An Introduction to Deep Learning

Marc'Aurelio Ranzato
Facebook AI Research
ranzato@fb.com

DeepLearn Summer School - Bilbao, 17 July 2017

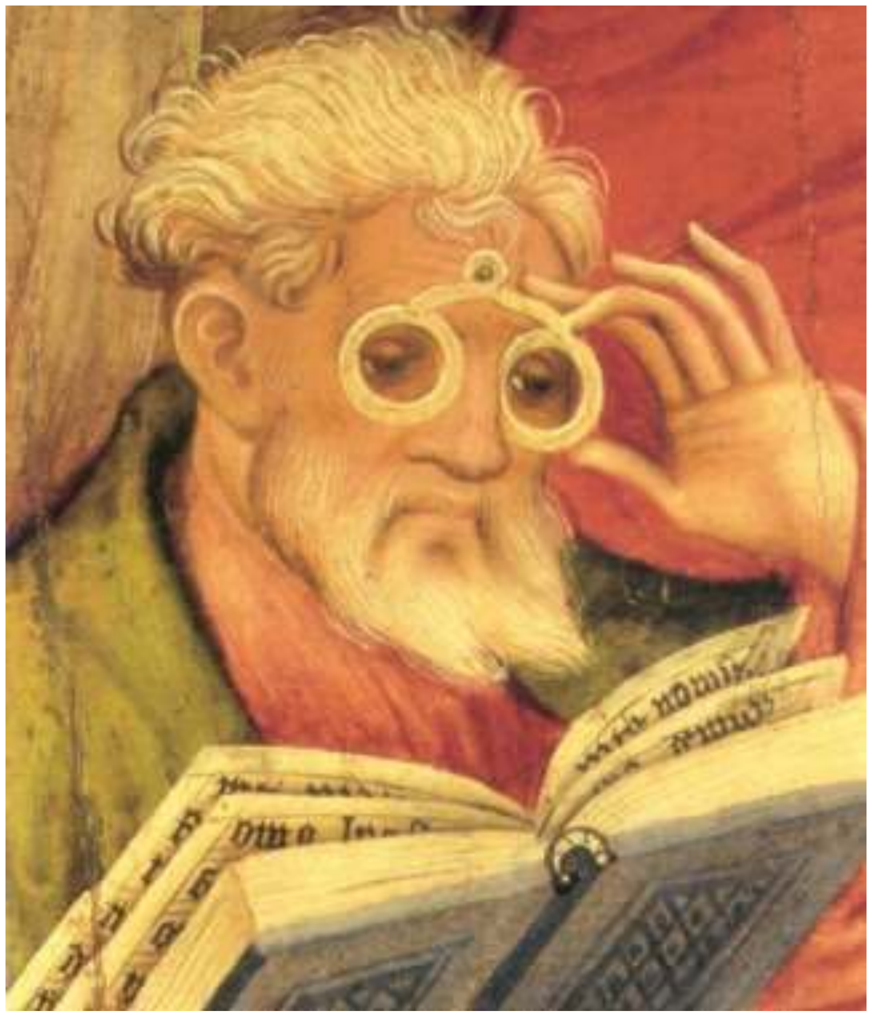
big picture first...

Goal

A.I. : build a system that is useful to people and that extends humans abilities.

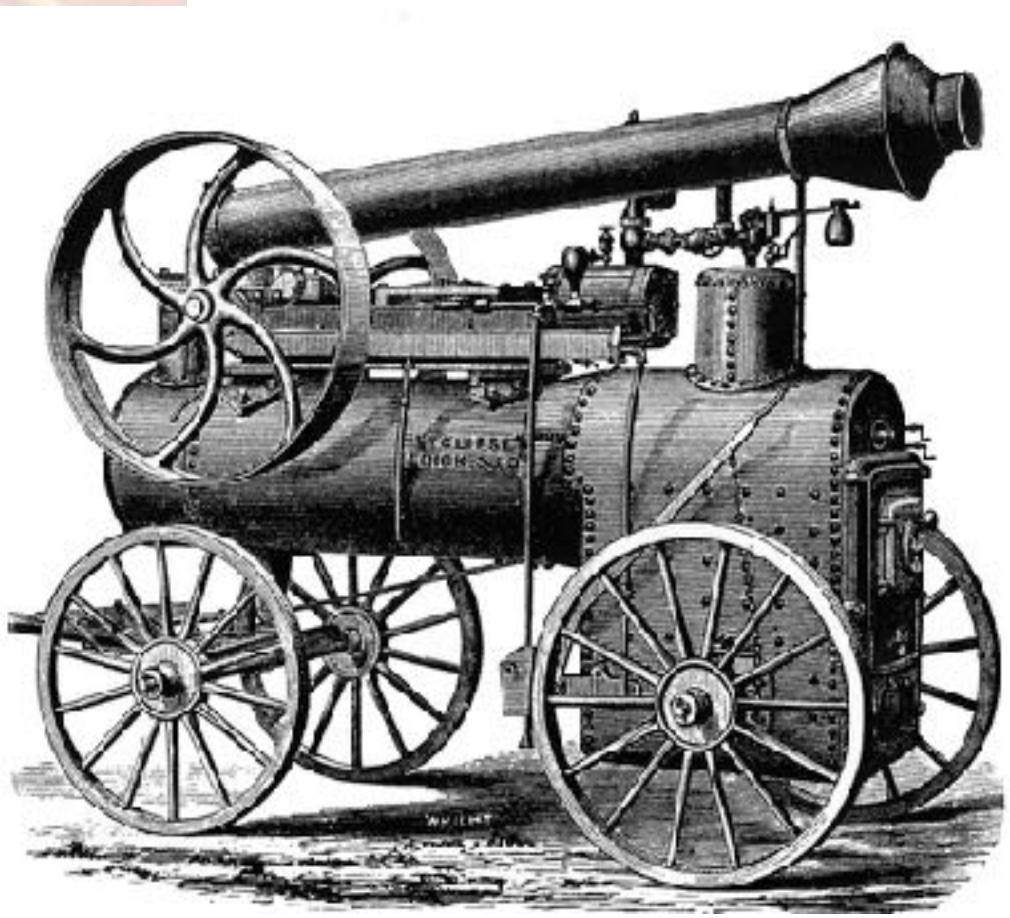
More interested in complementing human skills than necessarily replicating them.

Extending Human Abilities: Examples



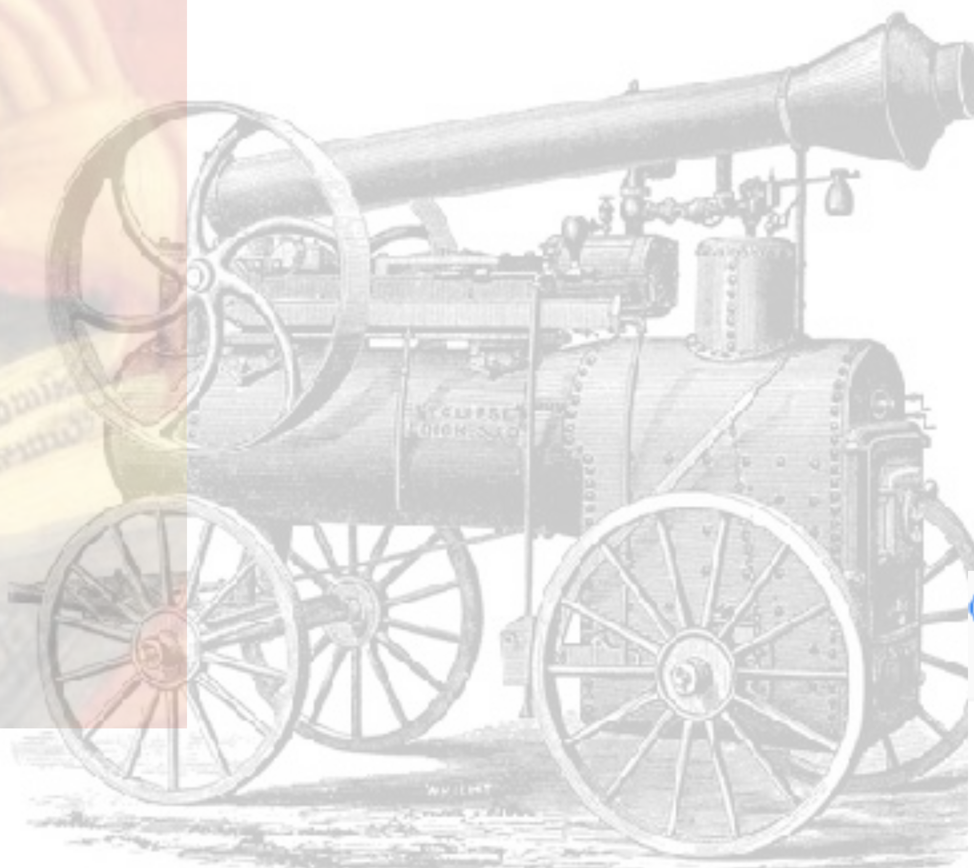
XIII century: extending human vision with eyeglasses

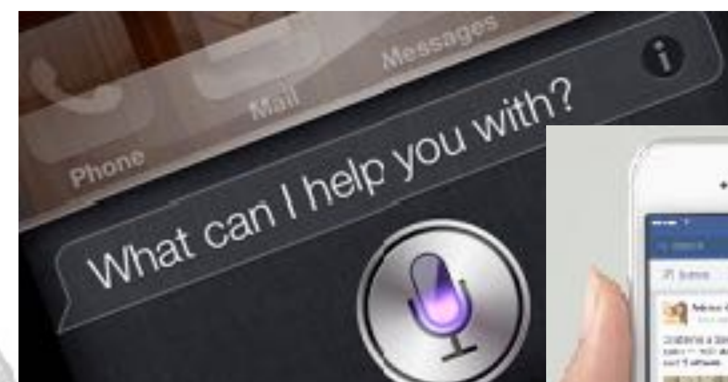
Extending Human Abilities: Examples

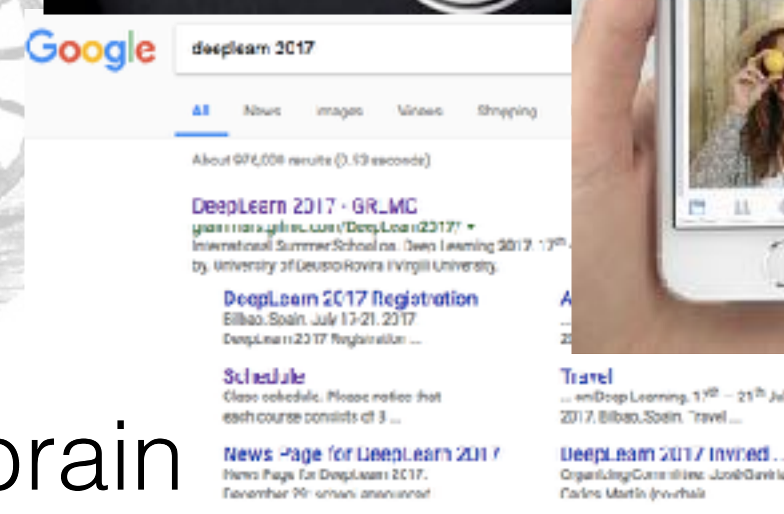


XVII-XVIII centuries: “extending” human legs with steam engine for faster transportation

Extending Human Abilities: Examples



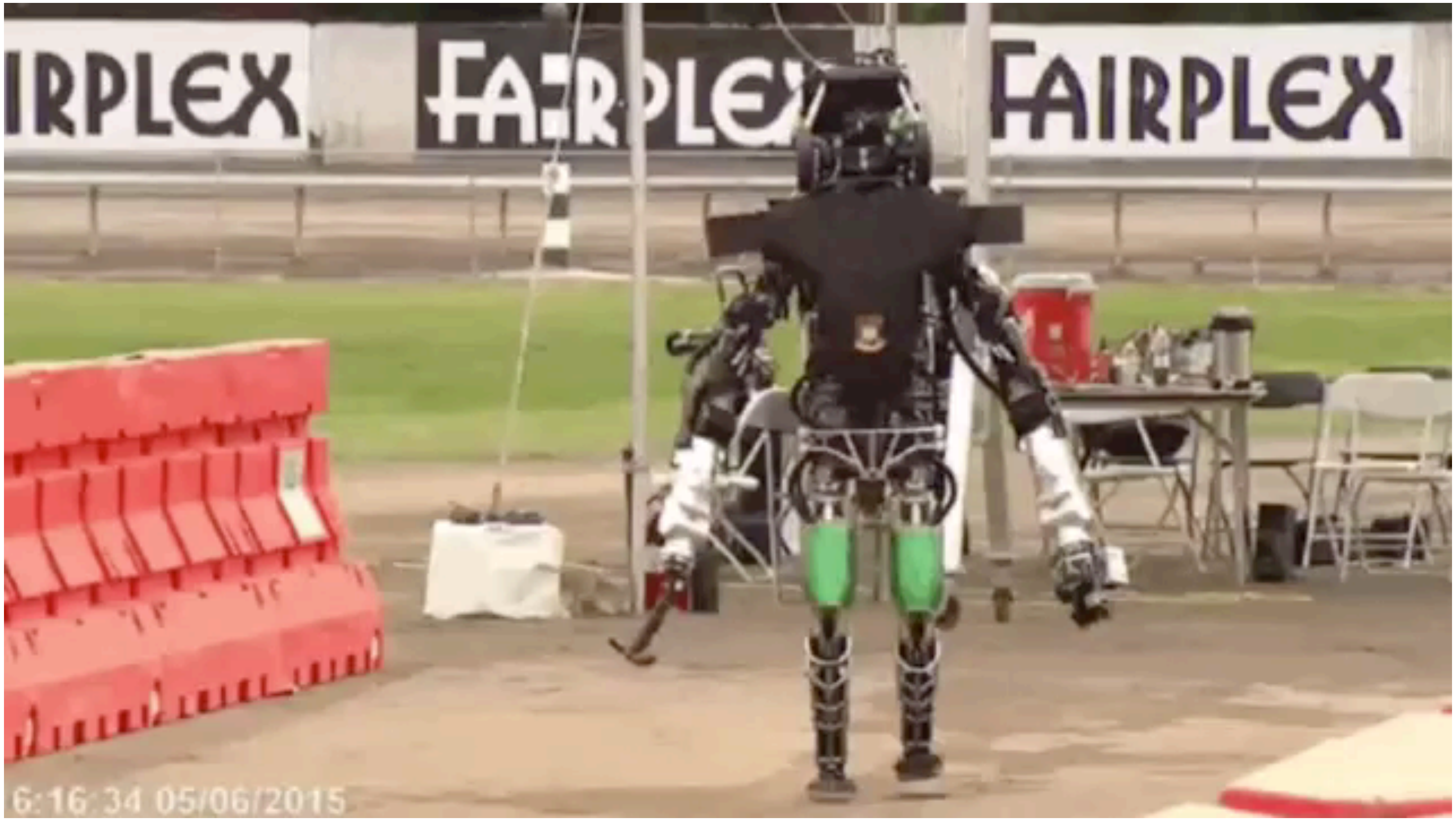




XXI century: extending the human brain
by making information more easily accessible

What's next?

- Build A.I. that actually works…



DARPA Challenge videos

Technical Challenges

- Content understanding
  - Vision
  - Audio
  - Text
- Learn as much as possible from data with as little as possible human engineering
- Sample and computational efficiency
- Learn with as little supervision as possible
- Knowledge transfer
- Memory
- Acquisition of common sense
- End-to-end logical reasoning, planning
- Robustness to uncertainty

What is Deep Learning and How Can It Help?

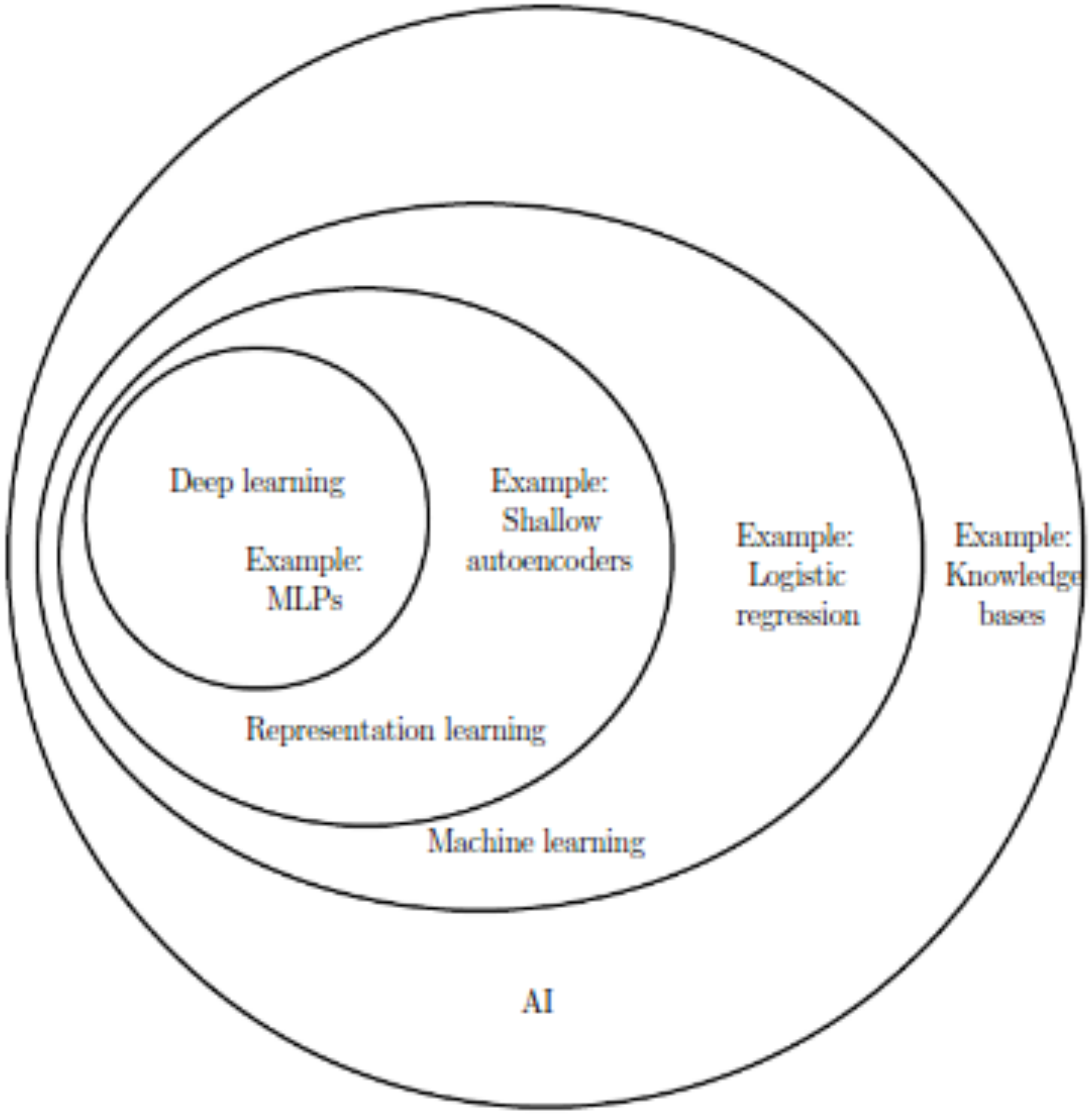


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

Goodfellow et al. “Deep Learning” MIT Press 2016

What is Deep Learning and How Can It Help?

Deep Learning (DL) is a class of Machine Learning methods that aims at learning **feature hierarchies**.

What is Deep Learning and How Can It Help?

Philosophical justification (to be further clarified later):

- Hierarchical models are potentially more efficient as they allow better feature sharing (compositionality).
- Intermediate representations are good candidate for transferring knowledge to other tasks.
- These models are inherently very modular.

DL is not the solution but a useful set of tools for our quest towards A.I.

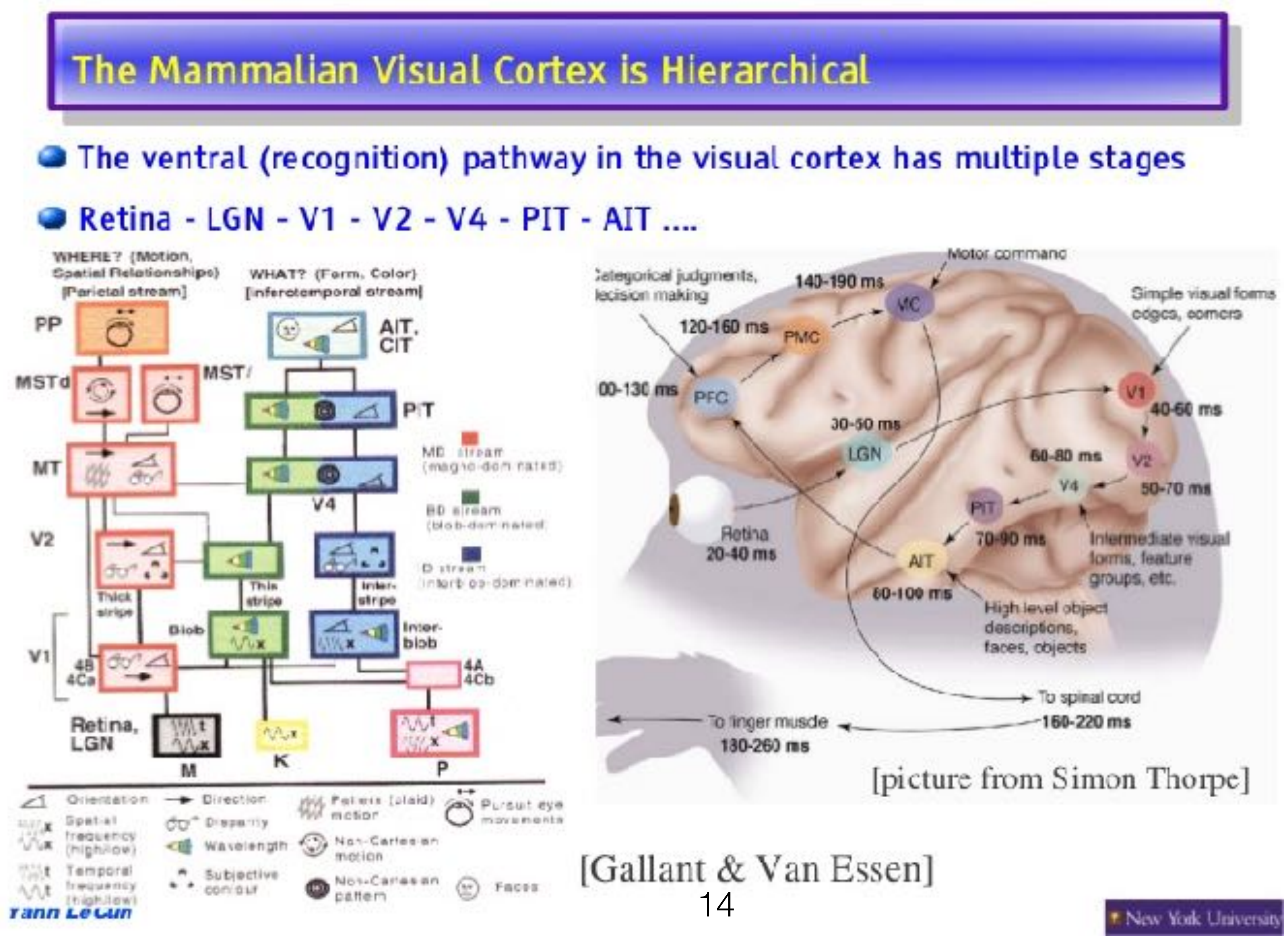
Hierarchical Structure: Vision

Images can be naturally decomposed in:

pixel -> edge -> texton -> super-pixel -> part -> object

Hierarchical Structure: Vision

There is evidence of a similar hierarchy in the mammalian visual cortex.



Hierarchical Structure: Vision

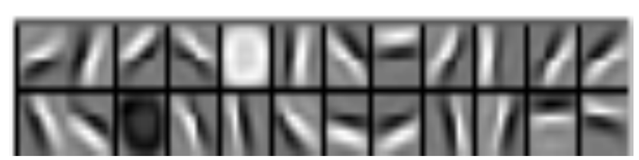
pixel -> edge -> texton -> motif -> part -> object

Several (deep) approaches mimic a similar structure

high-level parts  • • • 

mid-level parts 

Example 1

low level parts 

Input image 

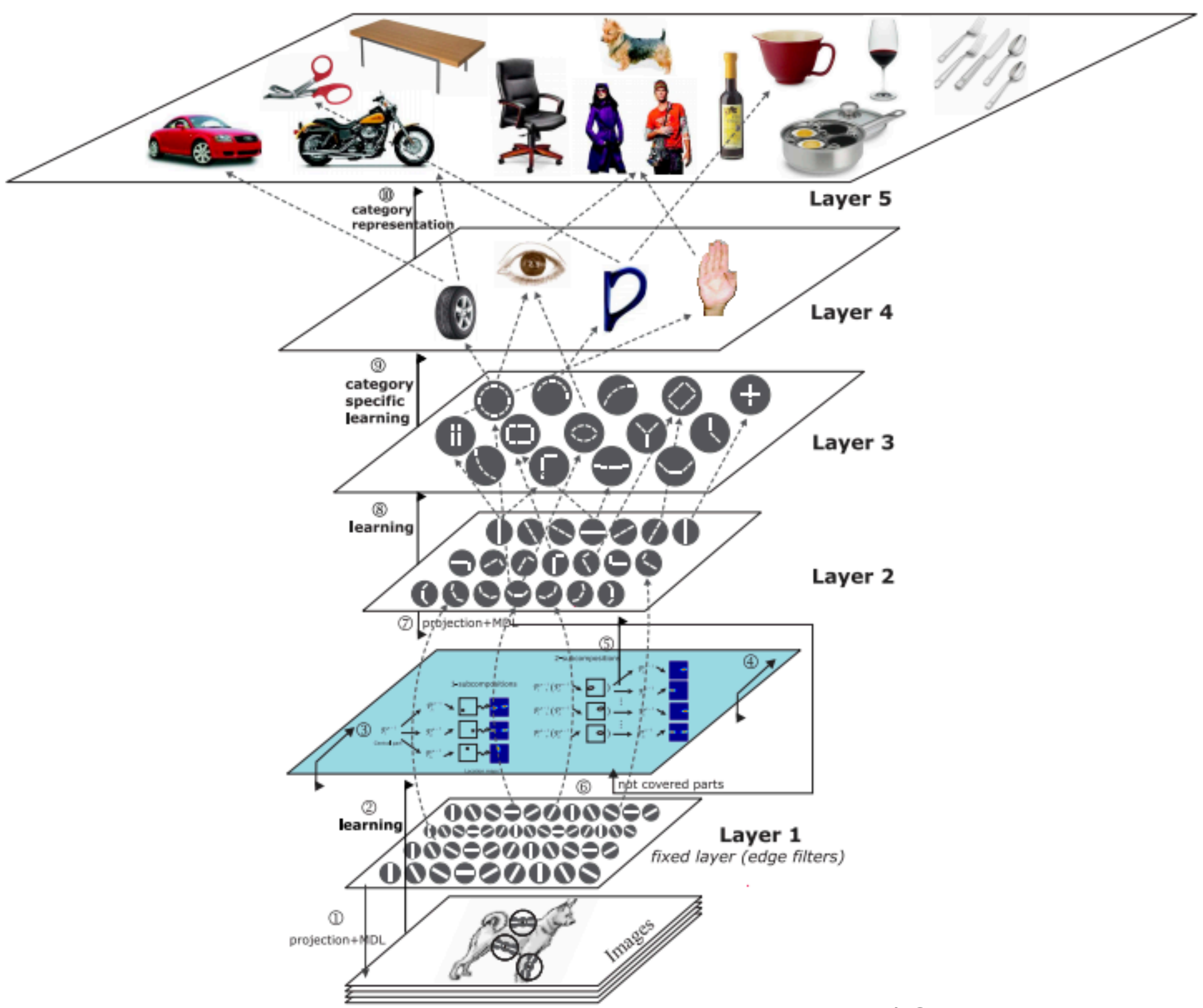
- Efficiency via compositionality
- Compositionality and knowledge transfer via feature sharing

Lee et al. “Convolutional DBNs…” ICML 09

Hierarchical Structure: Vision

pixel -> edge -> texton -> motif -> part -> object

Several (deep) approaches mimic a similar structure



Example 2

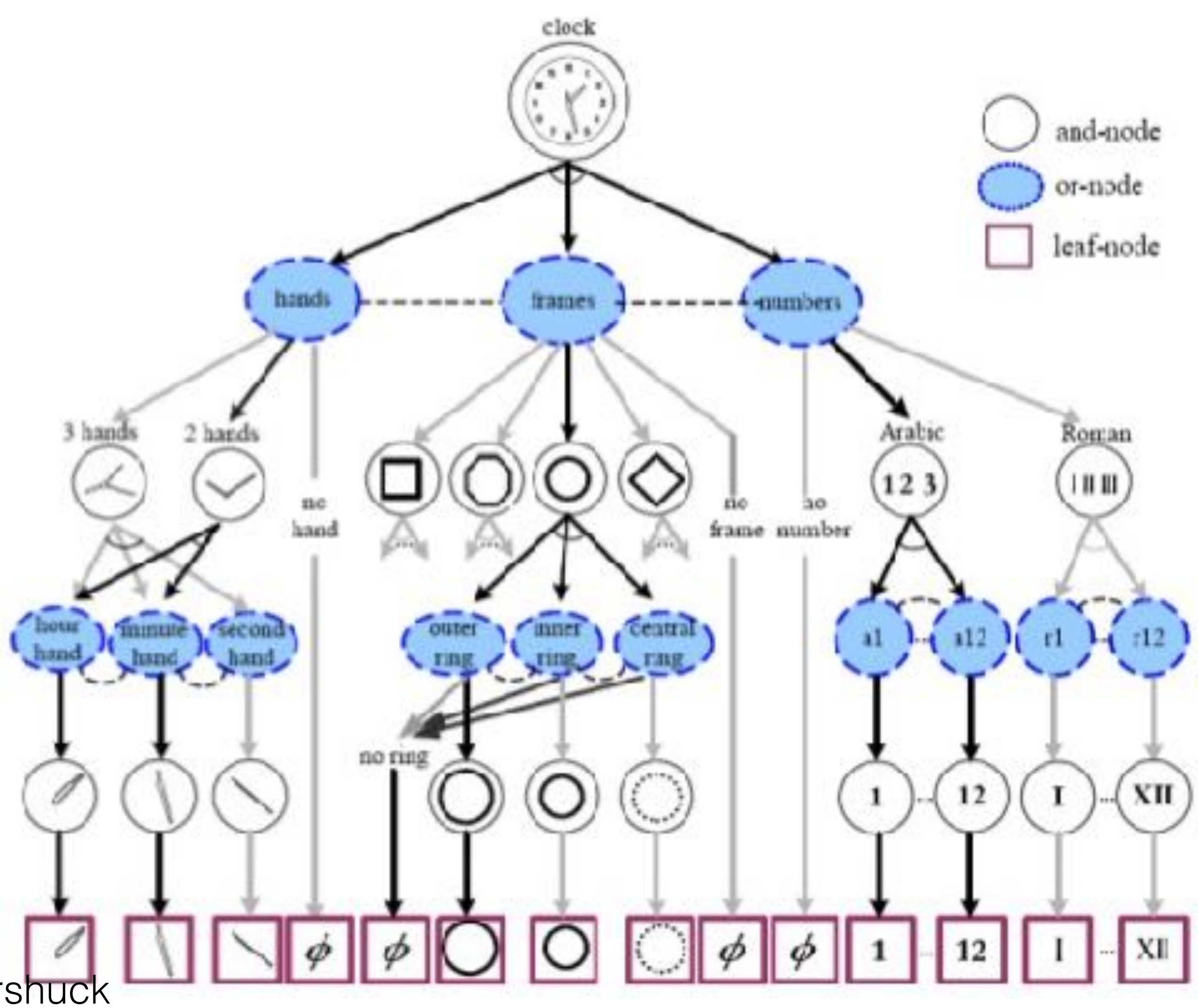
credit: R. Girshuck

Leonardis et al. “Learning hierarchical representations…” ISRR 07

Hierarchical Structure: Vision

pixel -> edge -> texton -> motif -> part -> object

Several (deep) approaches mimic a similar structure



Example 3

Zhu et al. “A stochastic grammar of images…” FTCGV 06

credit: R. Girshuck

Hierarchical Structure

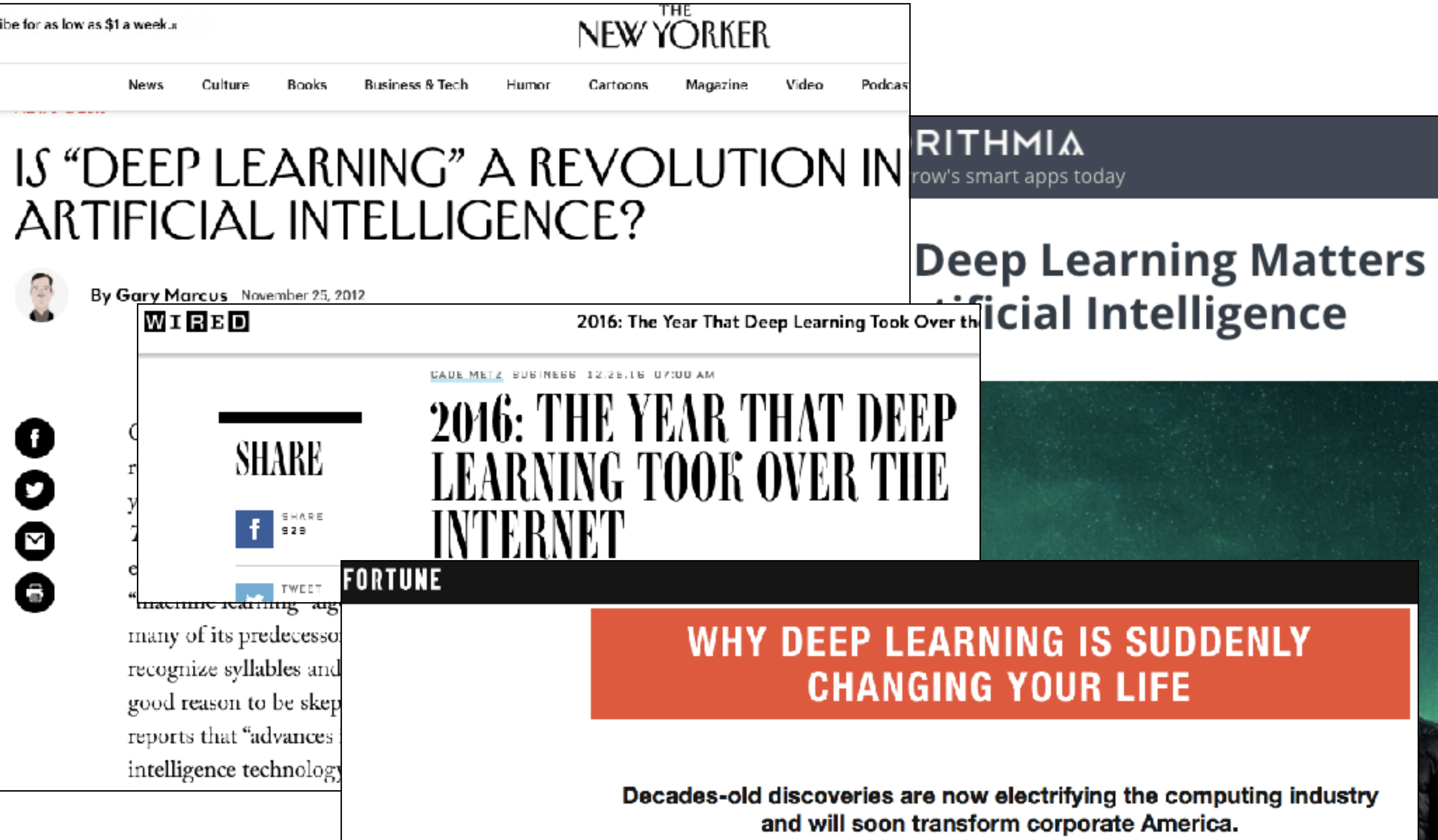
Speech Recognition

sample -> spectral band -> formant -> motif -> phone -> word

NLP

character -> word -> NP/VP/… -> clause -> sentence -> story

Deep Learning in Practice



ibe for as low as $1 a week.

THE NEW YORKER

News Culture Books Business & Tech Humor Cartoons Magazine Video Podcas

IS "DEEP LEARNING" A REVOLUTION IN ARTIFICIAL INTELLIGENCE?

