Advanced Computer Vision

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

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

LMH
Lady Margaret Hall

July 25, 2025

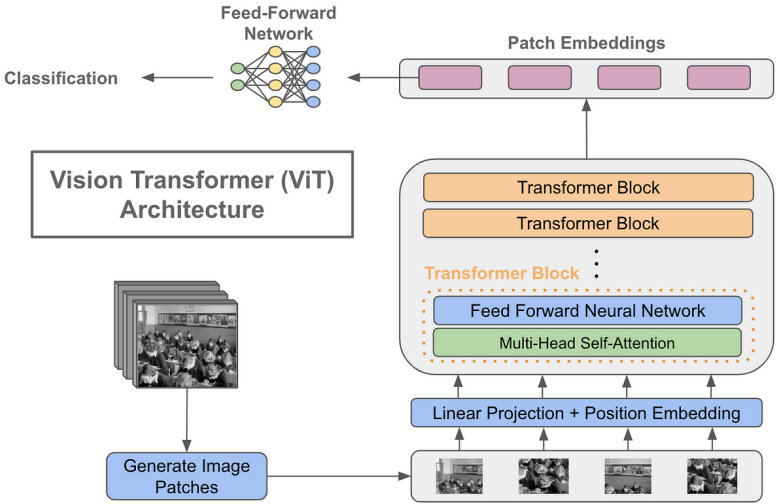

Feed-Forward Network
Classification
Patch Embeddings
Vision Transformer (ViT) Architecture
Transformer Block
Transformer Block
Transformer Block
Feed Forward Neural Network
Multi-Head Self-Attention
Linear Projection + Position Embedding
Generate Image Patches

Table of Contents

Table of Contents (cont.)

Motivation

Why this topic matters:

- Traditional CNNs struggle with sequential and long-range dependencies.
- RNNs were a start, but they come with their own baggage.
- Attention mechanisms and Transformers revolutionized NLP – now transforming vision tasks.
- Vision Transformers (ViTs) are outperforming CNNs on many benchmarks.
- *"Understanding the Transformer is a must for any modern deep learning practitioner."*

⇒ Learning Outcomes

By the end of this session, you will be able to:

- Explain the working and limitations of RNNs.
- Understand attention and self-attention mechanisms.
- Grasp the Transformer architecture.
- Apply Transformers to computer vision tasks.
- Compare CNNs and ViTs in terms of performance and design.
- Explore current applications and future directions.

Advanced Computer Vision: **Recurrent Neural Networks (RNN)**

Recurrent Neural Networks (RNN)

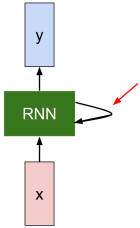
What are RNNs?

- Neural networks designed for sequence data.
- Maintain a hidden state h_t that carries information from previous time steps.
- Recurrent architecture: $h_t = f(h_{t-1}, x_t)$

Use Cases:

- Language modeling
- Time series prediction
- Sequence classification

Recurrent Neural Networks (RNN) (cont.)



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

$$h_t = f_W(h_{t-1}, x_t)$$

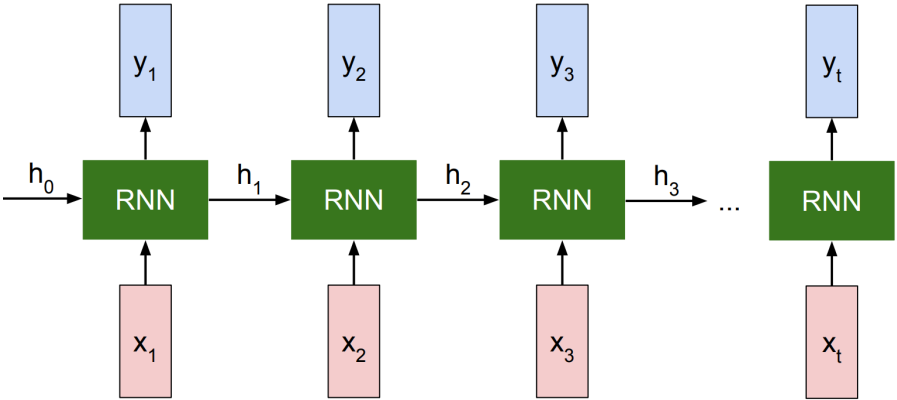
new state

old state

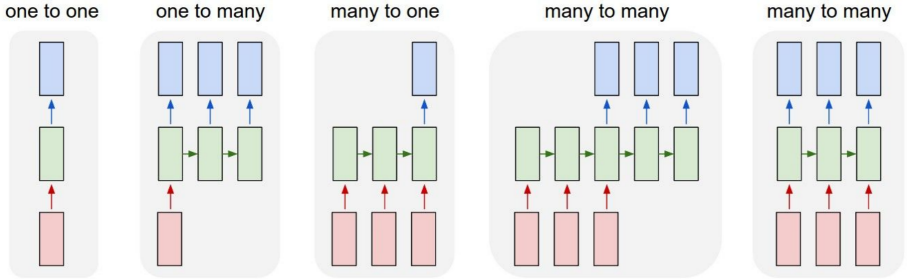
input vector at some time step

some function with parameters W

Recurrent Neural Networks (RNN) (cont.)



Recurrent Neural Networks (RNN) (cont.)



RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

Recurrent Neural Networks (RNN) (cont.)

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Limitations of RNNs:

- Vanishing/exploding gradients in long sequences.
- Sequential computation — slow to train.
- Difficulty in capturing long-term dependencies.
- Cannot parallelize easily due to recursive nature.

Advanced Computer Vision: **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

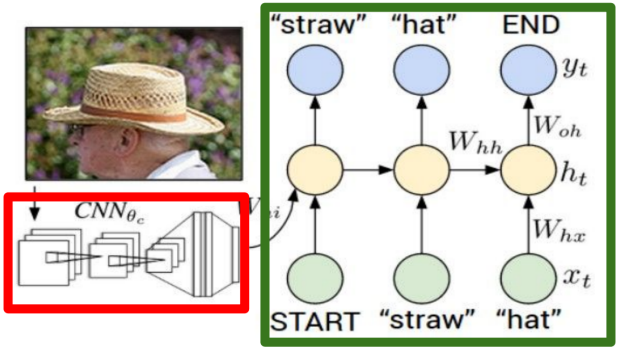
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Image Captioning (cont.)

