Deep Learning Workshop: Applied Computer Vision

Jeremy Pinto, jeremy.pinto@mila.quebec

Pierre-Luc St-Charles, pierreluc.stcharles@mila.quebec

Session 3

Course Structure

The course is spread over **4 sessions**, with a mix of theory and practical content.

Session 1: Theory	Session 2: Practical	Session 3: Theory	Session 4: Practical
Introduction to computer vision and deep learning concepts	Implementation of a classification algorithm on a rock, paper, scissors dataset	Review of state-of-the-art models, vision transformers, object detection	Overview of an applied project using modern tools and libraries and best practices

Contents

- Overview of modern CNN architectures
- Vision Transformers
- Fine-Tuning
- Foundation models
- Object Detection
- Deep learning for videos

Architectures

Architectures

So far, we have seen the most common **building blocks** used in CNNs.

There are many ways in which they can be **combined and stacked**. We will look at the common architectures used for in computer vision.

"A robot making deep learning algorithms made out of lego blocks that look like transformers" Generated with stable diffusion

Benchmarking

Is a “better” model:

- More accurate?
- Lightweight?
- Fast?
- Easily Trainable?

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8

https://keras.io/applications/

Comparing Models

In the deep learning community, model performance is usually evaluated on **publicly available datasets**. The model with the highest score wins and usually earns a publication.

	A Metric	Another Metric
Our Model	**41.57**	**32.45**
Their Model	37.45	-
Their Other Model	38.74	29.45

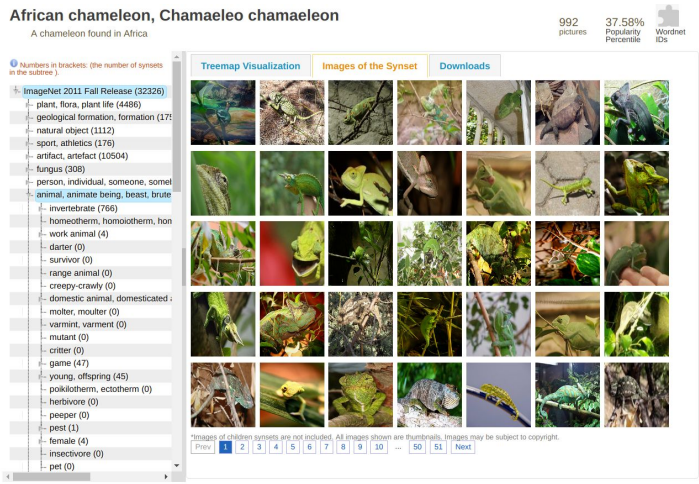
Table 1 - Our model outperforms all other models and is definitely the best model.

Mila

ImageNet

ImageNet is one of the most **widely used benchmarks** in image classification.

It has paved the way for many important discoveries in CNN architectures. It consists of **millions of images** manually labelled with thousands of categories.



http://imagenet.stanford.edu/

ImageNet history

ImageNet

Yet another advice: don't get fooled by people who claim to have a solution to Artificial General Intelligence, who claim to have AI systems that work "just like the human brain", or who claim to have figured out how the brain works (well, except if it's Geoff Hinton making the claim). Ask them what error rate they get on MNIST or ImageNet.

82 Reply Share Report Save Follow

Continue this thread →

Source

Image Classification

Model performance is evaluated on a **standardized test set**. Top-1/top-5 accuracy are usually reported.

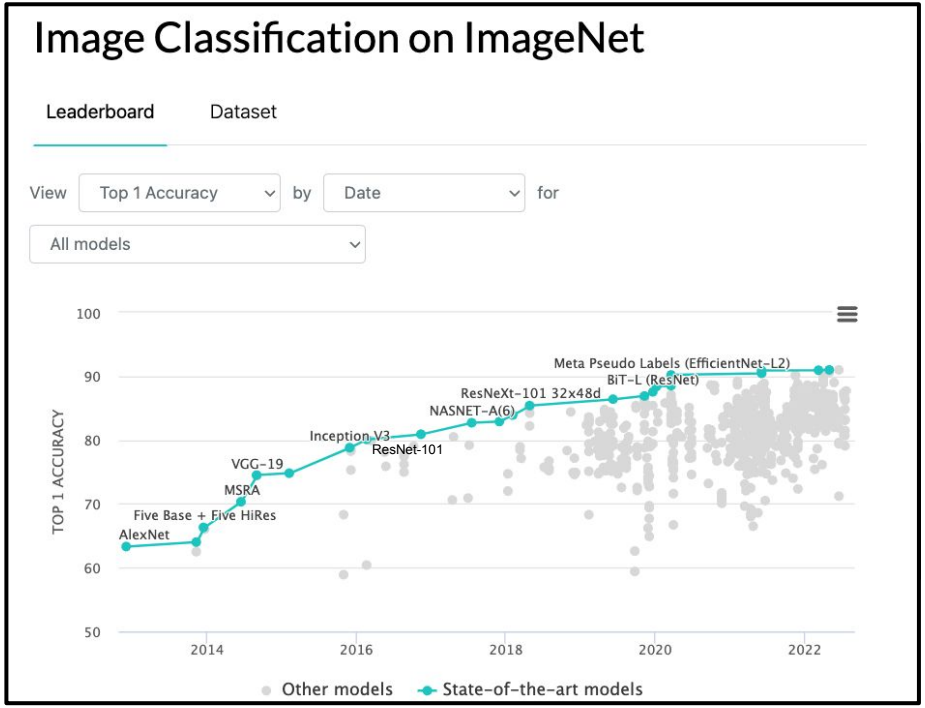
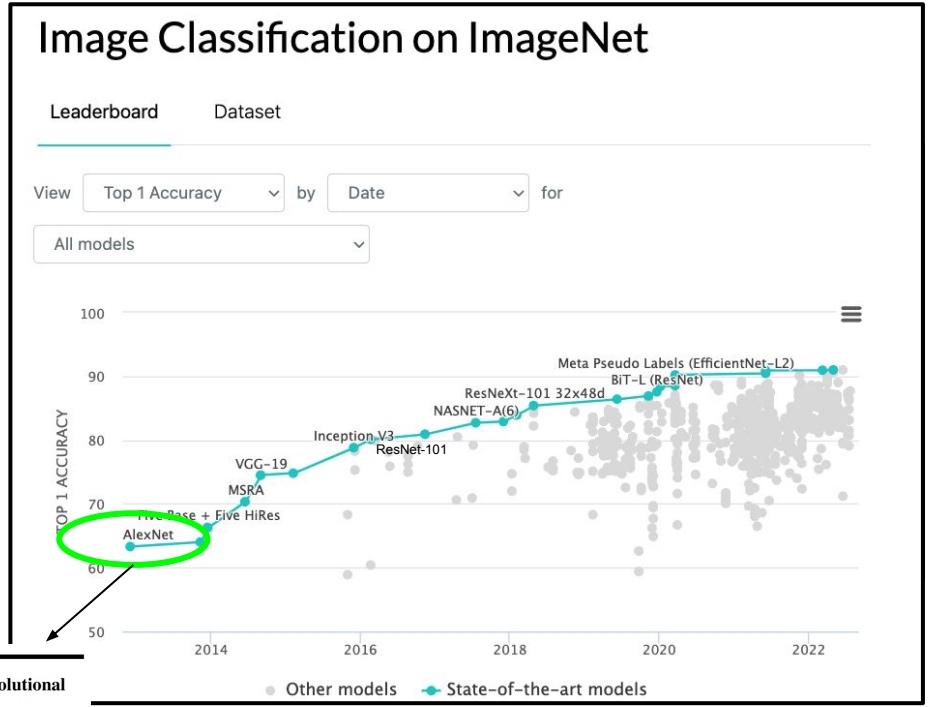
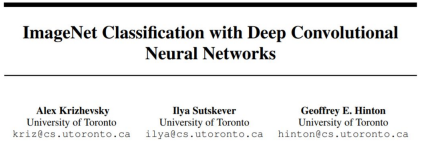


Image Classification

AlexNet was a **turning point** for deep learning research.

They were among the first to use **GPUs** for computation and leverage neural networks on large datasets.





ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

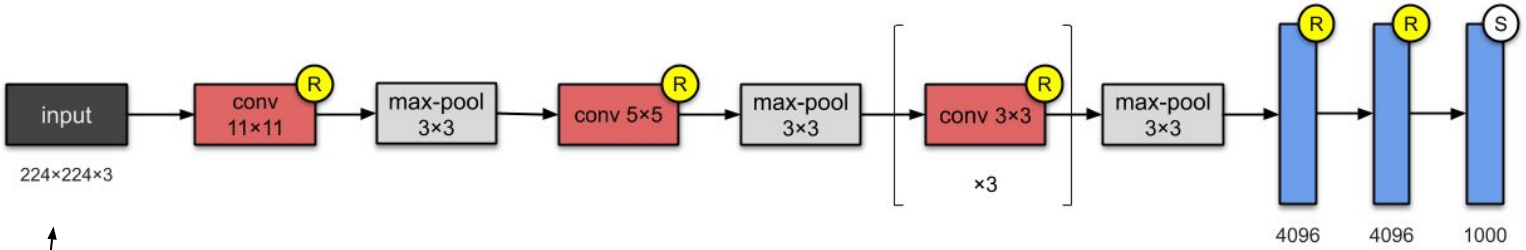
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Paper

AlexNet

AlexNet consists of 5 convolutional layers, ReLU and max-pooling. 2 fully connected (FC) layers followed by a softmax layer generated a probability distribution.



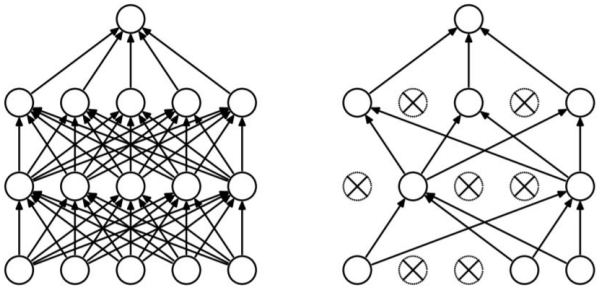
Cropped RGB image

	LeNet	AlexNet
Parameters	~30 x 10^3	~60 x 10^6
Layers	5	8

source

Dropout

AlexNet also used **dropout** in its architecture. This reduces the risk of overfitting and acts as a form of **regularization**. It works by setting weights to 0 with a given probability (~10%).



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

AlexNet

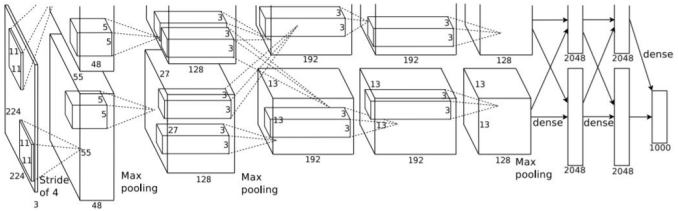
Imagenet classification with deep convolutional neural networks

A Krizhevsky, I Sutskever… - Advances in **neural** …, 2012 - proceedings.neurips.cc

… a large, **deep convolutional neural network** to **classify** the 1.2 million high-resolution images in the **ImageNet** … The **neural network**, which has 60 million parameters and 650,000 neurons, …

☆ Enregistrer Citer Cité 114348 fois Autres articles Les 128 versions »

"All of our experiments suggest that our results can be improved simply by waiting for **faster** GPUs and **bigger datasets** to become available."
- Krizhezvsky et al.



Paper

ImageNet

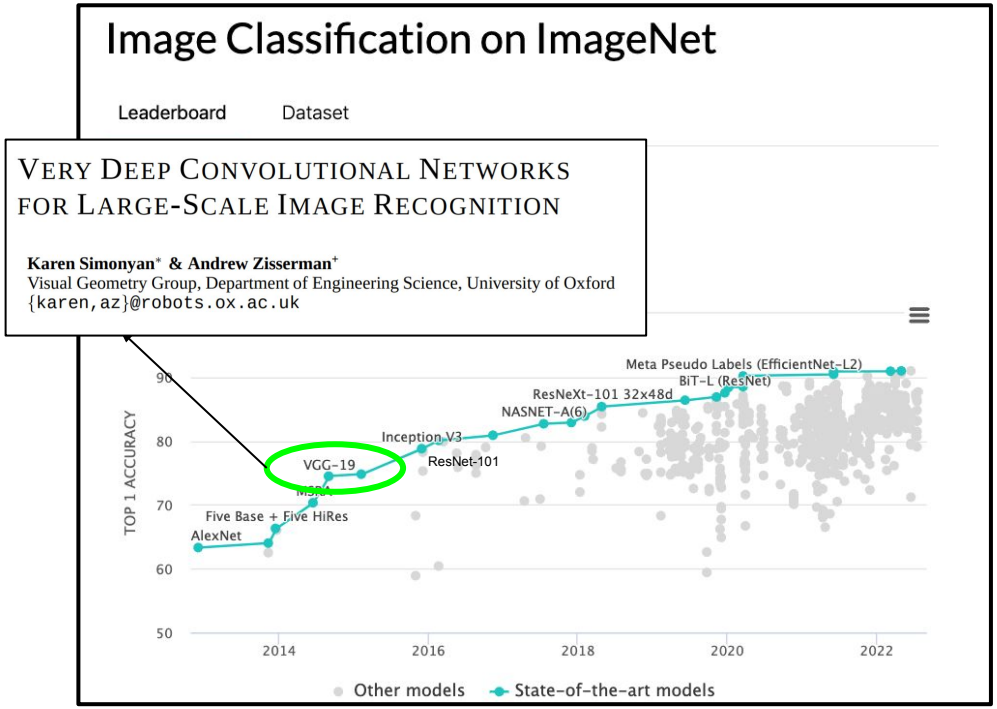
Following the success of AlexNet, researchers attempted to build **deeper networks**.

This was **challenging** both from an engineering perspective (memory issues, long train times) as well as from an **algorithmic perspective** (avoiding overfitting, underfitting, lots of hyperparameters, etc.)

VGG

VGG managed to train deeper networks and obtain better accuracy than previous state-of-the-art.

"Notably, **we did not depart** from the classical ConvNet architecture of LeCun et al. (1989) [LeNet], but improved it by substantially increasing the depth [of the network]." - Simonyan et al.

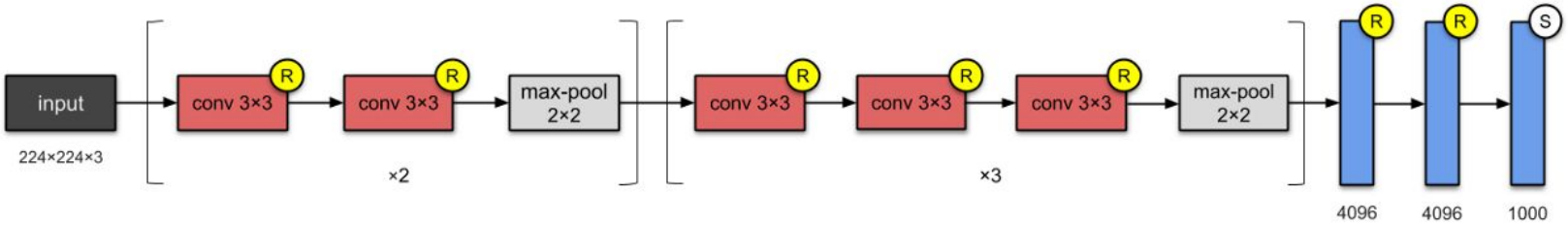


https://arxiv.org/pdf/1409.1556.pdf

VGG

VGG employed a similar structure to AlexNet. It added 2 contributions:

- smaller kernels everywhere (all 3x3)
- more layers (16 layers)



Here we have a diagram for VGG-16. Spatial resolution is preserved after every convolution layer and downsampling only occurs at max-pooling stages.

https://arxiv.org/pdf/1409.1556.pdf

VGG

Here we see the **different configurations** suggested in the original paper. They are very similar in structure with more or less convolutional layers.

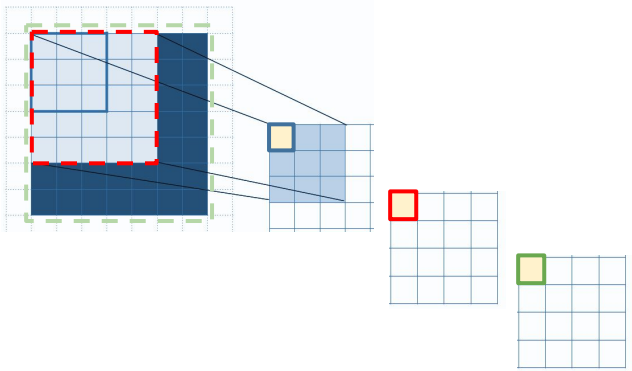
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

https://arxiv.org/pdf/1409.1556.pdf

VGG

One main contribution in VGG was using multiple layers of 3x3 convolutions to replace a single 7x7. This required less parameters but was more efficient than a single 7x7 convolution.

"It is easy to see that a stack of two 3×3 conv. layers (without spatial pooling in between) has an **effective receptive field** of 5×5; three such layers have a 7 × 7 effective receptive field." - Simonyan et al.



3 layers x (3 x 3) = 27 parameters
1 layer x (7 x 7) = 49 parameters

Field of view for 2D system

http://cs231n.github.io/convolutional-networks/

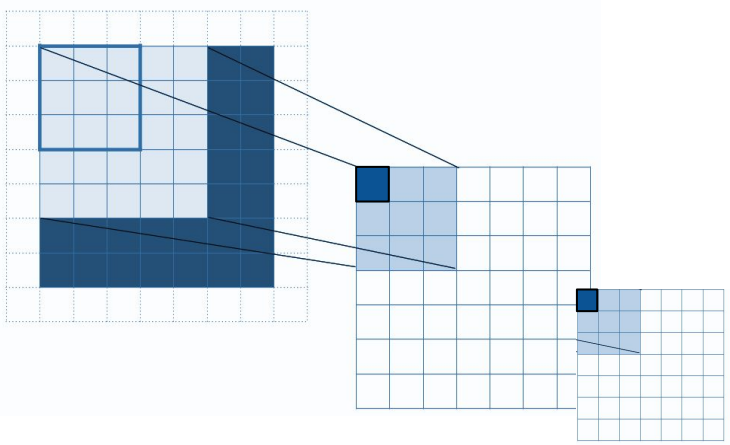
https://arxiv.org/pdf/1409.1556.pdf

VGG

One main contribution in VGG was using multiple chains of 3x3 convolutions to replace a single 7x7. This required less parameters but was more efficient than a single 7x7 convolution.

"It is easy to see that a stack of two 3×3 conv. layers (without spatial pooling in between) has an **effective receptive field** of 5×5; three such layers have a 7 × 7 effective receptive field." - Simonyan et al.

