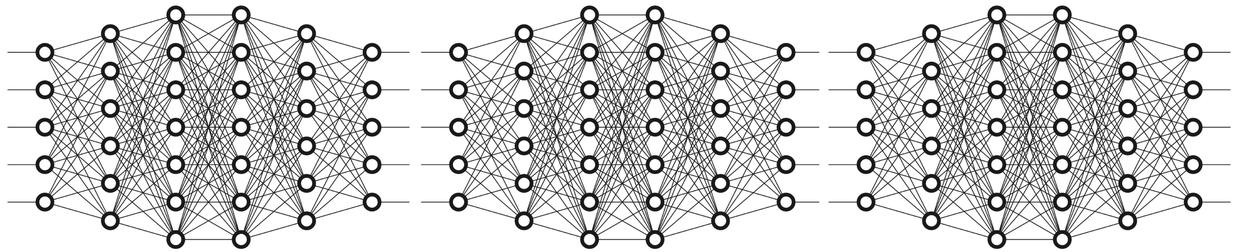
THE UNIVERSITY OF BRITISH COLUMBIA

# CPSC 425: Computer Vision



**Lecture 20:** Neural Networks 2

# **Menu** for Today

## Topics:

- **Neural Networks** part 2
- **Linear + Convolutional** layers
- **Deep nets,** AlexNet, VGG

## Readings:

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

## Reminders:

- **Assignment 6**: Deep Learning is now available
- **Final**: December 14th

Many slides from this lecture are from

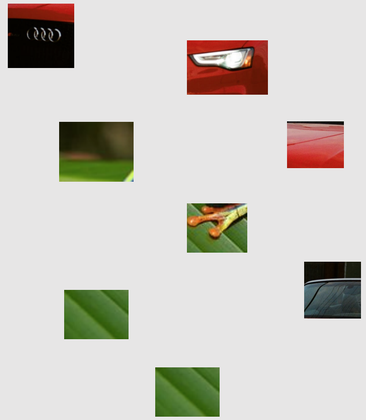
**Justin Johnson**, University of Michigan, EECS 498/598

https://web.eecs.umich.edu/~justincj/

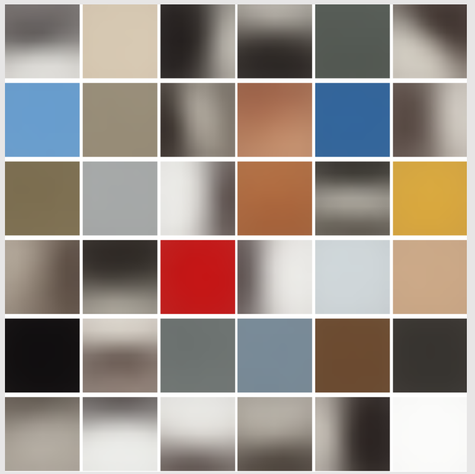
# Image Features: Bag of Words (Data-Driven!)

**Step 1: Build codebook**

Extract random patches



Cluster patches to form “codebook” of “visual words”

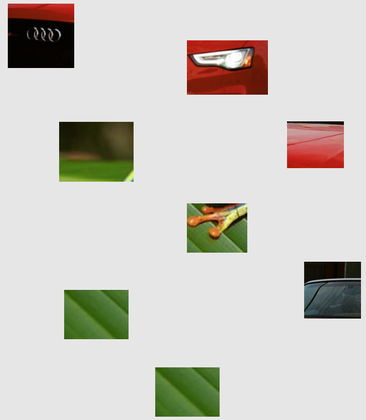


Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005

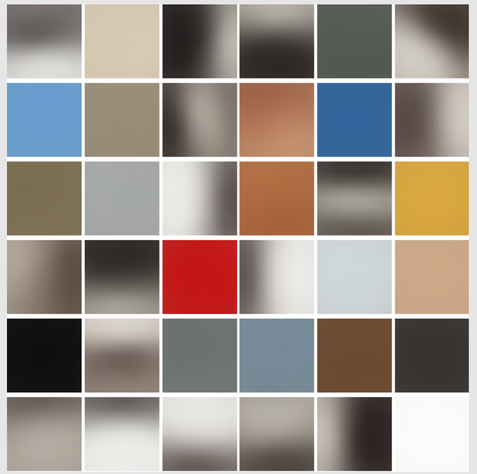
# Image Features: Bag of Words (Data-Driven!)

**Step 1: Build codebook**

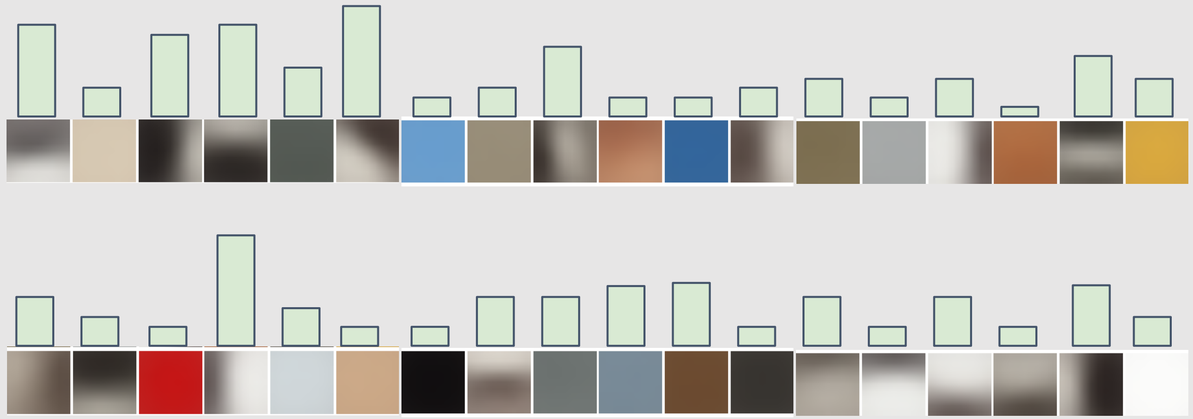
Extract random patches



Cluster patches to form “codebook” of “visual words”



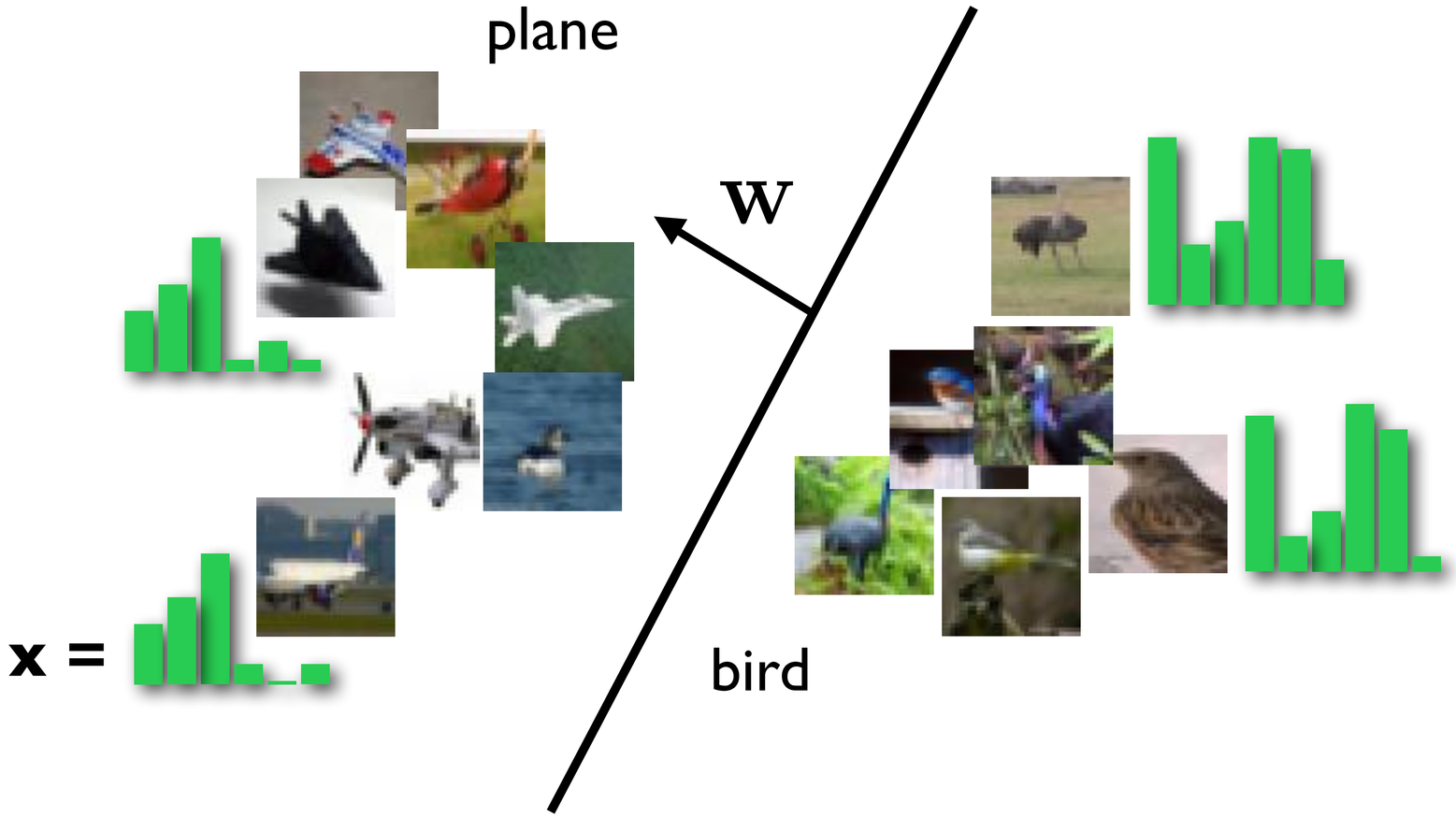
**Step 2: Encode images**



Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005

# Classify Visual Word Histograms

- e.g., bird vs plane classifier as linear classifier in space of histograms
- Histograms of visual word frequencies = vector **x**, linear classifier **w**



# Example: Winner of 2011 ImageNet challenge

Low-level feature extraction ≈ 10k patches per image

- SIFT: 128-dim
- color: 96-dim

} reduced to 64-dim with PCA

FV extraction and compression:

- N=1,024 Gaussians, R=4 regions ⇨ 520K dim x 2
- compression: G=8, b=1 bit per dimension

One-vs-all SVM learning with SGD

Late fusion of SIFT and color systems

F. Perronnin, J. Sánchez, "Compressed Fisher vectors for LSVRC", PASCAL VOC / ImageNet workshop, ICCV, 2011.

# IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:

1,000 object classes

1,431,167 images

**Output:**

Scale

T-shirt

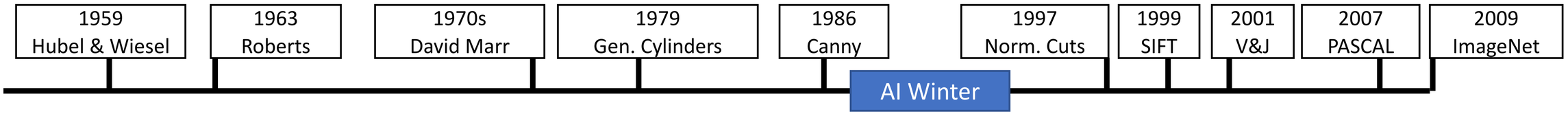
Steel drum

Drumstick

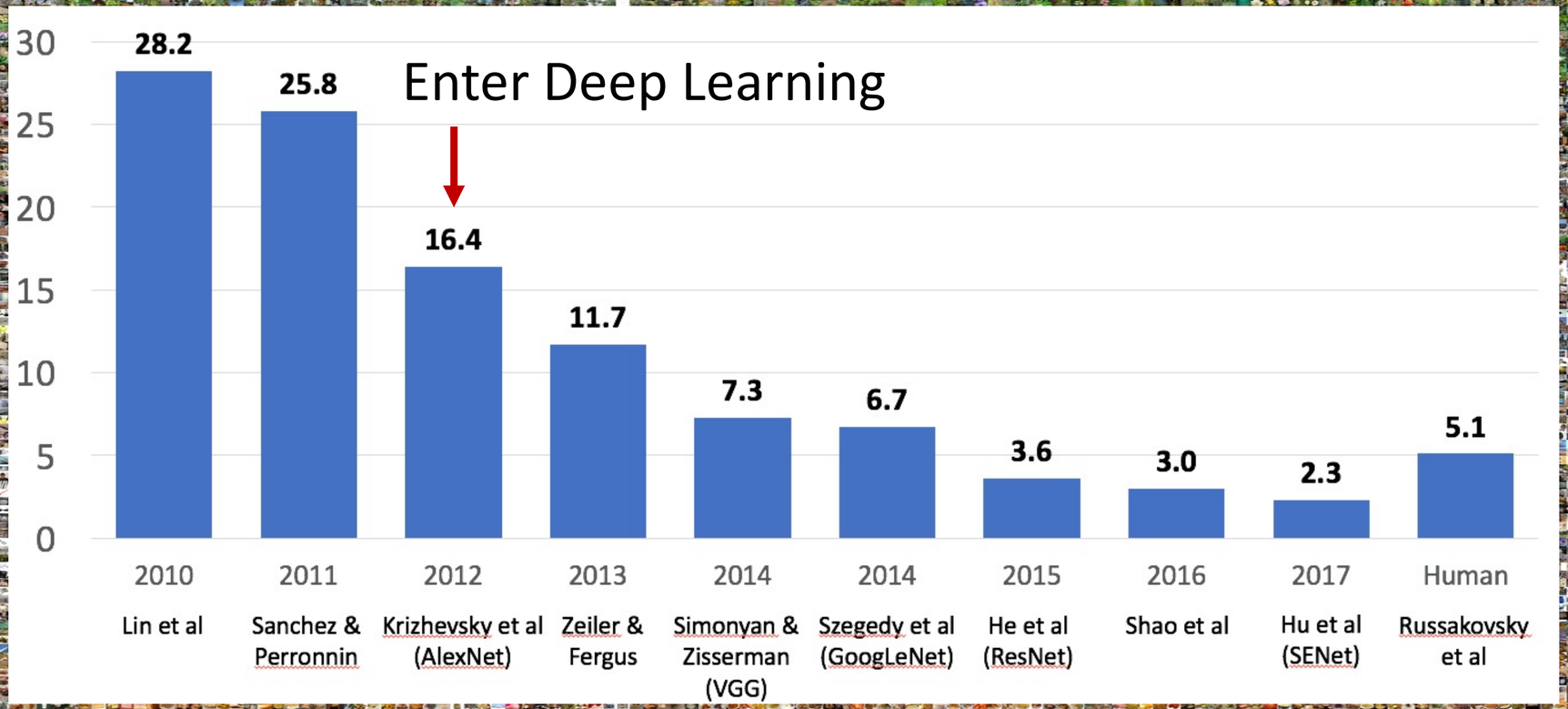
Mud turtle

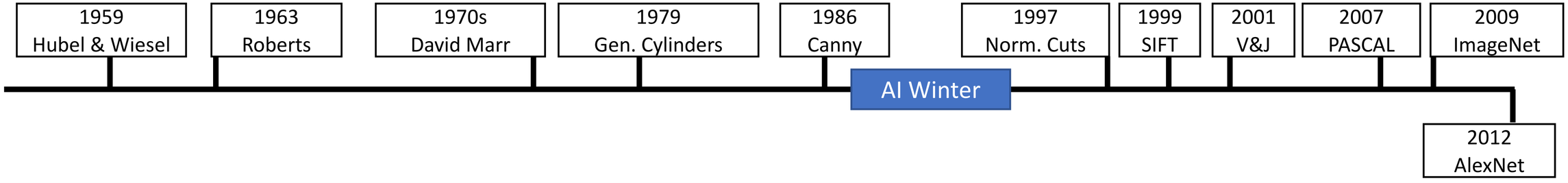
Deng et al, 2009

Russakovsky et al. IJCV 2015

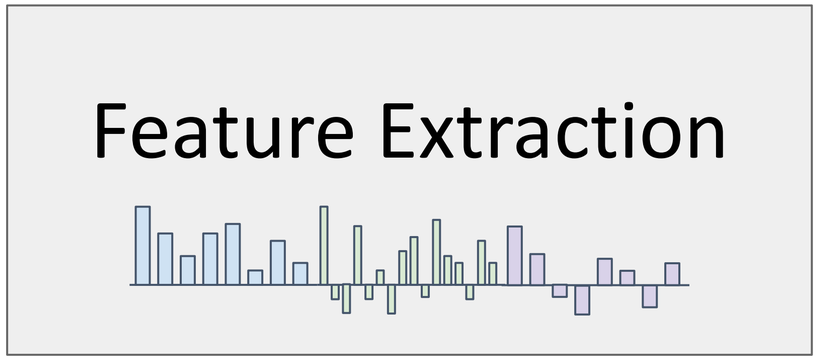


# IMAGENET Large Scale Visual Recognition Challenge



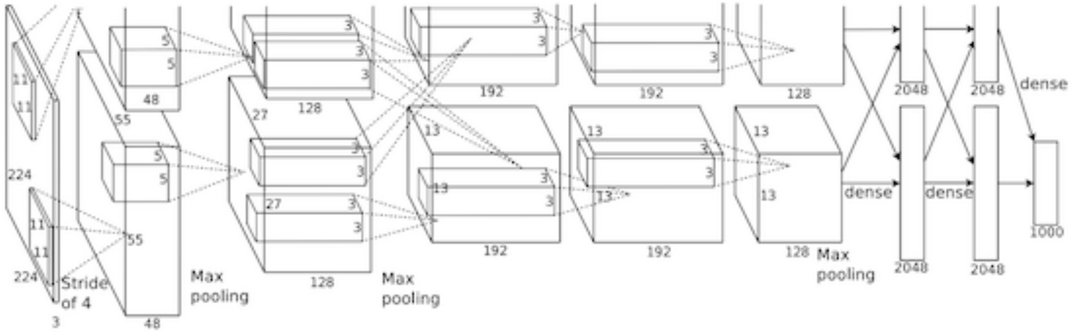
# Image Features vs Neural Networks




f

**10** numbers giving scores for classes

training

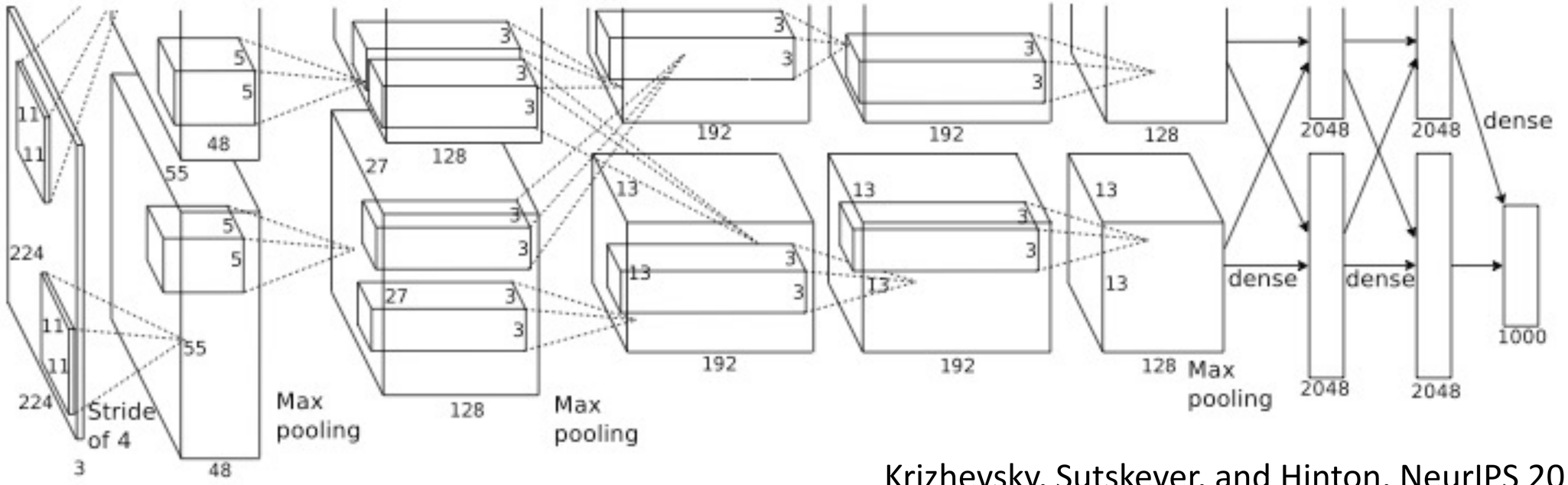



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

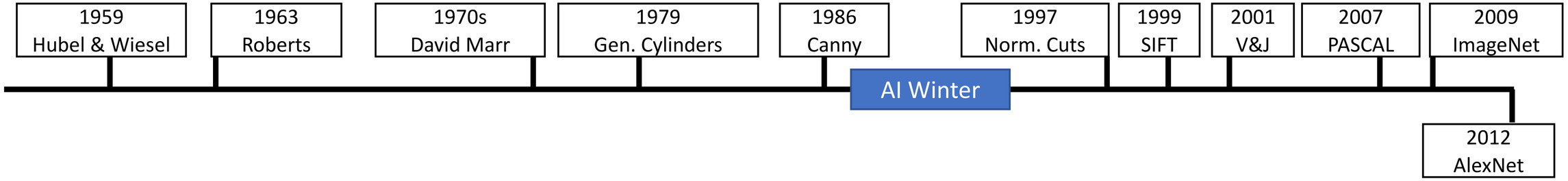
**10** numbers giving scores for classes

training

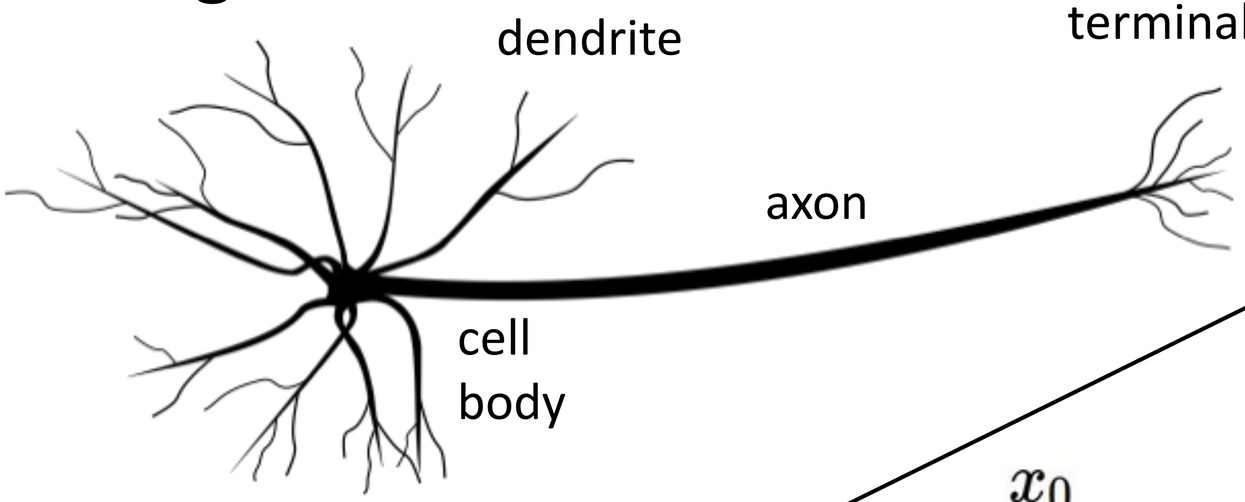
# AlexNet: Deep Learning Goes Mainstream



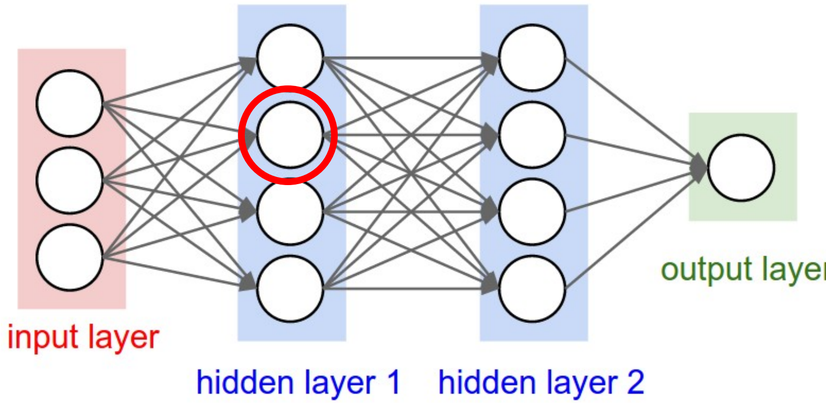

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

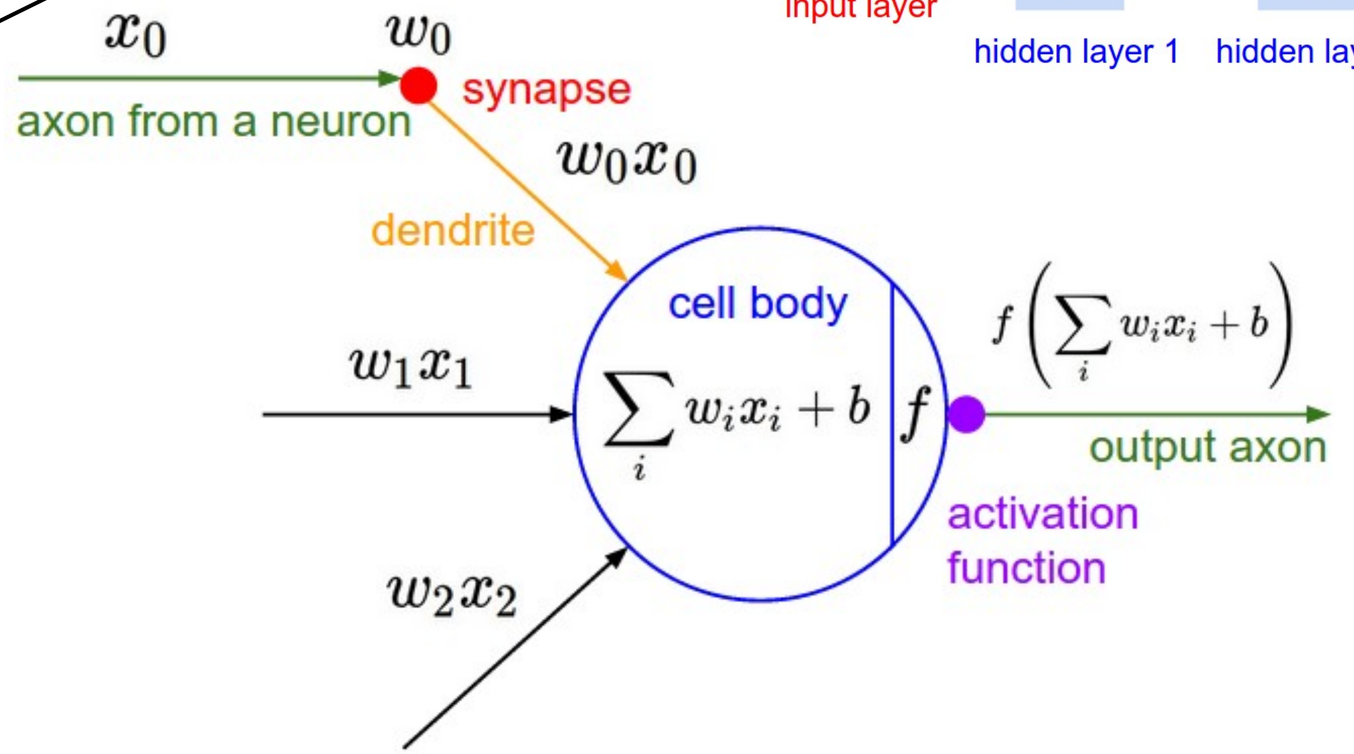


# Biological Neuron




# Artificial Neuron



# Neural Networks

(**Before**) Linear score function:

$$f = Wx$$

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

# Neural Networks

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

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

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

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

# Neural Networks

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

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

or 3-layer Neural Network

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

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

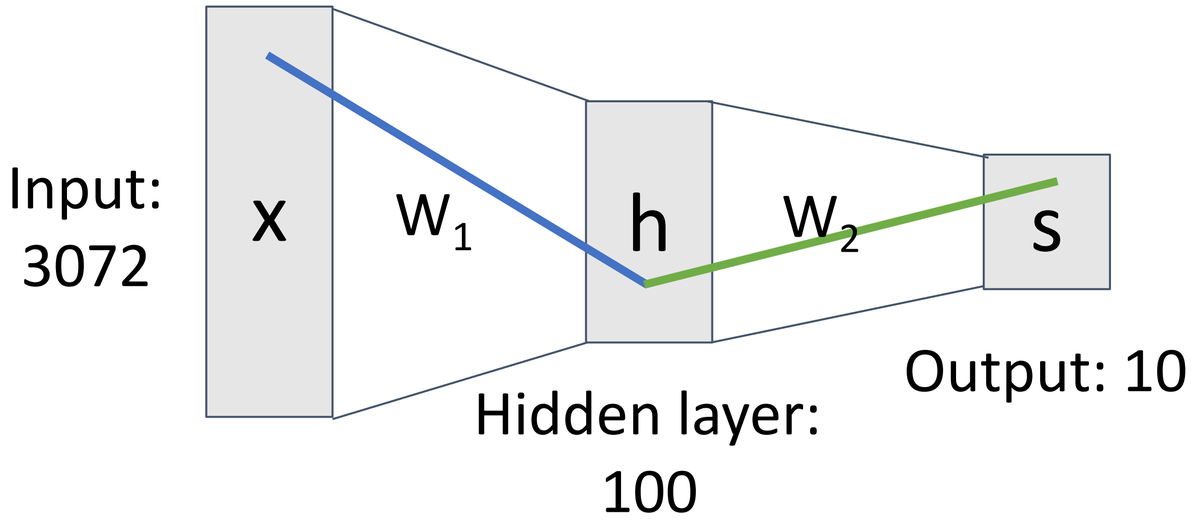
(In practice we will usually add a learnable bias at each layer as well)

