Advanced Architectures for Vision

Anurag Arnab

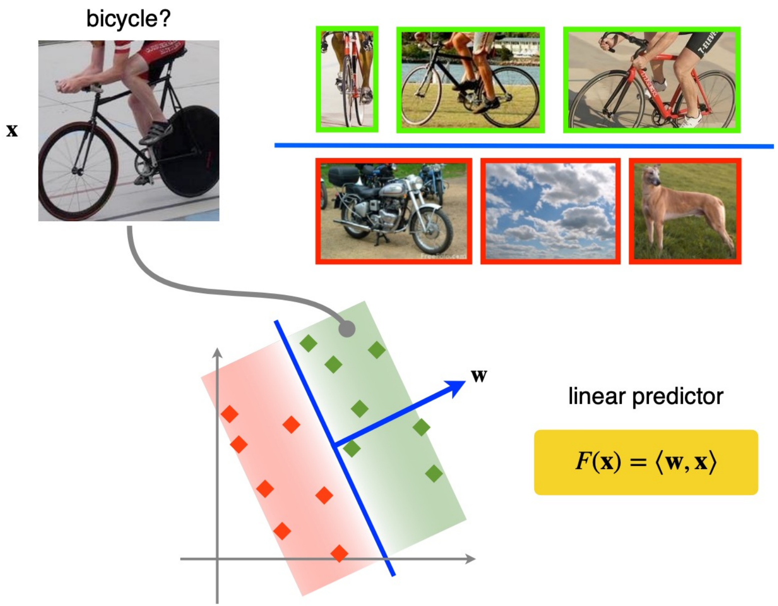
Google DeepMind

Outline

- Supervised learning fundamentals
    - Linear classifiers
    - Perceptrons
    - Convolutional Networks
- Transformers
    - Transformer deep-dive
    - Architectures for specialized tasks
- Connecting Vision and Language
    - Image-text models
    - Large Language Models
    - Vision Language Models

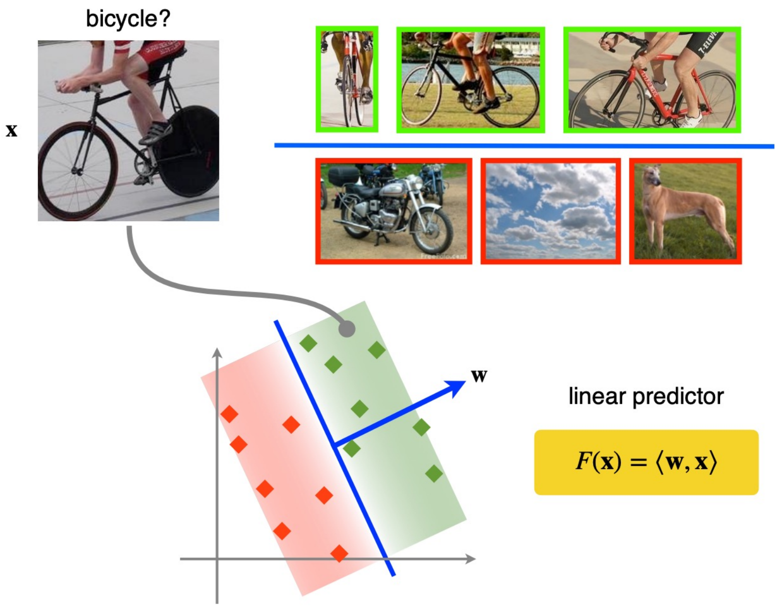
Linear Predictor

- We want to train a classifier that predicts whether an image, x, contains a certain class (ie ”bicycle”)
- We can learn this function, $F(x)$, from images that have, or do not have, the object.



Linear Predictor

- In the simplest case, the function is a linear classifier, $F(x)$.
    - Images are high dimensional vectors.
    - Compute the dot product, between a parameter vector, $\boldsymbol{w}$, and image, $\boldsymbol{x}$, to compute a score.
    - The sign of $F(x)$ is used as the prediction.



Data Representations

- We can apply a linear classifier to vectors, x.
- However, we want to process images, videos or other data that are not necessarily vectors.
- Representation function, $\phi(x)$, maps data to vectors.

- Non-linear classifier by applying linear predictor to non-linear representation function.

$\mathbf{x}$

representation

$\Phi(\mathbf{x}) \in \mathbb{R}^d$

possibly not a vector

A vector

linear predictor

$$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

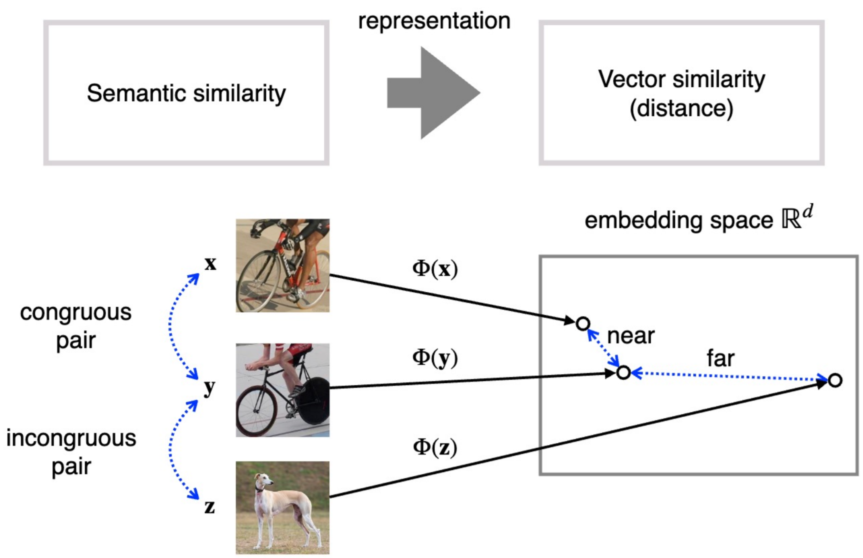
non-linear predictor

$$F(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$$

Meaningful Representations

- Representation should help a linear classifier to perform classification.
- *Semantic similarity* between data points needs to be mapped to a *vector similarity*.
- Therefore, a good representation needs to:
  - Be invariant to nuisance factors
  - Sensitive to semantic factors.

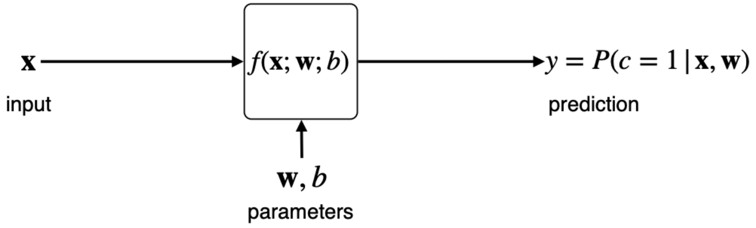
- How do we choose ϕ?

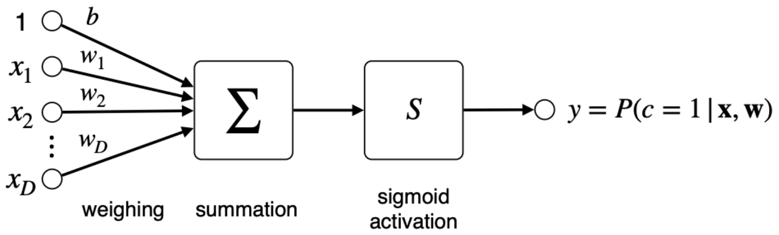


Google DeepMind

Perceptron

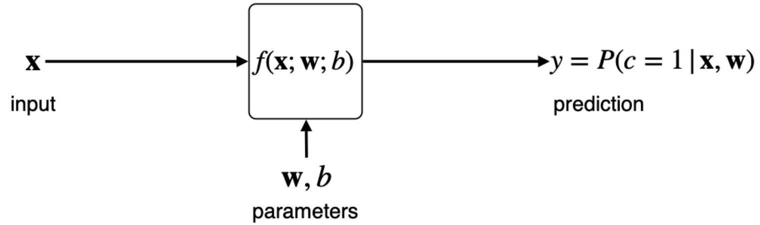
- One of, if not, the first neural networks by [Rosenblatt, 1957](#).
- The perceptron maps an input vector, x, to a probability, y.
- Example: y is the probability that image x is a bicycle.

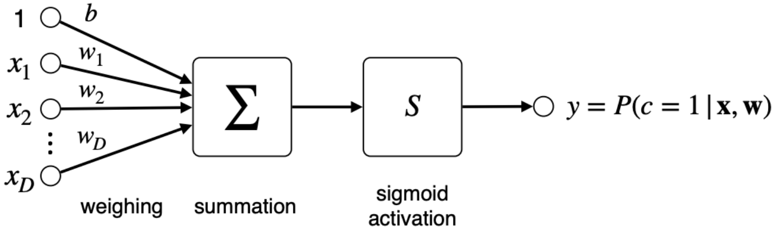




Perceptron

- Computes the probability by computing a weighted sum of the input with a learned vector w, and then applying a non-linear sigmoid activation function.
- Sigmoid makes the perceptron non-linear.
- Perceptron is effectively a linear classifier with a sigmoid activation function.

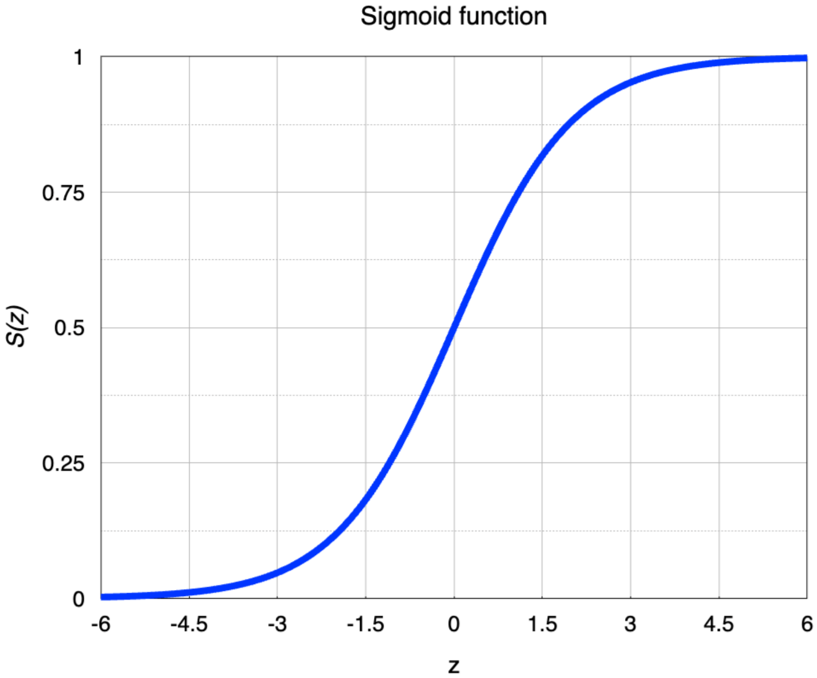




Google DeepMind

Sigmoid Function

- Non-linear activation function of the perceptron.
- $S(z) = \frac{1}{1+e^{-z}}$
- Converts real values, z, in the range $(-\infty, \infty)$ into probabilities in the range $(0, 1)$.



Google DeepMind

Training a Perceptron

- Minimise the cross entropy loss.
- $$L(w) = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_i \log\big(f(\boldsymbol{x}_i;\boldsymbol{w})\big)}_{\text{For positive labels, } y\,=\,1} + \underbrace{(1 - y_i)\log\big(f(\boldsymbol{x}_i;\boldsymbol{w})\big)}_{\text{For negative labels, } y\,=\,0}$$

Why Cross-Entropy Loss?

- We want to maximise the likelihood, $p(y_i \,|x_i \,; \boldsymbol{w})$.
- If we assume we sample N examples in an independent and identically distributed (IID) manner, then
- $p(\boldsymbol{y} \,|\boldsymbol{x}) = \prod_i p(y_i \,|x_i)$.
- And so to learn parameters, $\boldsymbol{w}$, we want to maximise
- $\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, p(\boldsymbol{y} \,|\boldsymbol{x} \,; \boldsymbol{w}) = \prod_i p(y_i \,|x_i \,; \boldsymbol{w})$.

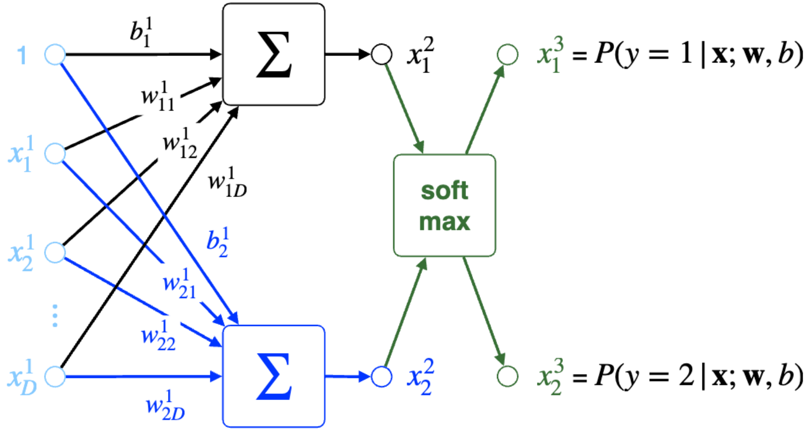
Google DeepMind

Why Cross-Entropy Loss?

- $\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, p(\boldsymbol{y}\,|\boldsymbol{x}\,;\boldsymbol{w}) = \prod_i p(y_i\,|x_i\,;\boldsymbol{w})$.
- If we take the logarithm, we obtain
- $\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \sum_i \log p(y_i\,|x_i\,;\boldsymbol{w})$.
- Since we like minimizing losses, we can minimise the negative of the log-likelihood
- $\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmax}} - \sum_i \log p(y_i\,|x_i\,;\boldsymbol{w})$.

Multi-Class Perceptron

- We can combine multiple perceptrons to predict more than one class.
- Each perceptron computes a score, $x_c^{(2)}$ for a class hypothesis of $c = 1, 2, \dots, C$.
  - Subscript denotes the class, superscript the layer index.
- The vector of scores, $\boldsymbol{x}^{(2)}$, is mapped to a vector of probabilities, $\boldsymbol{x}^{(3)}$, with a softmax function.

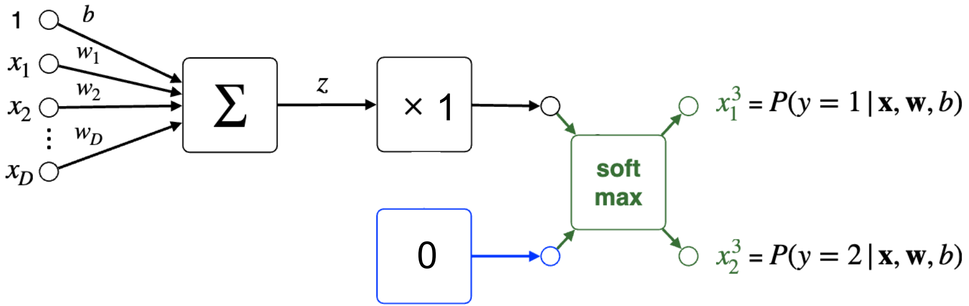


Softmax

- Maps a vector of scores to probabilities.
- $S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
- In the binary case, the softmax is the same as the sigmoid.

Softmax

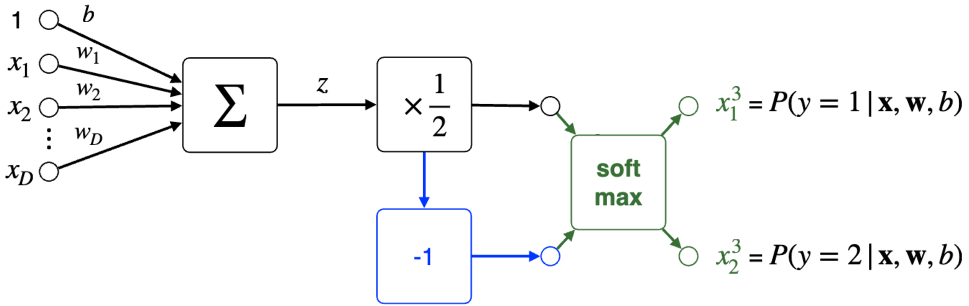
- Maps a vector of scores to probabilities.
- $S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
- In the binary case, the softmax is the same as the sigmoid.



$$x_1^3 = \frac{e^z}{e^z + e^0} \times \frac{e^{-z}}{e^{-z}} = \frac{1}{1 + e^{-z}}$$

Softmax

- Maps a vector of scores to probabilities.
- $S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
- In the binary case, the softmax is the same as the sigmoid.

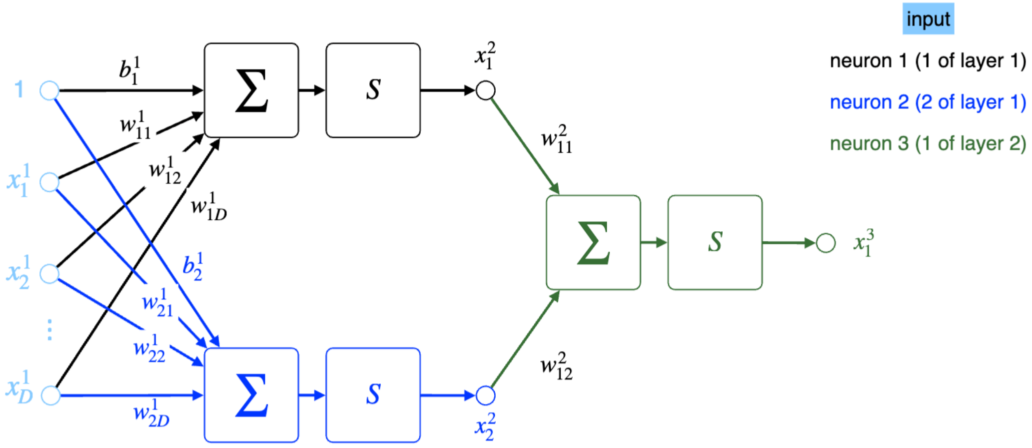


$$x_1^3 = \frac{e^{z/2}}{e^{z/2} + e^{-z/2}} = \frac{1}{1 + e^{-z}}$$

Google DeepMind

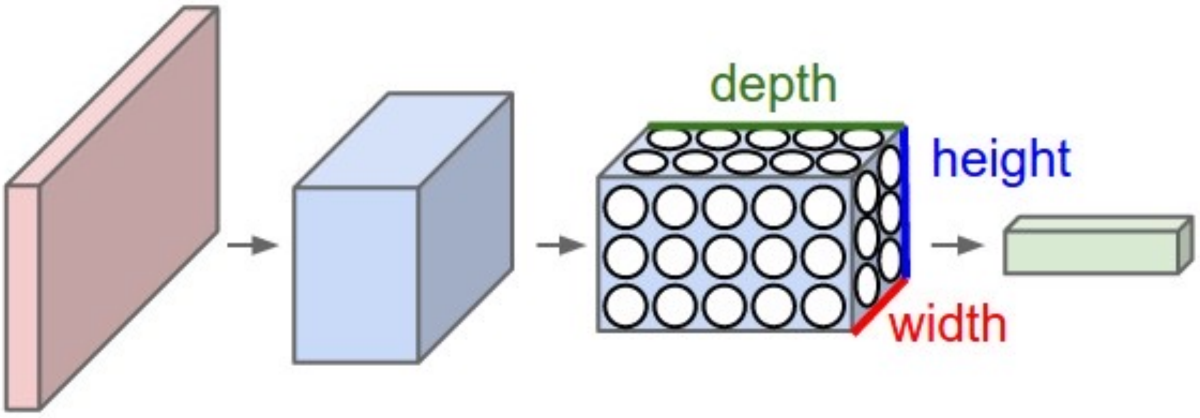
Multilayer Perceptron

- We can chain multiple perceptrons together, resulting in a deep neural network.
- Depth refers to the fact that the resulting function decomposes as a long, ”deep” chain of simpler perceptrons.
- 2-layer MLP is shown here



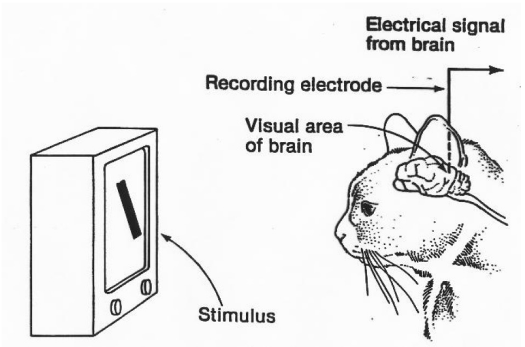
Convolutional Networks

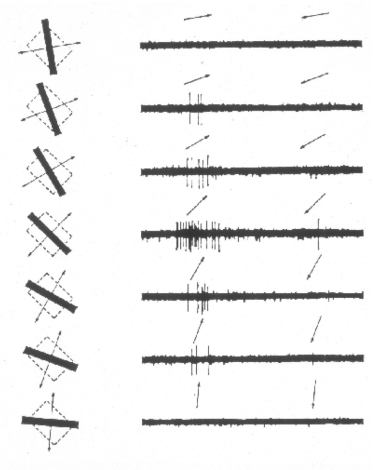
- Architectures designed specifically for images, operating on 2D (images) or 3D (video) grids.
- View data as a $B \times H \times W \times C$ grid, where B is the batch size, H the height, W the width and C the number of channels.



