

10-423/623: Generative AI

Lec 5 – Computer Vision

CNNs, Decoder-only, ViT

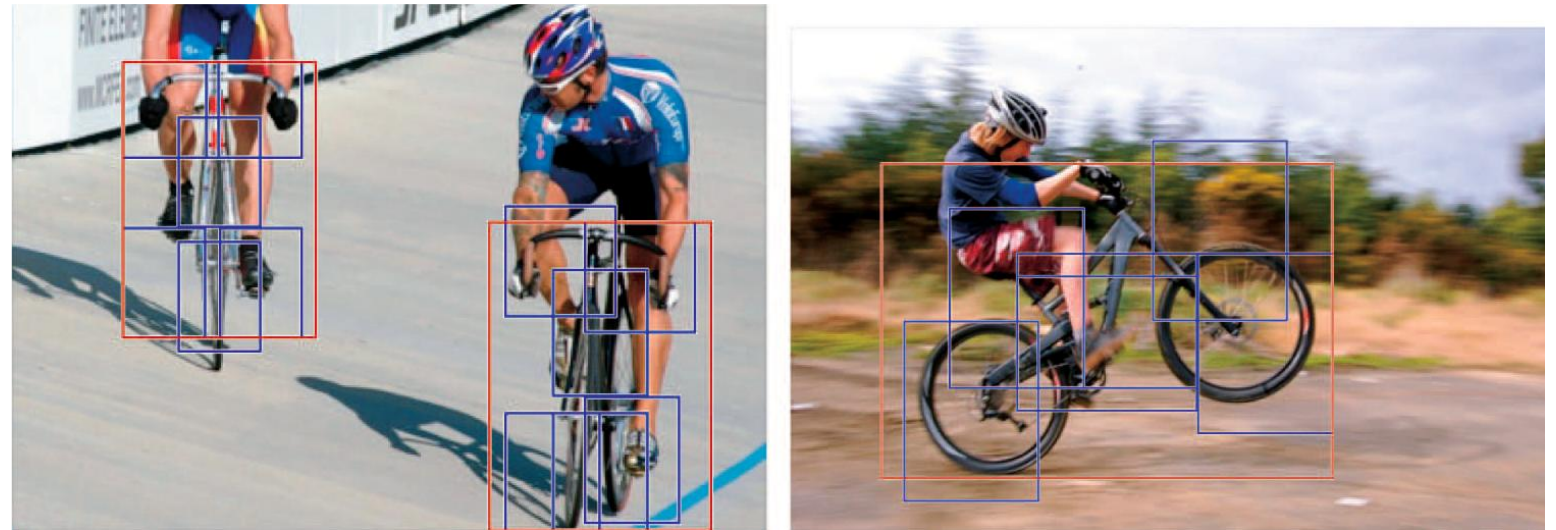
Pat Virtue & Matt Gormley

1/29/25

Slide credits: CMU MLD, Henry Chai

Convolutional Neural Networks

- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Insight: for spatially-structured inputs, many useful features are shift or location-invariant



Convolutional Neural Networks

- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Insight: for spatially-structured inputs, many useful features are shift or location-invariant
- Strategy:
 - a) learn a *filter* for *micro*-feature detection in a small window and apply it over the entire image
 - b) downsample (shrink) resulting feature image(s)
 - c) repeat at future layers to learn increasingly *macro* features

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter (or kernel) is just a small matrix that is *convolved* with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1	0
1	-4	1
0	1	0

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter (or kernel) is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1	0
1	-4	1
0	1	0

=

0			

$$(0 * 0) + (0 * 1) + (0 * 0) + (0 * 1) + (1 * -4) + (2 * 1) + (0 * 0) + (2 * 1) + (4 * 0) = 0$$

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter (or kernel) is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1	0
1	-4	1
0	1	0

=

0	-1		

$$(0 * 0) + (0 * 1) + (0 * 0) + (1 * 1) + (2 * -4) + (2 * 1) + (2 * 0) + (4 * 1) + (4 * 0) = -1$$

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter (or kernel) is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

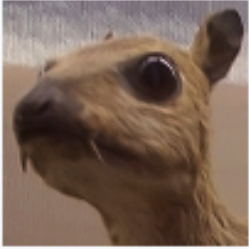



 *

0	1	0
1	-4	1
0	1	0




 =

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

Convolutional Filters

Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

More Filters

Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Convolutional Filters

- Convolutional layers vs Fully-connected layers
 1. Nodes in the input layer are only connected to some nodes in the next layer but not all nodes.
 2. Shared weight parameters across locations

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

Convolutional Filters

- Convolutional layers vs Fully-connected layers
 1. Nodes in the input layer are only connected to some nodes in the next layer but not all nodes.
 2. Shared weight parameters across locations

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

- Many fewer weights than a fully connected layer!
- **Convolution weights are learned using gradient descent/backpropagation, not prespecified**

Convolutional Filters: Padding

- Want to keep the same image size?
- Add *zeros* around the image to allow for the filter to be applied “everywhere” e.g. a *padding* of 1 with a 3x3 filter preserves image size and allows every pixel to be the center

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	2	2	1	0	0
0	0	2	4	4	2	0	0
0	0	1	3	3	1	0	0
0	0	1	2	3	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

*

0	1	0
1	-4	1
0	1	0

=

0	1	2	2	1	0
1	0	-1	-1	0	1
2	-2	-5	-5	-2	2
1	2	-2	-1	3	1
1	-1	0	-5	0	1
0	2	-1	0	2	0

Downsampling: Stride

- Only apply the convolution to some subset of the image
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1
1	-2

=

-2		

Downsampling: Stride

- Only apply the convolution to some subset of the image
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1
1	-2

=

-2	-2	

Downsampling: Stride

- Only apply the convolution to some subset of the image
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1
1	-2

=

-2	-2	1

Downsampling: Stride

- Only apply the convolution to some subset of the image
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$

0	1
1	-2

 $=$

-2	-2	1
0		

Downsampling: Stride

- Only apply the convolution to some subset of the image
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$

0	1
1	-2

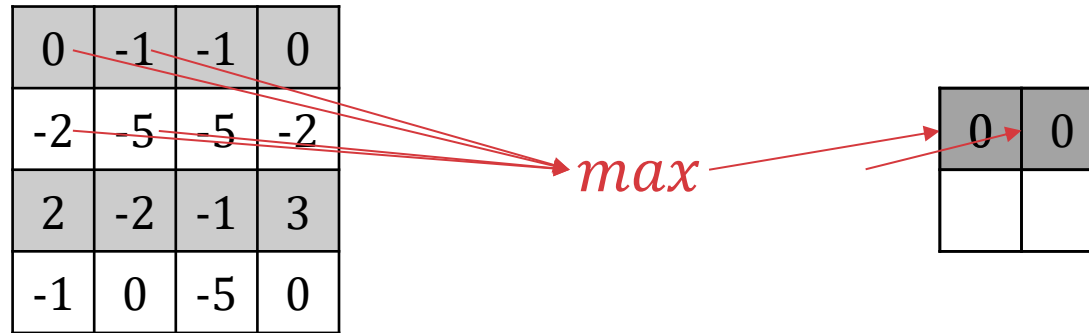
 $=$

-2	-2	1
0	1	1
1	2	0

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
- Many relevant macro-features will tend to span large portions of the image, so taking strides with the convolution tends not to miss out on too much

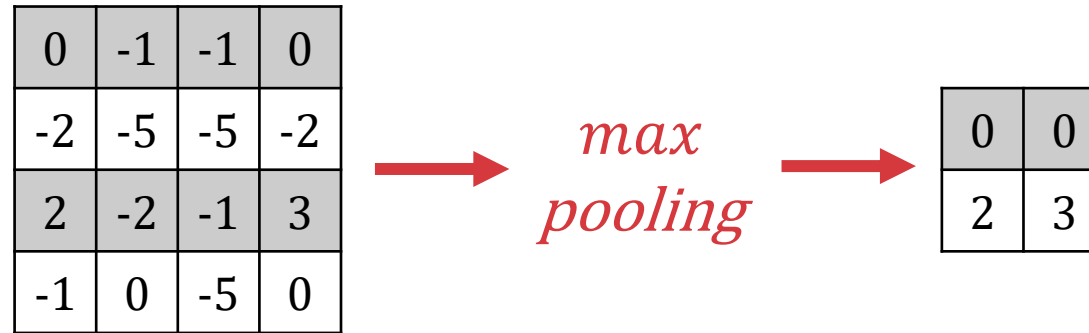
Downsampling: Pooling

- Combine multiple adjacent nodes into a single node

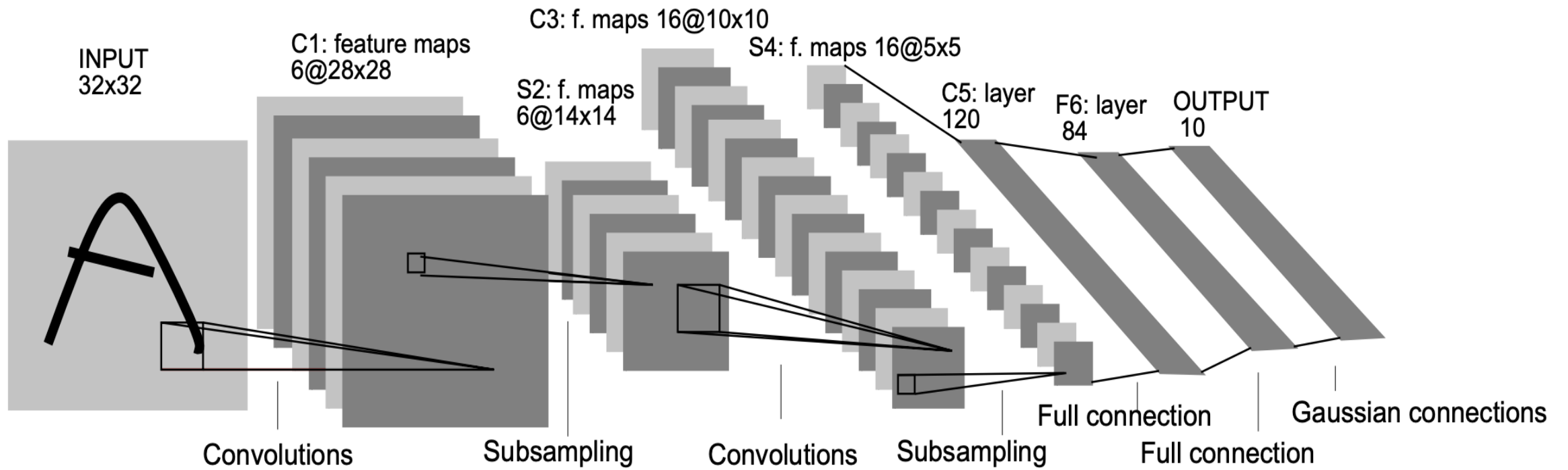


Downsampling: Pooling

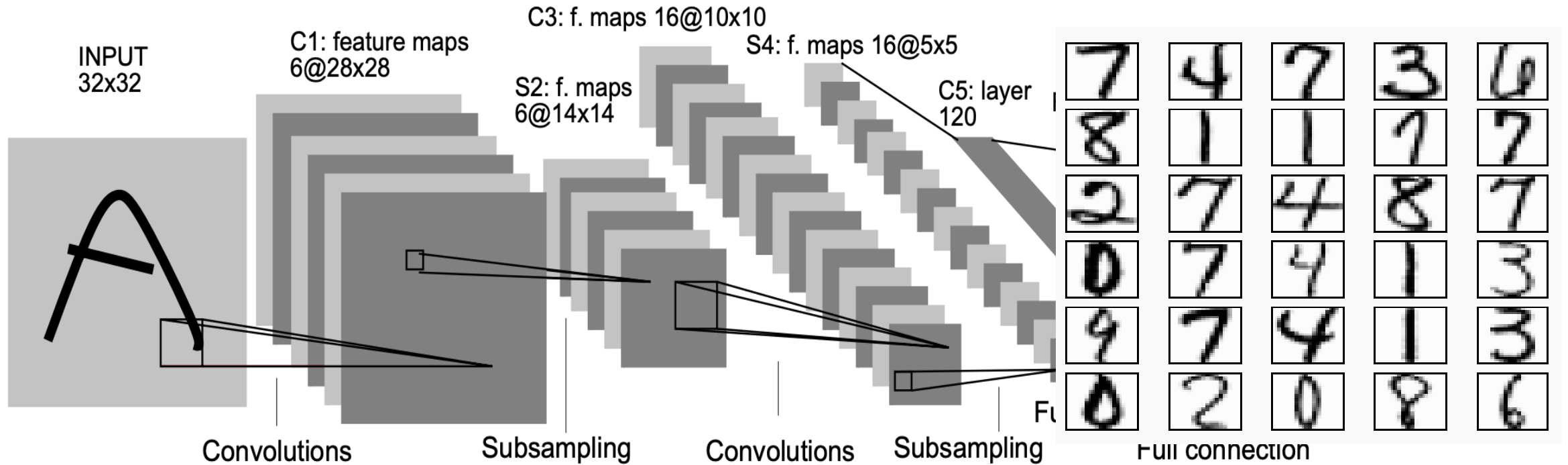
- Combine multiple adjacent nodes into a single node



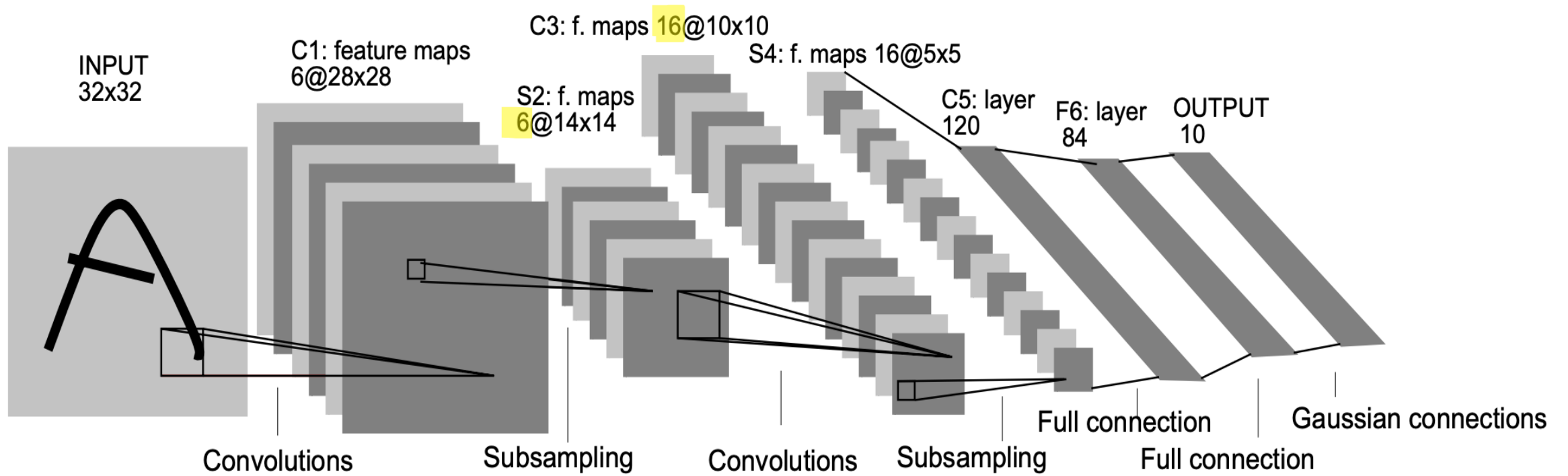
- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
 - Protects the network from (slightly) noisy inputs



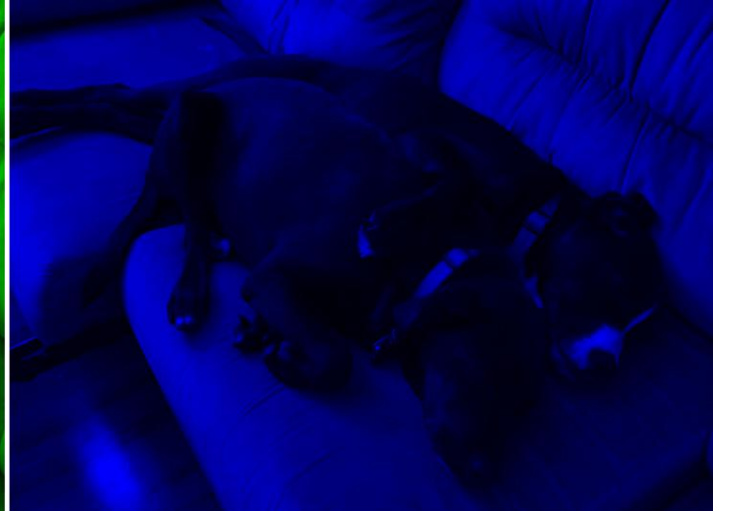
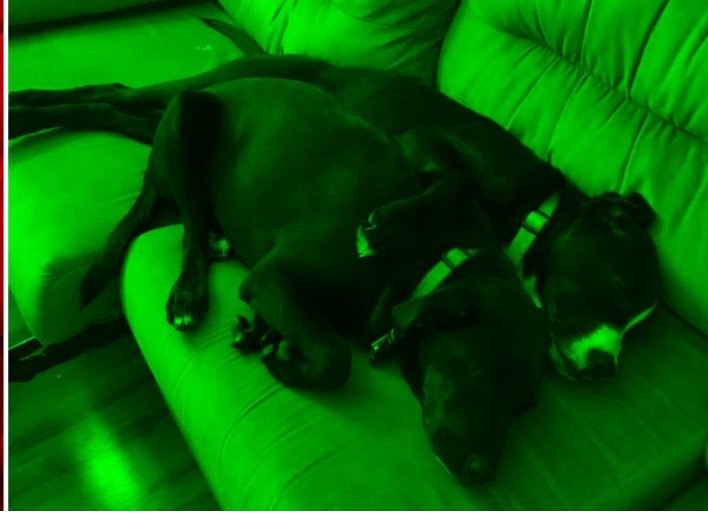
LeNet (LeCun et al., 1998)



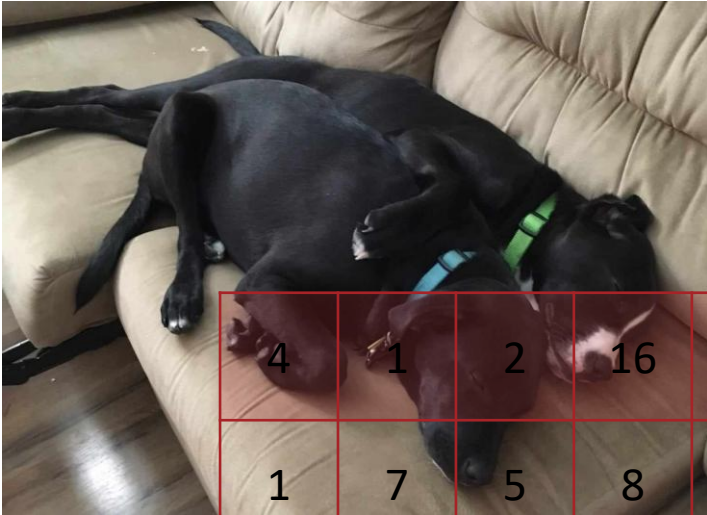
- One of the earliest, most famous deep learning models – achieved remarkable performance at handwritten digit recognition (< 1% test error rate on MNIST)
- Used sigmoid (or logistic) activation functions between layers and mean-pooling, both of which are pretty uncommon in modern architectures



Wait how did we go from 6 to 16?



