Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Neural Networks Architectures for RL

CMU 10703

Katerina Fragkiadaki

Part of slides borrowed from Russ who borrowed from Hugo Larochelle

Large-Scale Reinforcement Learning

- Reinforcement learning should be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter, robot, …: enormous continuous state space
  -

- Tabular methods clearly cannot handle this.. why?

Value Function Approximation (VFA)

- So far we have represented value functions by a lookup table
  - Every state s has an entry V(s), or
  - Every state-action pair (s,a) has an entry Q(s,a)
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
  - You cannot generalize across states!
- Solution for large MDPs:
  - Estimate value function with function approximation

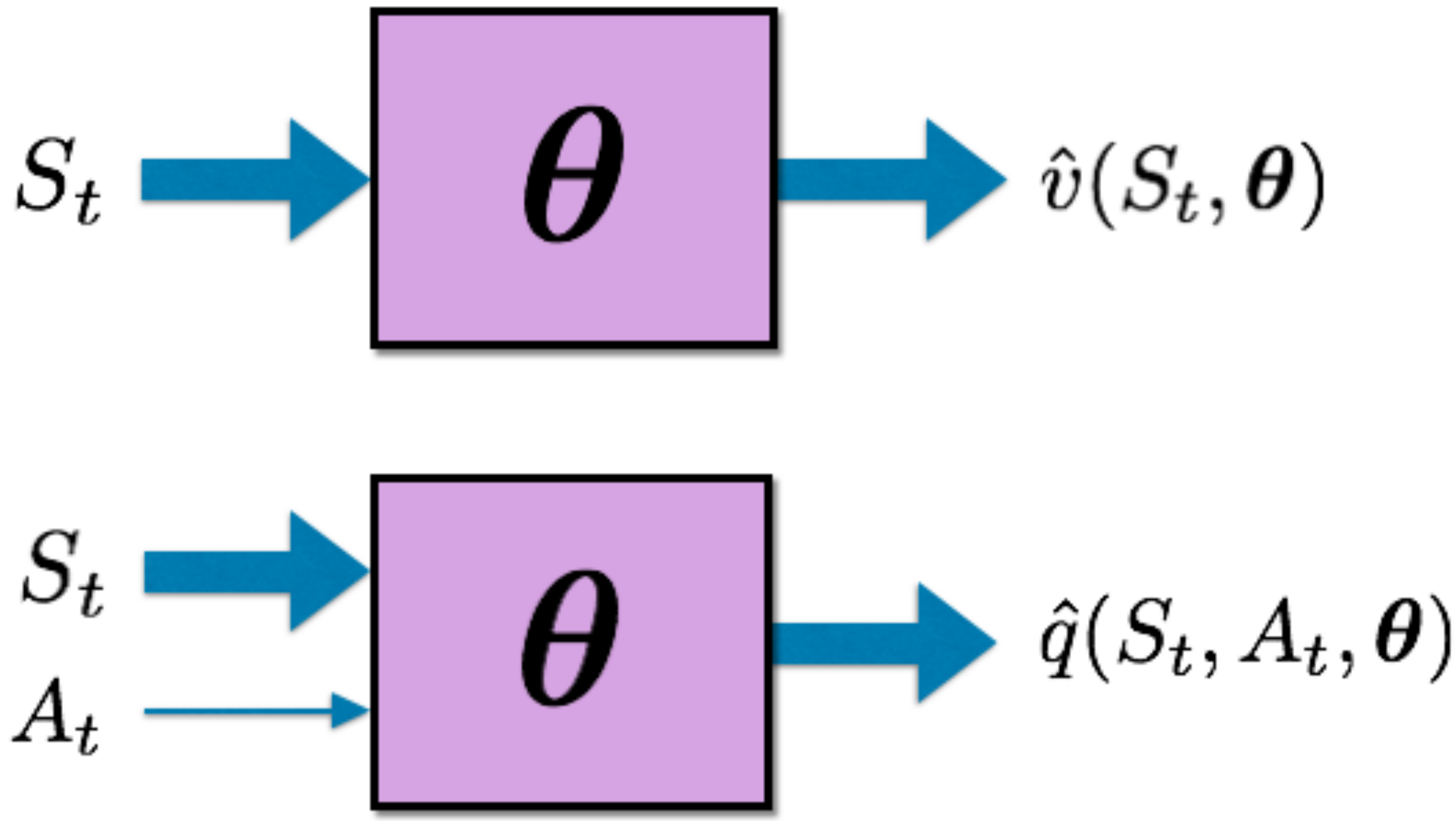
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states Why?

Value Function Approximation (VFA)

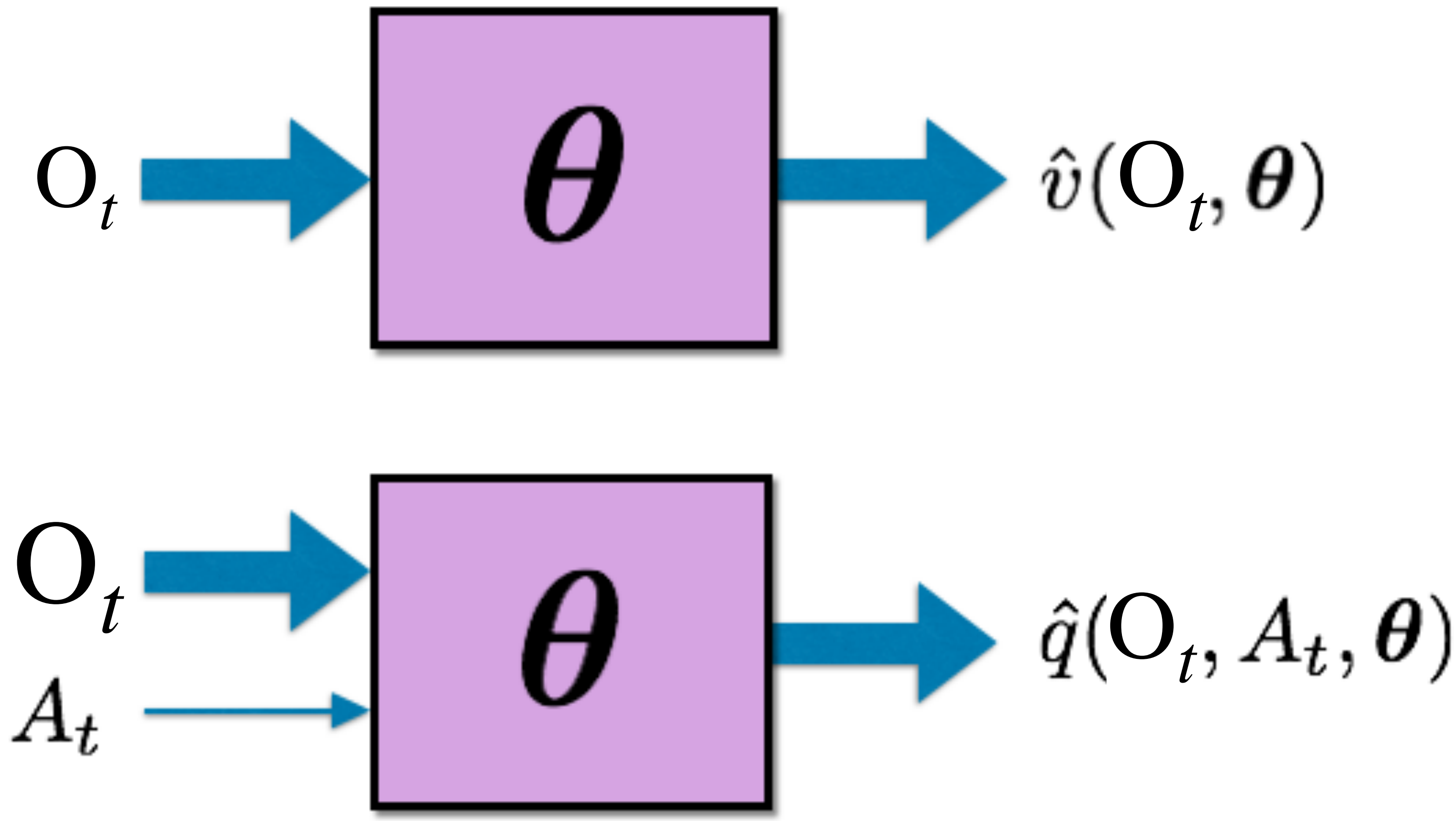
- Value function approximation (VFA) replaces the table with a general parameterized form:



Why this is a good idea?
Because those functions can be trained to map *similar* states to similar values. E.g., for a navigation agent the color of the carpet does not matter, or how many vases are on the table, let alone the amount of sun coming in from the window.

End-to-End RL

- End-to-end RL methods replace the hand-designed state representation with raw observations.



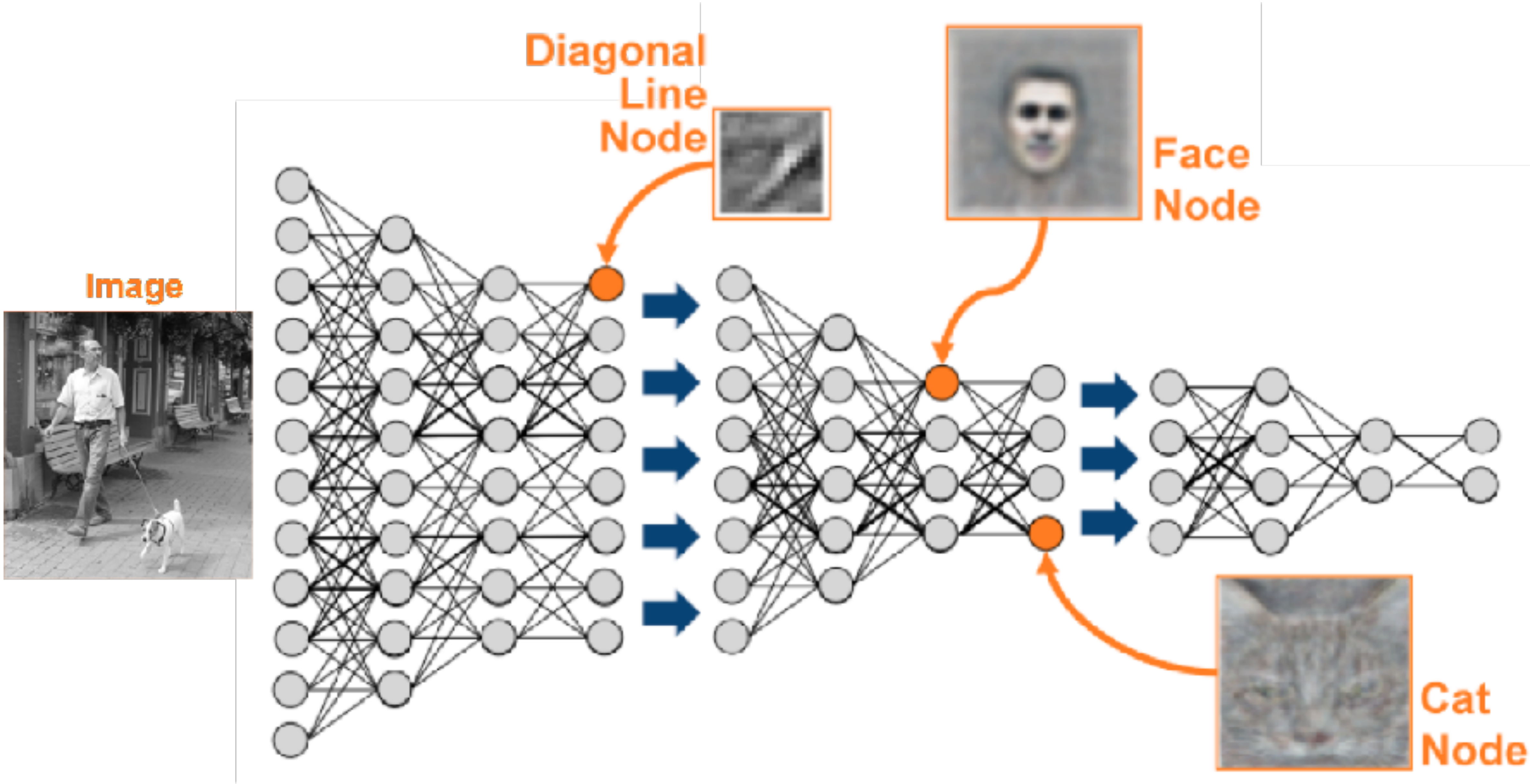
- We get rid of manual design of state representations :-)
- We need tons of data to train the network since O_t usually WAY more high dimensional than hand-designed S_t :-(

Which Function Approximation?

- There are many function approximators, e.g.
  - Linear combinations of features
  - Neural networks
  - Decision tree
  - Nearest neighbour
  - …

- In this lecture we will consider:
  - Linear combinations of features
  - **Neural networks**

Deep Learning



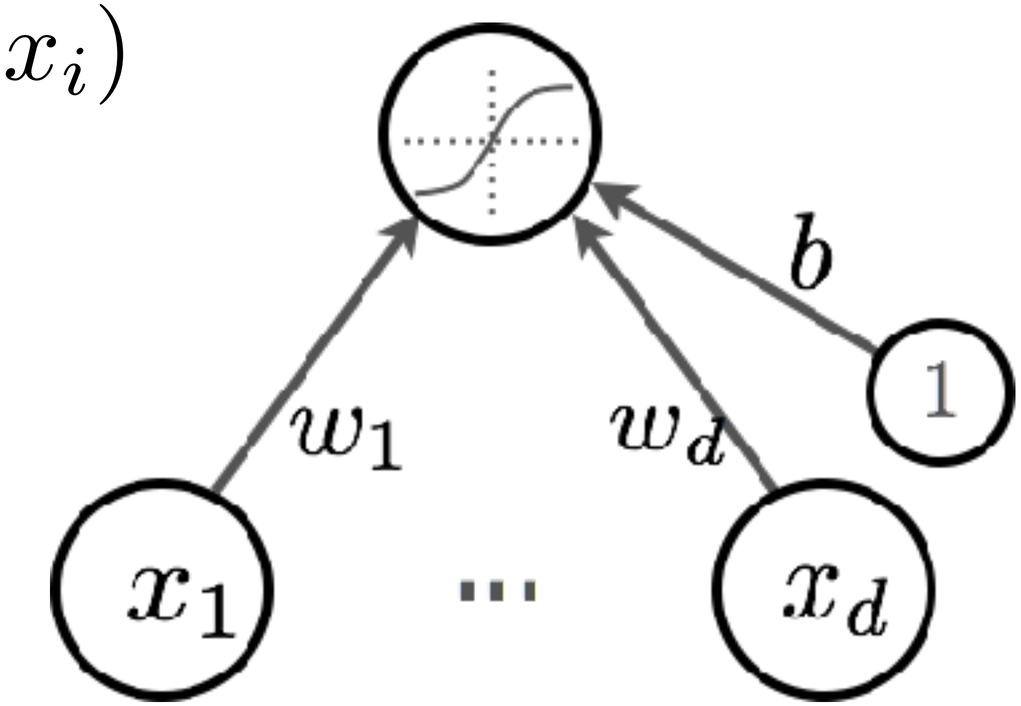
Artificial Neuron

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron output activation:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



where

$\mathbf{w}$ are the weights (parameters)

b is the bias term

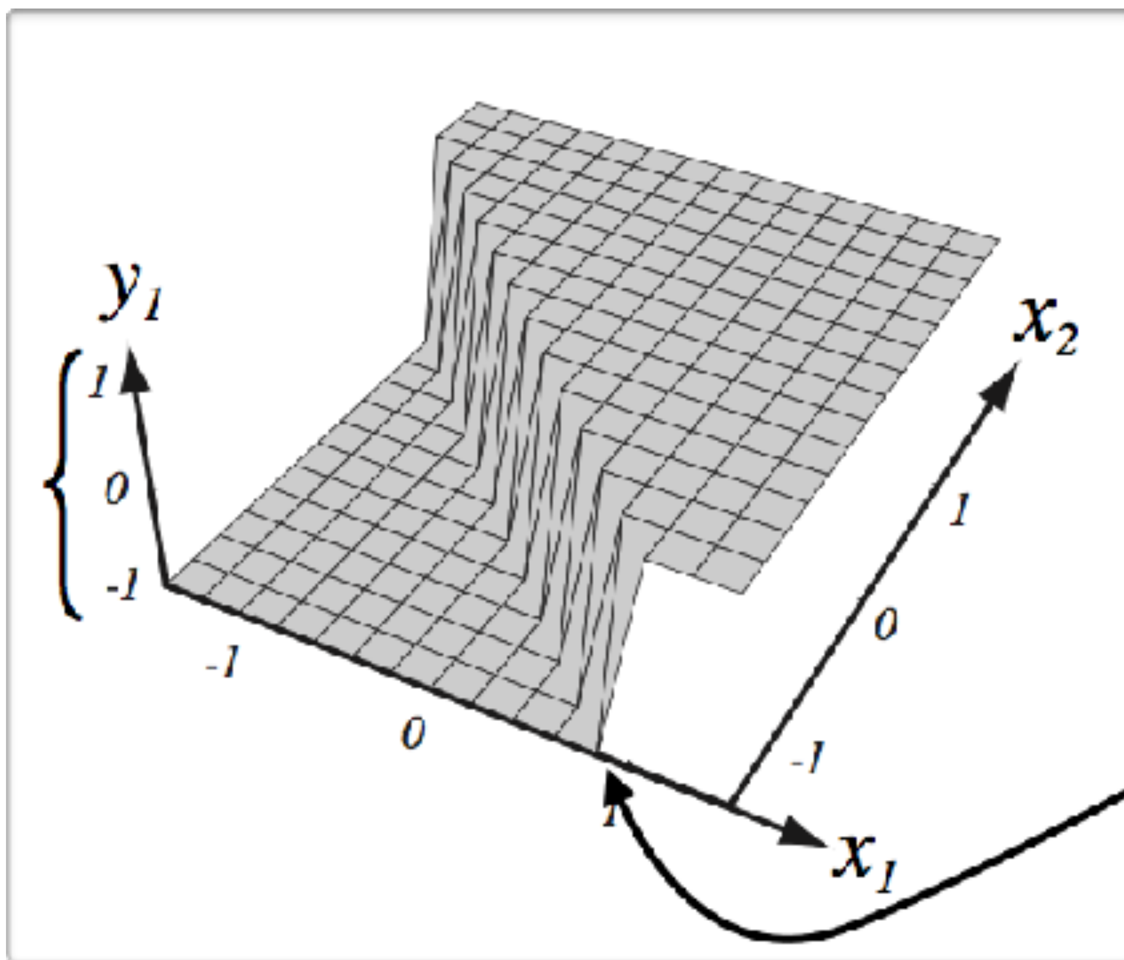
$g(\cdot)$ is called the activation function

Artificial Neuron

- Output activation of the neuron:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

Range is determined by $g(\cdot)$



Bias only changes the position of the riff

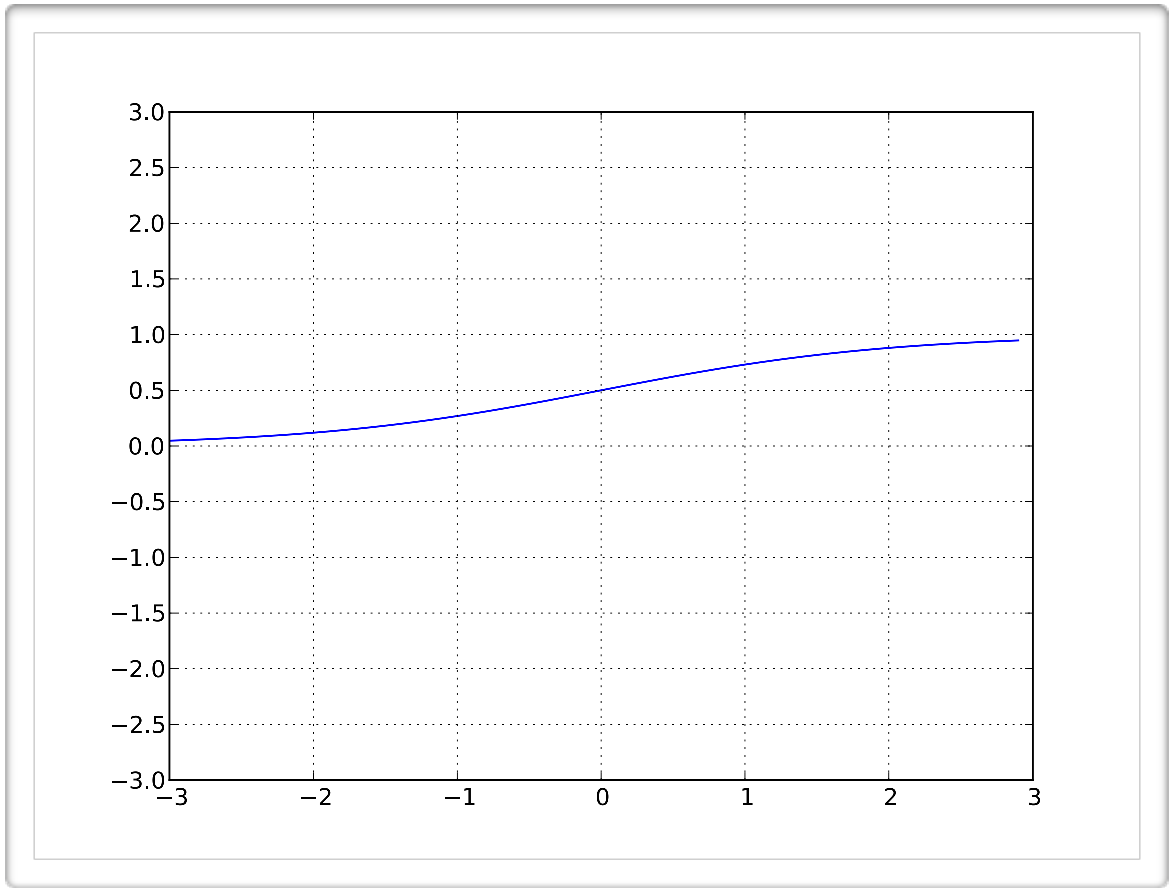
(from Pascal Vincent's slides)

Activation Function

- Sigmoid activation function:

$$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$$

  - ➢ Squashes the neuron's output between 0 and 1
  - ➢ Always positive
  - ➢ Bounded
  - ➢ Strictly Increasing

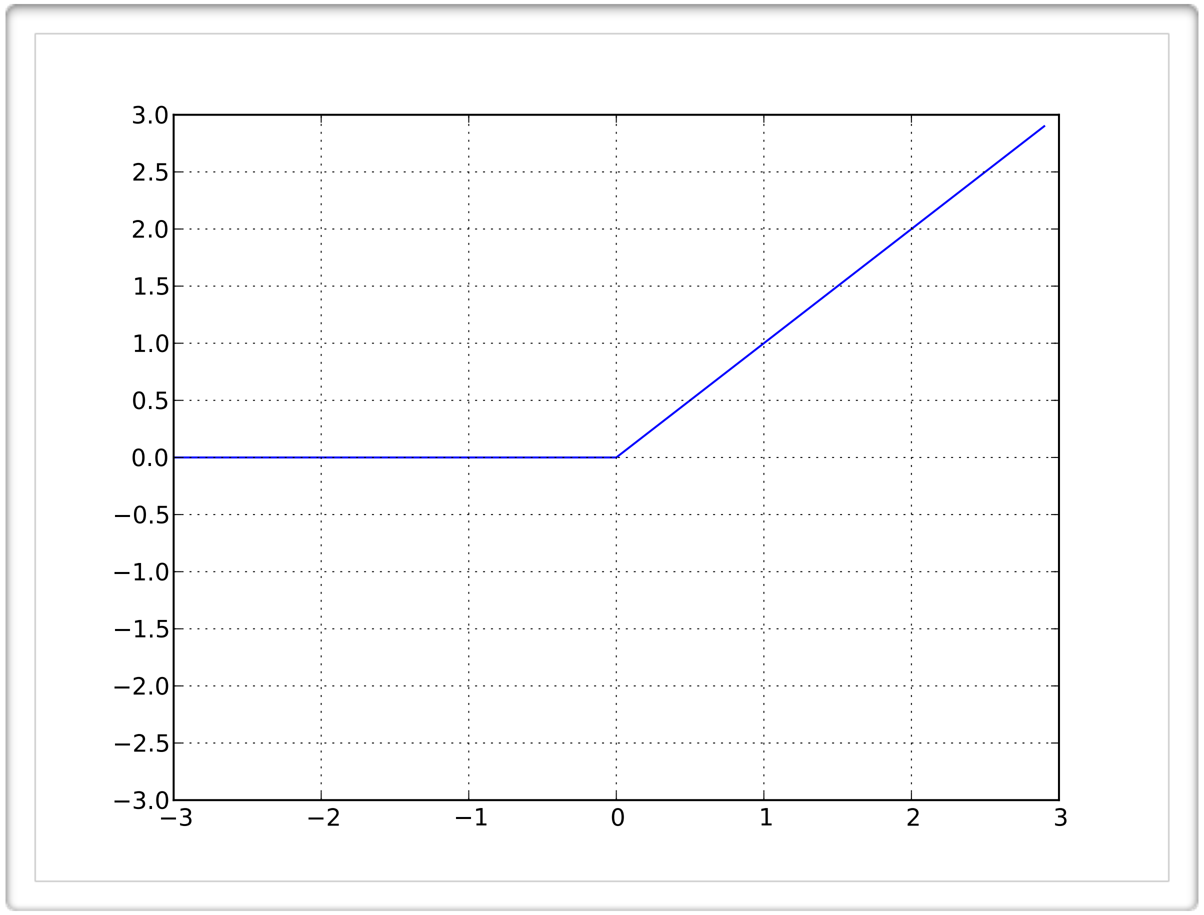


Activation Function

- Rectified linear (ReLU) activation function:

$$g(a) = \mathrm{reclin}(a) = \max(0, a)$$

  - ➢ Bounded below by 0 (always non-negative)
  - ➢ Tends to produce units with sparse activities
  - ➢ Not upper bounded
  - ➢ Strictly increasing

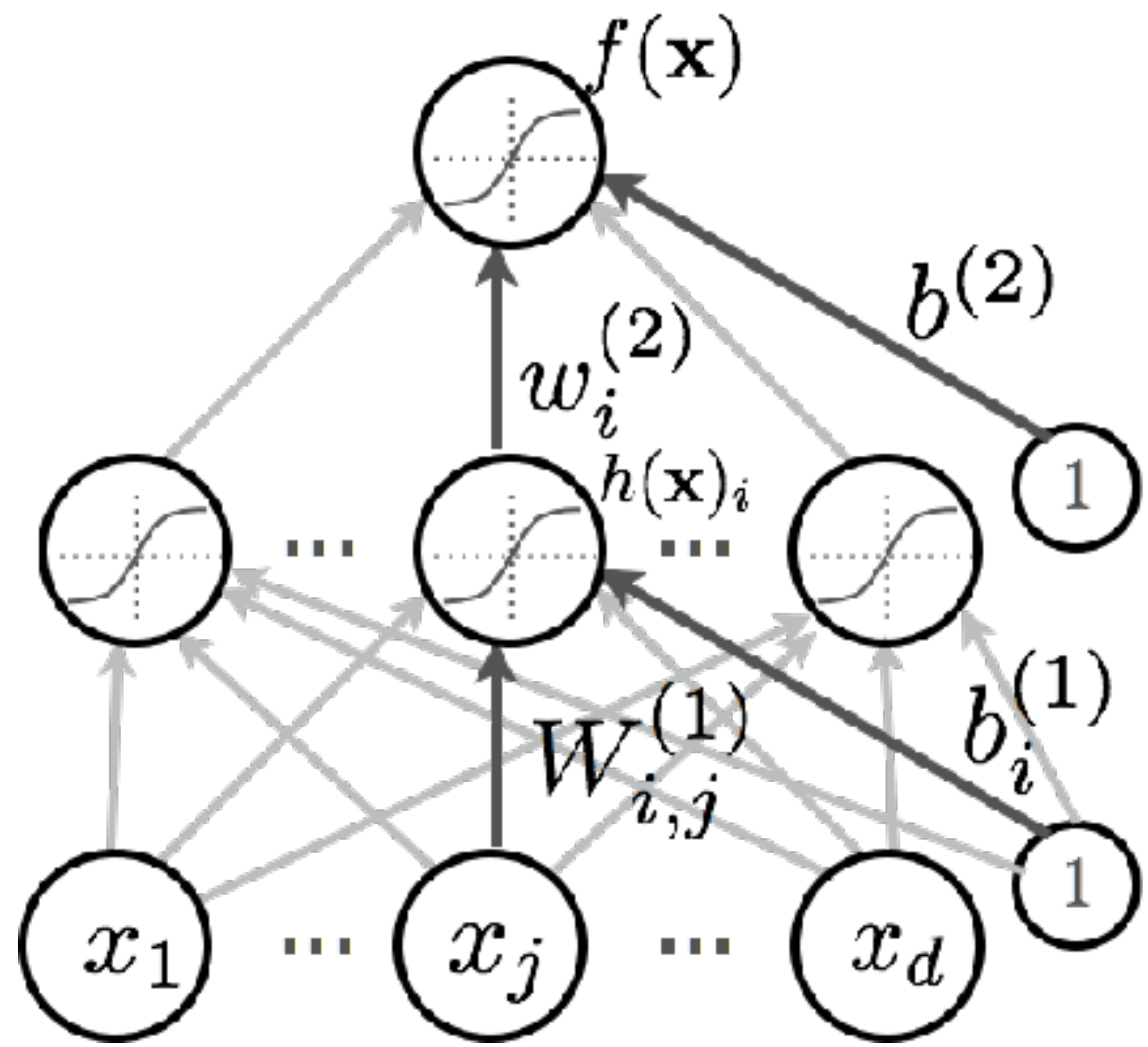


Single Hidden Layer Neural Net

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \textstyle\sum_j W_{i,j}^{(1)} x_j\right)$$



- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)^\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

Output activation function

$\mathbf{W}_{i,j}^{(k)}$:Weight from neuron i to neuron j in layer k

Multilayer Neural Net

- Consider a network with L hidden layers.
  - layer pre-activation for k>0

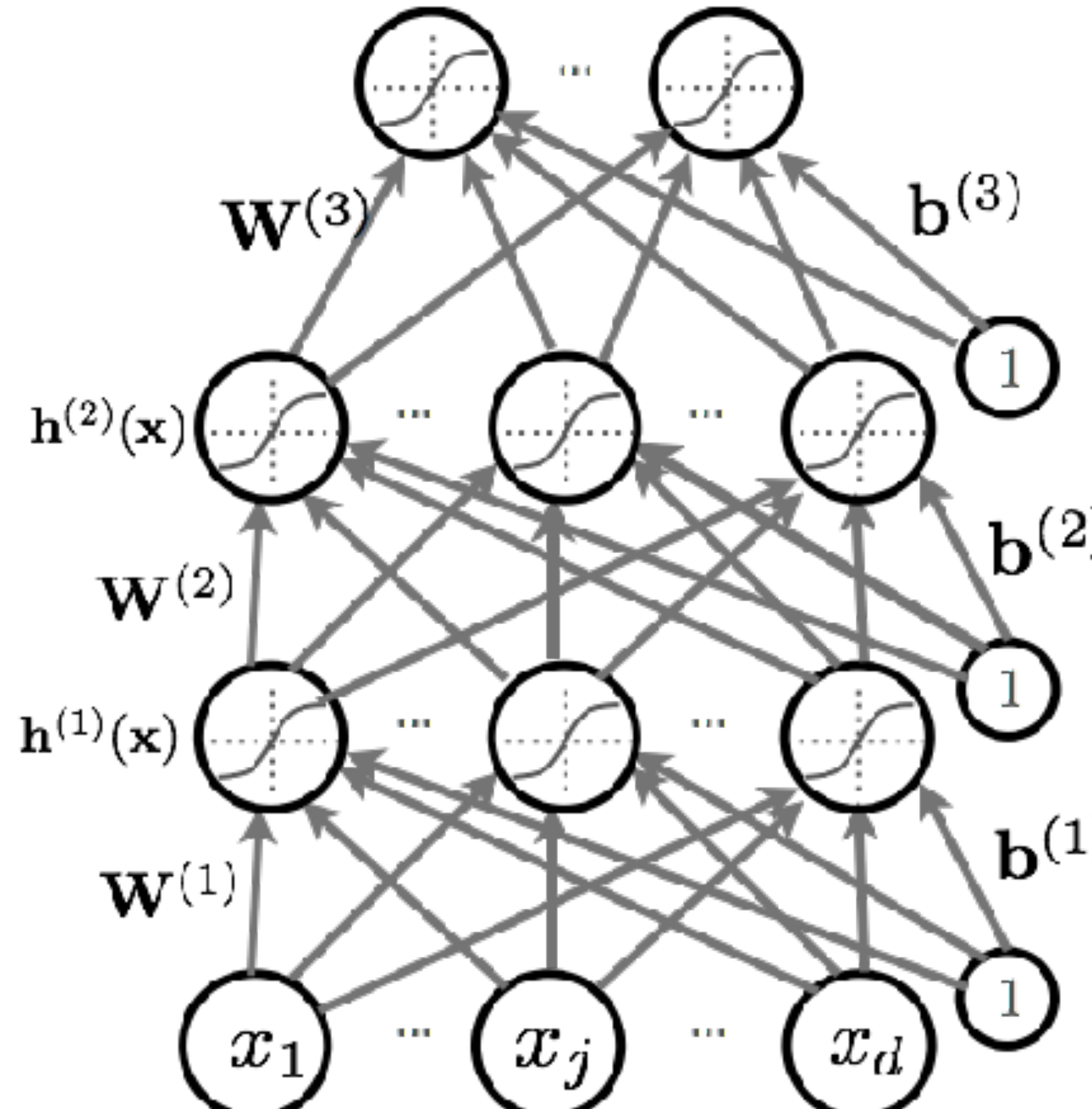
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation (k=L+1):

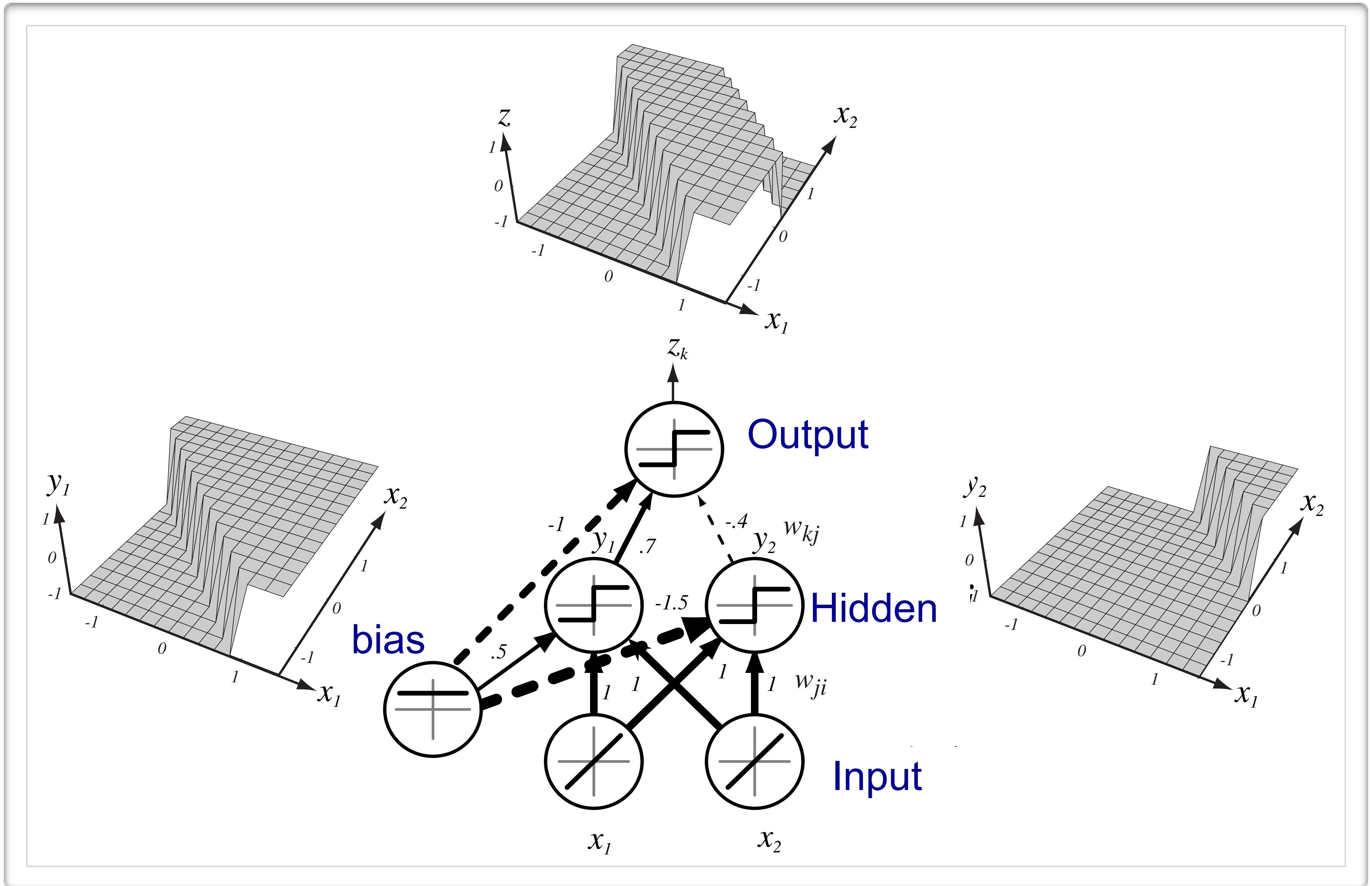
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

Capacity of Neural Nets

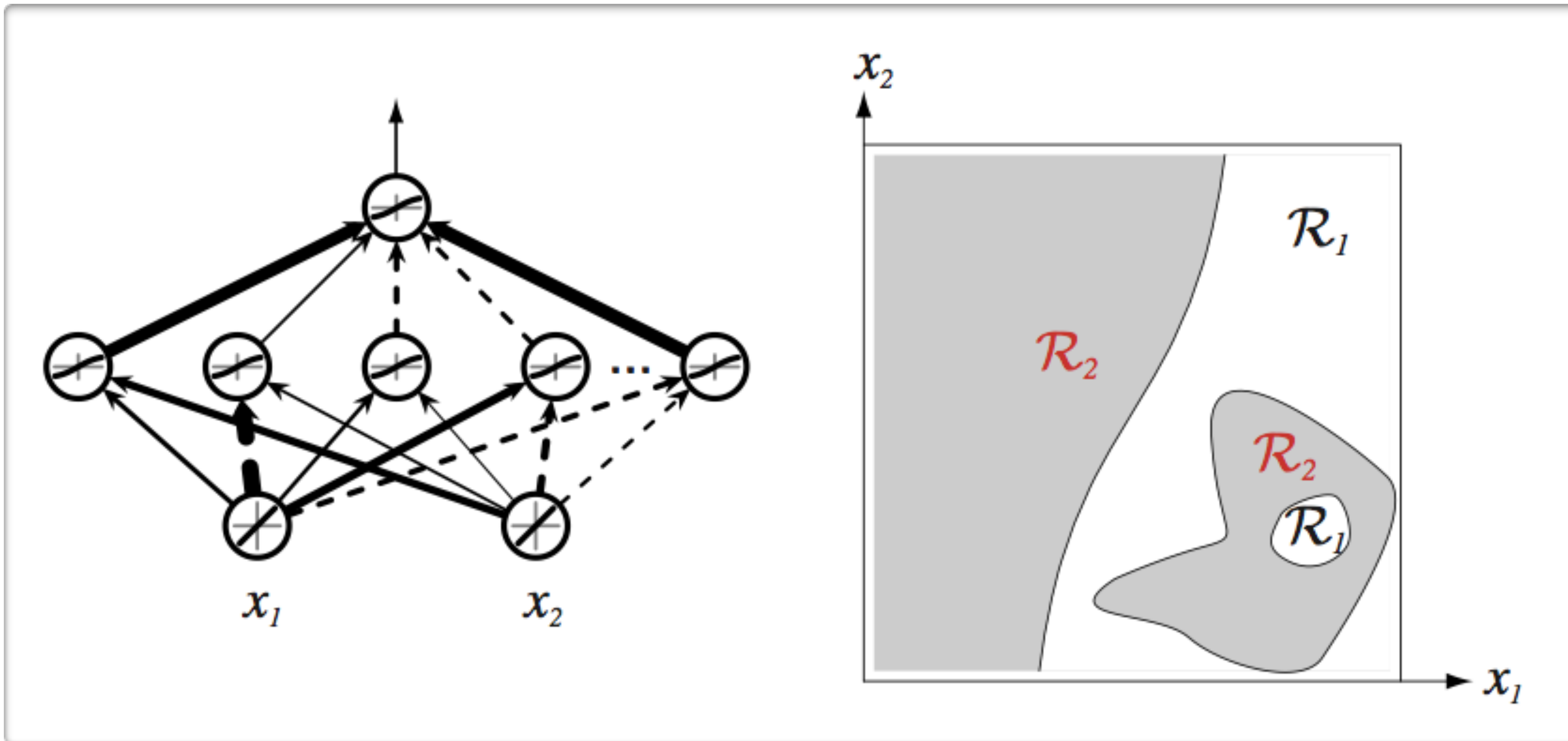
- Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

- Consider a single layer neural network



(from Pascal Vincent's slides)

Universal Approximation

- Universal Approximation Theorem (Hornik, 1991):
  - “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

- This applies for sigmoid, tanh and many other activation functions.

