Advanced Computer Vision

Tanveer Hussain
htanveer3797@gmail.com

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

Course designed by **Naeem Ullah Khan** (naeemullah.khan@kaust.edu.sa), updated by Tanveer Hussain

This Course

1. Recurrent Neural Networks and Attention
2. General Attention and ViT
3. Video Classification
4. Generative AI (Part 1)
5. Generative AI (Part 2), CLIP model

Image Captioning

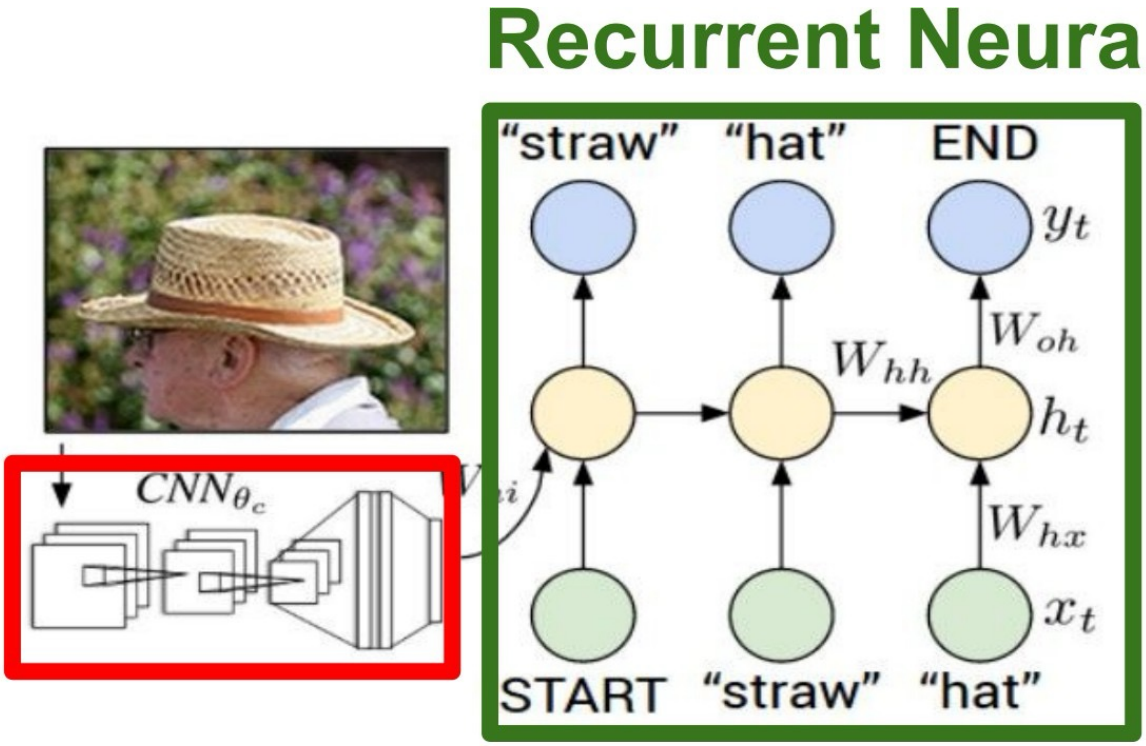


Image Captioning

- A computer Vision task in which we generate captions for images

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

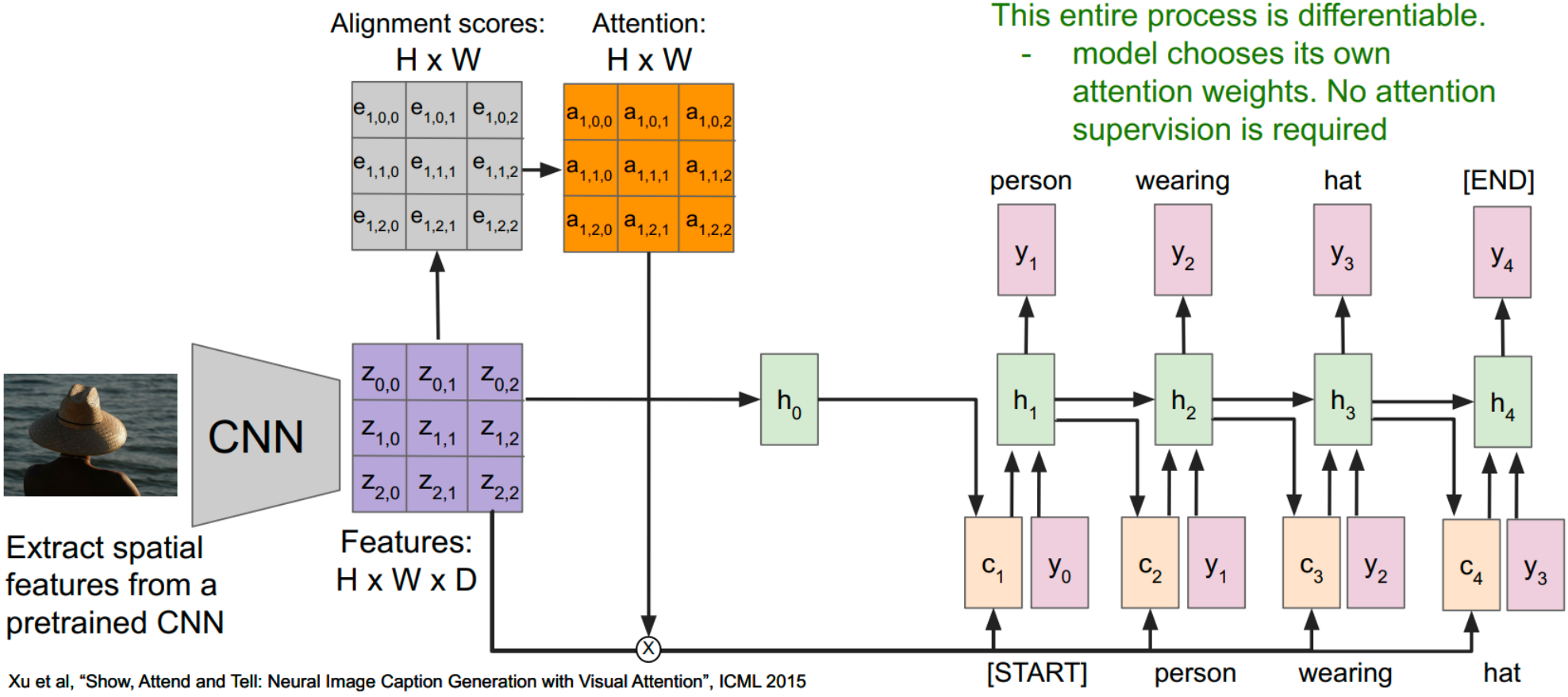
Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

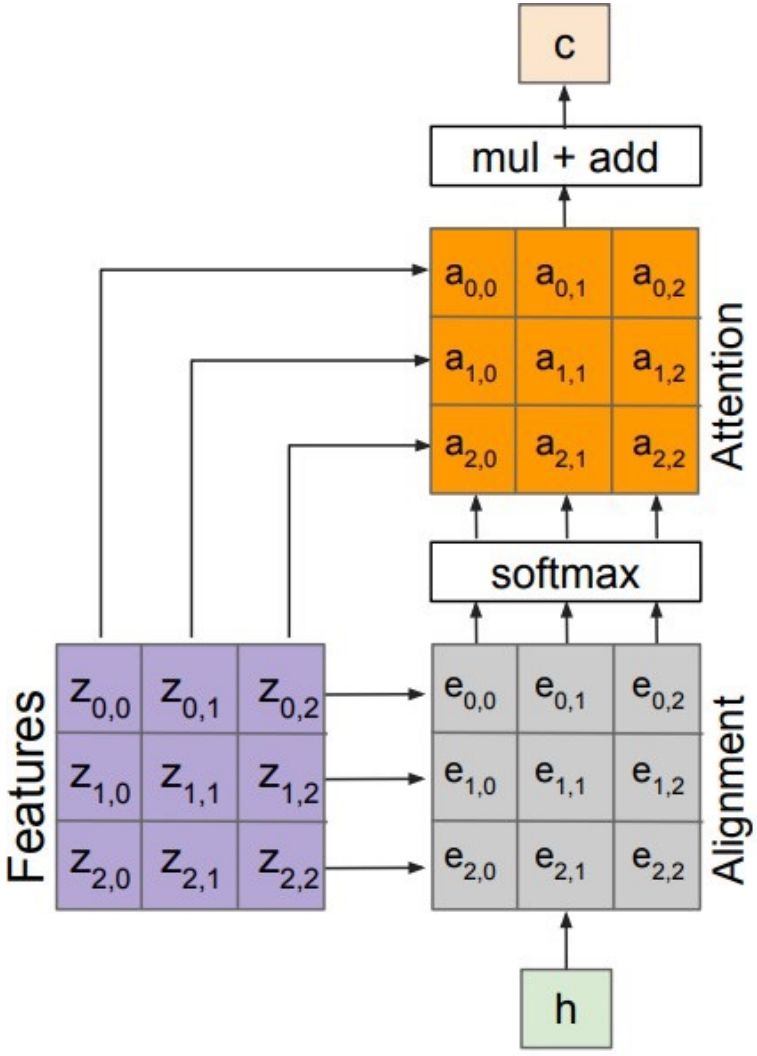
A man riding a dirt bike on a dirt track

Image Captioning with RNNs and Attention



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention we just saw in image captioning

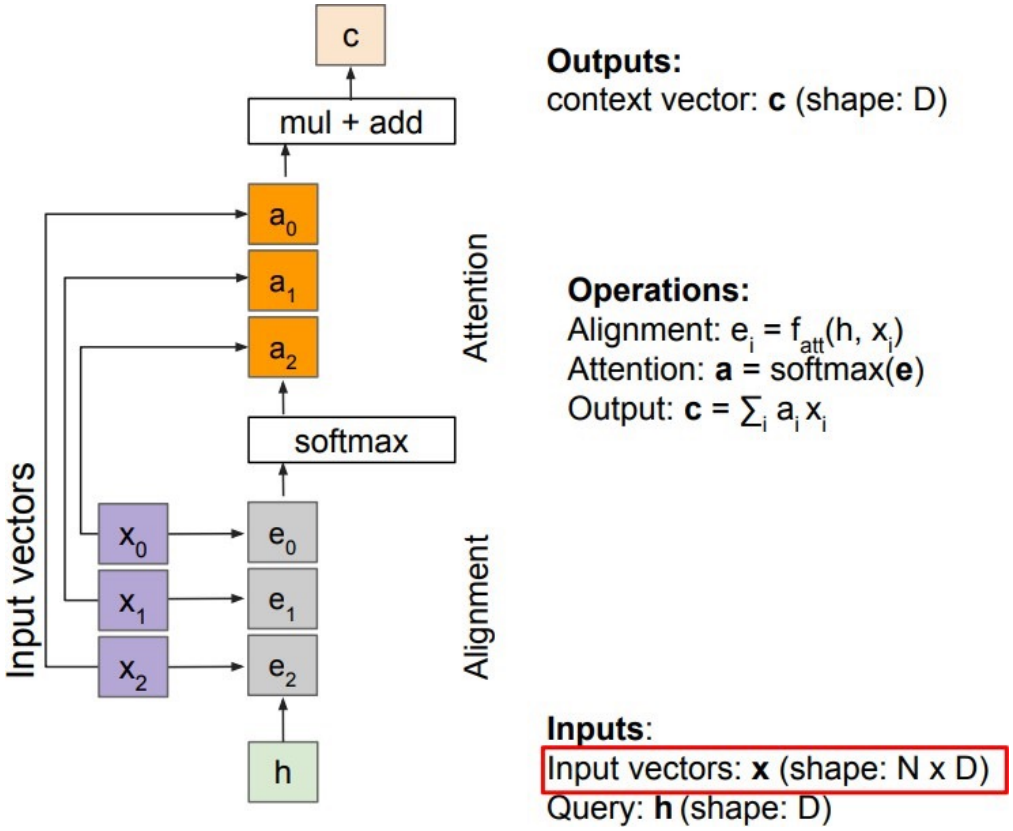


Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_{i,j} a_{i,j} z_{i,j}$

Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D)

