Deep Learning

370: Intro to Computer Vision

Subhransu Maji

April 24, 2025

College of INFORMATION AND COMPUTER SCIENCES
UMASS AMHERST

Motivation

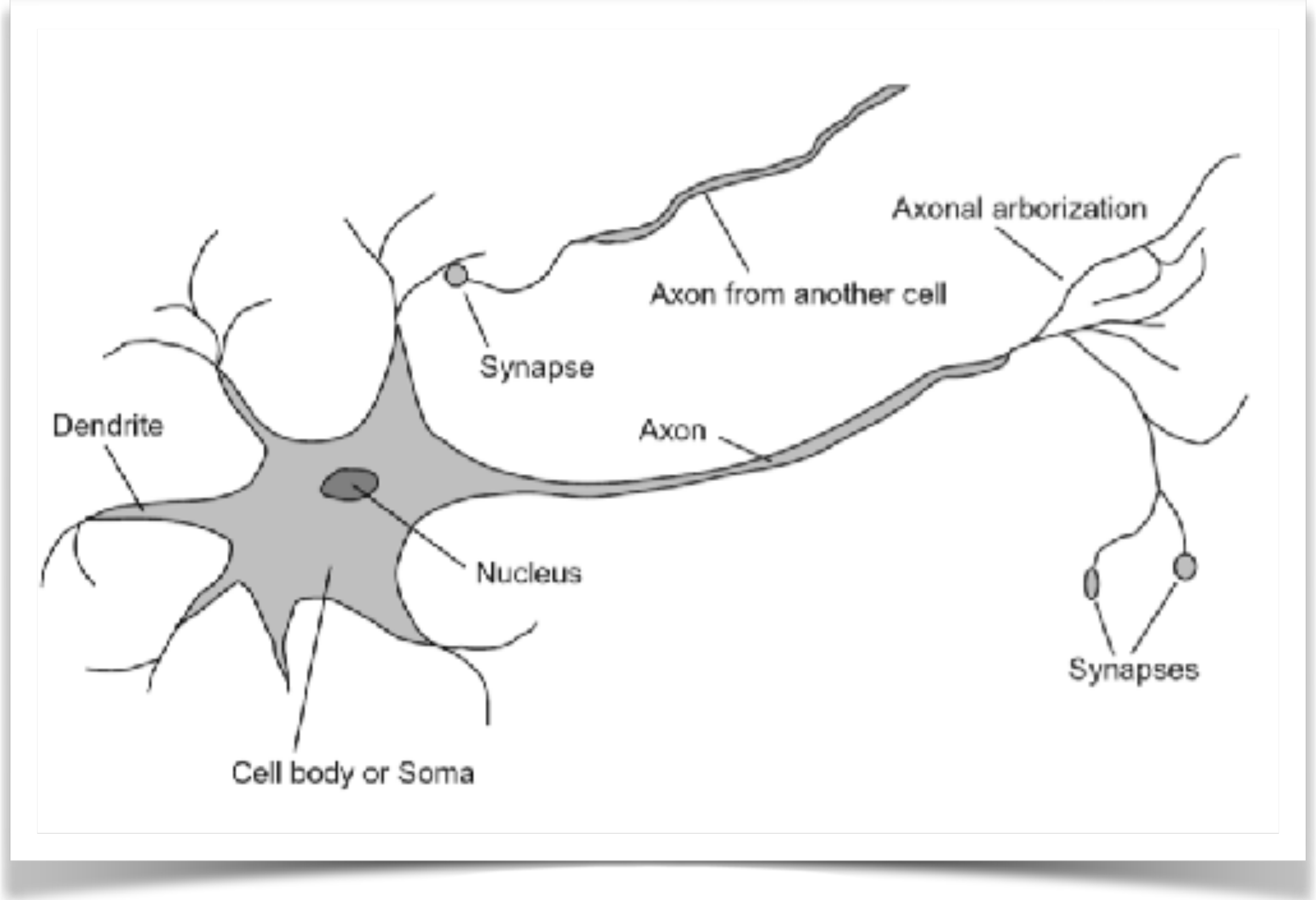
A weakness of linear models is that they are linear

- Nearest neighbor, decision trees, kernel SVMs can model non-linear boundaries
- Neural networks are yet another non-linear classifier

Takes the biological inspiration further by chaining together perceptrons

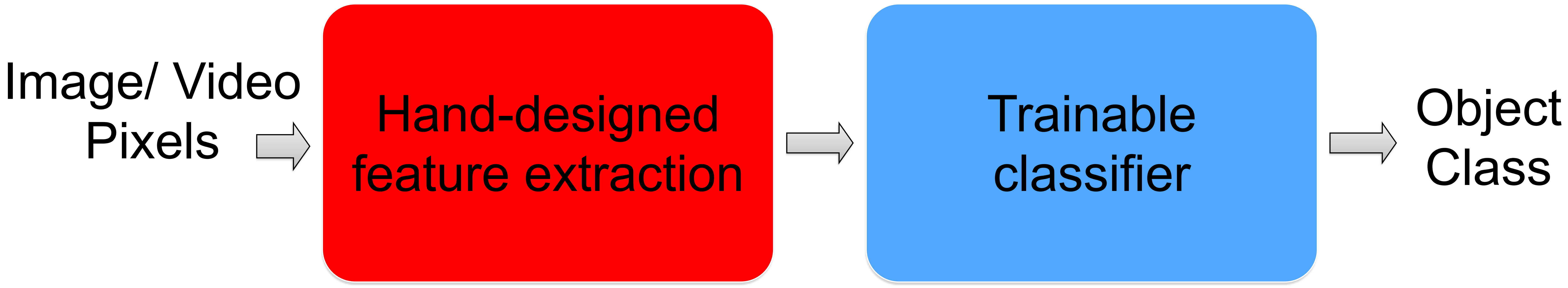
Allows us to use what we learned about linear models:

- Loss functions, regularization, optimization



neuron

Traditional recognition approach



- Features are not learned
- Trainable classifier is often generic (e.g. SVM)

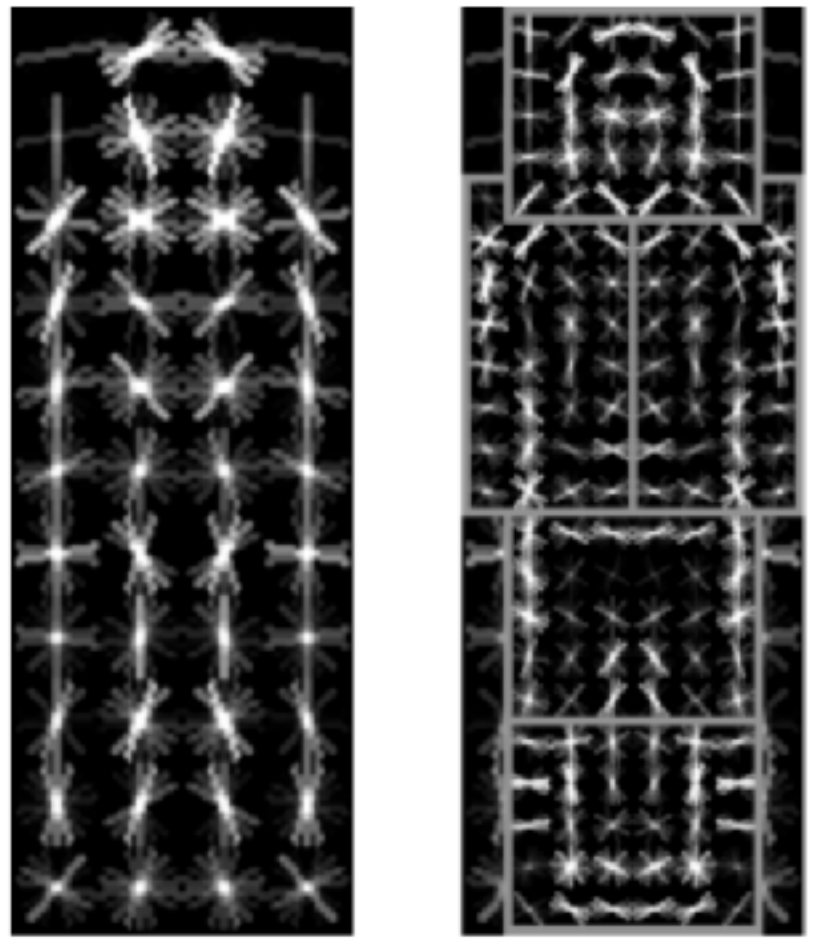
Traditional recognition approach

Circa 2010: Features have played a key role in recognition

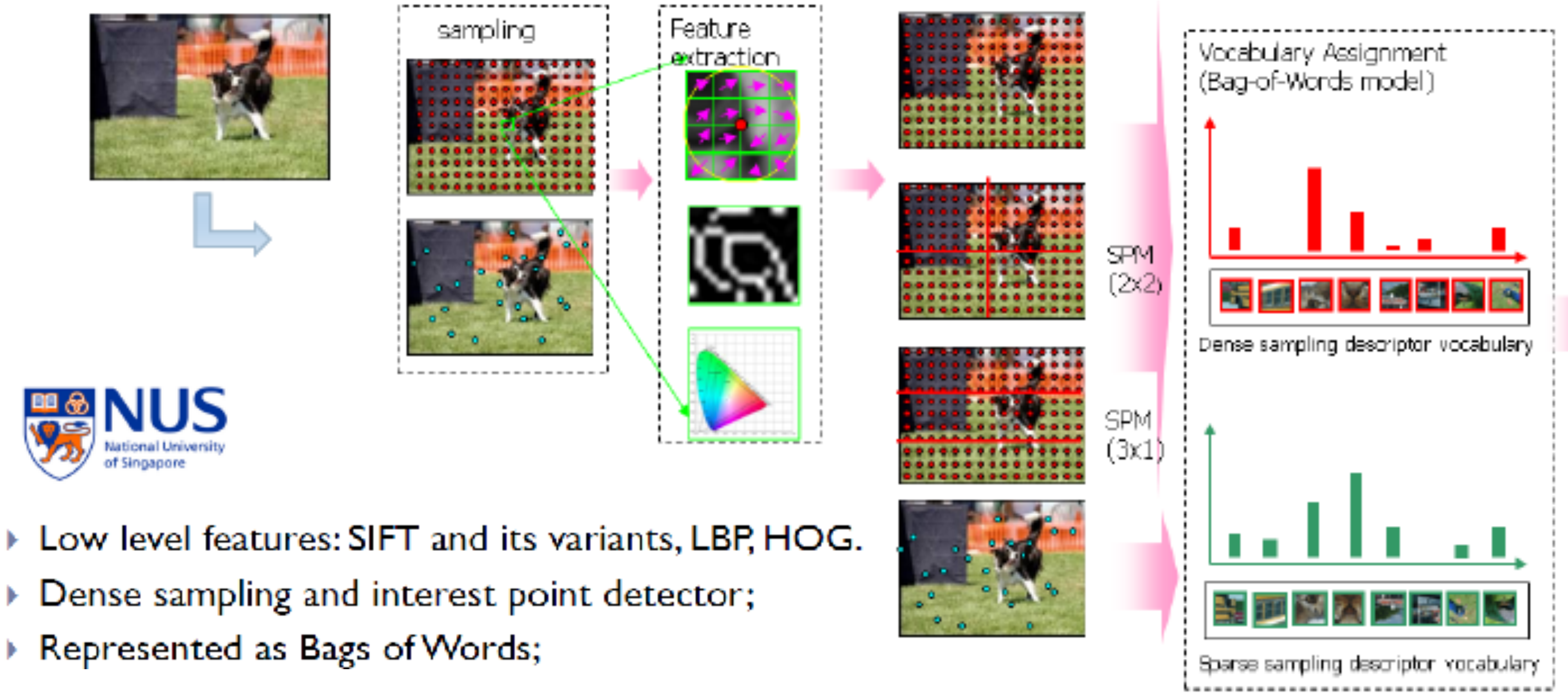
Multitude of hand-designed features currently in use

- SIFT, HOG, ………….

Where next? Better classifiers? Or keep building more features?



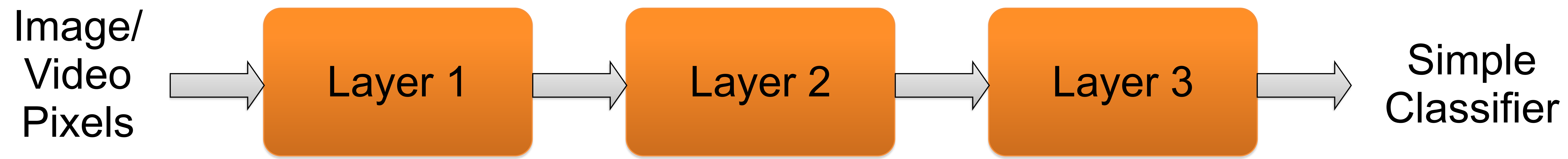
Felzenszwalb, Girshick, McAllester and Ramanan, PAMI 2007



Yan & Huang
(Winner of PASCAL 2010 classification competition)

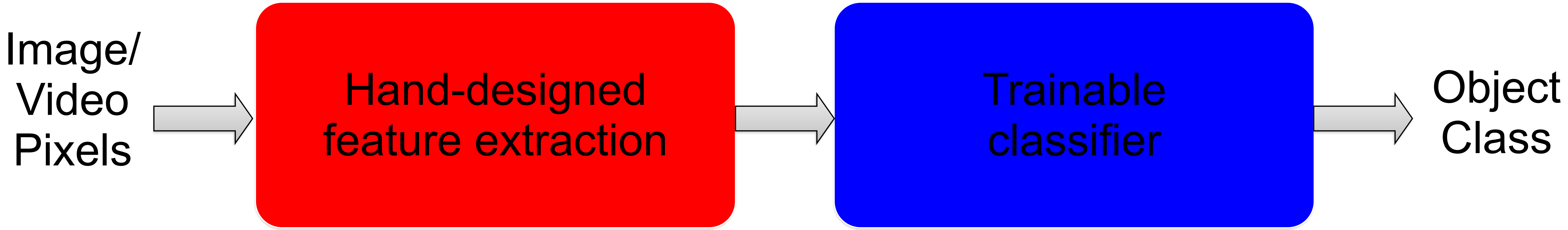
What about learning the features?

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

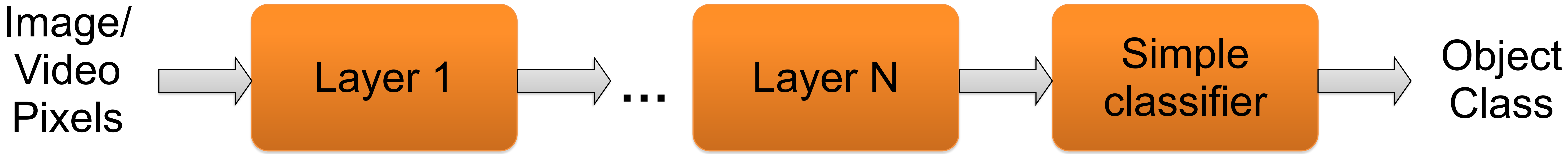


“Shallow” vs. “deep” architectures

Traditional recognition: “Shallow” architecture



Deep learning: “Deep” architecture



Neural Networks

Neural networks: the original linear classifier

(**Before**) Linear score function: $f = Wx$

$$x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D}$$

Neural networks: the original linear classifier

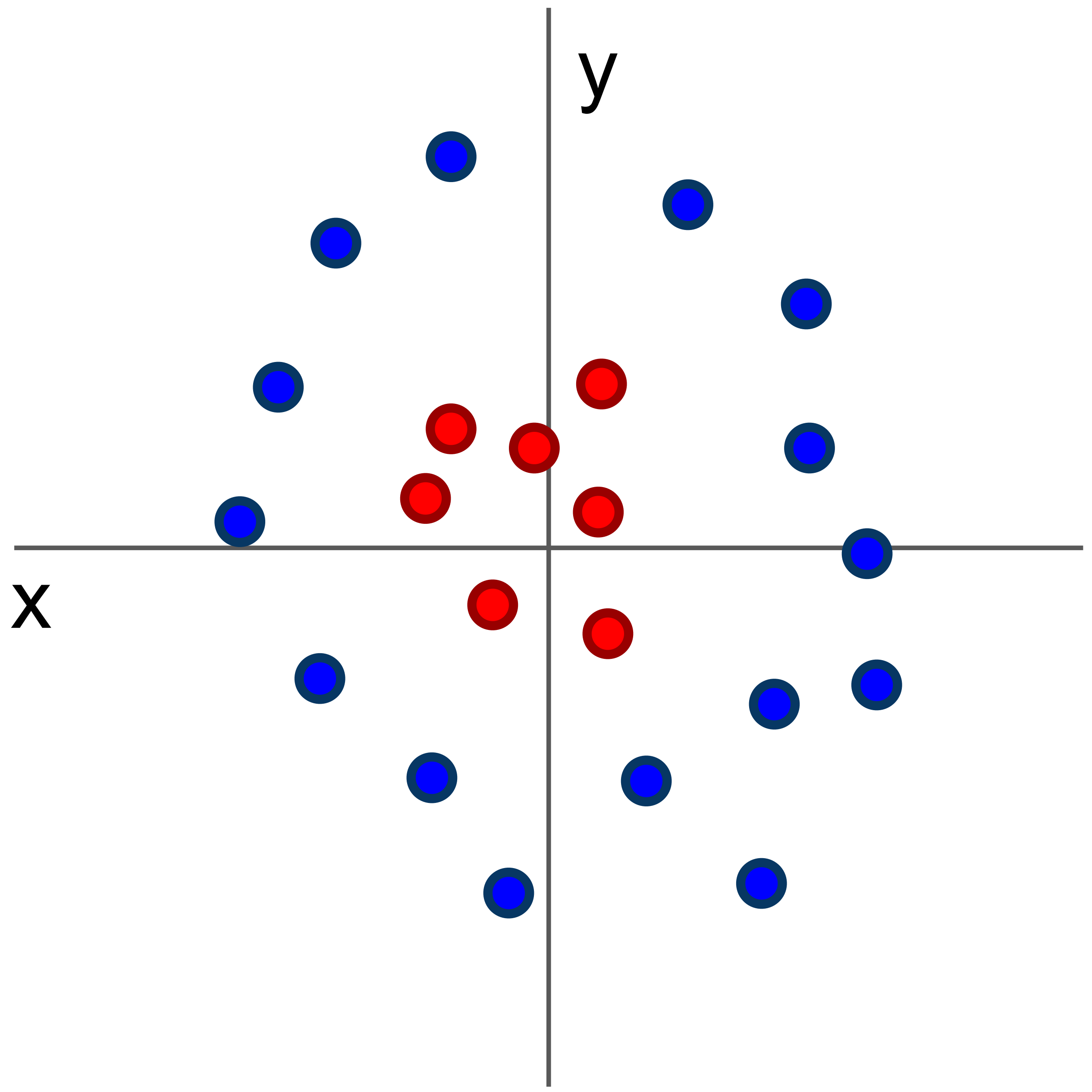
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

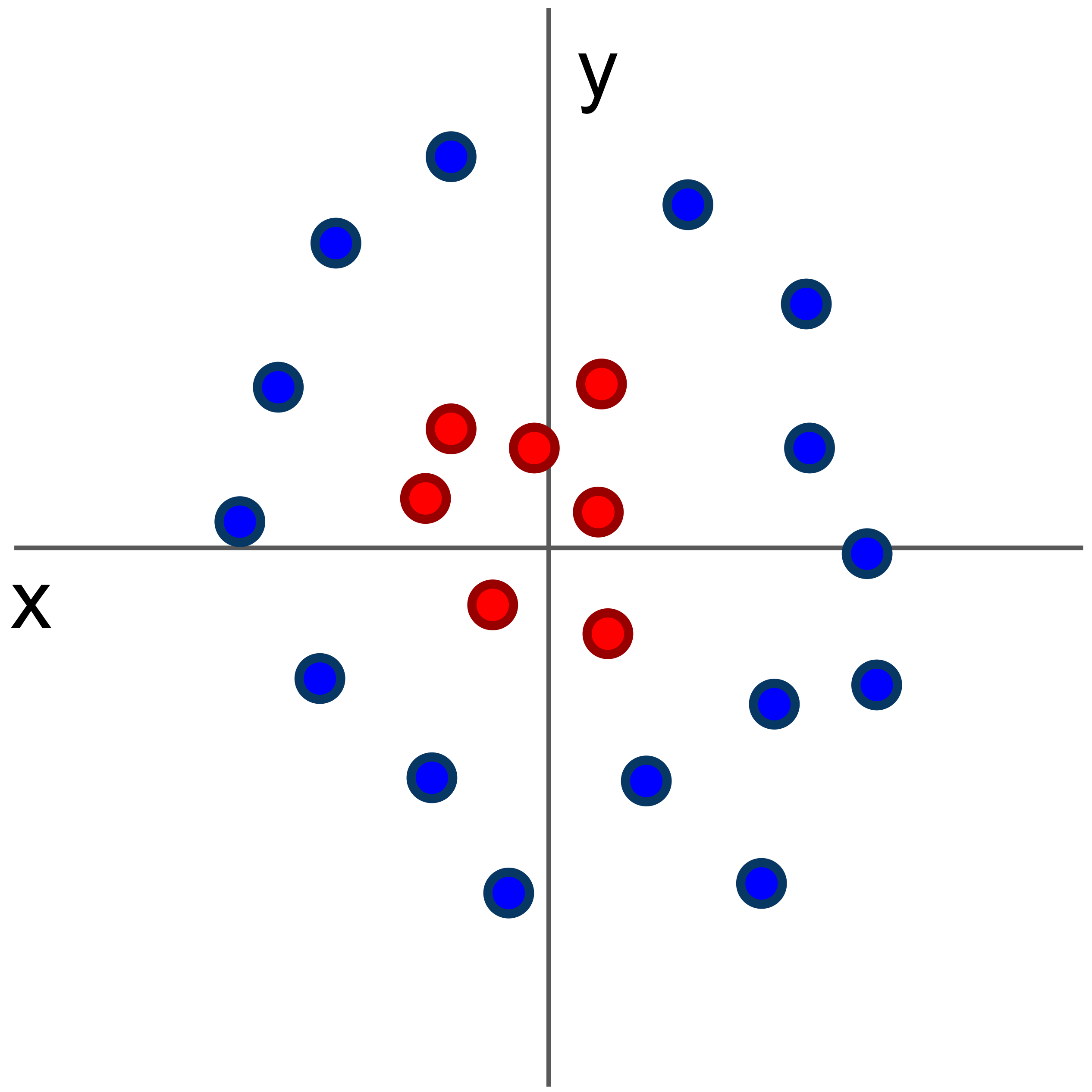
(In practice we will usually add a learnable bias at each layer as well)

Why do we want non-linearity?



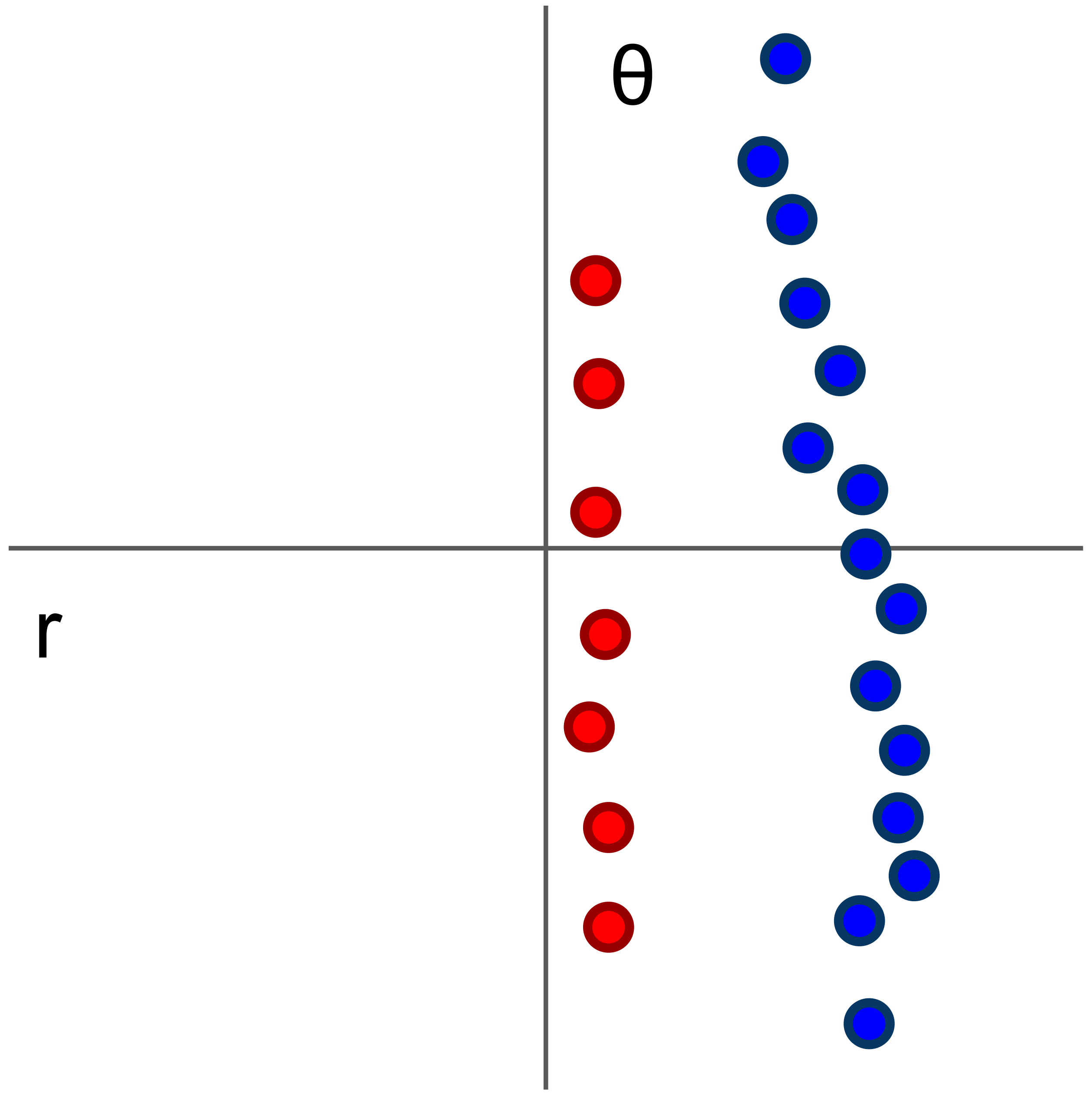
Cannot separate red and blue points with linear classifier

Why do we want non-linearity?



Cannot separate red and blue points with linear classifier

$f(x, y) = (r(x, y), \theta(x, y))$



After applying feature transform, points can be separated by linear classifier

Neural networks: also called fully connected network

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

"Neural Network" is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

(In practice we will usually add a learnable bias at each layer as well)

Neural networks: 3 layers

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

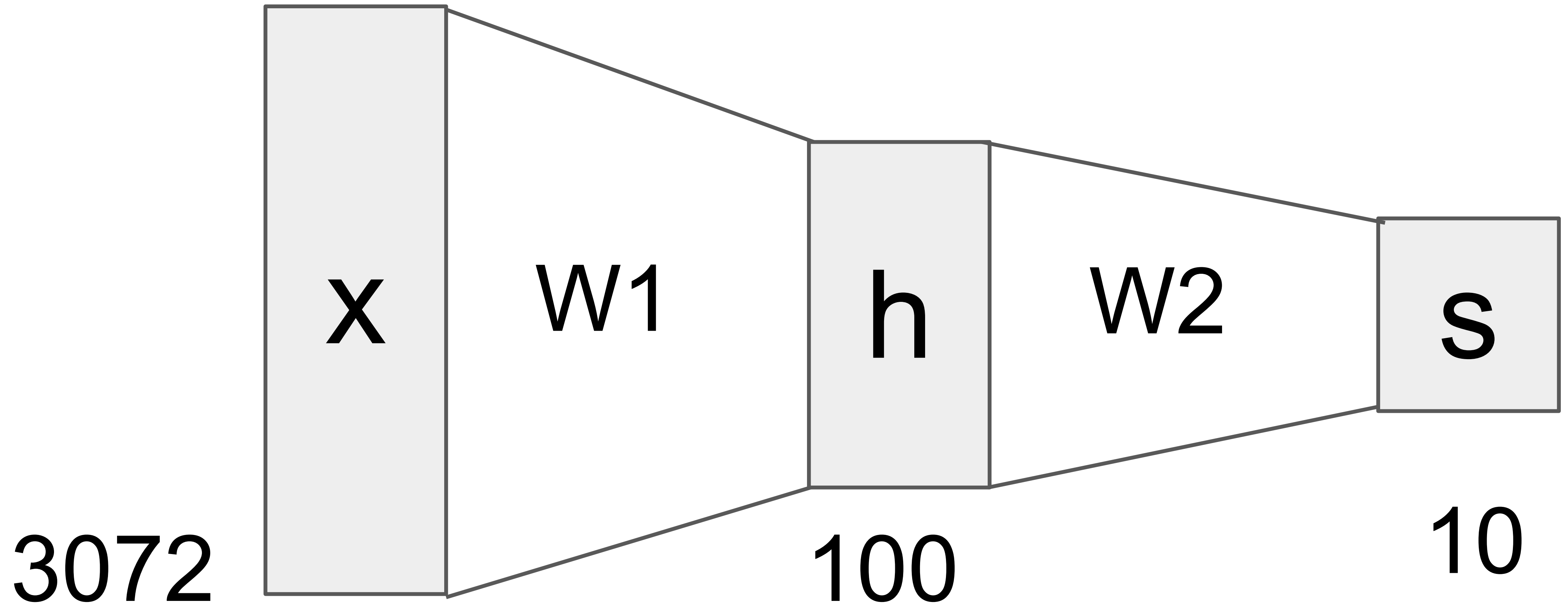
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

(In practice we will usually add a learnable bias at each layer as well)

Neural networks: hierarchical computation

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

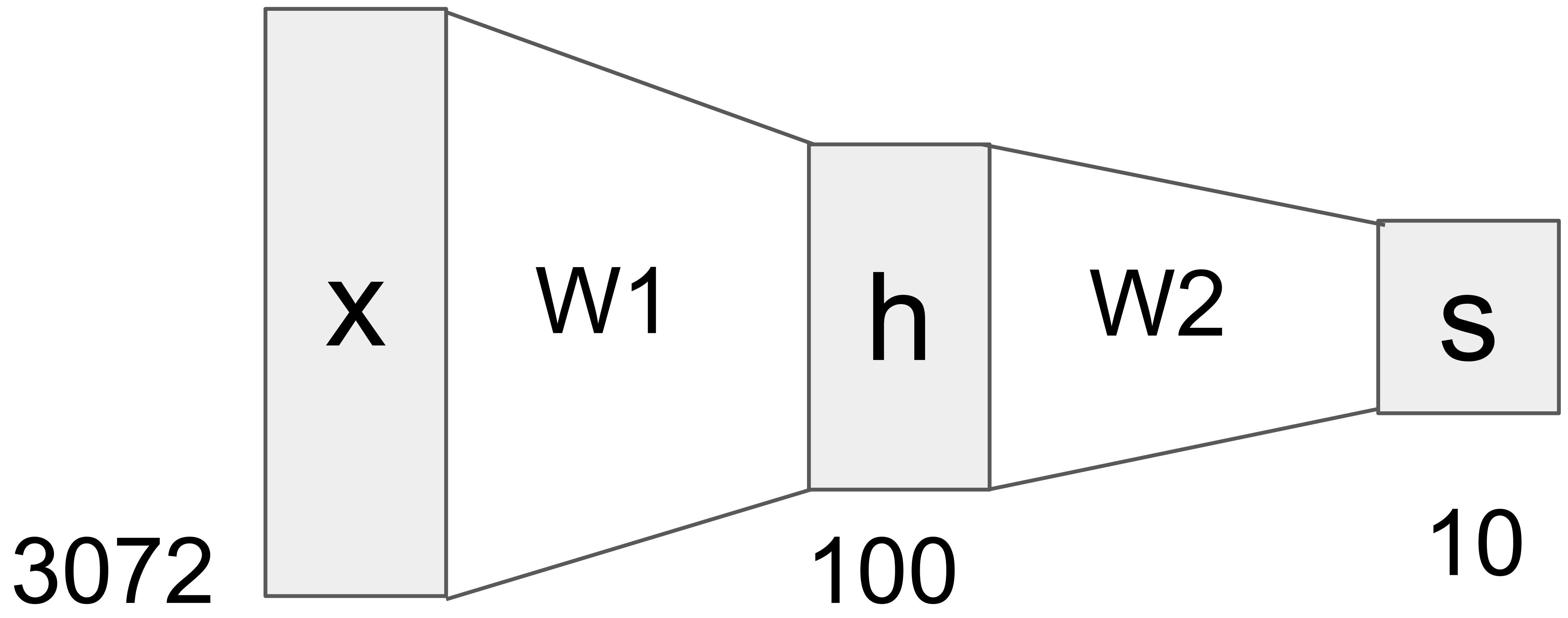


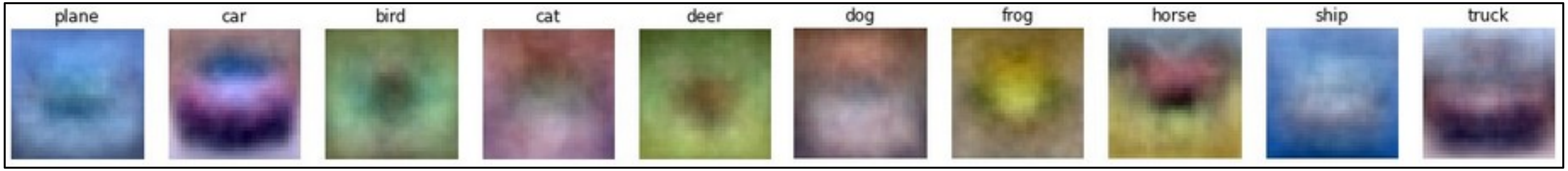
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Neural networks: learning 100s of templates

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$





Learn 100 templates instead of 10.

Share templates between classes

Neural networks: why is max operator important?

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \boxed{\max(0,} W_1 x)$

The function $\max(0, z)$ is called the **activation function.**

Q: What if we try to build a neural network without one?

$$f = W_2 W_1 x$$

Neural networks: why is max operator important?

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \boxed{\max(0,} W_1 x)$

The function $\max(0, z)$ is called the **activation function.**
Q: What if we try to build a neural network without one?

$$f = W_2 W_1 x \qquad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

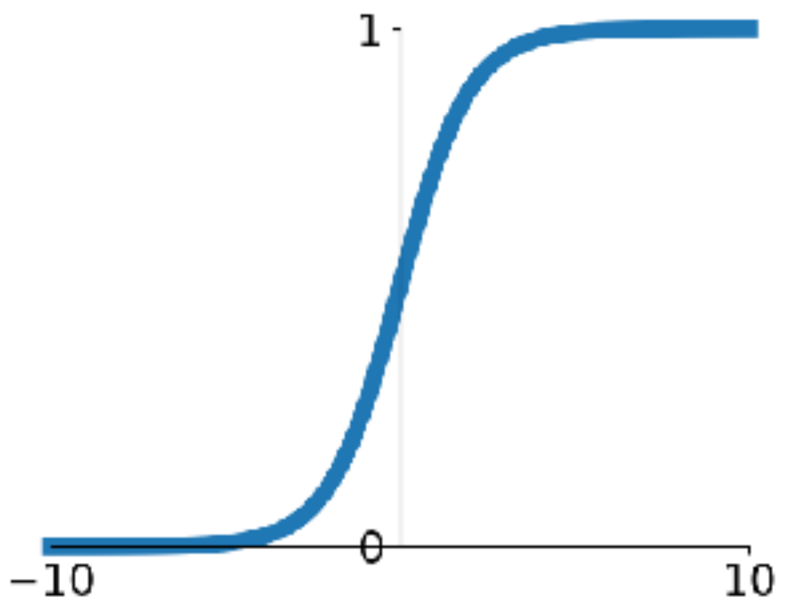
A: We end up with a linear classifier again!

Activation functions

ReLU is a good default choice for most problems

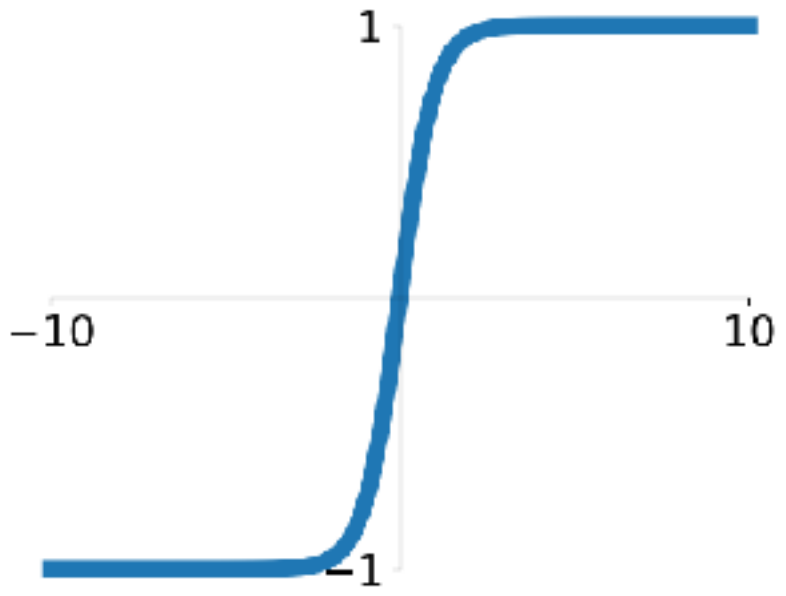
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



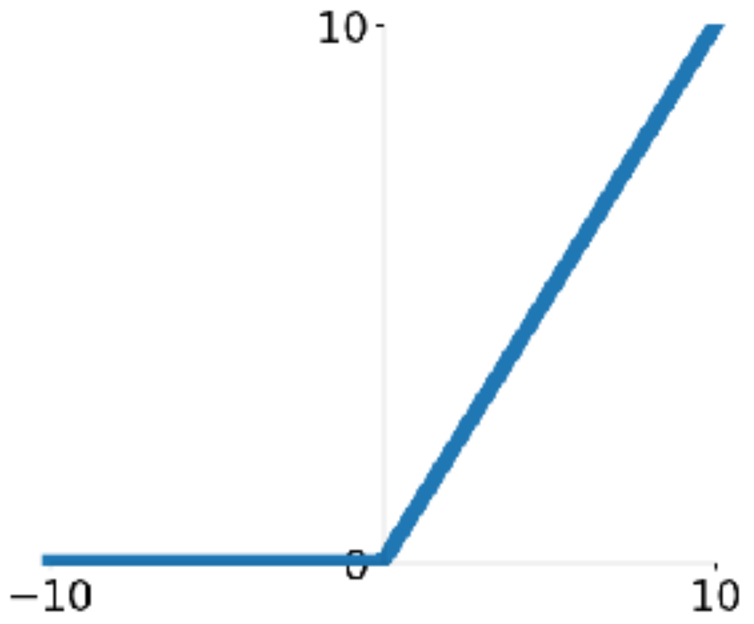
tanh

$\tanh(x)$



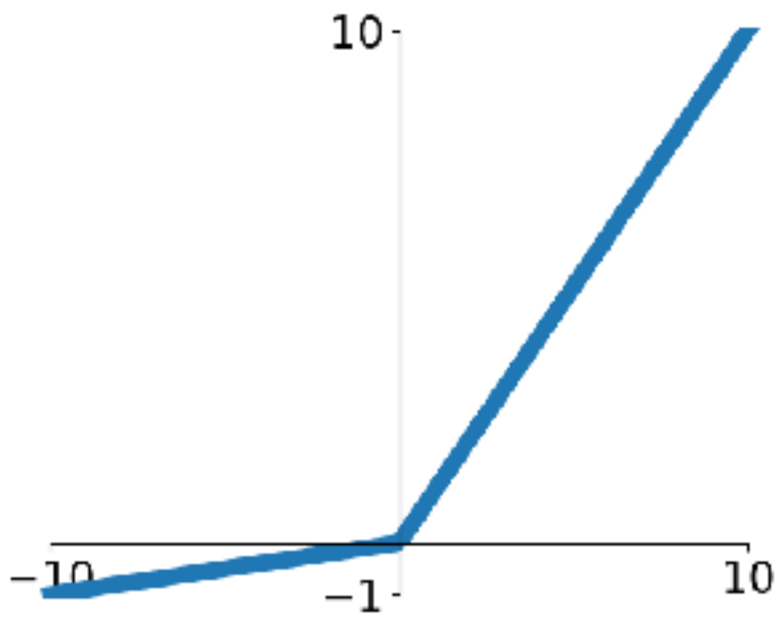
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

