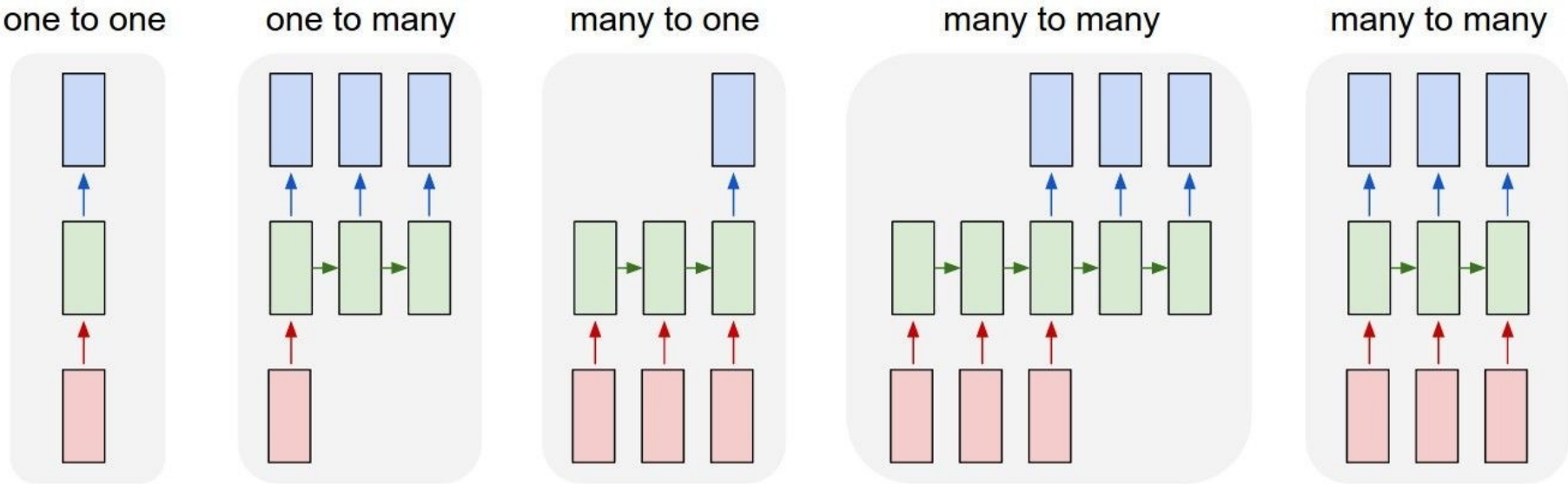
Introduction to Computer Vision

Lecture 12 –
Attention and Transformers

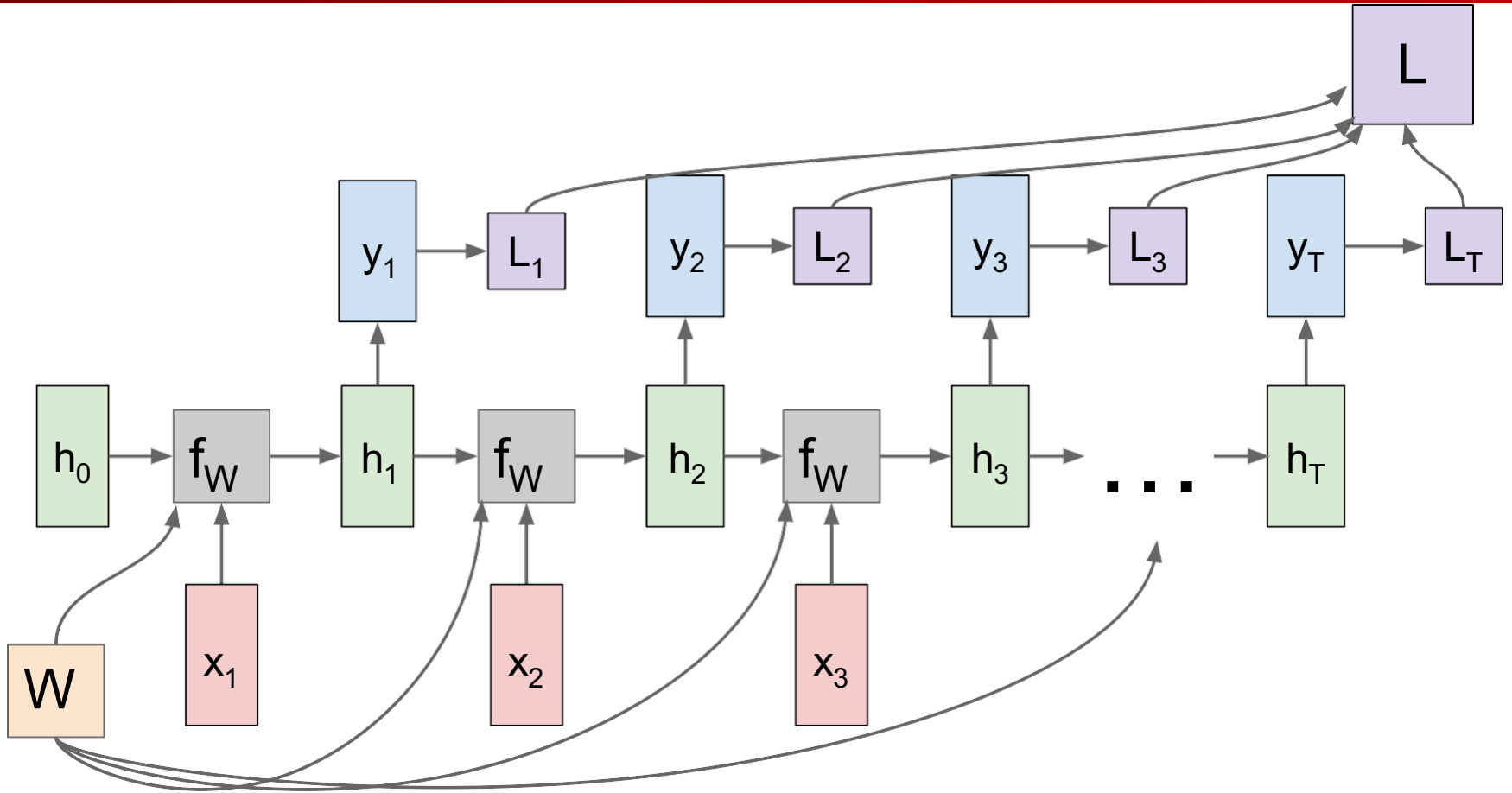
Zhizheng Zhang

2024-05-15

Last Time: Recurrent Neural Networks



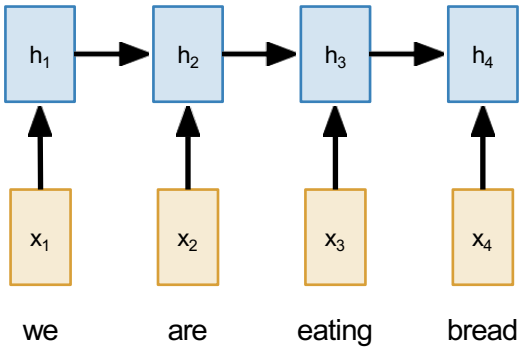
Last Time: Variable length computation graph with shared weights



Sequence to Sequence with RNNs

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

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



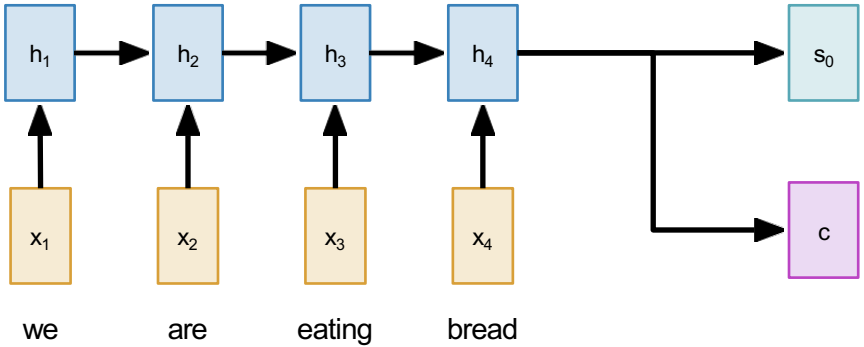
Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

Sequence to Sequence with RNNs

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 predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



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

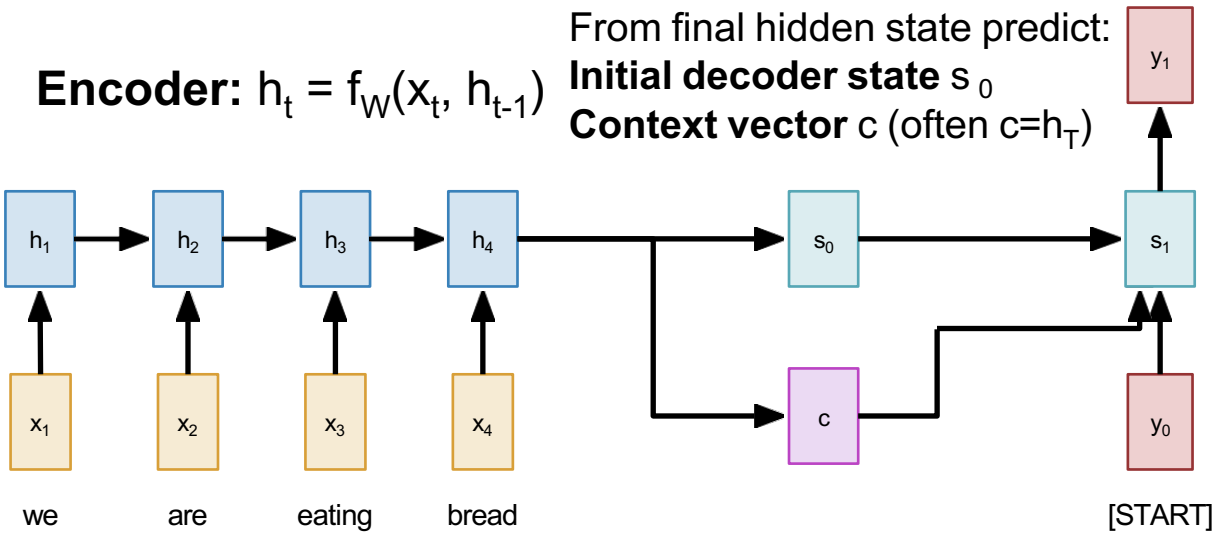
Sequence to Sequence with RNNs

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

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



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

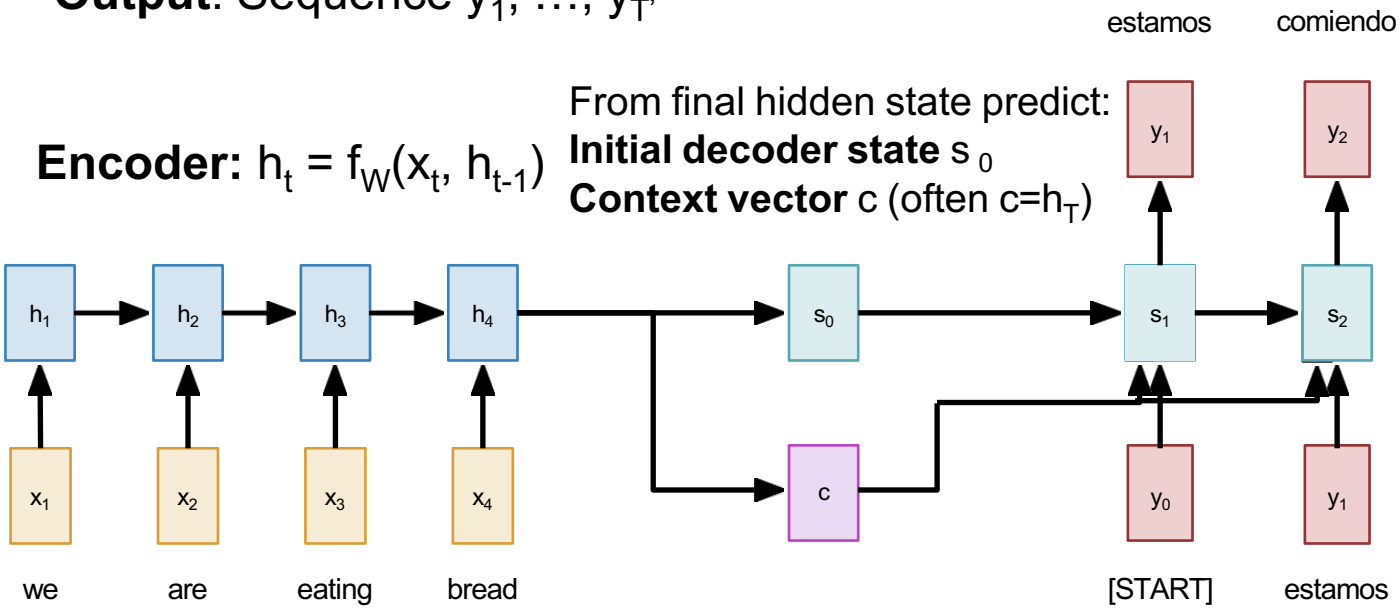
Sequence to Sequence with RNNs

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

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



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

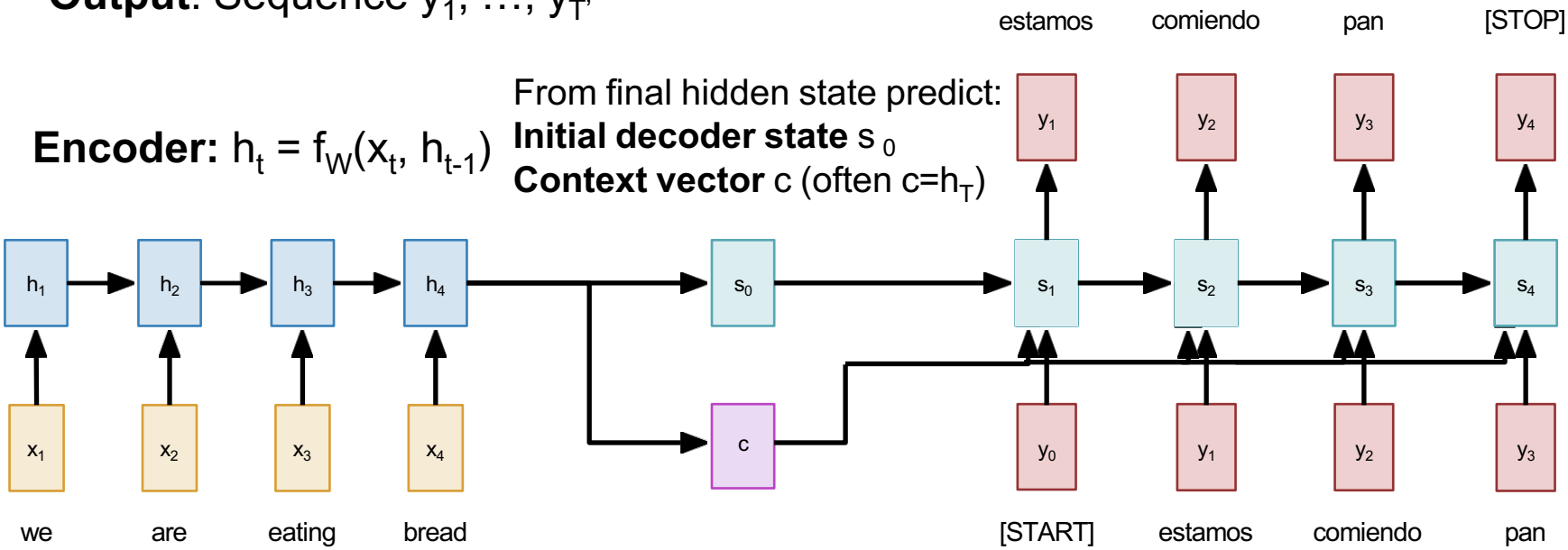
Sequence to Sequence with RNNs

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

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



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

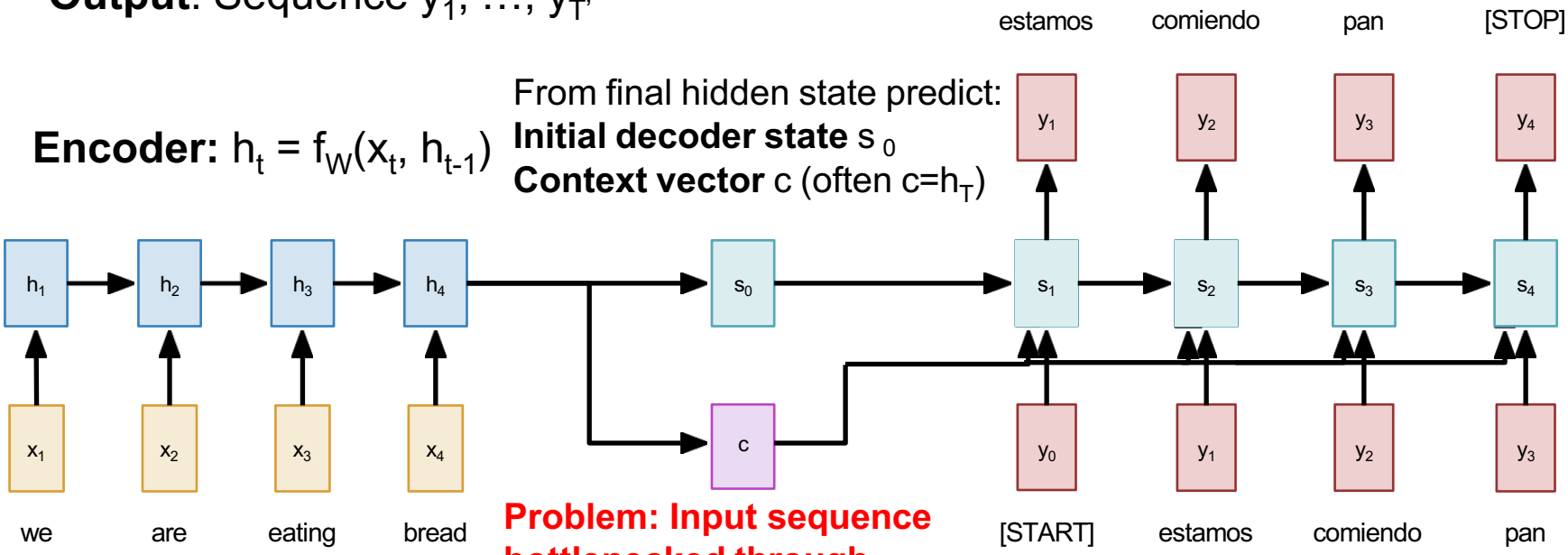
Sequence to Sequence with RNNs

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

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

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

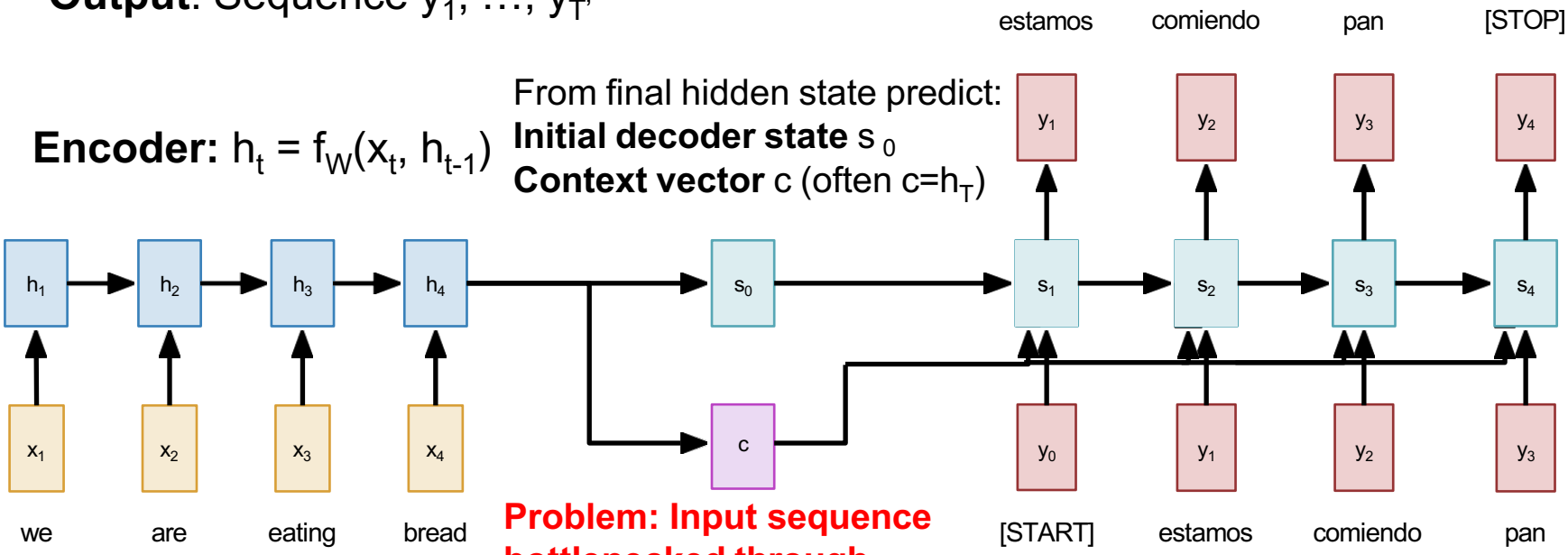
Sequence to Sequence with RNNs

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

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

Idea: use new context vector at each step of decoder!

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

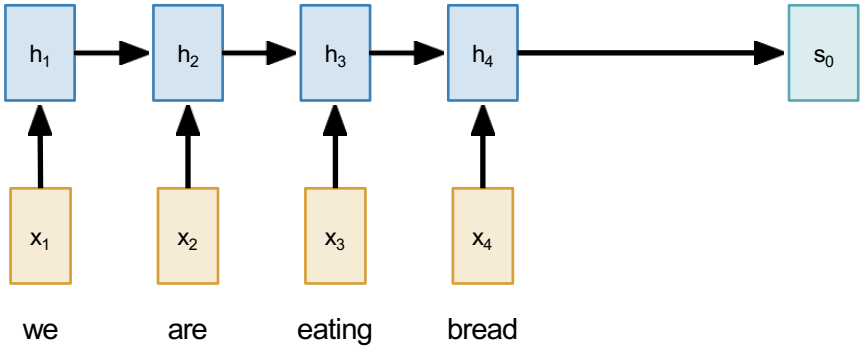
Attention

Sequence to Sequence with RNNs and Attention

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

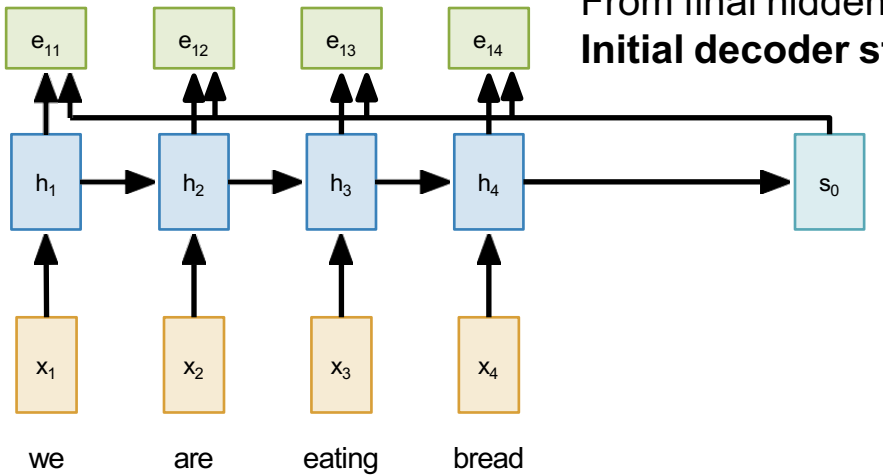


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

Compute (scalar) **alignment scores**

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

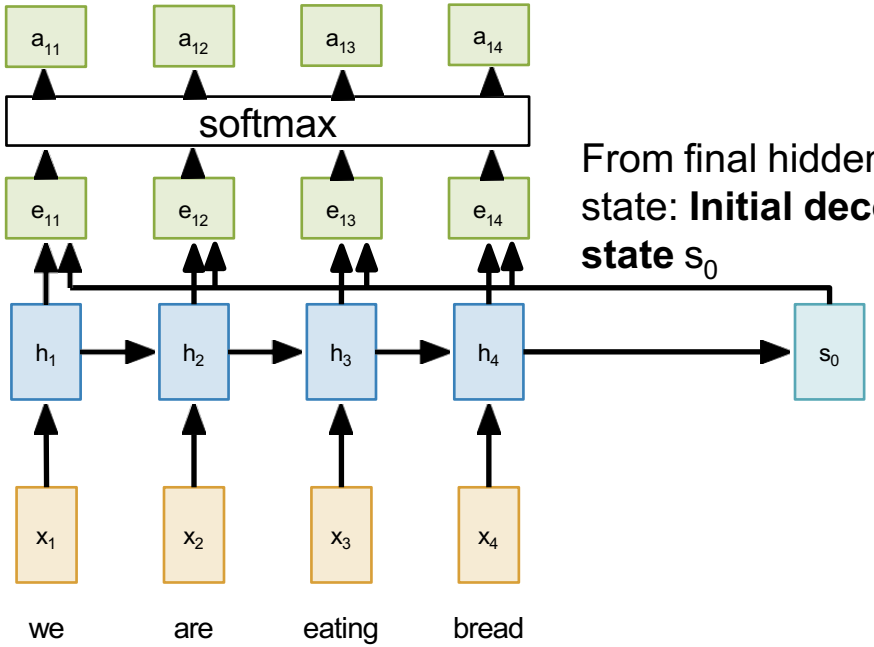


From final hidden state:

Initial decoder state s_0

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

Sequence to Sequence with RNNs and Attention



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

Compute (scalar) **alignment scores**

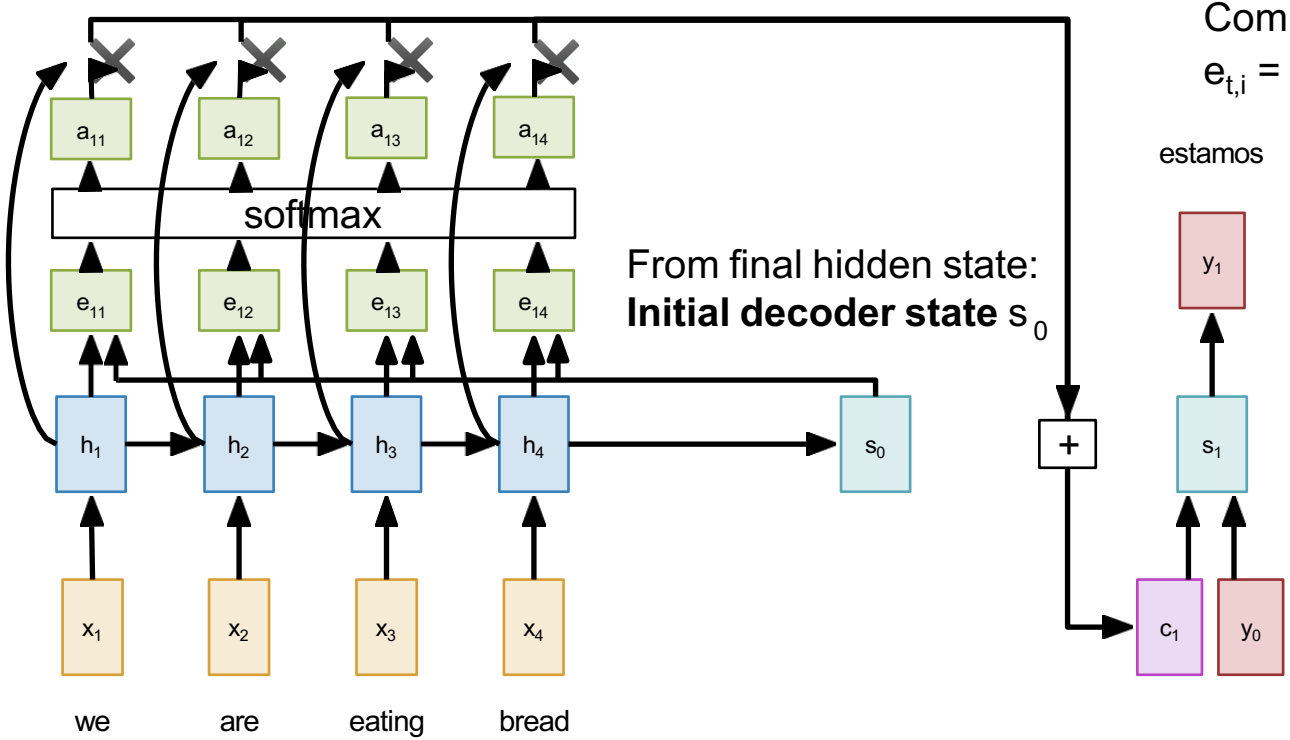
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

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

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

Sequence to Sequence with RNNs and Attention



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**

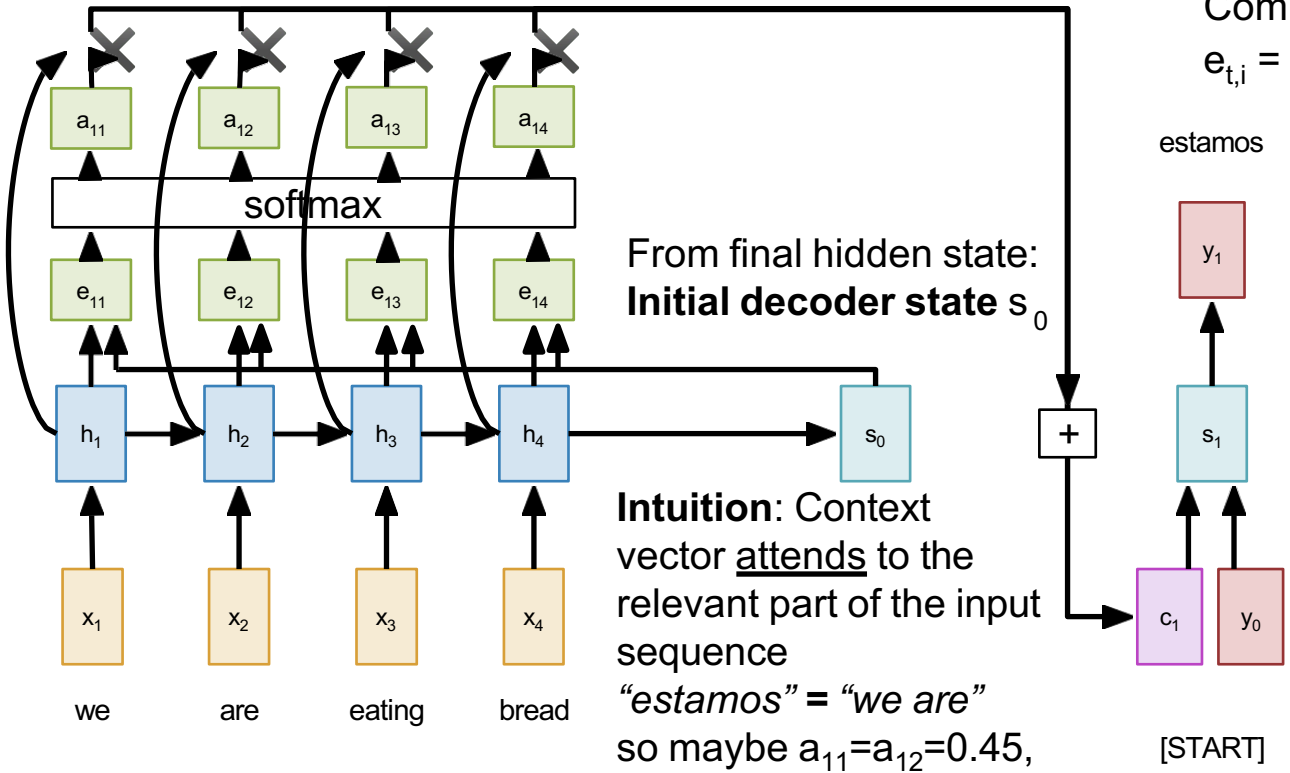
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

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

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

Sequence to Sequence with RNNs and Attention



From final hidden state:
Initial decoder state s_0

Intuition: Context vector attends to the relevant part of the input sequence
"estamos" **=** *"we are"*
so maybe a_{11}=a_{12}=0.45, a_{13}=a_{14}=0.05

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

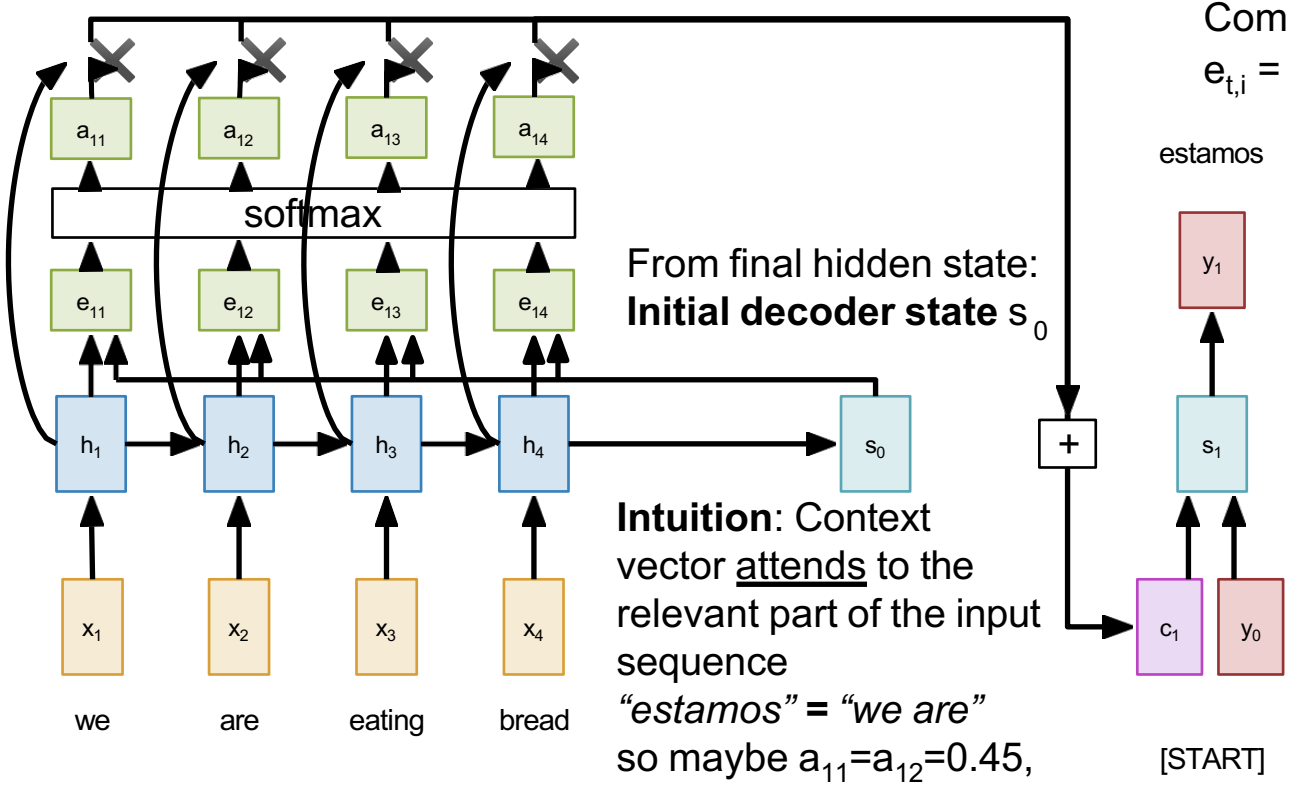
Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

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

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

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

Sequence to Sequence with RNNs and Attention



From final hidden state:
Initial decoder state s_0

Intuition: Context vector attends to the relevant part of the input sequence
"estamos" **=** *"we are"*
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

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

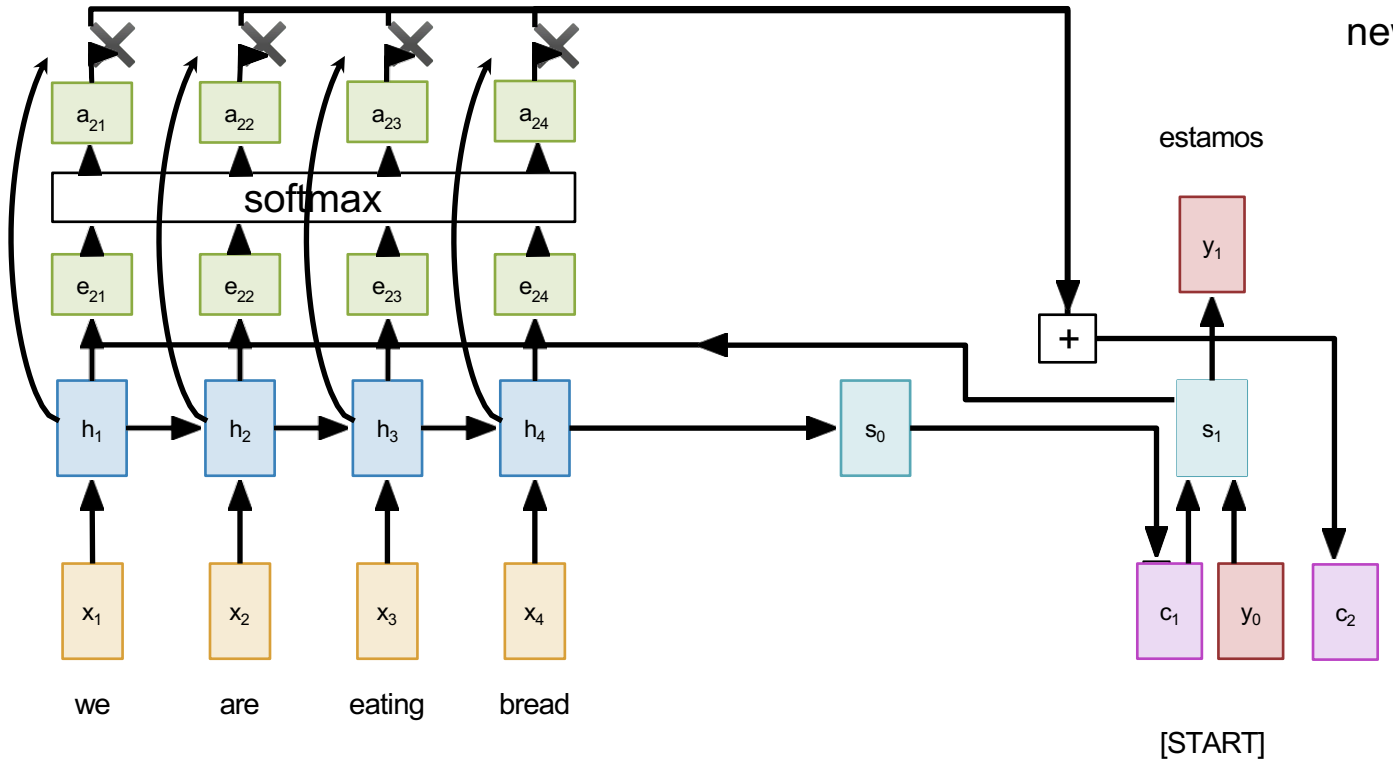
Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

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

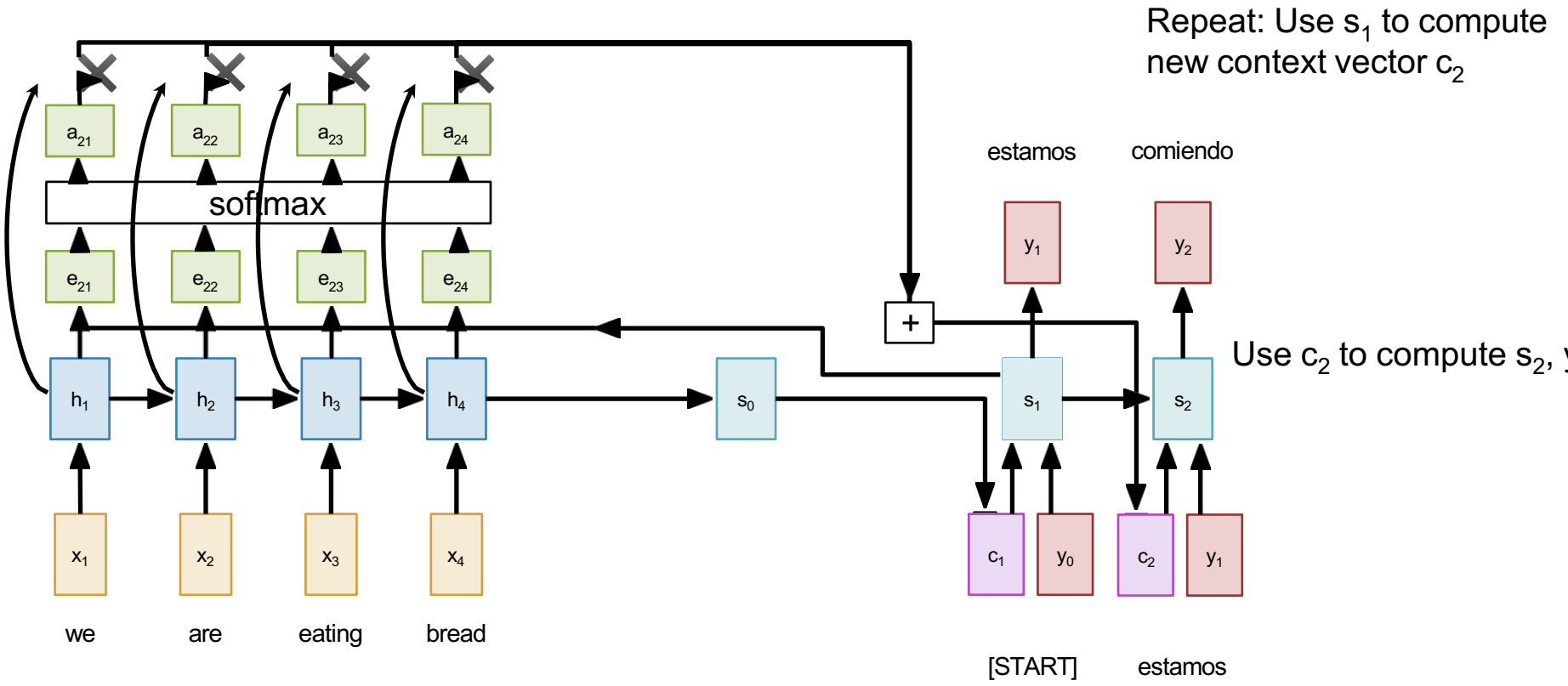
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention



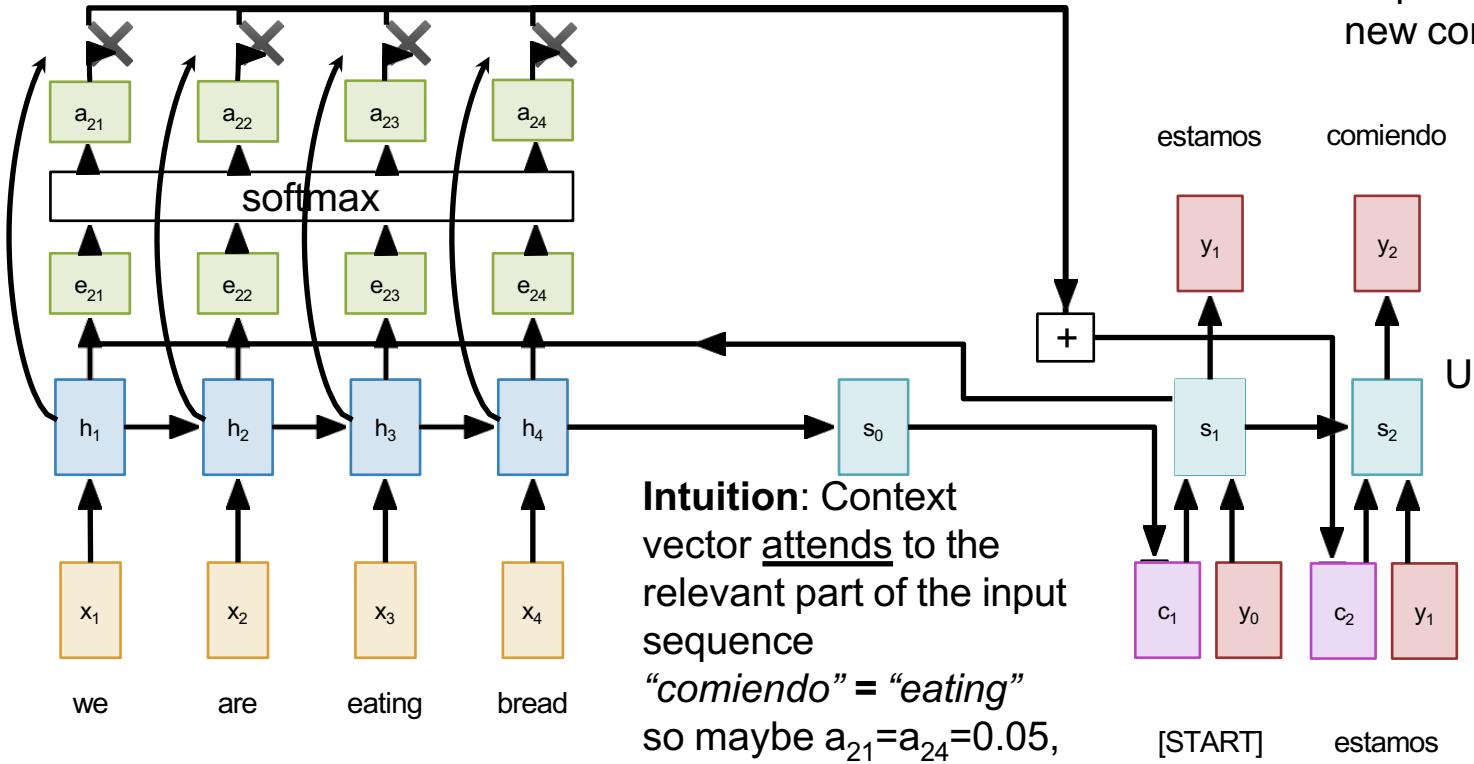
Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention



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

Sequence to Sequence with RNNs and Attention



Intuition: Context vector attends to the relevant part of the input sequence

"comiendo" **=** *"eating"*

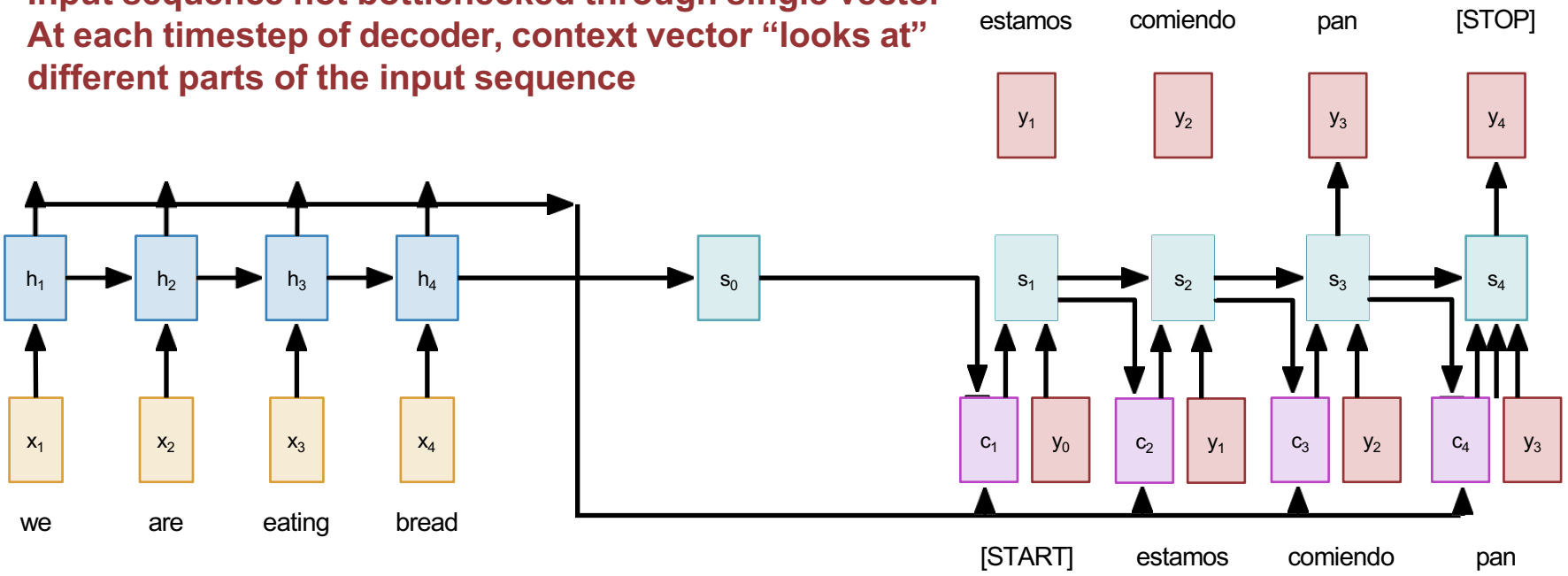
so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

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

Sequence to Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

- **Input sequence not bottlenecked through single vector**
- **At each timestep of decoder, context vector "looks at" different parts of the input sequence**



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

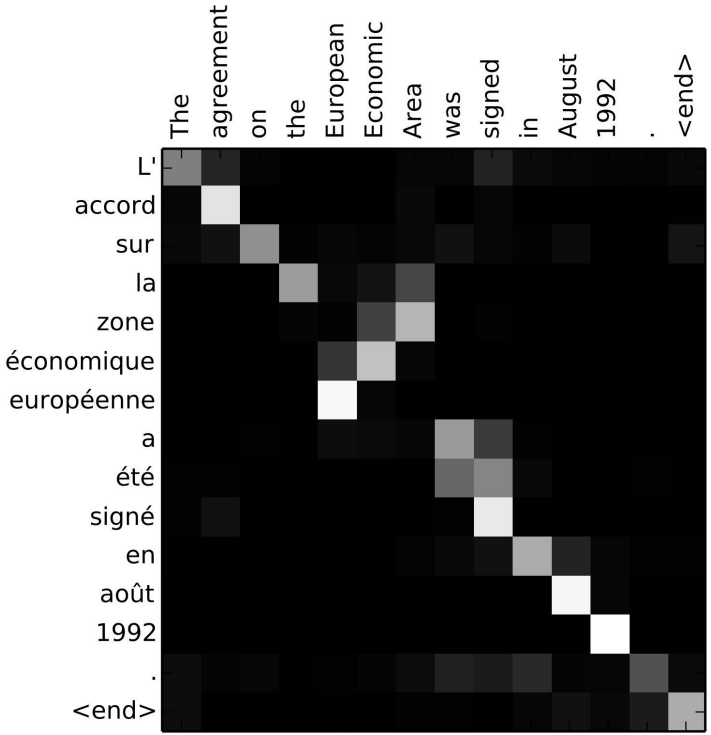
Sequence to Sequence with RNNs and Attention

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.”

Visualize attention weights $a_{t,i}$



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

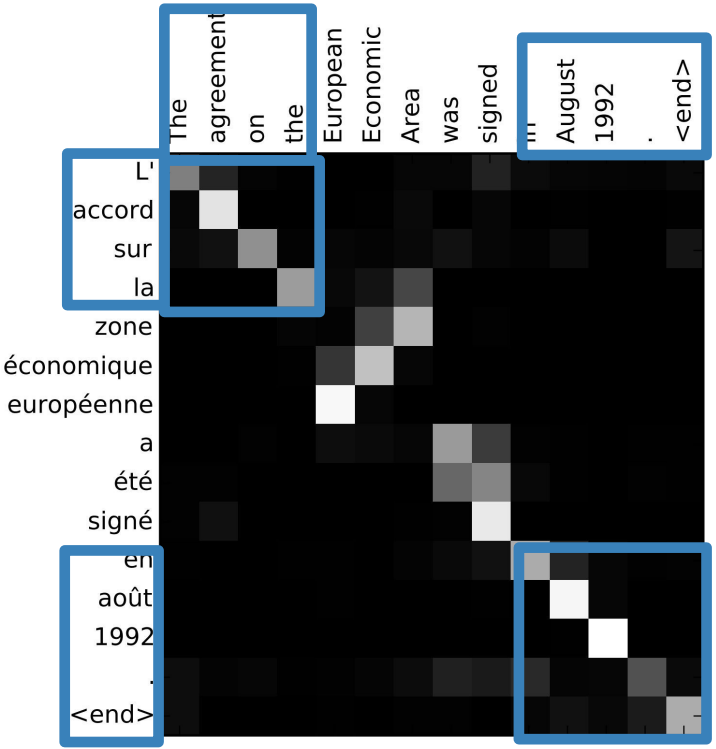
Sequence to Sequence with RNNs and Attention

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**."

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

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

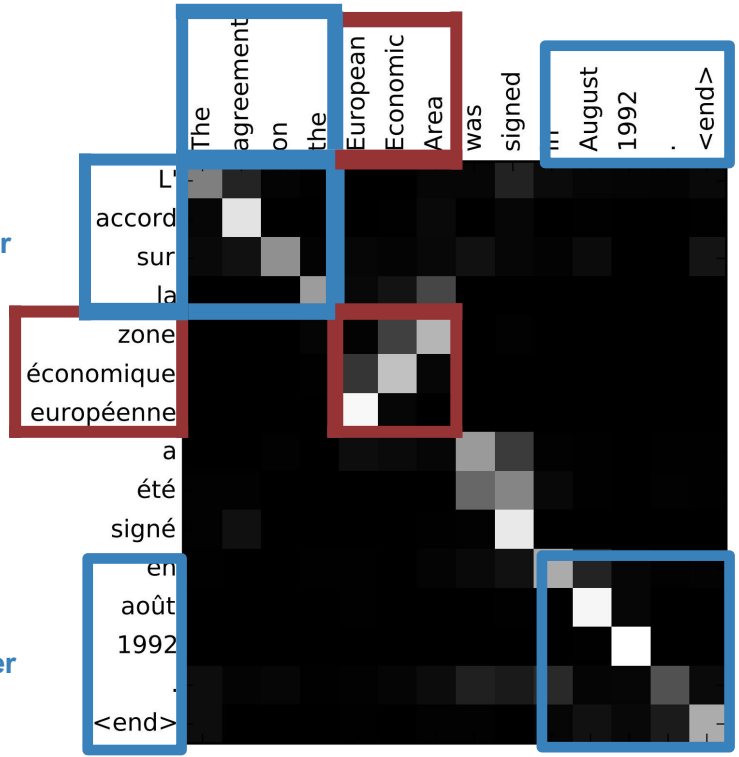
Sequence to Sequence with RNNs and Attention

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**."