General Attention Layer



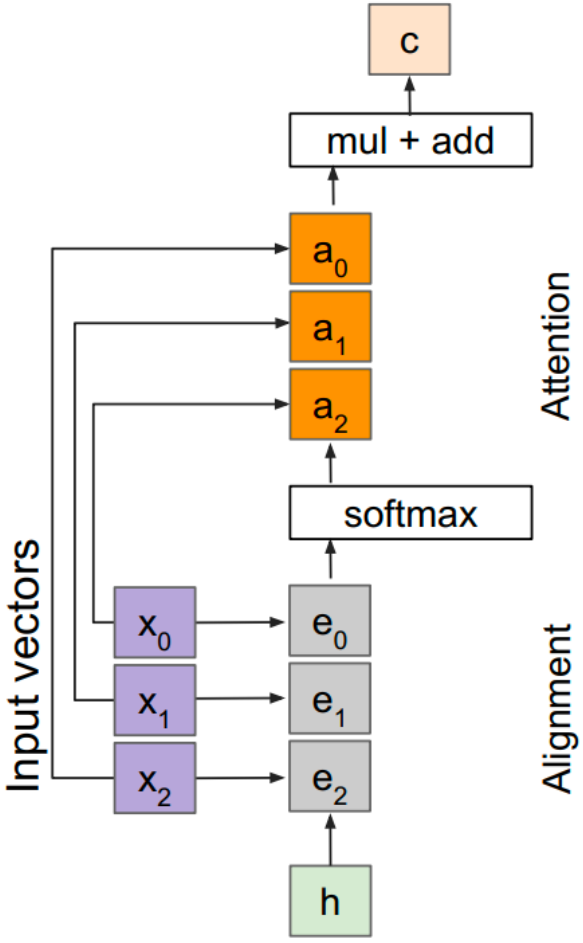
Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_i = f_{att}(h, x_i)$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_i a_i x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

- Attention operation is permutation invariant
- Doesn't care about ordering of the features
- Stretch H x W = N into N vectors

General Attention Layer



Outputs:
context vector: **c** (shape: D)

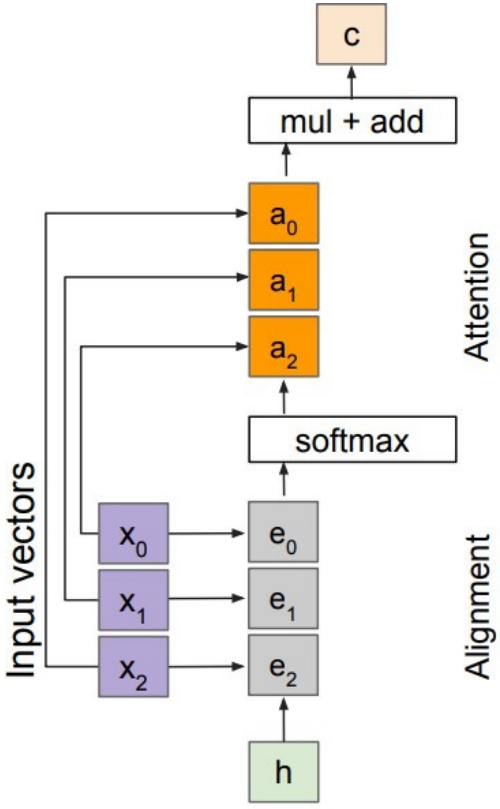
Operations:
Alignment: $e_i = h \cdot x_i$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

Change $f_{att}(.)$ to a simple dot product

- only works well with key & value transformation trick (will mention in a few slides)

Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

General Attention Layer



Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_i = h \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

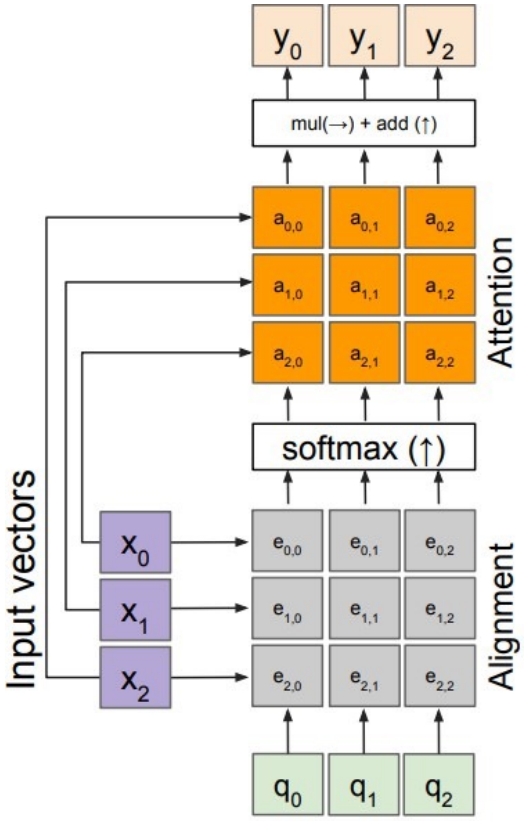
This scaling helps to:

- ❑ Normalize the logits to a more consistent range, reducing variance.
- ❑ Prevent extremely high values that lead to low-entropy softmax outputs.
- ❑ Ensures the softmax distribution remains balanced and informative.

Change $f_{att}(.)$ to a scaled simple dot product

- Larger dimensions means more terms in the dot product sum.
- So, the variance of the logits is higher. Large magnitude vectors will produce much higher logits.
- So, the post-softmax distribution has lower-entropy, assuming logits are IID.
- Ultimately, these large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Divide by $\sqrt{D}$ to reduce effect of large magnitude vectors

General Attention Layer



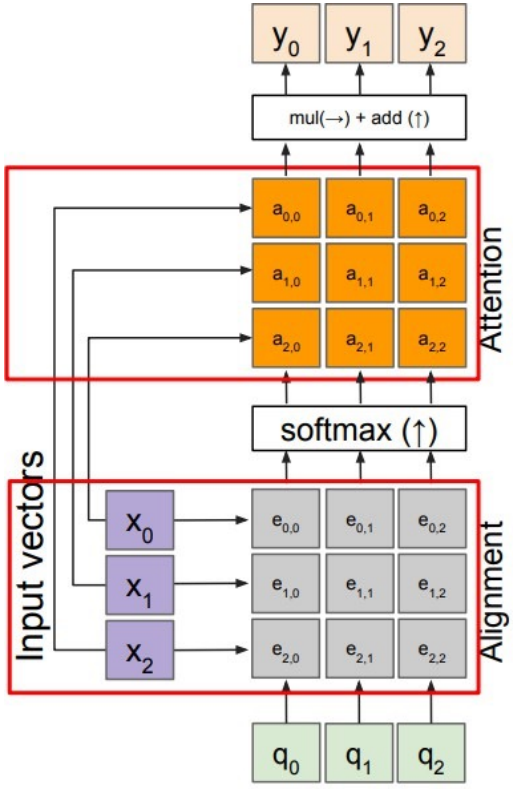
Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D)

- We can have multiple Query Vectors.
- Each query creates a new output context vector

General Attention Layer



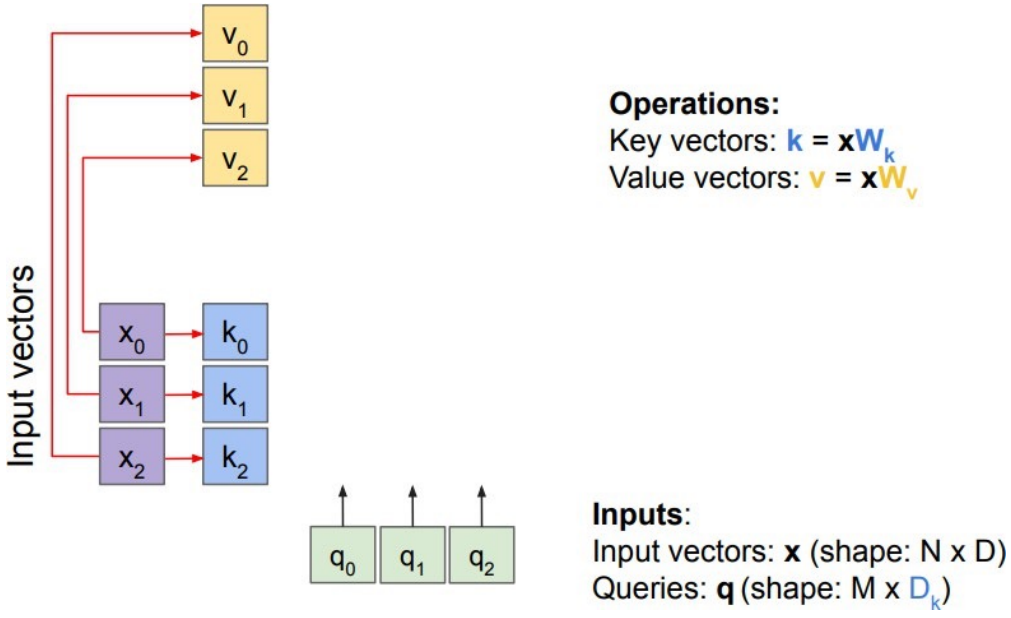
Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D)

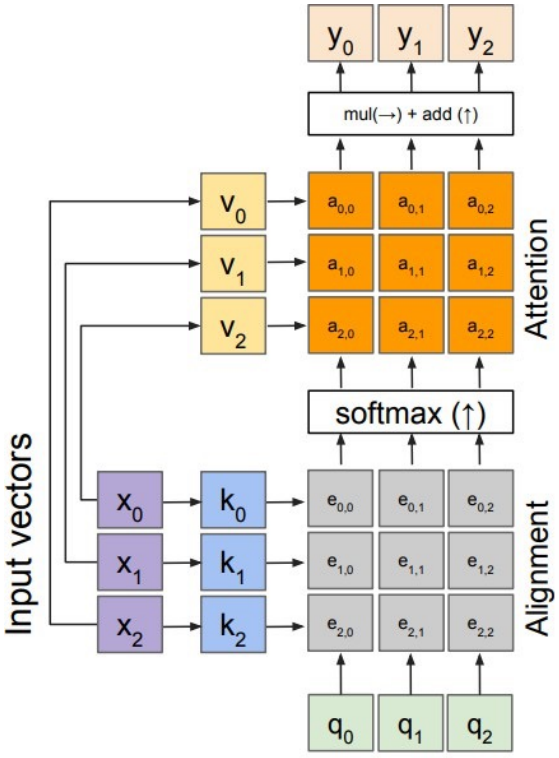
- Notice that the input vectors are used for both the alignment as well as the attention calculations.

