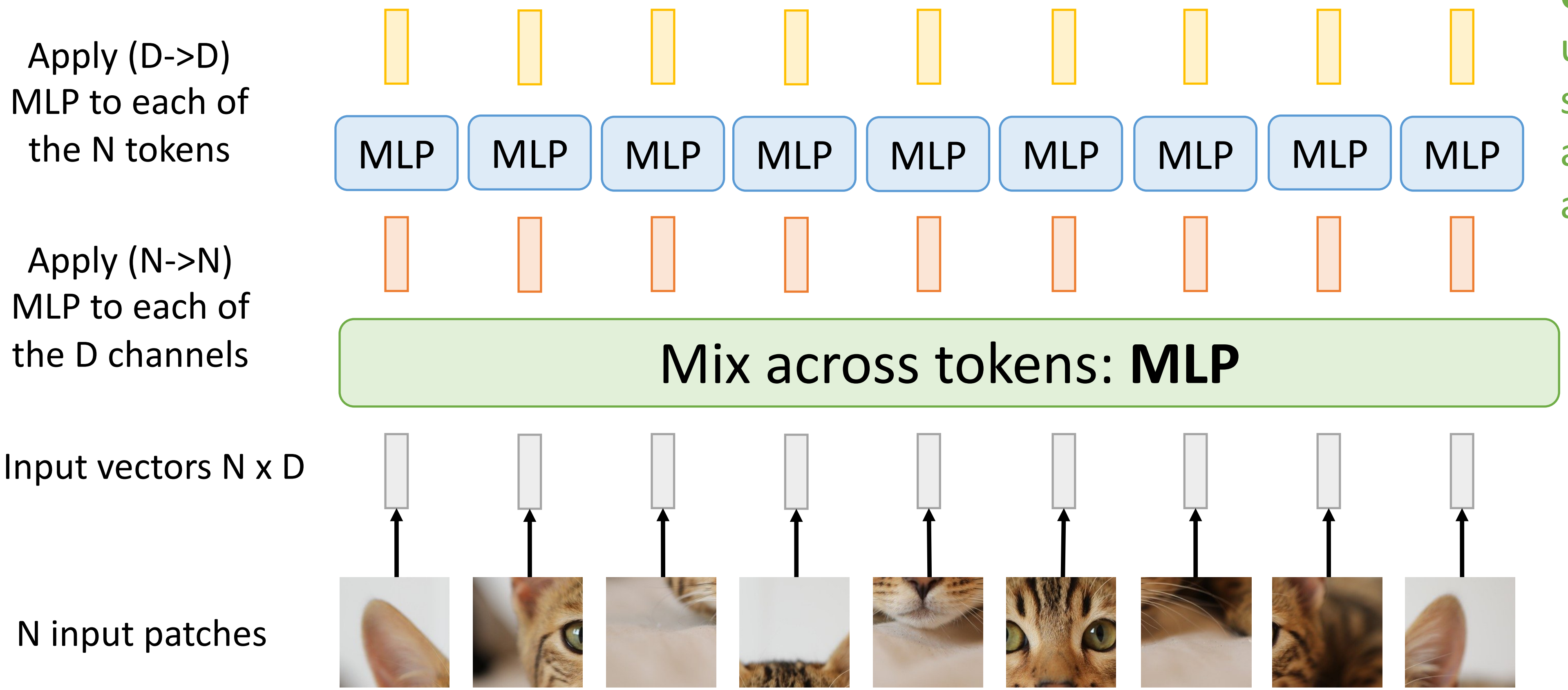
Vision Transformer: Another Look



Question: Can we use something simpler than self-attention to mix across tokens?