Channels



4	1	2	16	3	6
1	7	5	8	19	27
5	2	5	12	17	8
0	4	9	9	6	11

5	2	6	14	15	8
26	3	6	8	4	9
0	15	24	6	1	8
7	4	9	5	24	17

4	6	8	9	5	3
16	5	2	8	2	1
5	2	14	11	7	8
15	2	5	0	9	8

- An image can be represented as the sum of red, green and blue pixel intensities
- Each color corresponds to a *channel*



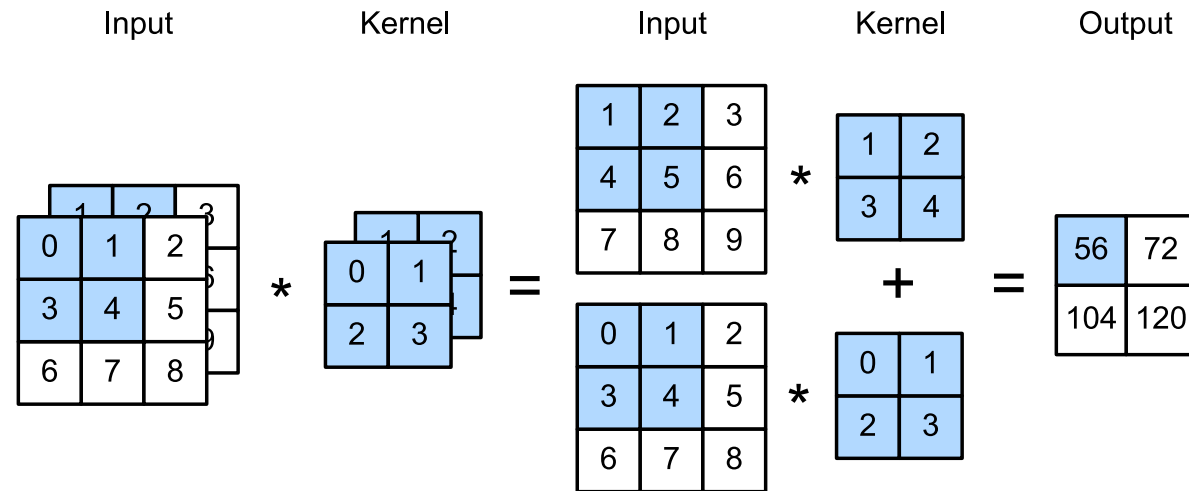
Example: $3 \times 4 \times 6$ tensor

4	1	2	16	3	6							
1						5	2	6	14	15	8	
5						26						
0						0						
						7						
							4	6	8	9	5	3
							16	5	2	8	2	1
							5	2	14	11	7	8
							15	2	5	0	9	8

- An image can be represented as the sum of red, green and blue pixel intensities
- Each color corresponds to a *channel*

Convolutions on Multiple Input Channels

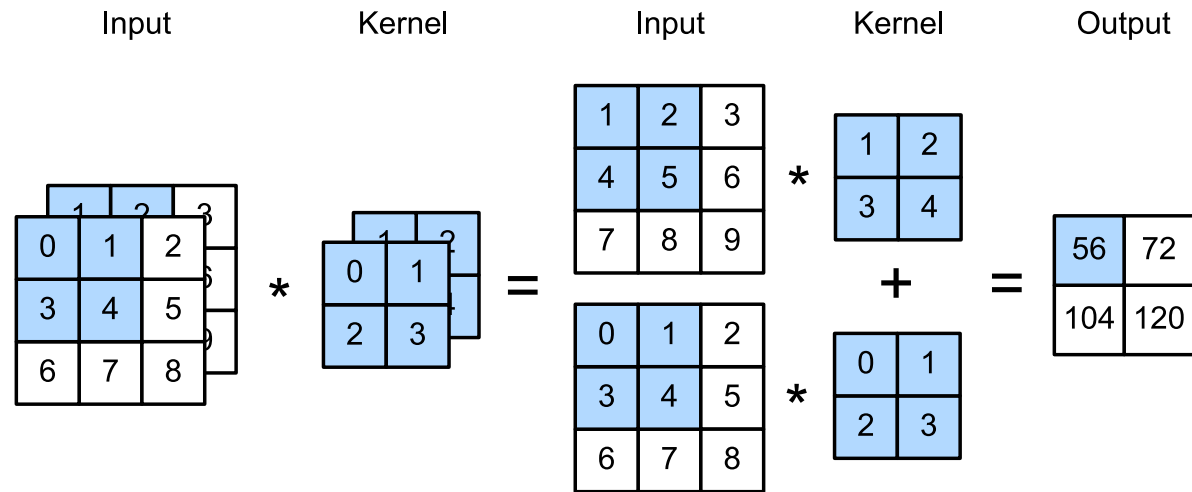
- Given multiple input channels, we can specify a filter for each one and sum the results to get a 2-D output tensor



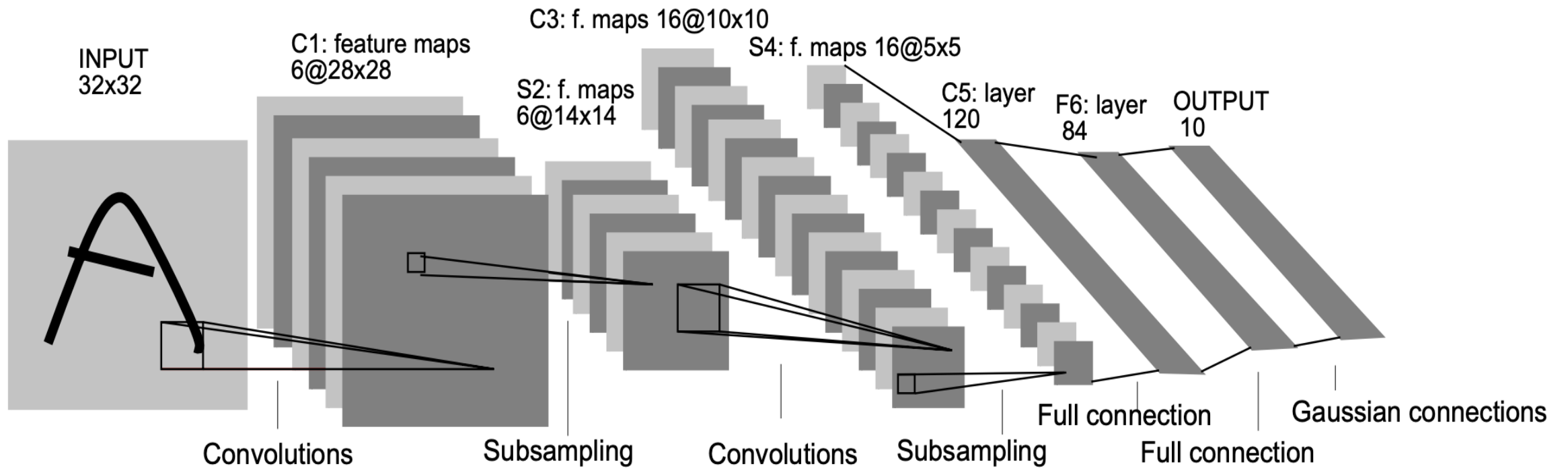
- For c channels and $h \times w$ filters, we have $chw + c$ learnable parameters (each filter has a bias term)

Convolutions on Multiple Input Channels

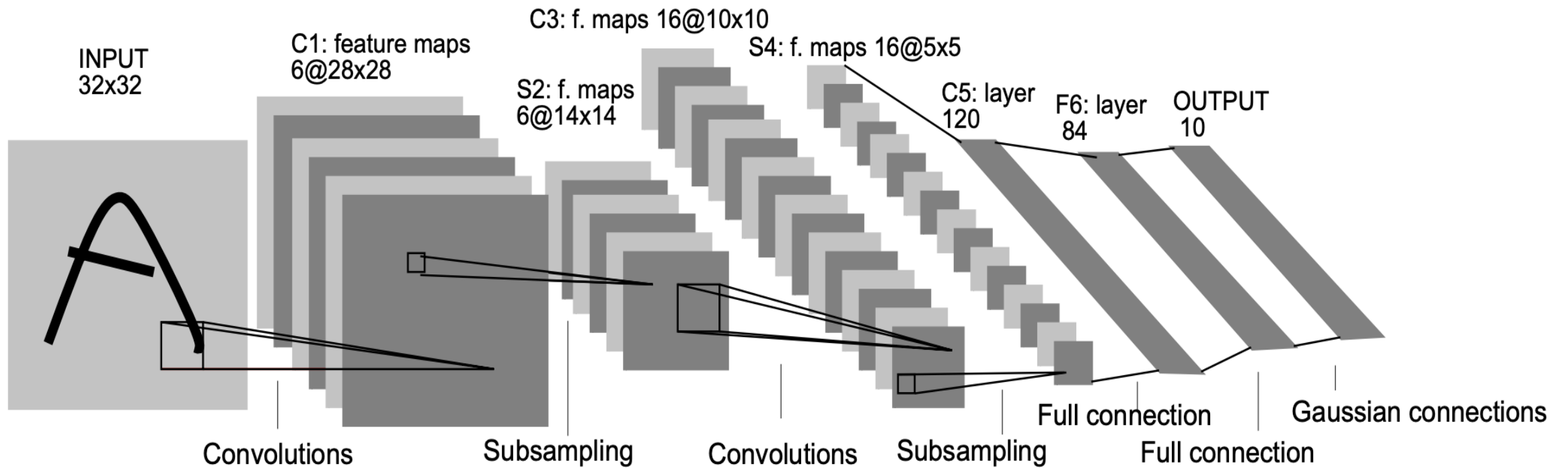
- Given multiple input channels, we can specify a filter for each one and sum the results to get a 2-D output tensor



- Questions:
 - Why might we want a different filter for each input?
 - Why do we combine them together into a single output channel?



- We can combine these macro-features into a new, interesting, “higher-level” feature



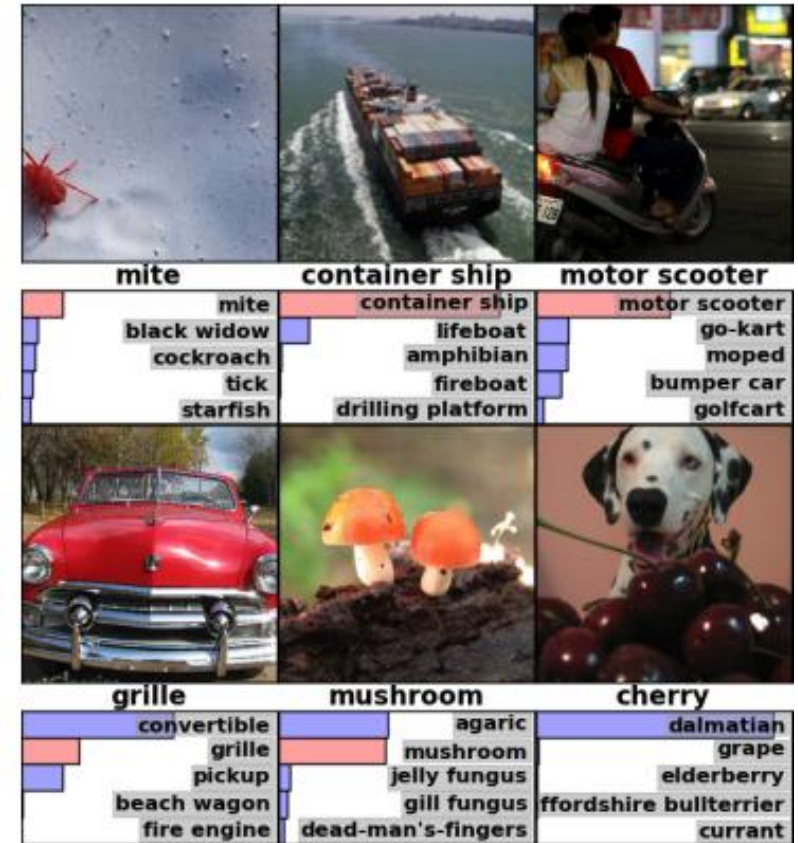
Alright, so what kind of stuff can we actually do with this thing?

Common Tasks in Computer Vision

- Image Classification
- Object Detection
- Semantic Segmentation
- Instance Segmentation
- Image Captioning
- Image Generation

Common Tasks in Computer Vision

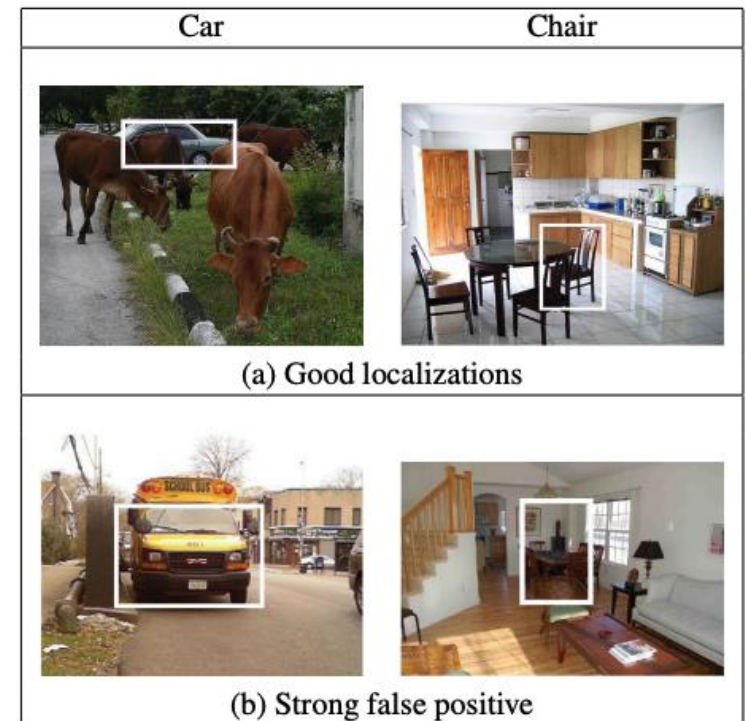
- Image Classification
- Object Localization
- Object Detection
- Semantic Segmentation
- Instance Segmentation
- Image Captioning
- Image Generation



Common Tasks in Computer Vision

- Image Classification
- **Object Detection**
- Semantic Segmentation
- Instance Segmentation
- Image Captioning
- Image Generation

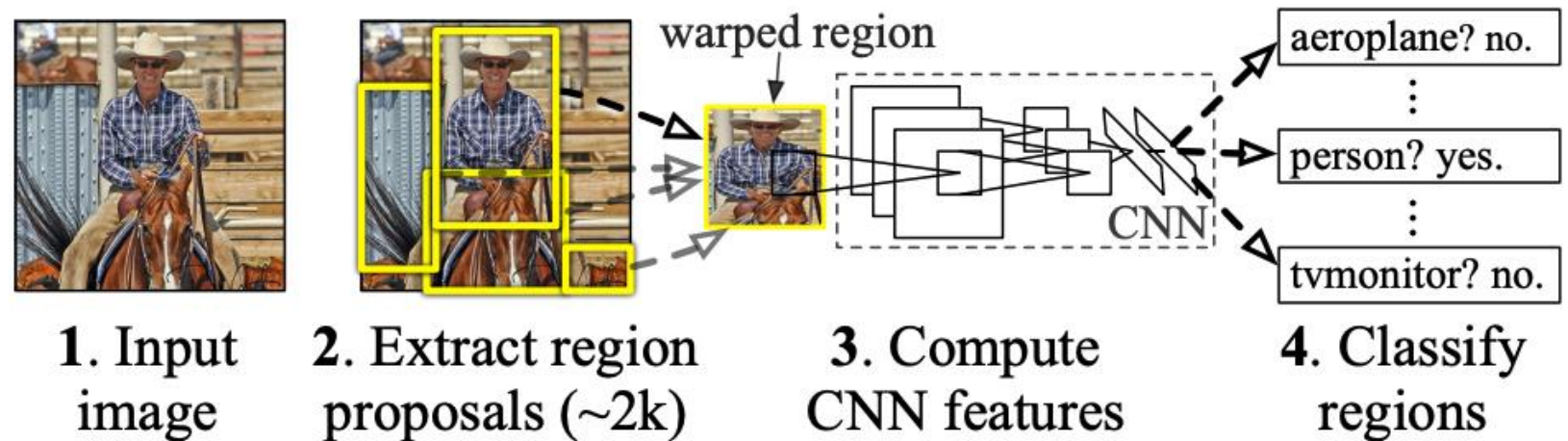
- Given an image, for each object predict a bounding box and a label, $l: (x, y, w, h, l)$