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

Attention figures out different word orders

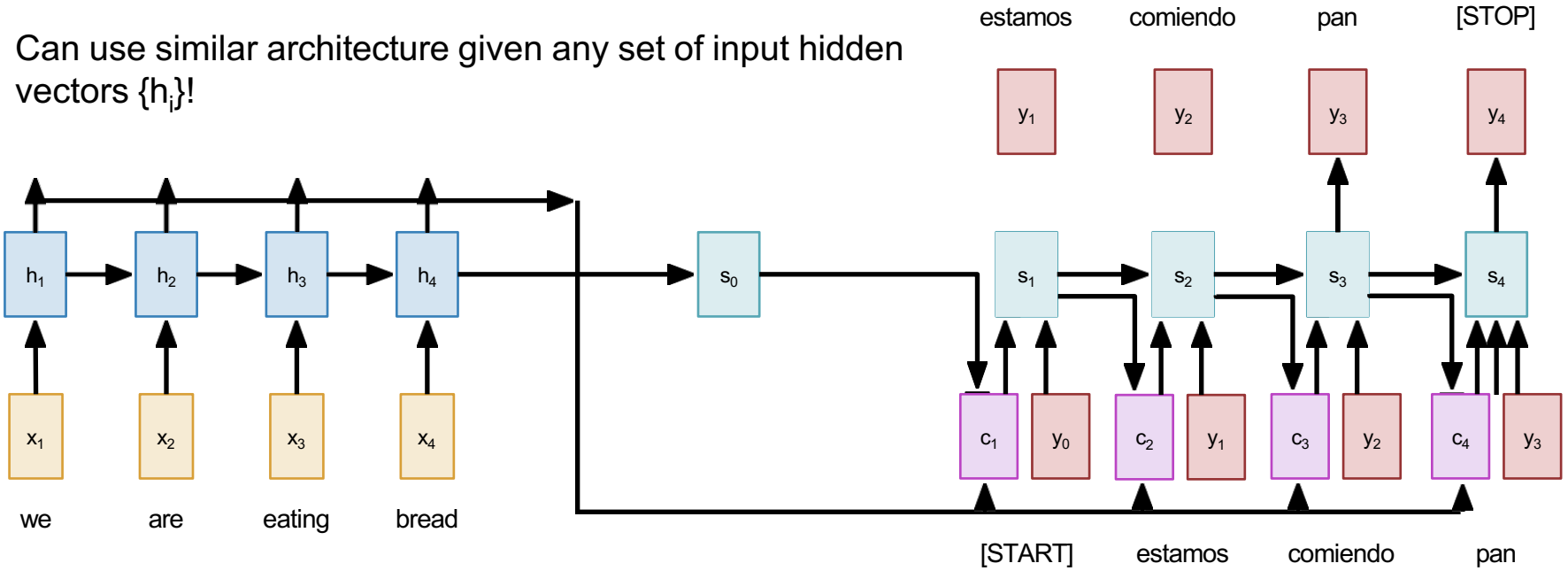
Diagonal attention means words correspond in order

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

Sequence to Sequence with RNNs and Attention

The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

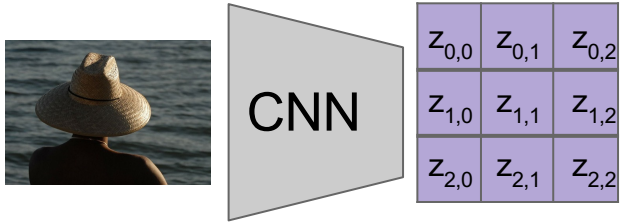
Can use similar architecture given any set of input hidden vectors $\{h_i\}$!



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

Image Captioning using spatial features

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$



Extract spatial features from a pretrained CNN

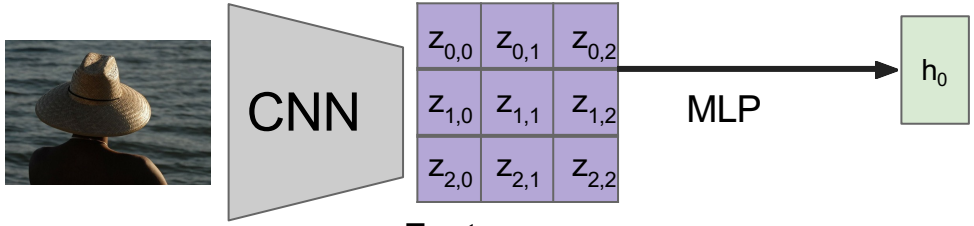
Features: H x W x D

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

Image Captioning using spatial features

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP



Extract spatial features from a pretrained CNN

Features: H x W x D

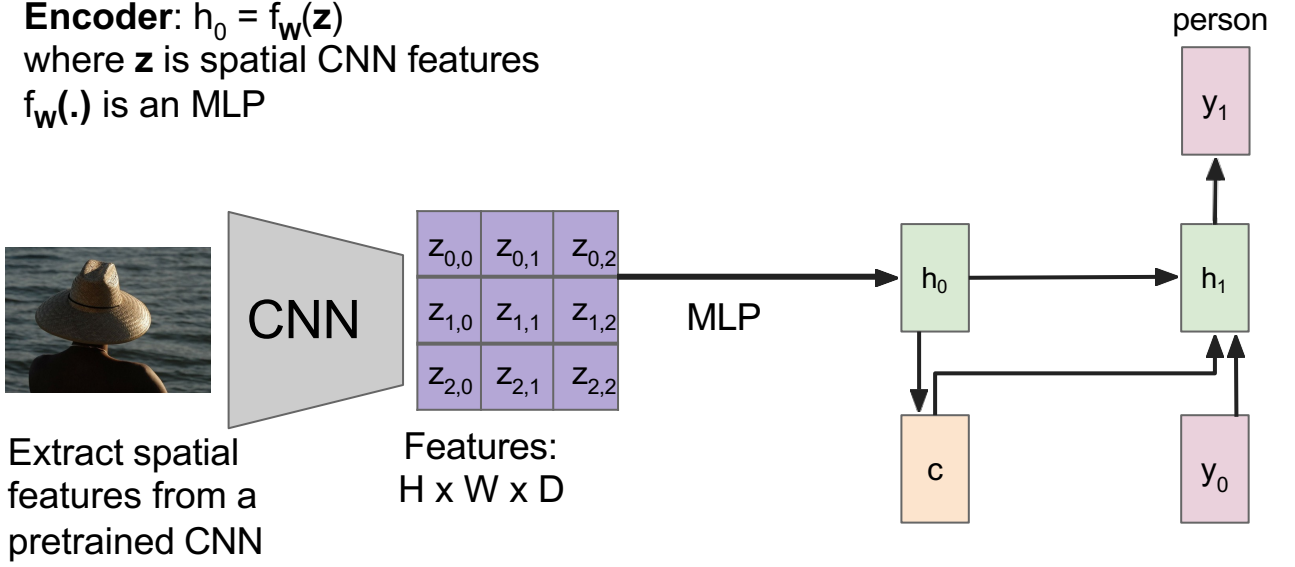
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning using spatial features

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



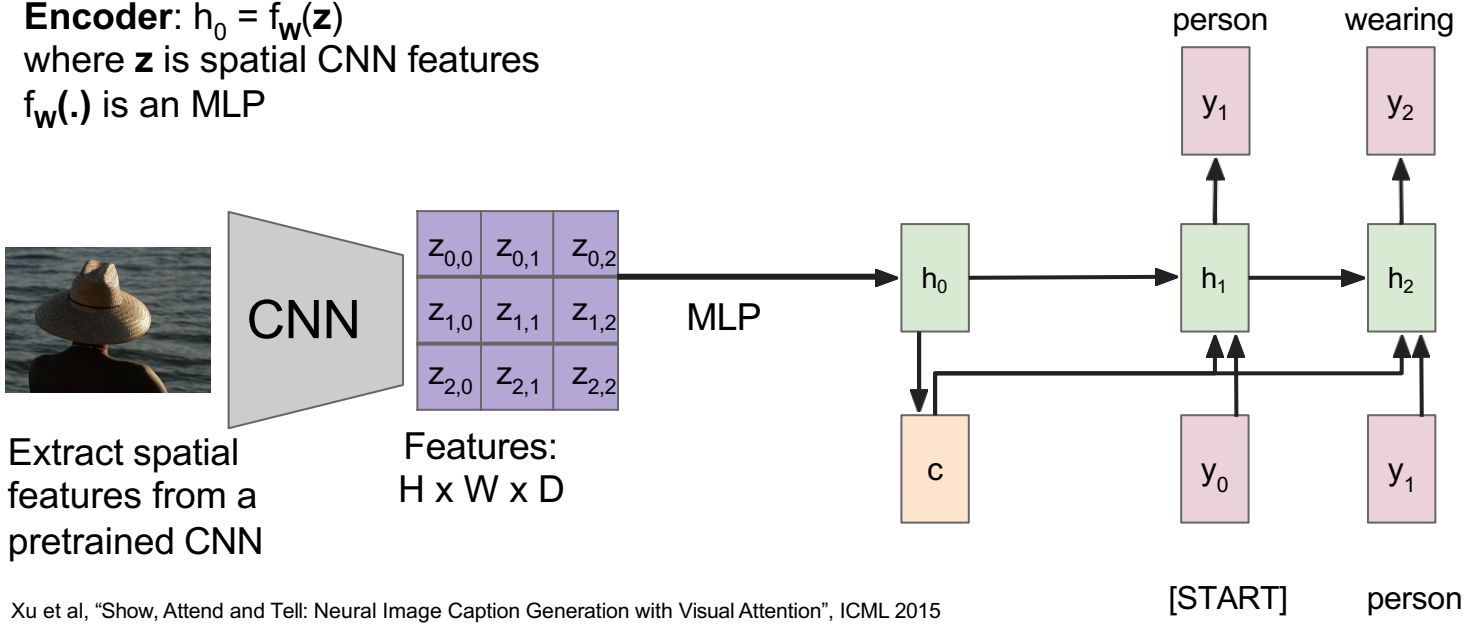
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Encoder: $h_0 = f_W(\mathbf{z})$
where **z** is spatial CNN features
$f_W(.)$ is an MLP



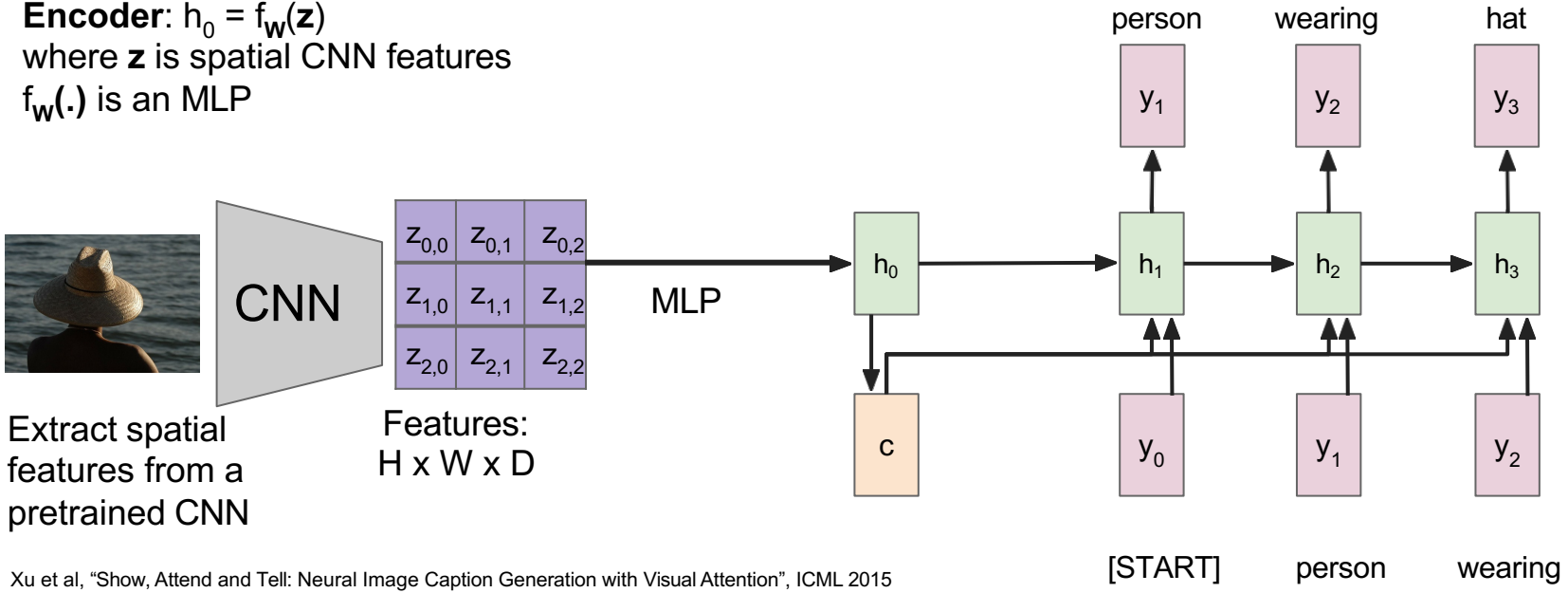
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



Extract spatial features from a pretrained CNN

Features: H x W x D

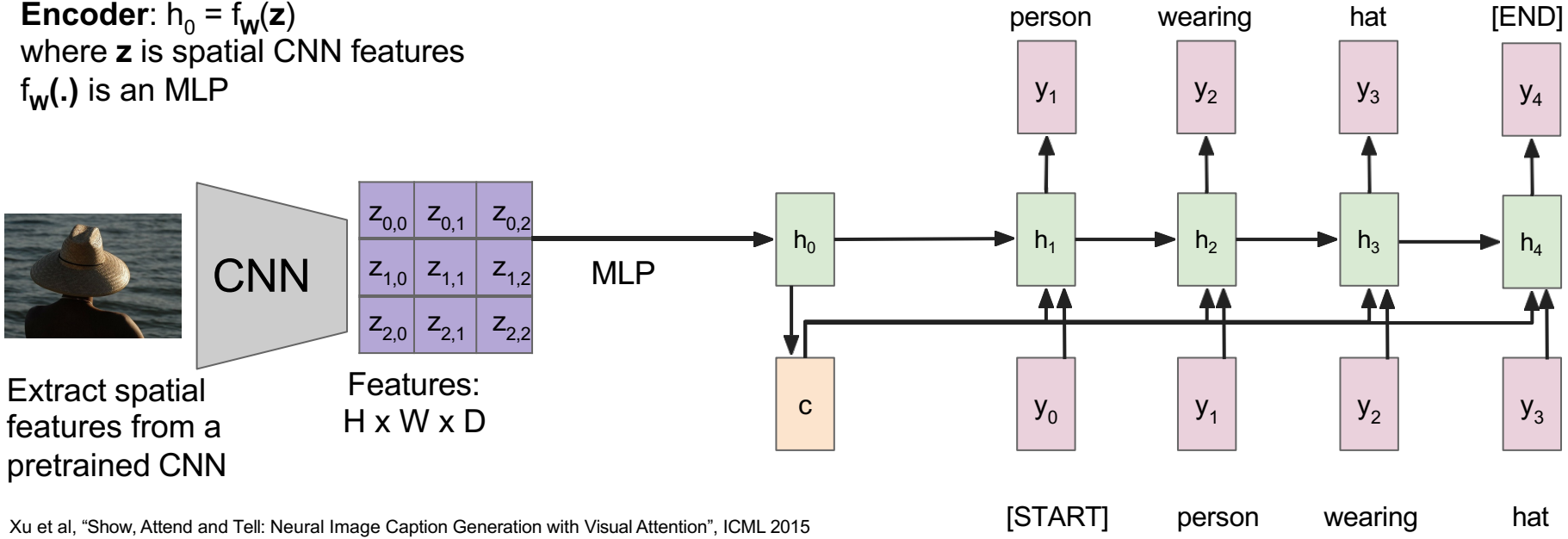
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



Extract spatial features from a pretrained CNN

Features: H x W x D

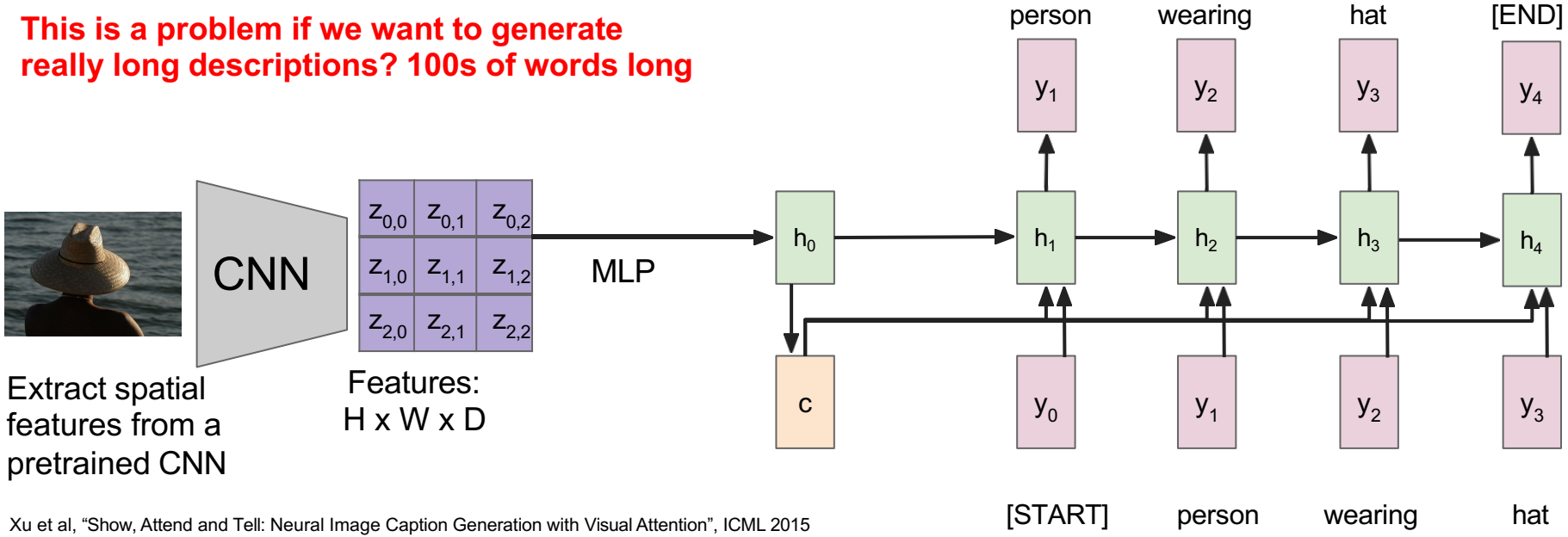
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

Problem: Input is "bottlenecked" through c

- Model needs to encode everything it wants to say within c

This is a problem if we want to generate really long descriptions? 100s of words long



Extract spatial features from a pretrained CNN

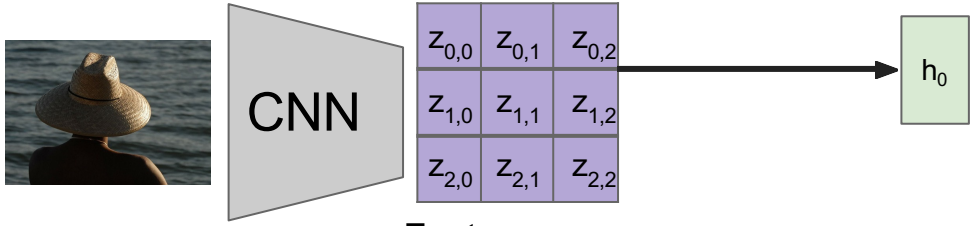
Features: H x W x D

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

Image Captioning with RNNs and Attention

Attention idea: New context vector at every time step.

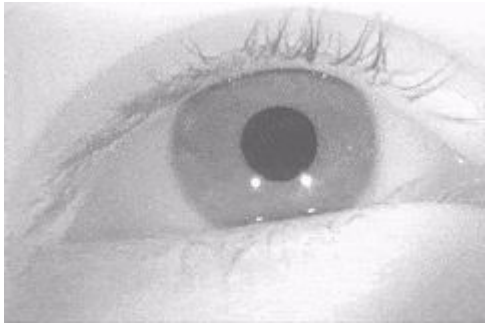
Each context vector will attend to different image regions



Extract spatial features from a pretrained CNN

Features: H x W x D

gif source



Attention Saccades in humans

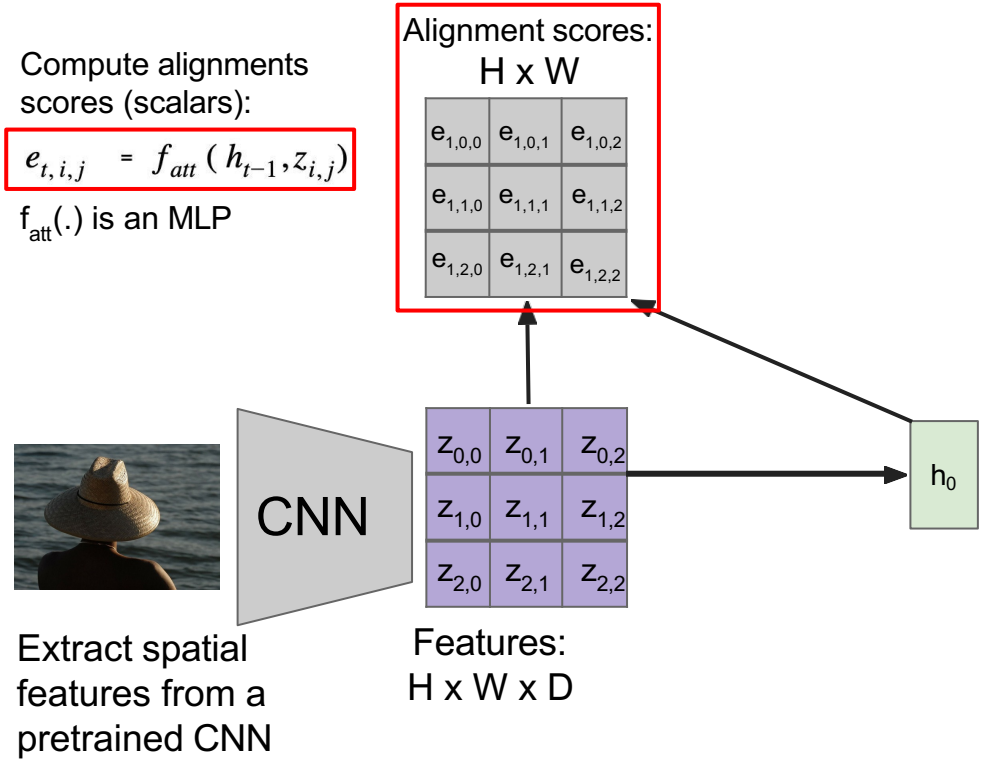
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

$f_{att}(.)$ is an MLP



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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

$f_{att}(.)$ is an MLP

Alignment scores: H x W

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

Attention: H x W

$a_{1,0,0}$	$a_{1,0,1}$	$a_{1,0,2}$
$a_{1,1,0}$	$a_{1,1,1}$	$a_{1,1,2}$
$a_{1,2,0}$	$a_{1,2,1}$	$a_{1,2,2}$

Normalize to get attention weights:

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

$0 < a_{t,i,j} < 1$,
attention values sum to 1

CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

h_0

Extract spatial features from a pretrained CNN

Features: H x W x D

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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

$f_{att}(.)$ is an MLP

Alignment scores: H x W

Attention: H x W

CNN

Extract spatial features from a pretrained CNN

Features: H x W x D

h_0

c_1

Normalize to get attention weights:

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

$0 < a_{t,i,j} < 1$,
attention values sum to 1

Compute context vector:

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

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

Image Captioning with RNNs and Attention

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

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

CNN

Extract spatial features from a pretrained CNN

$z_{0,0}$ $z_{0,1}$ $z_{0,2}$
$z_{1,0}$ $z_{1,1}$ $z_{1,2}$
$z_{2,0}$ $z_{2,1}$ $z_{2,2}$

Features: H x W x D

h_0

h_1

y_1

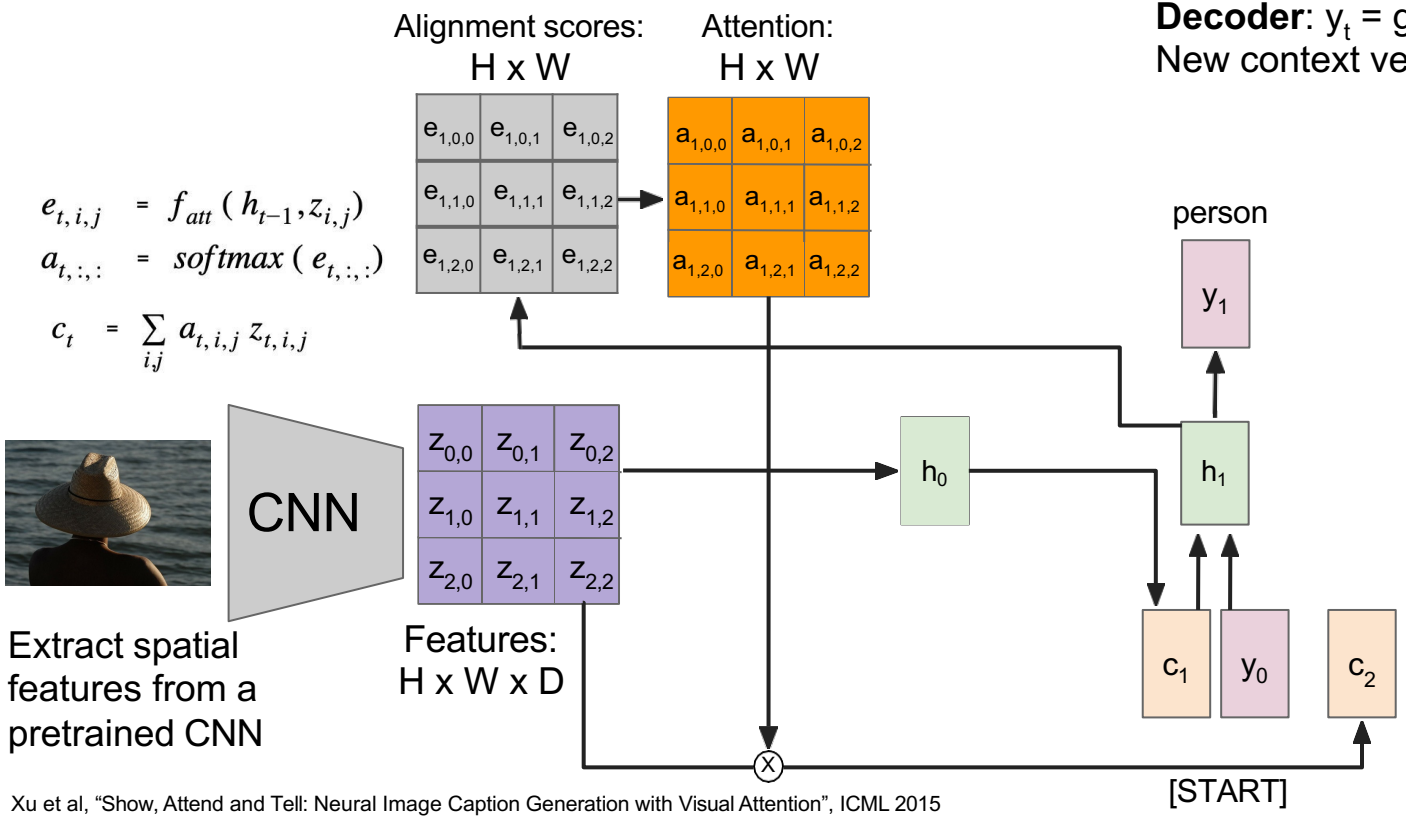
person

c_1 y_0

[START]

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

Image Captioning with RNNs and Attention



Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

Image Captioning with RNNs and Attention

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

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

CNN

Extract spatial features from a pretrained CNN

$z_{0,0}$ $z_{0,1}$ $z_{0,2}$
$z_{1,0}$ $z_{1,1}$ $z_{1,2}$
$z_{2,0}$ $z_{2,1}$ $z_{2,2}$

Features: H x W x D

h_0 h_1 h_2

y_1 y_2

person wearing

c_1 y_0 c_2 y_1

[START] person

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

Image Captioning with RNNs and Attention

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

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

CNN

Extract spatial features from a pretrained CNN

$z_{0,0}$ $z_{0,1}$ $z_{0,2}$
$z_{1,0}$ $z_{1,1}$ $z_{1,2}$
$z_{2,0}$ $z_{2,1}$ $z_{2,2}$

Features: H x W x D

h_0 h_1 h_2 h_3

y_1 y_2 y_3

person wearing hat

c_1 y_0 c_2 y_1 c_3 y_2

[START] person wearing

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

Image Captioning with RNNs and Attention

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

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

CNN

Extract spatial features from a pretrained CNN

$z_{0,0}$ $z_{0,1}$ $z_{0,2}$
$z_{1,0}$ $z_{1,1}$ $z_{1,2}$
$z_{2,0}$ $z_{2,1}$ $z_{2,2}$

Features: H x W x D

h_0 h_1 h_2 h_3 h_4

y_1 y_2 y_3 y_4

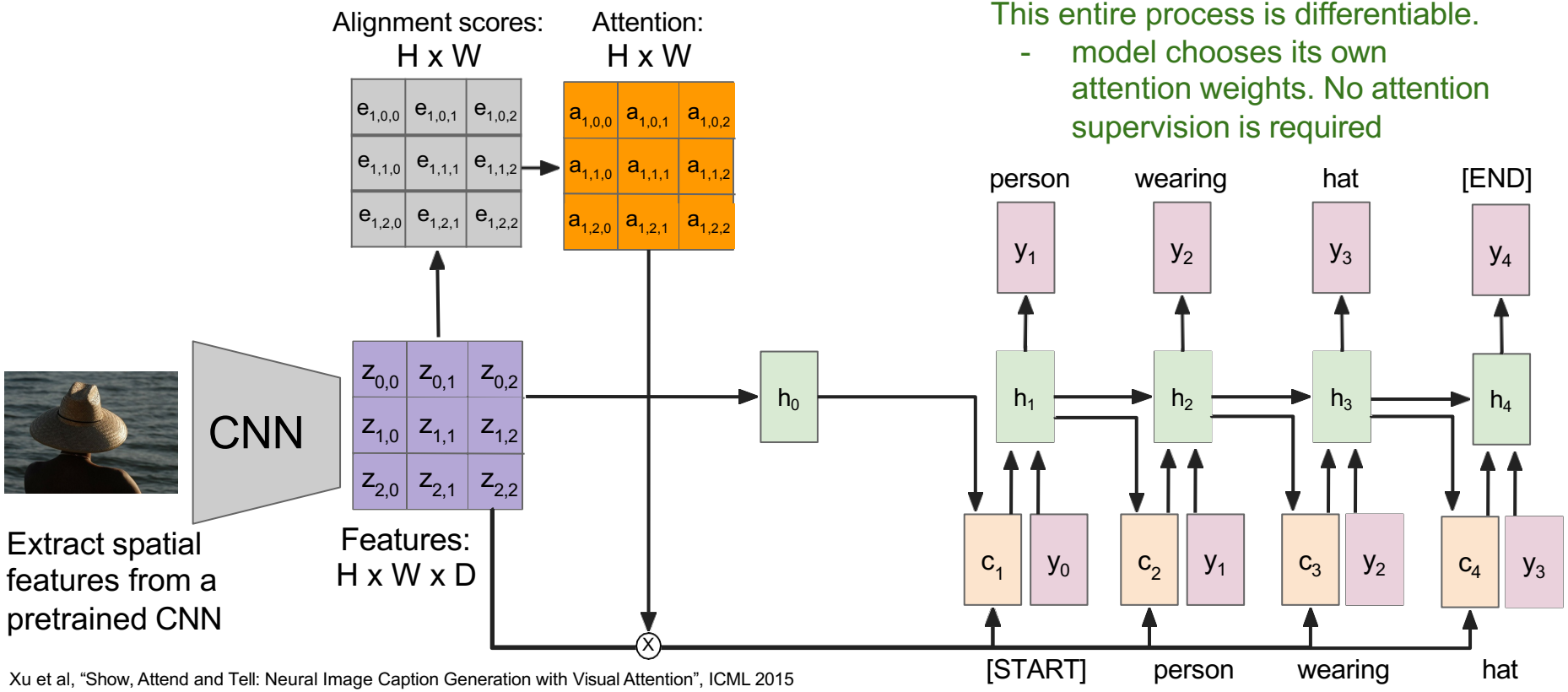
person wearing hat [END]

c_1 y_0 c_2 y_1 c_3 y_2 c_4 y_3

[START] person wearing hat

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

Image Captioning with RNNs and Attention

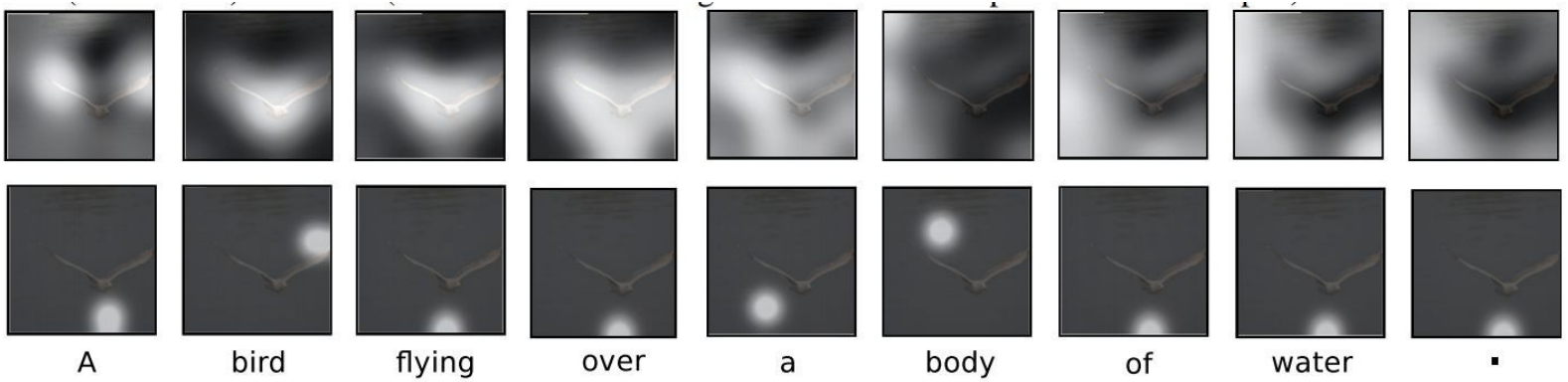


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention

Soft attention

Hard attention
(requires reinforcement learning)



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

Image Captioning with Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

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

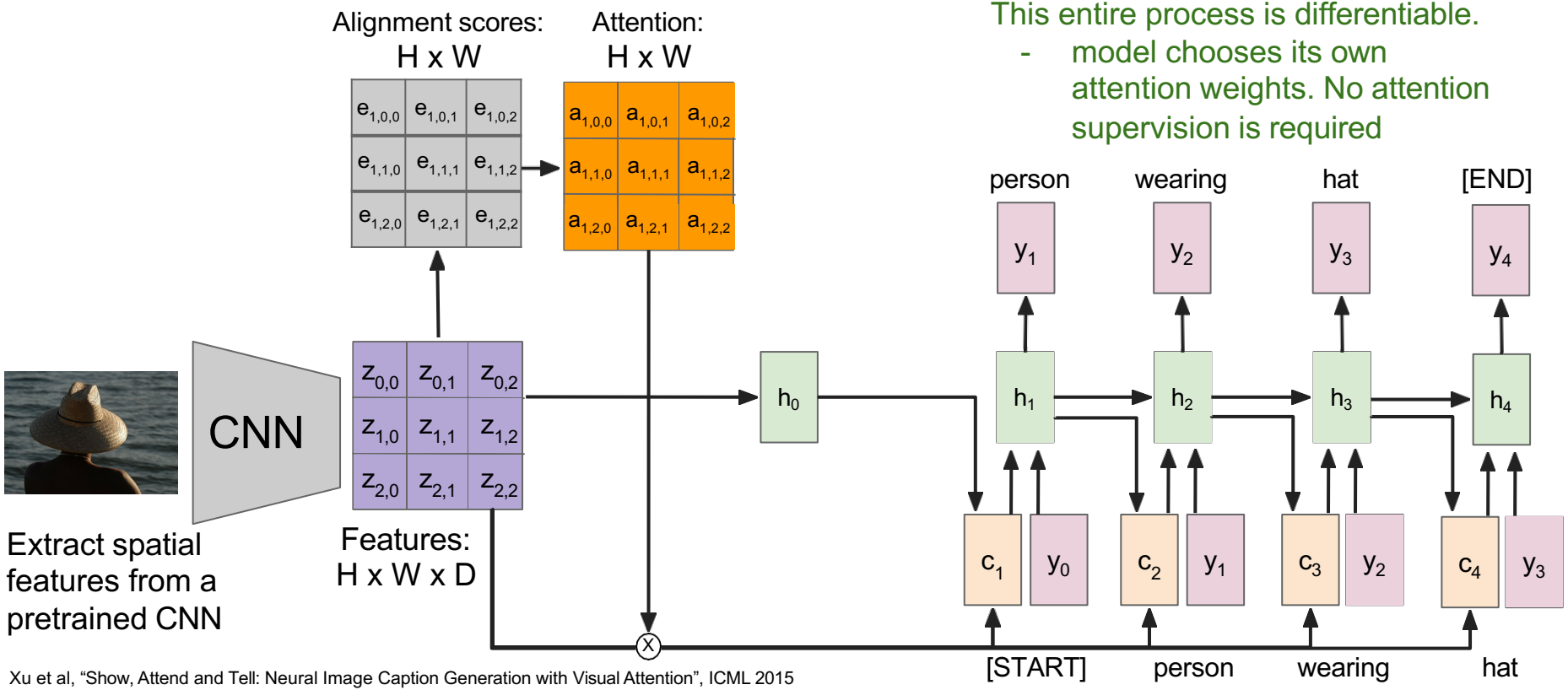
A little girl sitting on a bed with a teddy bear.

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

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

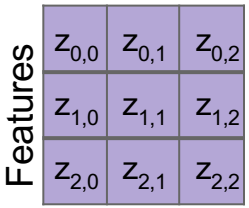
Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

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

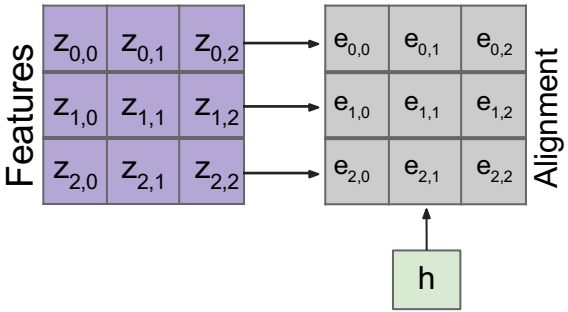


h

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

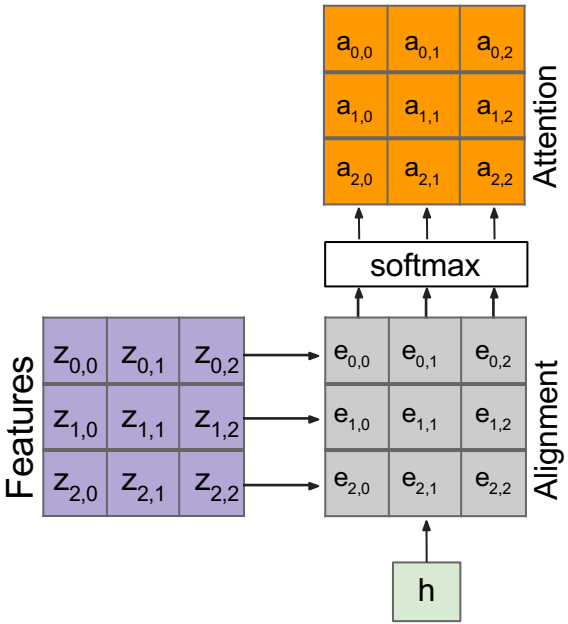
Attention we just saw in image captioning

Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$



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

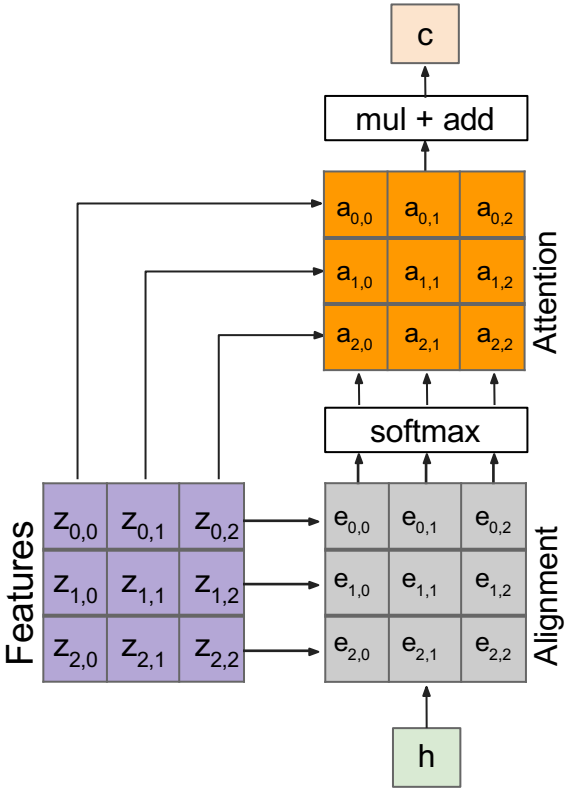
Attention we just saw in image captioning



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

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

Attention we just saw in image captioning

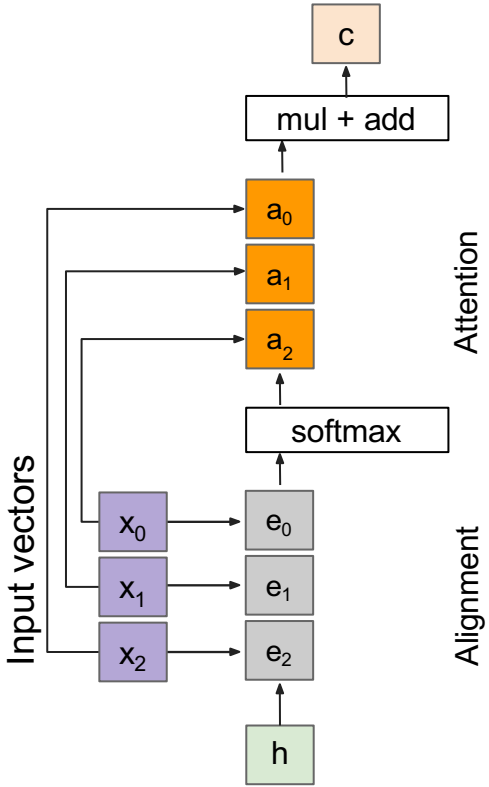


Outputs:
context vector: **c** (shape: D)

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

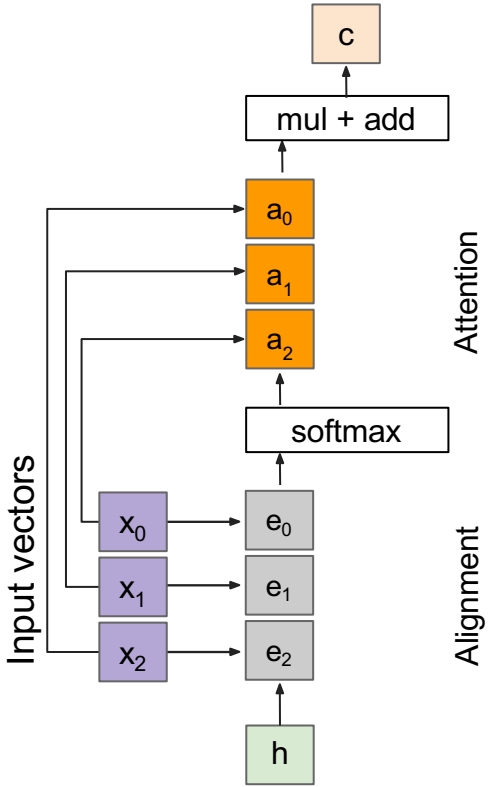
Operations:
Alignment: $e_i = f_{att}(h, x_i)$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

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)

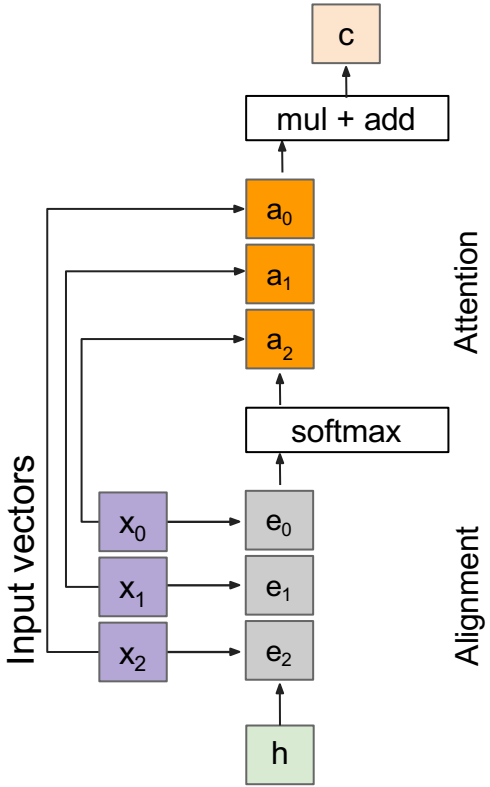
Operations:
Alignment: $e_i = h \cdot x_i$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

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)

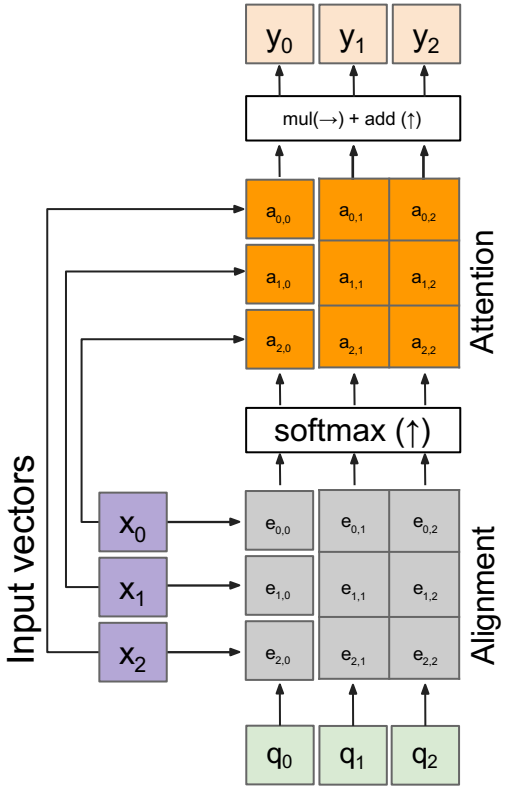
Operations:
Alignment: $e_i = h \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_i a_i x_i$

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

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

- Larger dimensions 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)

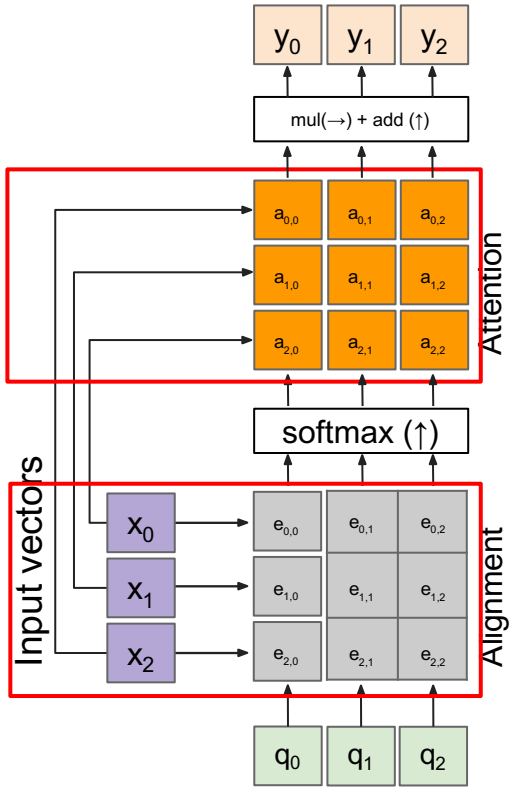
Multiple query vectors
- each query creates a new output context vector

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

Multiple query vectors

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

General attention layer



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

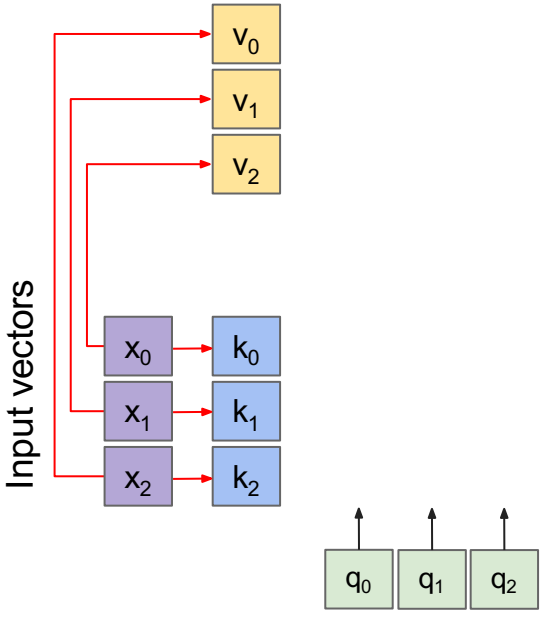
Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

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.

- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

General attention layer



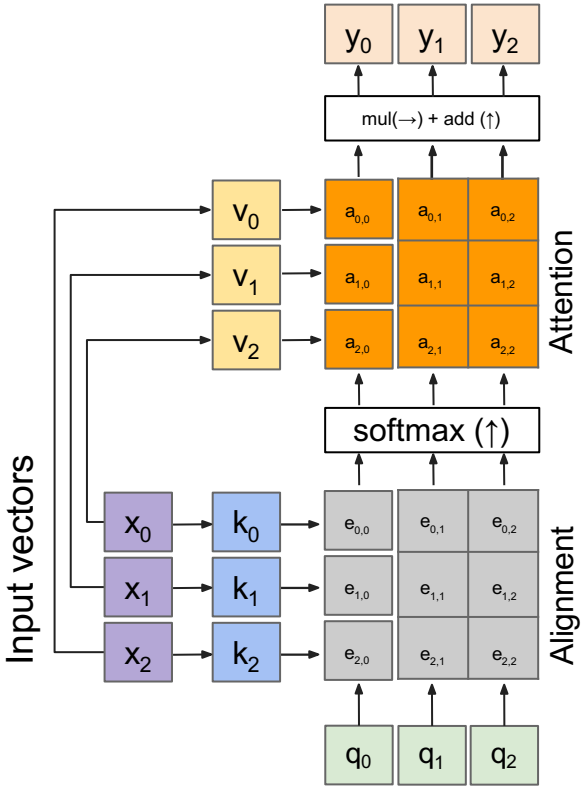
Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$

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.

General attention layer



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

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

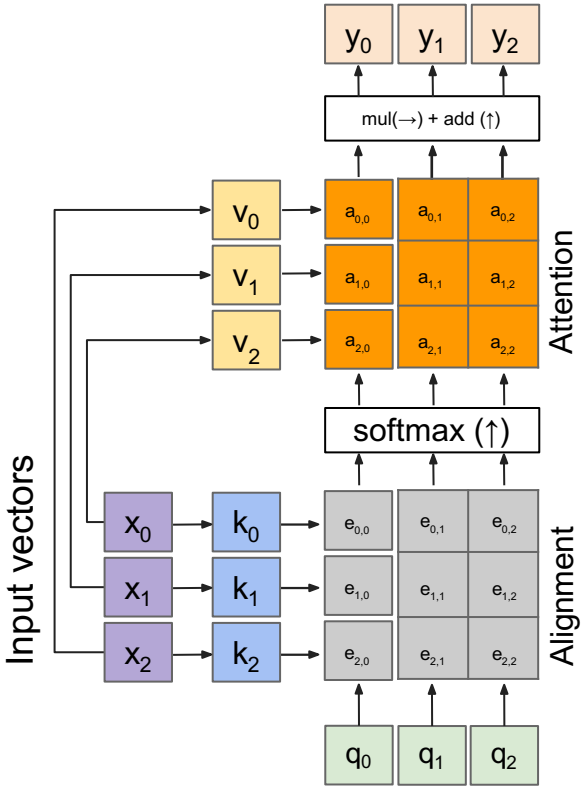
Inputs:
Input vectors: **x** (shape: N x D)
~~Queries: **q** (shape: M x D_k)~~

The input and output dimensions can now change depending on the key and value FC layers

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



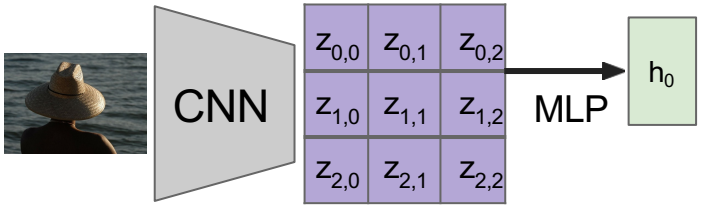
Outputs:
context vectors: **y** (shape: D_v)

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

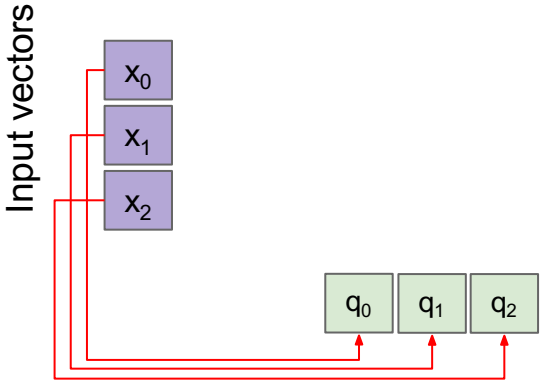
Inputs:
Input vectors: **x** (shape: N x D)
~~Queries: **q** (shape: M x D_k)~~

Recall that the query vector was a function of the input vectors

Encoder: $h_0 = f_W(\mathbf{z})$
where **z** is spatial CNN features
$f_W(.)$ is an MLP



Self attention layer



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: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

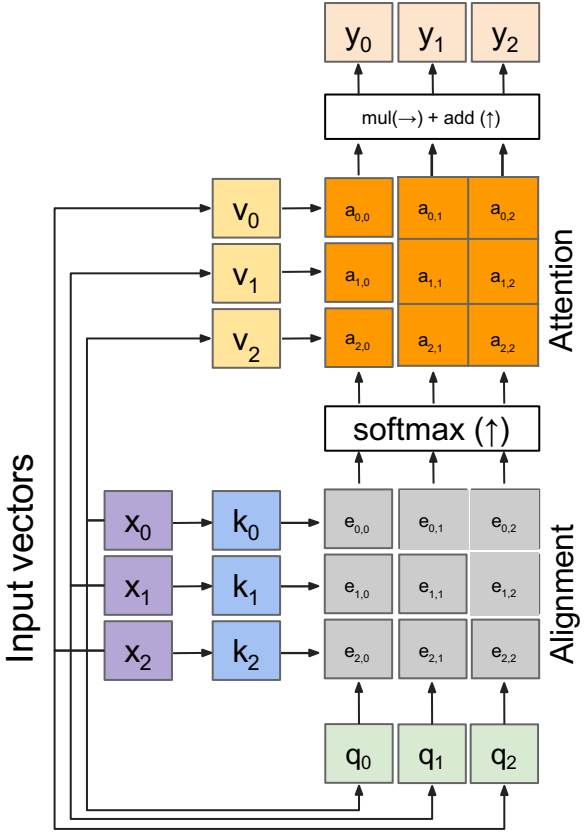
We can calculate the query vectors from the input vectors, therefore, defining a "self-attention" layer.

