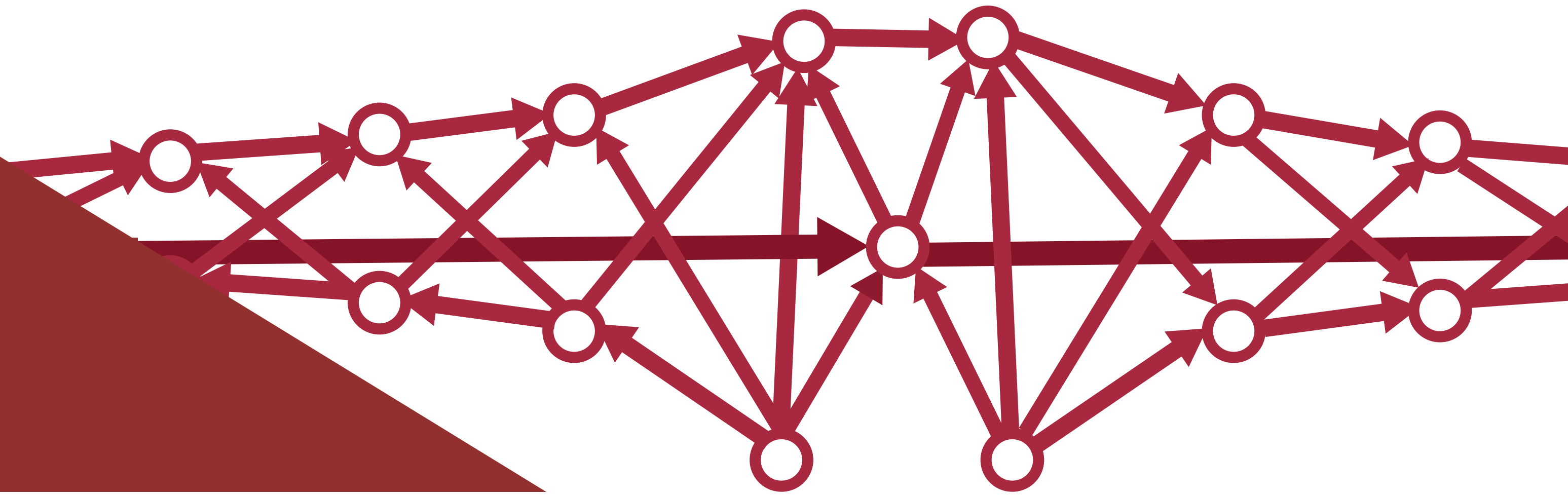


Machine Learning & Data Analysis



Week 10: Modern neural network architectures and transfer learning

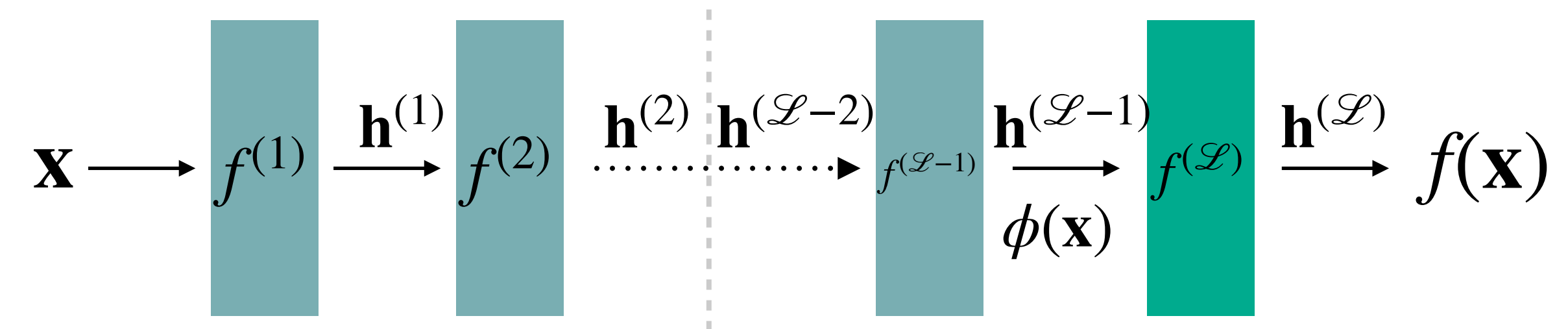
Elliot J. Crowley, 23/11/26



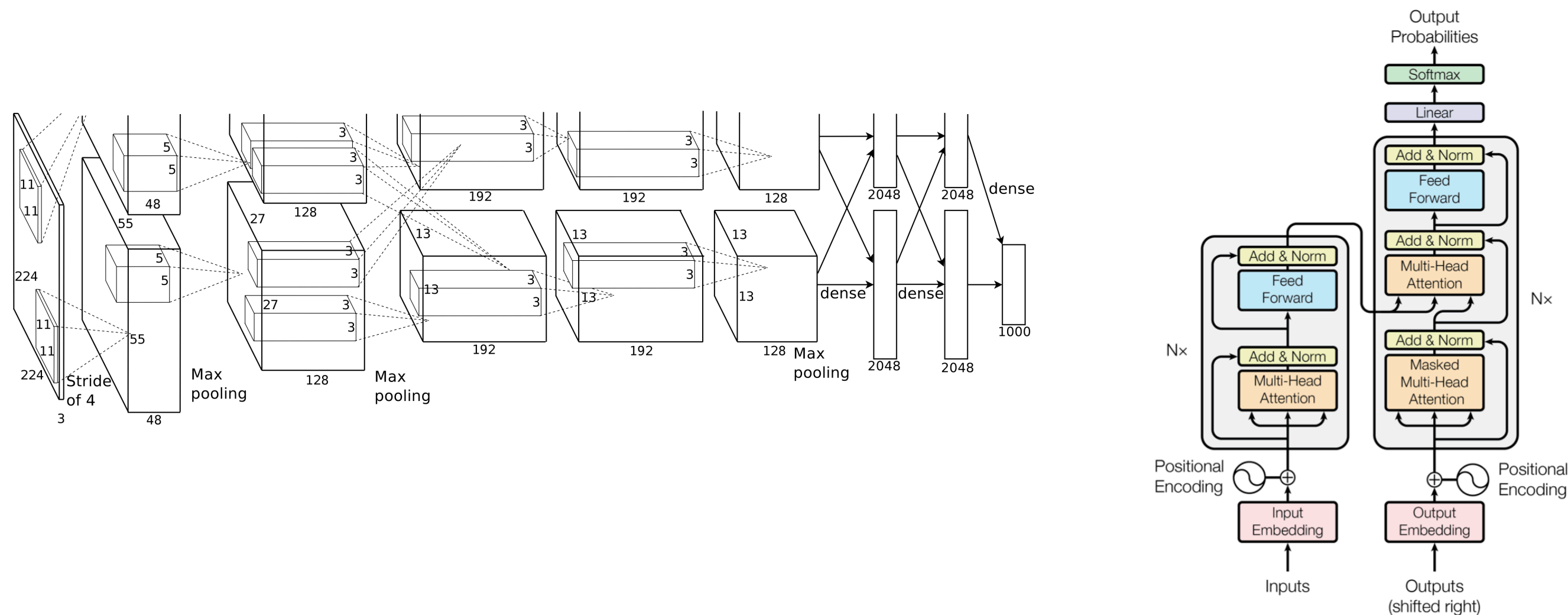
THE UNIVERSITY
of EDINBURGH

Recap

- We learnt about deep neural networks (DNNs) as models that incorporate feature learning into a given task

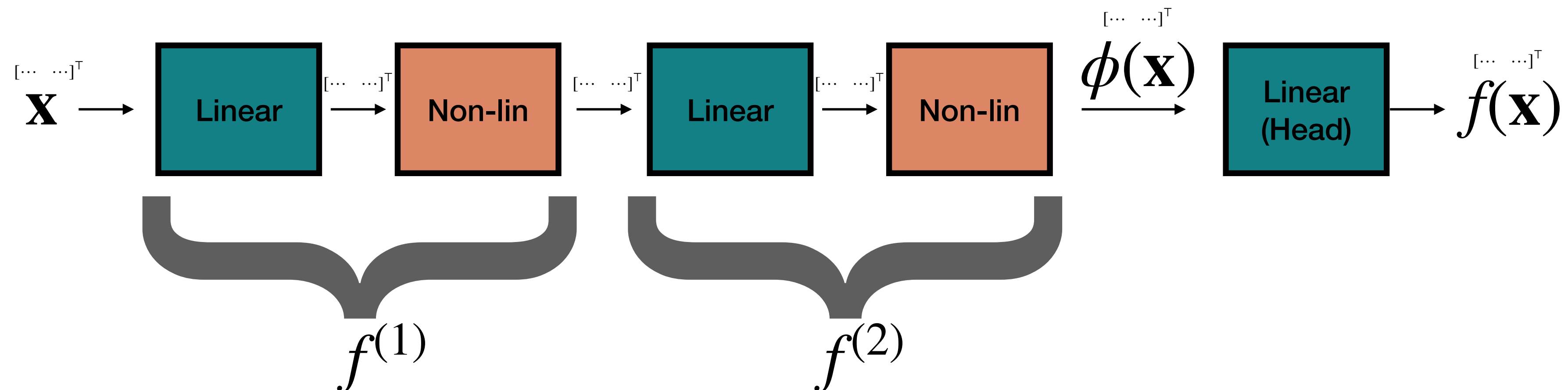


- We very briefly saw ConvNets and Transformers

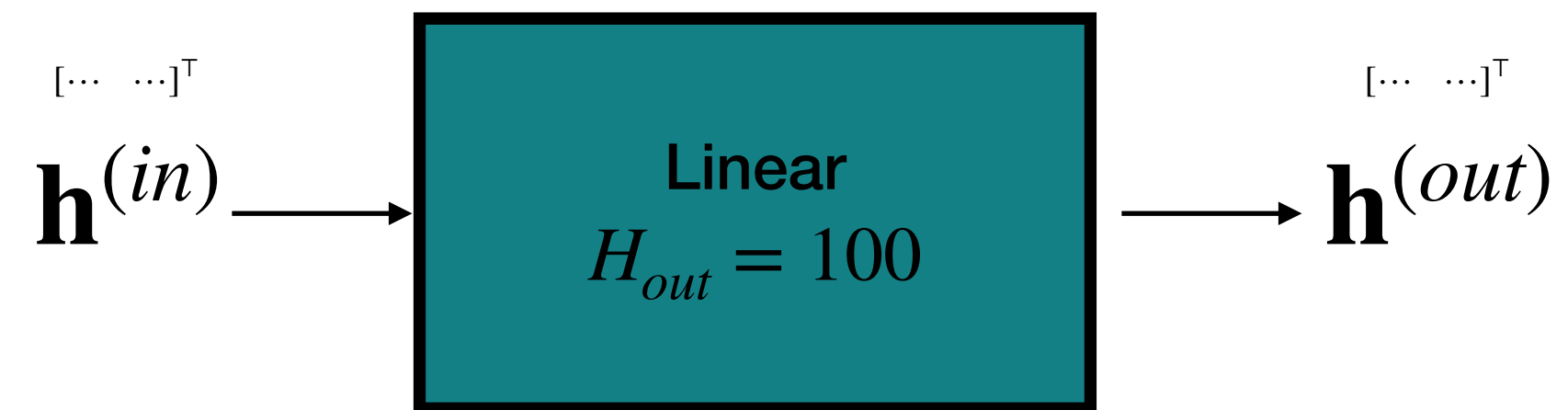


Building blocks

- DNNs are made up of differentiable building blocks (think Lego pieces)
- An MLP consists of two such blocks
- We can think of block outputs as *intermediate representations*
- Intermediate representations are all vectors for an MLP



The blocks so far



$$\mathbf{h}^{(out)} = \mathbf{W}\mathbf{h}^{(in)} + b$$

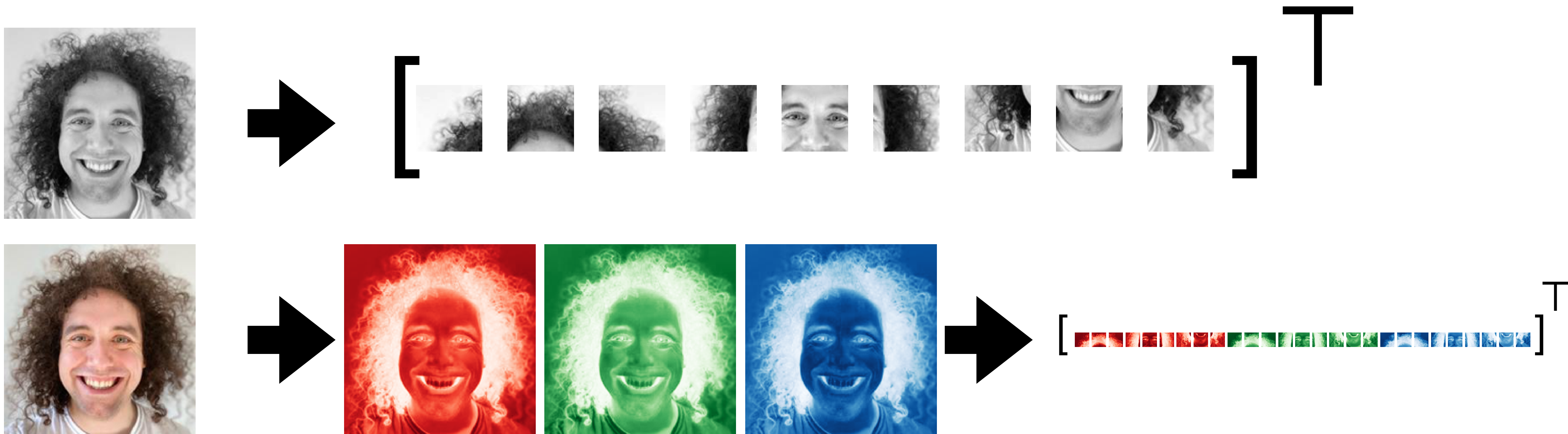
$$\mathbf{h}^{(out)} \in \mathbb{R}^{100}$$



$$\mathbf{h}^{(out)} = \text{ReLU}(\mathbf{h}^{(in)})$$

Images

- So far we have represented all our data points as vectors $\mathbf{x} \in \mathbb{R}^D$
- This makes sense with tabular data. Each dimension has a distinct meaning
- Does it make sense to vectorise images?



Location, location, location

- Objects can be in different places and at different scales across images
- If you vectorise then you are rarely comparing like-for-like at each dimension



Structure

- Objects have a spatial structure. The relative position of parts is important
- We lose this information if we vectorise



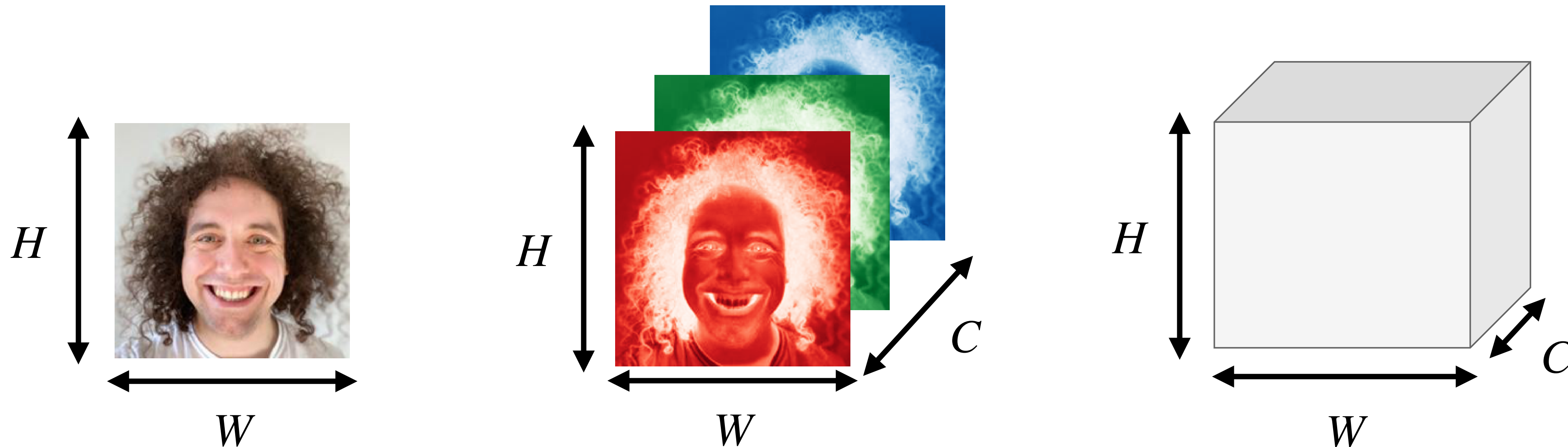
Locality

- In an image, pixels near each other tend to relate to the same object
- We lose any sense of locality when we vectorise images

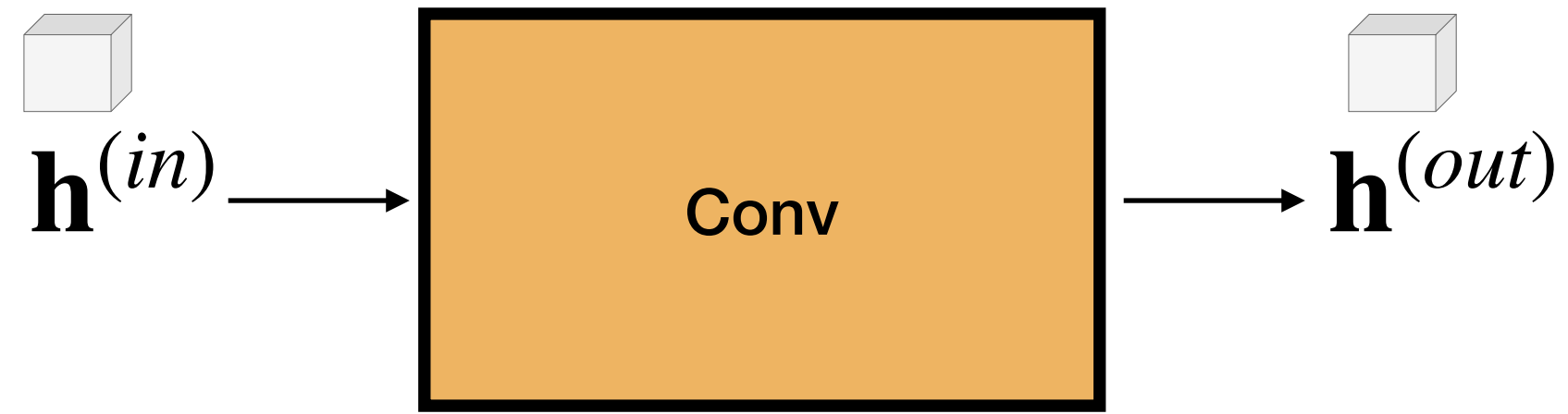
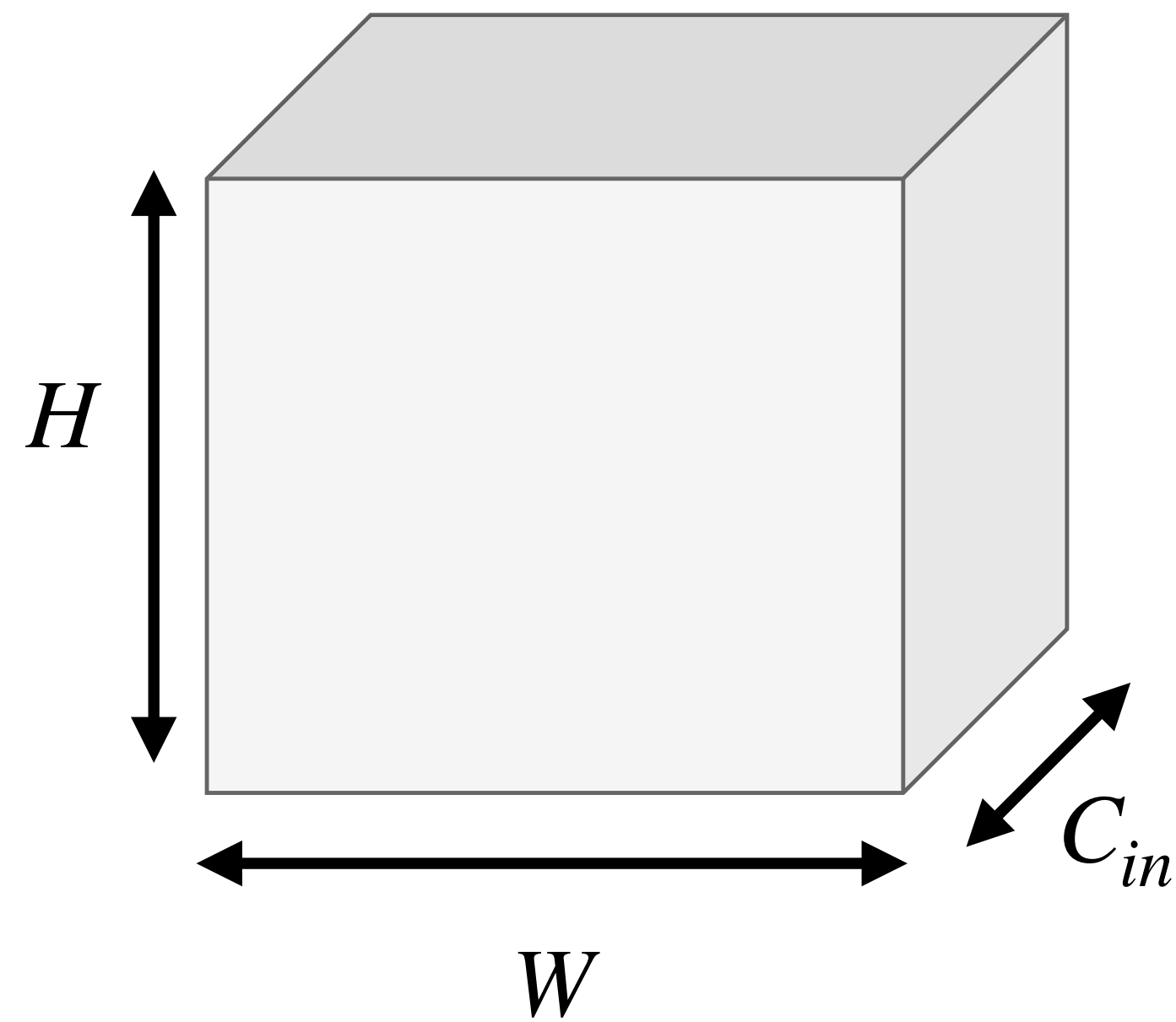