- The task is to generate a natural language description of the content of an image
- It combines techniques from computer vision and natural language processing
- The process typically involves:
  - Extracting features from the image using a convolutional neural network (CNN)
  - Using these features to generate a sequence of words that describe the image, often with a recurrent neural network (RNN) or transformer model

Recurrent Neural Network

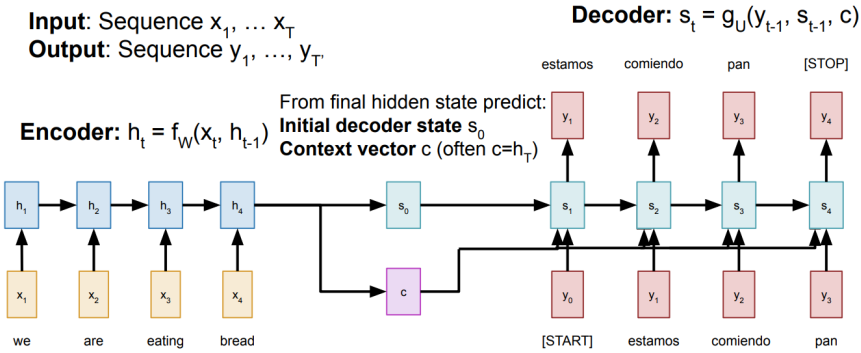


Convolutional Neural Network

Advanced Computer Vision:

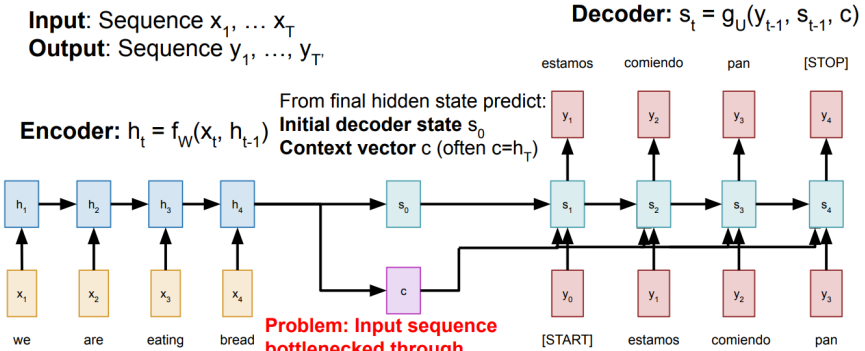
Sequence-to-Sequence with RNNs

Sequence-to-Sequence with RNNs



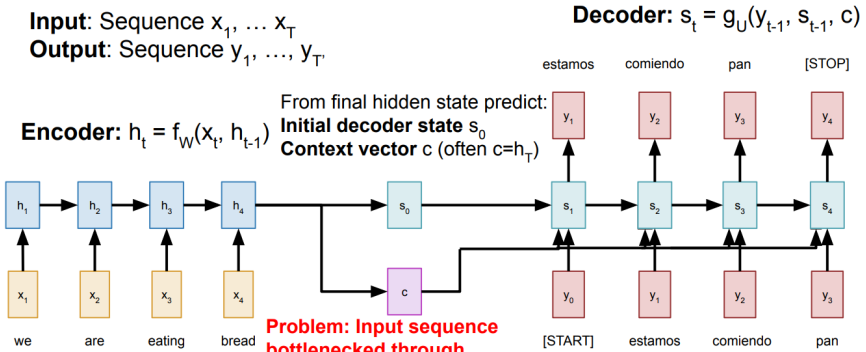
Sutskever et al, "Sequence to sequence learning with neural networks

Sequence-to-Sequence with RNNs (cont.)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs (cont.)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

- **Key idea:** "Don't process everything — just focus on the most relevant parts."
- Attention helps models decide where to look in a sequence.

Equation (simplified):

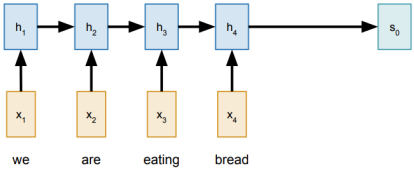
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- Q = Queries
- K = Keys
- V = Values

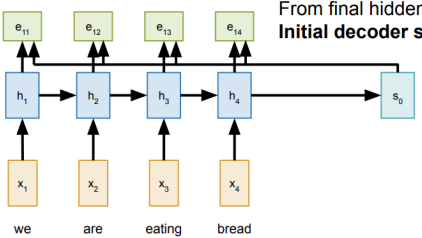
Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$

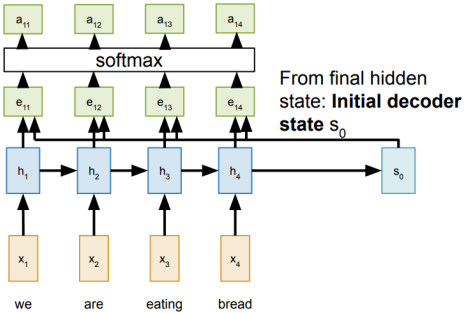
From final hidden state:
Initial decoder state s_0



Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)



From final hidden state:
Initial decoder state s_0



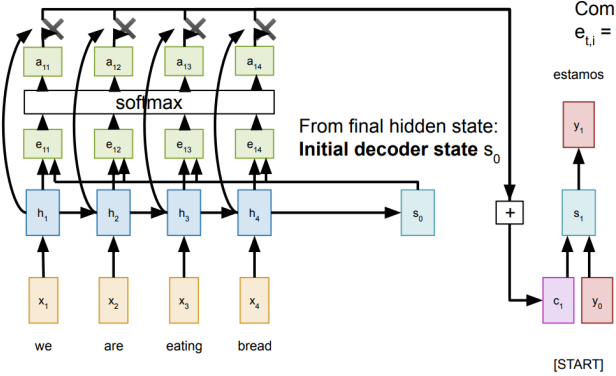
From final hidden state: **Initial decoder state** s_0

Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$



Compute (scalar) **alignment scores**

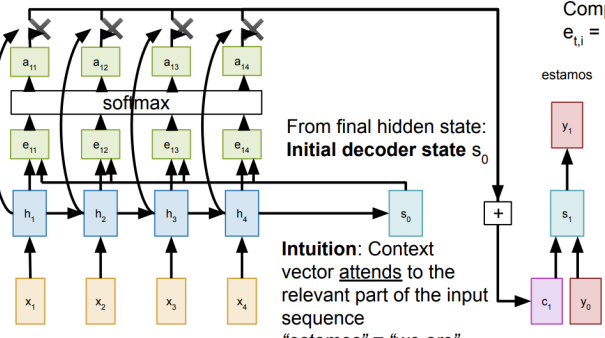
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$



Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

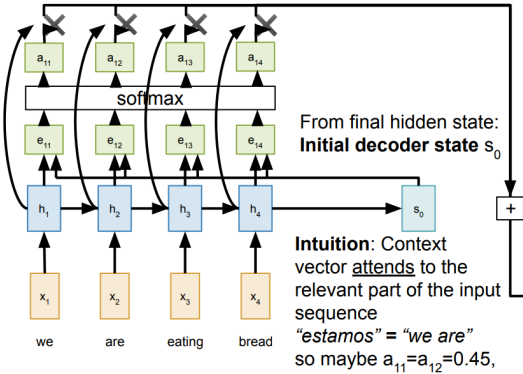
Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$



