Artificial Intelligence

Lecture 11：Deep Learning I

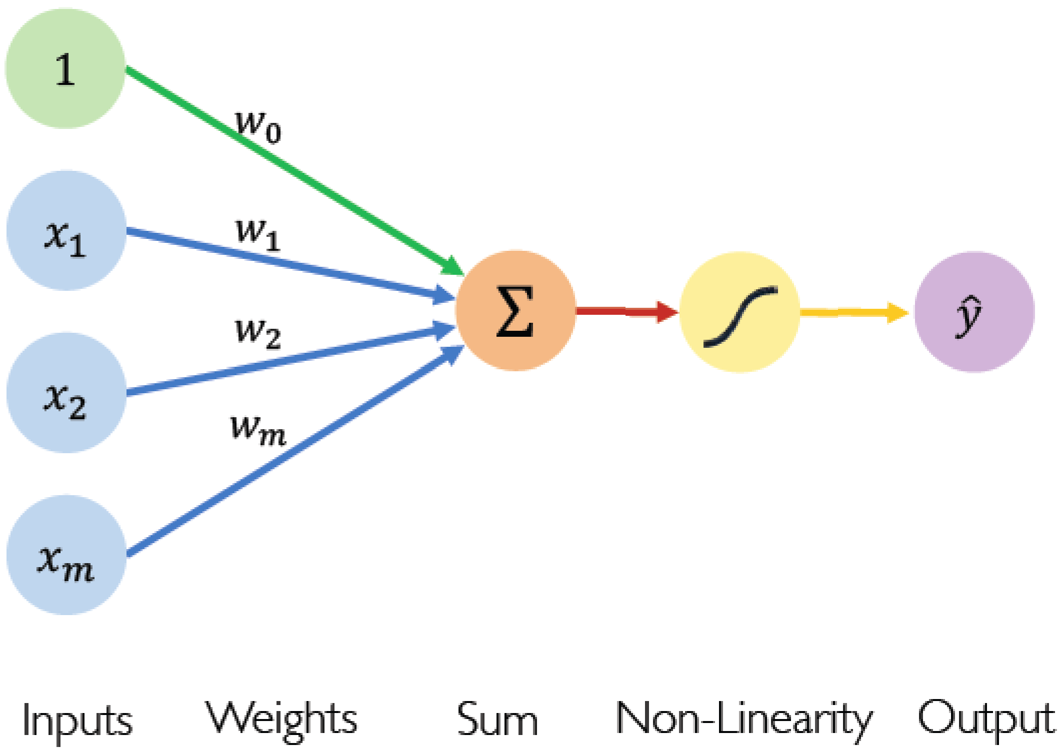
Xiaojin Gong

2022-05-16

Outline

- Convolutional Neural Networks
- Recurrent Neural Networks
- Autoencoder
- Generative Adversarial Networks

Review: Perceptron



Output

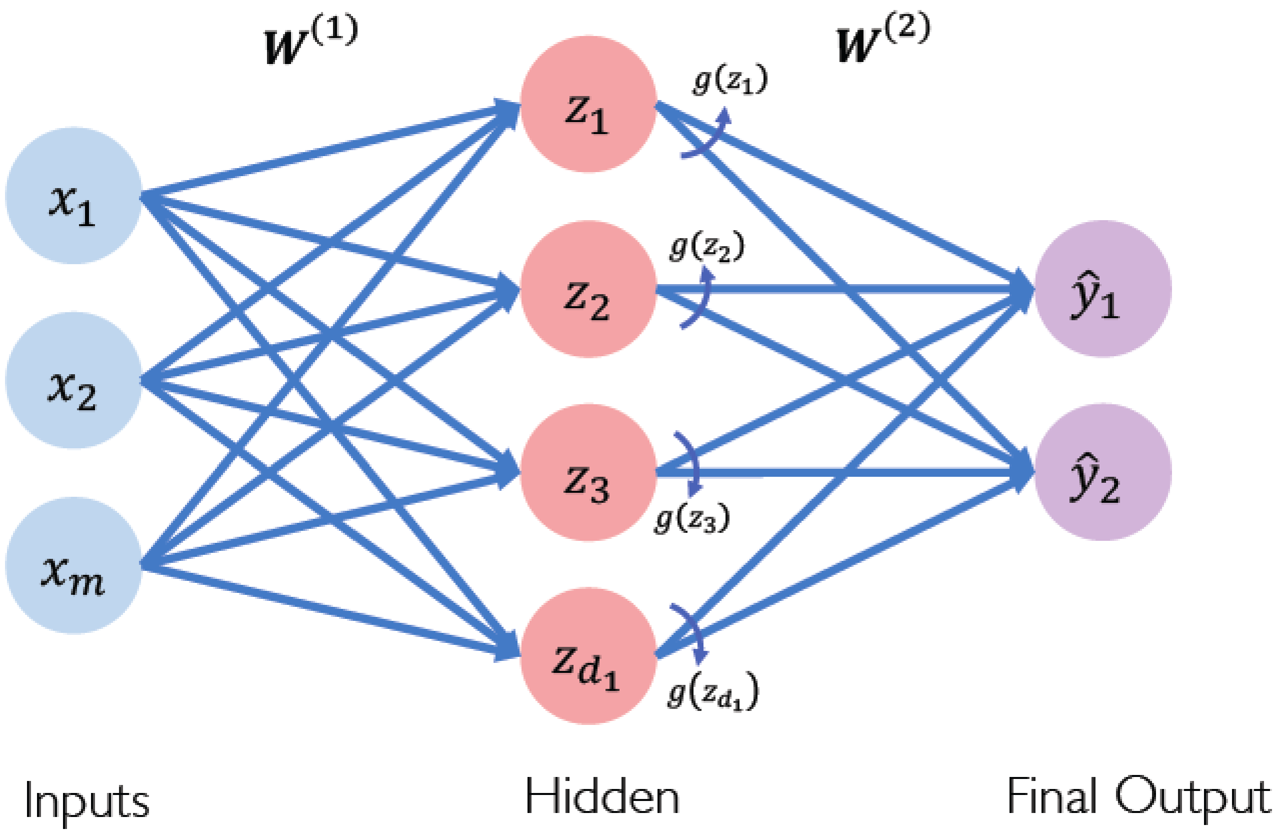
Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

Non-linear activation function

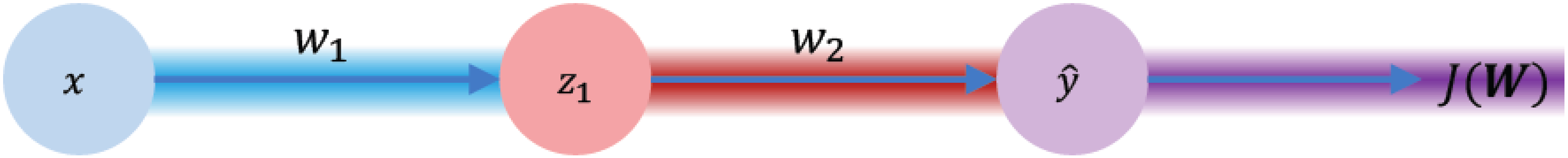
Bias

Review: ANN



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)} \right)$$

Review: ANN



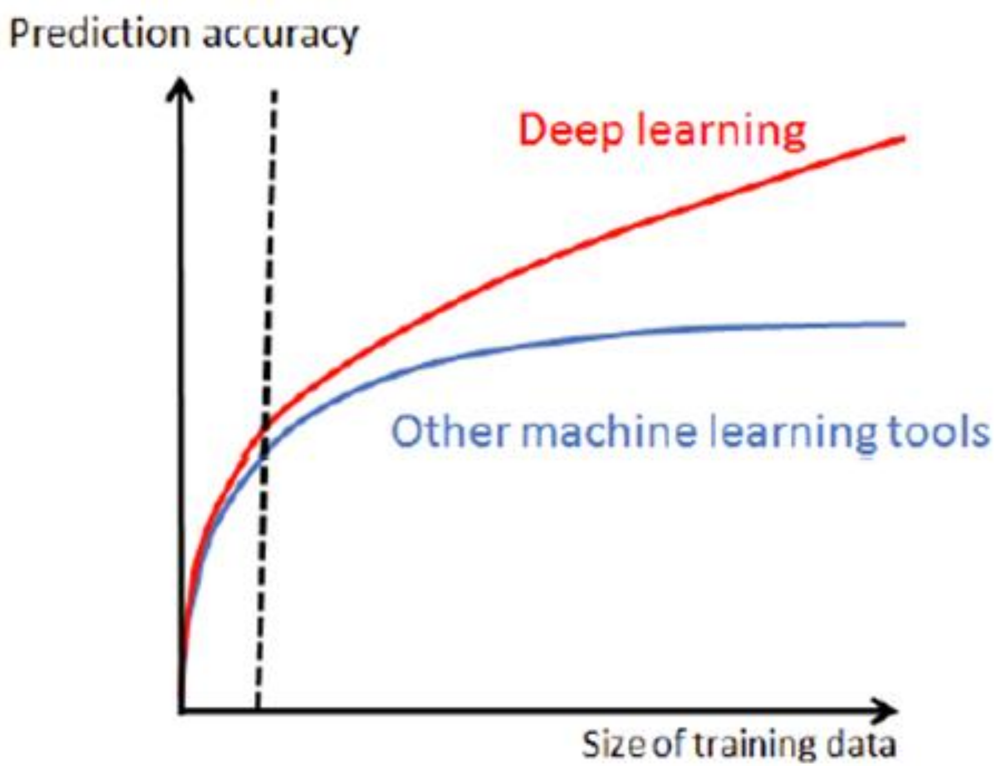
$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Deep Learning

- Machine learning with small data:
  - Overfitting, reducing model complexity
- Machine learning with big data:
  - Underfitting, increasing model complexity



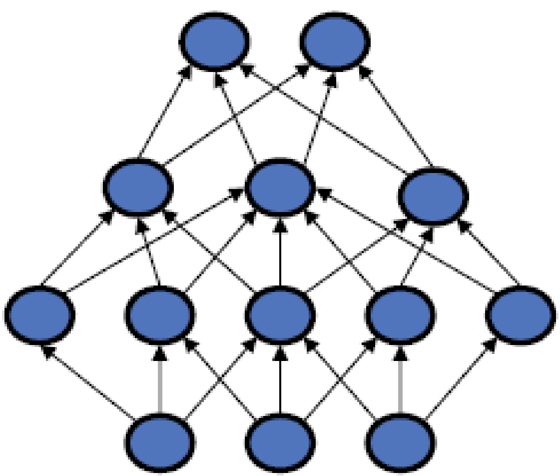
Deep Learning: History

Neural network
Back propagation

Nature

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagation errors. Nature, 1986.

1986



- Solve general learning problems
- Tied with biological system

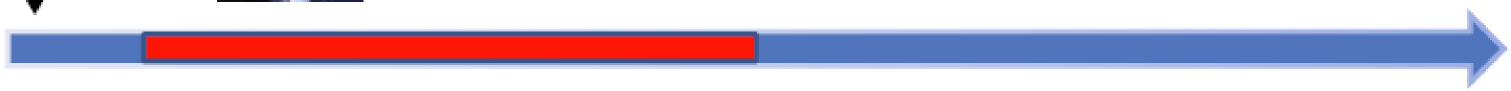
But it is given up…

- Hard to train
- Insufficient computational resources
- Small training sets
- Does not work well

Deep Learning: History

- Neural network
Back propagation

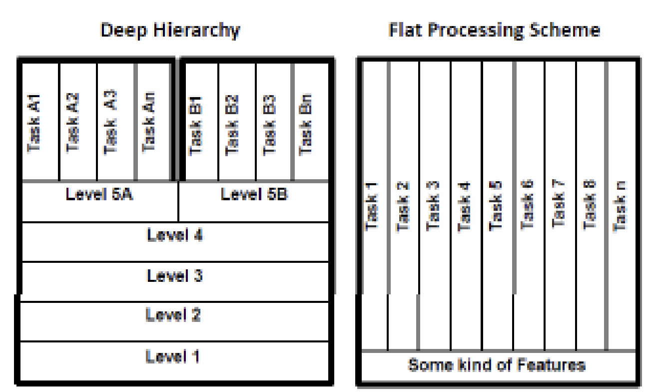
Nature



1986 2006

- SVM
- Boosting
- Decision tree
- KNN
- …

- Flat structures
- Loose tie with biological systems
- Specific methods for specific tasks
  - Hand crafted features (GMM-HMM, SIFT, LBP, HOG)



Kruger et al. TPAMI'13

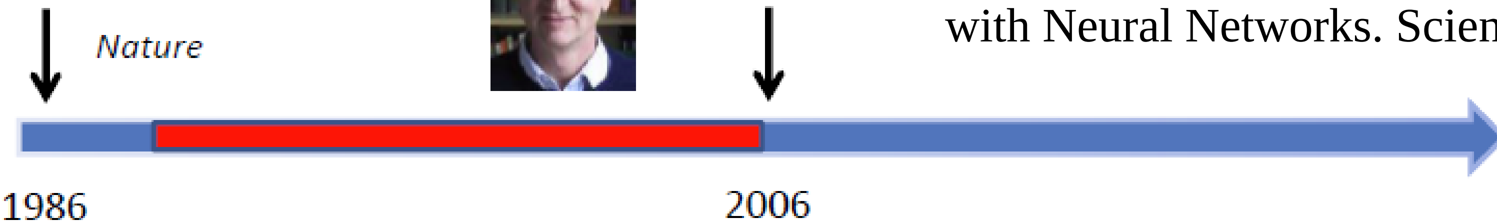
Deep Learning: History

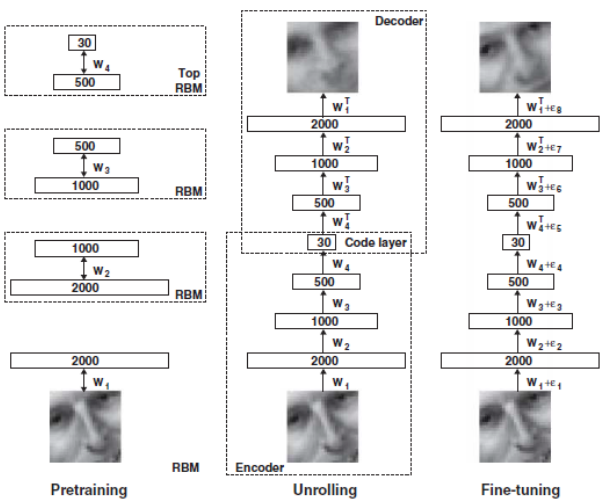
- Neural network
  Back propagation

Nature

Deep belief net
Science

G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. Science, 2006.

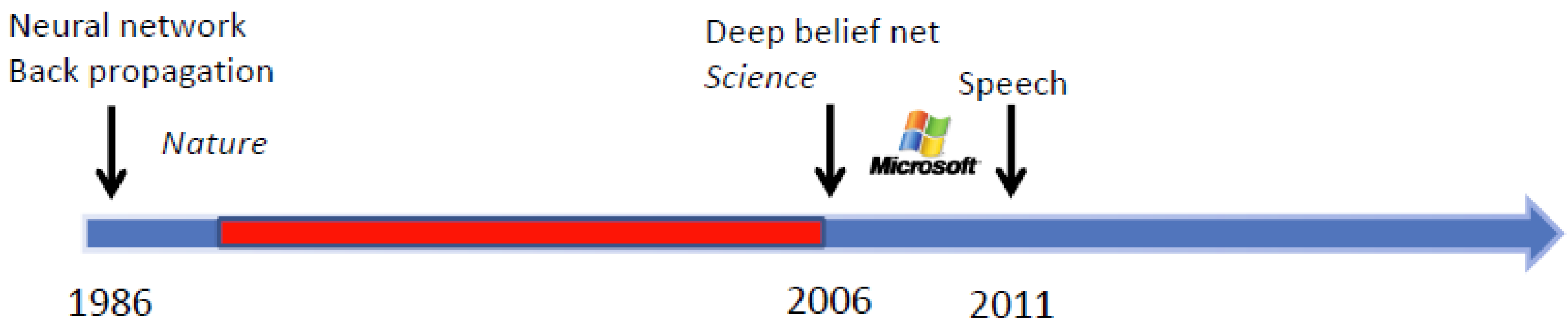




- Unsupervised & Layer-wised pre-training
- Better designs for modeling and training (normalization, nonlinearity, dropout)
- New development of computer architectures
  - GPU
  - Multi-core computer systems
- Large scale databases

Big Data !

Deep Learning: History



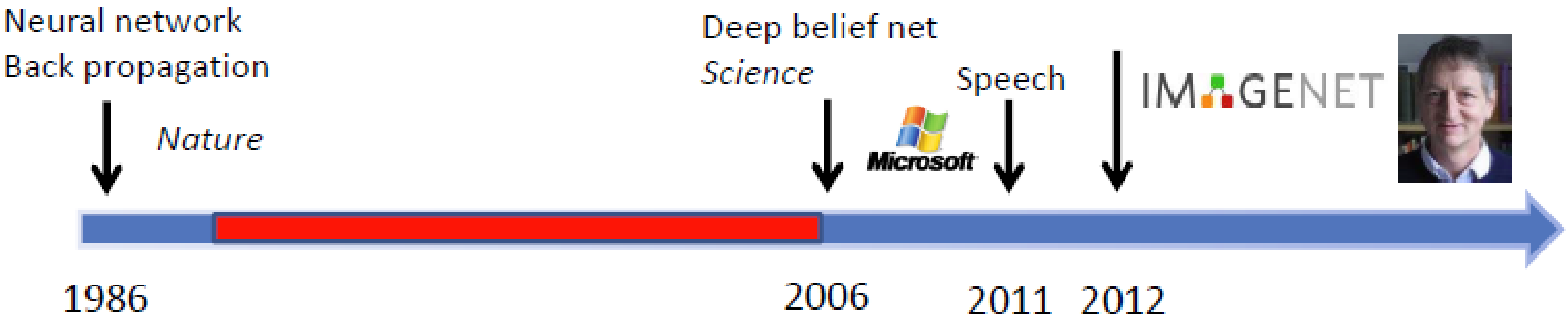
deep learning results

task	hours of training data	DNN-HMM	GMM-HMM with same data
Switchboard (test set 1)	309	18.5	27.4
Switchboard (test set 2)	309	16.1	23.6
English Broadcast News	50	17.5	18.8
Bing Voice Search (Sentence error rates)	24	30.4	36.2
Google Voice Input	5,870	12.3	
Youtube	1,400	47.6	52.3

Deep Networks Advance State of Art in Speech

Deep Learning leads to breakthrough in speech recognition at MSR.

Deep Learning: History

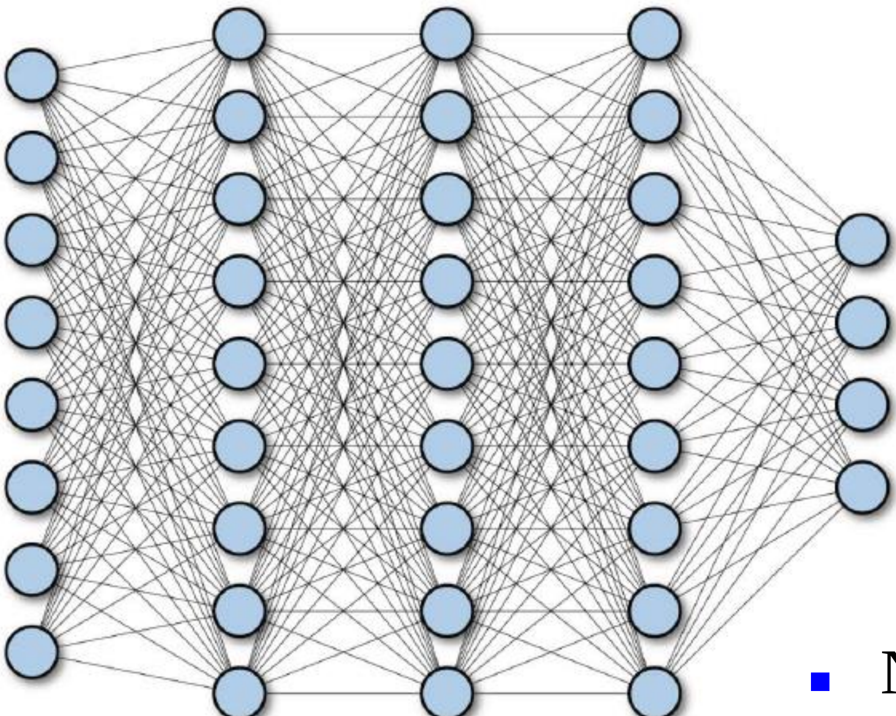
- 

Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

Object recognition over 1,000,000 images and 1,000 categories (2 GPU)

Convolutional Neural Network

- Fully connected neural network

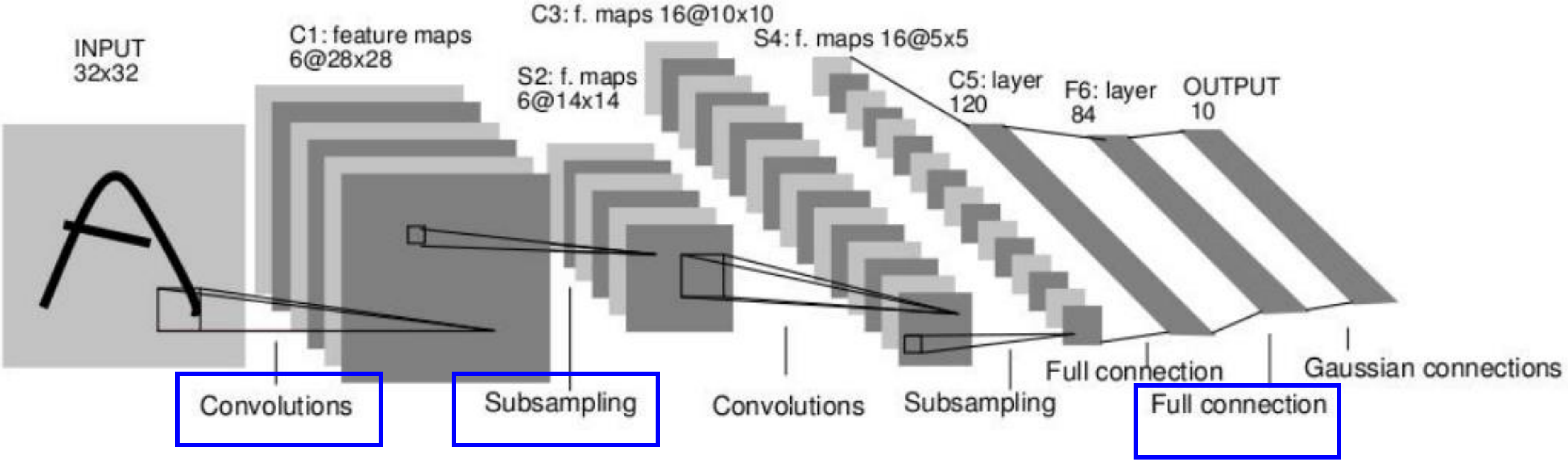


- Input:
  2D image
  ⇩
  Vector of pixel values

- No spatial information!
- Too many parameters!