# Neural Networks

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

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

Element (i, j) of $W_1$ gives the effect on $h_i$ from $x_j$




Element (i, j) of $W_2$ gives the effect on $s_i$ from $h_j$

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

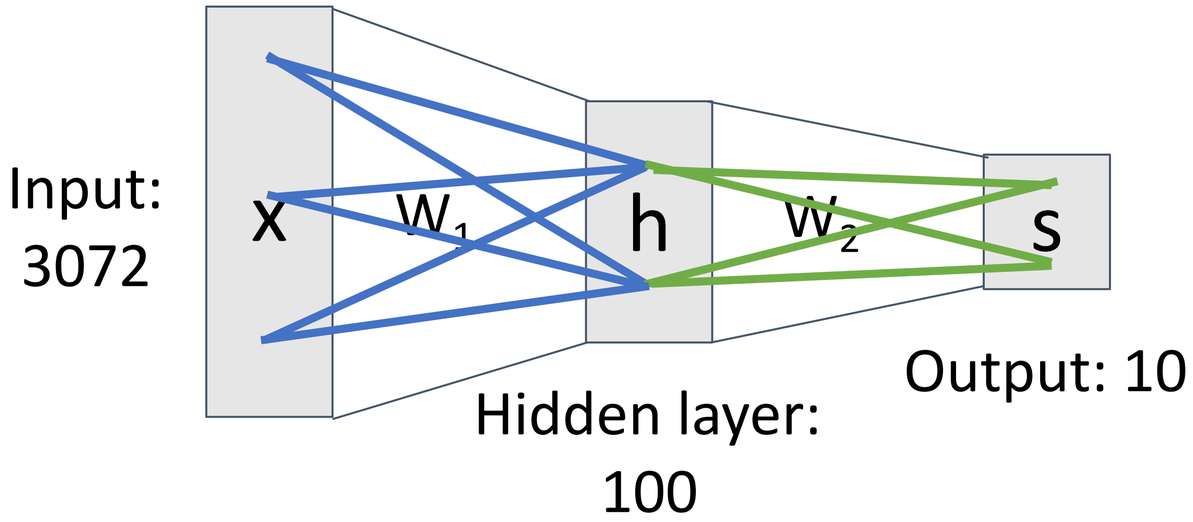
# Neural Networks

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

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

Element (i, j) of $W_1$ gives the effect on $h_i$ from $x_j$

All elements of x affect all elements of h




Element (i, j) of $W_2$ gives the effect on $s_i$ from $h_j$

All elements of h affect all elements of s

Fully-connected neural network

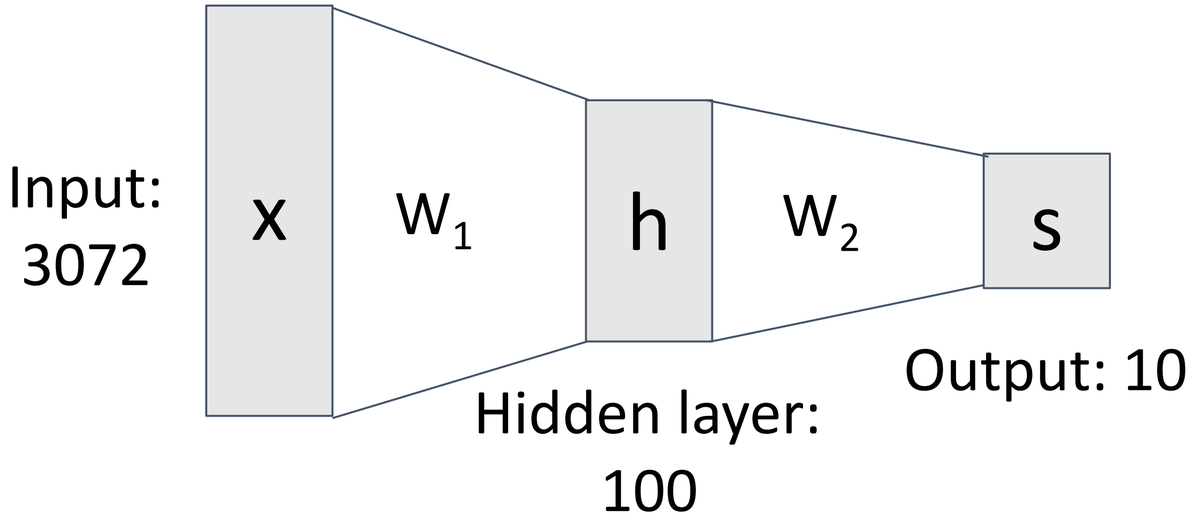
Also “Multi-Layer Perceptron” (MLP)

# Neural Networks

Linear classifier: One template per class


(**Before**) Linear score function:

(**Now**) 2-layer Neural Network



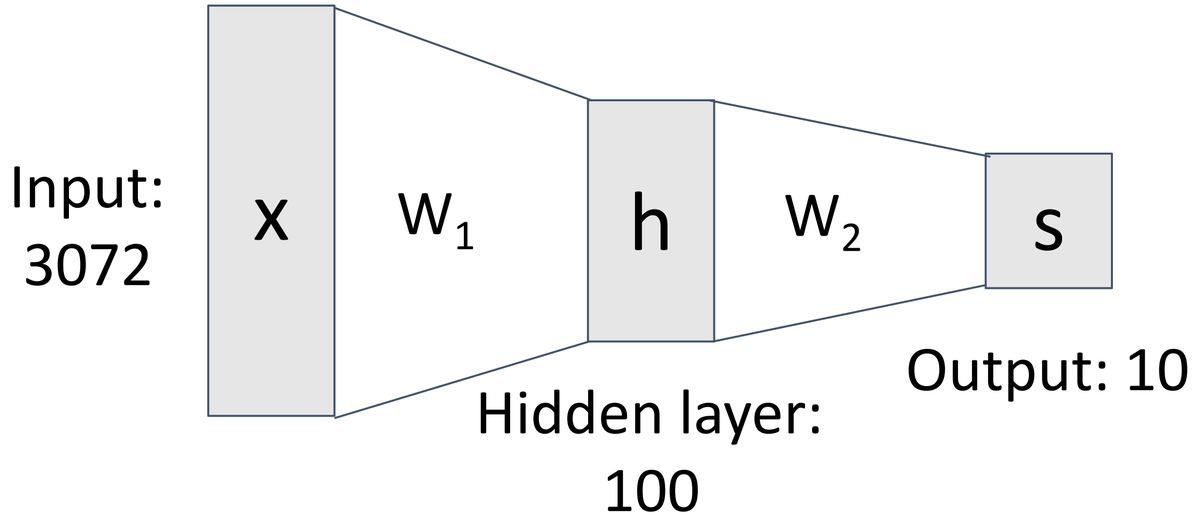

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

Neural net: first layer is bank of templates;
Second layer recombines templates

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network



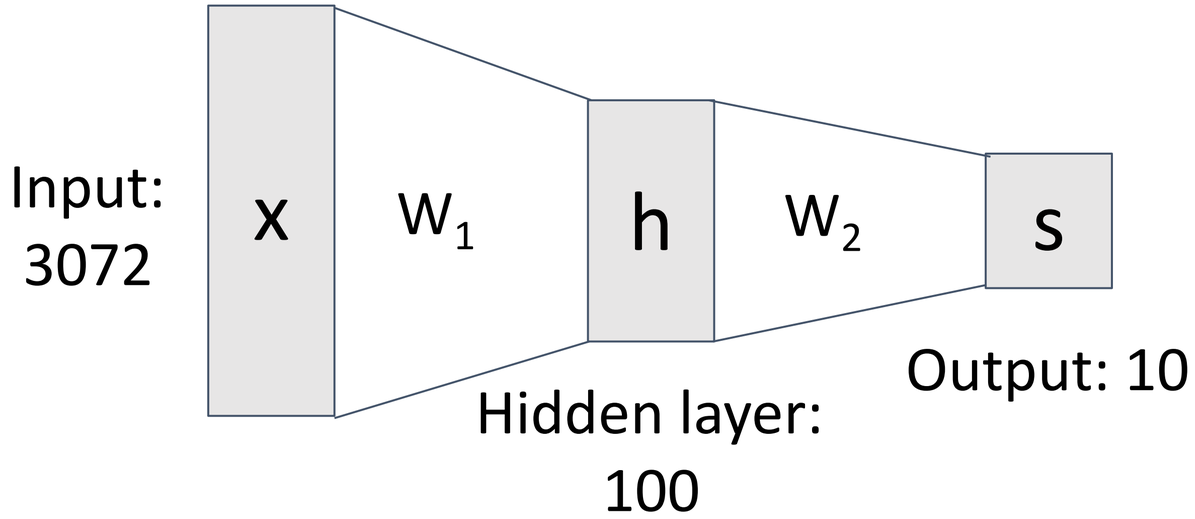

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

Can use different templates to cover multiple modes of a class!

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network



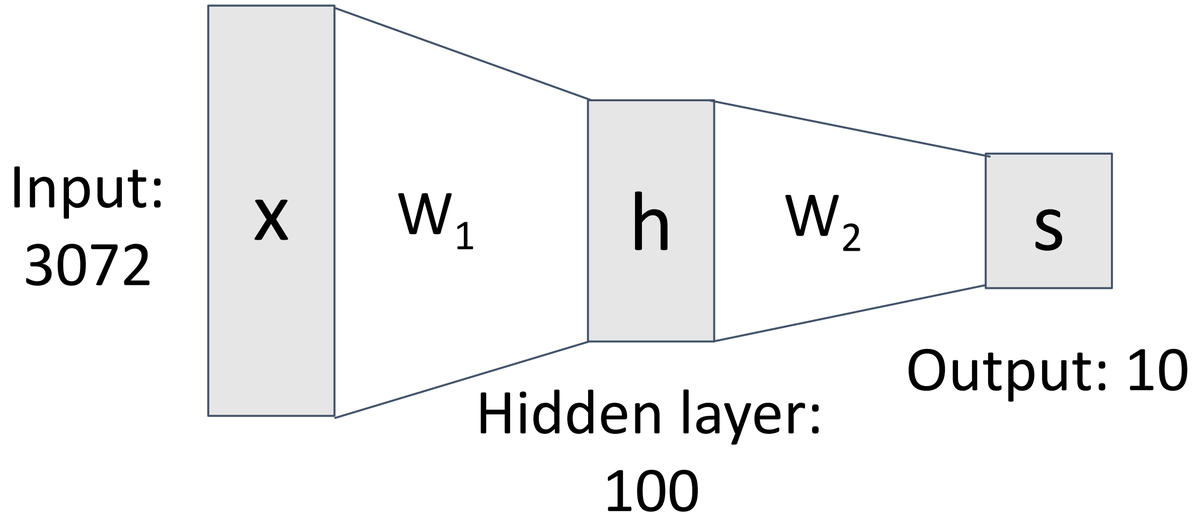

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

"Distributed representation":
Most templates not interpretable!

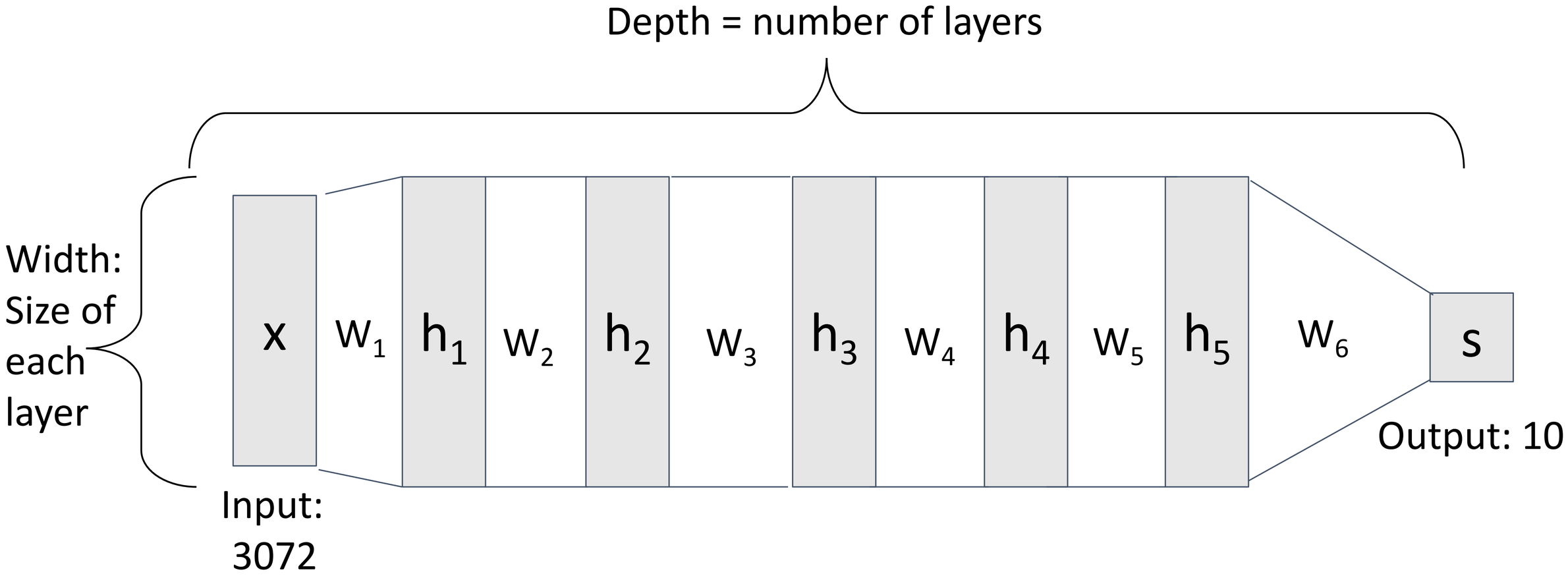
(**Before**) Linear score function:

(**Now**) 2-layer Neural Network




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

# Deep Neural Networks




$$s = W_6 \max(0, W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x))))))$$

# (Bad) Idea: 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

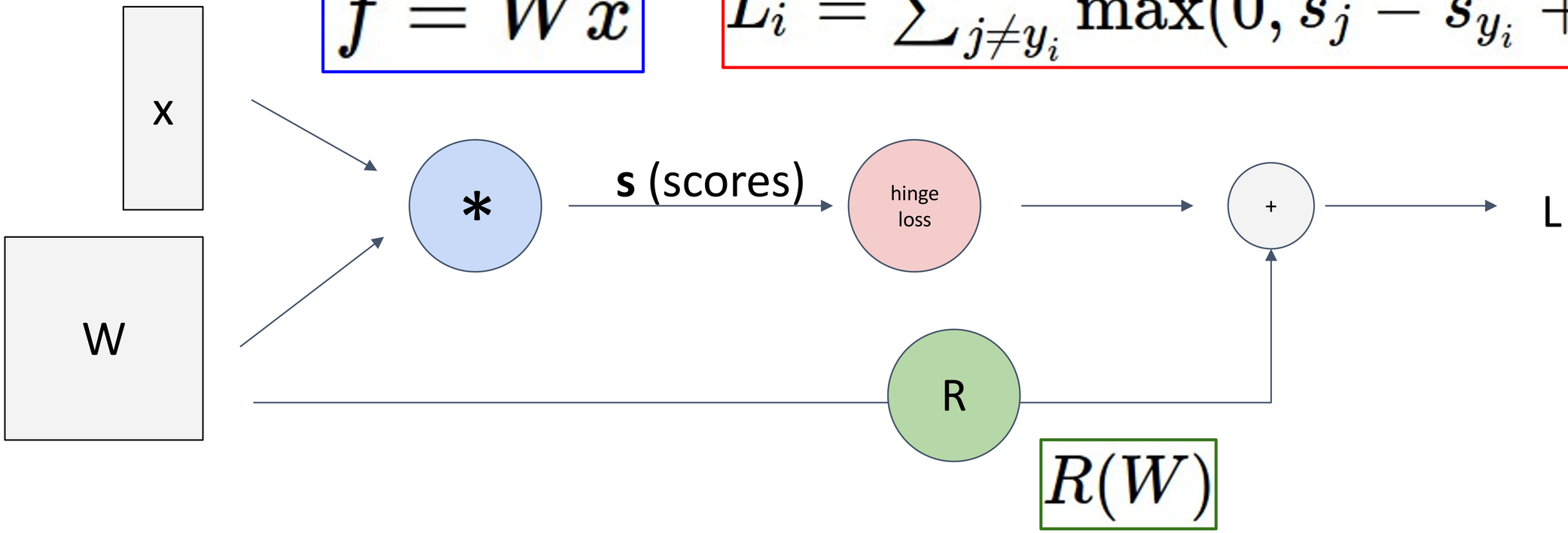
**Problem**: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

**Problem**: Not feasible for very complex models!

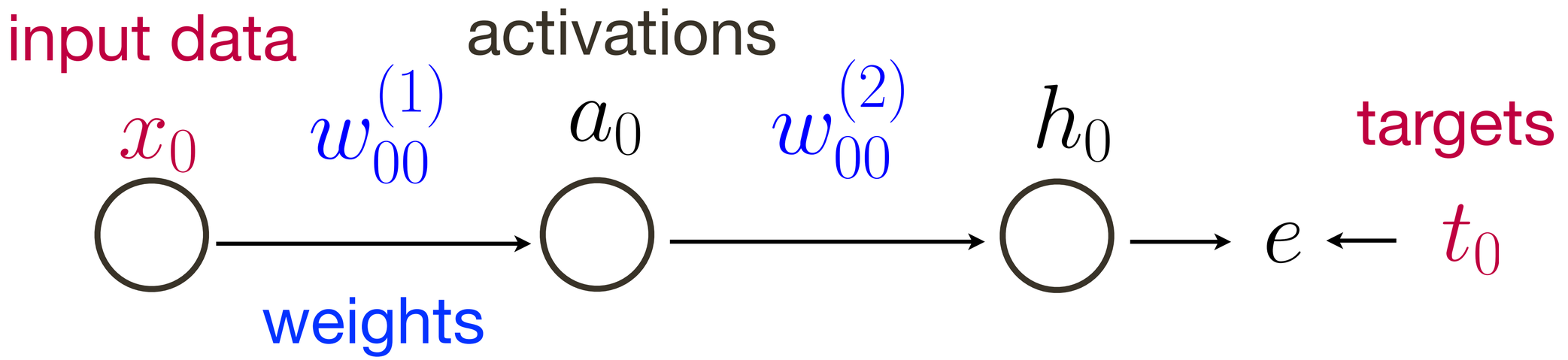
# Better Idea: **Computational Graphs**

$$f = Wx$$

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

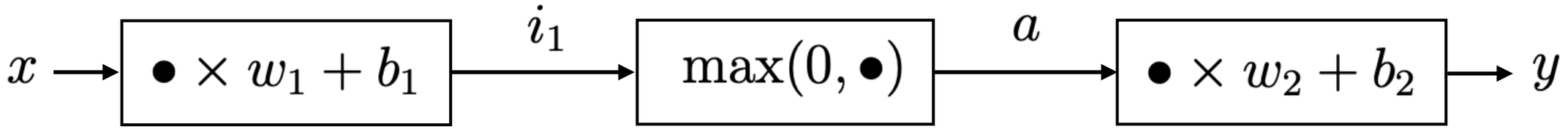


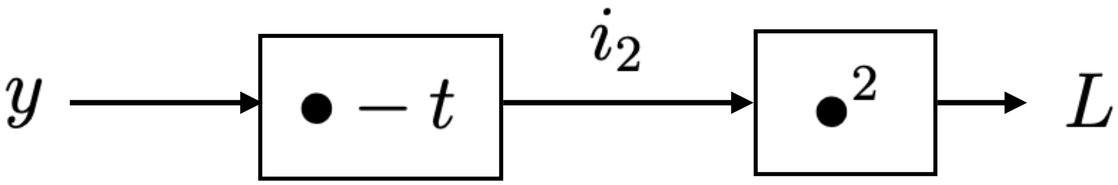
# 2-Layer **Neural** Network — 1 hidden, 1 input/output



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

$$y = w_2(\max(0, w_1 x + b_1)) + b_2 \qquad L = (y - t)^2$$



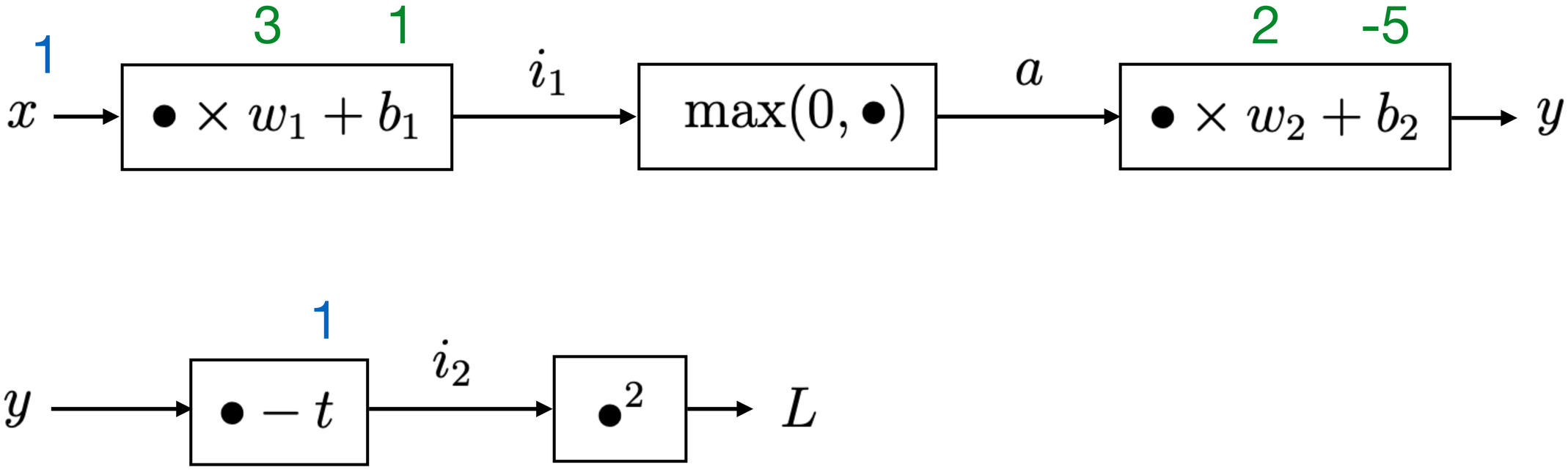




Alternative: build a **computational graph** to apply the **chain rule**

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

Input + /target

Initial weights

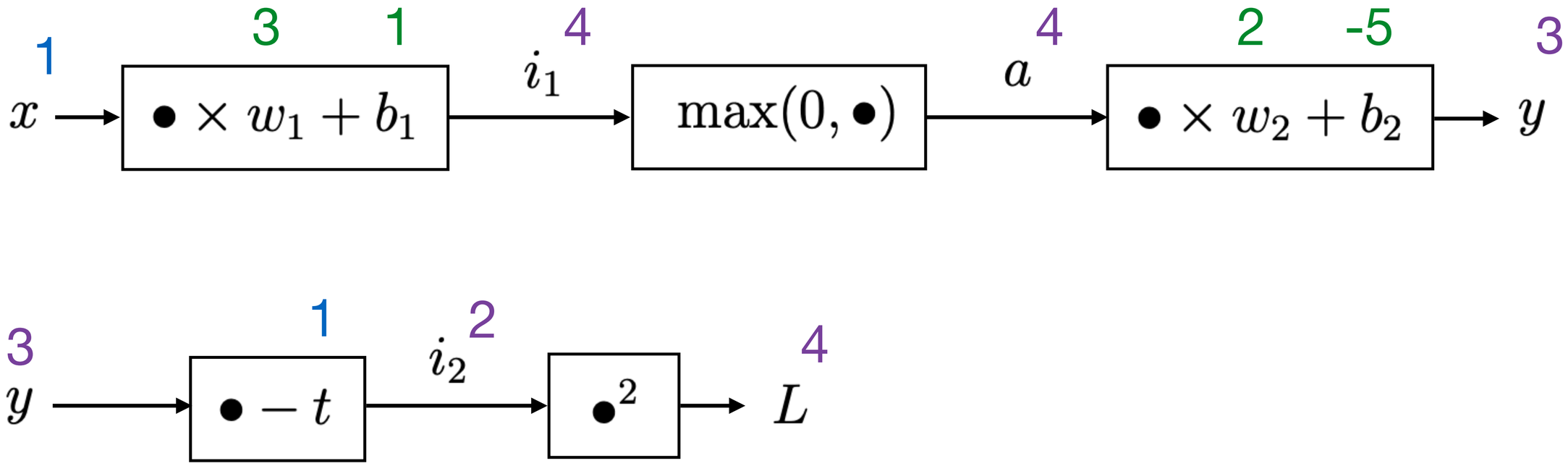


# 2-Layer **Neural** Network — 1 hidden, 1 input/output

Input + /target

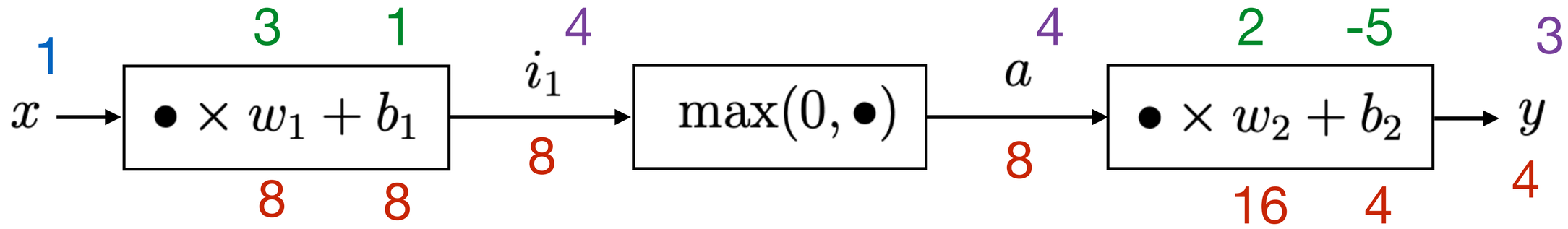
Initial weights

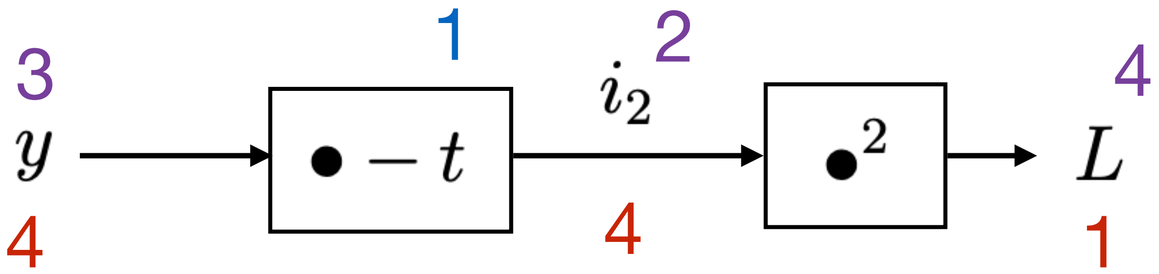
Forward pass



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

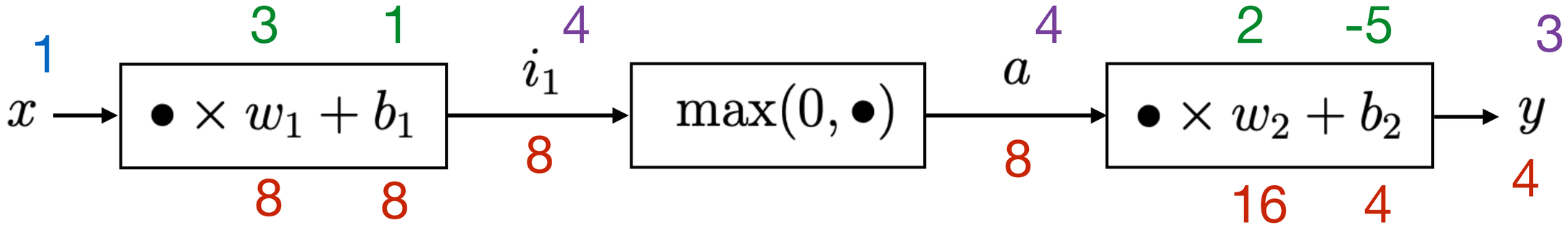
Input + /target | Initial weights | Forward pass | Backward pass $= \frac{\partial L}{\partial \bullet}$

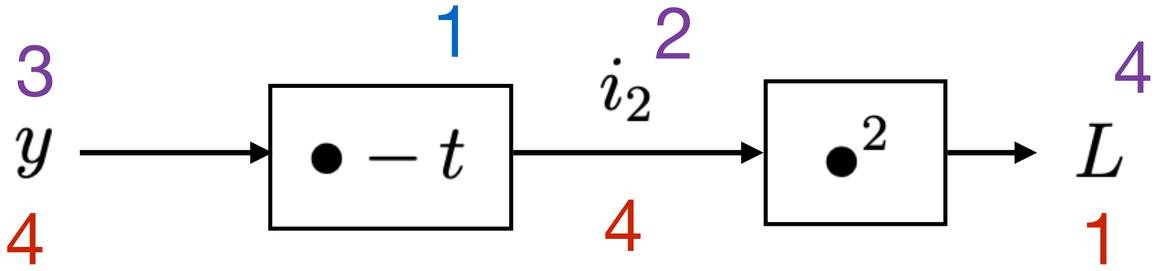


# 2-Layer **Neural** Network — 1 hidden, 1 input/output

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

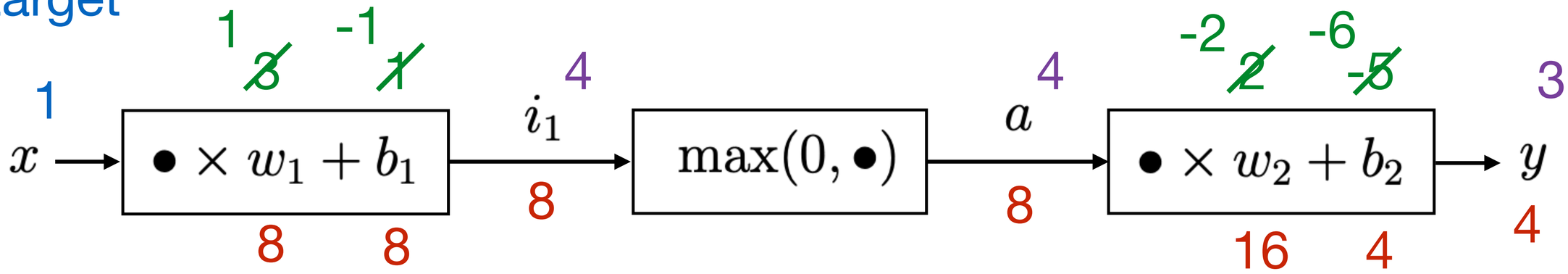


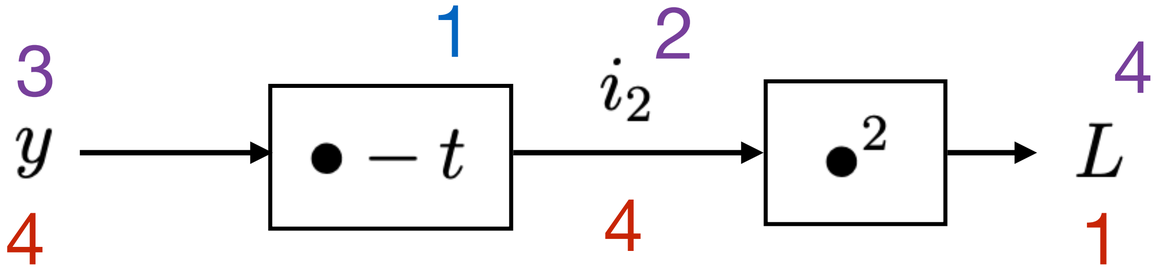




Gradient = $\begin{bmatrix} 8 \\ 8 \\ 16 \\ 4 \end{bmatrix}$

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

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




Gradient descent step

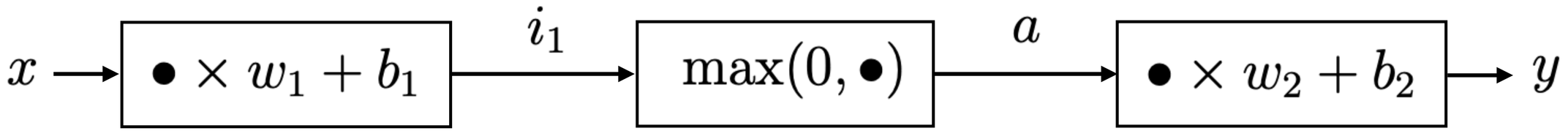
$$\begin{bmatrix} w_1 \\ b_1 \\ w_2 \\ b_2 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 1 \\ 2 \\ -5 \end{bmatrix} - \overset{1/4}{\alpha} \begin{bmatrix} 8 \\ 8 \\ 16 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -2 \\ -6 \end{bmatrix}$$

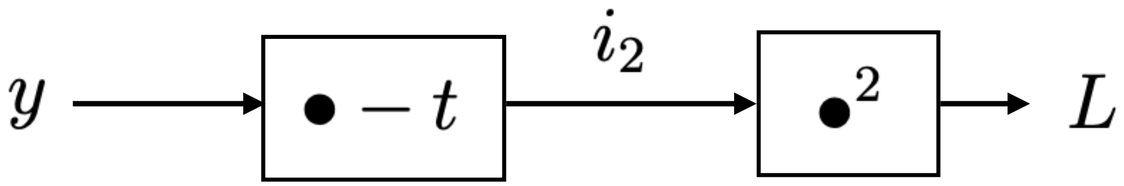
+ update weights

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

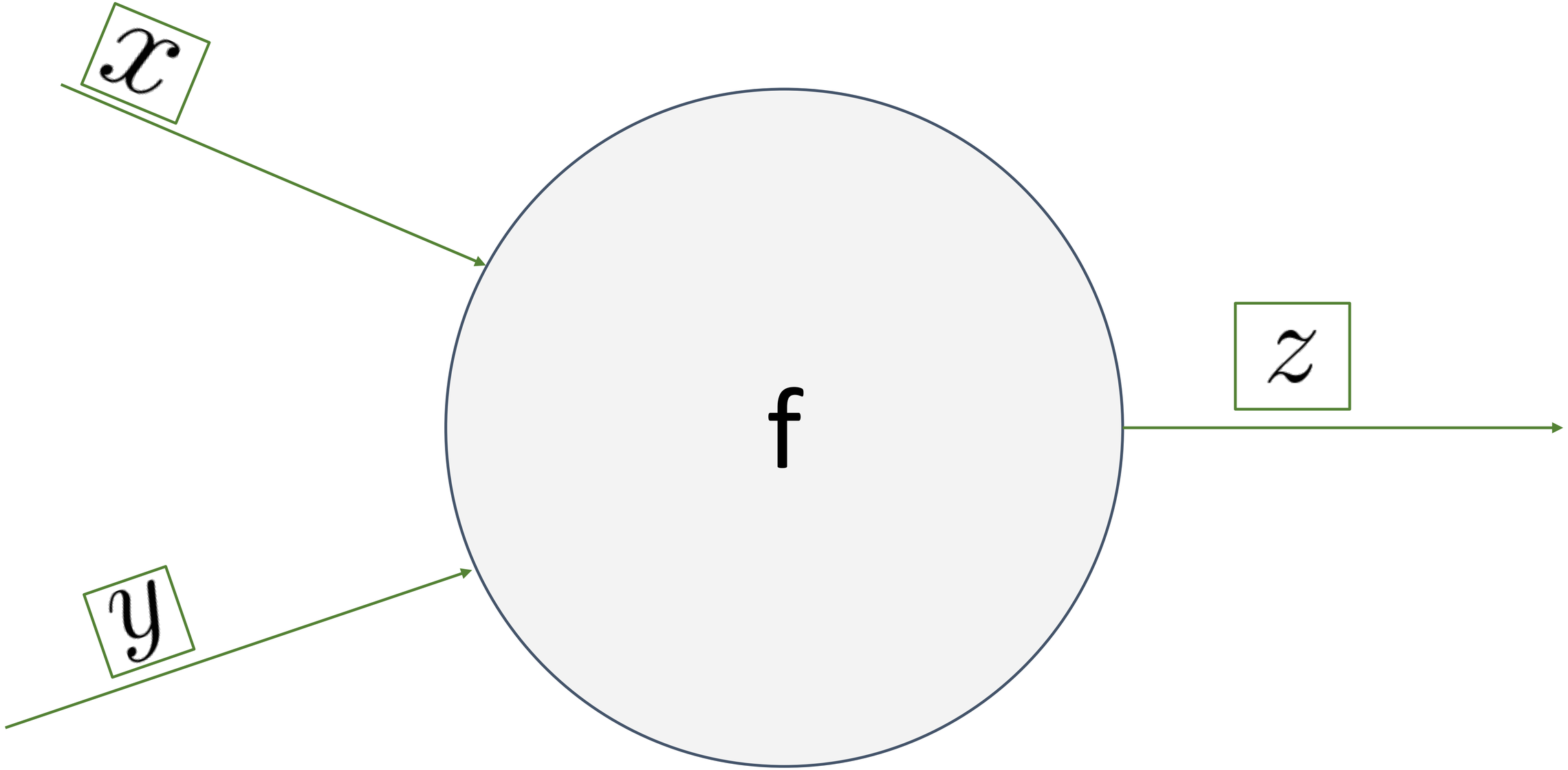
# Why **backwards**?