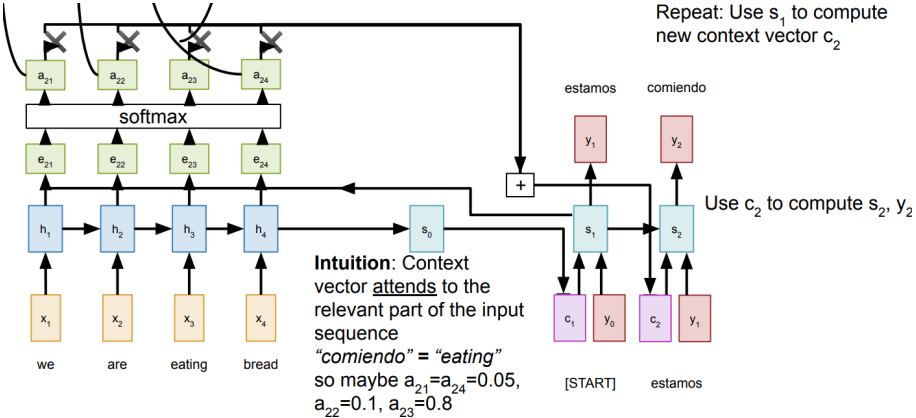
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

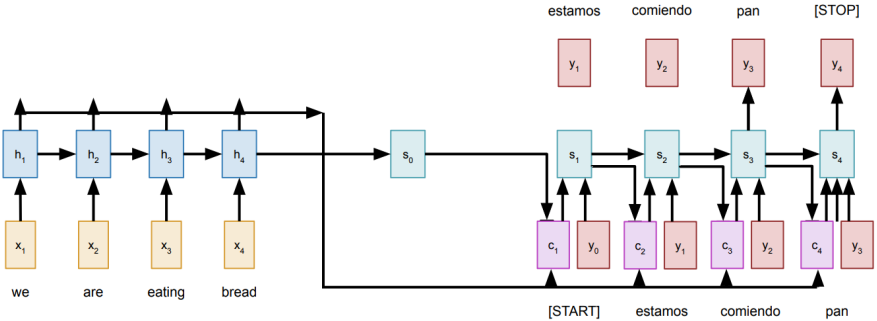
Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

This is all differentiable! No supervision on attention weights – backprop through everything


Repeat: Use s_1 to compute new context vector c_2
estamos
comiendo
softmax
Use c_2 to compute s_2, y_2
Intuition: Context vector attends to the relevant part of the input sequence
"comiendo" = "eating"
so maybe a_{21}=a_{24}=0.05, a_{22}=0.1, a_{23}=0.8
we
are
eating
bread
[START]
estamos

- Use a different context vector at each timestep of the decoder.
- The input sequence is not bottlenecked through a single vector.
- At each timestep of the decoder, the context vector "looks at" different parts of the input sequence.



Example: English to French translation

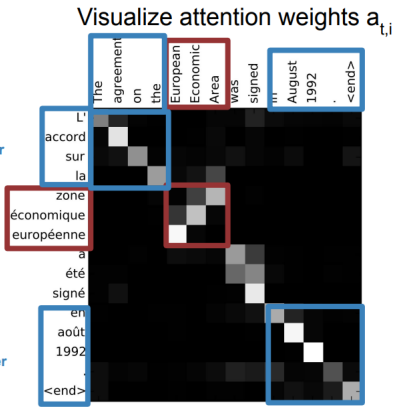
Input: "The agreement on the European Economic Area was signed in August 1992."

Output: "L'accord sur la zone économique européenne a été signé en août 1992."

Diagonal attention means words correspond in order

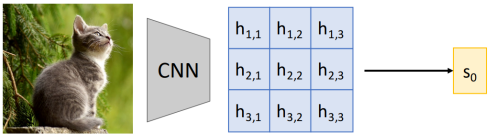
Attention figures out different word orders

Diagonal attention means words correspond in order



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Advanced Computer Vision: **Image Captioning with RNNs and Attention**



Use a CNN to compute a grid of features for an image

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

Alignment scores

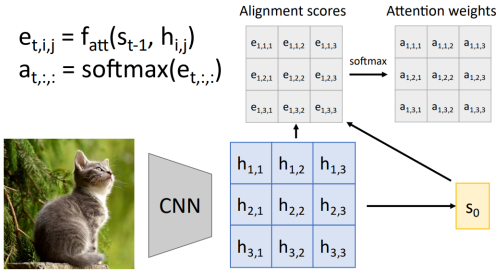
$e_{1,1,1}$	$e_{1,1,2}$	$e_{1,1,3}$
$e_{1,2,1}$	$e_{1,2,2}$	$e_{1,2,3}$
$e_{1,3,1}$	$e_{1,3,2}$	$e_{1,3,3}$

CNN

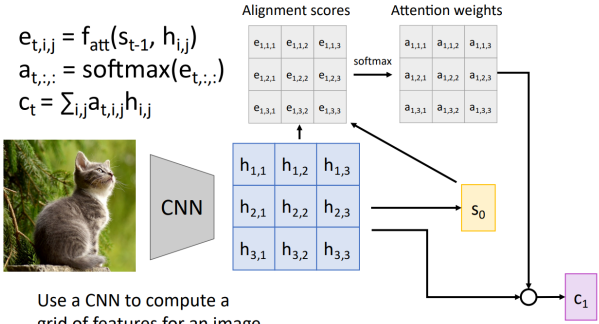
$h_{1,1}$	$h_{1,2}$	$h_{1,3}$
$h_{2,1}$	$h_{2,2}$	$h_{2,3}$
$h_{3,1}$	$h_{3,2}$	$h_{3,3}$

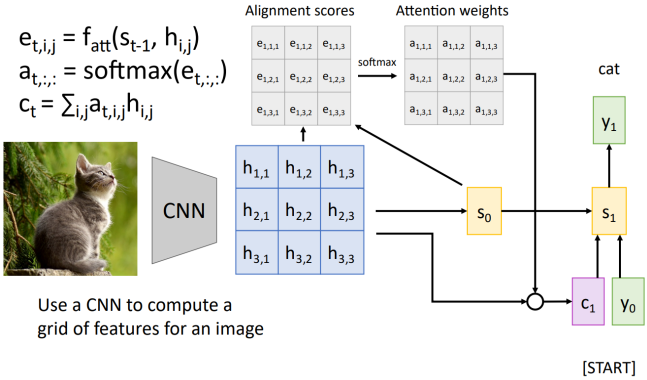
s_0

Use a CNN to compute a grid of features for an image



Use a CNN to compute a grid of features for an image


$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$
Alignment scores
Attention weights
softmax
CNN
s_0
c_1
Use a CNN to compute a grid of features for an image