By Gary Marcus November 25, 2012

many of its predecesso
recognize syllables and
good reason to be skep
reports that "advances
intelligence technolog

RITHMIA
row's smart apps today

Deep Learning Matters
icial Intelligence

WIRED

2016: The Year That Deep Learning Took Over th

CADE METZ BUSINESS 12.28.16 07:00 AM

SHARE

SHARE 929

TWEET

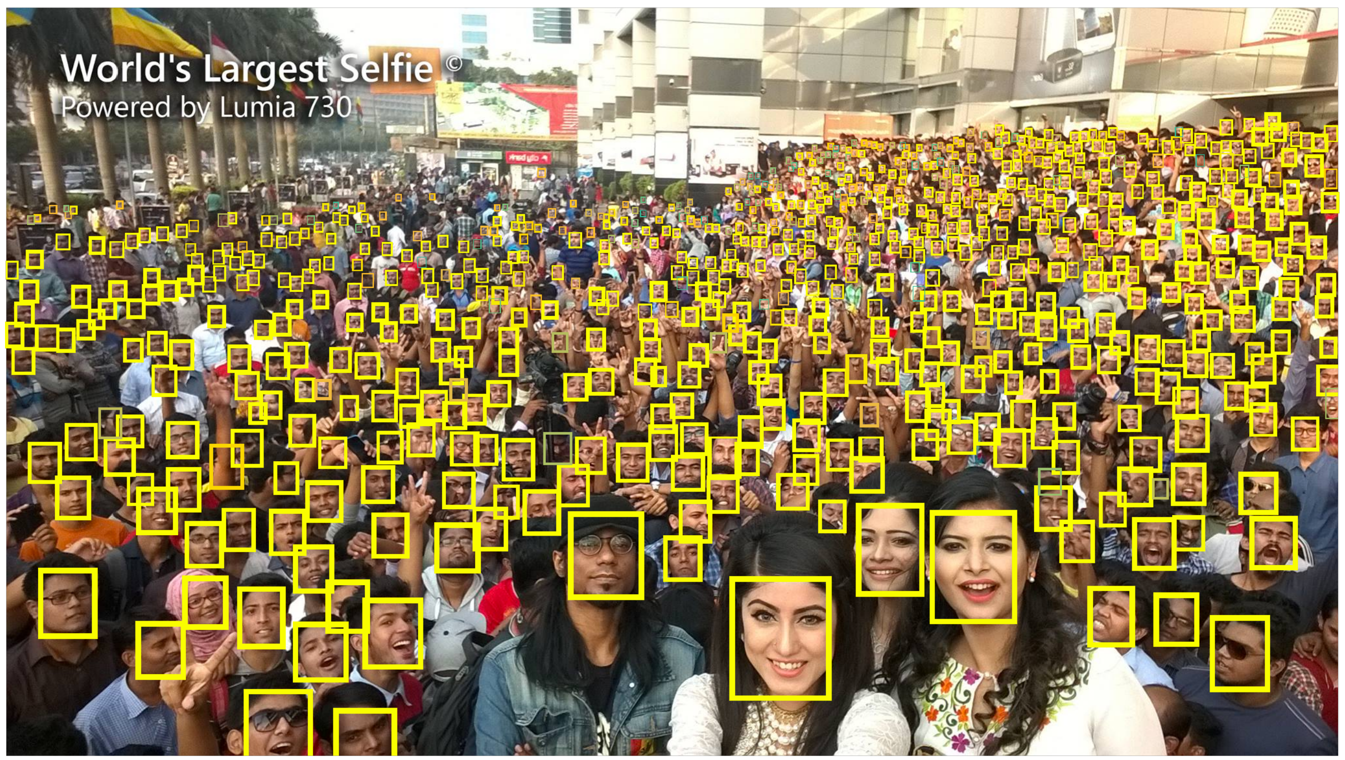
2016: THE YEAR THAT DEEP LEARNING TOOK OVER THE INTERNET

FORTUNE

WHY DEEP LEARNING IS SUDDENLY CHANGING YOUR LIFE

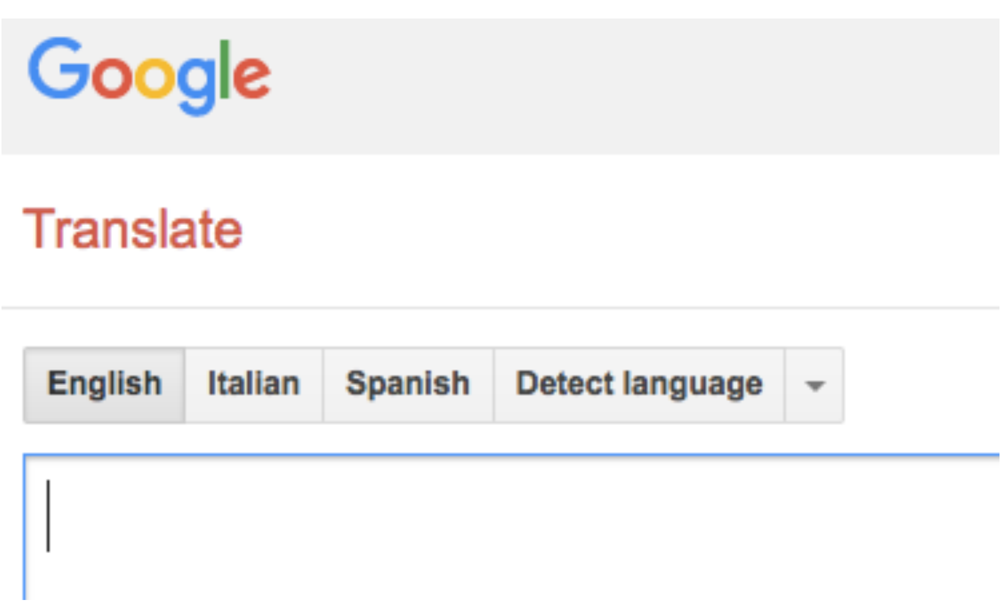
Decades-old discoveries are now electrifying the computing industry and will soon transform corporate America.

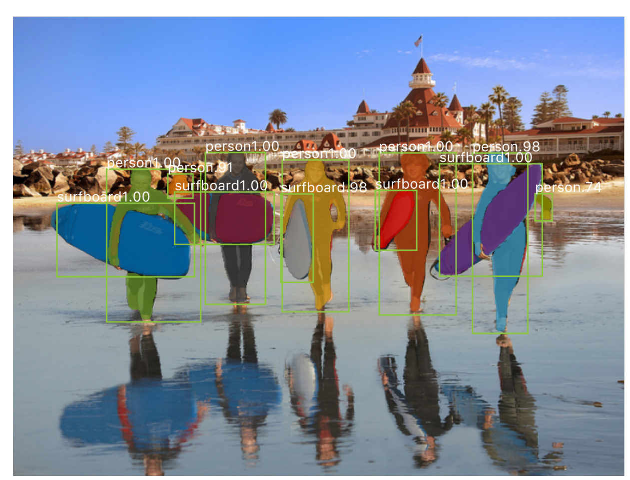
Deep Learning in Practice



Hu et al. "Finding tiny faces" 2016

ASR





He et al. "Mask R-CNN" 2017

Recap

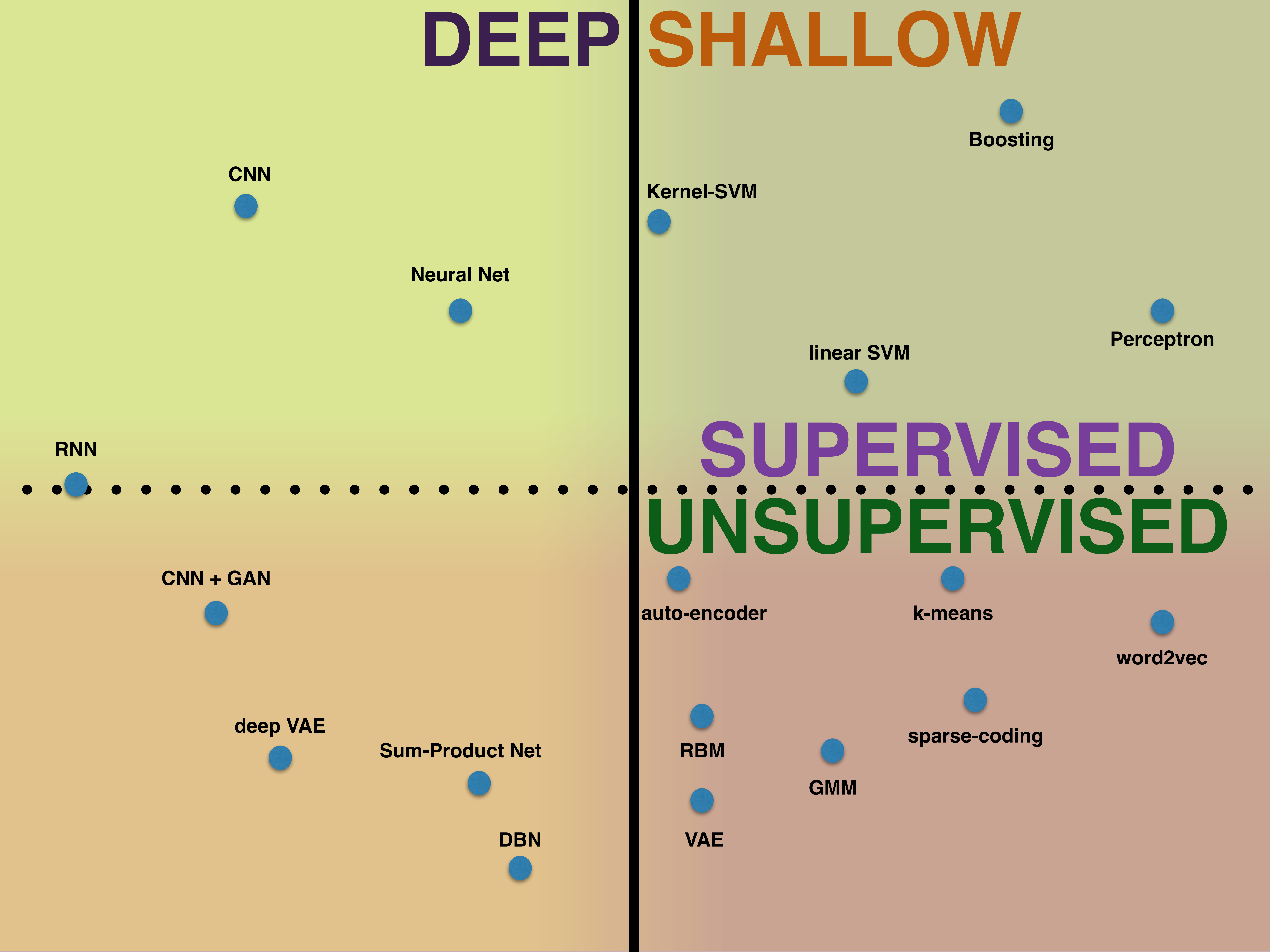
- Deep Learning = Methods to Learn Hierarchical Models.
- When data has intrinsic hierarchical structure, it’s natural to use model with similar inductive bias.
- Hierarchical Models are a useful tool for building AI.
- Lots of successful applications.

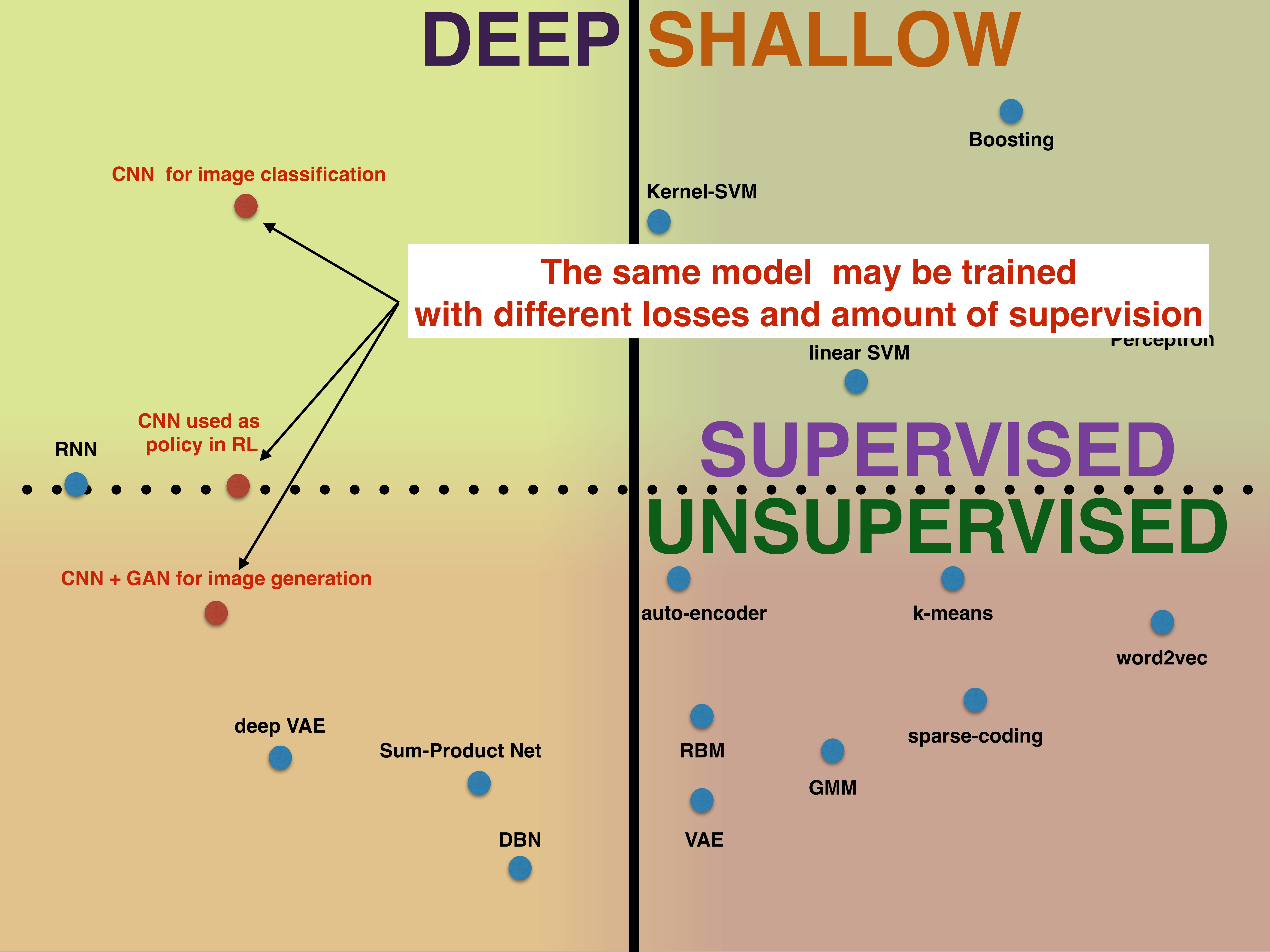
How many deep learning methods are out there?

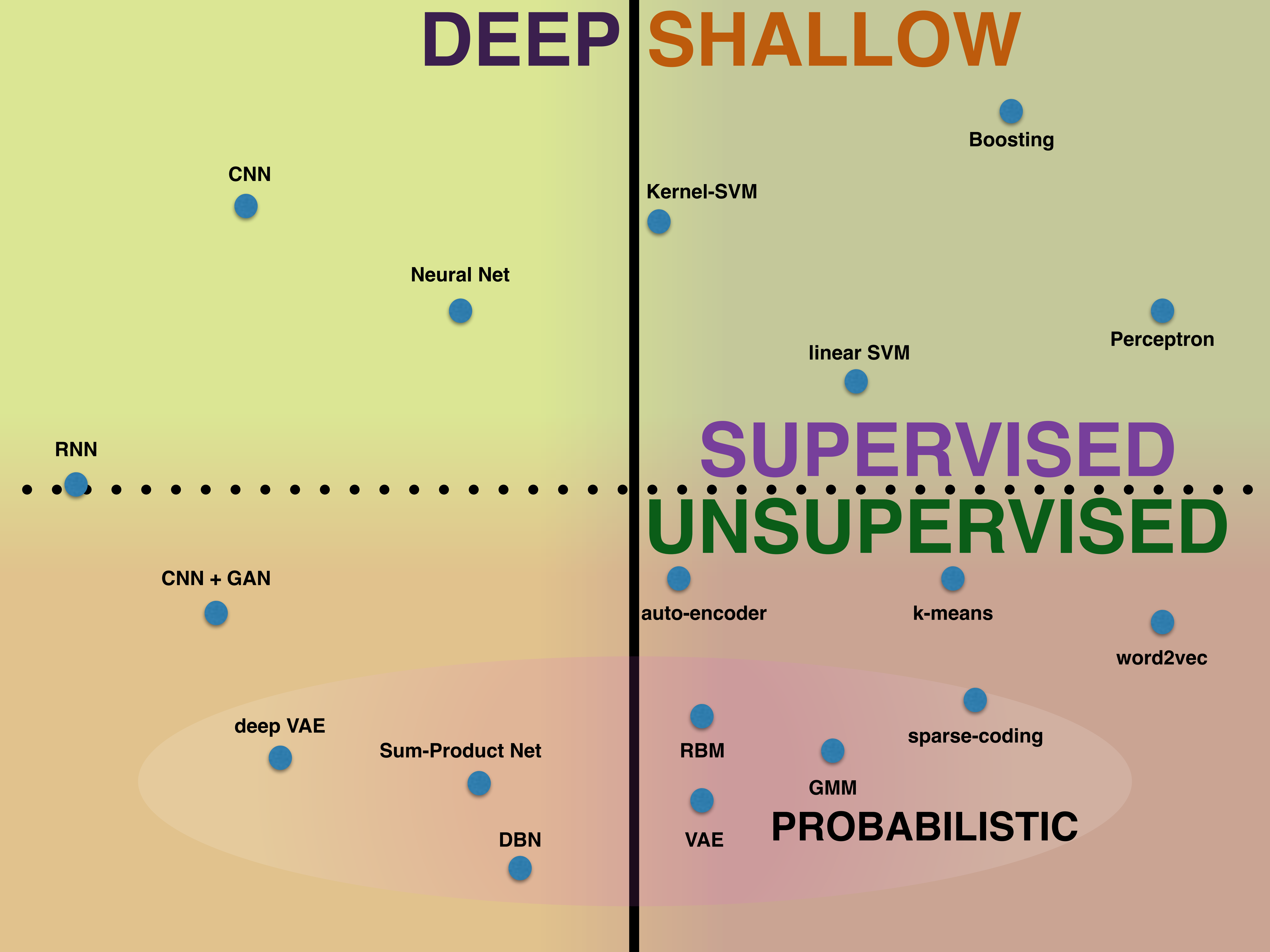
THE SPACE OF ML METHODS

Disclaimer: this is an over-simplified illustration!


DEEP
SHALLOW
CNN
Neural Net
RNN
CNN + GAN
deep VAE
Sum-Product Net
DBN
Boosting
Kernel-SVM
Perceptron
linear SVM
auto-encoder
k-means
word2vec
sparse-coding
RBM
GMM
VAE


DEEP
SHALLOW
Boosting
CNN
Kernel-SVM
Neural Net
Perceptron
linear SVM
RNN
SUPERVISED
UNSUPERVISED
CNN + GAN
auto-encoder
k-means
word2vec
deep VAE
Sum-Product Net
RBM
sparse-coding
GMM
DBN
VAE


DEEP
SHALLOW
Boosting
CNN for image classification
Kernel-SVM
The same model may be trained
with different losses and amount of supervision
Perceptron
linear SVM
SUPERVISED
RNN
CNN used as
policy in RL
UNSUPERVISED
CNN + GAN for image generation
auto-encoder
k-means
word2vec
deep VAE
Sum-Product Net
RBM
sparse-coding
GMM
VAE
DBN


DEEP
SHALLOW
CNN
Neural Net
RNN
Boosting
Kernel-SVM
Perceptron
linear SVM
SUPERVISED
UNSUPERVISED
CNN + GAN
auto-encoder
k-means
word2vec
deep VAE
Sum-Product Net
DBN
RBM
sparse-coding
GMM
VAE
PROBABILISTIC


DEEP
SHALLOW
Boosting
CNN
Kernel-SVM
Neural Net
Perceptron
linear SVM
RNN
SUPERVISED
UNSUPERVISED
CNN + GAN
auto-encoder
k-means
word2vec
deep VAE
Sum-Product Net
RBM
sparse-coding
GMM
DBN
VAE
PROBABILISTIC
Some of the methods we are going to discuss

Recap

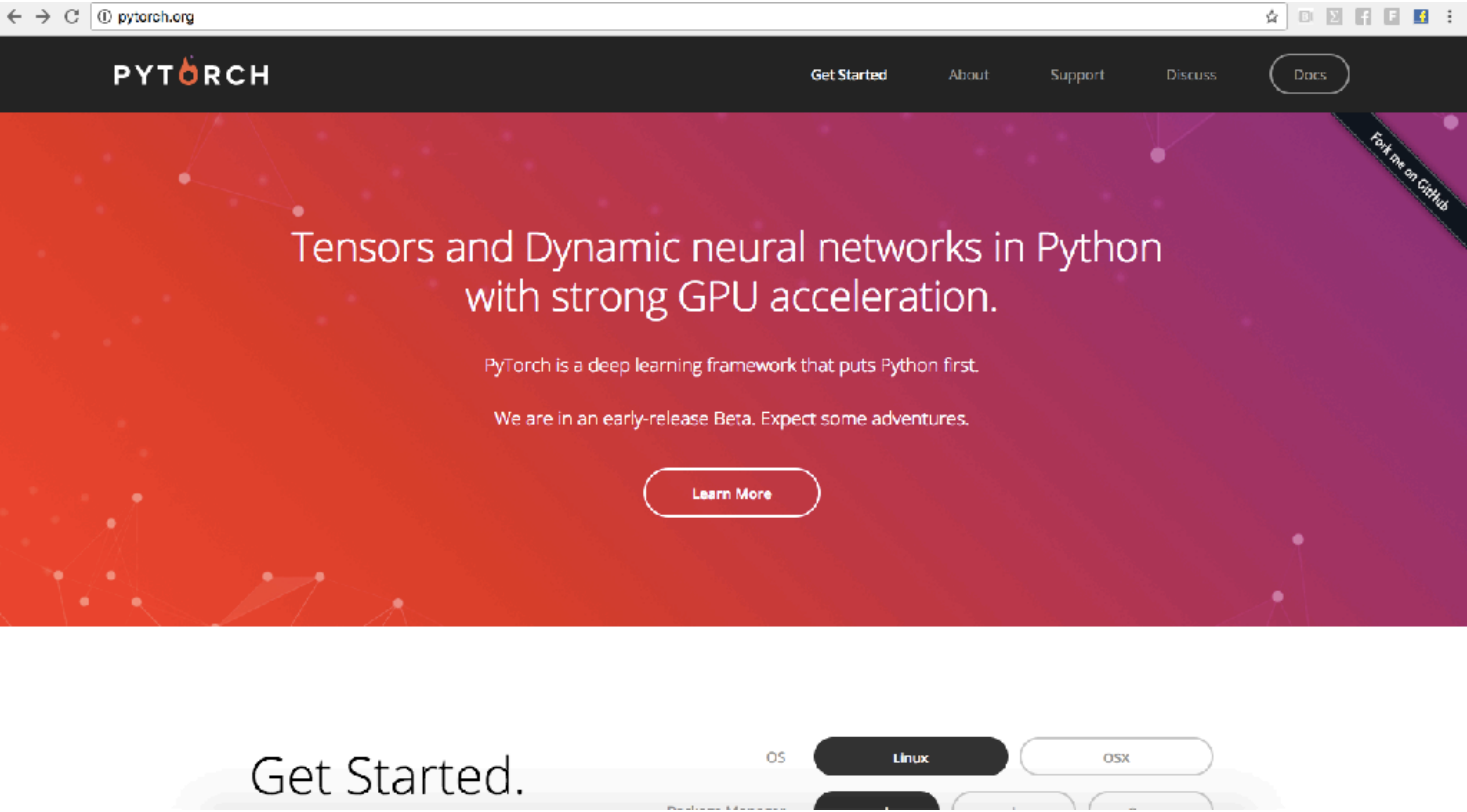
- Hierarchical models are a good tool for AI
- There are many ways to structure hierarchical models.
- Depending on the application (properties of the data and task to solve), hierarchical models may need to be more or less deep, and they may have particular structure / constraints.
- The amount of supervision strongly determines the training method.

Software Packages

- Caffe2: https://caffe2.ai/
- pyTorch: http://pytorch.org/
- TensorFlow: https://www.tensorflow.org/
- Theano: http://deeplearning.net/software/theano/
- Torch: http://torch.ch/

Software Packages

- Caffe2: https://caffe2.ai/
- **pyTorch**: http://pytorch.org/
- TensorFlow: https://www.tensorflow.org/
- Theano: http://deeplearning.net/software/theano/
- Torch: http://torch.ch/


pytorch.org
PYTORCH
Get Started
About
Support
Discuss
Docs
Fork me on GitHub
Tensors and Dynamic neural networks in Python with strong GPU acceleration.
PyTorch is a deep learning framework that puts Python first.
We are in an early-release Beta. Expect some adventures.
Learn More
Get Started.
OS
Linux
OSX

pytorch.org/tutorials/

PyTorch Tutorials

PYTORCH

0.1.12_2

Search docs

Welcome to PyTorch Tutorials

To get started with learning PyTorch, start with our Beginner Tutorials. The 60-minute blitz is the most common starting point, and gives you a quick introduction to PyTorch. If you like learning by examples, you will like the tutorial Learning PyTorch with Examples

If you would like to do the tutorials interactively via IPython / Jupyter, each tutorial has a download link for a Jupyter Notebook and Python source code.

We also provide a lot of high-quality examples covering image classification, unsupervised learning, reinforcement learning, machine translation and many other applications at https://github.com/pytorch/examples/

You can find reference documentation for PyTorch's API and layers at http://docs.pytorch.org or via inline help.

If you would like the tutorials section improved, please open a github issue here with your feedback: https://github.com/pytorch/tutorials

Beginner Tutorials

PYTORCH torch Examples

pytorch.org/tutorials/#

GitHub, Inc. [US] https://github.com/pytorch/examples

PyTorch Examples

A repository showcasing examples of using pytorch

- MNIST Convnets
- Word level Language Modeling using LSTM RNNs
- Training Imagenet Classifiers with Residual Networks
- Generative Adversarial Networks (DCGAN)
- Variational Auto-Encoders
- Superresolution using an efficient sub-pixel convolutional neural network
- Hogwild training of shared ConvNets across multiple processes on MNIST
- Training a CartPole to balance in OpenAI Gym with actor-critic
- Natural Language Inference (SNLI) with GloVe vectors, LSTMs, and torchtext
- Time sequence prediction - create an LSTM to learn Sine waves

Additionally, a list of good examples hosted in their own repositories:

- Neural Machine Translation using sequence-to-sequence RNN with attention (OpenNMT)

Outline

- **PART 0** [lecture 1]
  - Motivation
  - Training Fully Connected Nets with Backpropagation
- **Part 1** [lecture 1 and lecture 2]
  - Deep Learning for Vision: CNN
- **Part 2** [lecture 2]
  - Deep Learning for NLP
- **Part 3** [lecture 3]
  - Modeling sequences

Outline

- **PART 0** [lecture 1]
  - Motivation
  - **Training Fully Connected Nets with Backpropagation**
- **Part 1** [lecture 1 and lecture 2]
  - Deep Learning for Vision: CNN
- **Part 2** [lecture 2]
  - Deep Learning for NLP
- **Part 3** [lecture 3]
  - Modeling sequences

Neural Networks

Assumptions (for the next few slides):
- The input image is vectorized (disregard the spatial layout of pixels)
- The target label is discrete (classification)

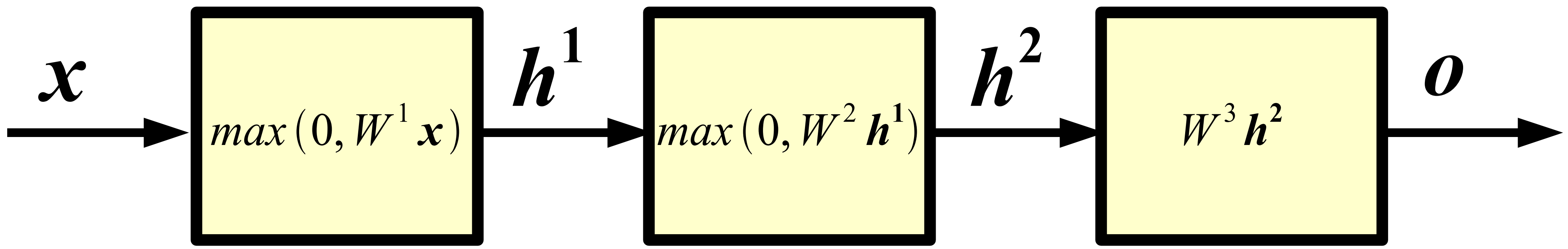
Question: what class of functions shall we consider to map the input into the output?

Answer: composition of simpler functions.

Follow-up questions: Why not a linear combination? What are the "simpler" functions? What is the interpretation?

Answer: later...

Neural Networks: example



$\boldsymbol{x}$ input

$\boldsymbol{h}^1$ 1-st layer hidden units

$\boldsymbol{h}^2$ 2-nd layer hidden units

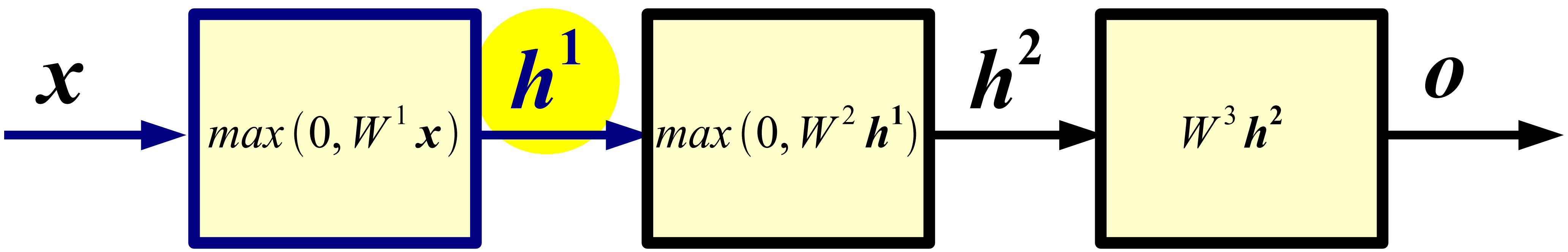
$\boldsymbol{o}$ output

Example of a 2 hidden layer neural network (or 4 layer network, counting also input and output).

Forward Propagation

Def.: Forward propagation is the process of computing the output of the network given its input.

Forward Propagation



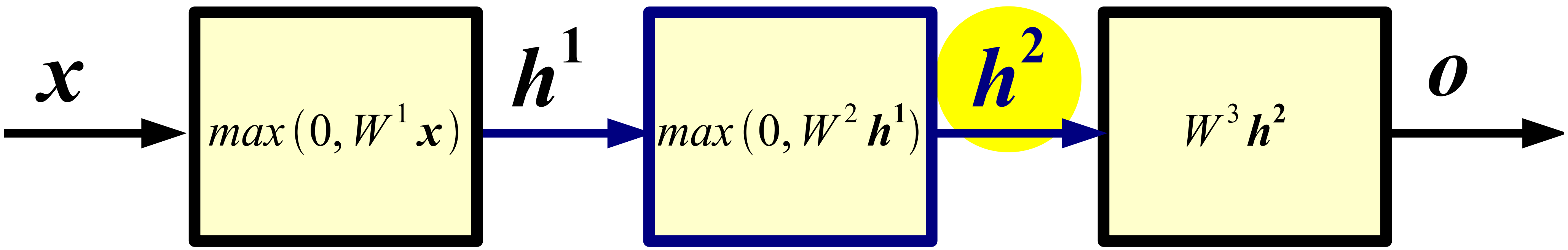
$\boldsymbol{x} \in R^D \quad W^1 \in R^{N_1 \times D} \quad \boldsymbol{b}^1 \in R^{N_1} \quad \boldsymbol{h}^1 \in R^{N_1}$

$$\boldsymbol{h}^1 = max(0, W^1 \boldsymbol{x} + \boldsymbol{b}^1)$$

W^1 1-st layer weight matrix or weights
$\boldsymbol{b}^1$ 1-st layer biases

The non-linearity $u = max(0, v)$ is called **ReLU** in the DL literature. Each output hidden unit takes as input all the units at the previous layer: each such layer is called "**fully connected**".

Forward Propagation



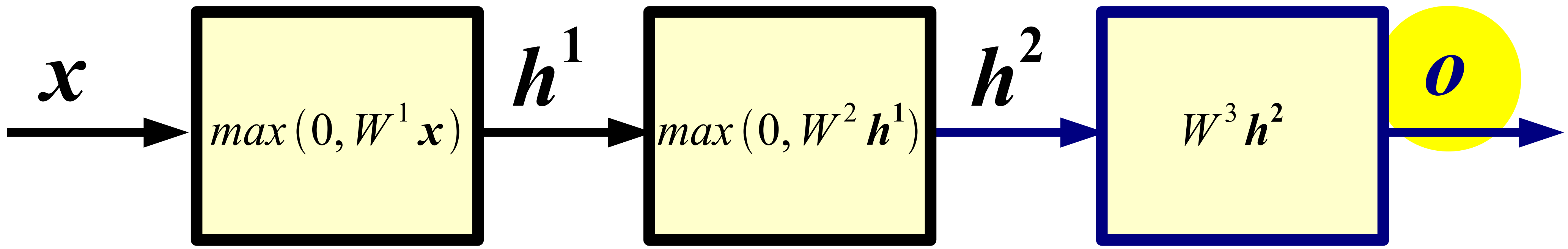
$\boldsymbol{h}^1 \in R^{N_1}$ $\quad W^2 \in R^{N_2 \times N_1}$ $\quad \boldsymbol{b}^2 \in R^{N_2}$ $\quad \boldsymbol{h}^2 \in R^{N_2}$

$$\boldsymbol{h}^2 = max(0, W^2 \boldsymbol{h}^1 + \boldsymbol{b}^2)$$

W^2 2-nd layer weight matrix or weights

$\boldsymbol{b}^2$ 2-nd layer biases

Forward Propagation

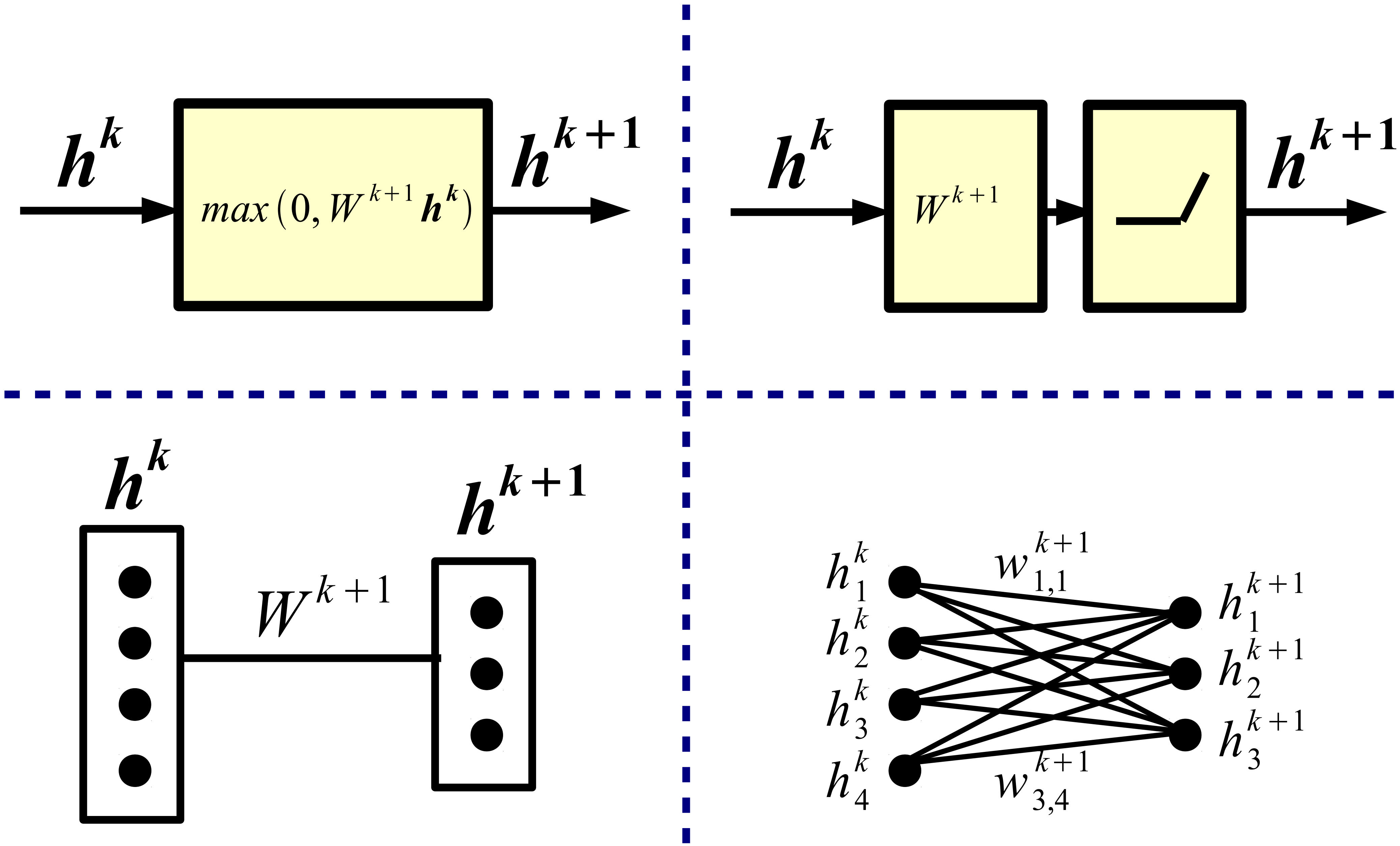


$\boldsymbol{h}^2 \in R^{N_2}$ $W^3 \in R^{N_3 \times N_2}$ $\boldsymbol{b}^3 \in R^{N_3}$ $\boldsymbol{o} \in R^{N_3}$

$$\boldsymbol{o} = max(0, W^3 \boldsymbol{h}^2 + \boldsymbol{b}^3)$$

W^3 3-rd layer weight matrix or weights
$\boldsymbol{b}^3$ 3-rd layer biases

Alternative Graphical Representation



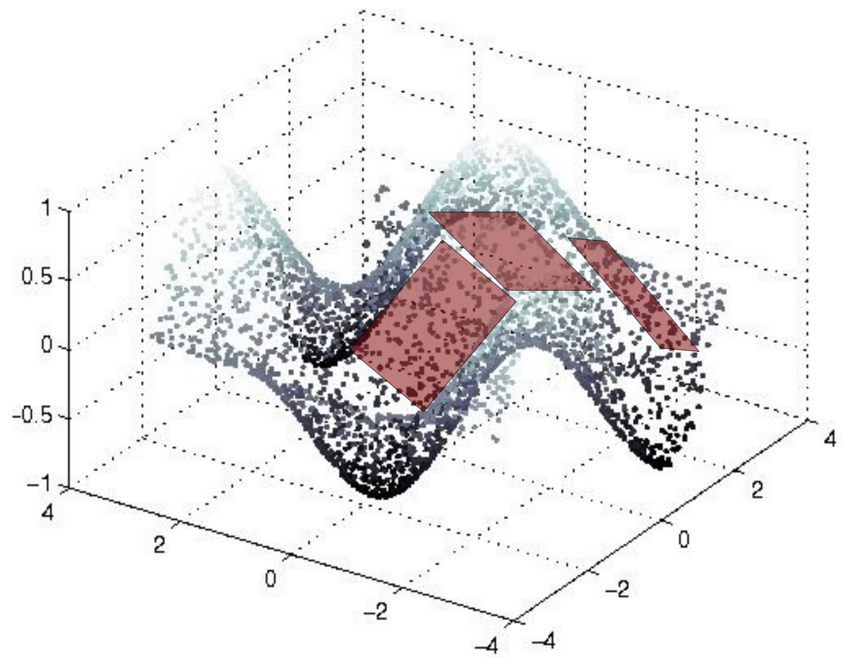
Interpretation

Question: Why can't the mapping between layers be linear?