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

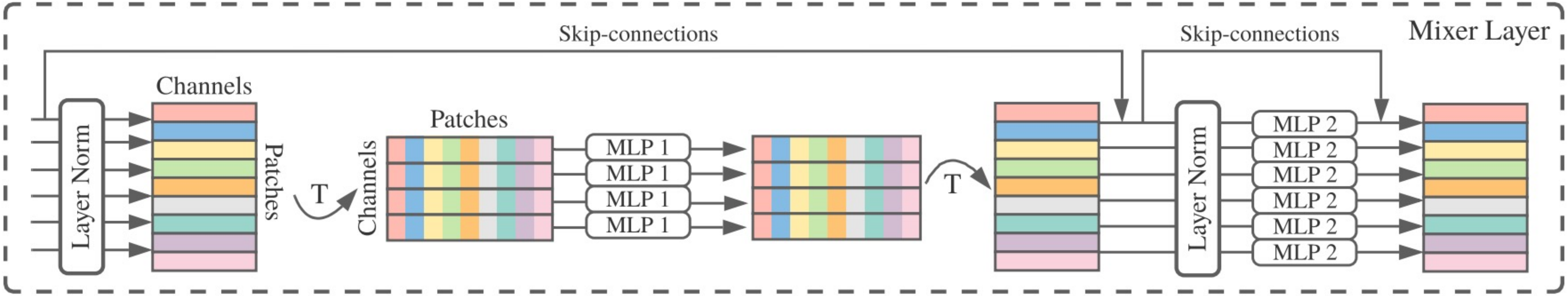
MLP-Mixer: An All-MLP Architecture



Question: Can we use something simpler than self-attention to mix across tokens?

Tolstikhin et al, "MLP-Mixer: An all-MLP architecture for vision", NeurIPS 2021

MLP-Mixer: An All-MLP Architecture



Input: N x C
N patches with
C channels each

MLP 1: C -> C,
apply to each of
the **N patches**

MLP 2: N -> N,
apply to each of
the **C channels**

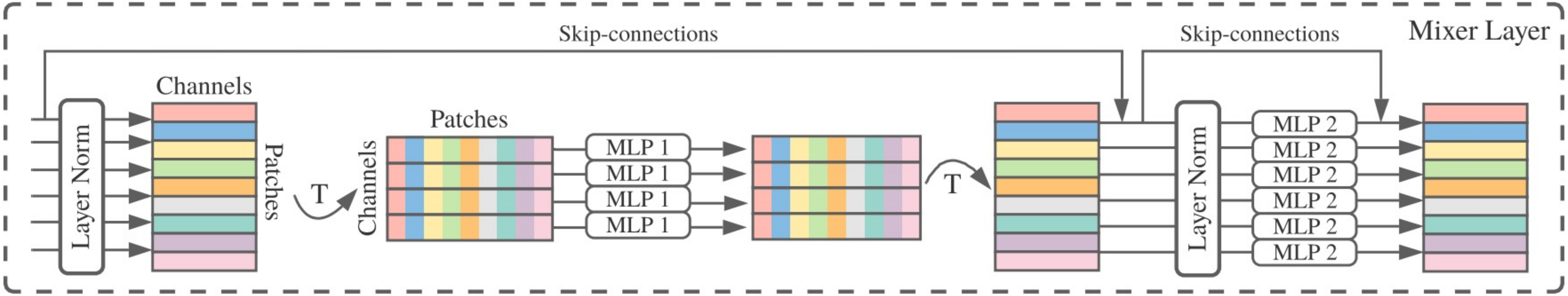
Tolstikhin et al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS 2021

MLP-Mixer: An All-MLP Architecture

MLP-Mixer is actually just a weird CNN???