Instead, query vectors are calculated using a FC layer.

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

No input query vectors anymore

Self attention layer

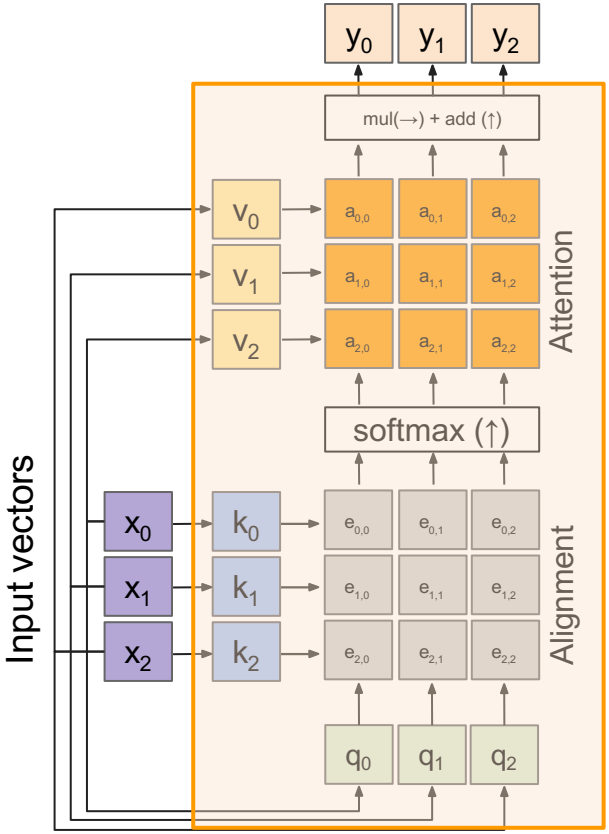


Outputs:
context vectors: **y** (shape: D_v)

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)

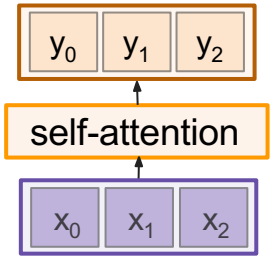
Self attention layer - attends over sets of inputs



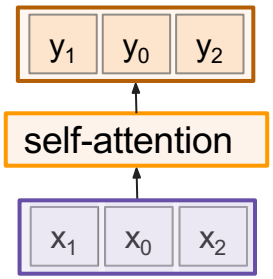
Outputs:
context vectors: **y** (shape: D_v)

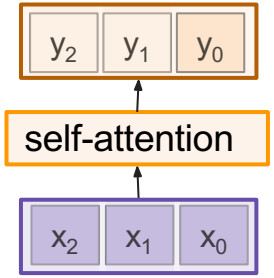
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

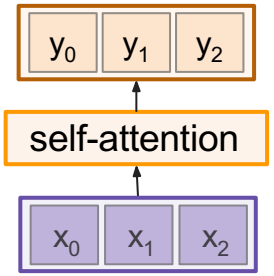
Inputs:
Input vectors: **x** (shape: N x D)



Self attention layer - attends over sets of inputs





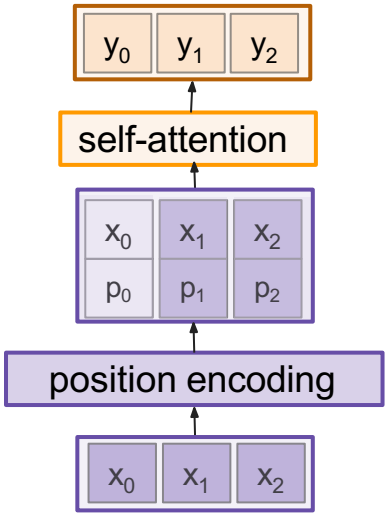


Permutation equivariant

Self-attention layer 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

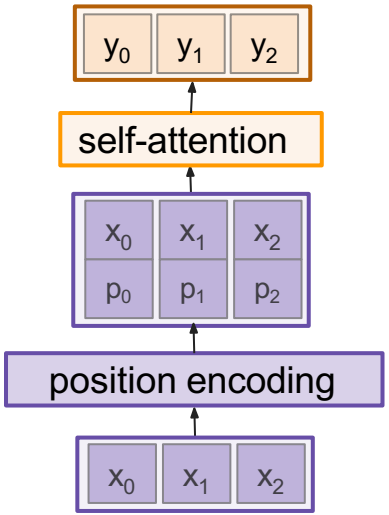
We use a function *pos*: $N \rightarrow R^d$
to process the position j of the vector into a d-dimensional vector

So, $p_j = pos(j)$

Desiderata of *pos*(.) :

1. It should output a **unique** encoding for each time-step (word's position in a sentence)
2. **Distance** between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to **longer** sentences without any efforts. Its values should be bounded.
4. It must be **deterministic**.

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function *pos*: $N \rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, $p_j = pos(j)$

Options for *pos*(.)

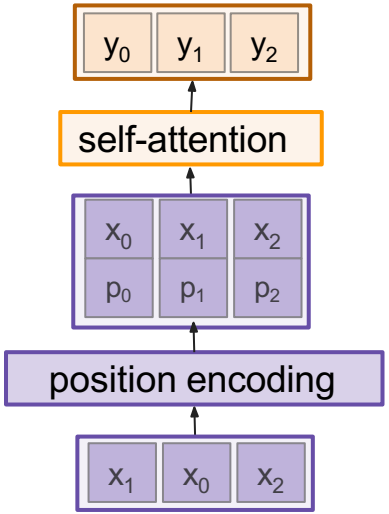
1. Learn a lookup table:
   - Learn parameters to use for *pos*(t) for $t \in [0, T)$
   - Lookup table contains T x d parameters.

Desiderata of *pos*(.) :

1. It should output a **unique** encoding for each time-step (word's position in a sentence)
2. **Distance** between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to **longer** sentences without any efforts. Its values should be bounded.
4. It must be **deterministic**.

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function *pos*: N $\rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, p_j = *pos*(j)

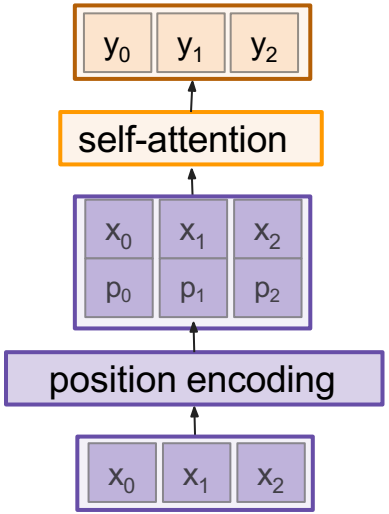
Options for *pos*(.)

1. Learn a lookup table:
   - Learn parameters to use for *pos*(t) for t ε [0, T)
   - Lookup table contains T x d parameters.
2. Design a fixed function with the desiderata

$$\mathbf{p(t)} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \vdots \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_d \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

Vaswani et al, “Attention is all you need”, NeurIPS 2017

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function *pos*: N $\rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, p_j = *pos*(j)

Options for *pos*(.)

1. Learn a lookup table:
   - Learn parameters to use for *pos*(t) for t ε [0, T)
   - Lookup table contains T x d parameters.
2. Design a fixed function with the desiderata

p(t) =

$$\begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \vdots \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_d$$

Intuition:

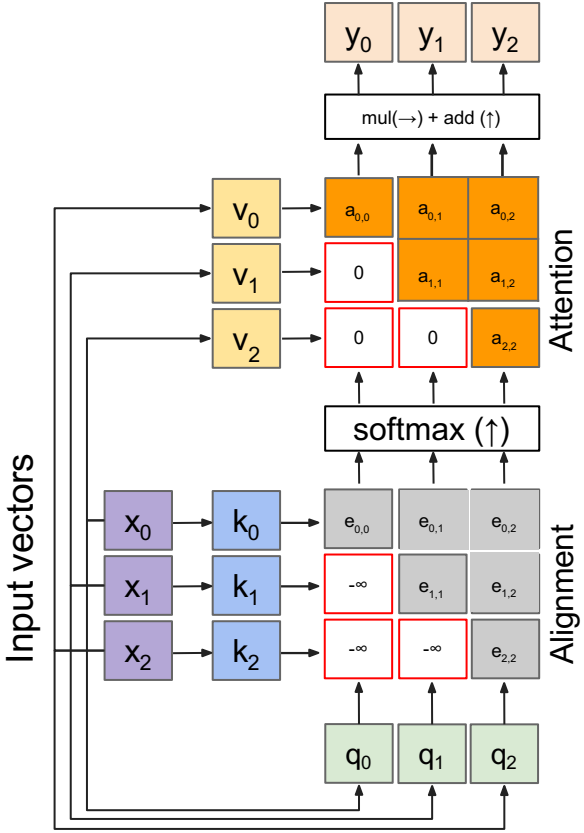
0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

where $\omega_k = \frac{1}{10000^{2k/d}}$

image source

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

Masked self-attention layer



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

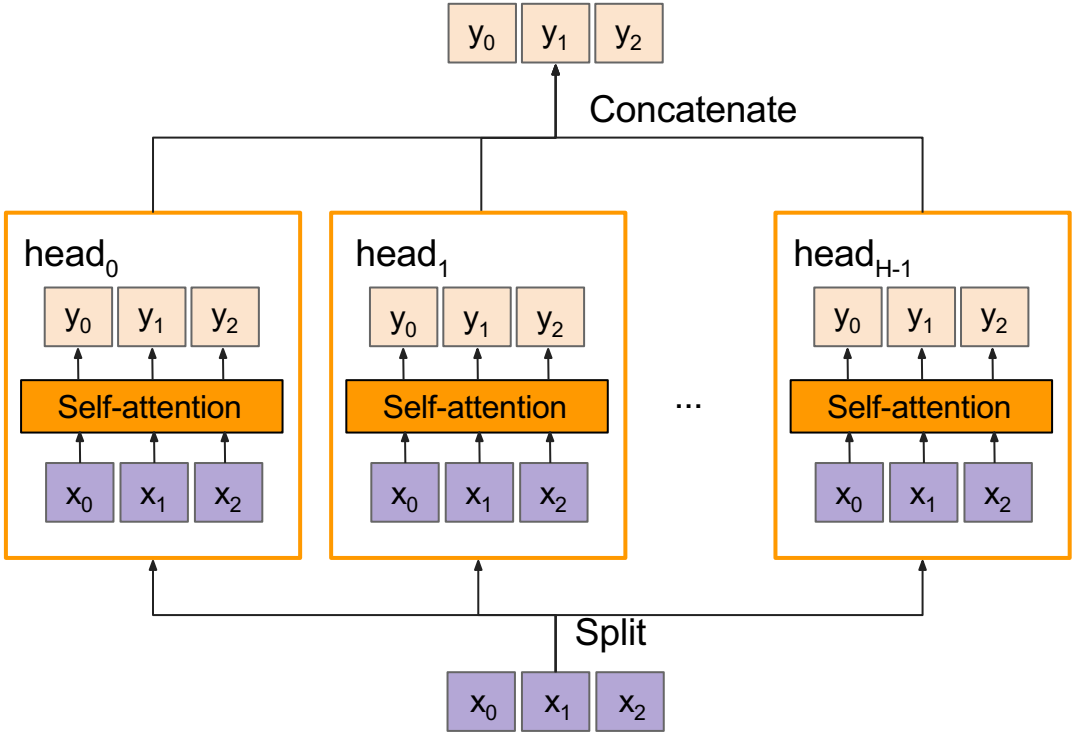
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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

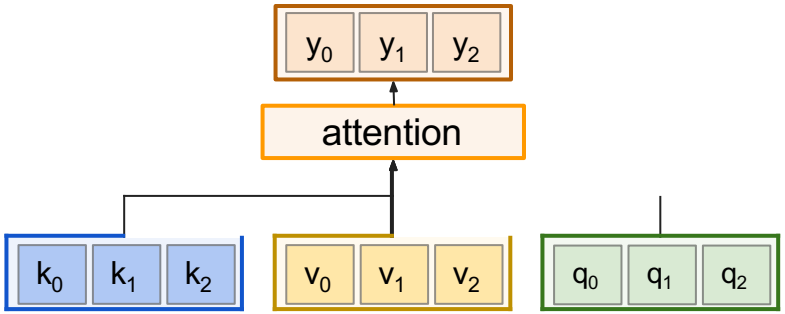
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

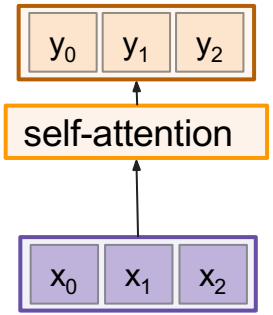
Multi-head self-attention layer

- Multiple self-attention heads in parallel



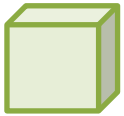
General attention versus self-attention





Example: CNN with Self-Attention

Input Image

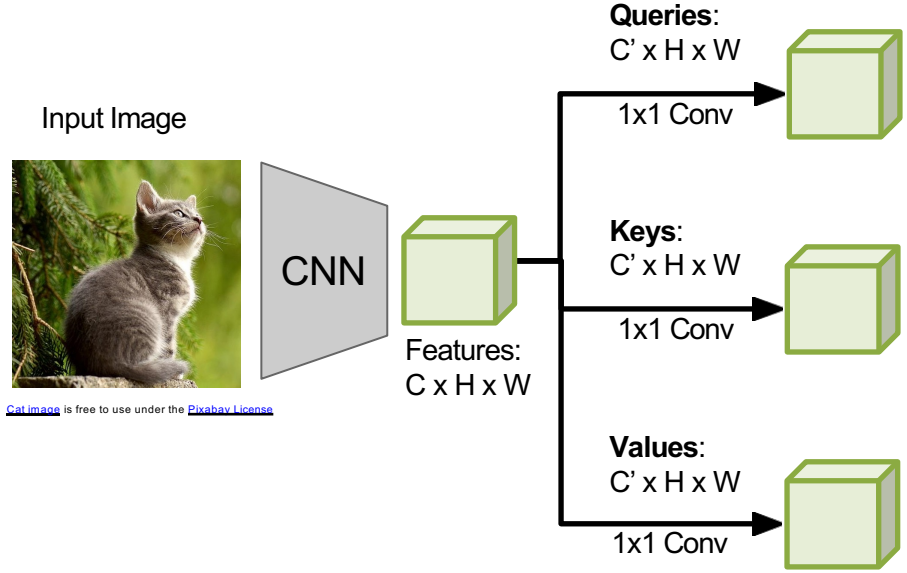


Features:
C x H x W

Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Slide credit: Justin Johnson

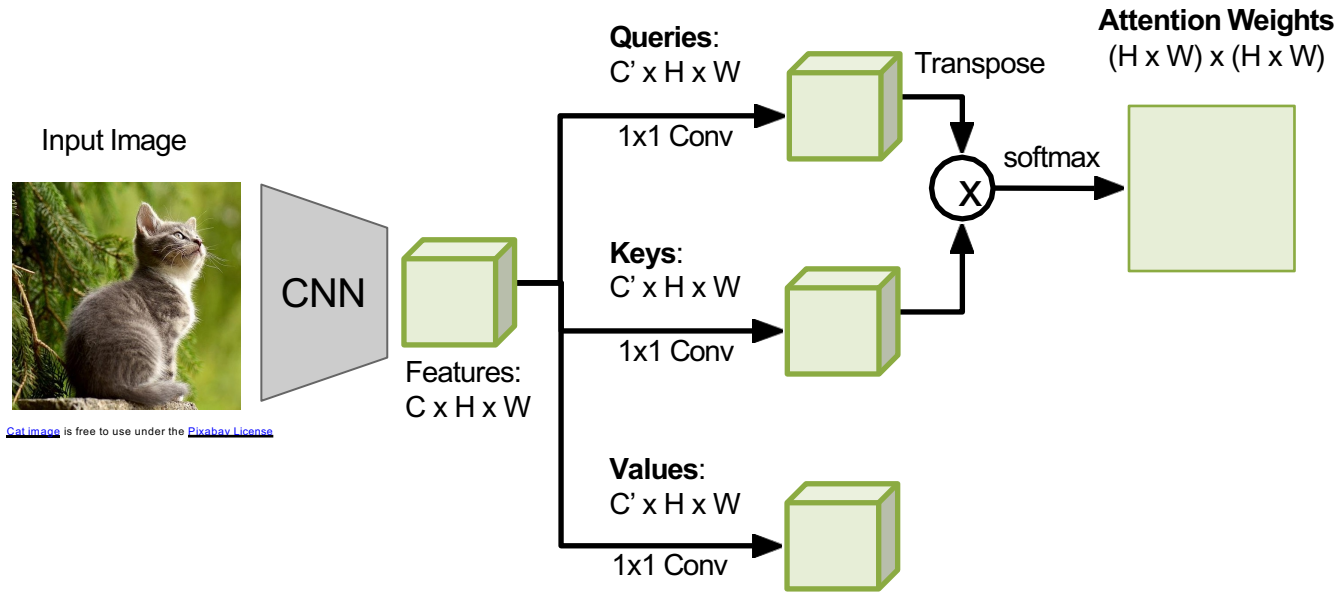
Example: CNN with Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson

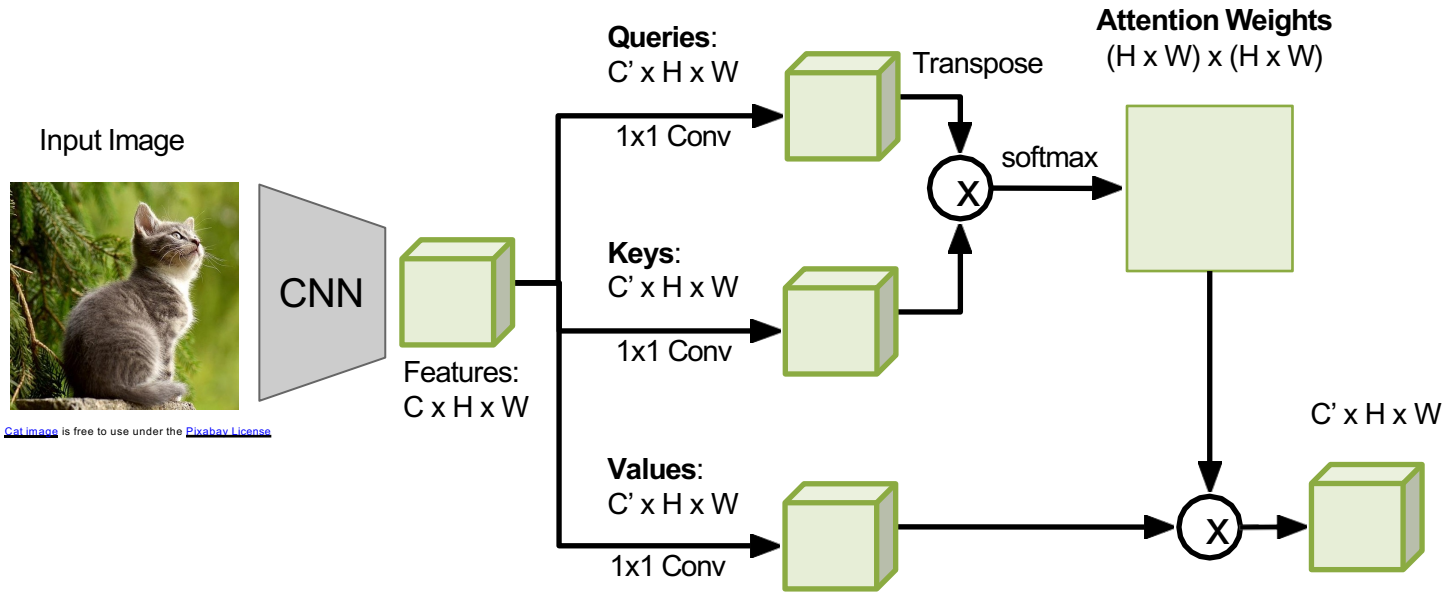
Example: CNN with Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson

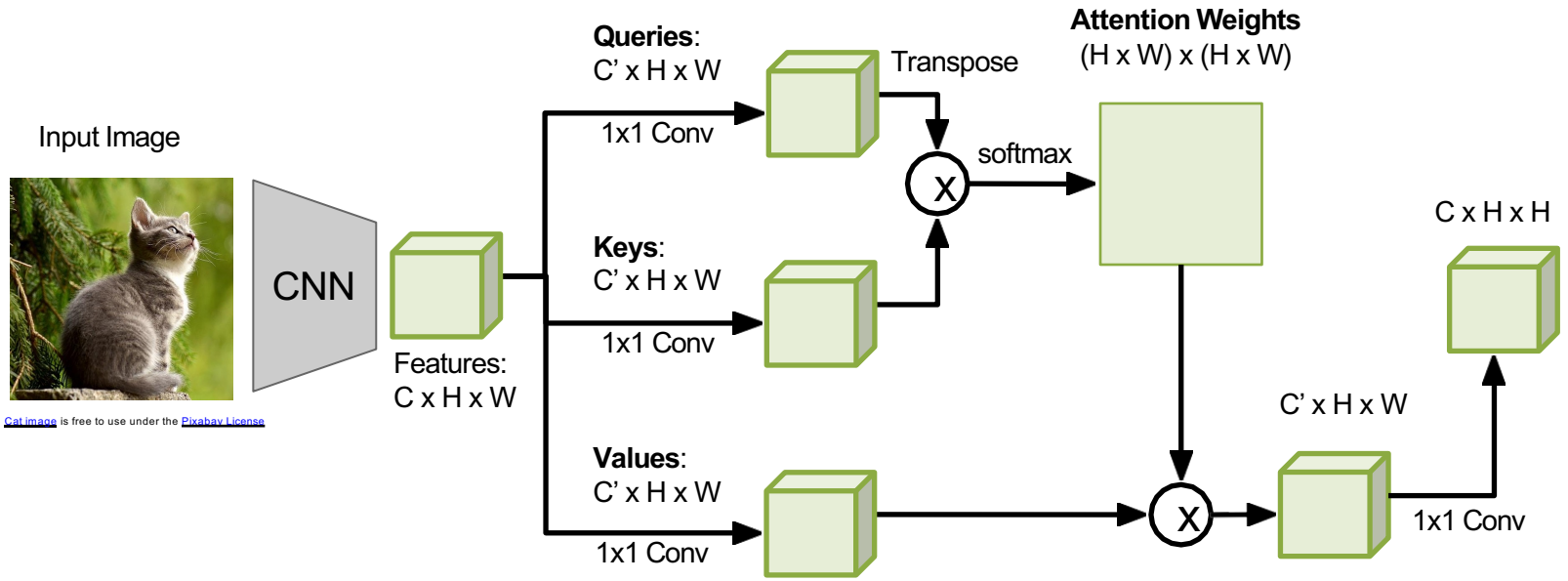
Example: CNN with Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson

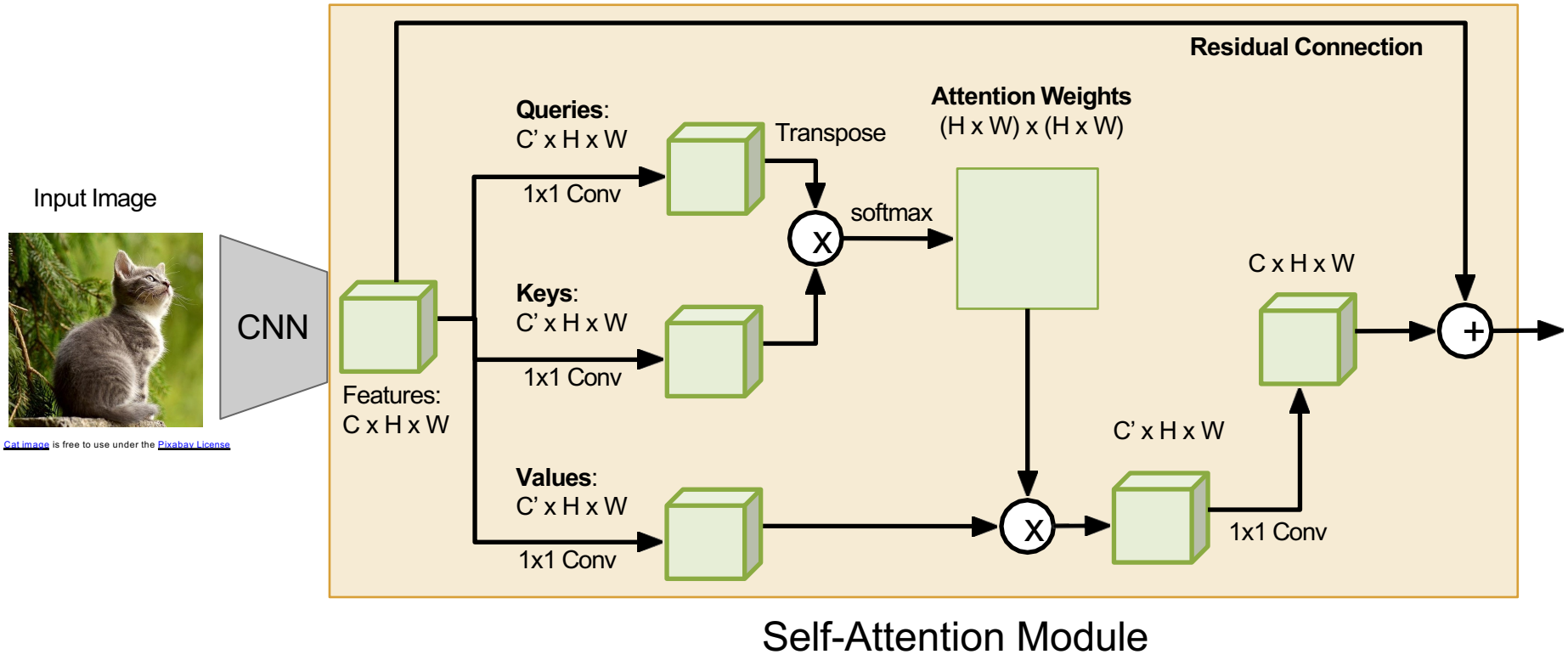
Example: CNN with Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson

Example: CNN with Self-Attention



Self-Attention Module

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson

Comparing RNNs to Transformer

RNNs
(+) LSTMs work reasonably well for long sequences.
(-) Expects an ordered sequences of inputs
(-) Sequential computation: subsequent hidden states can only be computed after the previous ones are done.

Transformer:
(+) Good at long sequences. Each attention calculation looks at all inputs.
(+) Can operate over unordered sets or ordered sequences with positional encodings.
(+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.
(-) Requires a lot of memory: N x M alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

"ImageNet Moment for Natural Language Processing"

Pretraining:
Download a lot of text from the internet

Train a giant Transformer model for language modeling

Finetuning:
Fine-tune the Transformer on your own NLP task

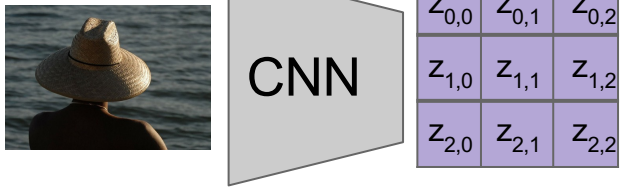
On the Opportunities and Risks of Foundation Models

Rishi Bommasani* Drew A. Hudson Ehsan Adeli Russ Altman Simran Arora
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut Emma Brunskill
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon Niladri Chatterji
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky Chris Donahue
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy Kawin Ethayarajh
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel Noah Goodman
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Henderson John Hewitt
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard Saahil Jain
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling Fereshte Khani
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Rohith Kuditipudi
Ananya Kumar Faisal Ladhak Mina Lee Tony Lee Jure Leskovec Isabelle Levent
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher D. Manning
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair Avanika Narayan
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Niebles Hamed Nilforoshan
Julian Nyarko Giray Ogut Laurel Orr Isabel Papadimitriou Joon Sung Park Chris Piech
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich Hongyu Ren
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher Ré Dorsa Sadigh
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasan Alex Tamkin
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang William Wang Bohan Wu
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan You Matei Zaharia
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zheng Kaitlyn Zhou
Percy Liang*[1]

Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University

Image Captioning using Transformers

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$



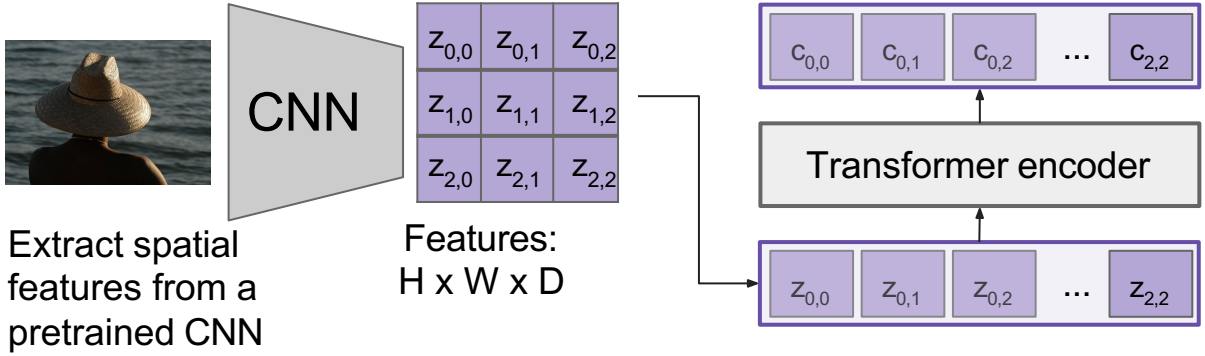
Extract spatial features from a pretrained CNN

Features: H x W x D

Image Captioning using Transformers

Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: **c** = $T_W(\mathbf{z})$
where **z** is spatial CNN features
$T_W(.)$ is the transformer encoder



Extract spatial features from a pretrained CNN

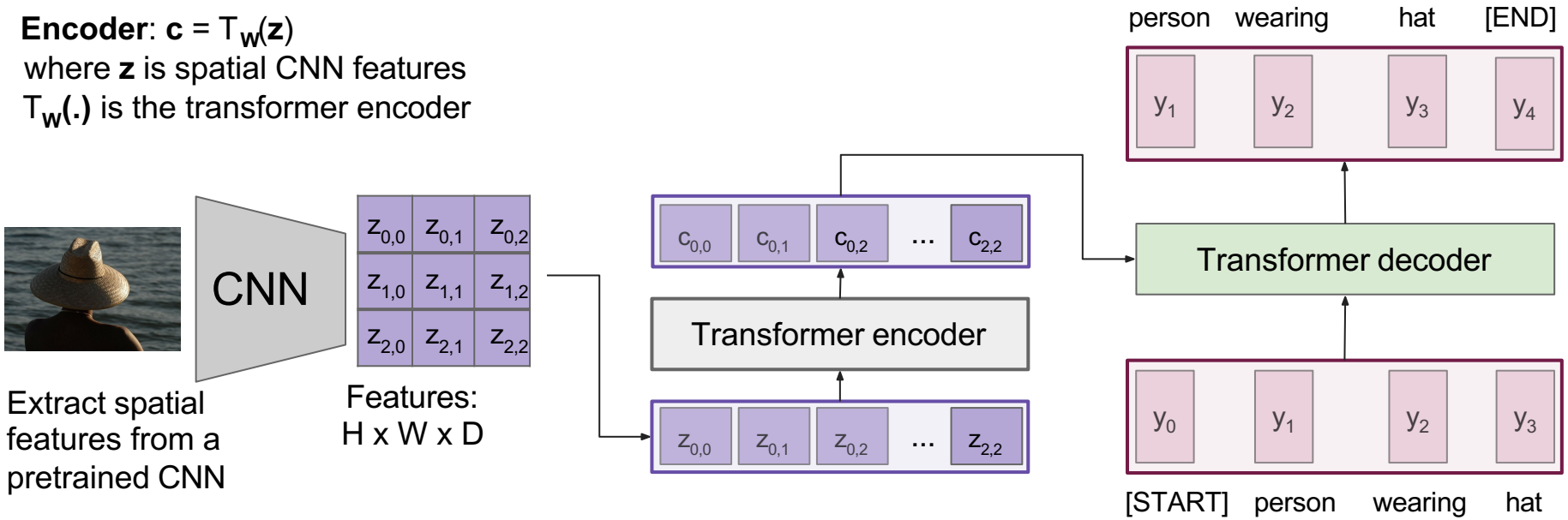
Features: H x W x D

Image Captioning using Transformers

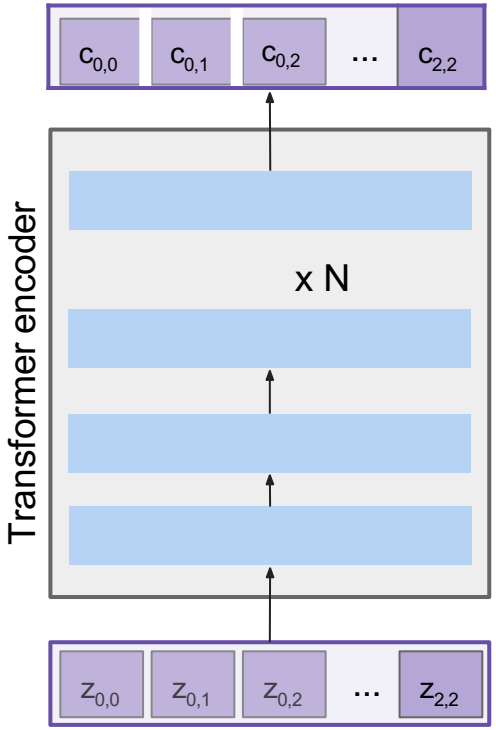
Input: Image **I**
Output: Sequence **y** = $y_1, y_2, ..., y_T$

Encoder: $\mathbf{c} = T_W(\mathbf{z})$
where **z** is spatial CNN features
$T_W(.)$ is the transformer encoder

Decoder: $y_t = T_D(\mathbf{y}_{0:t-1}, \mathbf{c})$
where $T_D(.)$ is the transformer decoder



The Transformer encoder block

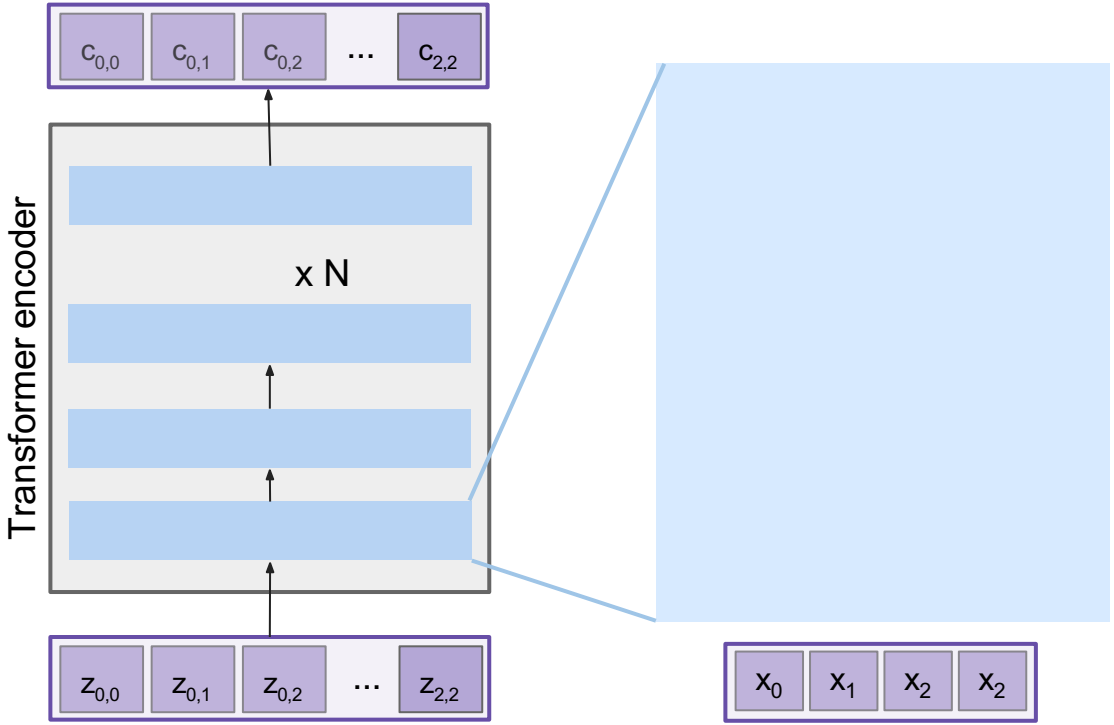


Made up of N encoder blocks.

In vaswani et al. N = 6, D_q= 512

Vaswani et al, “Attention is all you need”, NeurIPS 2017

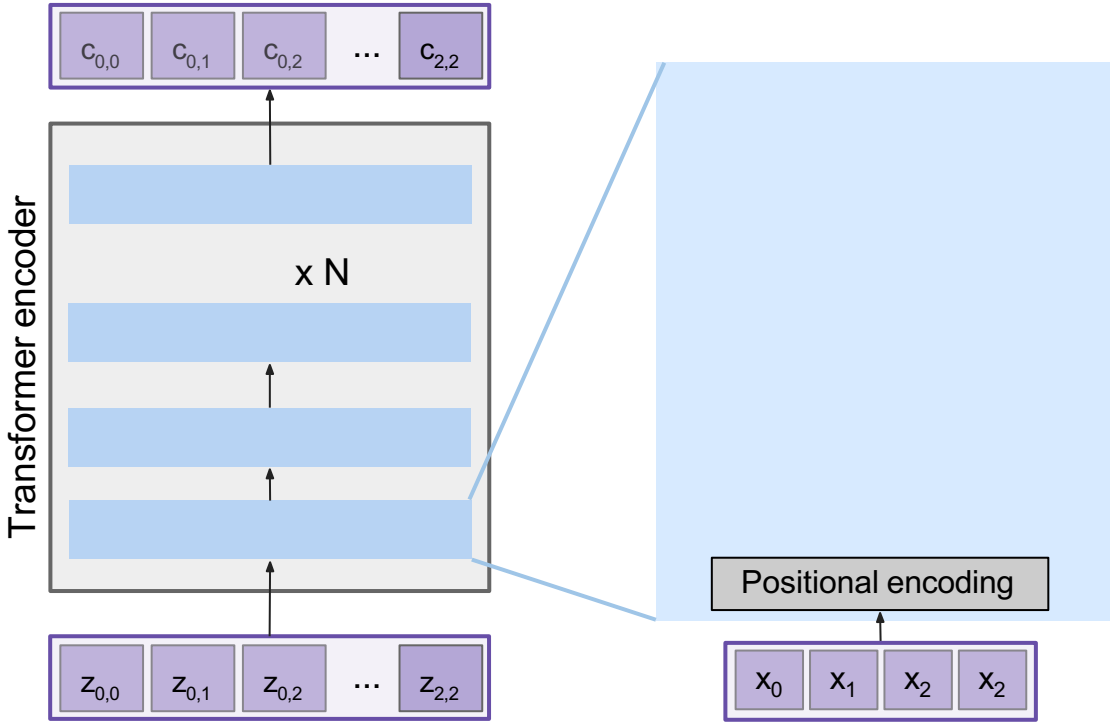
The Transformer encoder block



Let's dive into one encoder block

Vaswani et al, “Attention is all you need”, NeurIPS 2017

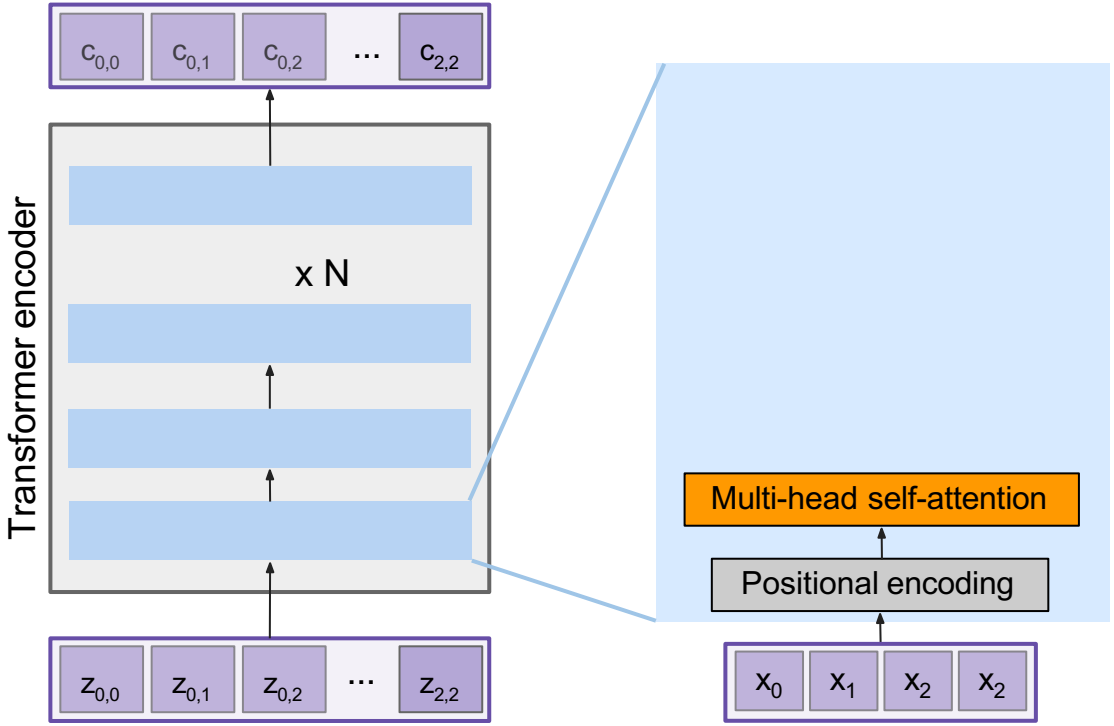
The Transformer encoder block



Add positional encoding

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block

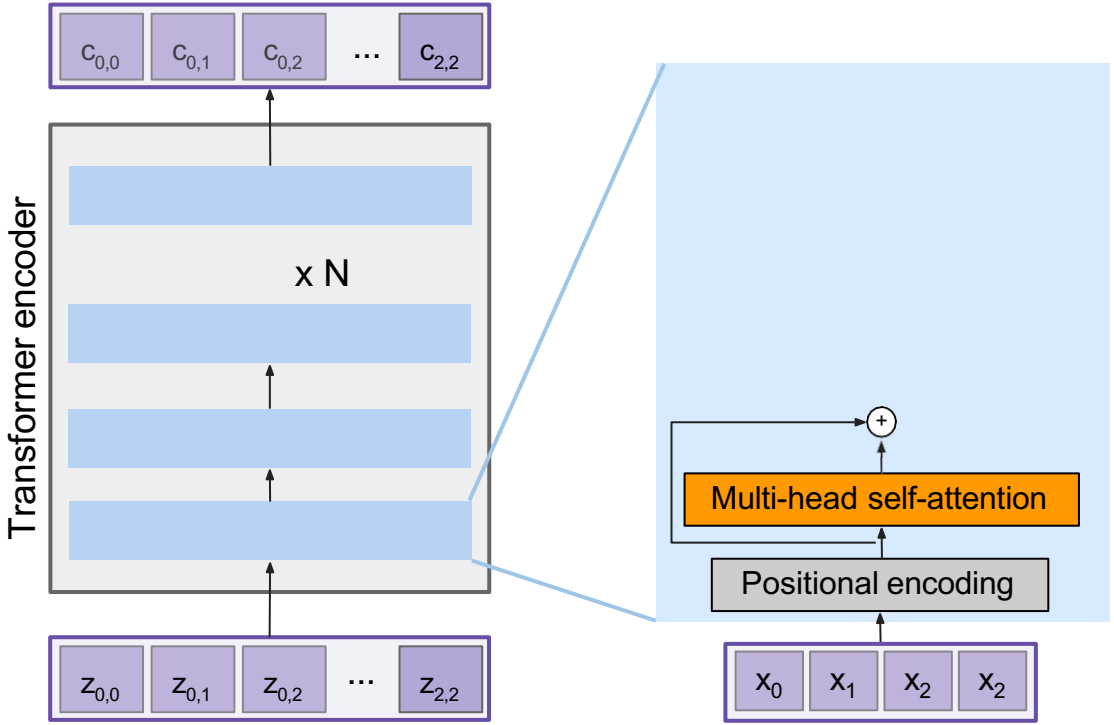


Attention attends over all the vectors

Add positional encoding

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block



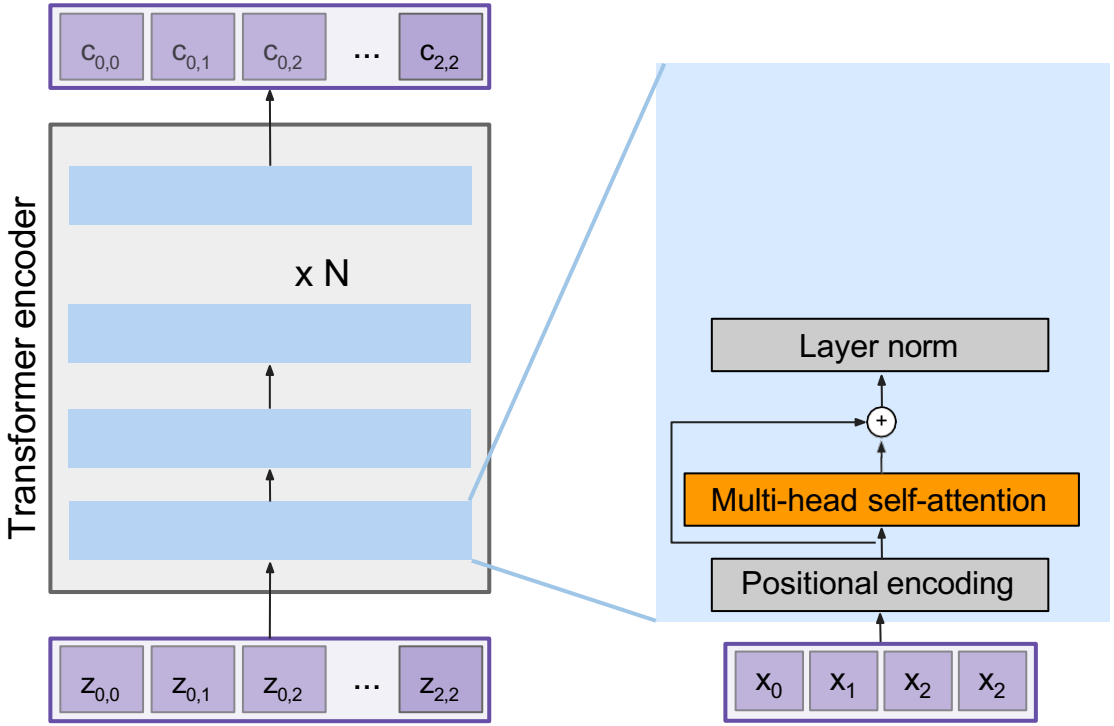
Residual connection

Attention attends over all the vectors

Add positional encoding

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block



LayerNorm over each vector individually

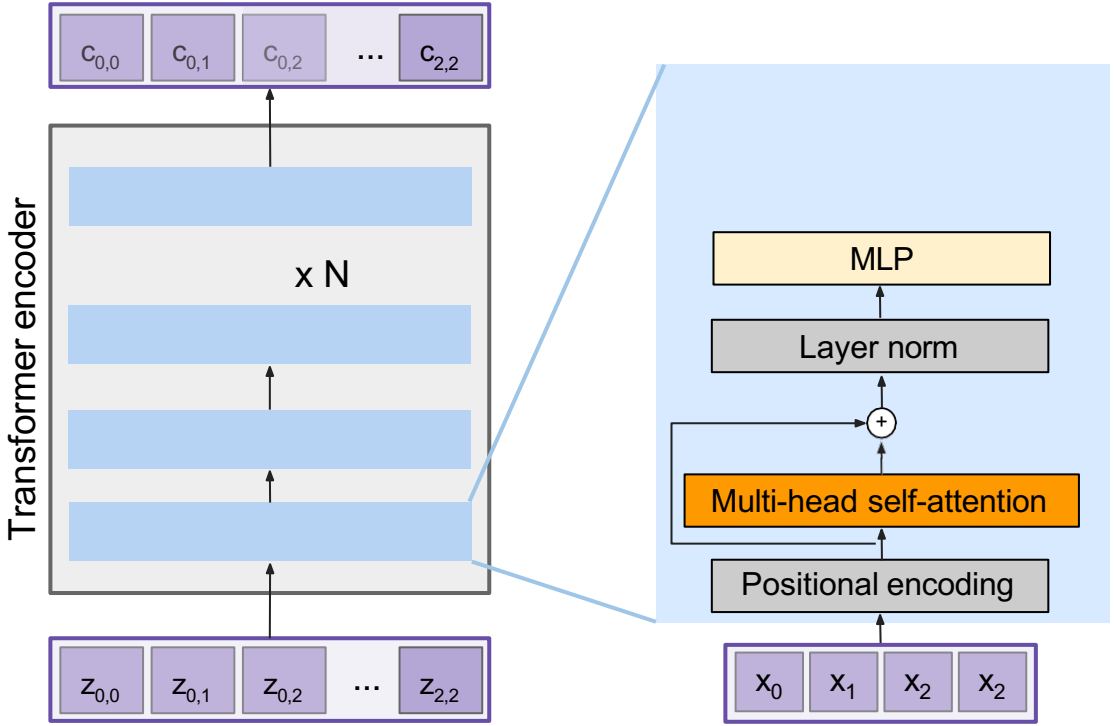
Residual connection

Attention attends over all the vectors

Add positional encoding

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block



MLP over each vector individually

LayerNorm over each vector individually

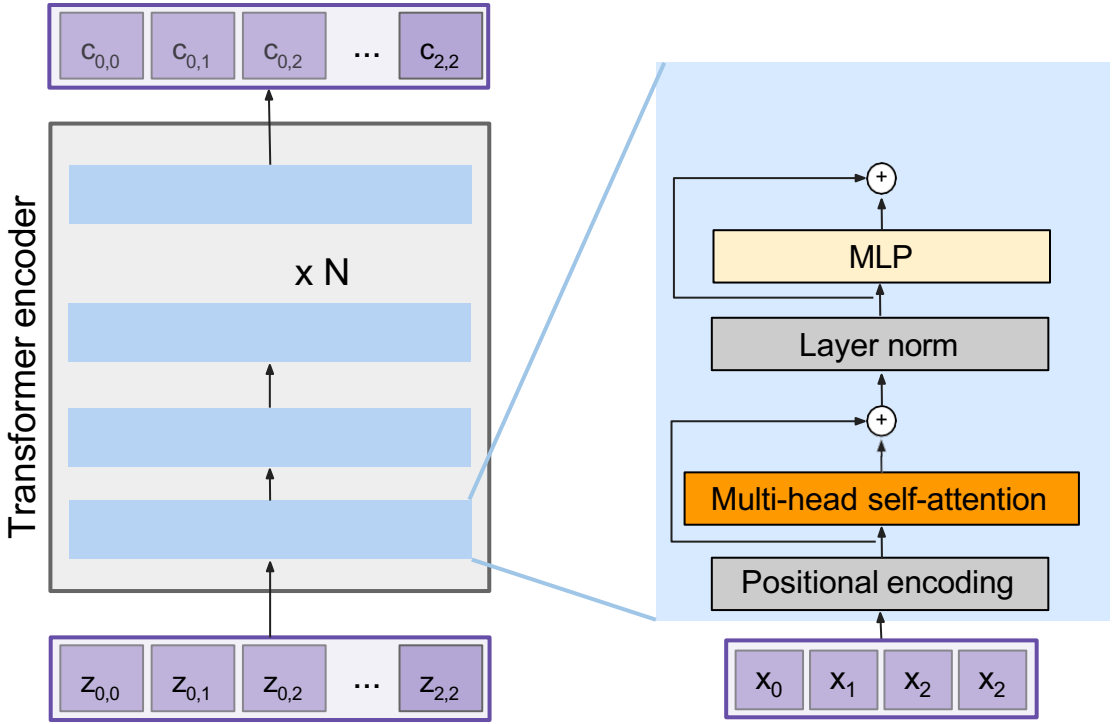
Residual connection

Attention attends over all the vectors

Add positional encoding

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block



Residual connection

MLP over each vector individually

LayerNorm over each vector individually

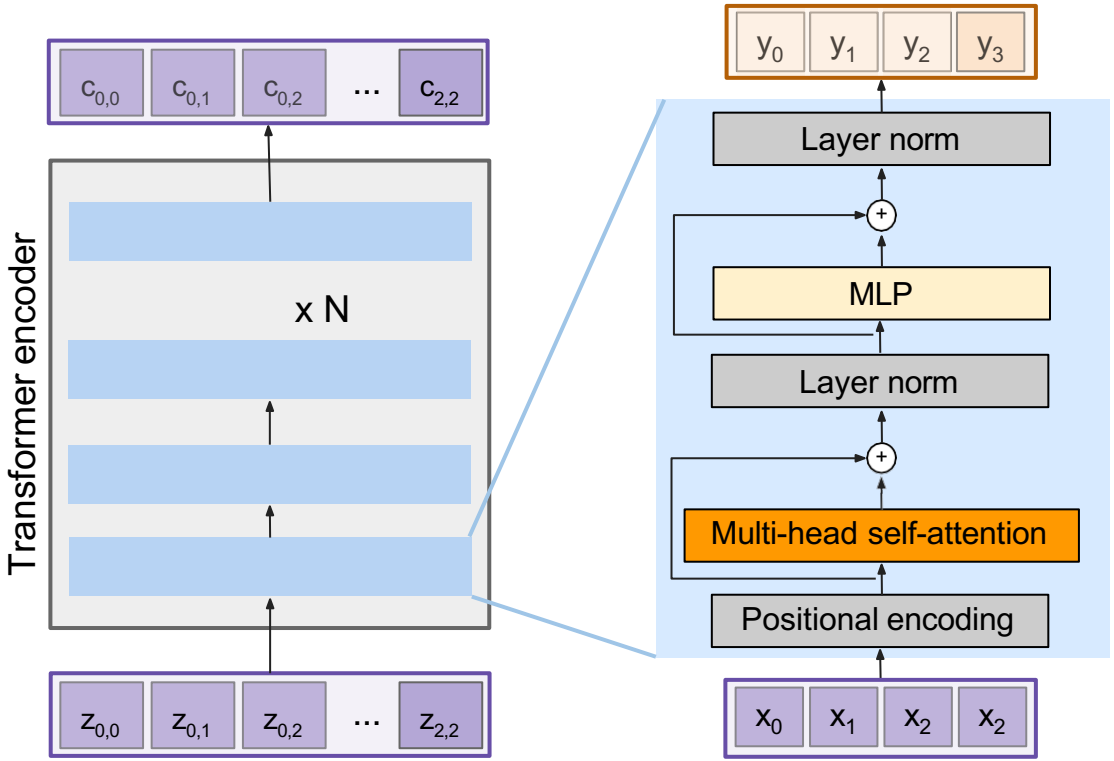
Residual connection

Attention attends over all the vectors

Add positional encoding

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block



Transformer Encoder Block:

Inputs: Set of vectors **x**
Outputs: Set of vectors **y**

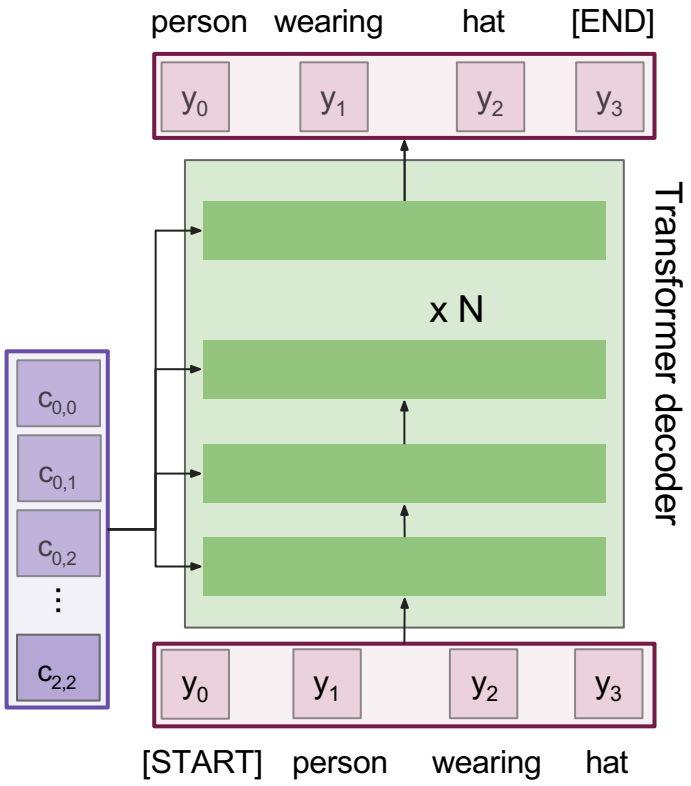
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer decoder block

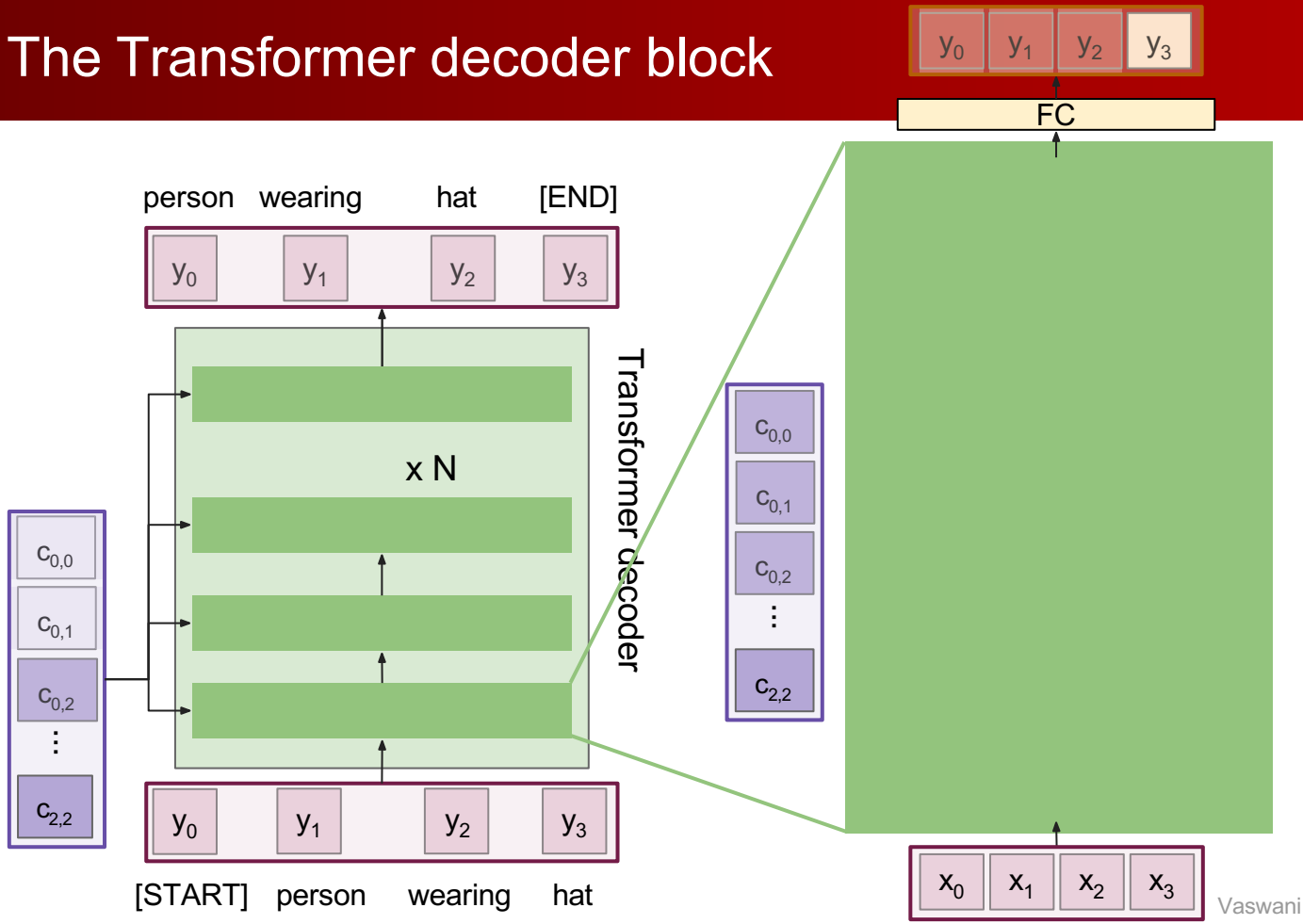


Made up of N decoder blocks.

In vaswani et al. N = 6, D_q= 512

Vaswani et al, “Attention is all you need”, NeurIPS 2017

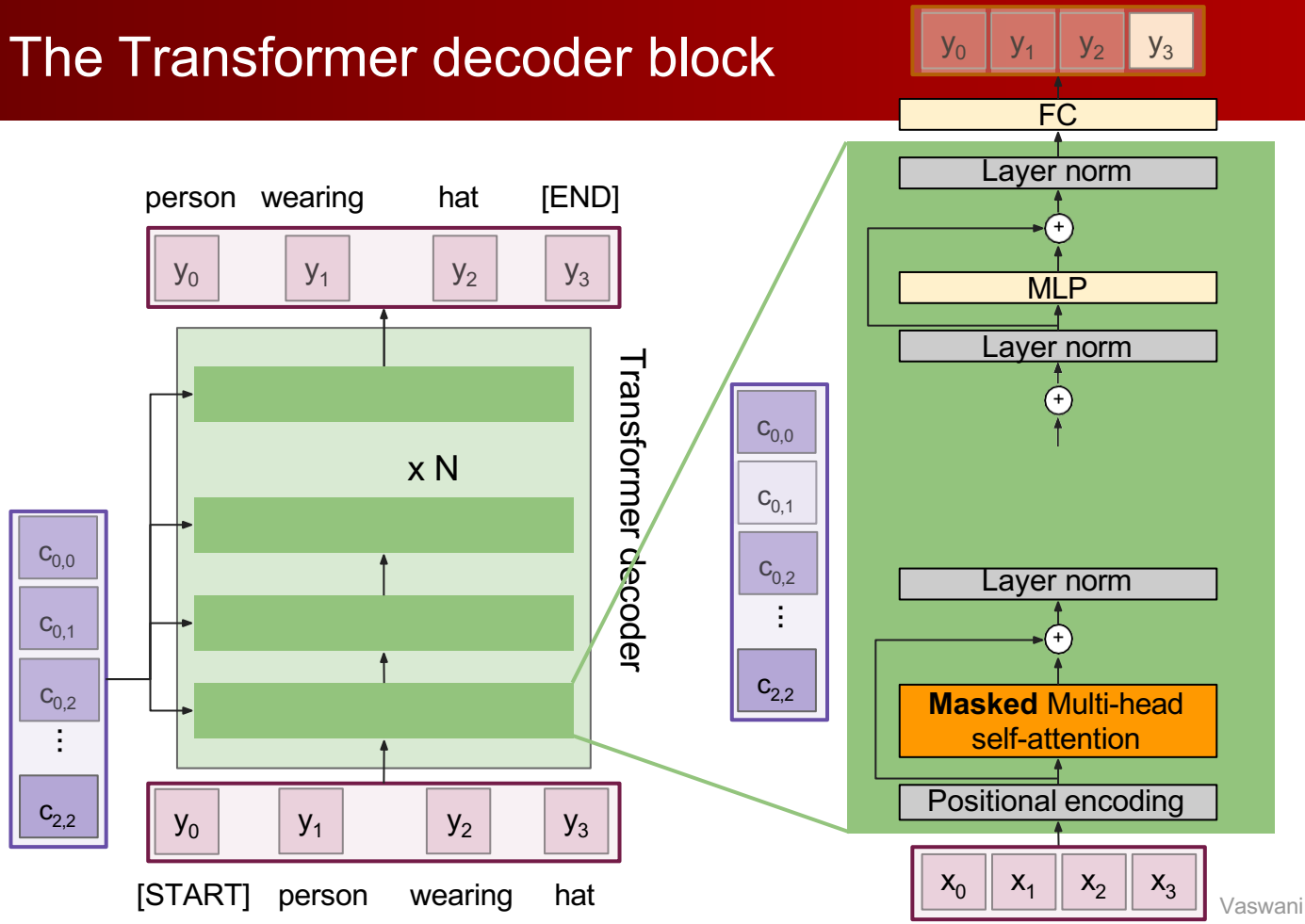
The Transformer decoder block



Let's dive into the transformer decoder block

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

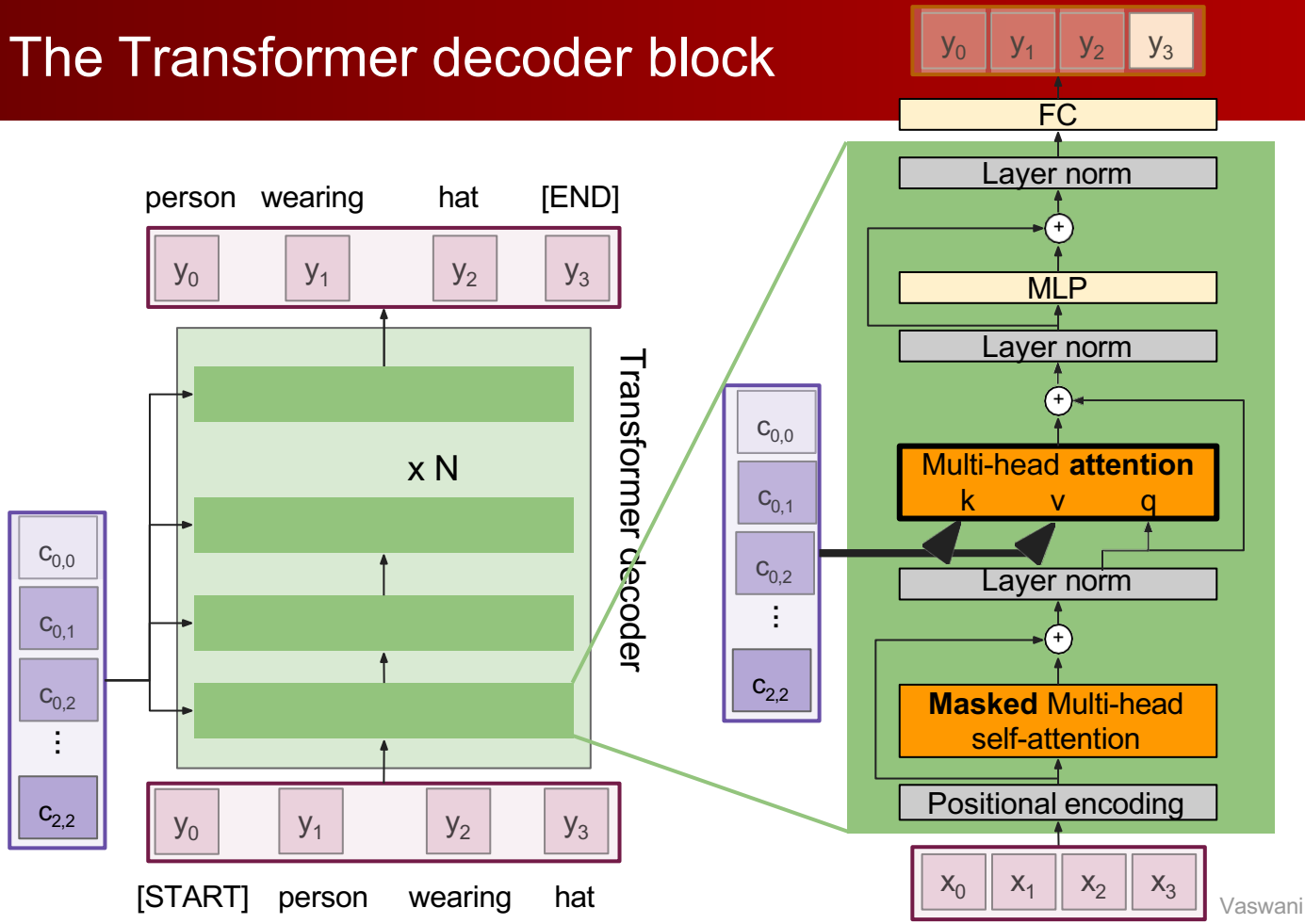
The Transformer decoder block



Most of the network is the same the transformer encoder.

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer decoder block

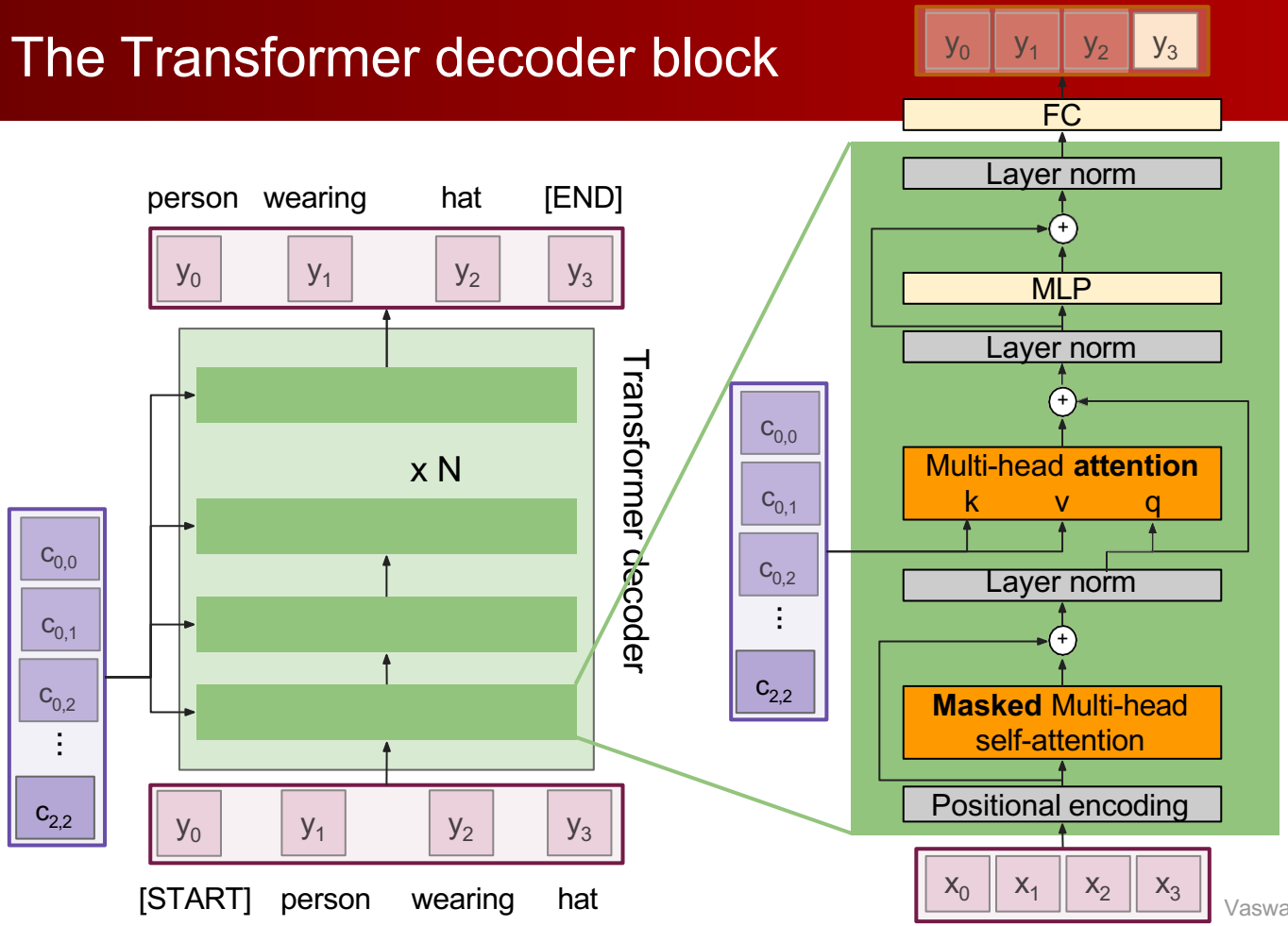


Multi-head attention block attends over the transformer encoder outputs.

For image captioning, this is how we inject image features into the decoder.

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

The Transformer decoder block



Transformer Decoder Block:

Inputs: Set of vectors **x** and Set of context vectors **c.**
Outputs: Set of vectors **y.**

Masked Self-attention only interacts with past inputs.

Multi-head attention block is NOT self-attention. It attends over encoder outputs.

Highly scalable, highly parallelizable, but high memory usage.

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

Image Captioning using transformers

- **No recurrence at all**

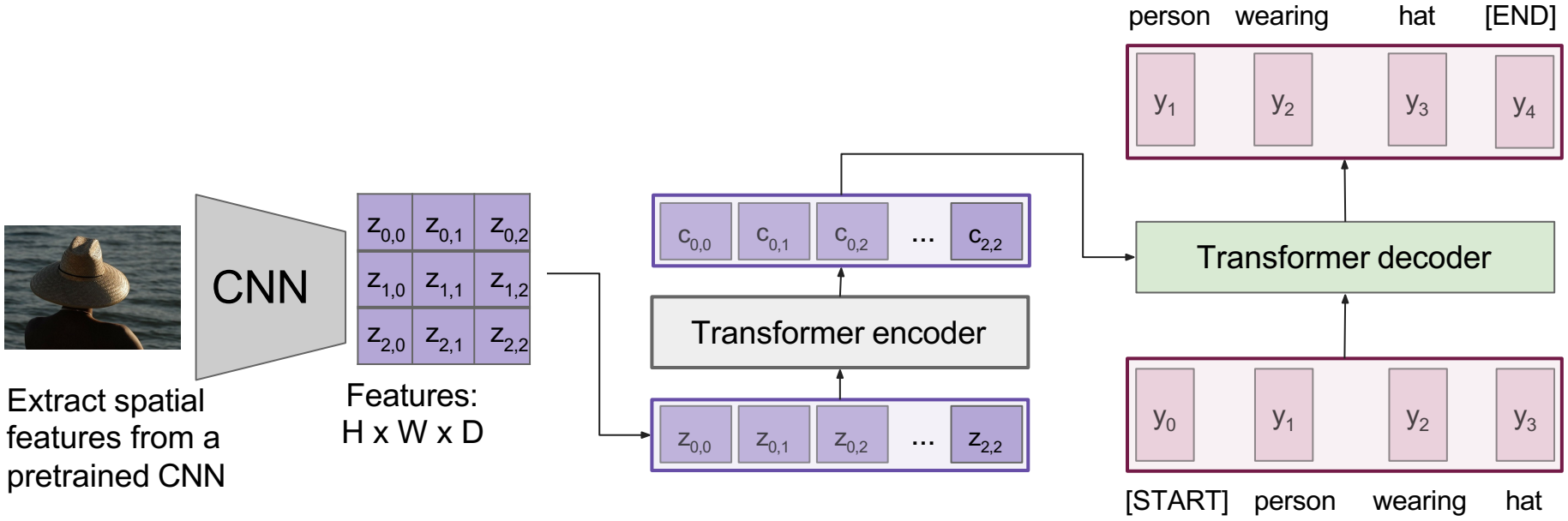


Image Captioning using transformers

- **Perhaps we don't need convolutions at all?**

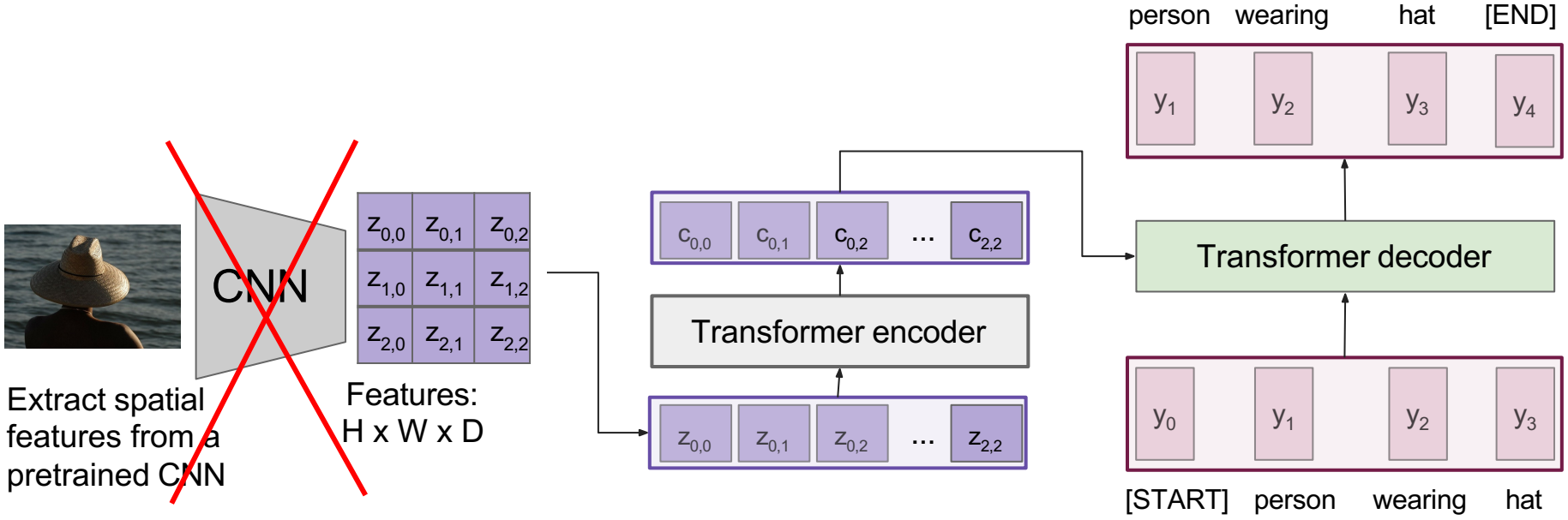
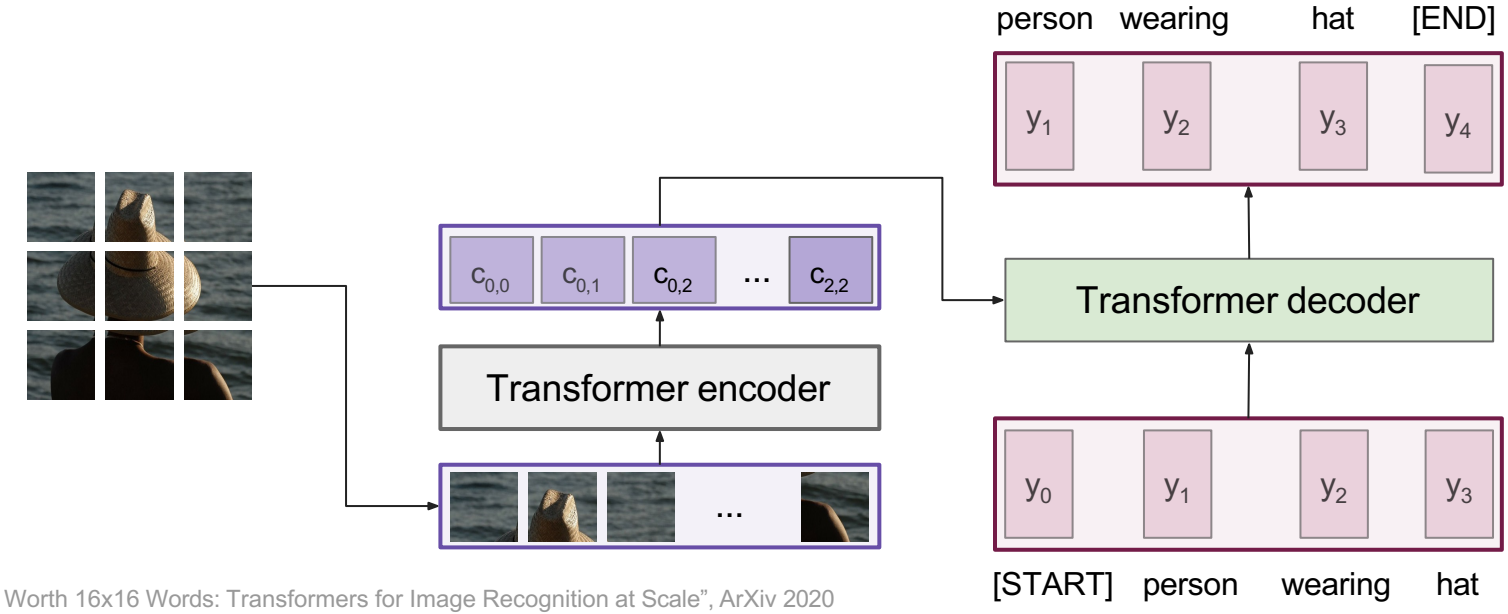


Image Captioning using ONLY transformers

- **Transformers from pixels to language**



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ArXiv 2020

Colab link to an implementation of vision transformers

Vision Transformers vs. ResNets

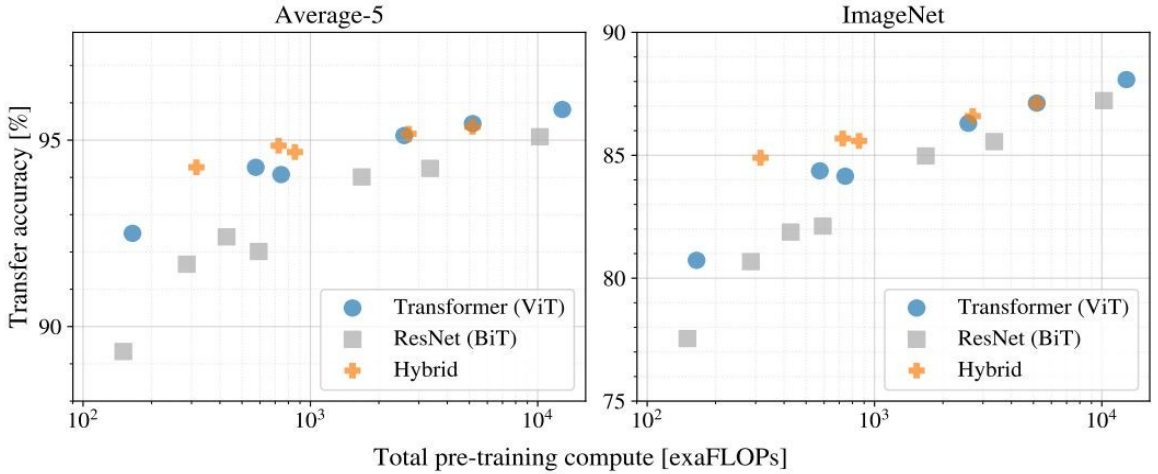
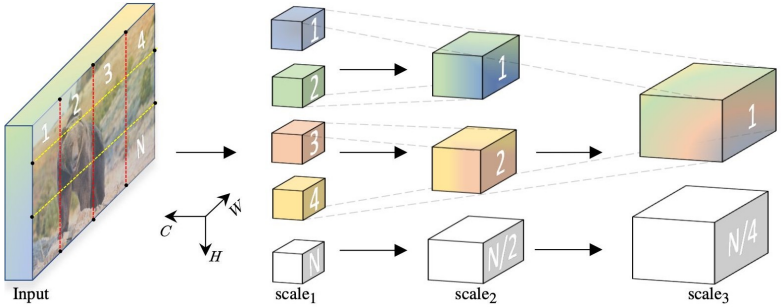


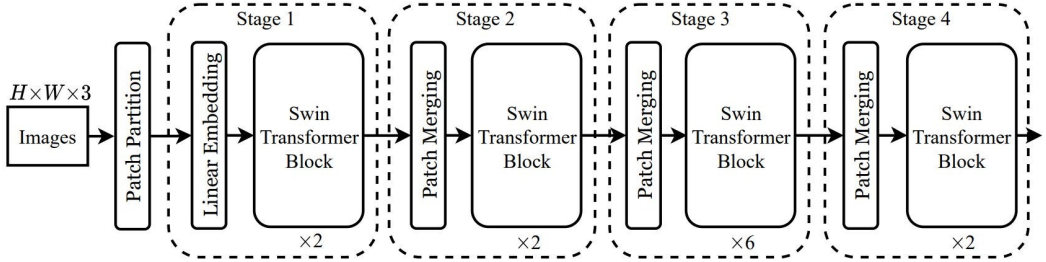
Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ArXiv 2020
Colab link to an implementation of vision transformers

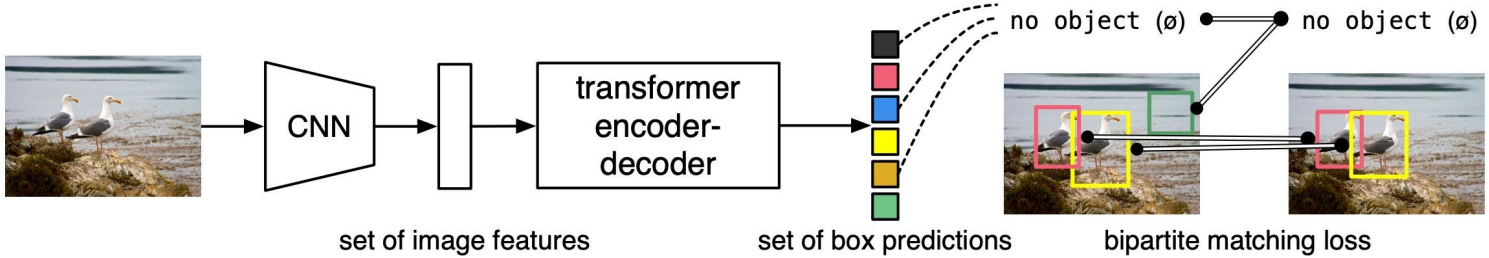
Vision Transformers



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

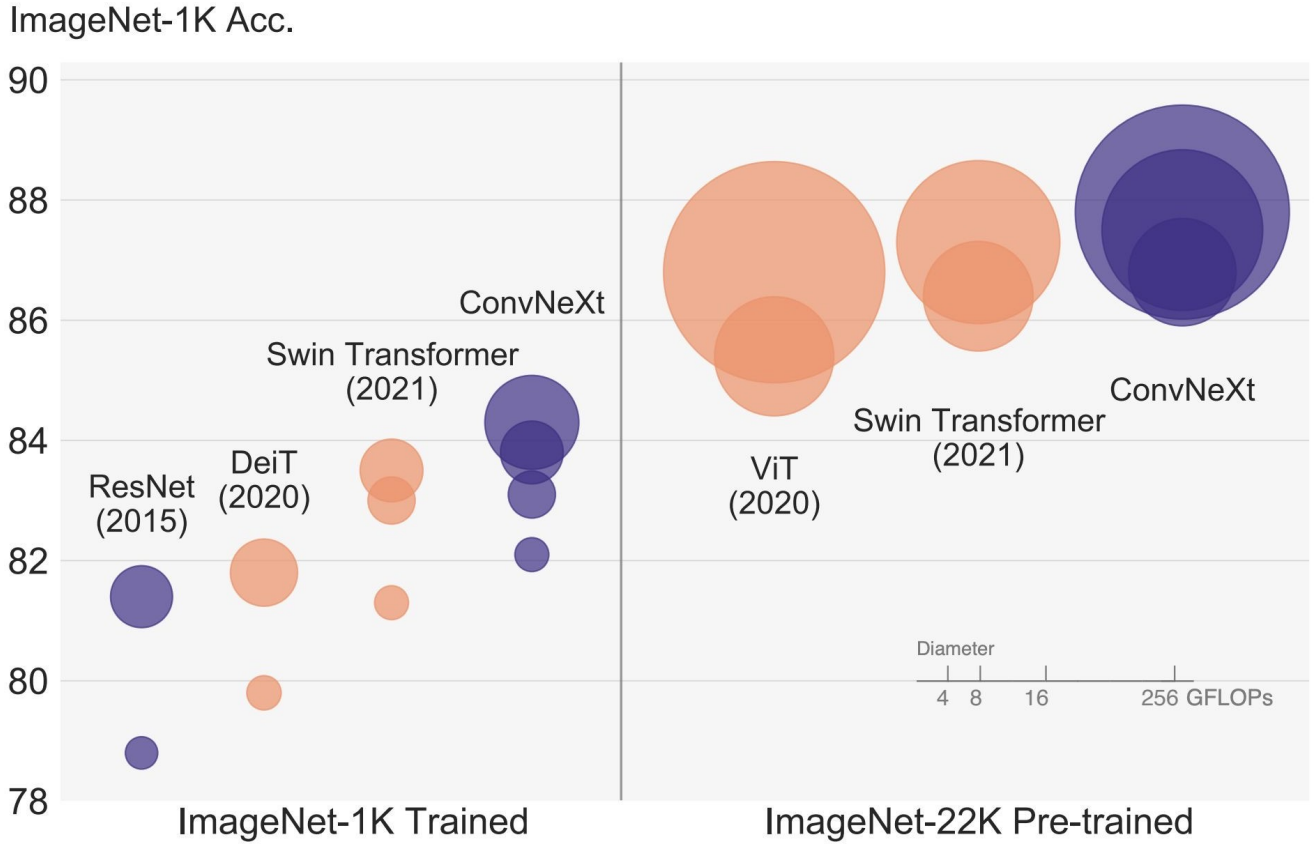


Liu et al, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”, CVPR 2021



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

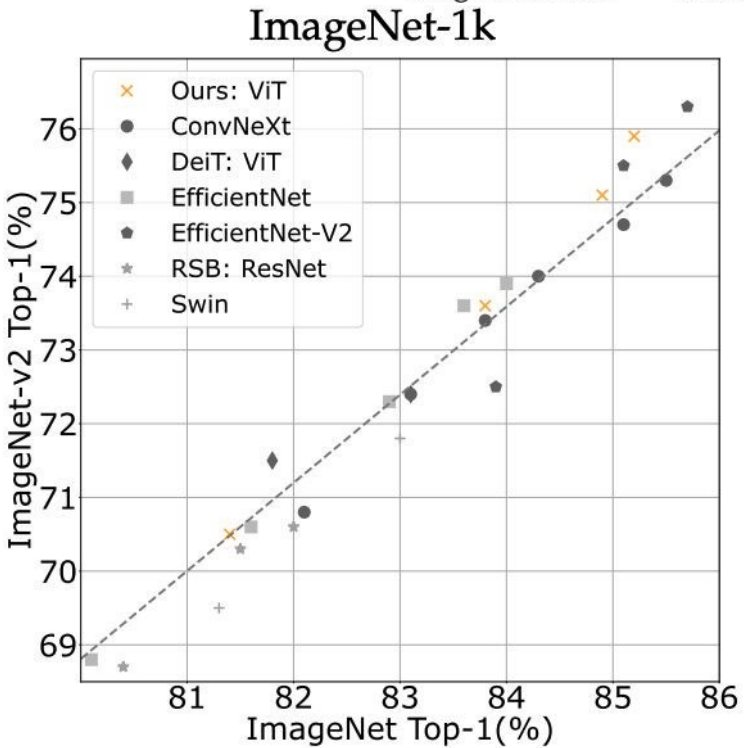
ConvNets strike back!

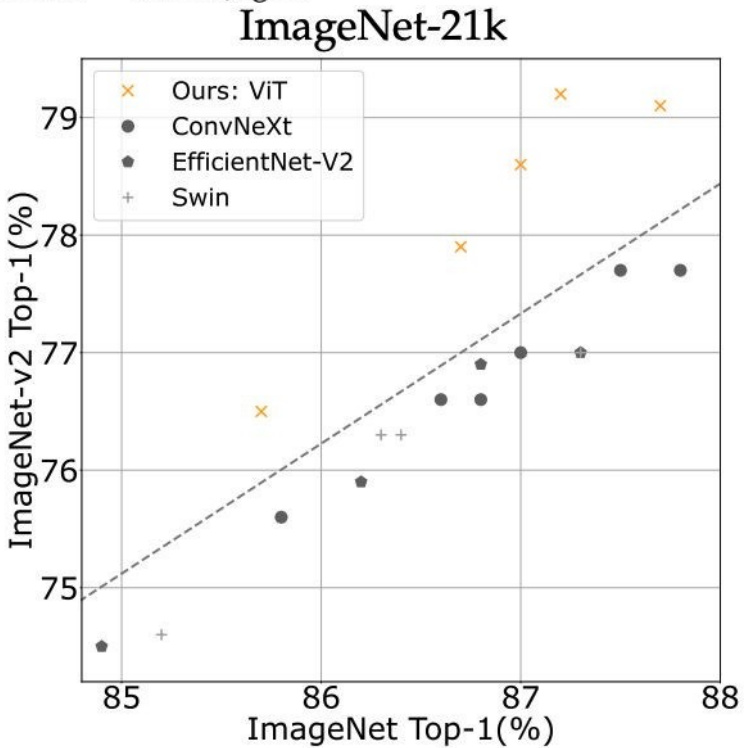


A ConvNet for the 2020s. Liu et al. CVPR 2022

DeiT III: Revenge of the ViT

Hugo Touvron*,† Matthieu Cord† Hervé Jégou*

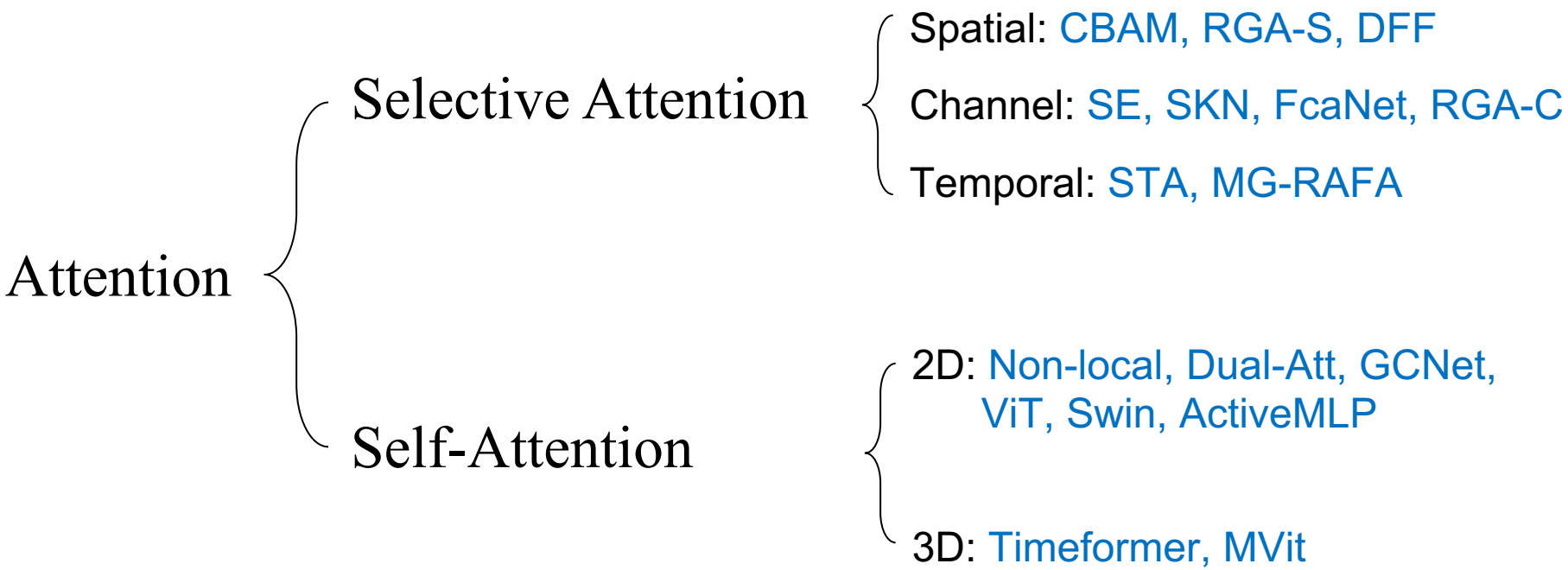




Summary

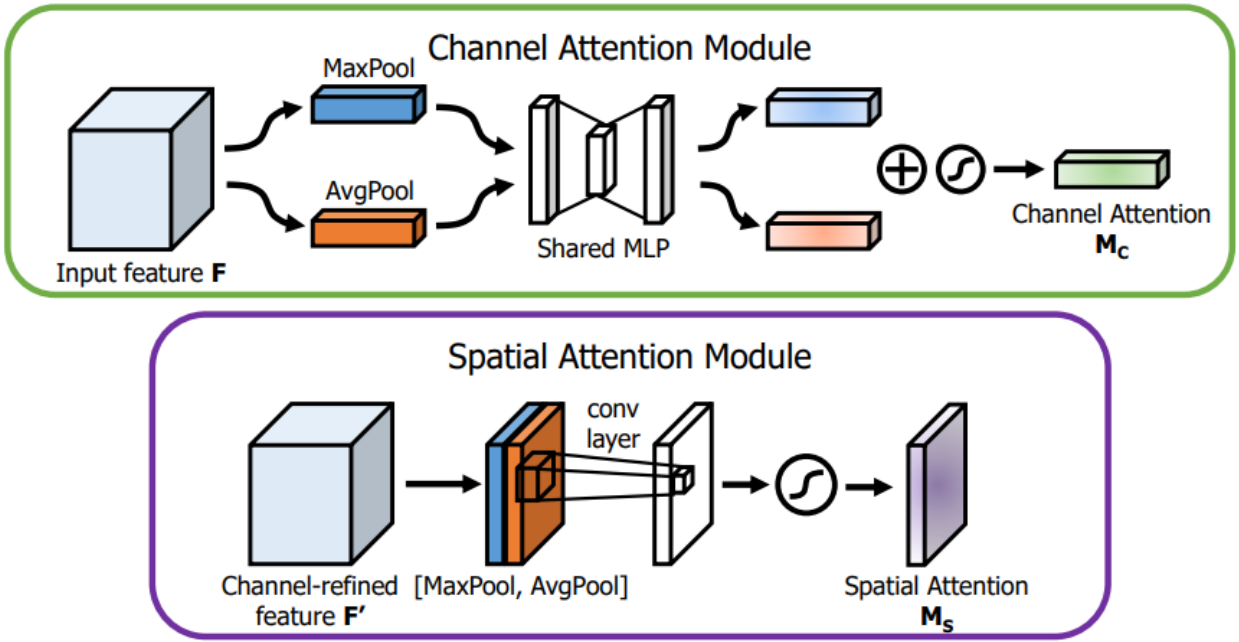
- Adding **attention** to RNNs allows them to "attend" to different parts of the input at every time step
- The **general attention layer** is a new type of layer that can be used to design new neural network architectures
- **Transformers** are a type of layer that uses **self-attention** and layer norm.
  - It is highly **scalable** and highly **parallelizable**
  - **Faster** training, **larger** models, **better** performance across vision and language tasks
  - They are quickly replacing RNNs, LSTMs, and may(?) even replace convolutions.

Selective Attention (CV Domain)



Selective Attention (CBAM)

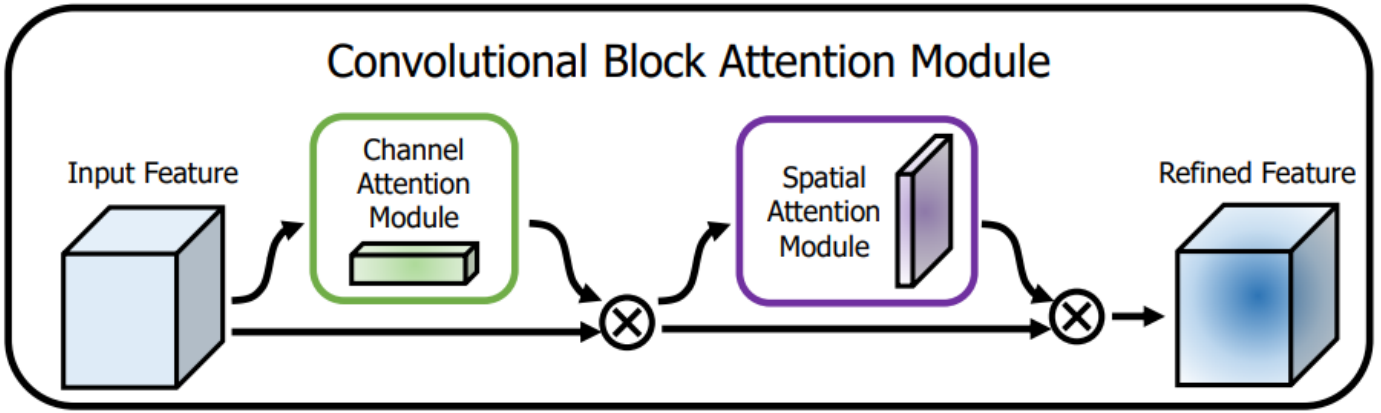
- Cbam: Convolutional block attention module



Woo S, Park J, Lee J Y, et al. “Cbam: Convolutional block attention module”. In ECCV, 2018.

Selective Attention (CBAM)

- CBAM

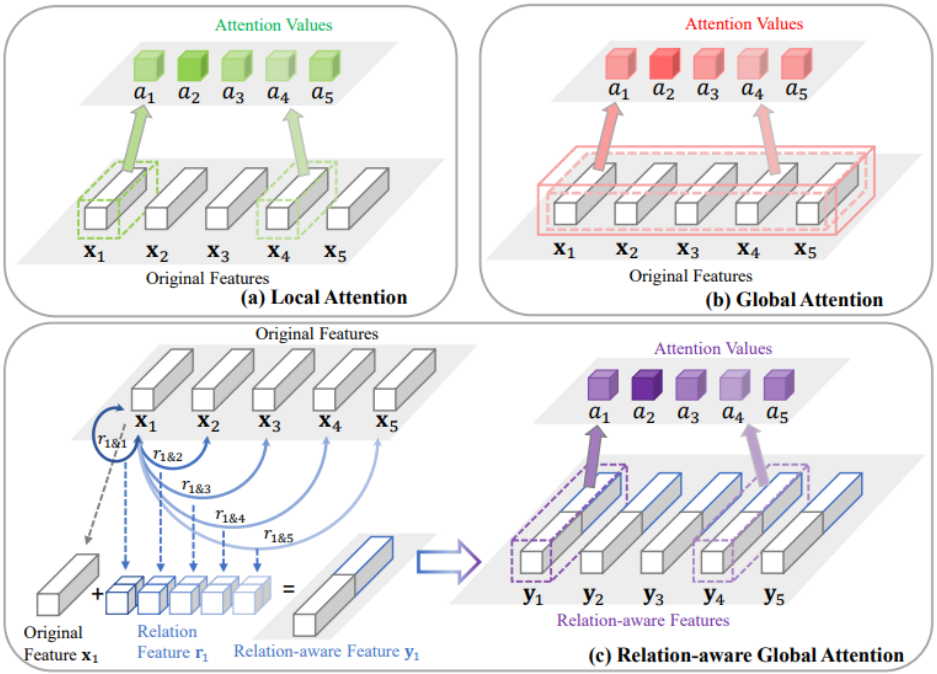


$$\mathbf{M_c}(\mathbf{F}) = \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F})))$$
$$= \sigma(\mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{avg}})) + \mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{max}}))),$$

$$\mathbf{M_s}(\mathbf{F}) = \sigma(f^{7\times7}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})]))$$
$$= \sigma(f^{7\times7}([\mathbf{F^s_{avg}}; \mathbf{F^s_{max}}])),$$

Selective Attention (RGA)

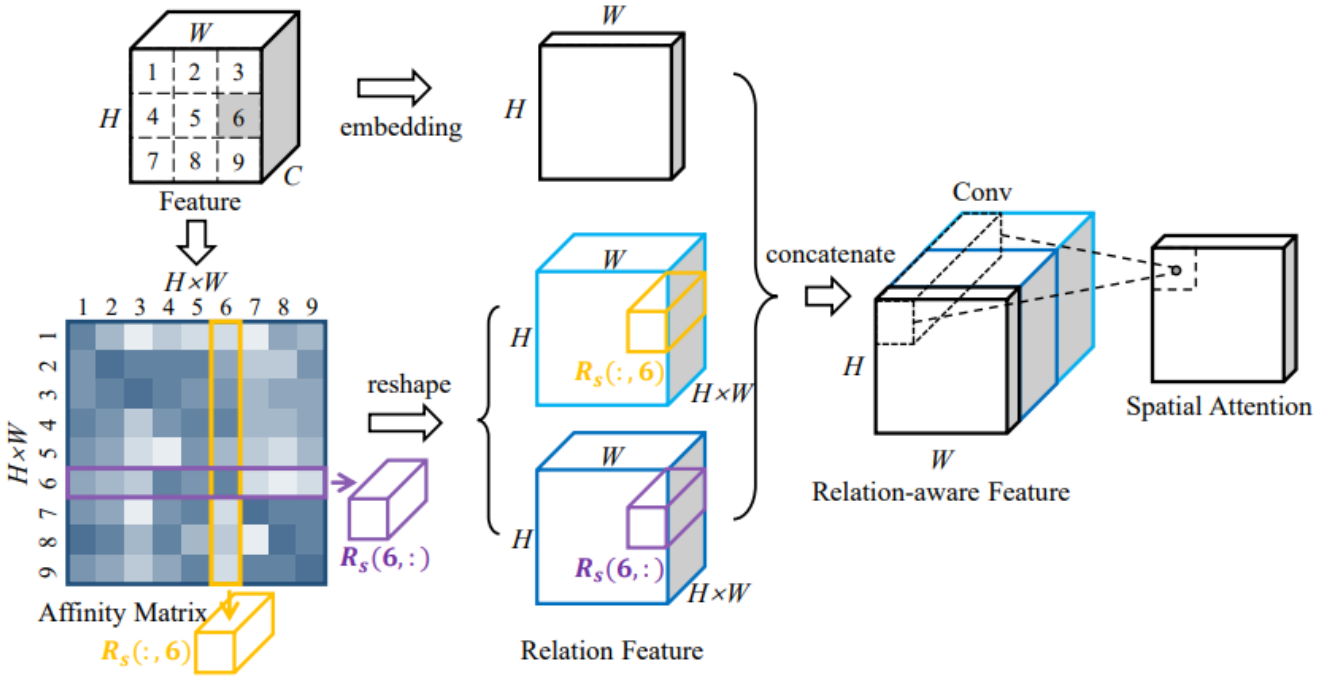
- Relation-Aware Global Attention



Zhizheng Zhang, Cuiling Lan, Wenjun Zeng, Xin Jin, Zhibo Chen, "Relation Aware Global Attention for Person Re-identification", In CVPR, 2020.

