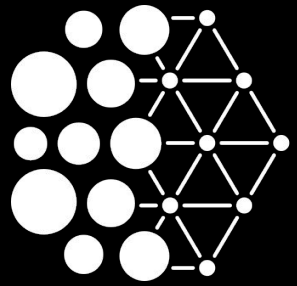
Quebec
Artificial
Intelligence
Institute



Mila

CNN - What's new
Week 3

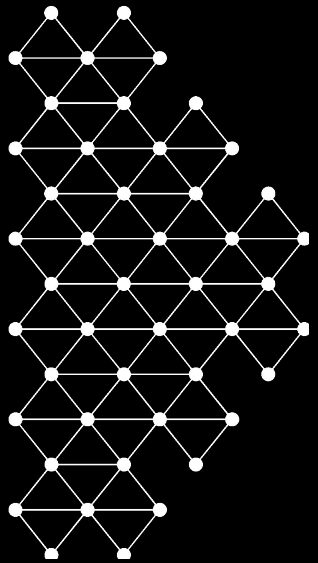
Jeremy Pinto
jeremy.pinto@mila.quebec

Content

- CNNs applied to videos
- Model interpretation
- ImageNet - Updates
- Transformers in Computer Vision

Photo by Jakob Owens on Unsplash

Quebec
Artificial
Intelligence
Institute



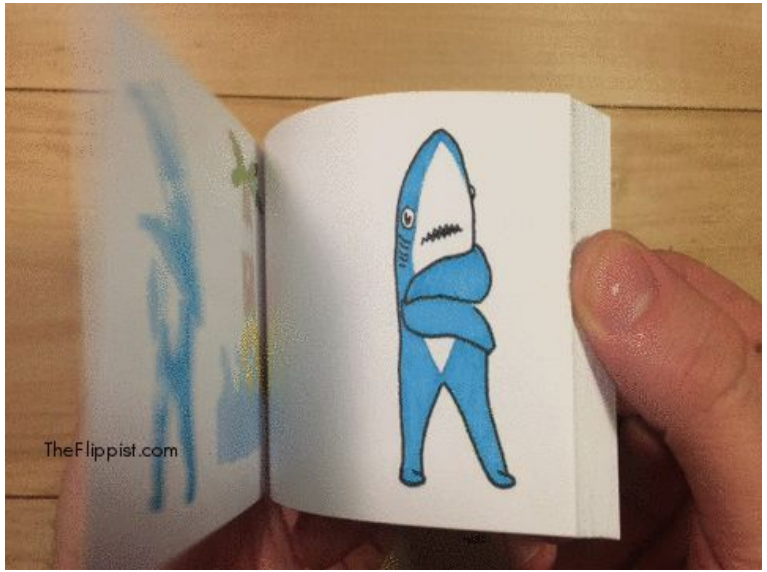
From images to videos

Videos

A video can be thought of as a **series** of **many images** over time (ignore the sound for now).

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.



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!

This is ok when time doesn't affect much the task at hand, for example in object detection and segmentation.

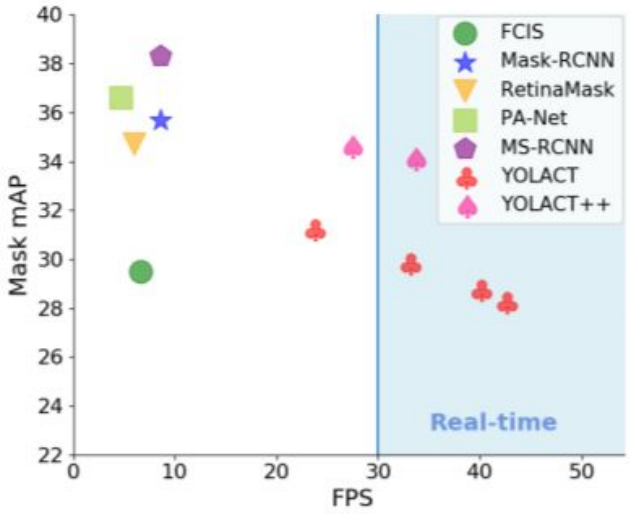
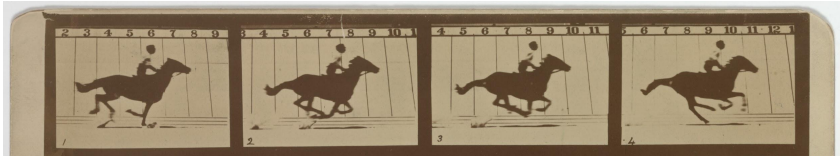


Fig. 1: Speed-performance trade-off for various instance segmentation methods on COCO. To our knowledge, ours is the first *real-time* (above 30 FPS) approach with over 30 mask mAP on COCO `test-dev`.

https://arxiv.org/pdf/1912.06218.pdf

Videos

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



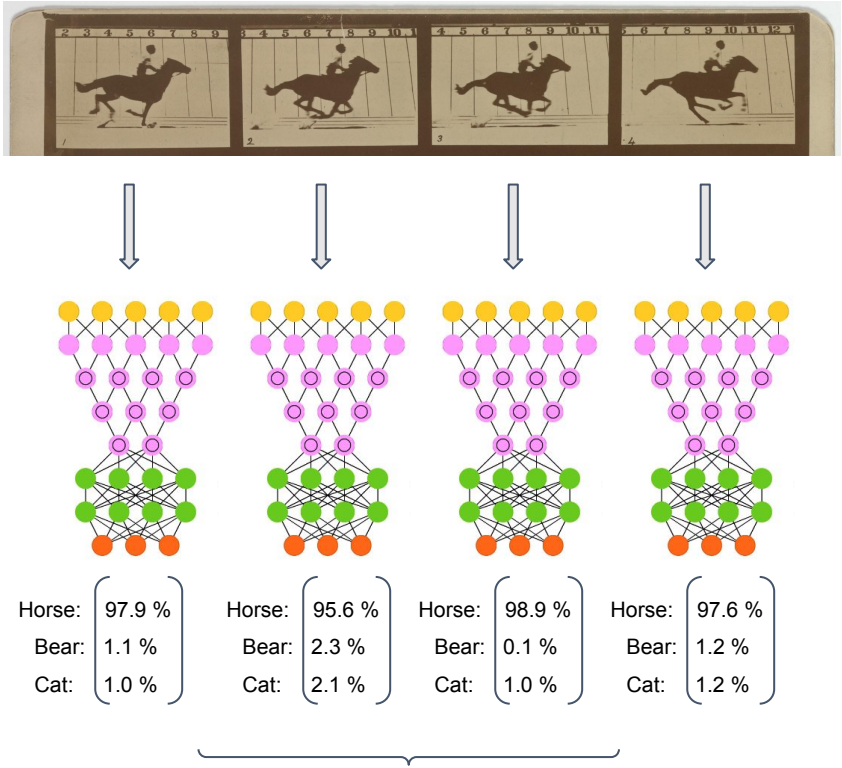
Horse: ??? %
Bear: ??? %
Cat: ??? %

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**.

Every frame is passed through the **same** model, and we can **combine** the results over time to predict what is in our video.



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

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”.

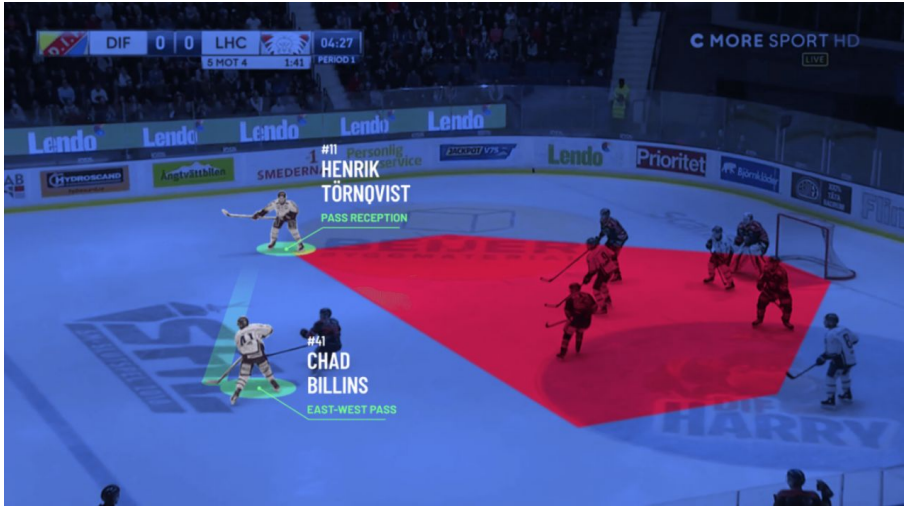
This particular model was **trained** on single images, not videos.

https://www.youtube.com/watch?v=0pMfmo8qfpQ

Videos

However, some tasks require temporal information:

- Action recognition
- Self-driving cars
- Robotics
- Video annotation (scene understanding)
- etc.



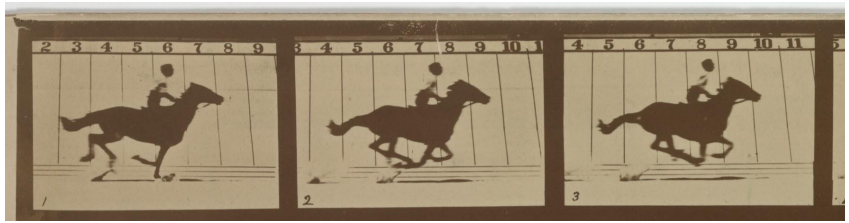
Videos

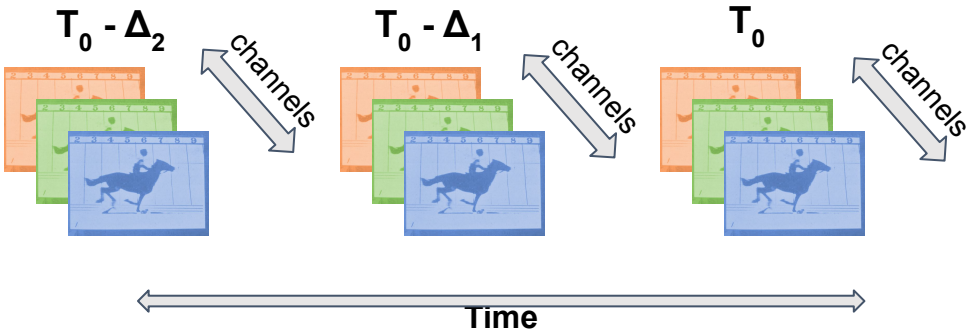
How do we **incorporate time** information to our neural networks?