Convolutional Neural Network

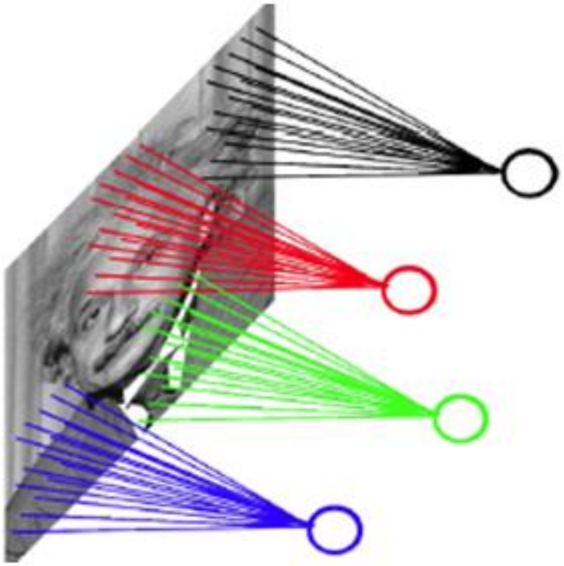
- LeNet-5



Y. Lecun, et al. Gradient-Based Learning Applied to Document Recognition, Proc. IEEE 86(11): 2278–2324, 1998.

Convolutional Neural Network

- The convolutional operation



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

image

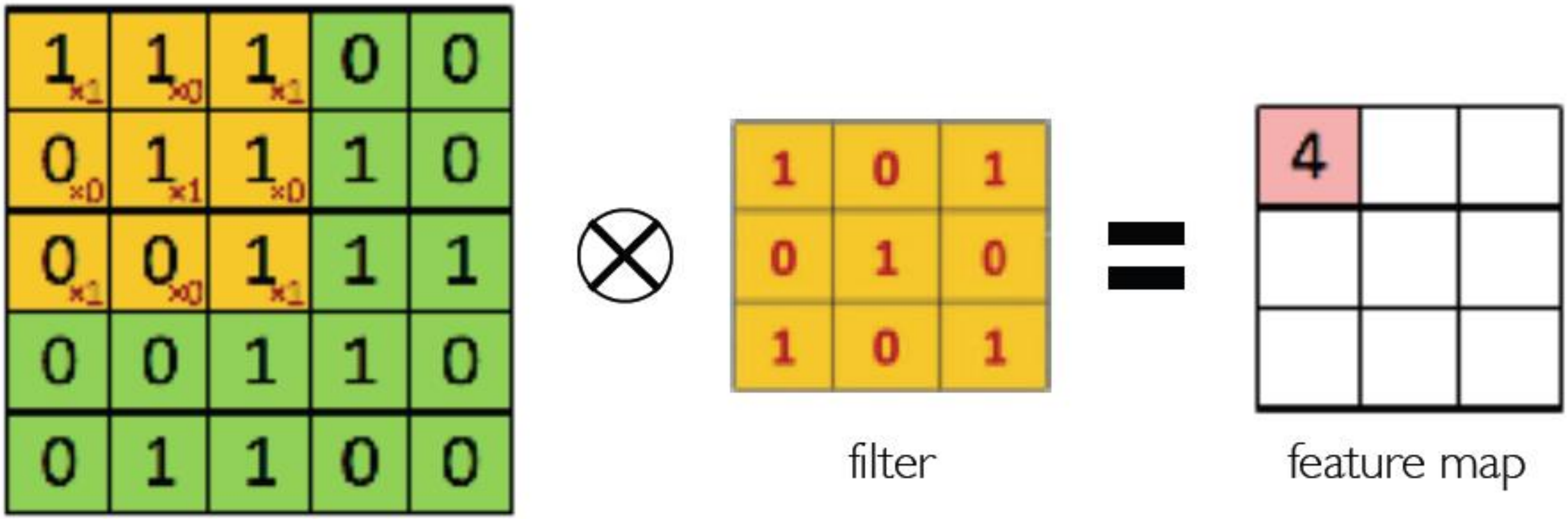
$\otimes$

1	0	1
0	1	0
1	0	1

filter

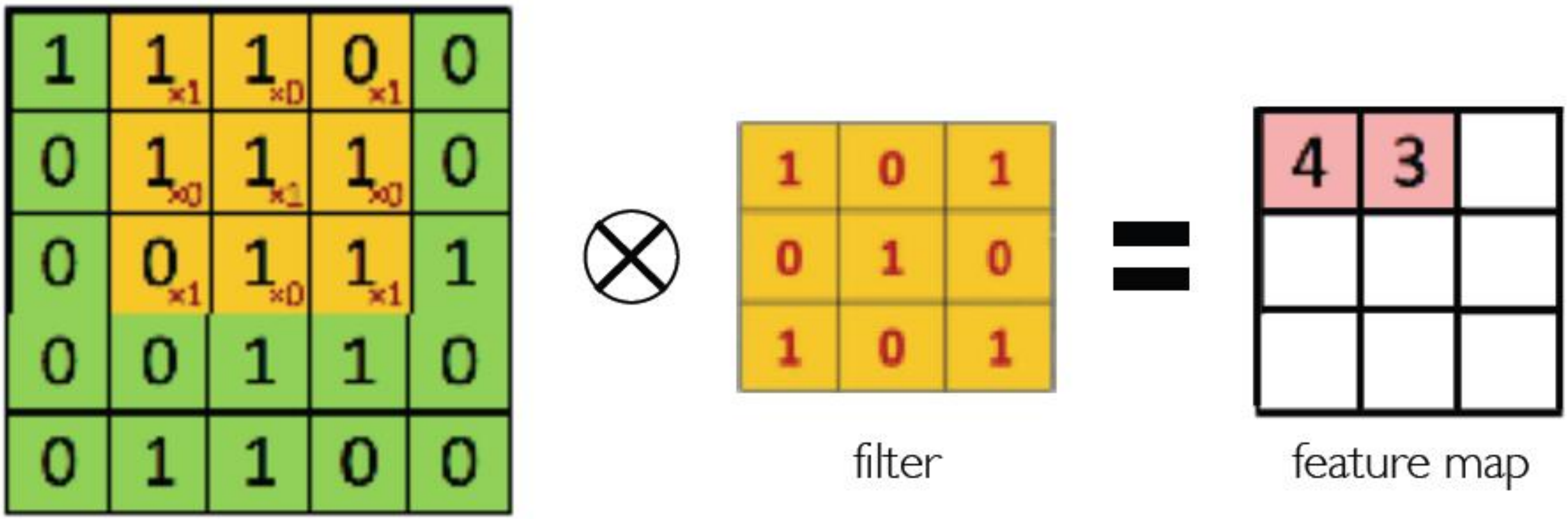
Convolutional Neural Network

- The convolutional operation



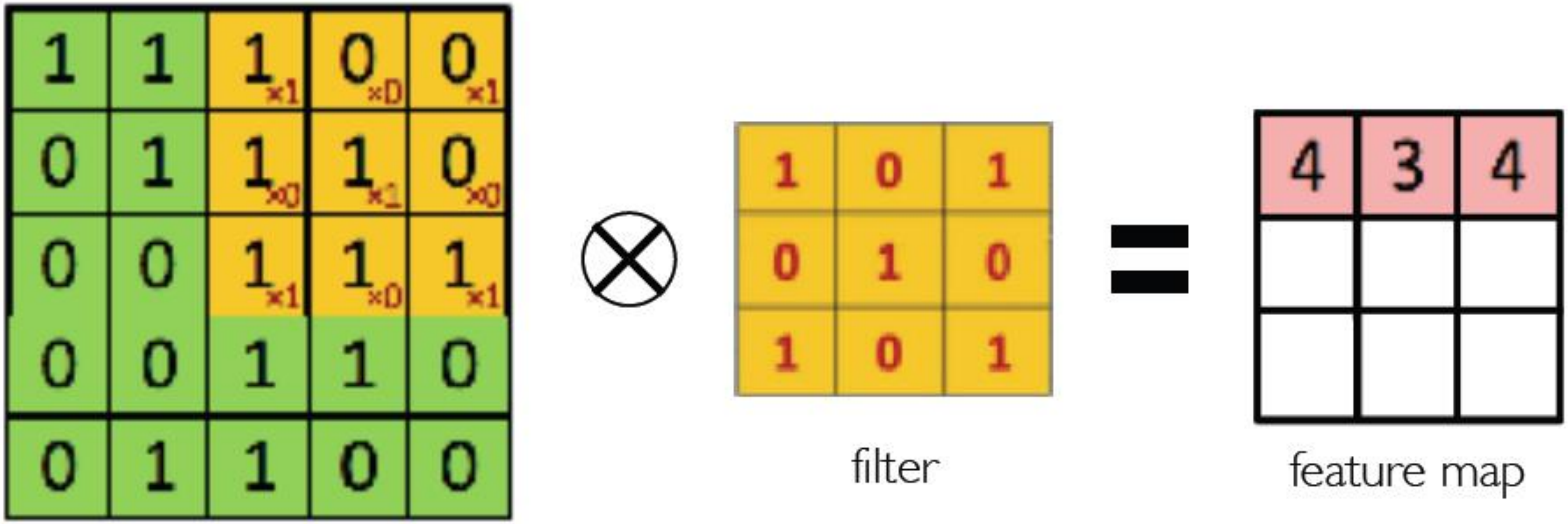
Convolutional Neural Network

- The convolutional operation



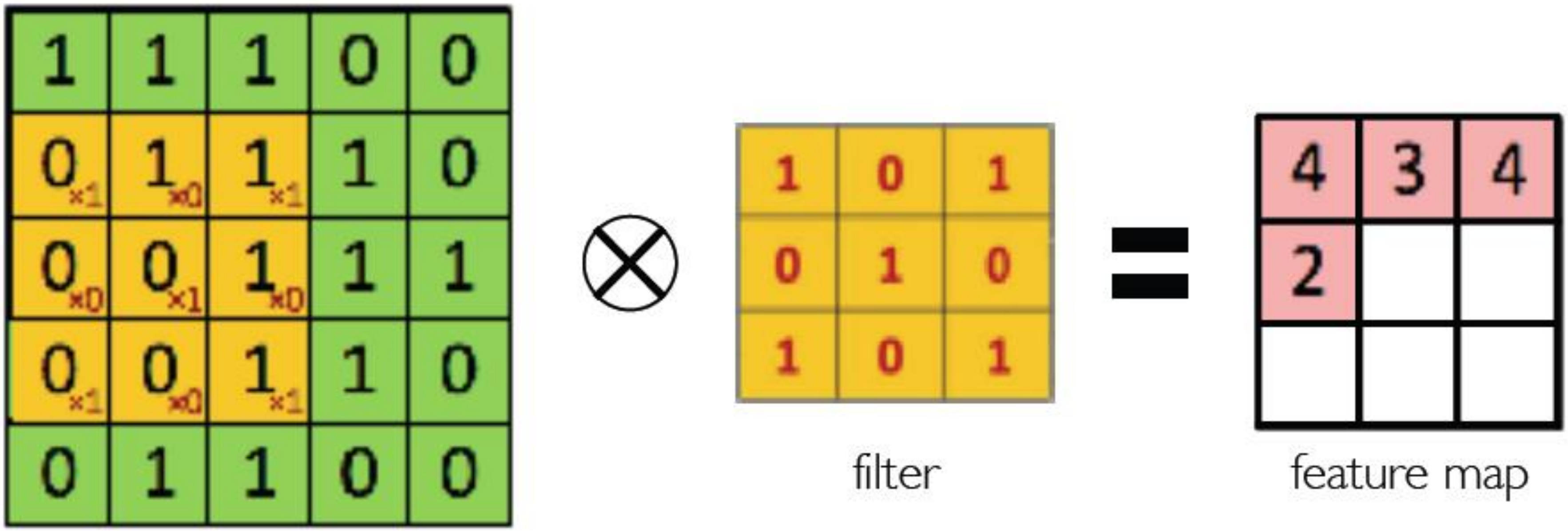
Convolutional Neural Network

- The convolutional operation



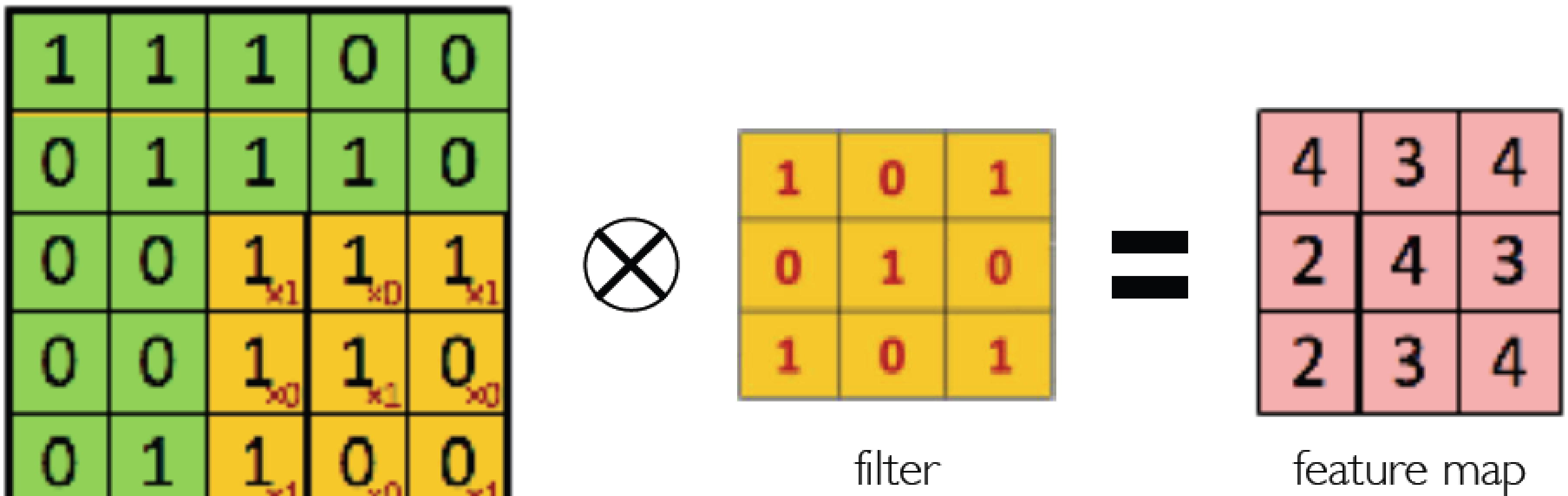
Convolutional Neural Network

- The convolutional operation



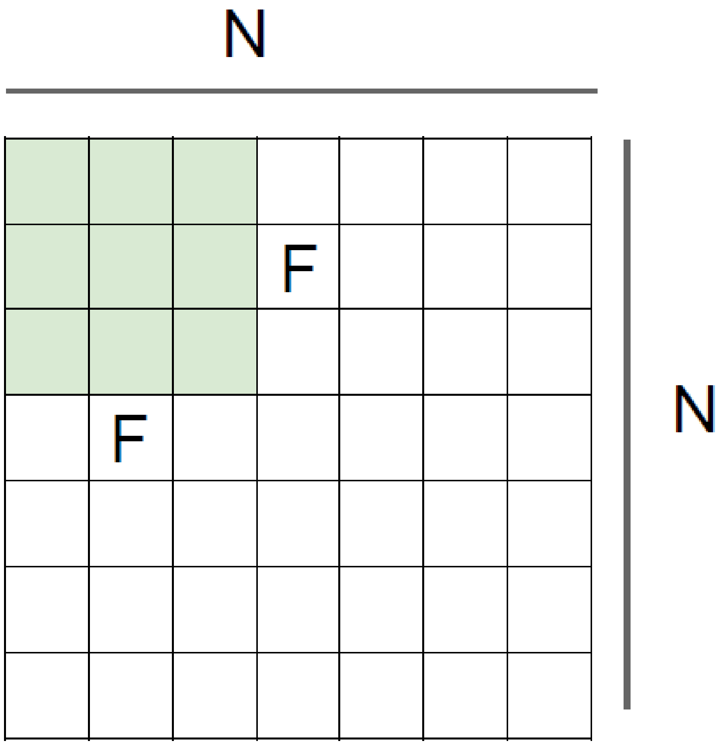
Convolutional Neural Network

- The convolutional operation



Convolutional Neural Network

- The convolutional operation



Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

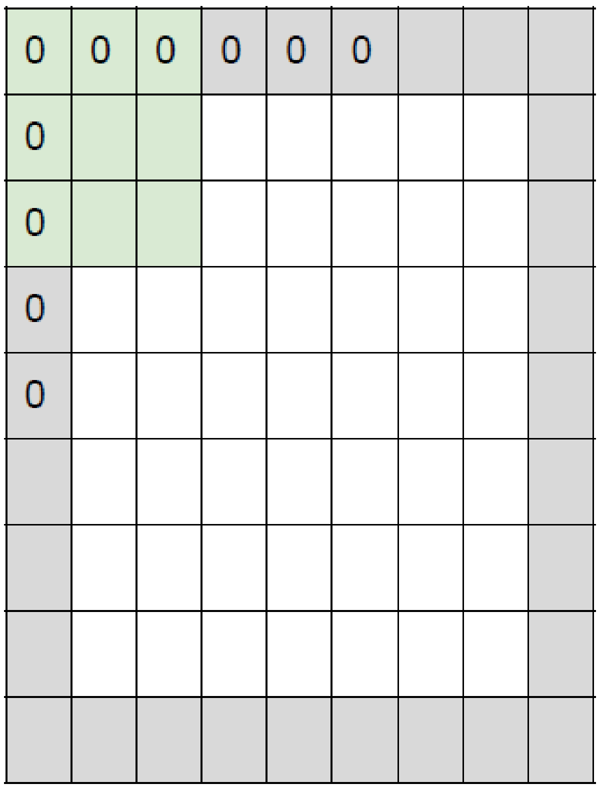
stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Convolutional Neural Network

- The convolutional layer

In practice: Common to zero pad the border

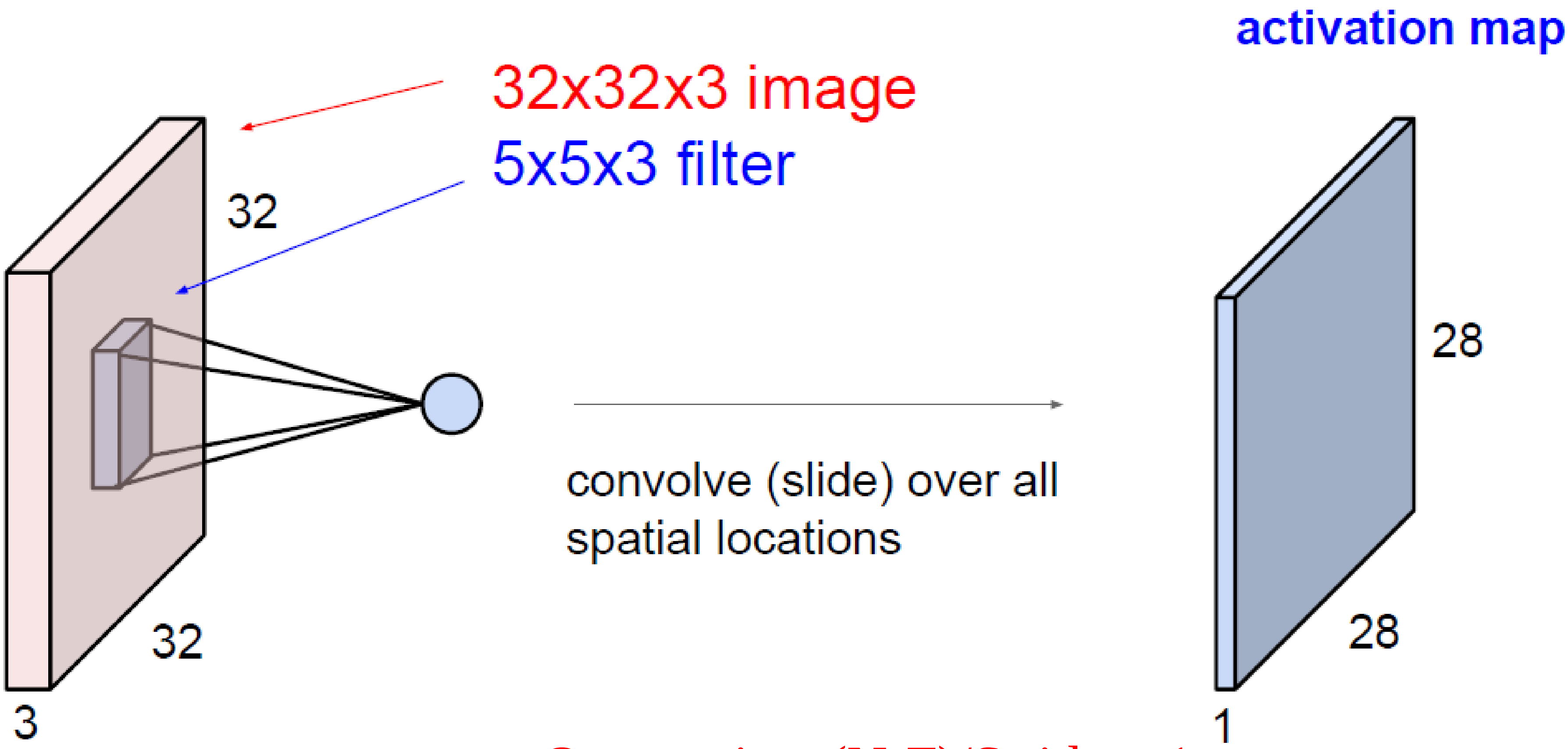


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Convolutional Neural Network