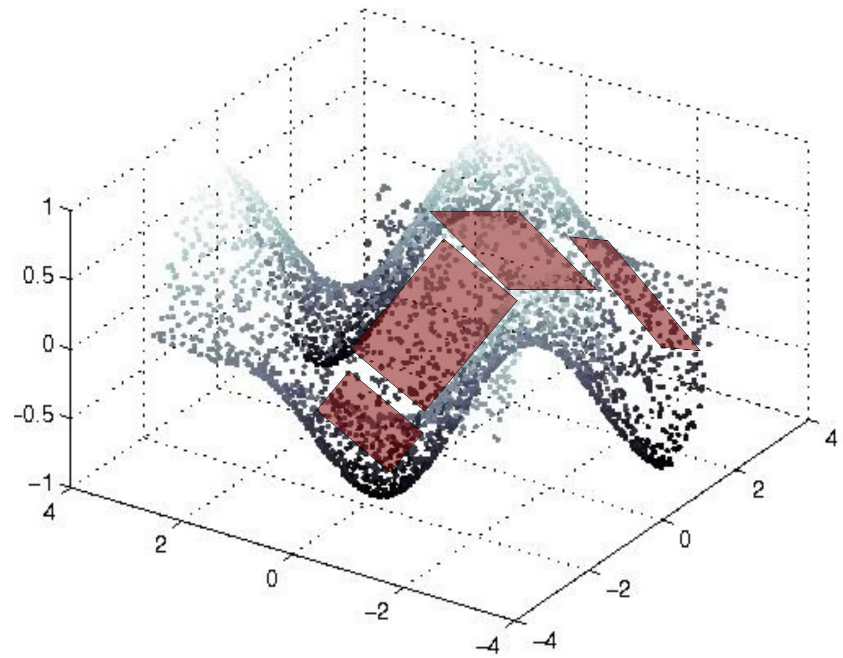
Answer: Because composition of linear functions is a linear function. Neural network would reduce to (1 layer) logistic regression.

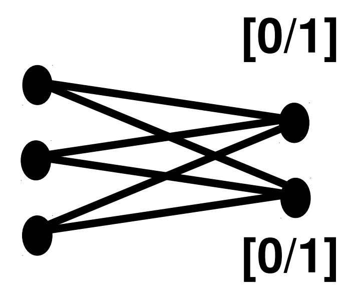
Question: What do ReLU layers accomplish?

Answer: Piece-wise linear tiling: mapping is locally linear.

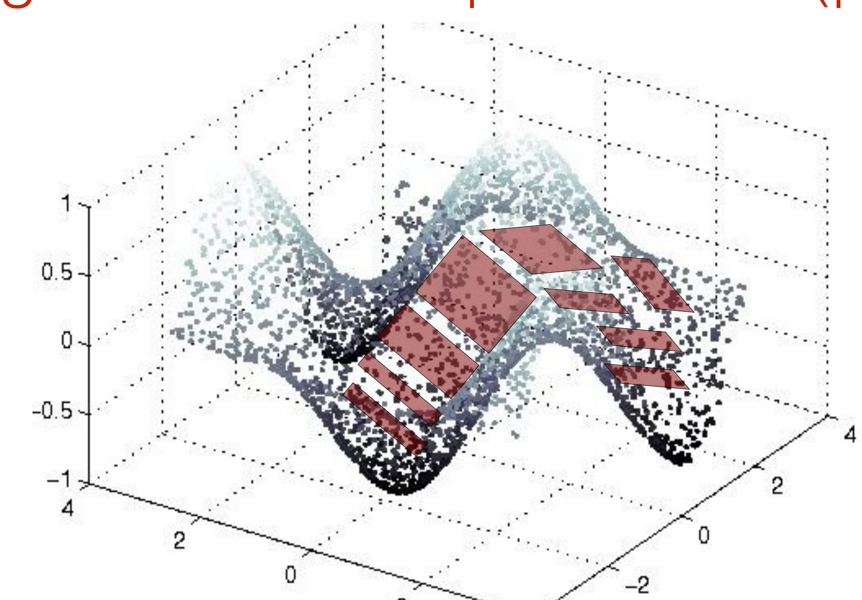


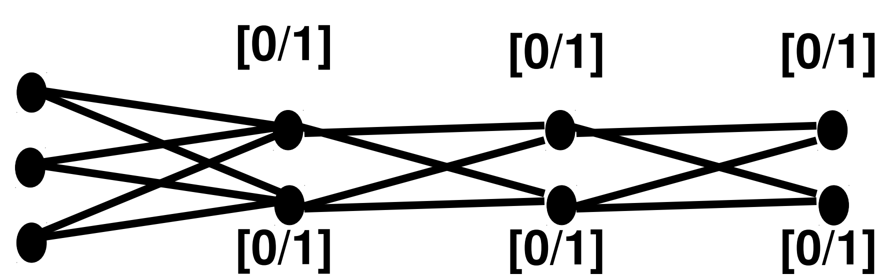
Montufar et al. "On the number of linear regions of DNNs" arXiv 2014





ReLU layers do local linear approximation. Number of planes grows exponentially with number of hidden units. Multiple layers yield exponential savings in number of parameters (parameter sharing).





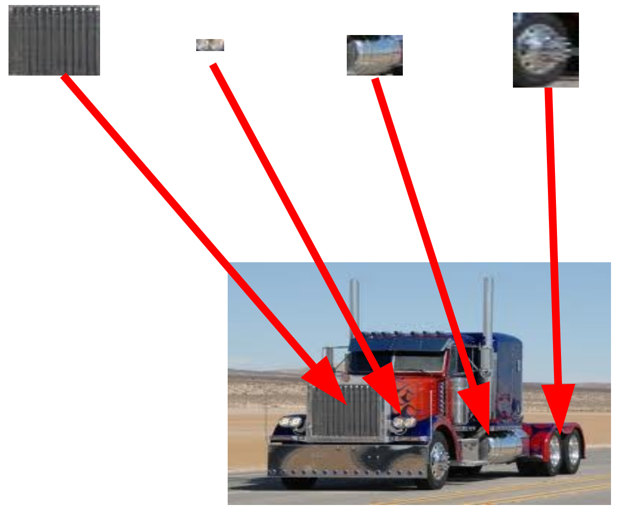
Montufar et al. "On the number of linear regions of DNNs" arXiv 2014

Interpretation

Question: Why do we need many layers?

Answer: When input has hierarchical structure, the use of a hierarchical architecture is potentially more efficient because intermediate computations can be re-used. DL architectures are efficient also because they use **distributed representations** which are shared across classes.

[0 0 **1** 0 0 0 0 **1** 0 0 **1** **1** 0 0 **1** 0 ...] truck feature

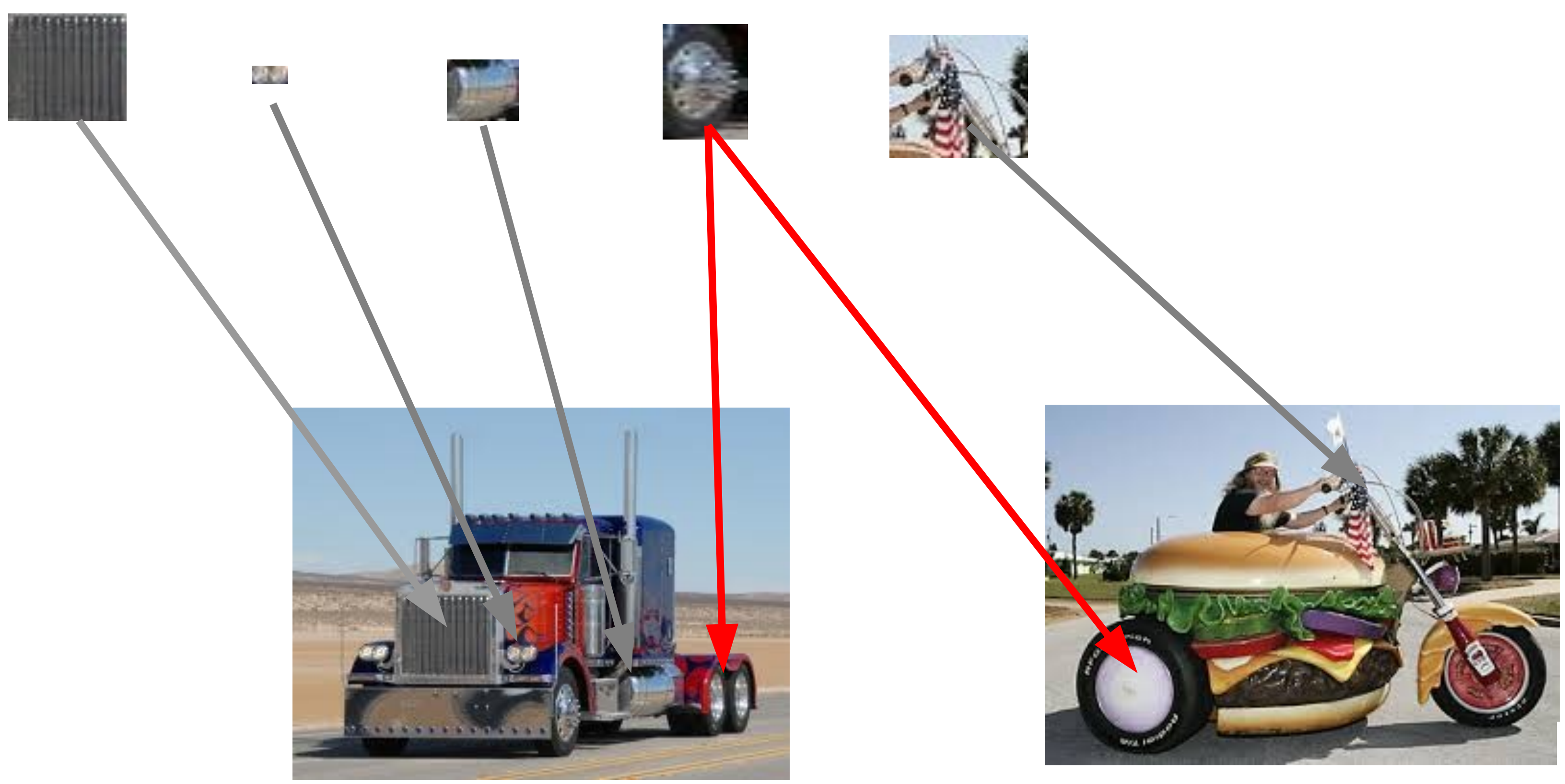


Exponentially more efficient than a 1-of-N representation (a la k-means)

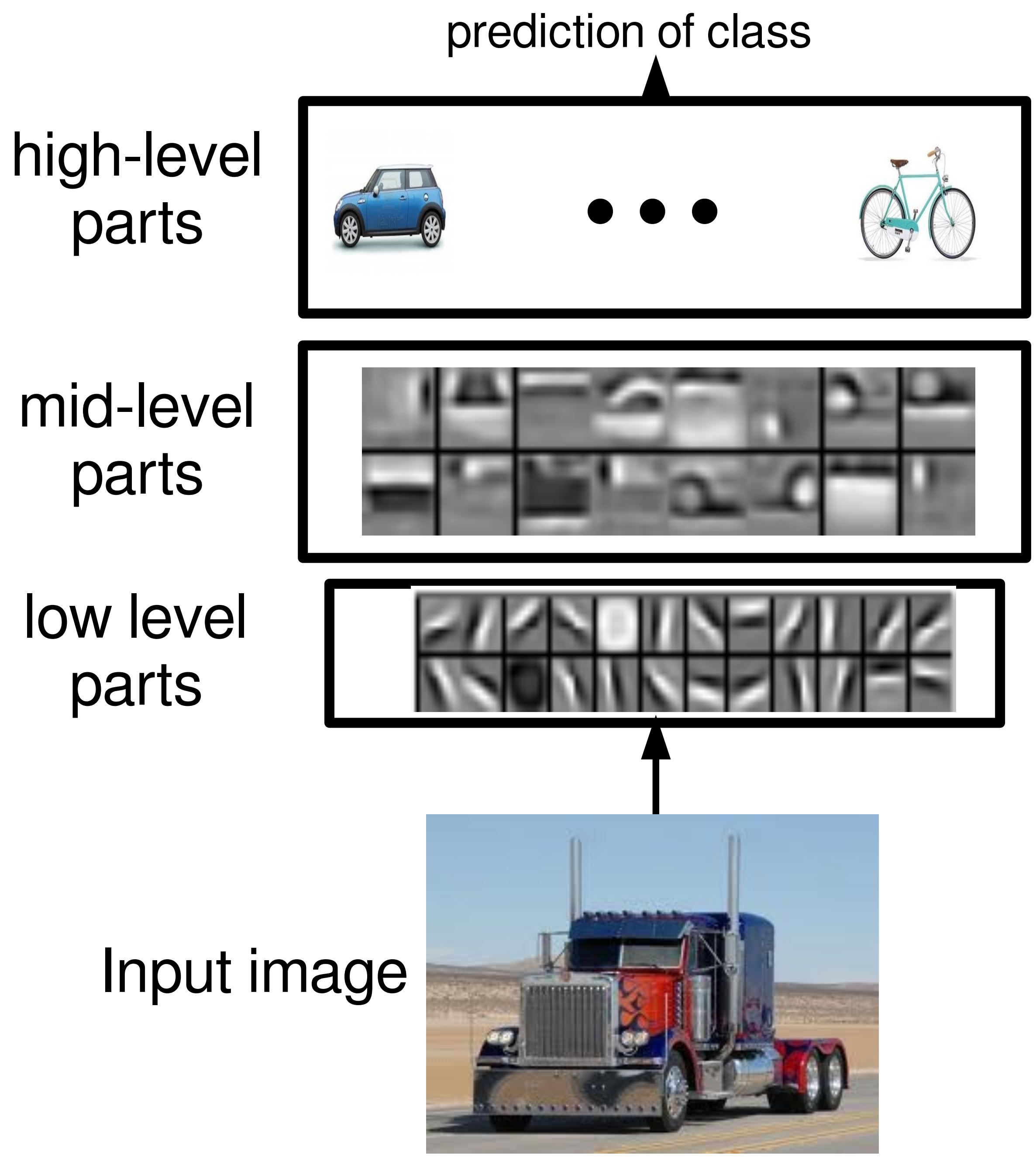
Interpretation

[1 1 0 0 0 1 0 **1** 0 0 0 0 1 1 0 1...] motorbike

[0 0 1 0 0 0 0 **1** 0 0 1 1 0 0 1 0 ...] truck



Interpretation



- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009

pyTorch demo

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

ndim = 1
nhid = 200
nout = 1
nsamples = 1000
net = torch.nn.Sequential(nn.Linear(ndim, nhid), nn.ReLU(),
                          nn.Linear(nhid, nhid), nn.ReLU(), nn.Linear(nhid, nout))
print(net)
inputs = torch.arange(-3,3,0.01).view(-1, 1)
outputs = net.forward(Variable(inputs))

fig, ax = plt.subplots()
ax.plot(inputs.squeeze().numpy(), outputs.data.squeeze().numpy())
plt.show()
```

Interpretation

Question: What does a hidden unit do?

Answer: It can be thought of as a classifier or feature detector.

Question: How many layers? How many hidden units?

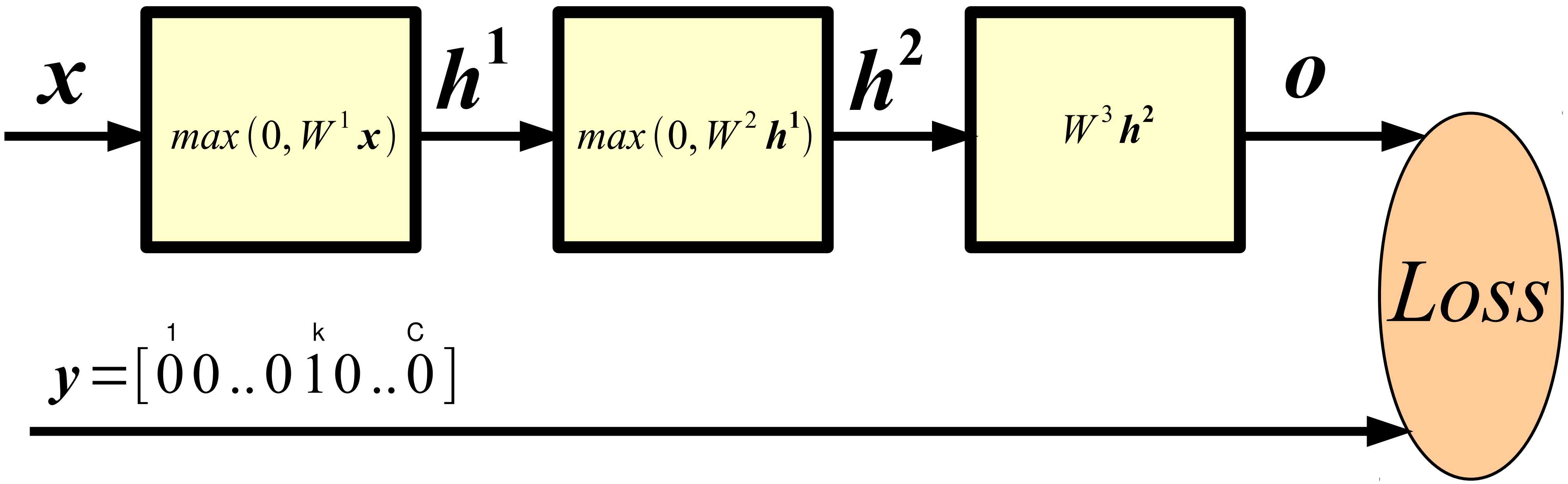
Answer: Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

Question: How do I set the weight matrices?

Answer: Weight matrices and biases are learned.
First, we need to define a measure of quality of the current mapping.
Then, we need to define a procedure to adjust the parameters.

Disclaimer: these are just suggestive conjectures. In practice, a fully connected net (as deep as you wish) has never worked well in vision/audio processing. We will shortly discuss how and what makes this work in practice…

How Good is a Network?



Probability of class k given input (softmax):

$$p(c_k = 1 | \boldsymbol{x}) = \frac{e^{o_k}}{\sum_{j=1}^{C} e^{o_j}}$$

(Per-sample) **Loss**; e.g., negative log-likelihood (good for classification of small number of classes):

$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j | \boldsymbol{x})$$ **Cross-Entropy Loss**

Training

Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

$$\boldsymbol{\theta}^* = arg\ min_{\boldsymbol{\theta}} \sum_{n=1}^{P} L(\boldsymbol{x}^{\boldsymbol{n}}, y^n; \boldsymbol{\theta})$$

Question: How to minimize a complicated function of the parameters?

Answer: Chain rule, a.k.a. **Backpropagation**! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.

Rumelhart et al. “Learning internal representations by back-propagating..” Nature 1986

Derivative w.r.t. Input of Softmax

$$p(c_k = 1 | \boldsymbol{x}) = \frac{e^{o_k}}{\sum_j e^{o_j}}$$

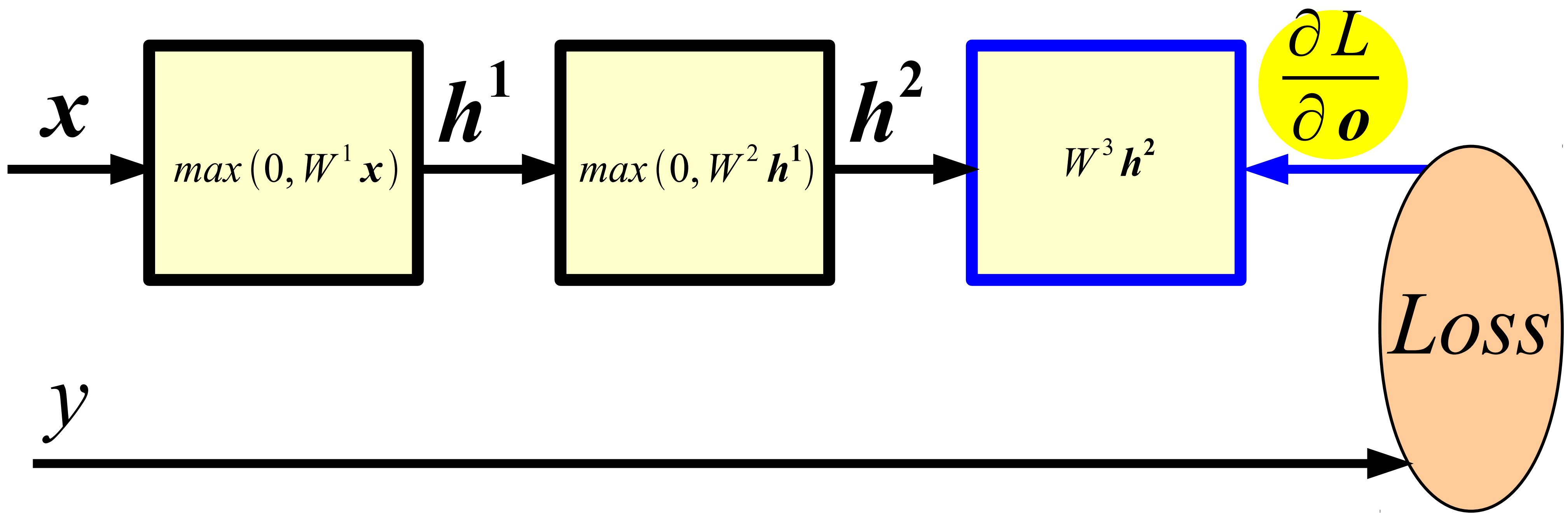
$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j | \boldsymbol{x}) \qquad \boldsymbol{y} = [\overset{1}{0}\,0\,..\,0\,\overset{k}{1}\,0\,..\,\overset{C}{0}]$$

By substituting the first formula in the second one, and taking the derivative w.r.t. $\boldsymbol{o}$ we get:

$$\frac{\partial L}{\partial o} = p(c | \boldsymbol{x}) - \boldsymbol{y}$$

HOMEWORK: prove it!

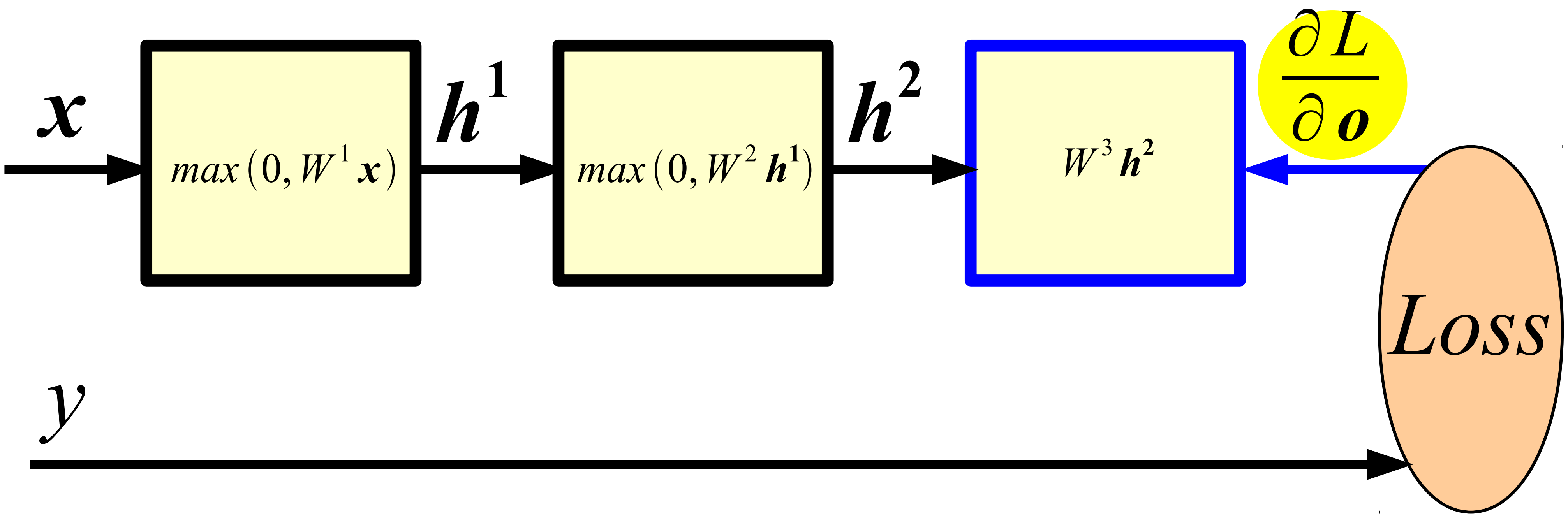
Backward Propagation



Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3}$$

Backward Propagation

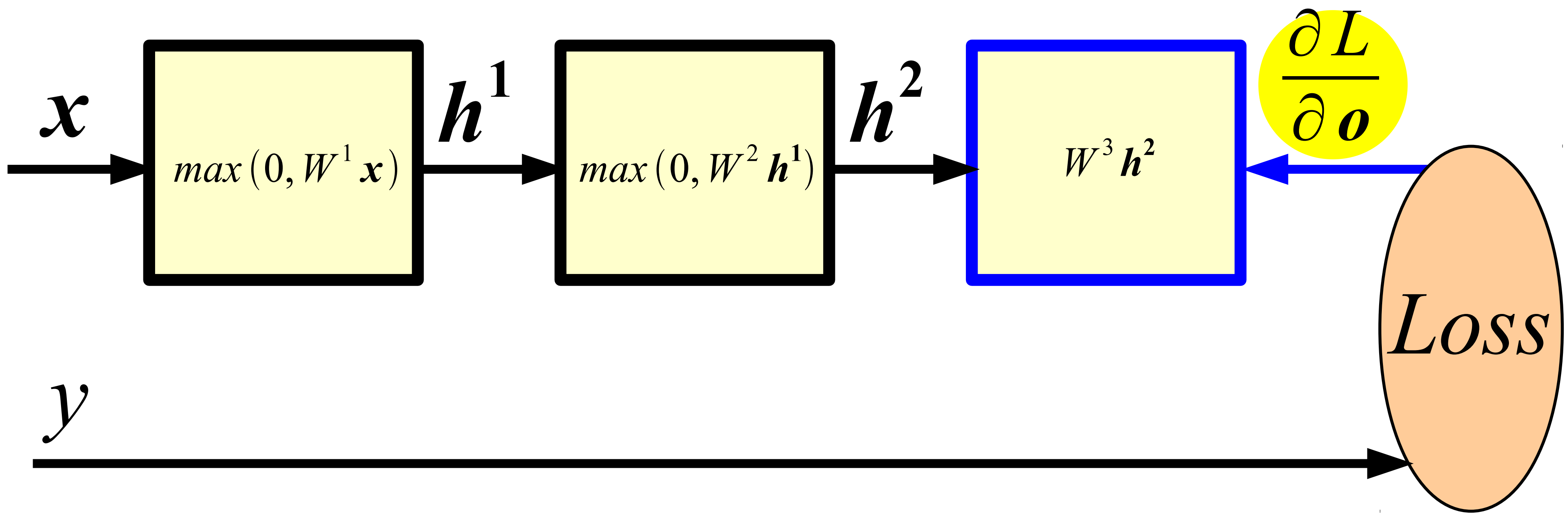


Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\boldsymbol{x}) - \boldsymbol{y}) \; \boldsymbol{h}^{2\,T}$$

Backward Propagation

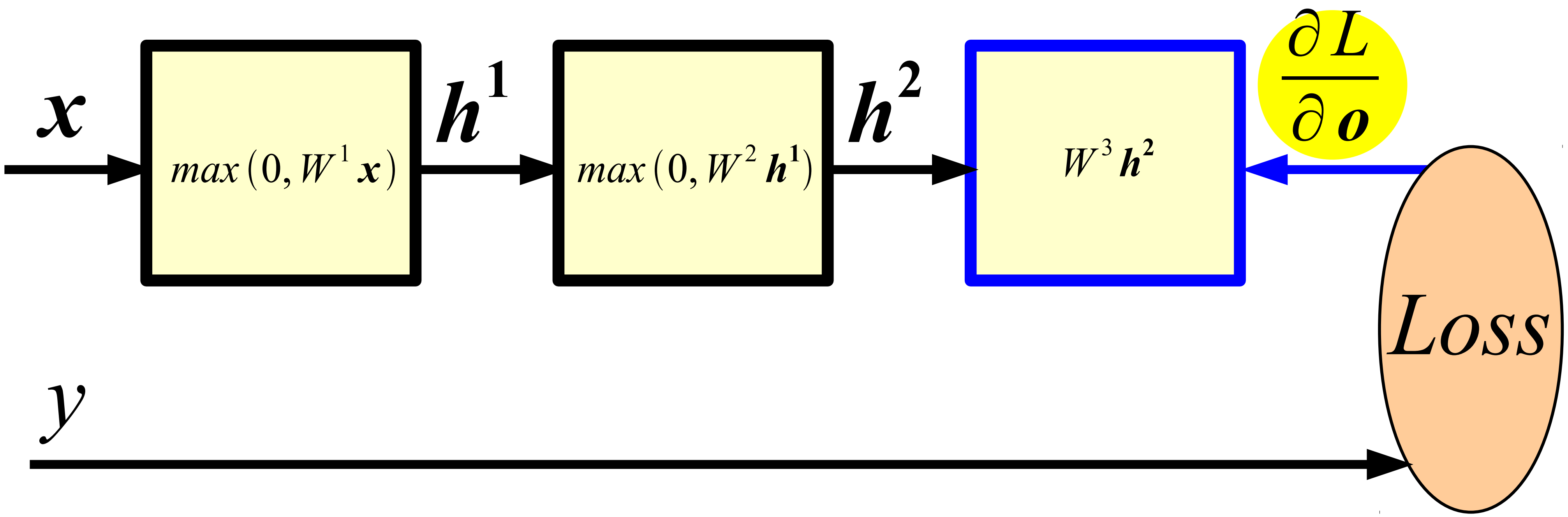


Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\boldsymbol{x}) - \boldsymbol{y}) \; \boldsymbol{h}^{2\,T}$$

Backward Propagation

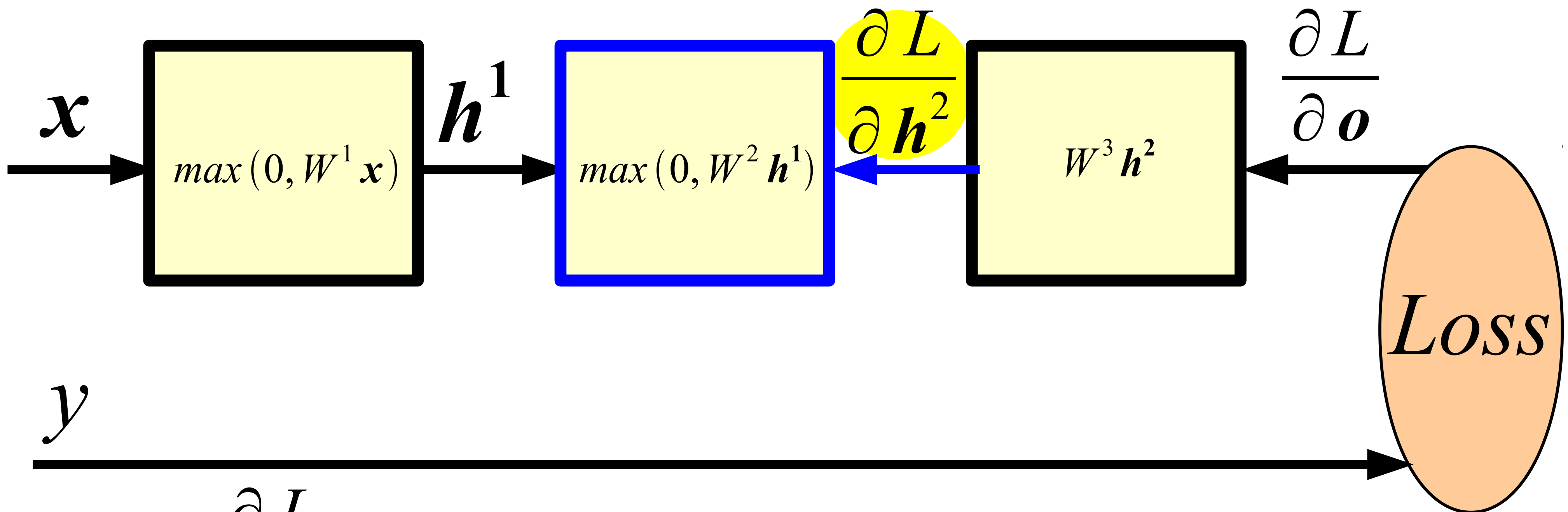


Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\boldsymbol{x}) - \boldsymbol{y})\ \boldsymbol{h}^{2\,T} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = W^{3\,T}(p(c|\boldsymbol{x}) - \boldsymbol{y})$$

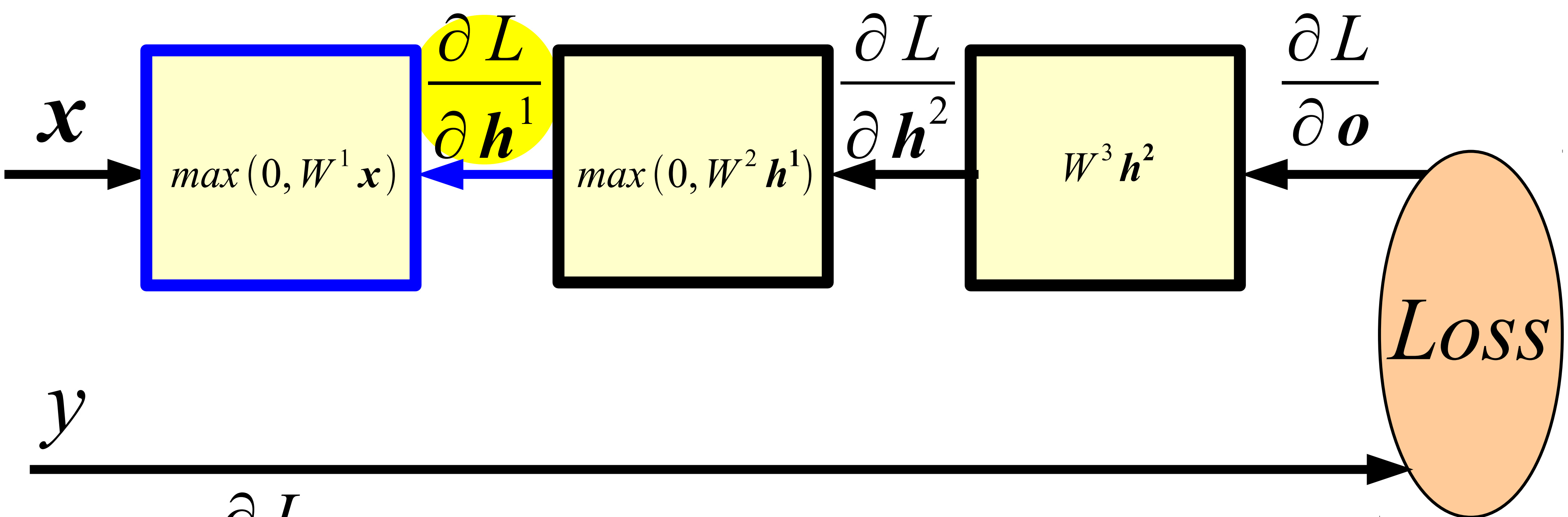
Backward Propagation



Given $\frac{\partial L}{\partial \boldsymbol{h}^2}$ we can compute now:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial \boldsymbol{h}^2} \frac{\partial \boldsymbol{h}^2}{\partial W^2} \qquad \frac{\partial L}{\partial \boldsymbol{h}^1} = \frac{\partial L}{\partial \boldsymbol{h}^2} \frac{\partial \boldsymbol{h}^2}{\partial \boldsymbol{h}^1}$$

Backward Propagation



Given $\frac{\partial L}{\partial \boldsymbol{h}^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial \boldsymbol{h}^1} \frac{\partial \boldsymbol{h}^1}{\partial W^1}$$

Backward Propagation

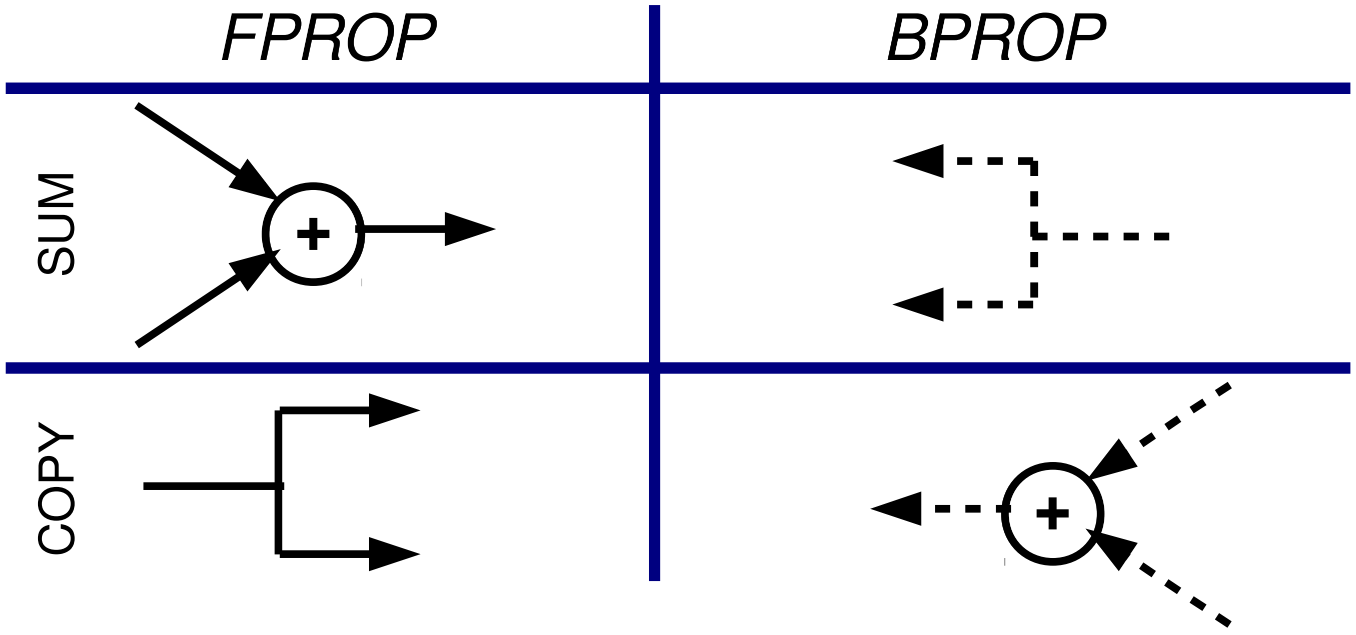
Question: Does BPROP work with ReLU layers only?

Answer: Nope, any a.e. differentiable transformation works.

Question: What's the computational cost of BPROP?

Answer: About twice FPROP (need to compute gradients w.r.t. input and parameters at every layer).

Note: FPROP and BPROP are dual of each other. E.g.,:



Optimization

more GPU friendly

Stochastic Gradient Descent (on mini-batches):

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial L}{\partial \boldsymbol{\theta}}, \eta \in (0, 1)$$

Stochastic Gradient Descent with Momentum:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{\Delta}$$

$$\boldsymbol{\Delta} \leftarrow 0.9 \boldsymbol{\Delta} + \frac{\partial L}{\partial \boldsymbol{\theta}}$$

Note: there are many other variants...

Optimization

Stochastic Gradient Descent (on mini-batches):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in (0, 1)$$

works always surprisingly well; learning rate should be annealed over time.

Stochastic Gradient Descent with Momentum:

$$\theta \leftarrow \theta - \eta \Delta$$

$$\Delta \leftarrow 0.9 \Delta + \frac{\partial L}{\partial \theta}$$

accelerates initial convergence at the beginning of training.

Note: there are many other variants...

there are 2nd order methods which take into account curvature, but so far they have never worked consistently better in terms of generalization. Optimization is surprisingly easy.

Recap

- Neural Net is a chain of non-linear operations, implementing highly non-linear functions.
- Forward pass computes the error.
- Backward pass computes gradients w.r.t. inputs at each layer and parameters.
- Optimization done by vanilla stochastic gradient descent.

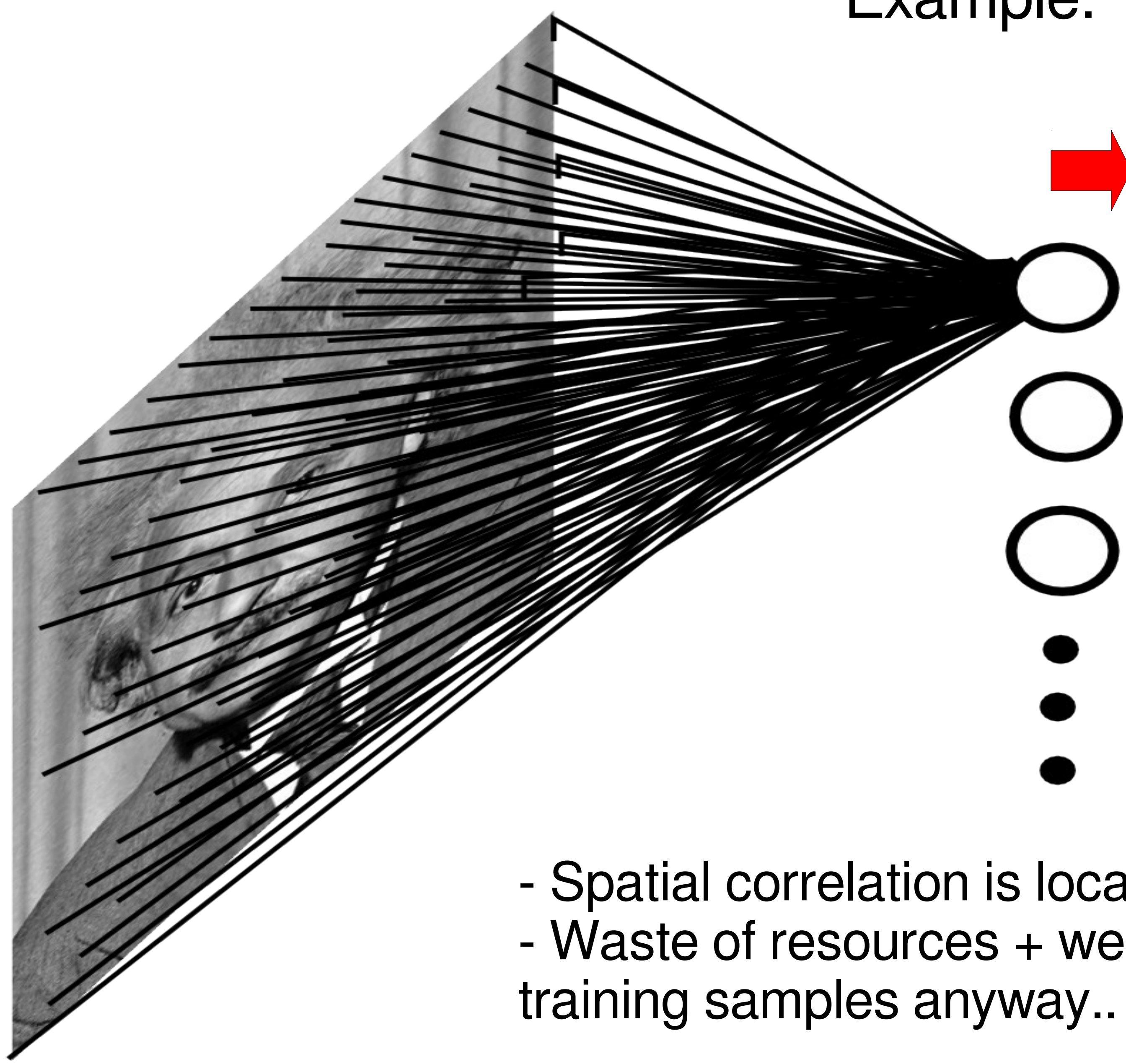
```python
import torch
from torch.autograd import Variable


class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        """
        In the constructor we instantiate two nn.Linear modules and assign them as
        member variables.
        """
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred


# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold inputs and outputs, and wrap them in Variables
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

# Construct our model by instantiating the class defined above
model = TwoLayerNet(D_in, H, D_out)

# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.MSELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x)

    # Compute and print loss
    loss = criterion(y_pred, y)
    print(t, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Question: How does all of this apply to vision?