Biological Motivation

- Hubel & Wiesel conducted seminal experiments in understanding the visual cortex of mammals.
- In cats and monkeys, they found the existence of neurons in the brain that activate (by measuring with implanted electrodes) to specific orientations and locations of a visual stimulus.
- Therefore, these neurons behave like local, translation-invariant operators.
- They later won the Nobel Prize in Physiology and Medicine in 1981.
- Their work inspired the Neurocognitron architecture which may be the first CNN (1980).

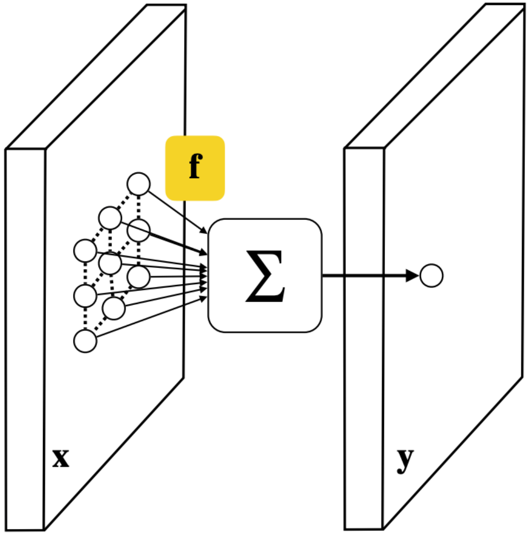




Google DeepMind

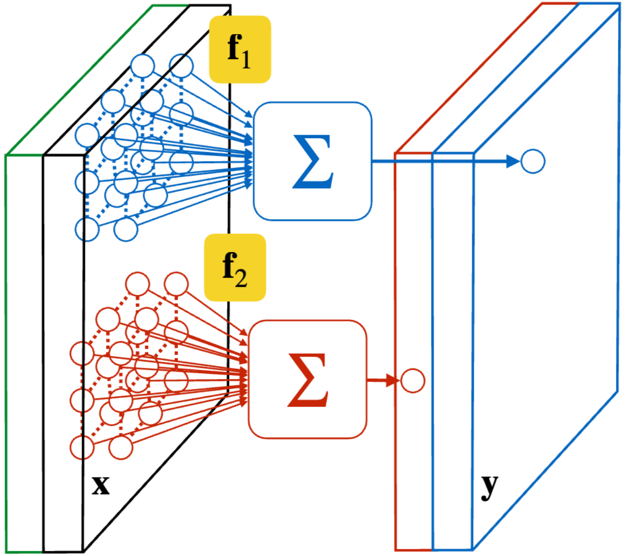
Convolution

- A linear filter, $\boldsymbol{f}$, computes the weighted summation of a window of the input, $\boldsymbol{x}$.
- Key properties
    - Linearity: Operation is linear in the input and the filter parameters.
    - Locality: Only looks at a small window of the data.
    - Translation invariance: All windows are processed using the same filter weights



Convolution

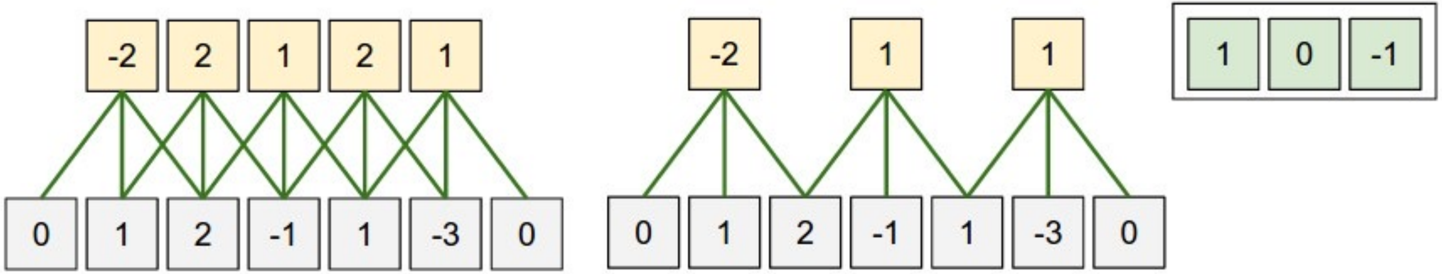
- A linear filter, f , computes the weighted summation of a window of the input, x.
- Key properties
  - Linearity: Operation is linear in the input and the filter parameters.
  - Locality: Only looks at a small window of the data.
  - Translation invariance: All windows are processed using the same filter weights.
- We use multiple filters – one output channel per filter.
  - Intuitively, each filter activates for a certain pattern in the input.



Google DeepMind

Convolution in 1D

- Convolve a filter (in green) with input (in grey) to get the output (in yellow).

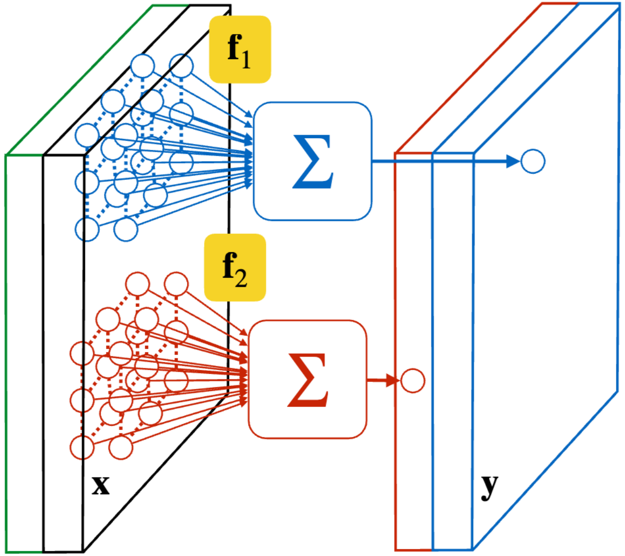


Stride = 1

Stride = 3

Convolution

- So if the input, $\boldsymbol{x}$, has shape $N \times H \times W \ \times C_{in}$
- The filter, f, has shape, $k_h \times k_w \times C_{in} \times C_{out}$
- The output $\boldsymbol{y}$, has shape (with stride 1)

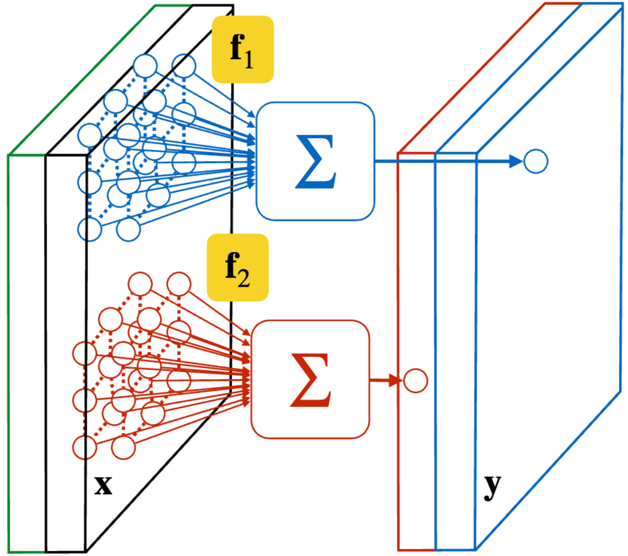


Google DeepMind

Convolution

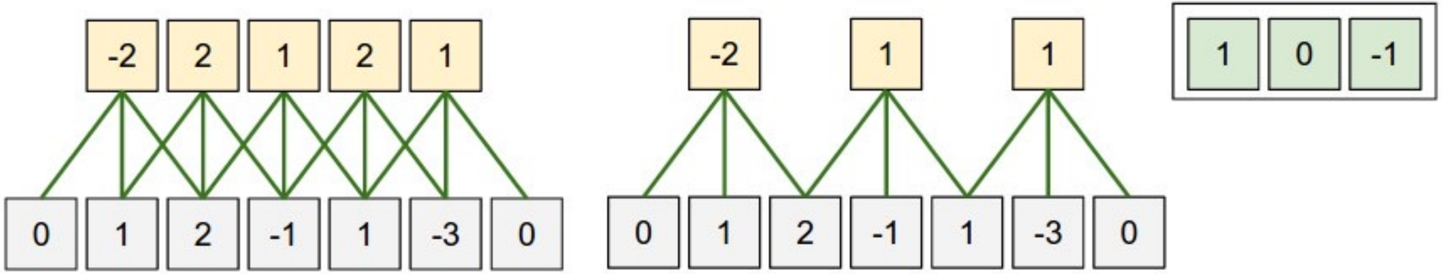
- So if the input, $\boldsymbol{x}$, has shape $N \times H \times W \times C_{in}$
- The filter, f, has shape, $k_h \times k_w \times C_{in} \times C_{out}$
- The output $\boldsymbol{y}$, has shape

$$N \times H - k_h - 1 \times W - k_w - 1 \times C_{out}$$



Convolution in 1D

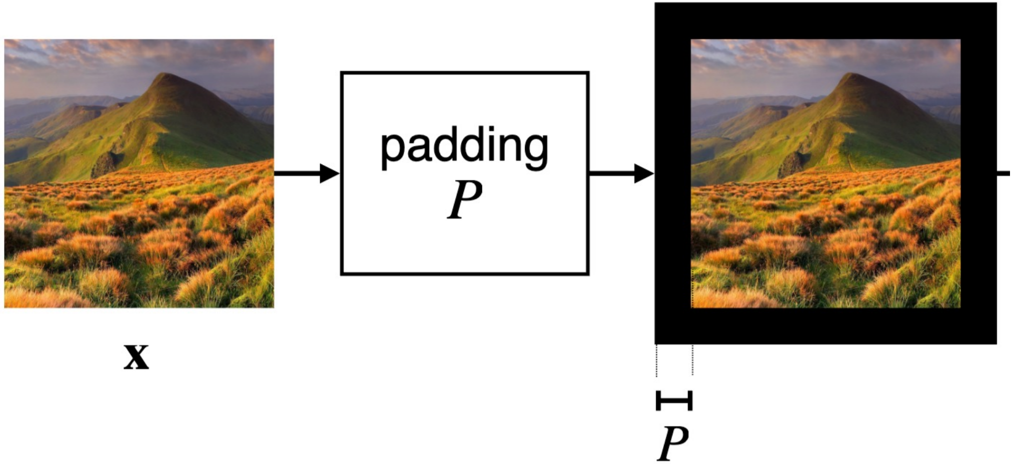
- The output size decreases progressively



Google DeepMind

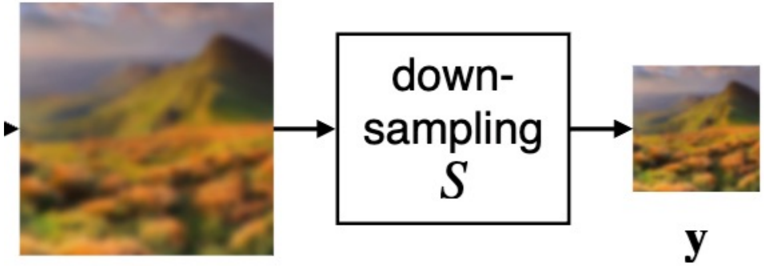
Padding

- Padding extends a tensor, $\boldsymbol{x}$, with a border filled with zeros.
- Typically used to retain the original input dimensions after each operation.



Striding

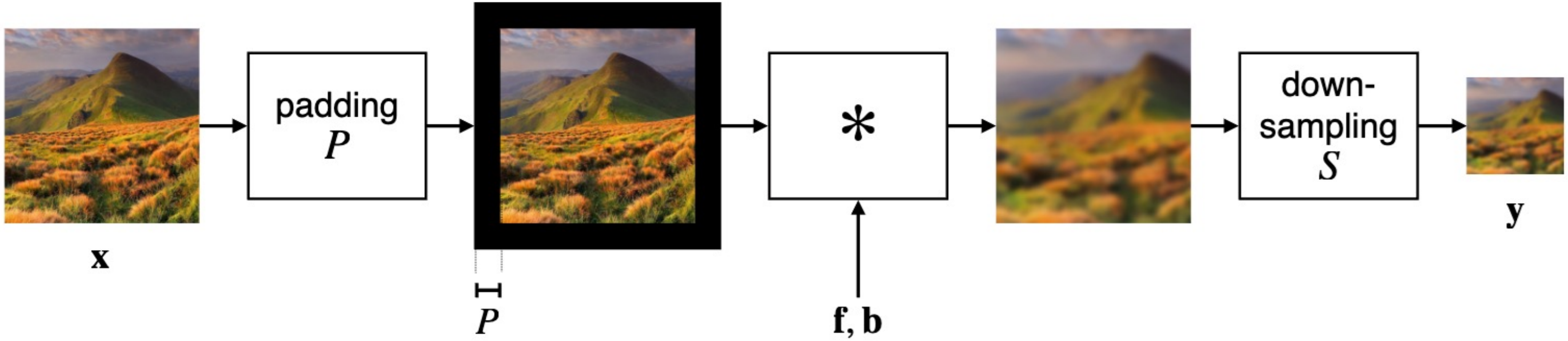
- Striding moves the filter S pixels at a time.
- It produces smaller output volumes. And increases the *receptive field* of subsequent operations.



Google DeepMind

Striding

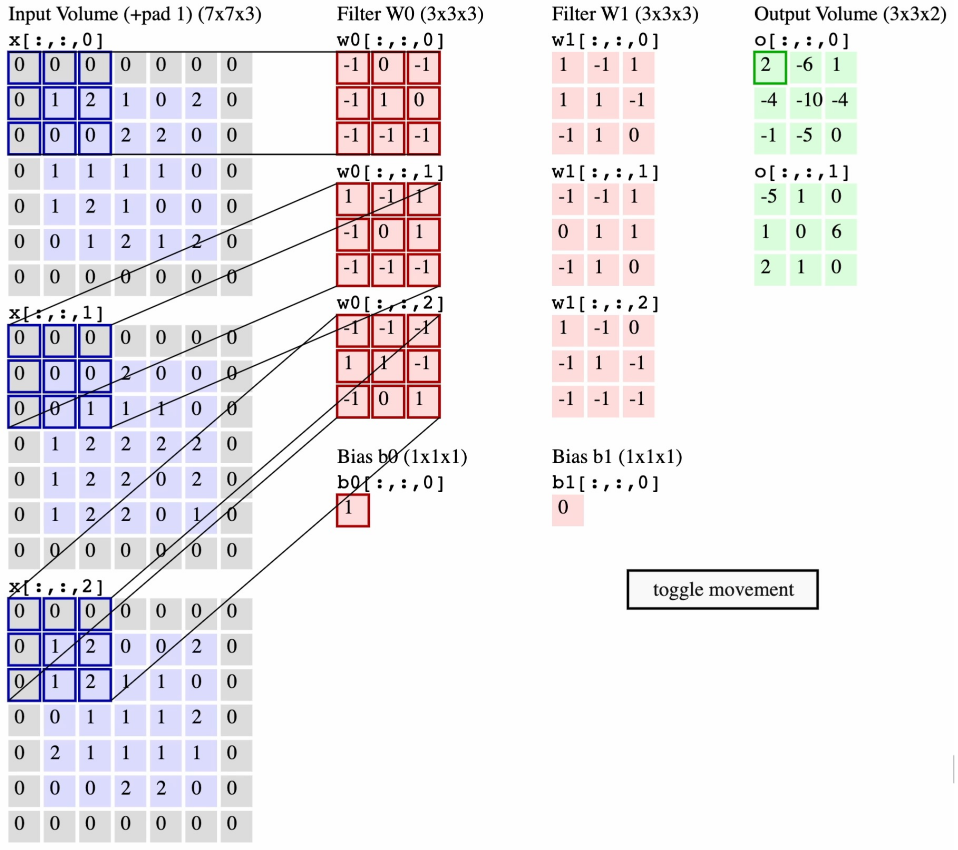
- We can also think of padding and striding as layers that we do before and after a standard convolution layer.



Google DeepMind

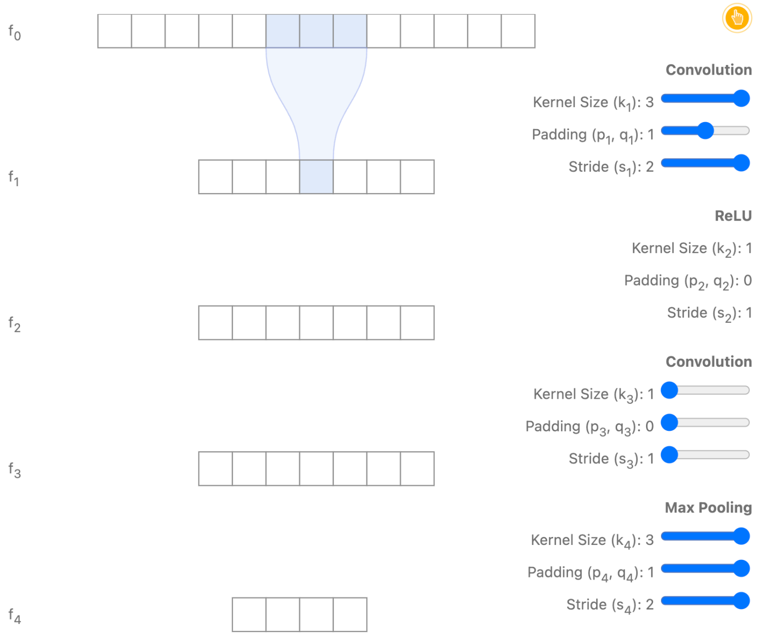
Convolution Example

- What is the padding?
- Filter dimensions?
- Input size?
- Output size?
- Demo here.