3 layers x (3 x 3) = 27 parameters
1 layer x (7 x 7) = 49 parameters



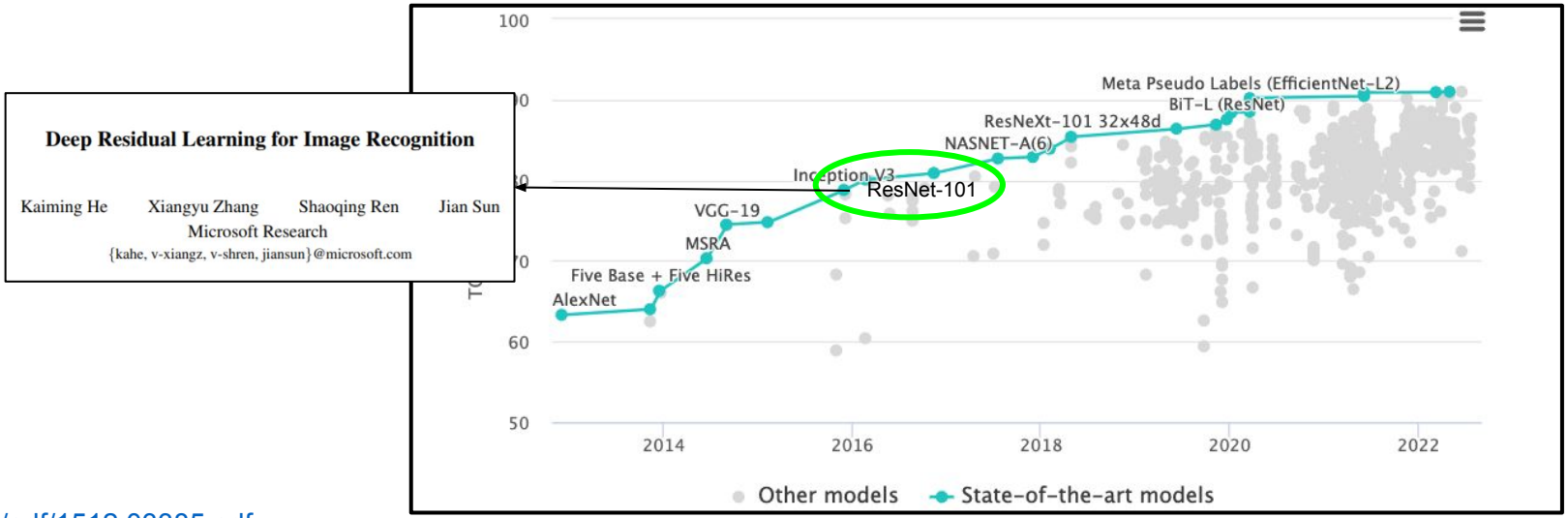
Field of view for 2D system

http://cs231n.github.io/convolutional-networks/

https://arxiv.org/pdf/1409.1556.pdf

ResNet

Following VGG, it was noticed that performance hit a limit when simply increasing depth.



https://arxiv.org/pdf/1512.03385.pdf

ResNet

After some number of layers, simply increasing depth would result in degrading **accuracy**.

However, since a network could simply learn to **bypass** layers via **identity mapping**, two networks that differ by depth only should in theory be able to produce the same accuracy.

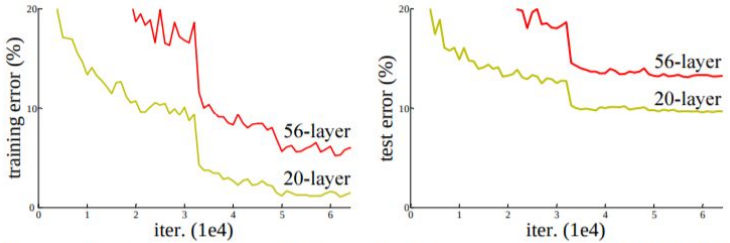


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

https://arxiv.org/pdf/1512.03385.pdf

ResNet

In ResNet, the idea of ***Residual blocks*** is introduced. This allows the networks to get deeper without accuracy degradation.

The main idea behind ResNet is that the network can always refer to **identity mappings** (skip connections) of the inputs from previous layers.

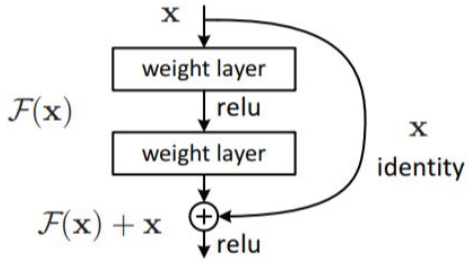


Figure 2. Residual learning: a building block.

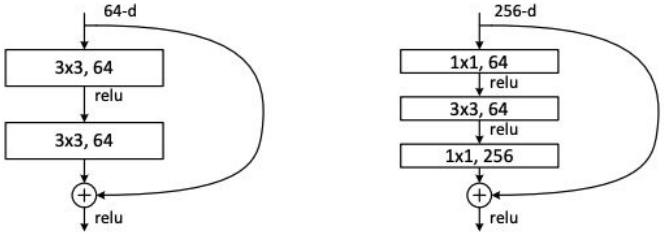


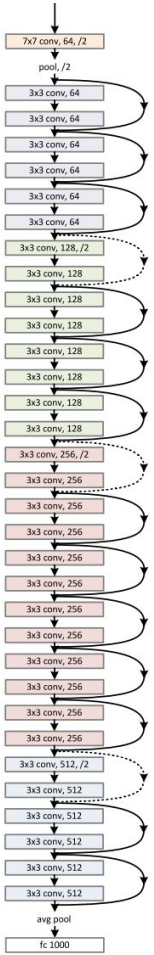
Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

https://arxiv.org/pdf/1512.03385.pdf

ResNet

The ResNet architecture consists of many stacked blocks. Different variations of ResNet were proposed, varying in the number of layers used.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



https://arxiv.org/pdf/1512.03385.pdf

Batch Normalization

ResNet also included **batch normalization** in its architecture. It acts as a form of regularization and is used after each convolutional layer.

The network learns at each layer “how much” to normalize its inputs according to batch statistics.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

https://arxiv.org/pdf/1502.03167v3.pdf

ResNet

ResNet achieved better accuracy with more depth and has less parameters than VGG. It was the first model to achieve **better performance** than humans on imagenet.

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

https://arxiv.org/pdf/1512.03385.pdf

ResNeXt

There have been many variations on ResNets in the following years, e.g. ResNeXt.

	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [14]	22.0	6.0	-	-
ResNet-200 [15]	21.7	5.8	20.1	4.8
Inception-v3 [39]	-	-	21.2	5.6
Inception-v4 [37]	-	-	20.0	5.0
Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (**64** × **4d**)	20.4	5.3	**19.1**	**4.4**

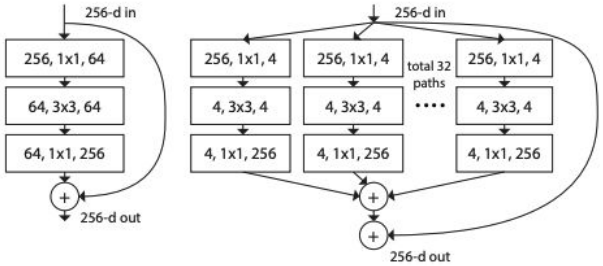


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

https://arxiv.org/pdf/1611.05431v2.pdf

Vision Transformers

Transformers

While CNNs were being widely adopted in computer vision, **transformers** started making their way into the field of natural language processing (NLP).

Their ability to **scale** to large datasets and clever implementations allowed them to dominate on NLP tasks and benchmarks.

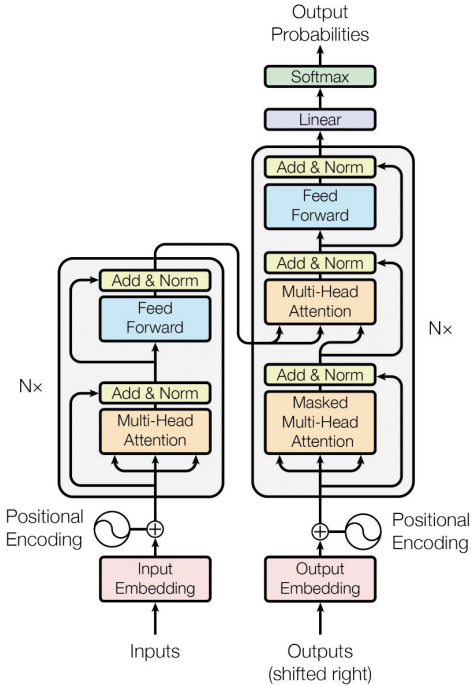
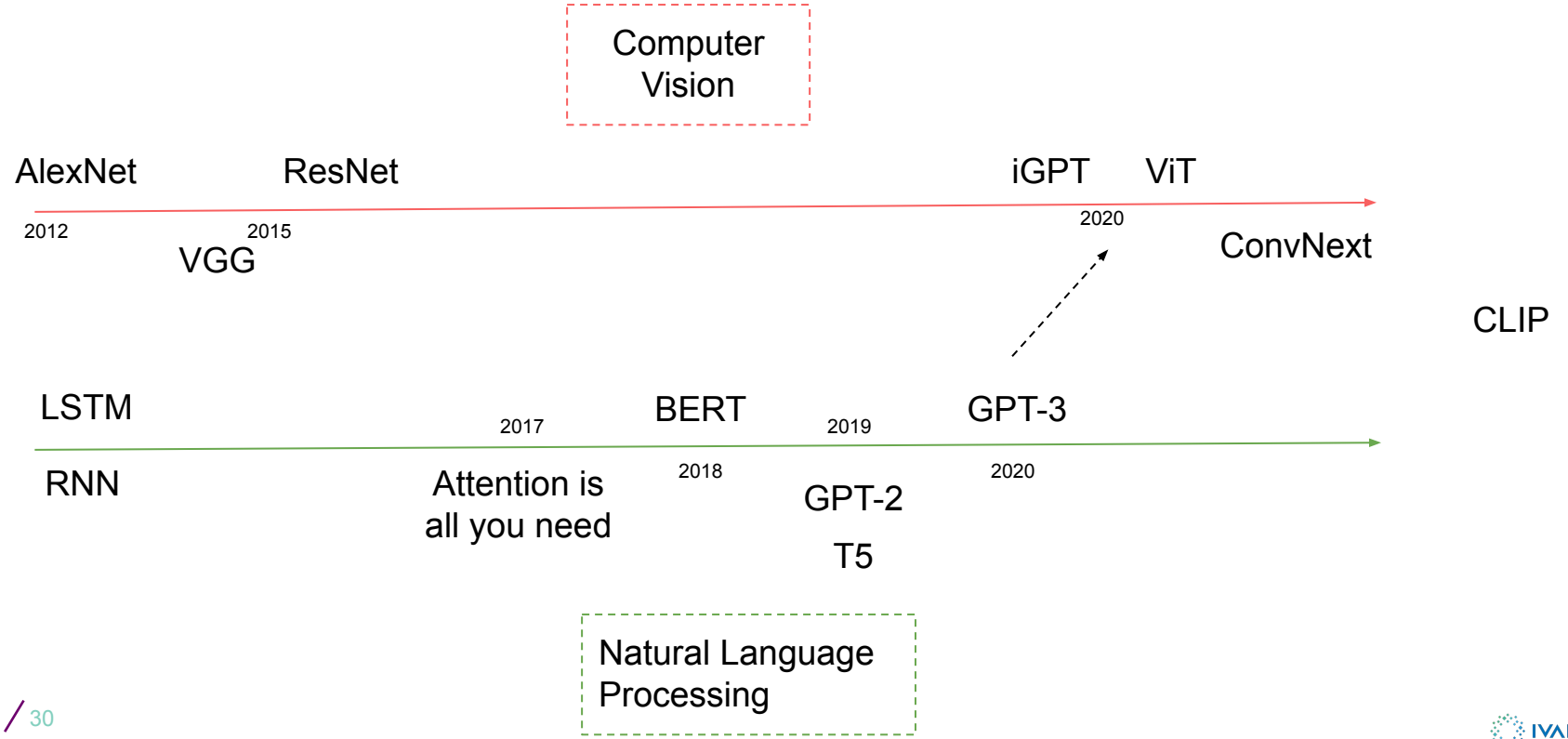


Figure 1: The Transformer - model architecture.

https://arxiv.org/abs/1706.03762?amp=1

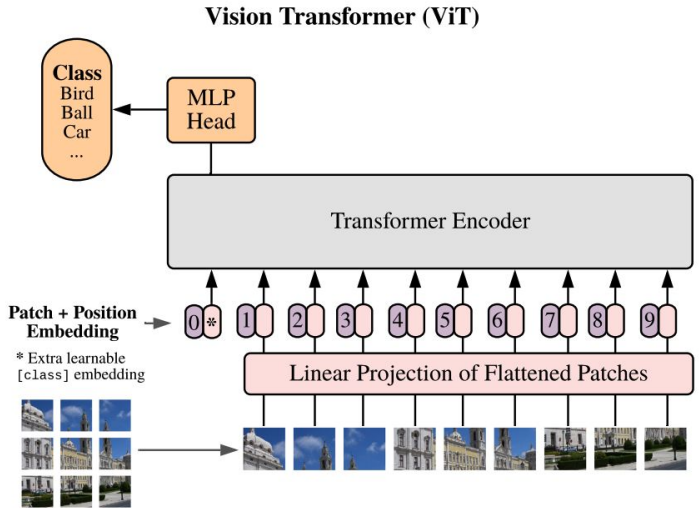
Transformers - Timeline



Vision Transformer

It took some time before transformers were successfully **adapted** for computer vision applications.

Arguably, the vision transformer (ViT) was a **turning point** for transformers applied to images.



https://arxiv.org/abs/2010.11929

Attention

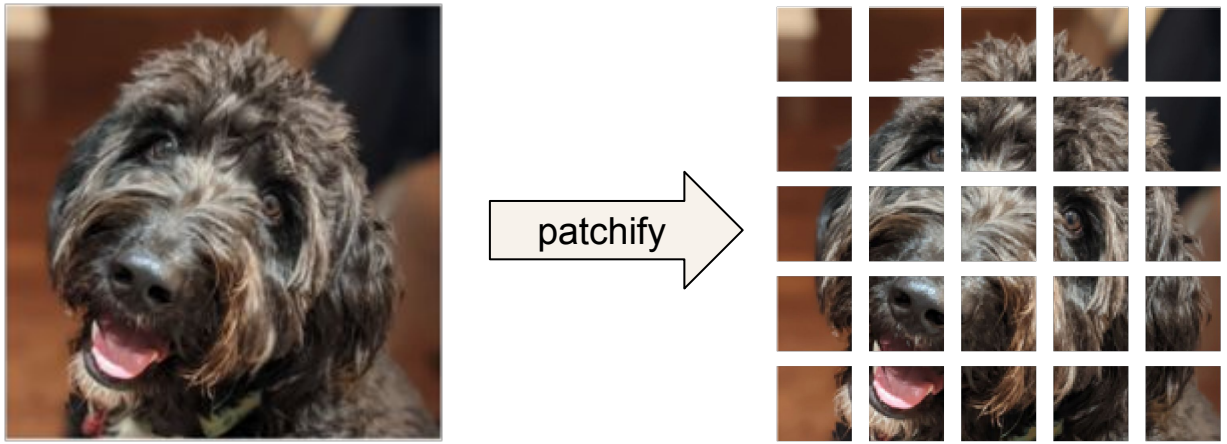
The central component of transformers is the **attention mechanism**.

It allows regions of images to **selectively** pay attention to other regions depending on the surrounding context.

https://arxiv.org/pdf/2010.11929.pdf

Image Patches

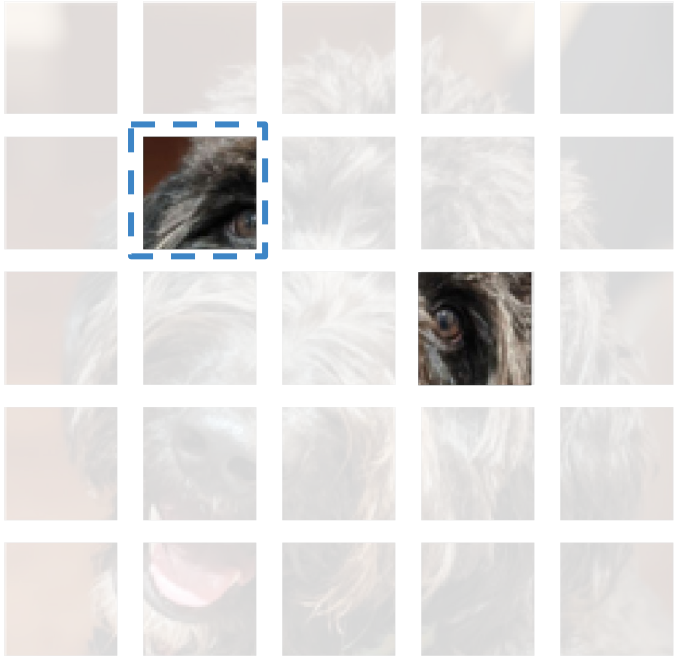
In vision transformers, Images are first divided to **patches**.



Attention

Each patch must then determine how much **attention** to pay to every other patch in the image based on its own content.

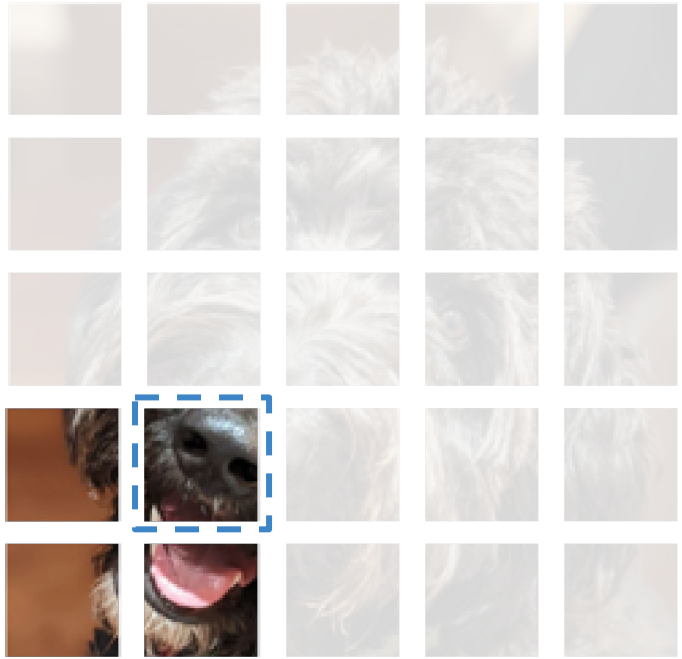
This is called **self-attention**.



Attention

Each patch must then determine how much **attention** to pay to every other patch in the image based on its own content.

This is called **self-attention**.