- However, this does not mean that there is learning algorithm that can find the necessary parameter values.

Training

- Empirical Risk Minimization:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \underbrace{\lambda \Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

- Learning is cast as optimization.

  ➢ For classification problems, we would like to minimize classification error.

  ➢ For regression problems, we would like to minimize regression error, e.g., L1 or L2 distance from ground-truth

Stochastic Gradient Descend

- Perform updates after seeing each example:
  - Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - For t=1:T
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$$

Training epoch
=
Iteration of all examples

- To train a neural net, we need:

  ➢ Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

  ➢ A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

  ➢ Regularizer and its gradient: $\Omega(\boldsymbol{\theta},)$ $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

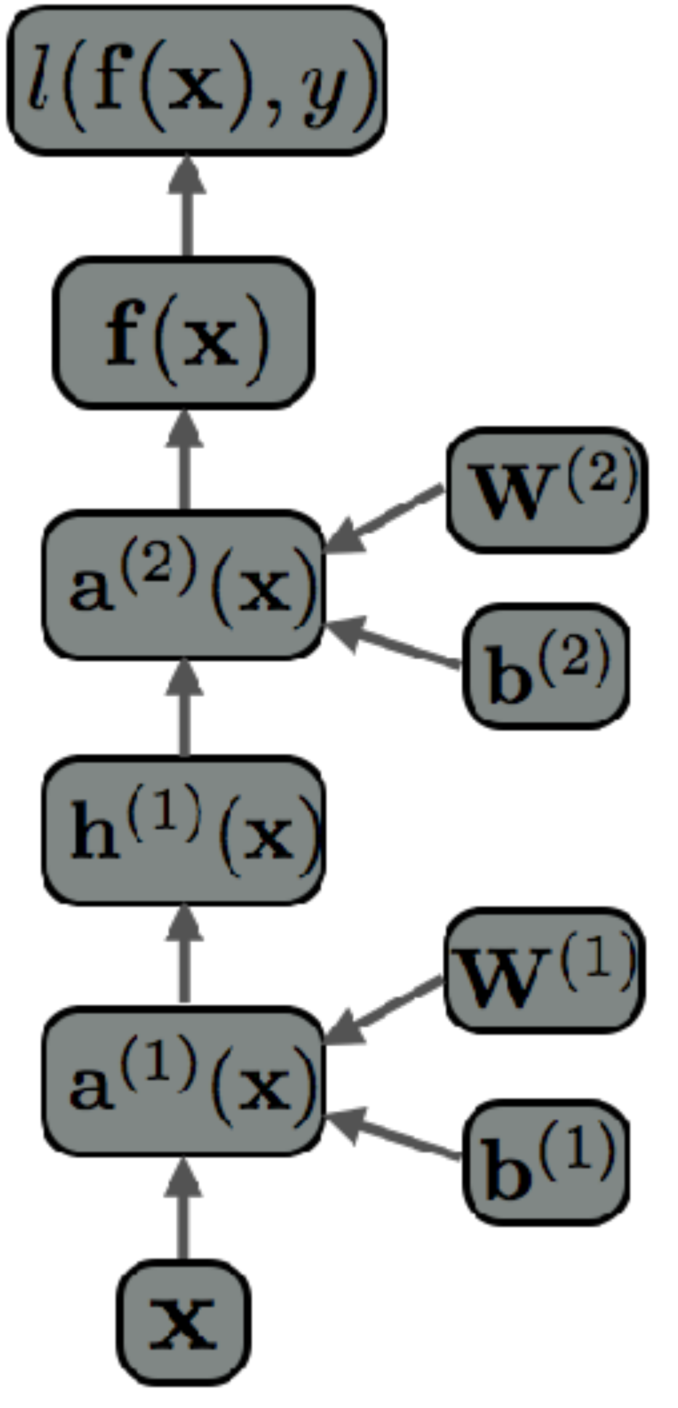
Computational Flow Graph

- Forward propagation can be represented as an acyclic flow graph

- Forward propagation can be implemented in a modular way:

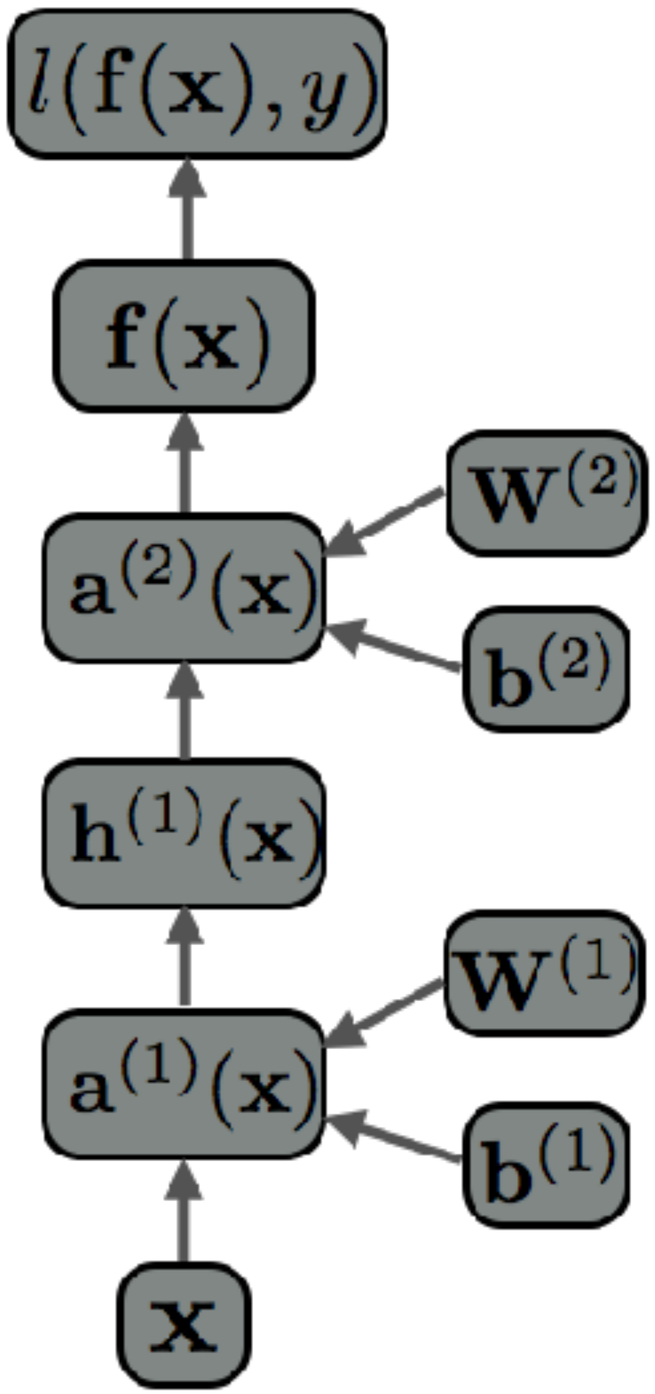
  ➢ Each box can be an object with an fprop method, that computes the value of the box given its children

  ➢ Calling the fprop method of each box in the right order yields forward propagation



Computational Flow Graph

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each child box.
- By calling bprop in the reverse order, we obtain backpropagation



Mini-batch, Momentum

Ideally uncorrelated!

- Make updates based on a mini-batch of examples (instead of a single example):
  - ➢ the gradient is the average regularized loss for that mini-batch
  - ➢ can give a more accurate estimate of the gradient

- Momentum: Can use an exponential average of previous gradients:

$$\overline{\nabla}_{\boldsymbol{\theta}}^{(t)} = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\boldsymbol{\theta}}^{(t-1)}$$

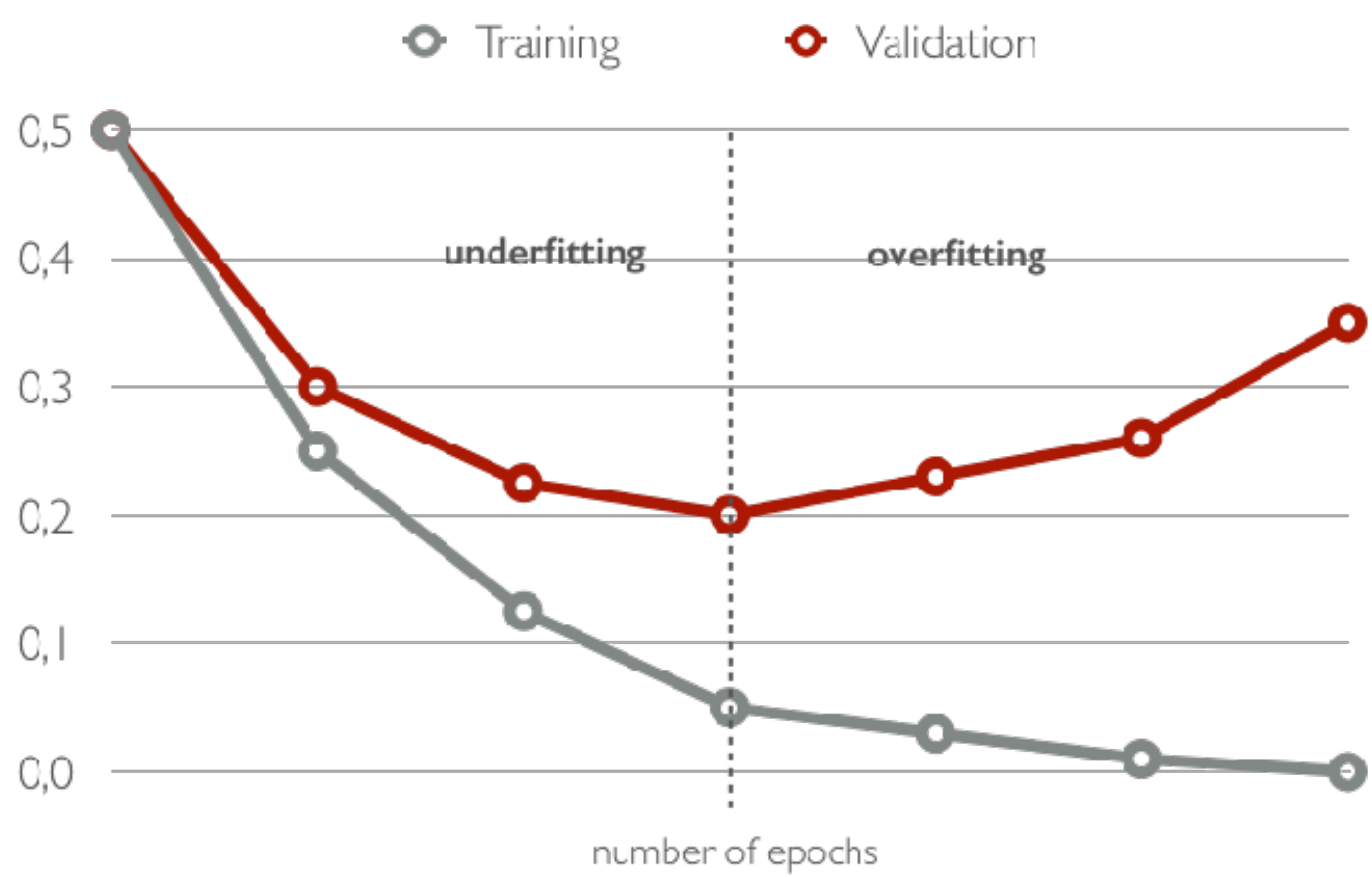
  - ➢ can get pass plateaus more quickly, by “gaining momentum”

Model Selection

- Training Protocol:
  - Train your model on the Training Set $\mathcal{D}^{\mathrm{train}}$
  - For model selection, use Validation Set $\mathcal{D}^{\mathrm{valid}}$
    - ➢ Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
  - Estimate generalization performance using the Test Set $\mathcal{D}^{\mathrm{test}}$
- Generalization is the behavior of the model on **unseen examples**.

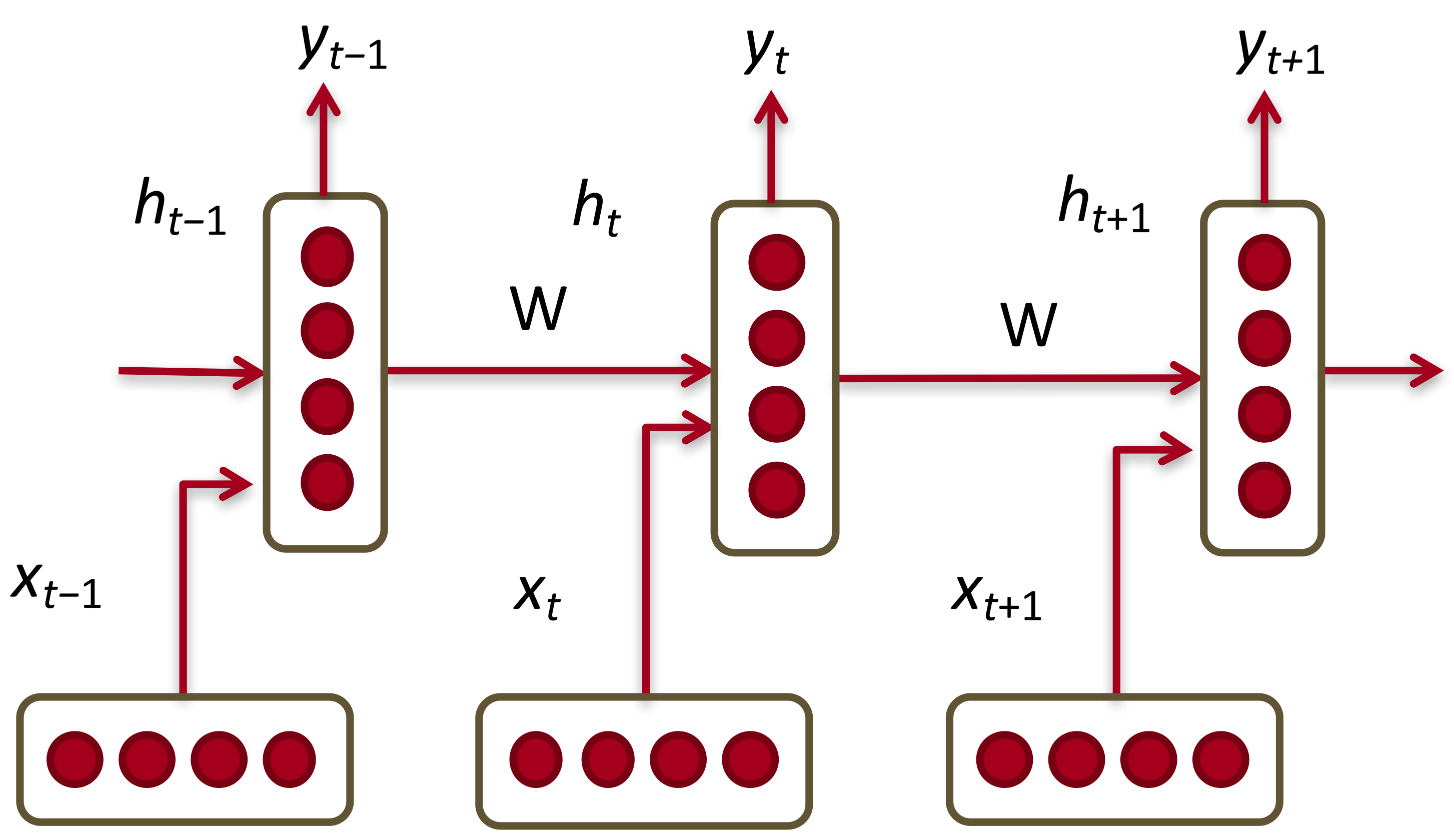
Early Stopping

- To select the number of epochs, stop training when validation set error increases (with some look ahead).



Recurrent Neural Networks

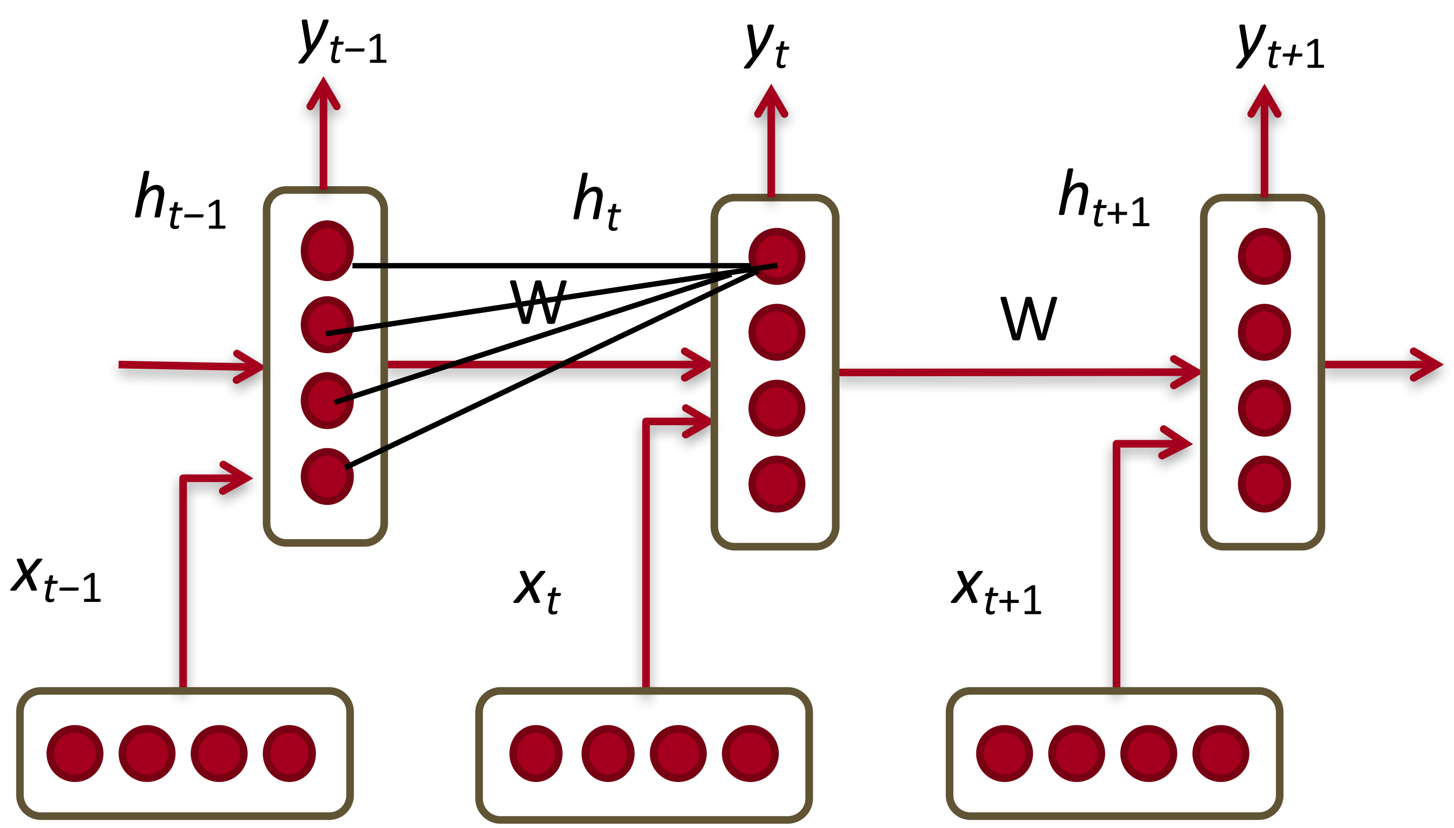
- RNNs tie the weights at each time step
- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled between inputs and outputs, as well as between output labels.
- In practice, limitations from SGD training, capacity, initialization etc.



Recurrent Neural Networks

- RNNs tie the weights at each time step

- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled between inputs and outputs, as well as between output labels.
- In practice, limitations from SGD training, capacity, initialization etc.

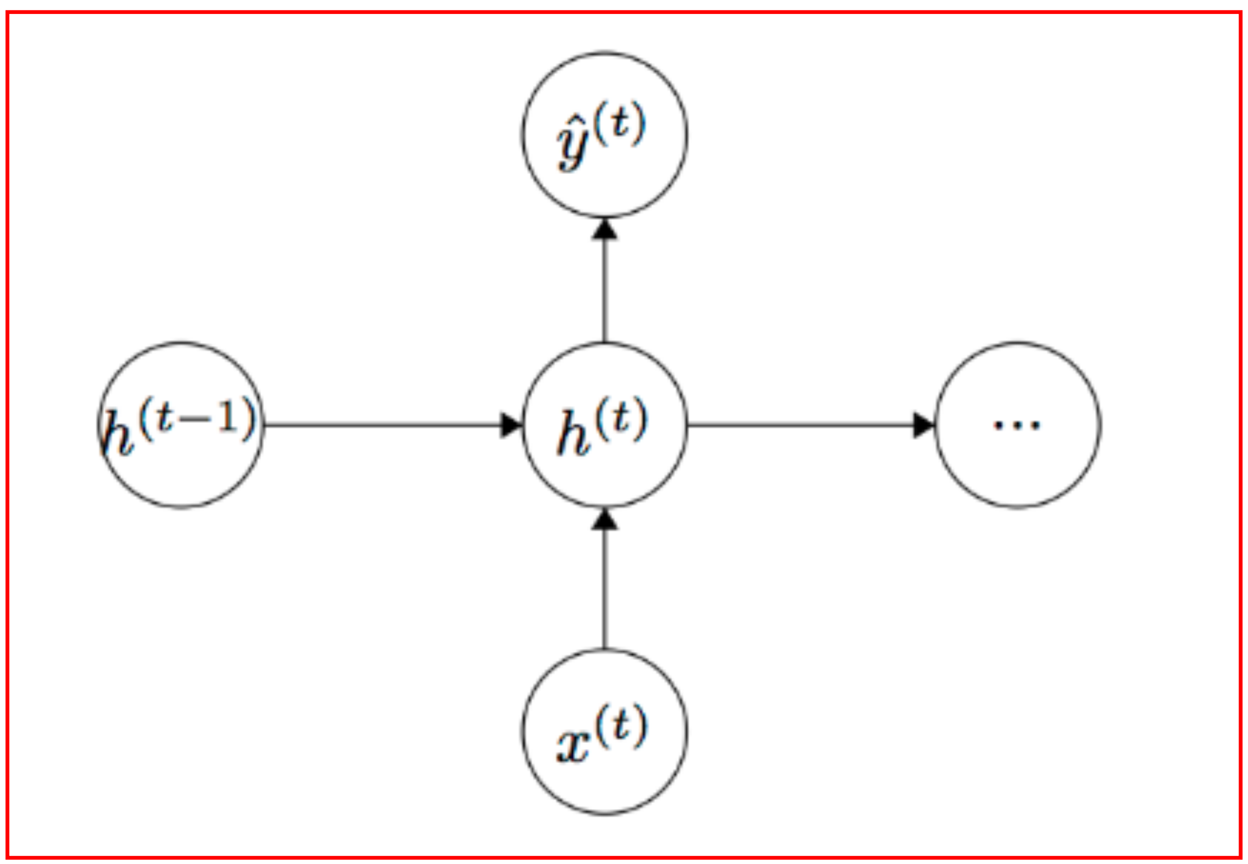


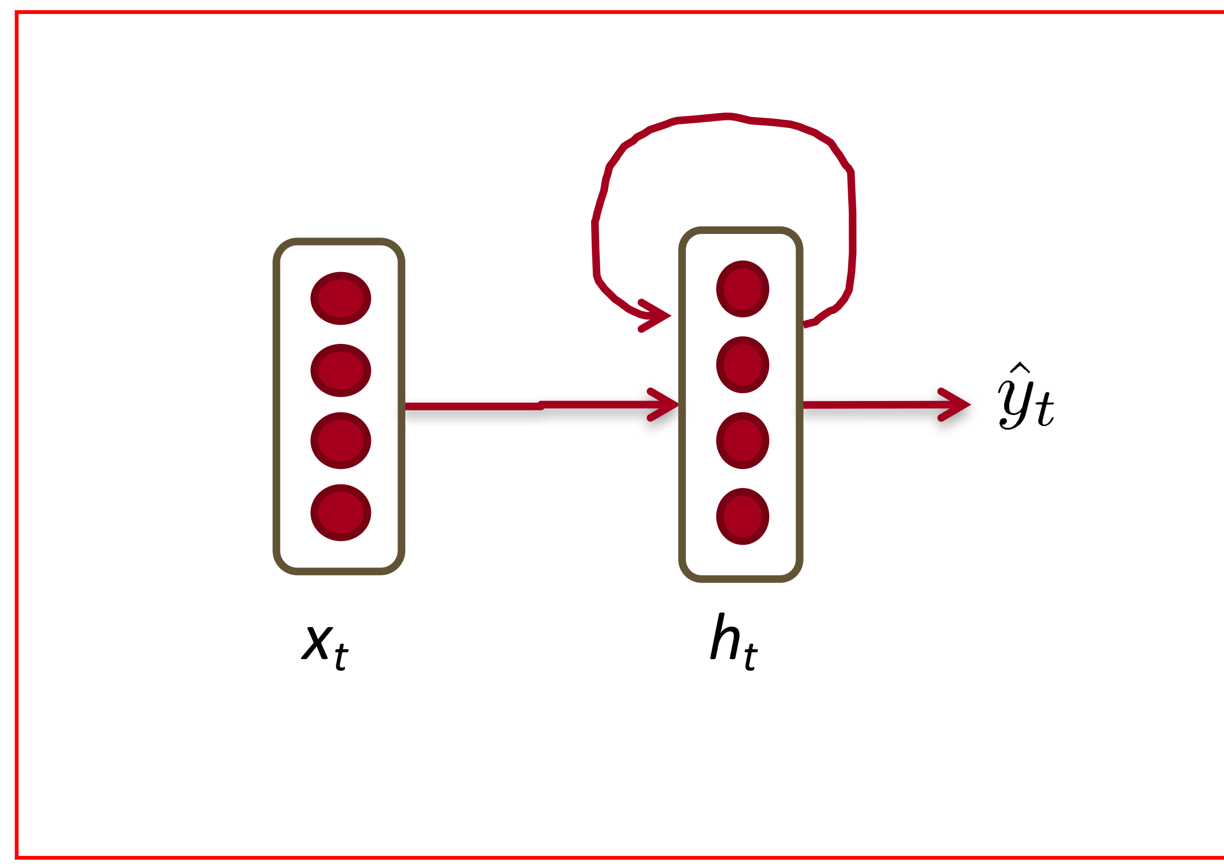
Recurrent Neural Network (single hidden layer)

- Given list of **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

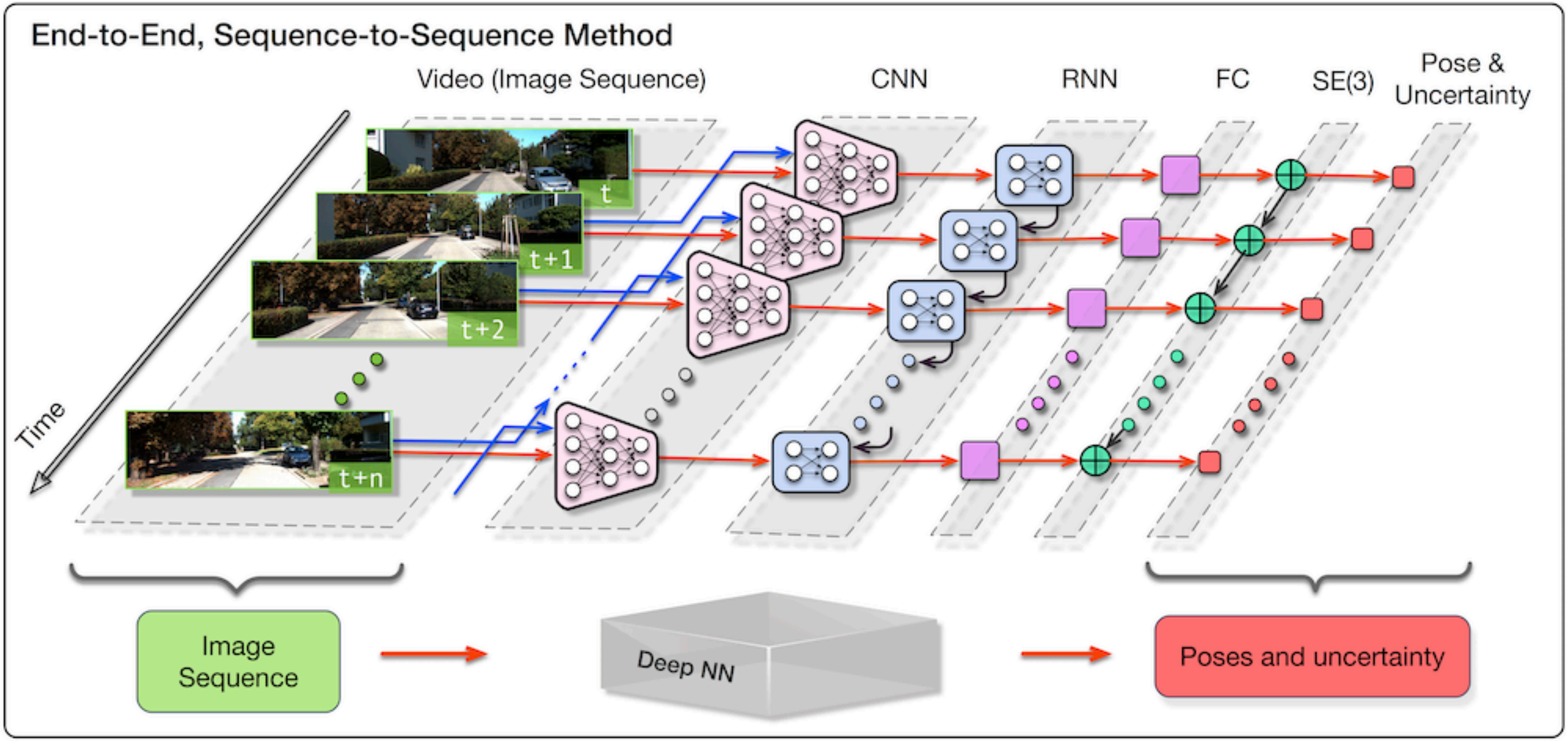
$$\hat{y}_t = \text{softmax}\left(W^{(S)} h_t\right)$$





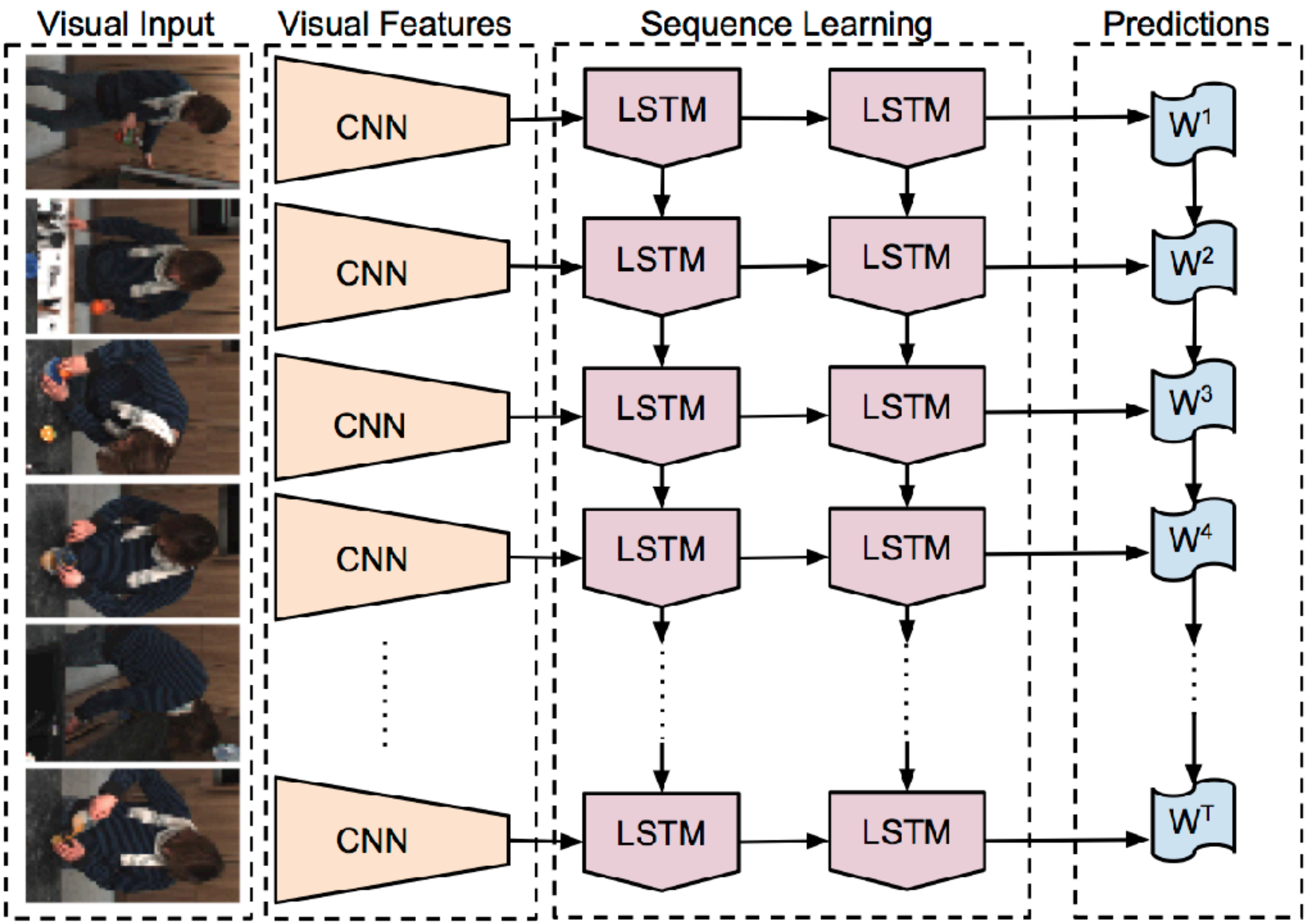
LSTMs and GRUs have more complex state update equations, learning when to explicitly replace or not update specific parts of the state h, but it is the same idea.

Recurrent Neural Networks for camera pose estimation



DeepVO : Towards Visual Odometry with Deep Learning

Recurrent Neural Networks for video captioning



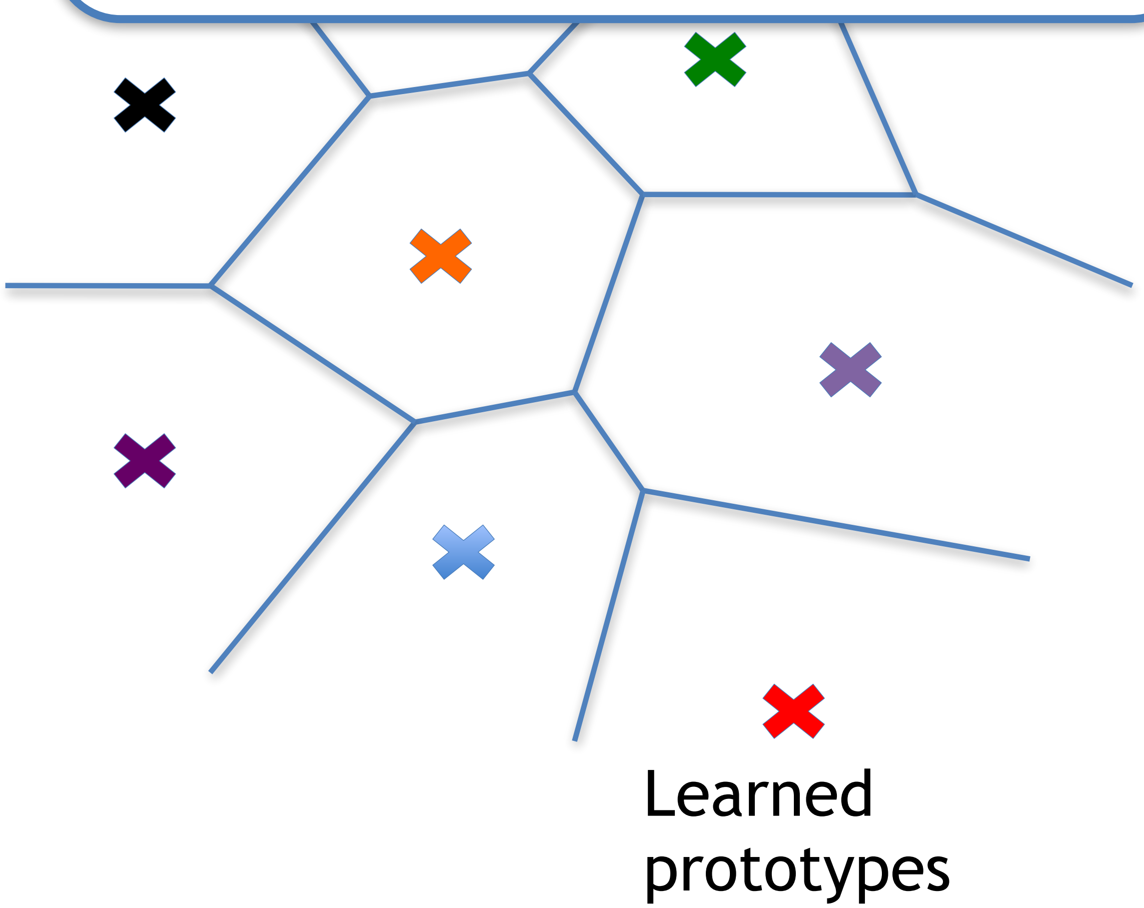
Learning Distributed Representations

- Deep learning is research on learning models with multilayer representations
  - ➢ multilayer (feed-forward) neural networks
  - ➢ multilayer graphical model (deep belief network, deep Boltzmann machine)
- Each layer learns "distributed representation"
  - ➢ Units in a layer are not mutually exclusive
    - each unit is a separate feature of the input
    - two units can be "active" at the same time
  - ➢ Units do not correspond to a partitioning (clustering) of the inputs
    - in clustering, an input can only belong to a single cluster

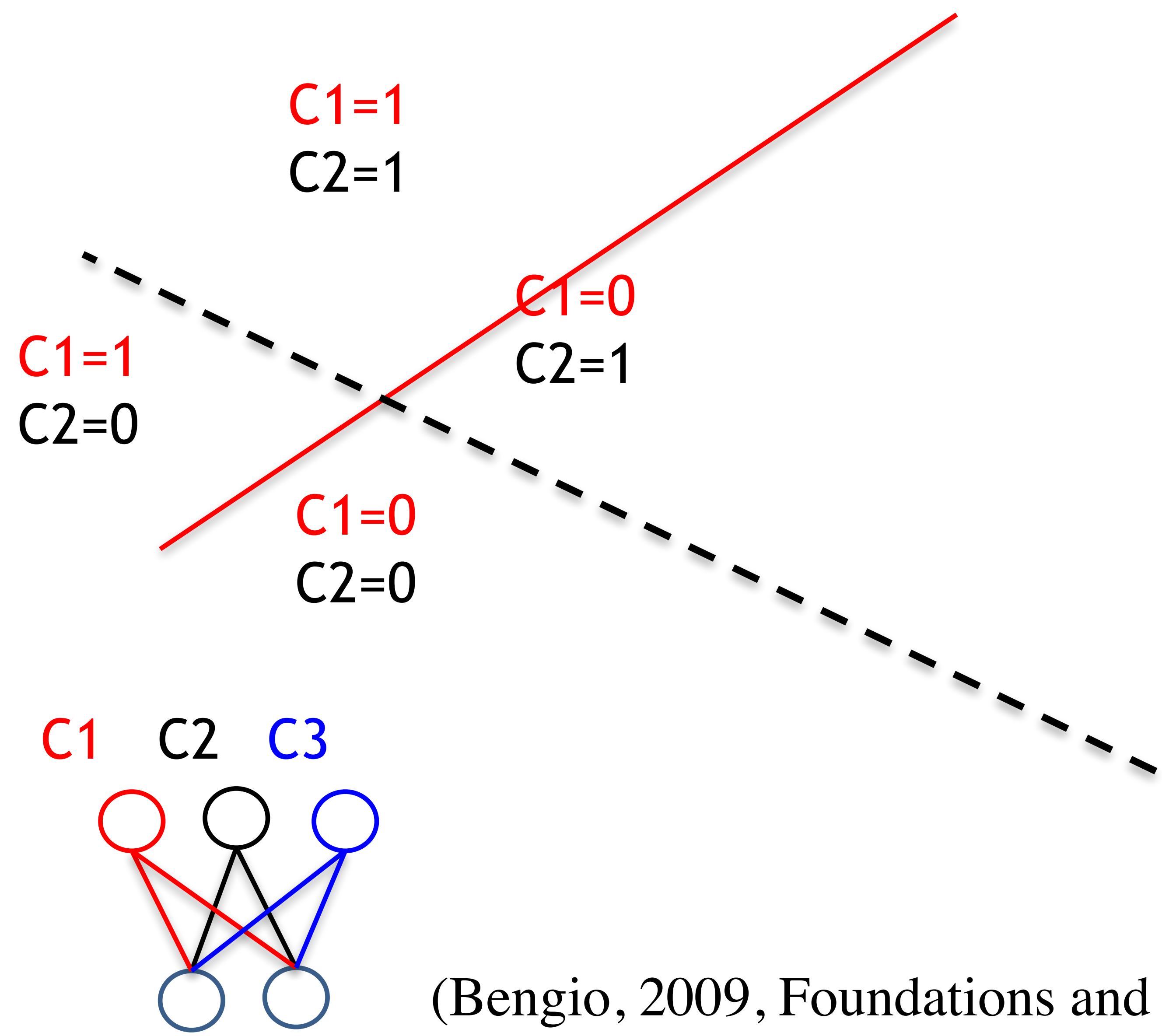
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators

- Parameters for each region.
- # of regions is linear with # of parameters.