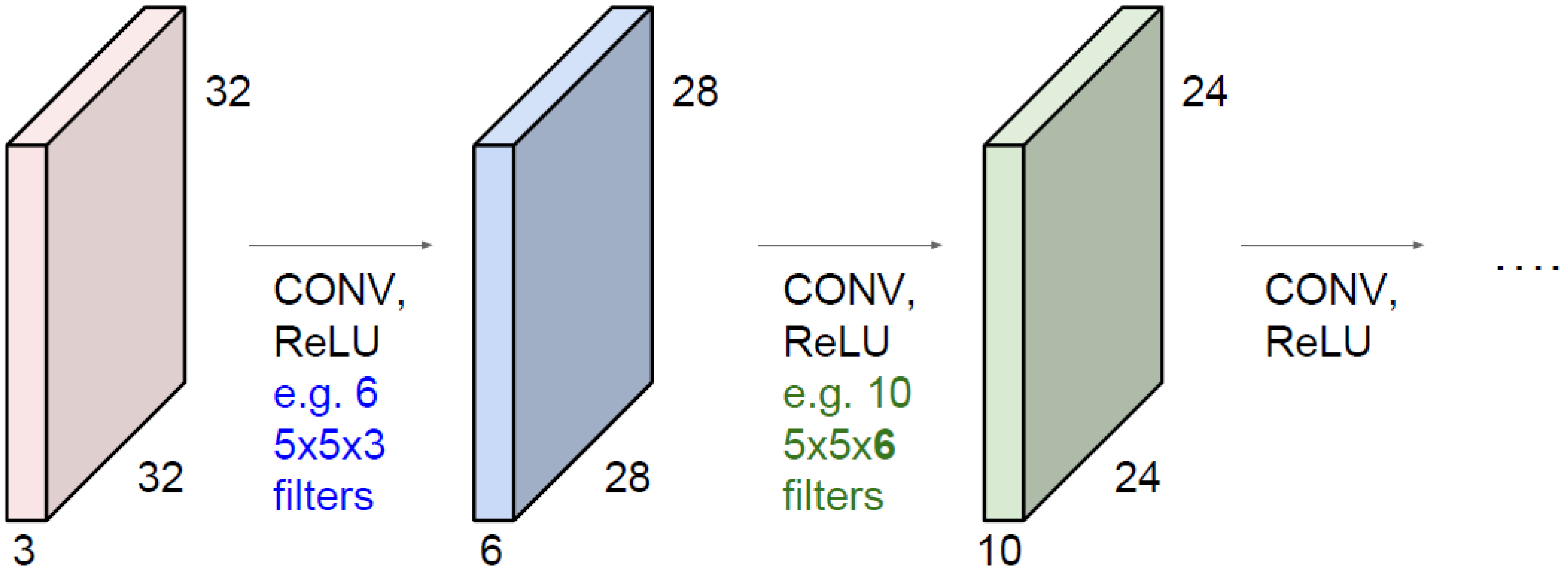
- The convolutional layer



- Output size: (N-F)/Stride + 1
- Zero padding
- Receptive field

Convolutional Neural Network

- The convolutional layer



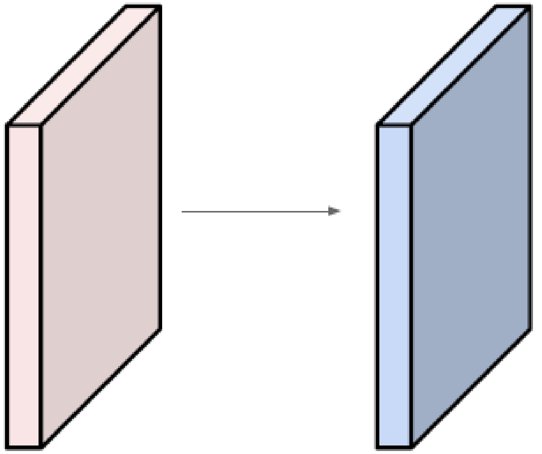
Convolutional Neural Network

- The convolutional layer

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

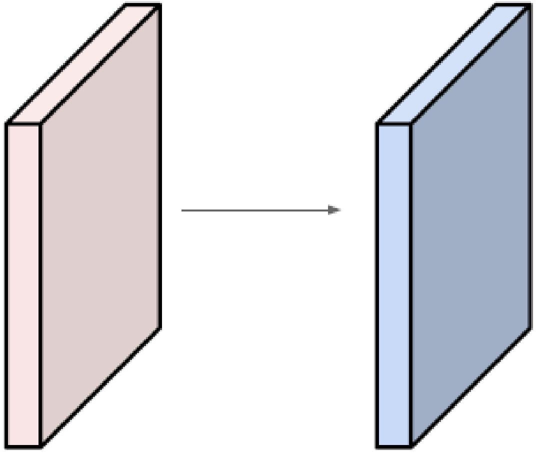
Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



Convolutional Neural Network

- The convolutional layer

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

Convolutional Neural Network

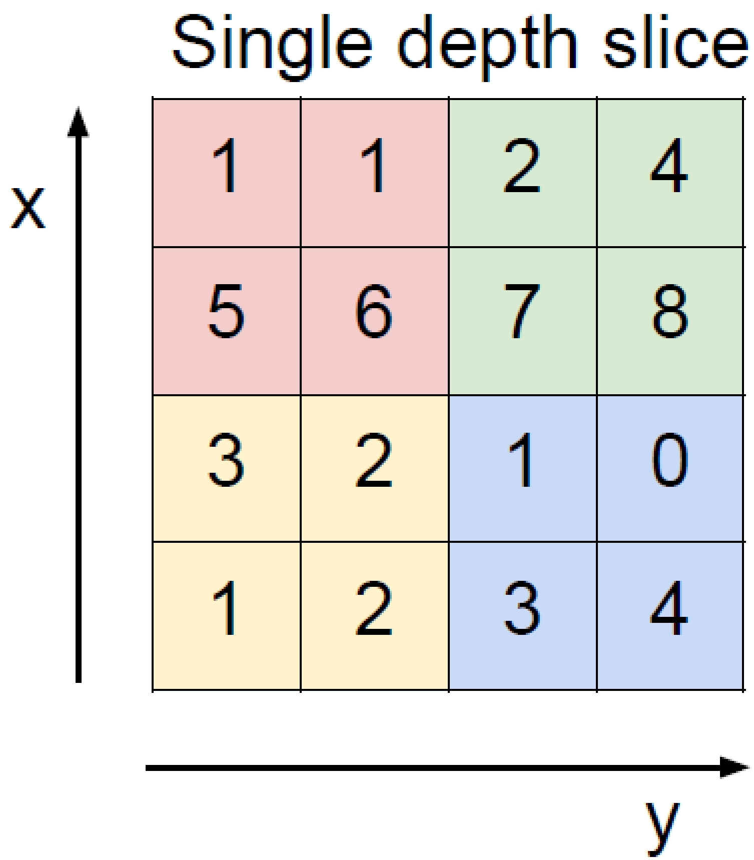
- The convolutional layer

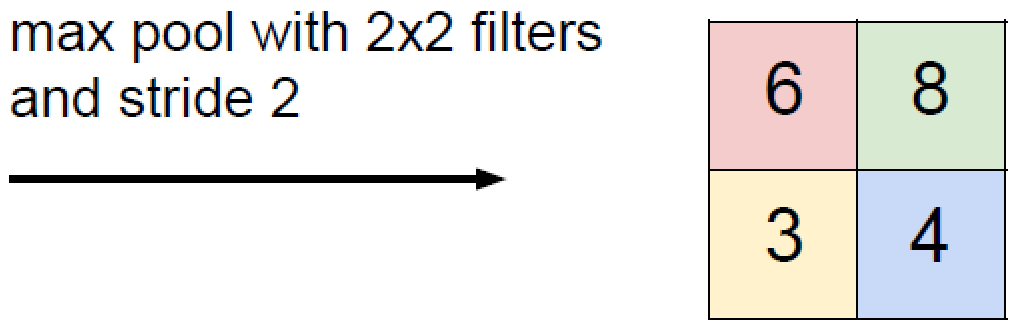
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolutional Neural Network

- The pooling layer





- Reduce dimensionality
- Preserve spatial variance
- Operates over each activation map independently

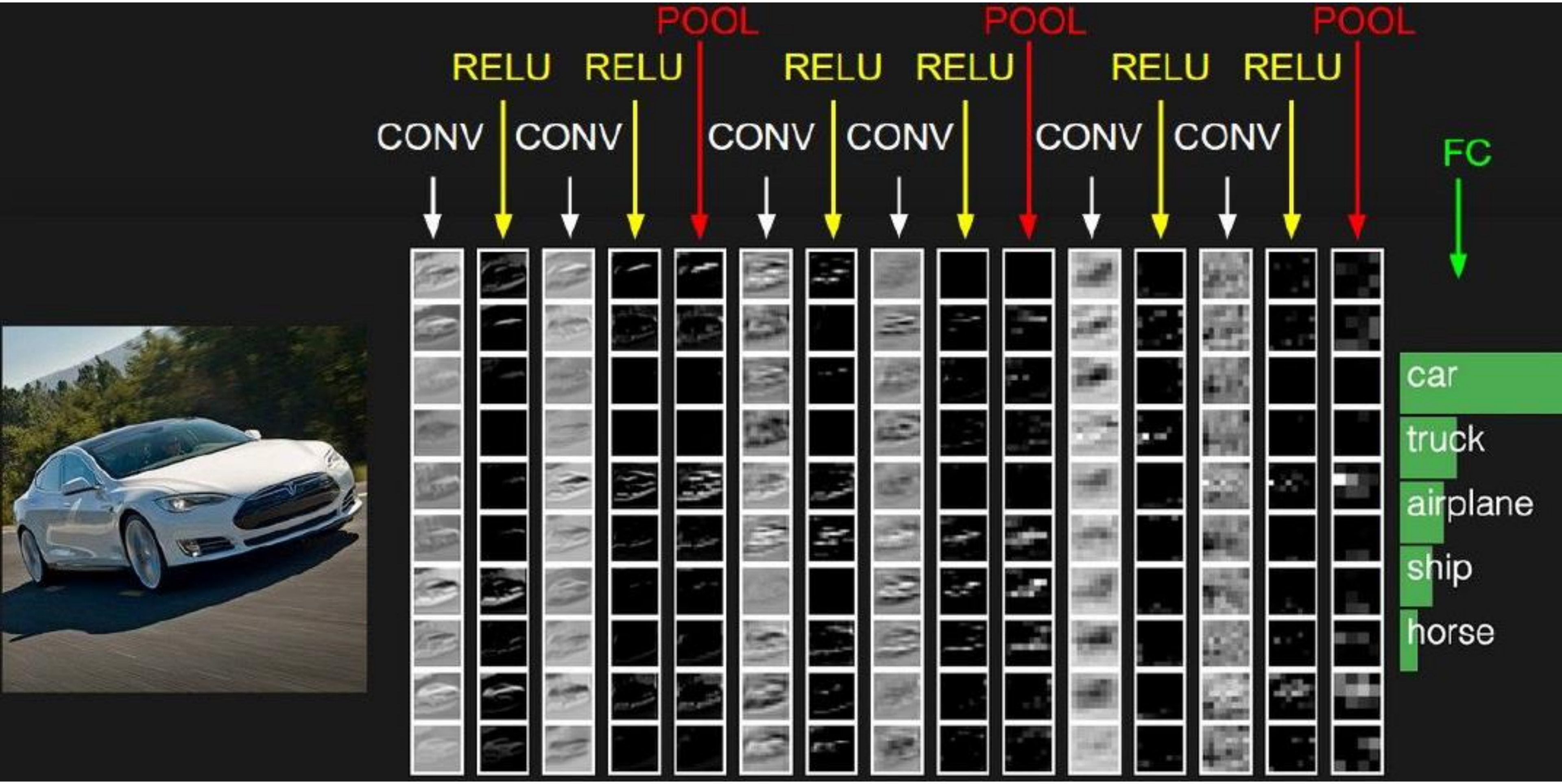
Convolutional Neural Network

- The pooling layer

  - Accepts a volume of size $W_1 \times H_1 \times D_1$
  - Requires three hyperparameters:
    - their spatial extent F,
    - the stride S,
  - Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F)/S + 1$
    - $H_2 = (H_1 - F)/S + 1$
    - $D_2 = D_1$
  - Introduces zero parameters since it computes a fixed function of the input
  - Note that it is not common to use zero-padding for Pooling layers

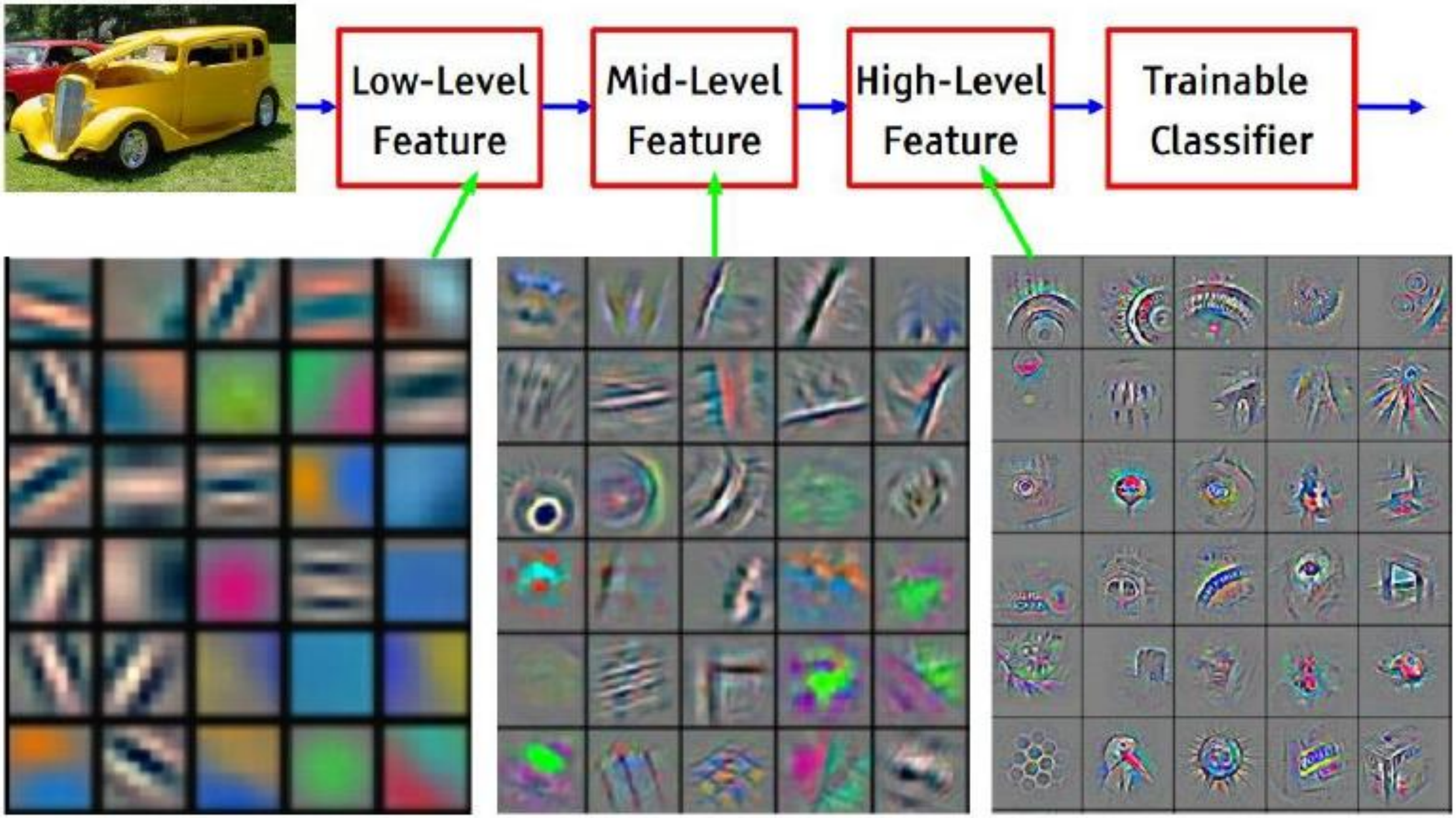
Convolutional Neural Network

- Fully connected layer



Convolutional Neural Network

- Feature visualization of CNN



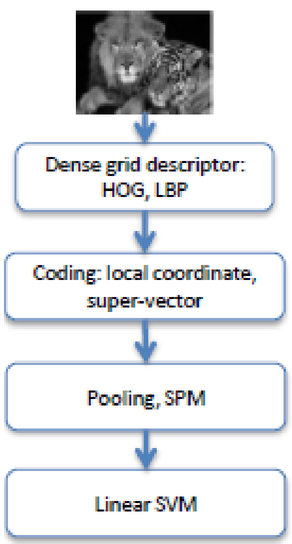
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional Neural Network

- IMAGENET Large Scale Visual Recognition Challenge

Year 2010

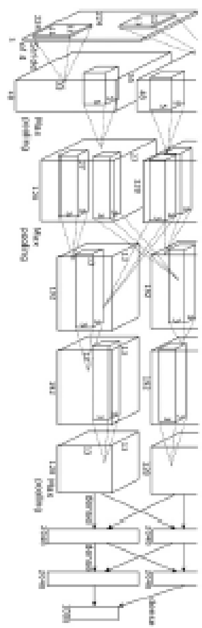
NEC-UIUC



[Lin CVPR 2011]

Year 2012

SuperVision

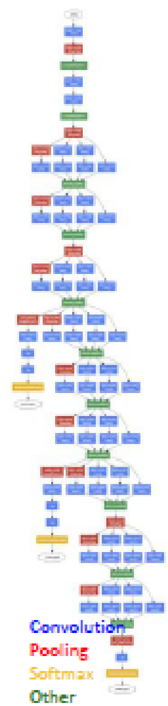


[Krizhevsky NIPS 2012]

AlexNet

Year 2014

GoogLeNet VGG

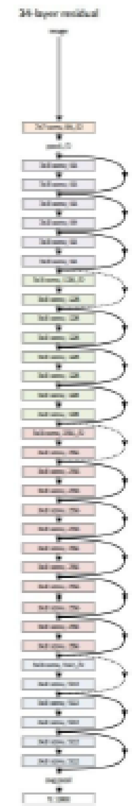


[Szegedy arxiv 2014] [Simonyan arxiv 2014]

GoogleNet VGGNet

Year 2015

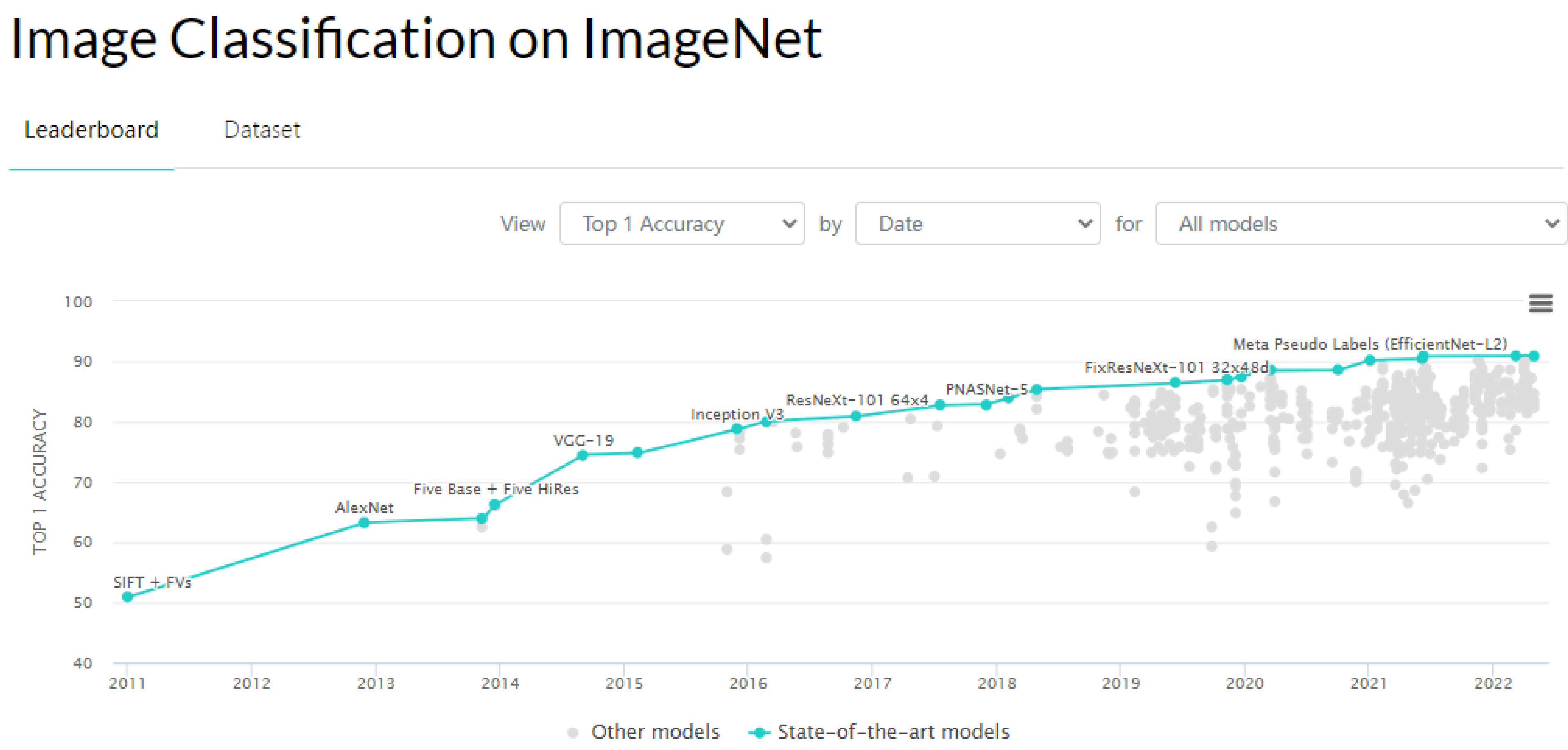
MSRA



ResNet

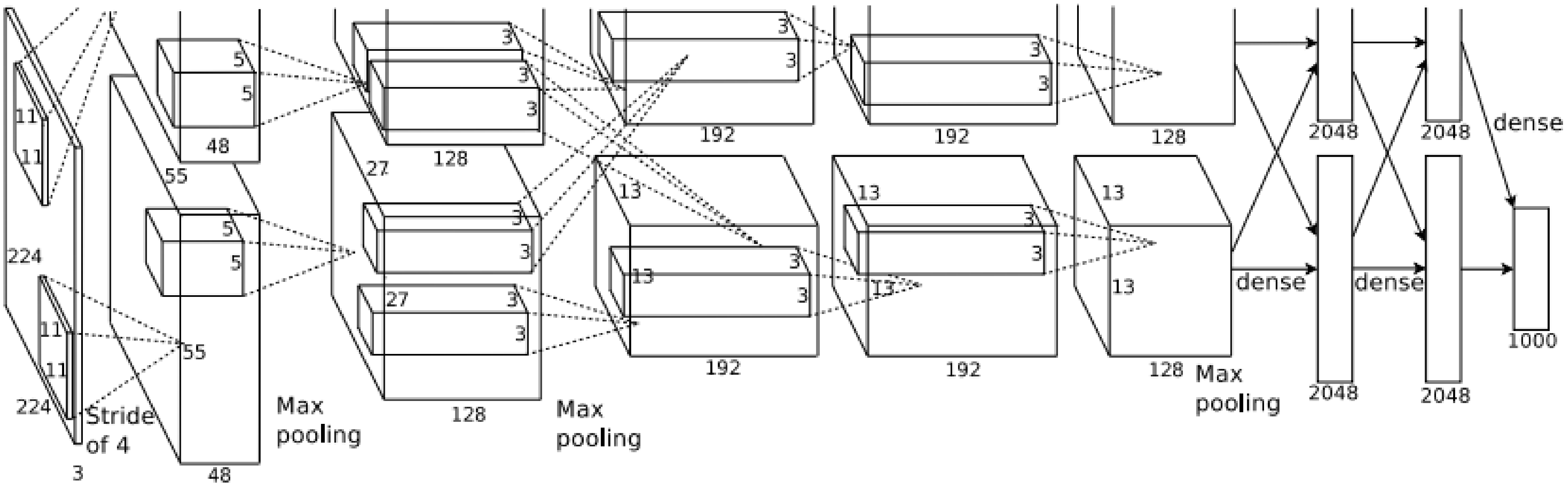
Convolutional Neural Network

■



Convolutional Neural Network

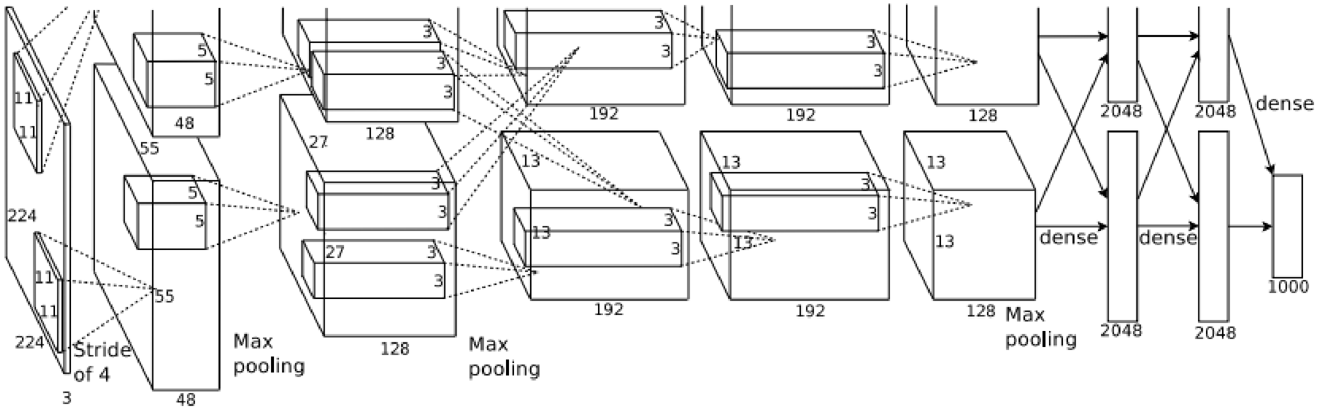
- AlexNet - 8 layers



A. Krizhevsky, H. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

Convolutional Neural Network

- AlexNet



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Convolutional Neural Network

- VGGNet

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

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition, ICLR 2015.

