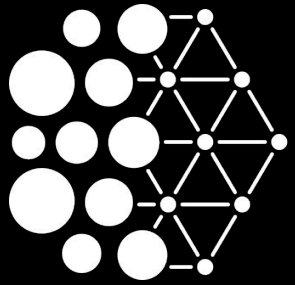
Quebec
Artificial
Intelligence
Institute



Mila

Video Processing, Network Interpretation and Updates in Computer Vision
Week 3

Jeremy Pinto
jeremy.pinto@mila.quebec

Jeremy Pinto

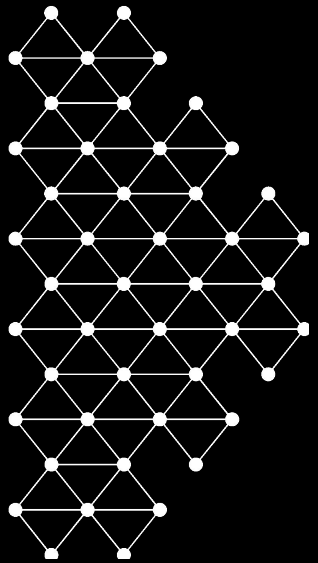
@jerpint

Content

- CNNs applied to Videos
- Model Interpretation
- Adversarial Attacks
- ImageNet - Updates
- Transformers in Computer Vision
- Piazza Questions

Photo by Jakob Owens on Unsplash

Quebec
Artificial
Intelligence
Institute



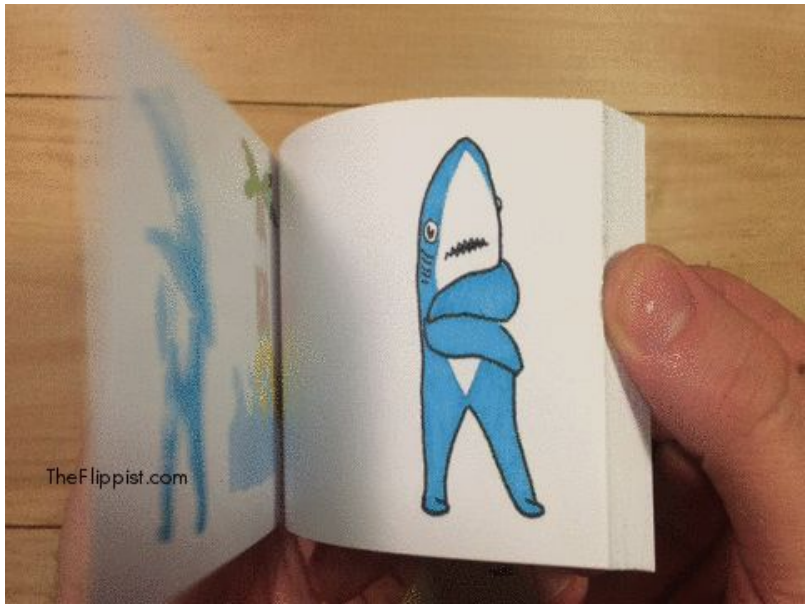
From images to videos

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**?

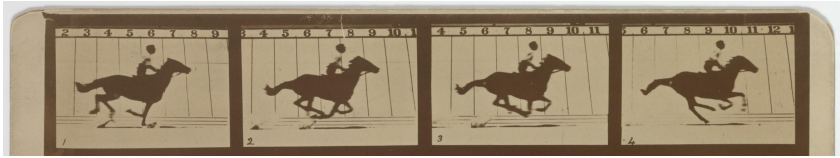
Since videos are just **streams** of images, we could just apply image models on **each frame**, independently!

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.



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

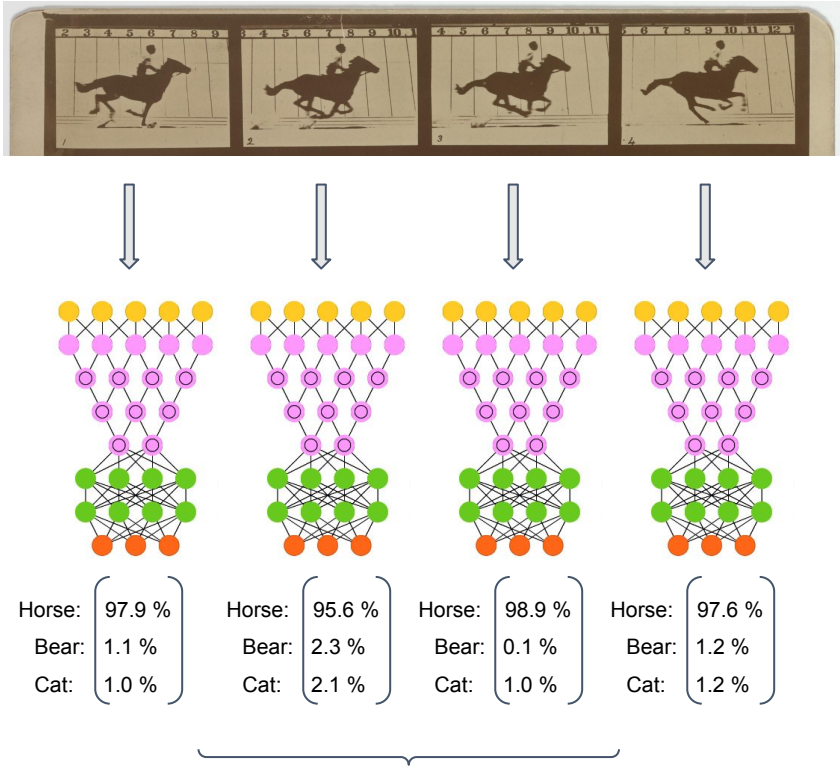
Image

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**.

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%

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

Here, the authors compare the speed vs. accuracy **tradeoff** they can get with their models.

They show that they can achieve real-time instance segmentation with comparable precision to state-of-the-art models.

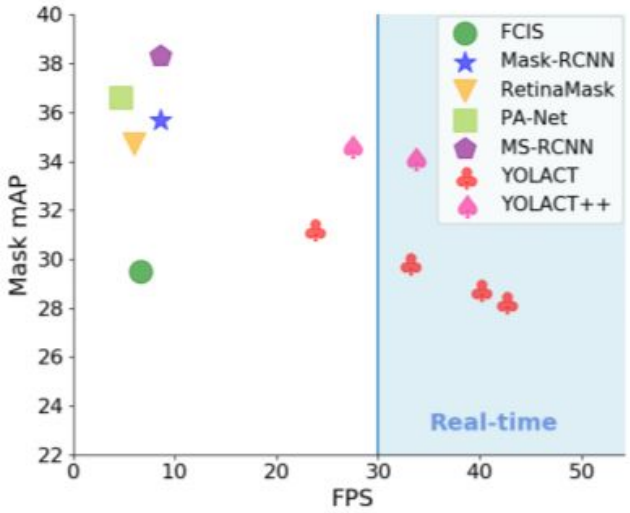


Fig. 1: Speed-performance trade-off for various instance segmentation methods on COCO. To our knowledge, ours is the first *real-time* (above 30 FPS) approach with over 30 mask mAP on COCO `test-dev`.

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

Videos

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
- Video annotation (scene understanding)
- Etc.

source

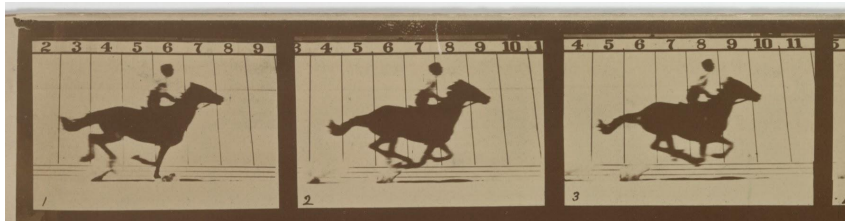
Videos

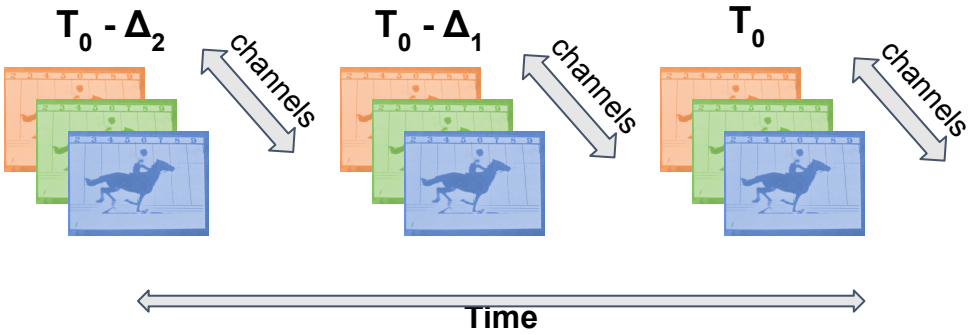
How do we **incorporate time** information to our neural networks?

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.





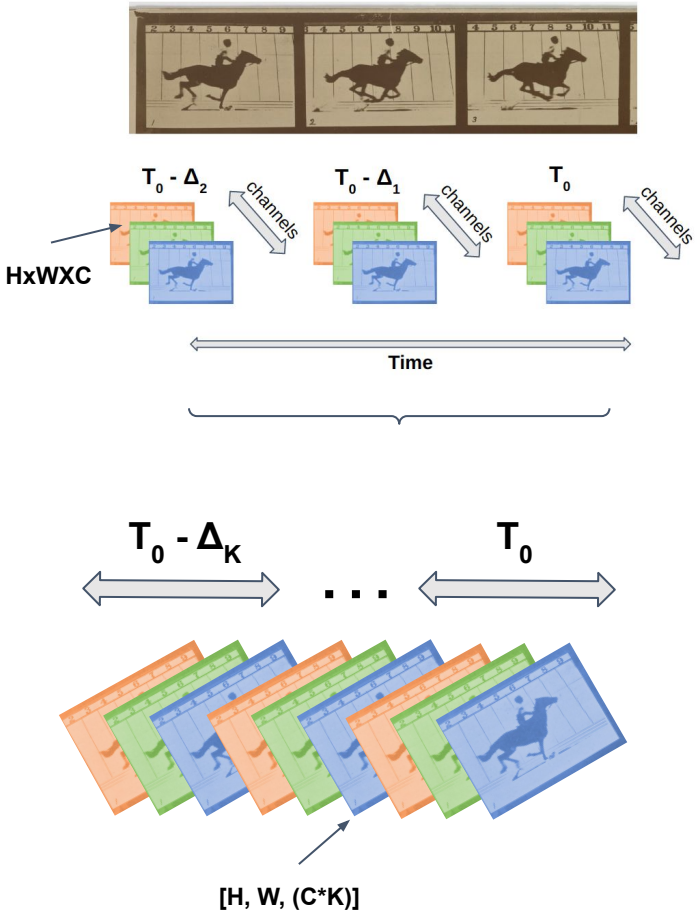
Image

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, 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.

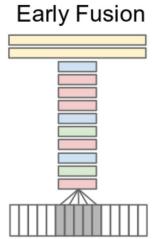
We would then have an input of shape **[H, W, (C*K)]**

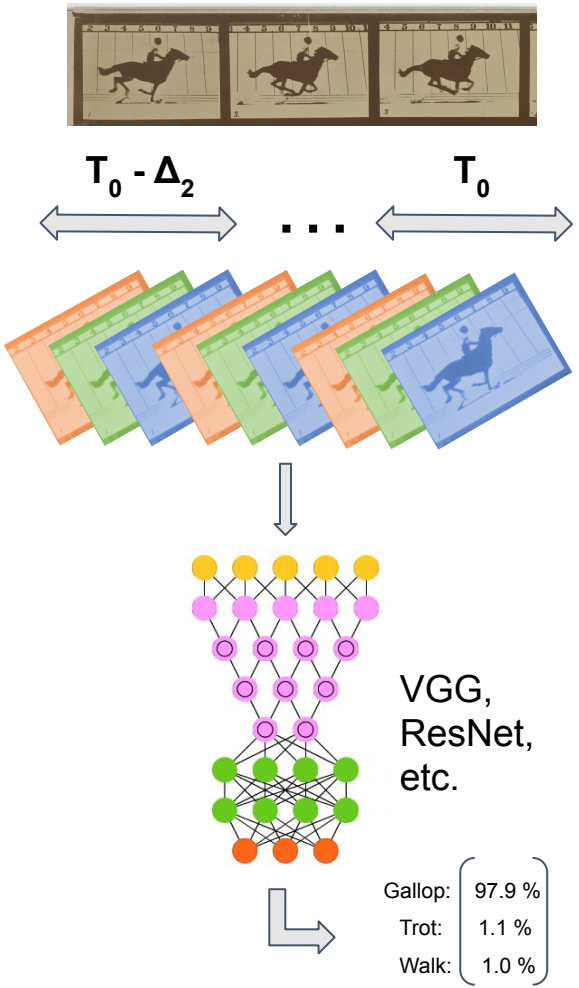


Videos

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**.



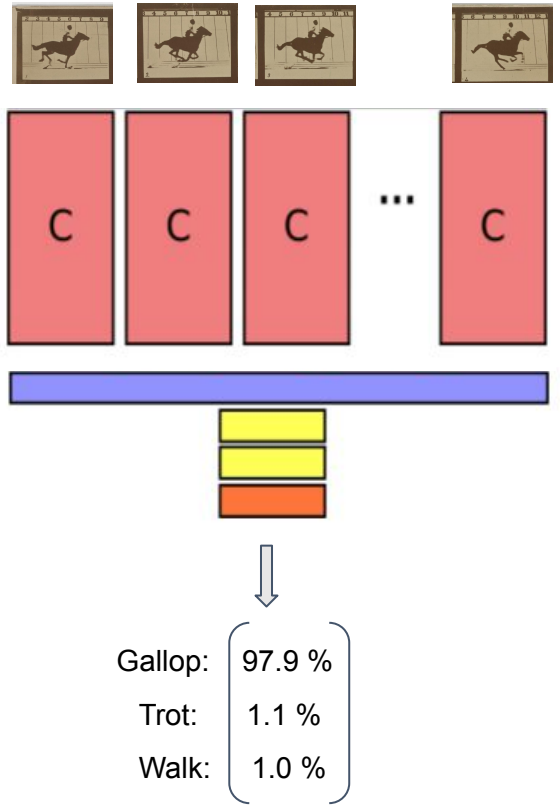


http://vision.stanford.edu/pdf/karpathy14.pdf

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).

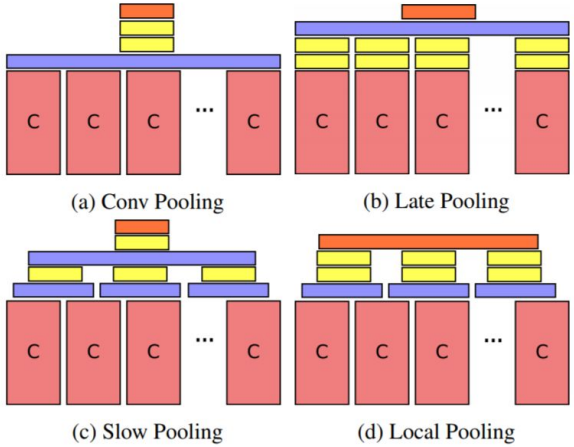


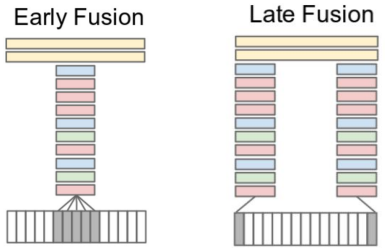
https://arxiv.org/pdf/1503.08909.pdf

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

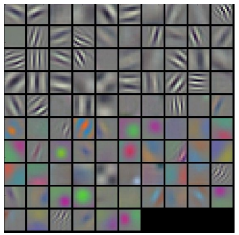
Conv3D

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

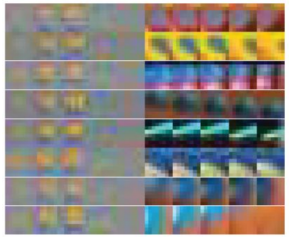
- Edges, contrasts
- ears, eyes noses



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).