Image mode

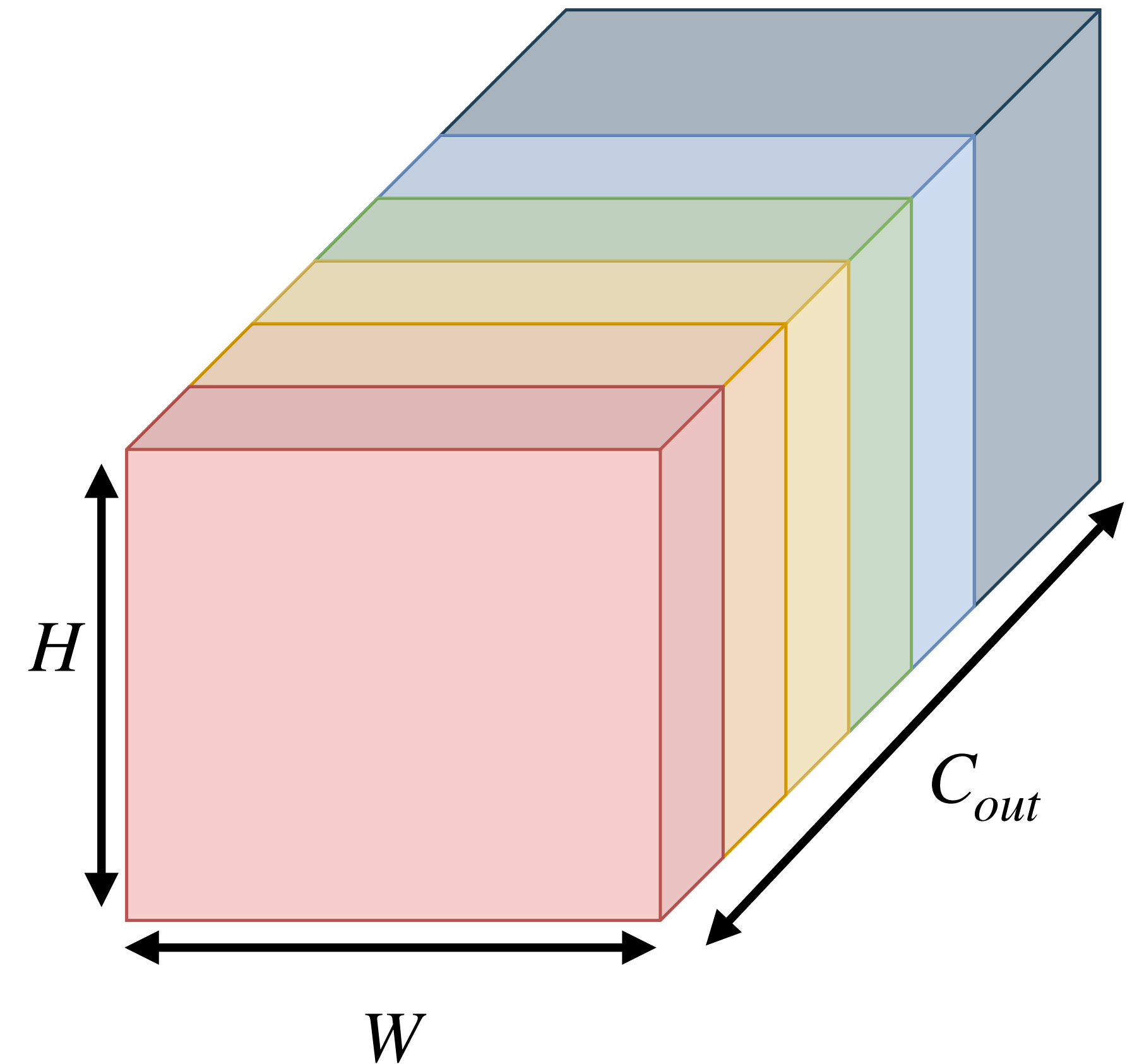
- Let's keep images in their original form
- This is a **cube(/oid)** with dimensions $C \times H \times W$ where C is the number of colour channels (almost always 3)
- We can represent this mathematically using a 3D tensor $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$



A convolutional block



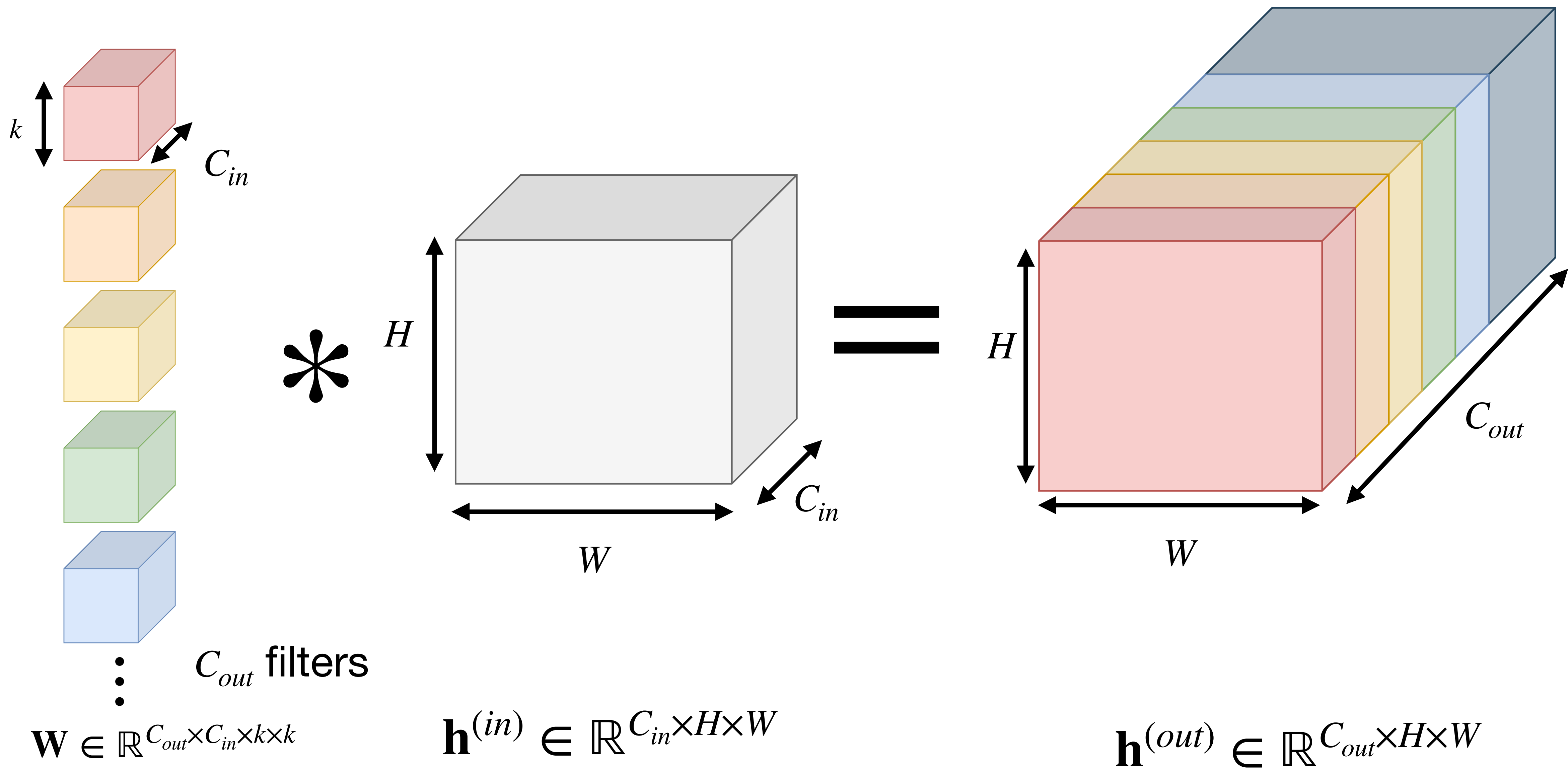
$$\mathbf{h}^{(in)} \in \mathbb{R}^{C_{in} \times H \times W}$$



$$\mathbf{h}^{(out)} \in \mathbb{R}^{C_{out} \times H \times W}$$

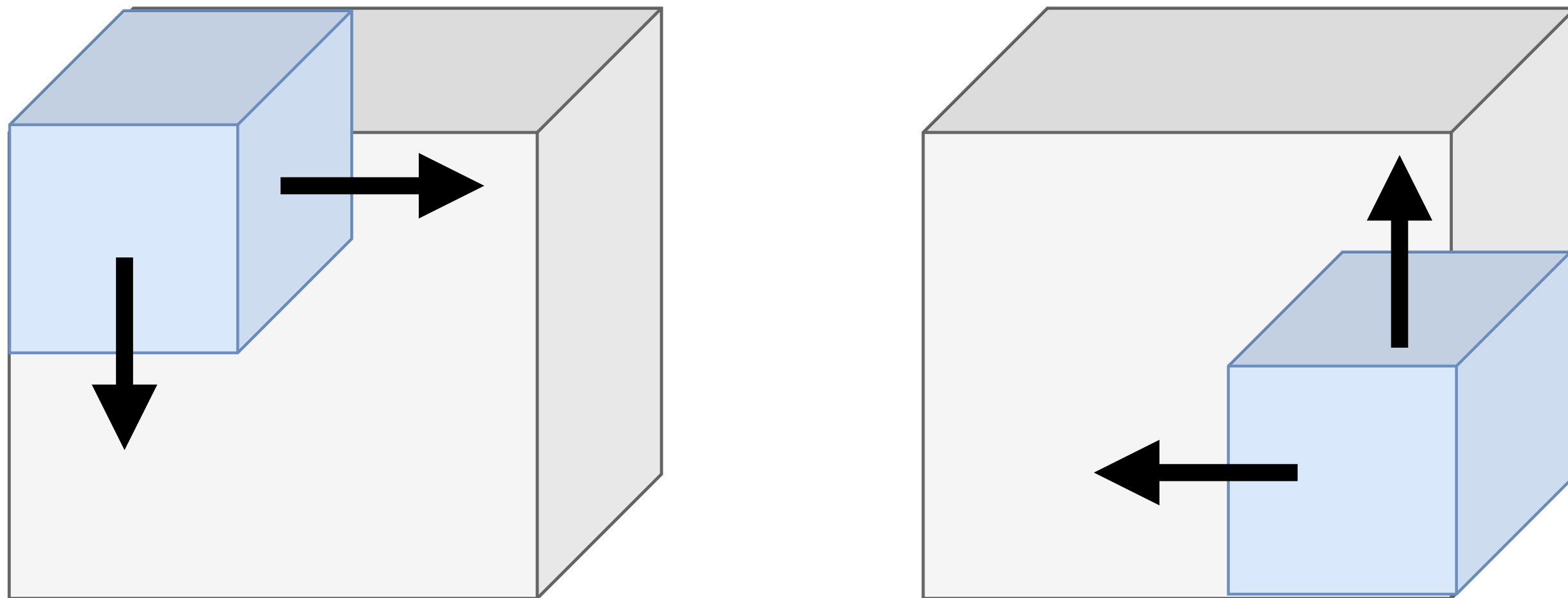
There will be implicit padding for some values of k

A convolutional block



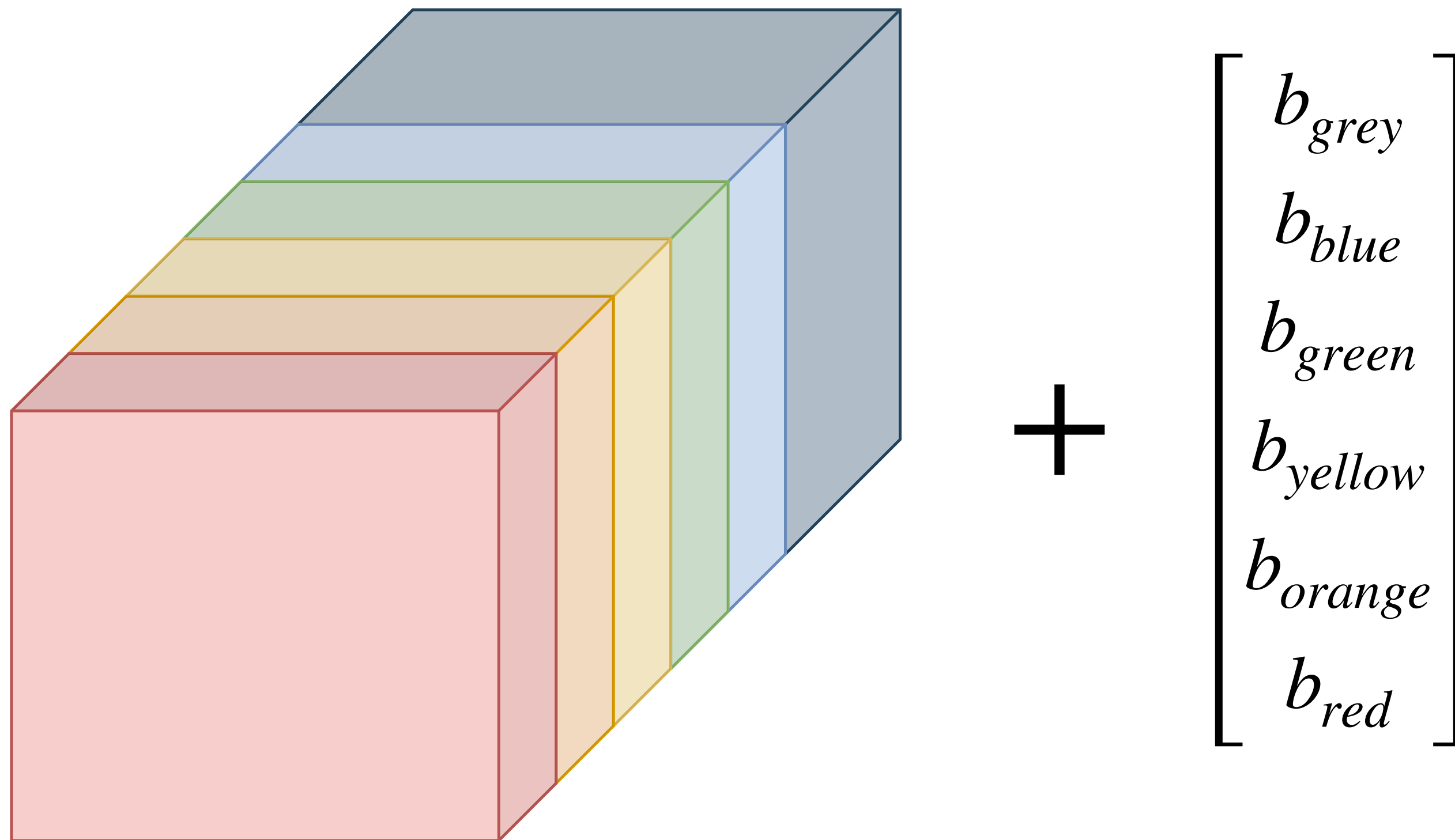
Cube in a cube

- Picture sliding the filter cube around inside the input cube
- It can't move along the z axis because the cubes have the same depth
- It can only move left/right and up/down
- At each position, you take a dot product and store it in a matrix