Selective Attention (Spatial RGA)

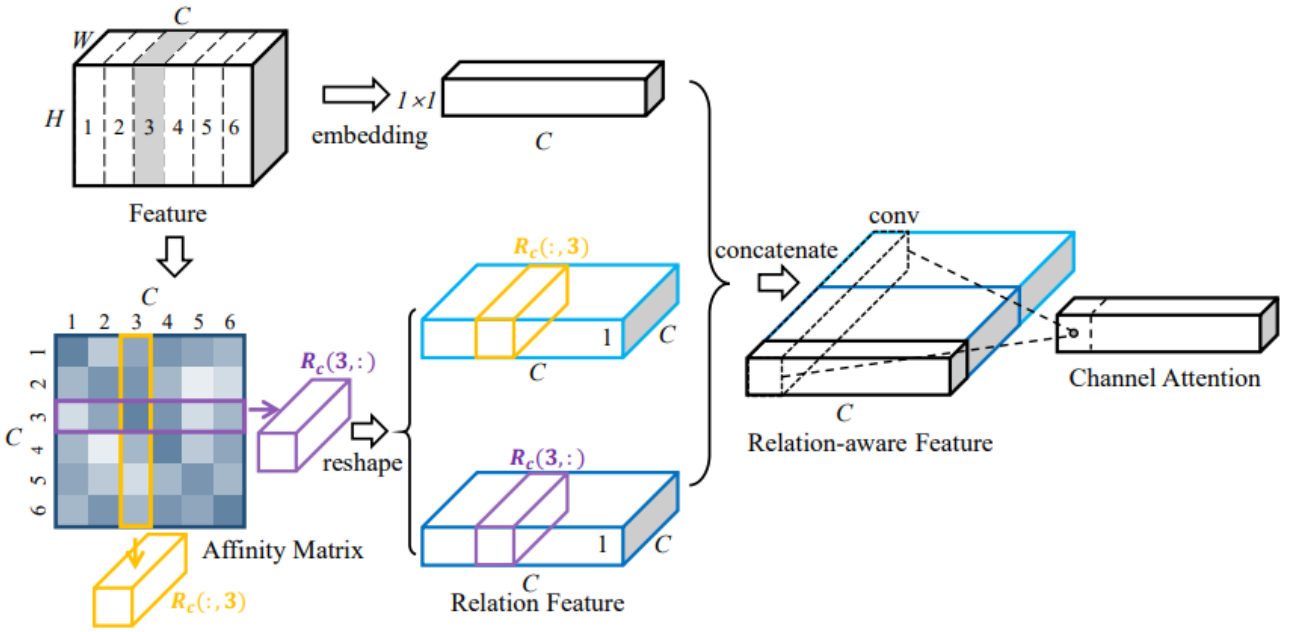
- Spatial RGA (RGA-S)



(a) Spatial Relation-Aware Global Attention

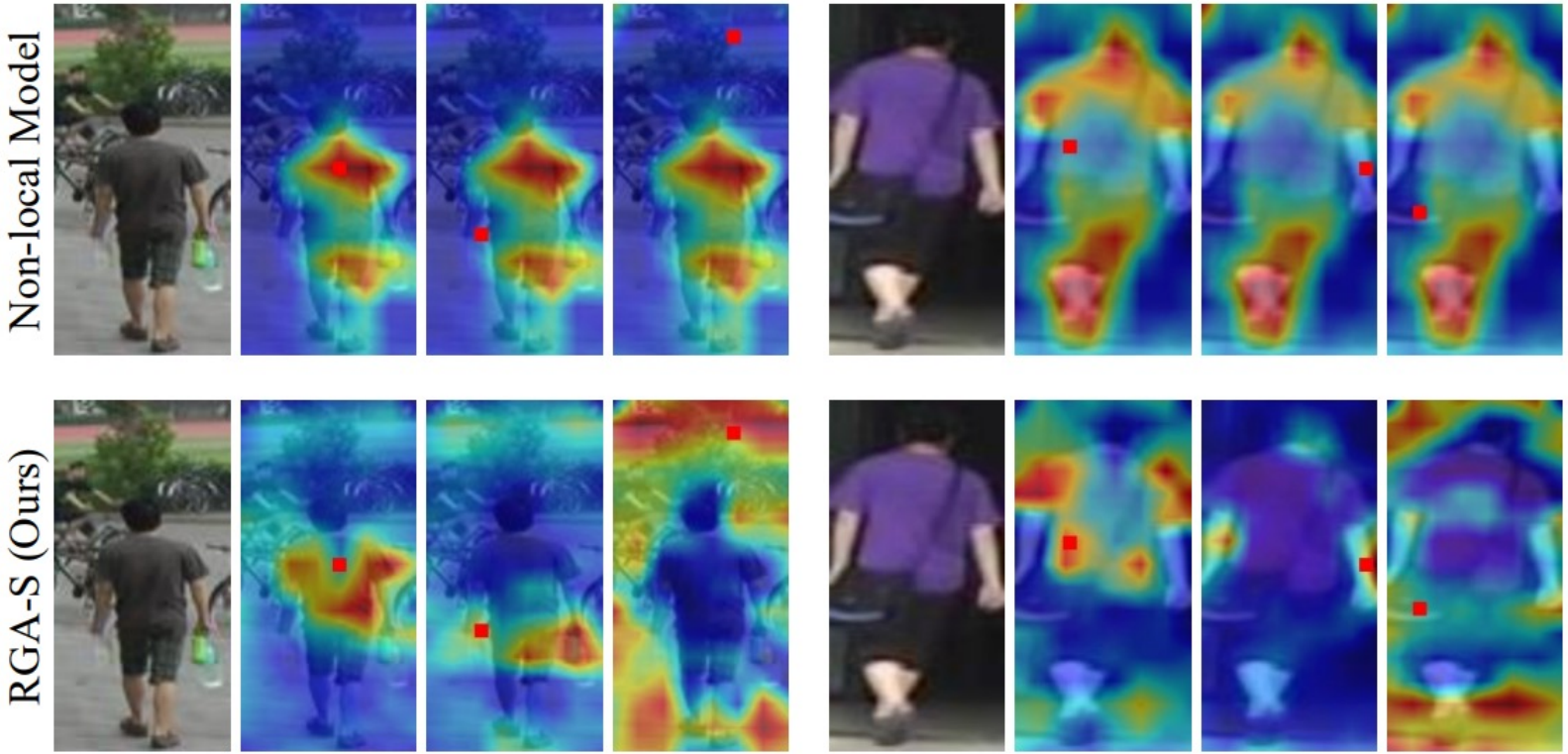
Selective Attention (Spatial RGA)

- Channel RGA (RGA-C)



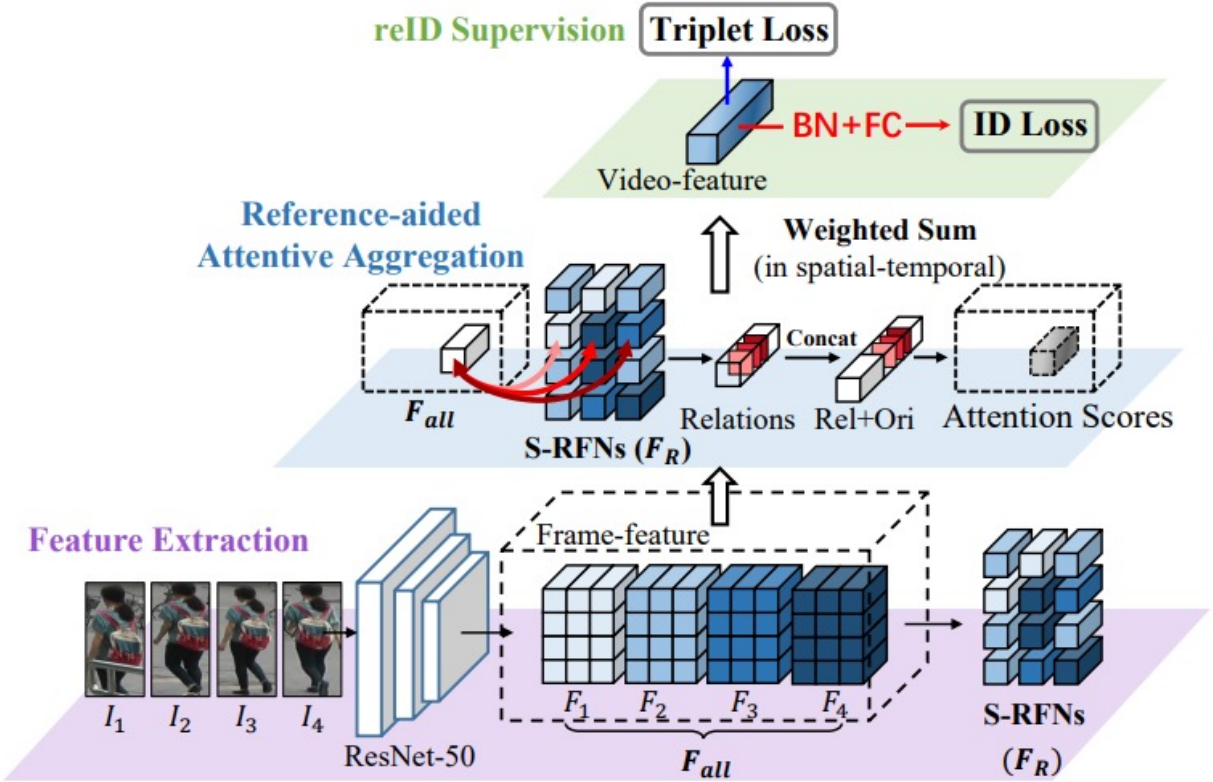
(b) Channel Relation-Aware Global Attention

Selective Attention (Non-local v.s. RGA)



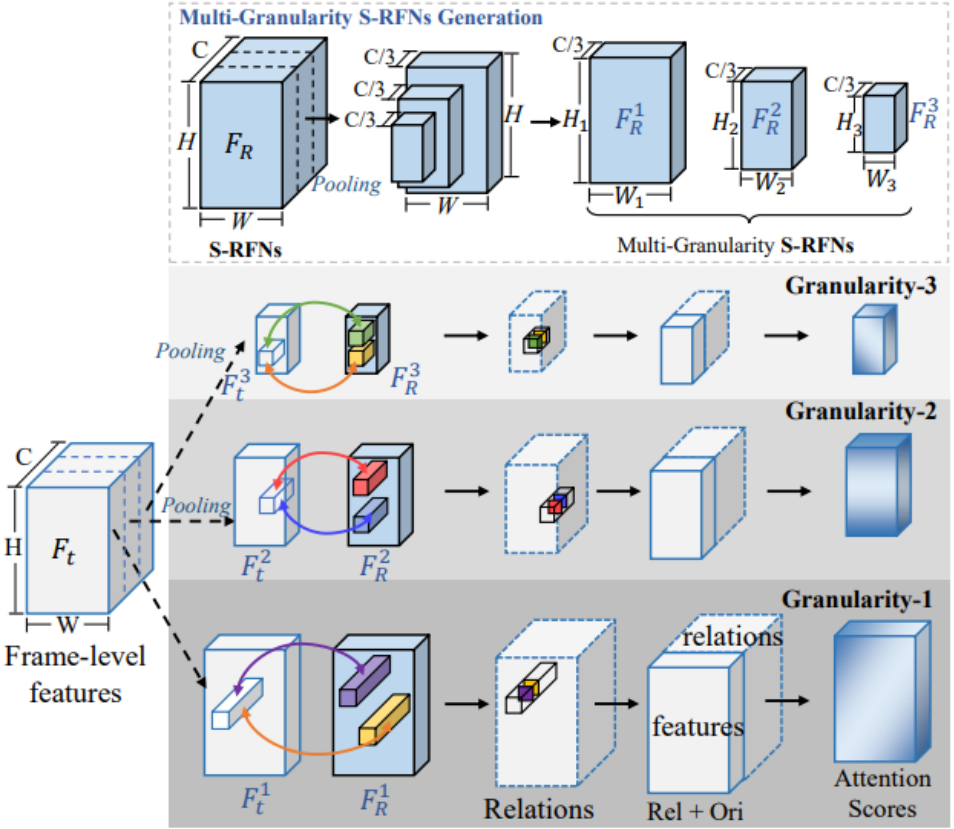
Selective Attention (Video RGA)

- Framework of MG-RAFA



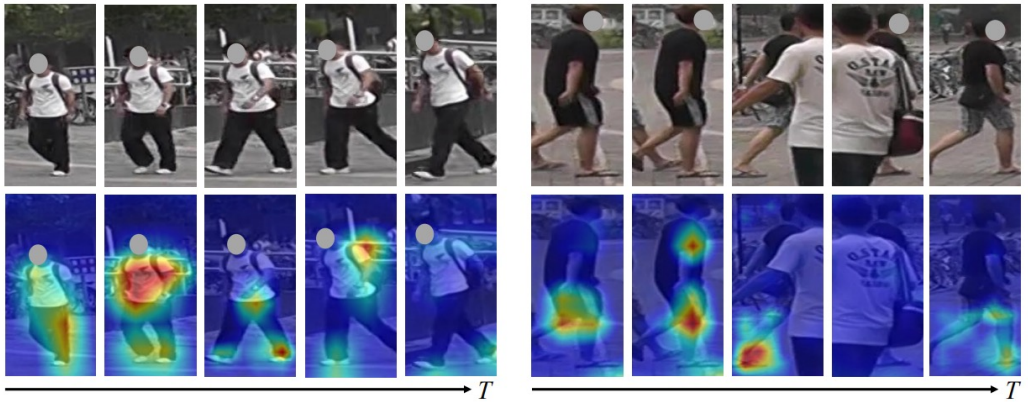
Selective Attention (Video RGA)

- Module of MG-RAFA

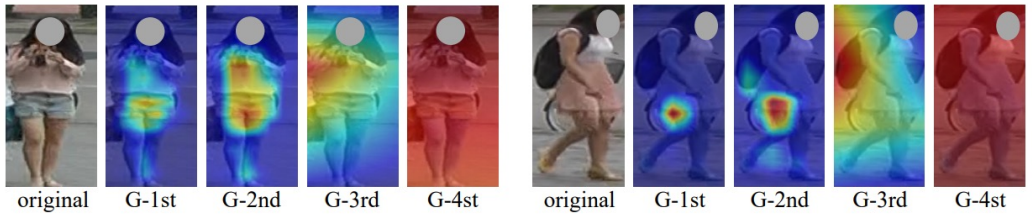


Selective Attention (Video RGA)

- Visualization (Video-based Person ReID)



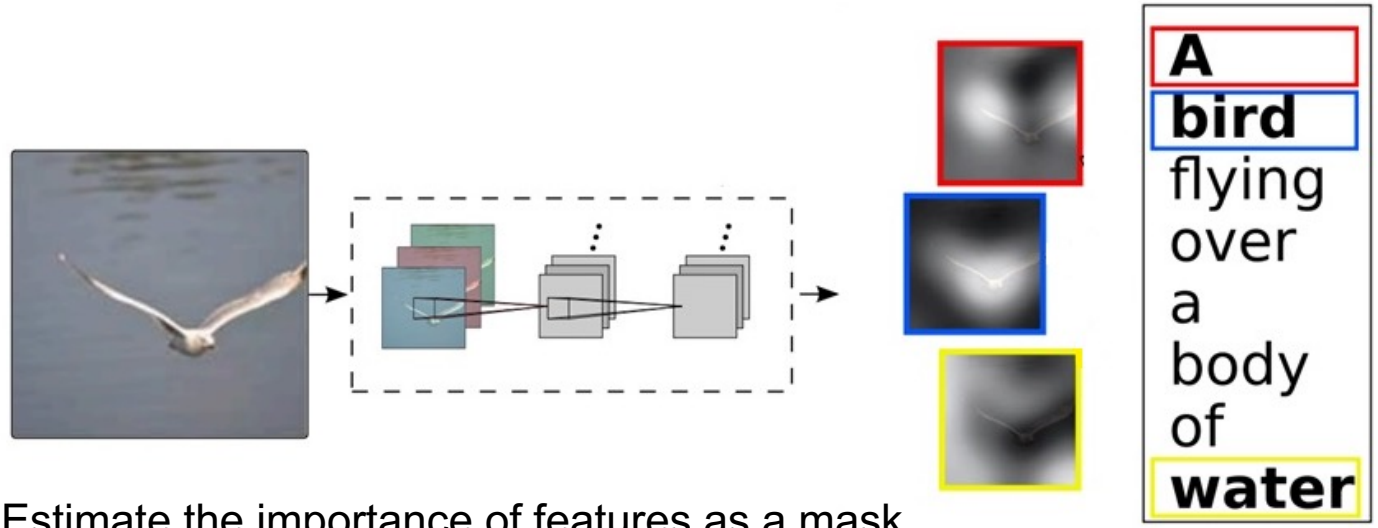
(a) Visualization on different frames at the 2^{nd} granularity.



(b) Visualization of different granularities at a given time.

Summary (Self-attention v.s. Selective Attention)

- What is the core idea of selective attention?
  - ➢ Explicitly enhance importance features while suppressing harmful ones

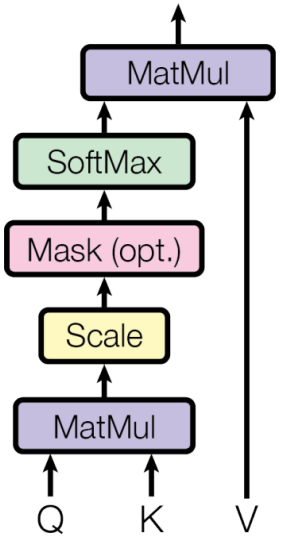


Estimate the importance of features as a mask

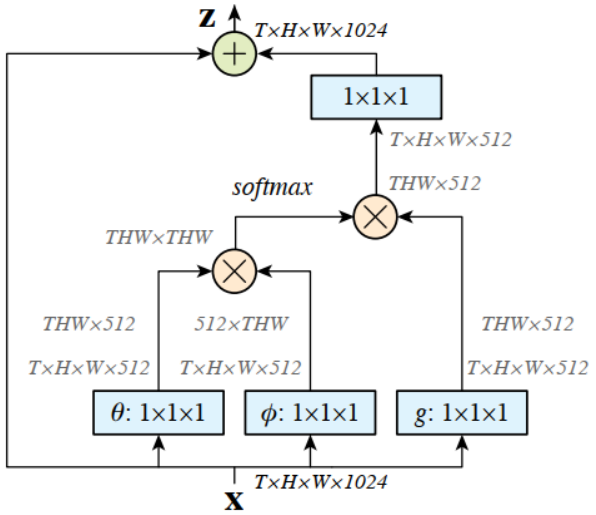
[Woo, et al. 2018]

Summary (Self-attention v.s. Selective Attention)

- What is the core idea of self-attention?
  - Non-local mean / affinity-based message passing



Self-attention in Language Models
[Vaswani, et al. 2017]

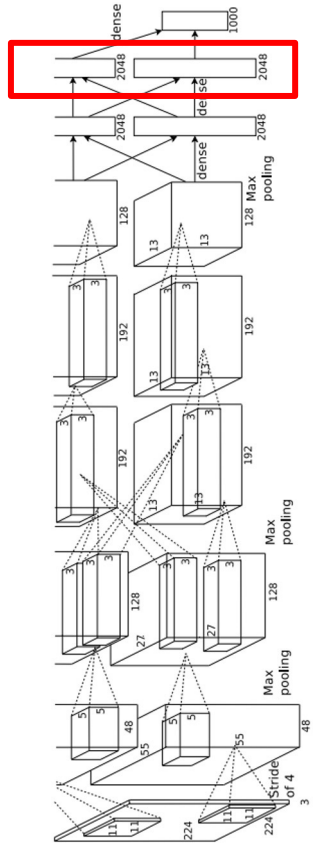


Self-attention in Vision Models
[Wang, et al. 2018]

Self-Supervised Learning (Pre-training)

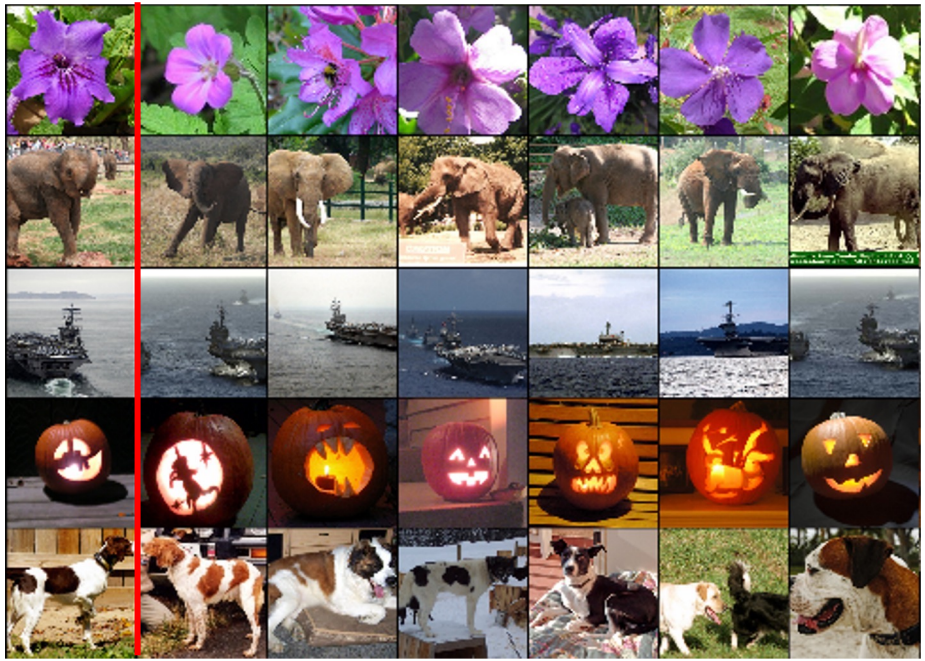
Learned Representations

4096-dim vector

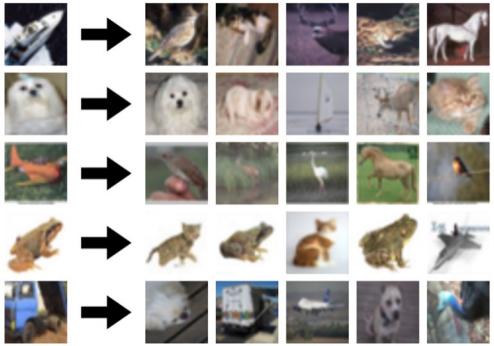


Test image

L2 Nearest neighbors in feature space



Recall: Nearest neighbors in pixel space



Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figures reproduced with permission.

Self-supervised Learning

- Both aim to learn from data without manual label annotation.
- Self-supervised learning methods solve “pretext” tasks that produce **good features** for downstream tasks.
  - Learn with supervised learning objectives, e.g., classification, regression.
  - Labels of these pretext tasks are generated *automatically*

Self-supervised pretext tasks

Example: learn to predict image transformations / complete corrupted images

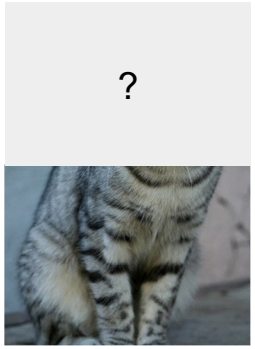
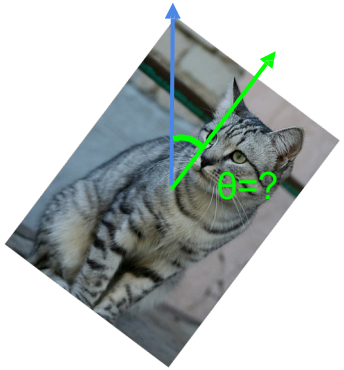
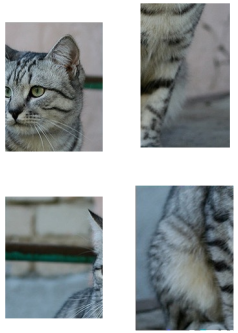


image completion



rotation prediction

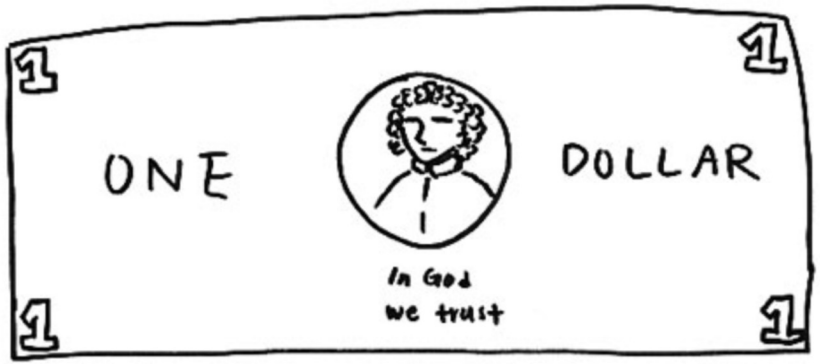


“jigsaw puzzle”

colorization

1. Solving the pretext tasks allow the model to learn good features.
2. We can automatically generate labels for the pretext tasks.

Generative vs. Self-supervised Learning



Left: Drawing of a dollar bill from memory. Right: Drawing subsequently made with a dollar bill present. Image source: Epstein, 2016

Learning to generate pixel-level details is often unnecessary; learn high-level semantic features with pretext tasks instead

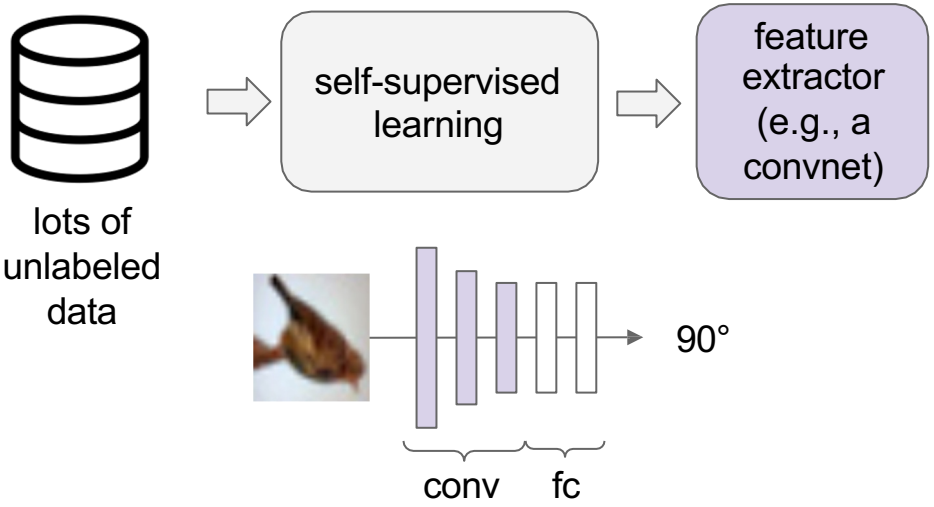
Source: Anand, 2020

How to evaluate a self-supervised learning method?

We usually don’t care about the performance of the self-supervised learning task, e.g., we don’t care if the model learns to predict image rotation perfectly.

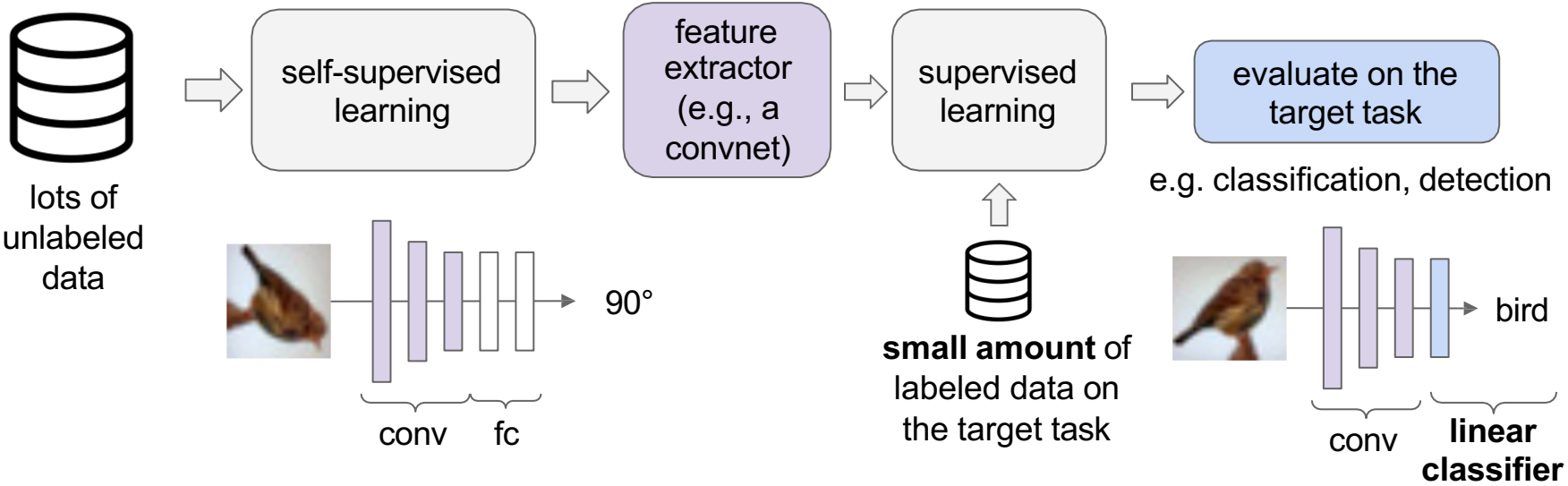
Evaluate the learned feature encoders on downstream *target tasks*

How to evaluate a self-supervised learning method?



1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

How to evaluate a self-supervised learning method?



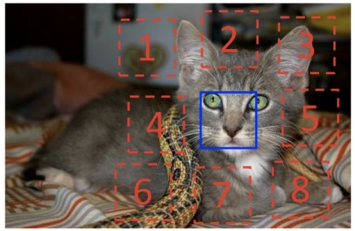
1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

2. Attach a shallow network on the feature extractor; train the shallow network on the target task with small amount of labeled data

Broader picture

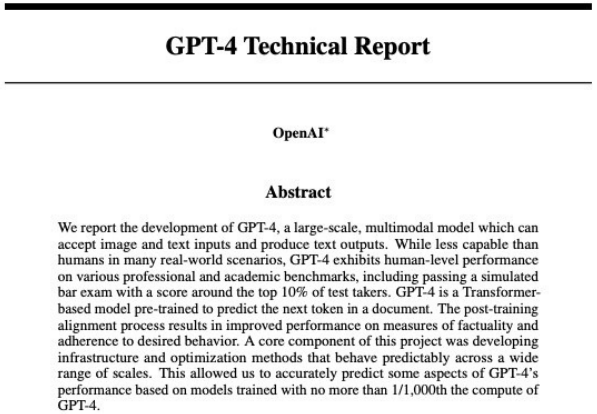
Today's lecture

computer vision



Doersch et al., 2015

language modeling



GPT-4 Technical Report

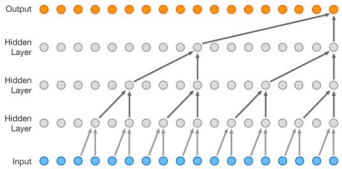
OpenAI*

Abstract

We report the development of GPT-4, a large-scale, multimodal model which can accept image and text inputs and produce text outputs. While less capable than humans in many real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-training alignment process results in improved performance on measures of factuality and adherence to desired behavior. A core component of this project was developing infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to accurately predict some aspects of GPT-4's performance based on models trained with no more than 1/1,000th the compute of GPT-4.

GPT-4 (OpenAI 2023)

speech synthesis



Wavenet (van den Oord et al., 2016)

robot / reinforcement learning

Dense Object Net (Florence and Manuelli et al., 2018)

. . .

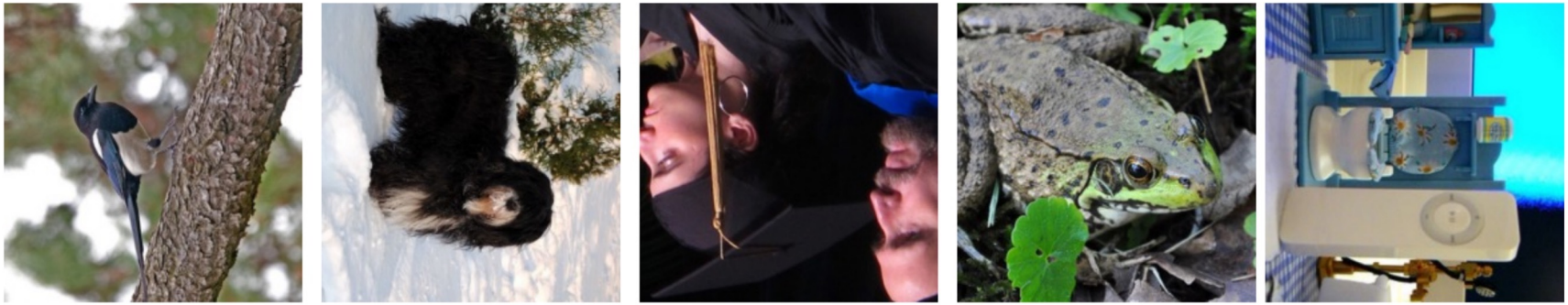
Today's Agenda

- Pretext tasks from image transformations
  - Rotation, inpainting, rearrangement, coloring
- Contrastive representation learning
  - Intuition and formulation
  - Instance contrastive learning: SimCLR and MOCO
  - Sequence contrastive learning: CPC

Today's Agenda

- Pretext tasks from image transformations
  - Rotation, inpainting, rearrangement, coloring
- Contrastive representation learning
  - Intuition and formulation
  - Instance contrastive learning: SimCLR and MOCO
  - Sequence contrastive learning: CPC

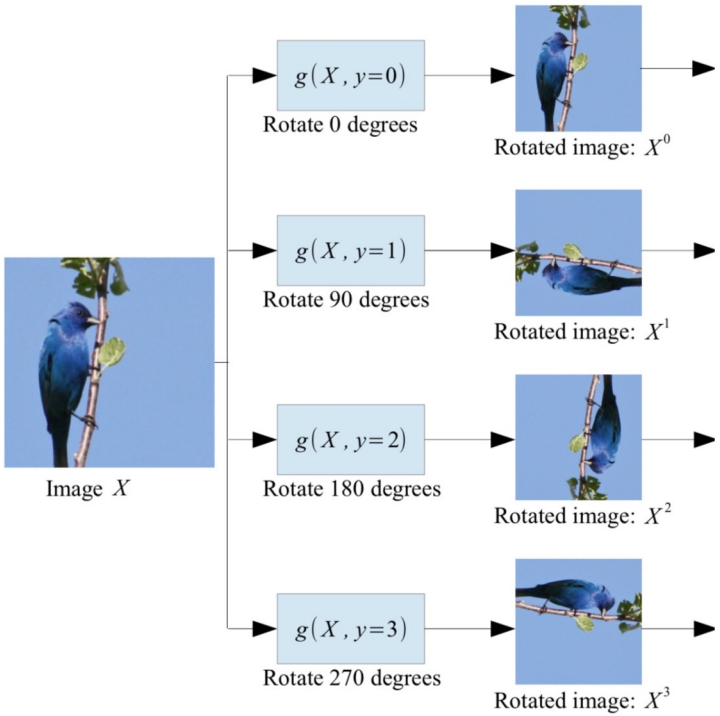
Pretext task: predict rotations



Hypothesis: a model could recognize the correct rotation of an object only if it has the “visual commonsense” of what the object should look like unperturbed.

(Image source: Gidaris et al. 2018)

Pretext task: predict rotations

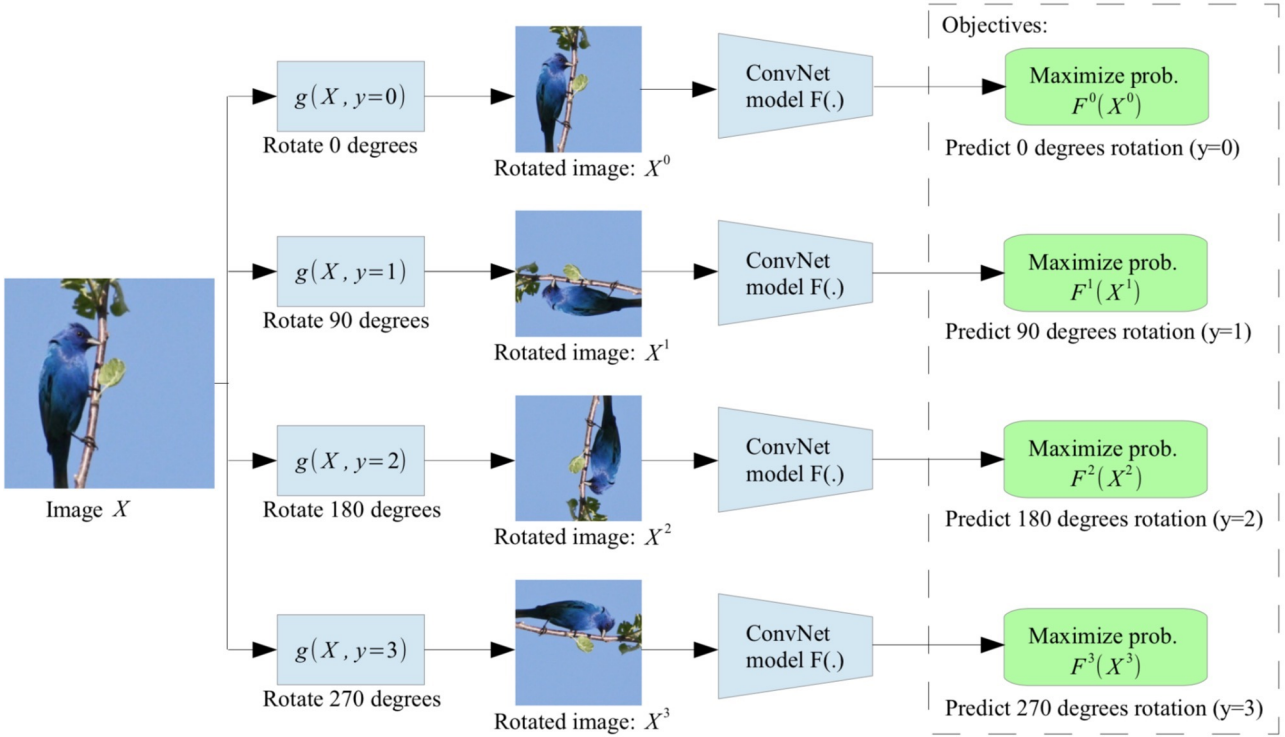


Self-supervised learning by rotating the entire input images.

The model learns to predict which rotation is applied (4-way classification)

(Image source: Gidaris et al. 2018)

Pretext task: predict rotations

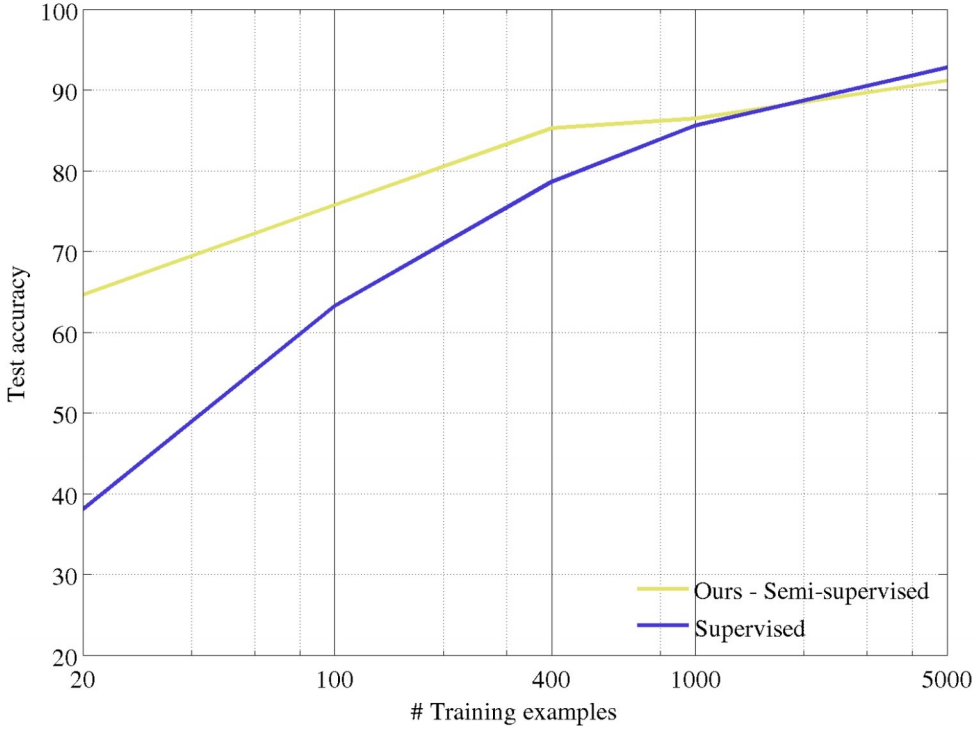


Self-supervised learning by rotating the entire input images.

The model learns to predict which rotation is applied (4-way classification)

(Image source: Gidaris et al. 2018)

Evaluation on semi-supervised learning



Self-supervised learning on **CIFAR10** (entire training set).

Freeze conv1 + conv2
Learn **conv3 + linear** layers with subset of labeled CIFAR10 data (classification).

(Image source: Gidaris et al. 2018)

Transfer learned features to supervised learning

	Classification (%mAP)		Detection (%mAP)	Segmentation (%mIoU)
Trained layers	fc6-8	all	all	all
ImageNet labels	78.9	79.9	56.8	48.0
Random		53.3	43.4	19.8
Random rescaled Krähenbühl et al. (2015)	39.2	56.6	45.6	32.6
Egomotion (Agrawal et al., 2015)	31.0	54.2	43.9	
Context Encoders (Pathak et al., 2016b)	34.6	56.5	44.5	29.7
Tracking (Wang & Gupta, 2015)	55.6	63.1	47.4	
Context (Doersch et al., 2015)	55.1	65.3	51.1	
Colorization (Zhang et al., 2016a)	61.5	65.6	46.9	35.6
BIGAN (Donahue et al., 2016)	52.3	60.1	46.9	34.9
Jigsaw Puzzles (Noroozi & Favaro, 2016)	-	67.6	53.2	37.6
NAT (Bojanowski & Joulin, 2017)	56.7	65.3	49.4	
Split-Brain (Zhang et al., 2016b)	63.0	67.1	46.7	36.0
ColorProxy (Larsson et al., 2017)		65.9		38.4
Counting (Noroozi et al., 2017)	-	67.7	51.4	36.6
(Ours) RotNet	**70.87**	**72.97**	**54.4**	**39.1**

Pretrained with full ImageNet supervision

No pretraining

Self-supervised learning on **ImageNet** (entire training set) with AlexNet.

Finetune on labeled data from **Pascal VOC 2007**.

Self-supervised learning with rotation prediction

source: Gidaris et al. 2018

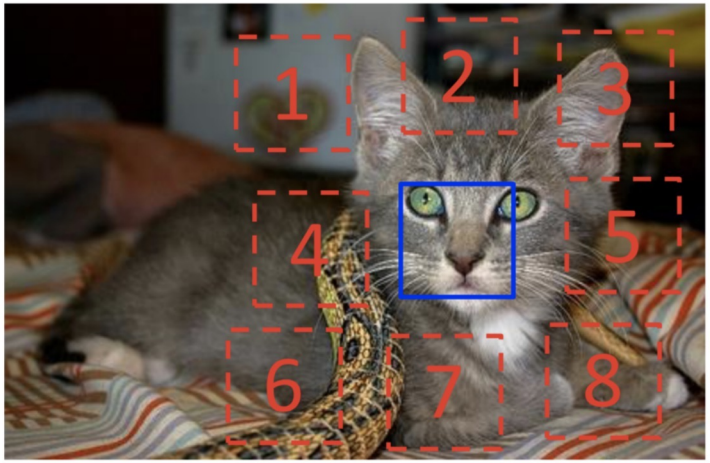
Visualize learned visual attentions

(a) **Attention maps of supervised model**

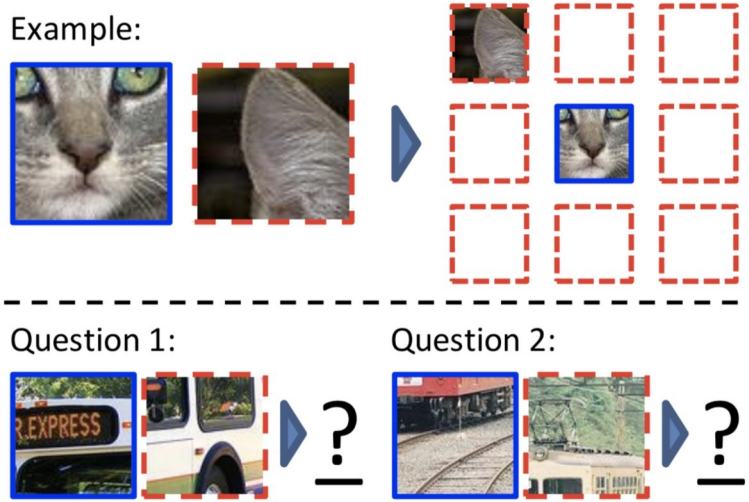
(b) **Attention maps of our self-supervised model**

(Image source: Gidaris et al. 2018)

Pretext task: predict relative patch locations

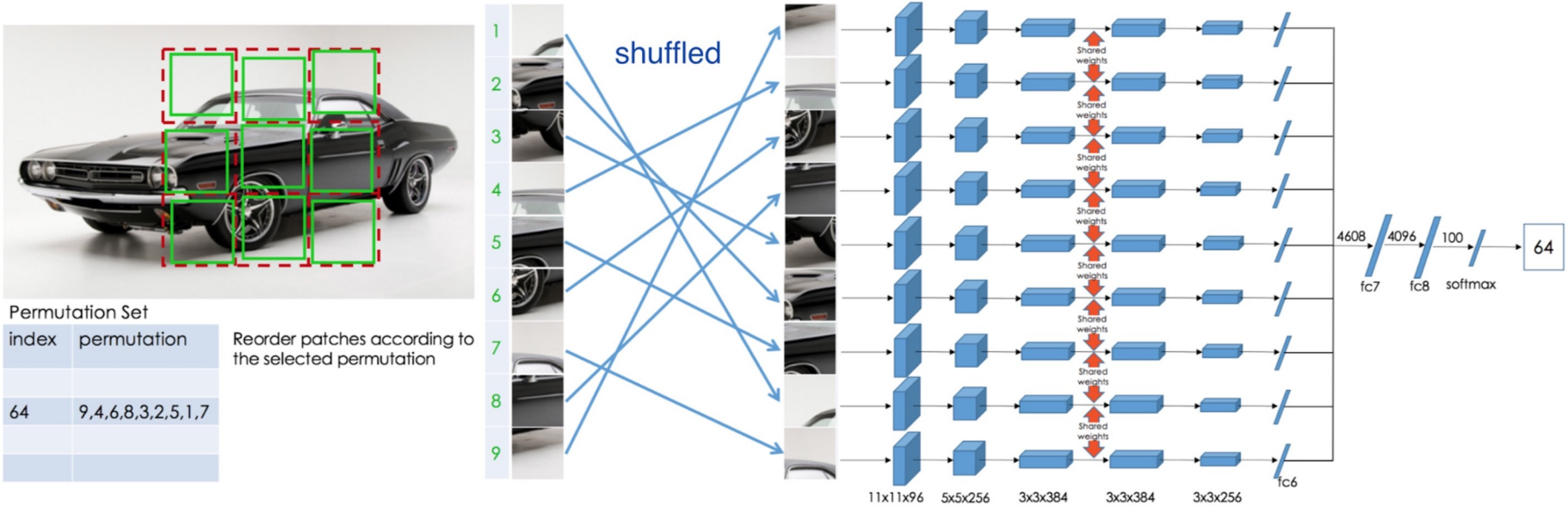


X = (,); Y = 3



(Image source: [Doersch et al., 2015])

Pretext task: solving “jigsaw puzzles”



(Image source: [Noroozi & Favaro, 2016])

Transfer learned features to supervised learning

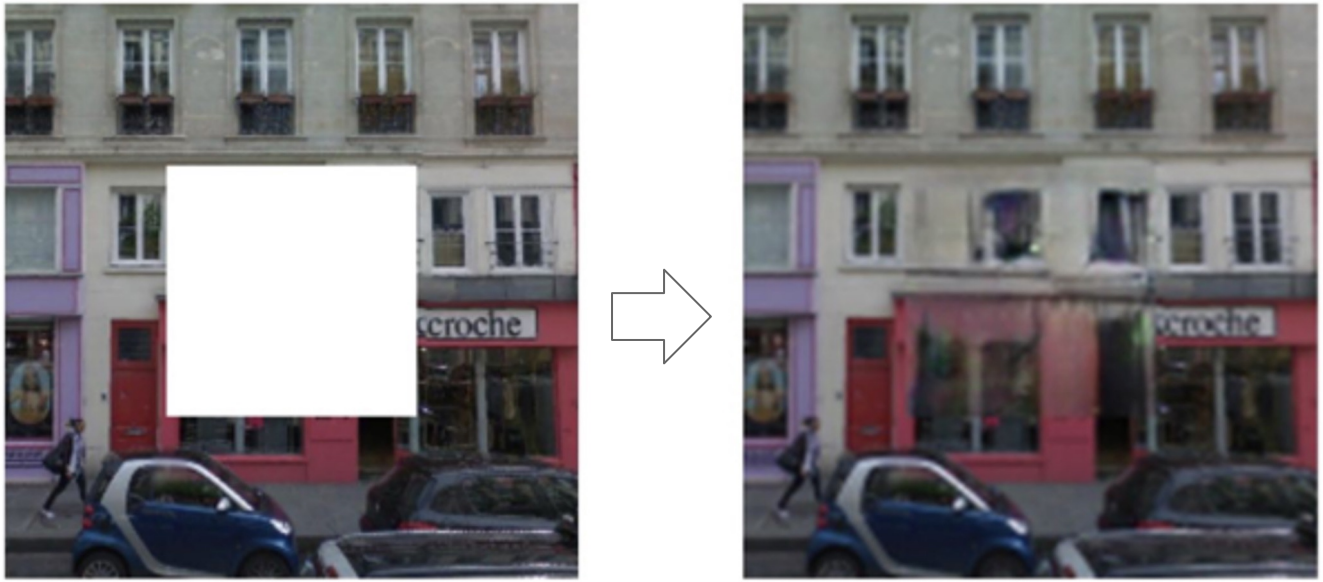
Table 1: Results on PASCAL VOC 2007 Detection and Classification. The results of the other methods are taken from Pathak *et al.* [30].