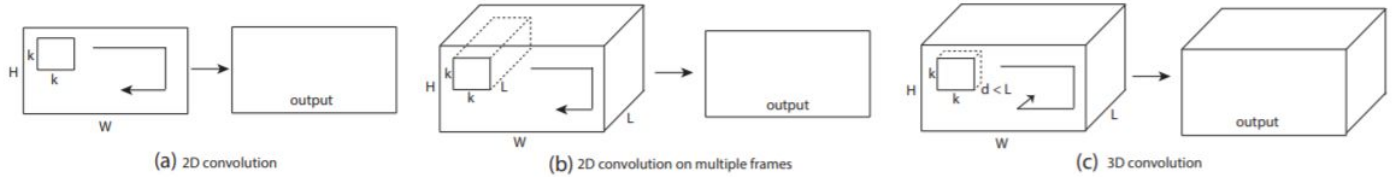


Figure 1. **2D and 3D convolution operations**. a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) Applying 3D convolution on a video volume results in another volume, preserving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

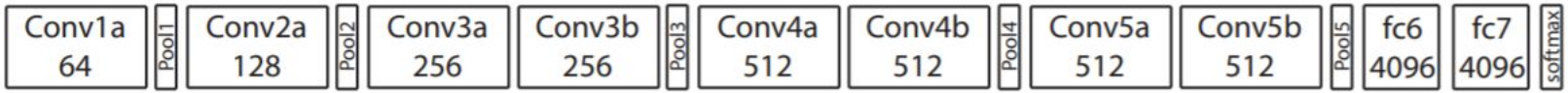


Figure 3. **C3D architecture**. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling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

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

3D convolutions extend very naturally from 2D convolutions.

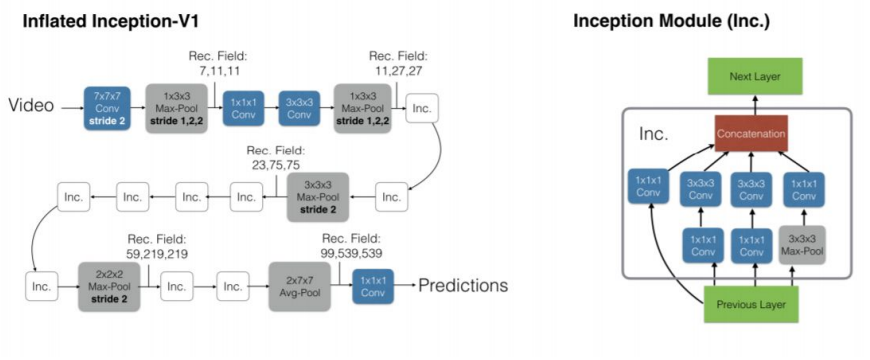


Figure 3. The Inflated Inception-V1 architecture (left) and its detailed inception submodule (right). The strides of convolution and pooling operators are 1 where not specified, and batch normalization layers, ReLu's and the softmax at the end are not shown. The theoretical sizes of receptive field sizes 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

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

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

```
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])
```

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)

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

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 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.

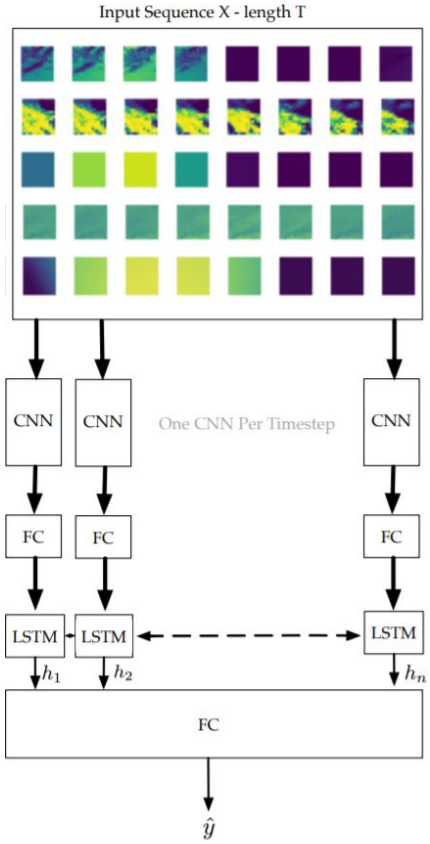
https://arxiv.org/pdf/1904.07846.pdf

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.

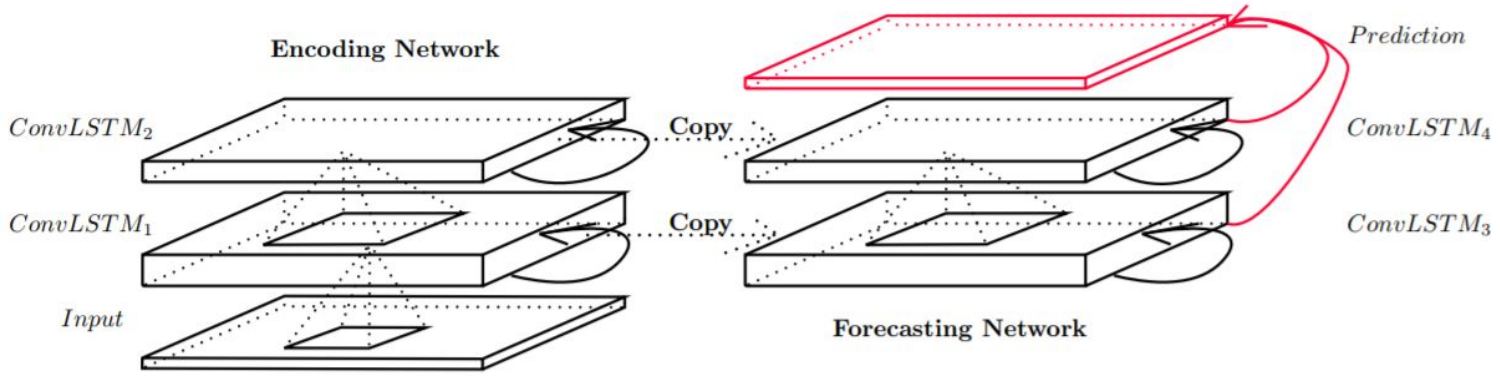
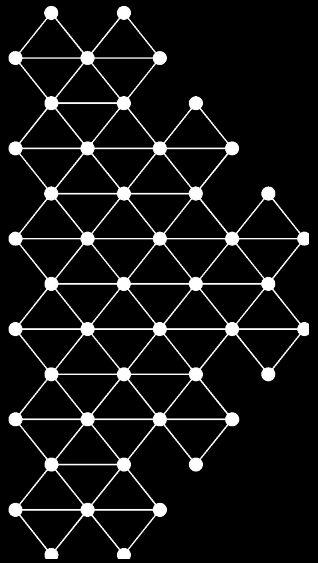


Figure 3: Encoding-forecasting ConvLSTM network for precipitation nowcasting

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

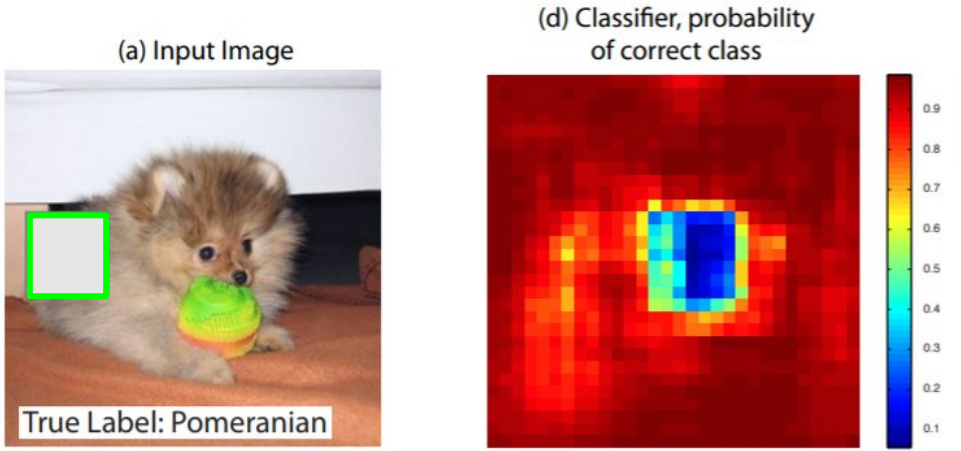
Due to the inherent structure of CNNs, it can be very difficult to **interpret how** they make decisions.

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

Photo by Zach Vessels on Unsplash

Model interpretation

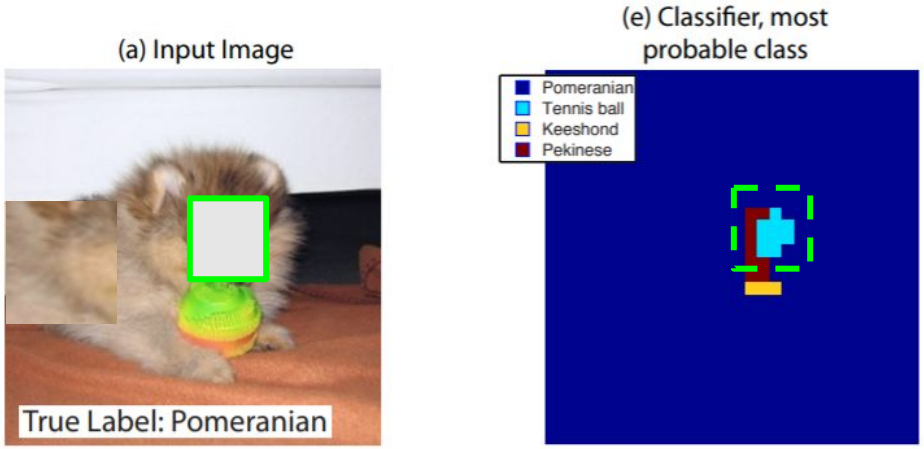
One strategy involves **blocking** out inputs of an image and looking at how the predictions vary over the network. These are known as **perturbation methods**.



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

Model interpretation

When we block out the face of the dog, the **tennis ball** get's predicted with high probability.

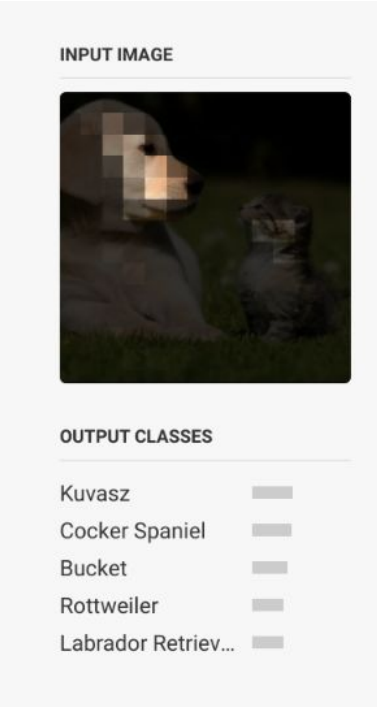


https://arxiv.org/pdf/1311.2901.pdf

Model Interpretation

We can also use the activation from a given layer to correlate the parts of an image with its predicted classes.

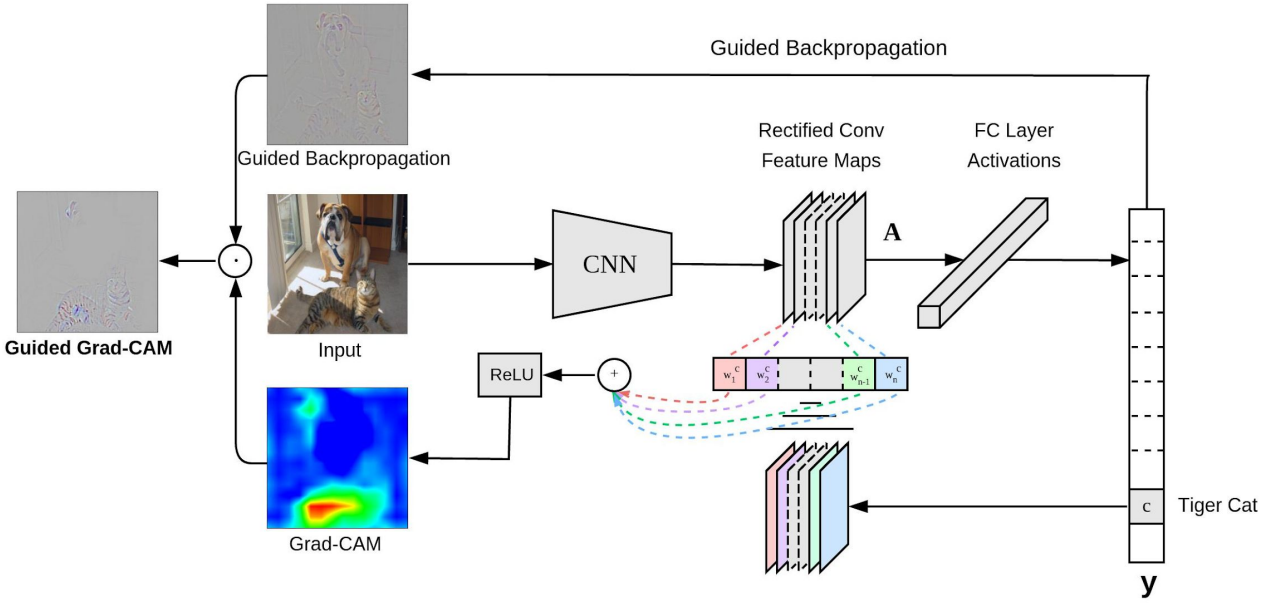
We see here that when focusing on a dog's face, the network hesitates between various dog classes.



https://distill.pub/2018/building-blocks/

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

These methods can be very useful to **identify bias** attributed to certain classes in a network.

Here, we see that in a biased model, a network focuses on the face of a female doctor when incorrectly attributing the label “nurse”.

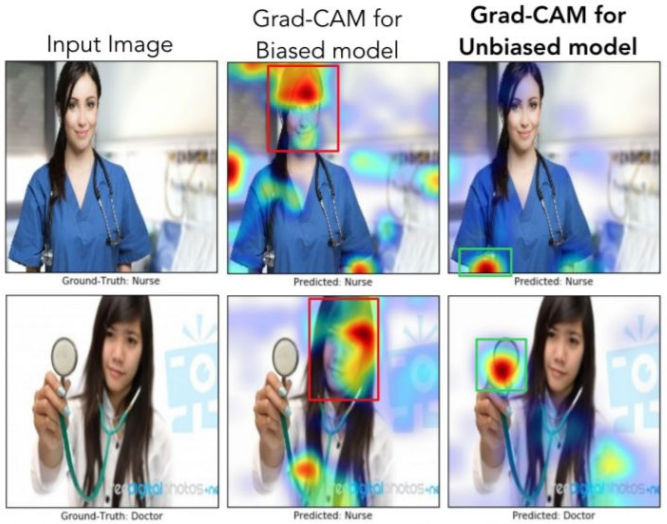


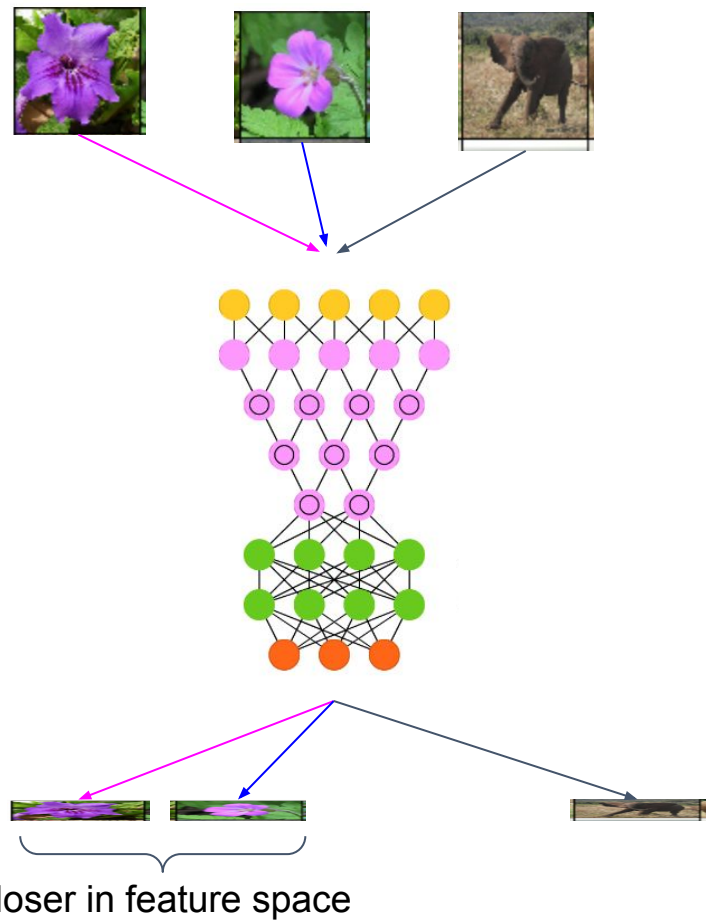
Fig. 8: In the first row, we can see that even though both models made the right decision, the biased model (model1) was looking at the face of the person to decide if the person was a nurse, whereas the unbiased model was looking at the short sleeves to make the decision. For the example image in the 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**.