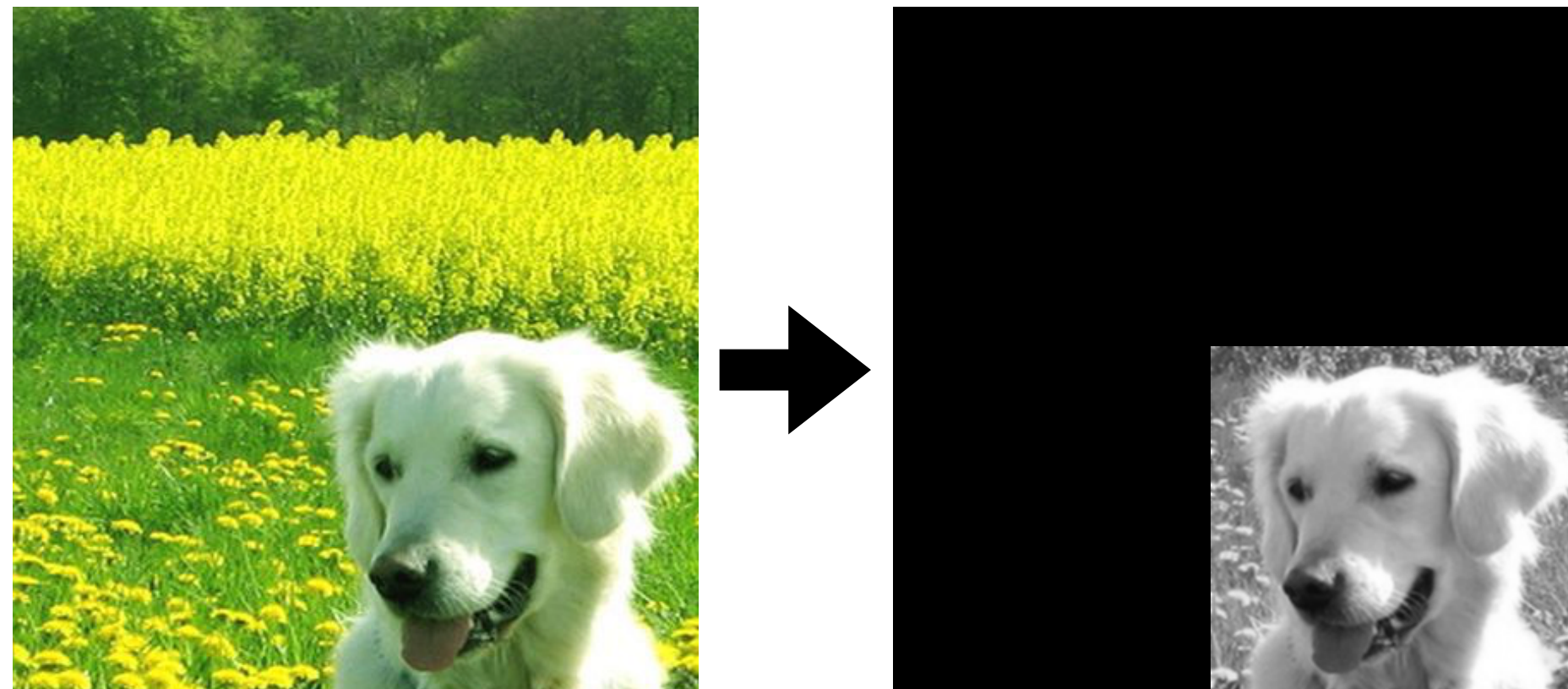
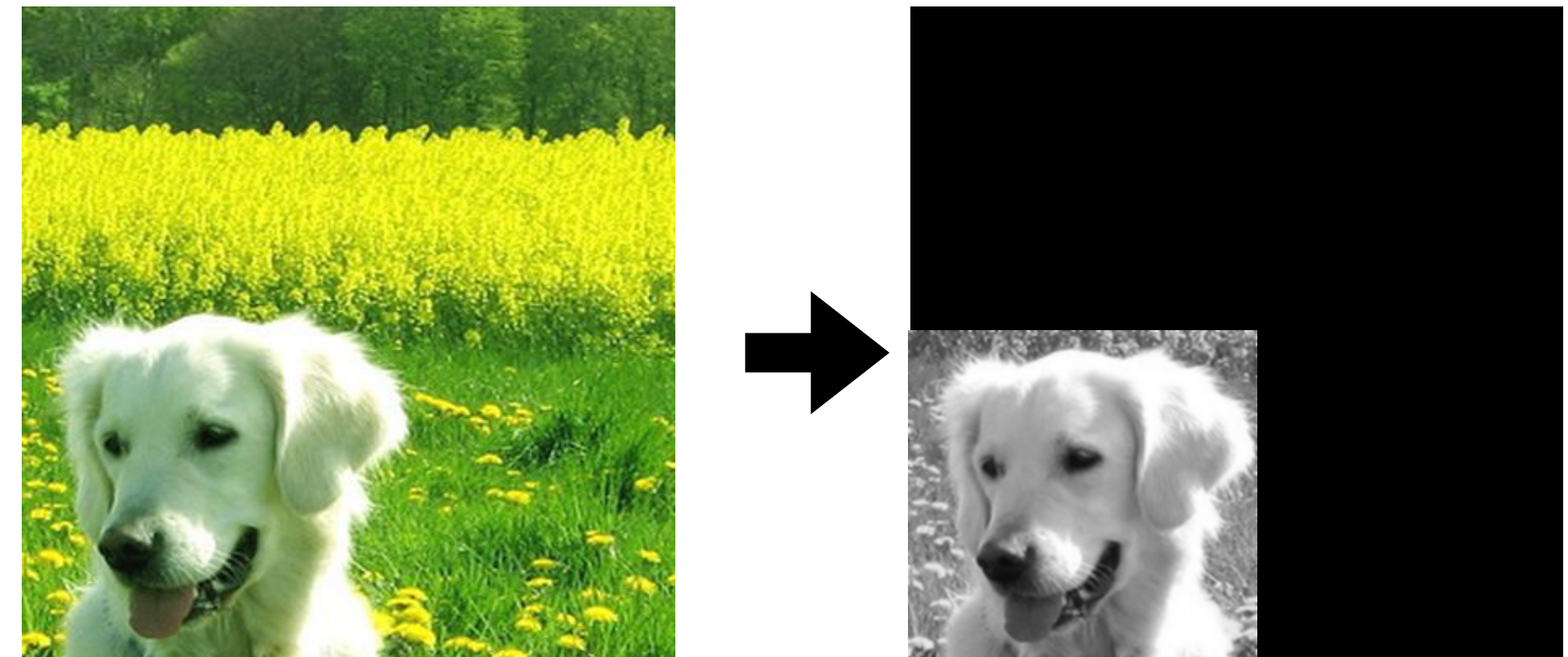
And the bias

- By default, we add a bias to the output of a convolution
- Specifically, each output channel has its own scalar bias that is added to all elements



Why convolutions?

- Filters are applied to the whole image, they aren't tied to a certain region
- This means they can deal with objects moving about; they'll produce a similar output response, just at a different location
- They are **translation equivariant**



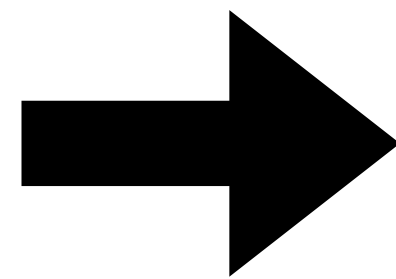
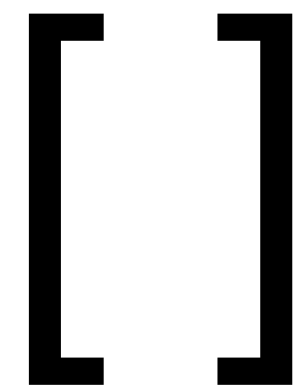
Why convolutions?

- They are much more parameter-efficient than the matrices seen in MLPs
- e.g. Let's have a simple two-layer MLP for classifying my face vs. other faces
- Let's go with hidden dimension 100. How many parameters does $\mathbf{W}^{(1)}$ use?



$$\mathbf{x} \in \mathbb{R}^{3 \times 224 \times 224}$$

vectorise



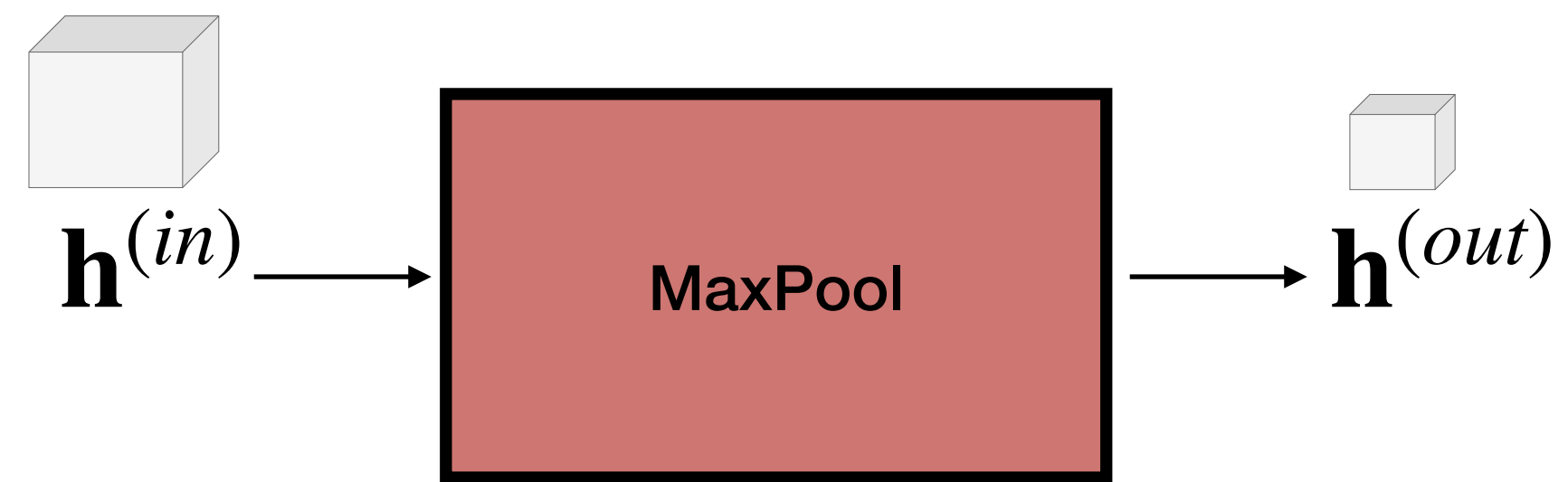
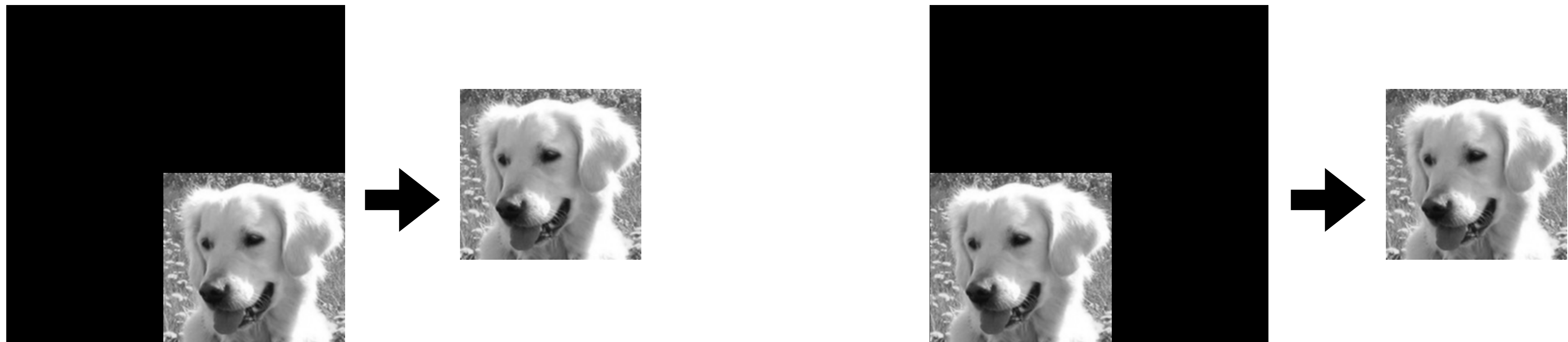
$$\mathbf{x} \in \mathbb{R}^{150528}$$

$\mathbf{W}^{(1)}$ needs to be 100×150528

That's 15 million parameters!

Translation invariance

- If we want the same response, regardless of location we can follow a Conv layer with a (maximum) **pooling layer**



Pooling layers

- These reduce the spatial resolution of their input by aggregating nearby elements
- Let's look at an example on a 2D input of a max pooling layer with $k = 2$

1	5	3	2
0	2	1	1
4	4	6	35
4	4	6	17

MaxPool ($k = 2$)



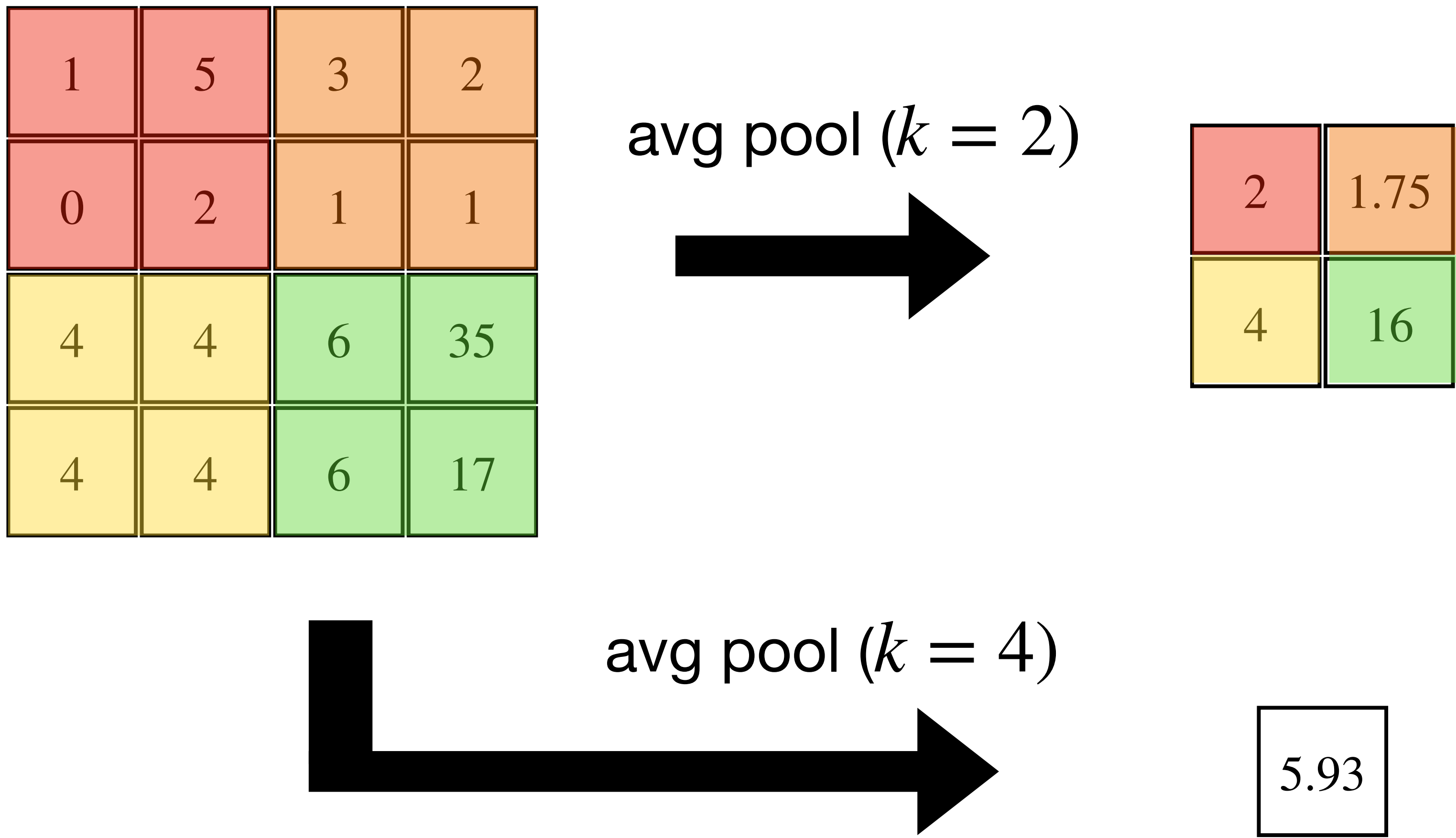
5	3
4	35

The input has been split into 2×2 blocks

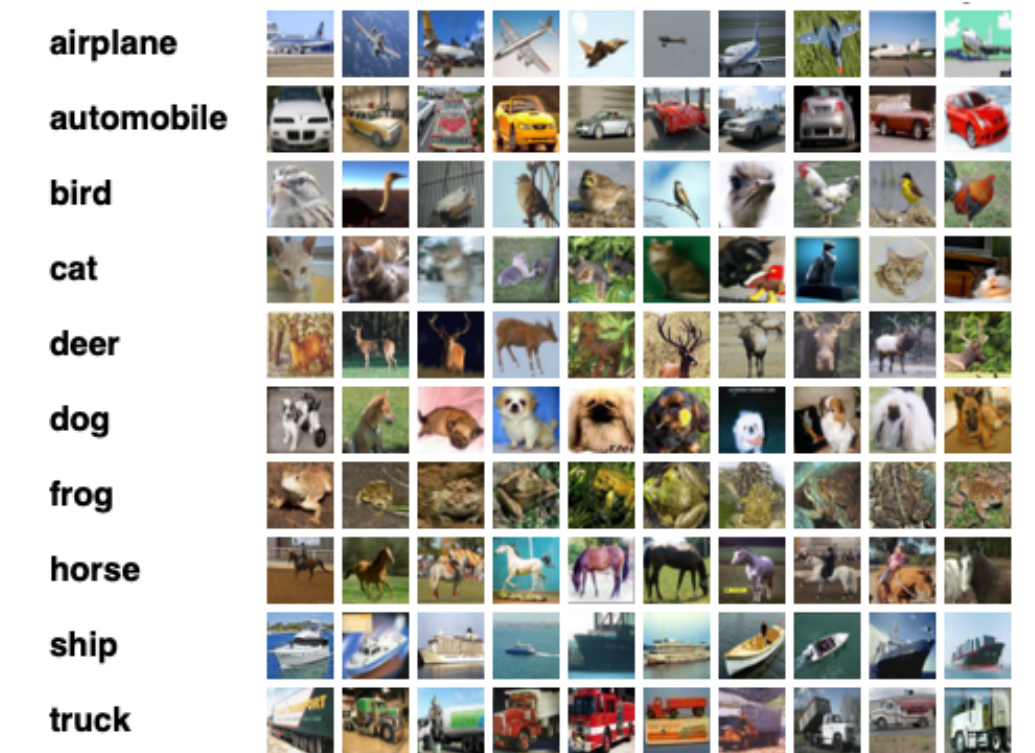
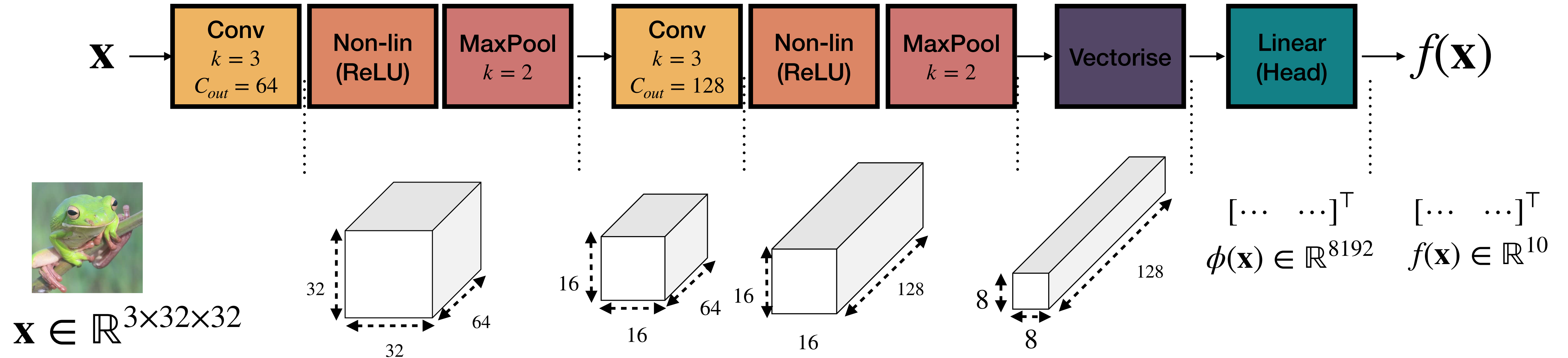
The output matrix contains the maximum value within each block

Its spatial resolution has been halved

Average pooling



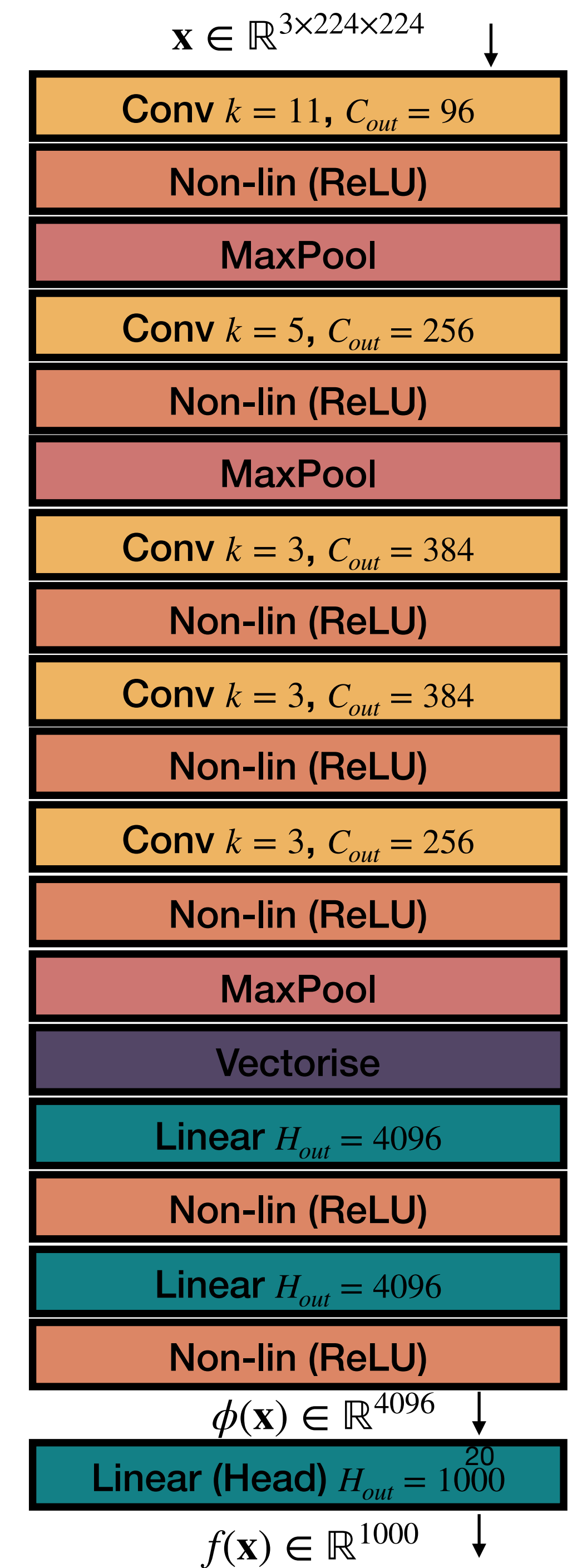
A small ConvNet for CIFAR-10 classification