Convolutional Neural Network

- VGG16

(not counting biases)

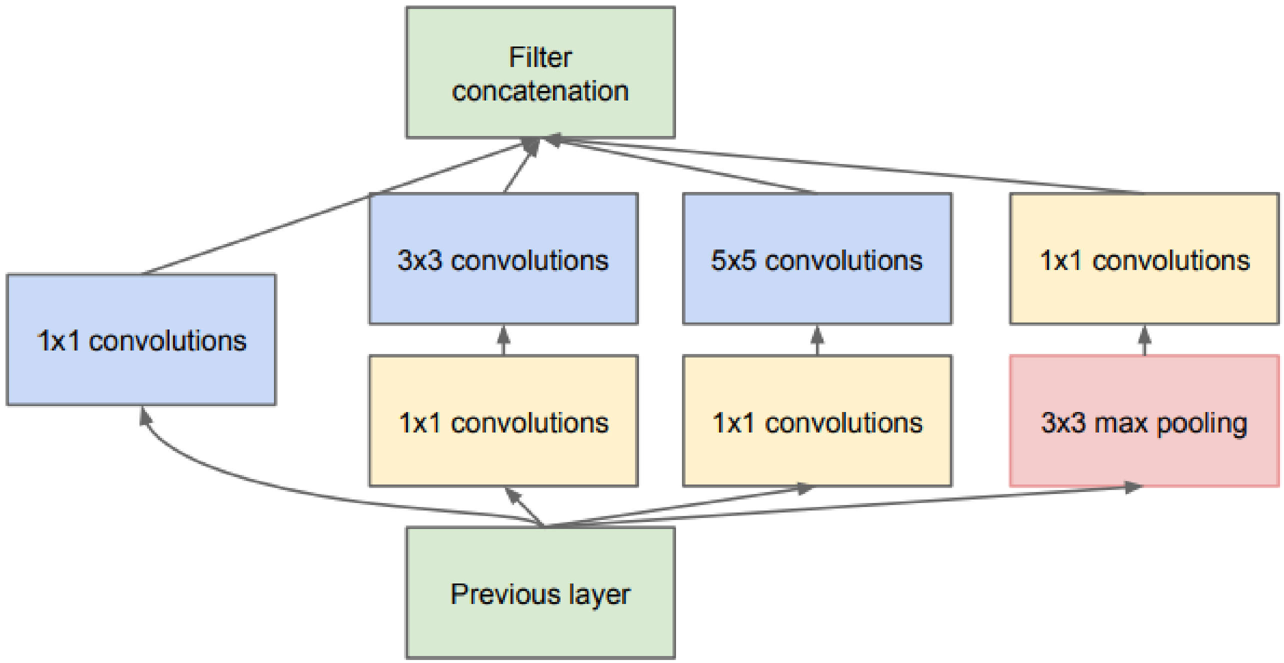
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

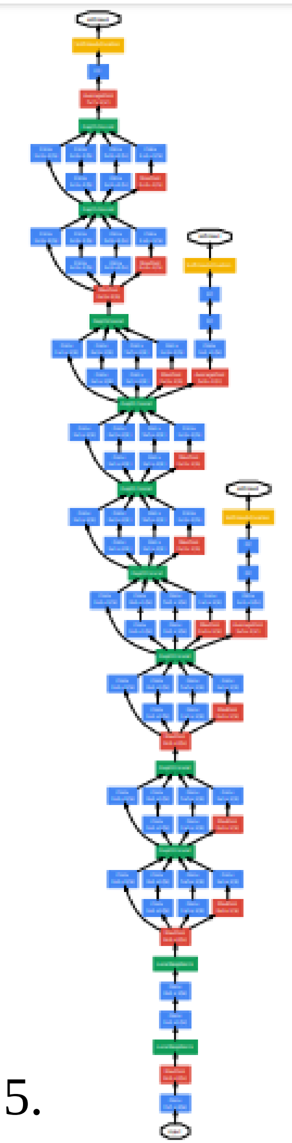
ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

Convolutional Neural Network

- GoogleNet – 22 layers



(b) Inception module with dimensionality reduction

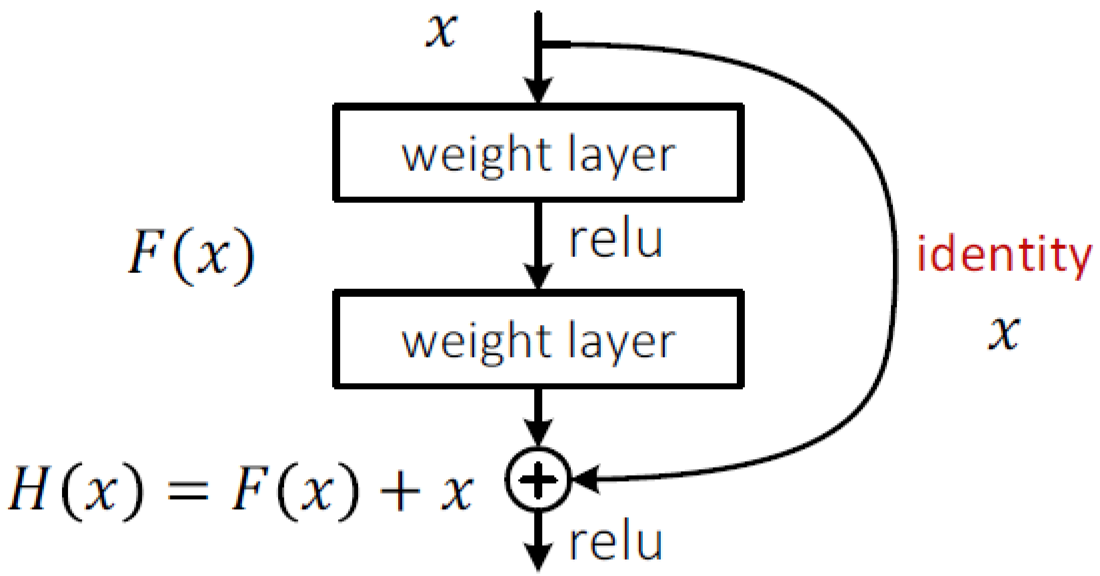


C. Szegedy, etal. Going Deeper with Convolutions, CVPR 2015.

Convolutional Neural Network

- ResNet – 152 layers

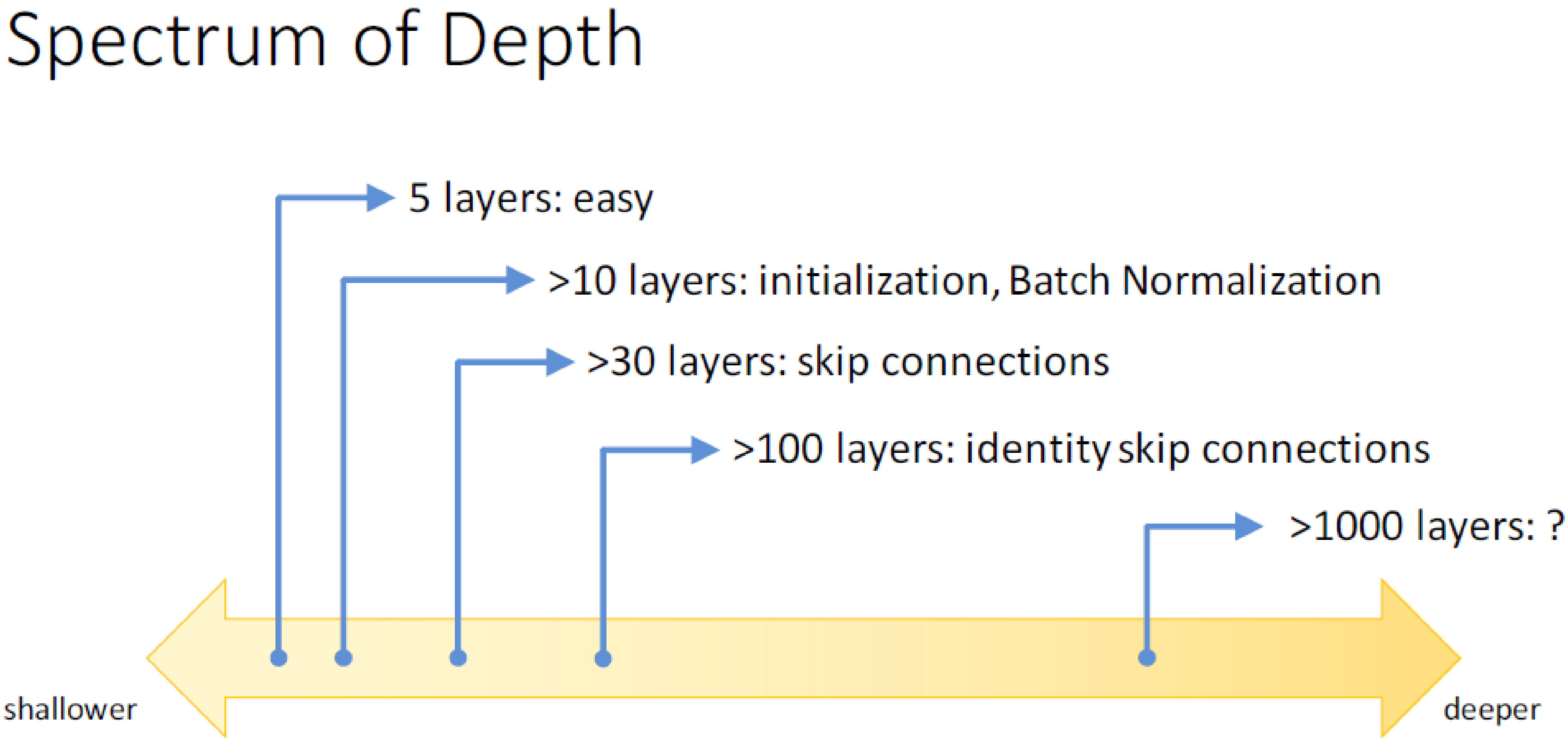
  - Identity skip connection



K. He, et al. Deep Residual Learning for Image Recognition, CVPR 2016

Convolutional Neural Network

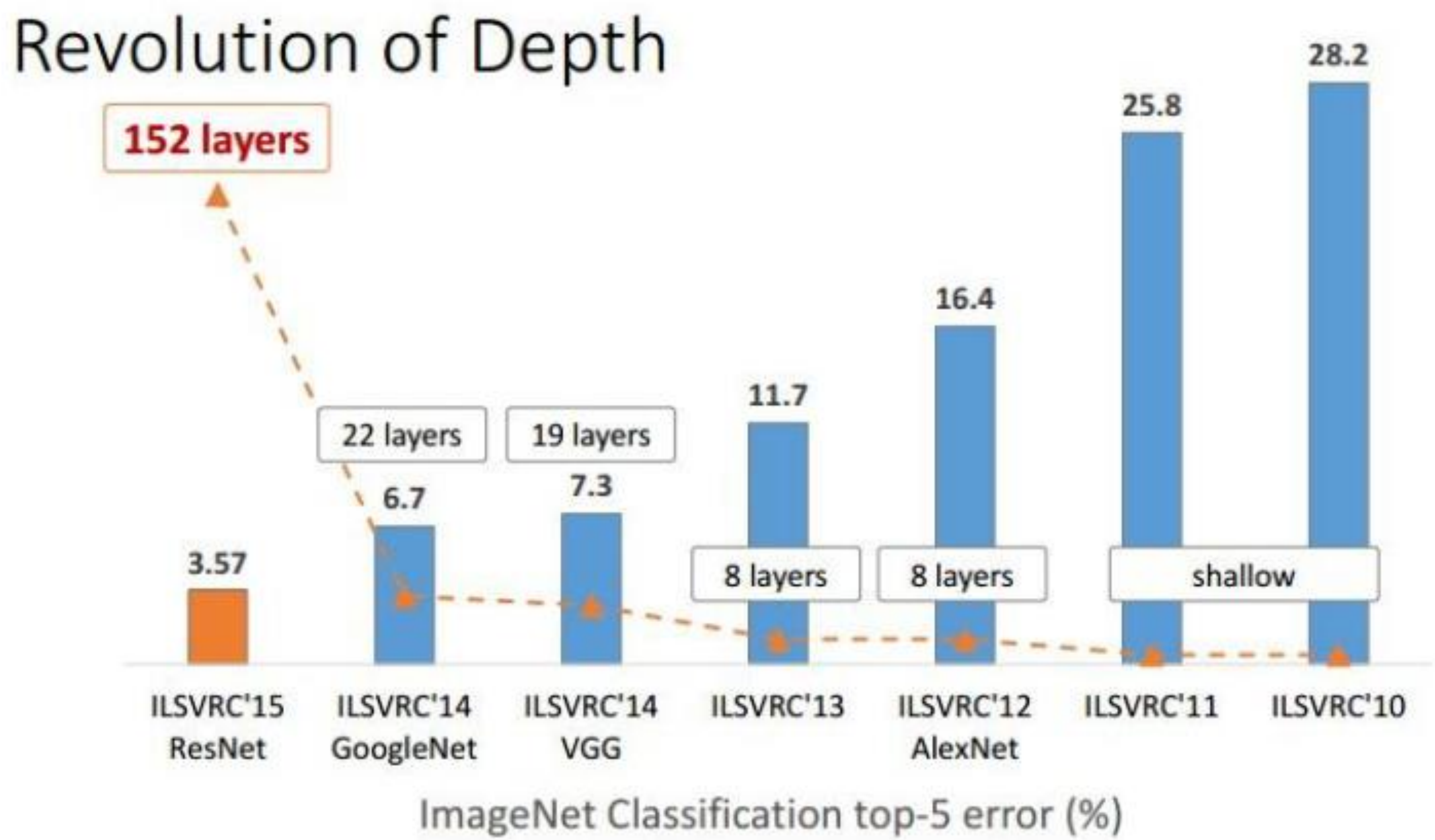
- ResNet – 152 layers



K. He, et al. Deep Residual Learning for Image Recognition, CVPR 2016

Convolutional Neural Network

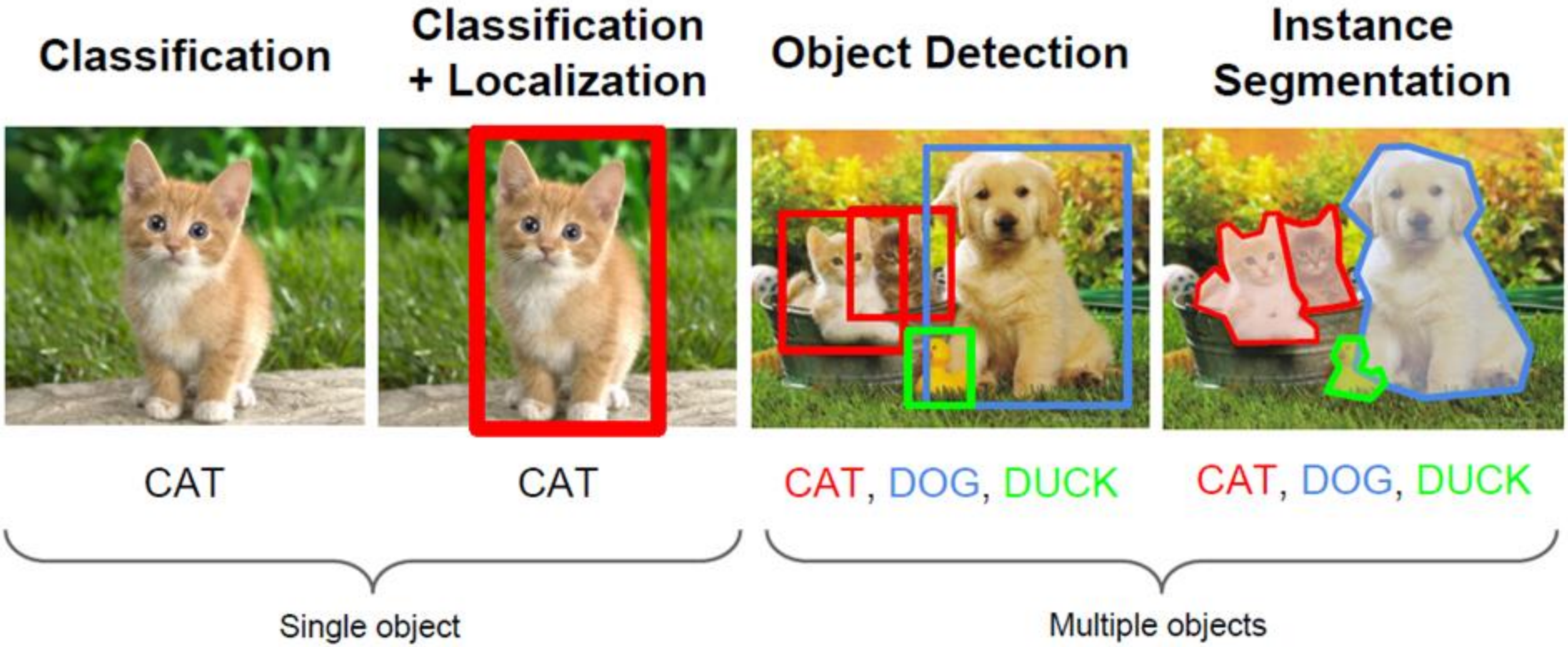
- ResNet – 152 layers



K. He, et al. Deep Residual Learning for Image Recognition, CVPR 2016

Convolutional Neural Network

- Applications



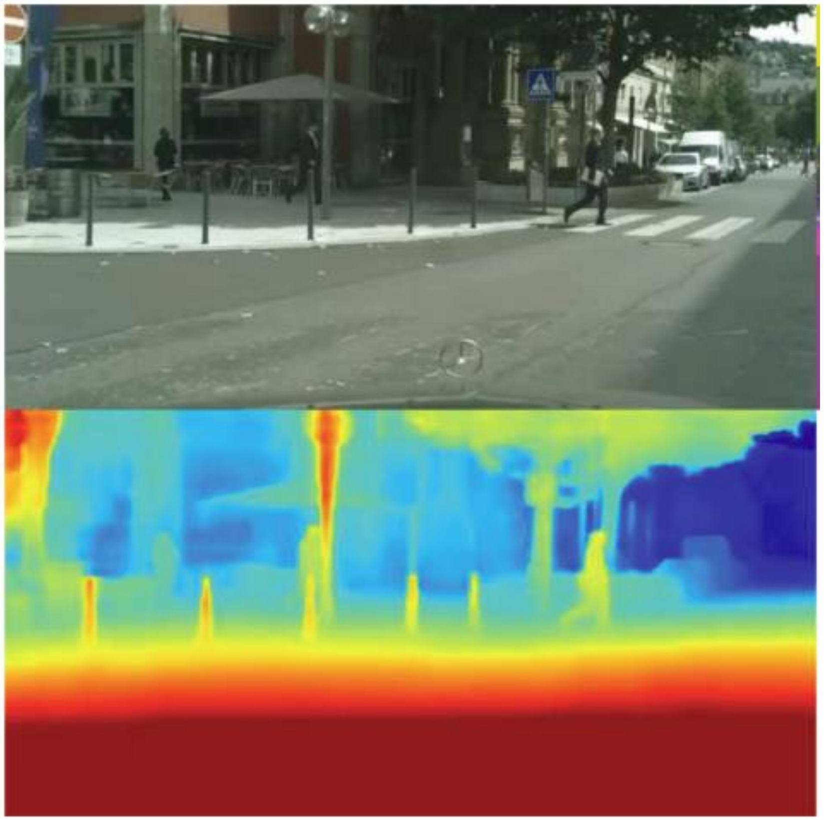
Convolutional Neural Network

- Applications

NeuralStyle

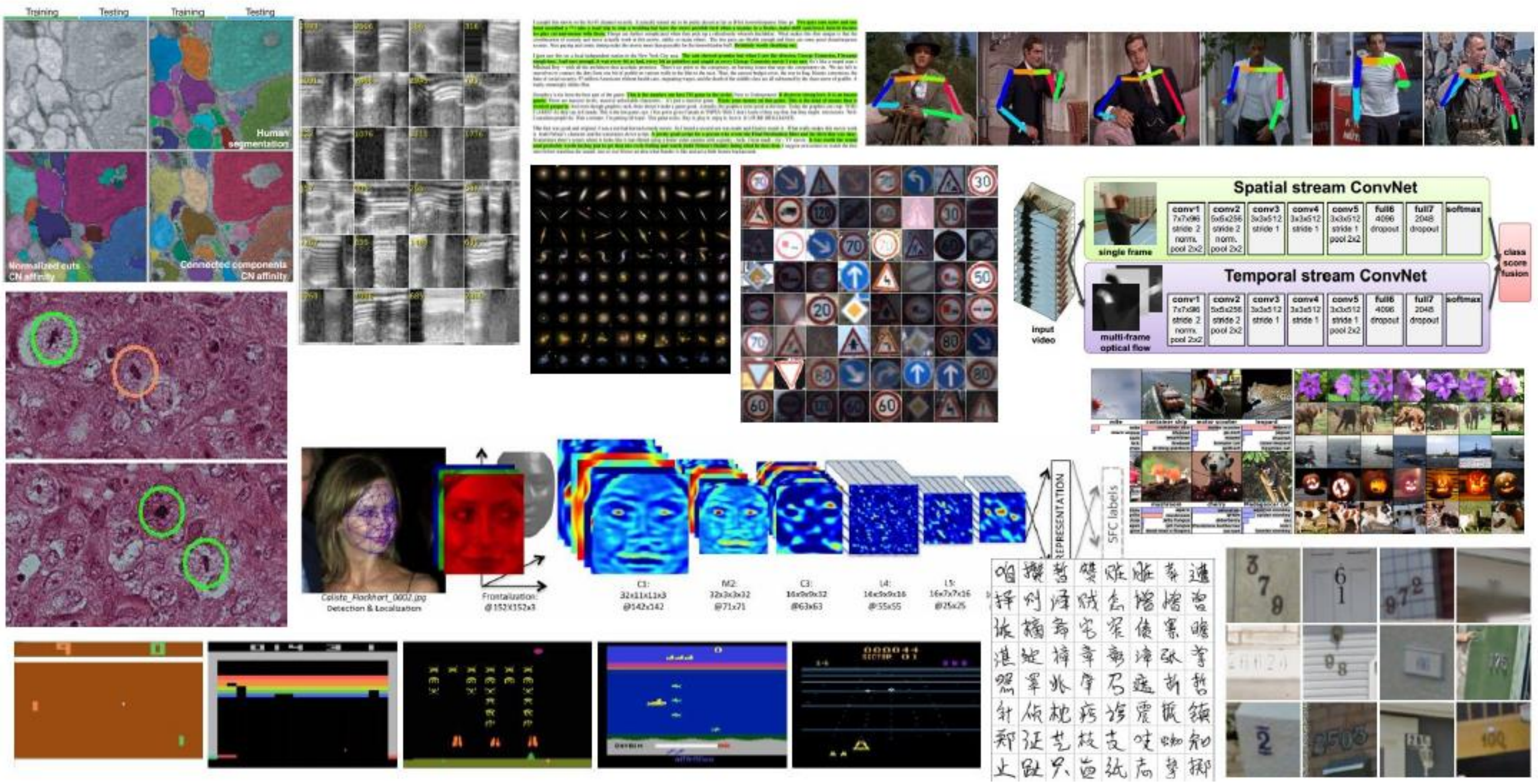
[A Neural Algorithm of Artistic Style by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, 2015]
good implementation by Justin in Torch:
https://github.com/jcjohnson/neural-style