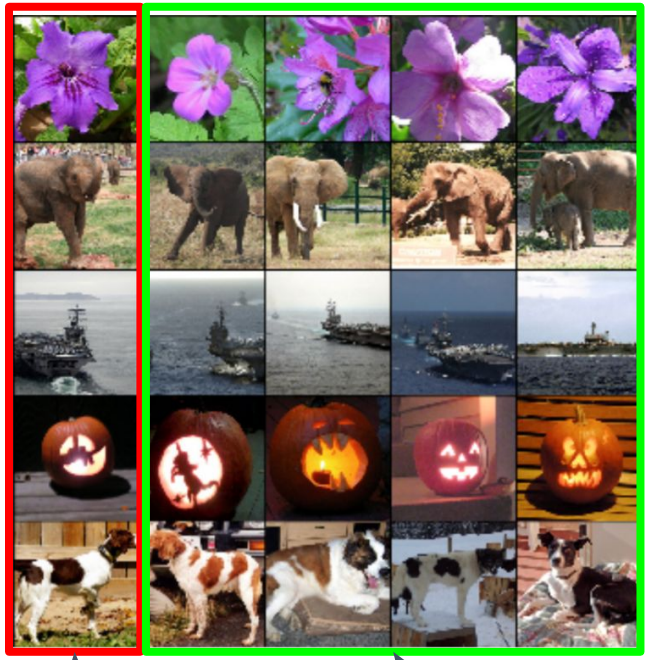
For example, in AlexNet, the authors reported the images that were closest in the **feature space** (last hidden layer) to the query sample.



Model interpretation

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

Here, a dot product is used to find the most **similar samples** to the query in the feature space.



Queries

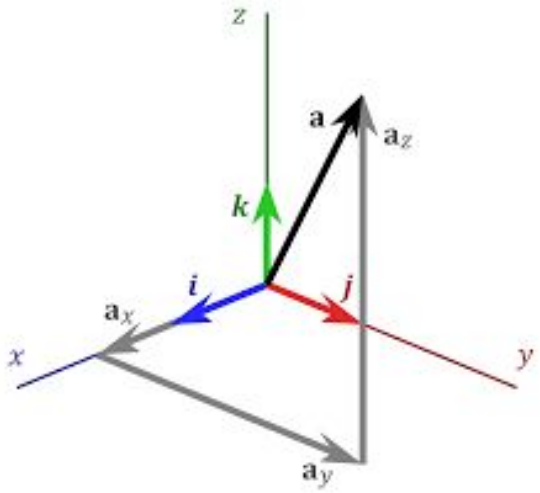
Closest matches in feature space

Dimensionality Reduction

However, these **feature spaces** can be very large. In AlexNet, the last dense layer is a vector with 2048 dimensions.

These numbers are **abstract**, non-linear relationships that the network learns. They are very difficult to interpret.

Dimensionality reduction can help to untangle these high dimensional mappings.



Dimensionality Reduction

Dimensionality reduction can really be a useful tool to see if **meaningful clusters** emerge.

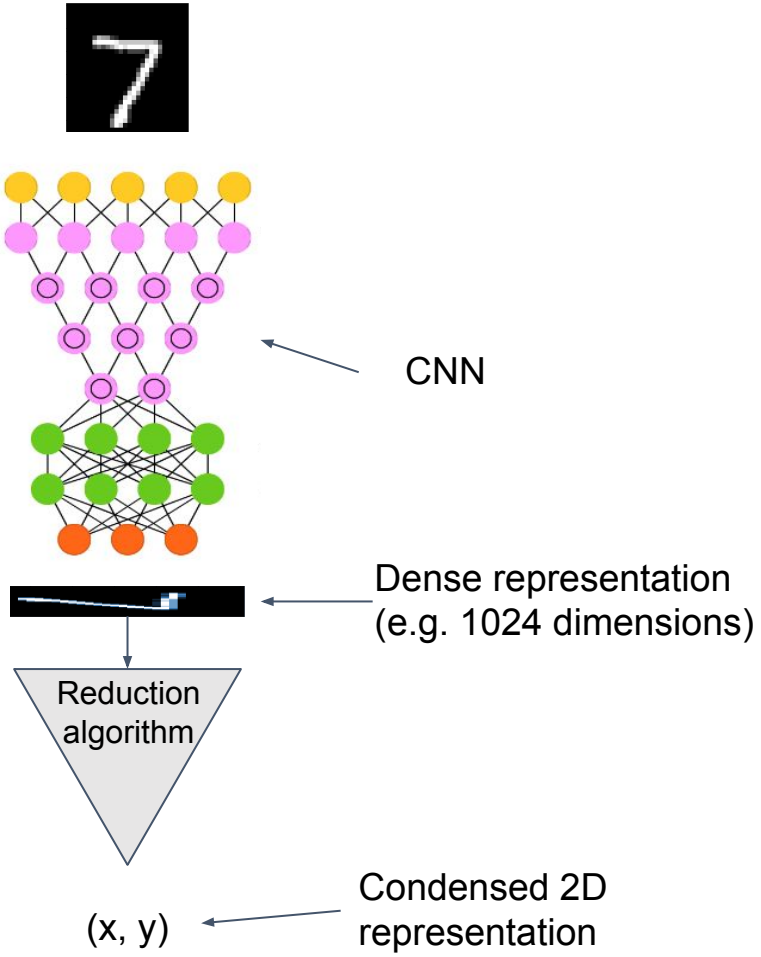
So **how** does it work?

Image

Dimensionality Reduction

Dimensionality reduction allow us to preserve **“useful” relationships** while **significantly reducing** the number of dimensions of the vectors.

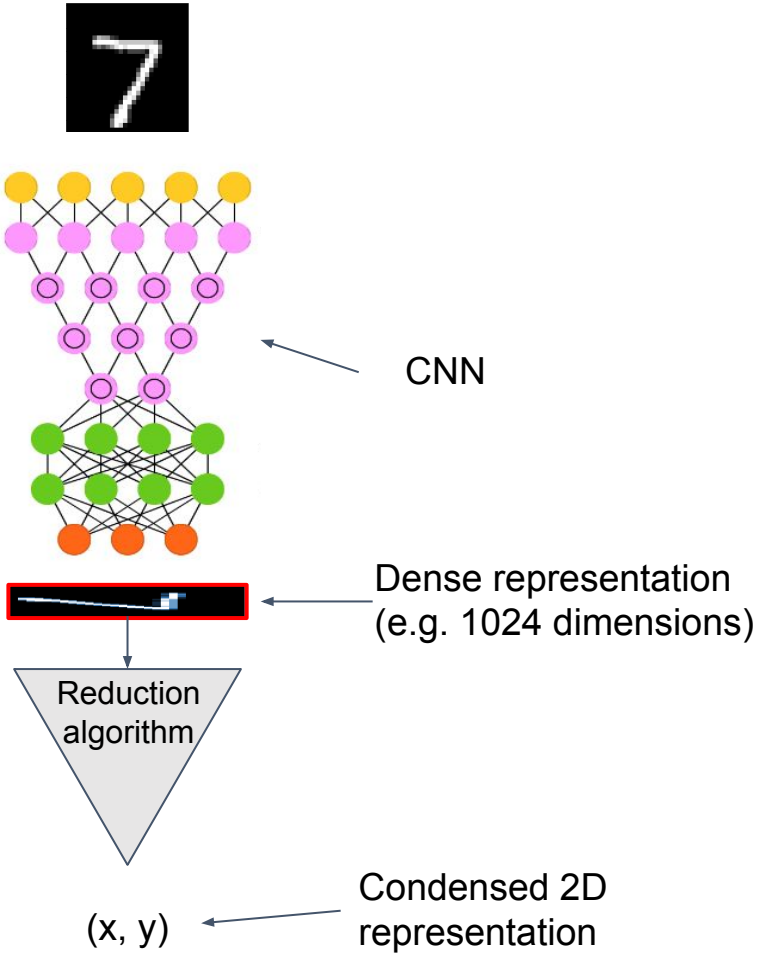
Dimensionality reduction can be used on the **dense representation** of the input - typically the last layers of a CNN.



Dimensionality Reduction

Some commonly used algorithms are:

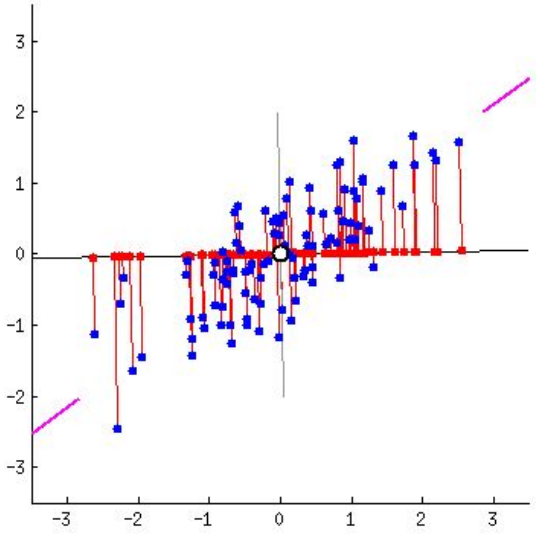
- Statistical methods
  - PCA
  - T-SNE
- Learning-based methods
  - Autoencoders



PCA

Principal component analysis (PCA) attempts to **reduce** the number of features representing while **maximizing the variance** of the dataset.

In this example, the “purple” direction represents the **direction** along which **maximal information** is preserved.



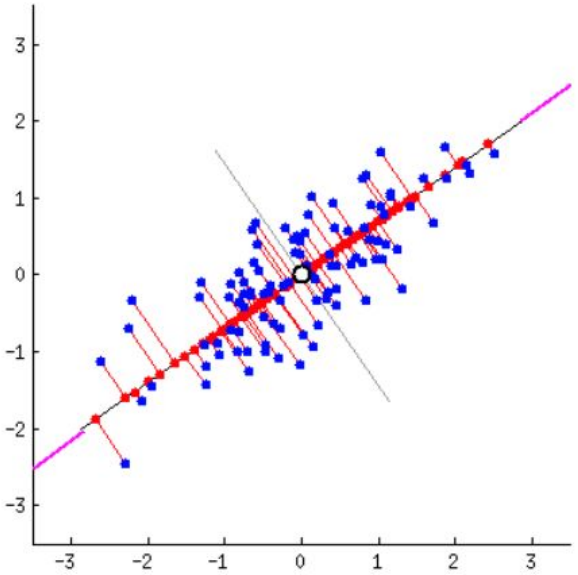
source

PCA

Principal component analysis (PCA) attempts to **reduce** the number of features representing while **maximizing the variance** of the dataset.

In this example, the “purple” direction represents the **direction** along which **maximal information** is preserved.

We can think of reducing our current scatter (x,y) to the form y = ax + b, going from 2 variables, to 1 variable.



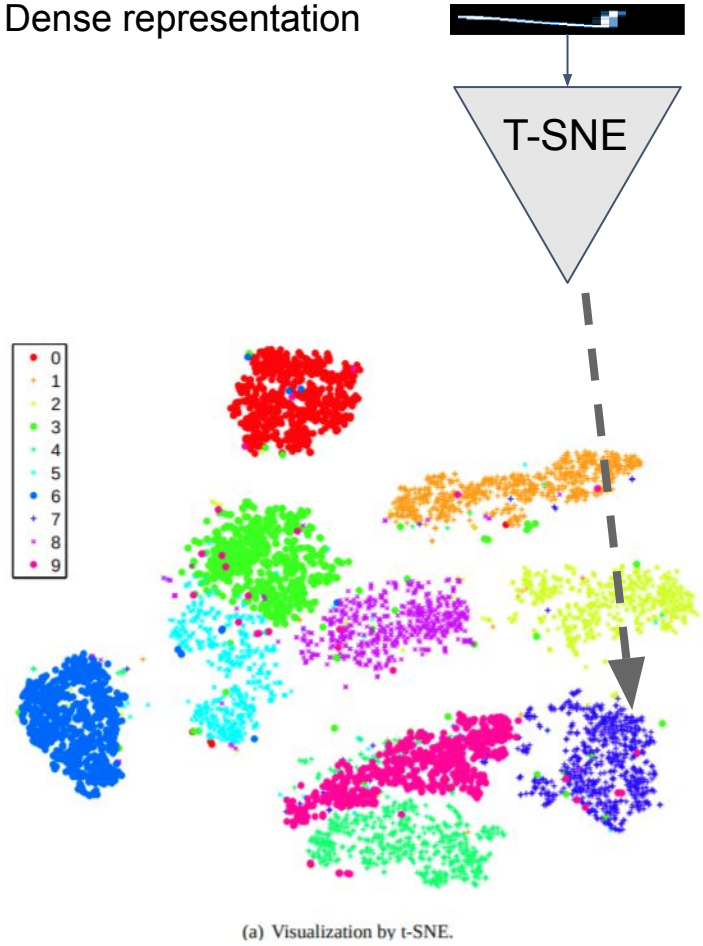
source

T-SNE

T-SNE looks for mappings that **preserve** the relative distances between samples across feature spaces.

Here, we **reduce** the feature vectors in MNIST (e.g. 1568 in LeNet) to 2 dimensions using T-SNE.

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



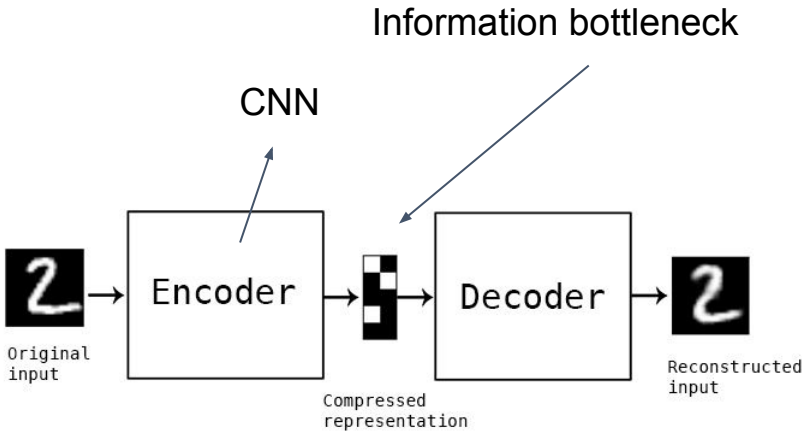
https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

Autoencoders

Autoencoders learn directly from the data to **reduce** the feature representations.

Given an input, we learn to **reconstruct** the exact same output but pass the input through an **information bottleneck**.

Once the network is trained, the **encoder** representation can be used in downstream tasks.

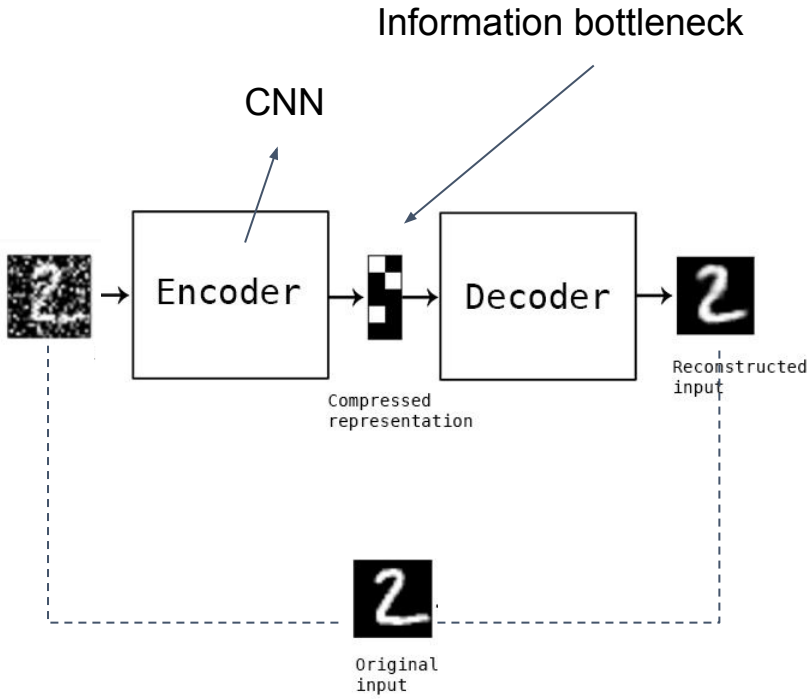


source

Denoising Autoencoder

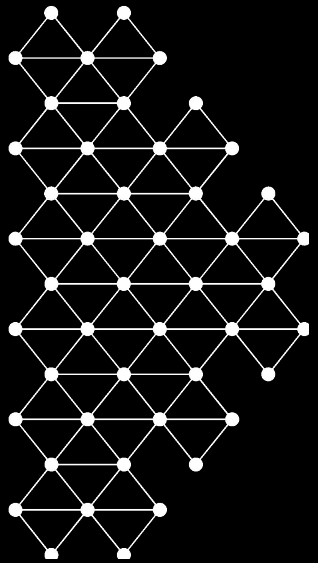
In a denoising autoencoder,
we **add noise** to our input and learn to **reconstruct** the clean original input.