$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$
Alignment scores
Attention weights
softmax
CNN
cat
s_0
s_1
y_1
c_1
y_0
[START]
Use a CNN to compute a grid of features for an image

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

CNN

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

s_0 s_1

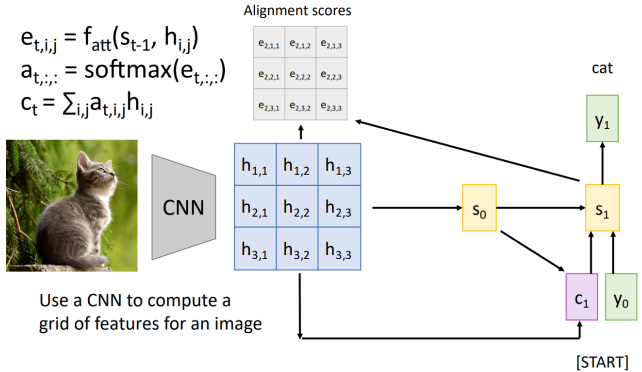
c_1 y_0

y_1

cat

[START]

Use a CNN to compute a grid of features for an image


$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$
Alignment scores
CNN
Use a CNN to compute a grid of features for an image
cat
[START]

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

Use a CNN to compute a grid of features for an image

CNN

Alignment scores

$e_{2,1,1}$ $e_{2,1,2}$ $e_{2,1,3}$
$e_{2,2,1}$ $e_{2,2,2}$ $e_{2,2,3}$
$e_{2,3,1}$ $e_{2,3,2}$ $e_{2,3,3}$

softmax

Attention weights

$a_{2,1,1}$ $a_{2,1,2}$ $a_{2,1,3}$
$a_{2,2,1}$ $a_{2,2,2}$ $a_{2,2,3}$
$a_{2,3,1}$ $a_{2,3,2}$ $a_{2,3,3}$

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

s_0 s_1 s_2

y_1 y_2

cat sitting

c_1 y_0 c_2 y_1

[START] cat

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

Each timestep of decoder uses a different context vector that looks at different parts of the input image

CNN

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

Use a CNN to compute a grid of features for an image

s_0 s_1 s_2 s_3 s_4

y_1 y_2 y_3 y_4

c_1 y_0 c_2 y_1 c_3 y_2 c_4 y_3

cat sitting outside [STOP]

[START] cat sitting outside

A dog is standing on a hardwood floor.

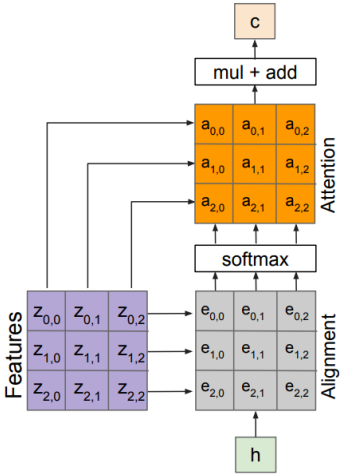
A stop sign is on a road with a mountain in the background.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Advanced Computer Vision: **General Attention Layer**

Attention We Just Saw in Image Captioning

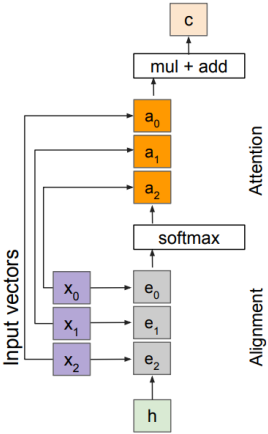


Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
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



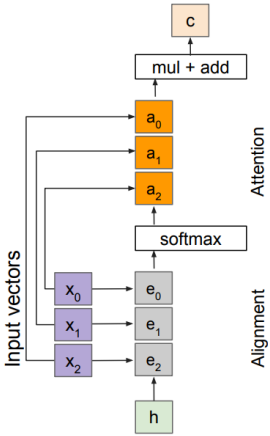
Outputs:
context vector: **c** (shape: D)

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)

- The attention operation is permutation invariant.
- It does not depend on the ordering of the features.
- Stretch $H \times W = N$ into N vectors.

General Attention Layer



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

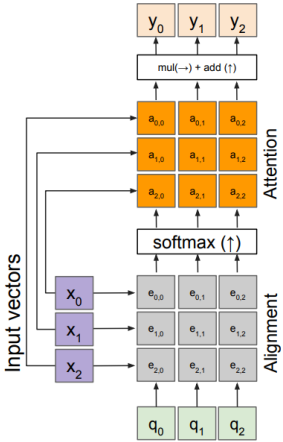
Operations:
Alignment: $e_i = h \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

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

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

- Larger dimensions mean more terms in the dot product sum.
- Therefore, the variance of the logits is higher. Large-magnitude vectors will produce much higher logits.
- As a result, 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 the effect of large-magnitude vectors.

General Attention Layer



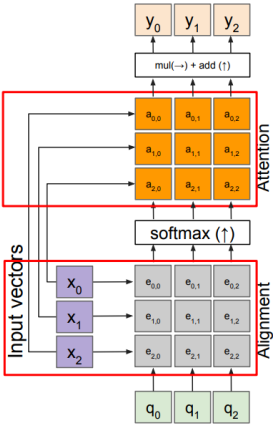
Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
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



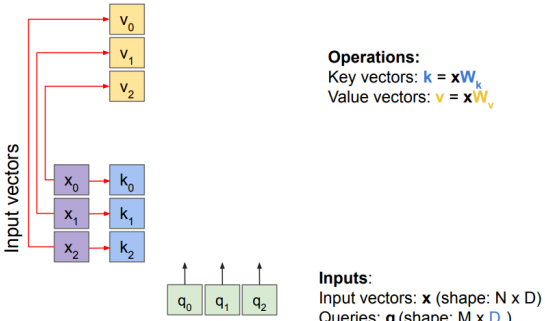
Outputs:
context vectors: **y** (shape: D)

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)

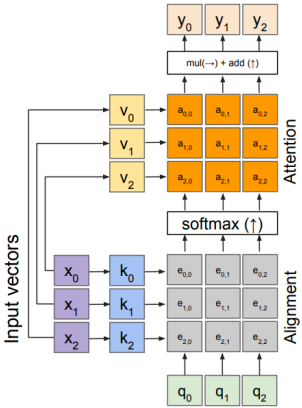
- Notice that the input vectors are used for both the alignment and the attention calculations.