Outline

- **PART 0** [lecture 1]
    - Motivation
    - Training Fully Connected Nets with Backpropagation
- **Part 1** [lecture 1 and lecture 2]
    - **Deep Learning for Vision: CNN**
- **Part 2** [lecture 2]
    - Deep Learning for NLP: word embeddings
- **Part 3** [lecture 3]
    - Modeling sequences: RNNs and Graph Transformer Networks

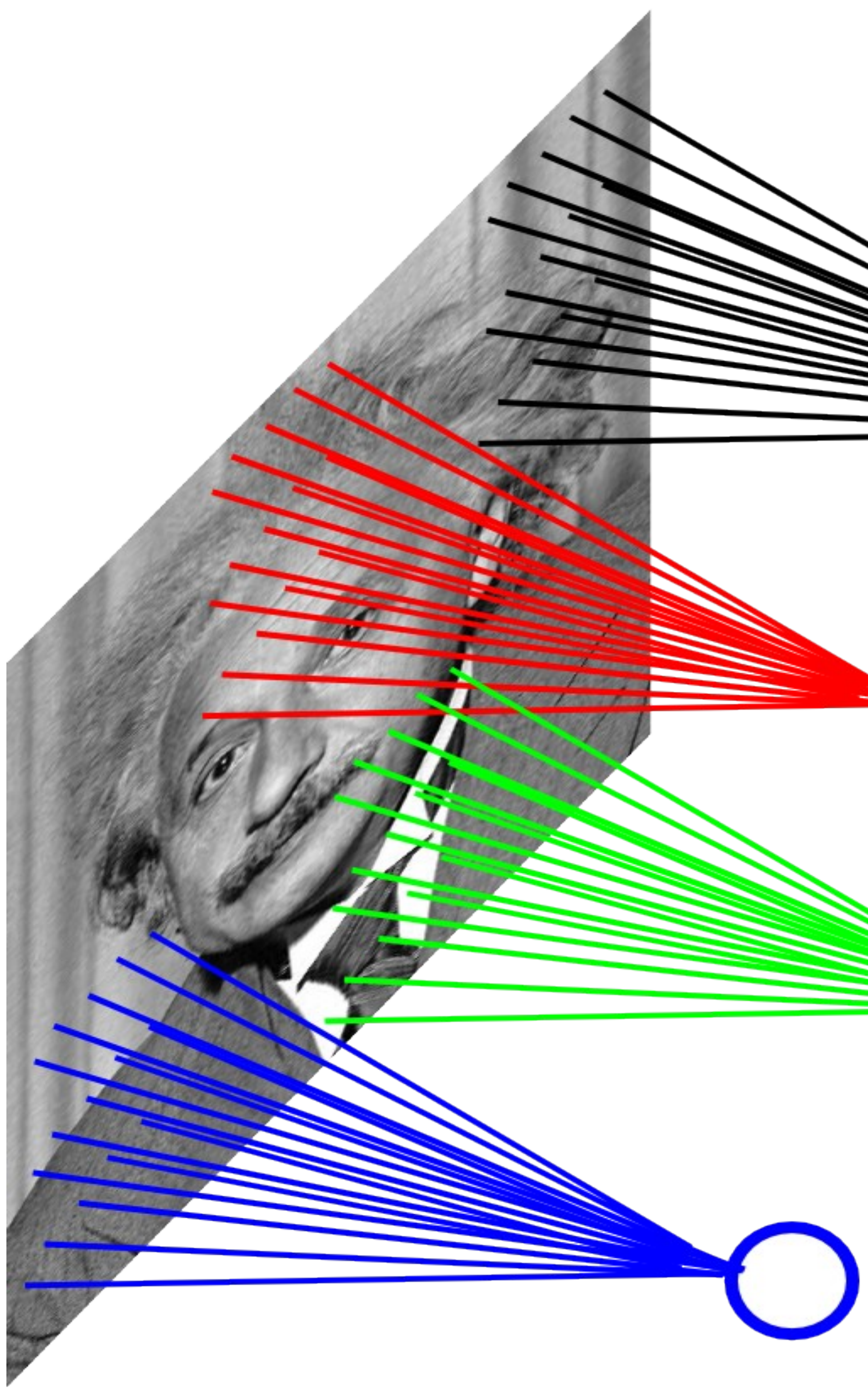
Fully Connected Layer



Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

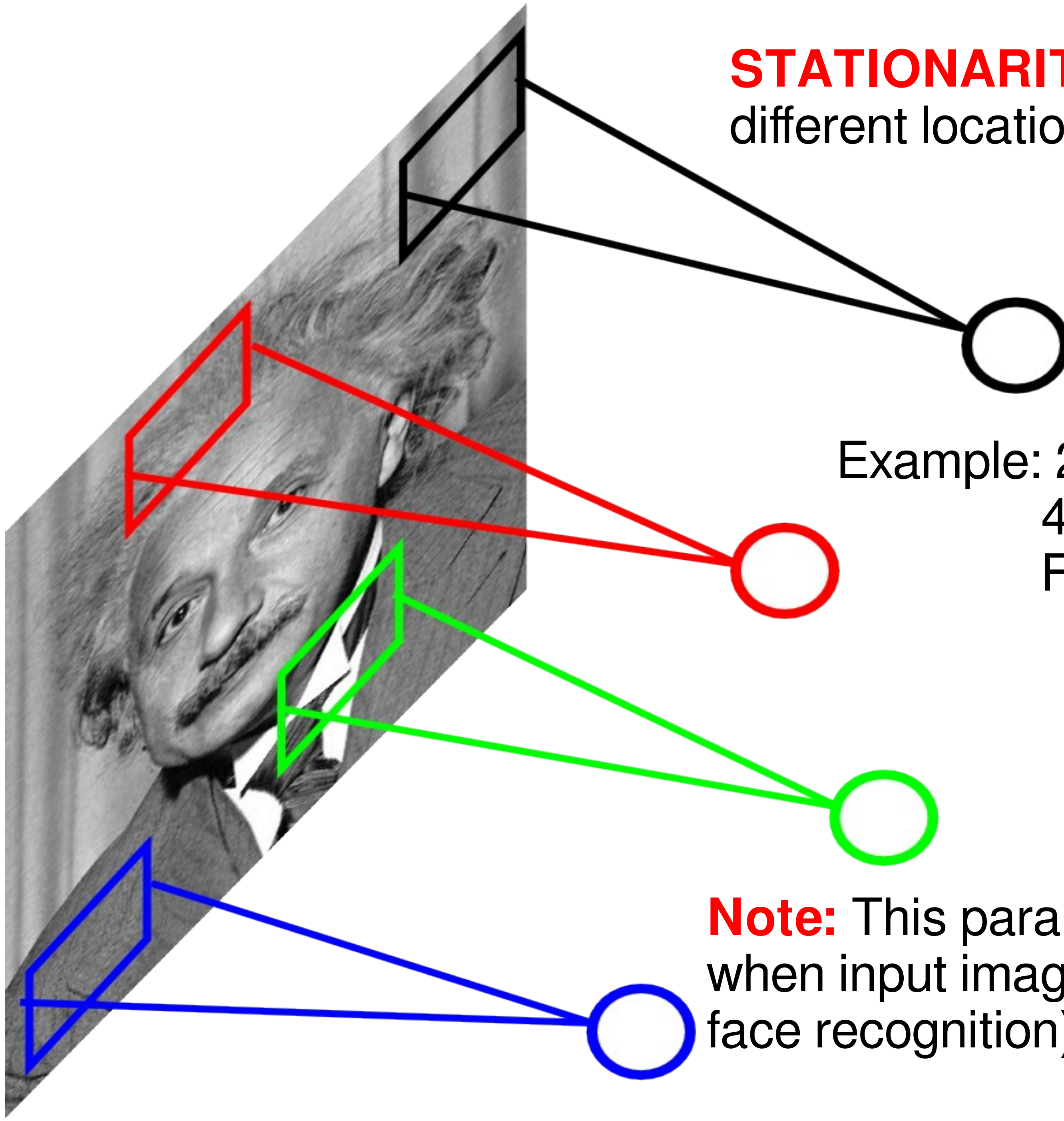
Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Locally Connected Layer

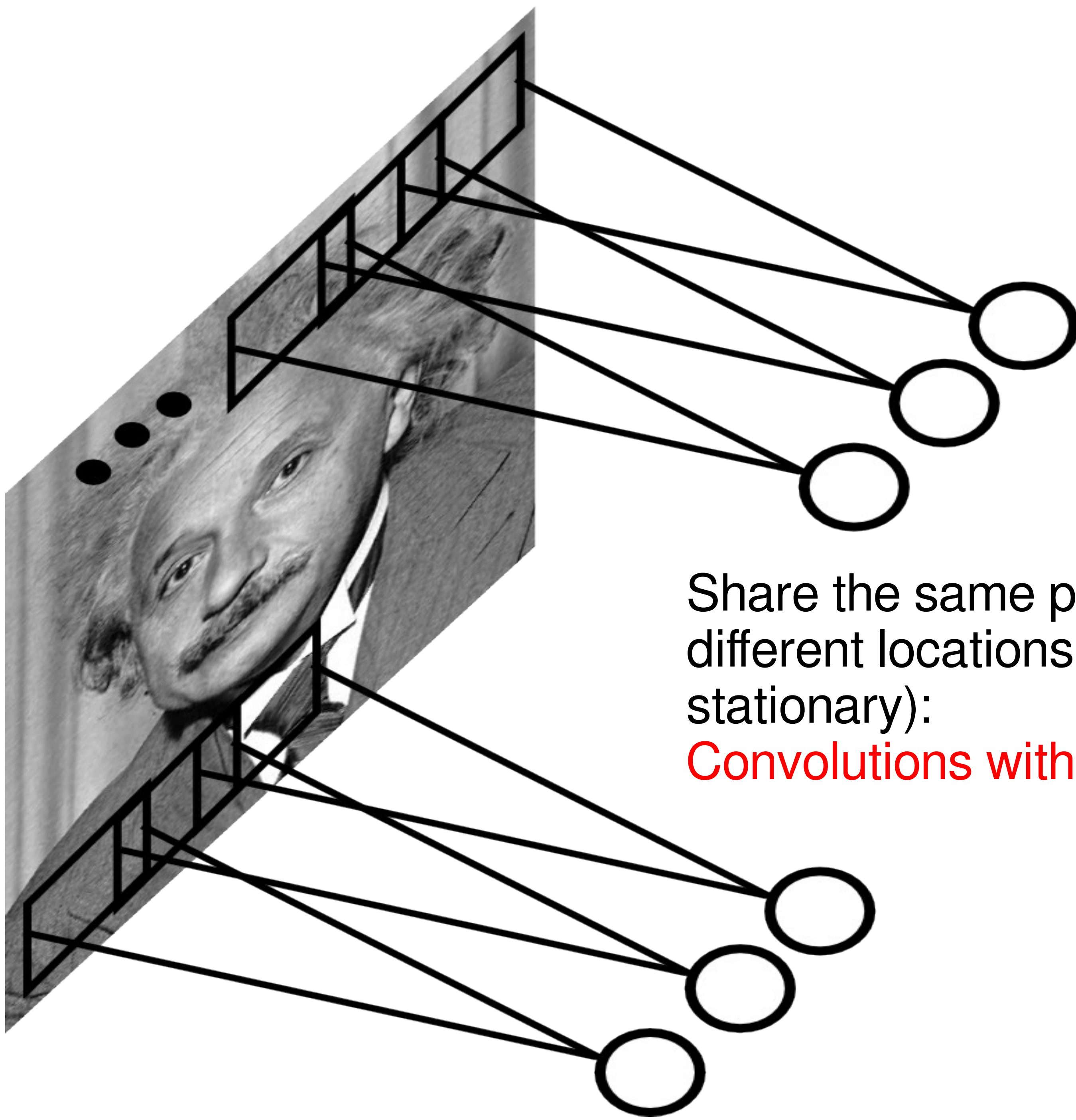


STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

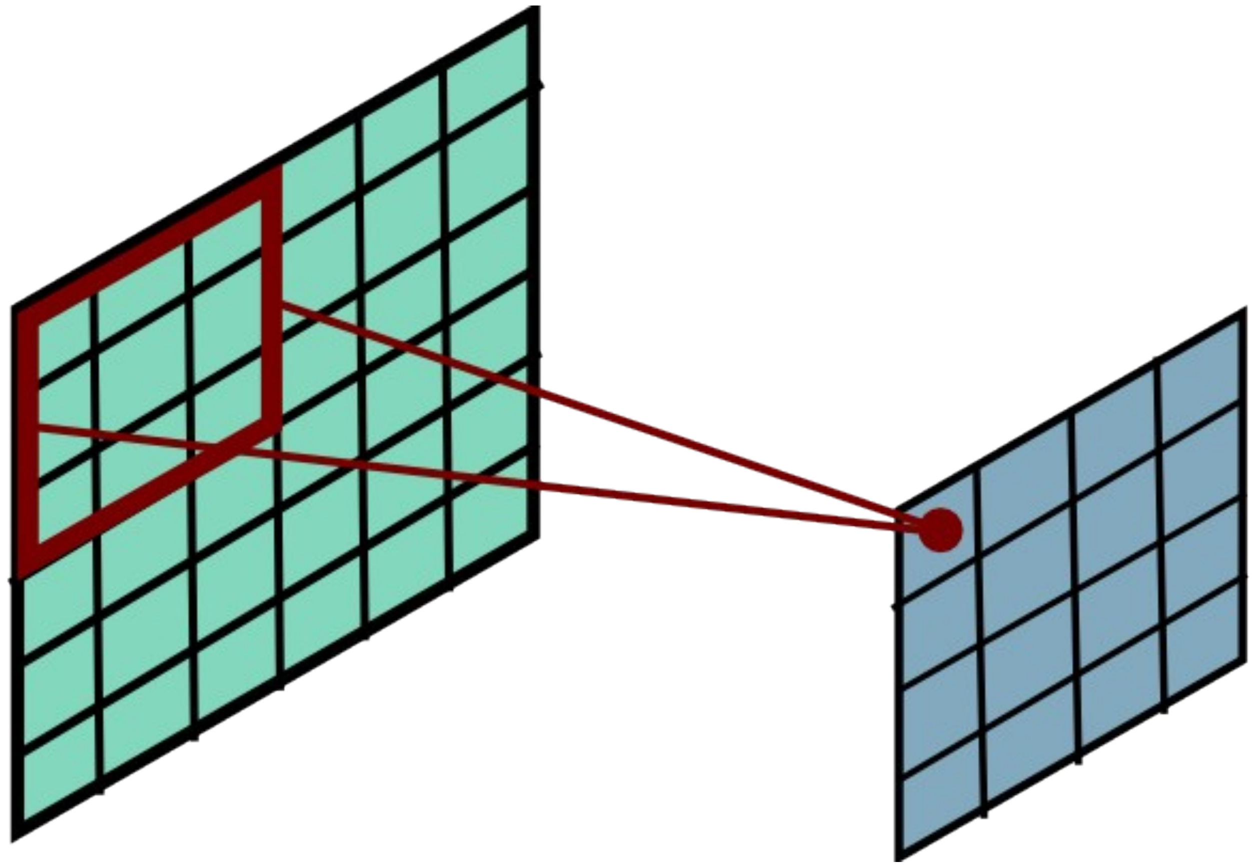
Note: This parameterization is good when input image is registered (e.g., face recognition).