Equivalent to
Conv(1x1, C->C, stride=1)

Equivalent to
Conv($N^{1/2}$ x $N^{1/2}$, C->C, groups=C)



Input: N x C
N patches with
C channels each

MLP 1: C -> C,
apply to each of
the **N patches**

MLP 2: N -> N,
apply to each of
the **C channels**

Tolstikhin et al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS 2021

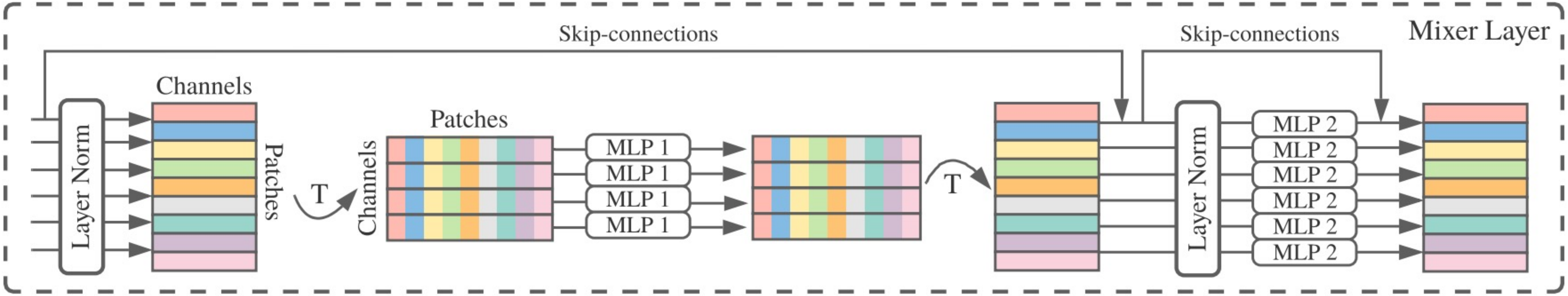
MLP-Mixer: An All-MLP Architecture

Cool idea; but initial ImageNet results not very compelling (but better with JFT pretraining)

MLP-Mixer is actually just a weird CNN???

Equivalent to
Conv(1x1, C->C, stride=1)

Equivalent to
Conv($N^{1/2}$ x $N^{1/2}$, C->C, groups=C)



Input: N x C
N patches with
C channels each

MLP 1: C -> C,
apply to each of
the **N patches**

MLP 2: N -> N,
apply to each of
the **C channels**

Tolstikhin et al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS 2021

MLP-Mixer: Many concurrent and followups

Touvron et al, "ResMLP: Feedforward Networks for Image Classification with Data-Efficient Training", arXiv 2021, https://arxiv.org/abs/2105.03404

Tolstikhin et al, "MLP-Mixer: An all-MLP architecture for vision", NeurIPS 2021, https://arxiv.org/abs/2105.01601

Liu et al, "Pay Attention to MLPs", NeurIPS 2021, https://arxiv.org/abs/2105.08050

Yu et al, "S2-MLP: Spatial-Shift MLP Architecture for Vision", WACV 2022, https://arxiv.org/abs/2106.07477

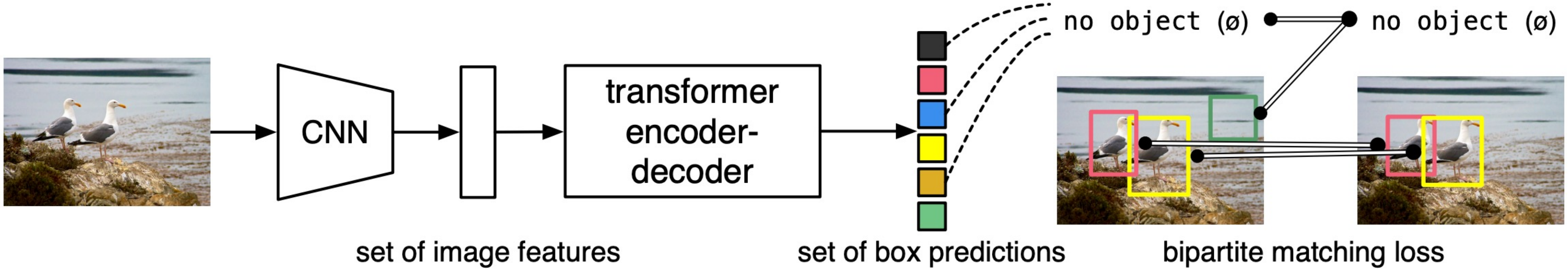
Chen et al, "CycleMLP: A MLP-like Architecture for Dense Prediction", ICLR 2022, https://arxiv.org/abs/2107.10224