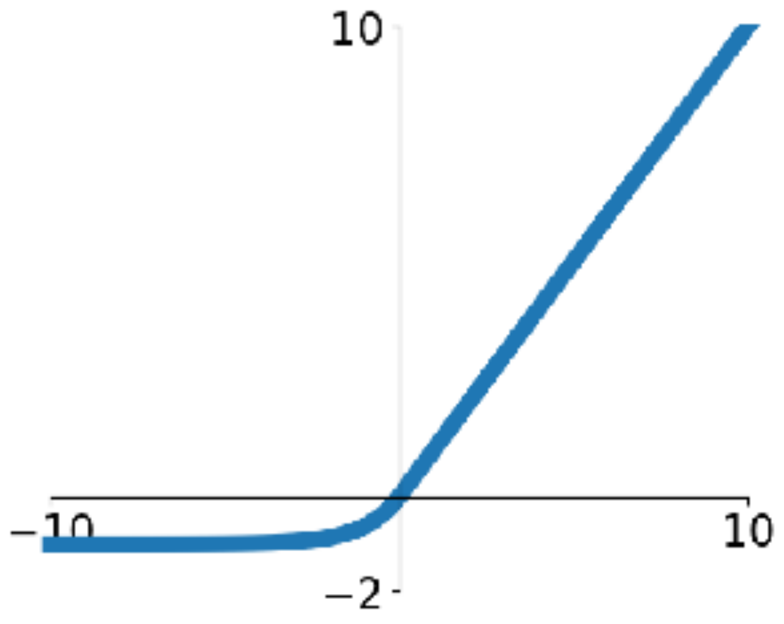


Maxout

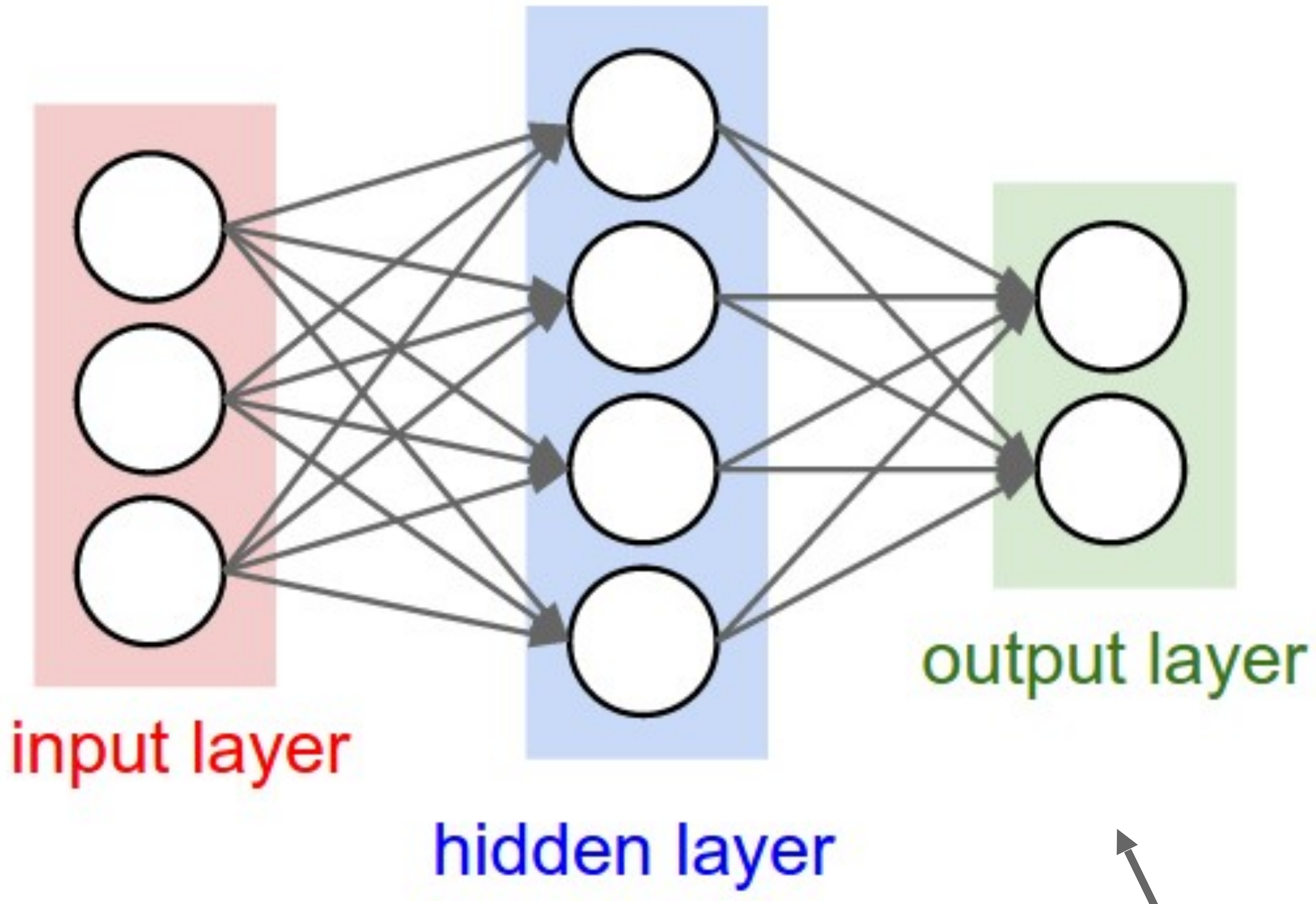
$\max(w_1^T x + b_1, w_2^T x + b_2)$

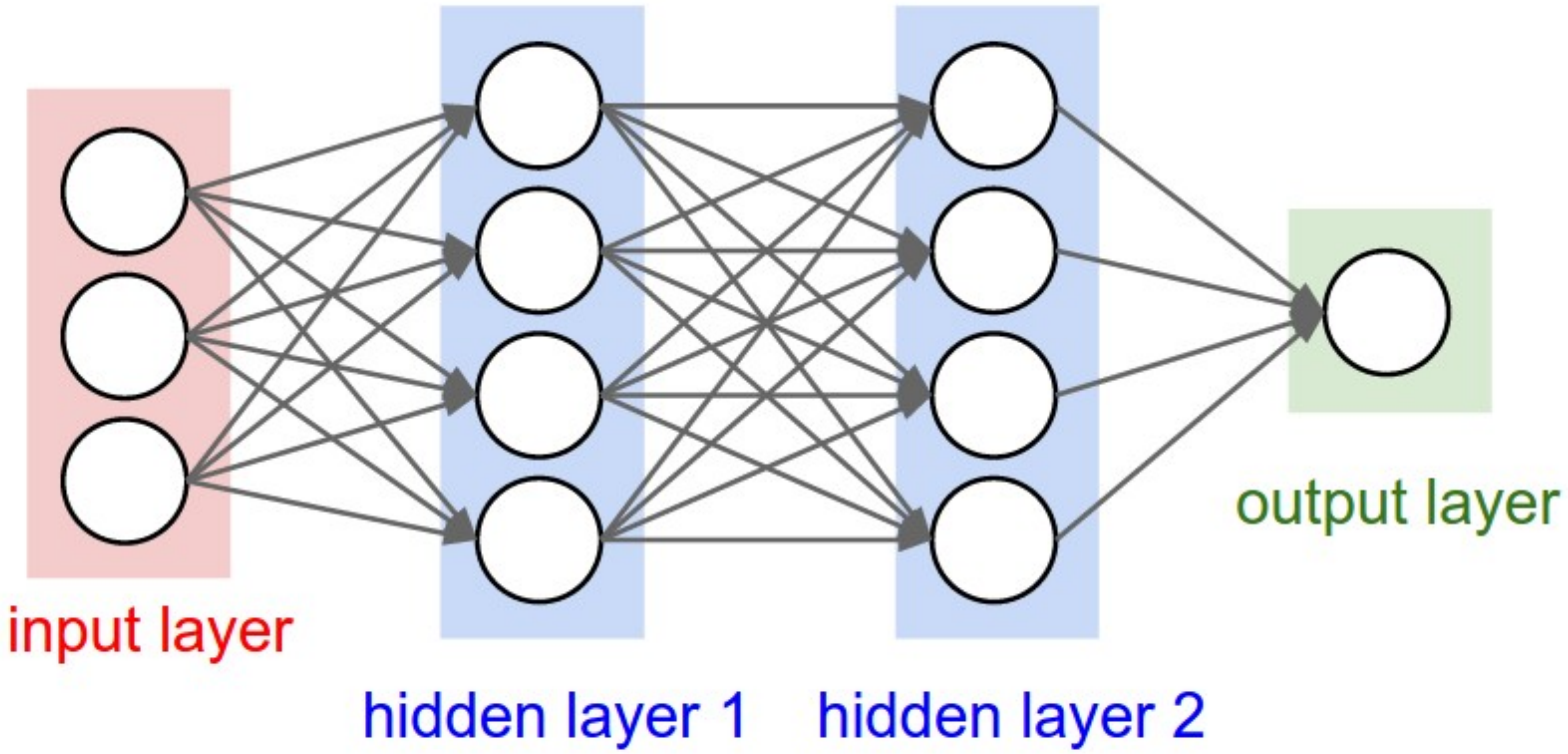
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural networks: Architectures



