Neural Network

A neural network comprises neurons connected in an acyclic graph

The outputs of neurons can become inputs to other neurons

Neural networks typically contain multiple layers of neurons

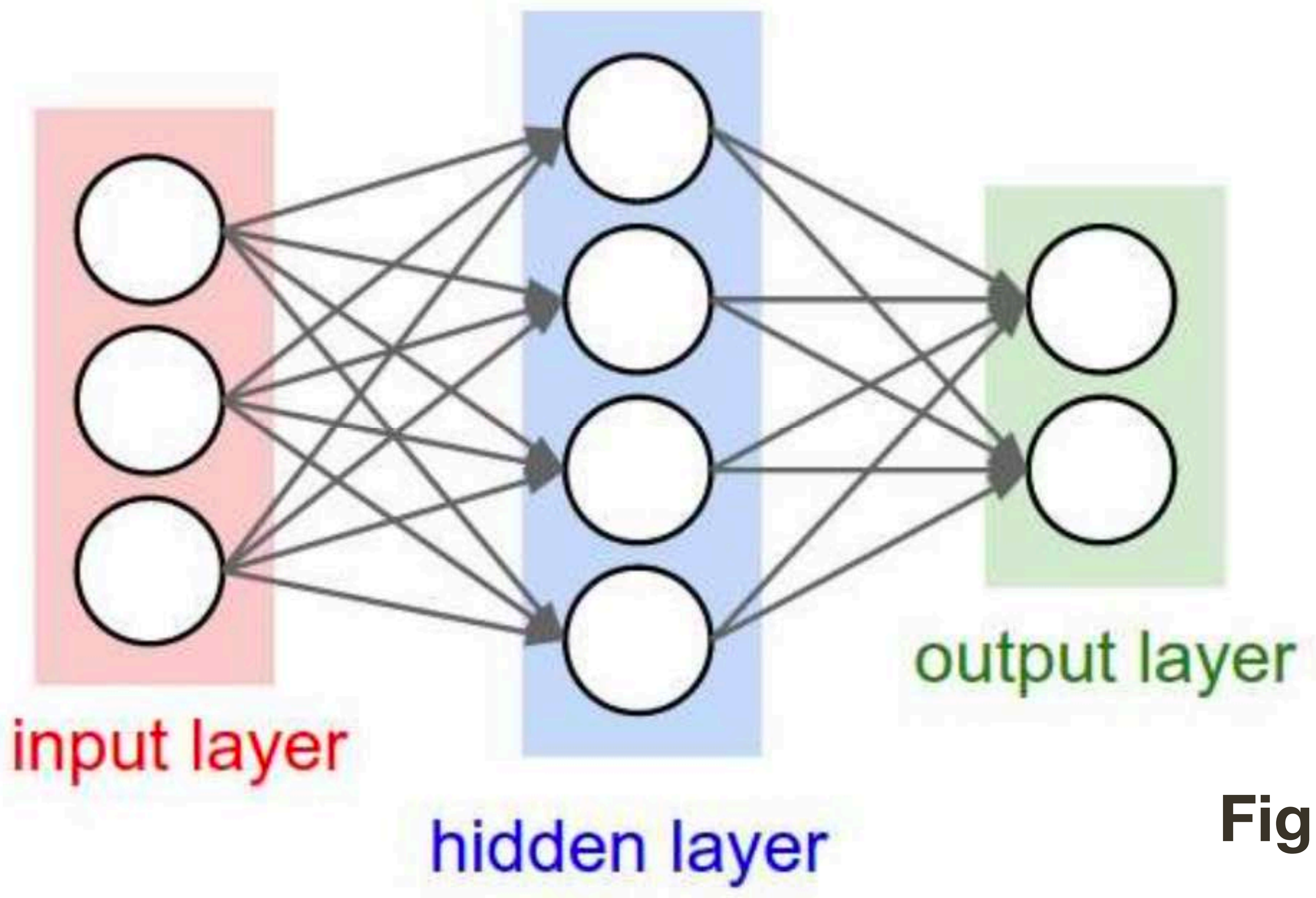
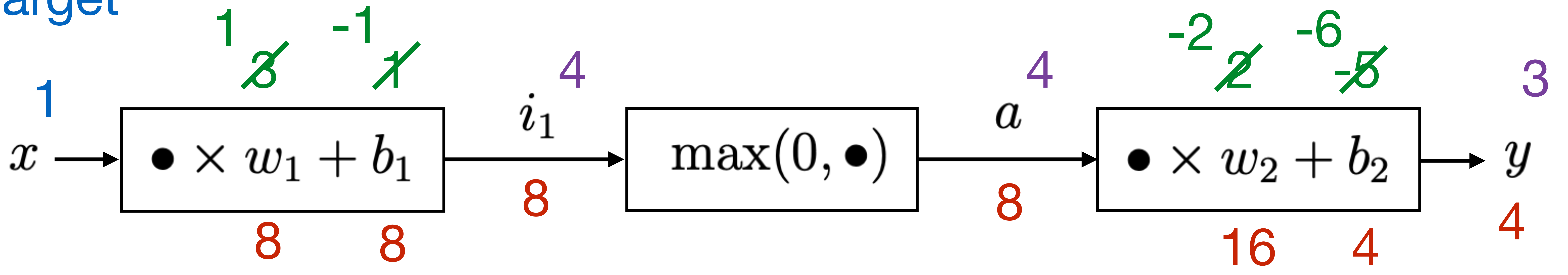


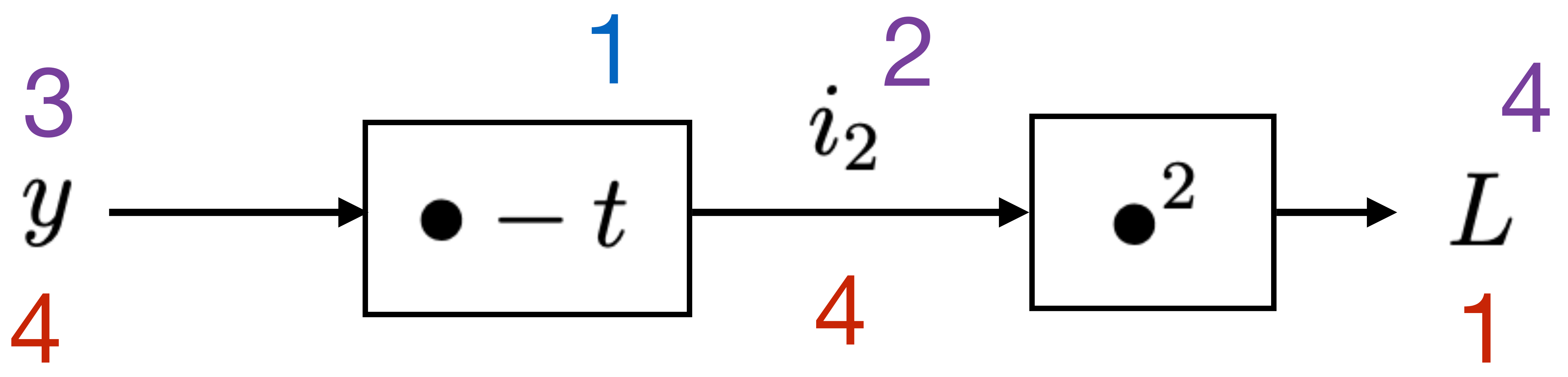
Figure credit: Fei-Fei and Karpathy

Example of a neural network with three inputs, a single hidden layer of four neurons, and an output layer of two neurons

2-Layer **Neural** Network — 1 hidden, 1 input/output

Input + Initial weights Forward pass Backward pass $= \frac{\partial L}{\partial \bullet}$
/target





Repeat: +Input/target, Forward, Backward, Update until convergence!

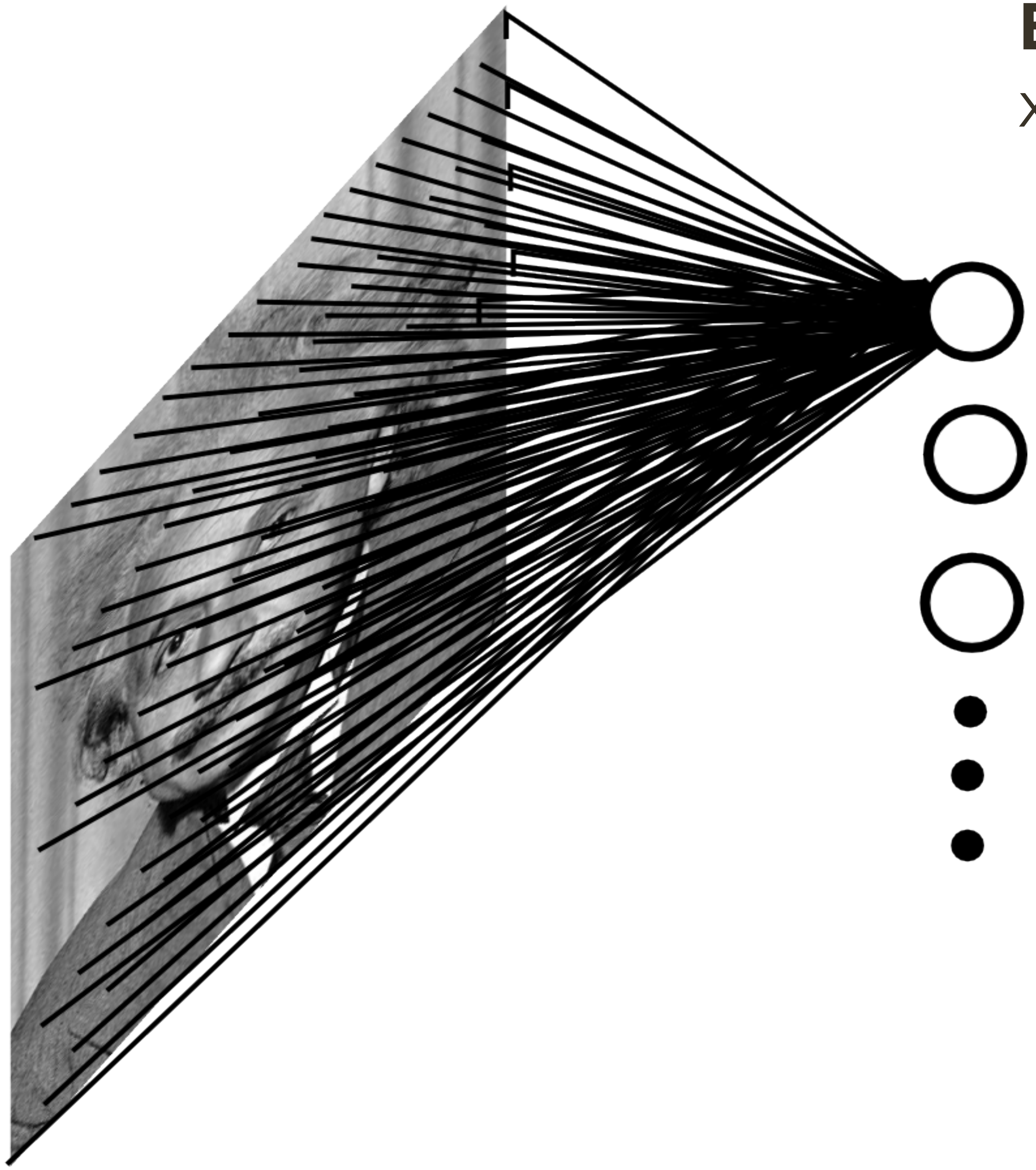
Gradient descent step

$$\begin{bmatrix} w_1\\ b_1\\ w_2\\ b_2\end{bmatrix} \to \begin{bmatrix}3\\1\\2\\-5\end{bmatrix} - \overset{1/4}{\alpha} \begin{bmatrix}8\\8\\16\\4\end{bmatrix} = \begin{bmatrix}1\\-1\\-2\\-6\end{bmatrix}$$

\+ update weights


x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial L}{\partial z}$
Upstream gradient

Fully Connected Layer



Example: 200 x 200 image (small)
x 40K hidden units (same size)

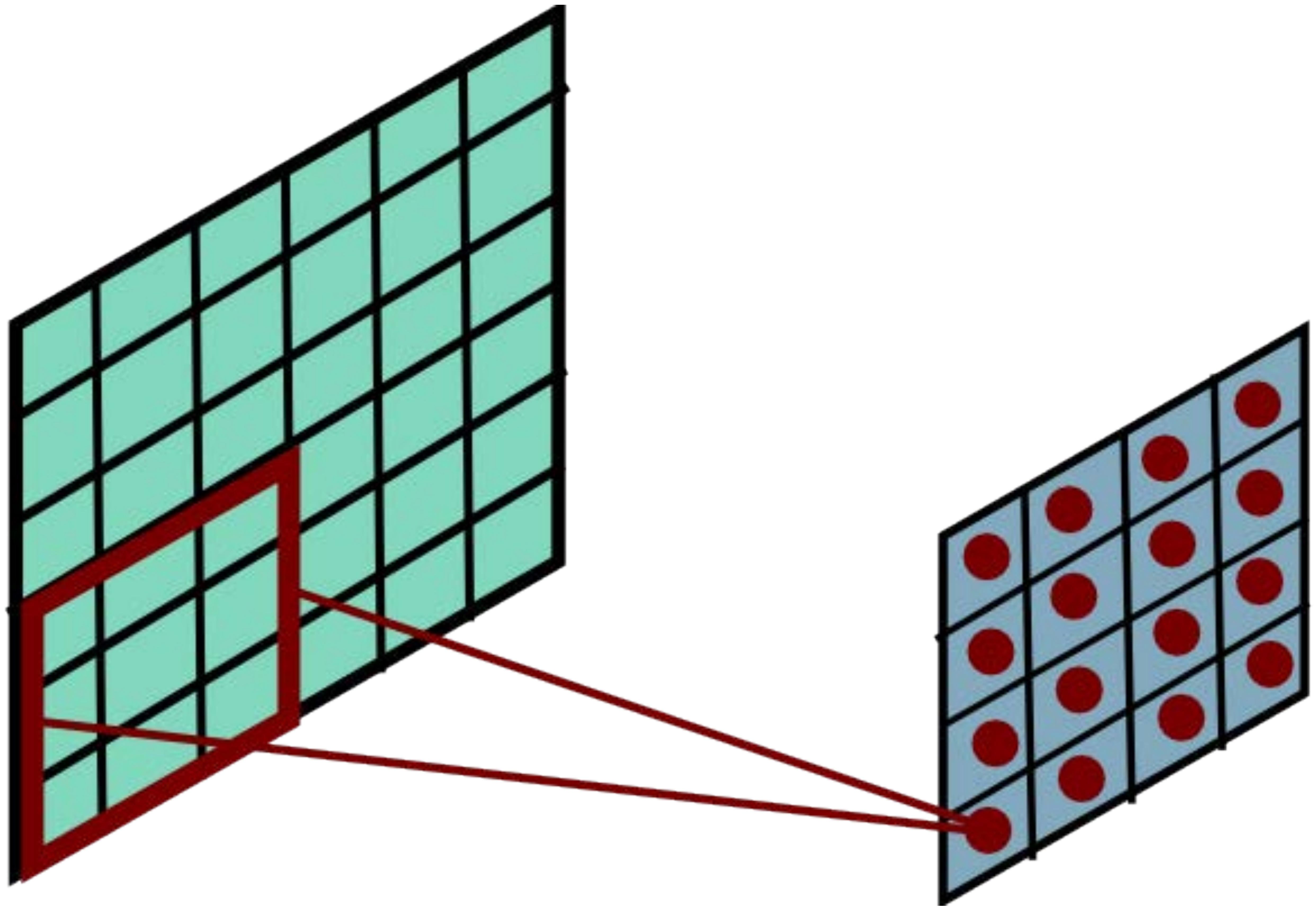
= **1.6 Billion** parameters (for one layer!)

Spatial correlations are generally local

Waste of resources + we don't have enough data to train networks this large

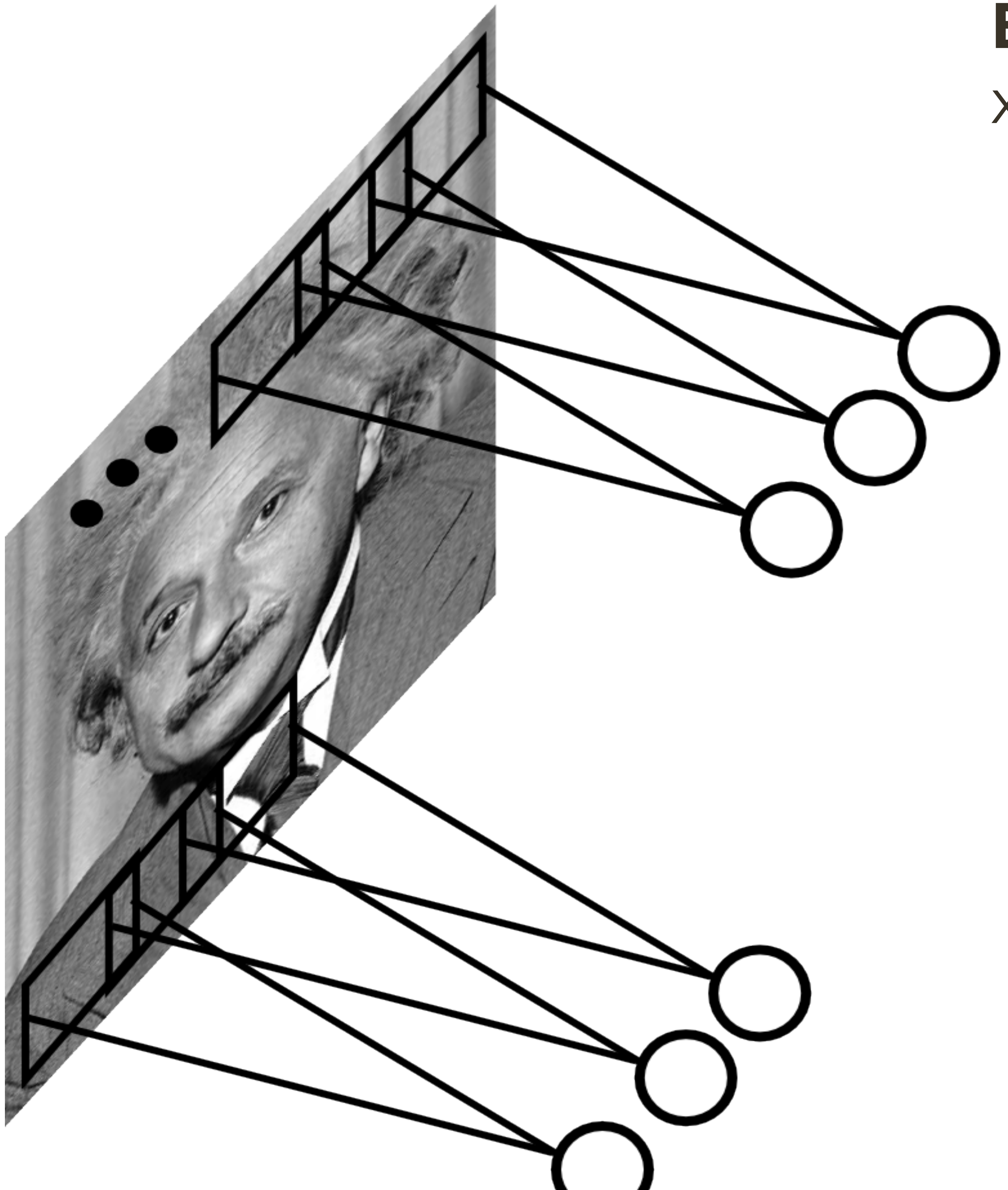
* slide from Marc'Aurelio Renzato

Convolutional Layer



* slide from Marc'Aurelio Renzato

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units (same size)

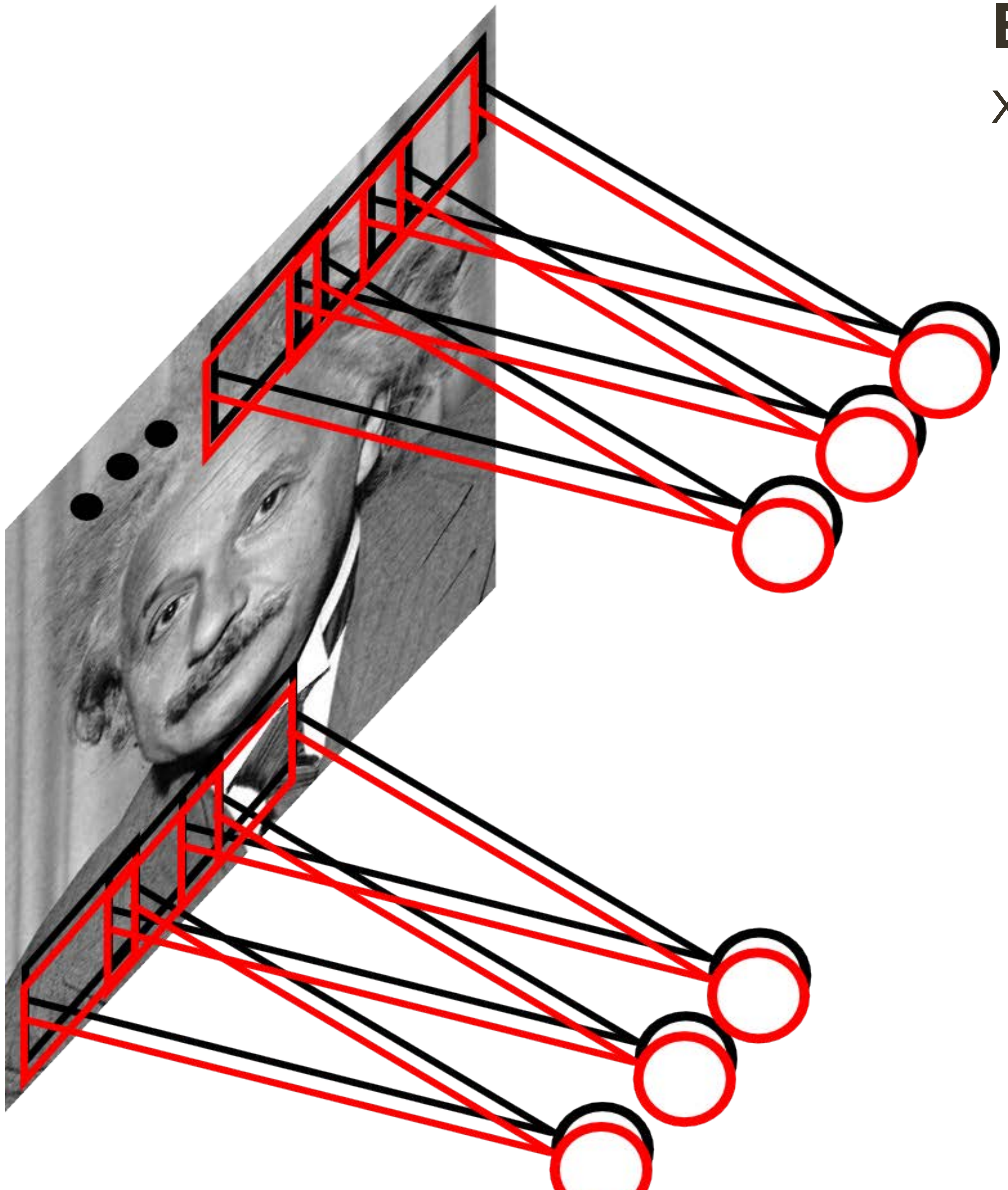
Filter size: 10 x 10

= 100 parameters

Share the same parameters across the locations (assuming input is stationary)

* slide adopted from Marc'Aurelio Renzato

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units (same size)

Filter size: 10 x 10

# of filters: 20

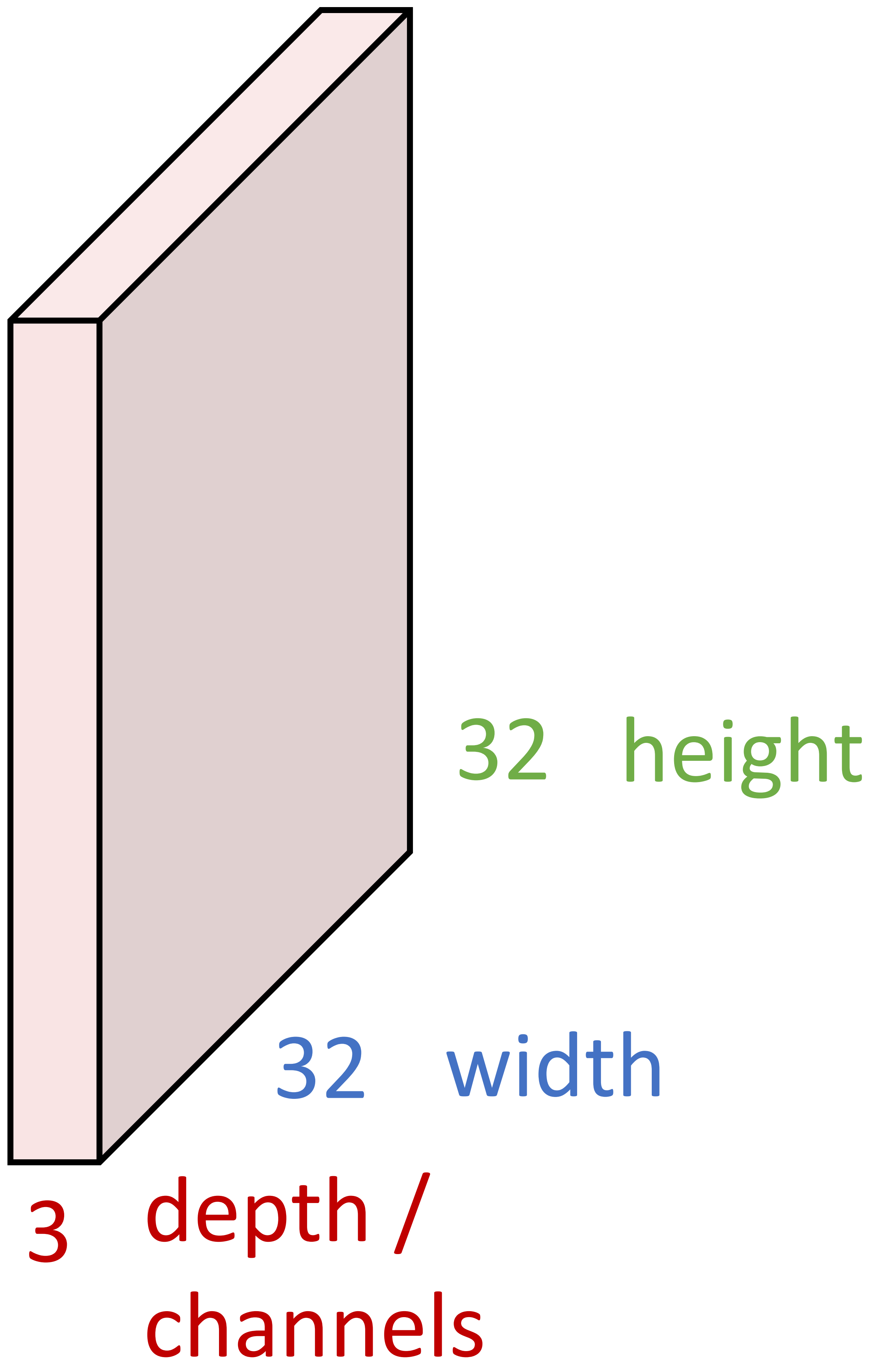
= 2000 parameters

Learn **multiple filters**
→ **multiple output channels**

* slide from Marc'Aurelio Renzato

Convolution Layer

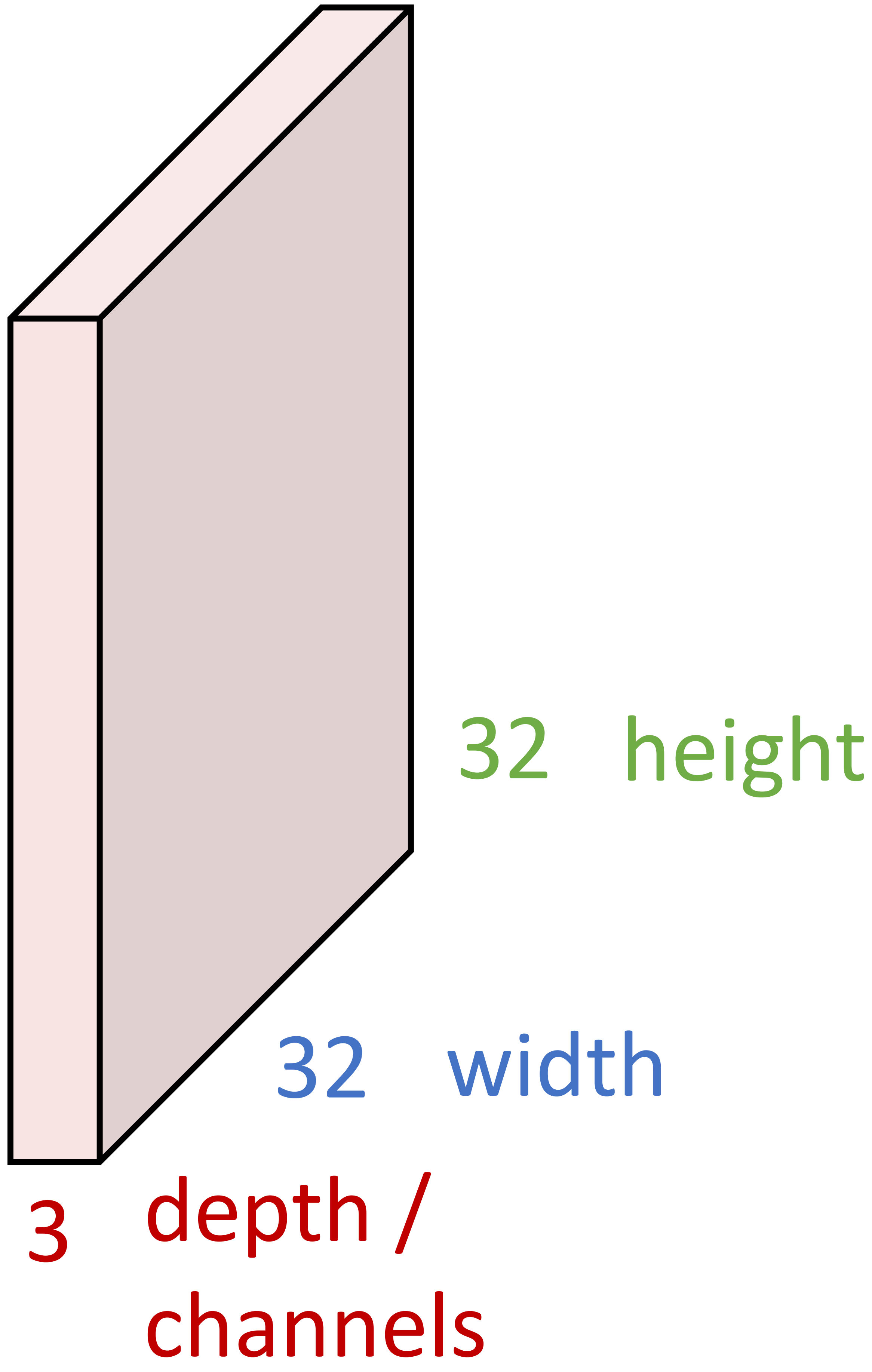
3x32x32 image: preserve spatial structure



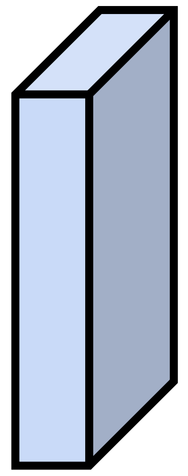
Convolution Layer

Filters always extend the full depth of the input volume

3x32x32 image

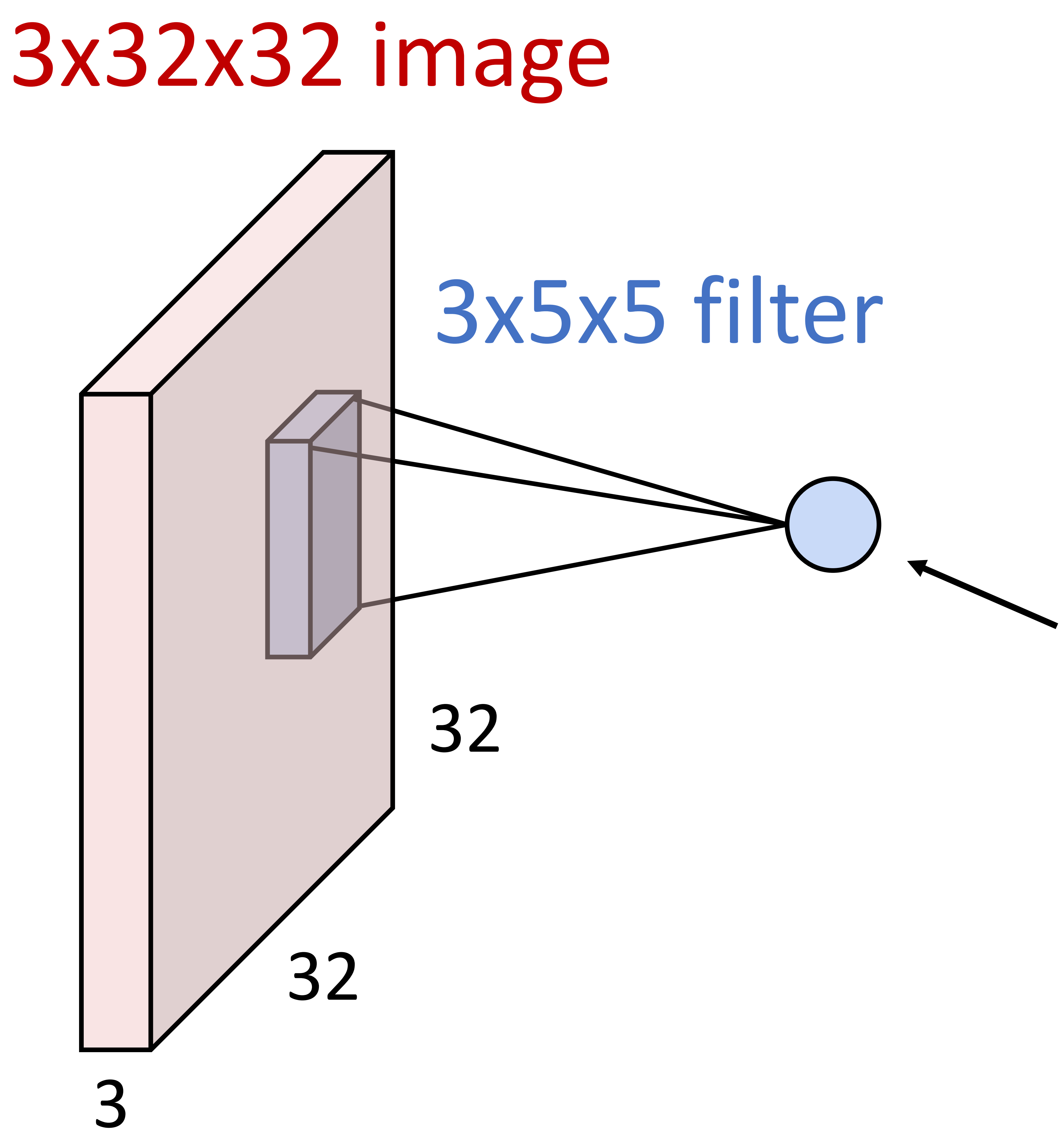


3x5x5 filter



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

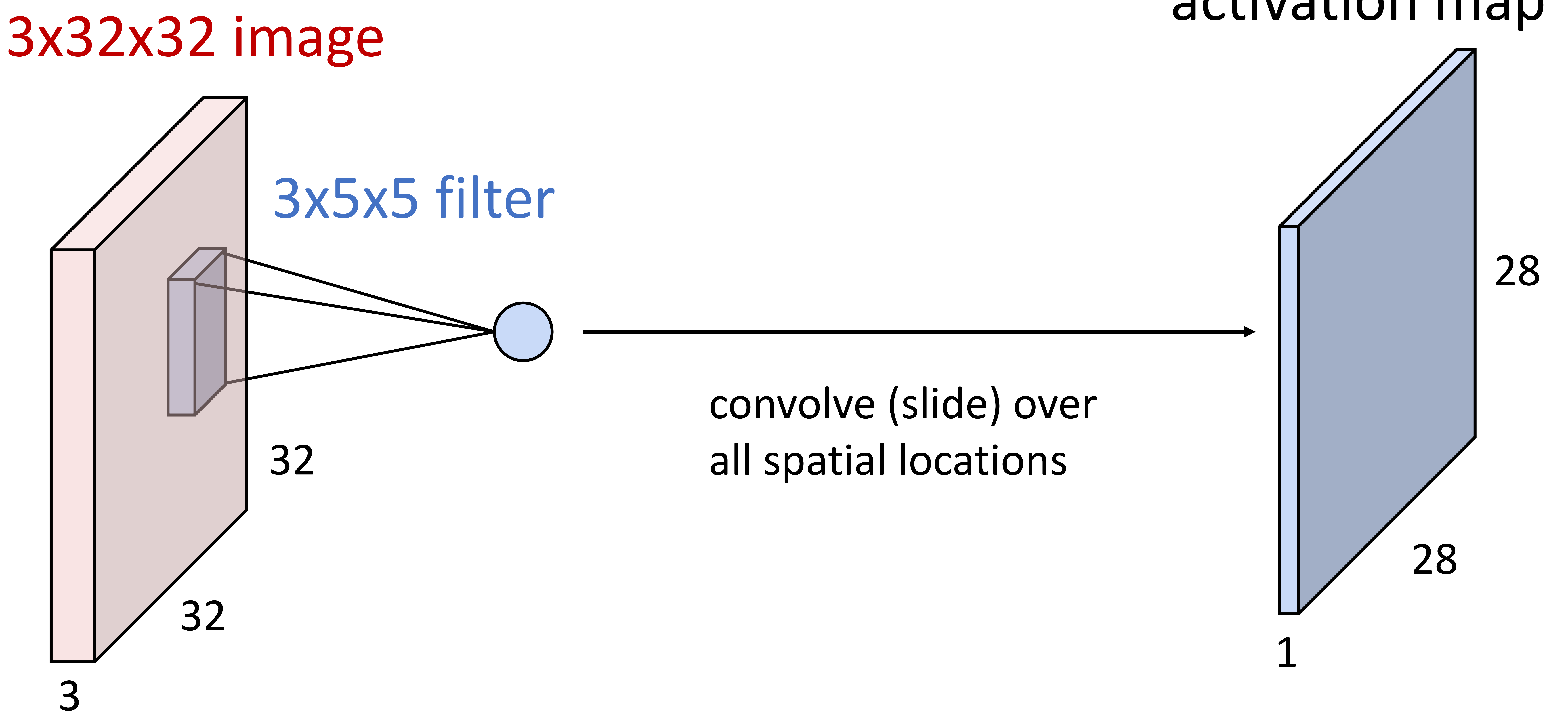
Convolution Layer



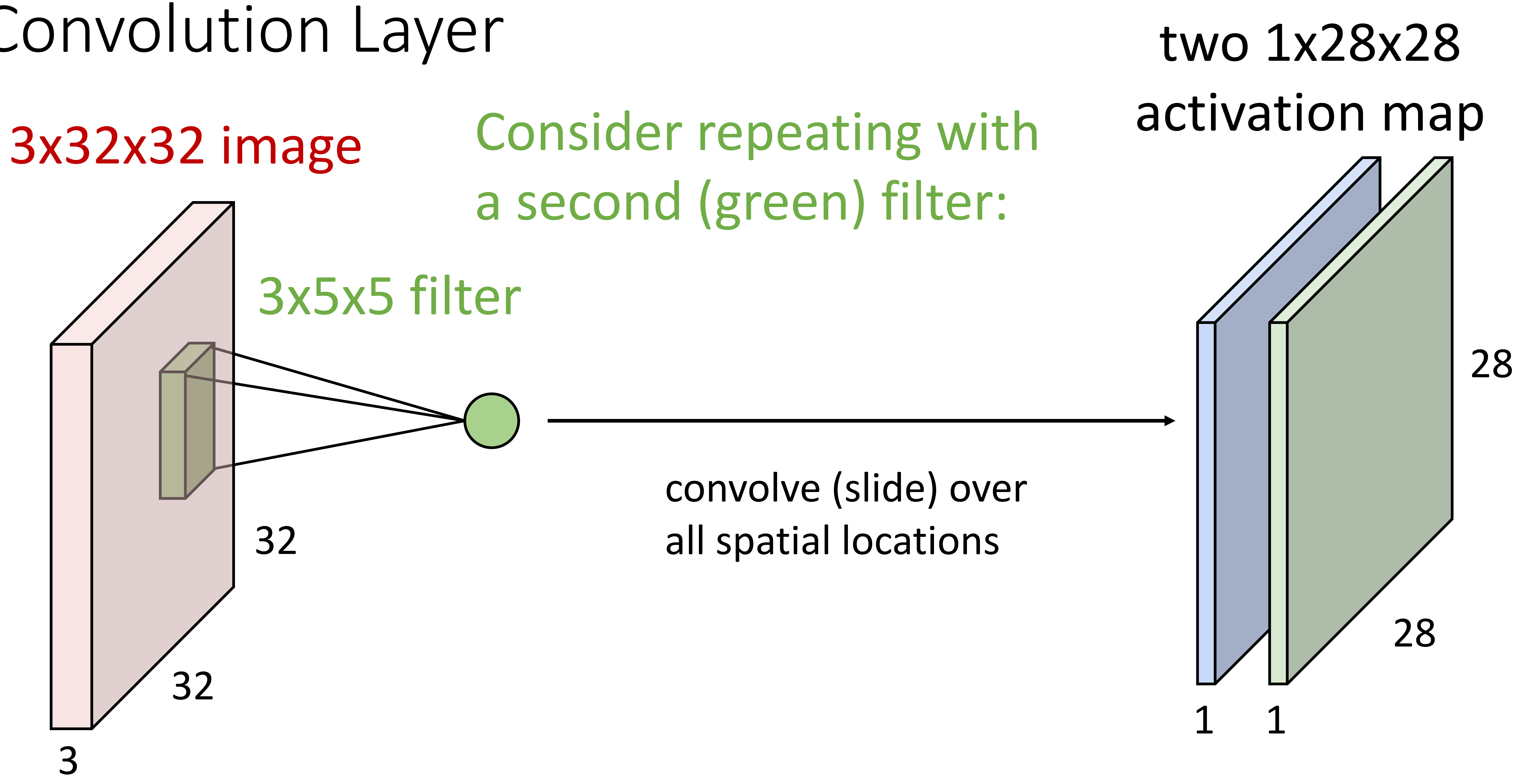
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

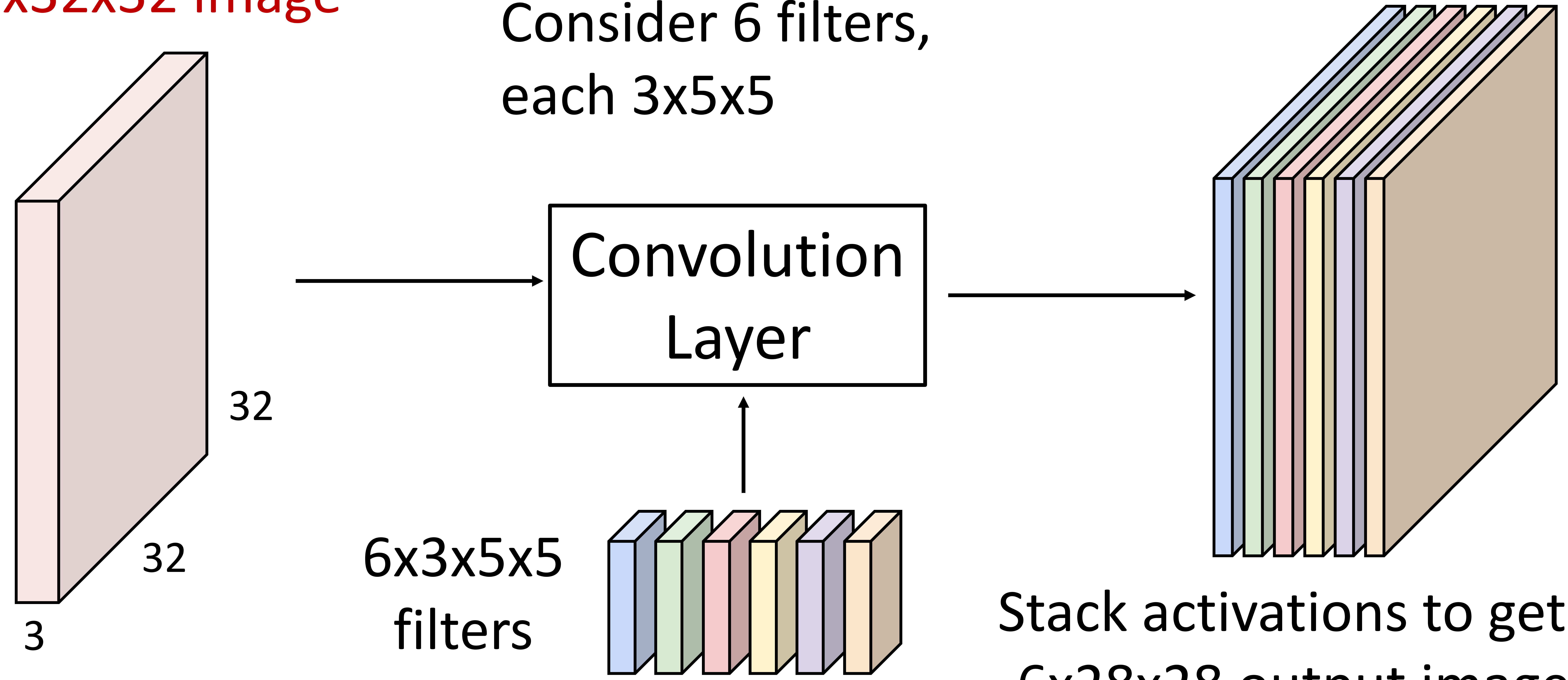
Convolution Layer



Convolution Layer



Convolution Layer

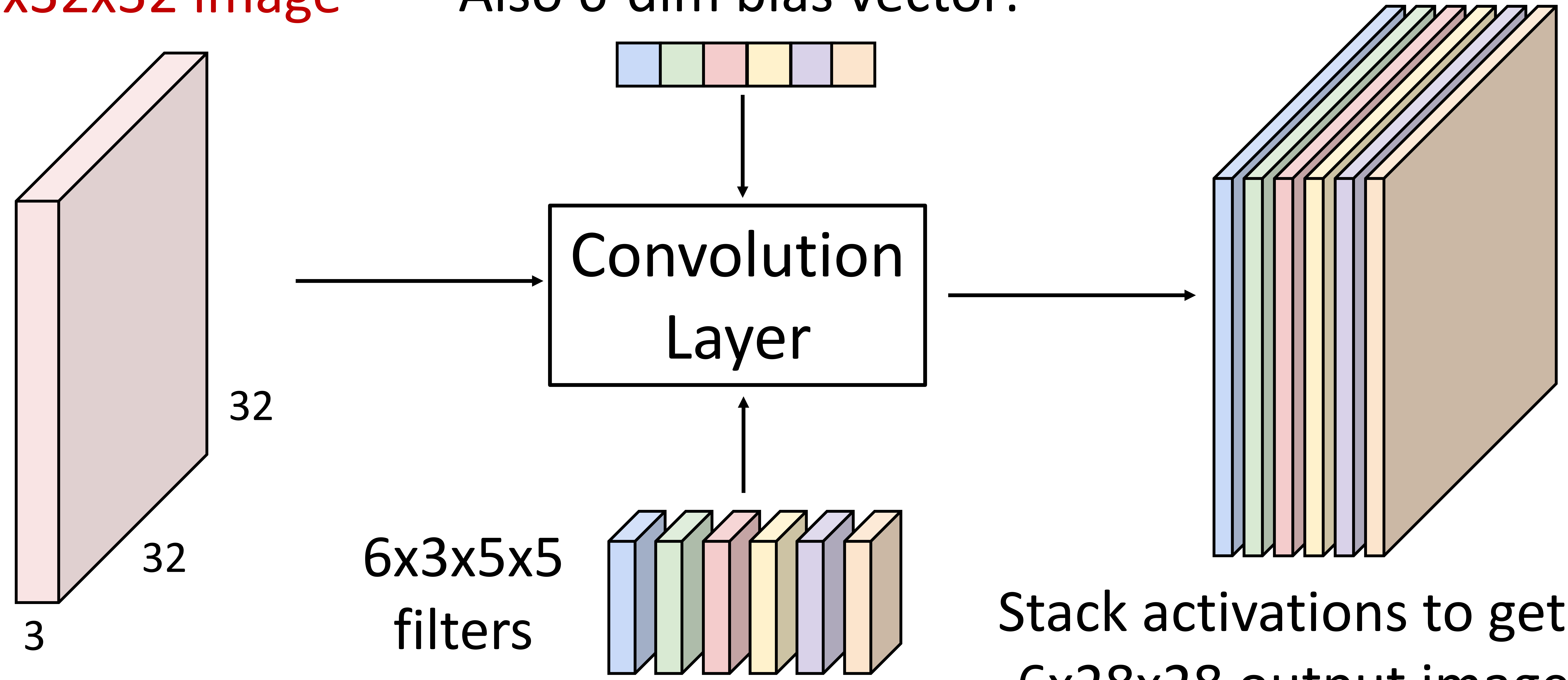


Convolution Layer

6 activation maps, each 1x28x28

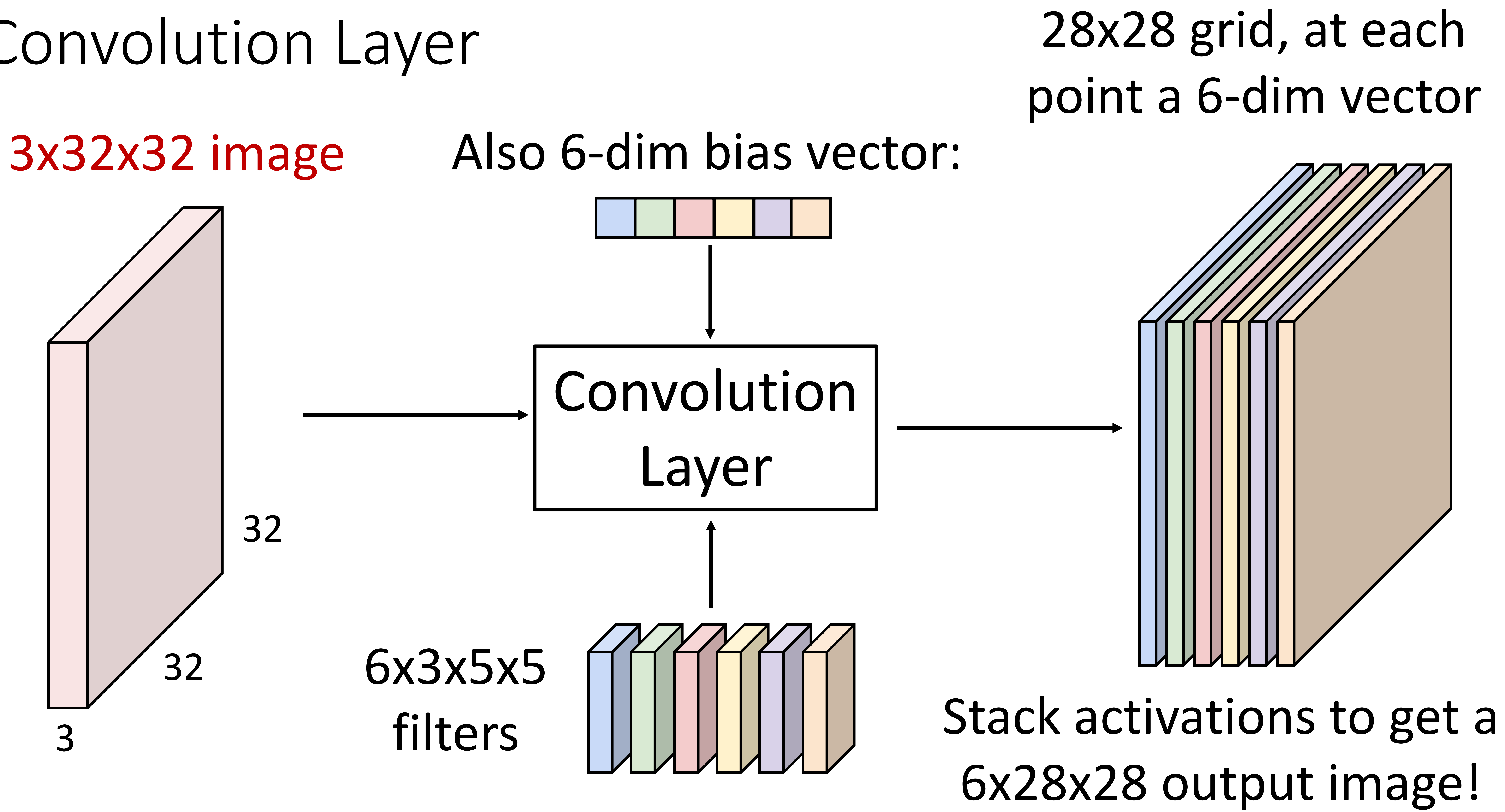
3x32x32 image

Also 6-dim bias vector:



Stack activations to get a 6x28x28 output image!

