

Introduction to High Performance Machine Learning

**Convolutional Neural Networks
Recurrent Neural Networks
Generative Models**

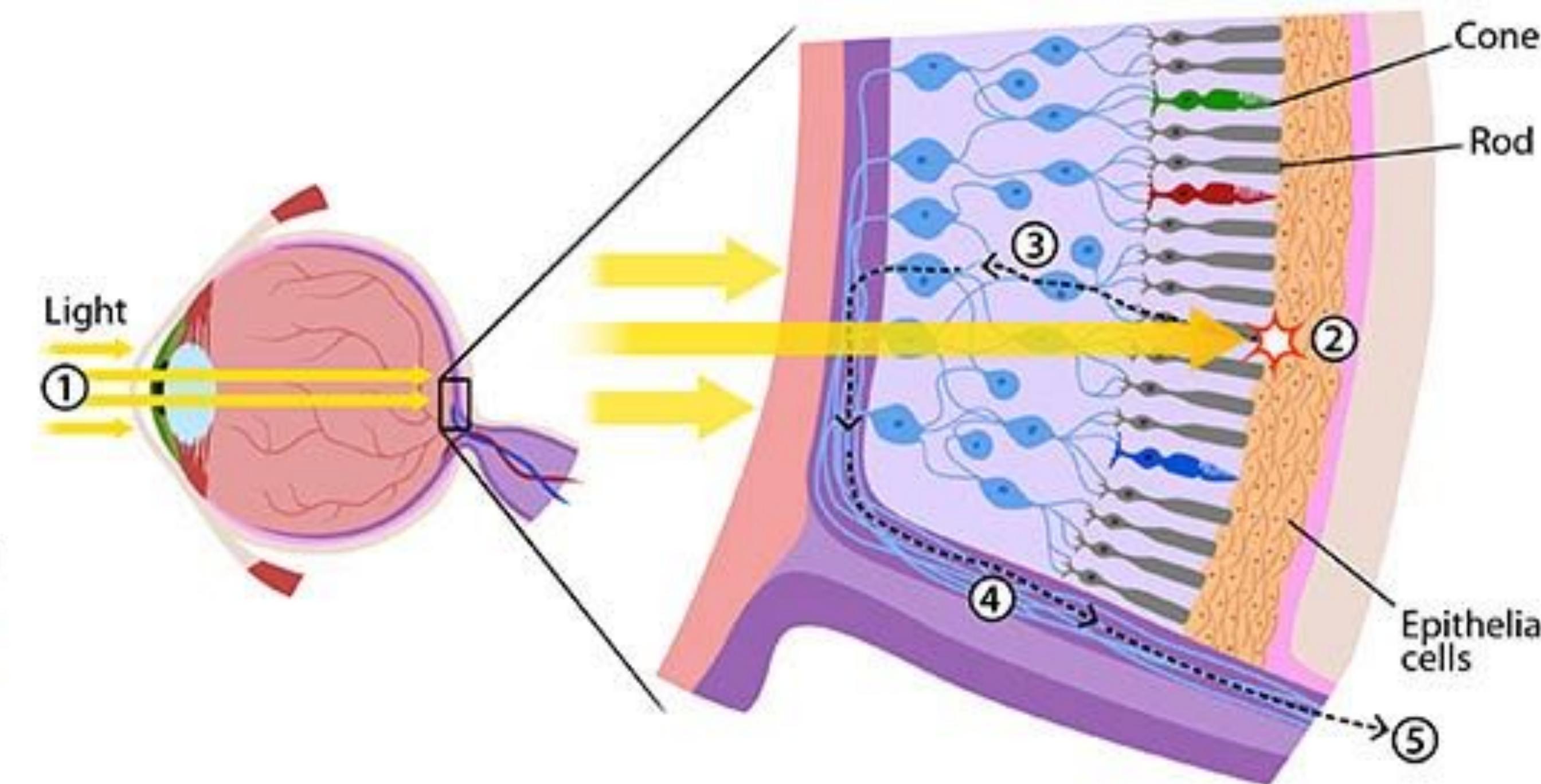
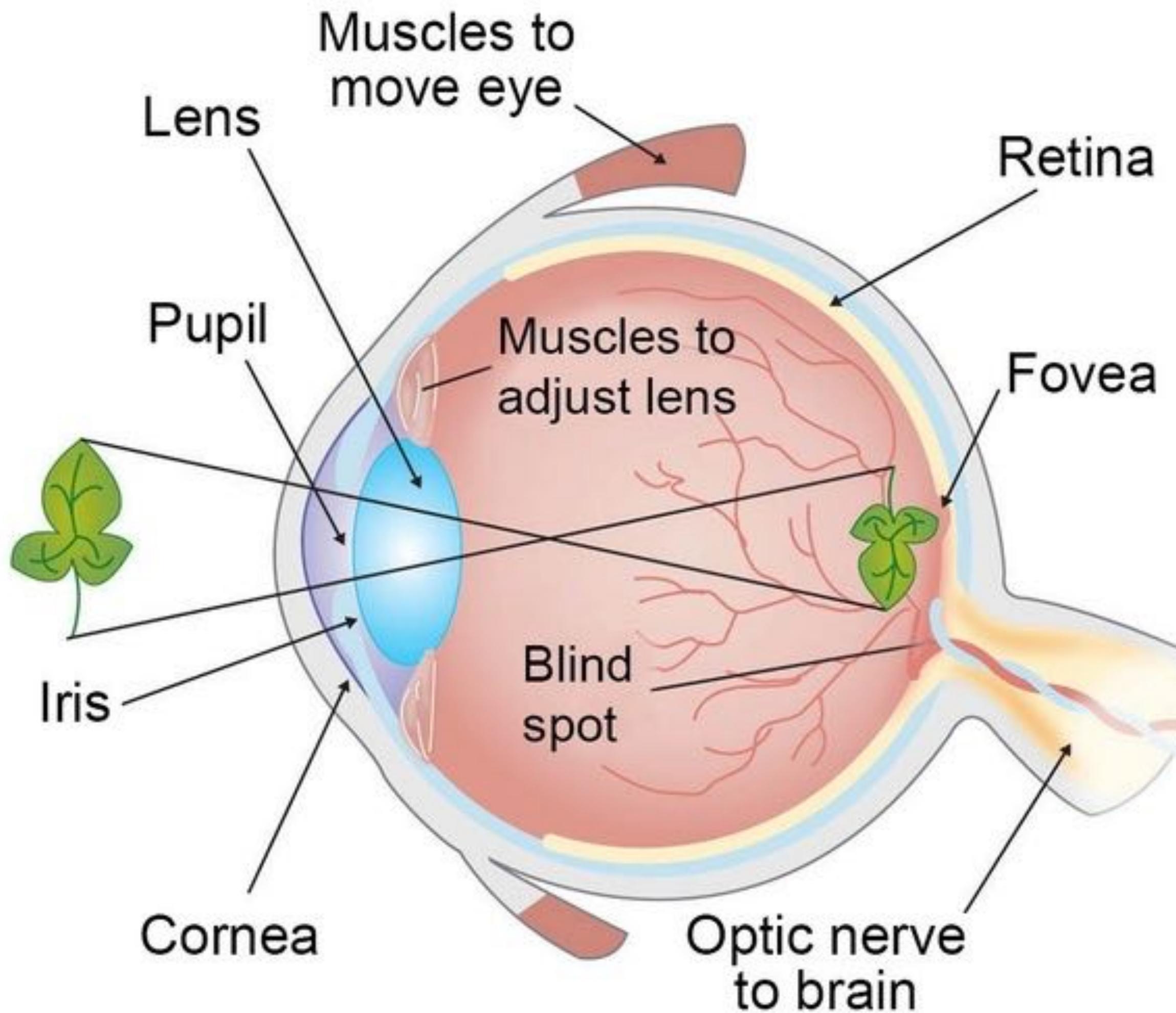
Maxwell Cai, PhD

Welcome to the world of computer vision!

How do we let computers “see” something?

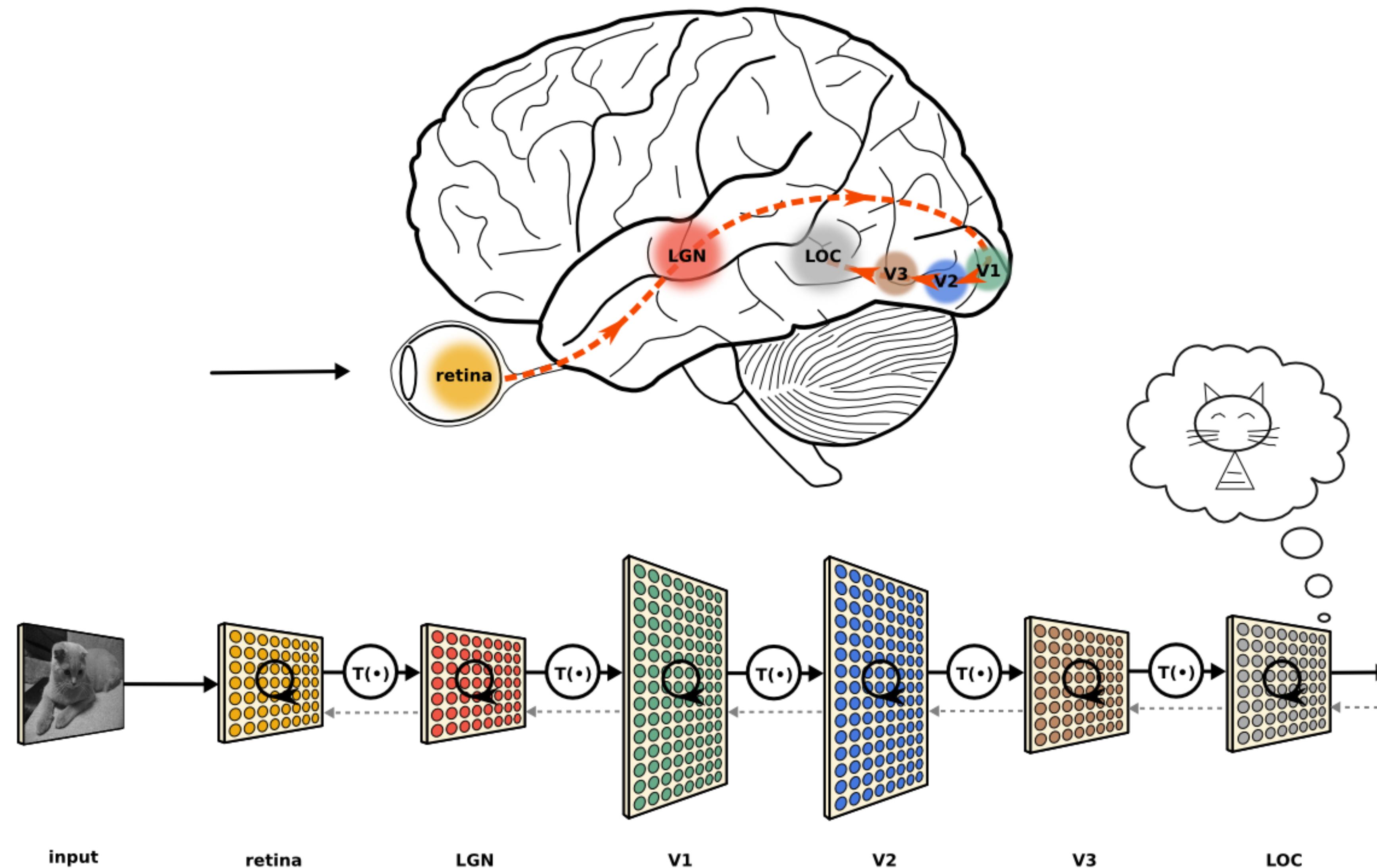
Ask ourselves first: How do we (humans) see something?

Human eyes



Human brain: the (real) neural network

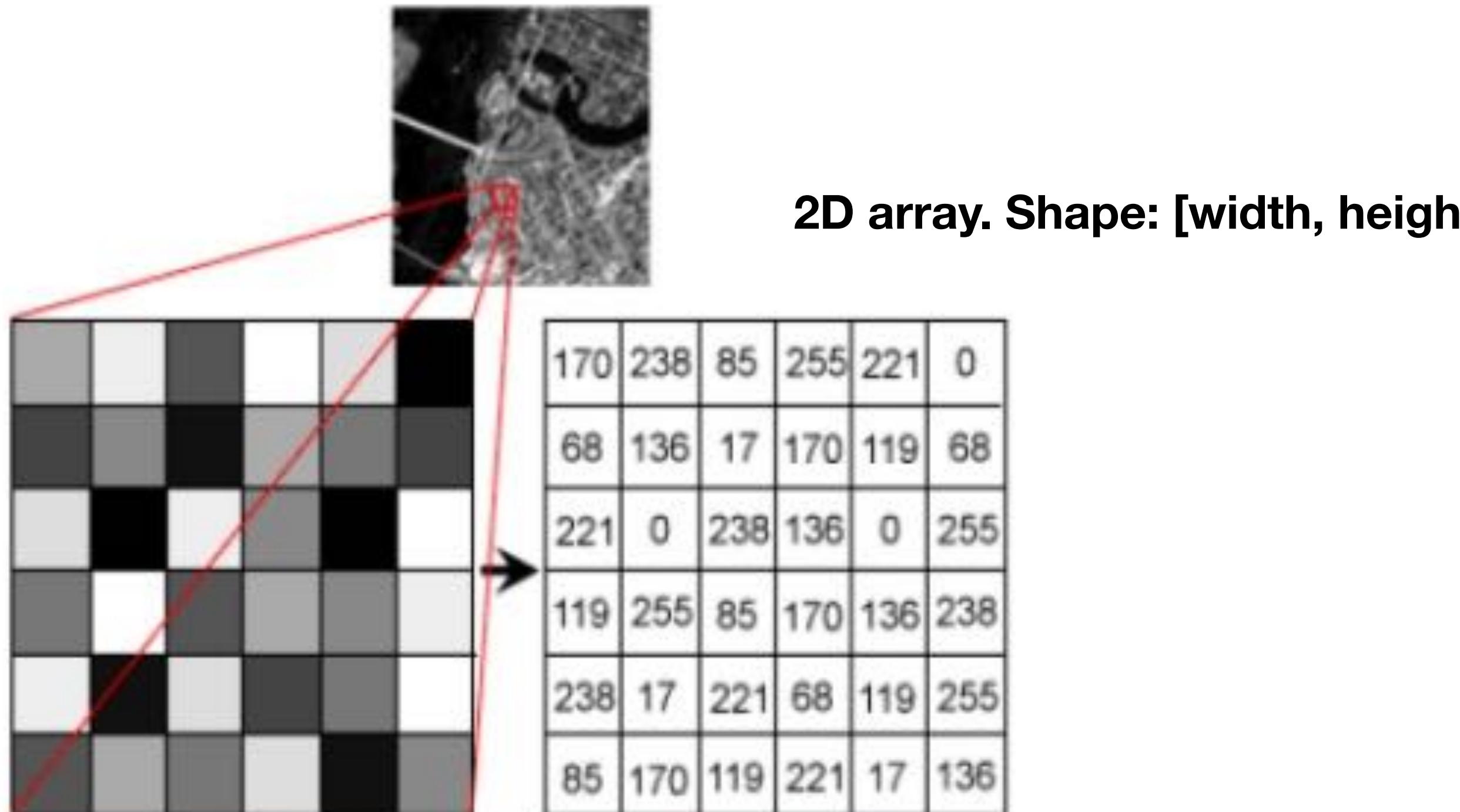
Source: arimaresearch.com



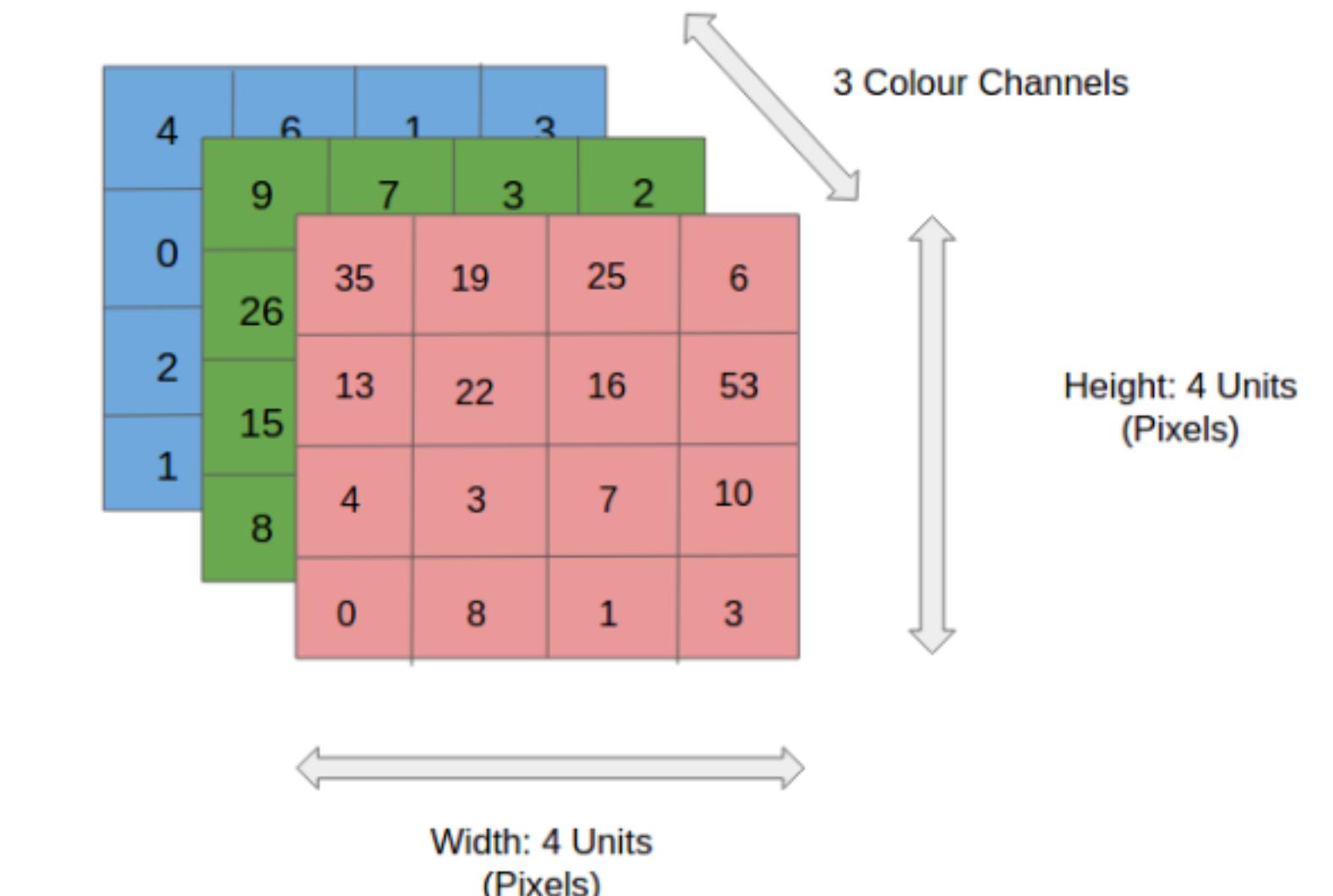
Representation of images in computers

An image

- ... is a **matrix/array** of intensity values
- ... usually consists integers of [0, 255] or float points of [0, 1]
- ... each element of this matrix is called a **pixel**
- ... can have 1 (greyscale) or multiple (color) **channels**



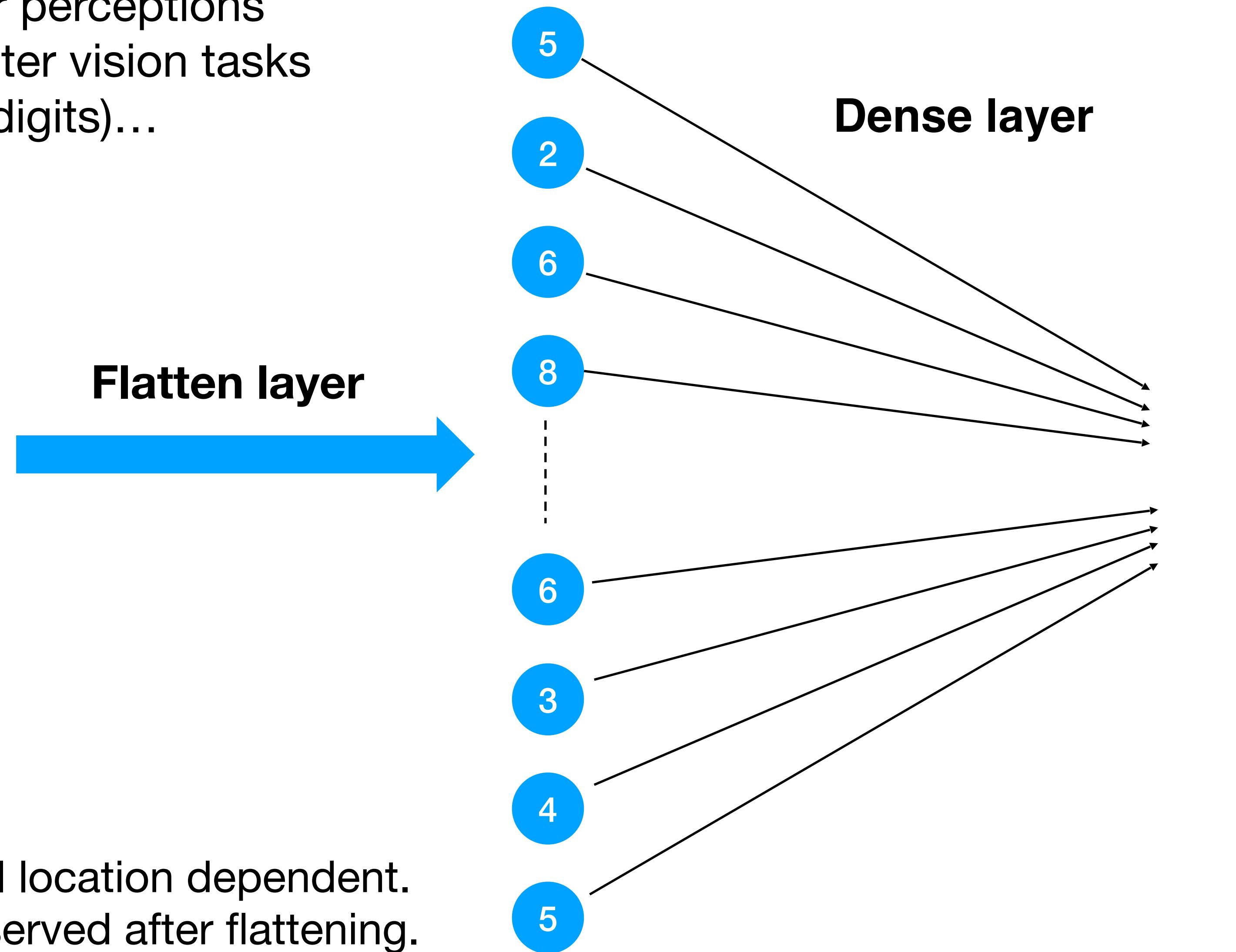
3D array. Shape: [width, height, channel]



Dense network classifier (MLP)

So far, we have used multi-layer perceptions (MLP) to carry out some computer vision tasks (e.g., recognizing hand-written digits)...

5	2	6	8	2	0	1	2
4	3	4	5	1	9	6	3
3	9	2	4	7	7	6	9
1	3	4	6	8	2	2	1
8	4	6	2	3	1	8	8
5	8	9	0	1	0	2	3
9	2	6	6	3	6	2	1
9	8	8	2	6	3	4	5



Problem:

- The resulting encoding is pixel location dependent.
- Spatial relationship is not preserved after flattening.

Filters in our daily lives



Forget about the commercial advertisements themselves...
Just think about the mathematical principles behind them...



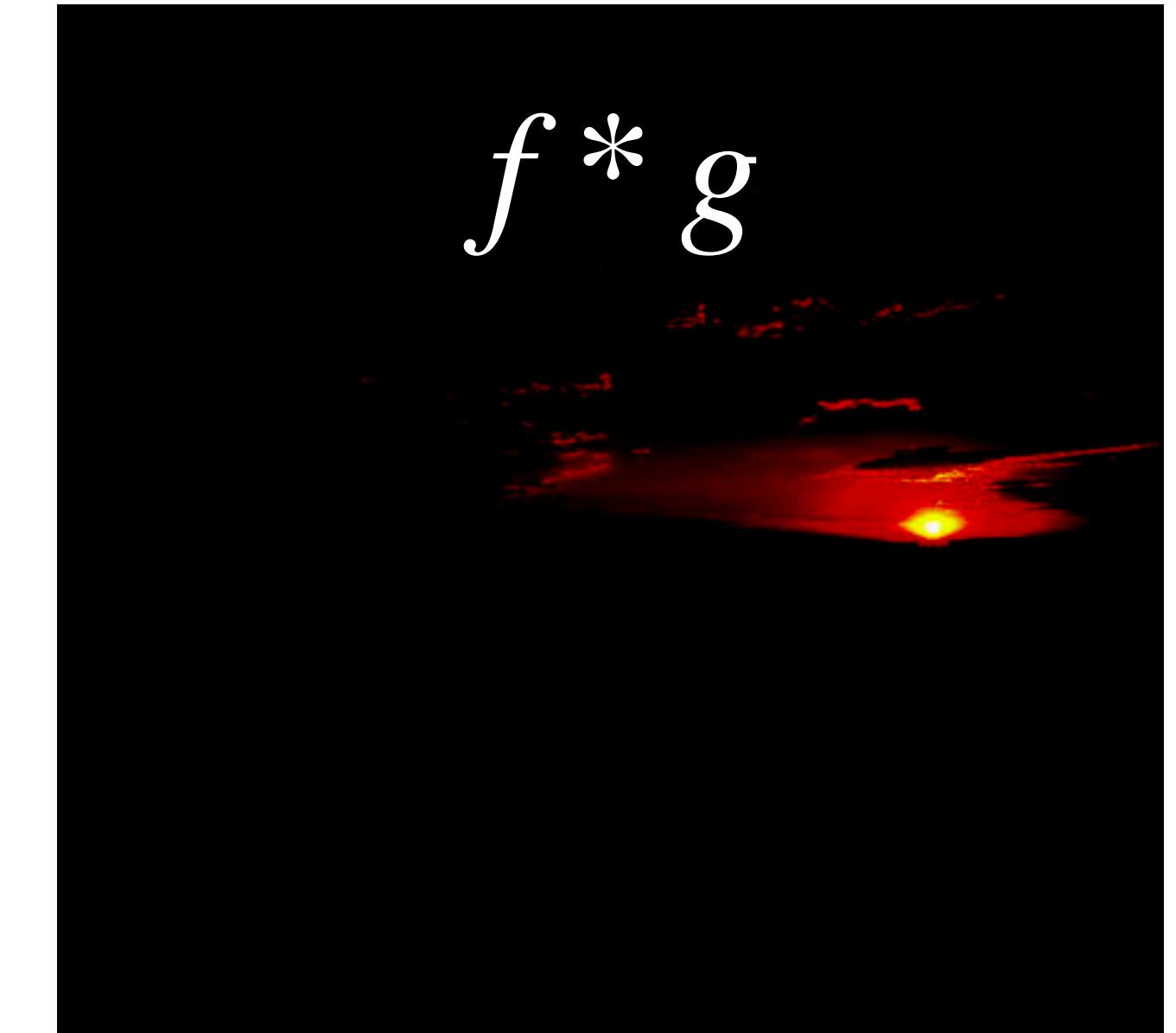
Filter in image processing

 f

+

Filter
 g

=

 $f * g$

Filter
 h

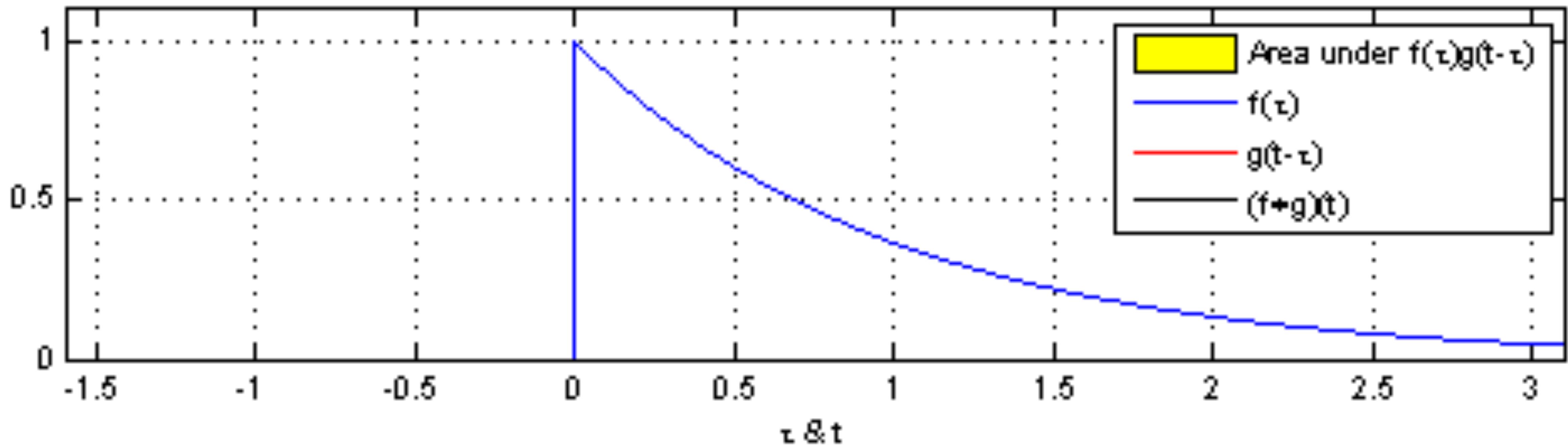
=

 $f * h$

Using **different filters**, we can see the **same signal** in **different perspectives**.

Convolution

Typically, a filter applies a **convolution** operation upon the original signal.