This gives us a more robust compressed representation - the network needs to learn to focus on **important details** while filtering out the noise.



source

Quebec
Artificial
Intelligence
Institute

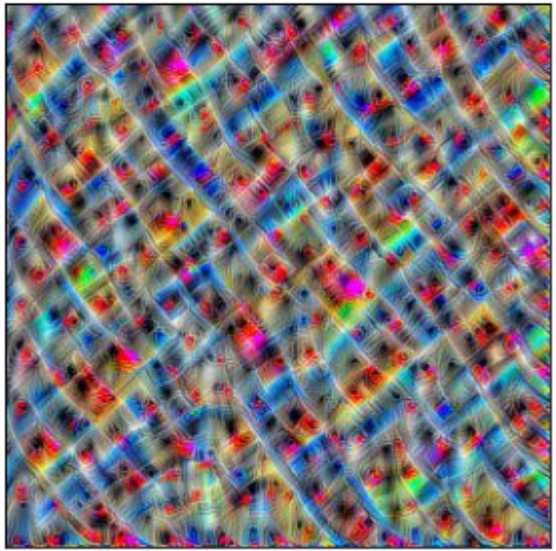


Feature Visualization

Feature Visualization

We've seen through dimensionality reduction how we can observe the clustering of our data.

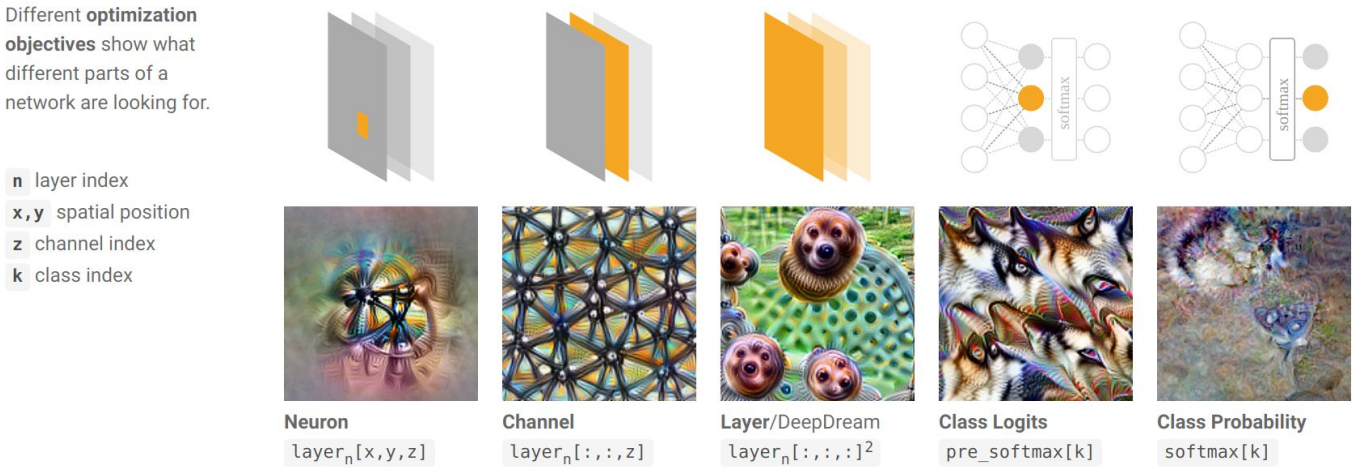
However, there are also techniques to **peek** at what our neural network has learned.



source

Feature Visualization

We can visualize what activates **components** in various areas of a network using **optimization**. It is helpful to “see” what the network might be thinking.

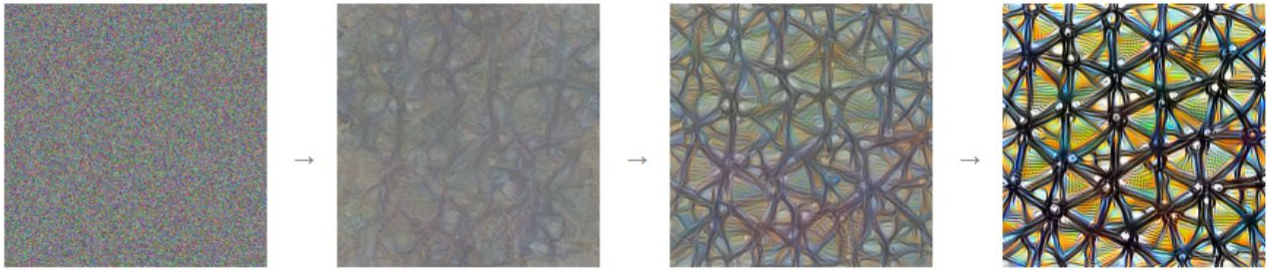


https://distill.pub/2017/feature-visualization/

Feature Visualization

Starting from a trained network, we use **noise** as an input and **optimize** the input to maximally activate the neuron we are seeking to interpret.

Starting from random noise, we optimize an image to activate a particular neuron (layer mixed4a, unit 11).



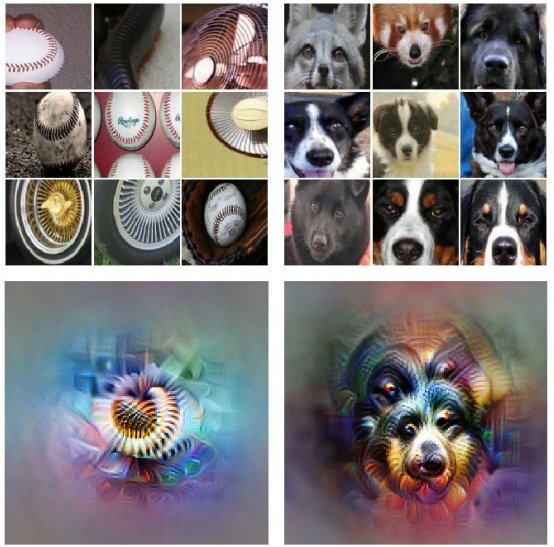
https://distill.pub/2017/feature-visualization/

Feature Visualization

We can use this **optimization** as a tool to probe what causes certain neurons to fire in our network.

Dataset Examples show us what neurons respond to in practice

Optimization isolates the causes of behavior from mere correlations. A neuron may not be detecting what you initially thought.



Baseball—or stripes?
mixed4a, Unit 6

Animal faces—or snouts?
mixed4a, Unit 240

https://distill.pub/2017/feature-visualization/

Feature Visualization

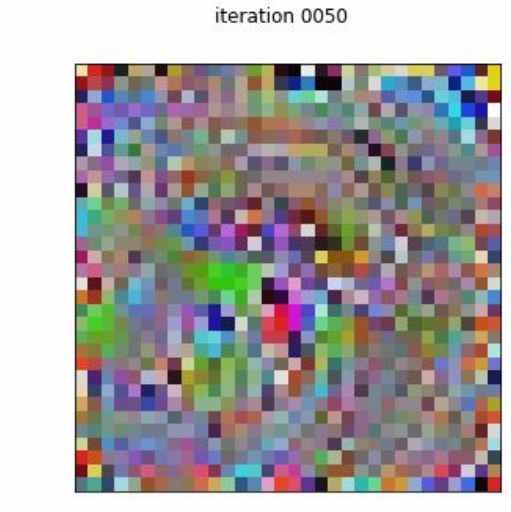
```python
class CustomVgg19(torch.nn.Module):
            def __init__(self, stop_layer, device, pretrained=True):
                '''Utility class to define our model.

                We can "slice" the network at any layer, using the stop_layer
                argument to define that layer. The model will return the
                feature map at that sliced layer. We use by default a pretrained
                network.
                '''
                super(CustomVgg19, self).__init__()
                vgg19 = models.vgg19(pretrained=pretrained)
                vgg19 = vgg19.to(device)
                self.features = torch.nn.Sequential(
                    *list(vgg19.features.children())[:stop_layer]
                )
            def forward(self, x):
                return self.features(x)


lr = 0.05
wd = 1e-6
n_iters = 2000
batch_size = 1
stop_layer = 20
target = init_target(width=128, height=128, device=device) # uniform noise
model = CustomVgg19(stop_layer=stop_layer, pretrained=True, device=device)
optimizer = torch.optim.Adam([target], lr=lr, weight_decay=wd)
for n_iter in tqdm(range(1, n_iters + 1)):
    optimizer.zero_grad()
    out = model.forward(target)
    loss = -torch.mean(out[:, 42, :, :])
    loss.backward()
    optimizer.step()

display_targets(target, caption=f'iteration {n_iter:04}', save=False, show=True)
```

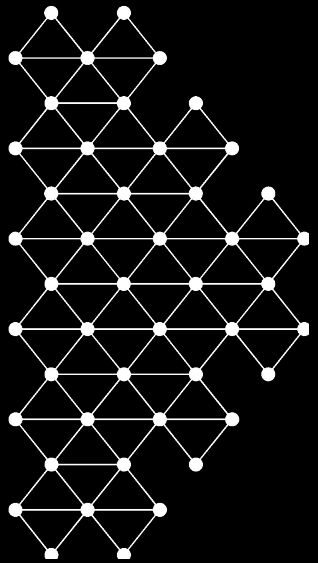
This can be implemented in just a few lines of pytorch code - however there are a lot of “tricks” that need to be implemented to make it look **pretty**.



source

Mila

Quebec
Artificial
Intelligence
Institute

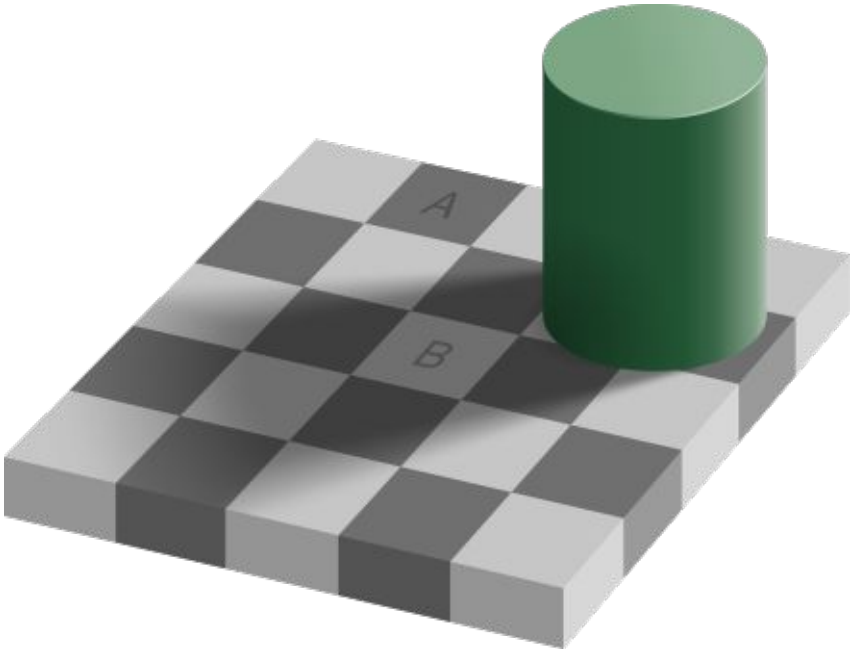


Adversarial Attacks

Adversarial Attacks

Our brain can easily be fooled using trickery. For example, look at this grid:

Are the colours of square A and B identical?

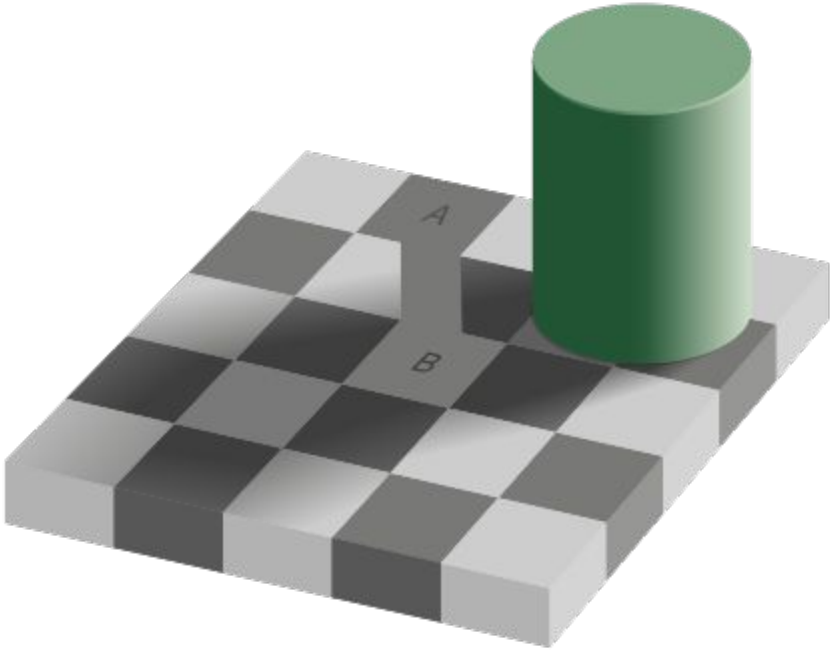


source

Adversarial Attacks

It turns out that they are. No **trickery** was used.

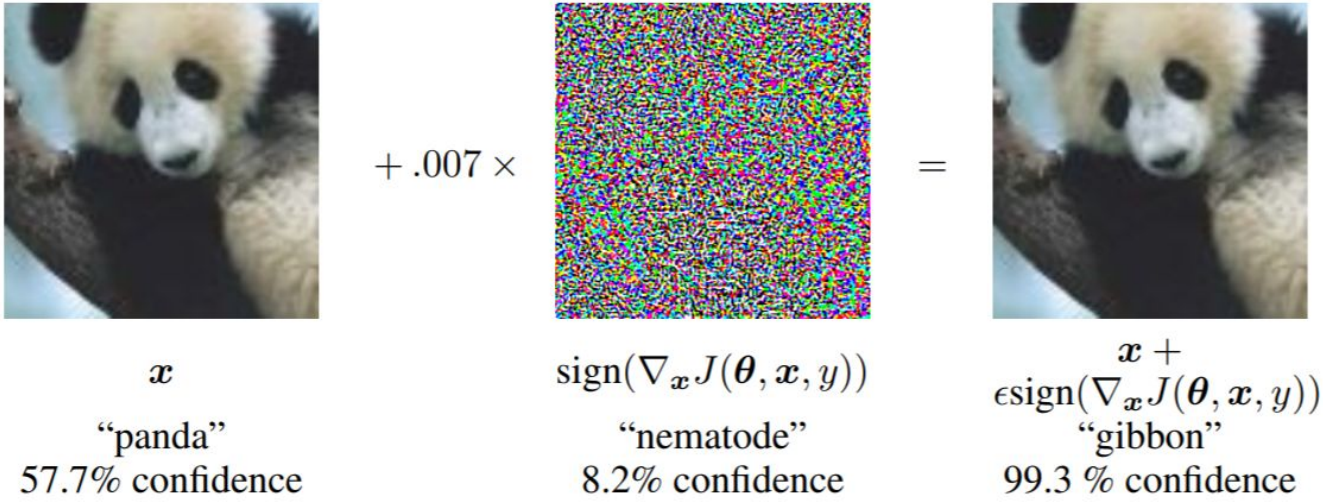
By simply playing with the **shadows**, our brains get fooled due to our preconceived expectations of what a regular grid should look like.