General Attention Layer



- Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

General Attention Layer



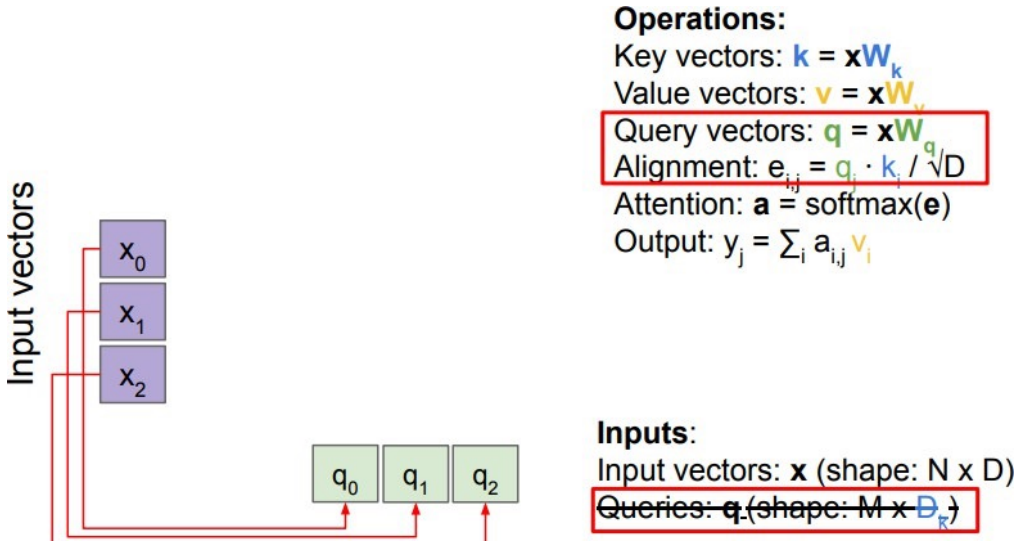
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

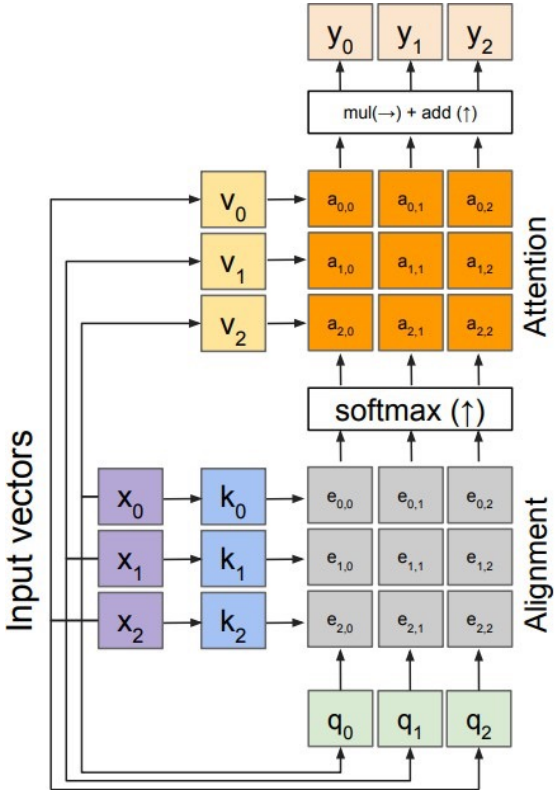
- Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.
- The input and output dimensions can now change depending on the key and value FC layers

Self Attention Layer



- Recall that the query vector was a function of the input vectors
- We can calculate the query vectors from the input vectors, therefore, defining a ”self-attention” layer.
- No input query vectors anymore
- Instead, query vectors are calculated using a FC layer.

Self Attention Layer



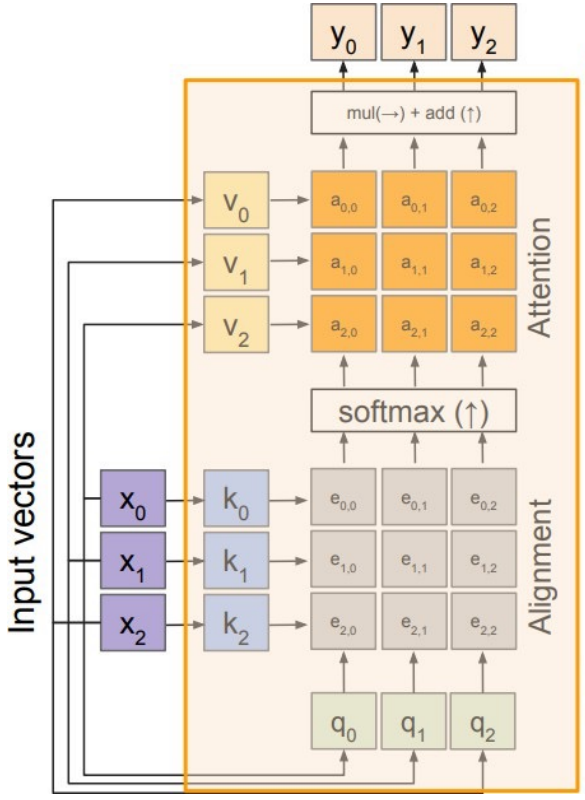
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)

- Recall that the query vector was a function of the input vectors
- We can calculate the query vectors from the input vectors, therefore, defining a ”self-attention” layer.
- No input query vectors anymore
- Instead, query vectors are calculated using a FC layer.

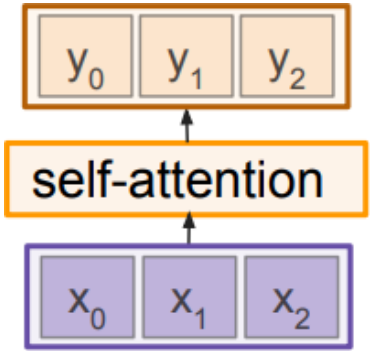
Self Attention Layer



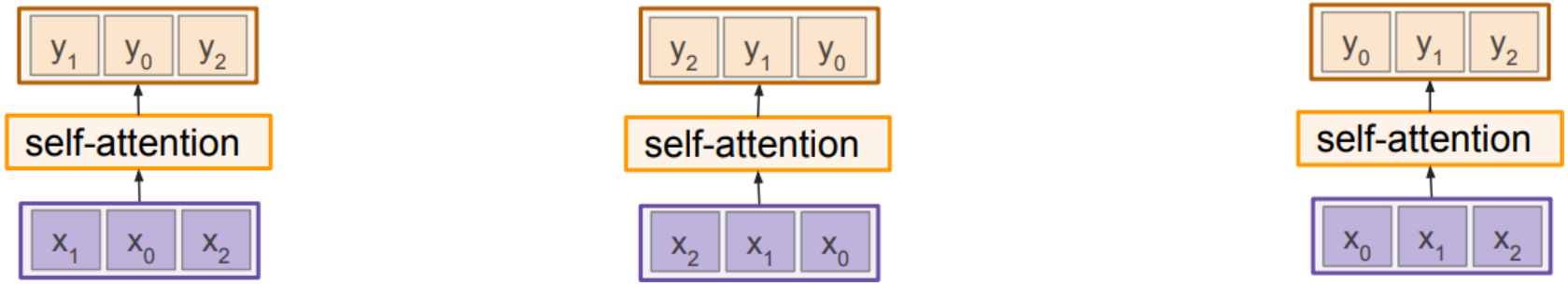
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)



Permutation Invariance

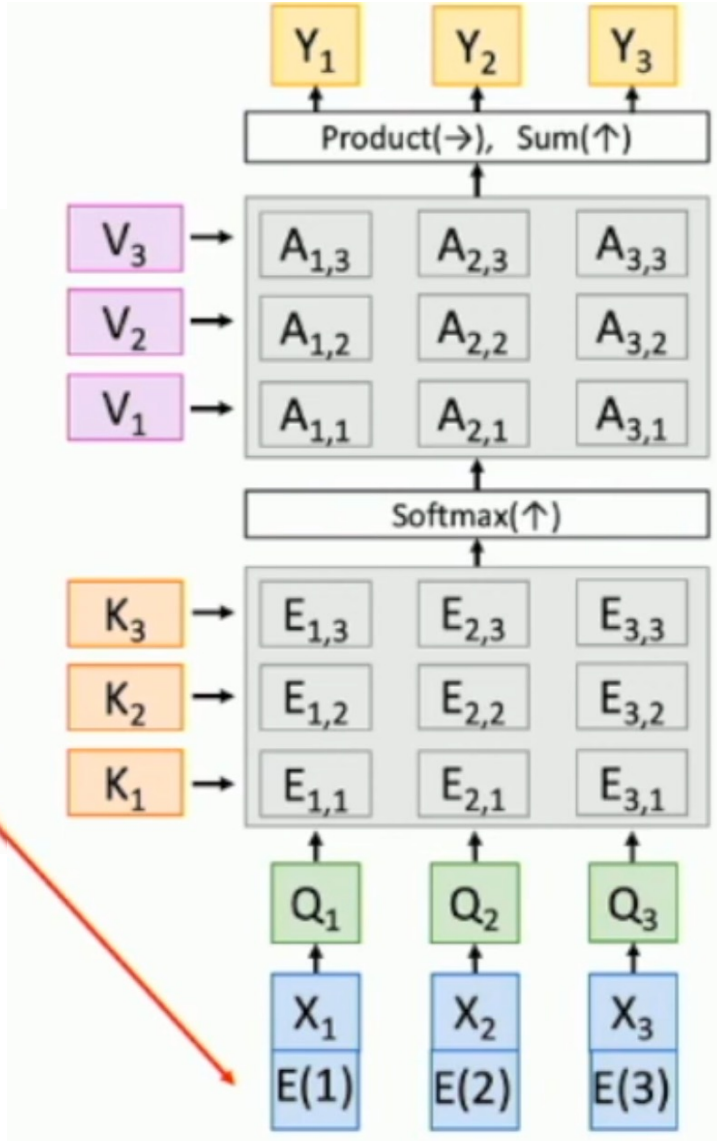


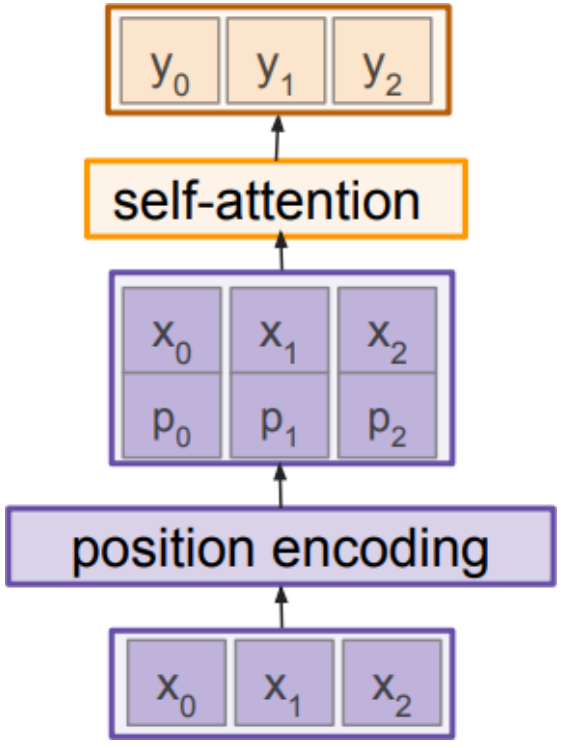
- Self-Attention Layer is Permutation equivariant
- It doesn't care about the orders of the inputs!
- **Problem:** How can we encode ordered sequences like language or spatially ordered image features?