Depth prediction



Convolutional Neural Network

- Applications

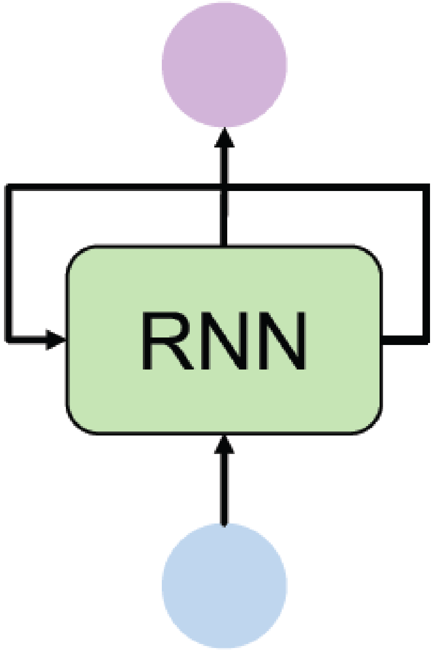


Recurrent Neural Network

- Sequence modeling

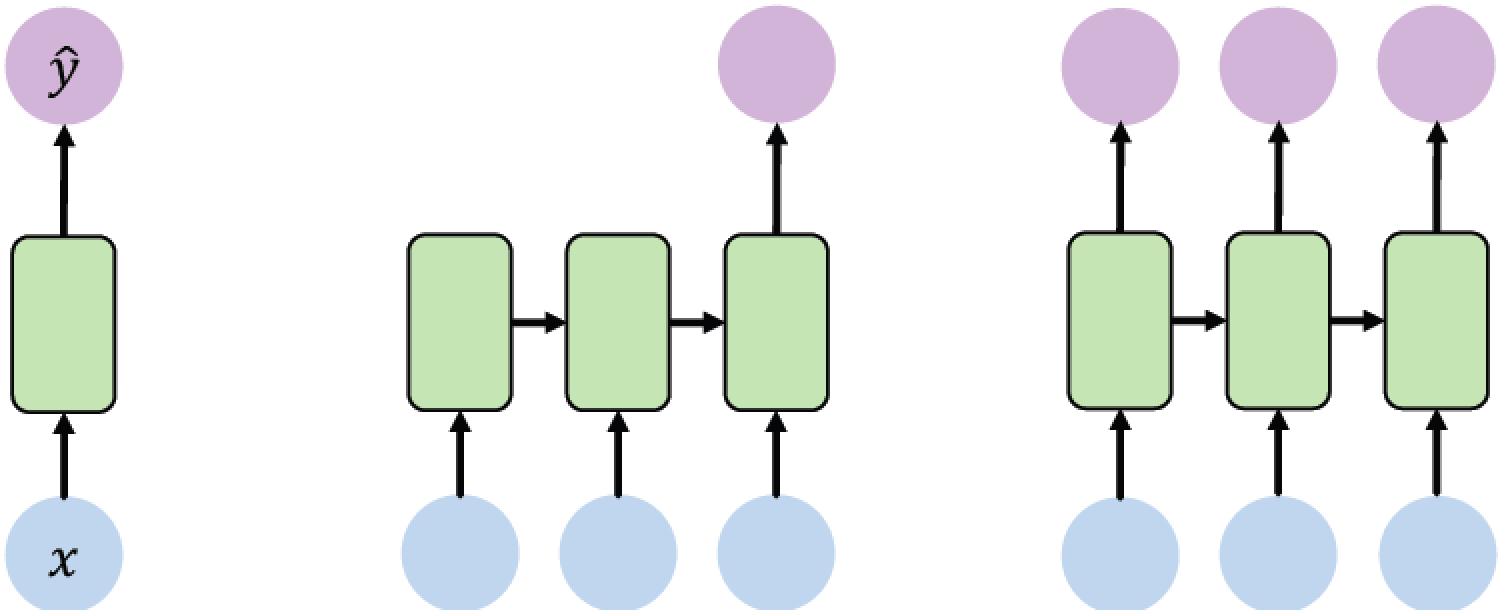
To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



Recurrent Neural Network

- Sequence modeling



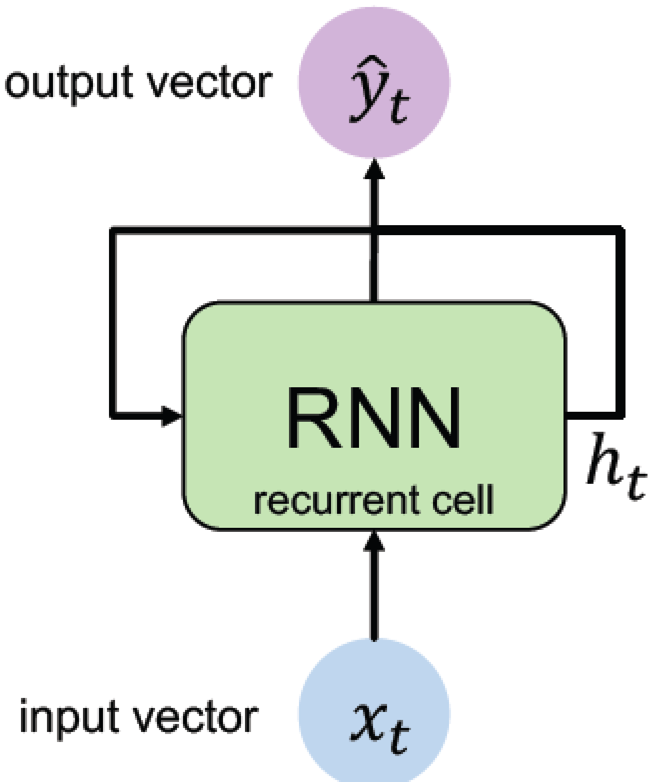
One to One
"Vanilla" neural network

Many to One
Sentiment Classification

Many to Many
Music Generation

Recurrent Neural Network

- RNN



Apply a **recurrence relation** at every time step to process a sequence:

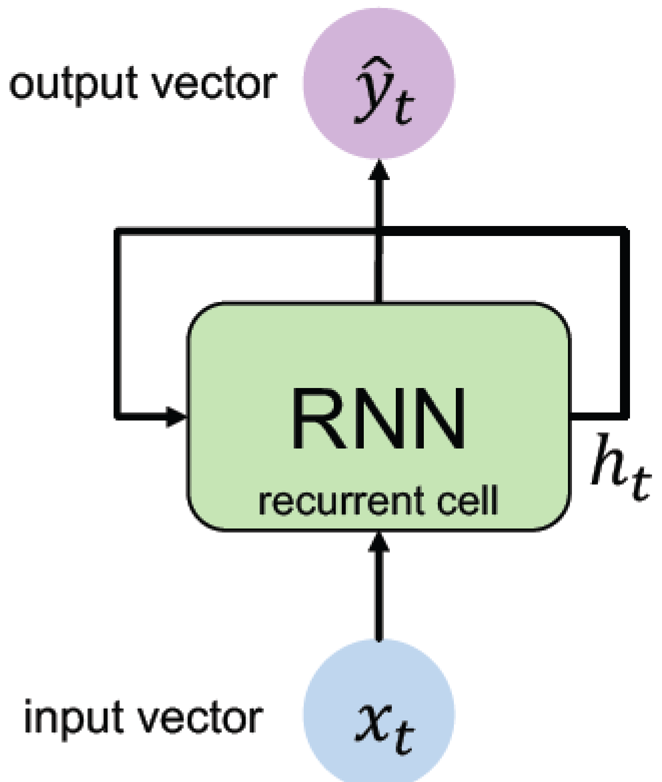
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state | function parameterized by W | old state | input vector at time step t

Note: the same function and set of parameters are used at every time step

Recurrent Neural Network

- RNN



Output Vector

$$\hat{y}_t = \boldsymbol{W_{hy}} h_t$$

Update Hidden State

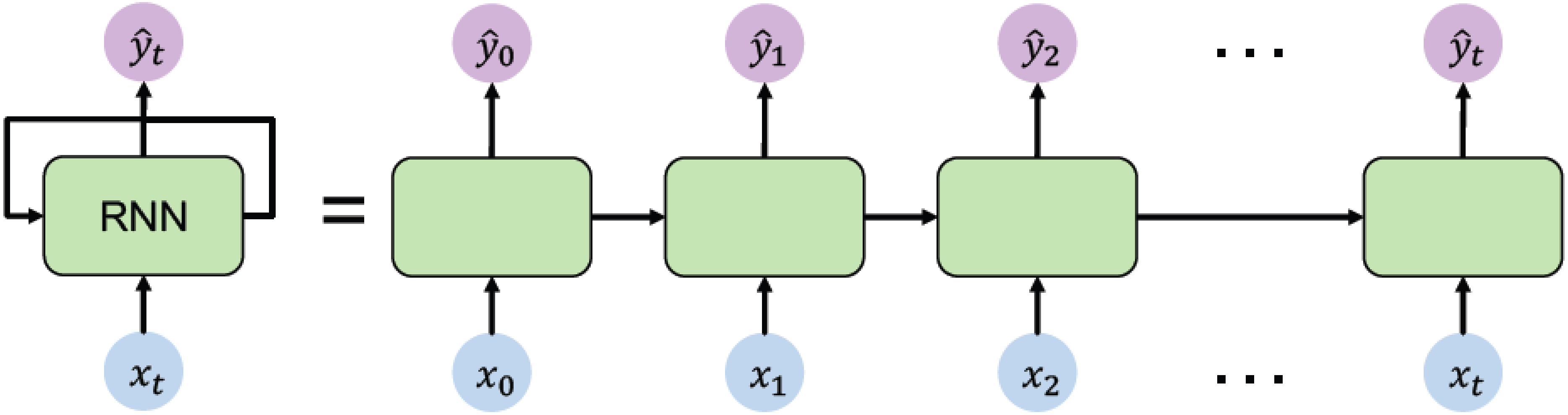
$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

$$\tanh z = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Input Vector

Recurrent Neural Network

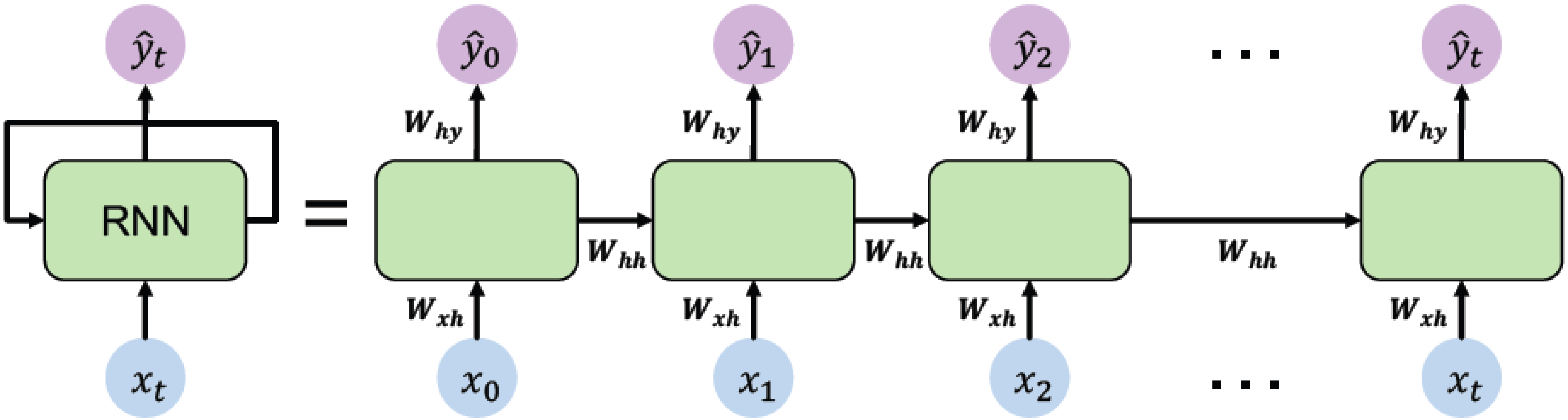
- RNN



Recurrent Neural Network

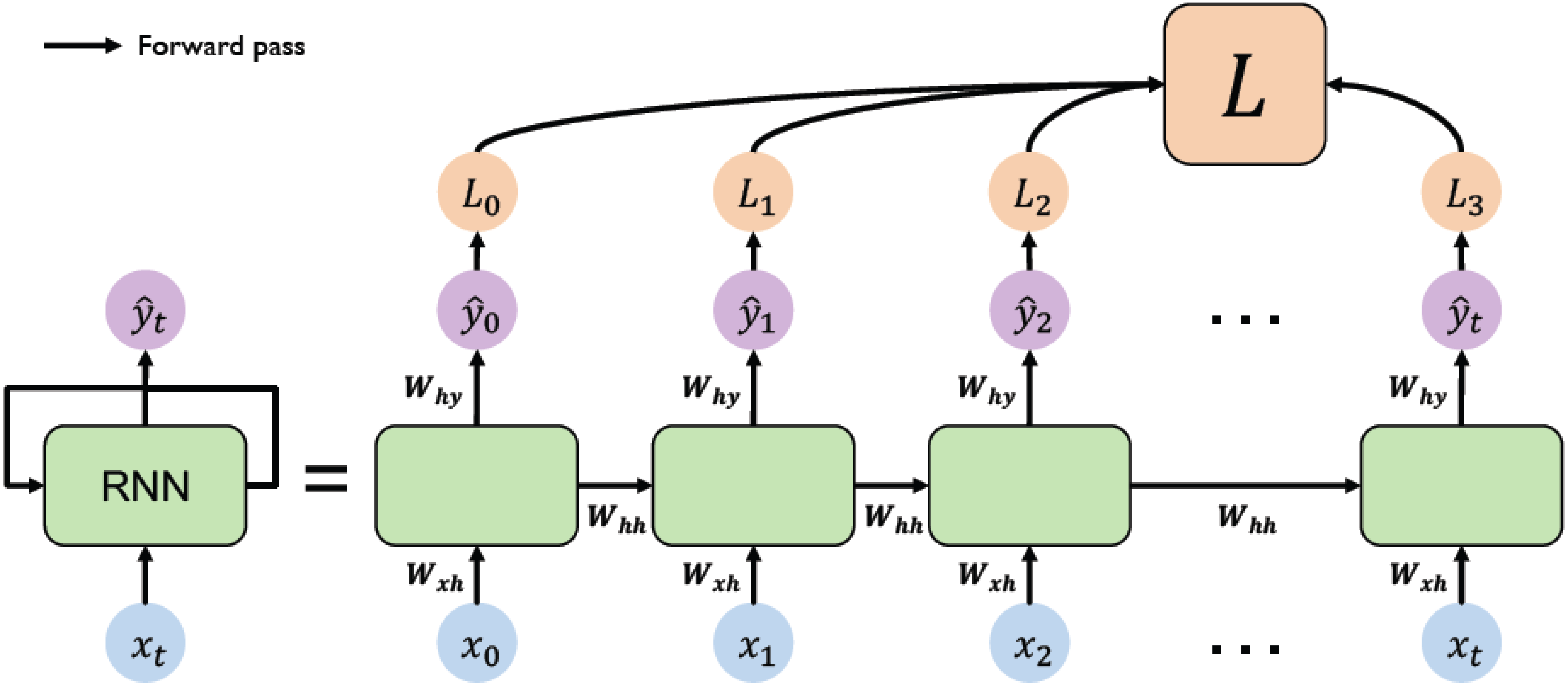
- RNN

Re-use the **same weight matrices** at every time step



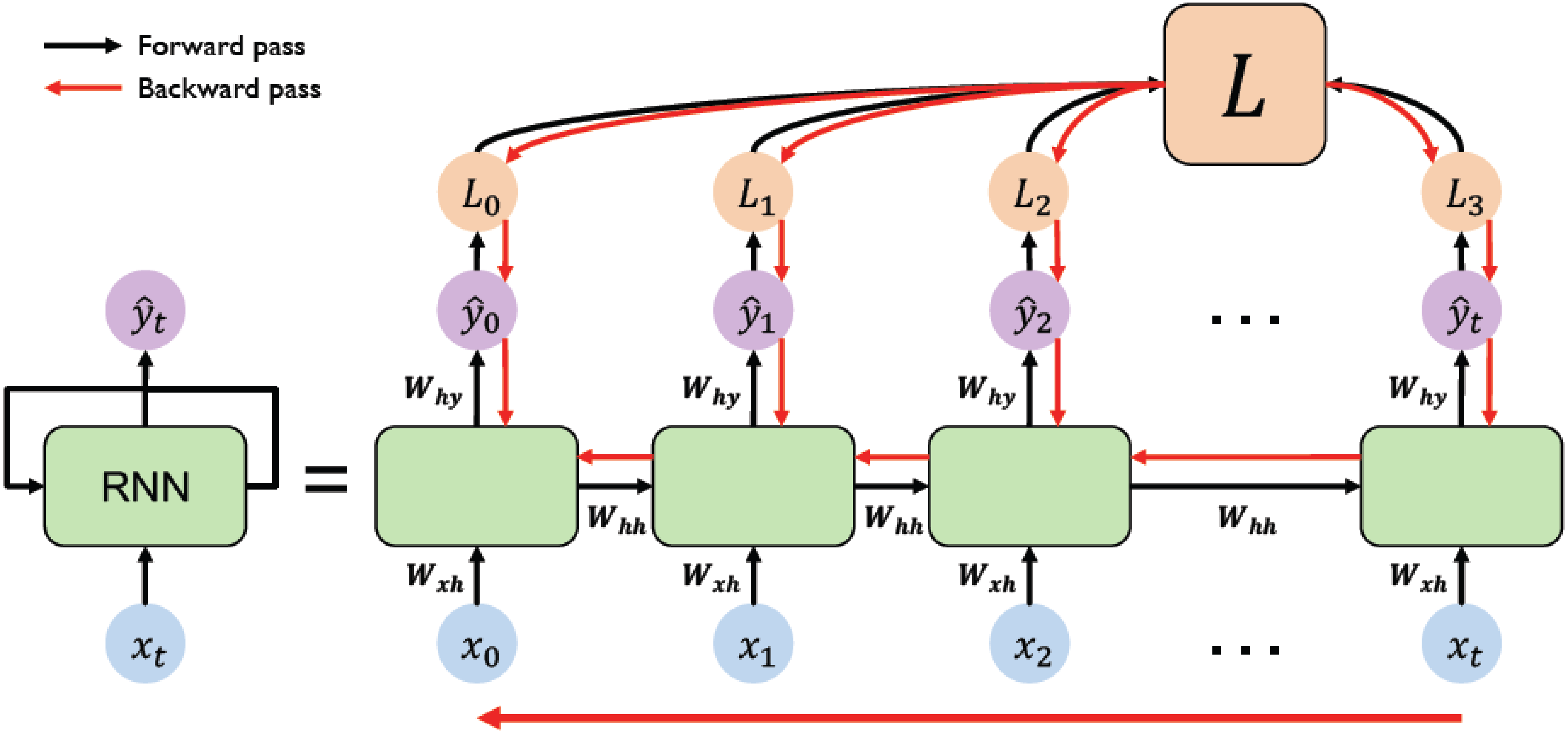
Recurrent Neural Network

- RNN



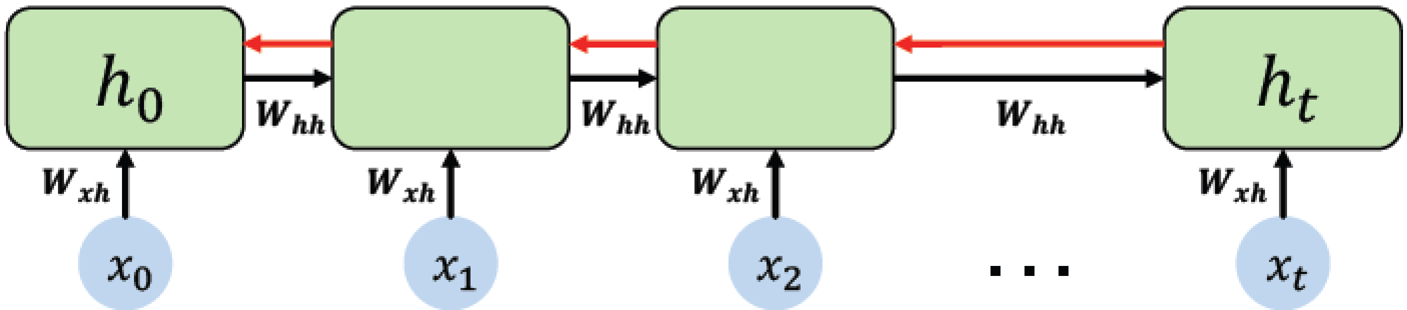
Recurrent Neural Network

- RNN



Recurrent Neural Network

- RNN



Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

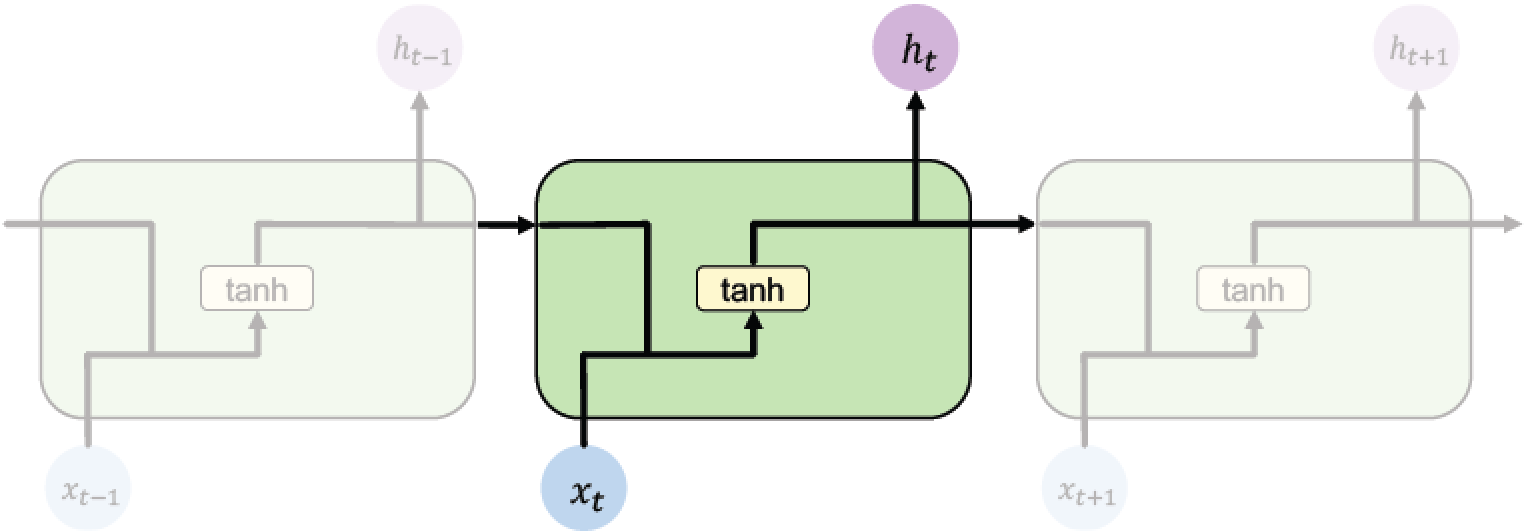
Largest singular value < 1:
vanishing gradients