Object Detection with Transformers: DETR

Simple object detection pipeline: directly output a set of boxes from a Transformer

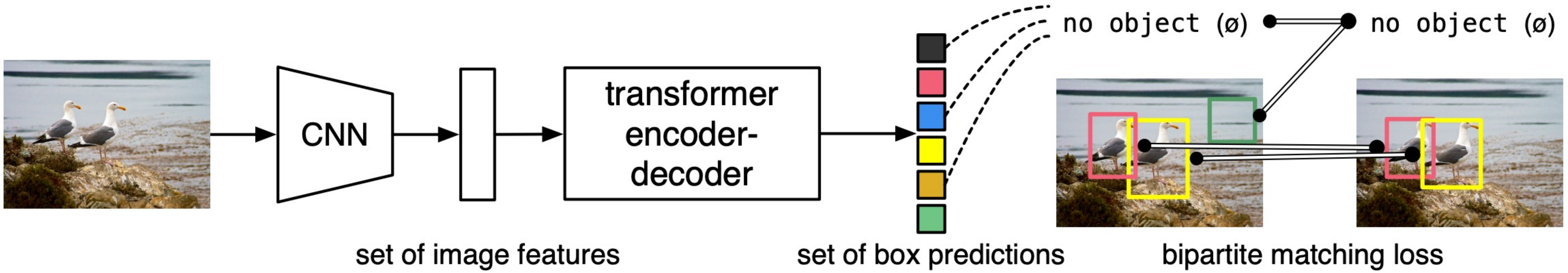
No anchors, no regression of box transforms

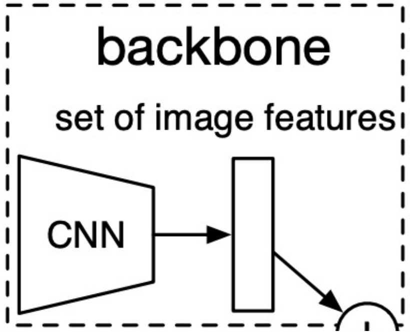
Match predicted boxes to GT boxes with bipartite matching; train to regress box coordinates



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

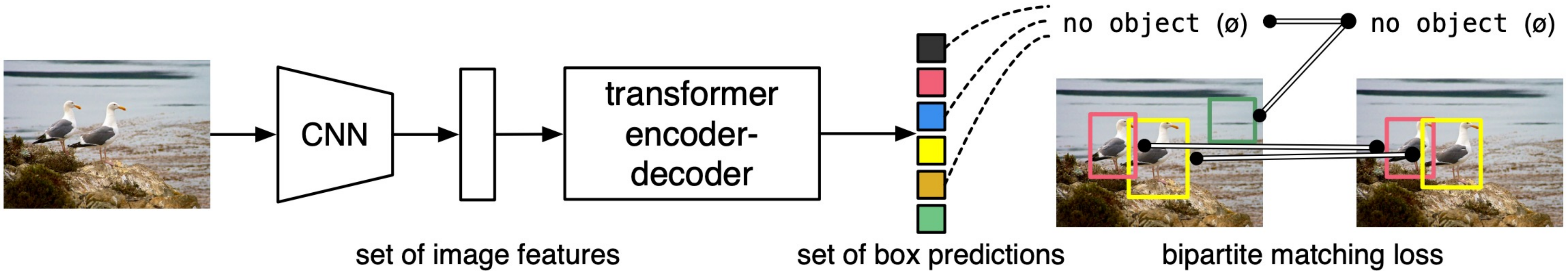
Object Detection with Transformers: DETR