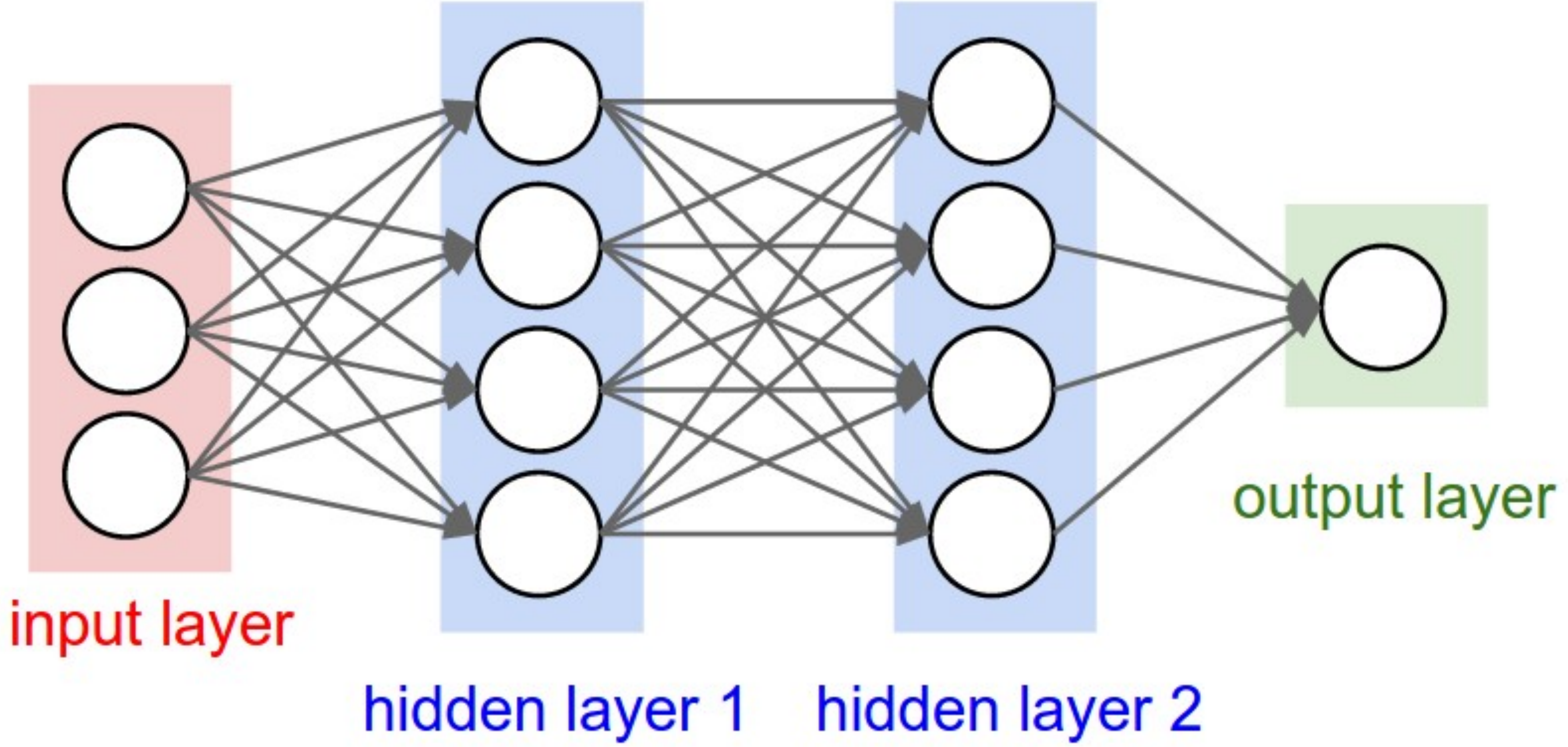


“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

“Fully-connected” layers

Example feed-forward computation of a neural network




```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Plugging in neural networks with loss functions

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM Loss on predictions

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

Problem: How to compute gradients?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM Loss on predictions

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn W_1 and W_2

Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

Problem: Very tedious: Lots of matrix calculus, need lots of paper

Problem: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

Problem: Not feasible for very complex models!

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Define the network

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Define the network

Forward pass

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Define the network

Forward pass

Calculate the analytical gradients

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

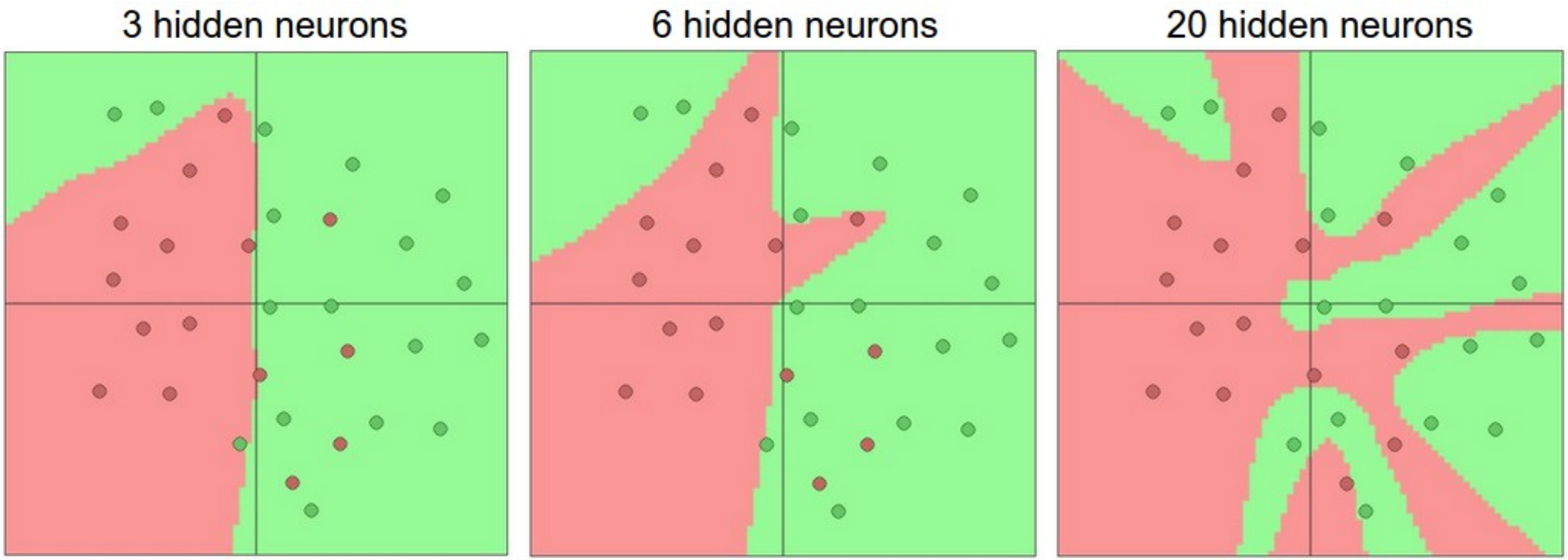
Define the network

Forward pass

Calculate the analytical gradients

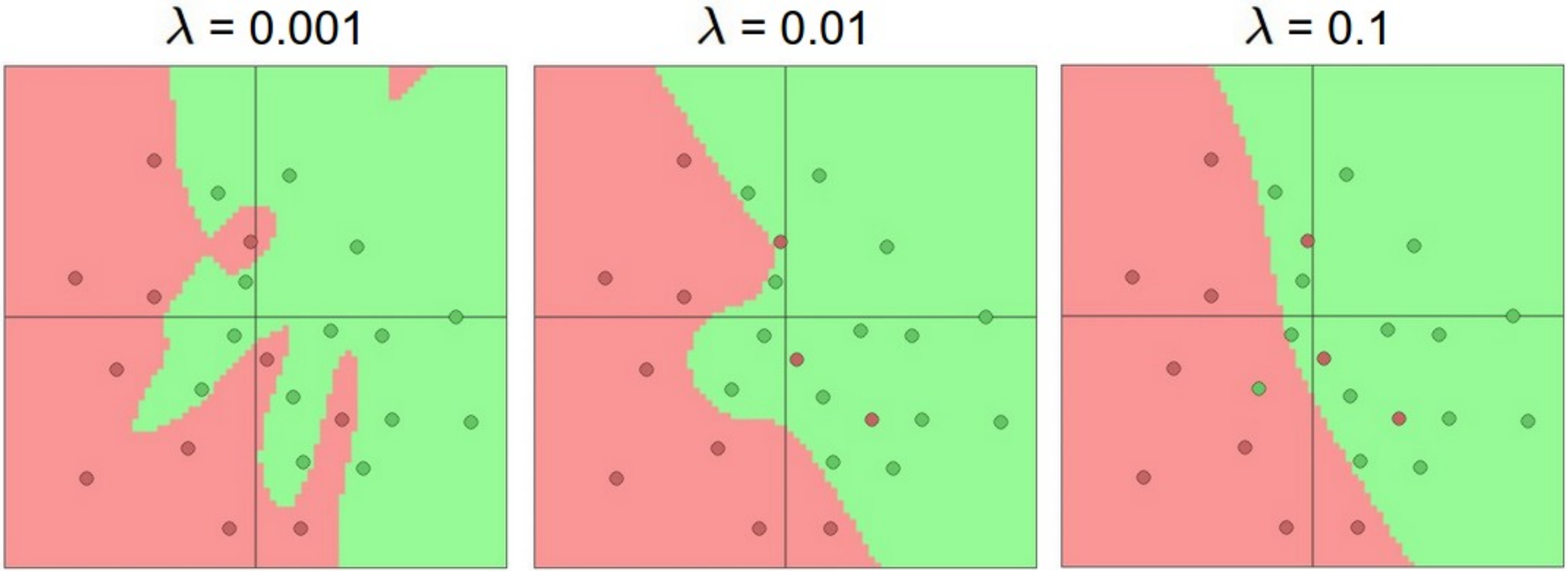
Gradient descent

Setting the number of layers and their sizes



more neurons = more capacity

Do not use size of neural network as a regularizer. Use stronger regularization instead:



(Web demo with ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Practical issues: gradient descent

Easy to get gradients wrong!

Solution — Automatic differentiation

Main idea

- All computations are compositions of elementary operations (+,-,*, /, cos, sin, max, etc.)
- We can write code to differentiate these basic operations
- For a complex function we can apply the chain rule of derivatives to write down a function that computes the gradients

Modern libraries will let you write an arbitrary forward function and will give you a function that computes the gradients (e.g., pytorch, tensorflow, Jax)

Practical issues: gradient descent

Computational and memory complexity

- Large size of gradients and activations on training examples
- Solution: mini-batch gradients

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{dL_n}{d\mathbf{w}}$$

$$\frac{dL}{d\mathbf{w}} = \sum_n \frac{dL_n}{d\mathbf{w}}$$

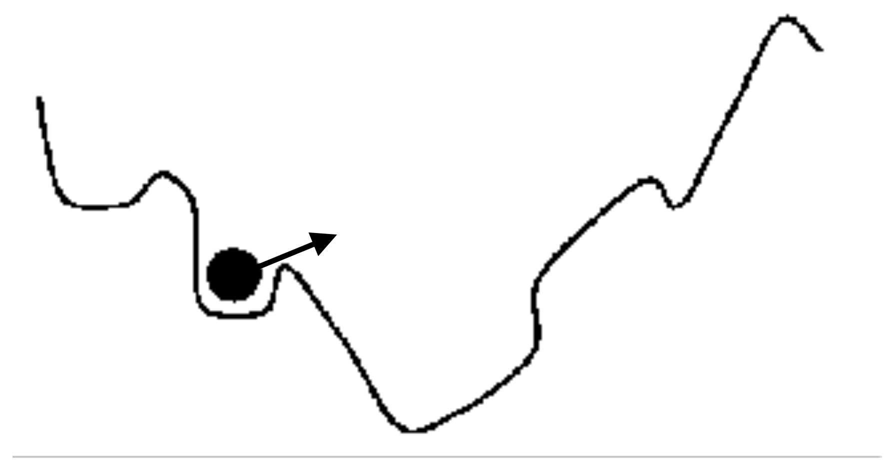
mini-batch
(n=256 << training set size)

Poor convergence

- Learning rate: start with a high value and reduce it when the validation error stops decreasing
- Momentum: move out small local minima
  - Usually set to a high value: $\beta = 0.9$

$$\Delta\mathbf{w}^{(t)} = \beta\Delta\mathbf{w}^{(t-1)} + (1-\beta)\left(-\eta\frac{dL_n}{d\mathbf{w}^{(t)}}\right)$$

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^t + \Delta\mathbf{w}^{(t)}$$



Practical issues: initialization

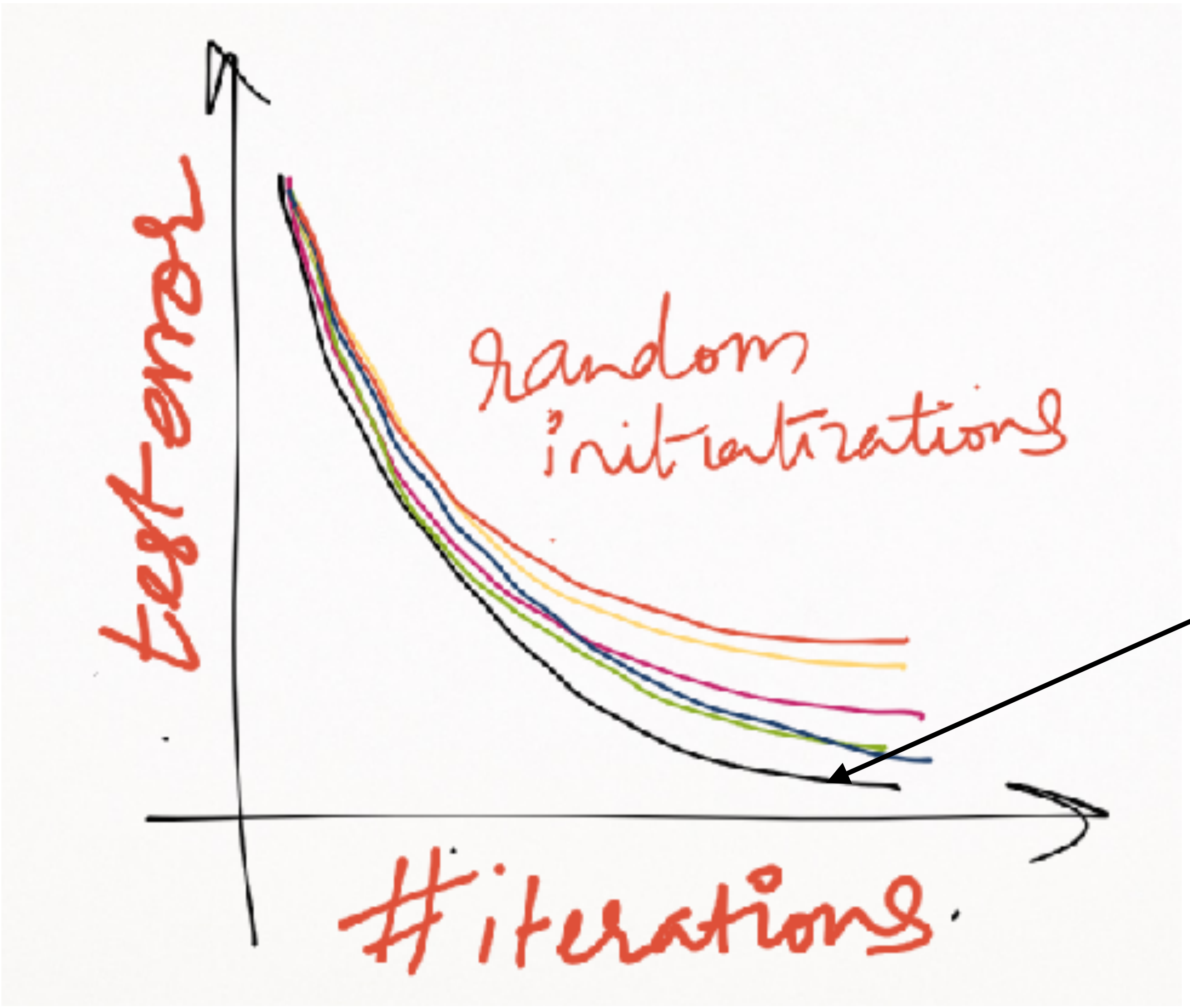
Initialization didn't matter for linear models

- Guaranteed convergence to global minima as long as step size is suitably chosen since the objective is convex

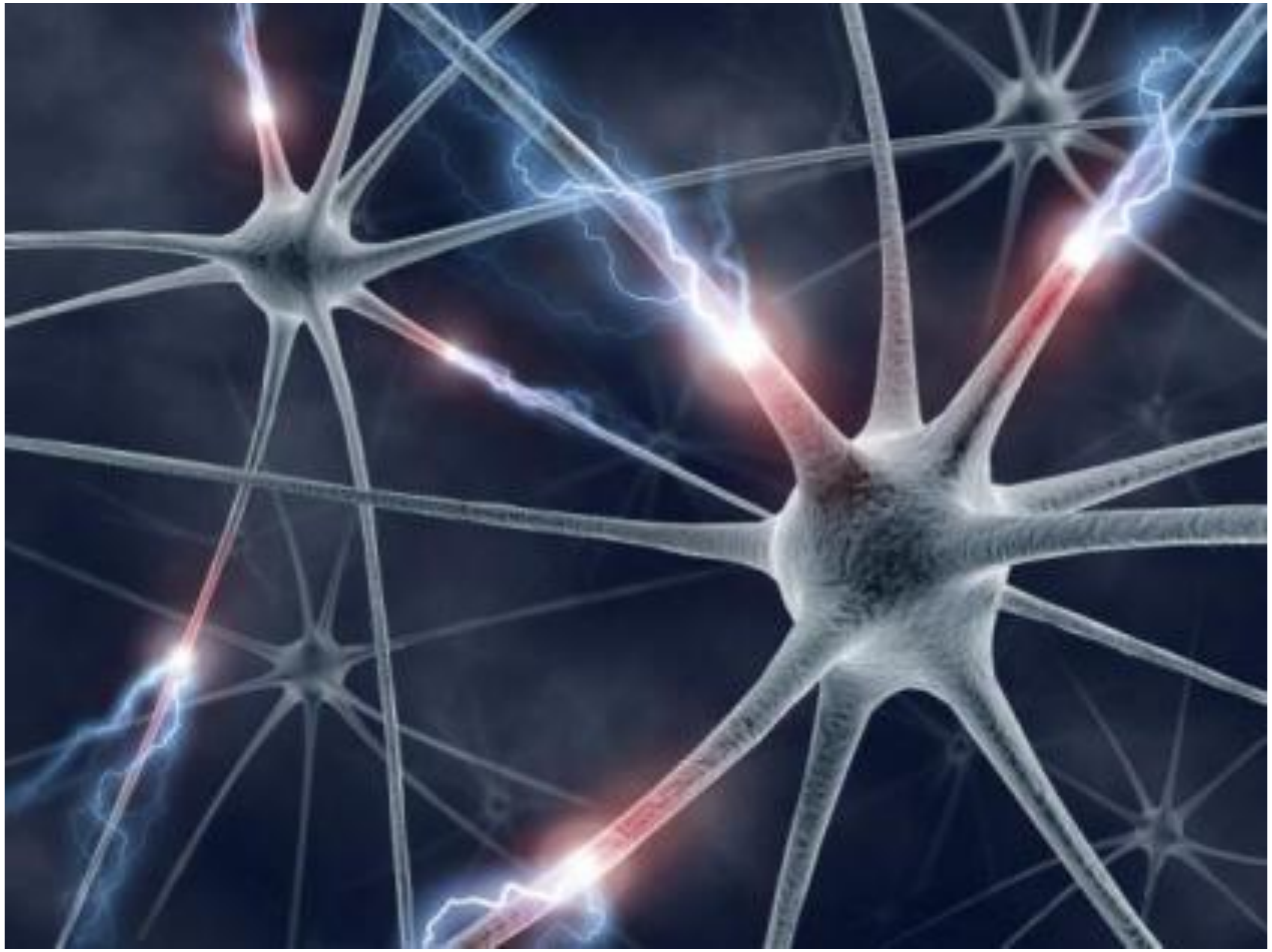
Neural networks are sensitive to initialization

- Many local minima
- **Symmetries:** reorder the hidden units and change the weights accordingly to get another network that produces identical outputs

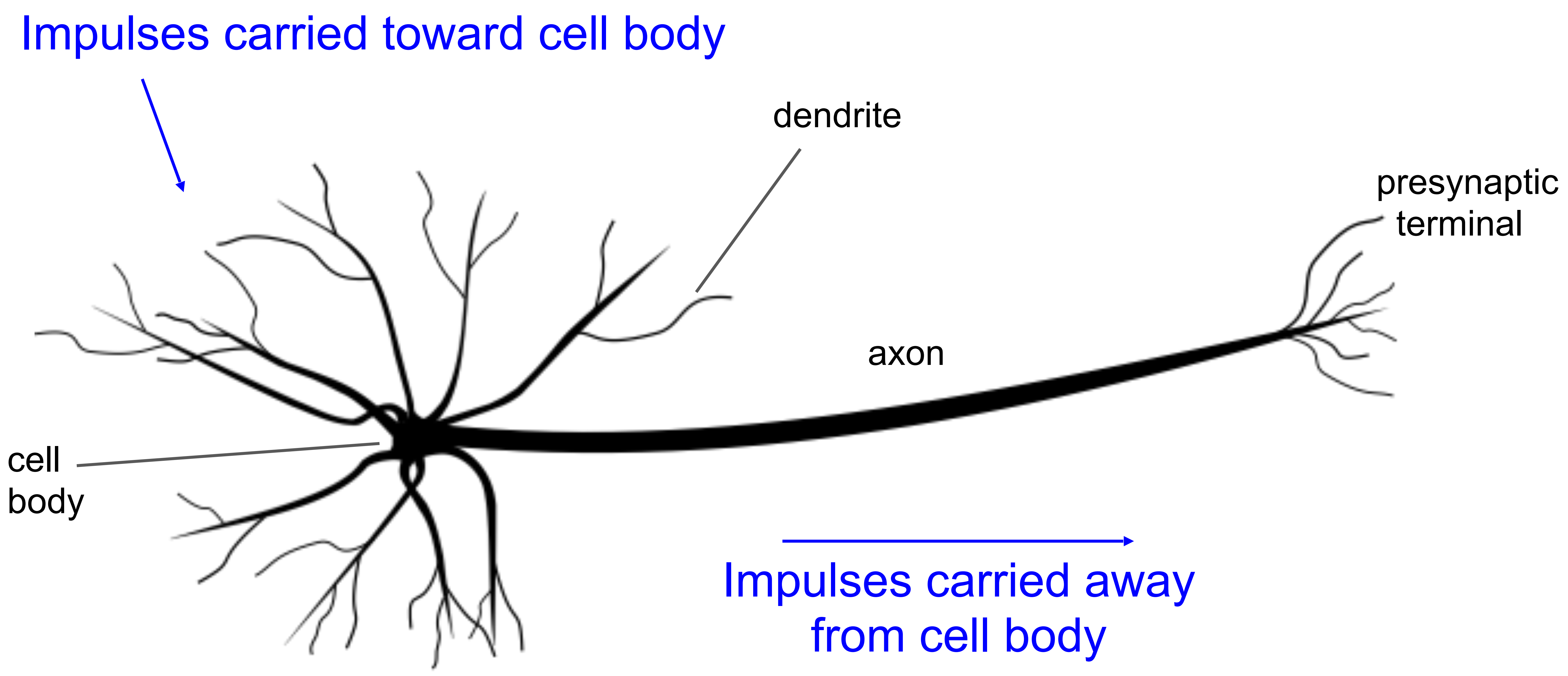