A big (at the time) ConvNet for ImageNet classification

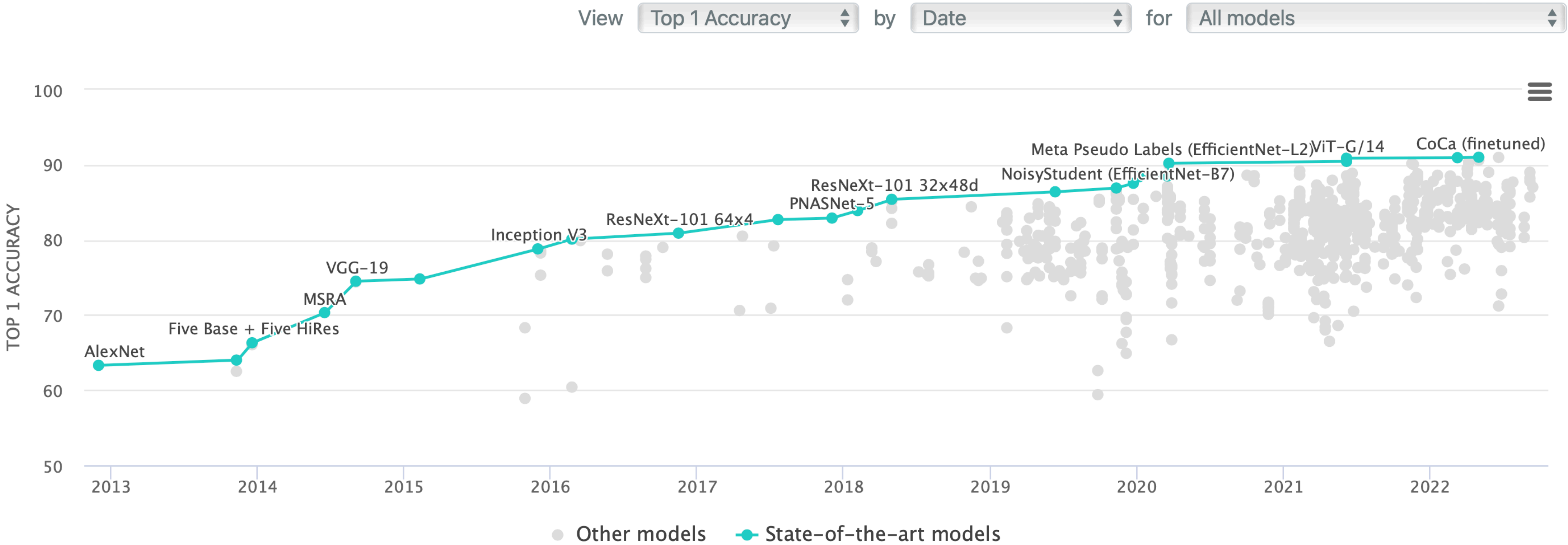


https://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_full_1k.jpg



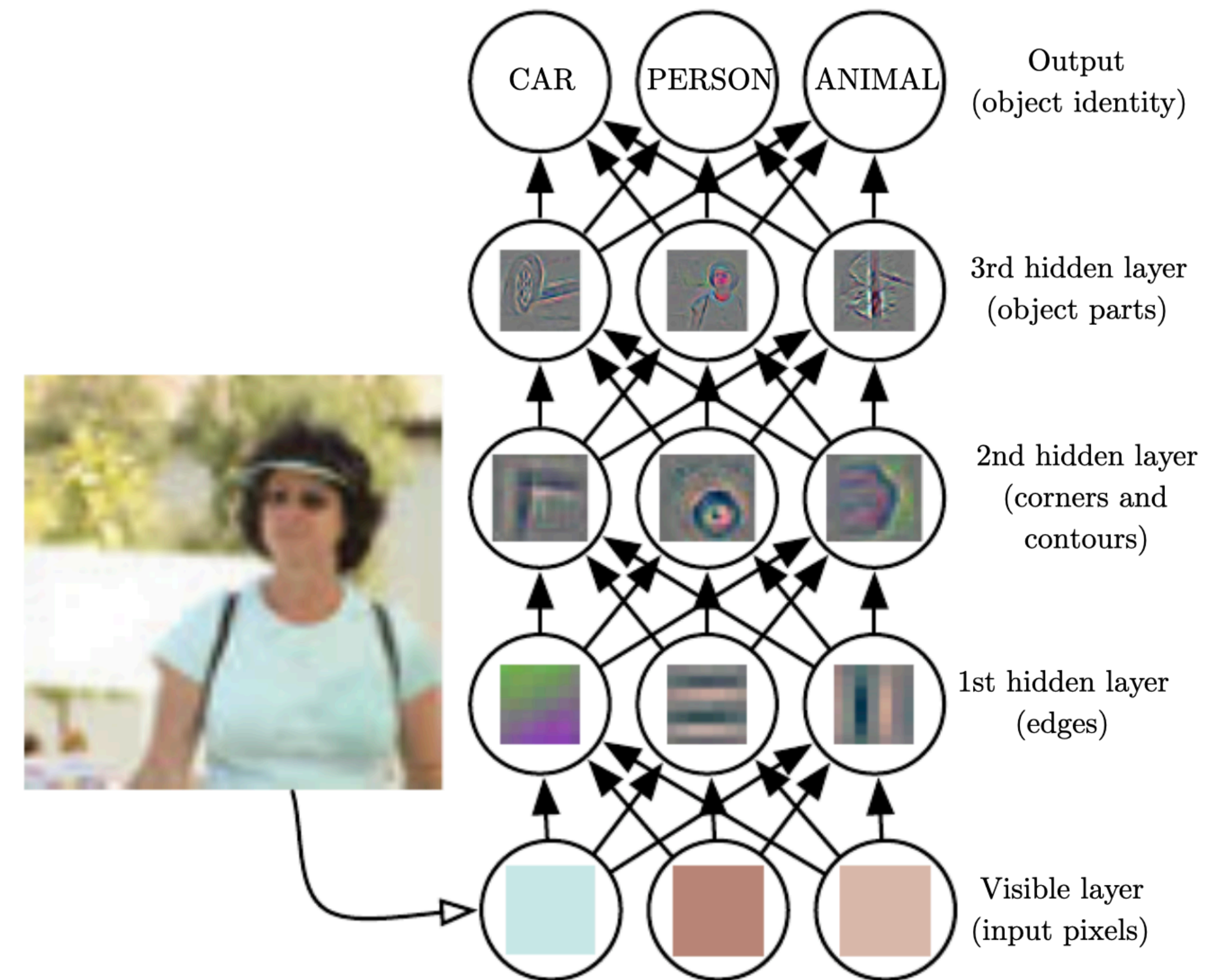
ImageNet top-1 accuracies

Leaderboard Dataset



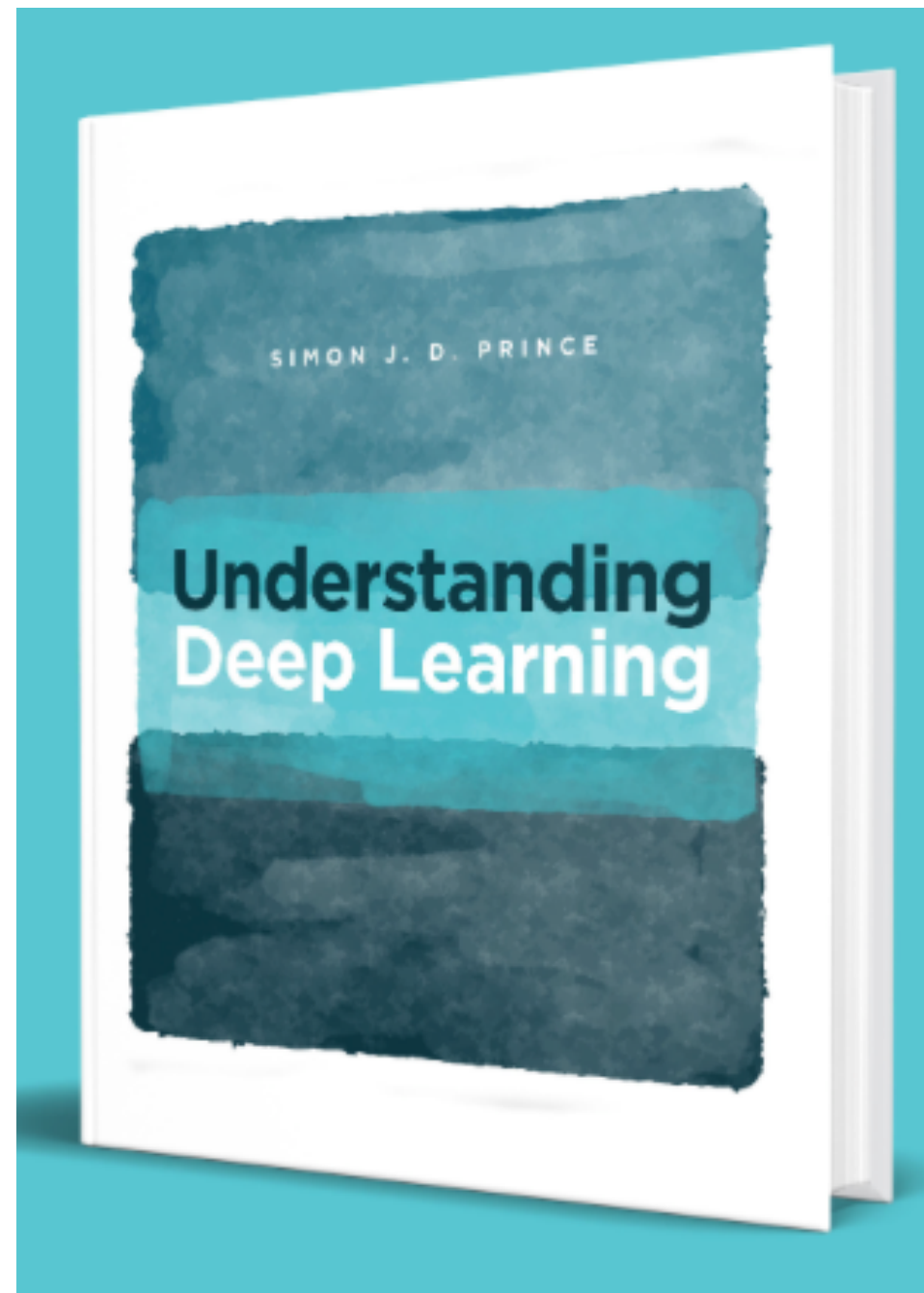
Why does deep learning work? The hand-wavy answer

- Hierarchical feature learning, of course!



Why does deep learning work? The honest answer

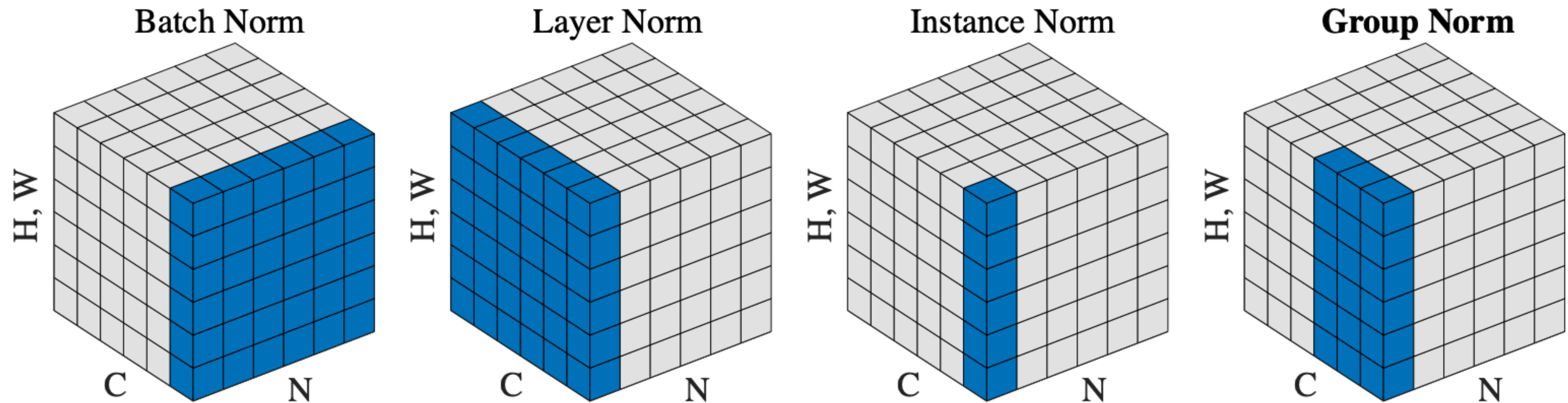
- I don't know



“The title is also partly a joke — no-one really understands deep learning at the time of writing. Modern deep networks learn piecewise linear functions with more regions than there are atoms in the universe and can be trained with fewer data examples than model parameters. It is neither obvious that we should be able to fit these functions reliably nor that they should generalize well to new data.”

Normalisation blocks

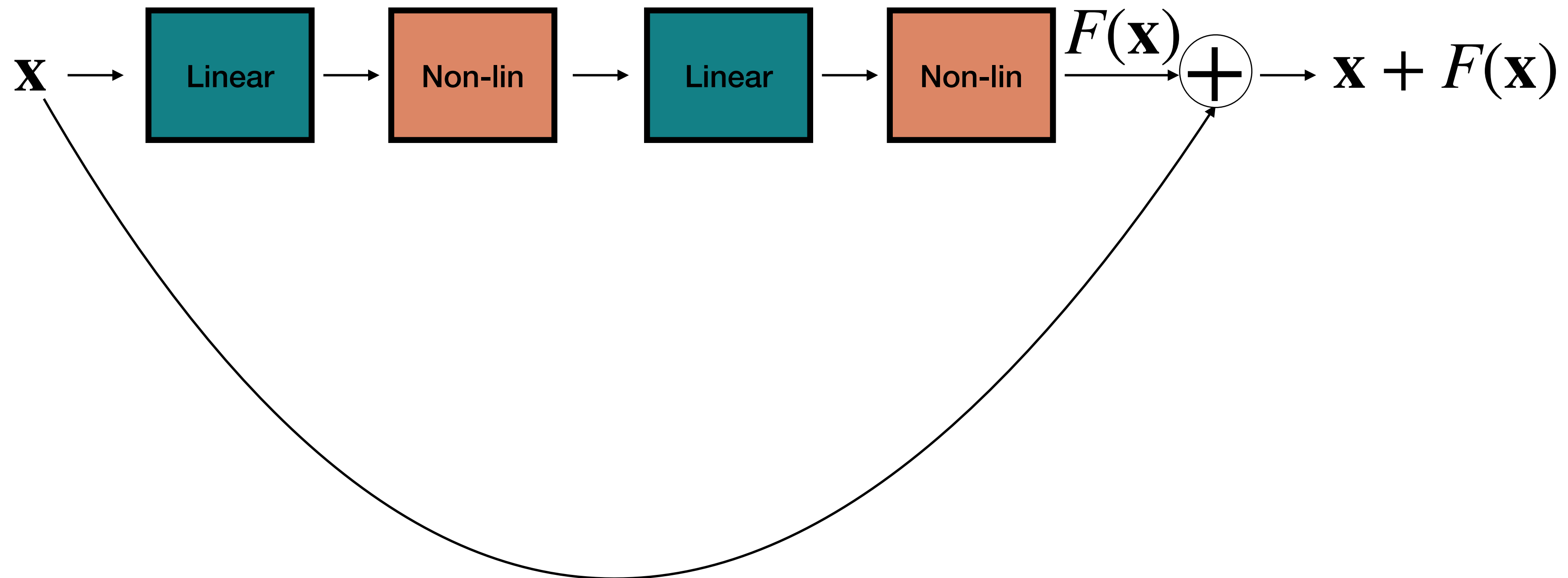
- These effectively standardise pre-activations across a mini-batch
- This prevents e.g. everything from being switched off or on by a ReLU



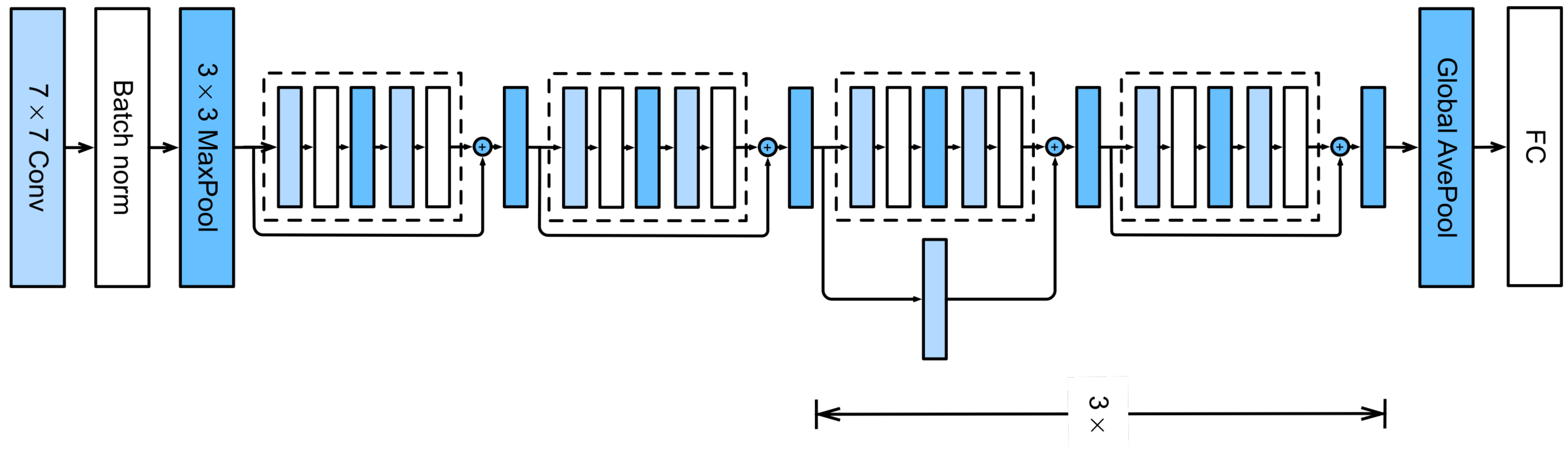
<https://arxiv.org/pdf/1803.08494>

Residual connections

- Deeper networks tend to be better but are hard to train
- Putting residual connections throughout networks remedies this
- These help prevent gradients from vanishing to zero