Common Tasks in Computer Vision

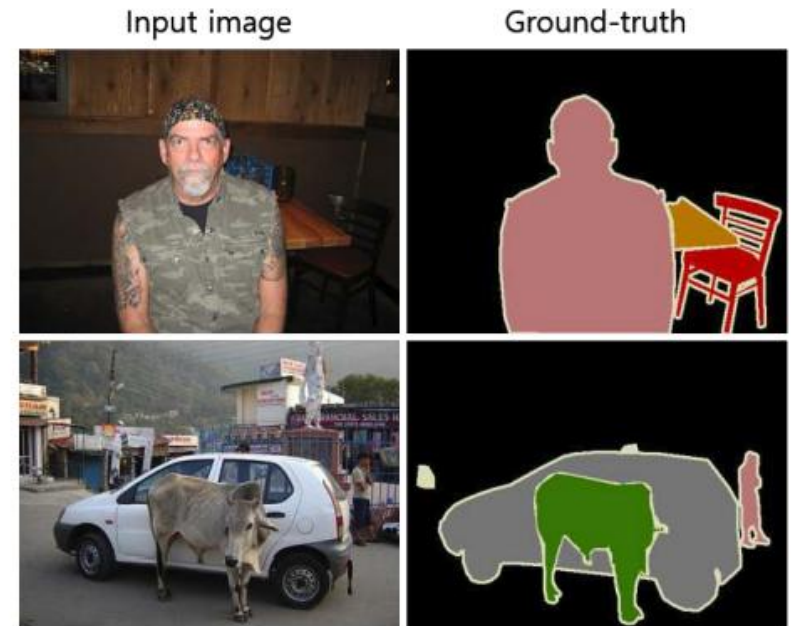
- Image Classification
 - **Object Detection**
- Given an image, for each object predict a bounding box and a label, $l: (x, y, w, h, l)$

R-CNN: *Regions with CNN features*



Common Tasks in Computer Vision

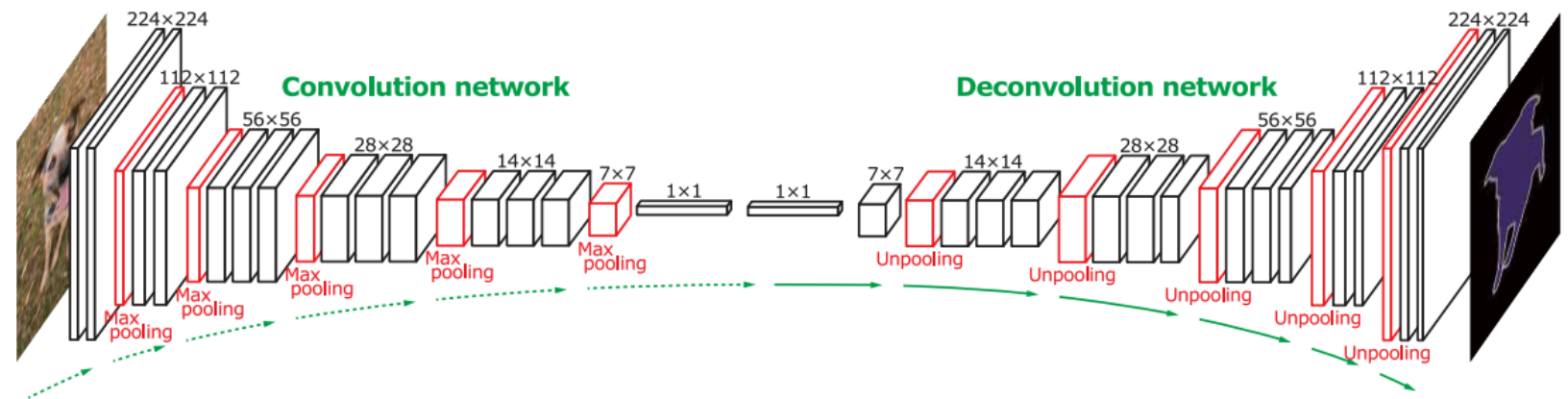
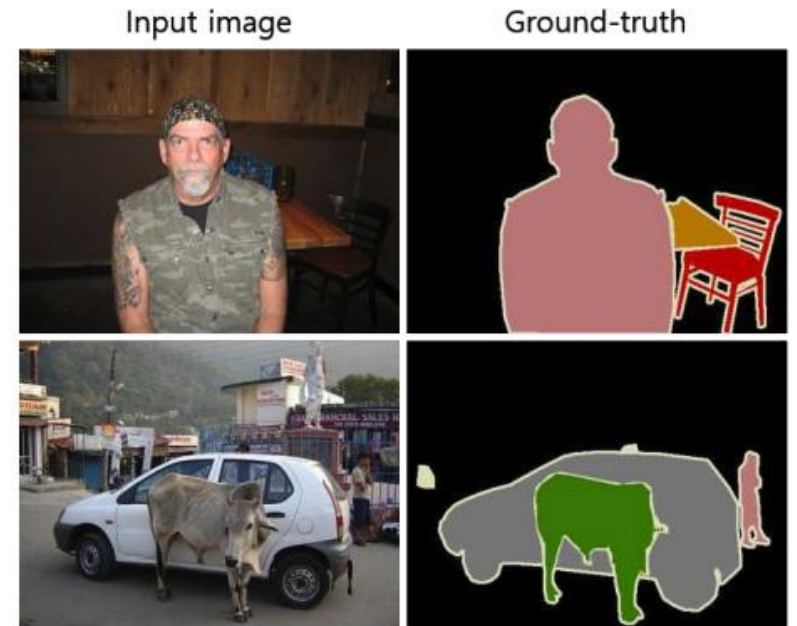
- Image Classification
- Object Detection
- **Semantic Segmentation**
- Instance Segmentation
- Image Captioning
- Image Generation



- Given an image, predict a label for every pixel in the image

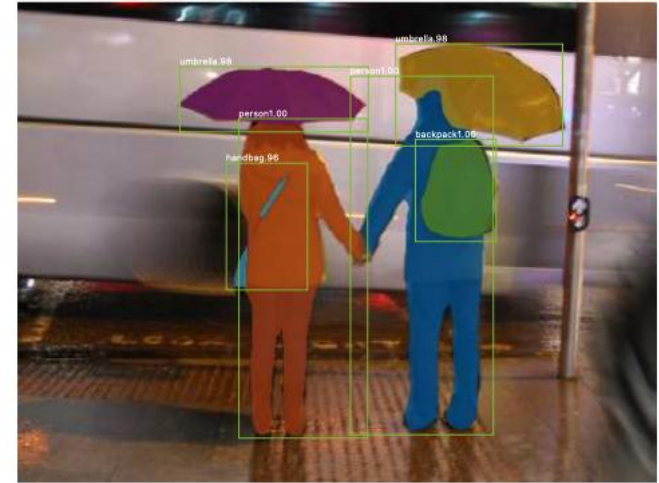
Common Tasks in Computer Vision

- Image Classification
- Object Detection
- **Semantic Segmentation**



Common Tasks in Computer Vision

- Image Classification
- Object Detection
- Semantic Segmentation
- **Instance Segmentation**
- Image Captioning
- Image Generation



- Predict per-pixel labels as in semantic segmentation, but differentiate between different instances of the same label e.g., given two people, one should be labeled **person-1** and one should be labeled **person-2**

Common Tasks in Computer Vision

- Image Classification
- Object Detection
- Semantic Segmentation
- **Instance Segmentation**
- Image Captioning
- Image Generation

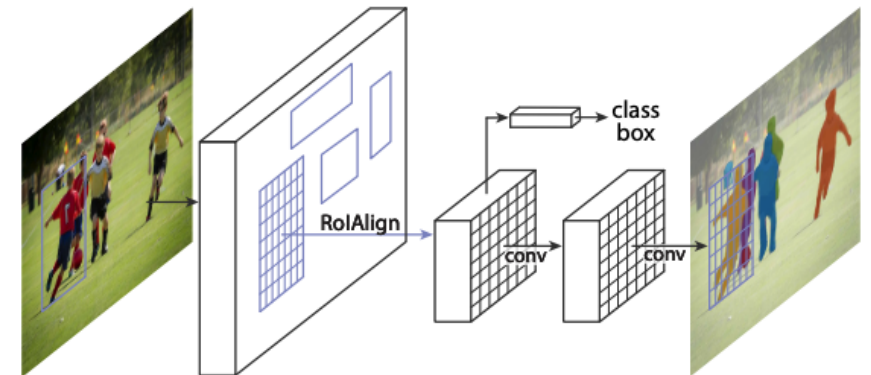
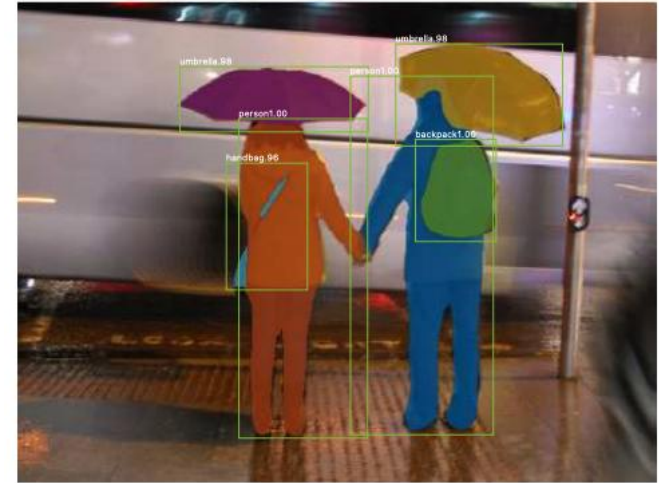


Figure 1. The **Mask R-CNN** framework for instance segmentation.

Common Tasks in Computer Vision

- Image Classification
- Object Detection
- Semantic Segmentation
- Instance Segmentation
- **Image Captioning**
- Image Generation



Ground Truth Caption: A little boy runs away from the approaching waves of the ocean.

Generated Caption: A young boy is running on the beach.

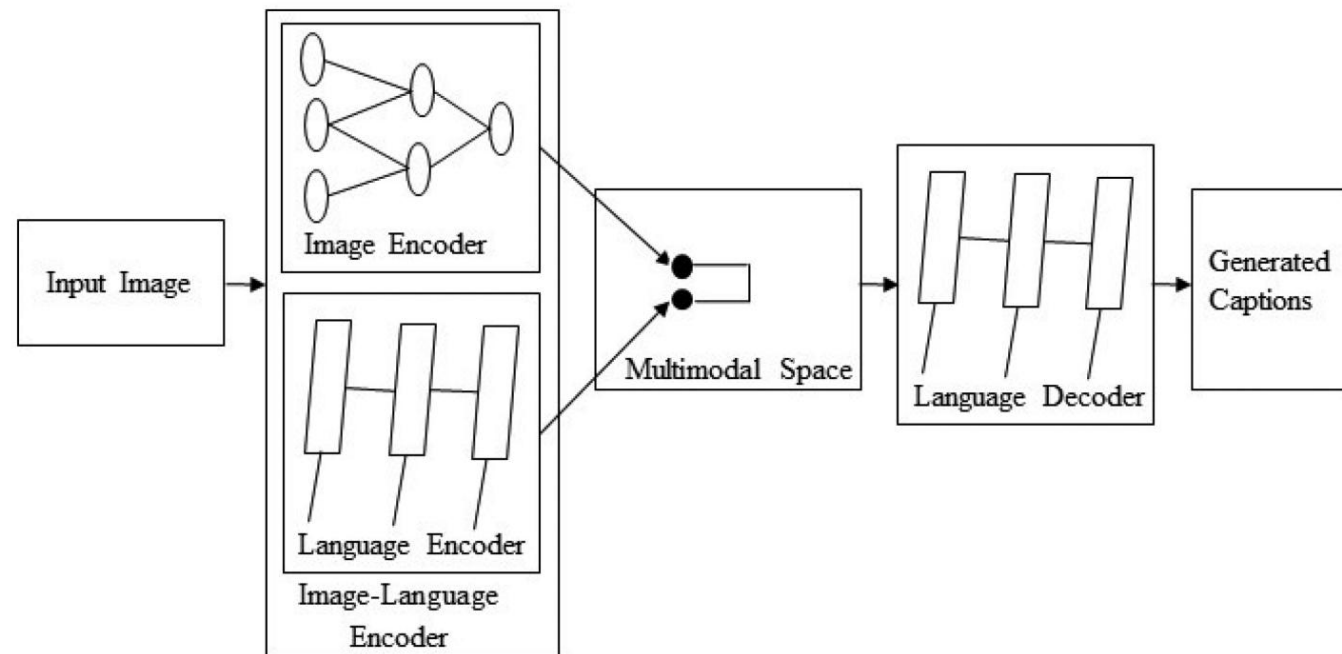


Ground Truth Caption: A brunette girl wearing sunglasses and a yellow shirt.

Generated Caption: A woman in a black shirt and sunglasses smiles.

- Take an image as input, and generate a sentence describing it as output
 - *Dense captioning* generates one description per bounding box

Common Tasks in Computer Vision



- **Image Captioning**
- Image Generation

- Typical architectures will combine a CNN and an RNN-like or transformer language model

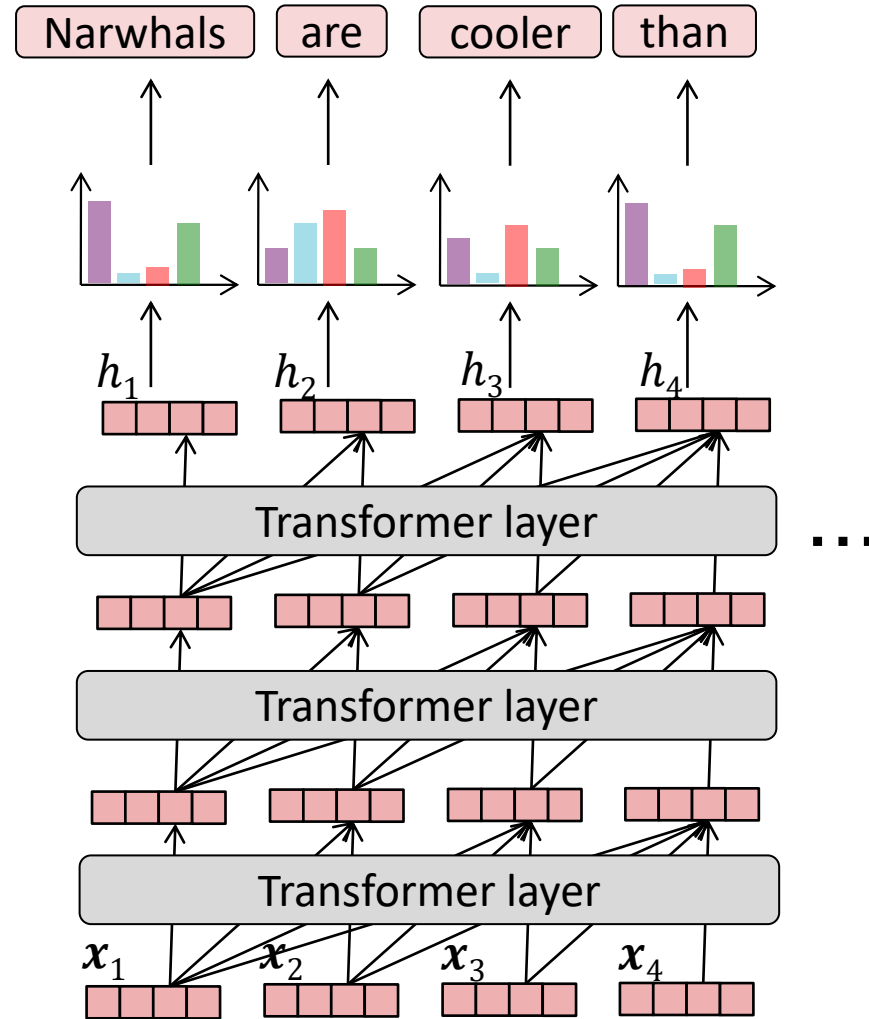
Common Tasks in Computer Vision

- Image Classification
- Object Detection
- Semantic Segmentation
- Instance Segmentation
- Image Captioning
- Image Generation?

Vision
Transformers

Time Out: LM Decoders, Encoders

Recall: Transformer Language Model



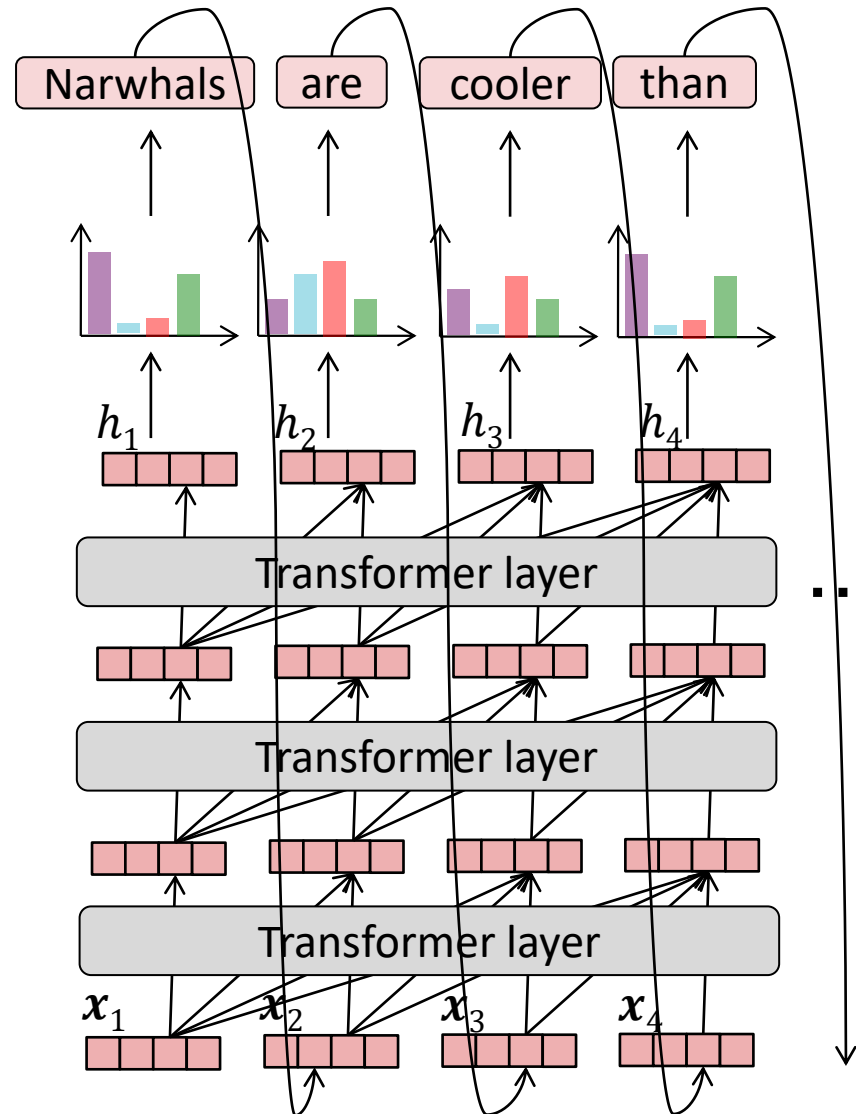
Each layer of a Transformer LM consists of:

1. causal attention
2. feed-forward neural network
3. layer normalization
4. residual connections

...