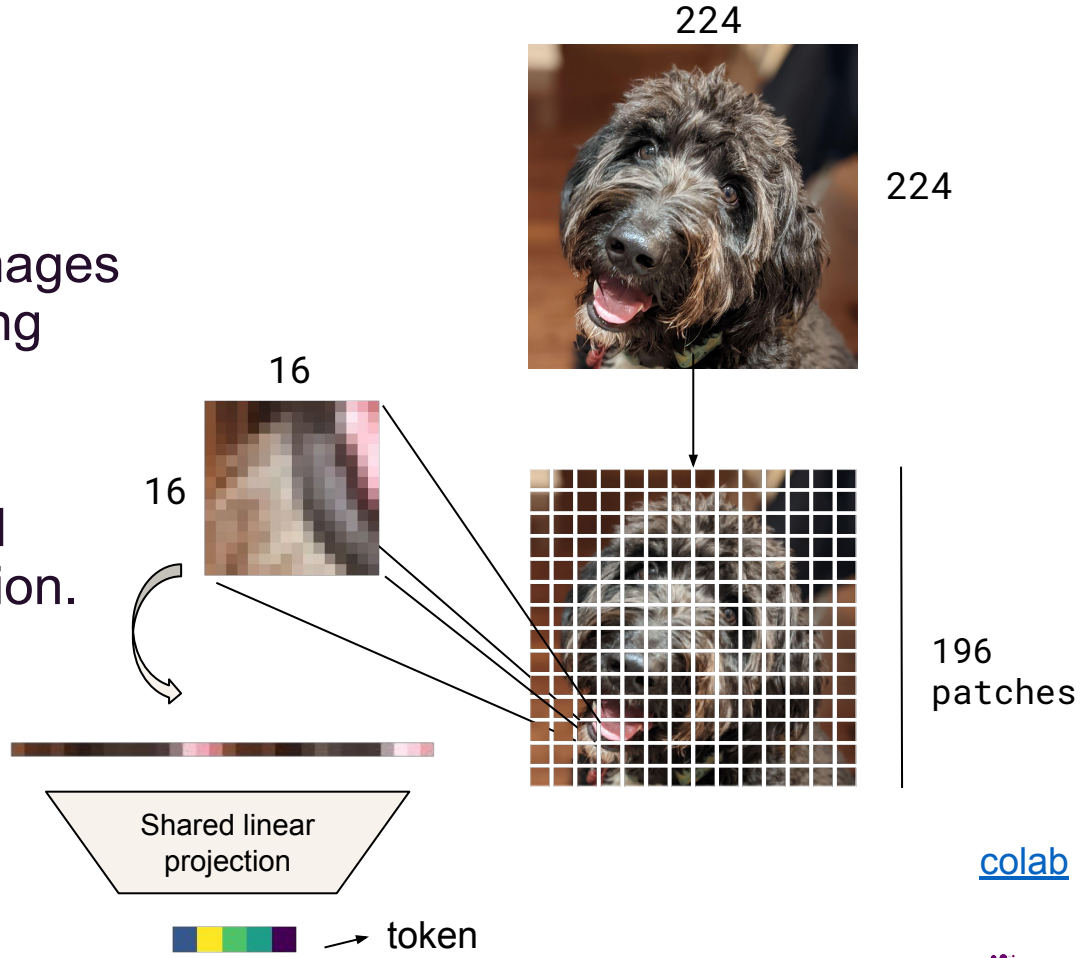
Vision Transformers

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

In Vision Transformers (ViT), images are first split into non-overlapping **patches** of 16x16 pixels.

Each patch is **flattened** and fed through a **shared** linear projection.

The resulting vector is called a **token**.



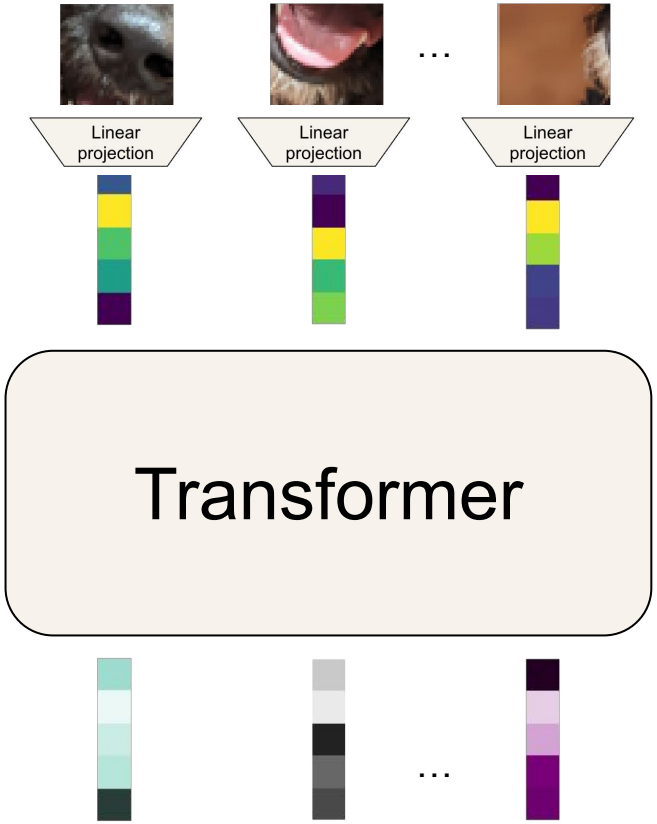
colab

Vision Transformers

The transformer operates on sequences of **tokens**. Each token is a 1xN vector.

Transformers take sequences of tokens as inputs and output **transformed** tokens.

A sequence in vision is composed of every patch in the image. Transformers can operate on **variable sizes of sequences**.

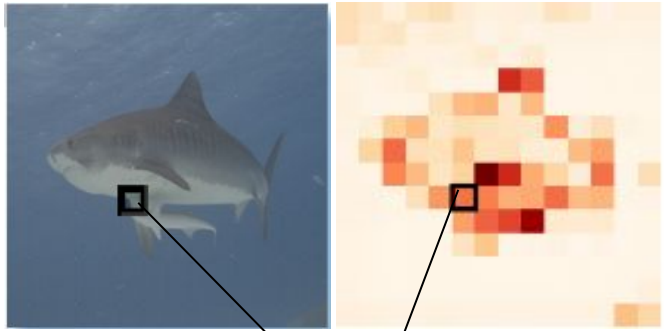


Self-Attention

The key ingredient to transformers is the **self-attention** mechanism.

It is a way of **combining** the different tokens together as they cascade through the transformer layers.

Tokens can “**pay attention**” to other tokens via attention mechanisms.



https://epfml.github.io/attention-cnn/

Self-Attention

The **attention mechanism** can be summarized as follows:

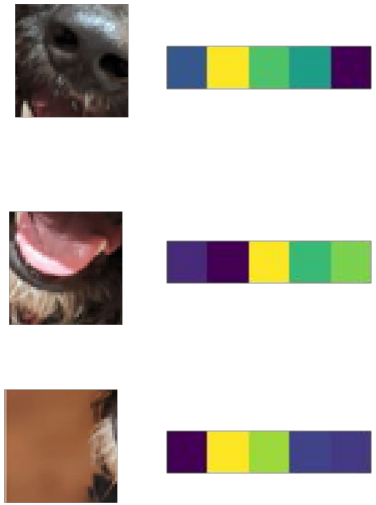
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

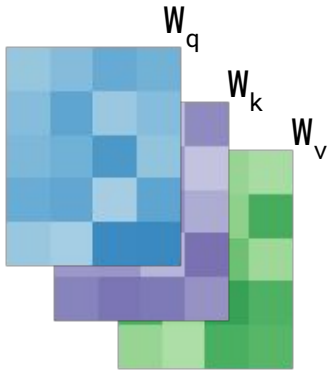
Here, **Q** stands for **Q**ueries, **K** for **K**eys a and **V** for **V**alues. These are obtained by projecting the tokens via matrix multiplications with learnable parameters, $\mathbf{W_q}$, $\mathbf{W_k}$, and $\mathbf{W_v}$, respectively.

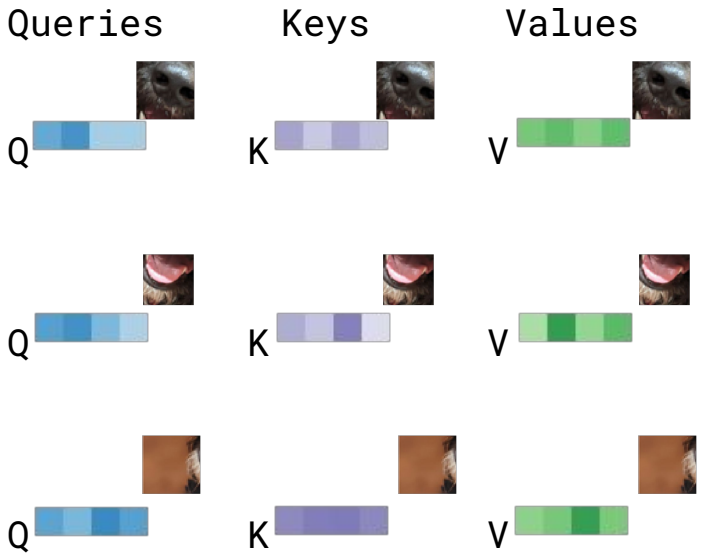
Step 1

We compute Q, K, V by multiplying the tokens with each W_q, W_k, W_v, matrix

Input image:



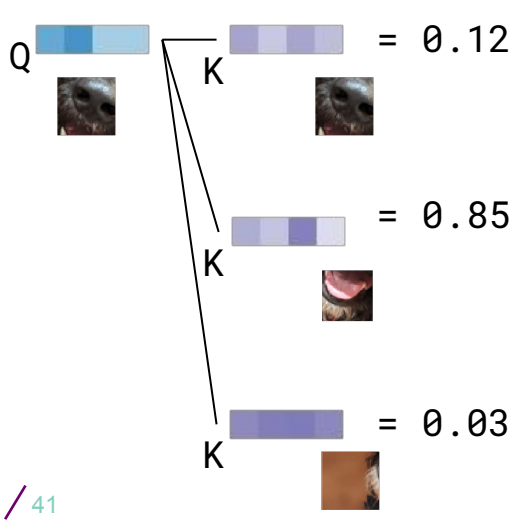




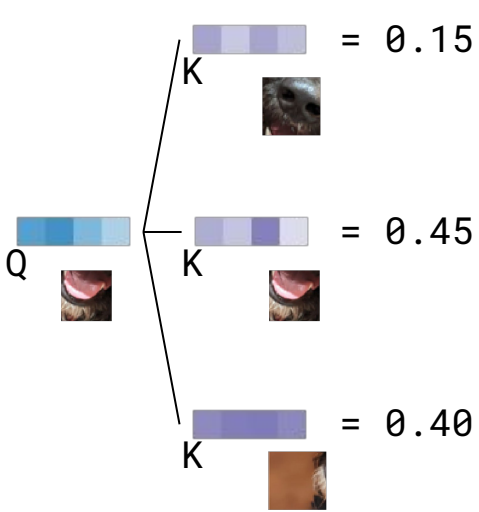
Step 2

For each **query**, we compute the **dot product** with every **key** and compute a normalized score

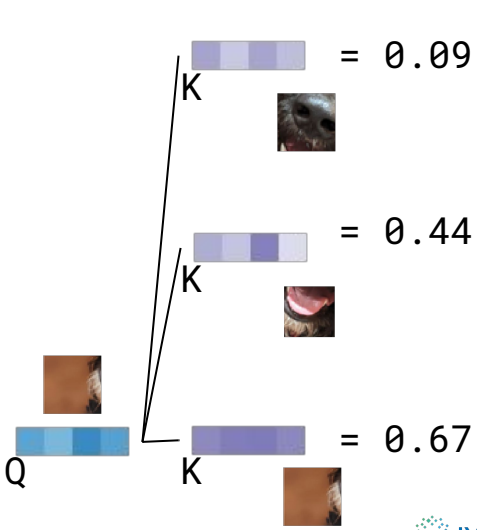
$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$$



$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$$



$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$$



Step 3

Finally, we multiply the value V with the normalized score and add them together

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

0.12 * + 0.85 * + 0.03 * =

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

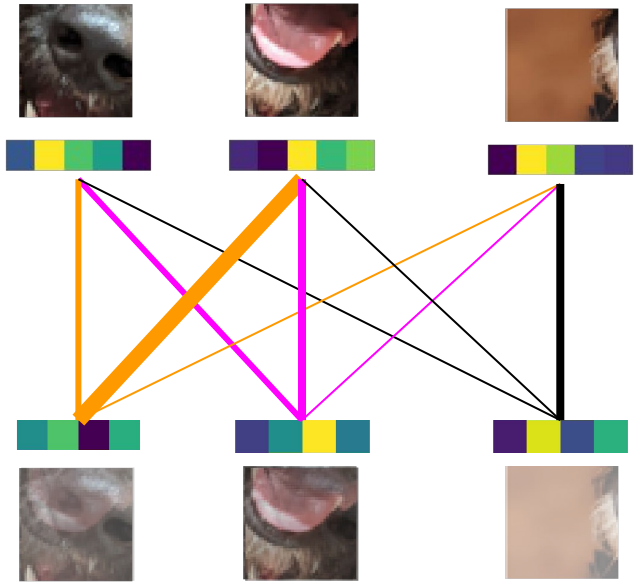
0.15 * + 0.45 * + 0.40 * =

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

0.09 * + 0.44 * + 0.67 * =

Step 3

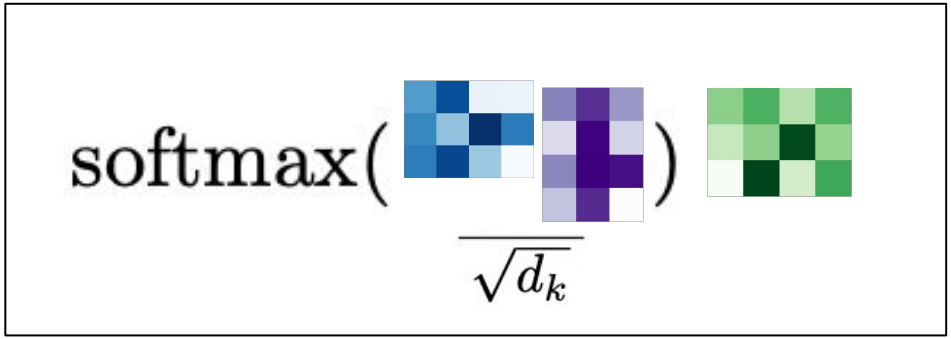
The resulting output vectors are weighted compositions of the input vectors. Each patch selectively **pays attention** to every other and can focus on the entire context.



Self-attention

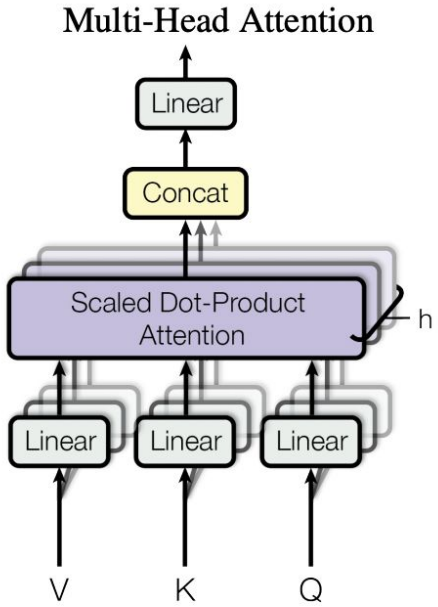
The entire procedure can be expressed as a series of matrix multiplications. This is called self-attention.

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad =$$



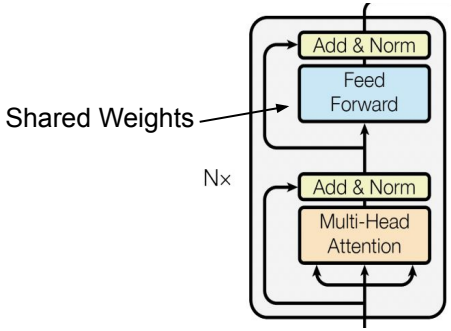
Multi-head self-attention

We can compute self-attention in parallel using **multi-head attention (MHA)**. The outputs of each head are concatenated and linearly projected to **preserve dimensions**.



Position-wise MLP

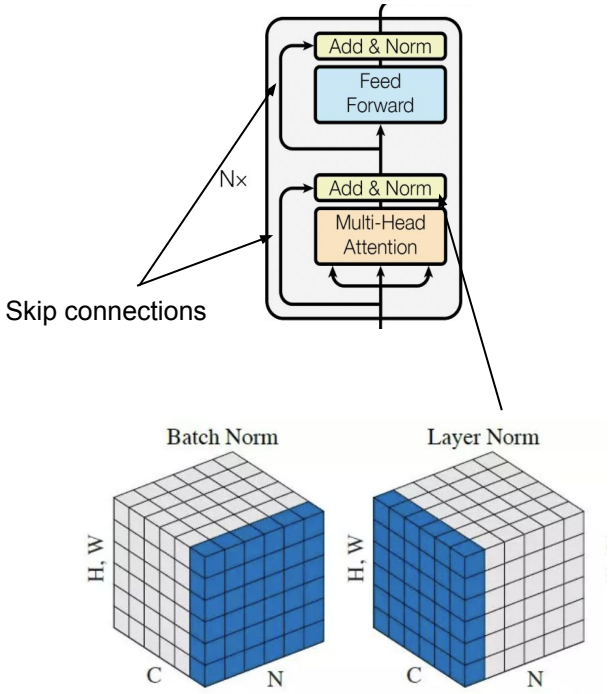
After the MHA head, a **position-wise** MLP is applied to every output vector (shared weights), e.g. 2 layers with ReLU activations.



Layer Normalization

As with ResNets, **skip connections** are introduced to ensure information can be propagated at all stages of the network.

Batch normalization is replaced by **layer normalization** and applied after every layer.

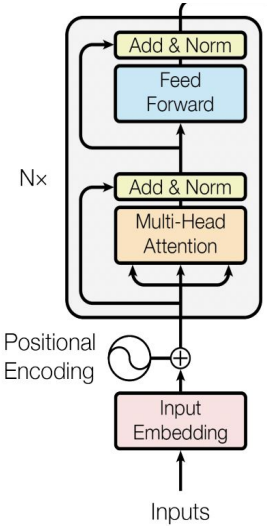


https://arxiv.org/pdf/1607.06450.pdf

Positional Encodings

Because of the dot-product operation in self-attention, **ordering** of the sequence is completely lost.

To overcome this limitation, **positional encodings** are added to every input.