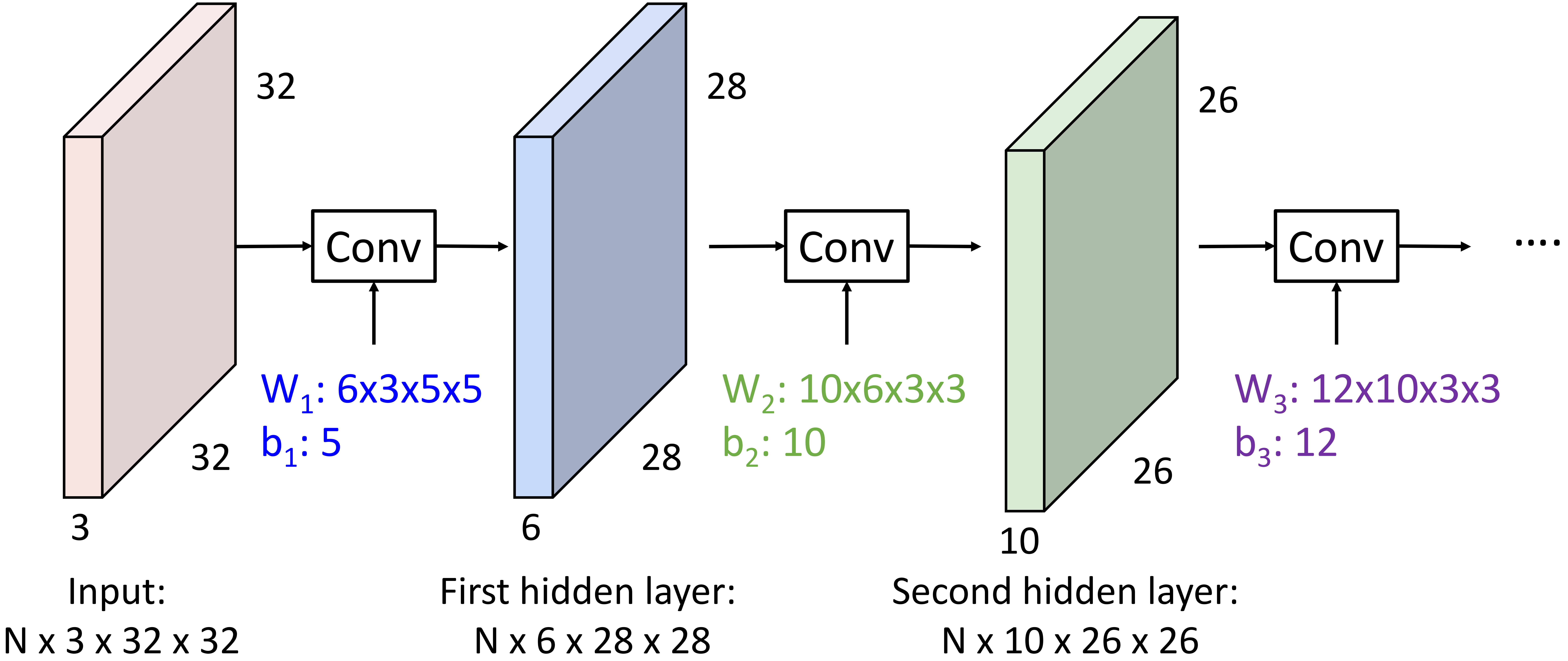
Convolution Layer



Convolution Layer

Convolution Layer

Stacking Convolutions

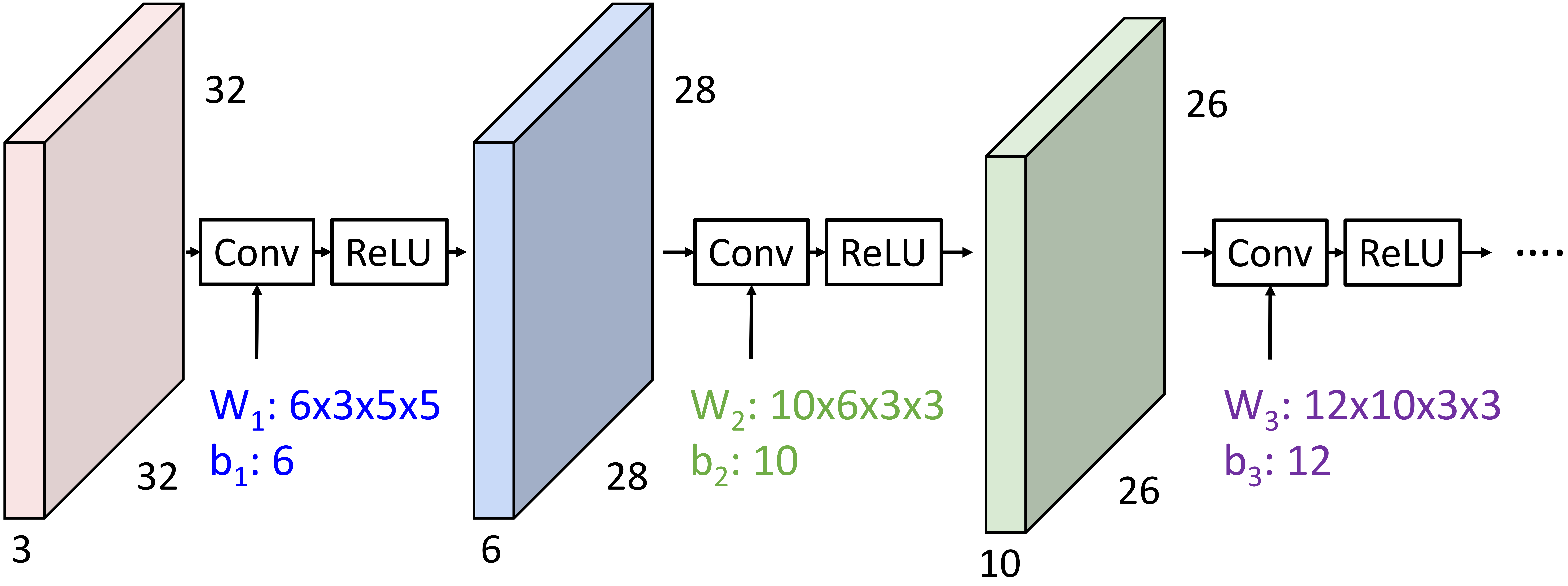


Stacking Convolutions

Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

A: We get another convolution!



Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Convolutional Neural Networks

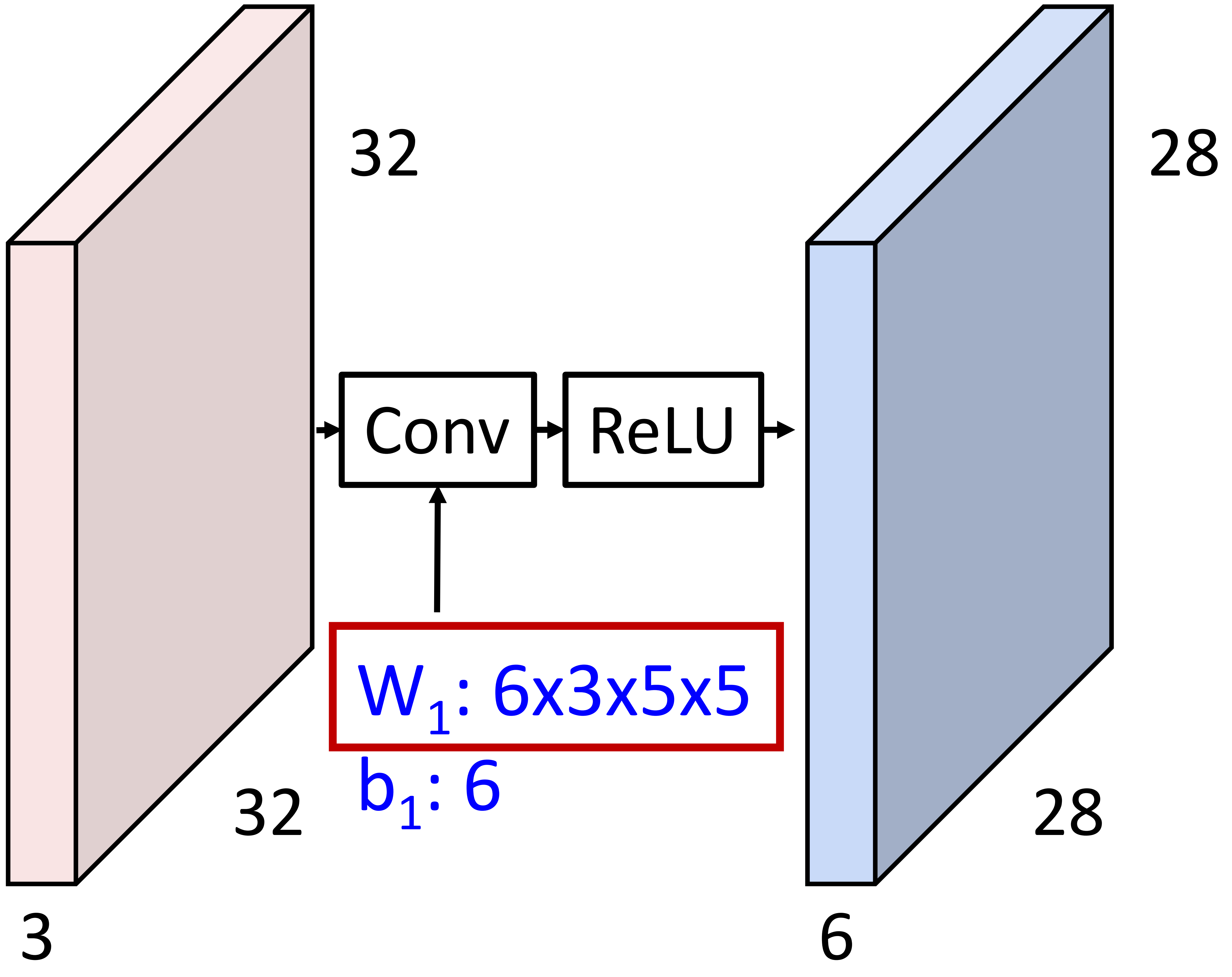
VGG-16 Network

Backward Pass for Some **Common Layers**

Convolutional layer

20.3

What do convolutional filters learn?

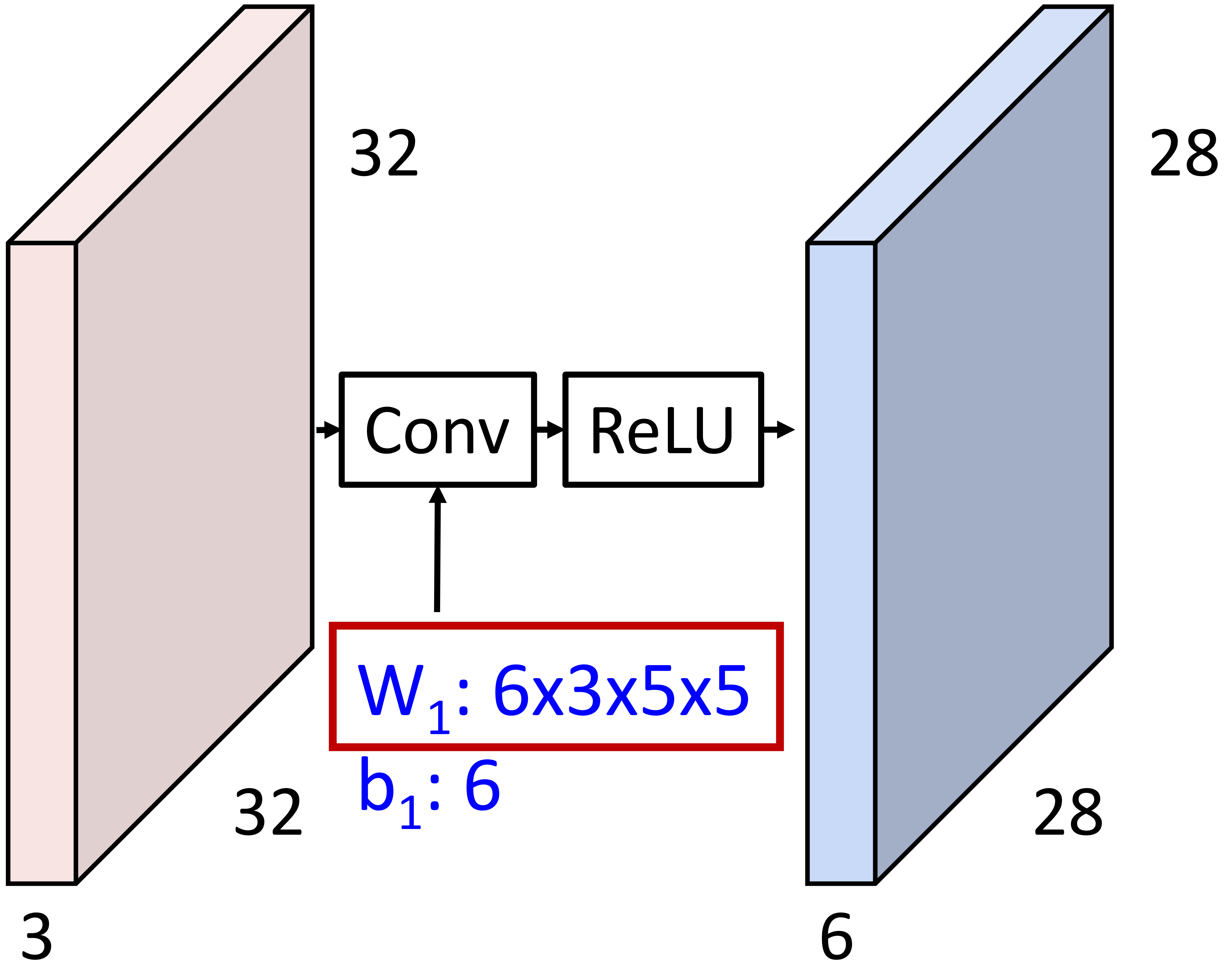


Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Linear classifier: One template per class

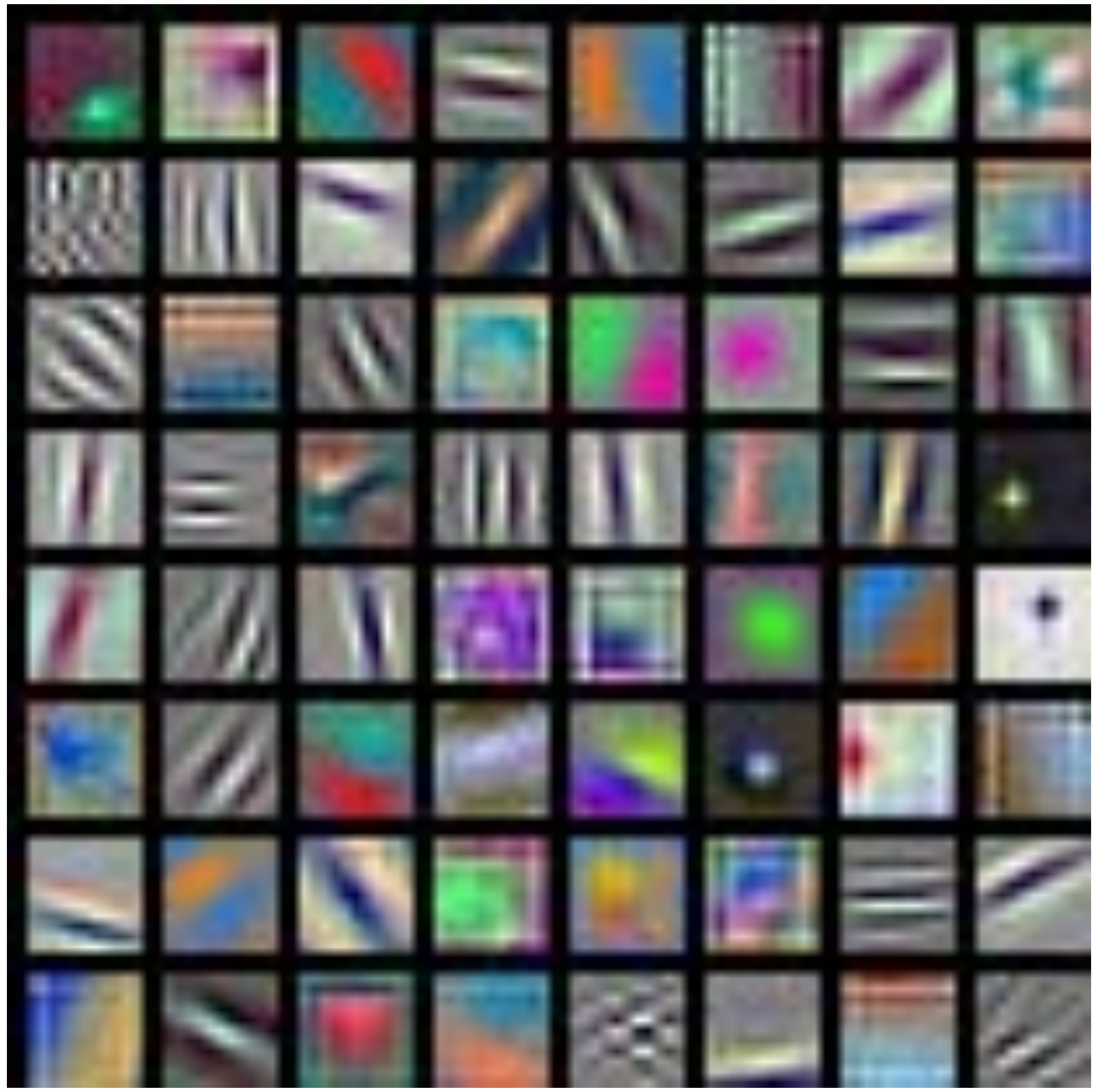
What do convolutional filters learn?



Input:
N x 3 x 32 x 32

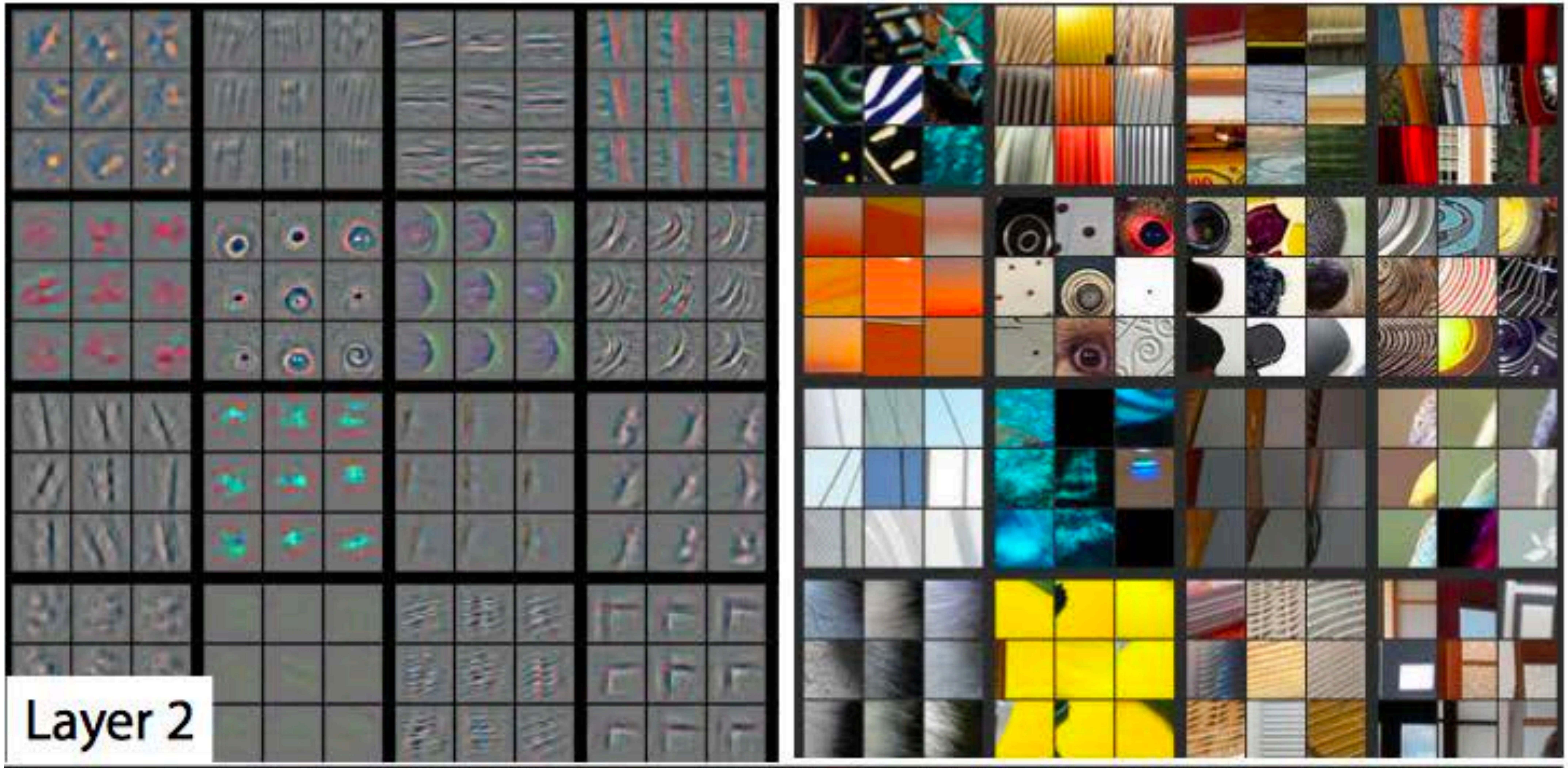
First hidden layer:
N x 6 x 28 x 28

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

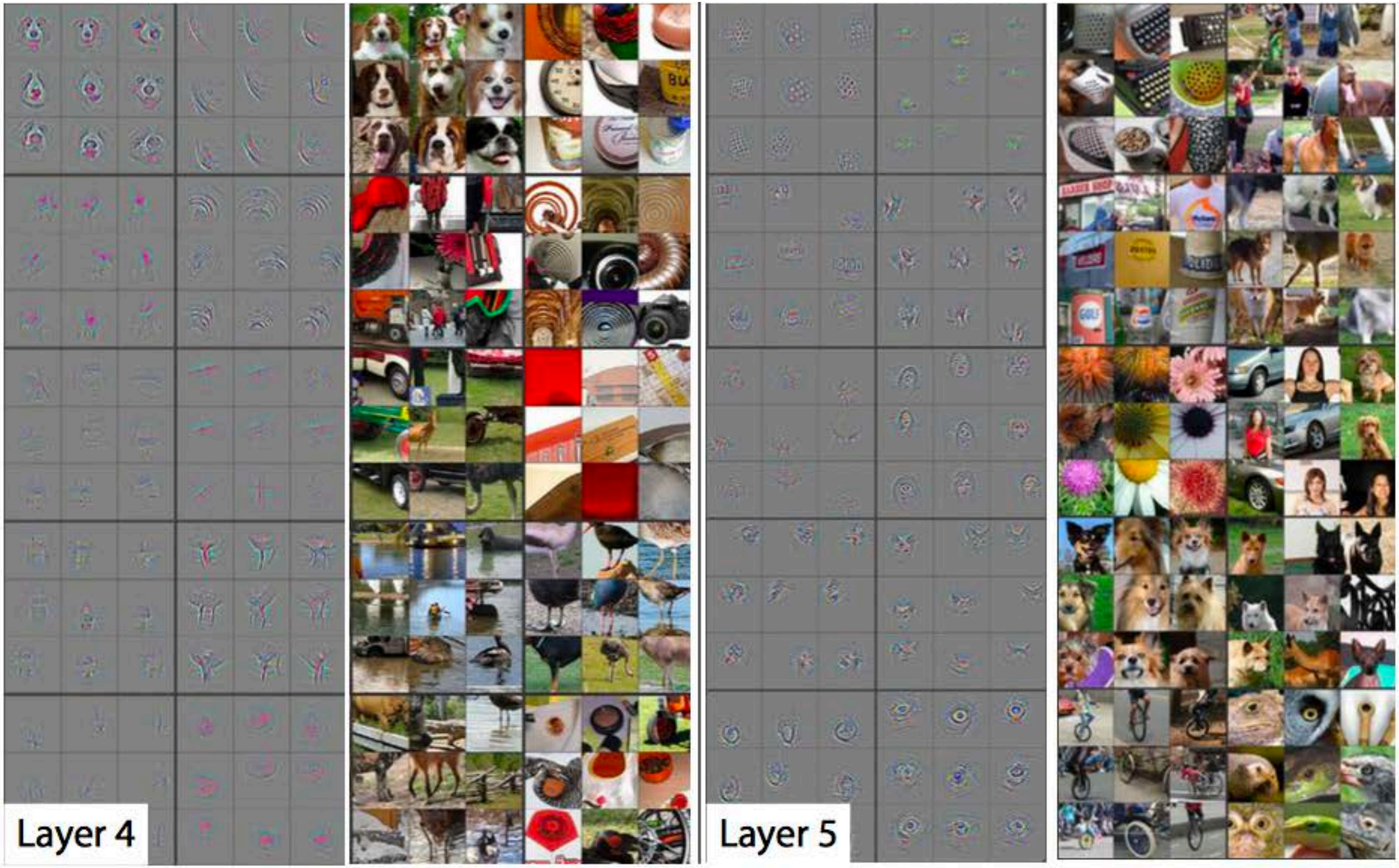


AlexNet: 64 filters, each 3x11x11

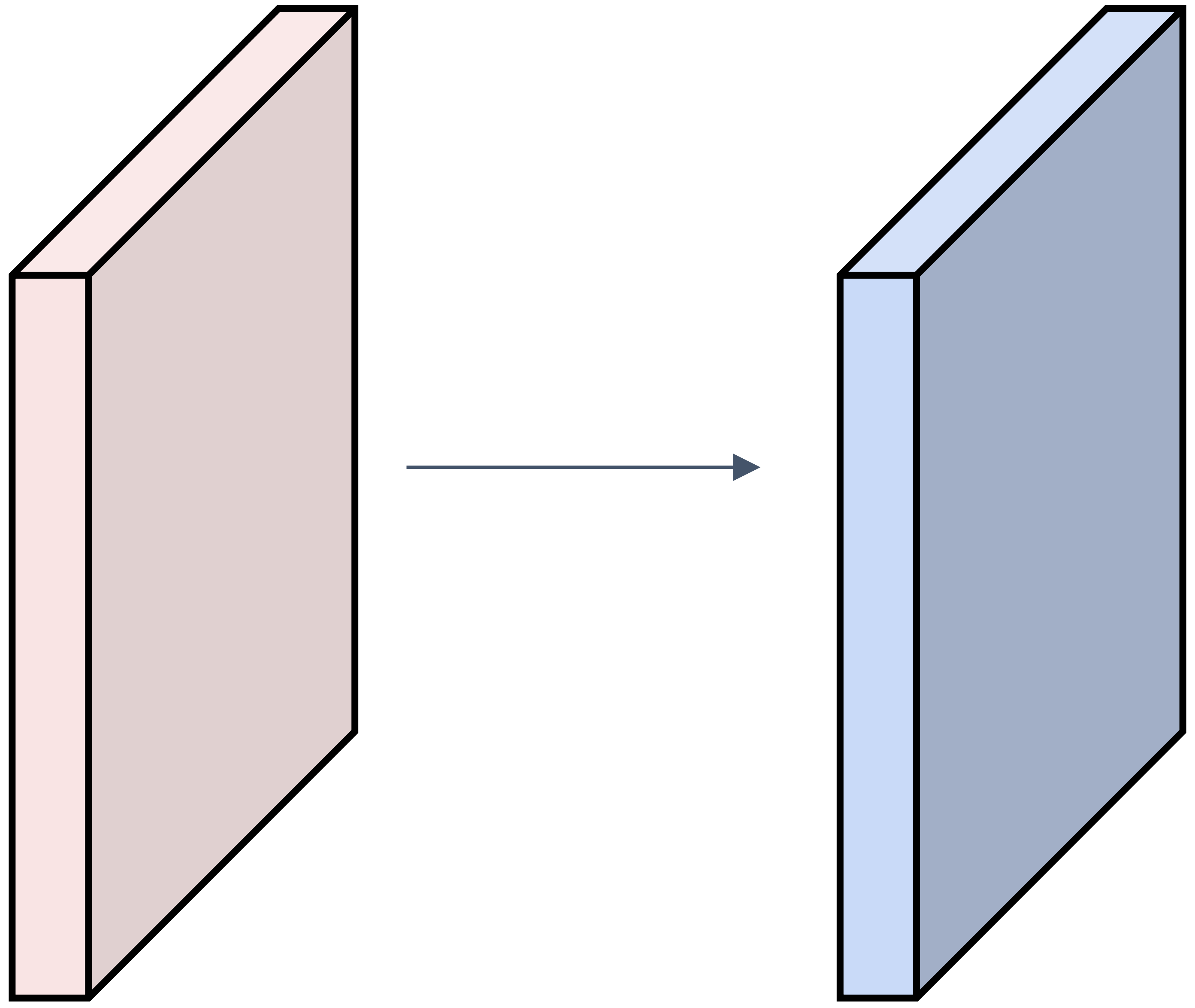
What **filters** do networks learn?



[Zeiler and Fergus, 2013]


Layer 4
Layer 5

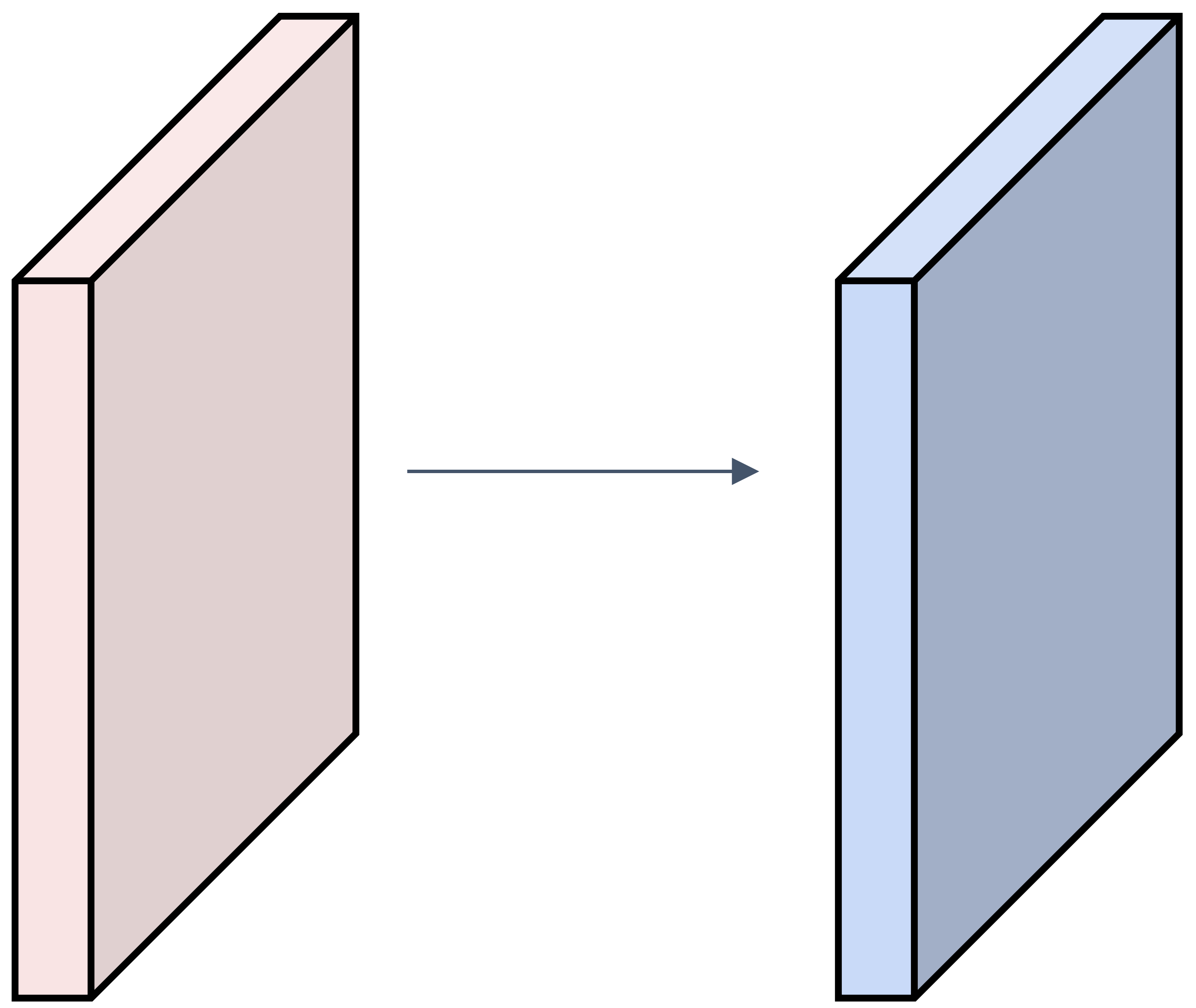
Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Convolution Example



Input volume: 3 x **32** x **32**

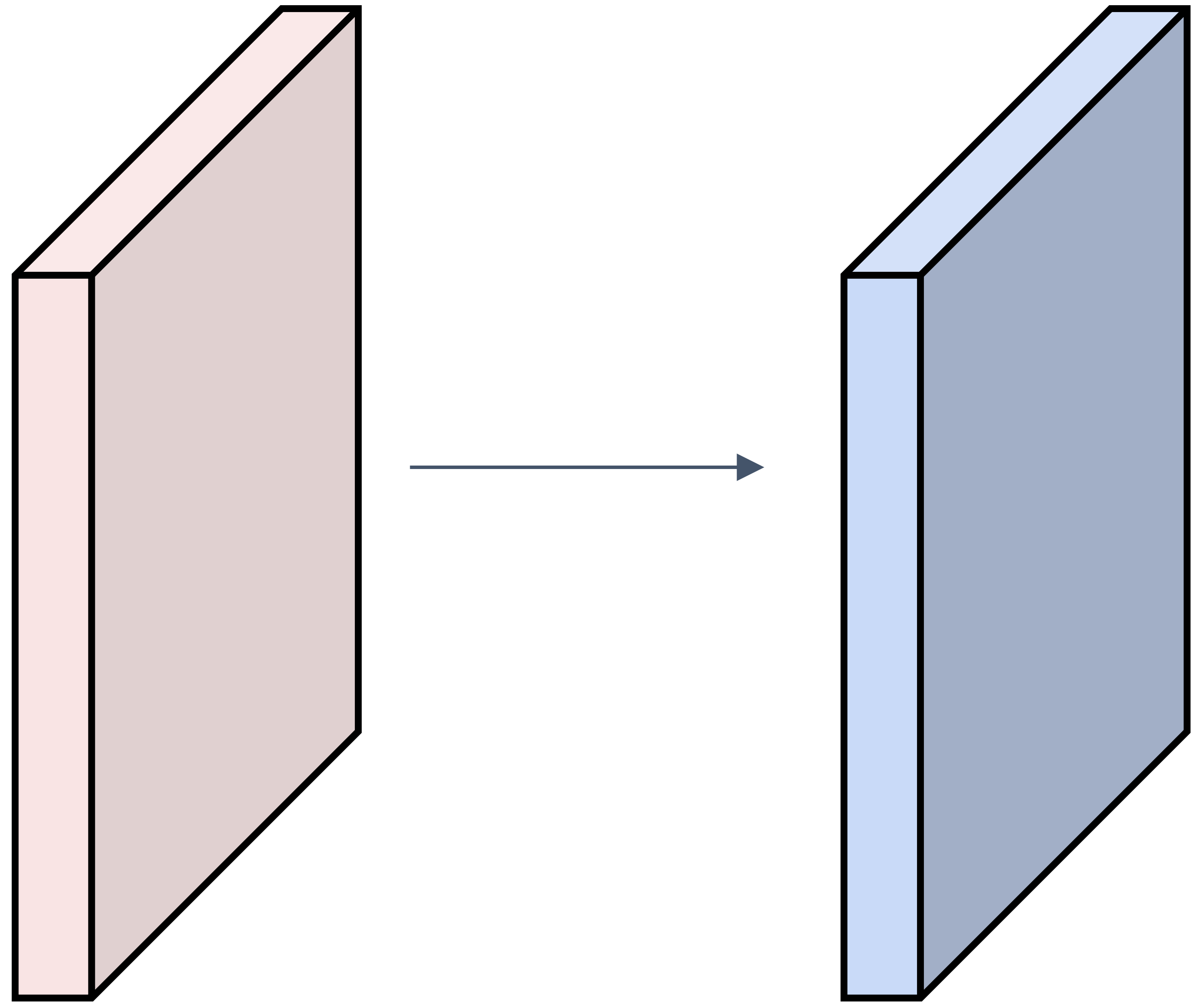
10 **5x5** filters with stride **1**, pad **2**

Output volume size:

(**32**+2*2-**5**)/**1**+1 = 32 spatially, so

10 x 32 x 32

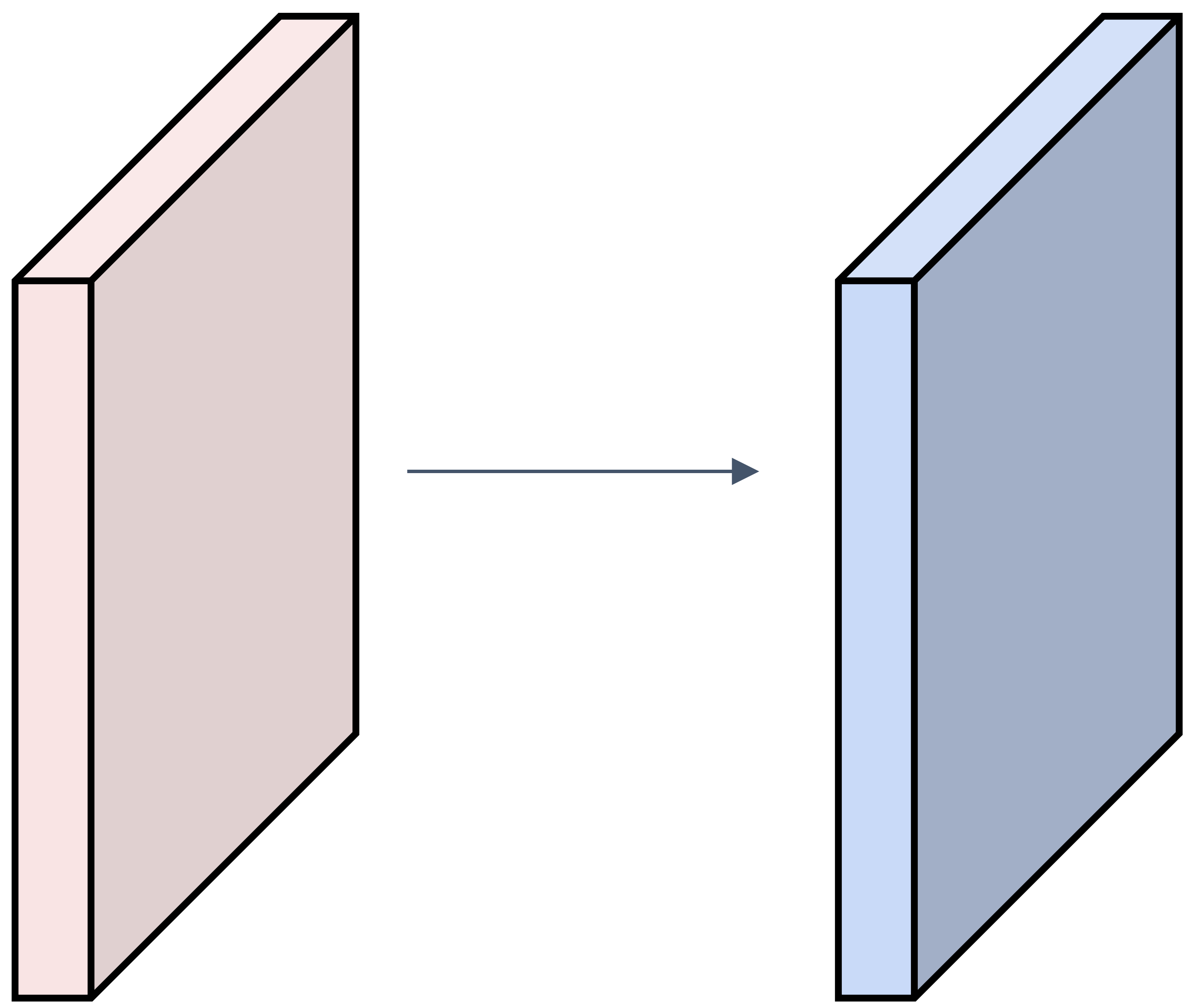
Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?

Convolution Example



Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

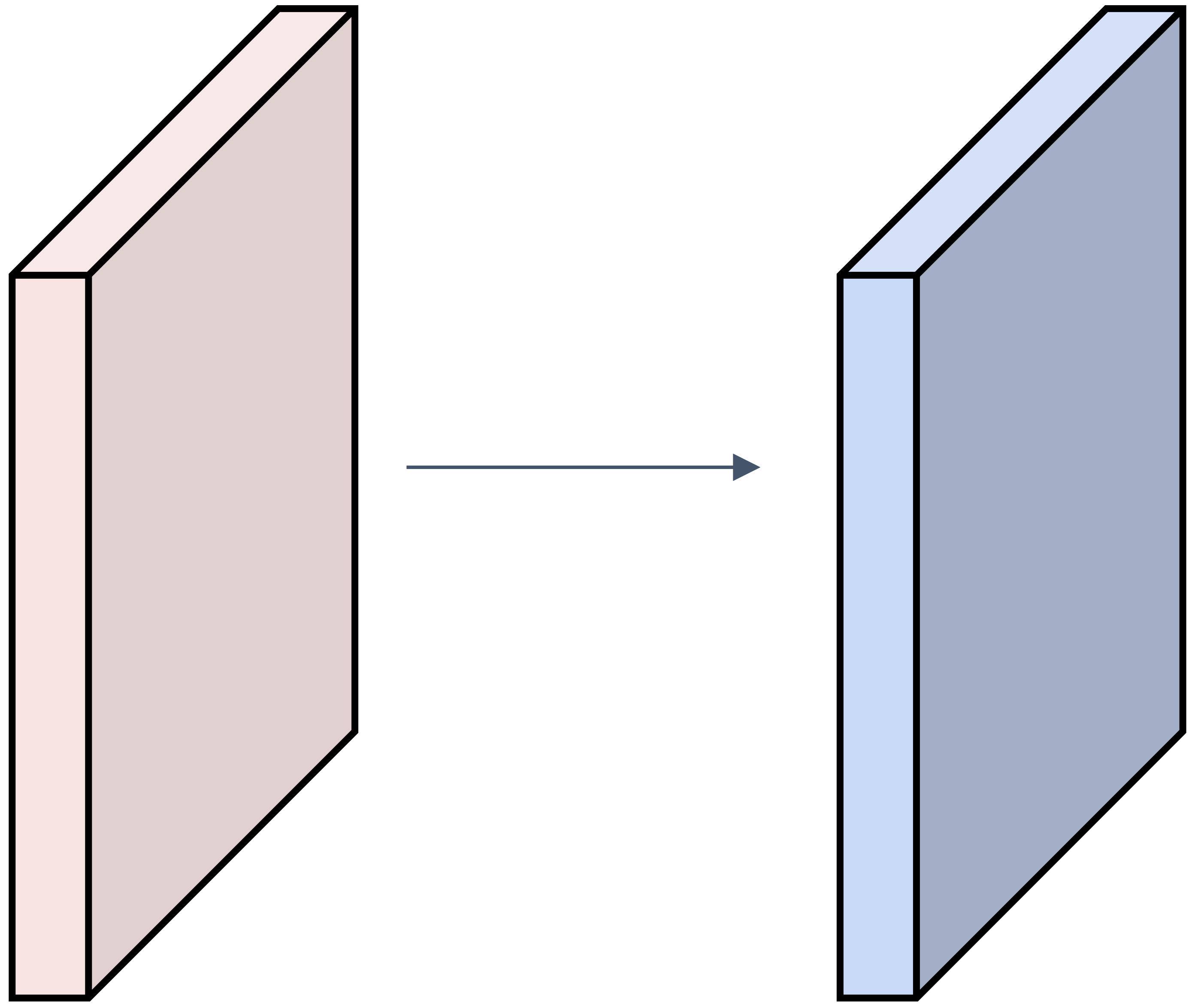
Output volume size: 10 x 32 x 32

Number of learnable parameters: **760**

Parameters per filter: 3*5*5 + 1 (for bias) = **76**

10 filters, so total is 10 * **76** = **760**

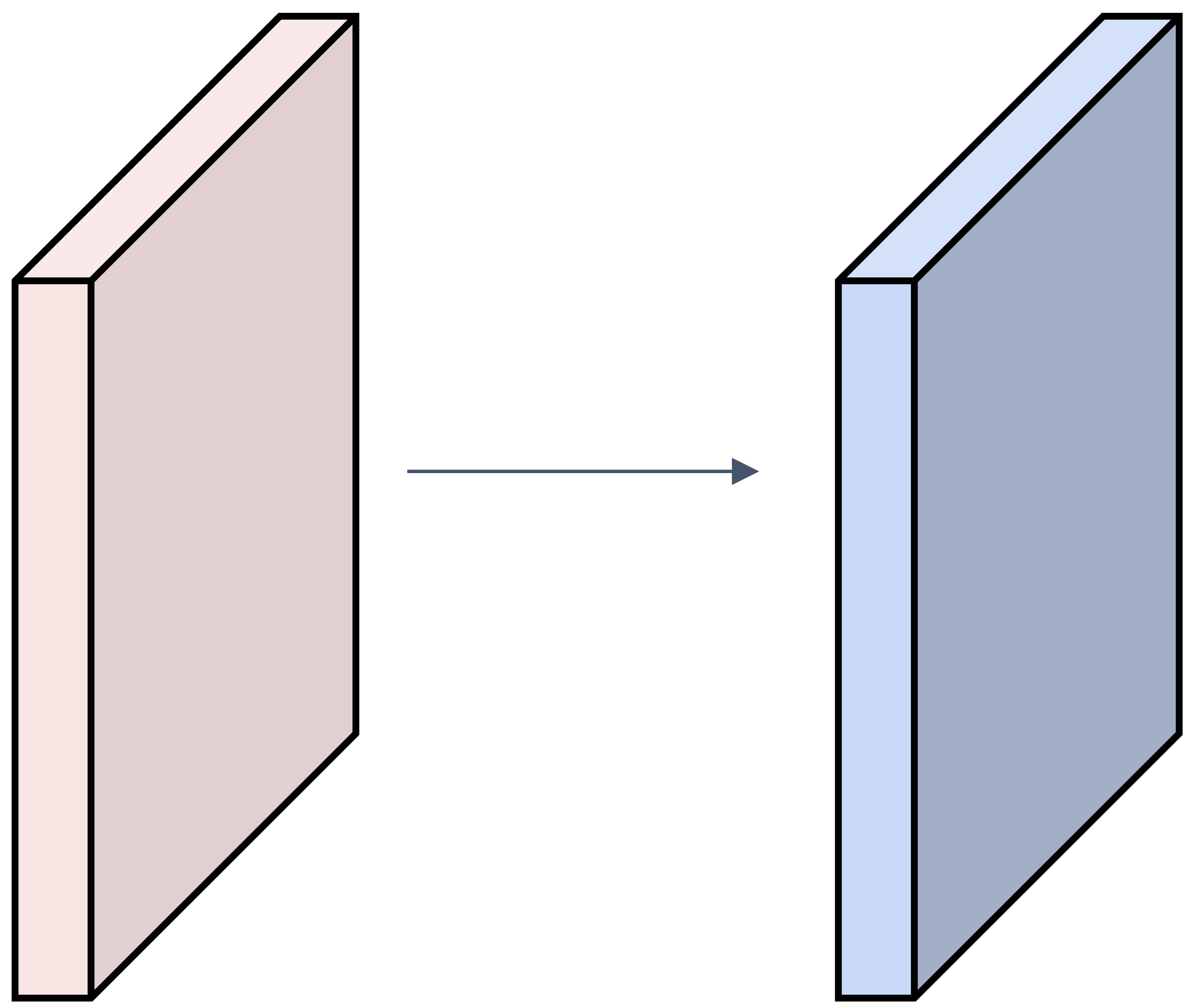
Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?