Positional encoding

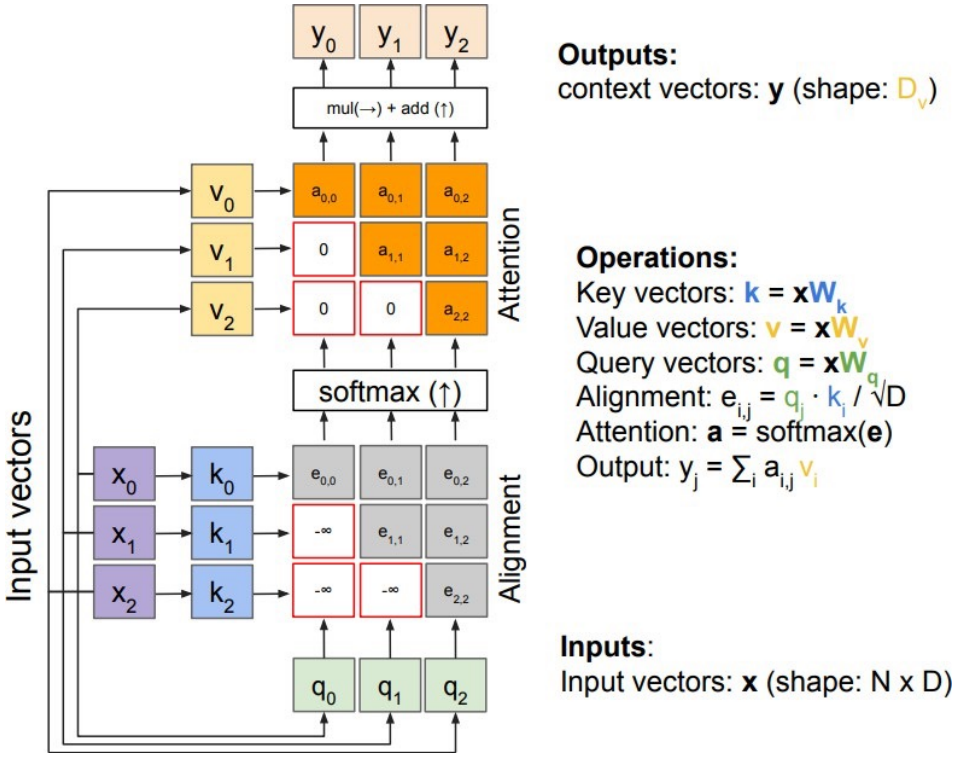
Self attention doesn't "know" the order of the vectors it is processing!

In order to make processing position-aware, concatenate input with **positional encoding**





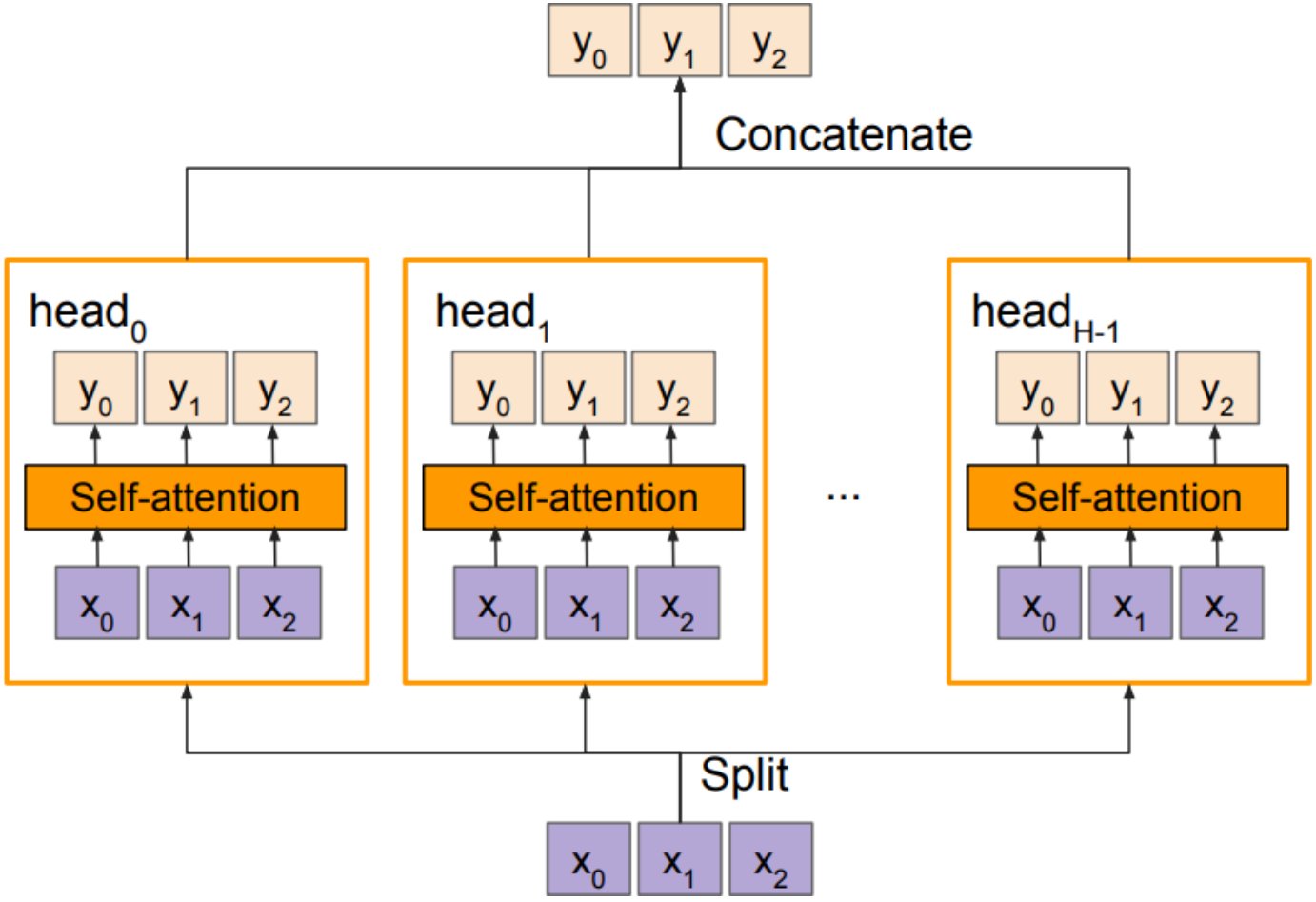
Masked self-attention layer



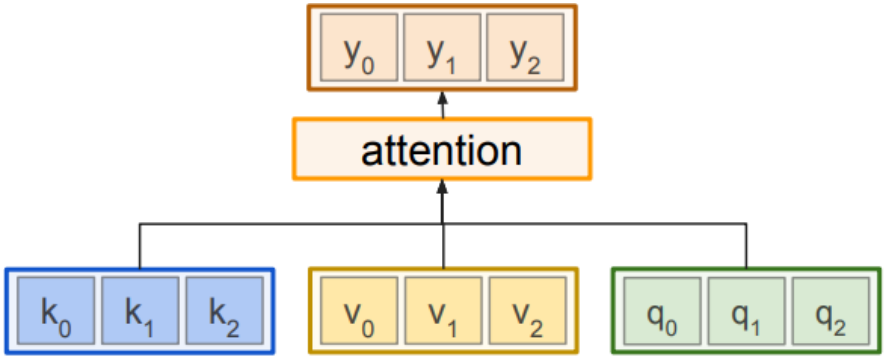
- Prevent vectors from looking at past/future vectors.
- Manually set alignment scores to $-\infty$