Receptive Field

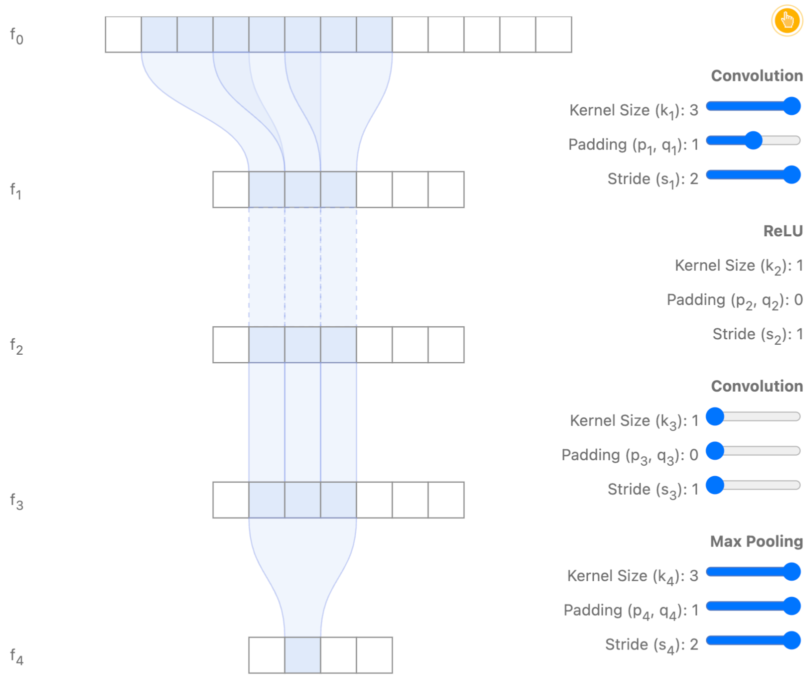
- How many input pixels are considered by a cell at a particular feature map of the network.
- Try out the demo here.



Google DeepMind

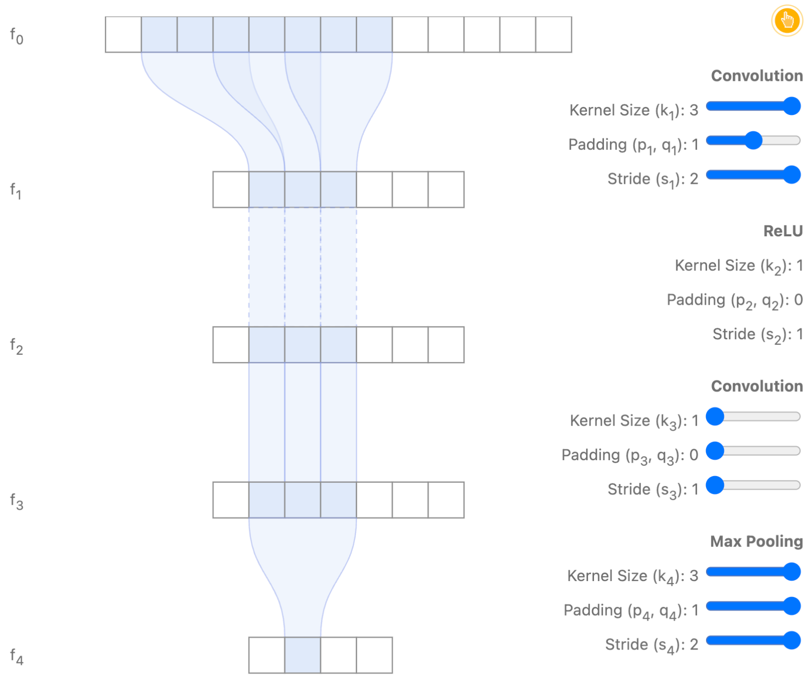
Receptive Field

- How many input pixels are considered by a cell at a particular feature map of the network.
- Try out the demo here.



Receptive Field

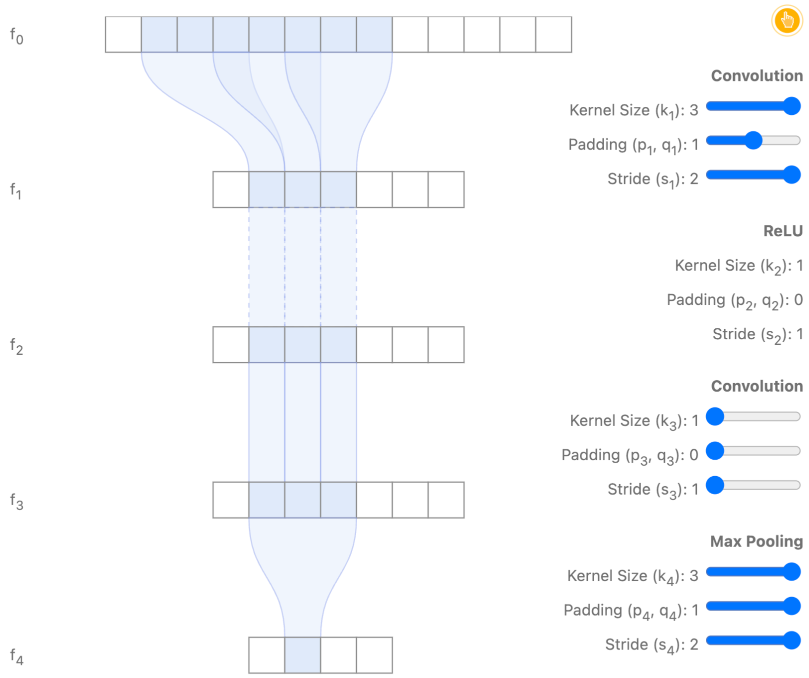
- How many input pixels are considered by a cell at a particular feature map of the network.
- Try out the demo [here](#).
- What do we want to be at the end of the network?



Google DeepMind

Receptive Field

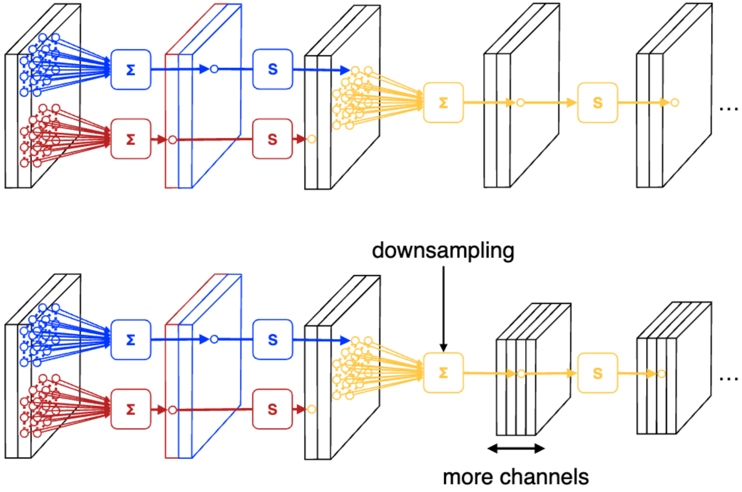
- How many input pixels are considered by a cell at a particular feature map of the network.
- Try out the demo here.
- What do we want to be at the end of the network?
    - For classification, should be the entire input.



Google DeepMind

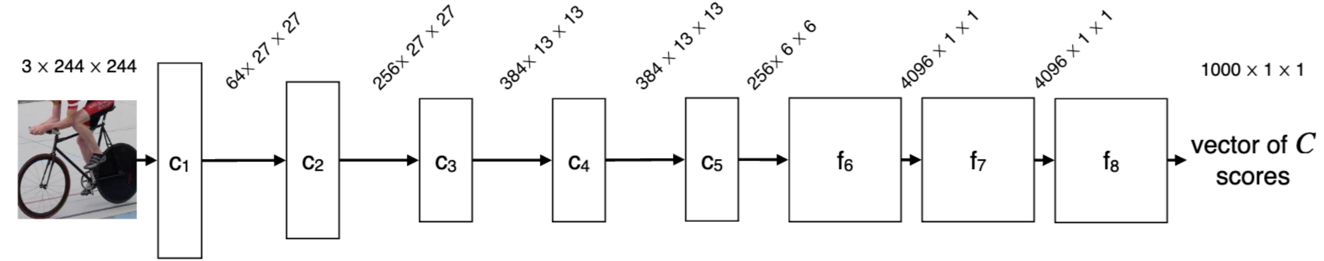
Deep Convolutional Neural Network

- A long sequence of layers!
- Typically alternate convolution, non-linear activation (ie ReLU).
- Perform pooling and/or striding to increase the receptive field, and decrease resolution.
- Usually decrease spatial dimensions, increase channel dimensions through the network



AlexNet

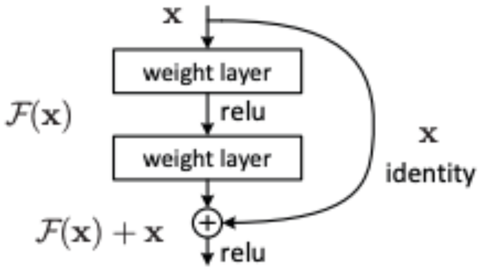
- Started the Deep Learning revolution in Computer Vision by winning the ImageNet challenge in 2012
  - The Top-5 error was 16%, compared to the runner-up with 26% error.



	c_1	c_2	c_3	c_4	c_5	f_6	f_7	f_8
K filters number	96	256	384	384	256	4096	4096	4096
F filter size	11	5	3	3	3	6	1	1
S filter stride	4	1	1	1	1	1	1	1
P filter padding	0	2	1	1	1	0	0	0
F' pooling size	3	3	-	-	3	-	-	-
S' pooling stride	2	2	-	-	2	-	-	-
P' pooling padding	1	0	-	-	0	-	-	-

Residual Networks

- Standard deep networks can become difficult to optimize when they become deep.
- Residual connections enable training very deep networks (even 1000 layers) in a stable manner.
- Intuition: Adding additional layers with identical connections to an existing network should not degrade performance; the weight layers can be 0 and the original function is maintained.



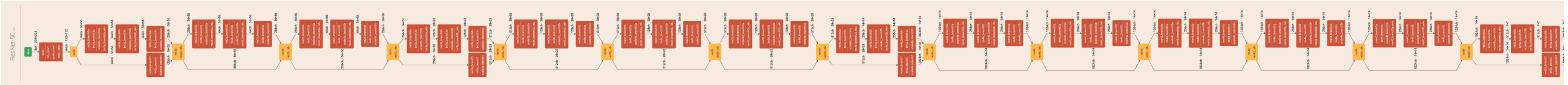
Google DeepMind

Residual Networks

- Principle of residual connections has been employed in subsequent architectures (both for more advanced CNNs, and other architectures like transformers).

AlexNet

ResNet-50



Questions?

Or just take a break ...

Transformers

Outline

- Deep dive of transformers and self-attention
- Transformers in Computer Vision.

Context

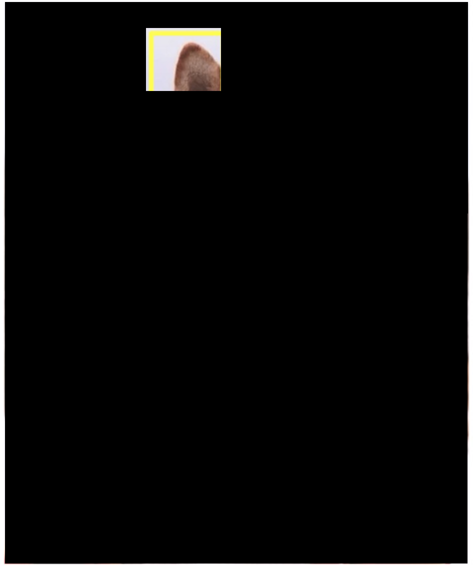


Image credit

Context

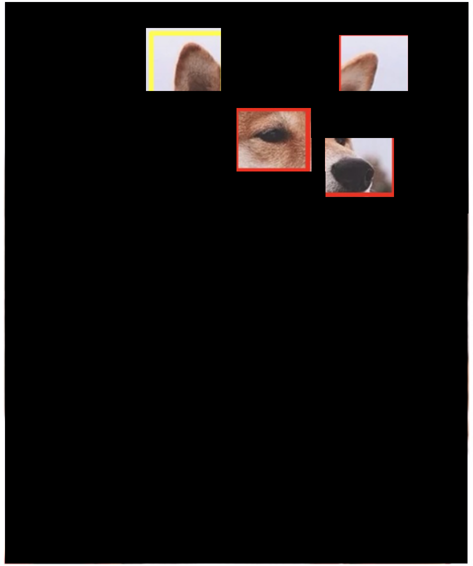


Image credit

Google DeepMind

Context

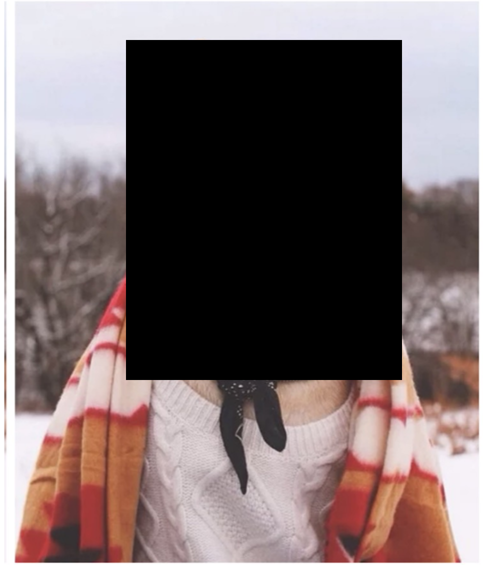


Image credit

Google DeepMind

Context

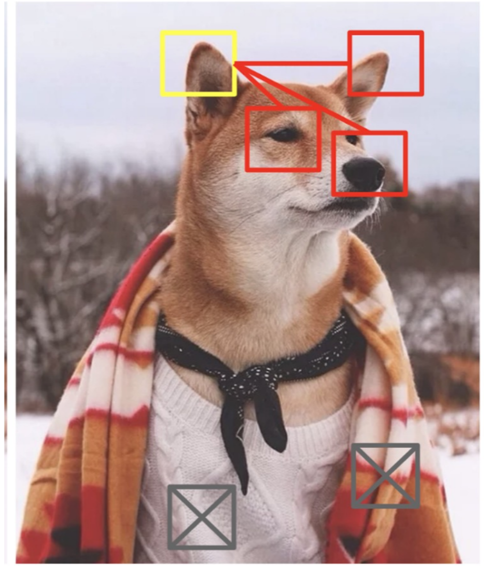
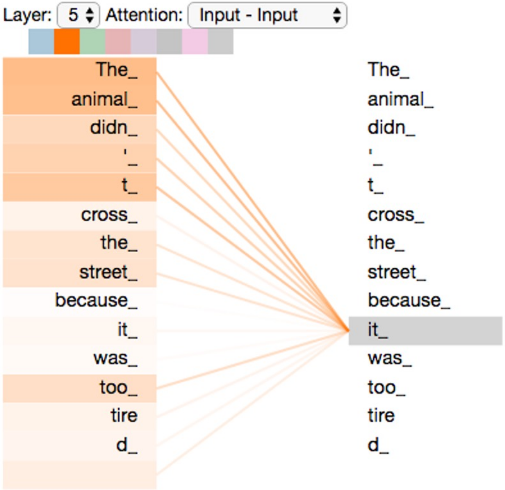


Image credit

Google DeepMind

Context

- “The animal didn't cross the street because it was too tired”
- What is “it”?

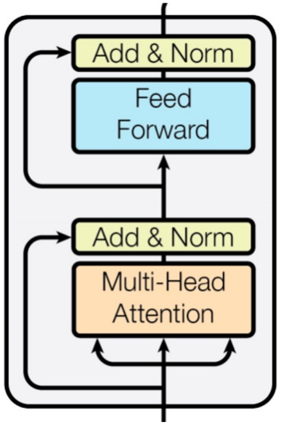


Try more examples [here]

Google DeepMind

Attention and Transformers

- Attention is a method of gathering relevant contextual information
- The transformer is a neural network layer that relies on attention
- In fact, state-of-the-art models across various domains consist almost entirely of transformer layers.



What is Attention?

High-Level Overview

- Use machine translation as initial example, as this is what Transformers were initially developed for

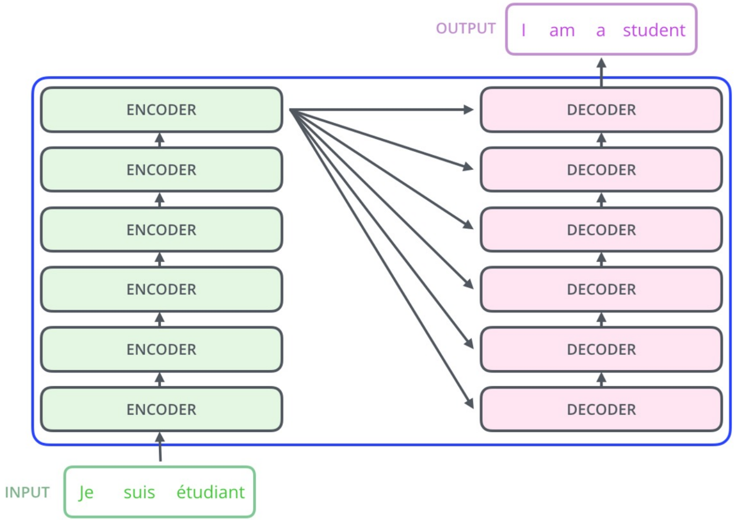
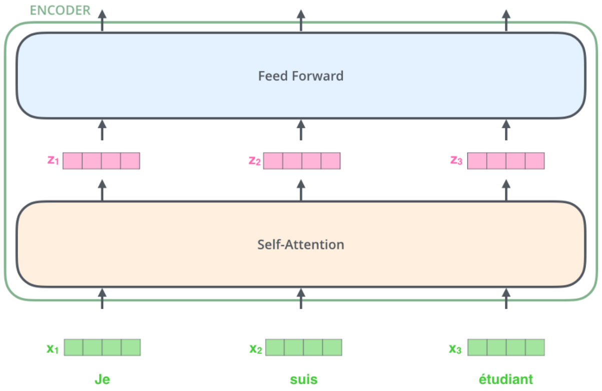


Figure credit for this, and next few slides

High-Level Overview

- Embed input into tokens (fixed dimensional vector)

- Process with encoder layer



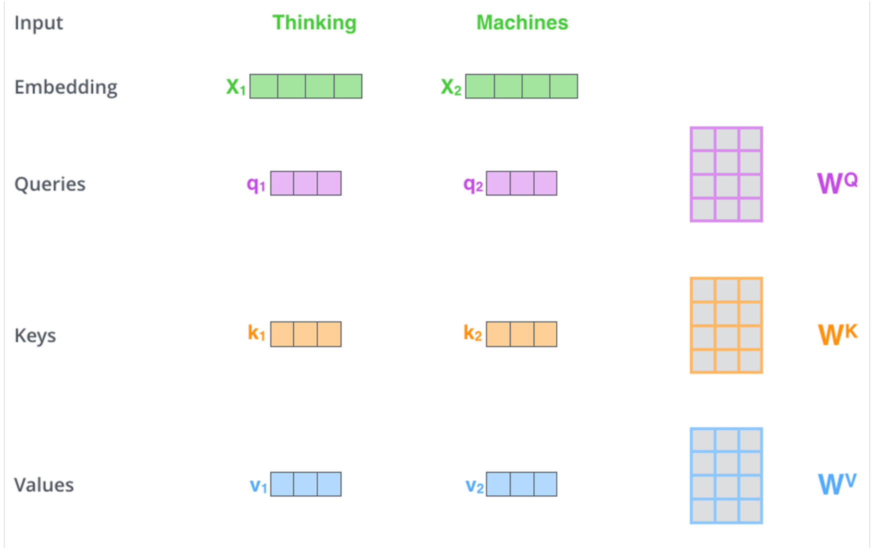
Google DeepMind

Self-Attention in Detail

- Given an input sequence, X.
- Project to Query, Key, Values using linear transforms.
- Head output = $\mathrm{Softmax}(QK^T)V$