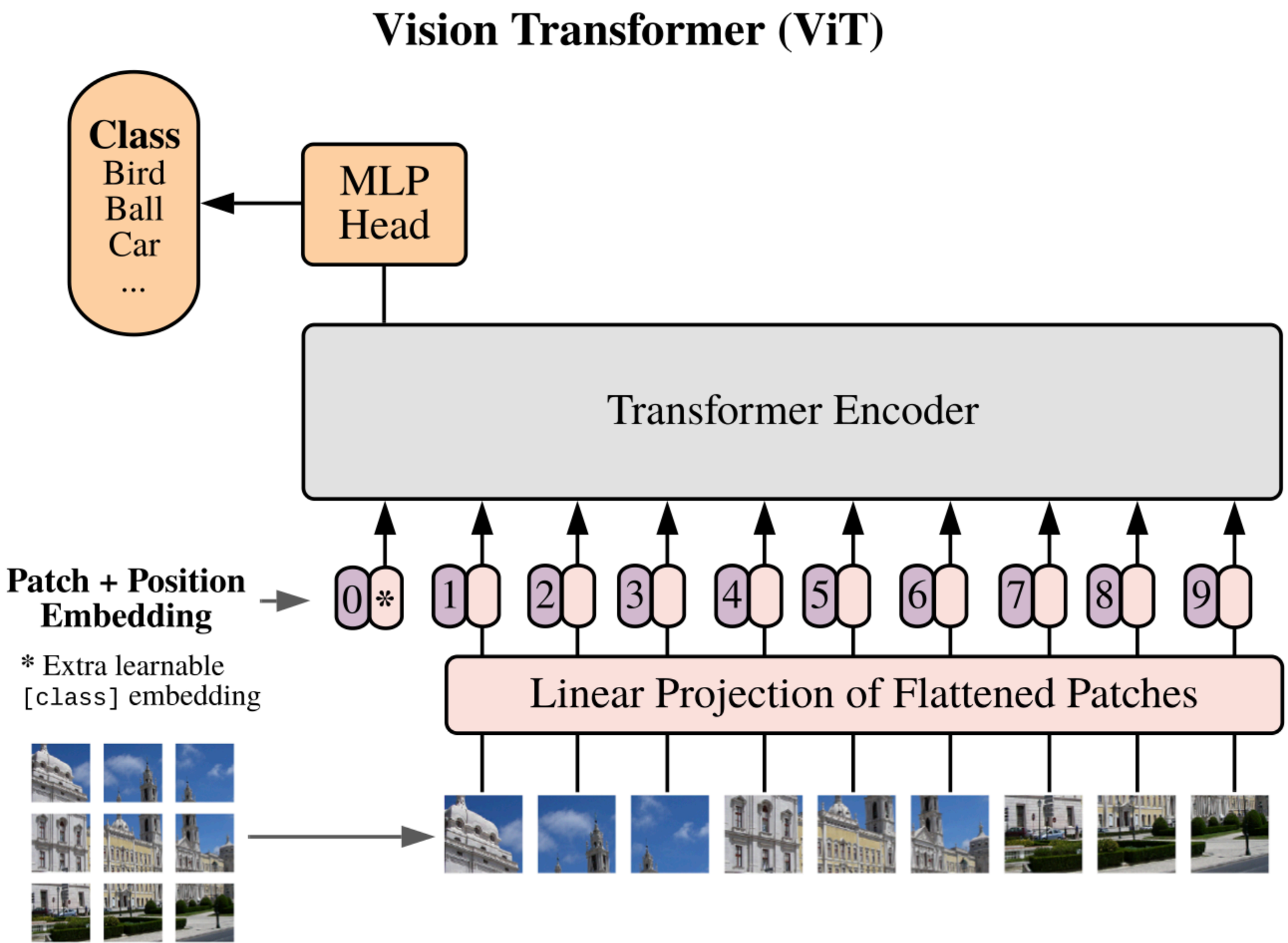
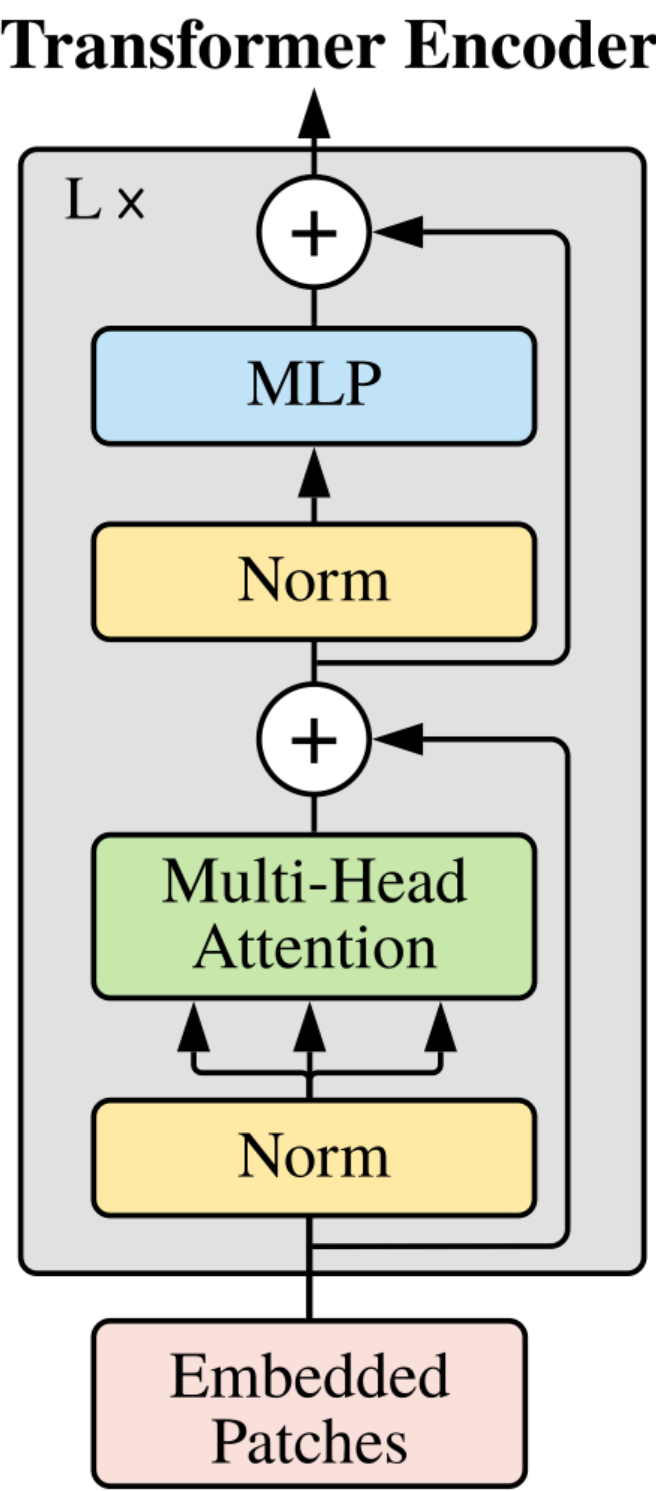


Old reliable: The ResNet



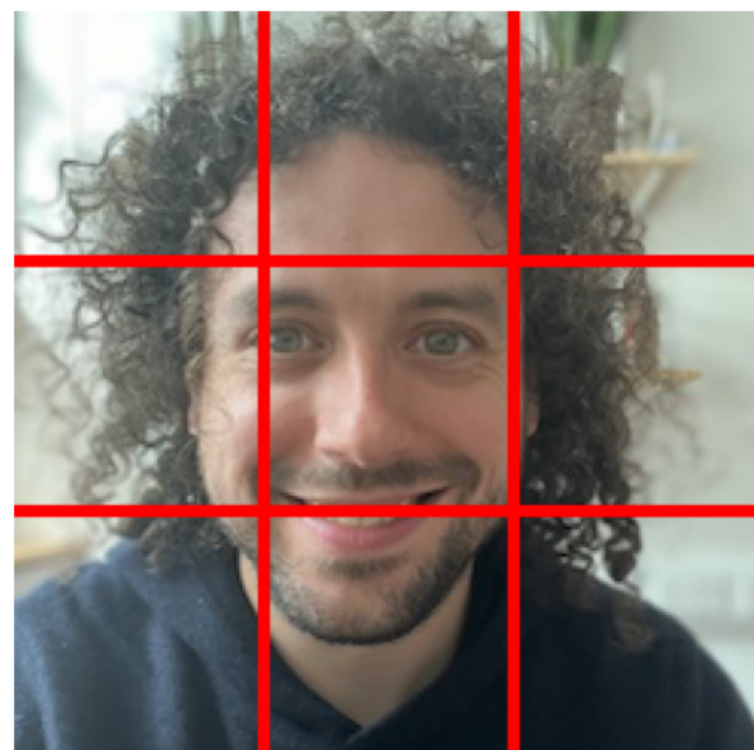
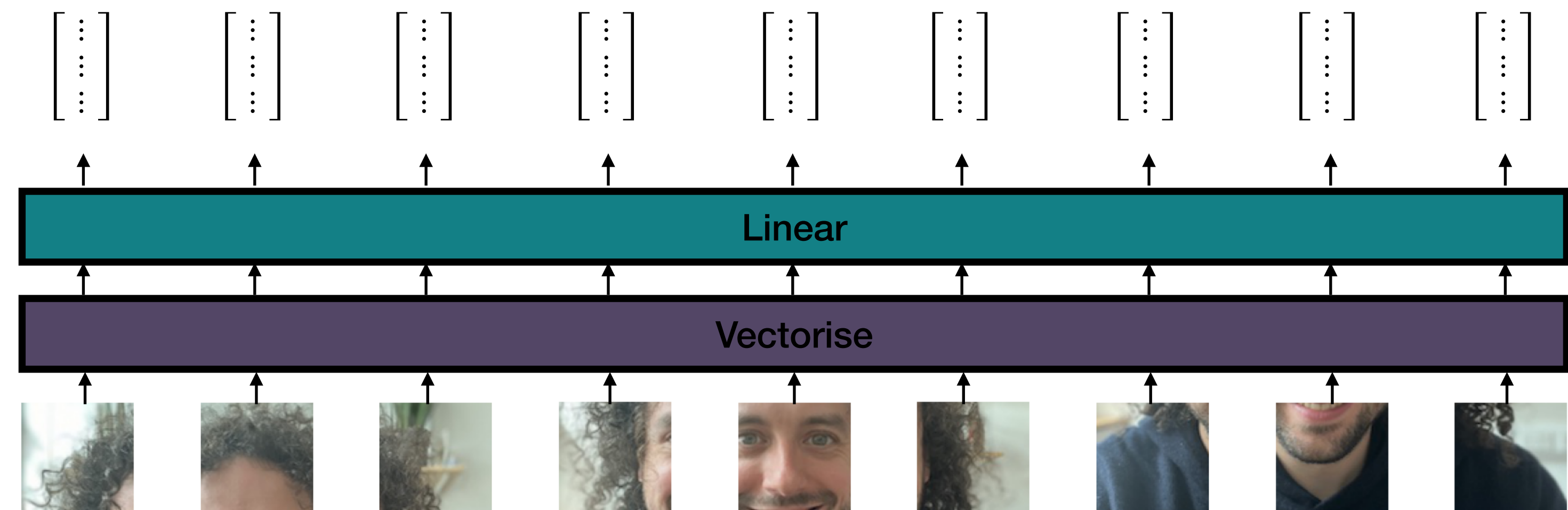
New reliable: The transformer

Specifically, this is a vision transformer as the inputs are images

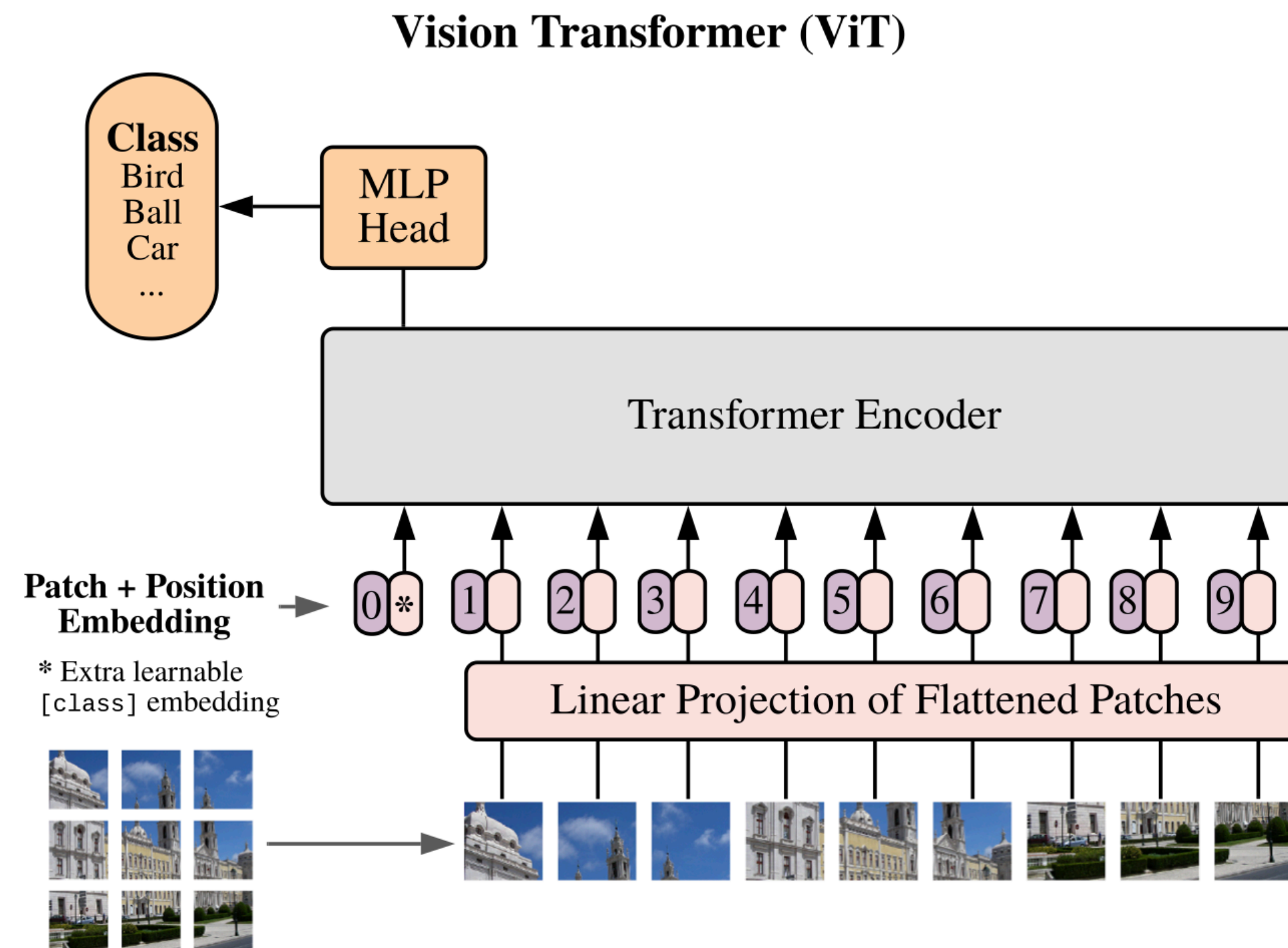
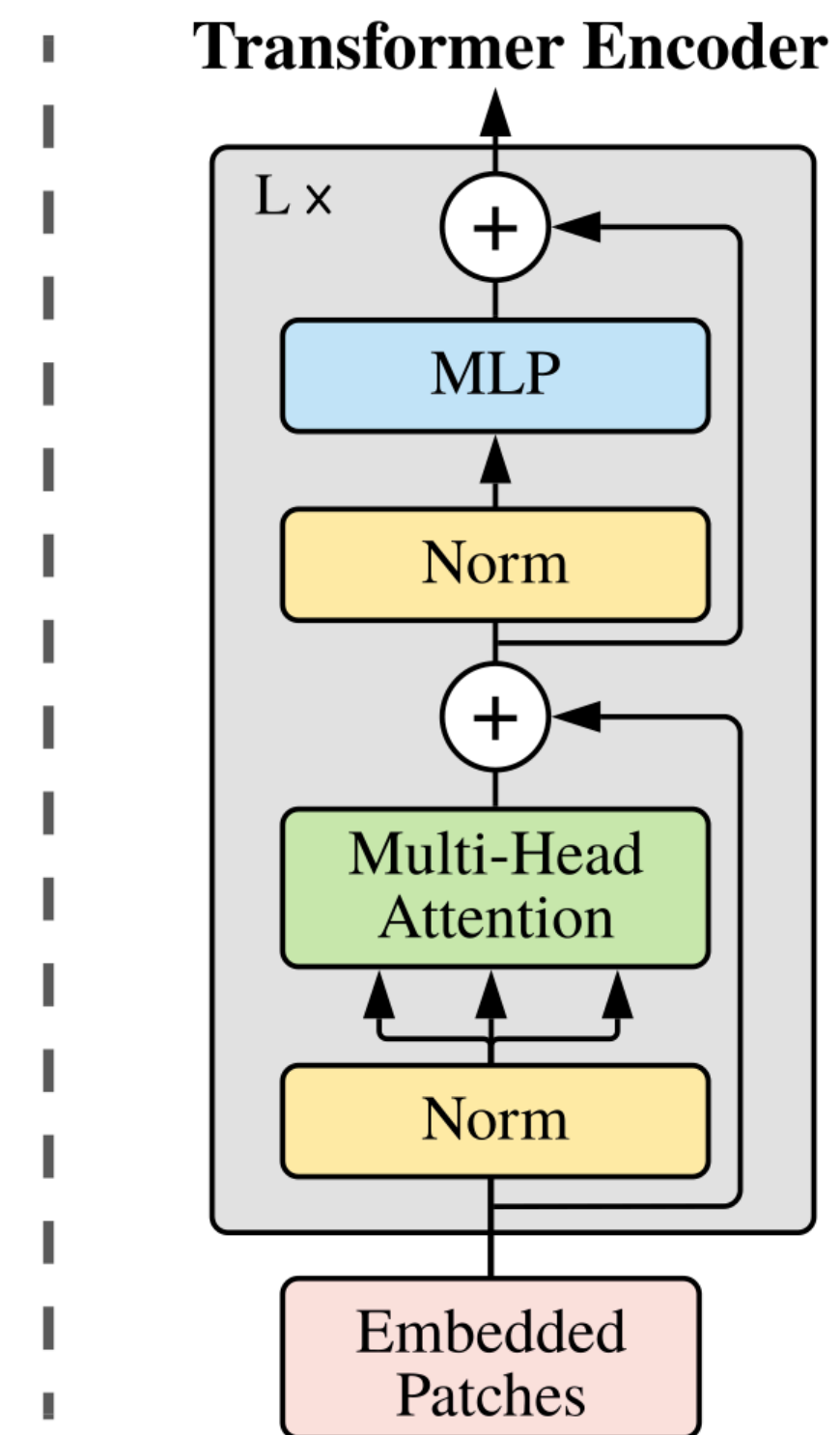


Tokens

- In a transformer block, each data point is represented by a sequence of vectors called **tokens**
- In the “stem” of a vision transformer, we split an image into patches, vectorise, and pump these through a learned linear layer to produce tokens
- These are then typically combined with positional information before entering a series of transformer blocks



What don't we know about (yet)?