Each hidden vector looks back at the hidden vectors of the current and previous timesteps in the previous layer.

Recall:
Transformer
Language
Model a.k.a.
Decoder-only
Transformer



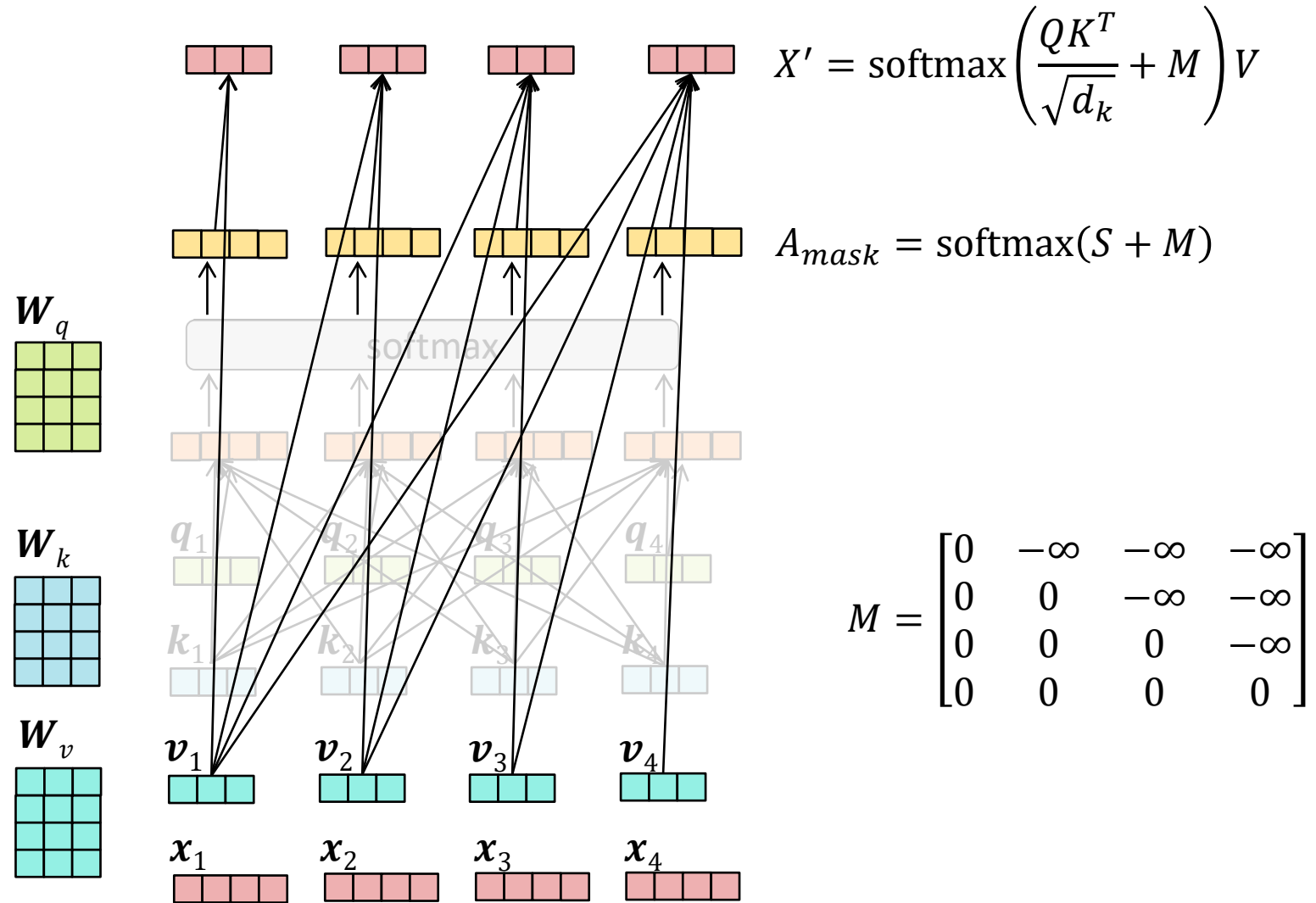
Each layer of a
Transformer LM
consists of:

1. causal attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector
looks back at the
hidden vectors of the
**current and previous
timesteps in the
previous layer.**

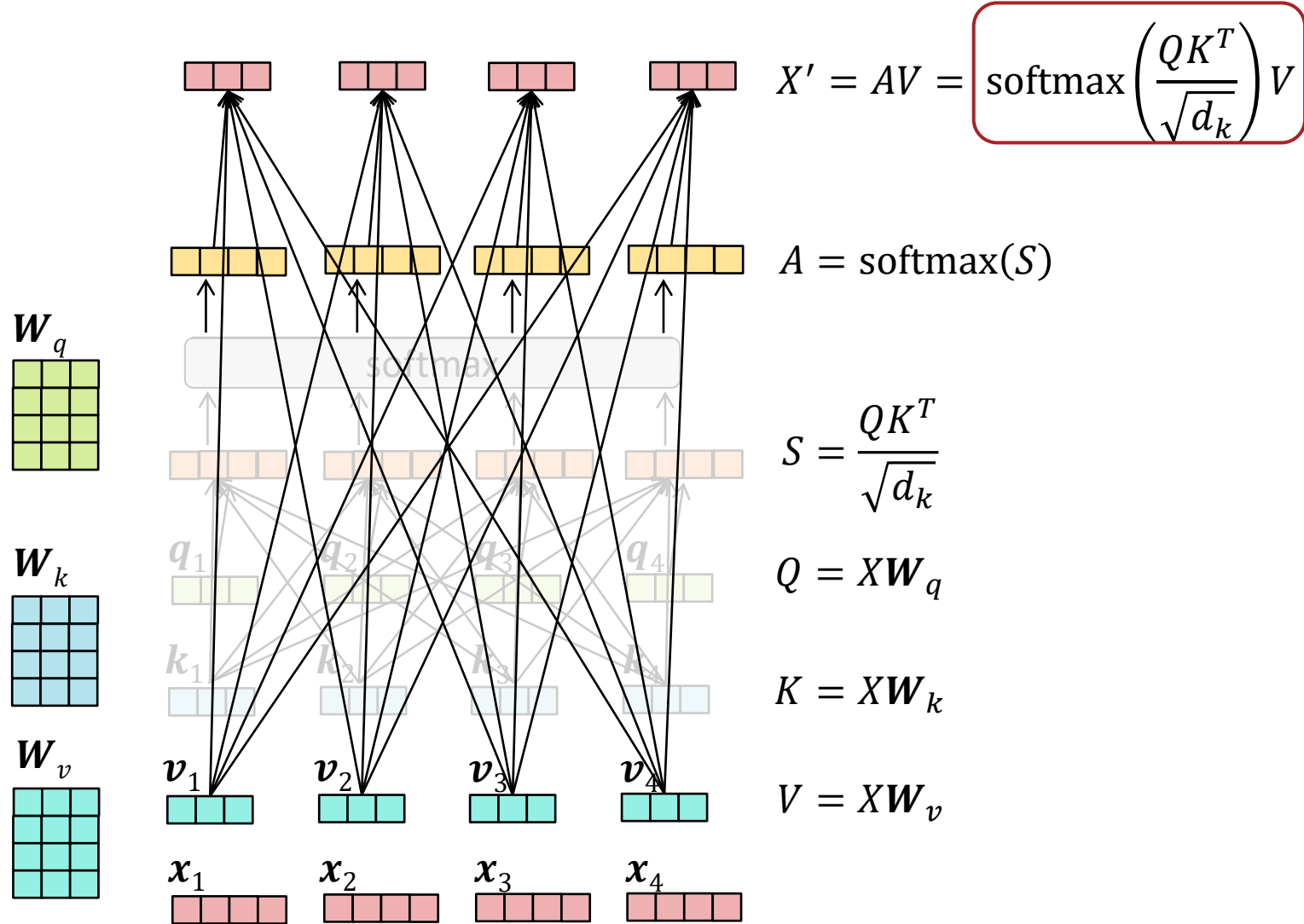
Recall: Causal Attention

Idea: we can effectively delete or “mask” some of these arrows by selectively setting attention weights to 0



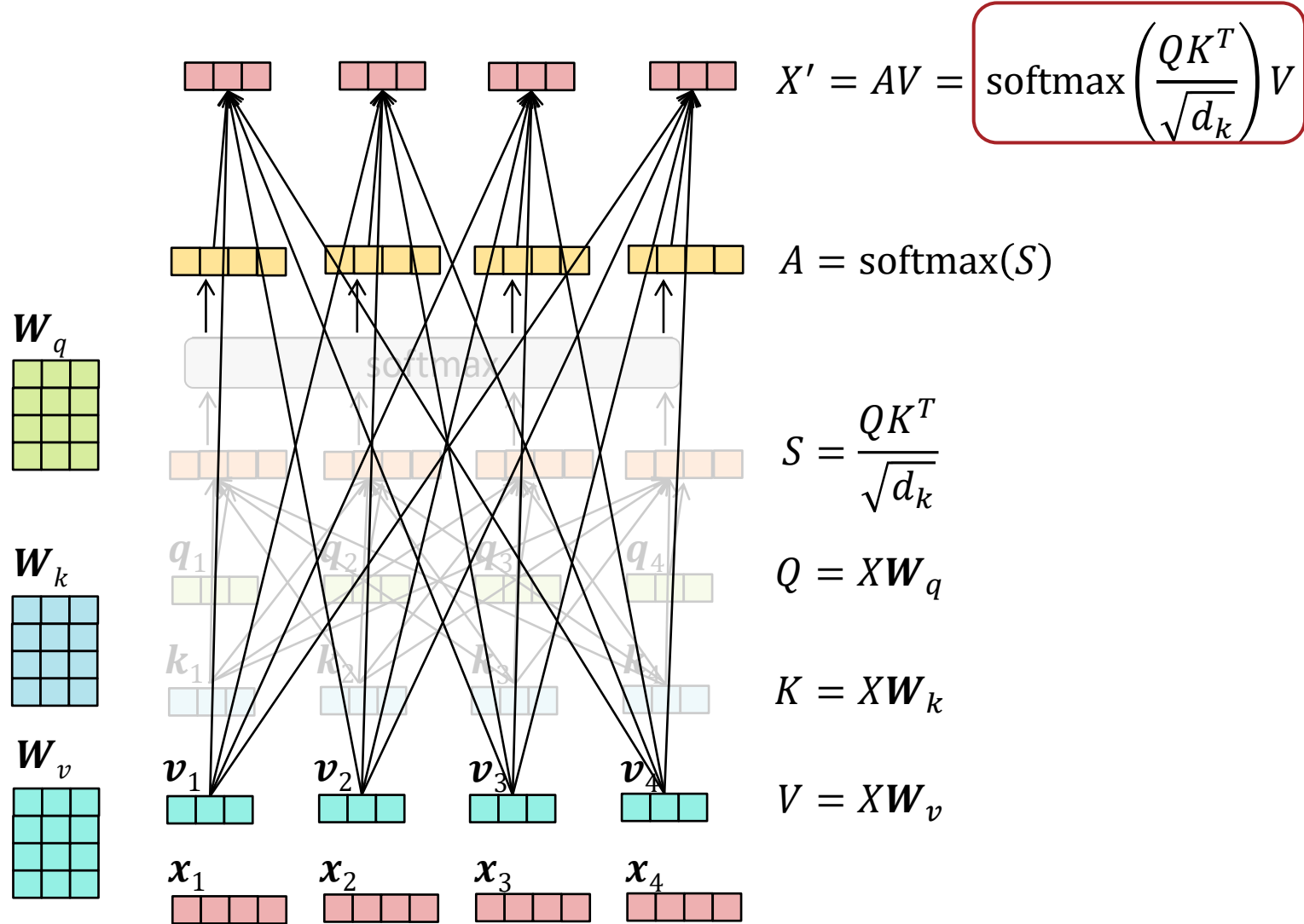
Holy cow,
that's a lot of
new arrows...
do we
always
want/need all
of those?

No...

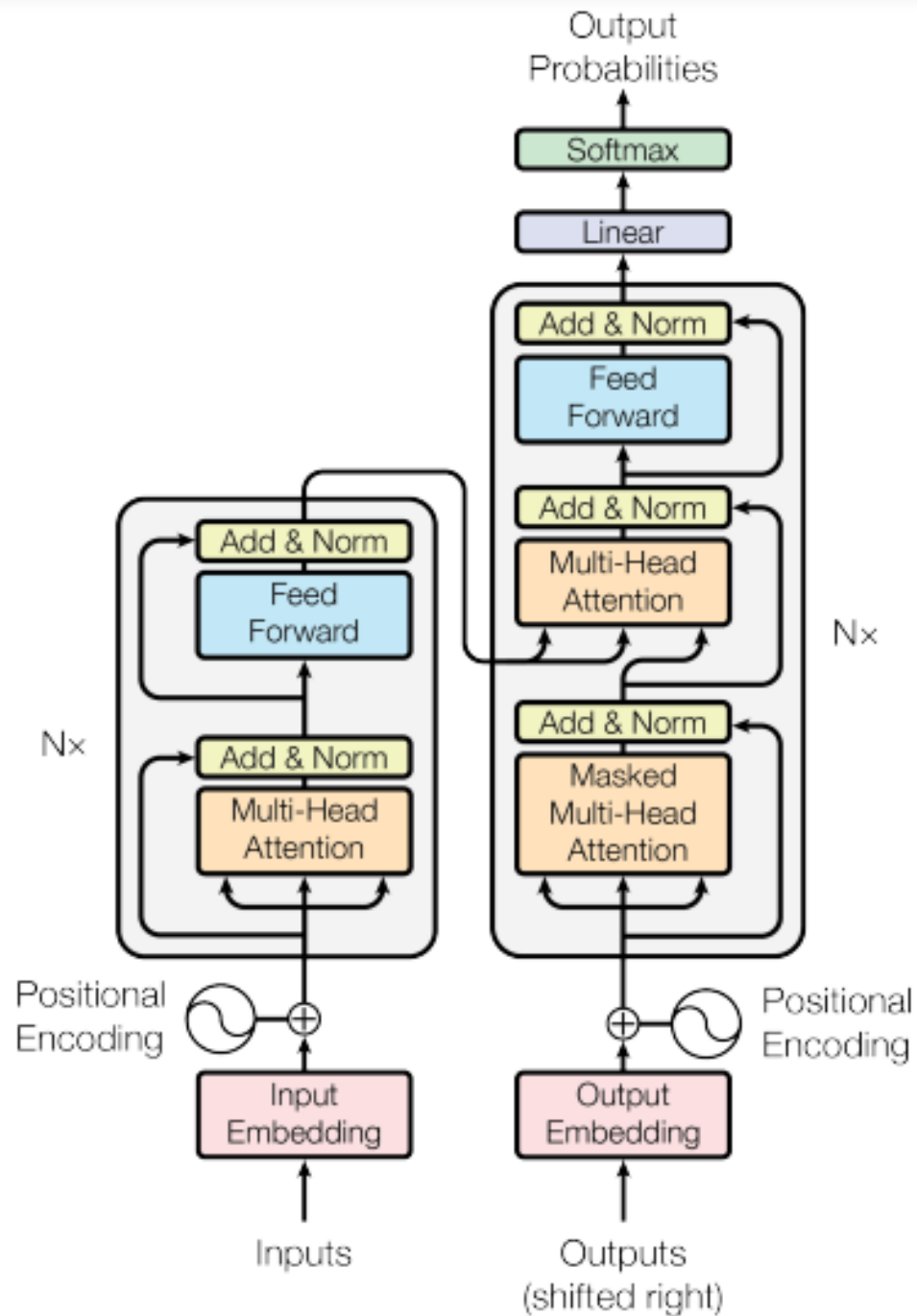


Holy cow,
that's a lot of
new arrows...
do we
sometimes
want/need all
of those?