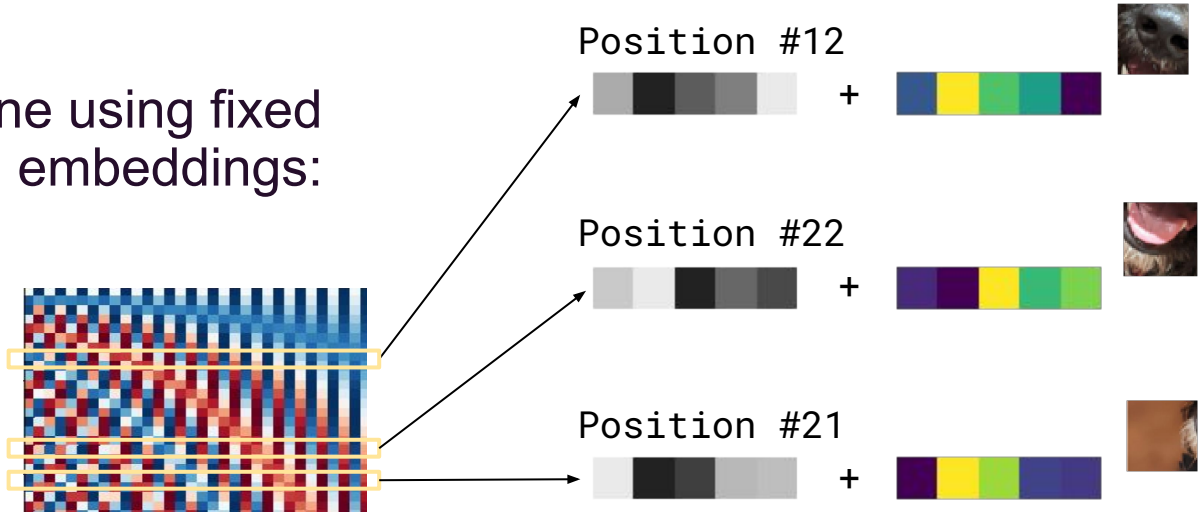


Positional Encodings

Positional encodings simply find a way to **encode** positional information to a vector.

In practice, this can be done using fixed (e.g. sinusoids) or learned embeddings:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

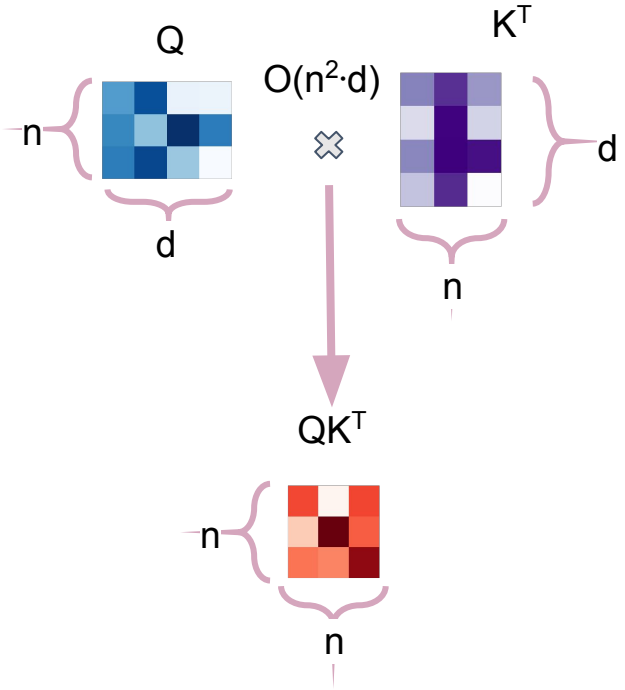


Quadratic Complexity

One big limitation of transformers is their **quadratic complexity** with input size.

Because self-attention requires computing the attention relative to **every other query**, this results in $O(n^2)$ complexity.

As an input grows linearly in length, the complexity increases quadratically.

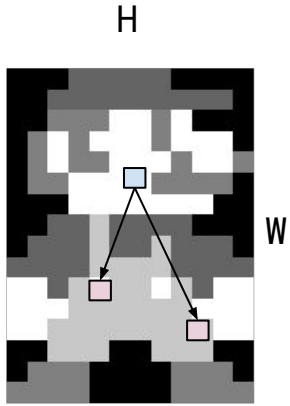


Quadratic Complexity

Consider applying attention to each **individual** pixel in an image.

The complexity for images becomes $O(H^2W^2)$ which is **impractical** for image applications.

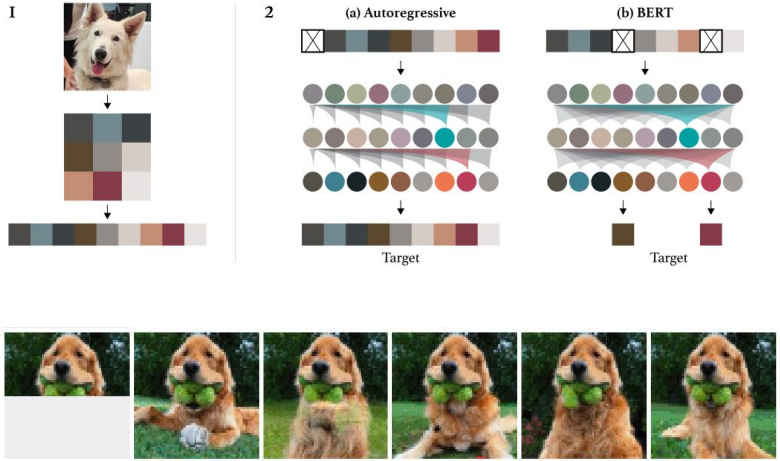
Applying transformers to **patches** instead of pixels was the biggest **innovation** in ViT.



iGPT

iGPT was one of the first attempts to use **transformers** on images. Images were heavily downsampled to 64x64 and used a single custom 9-bit colour channel.

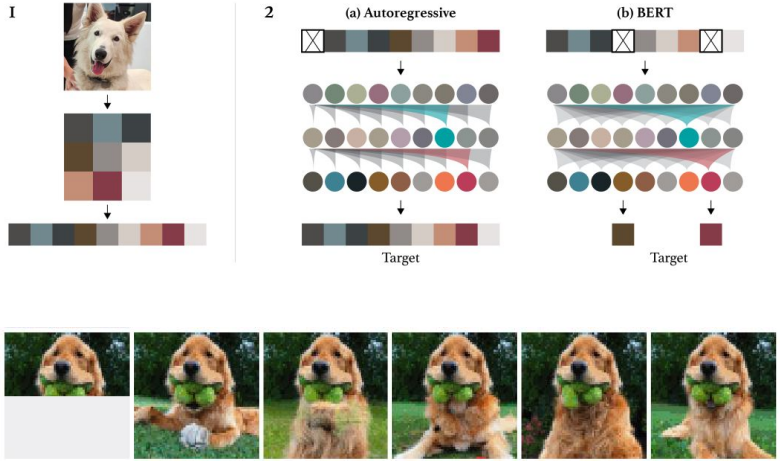
Results were promising but the methods were still **impractical** to be deployed at scale.



source

iGPT

"Although dense self-attention was a deliberate choice for this work [...], it becomes very memory and computationally **expensive** due to its **quadratic scaling** with sequence length"

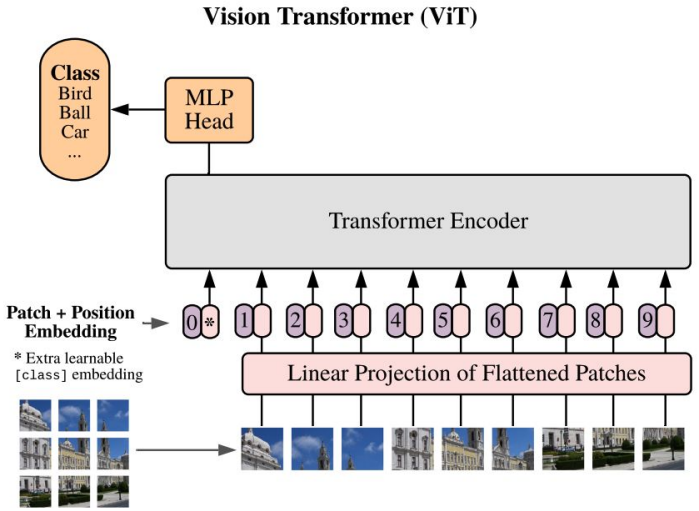


https://cdn.openai.com/papers/Generative_Pretraining_from_Pixels_V2.pdf

Vision Transformer

Vision Transformers (ViT) came up with the clever way of dealing with the quadratic scaling issues via **patches**.

Using their approach, they were finally able to secure the **top spot** on the ImageNet leaderboard.

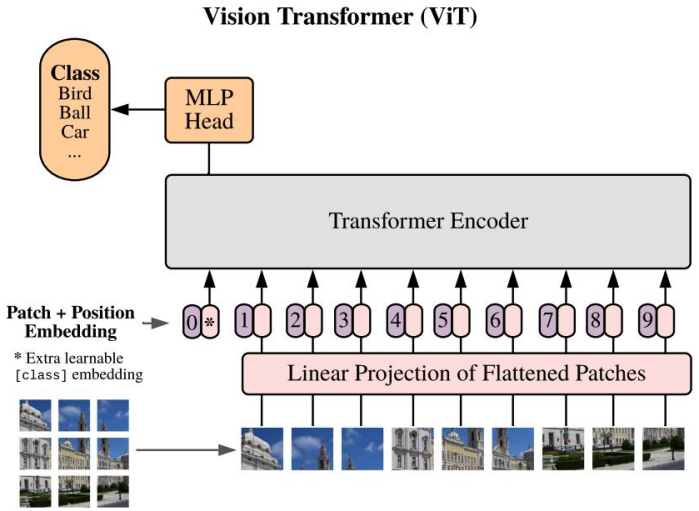


https://arxiv.org/abs/2010.11929

Vision Transformer

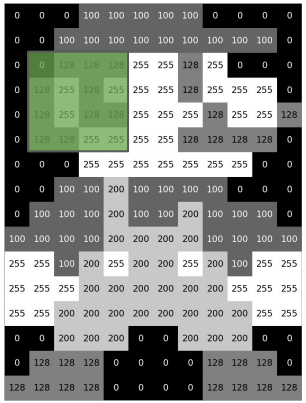
Similar to the NLP literature, an extra [class] **learned embedding** is prepended to every sequence.

An MLP head predicts the class of the token sequence via **cross-entropy** and can be trained end to end.



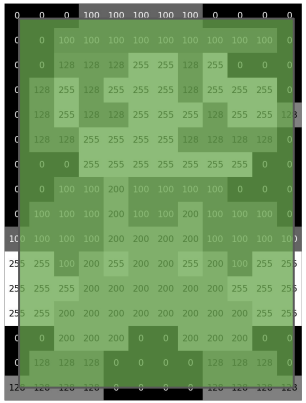
https://arxiv.org/abs/2010.11929

Summary



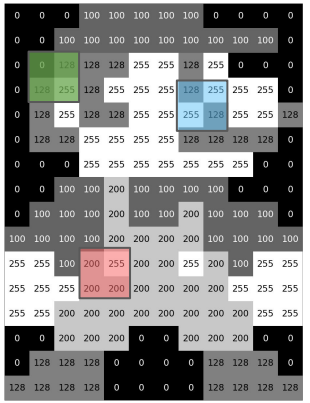
CNN

- Local context
- Shared parameters
- Information is combined downstream



MLP

- Global context
- Densely connected
- No inductive bias



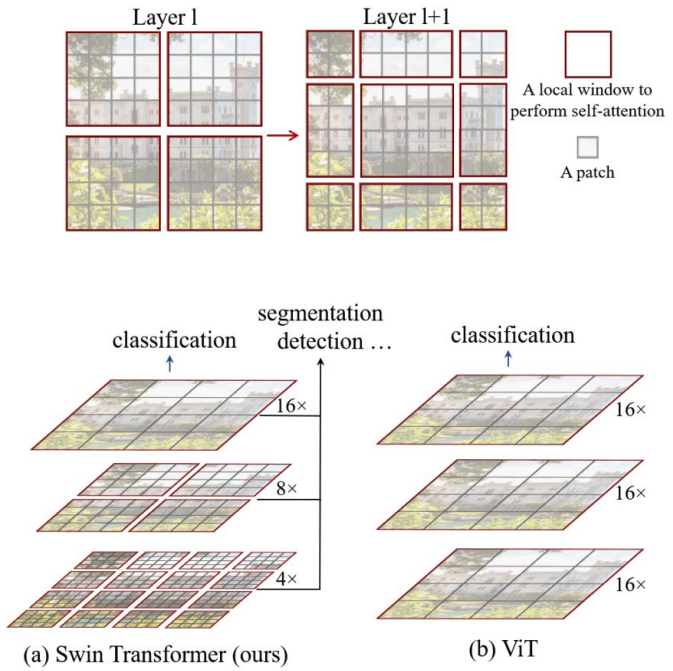
Transformers

- Global context
- Information is selectively chosen by attention

Swin Transformer

There have been many proposed improvements to the original ViTs to address scalability issues as well as to include more **inductive biases** like in CNNs.

For example, in Swin Transformers, attention is **constrained to local windows** (red) and patch sizes can vary (gray).

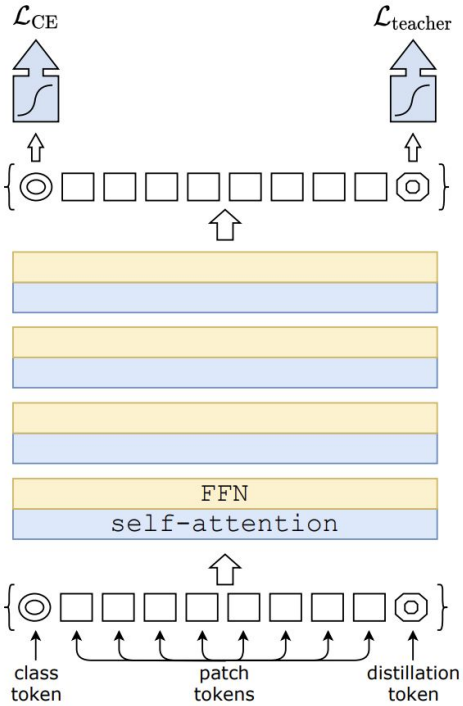


https://arxiv.org/abs/2103.14030

DEiT

One big limitation of ViT was the requirement of **large-scale** pre-training on much larger datasets for convergence.

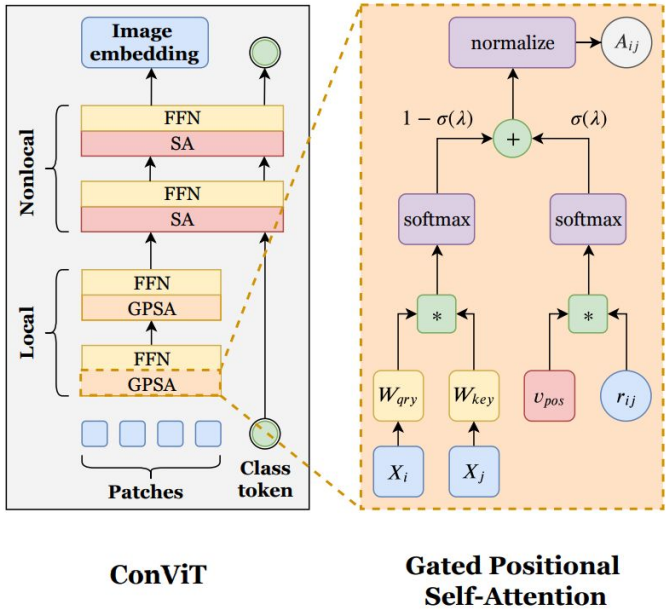
DEiT provided strategies for training on smaller datasets (e.g. ImageNet) via distillation.



https://arxiv.org/pdf/2012.12877.pdf

ConViT

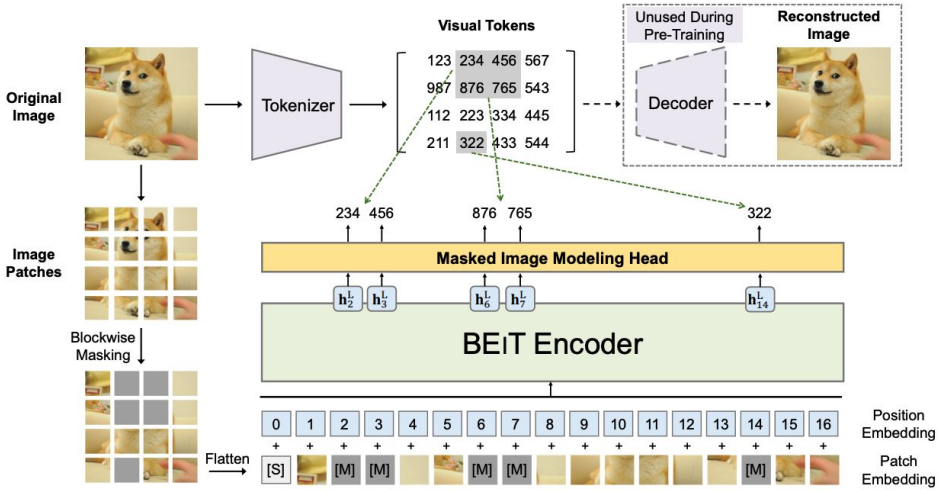
In ConViT, attention mechanisms are initialized as convolutions and the network can **learn to ignore** the convolutions if necessary.



https://arxiv.org/pdf/2103.10697.pdf

BeiT

Other methods have also been proposed to perform large-scale pre-training **without labels** by learning how to predict missing patches in a transformer.



https://arxiv.org/abs/2106.08254

Farewell Convolutions

So is this the end for convolutions?

Farewell Convolutions

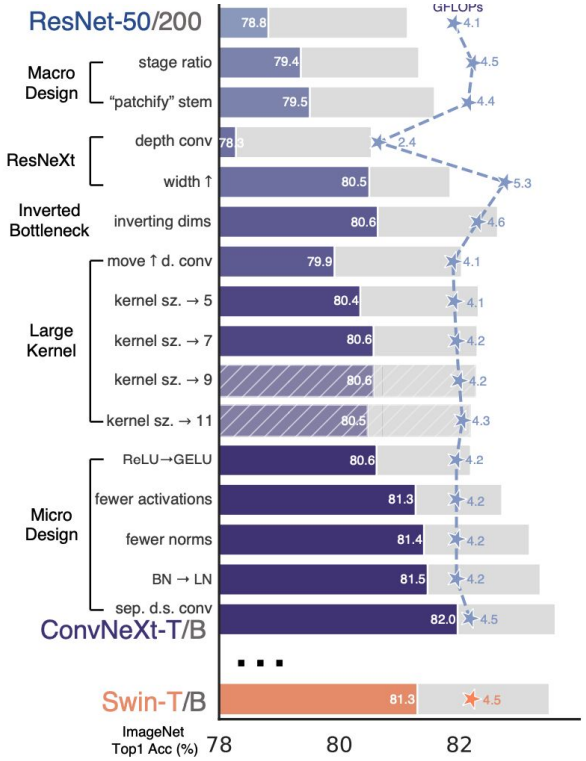
So is this the end for convolutions?