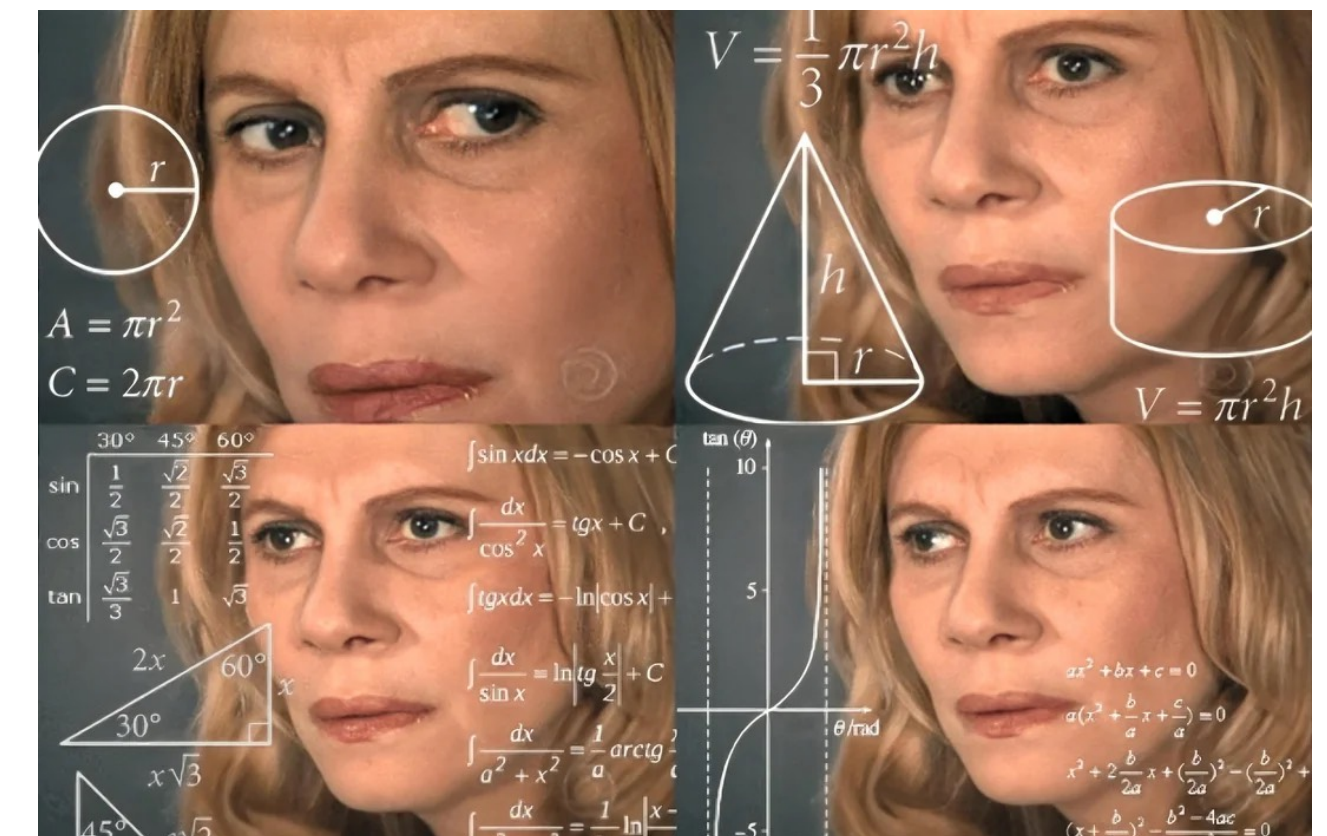
Attention

- Convolutions pick up local information i.e. patterns of things next to each other
- In an image, useful evidence can be far apart e.g. feet and head for the task of finding a human
- Attention lets us relate distant (or near!) patches to each other



(Self) Attention - the maths

- Consider a token representation of a single data point $\mathbf{X}_{in} \in \mathbb{R}^{T \times d}$ where there are T tokens each of dimensionality d
- Attention computes $\mathbf{X}_{out} = S\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$
- Here, $\mathbf{Q} = \mathbf{X}_{in}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{X}_{in}\mathbf{W}_K$, $\mathbf{V} = \mathbf{X}_{in}\mathbf{W}_V$ and $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ and S is the softmax function

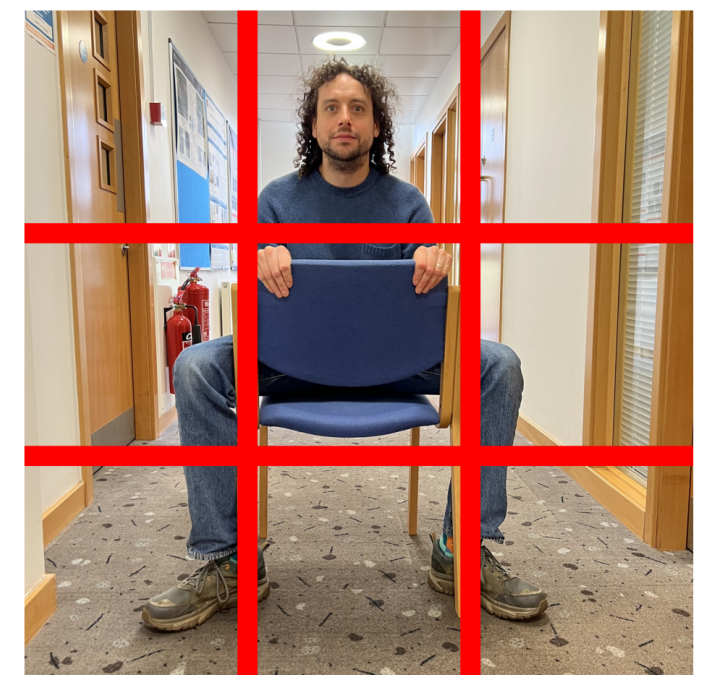


(Self) Attention

- $\mathbf{X}_{out} = S(\frac{\mathbf{QK}^T}{\sqrt{d_k}})\mathbf{V}$

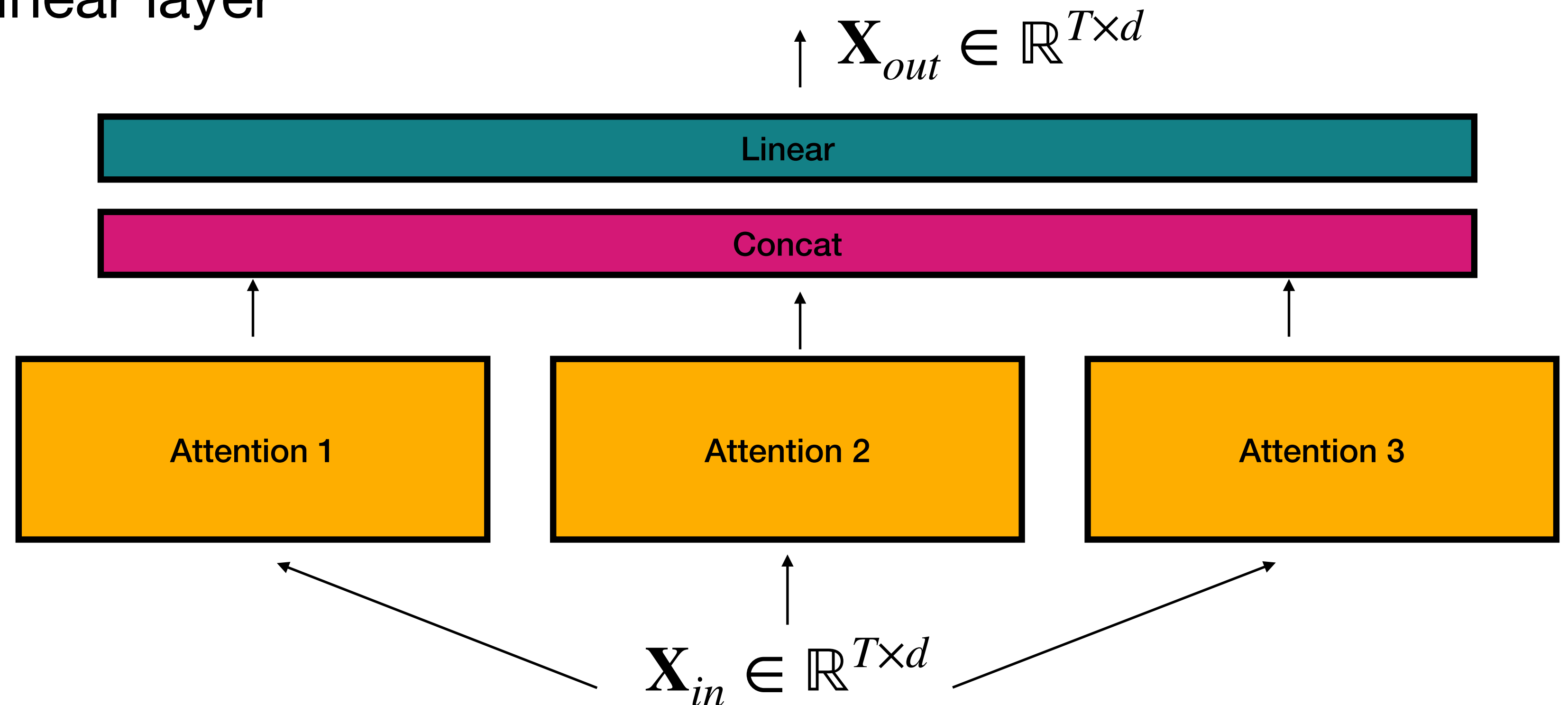
Q is a matrix of queries, **K** is a matrix of keys, **V** is a matrix of values.
These are all just linearly transformed tokens

- $\mathbf{QK}^T$ is a similarity matrix between queries and keys
- The softmax “picks” the most similar tokens for each token (including itself)
- Each new token then just becomes some weighted sum of values
- This allows tokens to mix regardless of location



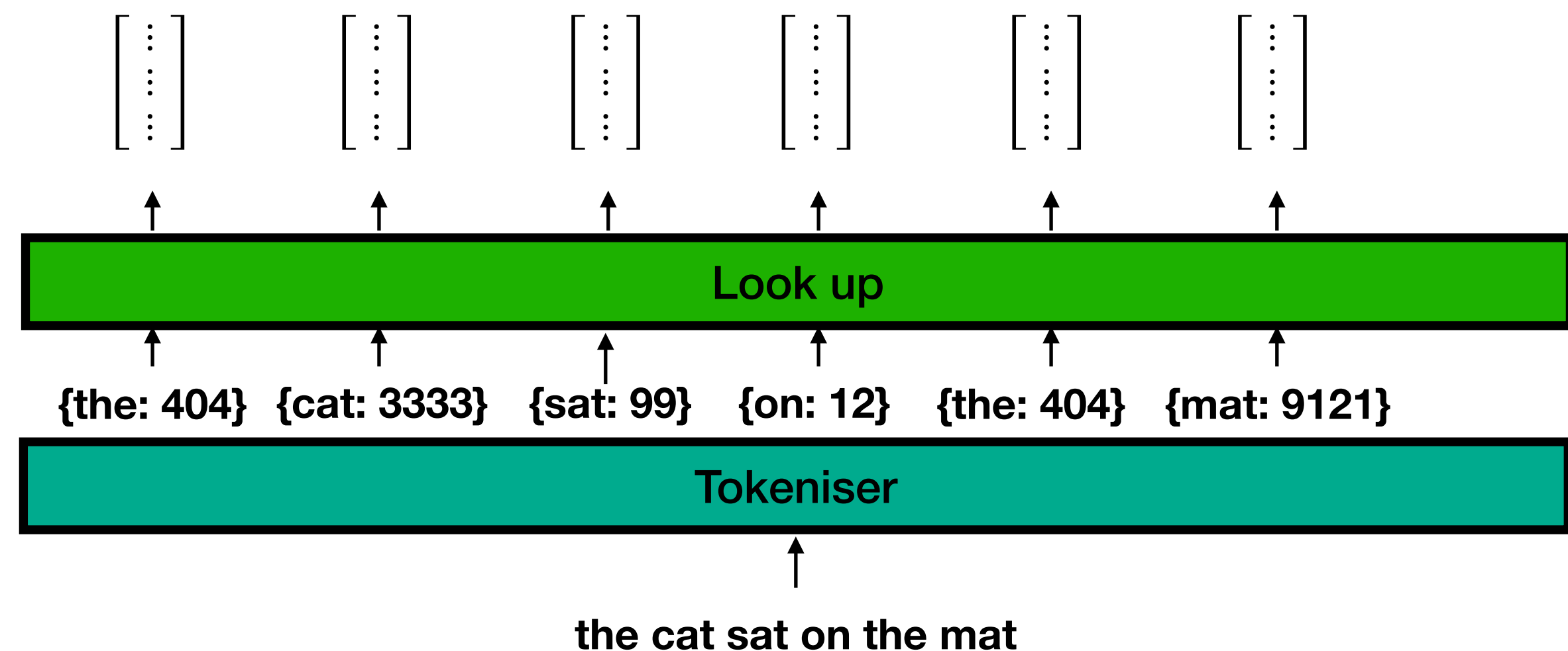
Multi-headed attention

- This comprises several attention operations (with their own queries, values, keys) in parallel. Each “head” can focus on different relationships
- The head outputs are concatenated and then squeezed back to the token dimension through a linear layer



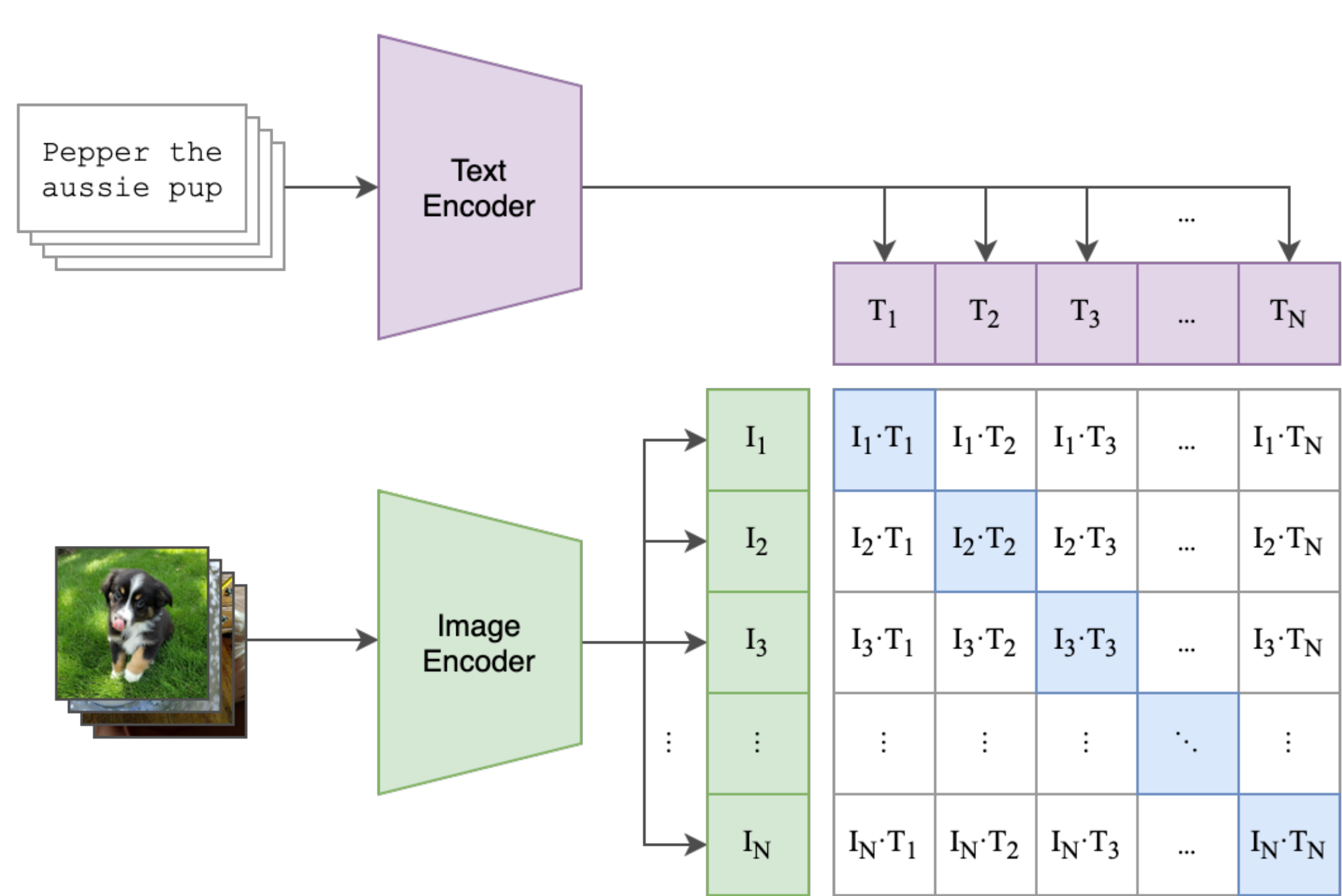
Transformers for text

- Practically all mainstream LLMs are transformers
- A text input is converted into token IDs through a **fixed** tokeniser
- Each token ID corresponds to a **learned** token embedding
- The input to the transformer block is a sequence of these embeddings combined with positional information