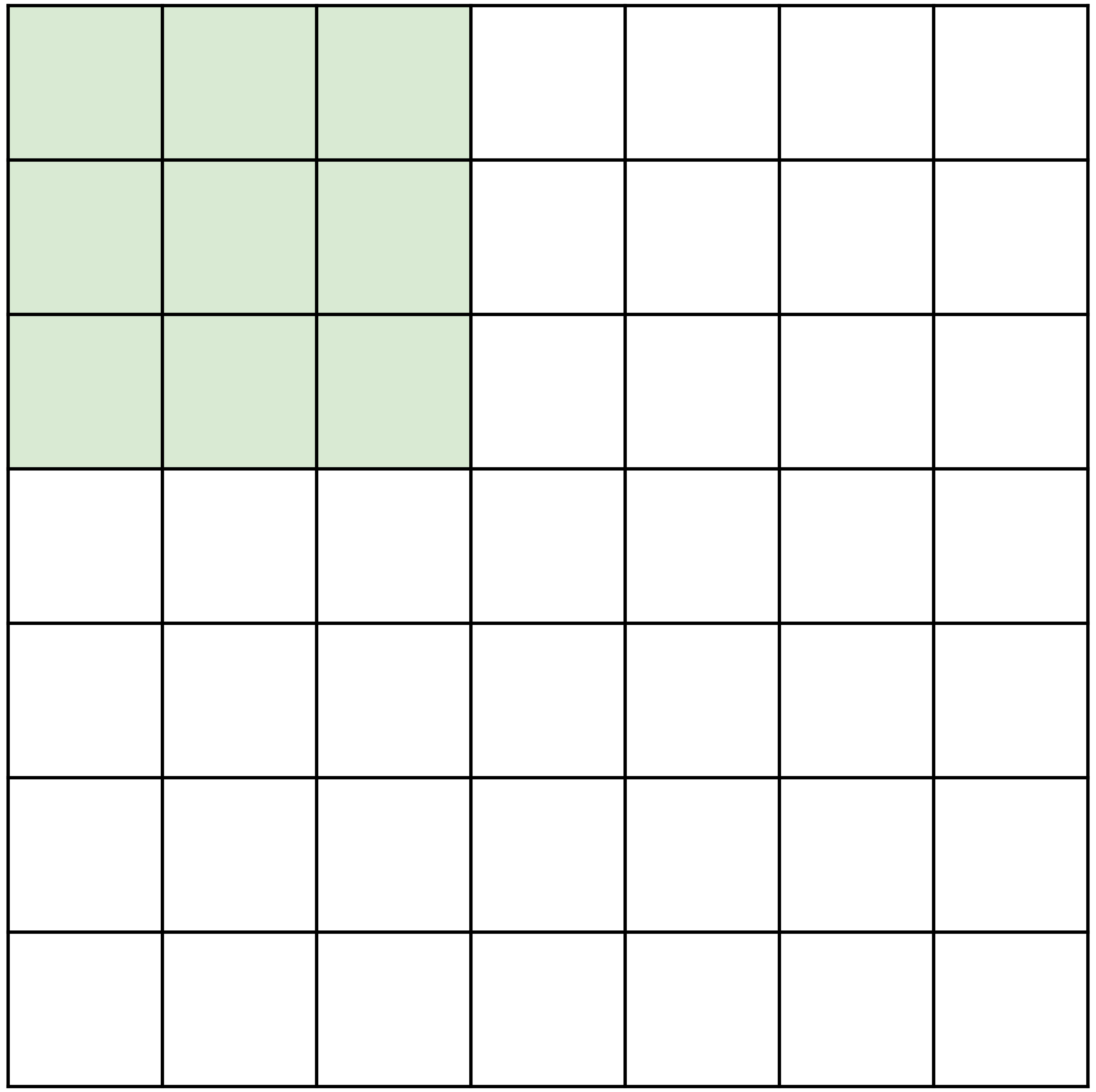
Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

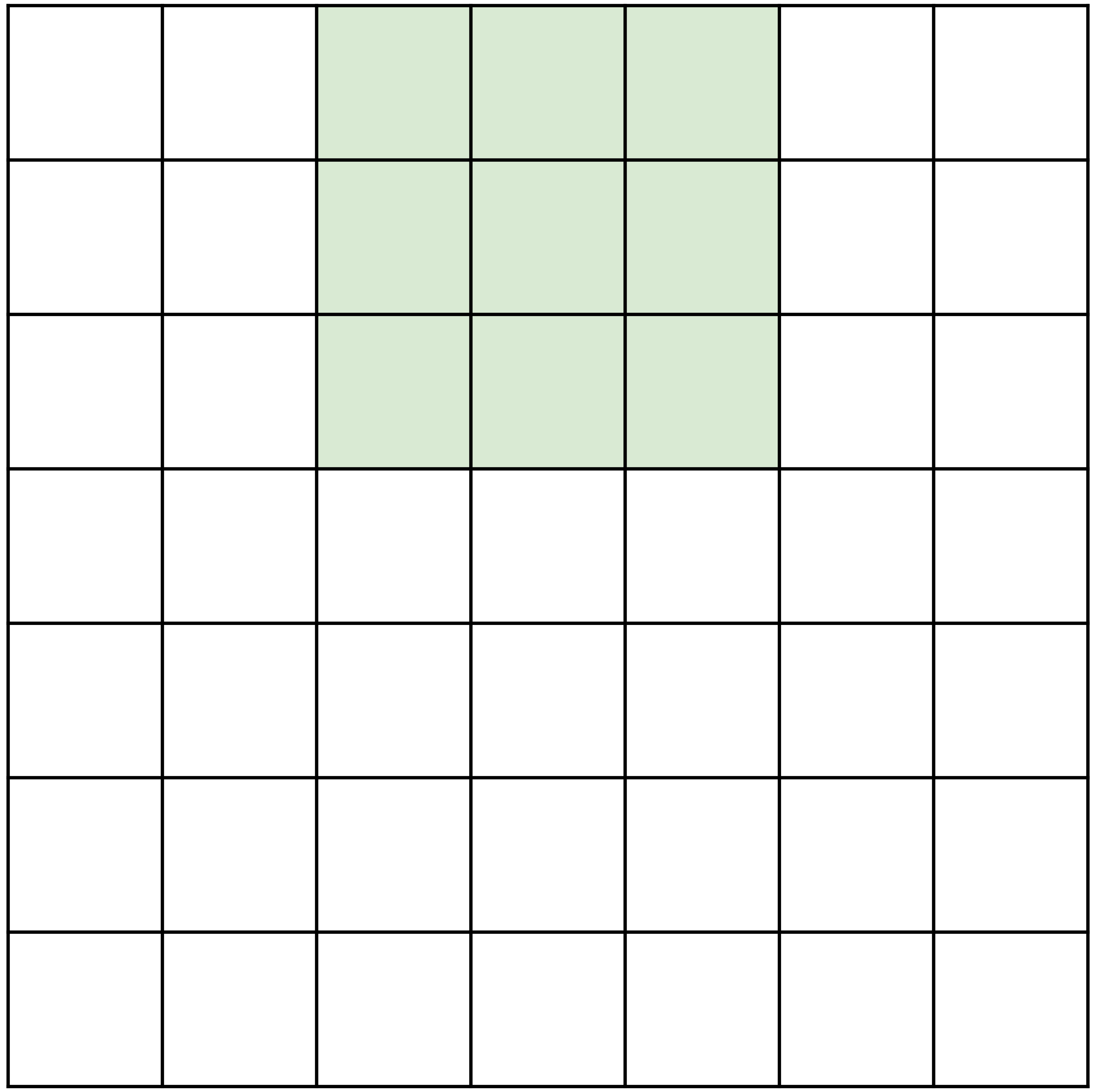
Output volume size: **10 x 32 x 32**
Number of learnable parameters: 760
Number of multiply-add operations: **768,000**
10*32*32 = 10,240 outputs; each output is the inner product of two **3**x**5x5** tensors (75 elems); total = 75*10240 = **768K**

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution

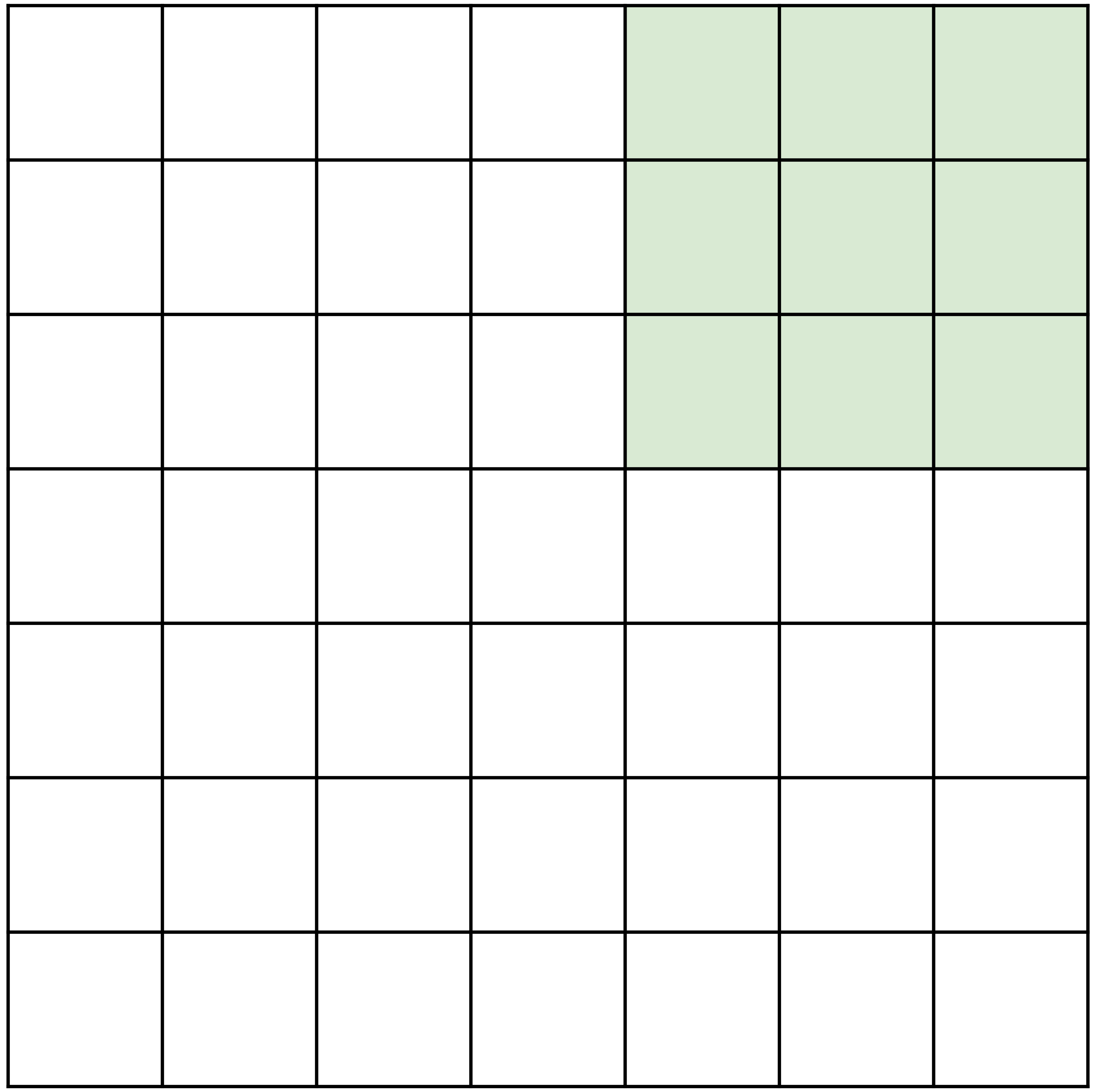


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution



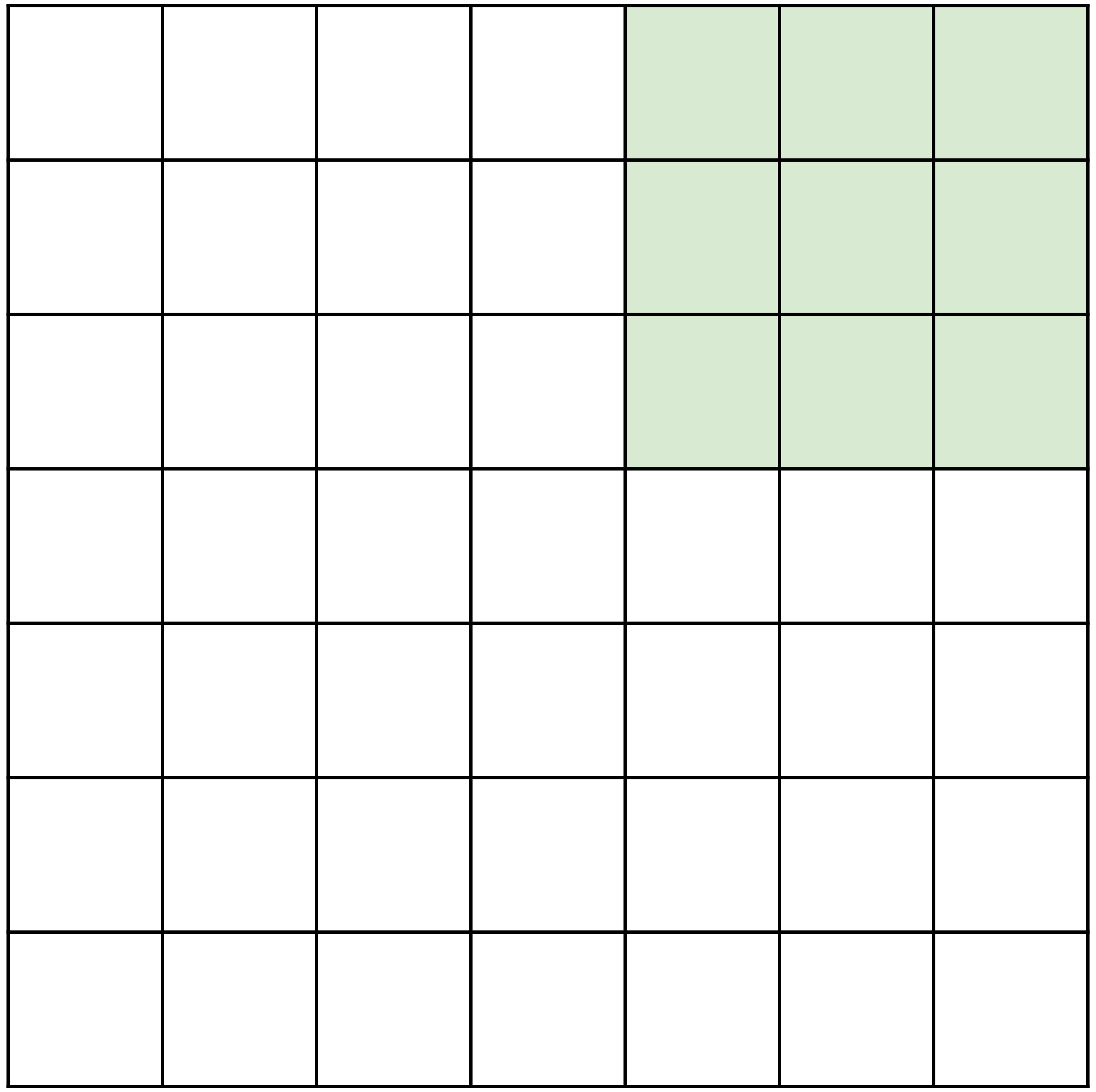
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

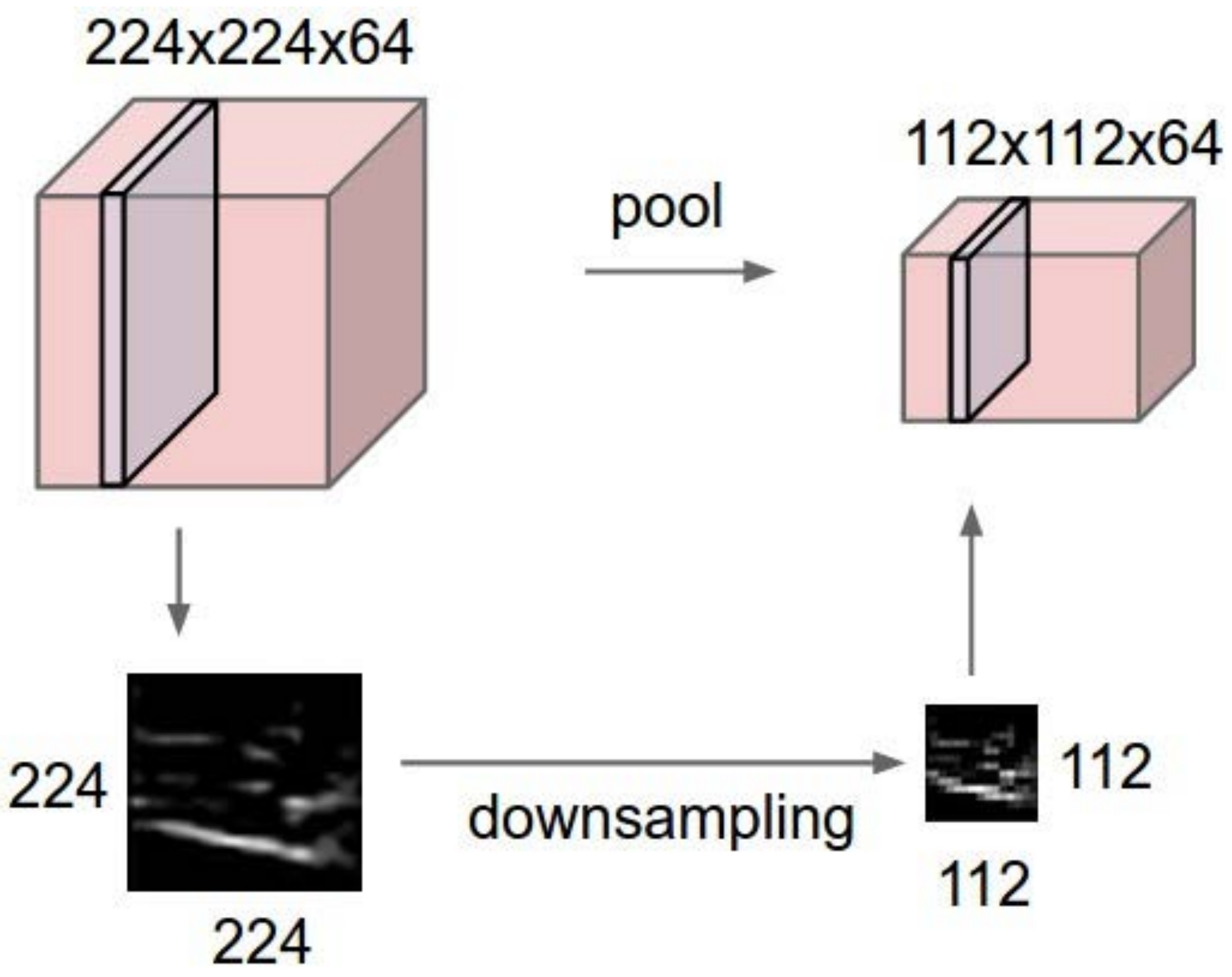
Filter: K

Padding: P

Stride: S

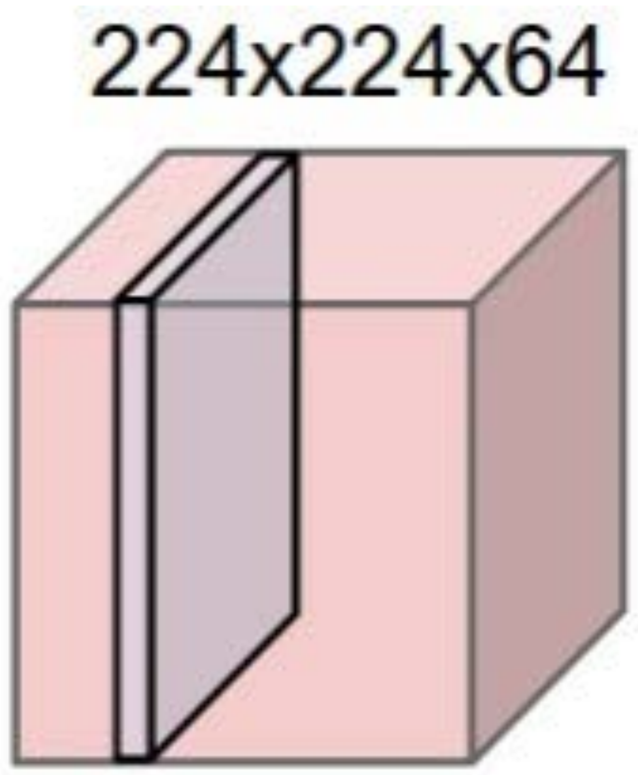
Output: (W – K + 2P) / S + 1

Pooling Layers: Another way to downsample



Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling



Single depth slice

x

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

y

Max pooling with 2x2 kernel size and stride 2

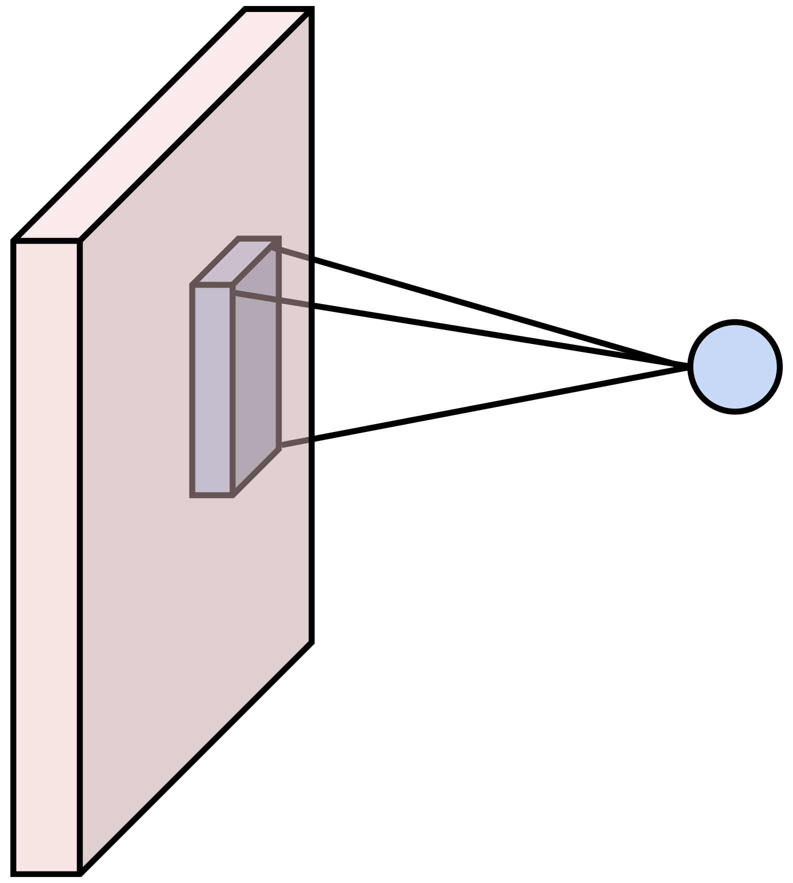
6	8
3	4

Introduces **invariance** to small spatial shifts

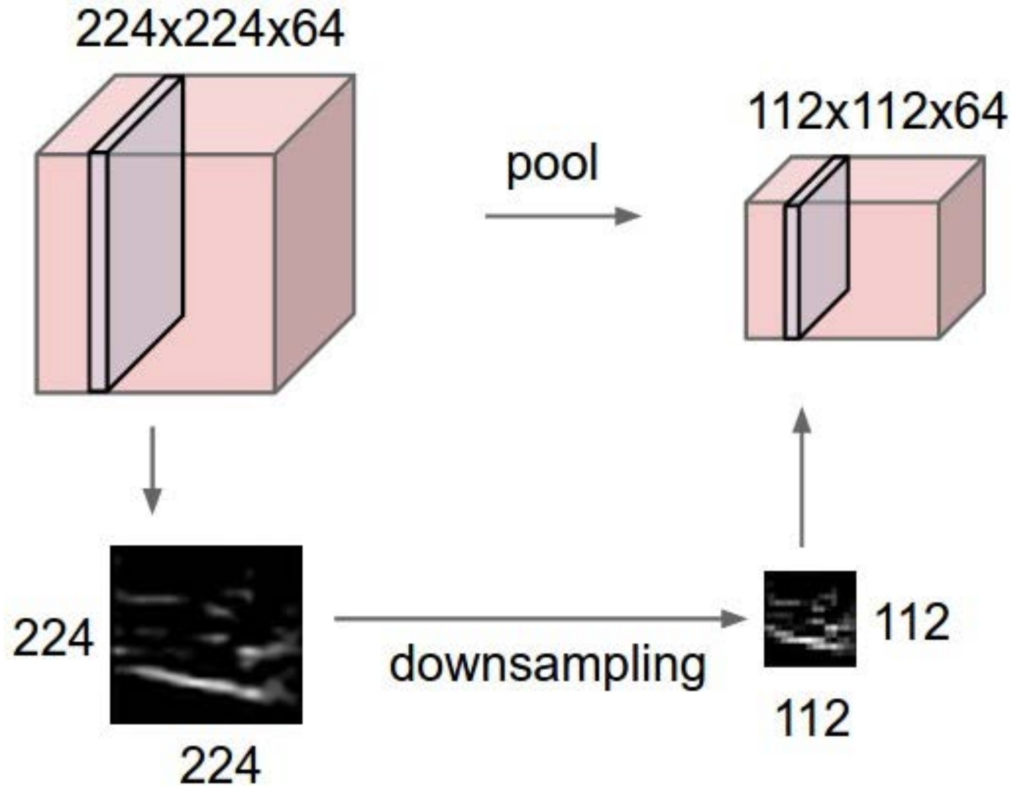
No learnable parameters!

Components of a Convolutional Network

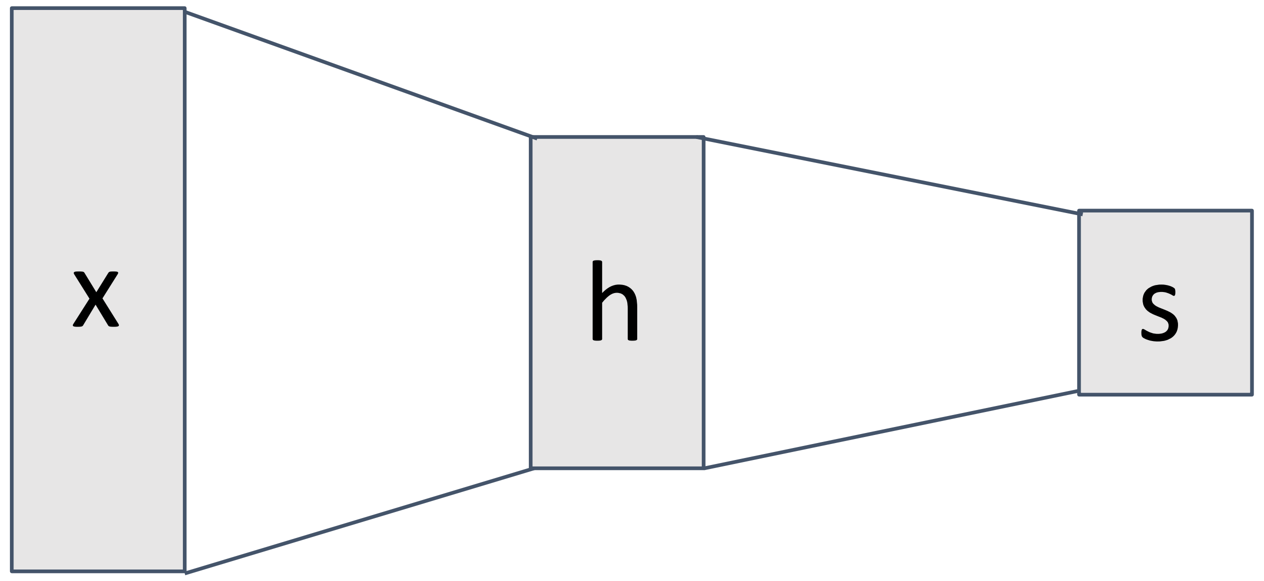
Convolution Layers



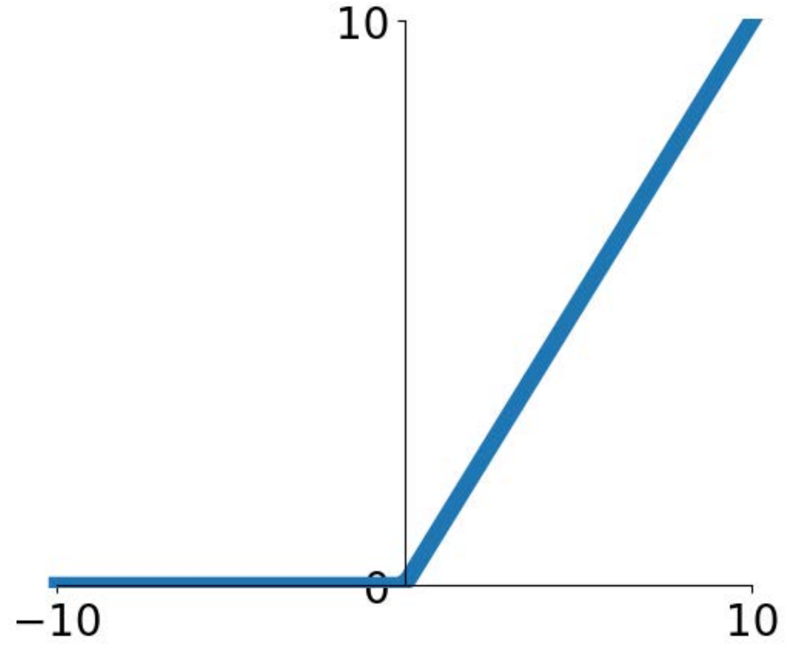
Pooling Layers



Fully-Connected Layers



Activation Function



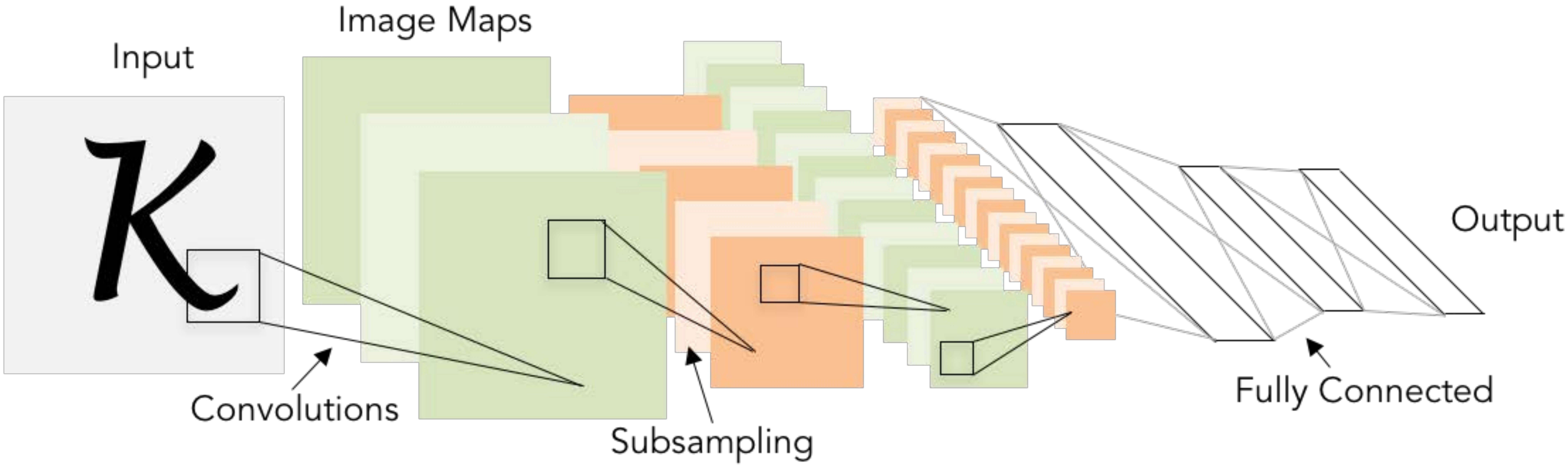
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Convolutional Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

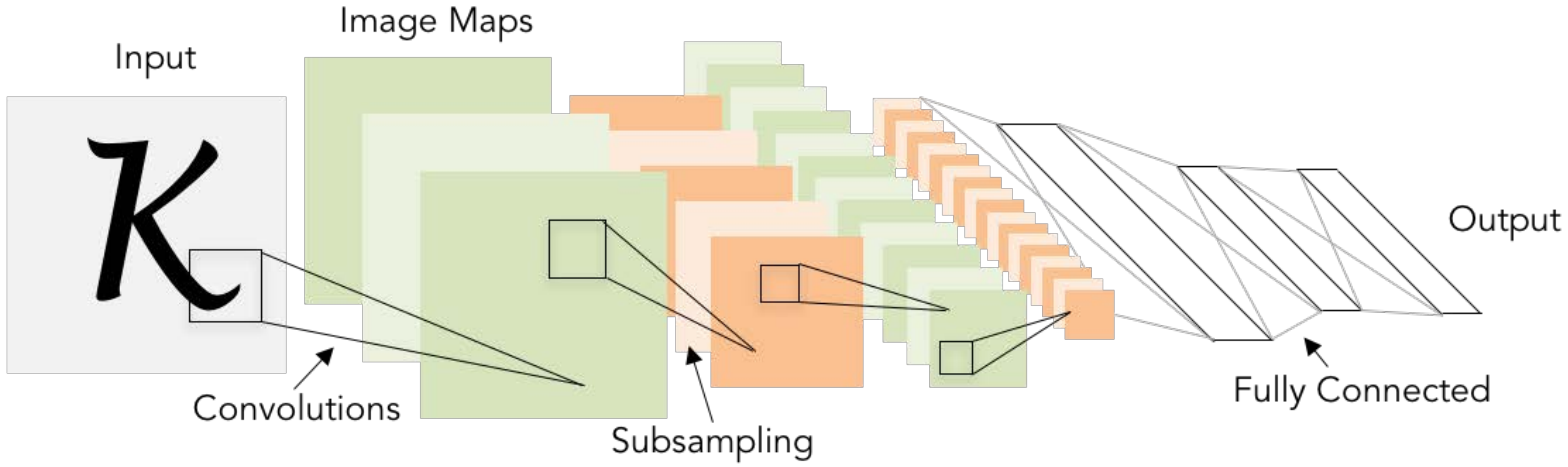
Example: LeNet-5



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10



As we go through the network:

Spatial size **decreases**
(using pooling or strided conv)

Number of channels **increases**
(total “volume” is preserved!)

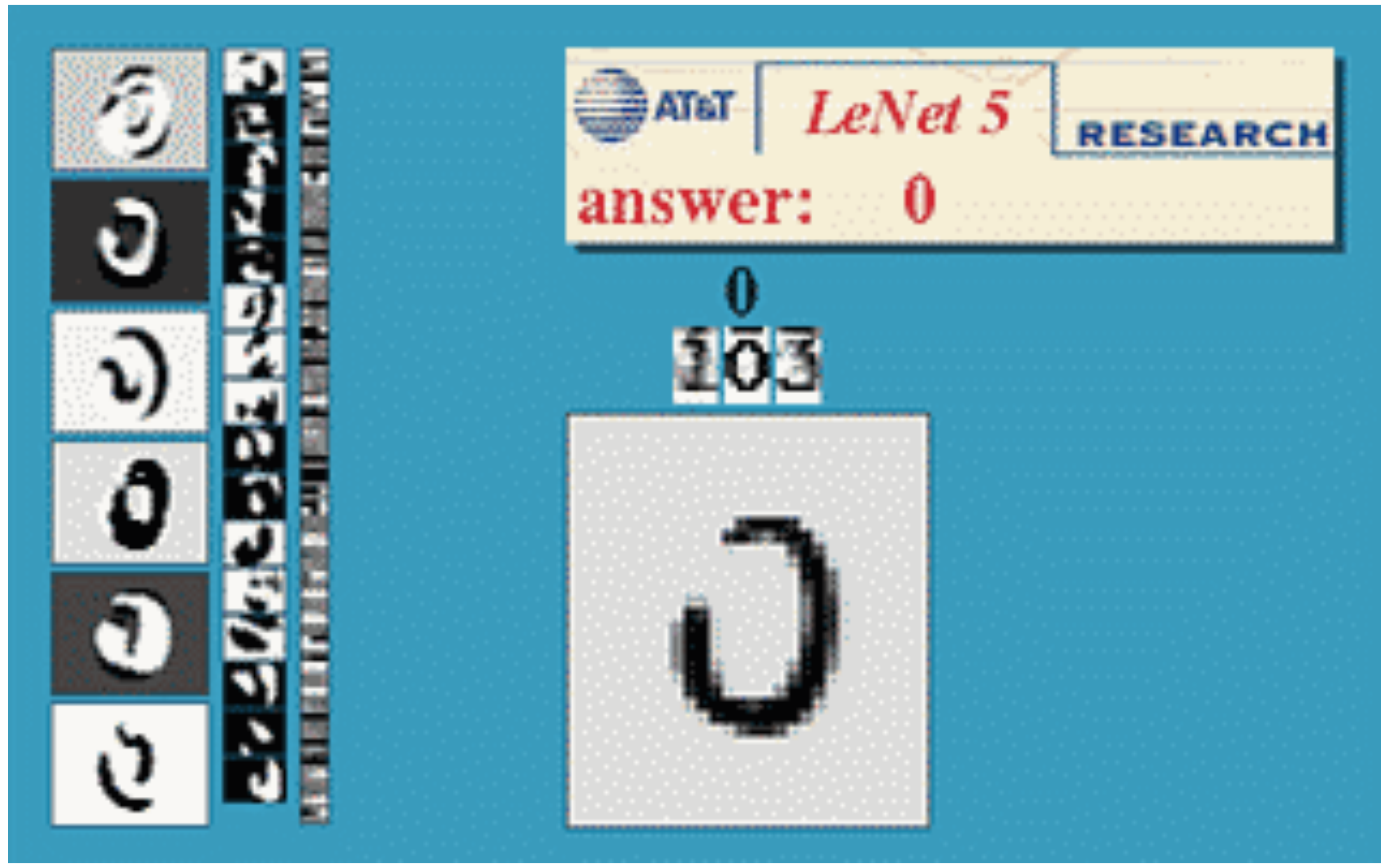
Lecun et al, “Gradient-based learning applied to document recognition”, 1998

Optical Character Recognition (**OCR**)

Technology to convert **scanned documents to text**

(comes with any scanner now days)