Method	Pretraining time	Supervision	Classification	Detection	Segmentation
Krizhevsky*et al.* [25]	3 days	1000 class labels	**78.2%**	**56.8%**	**48.0%**
Wang and Gupta[39]	1 week	motion	58.4%	44.0%	-
Doersch *et al.* [10]	4 weeks	context	55.3%	46.6%	-
Pathak *et al.* [30]	14 hours	context	56.5%	44.5%	29.7%
Ours	2.5 days	context	**67.6%**	**53.2%**	**37.6%**

"Ours" is feature learned from solving image Jigsaw puzzles (Noroozi & Favaro, 2016). Doersch et al. is the method with relative patch location

(source: Noroozi & Favaro, 2016)

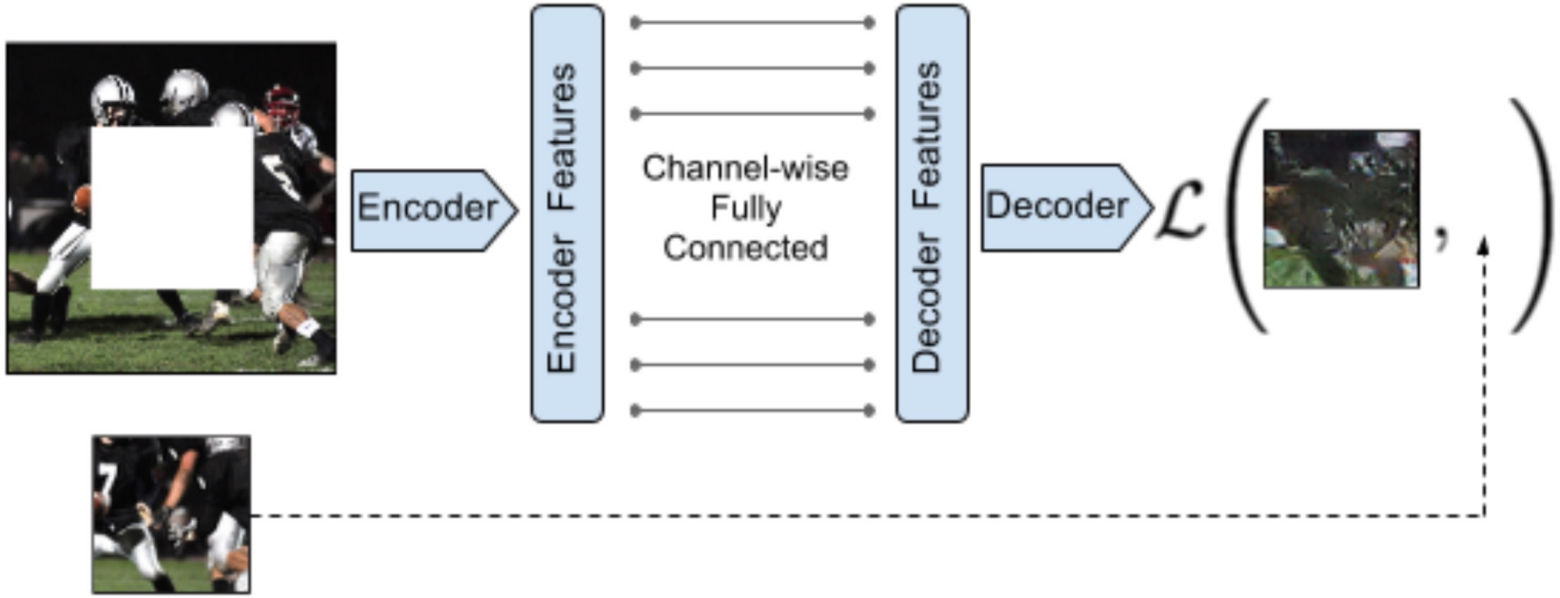
Pretext task: predict missing pixels (inpainting)



Context Encoders: Feature Learning by Inpainting (Pathak et al., 2016)

Source: Pathak et al., 2016

Learning to inpaint by reconstruction



Learning to reconstruct the missing pixels

Source: Pathak et al., 2016

Inpainting evaluation

Input (context)

reconstruction

Source: Pathak et al., 2016

Learning to inpaint by reconstruction

Loss = reconstruction + adversarial learning

$$L(x) = L_{recon}(x) + L_{adv}(x)$$

$$L_{recon}(x) = ||M * (x - F_\theta((1 - M) * x))||_2^2$$

$$L_{adv} = \max_D \mathbb{E}[\log(D(x))] + \log(1 - D(F((1 - M) * x)))]$$

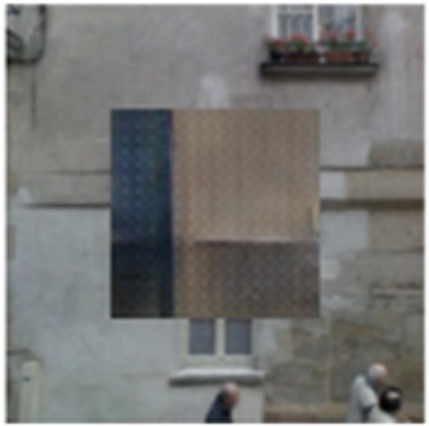
Adversarial loss between “real” images and *inpainted images*

Source: [Pathak et al., 2016]

Inpainting evaluation

Input (context)

reconstruction



adversarial

recon + adv

Source: [Pathak et al., 2016]

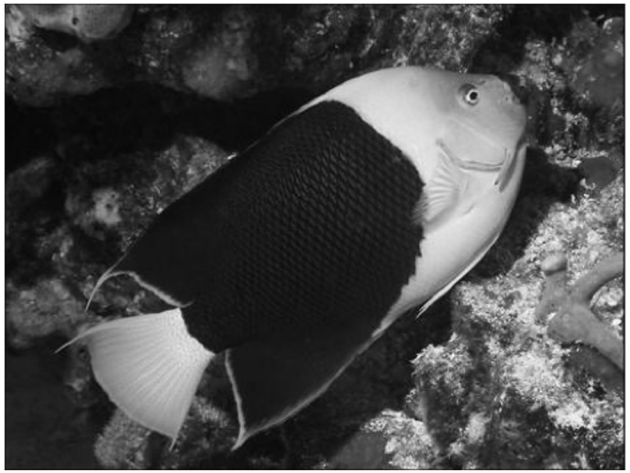
Transfer learned features to supervised learning

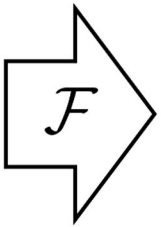
Pretraining Method	Supervision	Pretraining time	Classification	Detection	Segmentation
ImageNet [26]	1000 class labels	3 days	78.2%	56.8%	48.0%
Random Gaussian	initialization	< 1 minute	53.3%	43.4%	19.8%
Autoencoder	-	14 hours	53.8%	41.9%	25.2%
Agrawal *et al.* [1]	egomotion	10 hours	52.9%	41.8%	-
Wang *et al.* [39]	motion	1 week	58.7%	47.4%	-
Doersch *et al.* [7]	relative context	4 weeks	55.3%	46.6%	-
Ours	context	14 hours	56.5%	44.5%	30.0%

Self-supervised learning on ImageNet training set, transfer to classification (Pascal VOC 2007), detection (Pascal VOC 2007), and semantic segmentation (Pascal VOC 2012)

Source: Pathak et al., 2016

Pretext task: image coloring



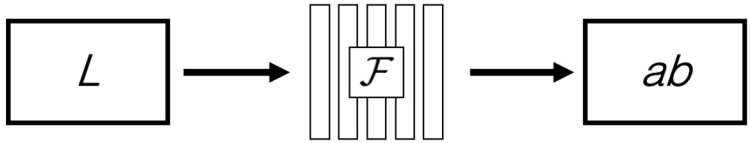


Grayscale image: *L* channel

$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$

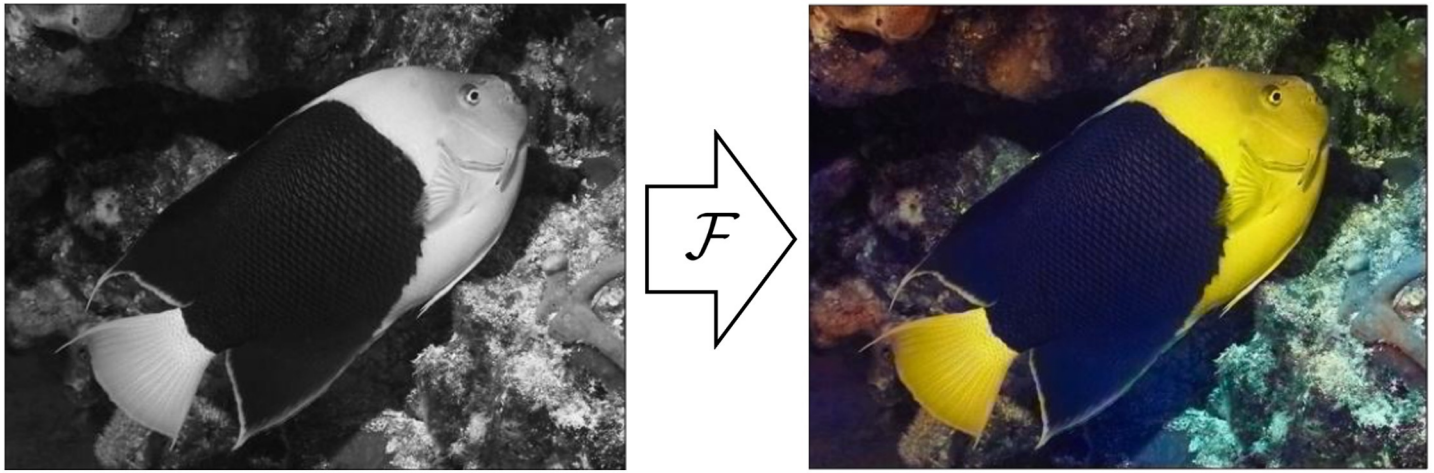
Color information: *ab* channels

$\hat{\mathbf{Y}} \in \mathbb{R}^{H \times W \times 2}$



Source: Richard Zhang / Phillip Isola

Pretext task: image coloring

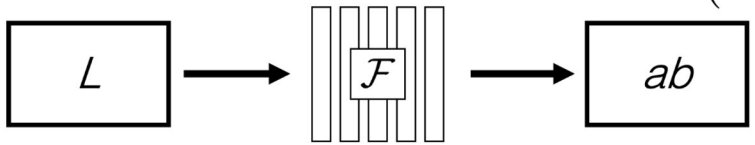


Grayscale image: *L* channel

$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$

Concatenate (*L*,*ab*) channels

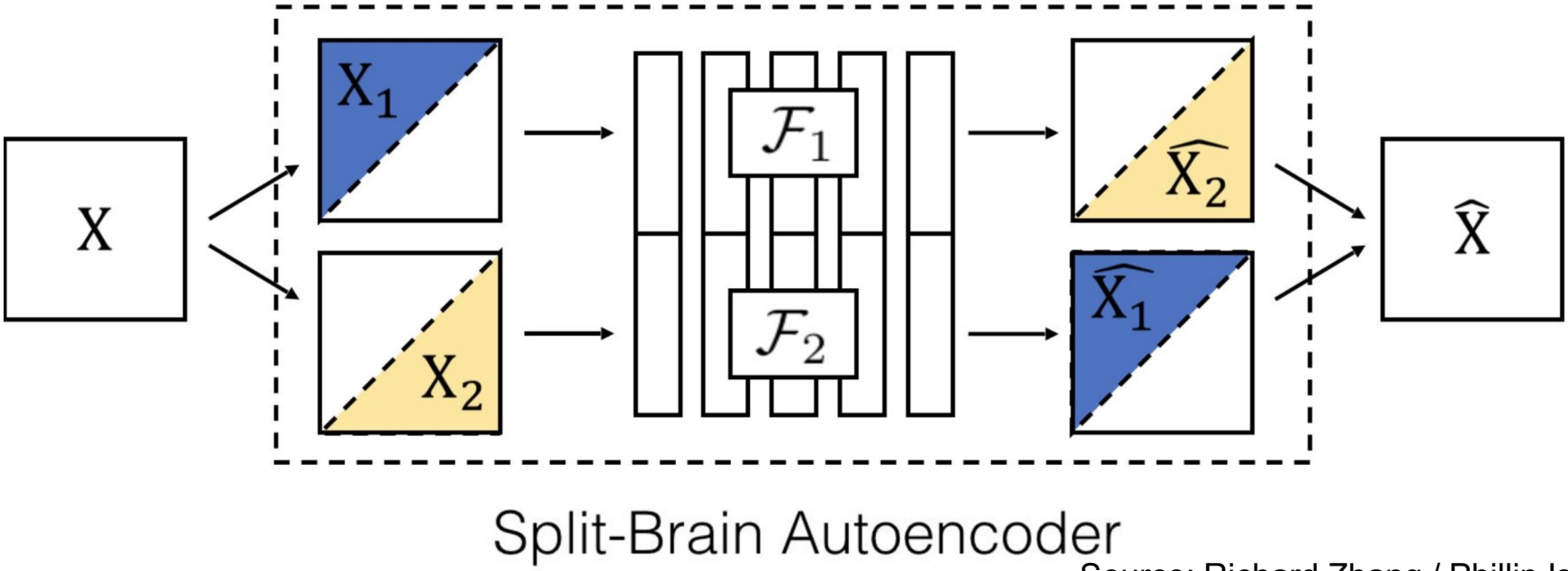
$(\mathbf{X}, \hat{\mathbf{Y}})$



Source: Richard Zhang / Phillip Isola

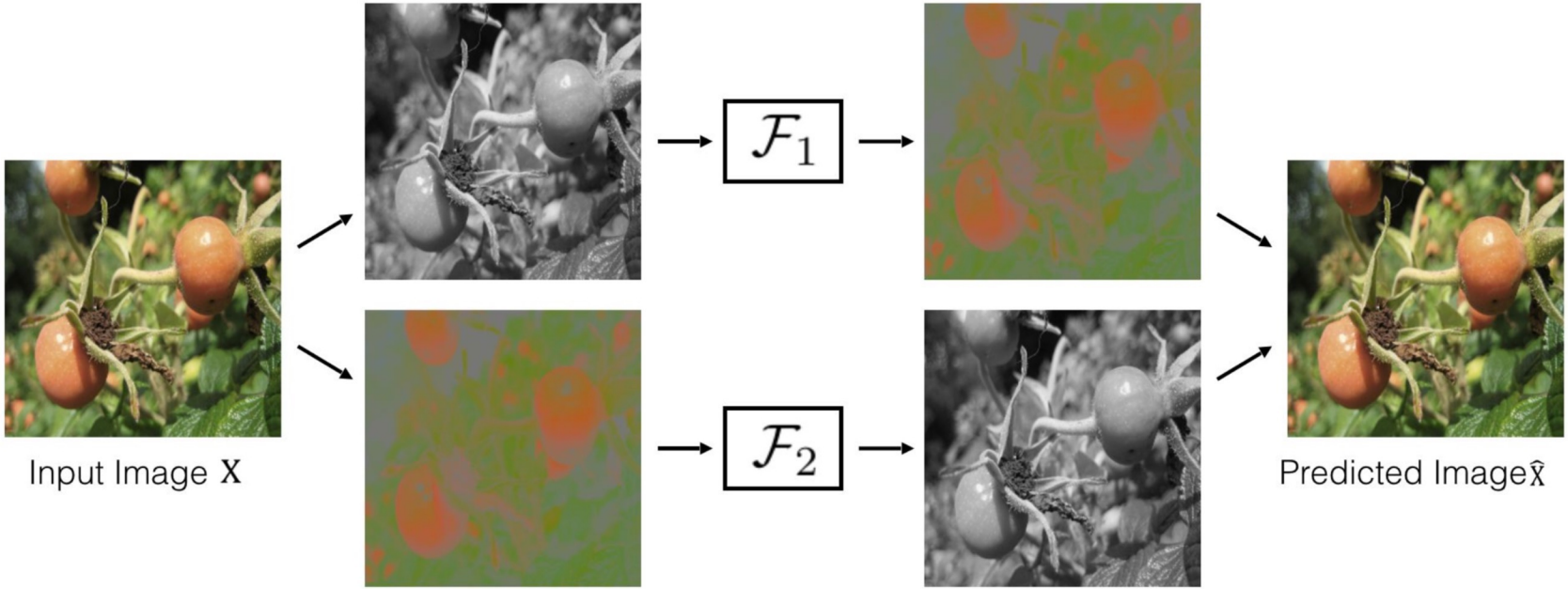
Learning features from colorization: Split-brain Autoencoder

Idea: cross-channel predictions



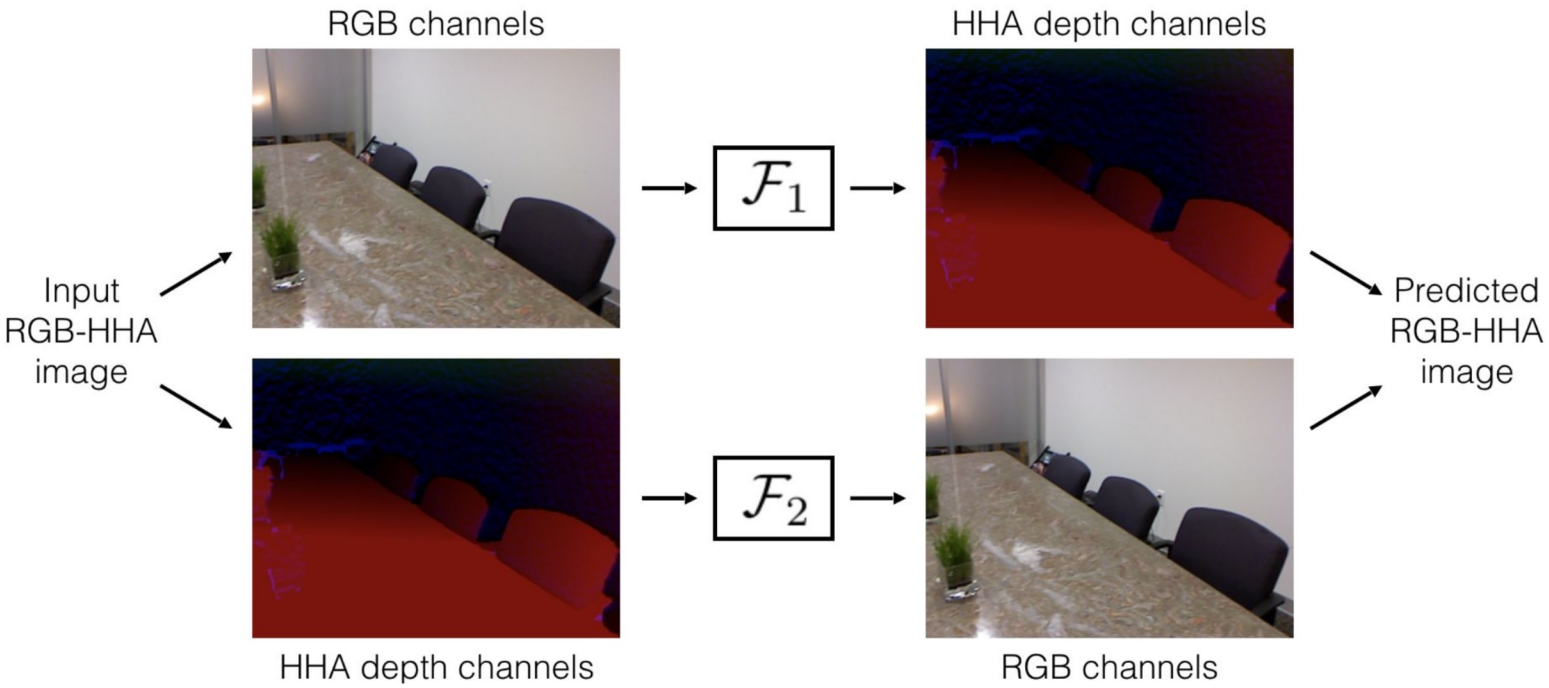
Source: Richard Zhang / Phillip Isola

Learning features from colorization: Split-brain Autoencoder



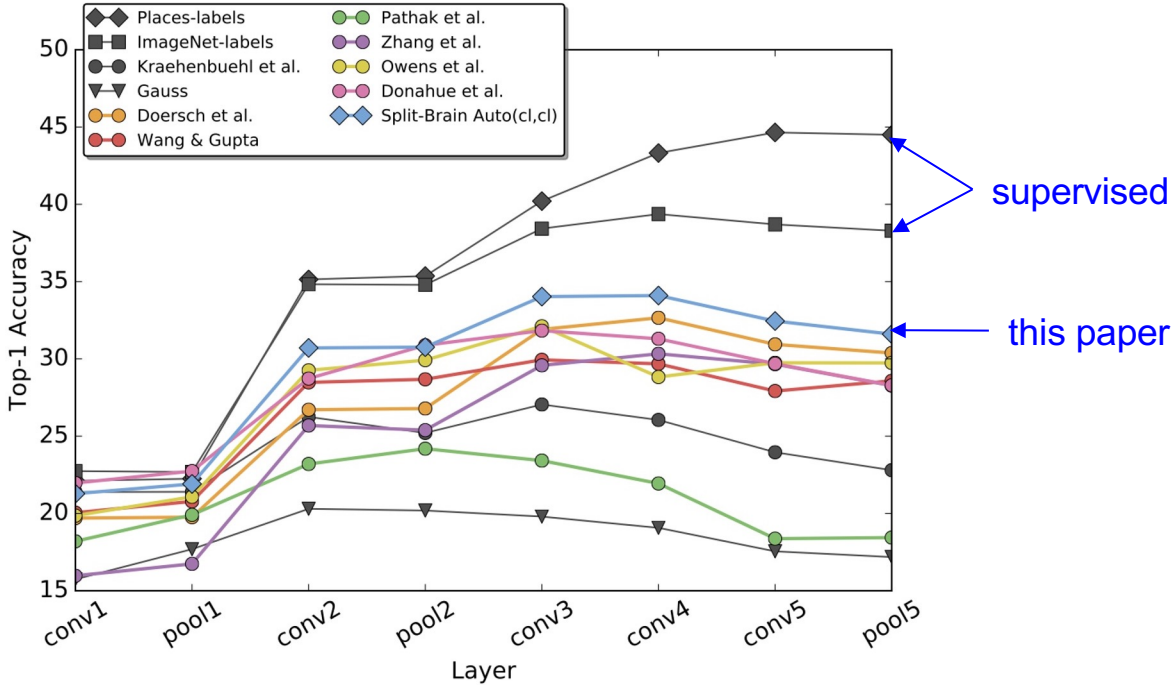
Source: Richard Zhang / Phillip Isola

Learning features from colorization: Split-brain Autoencoder



Source: Richard Zhang / Phillip Isola

Transfer learned features to supervised learning



Self-supervised learning on **ImageNet** (entire training set).

Use *concatenated features* from F_1 and F_2

Labeled data is from the **Places** (Zhou 2016).

Source: Zhang et al., 2017

Pretext task: image coloring

Source: Richard Zhang / Phillip Isola

Pretext task: image coloring

Source: Richard Zhang / Phillip Isola

Pretext task: video coloring

Idea: model the *temporal coherence* of colors in videos

reference frame

t = 0

how should I color these frames?

t = 1

t = 2

t = 3

...

Source: Vondrick et al., 2018

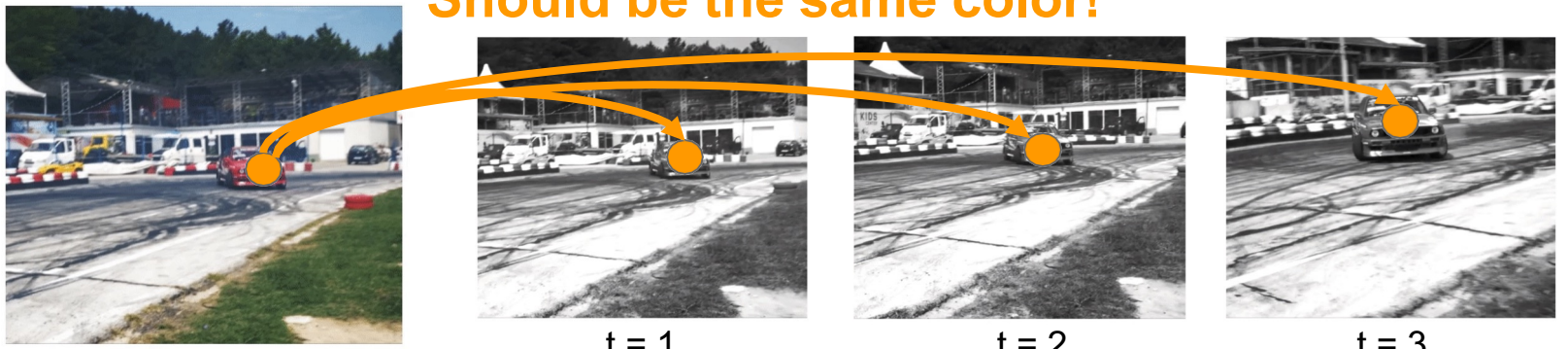
Pretext task: video coloring

Idea: model the *temporal coherence* of colors in videos

reference frame

how should I color these frames?

Should be the same color!



t = 0

t = 1

t = 2

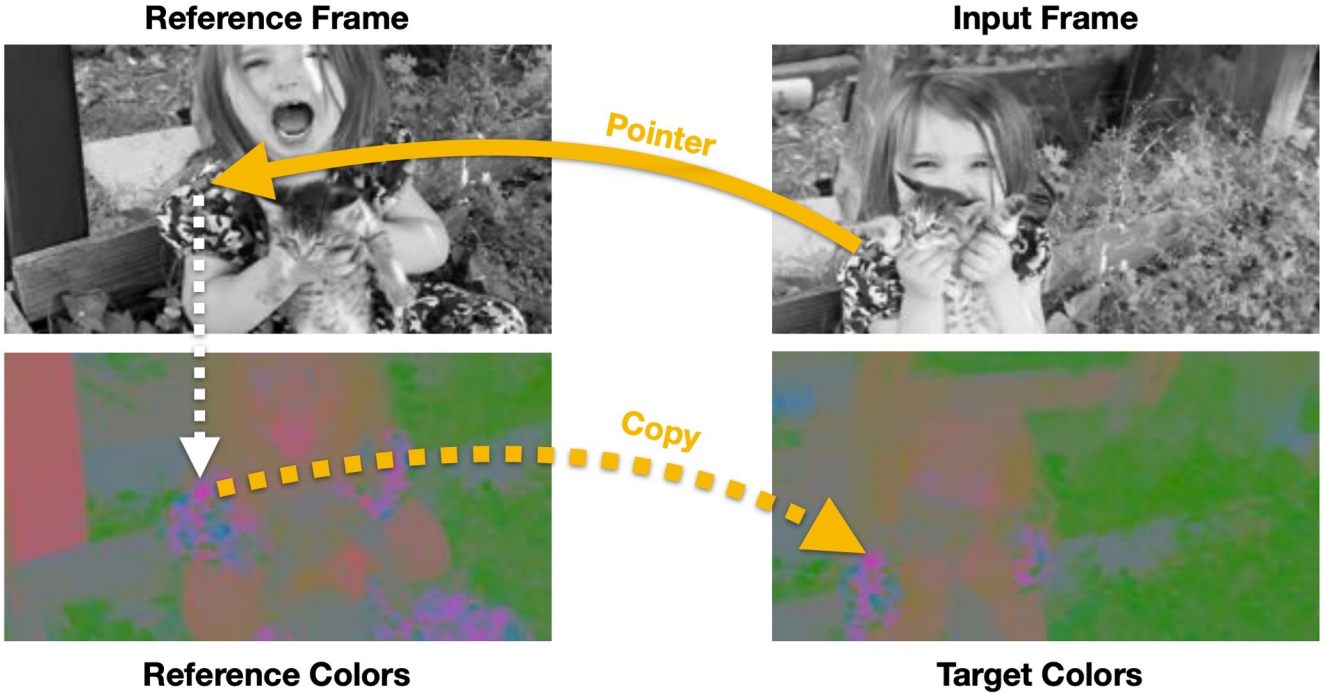
t = 3

...

Hypothesis: learning to color video frames should allow model to learn to track regions or objects without labels!

Source: Vondrick et al., 2018

Learning to color videos



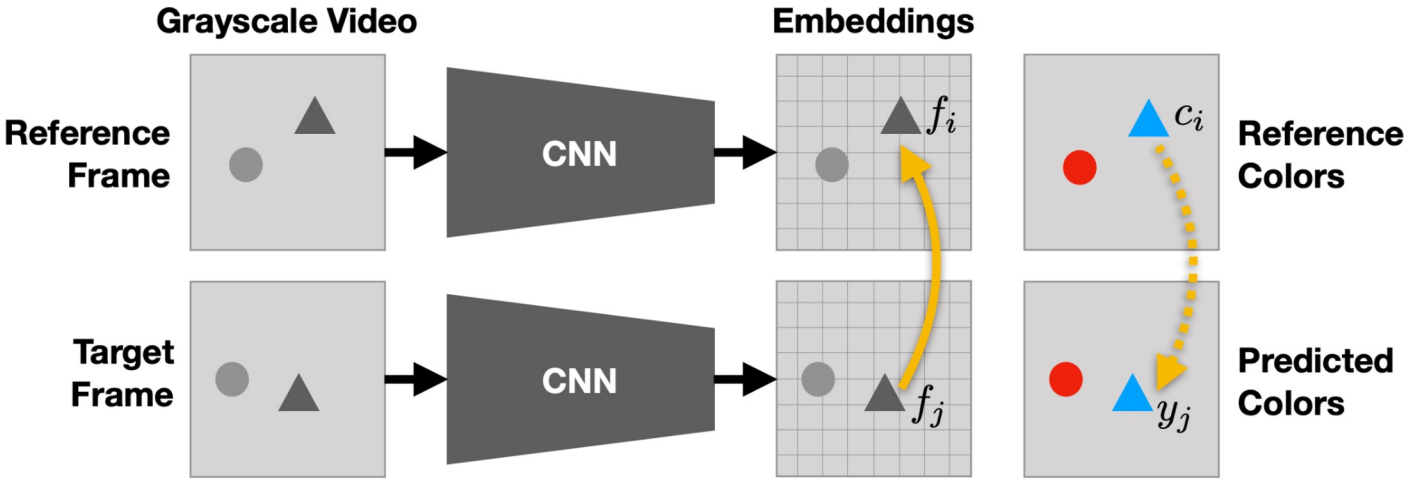
Learning objective:

Establish mappings between reference and target frames in a learned feature space.

Use the mapping as “pointers” to copy the correct color (LAB).

Source: Vondrick et al., 2018

Learning to color videos

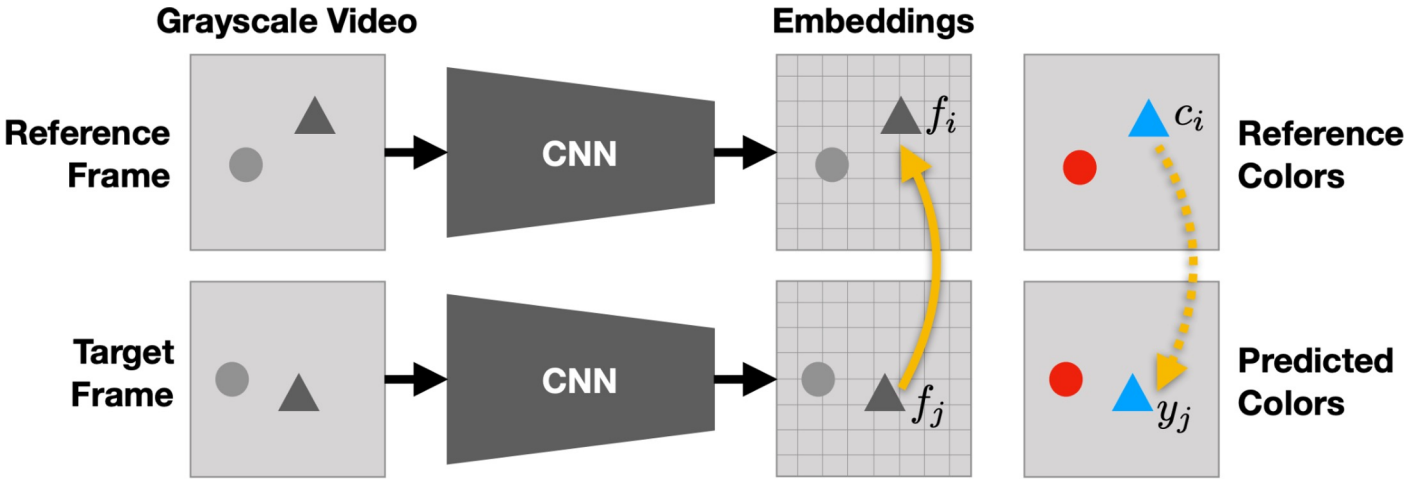


attention map on the reference frame

$$A_{ij} = \frac{\exp\left(f_i^T f_j\right)}{\sum_k \exp\left(f_k^T f_j\right)}$$

Source: Vondrick et al., 2018

Learning to color videos



attention map on the reference frame

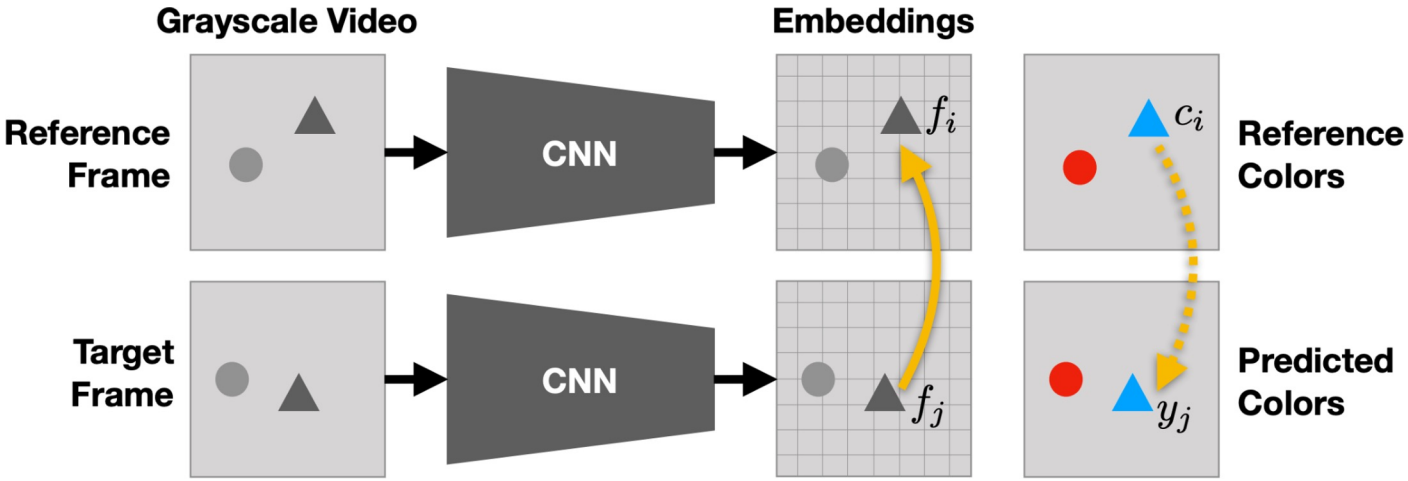
$$A_{ij} = \frac{\exp\left(f_i^T f_j\right)}{\sum_k \exp\left(f_k^T f_j\right)}$$

predicted color = weighted sum of the reference color

$$y_j = \sum_i A_{ij} c_i$$

Source: Vondrick et al., 2018

Learning to color videos



attention map on the reference frame

$$A_{ij} = \frac{\exp\left(f_i^T f_j\right)}{\sum_k \exp\left(f_k^T f_j\right)}$$

predicted color = weighted sum of the reference color

$$y_j = \sum_i A_{ij} c_i$$

loss between predicted color and ground truth color

$$\min_\theta \sum_j \mathcal{L}\left(y_j, c_j\right)$$

Source: Vondrick et al., 2018

Colorizing videos (qualitative)

reference frame

target frames (gray)

predicted color

Source: Google AI blog post

Colorizing videos (qualitative)

reference frame

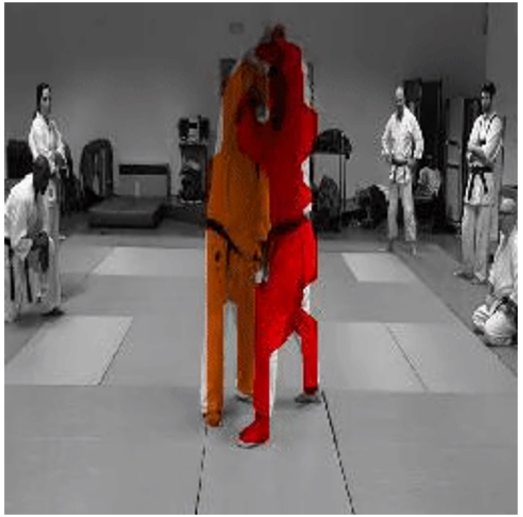
target frames (gray)

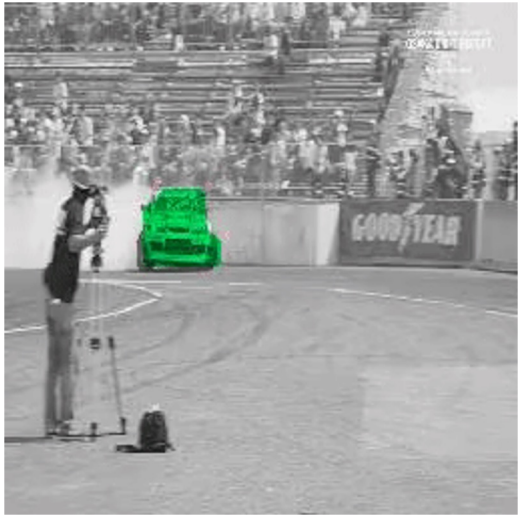
predicted color

Source: Google AI blog post

Tracking emerges from colorization

Propagate segmentation masks using learned attention

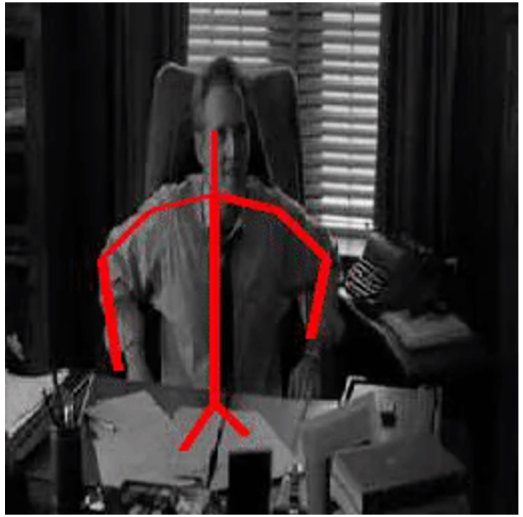




Source: [Google AI blog post]

Tracking emerges from colorization

Propagate pose keypoints using learned attention



Source: [Google AI blog post]

Summary: pretext tasks from image transformations

- Pretext tasks focus on “visual common sense”, e.g., predict rotations, inpainting, rearrangement, and colorization.
- The models are forced to learn good features about natural images, e.g., semantic representation of an object category, in order to solve the pretext tasks.
- We don’t care about the performance of these pretext tasks, but rather how useful the learned features are for downstream tasks (classification, detection, segmentation).

Summary: pretext tasks from image transformations

- Pretext tasks focus on “visual common sense”, e.g., predict rotations, inpainting, rearrangement, and colorization.
- The models are forced learn good features about natural images, e.g., semantic representation of an object category, in order to solve the pretext tasks.
- We don’t care about the performance of these pretext tasks, but rather how useful the learned features are for downstream tasks (classification, detection, segmentation).
- Problems: 1) coming up with individual pretext tasks is tedious, and 2) the learned representations may not be general.

Pretext tasks from image transformations

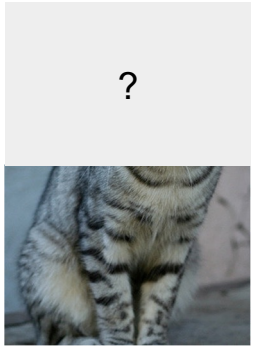
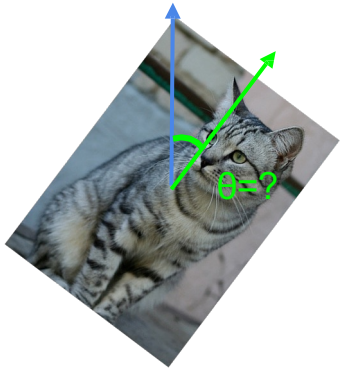


image completion



rotation prediction

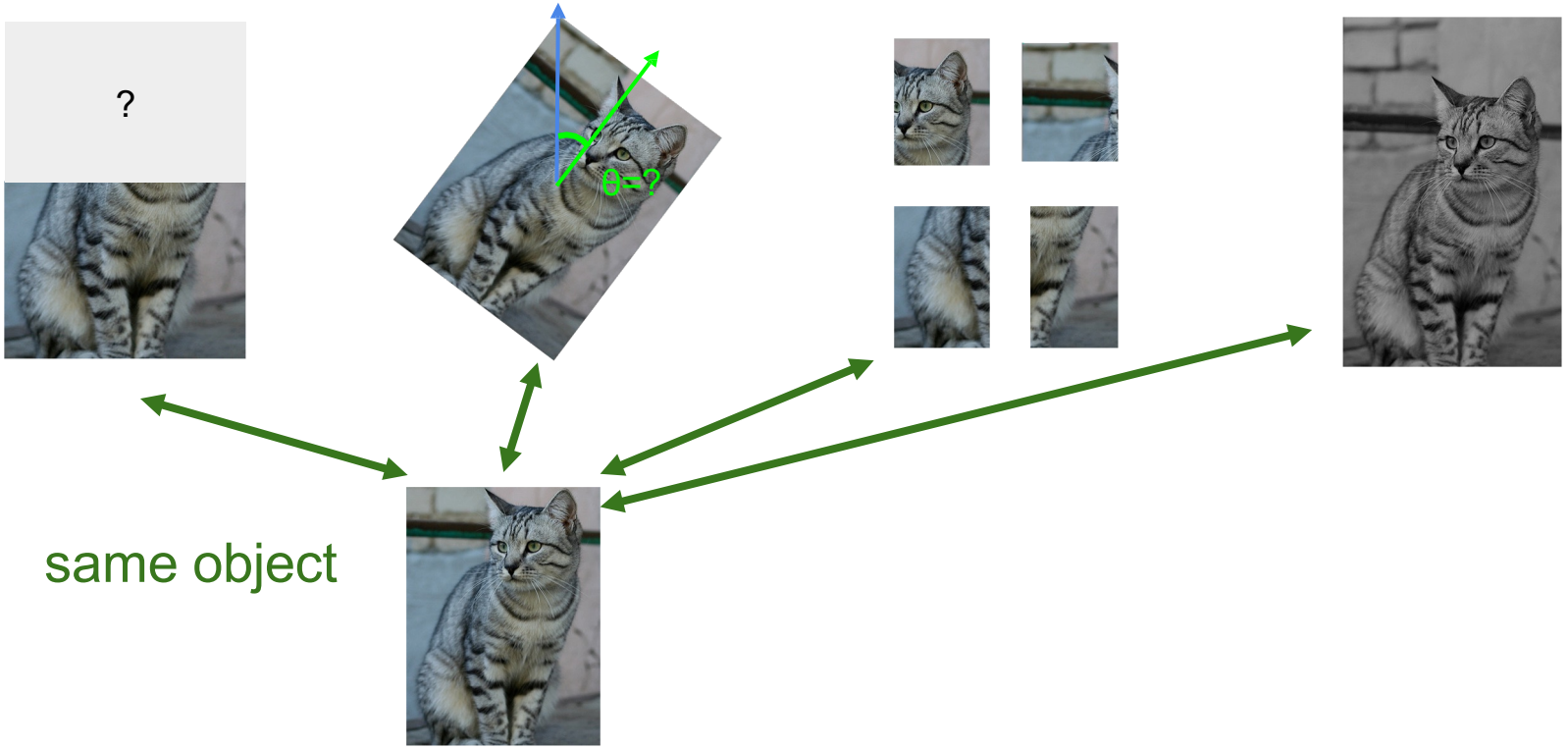
“jigsaw puzzle”

colorization

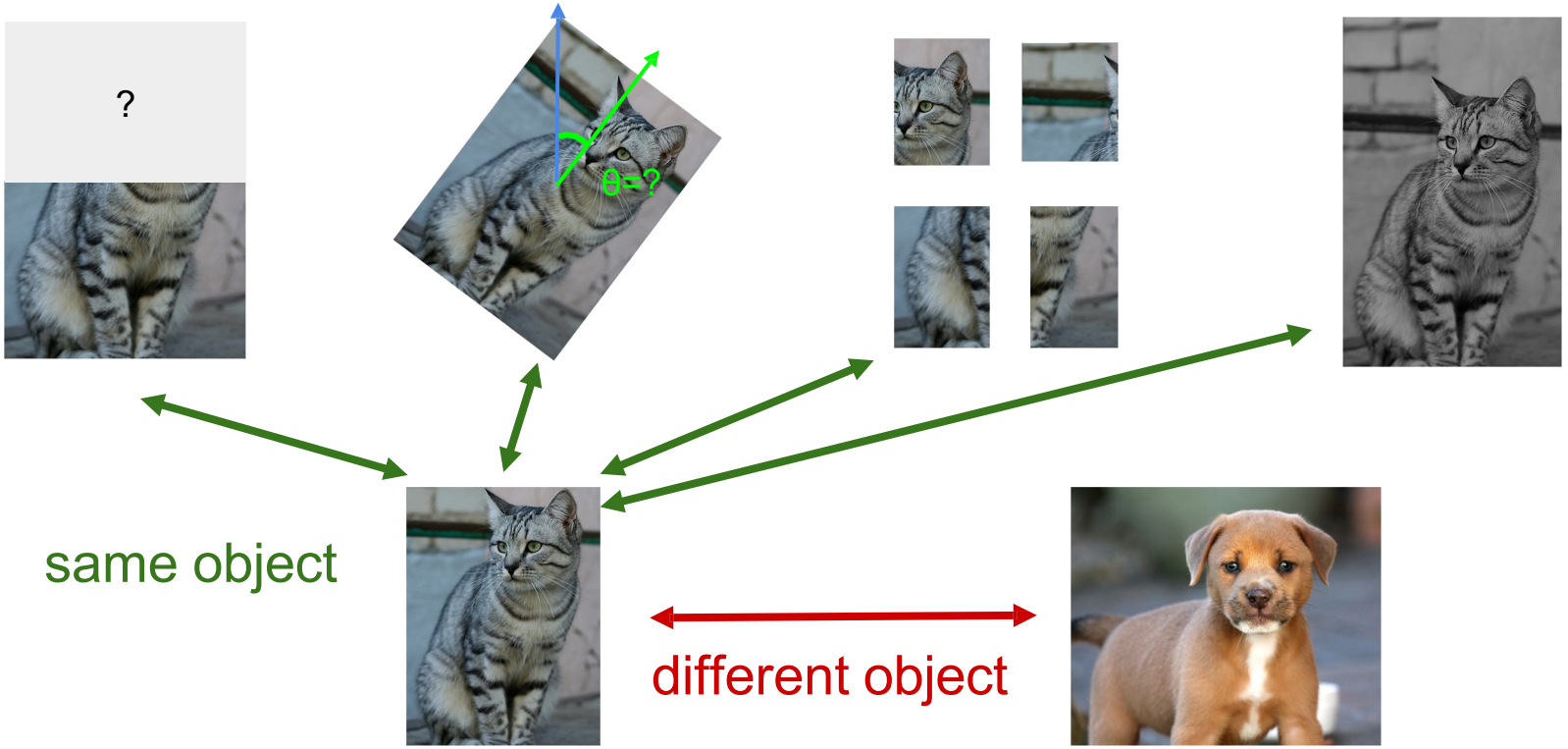
Learned representations may be tied to a specific pretext task!

Can we come up with a more general pretext task?

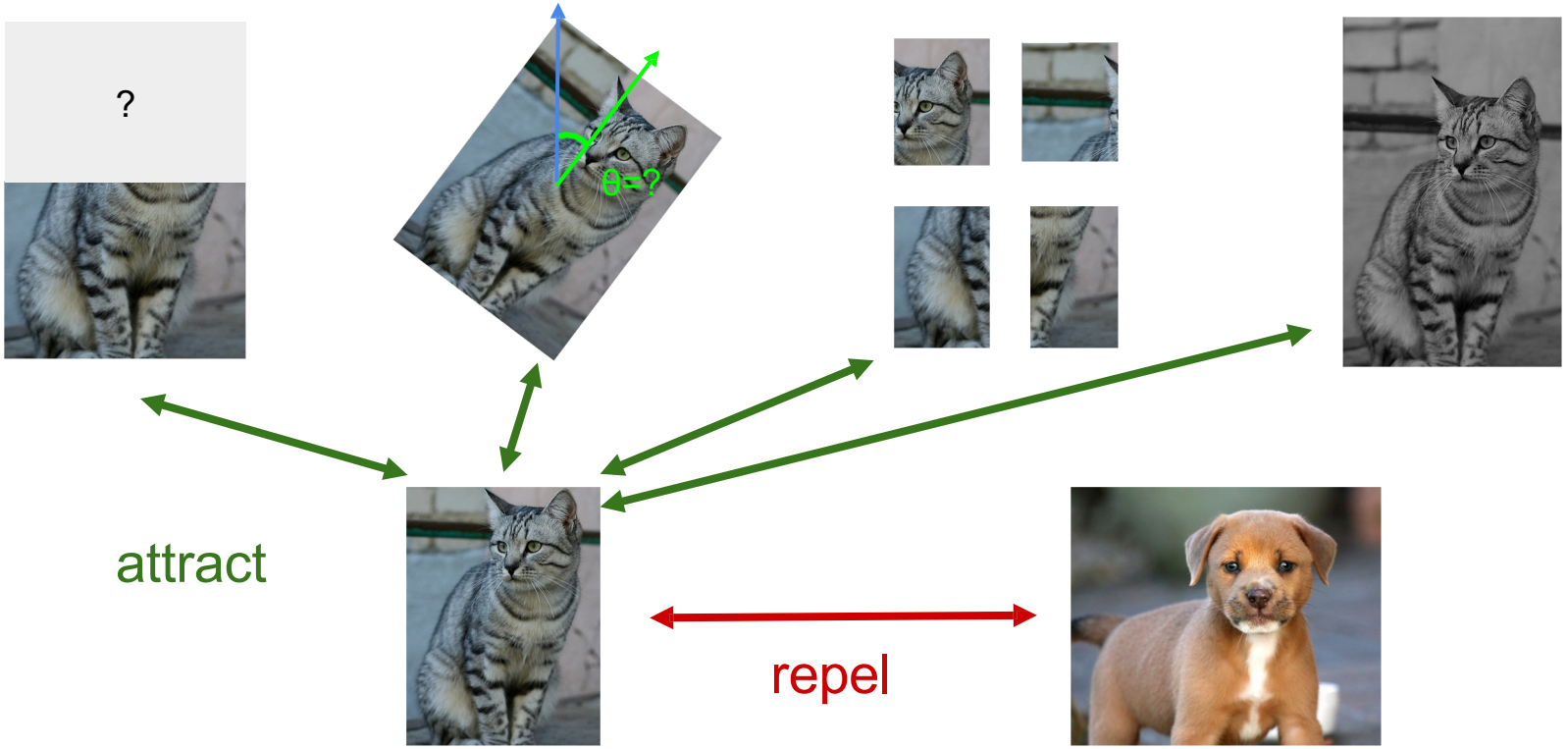
A more general pretext task?



A more general pretext task?



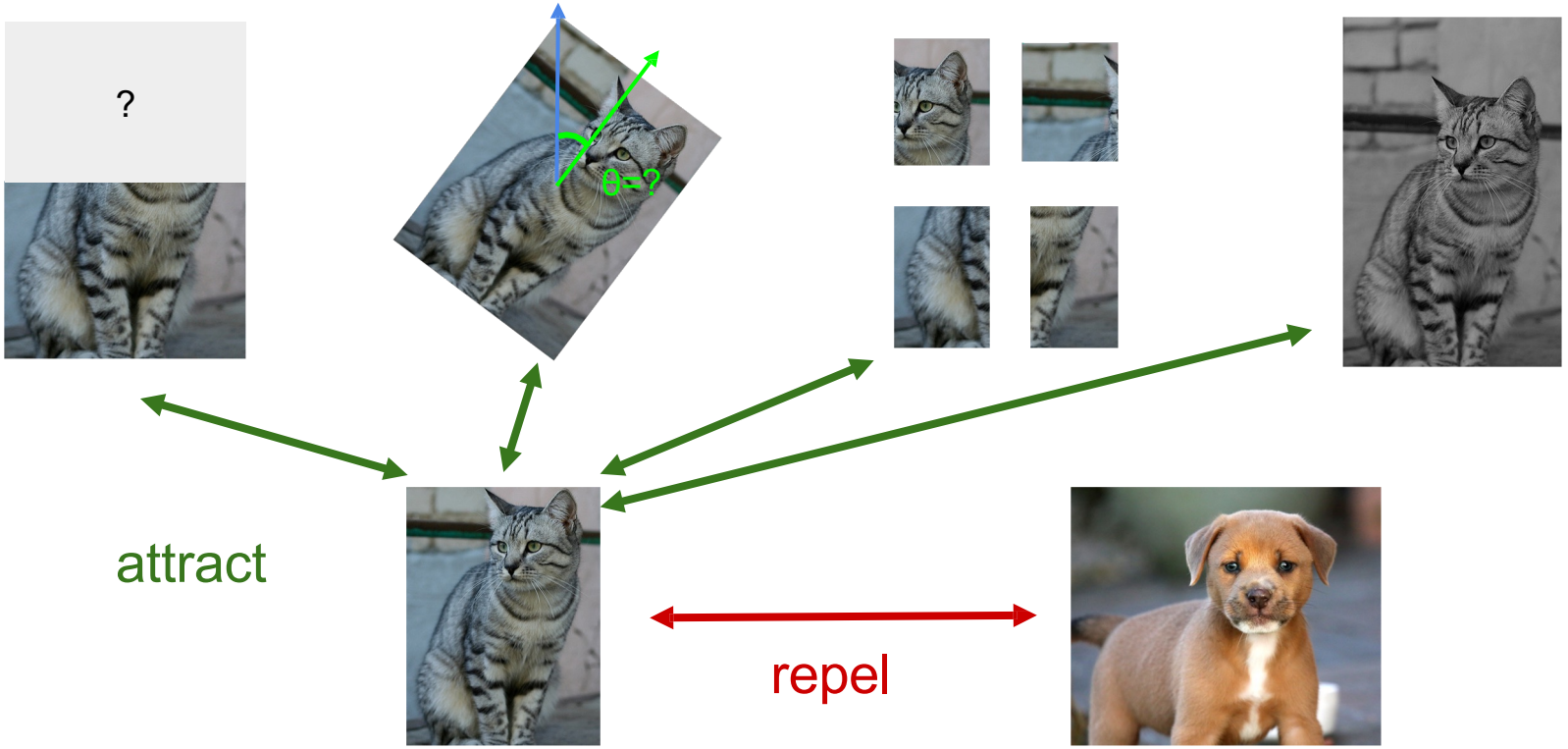
Contrastive Representation Learning



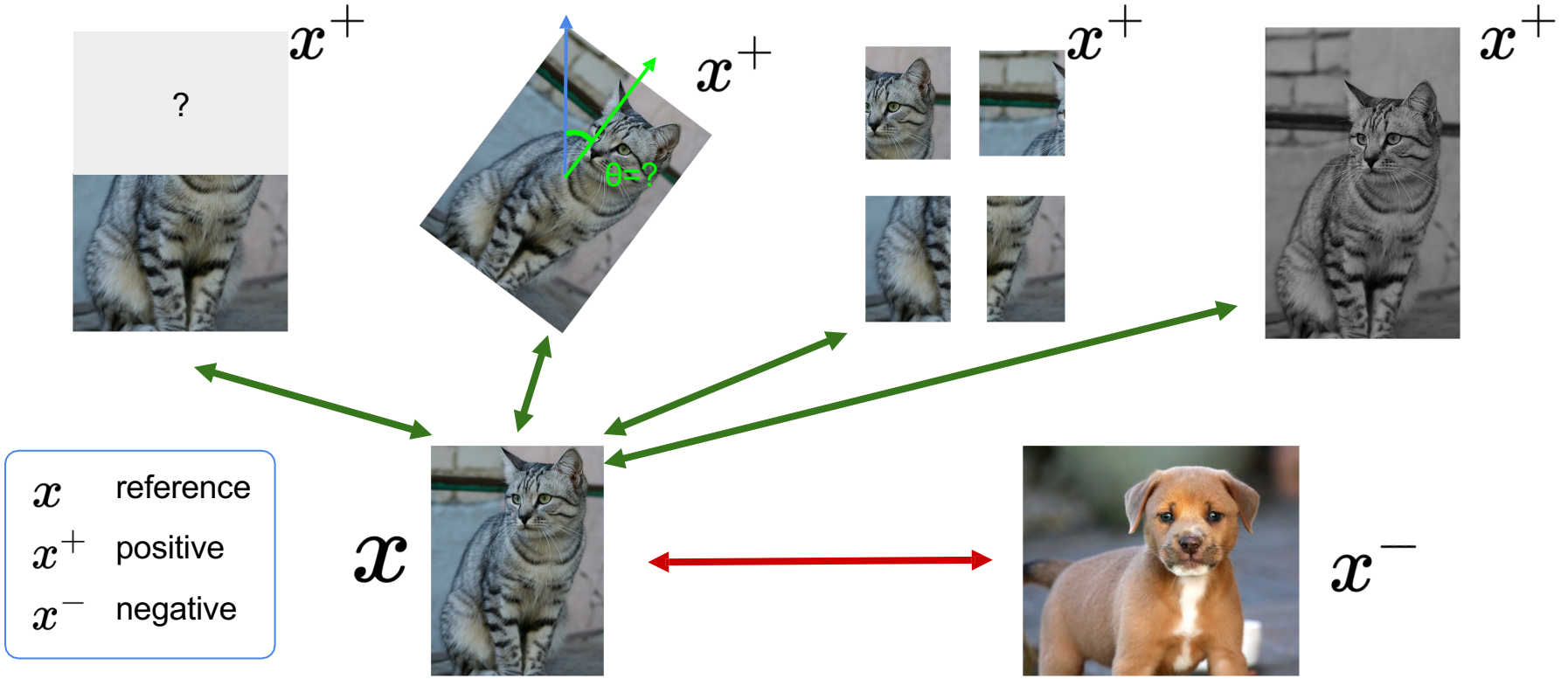
Today's Agenda

- Pretext tasks from image transformations
  - Rotation, inpainting, rearrangement, coloring
- Contrastive representation learning
  - Intuition and formulation
  - Instance contrastive learning: SimCLR and MOCO
  - Sequence contrastive learning: CPC

Contrastive Representation Learning



Contrastive Representation Learning



A formulation of contrastive learning

What we want:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

x: reference sample; x^+ positive sample; x^- negative sample

Given a chosen score function, we aim to learn an **encoder function** *f* that yields high score for positive pairs (x, x^+) and low scores for negative pairs (x, x^-).