Not really.

ConvNext

Transformer litterature is ripe with tricks borrowed from the NLP community.

It was shown that by modifying CNNs to incorporate some of these tricks, **ConvNext** could be as competitive as its transformer counterparts.



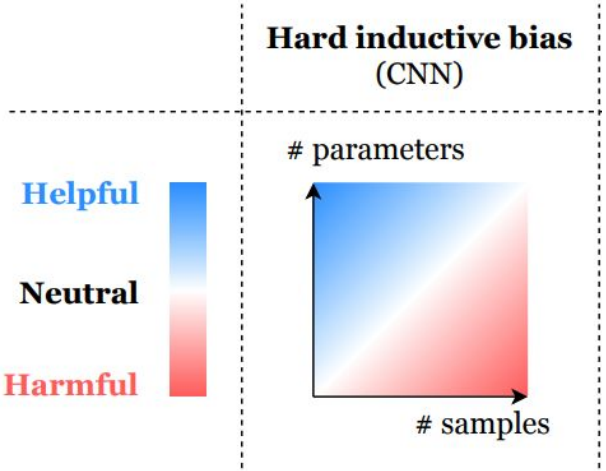
https://arxiv.org/abs/2201.03545

CNNs vs ViT

+ CNNs tend to operate better than ViTs in **lower-data regimes**.This is largely due to their hard-coded inductive bias.

+ CNNs tend to require much **less memory** and have **faster** inference/training speeds.

- Performance of CNNs doesn't scale as well as ViTs on **extremely large datasets** (>100 million images).

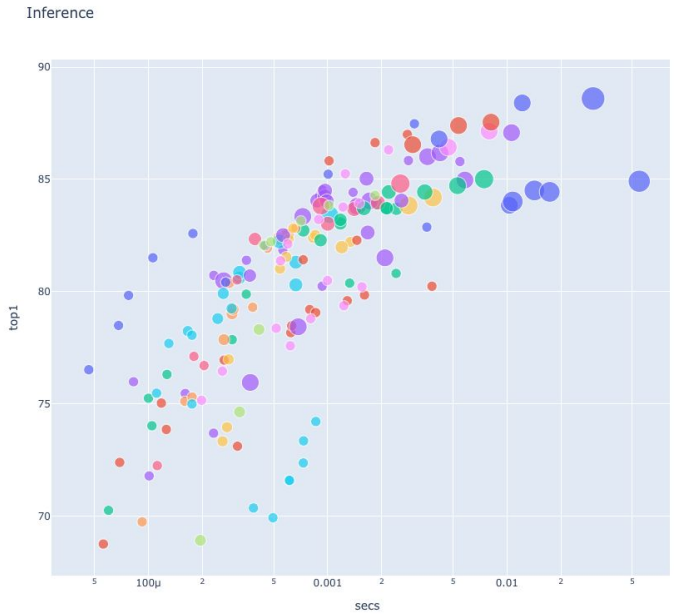


https://arxiv.org/pdf/2103.10697.pdf

Trade Offs

So which models should you choose?

- Pick a model that is easy to implement
- Pick a model that fits in memory
- Pick a model that you understand



https://www.kaggle.com/code/jhoward/which-image-models-are-best/notebook

MLP Mixer

Architectures like **MLP Mixer** have shown that replacing attention with MLP-like operations can lead to competitive results on ImageNet.

Note that **patches** and other **inductive biases** were crucial in the architecture.

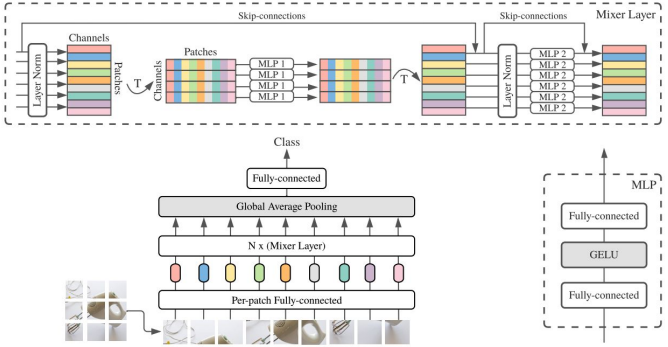
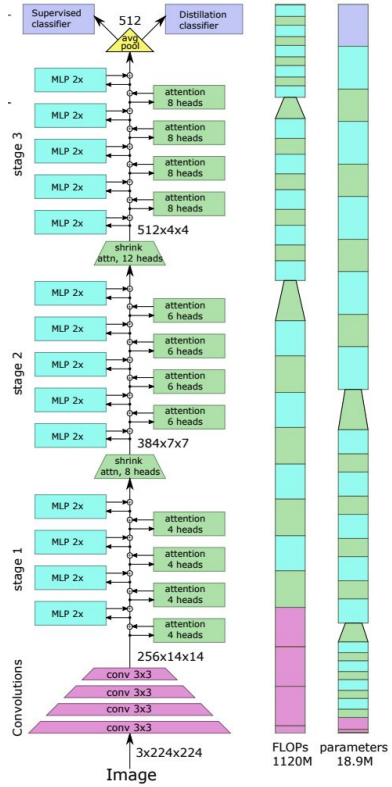


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer norm on the channels.

https://arxiv.org/abs/2105.01601

LeViT

Some models even try to use transformers in the original style of LeNet, with some success.



https://arxiv.org/pdf/2104.01136.pdf

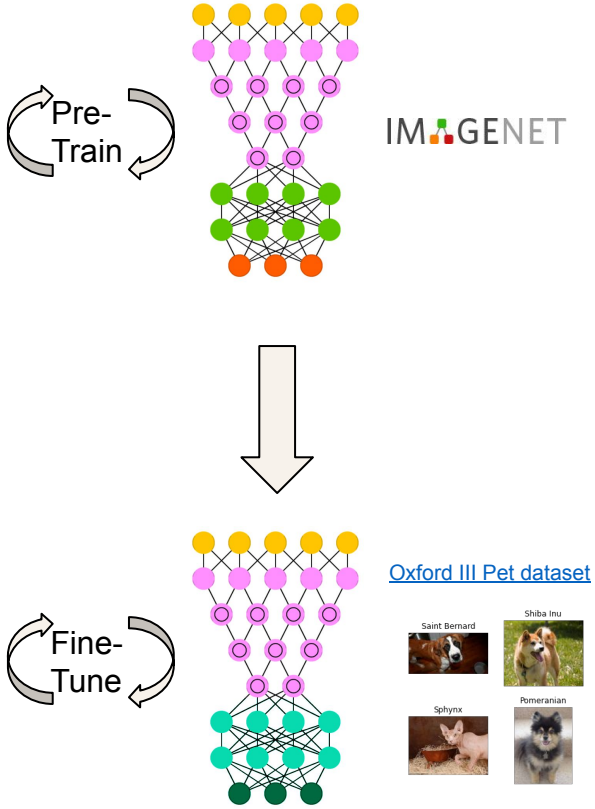
Fine-Tuning

Fine-Tuning

When training a model from scratch, weights are initialized (more-or-less) randomly.

By initializing with a model pre-trained on different tasks, convergence can be obtained **faster** and with **less data**.

This is known as **fine-tuning** or transfer learning.



ImageNet

Today, **almost all** top-scoring models on the ImageNet leaderboard are first pre-trained on **larger datasets** and fine-tuned on ImageNet.

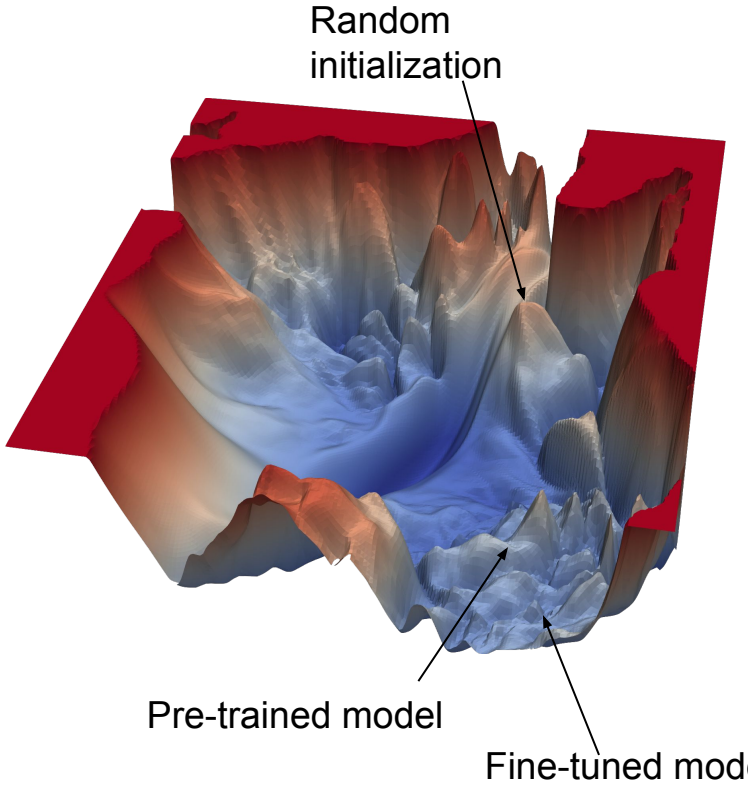
The size of currently available datasets are **orders of magnitude** larger than ImageNet.

https://engineering.fb.com/ml-applications/advancing-state-of-the-art-image-recognition-with-deep-learning-on-hashtags/

IVADO Mila

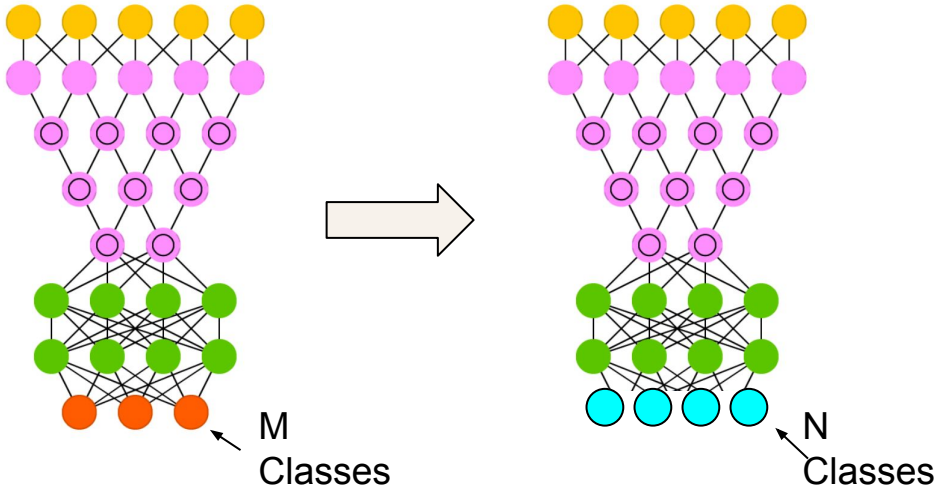
Fine-Tuning

The intuition is that pre-trained models on large and diverse datasets tend to learn **features** that are **useful** and **transferable** across a variety of tasks.



Fine-Tuning

To fine-tune a classification model, the first step is to **replace** the final classification layer with the correct number of classes for the new task.

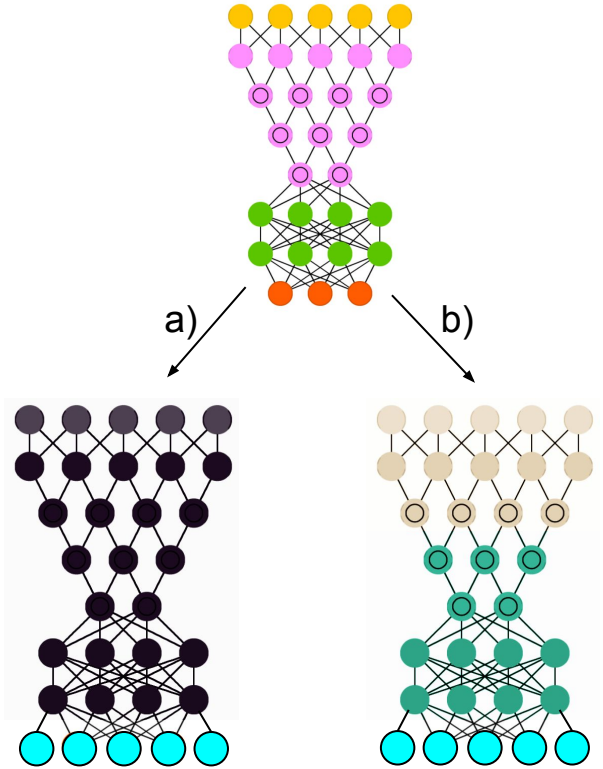


Fine-Tuning

The model is then trained on the new dataset. There can be different strategies when fine-tuning:

a) Freeze all layers **except** for the final classification layer

b) Re-train all layers simultaneously with a lower learning rate



Vision Transformers

Vision Transformers are only able to get top-1 accuracy when pre-trained on **much larger** datasets and fine-tuned on imagenet.

When trained only on ImageNet, they perform worse than vanilla resnets.

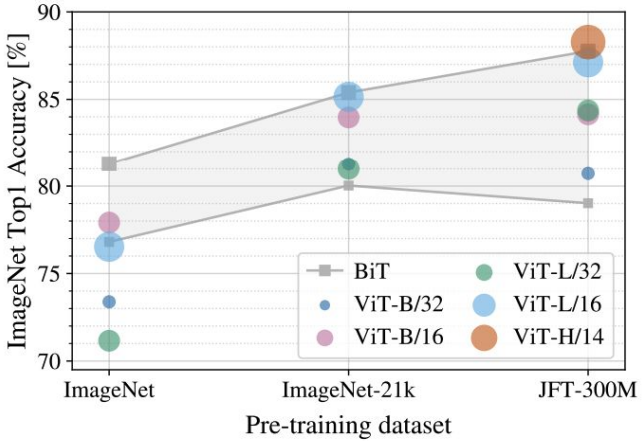


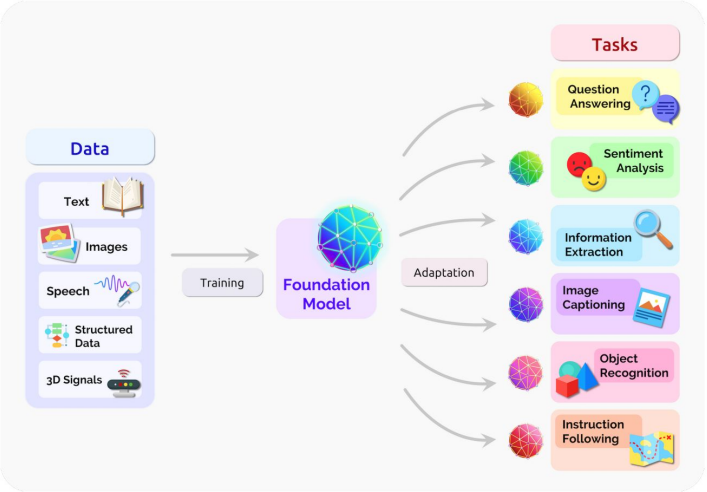
Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

https://arxiv.org/abs/2010.11929

Foundation Models

Foundation Models

“A foundation model is any model that is trained on **broad data** (generally using self-supervision at scale) that can be adapted (e.g., fine-tuned) to a wide range of downstream tasks” - Bommasani et al.

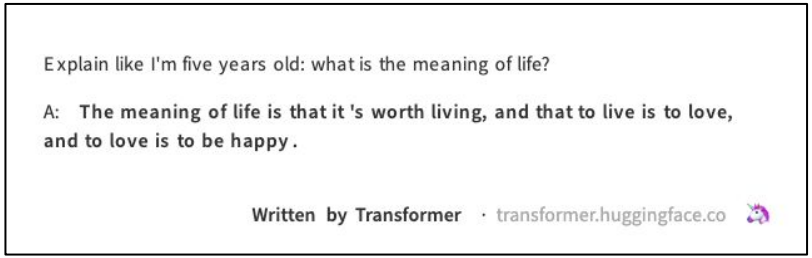


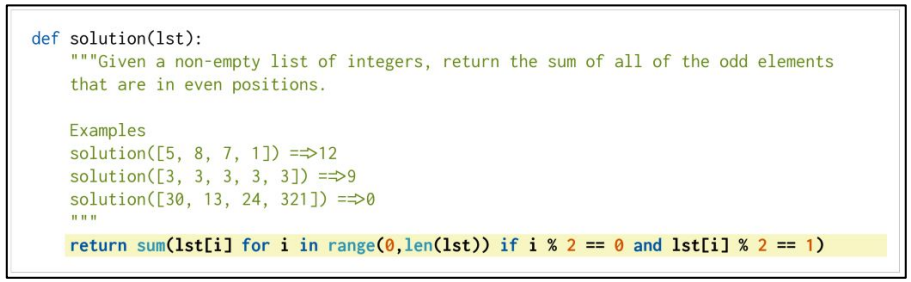
https://arxiv.org/abs/2108.07258

GPT

Some of the first foundation models (e.g. BERT, GPT) were trained on large piles of text (crawls of the entire internet).