Convolutional Layer

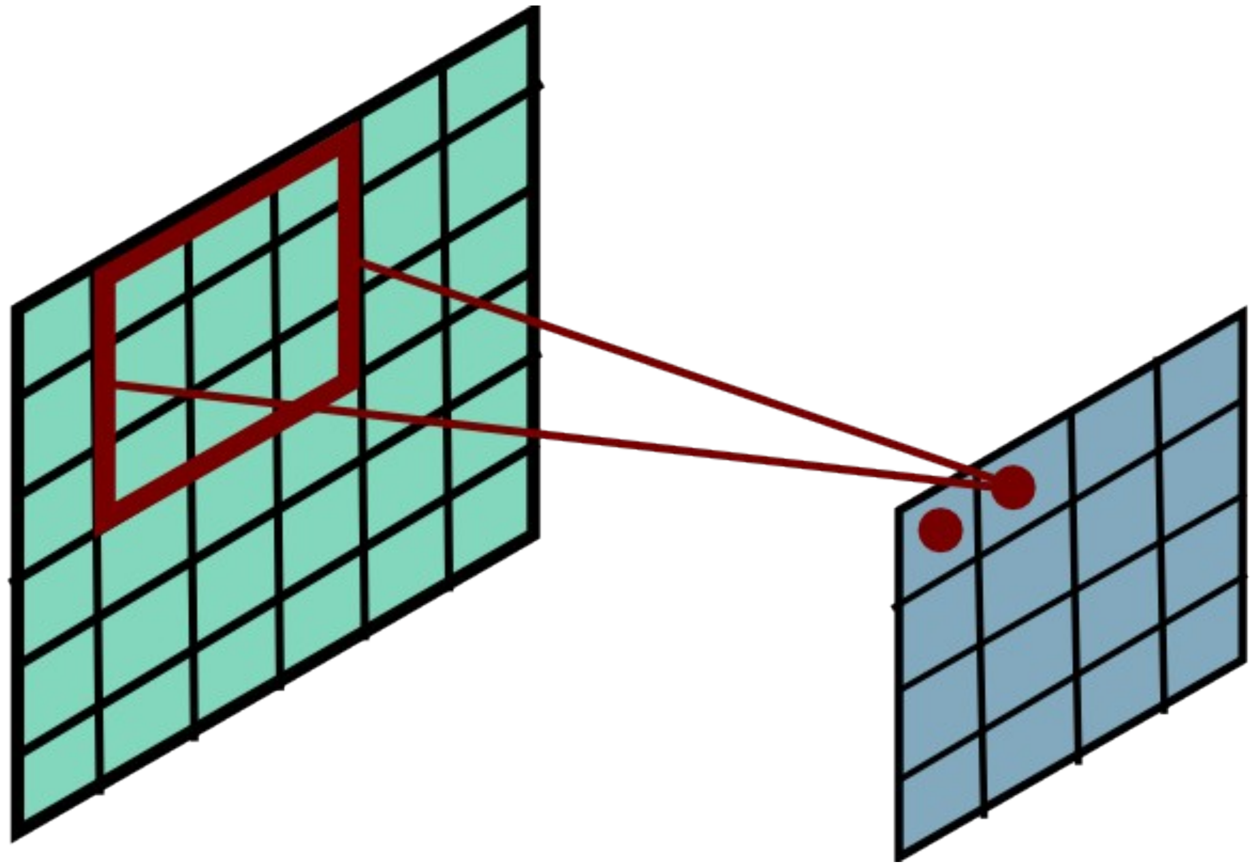


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

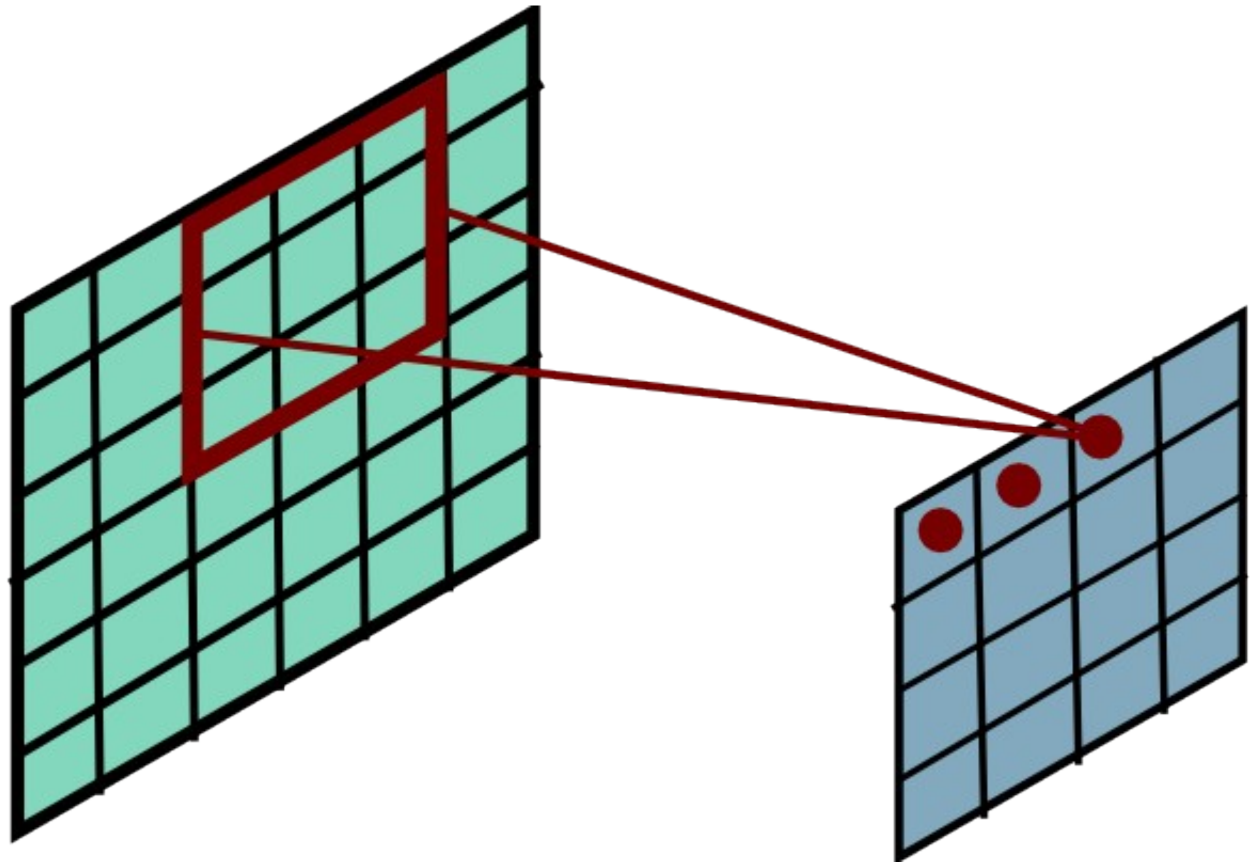
Convolutional Layer



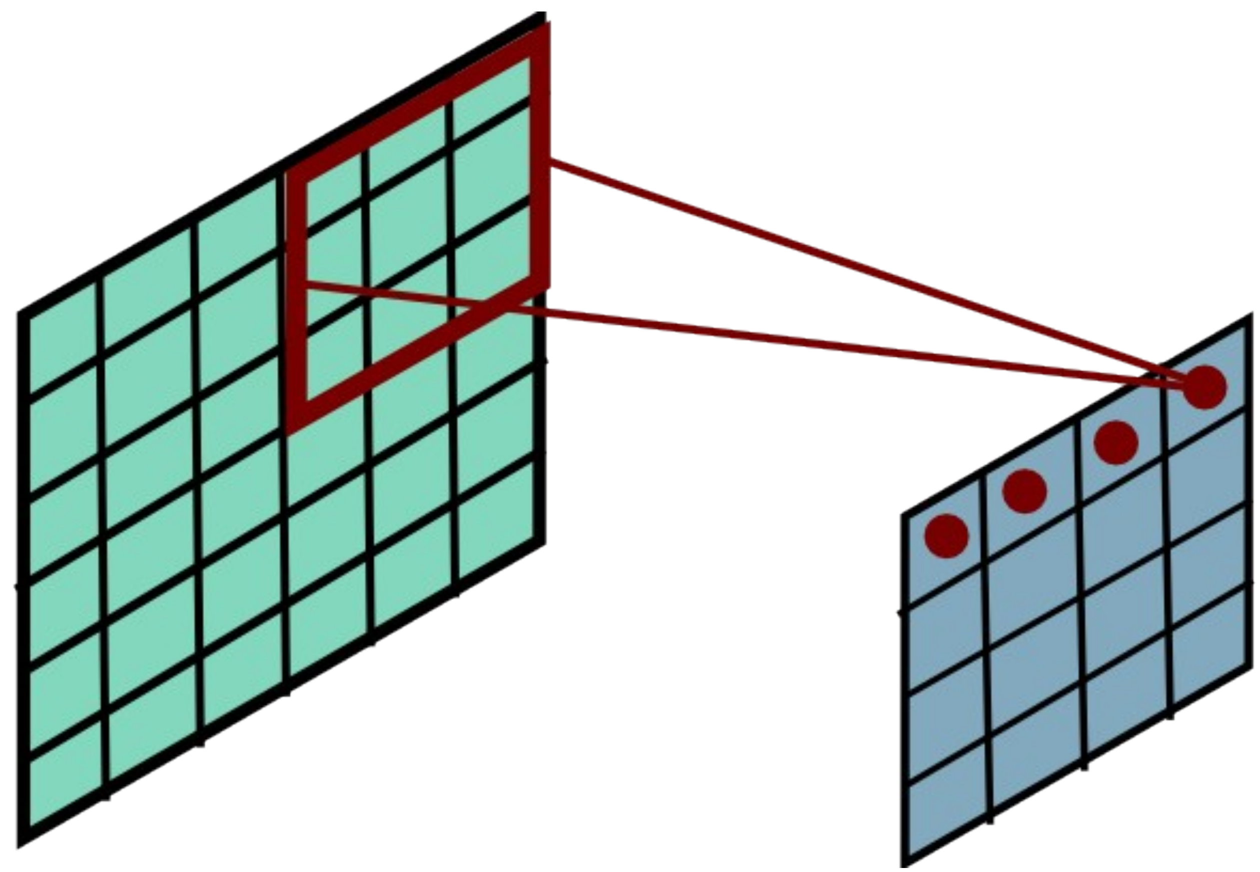
Convolutional Layer



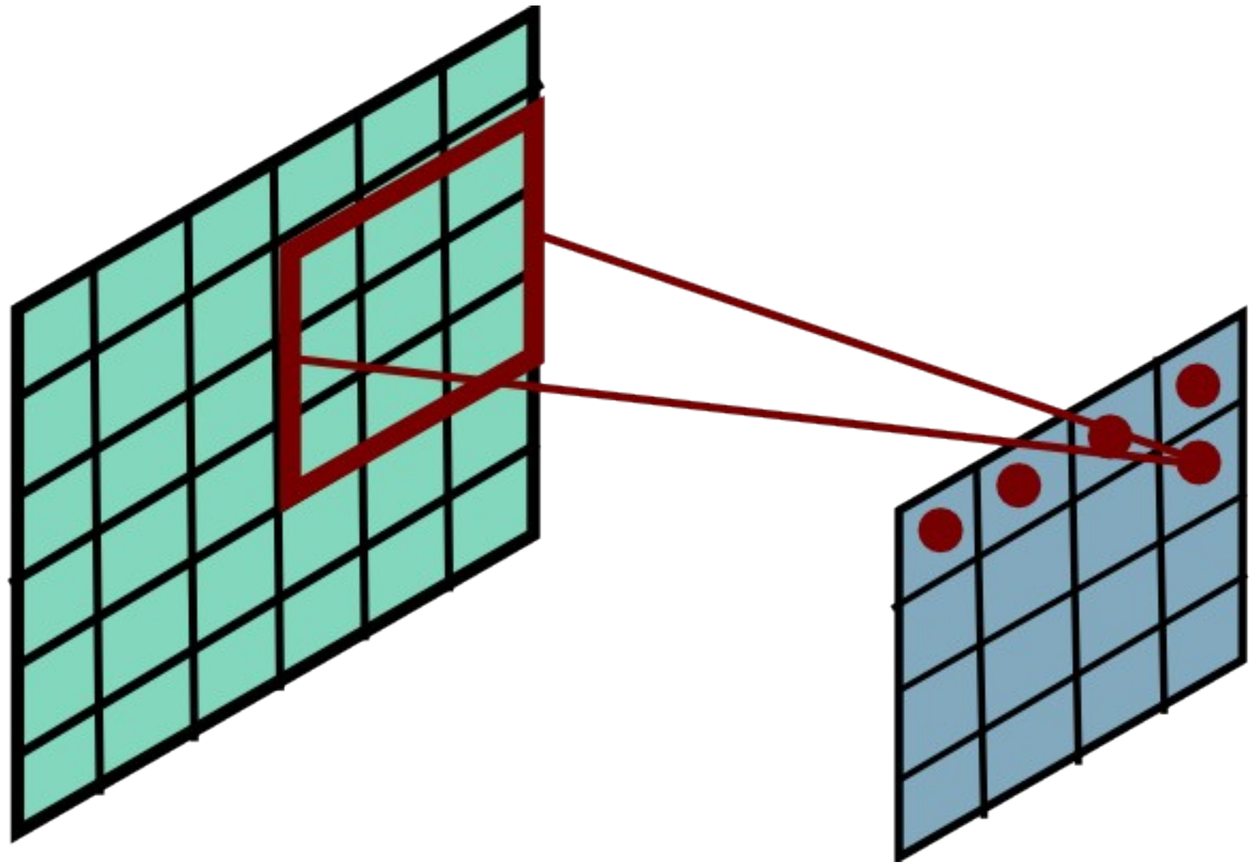
Convolutional Layer



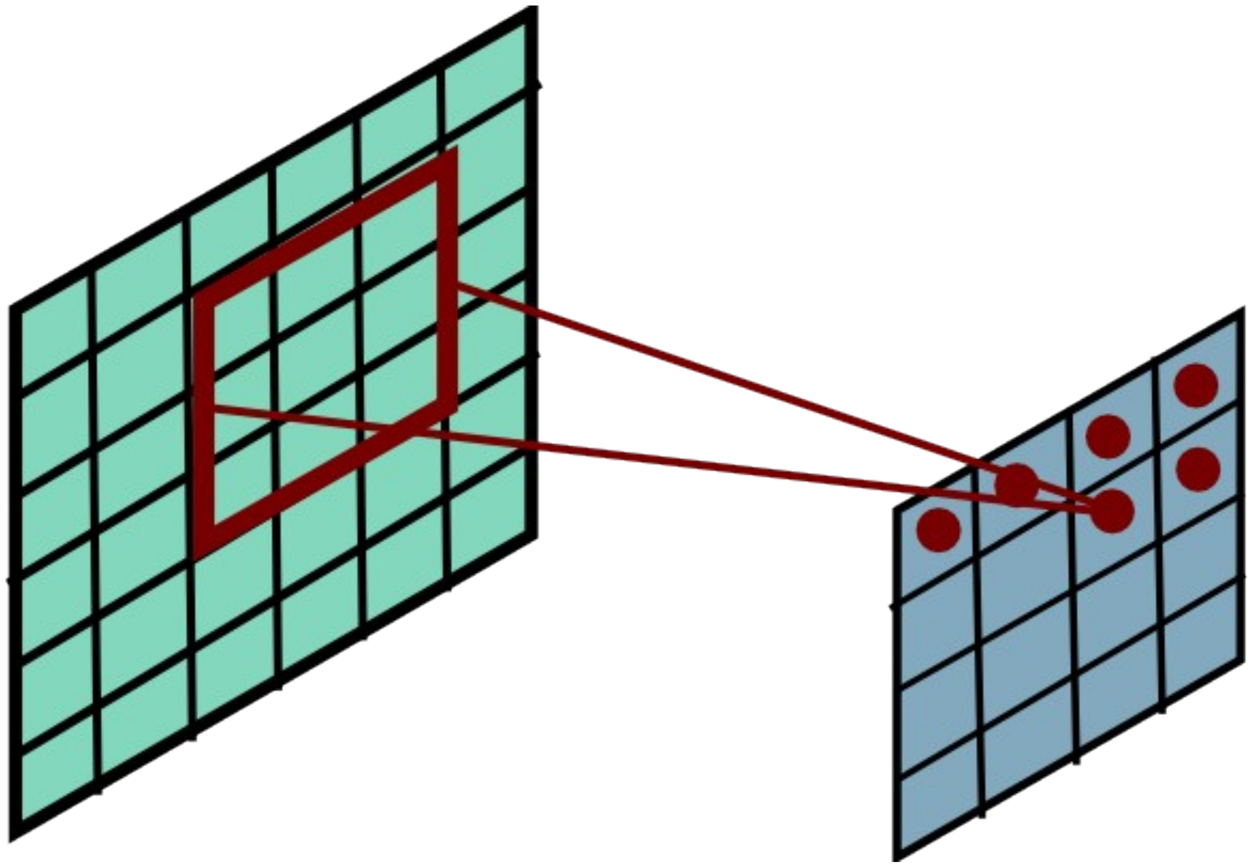
Convolutional Layer



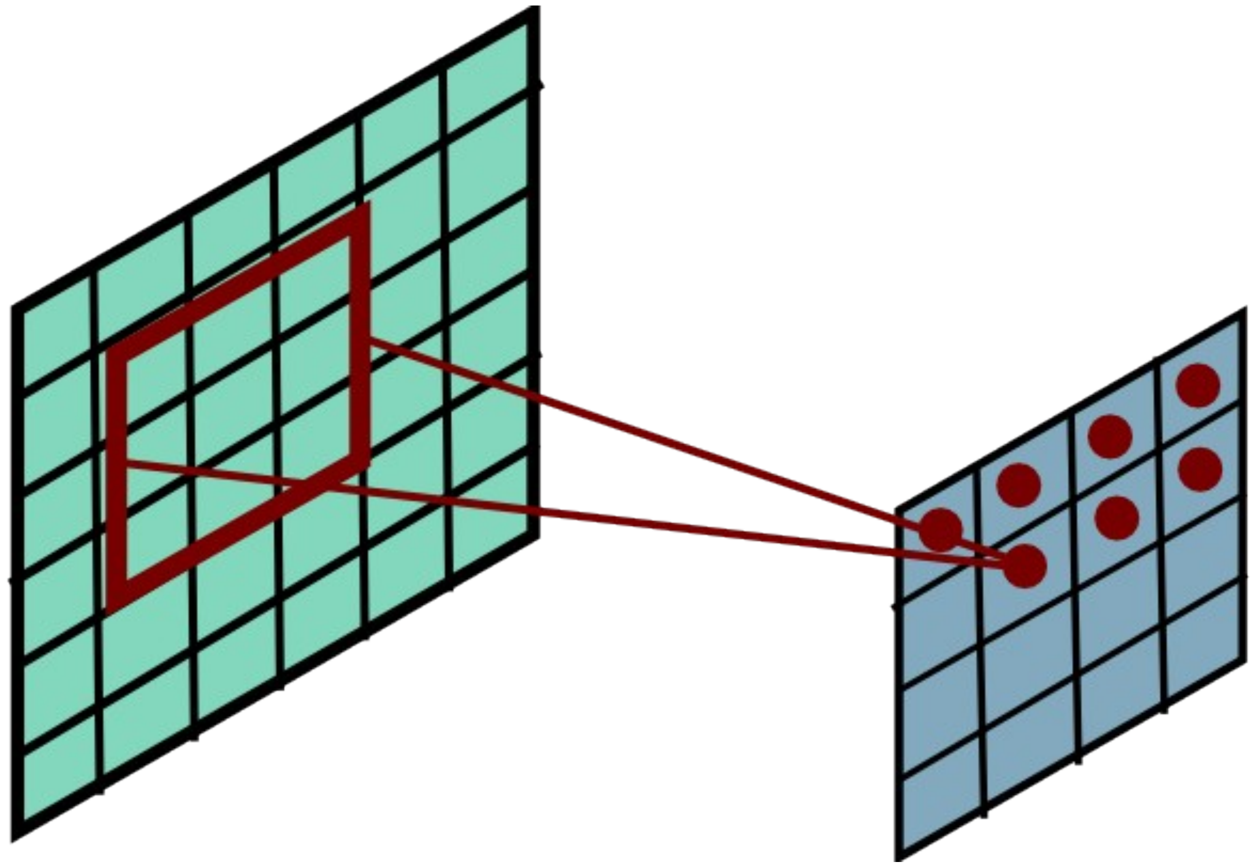
Convolutional Layer



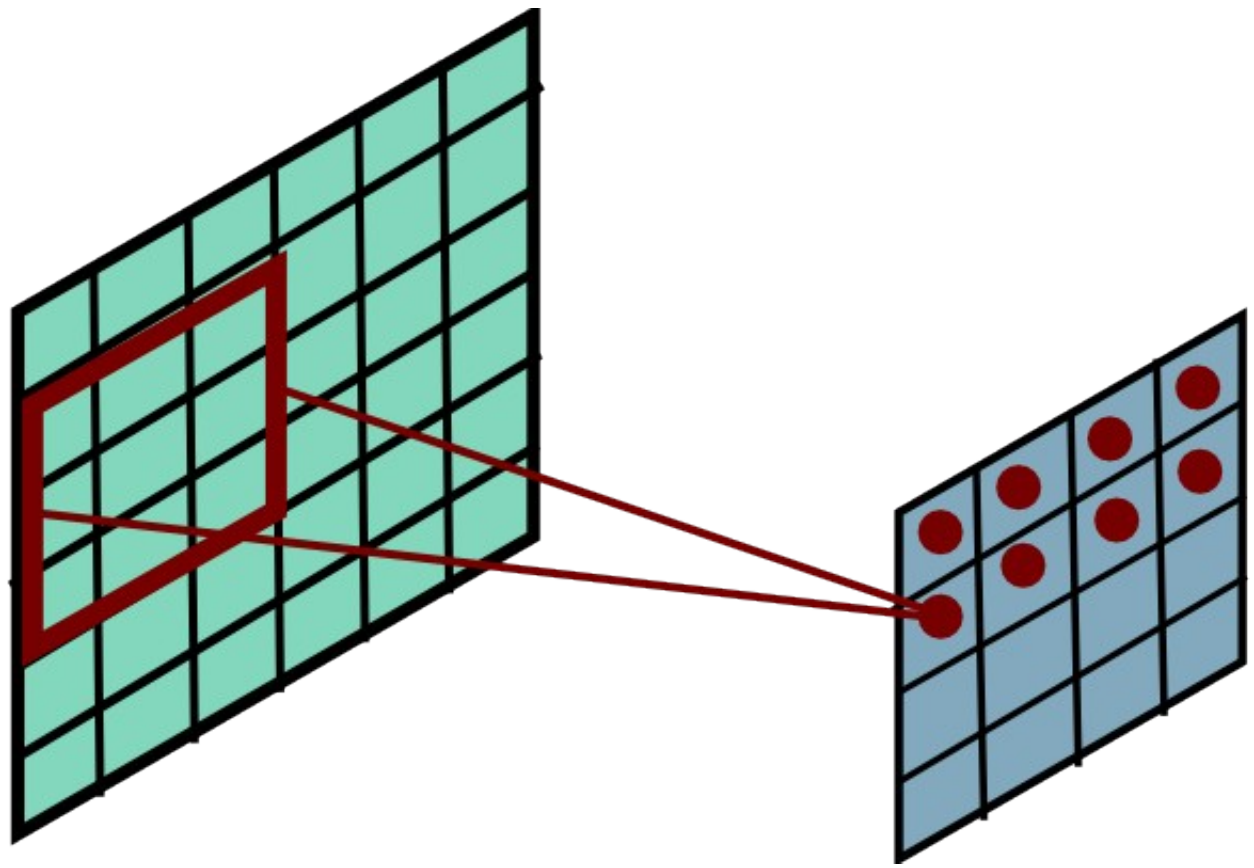
Convolutional Layer



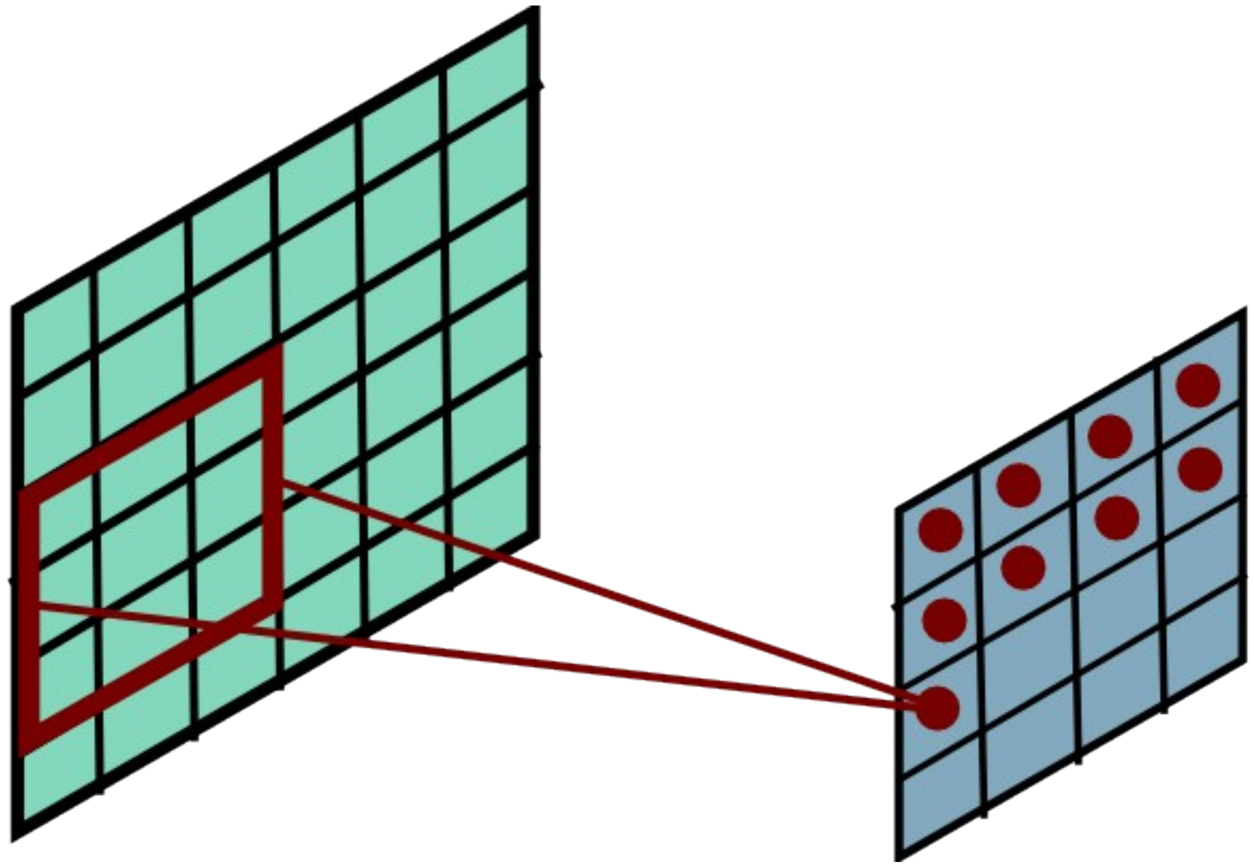
Convolutional Layer



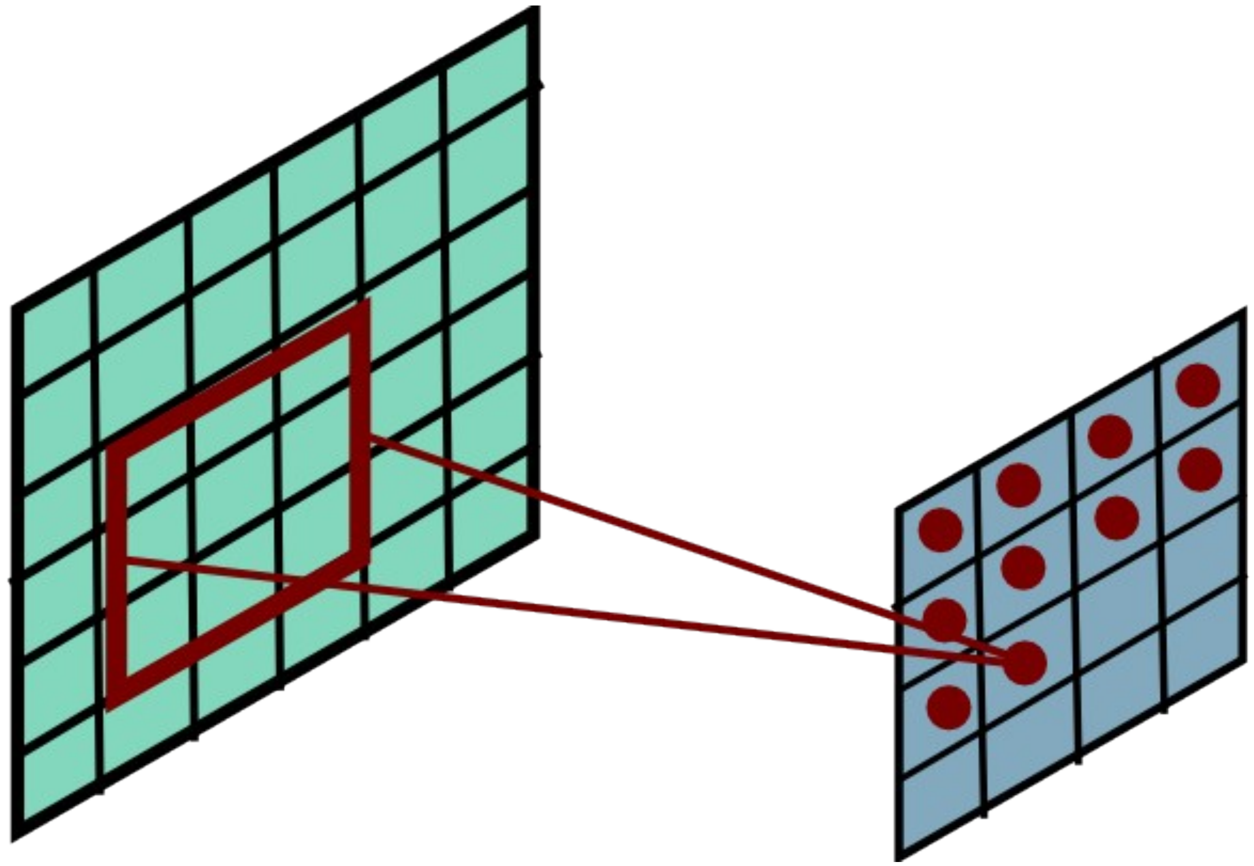
Convolutional Layer



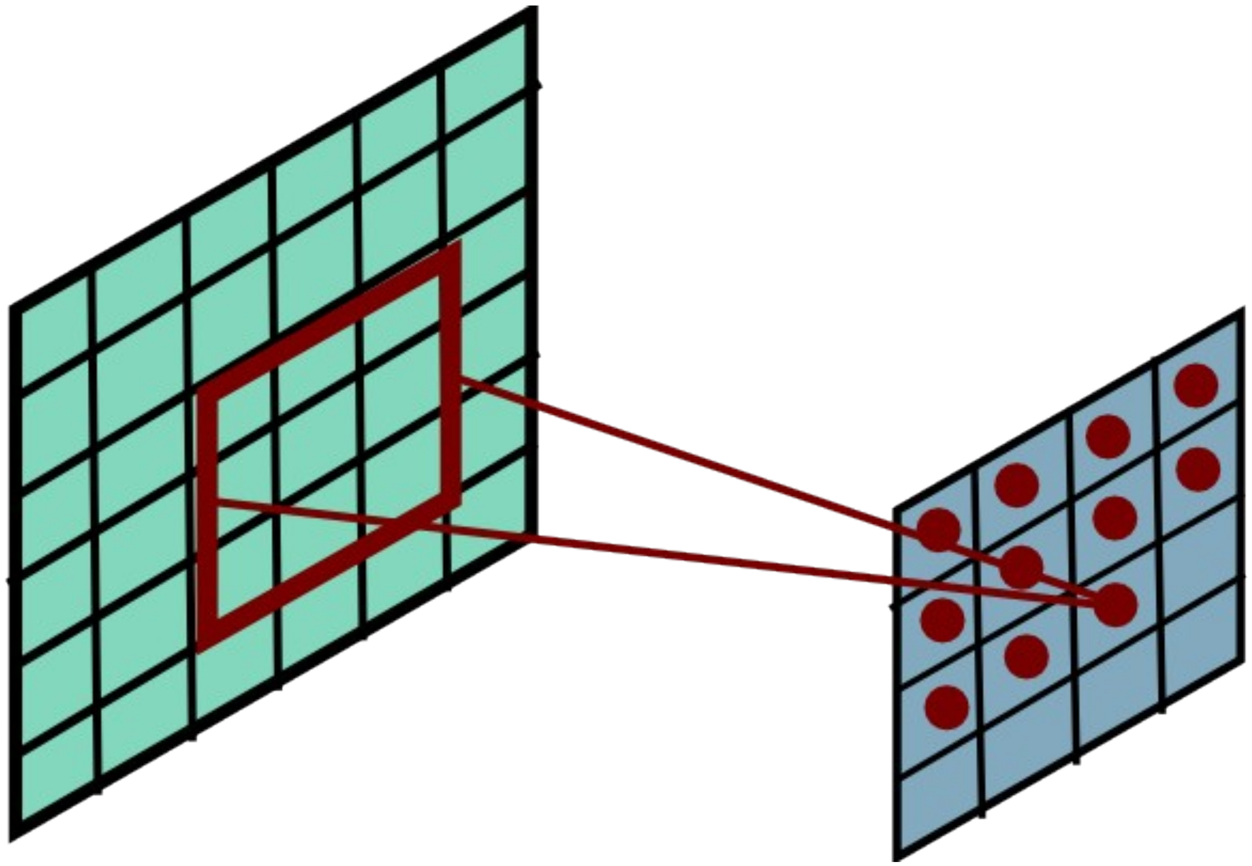
Convolutional Layer



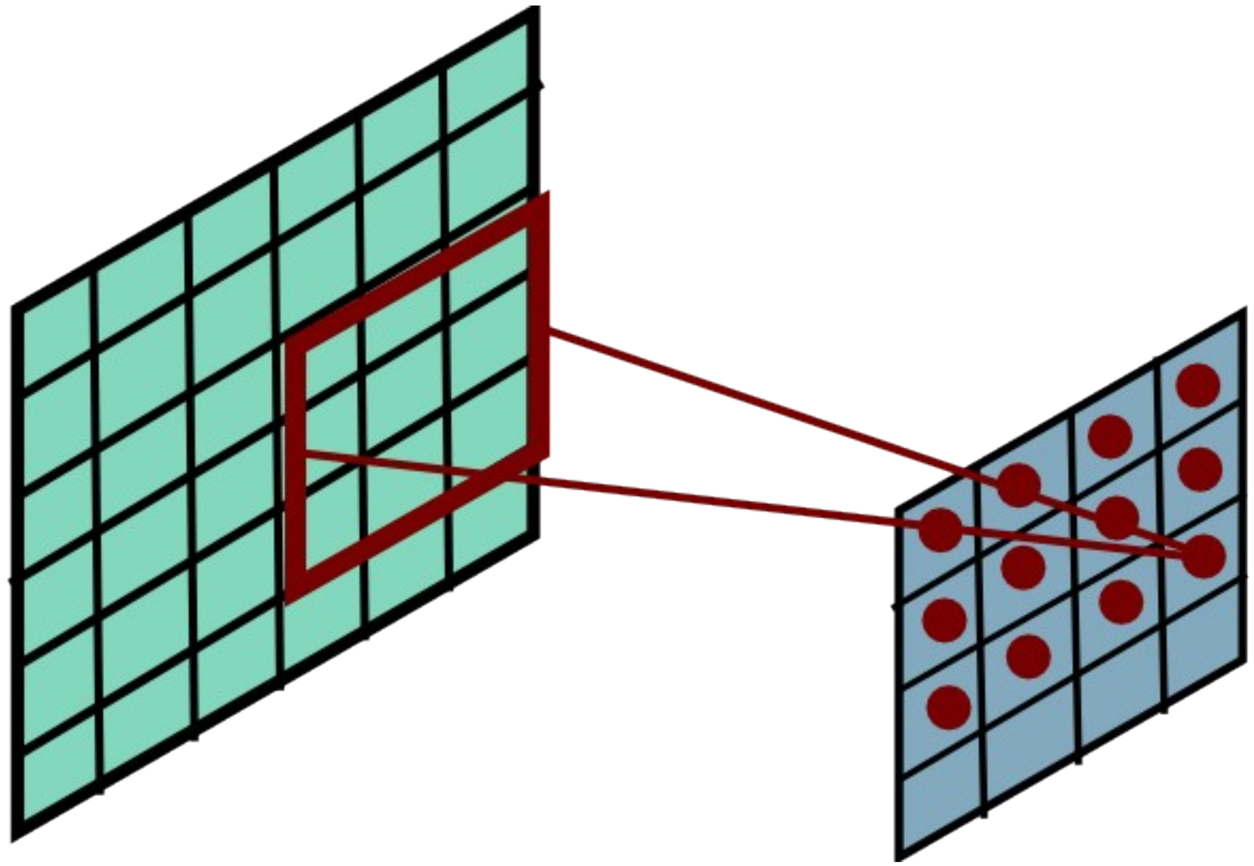
Convolutional Layer



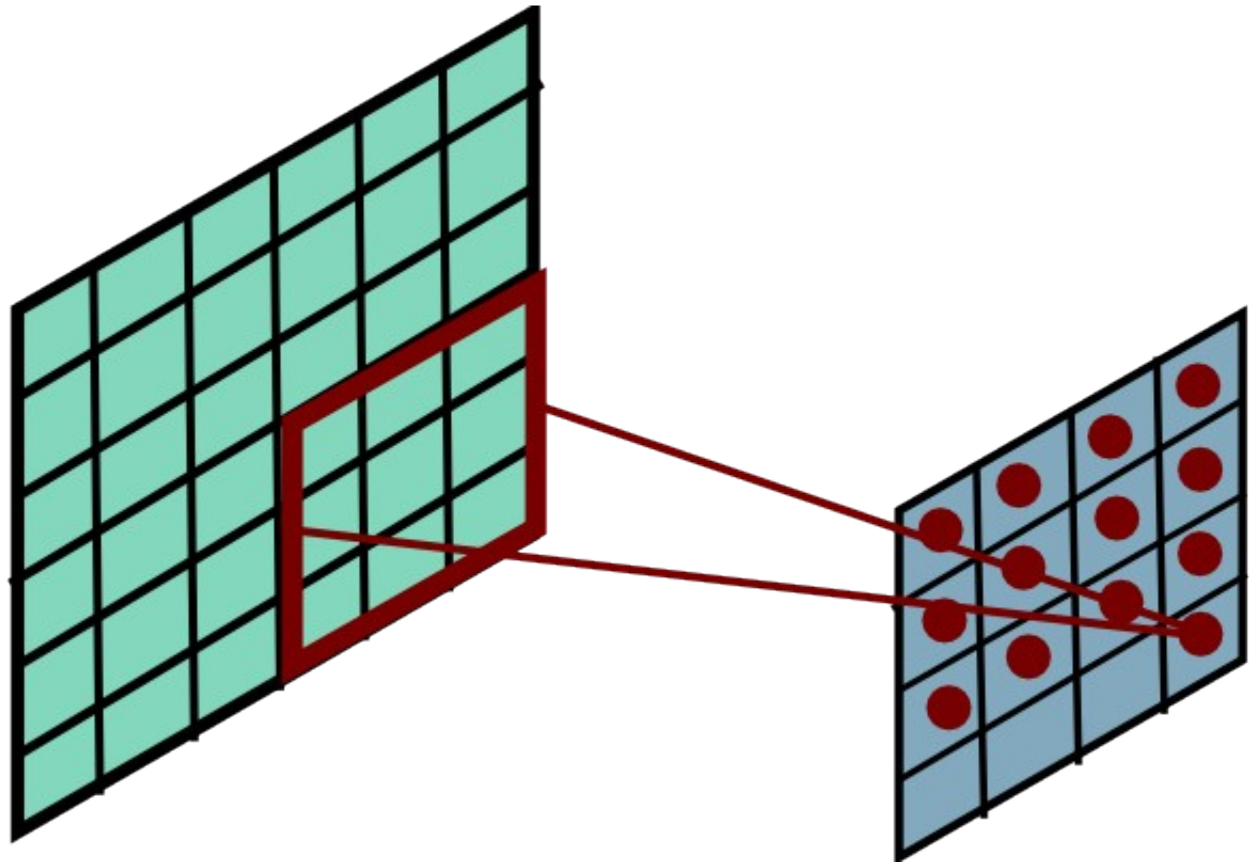
Convolutional Layer



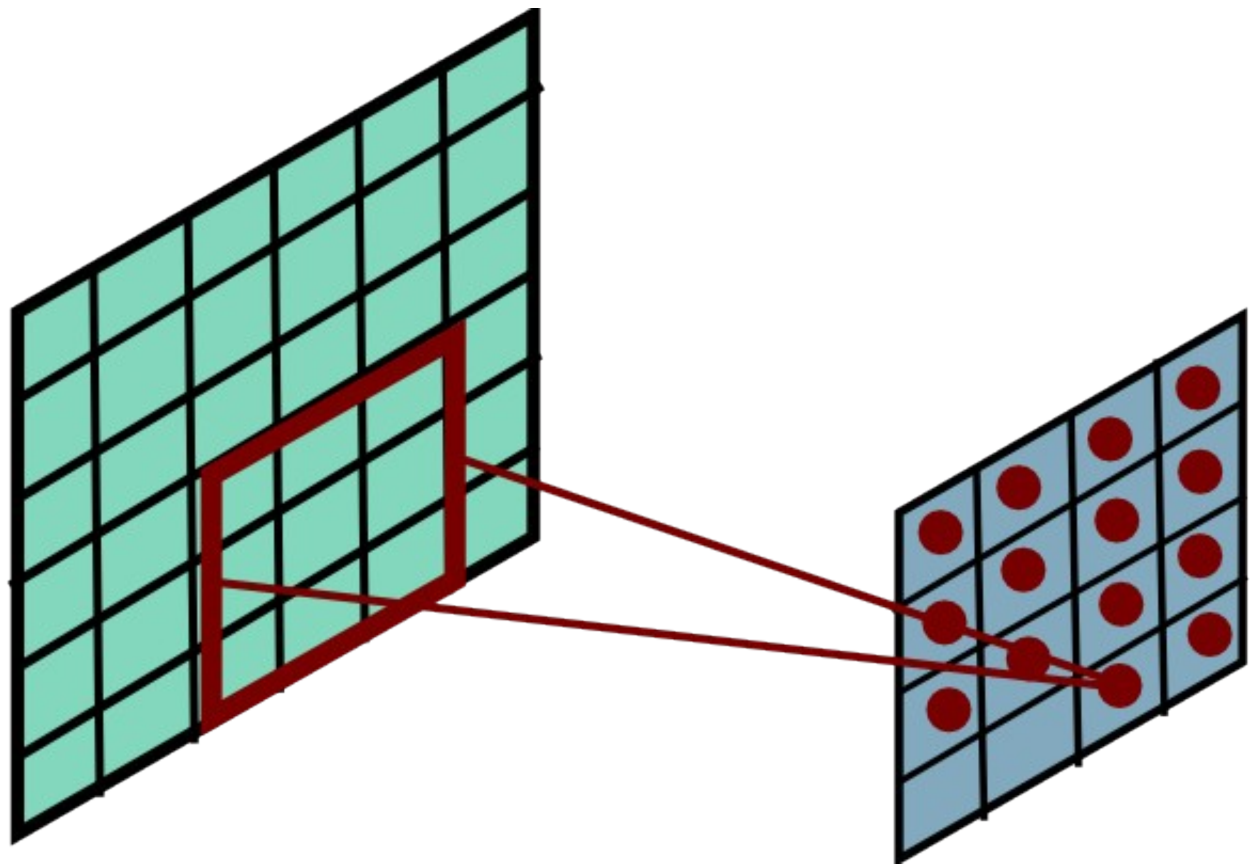
Convolutional Layer



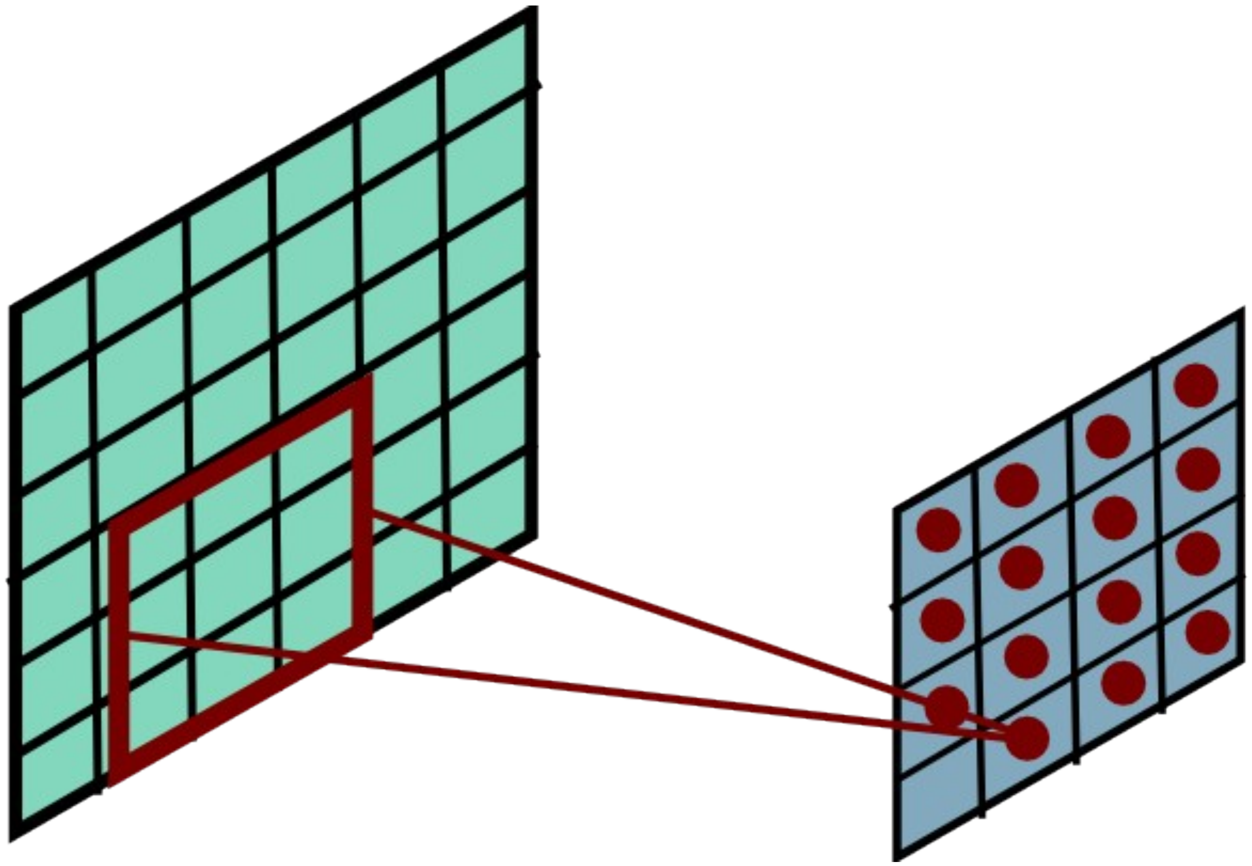
Convolutional Layer



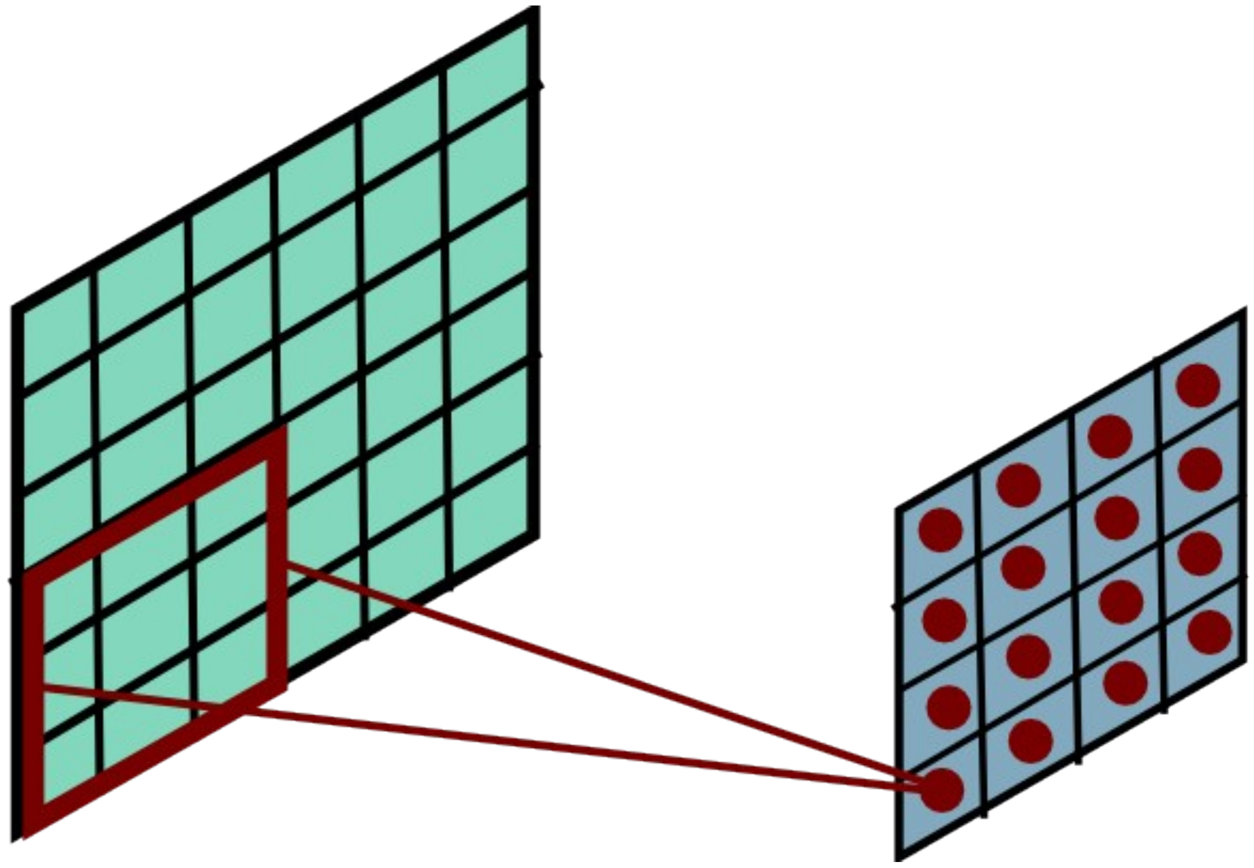
Convolutional Layer



Convolutional Layer

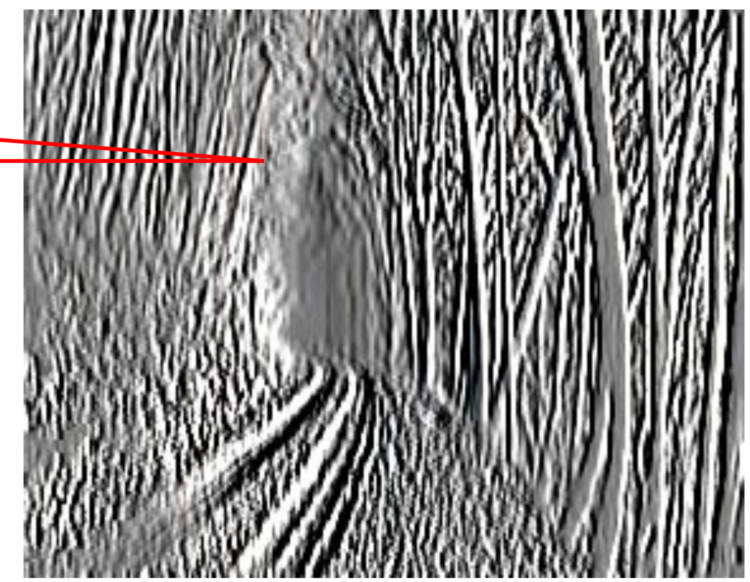


Convolutional Layer

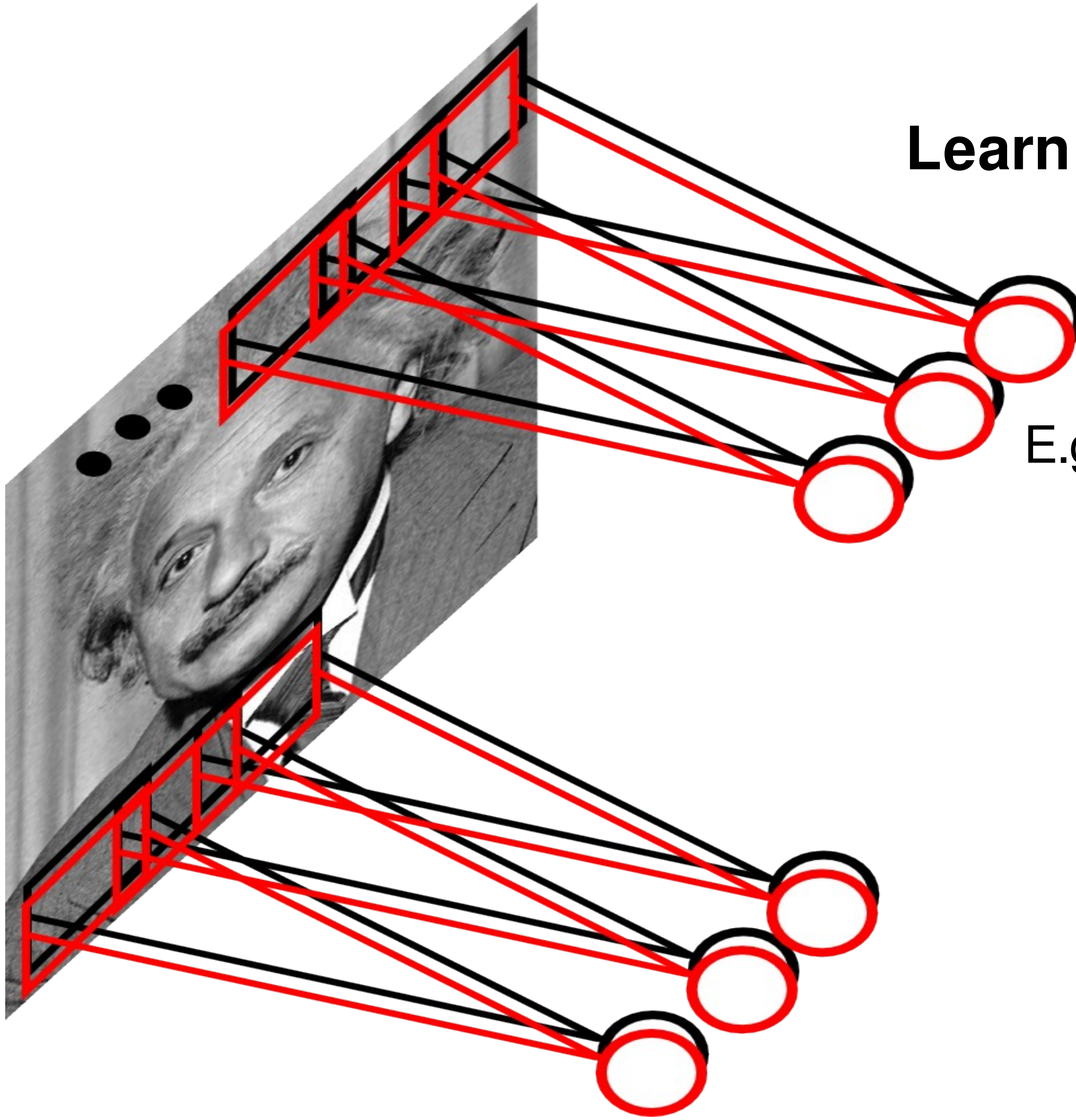


Convolutional Layer

$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

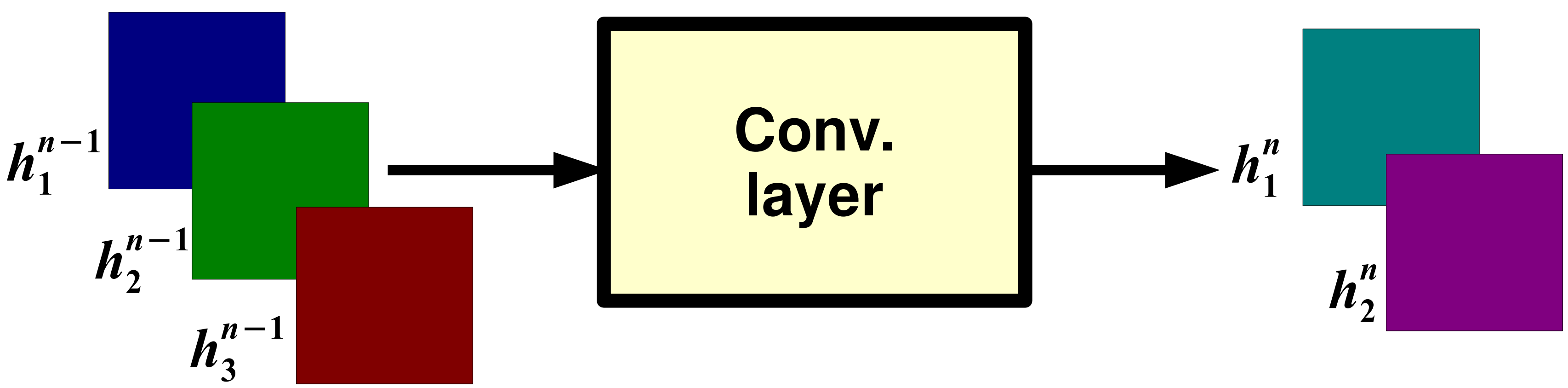
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



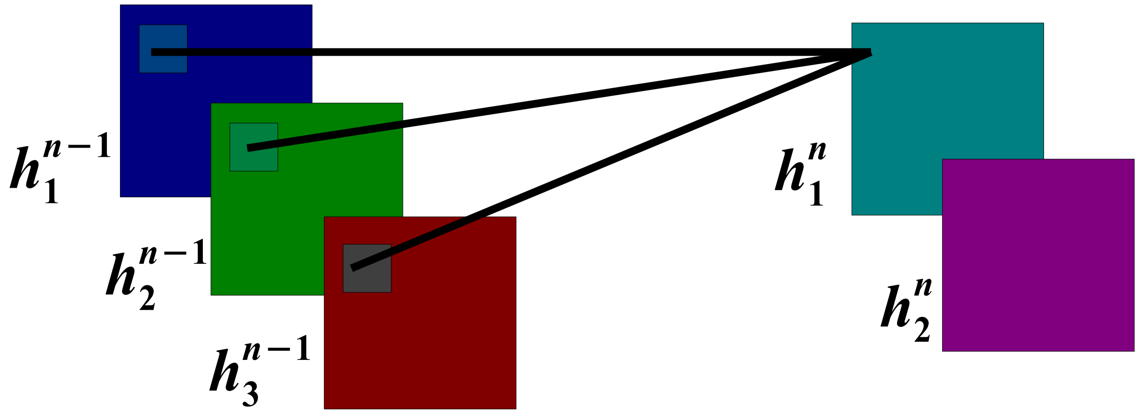
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output
feature map

input
feature map

kernel



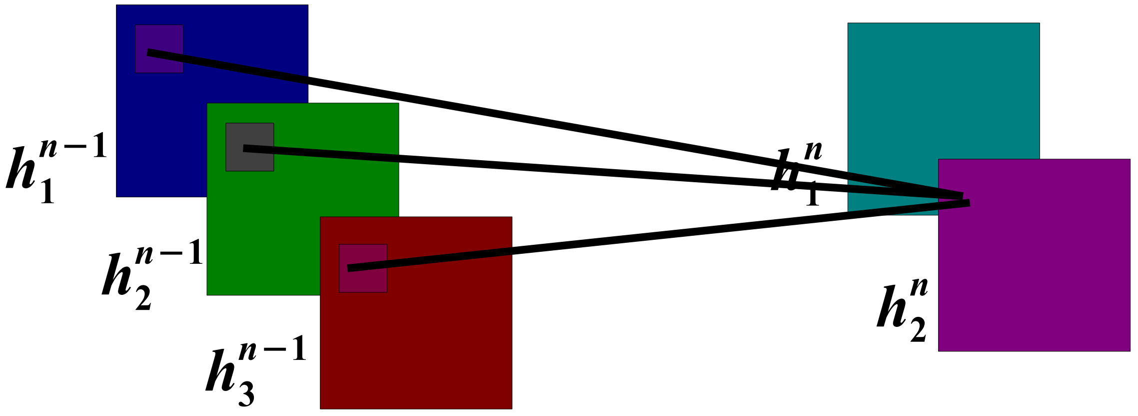
Convolutional Layer

$$h_j^n = max\,(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n)$$

output feature map

input feature map

kernel



Convolutional Layer

Question: What is the size of the output? What's the computational cost?