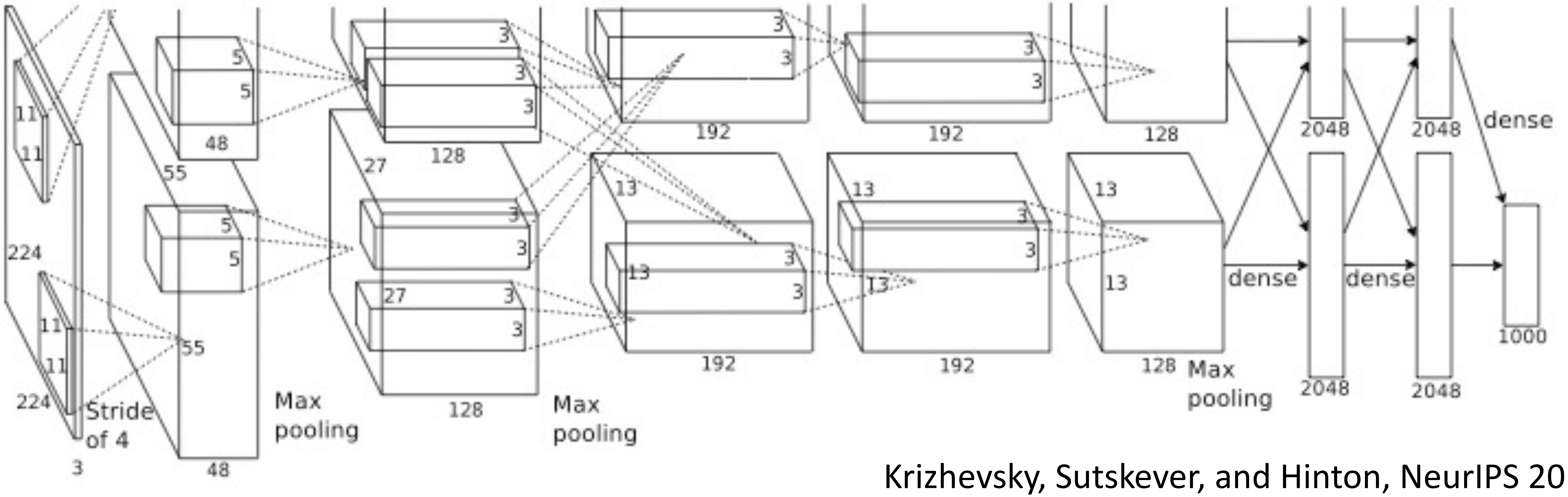
Yann LeCun



Digit recognition, AT&T labs
http://www.research.att.com/~yann/

License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

AlexNet: Deep Learning Goes Mainstream



Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

1959 Hubel & Wiesel

1963 Roberts

1970s David Marr

1979 Gen. Cylinders

1986 Canny

AI Winter

1997 Norm. Cuts

1999 SIFT

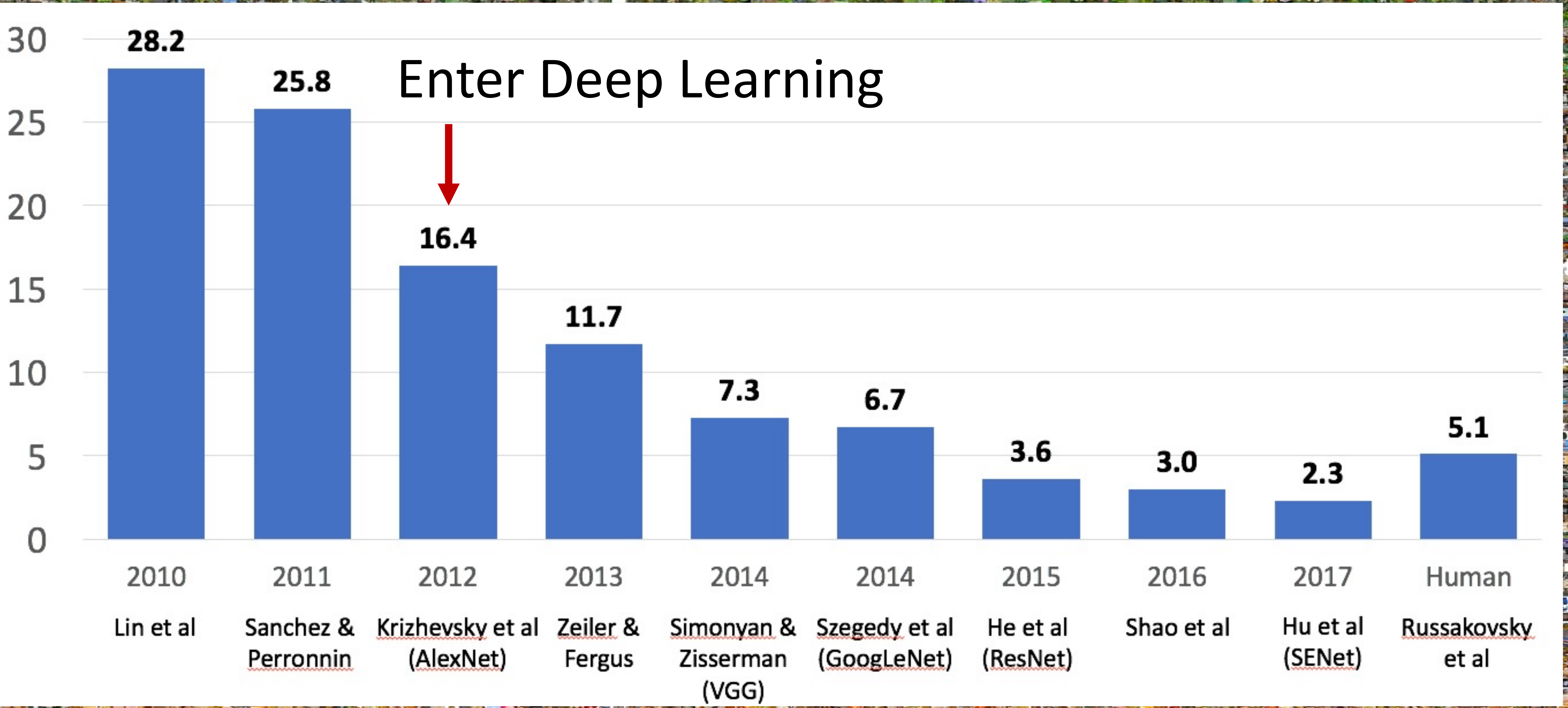
2001 V&J

2007 PASCAL

2009 ImageNet

2012 AlexNet

IMAGENET Large Scale Visual Recognition Challenge



1959 Hubel & Wiesel

1963 Roberts

1970s David Marr

1979 Gen. Cylinders

1986 Canny

AI Winter

1997 Norm. Cuts

1999 SIFT

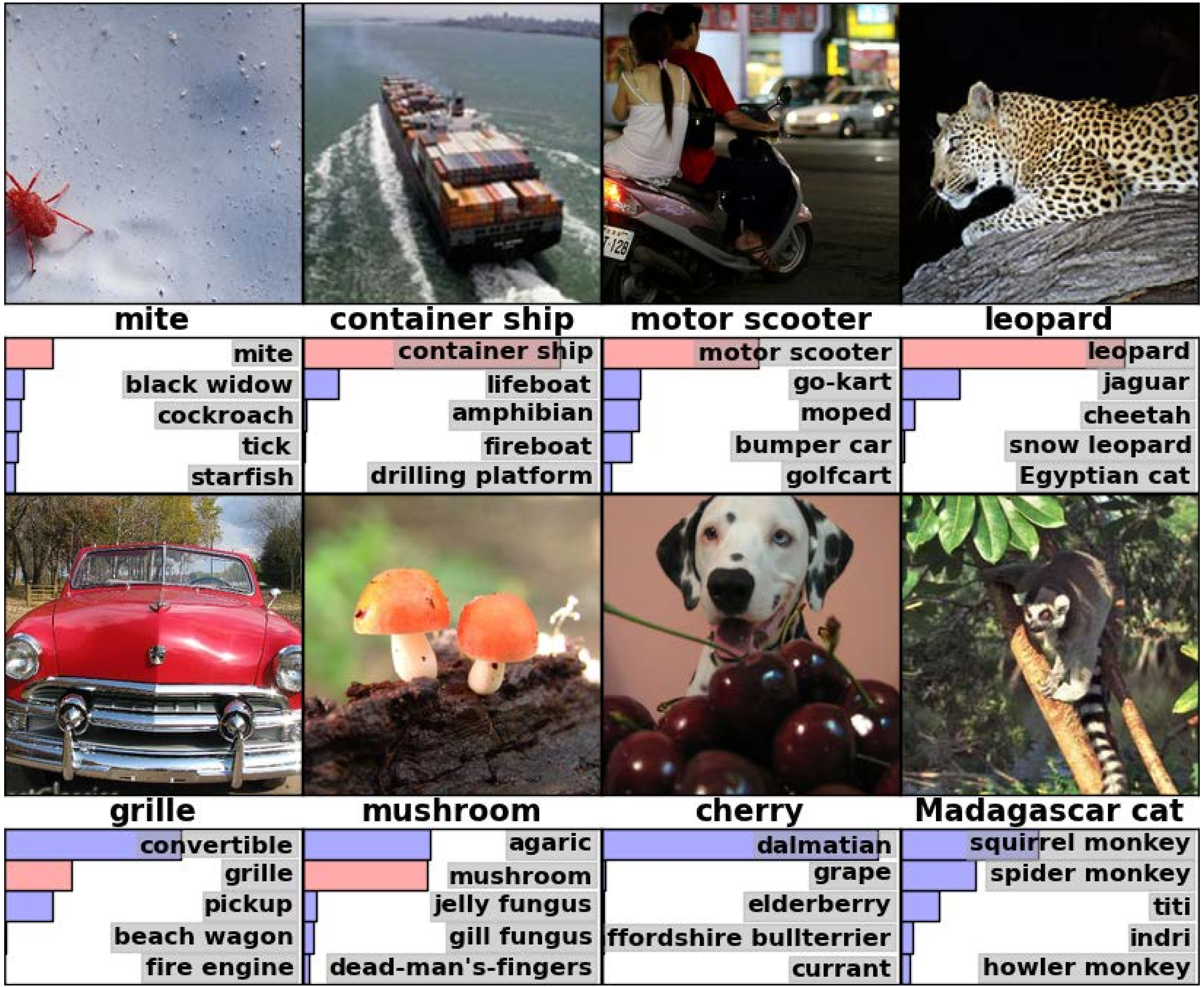
2001 V&J

2007 PASCAL

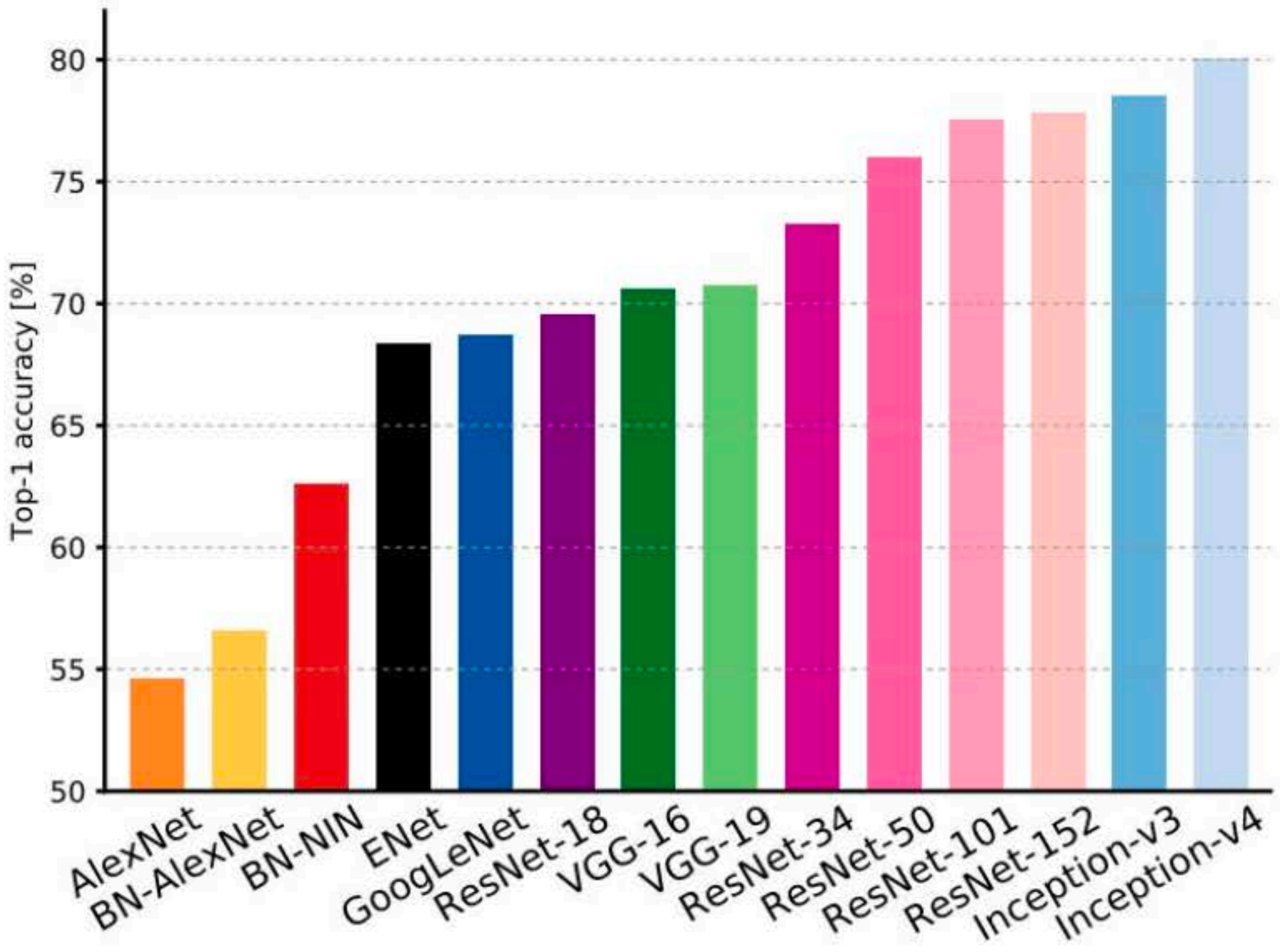
2009 ImageNet

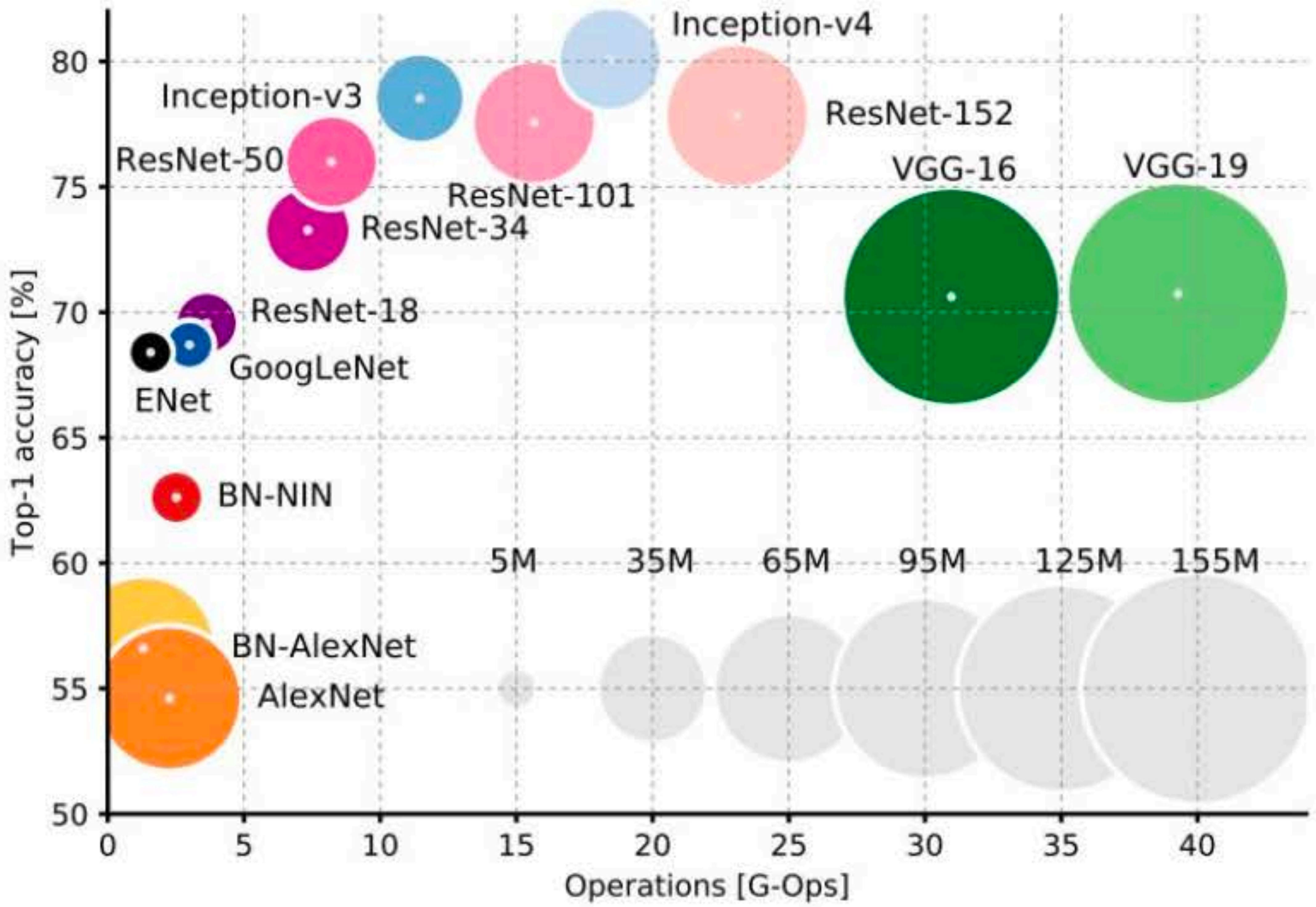
2012 AlexNet

AlexNet on ImageNet



Comparing **Complexity**





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Summary

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the chain rule

A **convolutional neural network** assumes inputs are images, and constrains the network architecture to reduce the number of parameters

A **convolutional layer** applies a set of learnable filters

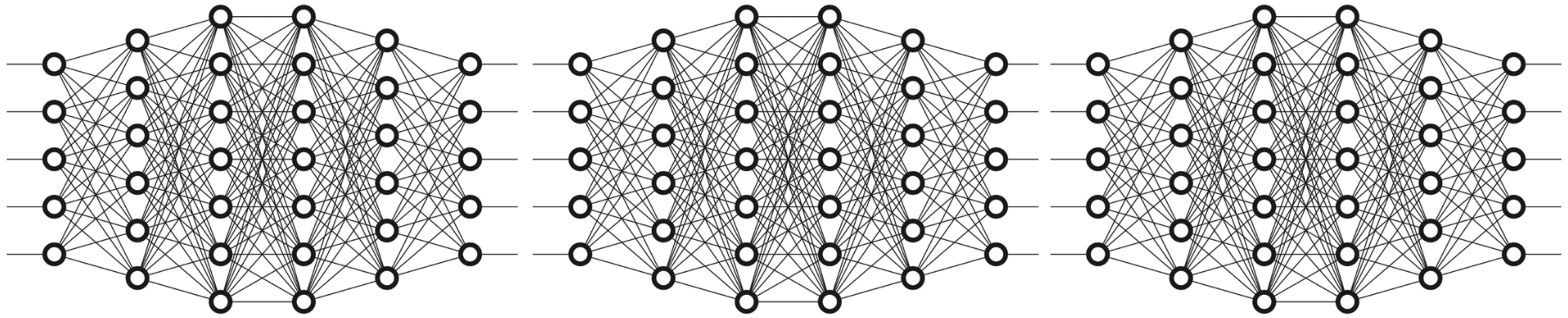
A **pooling layer** performs spatial downsampling

A **fully-connected** layer is the same as in a regular neural network

Convolutional neural networks can be seen as learning a hierarchy of filters

THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 22: Neural Networks 3

Menu for Today

Topics:

- **Neural Networks** part 3
- Weight **Initialization**
- **Normalization**
- Preventing **Overfitting**

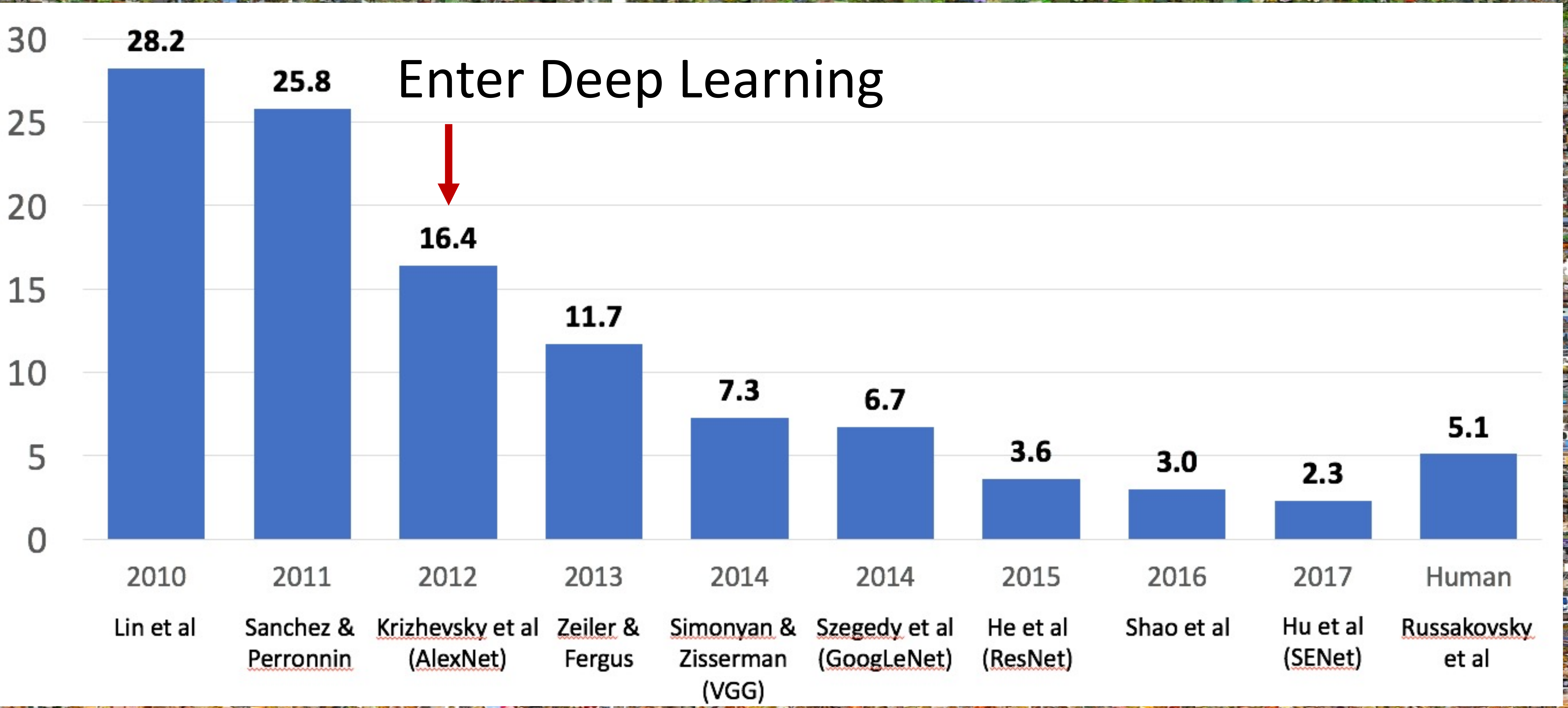
Readings:

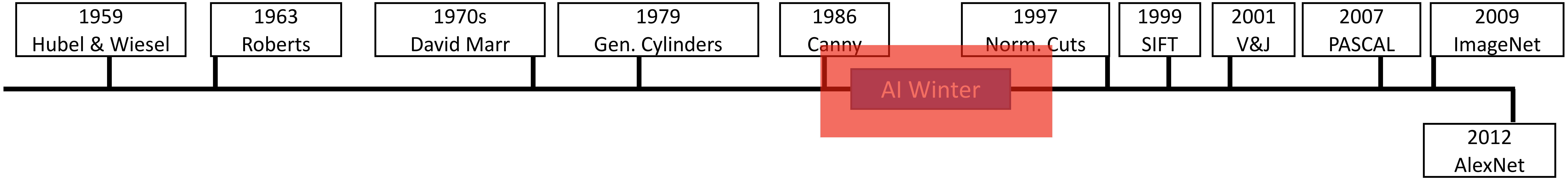
- **Today's** Lecture: Szeliski 5.1.3, 5.3-5.4, Justin Johnson Michigan EECS 498/598

Reminders:

- **Quiz 6**: Open Apr 10th, due **Apr 11th**!
- **Assignment 6**: Deep Learning due **April 12th!**
- **Final**: April 16th

IMAGENET Large Scale Visual Recognition Challenge





So why now?

Rise of large datasets

IMAGENET

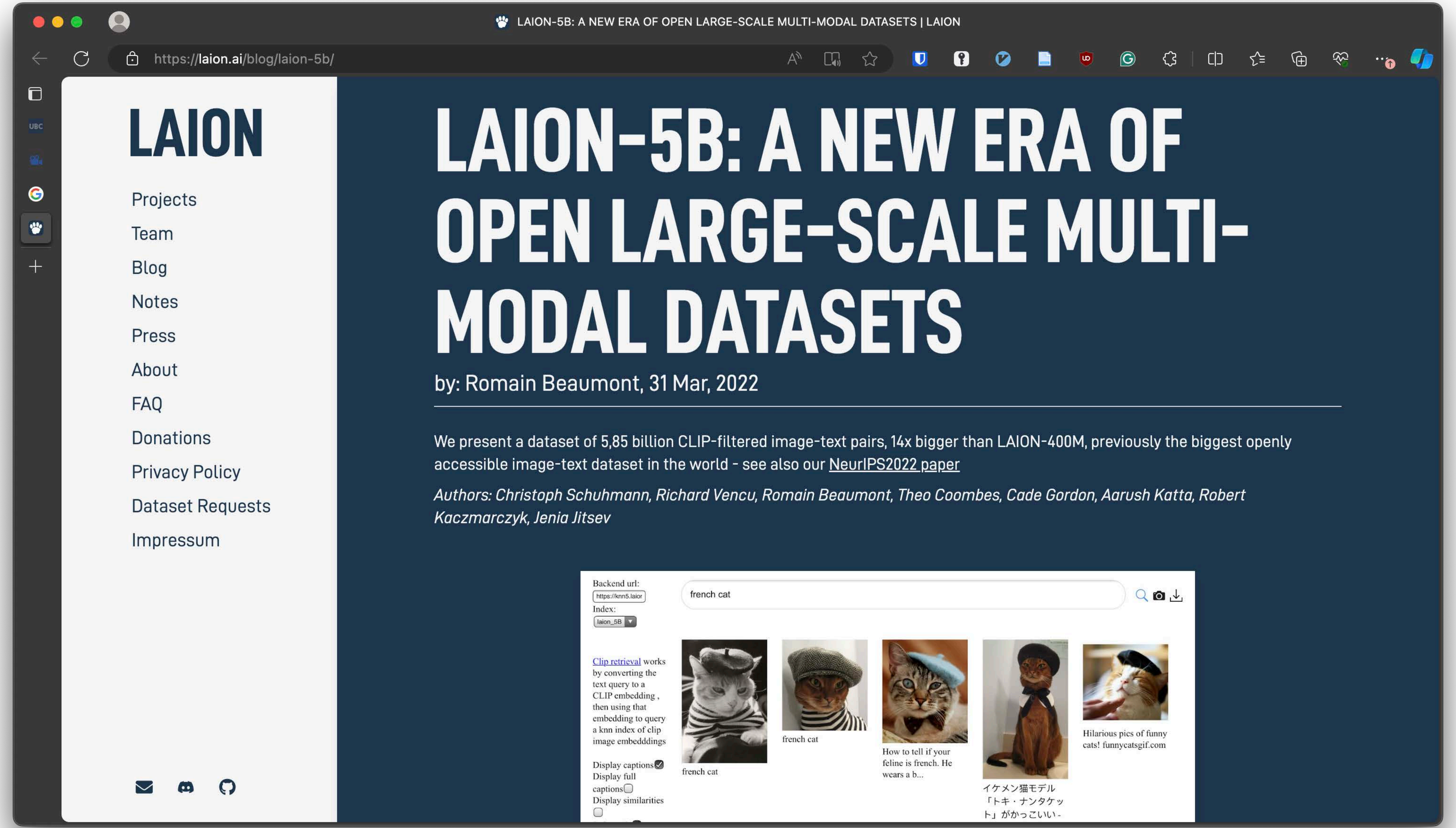
www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

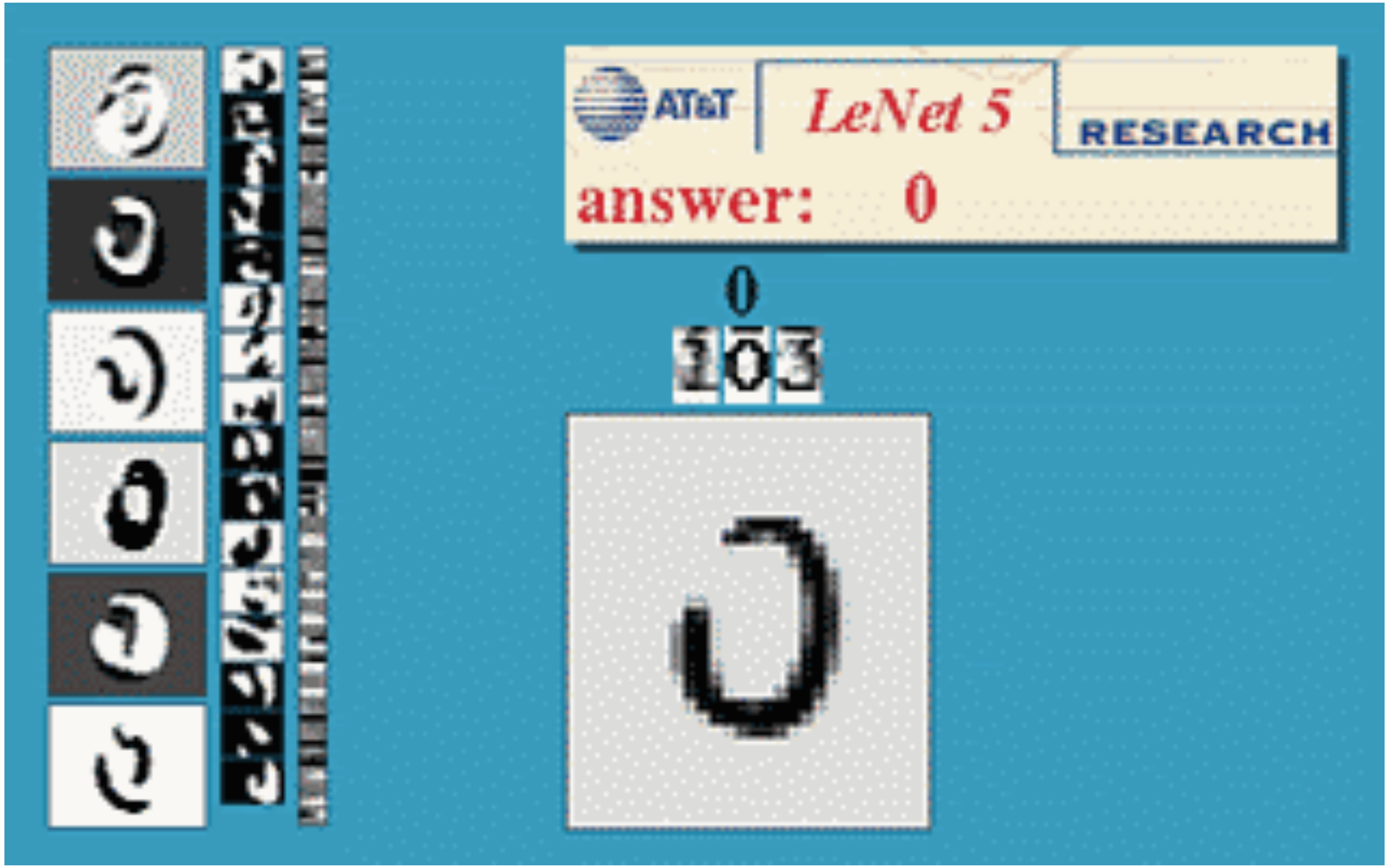
Rise of large datasets



https://laion.ai/blog/laion-5b/

Clever architectures

Convolutional neural networks

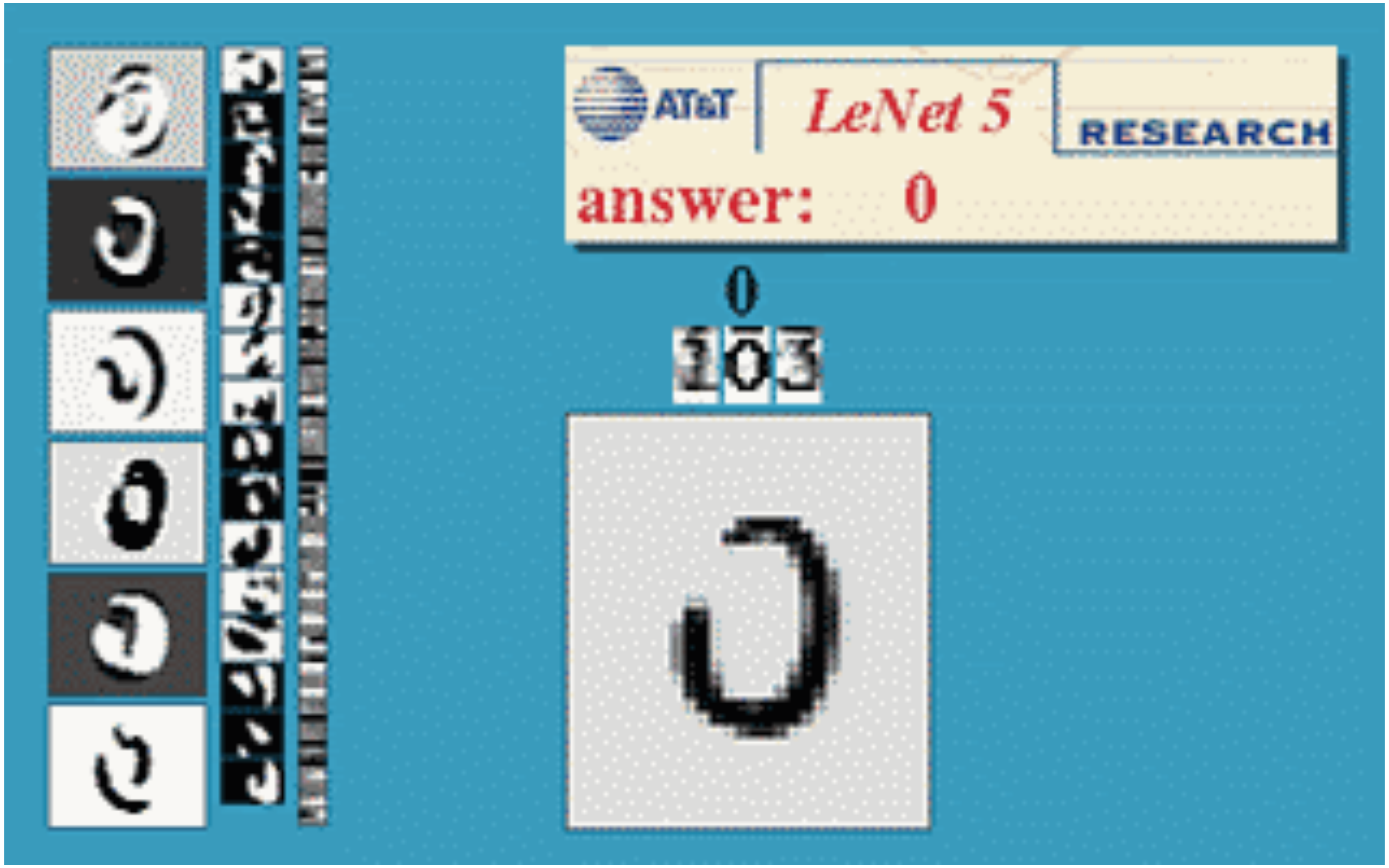


[Lecun, Bottou, Bengio, and Haffner, "Gradient-Based Learning Applied to Document Recognition", 1998]

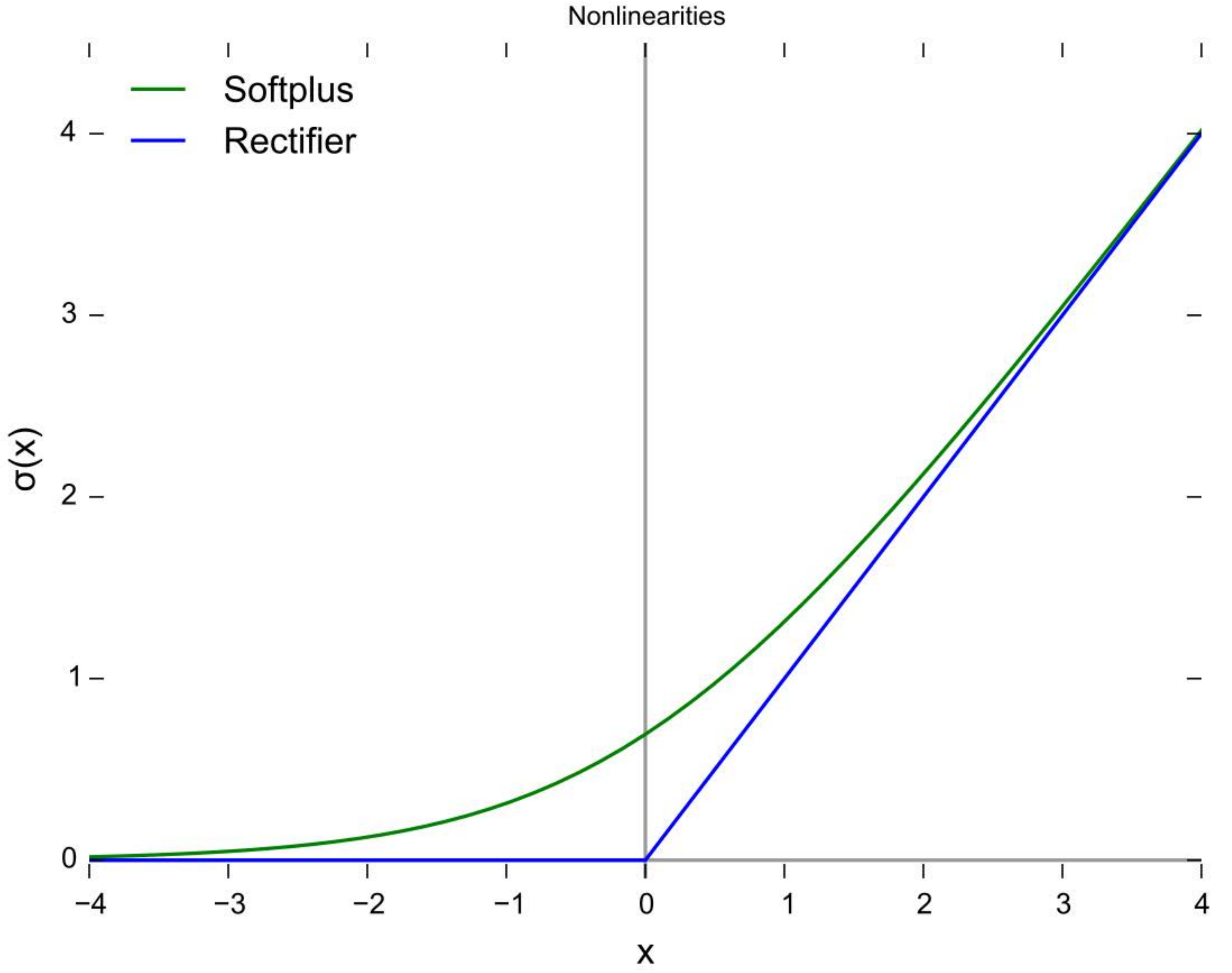
THE UNIVERSITY OF BRITISH COLUMBIA

Clever architectures

Convolutional neural networks



[Lecun, Bottou, Bengio, and Haffner, “Gradient-Based Learning Applied to Document Recognition”, 1998]



[Nair and Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines”, 2010]

THE UNIVERSITY OF BRITISH COLUMBIA

Clever architectures

Transformers, Vaswani et al., 2017

Attention is all you need

Authors	Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, Illia Polosukhin
Publication date	2017
Journal	Advances in neural information processing systems
Volume	30
Description	The dominant sequence transduction models are based on complex recurrent orconvolutional neural networks in an encoder and decoder configuration. The best performing such models also connect the encoder and decoder through an attentionm echanisms. We propose a novel, simple network architecture based solely onan attention mechanism, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superiorin quality while being more parallelizable and requiring significantly less timeto train. Our single model with 165 million parameters, achieves 27.5 BLEU onEnglish-to-German translation, improving over the existing best ensemble result by over 1 BLEU. On English-to-French translation, we outperform the previoussingle state-of-the-art with model by 0.7 BLEU, achieving a BLEU score of 41.1.
Total citations	Cited by 87925

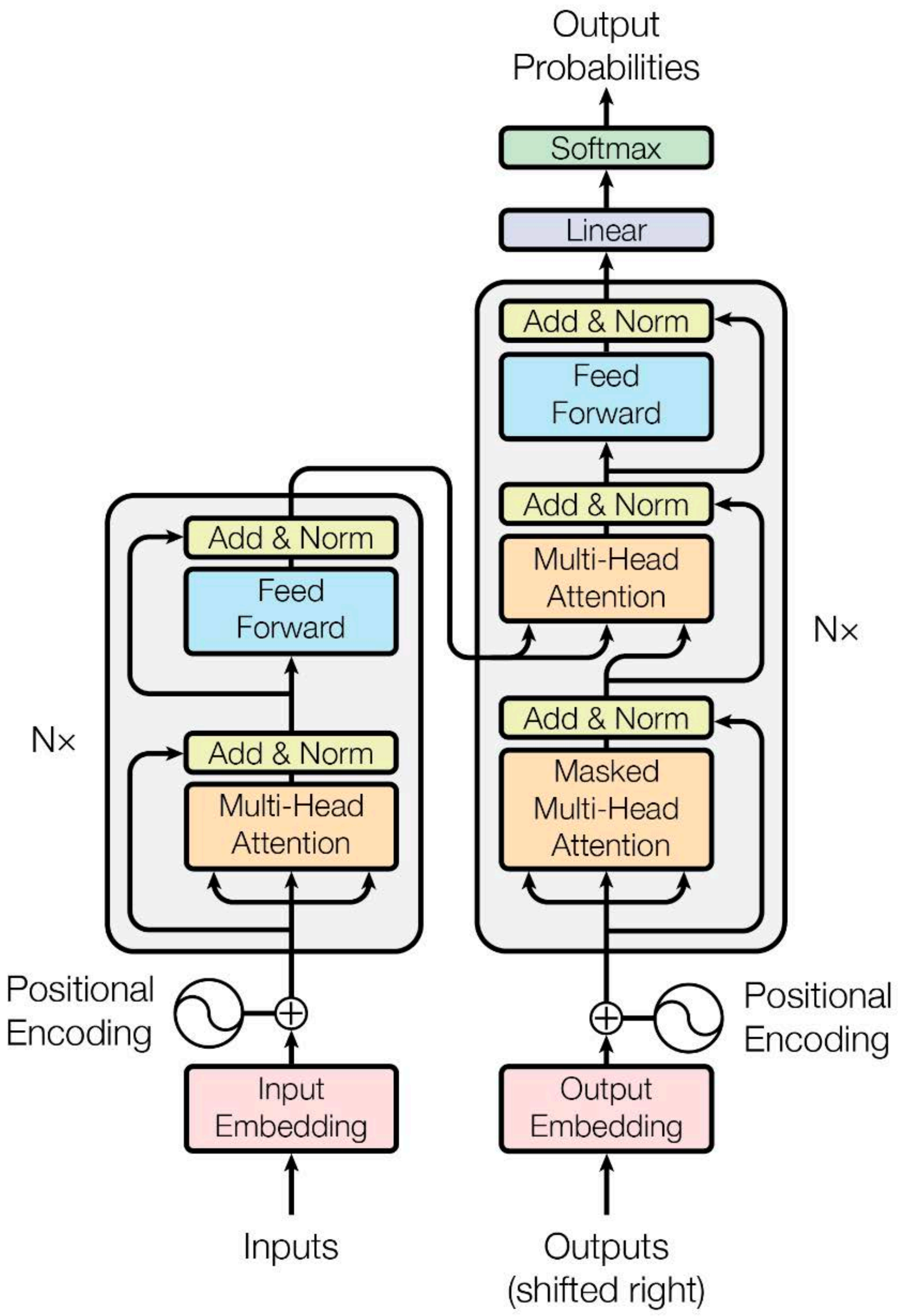


Figure 1: The Transformer - model architecture.

Proper initialization schemes

Much more important than you think

"Xavier initialization"
[Glorot et al., 2010]

Neuron activations are well spread

Less important now with "normalization"

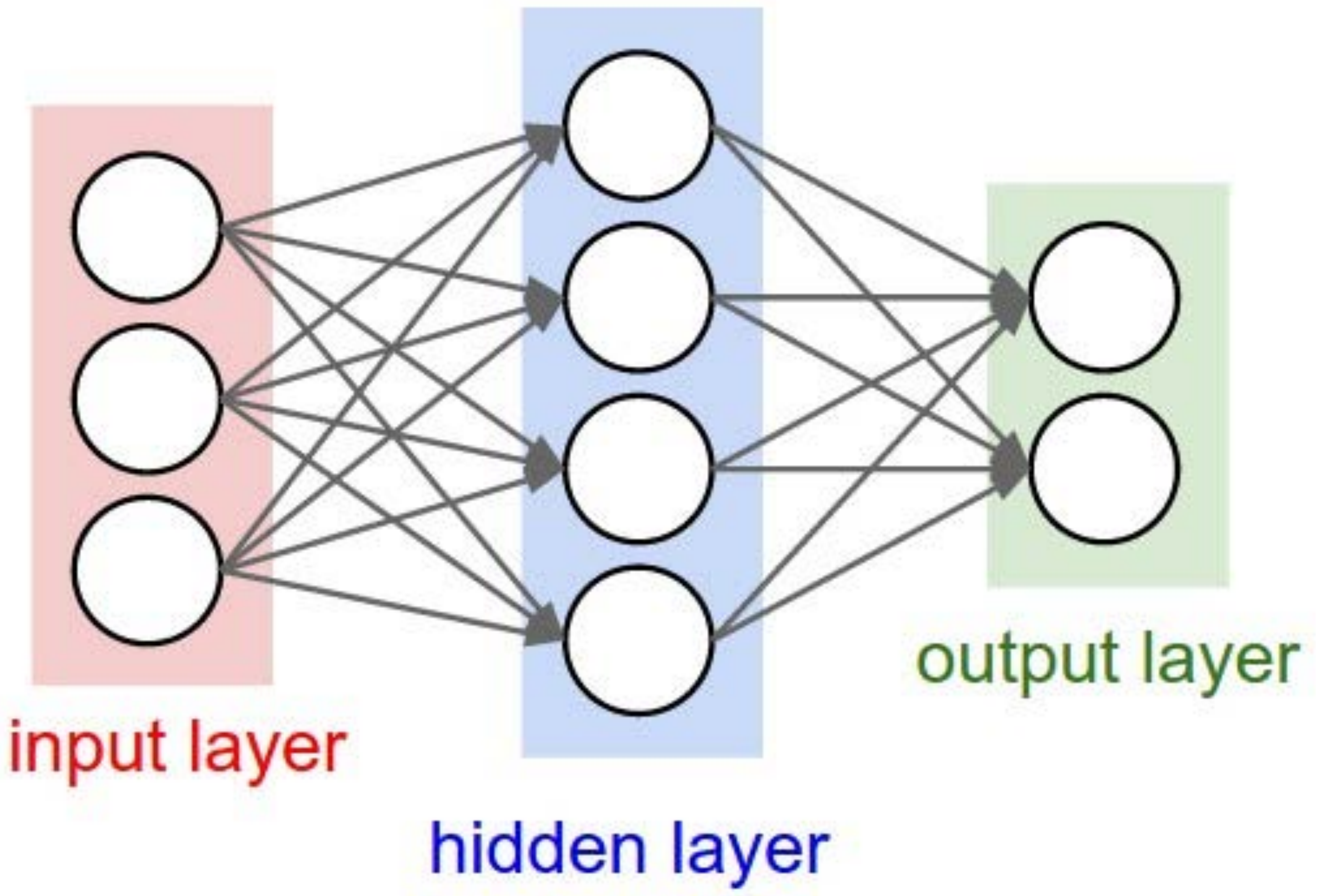
(But still can be VERY important in some cases)

Weight initialization

Weight initialization

A look into ways things can go wrong

Q: what happens when W=0 init is used?



Based on slides for [Stanford cs231n] by Li, Jonson, and Young. Modified and reused with permission

We

A loc

github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/lsgan/lsgan.py

```python
parser.add_argument("--img_size", type=int, default=32, help="size of each image dimension")
parser.add_argument("--channels", type=int, default=1, help="number of image channels")
parser.add_argument("--sample_interval", type=int, default=1000, help="number of image channels")
opt = parser.parse_args()
print(opt)

cuda = True if torch.cuda.is_available() else False


def weights_init_normal(m):
    classname = m.__class__.__name__
    if classname.find("Conv") != -1:
        torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find("BatchNorm") != -1:
        torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
        torch.nn.init.constant_(m.bias.data, 0.0)


class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.init_size = opt.img_size // 4
        self.l1 = nn.Sequential(nn.Linear(opt.latent_dim, 128 * self.init_size ** 2))
```