The scale of data and self-supervision led to **impressive** emergent capabilities.





```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

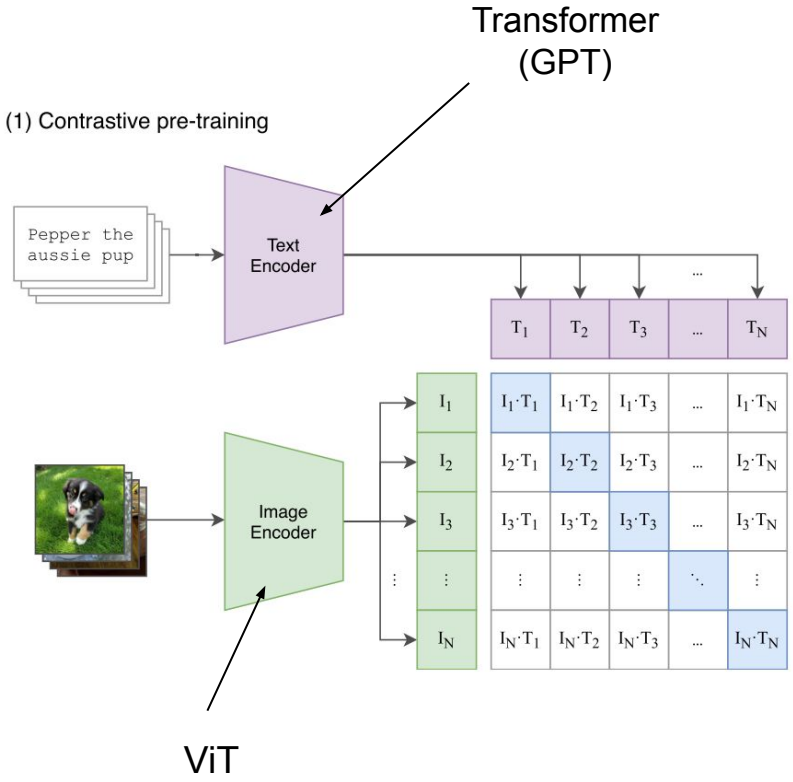
https://arxiv.org/pdf/2107.03374.pdf

IVADO Mila

CLIP

CLIP is a **multimodal** transformer trained on vast amounts (~400 million) of **images** and their associated **captions**.

CLIP predicts if an image aligns with its caption via **contrastive loss**.

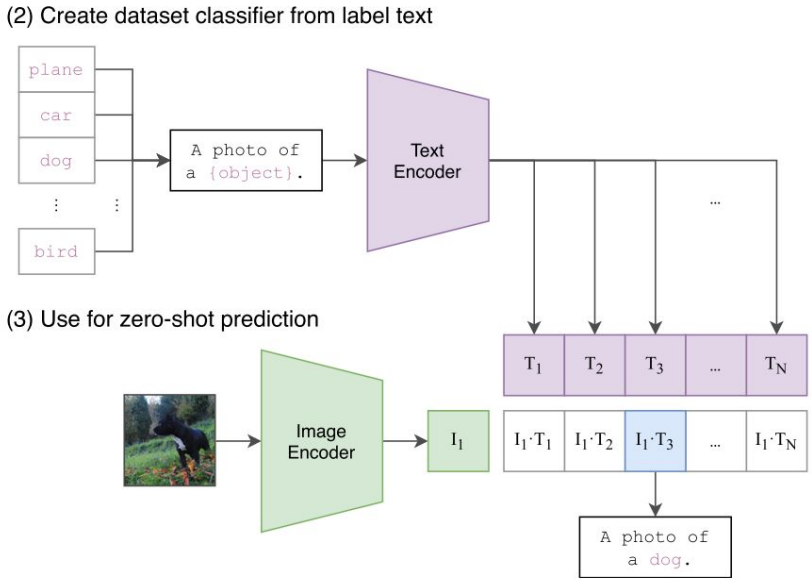


https://openai.com/blog/clip/
https://arxiv.org/abs/2103.00020

CLIP

Without fine-tuning, CLIP was shown to perform well on diverse downstream tasks.

For example: CLIP can be “programmed” with natural language to classify images into any user-defined categories.



https://openai.com/blog/clip/
https://arxiv.org/abs/2103.00020

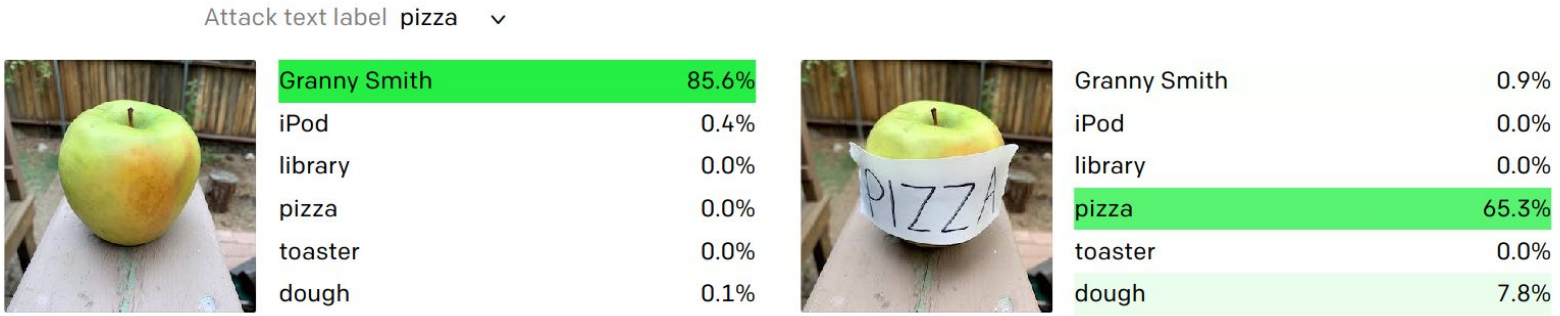
CLIP

CLIP demonstrates competitive results on ImageNet - and generalizes to other datasets too - without fine-tuning.

https://openai.com/blog/clip/
https://arxiv.org/abs/2103.00020

CLIP

Interestingly, CLIP seems to recognize written language.

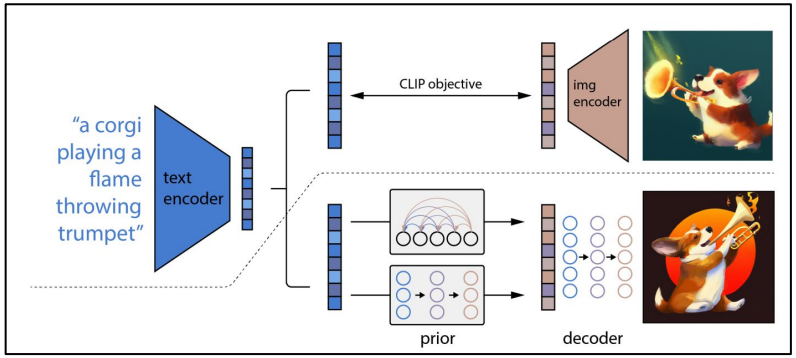


https://distill.pub/2021/multimodal-neurons/

DALL-E 2

With some clever adaptations, CLIP embeddings can be used to generate images from text prompts.

“An espresso machine that makes coffee from human souls, artstation”

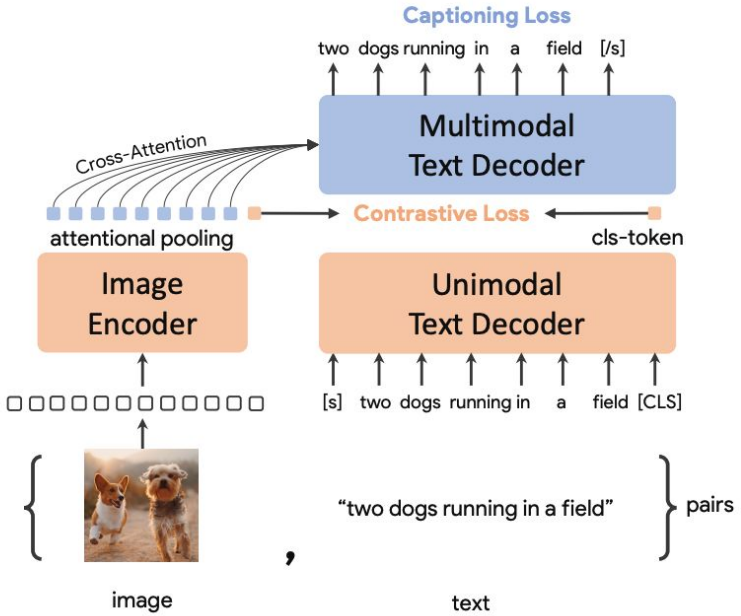


https://arxiv.org/pdf/2204.06125.pdf
https://openai.com/dall-e-2/

https://github.com/CompVis/stable-diffusion > open-source alternative

ImageNet

Currently, the **ImageNet leaderboard** is led by a foundational model trained on both language and images.



https://arxiv.org/pdf/2205.01917v2.pdf

Object Detection

Object Detection

Object detection seeks to find not only the **category** of the object in an image, but also the **coordinates** of the identified object.

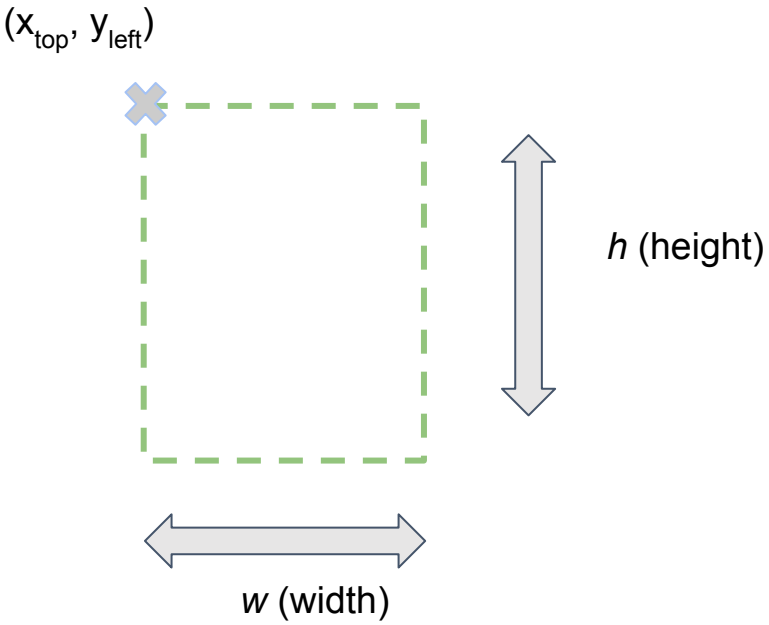
Classification:
“Bicycle”

Object Detection:
“Bicycle” + Bounding Box

Object Detection

Bounding boxes can be represented using 4 coordinates.

Object Detection:
"Bicycle" + Bounding Box



Mila

Loss functions

The bounding box coordinates can be learned using a **regression loss**, while the class label can be learned using a **cross-entropy** loss.

“bicycle”

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*)$$
$$+\lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*).$$

[x, y, w, h]

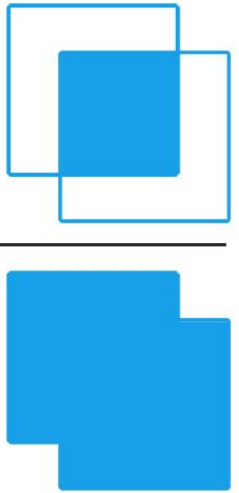
https://arxiv.org/pdf/1506.01497.pdf

Object Detection

To evaluate a bounding box, the Intersection over Union (IoU) metric can be used. It measures the overlap between prediction and ground truth. IoU is a value between 0 (worst) and 1 (best).

Object Detection:
"Bicycle" + Bounding Box

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



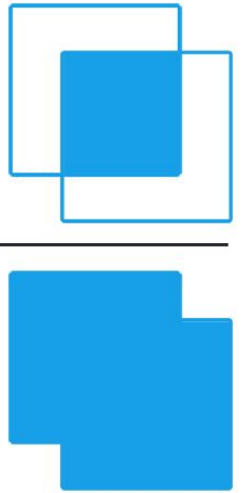
Mila

Object Detection

To evaluate a bounding box, the Intersection over Union (IoU) metric can be used. It measures the overlap between prediction and ground truth. IoU is a value between 0 (worst) and 1 (best).

Object Detection:
"Bicycle" + Bounding Box

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



https://farm3.staticflickr.com/2578/4232522029_9dffb2a2df_z.jpg

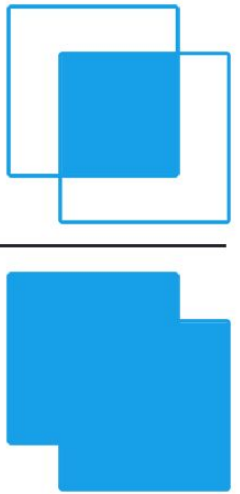
Mila

Structured Outputs

To evaluate a bounding box, the Intersection over Union (IoU) metric can be used. It measures the overlap between prediction and ground truth. IoU is a value between 0 (worst) and 1 (best).

Object Detection:
"Bicycle" + Bounding Box

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



https://farm3.staticflickr.com/2578/4232522029_9dffb2a2df_z.jpg

Mila

Faster R-CNN

Faster R-CNN can be used for **object detection**. A standard CNN (e.g. ResNet) is used to extract features, and an **RPN** detects positions of objects.

These are then **classified** accordingly and return for each box the associated labels, coordinates and confidence scores.

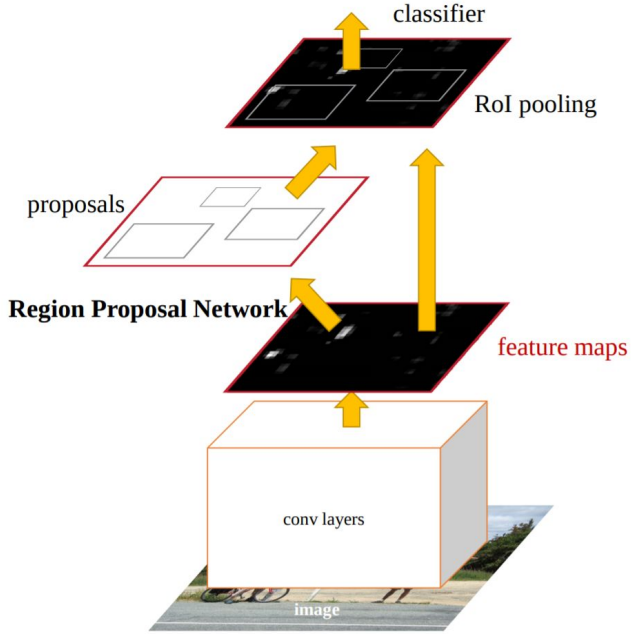


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

https://arxiv.org/pdf/1506.01497.pdf

DEtection TRansformer (DETR)

It has also been shown that **transformers** and **CNNs** can be **combined** to achieve competitive results at object detection.

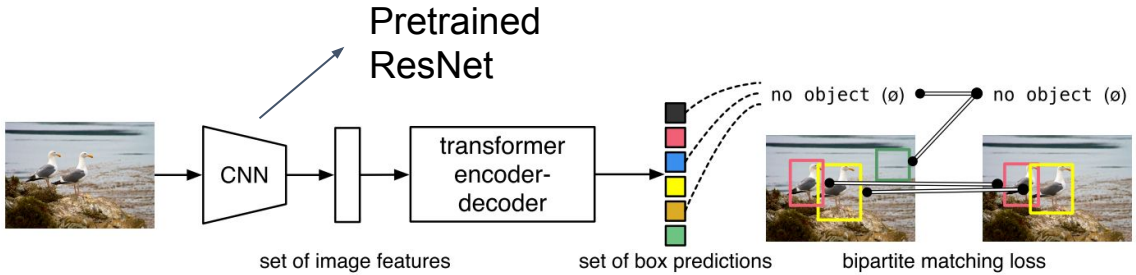


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

https://arxiv.org/pdf/2005.12872.pdf

Mila

Segmentation

Instead of bounding boxes, a **segmentation mask** can be used to identity an object at the pixel-level.

Each pixel is classified as belonging to an object or not.

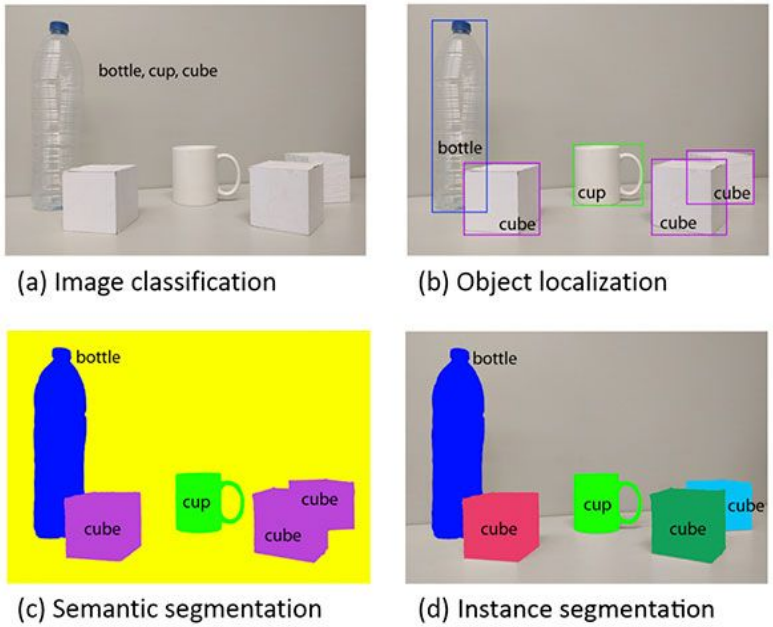
Segmentation:
“Bicycle” + Segmentation Mask

https://farm3.staticflickr.com/2578/4232522029_9dffb2a2df_z.jpg

Segmentation

In **semantic segmentation**, the entire image is classified, pixel by pixel and no distinction between different instances is made.

In **instance segmentation**, only regions of interest are segmented. **Distinction** between objects of the same class can be made.

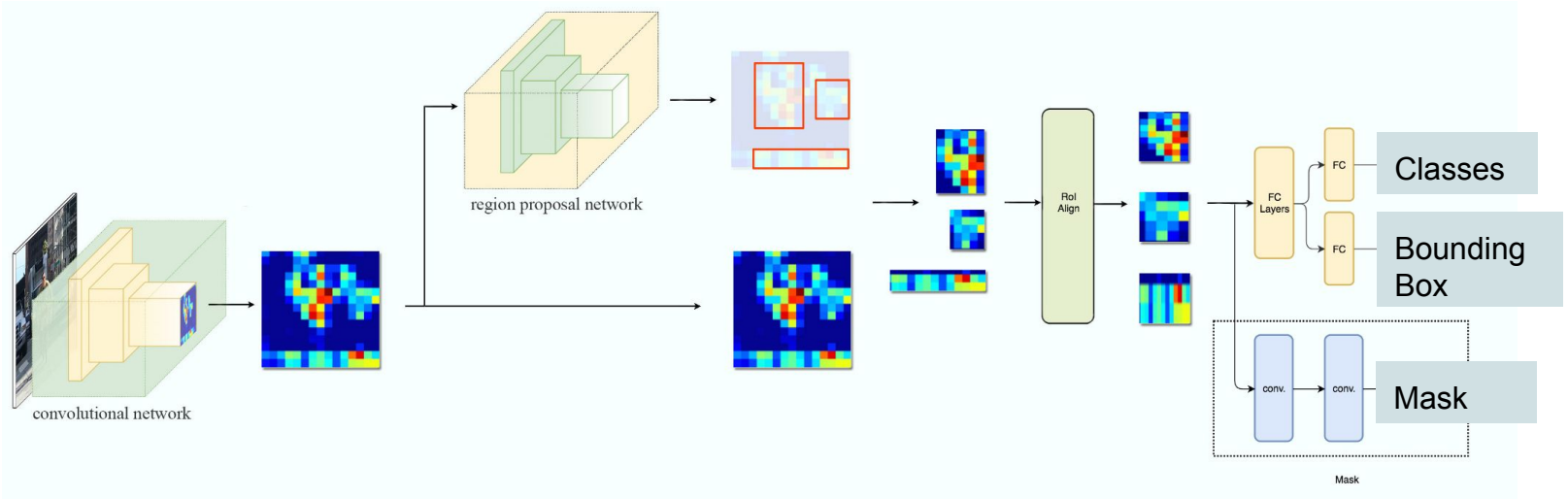


https://www.pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/
https://www.di.ens.fr/willow/events/cvml2010/materials/INRIA_summer_school_2010_Christoph.pdf

IVADO Mila

Mask R-CNN

Mask R-CNN is a **popular** architecture for **instance segmentation**. It works by predicting in **parallel** regions of interest (bounding boxes), which class they belong to, and the **segmentation mask** of these objects.

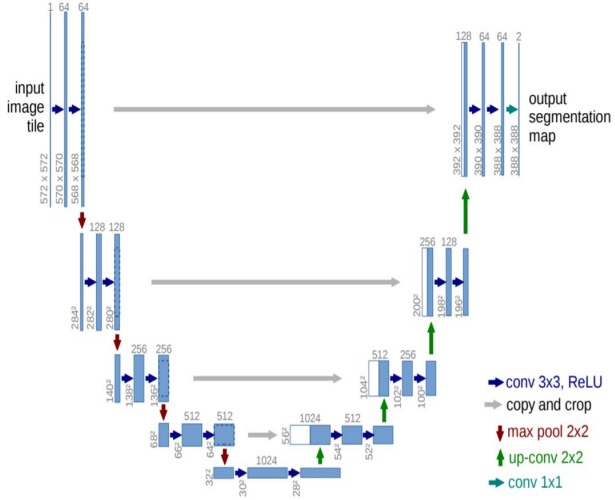


https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272

UNet

UNet is a model for **semantic segmentation** that was introduced in the context of biomedical image segmentation.