Function: $f(t)$

Kernel: $g(t)$

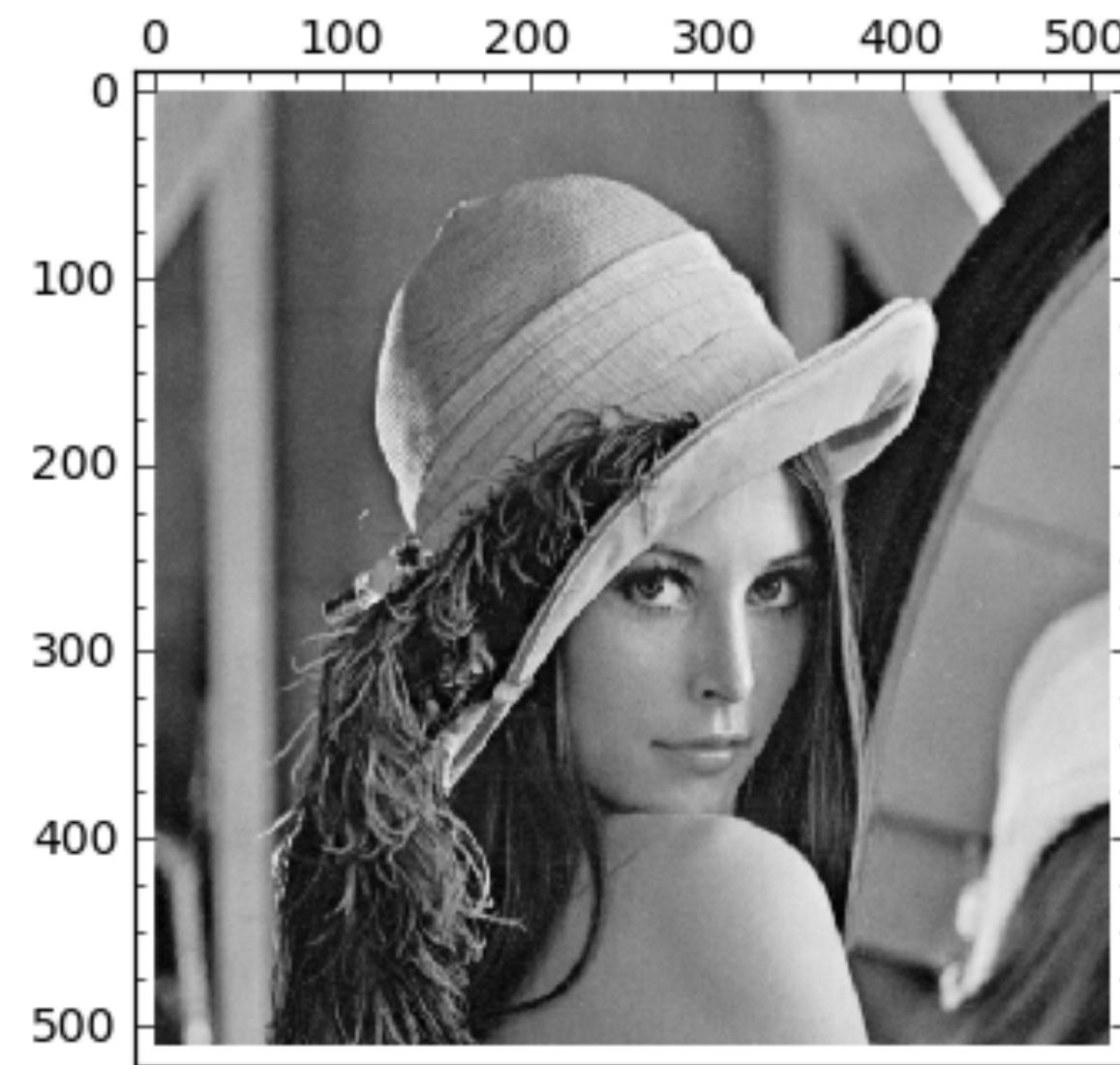
Convolution: $(f * g)(t)$

No signal → no response
Strong signal → strong response

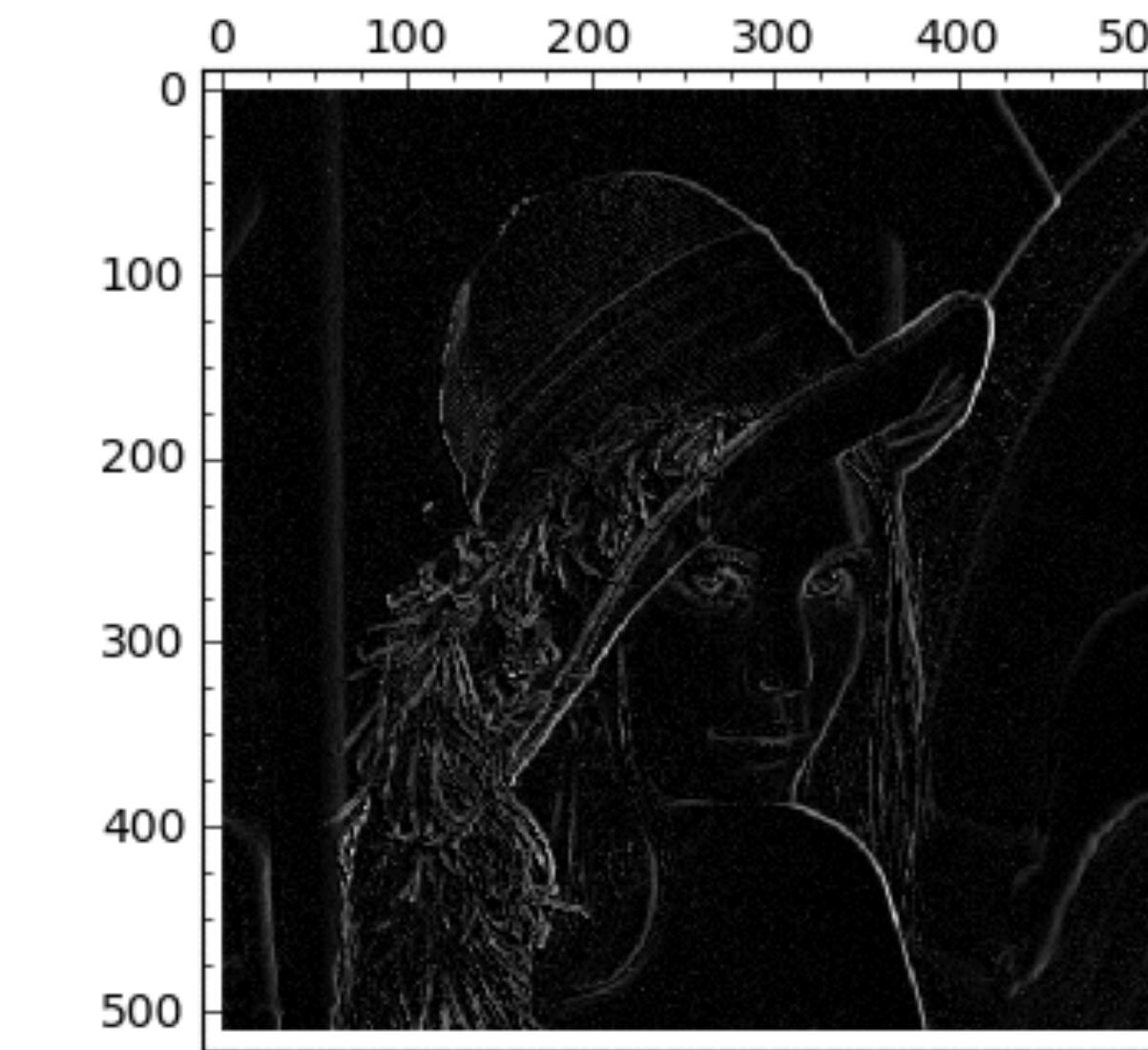
Filtering: a signal processing technique



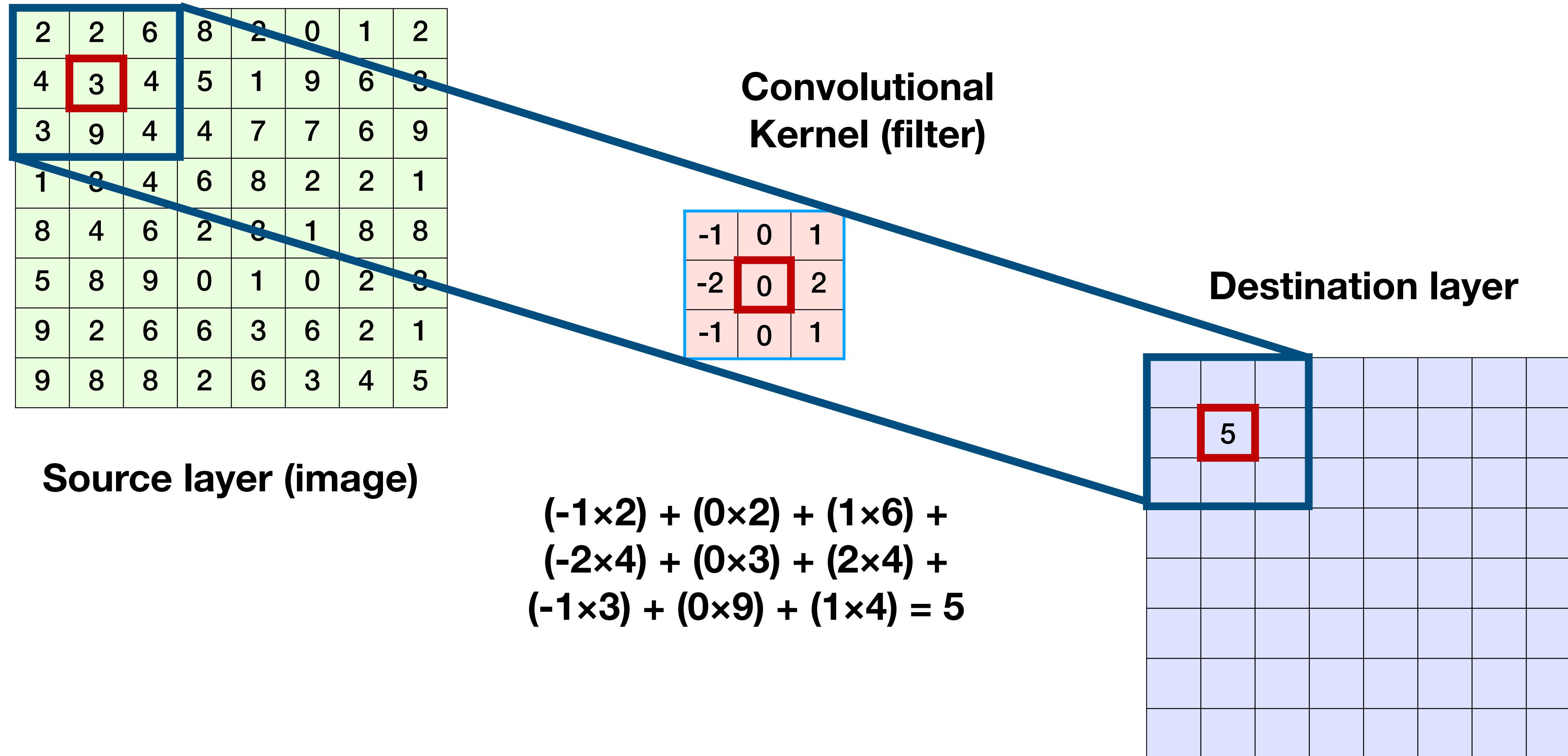
Convolution



Convolution



Convolution



Convolution

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

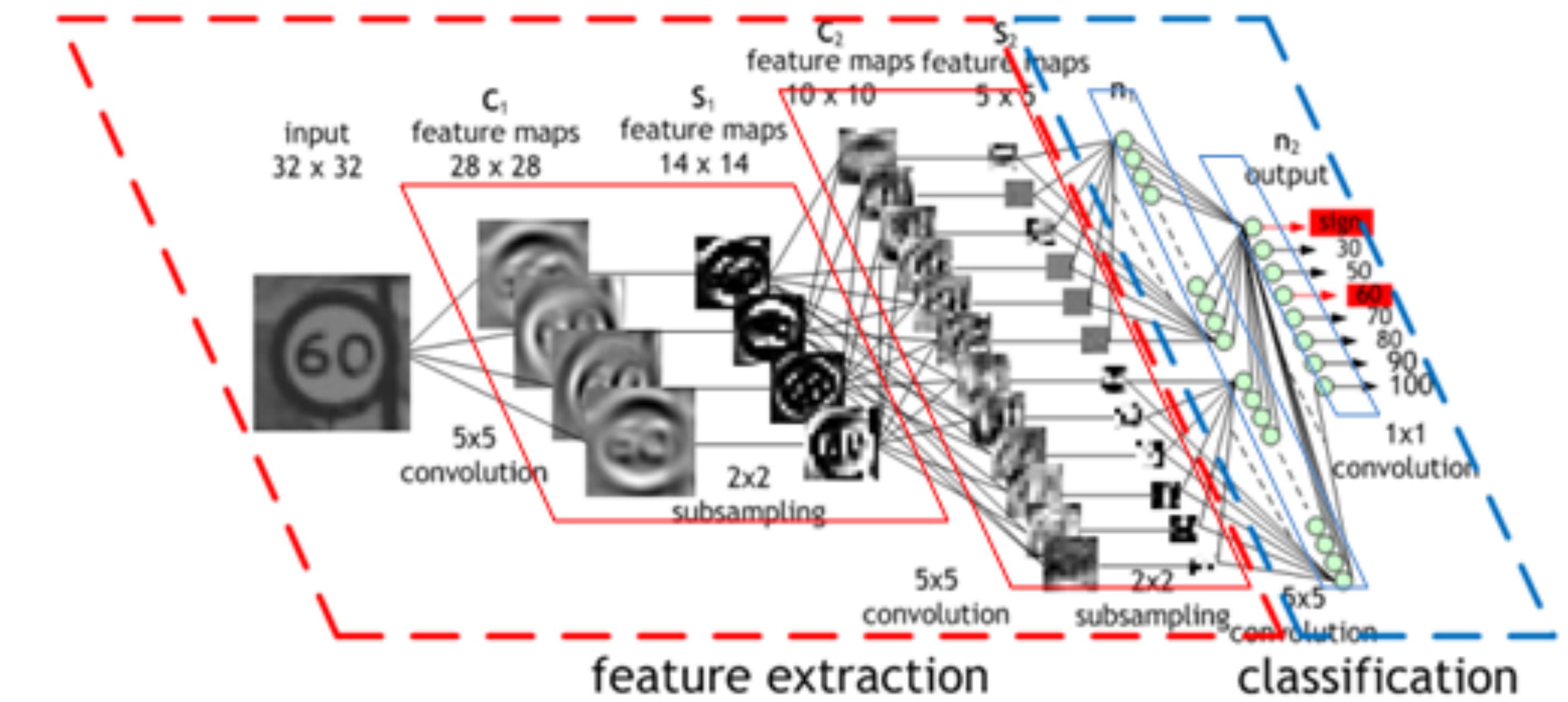
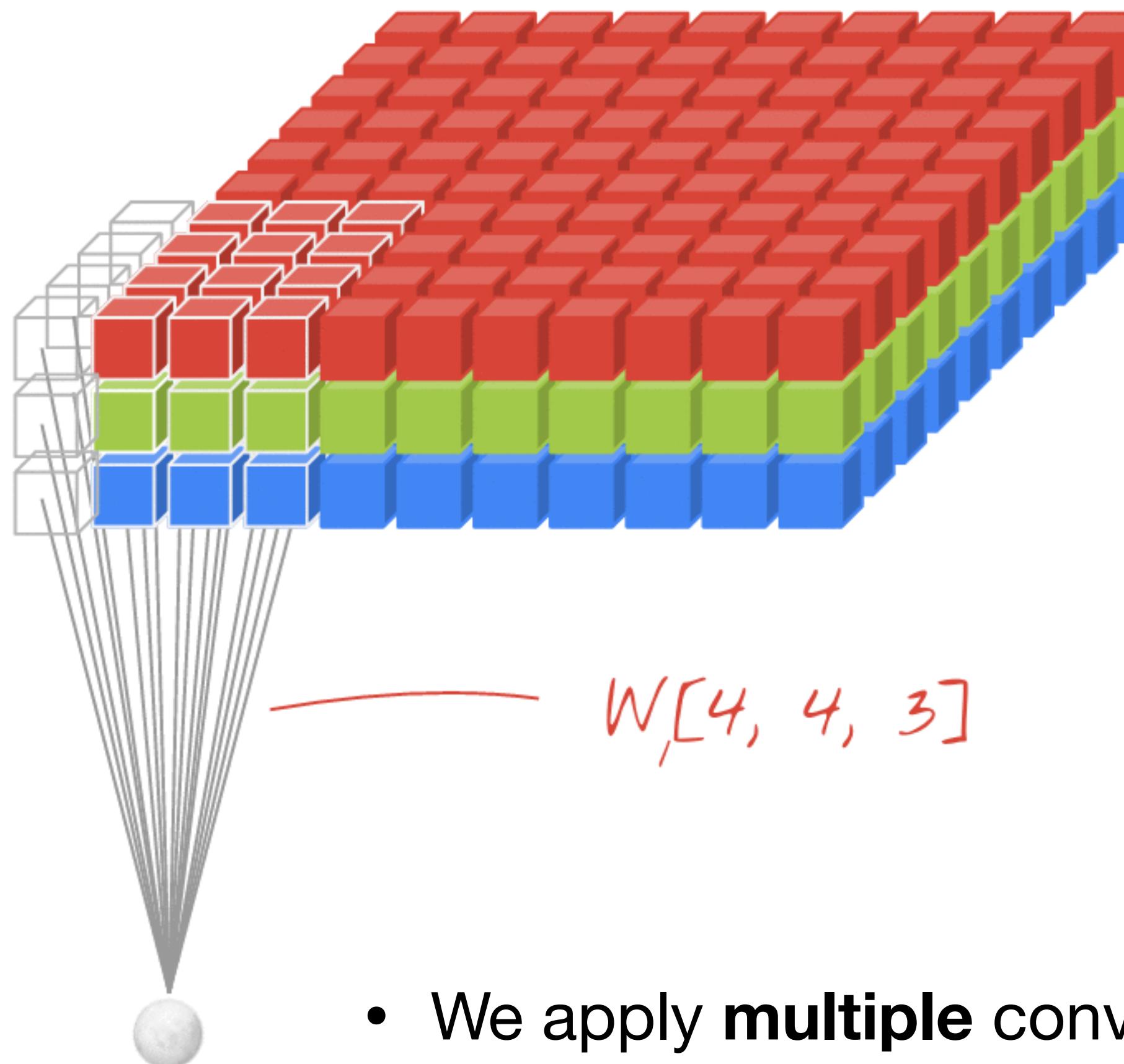
- The kernel is **shifted** over the image with a step size, and computes the output for each position.
- The step size is called **stride**.
- Output has **smaller** dimension than input:
 $\text{dim}(\text{output}) = \text{dim}(\text{input}) - (\text{dim}(\text{kernel}) - 1)$
- **Padding** is used to solve this problem, which artificially make the image “bigger” by adding synthesis data (typically 0-padding)

0	0	0	0	0	0	0
0	2	2	6	8	2	0
0	4	3	4	5	1	0
0	3	9	4	4	7	0
0	1	3	4	6	8	0
0	8	4	6	2	3	0
0	0	0	0	0	0	0

Original array

Padded array

Multiple convolutional channels



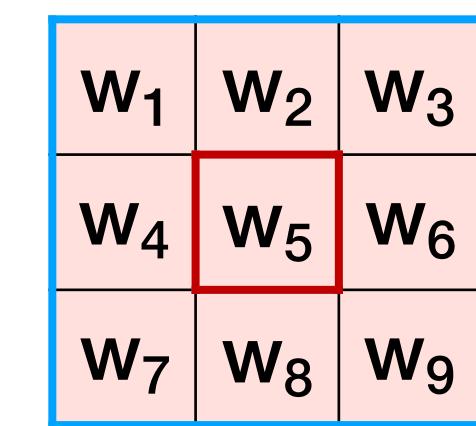
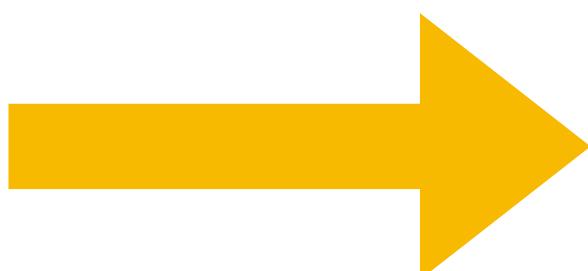
- We apply **multiple convolutional filters/kernels** to the **same** image.
- Each filter results in one convolutional **channel**.
- By learning the same image from different channels, one can detect complex patterns.

Automatic Kernel Determination

Source layer (image)

2	2	6	8	2	0	1	2
4	3	4	5	1	9	6	3
3	9	4	4	7	7	6	9
1	3	4	6	8	2	2	1
8	4	6	2	3	1	8	8
5	8	9	0	1	0	2	3
9	2	6	6	3	6	2	1
9	8	8	2	6	3	4	5

Feature map (activation map)



Kernel (filter)

How do we know which kernels to use?

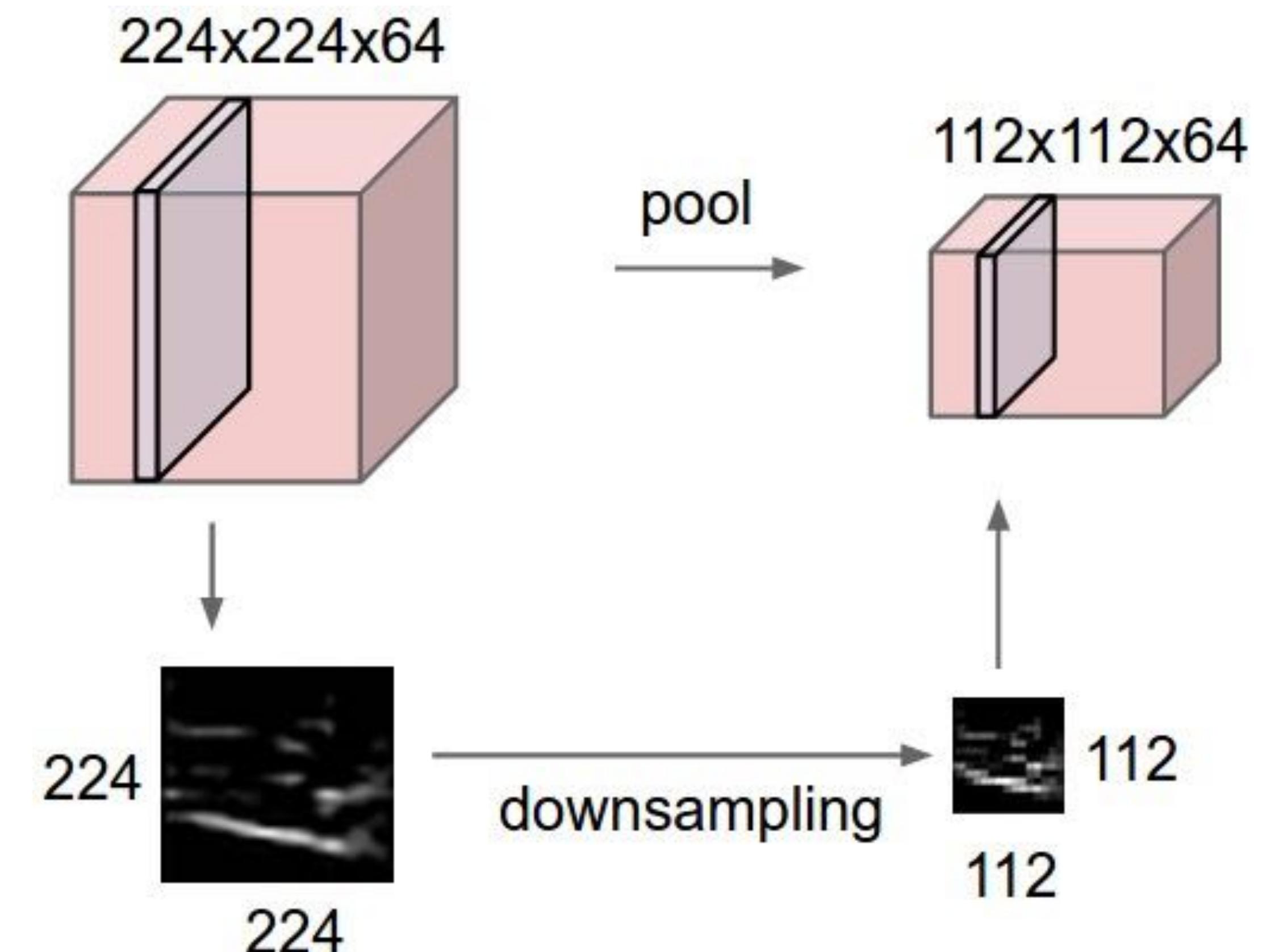
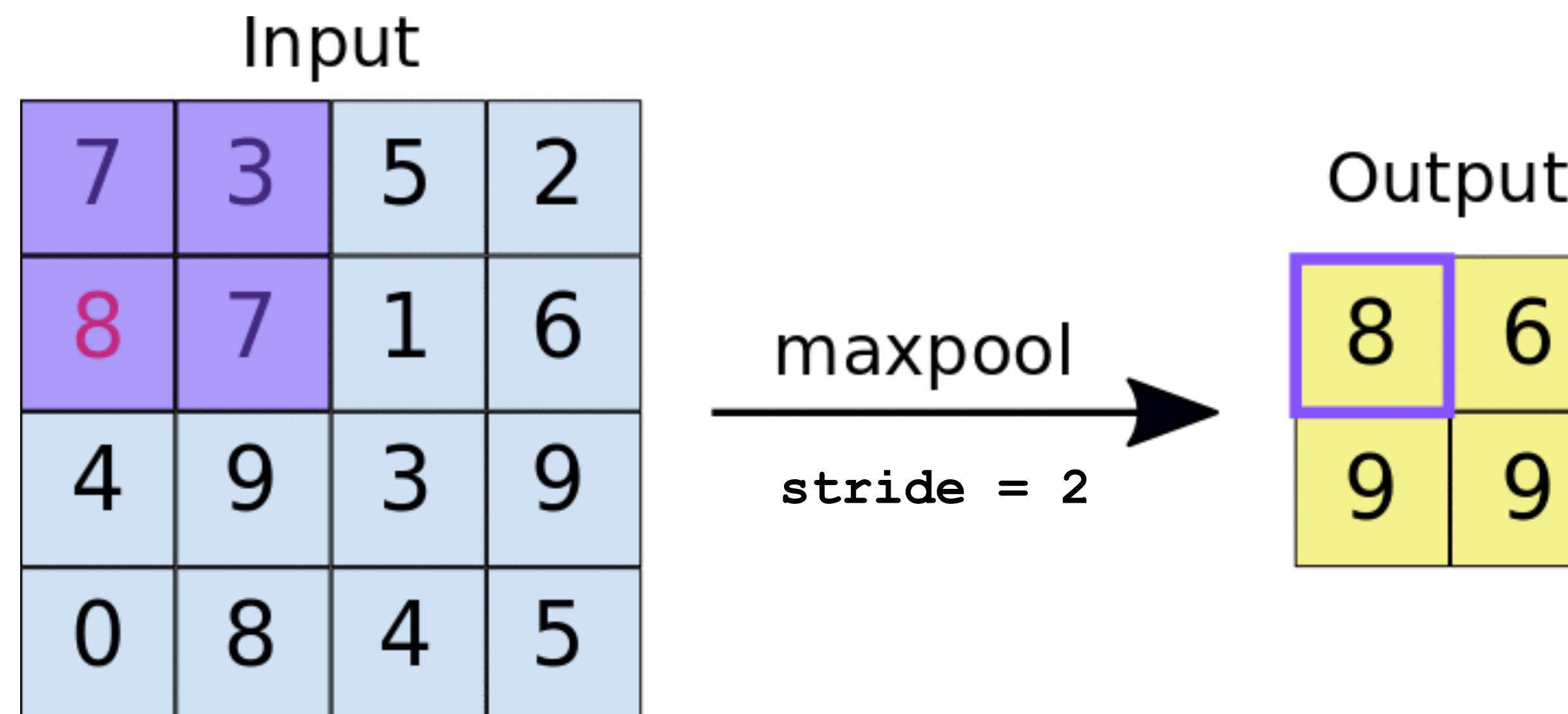
- In **traditional** signal processing, a filter is designed by human experts.
- In **deep learning**, data are too complex and too many different kernels are needed.
- Therefore, kernels are no longer fixed. They are initialized **randomly**.
- Kernels are updated through **backward propagation**, in the way that it **learns** which features to detect.
- **Gradient descent** algorithms are used.

Pooling (downsampling)

We need to **discard** information **gradually**.

Pooling is a way of information **abstraction**.

Pooling is usually applied **after** convolutions.



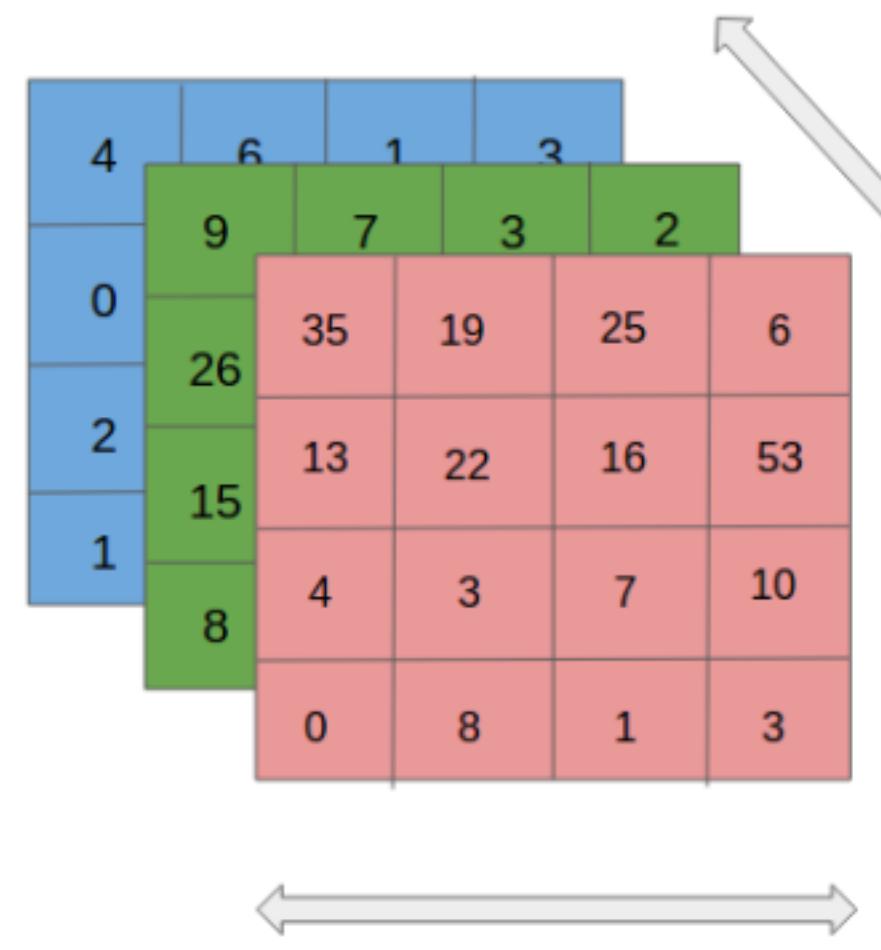
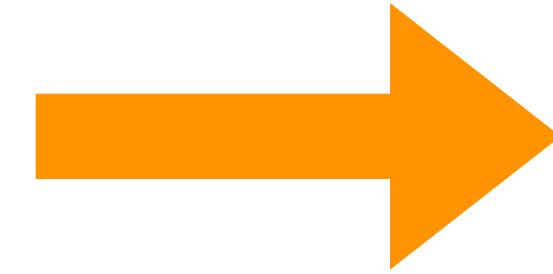
Max pooling: keep the **strongest** signal

Average pooling: use the **local average** as the signal

CNN Architecture



3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0
3.0	2.0	3.0



Input Volume
32x32x1

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1



ReLU Activation Fn.
Volume-28x28x3

Convolution
layer Stride 1

Output Volume
14x14x3

Max Pool
layer Stride 2

Flatten layer

Output Volume
588x1

Output Volume
20x1

Fully connected
Layer ReLU Activation
Fn.