1. **Activation function**
2. **Weight initialization**
3. **Network architecture**

Recurrent Neural Network

- RNN

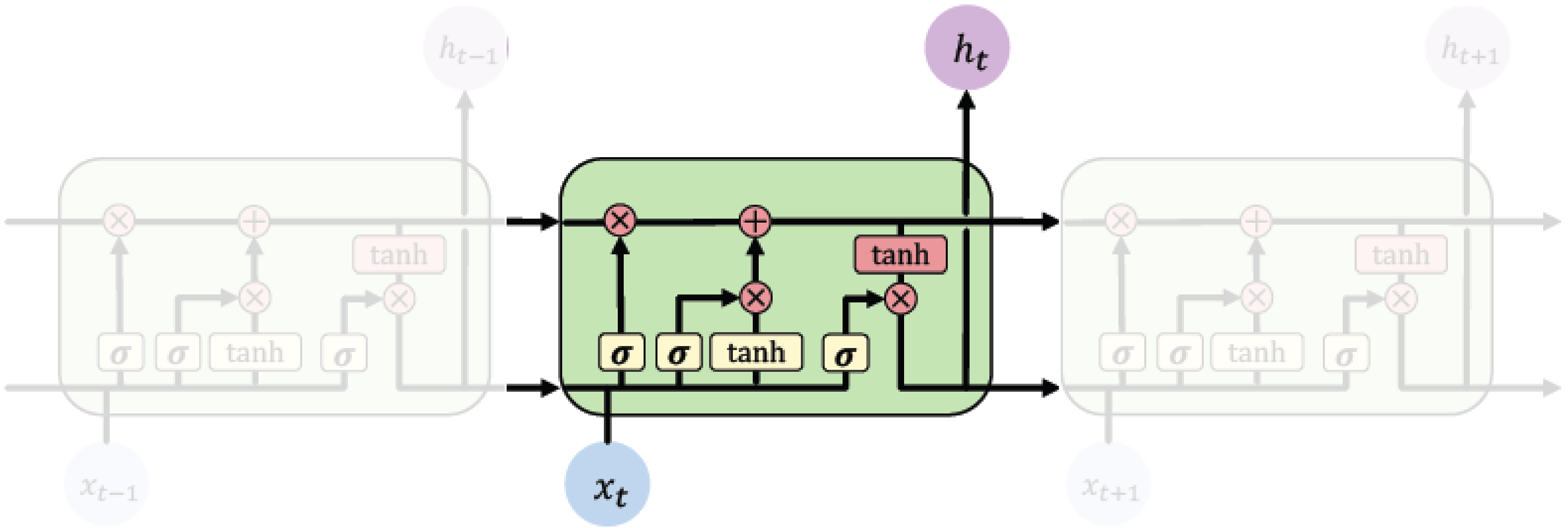
In a standard RNN, repeating modules contain a **simple computation node**



Recurrent Neural Network

- Long Short Term Memory (LSTM)

LSTM repeating modules contain **interacting layers** that **control information flow**



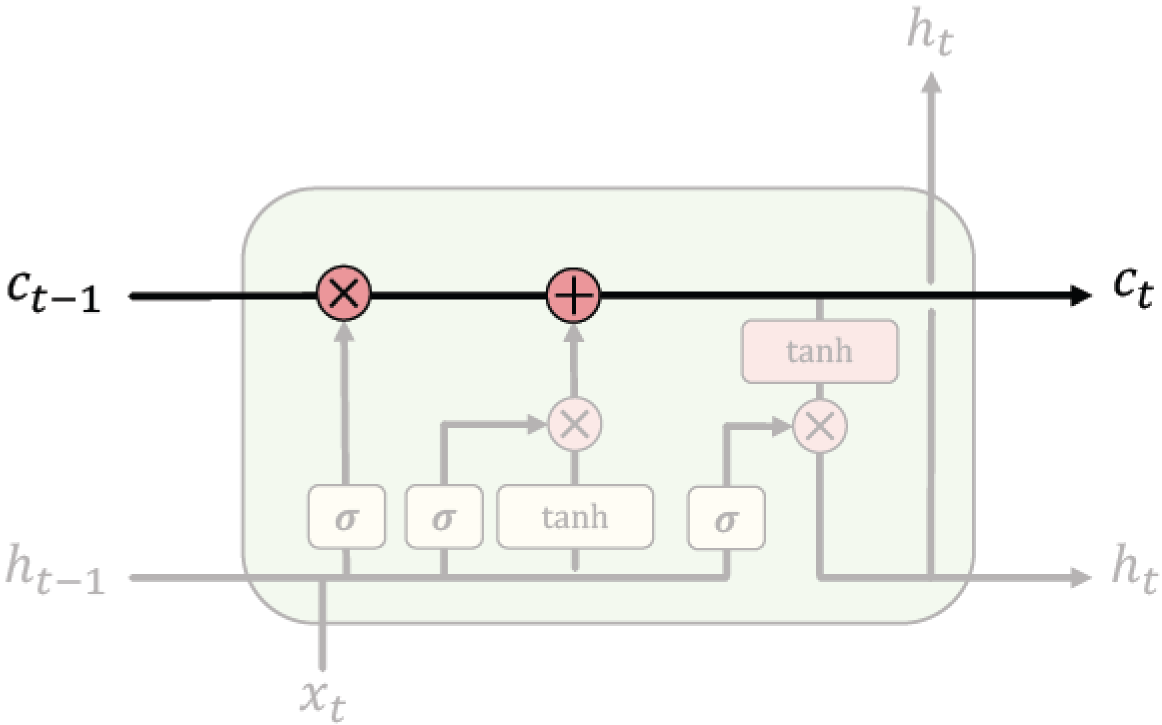
LSTM cells are able to track information throughout many timesteps

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network

- LSTM

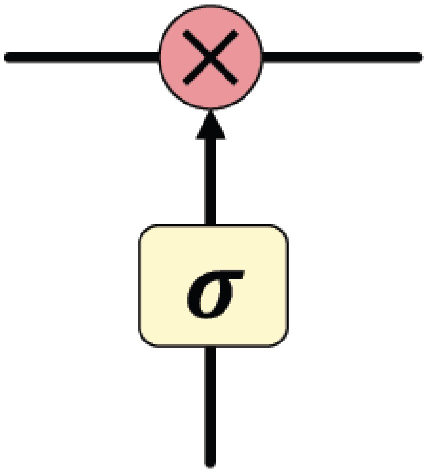
LSTMs maintain a **cell state** c_t where it's easy for information to flow



Recurrent Neural Network

- LSTM

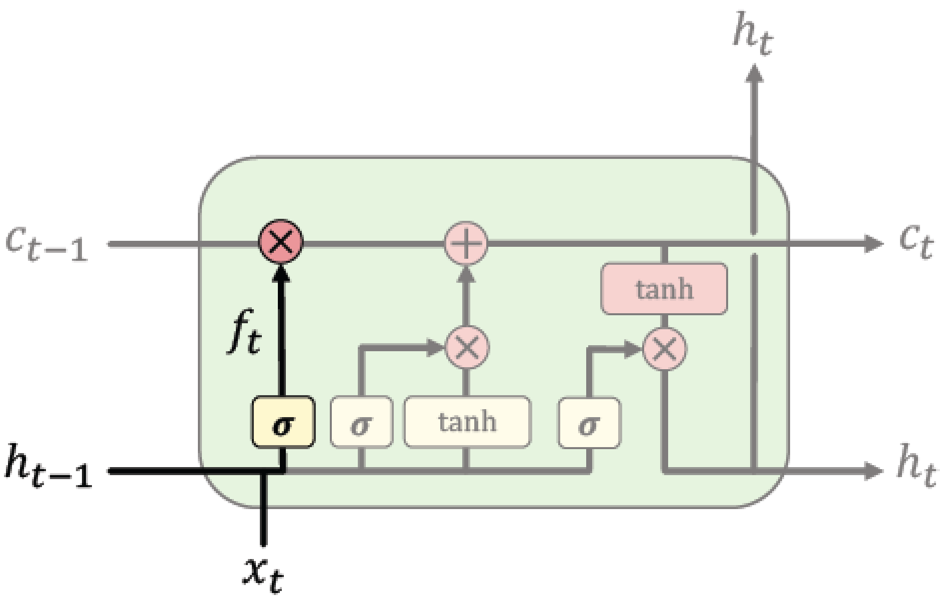
Information is **added** or **removed** to cell state through structures called **gates**



Gates optionally let information through, via a sigmoid neural net layer and pointwise multiplication

Recurrent Neural Network

- LSTM
  - Gate 1: forget irrelevant information

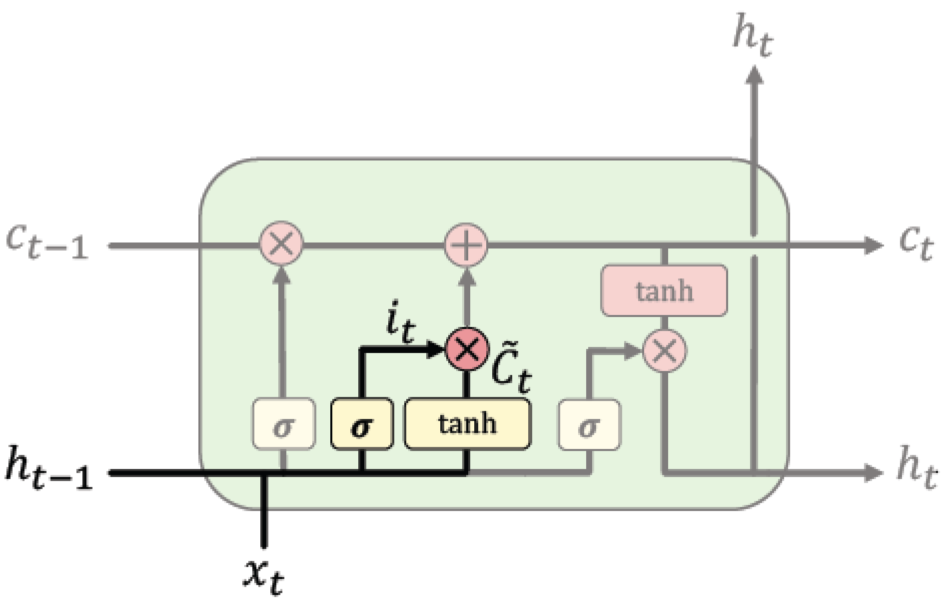


$$f_t = \sigma(\boldsymbol{W_i}[\, h_{t-1}, x_t \,] + b_f)$$

- Use previous cell output and input
- Sigmoid: value 0 and 1 – "completely forget" vs. "completely keep"

Recurrent Neural Network

- LSTM
  - Gate 2: identify new information to be stored



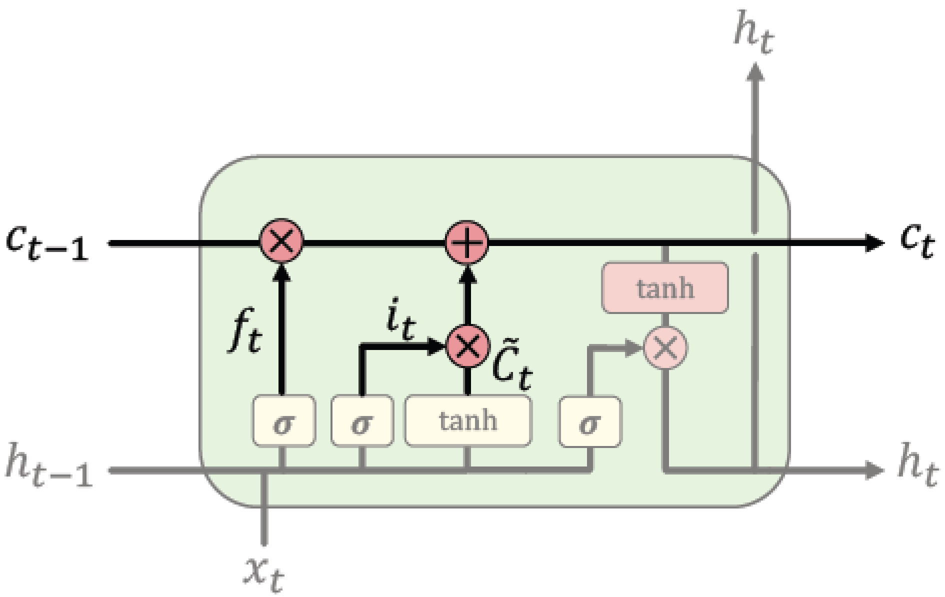
$$i_t = \sigma(\boldsymbol{W_i}[\,h_{t-1}, x_t\,] + b_i)$$

$$\tilde{C}_t = \tanh(\boldsymbol{W_C}[\,h_{t-1}, x_t\,] + b_C)$$

- Sigmoid layer: decide what values to update
- Tanh layer: generate new vector of "candidate values" that could be added to the state

Recurrent Neural Network

- LSTM
  - Gate 1+2: update cell state

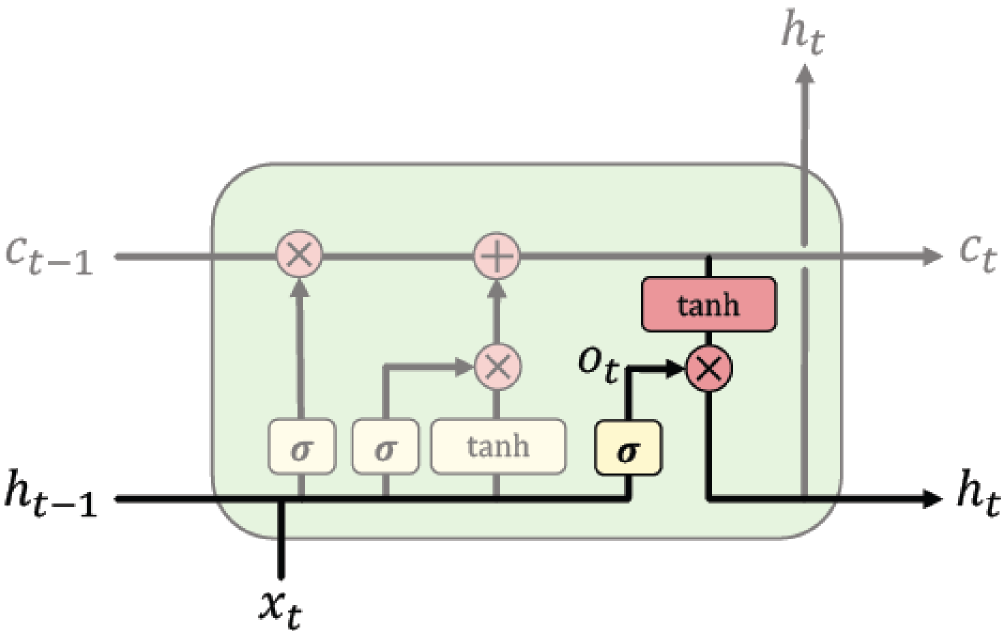


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Apply forget operation to previous internal cell state: $f_t * C_{t-1}$
- Add new candidate values, scaled by how much we decided to update: $i_t * \tilde{C}_t$

Recurrent Neural Network

- LSTM
  - Gate 3: output filtered version of cell state



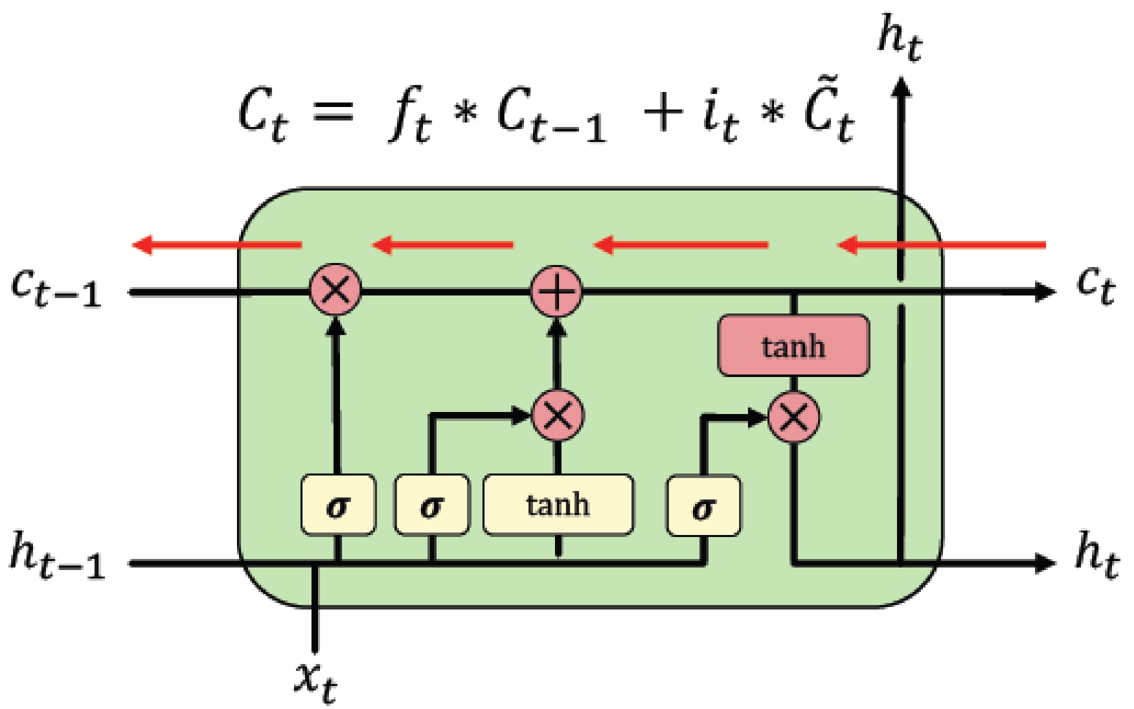
$$o_t = \sigma(\boldsymbol{W_o}[\, h_{t-1}, x_t \,] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

- Sigmoid layer: decide what parts of state to output
- Tanh layer: squash values between -1 and 1
- $o_t * \tanh(C_t)$: output filtered version of cell state

Recurrent Neural Network

- LSTM - backpropagation

Backpropagation from C_t to C_{t-1} requires only elementwise multiplication!
No matrix multiplication → avoid vanishing gradient problem.