A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

x

x^+

x

x_1^-

x_2^-

x_3^-

...

A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

score for the positive pair

score for the N-1 negative pairs

This seems familiar …

A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

score for the positive pair

score for the N-1 negative pairs

This seems familiar …

Cross entropy loss for a N-way softmax classifier!

I.e., learn to find the positive sample from the N samples

A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

Commonly known as the InfoNCE loss (van den Oord et al., 2018)

A *lower bound* on the mutual information between *f(x)* and *f(x⁺)*

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

The larger the negative sample size (*N*), the tighter the bound

Detailed derivation: Poole et al., 2019

SimCLR: A Simple Framework for Contrastive Learning

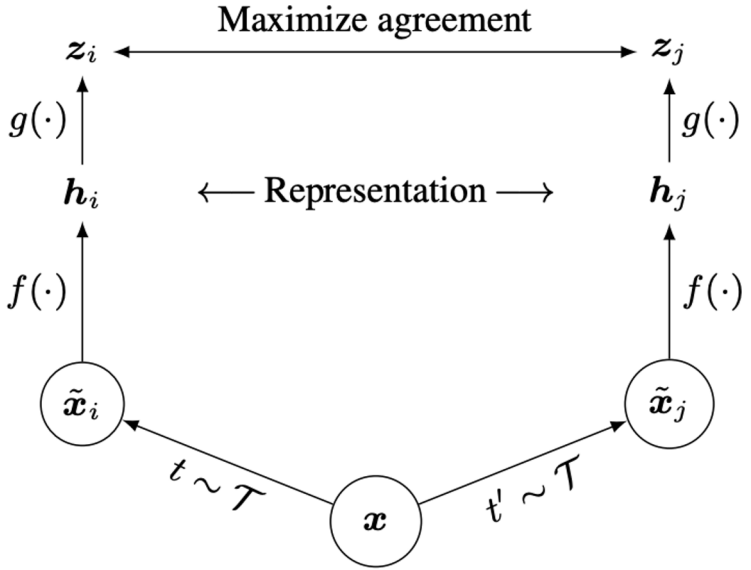
Cosine similarity as the score function:

$$s(u, v) = \frac{u^T v}{||u||||v||}$$

Use a projection network ***g(·)*** to project features to a space where contrastive learning is applied

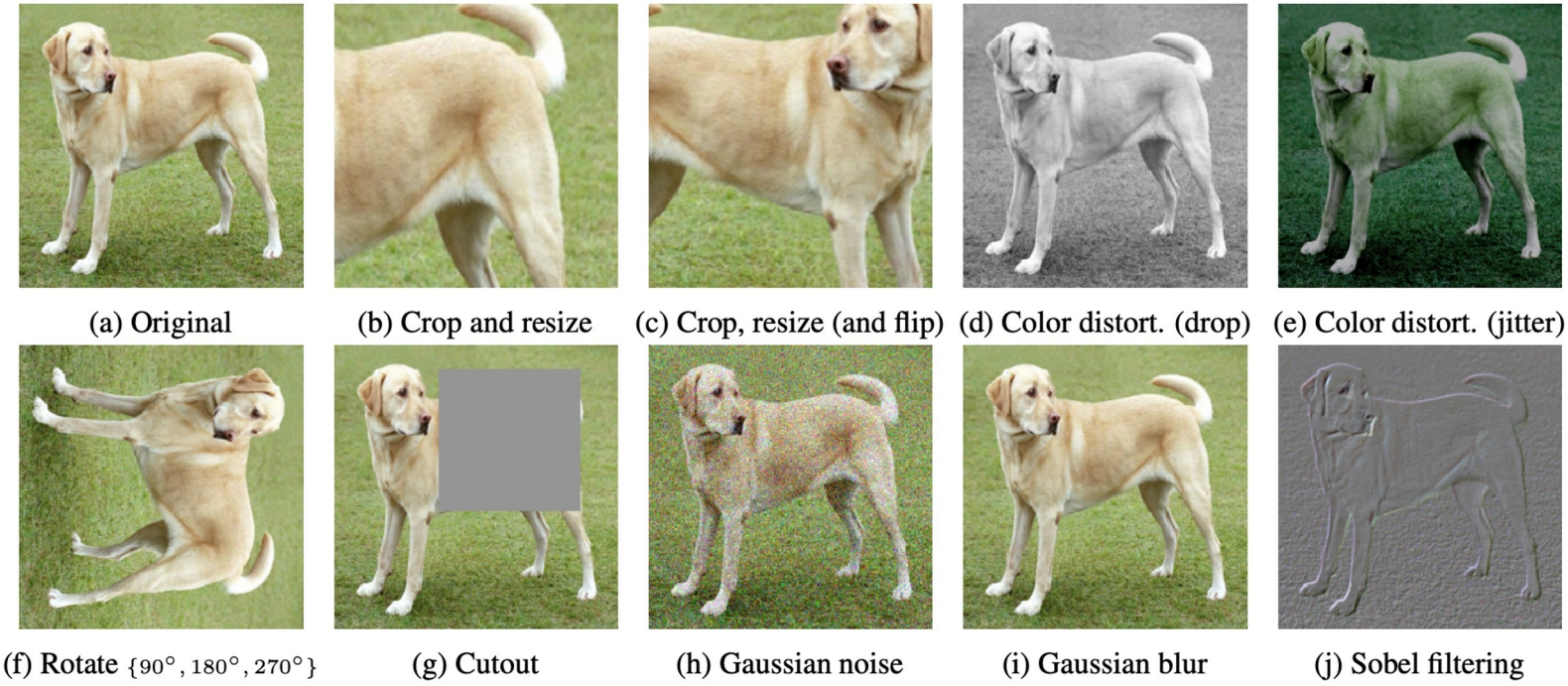
Generate positive samples through data augmentation:

- random cropping, random color distortion, and random blur.



Source: [Chen et al., 2020]

SimCLR: generating positive samples from data augmentation



(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

Source: Chen et al., 2020

SimCLR

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Generate a positive pair by sampling data augmentation functions

*We use a slightly different formulation in the assignment. You should follow the assignment instructions.

Source: Chen et al., 2020

SimCLR

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Generate a positive pair by sampling data augmentation functions

*We use a slightly different formulation in the assignment. You should follow the assignment instructions.

InfoNCE loss: Use all non-positive samples in the batch as x^-

Source: Chen et al., 2020

SimCLR

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Generate a positive pair by sampling data augmentation functions

Iterate through and use each of the 2N sample as reference, compute average loss

InfoNCE loss: Use all non-positive samples in the batch as x^-

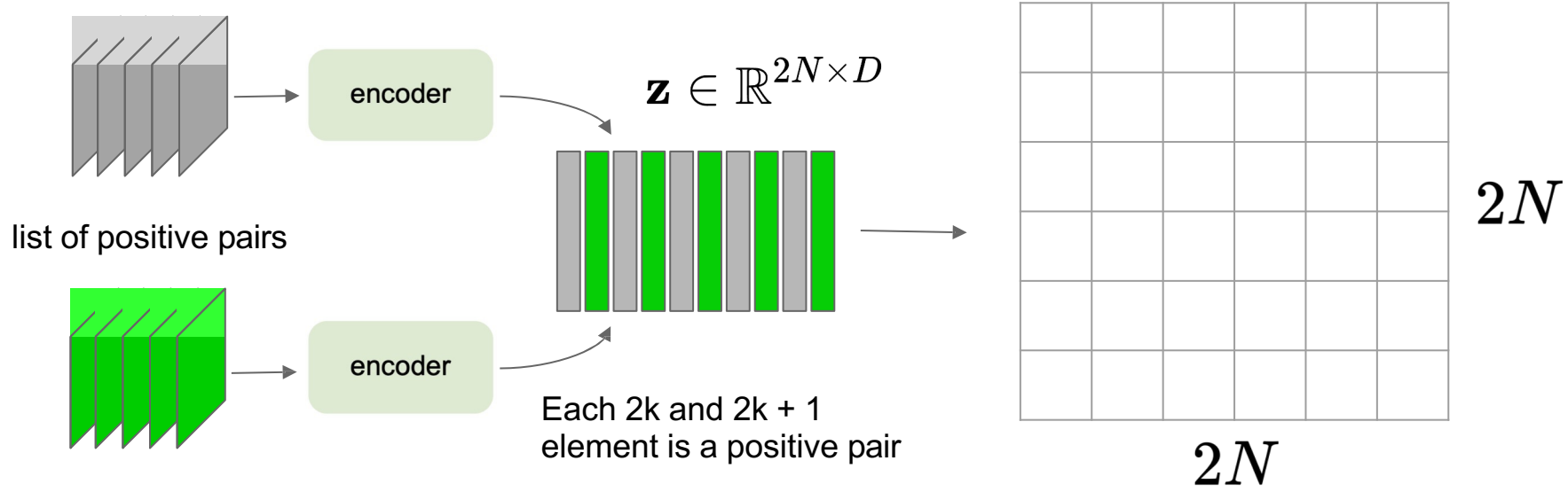
*We use a slightly different formulation in the assignment. You should follow the assignment instructions.

Source: Chen et al., 2020

SimCLR: mini-batch training

$$s_{i,j} = \frac{z_i^T z_j}{||z_i|| \, ||z_j||}$$

"Affinity matrix"

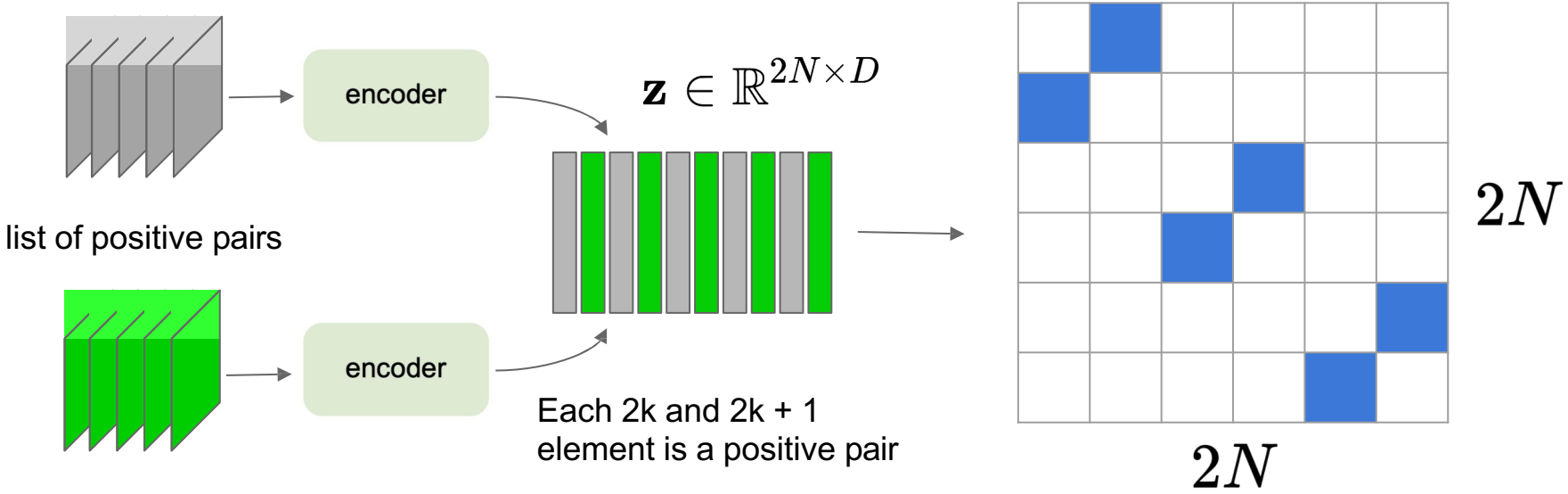


*We use a slightly different formulation in the assignment.
You should follow the assignment instructions.

SimCLR: mini-batch training

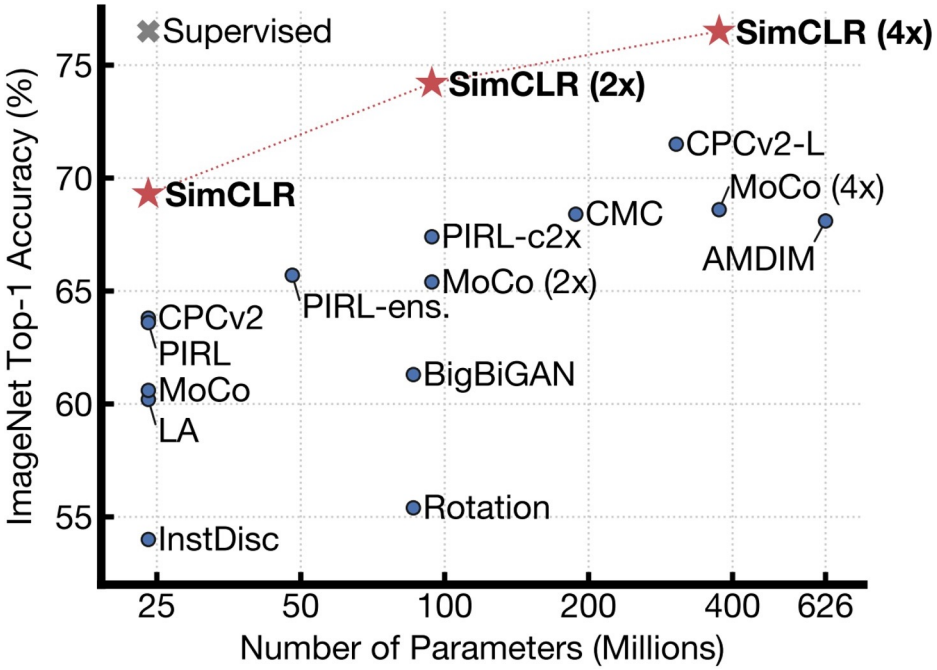
$$s_{i,j} = \frac{z_i^T z_j}{||z_i||\,||z_j||}$$

"Affinity matrix"



*We use a slightly different formulation in the assignment.
You should follow the assignment instructions.

Training linear classifier on SimCLR features



Train feature encoder on **ImageNet** (entire training set) using SimCLR.

Freeze feature encoder, train a linear classifier on top with labeled data.

Source: [Chen et al., 2020]

Semi-supervised learning on SimCLR features

Method	Architecture	Label fraction 1%	Label fraction 10%
		Top 5	
Supervised baseline	ResNet-50	48.4	80.4
Methods using other label-propagation:			
Pseudo-label	ResNet-50	51.6	82.4
VAT+Entropy Min.	ResNet-50	47.0	83.4
UDA (w. RandAug)	ResNet-50	-	88.5
FixMatch (w. RandAug)	ResNet-50	-	89.1
S4L (Rot+VAT+En. M.)	ResNet-50 (4×)	-	91.2
Methods using representation learning only:			
InstDisc	ResNet-50	39.2	77.4
BigBiGAN	RevNet-50 (4×)	55.2	78.8
PIRL	ResNet-50	57.2	83.8
CPC v2	ResNet-161(∗)	77.9	91.2
SimCLR (ours)	ResNet-50	75.5	87.8
SimCLR (ours)	ResNet-50 (2×)	83.0	91.2
SimCLR (ours)	ResNet-50 (4×)	**85.8**	**92.6**

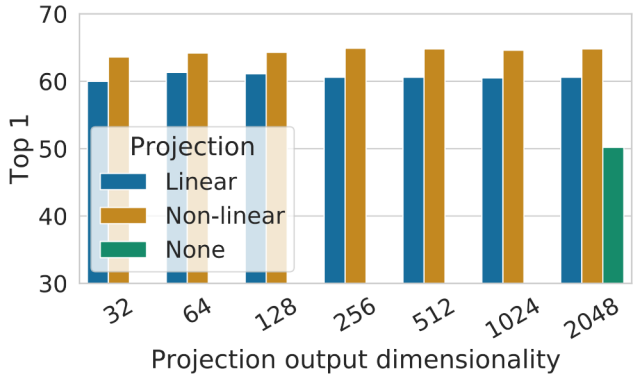
Table 7. ImageNet accuracy of models trained with few labels.

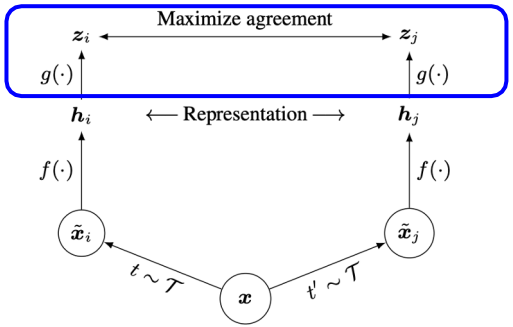
Train feature encoder on **ImageNet** (entire training set) using SimCLR.

Finetune the encoder with 1% / 10% of labeled data on ImageNet.

Source: Chen et al., 2020

SimCLR design choices: projection head





Linear / non-linear projection heads improve representation learning.

A possible explanation:

- contrastive learning objective may discard useful information for downstream tasks
- representation space ***z*** is trained to be invariant to data transformation.
- by leveraging the projection head **g(·)**, more information can be preserved in the **h** representation space

Source: [Chen et al., 2020]

SimCLR design choices: large batch size

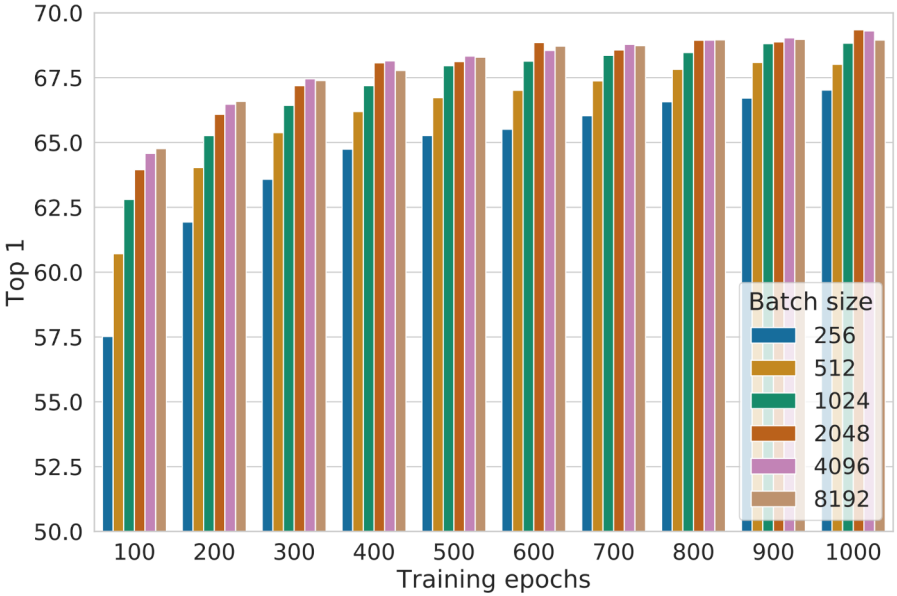


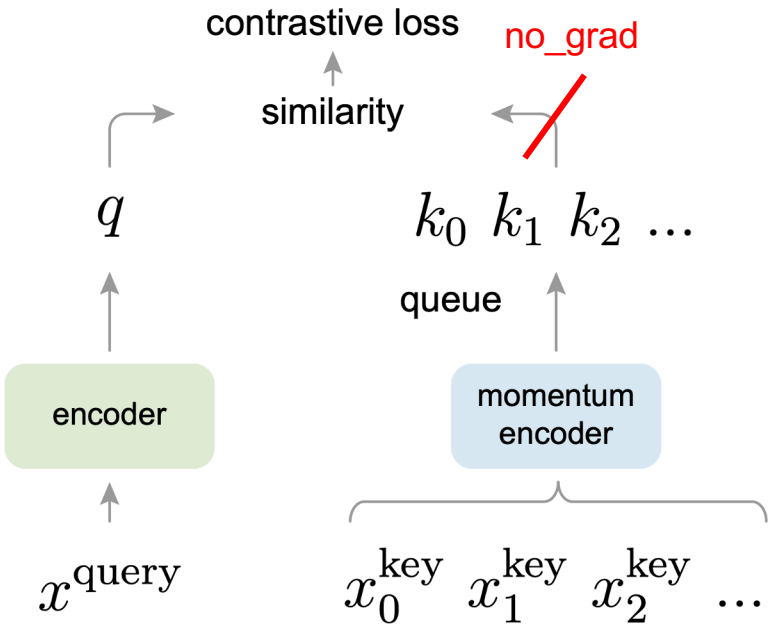
Figure 9. Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.[10]

Large training batch size is crucial for SimCLR!

Large batch size causes large memory footprint during backpropagation: requires distributed training on TPUs (ImageNet experiments)

Source: Chen et al., 2020

Momentum Contrastive Learning (MoCo)

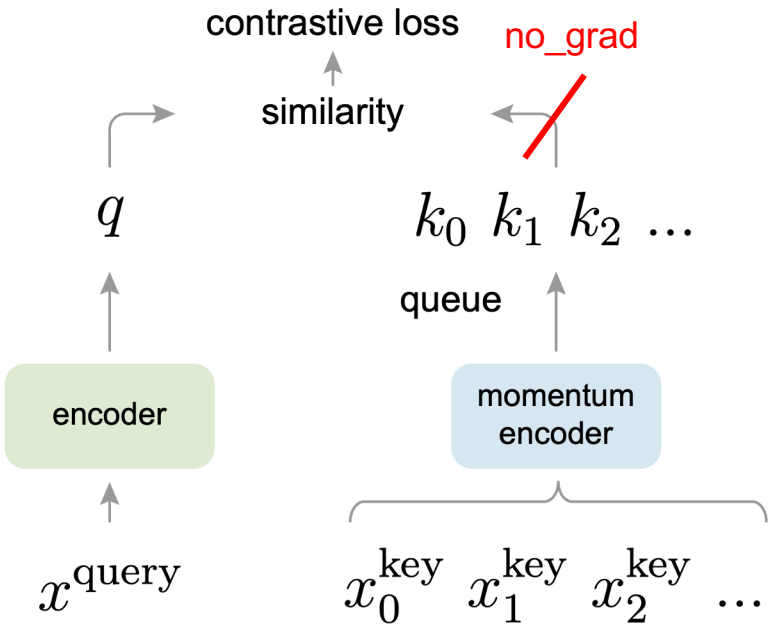


Key differences to SimCLR:

- Keep a running queue of keys (negative samples).
- Compute gradients and update the encoder only through the queries.
- Decouple min-batch size with the number of keys: can support a large number of negative samples.

Source: He et al., 2020

Momentum Contrastive Learning (MoCo)



Key differences to SimCLR:

- Keep a running queue of keys (negative samples).
- Compute gradients and update the encoder only through the queries.
- Decouple min-batch size with the number of keys: can support a large number of negative samples.
- The key encoder is slowly progressing through the momentum update rules:

$$\theta_{\text{k}} \leftarrow m\theta_{\text{k}} + (1-m)\theta_{\text{q}}$$

Source: He et al., 2020

MoCo

Generate a positive pair by sampling data augmentation functions

No gradient through the key

Use the running queue of keys as the negative samples

InfoNCE loss

Update f_k through momentum

Update the FIFO negative sample queue

Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
   x_q = aug(x) # a randomly augmented version
   x_k = aug(x) # another randomly augmented version

   q = f_q.forward(x_q) # queries: NxC
   k = f_k.forward(x_k) # keys: NxC
   k = k.detach() # no gradient to keys