Output Nodes
5x1

Class 1

Class 2

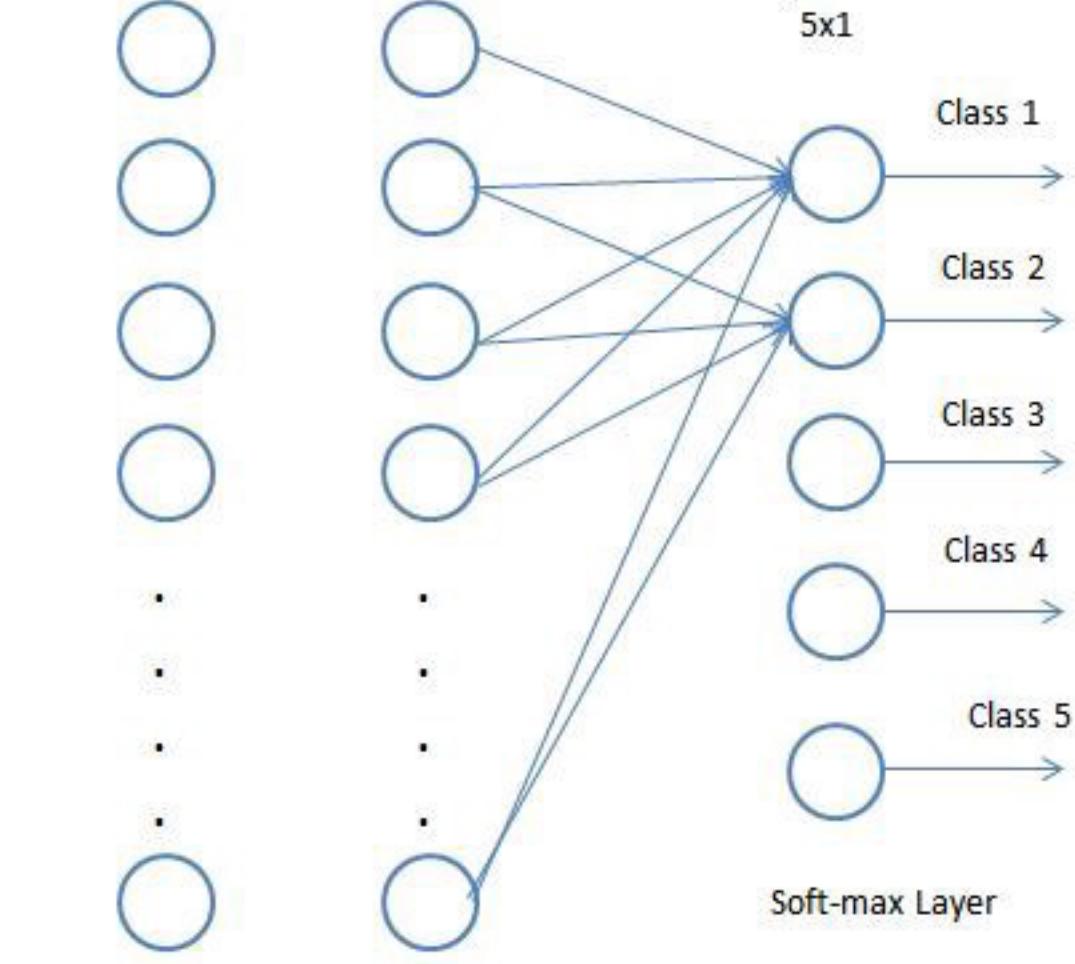
Class 3

Class 4

Class 5

Soft-max Layer

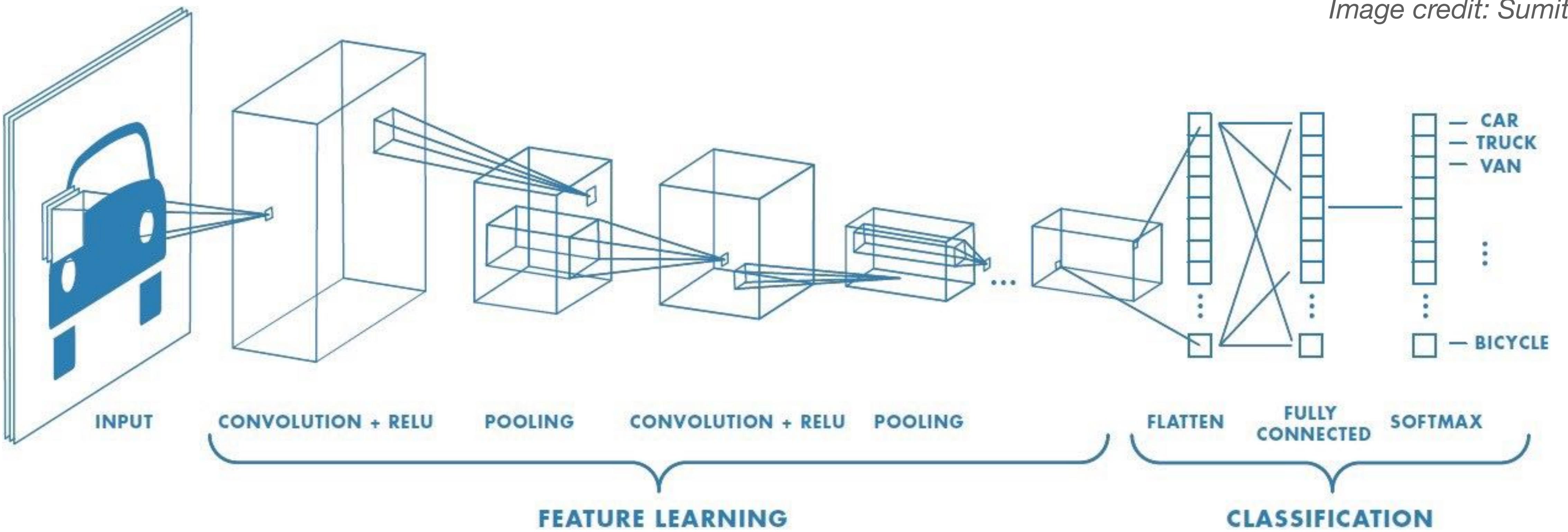
Soft-Max
Activation Fn.



Image

Convolved Feature

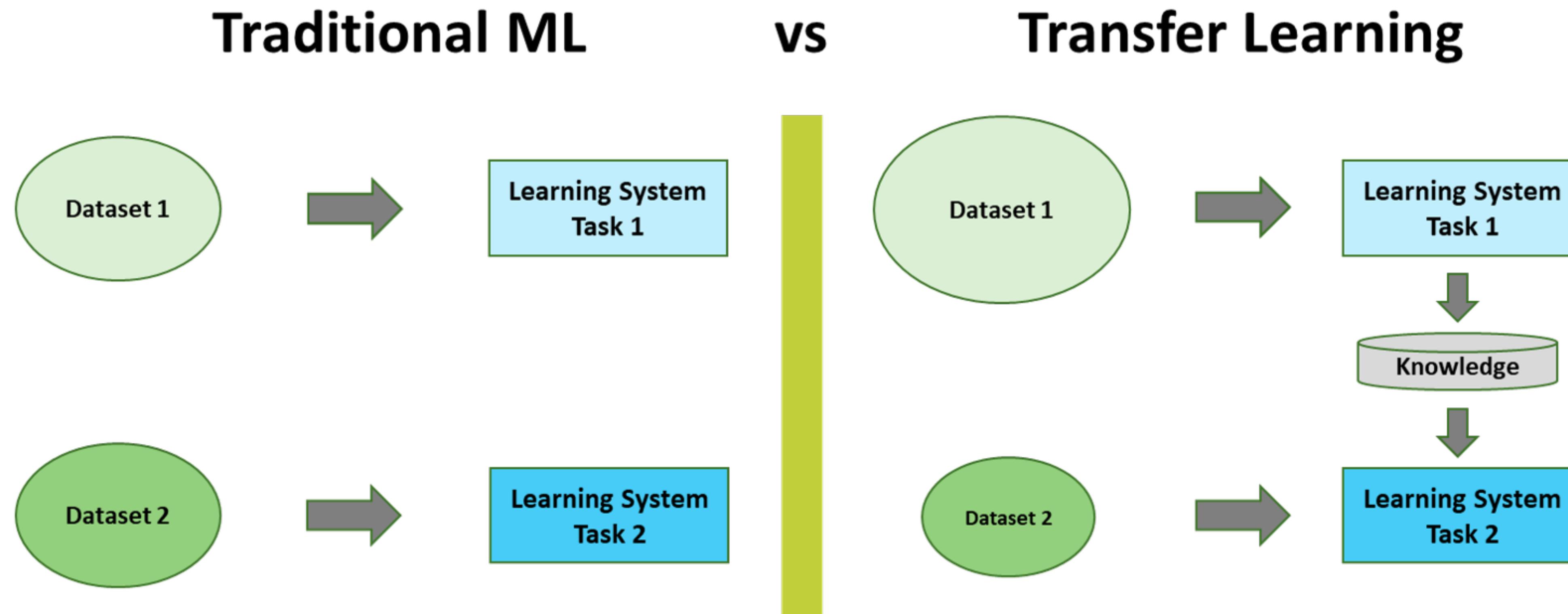
Convolution Neural Network (CNN)



- Typically, **images** become increasingly **smaller** as they go to the **deeper** layers;
- Typically, the number of **filters increases** in the **deeper** layers;
 - Low-level features are limited, thus requires few filters;
 - High-level features are rich, thus requires many filters;
- The **decision making** (i.e., classification) is made in the **last layers**, typically with **dense** layers and the **softmax** activation function.

Transfer Learning

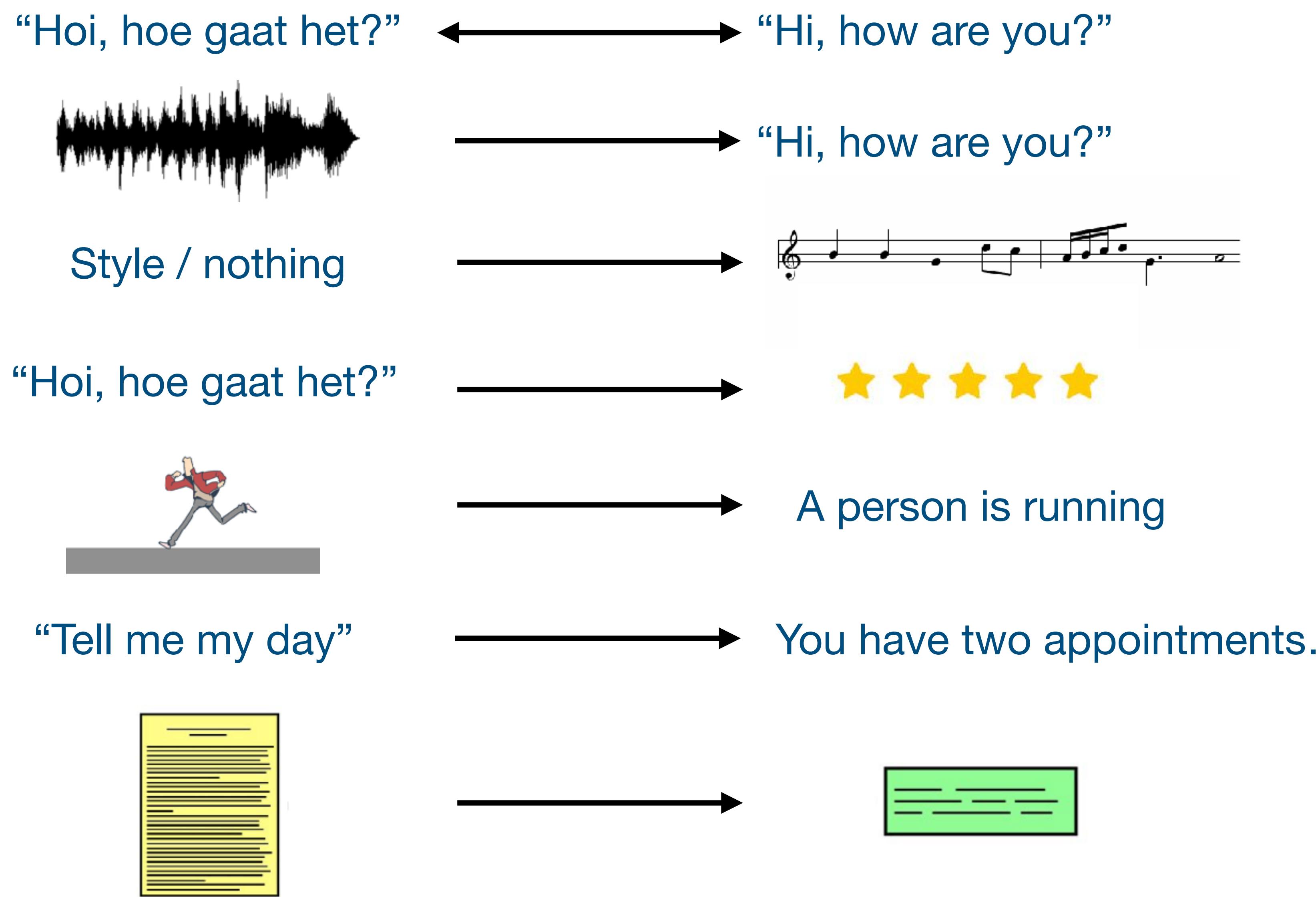
Recall that the convolutional kernels are **learned** during the training process through backward propagation. The trained kernels contains **knowledge** to detect patterns. Why not let a new CNN to **inherit these knowledge**?



How to model sequences?

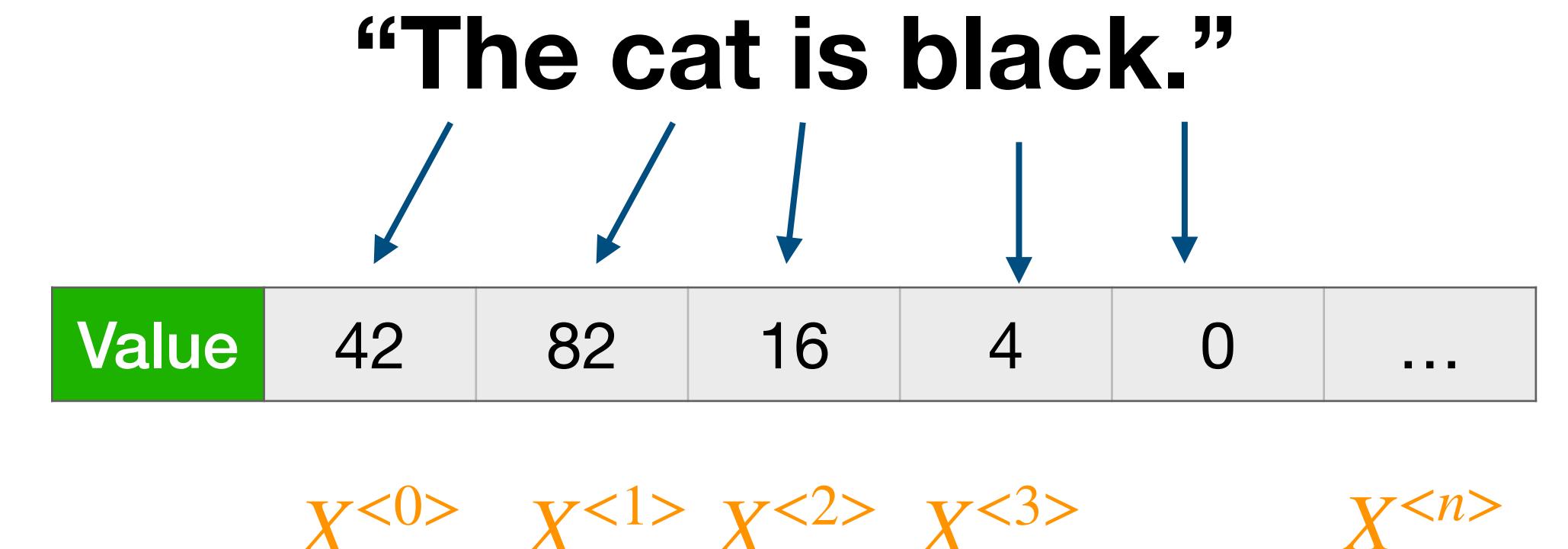
Sequence Modeling

- Language translation
- Speech recognition
- Music generation
- Sentiment classification
- Video captioning
- Chatbot
- Text summary
- ...



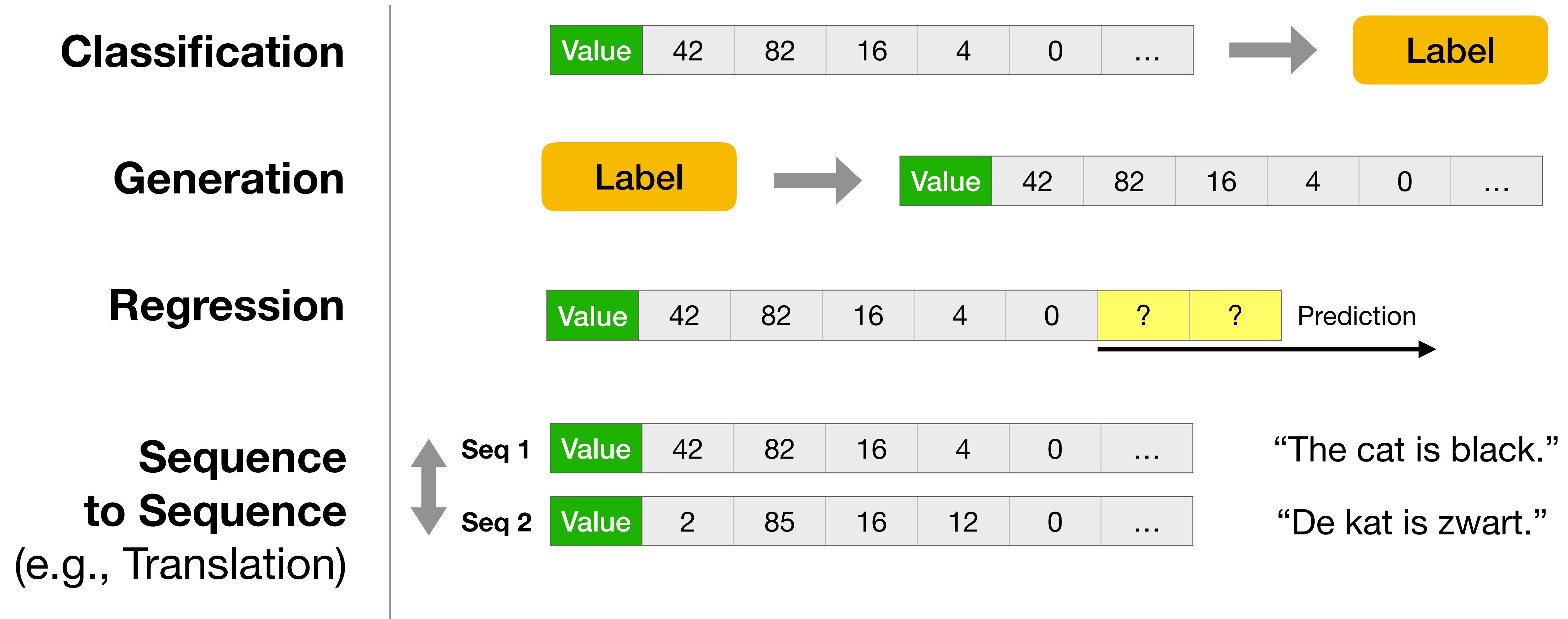
Sequential Data

Sequential data are represented with **1D or 2D arrays**.



- In **natural language processing** (NLP),
- Each **word** is converted into a **token**;
 - A token is typically represented as a **numerical** value;
 - A **sentence** is a **sequence** of tokens;
 - A sequence can have **arbitrary length**;
 - Tokens may be **logically related** to each other.

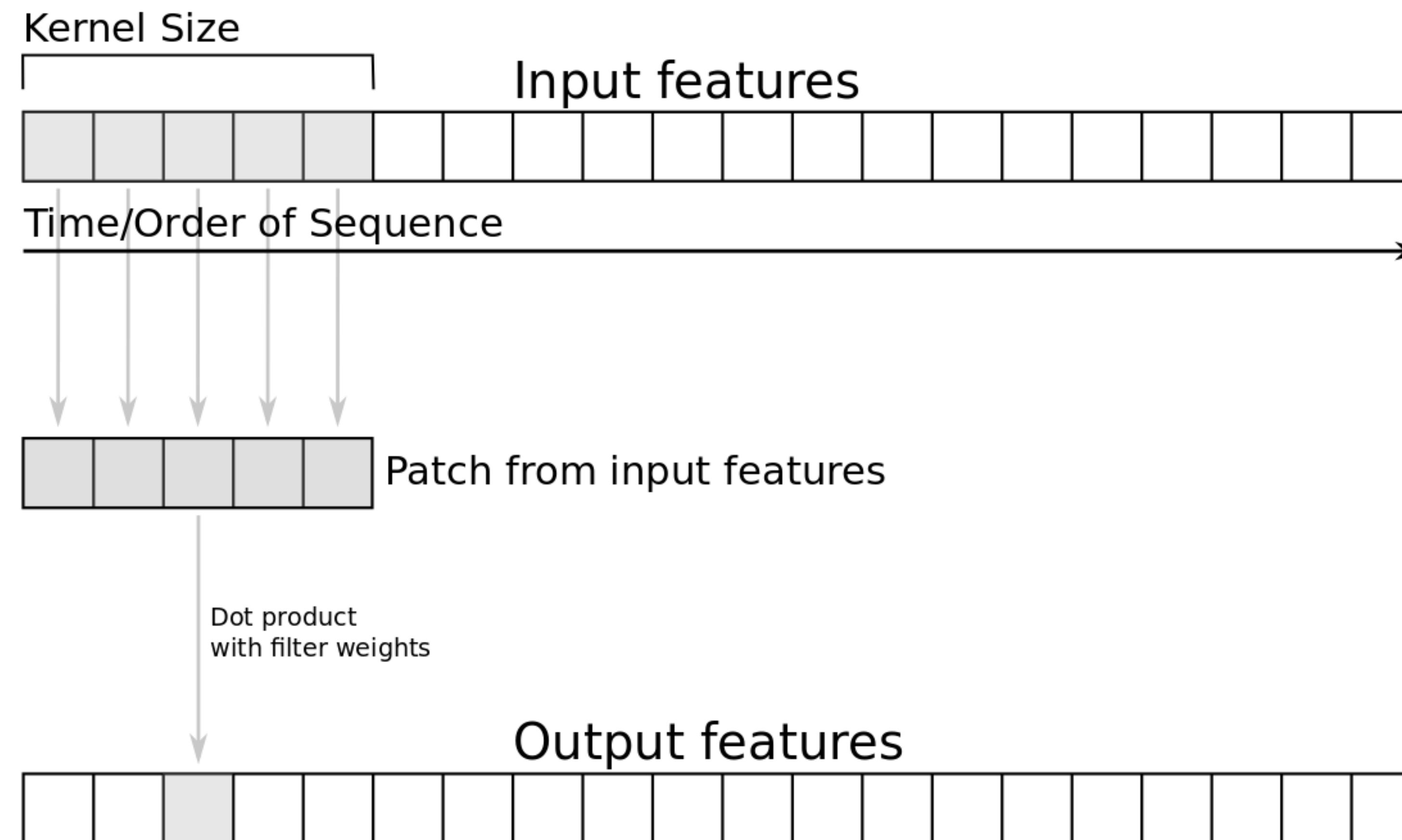
Sequential Data Modeling



The fundamental problem in sequence model is to understand the **relations between tokens/elements**.

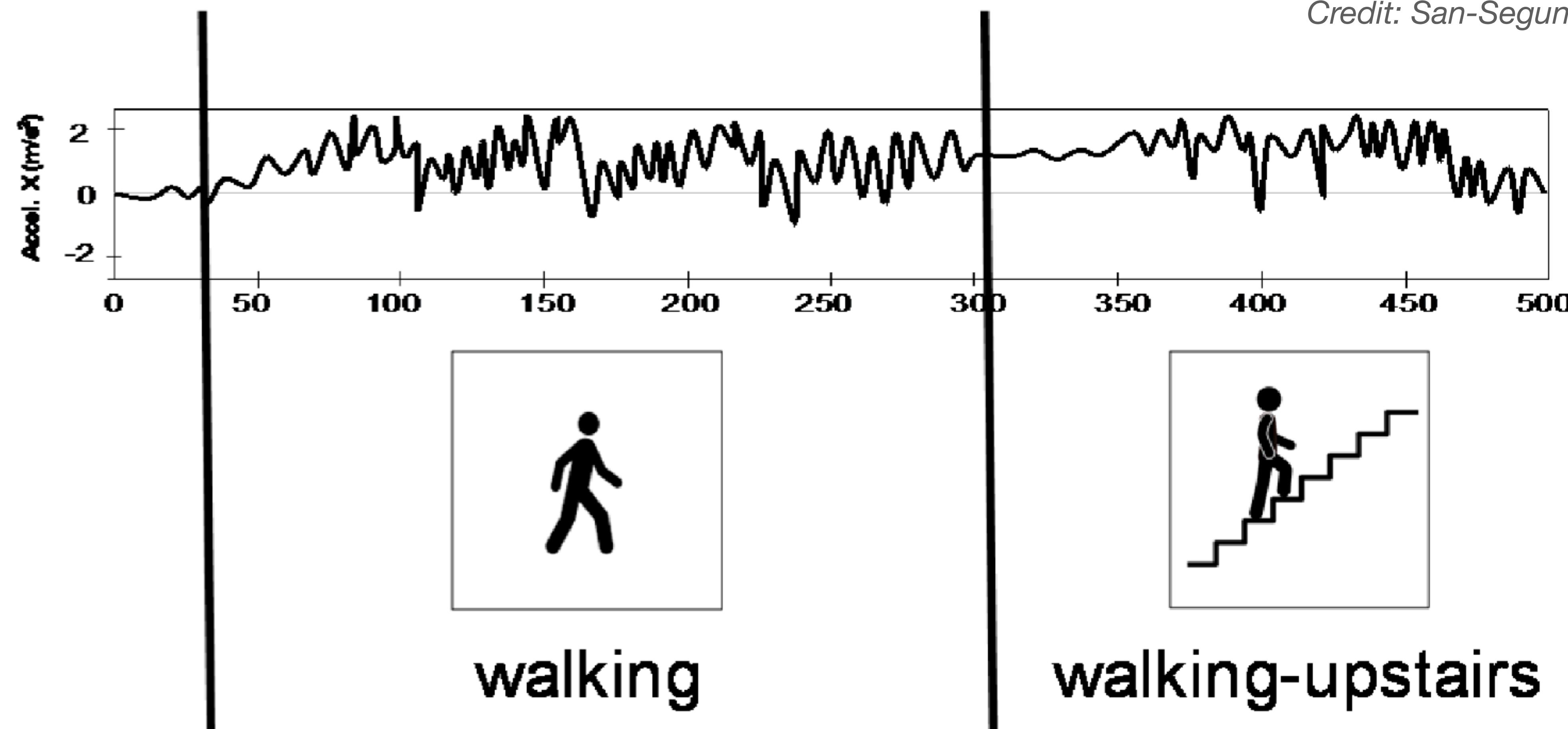
Sequence modeling with 1D convolution

CNNs works because they can take into account the relation between **nearby** tokens.