Train multiple networks with randomly initialized weights

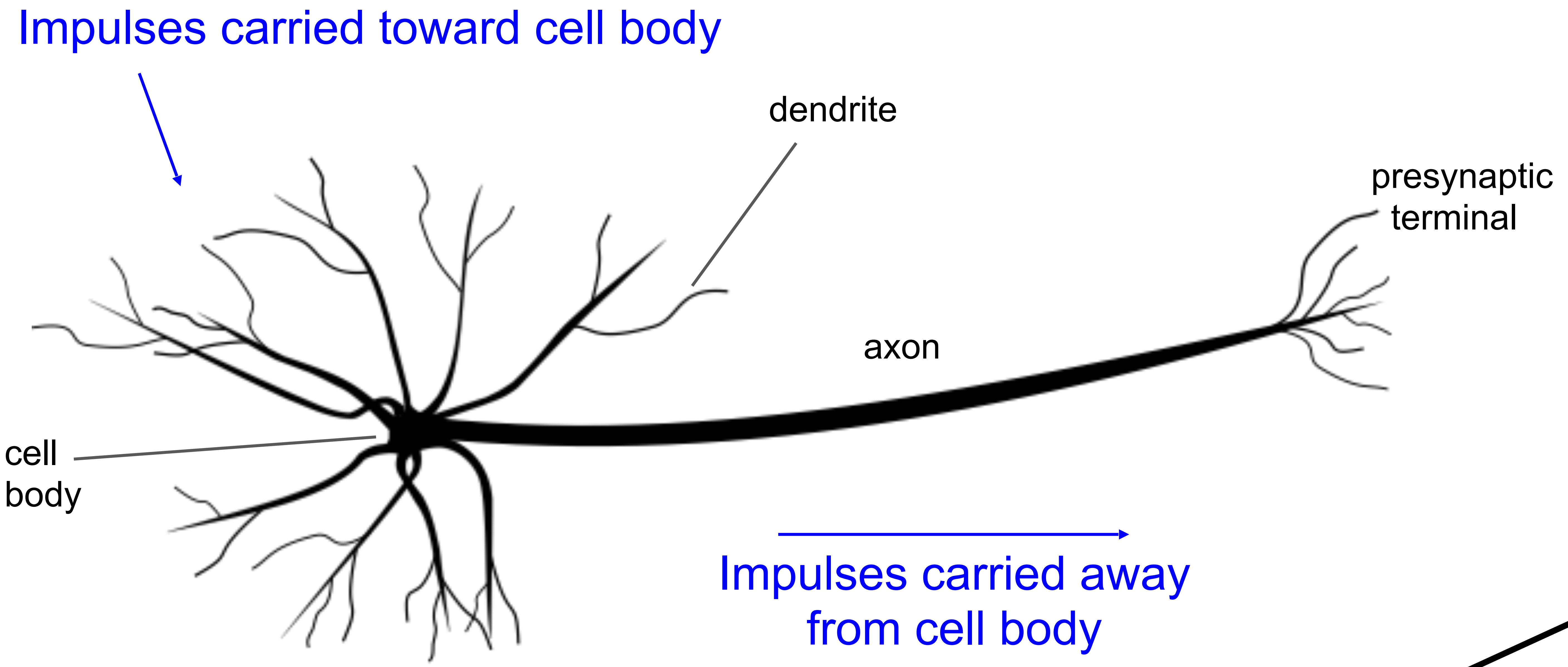


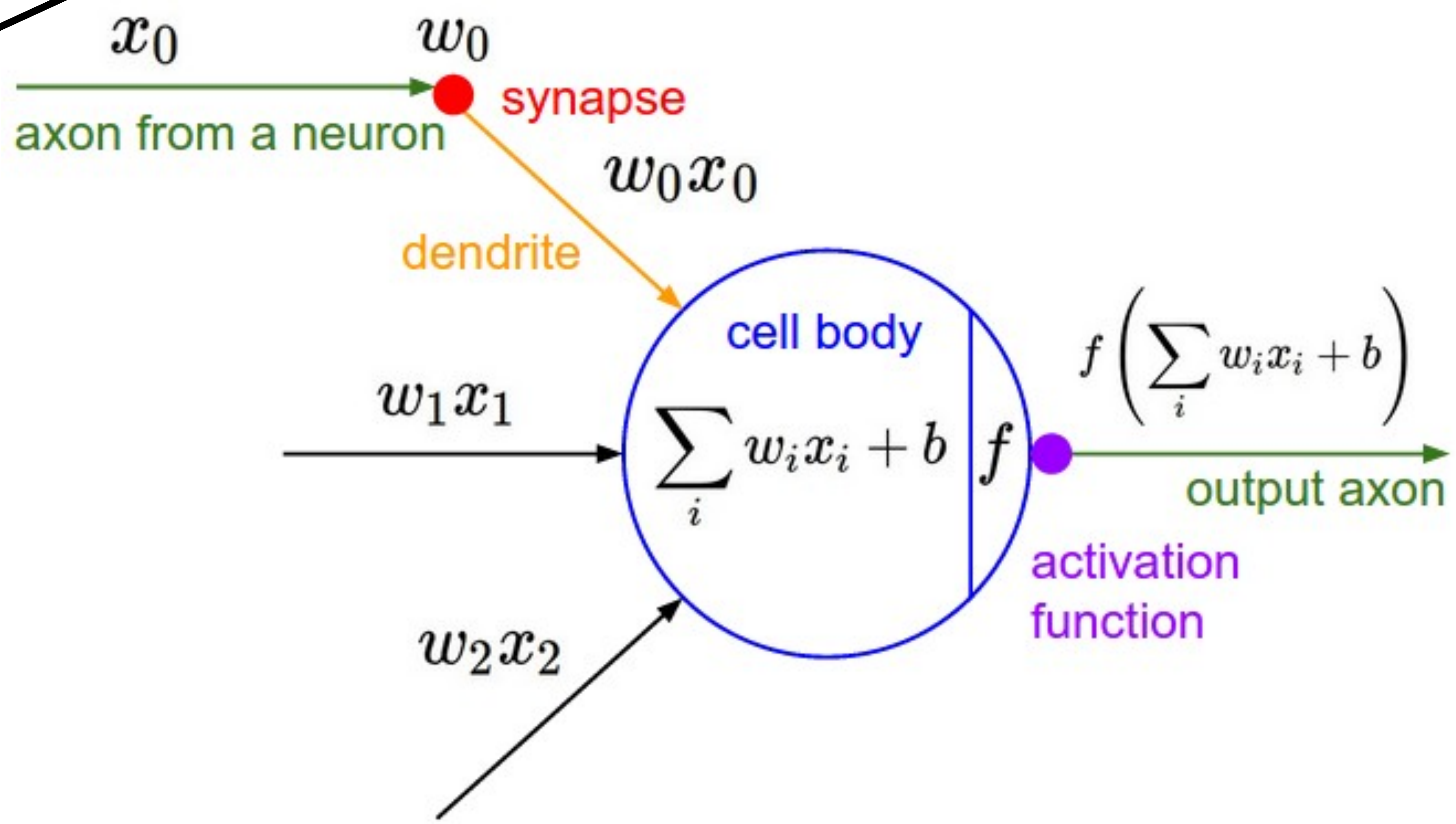
pick the best

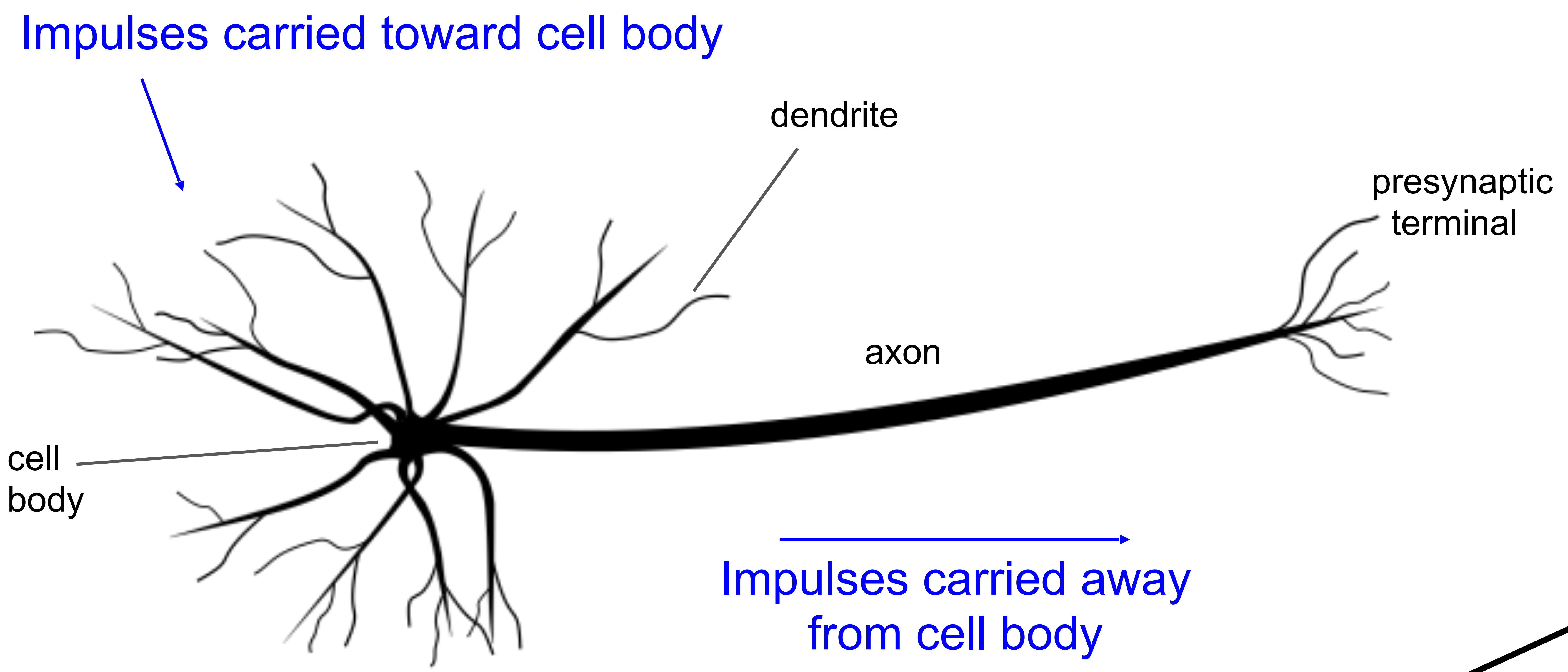


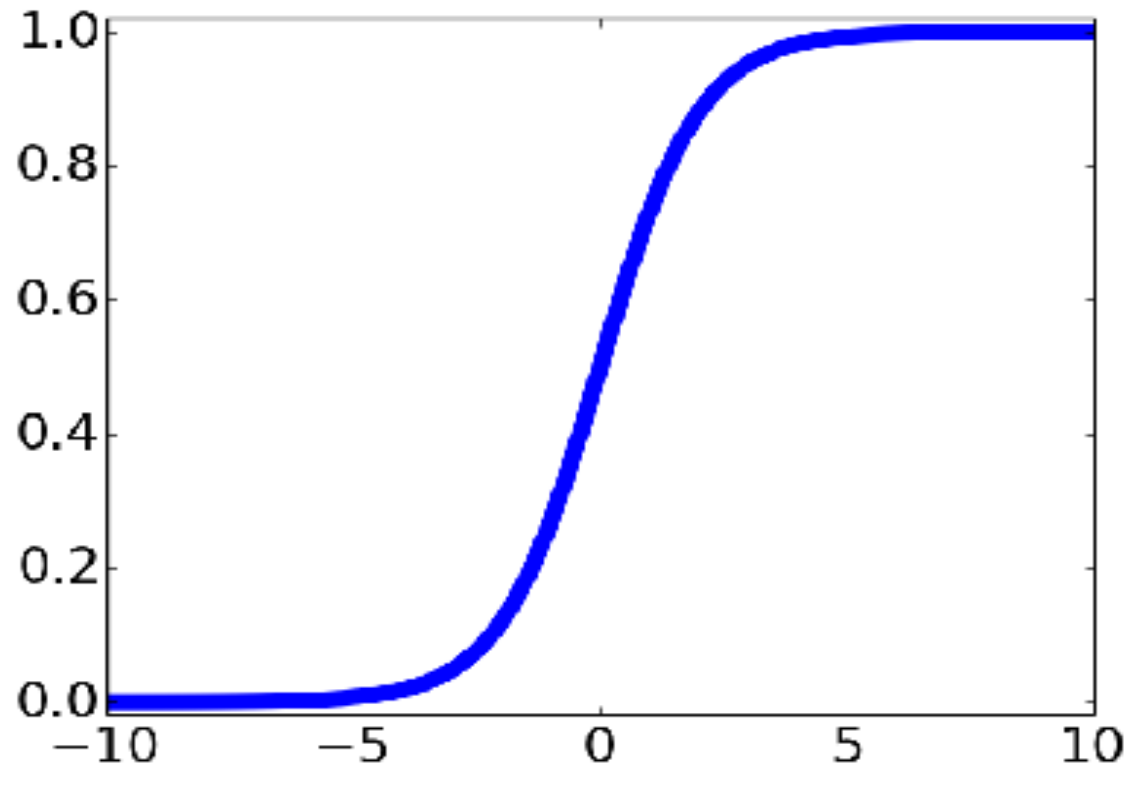
This image by Fotis Bobolas is licensed under CC-BY 2.0


Impulses carried toward cell body
dendrite
presynaptic terminal
axon
cell body
Impulses carried away from cell body


Impulses carried toward cell body
dendrite
presynaptic
terminal
axon
cell
body
Impulses carried away
from cell body

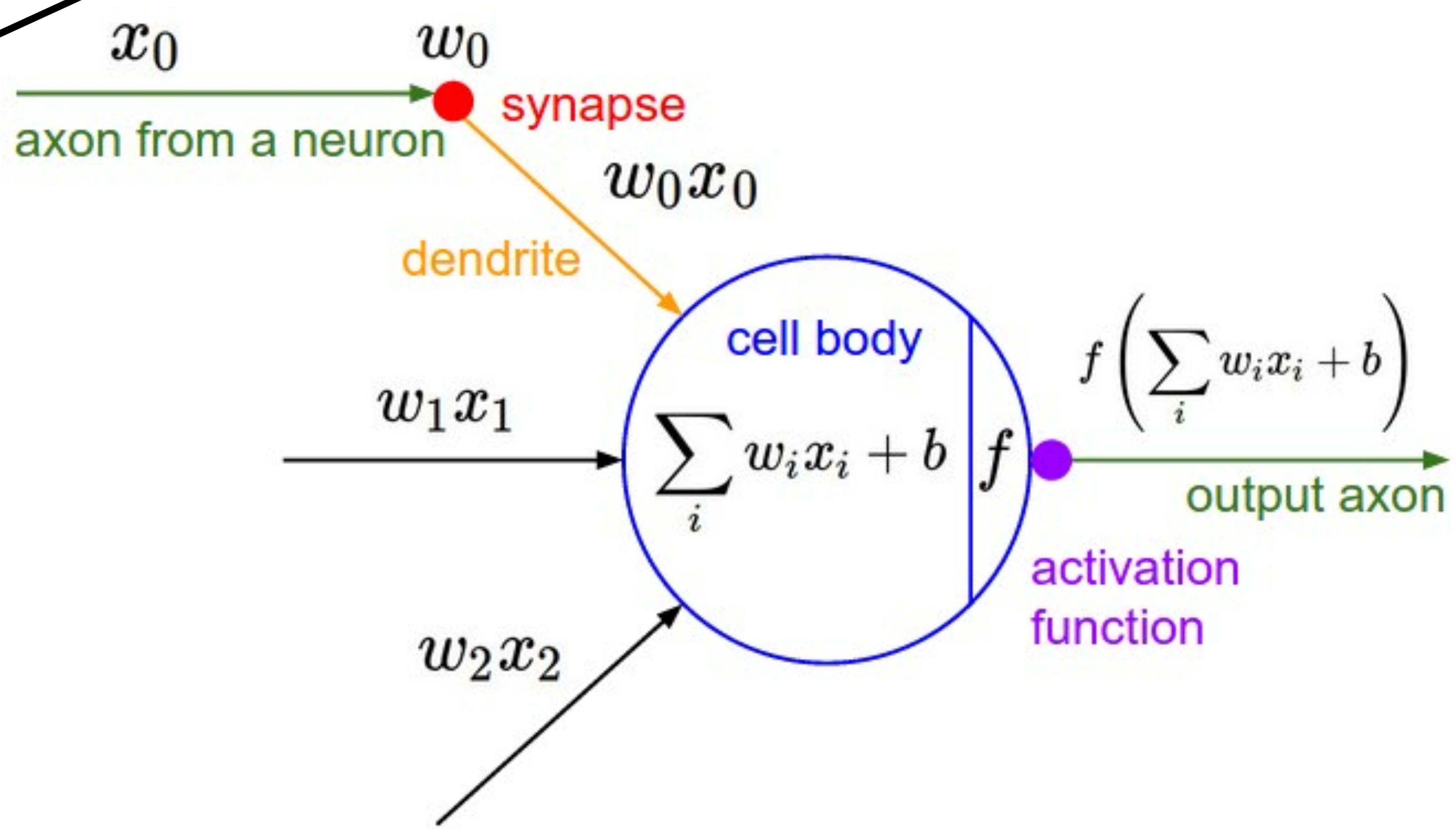

x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function
$w_2 x_2$



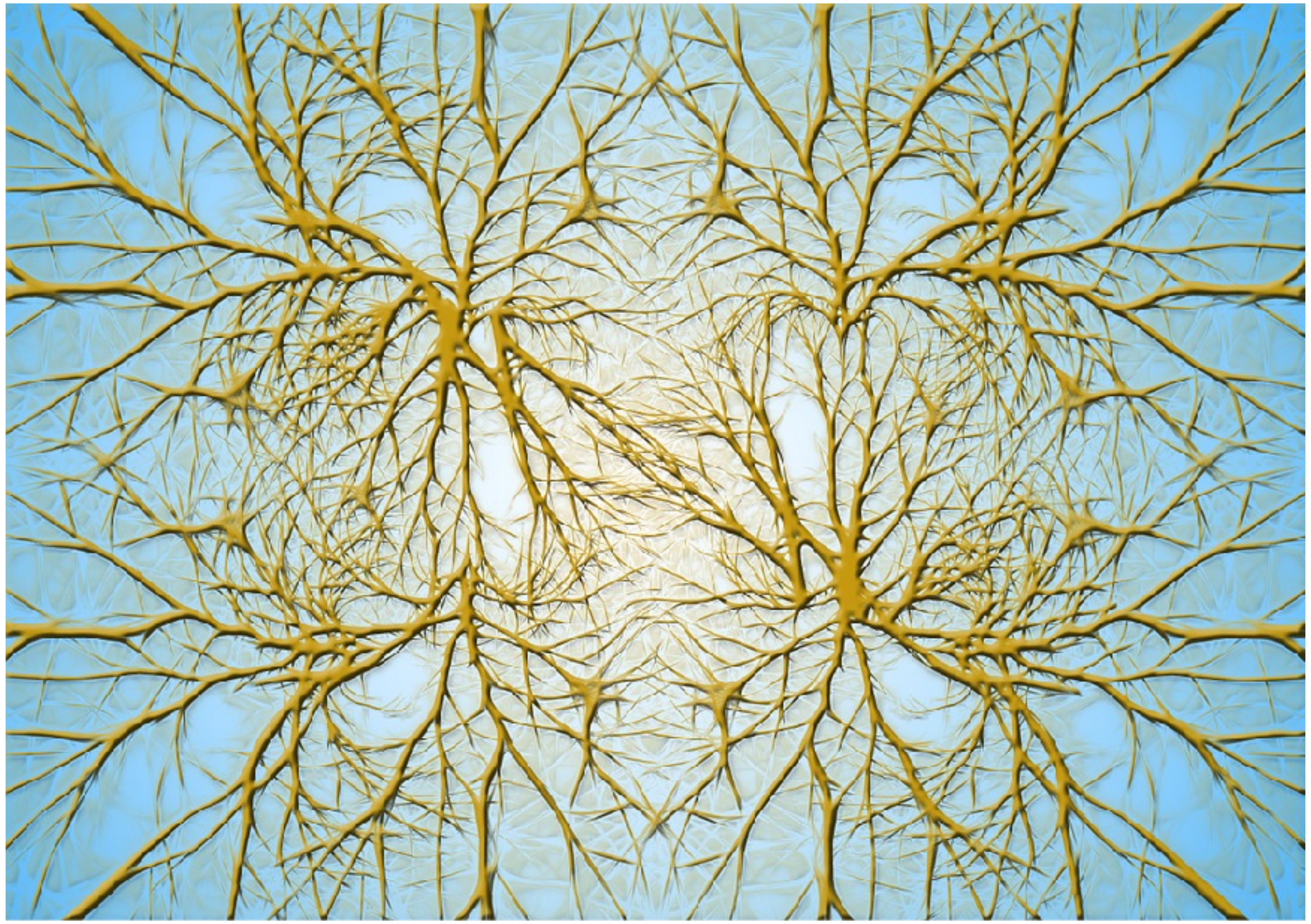


sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

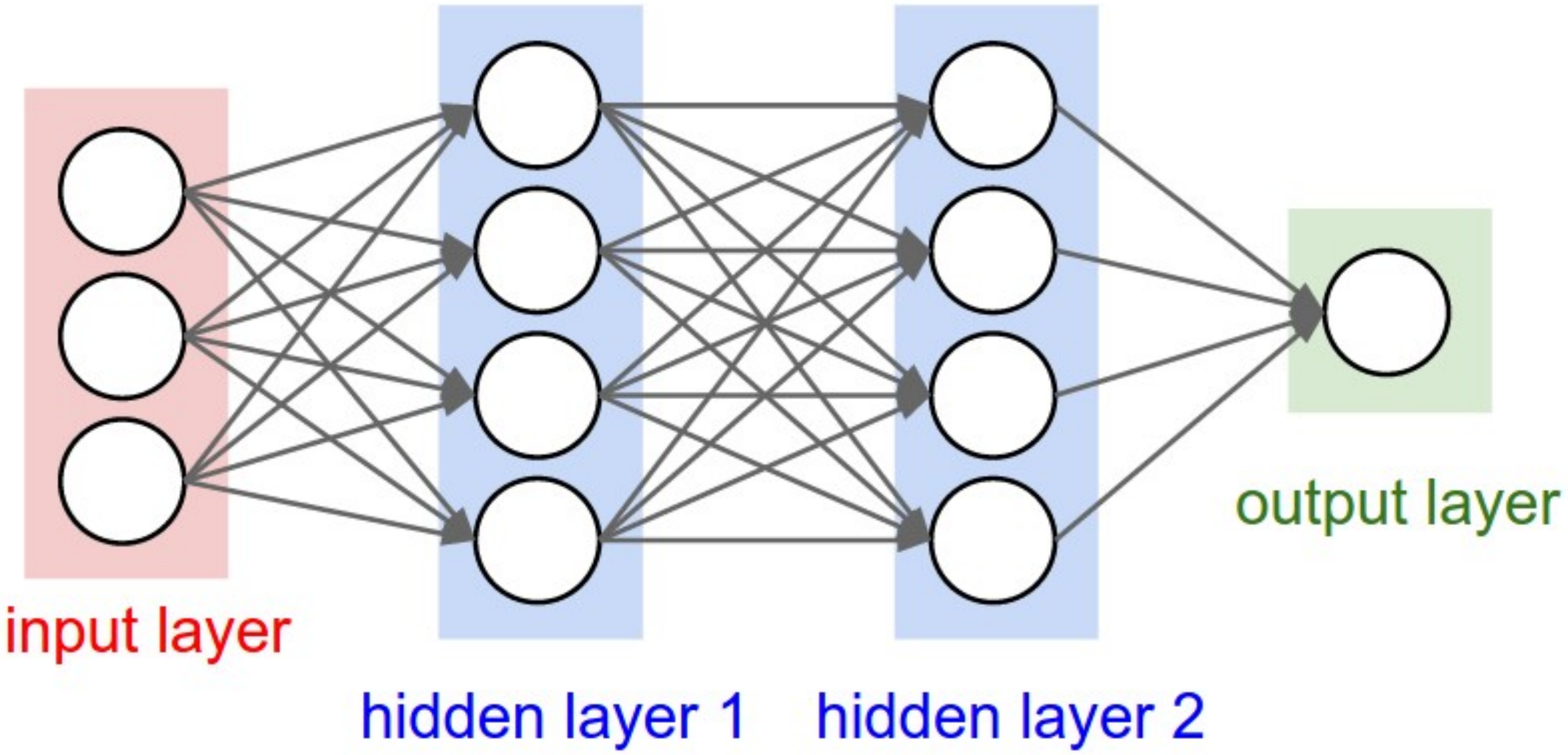


Biological Neurons:
Complex connectivity patterns

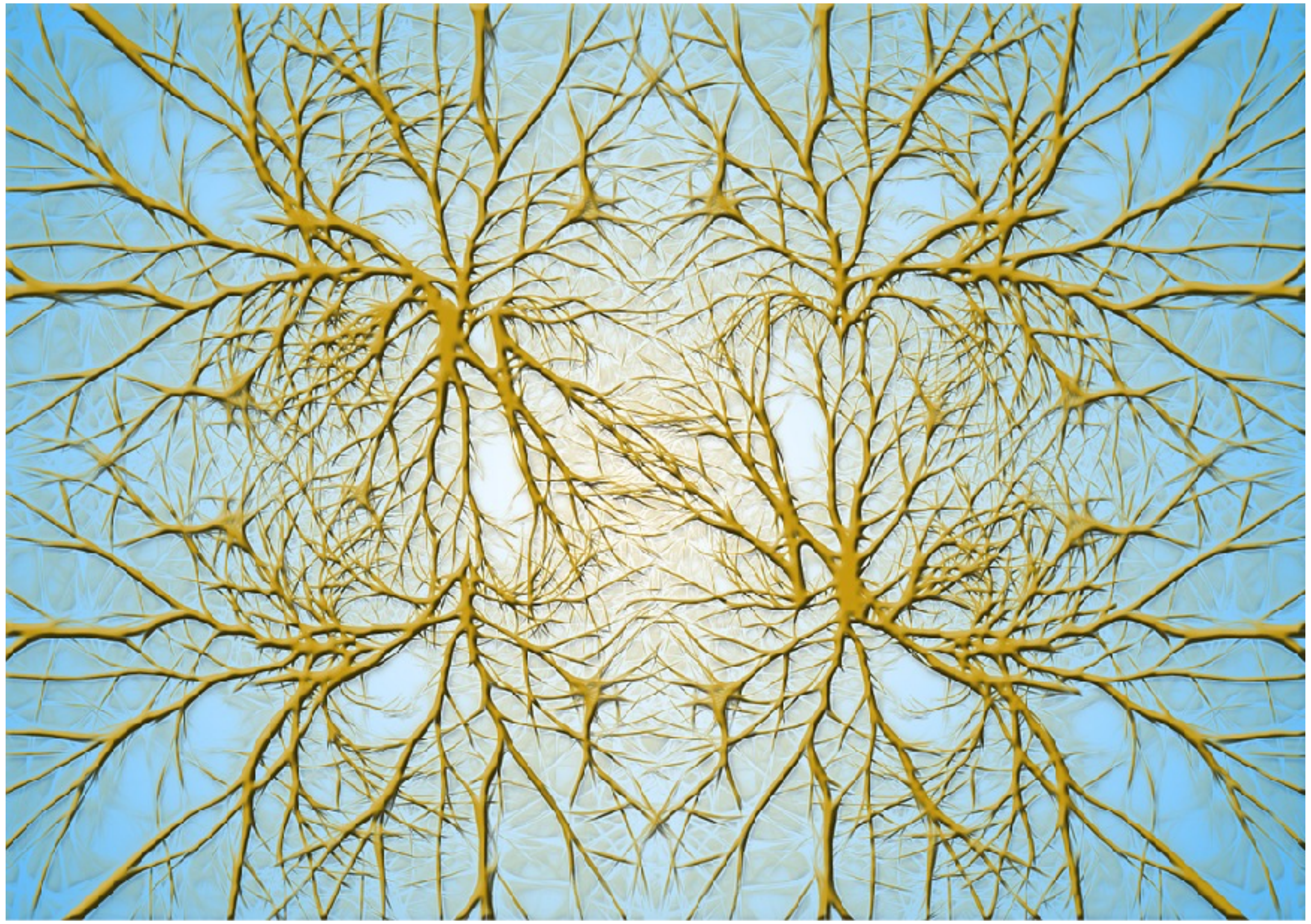


This image is CC0 Public Domain

Neurons in a neural network:
Organized into regular layers for computational efficiency

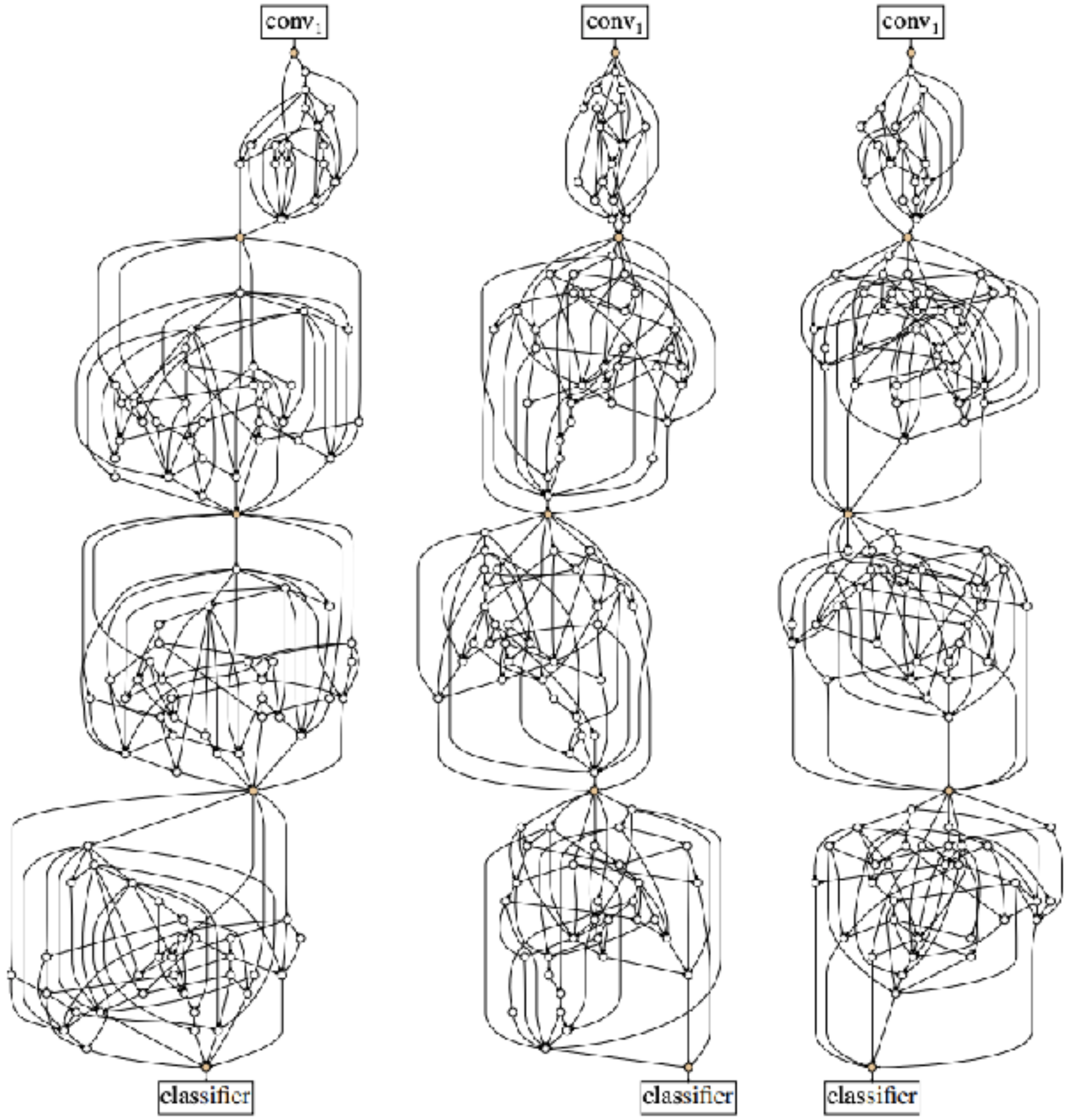


Biological Neurons:
Complex connectivity patterns



This image is CC0 Public Domain

But neural networks with random connections can work too!



Xie et al, “Exploring Randomly Wired Neural Networks for Image Recognition”, arXiv 2019

Be very careful with your brain analogies!

Biological Neurons:
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system

[Dendritic Computation. London and Hausser]

Convolutional Neural Networks

Convolutional neural networks

Images are not just a collection of pixels

- Locality: edges, corners, blobs
- Translation invariance

The convolution operation:

filter: horizontal edge

image

absolute value of the output of convolution of the image and filter

Convolutional neural networks

Images are not just a collection of pixels

- Locality: edges, corners, blobs
- Translation invariance

The convolution operation:

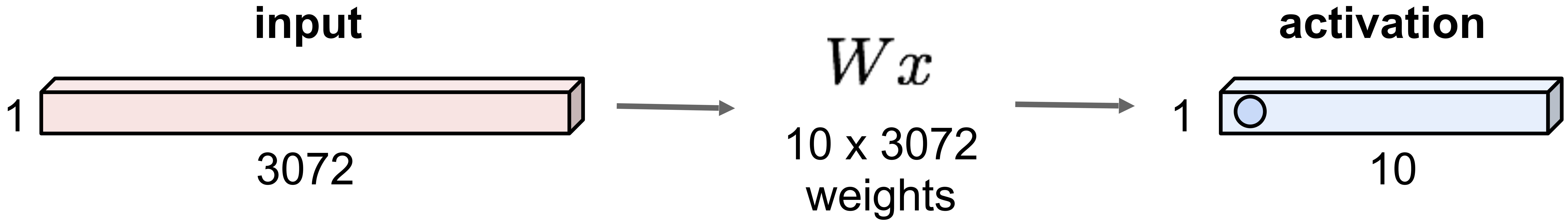
filter: vertical edge

image

absolute value of the output of convolution of the image and filter

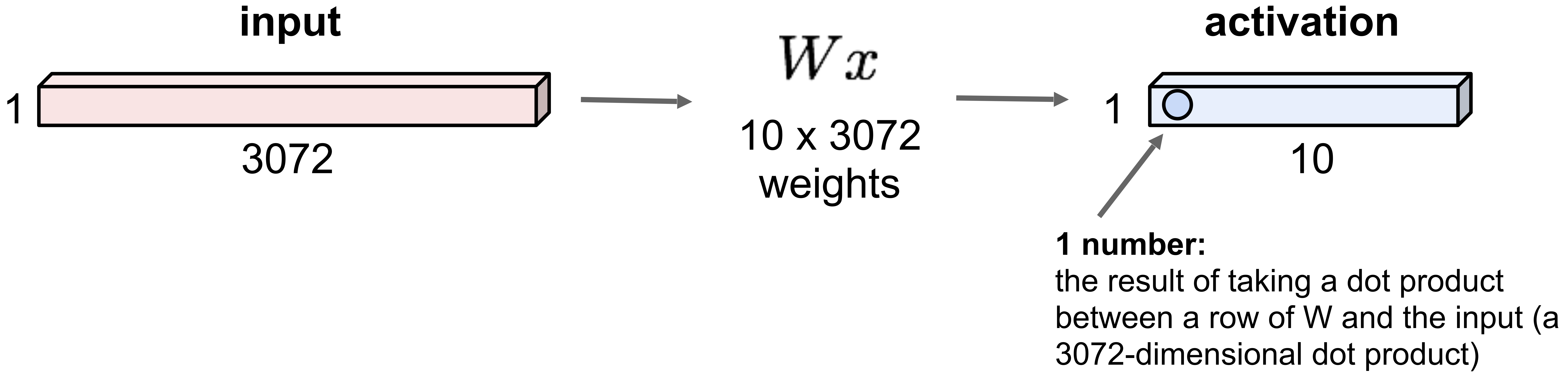
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



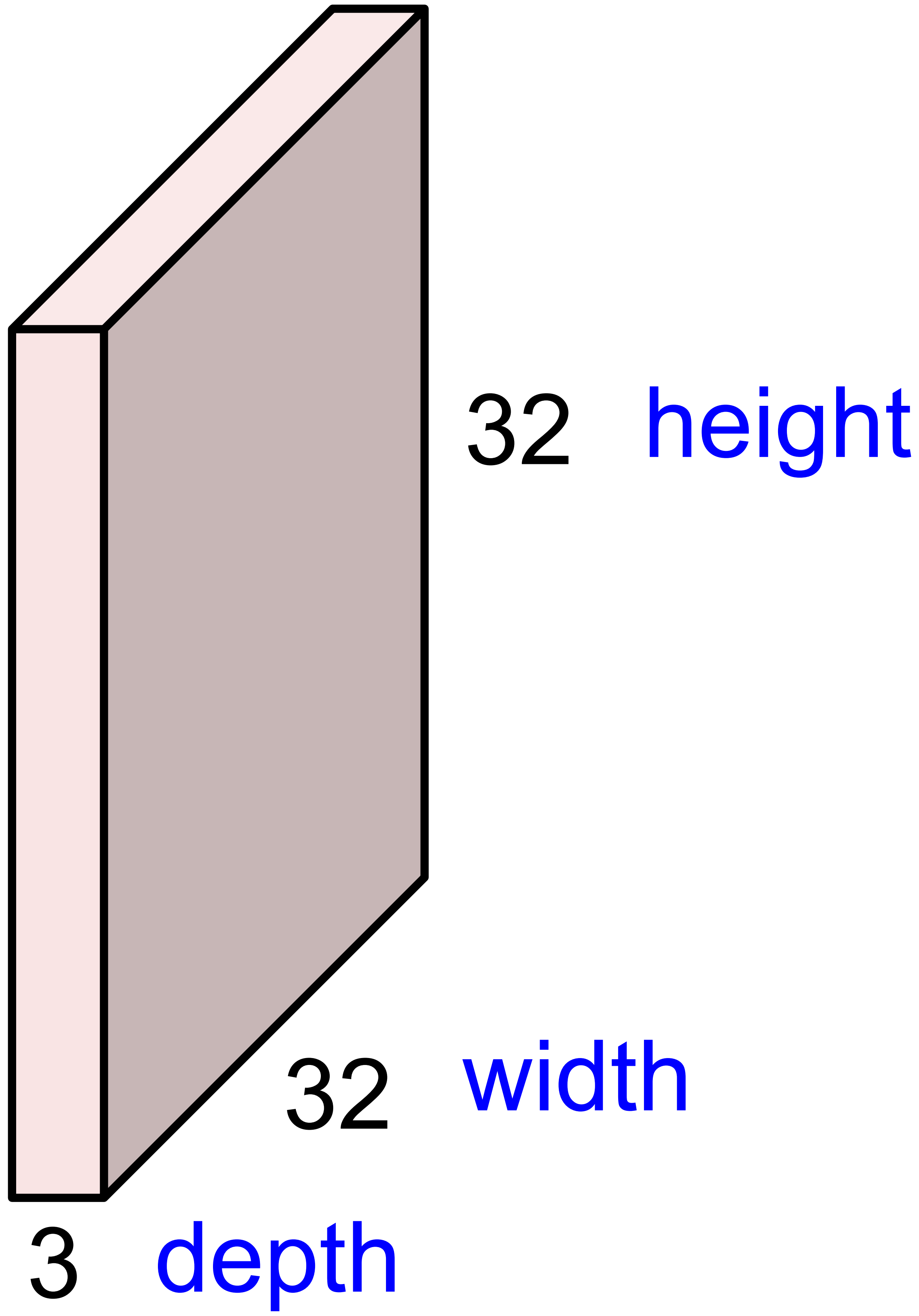
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



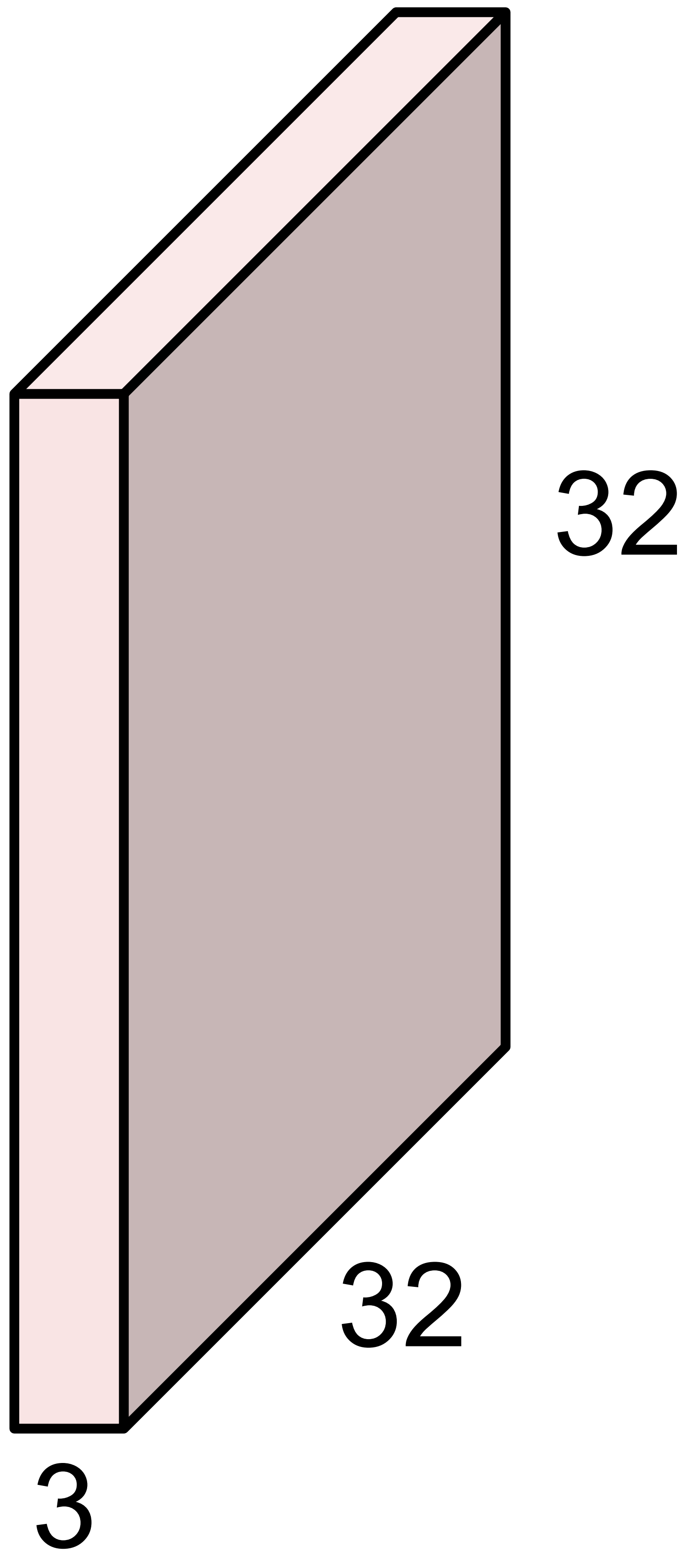
Convolution Layer

32x32x3 image -> preserve spatial structure

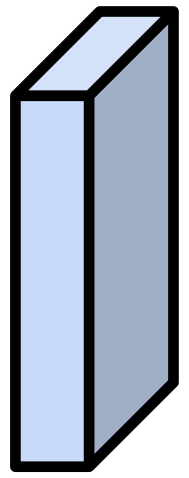


Convolution Layer

32x32x3 image



5x5x3 filter

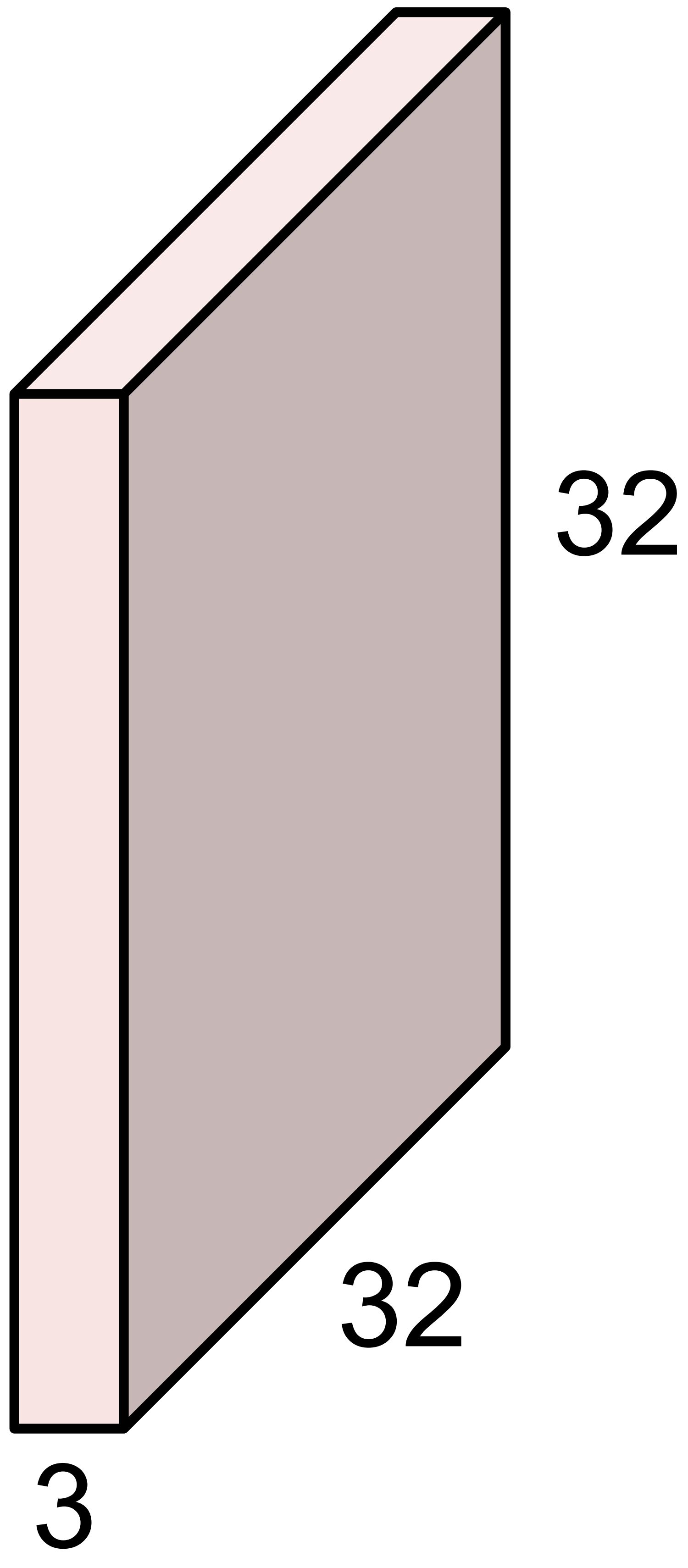


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

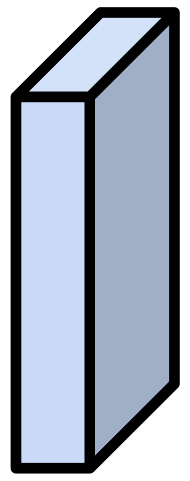
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

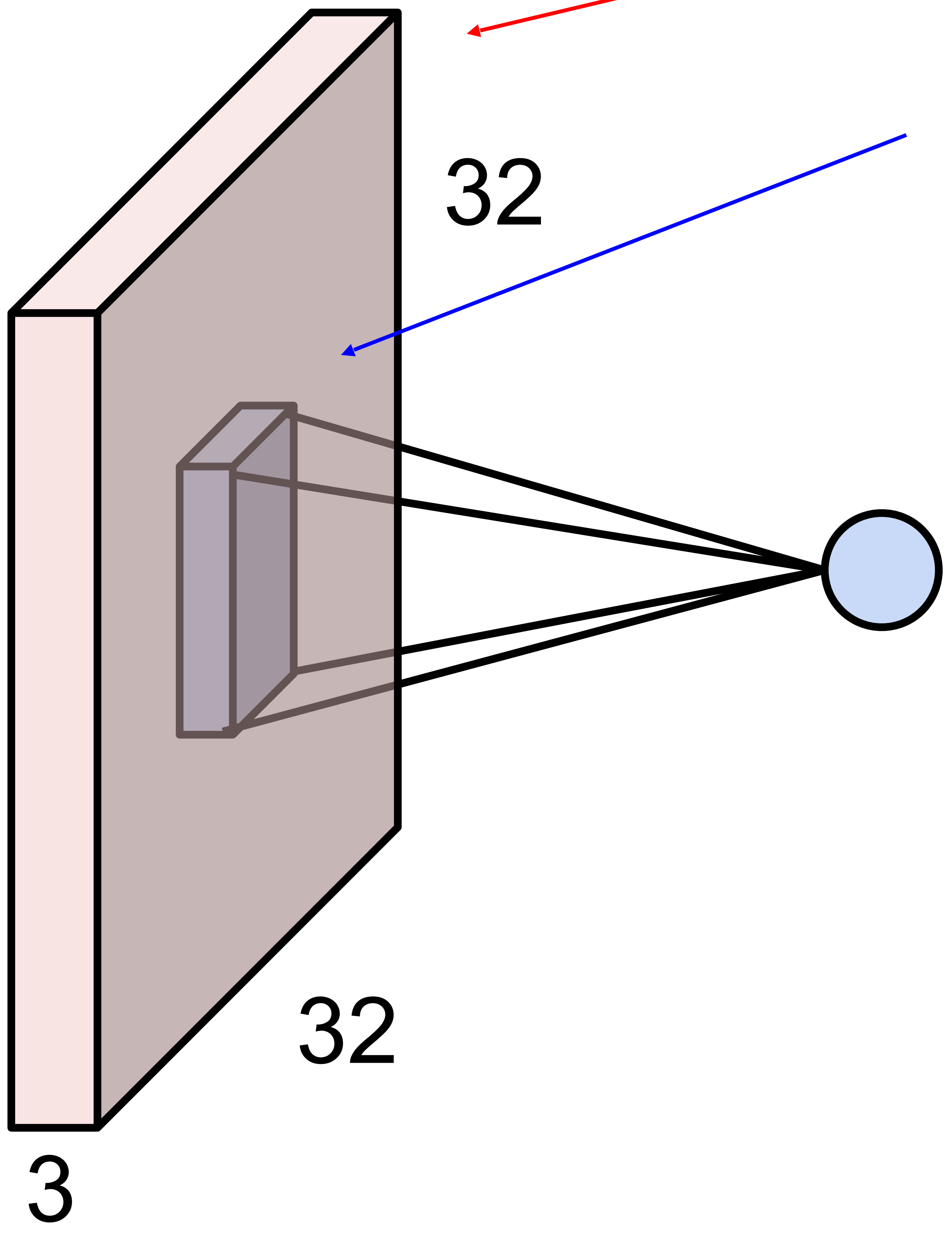


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

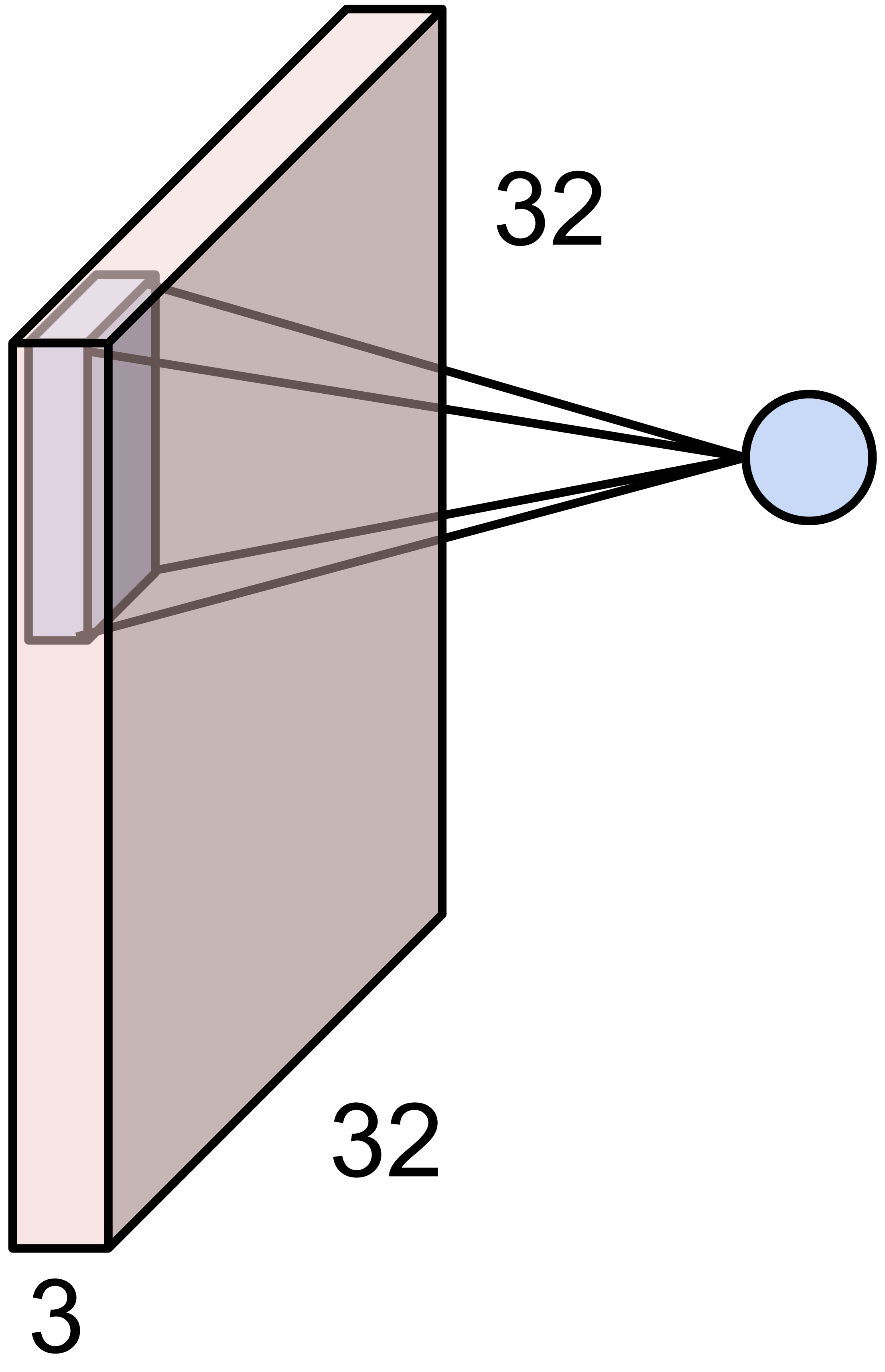
5x5x3 filter w



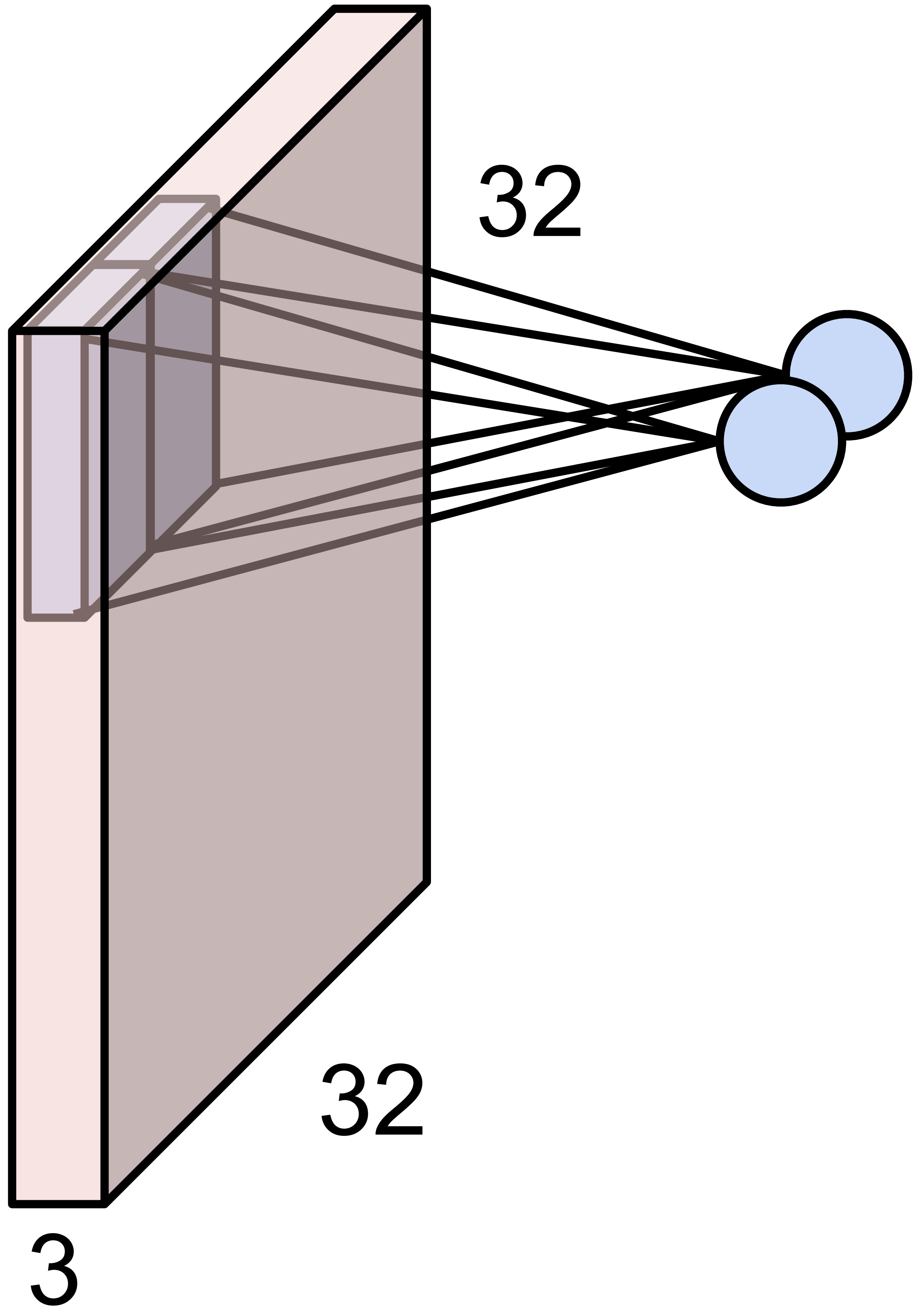
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

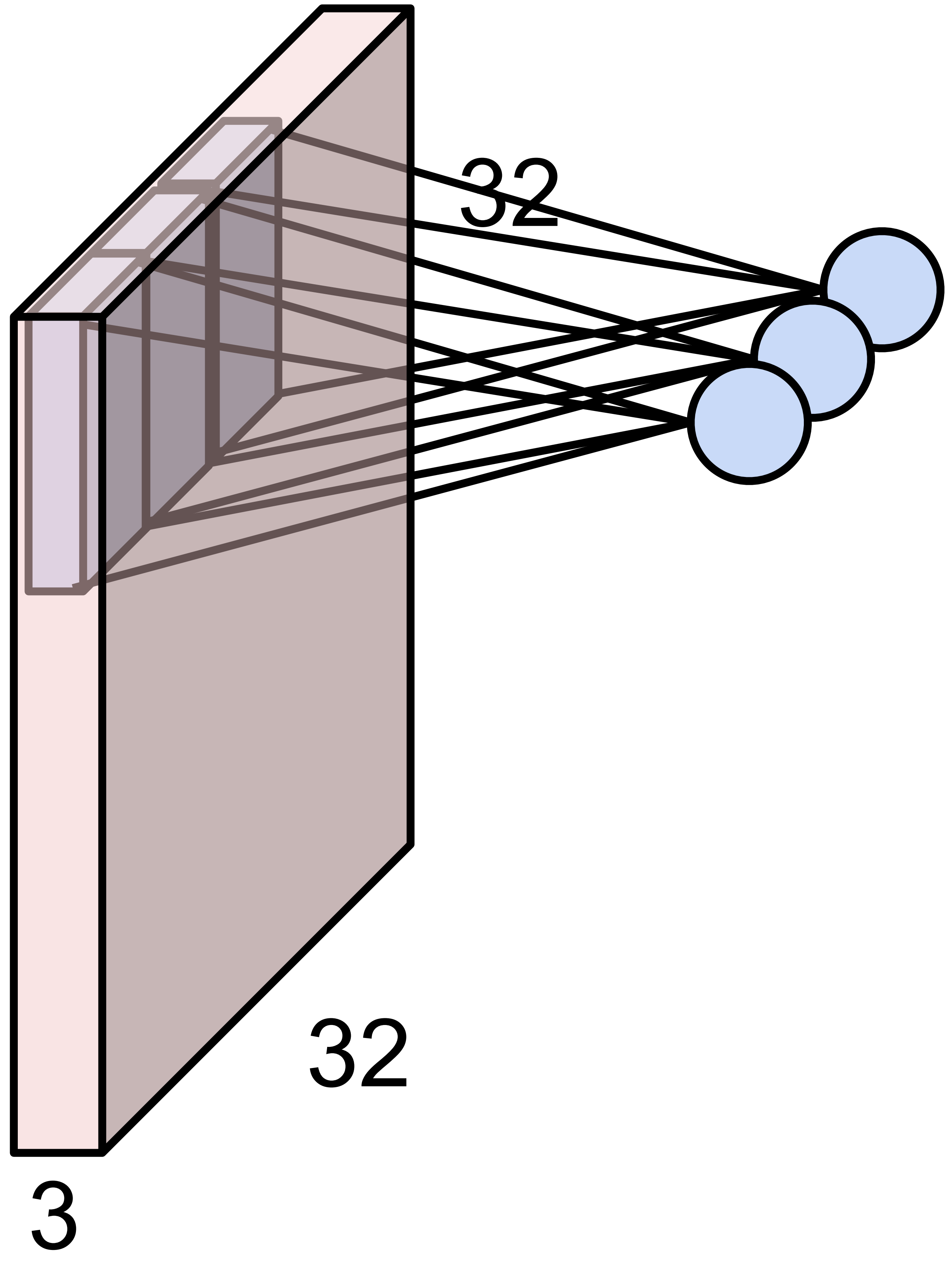
Convolution Layer



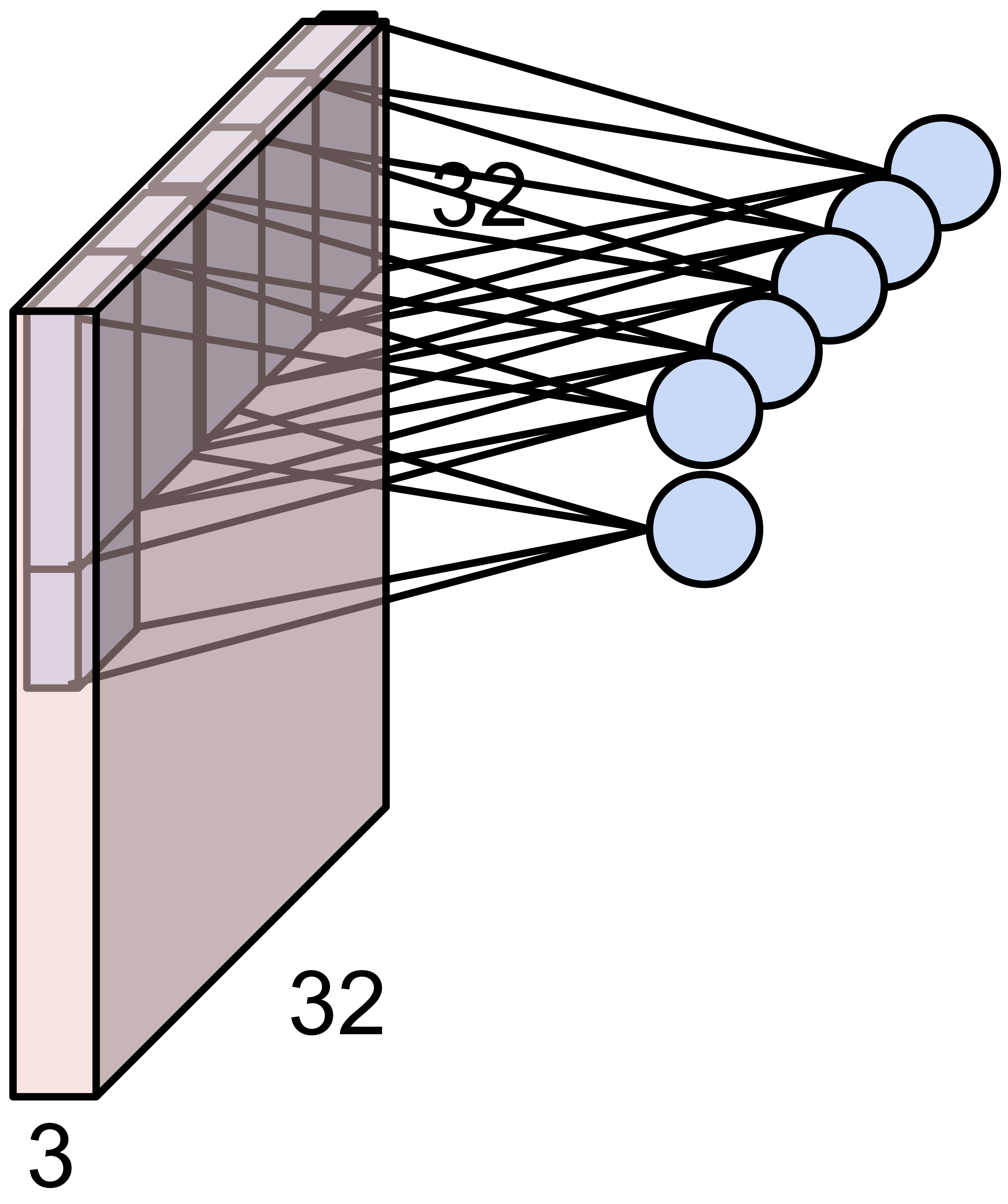
Convolution Layer



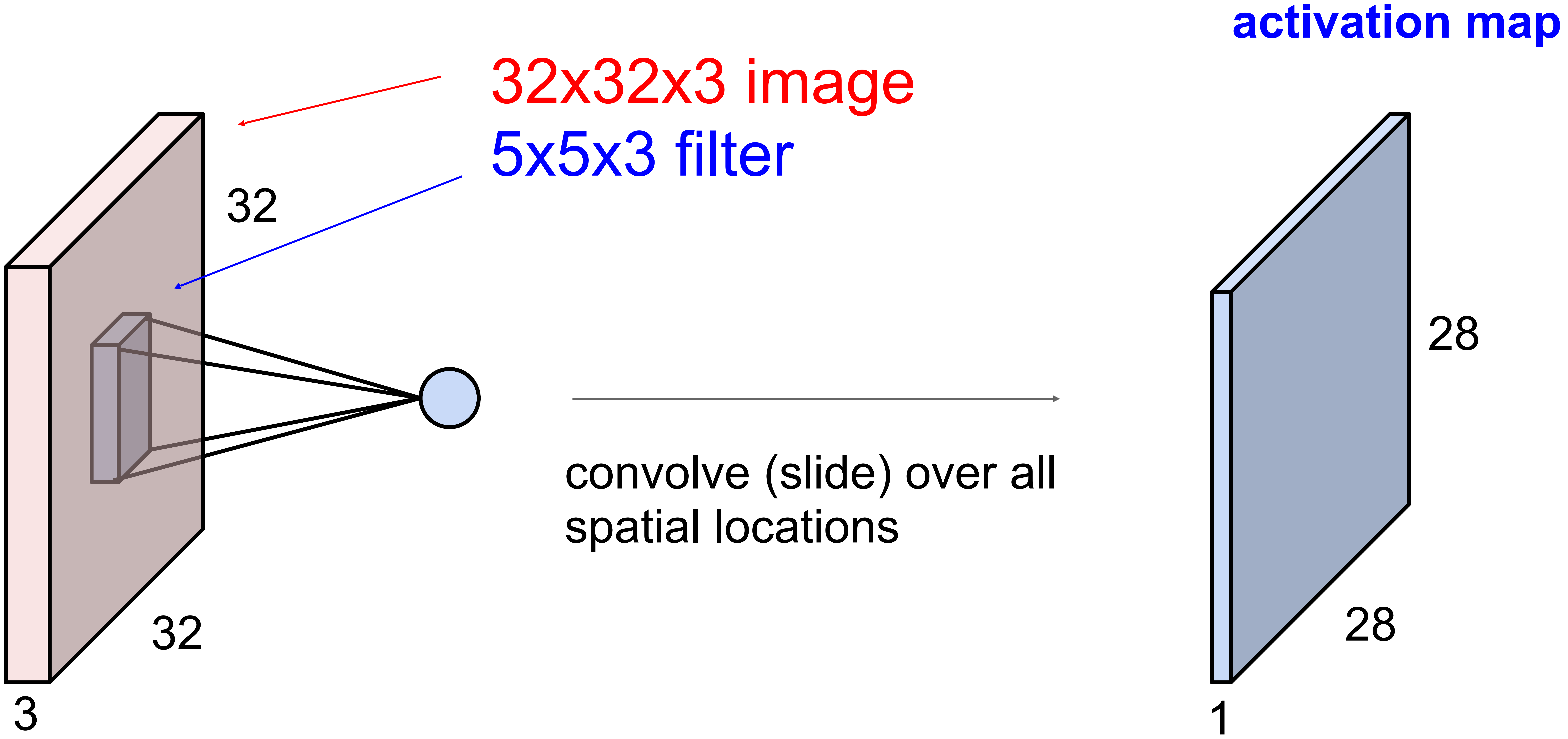
Convolution Layer



Convolution Layer

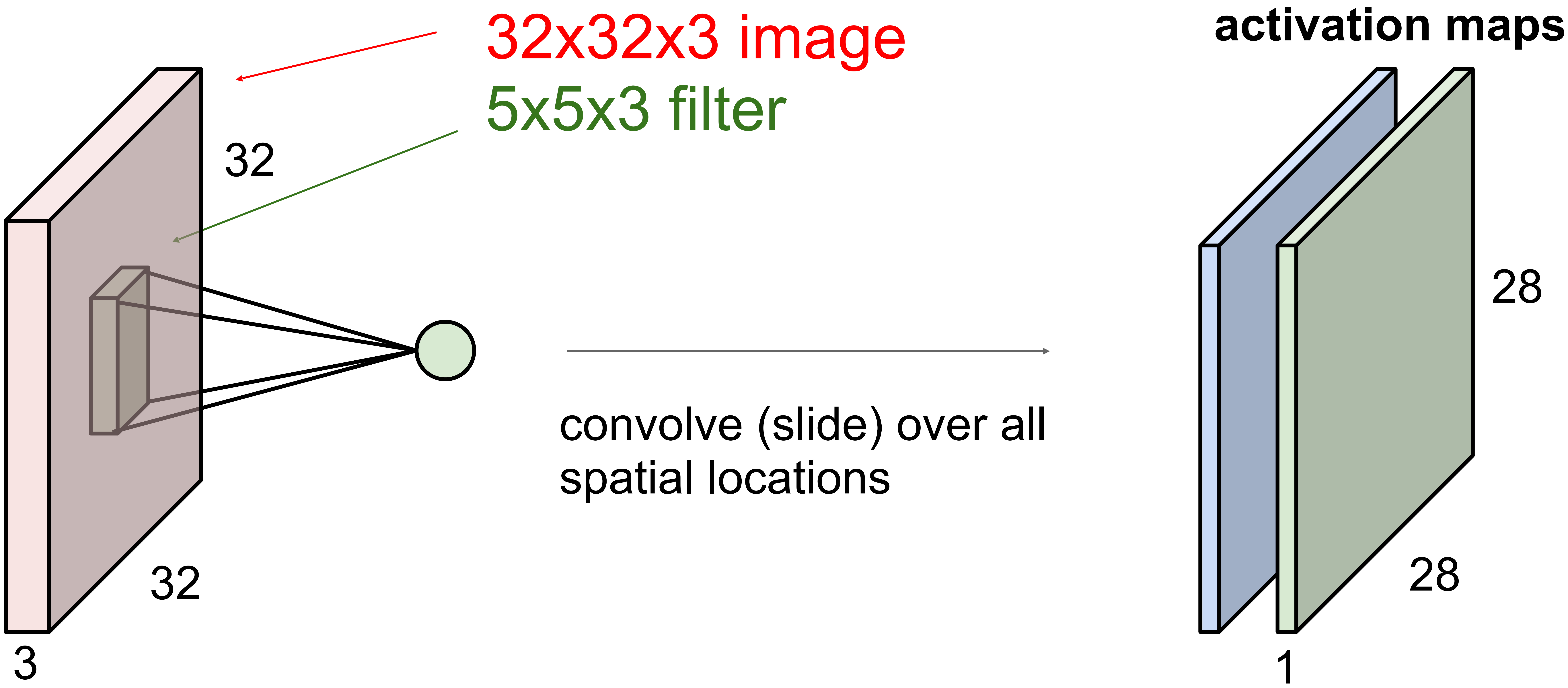


Convolution Layer

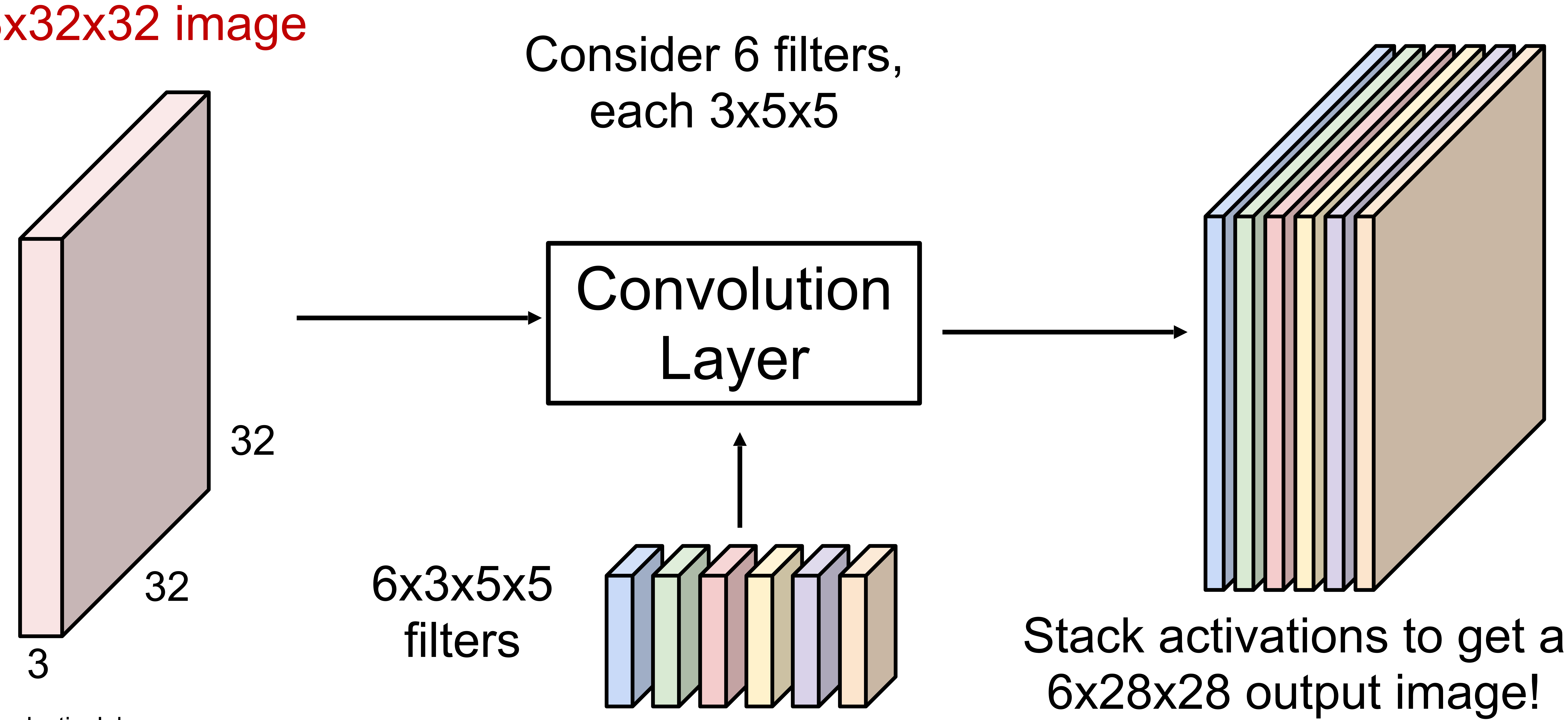


Convolution Layer

consider a second, green filter

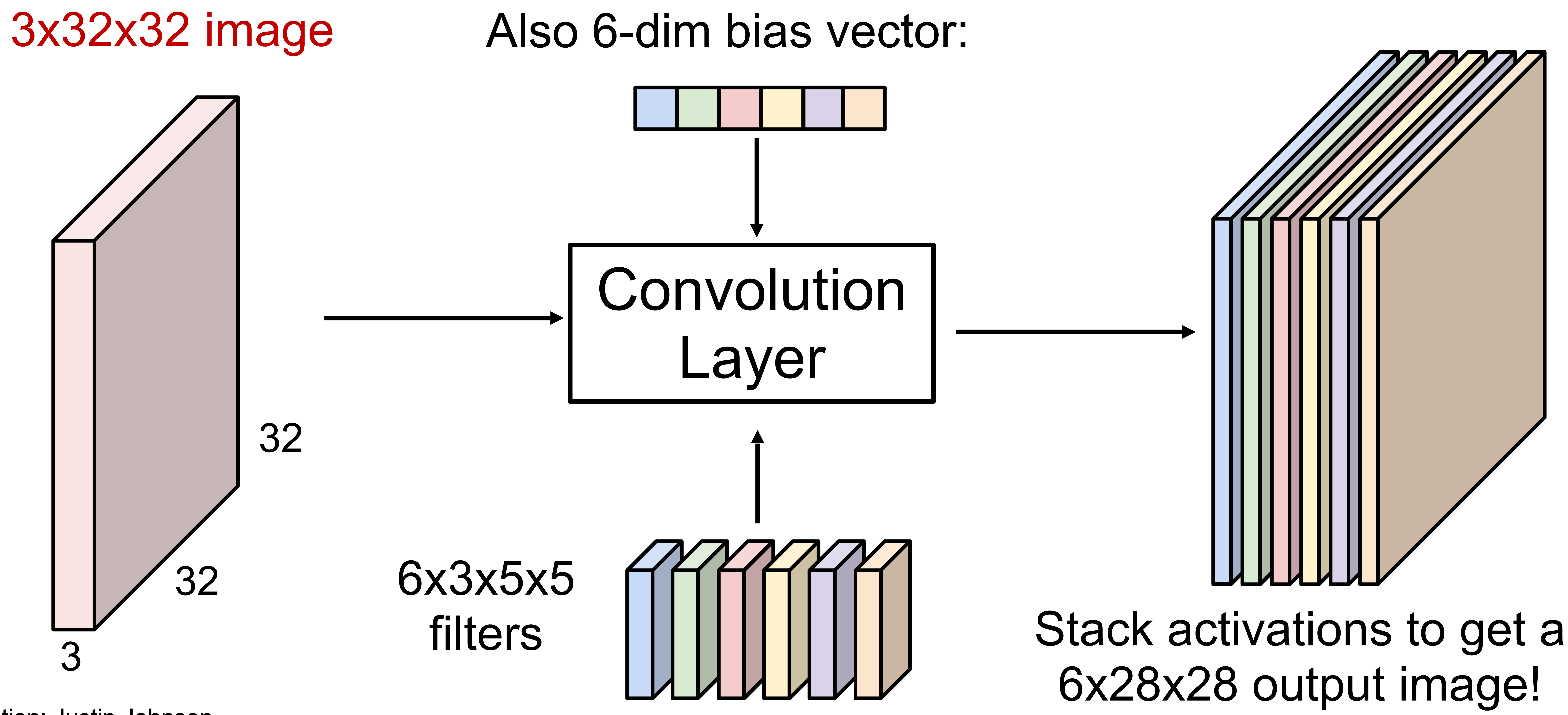


Convolution Layer



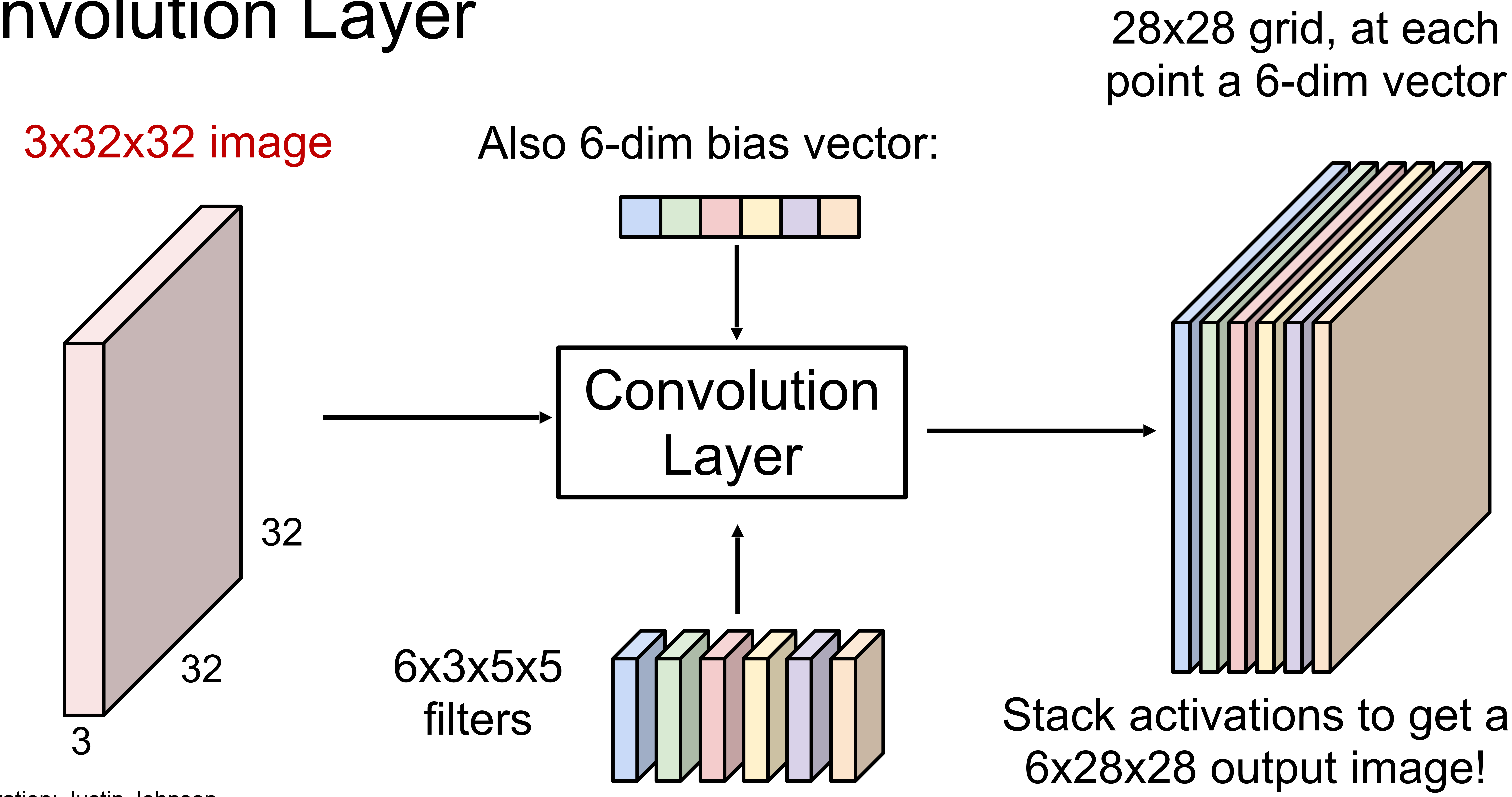
Slide inspiration: Justin Johnson

Convolution Layer