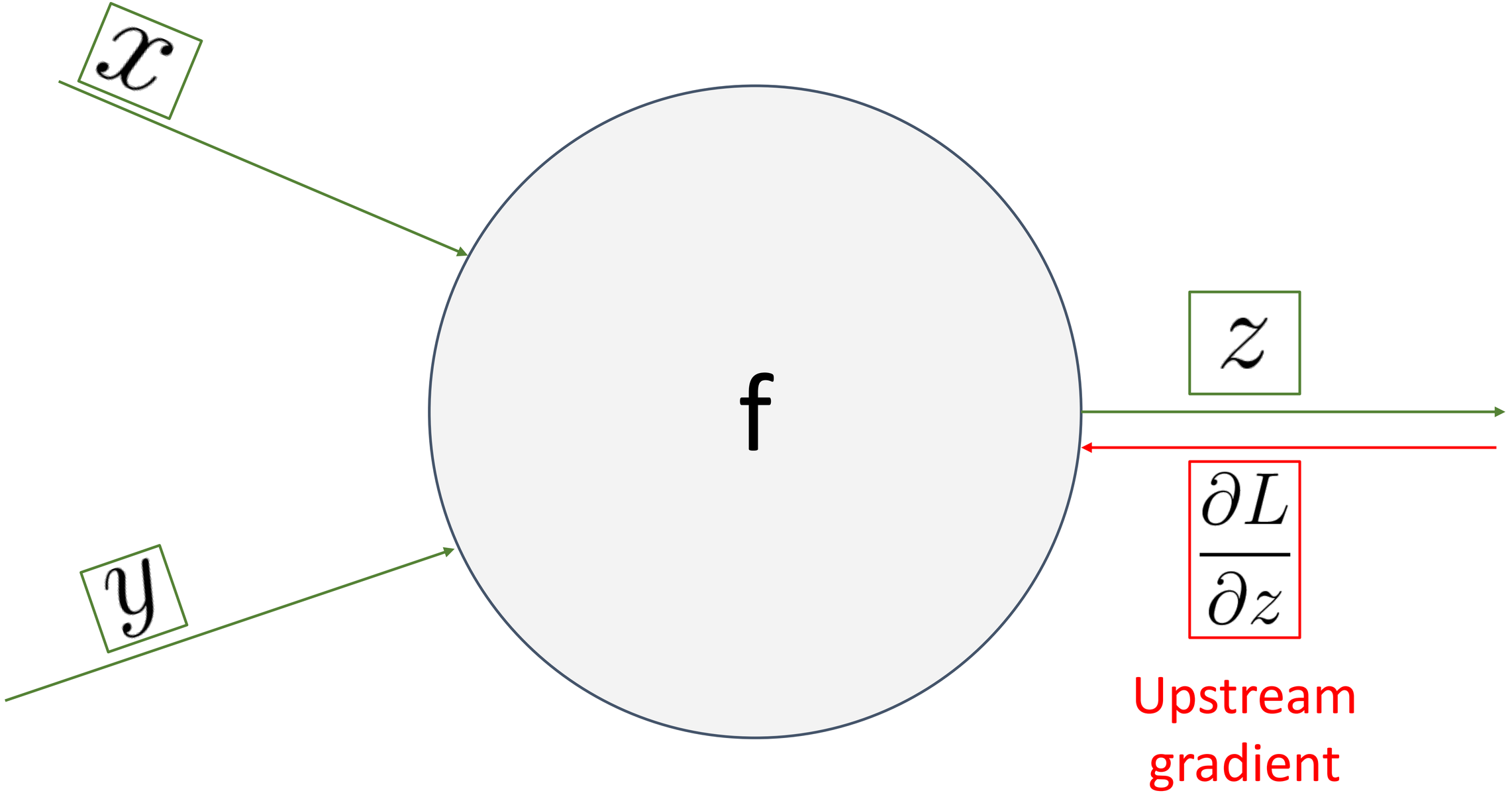
$$y = w_2(\max(0, w_1 x + b_1)) + b_2 \qquad L = (y - t)^2$$

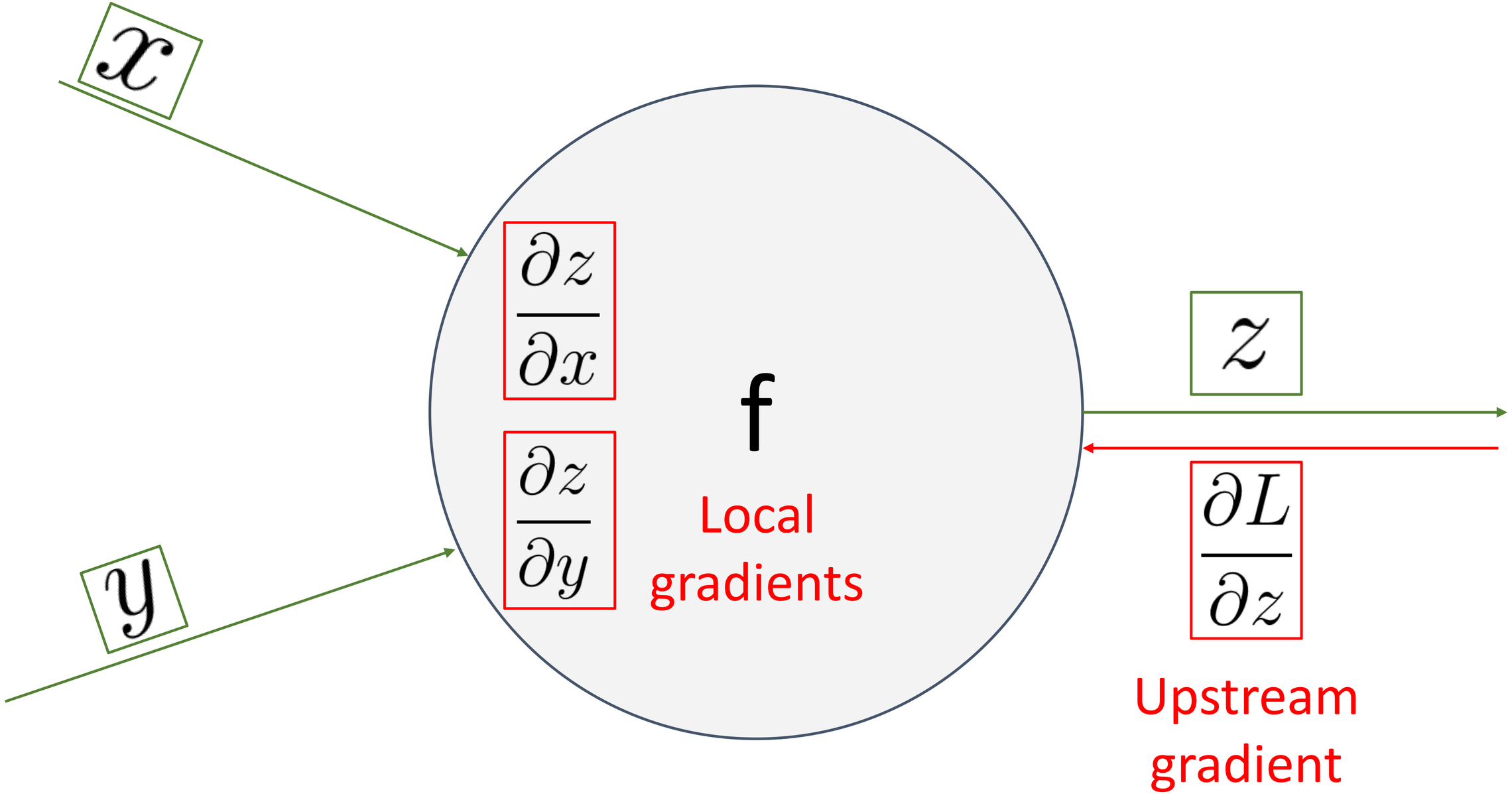


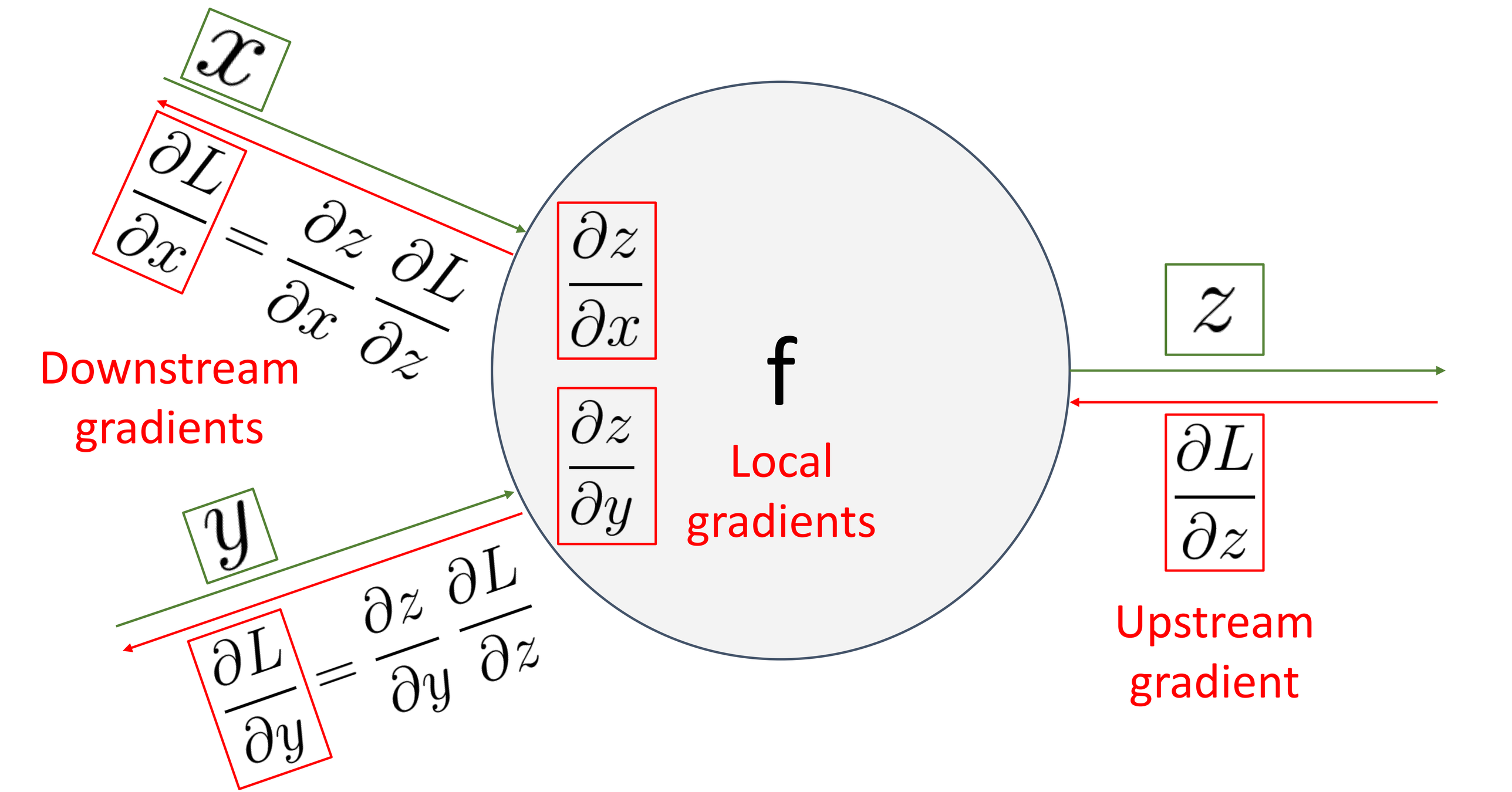



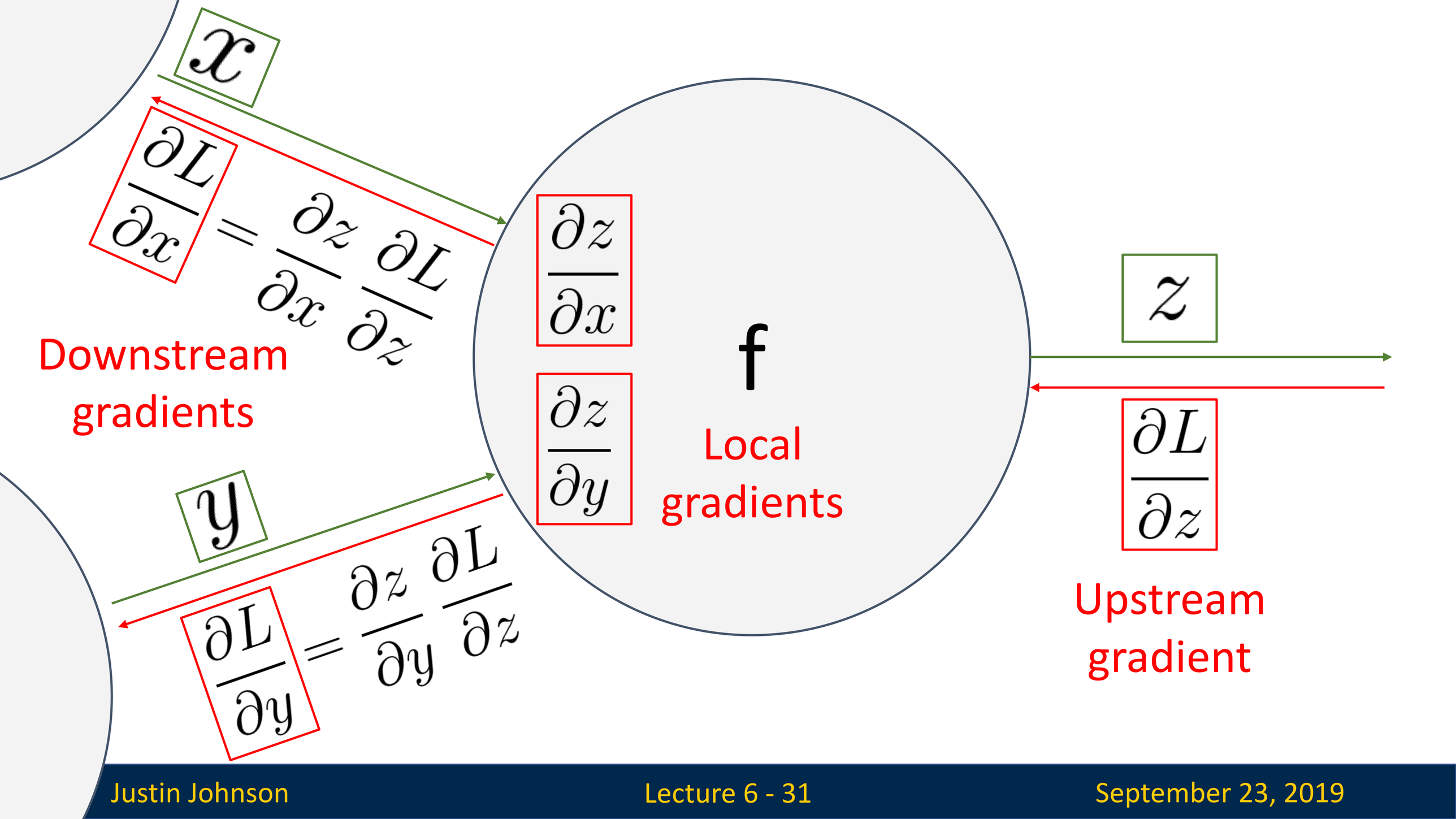

20.1


$x$
$y$
f
$z$

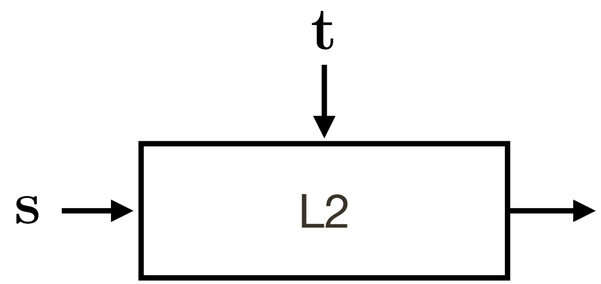

$x$
$y$
f
$z$
$\frac{\partial L}{\partial z}$
Upstream gradient


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


$x$
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
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

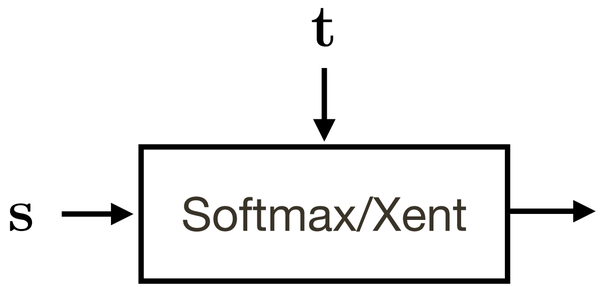

$x$
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
Downstream gradients
$y$
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
$z$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

# **Backward Pass** for Some **Common Layers**




$$e = \frac{1}{2}|\mathbf{s} - \mathbf{t}|^2$$

$$\frac{\partial e}{\partial \mathbf{s}} = \mathbf{s} - \mathbf{t}$$

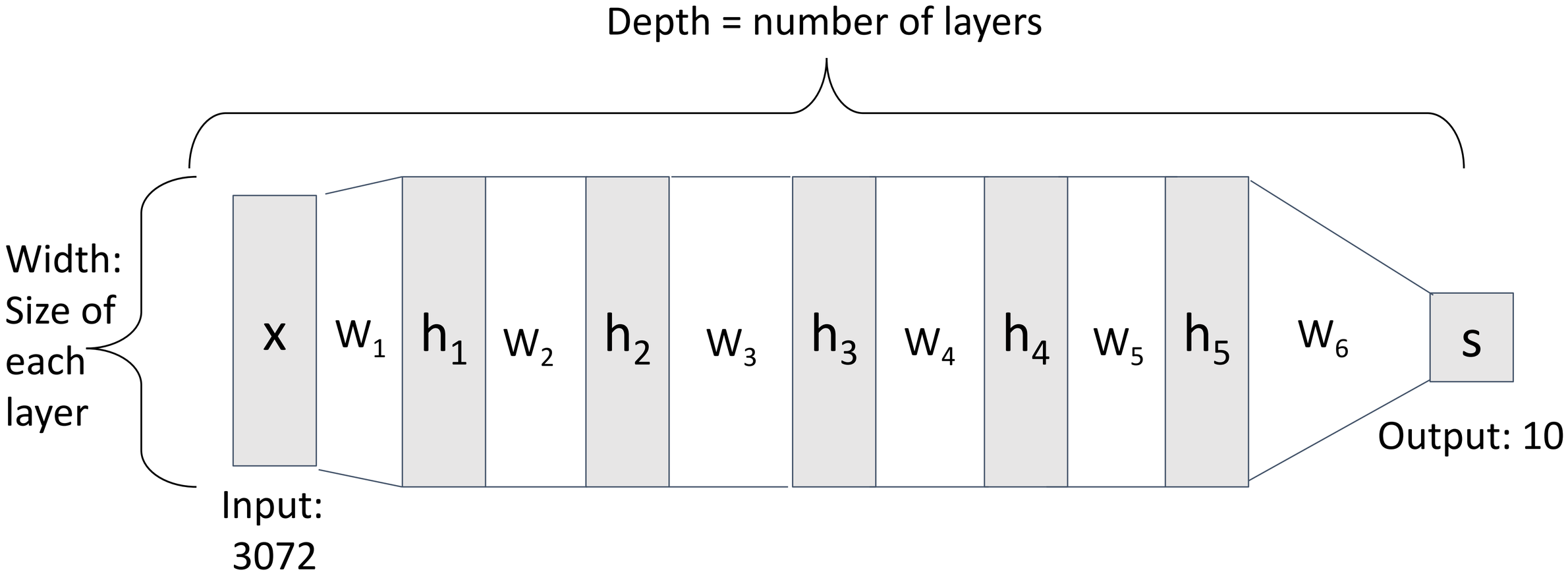



$$e = -\log \frac{e^{s_y}}{\sum_i e^{s_i}}$$

$$\frac{\partial e}{\partial \mathbf{s}} = \sigma(\mathbf{s}) - \mathbf{t}$$


[ You will do this for Assignment 6 ]

# Deep Neural Networks

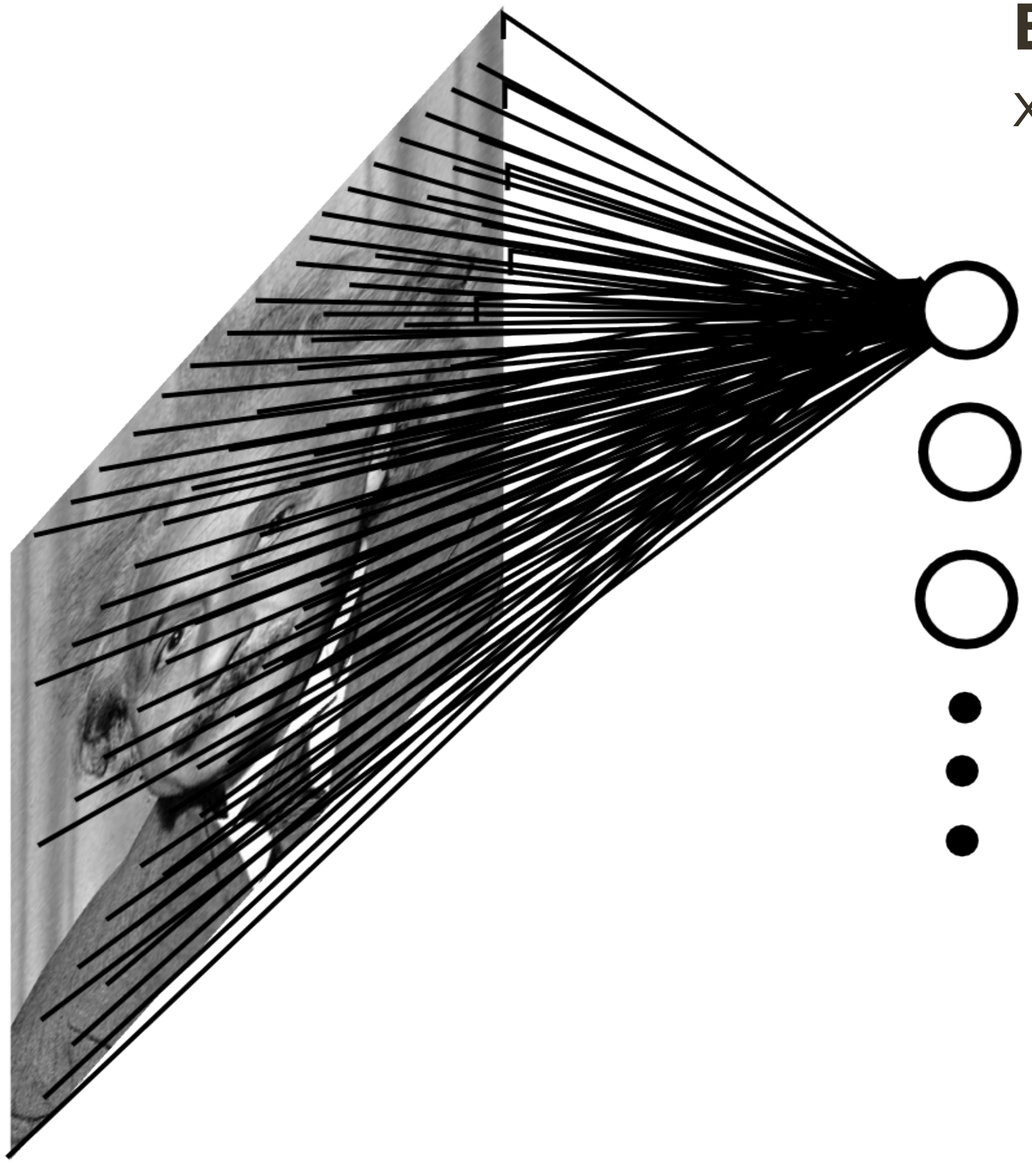



$$s = W_6 \max(0, W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x))))))$$

# **Backward Pass** for Some **Common Layers**

Linear layers — fully connected

# **Fully Connected** Layer



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

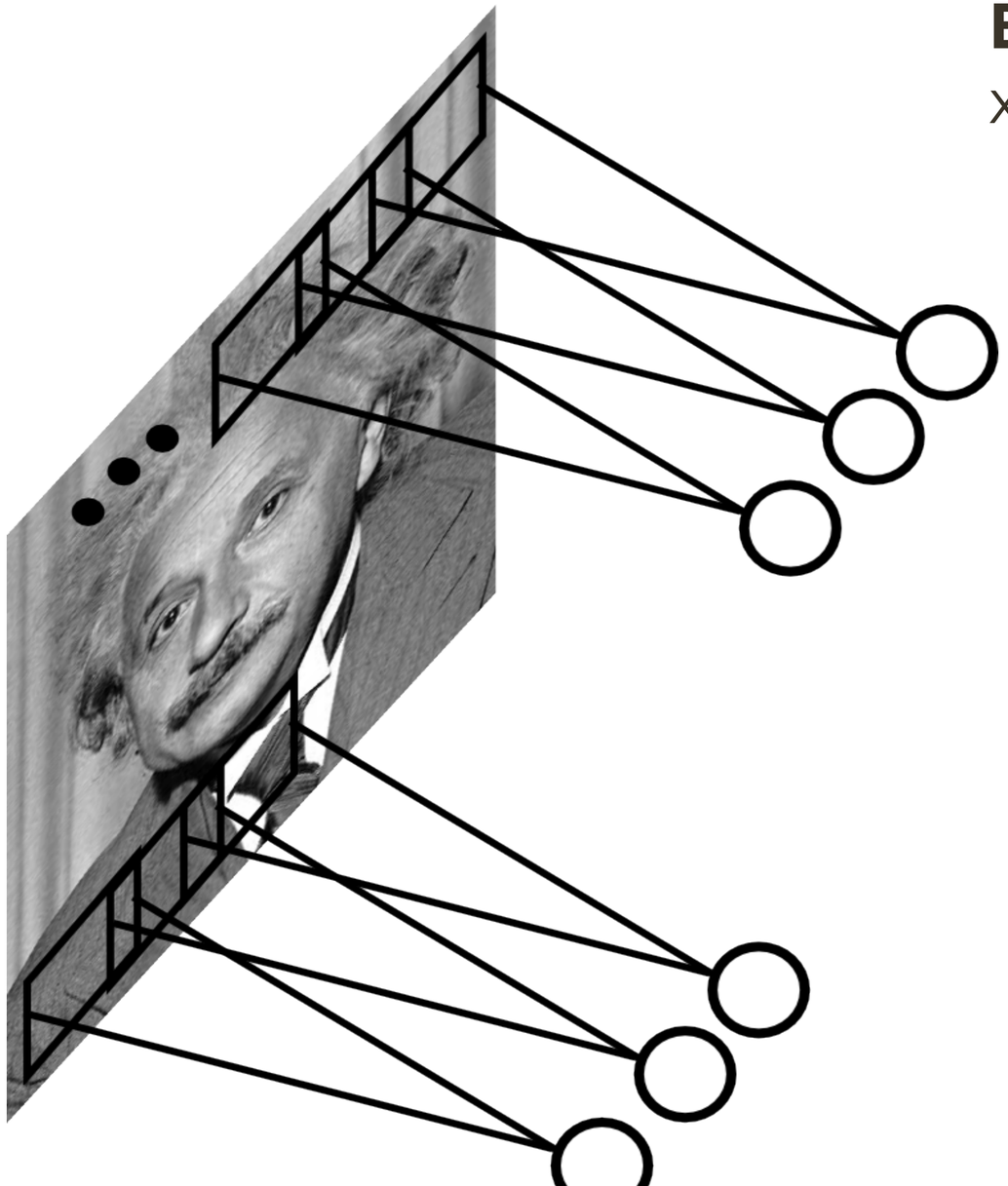
= **1.6 Billion** parameters (for one layer!)