It preserves higher-level convolutions and **concatenates** them with lower-level convolutional layers via u-shaped skip-connections.



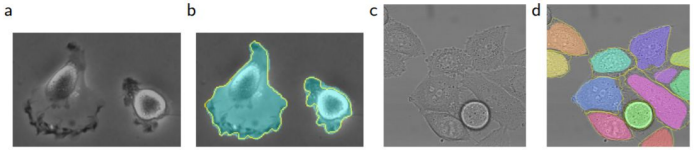


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

https://arxiv.org/pdf/1505.04597.pdf

IVADO Mila

TransUNet

In TransUNet, a CNN and a transformer are combined to to perform medical **image segmentation**.

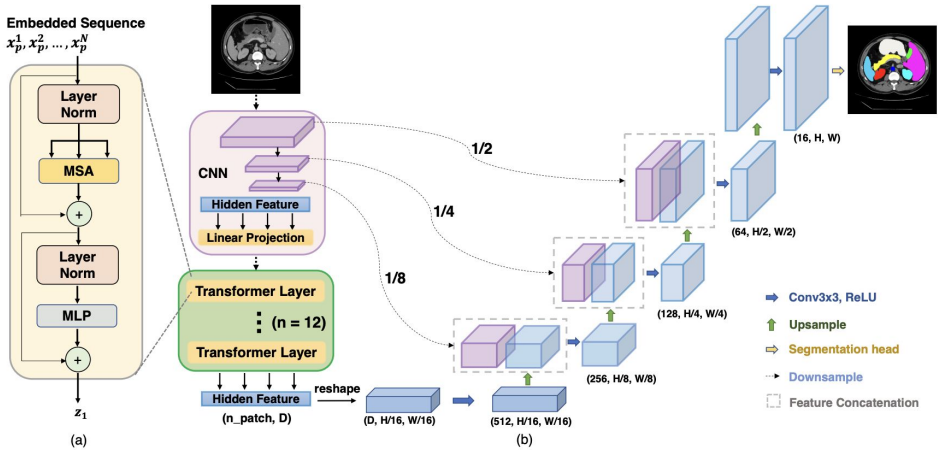


Fig. 1: Overview of the framework. (a) schematic of the Transformer layer; (b) architecture of the proposed TransUNet.

https://arxiv.org/pdf/2102.04306.pdf

DINO

In DINO, transformers were shown to generate segmentation maps **without** any supervision.

https://arxiv.org/pdf/2104.14294.pdf

Videos

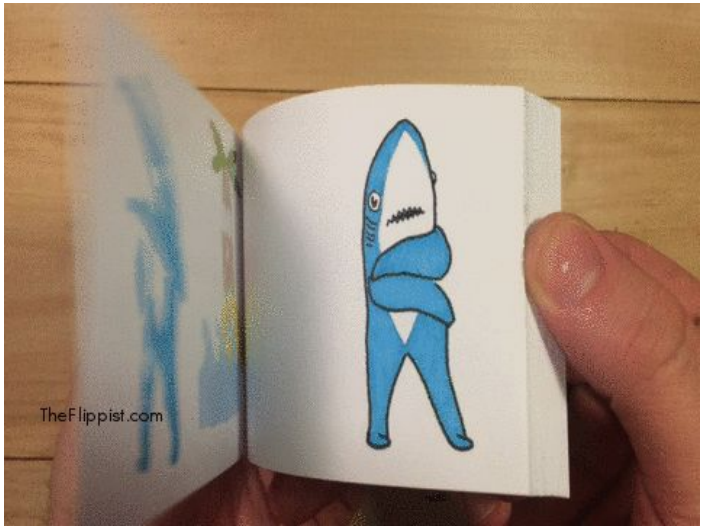
From images to videos

What is a Video?

A video can be thought of as a **series** of **images** (*frames*) over **time** (*seconds*)*.

Typical frame rates, i.e. number of images per second, is ~30 *frames per second* (FPS).

The stream of images give our eyes the **illusion** of fluid movement.



Source

*ignore the sound for now.

Videos

How can we use deep learning on **videos**?

Since videos are just **streams** of images, we could just apply image models on **each frame**, independently!

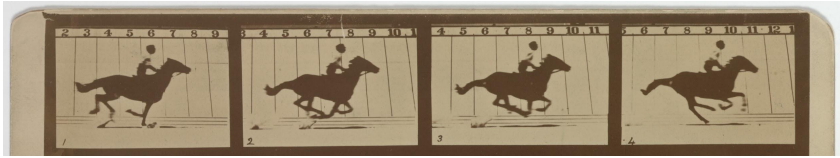
In fact, this is (basically) how cameras work - they capture each frame one at a time.

Photo by Alexander Andrews on Unsplash

Videos

Suppose we record the video of an animal and want to **identify** which animal is in the video.

We could analyze each frame **independently** through a single model.



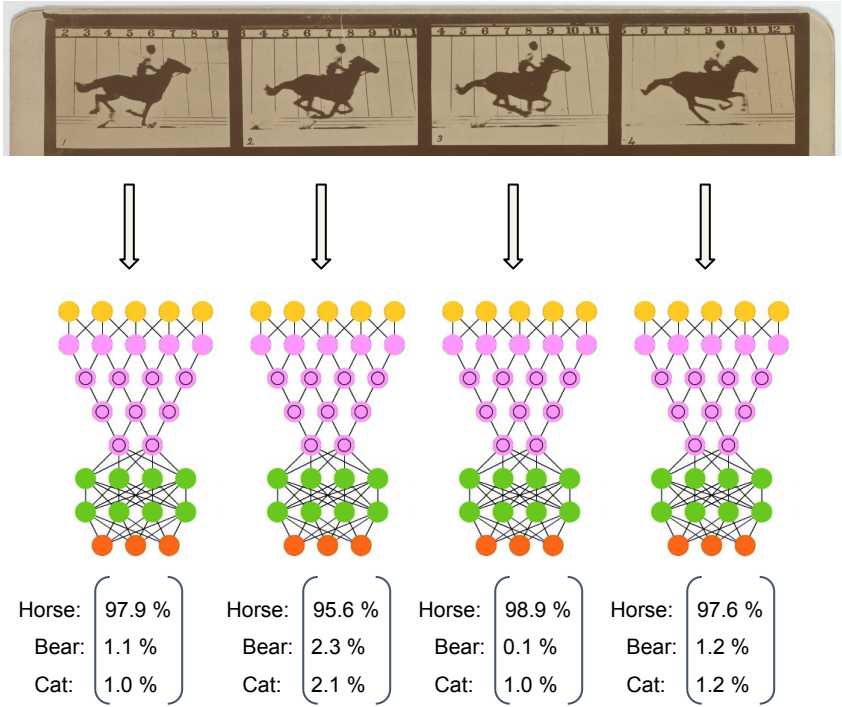
Horse: ??? %
Bear: ??? %
Cat: ??? %

Image

Videos

Suppose we record the video of an animal and want to **identify** which animal is in the video.

We could analyze each frame **independently** through a single model.



p(horse in video) = (97.9 + 95.6 + 98.9 + 97.6) / 4 = 97.5%

Image

Videos

This example shows segmentation in **real-time**. Each frame is segmented **independently**.

This approach **can be suitable** for **object detection** since temporal information rarely affects “objectness”.

This particular model was **trained** on single images, not videos.

source

Videos

The previous approaches assume time doesn't impact much the task at hand. However, some tasks **require** temporal information:

- Action recognition
- Self-driving cars
- Robotics
- Etc.

source

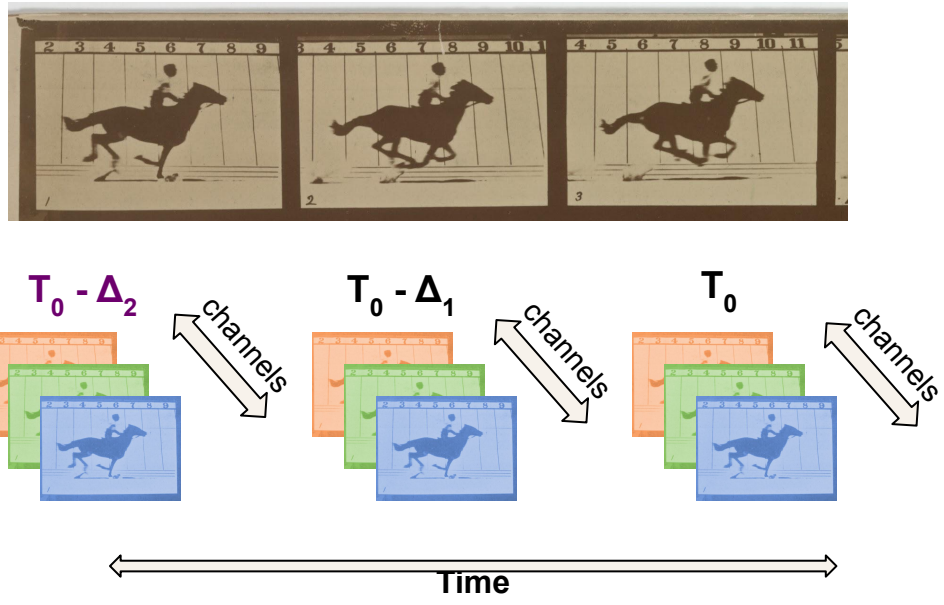
Videos

How do we **incorporate time** information to a **neural network**?

Photo by Heather Zabriskie on Unsplash

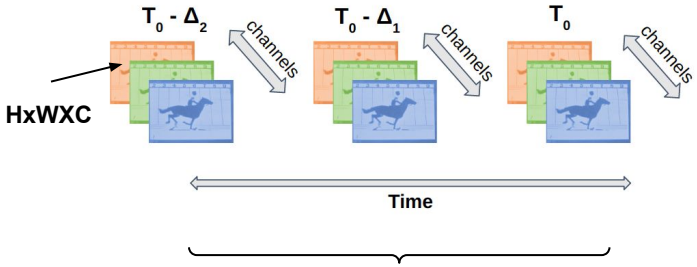
Videos

Recall that an image is a **stack** of three channels (RGB) and that a video is a **stack** of multiple images (frames) over time.



Image

Videos

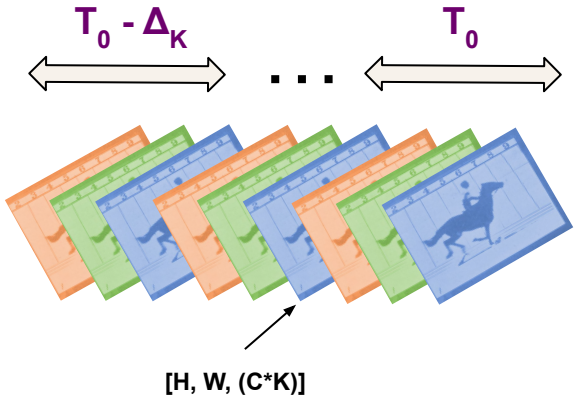


One approach is to **concatenate** past & present imagery to a single input to a CNN along the channel dimension.

Assume each image is of shape **[H, W, C]**, and we have **K** frames.

We would then have an input of shape

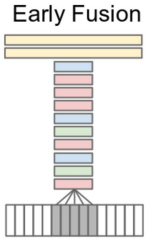
[H, W, (C*K)]

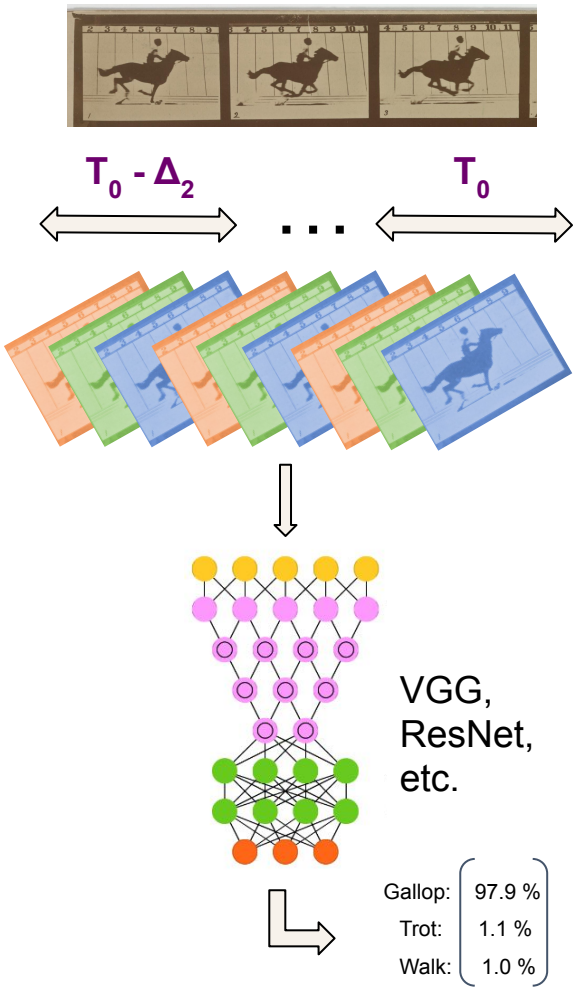


Videos

Using this approach, you could adapt almost any image CNN classifier architecture (VGG, ResNet, etc.) by changing the number of input channels of the architecture.

This is known as **early fusion**.



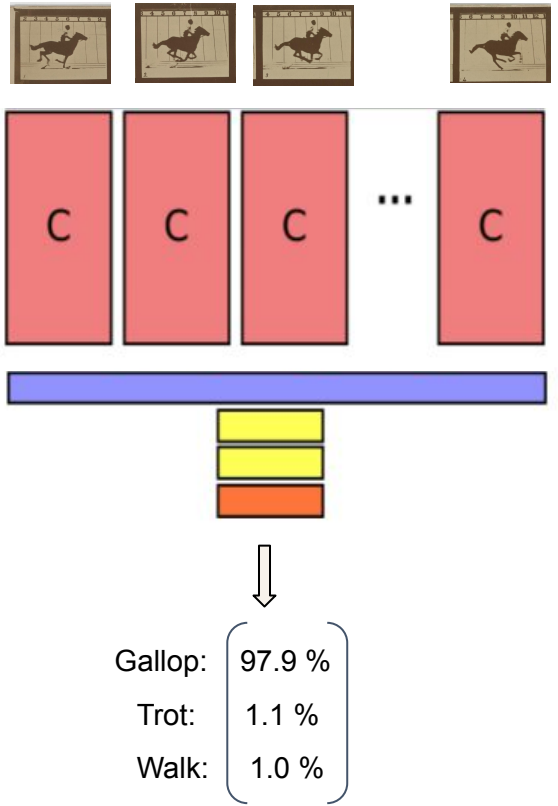


http://vision.stanford.edu/pdf/karpathy14.pdf

Videos

Alternatively, you could use a **shared** CNN backbone to extract features from independent frames and pool them later on in the network. This is known as **late fusion.**

This allows the use of **pre-trained networks** on large image datasets (e.g. ImageNet).

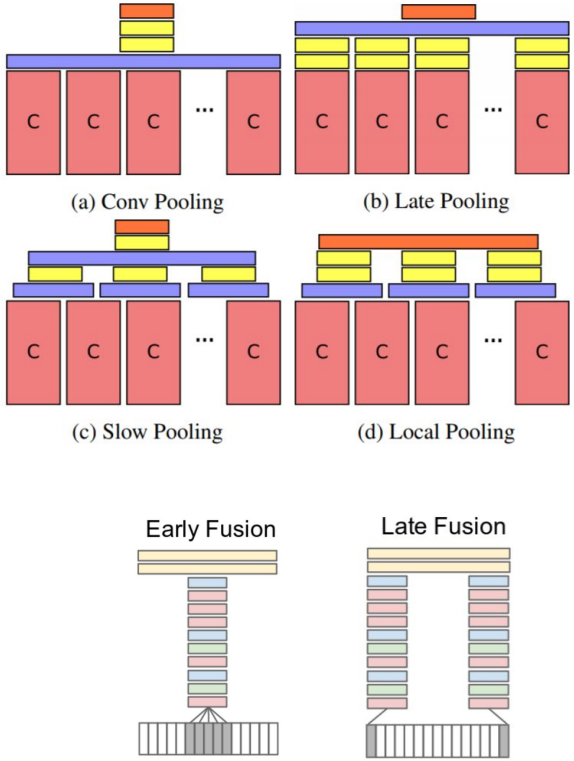


https://arxiv.org/pdf/1503.08909.pdf

Videos

There are **many ways** in which you could vary this setup.

You do not have to consider all **successive** frames either, as is shown in the late fusion model.



https://arxiv.org/pdf/1503.08909.pdf
http://vision.stanford.edu/pdf/karpathy14.pdf

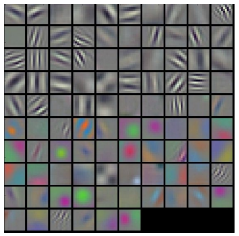
Conv3D

It can be useful to learn patterns of **motion**, just like it can be useful to learn patterns of **objects**. This can be achieved with **3D convolutions.**

2D Convolutions

Patterns of objects

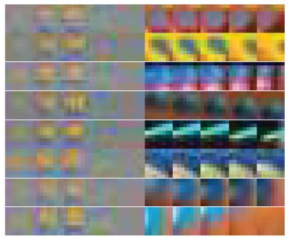
- Edges, contrasts
- ears, eyes noses



3D Convolutions

Patterns of motion

- Speed
- Movement
- Deformation
- Color changes



https://arxiv.org/pdf/1412.0767.pdf

Conv3D

In a **2D** convolution, the convolution operation takes place over **space** (x,y). In a **3D** convolution, we convolve over **space and time** (t).

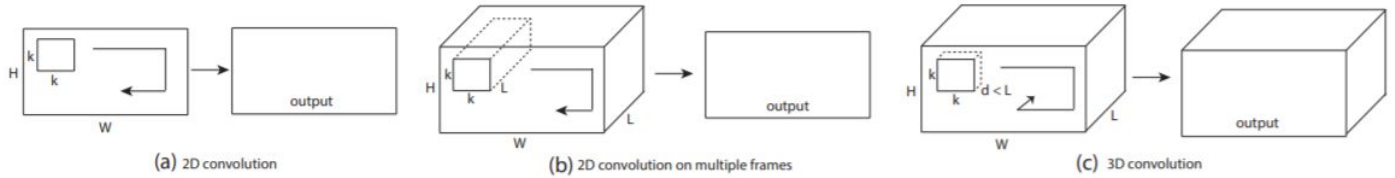


Figure 1. **2D and 3D convolution operations**. a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) Applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.

https://arxiv.org/pdf/1412.0767.pdf

Conv3D

Very much like 2D Convolutions, 3D convolutions can be stacked and used sequentially:

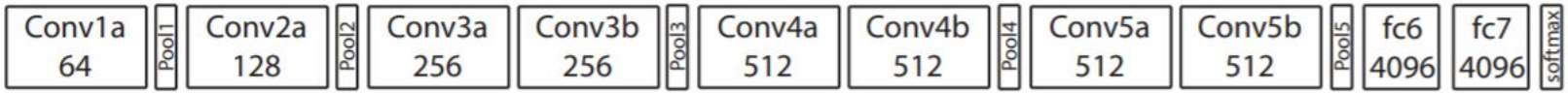


Figure 3. **C3D architecture**. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling layers are denoted from `pool1` to `pool5`. All pooling kernels are $2 \times 2 \times 2$, except for `pool1` is $1 \times 2 \times 2$. Each fully connected layer has 4096 output units.

https://arxiv.org/pdf/1412.0767.pdf

Conv3D

Luckily, 3D Convolutions are **common** in deep learning APIs - no need to reinvent the wheel.

1 video (batch size = 1), with 3 channels (RGB) of 30 frames (e.g. 1 second at 30 fps) with each frame being [height, width] dimensions (128x128)