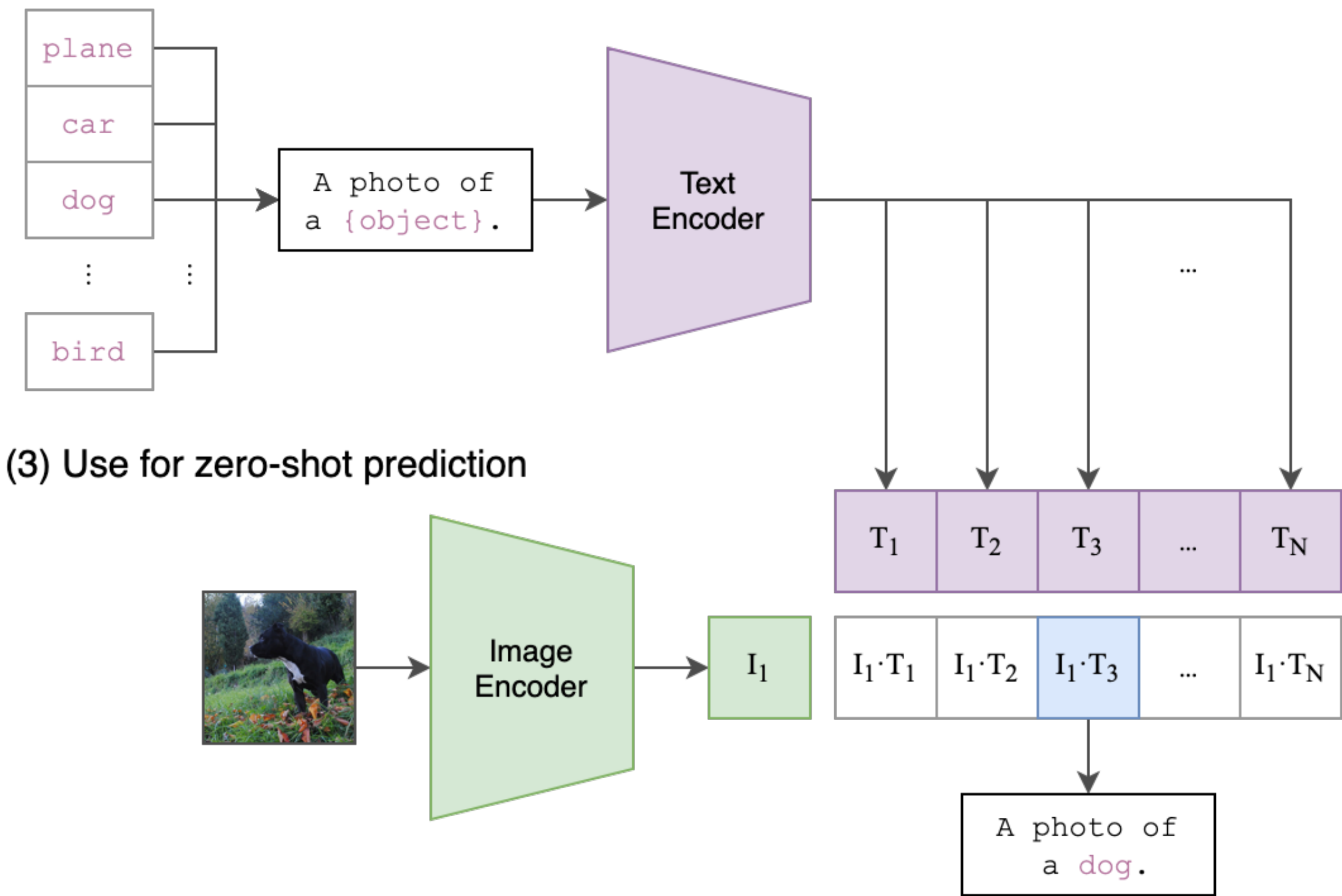


We can combine text and images if we compare in “token” form

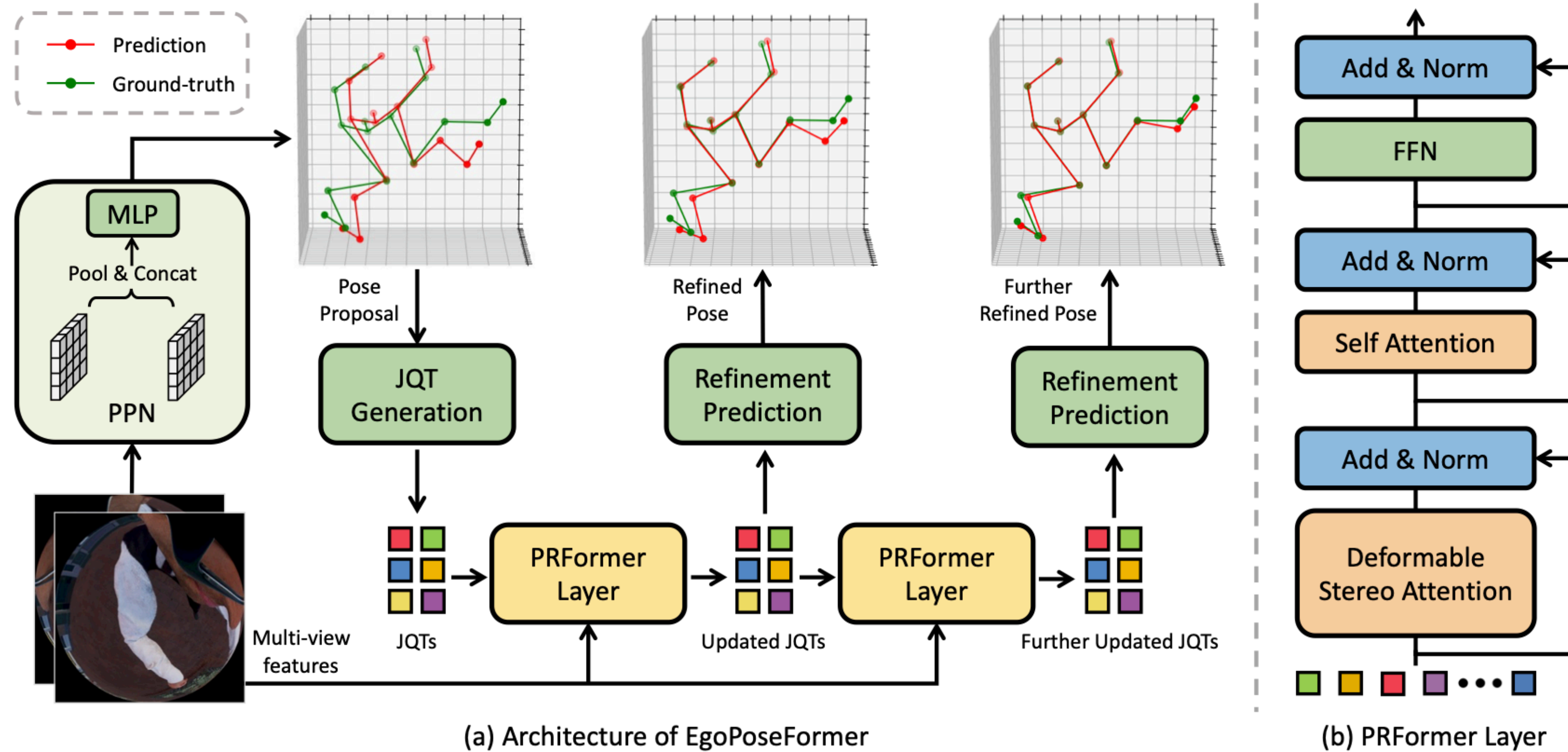
(1) Contrastive pre-training



(2) Create dataset classifier from label text



Attention lets us fuse different modalities



A big network circa 2018

- My “big” network experiment from a 2018 paper
- These were trained on over 1 million images across 4-8 GPUs

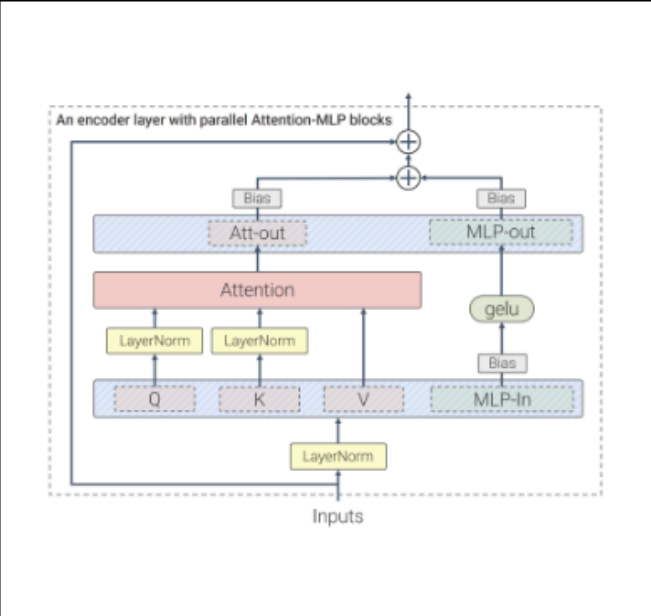
Model	Params	Mult-Adds	Scratch		AT	
			Top 1	Top 5	Top 1	Top 5
Res34 T	21.8M	3.669G	26.73	8.57	–	–
Res18	11.7M	1.818G	30.36	11.02	29.18	10.05
Res34-G(4)	8.1M	1.395G	26.61	8.62	26.58	8.43
Res18-0.5	3.2M	909M	36.96	15.01	37.20	15.02
Res34-G(N)	3.1M	559M	32.98	12.26	30.16	10.66

The meaning of “big” has changed

[Home](#) > [Blog](#) >

Scaling vision transformers to 22 billion parameters

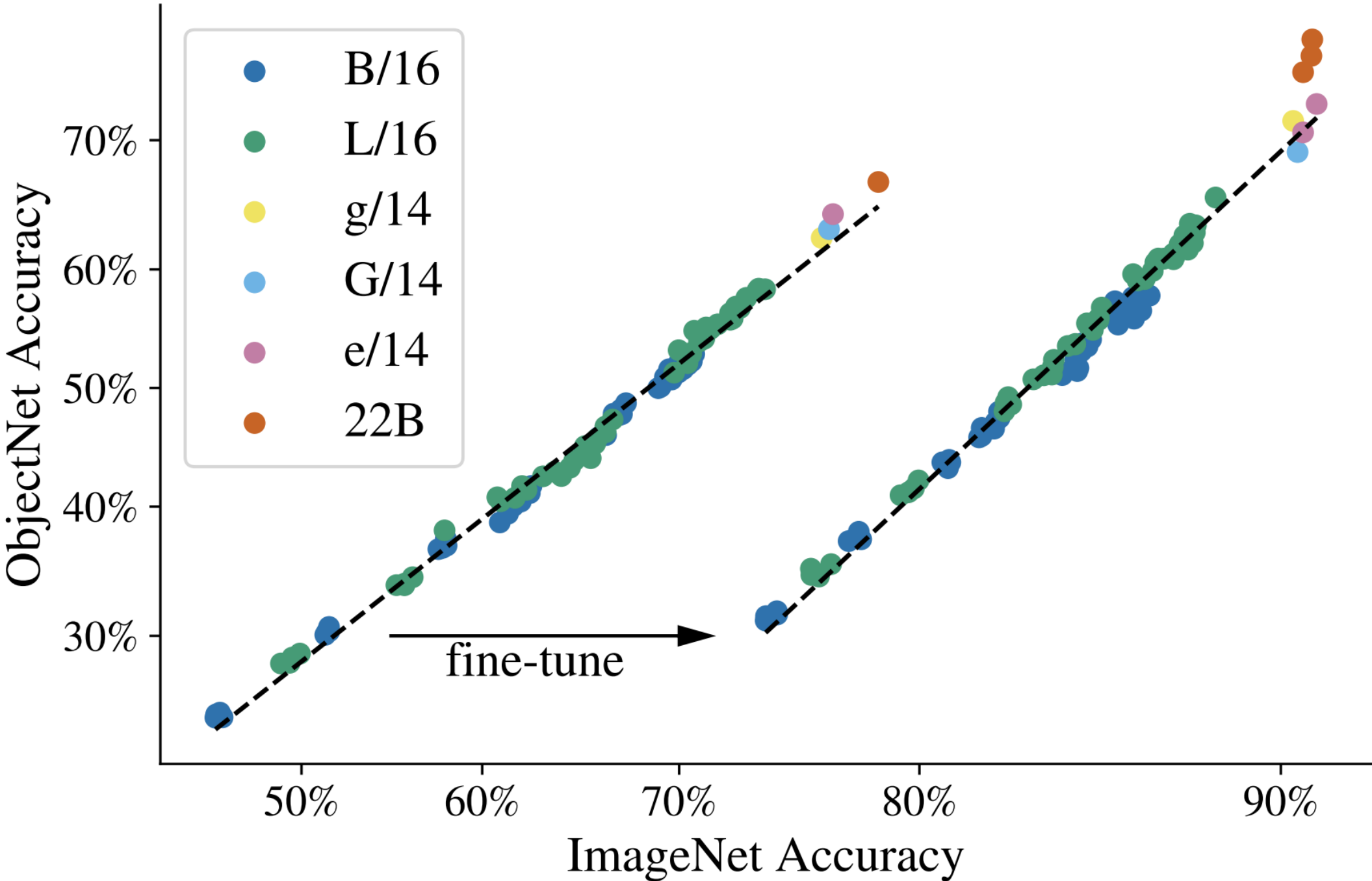
March 31, 2023 · Posted by Piotr Padlewski and Josip Djolonga, Software Engineers, Google Research



Large Language Models (LLMs) like [PaLM](#) or [GPT-3](#) showed that scaling transformers to hundreds of billions of parameters improves performance and [unlocks emergent abilities](#). The biggest dense models for image understanding, however, have reached only 4 billion parameters, despite research indicating that promising multimodal models like [PaLI](#) continue to benefit from scaling vision models alongside their language counterparts. Motivated by this, and the results from scaling LLMs, we decided to undertake the next step in the journey of scaling the [Vision Transformer](#).

QUICK LINKS

[Share](#)



Train from scratch or adapt?

- Modern neural networks can be extremely expensive to train
- Most people adapt existing “foundational models” developed by larger entities

Carbon Footprint of the Whole Project In order to compute the carbon footprint of the whole project, we use a rough estimate of a total 9M GPU hours. Using the same grid parameters as presented above, we estimate the total footprint to be roughly 2600 tCO₂eq. For comparison, a full Boeing 777 return flight between Paris and New York corresponds to approximately 560 tCO₂eq. Supposing 12 such flights per day, the environmental impact of our project represents half of all flights between these two cities for one day. This estimate only considers the electricity for powering the GPUs and ignores other emissions, such as cooling, manufacturing, and disposal.

DINOv3 was trained on over 1 billion images

Table 20: Carbon footprint of model training. We report the potential carbon emission of reproducing a full model pre-training, computed using a PUE of 1.1 and carbon intensity factor of 0.385kg CO₂eq/KWh.

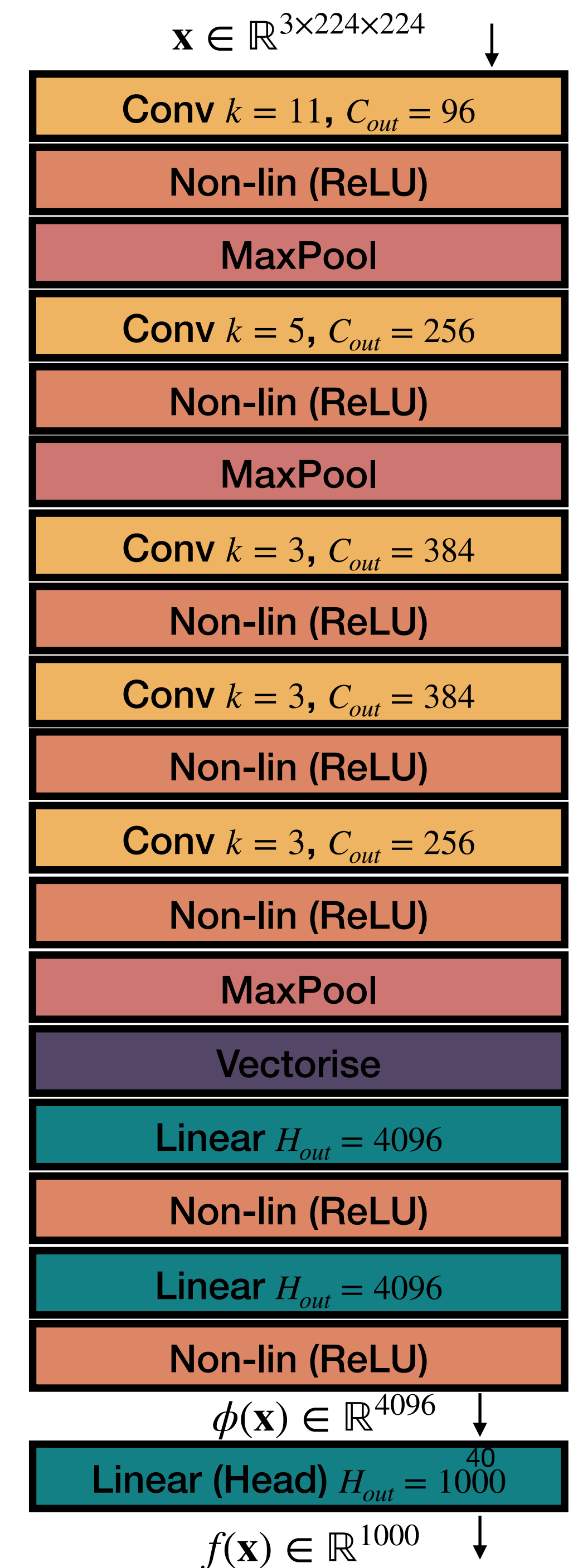
Model	Arch.	GPU type	Power (W)	Steps	GPU hours	PUE	Total power (MWh)	Emission (tCO ₂ eq)
MetaCLIP	ViT-G	A100-40GB	400W	390k	368,640	1.1	160	62
DINOv2	ViT-g	A100-40GB	400W	625k	22,016	1.1	9.7	3.7
DINOv3	ViT-7B	H100-SXM5	700W	1,000k	61,440	1.1	47	18

Feature extraction

- Our network maps images $\mathbf{x}$ to features $\phi(\mathbf{x})$ where we are able to linearly separate out 1000 different classes
- These features are **good**! We can use them elsewhere
- We can produce features for new images and then e.g. fit some linear model for a new task
- The same can be done for text with a pre-trained transformer

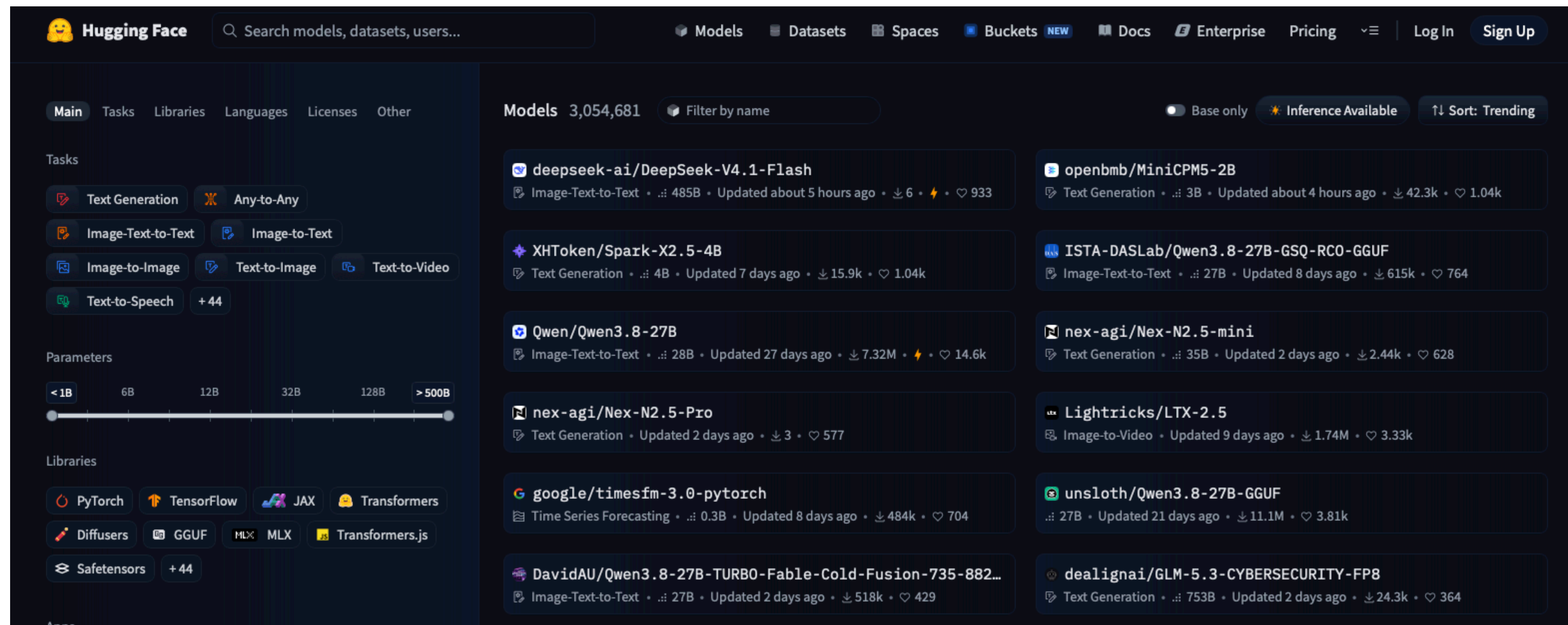


https://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_full_1k.jpg



Pre-trained networks

- There are loads of pre-trained networks available on the internet



<https://huggingface.co/models>

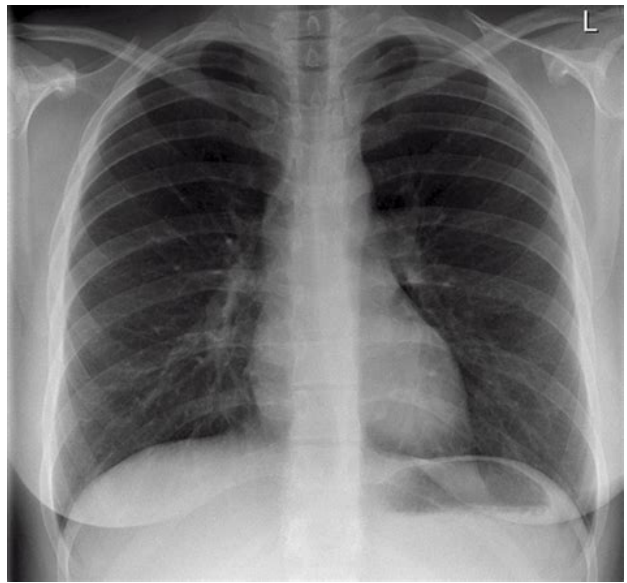
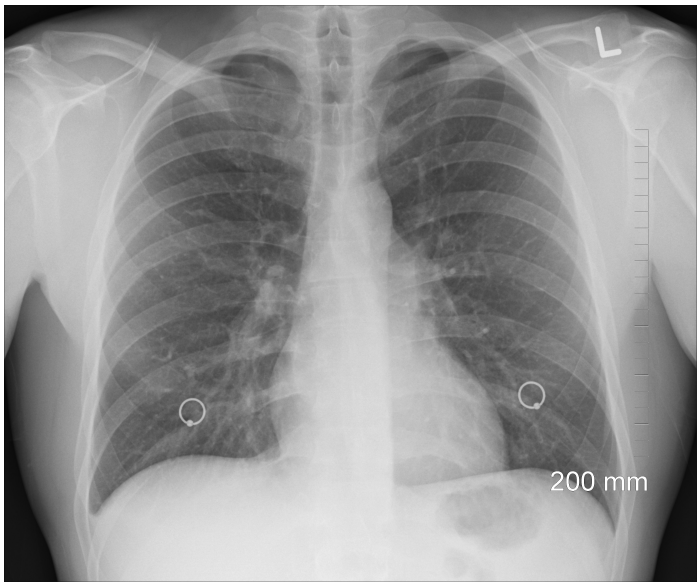
Domain shift



Is the dog in the photo wearing a hat?