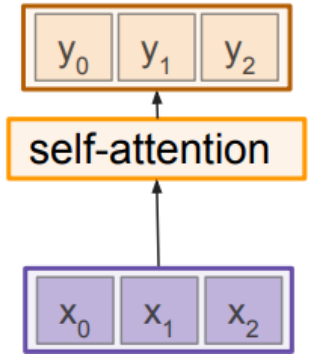
Multi-head self-attention layer

- Multiple self-attention heads in parallel

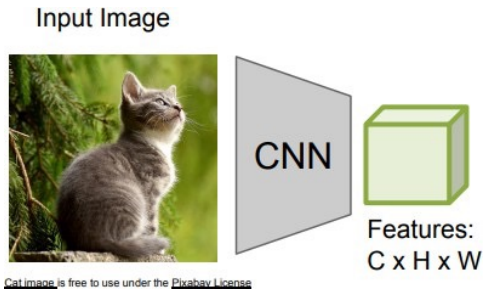


General attention versus self-attention

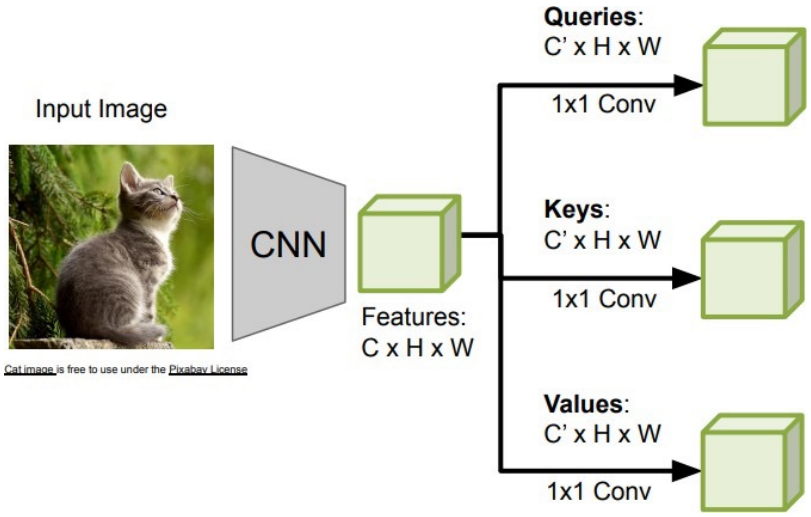




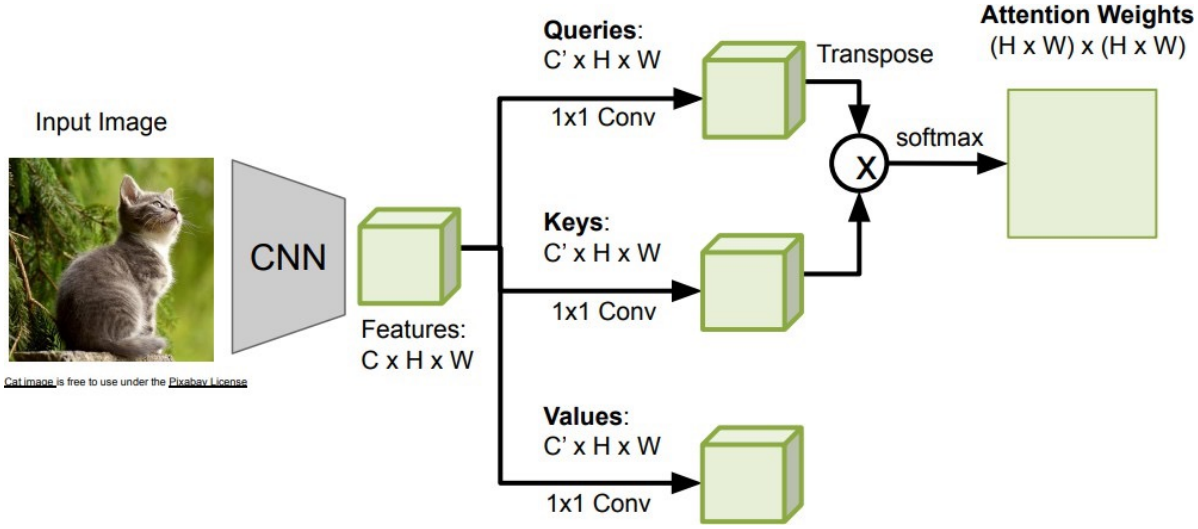
Example: CNN with Self-Attention



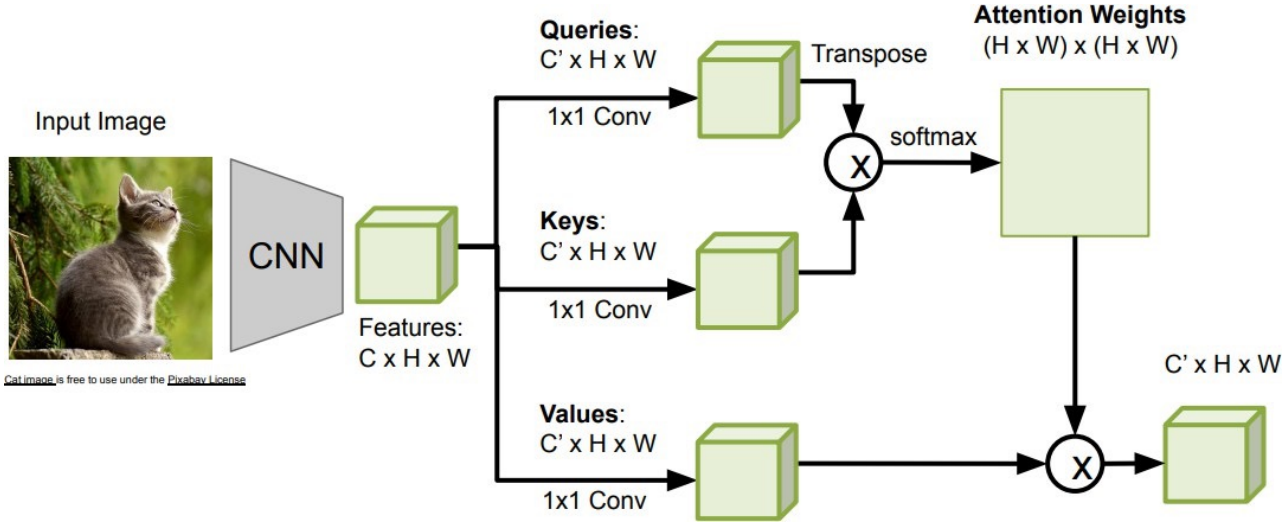
Example: CNN with Self-Attention



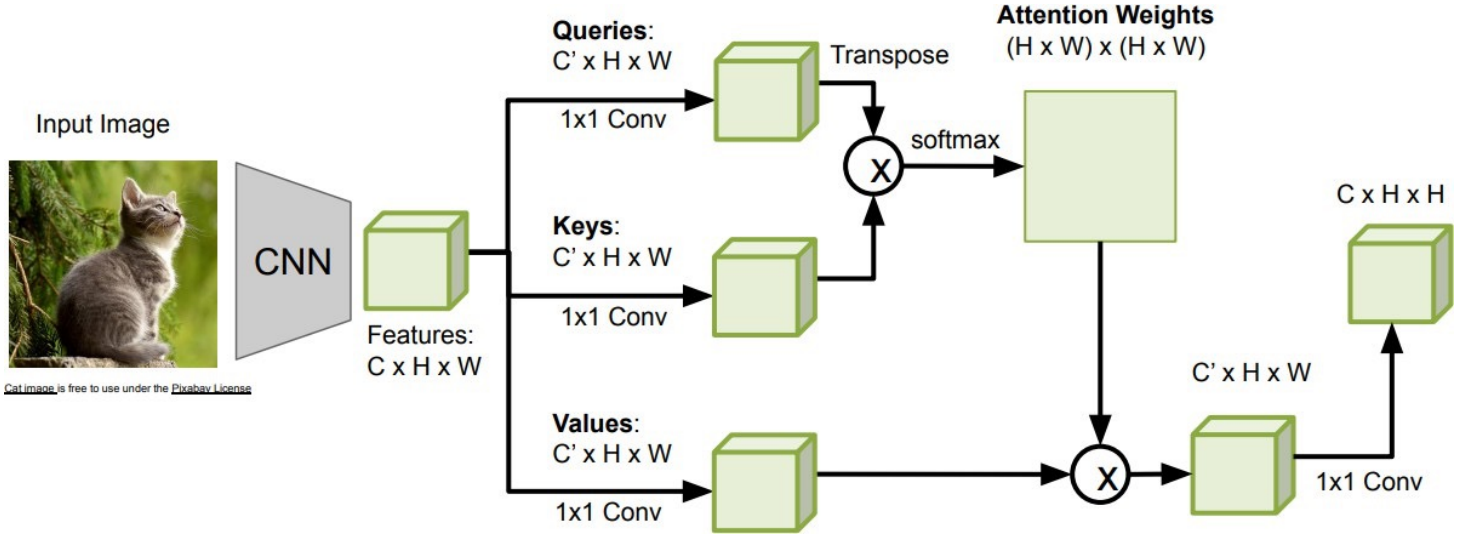
Example: CNN with Self-Attention



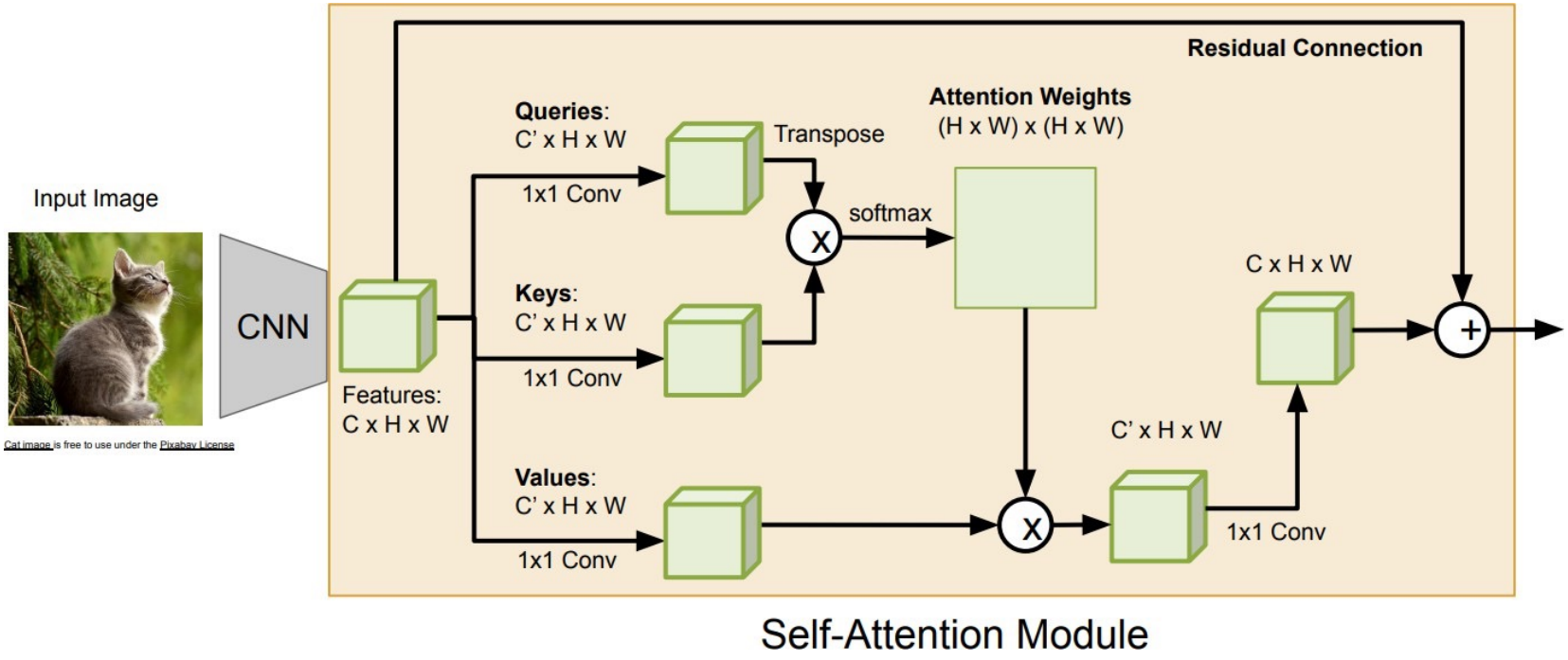
Example: CNN with Self-Attention



Example: CNN with Self-Attention



Example: CNN with Self-Attention



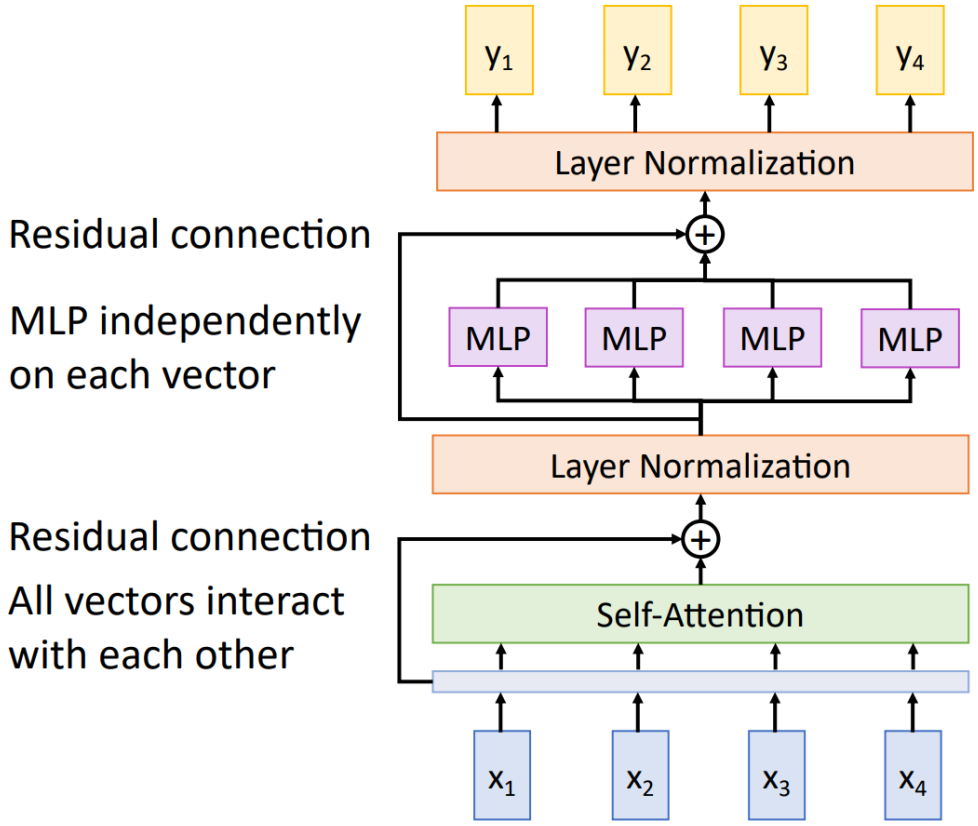
Attention is all you need

Vaswani et al, NeurIPS 2017

Transformer Block

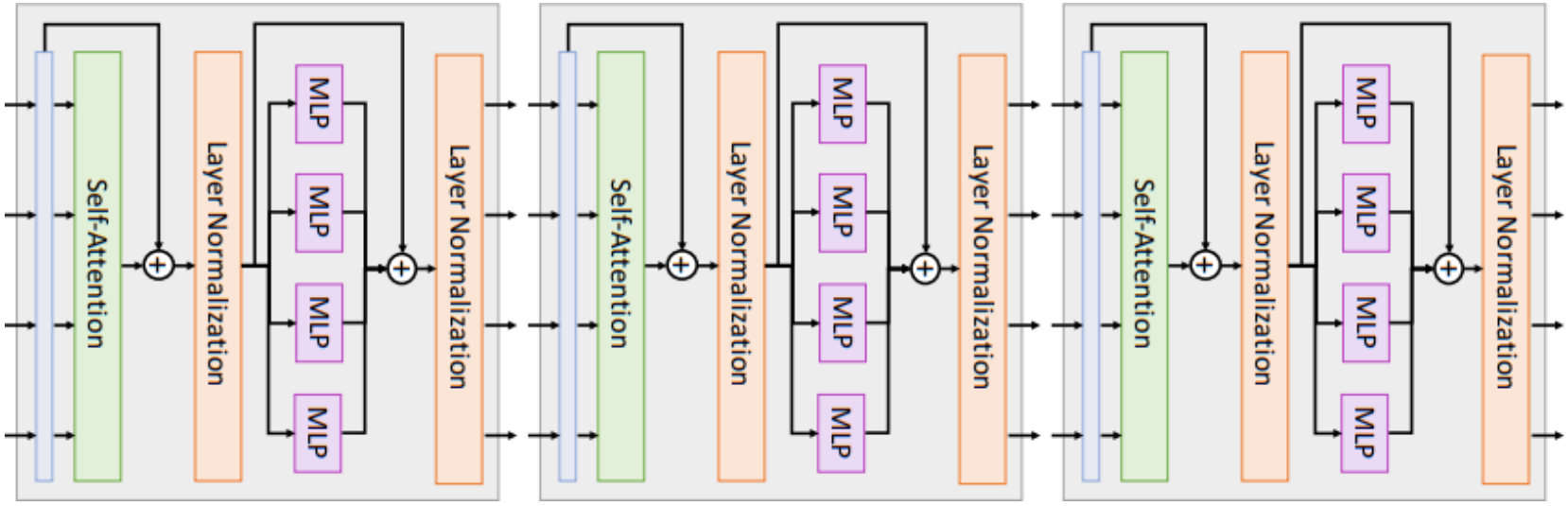
- **Input:** Set of vectors x
- **Output:** Set of vectors y