Self-Attention in Detail

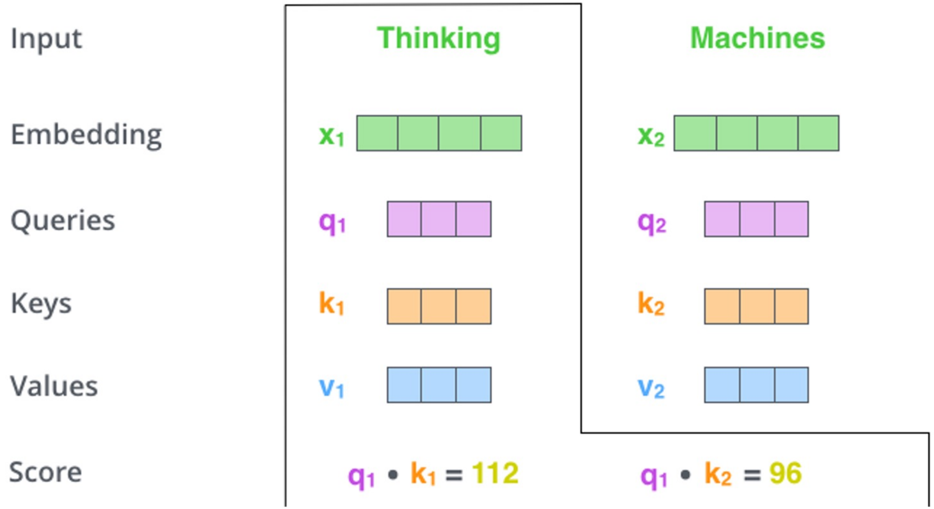
- Given an input sequence, X
- Project to Queries, Keys and Values using linear transforms.
    - $Q = W^q X,\ K = W^k X,\ V = W^V X$



Google DeepMind

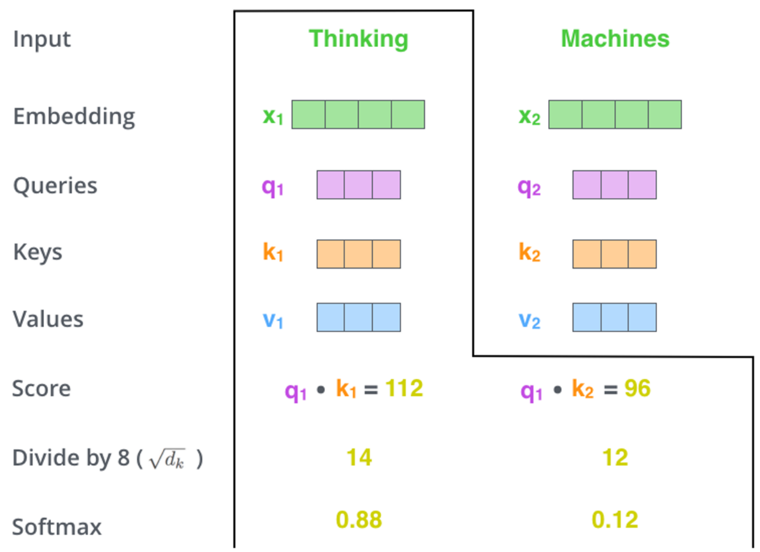
Self-Attention in Detail

- Given an input sequence, X
- Project to Queries, Keys and Values using linear transforms.
    - $\mathrm{Softmax}(QK^T)$



Self-Attention in Detail

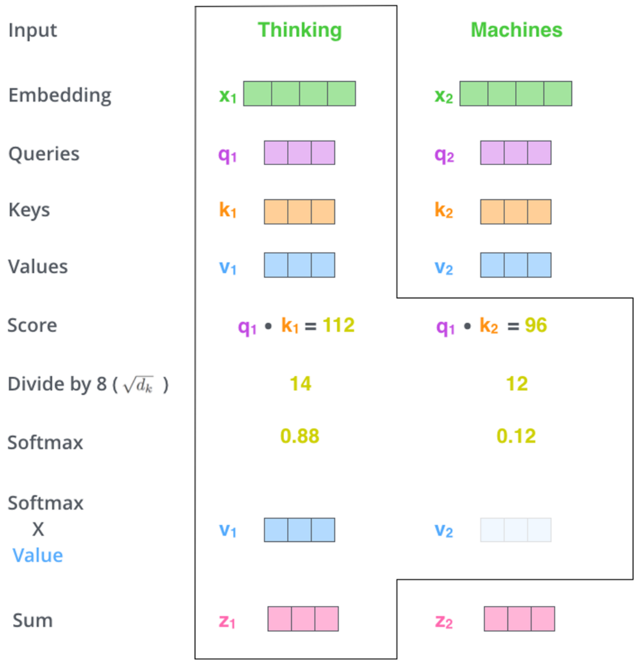
- Project to Queries, Keys and Values using linear transforms.
- Calculate a score: For each query, how relevant are all the other words?



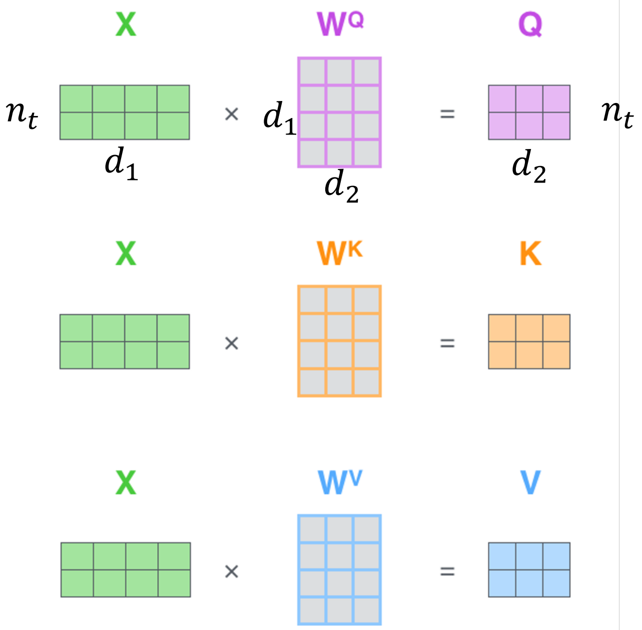
Google DeepMind

Self-Attention in Detail

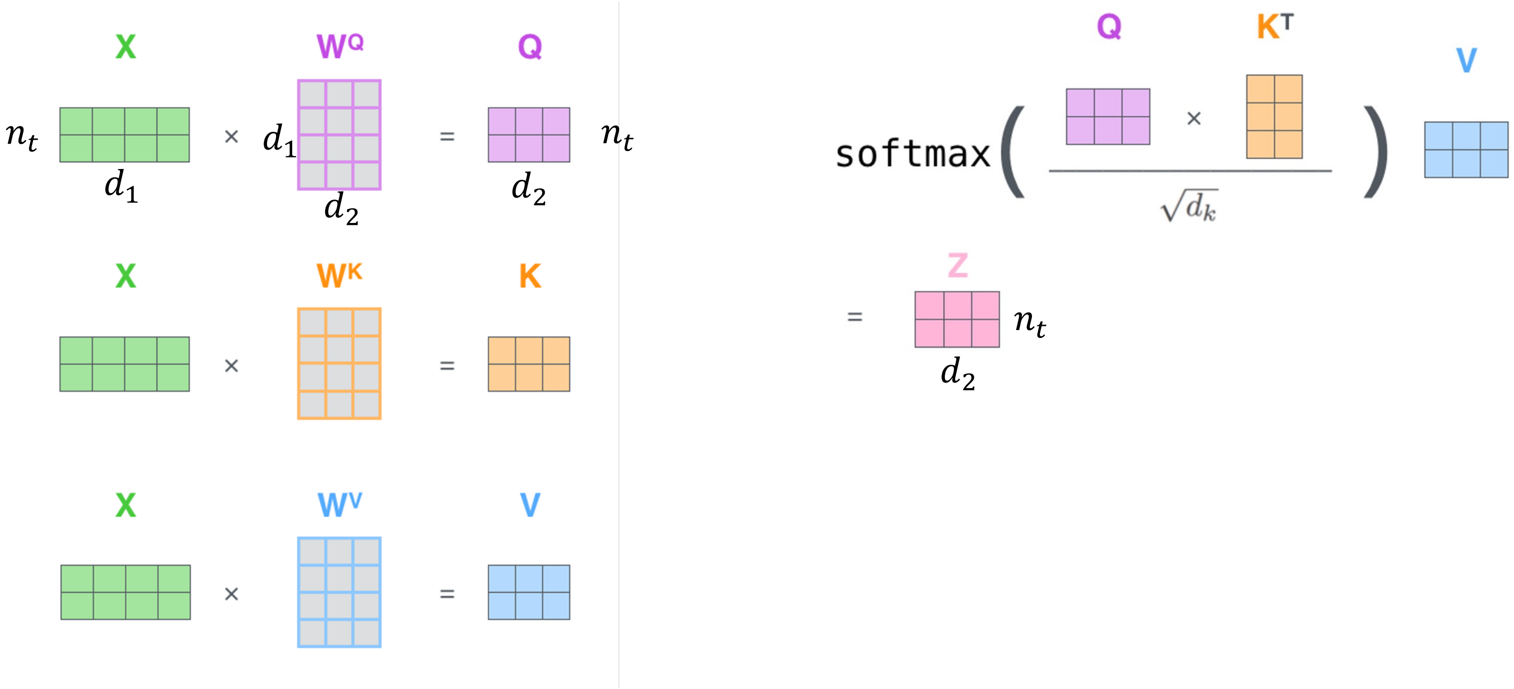
- Project to Queries, Keys and Values using linear transforms.
- Calculate a score
- Representation of each query token is attention-weighted sum of values.
  - $Z = \text{Softmax}(QK^T)V$



Self-Attention in Detail: As a Matrix



Self-Attention in Detail: As a Matrix



Self-Attention in Detail: Multiple Heads

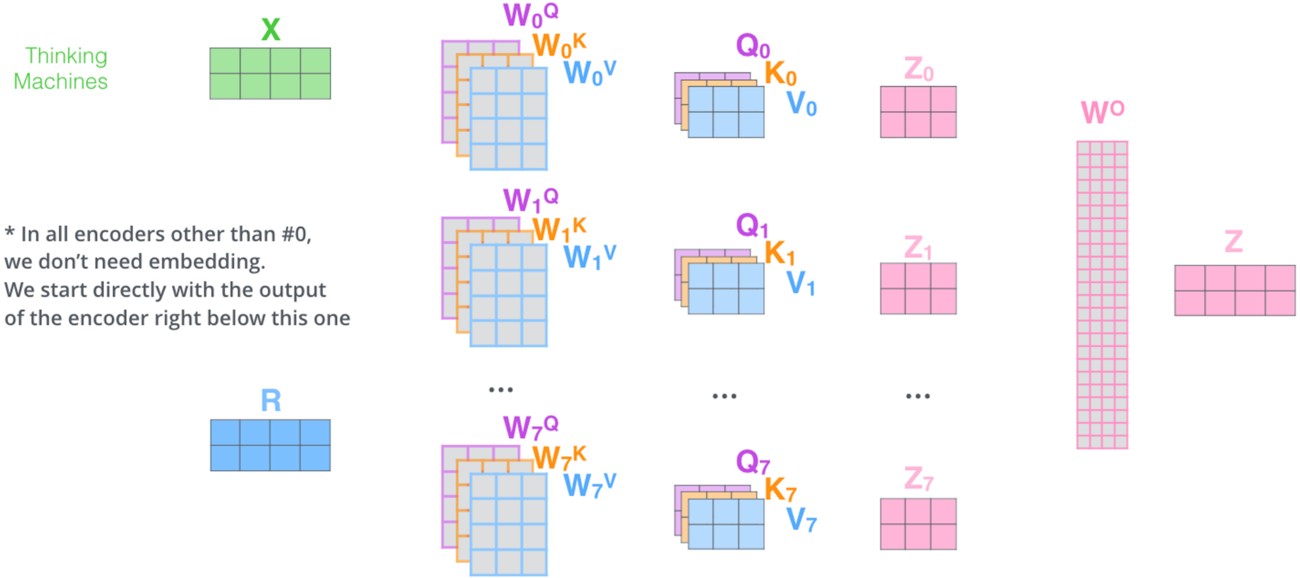
1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

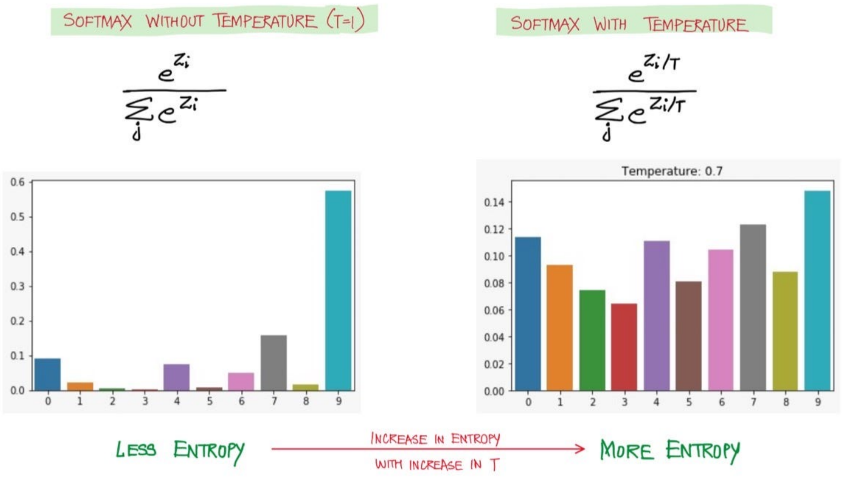
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

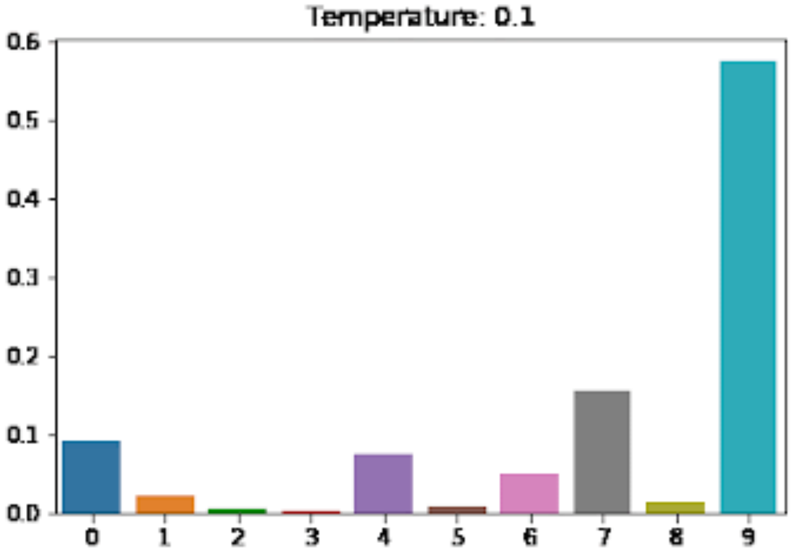
Self-Attention in Detail

- Where did the $\sqrt{d_k}$ come from
- Temperature of the softmax to control its “peakiness”



Self-Attention in Detail

- Where did the $\sqrt{d_k}$ come from
- Temperature of the softmax to control its "peakiness"



Google DeepMind

Positional Embeddings

- Self-attention is permutation invariant!
  - Say input is [x1, x2, x3]. And output is y1
  - If input is [x1, x3, x2]. Output is ...

Positional Embeddings

- Self-attention is permutation invariant!
    - Say input is [x1, x2, x3]. And output is y1
    - If input is [x1, x3, x2]. Output is y1

Positional Embeddings

- Self-attention is permutation invariant!
  - Say input is [x1, x2, x3]. And output is y1
  - If input is [x1, x3, x2]. Output is y1
- But what if the ordering of the input vectors conveys information as well?
  - The position of a word in a sentence matters!
  - ""The man ate a fish" != "The fish ate a man""

Positional Embeddings

- Self-attention is permutation invariant!
- Learned positional embedding
    - At the input, add a learned vector to each token
    - Representation of the token changes depending on its input position

Google DeepMind

Positional Embeddings

- Self-attention is permutation invariant!
- Sinusoidal positional embedding

$$\mathrm{PE}(i, \delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

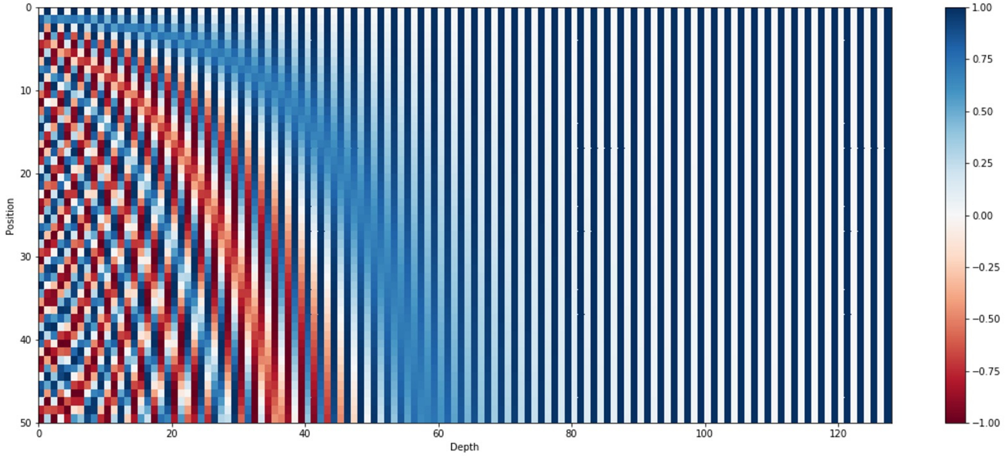


Figure credit

Putting it all together

- Original Transformer of [Vaswani et al. Attention is all You Need]

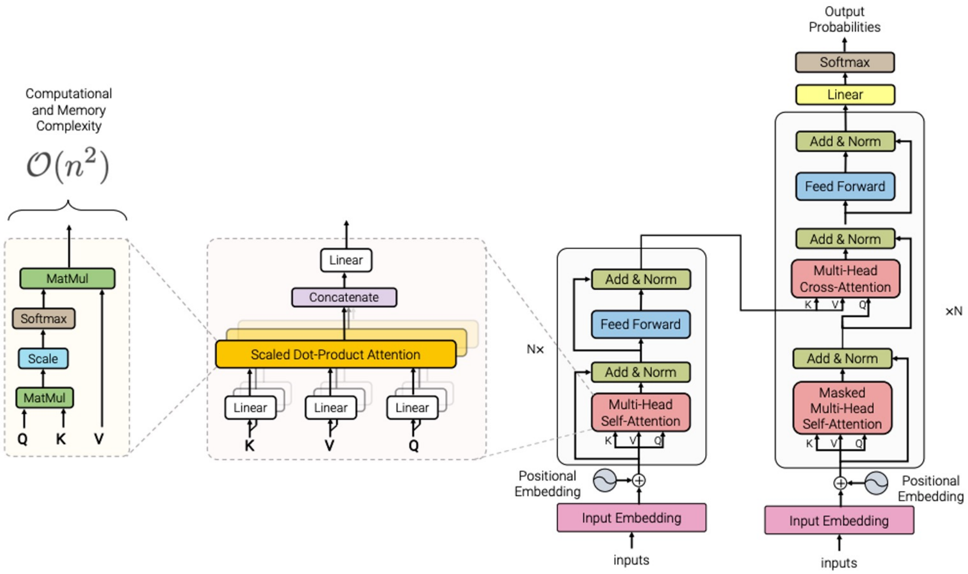


Figure 1: Architecture of the standard Transformer (Vaswani et al., 2017)