Example: human activity recognition

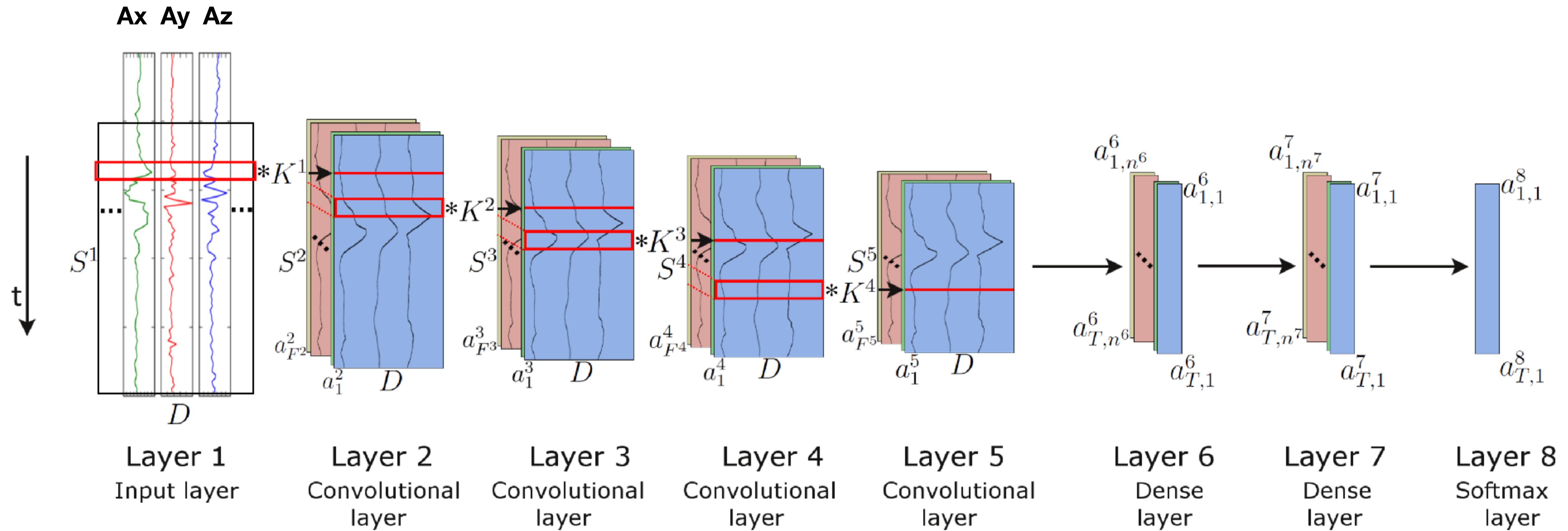
Credit: San-Segundo et al. (2016)



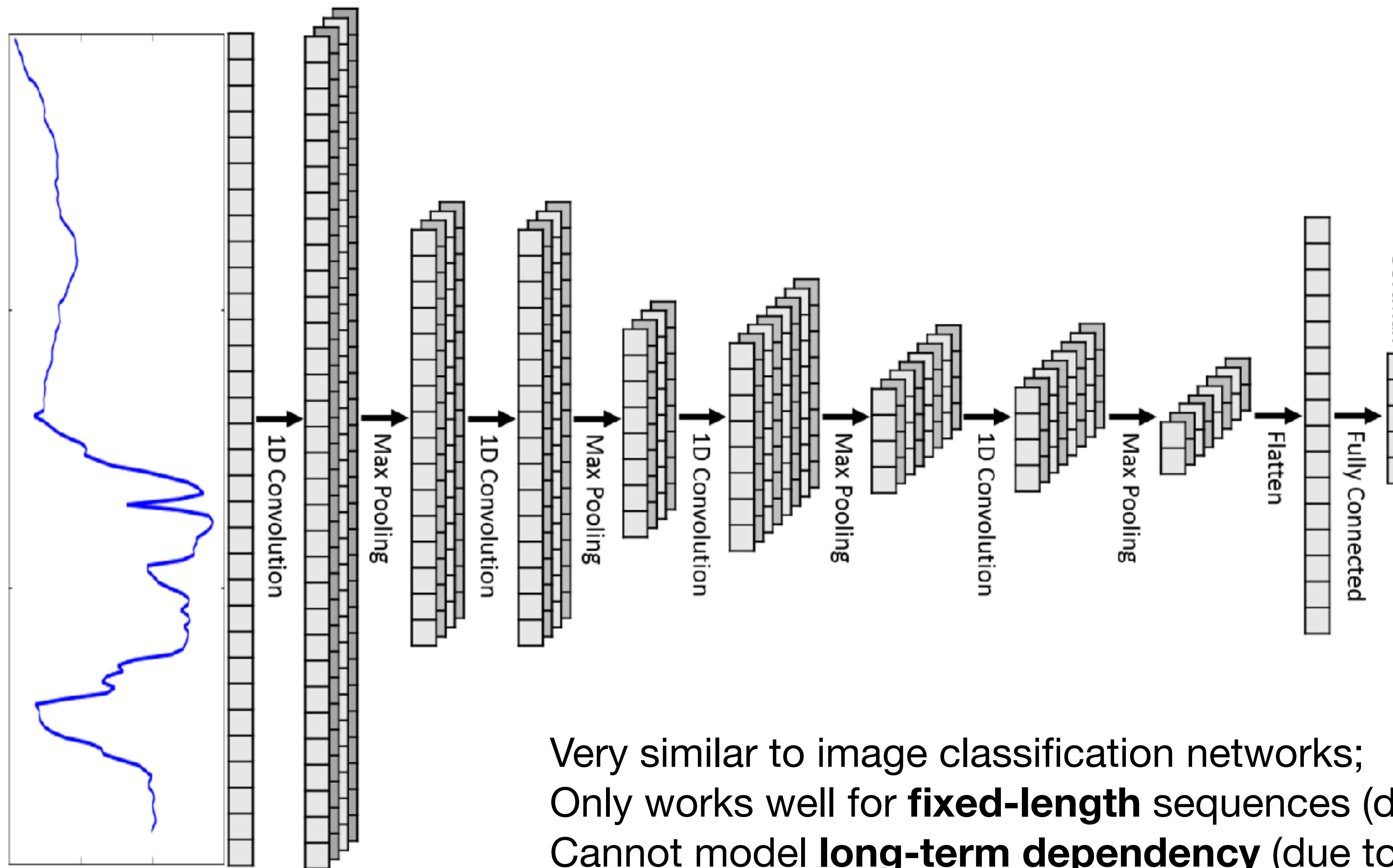
The DNN can be trained with **existing** open datasets!

e.g., <https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones>

Human activity recognition with 1D Convolution



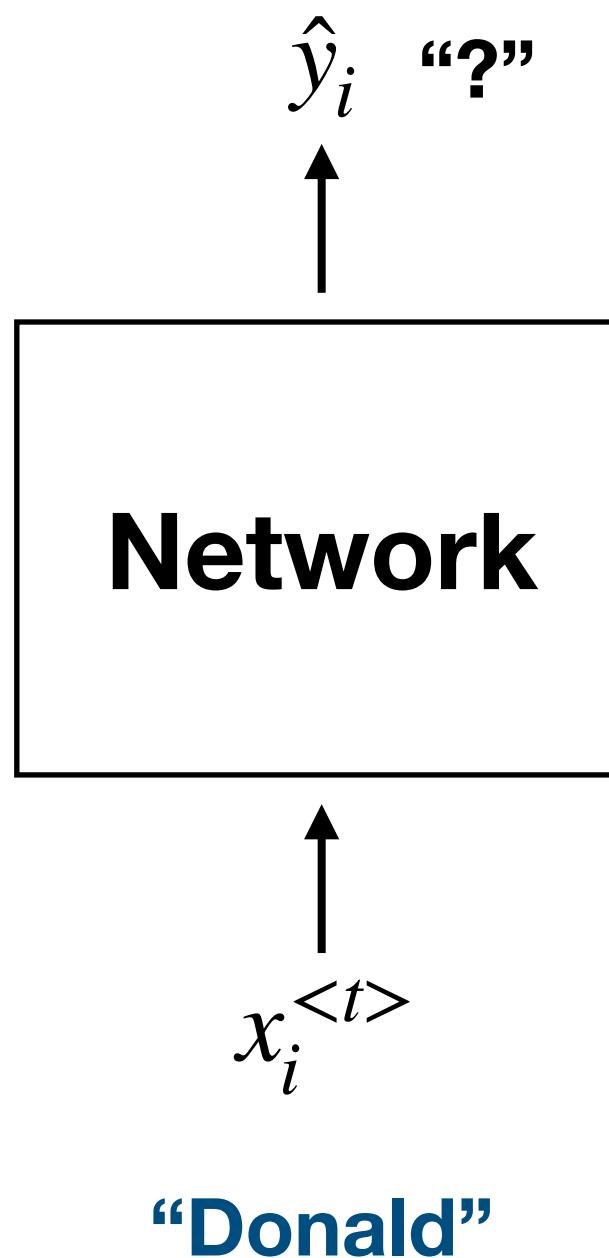
Sequence modeling with 1D convolution



Very similar to image classification networks;
Only works well for **fixed-length** sequences (due to the FC layer);
Cannot model **long-term dependency** (due to the limited kernel size).

Naive Approaches v2

How about modeling just the next word?



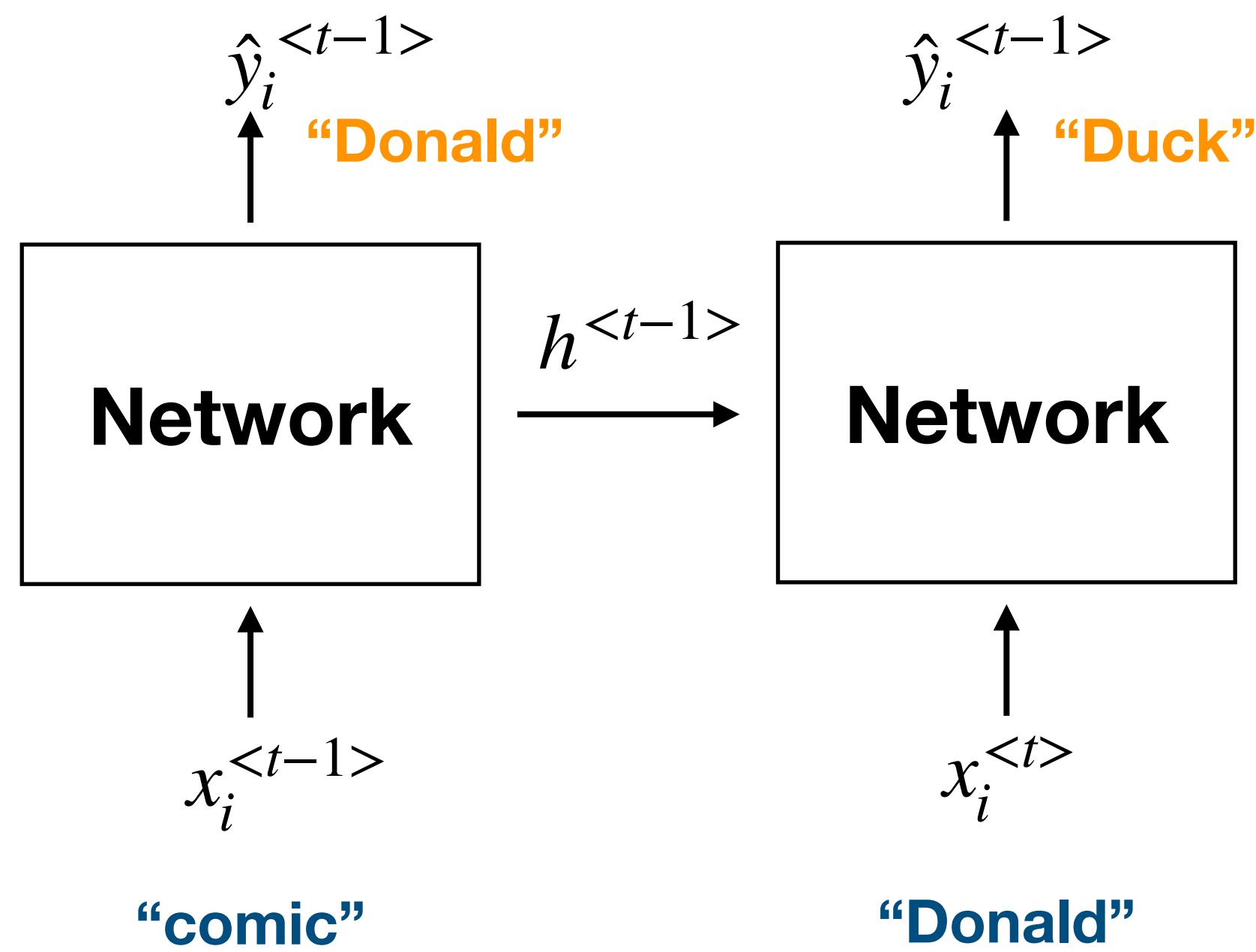
*The president of the United States is Donald Trump.
Wherever Donald Trump goes, security is tight.*

The comic Donald Duck is very famous.

Context matters to the prediction!

Context Propagation

The meaning of the sentence depends on the **context**.



The context at $x_i^{<t-1>}$ is propagated to the next time step via the **hidden state** $h_i^{<t-1>} = h^{<t-1>}$.

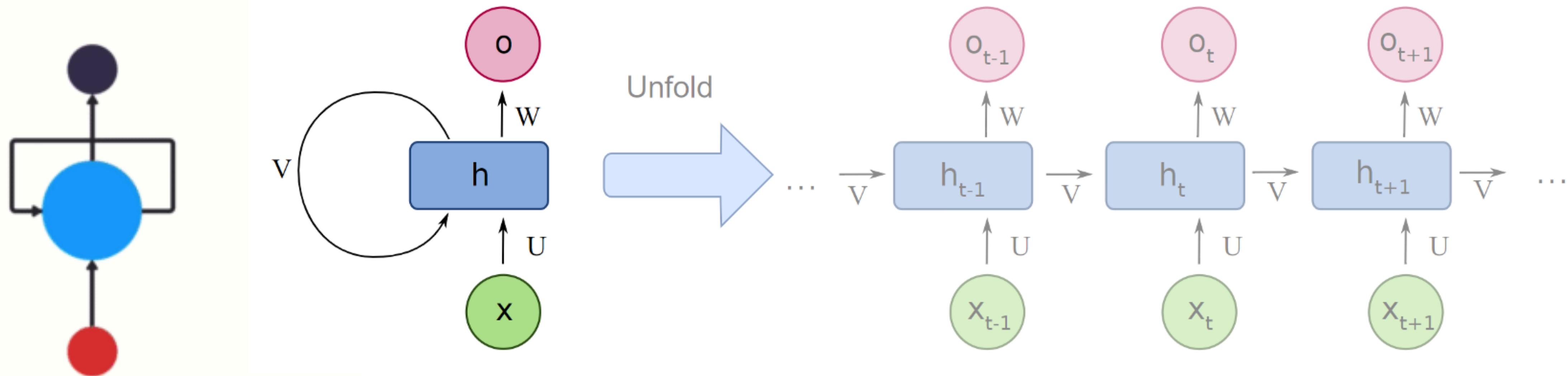
The next prediction is then modeled with both $x_i^{<t>}$ and $h_i^{<t-1>} = h^{<t-1>}$.

The introduction of the hidden state h_i allows the network to have **memory**.

"The comic Donald Duck is very famous."

Recurrent Neural Networks

In recurrent neural networks (RNNs), weights are **reused** across multiple time steps.



RNN

$$h_t = \sigma(Ux_t + Vh_{t-1} + b_h)$$

$$O_t = \sigma(Wh_t + b_o)$$

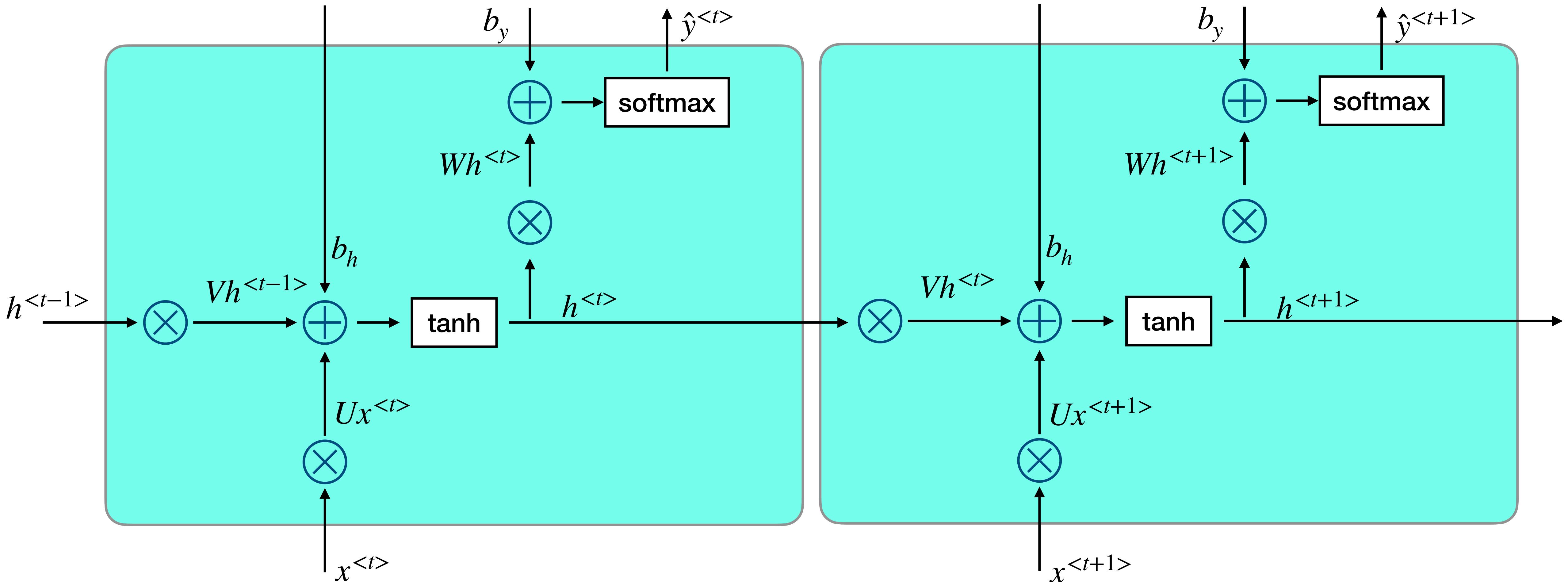
Back propagation through **time**

Non-RNN

$$O_t = \sigma(Wx_t + b)$$

Back propagation through **layers**

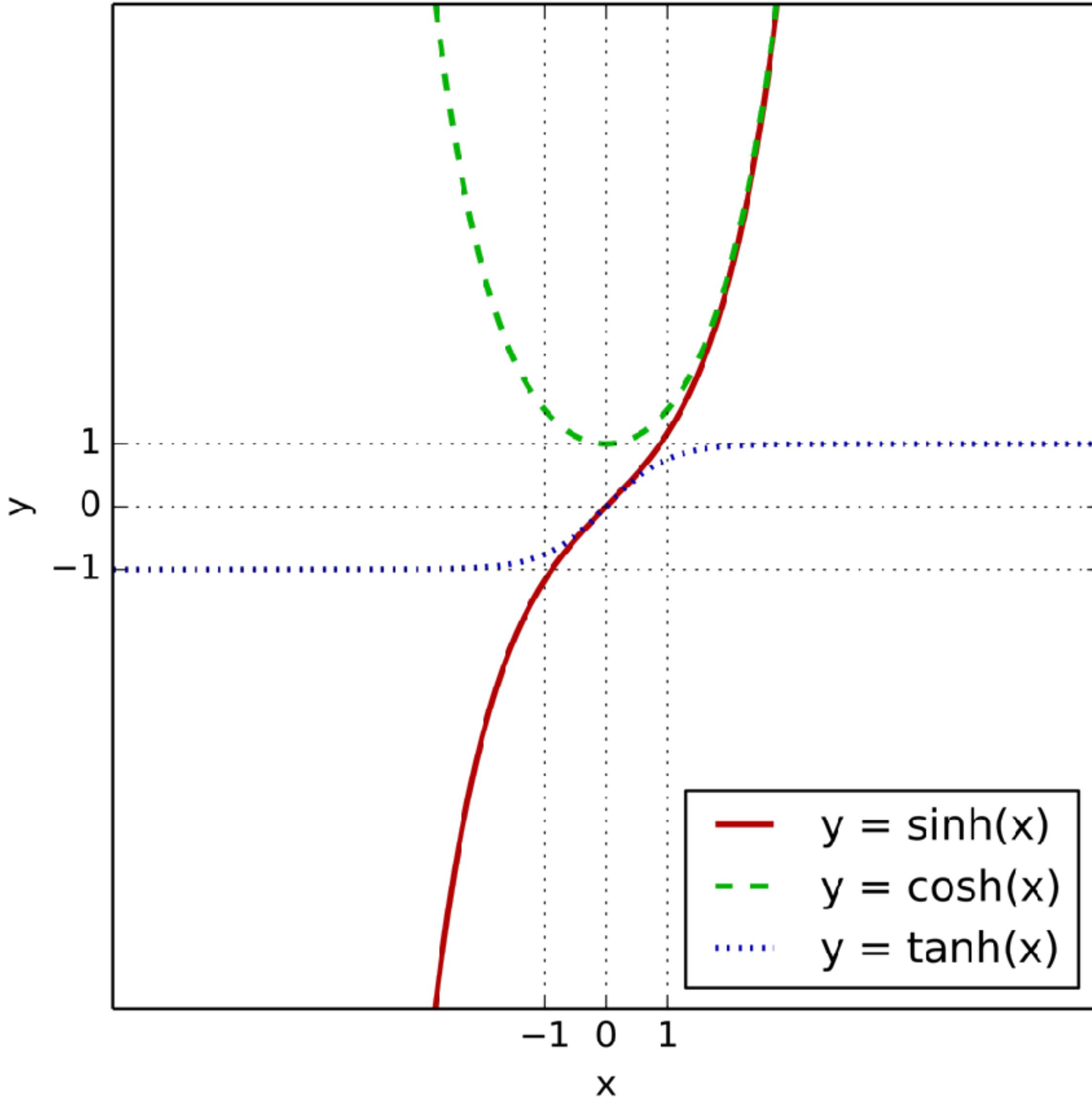
Anatomy of an RNN Cell



$$h^{<t>} = \tanh(Ux^{<t>} + Vh^{<t-1>} + b_h)$$

$$\hat{y}^{<t>} = \text{softmax}(Wh^{<t>} + b_y)$$

Hyperbolic Tangent as an Activation Function



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Between $[-1, 1]$, the function behaves **linearly** (more or less).

As long as the input range is within $[-1, 1]$, the derivative in this range is approximately 1.

The function always scales the input to the well-controlled range of $[-1, 1]$.

Limitation of standard RNNs

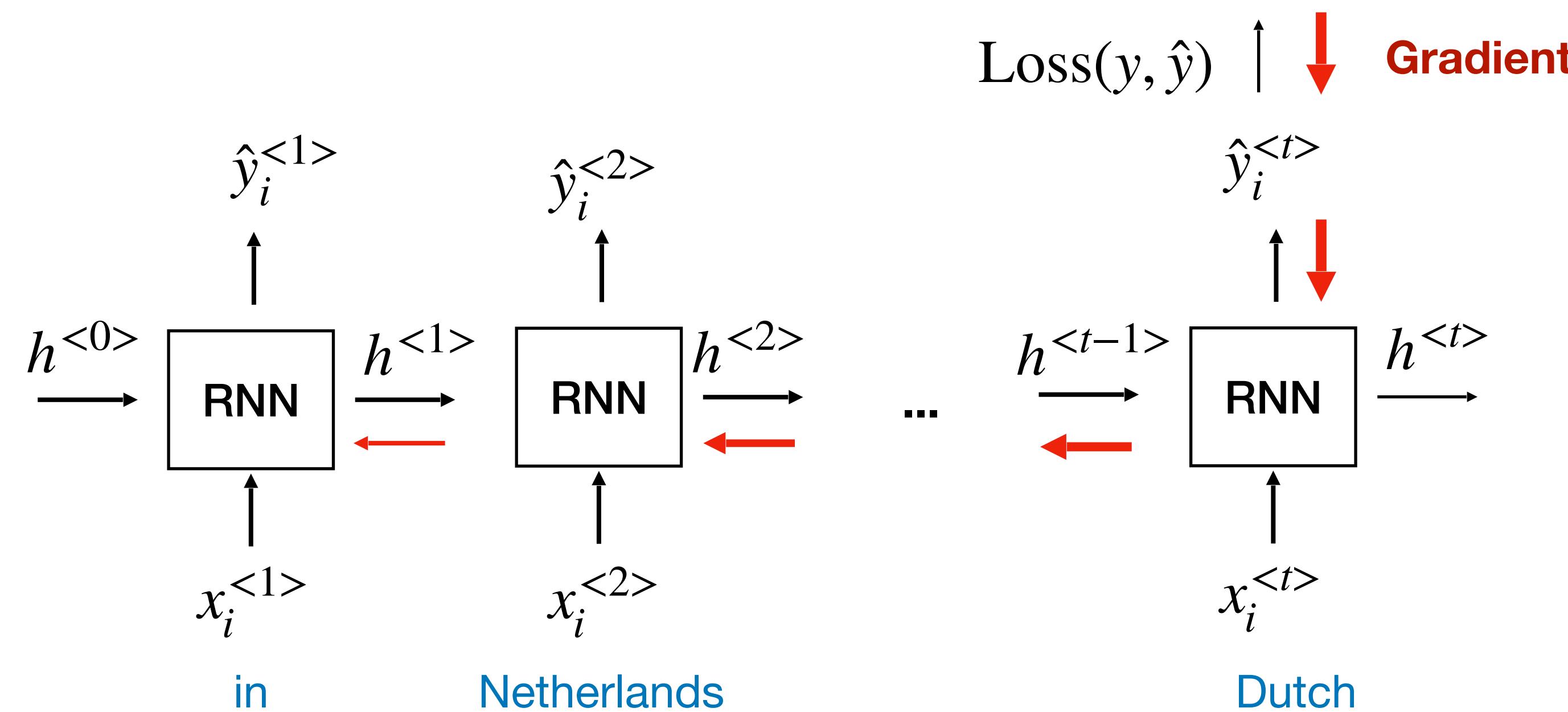
We are not done with the story...

Long-term Dependency

Sometimes, the contexts are far away.

The clouds are in the *sky*.

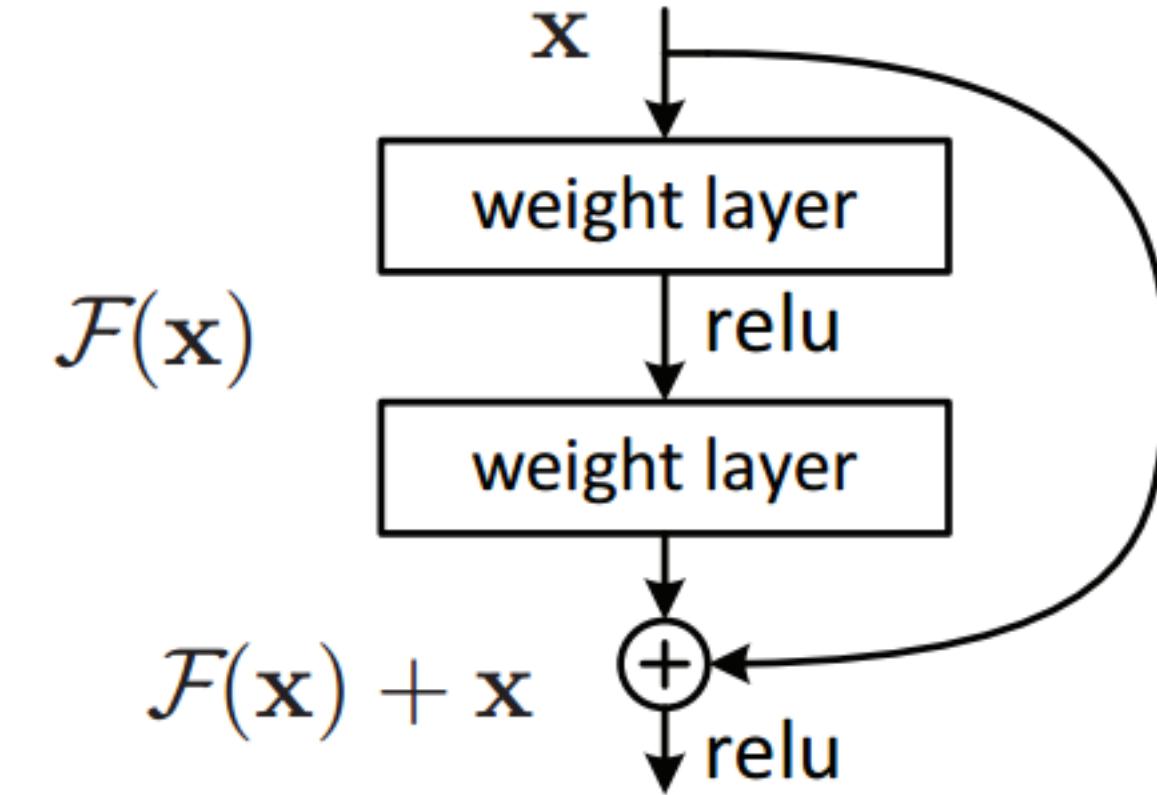
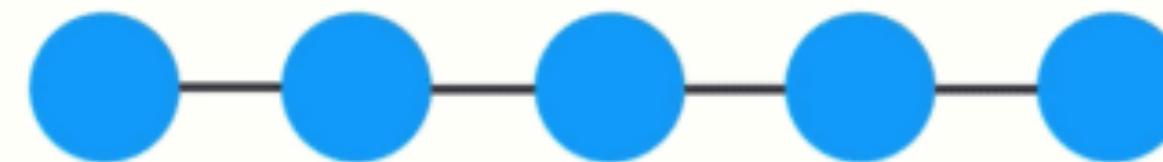
I grew up in the Netherlands... I speak fluent *Dutch*.



Contexts can be forgotten; the gradients are vanishing during back propagation.

Vanishing Gradient Problem in RNNs

Residual connection is a general solution to the vanishing gradient problem.

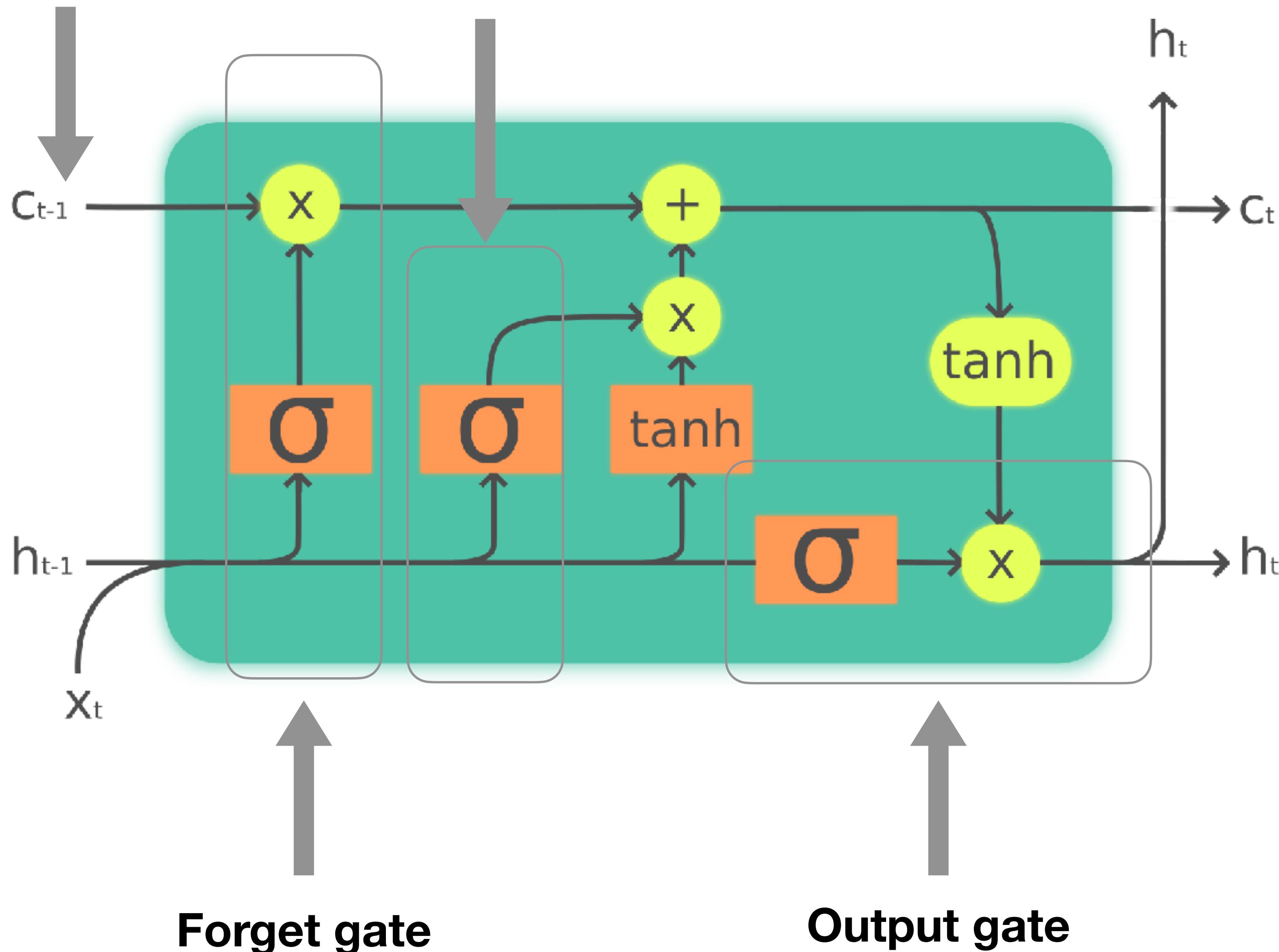


However, for very long dependency (which is common in RNNs), this solution is not sufficient. We should instead regulate which information to **keep** and which to **forget**.

Long Short-term Memory

Cell state

Input gate



Hochreiter & Schmidhuber (1997)

LSTM regulates the flow of information through gates.