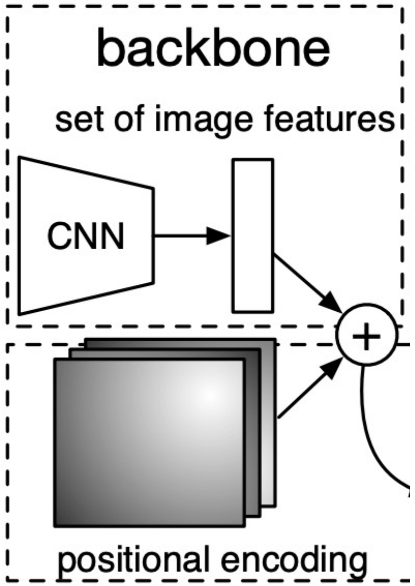




Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

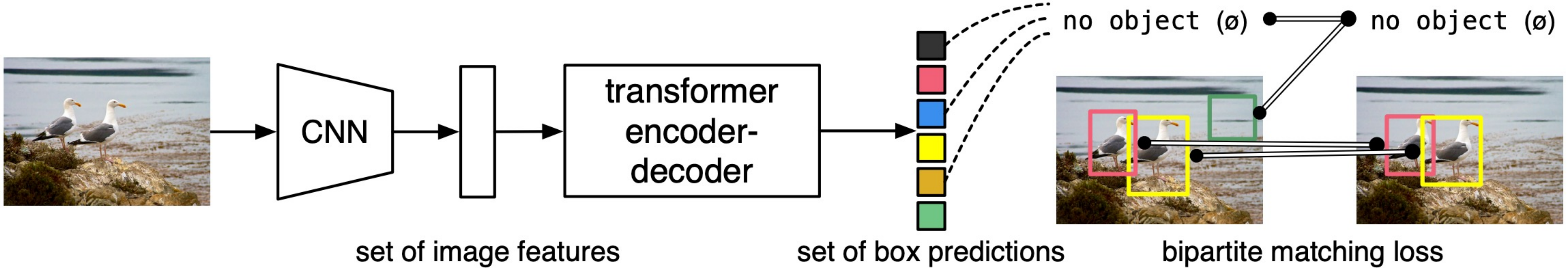
Object Detection with Transformers: DETR