General Attention Layer



- Notice that the input vectors are used for both the alignment and the attention calculations.
- We can add more expressivity to the layer by adding a different fully connected (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** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} v_i$

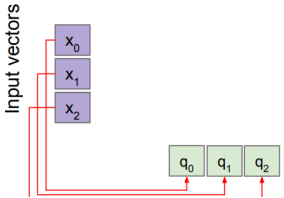
Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

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

Advanced Computer Vision: **Self Attention Layer**

- **Self-Attention:** Computes attention within a single sequence.
- **How it works:**
  - Each token attends to every other token.
  - Captures global context, not just local dependencies.

Self Attention Layer

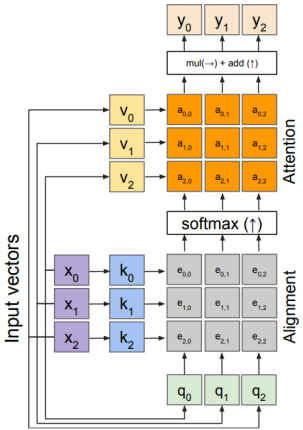


Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_i \cdot k_j / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

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

- Recall that the query vector is a function of the input vectors.
- We can calculate the query vectors from the input vectors, thus defining a "self-attention" layer.
- There are no input query vectors anymore.
- Instead, query vectors are calculated using a fully connected (FC) layer.

Self Attention Layer



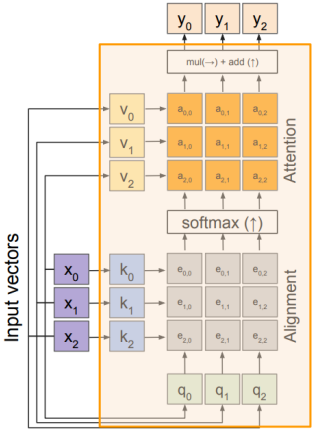
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
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 is a function of the input vectors.
- We can calculate the query vectors from the input vectors, thus defining a "self-attention" layer.
- There are no input query vectors anymore.
- Instead, query vectors are calculated using a fully connected (FC) layer.

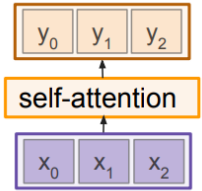
Self Attention Layer



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

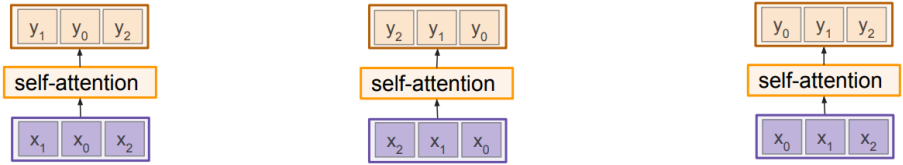
Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_i \cdot k_j / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

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



Advanced Computer Vision: **Permutation Invariance**

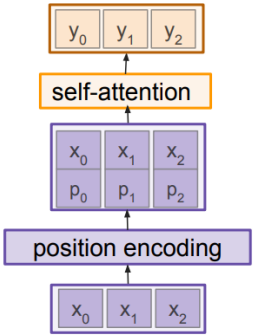
Permutation Invariance



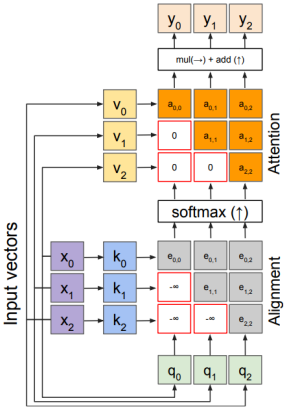
- Self-Attention Layer is Permutation equivariant
- It doesn't care about the orders of the inputs!
- **Problem:** How can we encode ordered sequences like language or spatially ordered image features?

Positional encoding

- Concatenate/add special positional encoding p_j to each input vector x_j
- We use a function pos: $N \rightarrow \mathbb{R}^D$ to process the position j of the vector into a d-dimensional vector
- So, $p_j = pos(j)$



Masked self-attention layer



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

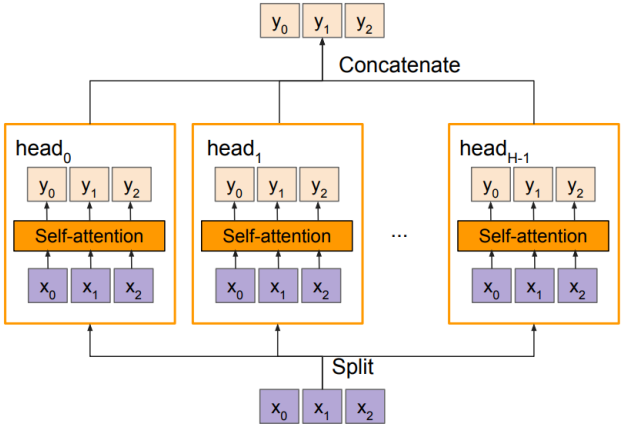
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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

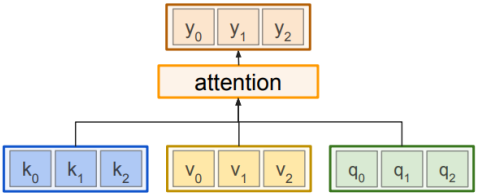
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to $-\infty$