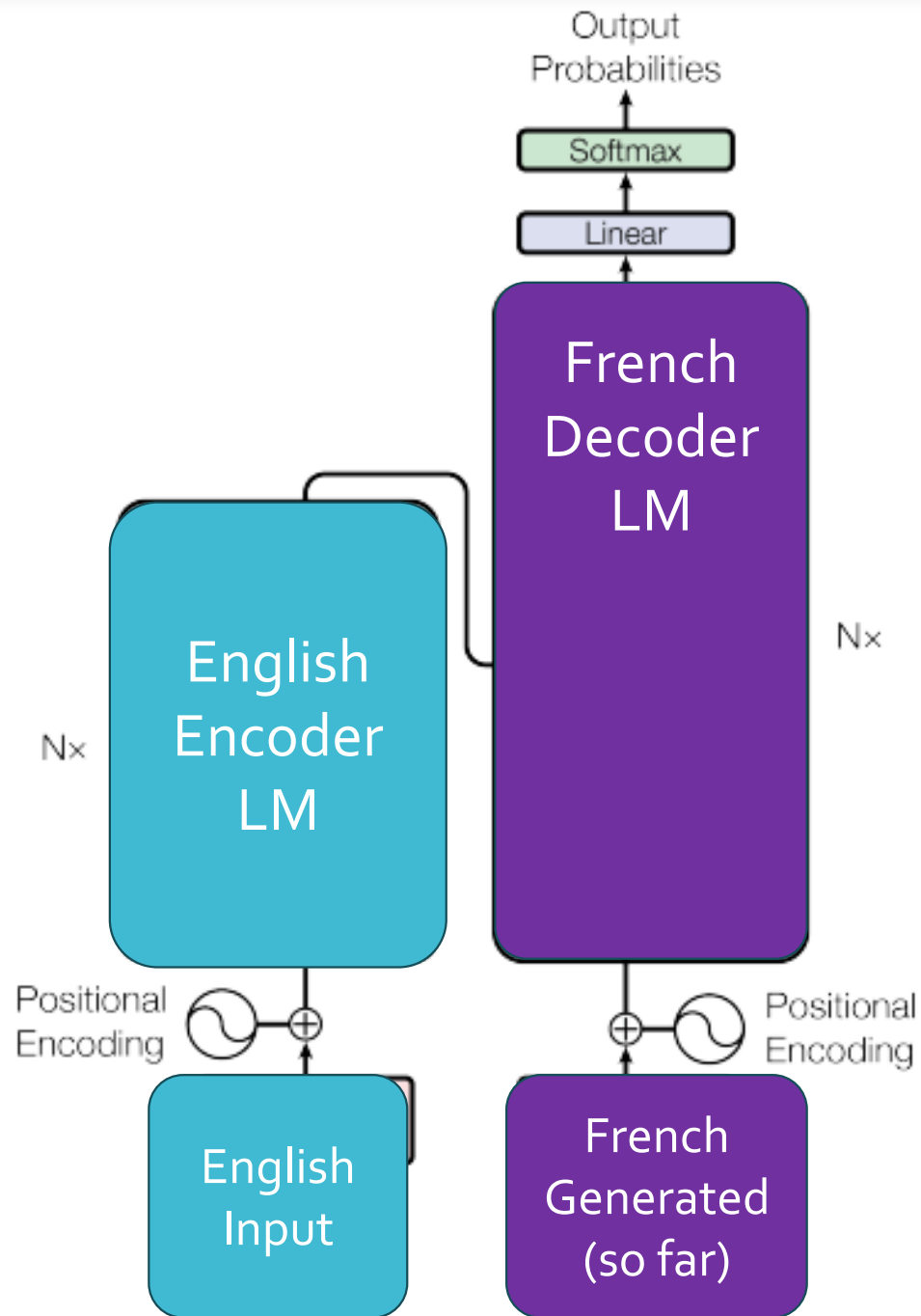
Yes!



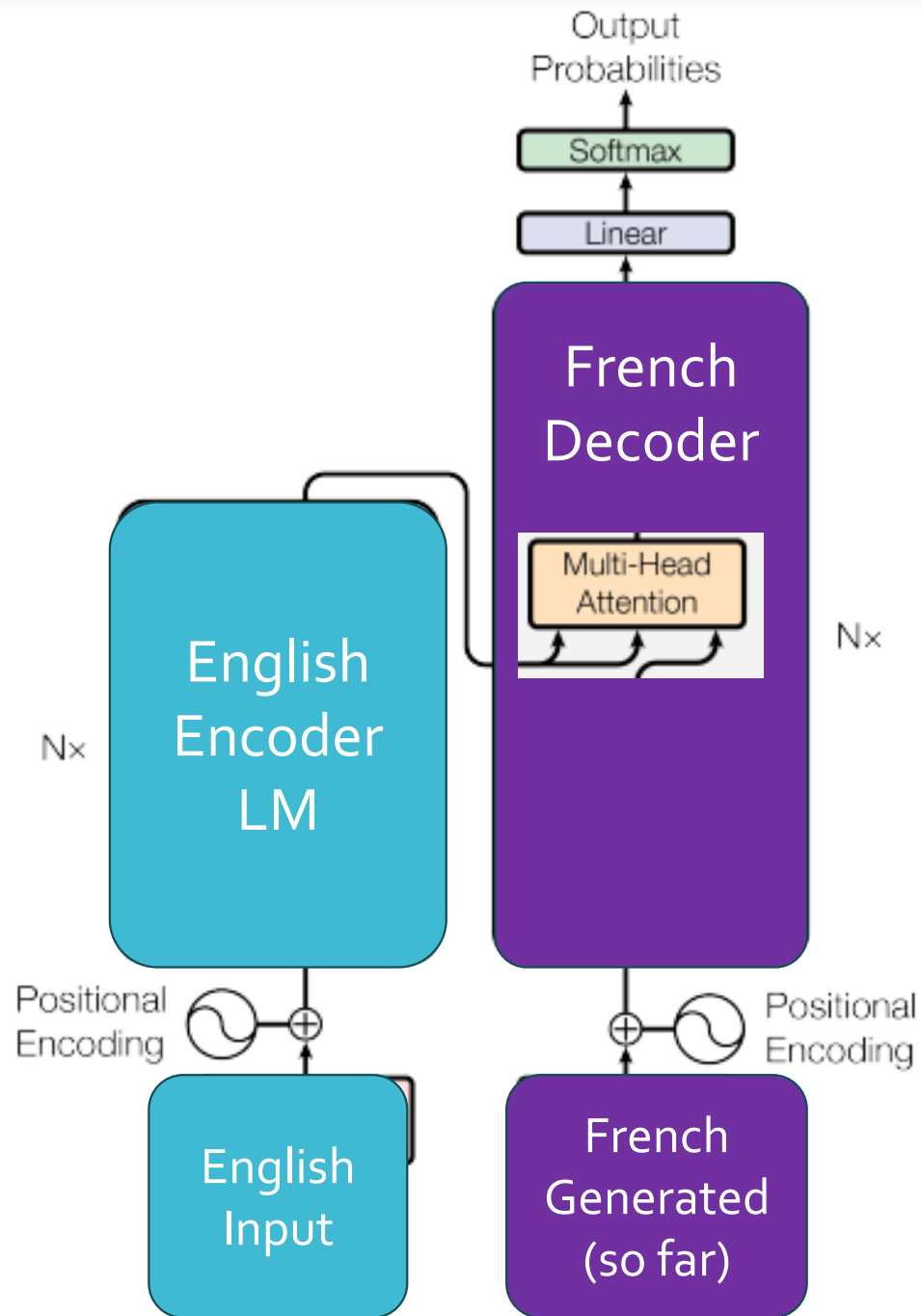
Language Translation



Language Translation

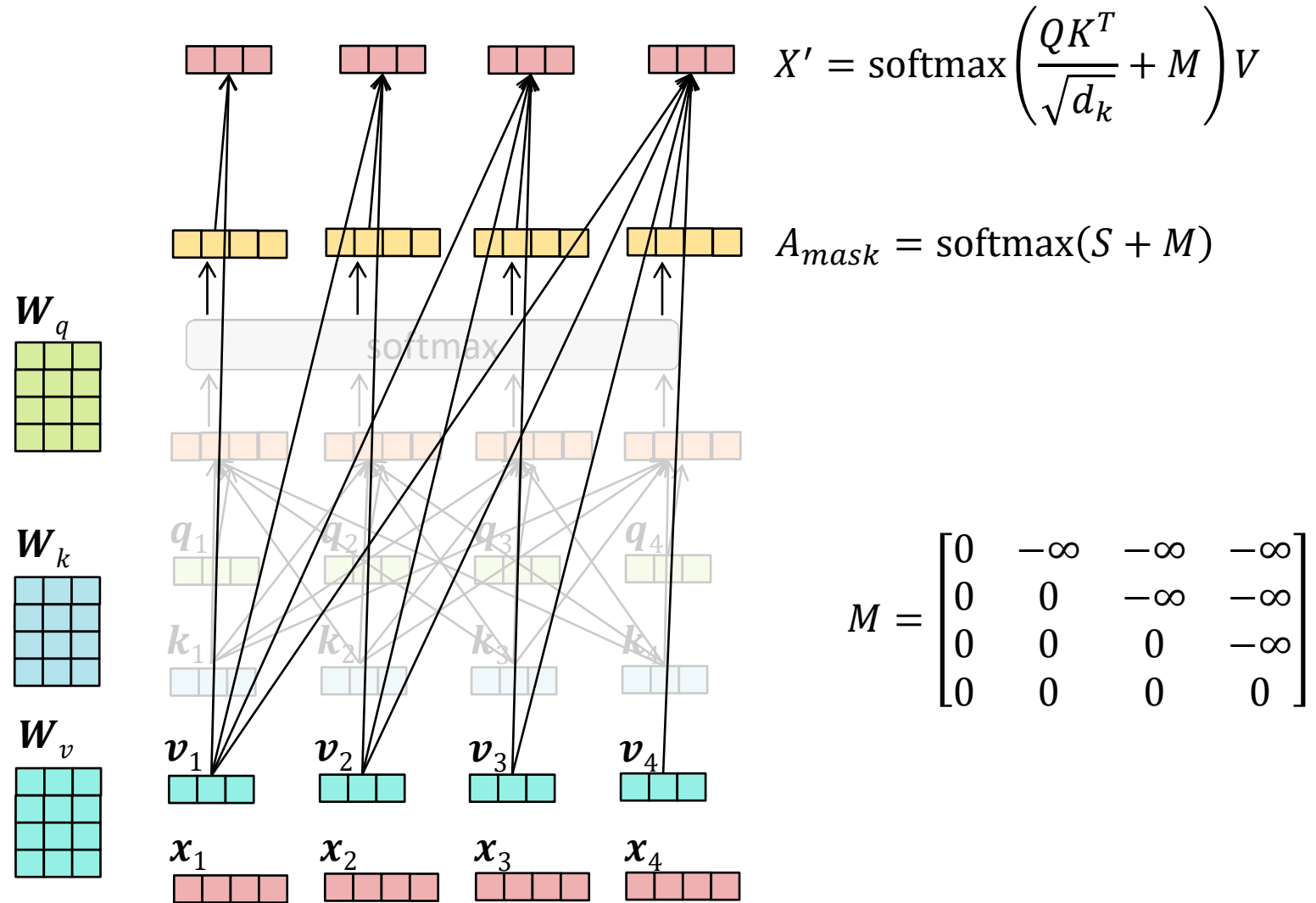


Language Translation

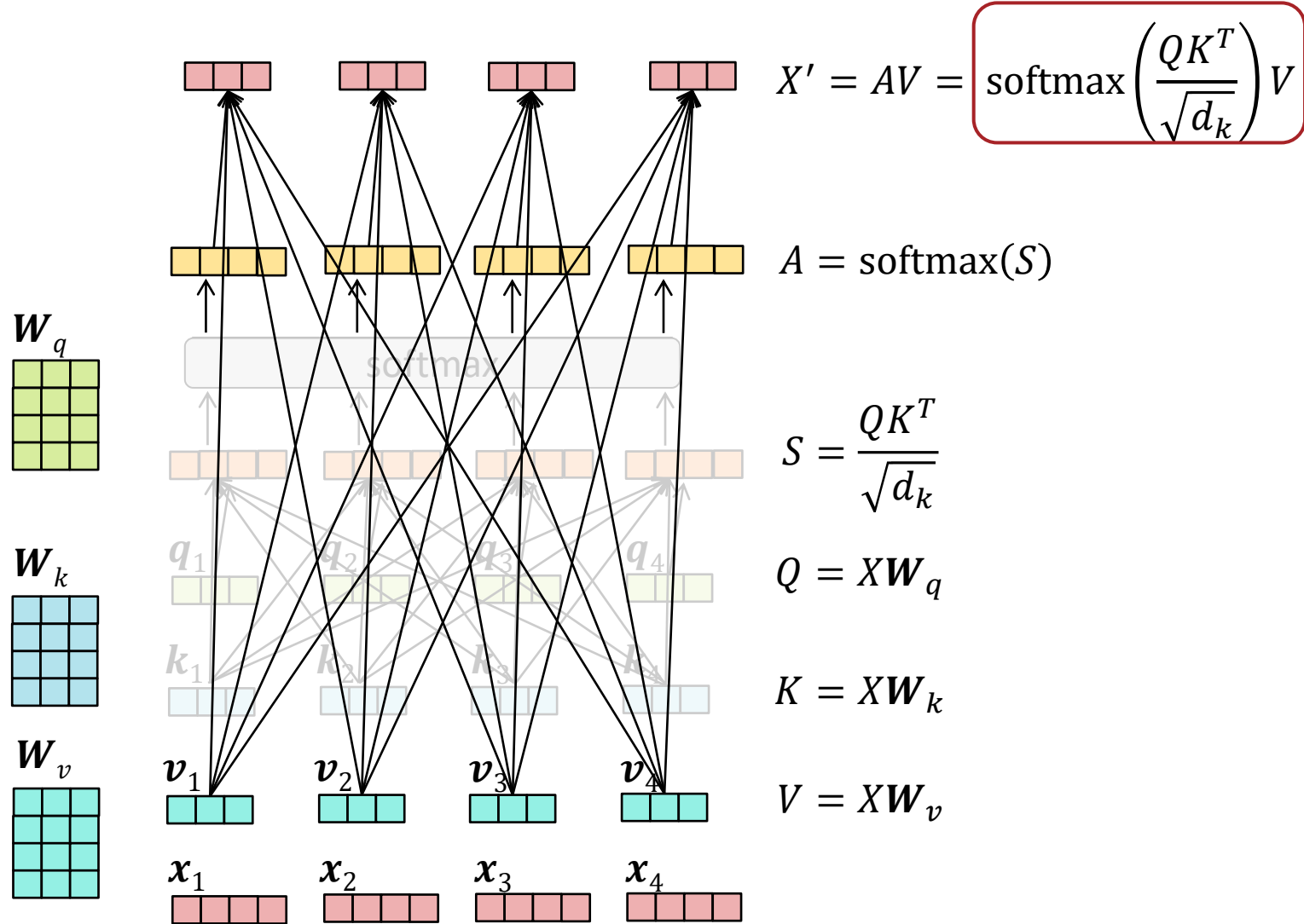


Decoder: Causal Attention

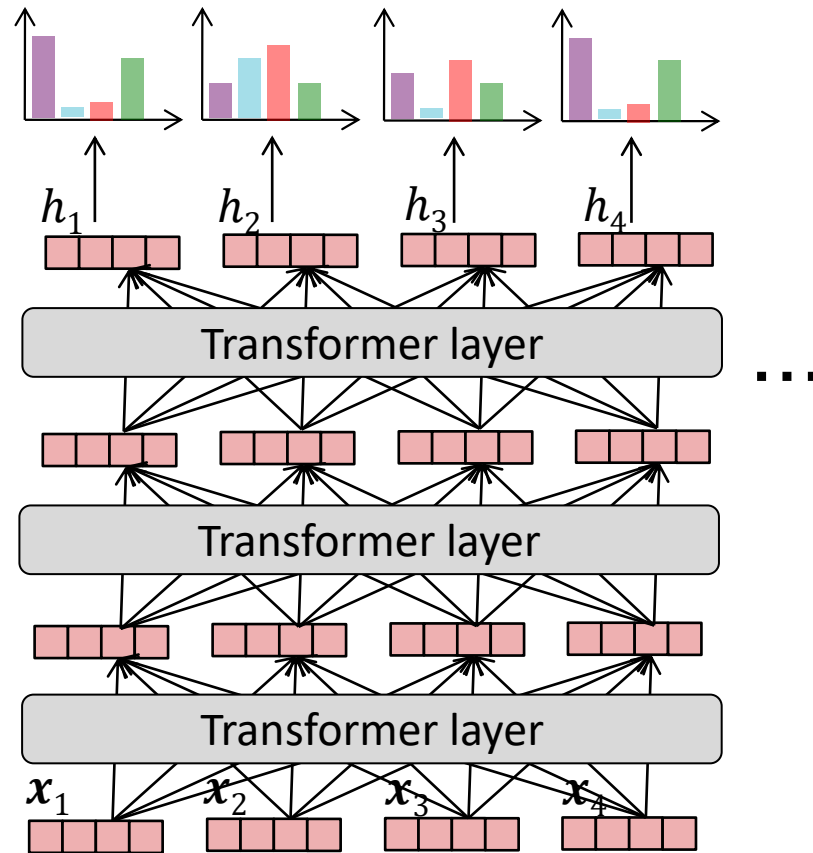
Idea: we can effectively delete or “mask” some of these arrows by selectively setting attention weights to 0



Encoder: Full Attention



Encoder-only Transformer

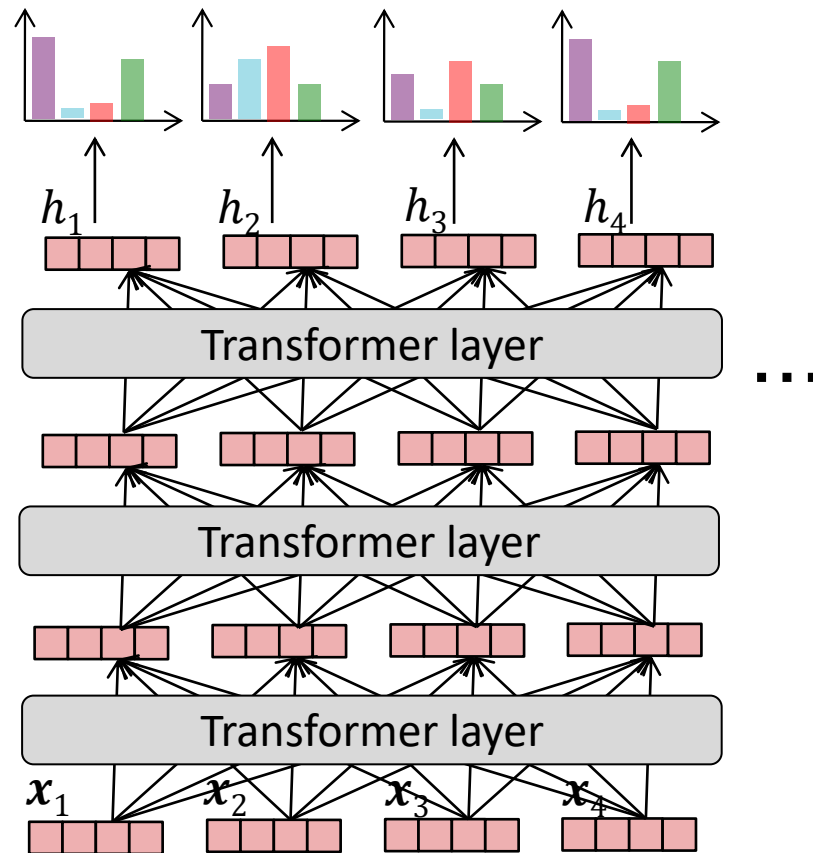


Each layer of a Transformer LM consists of:

1. non-causal attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of **all timesteps in the previous layer.**

Okay, but how would we train one of these things?