Answer: It is proportional to the number of filters and depends on the stride. If kernels have size KxK, input has size DxD, stride is 1, and there are M input feature maps and N output feature maps then:
- the input has size M@DxD
- the output has size N@(D-K+1)x(D-K+1)
- the kernels have MxNxKxK coefficients (which have to be learned)
- cost: M*K*K*N*(D-K+1)*(D-K+1)

Question: How many feature maps? What's the size of the filters?

Answer: Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors (and are expensive to compute).
The size of the filters has to match the size/scale of the patterns we want to detect (task dependent).

Key Ideas

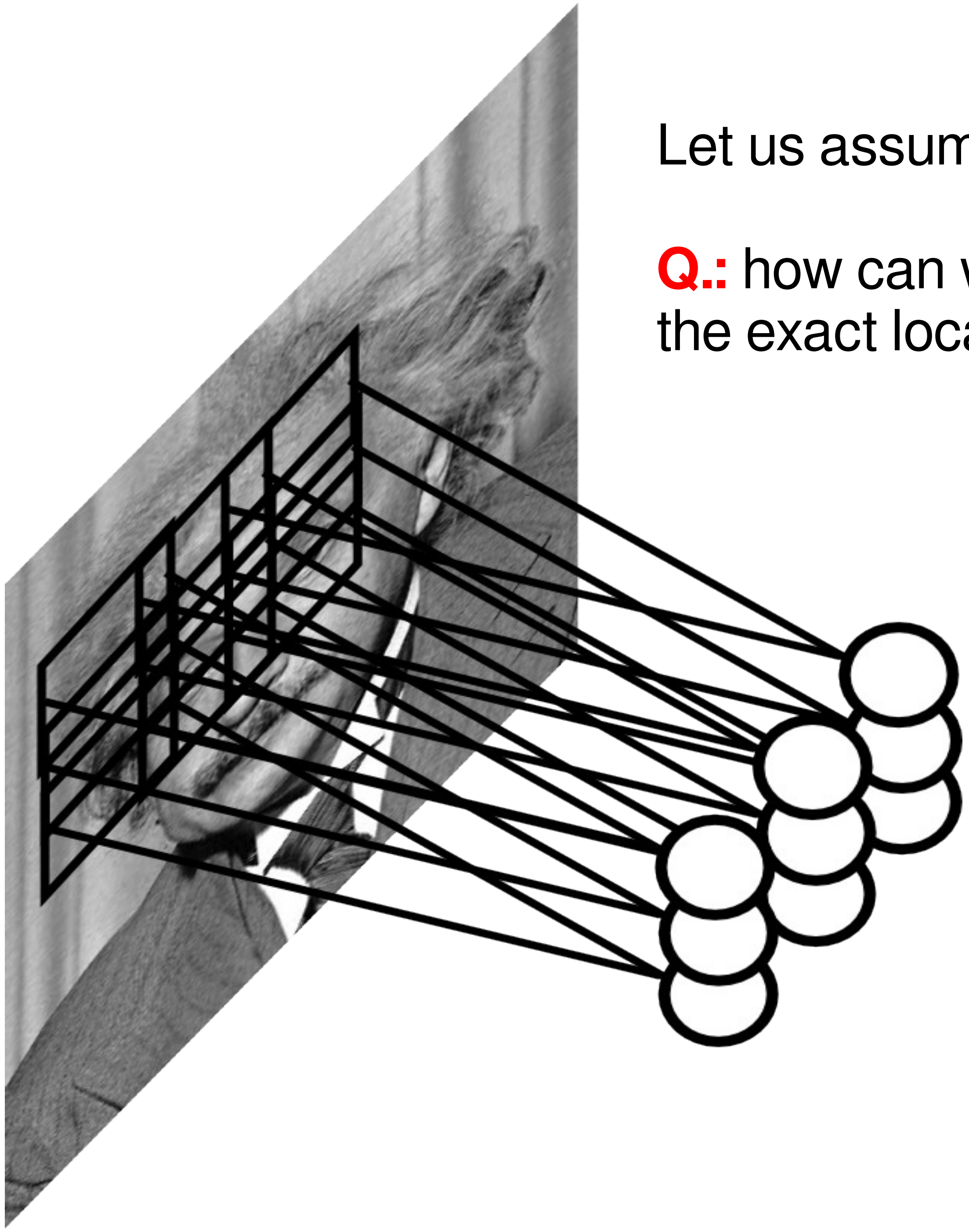
A standard neural net applied to images:
- scales quadratically with the size of the input
- does not leverage stationarity

Solution:
- connect each hidden unit to a small patch of the input
- share the weight across space

This is called: **convolutional layer.**
A network with convolutional layers is called **convolutional network.**

LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

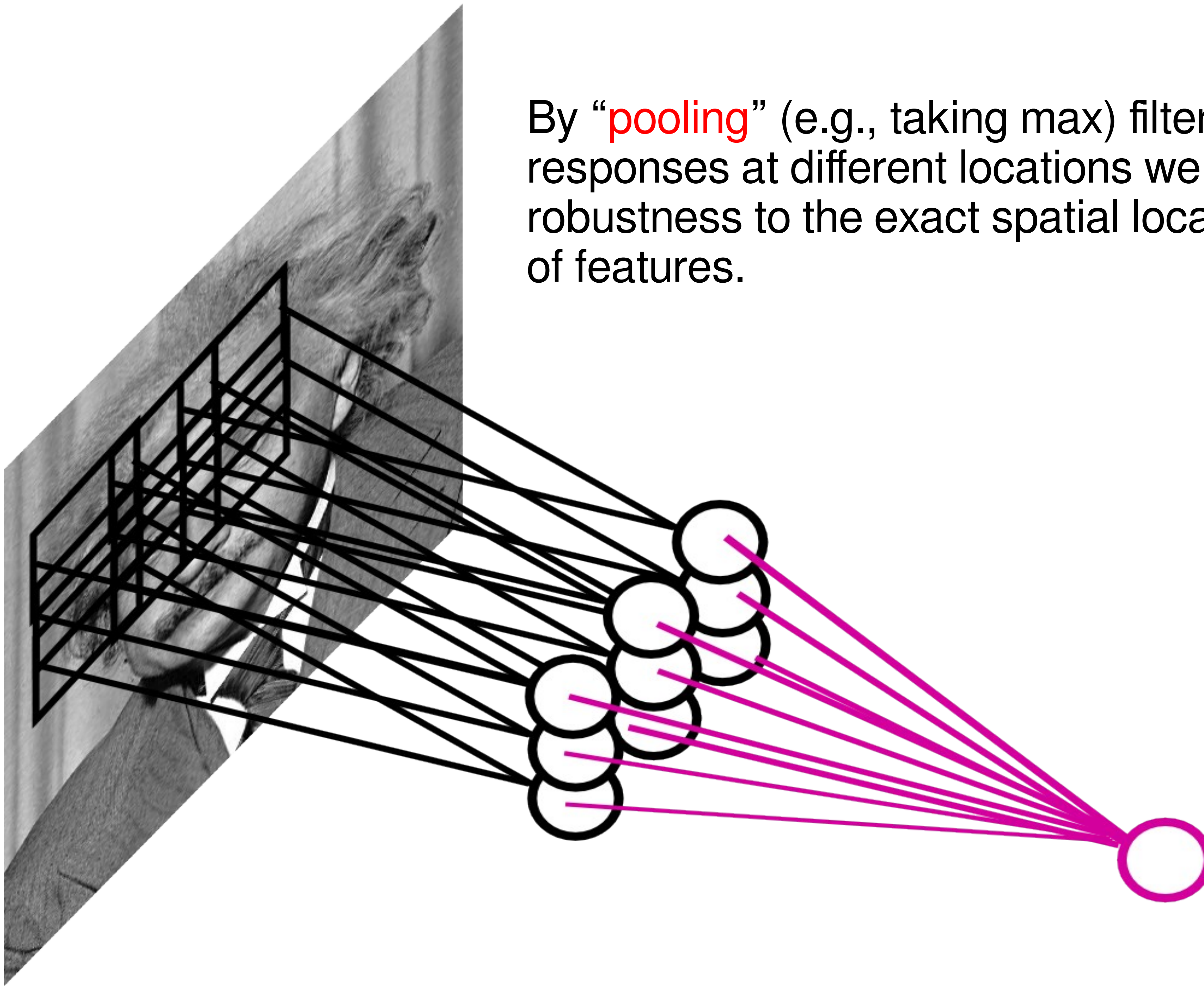
Pooling Layer



Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Pooling Layer: Examples

Max-pooling: **most popular version**

$$h_j^n(x,y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

L2-pooling:

$$h_j^n(x,y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x},\bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x,y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x,y)^2}$$

Pooling Layer

Question: What is the size of the output? What's the computational cost?

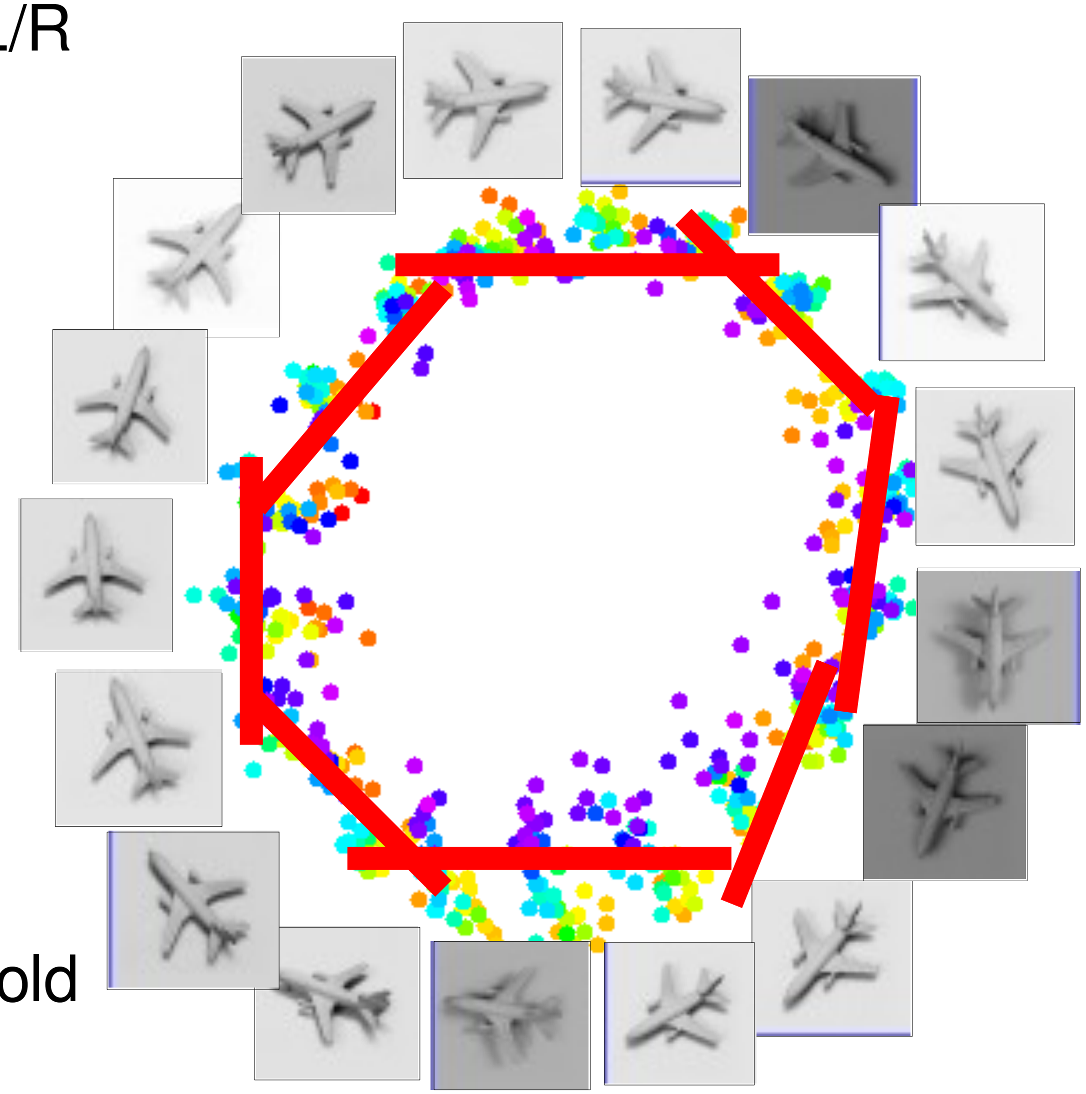
Answer: The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size KxK, and the input has size DxD with M input feature maps, then:
- output is M@(D/K)x(D/K)
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

Question: How should I set the size of the pools?

Answer: It depends on how much “invariant” or robust to distortions we want the representation to be. It is best to pool slowly (via a few stacks of conv-pooling layers).

Pooling Layer: Interpretation

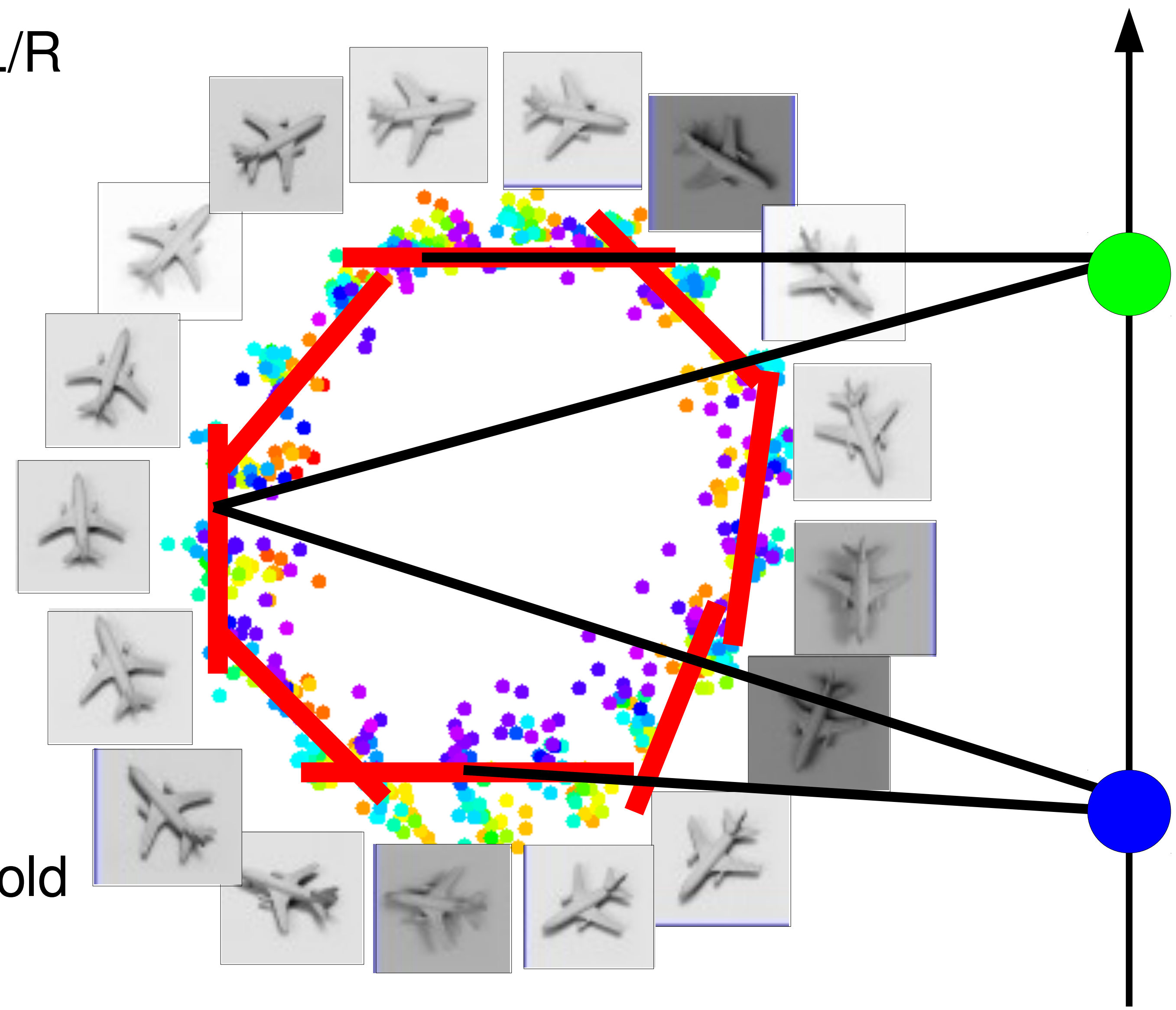
Task: detect orientation L/R



Conv layer:
linearizes manifold

Pooling Layer: Interpretation

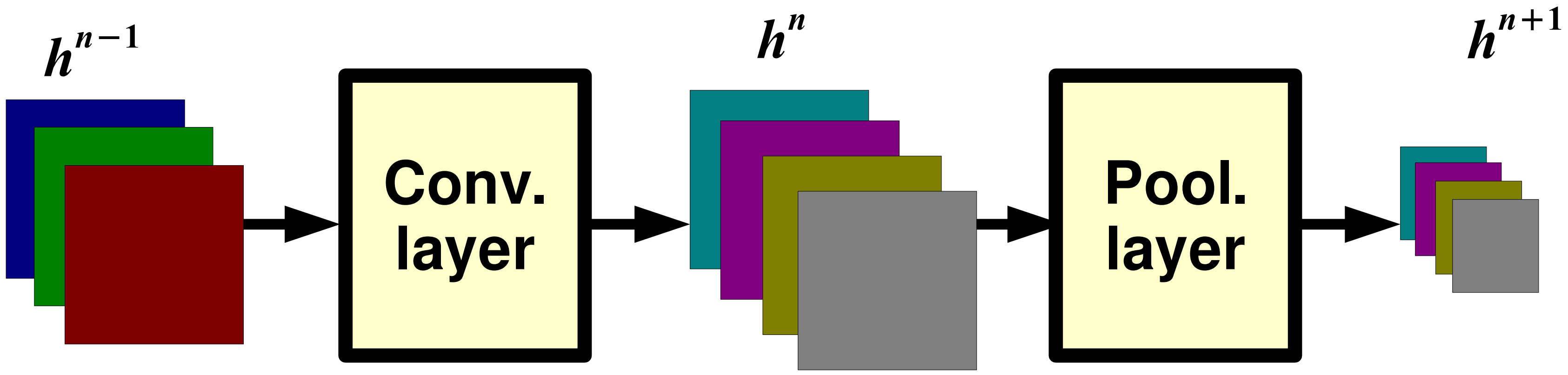
Task: detect orientation L/R



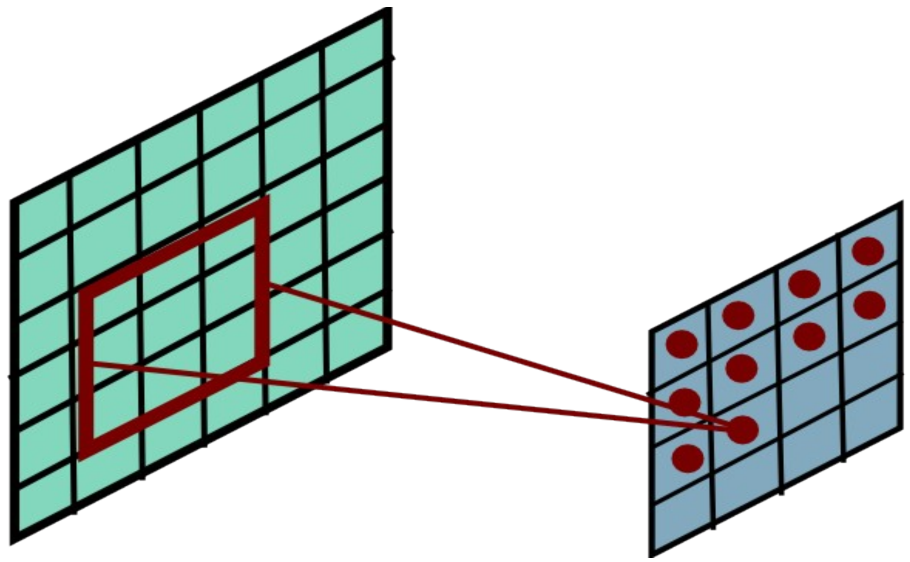
Conv layer:
linearizes manifold

Pooling layer:
collapses manifold

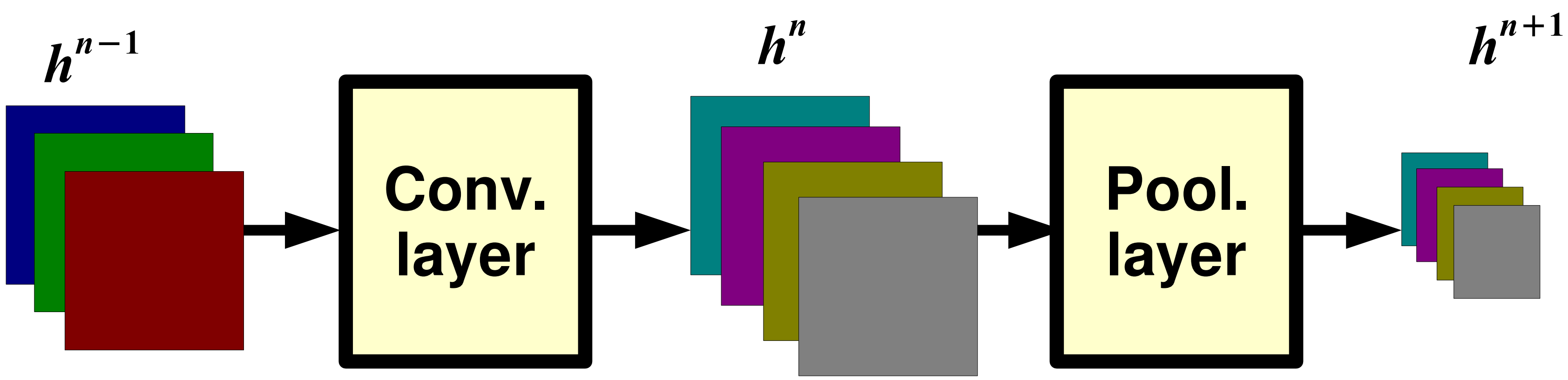
Pooling Layer: Receptive Field Size



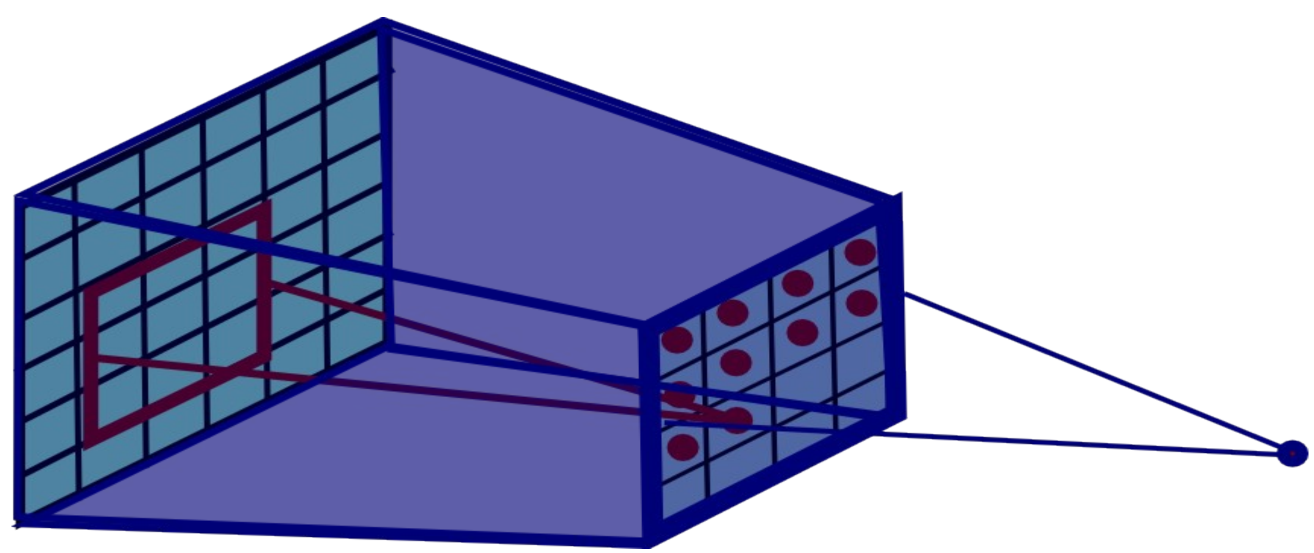
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Pooling Layer: Receptive Field Size

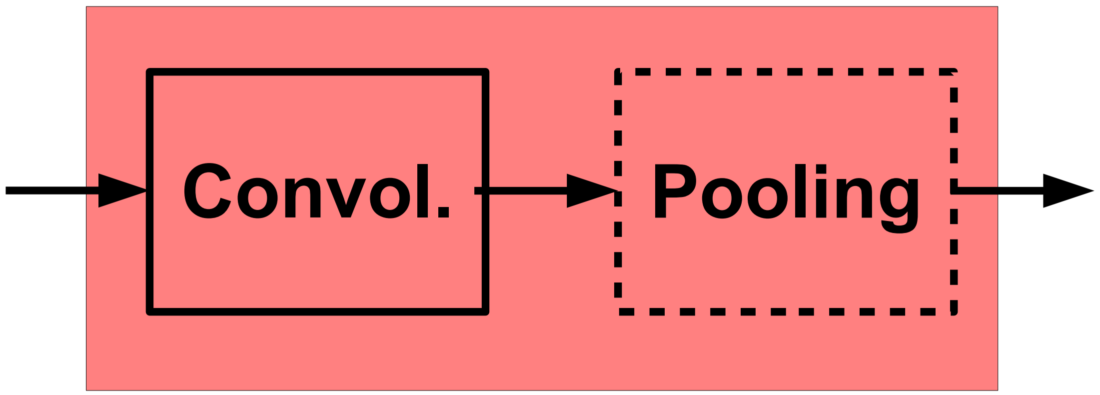


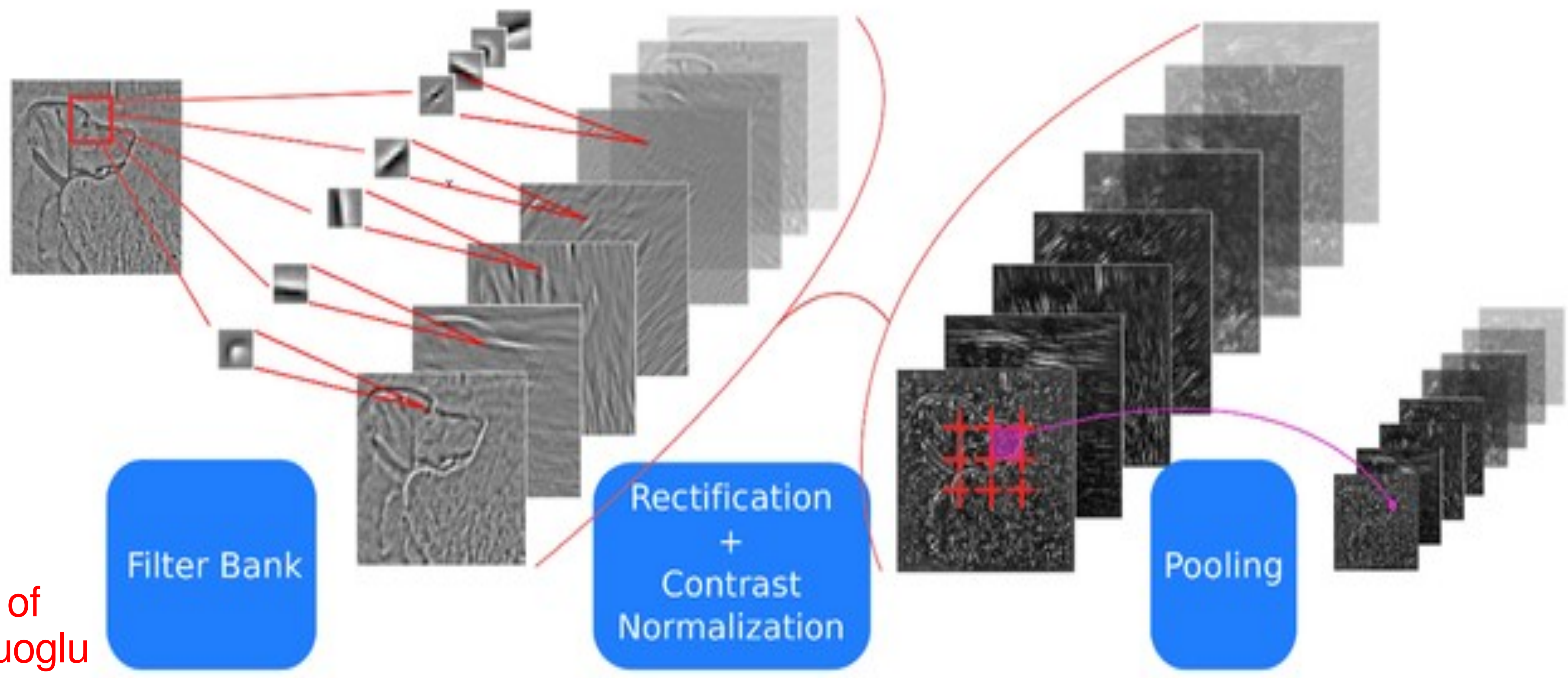
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



ConvNets: Typical Stage

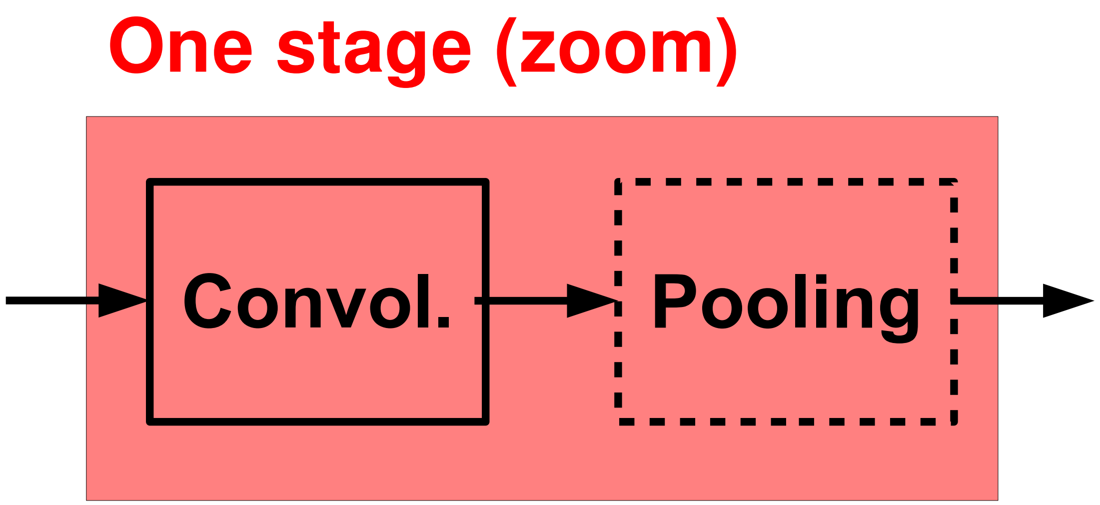
One stage (zoom)





courtesy of
K. Kavukcuoglu

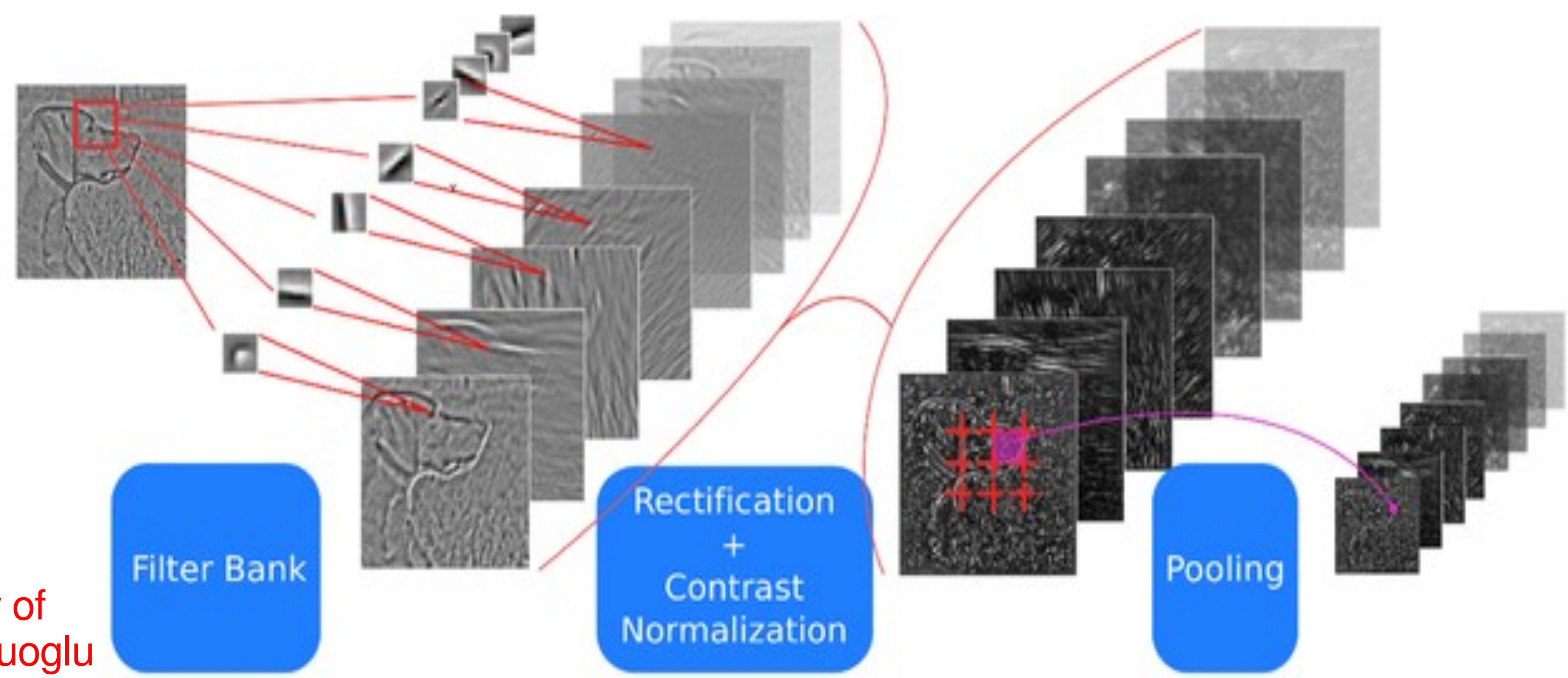
ConvNets: Typical Stage



Conceptually similar to: SIFT, HoG, etc.