Weight initialization

A look into ways things can go wrong

Let's look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

Init with small random numbers

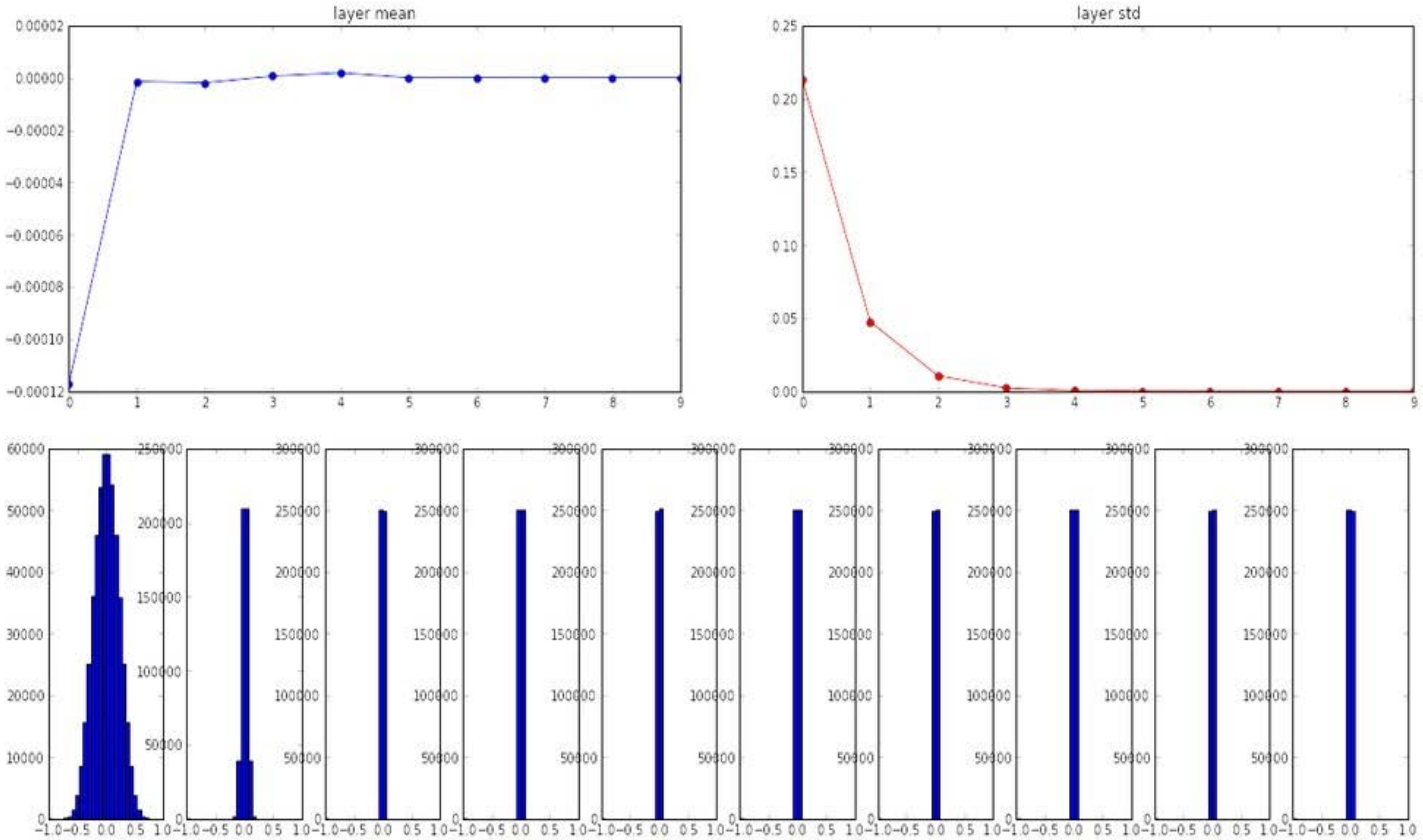
```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

Weight initialization

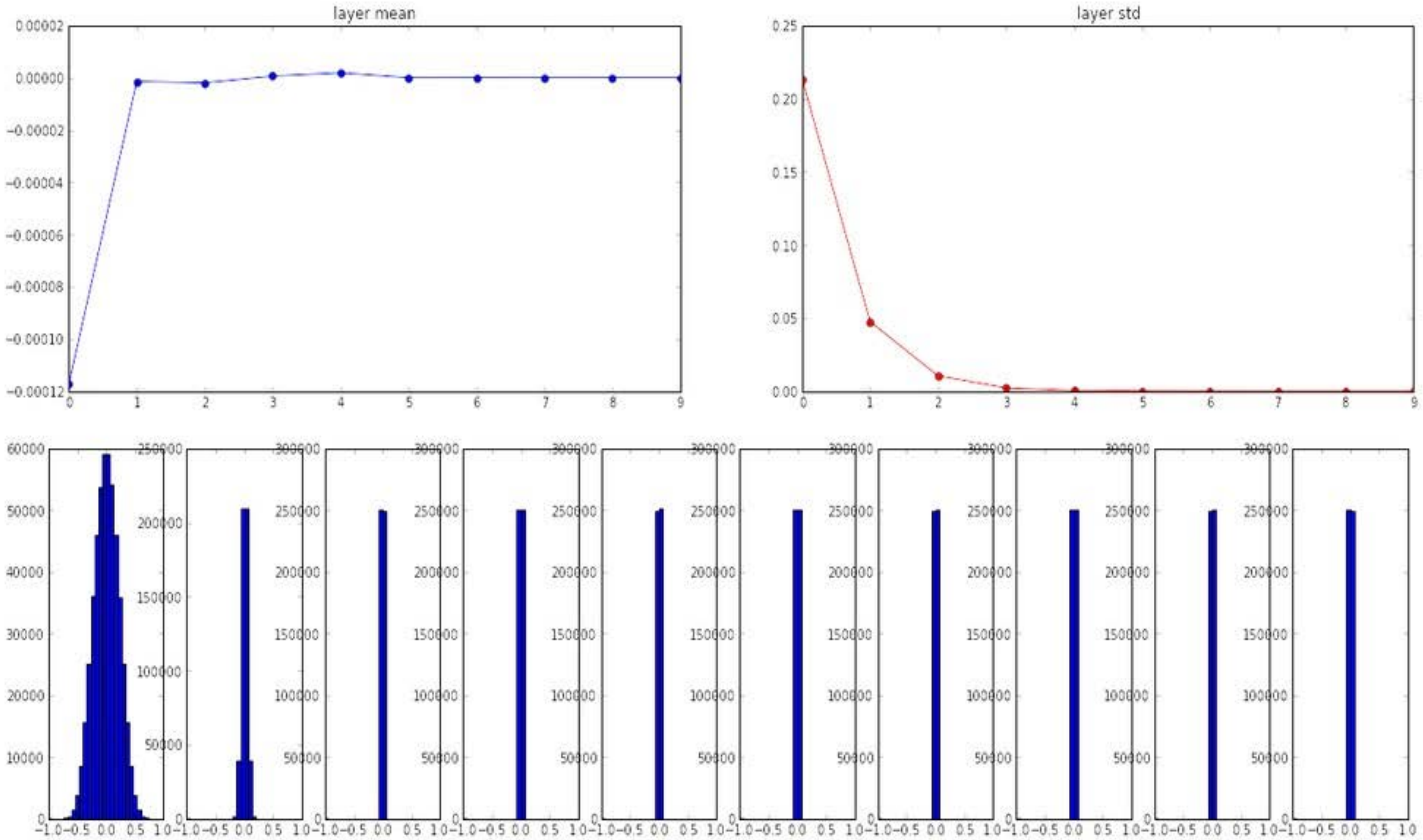
A look into ways things can go wrong



Based on slides for [Stanford cs231n](http://cs231n.stanford.edu/) by Li, Jonson, and Young. Modified and reused with permission

Weight initialization

A look into ways things can go wrong



All activations become zero!

Q: think about the backward pass. What do the gradients look like?

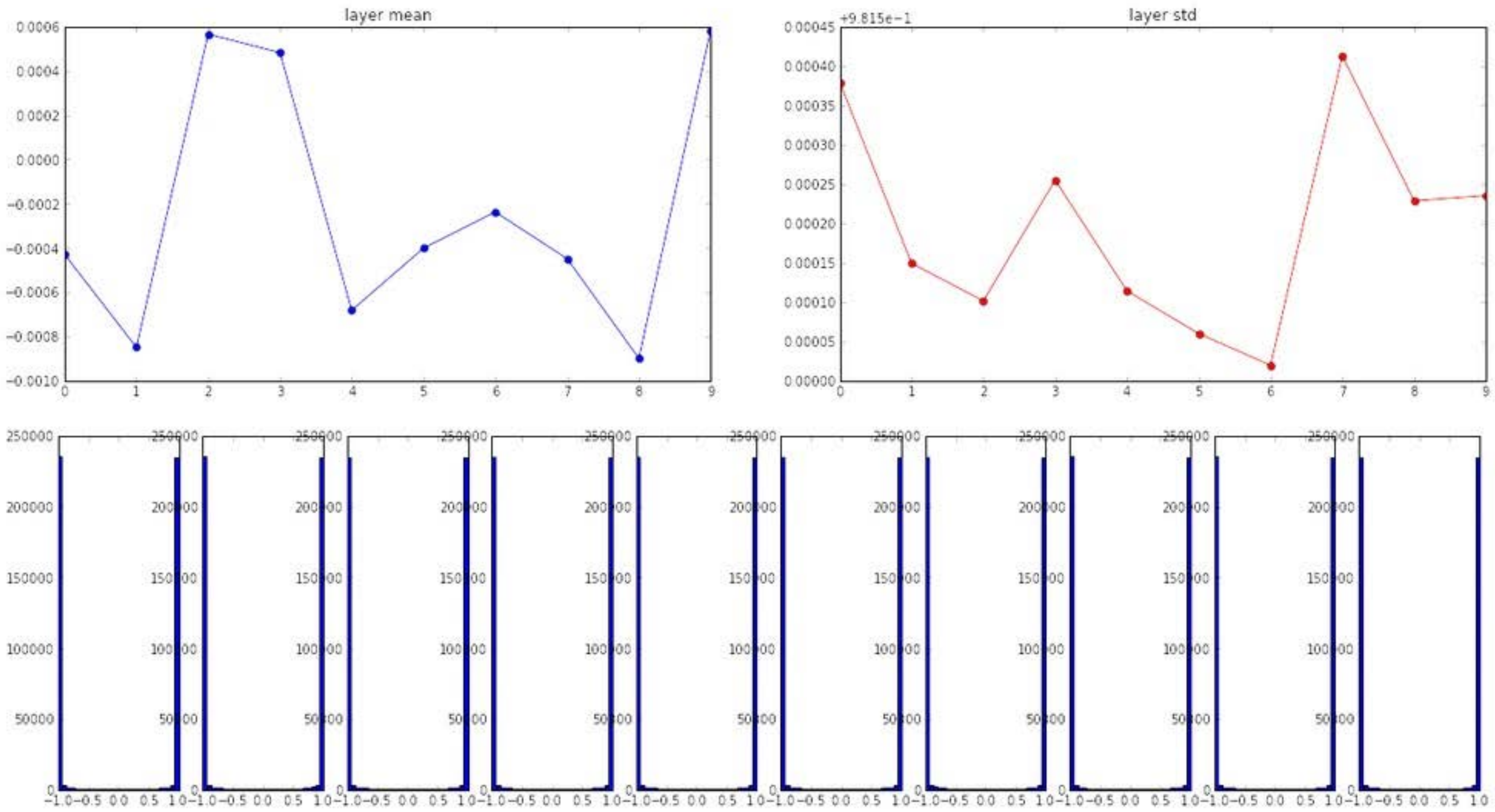
Hint: think about backward pass for a **W*X** gate.

Weight initialization

A look into ways things can go wrong

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

*1.0 instead of *0.01



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

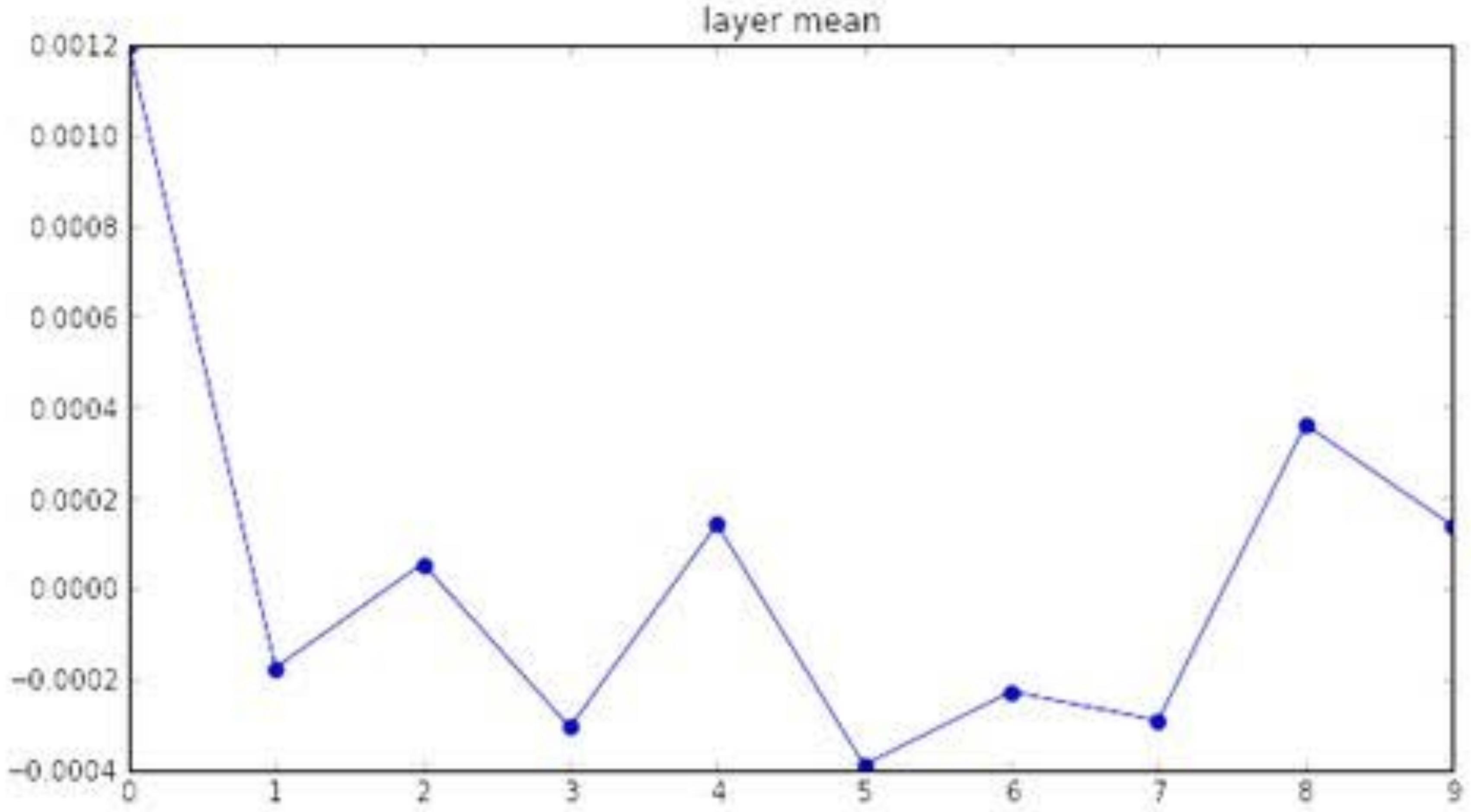
Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

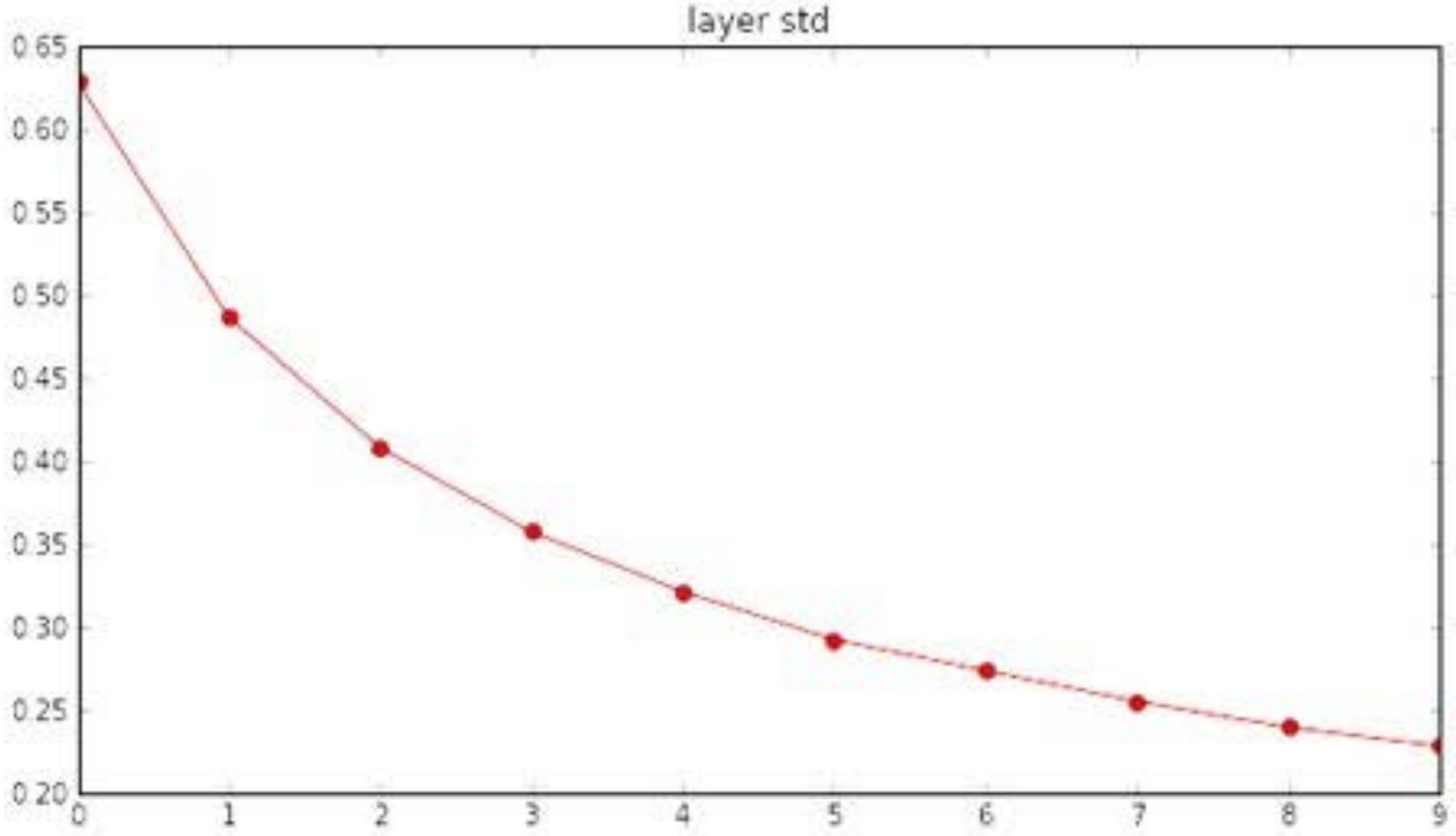
Weight initialization

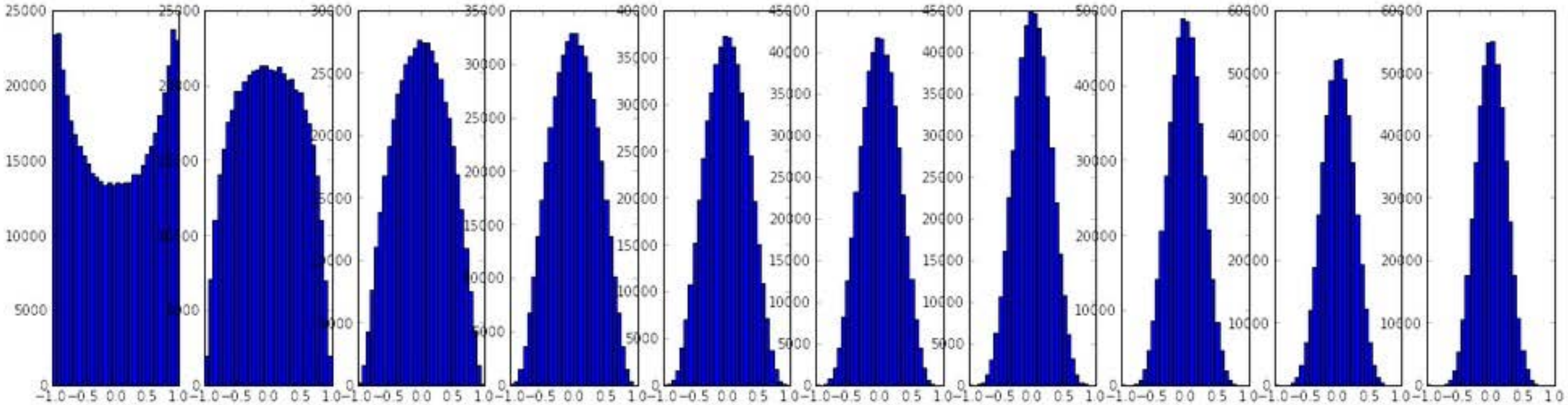
Glorot initialization [Glorot et al., 2010]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

Some number according to *"fan in"*







Statistically motivated

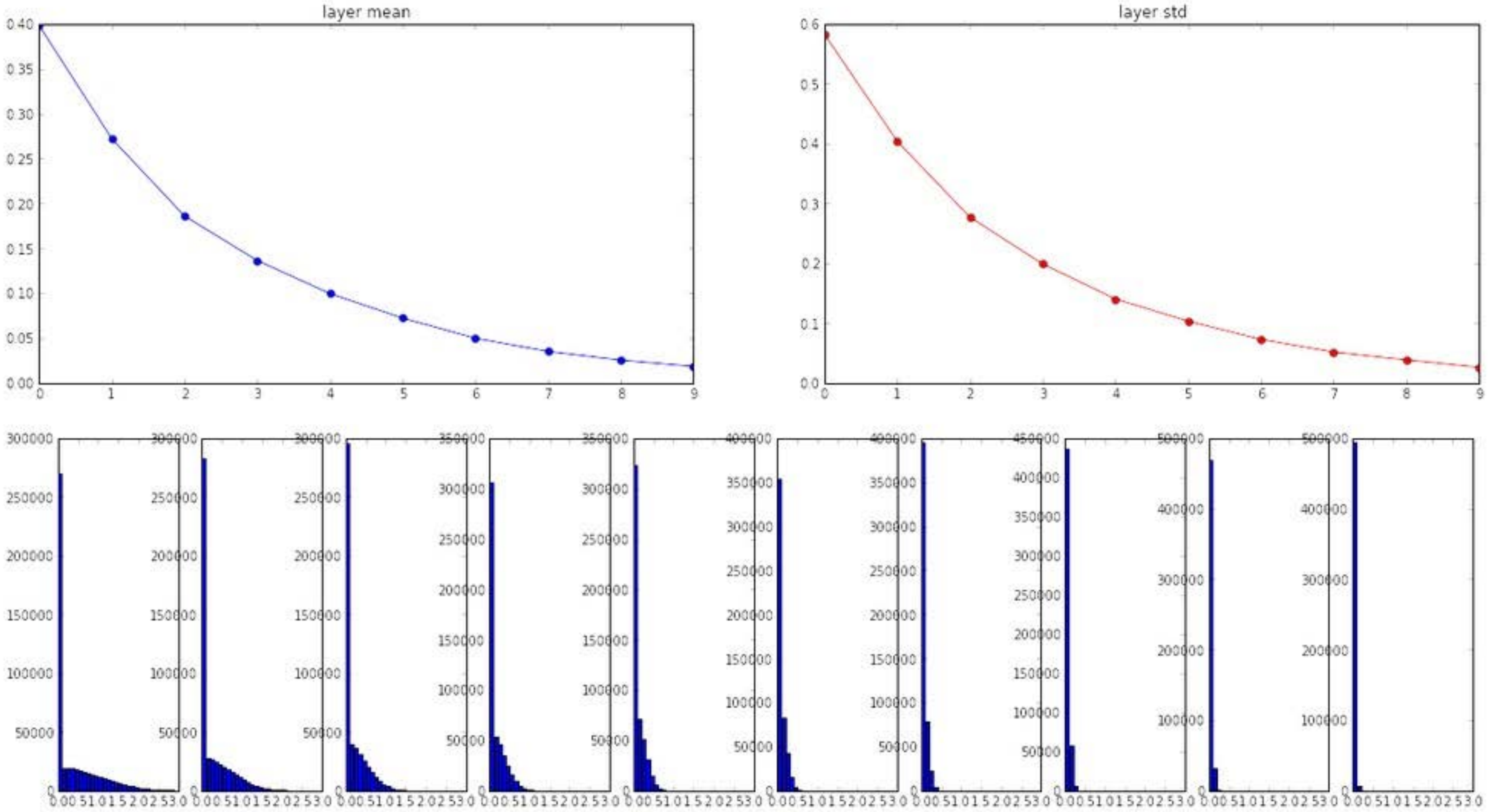
Good for tanh

Weight initialization

Glorot initialization [Glorot et al., 2010]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

Some number according to "*fan in*"



Statistically motivated

Good for tanh

Not so good for ReLU

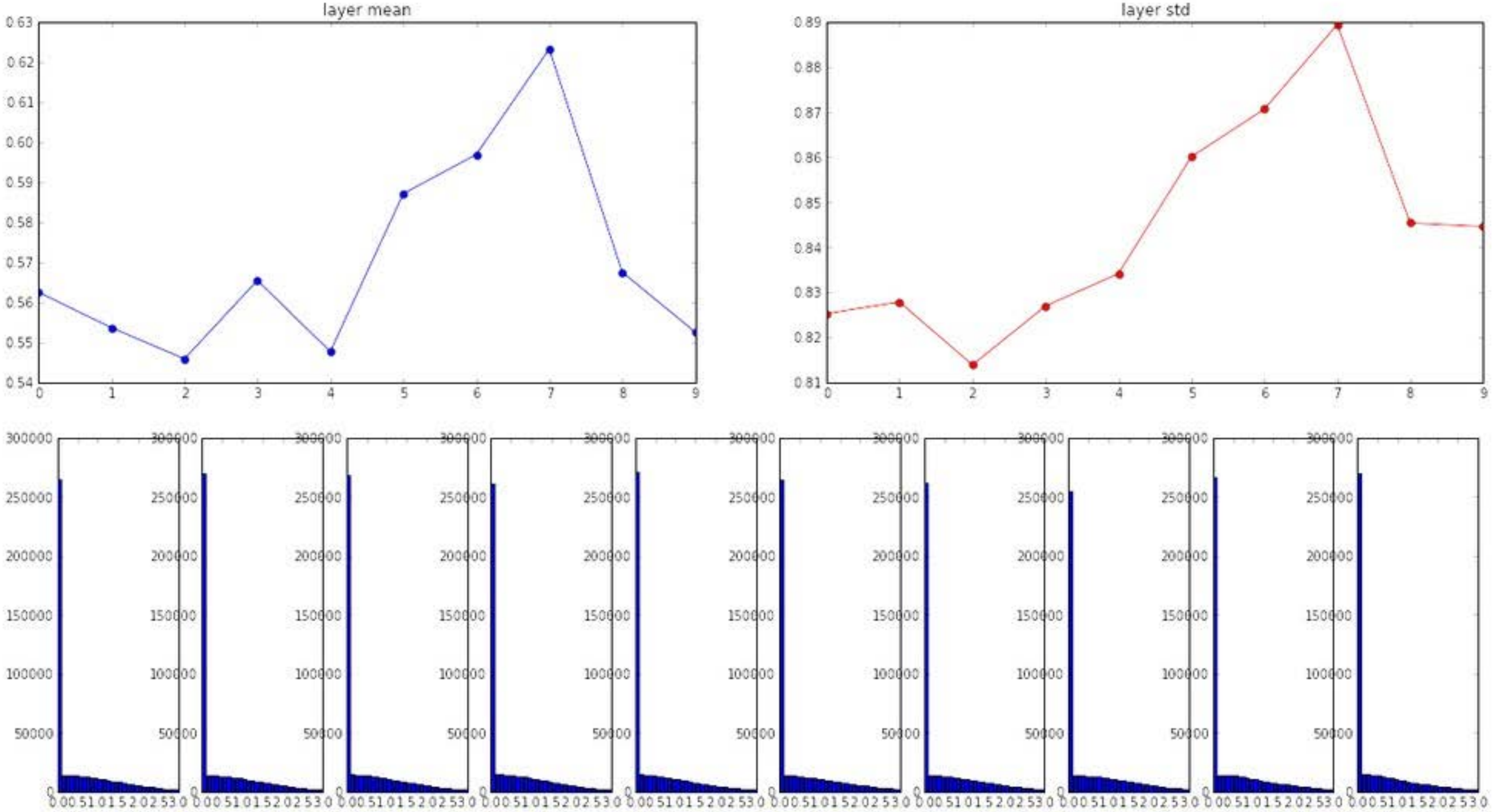
Based on slides for [Stanford cs231n] by Li, Jonson, and Young. Modified and reused with permission

Weight initialization

He initialization [He et al., 2015]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

magic number 2



Statistically motivated

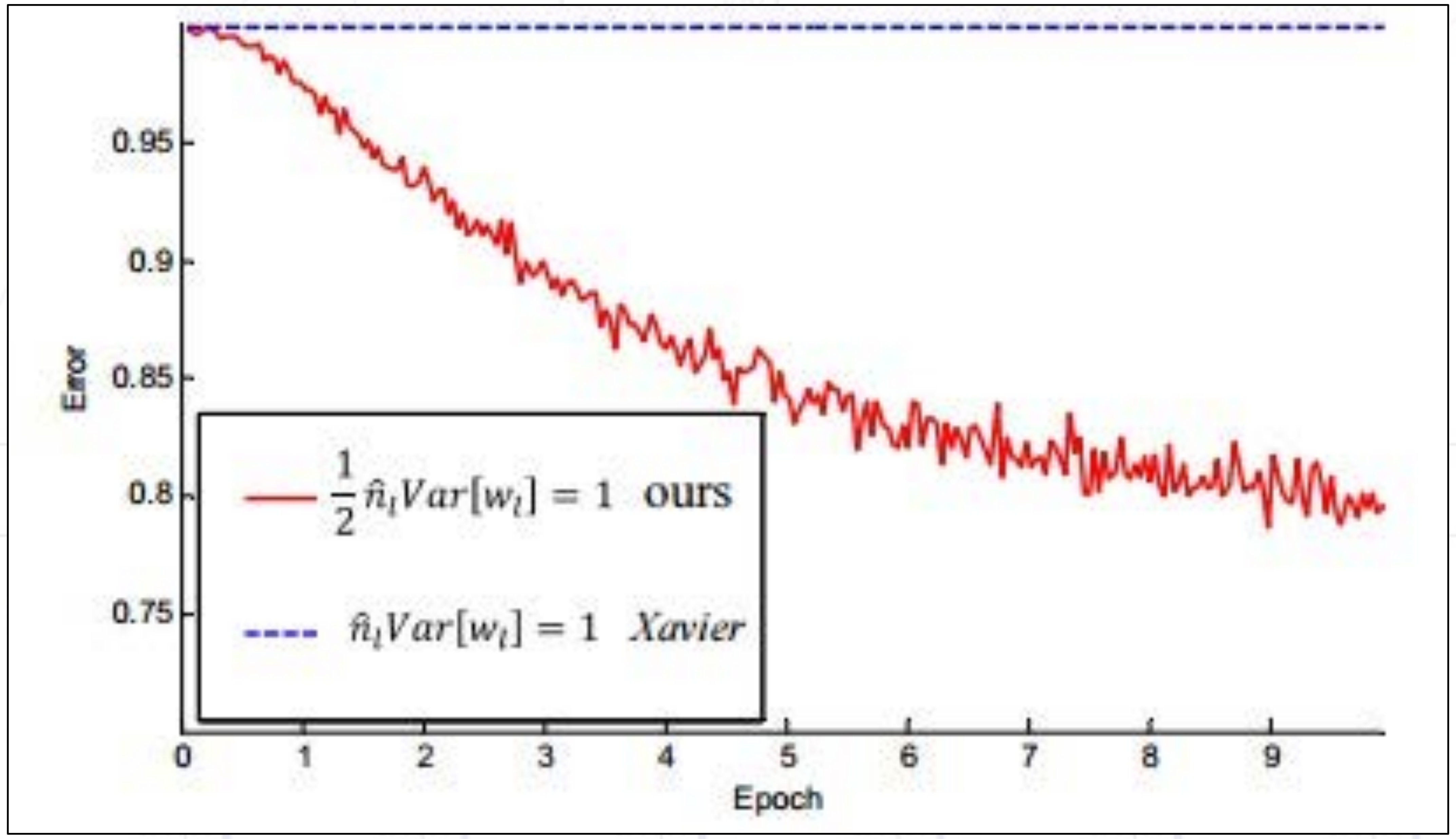
Good for ReLU

Weight initialization

He initialization [He et al., 2015]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

magic number 2

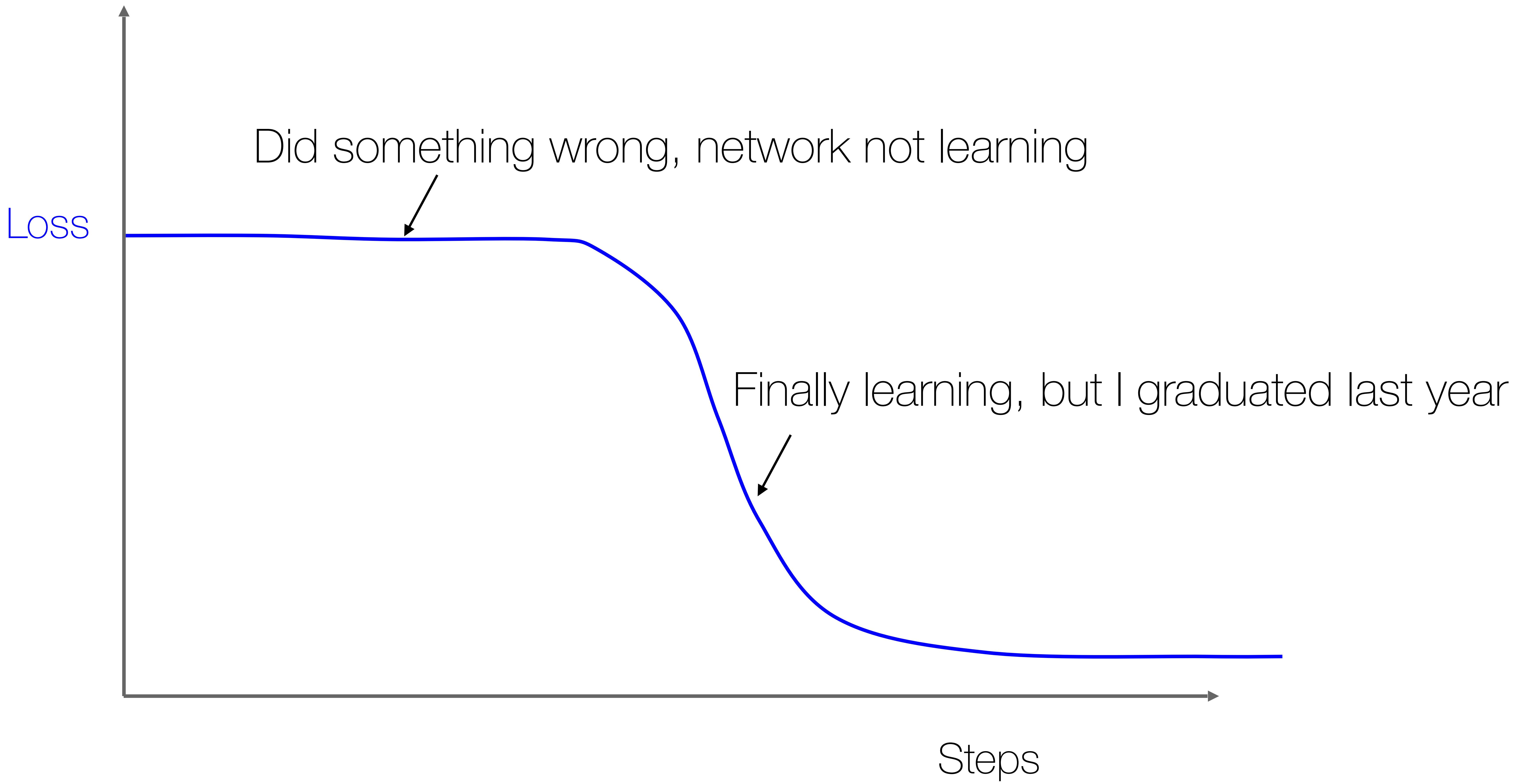


Statistically motivated

Good for ReLU

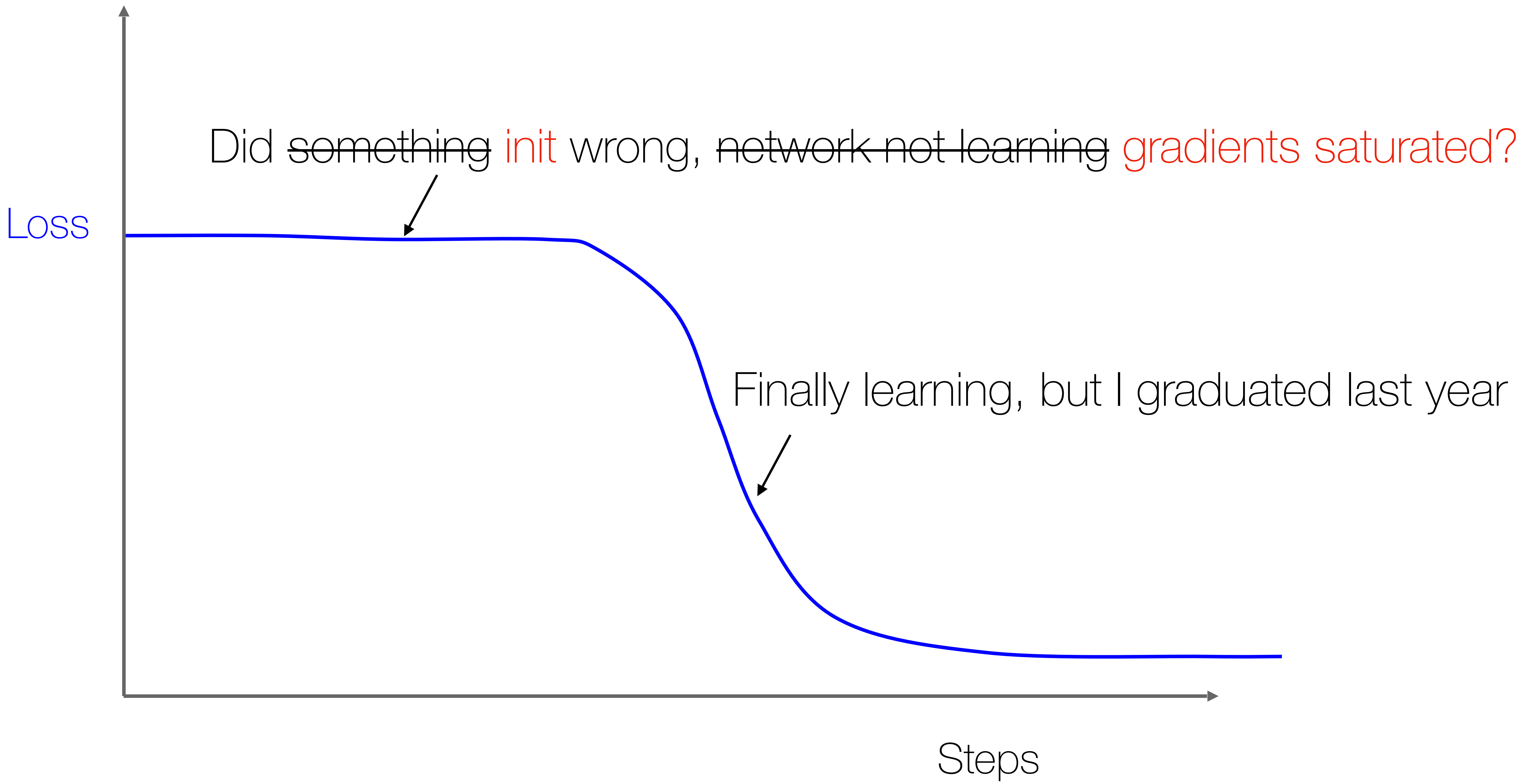
Recall: But it is never that easy

A typical sad loss curve



Recall: But it is never that easy

A typical sad loss curve

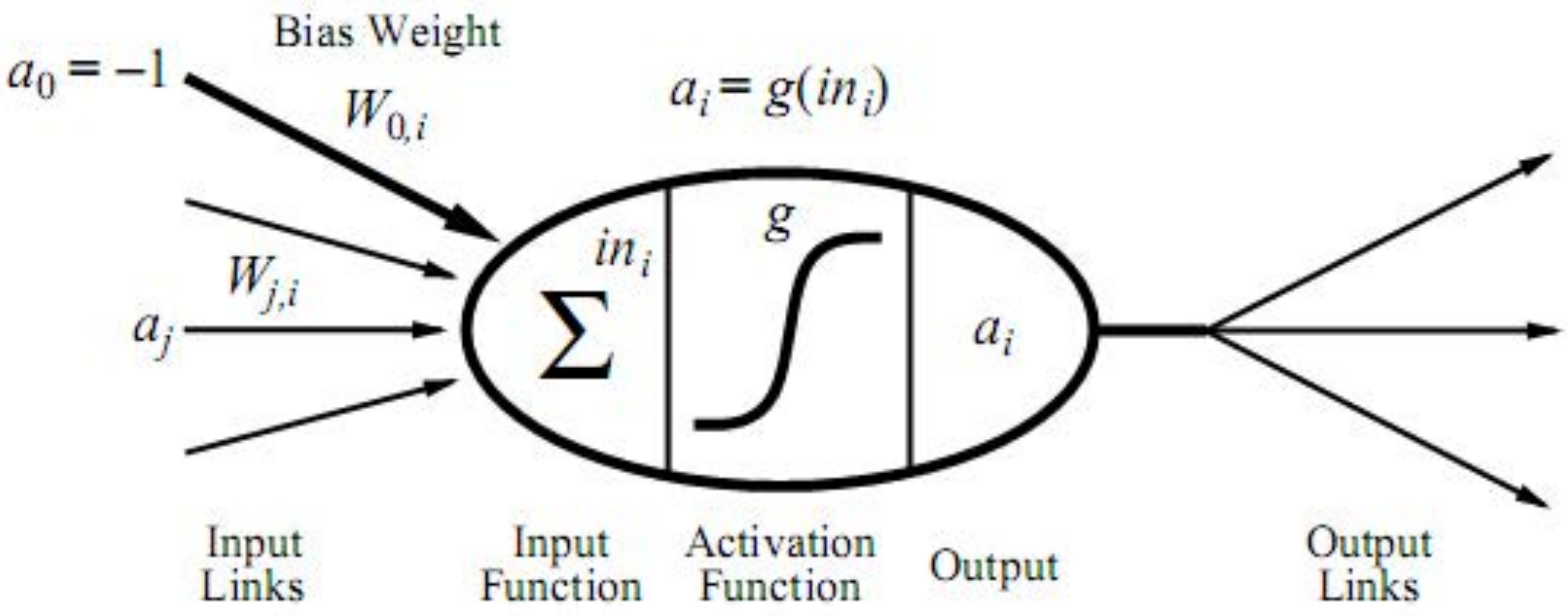


Normalization

Batch normalization [Ioffe and Szegedy, 2015]

Recall…

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$

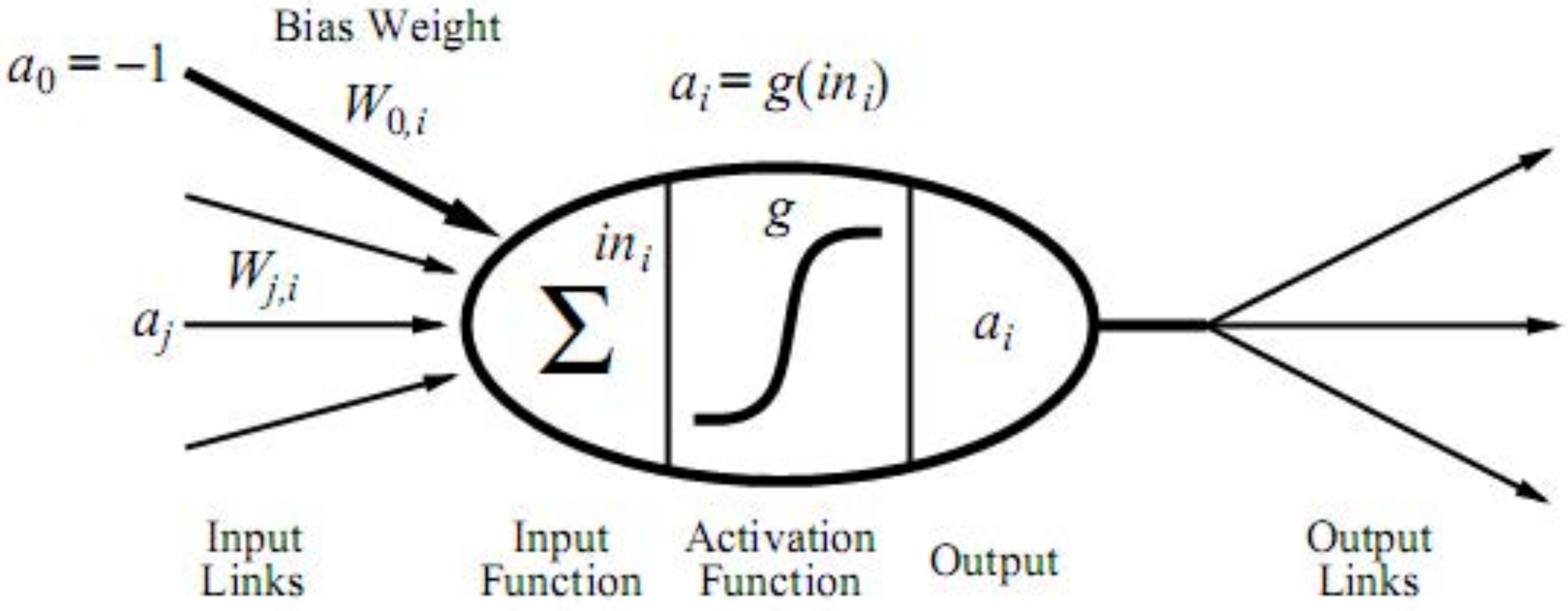


Batch normalization [Ioffe and Szegedy, 2015]

Recall…

Linear operations should cancel out

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



Batch normalization [Ioffe and Szegedy, 2015]

Forcing a zero-mean and unit standard deviation

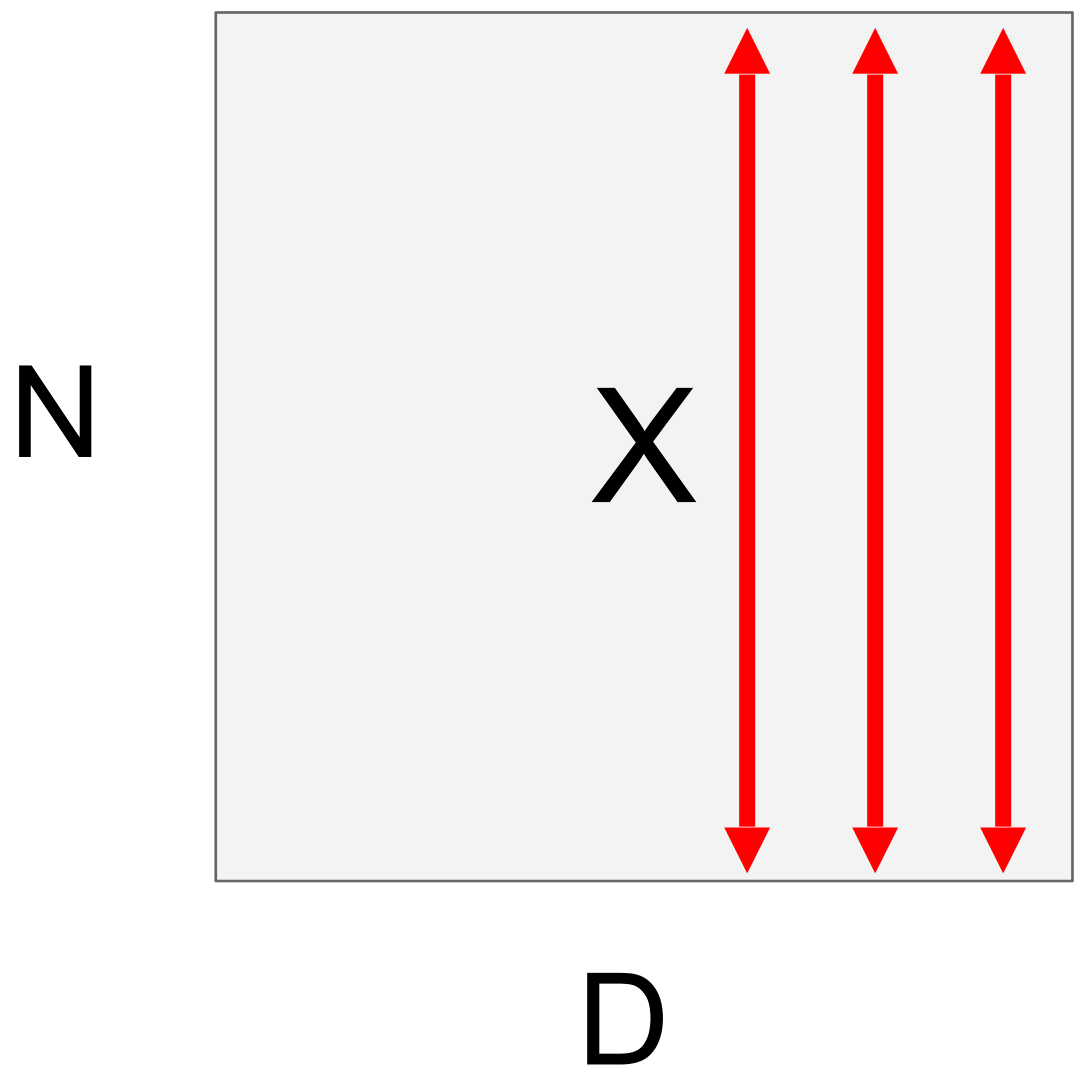
consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a linear differentiable function...

Batch normalization [Ioffe and Szegedy, 2015]

Forcing a zero-mean and unit standard deviation



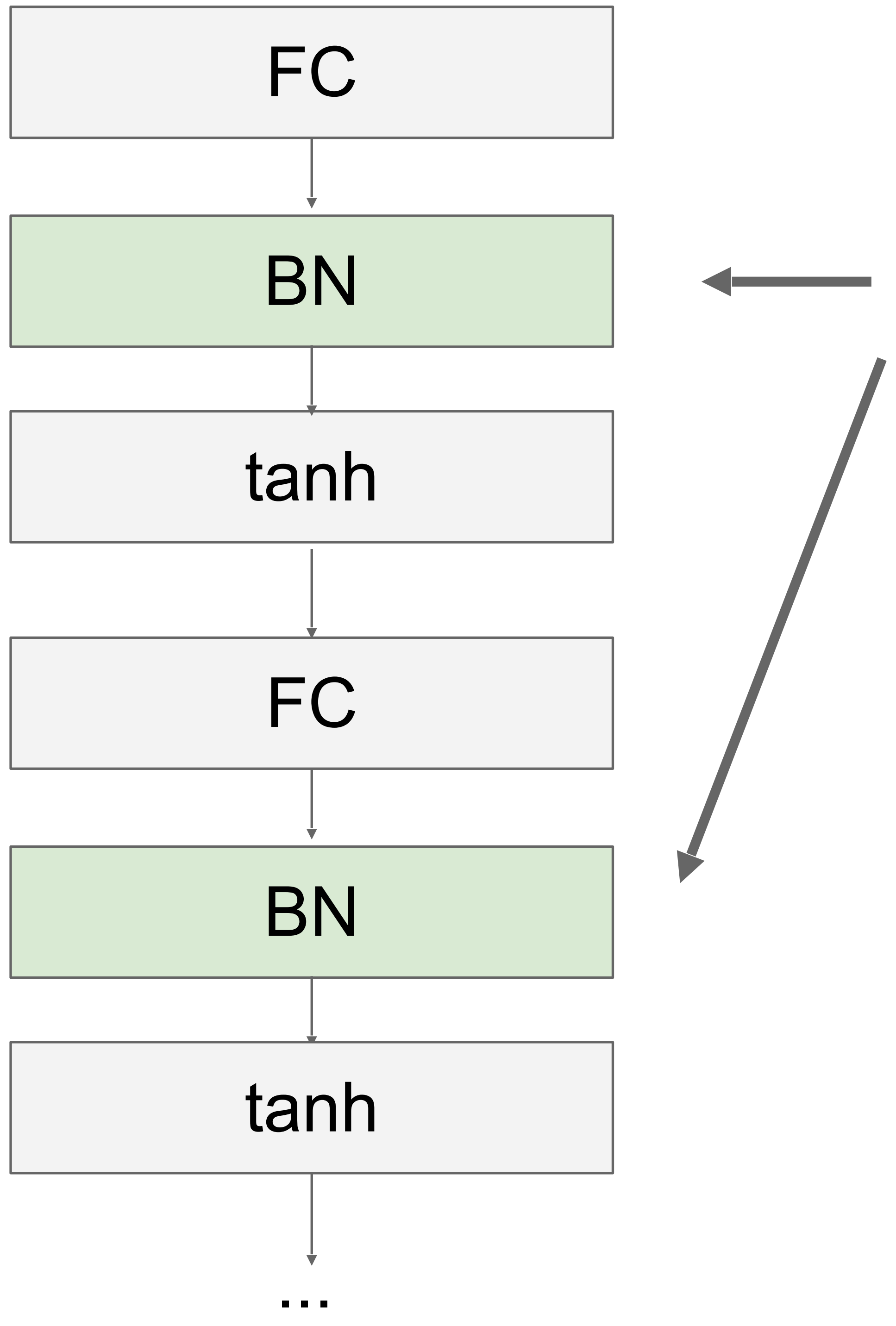
1. Compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch normalization [Ioffe and Szegedy, 2015]

Forcing a zero-mean and unit standard deviation



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch normalization [Ioffe and Szegedy, 2015]

Introducing learnable scale / shift

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Batch normalization [Ioffe and Szegedy, 2015]

Introducing learnable scale / shift

IMPORTANT: At test time, we don't have these — use training time stats

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

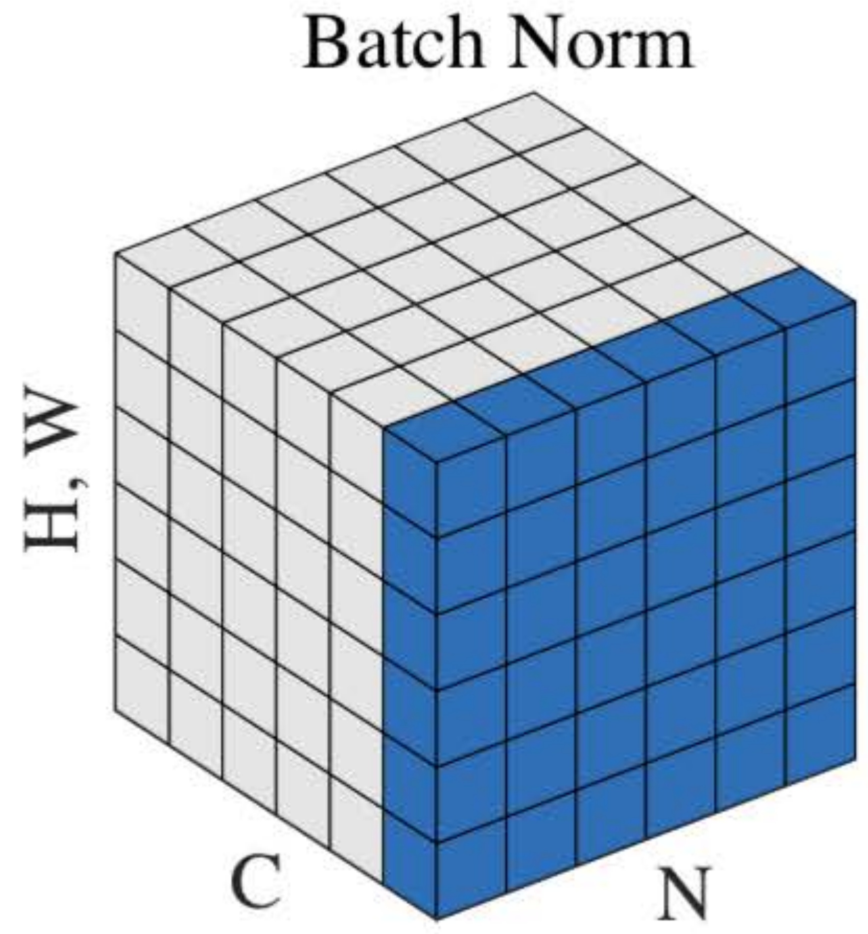


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Batch Normalization

Batch Normalization for **fully-connected** networks

$$\mathbf{x}: \; N \times D$$

Normalize ↓ (N)

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \; 1 \times D$$

$$\gamma, \beta: \; 1 \times D$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Batch Normalization for **convolutional** networks
(Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: \; N \times C \times H \times W$$

Normalize ↓ (N, H, W)

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \; 1 \times C \times 1 \times 1$$

$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

Batch Normalization for **fully-connected** networks

$\mathbf{x}: \boxed{\mathbf{N}} \times \mathbf{D}$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times \mathbf{D}$

$\gamma, \beta: 1 \times \mathbf{D}$