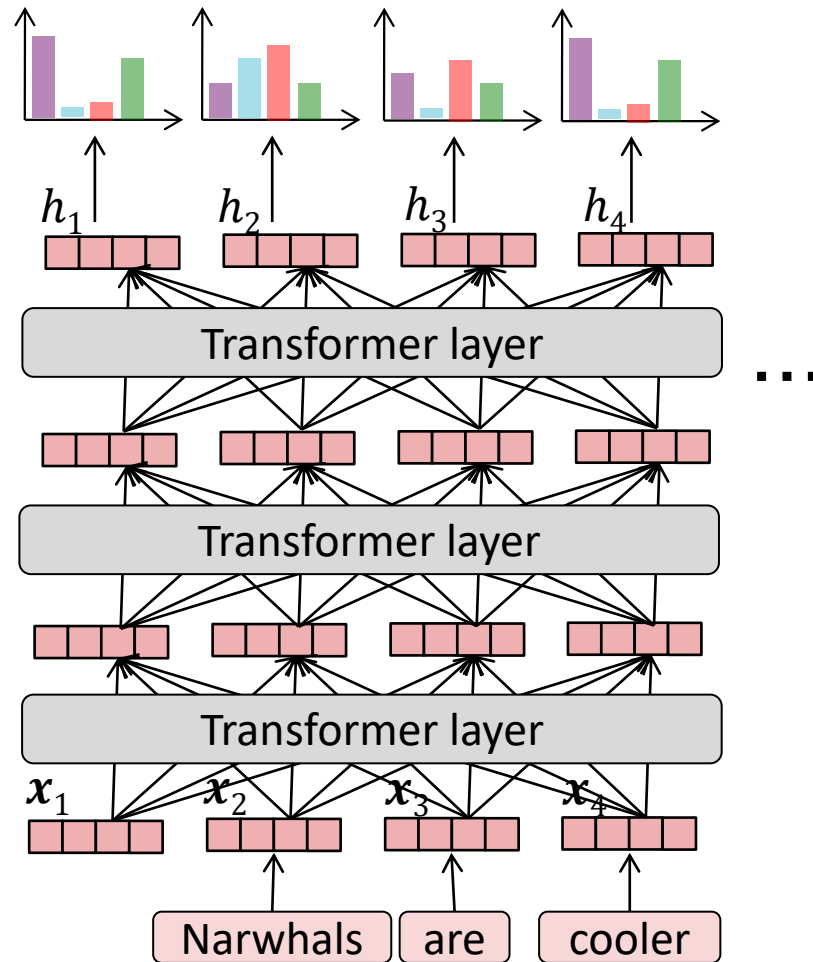
Each layer of a Transformer LM consists of:

1. non-causal attention
2. feed-forward neural network
3. layer normalization
4. residual connections

...

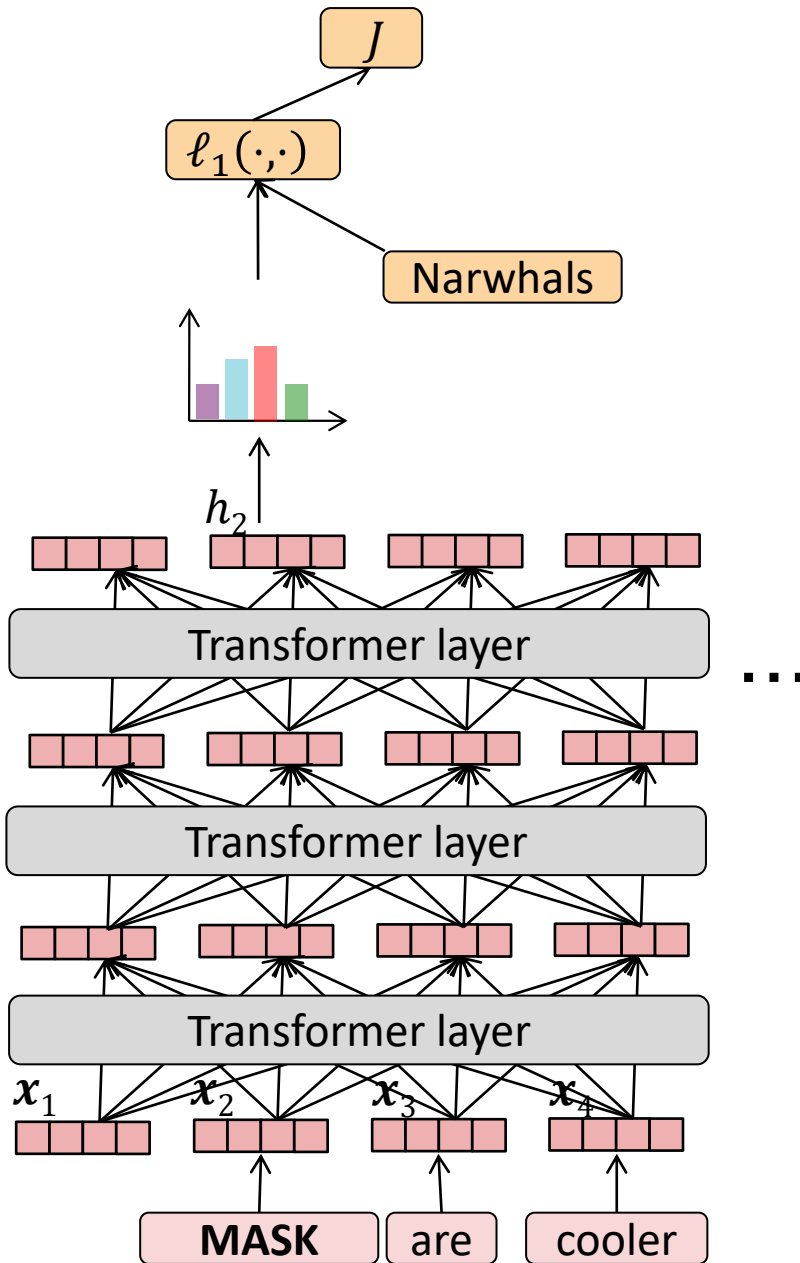
Each hidden vector looks back at the hidden vectors of **all timesteps in the previous layer.**

Masked Language Model Pre-training



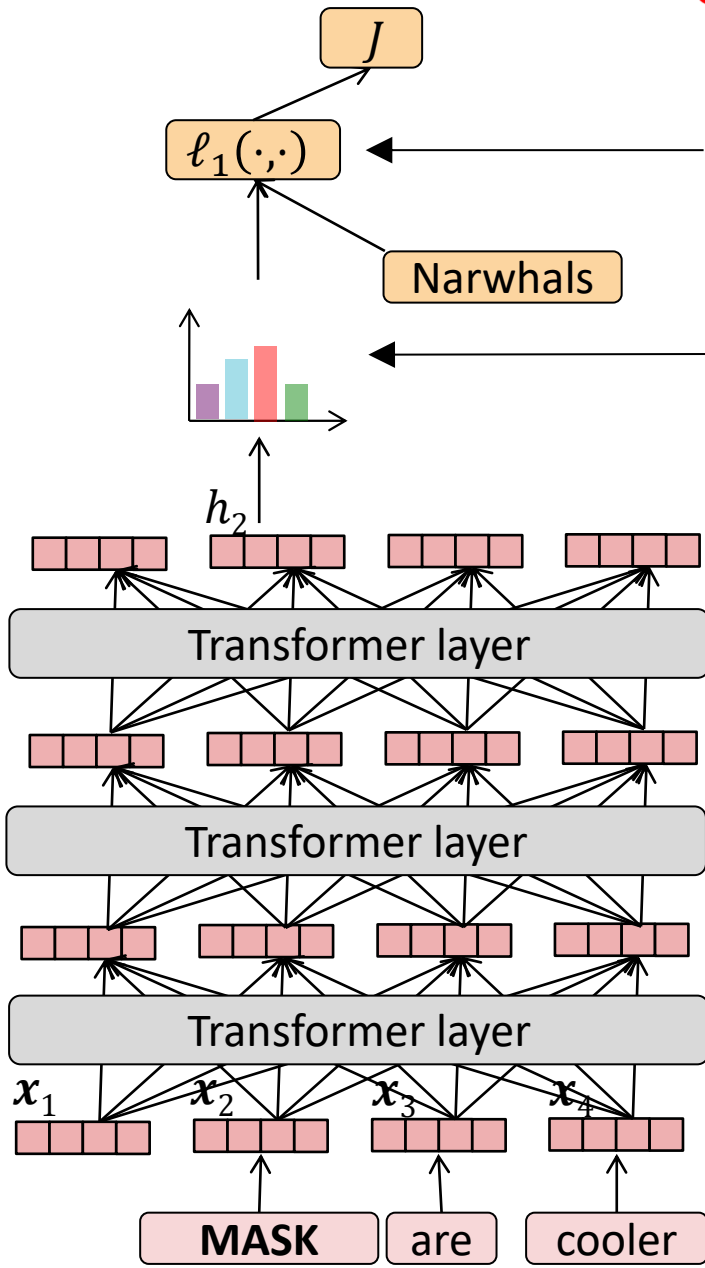
Rather than trying to predict the next token, *mask* out a few tokens in the sequence and train the model to predict the masked tokens.

Masked Language Model Pre-training



Rather than trying to predict the next token, *mask* out a few tokens in the sequence and train the model to predict the masked tokens.

Masked Language Model Pre-training

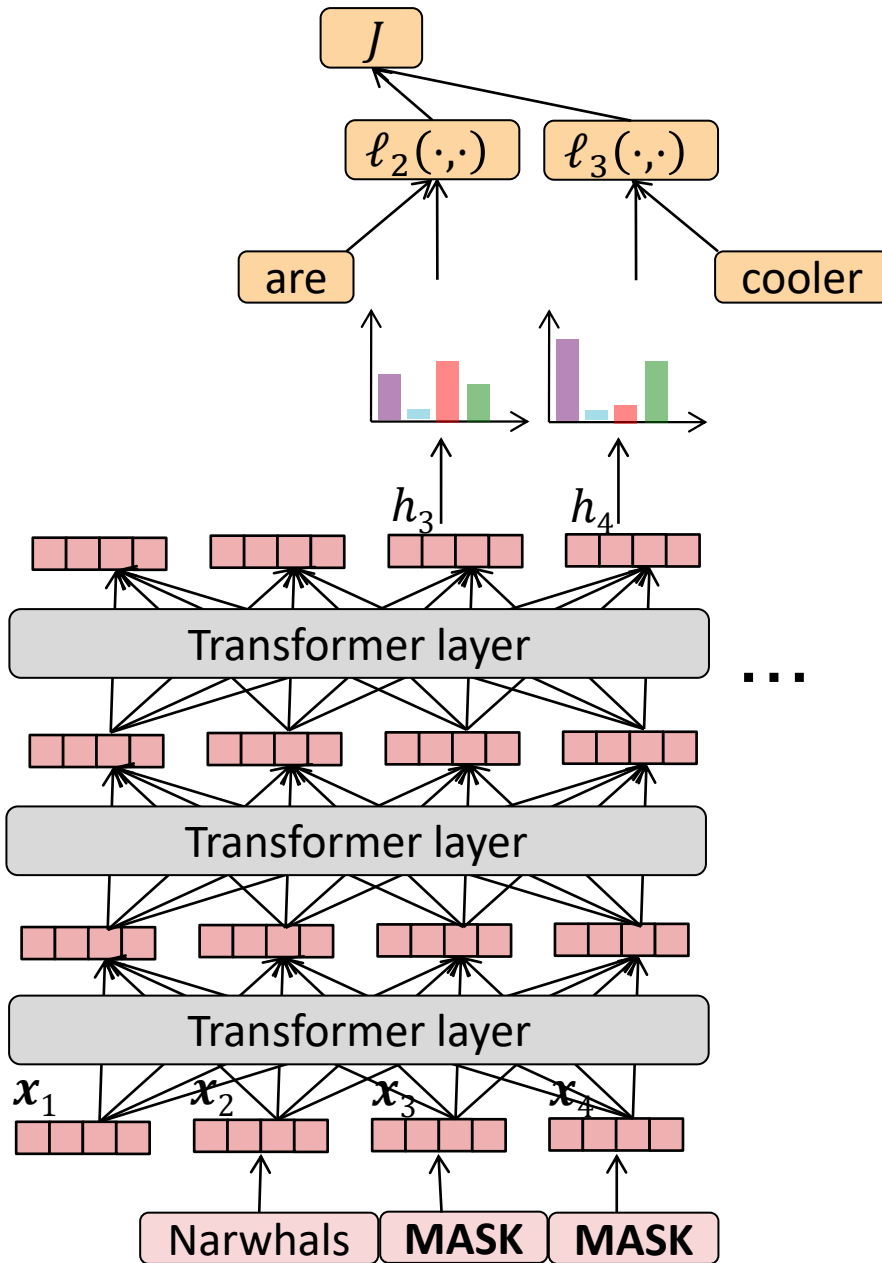


$-\log p(w_1 = \text{Narwhals} | w_2, w_3)$

What is this loss?

What is this probability distribution?
 $P(w_1 | w_2, w_3)$

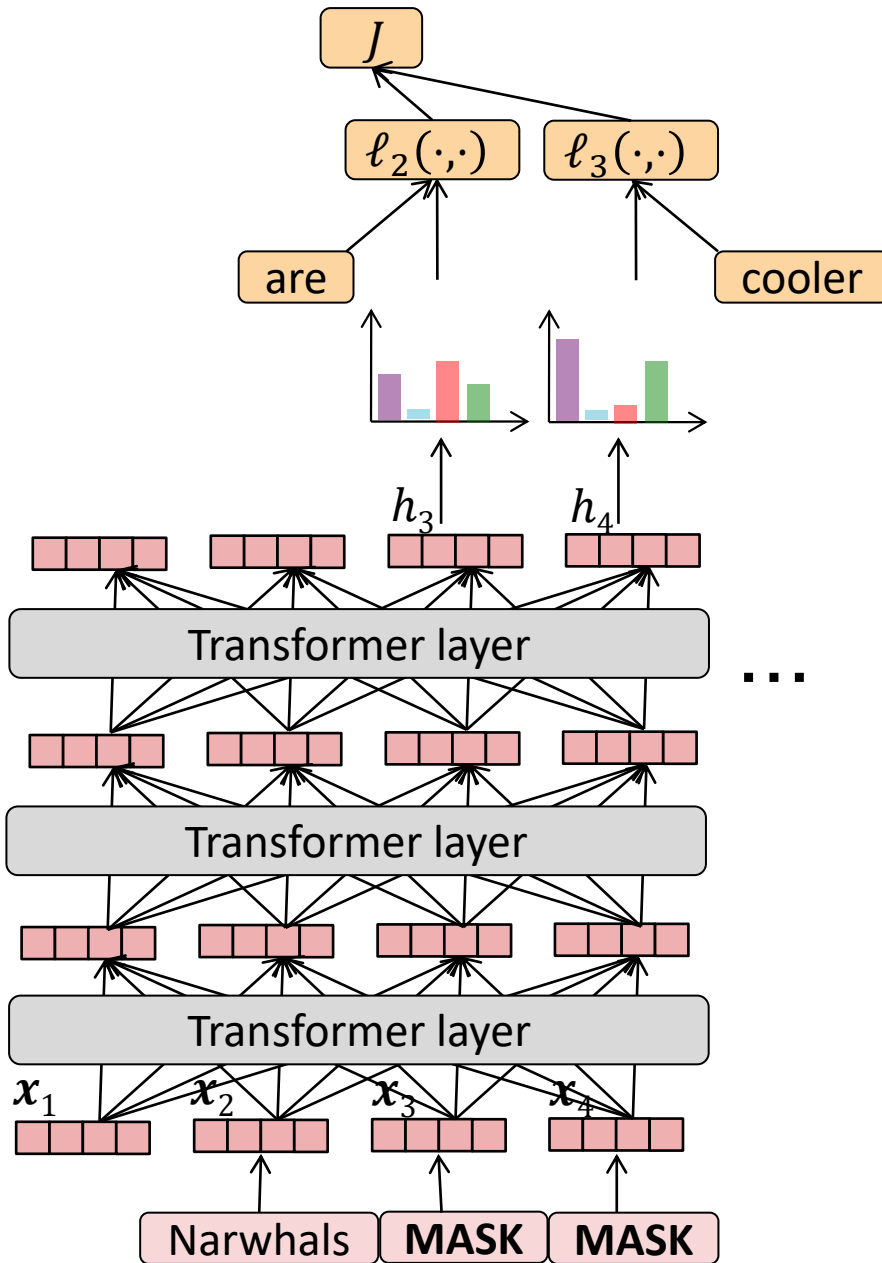
Masked Language Model Pre-training



Rather than trying to predict the next token, *mask* out a few tokens in the sequence and train the model to predict the masked tokens.