- Self-attention is the only interaction between vectors!
- Layer norm and MLP work independently per vector
- Highly scalable, highly parallelizable

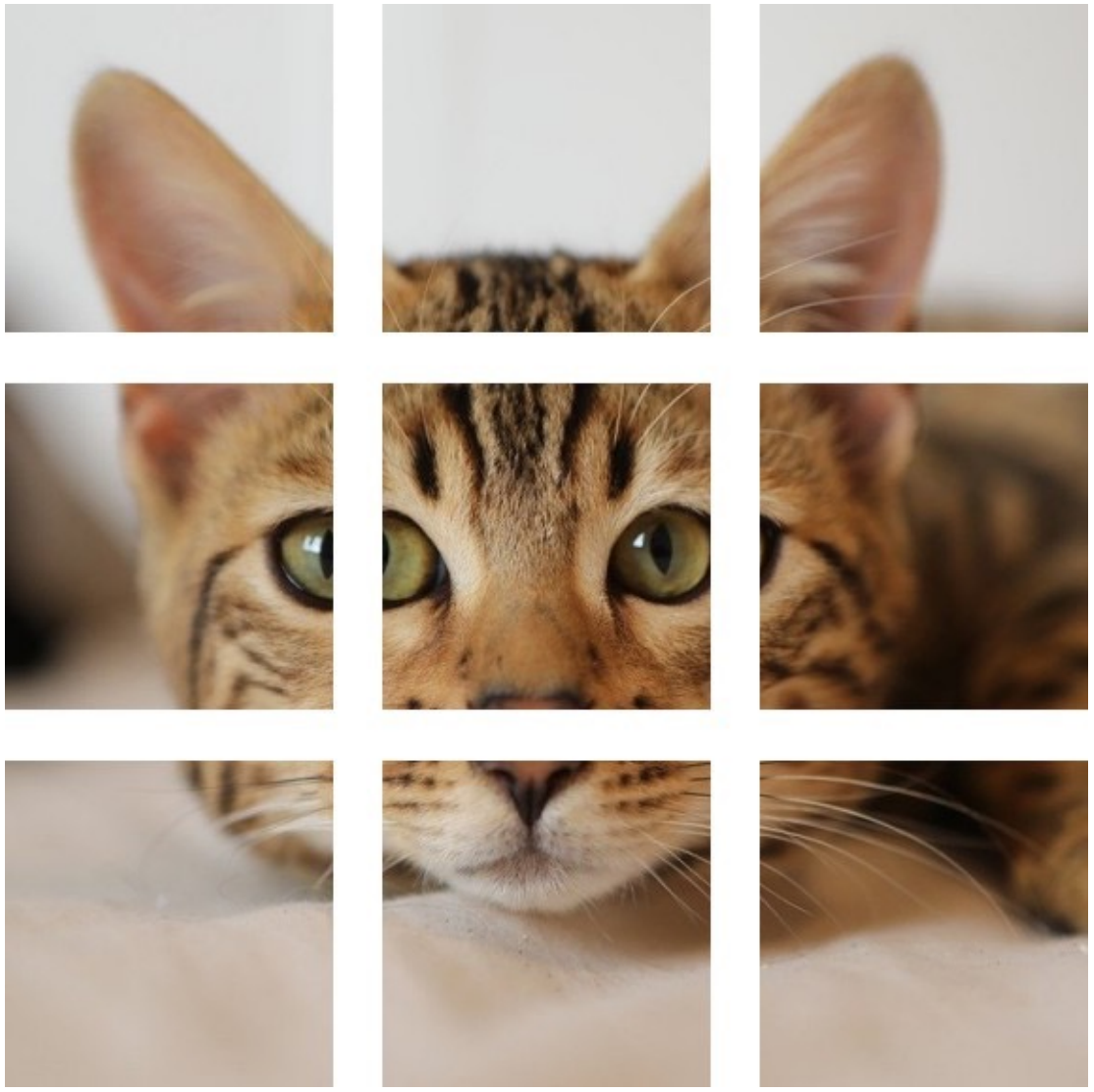


Transformers

- A Transformer is a sequence of transformer blocks
- Vaswani et al used 12 blocks, DQ = 512 and 6 attention heads