- RBMs, Factor models, PCA, Sparse Coding, Deep models



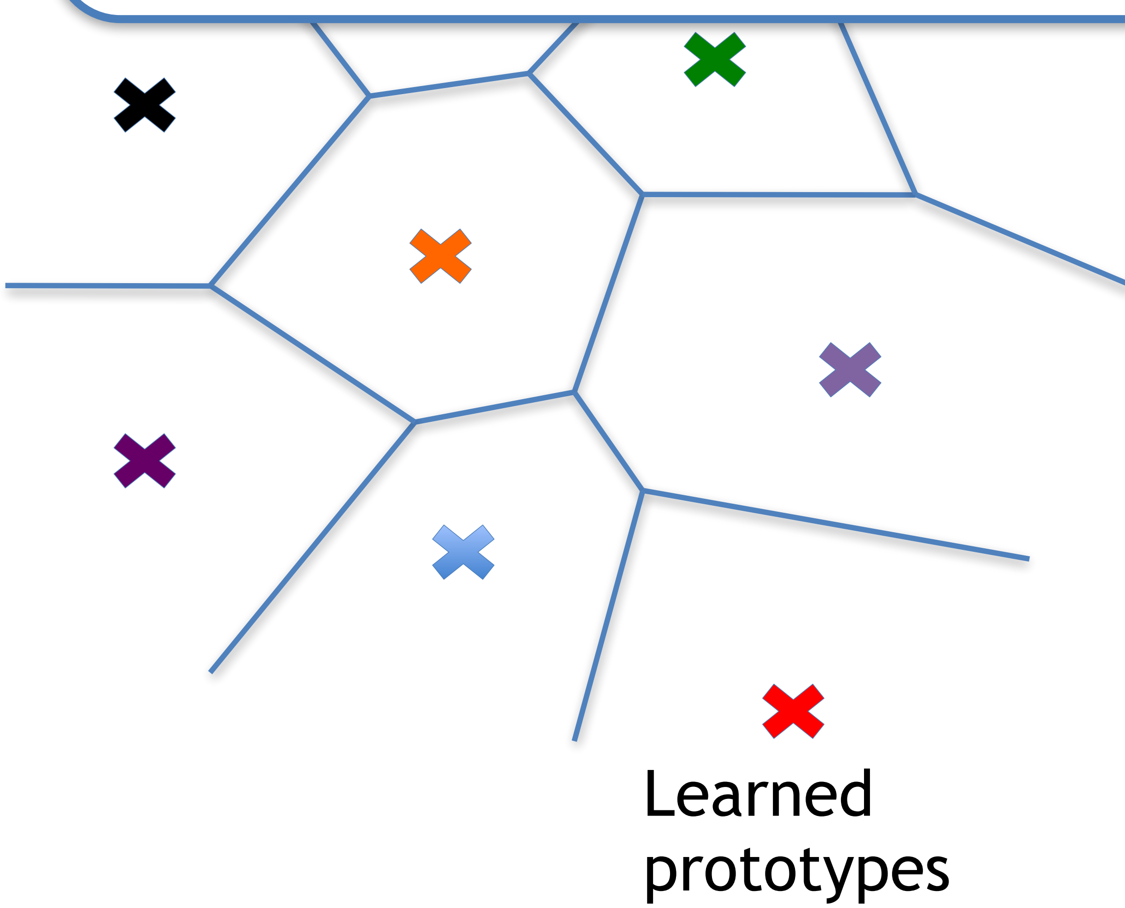
(Bengio, 2009, Foundations and Trends in Machine Learning)

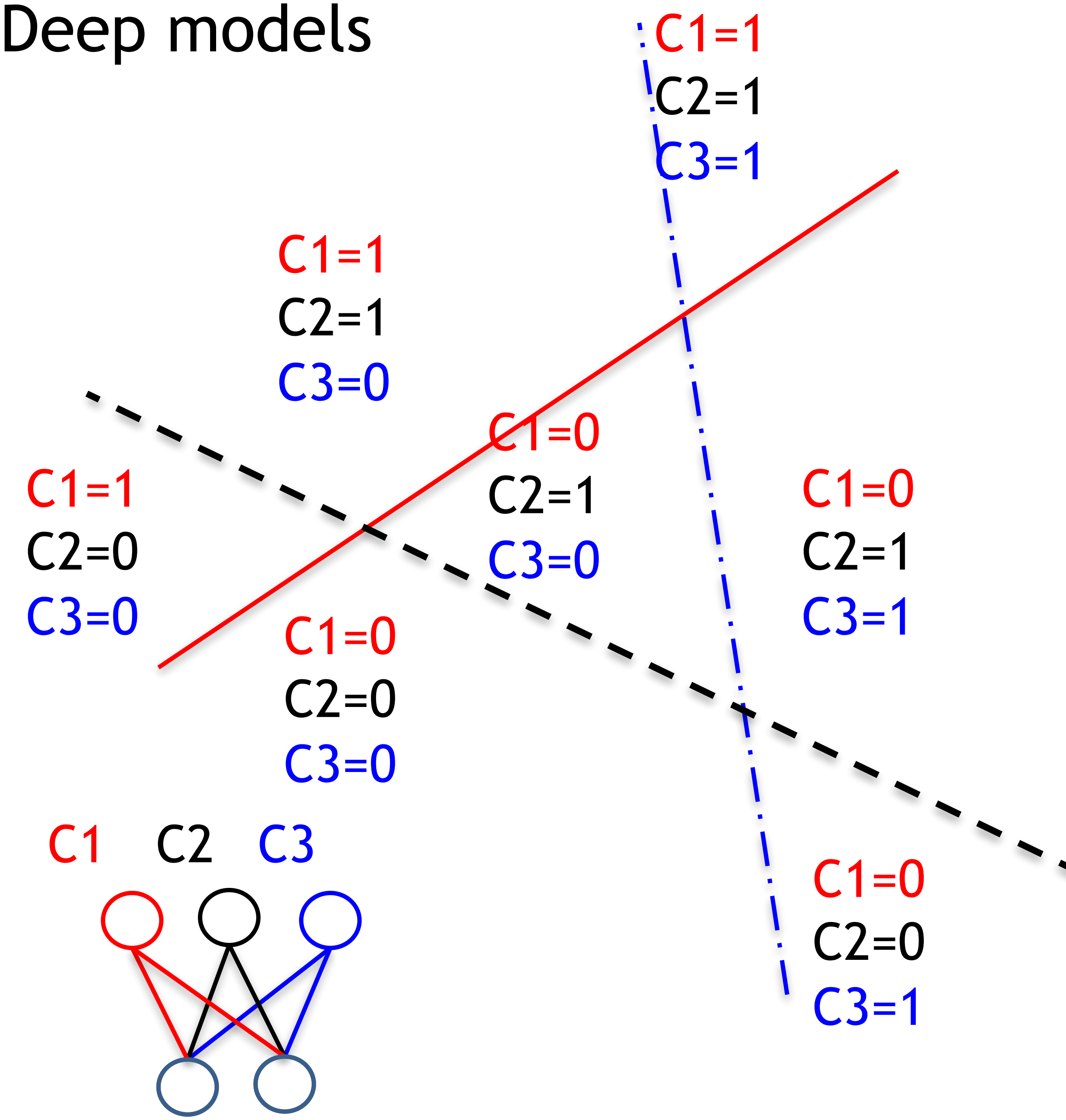
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators

- RBMs, Factor models, PCA, Sparse Coding, Deep models

- Parameters for each region.
- # of regions is linear with # of parameters.





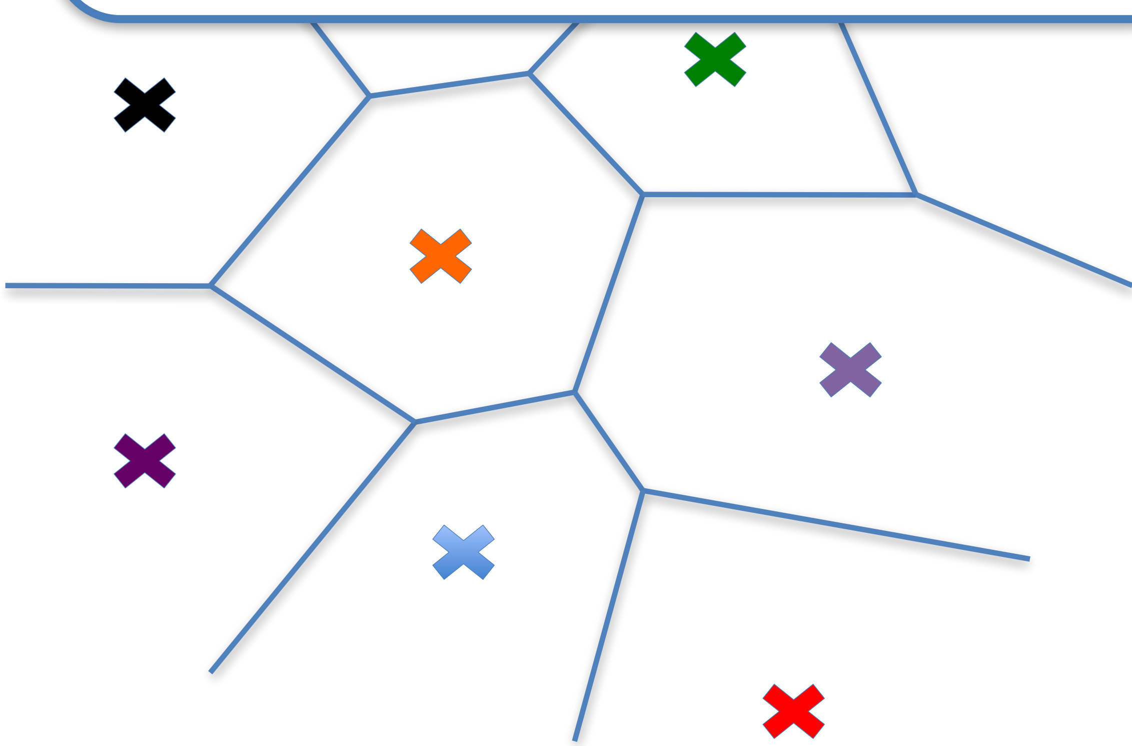
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators

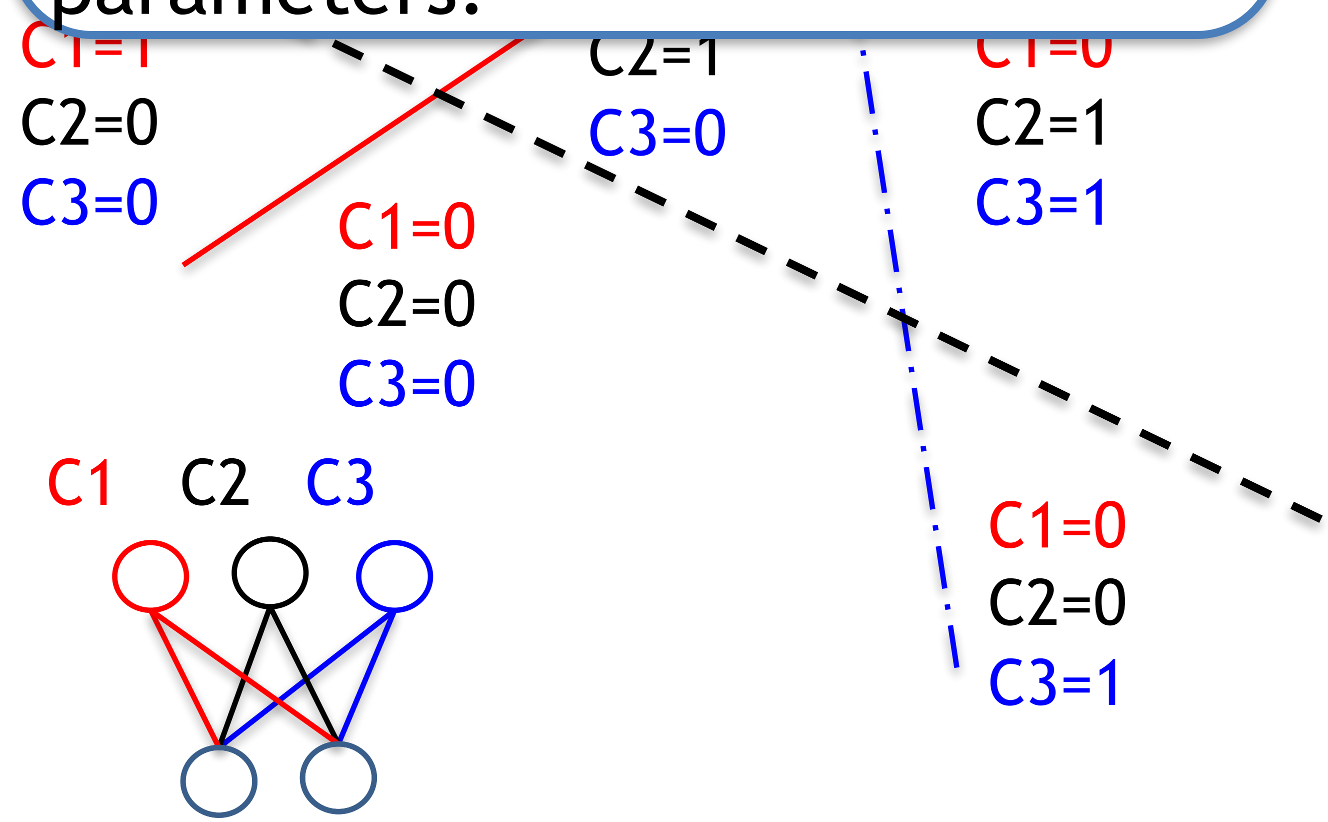
- RBMs, Factor models, PCA, Sparse Coding, Deep models

- Parameters for each region.
- # of regions is linear with # of parameters.

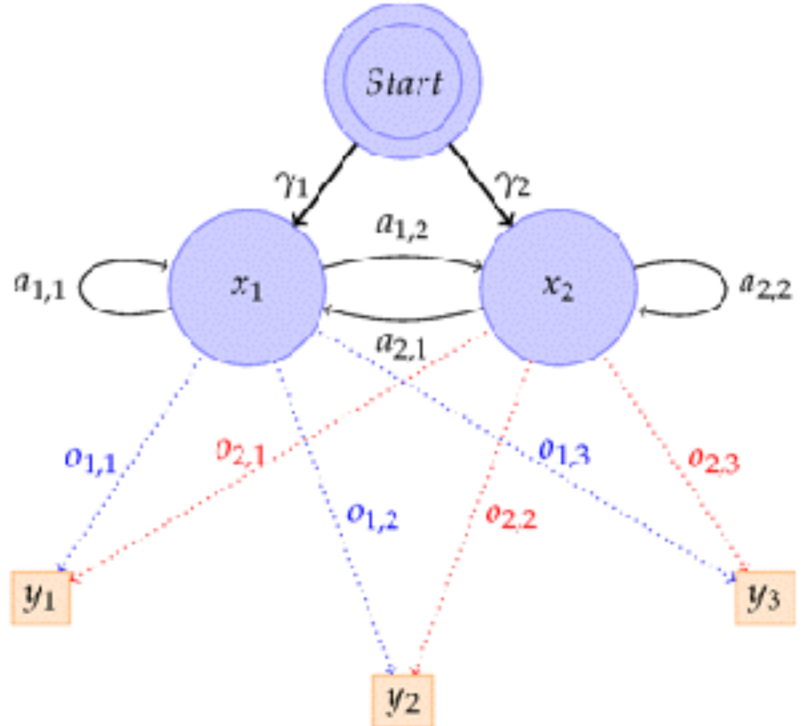
- Each parameter affects many regions, not just local.
- # of regions grows (roughly) exponentially in # of parameters.



Learned prototypes



RNNs VS HMMs



Initial Vector, Γ

x_1	x_2
γ_1	γ_2

Transition Matrix, A

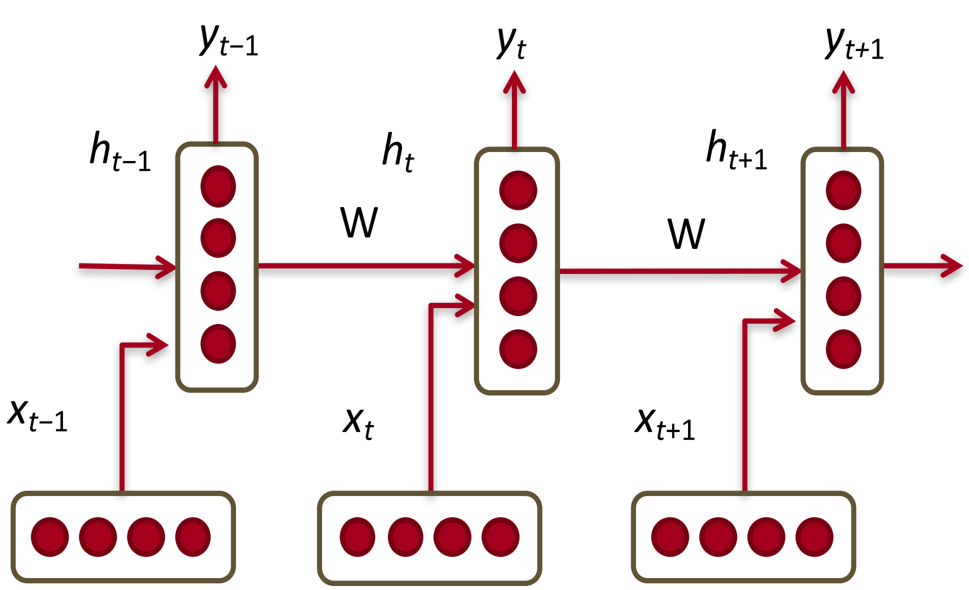
	x_1	x_2
x_1	$a_{1,1}$	$a_{1,2}$
x_2	$a_{2,1}$	$a_{2,2}$

Observation Matrix, O

	y_1	y_2	y_3
x_1	$o_{1,1}$	$o_{1,2}$	$o_{1,3}$
x_2	$o_{2,1}$	$o_{2,2}$	$o_{2,3}$

# of parameters scales quadratically w.r.t. the number of states

But! Number of world states encoded (in ideal case of binary activations and optimization attains all those combinations) **in the RNN is 2^N while it is N for the HMM**
The power of distributed representations



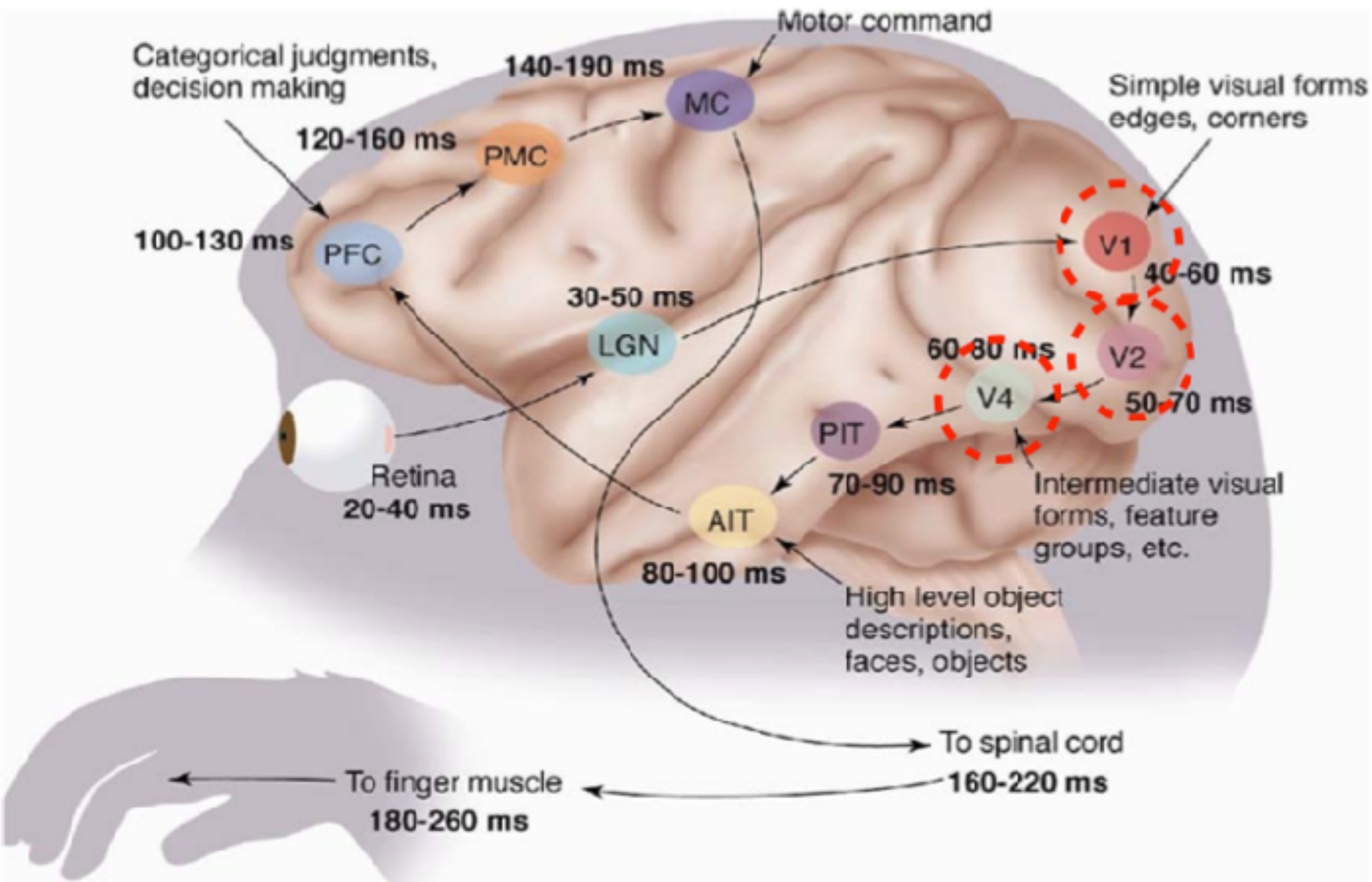
# of parameters scales quadratically w.r.t. the number of neurons in the hidden state

Computer Vision

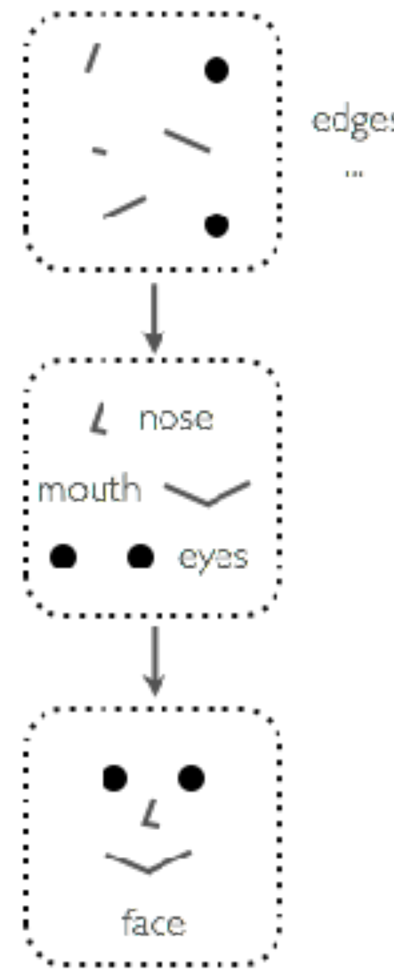
- Design algorithms that can process visual data to accomplish a given task:

  ➢ For example, object recognition: Given an input image, identify which object it contains

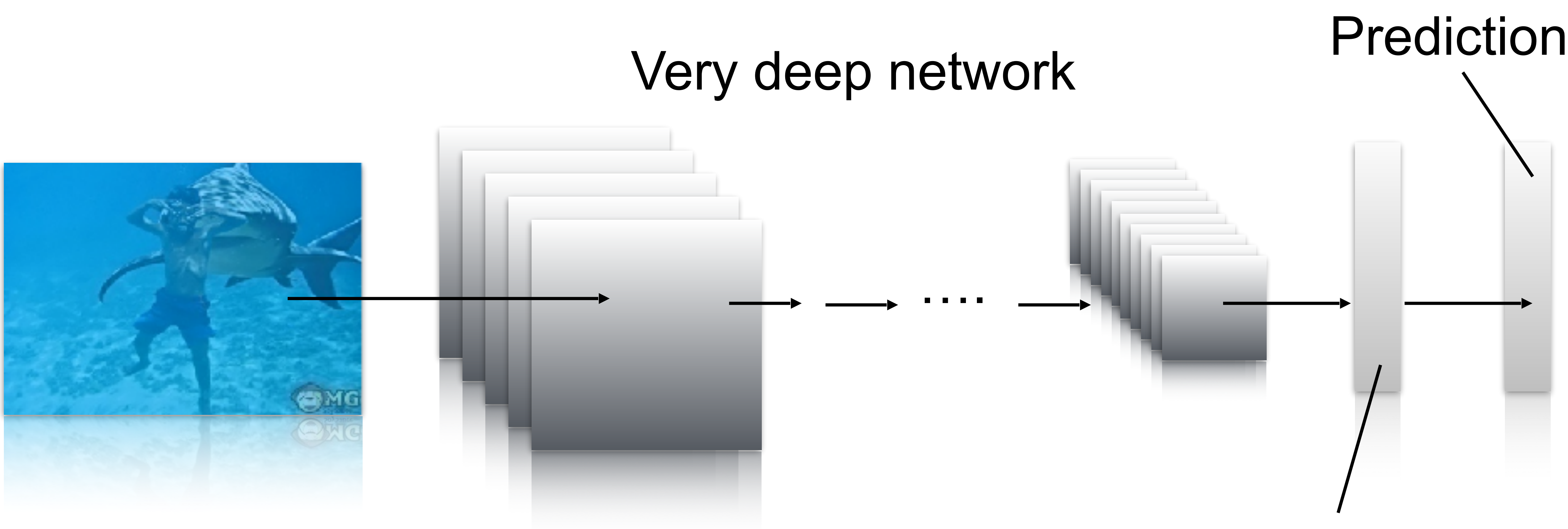
Inspiration from Visual Cortex



[picture from Simon Thorpe]



Deep Convolutional Nets



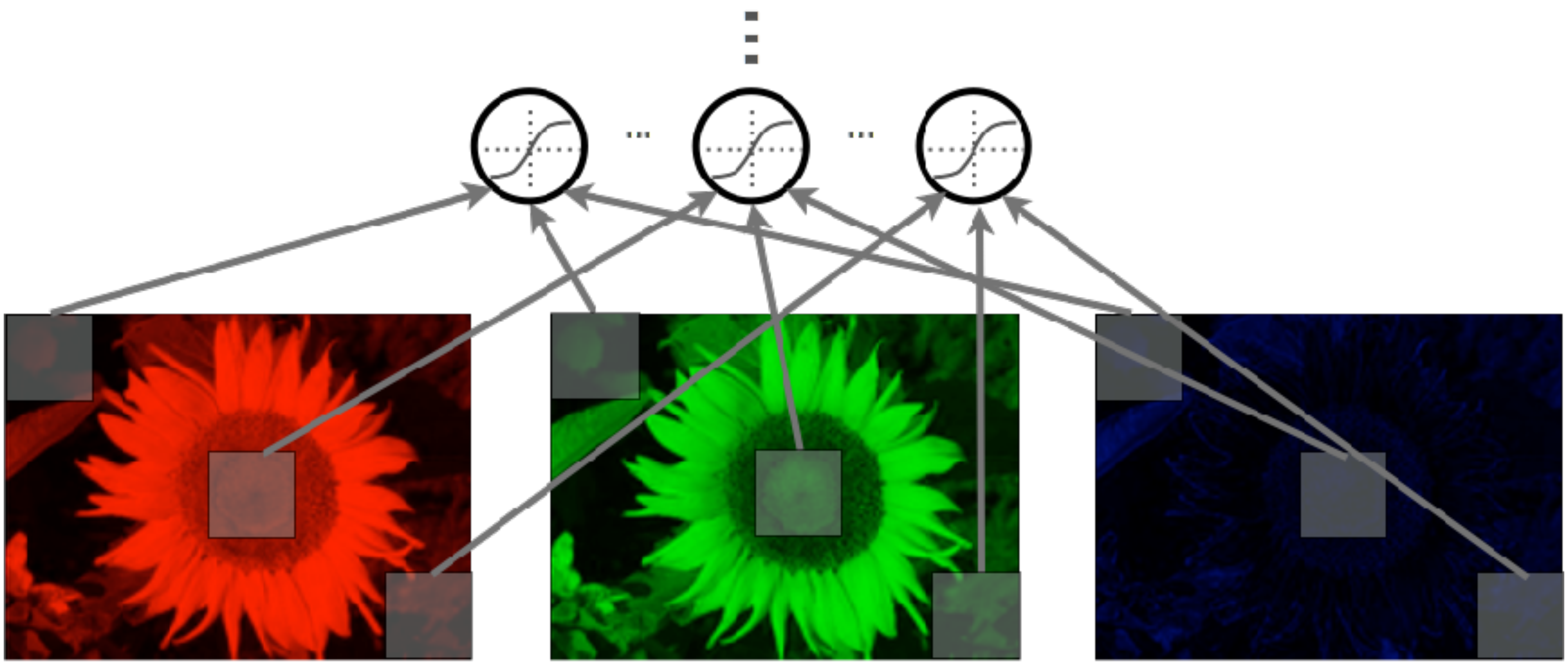
- Convolution
- Pooling
- Normalization
- Densely connected

Computer Vision

- Convolutional networks leverage these ideas
    - ➢ $\boxed{\text{Local connectivity}}$
    - ➢ Parameter sharing
    - ➢ Convolution
    - ➢ Pooling / subsampling hidden units

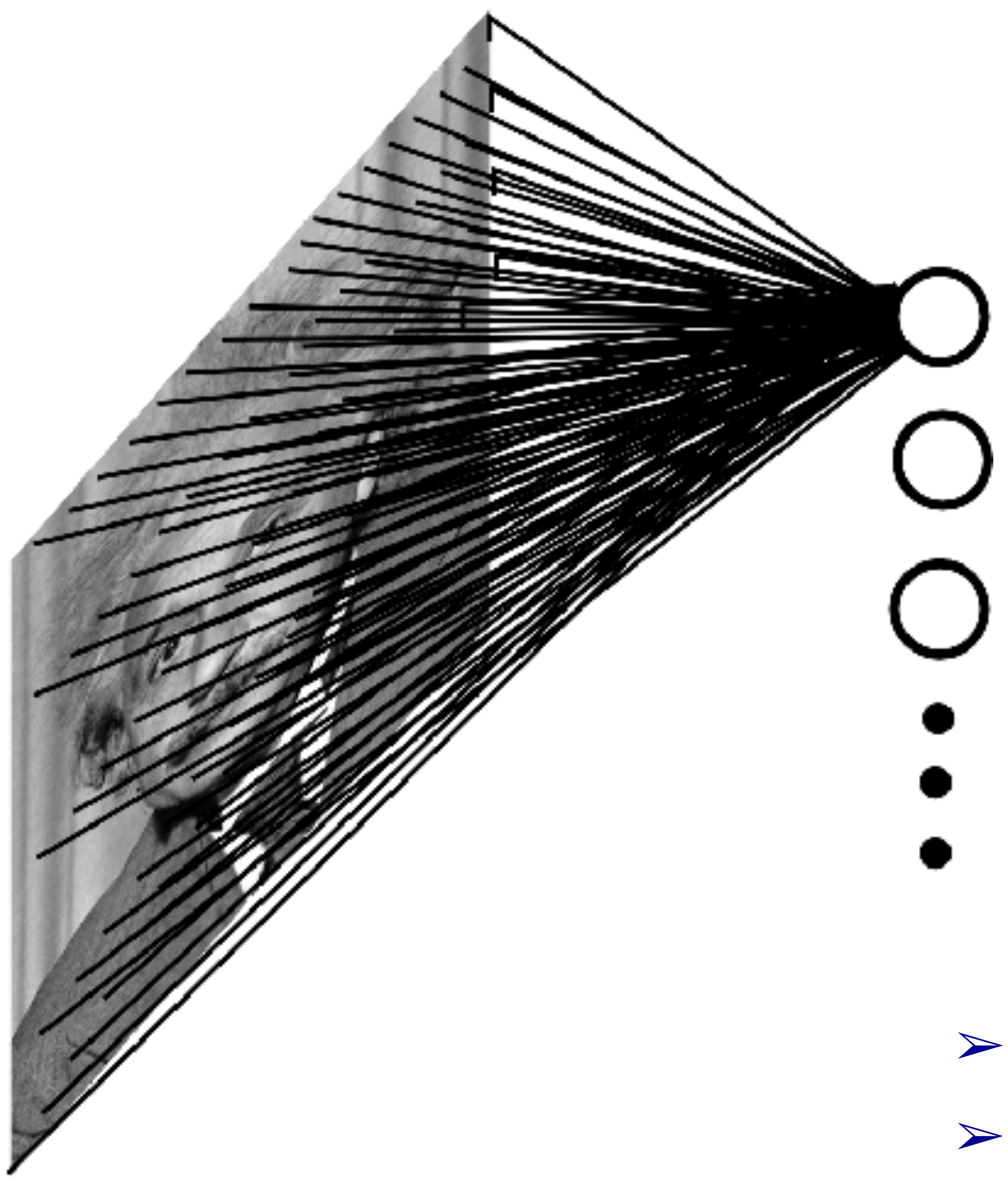
Local Connectivity

- Units are connected to all channels:
  - ➢ 1 channel if grayscale image,
  - ➢ 3 channels (R, G, B) if color image



Local Connectivity

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

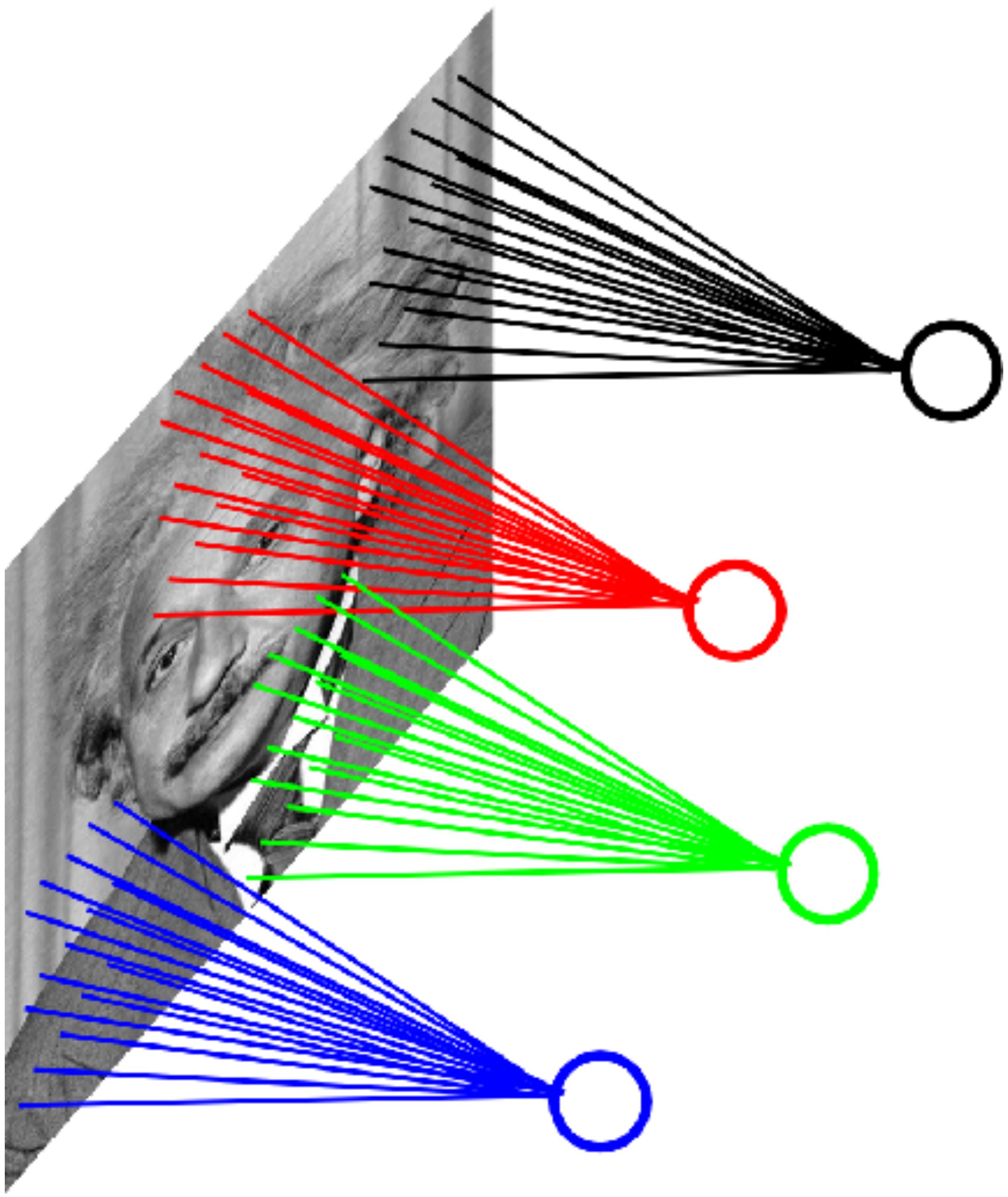


➢ Spatial correlation is local

➢ Too many parameters, will require a lot of training data!

Local Connectivity

- Example: 200x200 image, 40K hidden units, filter size 10x10, 4M parameters!