This kind of pre-training was popularized by the BERT language model

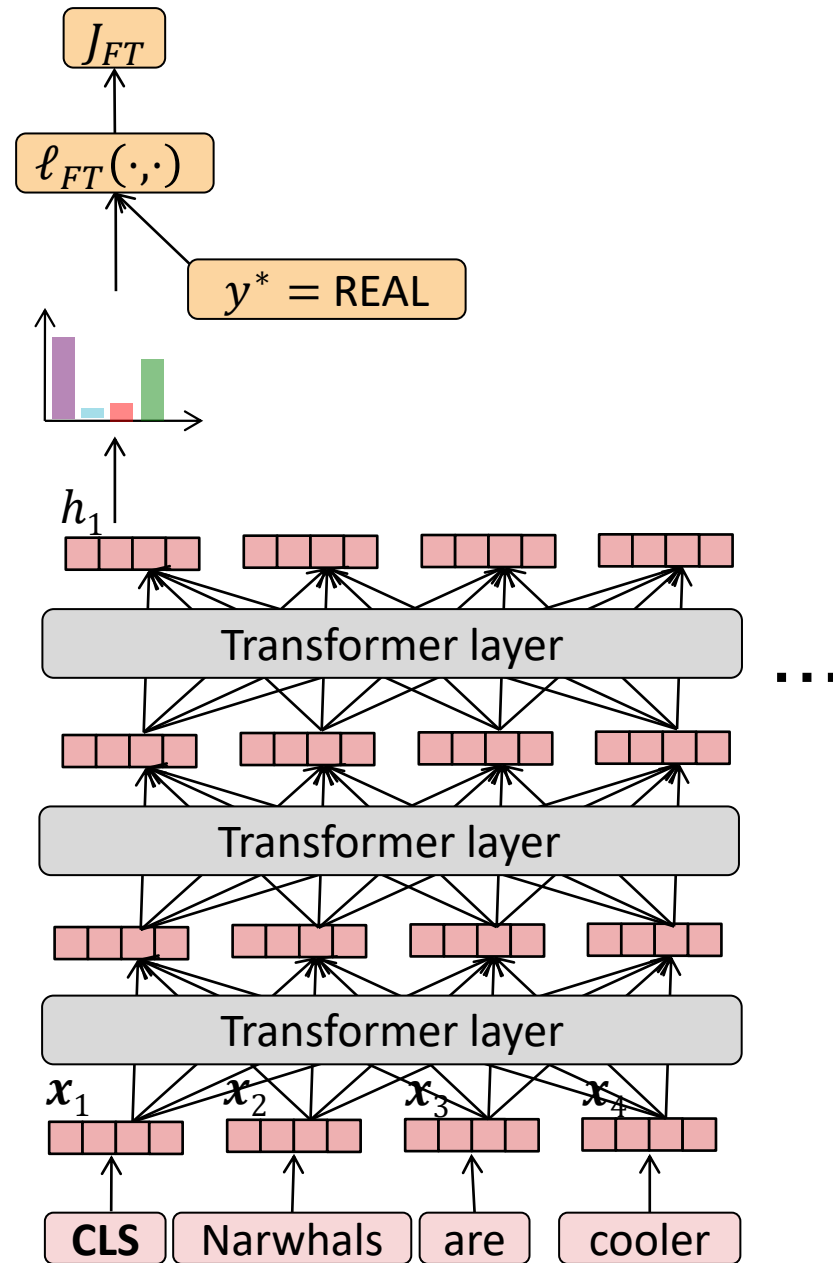
Masked Language Model Pre-training



Rather than trying to predict the next token, *mask* out a few tokens in the sequence and train the model to predict the masked tokens.

This kind of **pre-training** was popularized by the BERT language model

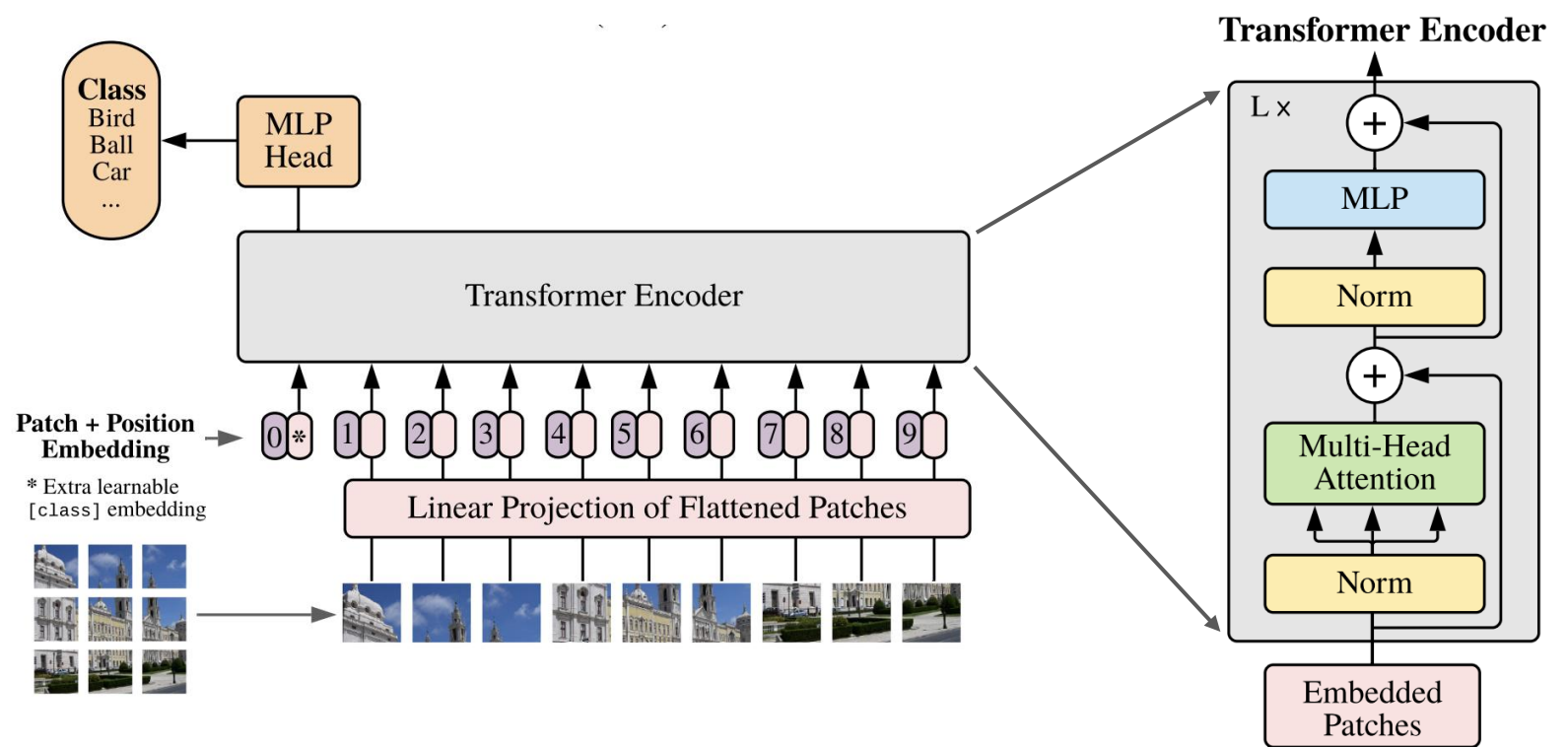
Supervised Fine-tuning



Prepend a special class token and *fine-tune* the (pre-trained) model to predict the label for each sequence

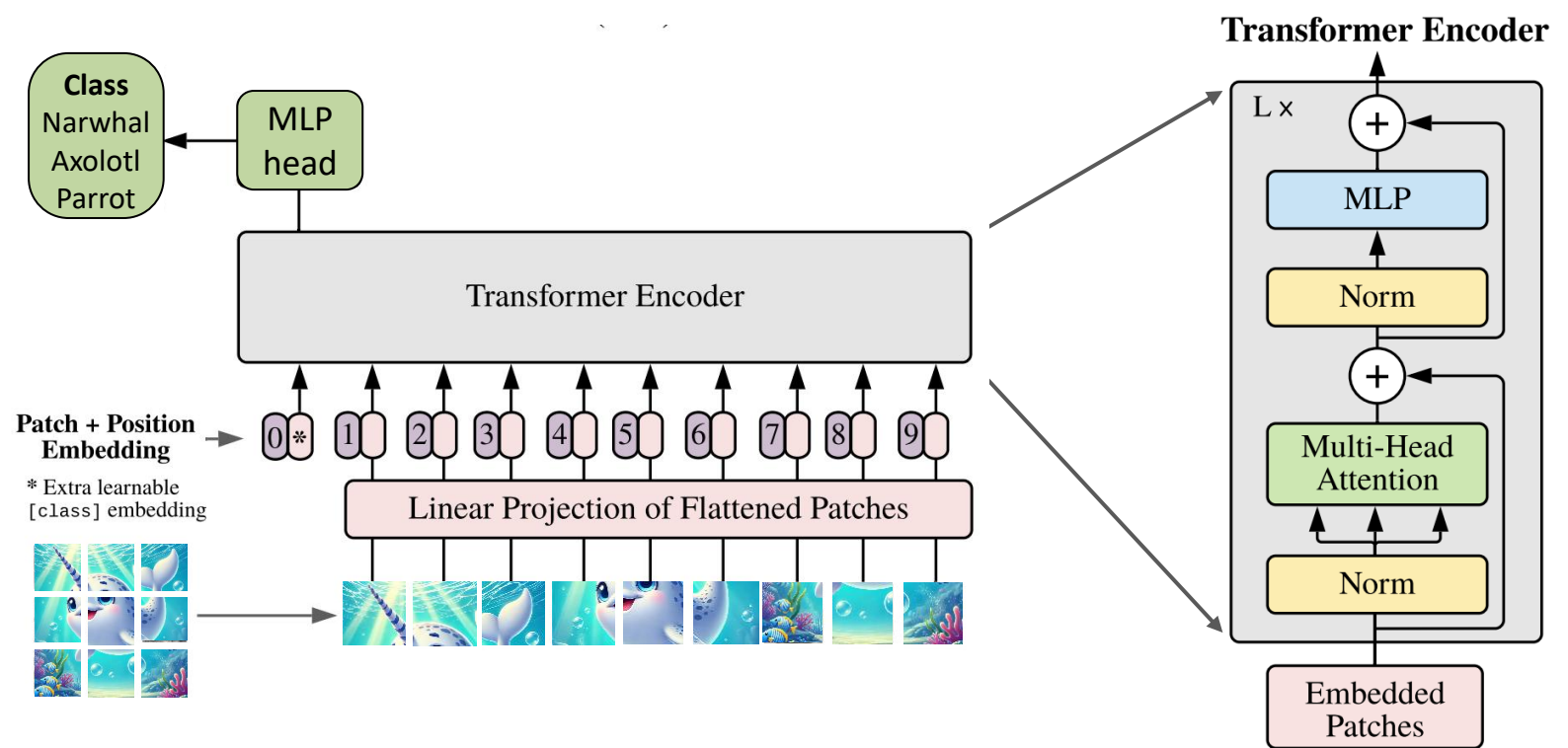
This model is not generative but has been shown to be a highly effective discriminator on a variety of tasks

Vision Transformer (ViT)



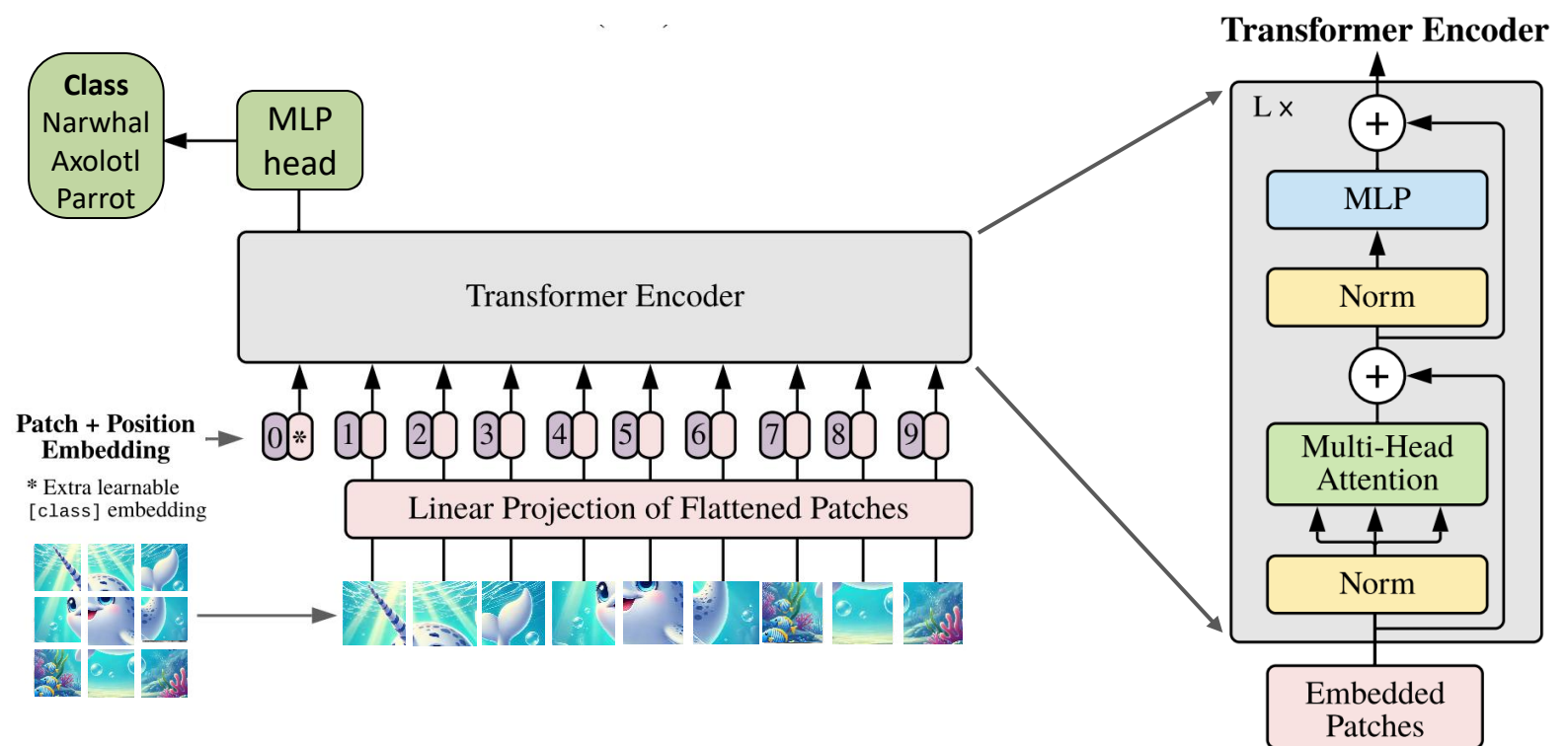
- Instead of words as input, the inputs are $P \times P$ pixel patches
- Each patch is embedded linearly into a vector of size 1024
- Uses 1D positional embeddings
- Pre-trained on a large, supervised dataset (e.g., ImageNet 21K, JFT-300M)

Vision Transformer (ViT)



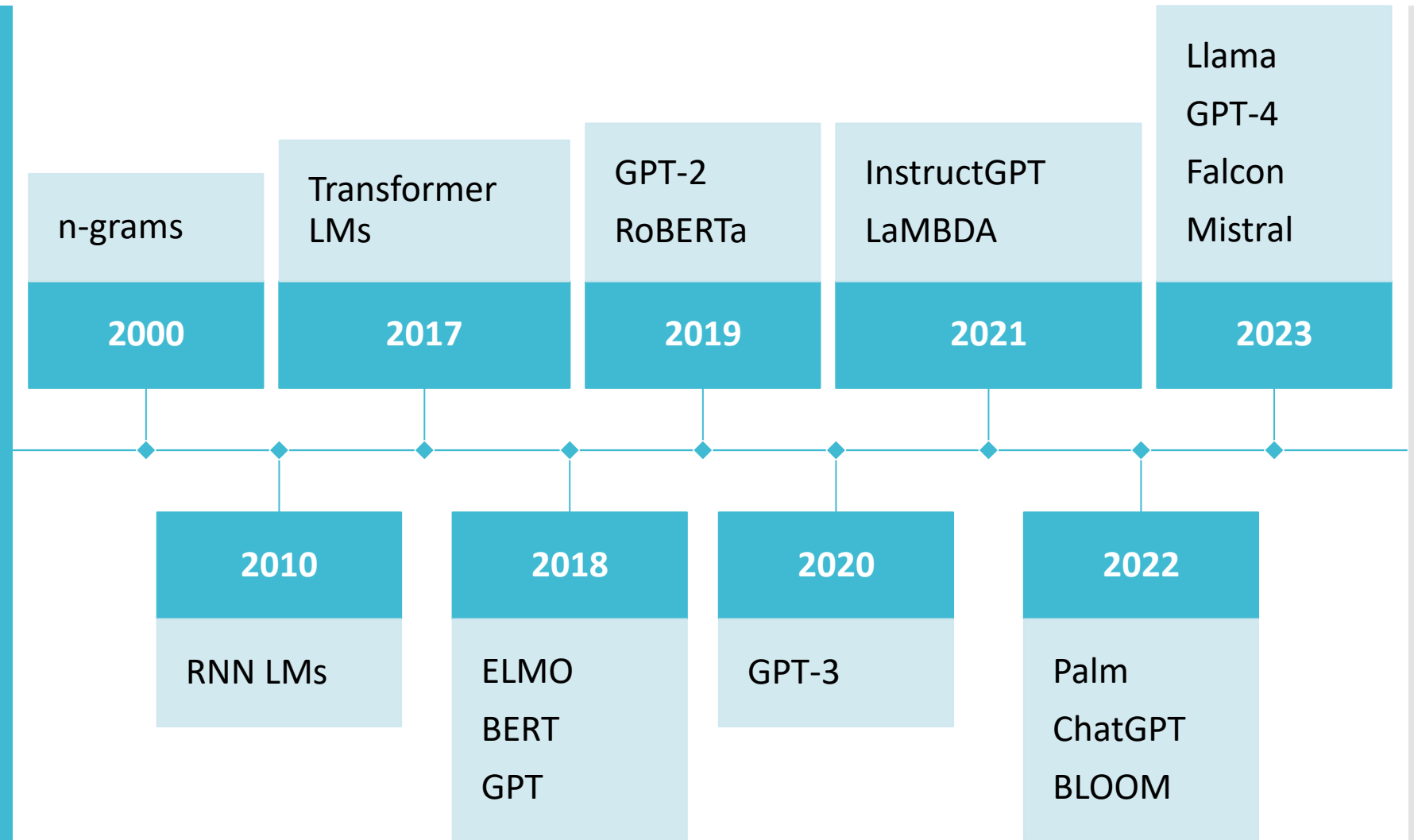
- Instead of words as input, the inputs are $P \times P$ pixel patches
- Each patch is embedded linearly into a vector of size 1024
- Uses 1D positional embeddings
- Can be fine-tuned by learning a new classification head on some (small) target dataset (e.g., CIFAR-100)

How can a ViT learn 2D positional information from a 1D positional embedding?