Google DeepMind

Advantages of Transformers

- Great for modelling context
    - Each token can have access to all other tokens in the sequence
    - *What is the receptive field of a transformer? Compared to a CNN?*
- A generic architecture:
    - Operates on any inputs that can be tokenized!
- Parallelizable
- Empirically shown to perform excellently at scale

Transformers at Scale

- Keep performing better with deeper models and more data
- [Scaling Laws for Neural Language Models]()

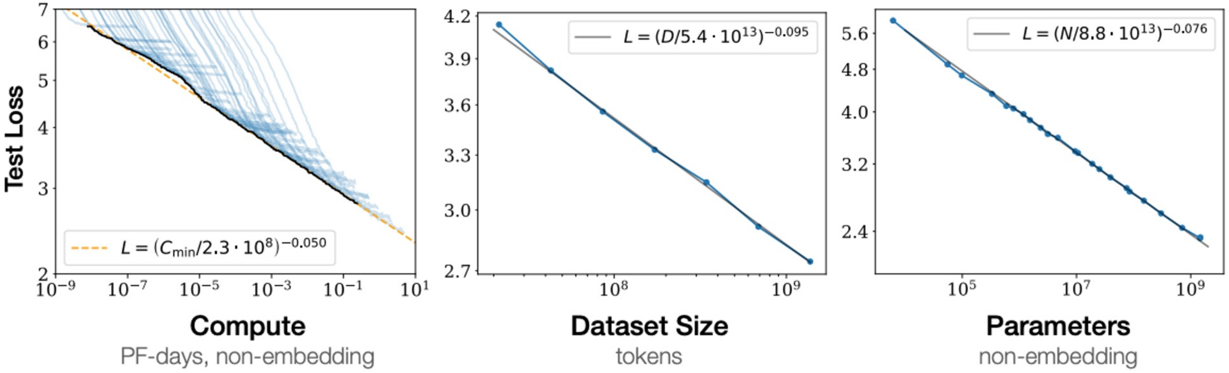
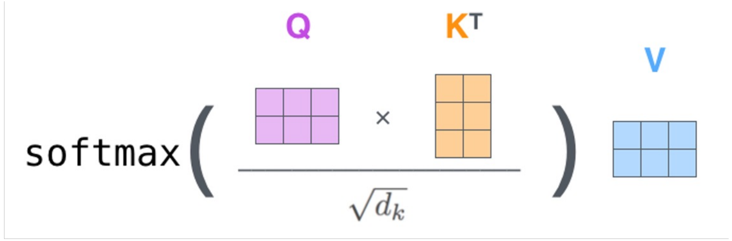


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Google DeepMind

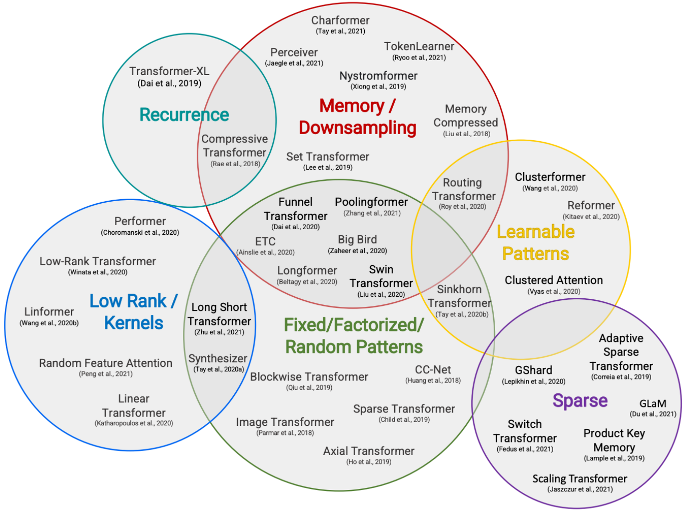
Weaknesses of Transformers



- Quadratic complexity
  - Each token attends to every other token
  - N tokens $\longrightarrow N^2$ operations
  - Prohibitive as the number of tokens increases!
- Most powerful language models are extremely expensive
- Large body of work on more efficient transformers.
  - Good survey paper
- Transformers can overfit easily on smaller datasets

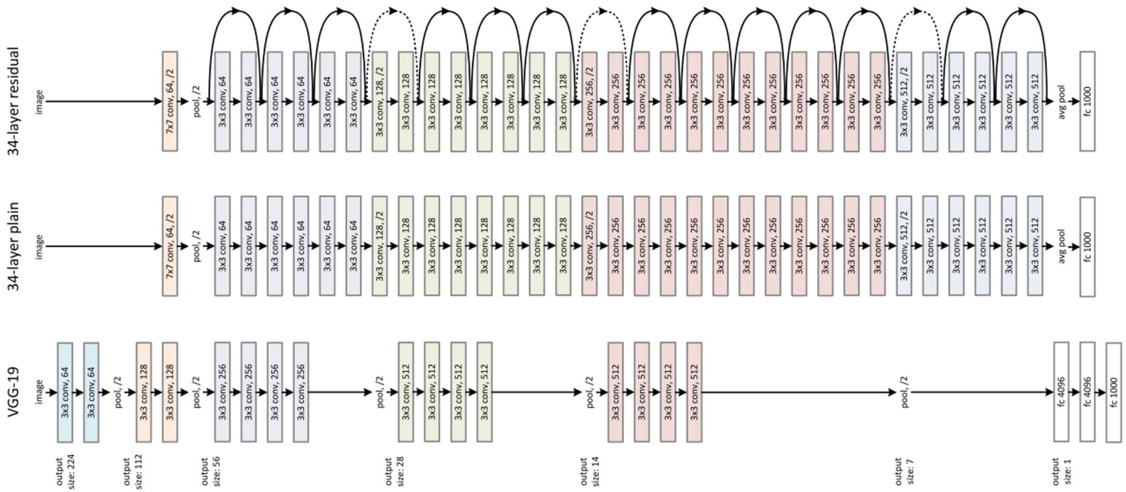
Weaknesses of Transformers

- Quadratic complexity
  - Each token attends to every other token
  - N tokens $\longrightarrow N^2$ operations
  - Prohibitive as the number of tokens increases!
- Most powerful language models are extremely expensive
- Large body of work on more efficient transformers.
  - [Good survey paper]
- Transformers can overfit easily on smaller datasets



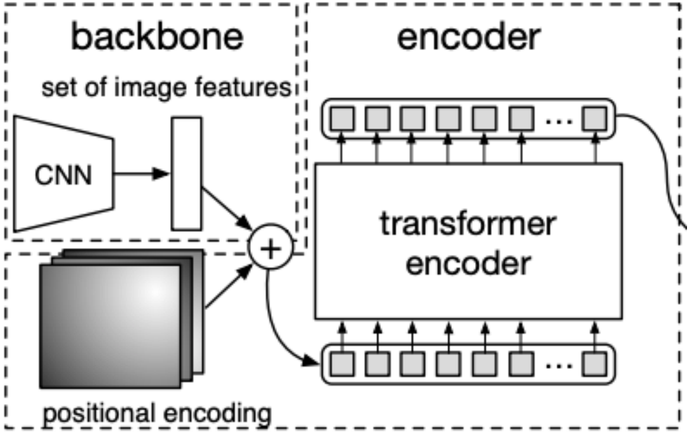
Transformers and Computer Vision

- CNNs used to be the architecture of choice in Vision
- Transformers were quite established as the architecture of choice in NLP



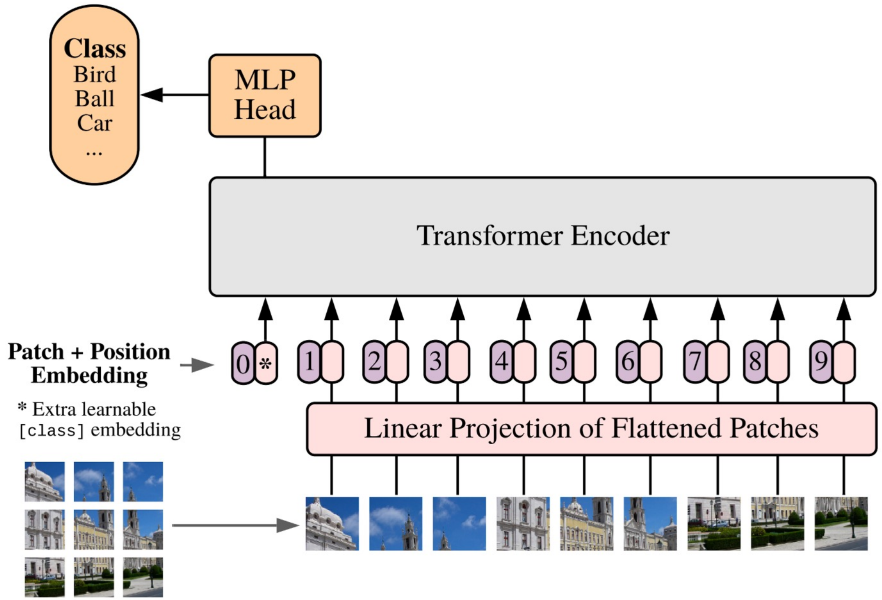
Transformers and Computer Vision

- CNNs used to be the architecture of choice in Vision
- Transformers were quite established as the architecture of choice in NLP
- Numerous attempts to incorporate self-attention into CNNs:
  - Wang CVPR 2018, Bello ICCV 2019, Huang ICCV 2019, Carion ECCV 2020
- Or to replace convolutions entirely with self-attention
  - Parmar ICML 2018, Ramachandran NeurIPS 2019



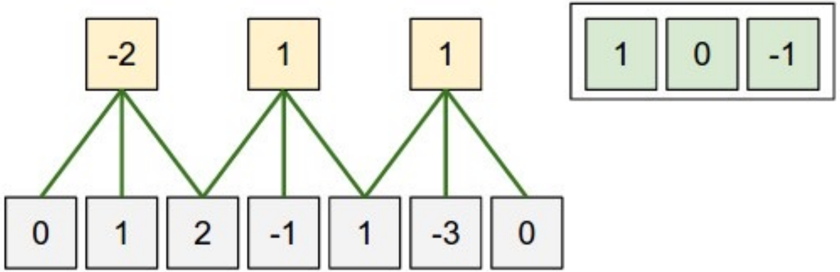
Vision Transformers

- [An Image is Worth 16x16 Words]
- "Tokenize" an image by splitting it into patches. Pass tokens through a transformer.



Vision Transformers

- [An Image is Worth 16x16 Words]
- ”Tokenize” an image by splitting it into patches. Pass tokens through a transformer.
- Same as convolution where the stride is the same as the filter size.



Google DeepMind

Vision Transformer Models

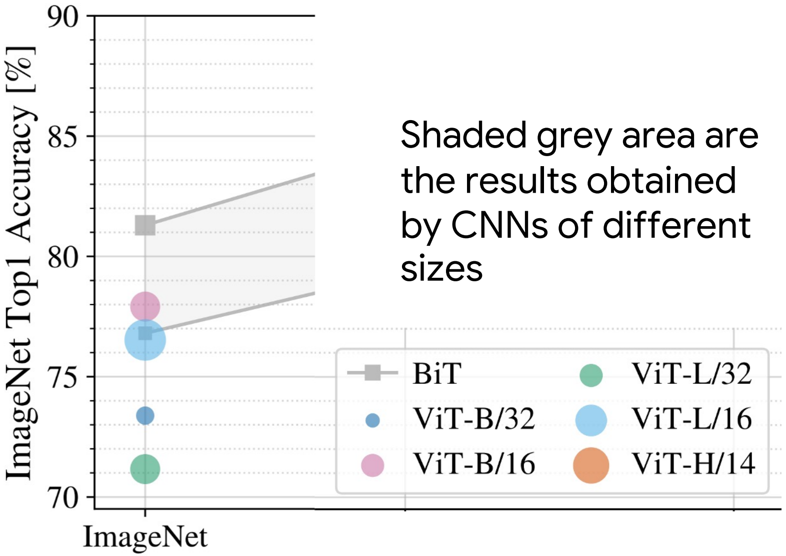
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Notation e.g. ViT-L/16

Google DeepMind

An Obvious Idea?

- Might appear obvious to replace CNN entirely with a transformer.
- Does not really work well at standard, ImageNet setting
- Larger models, ie ViT-L/16 are actually worse than smaller ones, ie ViT-B/16.

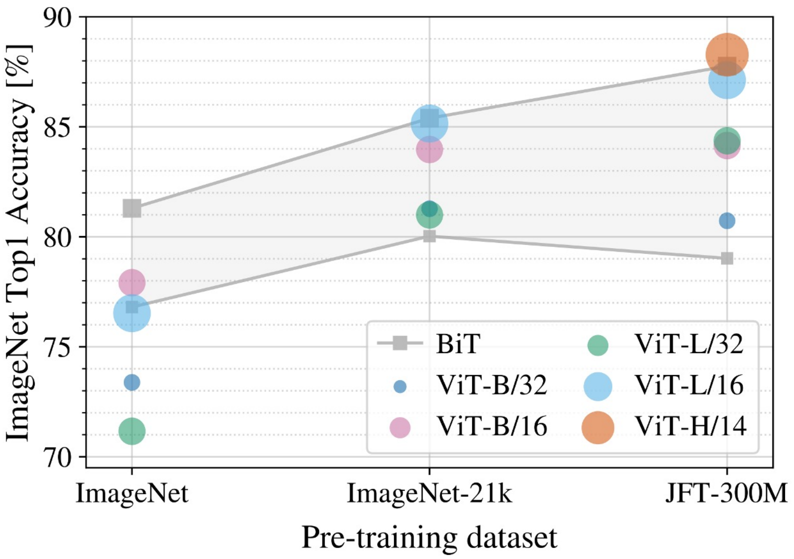


Shaded grey area are the results obtained by CNNs of different sizes

Google DeepMind

Vision Transformers are effective at scale

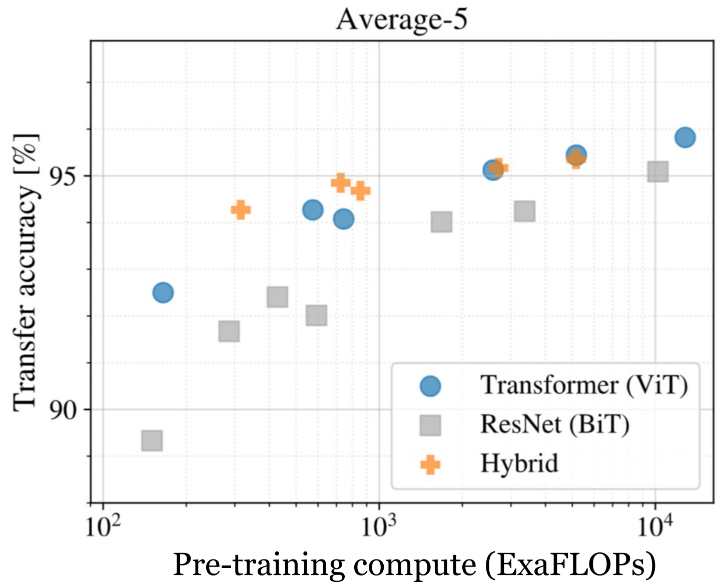
- Transformers have less inductive biases then Convolutional Networks (ie translational equivariance)
- So they need more data to train



Google DeepMind

Vision Transformers are effective at scale

- Transformers, are however, able to take advantage of large-scale data better than CNNs can
- And are more compute-efficient too in terms of computation to reach accuracy.



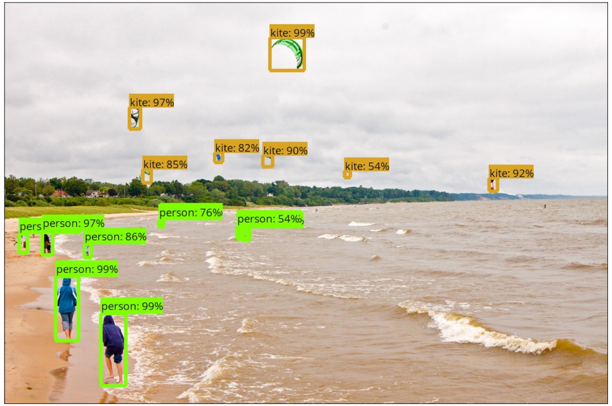
Google DeepMind

High Resolution Data

- Vision Transformers still have issue with quadratic complexity with respect to the number of tokens.
- Becomes a problem when we have high resolution images.
- Need to process images at high resolution for tasks like object detection and semantic segmentation.

CAT

Image classification. Process at 224 x 224. ? tokens!



Object detection. Process at 1024 x 1024. ? tokens!

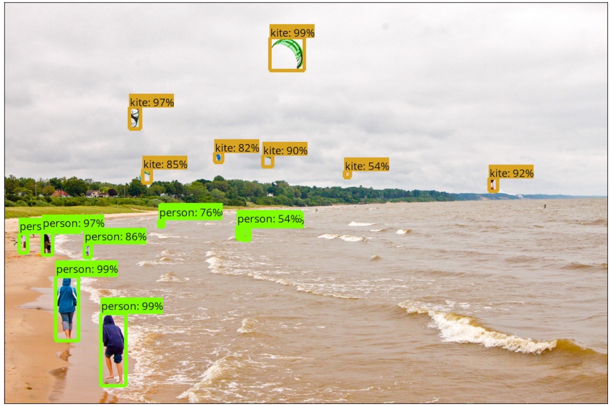
Google DeepMind

High Resolution Data

- Vision Transformers still have issue with quadratic complexity with respect to the number of tokens.
- Becomes a problem when we have high resolution images.
- Need to process images at high resolution for tasks like object detection and semantic segmentation.

CAT

Image classification. Process at 224 x 224. 196 tokens!



Object detection. Process at 1024 x 1024. 4096 tokens!

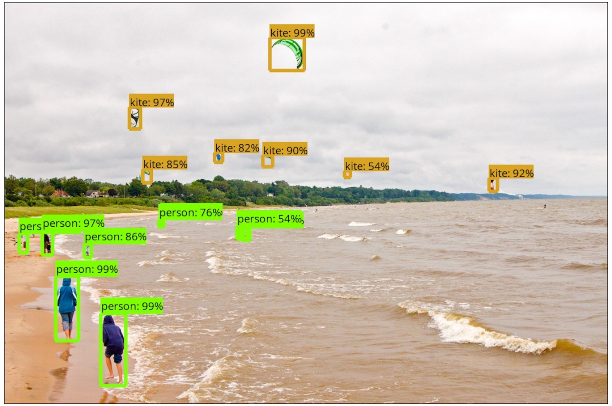
Google DeepMind

High Resolution Data

- Vision Transformers still have issue with quadratic complexity with respect to the number of tokens.
- Becomes a problem when we have high resolution images.
- Need to process images at high resolution for tasks like object detection and semantic segmentation.