Recurrent Neural Network

- LSTM – Key concepts:

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - Forget gate gets rid of irrelevant information
   - Selectively update cell state
   - Output gate returns a filtered version of the cell state
3. Backpropagation from c_t to c_{t-1} doesn't require matrix multiplication: **uninterrupted gradient flow**

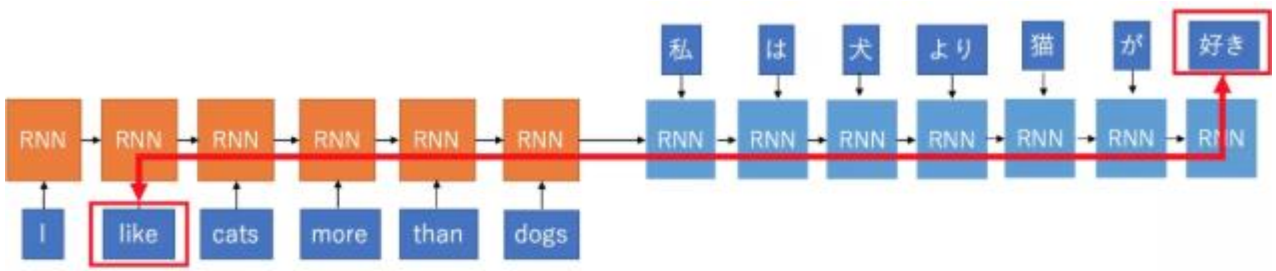
Recurrent Neural Network

- Applications

Speech recognition

Tweet sentiment classification

Machine translation



Recurrent Neural Network

- Applications

Image captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

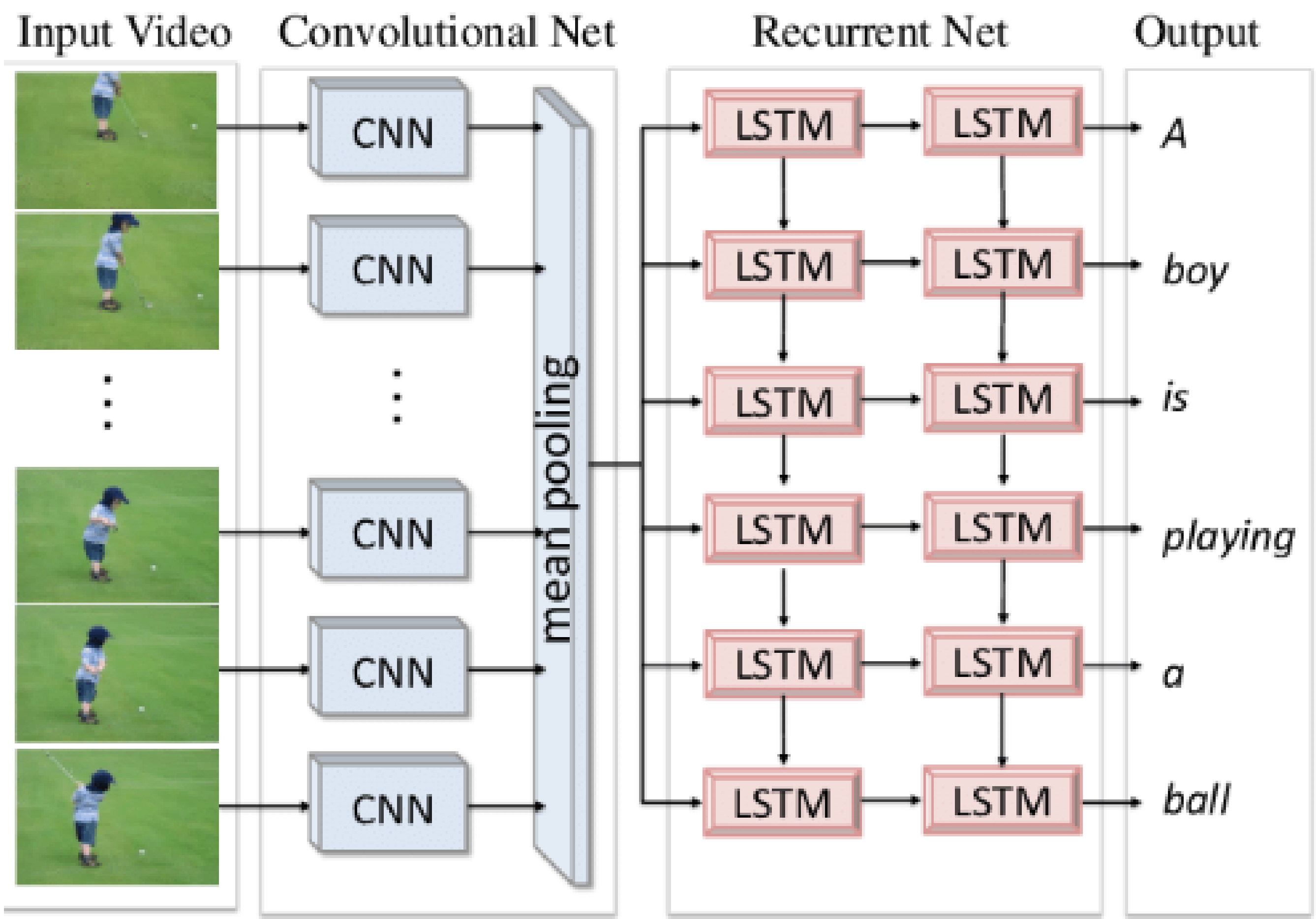
"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

Recurrent Neural Network

- Applications

Video understanding

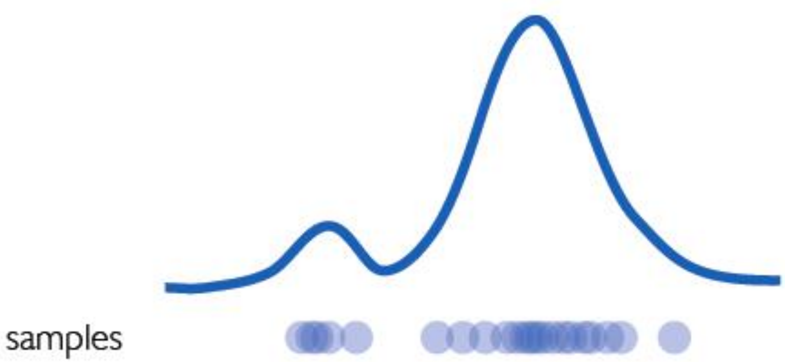


Deep Generative Models

- Autoencoder
- Generative Adversarial Networks

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation



Sample Generation

Input samples

Training data $\sim P_{data}(x)$

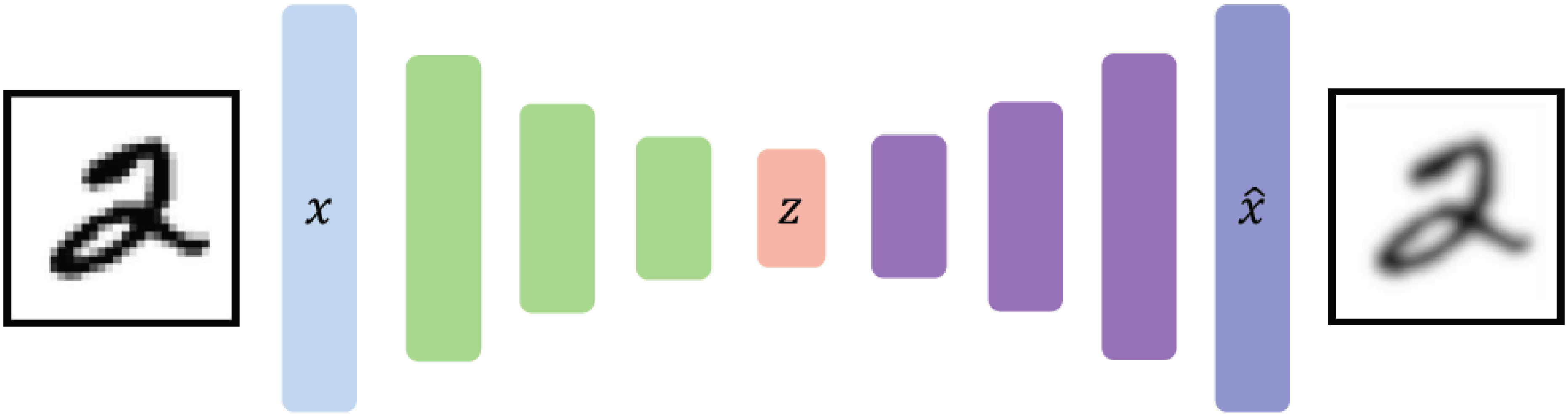
Generated samples

Generated $\sim P_{model}(x)$

How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

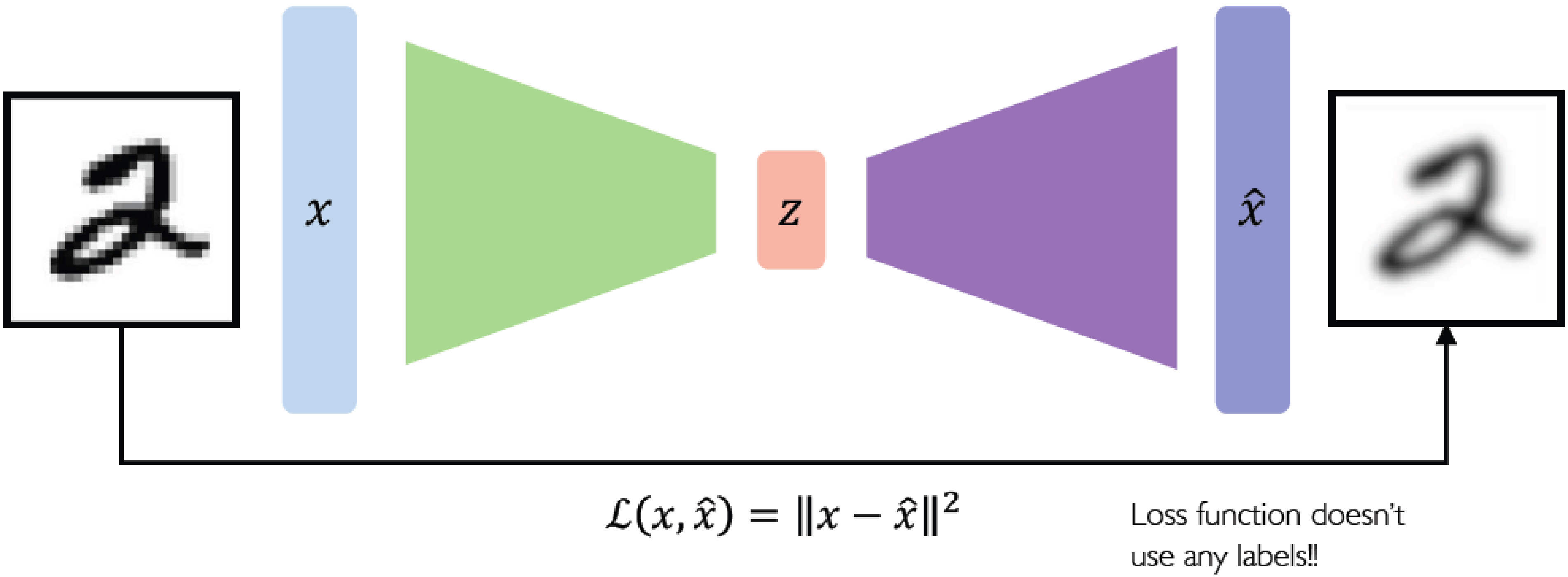
Autoencoder

- Autoencoder



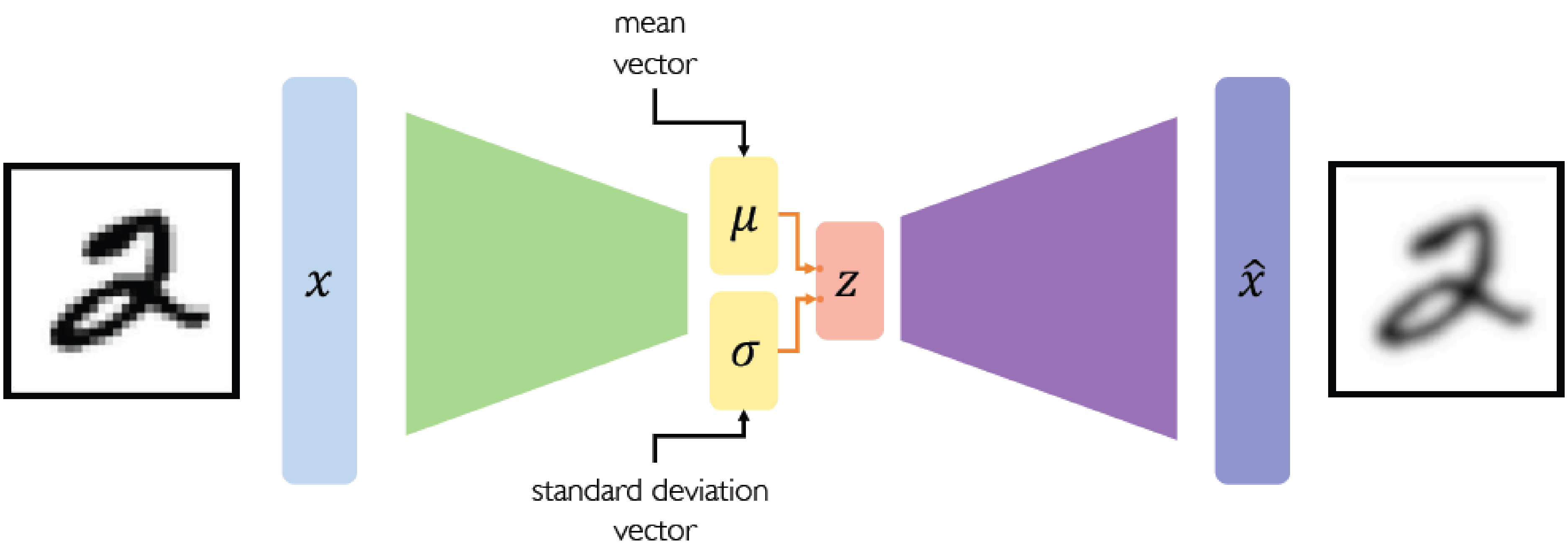
Autoencoder

- Autoencoder



Autoencoder

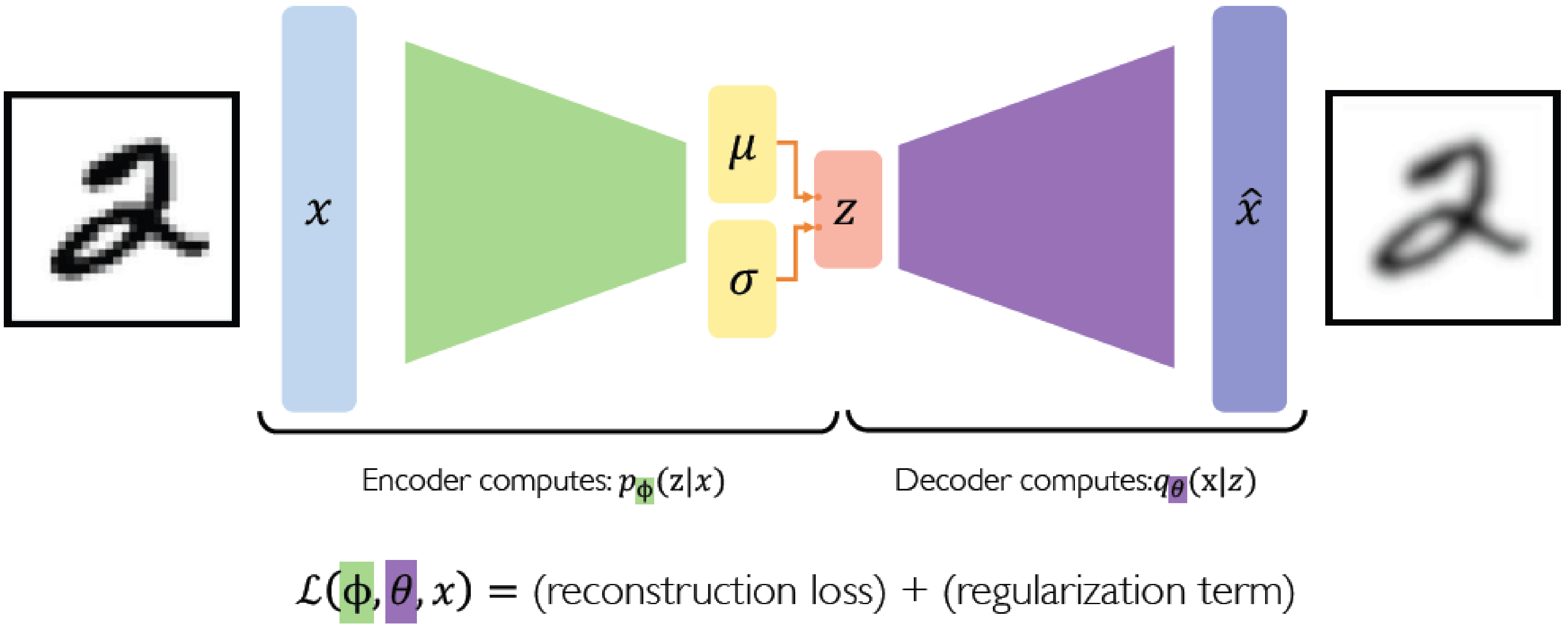
- Variational Autoencoders



Variational autoencoders are a probabilistic twist on autoencoders!
Sample from the mean and standard dev. to compute latent sample

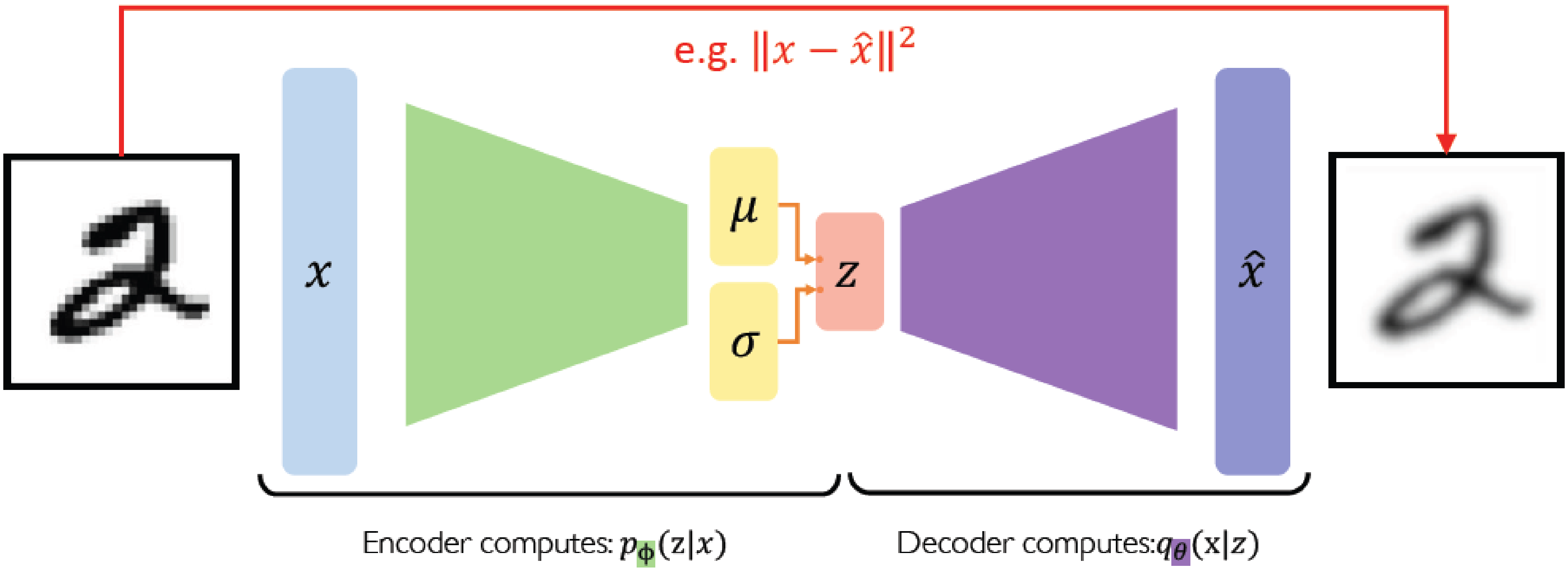
Autoencoder

- Variational Autoencoders



Autoencoder

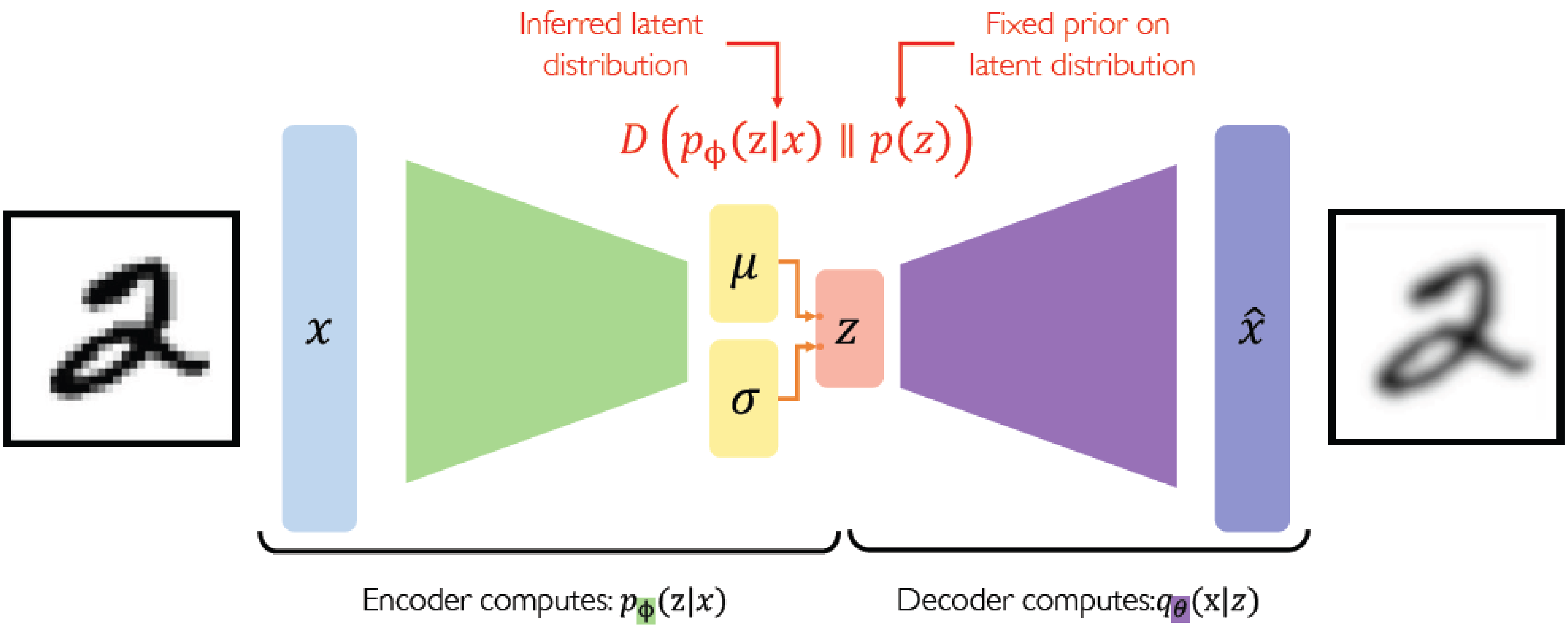
- Variational Autoencoders



$$\mathcal{L}(\phi, \theta, x) = \boxed{\text{(reconstruction loss)}} + \text{(regularization term)}$$

Autoencoder

- Variational Autoencoders

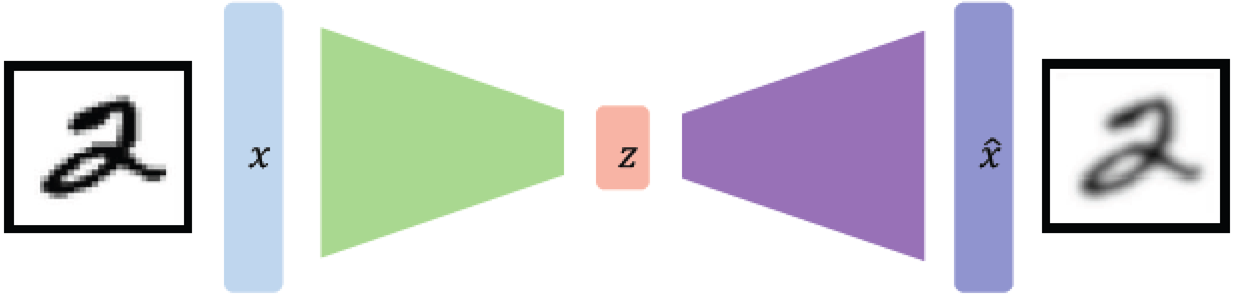


$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \boxed{\text{(regularization term)}}$$

Autoencoder

- Variational Autoencoders

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



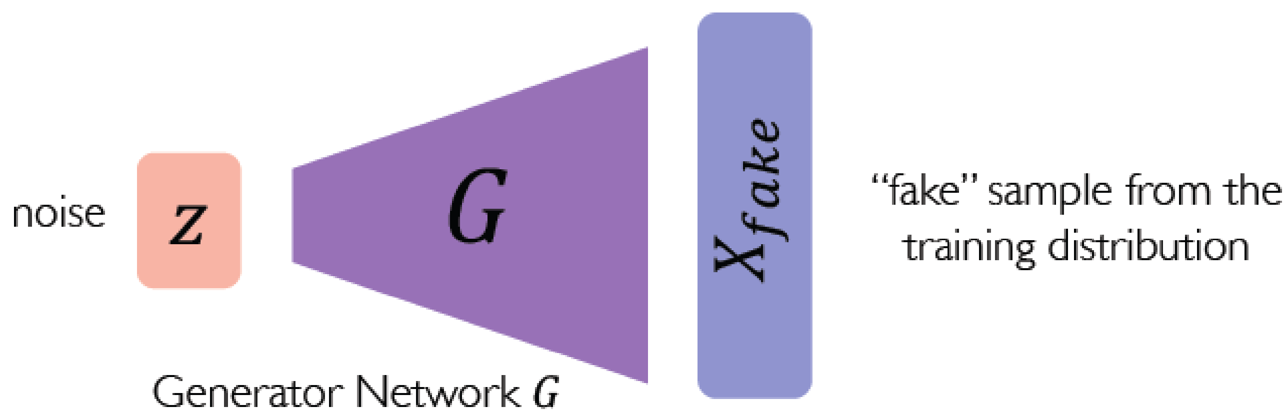
Generative Adversarial Networks

- Motivation

Idea: don't explicitly model density, and instead just sample to generate new instances.