Photo by Heather Zabriskie on Unsplash

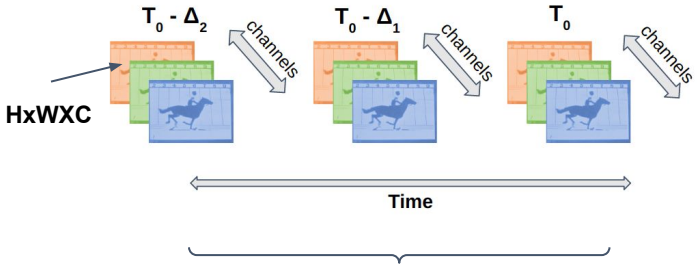
Videos

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





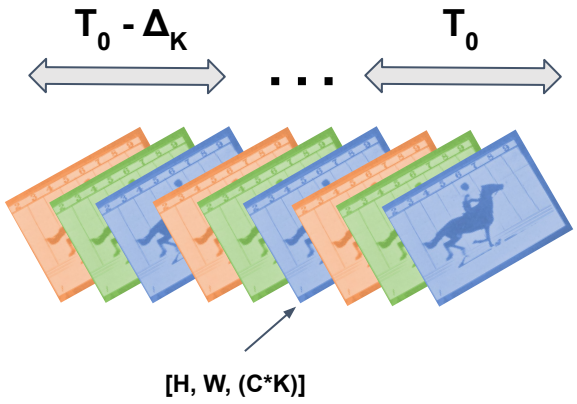
Videos



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, where **H** and **W** are the height and width of the input crops, **C** is the number of channels, and **K** is the number of time frames to use.

We would then have an input of shape **[H, W, (C*K)]**

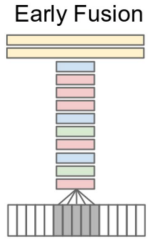


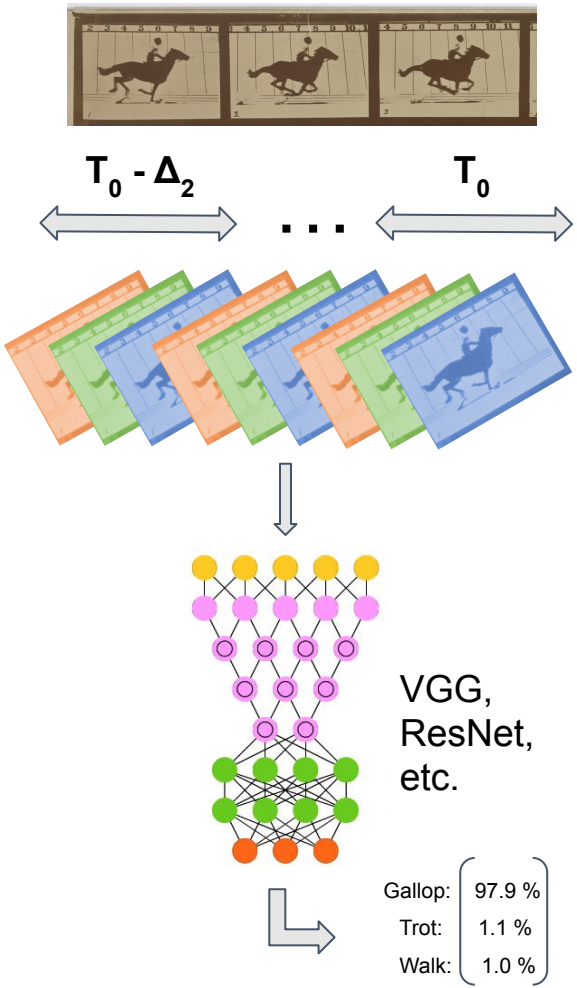
Mila

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**.



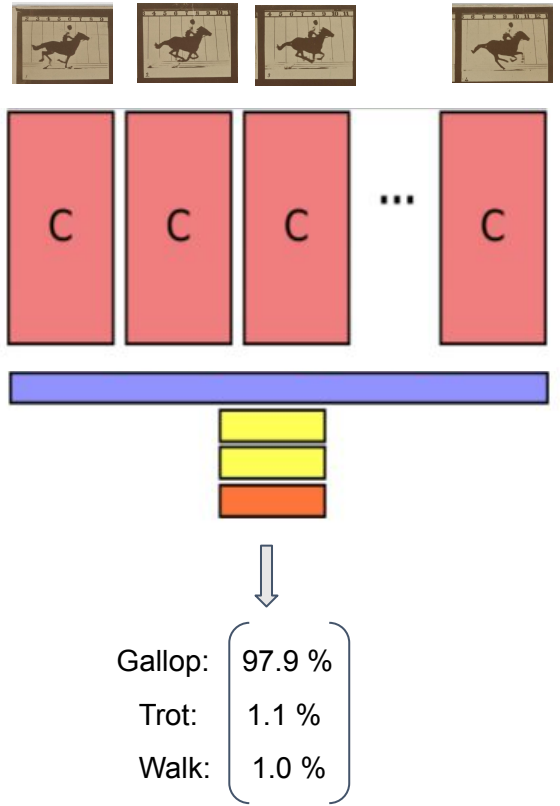


http://vision.stanford.edu/pdf/karpathy14.pdf

Videos

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

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

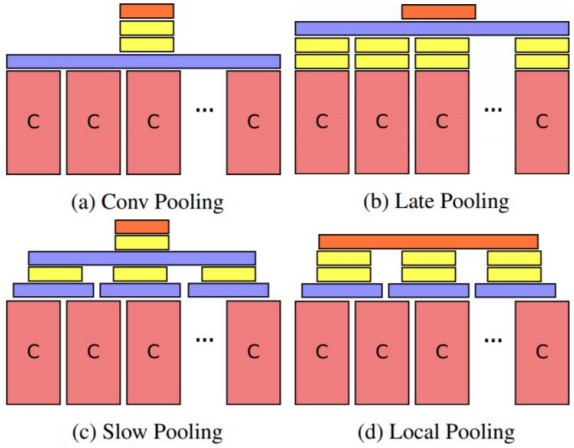


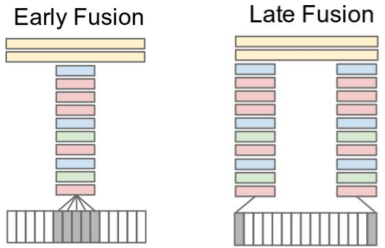
https://arxiv.org/pdf/1503.08909.pdf

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

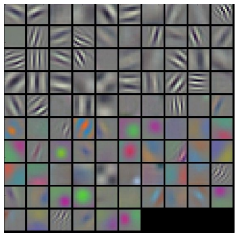
Conv3D

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

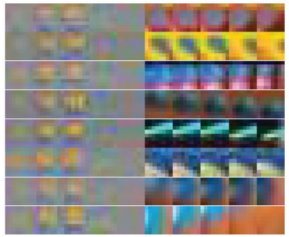
- Edges, contrasts
- ears, eyes noses



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).

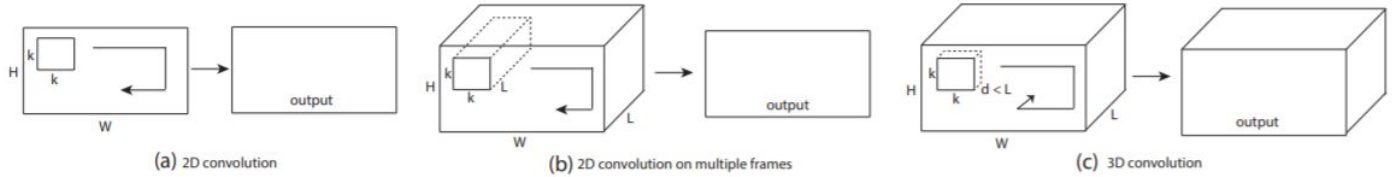


Figure 1. **2D and 3D convolution operations**. a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) Applying 3D convolution on a video volume results in another volume, preserving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

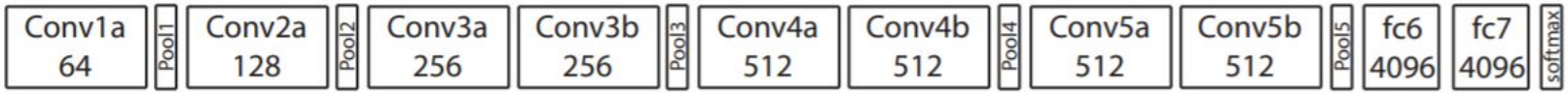


Figure 3. **C3D architecture**. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling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

Here is an example of how inception-like modules can be adapted to 3D convolutions.

3D convolutions extend very naturally from 2D convolutions.

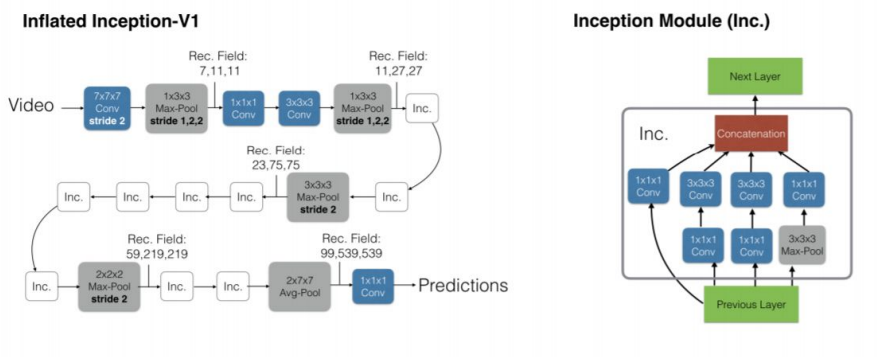


Figure 3. The Inflated Inception-V1 architecture (left) and its detailed inception submodule (right). The strides of convolution and pooling operators are 1 where not specified, and batch normalization layers, ReLu's and the softmax at the end are not shown. The theoretical sizes of receptive field sizes for a few layers in the network are provided in the format "time,x,y" – the units are frames and pixels. The predictions are obtained convolutionally in time and averaged.

https://arxiv.org/pdf/1705.07750.pdf

Conv3D

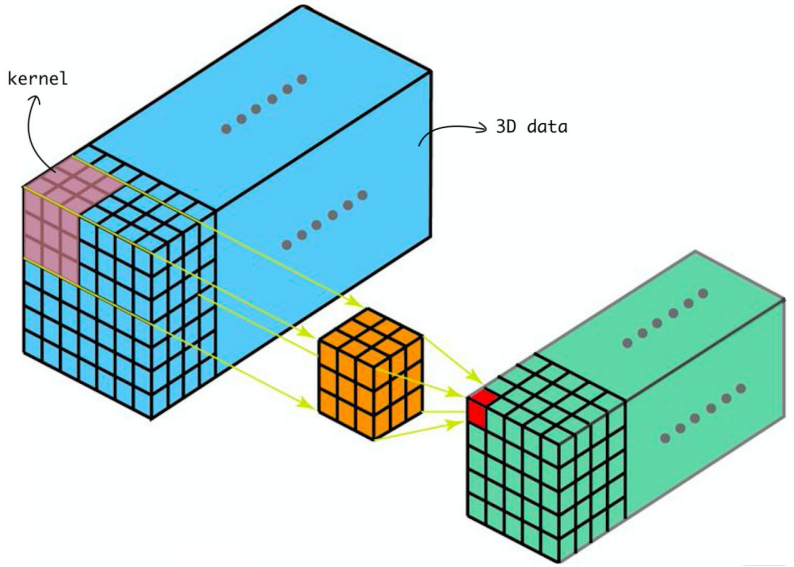
3D Convolutions are **common** in most standard deep learning APIs.

```
import torch
m = torch.nn.Conv3d(16, 32, (3,3,3), stride=1)
input = torch.randn(1, 16, 30, 128, 128)
out = m(input)
```

Note that in 2D convolutions, kernels are 3D. In 3D convolutions, kernels are 4D. When we stack many 3D kernels, we stack them along a 5th dimension (the output dimension).