➢ This parameterization is good when input image is registered

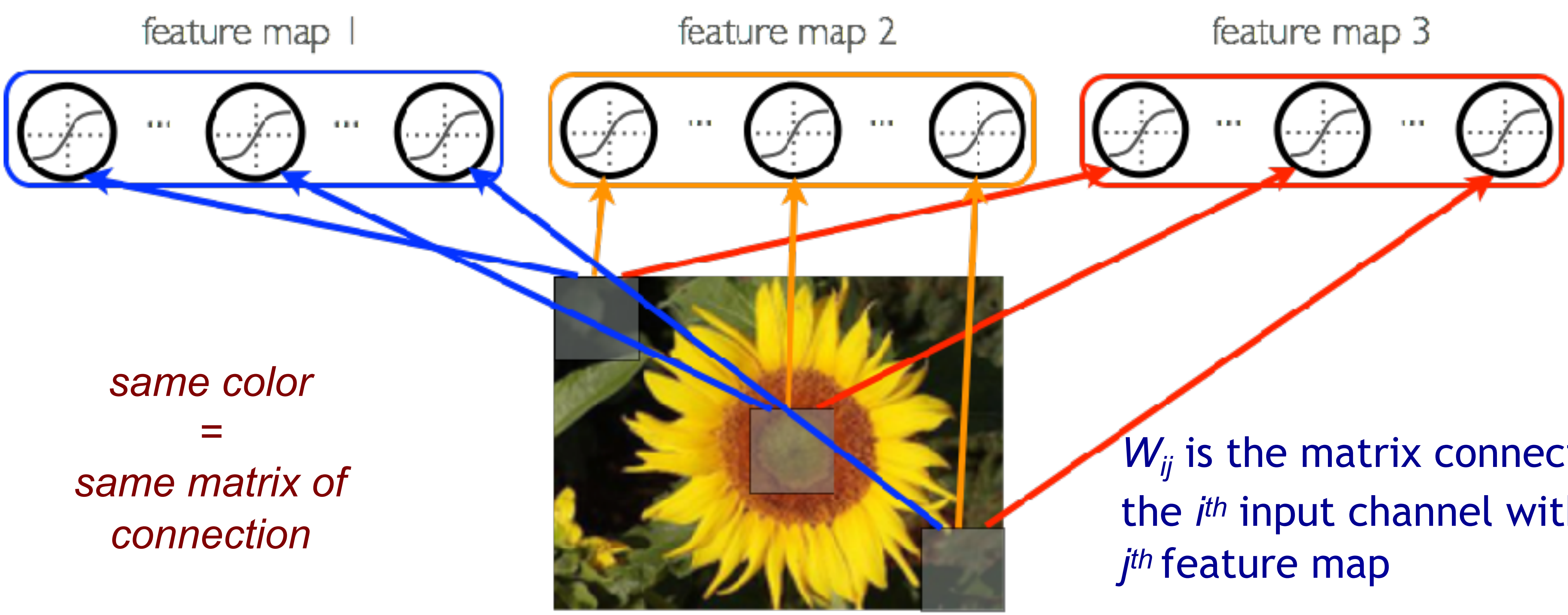
Computer Vision

- Convolutional networks leverage these ideas

  - ➢ Local connectivity
  - ➢ Parameter sharing
  - ➢ Convolution
  - ➢ Pooling / subsampling hidden units

Parameter Sharing

- Share matrix of parameters across some units
  - ➢ Units that are organized into the ‘feature map” share parameters
  - ➢ Hidden units within a feature map cover different positions in the image

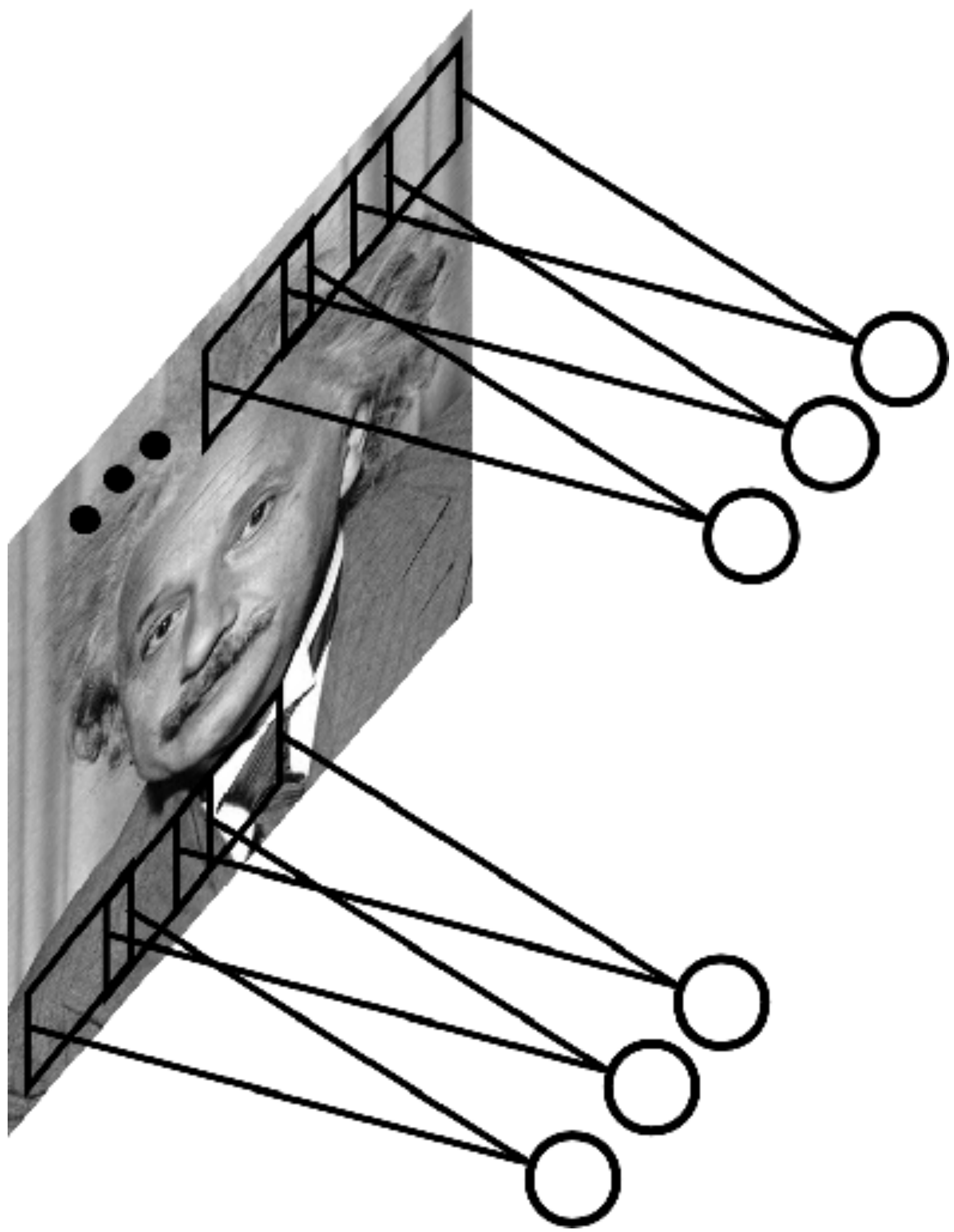


same color
=
same matrix of connection

W_{ij} is the matrix connecting the i^{th} input channel with the j^{th} feature map

Parameter Sharing

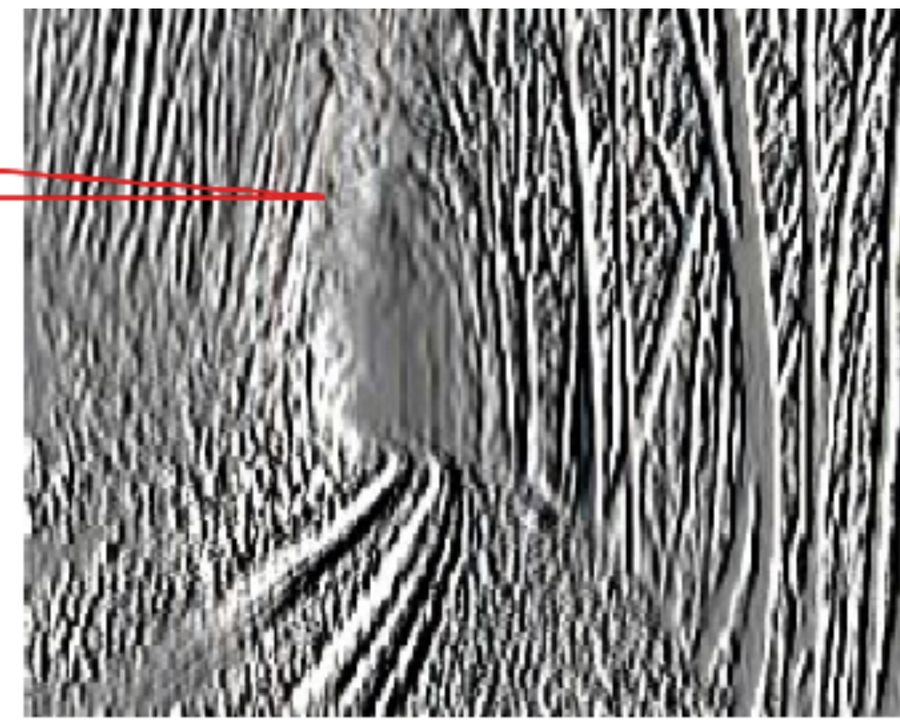
- Share matrix of parameters across certain units



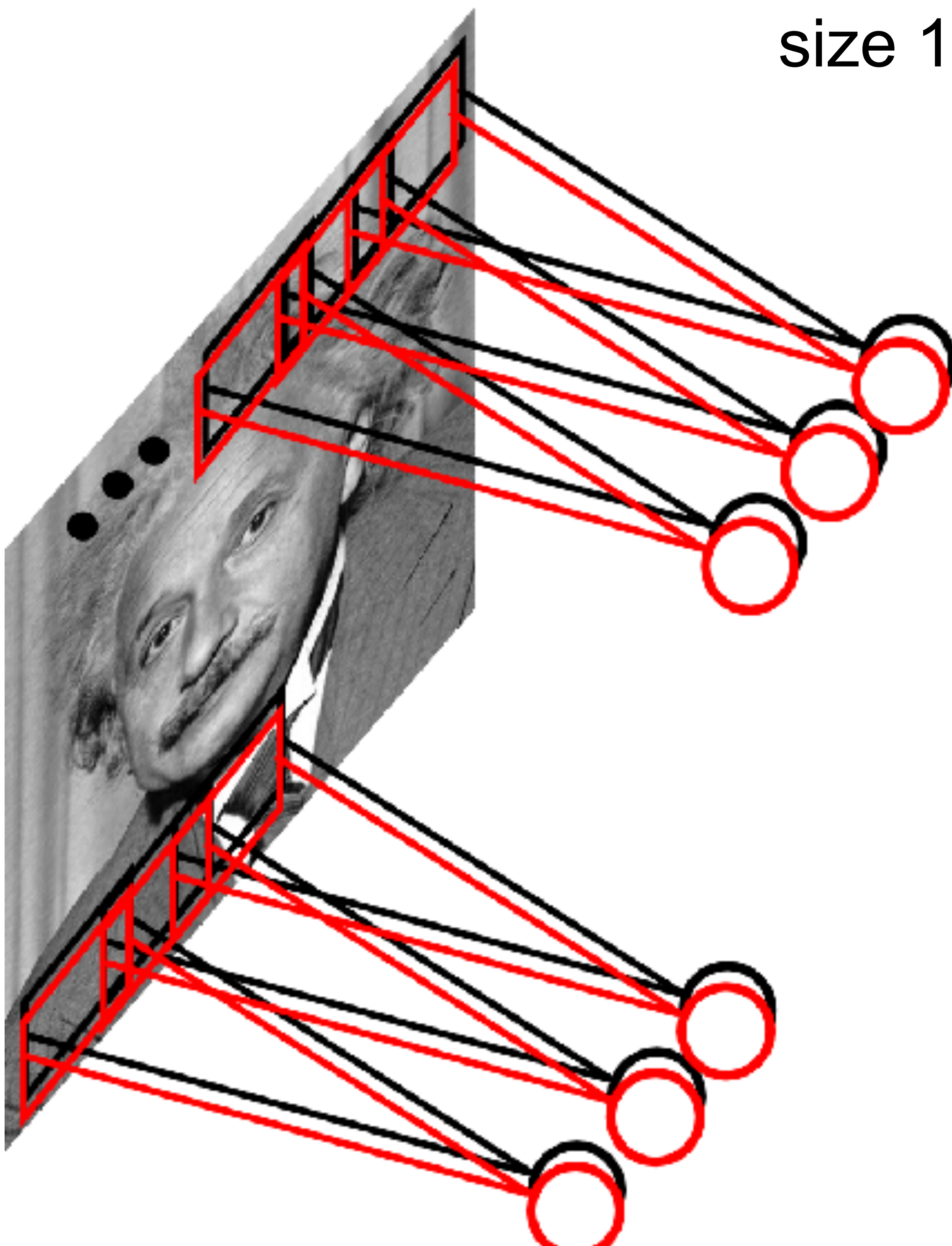
➢ Convolutions with certain kernels

Example

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



Multiple Feature Maps



- Example: 200x200 image, 100 filters, filter size 10x10, 10K parameters

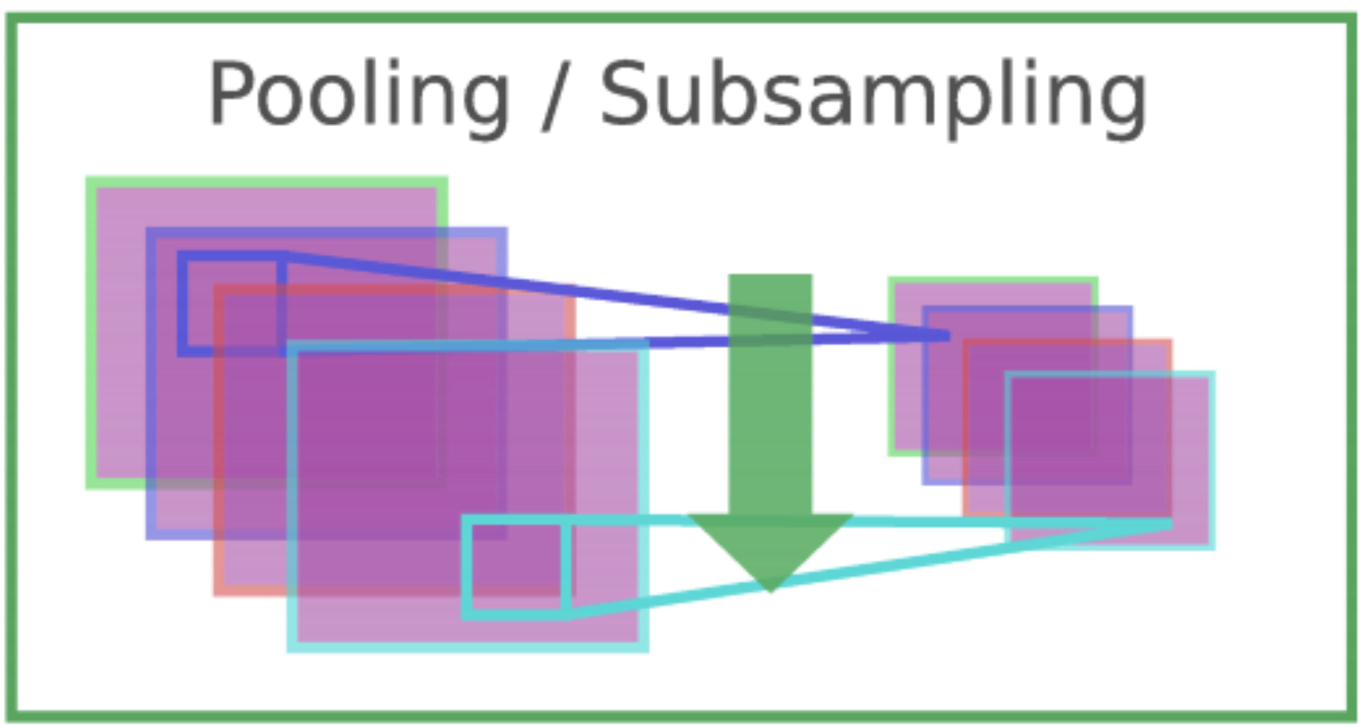
Computer Vision

- Convolutional networks leverage these ideas
    - ➢ Local connectivity
    - ➢ Parameter sharing
    - ➢ Convolution
    - ➢ Pooling / subsampling hidden units

Pooling

- Pool hidden units in same neighborhood
  - ➢ pooling is performed in non-overlapping neighborhoods (subsampling)

$$y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$$

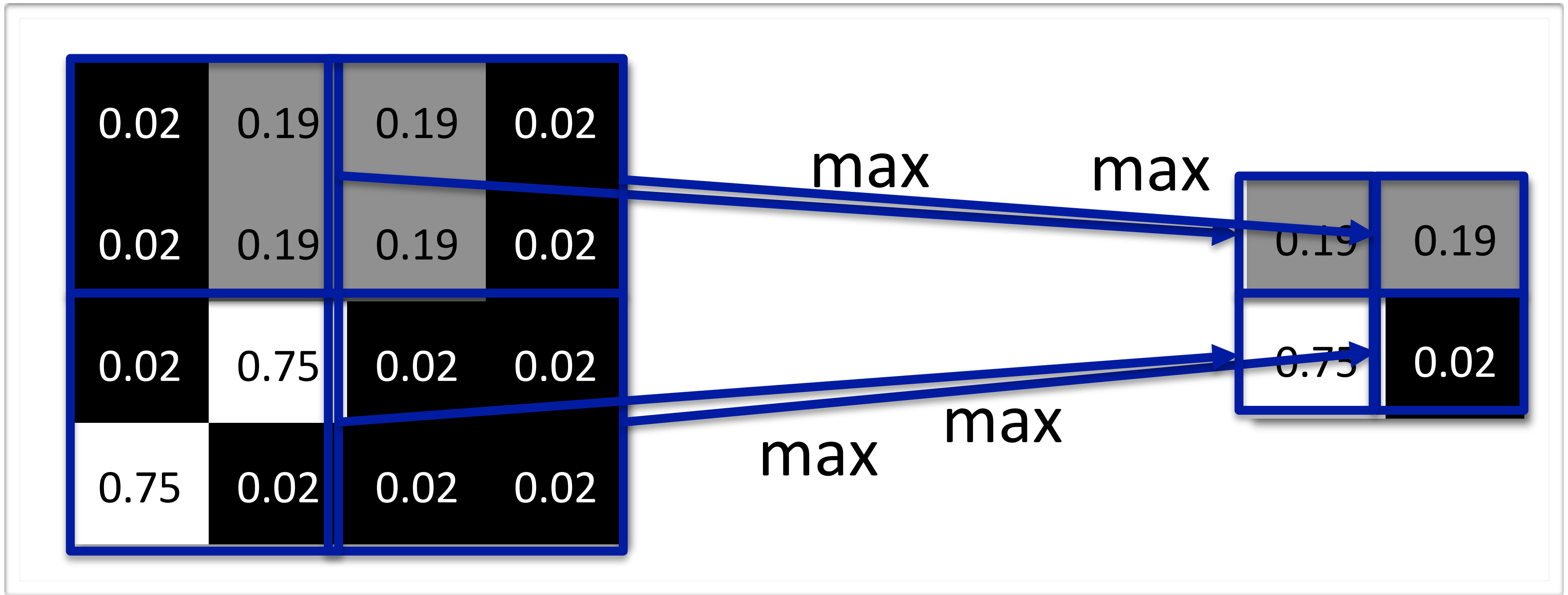


Jarret et al. 2009

- x_i is the i^{th} channel of input
- $x_{i,j,k}$ is value of the i^{th} feature map at position j,k
- p is vertical index in local neighborhood
- q is horizontal index in local neighborhood
- y_{ijk} is pooled / subsampled layer

Example: Pooling

- Illustration of pooling/subsampling operation

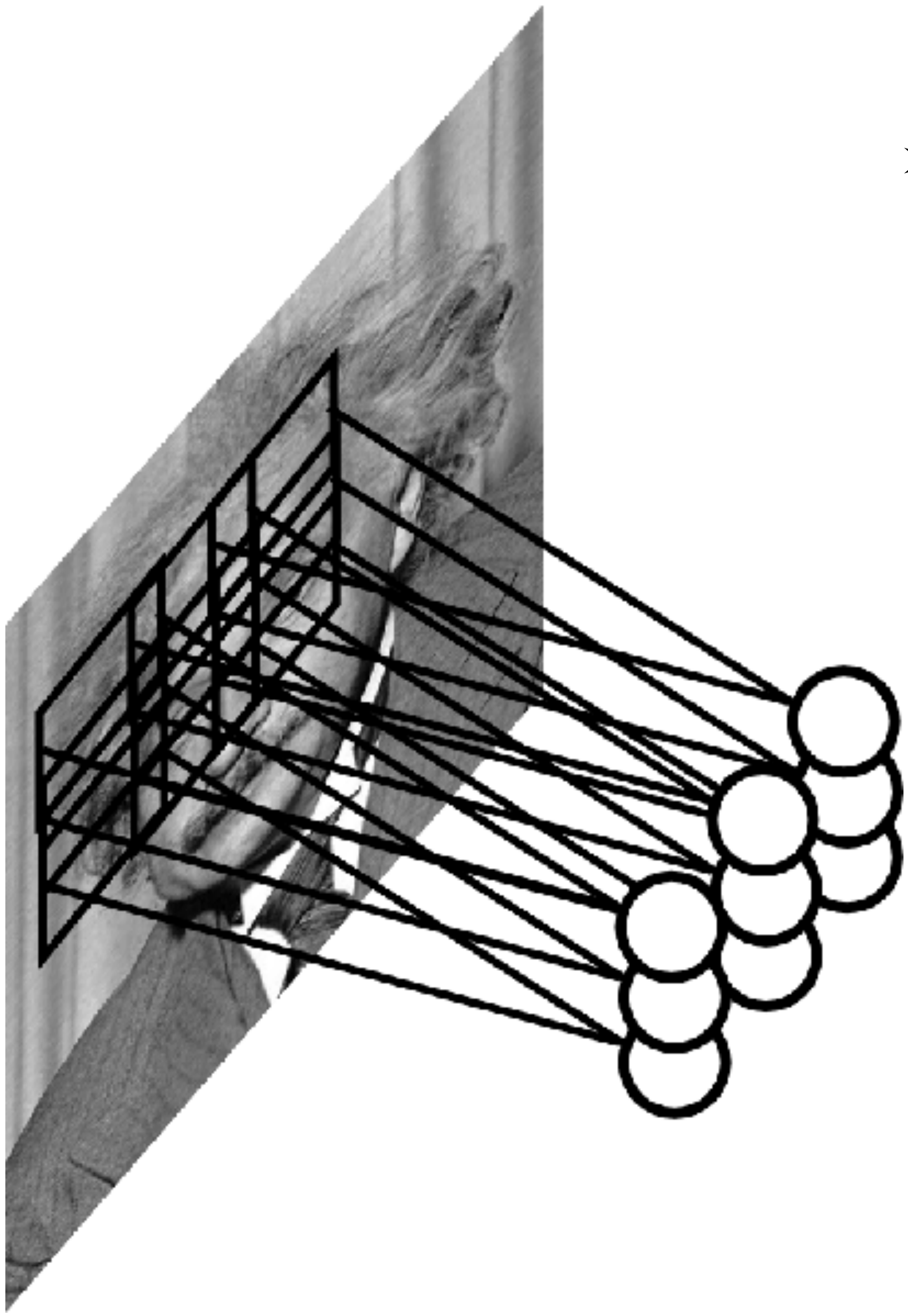


- Why pooling?

  ➢ Introduces invariance to local translations

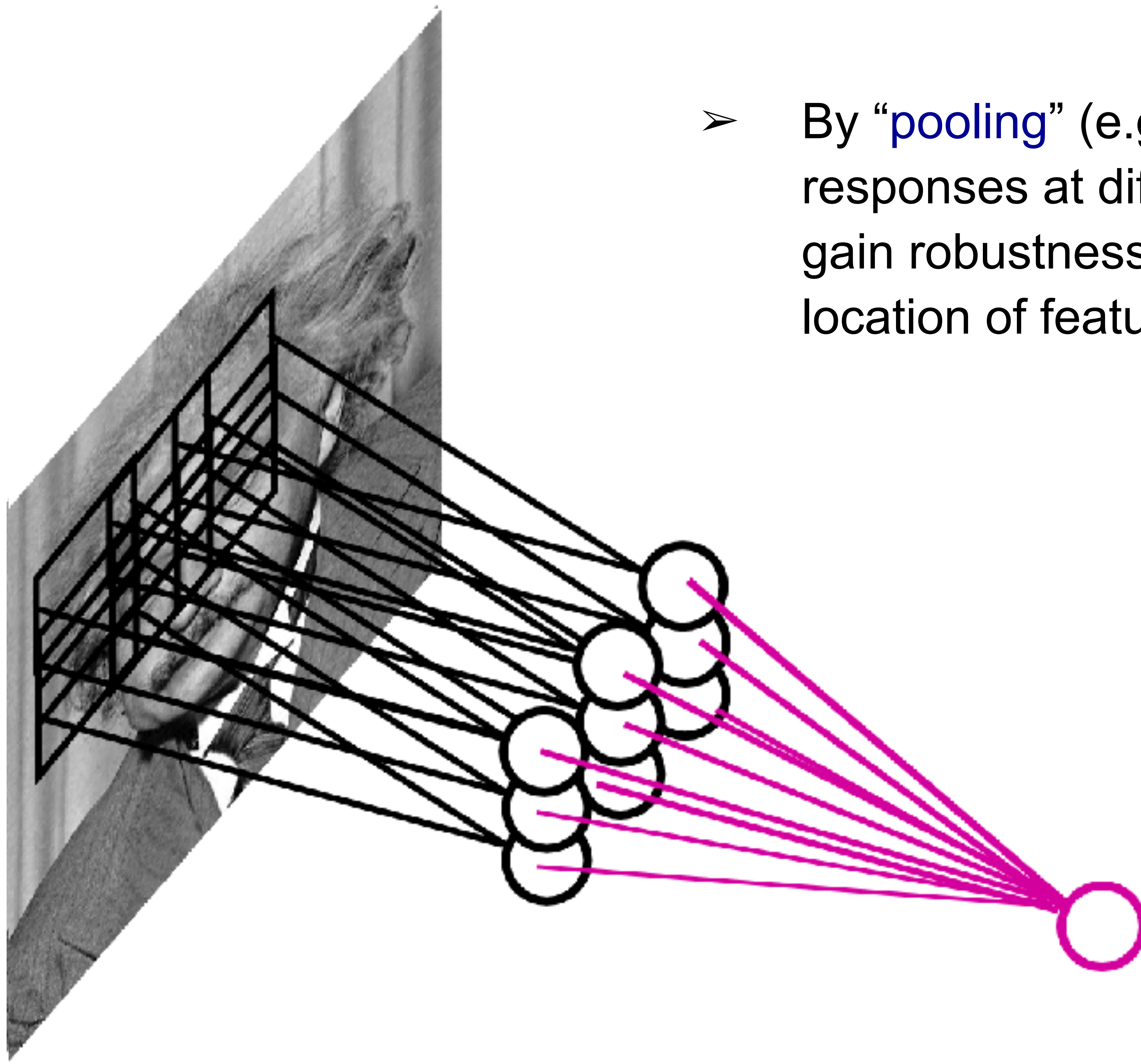
  ➢ Reduces the number of hidden units in hidden layer

Example: Pooling



- can we make the detection robust to the exact location of the eye?

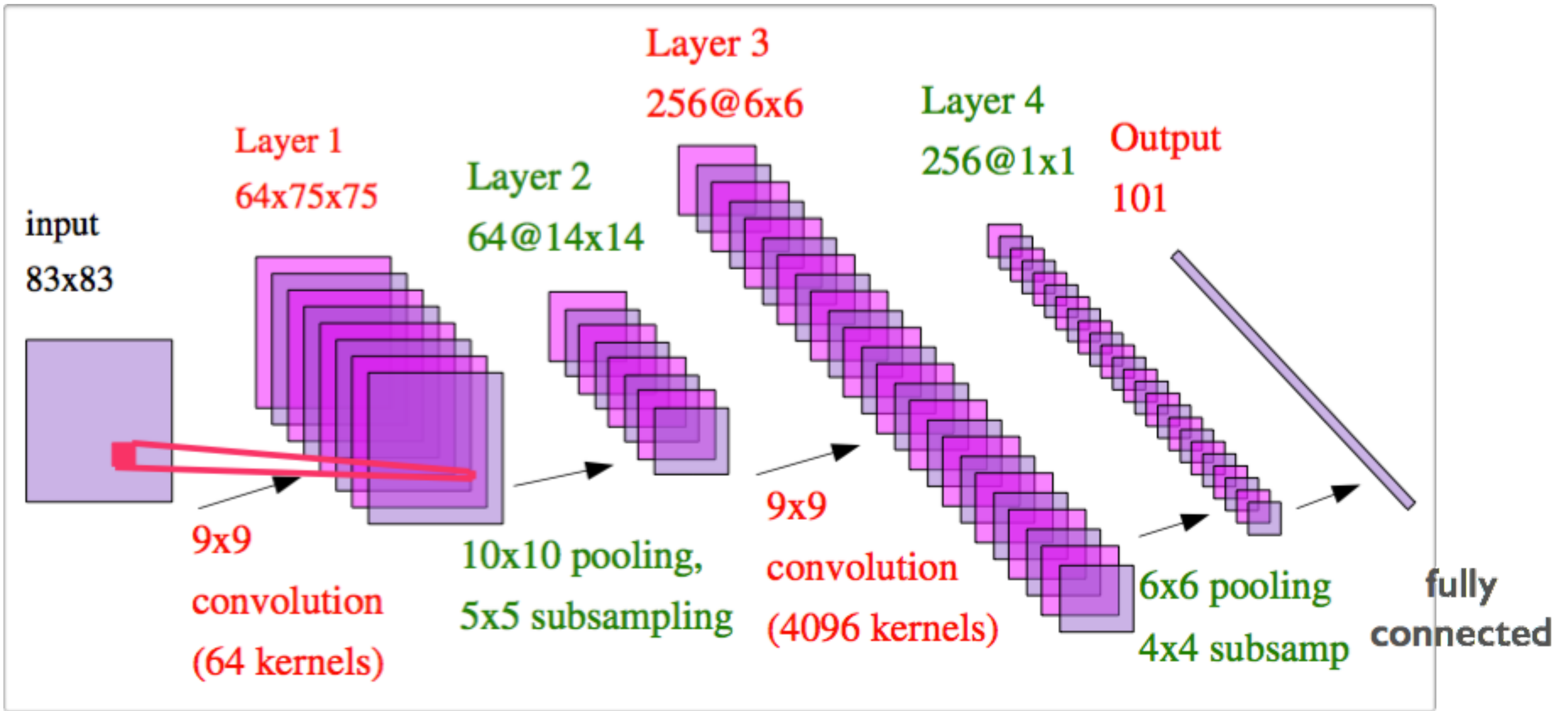
Example: Pooling



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

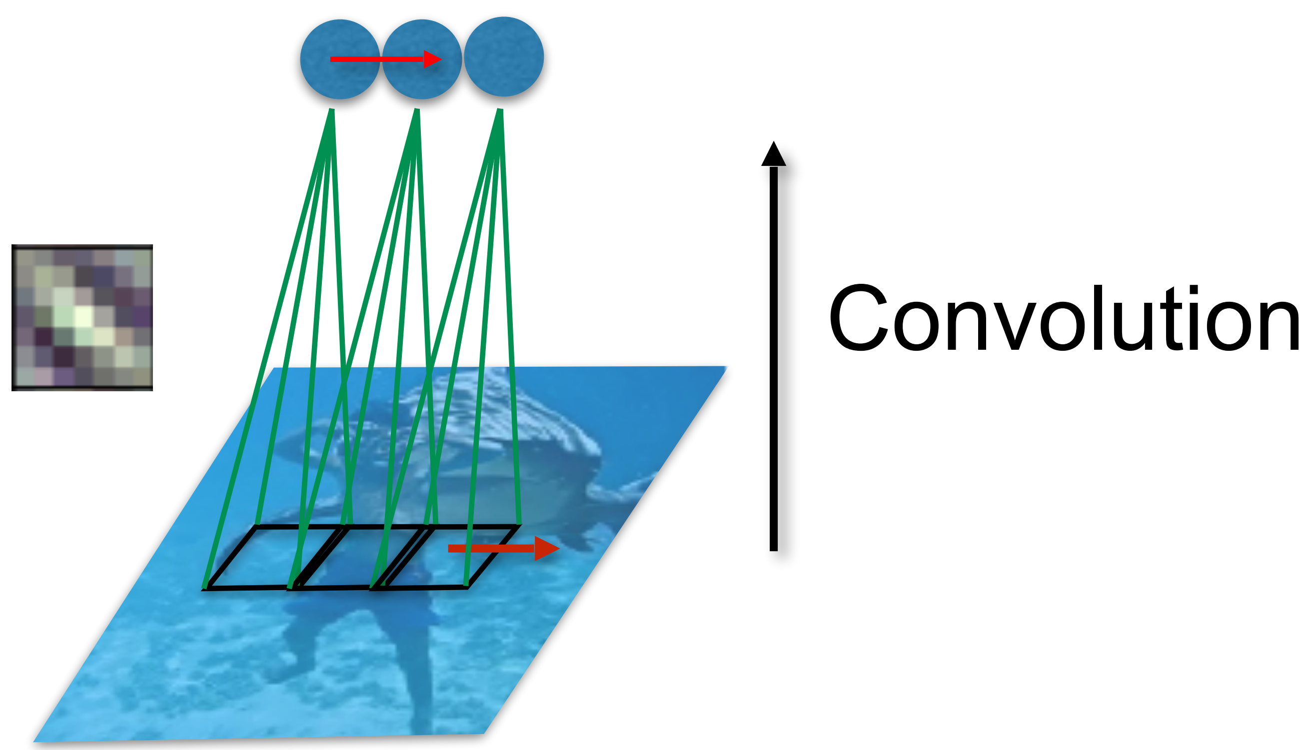
Convolutional Network

- Convolutional neural network alternates between the convolutional and pooling layers