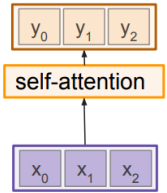
Multi-head self-attention layer

- Multiple self-attention heads in parallel

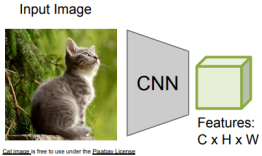


General attention versus self-attention

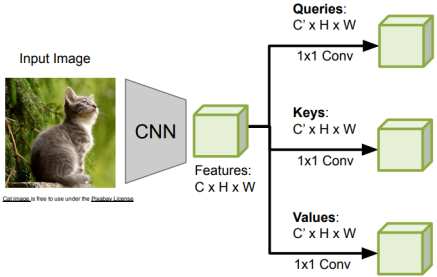




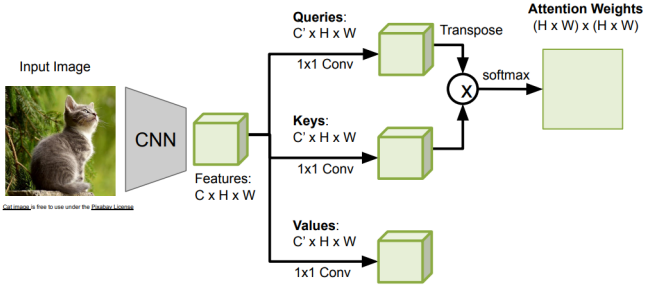
Example: CNN with Self-Attention



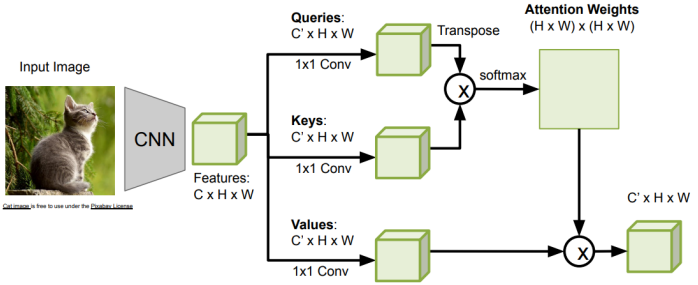
Example: CNN with Self-Attention



Example: CNN with Self-Attention

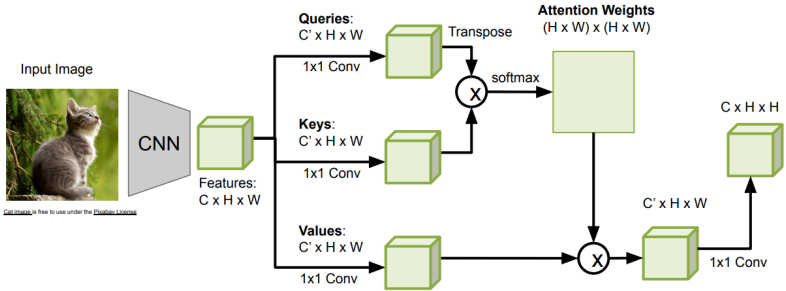


Example: CNN with Self-Attention



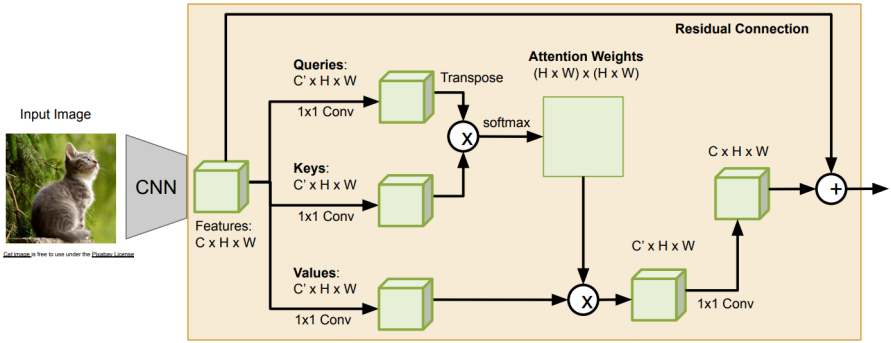
Cat image is free to use under the Pixabay License

Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

Self-Attention Module

Advanced Computer Vision: **Transformer Overview**

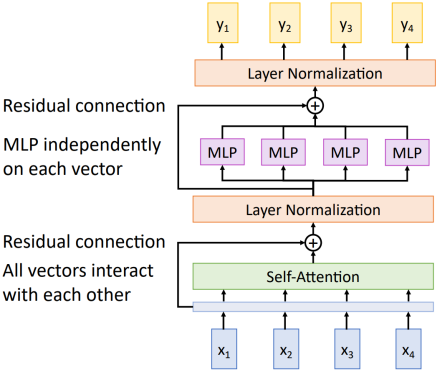
Attention is all you need

Vaswani et al, NeurIPS 2017

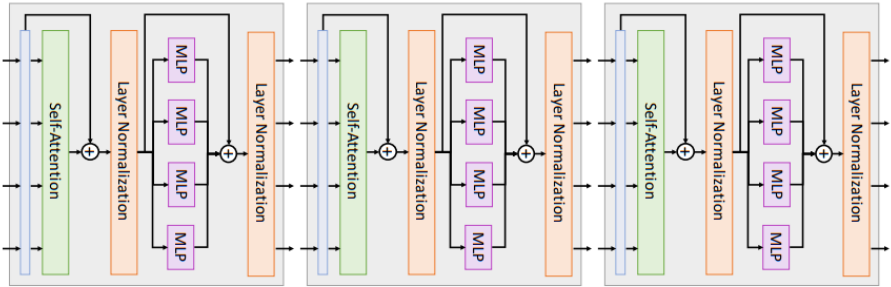
Transformer Block

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

- Self-attention is the only interaction between vectors!
- Layer normalization and MLP operate independently on each vector.
- Highly scalable and highly parallelizable.



- A Transformer consists of a sequence of transformer blocks.
- Vaswani et al. used 12 blocks, $D_Q = 512$, and 6 attention heads.



Advanced Computer Vision: **Vision Transformers**

Vision Transformer (ViT) Architecture

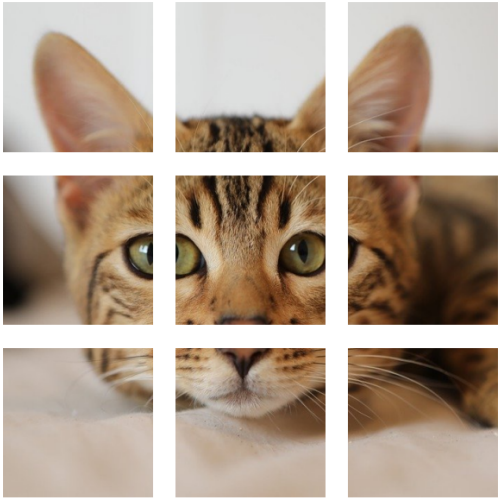
Key idea:

- Split image into fixed-size patches (e.g., 16×16)
- Flatten and project patches into embeddings
- Add positional encodings to embeddings
- Apply standard Transformer encoder

Architecture:

- Patch embedding
- Class token
- Transformer blocks
- MLP Head

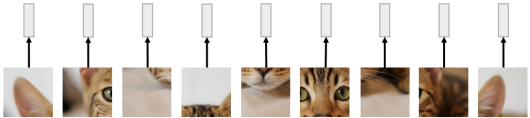
Source: Dosovitskiy et al., "An Image is Worth 16x16 Words", 2020