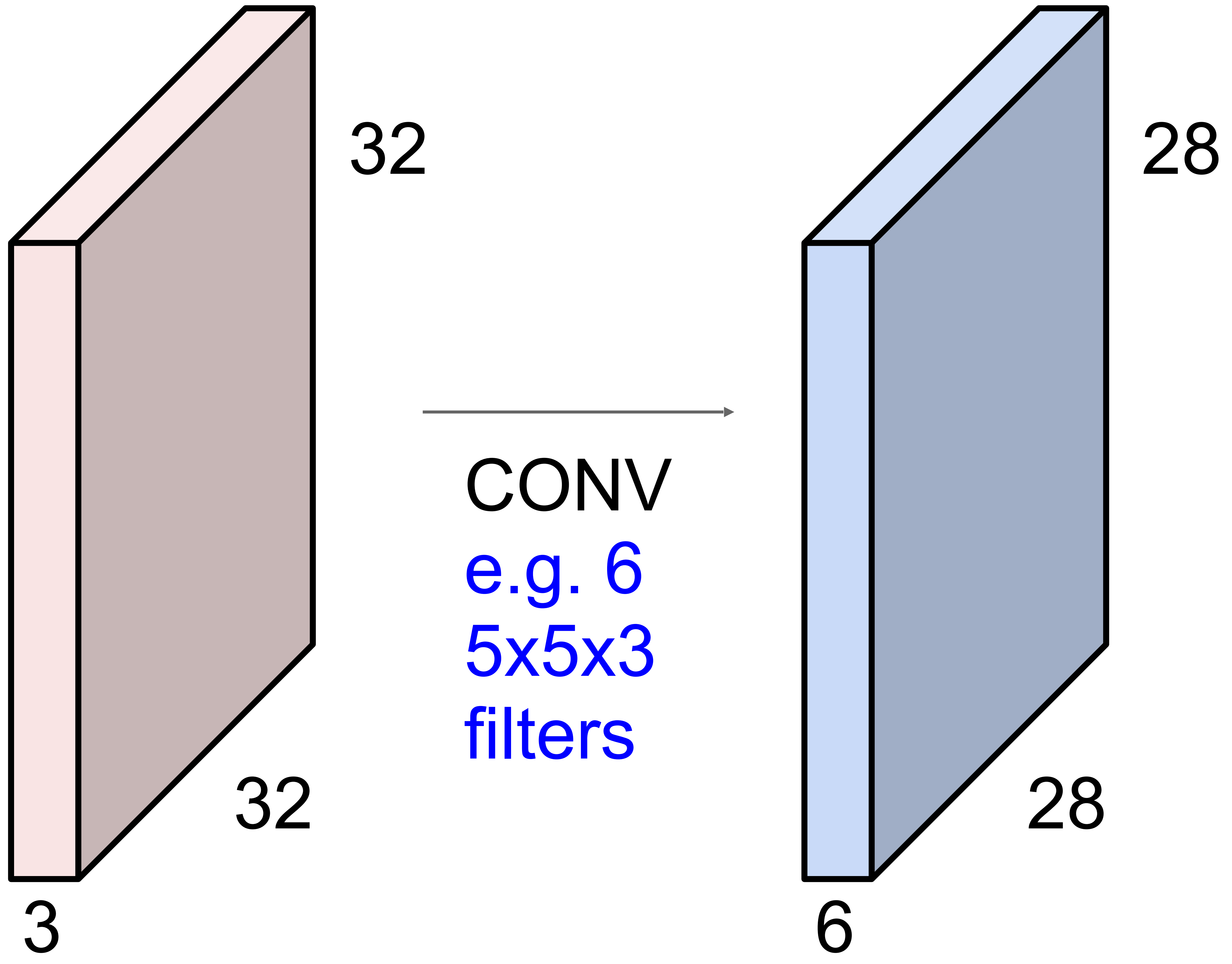
Slide inspiration: Justin Johnson

Convolution Layer

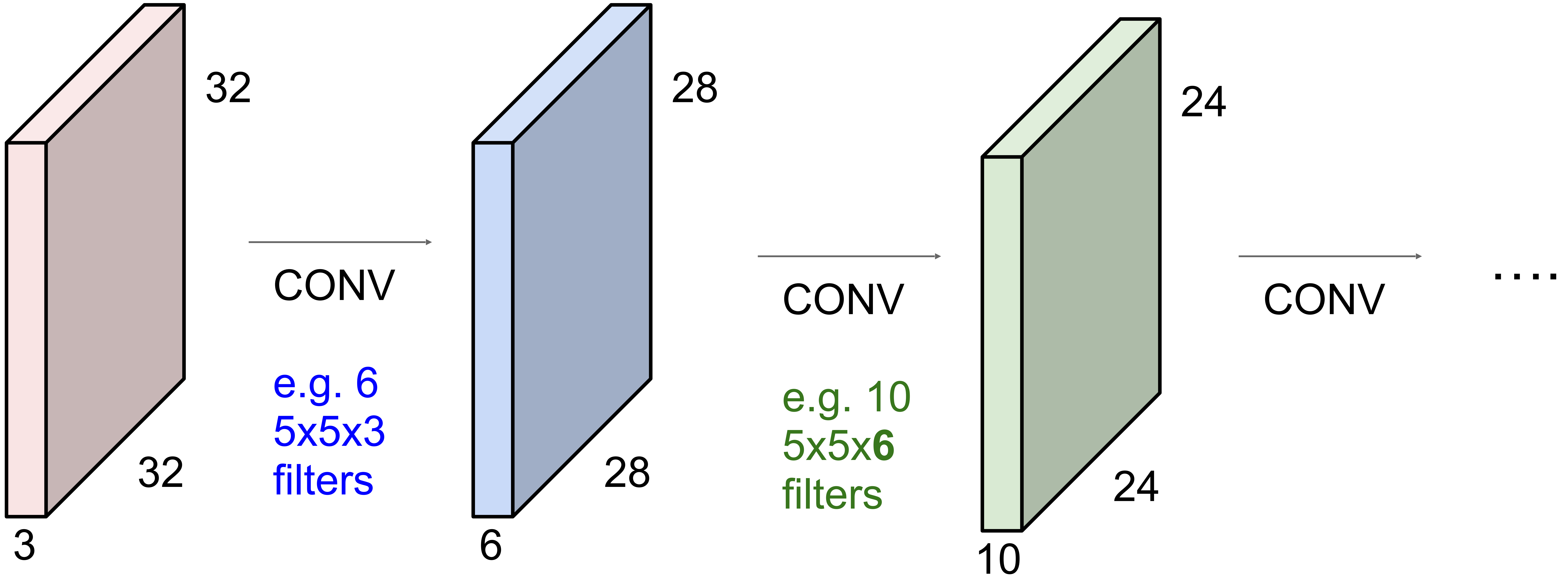


Slide inspiration: Justin Johnson

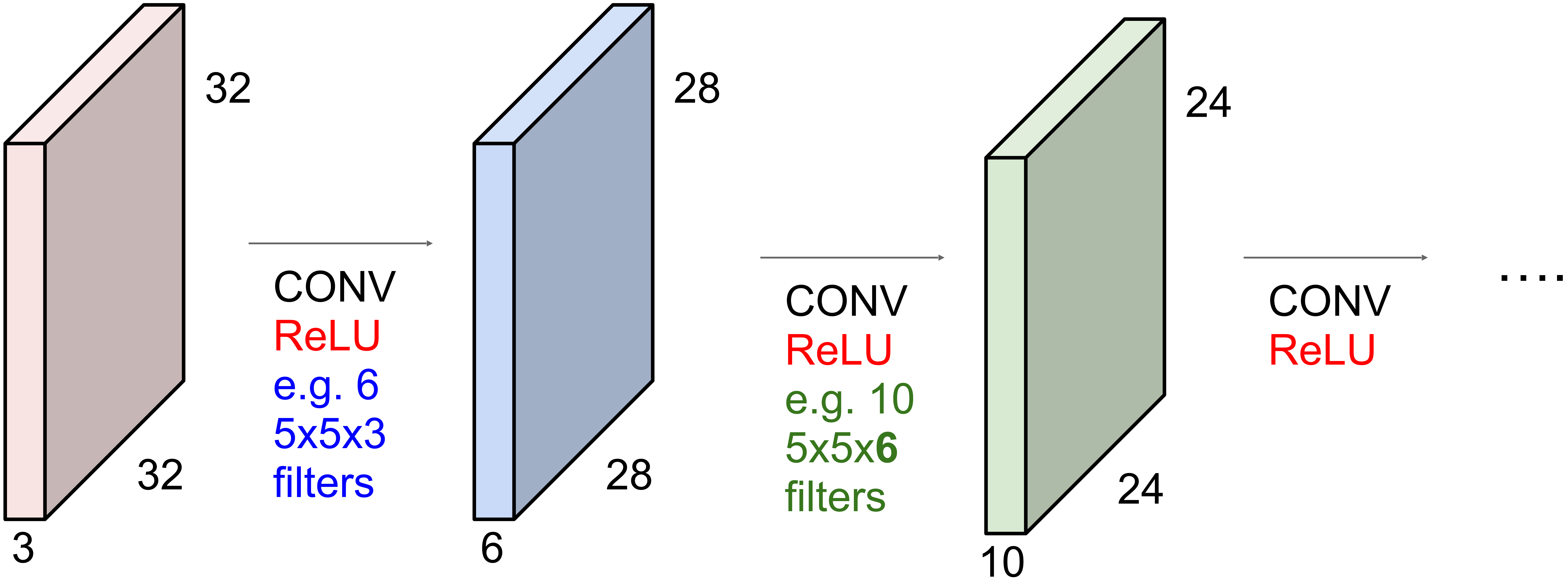
Preview: ConvNet is a sequence of Convolution Layers

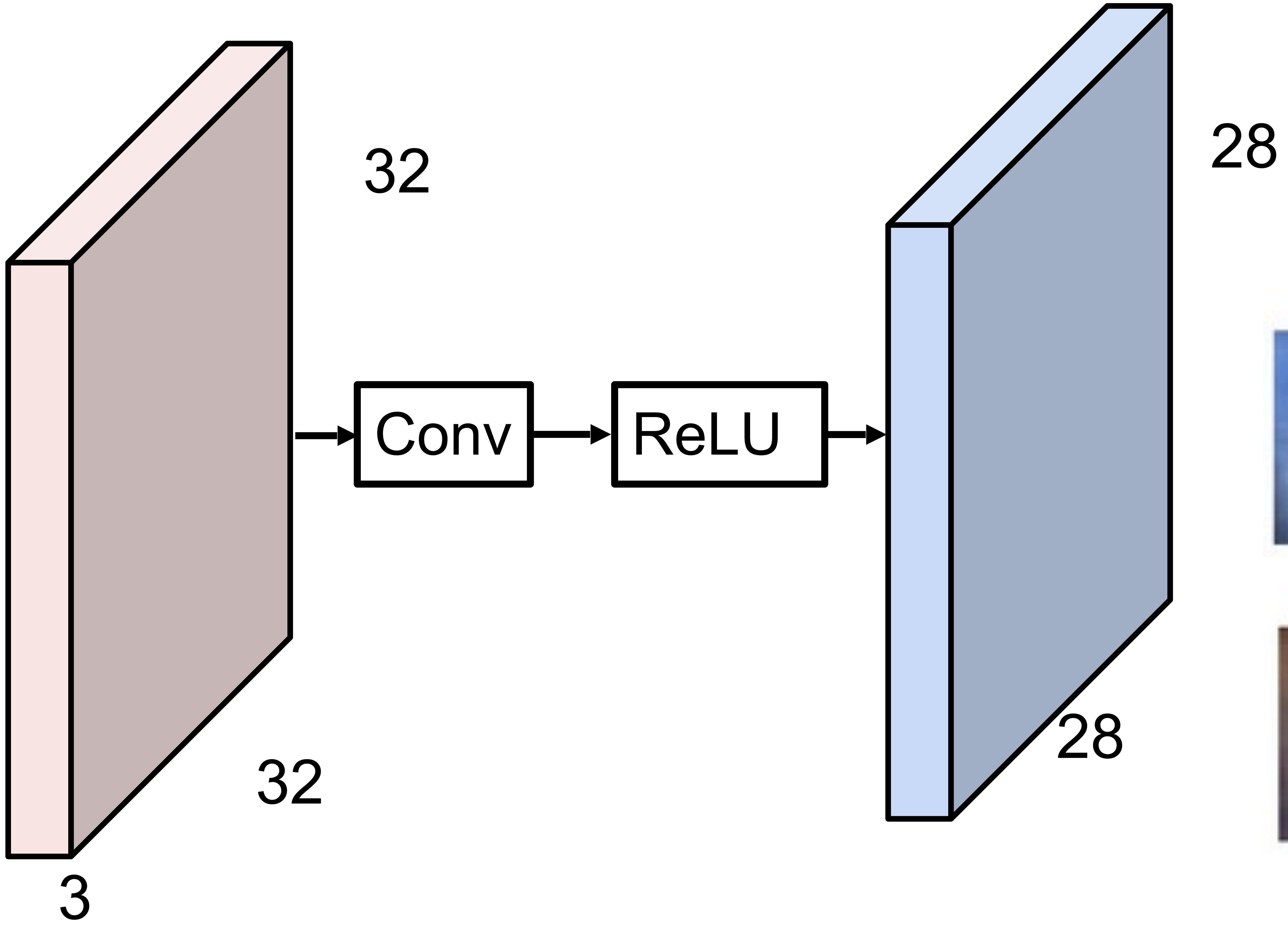


Preview: ConvNet is a sequence of Convolution Layers



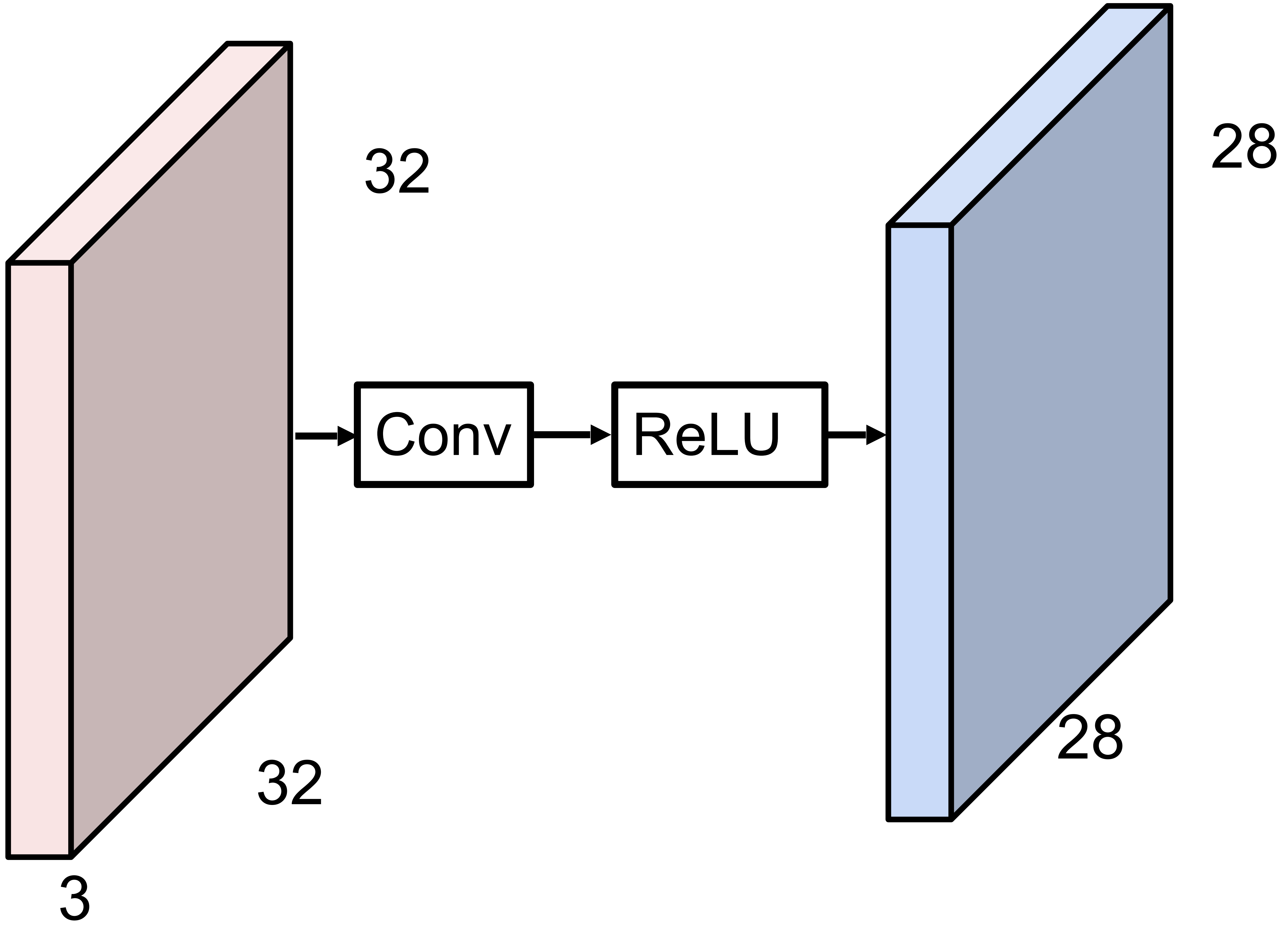
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



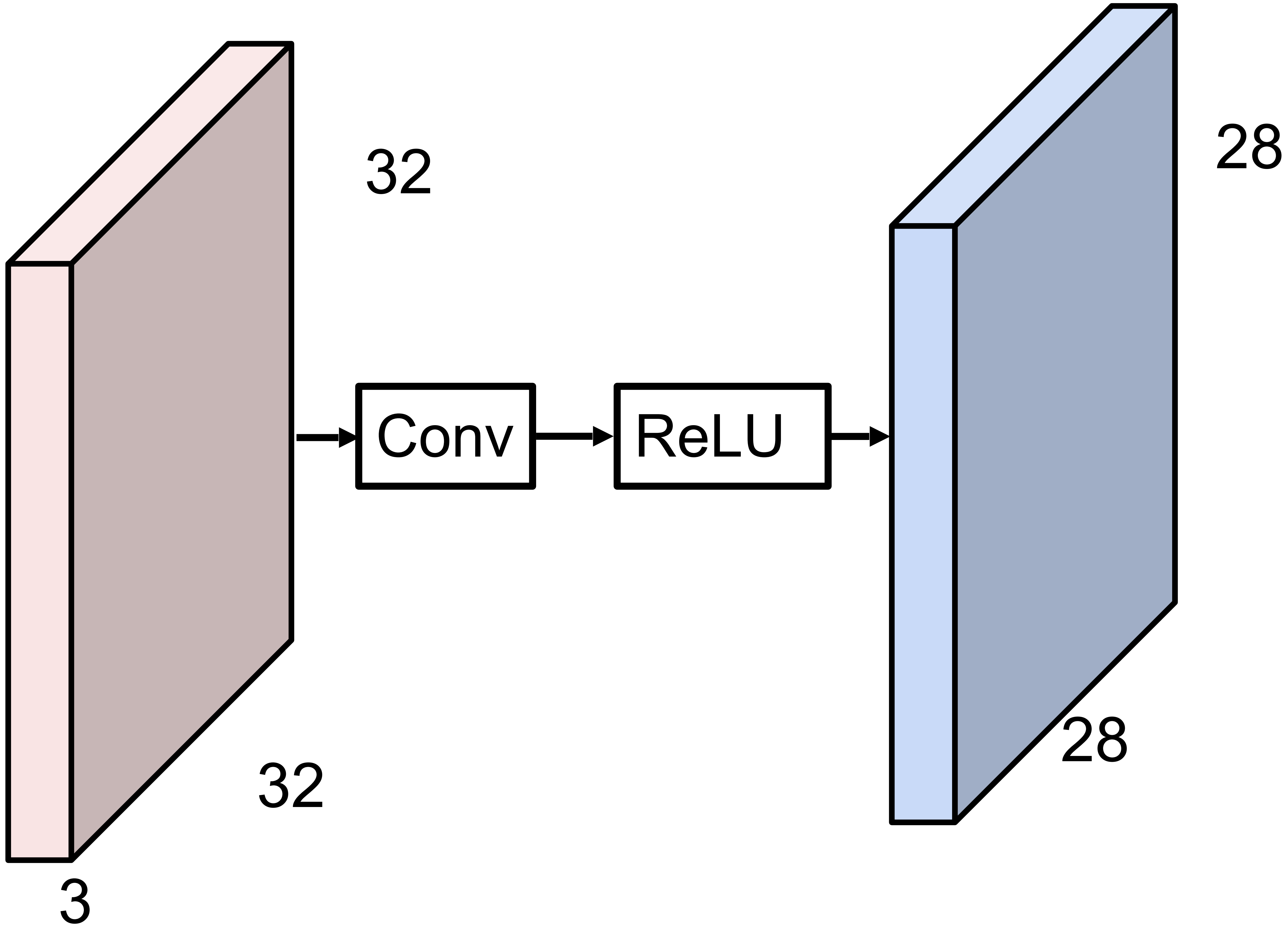


Linear classifier: One template per class

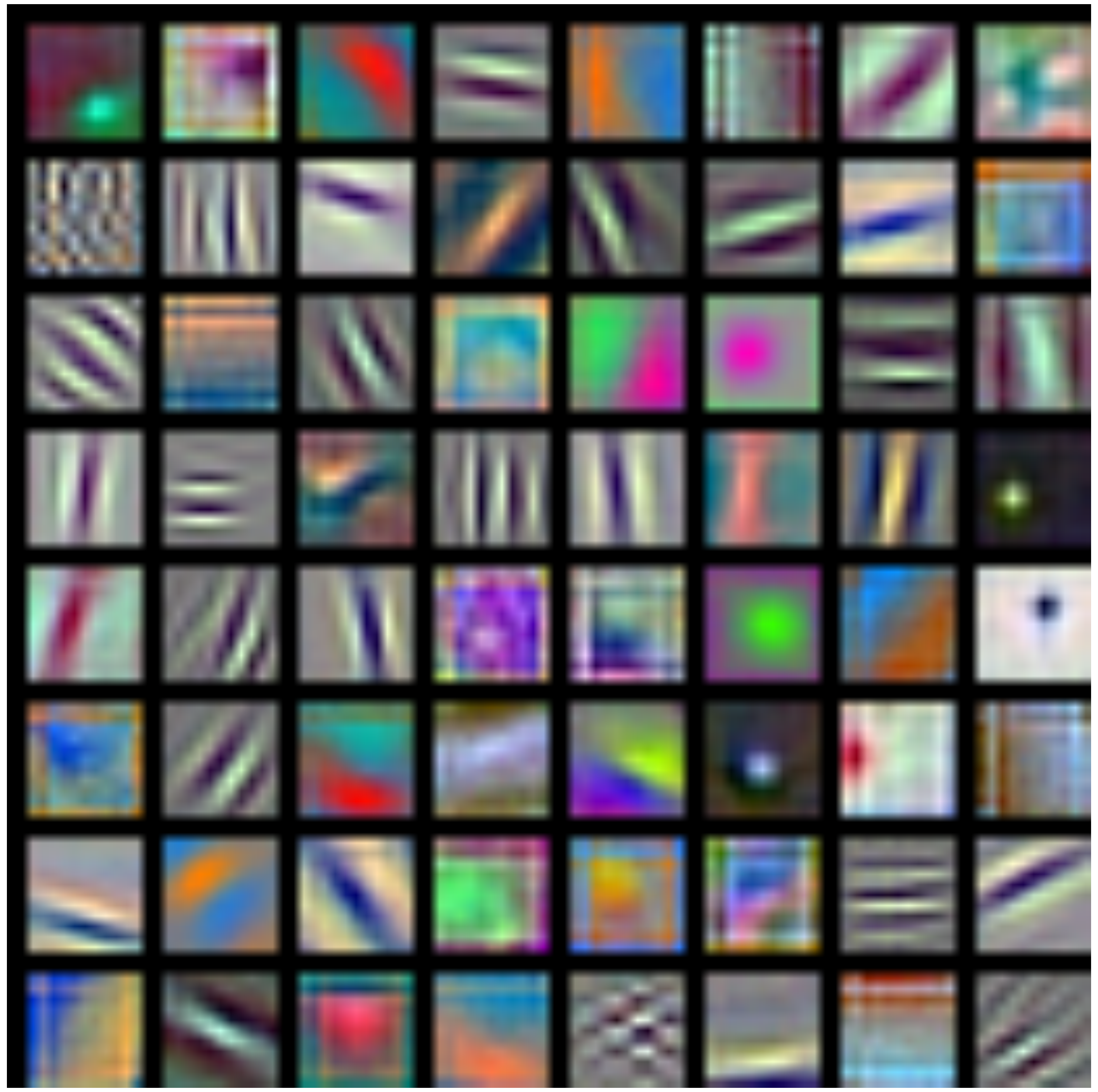
Preview: What do convolutional filters learn?



MLP: Bank of whole-image templates



First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11

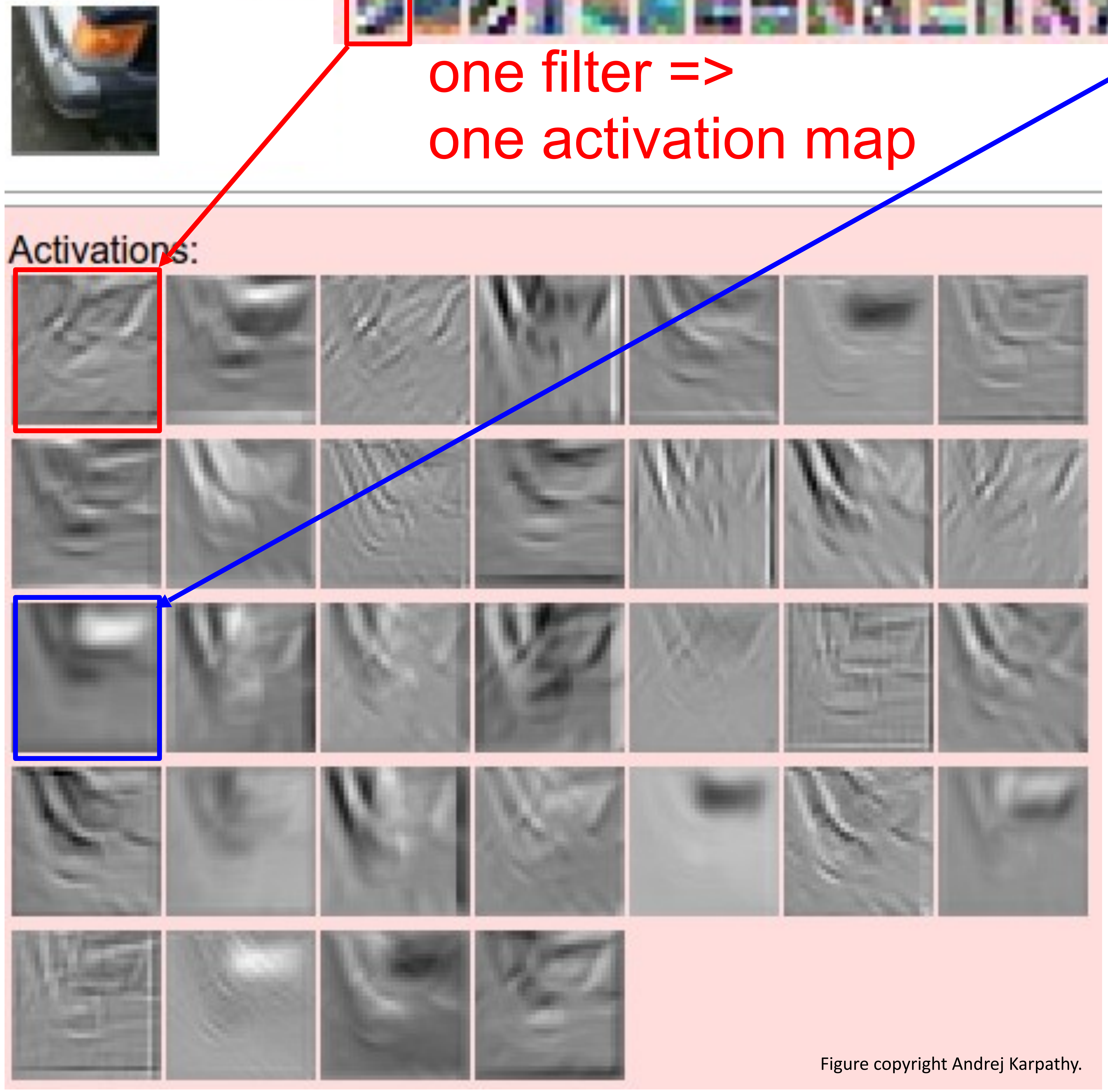


Figure copyright Andrej Karpathy.

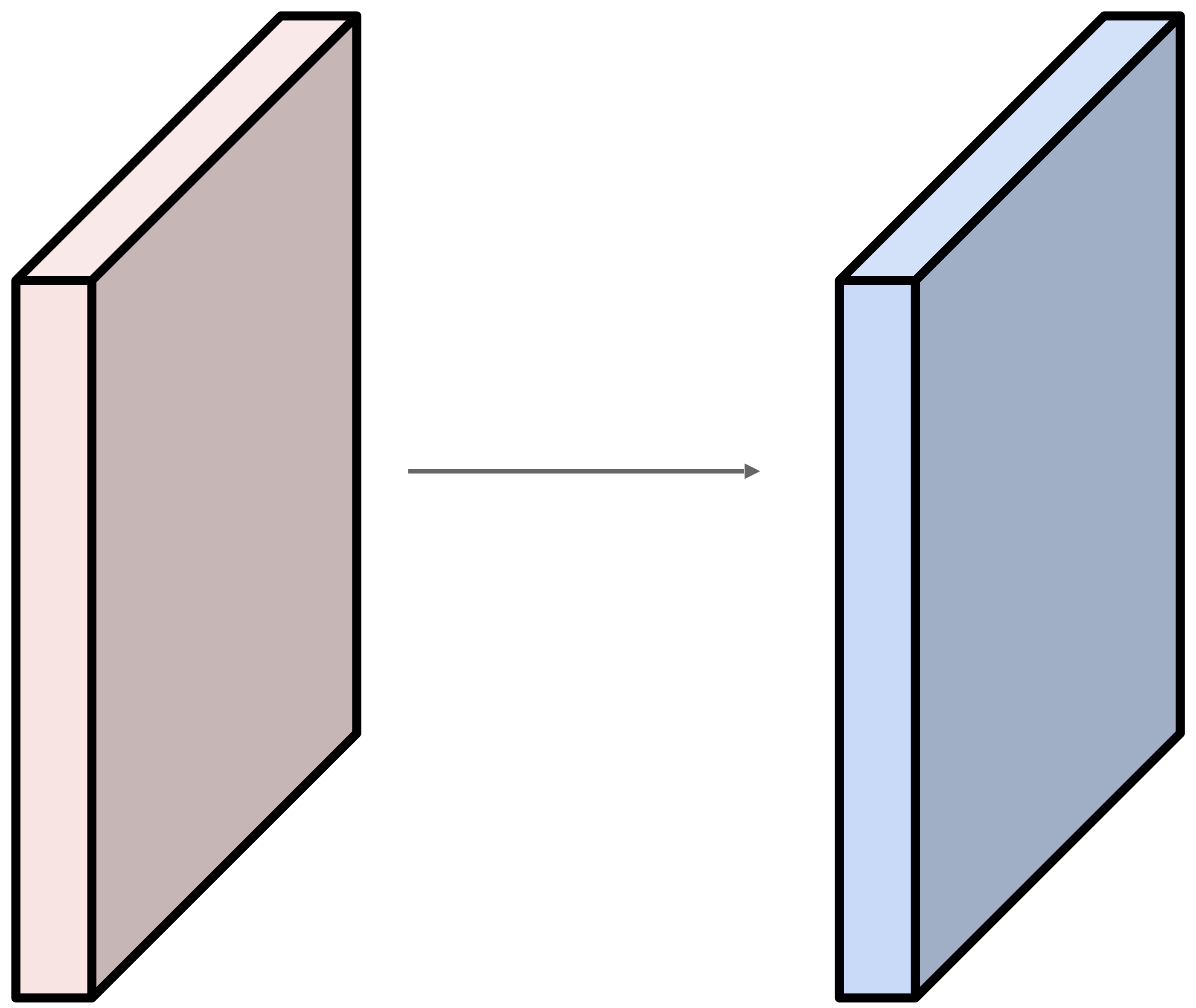
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y]*g[x,y] = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2]\cdot g[x-n_1,y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

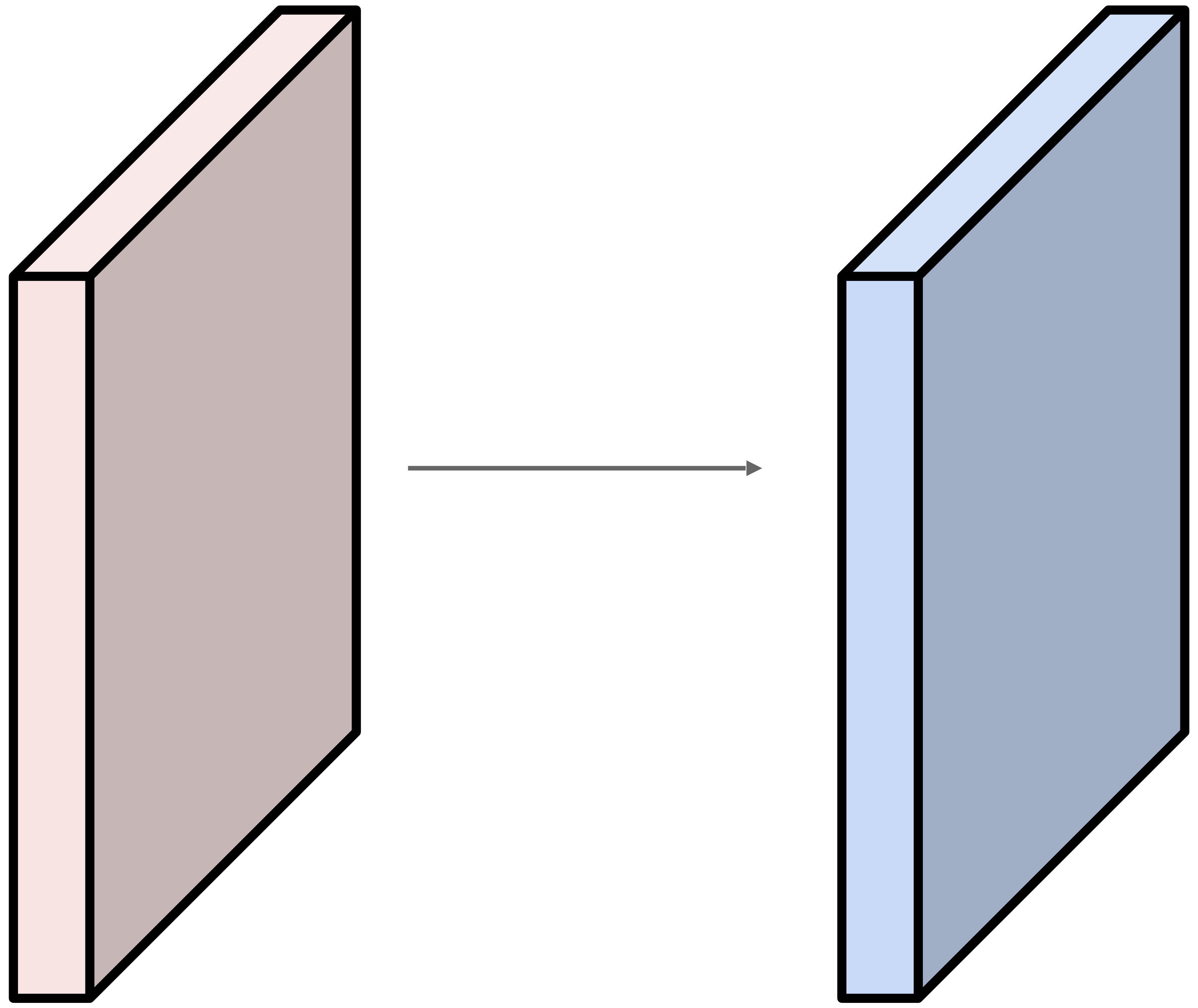
Examples time:



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

Number of parameters in this layer?

Examples time:

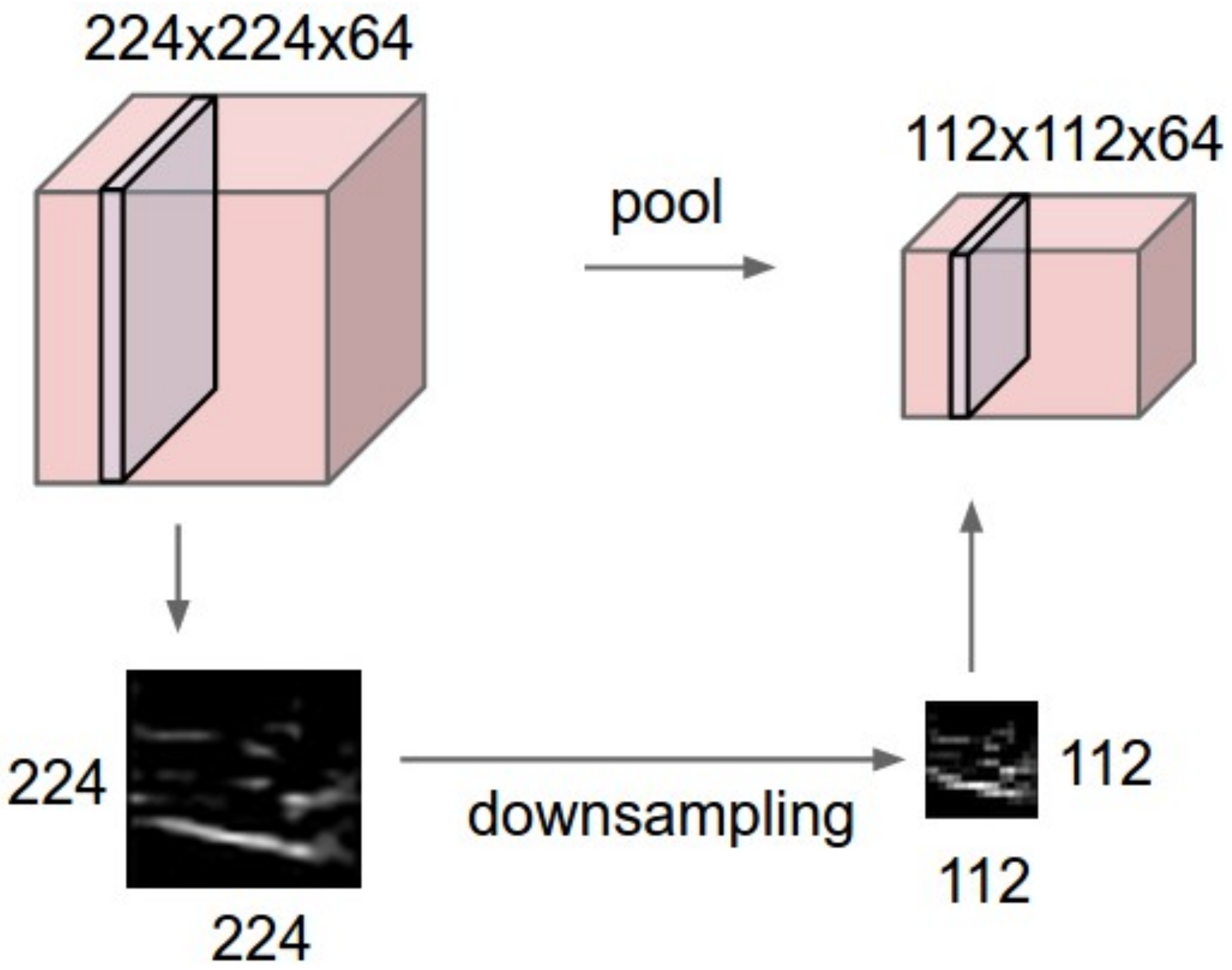


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently



MAX POOLING

Single depth slice

x

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

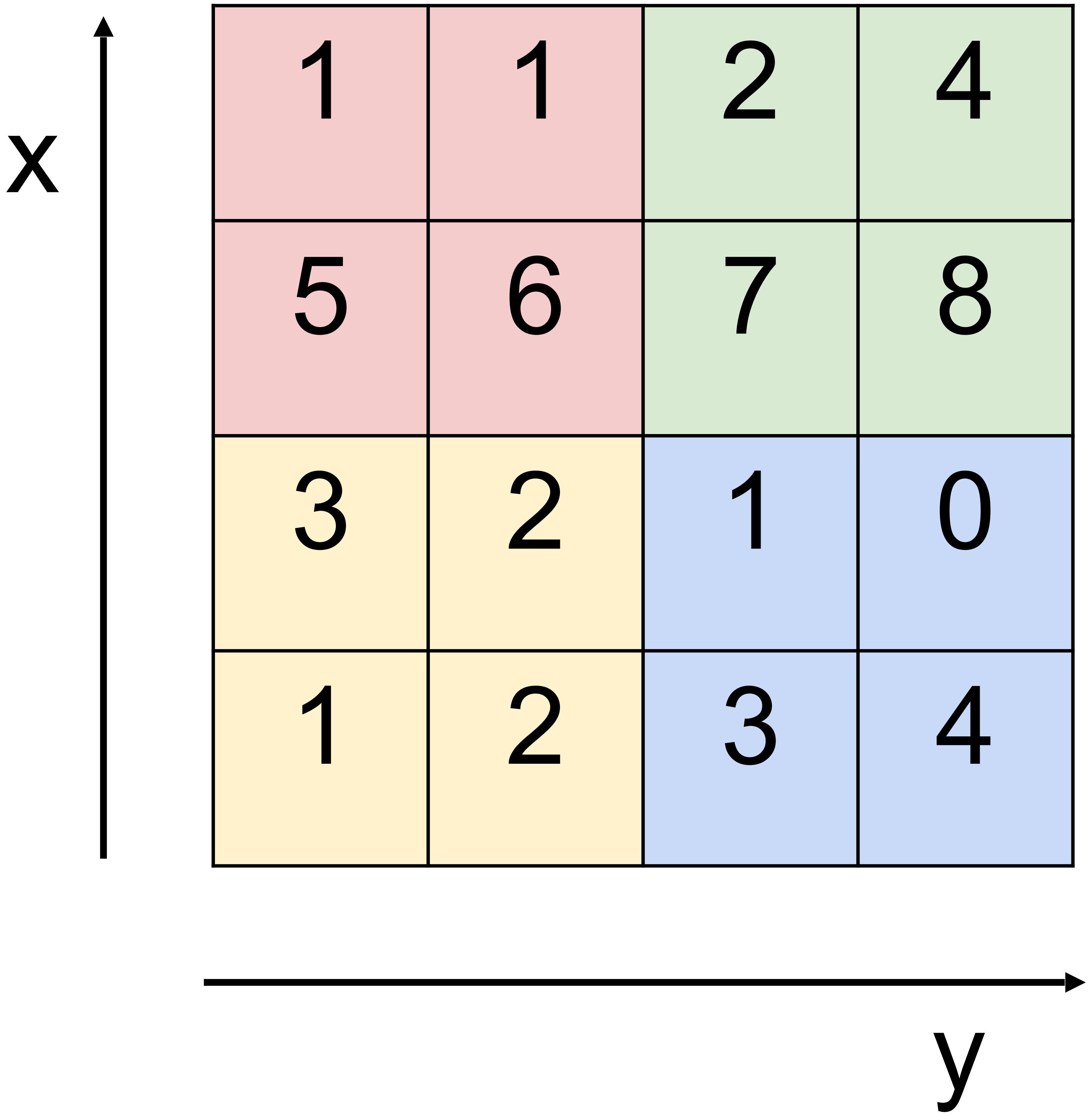
y

max pool with 2x2 filters
and stride 2

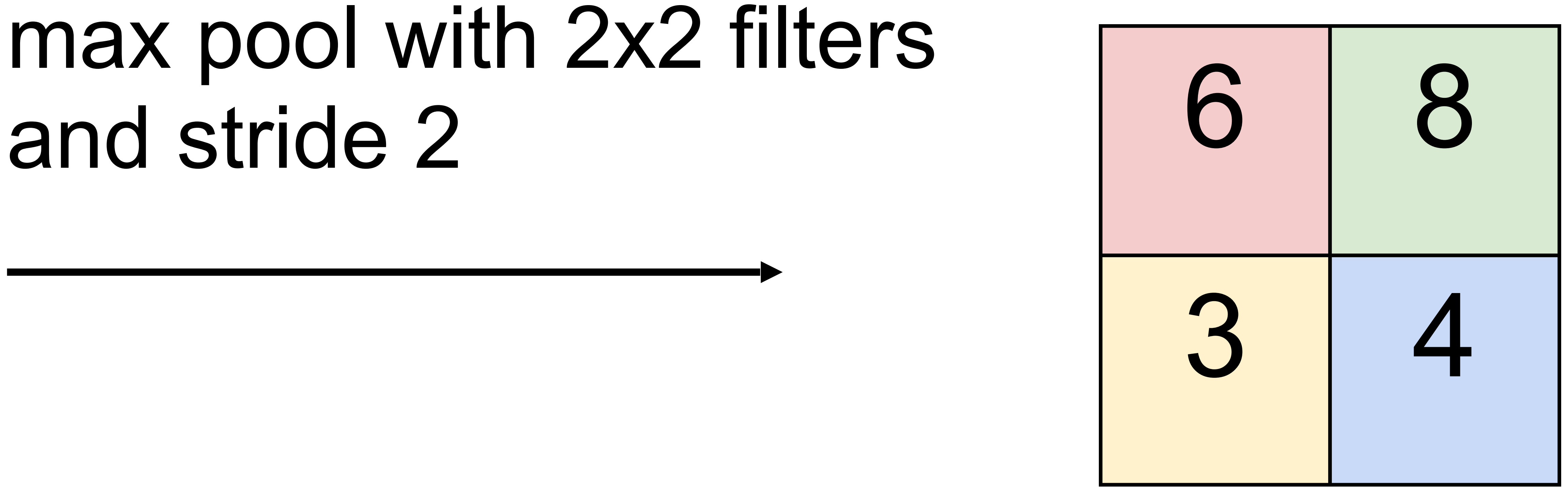
6	8
3	4

MAX POOLING

Single depth slice



max pool with 2x2 filters
and stride 2

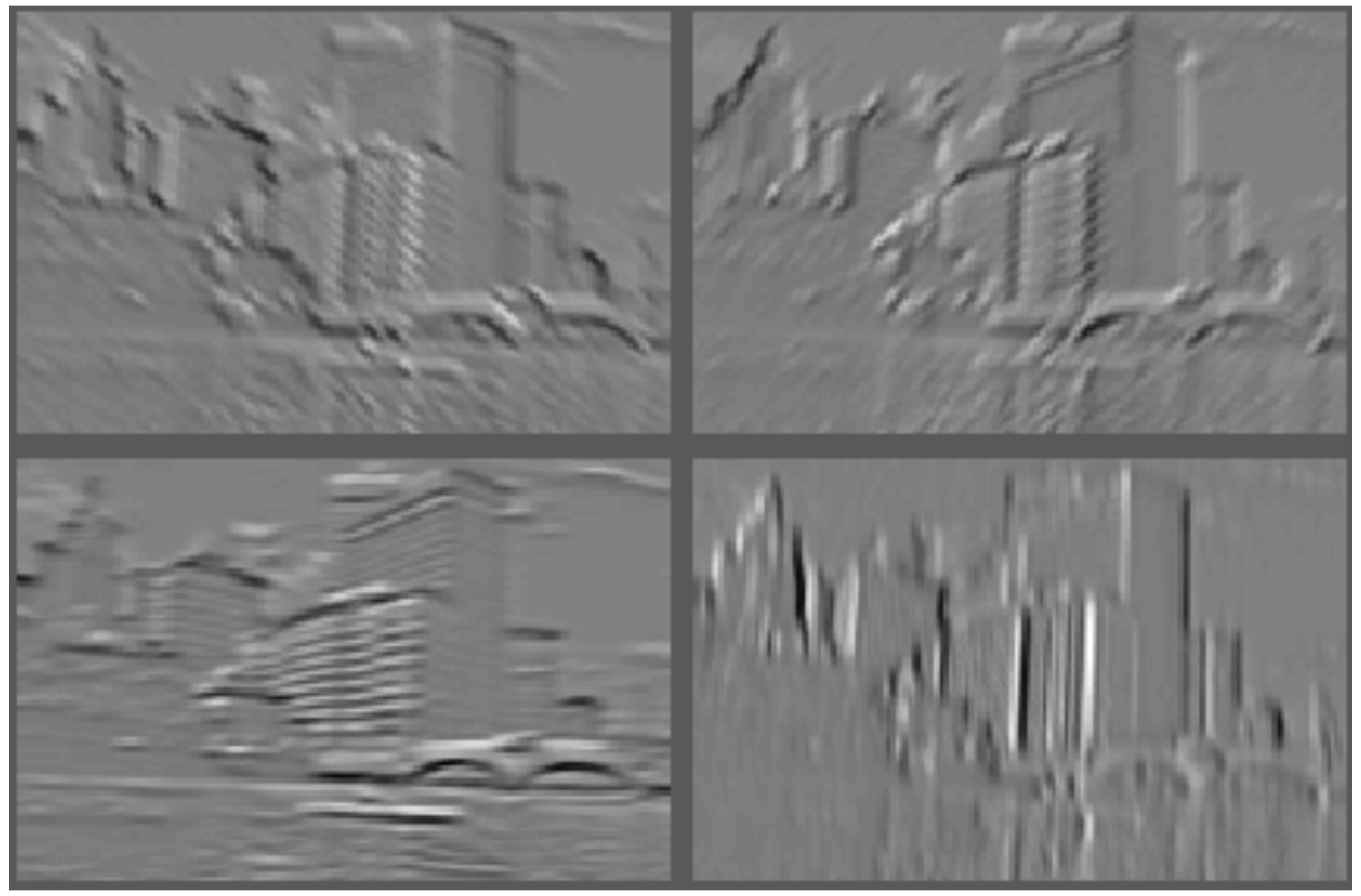


- No learnable parameters
- Introduces spatial invariance

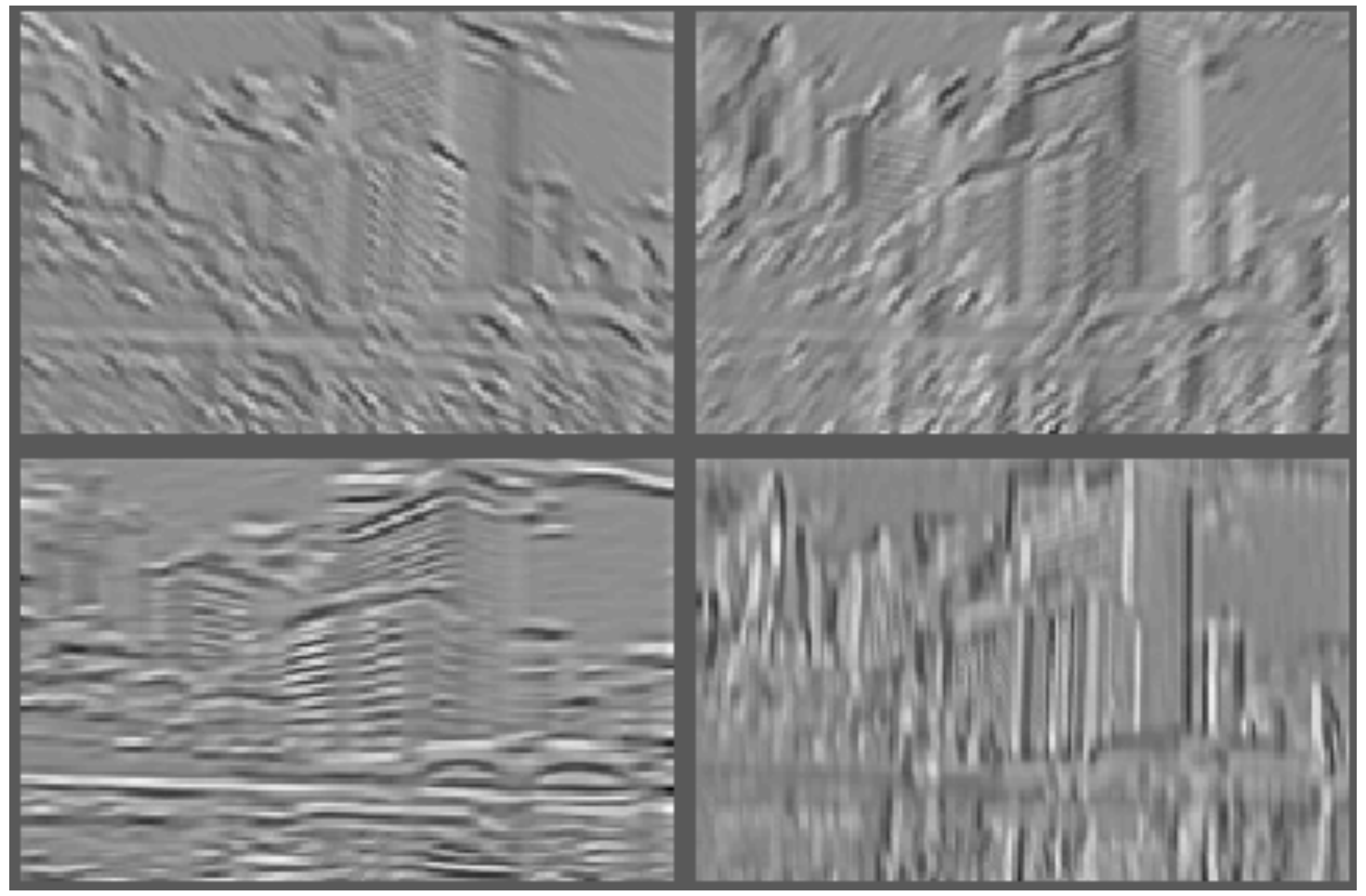
Normalization

Within or across feature maps

Before or after spatial pooling



Feature Maps



Feature Maps
After Contrast Normalization

Example: LeNet5

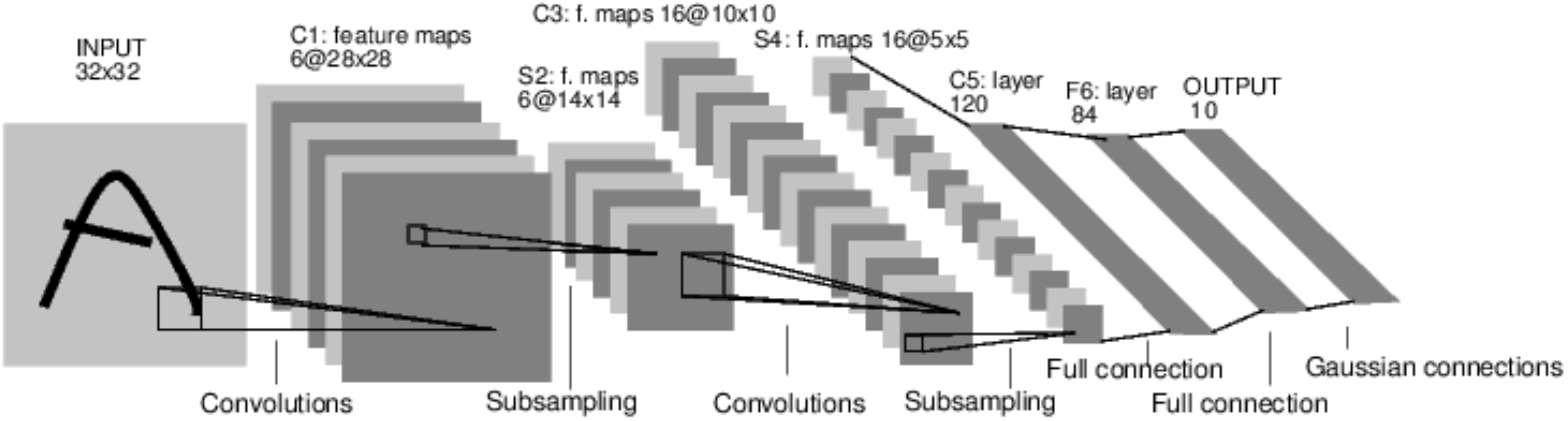


image → 6 5x5 → 2x2 → 16 6x6 → 2x2 → 120 5x5 → full → full

C1: Convolutional layer with 6 filters of size 5x5

Output: 6x28x28

Number of parameters: (5x5+1)*6 = 156

Connections: (5x5+1)x(6x28x28) = 122304

Connections in a fully connected network: (32x32+1)x(6X28x28)

LeCun 98

Example: LeNet5

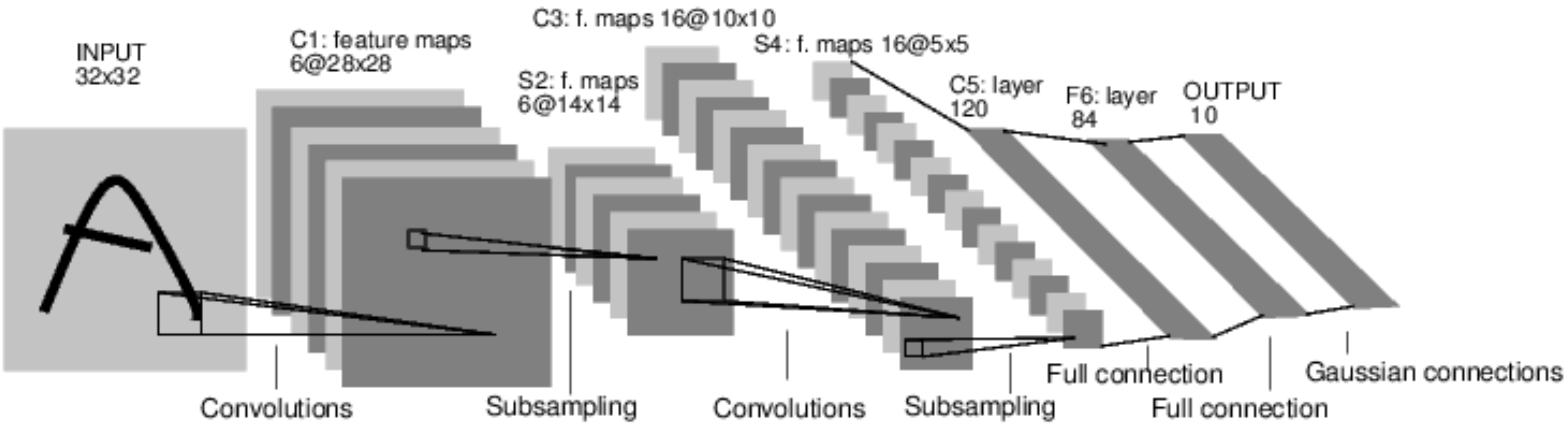


image ⟶ 6 5x5 ⟶ 2x2 ⟶ 16 6x6 ⟶ 2x2 ⟶ 120 5x5 ⟶ full ⟶ full

S2: Subsampling layer

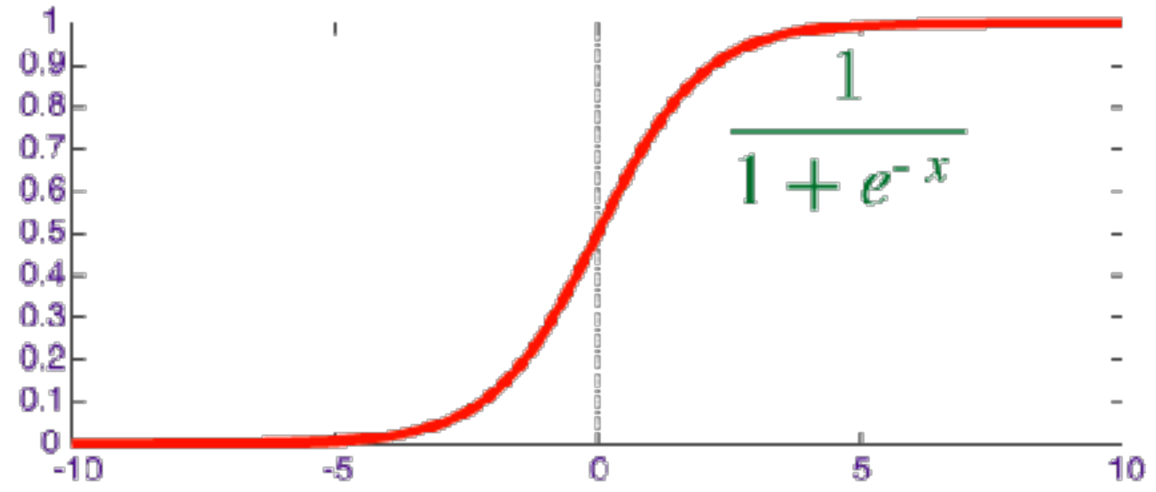
Subsample by taking the sum of non-overlapping 2x2 windows

- Multiply the sum by a constant and add bias

Number of parameters: 2x6=12

Pass the output through a sigmoid non-linearity

Output: 6x14x14



Example: LeNet5

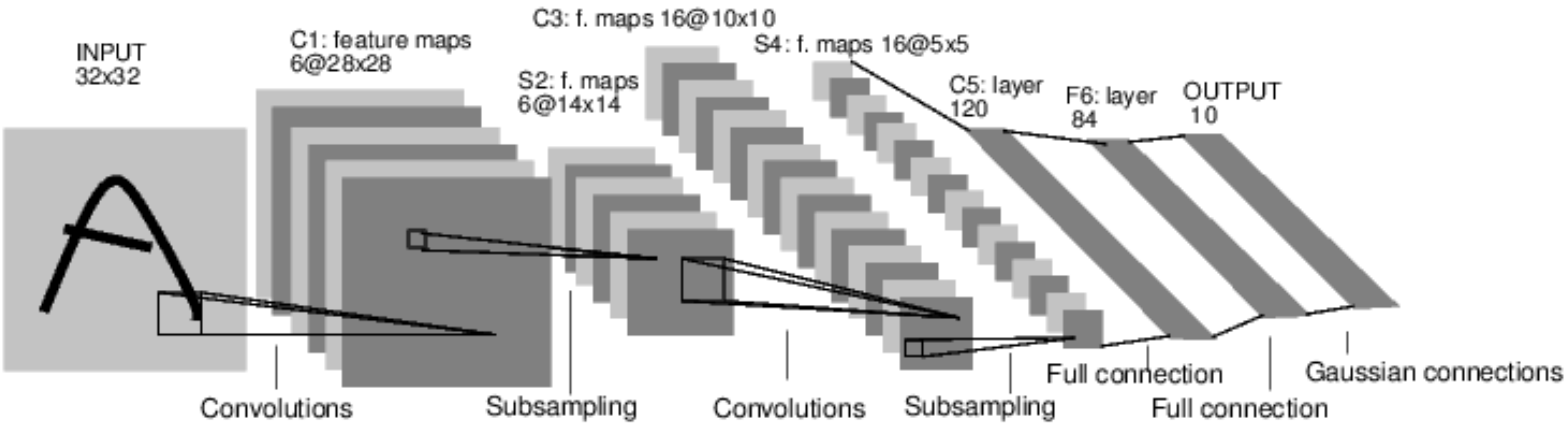


image ⟶ 6 5x5 ⟶ 2x2 ⟶ 16 6x6 ⟶ 2x2 ⟶ 120 5x5 ⟶ full ⟶ full

C3: Convolutional layer with 16 filters of size 6x6

Each is connected to a subset:

Number of parameters: 1,516

Number of connections: 151,600

Output: 16x10x10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Example: LeNet5

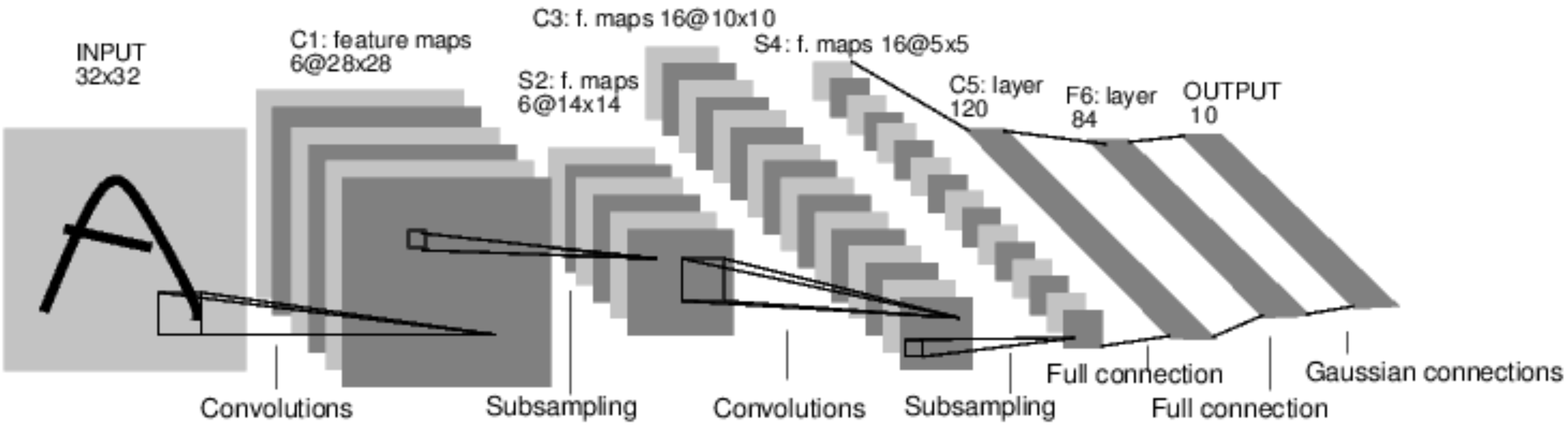


image ⟶ 6 5x5 ⟶ 2x2 ⟶ 16 6x6 ⟶ 2x2 ⟶ 120 5x5 ⟶ full ⟶ full

S4: Subsampling layer

Subsample by taking the sum of non-overlapping 2x2 windows

- Multiply by a constant and add bias

Number of parameters: 2x16 = 32

Pass the output through a sigmoid non-linearity

Output: 16x5x5

Example: LeNet5

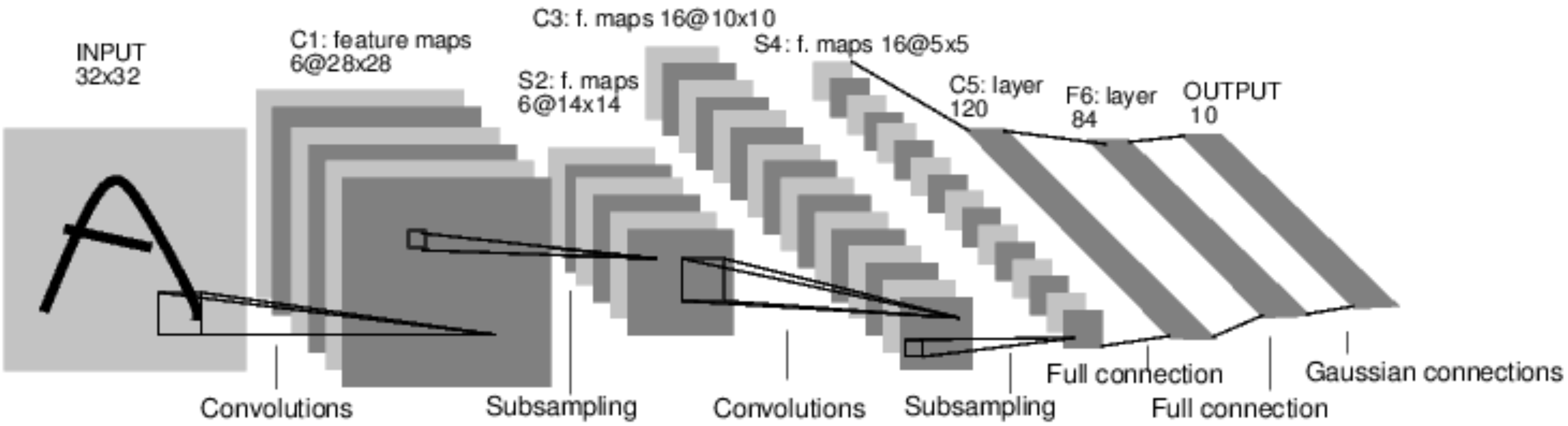


image ⟶ 6 5x5 ⟶ 2x2 ⟶ 16 6x6 ⟶ 2x2 ⟶ 120 5x5 ⟶ full ⟶ full