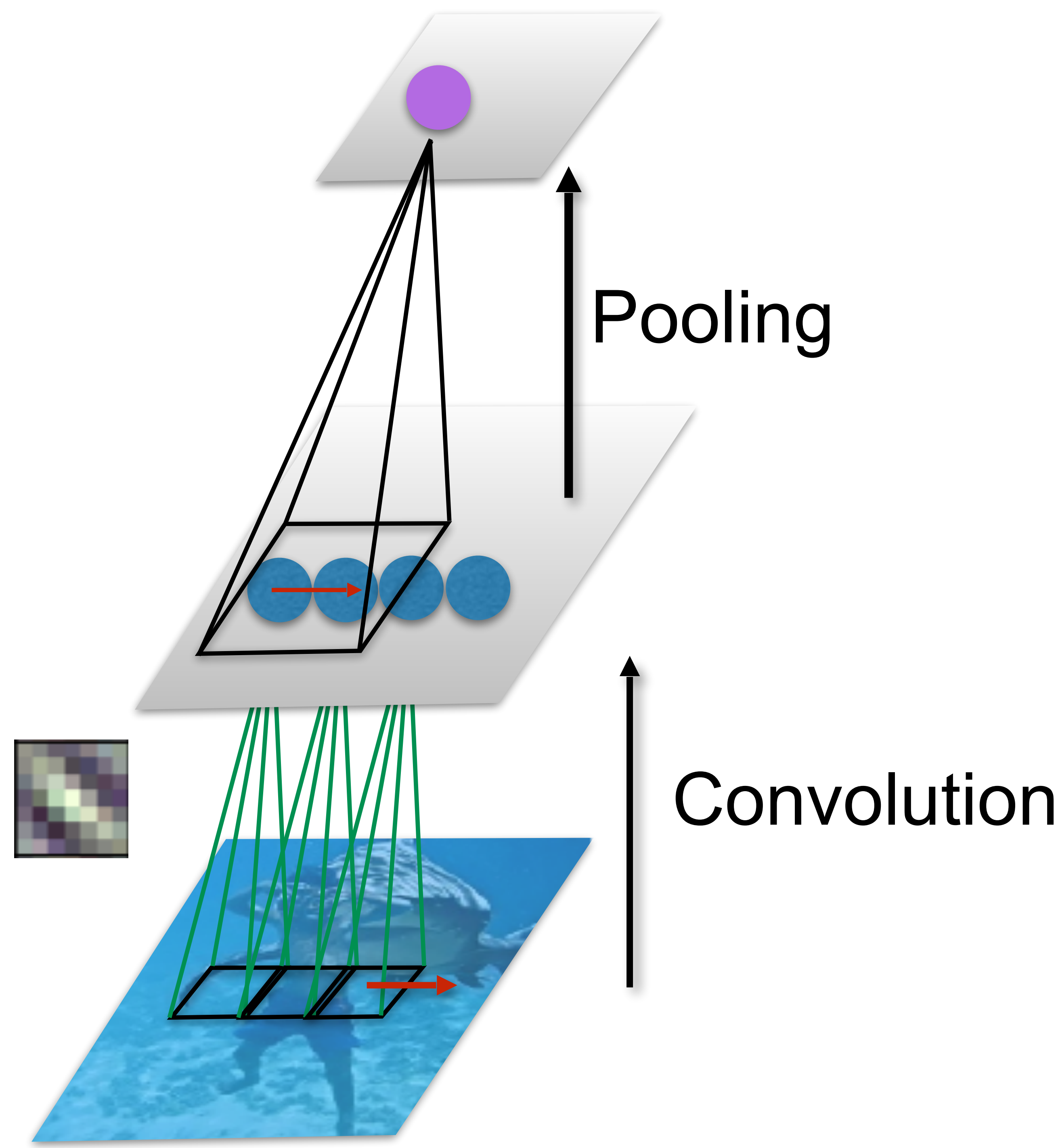


From Yann LeCun's slides

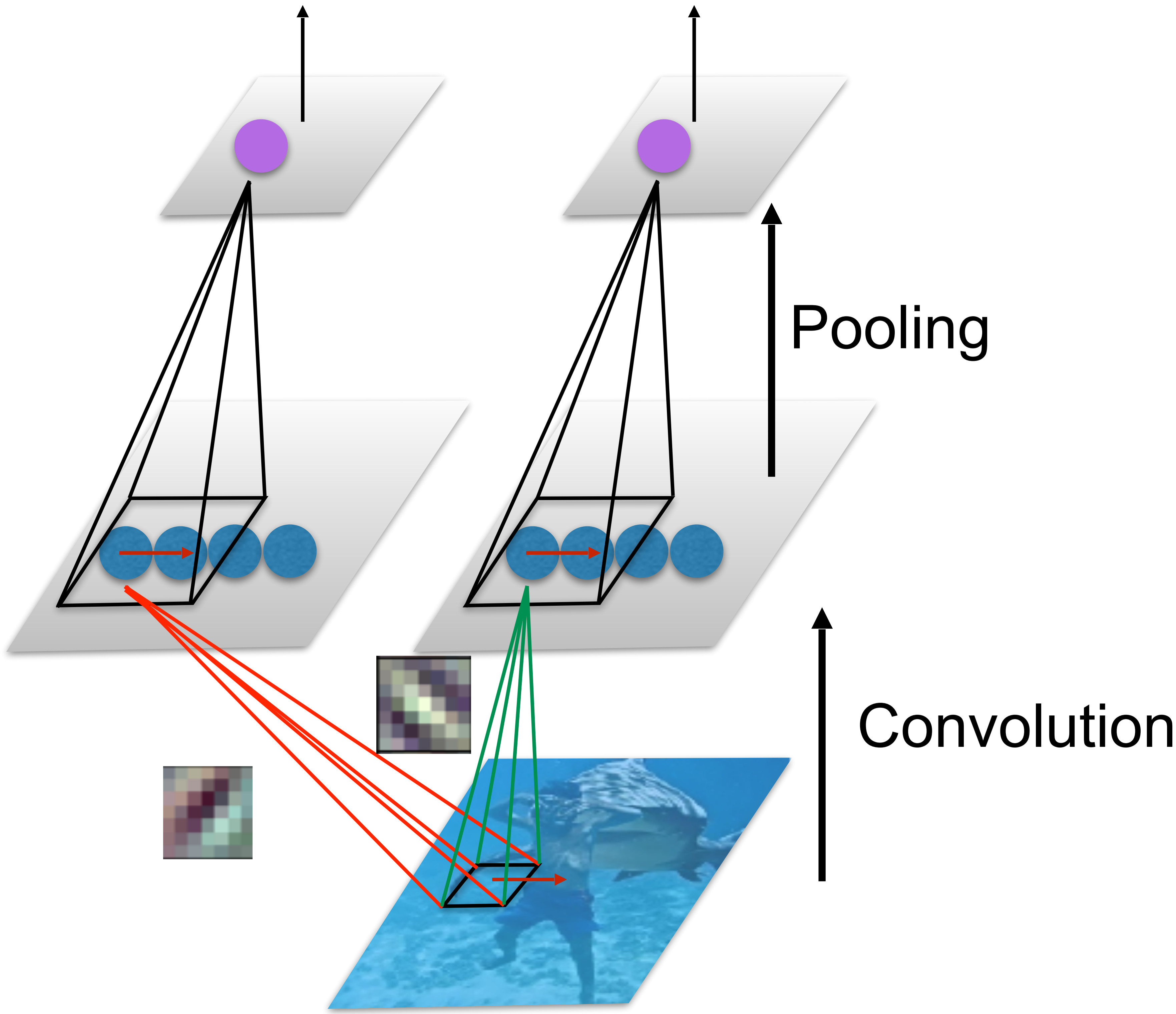
Deep Convolutional Nets



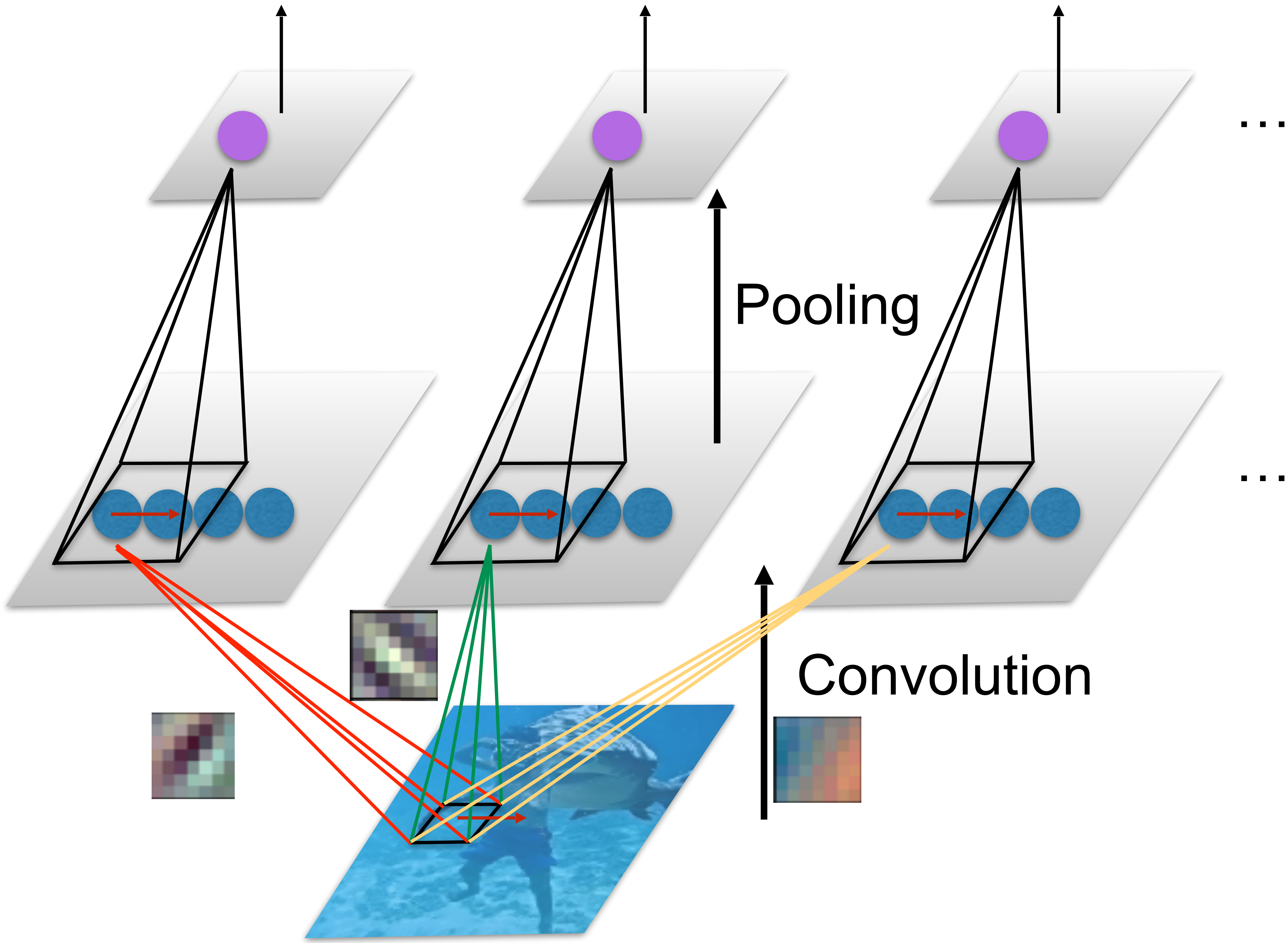
Deep Convolutional Nets



Deep Convolutional Nets



Deep Convolutional Nets

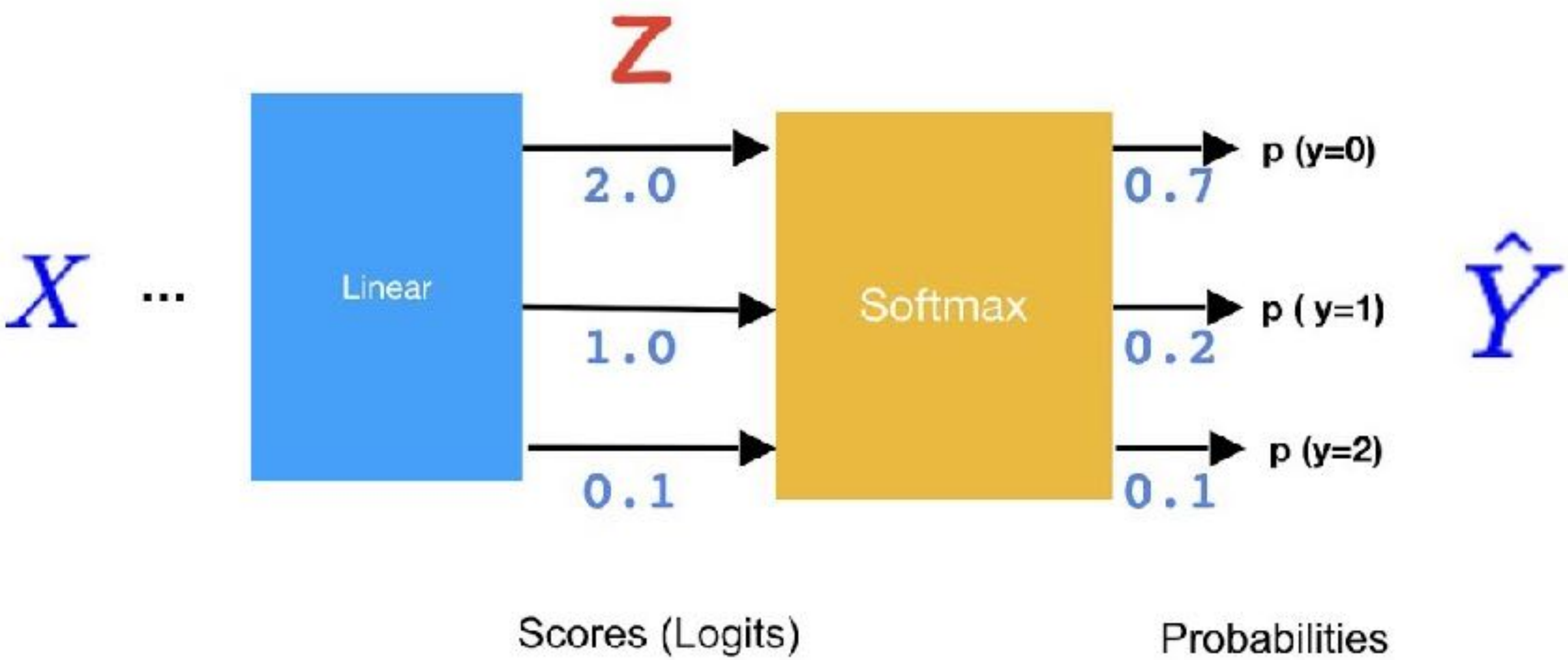


Convolutional Network

- For classification: Output layer is a regular, fully connected layer with softmax non-linearity
    - ➢ Output provides an estimate of the conditional probability of each class

Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



ImageNet Dataset

- 1.2 million images, 1000 classes

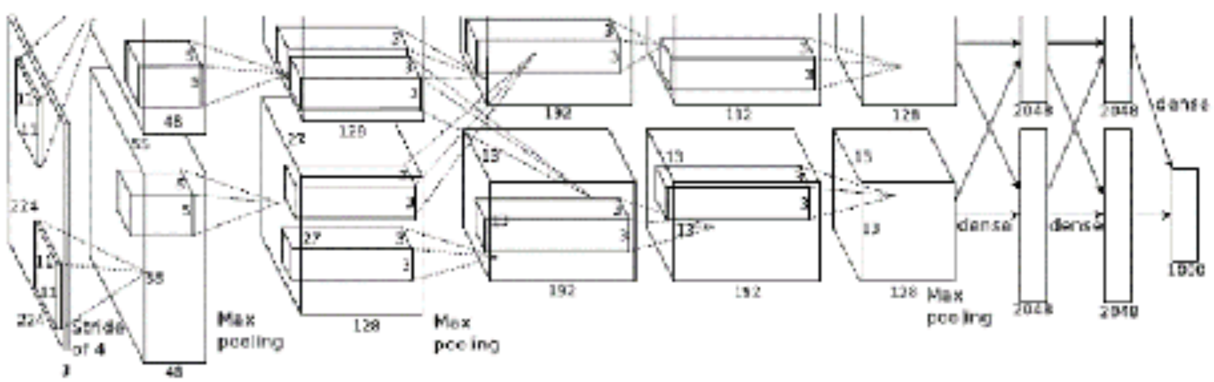
Examples of Hammer

(Deng et al., Imagenet: a large scale hierarchical image database, CVPR 2009)

Important Breakthrough

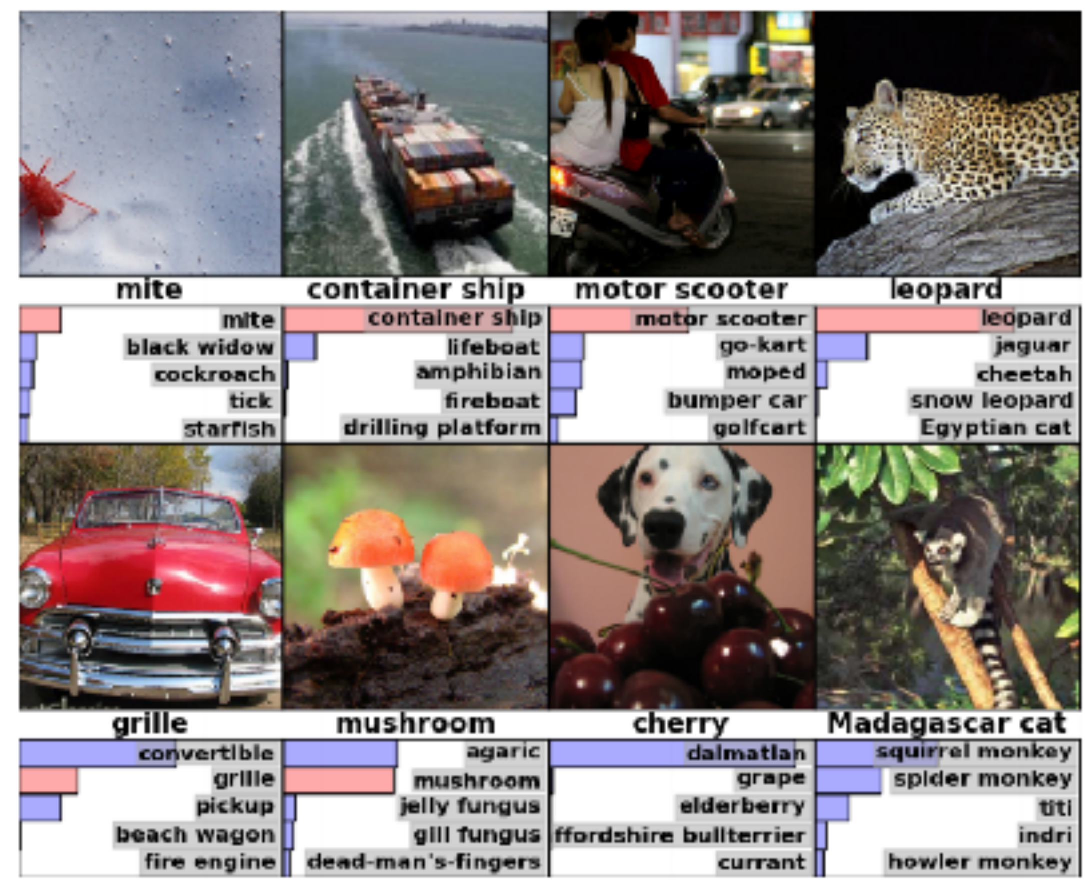
- Deep Convolutional Nets for Vision (Supervised)

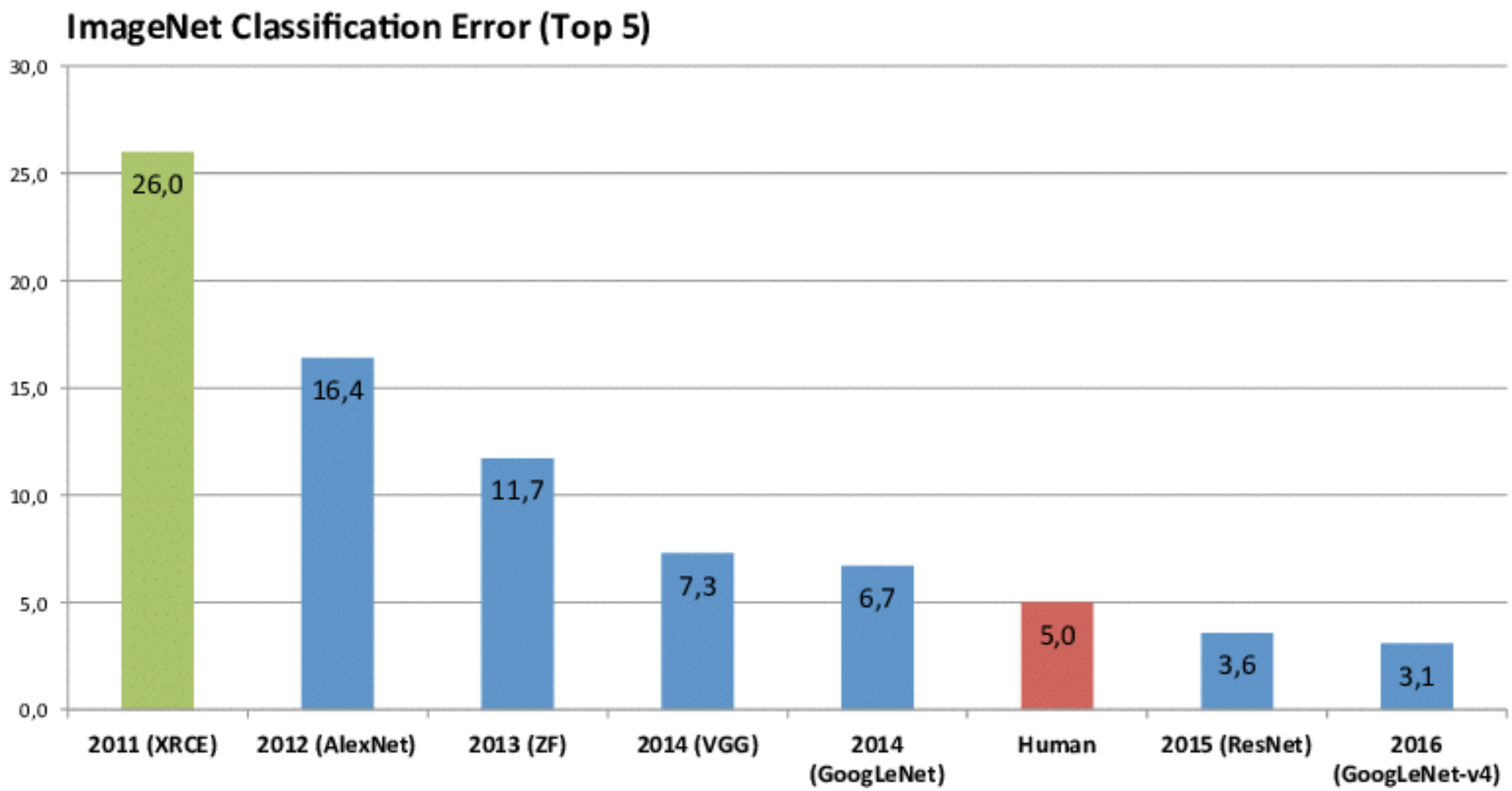
Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images
1000 classes



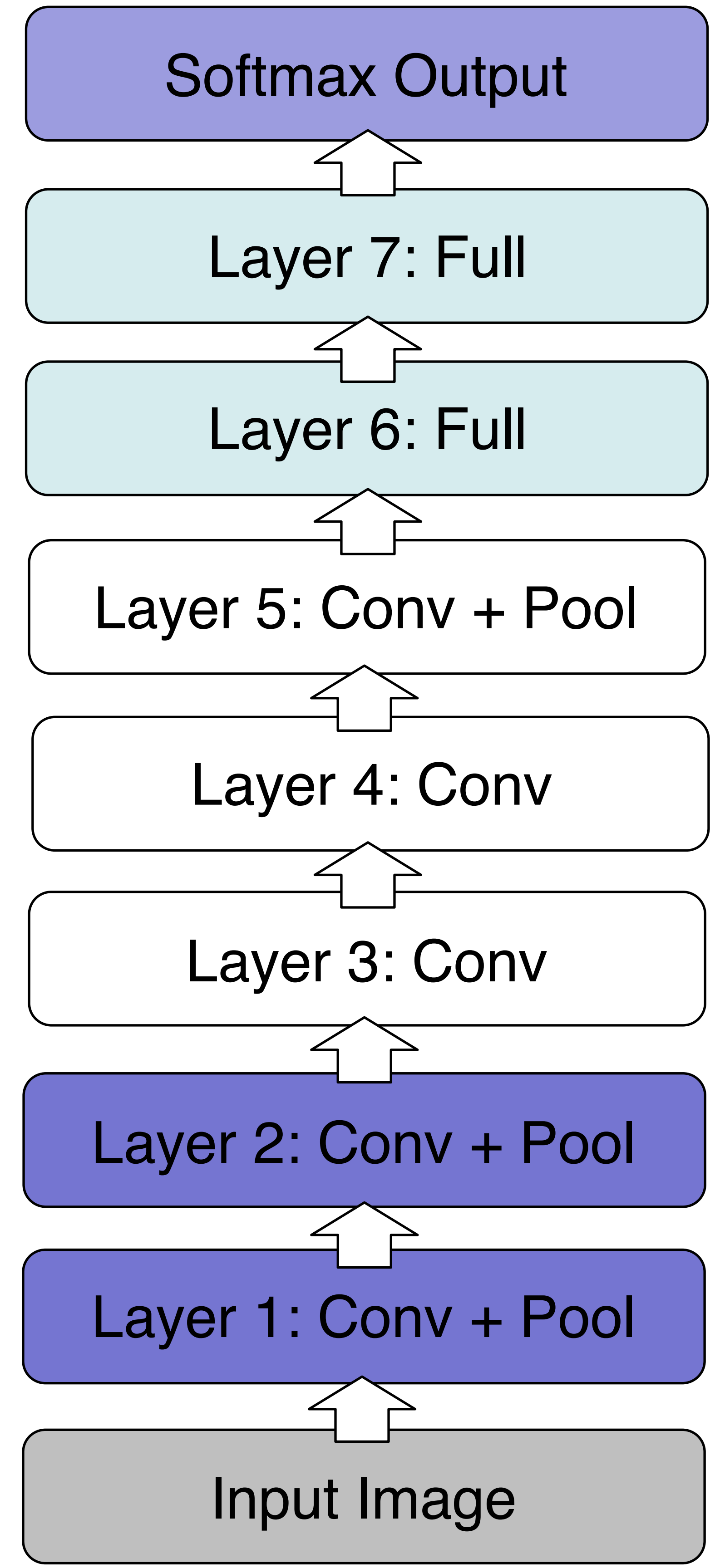

ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)

AlexNet

- 8 layers total

- Trained on Imagenet dataset [Deng et al. CVPR'09]

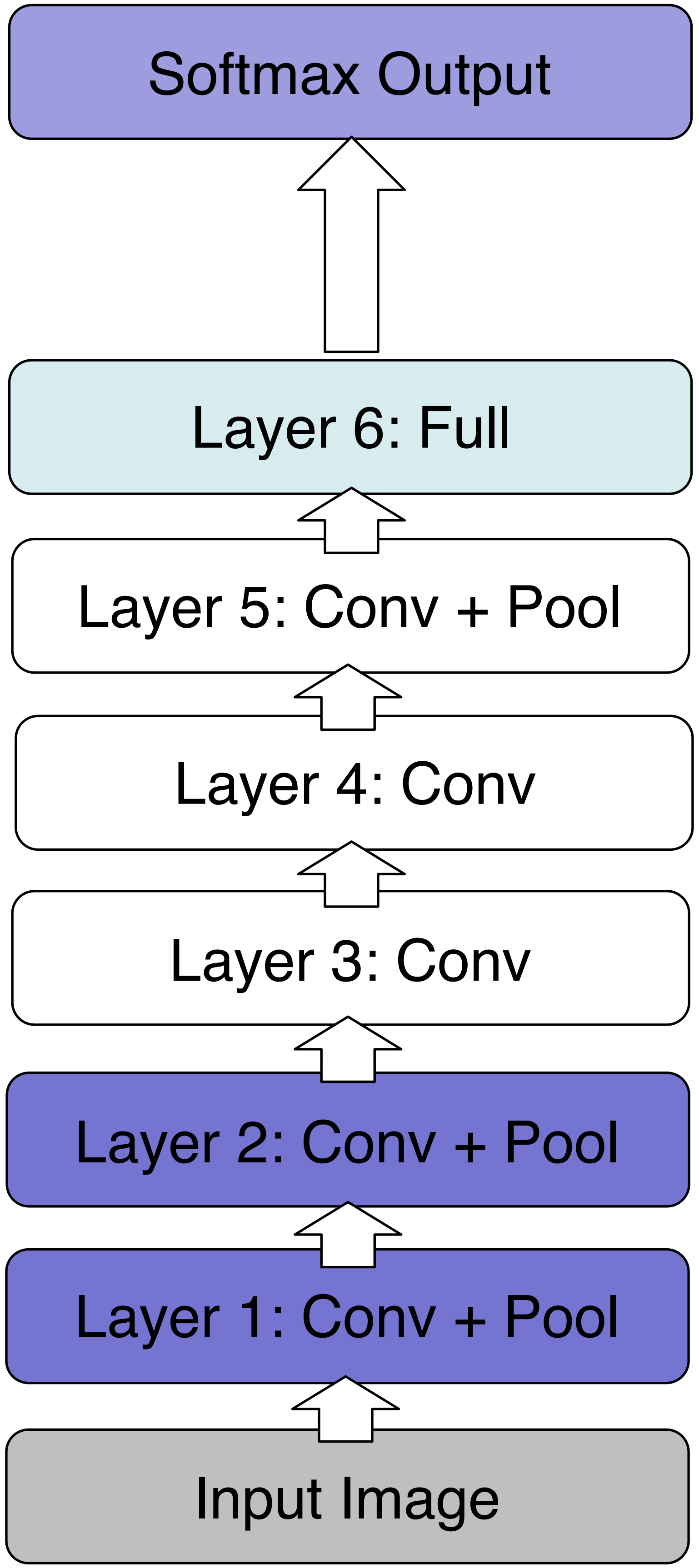
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

- Let us remove upper feature extractor layers and fully connected:

    ➢ Layers 3,4, 6 and 7

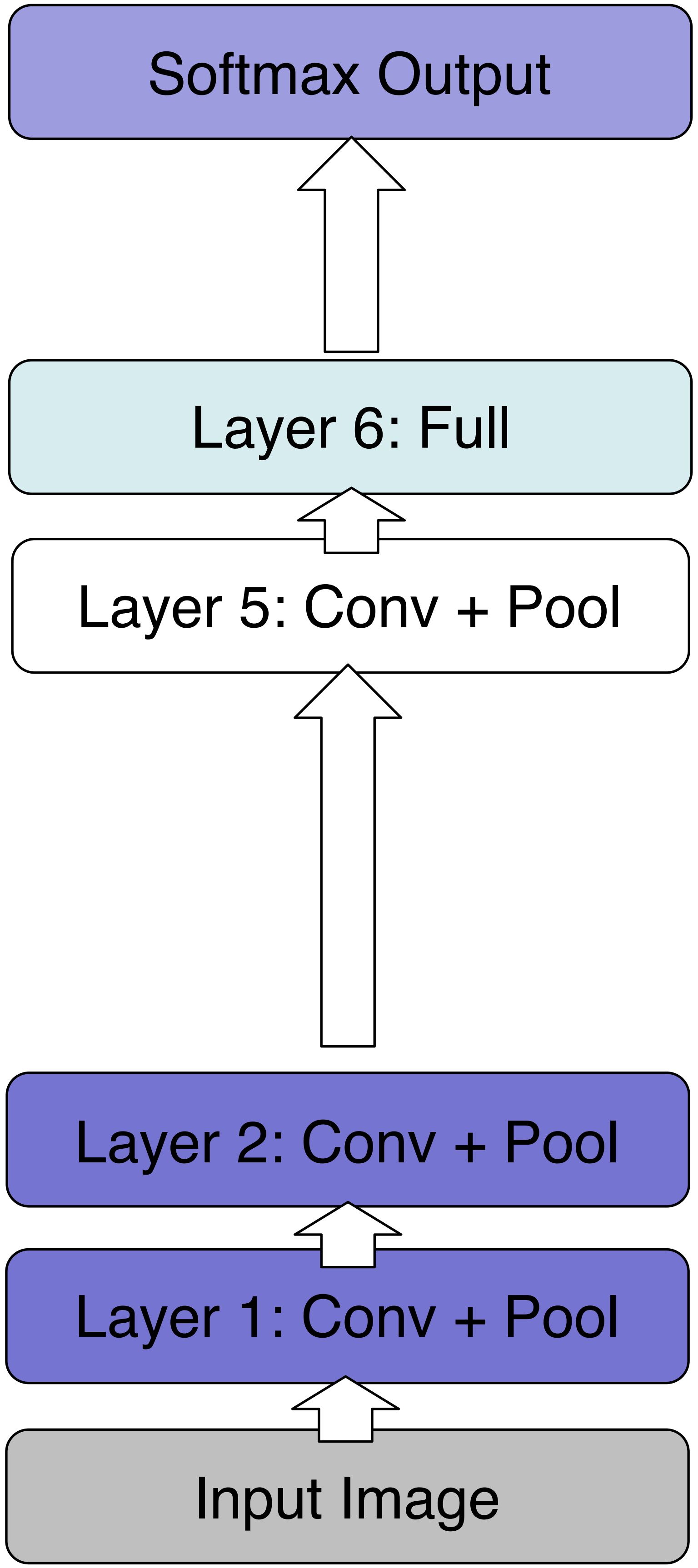
- Drop ~50 million parameters

- **33.5 drop in performance!**

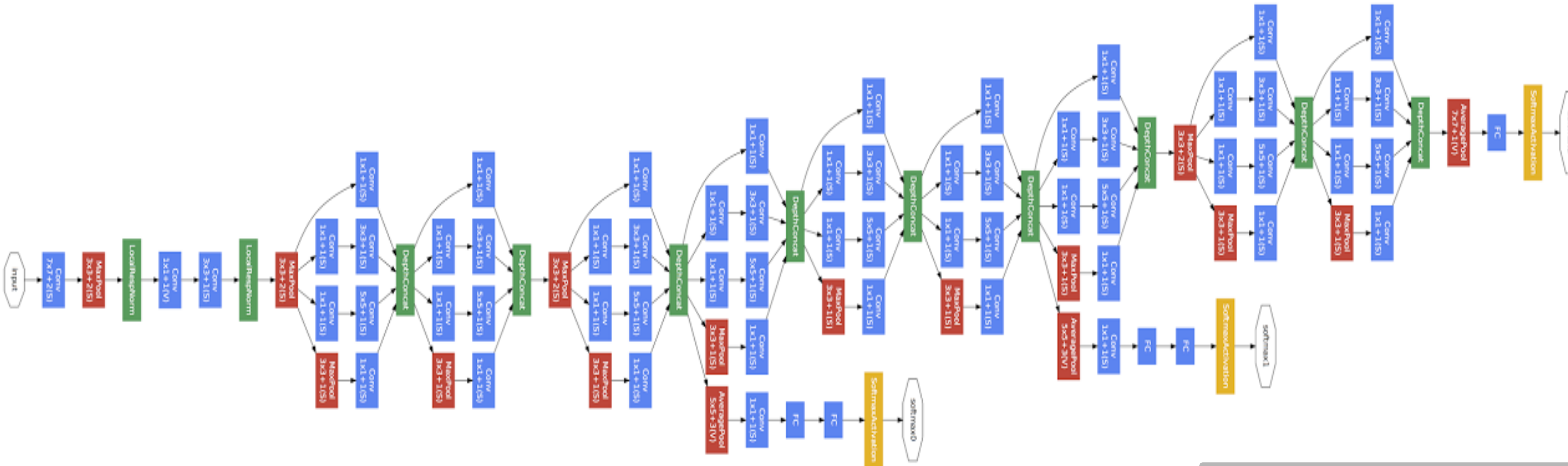
- Depth of the network is the key.

Later architectures avoid stacks of FC layers as the increase the # of parameter by a lot and thus, fear of overfitting

[From Rob Fergus' CIFAR 2016 tutorial]



GoogLeNet

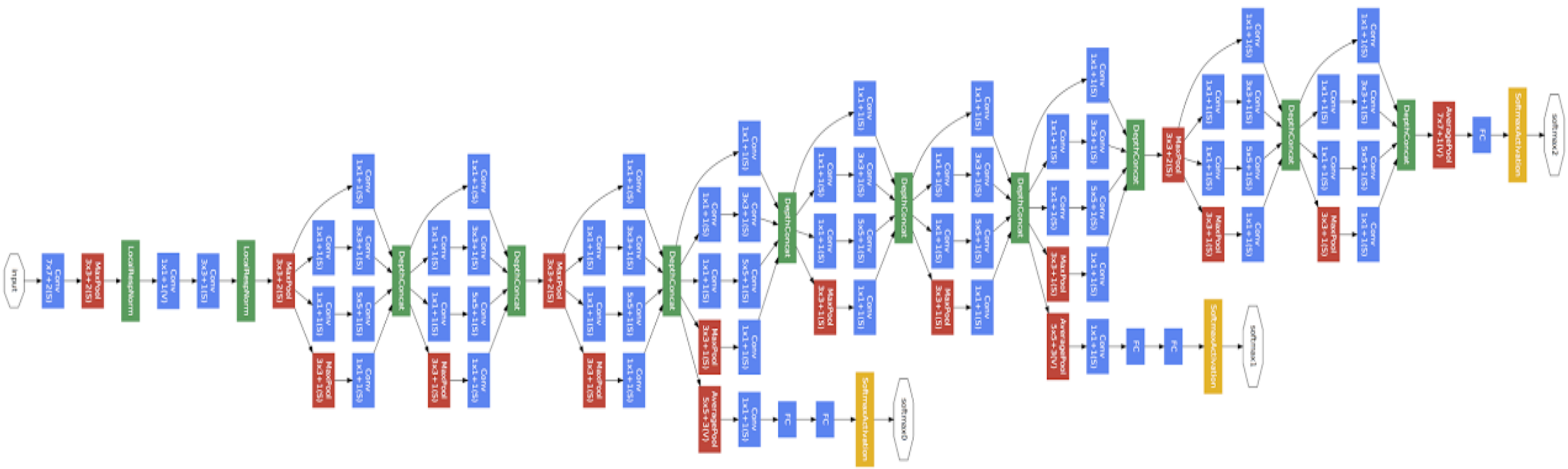


- 24 layer model that uses so-called inception module.

Convolution
Pooling
Softmax
Other

(Szegedy et al., Going Deep with Convolutions, 2014)

GoogLeNet



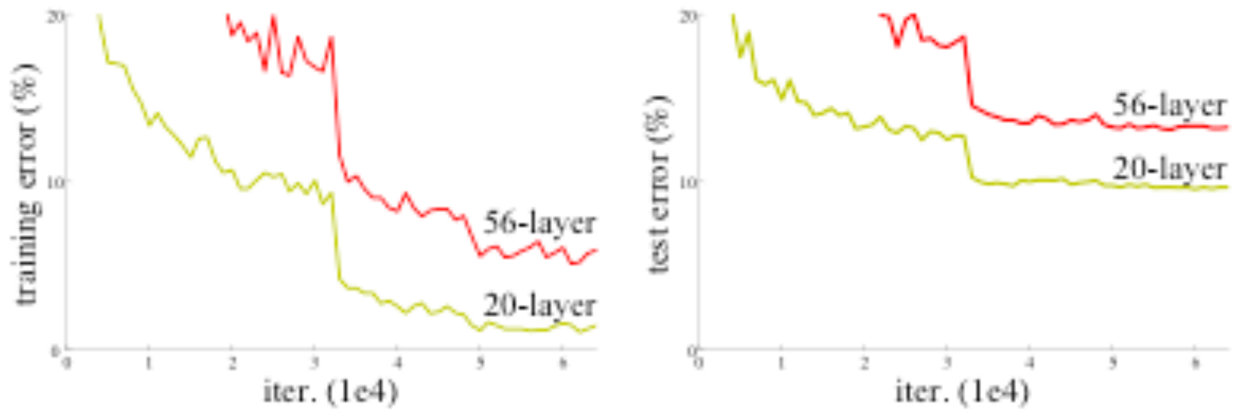
- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

(Szegedy et al., Going Deep with Convolutions, 2014)

Residual Networks

[He, Zhang, Ren, Sun, CVPR 2016]

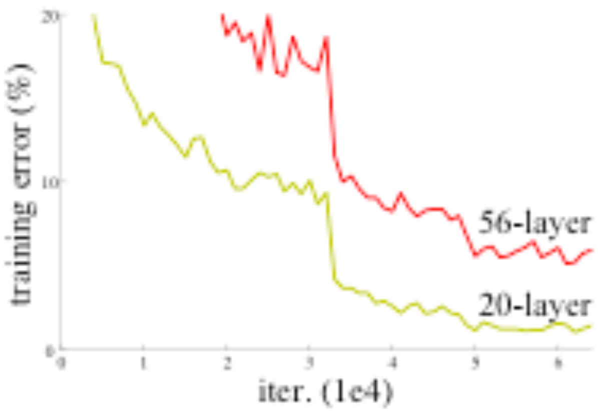
Really, really deep convnets don't train well,
E.g. CIFAR10:

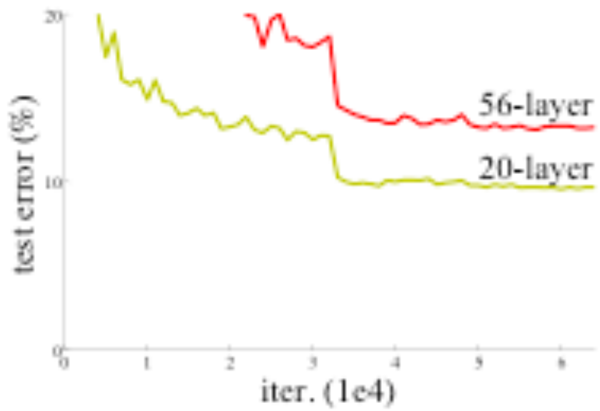


Residual Networks

[He, Zhang, Ren, Sun, CVPR 2016]

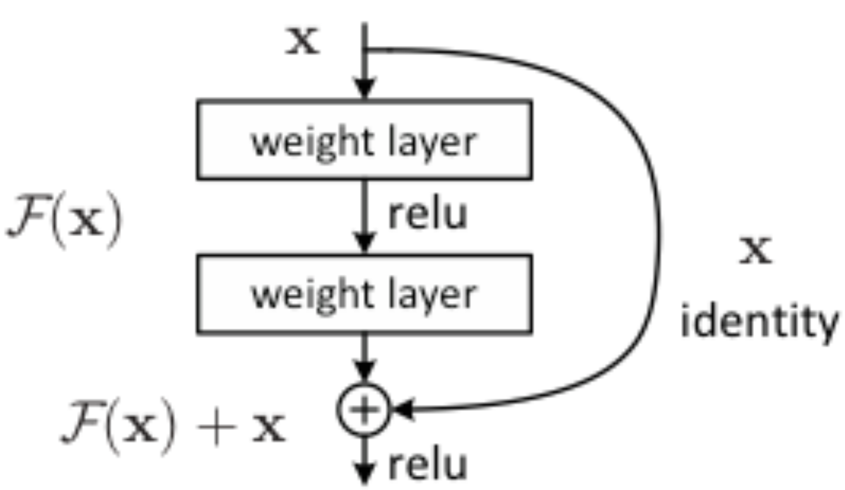
Really, really deep convnets don't train well,
E.g. CIFAR10:

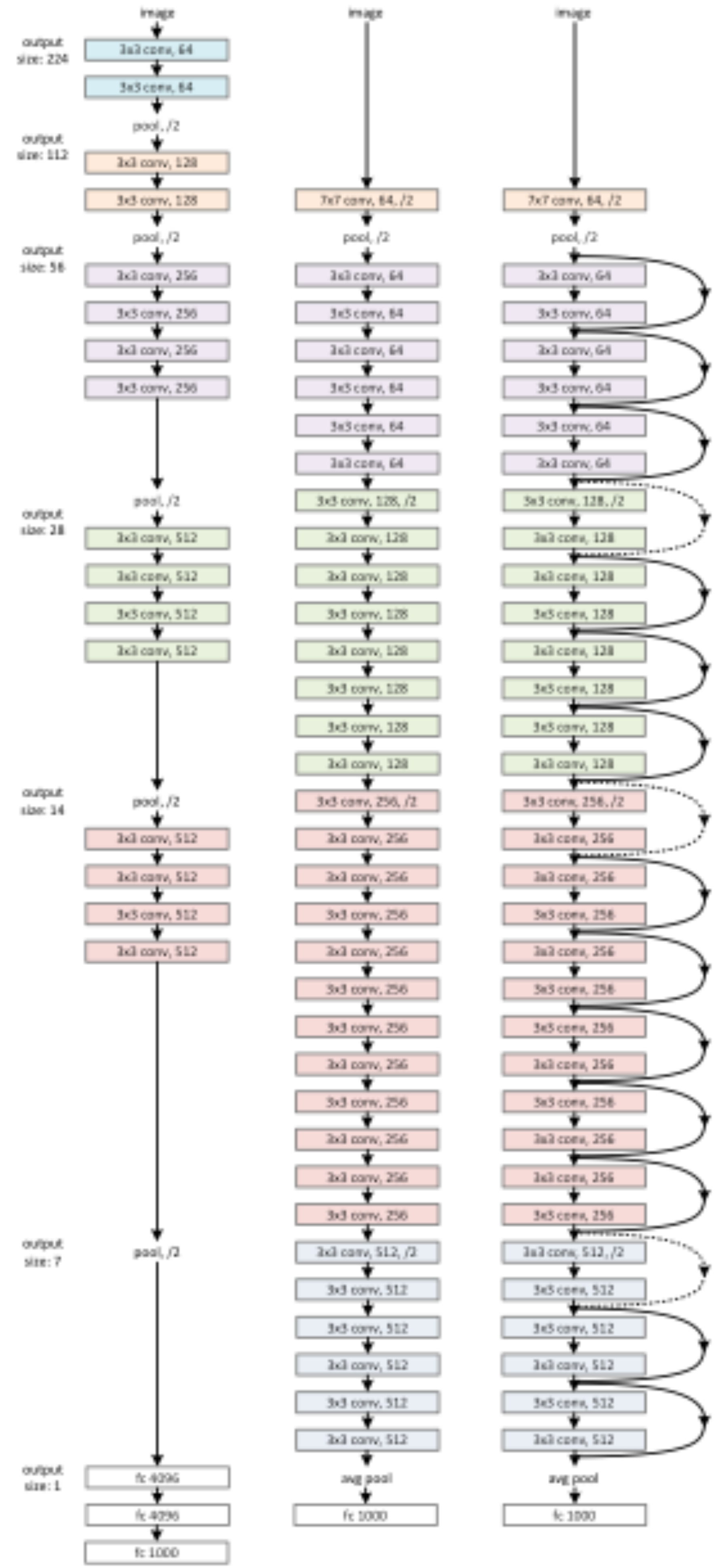




Key idea: introduce "pass through" into each layer

Thus only residual now needs to be learned

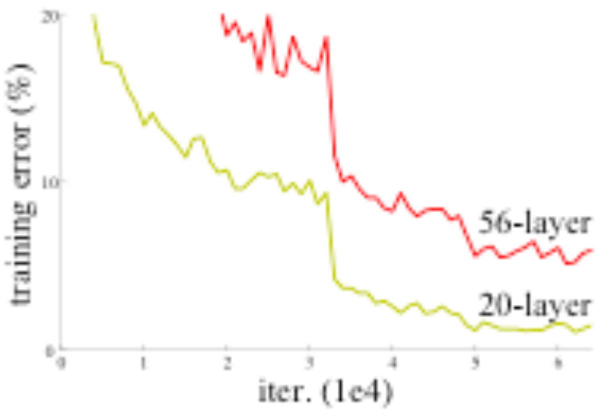


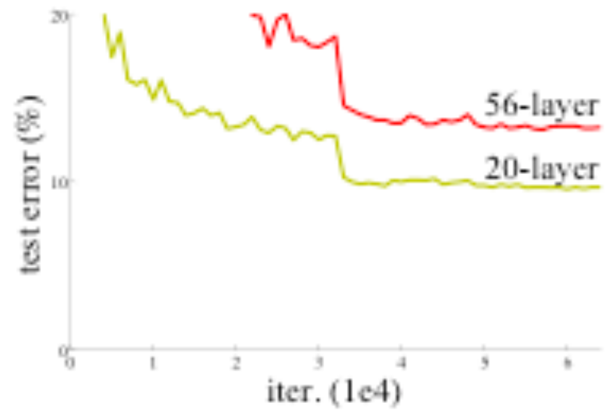


Residual Networks

[He, Zhang, Ren, Sun, CVPR 2016]