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$\mathbf{x}: \boxed{\mathbf{N}} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W}$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times \mathbf{C} \times 1 \times 1$

$\gamma, \beta: 1 \times \mathbf{C} \times 1 \times 1$

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

This is why train/test needs to be different

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

Batch Normalization for **fully-connected** networks

$\mathbf{x}: \boxed{N} \times D$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$\mathbf{x}: \boxed{N} \times C \times H \times W$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: \boxed{1} \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

Always watch out when implementing!!!

This is why train/test needs to be different

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Other normalization techniques

Batch Normalization

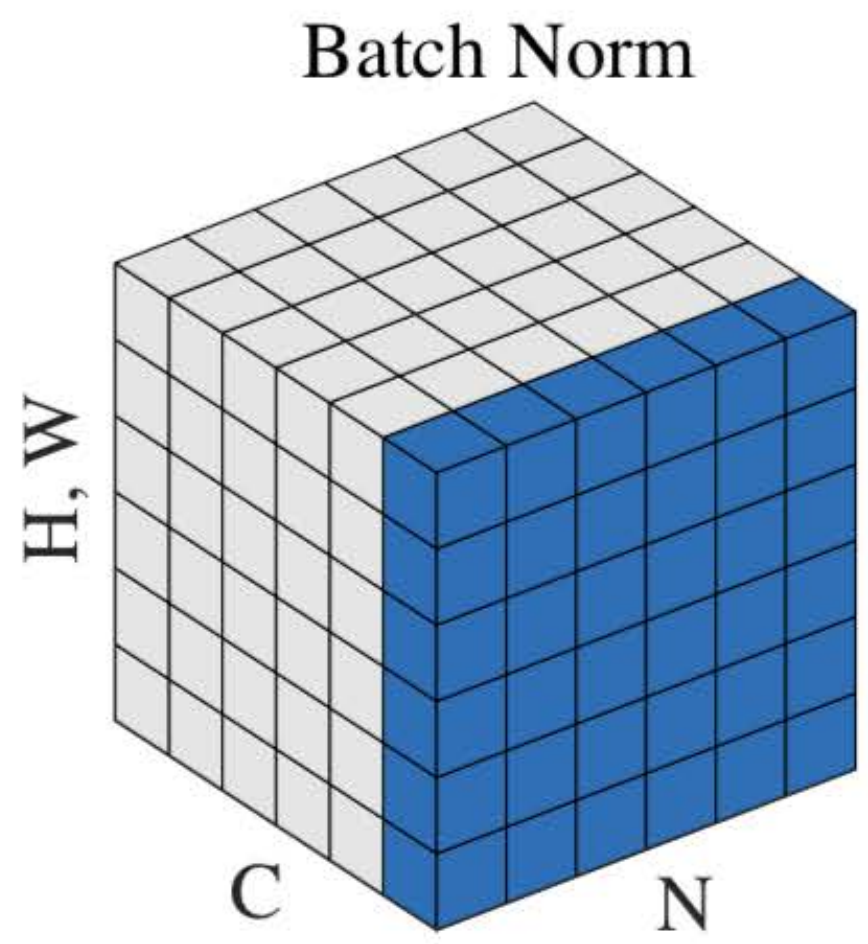
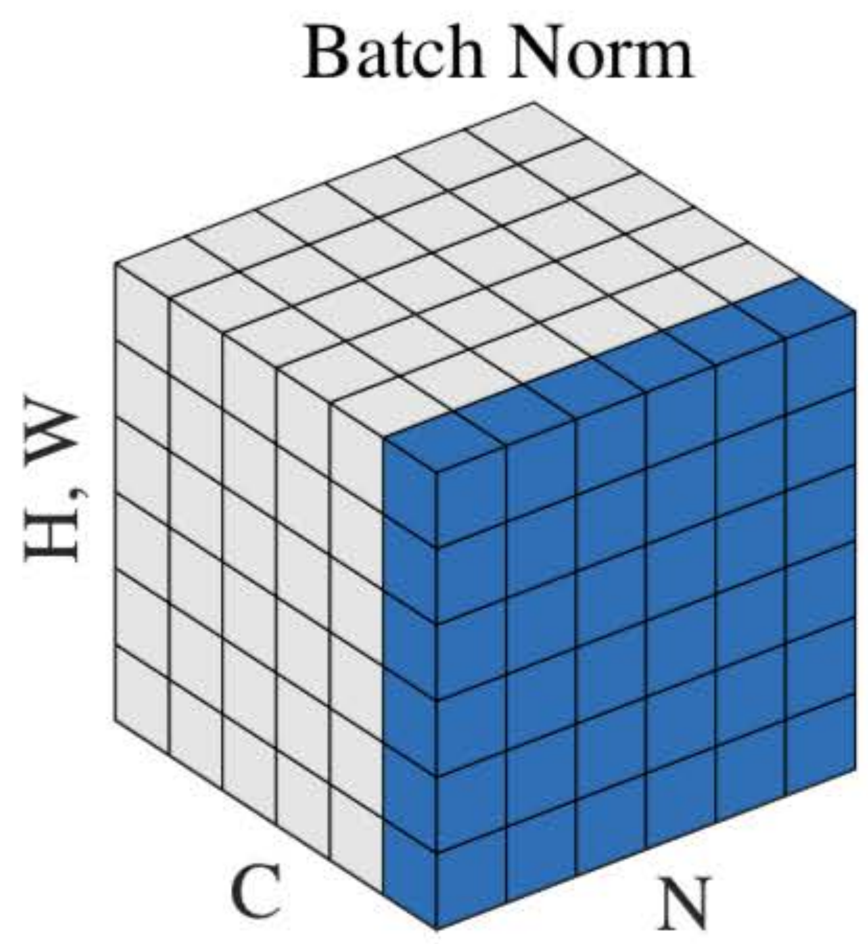


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Layer Normalization



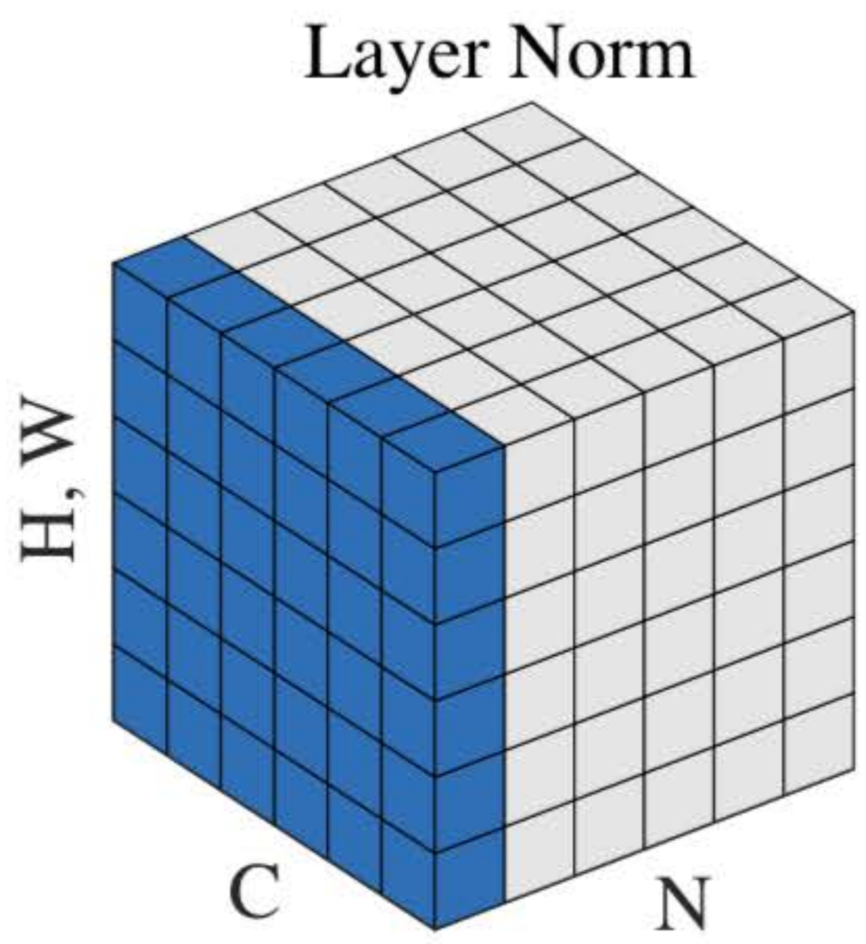


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Layer Normalization

Batch Normalization for **fully-connected** networks

$\mathbf{x}$: N × D

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}$: 1 × D

γ, β: 1 × D

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

Layer Normalization for fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$\mathbf{x}$: N × D

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}$: N × 1

γ, β: 1 × D

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Instance Normalization

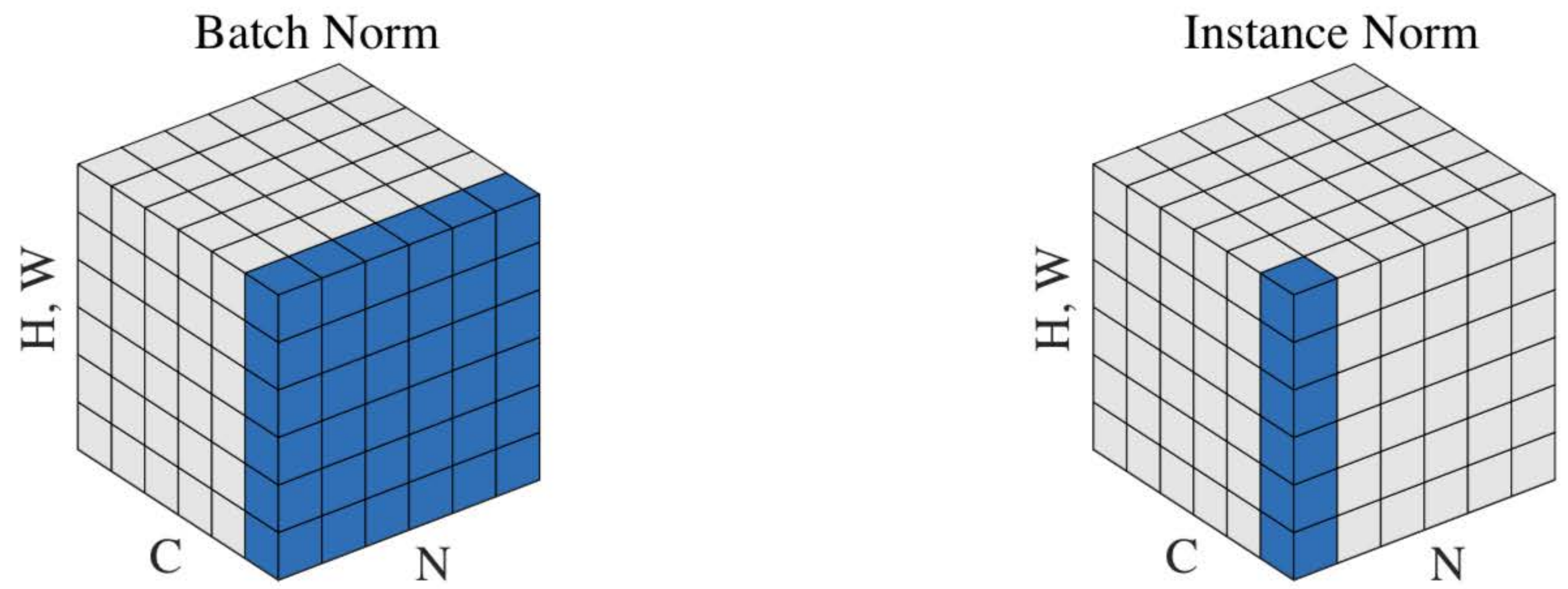


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Instance Normalization

Batch Normalization for convolutional networks

$$\boxed{\begin{array}{rl} \mathbf{x}: & N \times C \times H \times W \\ \text{Normalize} & \downarrow \qquad\quad \downarrow \quad \downarrow \\ \mu, \sigma: & 1 \times C \times 1 \times 1 \end{array}}$$

$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Instance Normalization for convolutional networks
Same behavior at train / test!

$$\boxed{\begin{array}{rl} \mathbf{x}: & N \times C \times H \times W \\ \text{Normalize} & \qquad\qquad\quad \downarrow \quad \downarrow \\ \mu, \sigma: & N \times C \times 1 \times 1 \end{array}}$$

$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Group Normalization

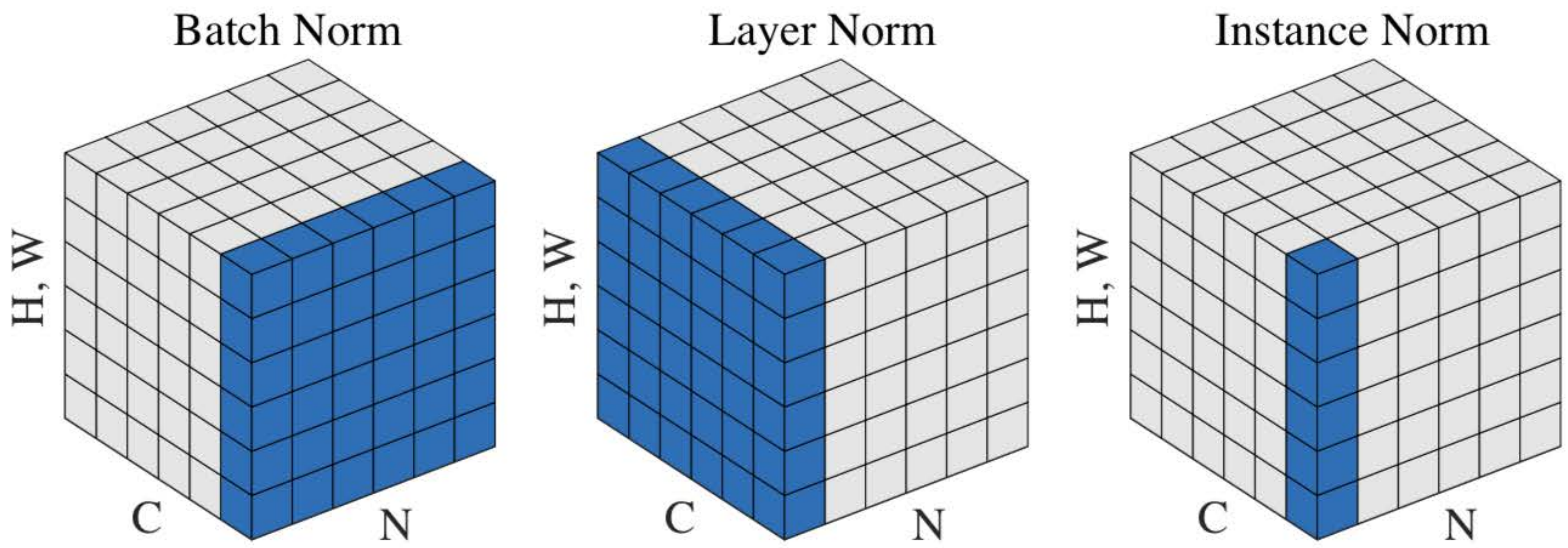


Image from Wu and He 2018. Reproduced for educational purposes.

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Other normalization techniques

Group Normalization

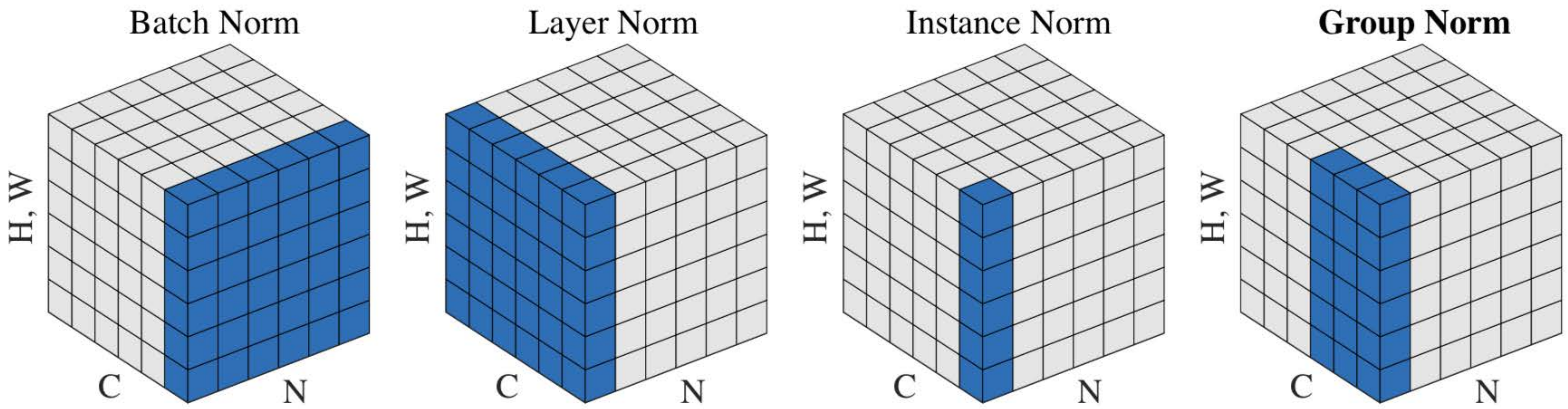
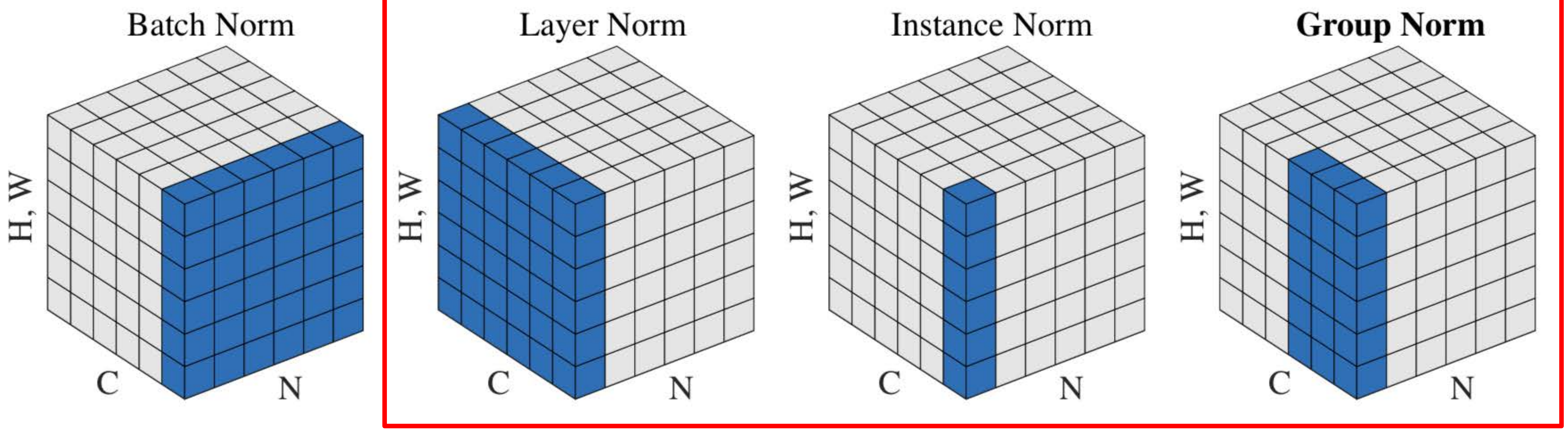


Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Group Normalization



No train/test-time differences.

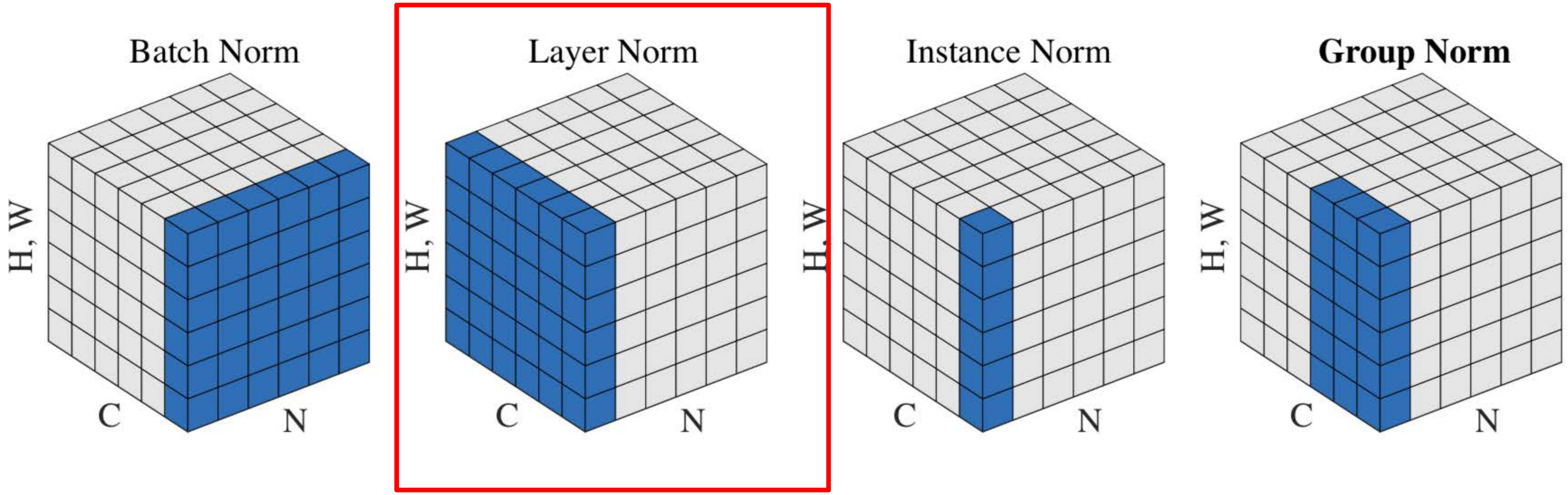
Much preferred in my opinion.

Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class (outside of scope)

Other normalization techniques

Group Normalization



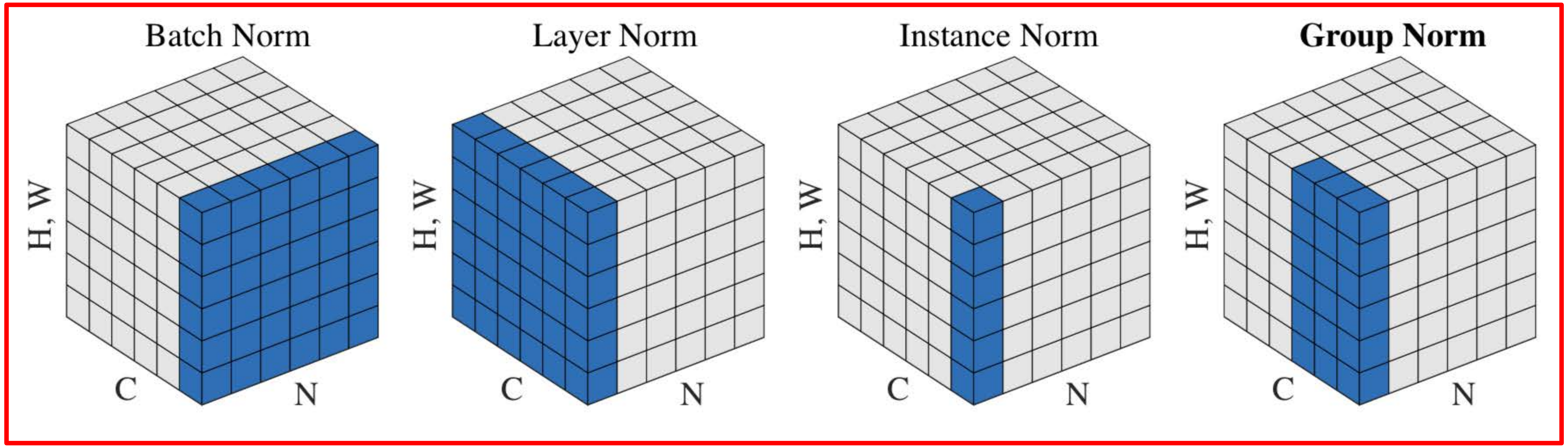
Can be implemented using PyTorch's Group norm.

Image from Wu and He 2018. Reproduced for educational purposes.

Skipped in class
(outside of scope)

Other normalization techniques

Group Normalization

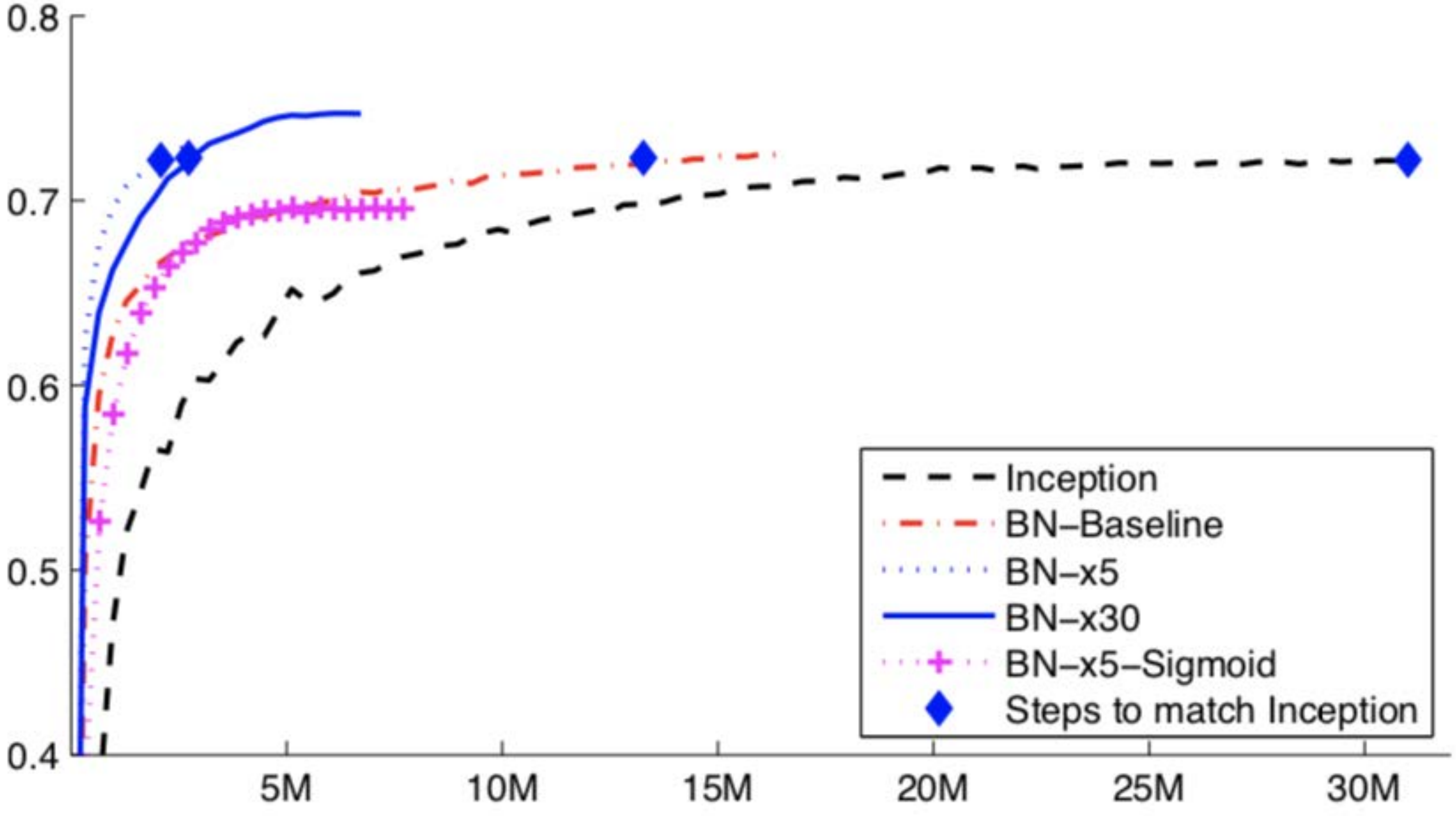


Choice of normalization should be data dependent

By the way… with normalization something else also happens

[Ioffe and Szegedy, 2015]

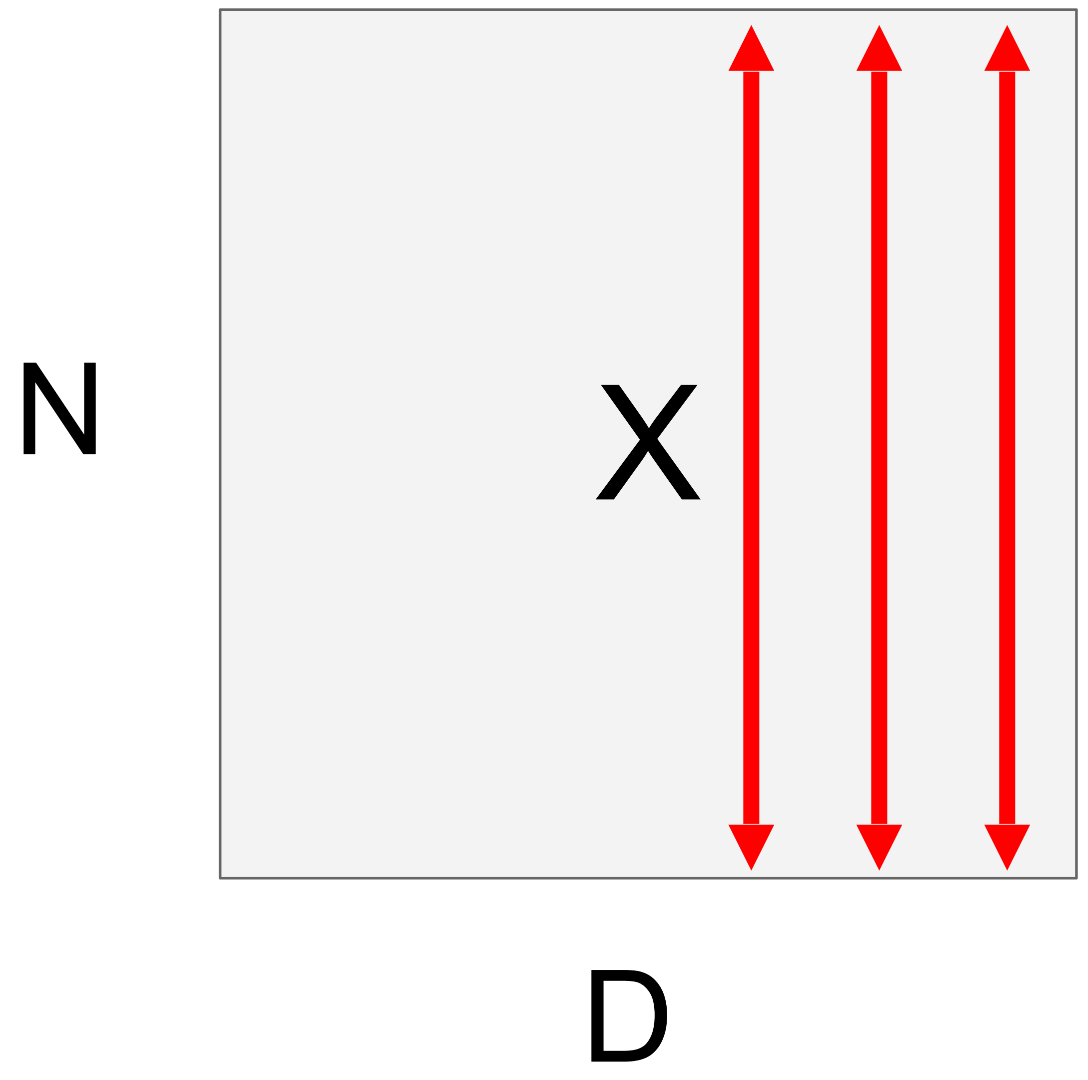
Batch normalization



[Ioffe and Szegedy, 2015]

Batch normalization

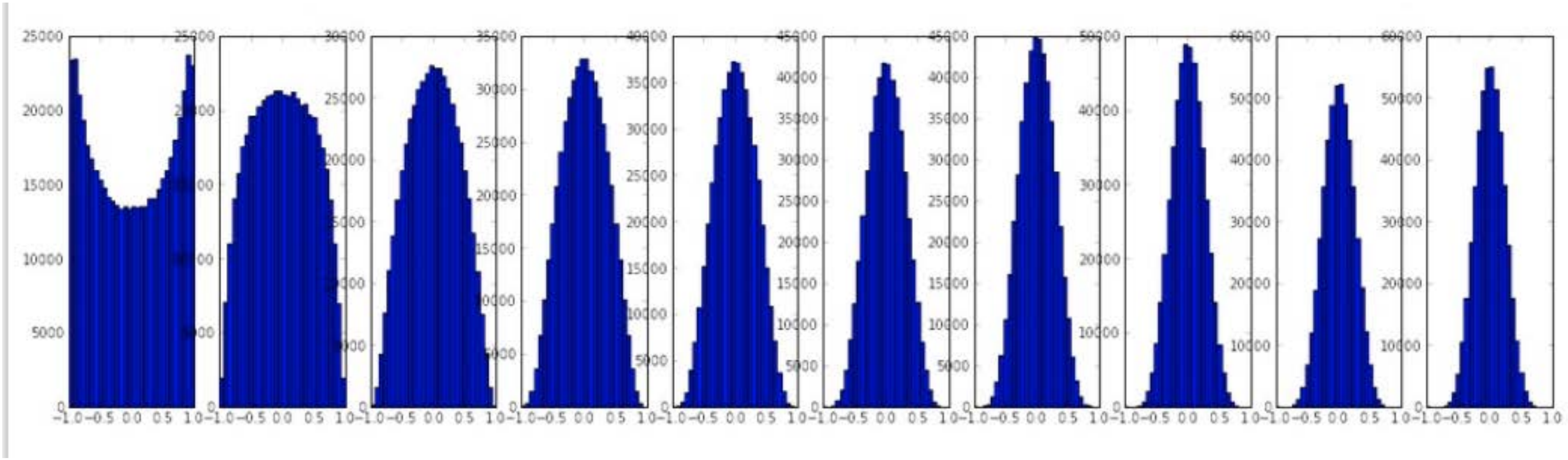
Recall...



1. compute the empirical mean and variance independently for each dimension.

2. Normalize

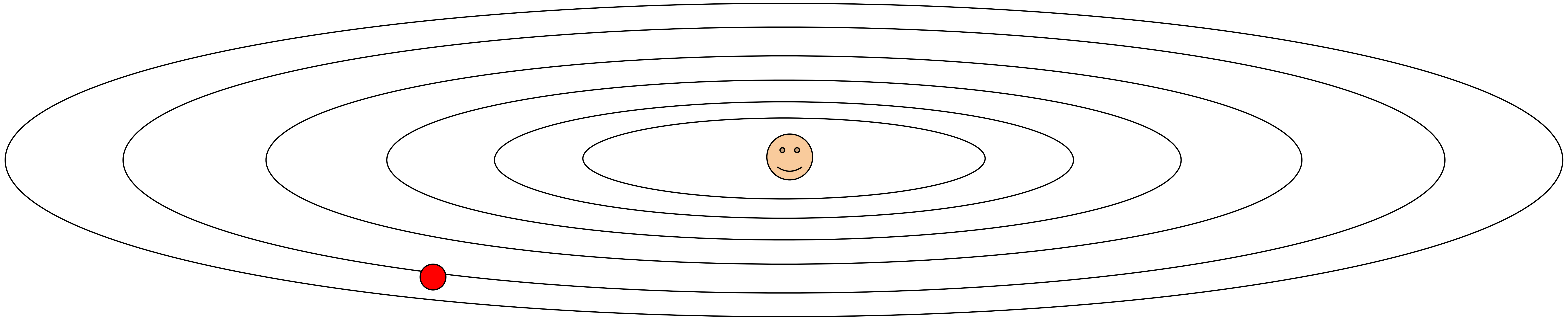
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$



[Ioffe and Szegedy, 2015]

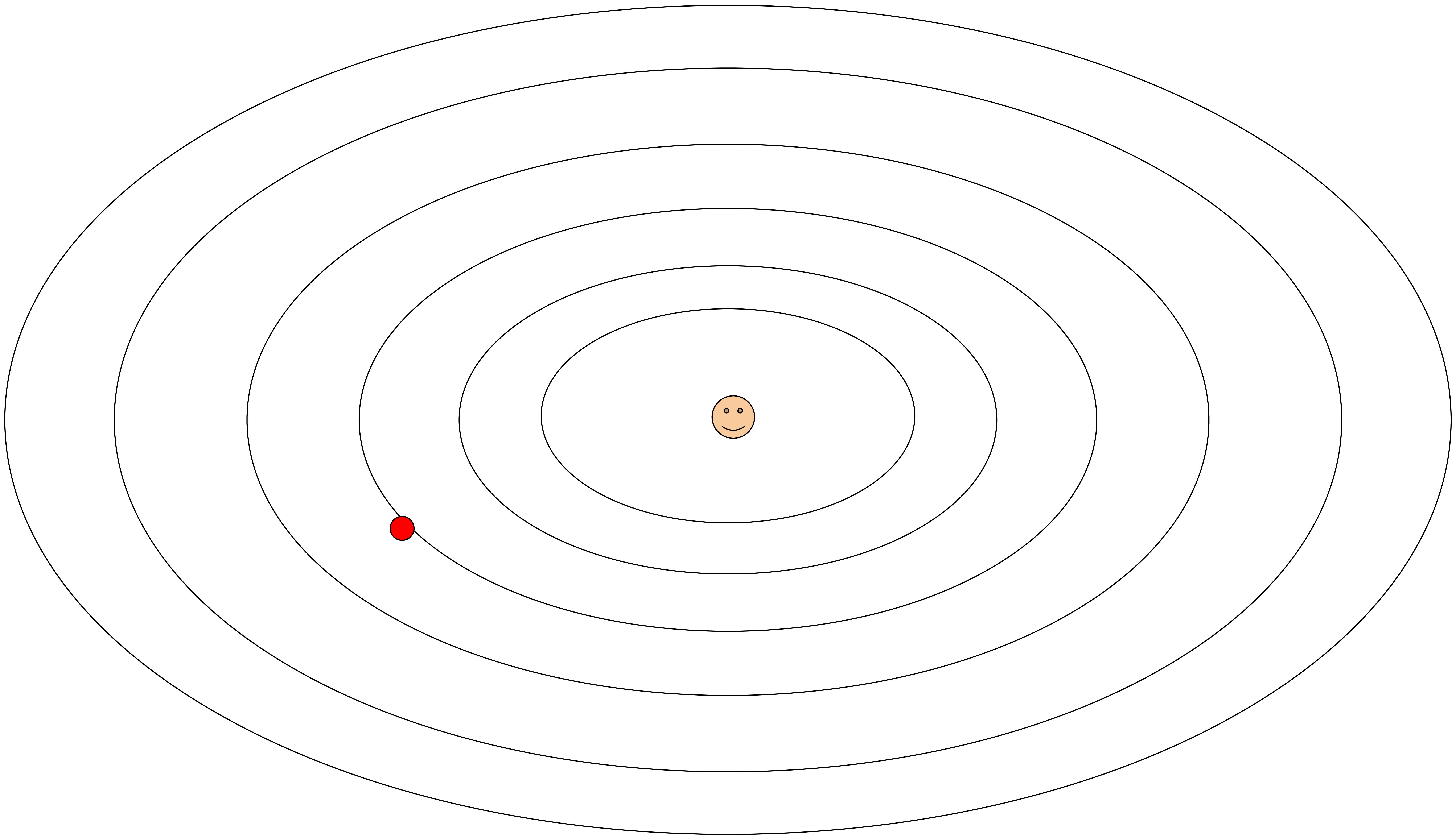
Batch normalization

This imbalance between dimensions is the problem



[Ioffe and Szegedy, 2015]

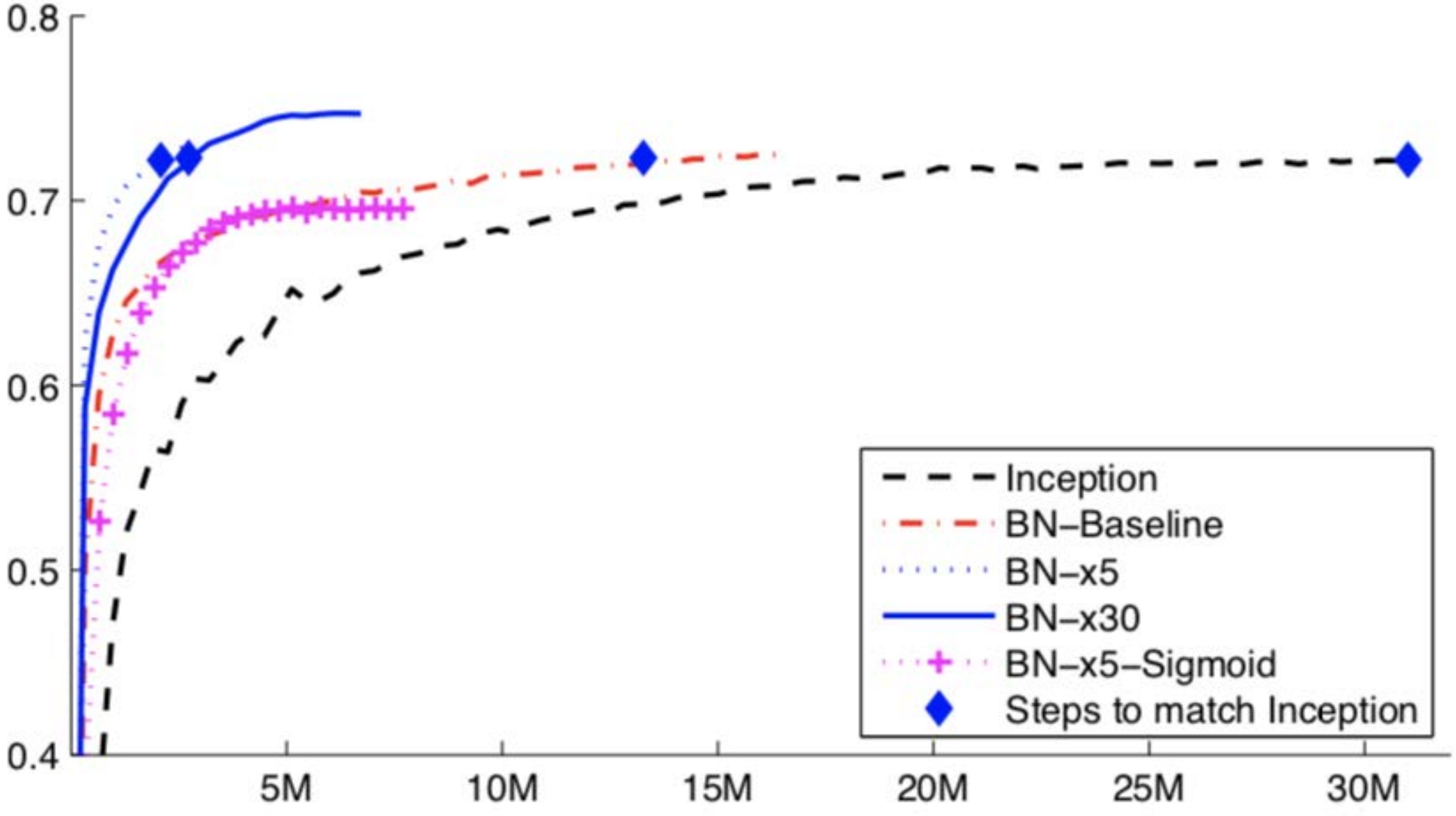
Batch normalization



Let's artificially make it like this!

[Ioffe and Szegedy, 2015]

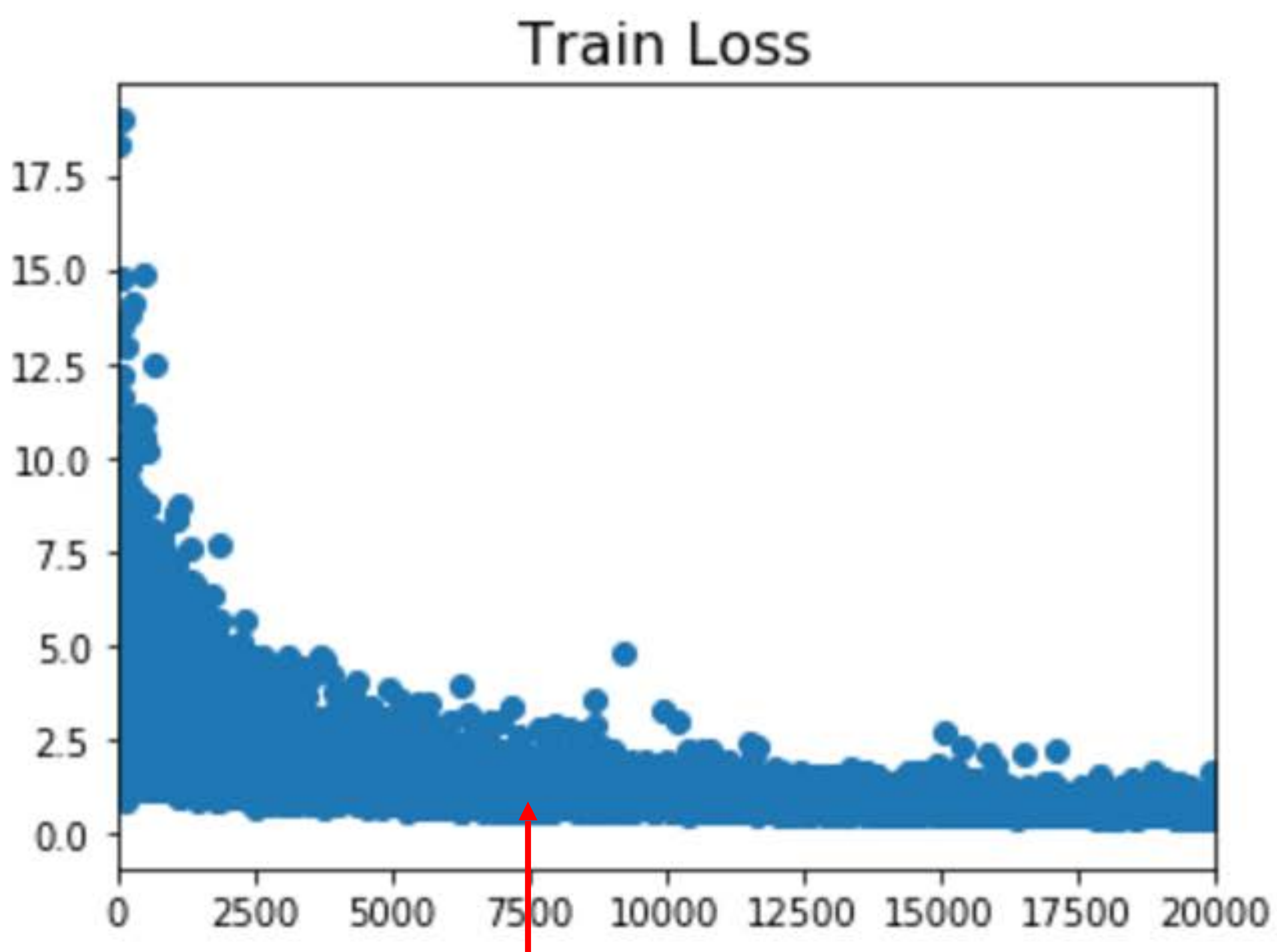
Batch normalization

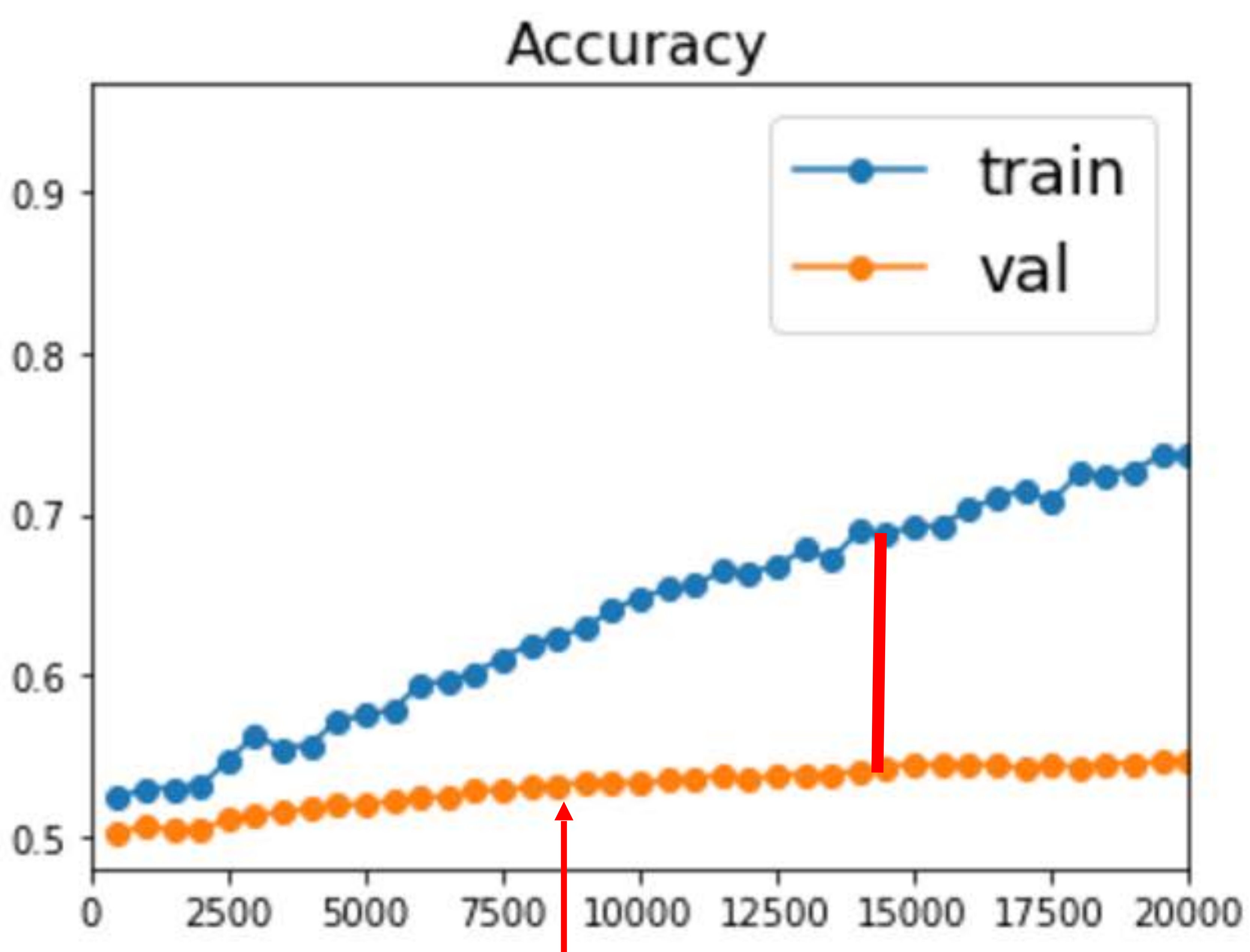


Preventing overfitting

Beyond training loss

Recall the other problem



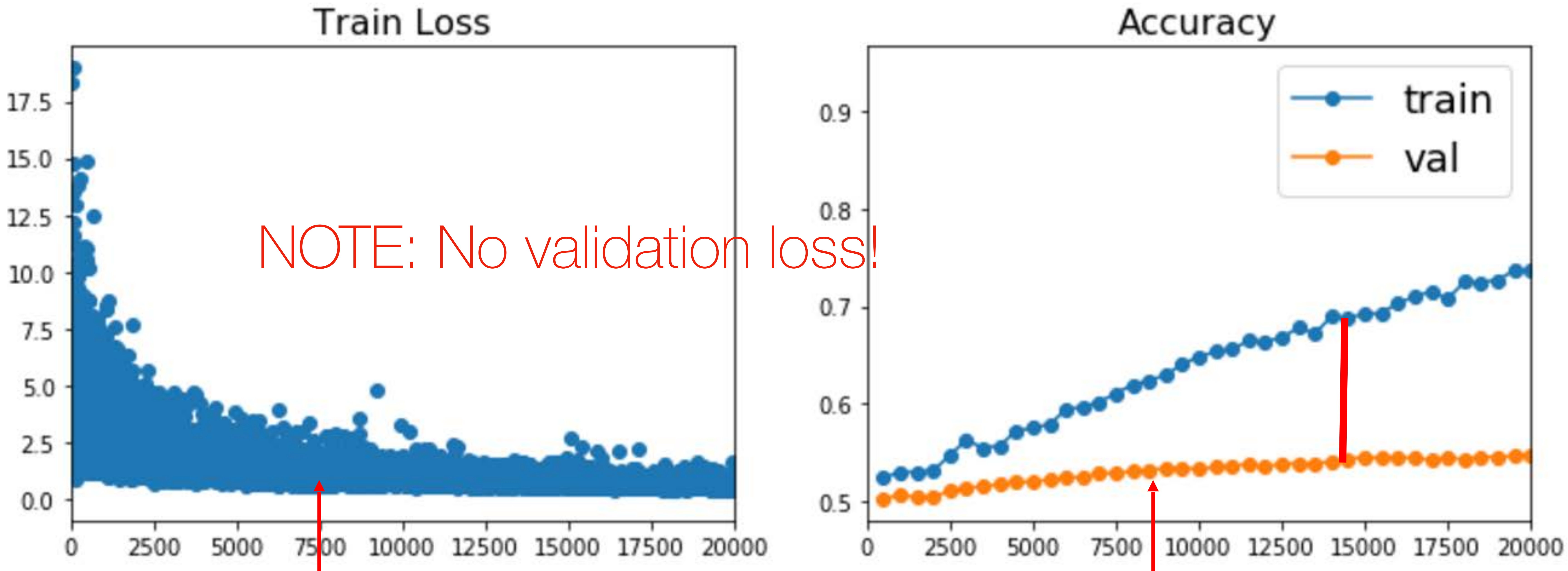


Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

Beyond training loss

Recall the other problem



NOTE: No validation loss!

Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