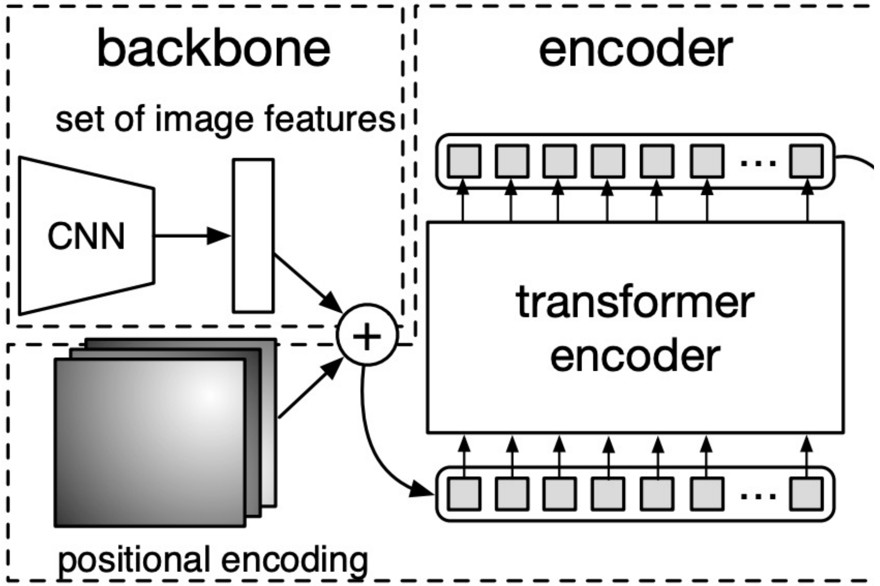




Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

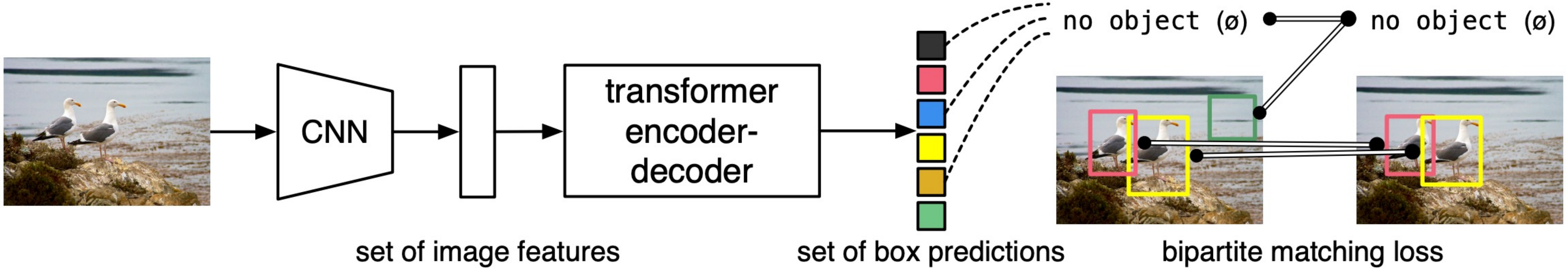
Object Detection with Transformers: DETR