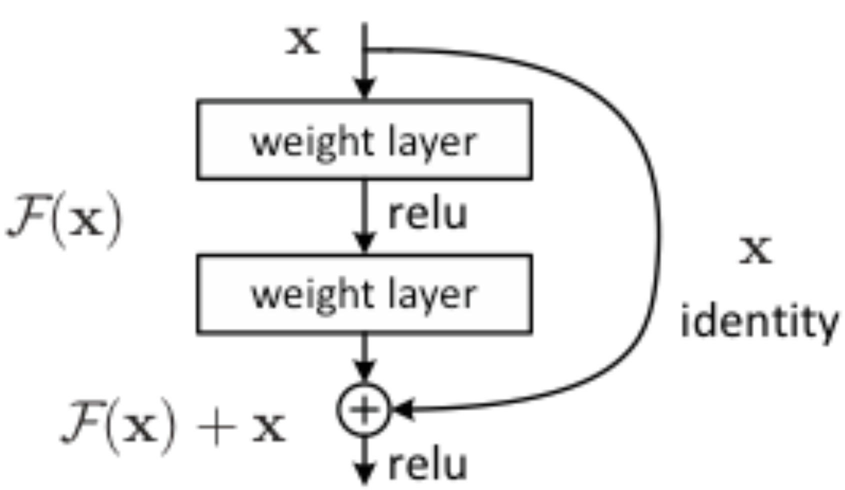
Really, really deep convnets don't train well,
E.g. CIFAR10:





Key idea: introduce "pass through" into each layer

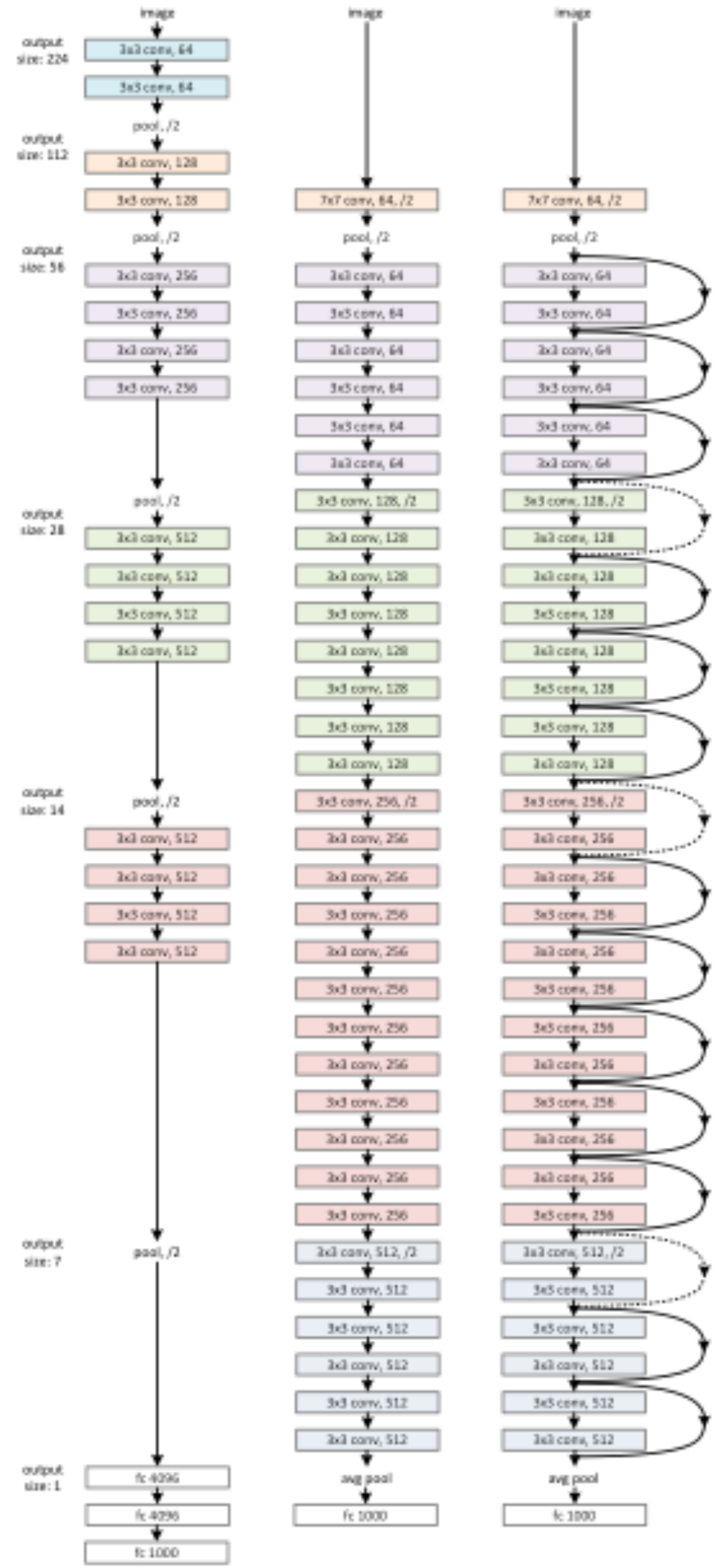
Thus only residual now needs to be learned



method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

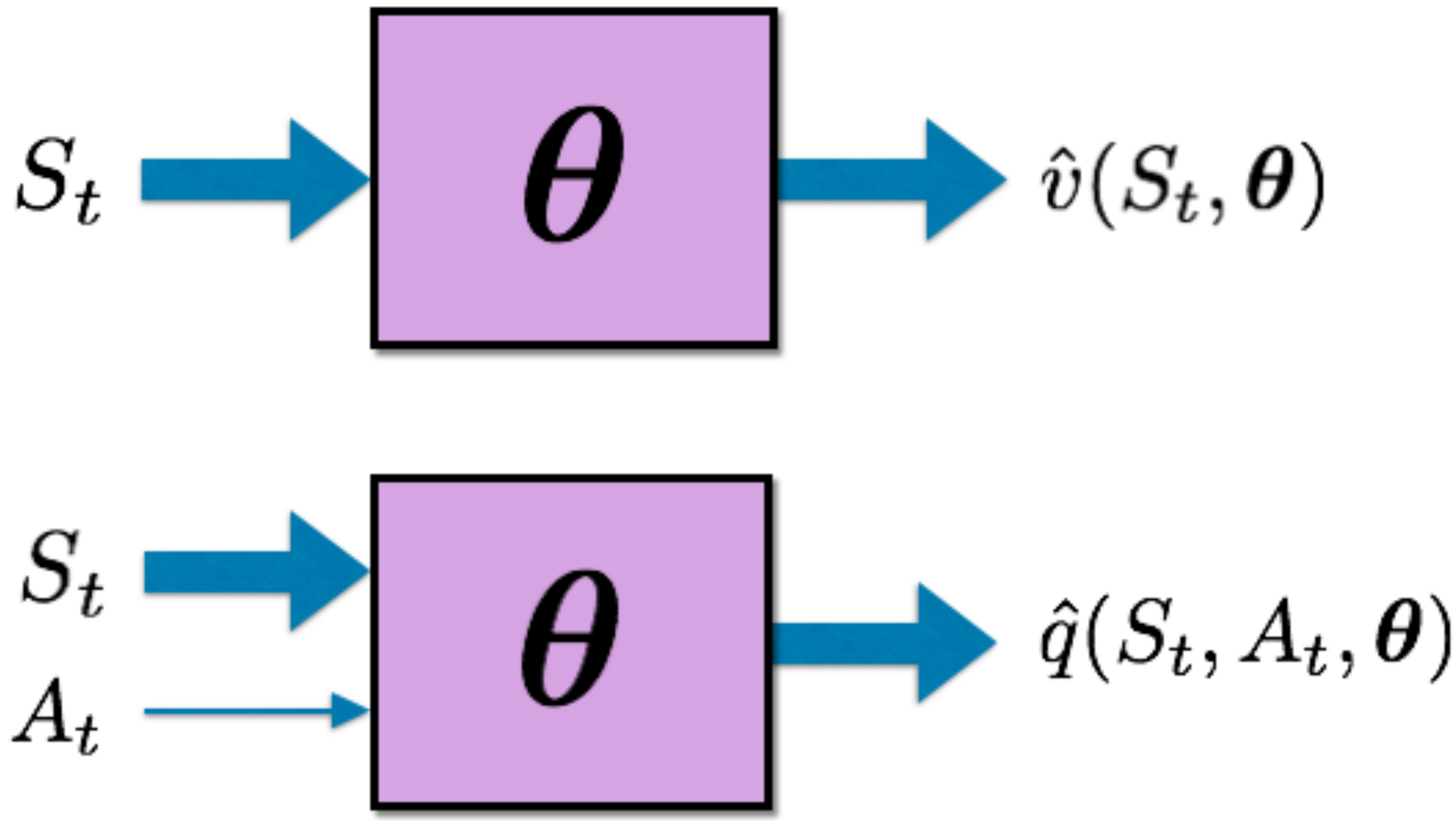
Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

With ensembles, 3.57% top-5 test error on ImageNet



Value Function Approximation (VFA)

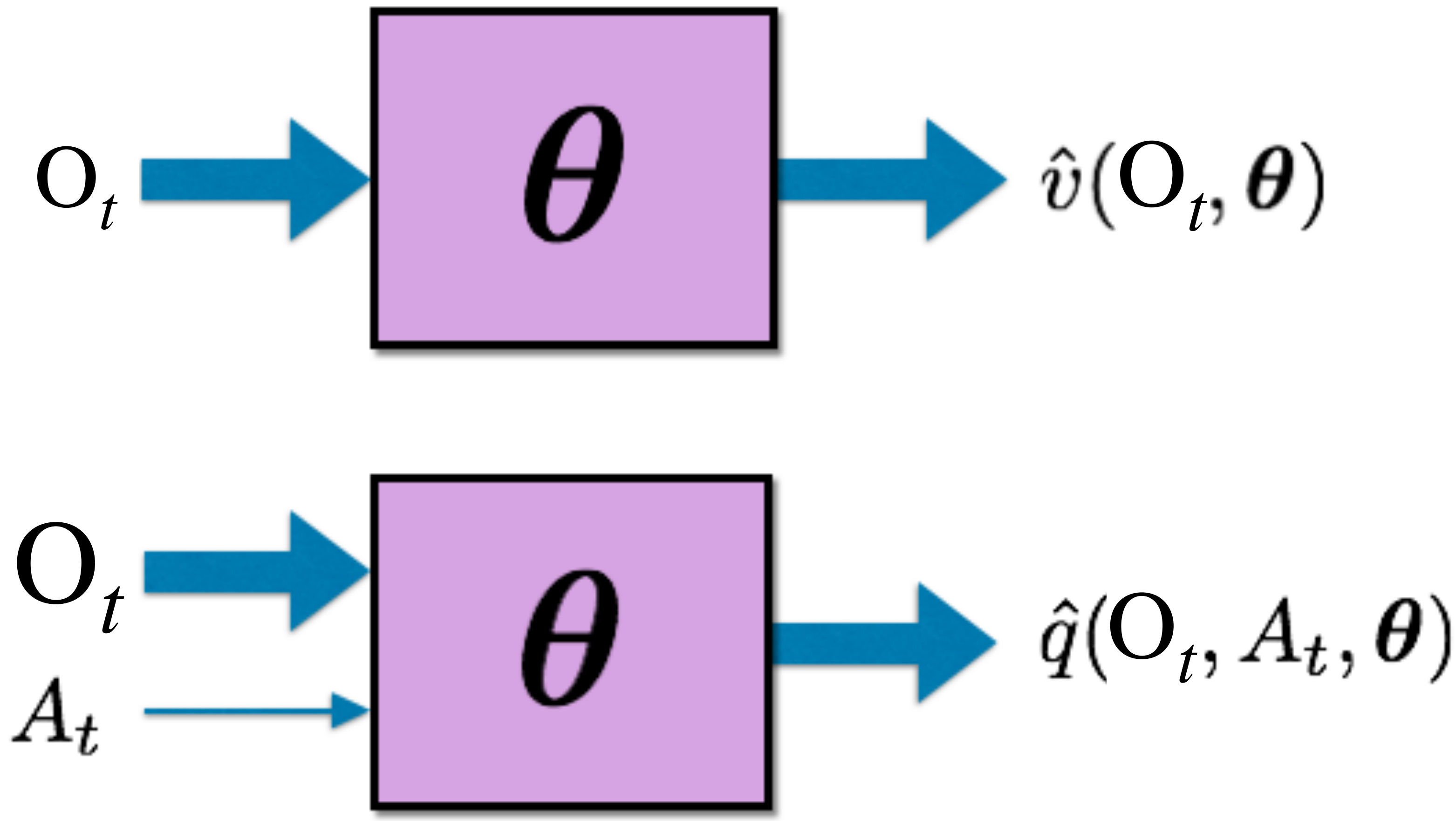
- Value function approximation (VFA) replaces the table with a general parameterized form:



Why this is a good idea?
Because those functions can be trained to map *similar* states to similar values. E.g., for a navigation agent the color of the carpet does not matter, or how many vases are on the table, let alone the amount of sun coming in from the window.

End-to-End RL

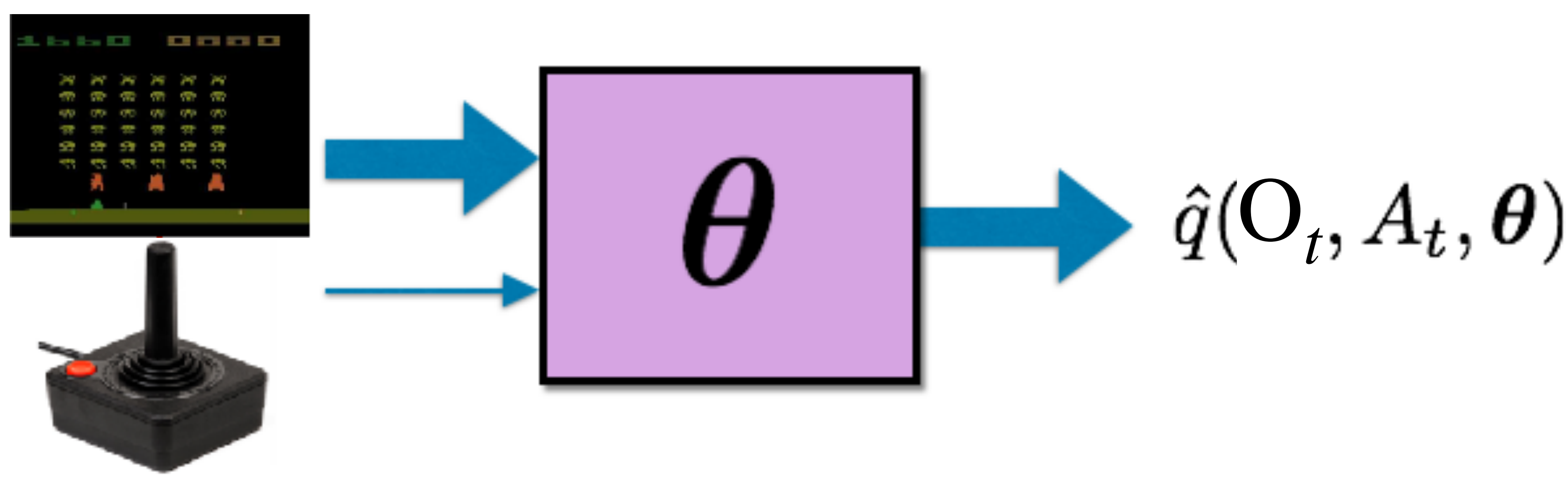
- End-to-end RL methods replace the hand-designed state representation with raw observations.



- We get rid of manual design of state representations :-)
- We need tons of data to train the network since O_t usually WAY more high dimensional than hand-designed S_t :-(

End-to-End RL

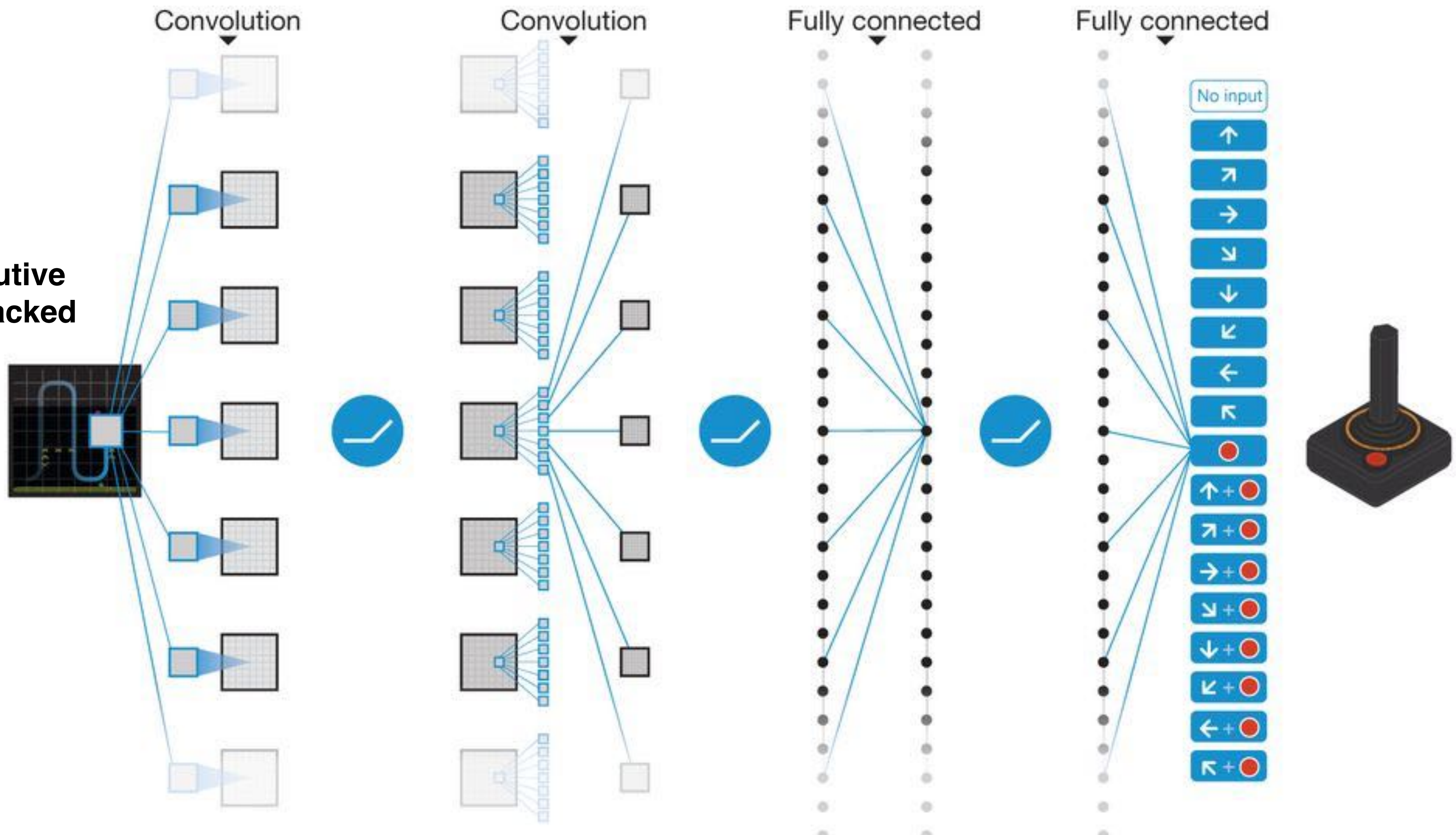
- End-to-end RL methods replace the hand-designed state representation with raw observations.



Playing Atari with Deep reinforcement learning, deepmind, 2013: The first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning

End-to-End RL

4 consecutive frames stacked



- The network learns to map observations to states to actions
- All Q values for all actions are compute in one forward pass!
- Initial image is 210X160-> downsample->110X84-> crop-> 84X84-> input to network

`Playing Atari with Deep reinforcement learning, 2013: The first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning

Architecture Search

- How can we select the right architecture:
  - ➢ Manual tuning of features is now replaced with the manual tuning of architectures

What does it mean to choose the right architecture?

- Depth
- Width
- Parameter count

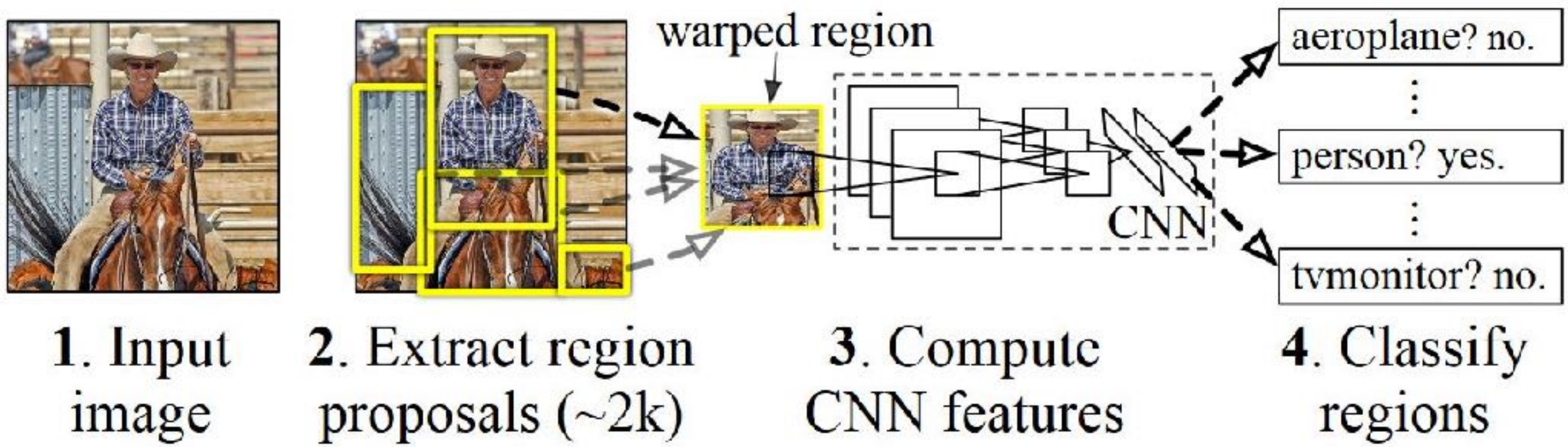
What structural biases we need to do well for learning action policies?

What-where decomposition

- Many tasks require fine grained understanding of the scene
- CNNs are good to predict ``what” is there, not ``where” it is, due to extensive pooling and loss of resolution in the upper layers.
- How can we represent both **what** is in the scene and **where it is**?

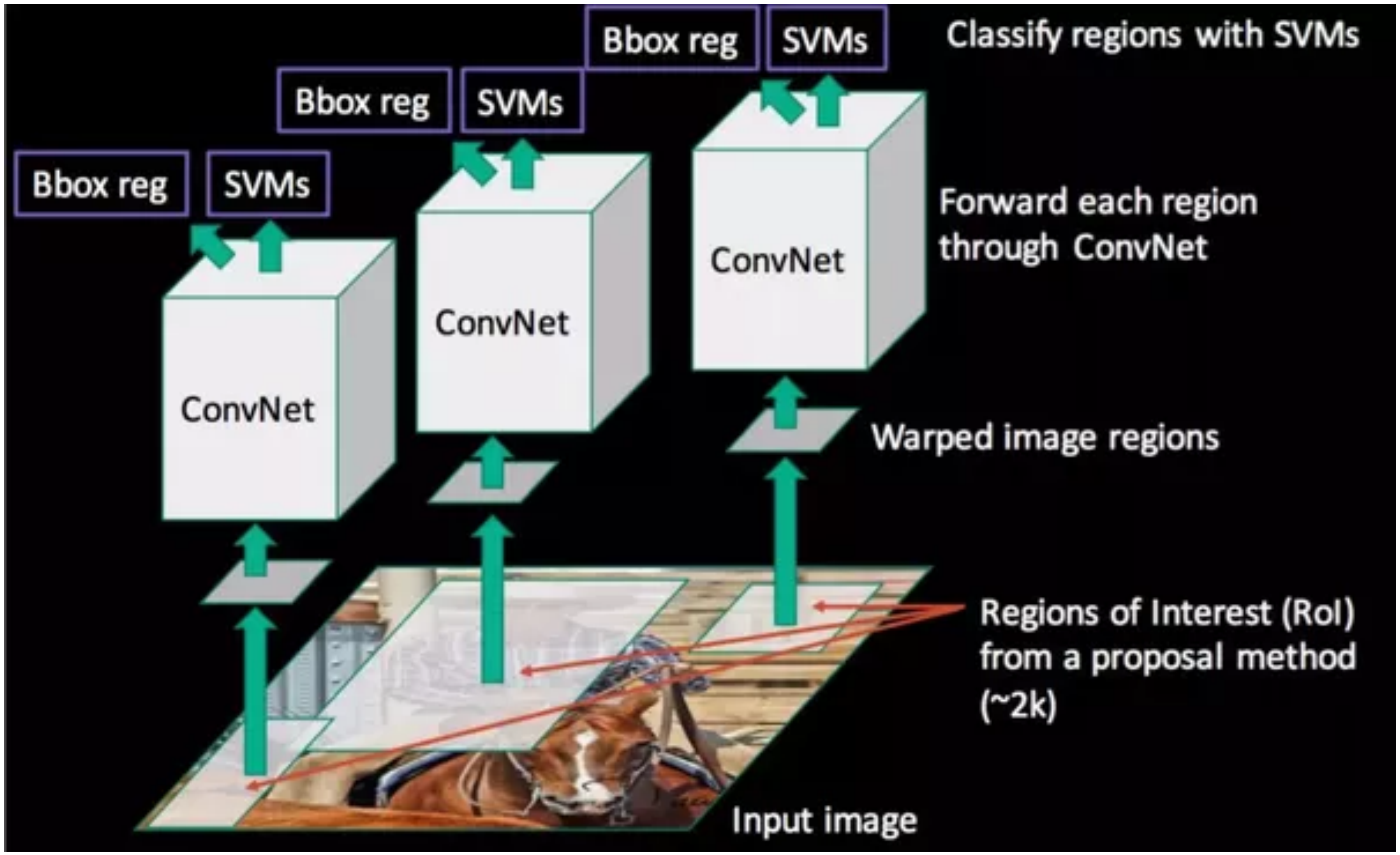
Object Detection

Deep Object detection



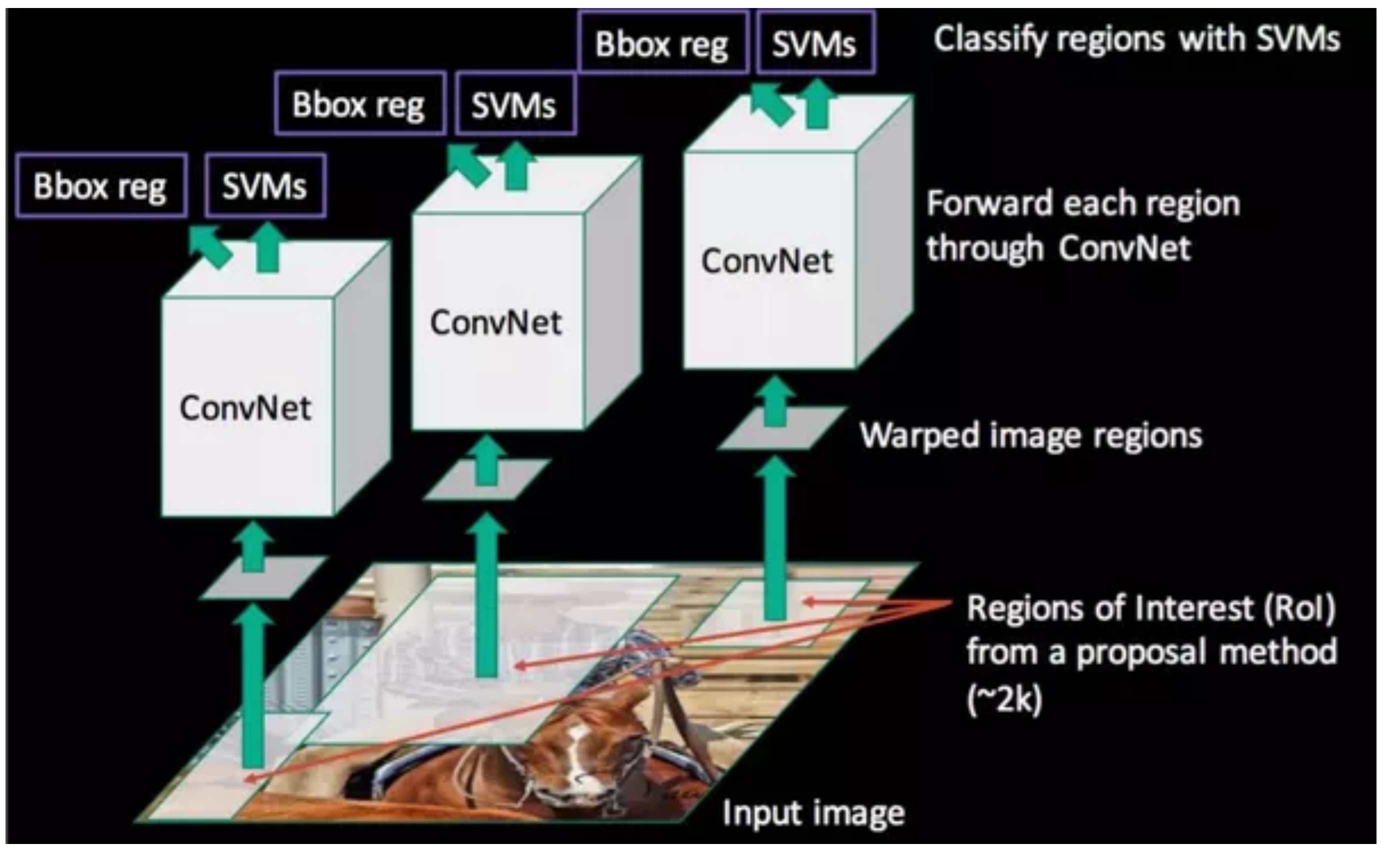
Region-CNN, Girshick et al., 2013

Object detection

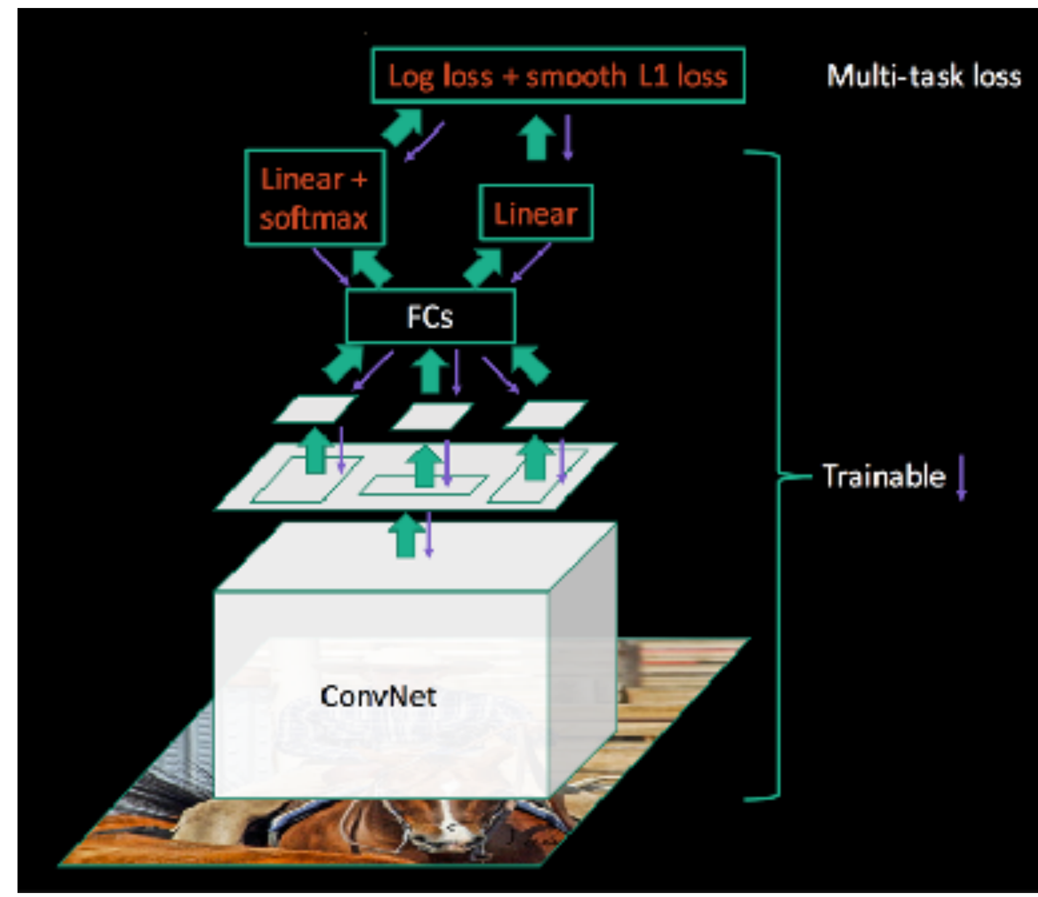


Region-CNN, Girshick et al., 2013

Object detection

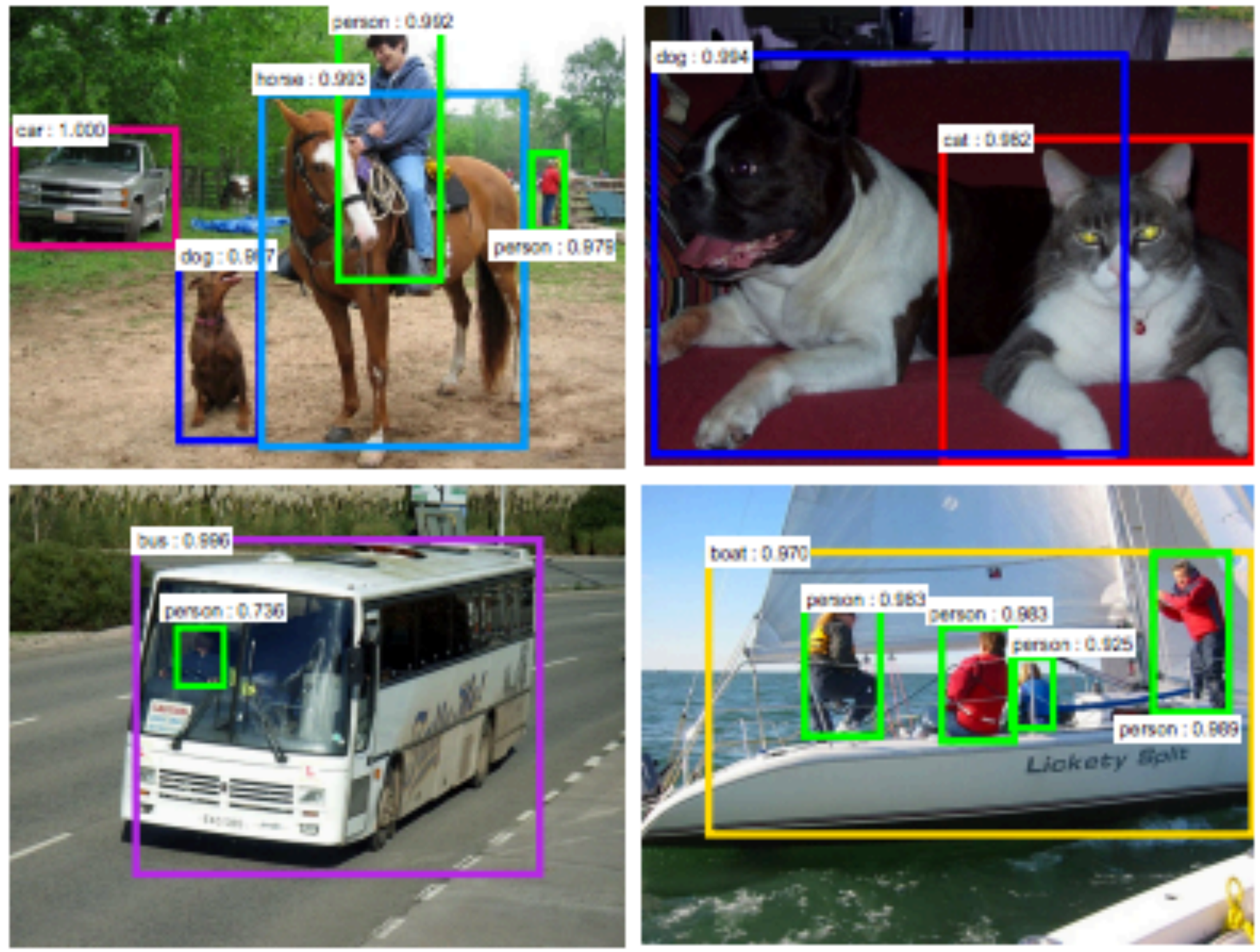


Region-CNN, Girshick et al., 2013



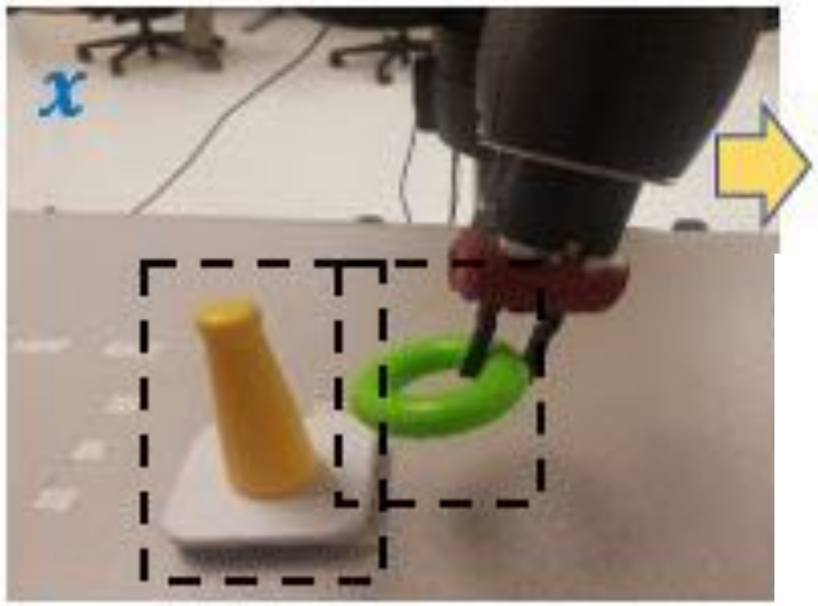
Faster RCNN, Girshick et al., 2014

Object detection for manipulation



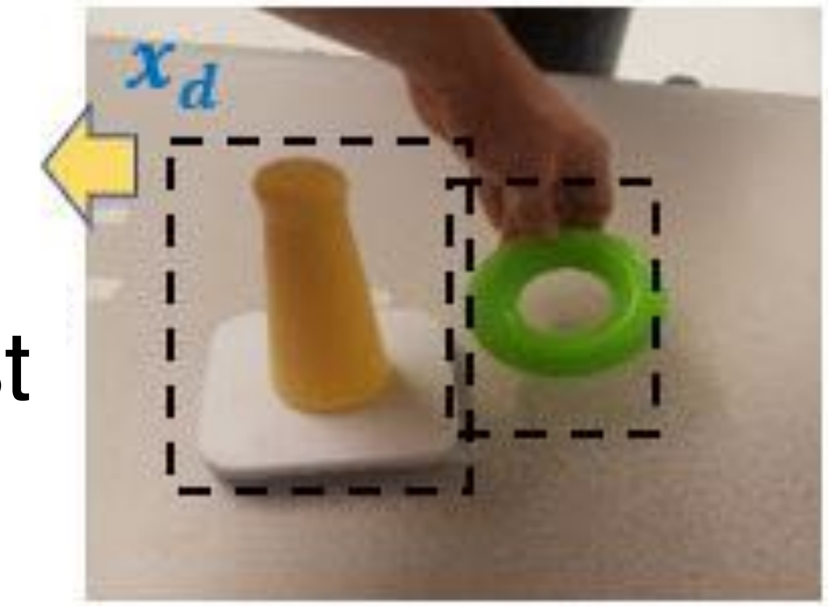
- Object detection works, when someone has labelled the objects we care about.
- How can we use this tools in CNNs for manipulation?