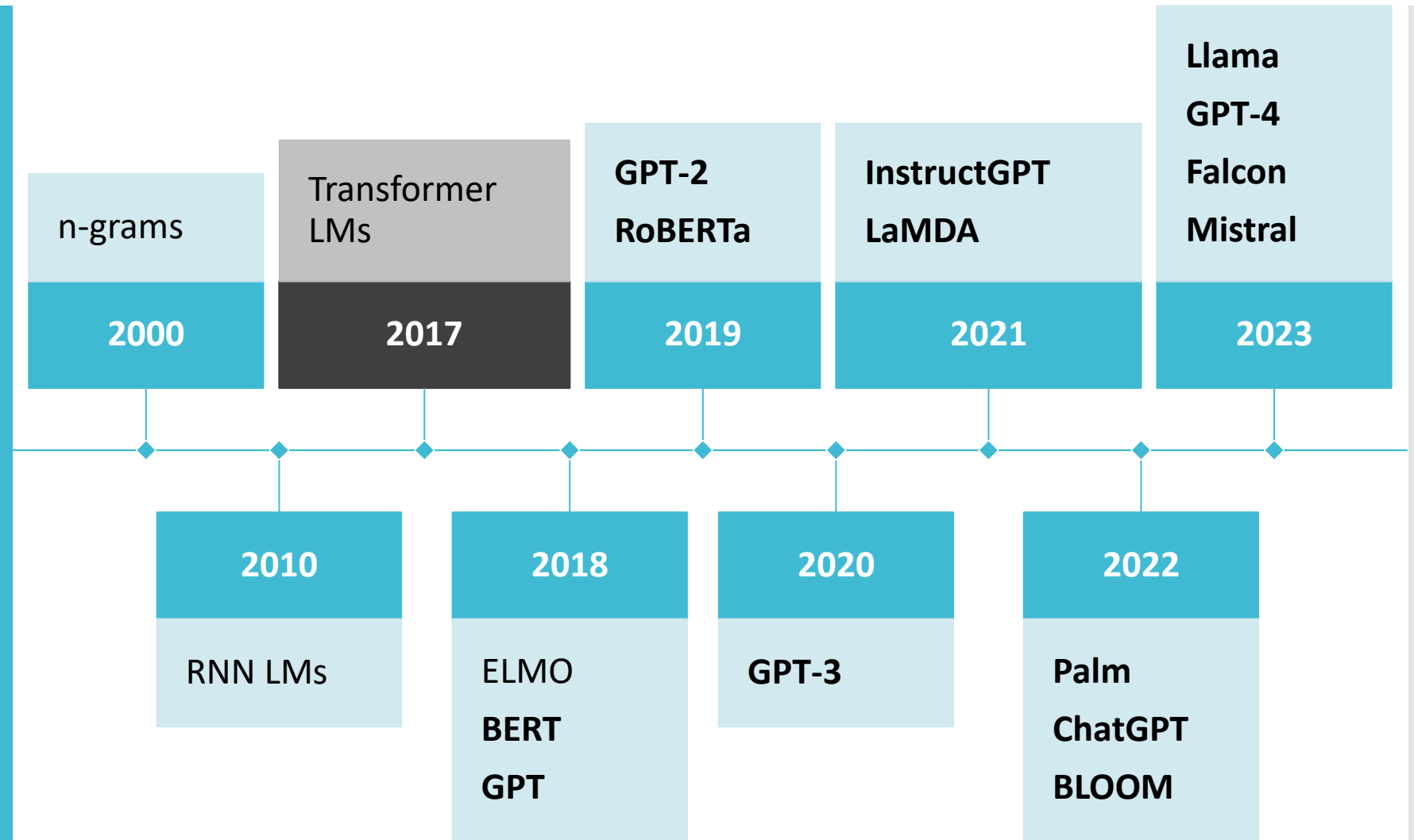


- Instead of words as input, the inputs are $P \times P$ pixel patches
- Each patch is embedded linearly into a vector of size 1024
- **Uses 1D positional embeddings**
- Can be fine-tuned by learning a new classification head on some (small) target dataset (e.g., CIFAR-100)

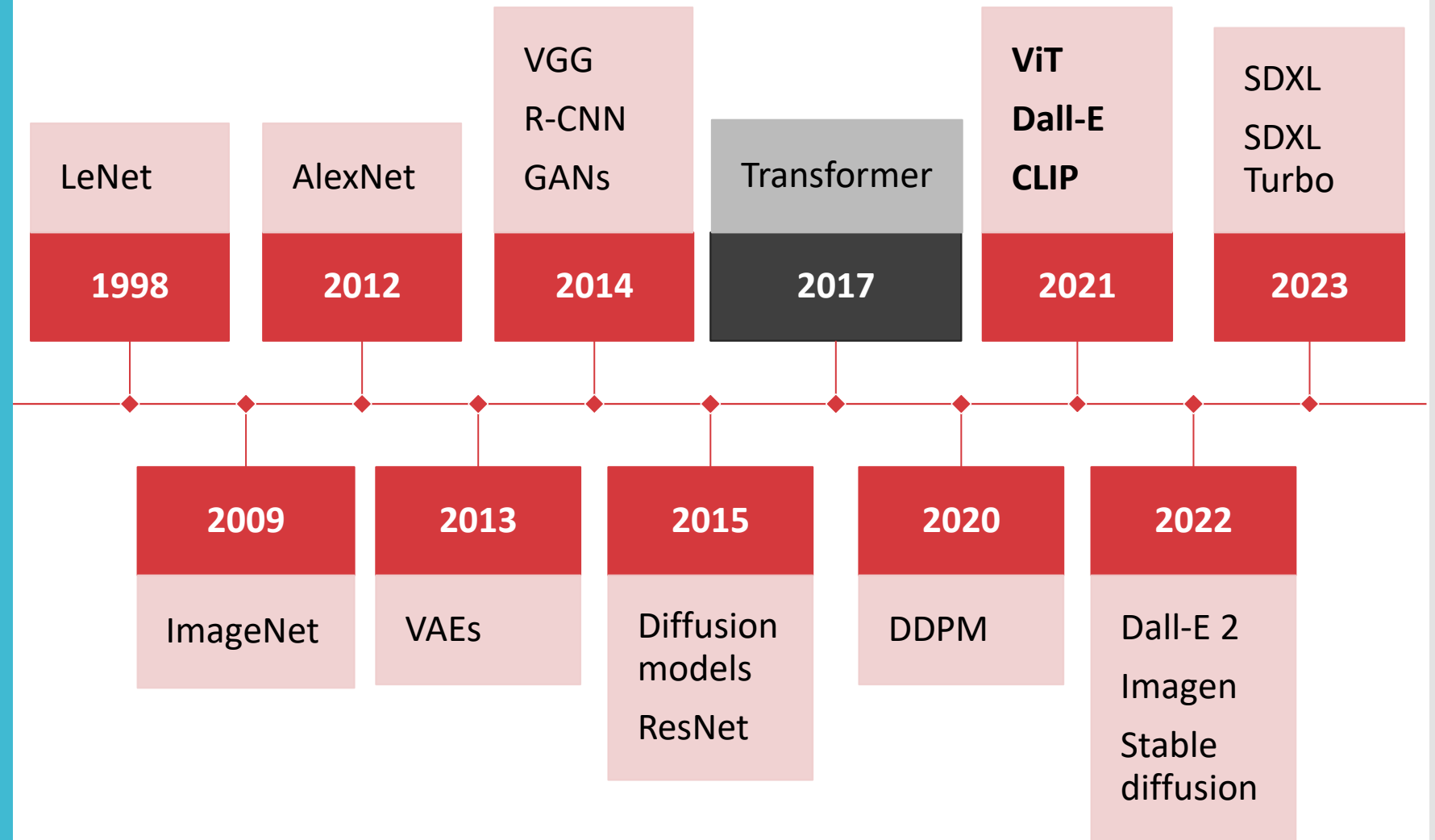
Language Modelling: Timeline



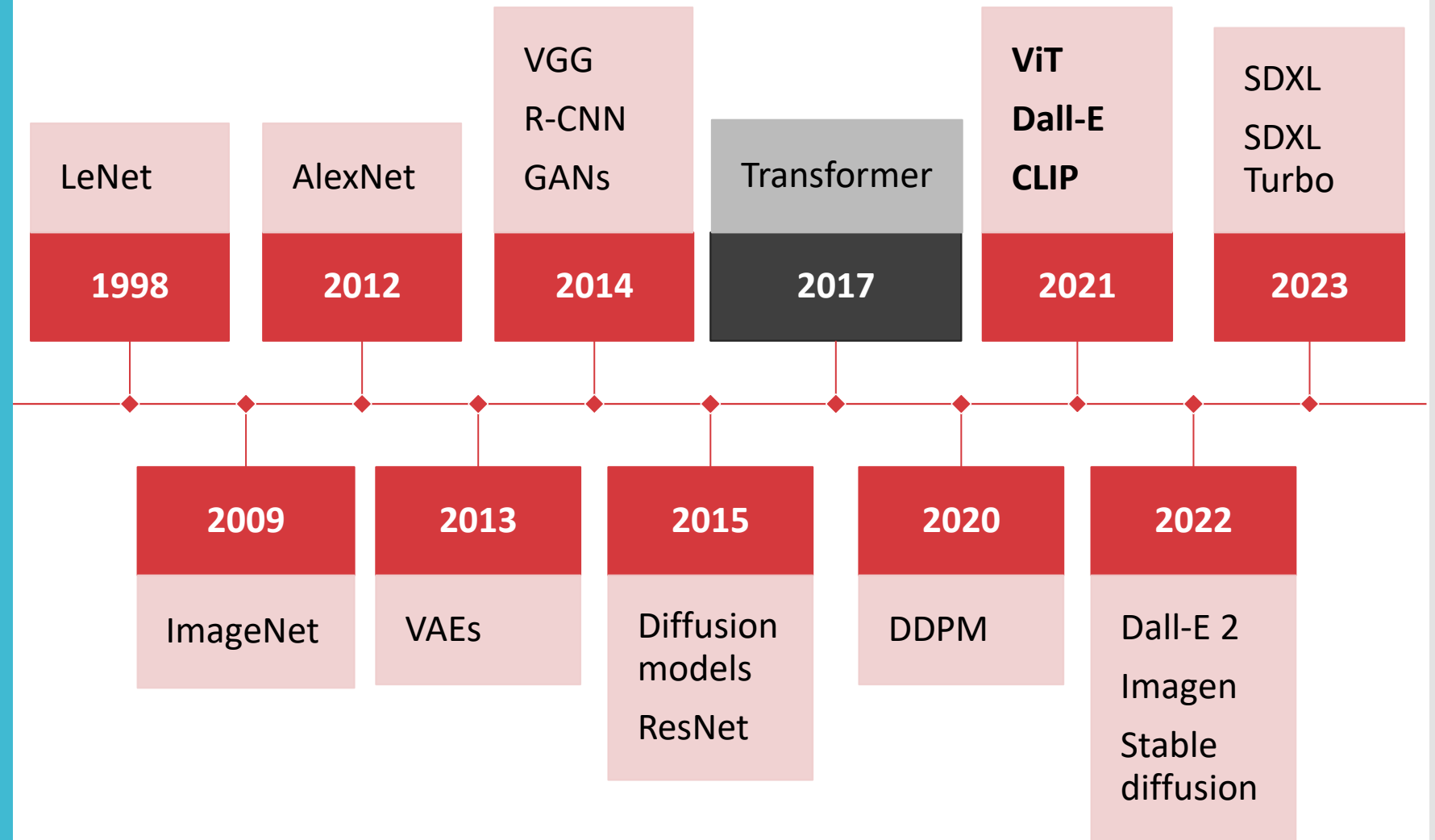
Language Modelling: Timeline



Language Modelling: Timeline



Why did Transformers take so long to gain traction in computer vision?



Why did
Transformers
take so long to
gain traction in
computer
vision?

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy^{*,†}, Lucas Beyer^{*}, Alexander Kolesnikov^{*}, Dirk Weissenborn^{*},
Xiaohua Zhai^{*}, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby^{*,†}

^{*}equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulby}@google.com

When trained on mid-sized datasets such as ImageNet without strong regularization, these models yield modest accuracies of a few percentage points below ResNets of comparable size. This seemingly discouraging outcome may be expected: **Transformers lack some of the inductive biases inherent to CNNs**, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data.

However, the picture changes if the models are trained on larger datasets (14M-300M images). We find that **large scale training trumps inductive bias**. Our Vision Transformer (ViT) attains excellent results when pre-trained at sufficient scale and transferred to tasks with fewer datapoints. When pre-trained on the public ImageNet-21k dataset or the in-house JFT-300M dataset, ViT approaches or beats state of the art on multiple image recognition benchmarks. In particular, the best model reaches the accuracy of 88.55% on ImageNet, 90.72% on ImageNet-Real, 94.55% on CIFAR-100, and 77.63% on the VTAB suite of 19 tasks.

Why did Transformers take so long to gain traction in computer vision?

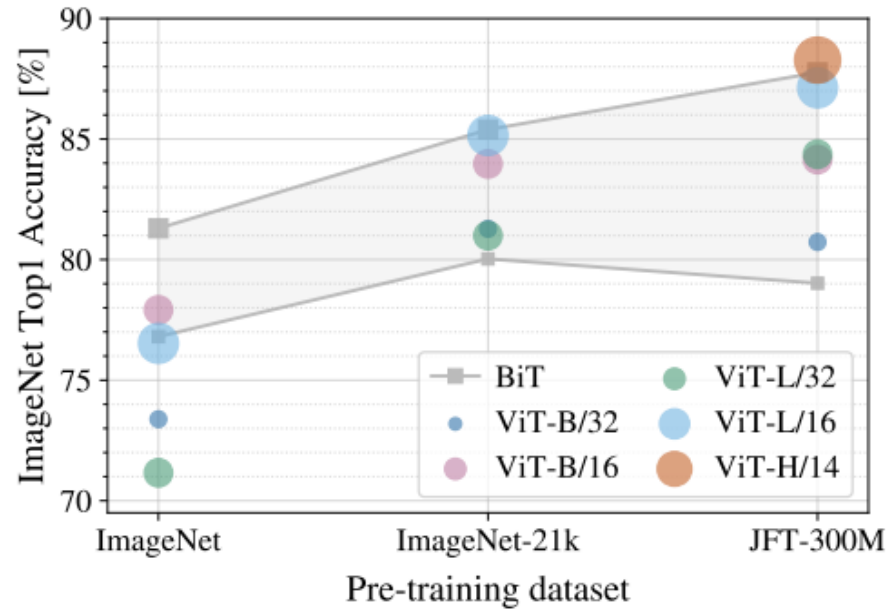


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

Wait, hang on:
is this even a
generative
model?

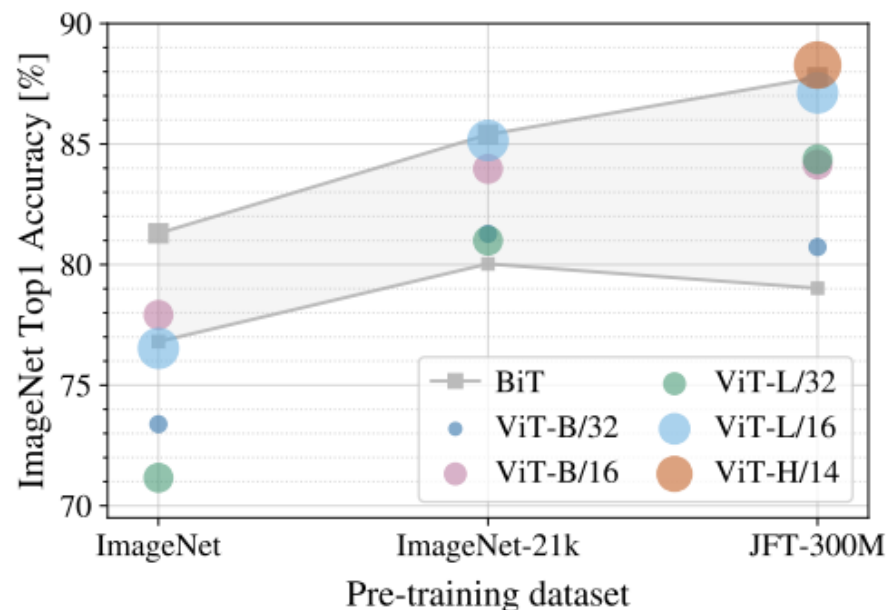


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

Common Tasks in Computer Vision

- Image Classification
 - Object Localization
 - Object Detection
 - Semantic Segmentation
 - Instance Segmentation
 - Image Captioning
 - **Image Generation**
- Class-conditional generation
 - Super resolution
 - Image Editing
 - Style transfer
 - Text-to-image (TTI) generation