CAT

Image classification. Process at 224 x 224. 196 tokens! 20 GFLOPs

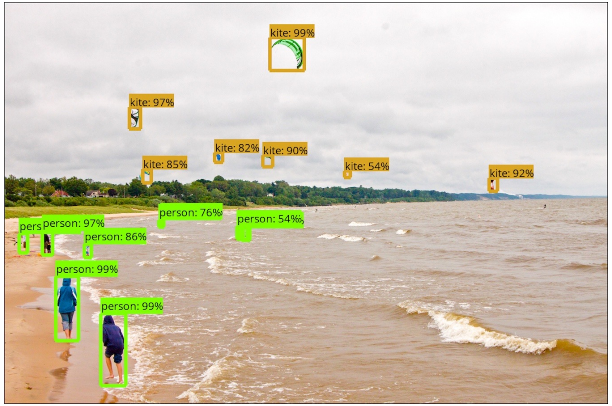


Object detection. Process at 1024 x 1024. 4096 tokens! ? GFLOPs

High Resolution Data

- Vision Transformers still have issue with quadratic complexity with respect to the number of tokens.
- Becomes a problem when we have high resolution images.
- Need to process images at high resolution for tasks like object detection and semantic segmentation.

CAT
Image classification. Process at 224 x 224. 196 tokens! 20 GFLOPs

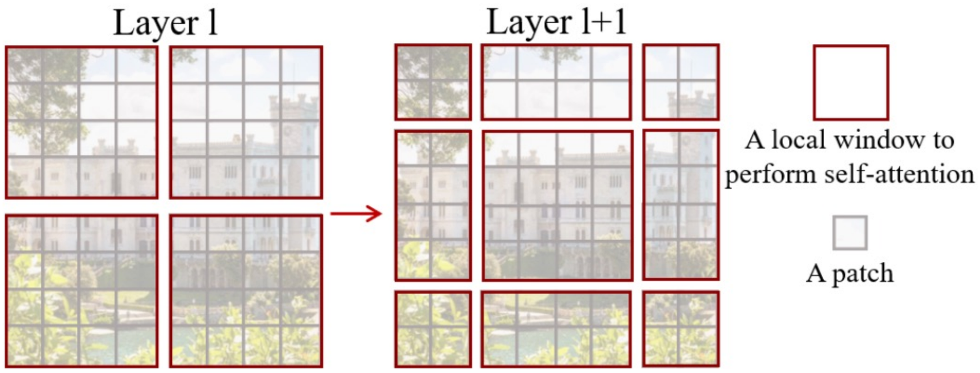


Object detection. Process at 1024 x 1024. 4096 tokens! 8734 GFLOPs

Google DeepMind

Swin Transformer

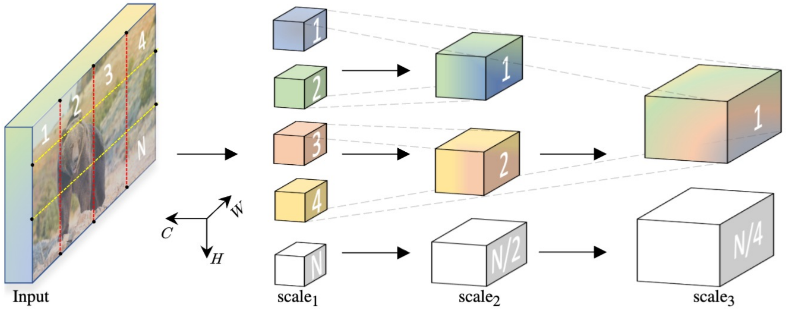
- Process alternating sliding windows of tokens at a time.
- Substantially reduces the computation required for high resolution data.
- Makes a transformer more “CNN”-like.
- Excels at tasks like object detection and segmentation.



Google DeepMind

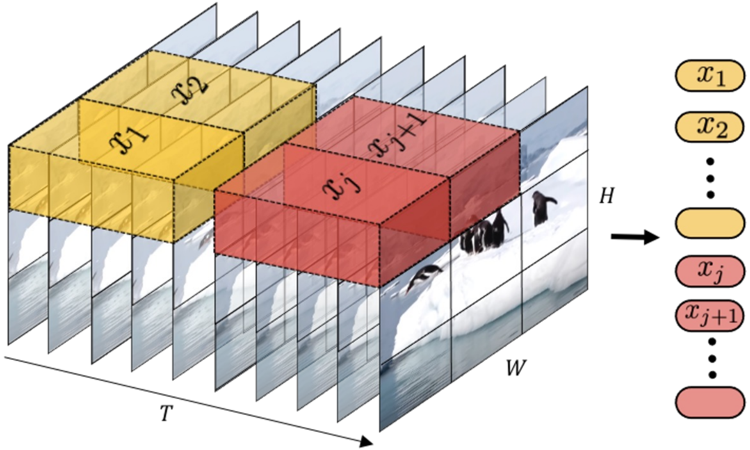
Hierarchical Pooling

- CNNs use pooling throughout the network to reduce spatial dimensions and increase the "receptive field"
- In transformers, we can use them to reduce the number of tokens and increase the efficiency of the model.
- Examples: [Multiscale ViT](), [Pyramid ViT](). Particularly suited for high-resolution tasks.



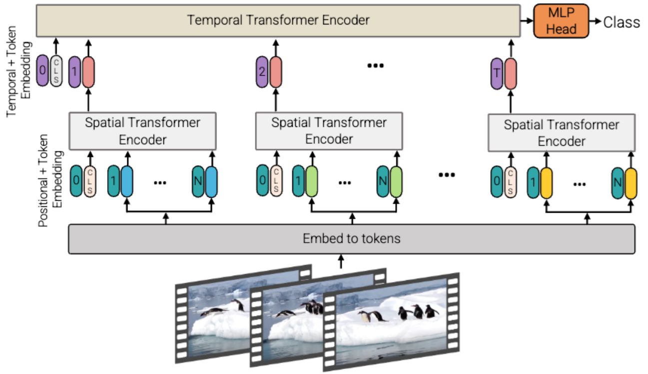
Video

- Video data is also computationally very demanding due to the additional “time” dimension.
- Can tokenize data in the same way as images, by extending “patches” to “tubelets” in time (ie ViViT).

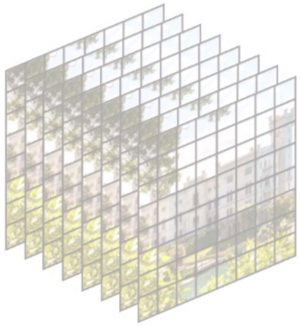


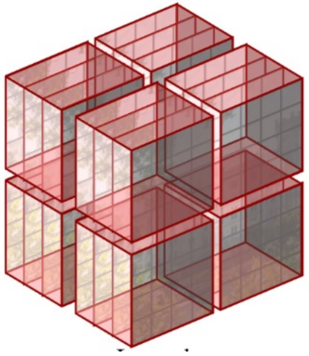
Video

- Manage complexity of the data by alternating attention between spatial and temporal axes in various manners (ViViT, Timesformer).
- Sliding windows in 3D (Video Swin)



Axial Attention
(T+W+H)





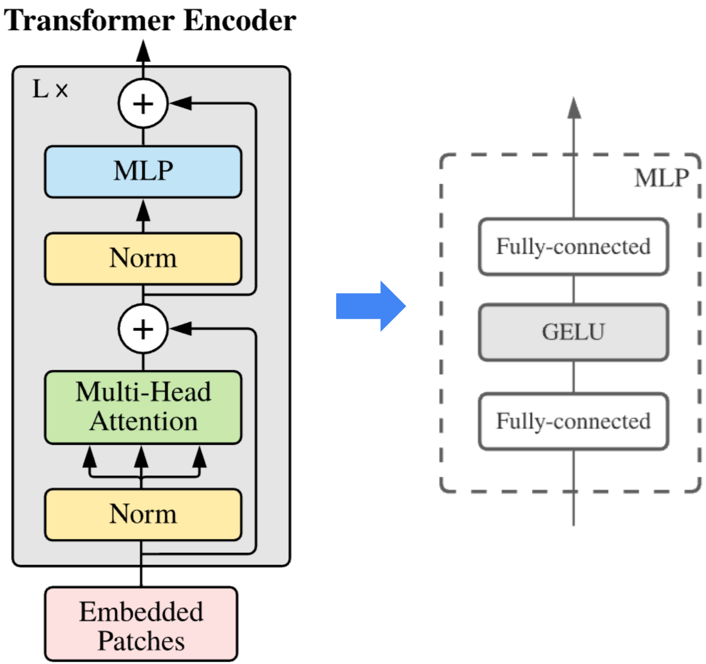
Google DeepMind

Are Transformers All You Need?

- Vision Transformers outperform CNNs at large scale (model size and dataset size).
- Vision transformers need more data as they have fewer inductive biases?
- Are there other architectures besides CNNs and transformers that we could be using?

MLP-Mixers

- Transformer encoder block has two main operations:
  - Multi-head attention
  - MLP / Feedforward
- What if we use only MLPs?
- MLP-Mixer: Alternate between MLPs on the ”channel” and “token” axes



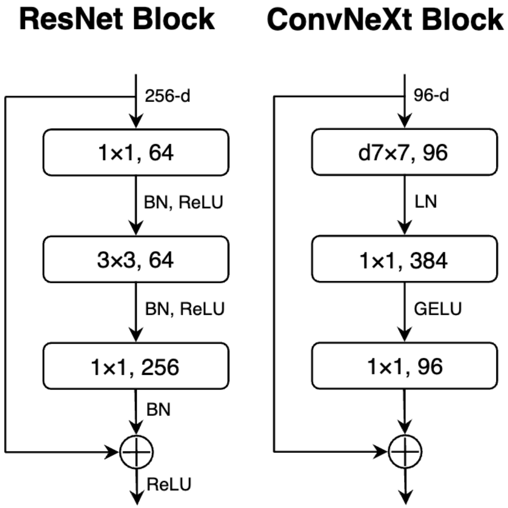
MLP-Mixers

- Competitive with Vision Transformers and CNNs.
- Again, only effective at larger scales

	ImNet top-1	ReaL top-1	Avg 5 top-1	VTAB-1k 19 tasks	Throughput img/sec/core	TPUv3 core-days
Pre-trained on ImageNet-21k (public)						
● HaloNet [51]	85.8	—	—	—	120	0.10k
● Mixer-L/16	84.15	87.86	93.91	74.95	105	0.41k
● ViT-L/16 [14]	85.30	88.62	94.39	72.72	32	0.18k
● BiT-R152x4 [22]	85.39	—	94.04	70.64	26	0.94k
Pre-trained on JFT-300M (proprietary)						
● NFNet-F4+ [7]	89.2	—	—	—	46	1.86k
● Mixer-H/14	87.94	90.18	95.71	75.33	40	1.01k
● BiT-R152x4 [22]	87.54	90.54	95.33	76.29	26	9.90k
● ViT-H/14 [14]	88.55	90.72	95.97	77.63	15	2.30k
Pre-trained on unlabelled or weakly labelled data (proprietary)						
● MPL [34]	90.0	91.12	—	—	—	20.48k
● ALIGN [21]	88.64	—	—	79.99	15	14.82k

ConvNeXt

- Redesign ConvNet using modern “tricks” from transformers
  - ReLU -> GeLU
  - Batch Norm -> Layer Norm
  - Fewer activation functions and normalization
  - Further regularization during training (ie label smoothing mixup, data augmentation)
- Performs on-par with vision transformers.
- Good at high-resolution tasks.



Google DeepMind

ConvNeXt

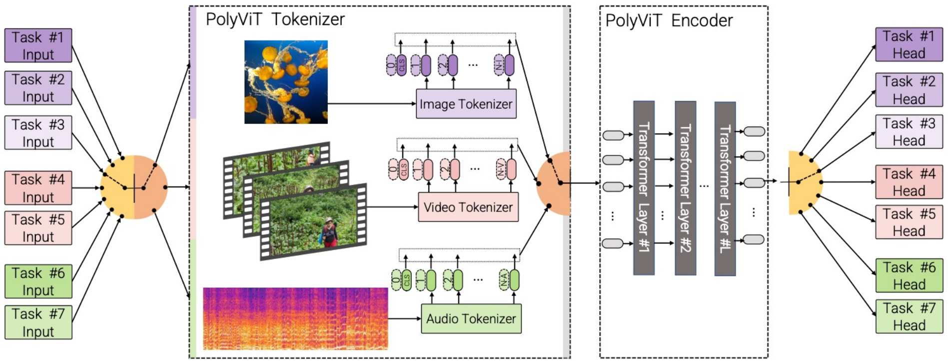
- Redesign ConvNet using modern "tricks" from transformers
    - ReLU -> GeLU
    - Batch Norm -> Layer Norm
    - Fewer activation functions and normalization
    - Further regularization during training (ie label smoothing mixup, data augmentation)
- Performs on-par with vision transformers.
- Good at high-resolution tasks.

Semantic segmentation on ADE20K

backbone	input crop.	mIoU	#param.	FLOPs
	ImageNet-1K pre-trained			
○ Swin-T	512^2	45.8	60M	945G
● ConvNeXt-T	512^2	**46.7**	60M	939G
○ Swin-S	512^2	49.5	81M	1038G
● ConvNeXt-S	512^2	**49.6**	82M	1027G
○ Swin-B	512^2	49.7	121M	1188G
● ConvNeXt-B	512^2	**49.9**	122M	1170G
	ImageNet-22K pre-trained			
○ Swin-B‡	640^2	51.7	121M	1841G
● ConvNeXt-B‡	640^2	**53.1**	122M	1828G
○ Swin-L‡	640^2	53.5	234M	2468G
● ConvNeXt-L‡	640^2	**53.7**	235M	2458G
● ConvNeXt-XL‡	640^2	**54.0**	391M	3335G

So Why Transformers?

- Can handle variable resolution inputs
  - Videos or images of different sizes.
  - Problematic for a Mixer since “token mixing” depends on the number of tokens.
- Architecture that works well for multiple modalities
  - Language, images, video, audio …
  - Any data that can be tokenized can be processed by a transformer.



Questions?

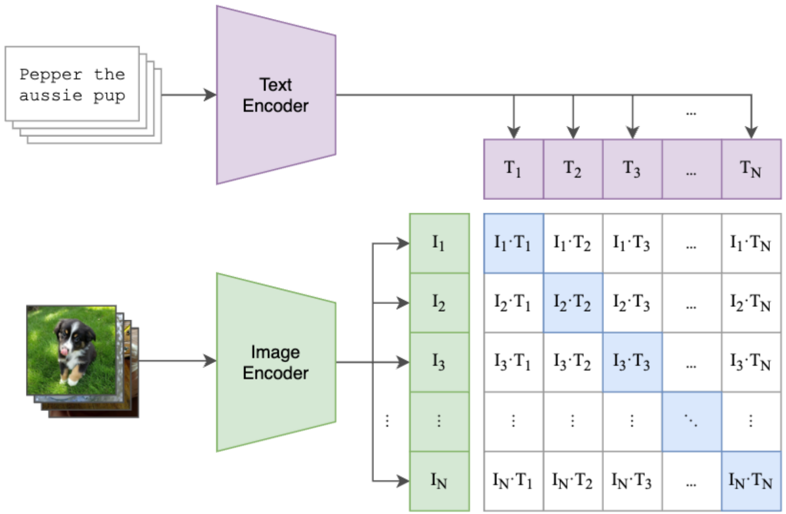
Or just take a break ...

Connecting Vision and Language

Google DeepMind

Discriminative Image-Text Modelling

- CLIP performs *contrastive pretraining* of image and text.
- Scrapes a dataset of image-text pairs from the web.
- Pretraining loss function encourages the model to match a feature representations from an image to the representation from text.



Google DeepMind

CLIP Architecture and Loss

- CLIP architecture is simply a vision encoder and a text encoder
    - Both are transformers in practice.
- Contrastive loss encourages a high score for the correct image-text pair. And a low score for all other examples in the batch
    - What do we need to be careful of here?

Image Text

$$\max_{f,g} \log \left(\frac{\exp(f(x)^\top g(y))}{\exp(f(x)^\top g(y)) + \sum_{(x',y') \in \mathcal{N}} \exp(f(x')^\top g(y'))} \right)$$

Positive Image-Text pair

Negative Image-Text pairs

Image encoder

Text encoder

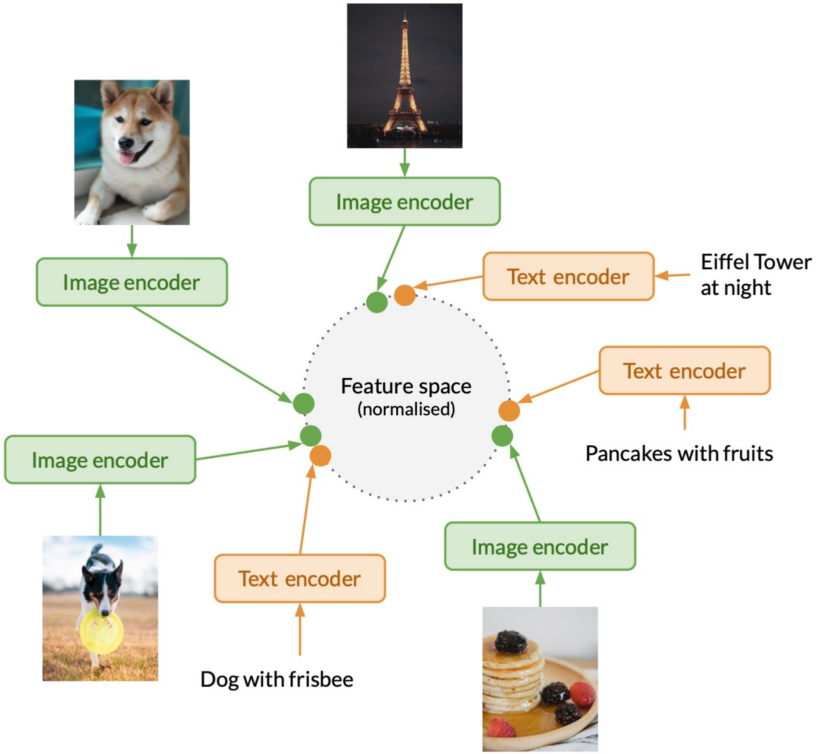
Google DeepMind

CLIP Architecture and Loss

- CLIP architecture is simply a vision encoder and a text encoder
    - Both are transformers in practice.
- Contrastive loss encourages a high score for the correct image-text pair. And a low score for all other examples in the batch
    - The objective can be too easy if there are no other “hard examples” in the batch.
    - Typically means that we need a large batch size during training in order to have a higher chance of having “hard examples”
    - It is also possible to construct batches in the data-loader to be “harder”. But this does not scale well to large-scale training.