Learning from Visual Demonstrations



$\|x - x_d\|$

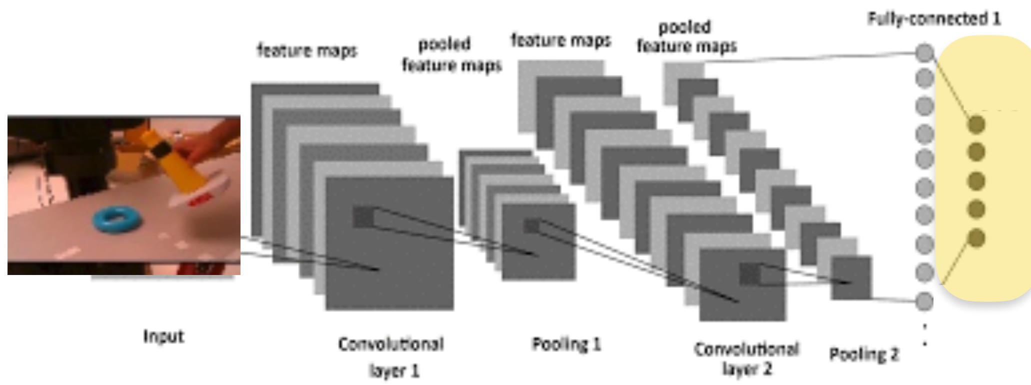
similarity cost



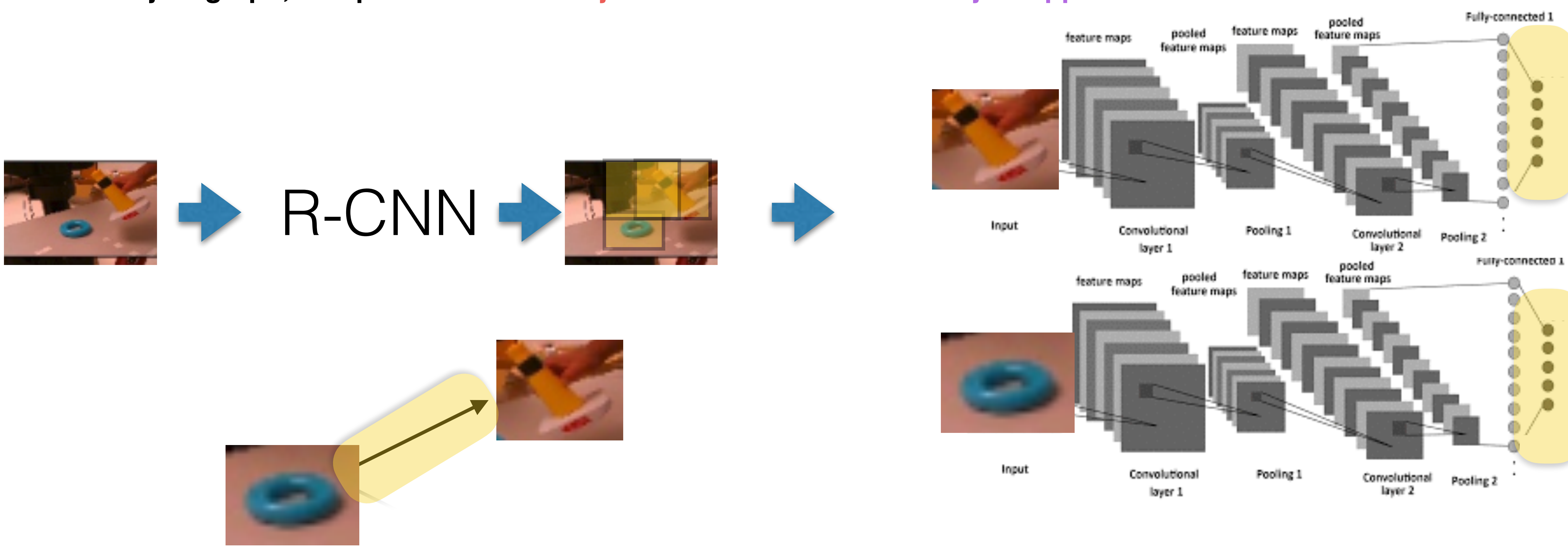
- Learn a function that maps RGB images o_t into state vectors x_t so that Euclidean distances in the state space reflect task progress (reward)

Architectures for deep state representations

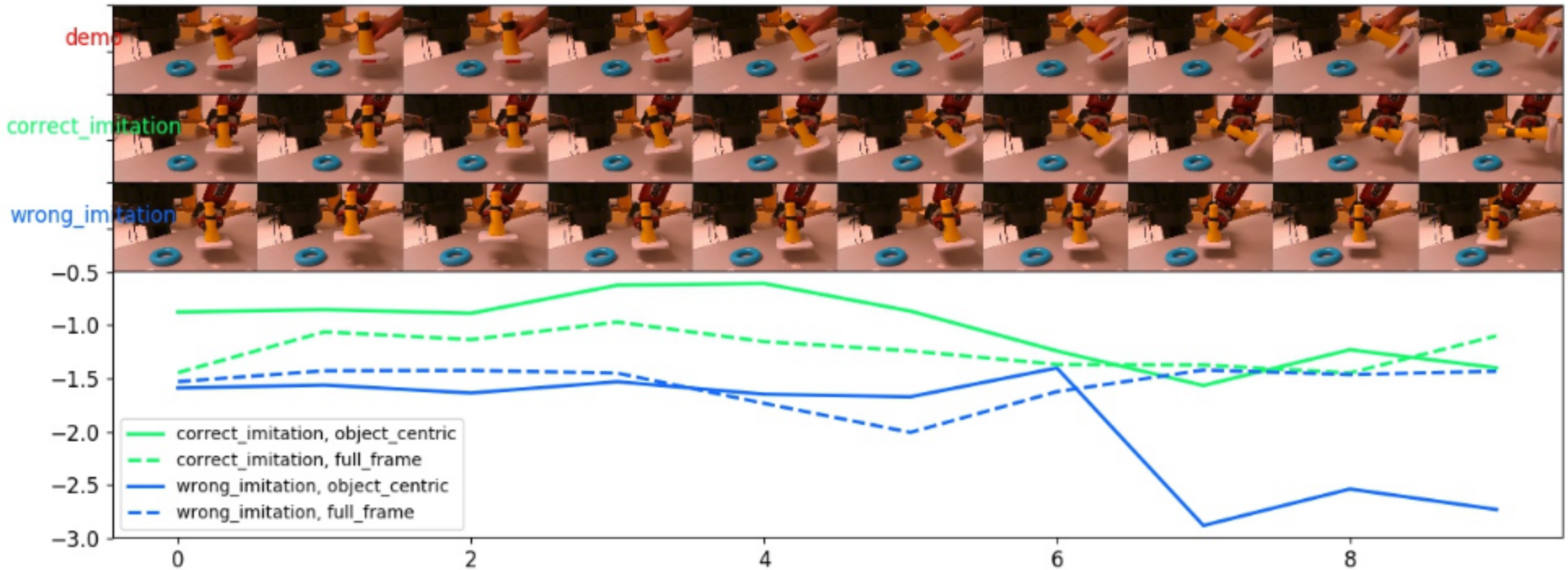
State is a latent vector extracted convolutionally from the full frame



State is an object graph, comprised of cross-object distances and within object appearance features



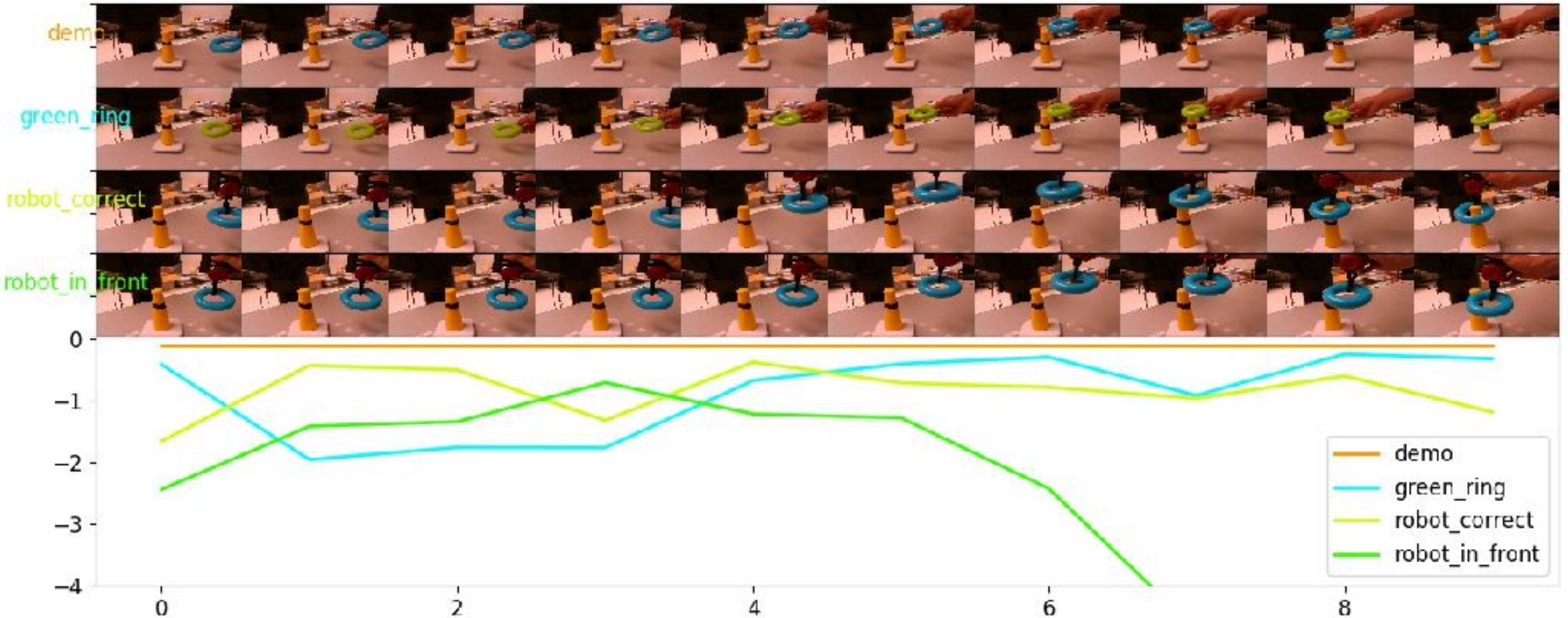
Good reward function



Y axis shows negative exponential of Euclidean distance

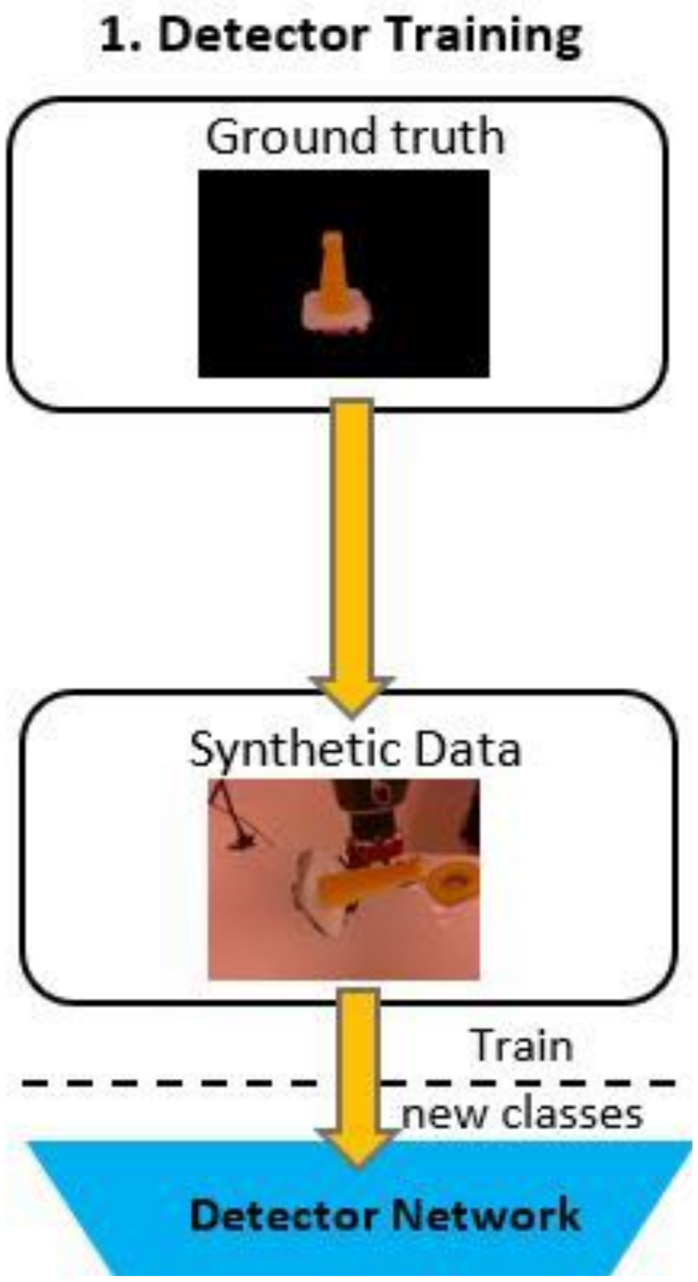
Explicit attention on objects helps

Object-centric Visual Feature Learning

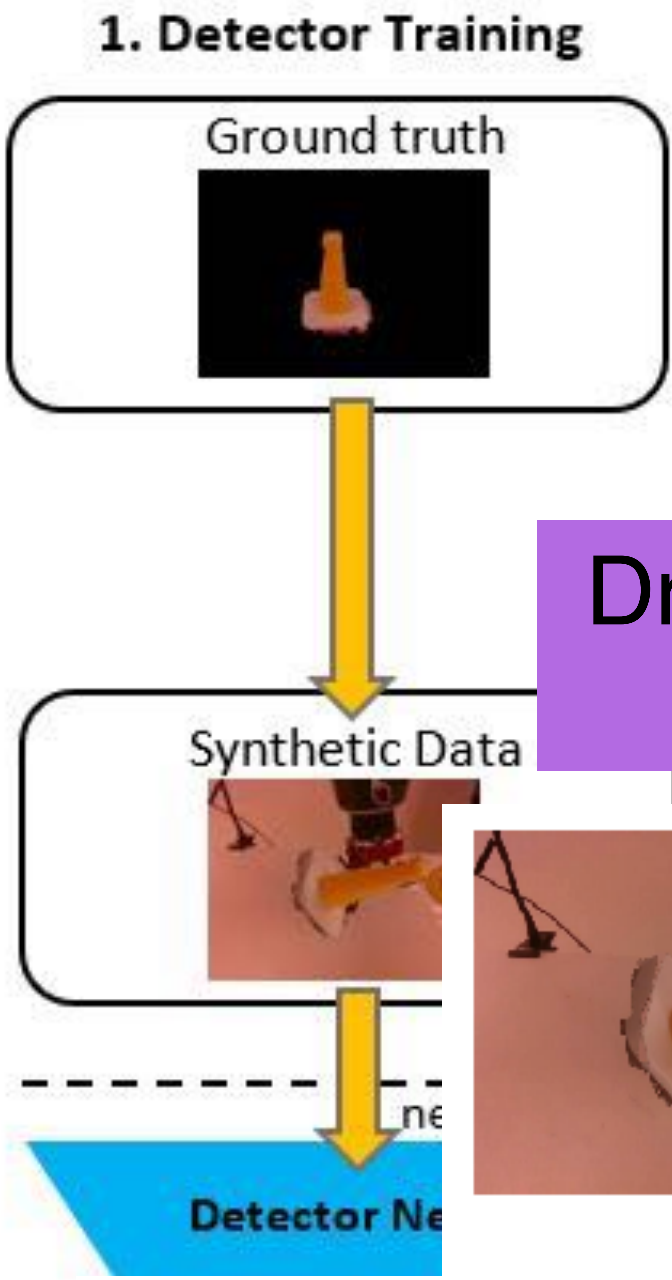


But we do not have labels for our objects! What do we do?

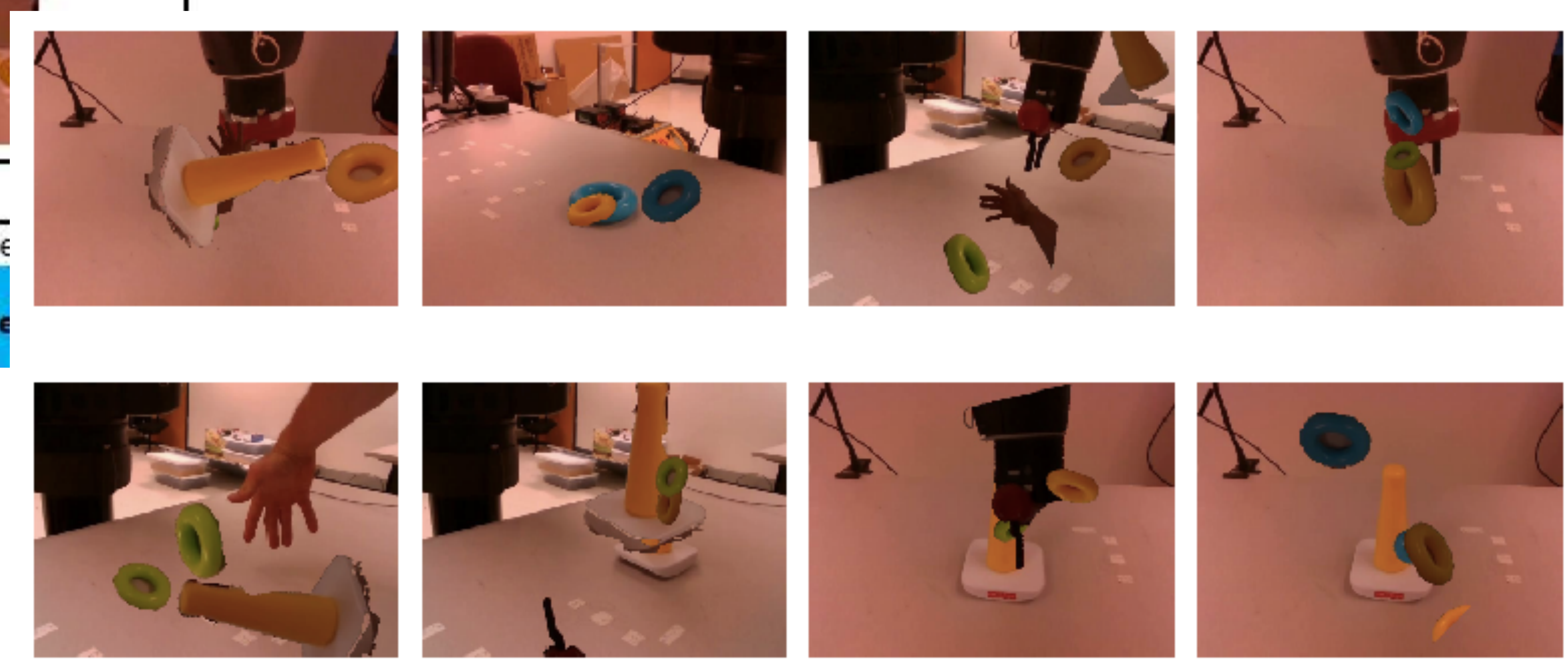
Object-centric Visual Feature Learning



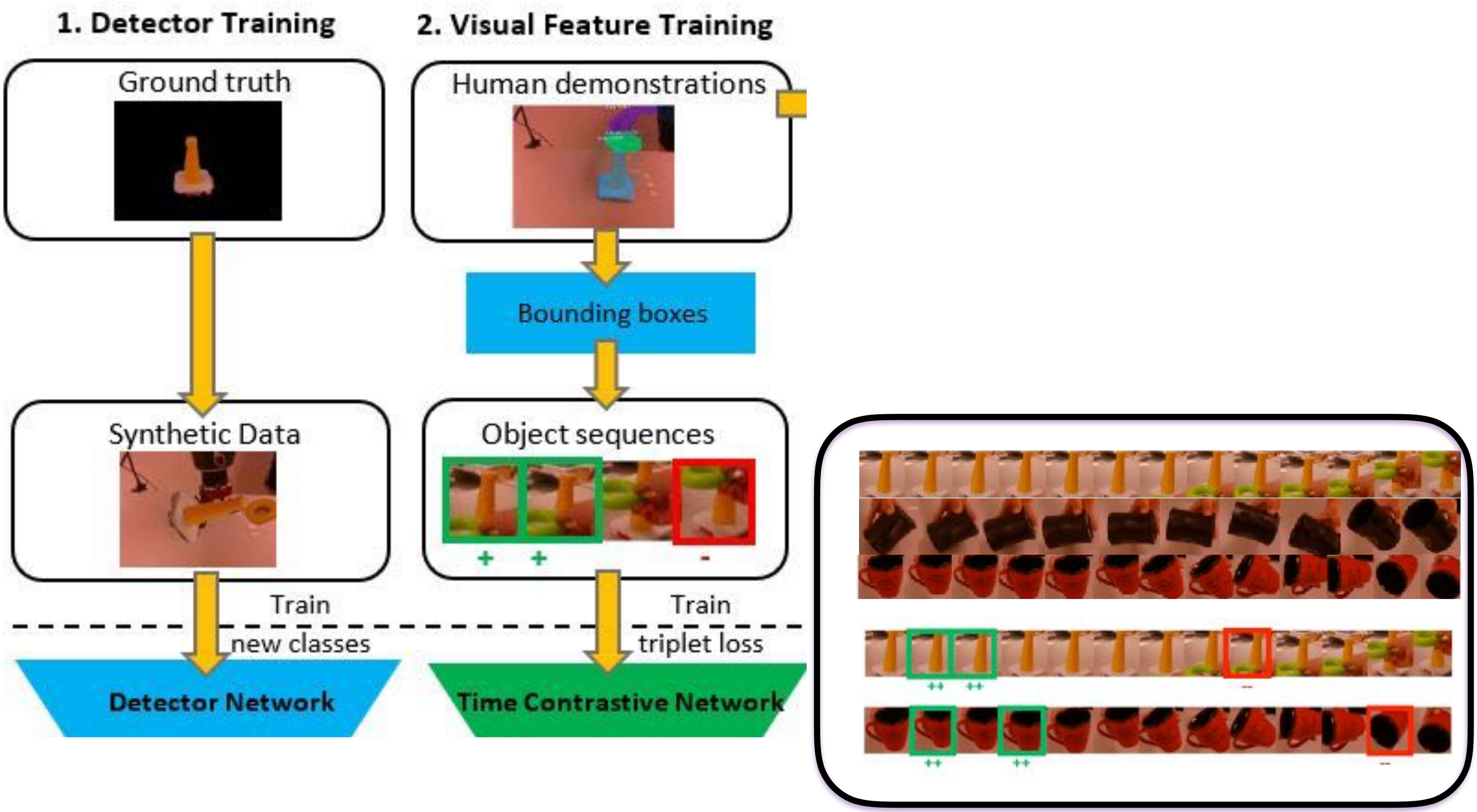
Object-centric Visual Feature Learning



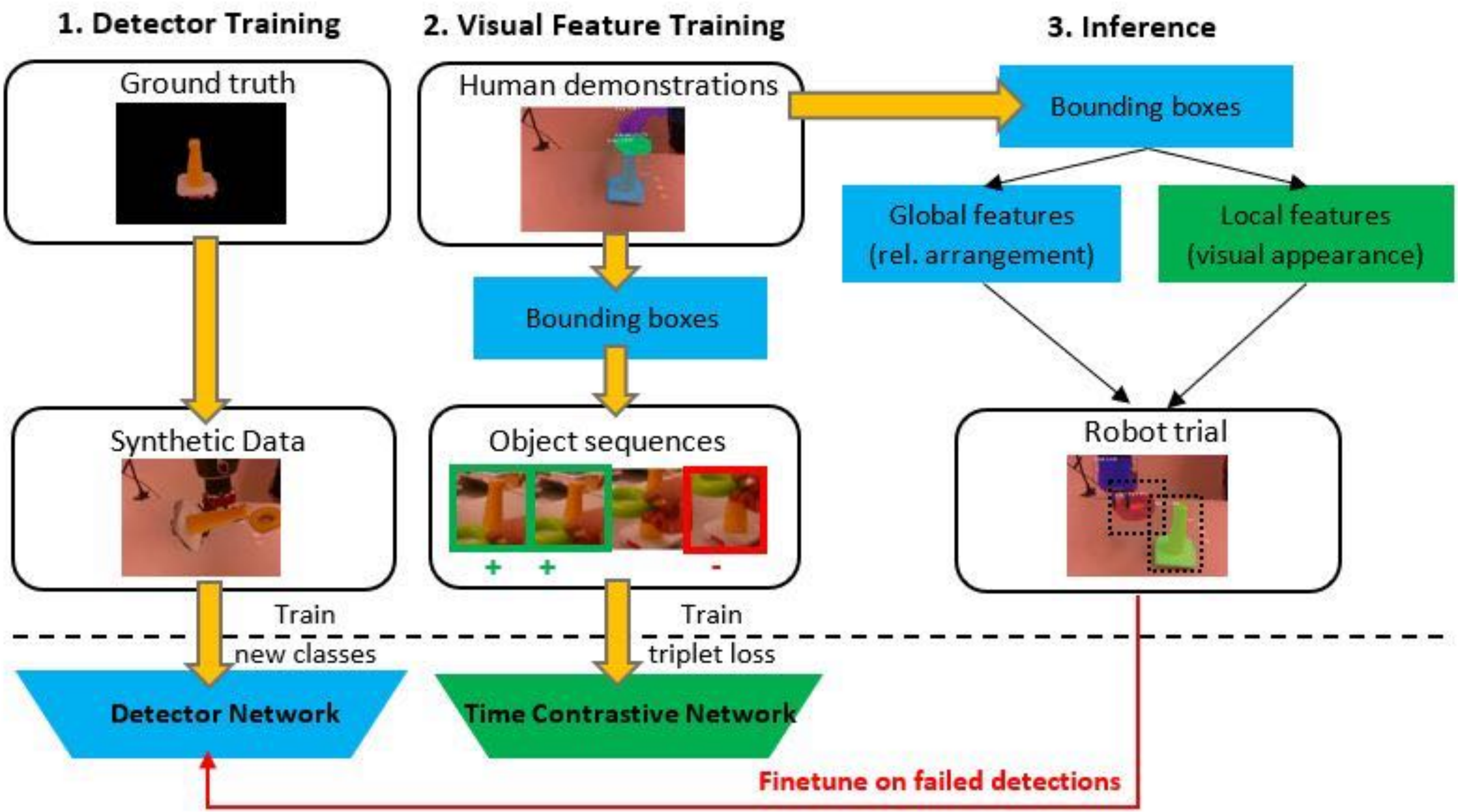
Dreaming data for learning on-the-fly state-of-the-art object detectors!



Object-centric Visual Feature Learning



Object-centric Visual Feature Learning



Object-centric Visual Feature Learning

We use a trajectory optimization method, attempting to minimize visual dissimilarity between execution and demonstration

Given: $\bar{x}_0$

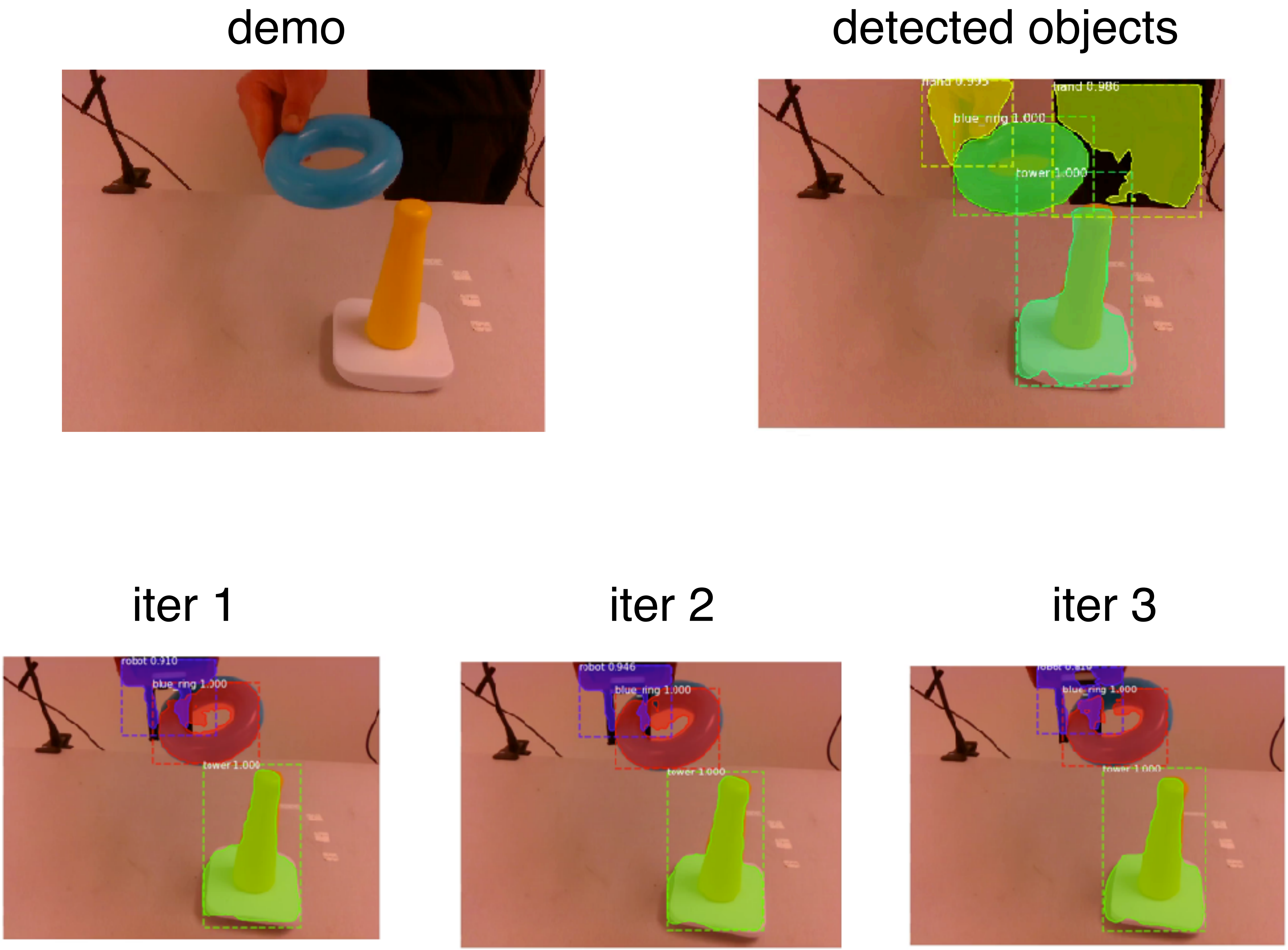
For $t = 0, 1, 2, ..., T$

- Solve

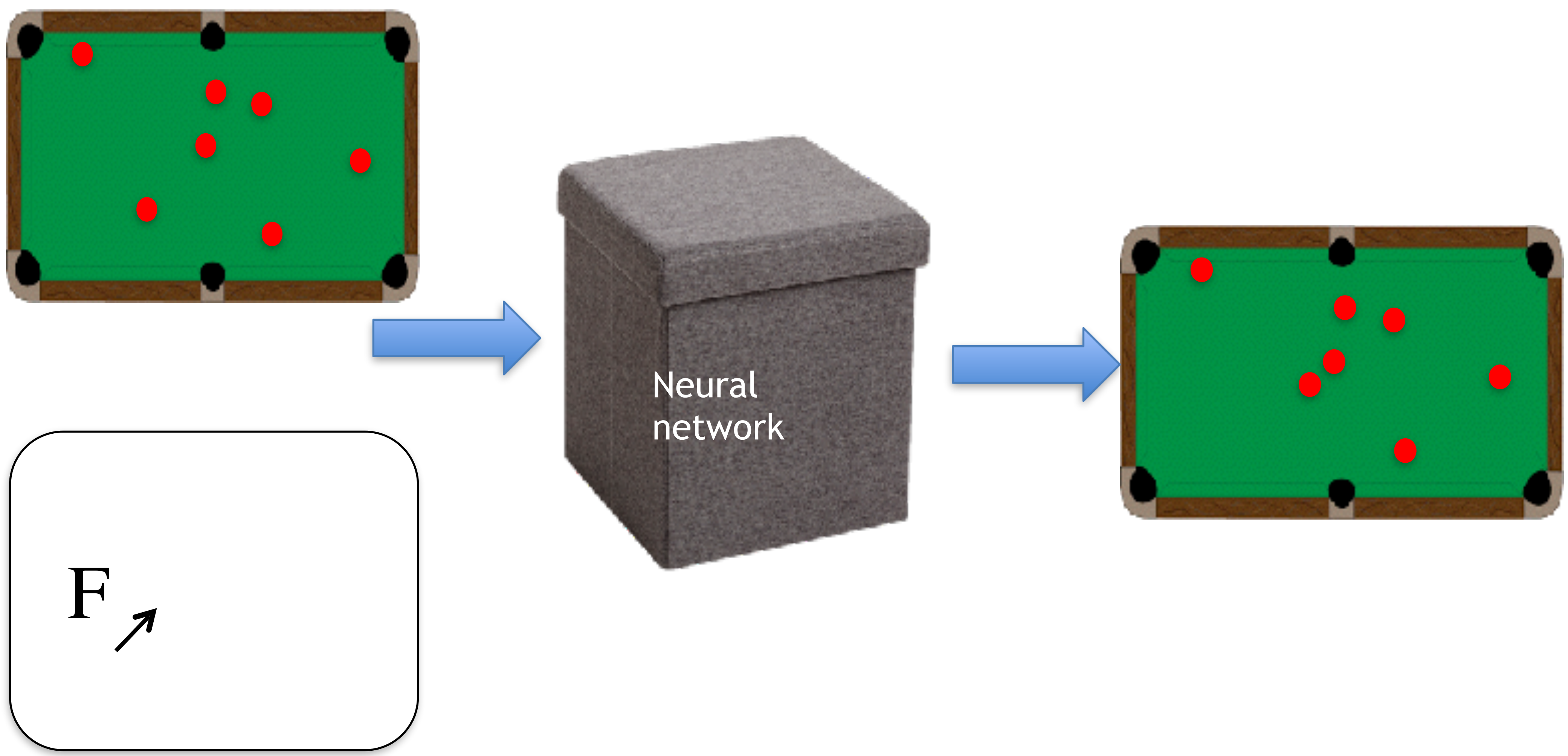
$$\min_{x,u} \sum_{k=t}^{T} \|\mathrm{x}_k - \mathrm{x}_k^d\|$$
$$\text{s.t. } x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, ..., T-1\}$$
$$x_t = \bar{x}_t$$

- Execute u_t
- Observe resulting state, $\bar{x}_{t+1}$
- Initialize with solution from $t-1$ to solve fast at time t

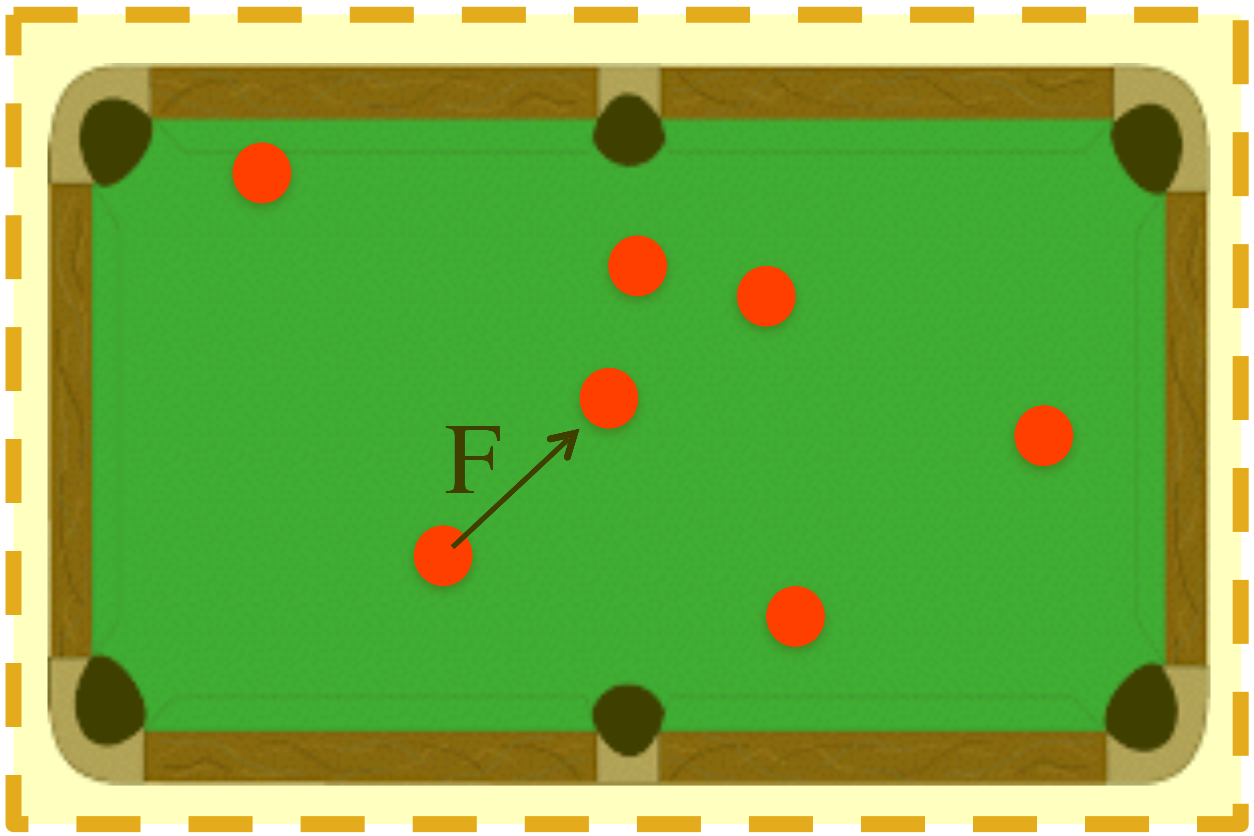
PILQR


demo
detected objects
hand 0.986
blue_ring 1.000
tower 1.000
iter 1
robot 0.910
blue_ring 1.000
tower 1.000
iter 2
robot 0.946
blue_ring 1.000
tower 1.000
iter 3
blue_ring 1.000
tower 1.000