- **Input gate**: controls the extent to which a new value flows into the cell;
- **Forget gate**: controls the extent a value remains in the cell;
- **Output gate**: controls the extent to which the value in the cell is used to calculate the output activation.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

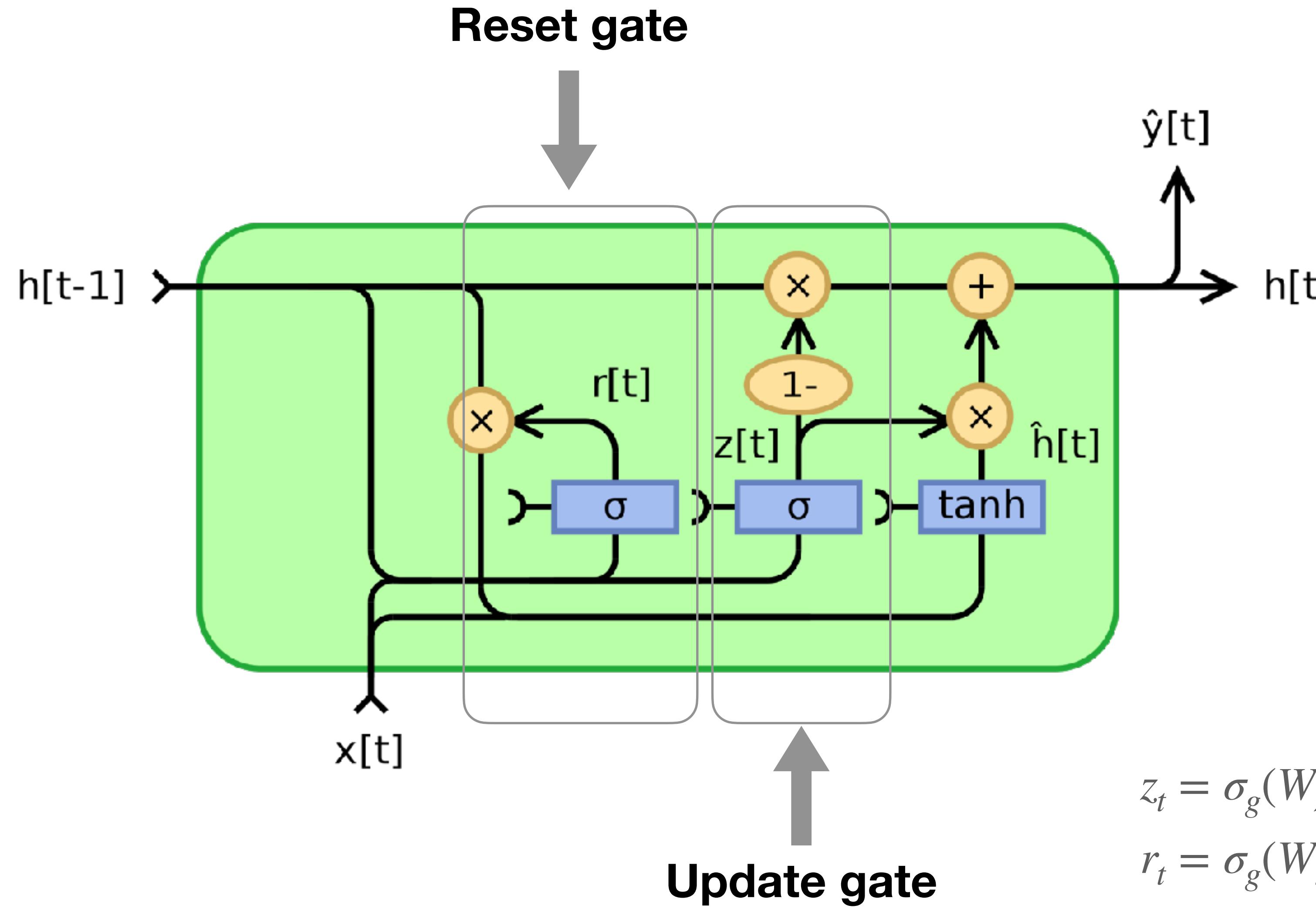
$$\tilde{c}_t = \sigma_h(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \sigma_h(c_t)$$

Image source: Wikipedia

Gated Recurrent Unit



Cho et al. 2014 (arXiv:1406.1078)

GRU regulates the flow of information through gates:

- **Reset gate**: controls how much past information to forget;
- **Update gate**: controls what information to discard and what to keep (similar to the LSTM's input gate and forget gate combined)

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \phi_h(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$

Generating Input Sequences

To apply a long sequence to a RNN, we need to define the training data x and the labels y :

```
(1, ... ) (2, ... ) (3, ... ) (4, ... ) (5, ... ) (6, ... ) (7, ... ) (8, ... ) (9, ... ) (10, ... ) (11, ... ) (12, ... ) (13, ... ) (14, ... )
```

We could either **reshape** it:

x

```
(1, ... ) (2, ... ) (3, ... ) (4, ... ) (5, ... ) (6, ... ) (7, ... )
```

y

```
(8, ... ) (9, ... ) (10, ... ) (11, ... ) (12, ... ) (13, ... ) (14, ... )
```

Or truncate it using a **rolling window**:

```
(1, ... ) (2, ... ) (3, ... ) (4, ... ) (5, ... ) (6, ... ) (7, ... )
```

```
(3, ... ) (4, ... ) (5, ... ) (6, ... ) (7, ... ) (8, ... ) (9, ... )
```

```
(5, ... ) (6, ... ) (7, ... ) (8, ... ) (9, ... ) (10, ... ) (11, ... )
```

```
(13, ... )
```

```
(7, ... ) (8, ... ) (9, ... ) (10, ... ) (11, ... ) (12, ... ) (13, ... )
```

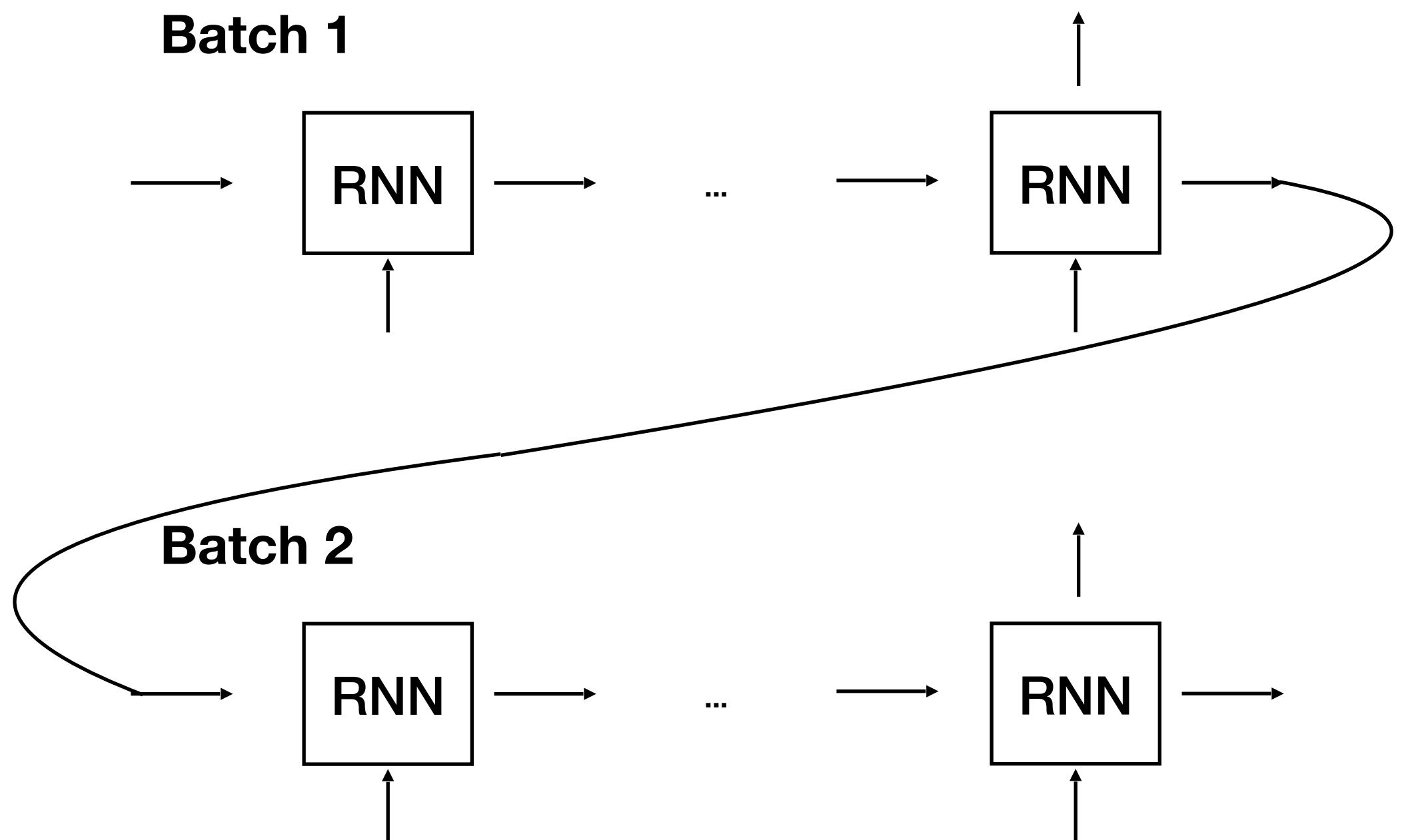
x
 y

Stateful RNNs

A stateful RNN passes the **last state** of the **previous batch** as the **initial state** of the **next batch**.

This is useful if there is some connections between batches. This feature essentially allows long sequences to be **folded** into batches.

A non-stateful RNN initialises state of each batch to zero.



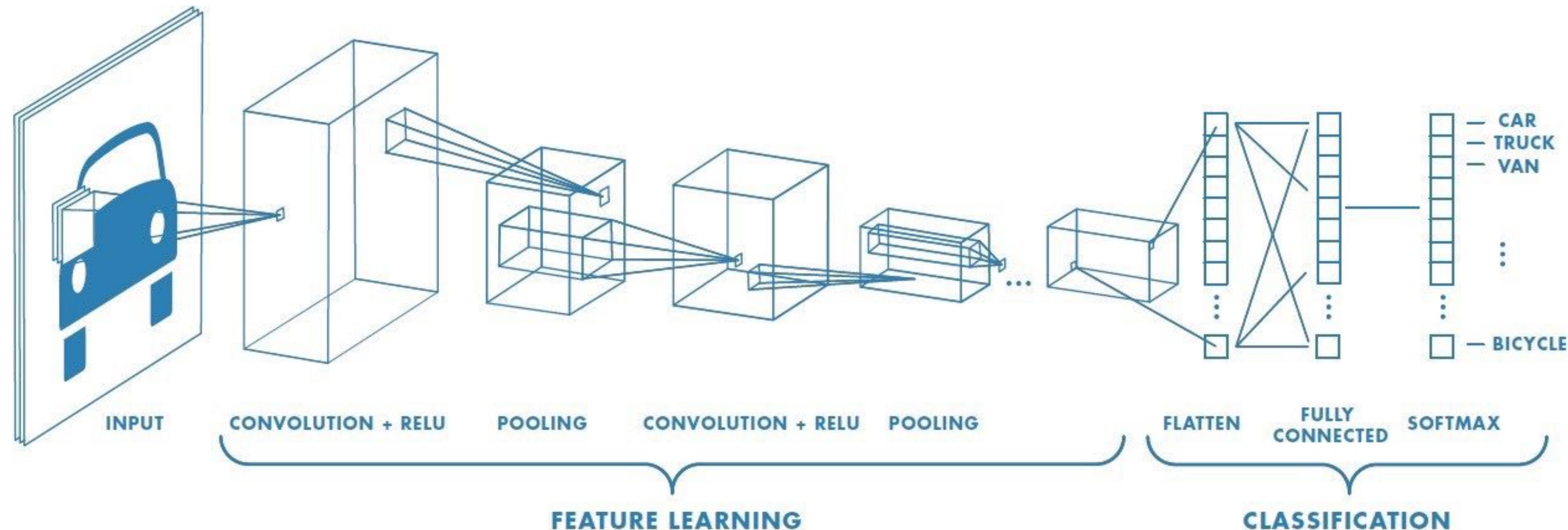
Batch 1 $(1, \dots) (2, \dots) (3, \dots) (4, \dots) (5, \dots) (6, \dots) (7, \dots)$

Batch 2 $(8, \dots) (9, \dots) (10, \dots) (11, \dots) (12, \dots) (13, \dots) (14, \dots)$

Generative Models

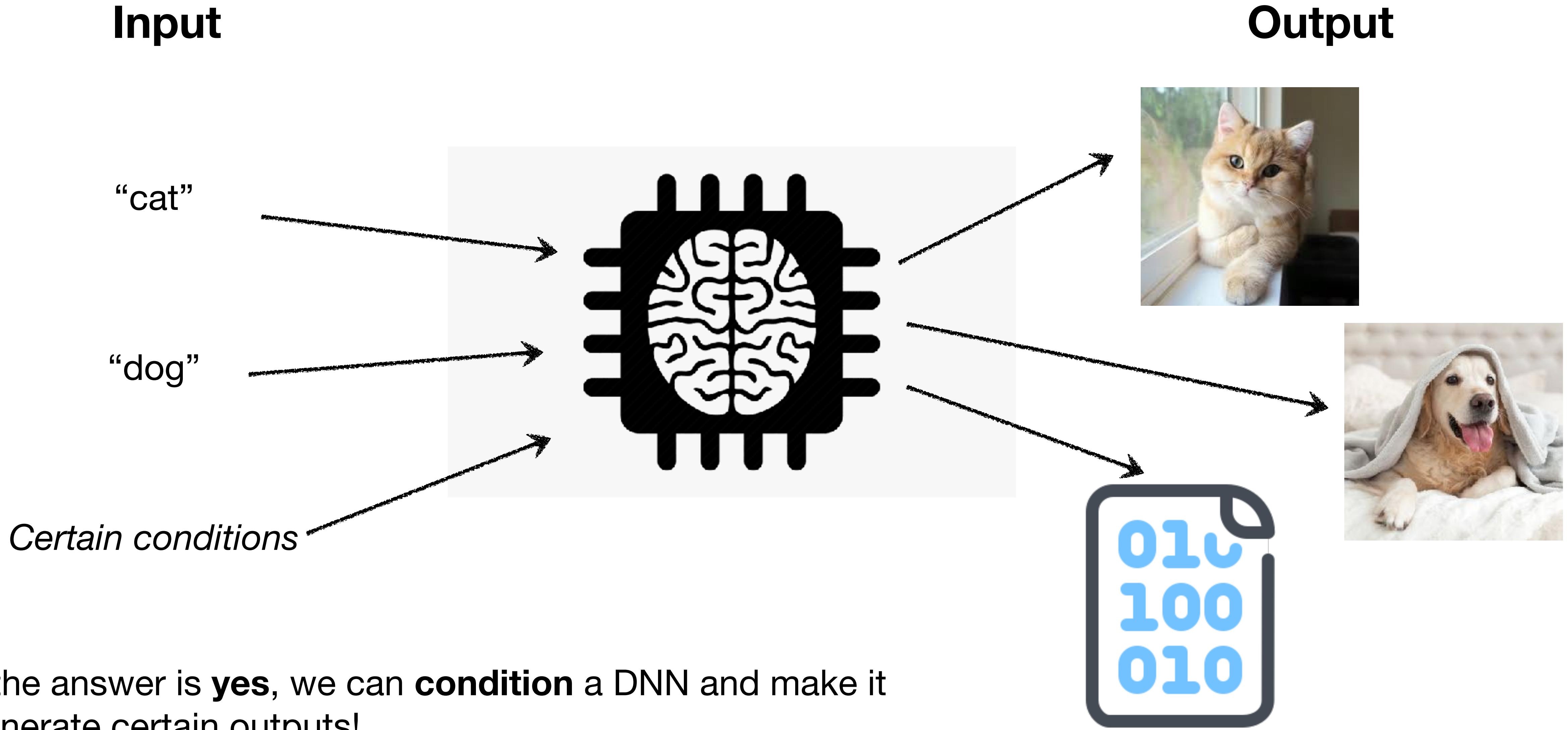
Can we invert a CNN classifier?

CNN image classifier



Generative models?

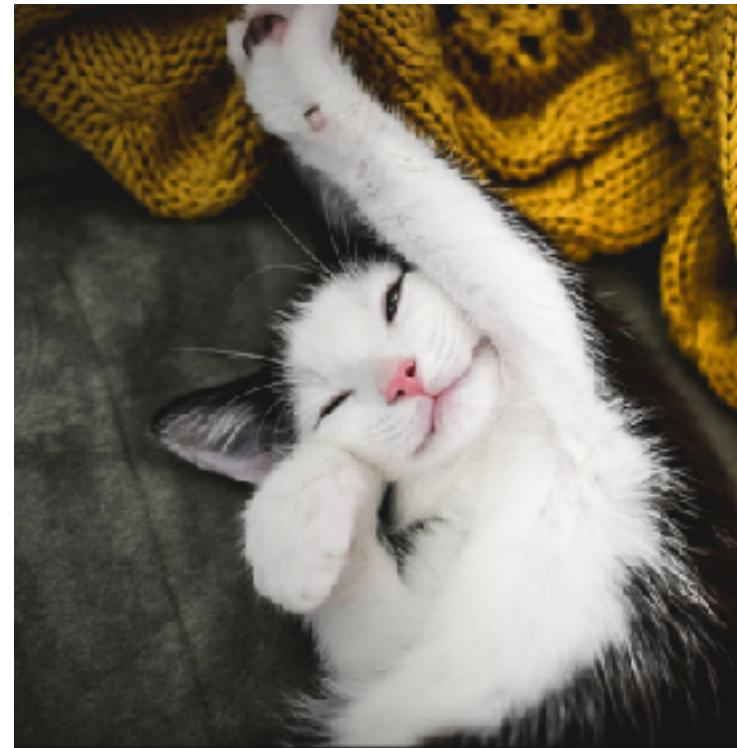
Can computers have “imaginations”?



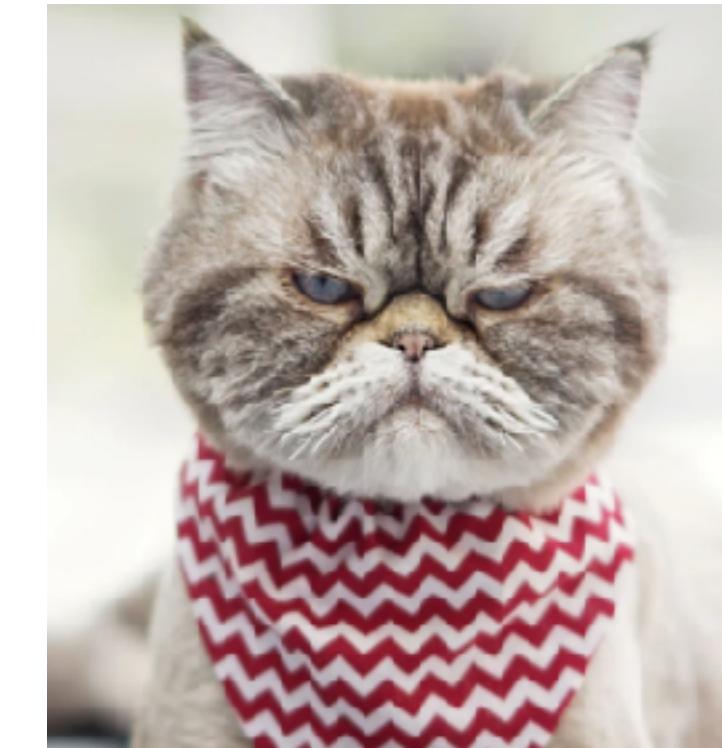
If the answer is **yes**, we can **condition** a DNN and make it generate certain outputs!

Move beyond associating inputs and outputs

Classification: mapping a distribution to a **certain** output



Happy cat



Angry cat



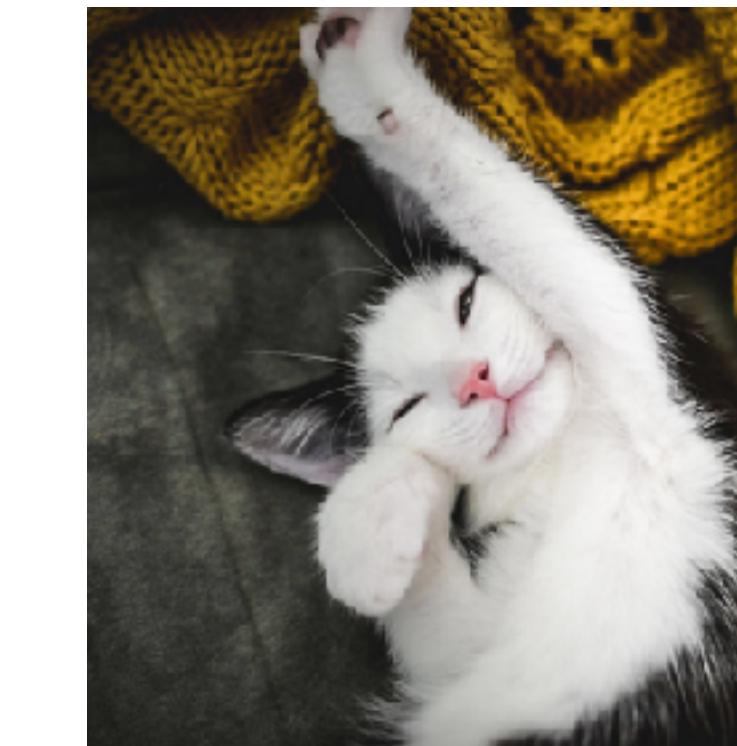
Relaxed cat



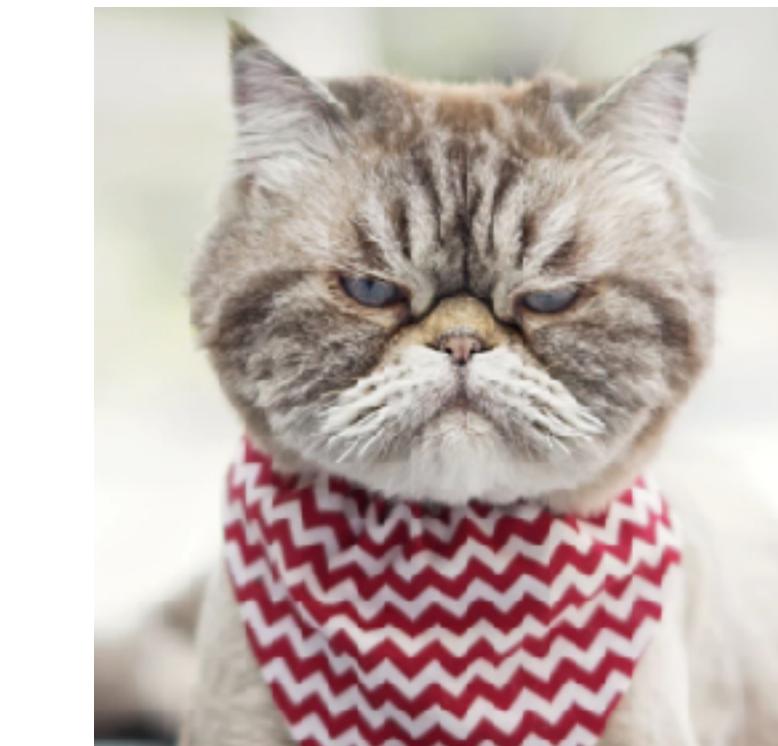
"Cats"

Generation: mapping an input to a probability **distribution**

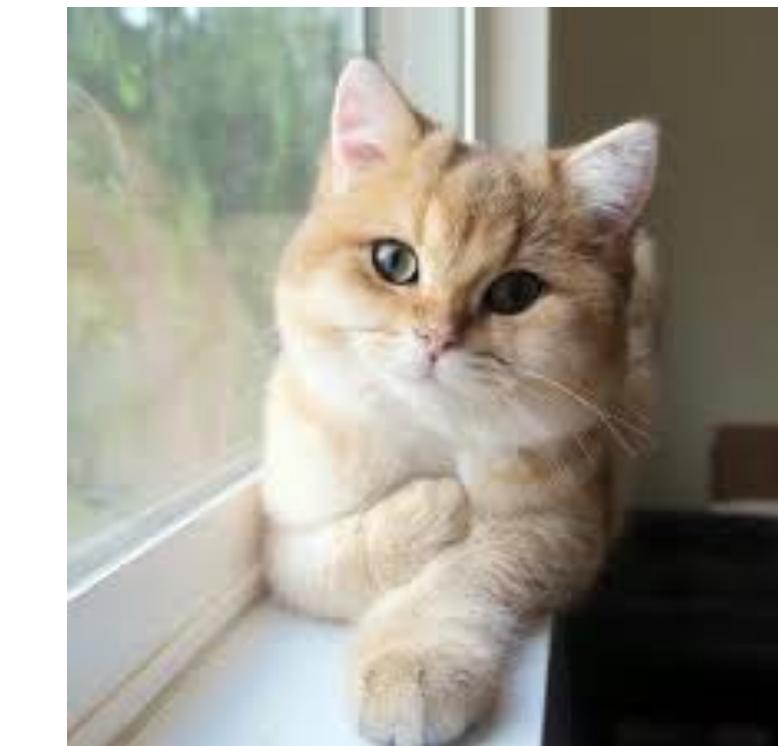
"Cats"



Happy cat



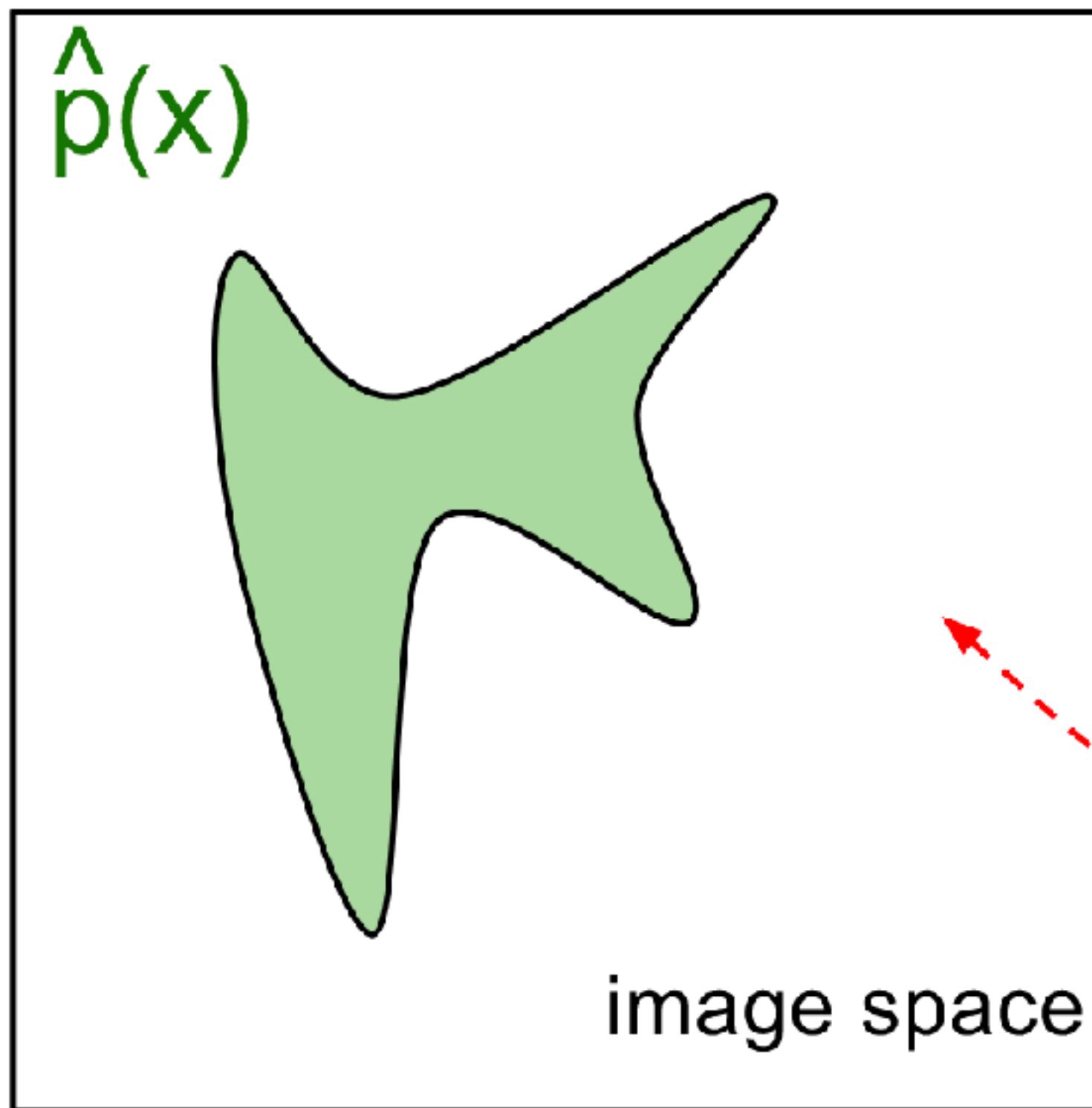
Angry cat



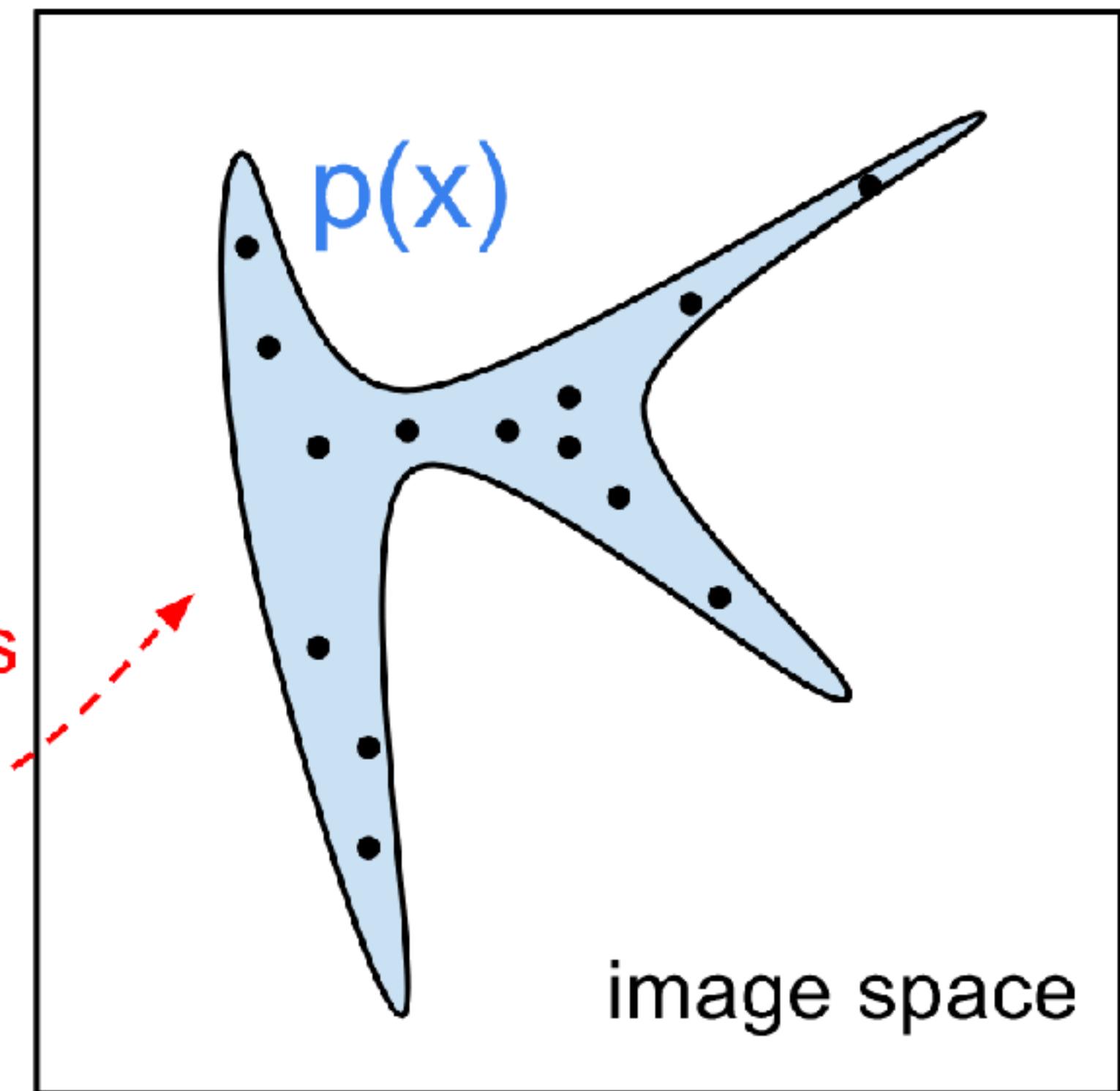
Relaxed cat

What is a Generative Model?