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



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

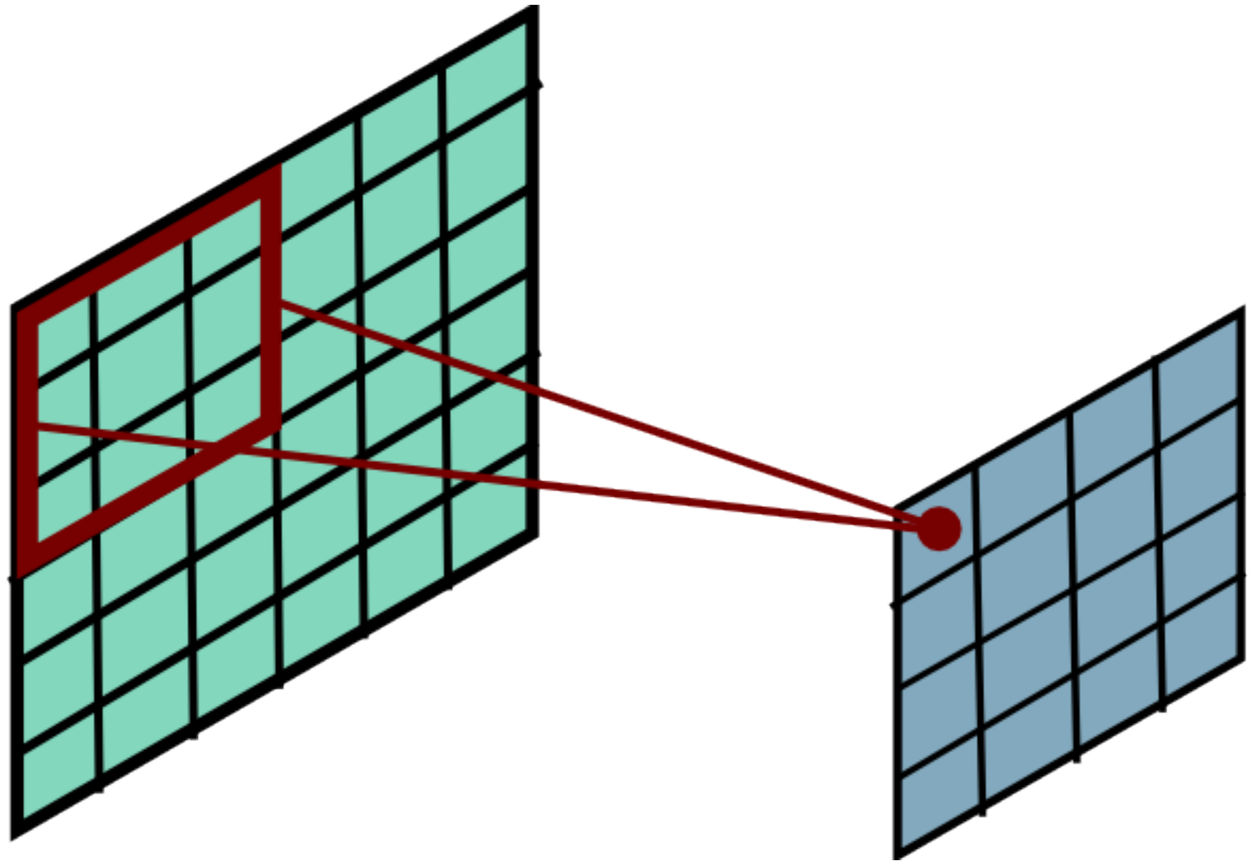
**Filter size:** 10 x 10

= 100 parameters

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

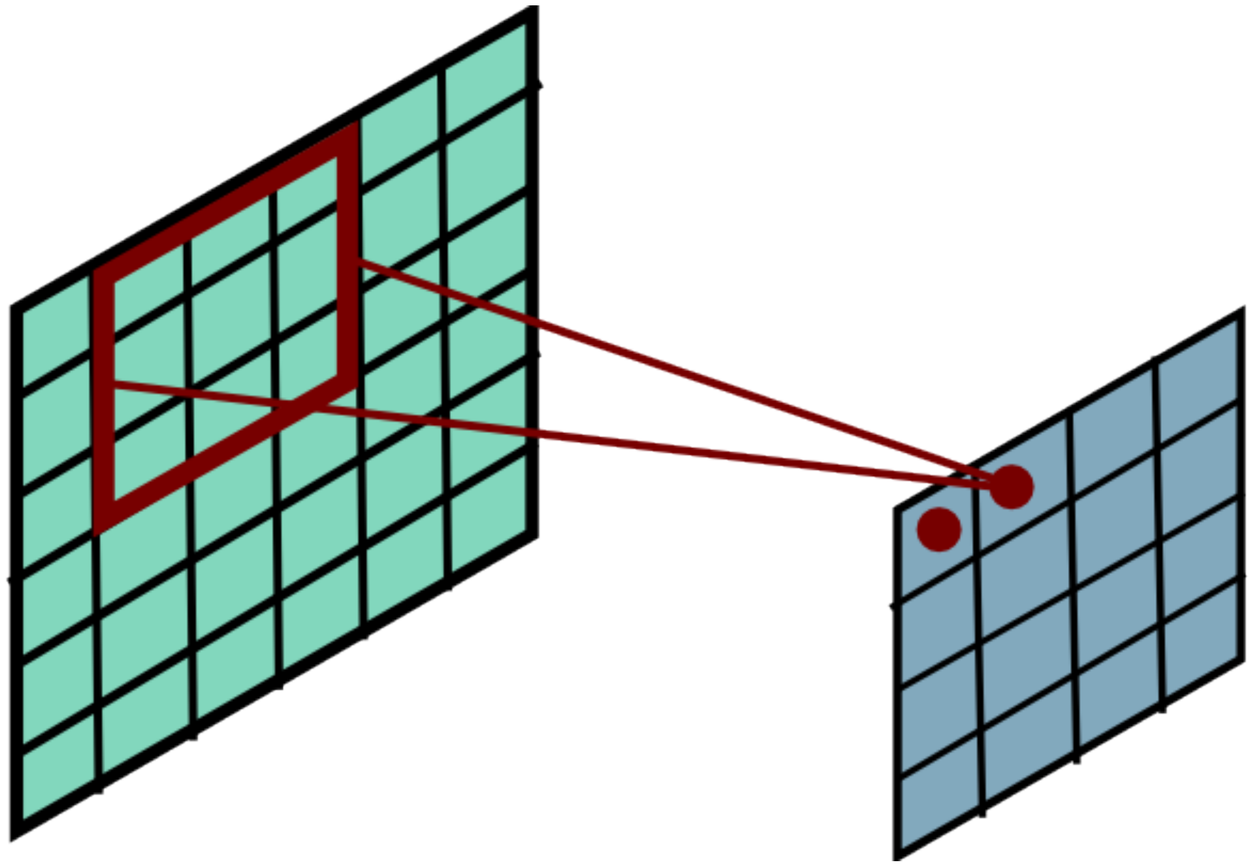
* slide adopted from Marc'Aurelio Renzato

# **Convolutional** Layer



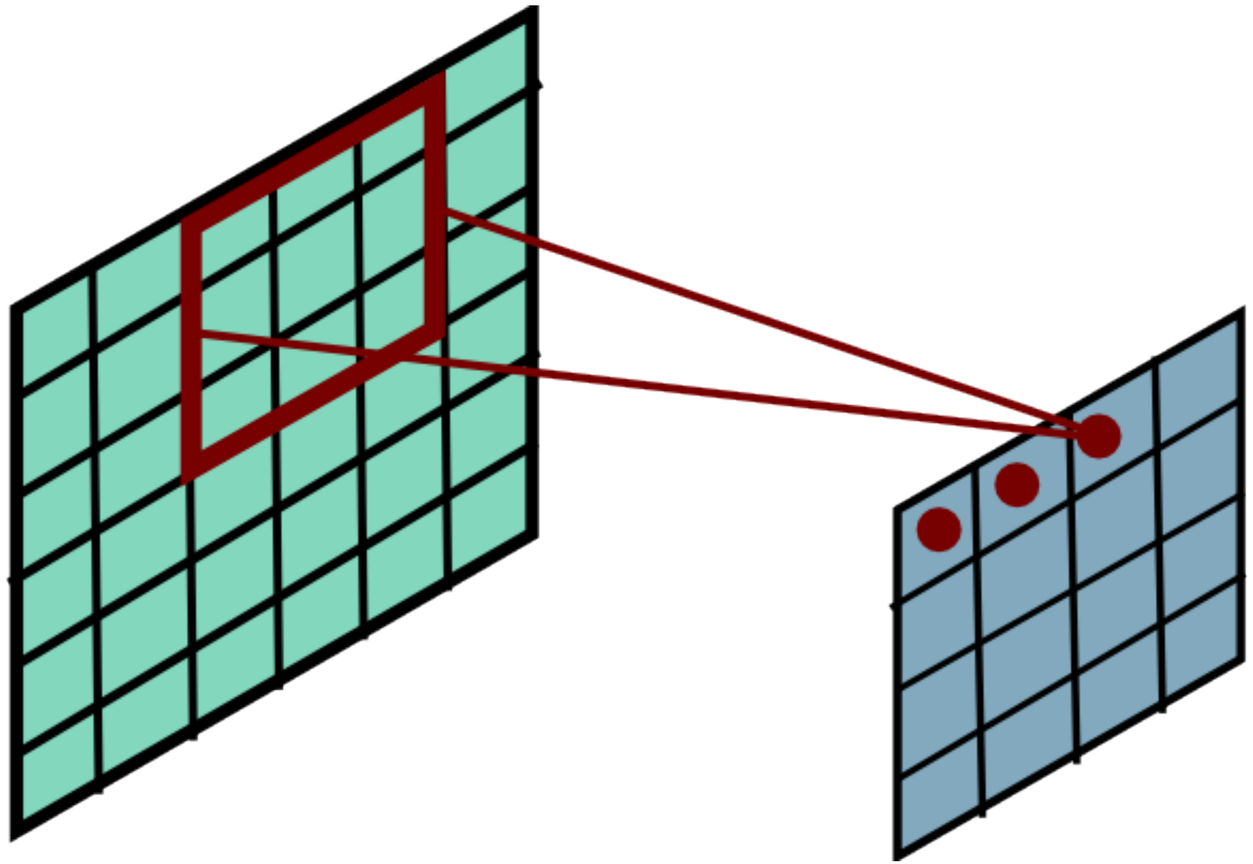
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



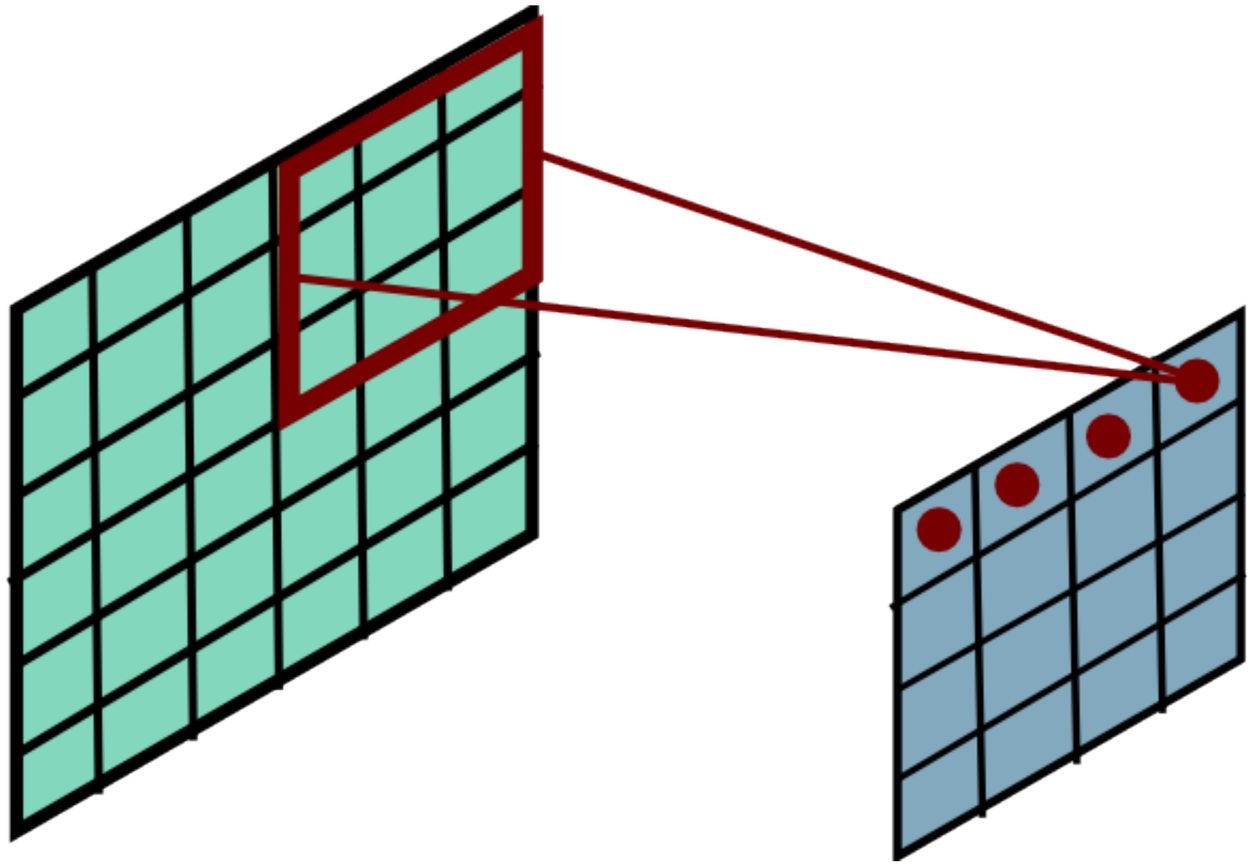
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



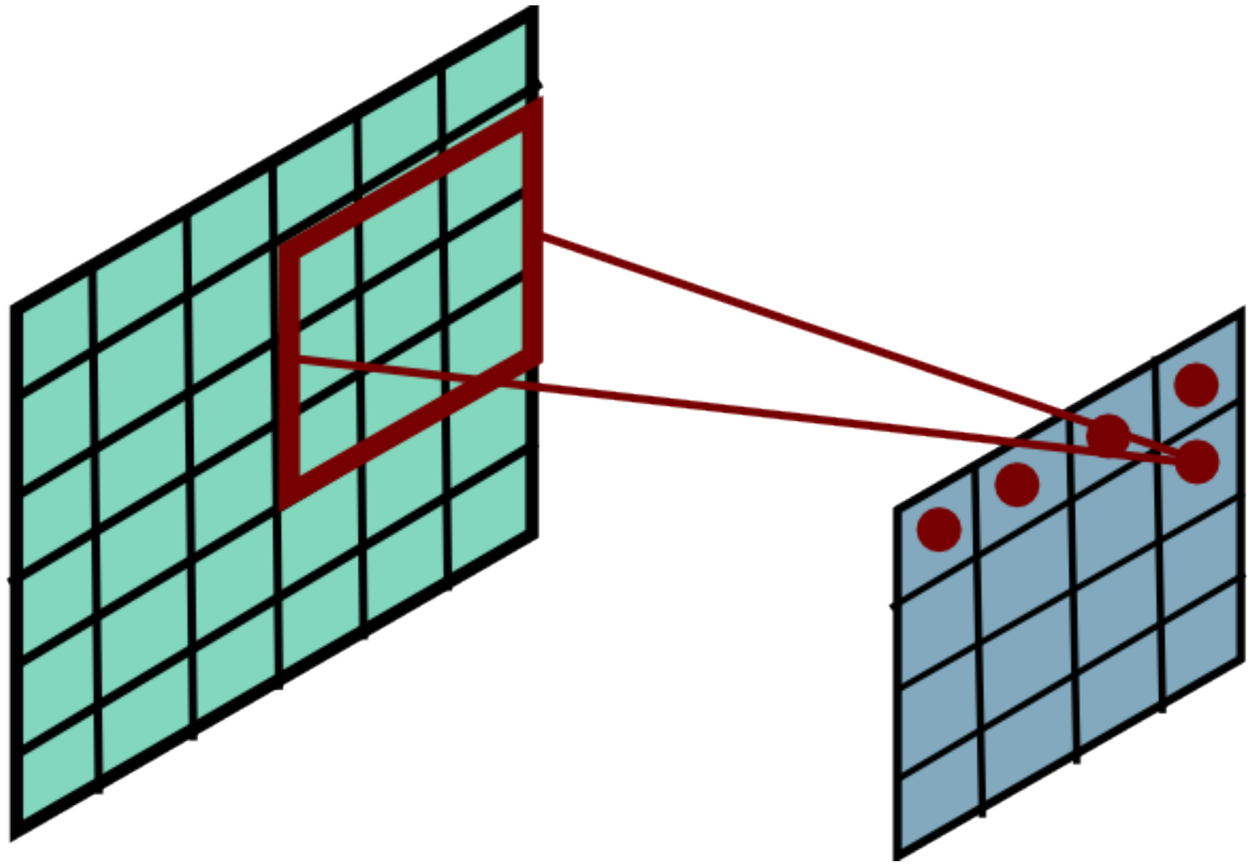
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



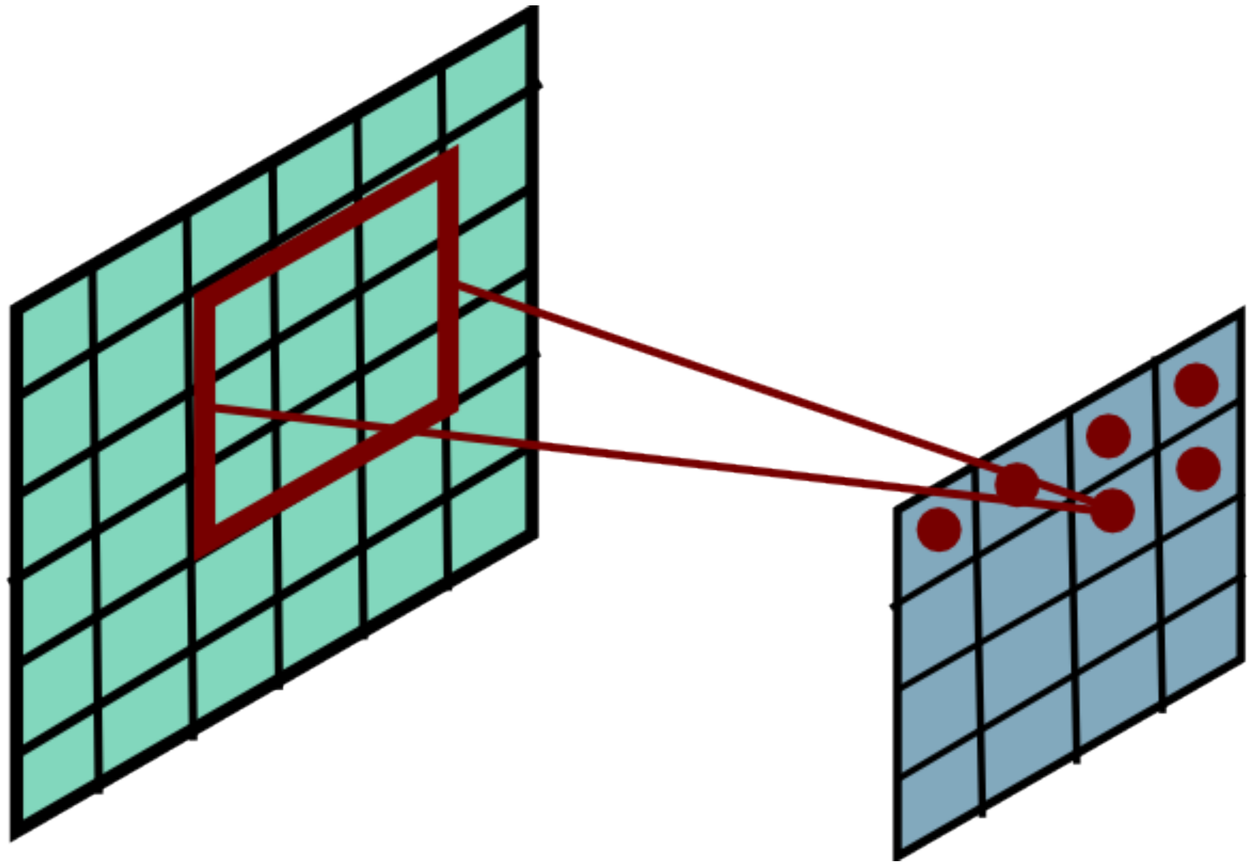
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



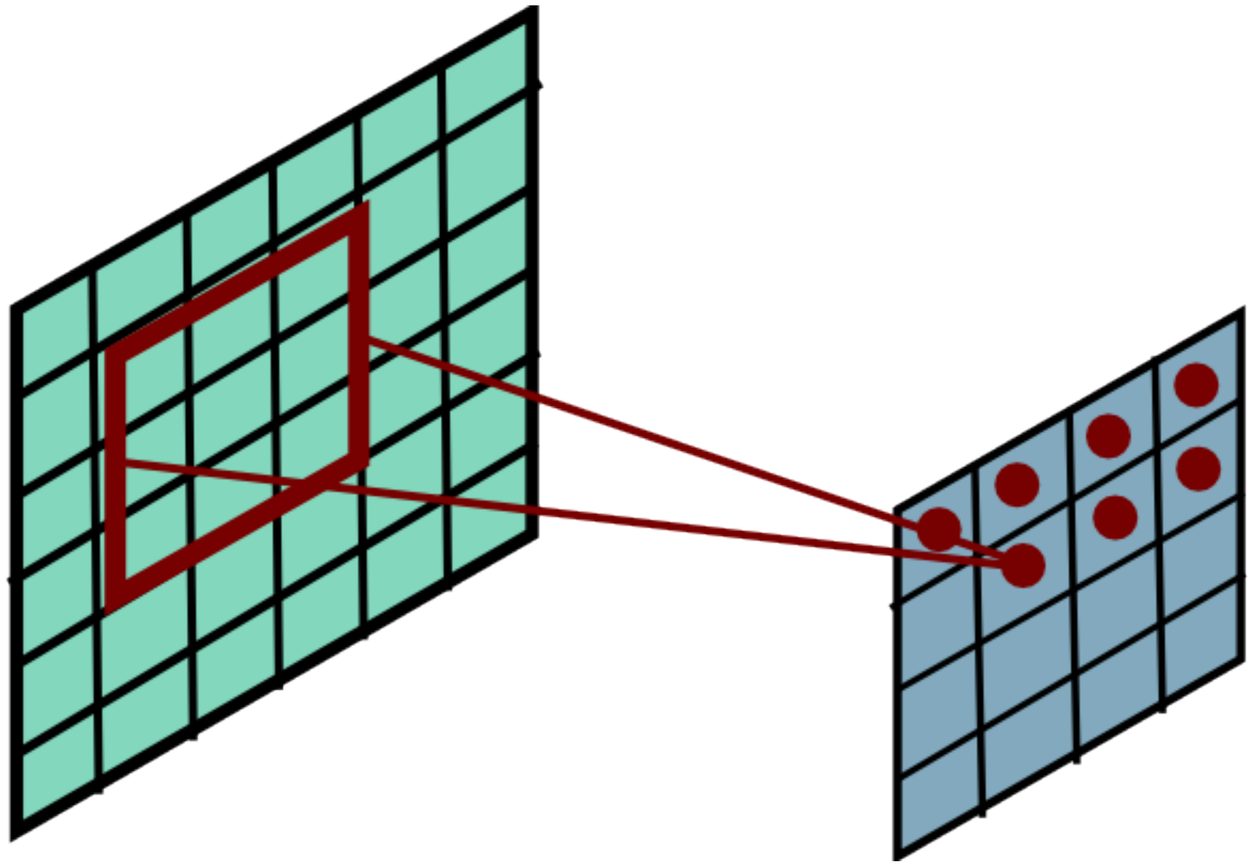
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



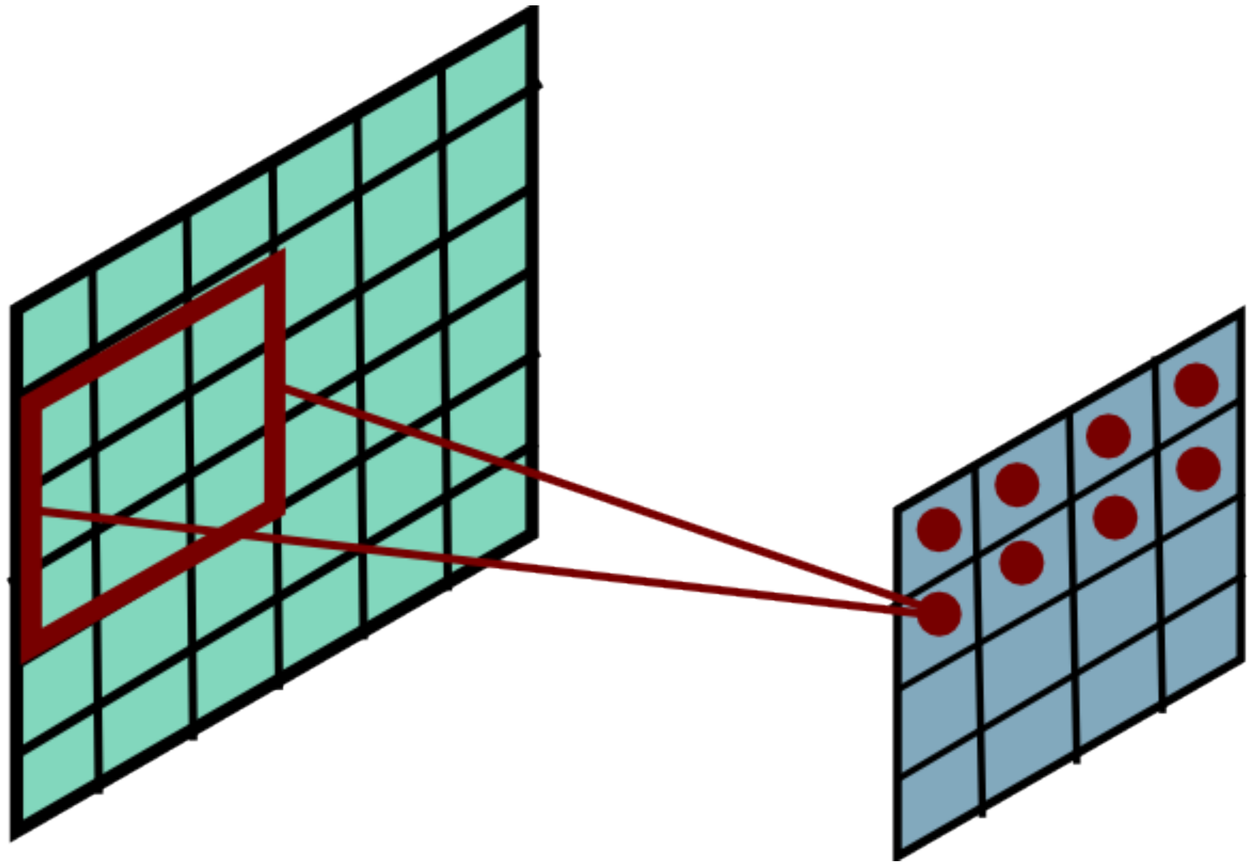
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



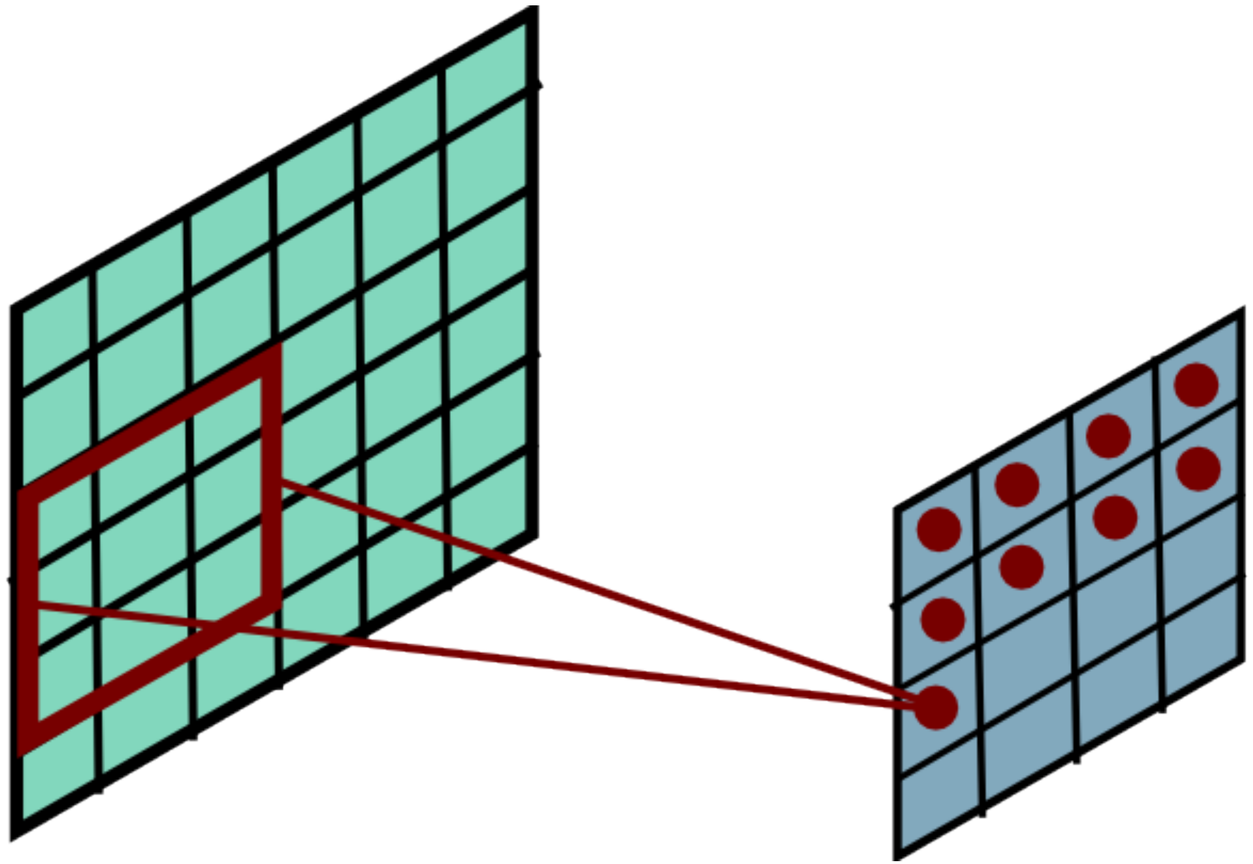
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