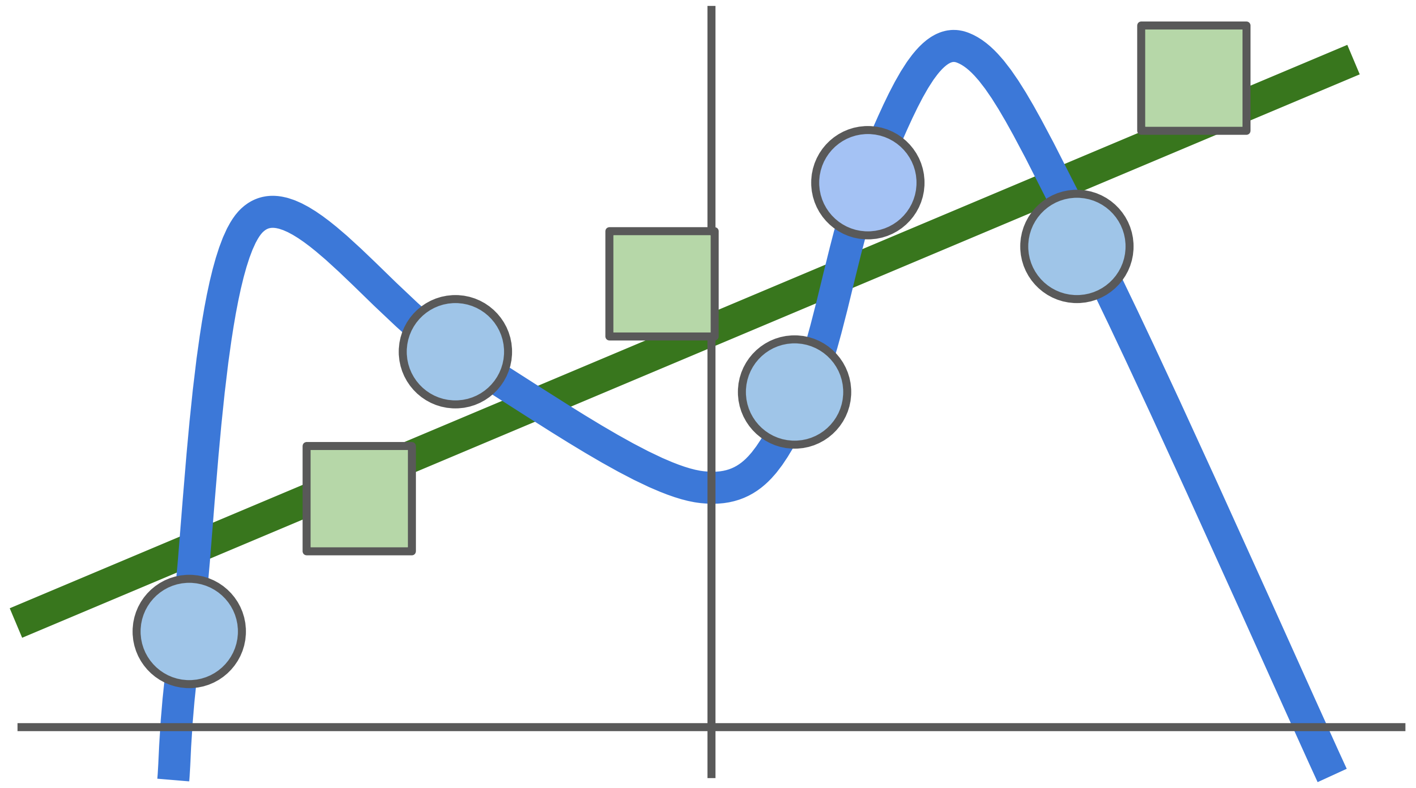
A typical approach to overfitting

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be “simple”, so it works on test data $\|\mathbf{W}\|_2^2$



Occam’s Razor:
“Among competing hypotheses, the simplest is the best”
William of Ockham, 1285 - 1347

Common regularizers

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Common regularizers

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \boxed{\sum_k \sum_l W_{k,l}^2 \quad \text{(Weight decay)}}$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Common regularizers

My personal warning against L2

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

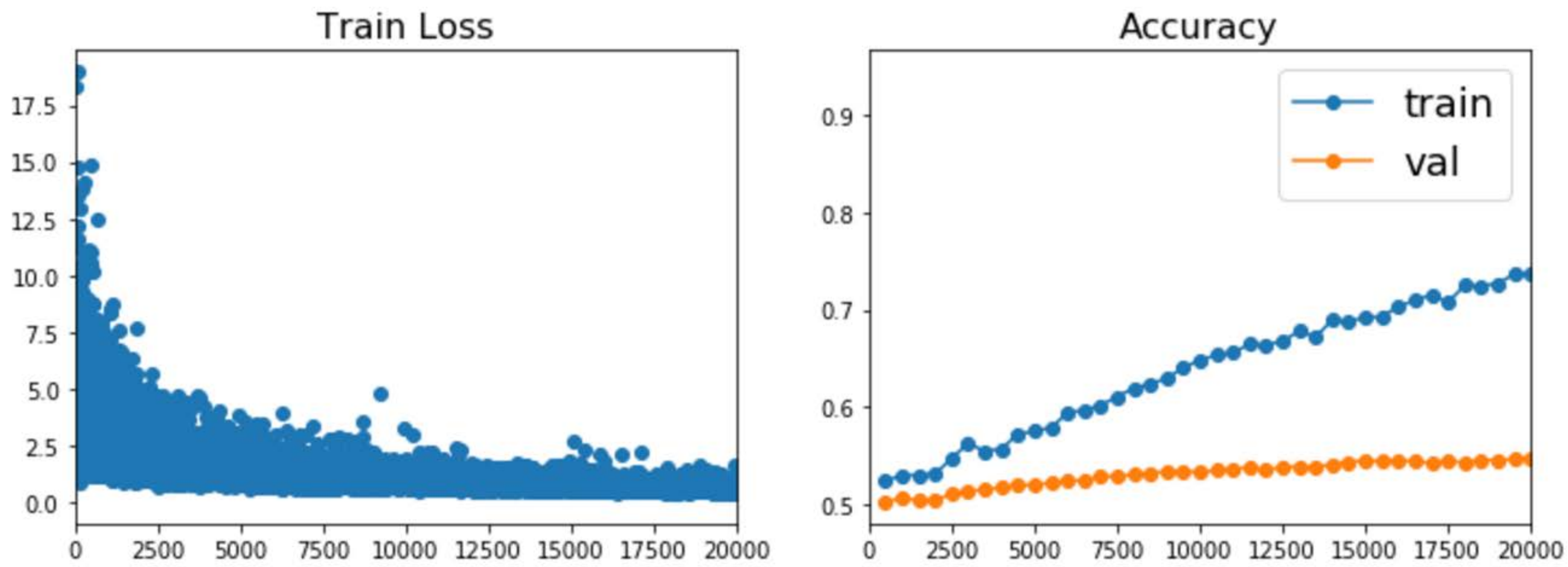
L2 regularization $R(W) = \boxed{\sum_k \sum_l W_{k,l}^2}$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

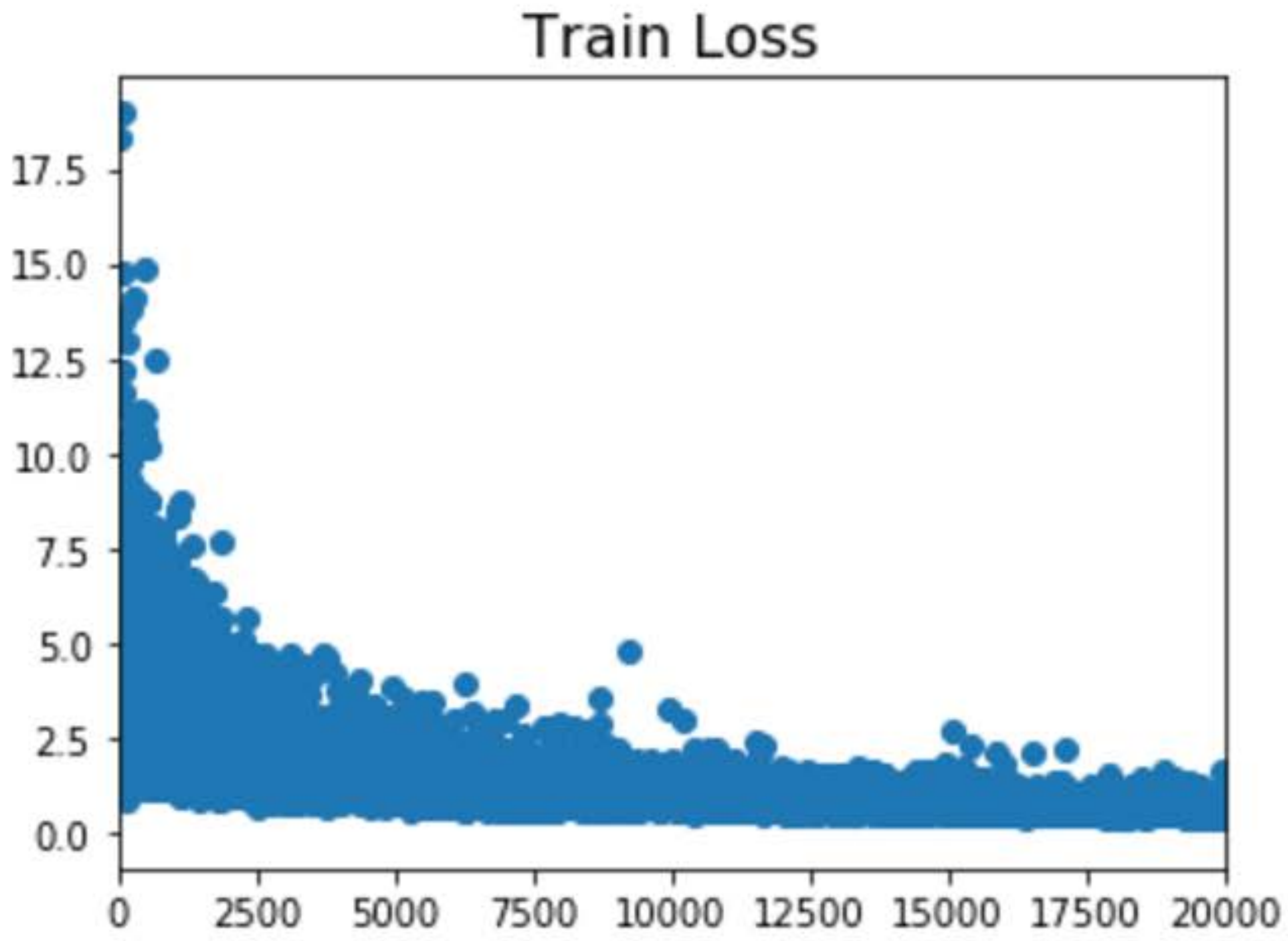
Ela

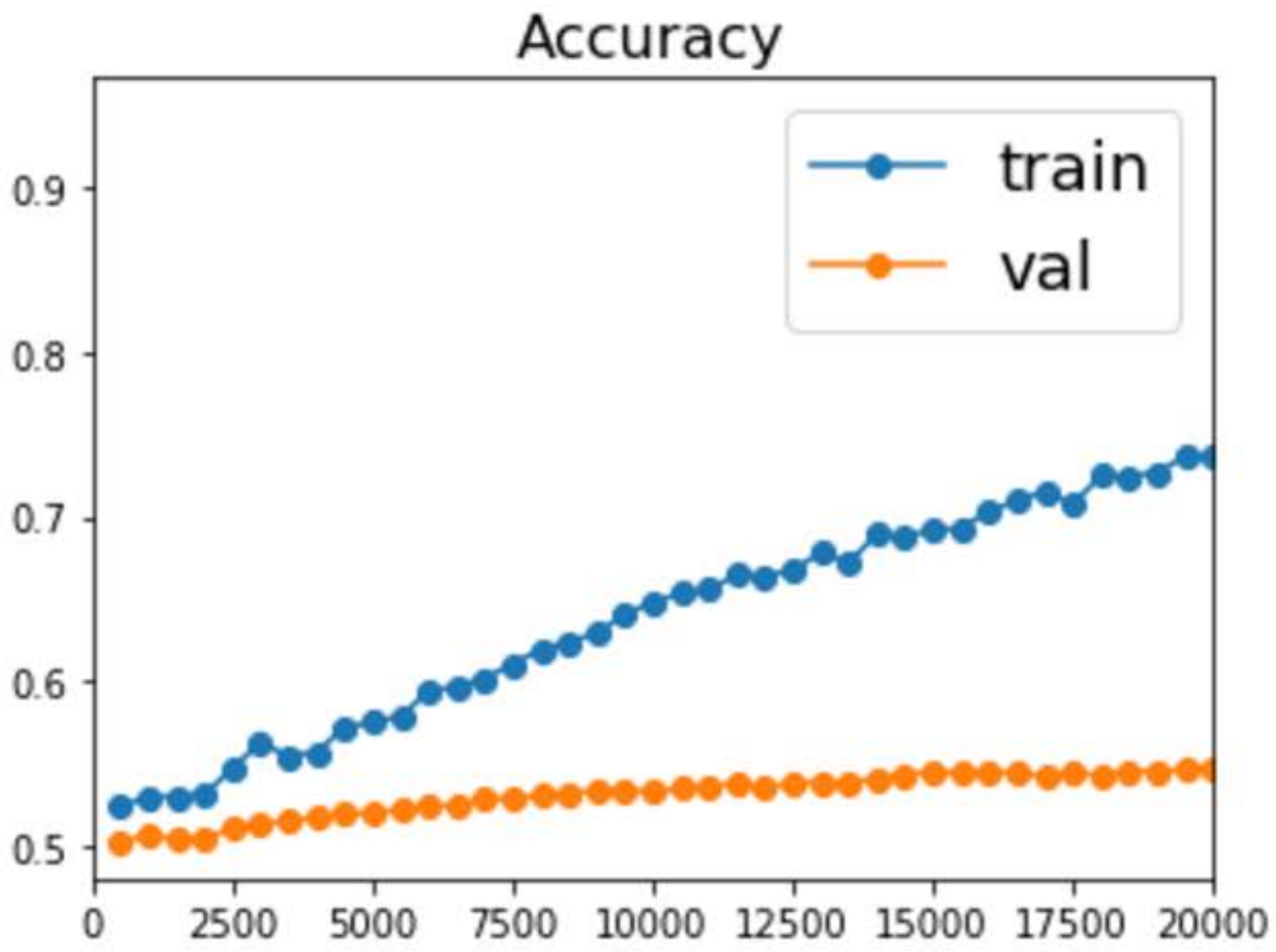
Laarhoven, 2017, "However, we show that L2 regularization has **no regularizing effect when combined with normalization**. Instead, regularization has an influence on the scale of weights, and thereby on the effective learning rate."

Why does this happen in the first place?



Why does this happen in the first place?



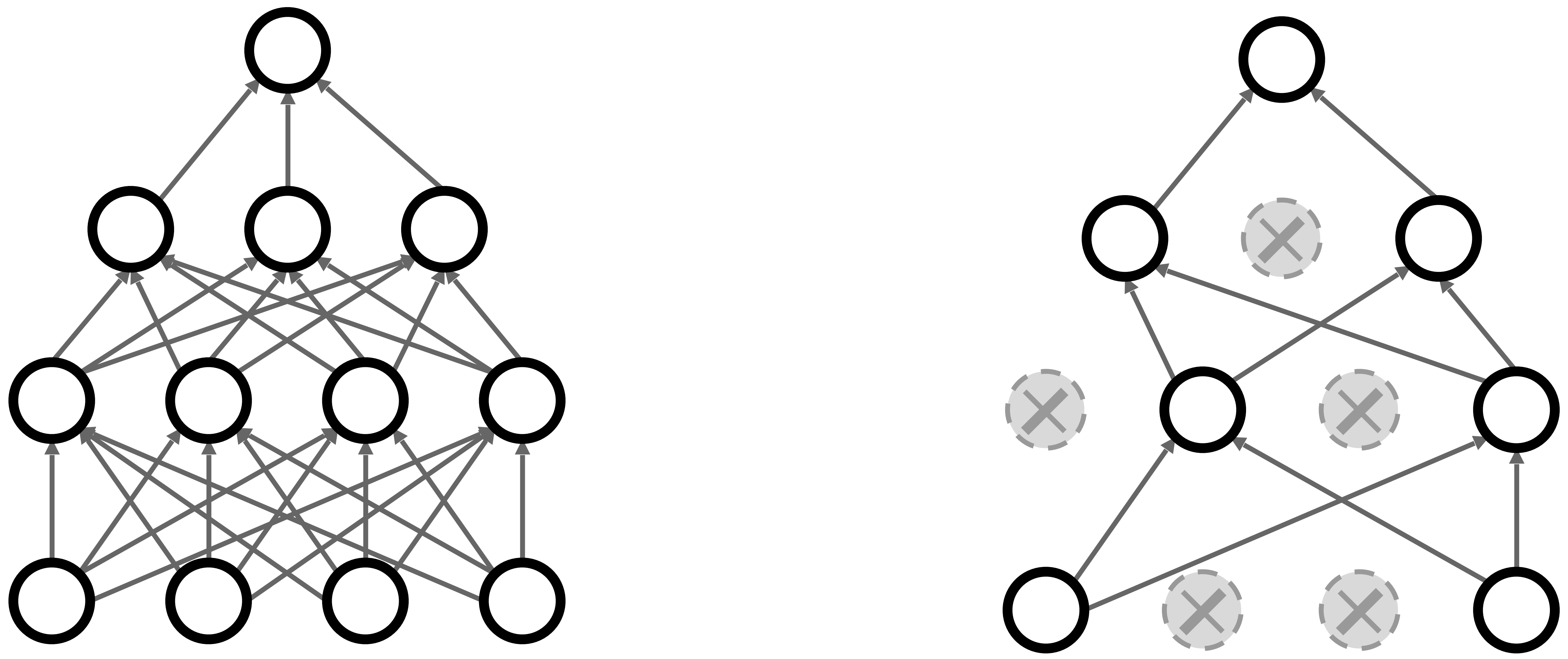


Can we somehow encode
uncertainty in data?

Regularization: Dropout

Making it impossible to trust the data 100%

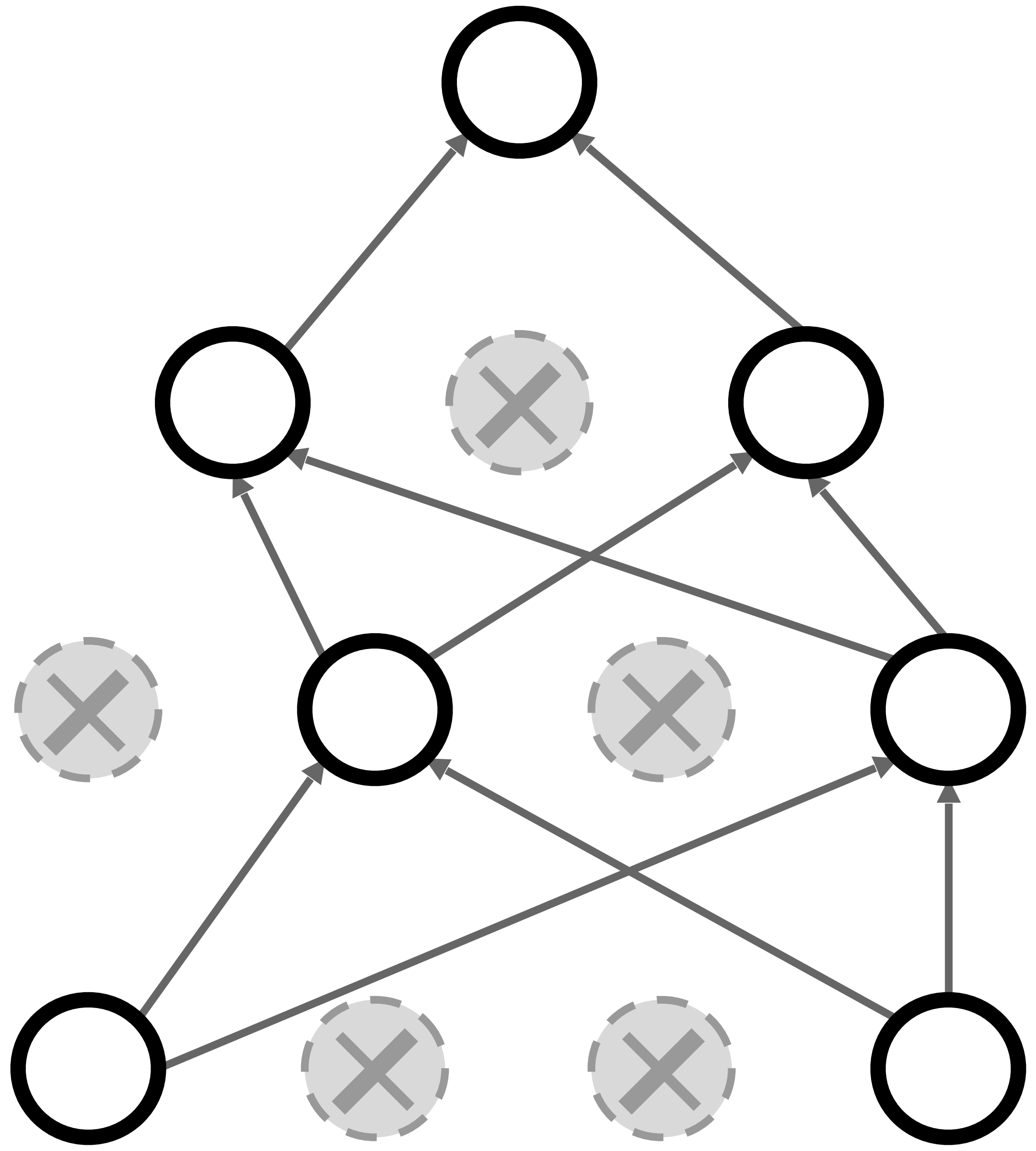
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Regularization: Dropout

Making it impossible to trust the data 100%



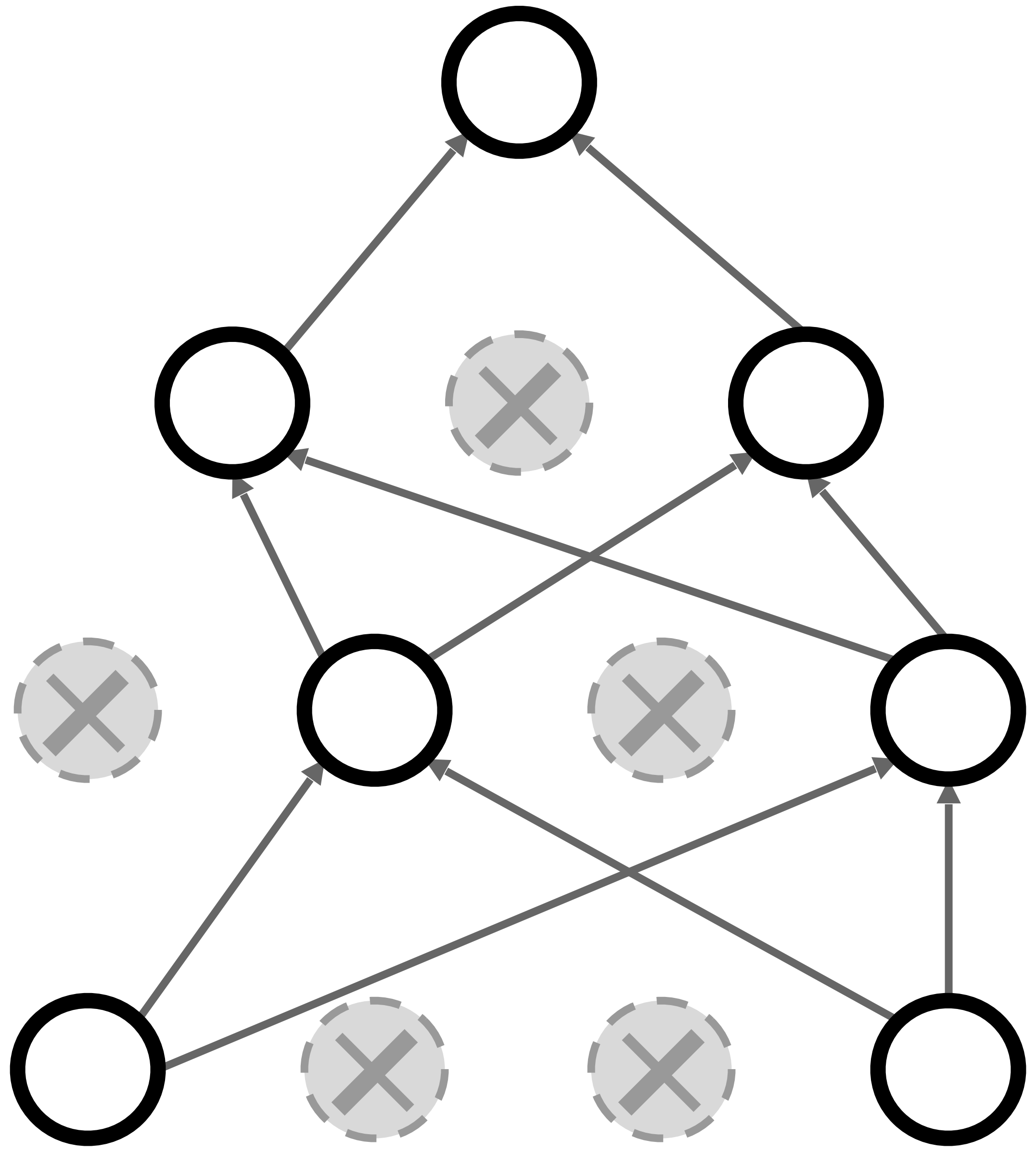
Forces the network to have a redundant representation;
Prevents co-adaptation of features

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Skipped in class (outside of scope)

Regularization: Dropout

Making it impossible to trust the data 100%



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Skipped in class (outside of scope)

Regularization: Dropout at **test** time

Again the train / test gap

Dropout makes our output random!

Output (label) | Input (image) | Random mask

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Want to "average out" the randomness at test-time

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

But this integral seems hard …

Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

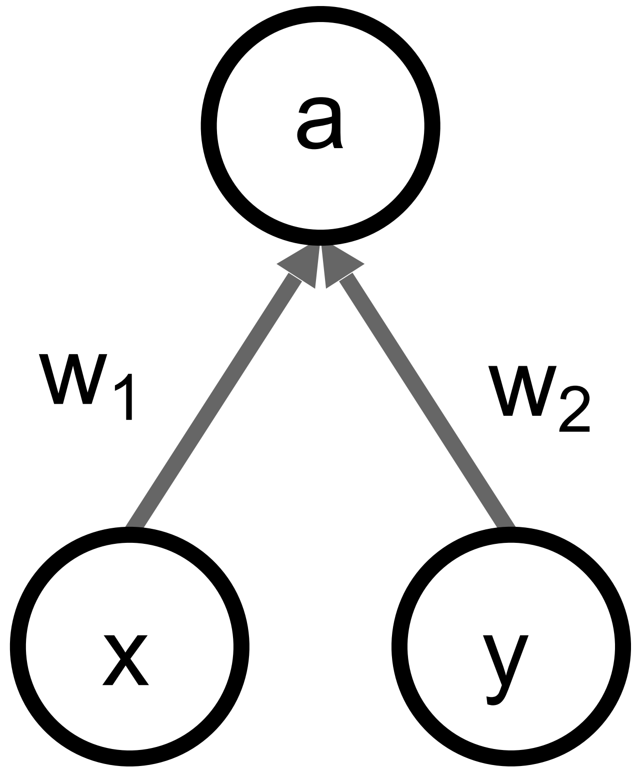
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Consider a single neuron.



Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

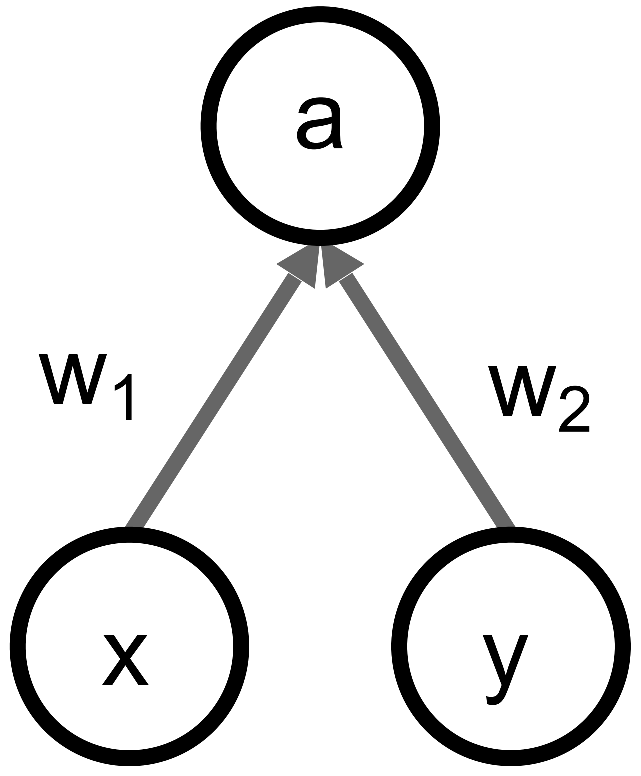
Skipped in class
(outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $y = f(x) = E_z\big[f(x,z)\big] = \int p(z)f(x,z)dz$

Consider a single neuron.



At test time we have: $E\big[a\big] = w_1x + w_2y$

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

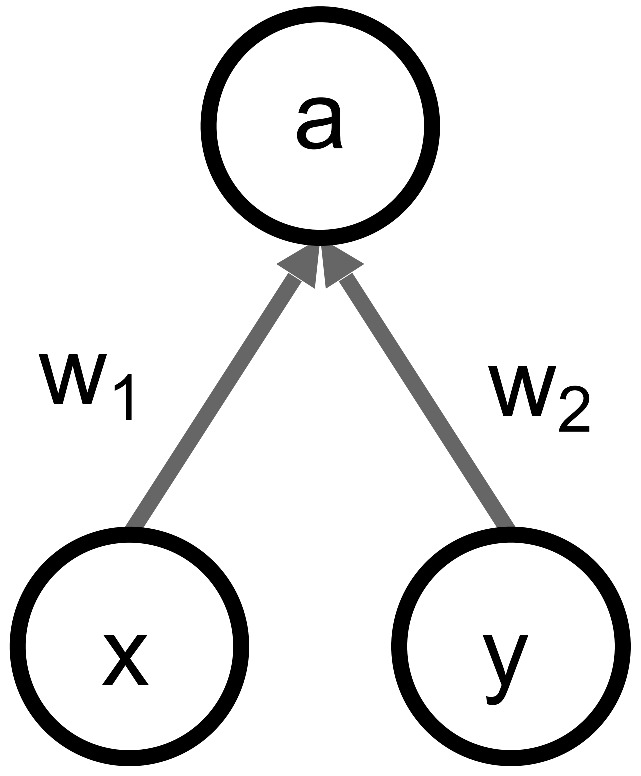
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $$y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$$

Consider a single neuron.



At test time we have: $E\left[a\right] = w_1 x + w_2 y$

During training we have:
$$\begin{aligned} E\left[a\right] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

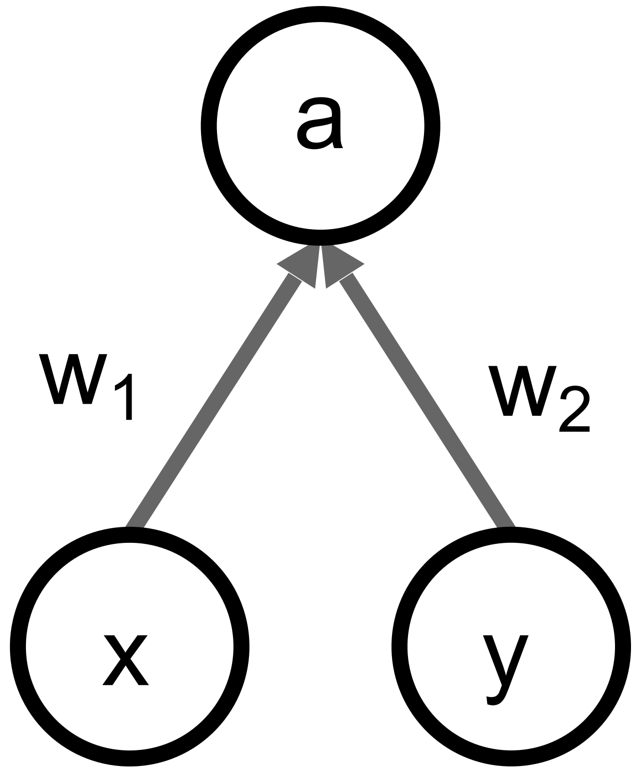
Skipped in class (outside of scope)

Regularization: Dropout at **test** time

An approximate solution

Want to approximate the integral $y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$

Consider a single neuron.



At test time we have: $E\big[a\big] = w_1 x + w_2 y$

During training we have:

$$E\big[a\big] = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) = \frac{1}{2}(w_1 x + w_2 y)$$

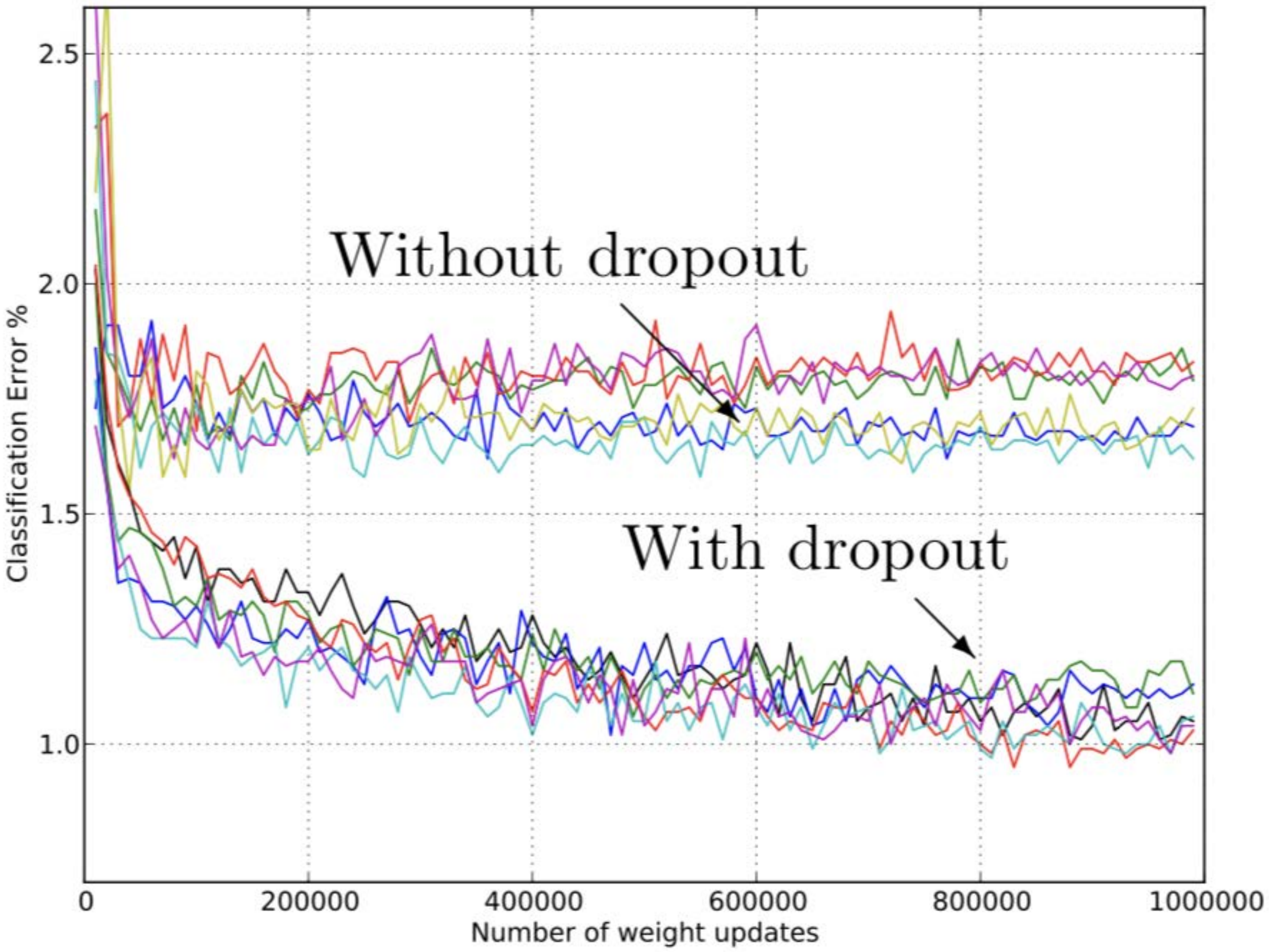
At test time, **multiply** by dropout probability

Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Skipped in class (outside of scope)

Regularization: Dropout

How good is it?



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Skipped in class (outside of scope)

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Skipped in class
(outside of scope)

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

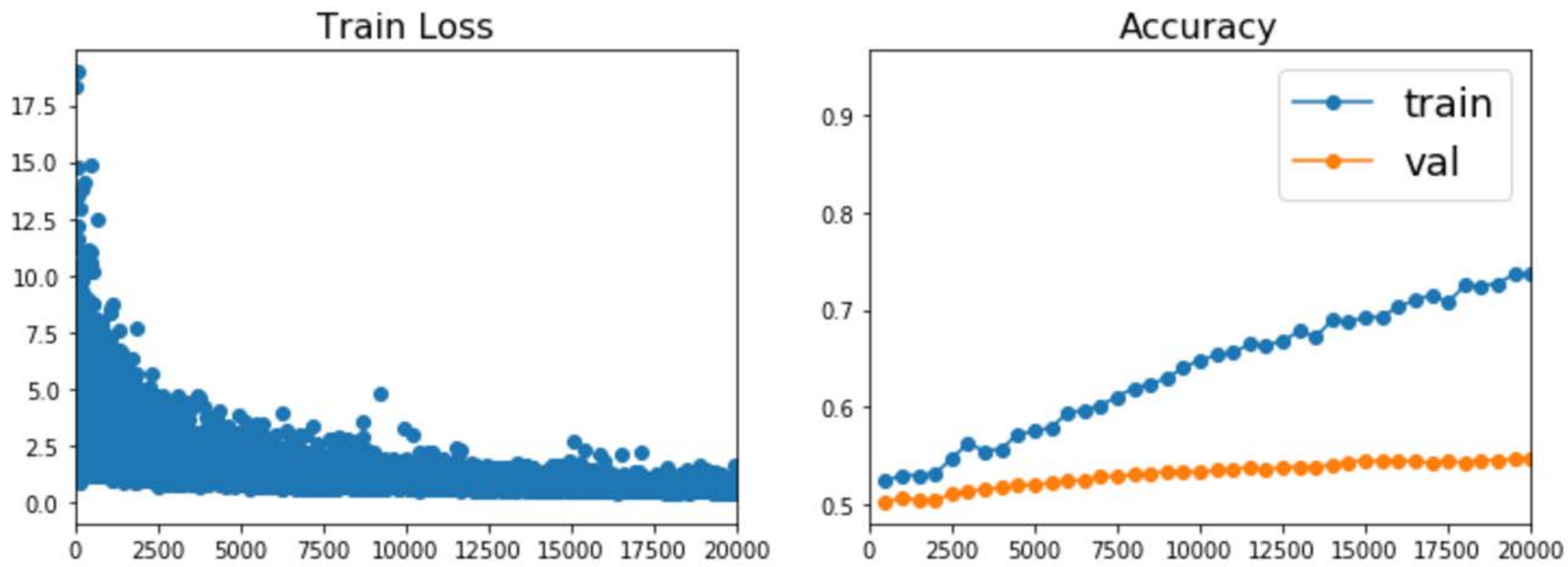
Example: Batch Normalization

Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

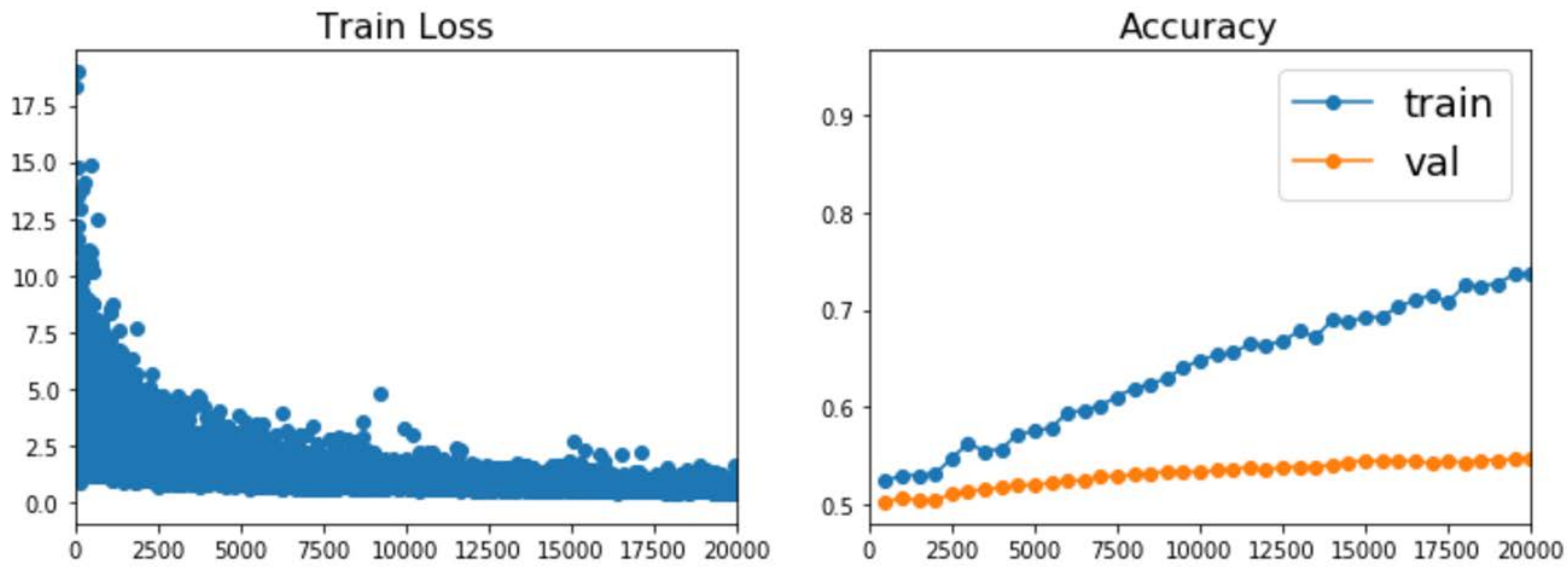
Skipped in class (outside of scope)

Why does this happen in the first place?



Skipped in class (outside of scope)

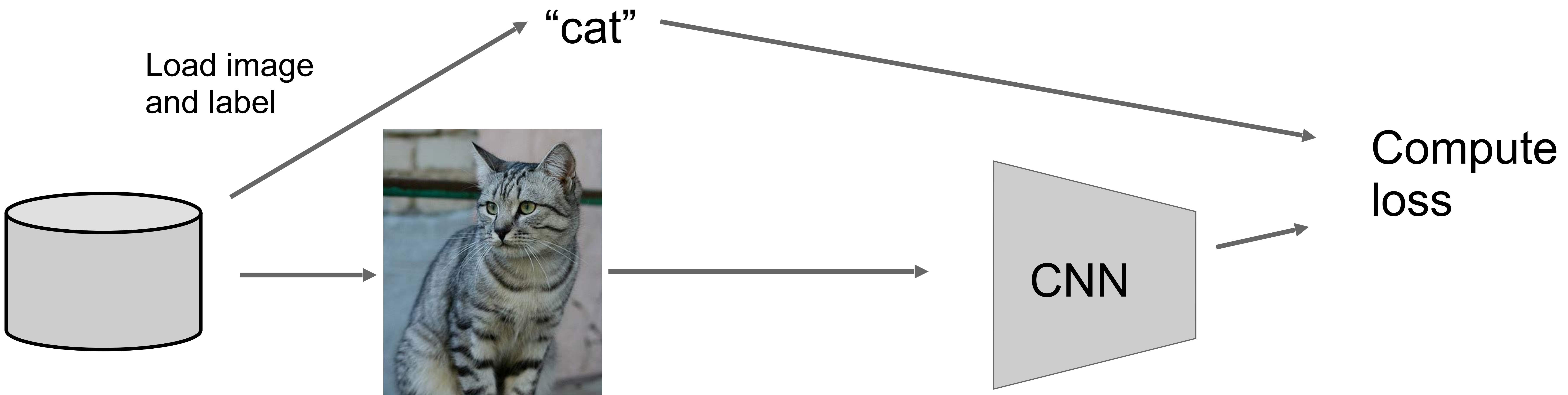
Why does this happen in the first place?



How can we have more data?

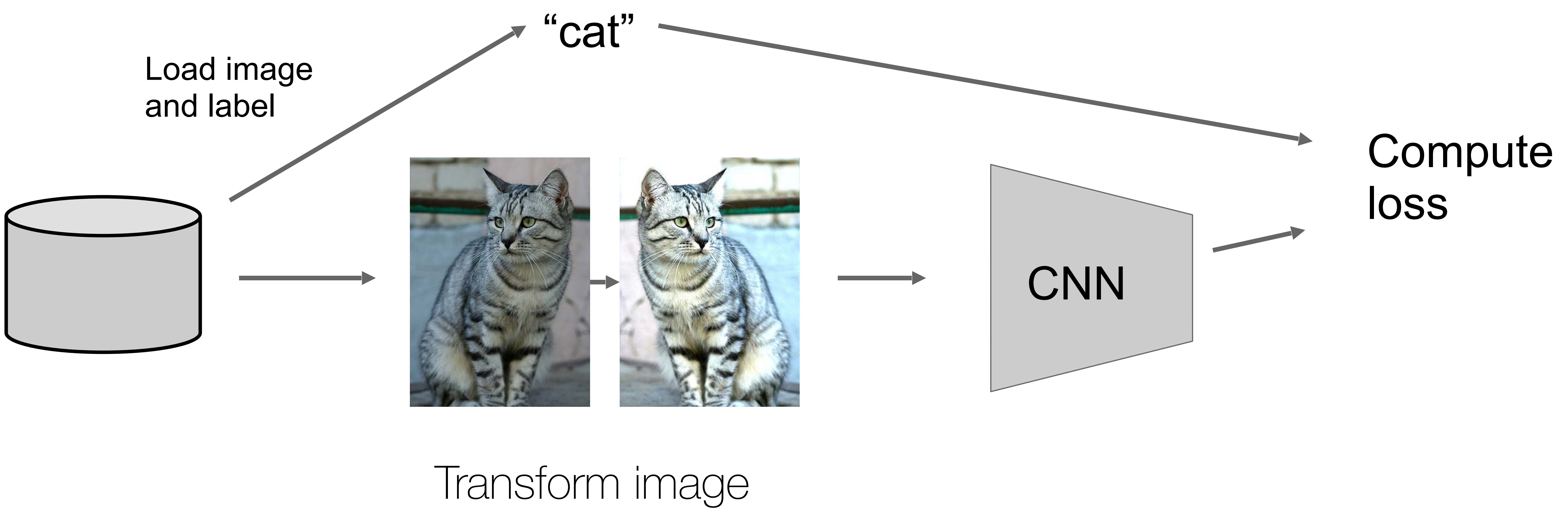