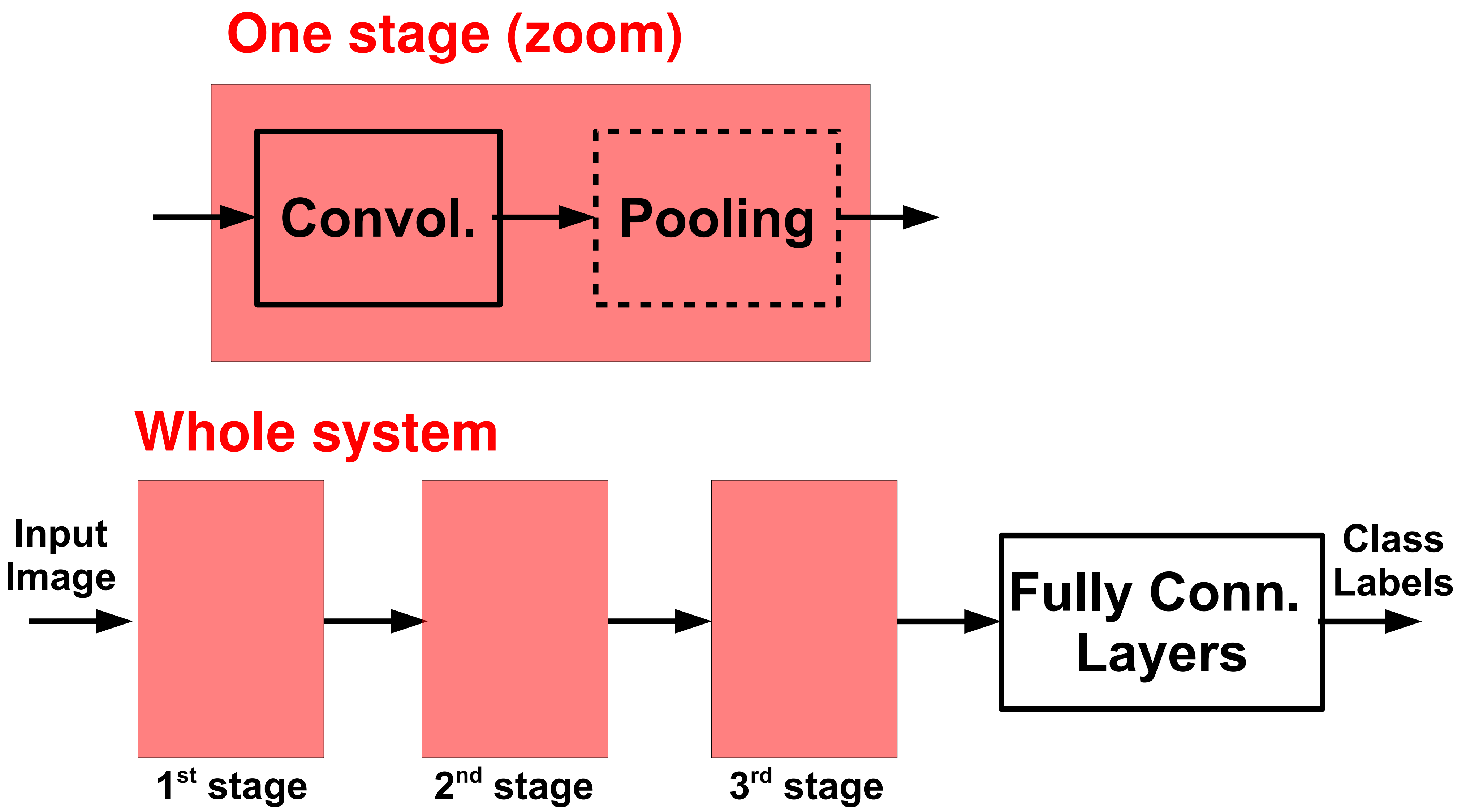
Note: after one stage the number of feature maps is usually increased (conv. layer) and the spatial resolution is usually decreased (stride in conv. and pooling layers). Receptive field gets bigger.

Reasons:
- gain invariance to spatial translation (pooling layer)
- increase specificity of features (approaching object specific units)

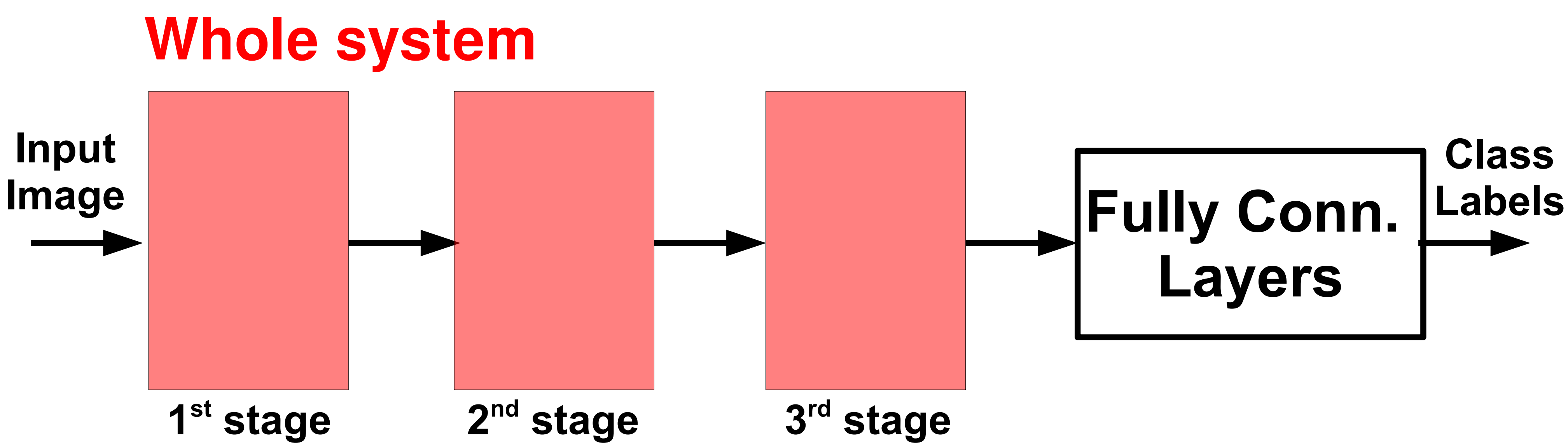


courtesy of
K. Kavukcuoglu

ConvNets: Typical Architecture



ConvNets: Typical Architecture



Conceptually similar to:

SIFT → K-Means → Pyramid Pooling → SVM

Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

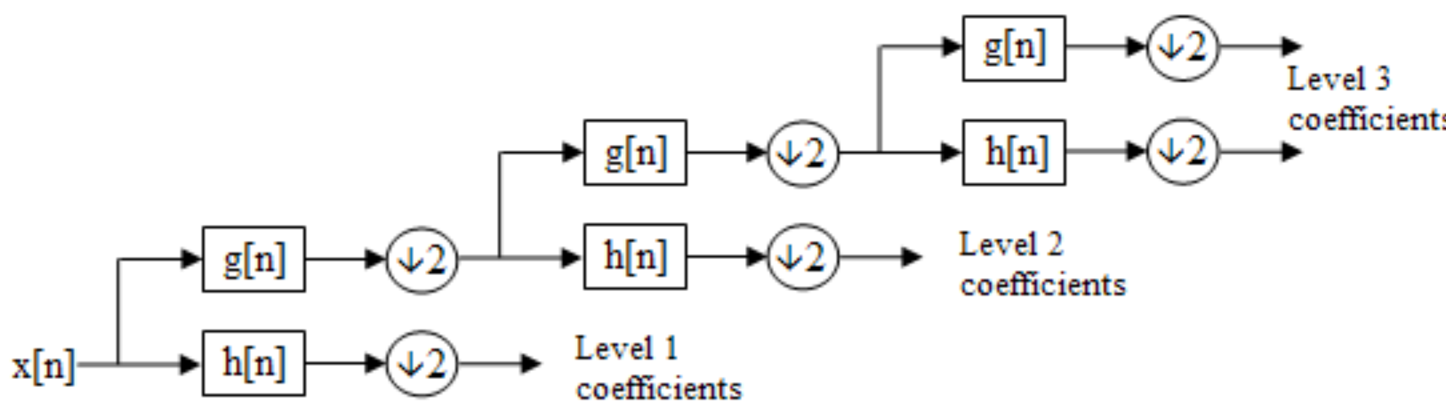
SIFT → Fisher Vect. → Pooling → SVM

Sanchez et al. "Image classifcation with F.V.: Theory and practice" IJCV 2012

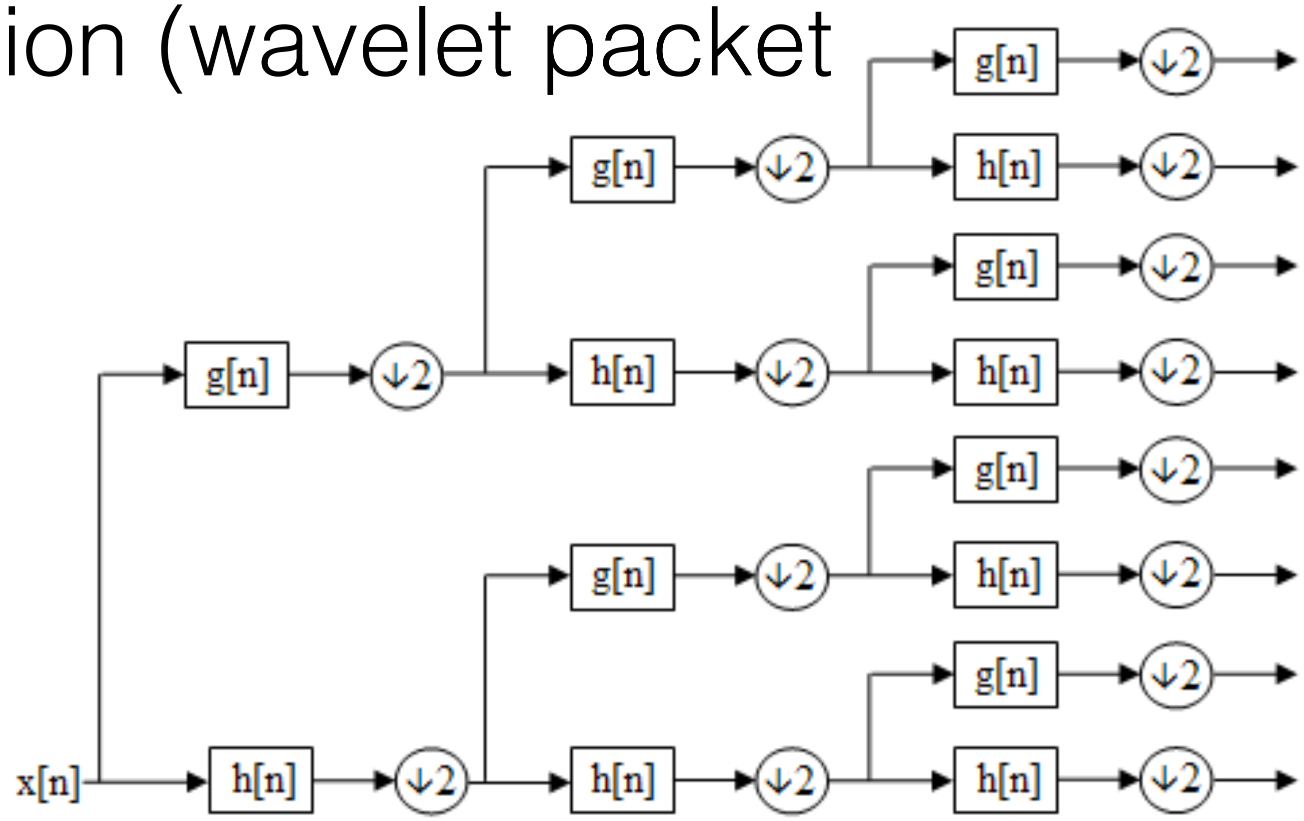
Note: all of them derive from…

ConvNets & Signal Processing

Recall a discrete wavelet transform:



and its generalization (wavelet packet decomposition):



credit: wikipedia

Why ConvNets work?

- Natural image properties:
    - spatial correlations are local
    - spatial stationarity
    - scale invariance
- Natural inductive bias:
    - Use convolutional filters of different sizes.. or even better (much more efficient in terms of compute and memory): cascade filter banks like in wavelet packet decomposition
    - Precursors of “deep” nets, except that they were linear
- CNNs extend wavelet packets by making the processing non-linear (makes the whole system more powerful and robust to noise) and by slightly adapting the filters to the task & data.
    - Note: even (small) random filters have frequency/orientation selectivity!

Bruna et al. “A mathematical motivation for complex-valued convolutional networks” Neural Comp. 2016

Why ConvNets work?

- Natural image properties:
  - spatial correlations are local
  - spatial stationarity
  - scale invariance
- Natural inductive bias:
  - Use convolutional filters of different sizes.. or even better (much more efficient in terms of compute and memory): cascade filter banks like in wavelet packet decomposition
  - Precursors of “deep” nets, except that they were linear
- CNNs extend wavelet packets by making the processing non-linear (makes the whole system more powerful and robust to noise) and by slightly adapting the filters to the task & data.
  - Note: even (small) random filters have frequency/orientation selectivity!

This is the most successful story of **deep** learning

ConvNets: Training

All layers are differentiable (a.e.).
We can use standard back-propagation.

Algorithm:

Given a small mini-batch

- F-PROP

- B-PROP

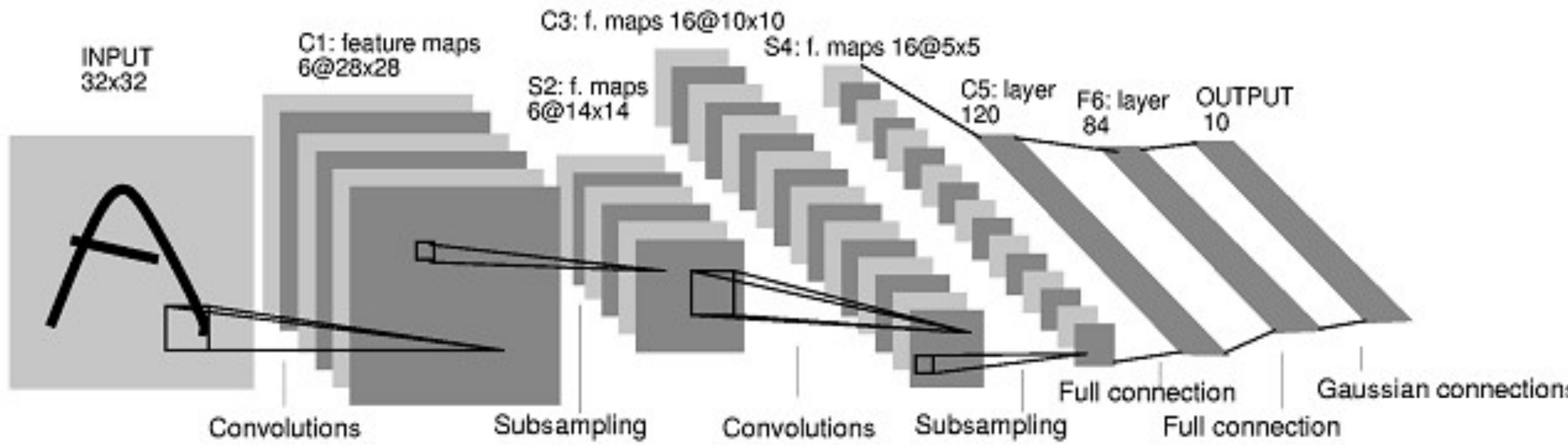
- PARAMETER UPDATE

pyTorch example of a CNN

Note: After several stages of convolution-pooling, the spatial resolution is greatly reduced (usually to about 5x5) and the number of feature maps is large (several hundreds depending on the application).

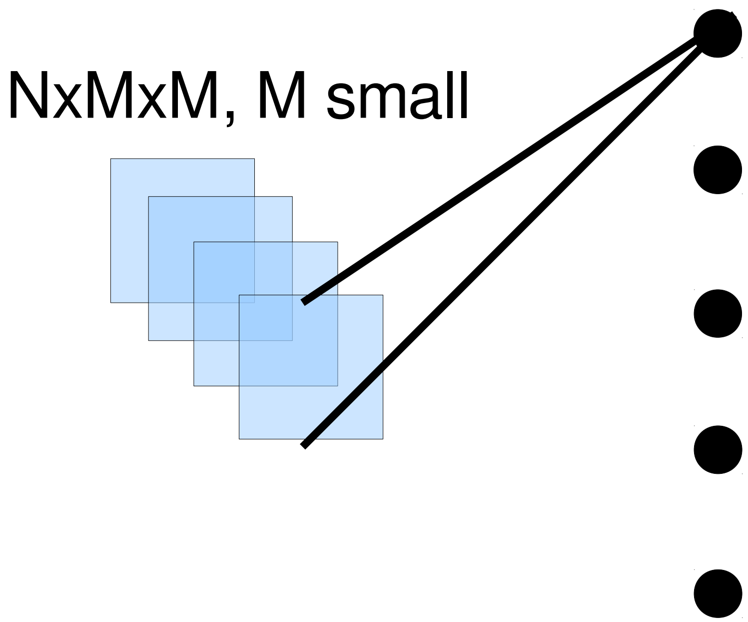
It would not make sense to convolve again (there is no translation invariance and support is too small). Everything is vectorized and fed into several fully connected layers.

If the input of the fully connected layers is of size Nx5x5, the first fully connected layer can be seen as a conv. layer with 5x5 kernels.
The next fully connected layer can be seen as a conv. layer with 1x1 kernels.

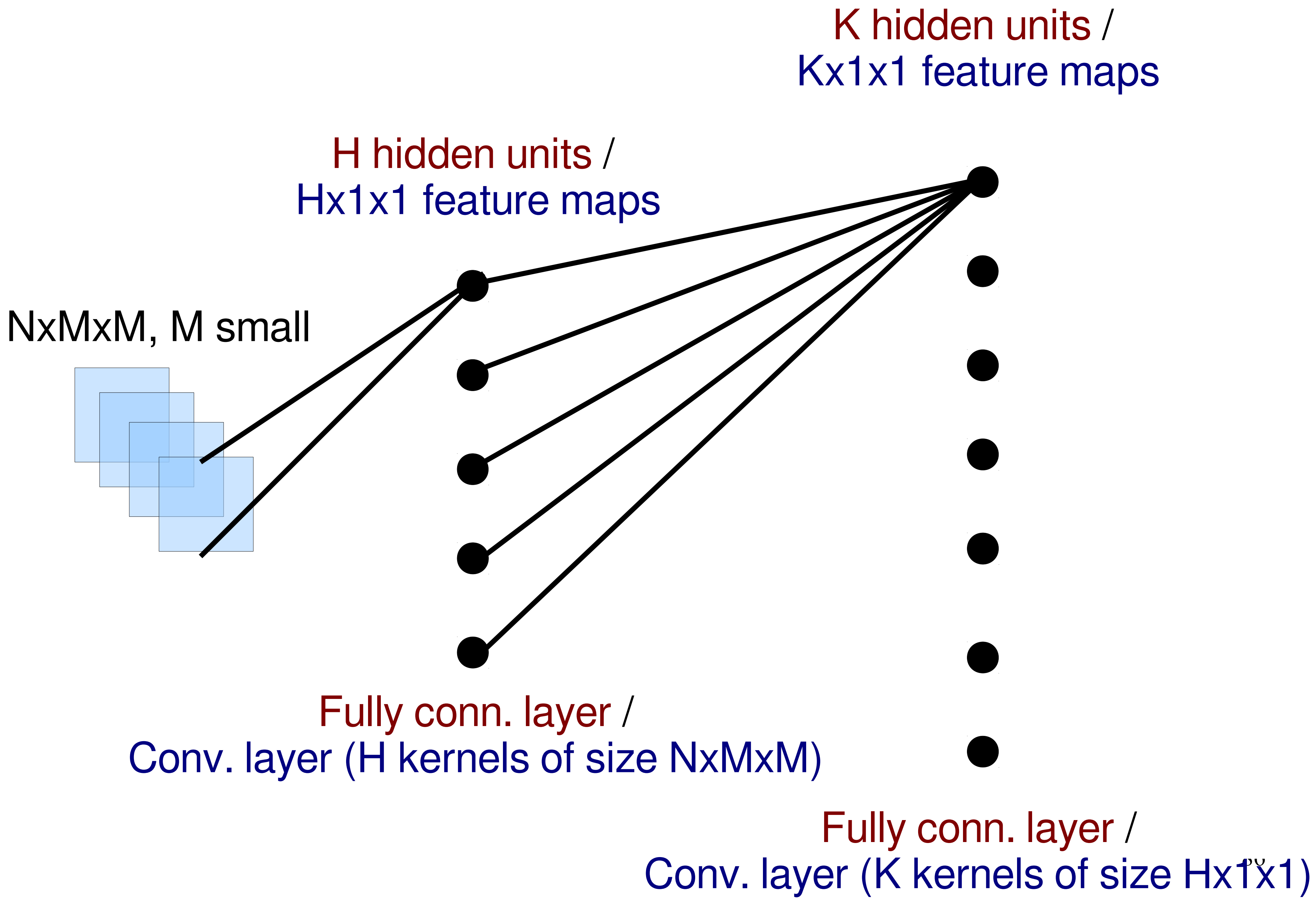


LeCun et al. “Gradient based learning applied to document recognition” IEEE 1998

H hidden units /
Hx1x1 feature maps

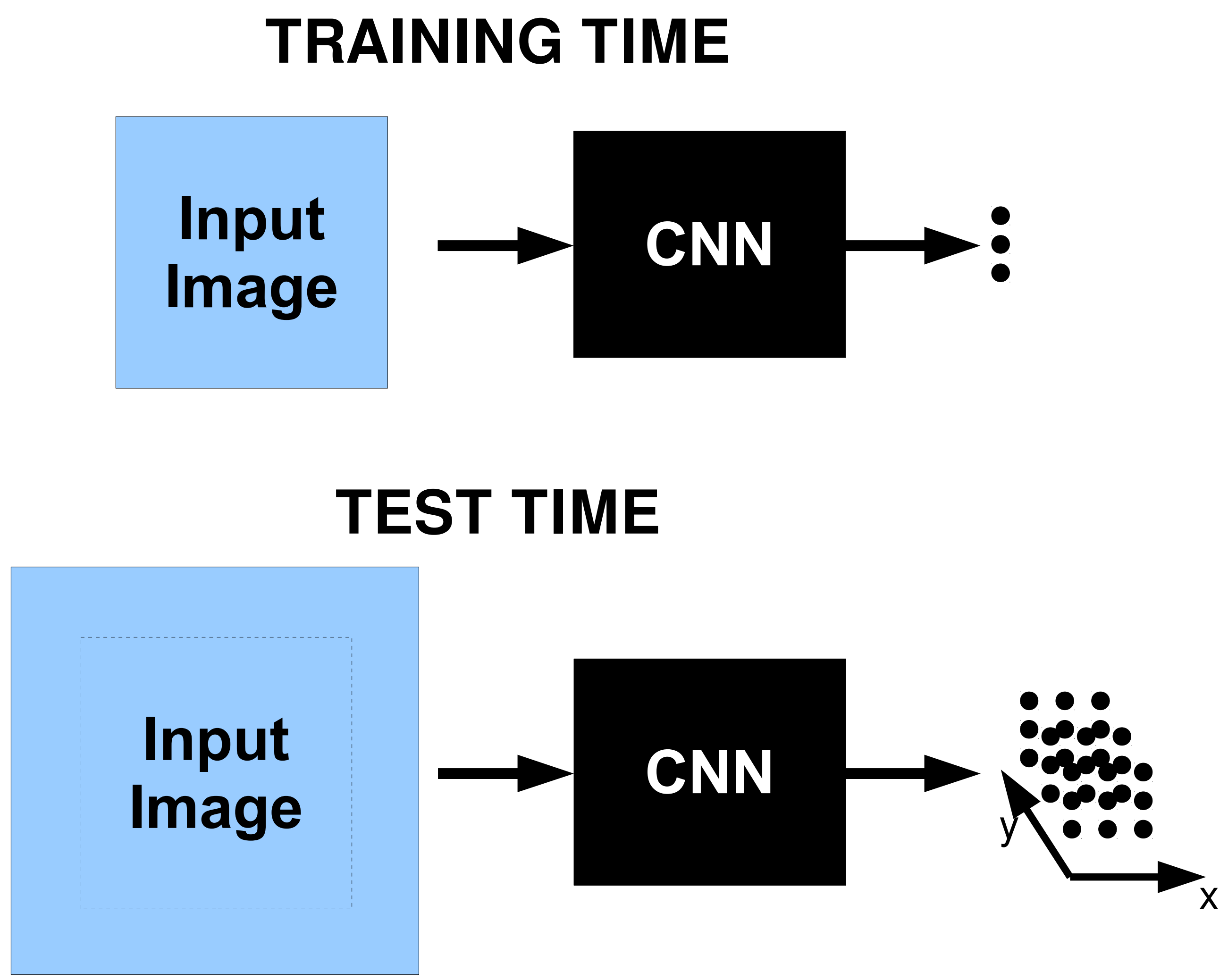


Fully conn. layer /
Conv. layer (H kernels of size NxMxM)

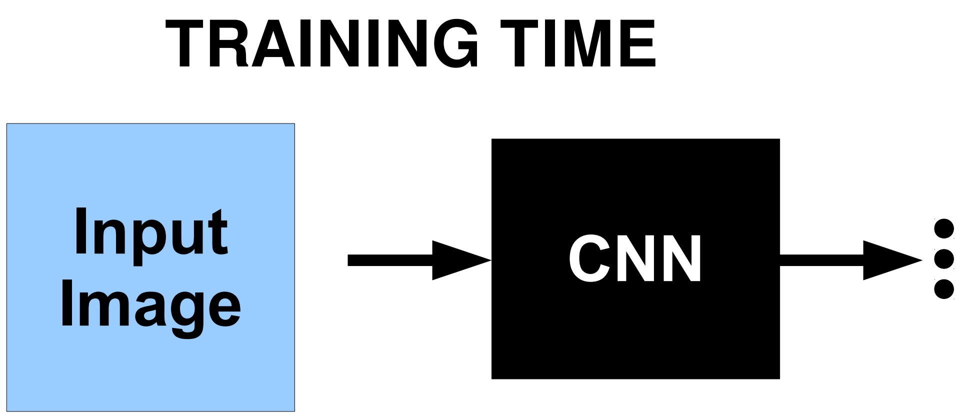


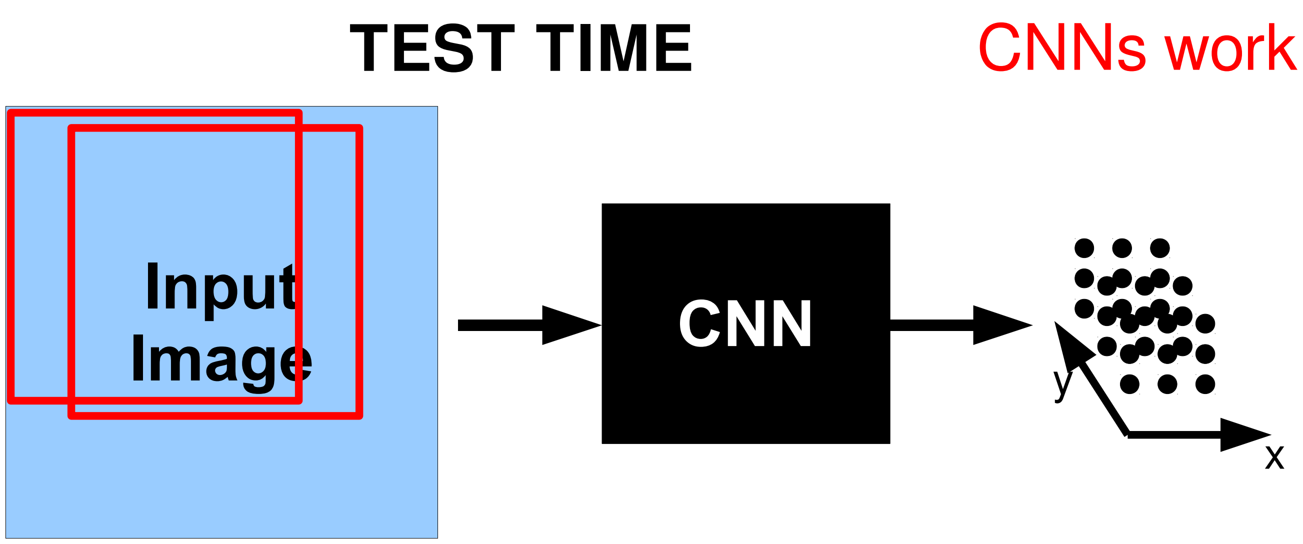
K hidden units /
Kx1x1 feature maps
H hidden units /
Hx1x1 feature maps
NxMxM, M small
Fully conn. layer /
Conv. layer (H kernels of size NxMxM)
Fully conn. layer /
Conv. layer (K kernels of size Hx1x1)

Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).



Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).



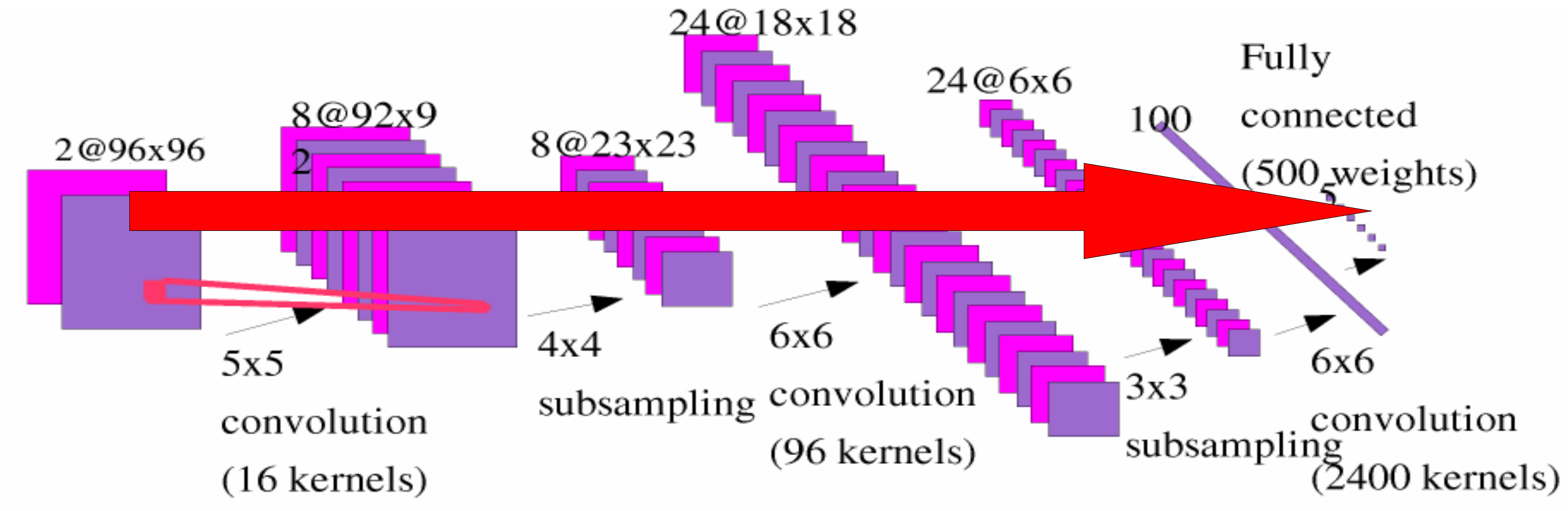


CNNs work on any image size!

Unrolling is order of magnitudes more eficient than sliding windows!

ConvNets: Test

At test time, run only is forward mode (FPROP).



Latest & Greatest CNNs: BatchNormalization

- Before a non-linearity, this layer ensures that features are well scaled.
- Improves optimization (convergence speed) and generalization.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Ioffe et al. "Batch Normalization: …" ICML 2015

Latest & Greatest CNNs: BatchNormalization

- Before a non-linearity, this layer ensures that features are well scaled.
- Improves optimization (convergence speed) and generalization.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

At test time, use running averages of mean and std.

Ioffe et al. "Batch Normalization: …" ICML 2015

Latest & Greatest CNNs: ResNet

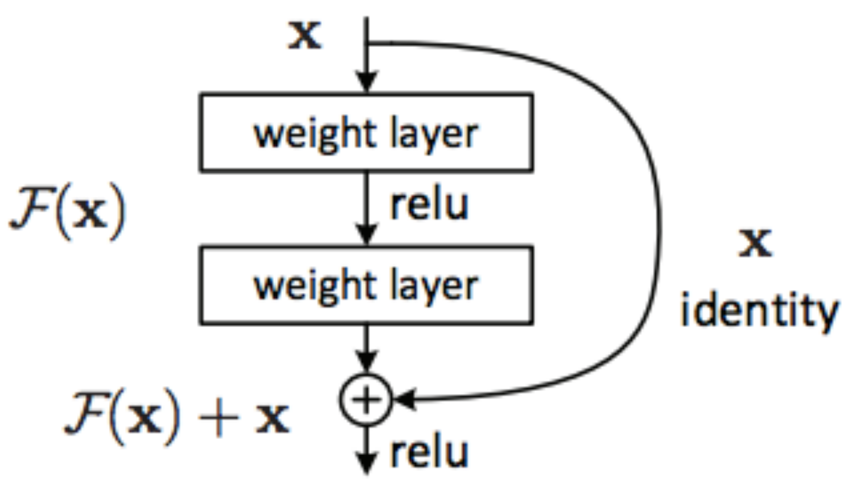
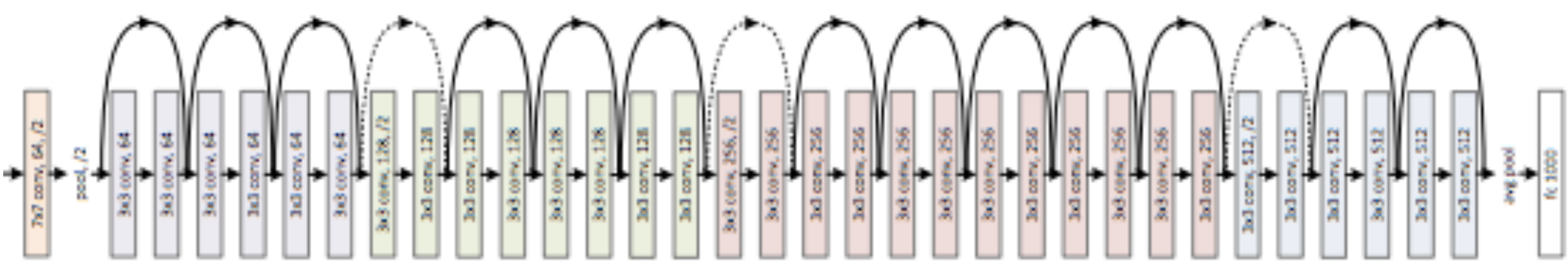


Figure 2. Residual learning: a building block.

- After each conv. layer, a batch norm. layer
- after N conv. layers, a skip connection is **summed** at the output
- No pooling layer, just strided convolutions. Whenever convolution is strided, increase number of feature maps accordingly
- No fully connected layers
- Much deeper nets (>100 layers)



He et al. "Deep Residual Learning for image recognition" arXiv 2015

Latest & Greatest CNNs: ResNet

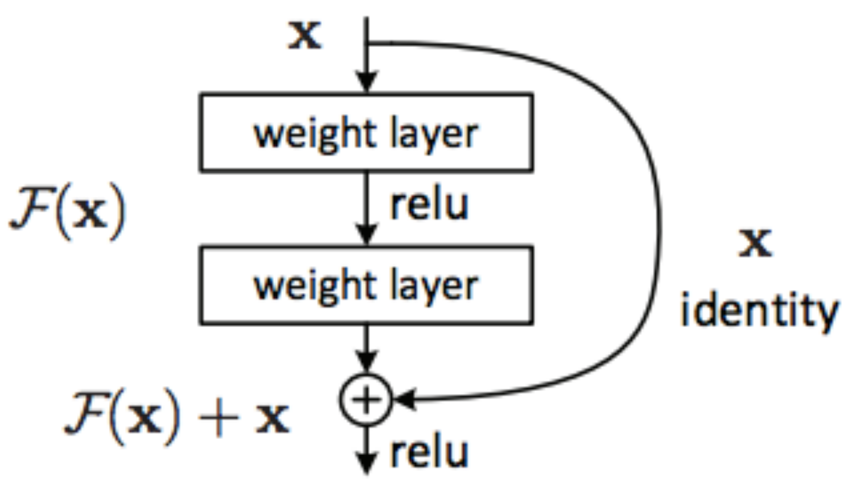
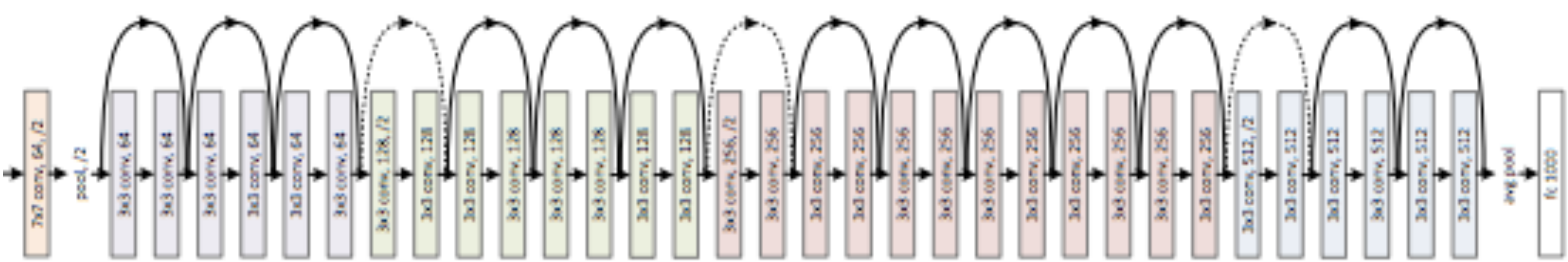


Figure 2. Residual learning: a building block.

output of 2nd block: $F_2(F_1(x) + x) + F_1(x) + x$

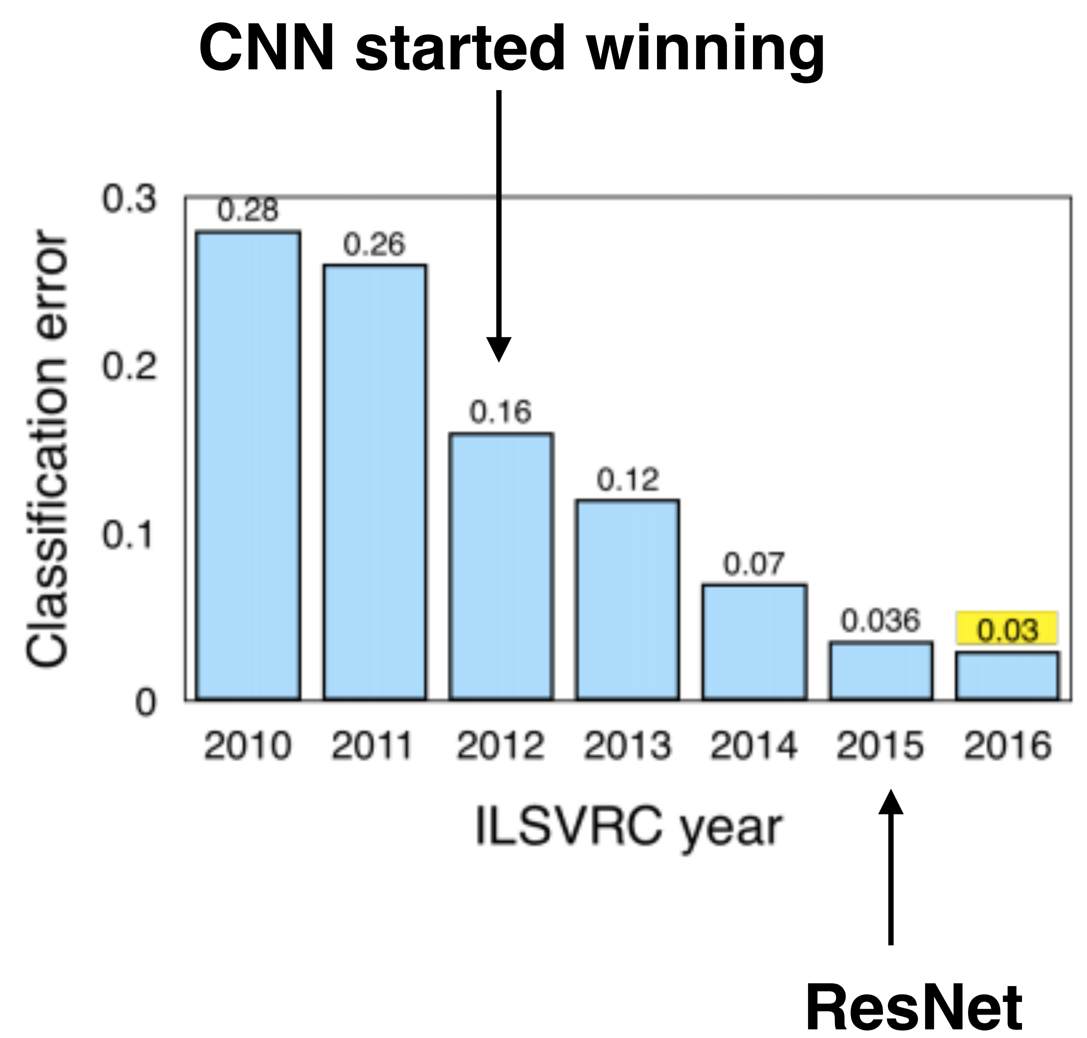
- Skip connections let gradients flow
- Features are refined at every block
- There is no massive number of parameters at the topmost layers (better generalization)
- Striding (as opposed to pooling) may introduce slight aliasing, but it does not matter and makes processing faster.