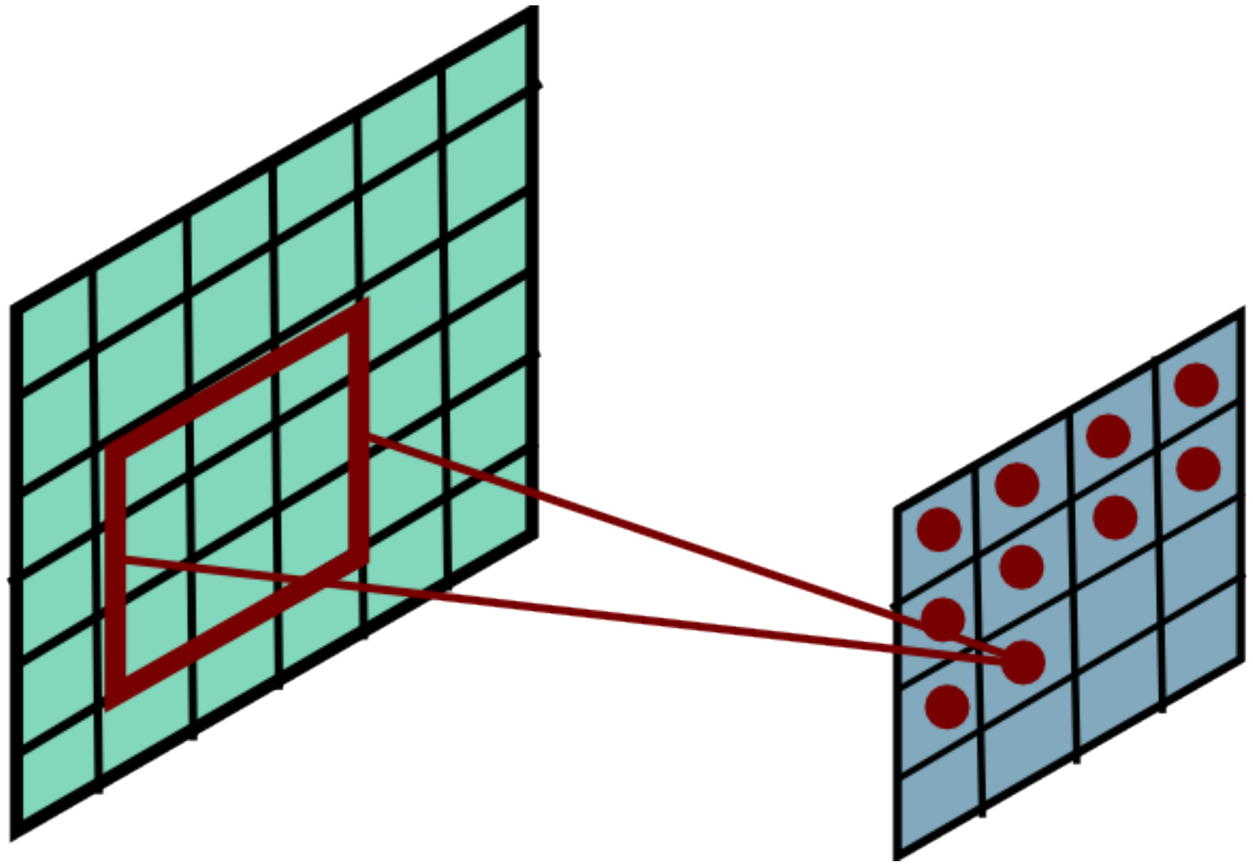
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



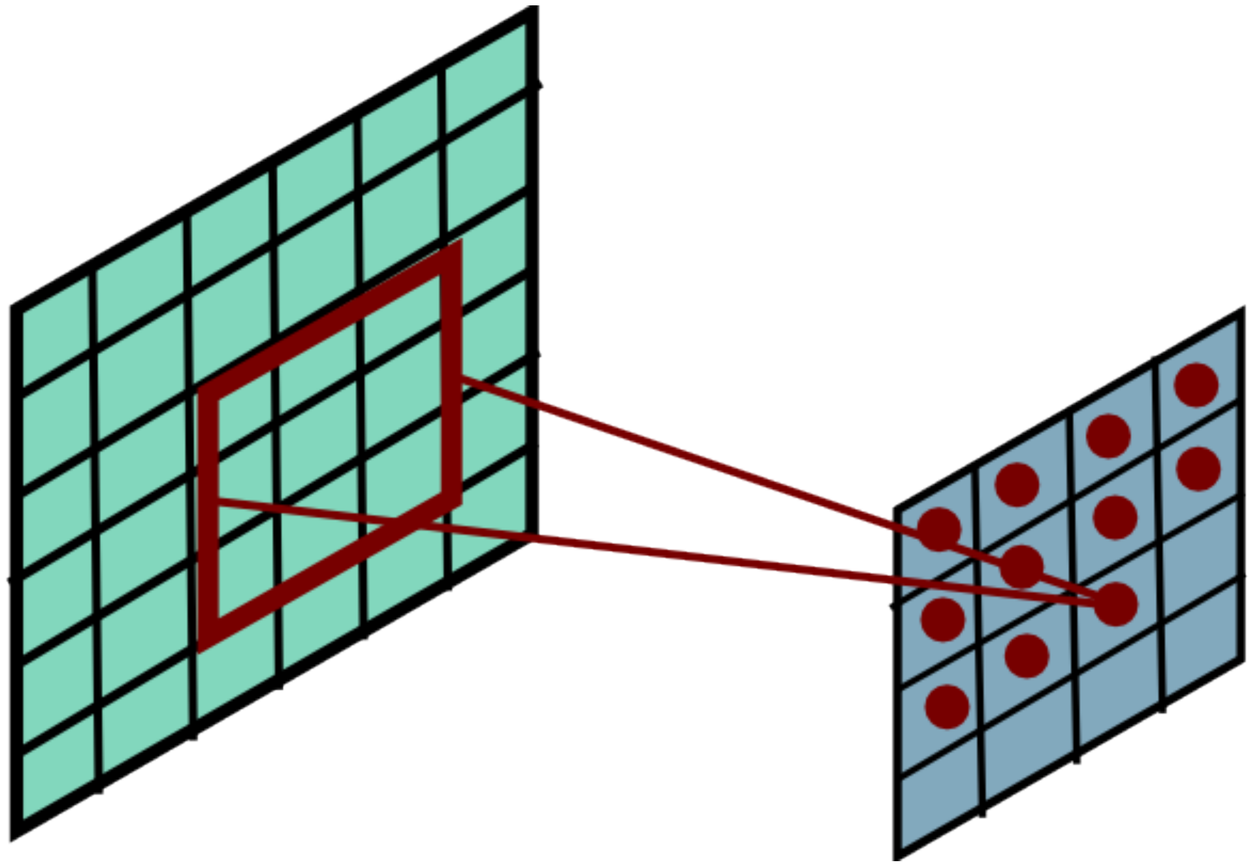
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



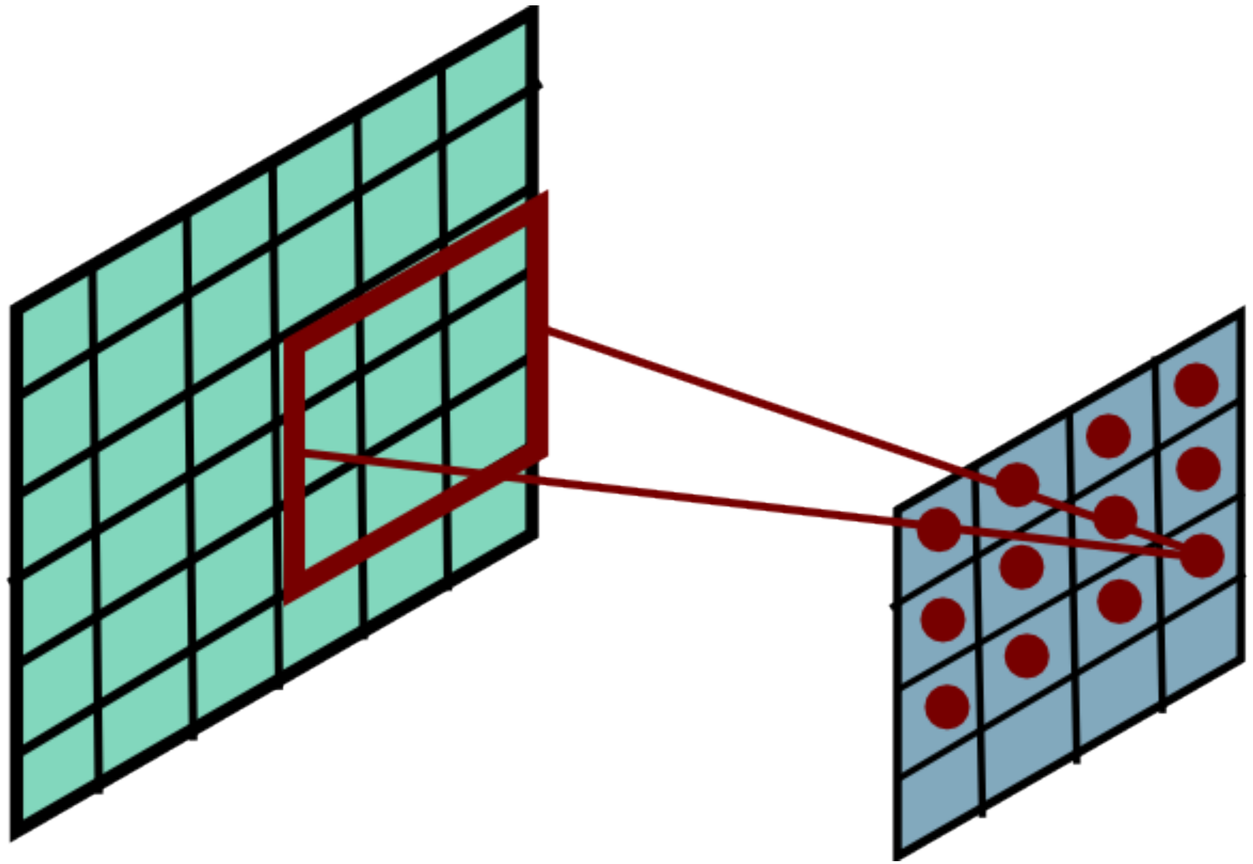
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



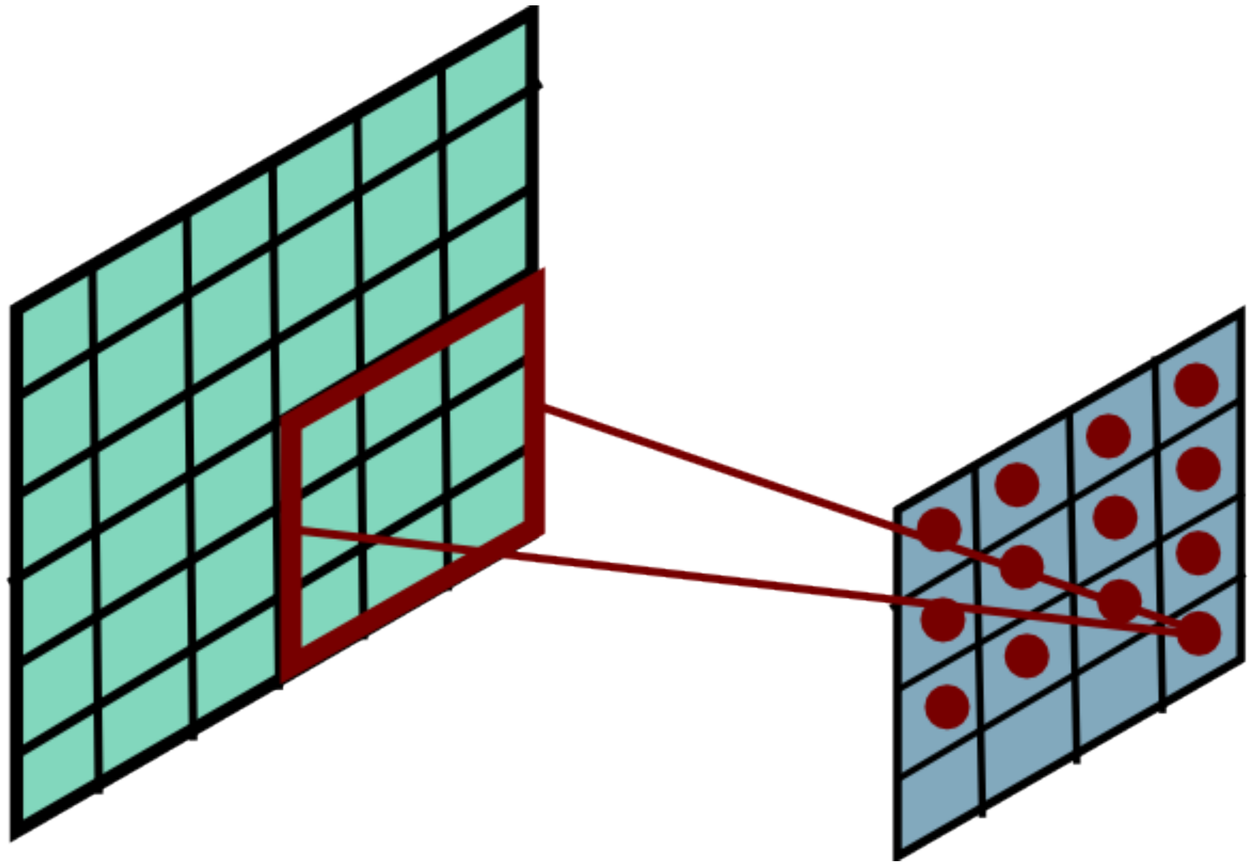
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



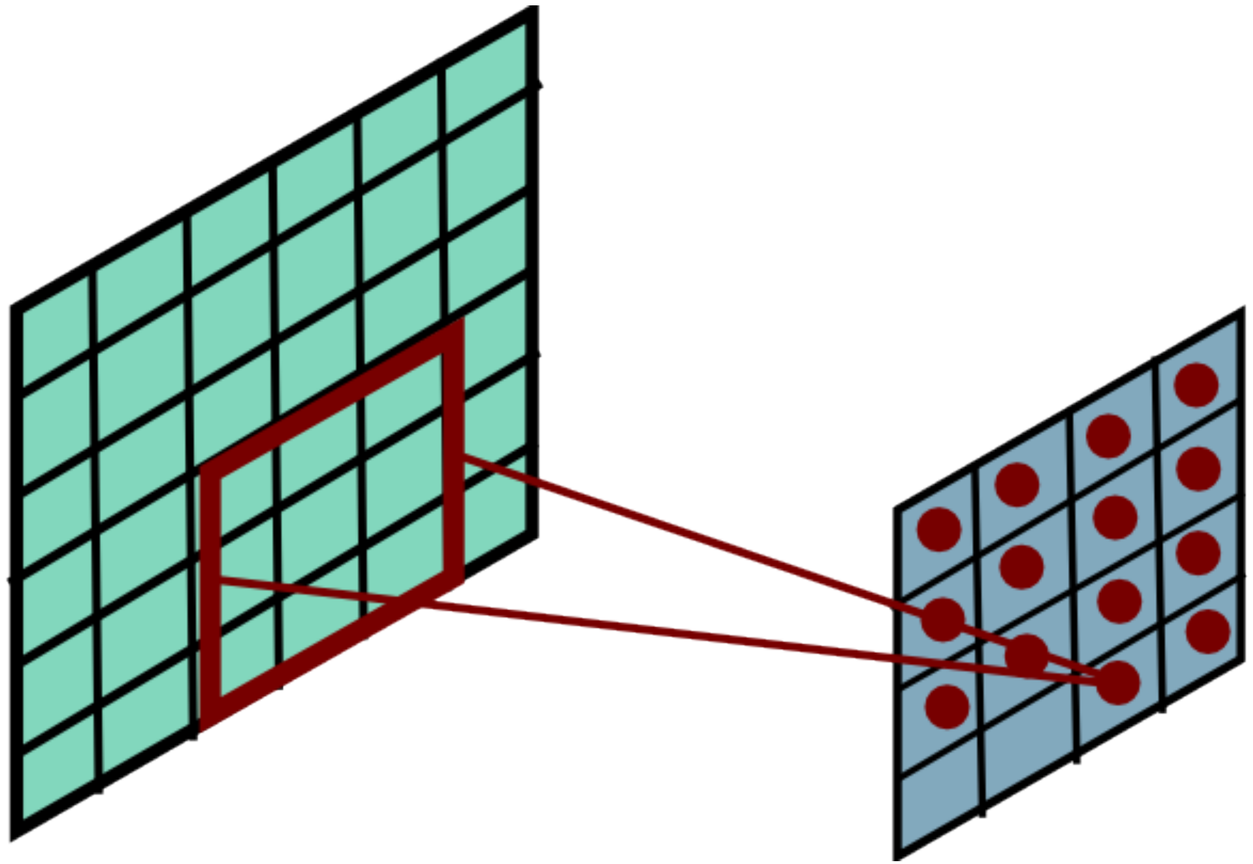
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



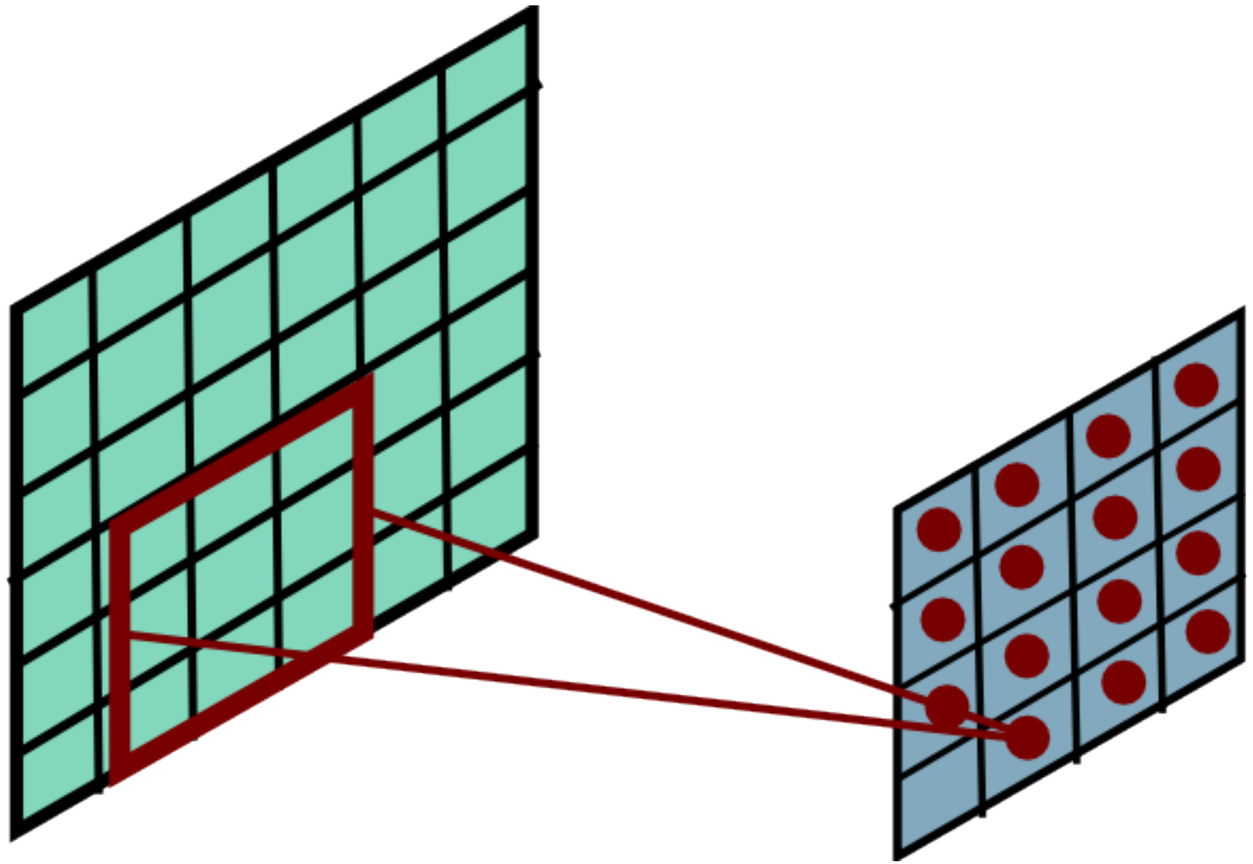
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



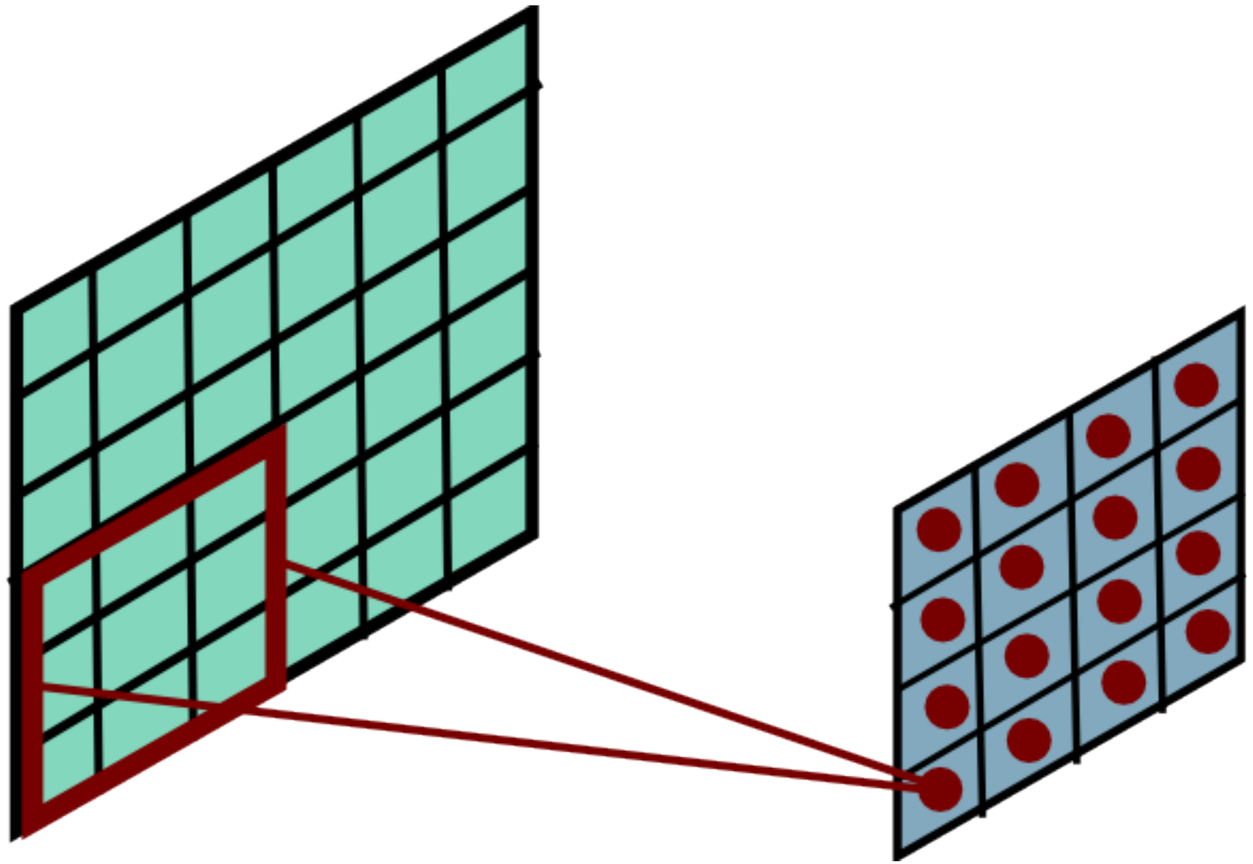
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



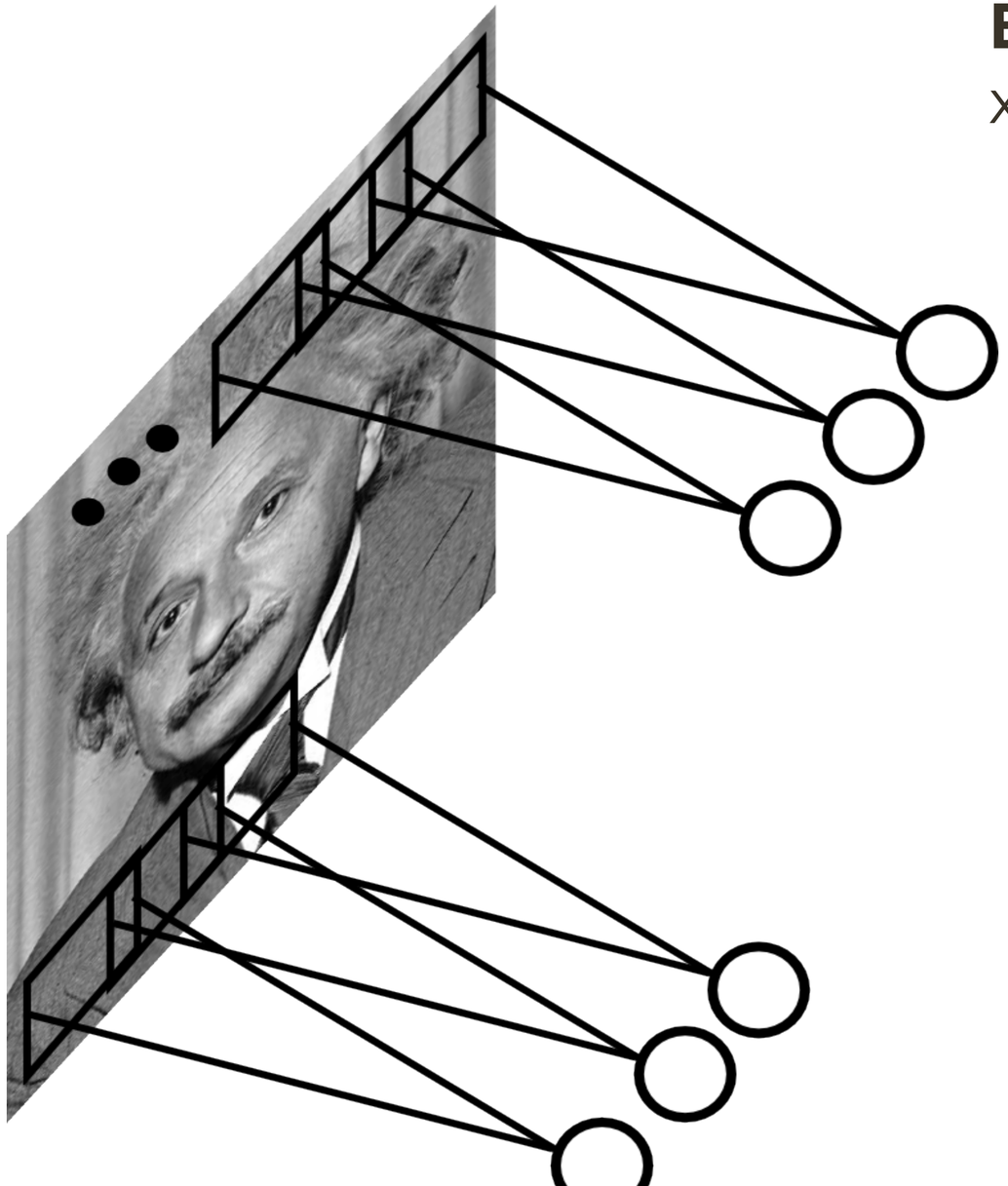
* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



* slide from Marc'Aurelio Renzato

# **Convolutional** Layer



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

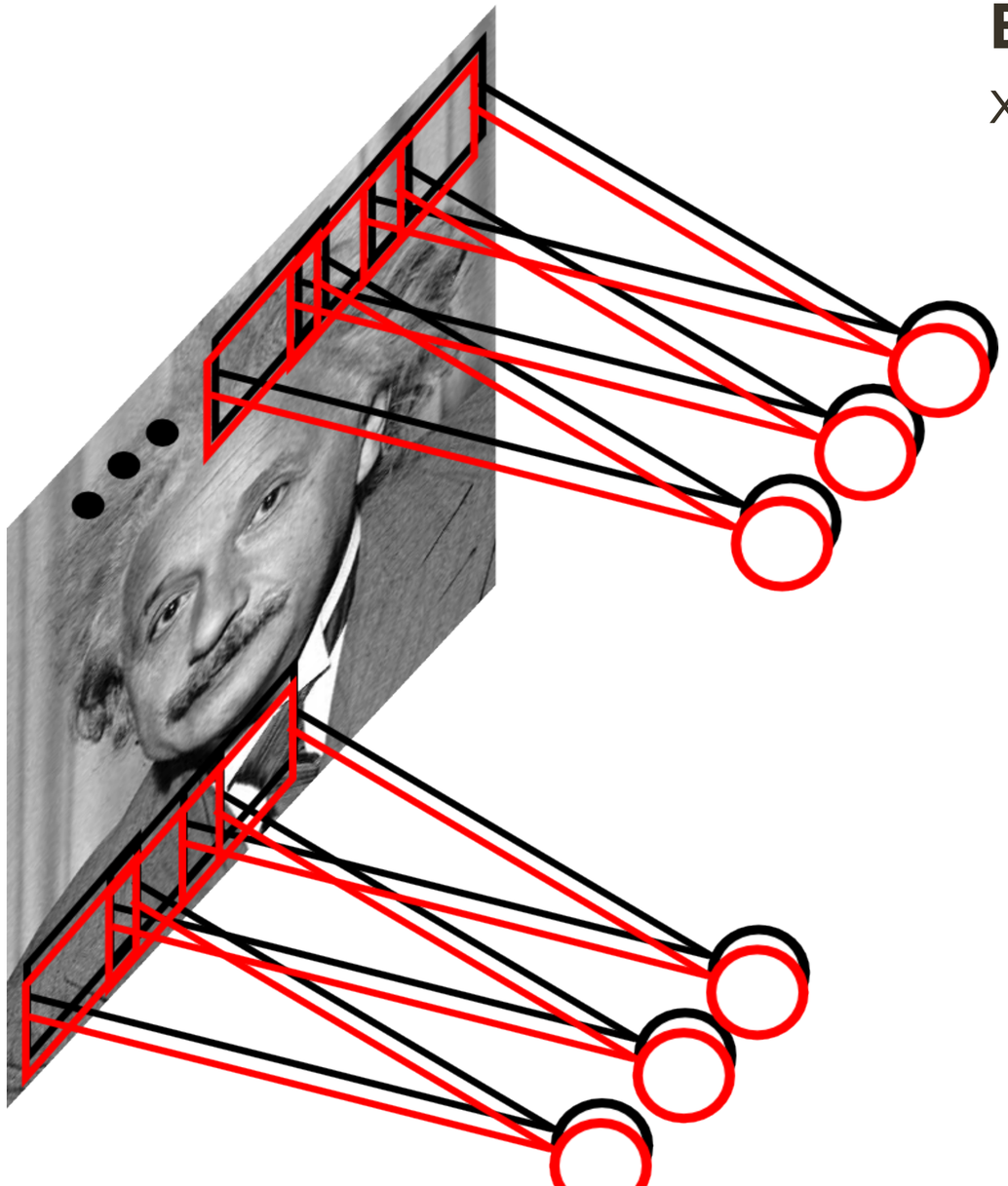
**Filter size:** 10 x 10

= 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

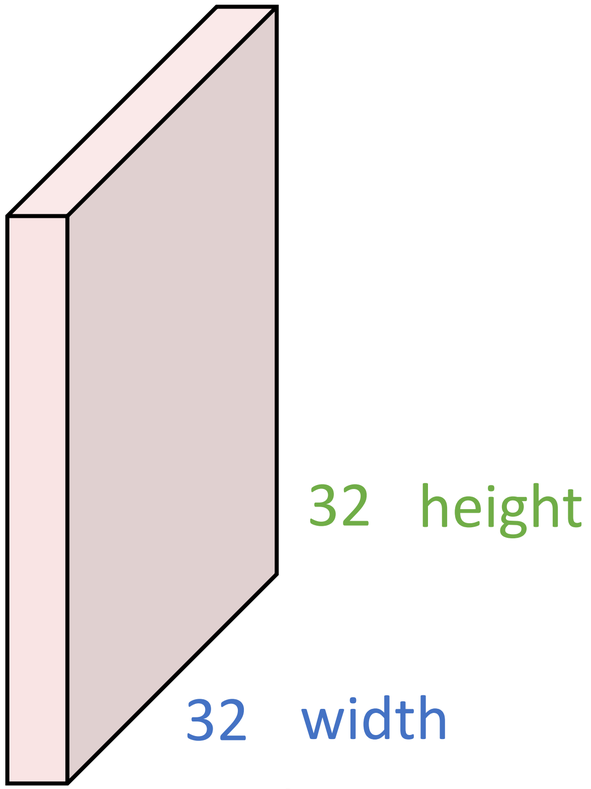
= 2000 parameters

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

* slide from Marc'Aurelio Renzato

# Convolution Layer

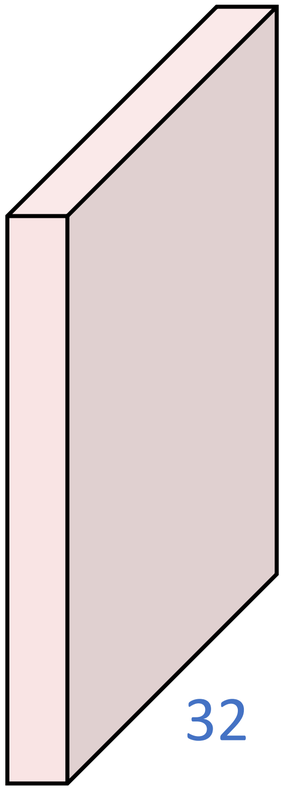
3x32x32 image: preserve spatial structure