Inputs = (batch, chanels, time, height, width)

```
input shapes:  torch.Size([1, 16, 30, 128, 128])
weight shapes:  torch.Size([32, 16, 3, 3, 3])
output shapes:  torch.Size([1, 32, 28, 126, 126])
```



Source

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 imagenet networks

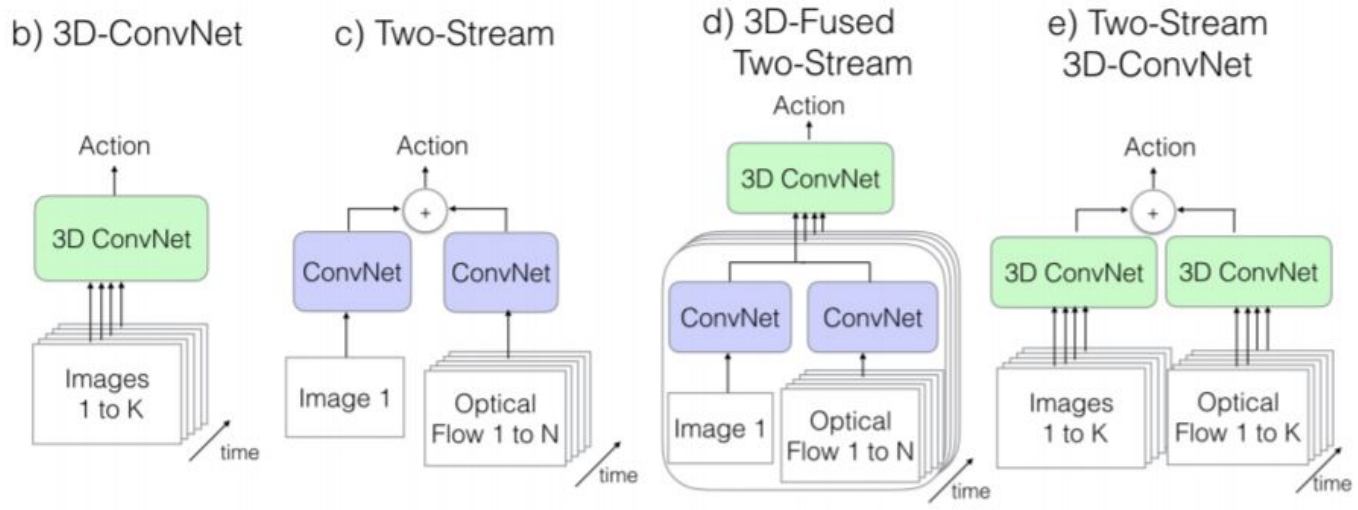
Model	Layer	Output Size	Pre-trained ResNet-50	VGG-M Like (Scratch)
Base Network	conv1	$112\times112\times c_1$	7×7, 64, stride 2	
			3×3 max pool, stride 2	
	conv2_x	$56\times56\times c_2$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times1$
	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

Two-stream networks

Another strategy involves using **priors** like optical flow as inputs to the neural network. This is known as two-stream networks and were shown to generalise well.



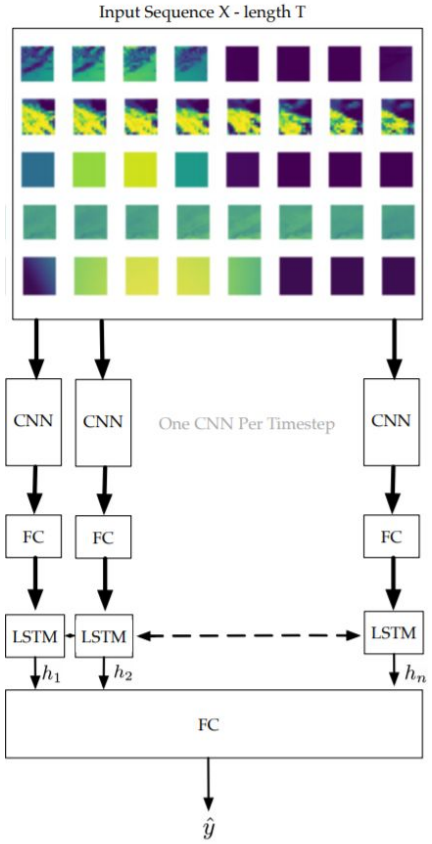
https://arxiv.org/pdf/1705.07750.pdf

CNN + LSTM

Another strategy is to use **2D convolution** features as an input to recurrent models (like LSTMs).

This also allows for using **pretrained** ImageNet networks.

You will see **recurrent models** in more detail on week 4 with Mirko.



https://arxiv.org/abs/1902.01453

ConvLSTM

2D convolutions can also be passed through a modified **LSTM** which accepts matrices as input (instead of vectors). The model was proposed to predict rainfall in the near future.

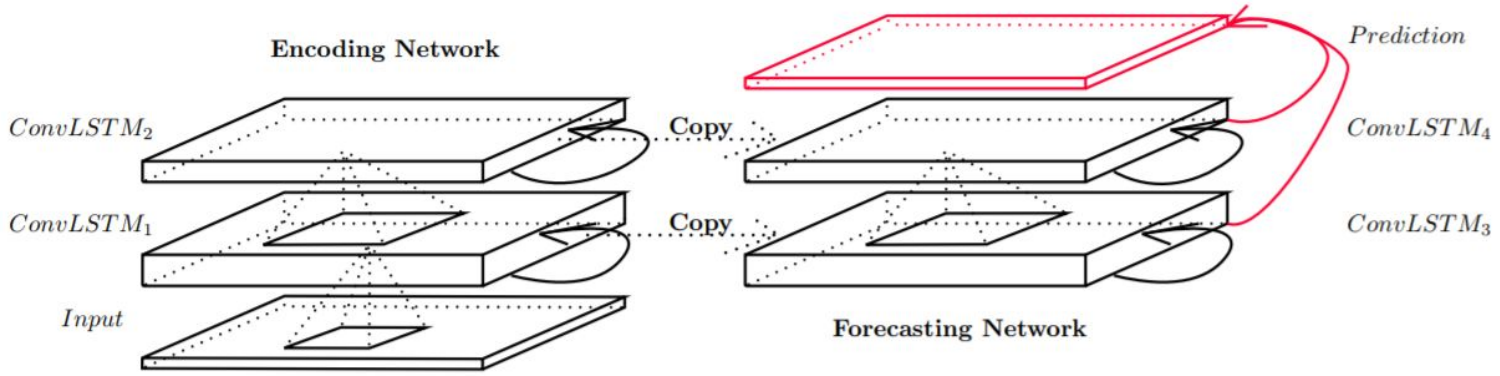
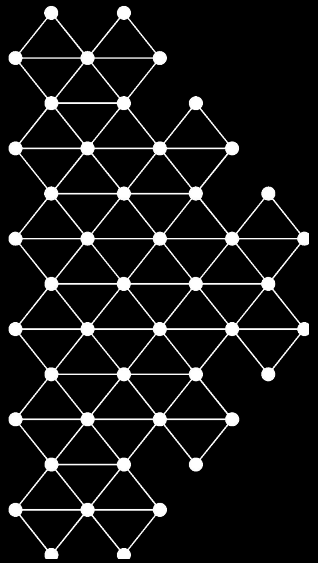


Figure 3: Encoding-forecasting ConvLSTM network for precipitation nowcasting

https://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf

Quebec
Artificial
Intelligence
Institute

Mila



Model interpretation

Model interpretation

Due to the inherent structure of CNNs, it can be very difficult to **interpret how** they make decisions.

There are nevertheless certain techniques which help us **understand** what is happening **under the hood**.