Skipped in class
(outside of scope)

Regularization: Data augmentation



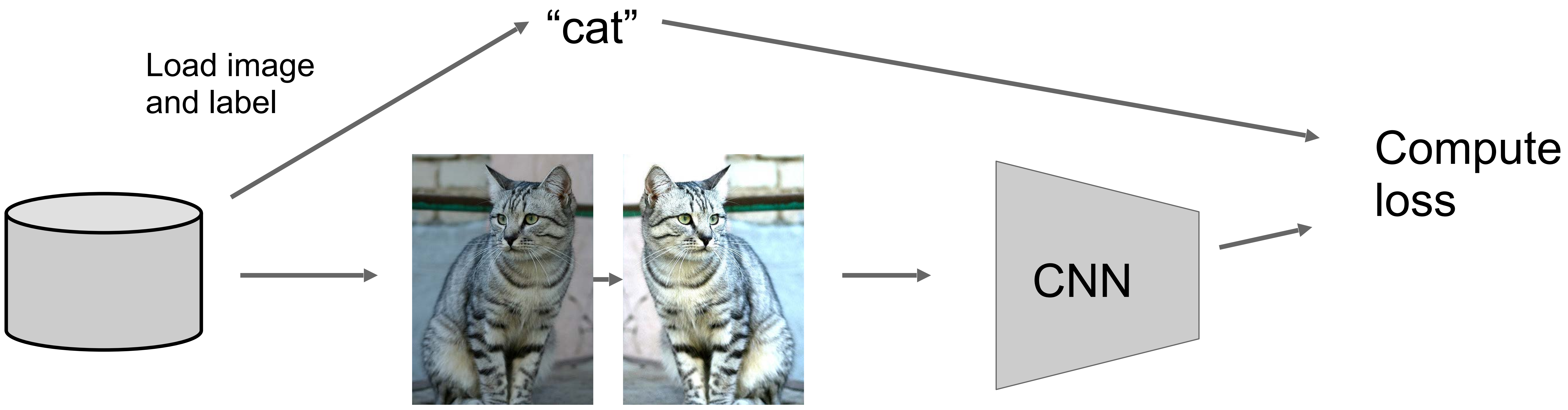
Skipped in class (outside of scope)

Regularization: Data augmentation



Skipped in class (outside of scope)

Regularization: Data augmentation

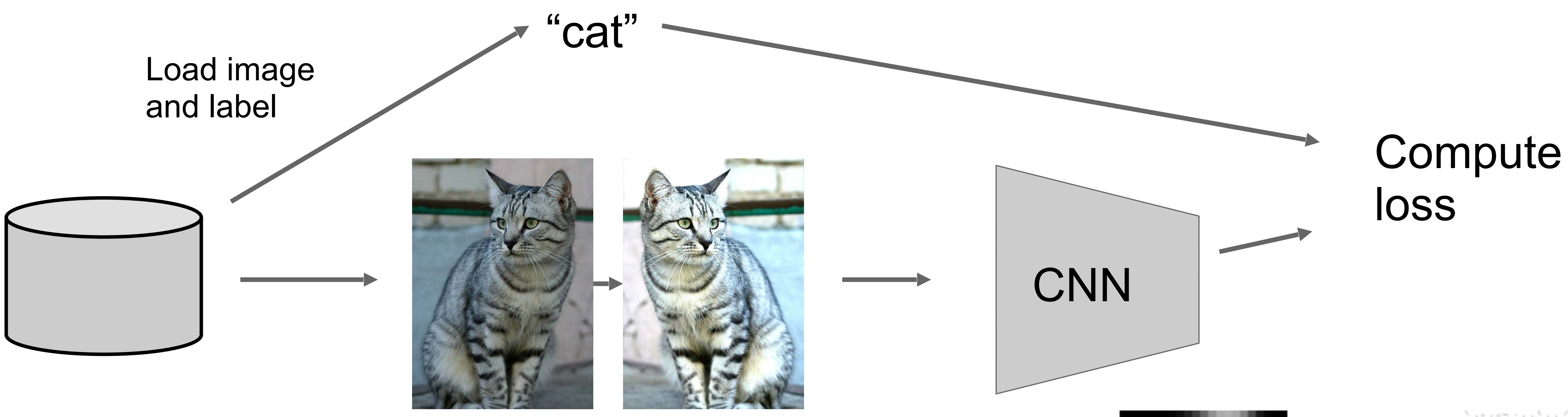


Transform image

- Horizontal / vertical flips
- Color / brightness
- Rotations / scaling
- Elastic transformation

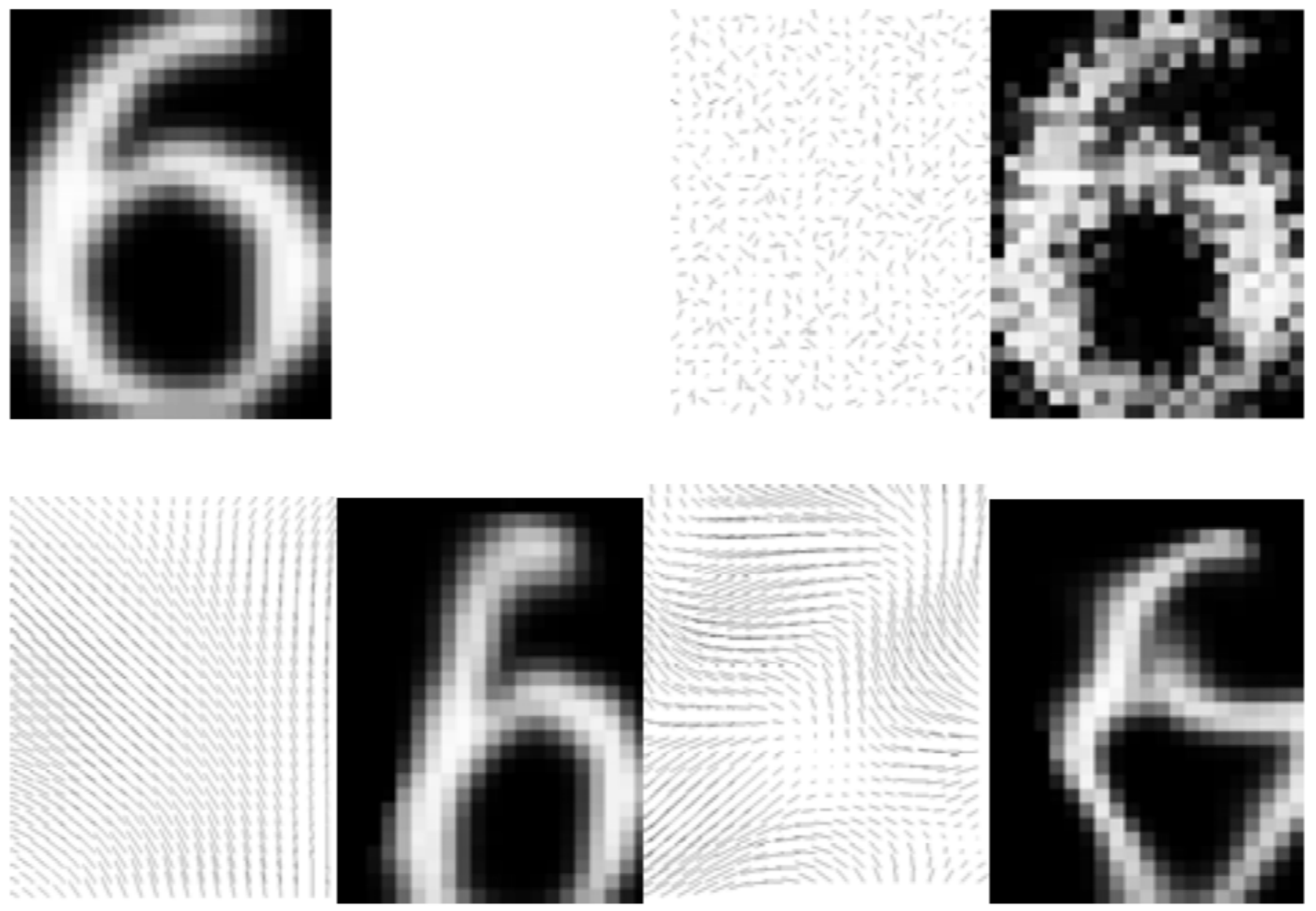
Skipped in class (outside of scope)

Regularization: Data augmentation



Transform image

- Horizontal / vertical flips
- Color / brightness
- Rotations / scaling
- Elastic transformation

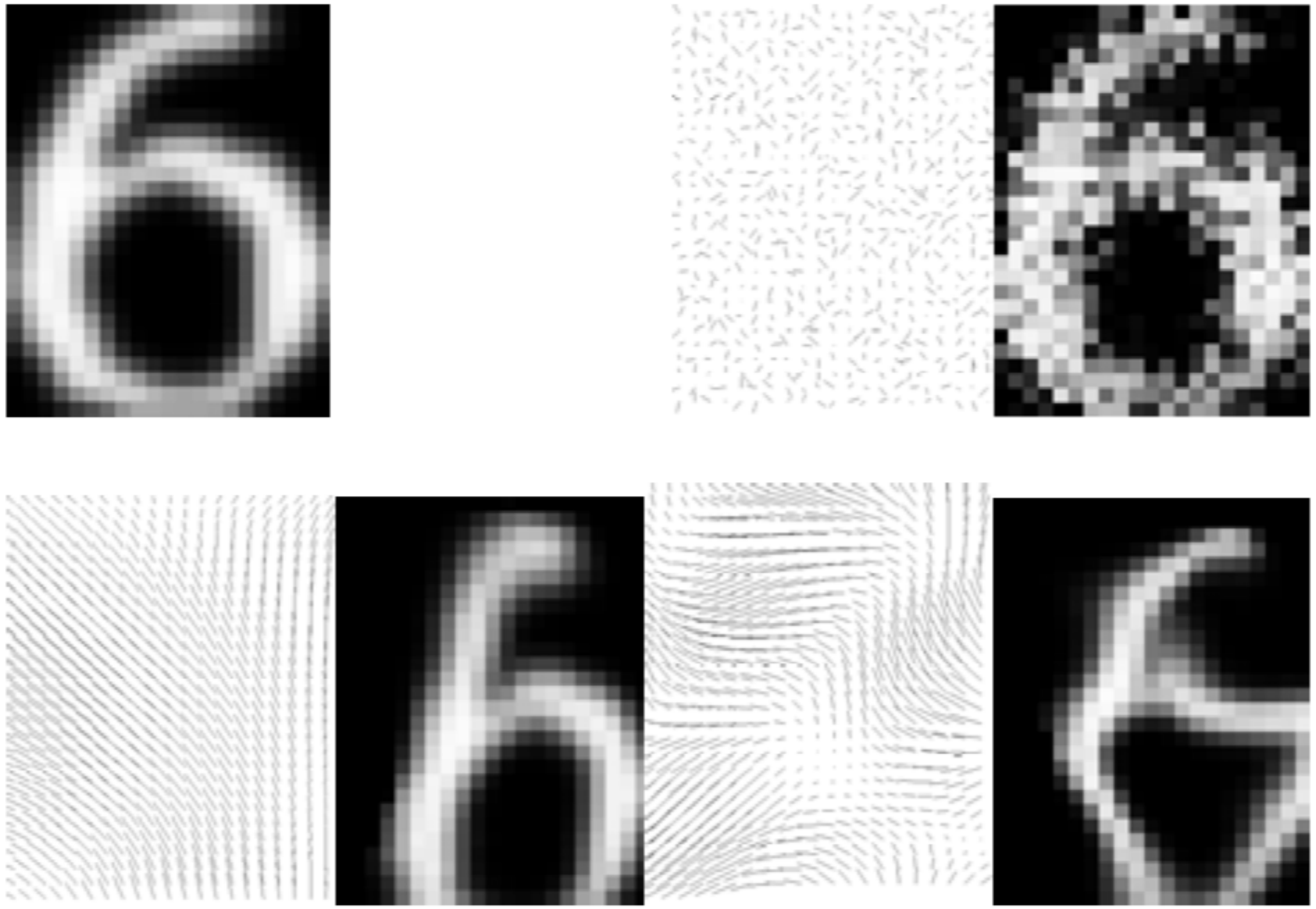


Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", ICDAR, 2003

Skipped in class (outside of scope)

Regularization: Data augmentation

Elastic deformations



1. Create random displacement field with uniform distribution

2. Smooth the displacement field with a Gaussian

Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", ICDAR, 2003

THE UNIVERSITY OF BRITISH COLUMBIA

Skipped in class (outside of scope)

Regularization: Data augmentation

Elastic deformations

Algorithm	Distortion	Error	Ref.
2 layer MLP (MSE)	affine	1.6%	[3]
SVM	affine	1.4%	[9]
Tangent dist.	affine+thick	1.1%	[3]
Lenet5 (MSE)	affine	0.8%	[3]
Boost. Lenet4 MSE	affine	0.7%	[3]
Virtual SVM	affine	0.6%	[9]
2 layer MLP (CE)	none	1.6%	this paper
2 layer MLP (CE)	affine	1.1%	this paper
2 layer MLP (MSE)	elastic	0.9%	this paper
2 layer MLP (CE)	elastic	0.7%	this paper
Simple conv (CE)	affine	0.6%	this paper
Simple conv (CE)	elastic	**0.4%**	this paper

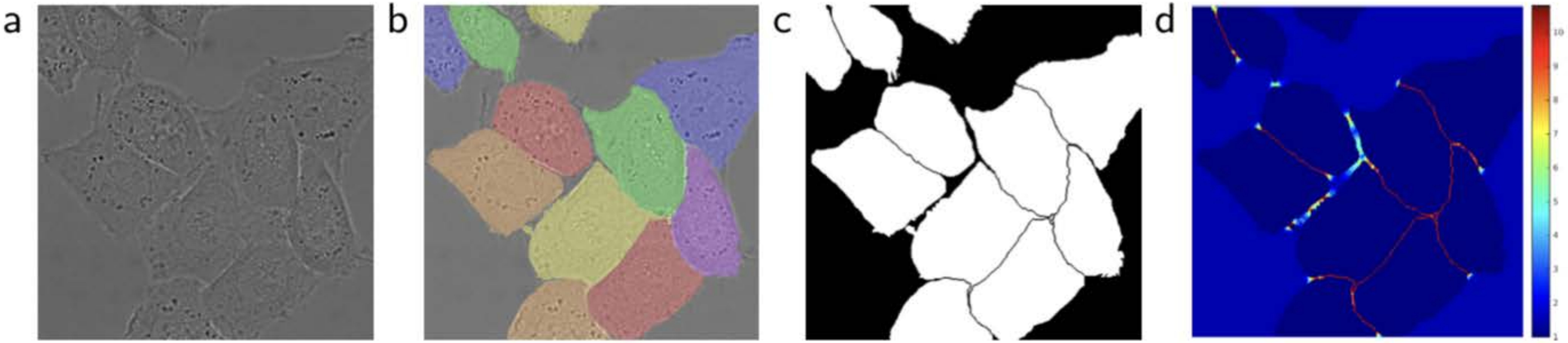
Table 1. Comparison between various algorithms.

Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", ICDAR, 2003

Skipped in class (outside of scope)

Regularization: Data augmentation

Elastic deformations



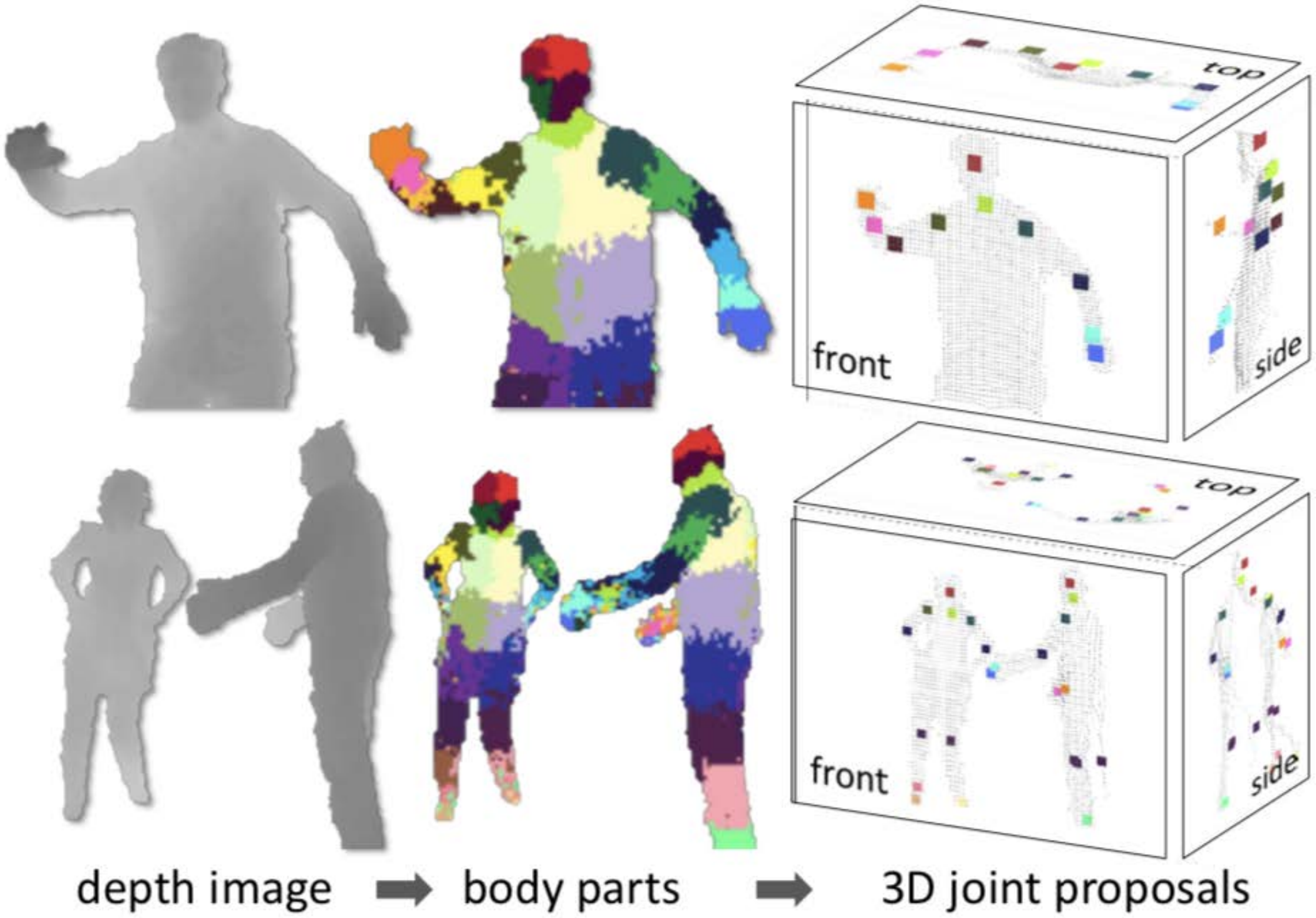
Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

Ronneberger et. al,, "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015

Skipped in class (outside of scope)

Regularization: Data augmentation

Synthetic data



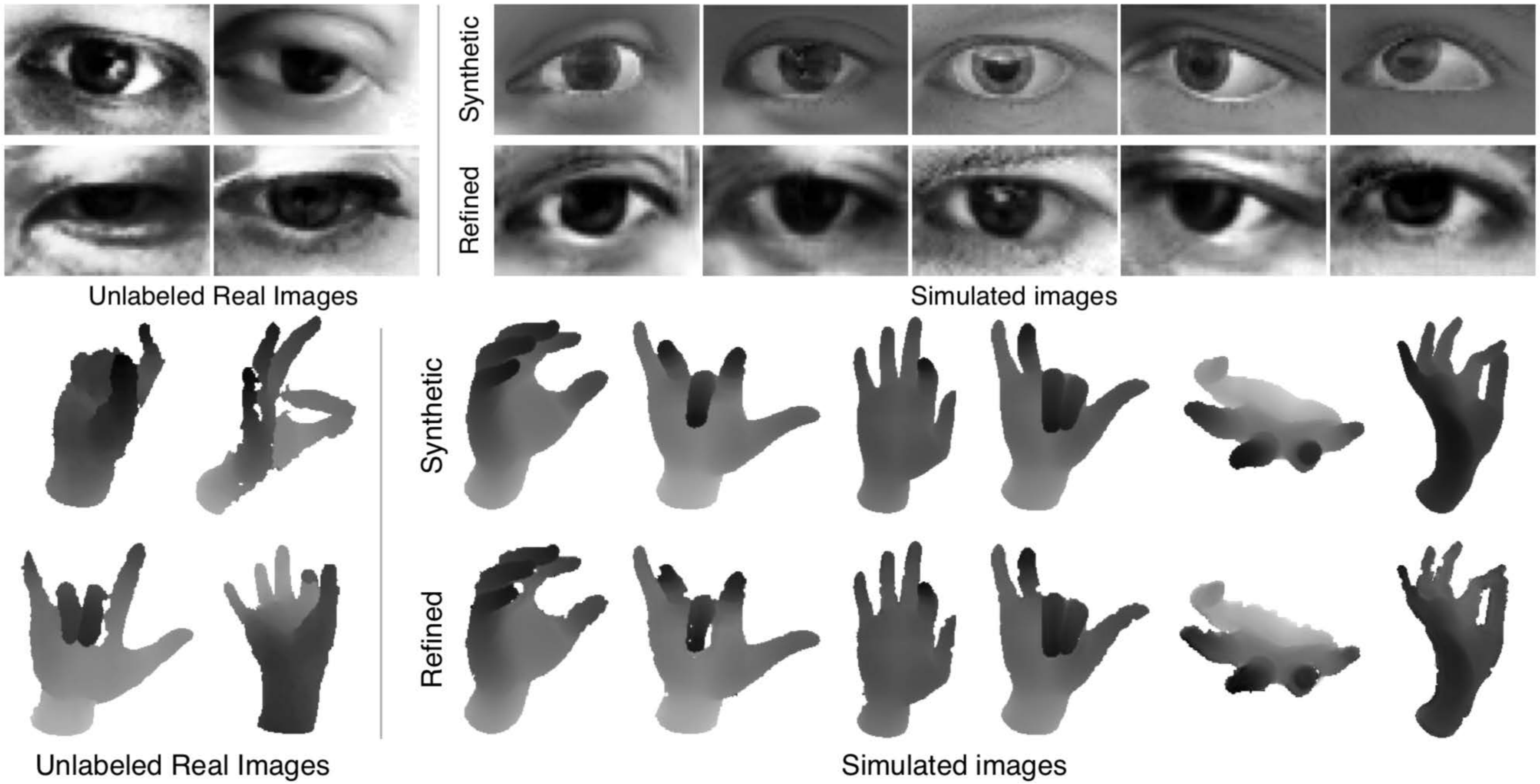
Shotton et. al,, "Real-Time Human Pose Recognition in Parts from Single Depth Images", 2011

Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

Skipped in class (outside of scope)

Regularization: Data augmentation

Synthetic data + generative models



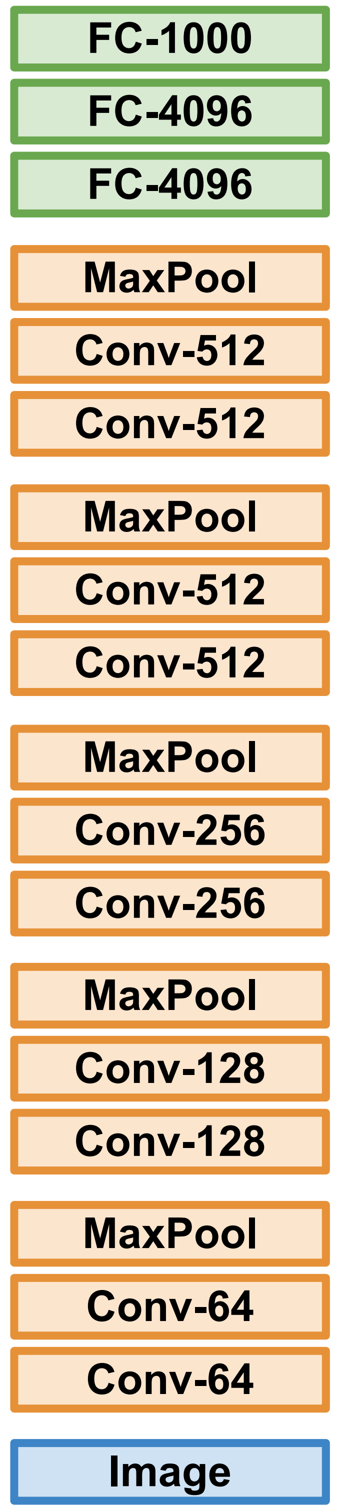
Shrivastava et. al,, "Learning from Simulated and Unsupervised Images through Adversarial Training", 2011

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Skipped in class (outside of scope)

Using pretrained networks

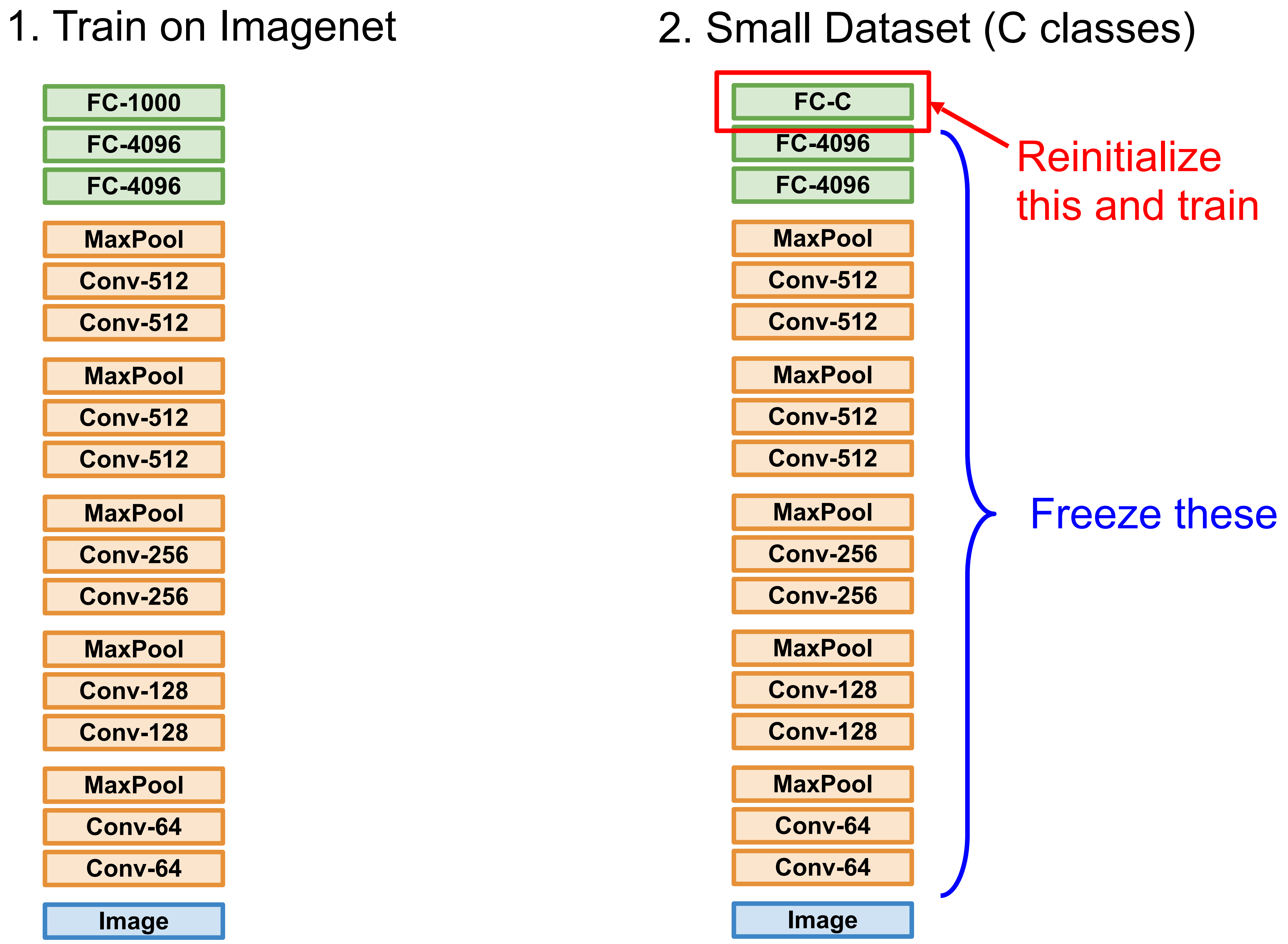
1. Train on Imagenet



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Skipped in class
(outside of scope)

Using pretrained networks

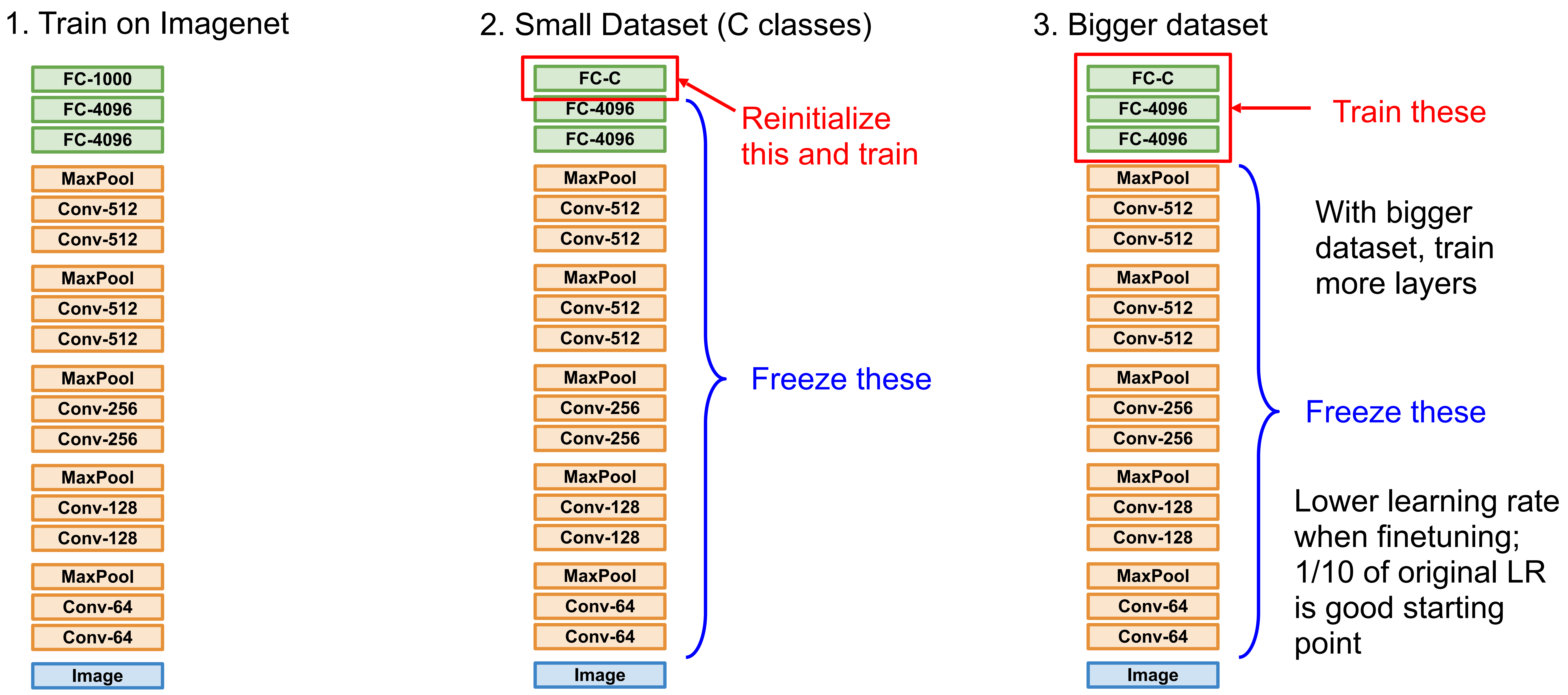


THE UNIVERSITY OF BRITISH COLUMBIA

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Skipped in class (outside of scope)

Using pretrained networks



Large generative models

Skipped in class (outside of scope)

[Video from https://twitter.com/HaiperGenAI/status/1745845670844522760

Fishing information within SD

Skipped in class (outside of scope)

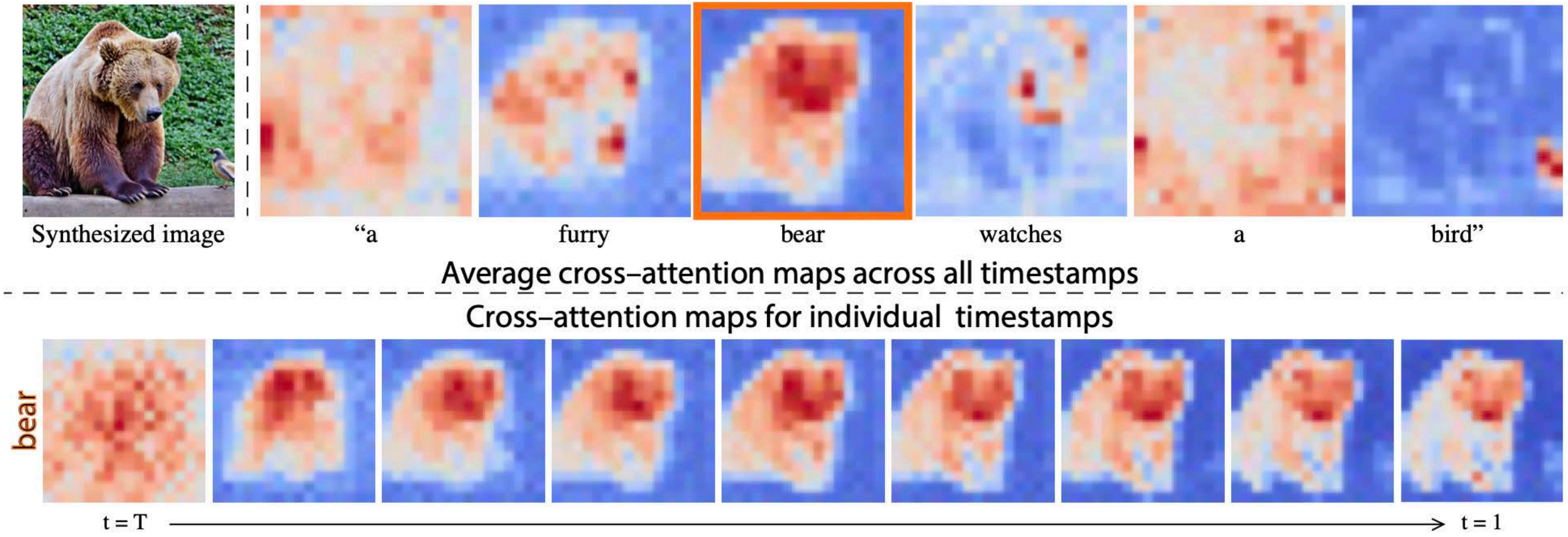
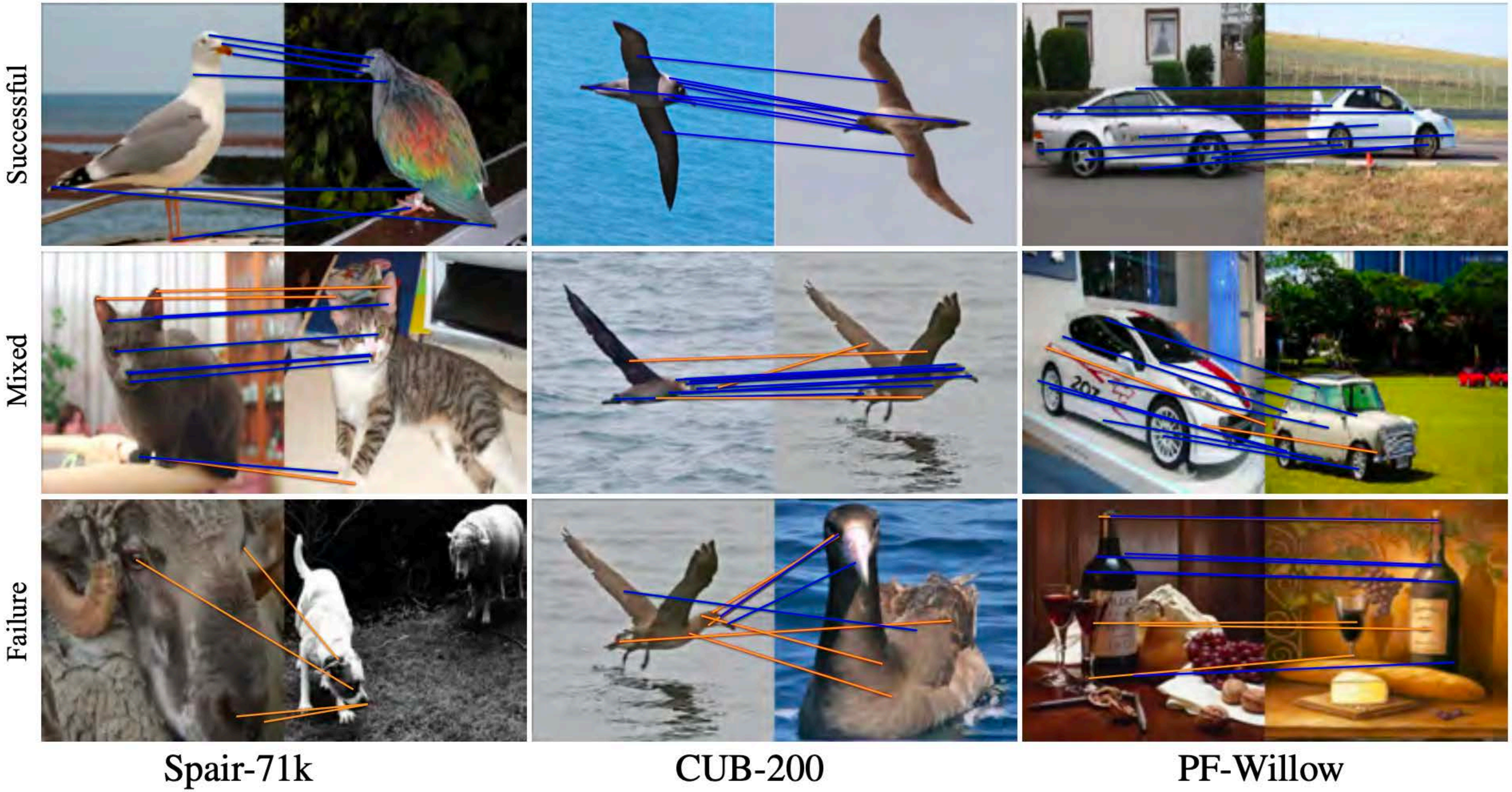


Image from [Hertz et al., ICLR, 2023]

THE UNIVERSITY OF BRITISH COLUMBIA

Correspondences from SD

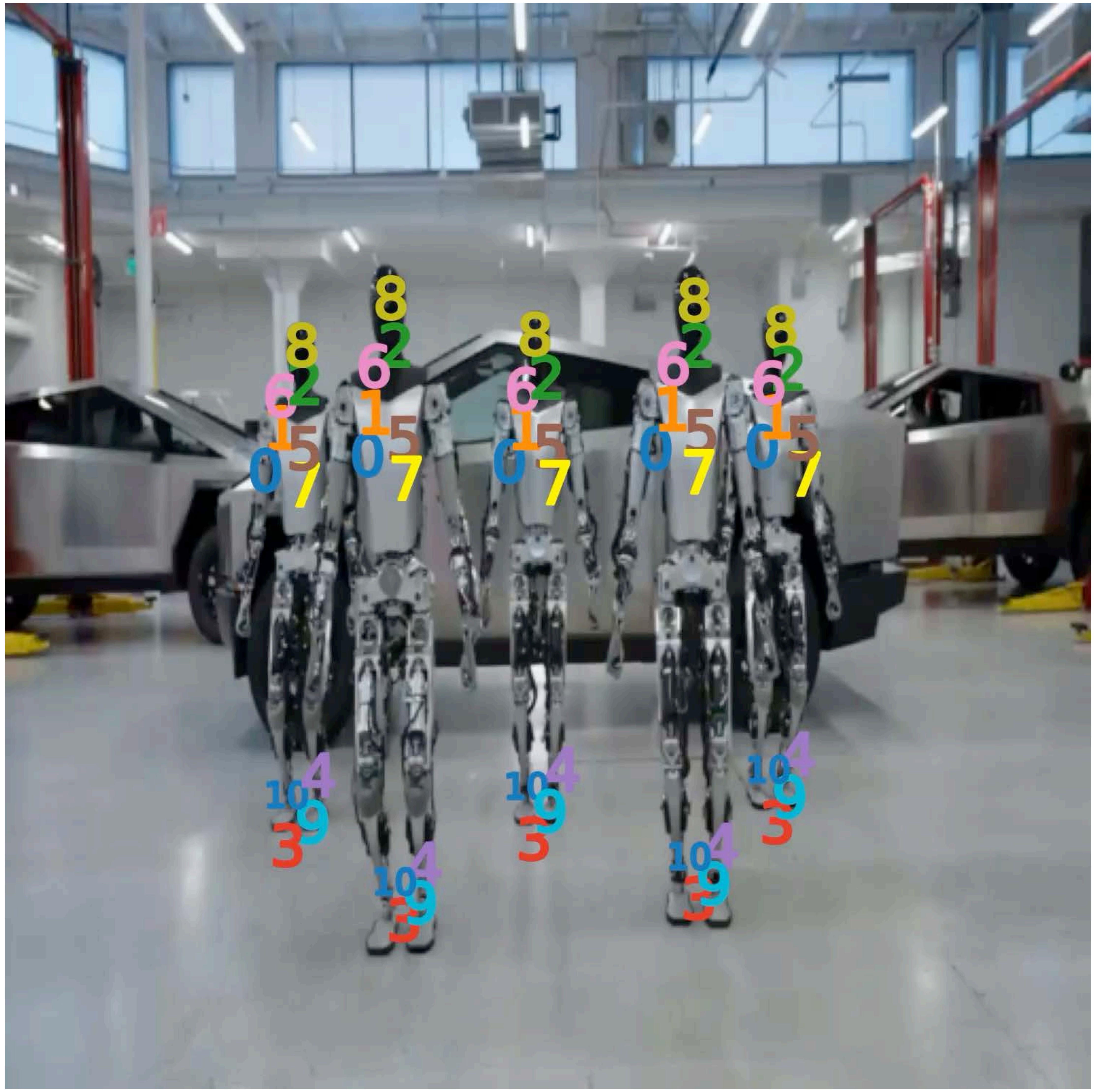
Skipped in class (outside of scope)

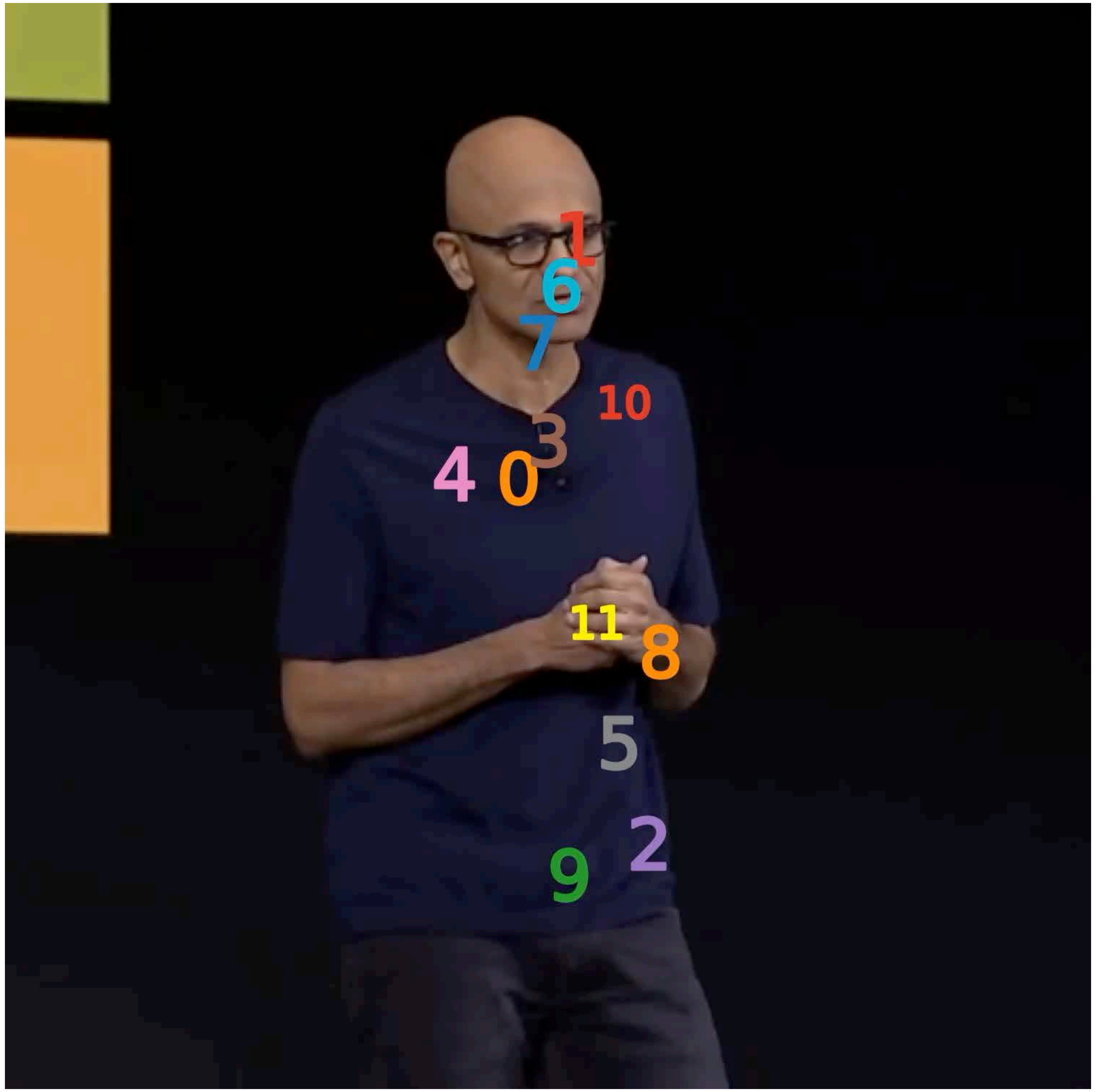


THE UNIVERSITY OF BRITISH COLUMBIA

Keypoints from SD

Skipped in class (outside of scope)

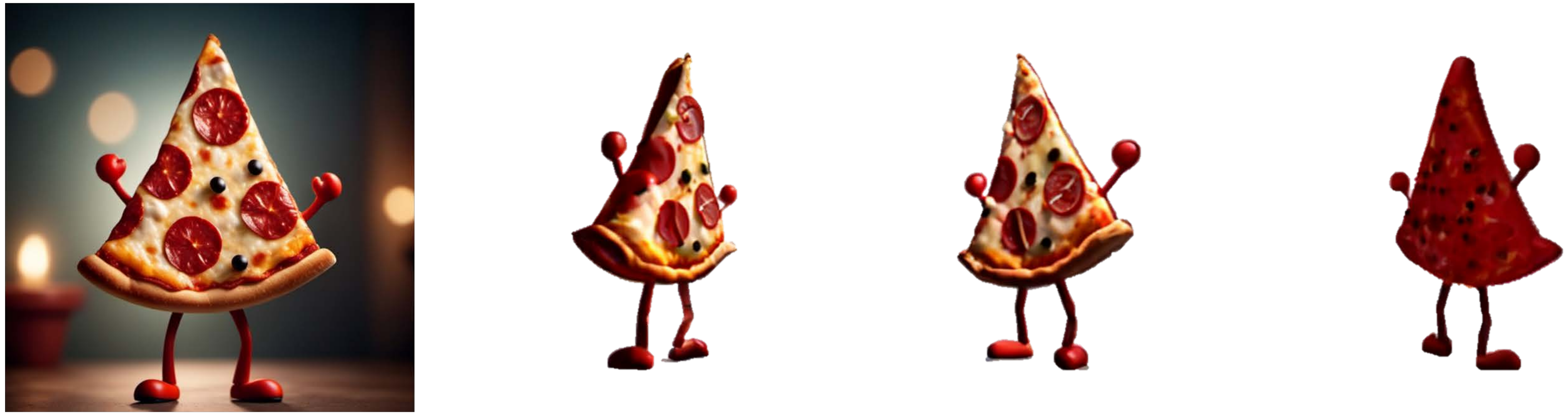




Text-to-3D from SD

Skipped in class
(outside of scope)

Input **Multi-view images**

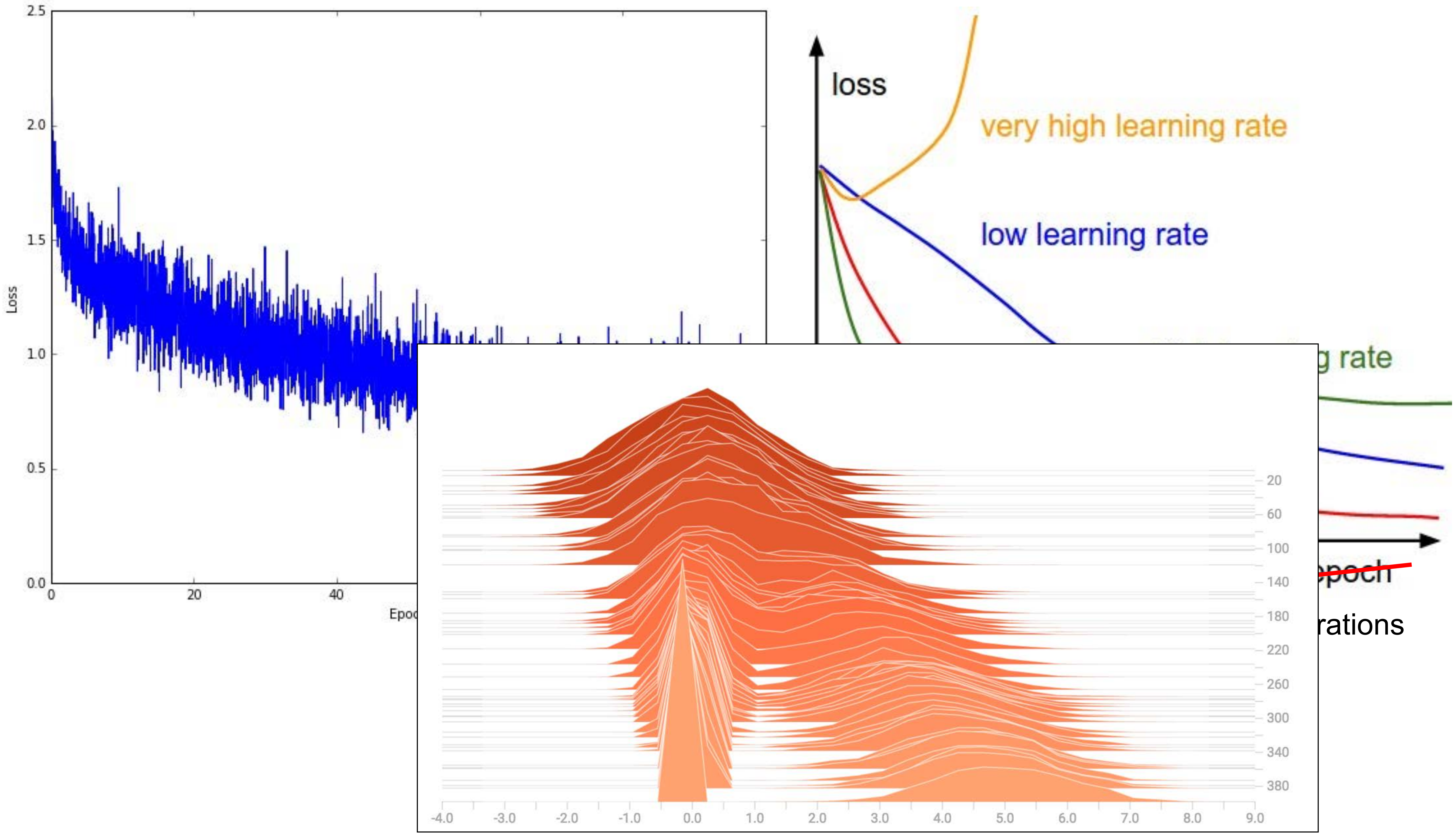


"A pepperoni pizza with arms and legs"

"A cute squirrel"

Skipped in class

Visualize VISUALIZE **VISUALIZE** (outside of scope)



Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

More on Neural Networks

Skipped in class
(outside of scope)

Lots more to learn! A good place to start is

Justin Johnson, University of Michigan, EECS 498/598, e.g.,

https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Training Neural Nets: **Clever** Hans

Skipped in class
(outside of scope)

Hans could get 89% of the math questions right

Training Neural Nets: **Clever** Hans

Skipped in class (outside of scope)

Hans could get 89% of the math questions right