Problem: want to sample from complex distribution – can't do this directly!

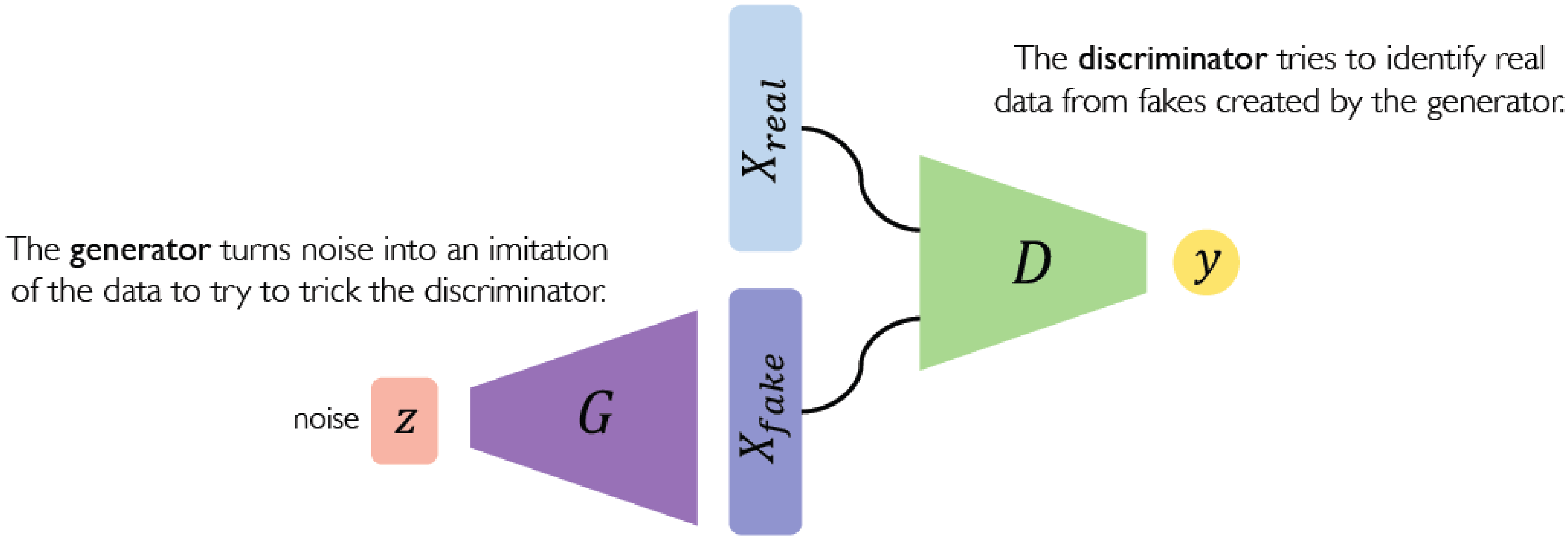
Solution: sample from something simple (noise), learn a transformation to the training distribution.



Generative Adversarial Networks

- GAN

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.



Generative Adversarial Networks

- GAN Training

Discriminator tries to identify real data from fakes created by the generator.
Generator tries to create imitations of data to trick the discriminator.

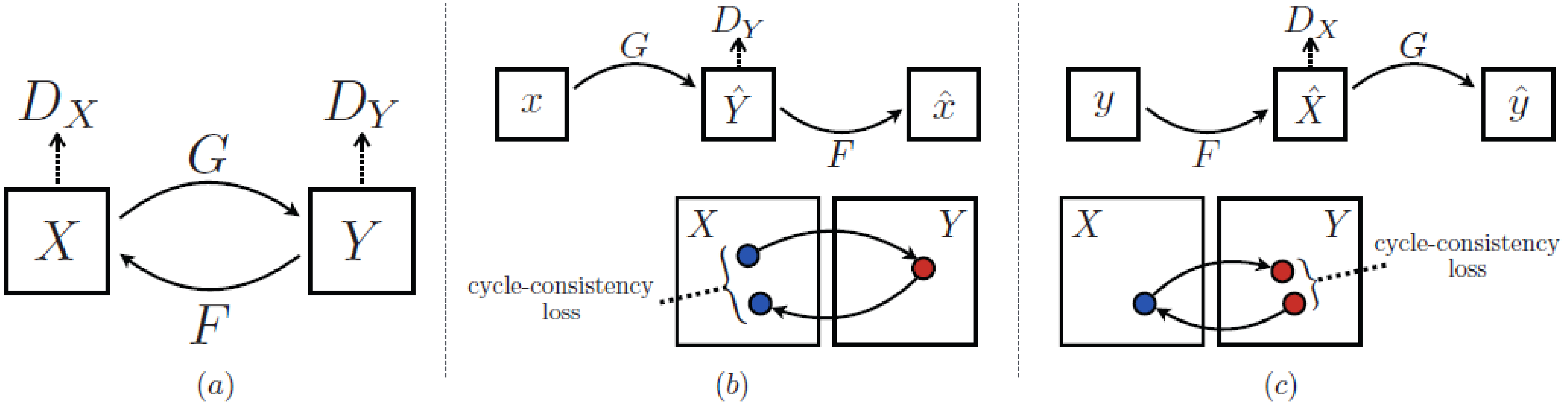
Train GAN jointly via **minimax** game:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

Discriminator wants to maximize objective s.t. $D(x)$ close to 1, $D(G(z))$ close to 0.
Generator wants to minimize objective s.t. $D(G(z))$ close to 1.

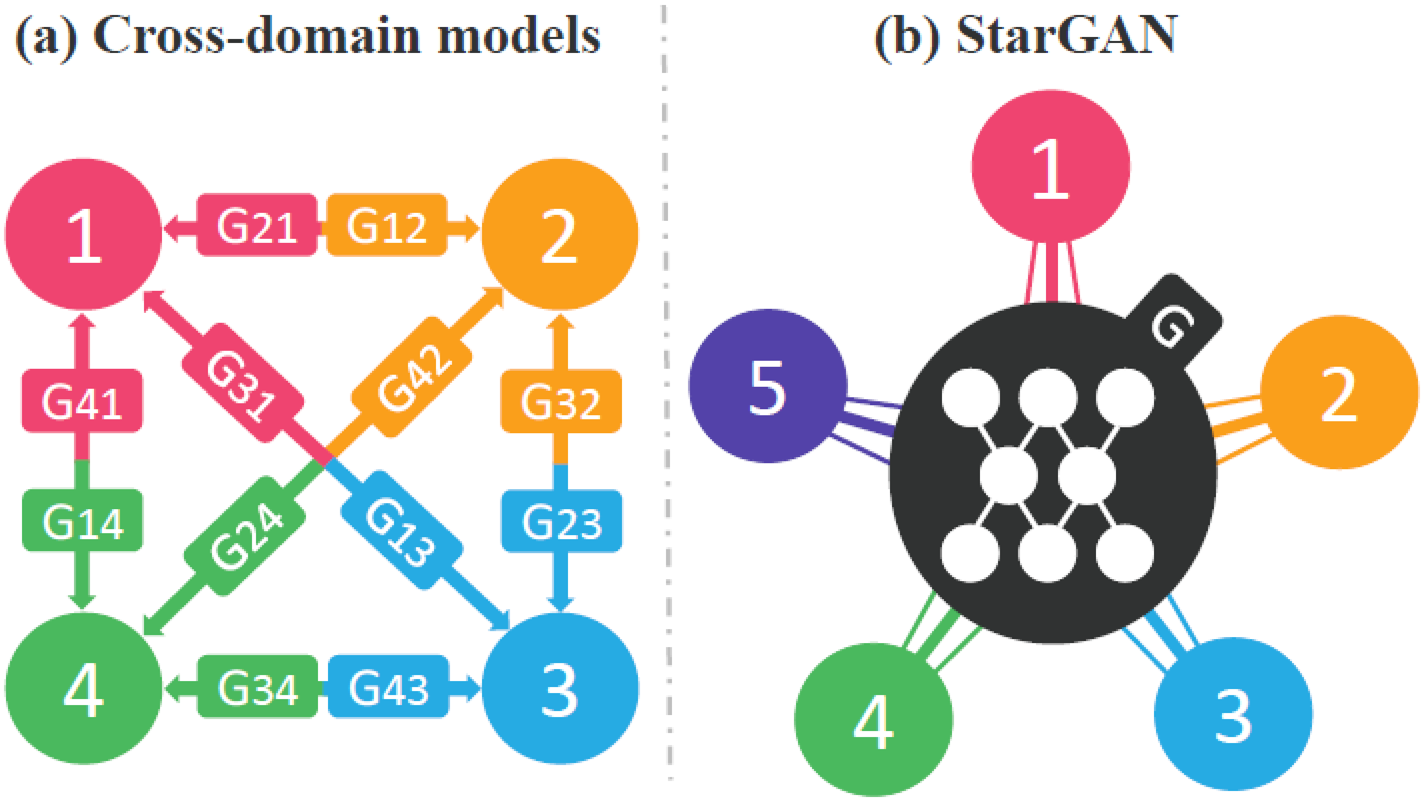
Generative Adversarial Networks

- GAN Variants – CycleGAN [CVPR, 2017]



Generative Adversarial Networks

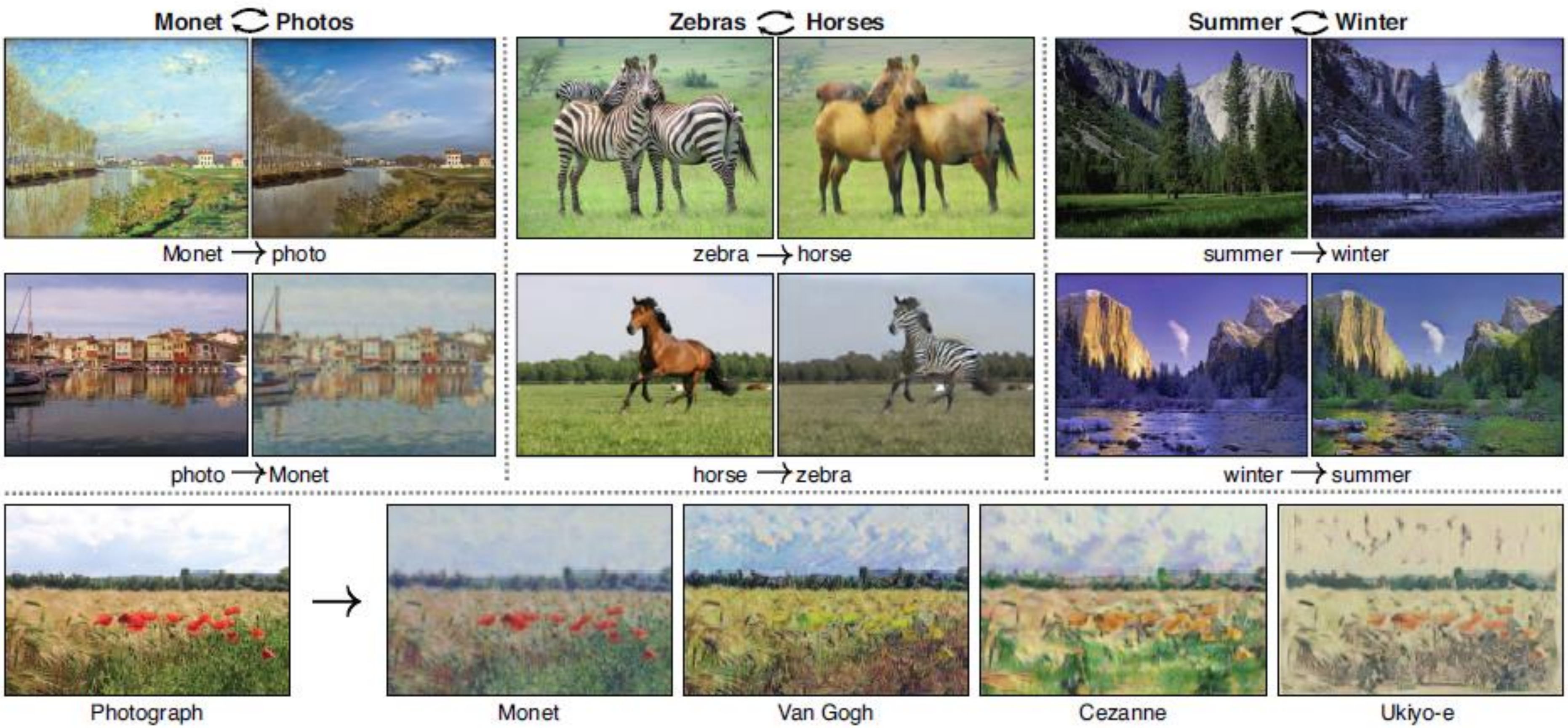
- GAN Variants – StarGAN [CVPR, 2018]



Generative Adversarial Networks

- Applications

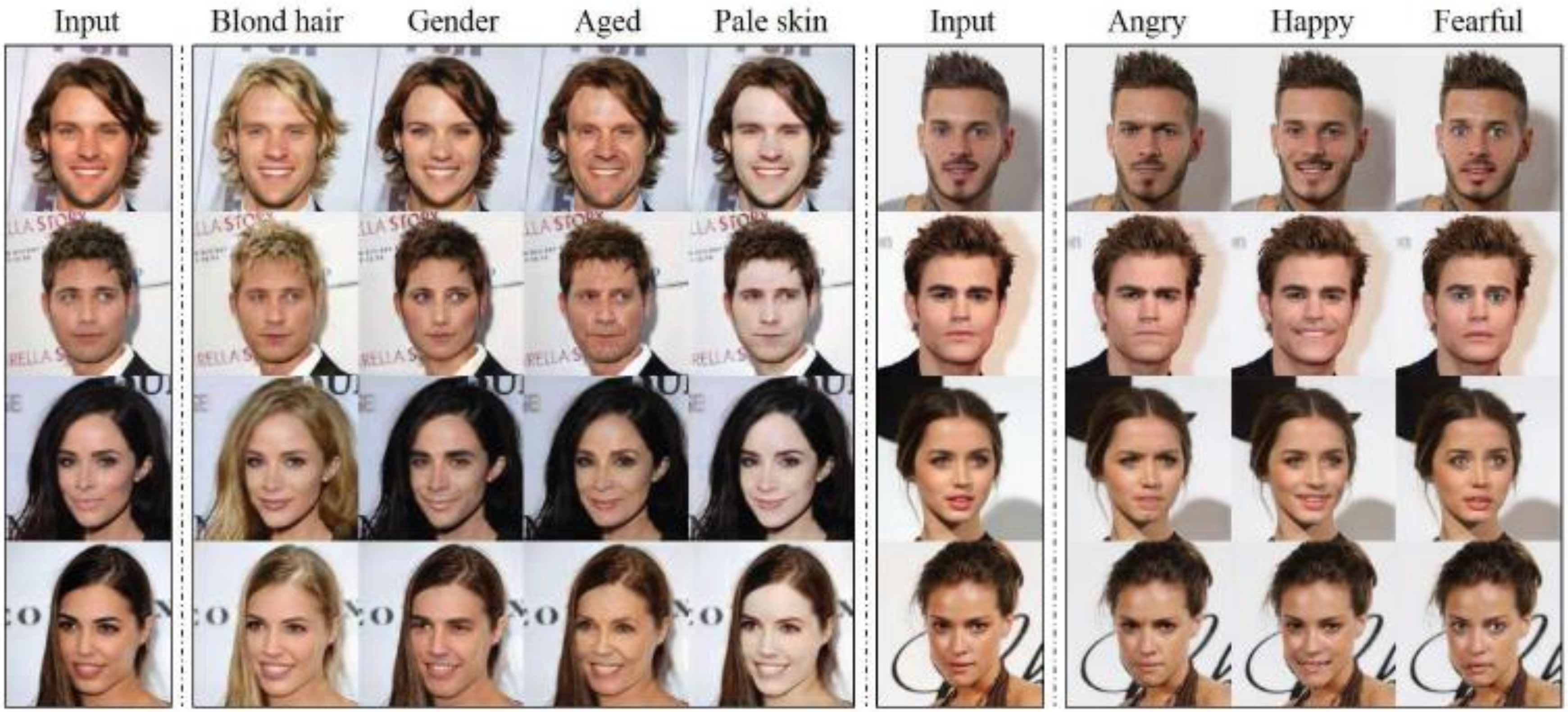
Style transfer



Generative Adversarial Networks

- Applications

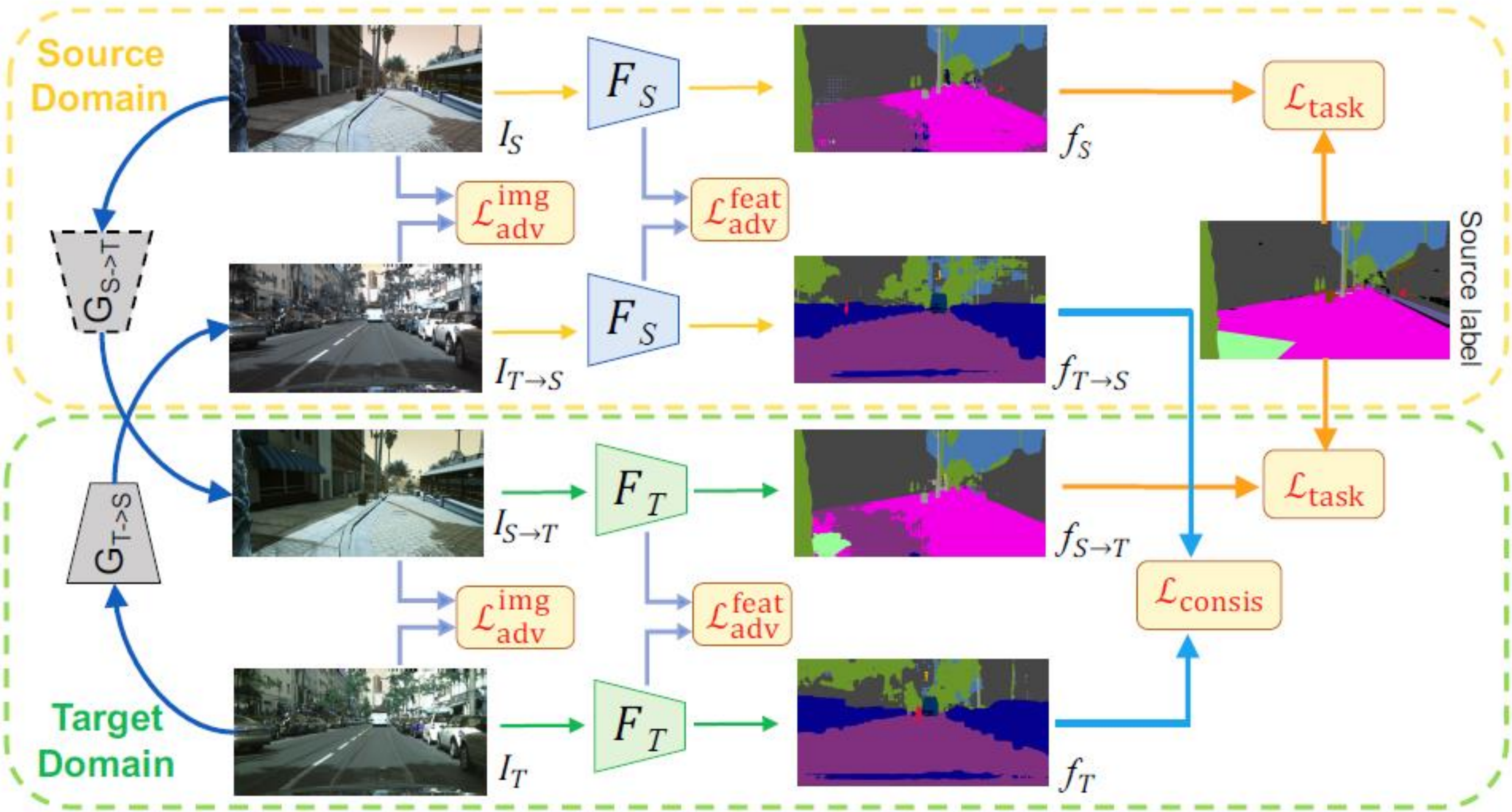
Synthetic face



Generative Adversarial Networks

- Applications

Domain Adaptation [CVPR, 2019]



Readings

- Artificial Intelligence
    - Chapter 18.7