C5: Convolutional layer with 120 outputs of size 1x1

Each unit in C5 is connected to all inputs in S4

Number of parameters: (16x5x5+1)*120 = 48120

Example: LeNet5

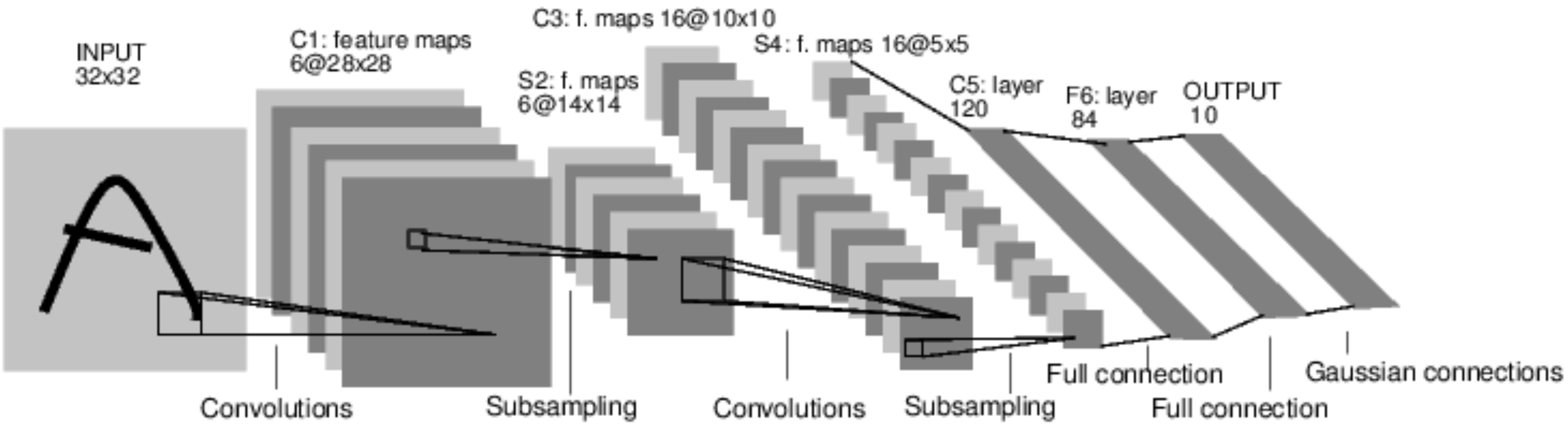


image ⟶ 6 5x5 ⟶ 2x2 ⟶ 16 6x6 ⟶ 2x2 ⟶ 120 5x5 ⟶ full ⟶ full

F6: fully connected layer

Output: 1x1x84

Number of parameters: (120+1)*84 = 10164

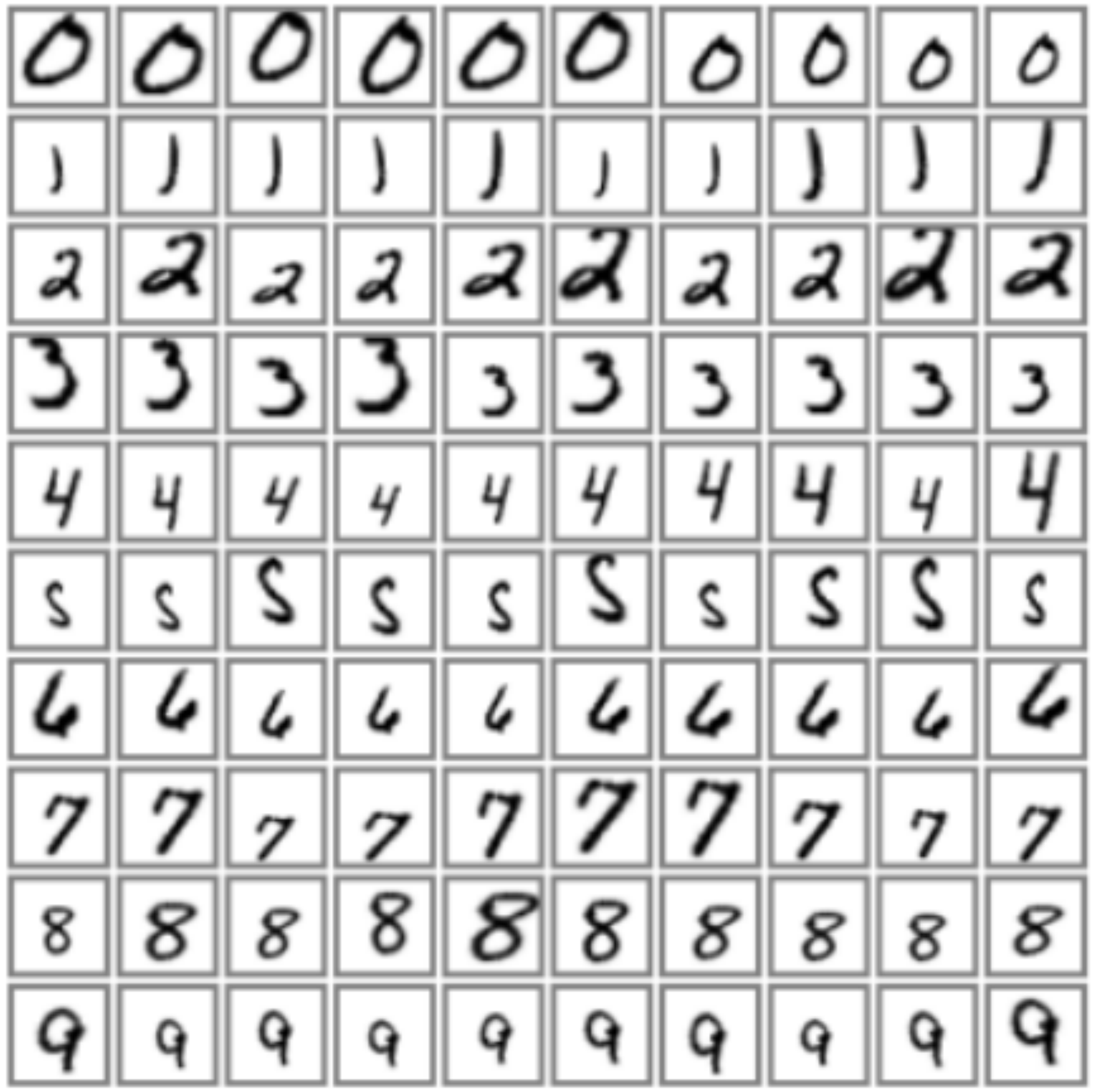
OUTPUT: 10 Euclidean RBF (Gaussian) units (one for each class)

$$y_i = \sum_j (x_j - w_{ij})^2.$$

MNIST dataset

60,000 original datasets

Test error: 0.95%



540,000 artificial distortions

+ 60,000 original

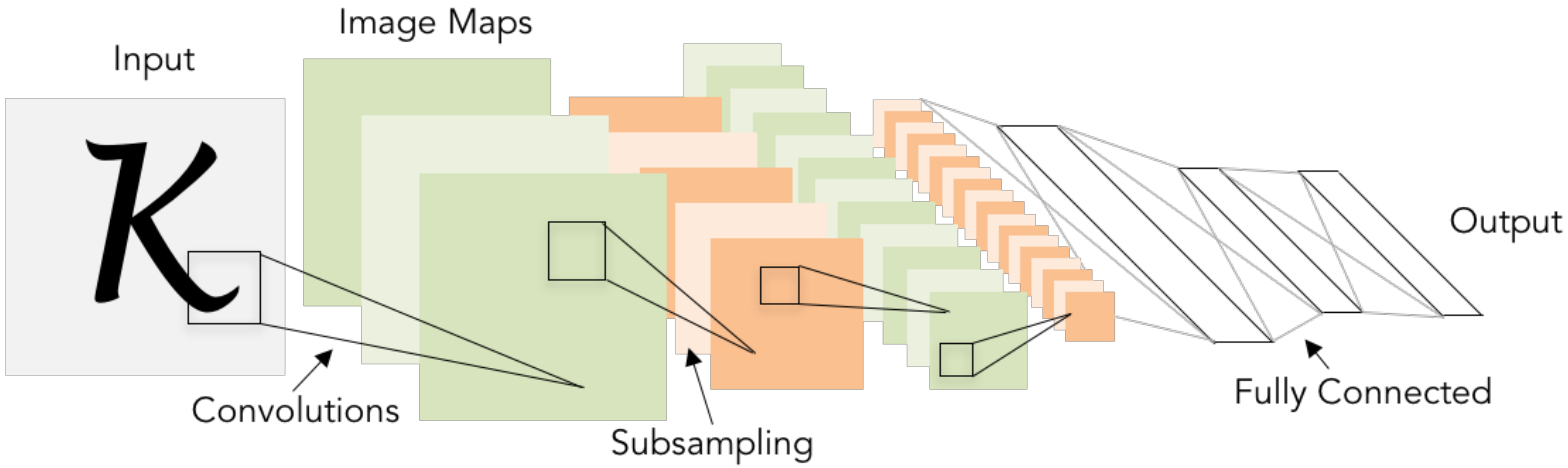
Test error: 0.8%

3-layer NN, 300+100 HU [distortions]
Test error: 2.5%

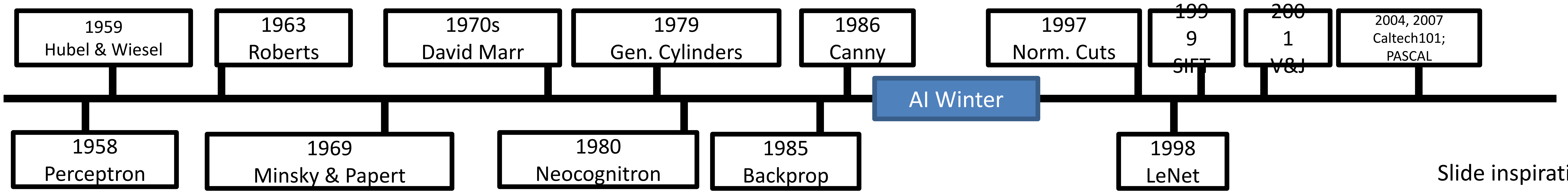
http://yann.lecun.com/exdb/mnist/

MNIST dataset: errors on the test set

Convolutional Networks: LeCun et al, 1998



Applied backprop algorithm to a Neocognitron-like architecture
Learned to recognize handwritten digits
Was deployed in a commercial system by NEC, processed handwritten checks
Very similar to our modern convolutional networks!



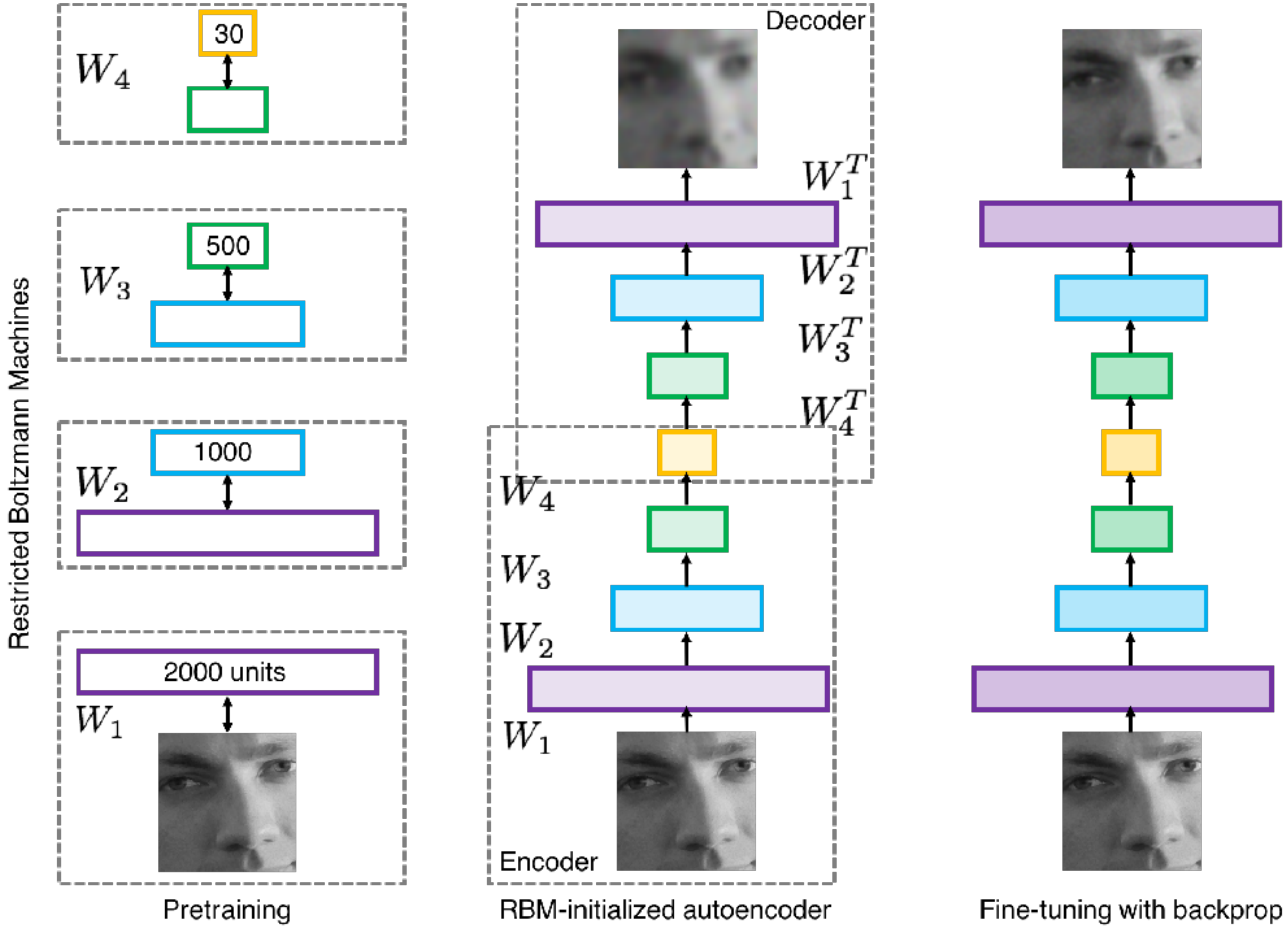
Slide inspiration: Justin Johnson

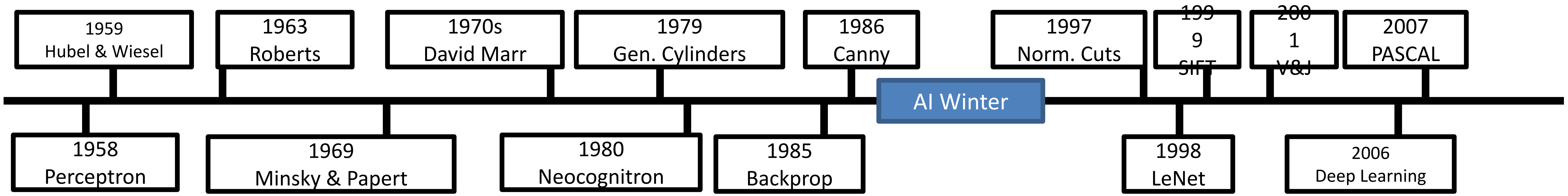
2000s: “Deep Learning”

People tried to train neural networks that were deeper and deeper

Not a mainstream research topic at this time

Hinton and Salakhutdinov, 2006
Bengio et al, 2007
Lee et al, 2009
Glorot and Bengio, 2010





Slide inspiration: Justin Johnson

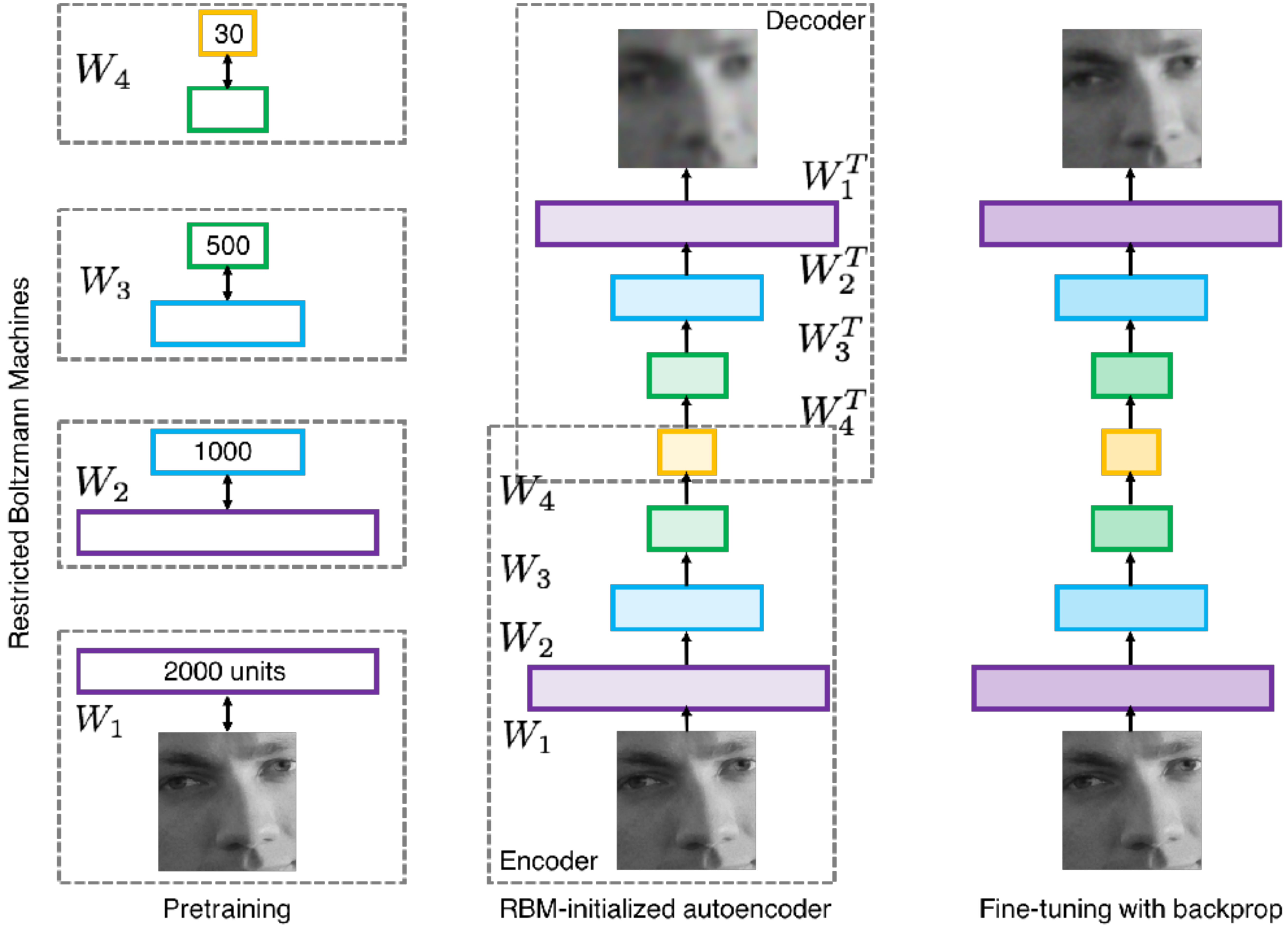
2000s: “Deep Learning”

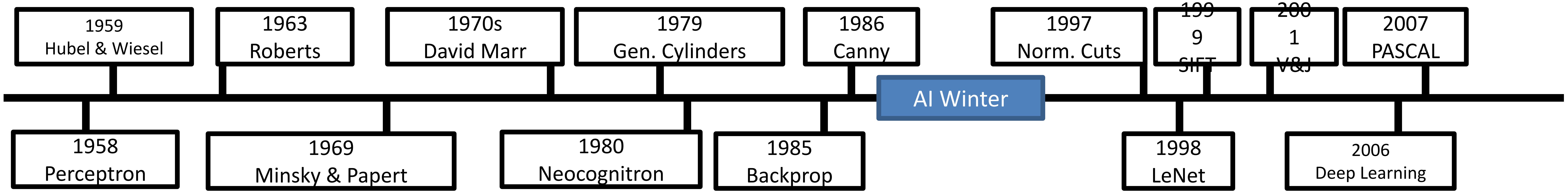
People tried to train neural networks that were deeper and deeper

Not a mainstream research topic at this time

No good dataset to work on

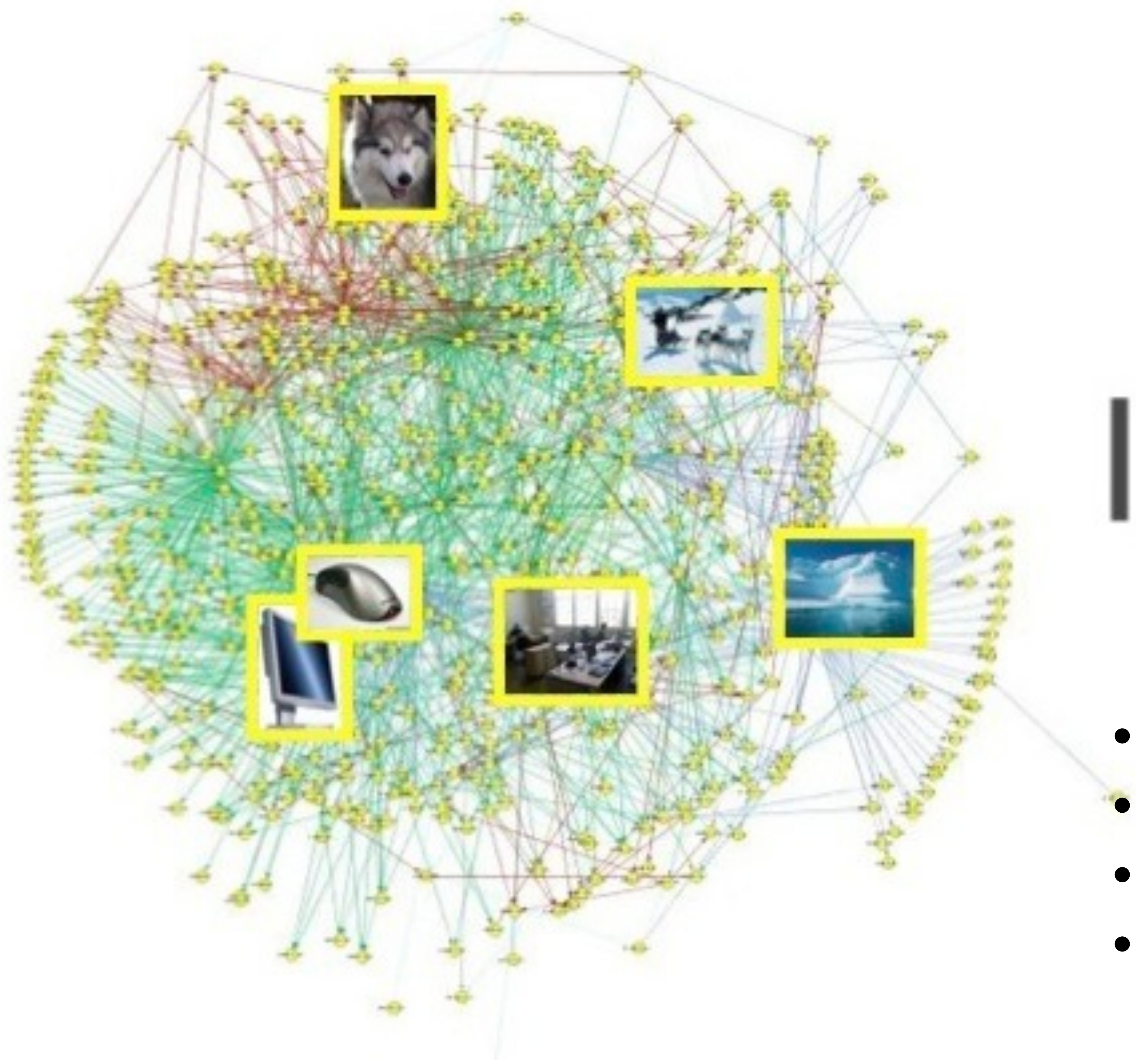
Hinton and Salakhutdinov, 2006
Bengio et al, 2007
Lee et al, 2009
Glorot and Bengio, 2010





Slide inspiration: Justin Johnson

ImageNet Challenge 2010-17



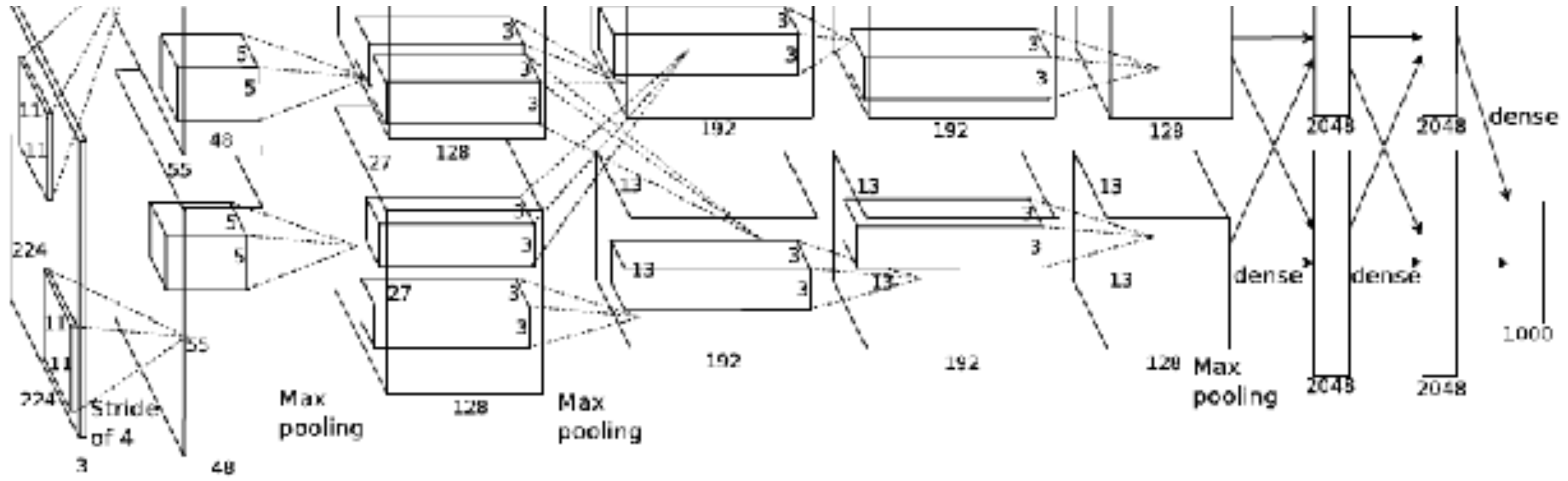
IMAGENET

- 14+ million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- The challenge dataset: 1.2 million training images, 1000 classes

[Deng et al. CVPR 2009]

ImageNet Challenge 2012

AlexNet (2012)



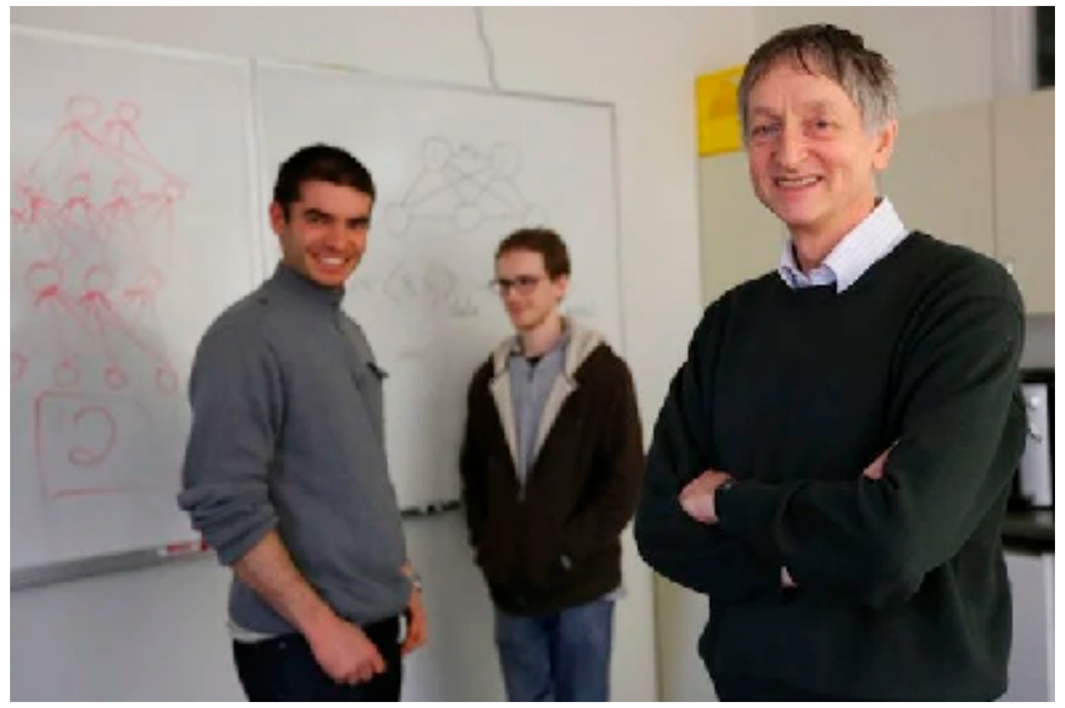
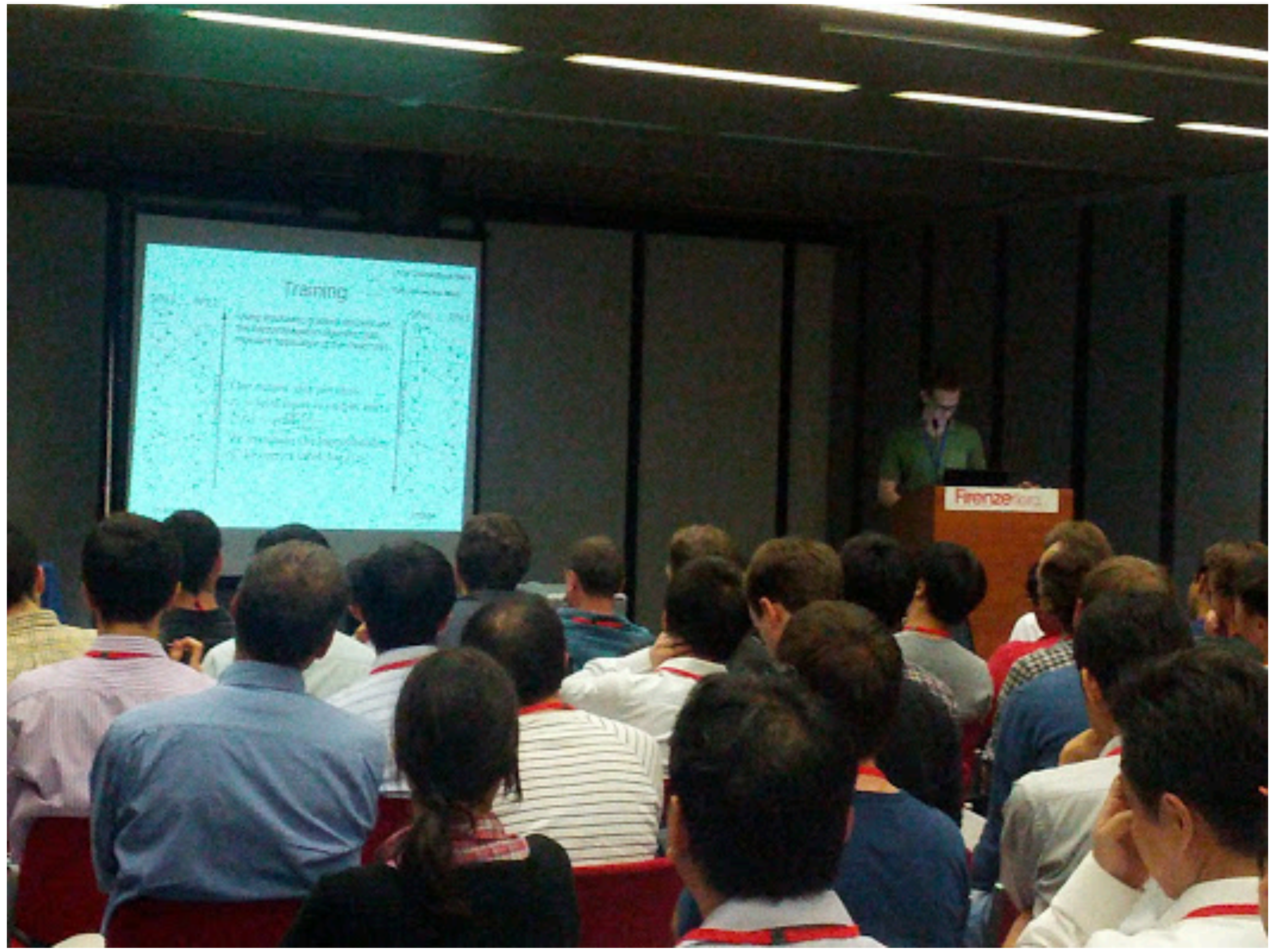


Photo source

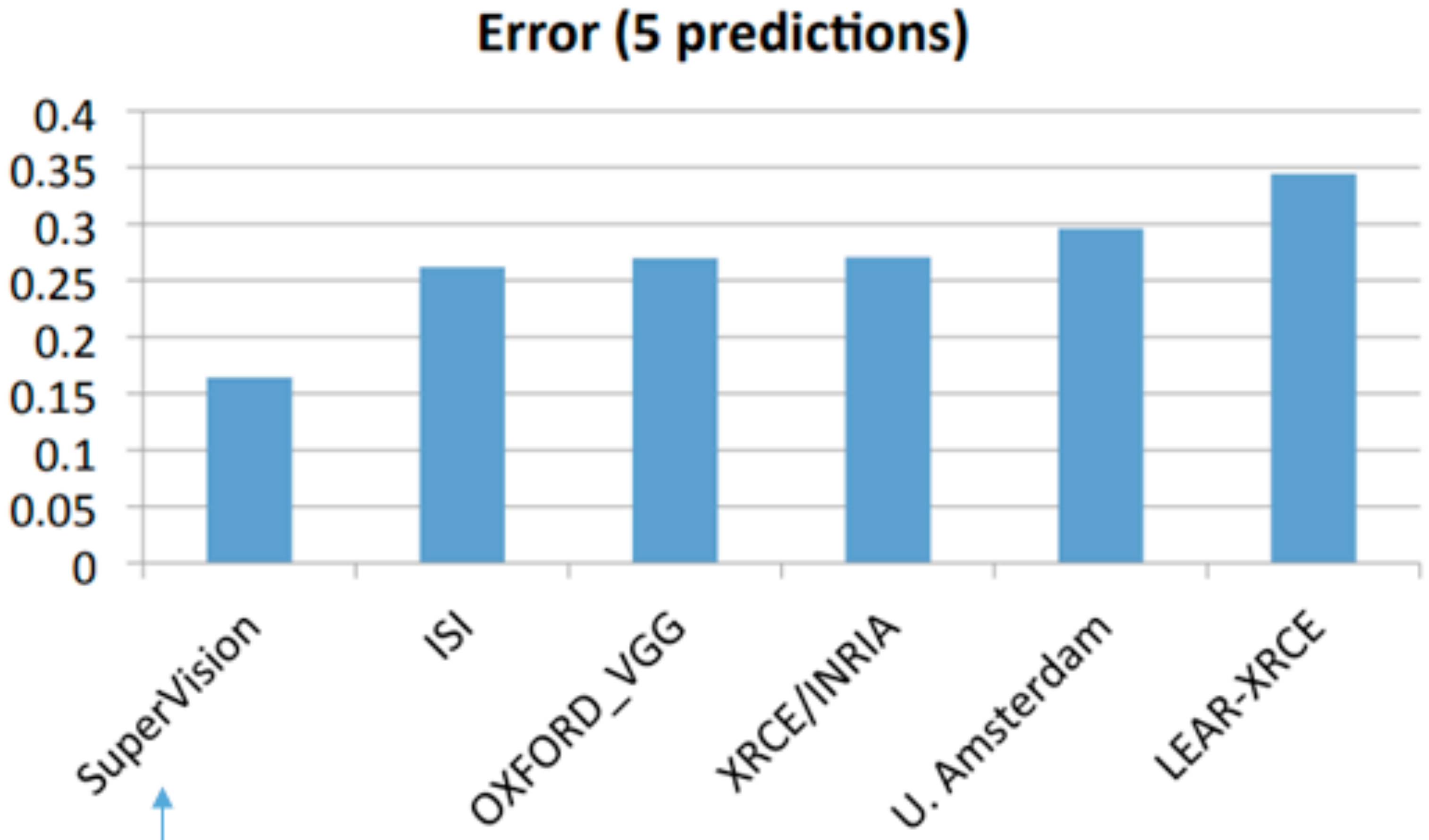
Yann LeCun ▸ Public

Oct 13, 2012

+Alex Krizhevsky's talk at the ImageNet ECCV workshop yesterday made a bit of a splash. The room was overflowing with people standing and sitting on the floor. There was a lively series of comments afterwards, with +Alyosha Efros, Jitendra Malik, and I doing much of the talking.



Ranking of the best results from each team

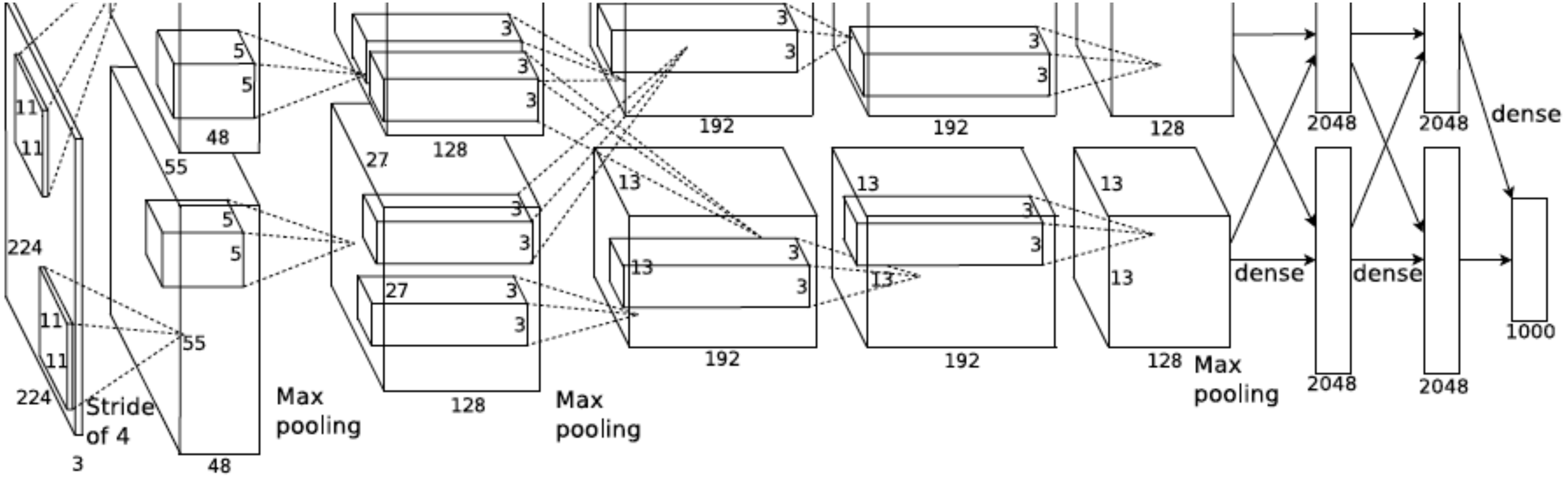


AlexNet

ImageNet Challenge 2012

Similar to LeCun'98 with "some" differences:

- Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
- More data (10^6 vs. 10^3 images) — ImageNet dataset [Deng et al.]
- GPU implementation (50x speedup over CPU) ~ 2 weeks to train
- Some twists: Dropout regularization, ReLU max(0,x)



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

What do these networks learn?

How do we visualize a complicated, non-linear function?

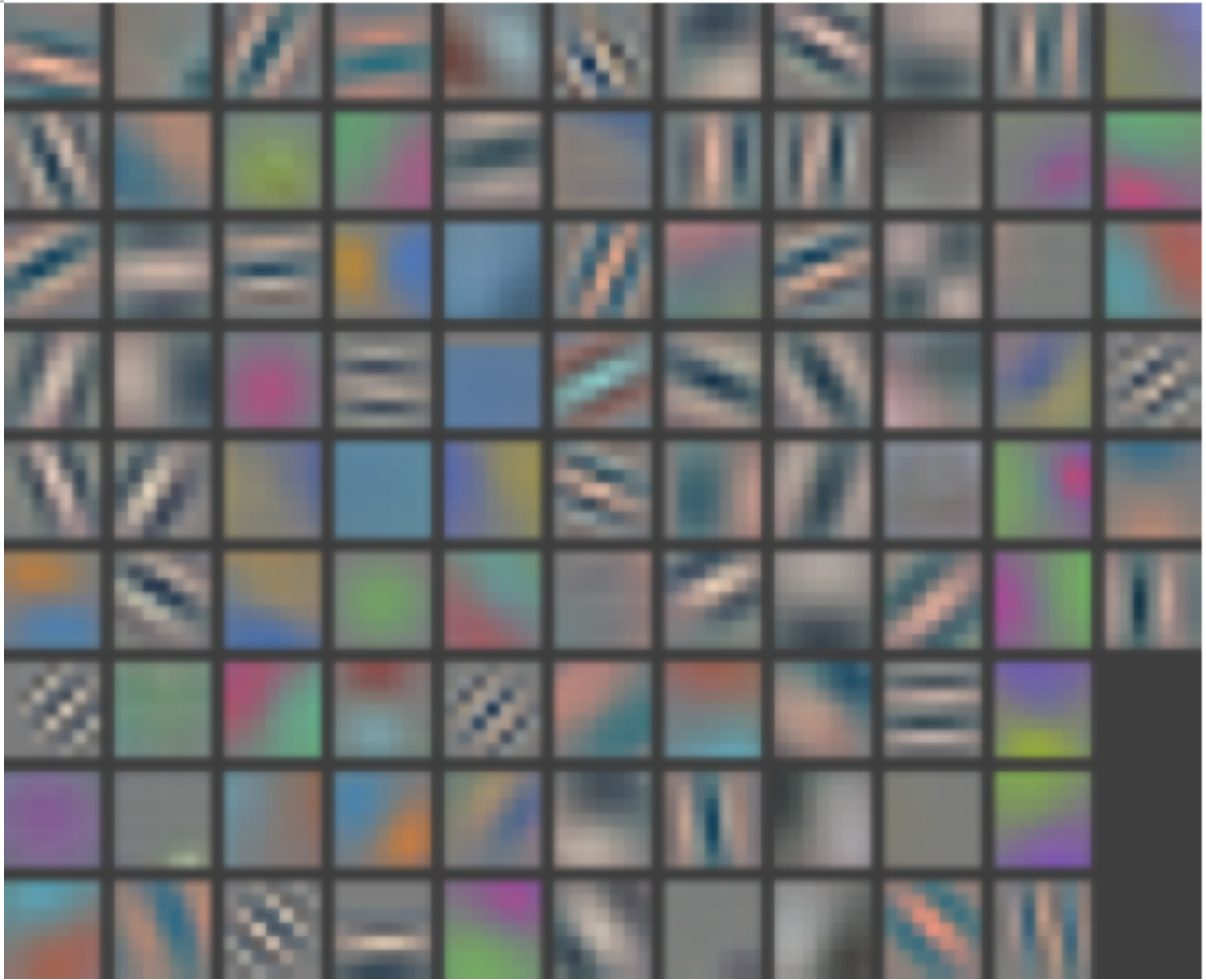
Good paper: Visualizing and Understanding Convolutional Networks, Matthew D. Zeiler, Rob Fergus, ECCV 2014

Good toolboxes