source

Adversarial Attacks

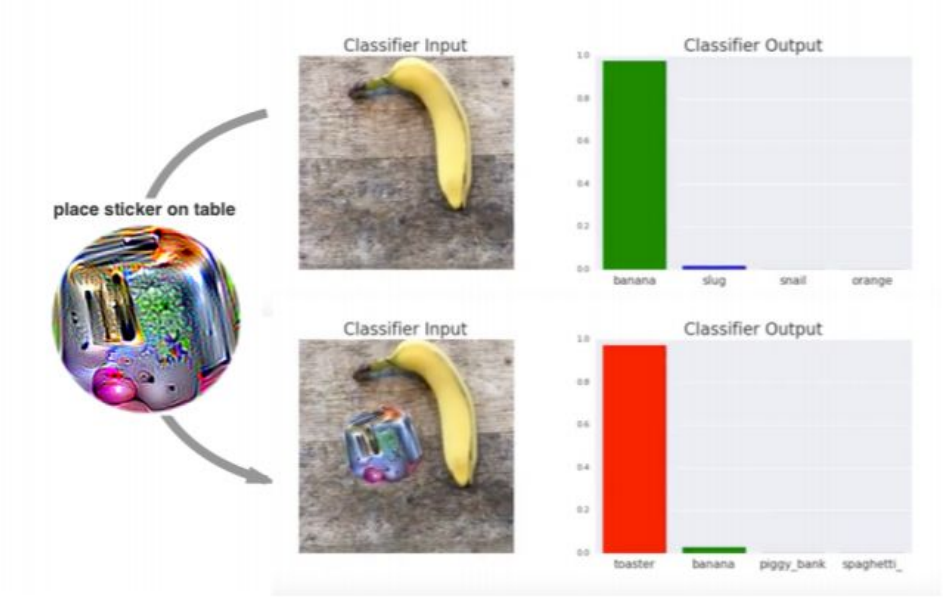
It is equally possible to **fool** networks with adversarial attacks. Here, we use a noisy perturbation based on the gradient of the network. We can easily overturn the classification of a class in a network:



https://arxiv.org/pdf/1412.6572.pdf

Adversarial Attacks

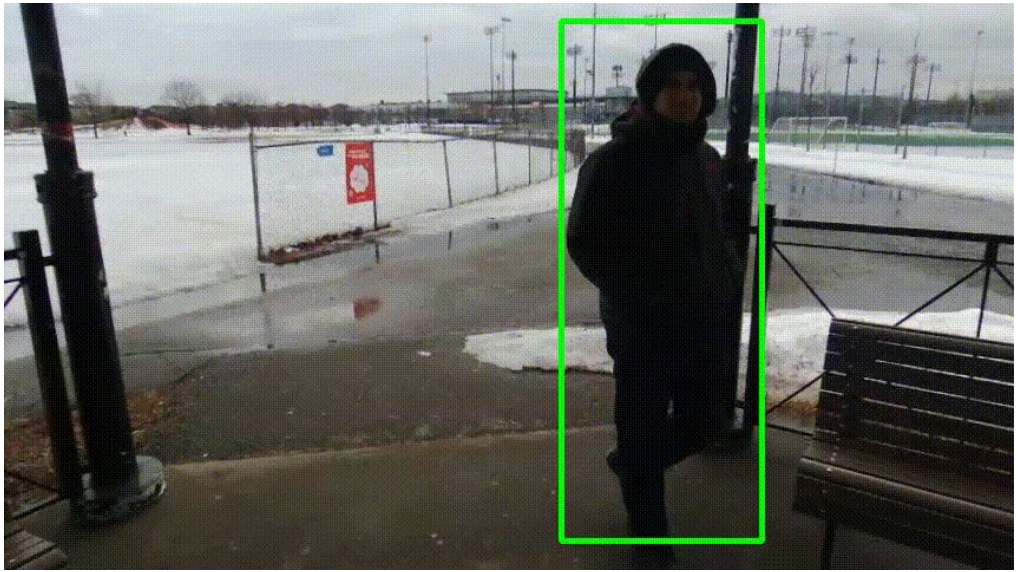
In this example, a carefully crafted **sticker** can be used to fool a detector to always output a toaster.



https://arxiv.org/pdf/1712.09665.pdf

Adversarial Attacks

Here, a physical object is placed in the real world and “intercepts” the tracking of a person.



source

Adversarial Attacks

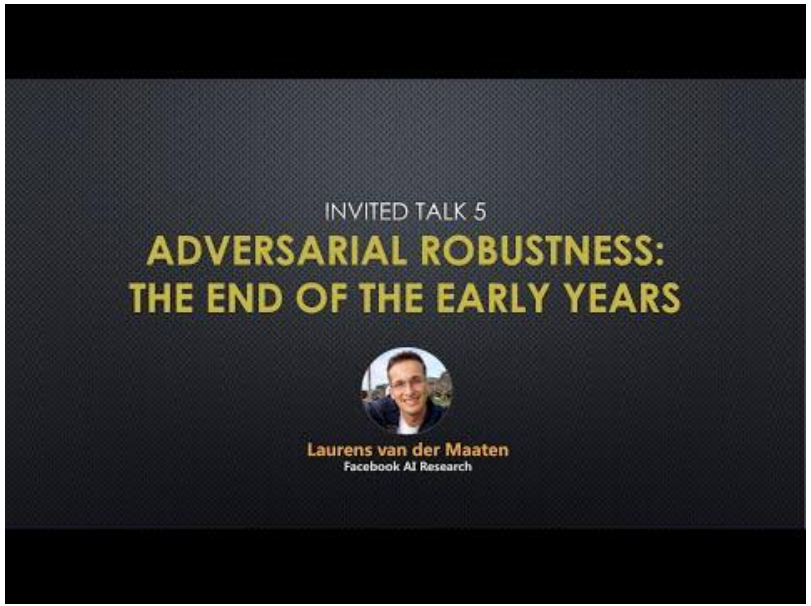
These attacks can have **catastrophic** consequences in real-world environments - e.g. self-driving cars.

Figure 1: We cause the network to generate a *minion* as segmentation for the adversarially perturbed version of the original image. Note that the original and the perturbed image are indistinguishable.

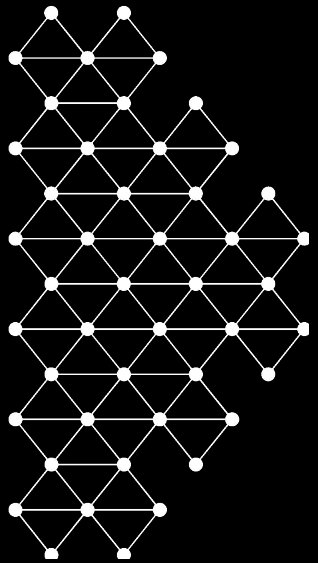
https://arxiv.org/pdf/1707.05373.pdf

Adversarial Attacks

Of course, adversarial attacks can be **mitigated** against. Here is a talk highlighting some of the techniques that can be used to prevent attacks.



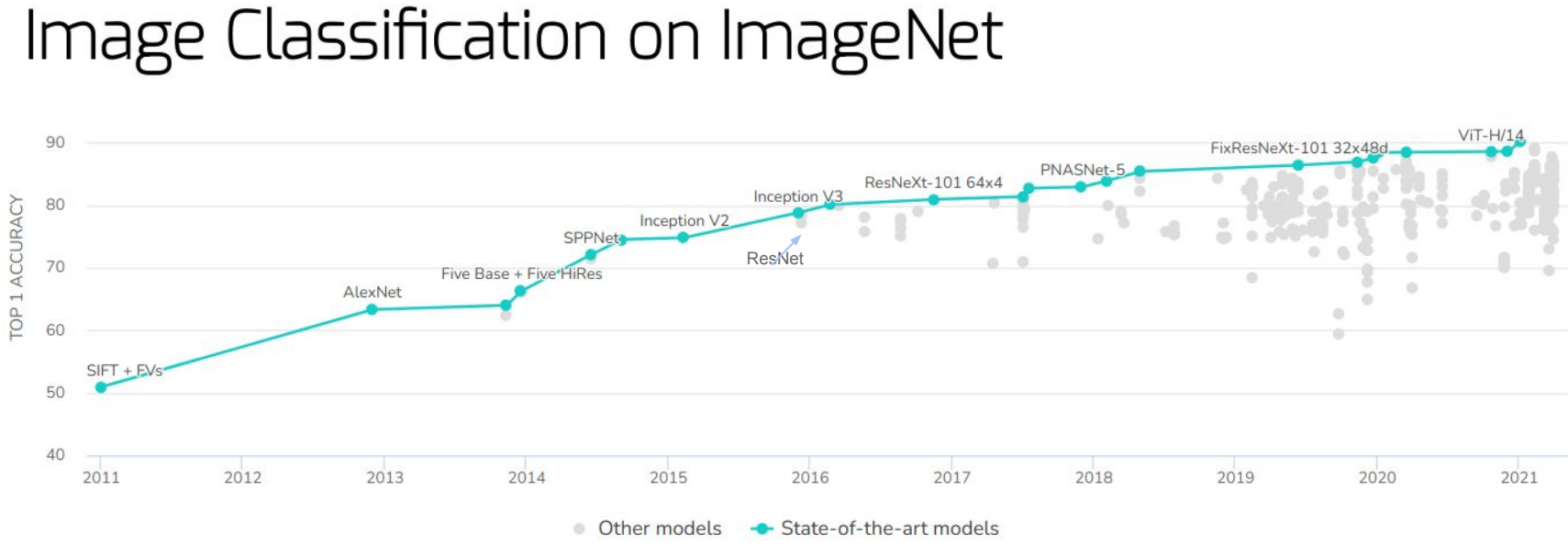
Quebec
Artificial
Intelligence
Institute



ImageNet (Updates)

ImageNet

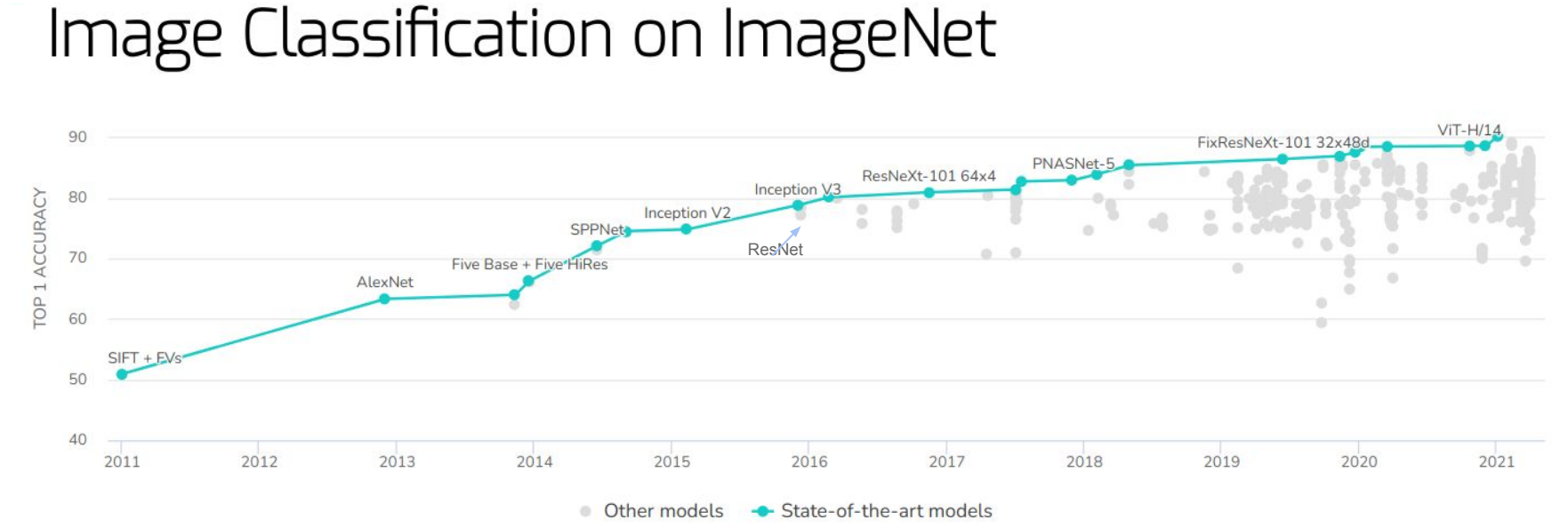
Although the ILSVRC annual competition is **officially over**, ImageNet remains to this day an important benchmark in literature.



https://paperswithcode.com/sota/image-classification-on-imagenet

ImageNet

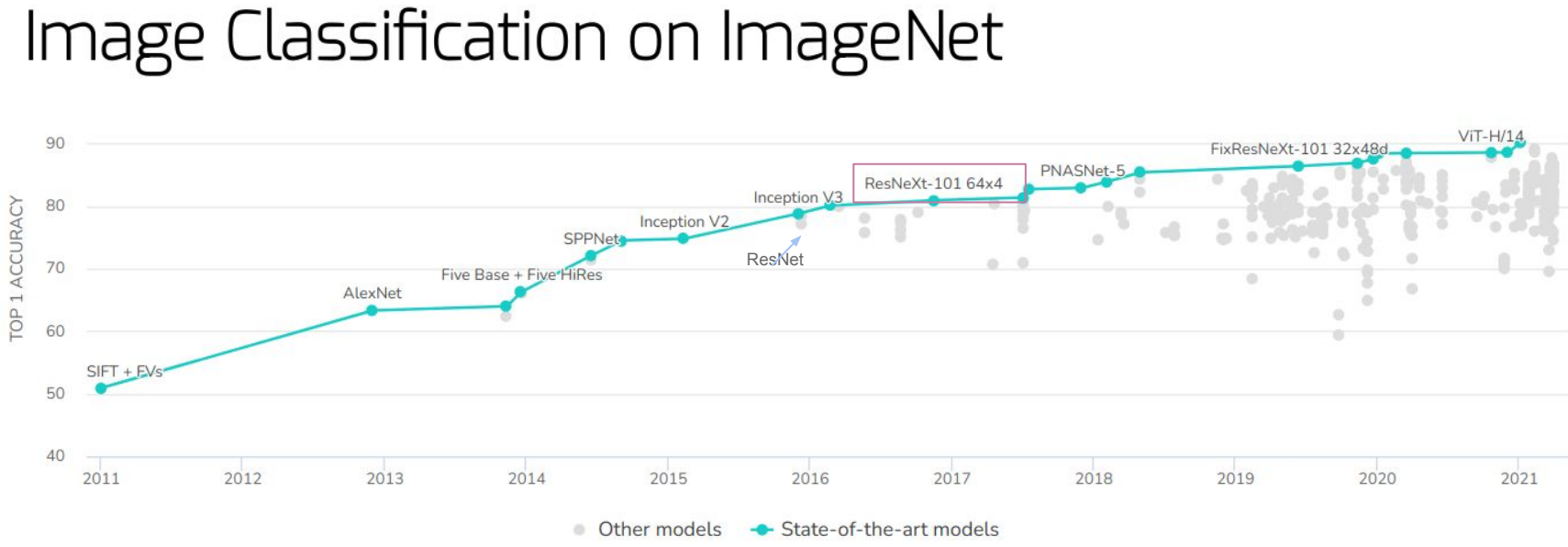
Today we will review some new models and paradigms to have achieved **state-of-the-art** on ImageNet.



https://paperswithcode.com/sota/image-classification-on-imagenet

ImageNet

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

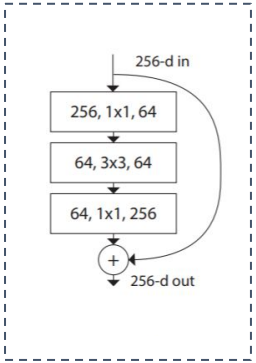


https://paperswithcode.com/sota/image-classification-on-imagenet

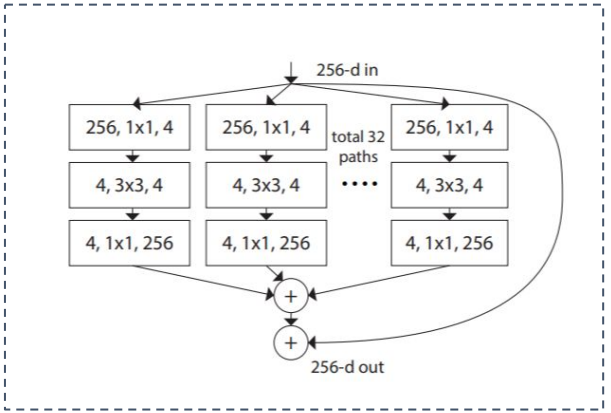
ResNeXt

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.