generated distribution



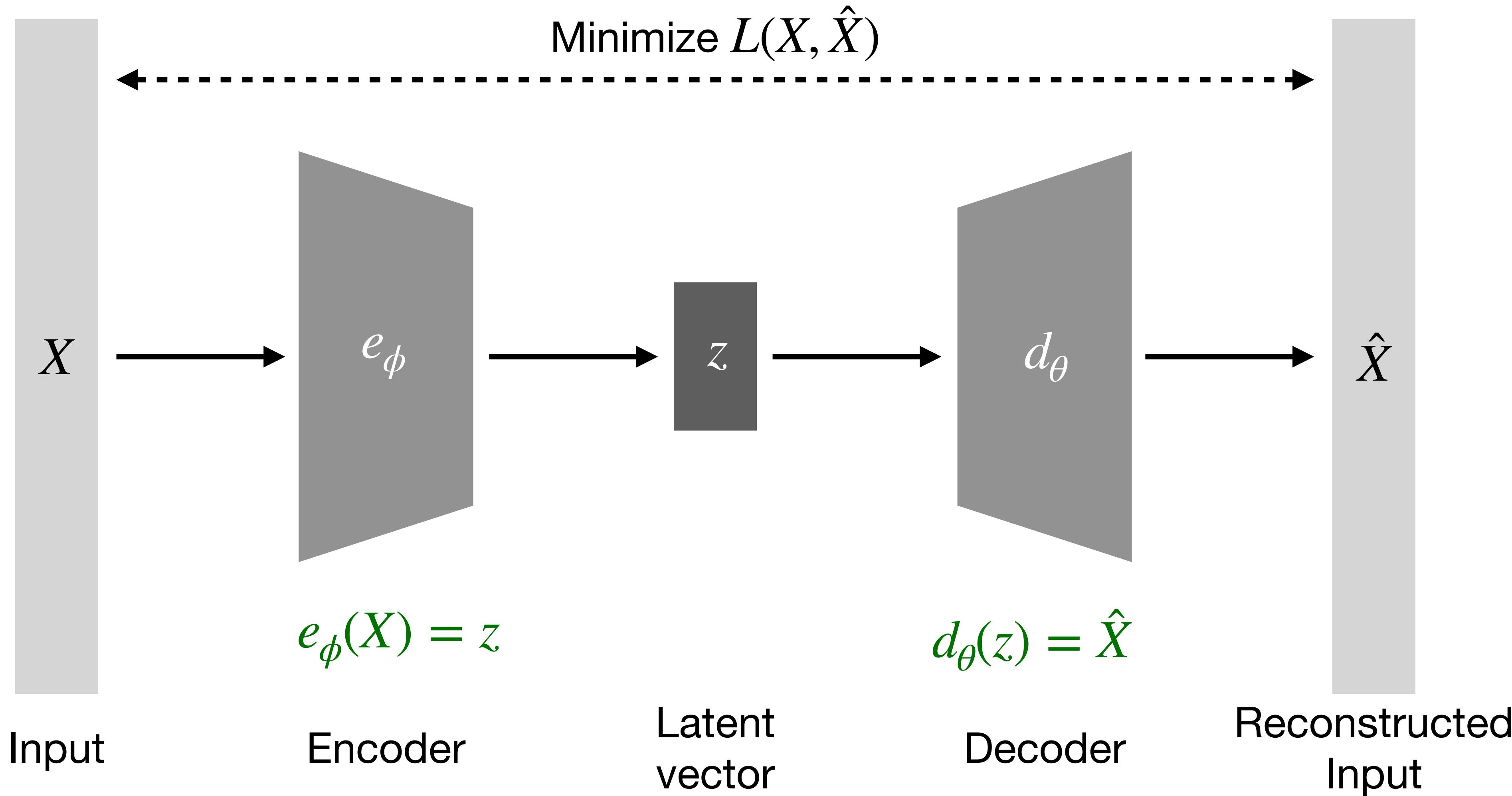
true data distribution



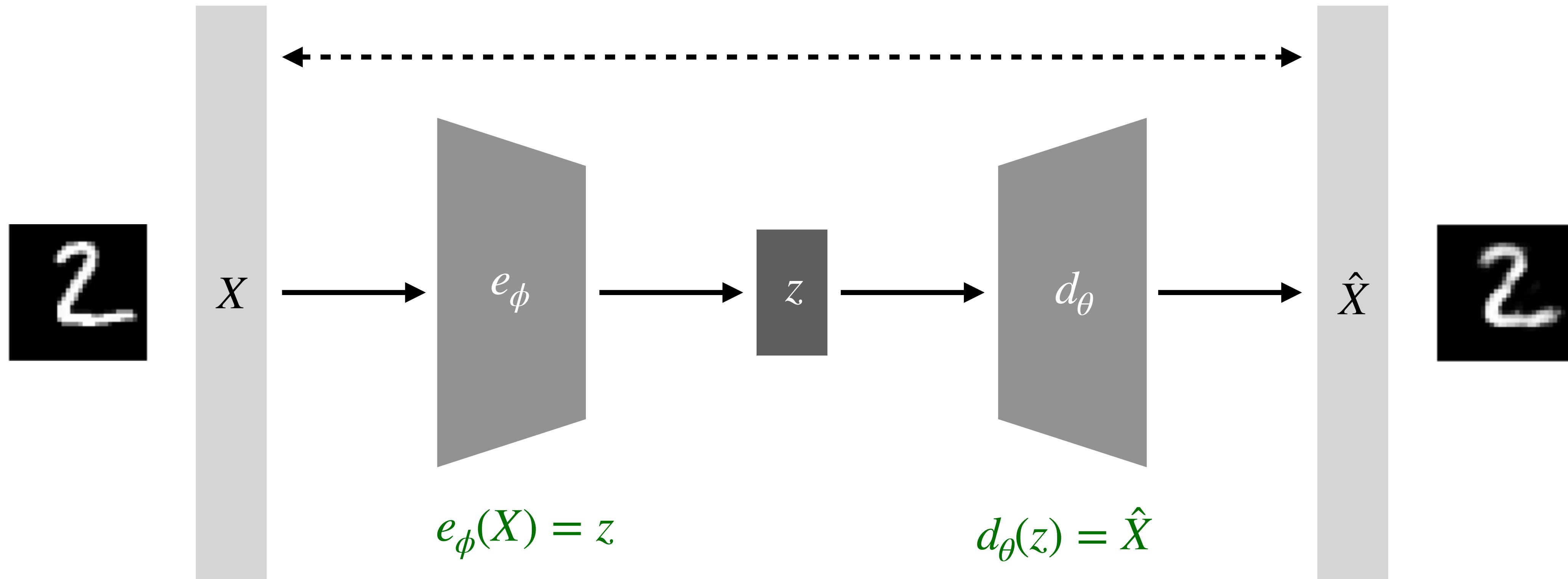
A (deep) generative model

- ... learns a distribution instead of a certain value
- ... allows uncertainties to be captured
- ... allows for (conditional) density estimation
- ... belongs to the category of unsupervised learning

Autoencoder



Autoencoder

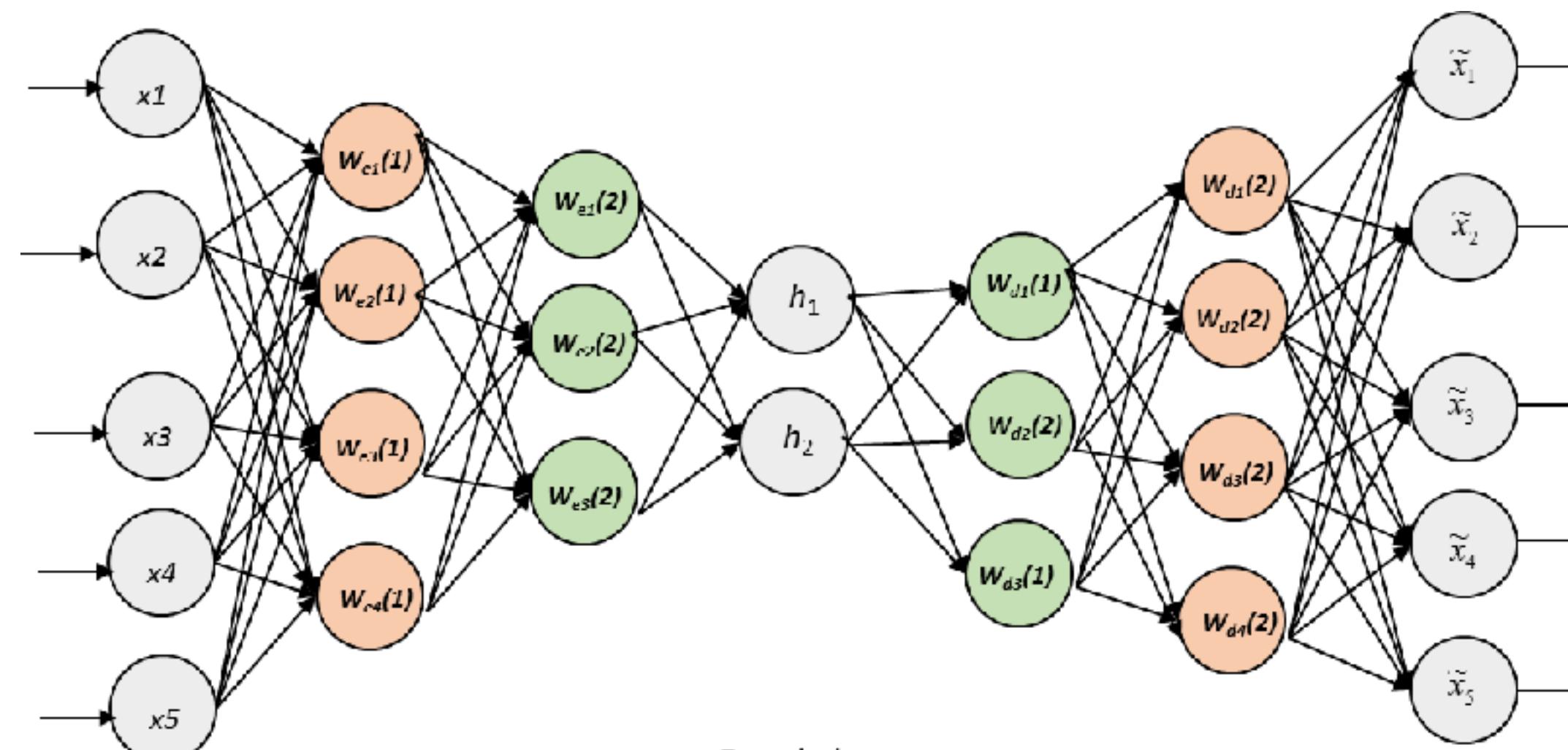


It looks like an autoencoder trivially copies X to \hat{X} , **but**:

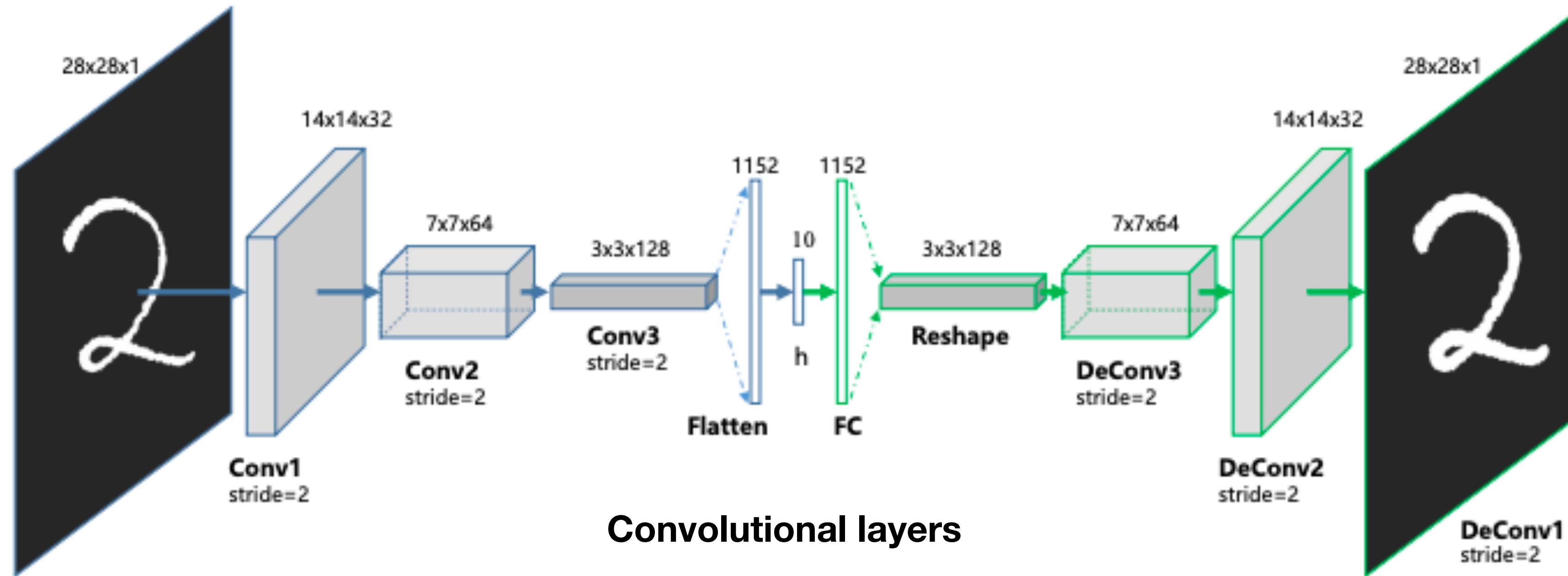
- The **latent vector** z has lower dimension than the input X , and therefore information are **compressed**.
- A good autoencoder selects relevant information and **discards irrelevant information**.

Autoencoder: Basic Implementations

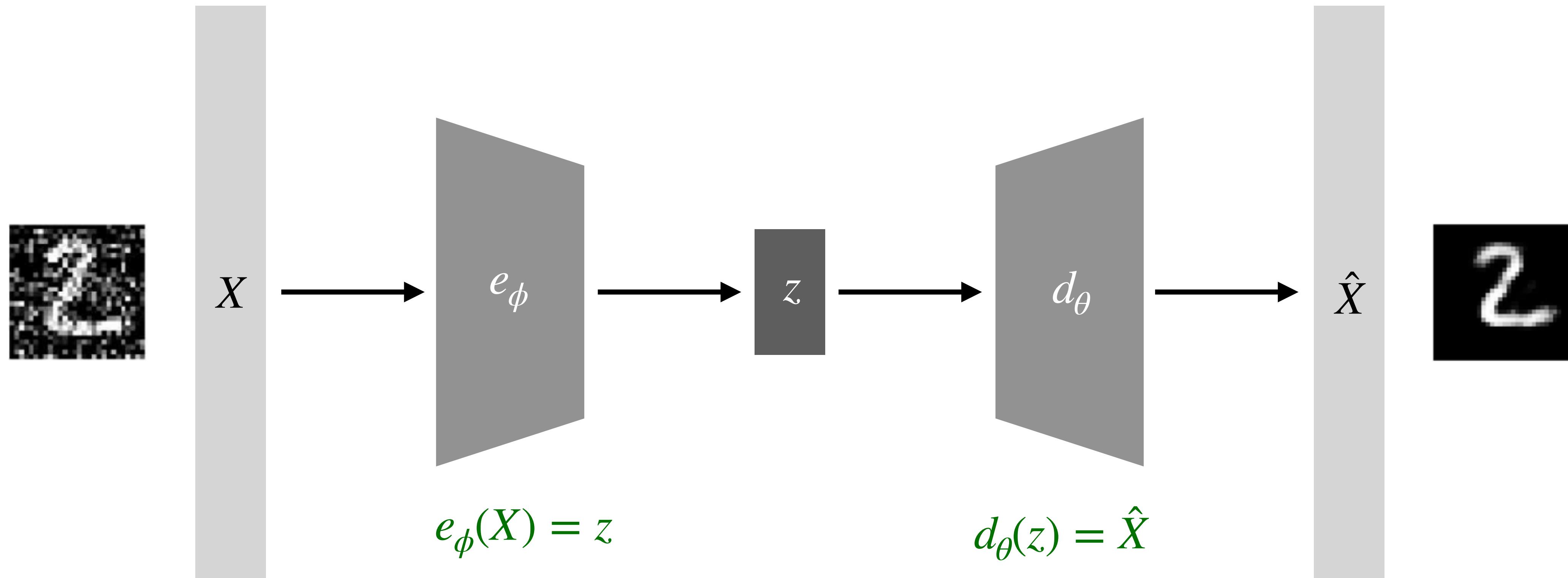
Fully connected layers



Convolutional layers



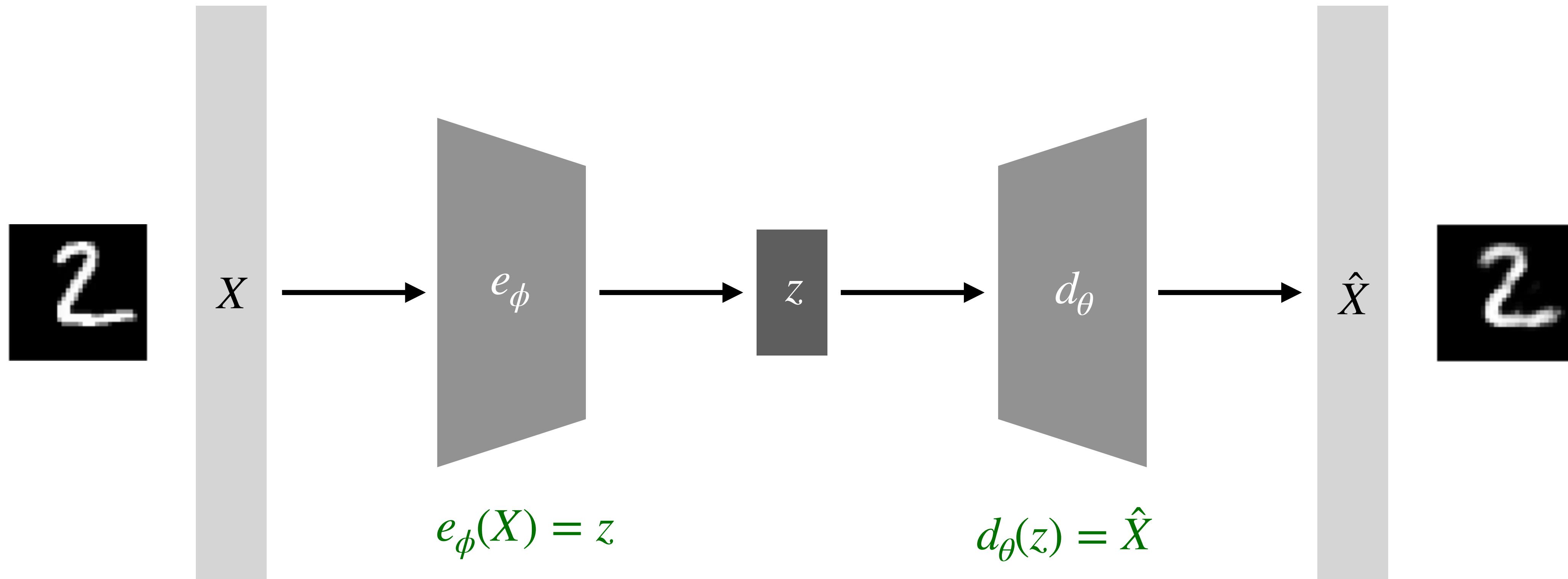
Autoencoder



Denoising autoencoder:

- Trained with noisy (perturbed) input data;
- The bottleneck z has limited capacity, and therefore **prevents overfitting**;
- The reconstruction data is clean of noise, because z doesn't have enough capacity to learn noise.

Autoencoder



Autoencoder for anomaly detection:

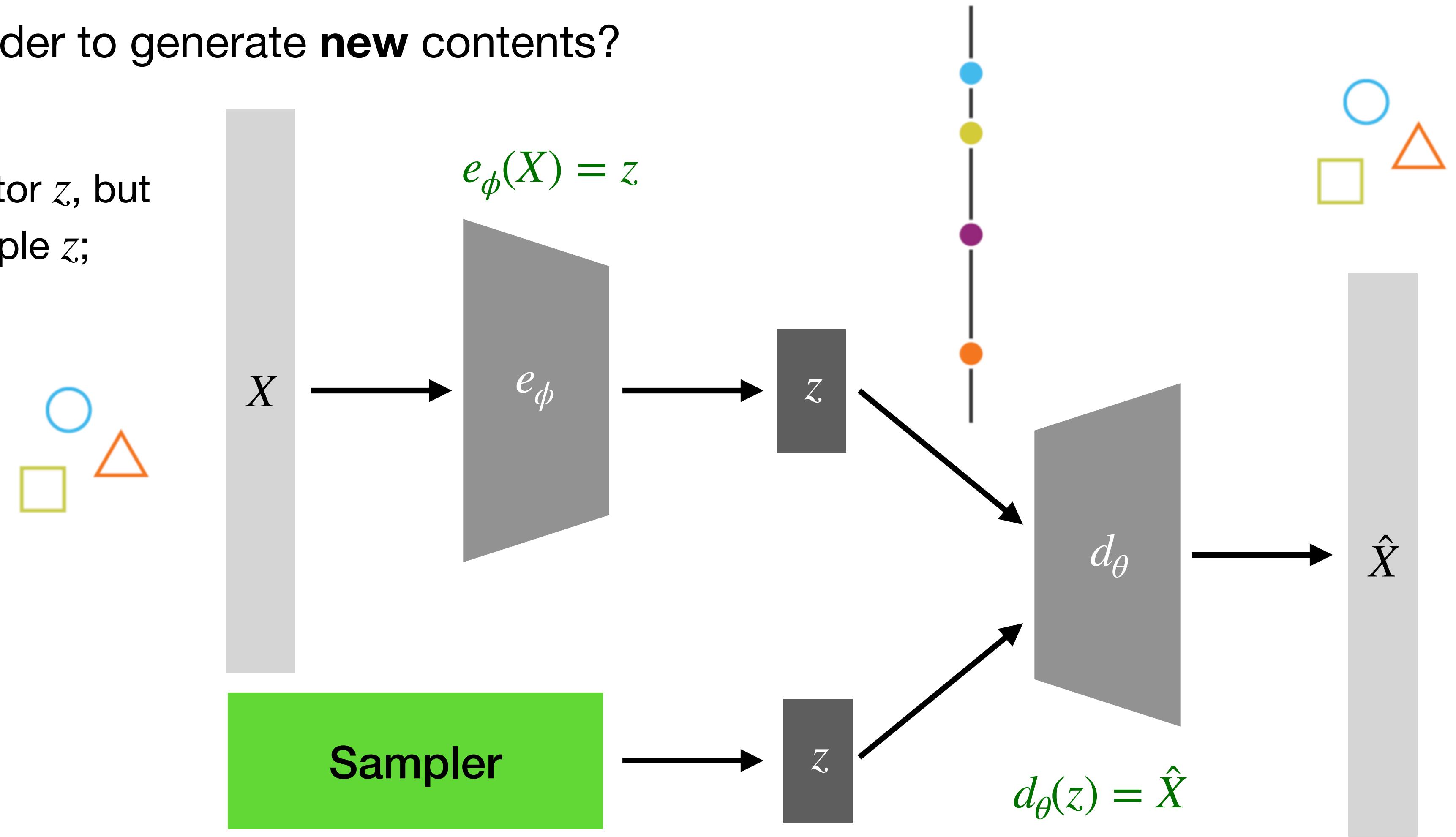
- Trained with regular input data (i.e., no anomaly);
- The bottleneck z encodes the statistical properties of the input data;
- Anomalies have different statistical properties; the autoencoder can't properly reconstruct them;
- Large reconstruction error → Anomaly.

Autoencoder

How can we let an autoencoder to generate **new** contents?

We can sample the latent vector z , but

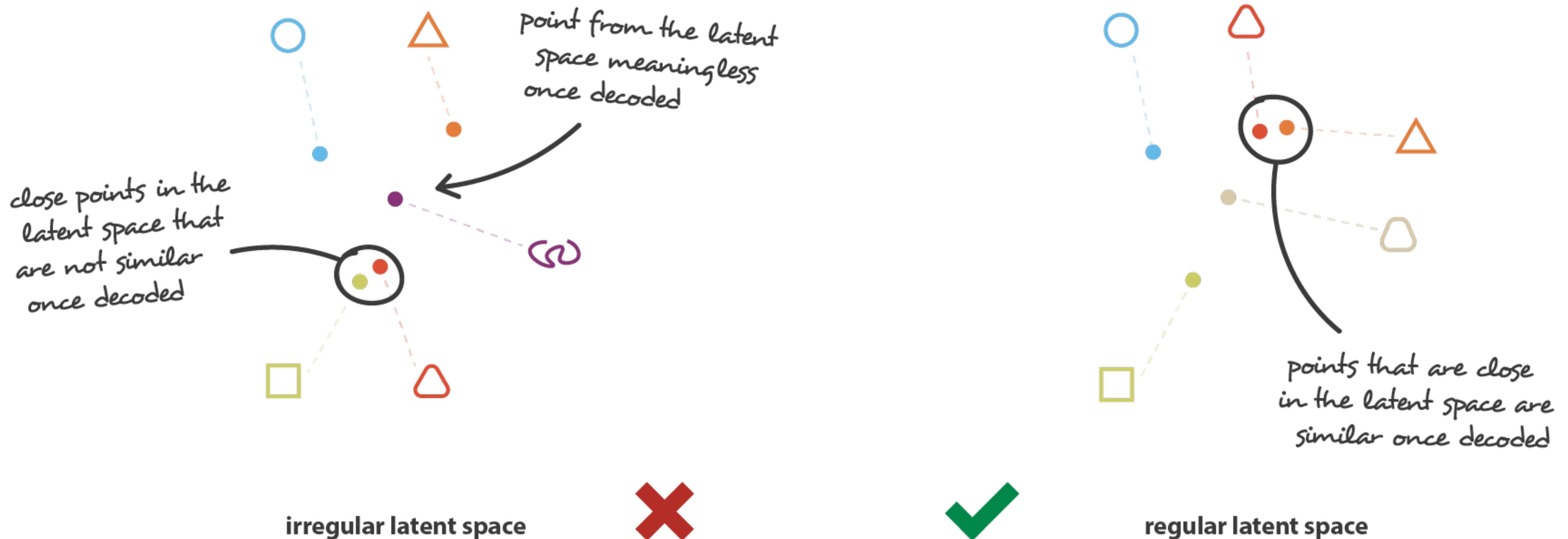
- We don't know how to sample z ;
- z can be very irregular.



We need to **regularize** the distribution (i.e., arrangement) of the latent space!



Latent Space Distribution

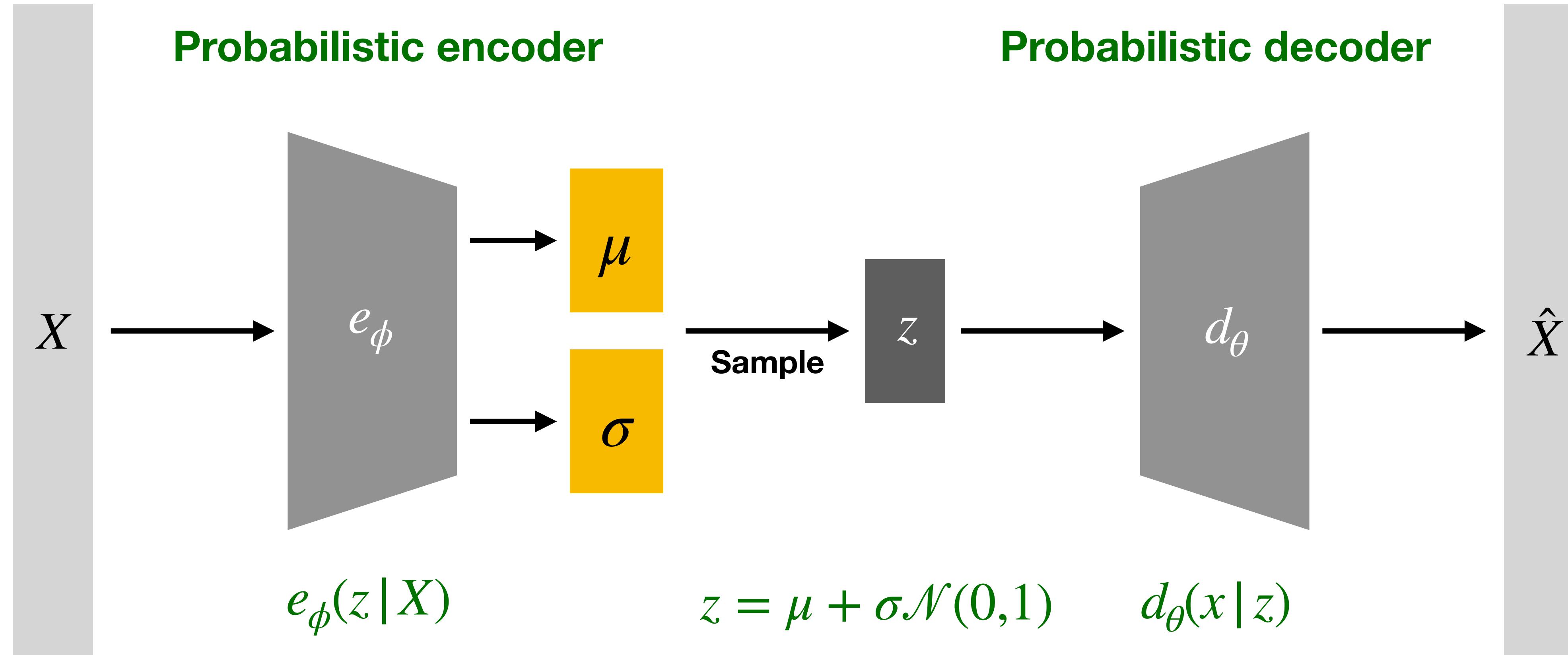


A regular latent space allows us to do **interpolation**.

New contents are generated by **interpolating** the latent space.