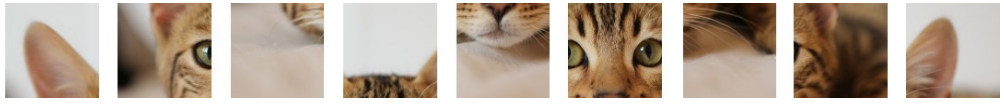


Vision Transformers



Vision Transformers

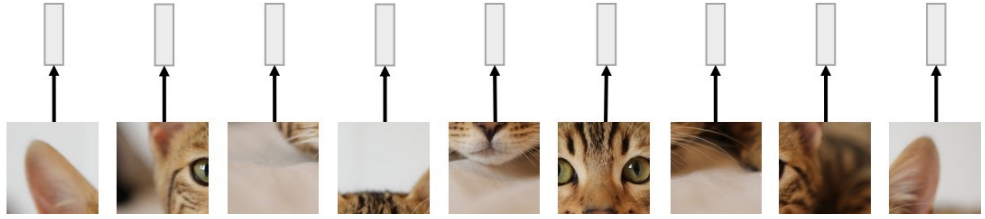
N input patches, each of shape 3x16x16



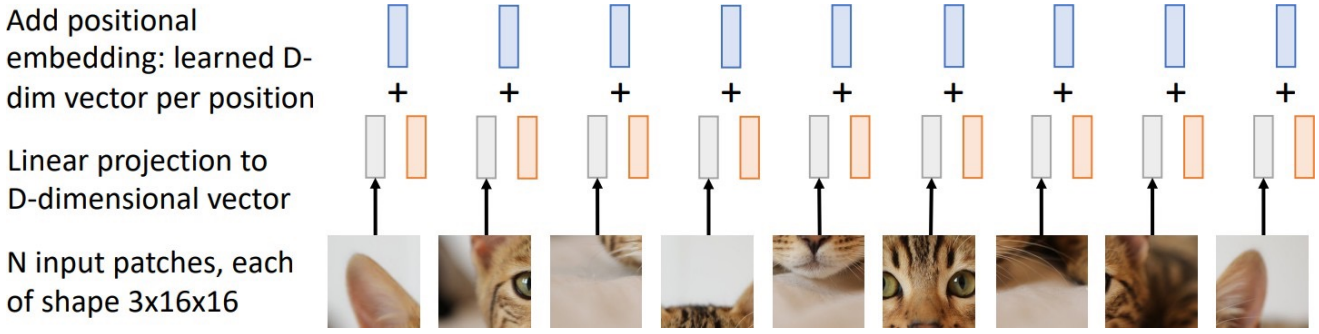
Vision Transformers

Linear projection to D-dimensional vector

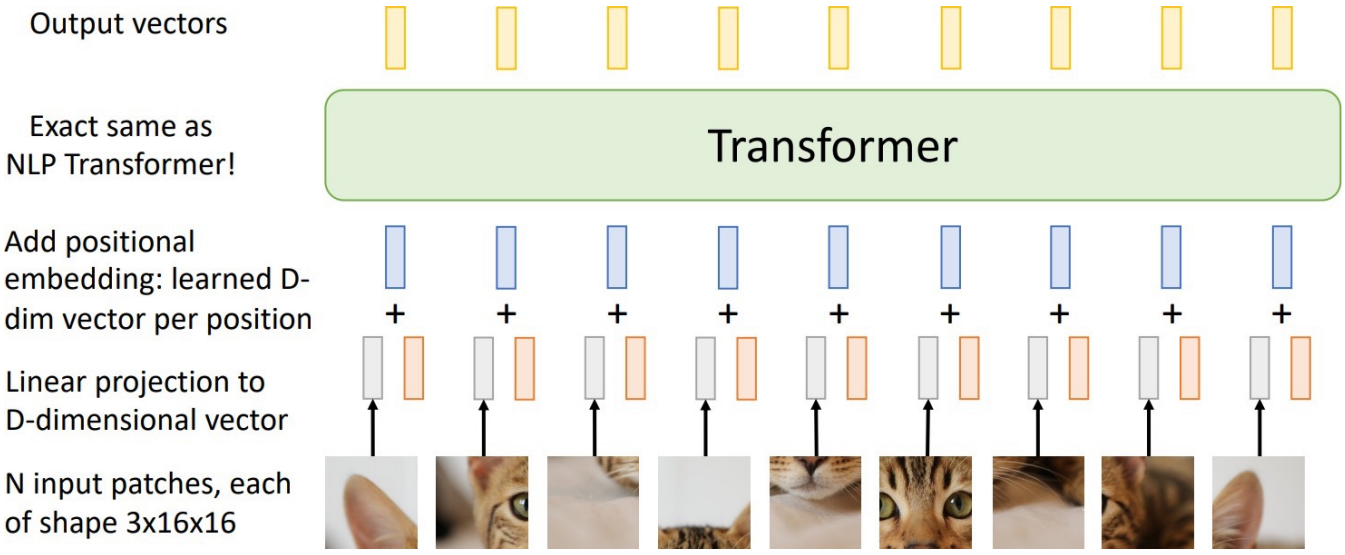
N input patches, each of shape 3x16x16



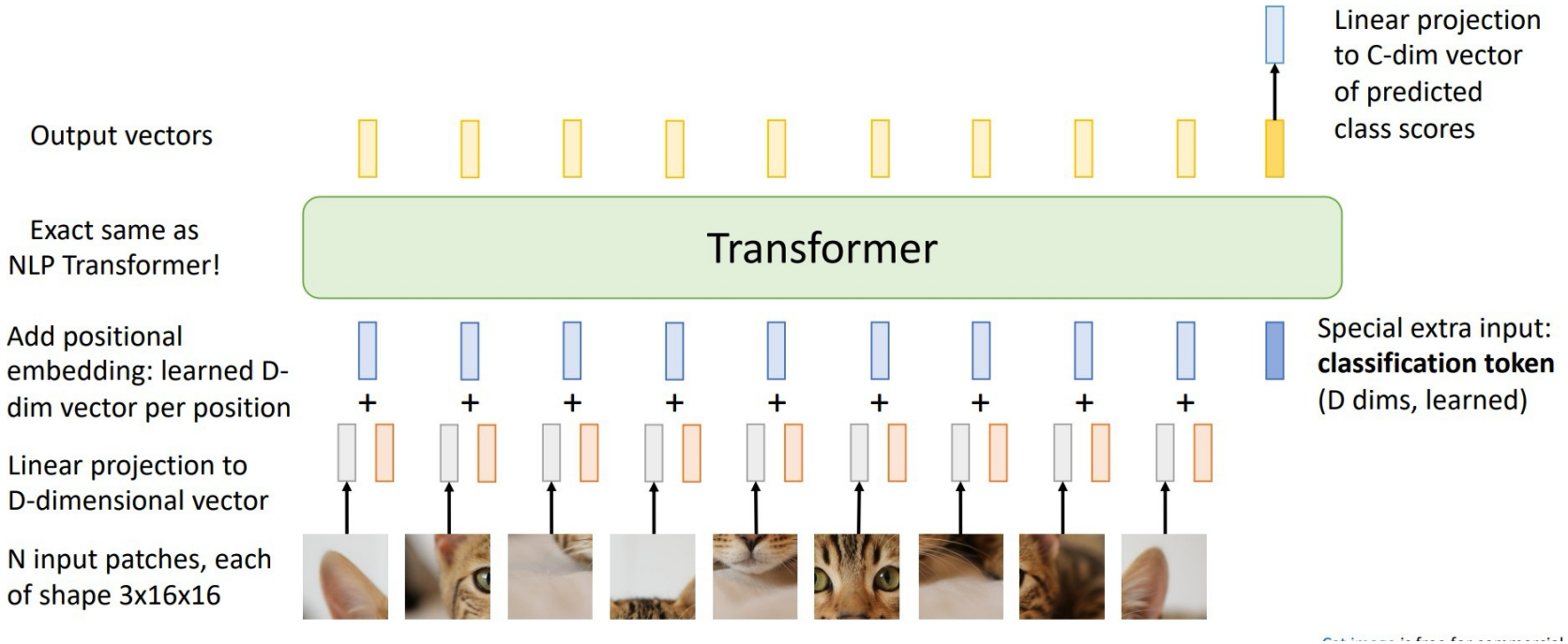
Vision Transformers



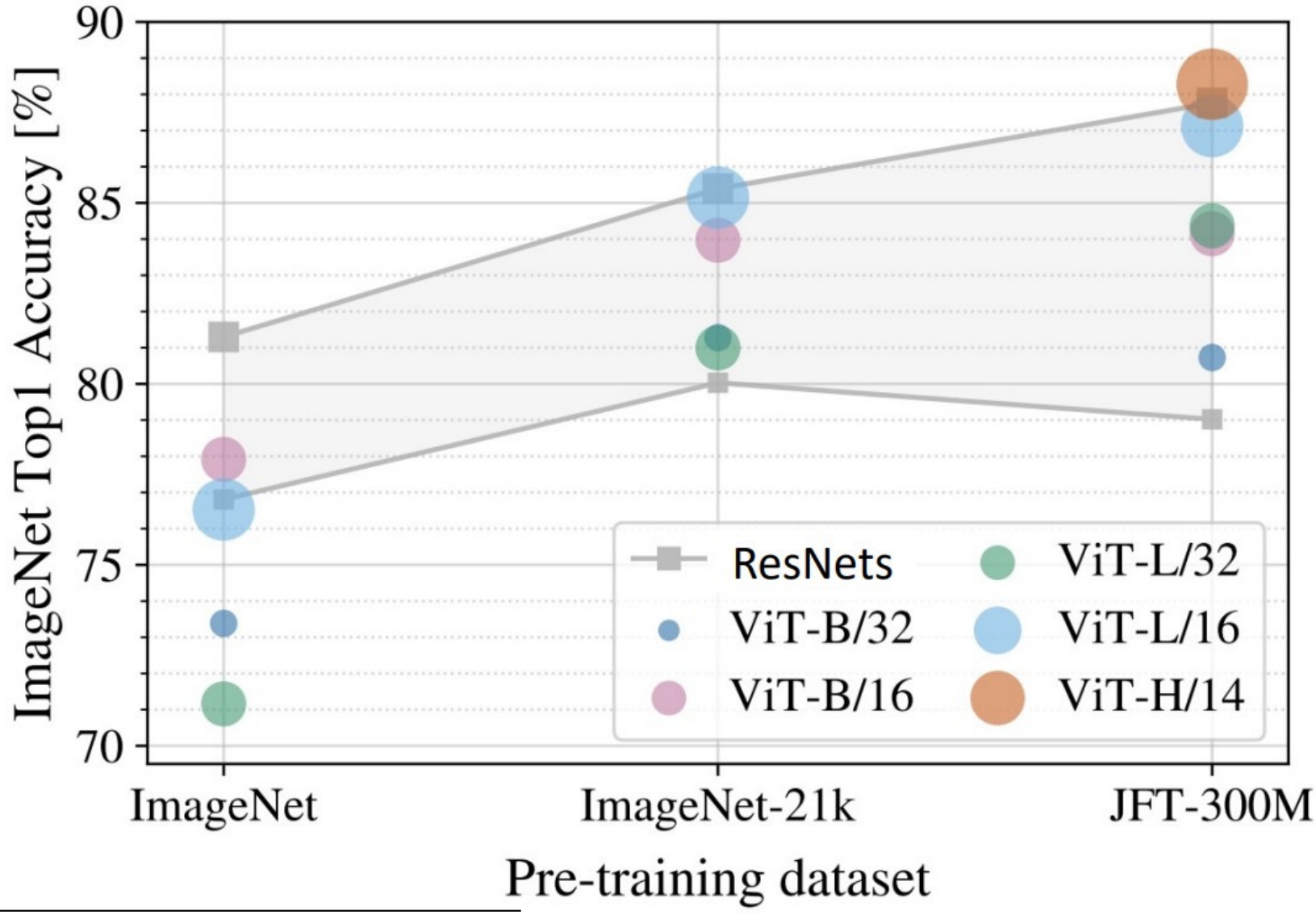
Vision Transformers



Vision Transformers



Vision Transformers



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

Improving ViT: Distillation

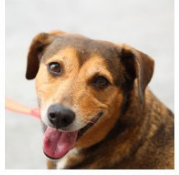
Step 1: Train a teacher CNN on ImageNet

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

Cross Entropy Loss

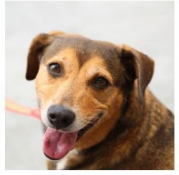
GT label: Cat

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

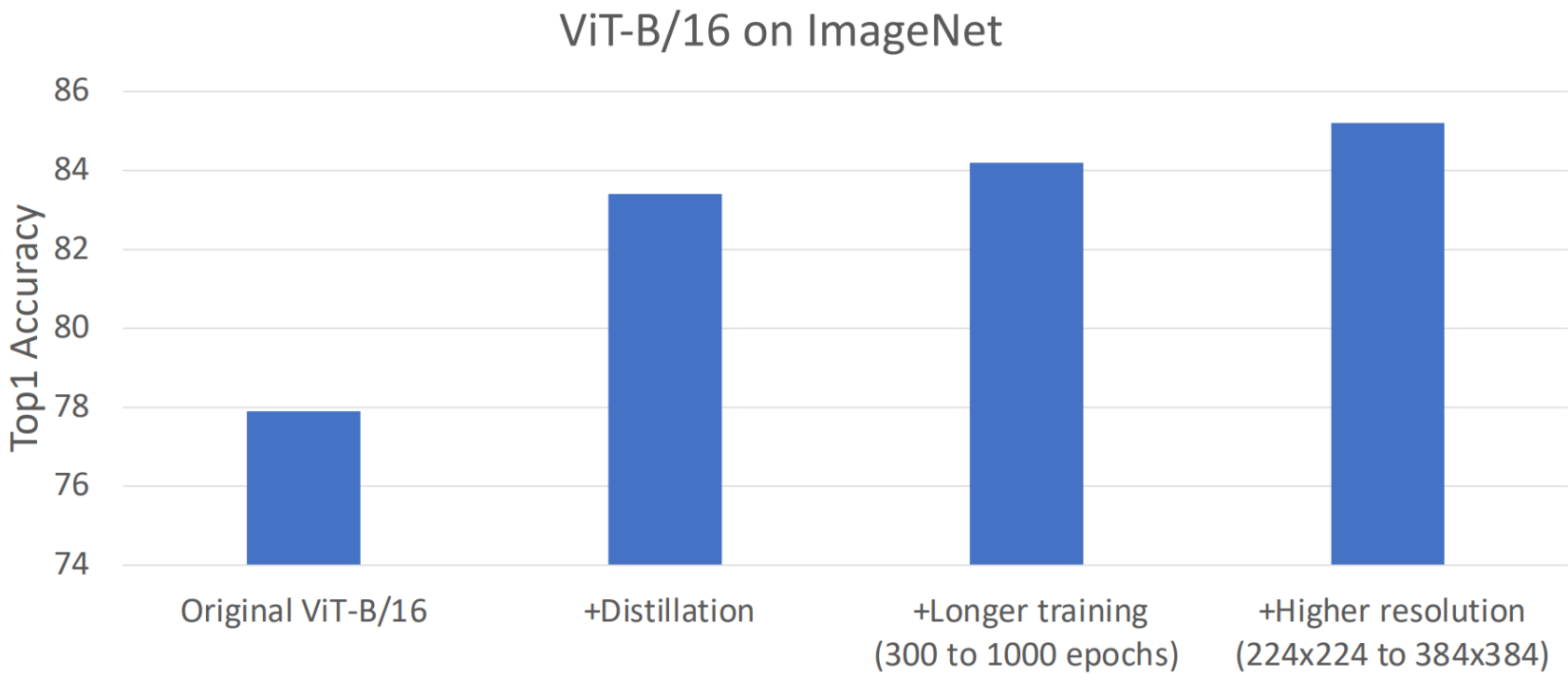


P(cat) = 0.2
P(dog) = 0.8

Cross Entropy Loss

GT label: Dog

Improving ViT: Distillation



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

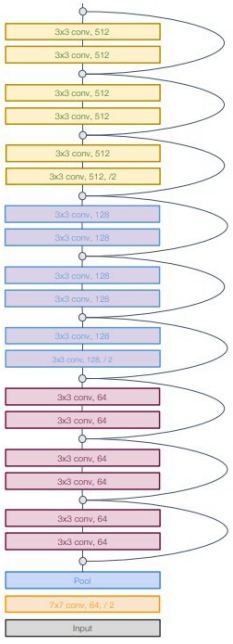
CNN vs ViT

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

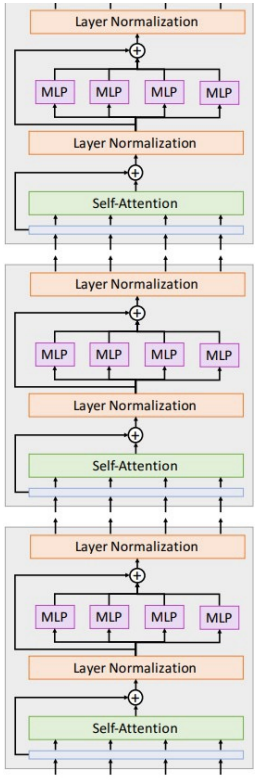


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

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?