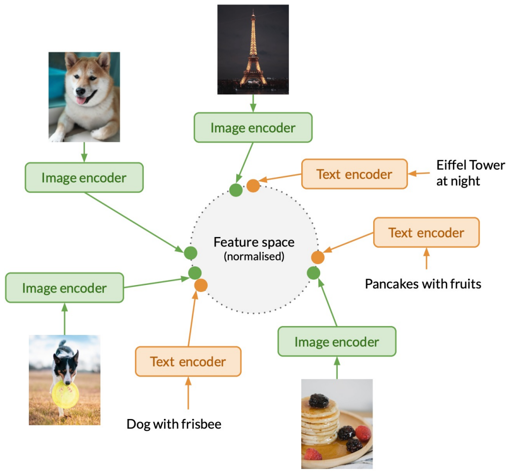
Joint Image-Text Embedding

- Images and text are embedded into a common latent space.
- This facilitates many different applications



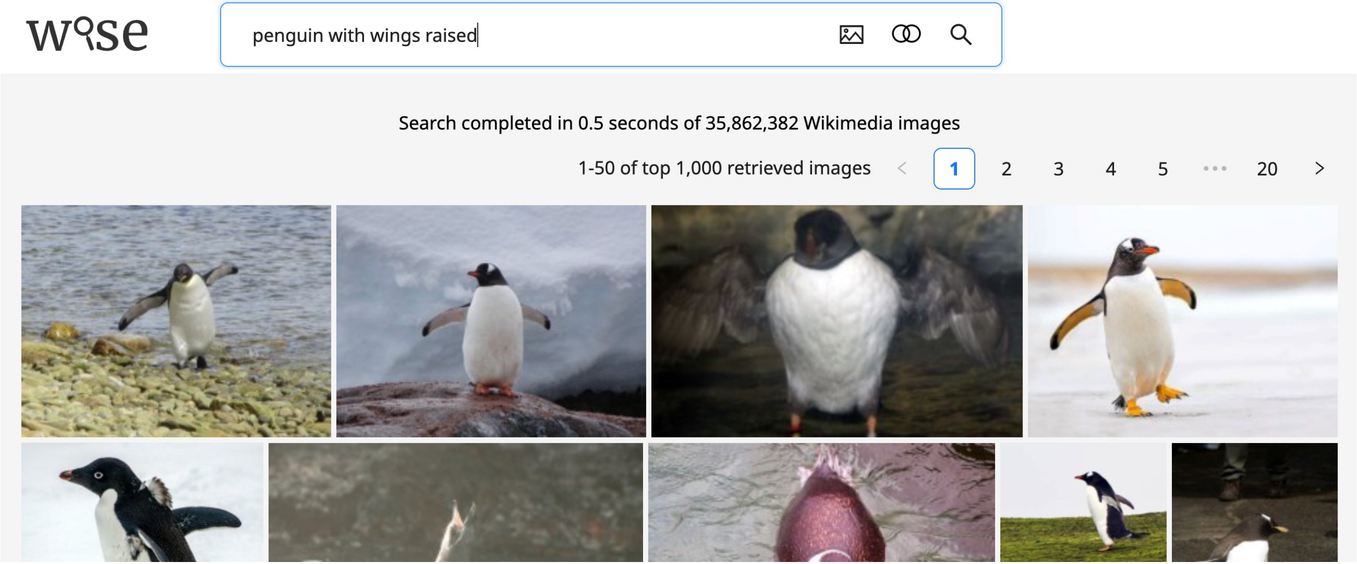
Retrieval: Image- and Video Search

- Image-Text pretraining enables text-to-image search
- We embed a text query, and l2-normalize it.
- We construct an "index" of images, by precomputed computing their l2-normalized embeddings.
- We then find the the image which best matches the text query
  - "Best matches" means the lowest distance / highest inner product
  - Why are the two equivalent?



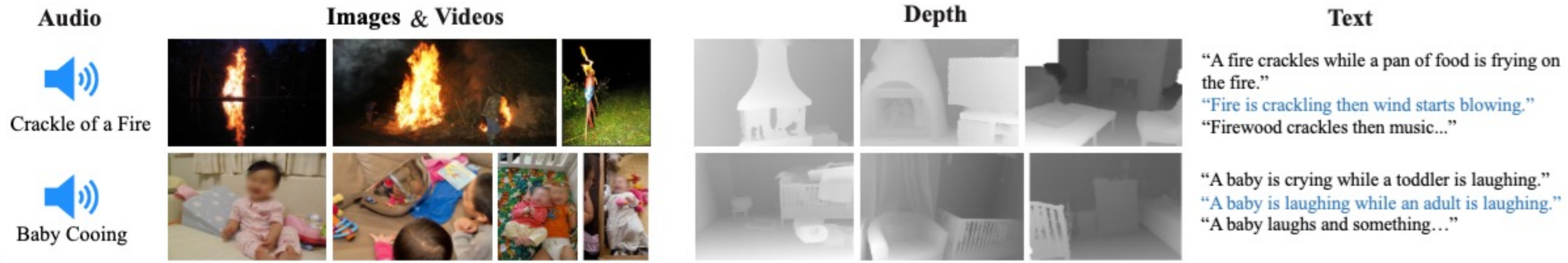
Text to Image Search

- Image-Text pretraining enables text-to-image search.
- Demo / code for VGG Wise.



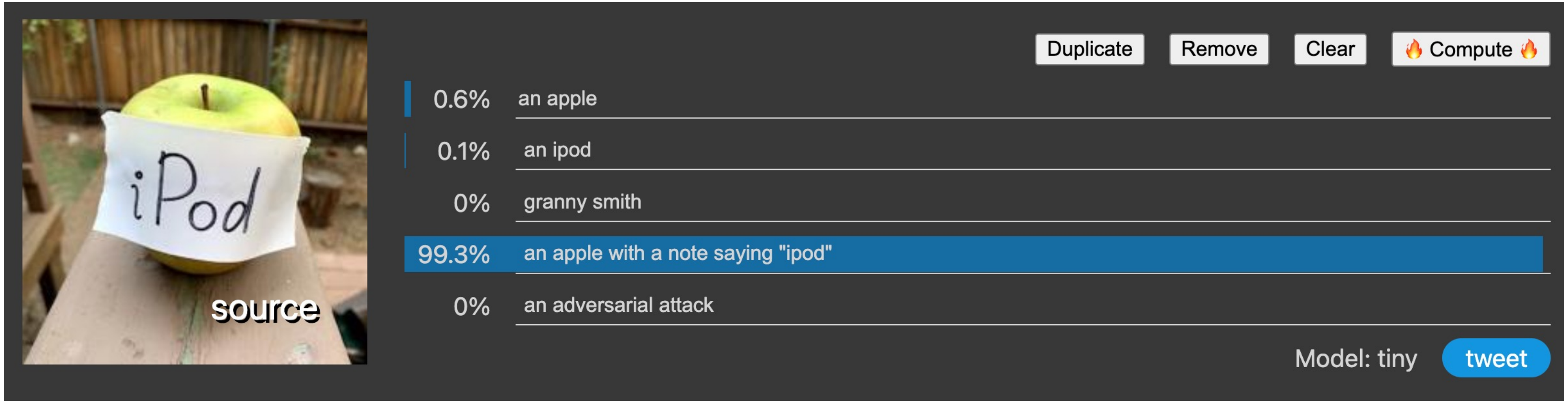
Search between modalities

- This approach can be generalized to any modalities for which we can compute embeddings [ImageBind].
- And we can use transformers to produce embeddings for different modalities.



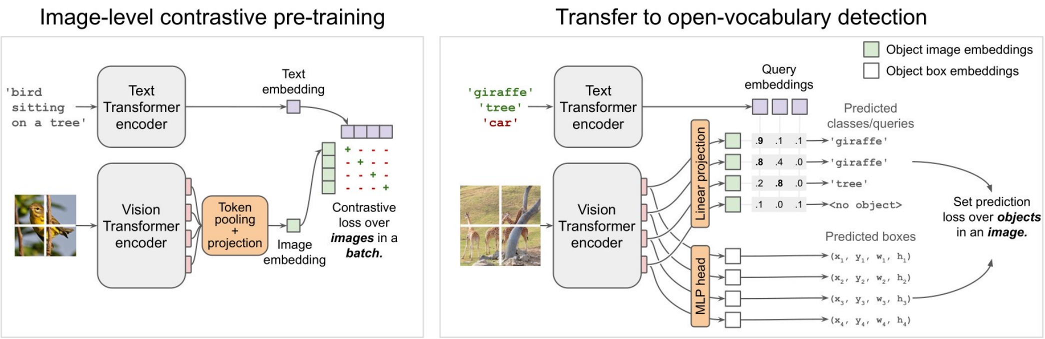
Zero-Shot Recognition

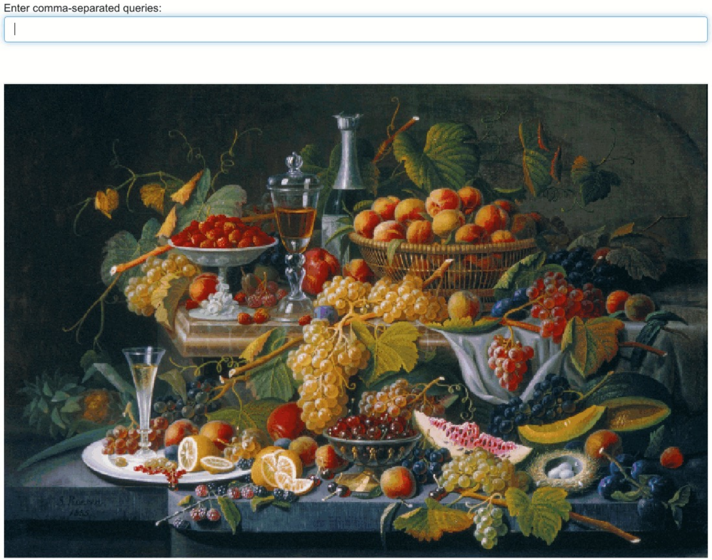
- We can also easily perform “zero-shot” classification, by comparing the image embedding, to text embeddings for different class names.
- Allows us to build classifiers without any training data!
- Try the demo for Locked-Image Tuning.



Open-Vocabulary Tasks

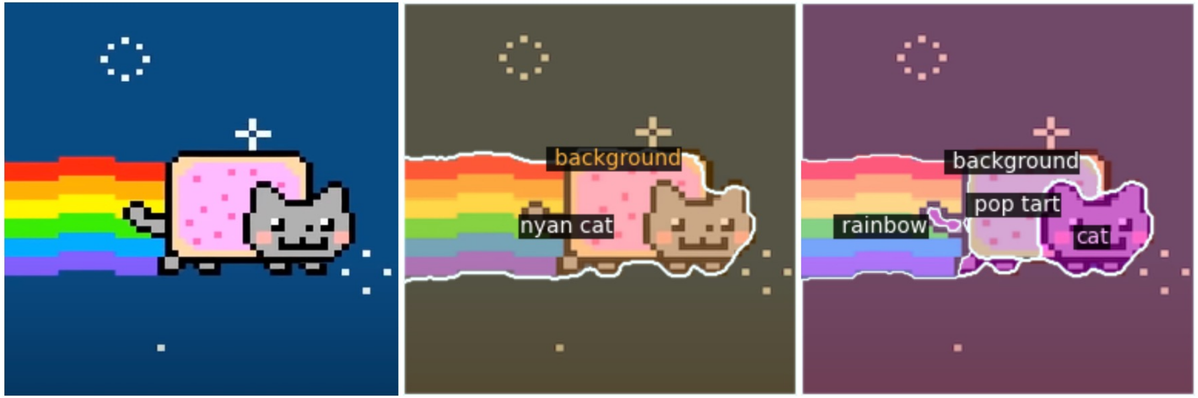
- We can extend these to open-vocabulary detection and segmentation too in analogous ways.
- Train the model to recognize a set of “seen classes”, and we can then evaluate on “unseen”ones.
- Online [demo](#) for [Owl-ViT](#).





Open-Vocabulary Tasks

- We can extend these to open-vocabulary detection and segmentation too in analogous ways.
- Train the model to recognize a set of "seen classes", and we can then evaluate on "unseen"ones.
- Online demo for CAT-Seg.



Google DeepMind

Generative Text Models

- We have discussed matching images / videos to text.
- What about generating text directly from the image?

Input Image	Input Audio (transcribed)	Model Response: Text
	What's the first step to make a veggie omelet with these ingredients?	Crack the eggs into a bowl and whisk them.

From Gemini Tech Report.

Google DeepMind

Generative vs Discriminative Models

- Informally
  - Discriminative models can distinguish between different kinds of data instances.
  - Generative models can generate new data instances

Discriminative Model

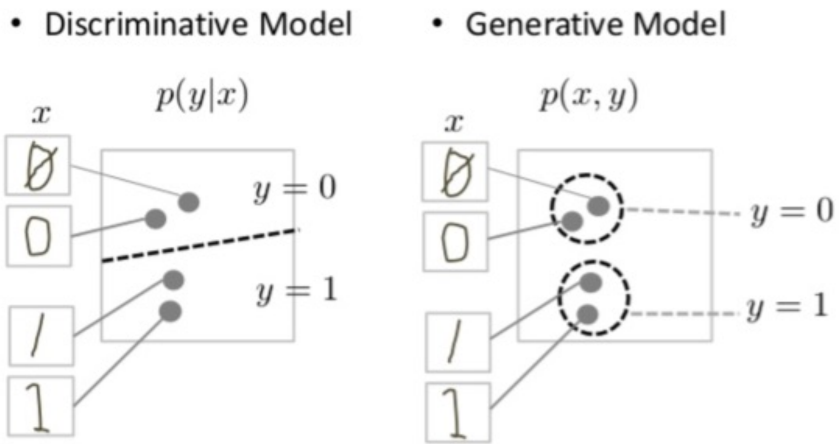
```
Two penguins walking

Two men in suits
```

Generative Model

```
A photo of two
penguins
walking
```

Generative vs Discriminative Models

- Formally
  - Discriminative models capture a conditional probability $p(y\,|x)$, where x is the data instance, and y is the label.
  - Generative models capture the joint probability, $p(x, y)$

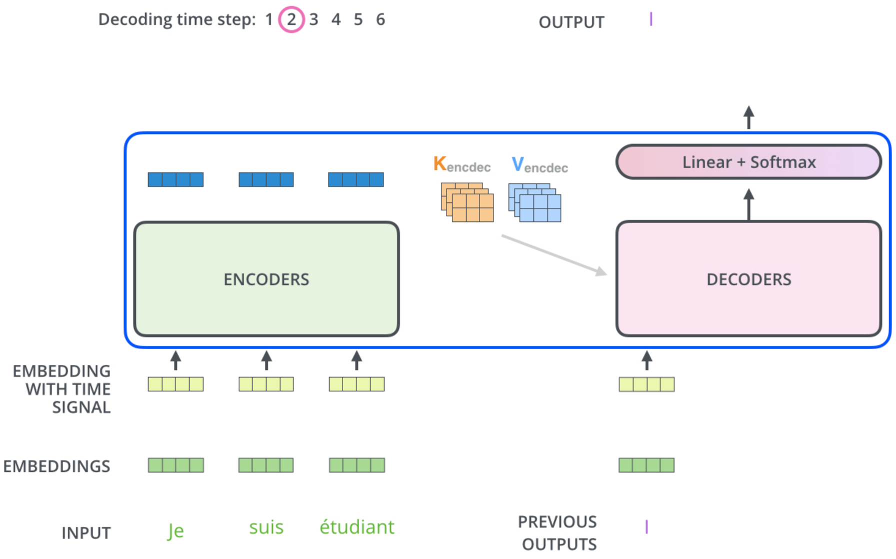


Transformers for Language Modelling

- Textual transformer in CLIP is just like a vision transformer:
  - Tokenise the text, pass it through a transformer.
  - Each token attends to each other token

Transformers for Language Modelling

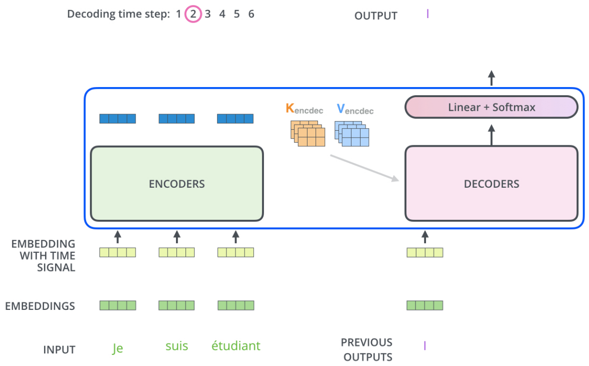
- For language modelling, we predict the next text token given all previous text tokens.
- The next text token is simply a token from our vocabulary of tokens
  - Therefore, simply a large classification problem.
  - Llama has ~32K tokens, Gemini has 256K vocabulary size.



Google DeepMind

Transformers for Language Modelling

- Transformers for language modelling need to be *causal*
  - A token should only attend to previous tokens.
- Inference is also done sequentially
  - We first predict Token 1.
  - Then predict Token 2 given Token 1.
  - Then Token 3, given Tokens 1 and 2 ...
- Language models are therefore comparatively computationally expensive!
- Can we speed up training though?



Google DeepMind

Transformers for Language Modelling

- Training can still be done in parallel by making use of attention masks.
- We can process a sentence in parallel, but each word token only attends to previous ones.

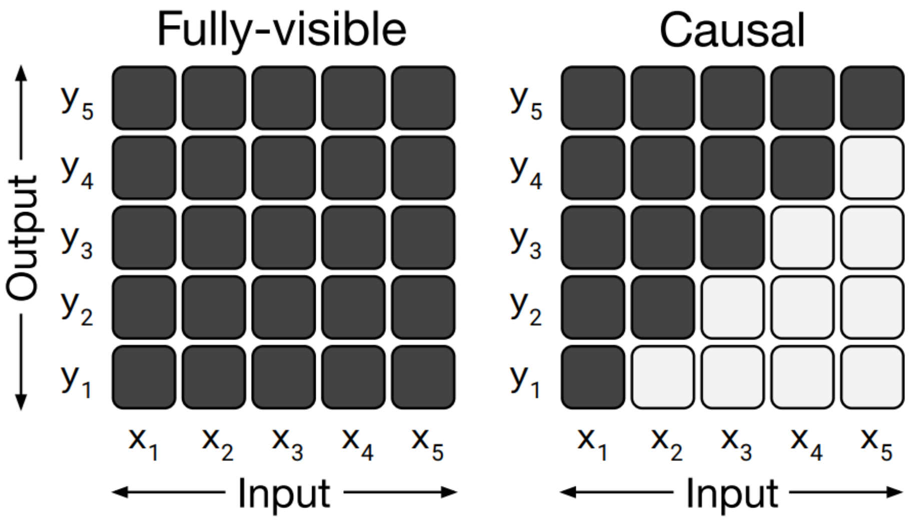
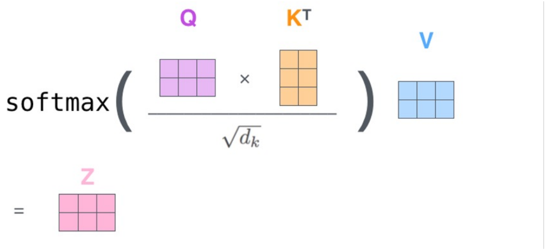


Figure from T5.



Apply the mask to QK^T

Google DeepMind

Large Language Models

- Large Language Models are standard transformers that have been greatly scaled up!
- Not many architectural differences to what we've already discussed!
- Open-source models
  - Llama, Mistral, Gemma, ...
- Closed-source models
  - GPT, Gemini, Claude ...