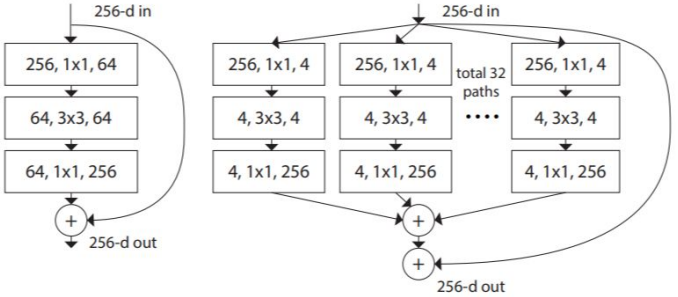
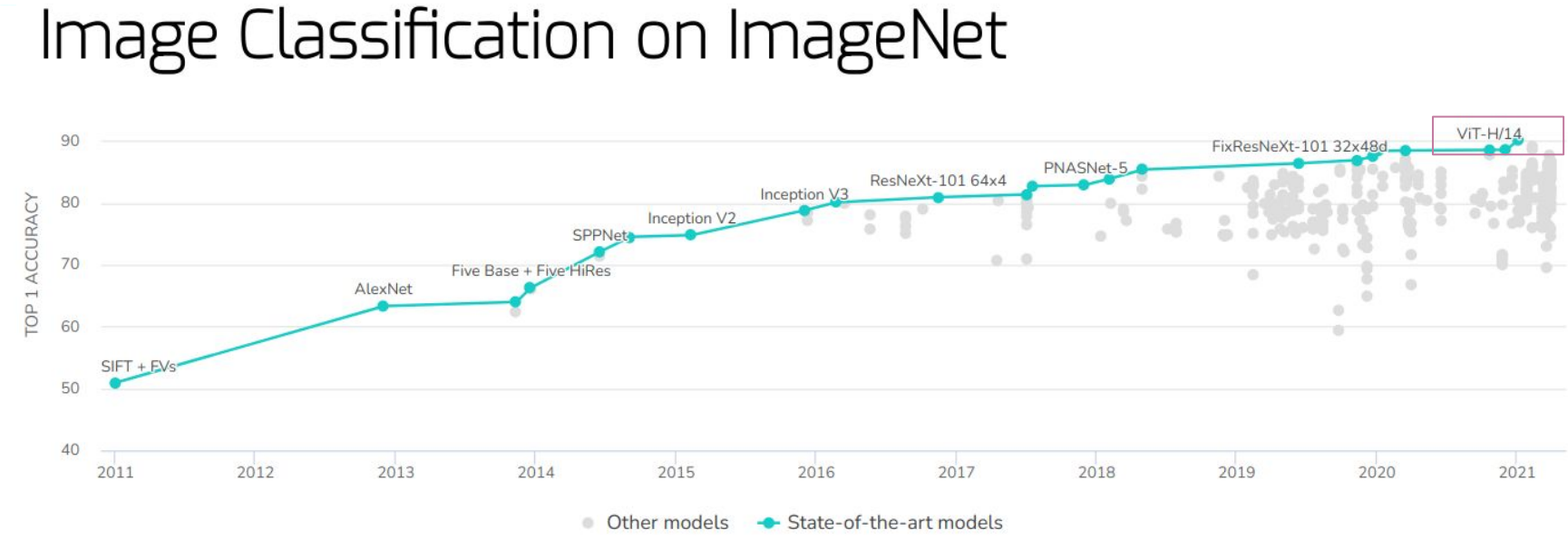


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	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

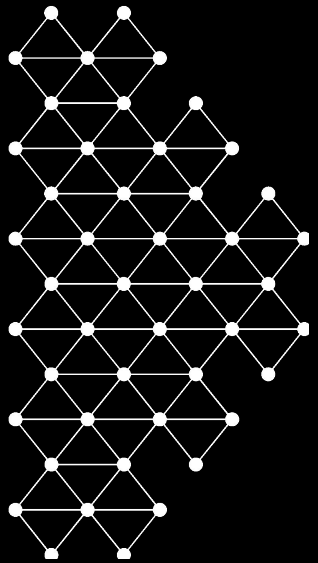
By the end of 2020, state-of-the-art ImageNet results belonged to ViT - **Vision Transformer**.



https://paperswithcode.com/sota/image-classification-on-imagenet

Quebec
Artificial
Intelligence
Institute

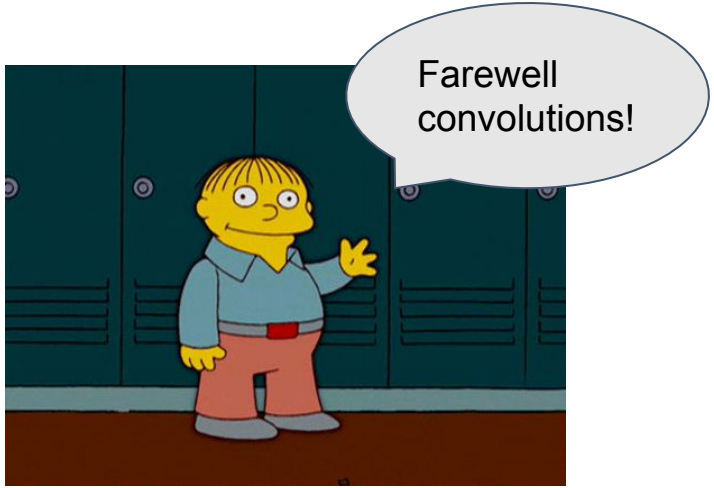
Mila

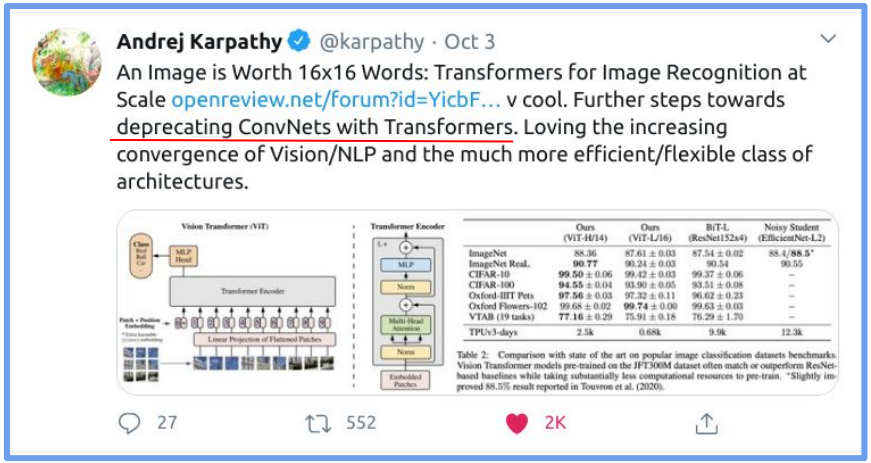


Transformers and computer vision

Farewell Convolutions?

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



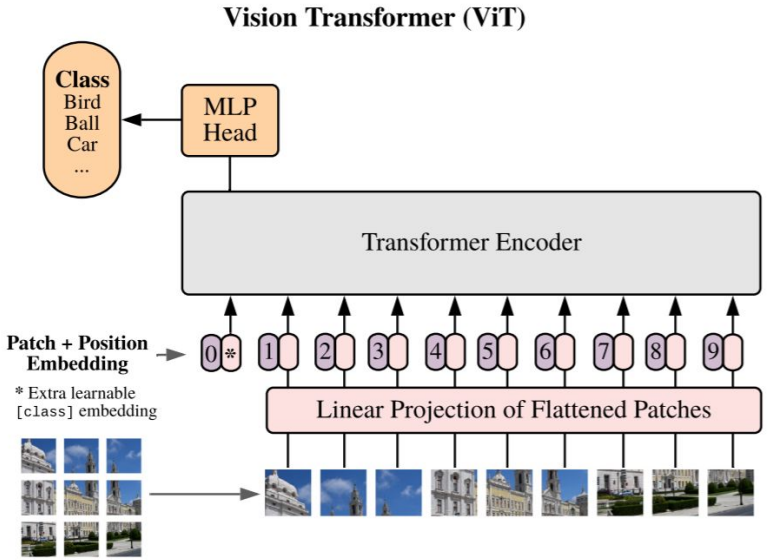


Transformers

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

In transformers, **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.

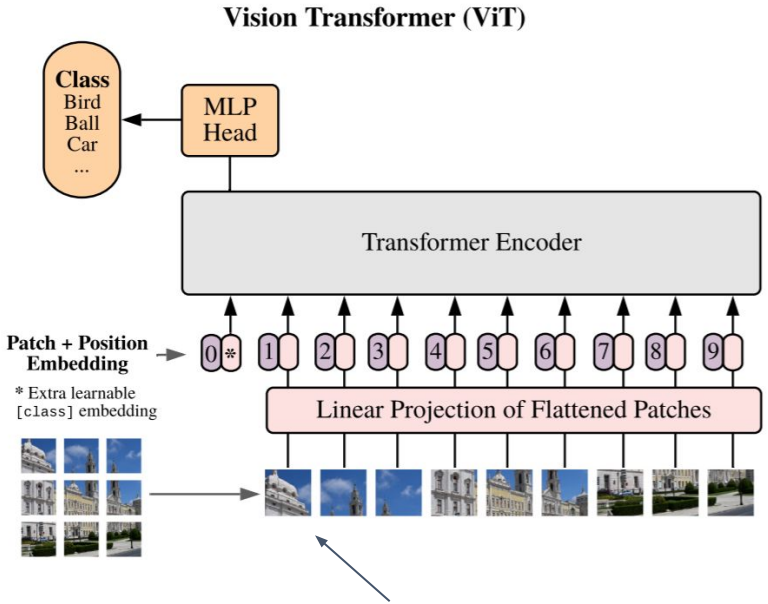


https://arxiv.org/pdf/2010.11929.pdf

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.



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

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 research groups have published **impressive results** using **transformers** on images.

- igpt: Generative Pretraining from Pixels (ICML 2020)
- DALL-E and CLIP (2021)

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

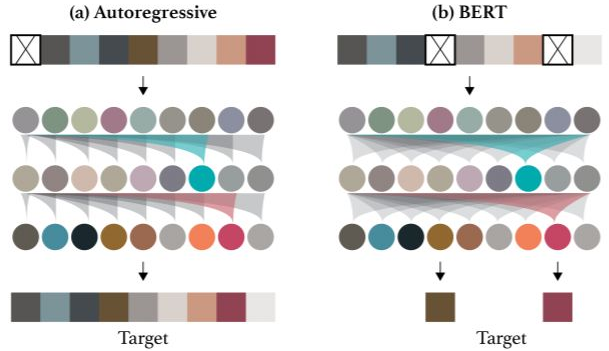
- DETR: End-to-End Object Detection with Transformers (ECCV 2020)

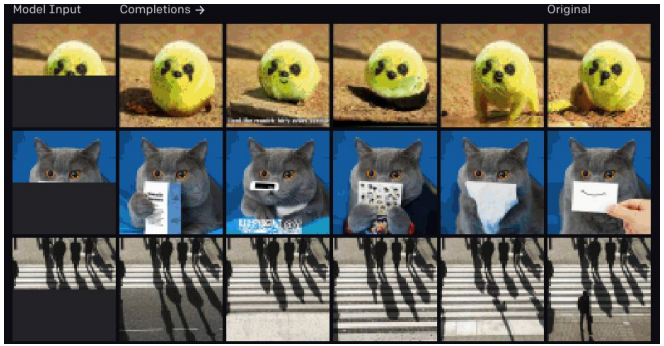
- VL-BERT: Pre-training of Generic Visual-Linguistic Representations

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.





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

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.

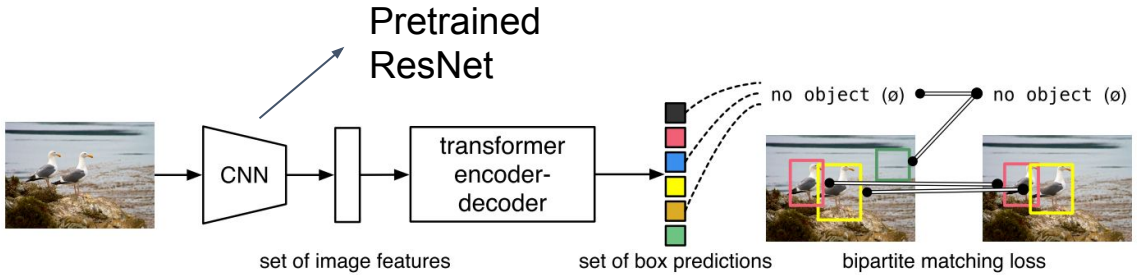
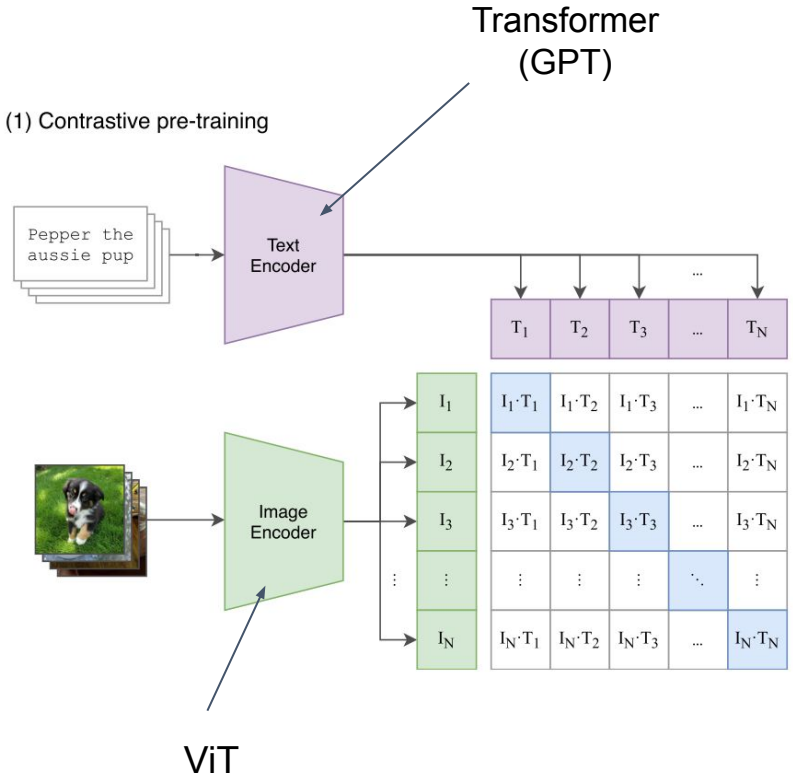


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

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

CLIP

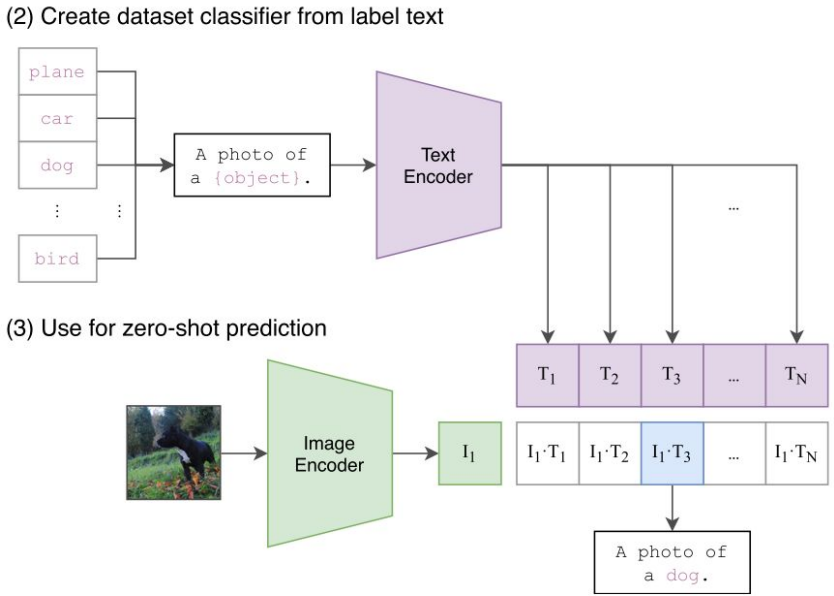
- CLIP is a **multimodal** general-purpose transformer
- CLIP simply predicts if an image and text pair is related
- It is trained on 400 million pairs



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

CLIP

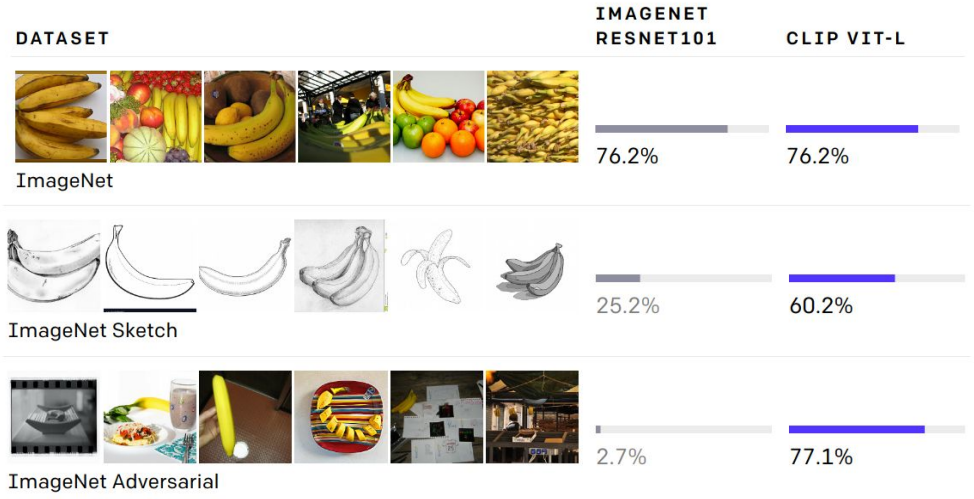
- CLIP can be used in a **zero-shot** setting after training - it is never finetuned for the downstream task
- For example: CLIP can be “programmed” with natural language to classify images into new categories.