- Understanding Neural Networks Through Deep Visualization, Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, ICML Deep Learning Workshop, 2015 (http://yosinski.com/deepvis)

Many other resources online (search for visualizing deep networks)

Layer 1: Learned filters



"edge" and "blob" detectors

Layer 1: Top-9 Patches

- Patches from validation images that give maximal activation of a given feature map

Layer 2: Top-9 Patches

Layer 3: Top-9 Patches

Layer 4: Top-9 Patches

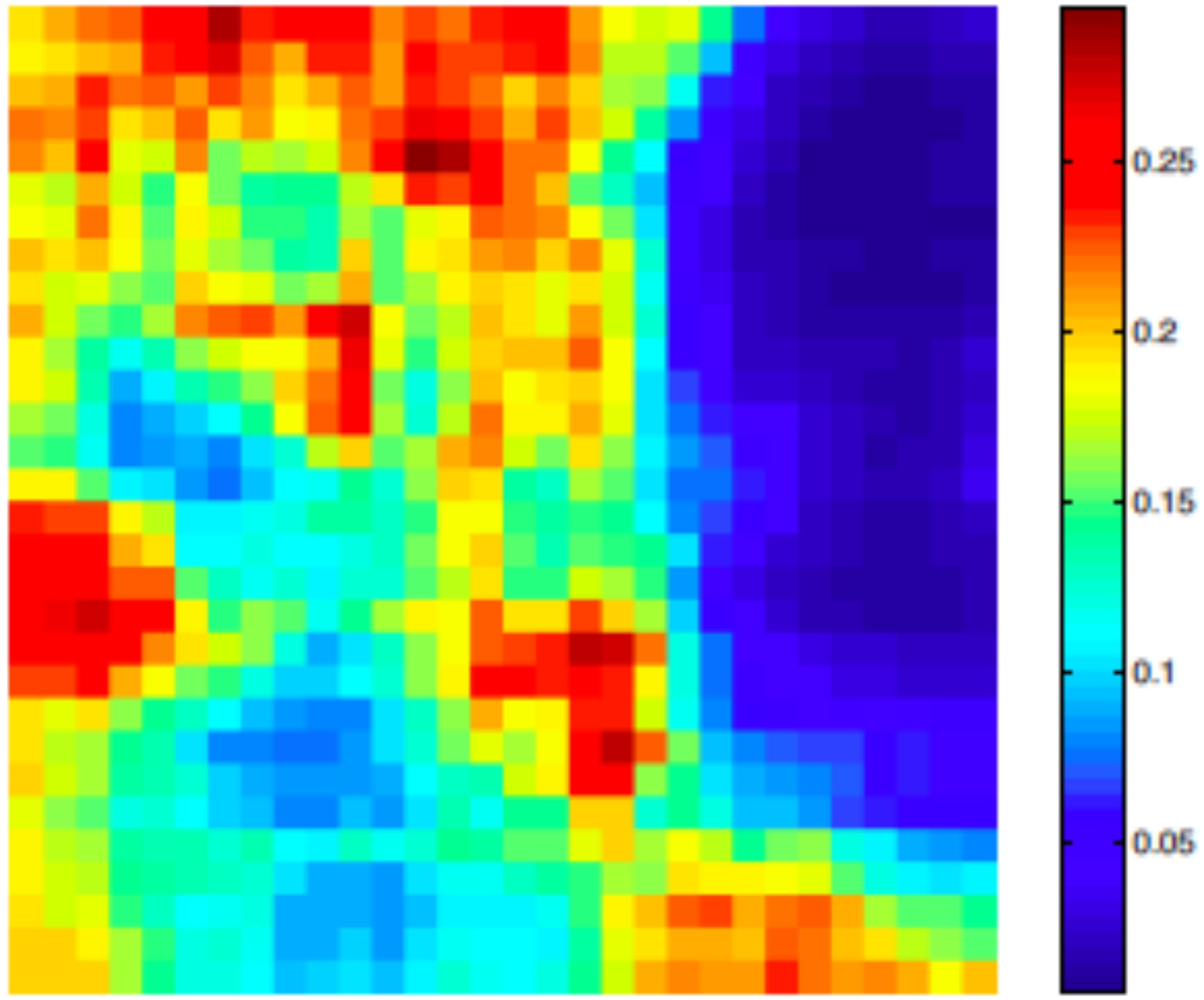
Layer 5: Top-9 Patches

Occlusion Experiment

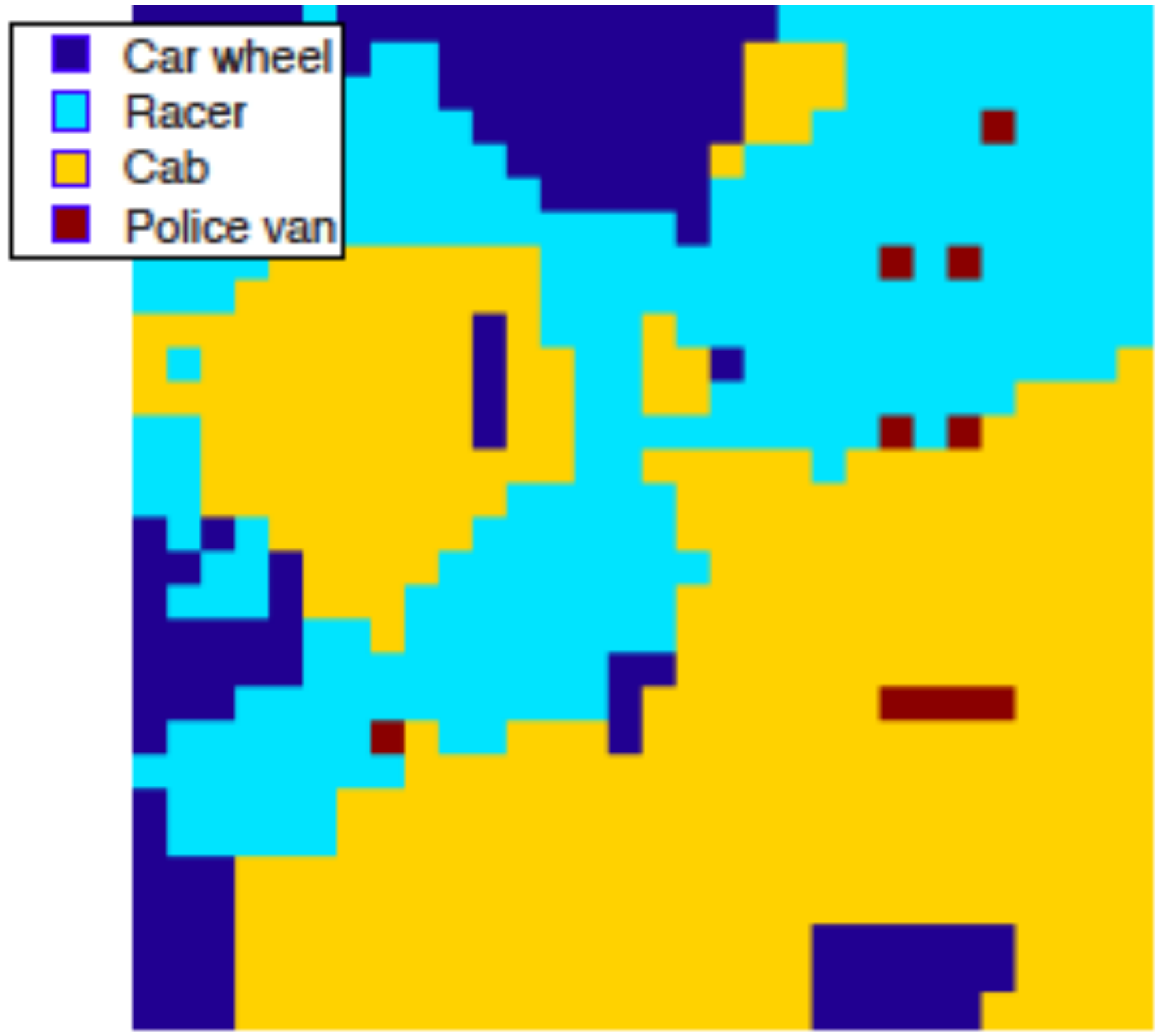
Mask parts of input with occluding square

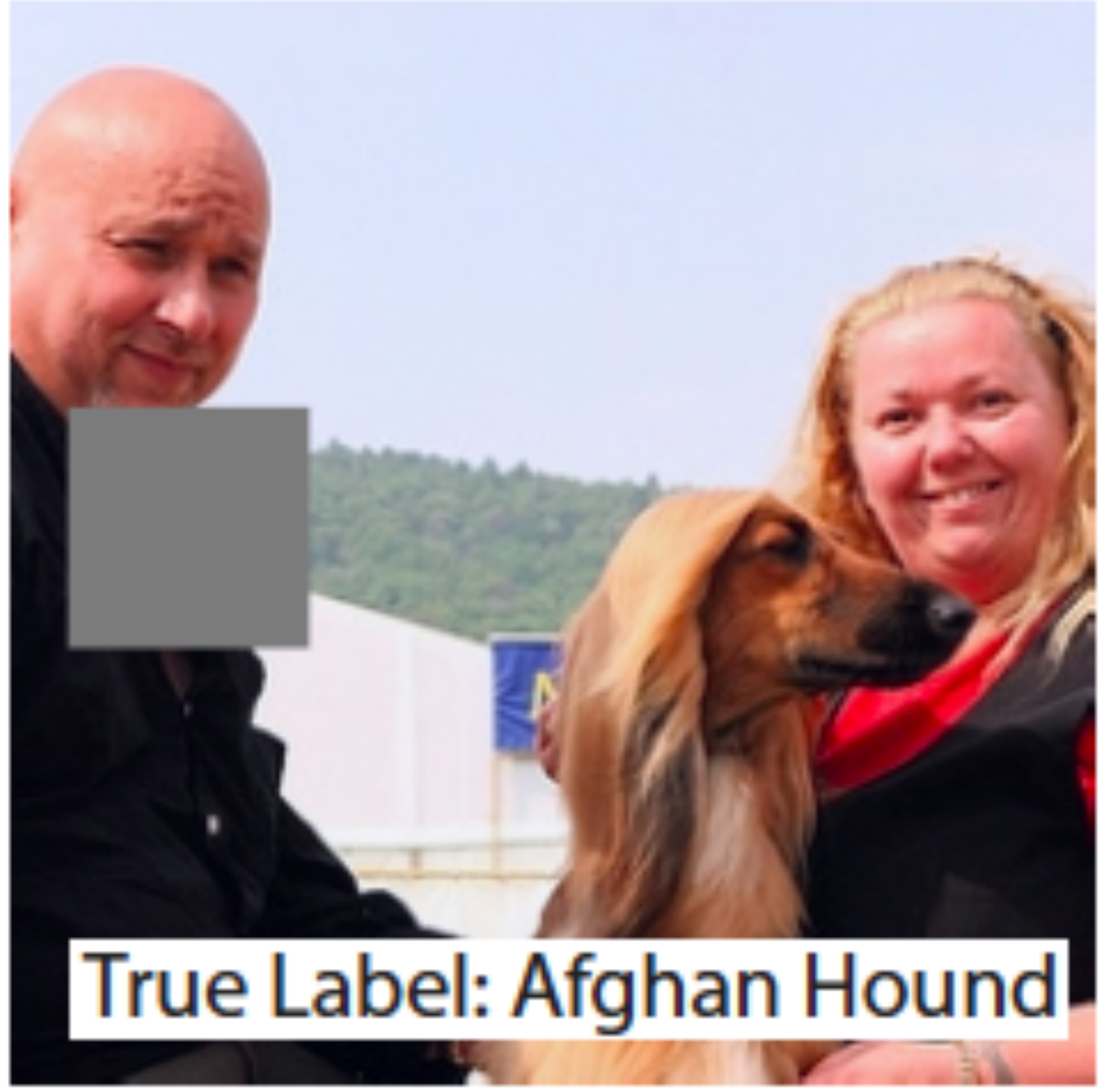
Monitor output (class probability)

p(True class)

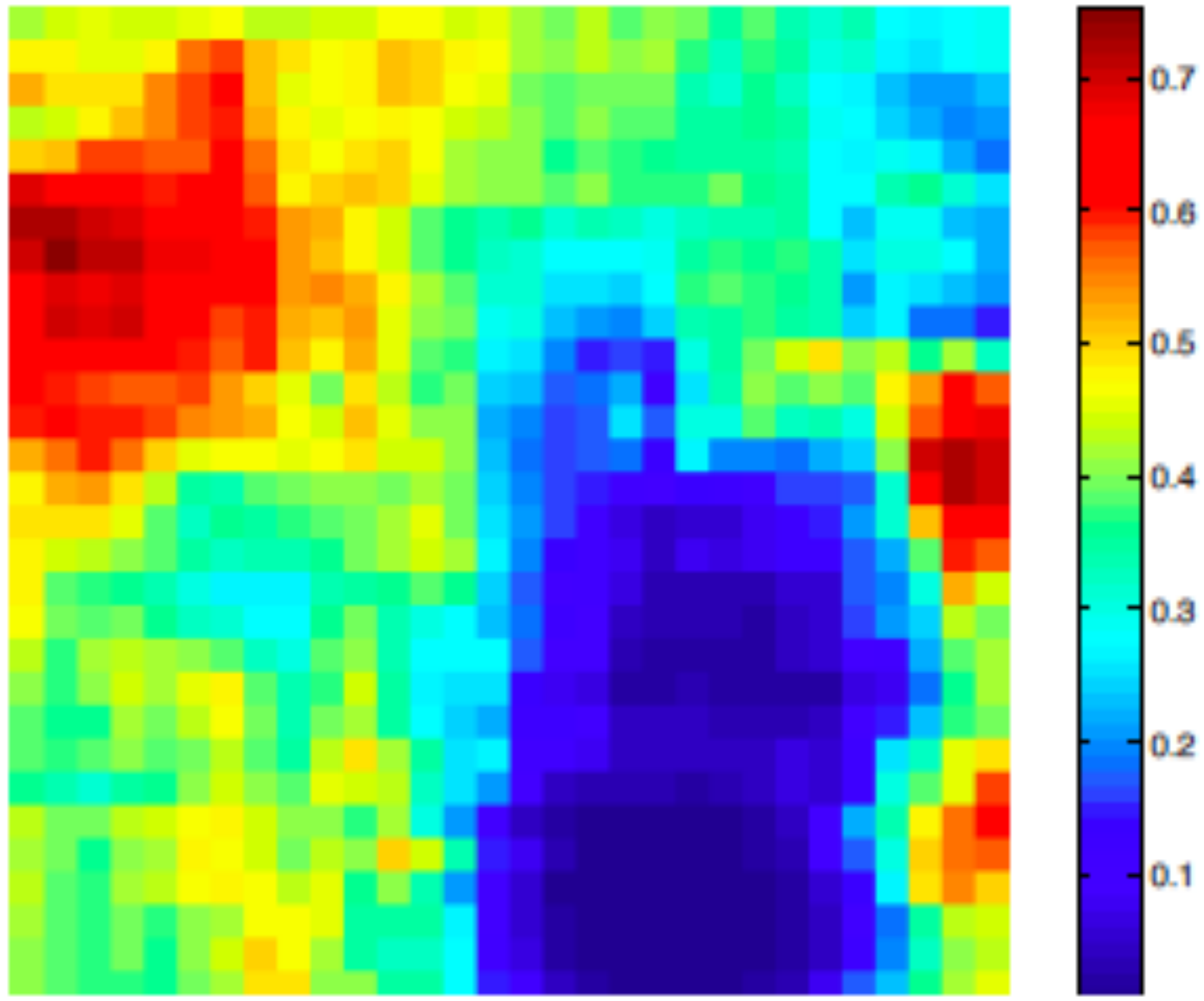


Most probable class

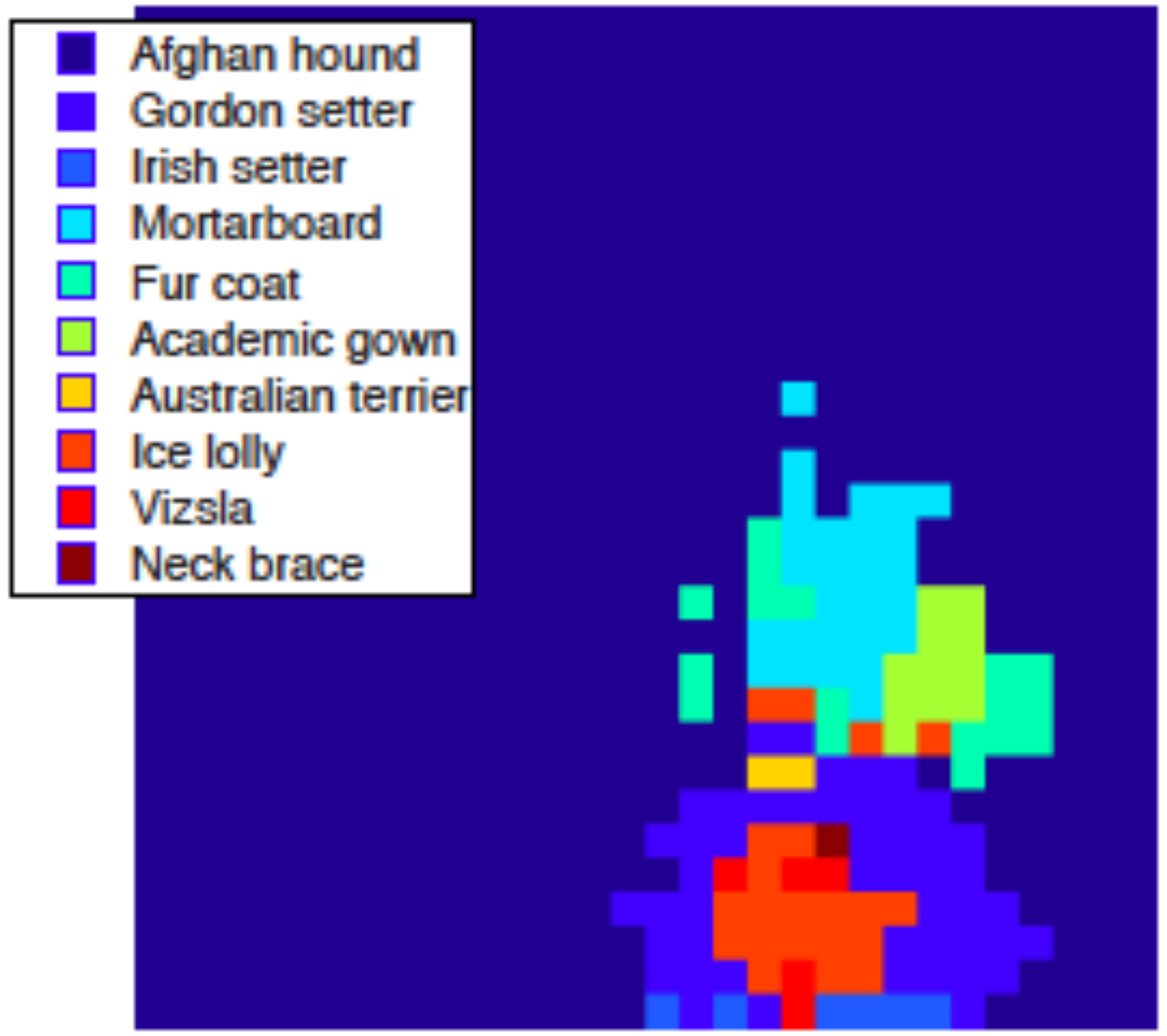




p(True class)

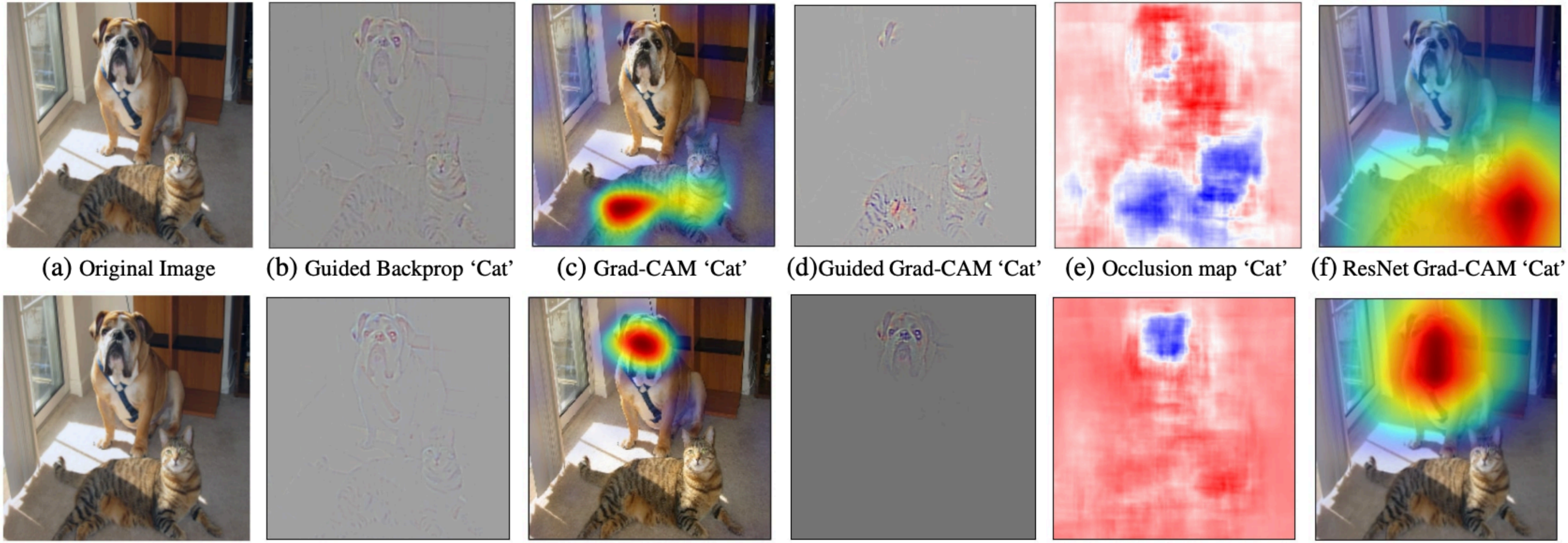


Most probable class



Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

Ramprasaath R. Selvaraju · Michael Cogswell · Abhishek Das · Ramakrishna Vedantam · Devi Parikh · Dhruv Batra

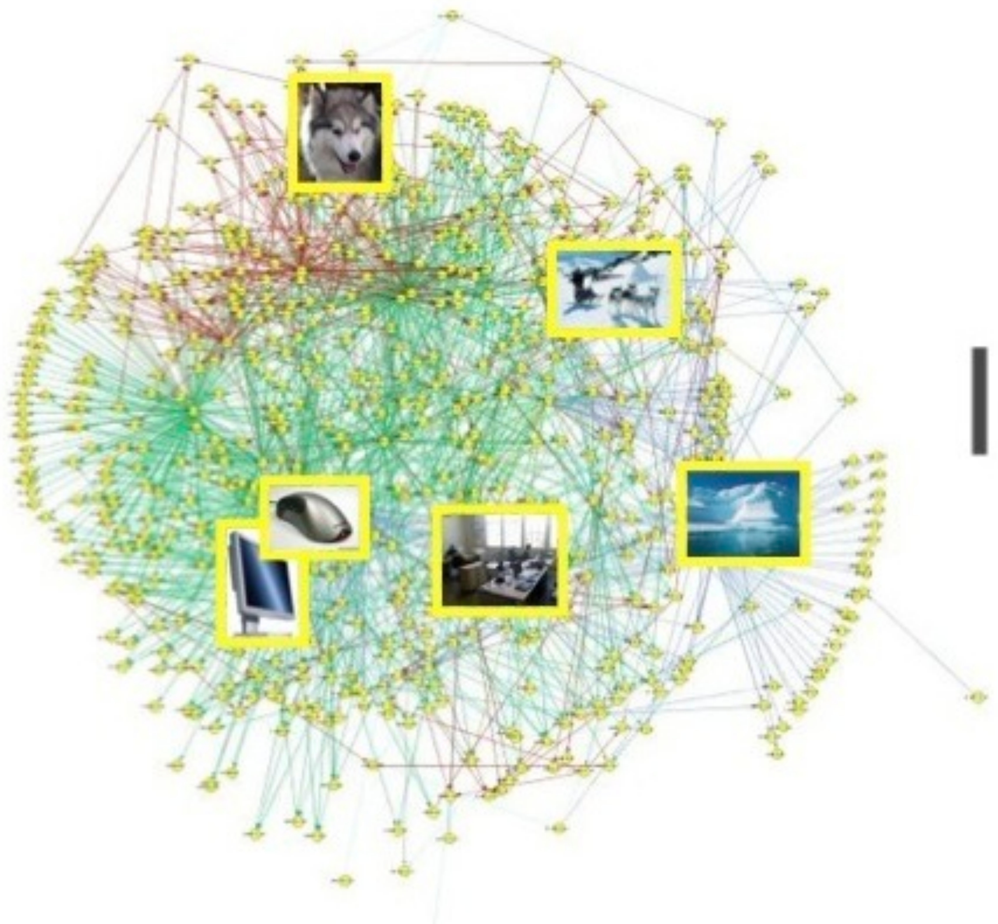


(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d)Guided Grad-CAM 'Cat' (e) Occlusion map 'Cat' (f) ResNet Grad-CAM 'Cat'

(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (j)Guided Grad-CAM 'Dog' (k) Occlusion map 'Dog' (l)ResNet Grad-CAM 'Dog'

Transfer learning

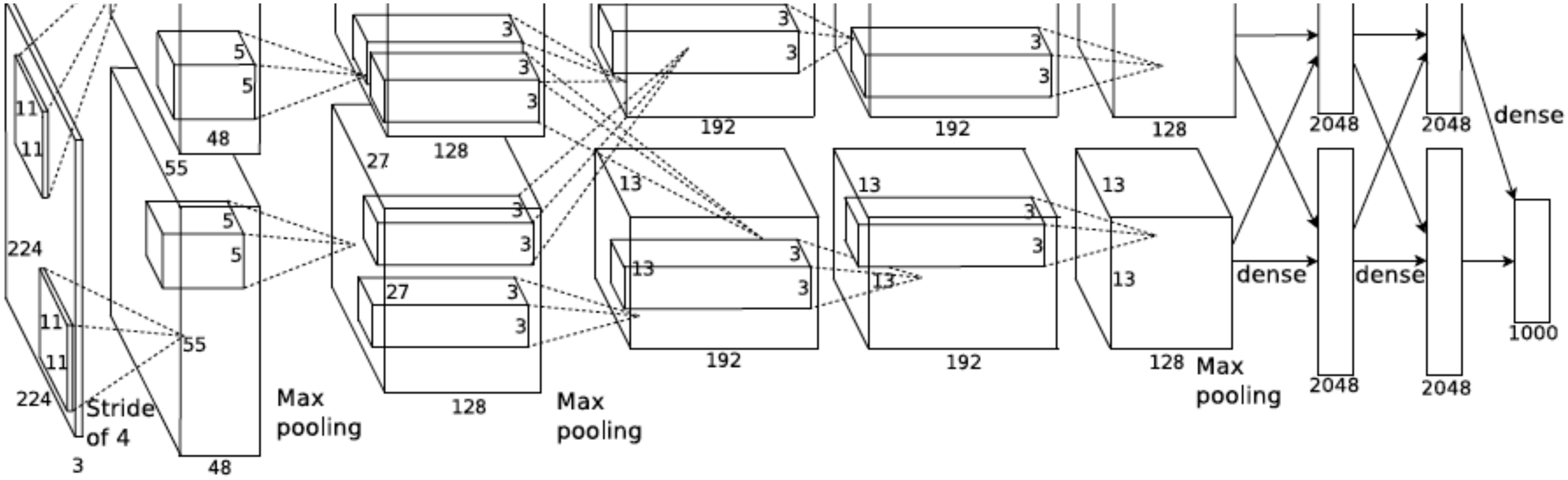
ImageNet challenge



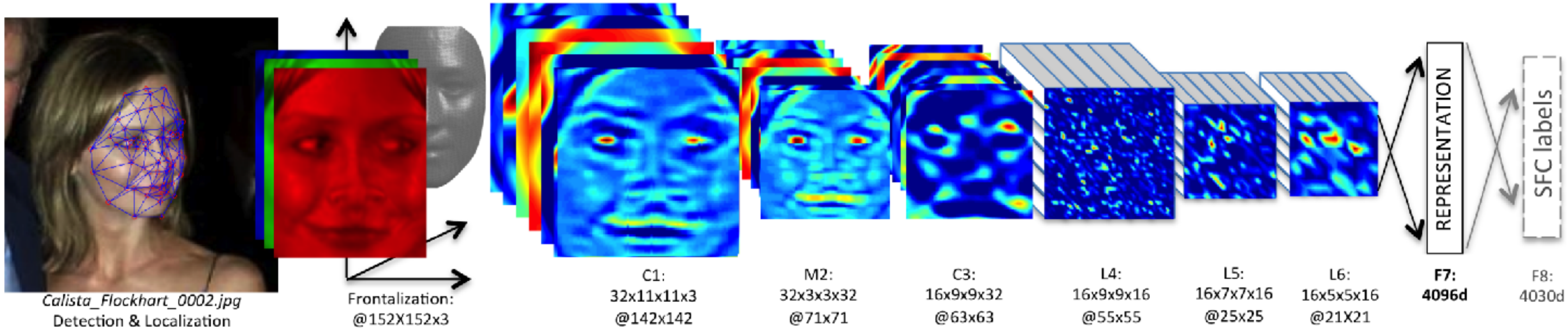
- 14+ million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- The challenge: 1.2 million training images, 1000 classes

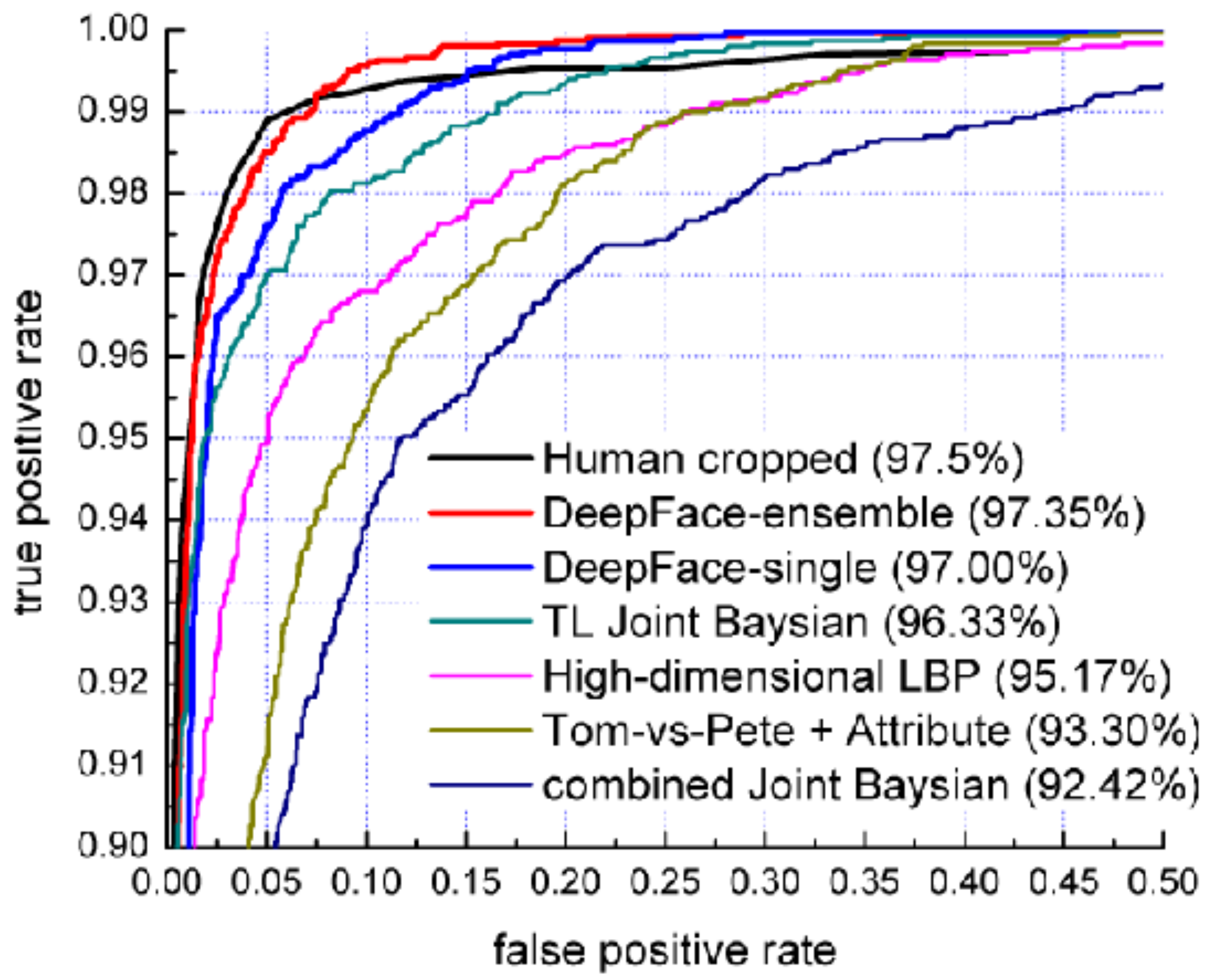
+

Multi-layer CNNs (60M parameters)



Face recognition





Y. Taigman, M. Yang, M. Ranzato, L. Wolf, DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR 2014

Deep learning — reality vs. practice

source: reddit

Issues with learning from little data

Not only a computational, but also a statistical challenge …

- Overfitting,
- Bias,
- Calibration,
- Label noise, …

Little data

solutions

Unlabeled examples

- Self-/Semi-supervised learning
- Active learning

Related datasets

- Transfer learning
- Multi-tasking
- Meta learning

Pre-trained models

- Robust finetuning, adaptors

Transfer learning

How do we learn parameter-rich models on small datasets?

- # parameters >> # training data

Solution: Learn from related tasks

- Training and testing tasks can be different!
- In general, we can't expect much when the tasks are too different
  - Will learning how to drive in Amherst help you drive in Cambridge?

For images we might expect that learning to solve classification tasks on large datasets such as ImageNet might help us solve other visual recognition tasks.

Transfer learning with CNNs

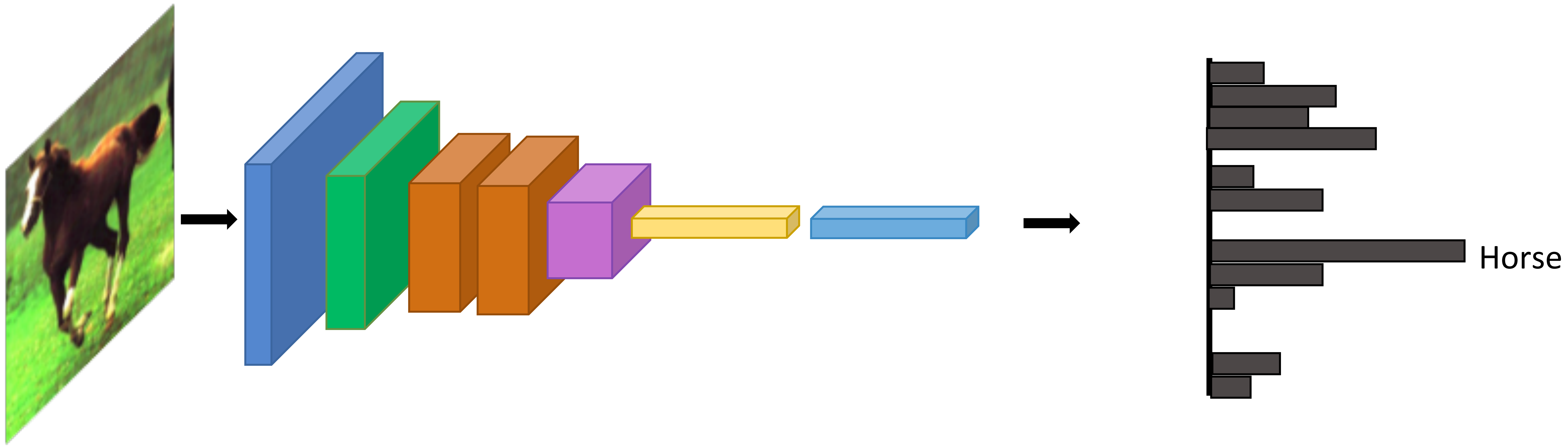
Train a model on ImageNet

Take outputs of an intermediate layer as features

Train linear classifier on these features

Pros: simpler learning, efficiency

Con: no end-to-end learning



Transfer learning with CNNs

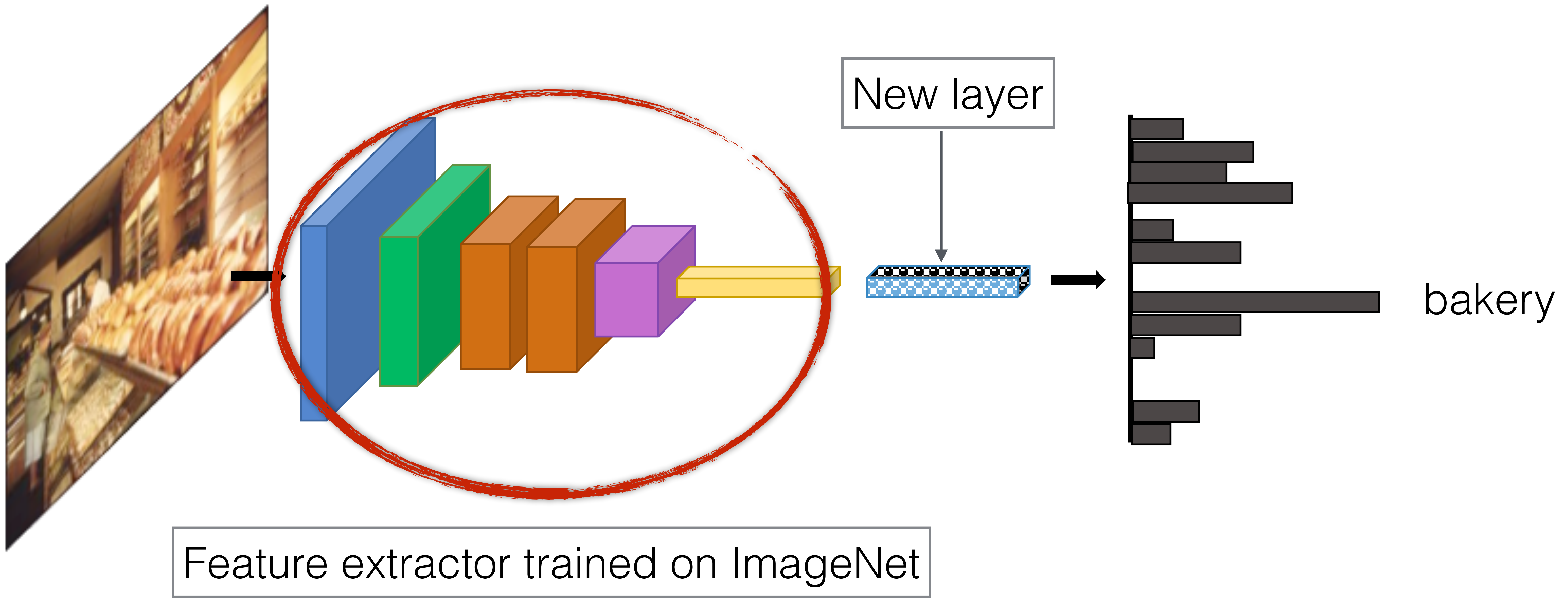
Train a model on ImageNet

Take outputs of an intermediate layer as features

Train linear classifier on these features

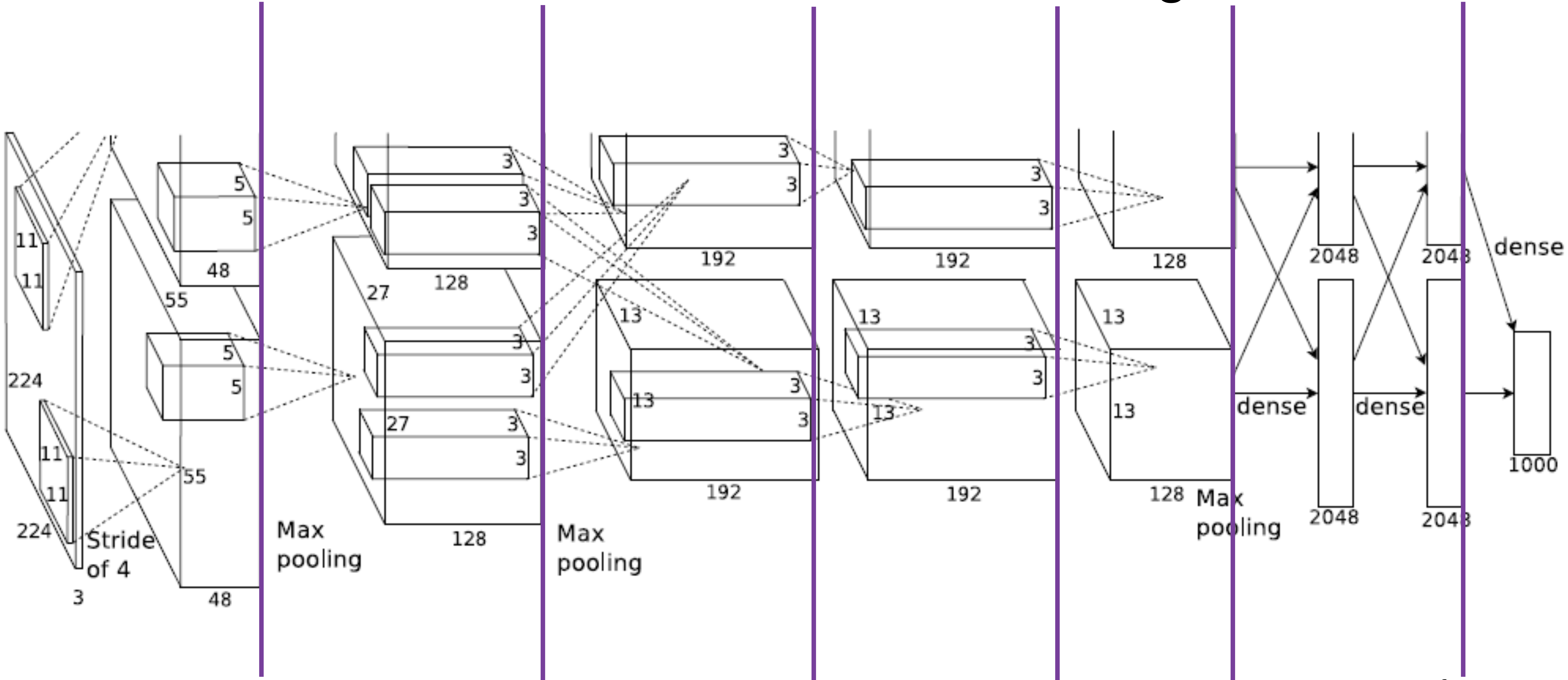
<u>Pros:</u> simpler learning, efficiency

<u>Con:</u> no end-to-end learning

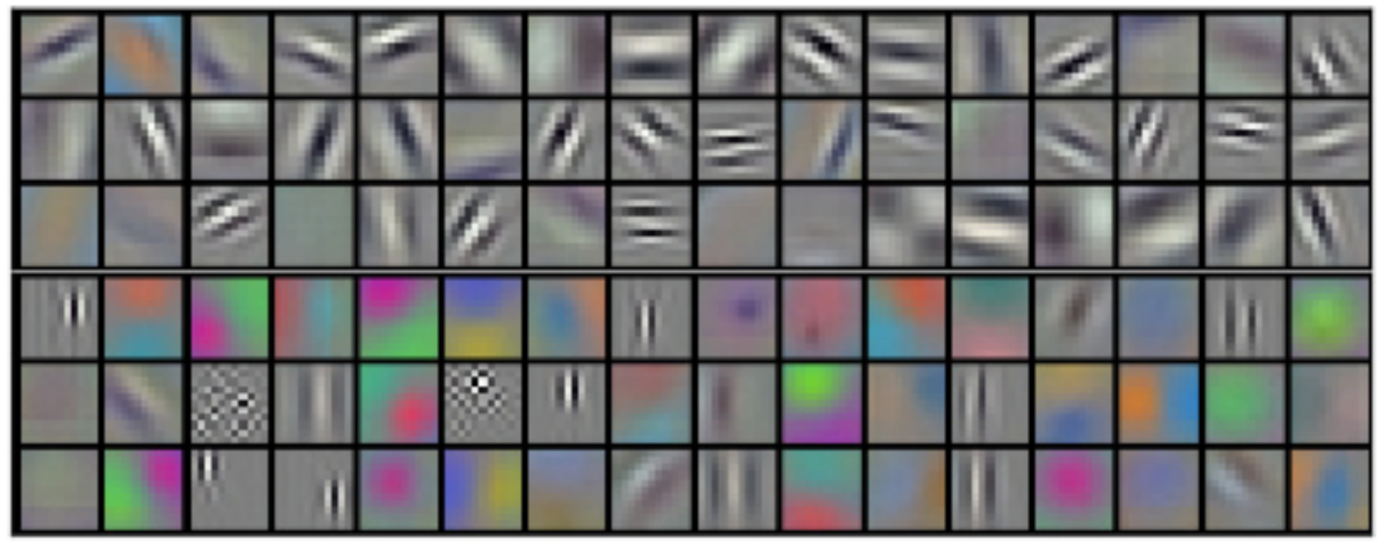


Tapping off features at each layer

low-level → high-level



conv1 conv2 conv3 conv4 conv5 fc7