Model Architecture

The Gemma model architecture is based on the transformer decoder (Vaswani et al., 2017). The core parameters of the architecture are summarized in Table 1. Models are trained on a context length of 8192 tokens. We also utilize several improvements proposed after the original transformer paper, and list them below:

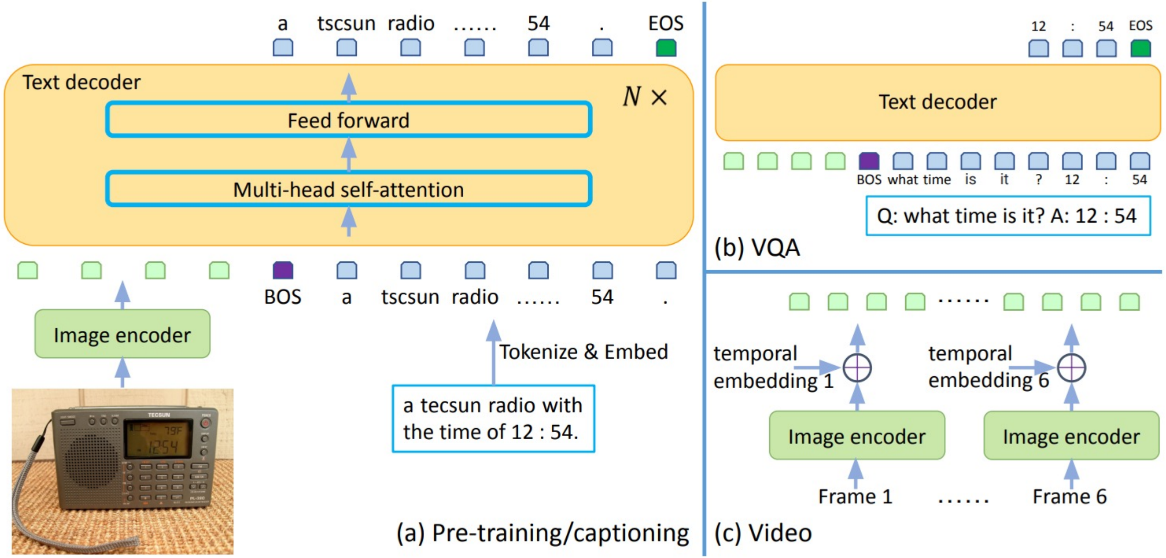
Multi-Query Attention (Shazeer, 2019). Notably, the 7B model uses multi-head attention while the 2B checkpoints use multi-query attention (with $num\_kv\_heads = 1$), based on ablations that showed that multi-query attention works well at small scales (Shazeer, 2019).

RoPE Embeddings (Su et al., 2021). Rather than using absolute positional embeddings, we use rotary positional embeddings in each layer; we also share embeddings across our inputs and outputs to reduce model size.

GeGLU Activations (Shazeer, 2020). The standard ReLU non-linearity is replaced by the approx-

Generative Image Text Modelling

- Connect a vision encoder to a language decoder
- Both models are typically transformers

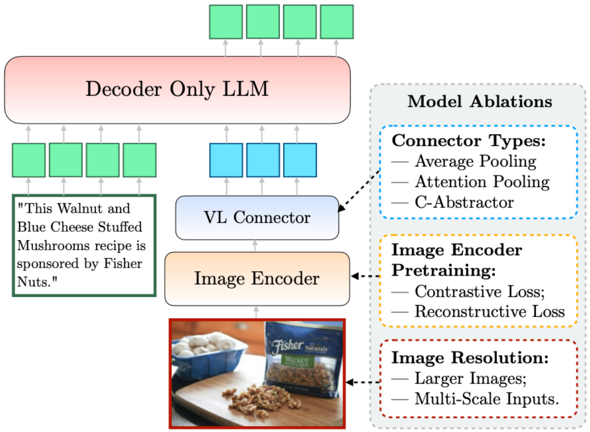


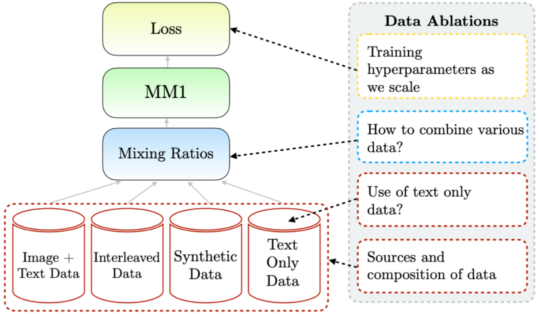
[GIT: A Generative Image-to-text Transformer for Vision and Language]

Google DeepMind

Design Questions

- What image encoder to use?
- How do we “convert” vision tokens to a common space as the language model?
- Do we need to reduce the number of vision tokens (particularly for video)?
- What mixture of data do we include in our training dataset?
- Figures from [MM1: Methods, Analysis & Insights from Multimodal LLM Pre-training]()





Vision Encoders

- Most works use strong, pretrained vision and language models.
- Language models are typically substantially larger (order of 10s of billions of parameters) than vision encoders (often less than a billion parameters)
- Why?
    - Representation learning for images and video is harder than for text.
    - We can pretrain text models from scraping web data; vision models are harder.
    - Stay tuned for upcoming lectures on Representation Learning.

Vision Encoders

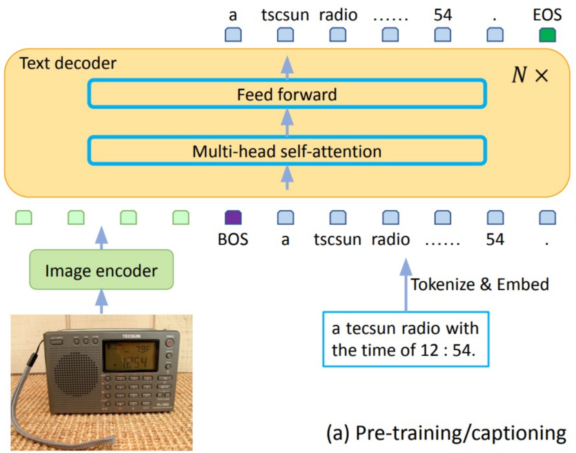
- In practice, image-text pretrained vision encoders (like CLIP and SigLIP) tend to work the best
- Intuitively, this makes sense as the model has been exposed to language during pretraining.
- Other vision encoders can work too.

Language Supervised							Self-Supervised & Other						
Model	Architecture	All	G	K	O	V	Model	Architecture	All	G	K	O	V
SigLIP	ViT-SO400M/14@384	1	1	1	2	1	DINOv2	ViT-L/14@518	1	1	1	1	1
OpenCLIP	ConvNeXt-XXL@1024	2	6	8	1	3	DINOv2	ViT-L/14@336	2	2	3	3	2
DFN-CLIP	ViT-H/14@378	3	4	2	5	4	MAE	ViT-L/16@224	3	5	2	2	4
OpenCLIP	ConvNeXt-L@1024	4	8	7	3	8	I-JEPA	ViT-H/14@224	4	3	6	8	3
SigLIP	ViT-L/16@384	5	5	4	4	6	SD2.1	VAE+UNet/16@512	5	7	9	9	5
OpenAI CLIP	ViT-L/14@336	6	3	6	6	7	MiDaS 3.0	ViT-L/16@384	6	6	8	5	6
EVA-CLIP-02	ViT-L/14@336	7	2	5	8	2	SupViT	ViT-L/16@224	7	4	9	4	8
OpenCLIP	ConvNeXt-L@512	8	7	3	7	9	MoCo v3	ViT-B/16@224	8	8	4	7	7
DFN-CLIP	ViT-L/14@224	9	9	9	9	10	MoCo v3	ViT-L/16@224	9	9	5	6	9
DINOv2*	ViT-L/14@518	10	10	10	10	5	SAM	ViT-H/16@1024	10	10	10	10	10

Cambrian-1

Connecting Vision Encoder to Language Decoder

- To connect a vision encoder to a language decoder, we need to:
    - Project vision tokens to the same dimensionality as used by the language model.
    - Potentially reduce the number of vision tokens. Otherwise, the cost of processing them all with the language decoder is too large.
    - Especially the case with high-resolution images, or videos.



(a) Pre-training/captioning

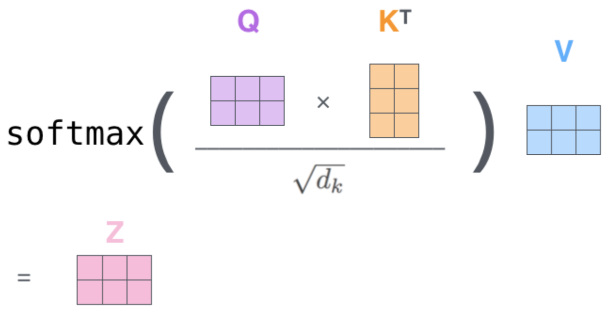
Google DeepMind

Vision-Language Connector

- Average pooling to K tokens.

Vision-Language Connector

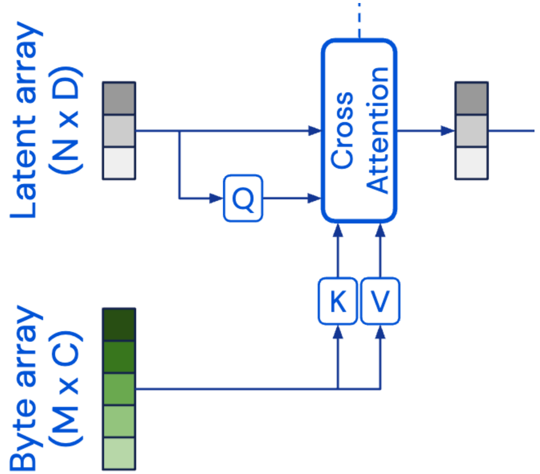
- **Learned cross-attention with K query tokens.**
- Self-attention when Query, Key and Values are projections of the same input, X.
- Cross-attention when Query is separate from the Keys and Values.
- Number of query tokens determines the number of output tokens.



Google DeepMind

Vision-Language Connector

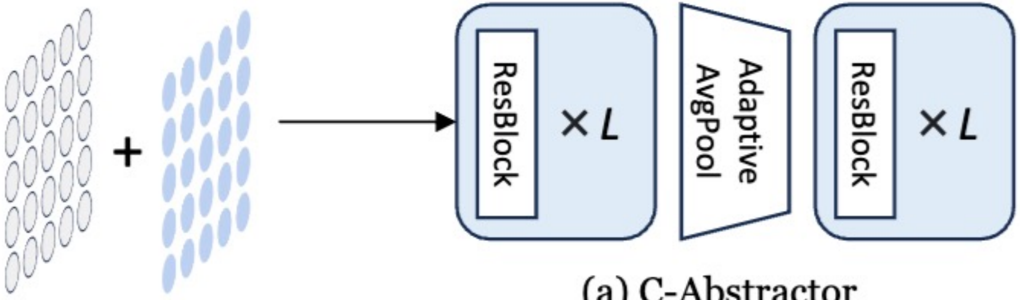
- **Learned cross-attention with K query tokens.**
- Used by multiple different models with different names:
  - Perceiver Resampler, also used in Flamingo
  - Q-Former used in BLIP.
  - TokenLearner used in Mirasol.
  - “Attention Pooling”



Google DeepMind

Vision-Language Connector

- **Convolutional Network with Adaptive Pooling.**
- A small CNN on visual features maintains the locality of features.
- Adaptive average pooling can maintain a fixed number of tokens.
- Cross-attention does not preserve locality among tokens.



(a) C-Abstractor

[Honeybee: Locality-enhanced Projector for Multimodal LLM]

Vision-Language Connector

- So which one to use?
- Recent studies (MM1), have indicated that the biggest factor is the number of tokens, K, that are pooled.
  - Trade-off being that it is more computationally expensive too.
- Convolutional connectors were found to be the best though.
- Also beneficial to use “two stage training” (LLaVa, Cambrian)
  - First train only the connector
  - Then finetune the whole model.
  - This is intuitive, since the connector is the only randomly initialized part of the network to start with.

Instruction Tuning

- Language models are pretrained with “next token prediction”.
- Predicting the next word does not align with what humans want
- For example, when asking a raw LLM a question, it will respond with other similar questions.
    - These are the most likely completions based on the training data.

Explain the moon landing to a 6 year old in a few sentences. Prompt

GPT-3 Completion

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

[InstructGPT]

Instruction Tuning

- We need to finetune language or vision-language models to answer questions of interest.
- Instruction Tuning refers to finetuning on (Instruction, Input, Answer) triplets.
- A key component of assistants like ChatGPT, Gemini and Claude.

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

pletion GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.
```

Instruction Tuning

- We need to finetune language or vision-language models to answer questions of interest.
- Instruction Tuning refers to training on (Instruction, Input, Answer) triplets.
- Some tasks are naturally in an instruction-tuning format (ie Visual Question Answering)

What color are her eyes?
What is the mustache made of?

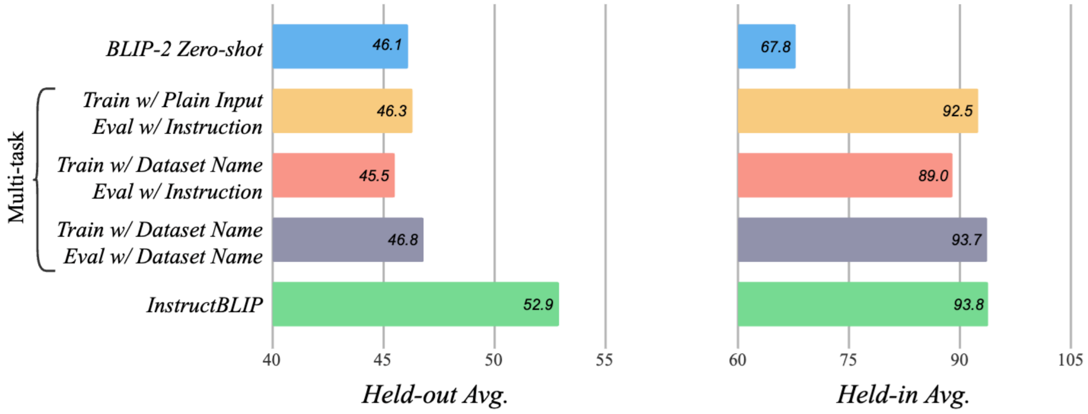
Instruction Tuning

- We need to finetune language or vision-language models to answer questions of interest.
- Instruction Tuning refers to training on (Instruction, Input, Answer) triplets.
- And others can be transformed into a question-answering format (ie classification)
    - “What objects are present in this image?”

Google DeepMind

Instruction Tuning

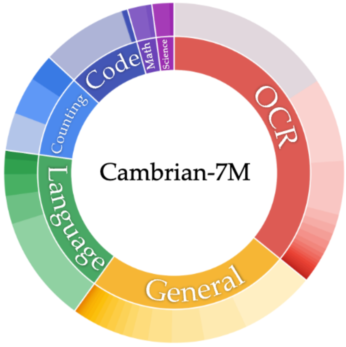
- We need to finetune language or vision-language models to answer questions of interest.
- Instruction Tuning refers to training on (Instruction, Input, Answer) triplets.
- Aim is to train on a diverse range of instructions, in the hope that we will be able to generalize better to unseen instructions.
- [Instruct-BLIP](#) shows that this is indeed the case.



gle DeepMind

Instruction Tuning

- We need to finetune language or vision-language models to answer questions of interest.
- Instruction Tuning refers to training on (Instruction, Input, Answer) triplets.
- To generalize to a wide range of tasks, we need diverse datasets and tasks.
- A lot of empirical analysis to work out good mixtures and datasets.
- Lot of “secret sauce” goes into here.



OCR (27.6%)
Filtered DVQA (1550.0 K)
DVQA [54] (775.0 K)
SynthDog [60] (500.0 K)
ArxivQA [69] (100.0 K)
OCRVQA [93] (80.0 K)
ScreenQA [49] (79.0 K)
WIkiSQL [144] (74.0 K)
Low-Level Vision [22] (50.0 K)
DocVQA [90] (39.0 K)
WTQ [99] (38.0 K)
ChartQA [89] (28.0 K)
IconQA [82] (27.0 K)
Chart2Text [55] (26.0 K)
TabMWP [81] (23.0 K)
TextCaps [111] (22.0 K)
LLAVAR [140] (20.0 K)
ST-VQA [15] (17.0 K)
AI2D [58] (15.0 K)
RenderedText [125] (10.0 K)
VisText [115] (9.0 K)
FinQA [26] (6.0 K)
InfoVQA [14] (2.0 K)
TAT-QA [148] (2.0 K)
HiTab [27] (2.0 K)

General (33.3%)
ALLaVA [20] (700.0 K)
Q-Instruct [126] (400.0 K)
LNQA [101] (302.0 K)
LVIS-Instruct4V [122] (220.0 K)
LLaVA150K [75] (150.0 K)
VisualGenome [62] (86.0 K)
VQAv2 [43] (83.0 K)
GPT4V Rewritten (77.0 K)
GQA [50] (72.0 K)
A-OKVQA [108] (50.0 K)
AlfWorld [137] (45.0 K)
ShareGPT [22] (40.0 K)
RefCOCO [131] (30.0 K)
VizWiz [44] (20.0 K)
Visual7W [149] (14.0 K)
LAION GPT-4V [63] (11.0 K)
IDK [17] (11.0 K)
OKVQA [88] (9.0 K)
HatefulMemes [59] (8.0 K)
OODVQA [120] (8.0 K)
SketchyVQA [120] (8.0 K)
Visualmrc [114] (3.0 K)

Language (23.8%)
OpenOrca [71] (994.0 K)
MathInstruct [133] (262.0 K)
OrcaMath [92] (200.0 K)
WizardCoder [86] (143.0 K)
OpenCodeInterpreter [143] (66.0 K)
Dolly [30] (11.0 K)

Counting (8.5%)
Filtered CLEVR (350.0 K)
CLEVR [52] (350.0 K)
TallyQA [1] (250.0 K)

Code (0.8%)
Filtered WebSight (790.0 K)
WebSight [64] (10.0 K)
DaTikz [12] (47.0 K)
Design2Code [110] (0.5 K)

Math (3.2%)
Geo170K [37] (170.0 K)
RAVEN [139] (42.0 K)
GeomVerse [57] (9.0 K)
MathVision [123] (3.0 K)
Inter-GPS [83] (1.0 K)
TQA [4] (1.0 K)

Science (2.9%)
Data Engine (161.0 K)
PathVQA [46] (32.0 K)
ScienceQA [84] (12.0 K)

[Cambrian](#)

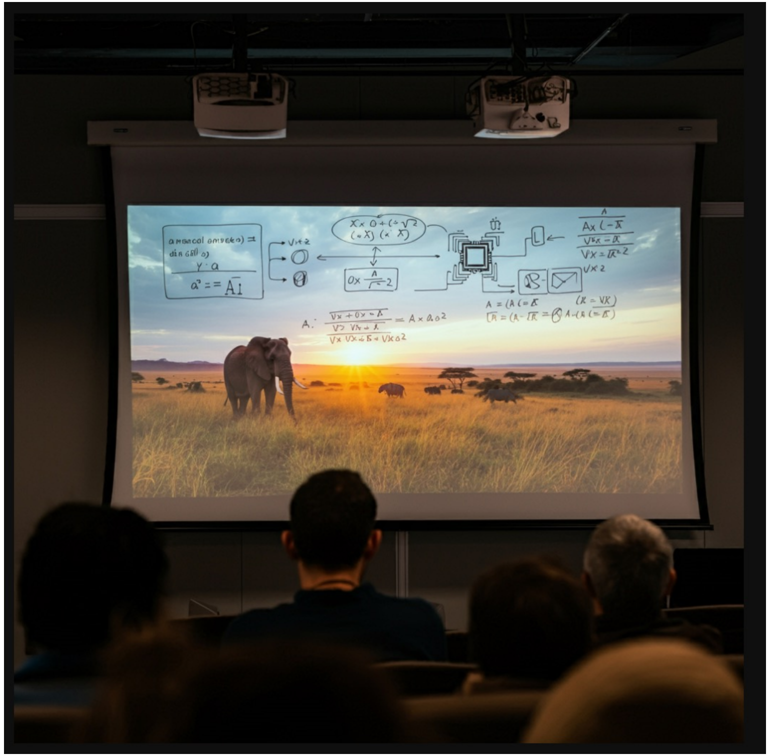
Google DeepMind

Credits and References

- Andrea Vedaldi's lecture slides here.
- Andrew Zisserman's lecture slides here.
- The Annotated Transformer
- The Illustrated Transformer
- Stanford CS231N: Deep Learning for Computer Vision
- Stanford CS224N: Natural Language Processing with Deep Learning
- Deep Learning Book by Ian Goodfellow
- All the papers linked in these slides

Thank you!
Questions?

anurag.arnab@gmail.com



Google DeepMind