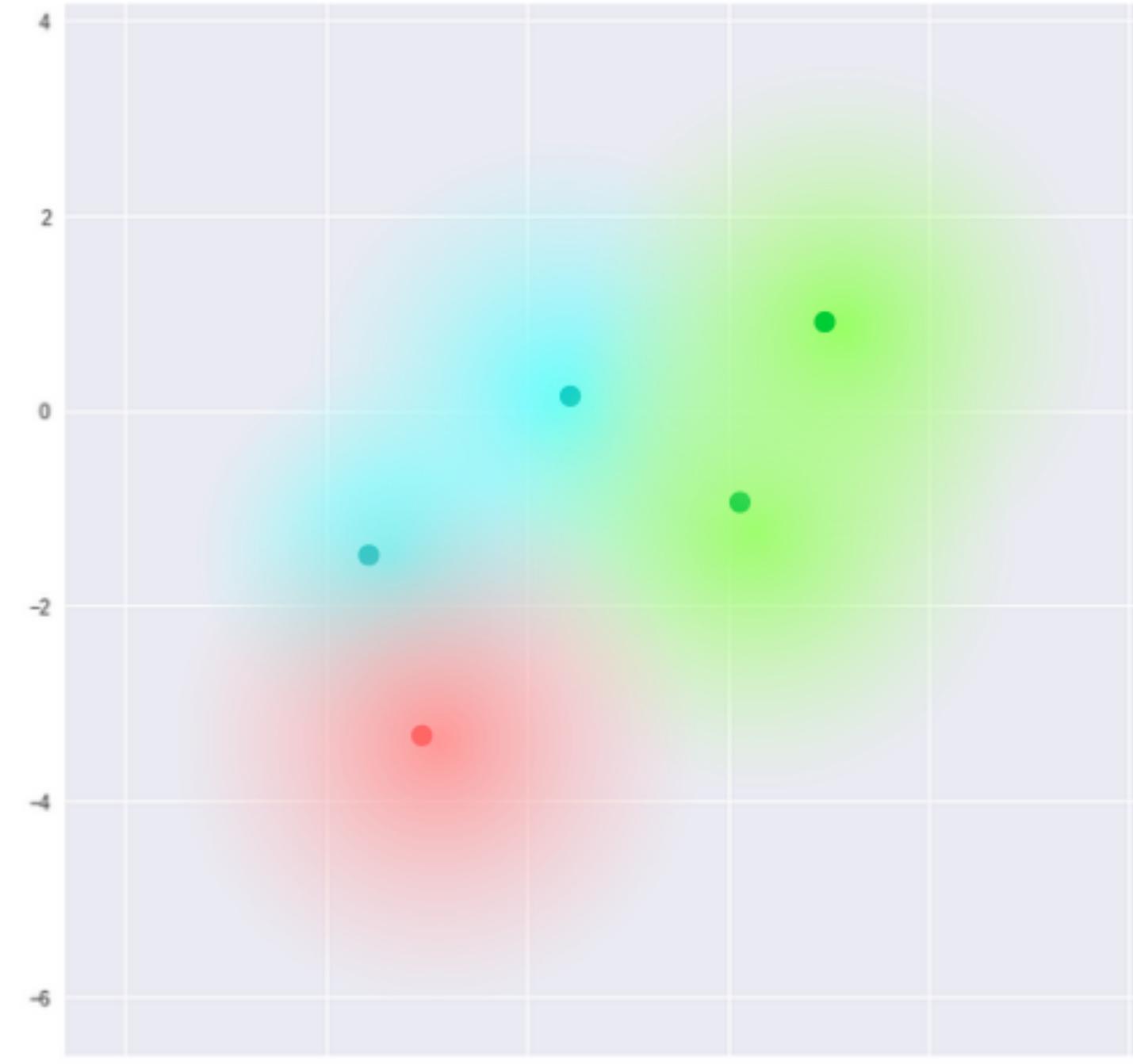
Image: Joseph Rocca (TowardDataScience)

Variational Autoencoder

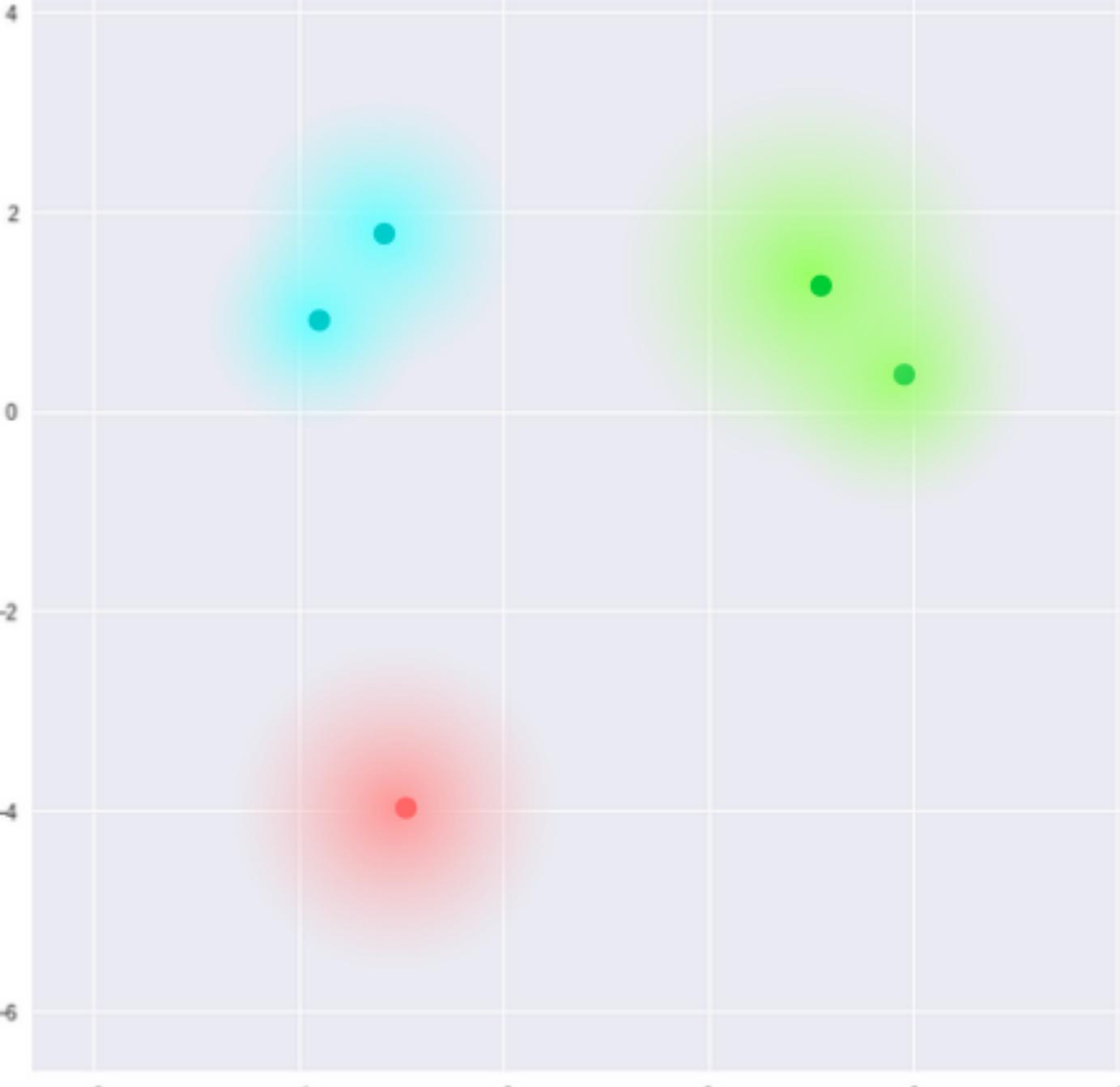


Latent space: tight, smooth, and complete

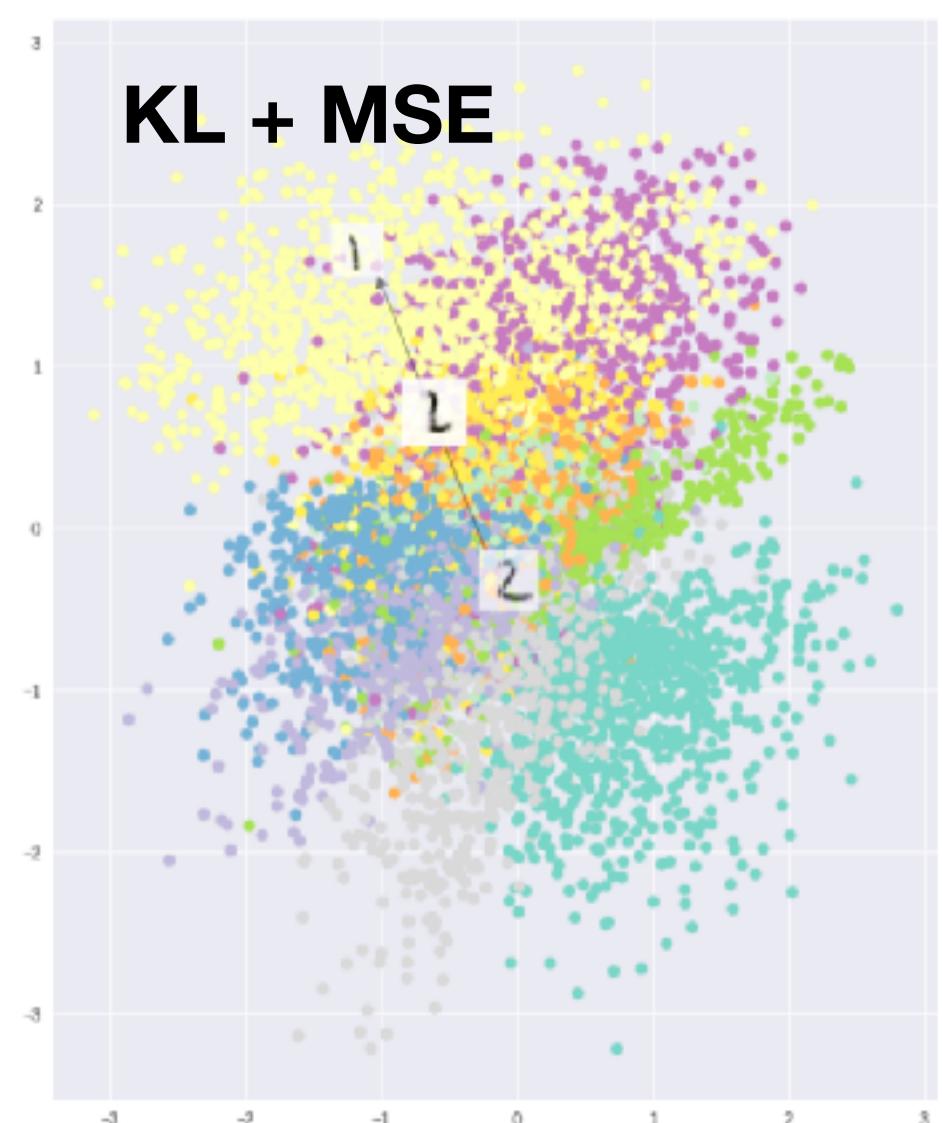
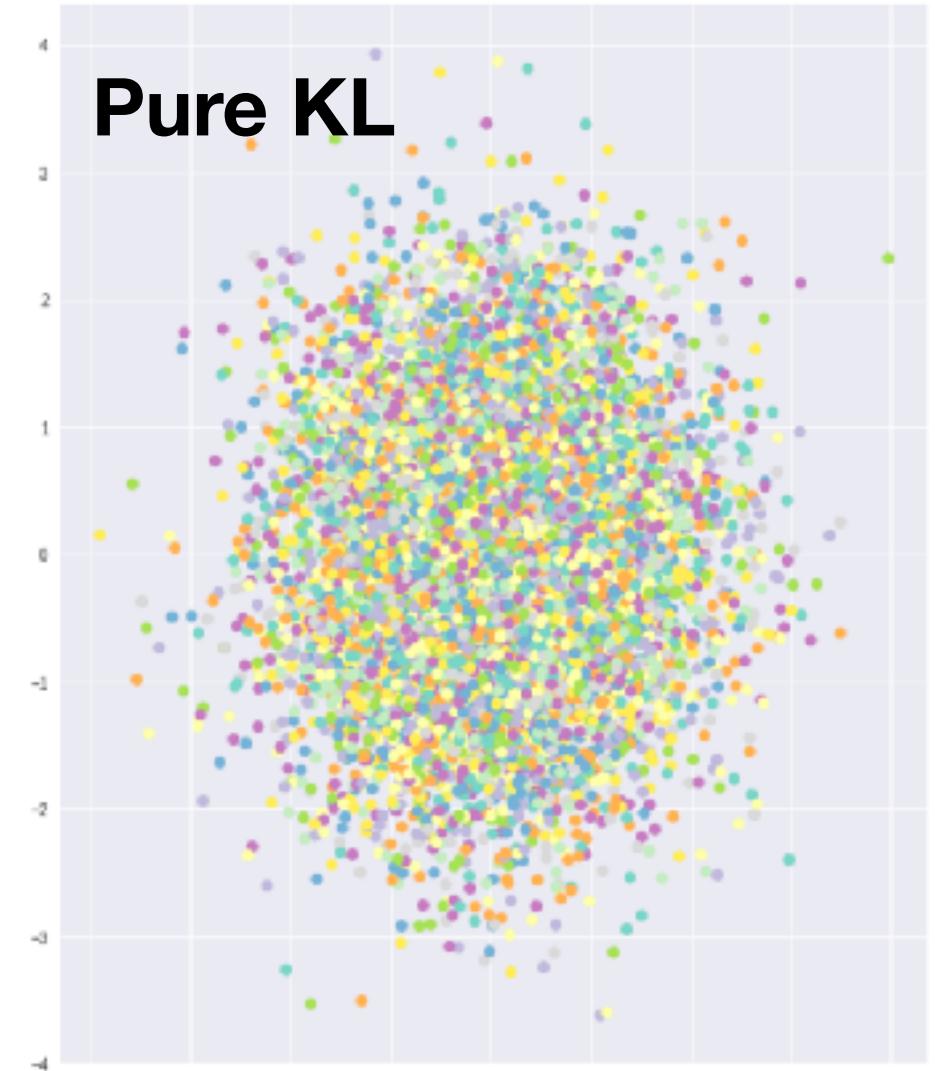
Without constraining μ and σ :



What we require



What we may inadvertently end up with

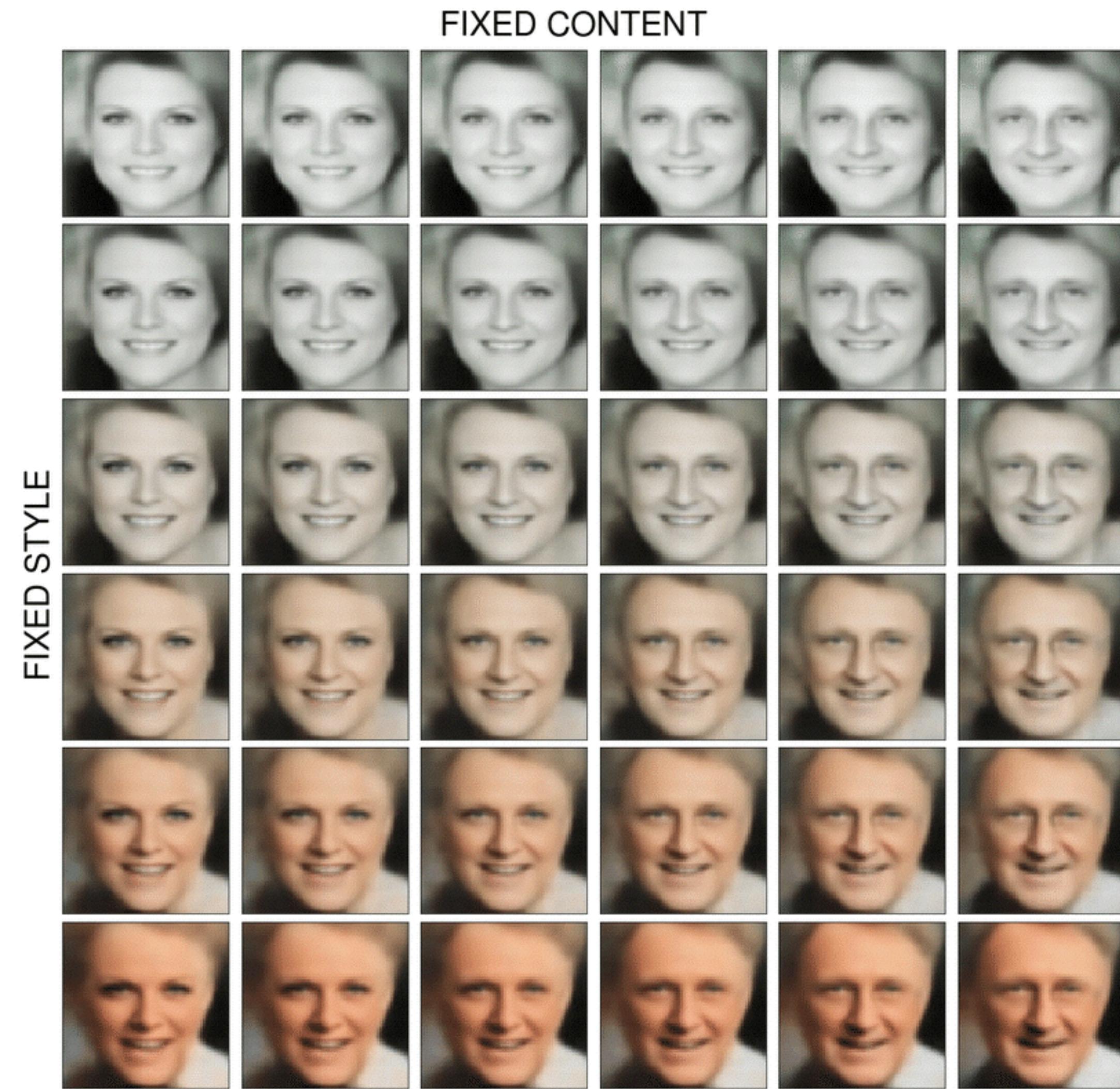


Solution:

- Use **KL divergence** to force clusters to get **close to each other**;
- Use **reconstruction losses** (e.g., MSE) to **avoid clusters from overlapping**.

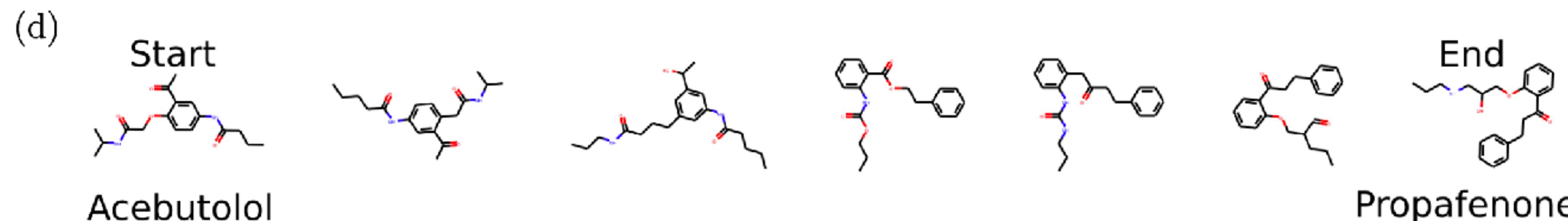
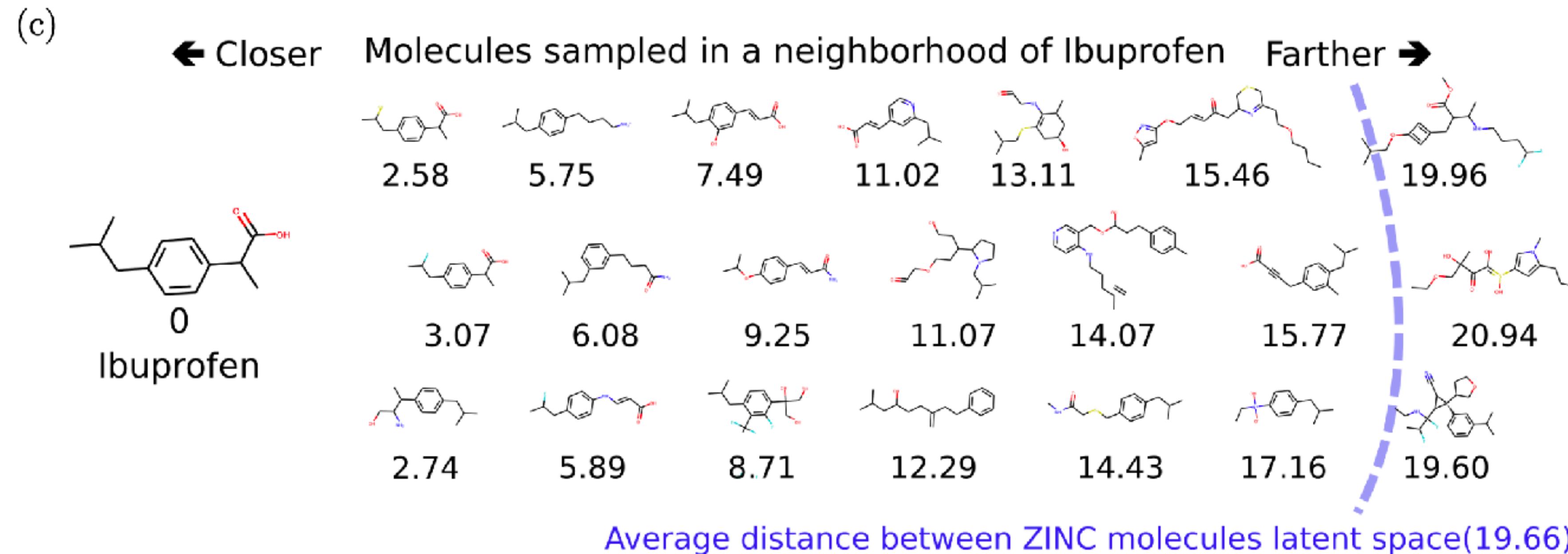
Latent vector interpolation

6 6 6 6 6 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9 4 4 4 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 2
9 2 2 2 2 2 2 2 2 8 5 5 8 0 0 0 0 0 0 0 0 2
9 4 2 2 2 2 2 2 2 3 3 5 5 5 8 0 0 0 0 0 0 2
9 9 4 2 2 2 2 2 3 3 3 3 5 5 8 8 5 5 3 3
9 9 9 4 2 2 2 2 3 3 3 3 5 5 5 5 5 5 3 3
9 9 9 9 9 3 2 3 3 3 3 3 5 5 5 5 5 3 7
9 9 9 9 9 9 8 3 3 3 3 3 3 5 5 8 8 8 7
9 9 9 9 9 9 8 8 3 3 3 3 8 8 8 8 8 8 7
9 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8 8 8 7
9 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8 8 8 7
9 9 9 9 9 9 8 8 8 8 8 8 8 8 6 6 6 6 5 7
9 9 9 9 9 9 9 8 8 8 8 8 8 8 6 6 6 6 6 5 7
9 9 9 9 9 9 9 9 8 8 8 8 8 8 6 6 6 6 6 6 5 7
9 9 9 9 9 9 9 9 9 8 8 8 8 8 8 6 6 6 6 6 6 5 7
9 9 9 9 9 9 9 9 9 8 8 8 8 8 8 6 6 6 6 6 6 6 5 7
9 9 9 9 9 9 9 9 9 8 8 8 8 8 8 6 6 6 6 6 6 6 6 6 7 1
9 9 9 9 9 9 9 9 9 9 8 8 8 8 8 6 6 6 6 6 6 6 6 6 6 6 6 7 1
9 9 9 9 9 9 9 9 9 9 9 8 8 8 8 6 6 6 6 6 6 6 6 6 6 6 6 7 1
9 9 9 9 9 9 9 9 9 9 9 9 8 8 8 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 1
9 9 9 9 9 9 9 9 9 9 9 9 9 8 8 6 7 1

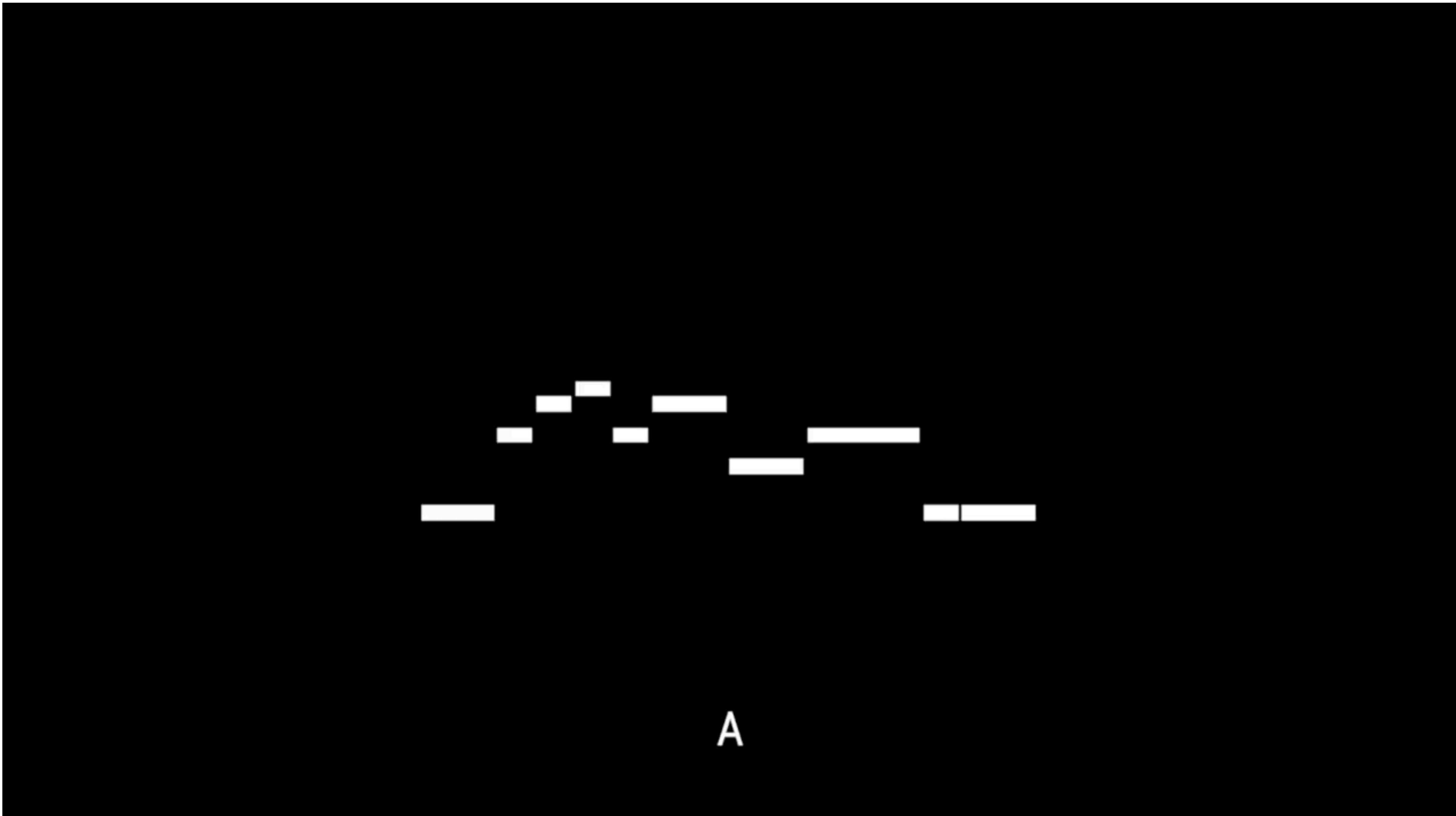


Application: VAE for Drug Discovery

Proposing candidate molecules by interpolating the latent space.



Application: VAE for music generation



Credit: Google AI

Alternative Approaches?

Variational autoencoders is elegant and theoretically pleasing.
However, the generated images (data) are **blurry** (causes unknown).

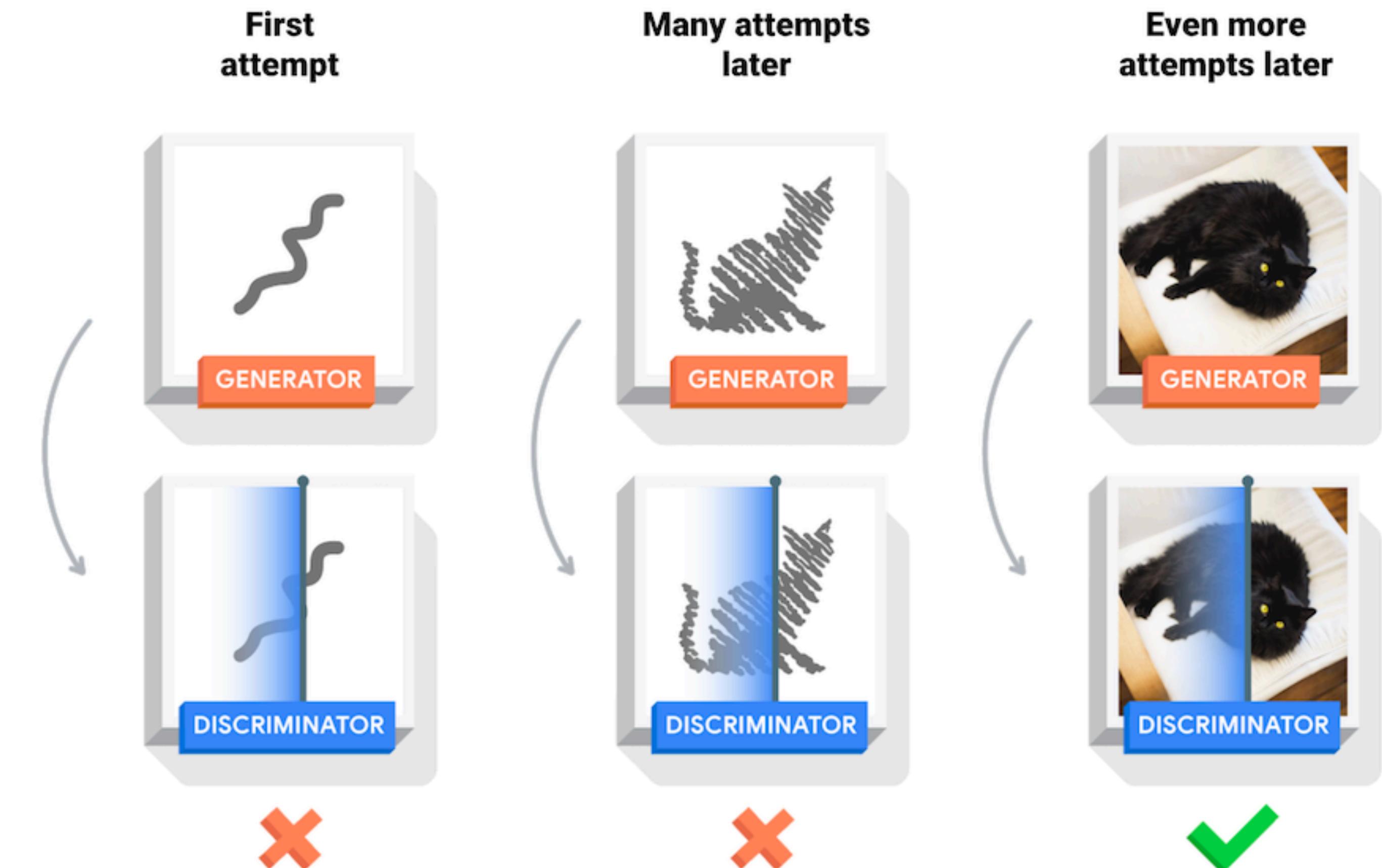
How to generate high resolution images (data)?