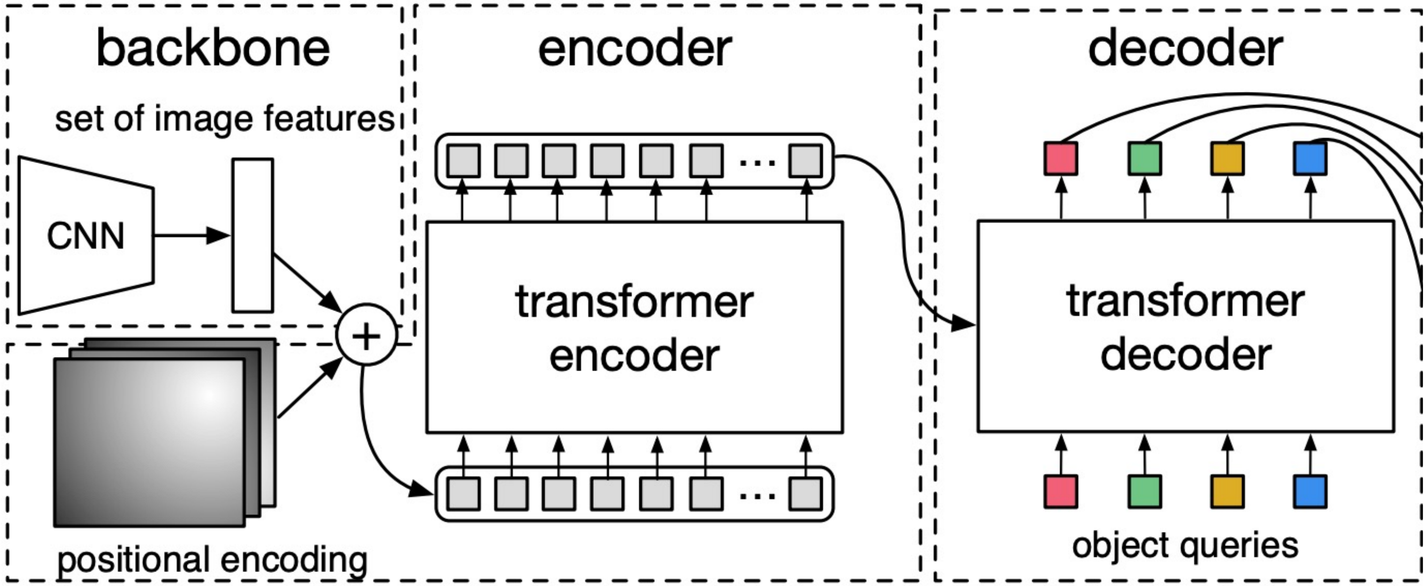




Carion et al, “End-to-End Object Detection with Transformers”, ECCV 2020

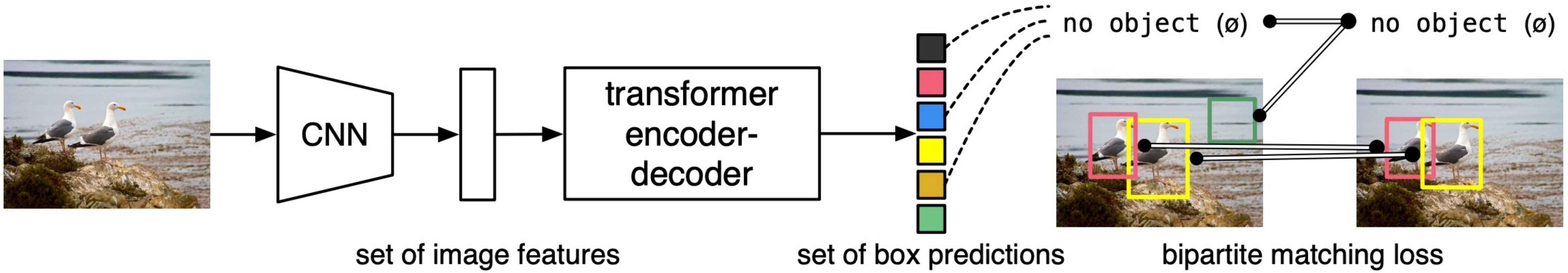
Object Detection with Transformers: DETR

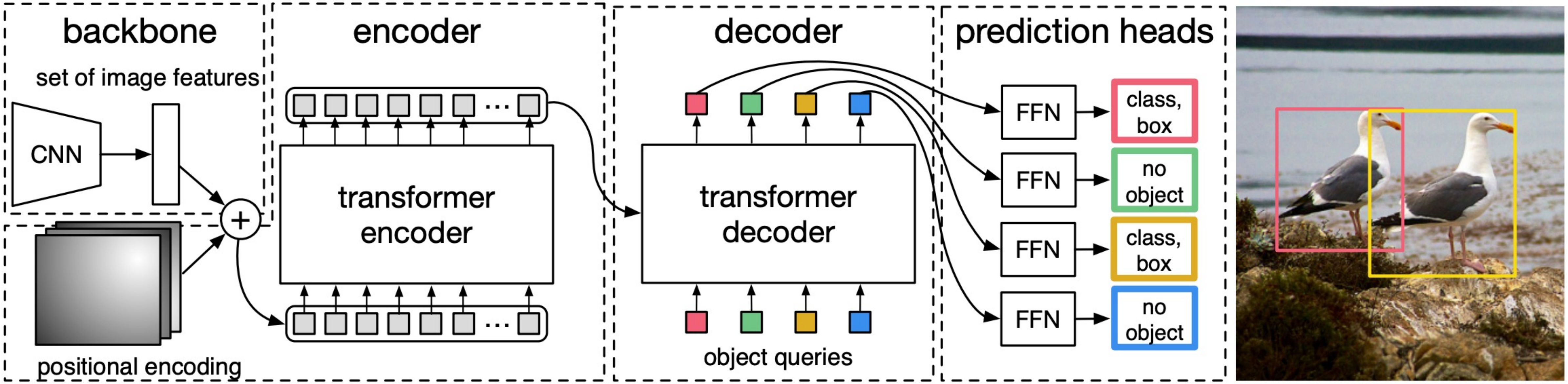




Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object Detection with Transformers: DETR





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

Summary

Vision Transformers have been a super hot topic the past ~1-2 years!

Very different architecture vs traditional CNNs

Applications to all tasks: classification, detection, segmentation, etc

My takeaway: Vison transformers are an evolution, not a revolution. We can still fundamentally solve the same problems as with CNNs.

Main benefit is probably speed: Matrix multiply is more hardware-friendly than convolution, so ViTs with same FLOPs as CNNs can train and run much faster

Next week: Generative Models