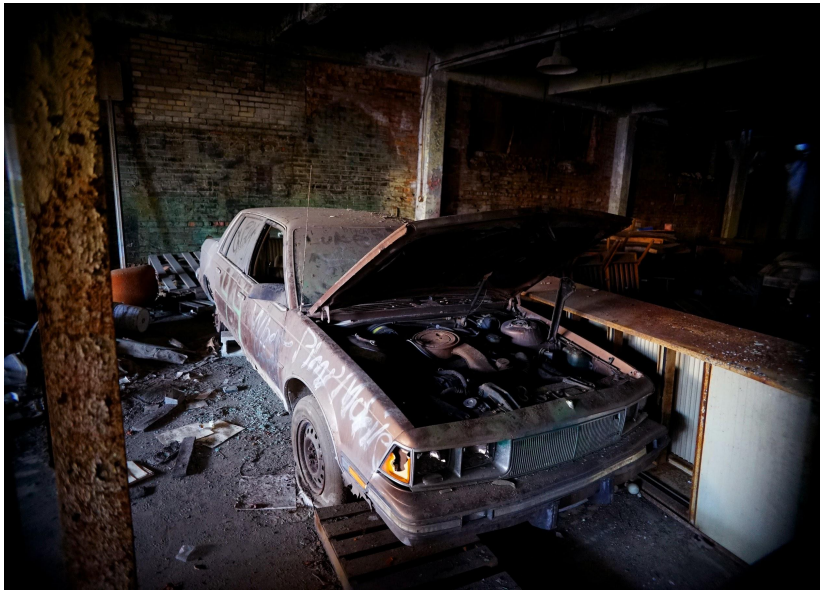
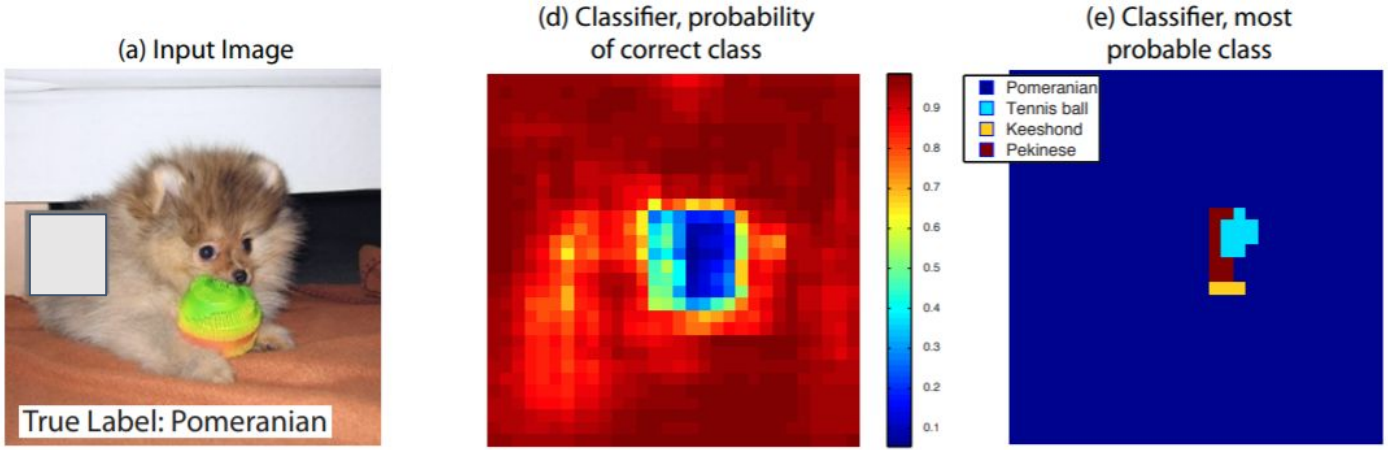


Photo by Zach Vessels on Unsplash

Model interpretation

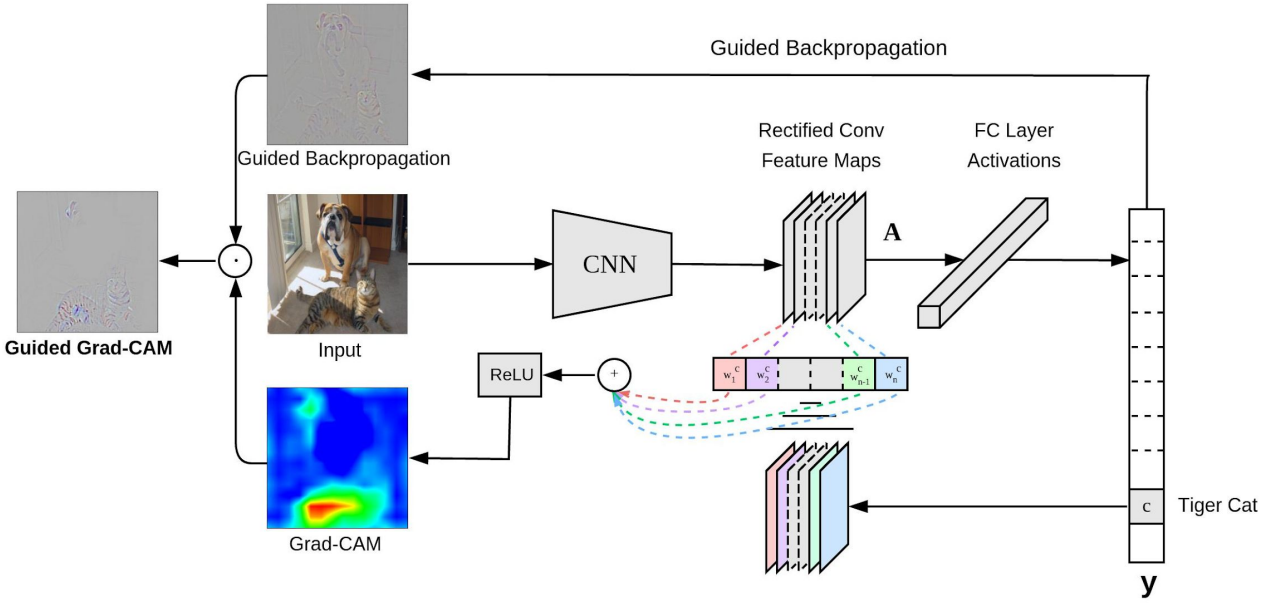
One strategy involves **blocking** out inputs of an image and looking at how the predictions vary over the network. These are known as **perturbation methods**.



https://arxiv.org/pdf/1311.2901.pdf

Model interpretation

Other methods, such as **GradCam**, use the information of the **gradient** of a single-class prediction to interpret what the model is looking at.



https://arxiv.org/abs/1610.02391

Model interpretation

This can be very useful to **identify bias** attributed to certain classes in a network:

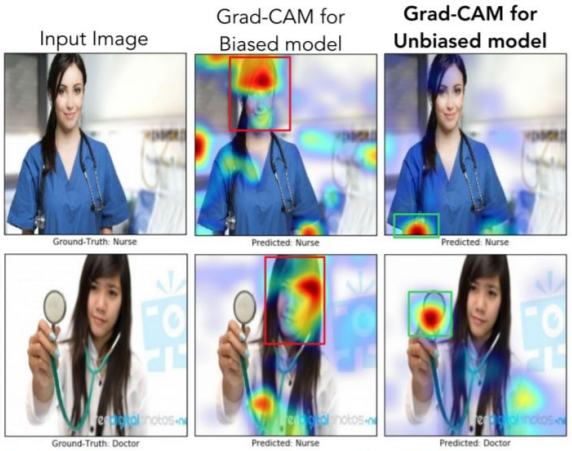


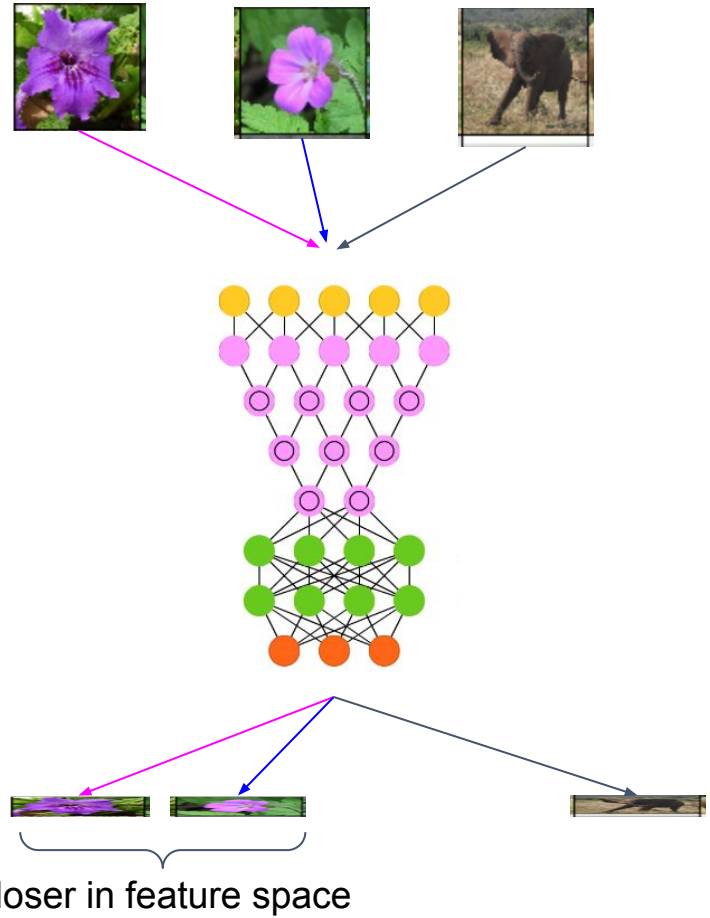
Fig. 8: In the first row, we can see that even though both models made the right decision, the biased model (model1) was looking at the face of the person to decide if the person was a nurse, whereas the unbiased model was looking at the short sleeves to make the decision. For the example image in the second row, the biased model made the wrong prediction (misclassifying a doctor as a nurse) by looking at the face and the hairstyle, whereas the unbiased model made the right prediction looking at the white coat, and the stethoscope.

https://arxiv.org/abs/1610.02391

Model interpretation

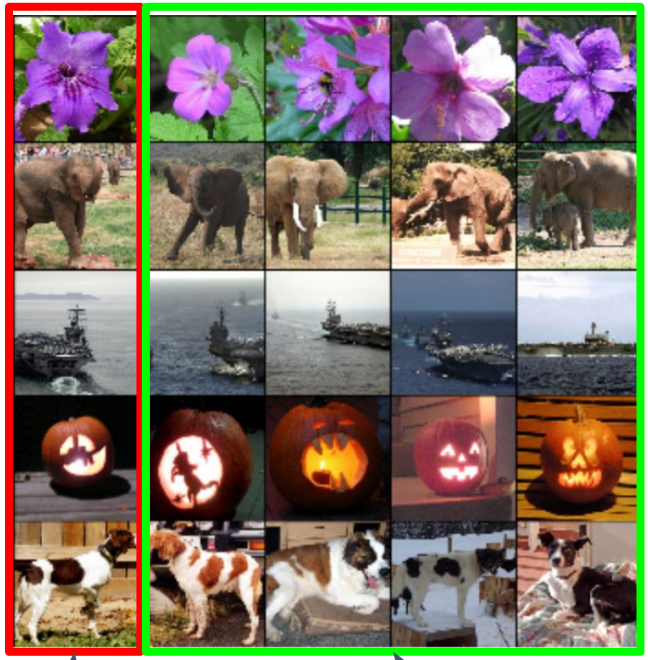
Another useful trick to visualize what your model might be picking up on is by analyzing the **feature space**.

For example, in AlexNet, the authors reported the images that were closest in the **feature space** to the query sample (e.g. using a dot product).



Model interpretation

Analyzing the correlations in feature space can help **debug** if your network is learning “meaningful” representations or not for your given task.