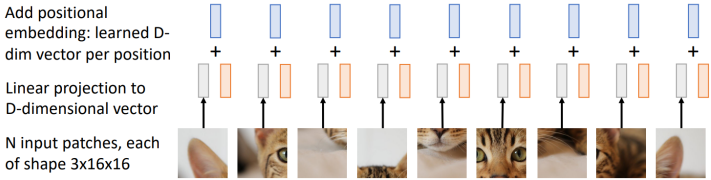


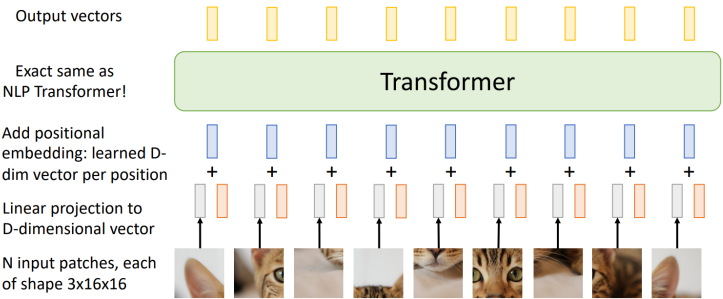
N input patches, each of shape 3x16x16

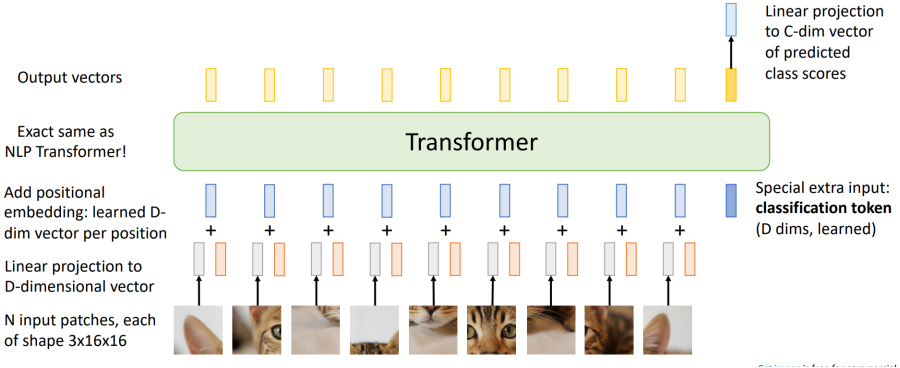
Linear projection to D-dimensional vector

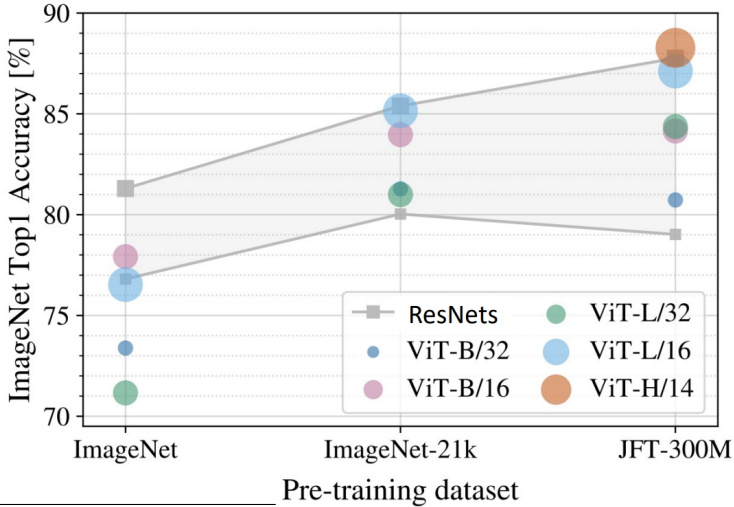
N input patches, each of shape 3x16x16




Add positional embedding: learned D-dim vector per position
Linear projection to D-dimensional vector
N input patches, each of shape 3x16x16


Output vectors
Exact same as NLP Transformer!
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
Output vectors
Exact same as NLP Transformer!
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

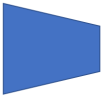


[0]Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Advanced Computer Vision: **Improving ViT: Distillation**

Improving ViT: Distillation

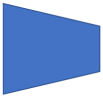
Step 1: Train a **teacher CNN** on ImageNet



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

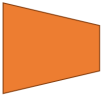
→ Cross Entropy Loss ← GT label: Cat

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

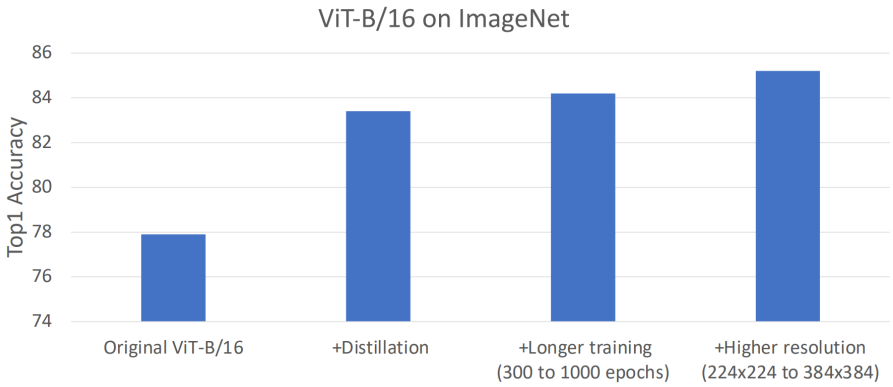
→ KL Divergence Loss



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

Advanced Computer Vision: **CNN vs ViT**

CNN		ViT
✓	1. Maintain 2D structure logic	✓
✓	2. Shift equivariant	✗
✓	3. Consider only local correlations	✗
✓	4. Hierarchically growing field of view	✗
✓	5. Hierarchically progressing complexity	✓
✓	6. Reasonable amount of params	✓
✗	7. Global representation	✓