He et al. “Deep Residual Learning for image recognition” arXiv 2015

Latest & Greatest CNNs: ResNet

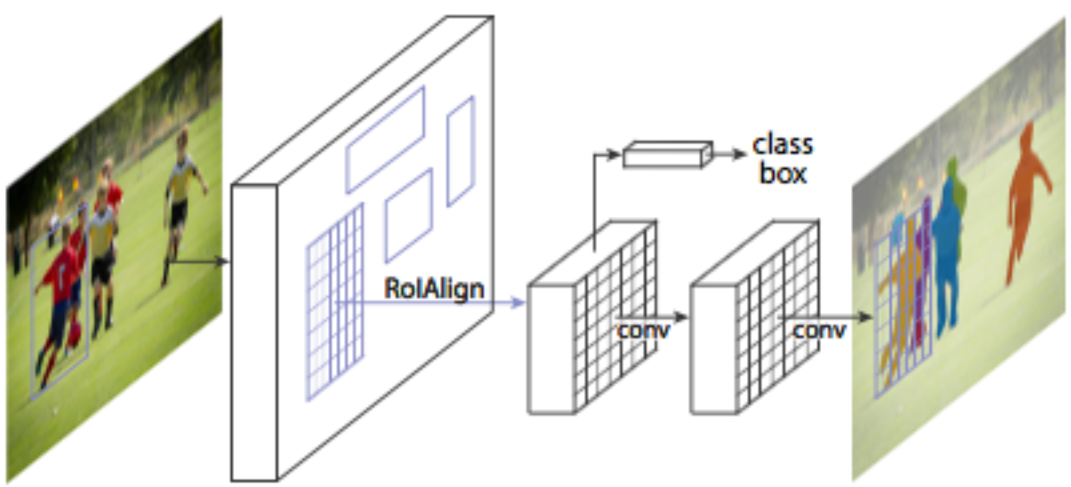
ImageNet competition
(1M images, 1K categories):



He et al. "Deep Residual Learning for image recognition" arXiv 2015

Latest & Greatest CNNs: Mask R-CNN

A much more challenging task: instance segmentation



For every object predict:

- Predict bounding box
- Predict class label
- Predict mask

He et al. “Mask R-CNN” arXiv 2017

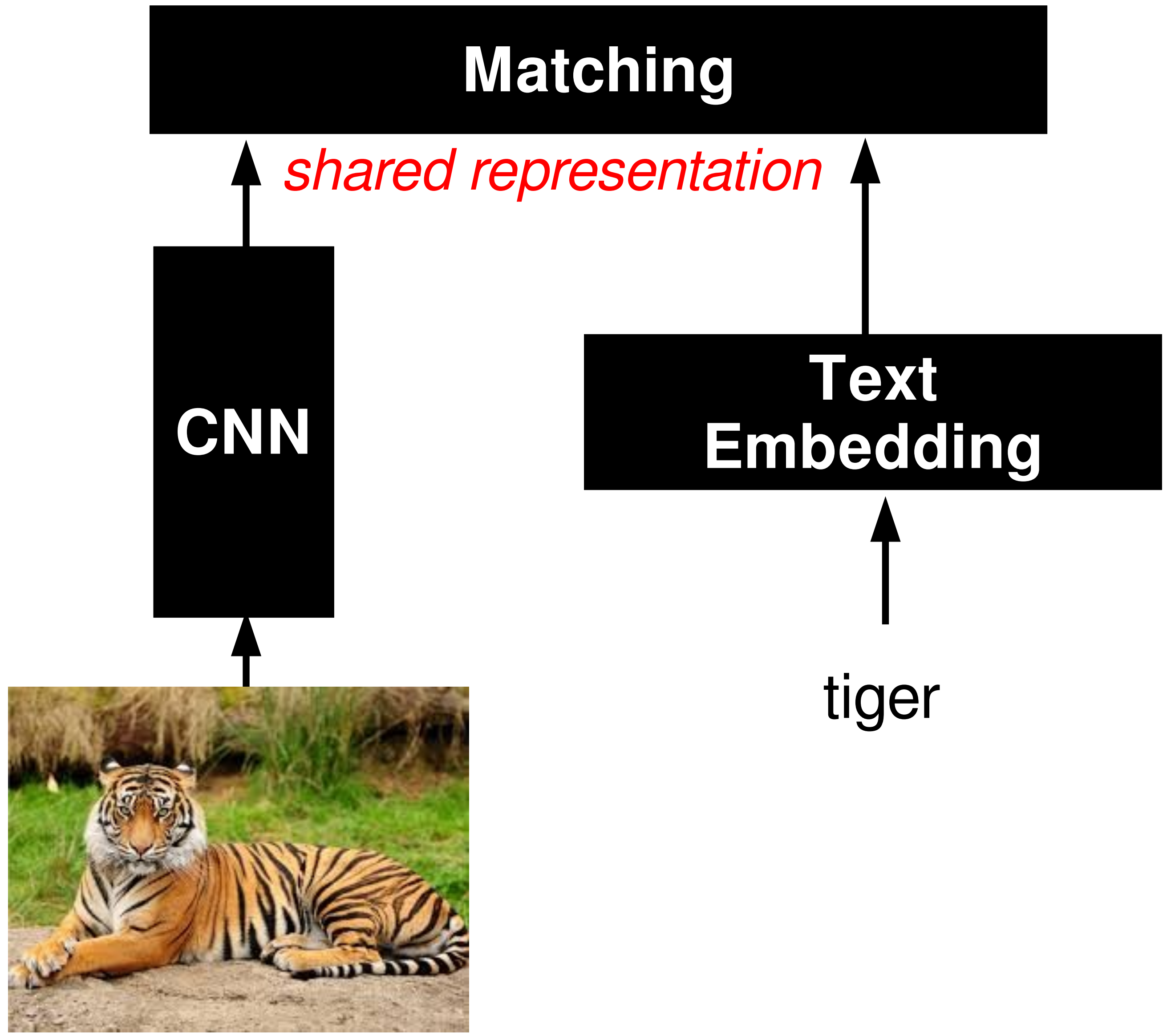
Latest & Greatest CNNs: Mask R-CNN

He et al. “Mask R-CNN” arXiv 2017

Latest & Greatest CNNs: Mask R-CNN

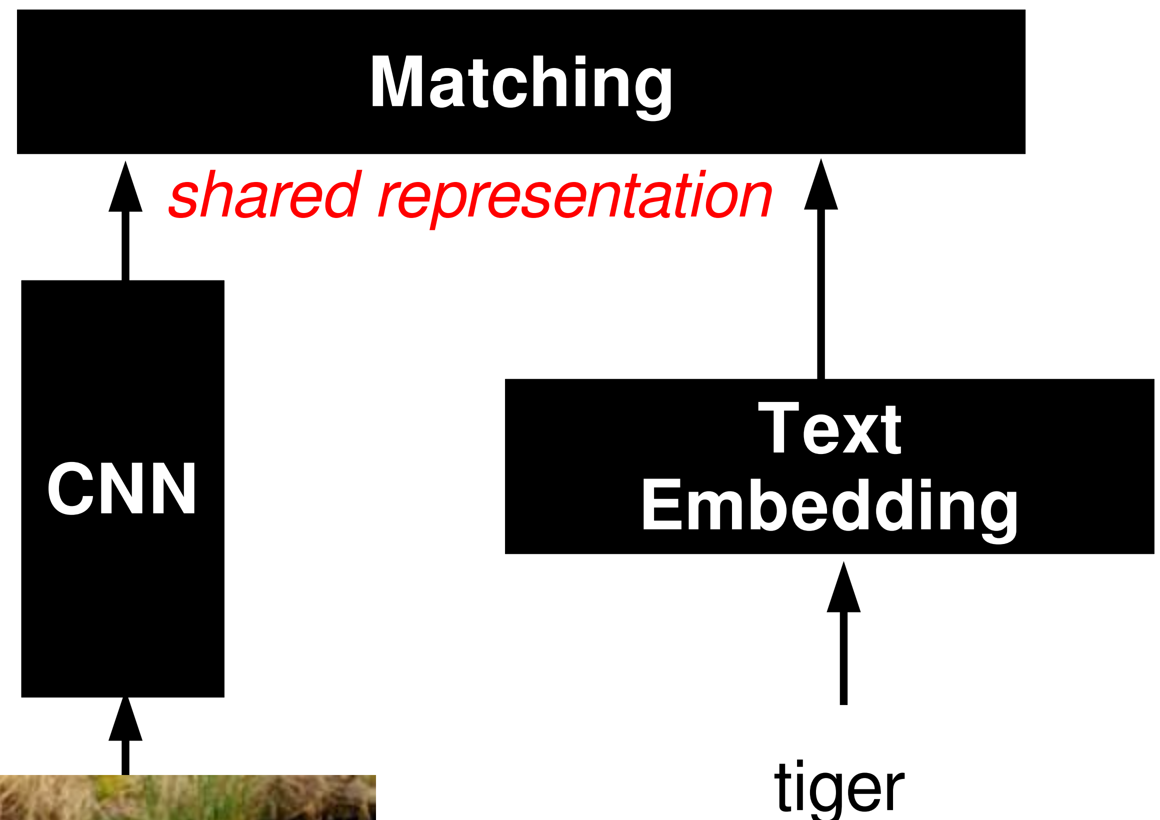
He et al. “Mask R-CNN” arXiv 2017

Fancier Architectures: Multi-Modal



Frome et al. “Devise: a deep visual semantic embedding model” NIPS 2013

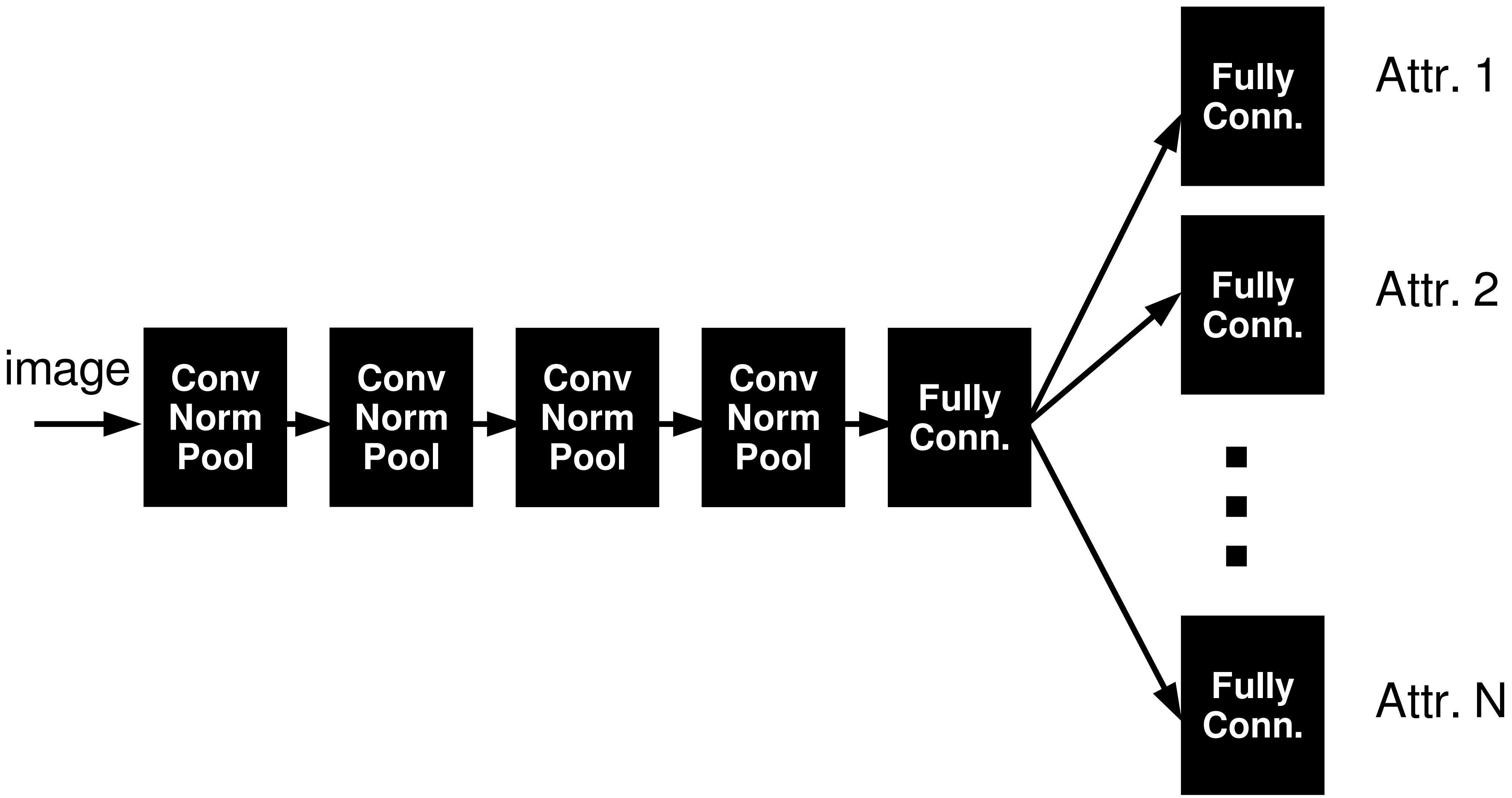
Fancier Architectures: Multi-Modal



We will discuss more recent works during the 3rd lecture!

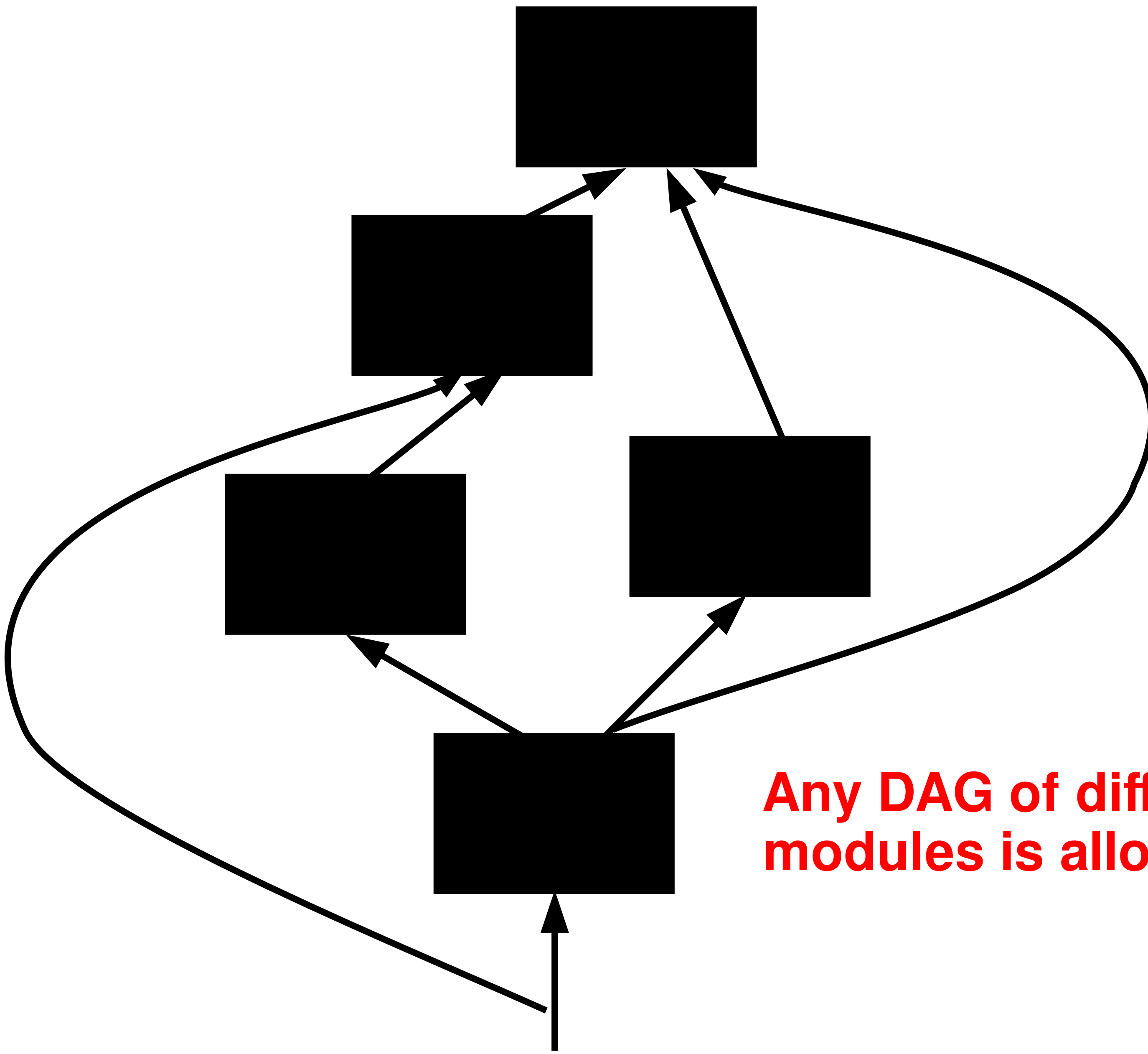
Frome et al. “Devise: a deep visual semantic embedding model” NIPS 2013

Fancier Architectures: Multi-Task



Zhang et al. “PANDA..” CVPR 2014

Fancier Architectures: Generic DAG

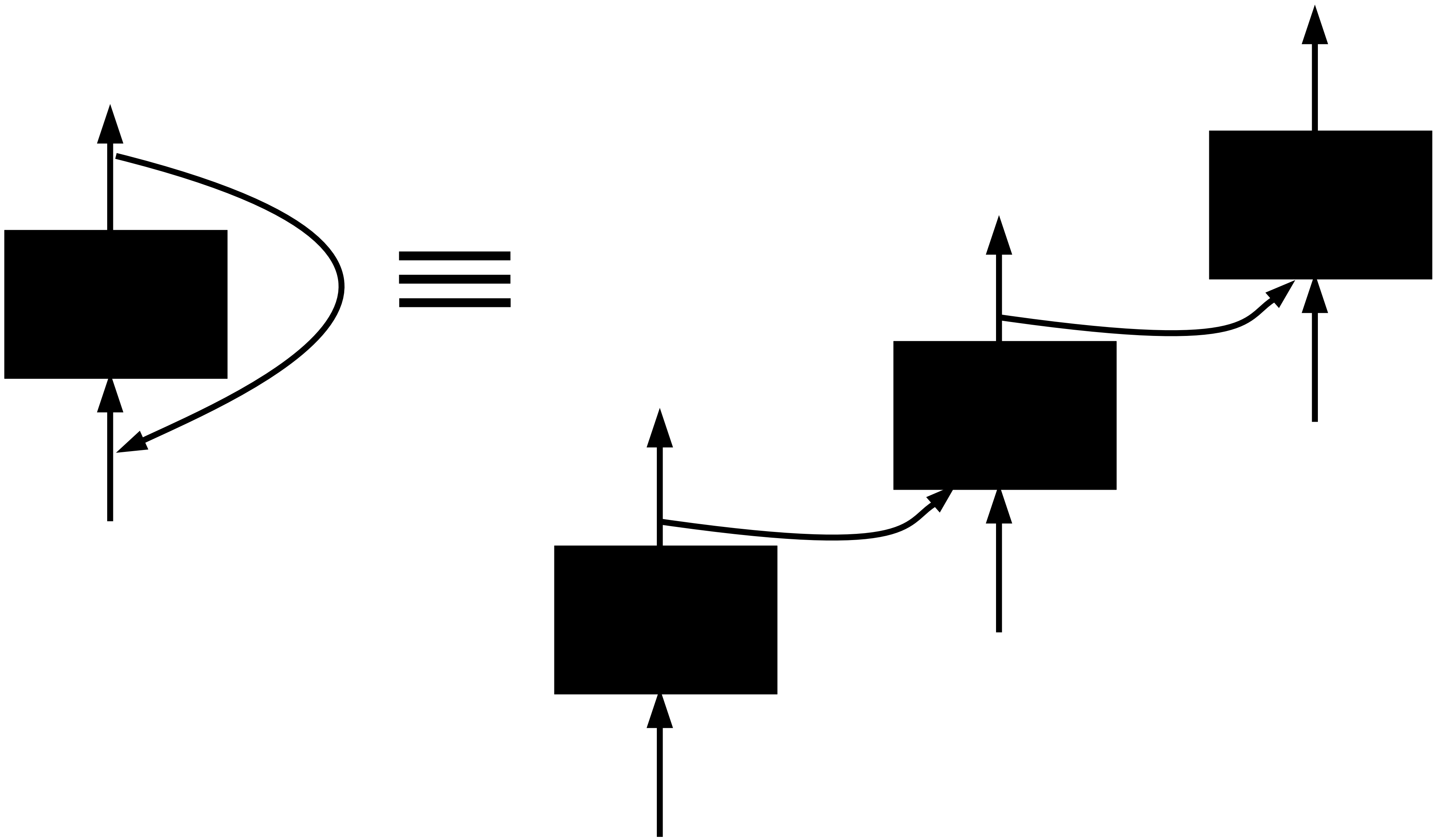


Any DAG of differentialble modules is allowed!

Andreas et al. “Learning to compose neural networks for Q&A” NAACL 2016
Johnson et al. “Inferring and executing programs for visual reasoning” arXiv 2017

Fancier Architectures: Generic DAG

If there are cycles (RNN), one needs to un-roll it.



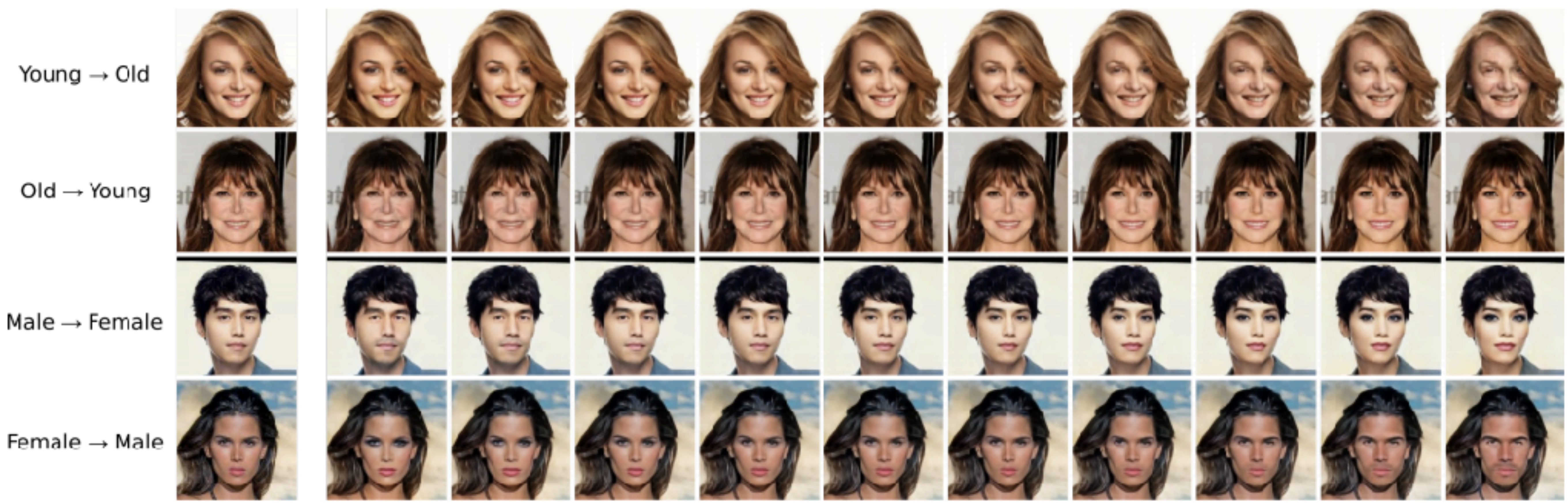
Pinheiro, Collobert "Recurrent CNN for scene labeling" ICML 2014
Graves "Offline Arabic handwriting recognition.." Springer 2012

CNNs for Image Generation

Radford et al. “Unsupervised representation learning…” ICLR 2016

CNNs for Image Generation

Fantasizing faces with different attributes (age, gender, glasses, etc.):



Lampe et al. "Fader Networks:…" arXiv 2017

Tips of the trade

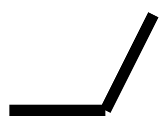
Choosing the Architecture

- It's totally task dependent. What works for recognition is rather different than generation, for instance.
- For classification of natural images, ResNet is probably the best bet, as of today.
- If the task is related to classification of natural looking images and data is scarce, it's usually a good idea to initialize from a pre-trained model. CNNs features generalize surprisingly well!
- Ultimately, one needs to cross-validate.
- The more labeled data is available, the more layers and the more filters usually yield better accuracy. Computational resources should be taken into account.
- Leverage domain knowledge to design the architecture, be creative :)

How To Optimize [nonissue]

- SGD (with momentum) usually works very well

- Pick learning rate by running on a subset of the data
  Bottou "Stochastic Gradient Tricks" Neural Networks 2012
  - Start with large learning rate and divide by 2 until loss does not diverge
  - Decay learning rate by a factor of ~1000 or more by the end of training

- Use  non-linearity

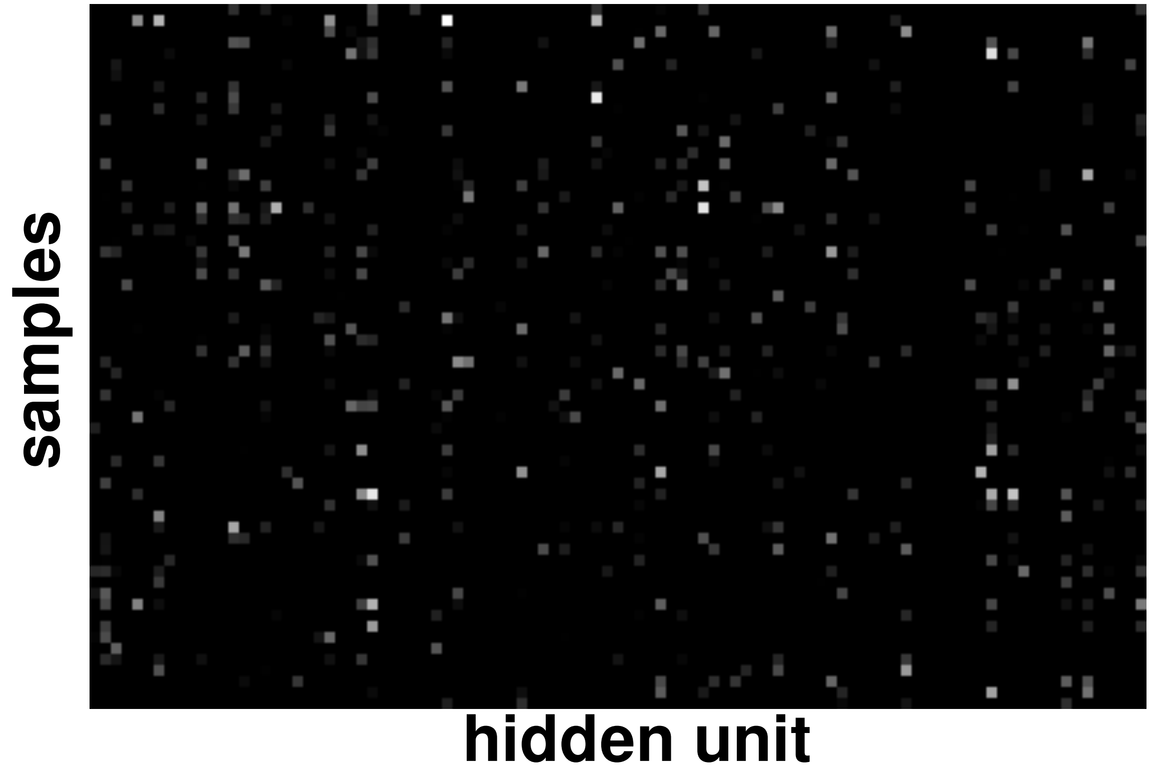
- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

Improving Generalization

- Weight sharing (greatly reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, etc.)
- Dropout
  Hinton et al. "Improving Nns by preventing co-adaptation of feature detectors" arxiv 2012
- Weight decay (L2, L1)
- Sparsity in the hidden units
- Multi-task (unsupervised learning)

Good To Know

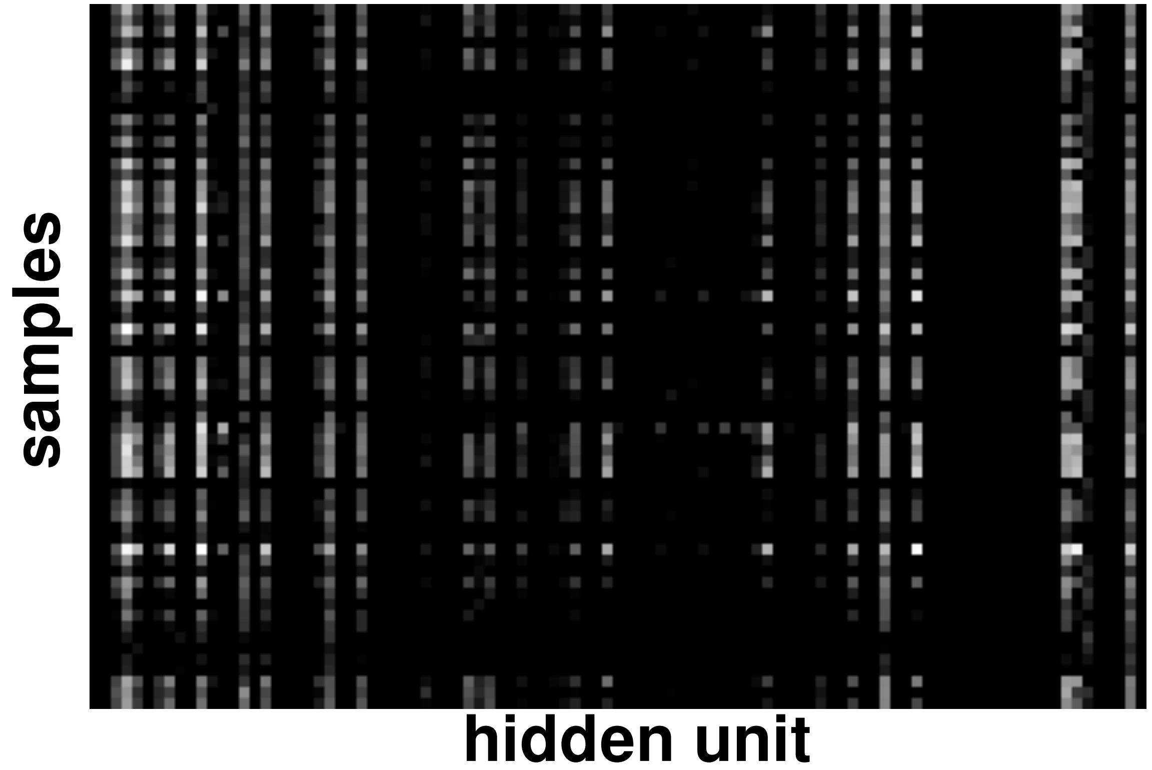
- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.



Good training: hidden units are sparse across samples and across features.

Good To Know

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.

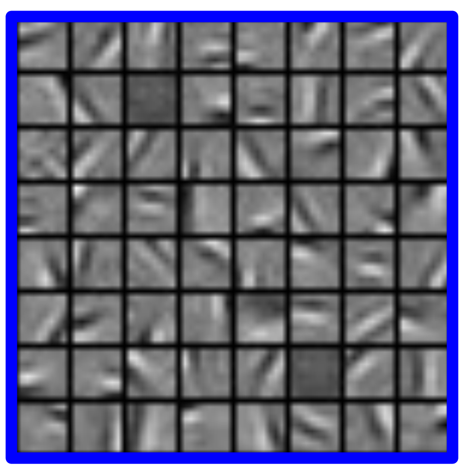


Bad training: many hidden units ignore the input and/or exhibit strong correlations.

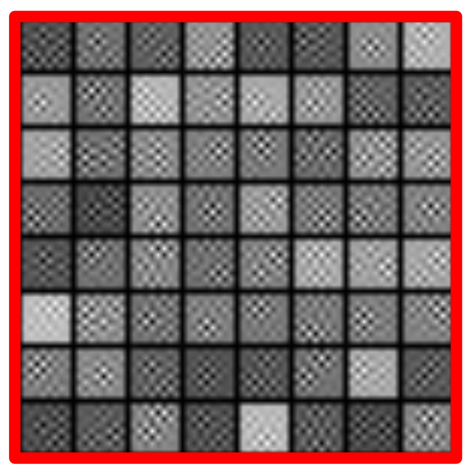
Good To Know

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters

GOOD

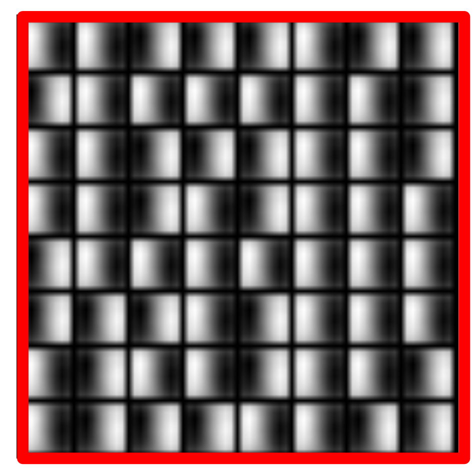


BAD



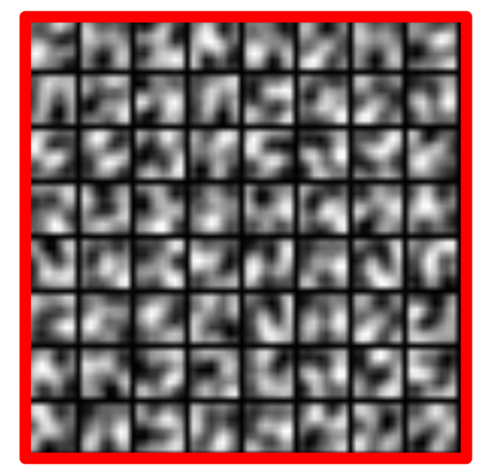
too noisy

BAD



too correlated

BAD



lack structure

Good training: learned filters exhibit structure and are uncorrelated.

Zeiler, Fergus “Visualizing and understanding CNNs” arXiv 2013
Simonyan, Vedaldi, Zisserman “Deep inside CNNs: visualizing image classification models..” ICLR 2014

Good To Know

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters
- Measure error on both training and validation set.
- Train and test on a small subset of the data and check that the error goes to 0 quickly.

What If It Does Not Work?

- Training diverges:
  - Learning rate may be too large $\rightarrow$ decrease learning rate
  - BPROP is buggy $\rightarrow$ numerical gradient checking
- Parameters collapse / loss is minimized but accuracy is low
  - Check loss function:
    - Is it appropriate for the task you want to solve?
    - Does it have degenerate solutions? Check "pull-up" term.
- Network is underperforming
  - Compute flops and nr. params. $\rightarrow$ if too small, make net larger
  - Visualize hidden units/params $\rightarrow$ fix optmization
- Network is too slow
  - Compute flops and nr. params. $\rightarrow$ GPU,distrib. framework, make net smaller

Questions?

Acknowledgements

I would like to thank Ross Girshick for providing slide material about ResNet & Mask R-CNN, and Arthur Szlam for sharing his insights about why CNNs work.