Query

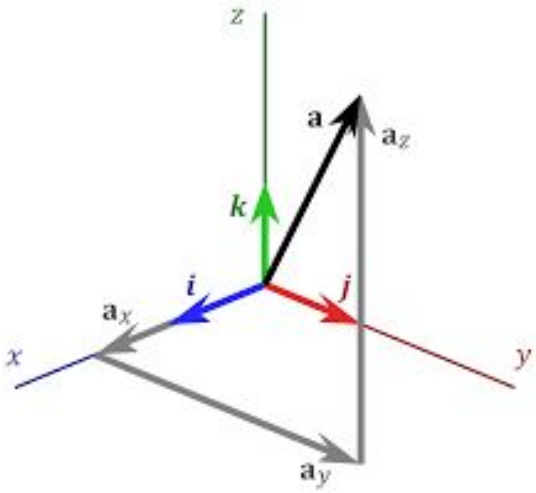
Closest matches in feature space

Model interpretation

Dimensionality reduction can also be used to visualize what your model is learning.

Vectors in the feature space are usually of very **high dimensions** (e.g. 1024x1).

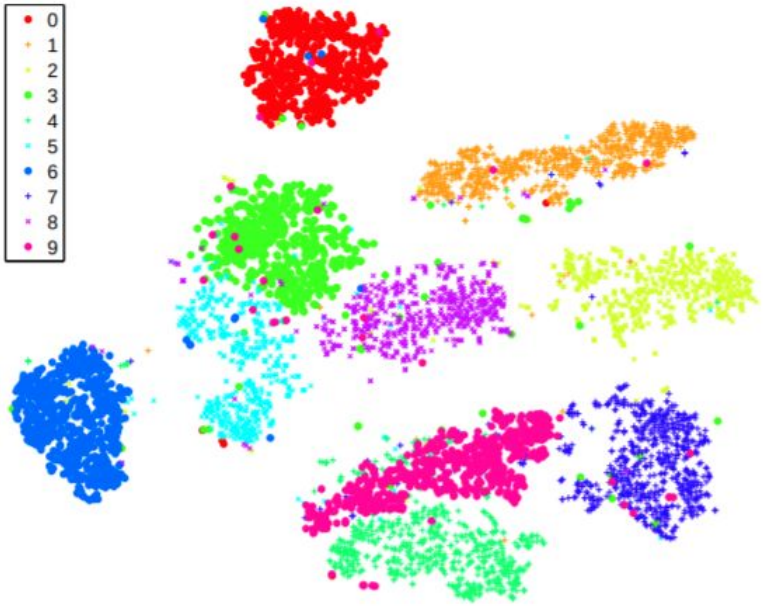
These numbers are abstract, non-linear relationships that the network learned. They are very difficult to visualize.



Model interpretation

Dimensionality reduction allow us to **preserve relationships** while **significantly reducing** the number of dimensions of the vectors. Some common methods used are:

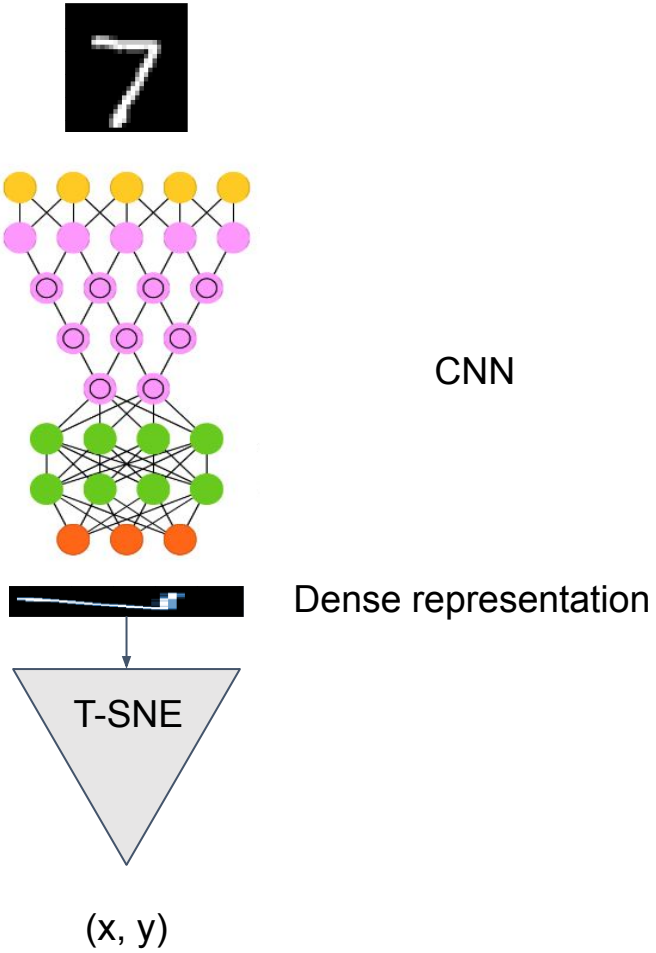
- PCA
- t-sne
- autoencoders



(a) Visualization by t-SNE.

Model interpretation

In this example, we reduce the feature vectors in MNIST using t-sne to 2 dimensions.

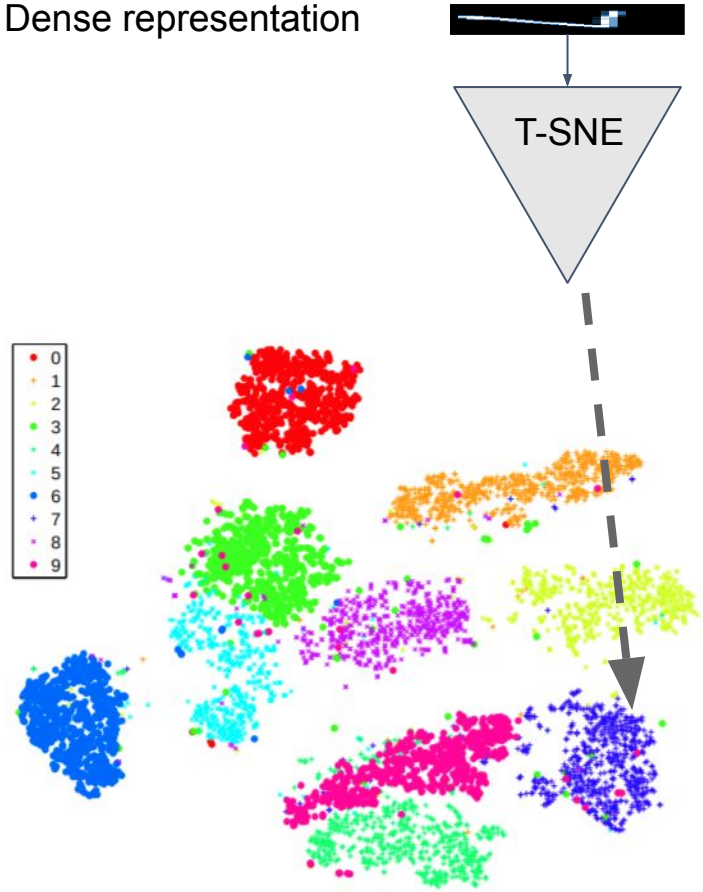


https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

Model interpretation

In this example, we reduce the feature vectors in MNIST using t-sne to 2 dimensions.

By plotting the 2 features and color-coding by category, we observe clusters.



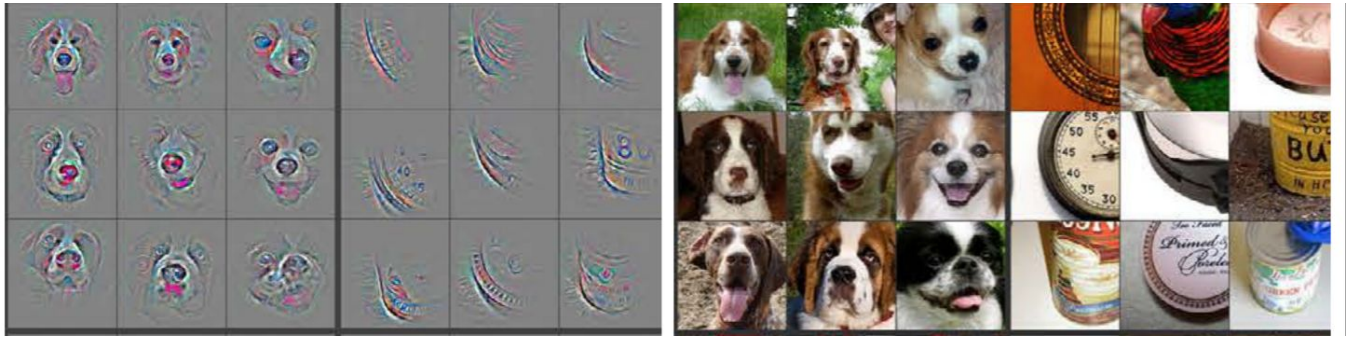
https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

Model interpretation

In the context of images and text, dimensionality reduction can really be a useful tool to see if meaningful clusters emerge.

Model interpretation

We can also visualize the most prominent **features** in various layers of a network using **transposed convolutions**. It is helpful to see what features contribute most to classification.



https://arxiv.org/pdf/1311.2901.pdf

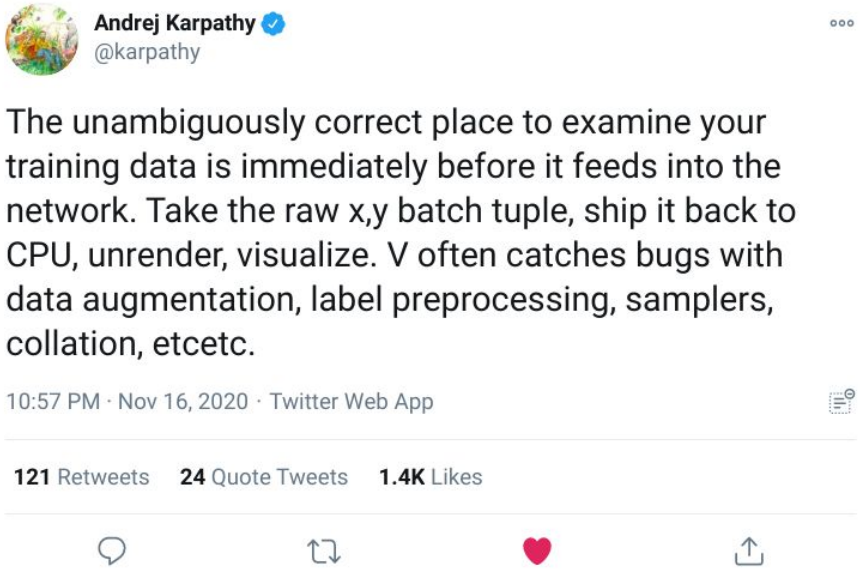
Model interpretation

- Showcase work of Colah et al. on feature representations
- Show latest results of CLIP + model interpretation

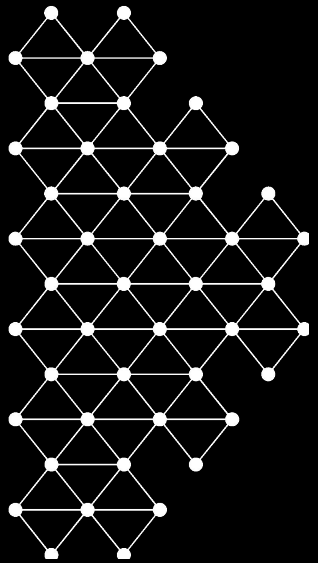
https://arxiv.org/pdf/1311.2901.pdf

Model Interpretation

As an aside, this is always a **great tip** to keep in mind for debugging your CNNs.