# Convolution Layer

Filters always extend the full depth of the input volume

3x32x32 image

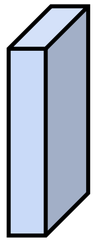


32 height

32 width

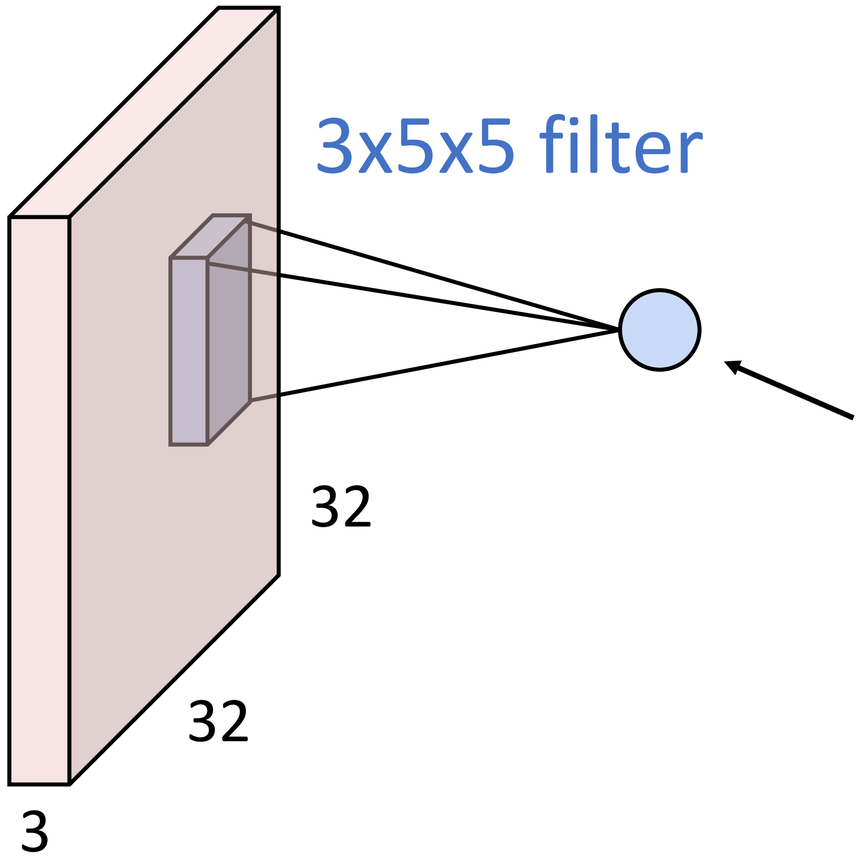
3 depth / channels

3x5x5 filter



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

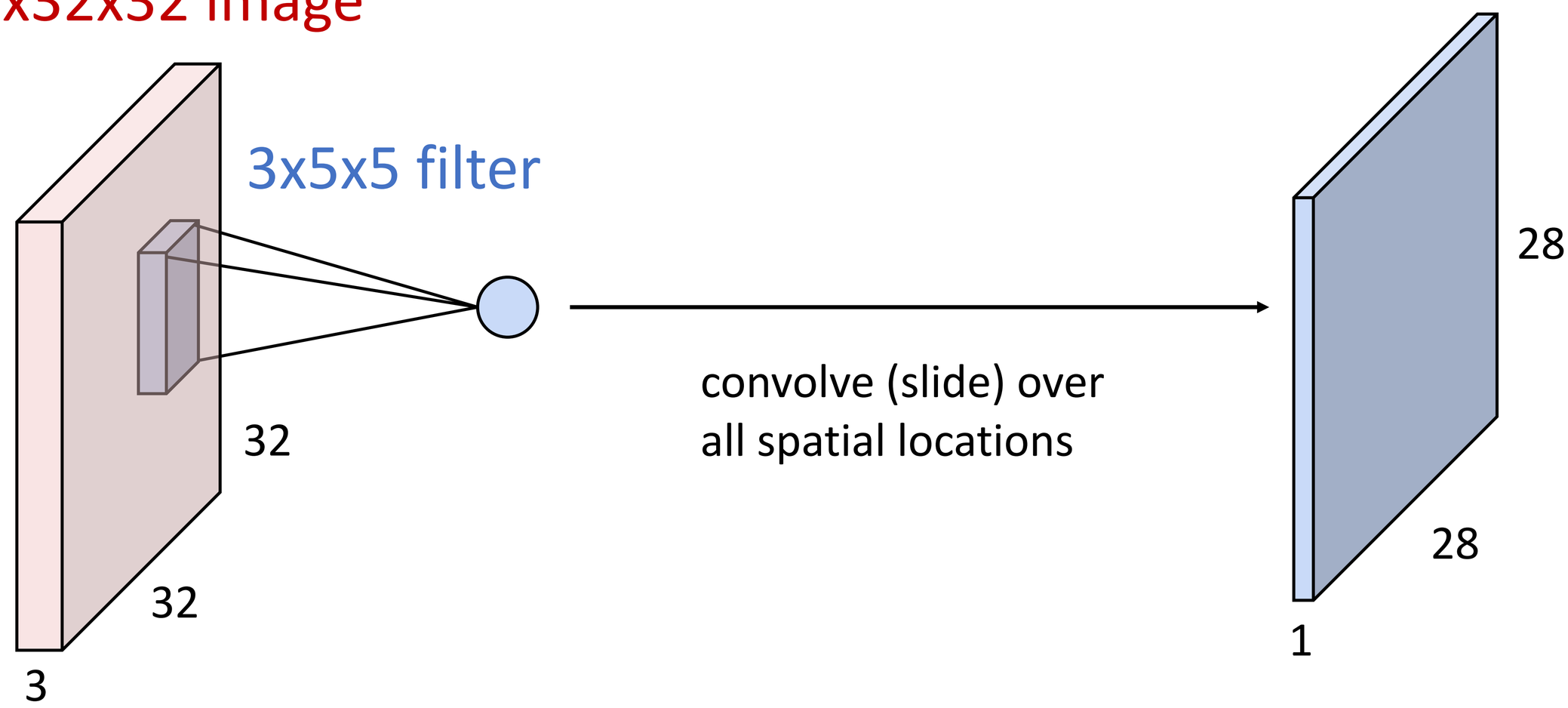
# Convolution Layer



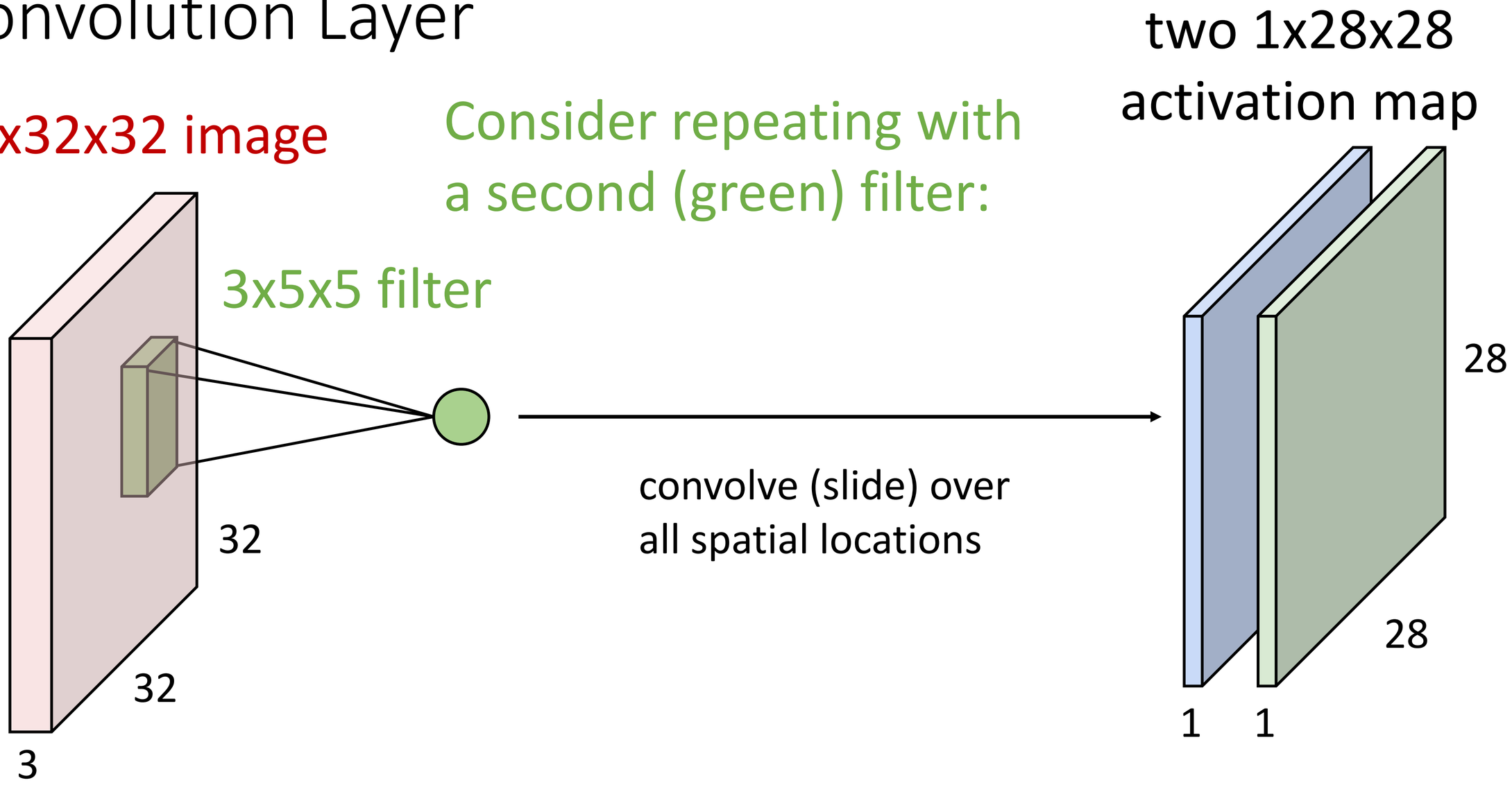

**1 number:**
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

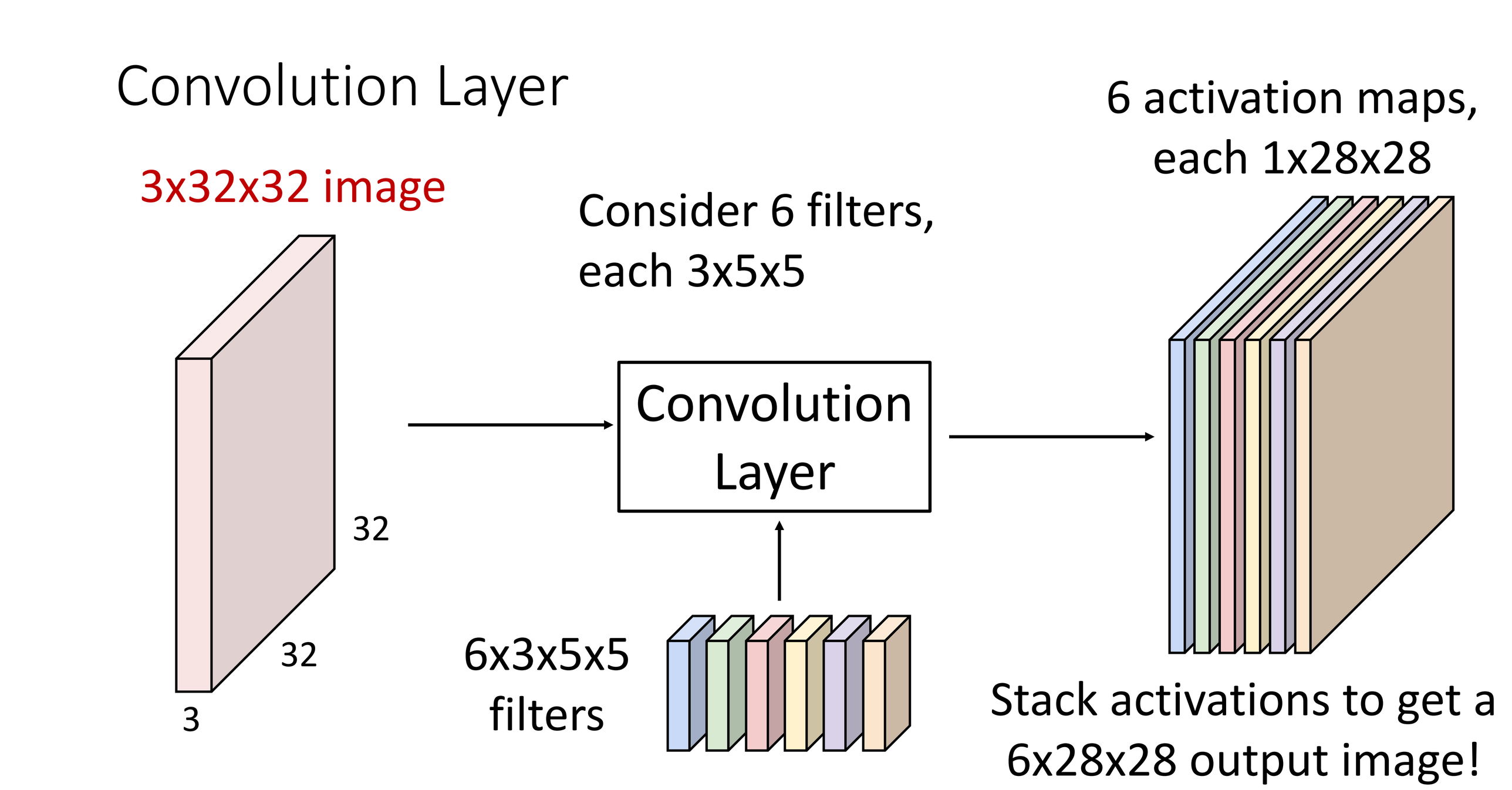
# Convolution Layer



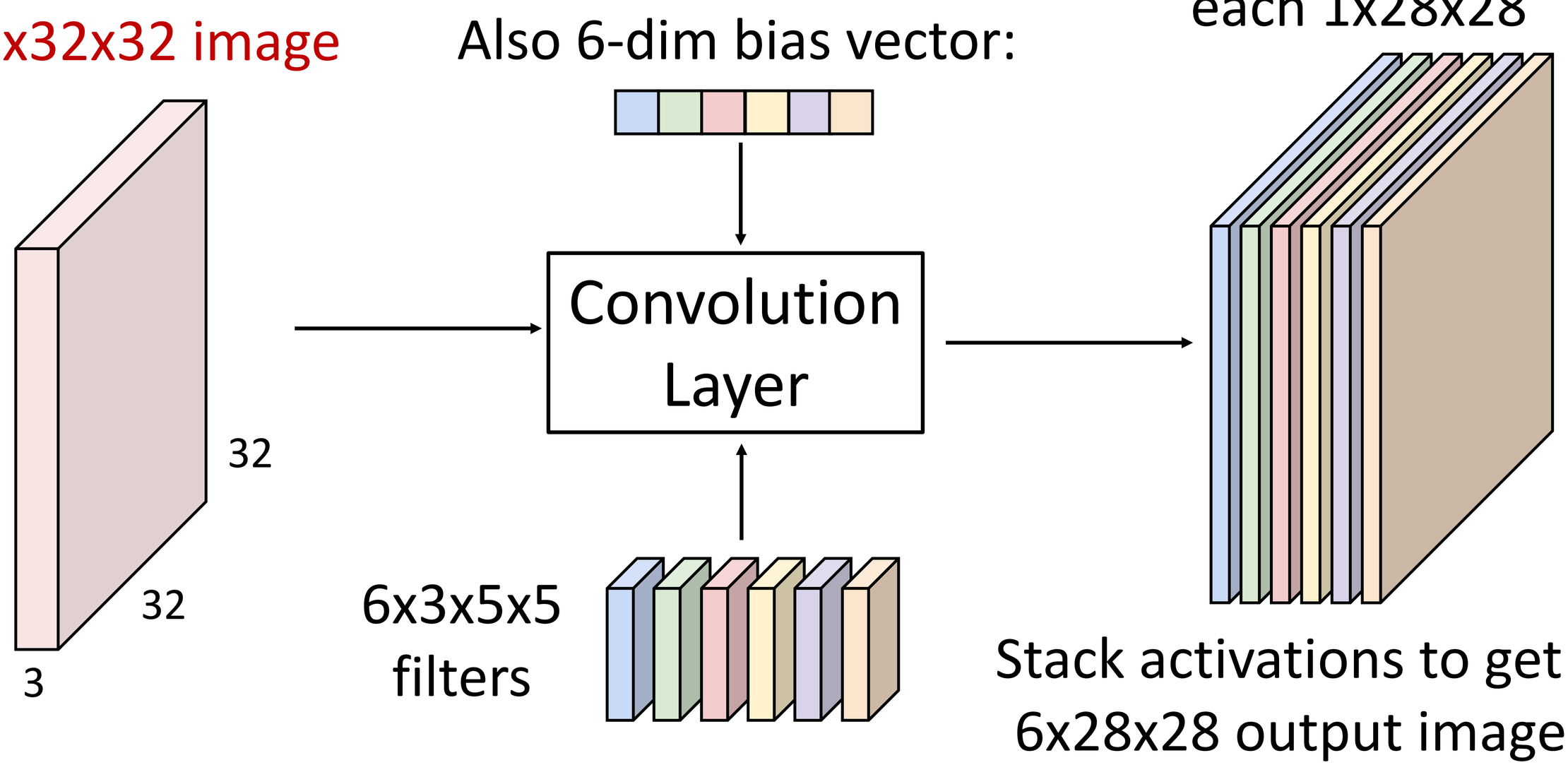
# Convolution Layer



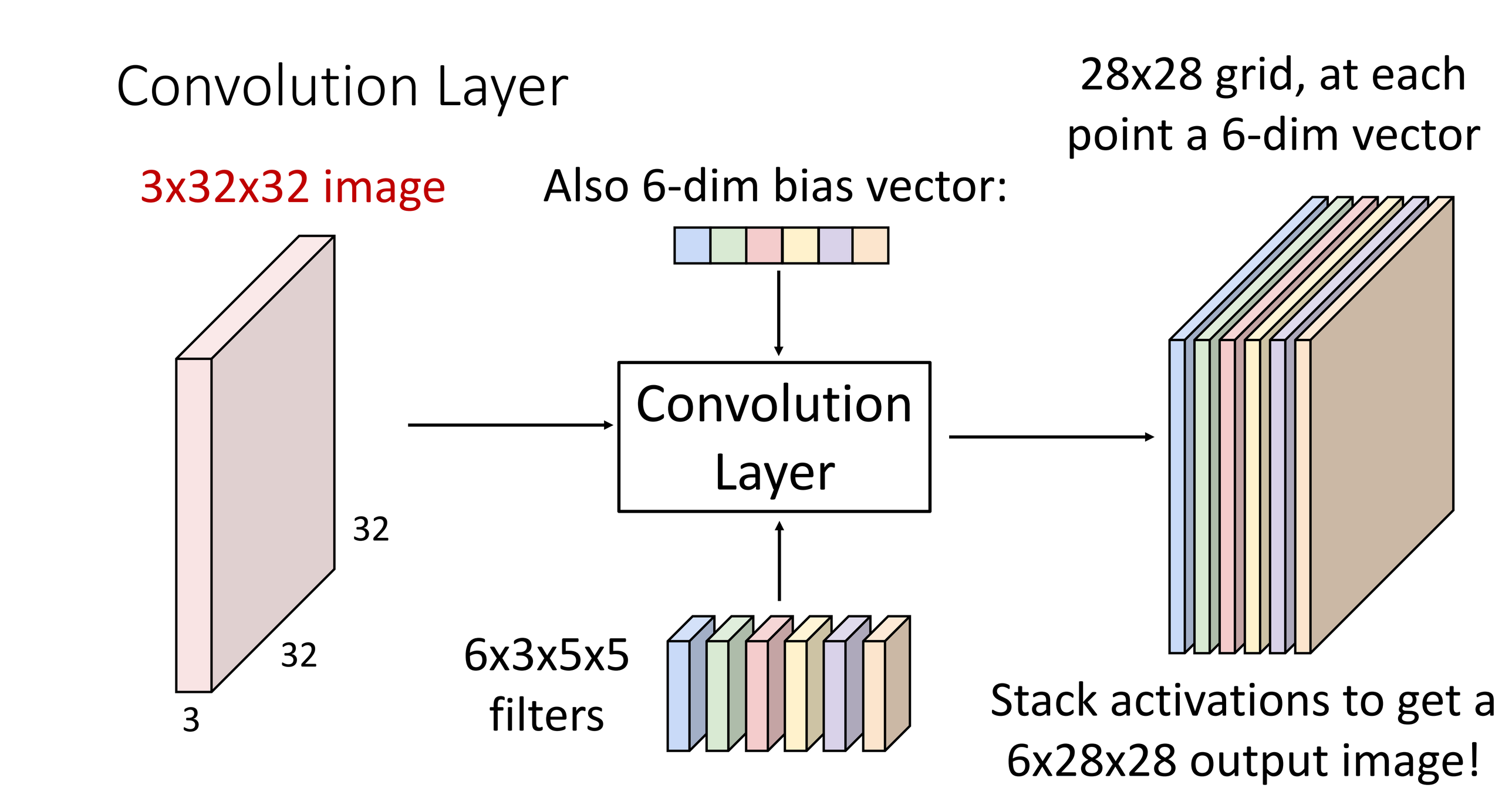
# Convolution Layer



# Convolution Layer

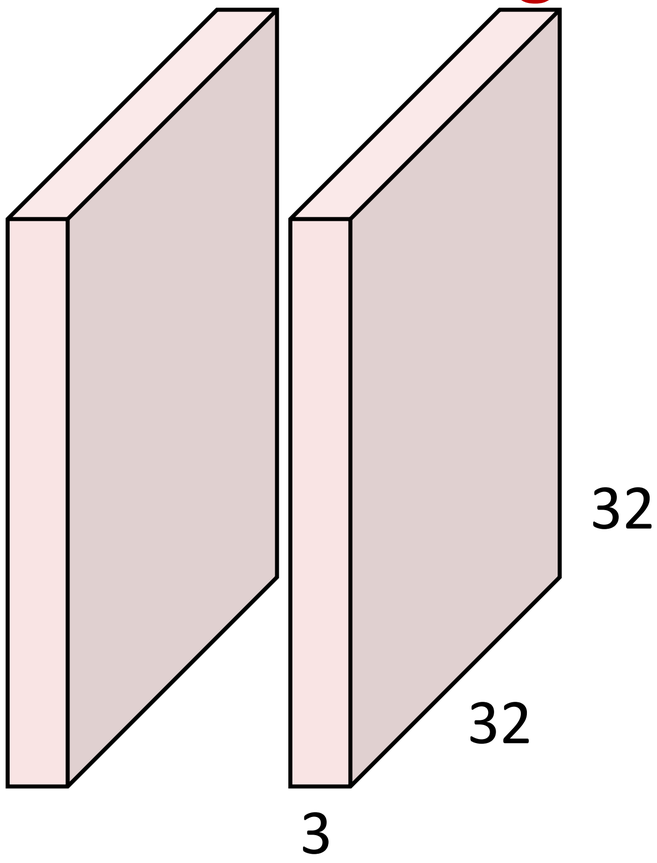


# Convolution Layer

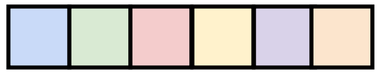


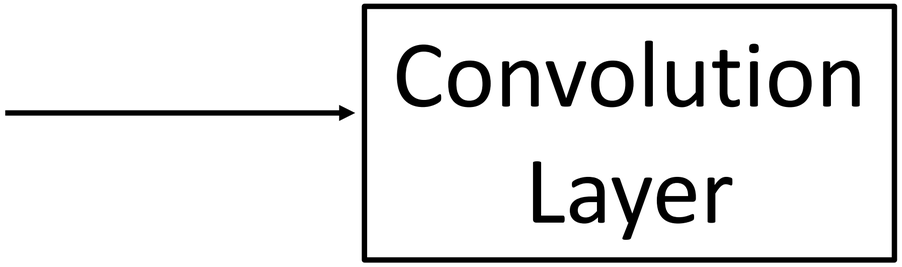
# Convolution Layer

2x3x32x32
Batch of images

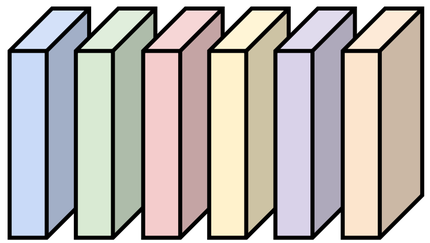



Also 6-dim bias vector:

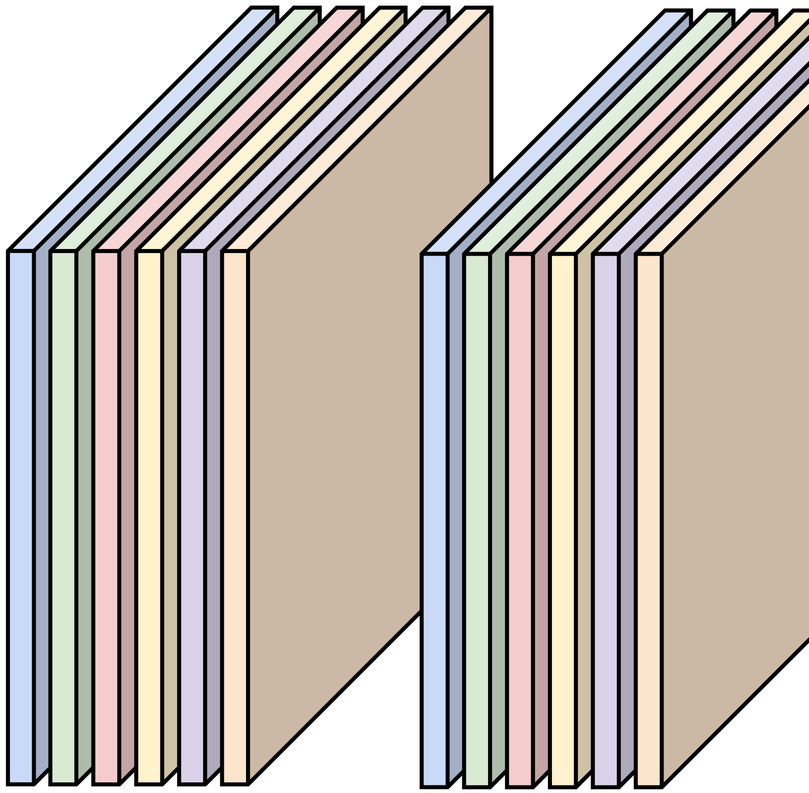





6x3x5x5
filters

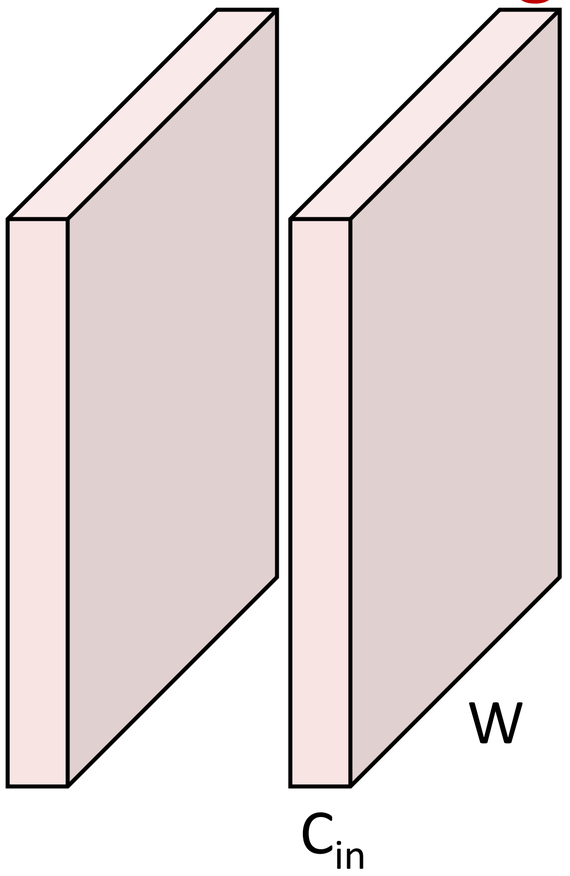


2x6x28x28
Batch of outputs

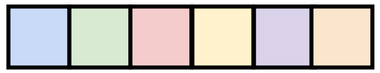


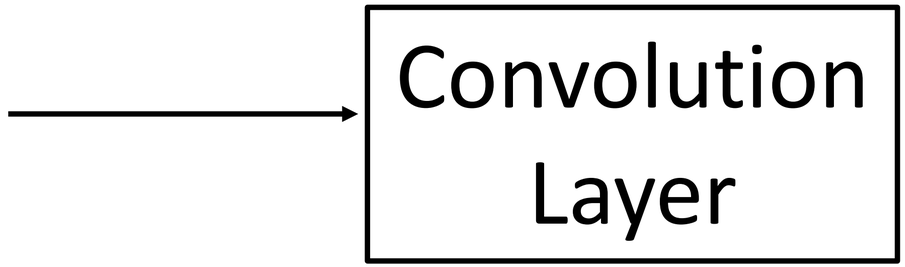
# Convolution Layer

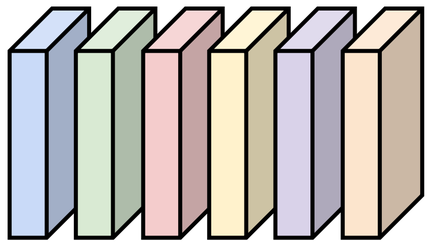
$N \times C_{in} \times H \times W$
Batch of images




Also $C_{out}$-dim bias vector:

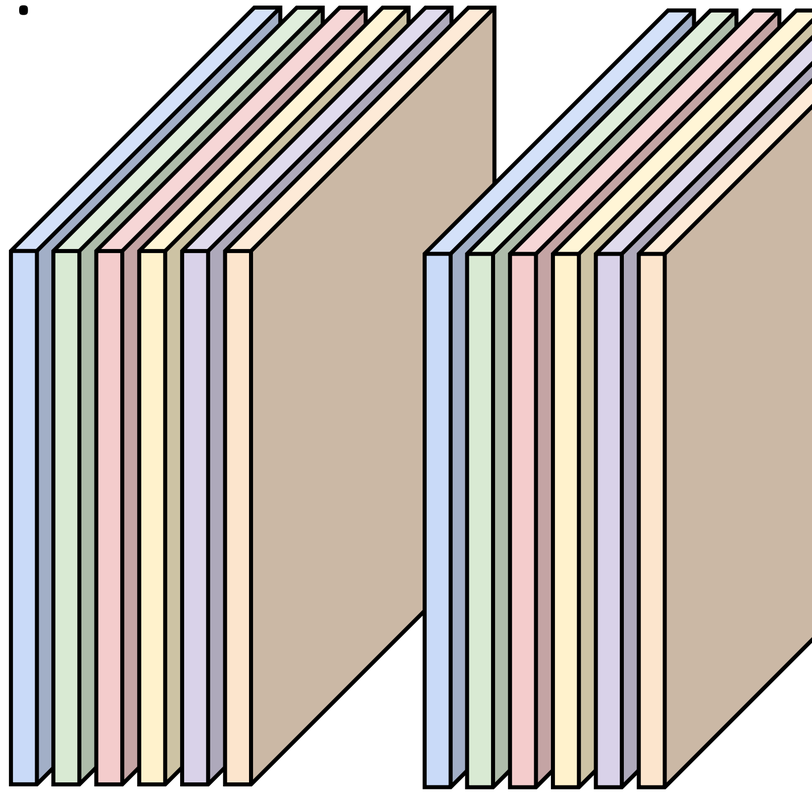




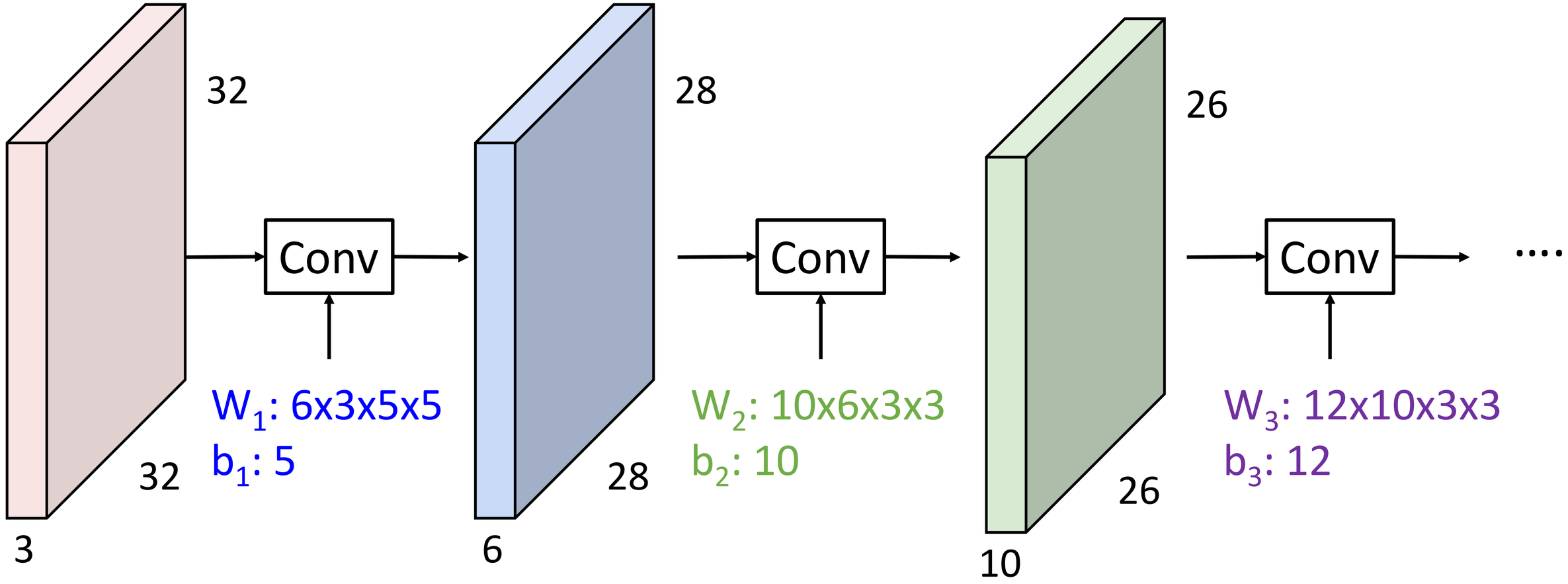



$C_{out} \times C_{in} \times K_w \times K_h$
filters

$N \times C_{out} \times H' \times W'$
Batch of outputs



# Stacking Convolutions




Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

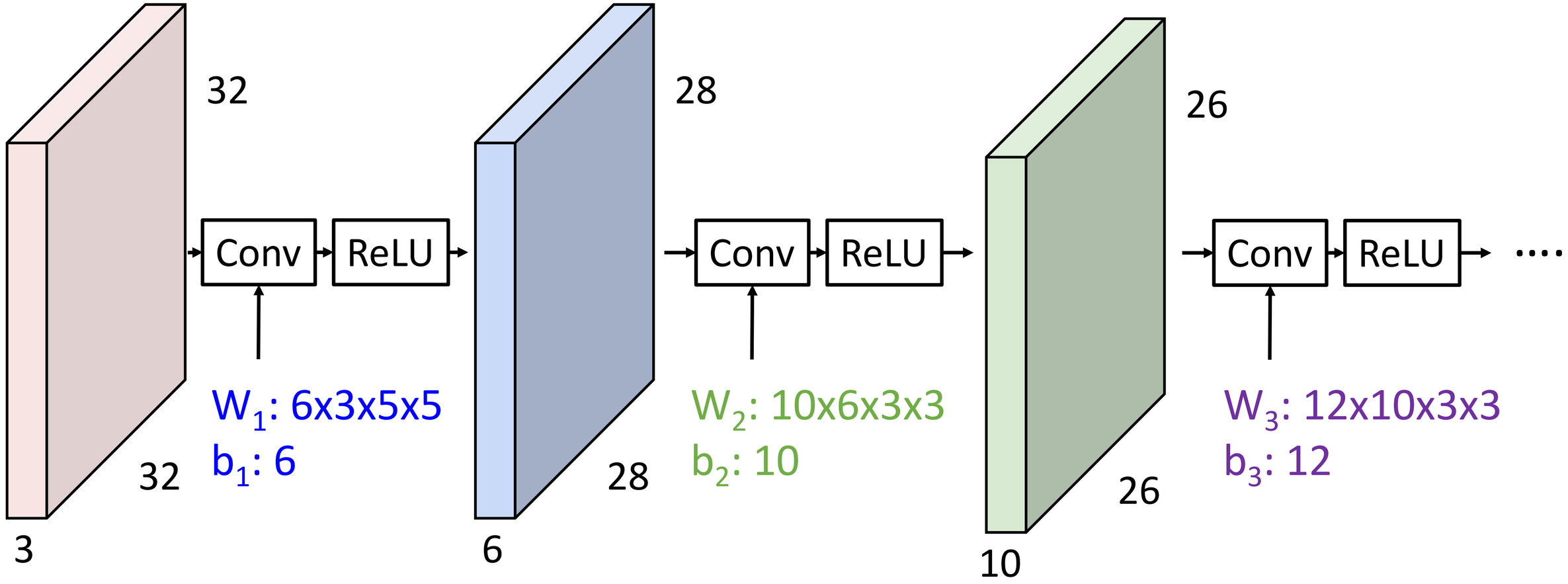
Second hidden layer:
N x 10 x 26 x 26

# Stacking Convolutions

**Q**: What happens if we stack two convolution layers?

**A**: We get another convolution!

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




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

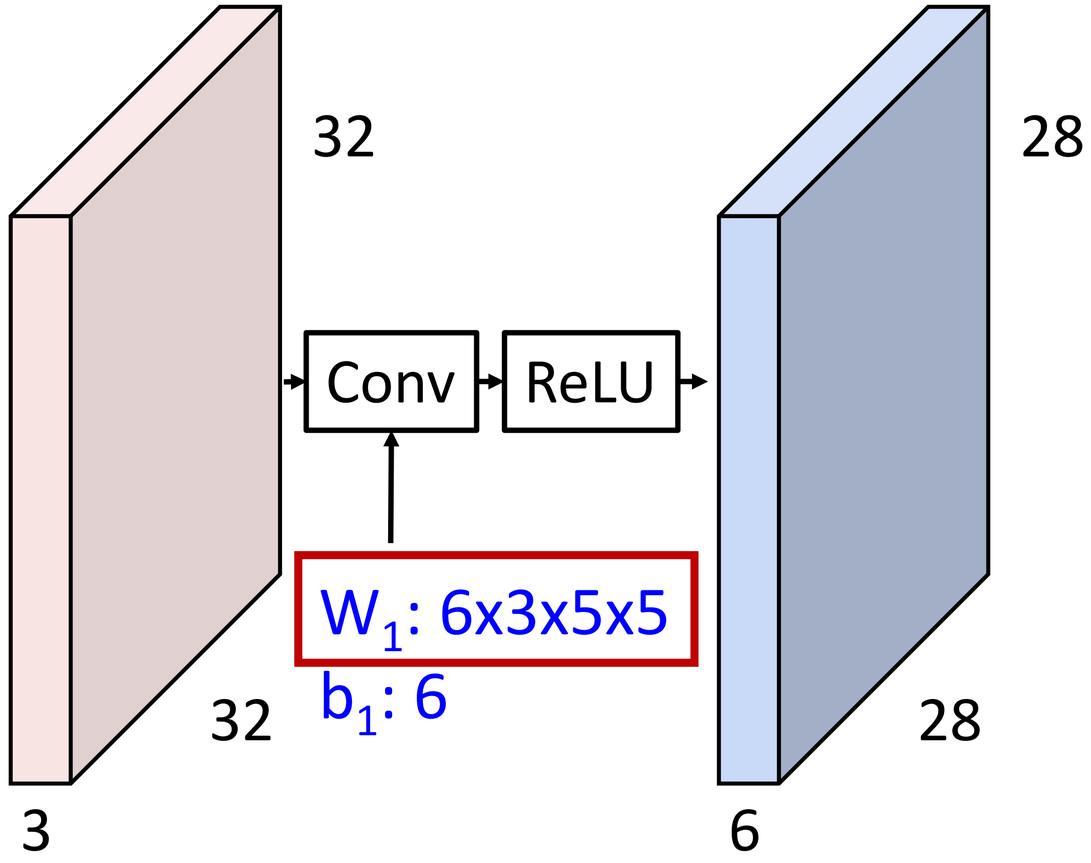
# **Convolutional** Neural Networks


**VGG-16** Network

# **Backward Pass** for Some **Common Layers**

Convolutional layer

# What do convolutional filters learn?

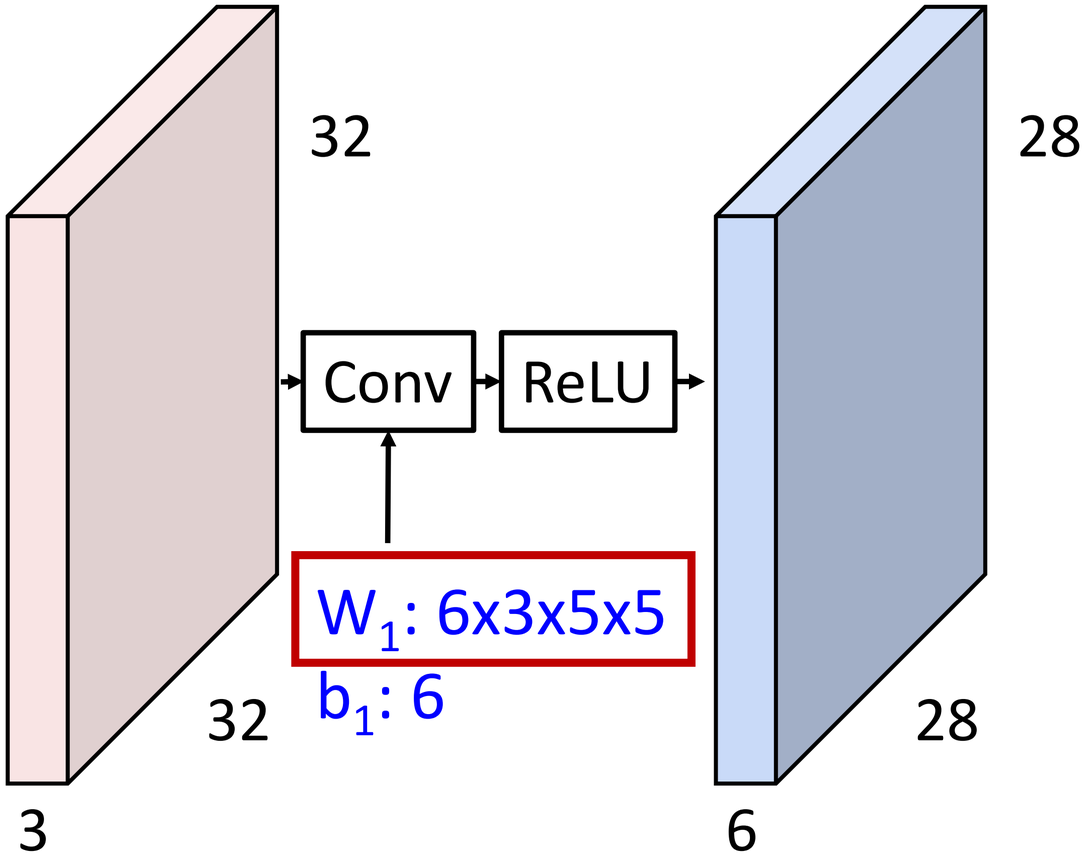



Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Linear classifier: One template per class

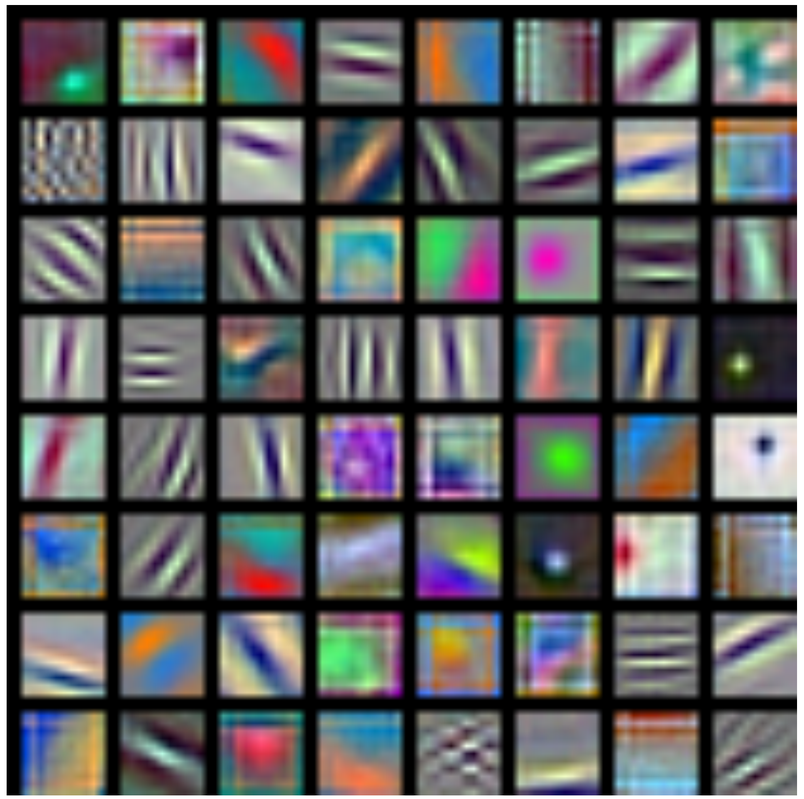
# What do convolutional filters learn?




Input:
N x 3 x 32 x 32

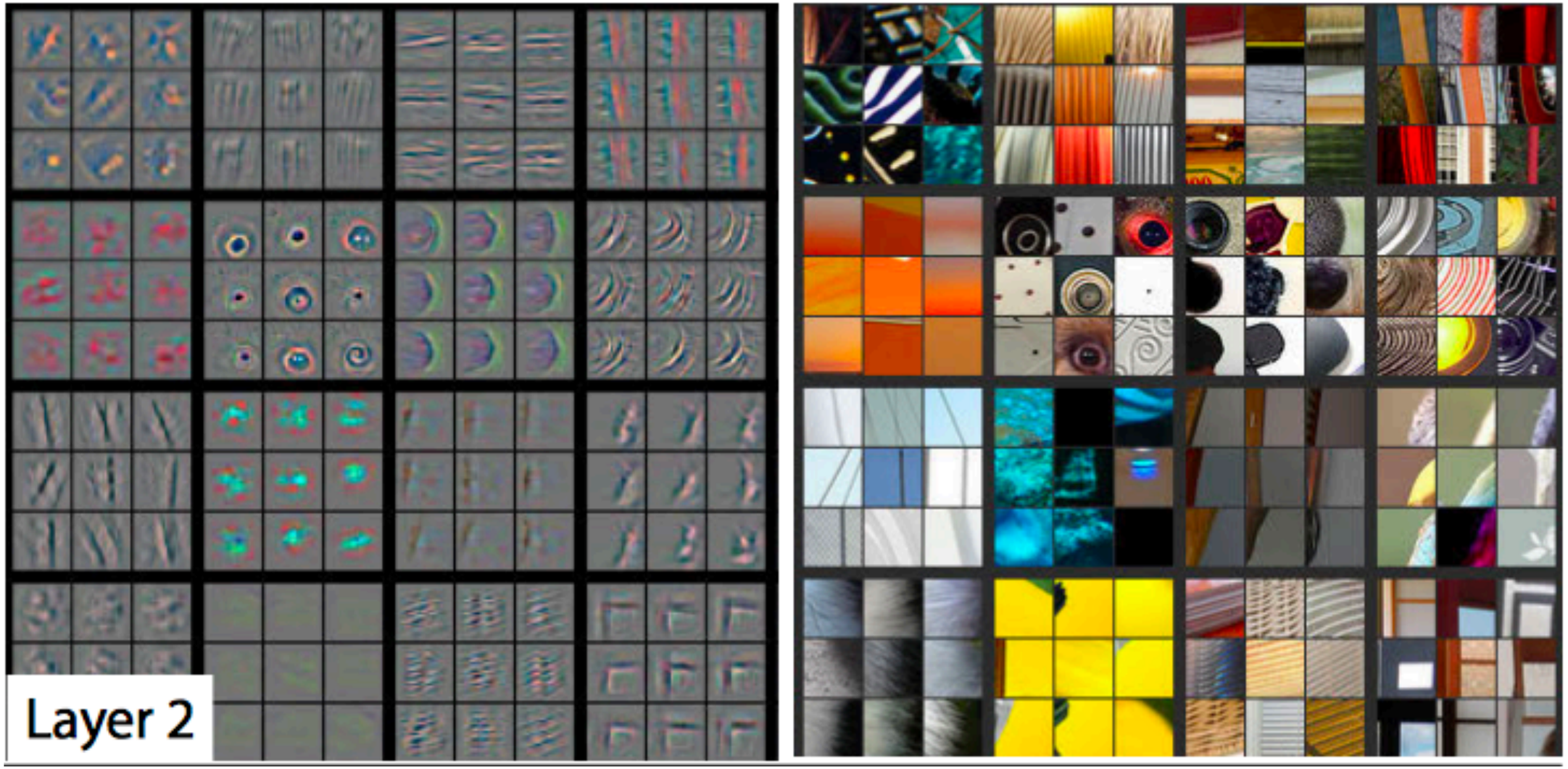
First hidden layer:
N x 6 x 28 x 28

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

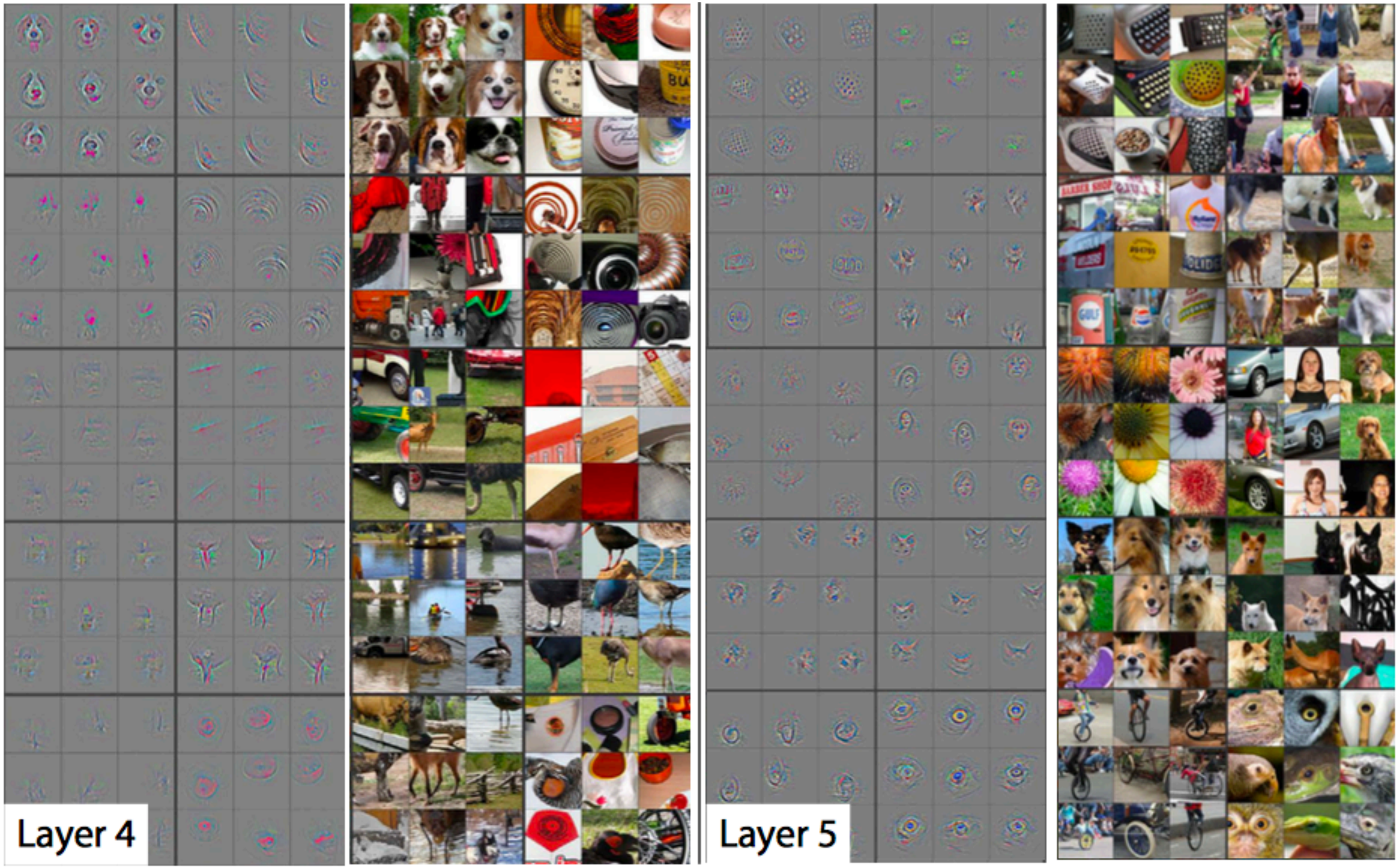


AlexNet: 64 filters, each 3x11x11

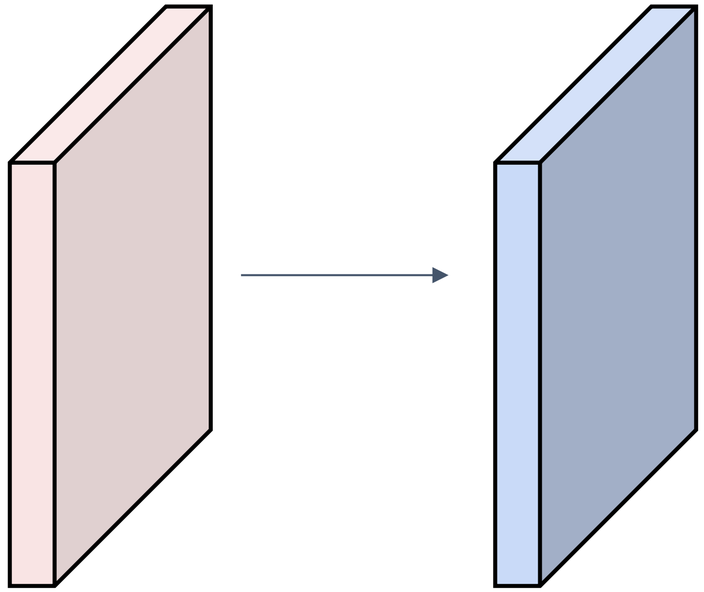
# What **filters** do networks learn?




[ Zeiler and Fergus, 2013 ]


Layer 4
Layer 5

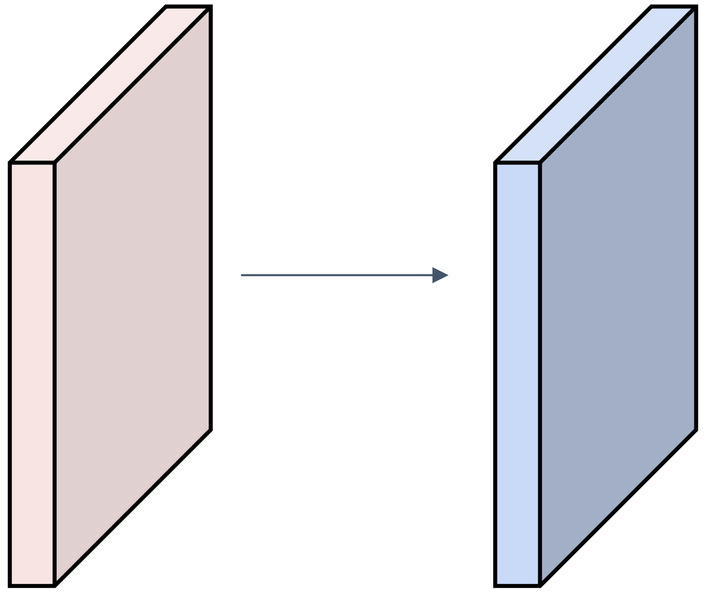
# Convolution Example



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

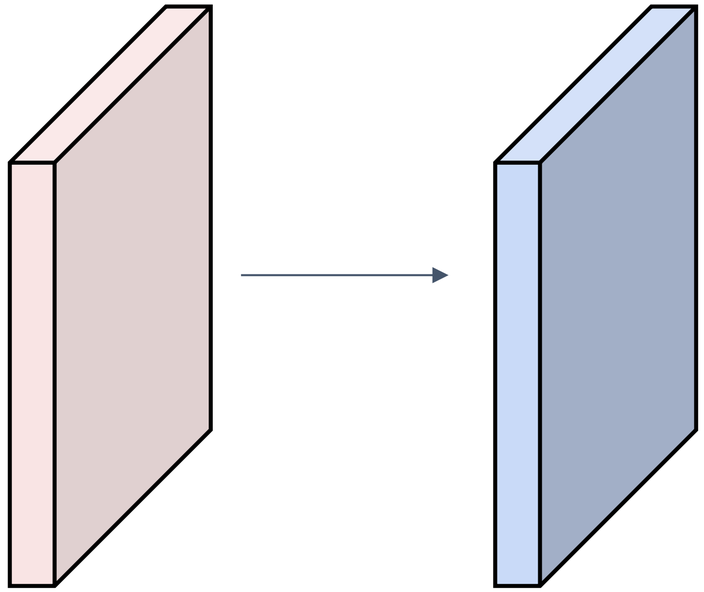
10 5x5 filters with stride 1, pad 2

Output volume size:

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

10 x 32 x 32

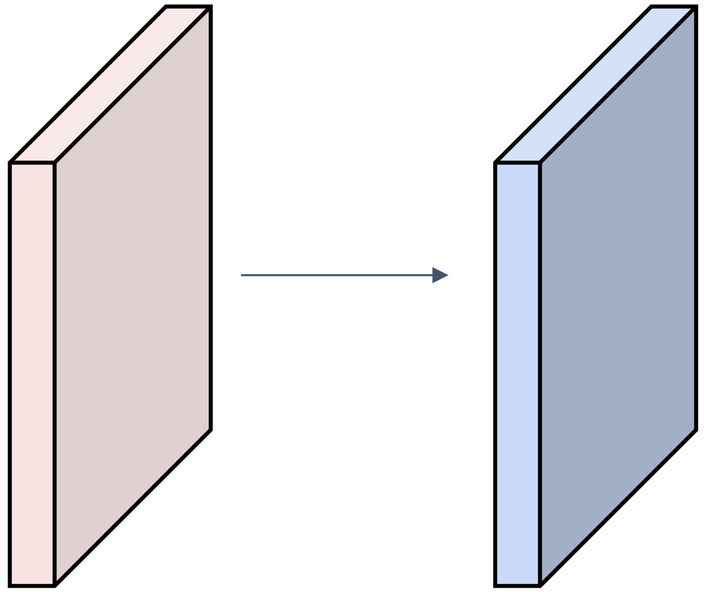
# Convolution Example



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** **5**x**5** filters with stride 1, pad 2

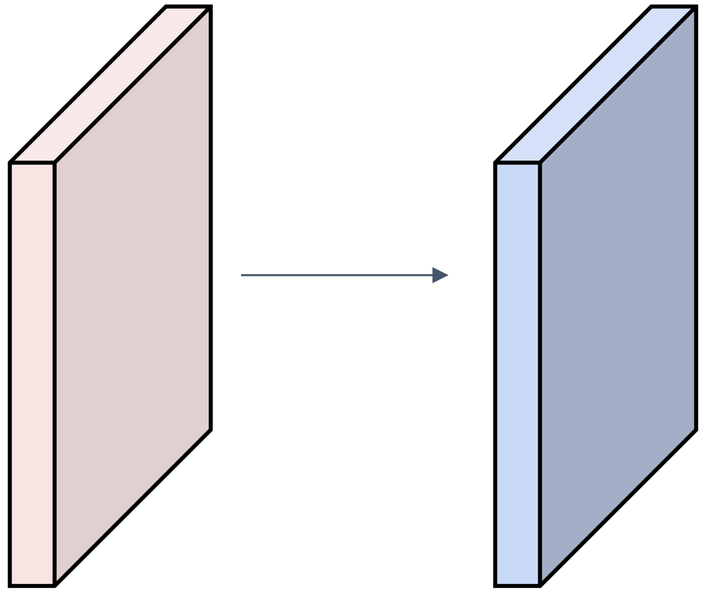
Output volume size: 10 x 32 x 32

Number of learnable parameters: **760**

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

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

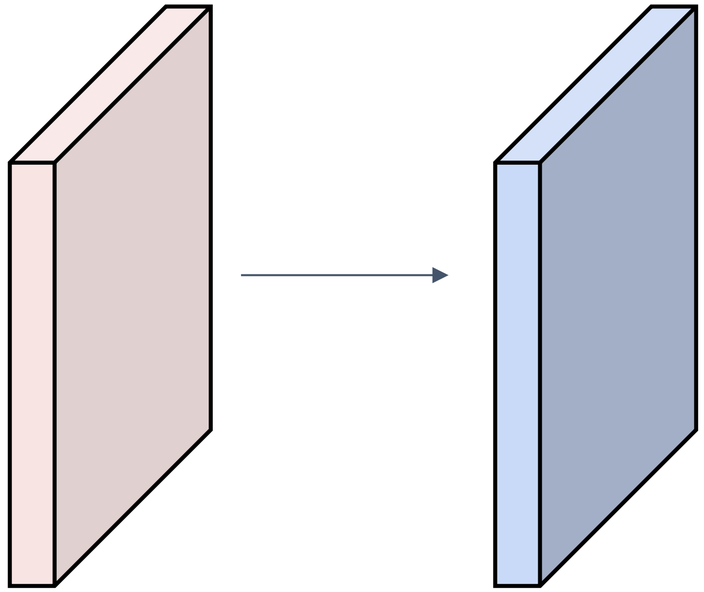
# Convolution Example



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: ?