11x11x3x96 filters

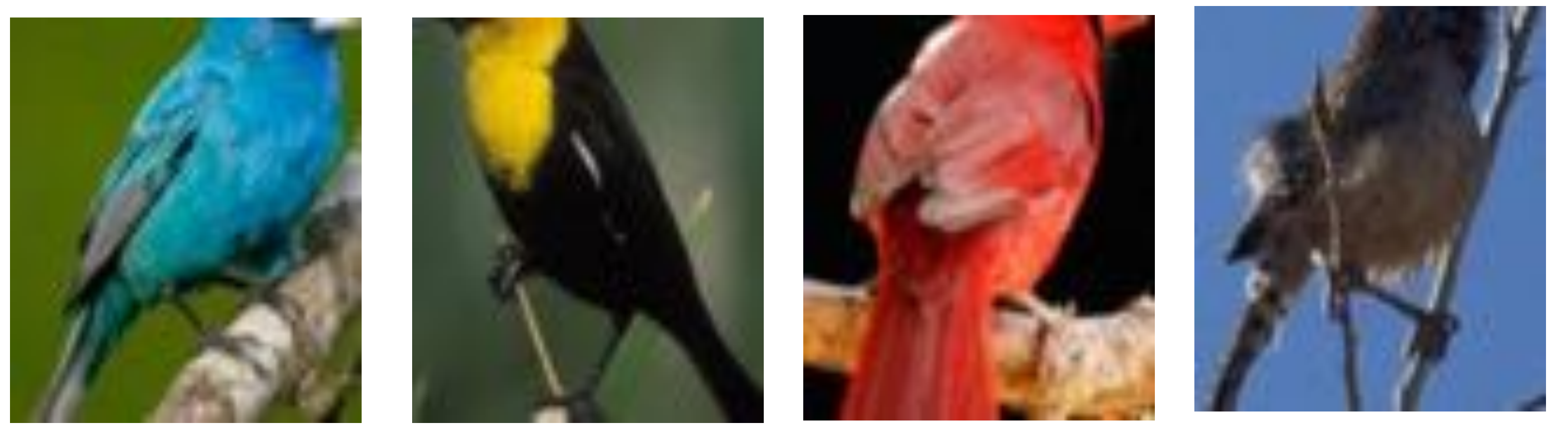
Different layers

Datasets and benchmarks

Caltech 101/256
[Fei-Fei et al. 04]

Fine-grained recognition (Aircraft)
[Maji et al. 13]

Fine-grained recognition (CUB)
[Wah et al. 11]



Scene recognition (MIT Indoors)
[Quattoni and Torralba 09]

Object recognition (VOC07)
[Everingham et al. 07]

Fine-grained recognition (Cars)
[Krause et al. 13]

Tapping off features at each Layer

Plug features from each layer into linear classifier

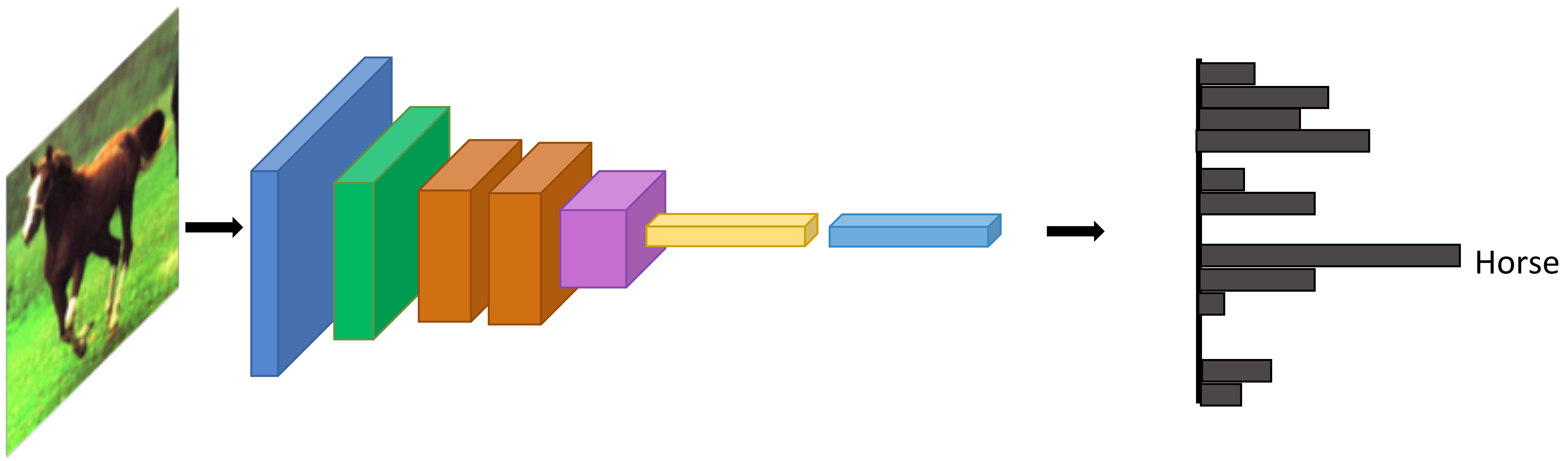
	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$

Results on benchmarks

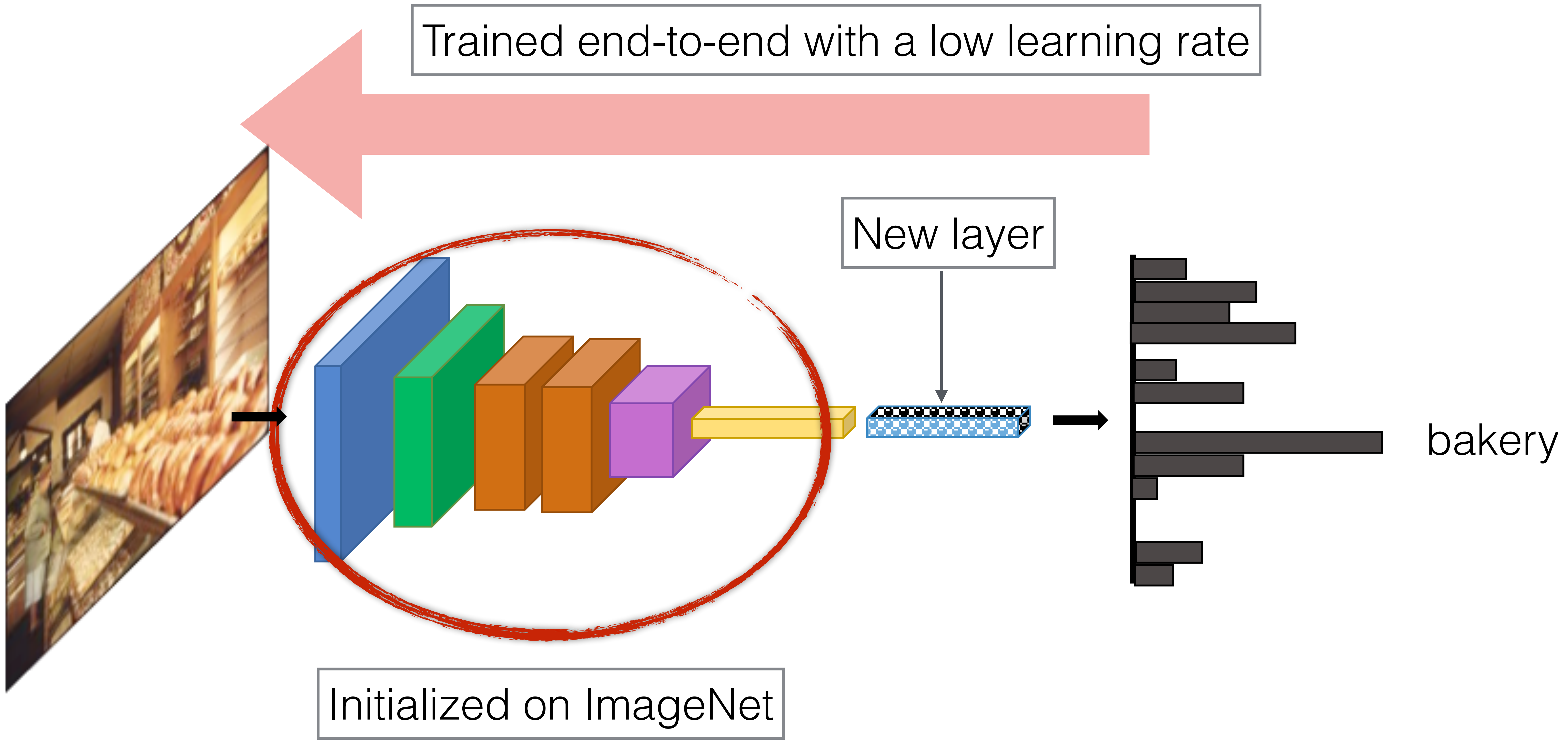
Dataset	Non-Convnet Method	Non-Convnet perf	Pretrained convnet + classifier	Improvement
Caltech 101	MKL	84.3	87.7	+3.4
VOC 2007	SIFT+FK	61.7	79.7	+18
CUB 200	SIFT+FK	18.8	61.0	+42.2
Aircraft	SIFT+FK	61.0	45.0	-16
Cars	SIFT+FK	59.2	36.5	-22.7

Finetuning

Train a model on ImageNet



Finetuning

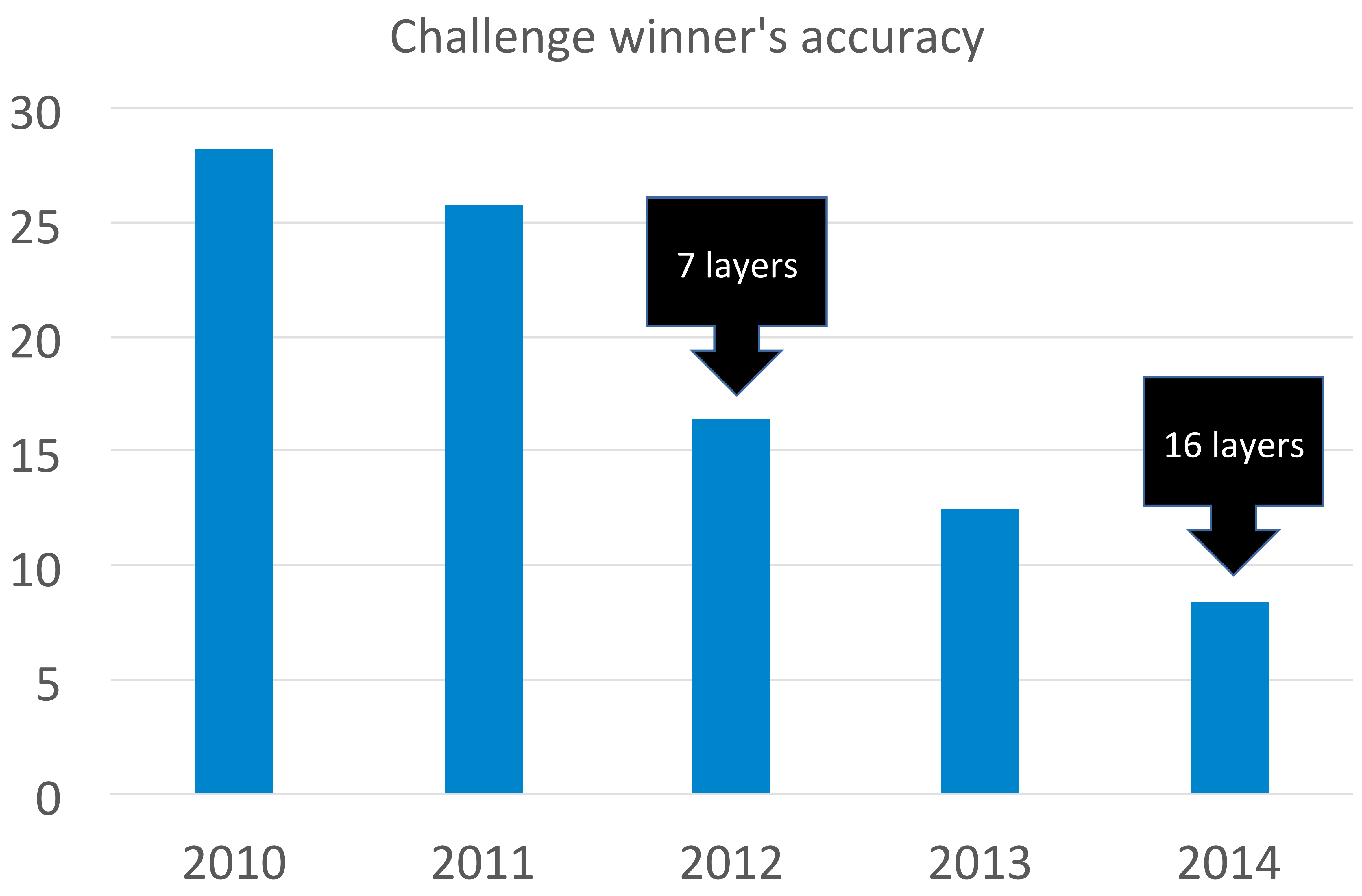

Trained end-to-end with a low learning rate
New layer
bakery
Initialized on ImageNet

Results on benchmarks

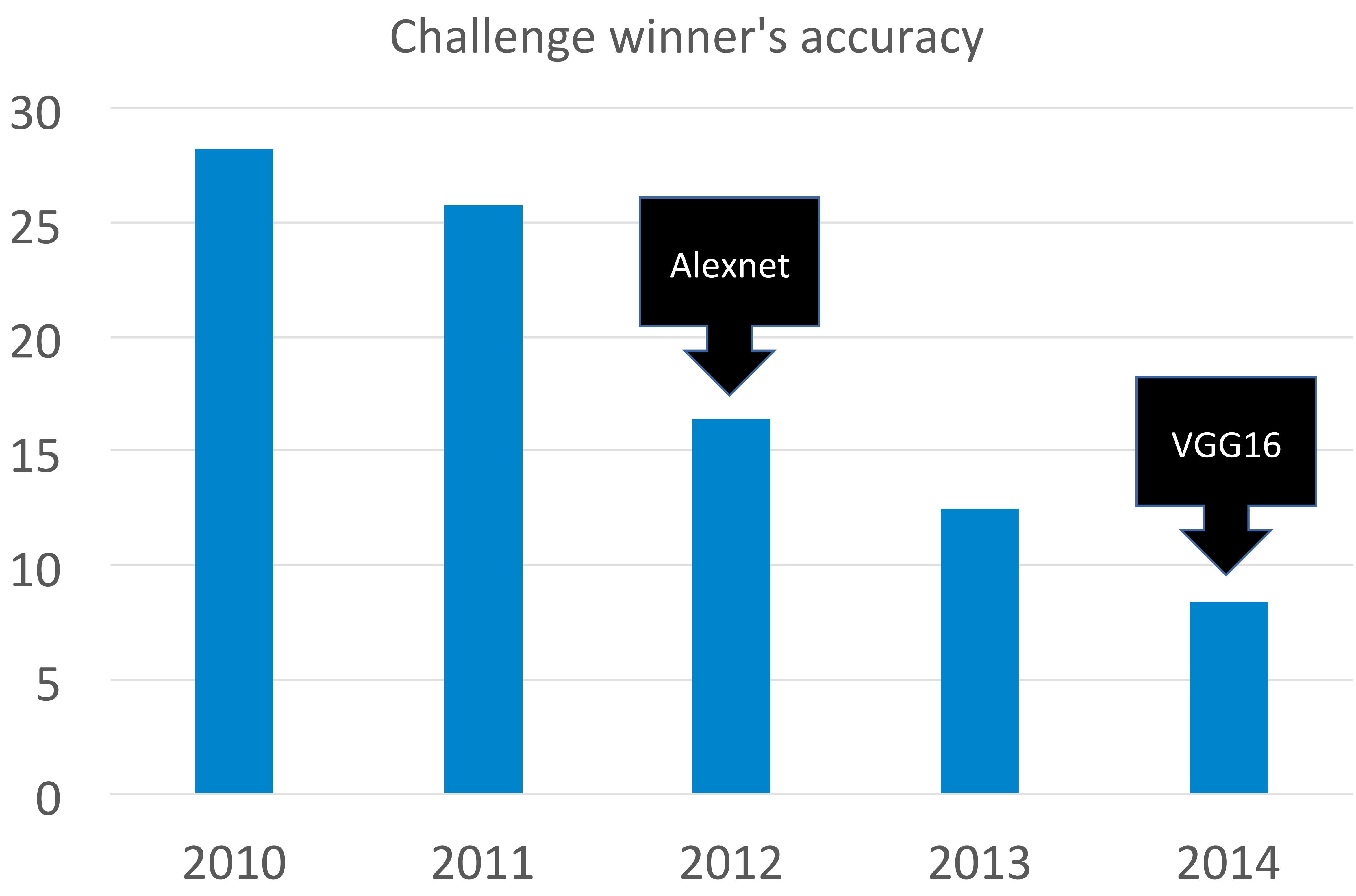
Dataset	Non-Convnet Method	Non-Convnet perf	Pretrained convnet + classifier	Finetuned convnet	Improvement
Caltech 101	MKL	84.3	87.7	88.4	+4.1
VOC 2007	SIFT+FK	61.7	79.7	82.4	+20.7
CUB 200	SIFT+FK	18.8	61.0	70.4	+51.6
Aircraft	SIFT+FK	61.0	45.0	74.1	+13.1
Cars	SIFT+FK	59.2	36.5	79.8	+20.6

Network architectures

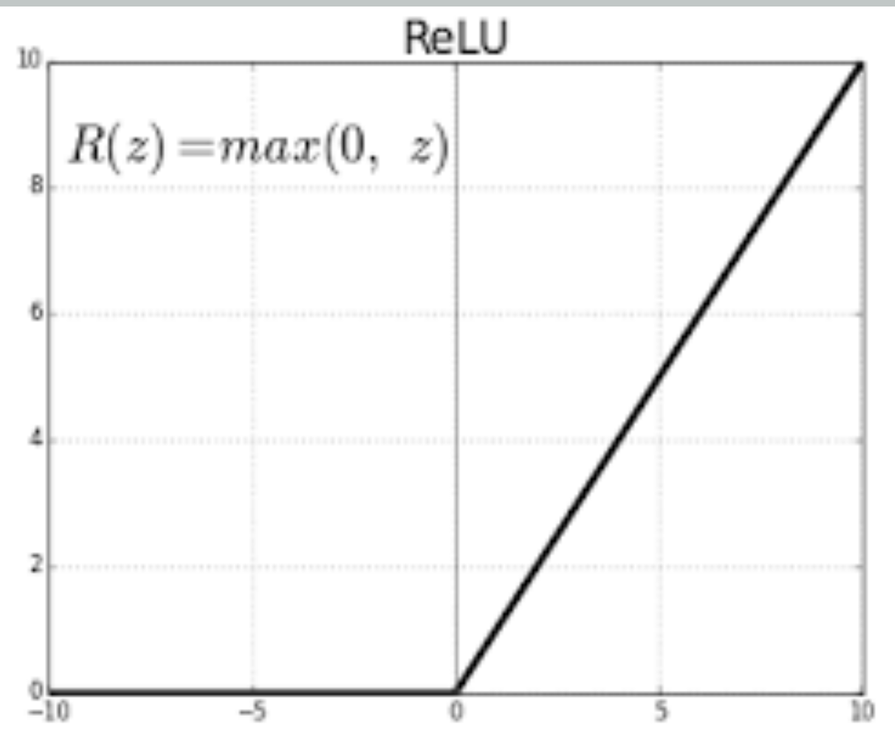
Deeper is better



Deeper is better



The VGG pattern



Every convolution is 3x3, padded by 1

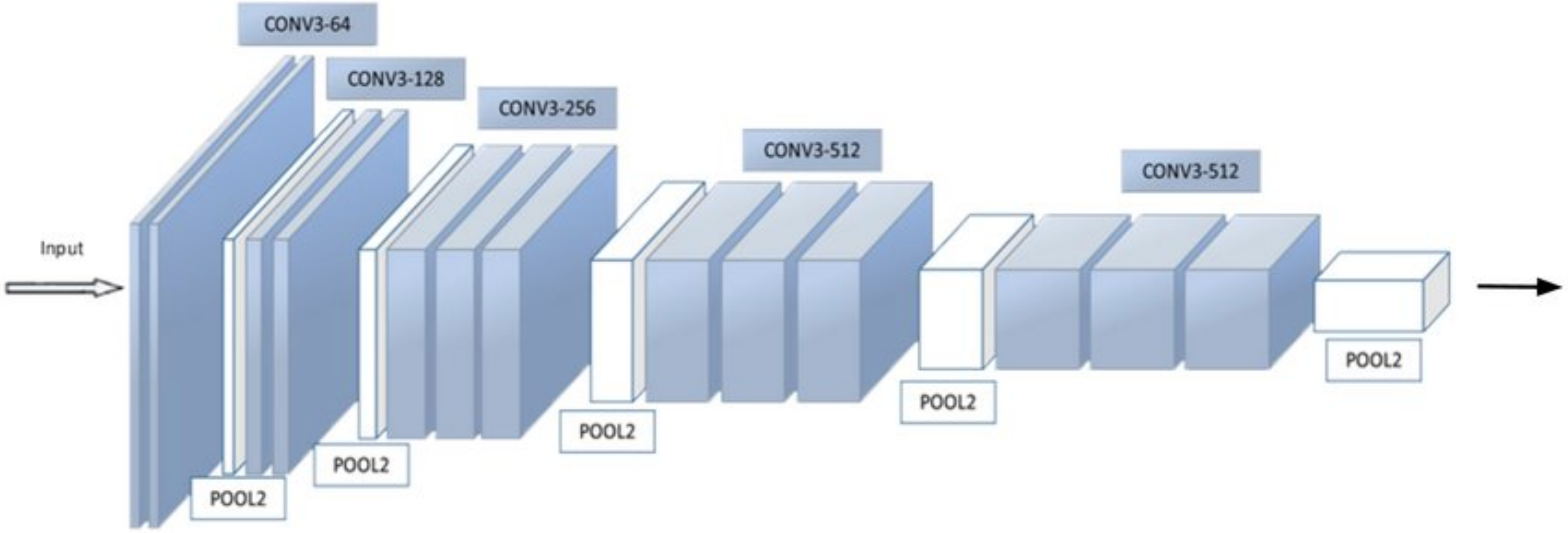
Every convolution is followed by a ReLU

Network is divided into “stages”

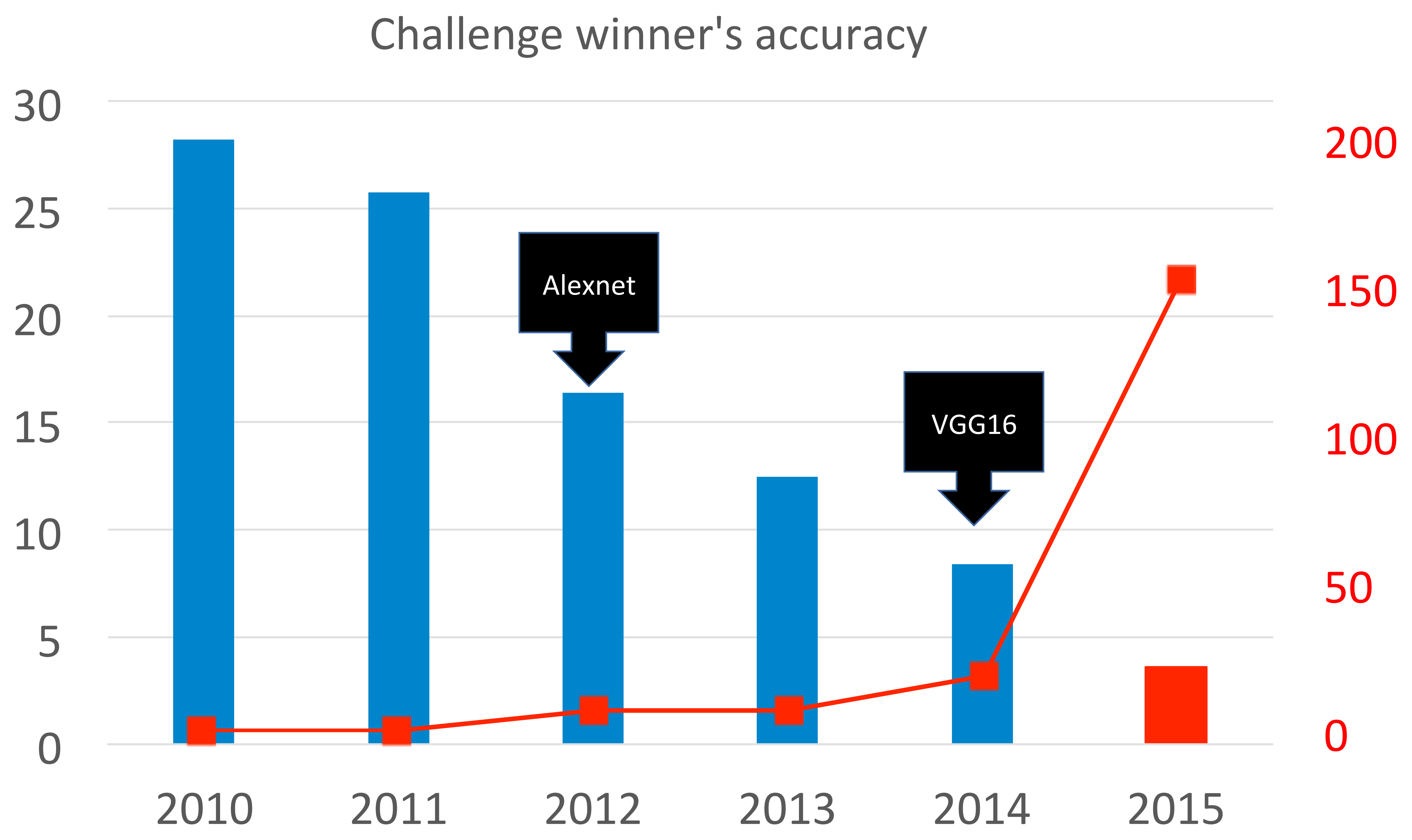
- Layers within a stage: no subsampling
- Subsampling by 2 at the end of each stage

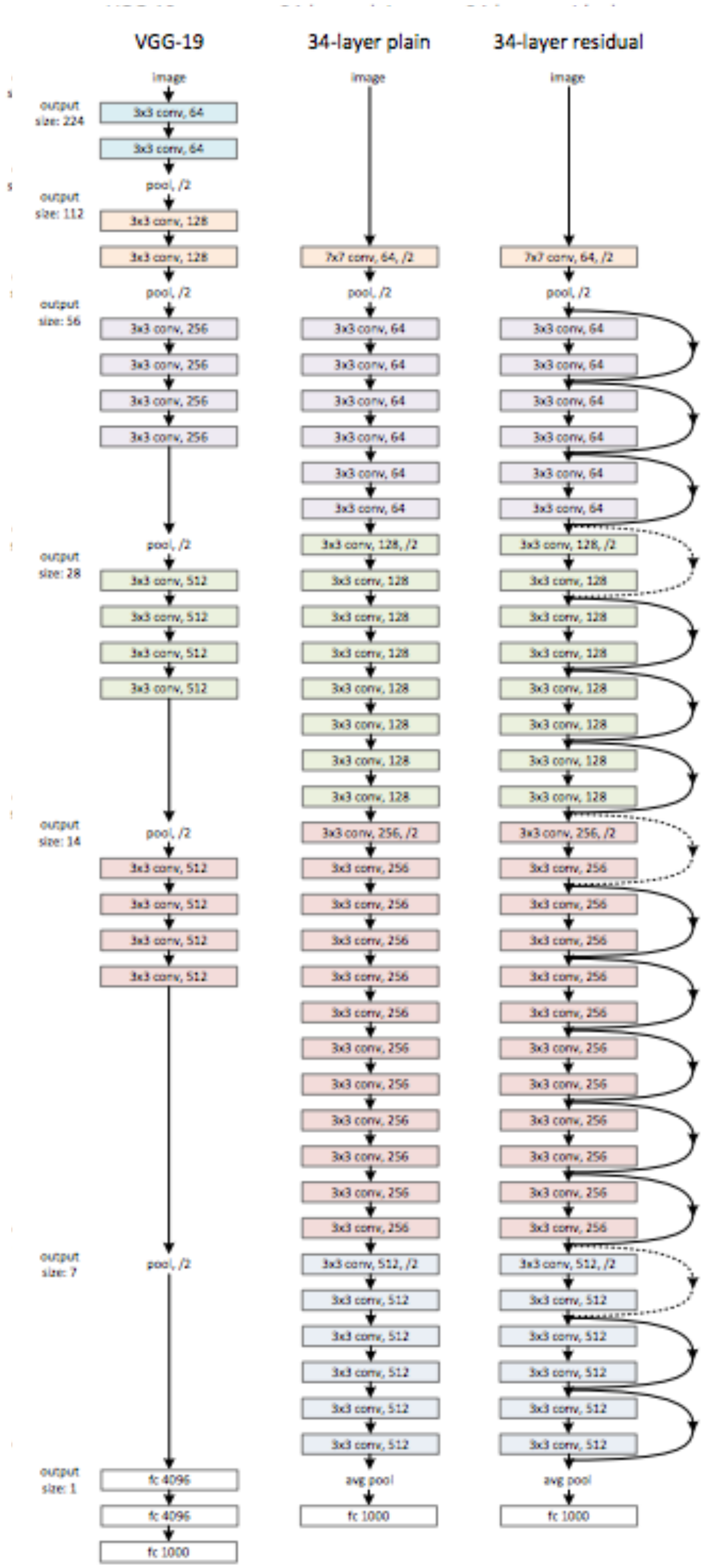
Layers within a stage have the same number of channels

Every subsampling ⇨ double the number of channels



Residual Networks



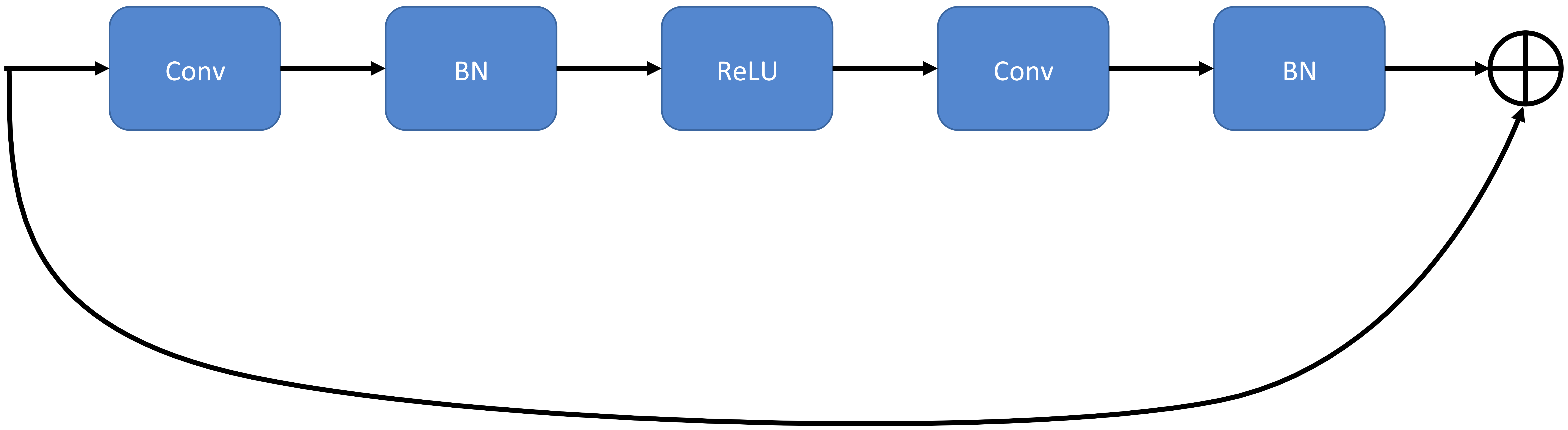


Deep Residual Learning for Image Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

A residual block

Instead of single layers, have residual connection over a block of layers



Summary

Motivation: non-linearity

Ingredients of a neural network (multi-layer perceptron)

- hidden units, link functions

Training by back-propagation

- random initialization, chain rule, stochastic gradients, momentum
- Practical issues: learning, network architecture

Convolutional networks:

- Good for vision problems where inputs have spatial structure and locality
- Shared structure of weights leads to significantly fewer parameters

ImageNet pre-training is a great source of image representations!

Lots of research on network architectures, datasets, and training strategies

Slides credit

Multilayer neural network figure source:

- http://www.ibimapublishing.com/journals/CIBIMA/2012/525995/525995.html

Cat image: http://www.playbuzz.com/abbeymcneill10/which-cat-breed-are-you

More about the structure of the visual processing system

- http://www.cns.nyu.edu/~david/courses/perception/lecturenotes/V1/lgn-V1.html

ImageNet visualization slides are by Rob Fergus @ NYU/Facebook http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

LeNet5 figure from: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Chain rule of derivatives: http://en.wikipedia.org/wiki/Chain_rule