Quebec
Artificial
Intelligence
Institute

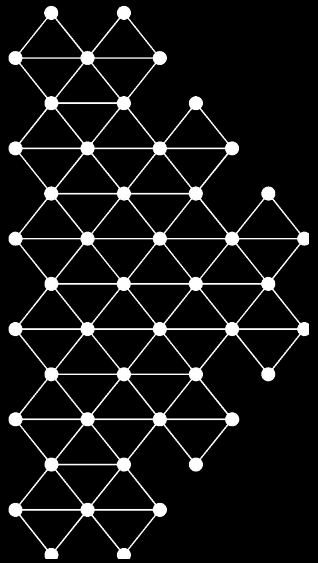


Adversarial Attacks

Adversarial Attacks

Learn how robust your model is

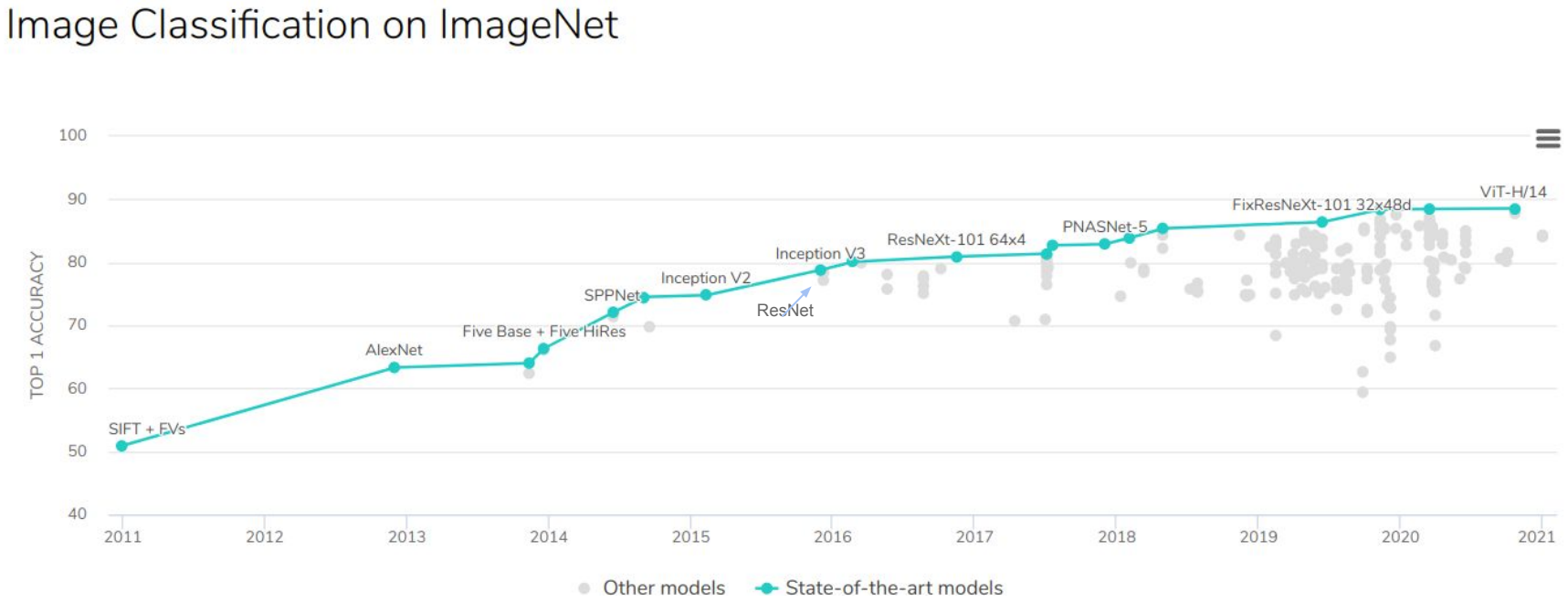
Quebec
Artificial
Intelligence
Institute



ImageNet (Updates)

ImageNet

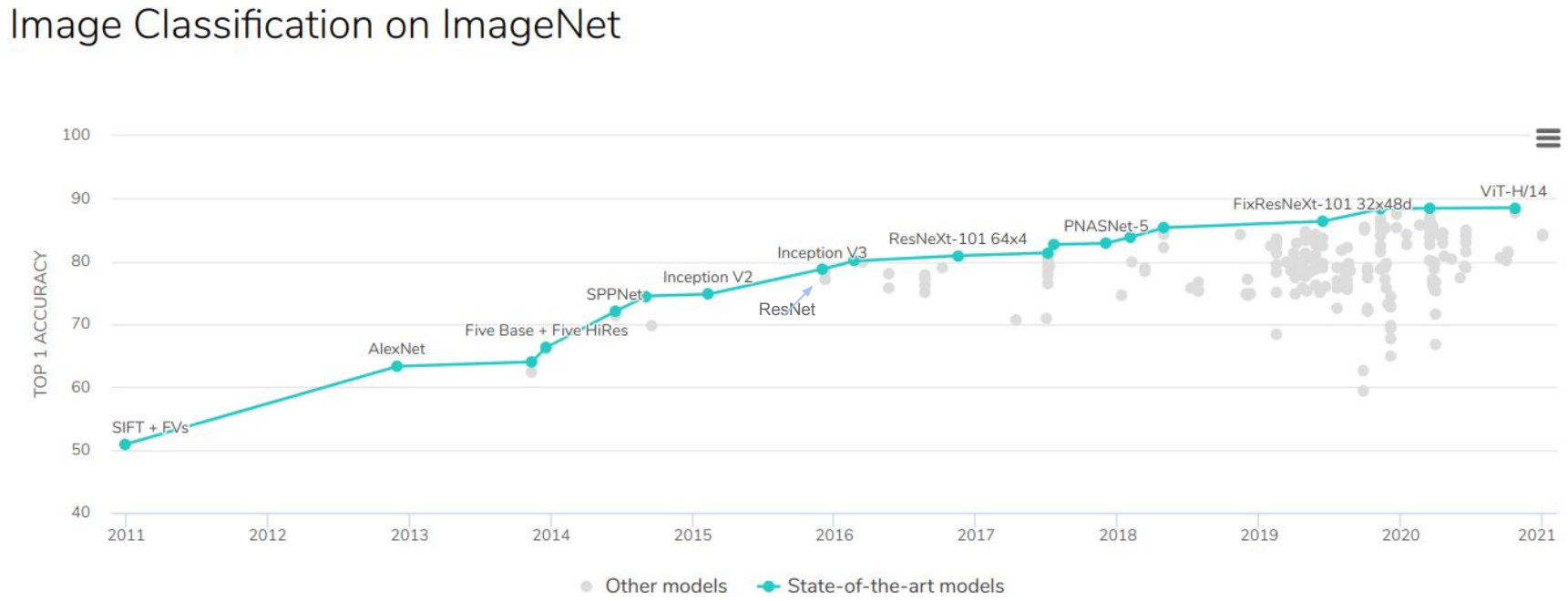
Although the ILSVRC annual competition is **officially over**, ImageNet remains to this day an important benchmark in literature.



Mila

ImageNet

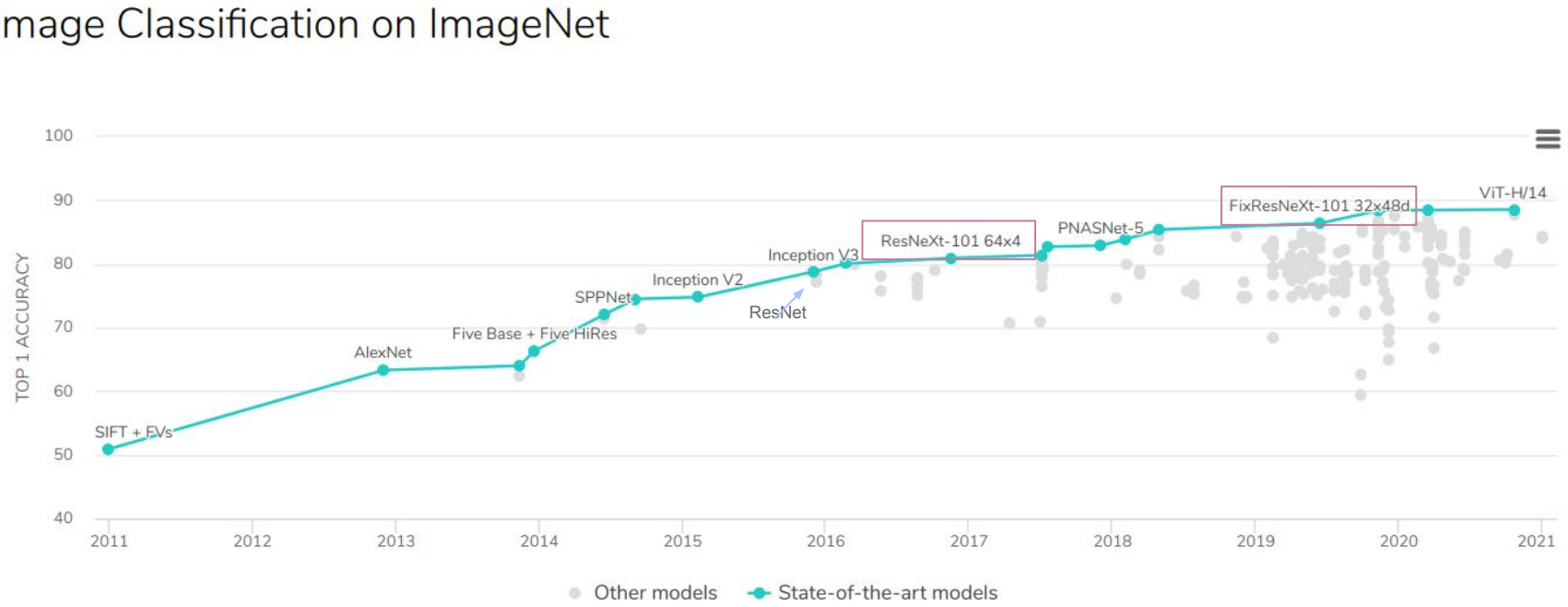
Today we will review some new models and paradigms to have achieved **state-of-the-art** on ImageNet.