```
1 import torch
2 m = torch.nn.Conv3d(3, 32, (5,5,5), stride=1)
3 input = torch.randn(1, 3, 30, 128, 128)
4 # input = [batch, channel, time, height, width]
5 out = m(input)
6 print("input shapes: ", input.shape)
7 print("weight shapes: ", m.weight.shape)
8 print("output shapes: ", out.shape)
```

```
input shapes:  torch.Size([1, 3, 30, 128, 128])
weight shapes:  torch.Size([32, 3, 5, 5, 5])
output shapes:  torch.Size([1, 32, 26, 124, 124])
```

Weights are a tensor with 5 Dimensions: [out_channels, in_channels, time, height, width]

Image

Conv3D

However, 3D convolutions are **memory intensive**.

One strategy involves using 2D convolutions on a **frame level** and stacking features temporally before using 3D Convolutions.

This also allows for using pretrained networks on ImageNet

Model	Layer	Output Size	Pre-trained ResNet-50	VGG-M Like (Scratch)
Base Network	conv1	$112\times112\times c_1$	7×7, 64, stride 2	
			3×3 max pool, stride 2	
	conv2_x	$56\times56\times c_2$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times1$
	conv3_x	$28\times28\times c_3$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times1$
	conv4_x	$14\times14\times c_4$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times3$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times1$
Embedder Network	Temporal Stacking	$k\times14\times14\times c_4$	Stack k context frame features in time axis	
	conv5_x	$k\times14\times14\times512$	$\begin{bmatrix}3\times3\times3, 512\\3\times3\times3, 512\end{bmatrix}\times1$	
	Spatio-temporal Pooling	512	Global 3D Max-Pool	
	fc6_x	512	$\begin{bmatrix}512\\512\end{bmatrix}\times1$	
	Embedding	128	128	

Table 8: Architectures used in our experiments. The network produces an embedding for each frame (and its context window). c_i depends on the choice of the base network. Inside the square brackets, the parameters in the form of: (1) $[n \times n, c]$ refers to 2D Convolution filter size and number of channels respectively (2) $[n \times n \times n, c]$ refers to 3D Convolution filter size and number of channels respectively (3) $[c]$ refers to channels in a fully-connected layer. Downsampling in ResNet-50 is done using convolutions with stride 2, while in VGG-M models we use MaxPool with stride 2 for downsampling.

https://arxiv.org/pdf/1904.07846.pdf

Transformers for video

Just like with CNNs, there are many different strategies in which **time** can be incorporated to **vision transformers**.

In **ViVit**, they explore different strategies for video classification as shown in the following figures.

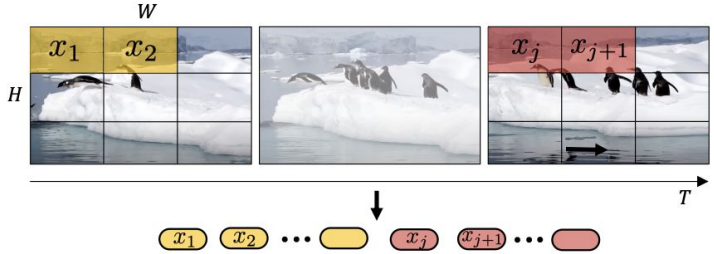


Figure 2: Uniform frame sampling: We simply sample n_t frames, and embed each 2D frame independently following ViT [17].

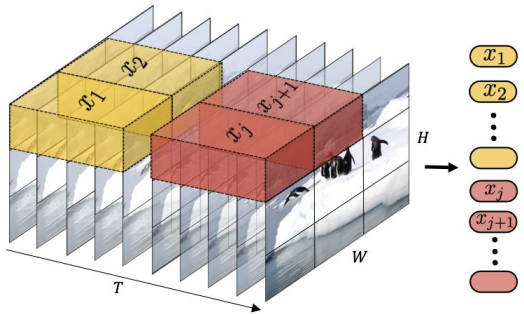


Figure 3: Tubelet embedding. We extract and linearly embed non-overlapping tubelets that span the spatio-temporal input volume.

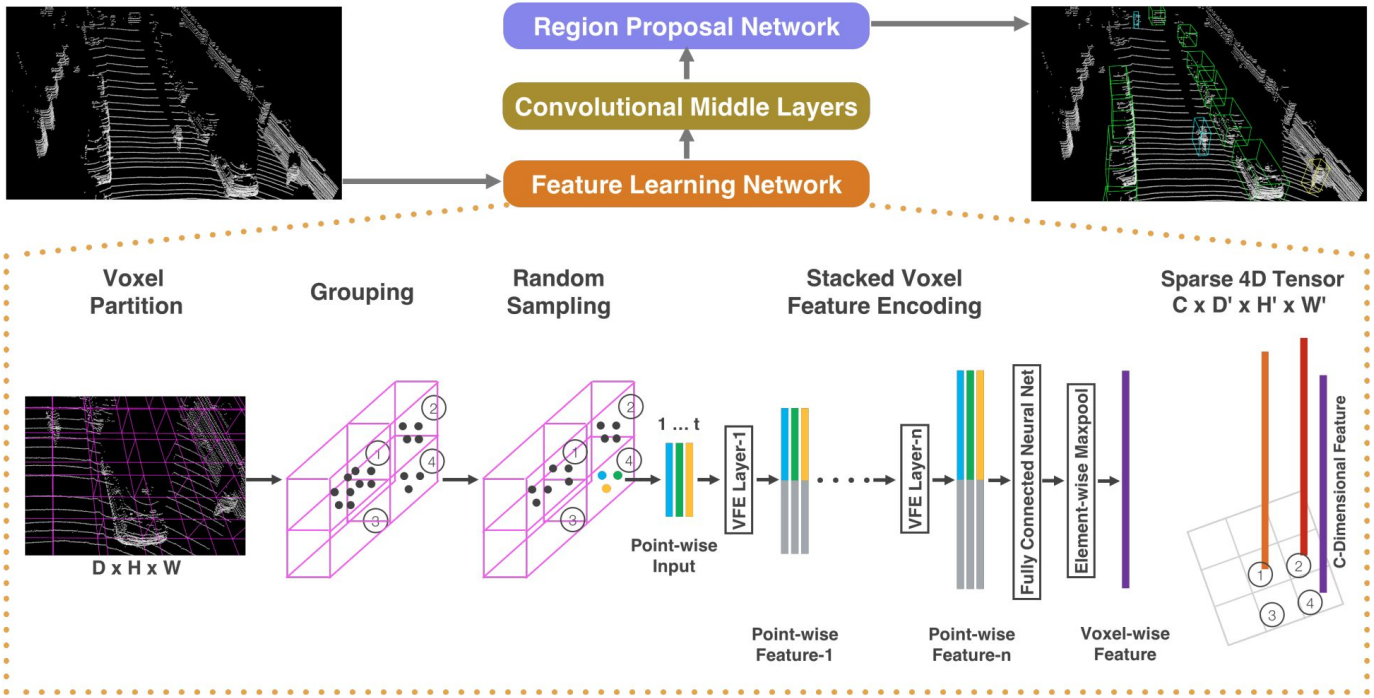
source

Lidar data

LiDAR (point cloud) processing

VoxelNet

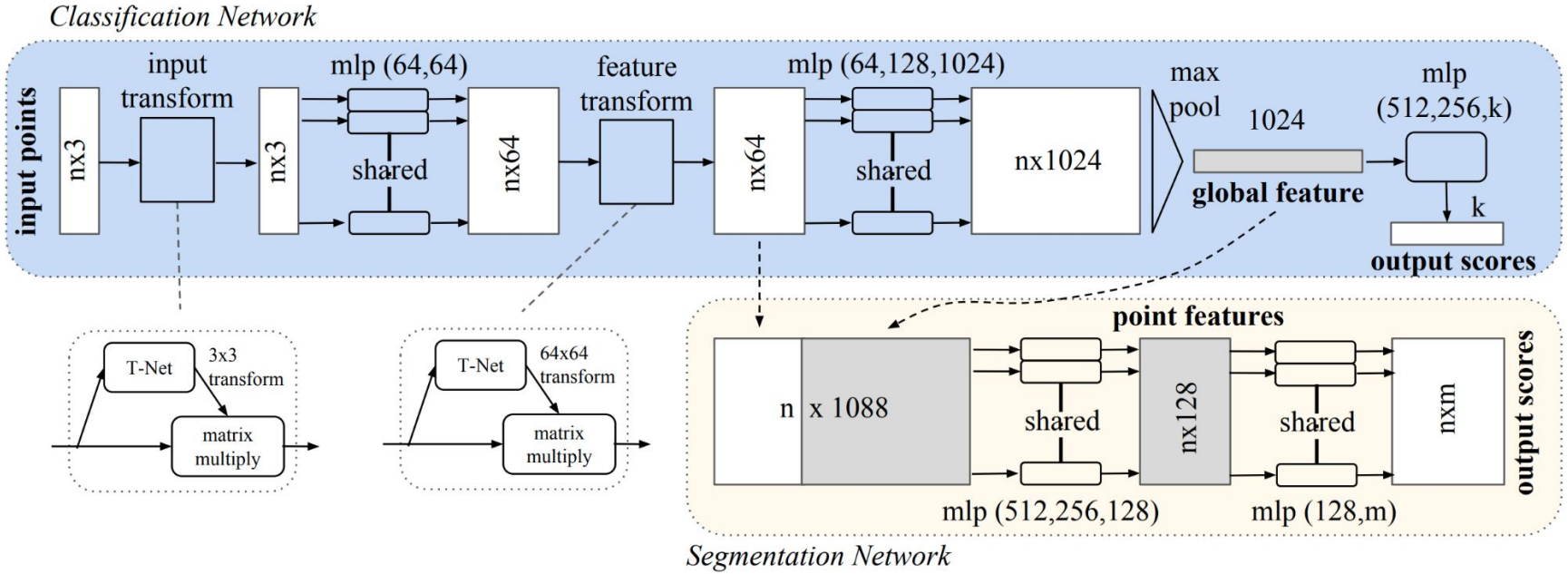
… voxelize the point cloud and use 3D encoders to process it!



LiDAR (point cloud) processing

PointNet

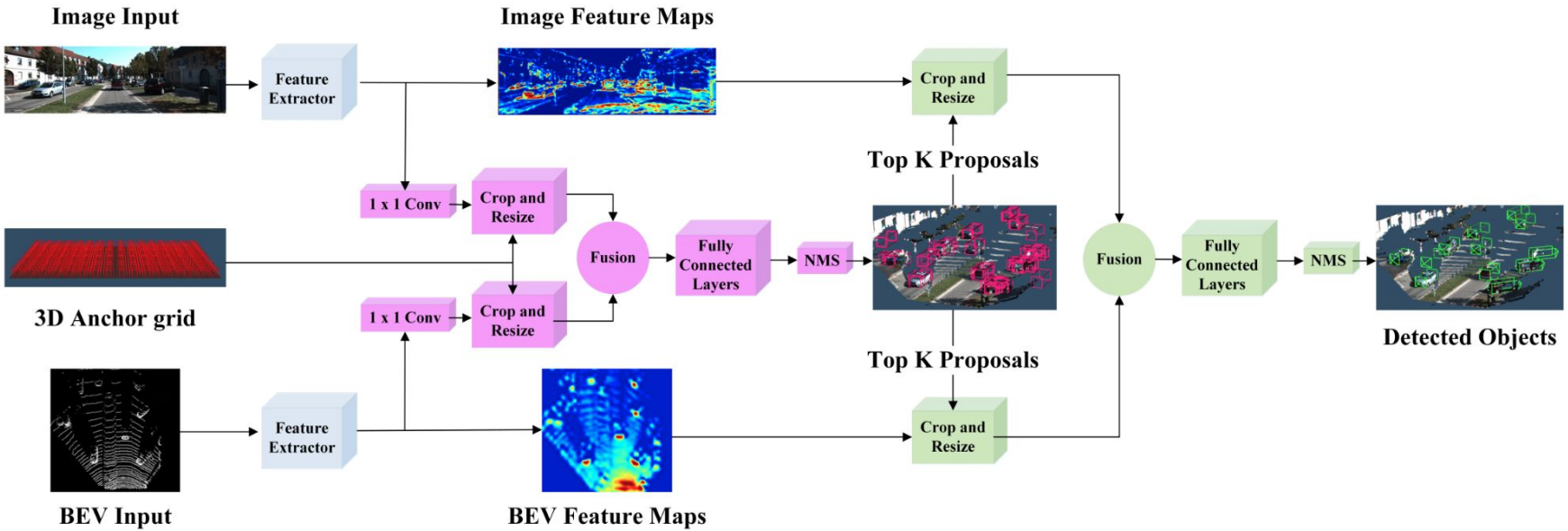
… use MLP-based blocks to process (small) arrays of points directly



LiDAR (point cloud) processing

AVOD

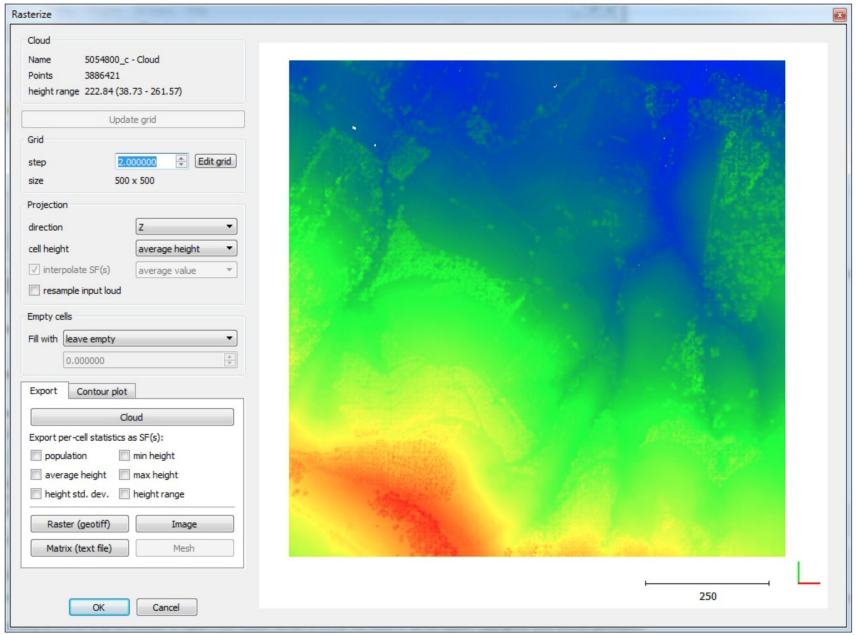
… use a projection (in this case: “bird’s eye view”, BoV) and feature fusion



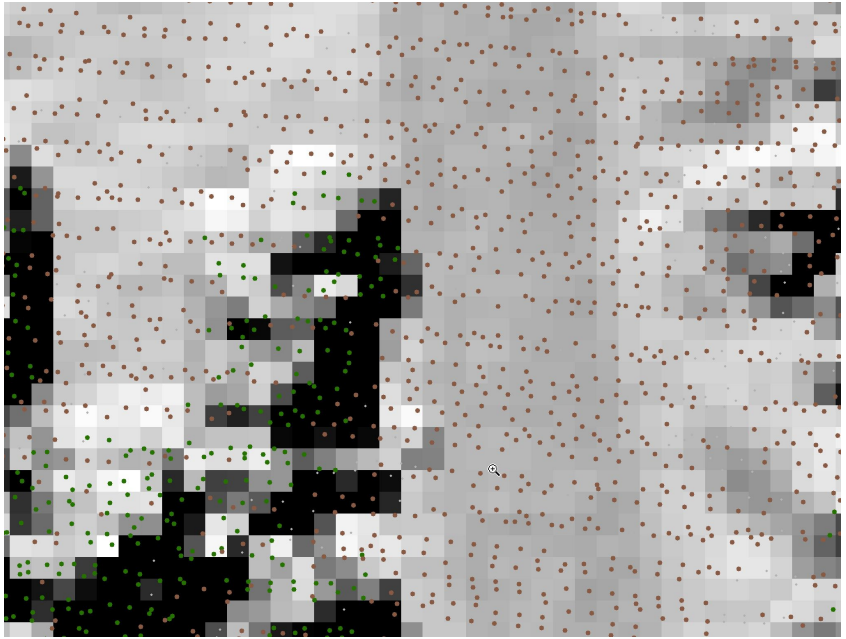
LiDAR (point cloud) processing

"Lazy approach":

… just rasterize the point data as an image, and use classic encoders!



CloudCompare



EarthDataScience.org

Questions?

jeremy.pinto@mila.quebec
pierreluc.stcharles@mila.quebec