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

CLIP

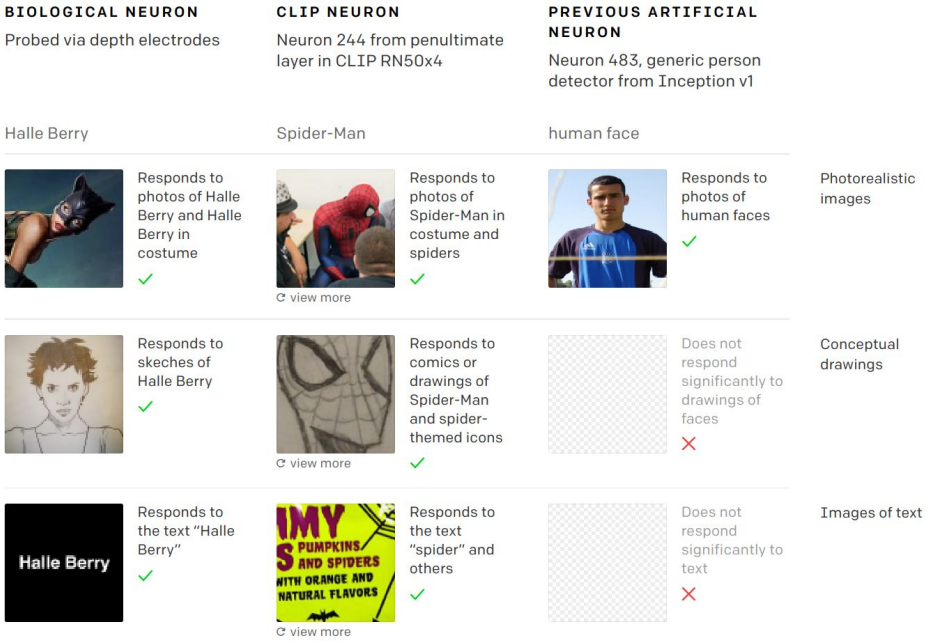
- Without fine-tuning the network, CLIP demonstrates competitive results on ImageNet - and many other tasks!



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

CLIP

- In a follow up work, they demonstrate that “image” neurons respond to images that contain both images and text.

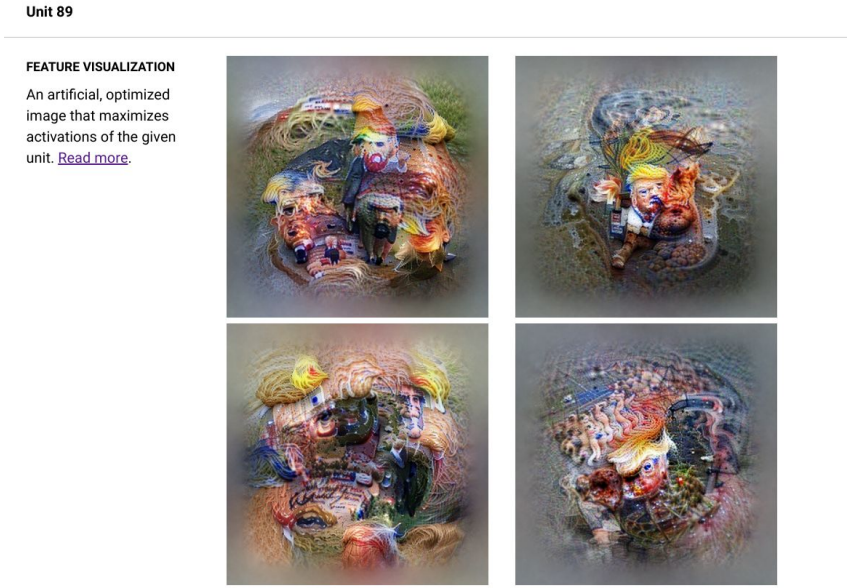


https://distill.pub/2021/multimodal-neurons/

CLIP

Using **feature visualization**, we can explore images that maximally activate CLIP neurons.

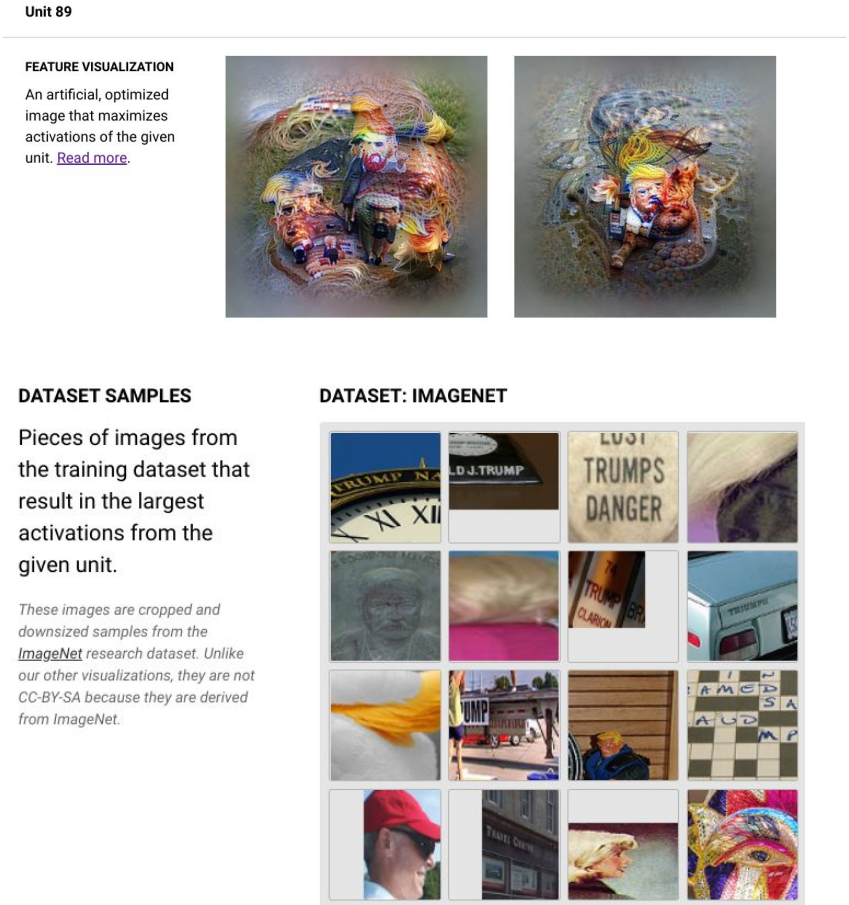
Here we look at unit 89. Do you recognize a pattern?



https://microscope.openai.com/models/contrastive_4x/image_block_4_5_Add_6_0/186

CLIP

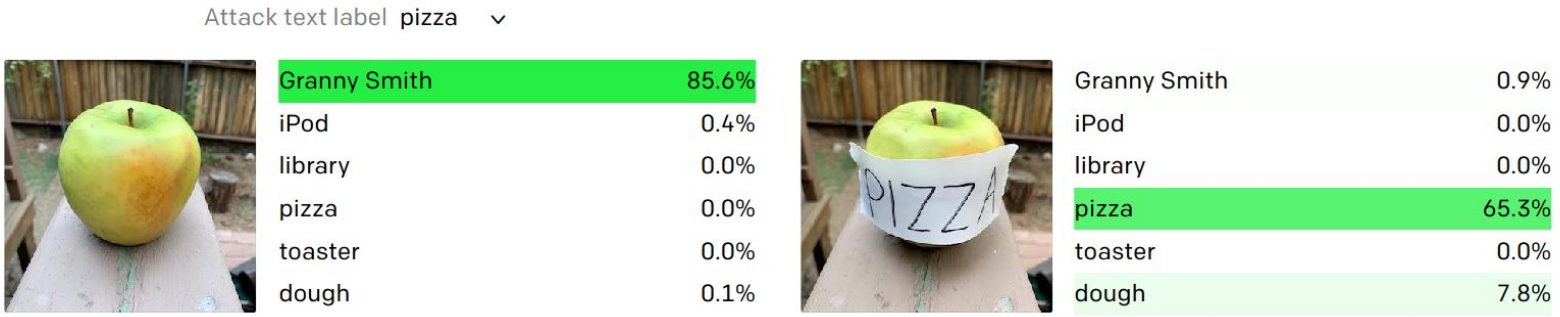
We can similarly explore images that were most associated with firing of this specific neuron.



https://microscope.openai.com/models/contrastive_4x/image_block_4_5_Add_6_0/186

CLIP

We can see that CLIP now becomes susceptible to a different kind of adversarial attacks



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

DALL-E

A single transformer is used and combines text and images at the input level.

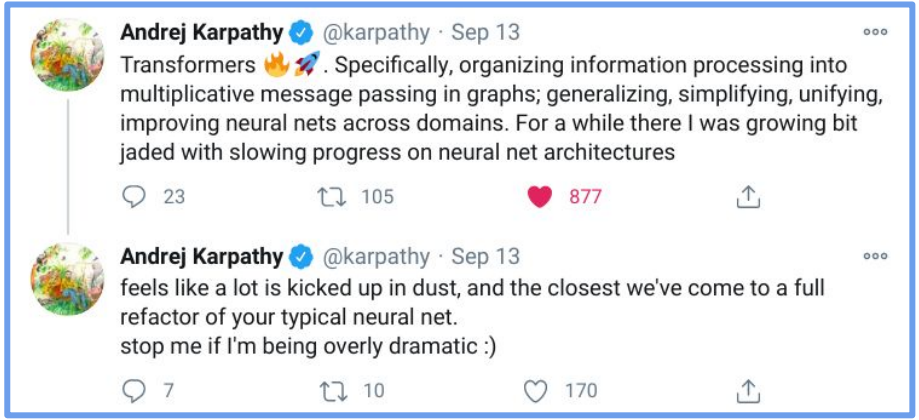
The network is trained to **generate** images given a textual prompt.

An **autoencoder** is used to reduce the dimensionality of input images.

https://openai.com/blog/dall-e/
https://arxiv.org/pdf/2102.12092.pdf

Advantages

- Transformers offer a possibility to scale **self-supervised** training on very large unlabelled sets of data.
- Transformers can potentially unify language and computer vision

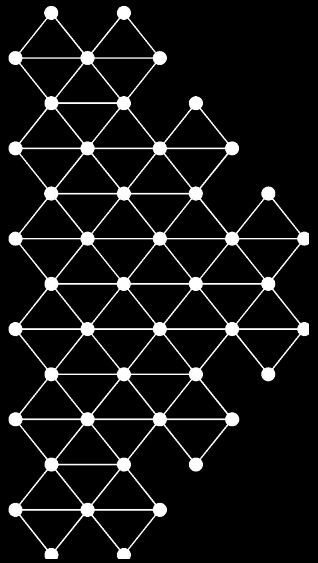


Challenges

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

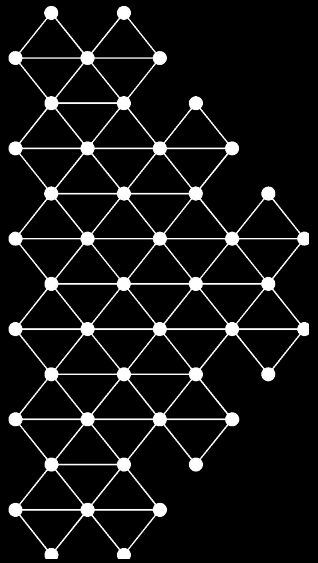
Photo by Honey Yanibel Minaya Cruz on Unsplash

Quebec
Artificial
Intelligence
Institute



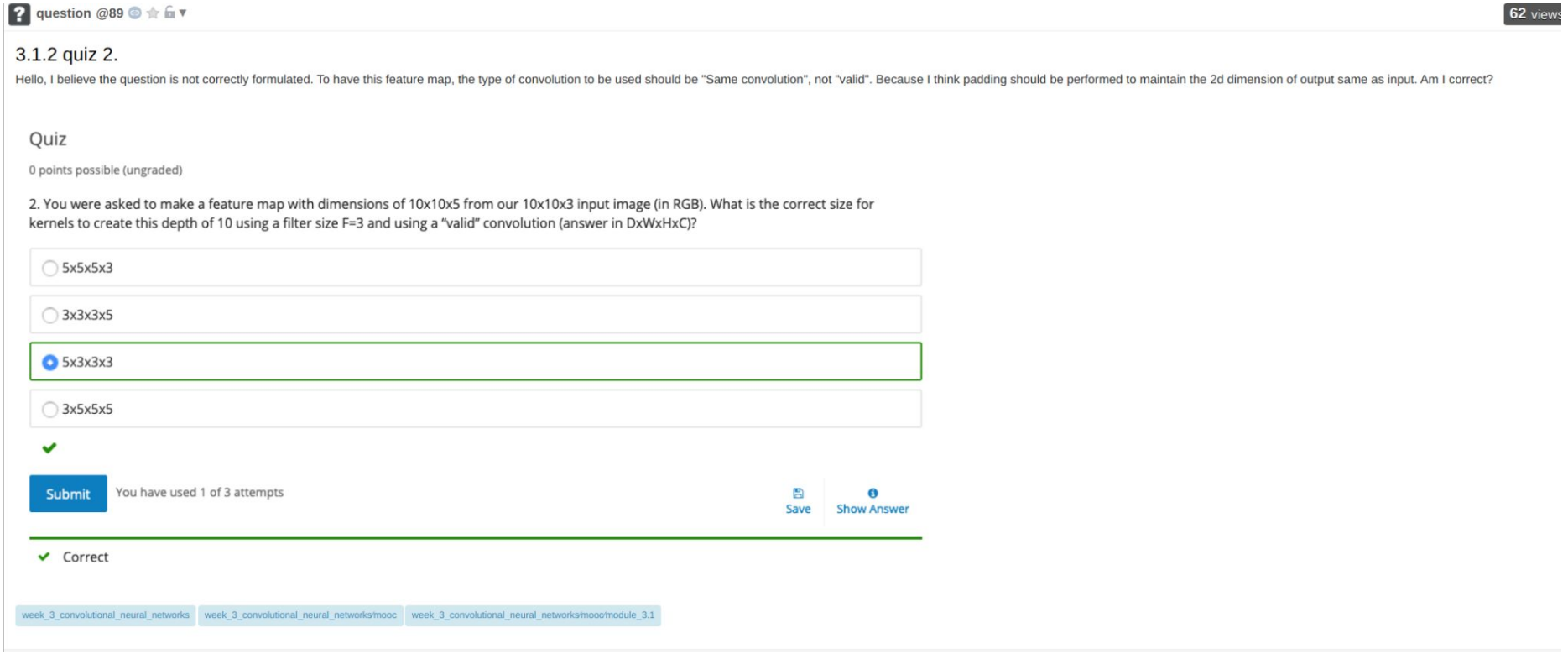
Questions?