Mila

ImageNet

Today we will review some new models and paradigms to have achieved **state-of-the-art** on ImageNet. We will begin with **ResNeXt.**

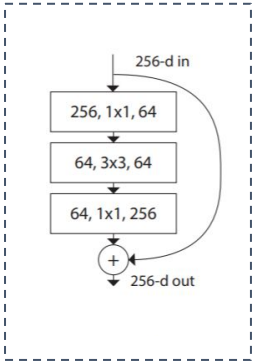


Mila

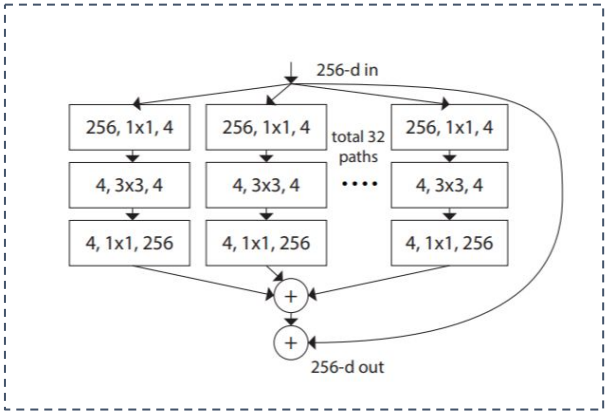
ResNeXt

ResNeXt is a direct enhancement of ResNet. Similar to inception, it has a module with parallel paths. It is designed to preserve the number of FLOPS in each block when compared to a ResNet.

Similar to ResNet, we have the **skip connection**. However, instead of having a single linear path, we now have parallel paths, denoted as *cardinality*.



ResNet block



ResNeXt block

https://arxiv.org/pdf/1512.03385.pdf
https://arxiv.org/pdf/1611.05431.pdf

ResNeXt

It can be thought of as having **multiple networks** in parallel on the same input (grouped convolutions).

It has been shown to **outperform** its ResNet counterpart while being able to preserve the amount of overall parameters in the network.

It was used as a backbone in the original Mask R-CNN implementation.

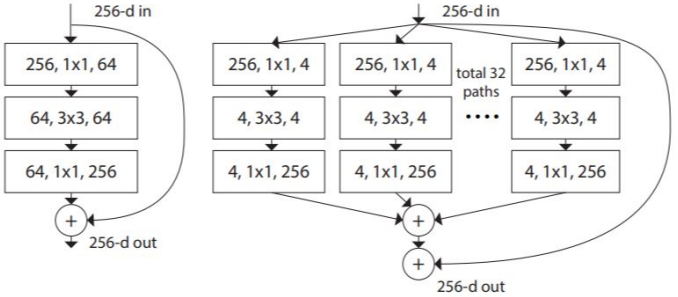
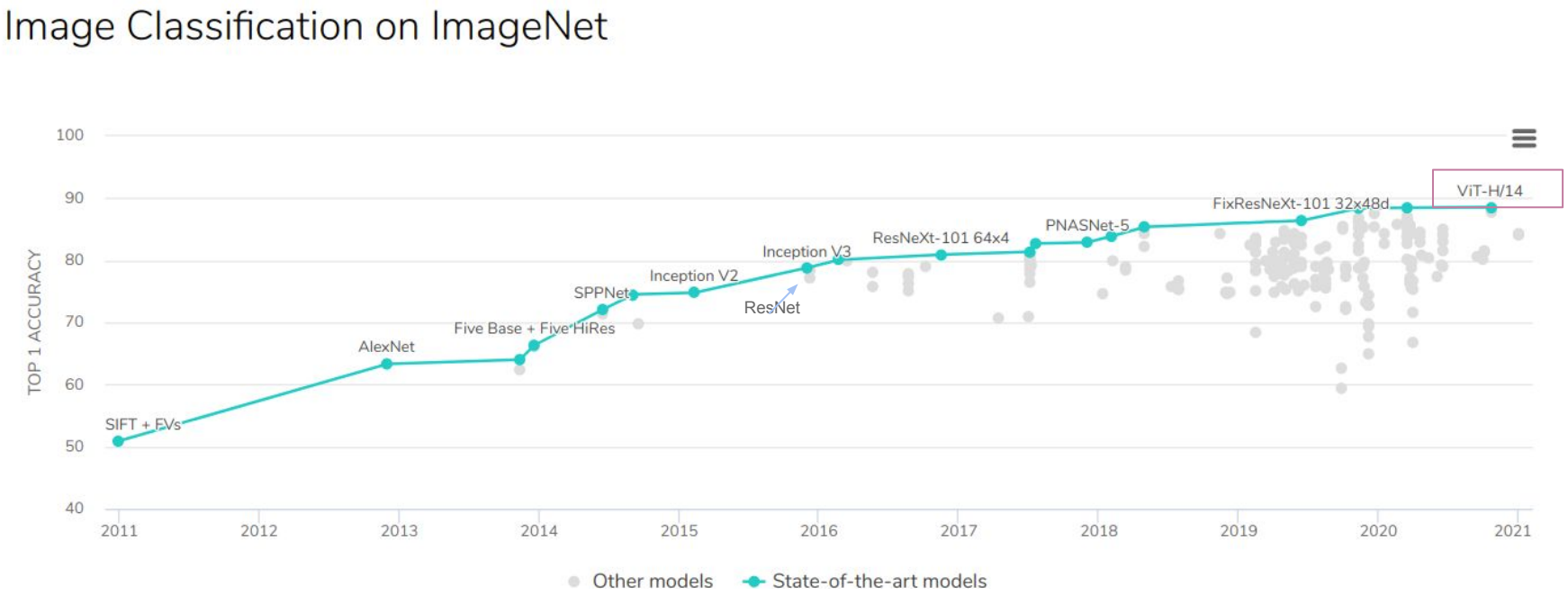


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

ImageNet top-1

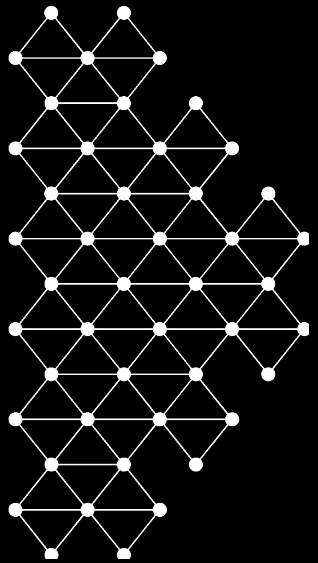
Currently, state-of-the-art belongs to ViT - **Vision Transformer**.



Mila

Quebec
Artificial
Intelligence
Institute

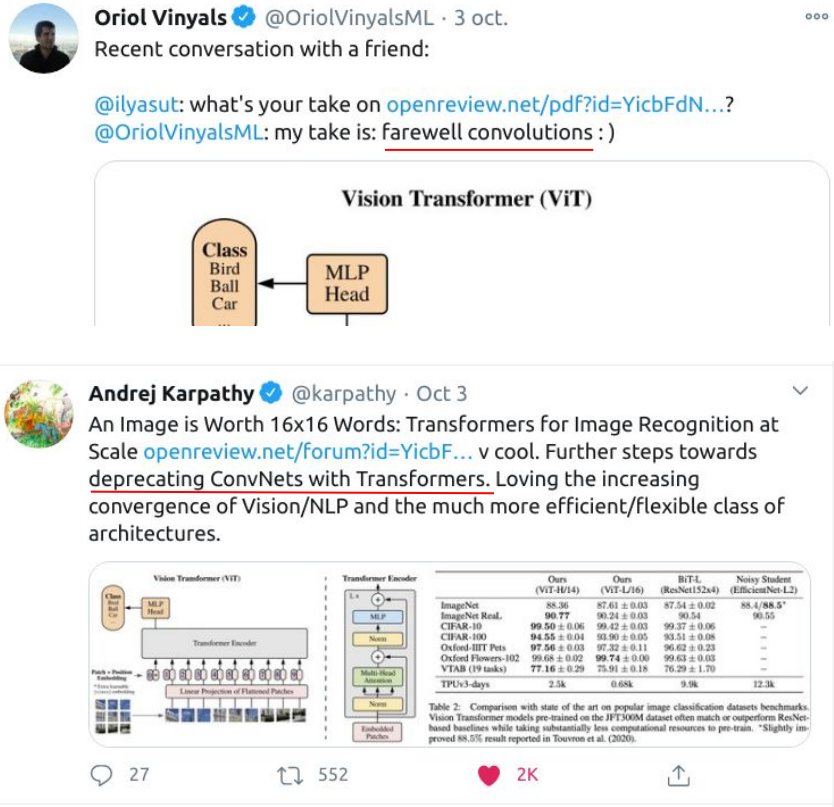
Mila



Transformers and computer vision

Farewell Convolutions?

A new paradigm is starting to emerge in computer vision. It involves adapting **transformers** to images instead of using CNNs.

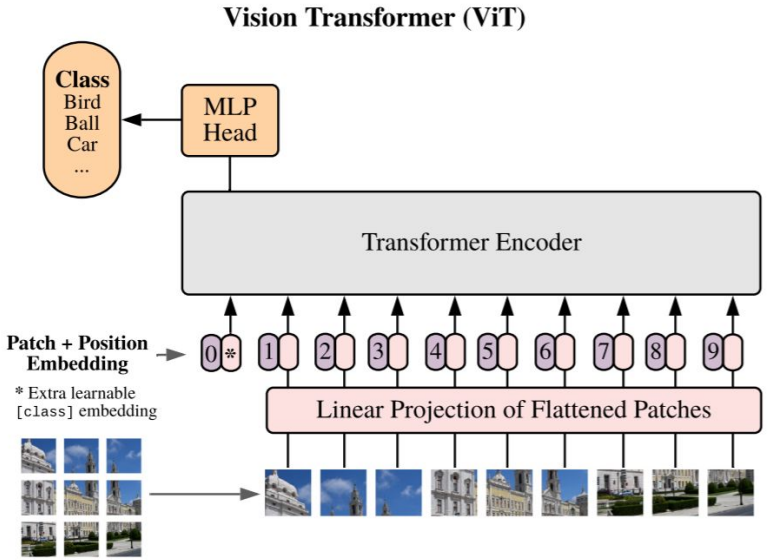


Transformers

You will see the details of how transformers work in **Module 4** with Mirko.