Adversarial Training

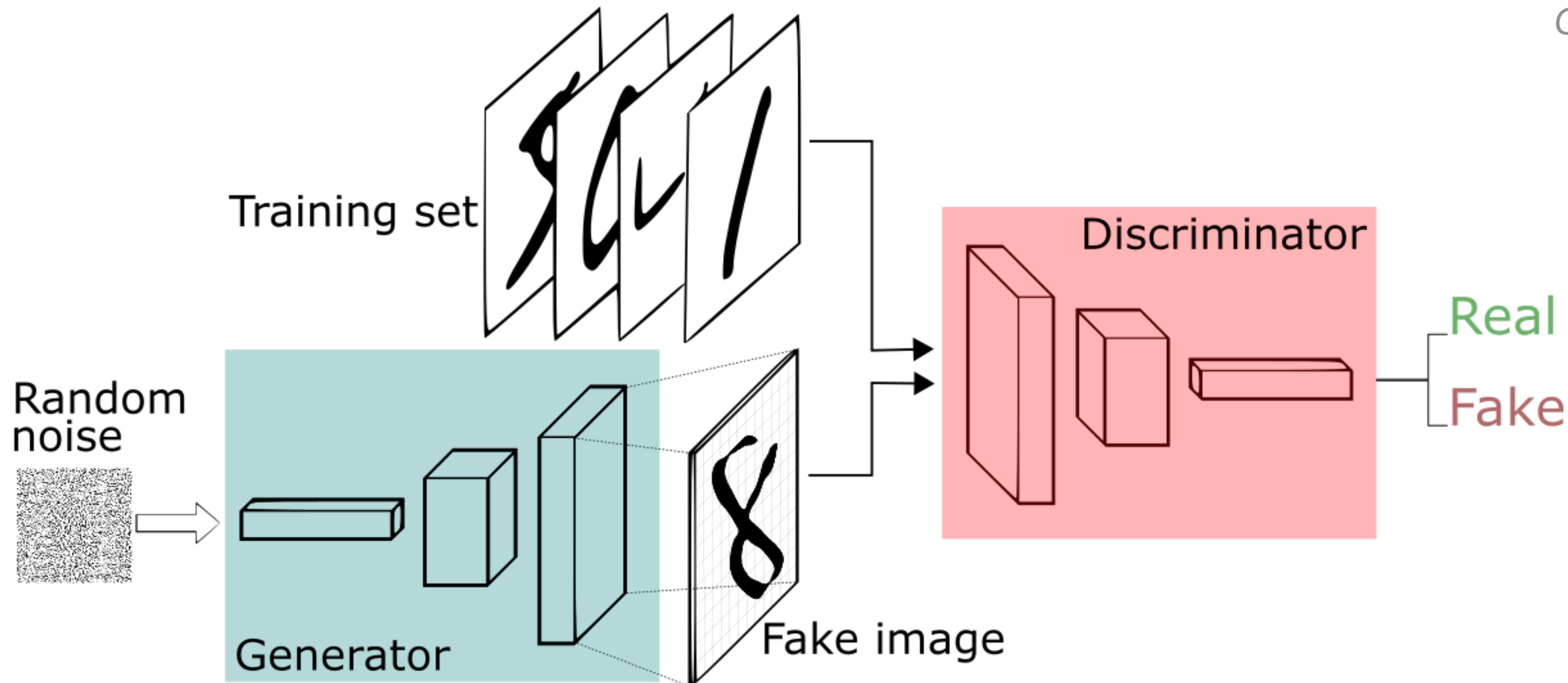
During the training:

- The **generator** progressively **becomes better** in creating images that looks real;
- The **discriminator** progressively **becomes better** in telling them apart.
- The process reaches an **equilibrium** when the discriminator can no longer distinguish real images from fakes.



Adversarial Training

Goodfellow et al. (2014, arXiv:1406.2661)



Log proba of D predicting that
the **real** images are **genuine**

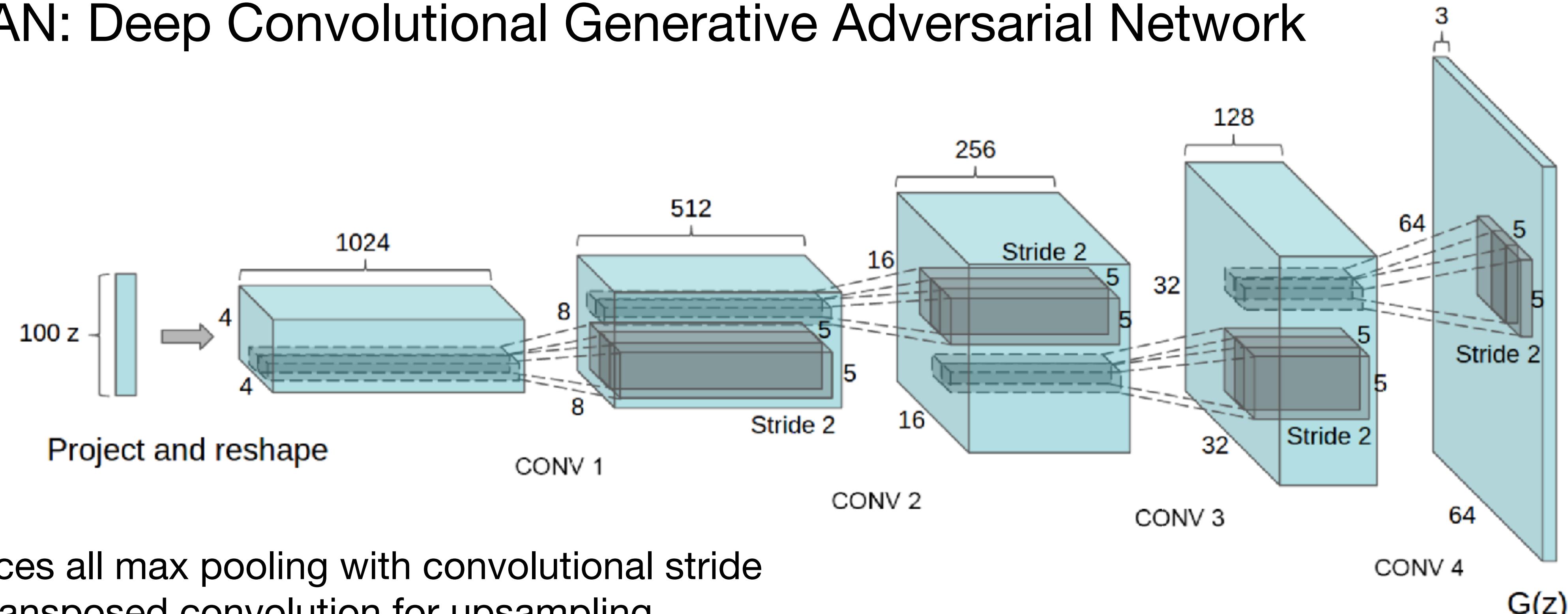
Log proba of G predicting that
the **generated** images are **fake**

Nash equilibrium: $\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_{\text{model}}(z)}[\log(1 - D(G(z)))]$

“Nash equilibrium is a proposed solution of a **non-cooperative game** involving two or more players in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only their own strategy.” — [Wikipedia](#)

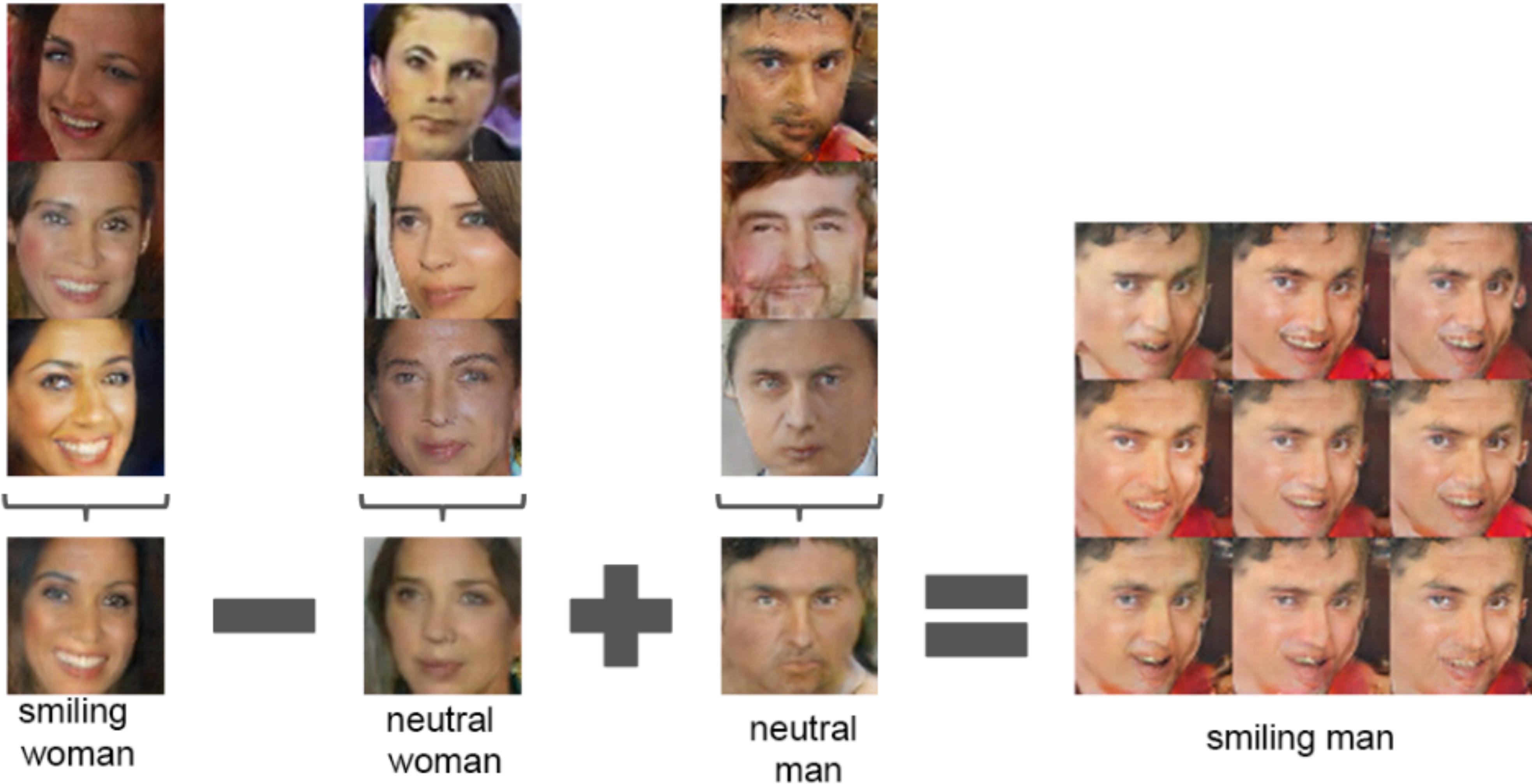
Implementation: DCGAN

DCGAN: Deep Convolutional Generative Adversarial Network



- Replaces all max pooling with convolutional stride
- Use transposed convolution for upsampling
- Eliminate fully connected layers
- Use batch normalization except the output layer for the generator
- Use batch normalization for the input layer of the discriminator
- Use ReLU in the generator except for the output (which uses tanh)
- Use LeakyReLU in the discriminator.

Implementation: DCGAN

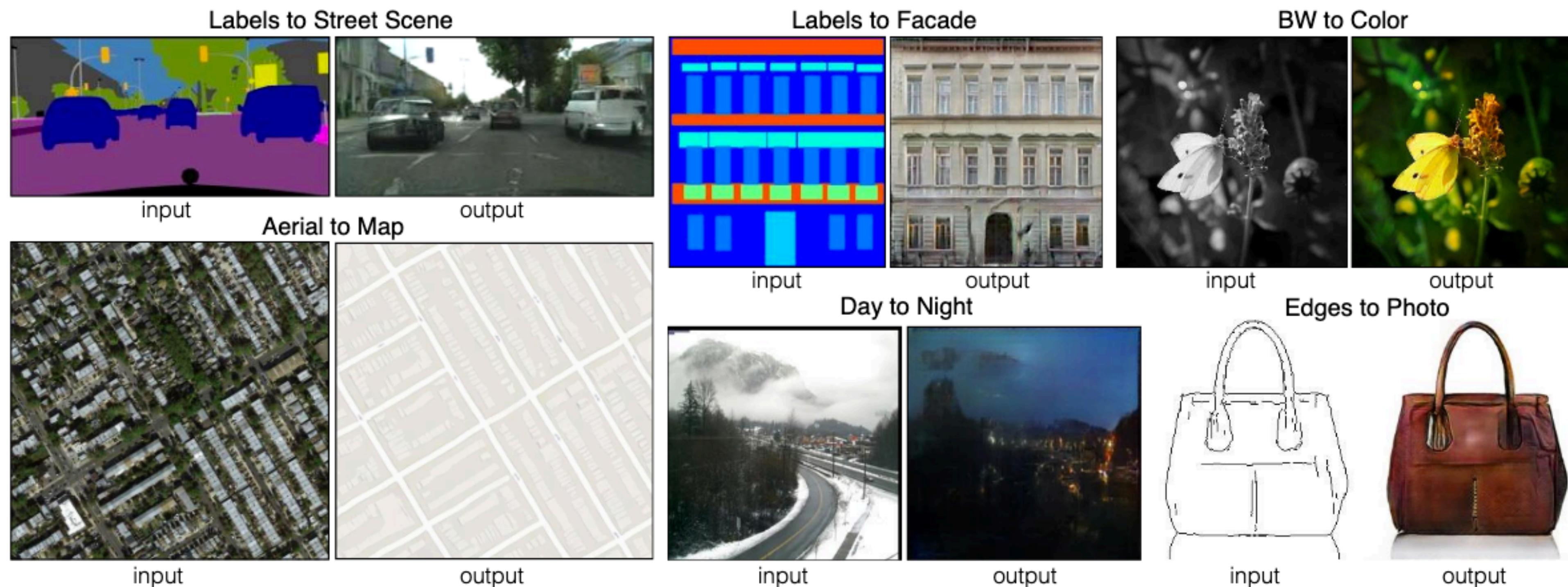


Implementation: CGAN

CGAN: Conditional Generative Adversarial Network

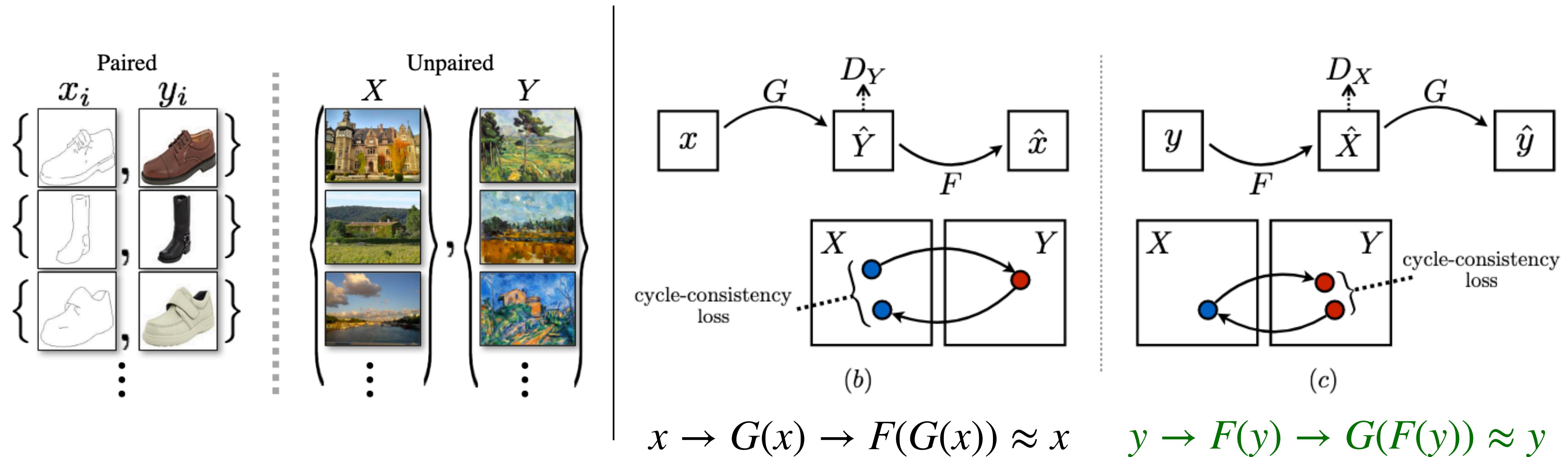
Normal GAN: $\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_{\text{model}}(z)}[\log(1 - D(G(z)))]$

Conditional GAN: $\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x | y)] + \mathbb{E}_{z \sim p_{\text{model}}(z)}[\log(1 - D(G(z | y)))]$

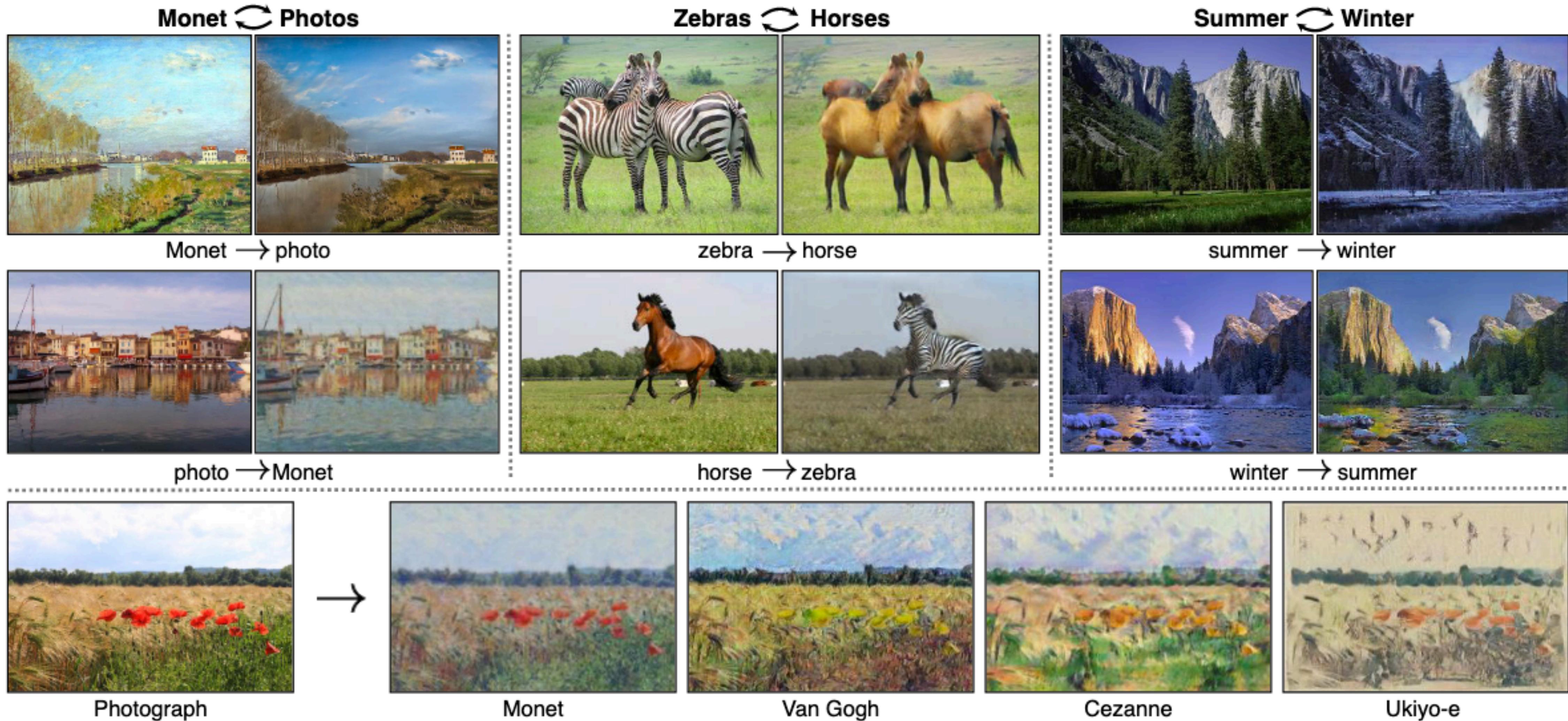


Implementation: CycleGAN

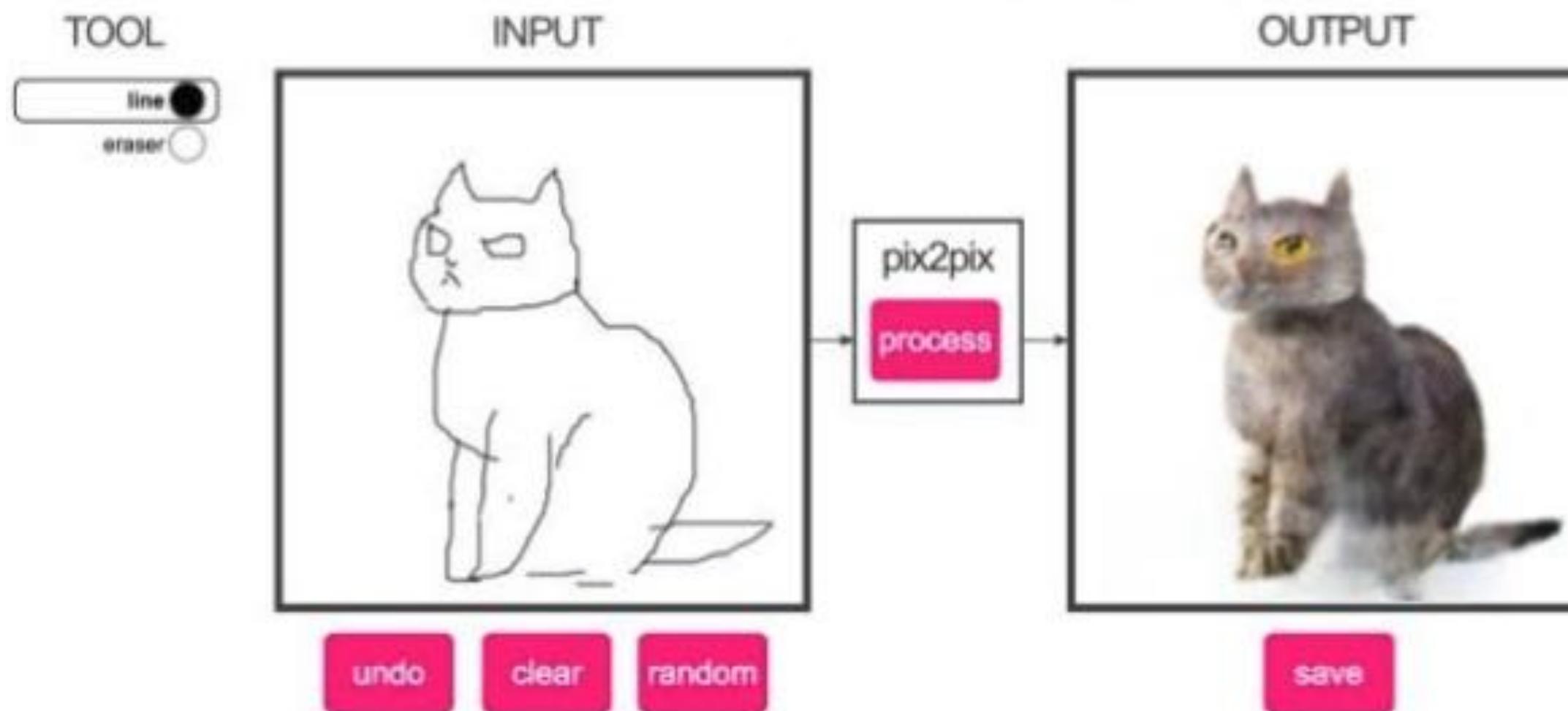
Image translation typically requires **paired** datasets. What if we don't have paired datasets?



Implementation: CycleGAN



Implementation: CycleGAN



Works!



Hmmm...

The quality of GAN-generated images

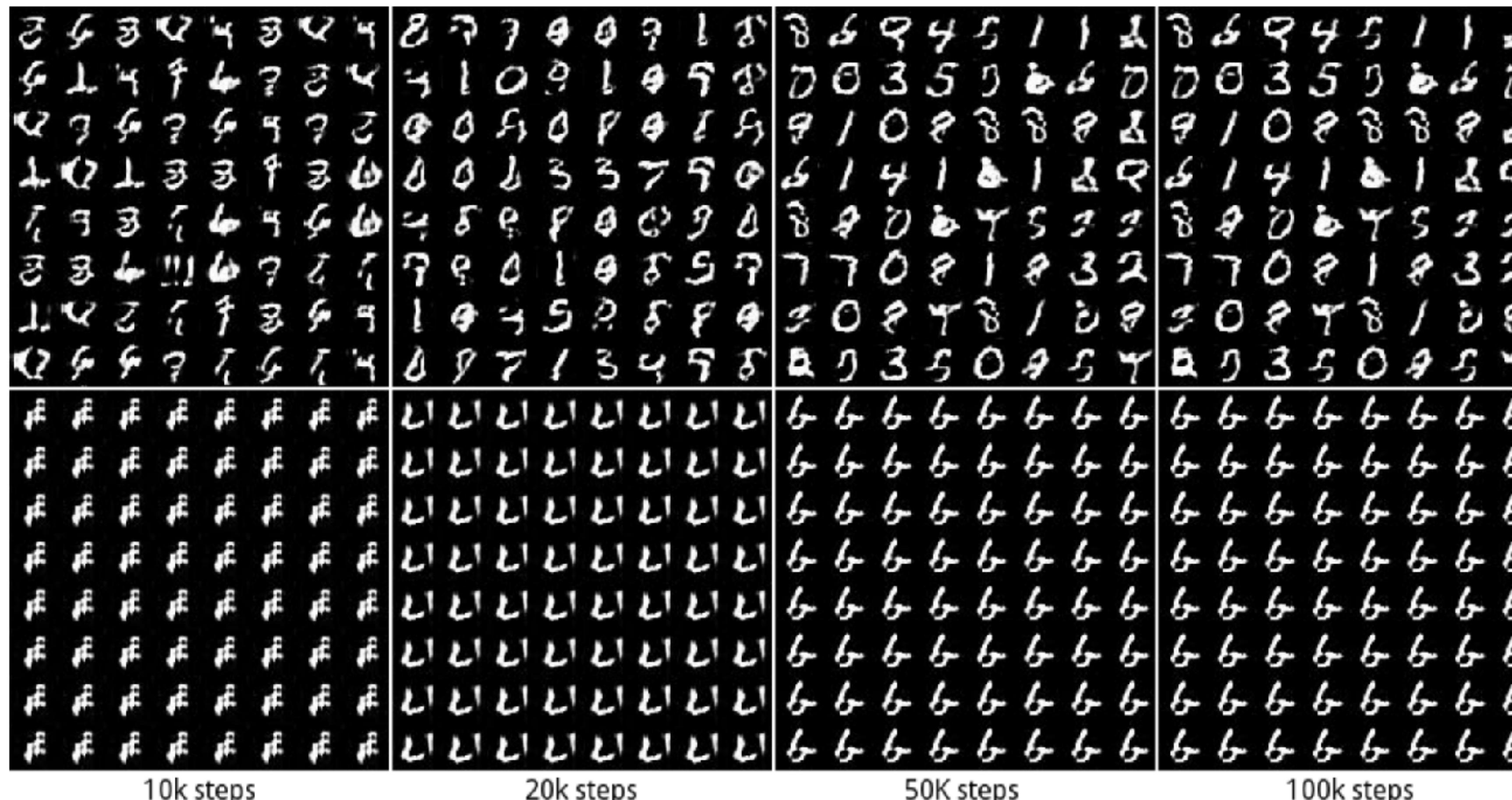


Problems with GANs

Convergence not guaranteed: the two models may oscillate and never converge.

Mode collapse: the generator only produces limited variation.

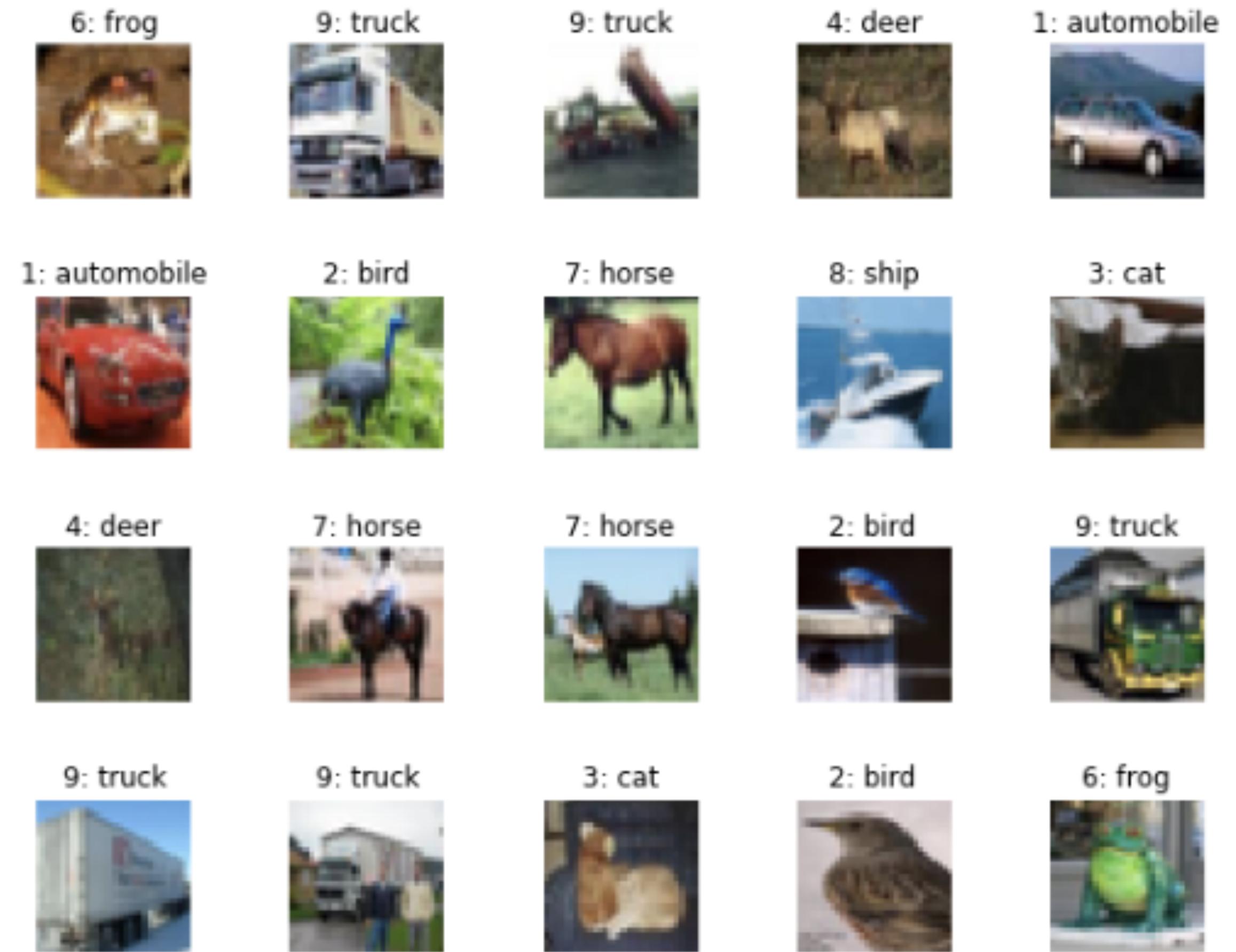
Diminished gradient: the discriminator gets too successful and the generator learns nothing.



Mode collapse of GANs



<https://jupyter.lisa.surfsara.nl/jhlptc008/>



CNN Classification on the CIFAR-10 Dataset

CIFAR = Canadian Institute for Advanced Research