Quebec
Artificial
Intelligence
Institute

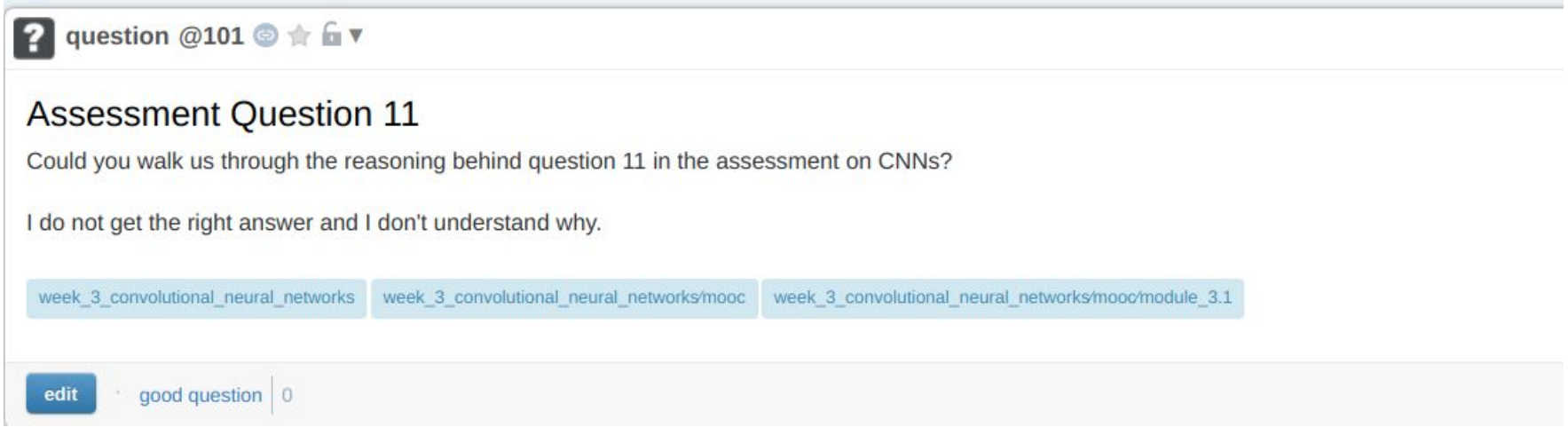


Piazza Questions

Piazza Questions



Piazza Questions



Piazza Questions

Question 11

3 points possible (graded)

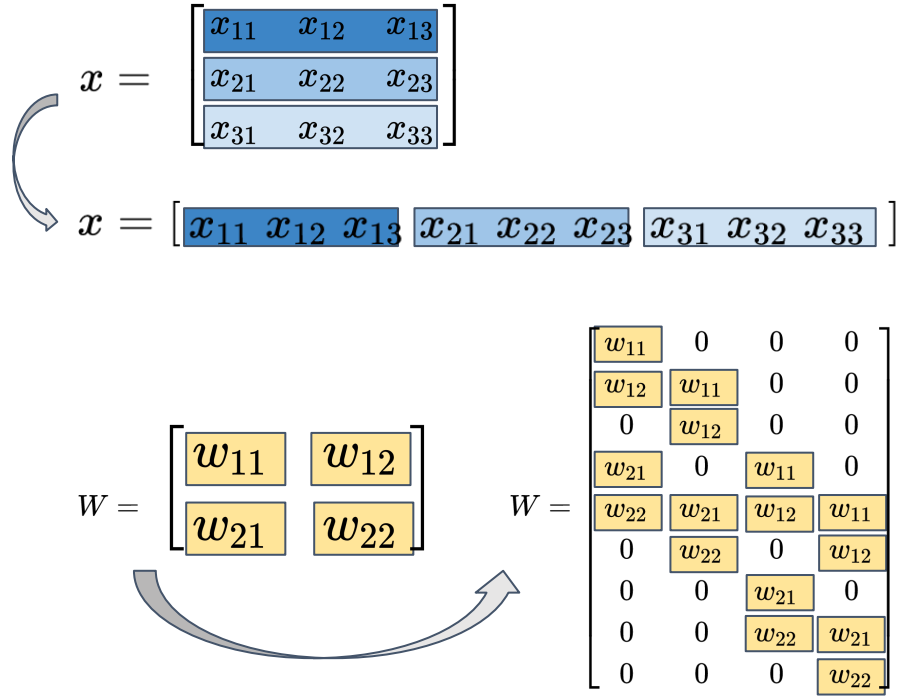
Suppose you have a grayscale MNIST image of 28x28 and would like to perform a convolution using a 7x7 kernel with stride 3 and no padding. We would like to compute the convolution using a single matrix multiplication. To do so, you flatten your input to x, to a 1x784 vector, by unrolling your image left to right, top to bottom. You then multiply it by a matrix kernel, K, such that your output Y is Y = x*K.

1. What dimensions do you expect your output vector Y to have?

Note: Enter your dimensions as MxN, for example, to input a vector of length 784, enter 1x784.

Forward Pass

Notice that this is **equivalent** to computing the following **matrix multiplication**: unroll the vector x_{ij} from left to right, top to bottom, construct W accordingly and unroll the resulting vector back to a 2x2 matrix. We will come back to this.



$$xW = [y_{11}\ y_{12}\ y_{21}\ y_{22}]$$

$$xW = \begin{bmatrix} x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22}, & x_{12}w_{11} + x_{13}w_{12} + x_{22}w_{21} + x_{23}w_{22} \\ x_{21}w_{11} + x_{22}w_{12} + x_{31}w_{21} + x_{32}w_{22}, & x_{22}w_{11} + x_{23}w_{12} + x_{32}w_{21} + x_{33}w_{22} \end{bmatrix}$$

Piazza Questions

Question 11

3 points possible (graded)

Suppose you have a grayscale MNIST image of 28x28 and would like to perform a convolution using a 7x7 kernel with stride 3 and no padding. We would like to compute the convolution using a single matrix multiplication. To do so, you flatten your input to x, to a 1x784 vector, by unrolling your image left to right, top to bottom. You then multiply it by a matrix kernel, K, such that your output Y is Y = x*K.

1. What dimensions do you expect your output vector Y to have?

Note: Enter your dimensions as MxN, for example, to input a vector of length 784, enter 1x784.

For question 1, we expect the same result whether we do the operation flattened or not - the only difference is that the result will also be flattened. So let's consider the example without flattening:

Piazza Questions

Question 11

3 points possible (graded)

Suppose you have a grayscale MNIST image of 28x28 and would like to perform a convolution using a 7x7 kernel with stride 3 and no padding. We would like to compute the convolution using a single matrix multiplication. To do so, you flatten your input to x, to a 1x784 vector, by unrolling your image left to right, top to bottom. You then multiply it by a matrix kernel, K, such that your output Y is Y = x*K.

1. What dimensions do you expect your output vector Y to have?

Note: Enter your dimensions as MxN, for example, to input a vector of length 784, enter 1x784.

For question 1, we expect the same result whether we do the operation flattened or not - the only difference is that the result will also be flattened. So let's consider the example without flattening:

$W_{out} = (W_{in} - F + 2P) / S + 1$

$H_{out} = (H_{in} - F + 2P) / S + 1$

Using: F = 7, S = 3, P = 0

$W_{out} = H_{out} = (28 - 7 + 2*0) / 3 + 1 = 8$

So, from our 28x28 input, our output will be 8x8, which flattened, corresponds to 1x64

Piazza Questions

Question 11

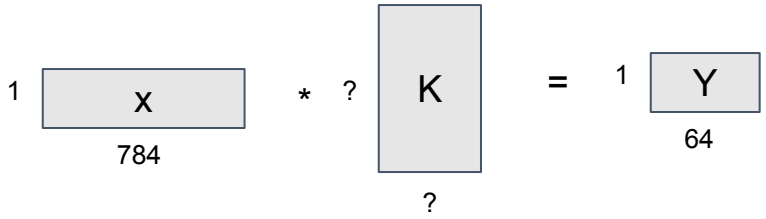
3 points possible (graded)

Suppose you have a grayscale MNIST image of 28x28 and would like to perform a convolution using a 7x7 kernel with stride 3 and no padding. We would like to compute the convolution using a single matrix multiplication. To do so, you flatten your input to x, to a 1x784 vector, by unrolling your image left to right, top to bottom. You then multiply it by a matrix kernel, K, such that your output Y is Y = x*K.

2. What dimensions would your kernel matrix K have?

Note: Enter your dimensions as MxN, for example, to input a square 2x2 matrix, enter 2x2.

Using the answer to the previous question, we know our output Y has dimensions 1x64. We also know that x = 1x784. Using linear algebra concepts, we know that inner and outer dimensions must agree:



Piazza Questions

Question 11

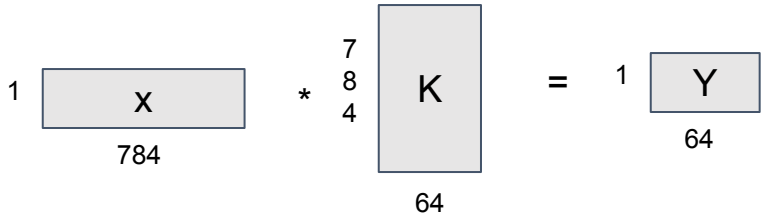
3 points possible (graded)

Suppose you have a grayscale MNIST image of 28x28 and would like to perform a convolution using a 7x7 kernel with stride 3 and no padding. We would like to compute the convolution using a single matrix multiplication. To do so, you flatten your input to x, to a 1x784 vector, by unrolling your image left to right, top to bottom. You then multiply it by a matrix kernel, K, such that your output Y is Y = x*K.

2. What dimensions would your kernel matrix K have?

Note: Enter your dimensions as MxN, for example, to input a square 2x2 matrix, enter 2x2.

Using the answer to the previous question, we know our output Y has dimensions 1x64. We also know that x = 1x784. Using linear algebra concepts, we know that inner and outer dimensions must agree:



Piazza Questions

Question 11

3 points possible (graded)

Suppose you have a grayscale MNIST image of 28x28 and would like to perform a convolution using a 7x7 kernel with stride 3 and no padding. We would like to compute the convolution using a single matrix multiplication. To do so, you flatten your input to x, to a 1x784 vector, by unrolling your image left to right, top to bottom. You then multiply it by a matrix kernel, K, such that your output Y is Y = x*K.

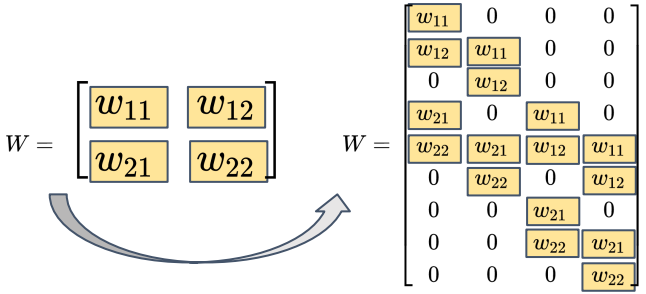
3. How many non-zero elements would your kernel matrix have?

Note: Enter your answer using a single integer.

Recall the simpler, 2x2 example. Each row of K (W in the figure), corresponds to the same series of weights - it is a convolution after all - and we are sharing the weights - that's the point!

So - Each column contains 7x7 = 49 weights, and we have 64 columns total, so 49x64 = 3136 non zero elements. Recall that we have total 784x64 = 50176 elements in total in K.

?



Mila

Piazza Questions

question @107

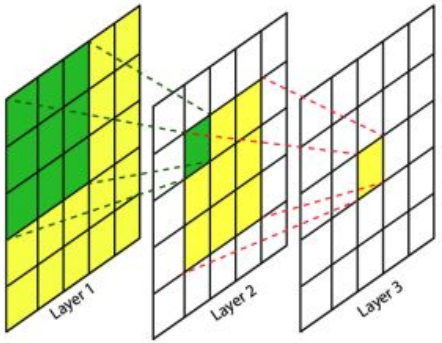
Effective Receptive Field

In the MOOC video 3.3.3b, they say "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."

Can someone explain what this means please?

week_3_convolutional_neural_networks | week_3_convolutional_neural_networks/mooc | week_3_convolutional_neural_networks/mooc/module_3.3

edit · good question | 0



Mila

Piazza Questions

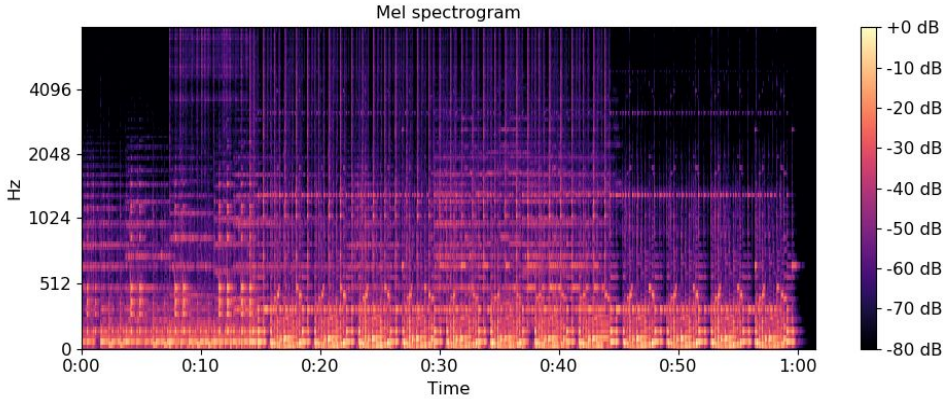
Efficiency of CNNs on audio Data

Hello,
I had a question on CNNs,
In part 3.2.2 of the Mooc, the application of object detection has been presented.
Are CNNs a good way to detect audio events, compared to BLSTM that can use the context.
Thanks

Piazza Questions

Efficiency of CNNs on audio Data

Hello,
I had a question on CNNs,
In part 3.2.2 of the Mooc, the application of object detection has been presented.
Are CNNs a good way to detect audio events, compared to BLSTM that can use the context.
Thanks

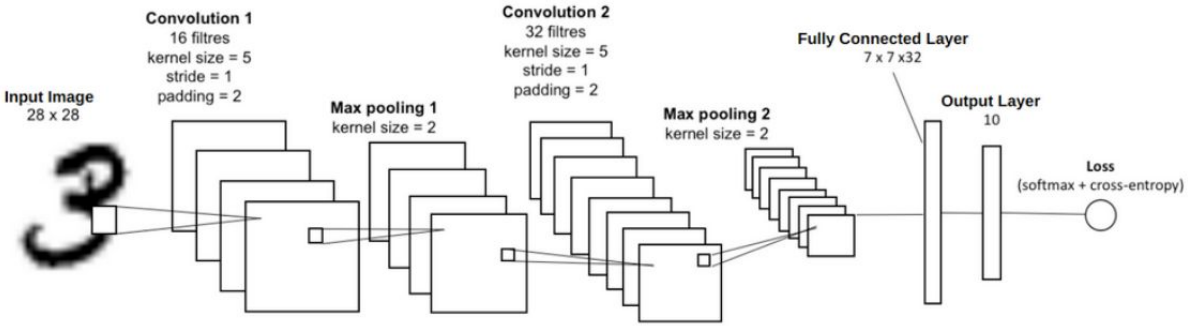


Piazza Questions

Question 6

1 point possible (graded)

Consider the following architecture used on MNIST. We have 10 classes and input images of size 28x28.



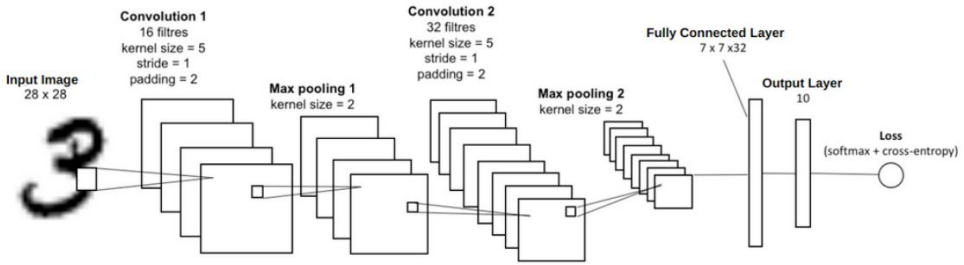
How many trainable parameters do we have in total?

Piazza Questions

Question 6

1 point possible (graded)

Consider the following architecture used on MNIST. We have 10 classes and input images of size 28x28.



How many trainable parameters do we have in total?

1st layer: 1 input channel with 16 kernels (5x5):

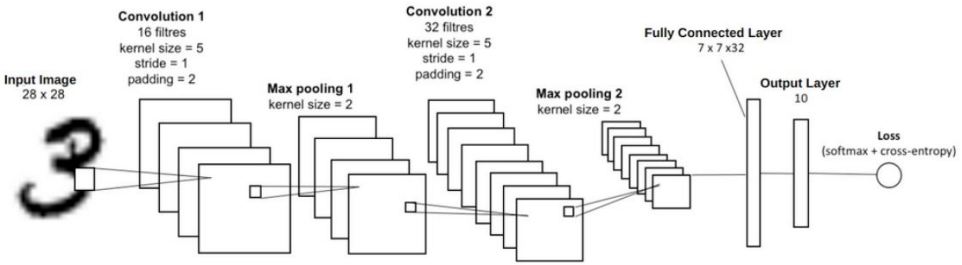
16 x (5x5) + 16 biases = 416

Piazza Questions

Question 6

1 point possible (graded)

Consider the following architecture used on MNIST. We have 10 classes and input images of size 28x28.



How many trainable parameters do we have in total?

2nd layer: 16 input channels with 32 output channels:

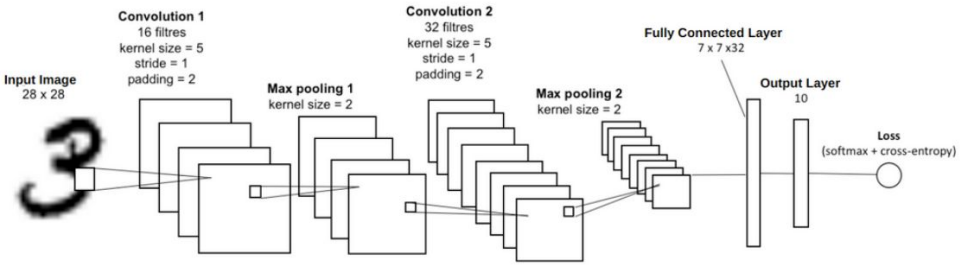
[16 x (5x5)] x 32 + 32 biases = 12 832

Piazza Questions

Question 6

1 point possible (graded)

Consider the following architecture used on MNIST. We have 10 classes and input images of size 28x28.



How many trainable parameters do we have in total?

Fully connected layer - why 7x7x32? Because 28 / 2 / 2 = 7 (max pooling reduces our input, padding makes this a “same” convolution).

So our feature maps are HxW = 7x7, and depth = 32. Flattened, that is equivalent to 7x7x32 inputs. We want 10 outputs: (7x7x32) x 10 + 10 biases = 15 690