   # positive logits: Nx1
   l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

   # negative logits: NxK
   l_neg = mm(q.view(N,C), queue.view(C,K))

   # logits: Nx(1+K)
   logits = cat([l_pos, l_neg], dim=1)

   # contrastive loss, Eqn.(1)
   labels = zeros(N) # positives are the 0-th
   loss = CrossEntropyLoss(logits/t, labels)

   # SGD update: query network
   loss.backward()
   update(f_q.params)

   # momentum update: key network
   f_k.params = m*f_k.params+(1-m)*f_q.params

   # update dictionary
   enqueue(queue, k) # enqueue the current minibatch
   dequeue(queue) # dequeue the earliest minibatch
```

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.

Source: He et al., 2020

“MoCo V2”

Improved Baselines with Momentum Contrastive Learning

Xinlei Chen Haoqi Fan Ross Girshick Kaiming He
Facebook AI Research (FAIR)

A hybrid of ideas from SimCLR and MoCo:

- **From SimCLR**: non-linear projection head and strong data augmentation.
- **From MoCo**: momentum-updated queues that allow training on a large number of negative samples (no TPU required!).

Source: Chen et al., 2020

MoCo vs. SimCLR vs. MoCo V2

case	unsup. pre-train MLP	aug+	cos	epochs	ImageNet acc.	VOC detection AP_{50}	AP	AP_{75}
supervised					76.5	81.3	53.5	58.8
MoCo v1				200	60.6	81.5	55.9	62.6
(a)	✓			200	66.2	82.0	56.4	62.6
(b)		✓		200	63.4	82.2	56.8	63.2
(c)	✓	✓		200	67.3	**82.5**	57.2	63.9
(d)	✓	✓	✓	200	67.5	82.4	57.0	63.6
(e)	✓	✓	✓	**800**	**71.1**	**82.5**	**57.4**	**64.0**

Table 1. **Ablation of MoCo baselines**, evaluated by ResNet-50 for (i) ImageNet linear classification, and (ii) fine-tuning VOC object detection (mean of 5 trials). "**MLP**": with an MLP head; "**aug+**": with extra blur augmentation; "**cos**": cosine learning rate schedule.

Key takeaways:

- Non-linear projection head and strong data augmentation are crucial for contrastive learning.

Source: [Chen et al., 2020]

MoCo vs. SimCLR vs. MoCo V2

case	unsup. pre-train MLP	aug+	cos	epochs	batch	ImageNet acc.
MoCo v1 [6]				200	256	60.6
SimCLR [2]	✓	✓	✓	200	256	61.9
SimCLR [2]	✓	✓	✓	200	8192	66.6
MoCo v2	✓	✓	✓	200	256	**67.5**
*results of **longer** unsupervised training follow:*						
SimCLR [2]	✓	✓	✓	1000	4096	69.3
MoCo v2	✓	✓	✓	800	256	**71.1**

Table 2. **MoCo *vs.* SimCLR**: ImageNet linear classifier accuracy (**ResNet-50, 1-crop 224×224**), trained on features from unsupervised pre-training. "aug+" in SimCLR includes blur and stronger color distortion. SimCLR ablations are from Fig. 9 in [2] (we thank the authors for providing the numerical results).

Key takeaways:

- Non-linear projection head and strong data augmentation are crucial for contrastive learning.
- Decoupling mini-batch size with negative sample size allows MoCo-V2 to outperform SimCLR with smaller batch size (256 vs. 8192).

Source: Chen et al., 2020

MoCo vs. SimCLR vs. MoCo V2

mechanism	batch	memory / GPU	time / 200-ep.
MoCo	256	**5.0G**	**53 hrs**
end-to-end	256	7.4G	65 hrs
end-to-end	4096	93.0G†	n/a

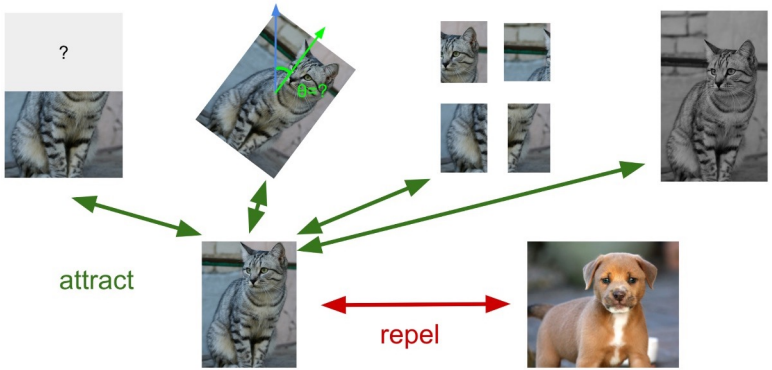
Table 3. **Memory and time cost** in 8 V100 16G GPUs, implemented in PyTorch. †: based on our estimation.

Key takeaways:

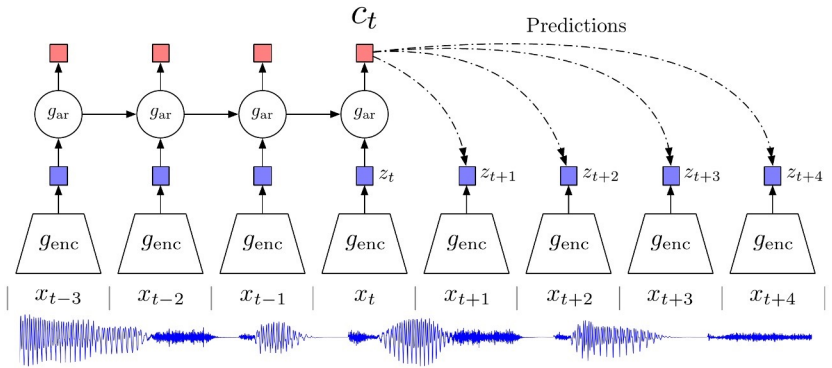
- Non-linear projection head and strong data augmentation are crucial for contrastive learning.
- Decoupling mini-batch size with negative sample size allows MoCo-V2 to outperform SimCLR with smaller batch size (256 vs. 8192).
- … all with much smaller memory footprint! (“end-to-end” means SimCLR here)

Source: Chen et al., 2020

Instance vs. Sequence Contrastive Learning



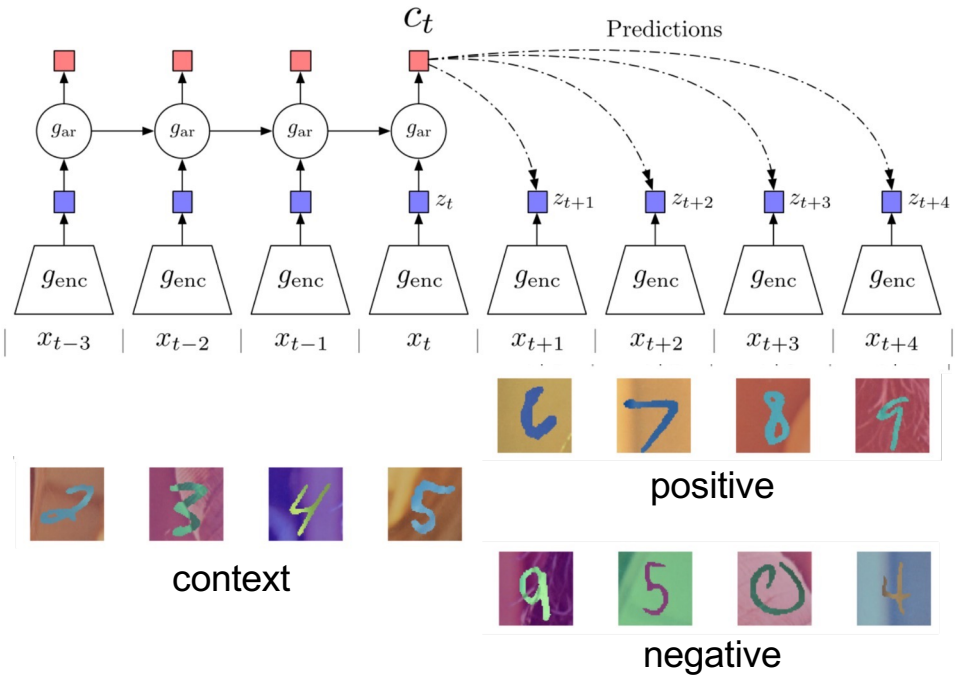
Instance-level contrastive learning:
contrastive learning based on
positive & negative instances.
Examples: SimCLR, MoCo



Source: van den Oord et al., 2018

Sequence-level contrastive learning:
contrastive learning based on
sequential / temporal orders.
Example: **Contrastive Predictive Coding (CPC)**

Contrastive Predictive Coding (CPC)



Contrastive: contrast between "right" and "wrong" sequences using contrastive learning.

Predictive: the model has to predict future patterns given the current context.

Coding: the model learns useful feature vectors, or "code", for downstream tasks, similar to other self-supervised methods.

Figure source

Source: van den Oord et al., 2018,

Contrastive Predictive Coding (CPC)

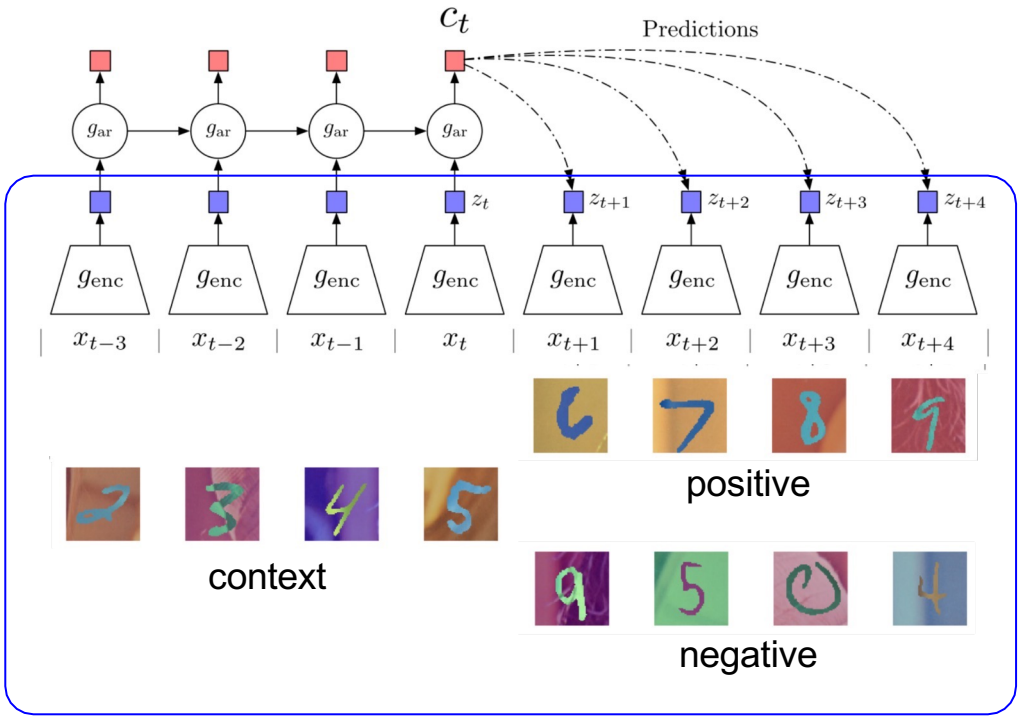
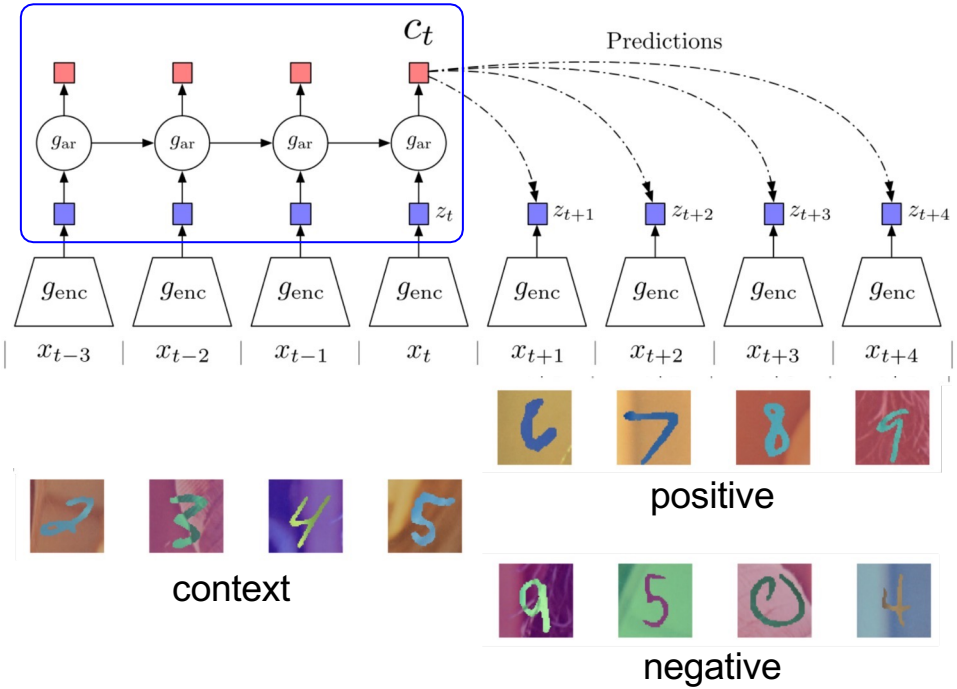


Figure [source]

1. Encode all samples in a sequence into vectors $z_t = g_{enc}(x_t)$

Source: [van den Oord et al., 2018],

Contrastive Predictive Coding (CPC)



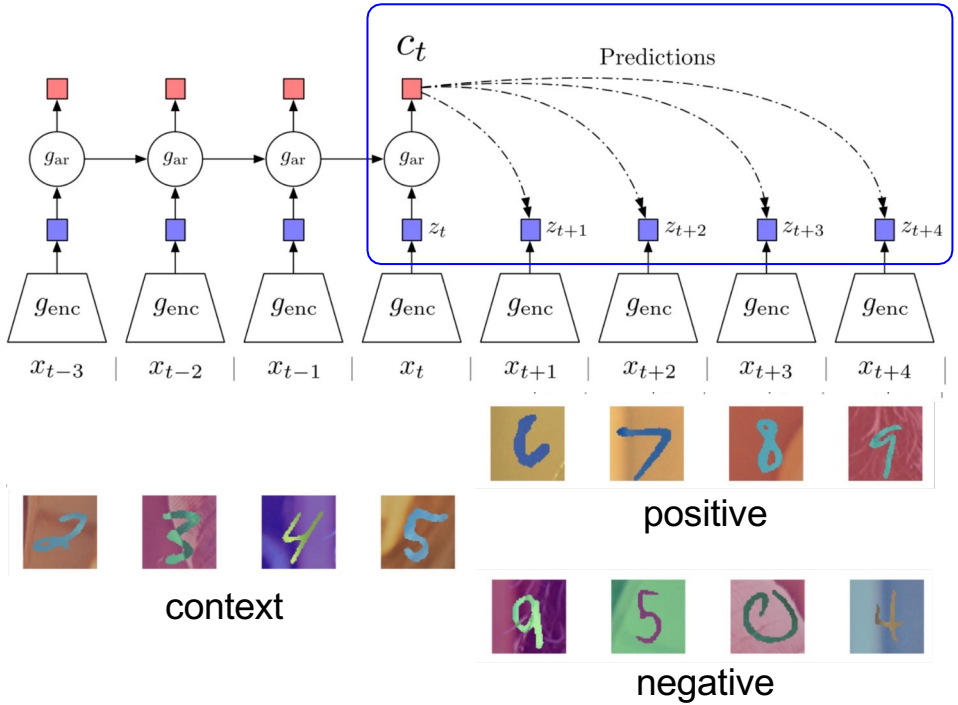
1. Encode all samples in a sequence into vectors $z_t = g_{enc}(x_t)$

2. Summarize context (e.g., half of a sequence) into a context code c_t using an auto-regressive model (g_{ar}). The original paper uses GRU-RNN here.

Figure source

Source: van den Oord et al., 2018,

Contrastive Predictive Coding (CPC)



1. Encode all samples in a sequence into vectors $z_t = g_{enc}(x_t)$

2. Summarize context (e.g., half of a sequence) into a context code c_t using an auto-regressive model (g_{ar})

3. Compute InfoNCE loss between the context c_t and future code z_{t+k} using the following time-dependent score function:

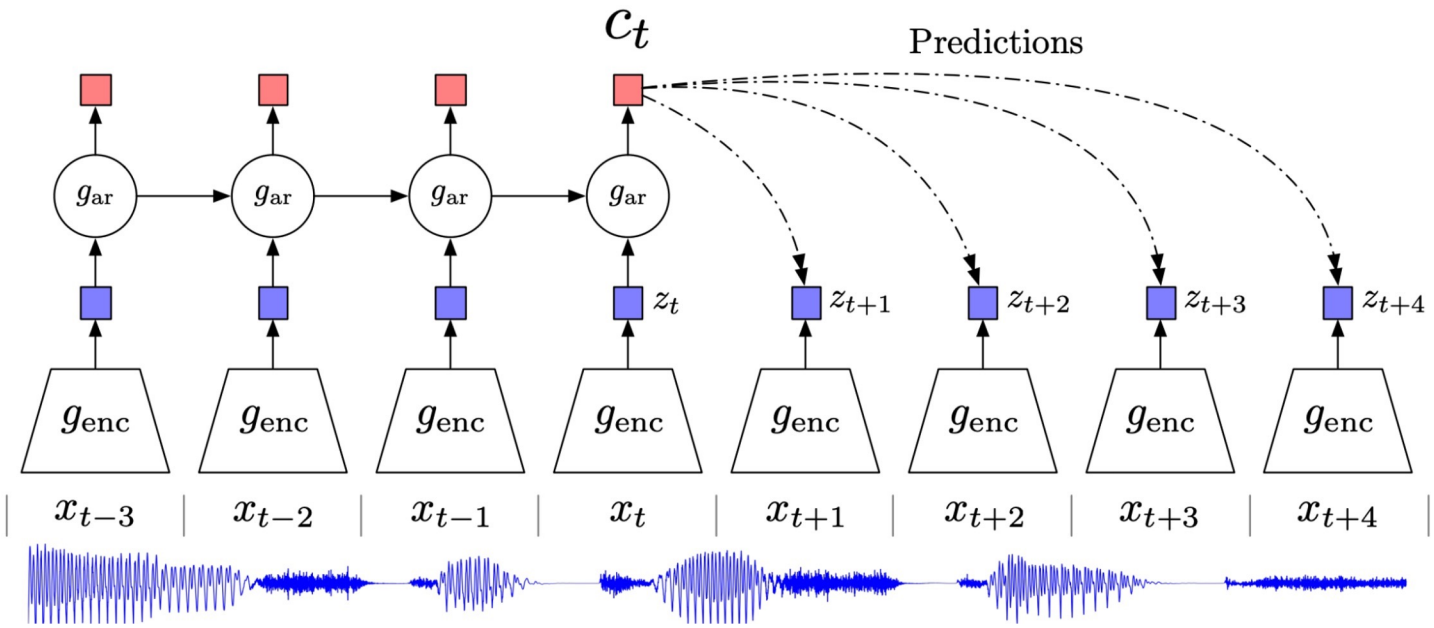
$$s_k(z_{t+k}, c_t) = z_{t+k}^T W_k c_t$$

, where W_k is a trainable matrix.

Figure source

Source: van den Oord et al., 2018,

CPC example: modeling audio sequences



Source: [van den Oord et al., 2018](),

CPC example: modeling audio sequences

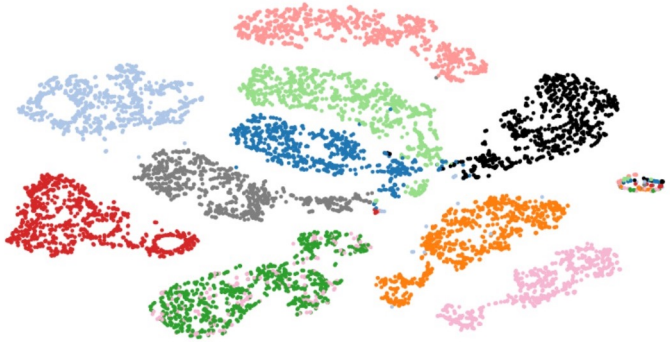


Figure 2: t-SNE visualization of audio (speech) representations for a subset of 10 speakers (out of 251). Every color represents a different speaker.

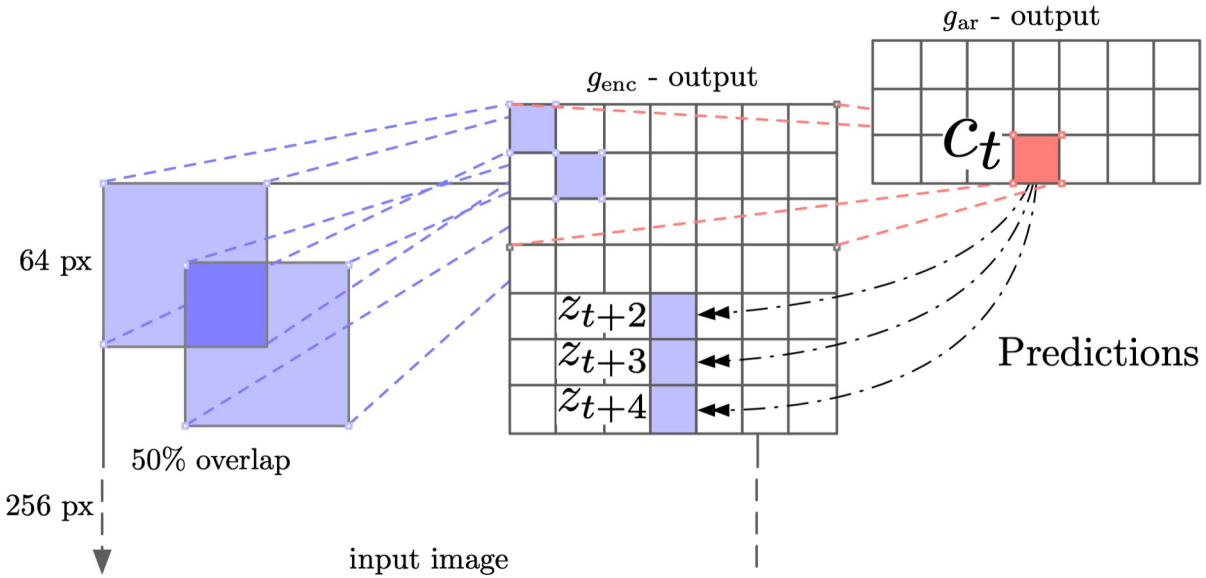
Method	ACC
Phone classification	
Random initialization	27.6
MFCC features	39.7
CPC	64.6
Supervised	74.6
Speaker classification	
Random initialization	1.87
MFCC features	17.6
CPC	97.4
Supervised	98.5

Linear classification on trained representations (LibriSpeech dataset)

Source: van den Oord et al., 2018,

CPC example: modeling visual context

Idea: split image into patches, model rows of patches from top to bottom as a sequence. I.e., use top rows as context to predict bottom rows.



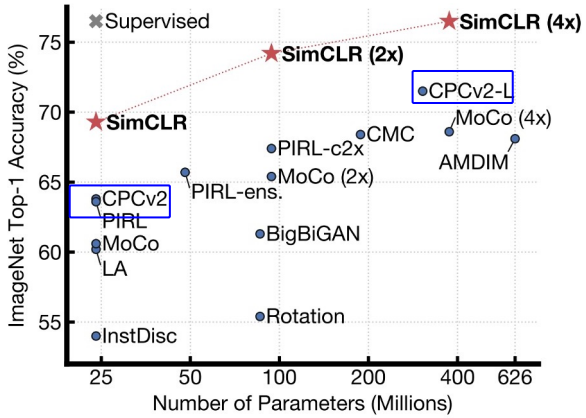
Source: [van den Oord et al., 2018](),

CPC example: modeling visual context

Method	Top-1 ACC
Using AlexNet conv5	
Video [28]	29.8
Relative Position [11]	30.4
BiGan [35]	34.8
Colorization [10]	35.2
Jigsaw [29] *	38.1
Using ResNet-V2	
Motion Segmentation [36]	27.6
Exemplar [36]	31.5
Relative Position [36]	36.2
Colorization [36]	39.6
CPC	**48.7**

Table 3: ImageNet top-1 unsupervised classification results. *Jigsaw is not directly comparable to the other AlexNet results because of architectural differences.

- Compares favorably with other pretext task-based self-supervised learning method.
- Doesn't do as well compared to newer instance-based contrastive learning methods on image feature learning.



Source: van den Oord et al., 2018,

Summary: Contrastive Representation Learning

A general formulation for contrastive learning:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

InfoNCE loss: N-way classification among positive and negative samples

$$L = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

Commonly known as the InfoNCE loss (van den Oord et al., 2018)

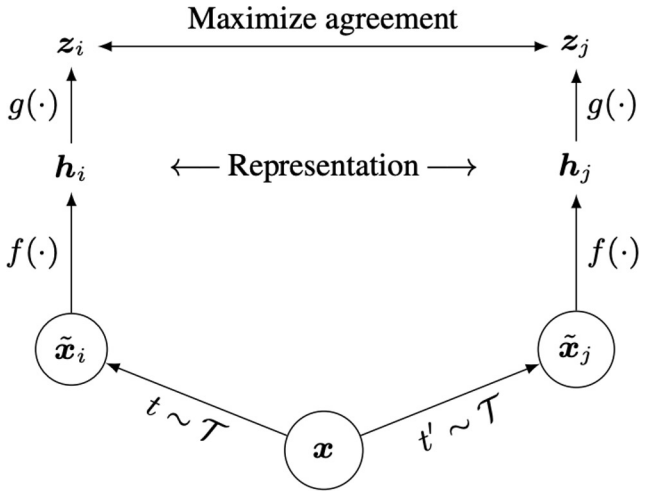
A *lower bound* on the mutual information between *f(x)* and *f(x⁺)*

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

Summary: Contrastive Representation Learning

SimCLR: a simple framework for contrastive representation learning

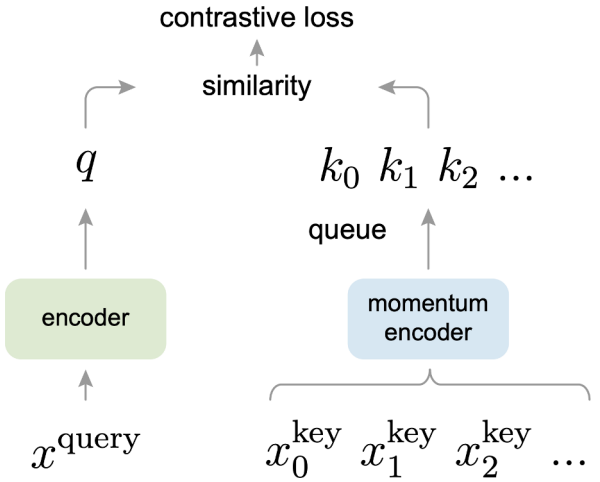
- **Key ideas**: non-linear projection head to allow flexible representation learning
- Simple to implement, effective in learning visual representation
- Requires large training batch size to be effective; large memory footprint



Summary: Contrastive Representation Learning

MoCo (v1, v2): contrastive learning using momentum sample encoder

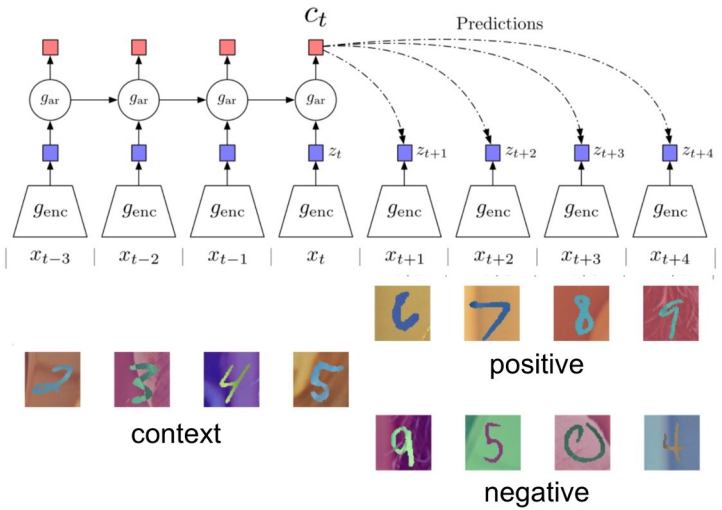
- Decouples negative sample size from minibatch size; allows large batch training without TPU
- MoCo-v2 combines the key ideas from SimCLR, i.e., nonlinear projection head, strong data augmentation, with momentum contrastive learning



Summary: Contrastive Representation Learning

CPC: sequence-level contrastive learning

- Contrast “right” sequence with “wrong” sequence.
- InfoNCE loss with a time-dependent score function.
- Can be applied to a variety of learning problems, but not as effective in learning image representations compared to instance-level methods.



Other examples: MoCo v3

"This paper does not describe a novel method."

An Empirical Study of Training Self-Supervised Vision Transformers

Xinlei Chen* Saining Xie* Kaiming He
Facebook AI Research (FAIR)

Code: https://github.com/facebookresearch/moco-v3

Abstract

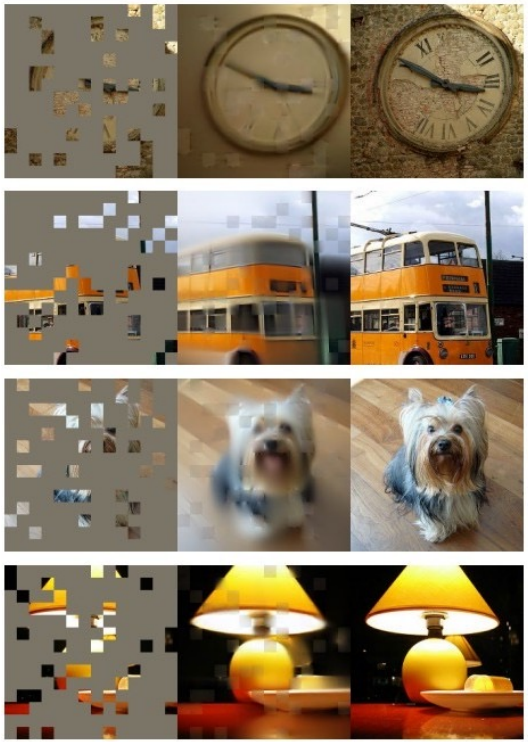
This paper does not describe a novel method. Instead, it studies a straightforward, incremental, yet must-know baseline given the recent progress in computer vision: self-supervised learning for Vision Transformers (ViT). While the training recipes for standard convolutional networks have been highly mature and robust, the recipes for ViT are yet to be built, especially in the self-supervised scenarios where training becomes more challenging. In this work, we go back to basics and investigate the effects of several fundamental components for training self-supervised ViT. We observe that instability is a major issue that degrades accuracy, and it can be hidden by apparently good results. We reveal that these results are indeed partial failure, and they can be improved when training is made more stable. We benchmark ViT results in MoCo v3 and several other self-supervised frameworks, with ablations in various aspects. We discuss the currently positive evidence as well as challenges and open questions. We hope that this work will provide useful data points and experience for future research.

framework	model	params	acc. (%)
linear probing:			
iGPT [9]	iGPT-L	1362M	69.0
iGPT [9]	iGPT-XL	6801M	72.0
MoCo v3	ViT-B	86M	76.7
MoCo v3	ViT-L	304M	77.6
MoCo v3	ViT-H	632M	78.1
MoCo v3	ViT-BN-H	632M	79.1
MoCo v3	ViT-BN-L/7	304M	**81.0**
end-to-end fine-tuning:			
masked patch pred. [16]	ViT-B	86M	79.9†
MoCo v3	ViT-B	86M	83.2
MoCo v3	ViT-L	304M	**84.1**

Table 1. **State-of-the-art Self-supervised Transformers** in ImageNet classification, evaluated by linear probing (top panel) or end-to-end fine-tuning (bottom panel). Both iGPT [9] and masked patch prediction [16] belong to the masked auto-encoding paradigm. MoCo v3 is a contrastive learning method that compares two (224×224) crops. ViT-B, -L, -H are the Vision Transformers proposed in [16]. ViT-BN is modified with BatchNorm, and "/7" denotes a patch size of 7×7. †: pre-trained in JFT-300M.

Chen et al., An Empirical Study of Training Self-Supervised Vision Transformers, FAIR

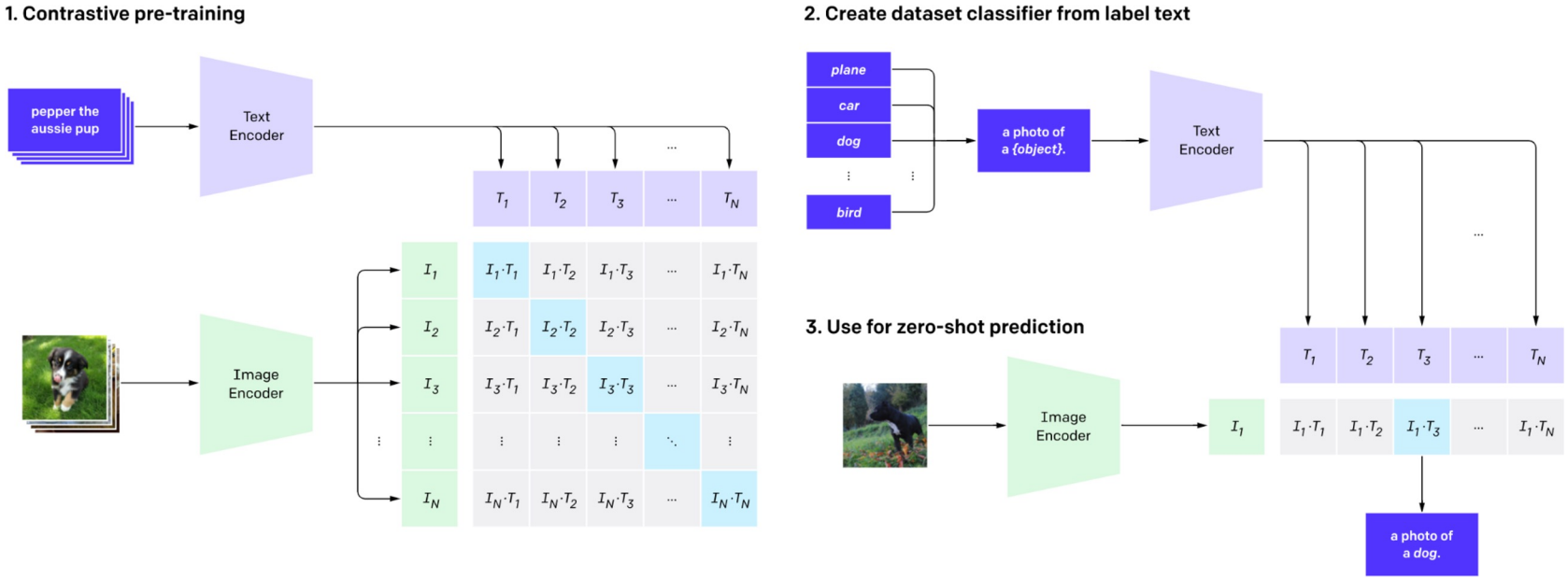
Other examples: Masked Autoencoder



method	pre-train data	ViT-B	ViT-L	ViT-H	ViT-H_{448}
scratch, our impl.	-	82.3	82.6	83.1	-
DINO [5]	IN1K	82.8	-	-	-
MoCo v3 [9]	IN1K	83.2	84.1	-	-
BEiT [2]	IN1K+DALLE	83.2	85.2	-	-
MAE	IN1K	83.6	85.9	86.9	**87.8**

He et al., Masked Autoencoders Are Scalable Vision Learners, FAIR

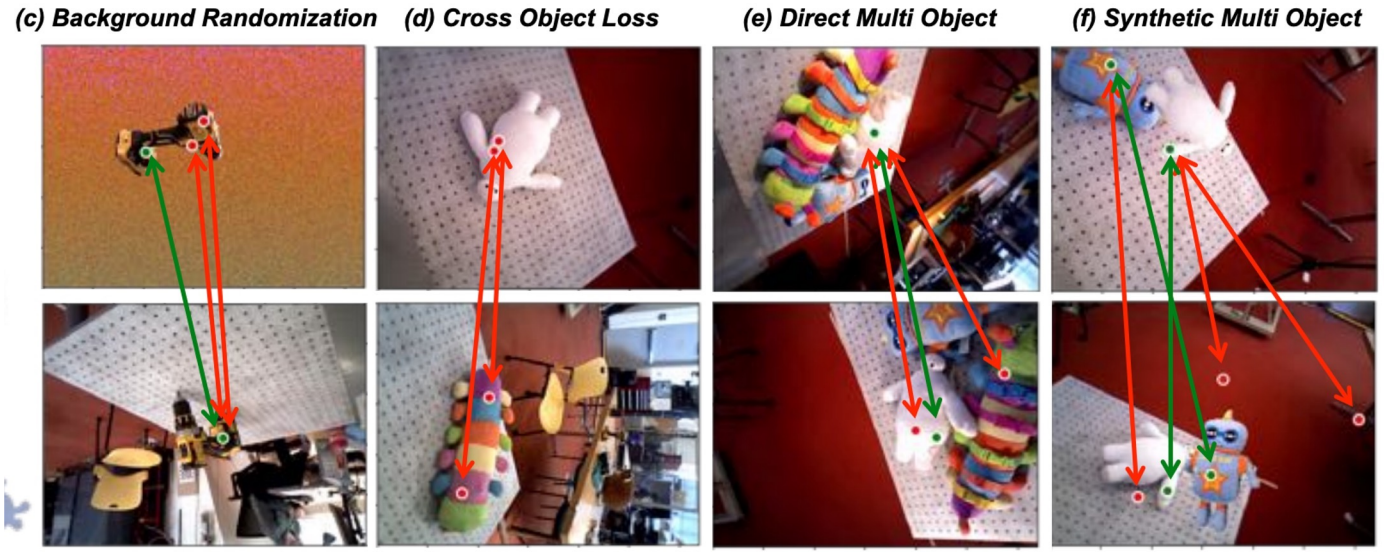
Other examples: CLIP



CLIP (*Contrastive Language–Image Pre-training*) Radford *et al.*, 2021

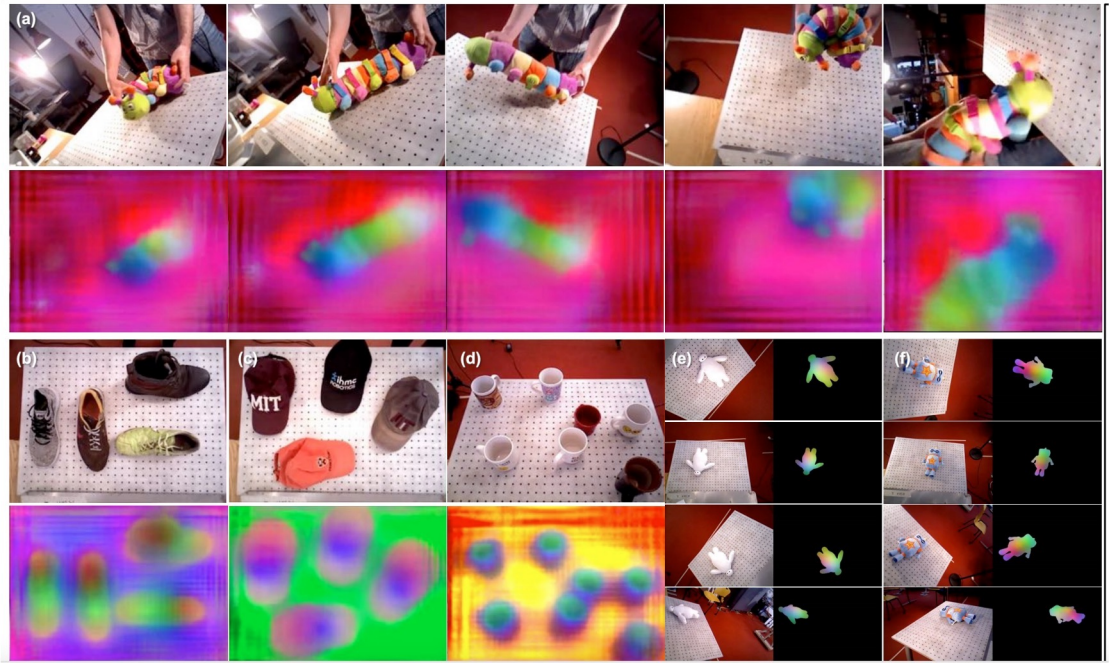
Other examples: Dense Object Net

Contrastive learning on pixel-wise feature descriptors



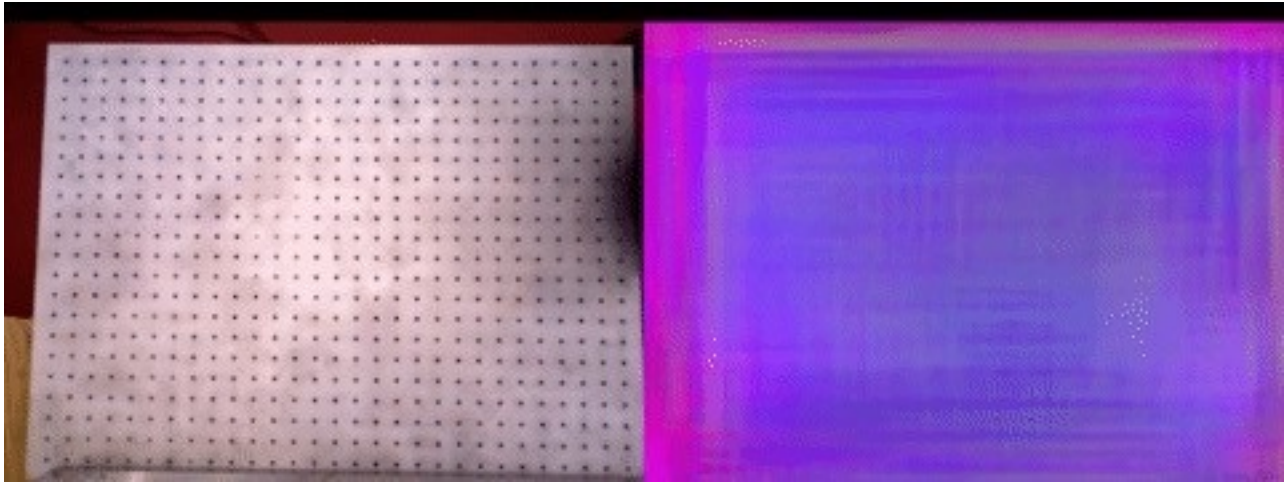
Dense Object Net, Florence et al., 2018

Other examples: Dense Object Net



Dense Object Net, Florence et al., 2018

Other examples: Dense Object Net



Dense Object Net, Florence et al., 2018

Other examples: DINO

Emerging Properties in Self-Supervised Vision Transformers

Mathilde Caron[1,2] Hugo Touvron[1,3] Ishan Misra[1] Hervé Jegou[1]
Julien Mairal[2] Piotr Bojanowski[1] Armand Joulin[1]

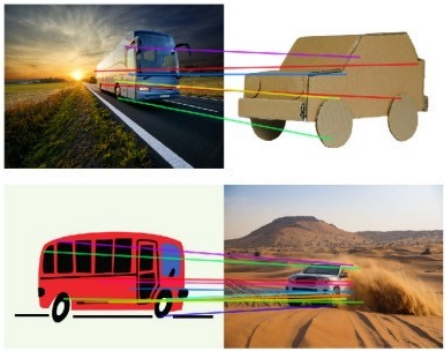
[1] Facebook AI Research [2] Inria* [3] Sorbonne University

Figure 1: **Self-attention from a Vision Transformer with 8×8 patches trained with no supervision.** We look at the self-attention of the `[CLS]` token on the heads of the last layer. This token is not attached to any label nor supervision. These maps show that the model automatically learns class-specific features leading to unsupervised object segmentations.

Other examples: DINO v2

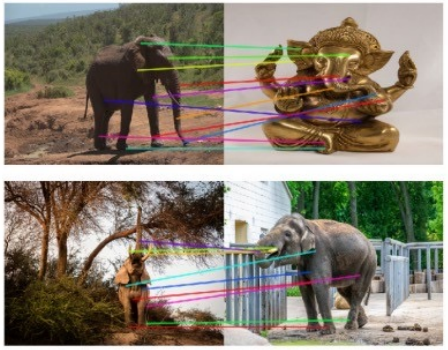
Figure 1: **Visualization of the first PCA components.** We compute a PCA between the patches of the images from the same column (a, b, c and d) and show their first 3 components. Each component is matched to a different color channel. Same parts are matched between related images despite changes of pose, style or even objects. Background is removed by thresholding the first PCA component.

Other examples: DINO v2

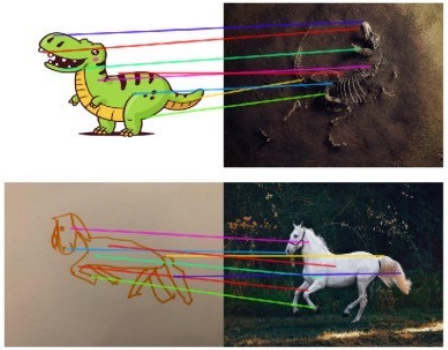


(Vehicles)

(Birds / Airplanes)

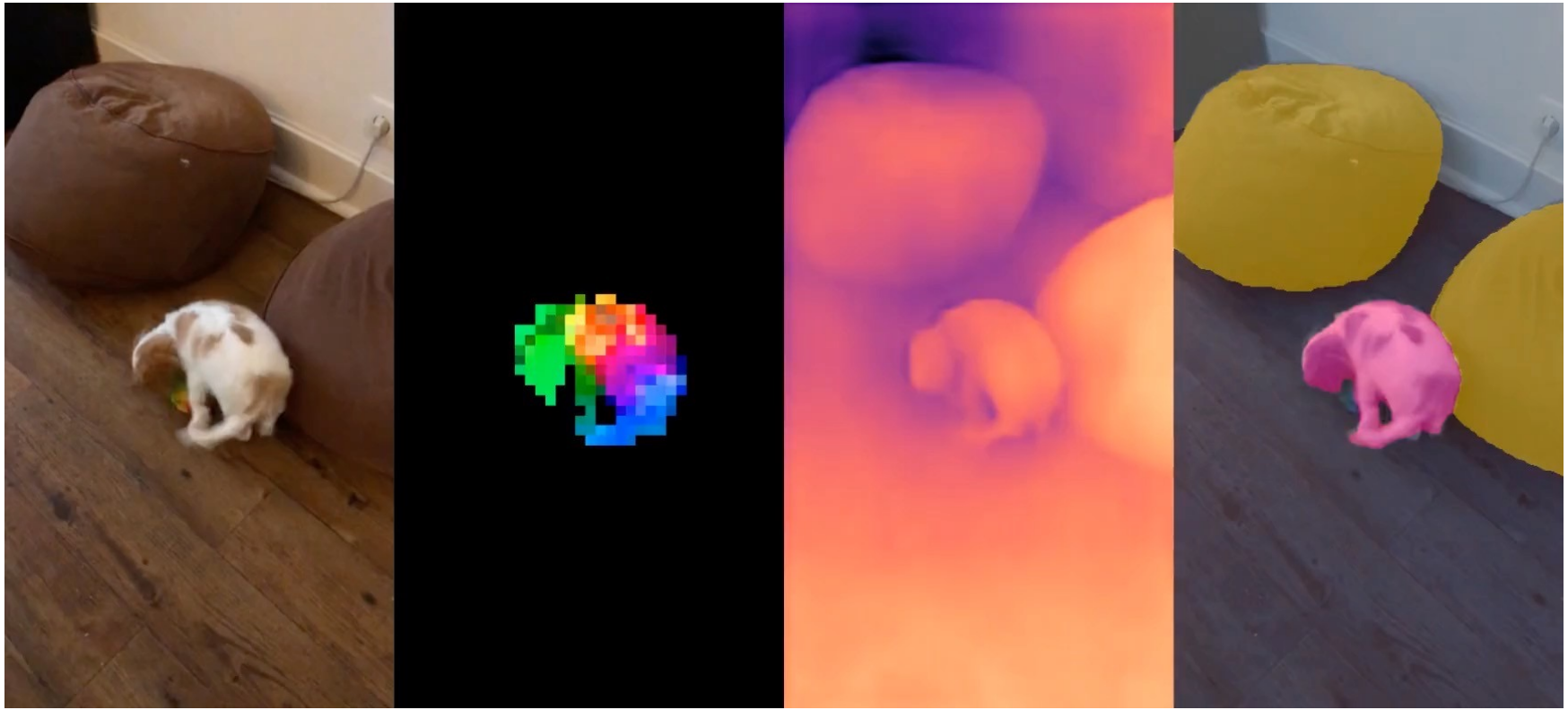


(Elephants)



(Drawings / Animals)

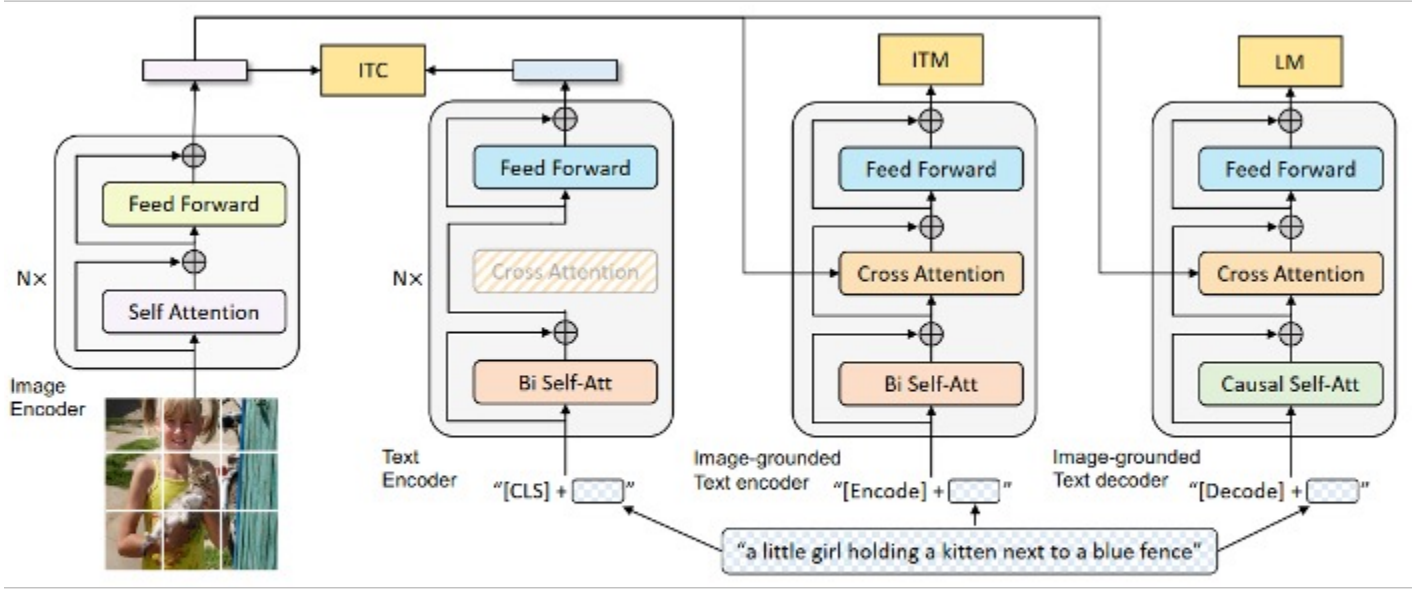
Other examples: DINO v2



Large Multi-modal Models

Large Multi-modal Models (BLIP)

- Pretraining Architecture and Objectives

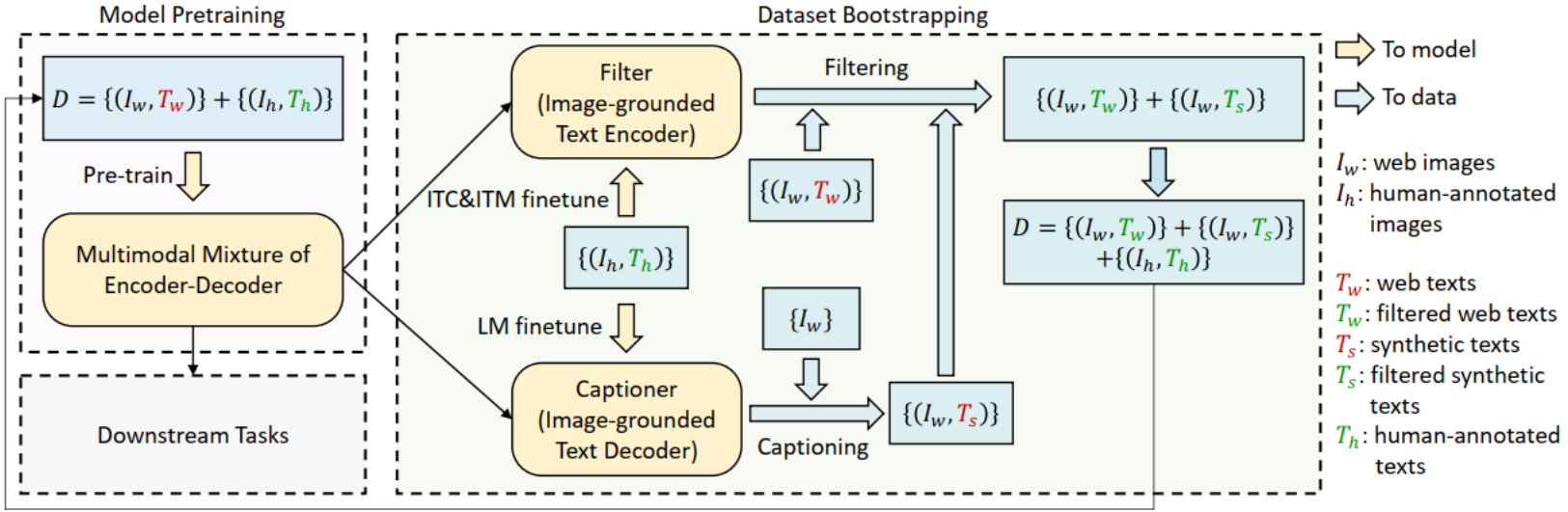


ITC: Image-Text Contrastive Learning; **ITM**: Image-Text Match Learning; **LM**: Language Modelling

Li, Junnan, et al. "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation." International Conference on Machine Learning. PMLR, 2022.

Large Multi-modal Models (BLIP)

- Learning Framework of BLIP



Large Multi-modal Models (BLIP)

- Evaluation of the effect of the captioner (C) and filter (F) for dataset bootstrapping.

Pre-train dataset	Bootstrap C	Bootstrap F	Vision backbone	Retrieval-FT (COCO) TR@1	Retrieval-FT (COCO) IR@1	Retrieval-ZS (Flickr) TR@1	Retrieval-ZS (Flickr) IR@1	Caption-FT (COCO) B@4	Caption-FT (COCO) CIDEr	Caption-ZS (NoCaps) CIDEr	Caption-ZS (NoCaps) SPICE
COCO+VG +CC+SBU (14M imgs)	✗	✗	ViT-B/16	78.4	60.7	93.9	82.1	38.0	127.8	102.2	13.9
	✗	✓$_B$		79.1	61.5	94.1	82.8	38.1	128.2	102.7	14.0
	✓$_B$	✗		79.7	62.0	94.4	83.6	38.4	128.9	103.4	14.2
	✓$_B$	✓$_B$		80.6	63.1	94.8	84.9	38.6	129.7	105.1	14.4
COCO+VG +CC+SBU +LAION (129M imgs)	✗	✗	ViT-B/16	79.6	62.0	94.3	83.6	38.8	130.1	105.4	14.2
	✓$_B$	✓$_B$		81.9	64.3	96.0	85.0	39.4	131.4	106.3	14.3
	✓$_L$	✓$_L$		81.2	64.1	96.0	85.5	39.7	133.3	109.6	14.7
	✗	✗	ViT-L/16	80.6	64.1	95.1	85.5	40.3	135.5	112.5	14.7
	✓$_L$	✓$_L$		82.4	65.1	96.7	86.7	40.4	136.7	113.2	14.8

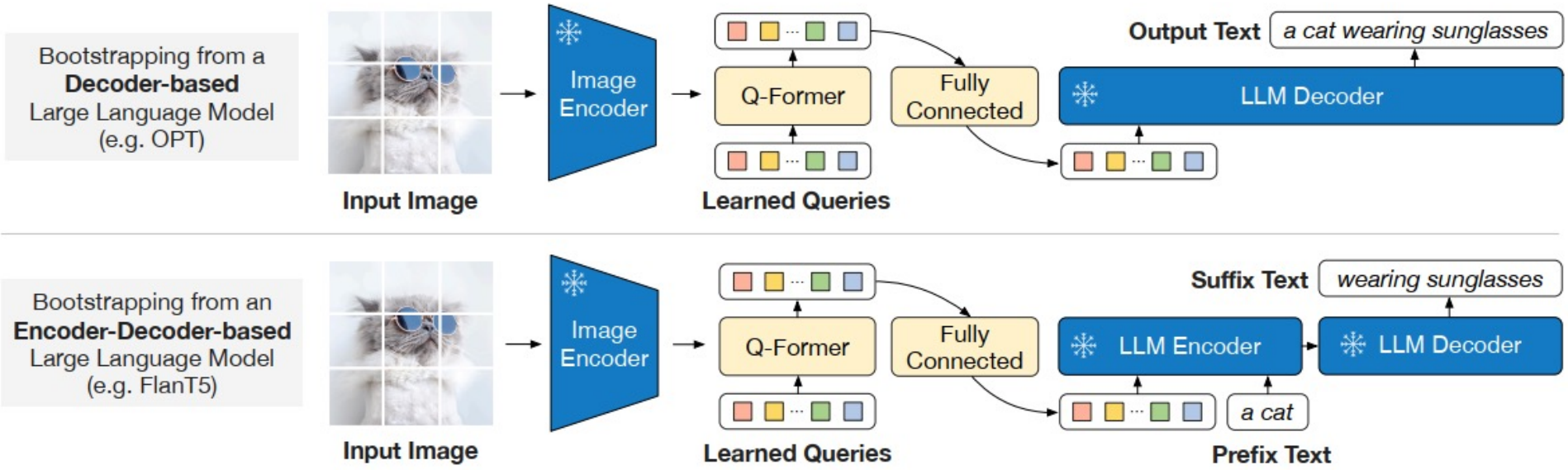
Downstream tasks include image-text retrieval and image captioning with finetuning (FT) and zero-shot (ZS) settings.

- Effect of sharing parameters between the captioner and filter.

Captioner & Filter	Noise ratio	Retrieval-FT (COCO) TR@1	Retrieval-FT (COCO) IR@1	Retrieval-ZS (Flickr) TR@1	Retrieval-ZS (Flickr) IR@1	Caption-FT (COCO) B@4	Caption-FT (COCO) CIDEr	Caption-ZS (NoCaps) CIDEr	Caption-ZS (NoCaps) SPICE
Share parameters	8%	79.8	62.2	94.3	83.7	38.4	129.0	103.5	14.2
Decoupled	25%	80.6	63.1	94.8	84.9	38.6	129.7	105.1	14.4

Large Multi-modal Models (BLIP-2)

- Pretraining Pipeline

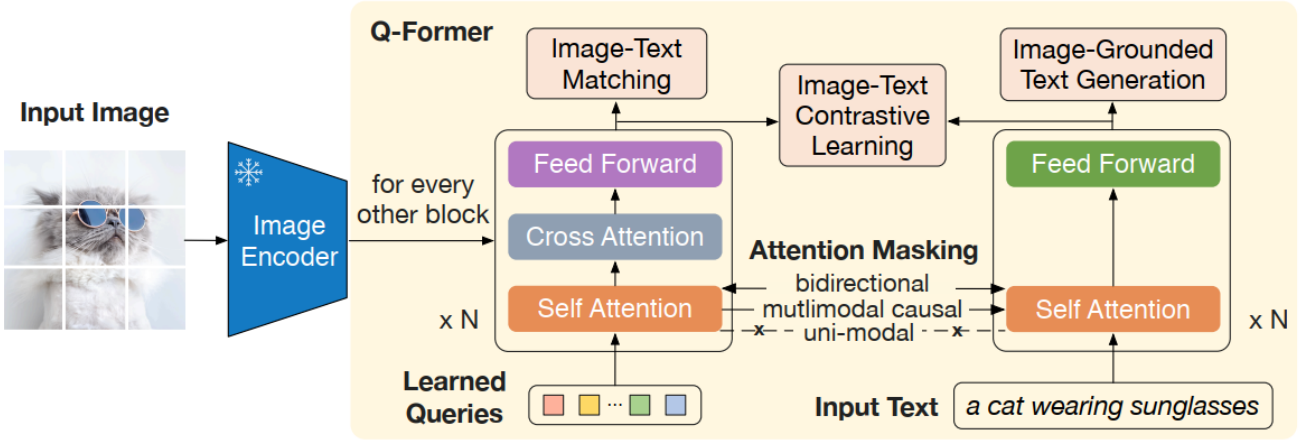


Li, Junnan, et al. "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models." *arXiv preprint arXiv:2301.12597* (2023).

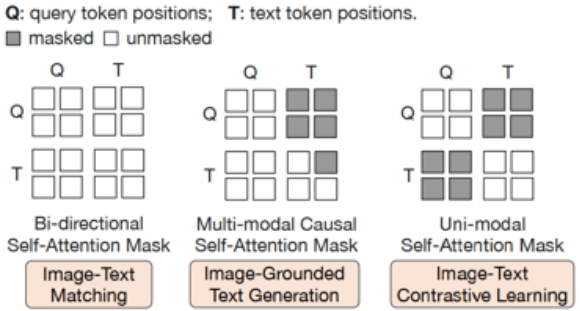
Large Multi-modal Models (BLIP-2)

- Pretraining Pipeline

Model architecture of Q-Former and BLIP-2's first-stage vision-language representation learning objectives.



The self-attention masking strategy for each objective to control query-text interaction.



Large Multi-modal Models (BLIP-2)

- Results on Zero-shot Vision-Language Models

Overview of BLIP-2 results on various zero-shot vision-language tasks.

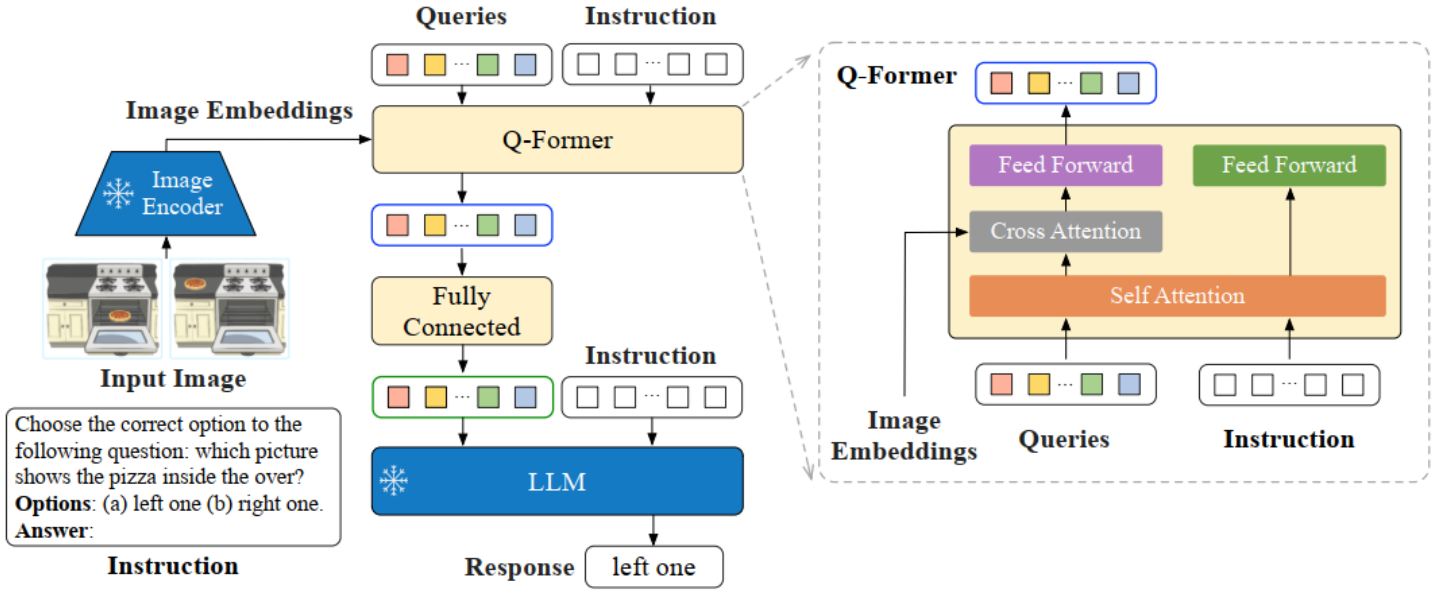
Models	#Trainable Params	Open-sourced?	Visual Question Answering VQAv2 (test-dev) VQA acc.	Image Captioning NoCaps (val) CIDEr	Image Captioning NoCaps (val) SPICE	Image-Text Retrieval Flickr (test) TR@1	Image-Text Retrieval Flickr (test) IR@1
BLIP (Li et al., 2022)	583M	✓	-	113.2	14.8	96.7	86.7
SimVLM (Wang et al., 2021b)	1.4B	✗	-	112.2	-	-	-
BEIT-3 (Wang et al., 2022b)	1.9B	✗	-	-	-	94.9	81.5
Flamingo (Alayrac et al., 2022)	10.2B	✗	56.3	-	-	-	-
BLIP-2	188M	✓	**65.0**	**121.6**	**15.8**	**97.6**	**89.7**

Comparison with state-of-the-art methods on zero-shot visual question answering.

Models	#Trainable Params	#Total Params	VQAv2 val	VQAv2 test-dev	OK-VQA test	GQA test-dev
VL-T5$_{no\text{-}vqa}$	224M	269M	13.5	-	5.8	6.3
FewVLM (Jin et al., 2022)	740M	785M	47.7	-	16.5	29.3
Frozen (Tsimpoukelli et al., 2021)	40M	7.1B	29.6	-	5.9	-
VLKD (Dai et al., 2022)	406M	832M	42.6	44.5	13.3	-
Flamingo3B (Alayrac et al., 2022)	1.4B	3.2B	-	49.2	41.2	-
Flamingo9B (Alayrac et al., 2022)	1.8B	9.3B	-	51.8	44.7	-
Flamingo80B (Alayrac et al., 2022)	10.2B	80B	-	56.3	**50.6**	-
BLIP-2 ViT-L OPT$_{2.7B}$	104M	3.1B	50.1	49.7	30.2	33.9
BLIP-2 ViT-G OPT$_{2.7B}$	107M	3.8B	53.5	52.3	31.7	34.6
BLIP-2 ViT-G OPT$_{6.7B}$	108M	7.8B	54.3	52.6	36.4	36.4
BLIP-2 ViT-L FlanT5$_{XL}$	103M	3.4B	62.6	62.3	39.4	44.4
BLIP-2 ViT-G FlanT5$_{XL}$	107M	4.1B	63.1	63.0	40.7	44.2
BLIP-2 ViT-G FlanT5$_{XXL}$	108M	12.1B	**65.2**	**65.0**	45.9	**44.7**

Large Multi-modal Models (InstructBLIP)

- Model Architecture



Dai, Wenliang, et al. "InstructBLIP: Towards General-purpose Vision-Language Models with Instruction Tuning." *arXiv preprint arXiv:2305.06500* (2023).

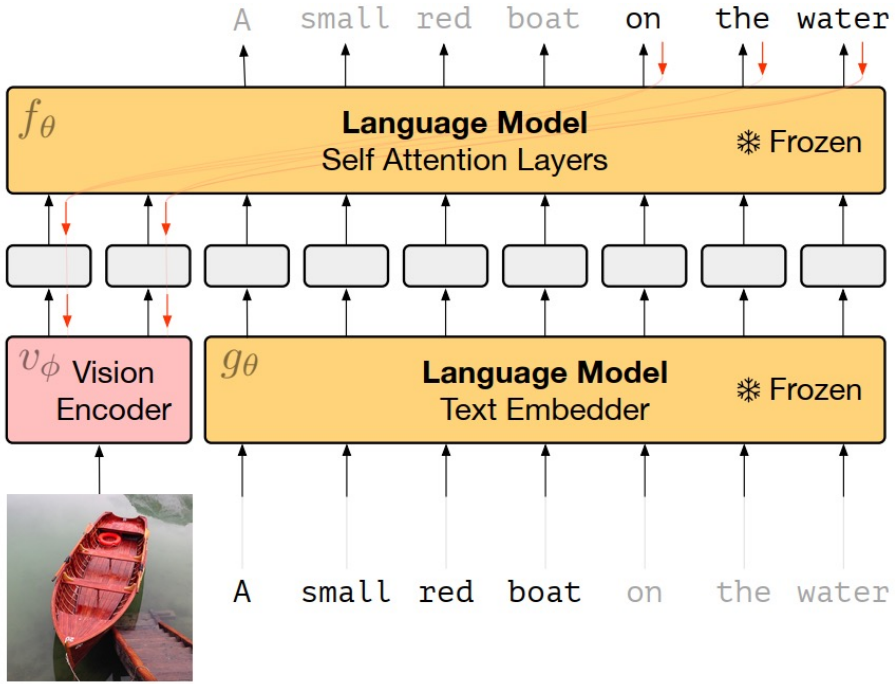
Large Multi-modal Models (InstructBLIP)

- Comparison between BLIP-2 and InstructBLIP

	ScienceQA IMG	OCR-VQA	OKVQA	A-OKVQA Direct Answer Val	A-OKVQA Direct Answer Test	A-OKVQA Multi-choice Val	A-OKVQA Multi-choice Test
Previous SOTA	LLaVA [25] 89.0	GIT [42] 70.3	PaLM-E(562B) [9] **66.1**	[15] 56.3	[36] 61.6	[15] 73.2	[36] 73.6
BLIP-2 (FlanT5$_{XXL}$)	89.5	72.7	54.7	57.6	53.7	80.2	76.2
InstructBLIP (FlanT5$_{XXL}$)	**90.7**	**73.3**	55.5	57.1	54.8	**81.0**	**76.7**
BLIP-2 (Vicuna-7B)	77.3	69.1	59.3	60.0	58.7	72.1	69.0
InstructBLIP (Vicuna-7B)	79.5	72.8	62.1	**64.0**	**62.1**	75.7	73.4

Large Multi-modal Models (Frozen)

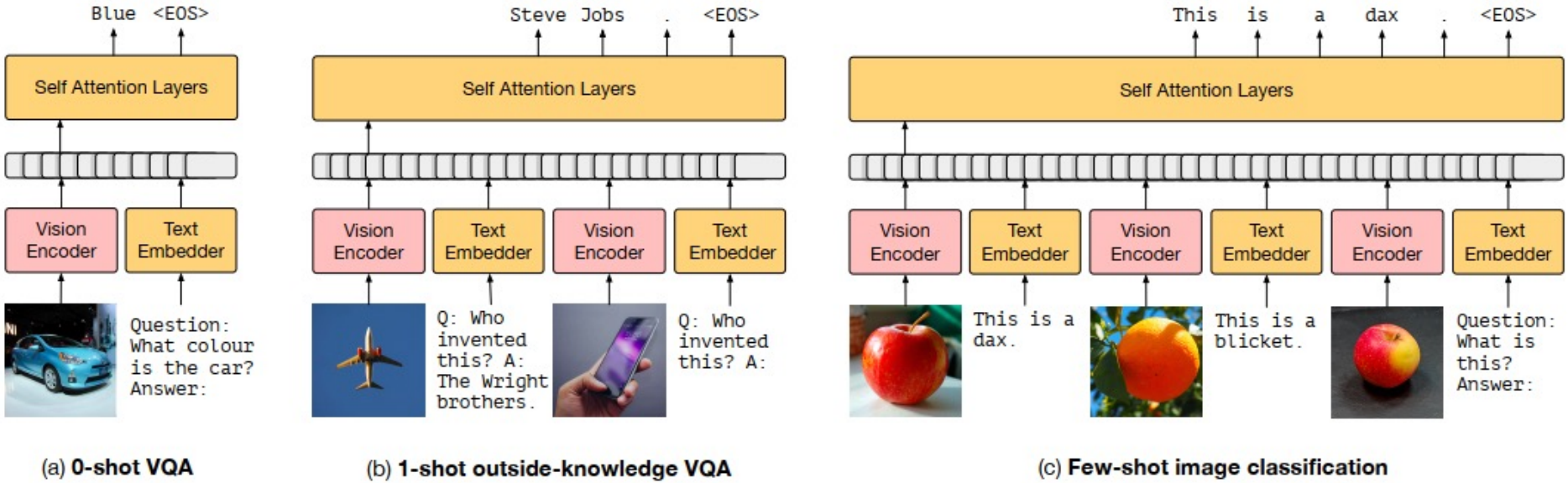
- Model Architecture



Tsimpoukelli, Maria, et al. "Multimodal few-shot learning with frozen language models." Advances in Neural Information Processing Systems 34 (2021): 200-212.

Large Multi-modal Models (Frozen)

- Inference-time Interface



(a) **0-shot VQA** (b) **1-shot outside-knowledge VQA** (c) **Few-shot image classification**

Large Multi-modal Models (Frozen)

- Experiment Results

n-shot Acc.	n=0	n=1	n=4	τ
Frozen	29.5	35.7	38.2	✗
Frozen scratch	0.0	0.0	0.0	✗
Frozen finetuned	24.0	28.2	29.2	✗
Frozen train-blind	26.2	33.5	33.3	✗
Frozen VQA	48.4	–	–	✓
Frozen VQA-blind	39.1	–	–	✓
Oscar [23]	73.8	–	–	✓

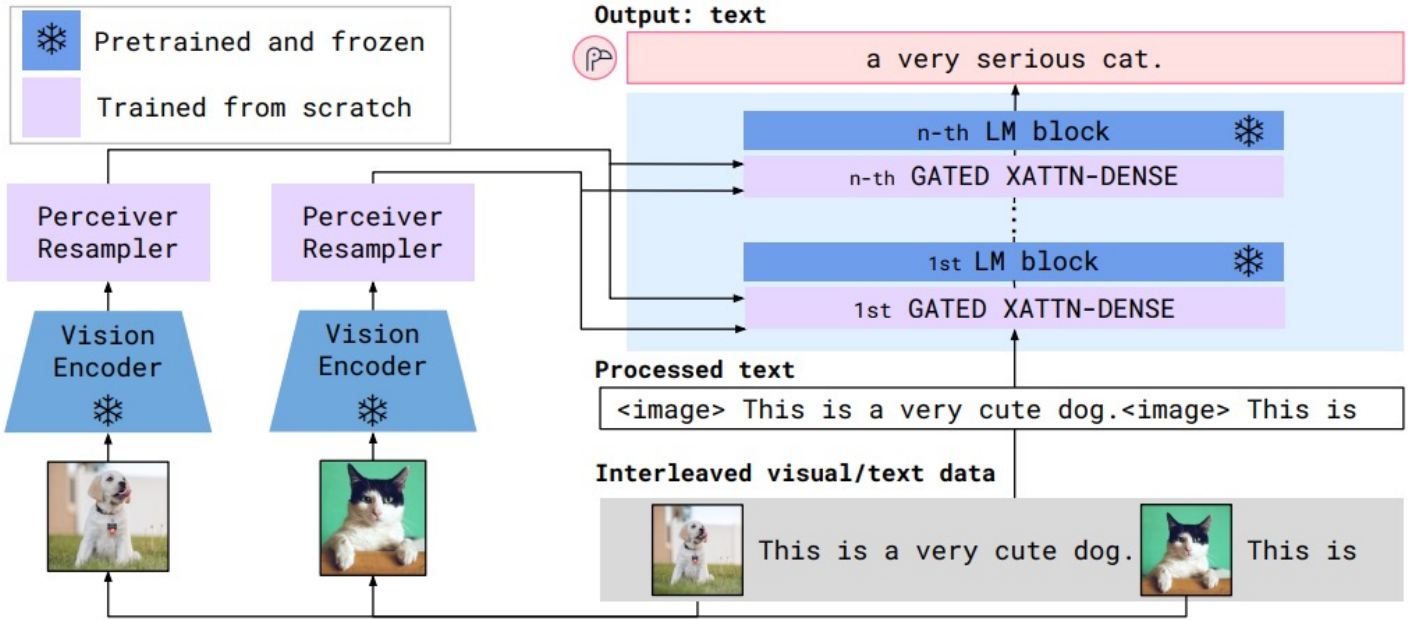
Table 1: Transfer from Conceptual Captions to VQAv2. The τ column indicates whether a model uses training data from the VQAv2 training set. The row denoted *Frozen* train-blind is the blind baseline described in subsection 4.1. *Frozen* VQA is a

n-shot Acc.	n=0	n=1	n=4	τ
Frozen	5.9	9.7	12.6	✗
Frozen 400mLM	4.0	5.9	6.6	✗
Frozen finetuned	4.2	4.1	4.6	✗
Frozen train-blind	3.3	7.2	0.0	✗
Frozen VQA	19.6	–	–	✗
Frozen VQA-blind	12.5	–	–	✗
MAVEx [42]	39.4	–	–	✓

Table 2: Transfer from Conceptual Captions to OKVQA. The τ column indicates if a model uses training data from the OKVQA training set. *Frozen* does not train on VQAv2 except in the baseline row, and it never trains on OKVQA.

Large Multi-modal Models (Flamingo)

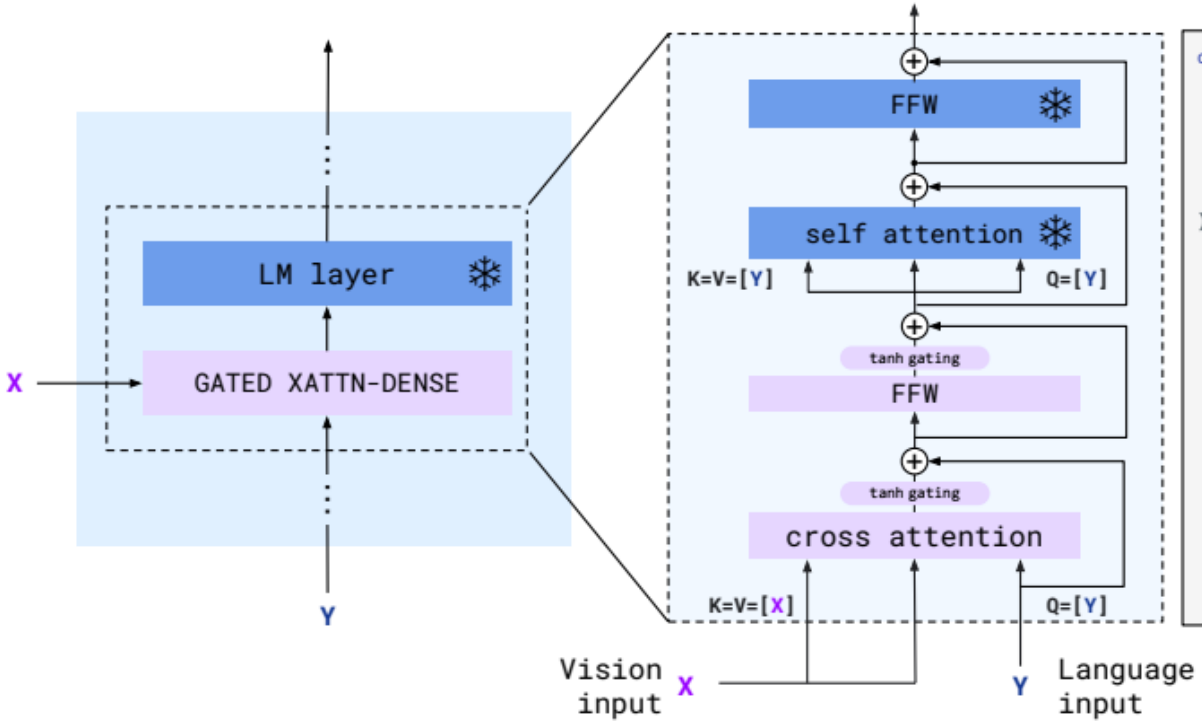
- Architecture Overview



Alayrac, Jean-Baptiste, et al. "Flamingo: a visual language model for few-shot learning." Advances in Neural Information Processing Systems 35 (2022): 23716-23736.

Large Multi-modal Models (Flamingo)

- GATED XATTN-DENSE layers




```
def gated_xattn_dense(
    y,  # input language features
    x,  # input visual features
    alpha_xattn, # xattn gating parameter – init at 0.
    alpha_dense, # ffw gating parameter – init at 0.
):
  """Applies a GATED XATTN-DENSE layer."""

  # 1. Gated Cross Attention
  y = y + tanh(alpha_xattn) * attention(q=y, kv=x)
  # 2. Gated Feed Forward (dense) Layer
  y = y + tanh(alpha_dense) * ffw(y)

  # Regular self-attention + FFW on language
  y = y + frozen_attention(q=y, kv=y)
  y = y + frozen_ffw(y)
  return y  # output visually informed language features
```

Large Multi-modal Models (Flamingo)

- Experiment Results

Method	FT	Shot	OKVQA (I)	VQAv2 (I)	COCO (I)	MSVDQA (V)	VATEX (V)	VizWiz (I)	Flick30K (I)	MSRVTTQA (V)	iVQA (V)	YouCook2 (V)	STAR (V)	VisDial (I)	TextVQA (I)	NextQA (I)	HatefulMemes (I)	RareAct (V)
Zero/Few shot SOTA	✗	(X)	[34] 43.3 (16)	[114] 38.2 (4)	[124] 32.2 (0)	[58] 35.2 (0)	-	-	-	[58] 19.2 (0)	[135] 12.2 (0)	-	[143] 39.4 (0)	[79] 11.6 (0)	-	-	[85] 66.1 (0)	[85] 40.7 (0)
Flamingo-3B	✗	0	41.2	49.2	73.0	27.5	40.1	28.9	60.6	11.0	32.7	55.8	39.6	46.1	30.1	21.3	53.7	58.4
	✗	4	43.3	53.2	85.0	33.0	50.0	34.0	72.0	14.9	35.7	64.6	41.3	47.3	32.7	22.4	53.6	-
	✗	32	45.9	57.1	99.0	42.6	59.2	45.5	71.2	25.6	37.7	76.7	41.6	47.3	30.6	26.1	56.3	-
Flamingo-9B	✗	0	44.7	51.8	79.4	30.2	39.5	28.8	61.5	13.7	35.2	55.0	41.8	48.0	31.8	23.0	57.0	57.9
	✗	4	49.3	56.3	93.1	36.2	51.7	34.9	72.6	18.2	37.7	70.8	**42.8**	50.4	33.6	24.7	62.7	-
	✗	32	51.0	60.4	106.3	47.2	57.4	44.0	72.8	29.4	40.7	77.3	41.2	50.4	32.6	28.4	63.5	-
Flamingo	✗	0	50.6	56.3	84.3	35.6	46.7	31.6	67.2	17.4	40.7	60.1	39.7	52.0	35.0	26.7	46.4	**60.8**
	✗	4	57.4	63.1	103.2	41.7	56.0	39.6	75.1	23.9	44.1	74.5	42.4	**55.6**	36.5	30.8	68.6	-
	✗	32	**57.8**	**67.6**	**113.8**	**52.3**	**65.1**	**49.8**	**75.4**	**31.0**	**45.3**	**86.8**	42.2	**55.6**	**37.9**	**33.5**	**70.0**	-
Pretrained FT SOTA	✔	(X)	54.4 [34] (10K)	80.2 [140] (444K)	143.3 [124] (500K)	47.9 [28] (27K)	76.3 [153] (500K)	57.2 [65] (20K)	67.4 [150] (30K)	46.8 [51] (130K)	35.4 [135] (6K)	138.7 [132] (10K)	36.7 [128] (46K)	75.2 [79] (123K)	54.7 [137] (20K)	25.2 [129] (38K)	79.1 [62] (9K)	-

Summary: Large (Multi-modal) Models

Large Multi-modal Models: Multiple Modalities Inputs.

- **Modality-specific Encoder**: Encoding each modality independently
- **Projector:** Projecting representations in specific spaces to an aligned space
- **Decoder (Commonly unified)** : Decoding the representations to the target modality

What is the true scaling law to AGI? (My personal believe.)

- Scale up data or model? **Maybe not enough!**

The true scaling law is: continuously learn to interact, and interact to learn!

Please look forward to the upcoming course on **Embodied AI** by Prof. He Wang!

And welcome to join us: zhangzz@galbot.com.