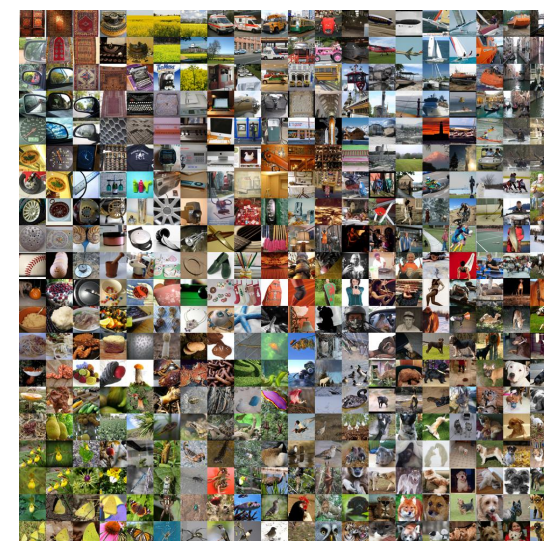
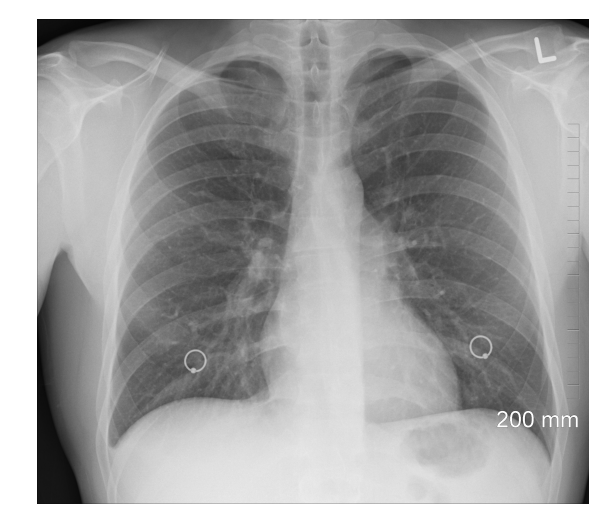


Is the lung in the X-ray cancerous?



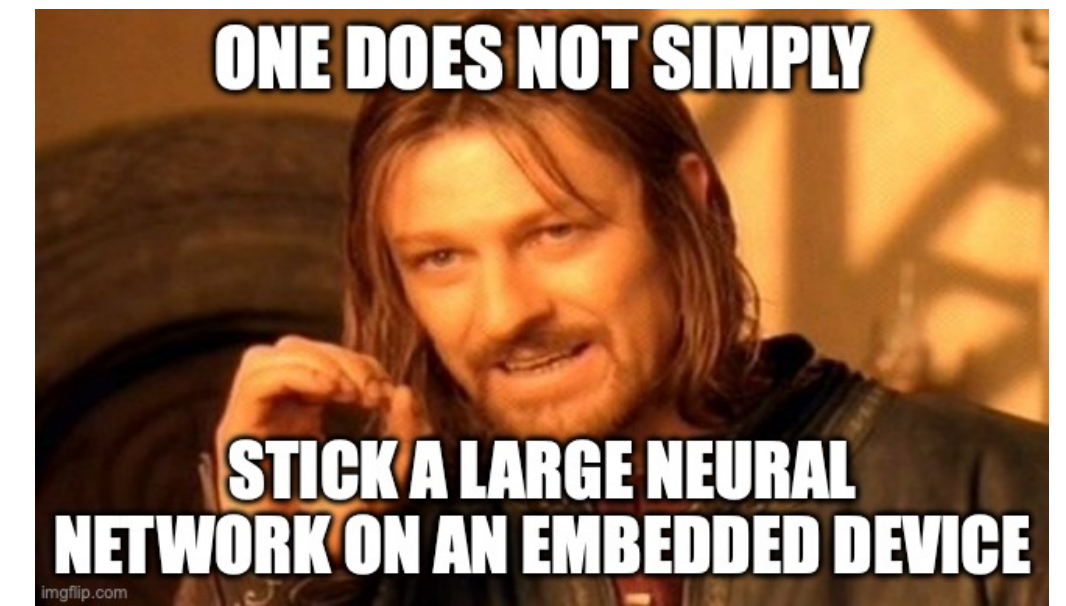
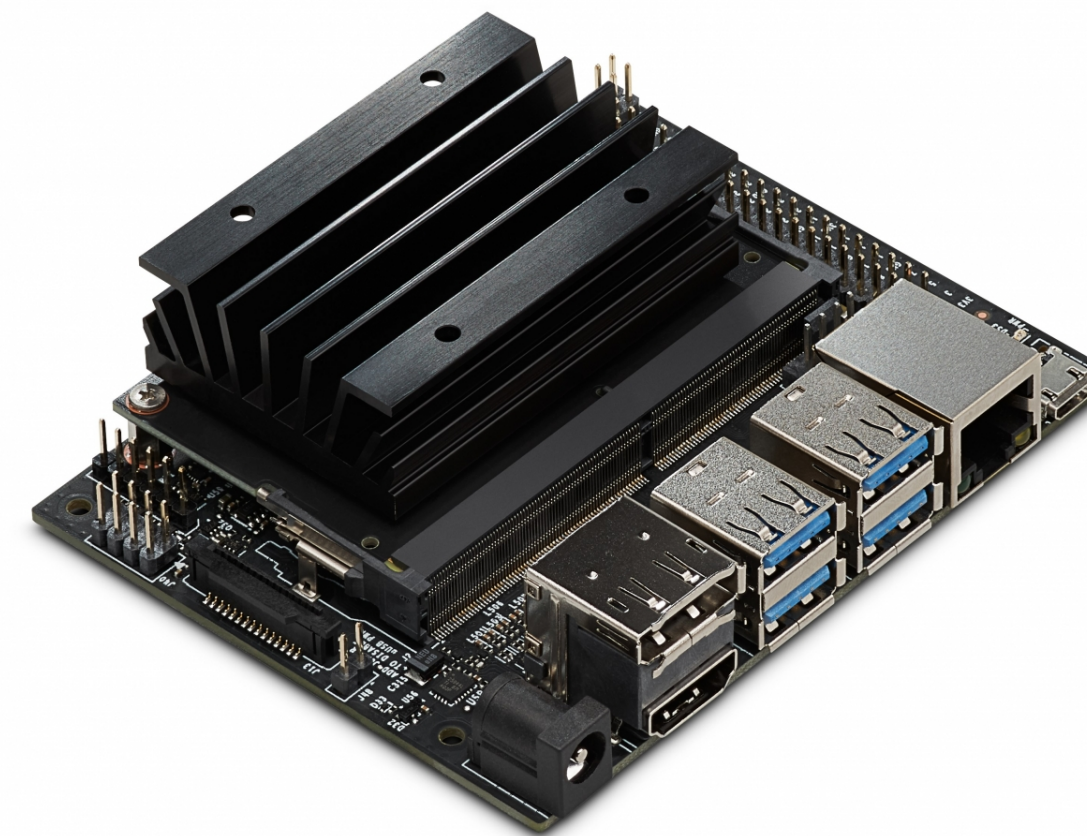
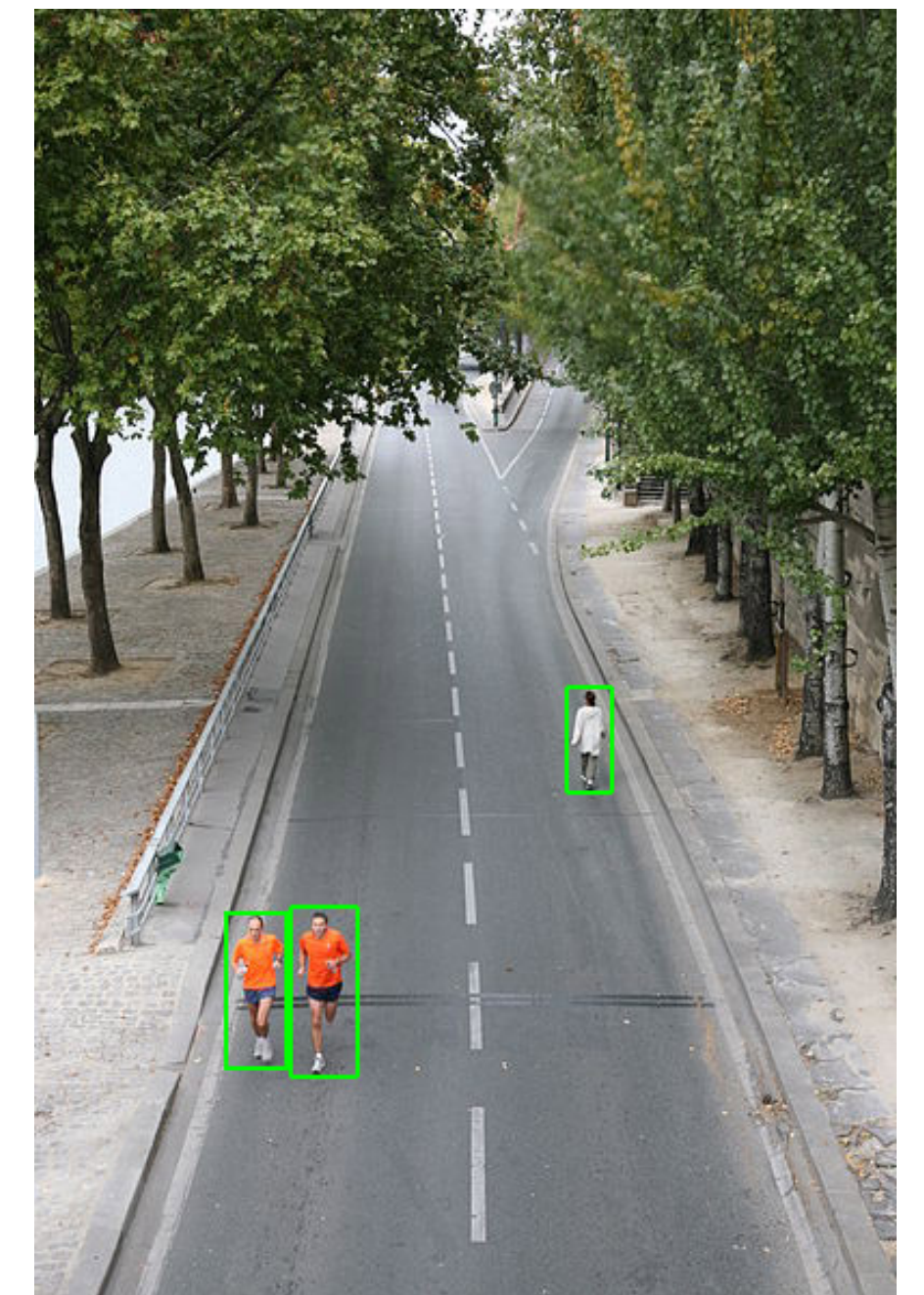
Fine-tuning

- Fine-tuning is when we further train a pre-trained network on a different (often smaller) dataset
- This is typically done with a low learning rate, and with a different (task-dependent) classification head
- This lets the model adapt its features to the task, which is useful when there is domain shift
- It's usually much cheaper than training from scratch: you typically need less data and less time/compute



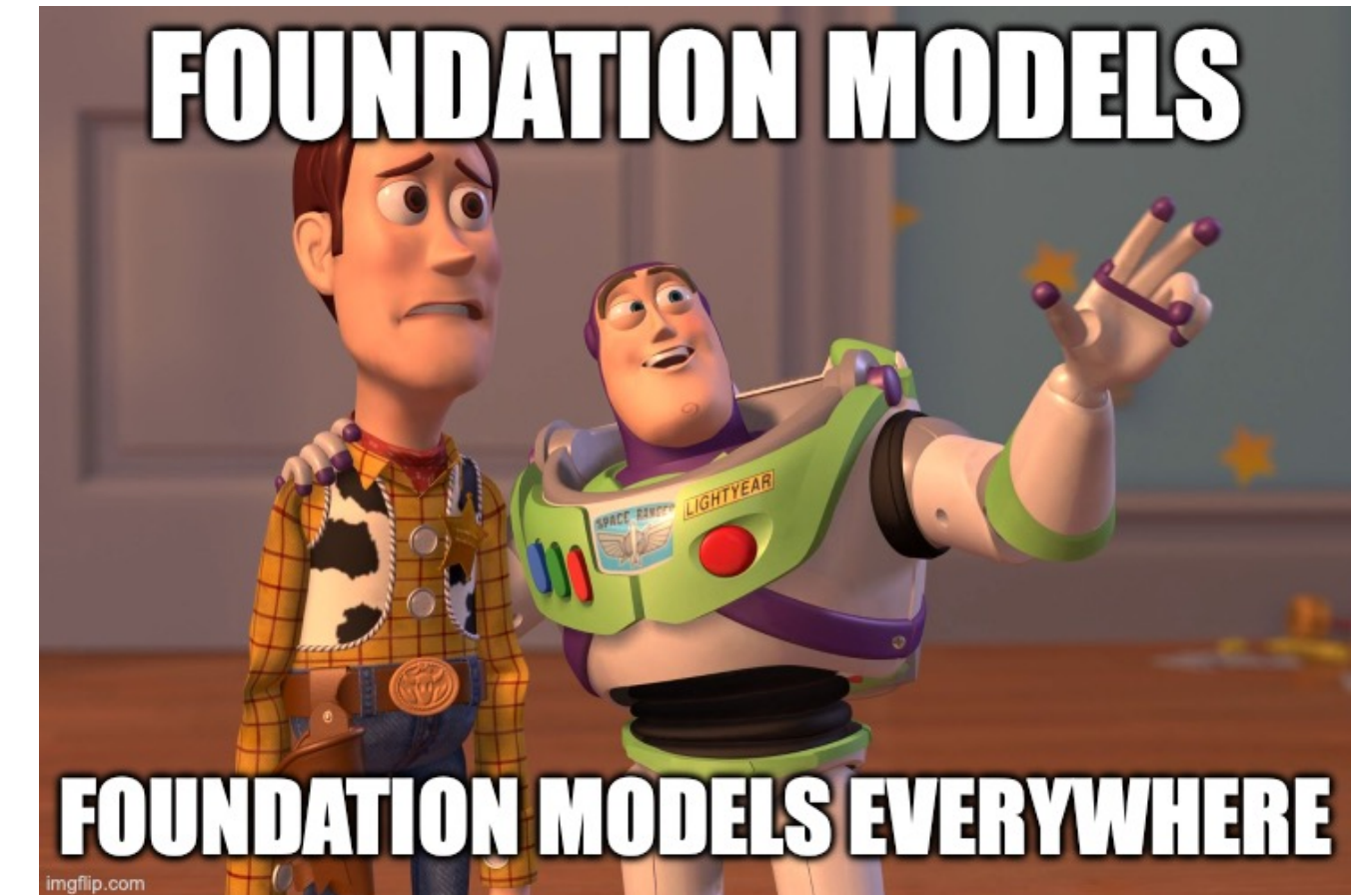
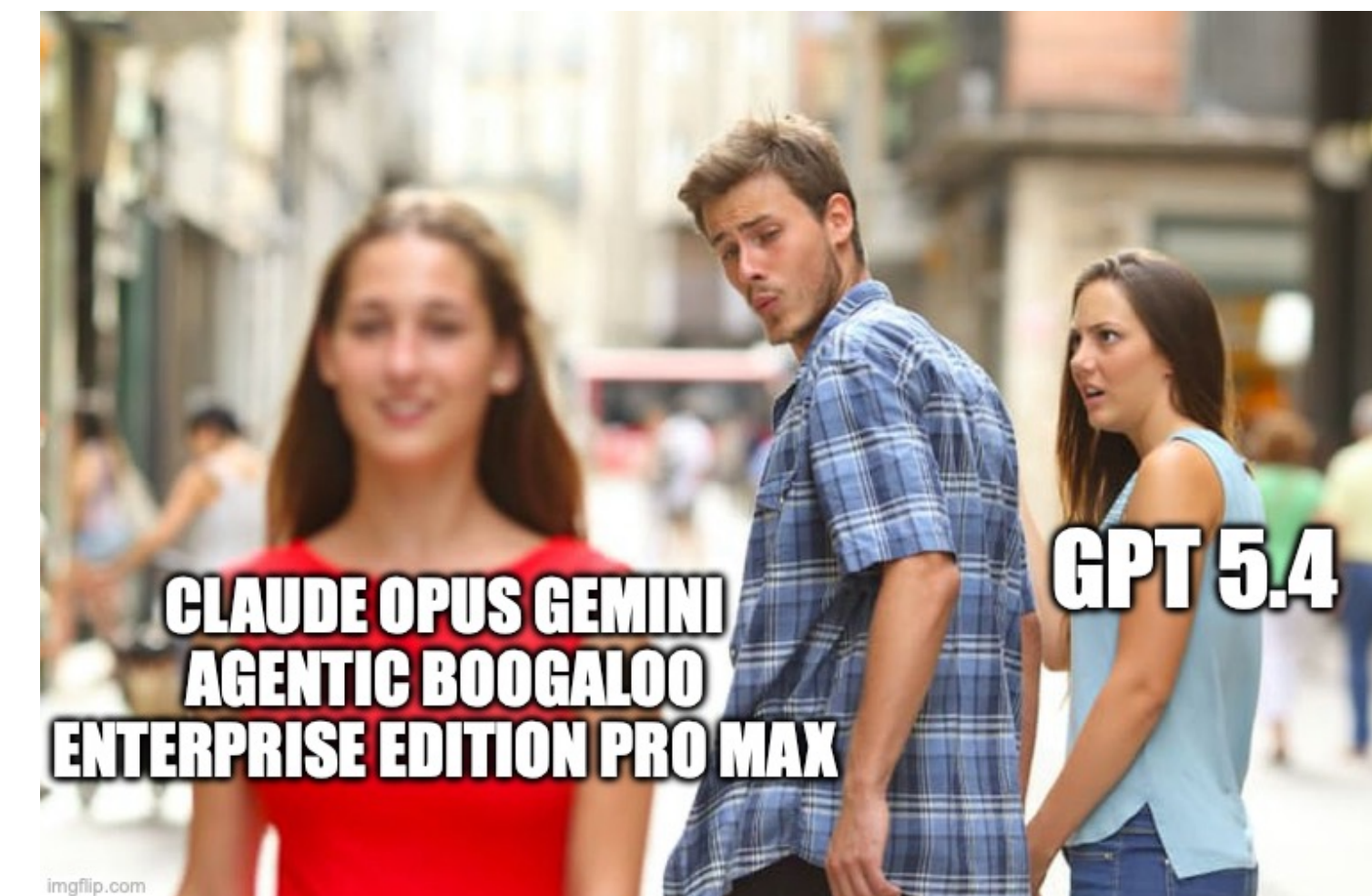
Efficient models are needed

- Using less energy = good
- You may have resource-constrained hardware e.g. for edge AI
- You may have a task that has to be done in limited time
- How can we develop small specialised networks?



Automated Machine Learning is needed

- There are loads of models and design choices
- Given a specific problem and constraints, how can we automate the search for a good solution?
- How can we do this efficiently?



New networks are out there!



The end (of the lectures)

- You have visualised and analysed data
- You have considered the ethical implications of deploying ML in society
- You have learnt about linear models for classification and regression
- You have learnt about non-parametric and non-linear models
- You have written code to use these models



Questions

Q1

A convolutional layer takes in a 32×32 RGB image and uses 16 filters of kernel size 5×5 . There is no input padding and a stride of 1.

What size is the output? How many trainable parameters does this layer have? Assume each filter has a bias.

Q2

A Vision Transformer takes in a 224×224 RGB image. The image is split into non-overlapping patches of size 16×16 , and each patch is flattened and linearly projected to an embedding dimension of 768.

How many patches are produced, and what is the shape of the token sequence after projection?

How many trainable parameters are in the patch-embedding projection (assume it has a bias)?

Q3

- Explain what transfer learning is and why it is useful when training a neural network on a small dataset.