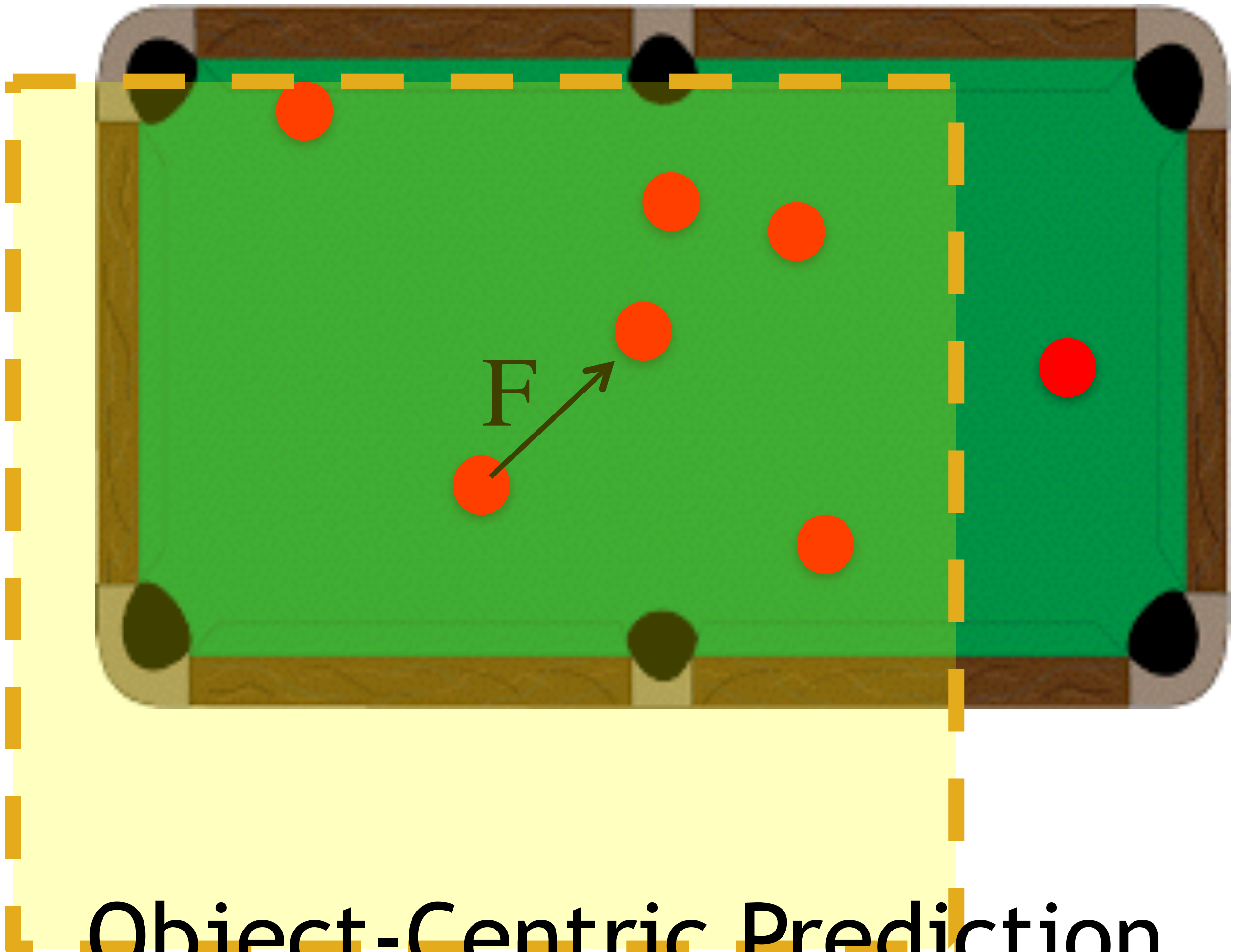
Learning motor conditioned Dynamics



Object-centric prediction

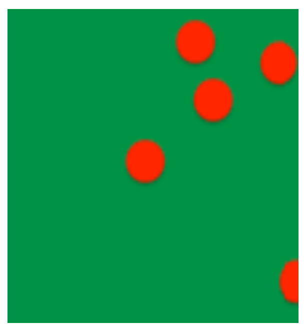


World-Centric Prediction

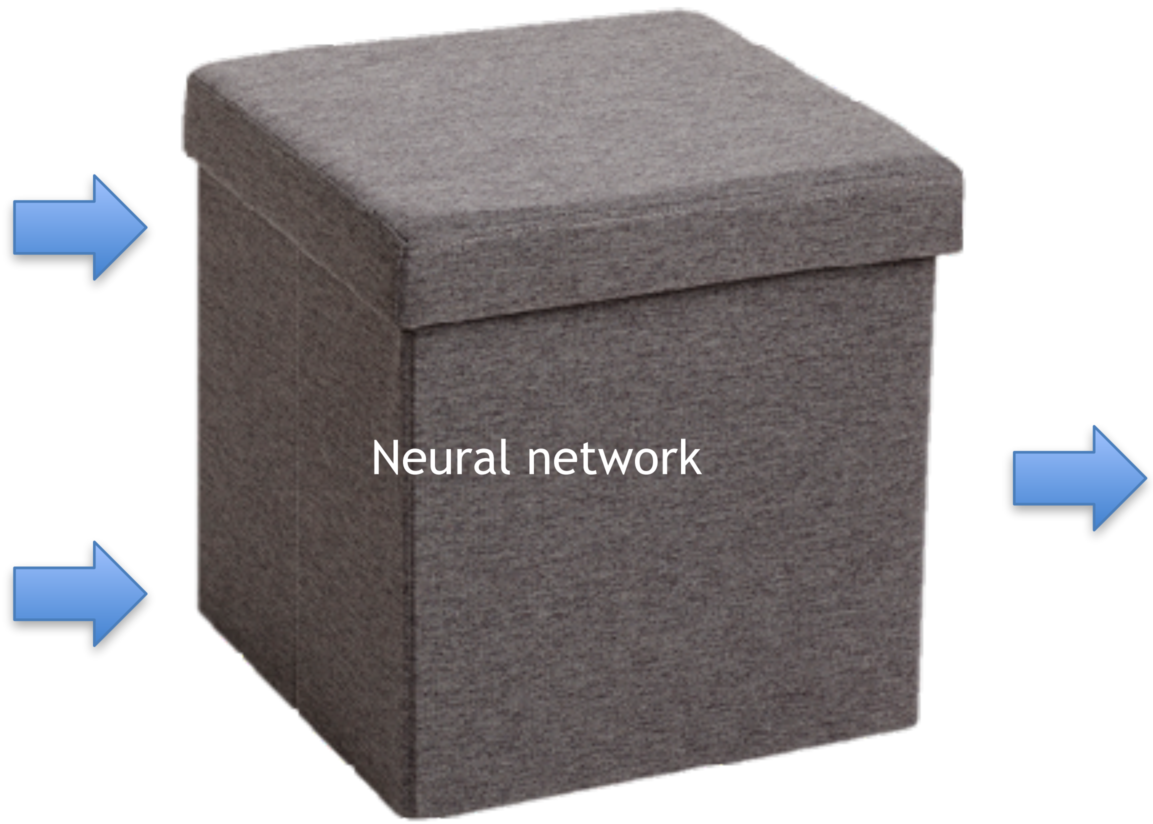


Object-Centric Prediction

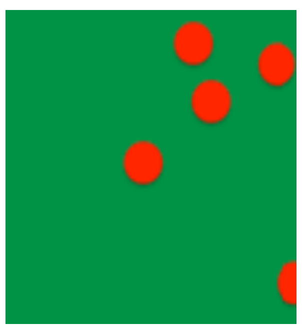
Object-centric predictions



$$\mathbf{F}_1 \cdots \mathbf{F}_t$$

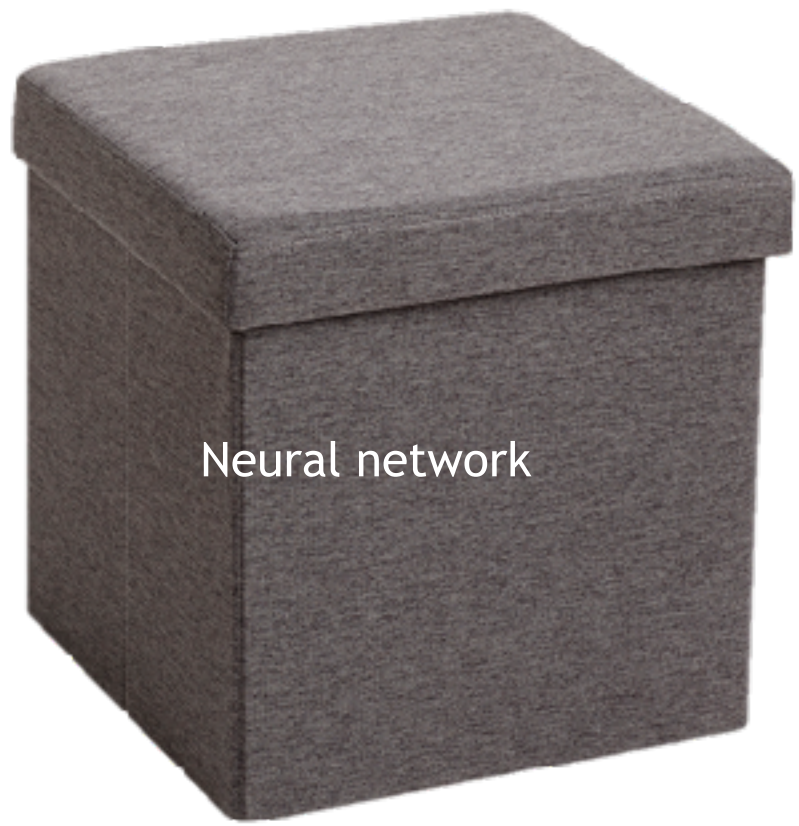


Training Time



$$\mathbf{F}_1 \cdots \mathbf{F}_t$$

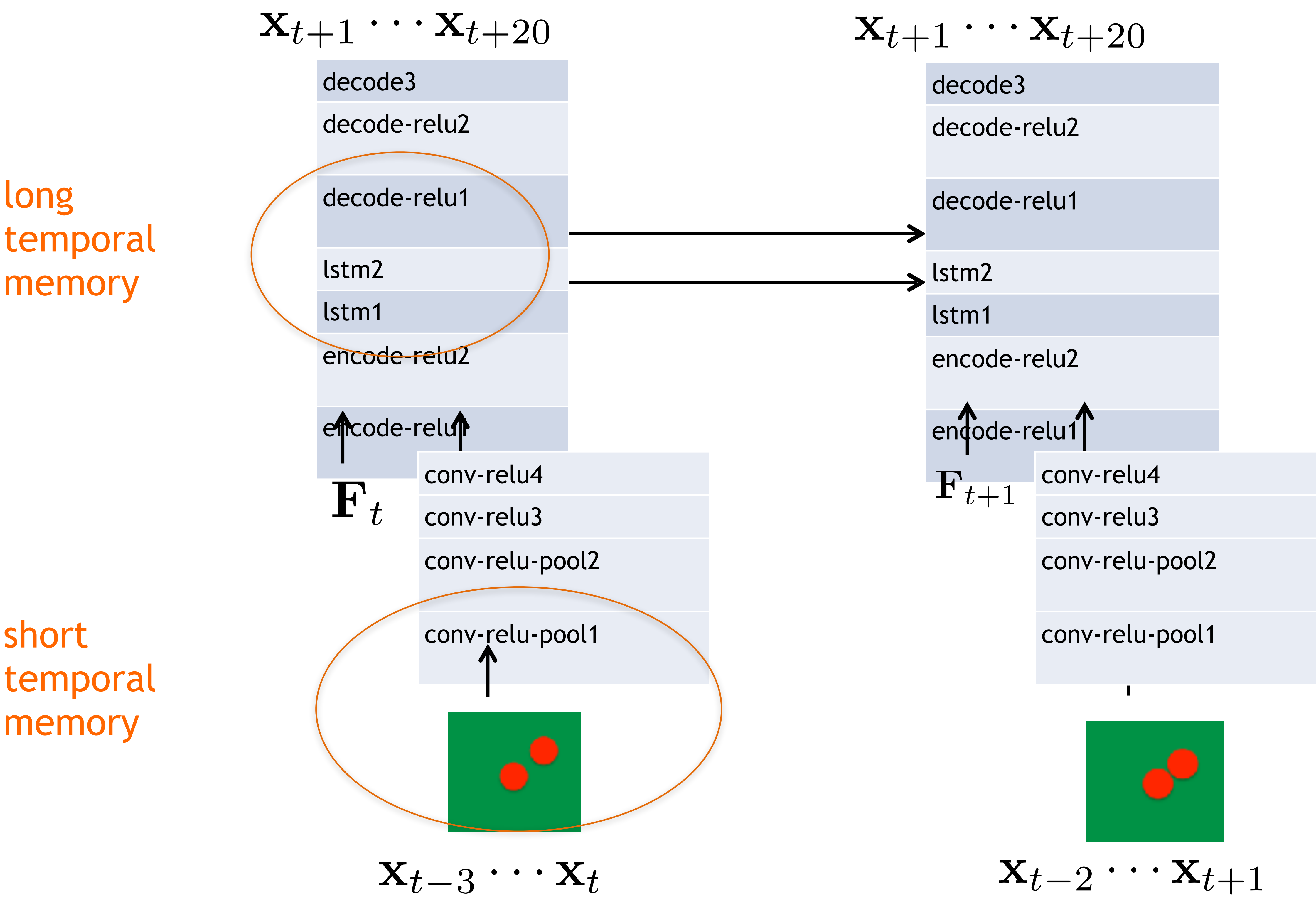
Forces applied to the centered ball



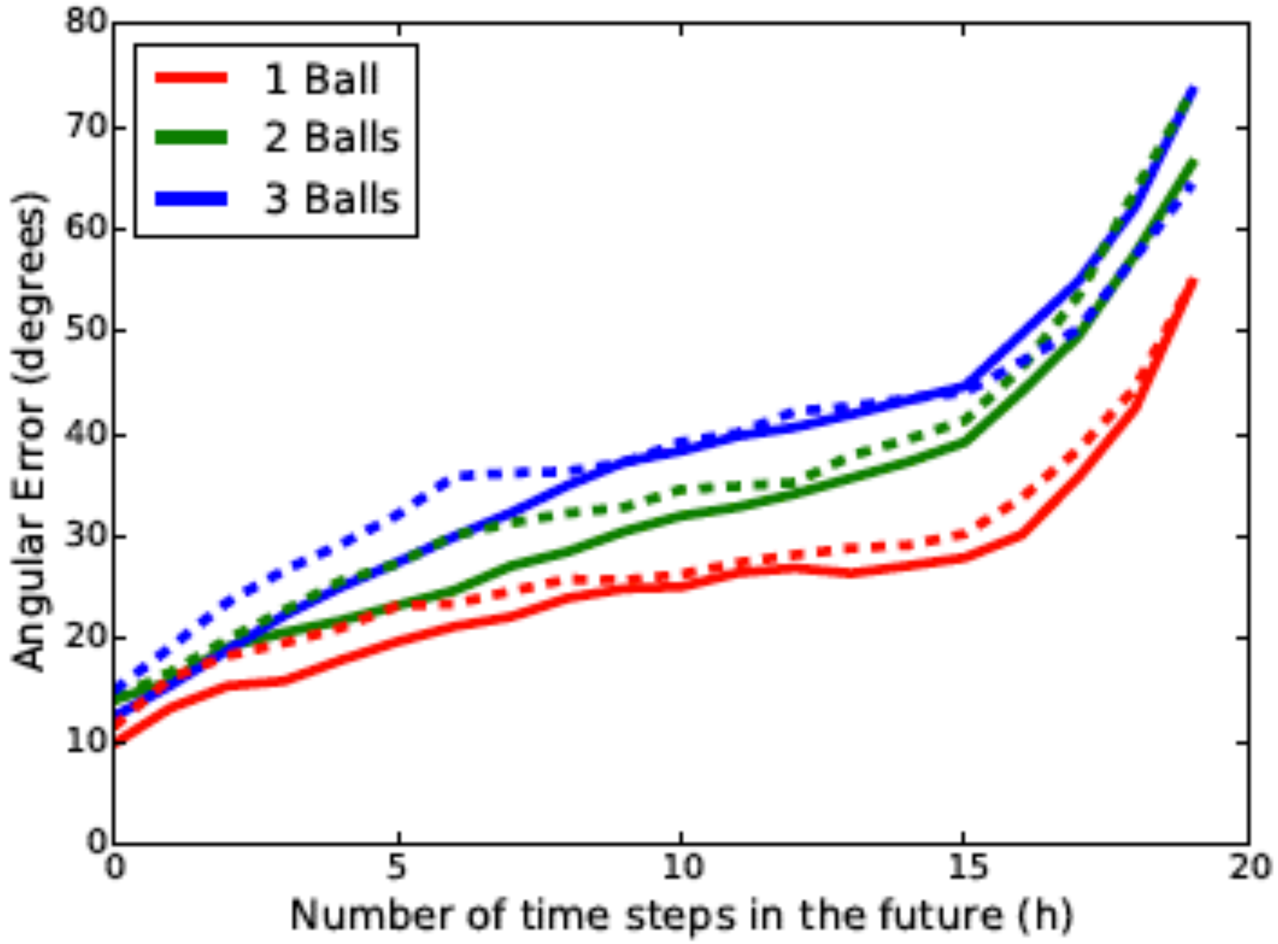
$$\mathbf{x}_{t+1} \cdots \mathbf{x}_{t+20}$$

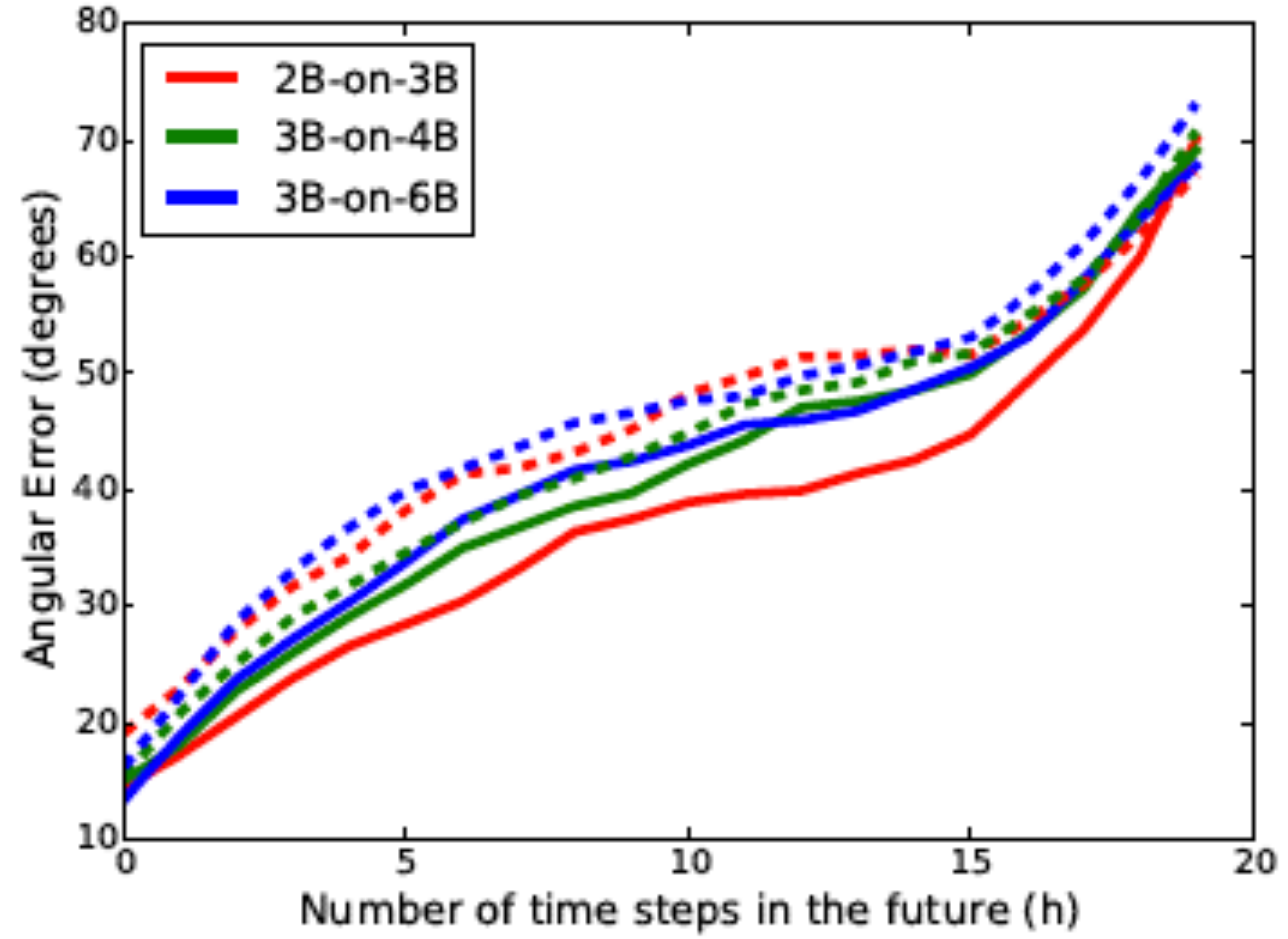
Displacements of the centered ball in the next 20 frames

Network Architecture



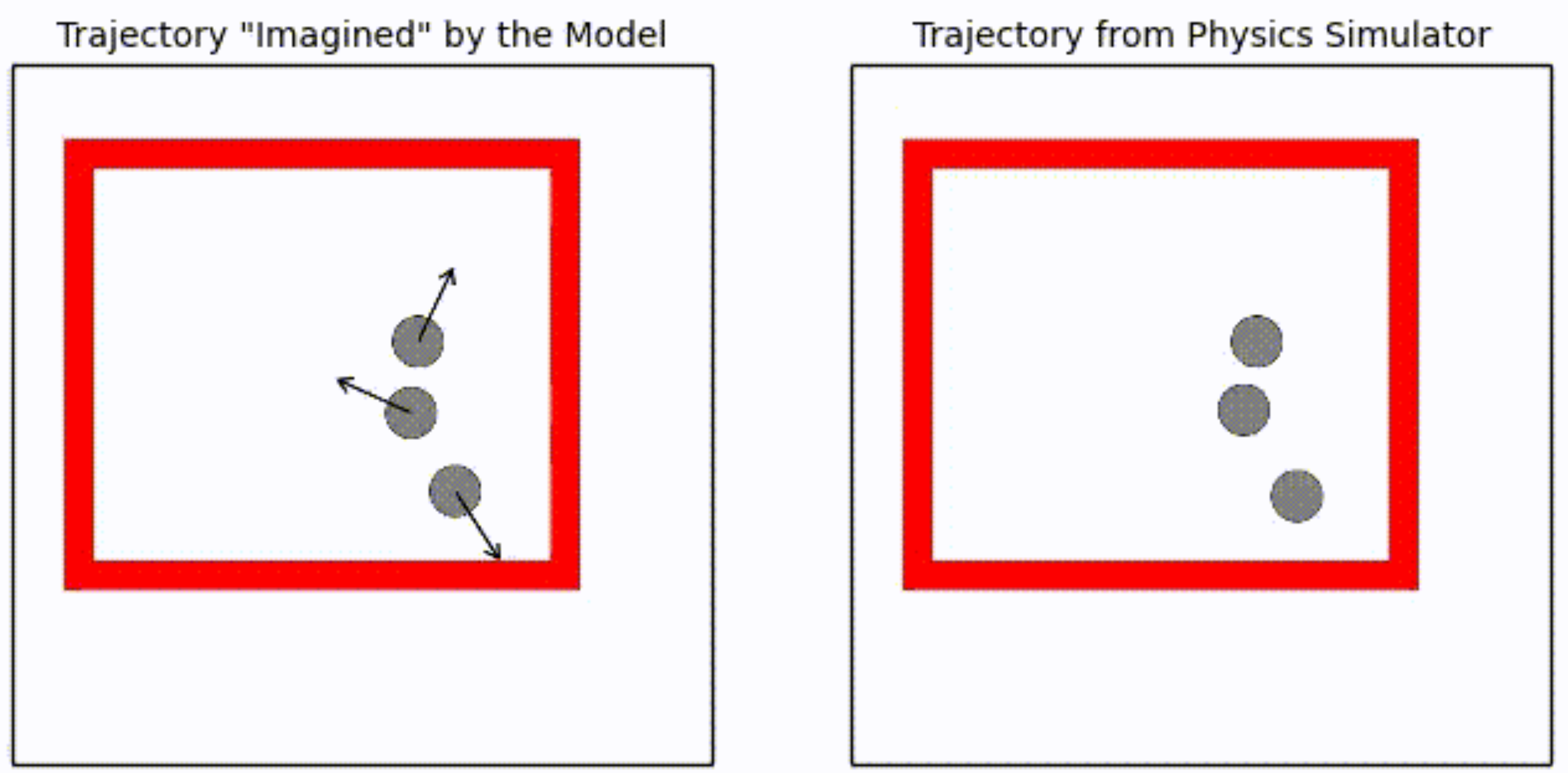
Generalization



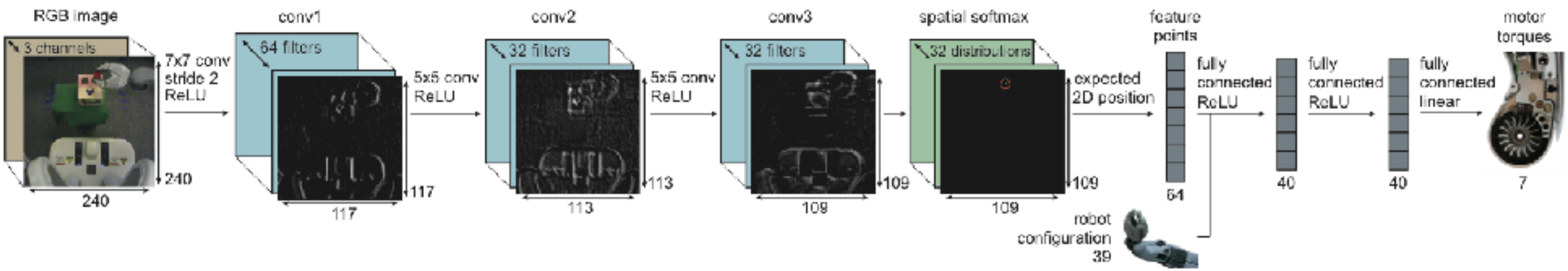


object-cetric: solid
frame-centric: dashed

Visual Imaginations

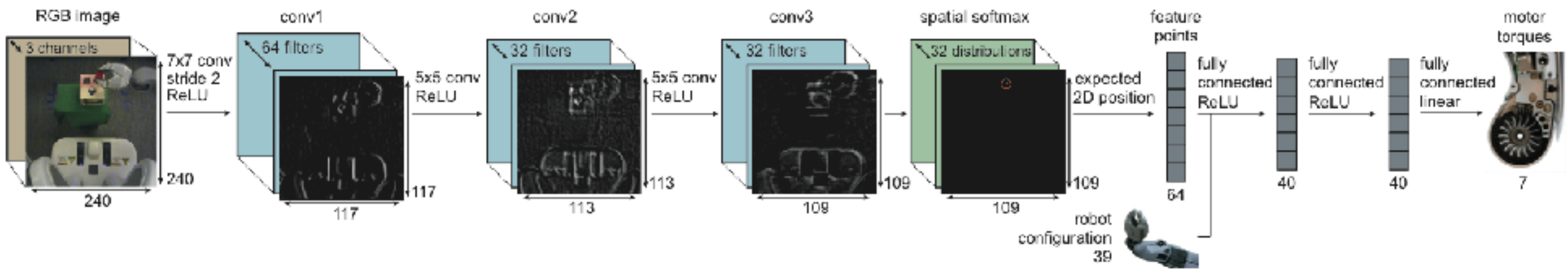


Spatial Softmax



End-to-end learning of visuomotor policies, Levine et al. 2015

Spatial Softmax

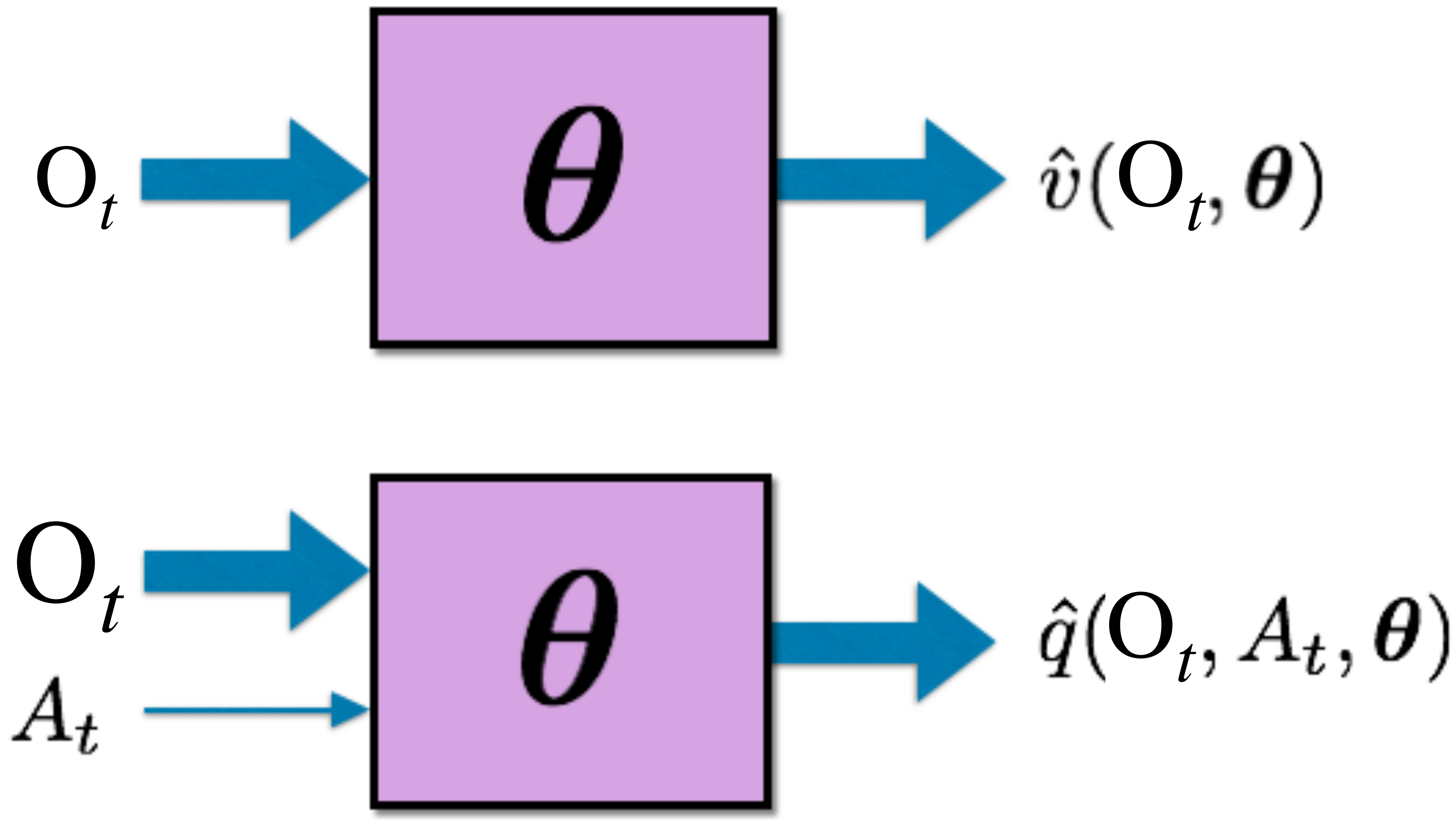


End-to-end learning of visuomotor policies, Levine et al. 2015

- For each feature map, ``flatten” it and compute a softmax
- Then take X and Y grid coordinates and compute the corresponding weighted averages
- Imposes a very tight bottleneck and avoids overfitting

End-to-End RL

- End-to-end RL methods replace the hand-designed state representation with raw observations.



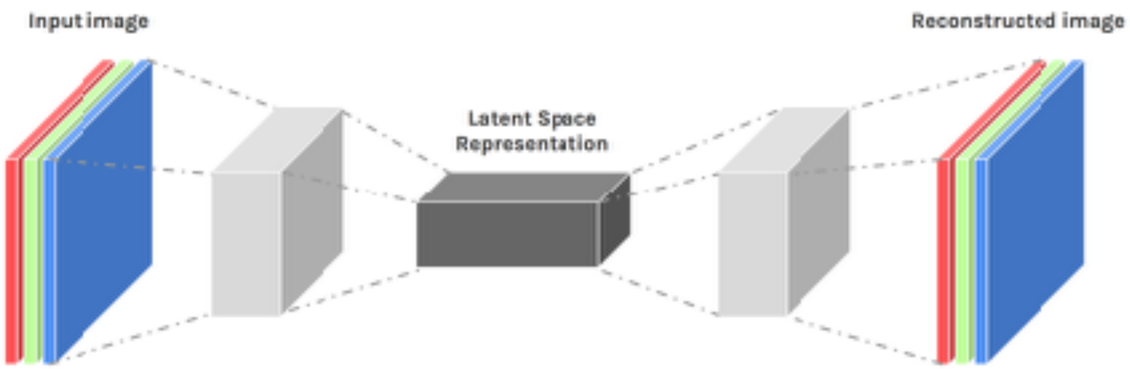
- We get rid of manual design of state representations :-)
- We need tons of data to train the network since O_t usually WAY more high dimensional than hand-designed S_t :-(
- We can pre-train or jointly train with additional losses (auxiliary tasks) :-) For example?

Unsupervised Losses / Pretraining

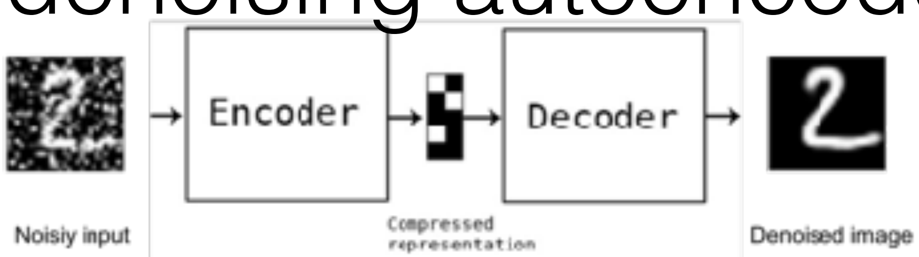
- We can always fine-tune from weights trained on a supervised visual task.
- We can use auxiliary tasks, e.g., autoencoders
- We can use prediction of griper key points (we know where they are using forward kinematics and camera calibration
- We can use inverse model learning

Autoencoders

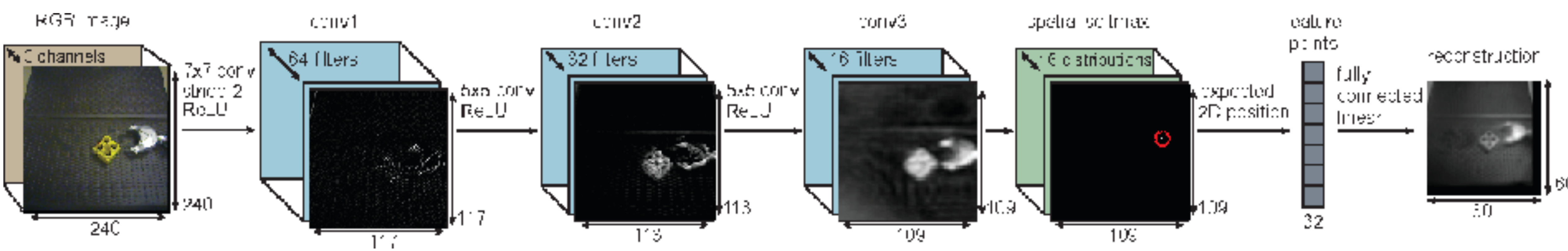
autoencoder



denoising autoencoder



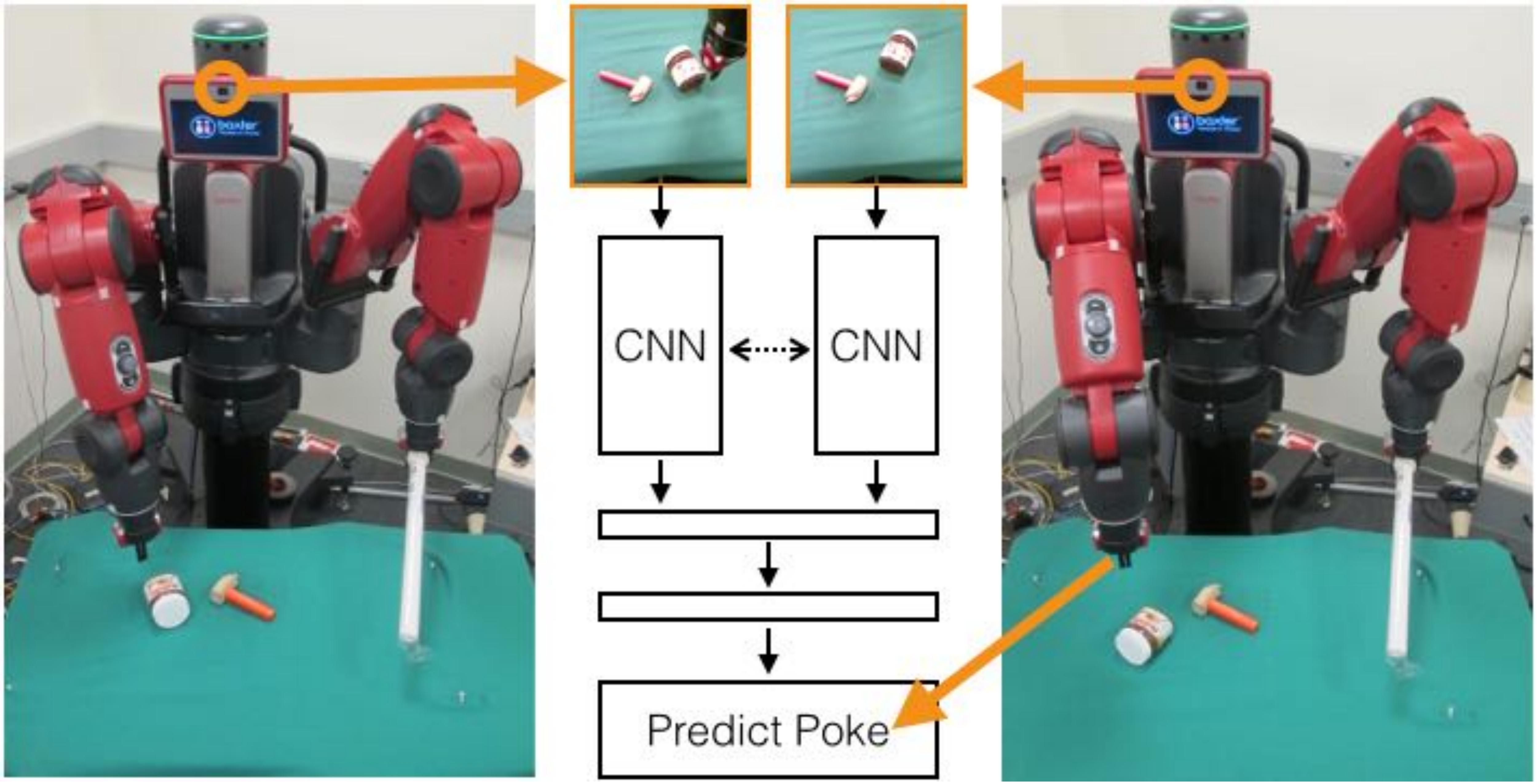
what/where autoencoder



Autoencoders are trained to reconstruct the input (e.g., L2 pixel loss) after they pass through a tight bottleneck layer (the state representation)

What can go wrong?

Train to predict the robotic action



Learning to poke by poking, Agrawal et al., 2015