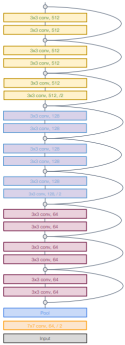
CNN vs ViT

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

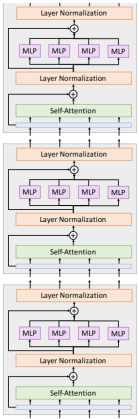


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

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

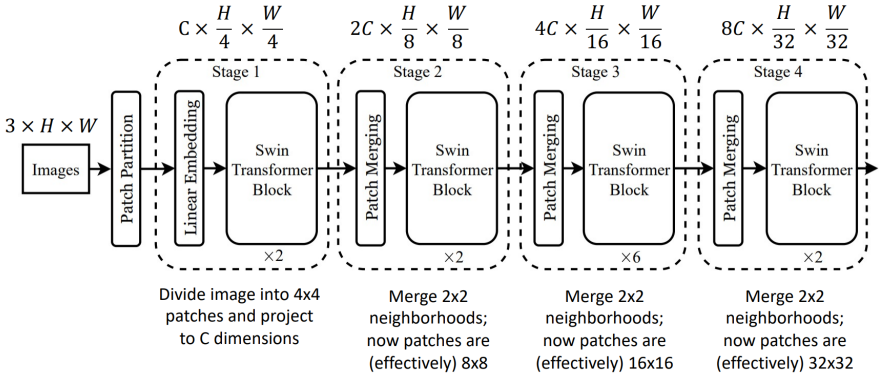
Input:
3 x 224 x 224

CNNs vs Vision Transformers: Key Differences

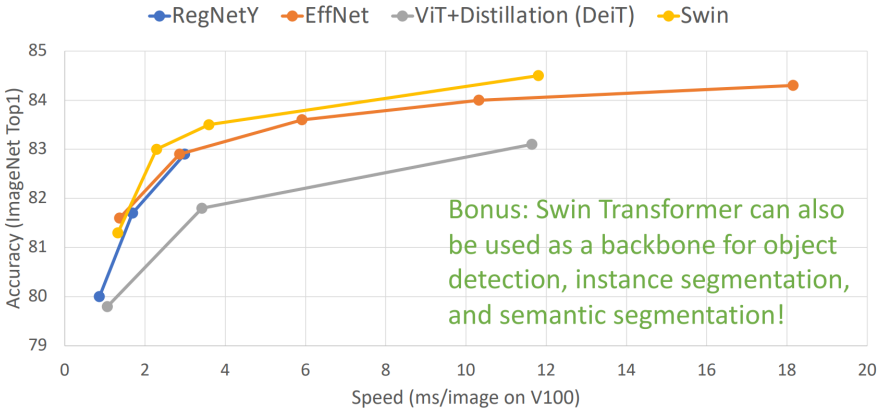
Feature	CNNs	Vision Transformers
Locality	Strong inductive bias	Weak (learns from data)
Parameter sharing	Yes	Yes
Global context	Limited	Native support
Training data	Efficient on small data	Needs large data or pretraining
Flexibility	Less modular	More modular

Advanced Computer Vision: **Hierarchical ViT: Swin Transformer**

Hierarchical ViT: Swin Transformer



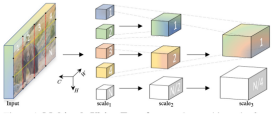
Hierarchical ViT: Swin Transformer



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

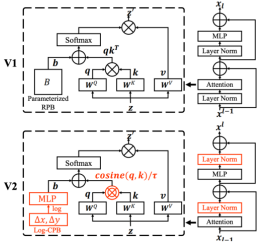
Advanced Computer Vision: **Other Hierarchical Vision Transformers**

MViT



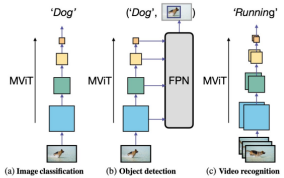
Fan et al, "Multiscale Vision Transformers", ICCV 2021

Swin-V2



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

Improved MViT

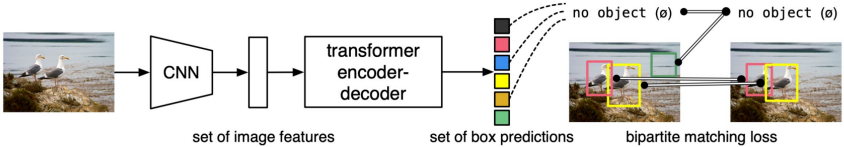


Li et al, "Improved Multiscale Vision Transformers for Classification and Detection", arXiv 2021

- Simple object detection pipeline: directly outputs a set of boxes from a Transformer
- No anchors, no regression of box transforms
- Matches predicted boxes to ground truth boxes with bipartite matching; trains 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

Advanced Computer Vision: **Limitations and Future Directions**

Limitations of Transformers in Computer Vision

- **Data hungry**: Performance drops significantly on small datasets.
- **Computationally expensive**: Quadratic complexity of self-attention.
- **Lacks built-in locality bias**: Unlike CNNs; some variants (e.g., Swin Transformer) address this.
- **Requires large-scale pretraining**: Needs datasets like JFT-300M, ImageNet-22K for strong performance.

Future Directions

- **Efficient transformers**: Linformer, Performer, Longformer.
- **Hybrid models**: Combining CNNs and Transformers (e.g., CoaT, CvT).
- **Low-resource ViTs**: Data-efficient training (e.g., DeiT).
- **Vision-language models**: CLIP, Flamingo, GPT-4V.
- **Hardware acceleration**: Edge ViTs, quantized ViTs for deployment.

Advanced Computer Vision: **References**

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. *Advances in Neural Information Processing Systems*, 30. https://arxiv.org/abs/1706.03762

[2] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations*. https://arxiv.org/abs/2010.11929

[3] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers distillation through attention. *International Conference on Machine Learning*. https://arxiv.org/abs/2012.12877

References (cont.)

[4] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *CVPR 2021*. https://arxiv.org/abs/2103.14030

[5] Chen, J., Chen, J., Chao, H., & Yang, M. (2021). Pre-Trained Image Processing Transformer. *CVPR 2021*. https://arxiv.org/abs/2012.00364

[6] Lu, Y., Dosovitskiy, A., Houlsby, N., & Beyer, L. (2021). Vision Transformer: A Survey. *arXiv preprint arXiv:2111.06091*. https://arxiv.org/abs/2111.06091

[7] Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., & Luo, P. (2021). SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. *NeurIPS 2021*. https://arxiv.org/abs/2105.15203

References (cont.)

[8] Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., ... & Fu, Y. (2021). Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers (SETR). *CVPR 2021*. https://arxiv.org/abs/2012.15840

[9] Strudel, R., Garcia, R., Laptev, I., & Schmid, C. (2021). Segmenter: Transformer for Semantic Segmentation. *ICCV 2021*. https://arxiv.org/abs/2105.05633

[10] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision (CLIP). *ICML 2021*. https://arxiv.org/abs/2103.00020

References (cont.)

[11] Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., ... & Simonyan, K. (2022). Flamingo: a Visual Language Model for Few-Shot Learning. *arXiv preprint arXiv:2204.14198*. https://arxiv.org/abs/2204.14198

[12] Oquab, M., Darcet, T., Moutakanni, T., Vo, D. H., Szafraniec, M., Khalidov, V., ... & Jegou, H. (2023). DINOv2: Learning Robust Visual Features without Supervision. *arXiv preprint arXiv:2304.07193*. https://arxiv.org/abs/2304.07193

[13] Fei-Fei Li, Y., Li, Y., & Gao, R. Stanford CS231n: Deep Learning for Computer Vision. http://cs231n.stanford.edu/

[14] Shocher, A., Bagon, S., Galun, M., & Dekel, T. WAIC DL4CV: Deep Learning for Computer Vision: Fundamentals and Applications. https://www.cs.tau.ac.il/~assafshocher/WAIC-DL4CV/

References (cont.)

[15] Johnson, J. UMich EECS 498.008/598.008: Deep Learning for Computer Vision. https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2023/

[16] Hugging Face. Vision Transformer Playground. https://huggingface.co/spaces/akhaliq/ViT

[17] Stanford CS231n. Lecture Slides: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/slides/

[18] Justin Johnson. UMich EECS 498.008/598.008: Deep Learning for Computer Vision Slides. https://web.eecs.umich.edu/~justincj/teaching/eecs498/slides.html

[19] PyTorch. PyTorch Tutorials: Vision. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.