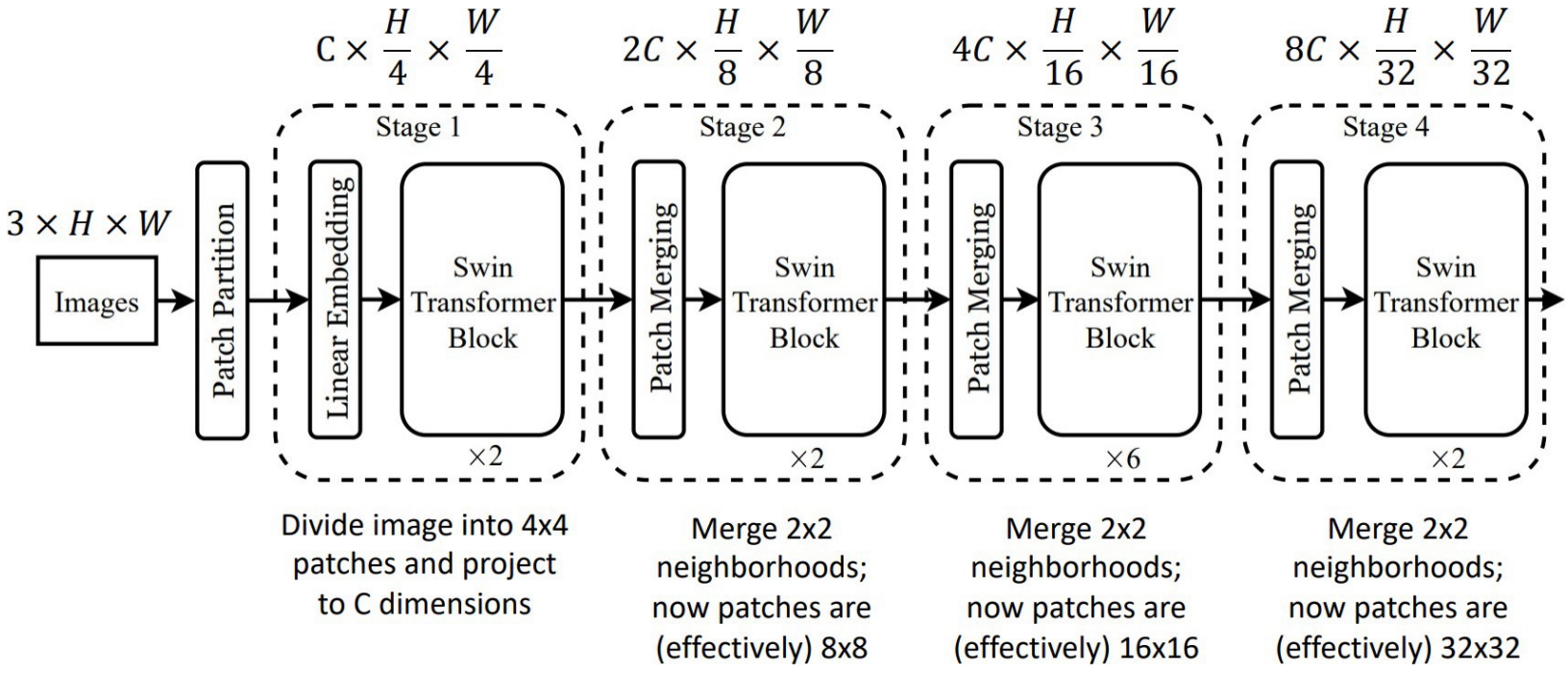
3rd block:
768 x 14 x 14

2nd block:
768 x 14 x 14

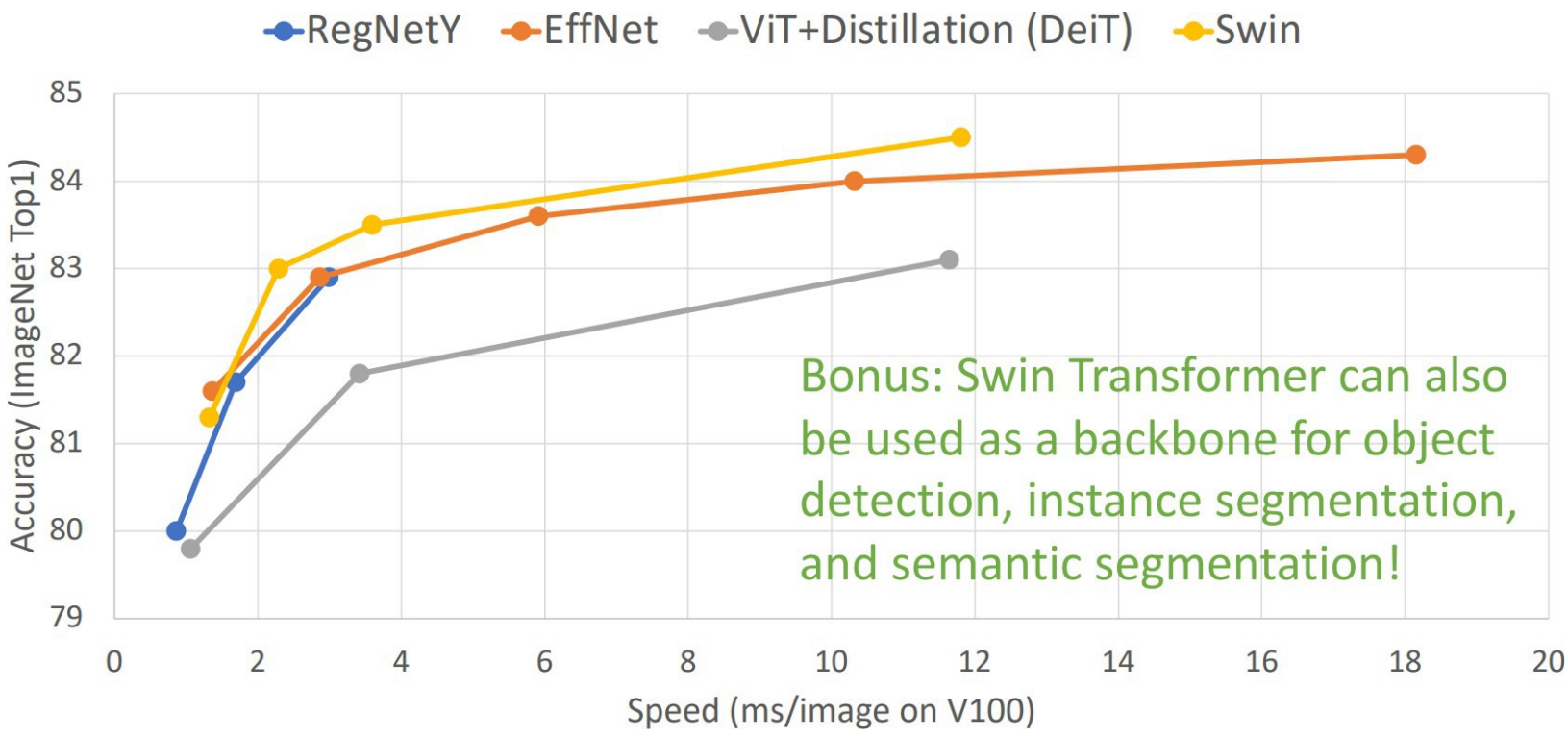
1st block:
768 x 14 x 14

Input:
3 x 224 x 224

Hierarchical ViT: Swin Transformer



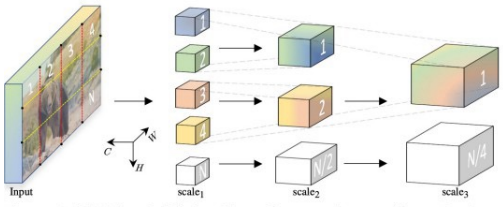
Hierarchical ViT: Swin Transformer



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

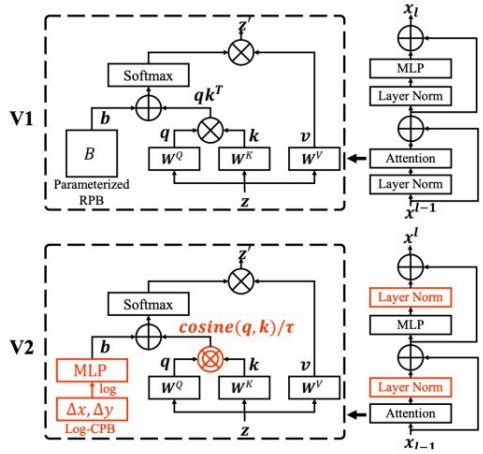
Other Hierarchical Vision Transformers

MViT



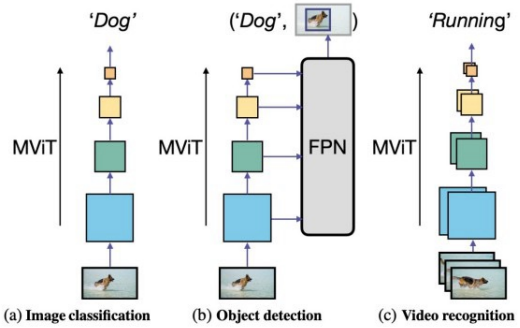
Fan et al, "Multiscale Vision Transformers", ICCV 2021

Swin-V2



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

Improved MViT



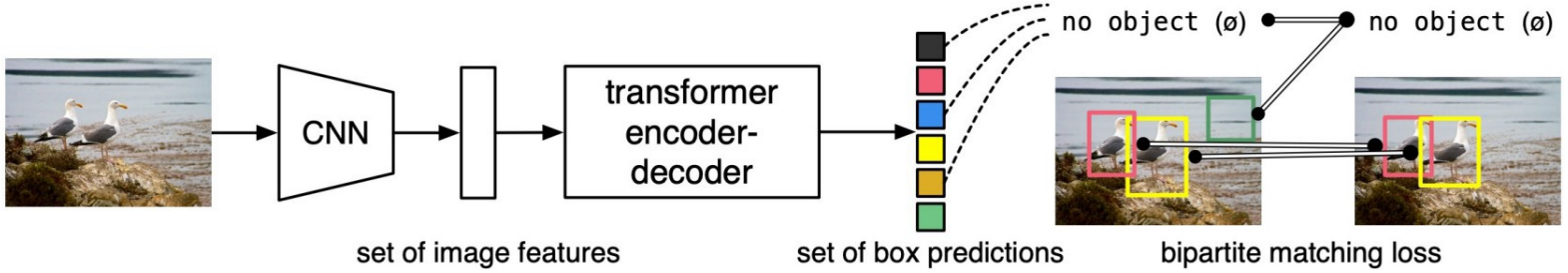
Li et al, "Improved Multiscale Vision Transformers for Classification and Detection", arXiv 2021

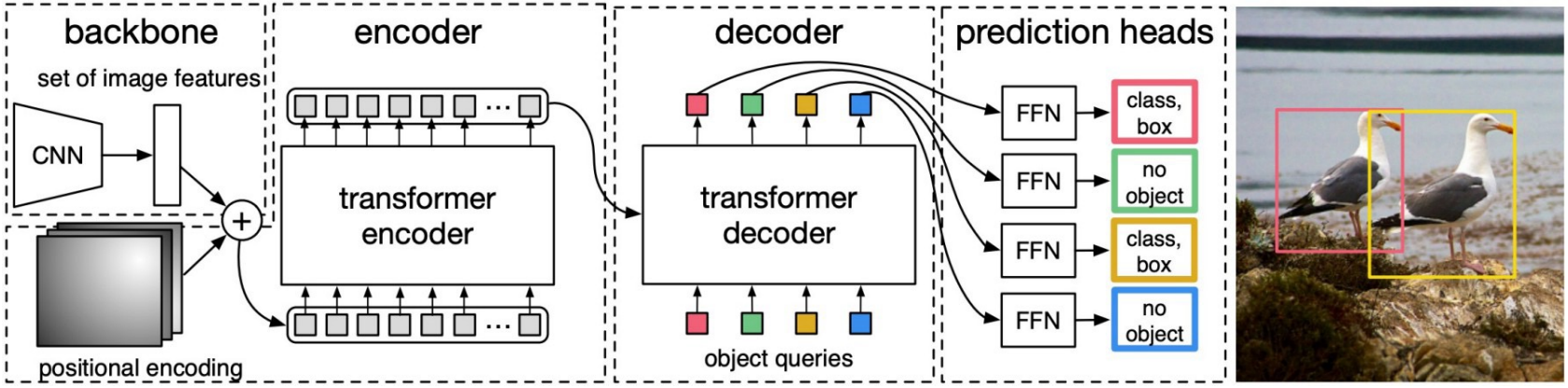
Object Detection with Transformers: DETR

- 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

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

Object Detection with Transformers: DETR





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

References

These slides have been adapted from

- Fei-Fei Li, Yunzhu Li & Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision
- Assaf Shocher, Shai Bagon, Meirav Galun & Tali Dekel, WAIC DL4CV Deep Learning for Computer Vision: Fundamentals and Applications
- Justin Johnson, UMich EECS 498.008/598.008: Deep Learning for Computer Vision