In short, **vectors** are passed as **input**, and **vectors** are retrieved at the **output.** They can be **adapted** to many tasks (e.g. classification, object detection).

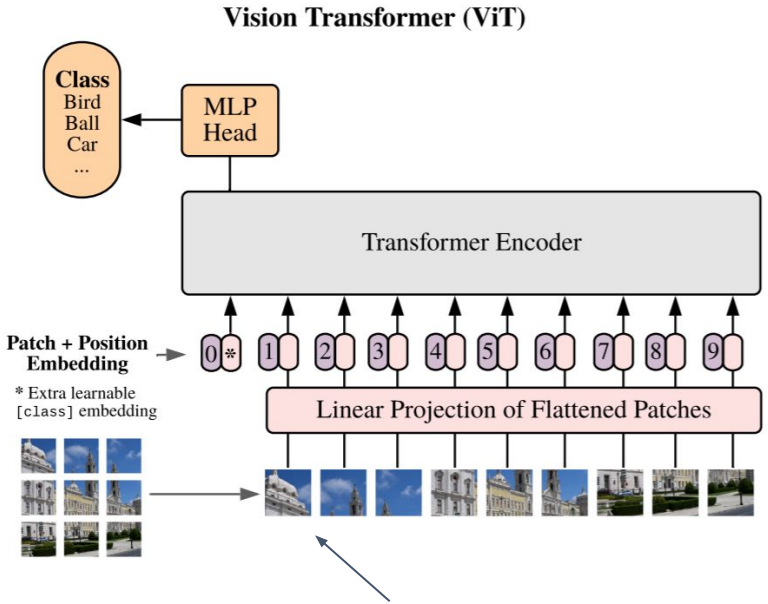
An **attention mechanism** is used which means all vectors can “attend” to all other vectors.



Vision Transformer

In Vision Transformer (ViT), it was shown that **transformers** could achieve **state-of-the-art** results on ImageNet.

This was achieved by **finetuning** their model on ImageNet after training on a much larger dataset.



Each image patch is converted to a vector using an MLP

Transformers for Computer Vision

Transformers have been **dominating** SOTA models in NLP (BERT, GPT, etc.)

We seem to be in an arms race for doing the same in **computer vision**.

Big tech companies have published **impressive results** using **transformers** on images.

- igpt: Generative Pretraining from Pixels (ICML 2020)

- Axial DeepLab (ECCV 2020)
- Vision Transformers (ICLR 2021)
- Multi-modal Transformer for Video Retrieval (ECCV 2020)

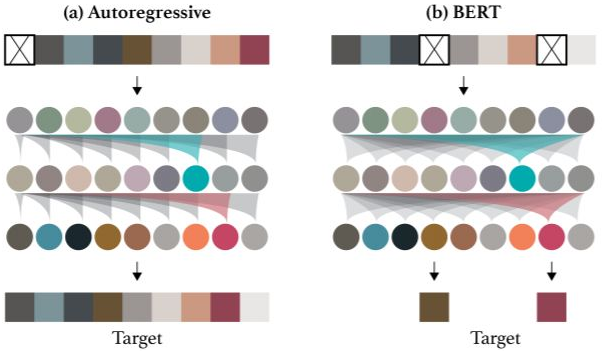
- DETR: End-to-End Object Detection with Transformers (ECCV 2020)

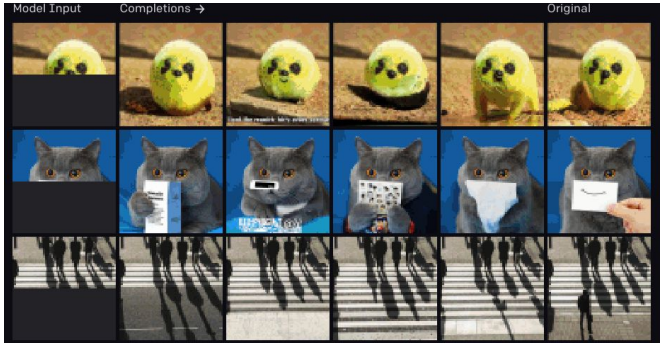
- VL-BERT: Pre-training of Generic Visual-Linguistic Representations

imageGPT

In iGPT, the same transformer architectures as those used in NLP are applied to raw images and used to complete **masked** images.

They show that they can get interesting results using **self-supervised** learning.





DEtection TRansformer (DETR)

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

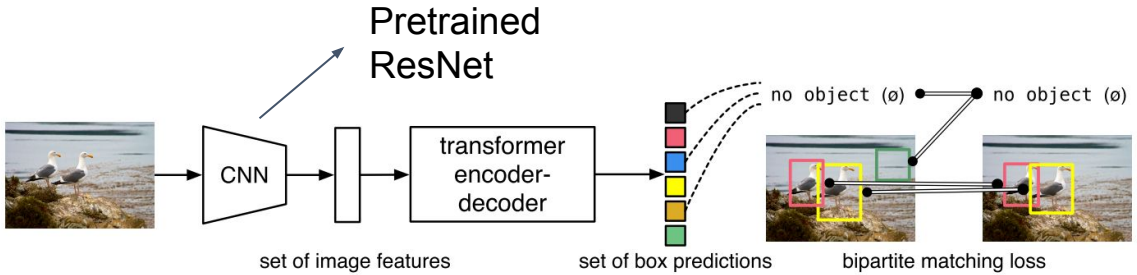


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

WALL-E + CLIP?

Advantages

- Transformers offer a possibility to do self-supervised training on very large unlabelled sets of data.
- Transformers allow for global receptive fields.
- Transformers can potentially unify language and computer vision

Challenges

- **Curse of dimensionality**: Attention mechanisms don’t scale well to large inputs (...for now)
- Very **large datasets** are needed to achieve interesting results
- Attention mechanisms are **invariant to position**, which means loss of structural information.

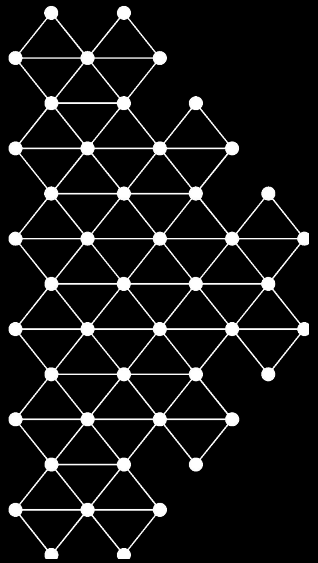
Photo by Honey Yanibel Minaya Cruz on Unsplash

Transformers:

- Implications, cost, benefit,
- USE AT YOUR OWN RISK

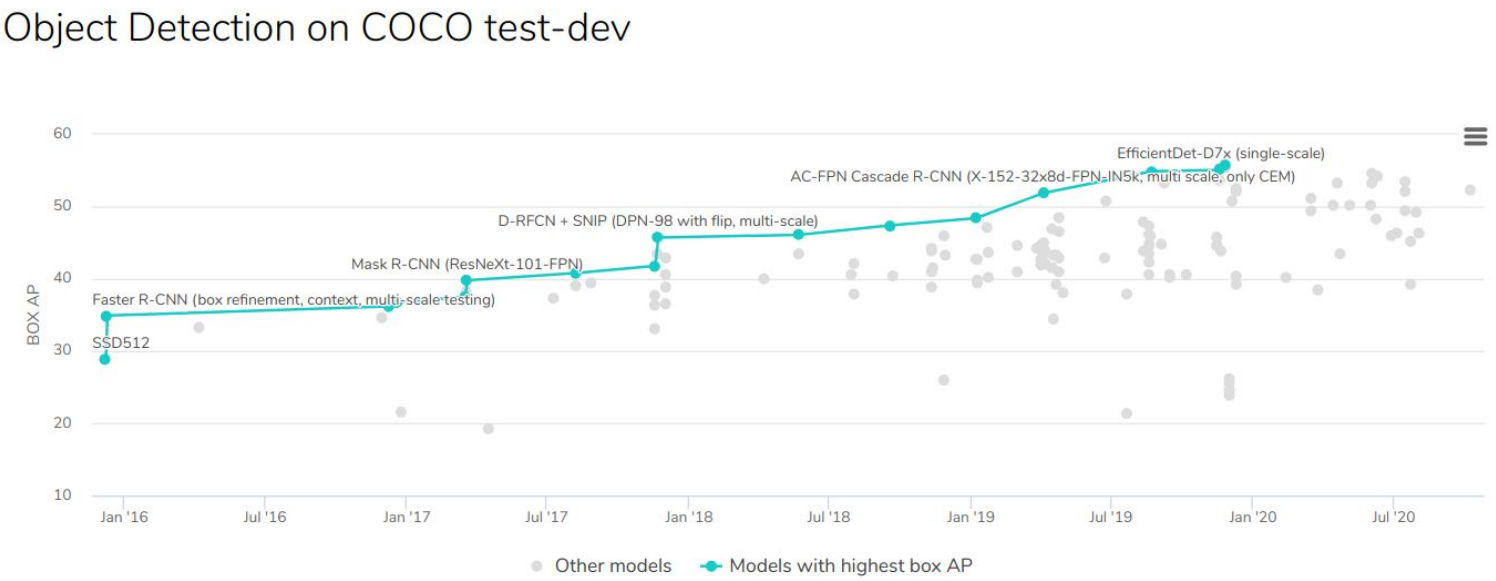
Quebec
Artificial
Intelligence
Institute

Mila

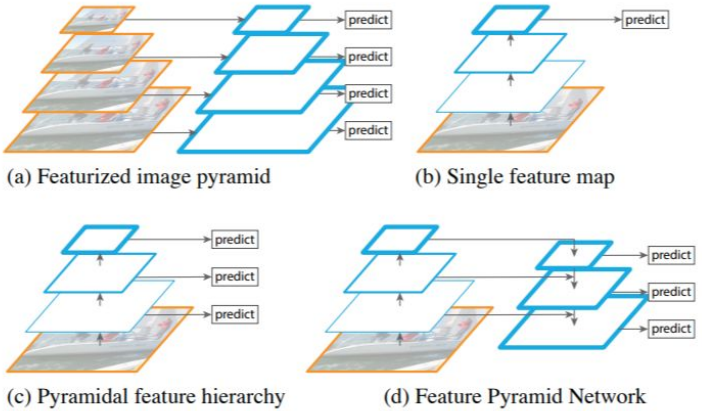


COCO

Object detection has made a lot of progress in recent years. We will highlight some tricks that are useful to understand.



Feature Pyramid Networks



(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 1. (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.