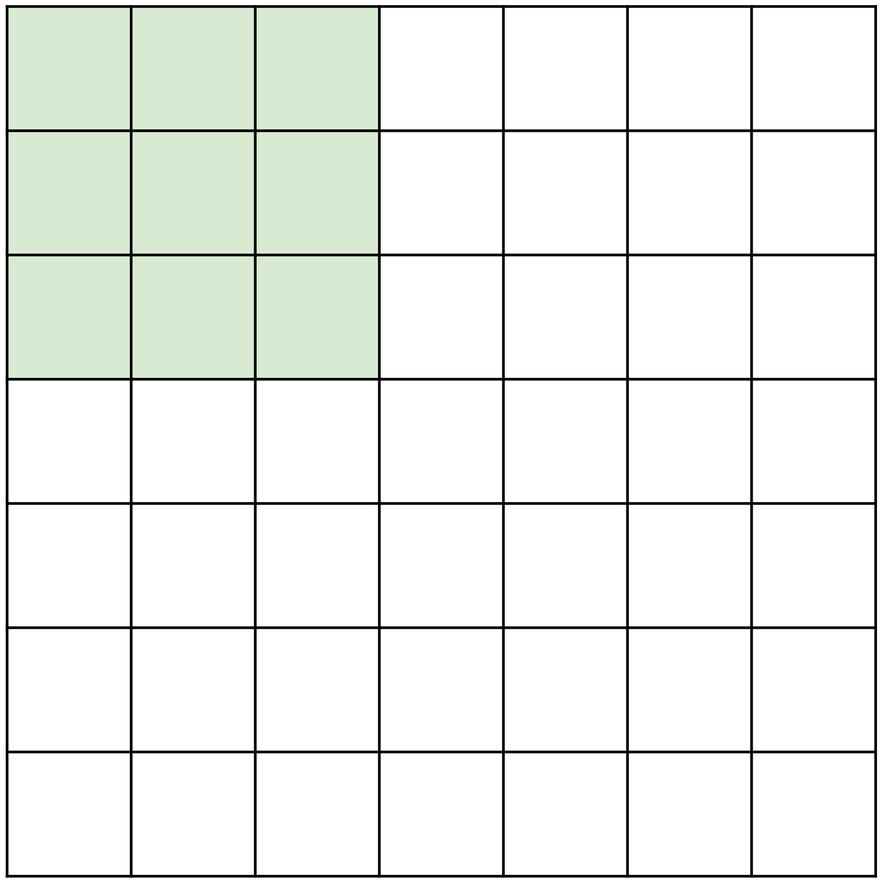
# Convolution Example



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

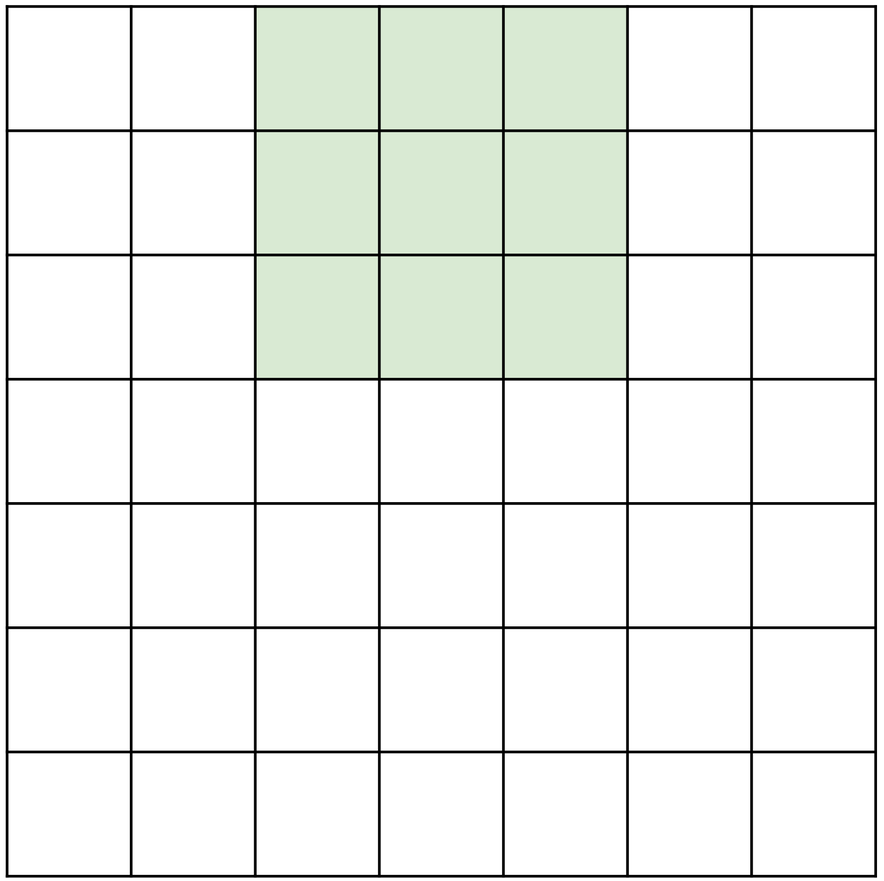
Output volume size: **10 x 32 x 32**
Number of learnable parameters: 760
Number of multiply-add operations: **768,000**
**10*32*32** = 10,240 outputs; each output is the inner product of two **3**x**5x5** tensors (75 elems); total = 75*10240 = **768K**

# Strided Convolution



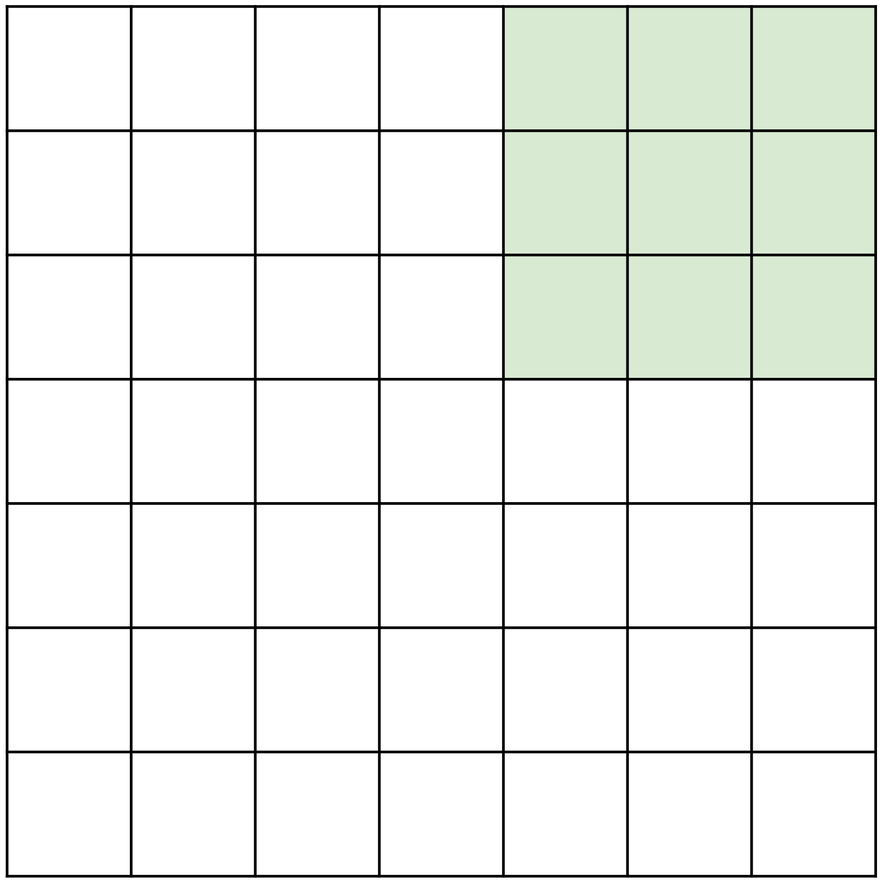
Input: 7x7
Filter: 3x3
Stride: 2

# Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

# Strided Convolution



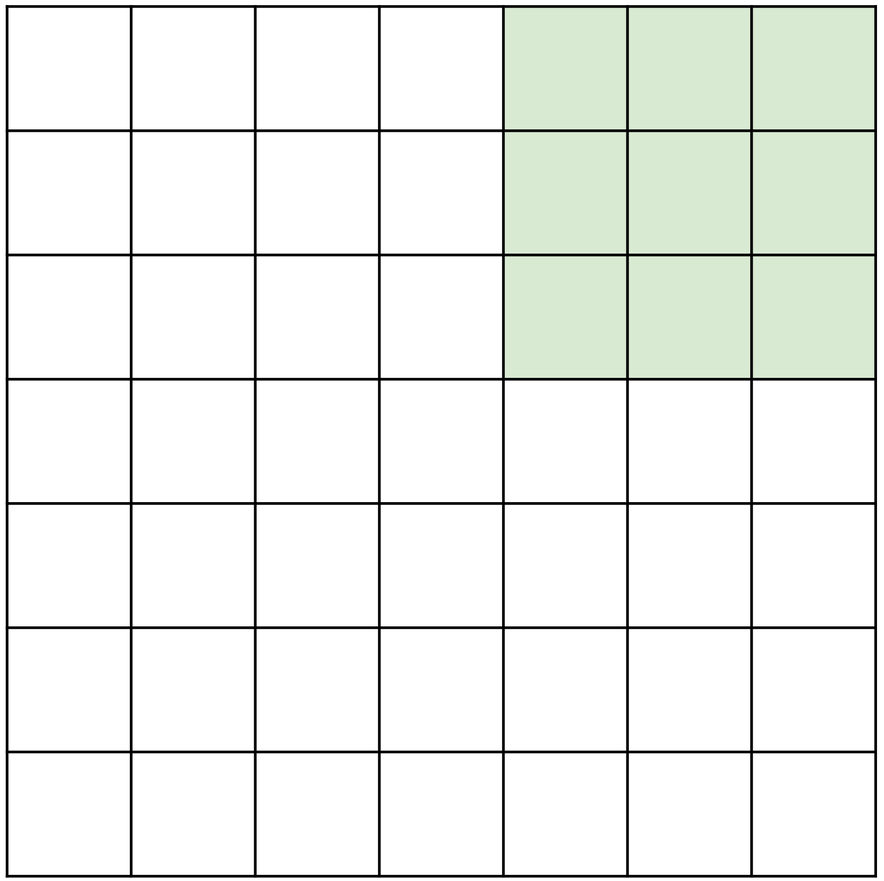
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

# Strided Convolution

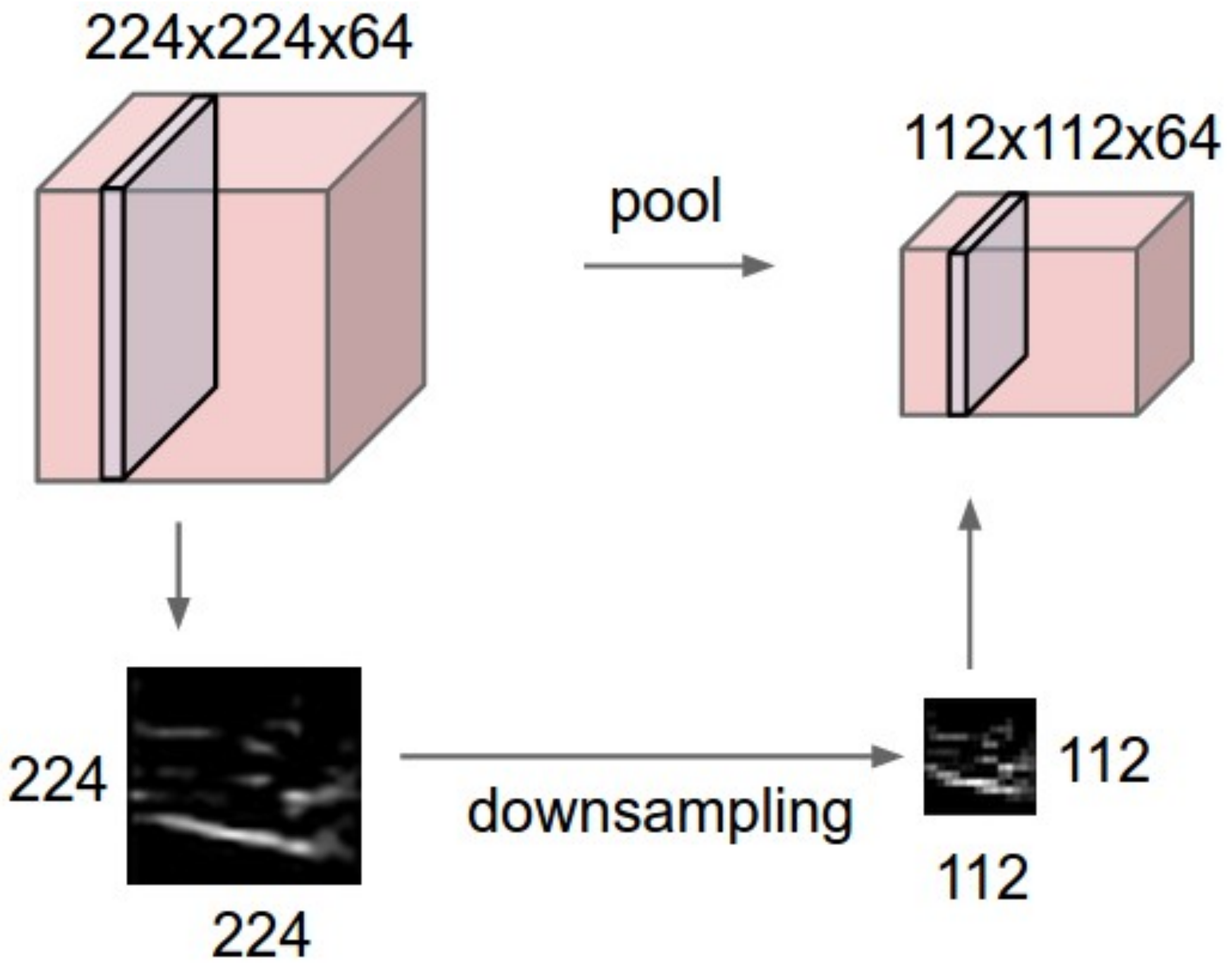


Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

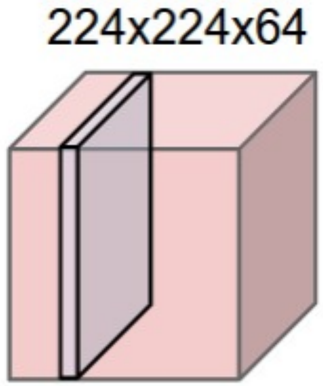
In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: (W – K + 2P) / S + 1

# **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

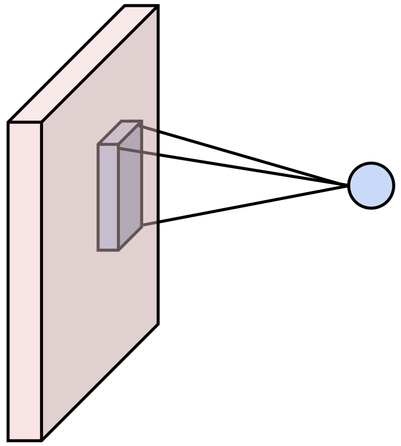
| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

Introduces **invariance** to small spatial shifts

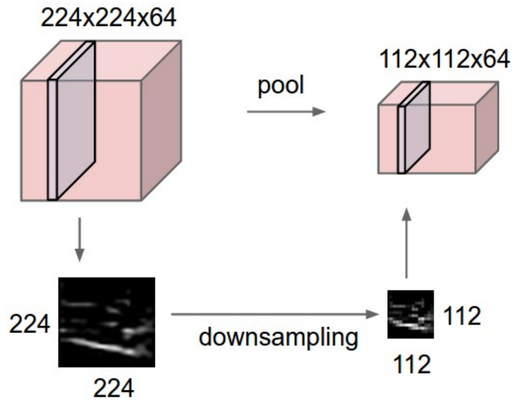
No learnable parameters!

# Components of a Convolutional Network

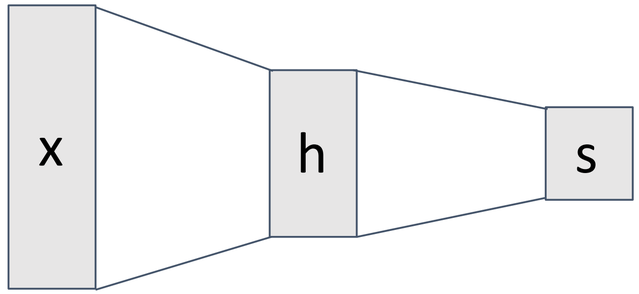
## Convolution Layers



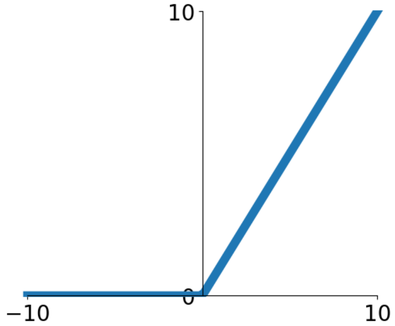
## Pooling Layers




## Fully-Connected Layers




## Activation Function



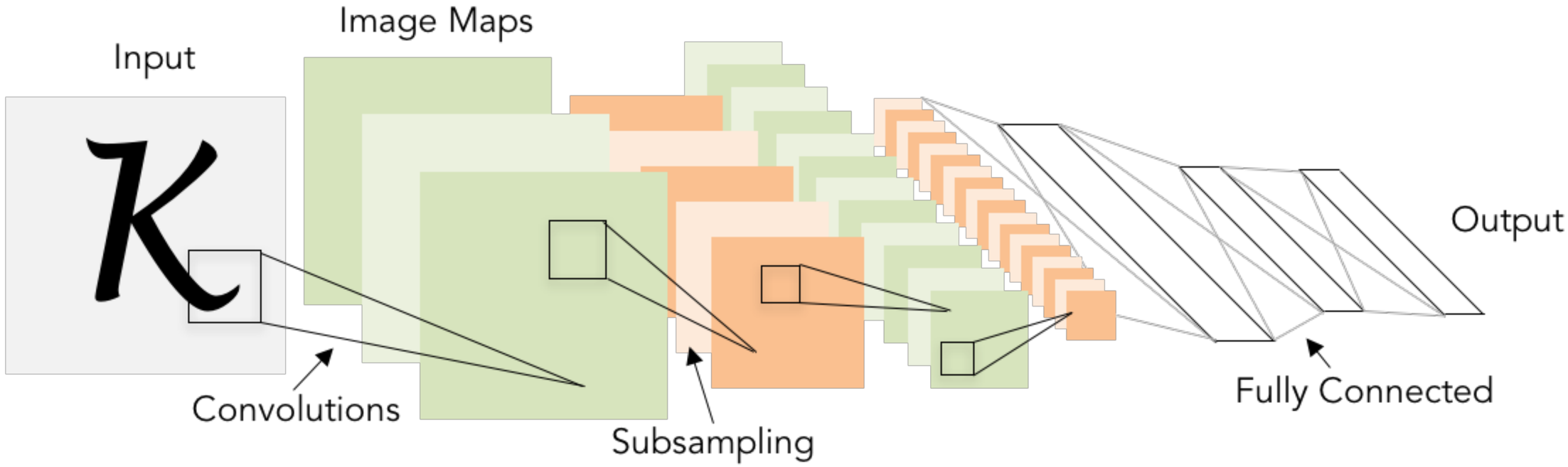

## Normalization

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

# Convolutional Networks

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

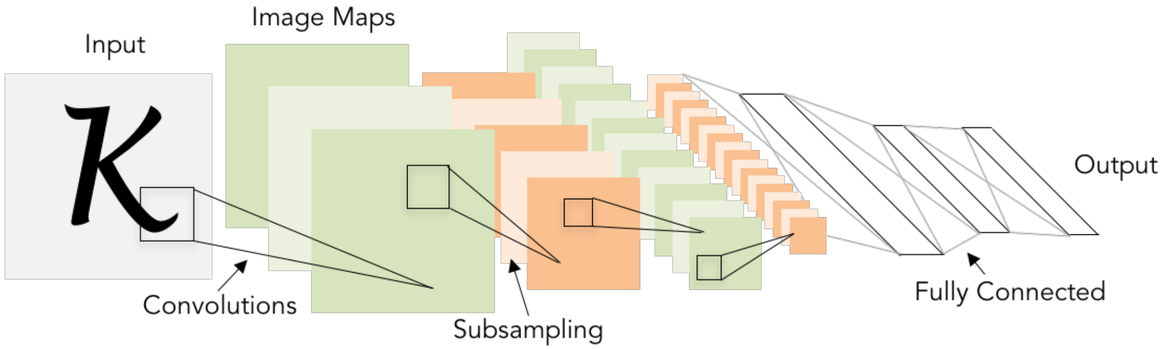
Example: LeNet-5




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)

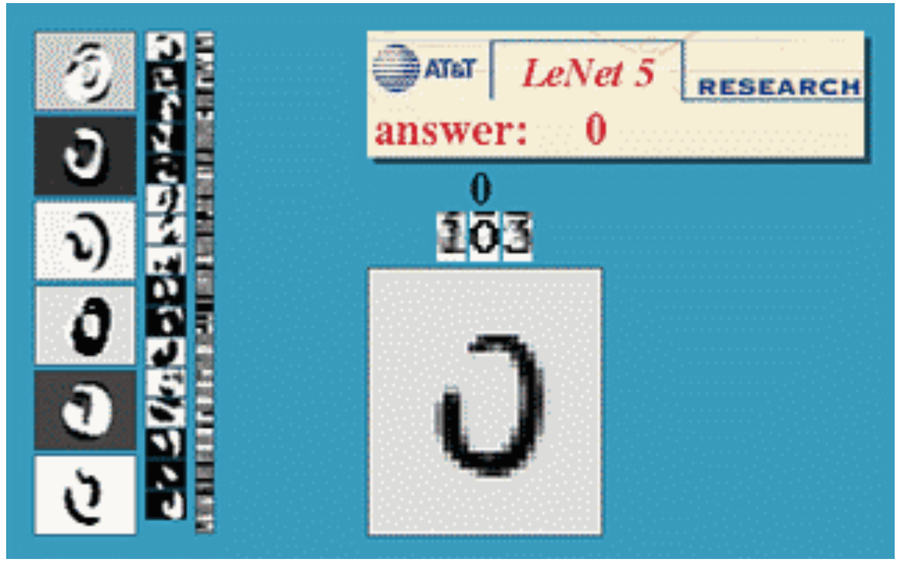
Number of channels **increases**
(total "volume" is preserved!)

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)

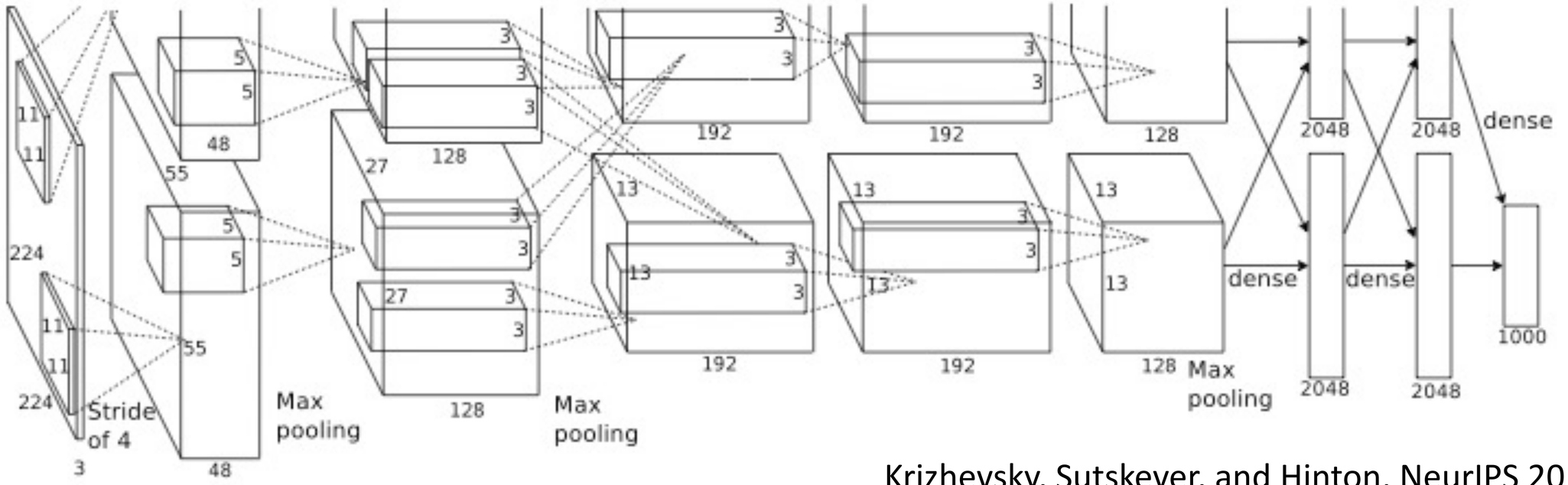



Digit recognition, AT&T labs
http://www.research.att.com/~yann/

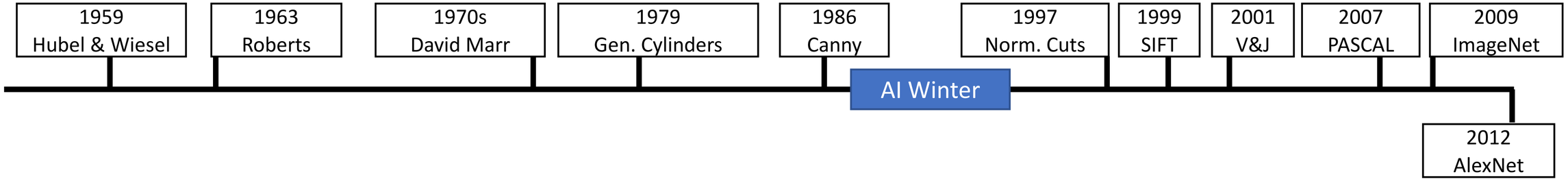

License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

Yann LeCun

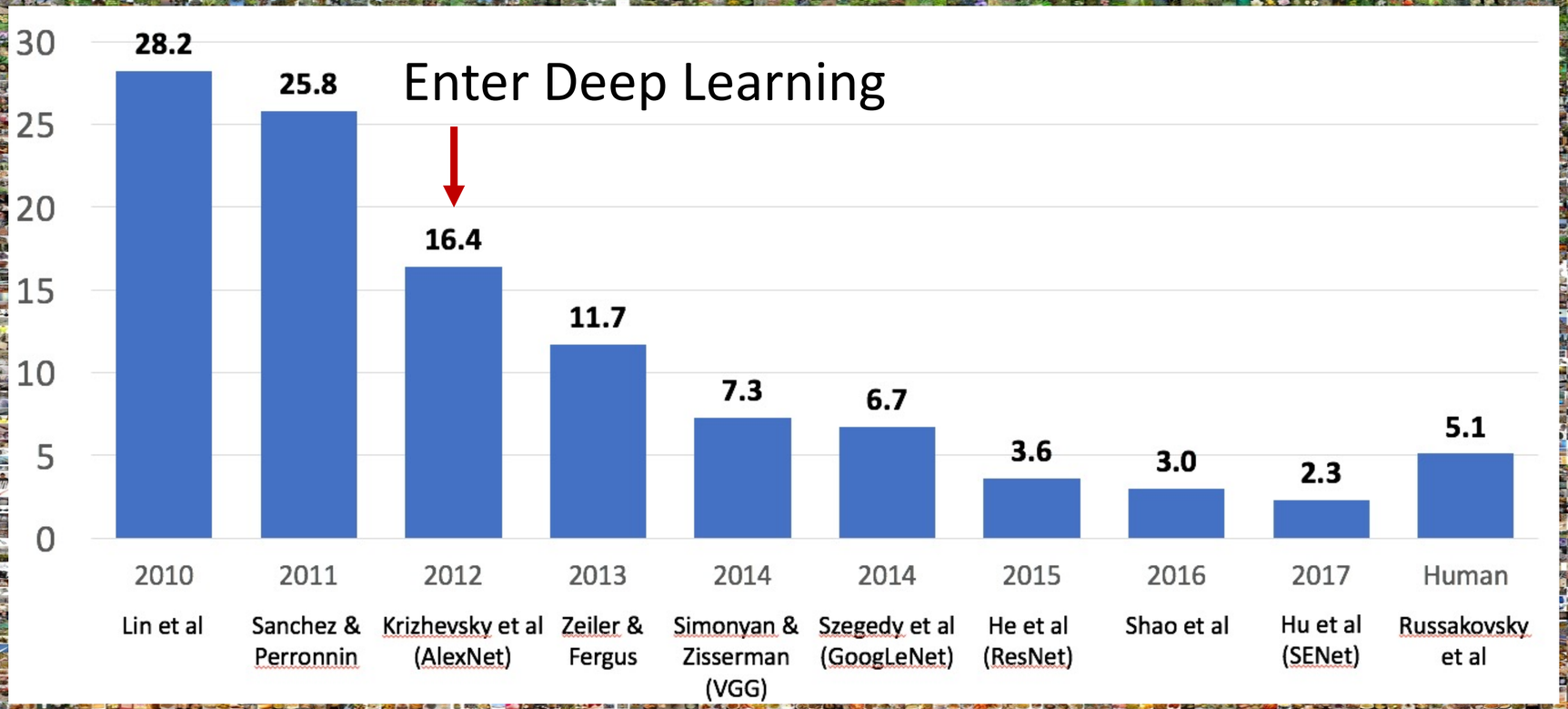
# AlexNet: Deep Learning Goes Mainstream

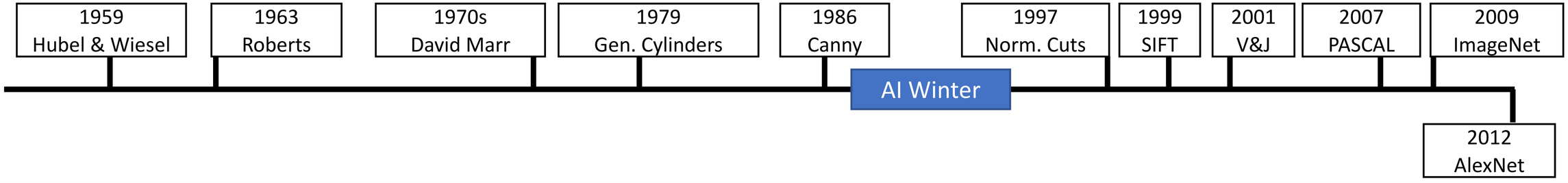



Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

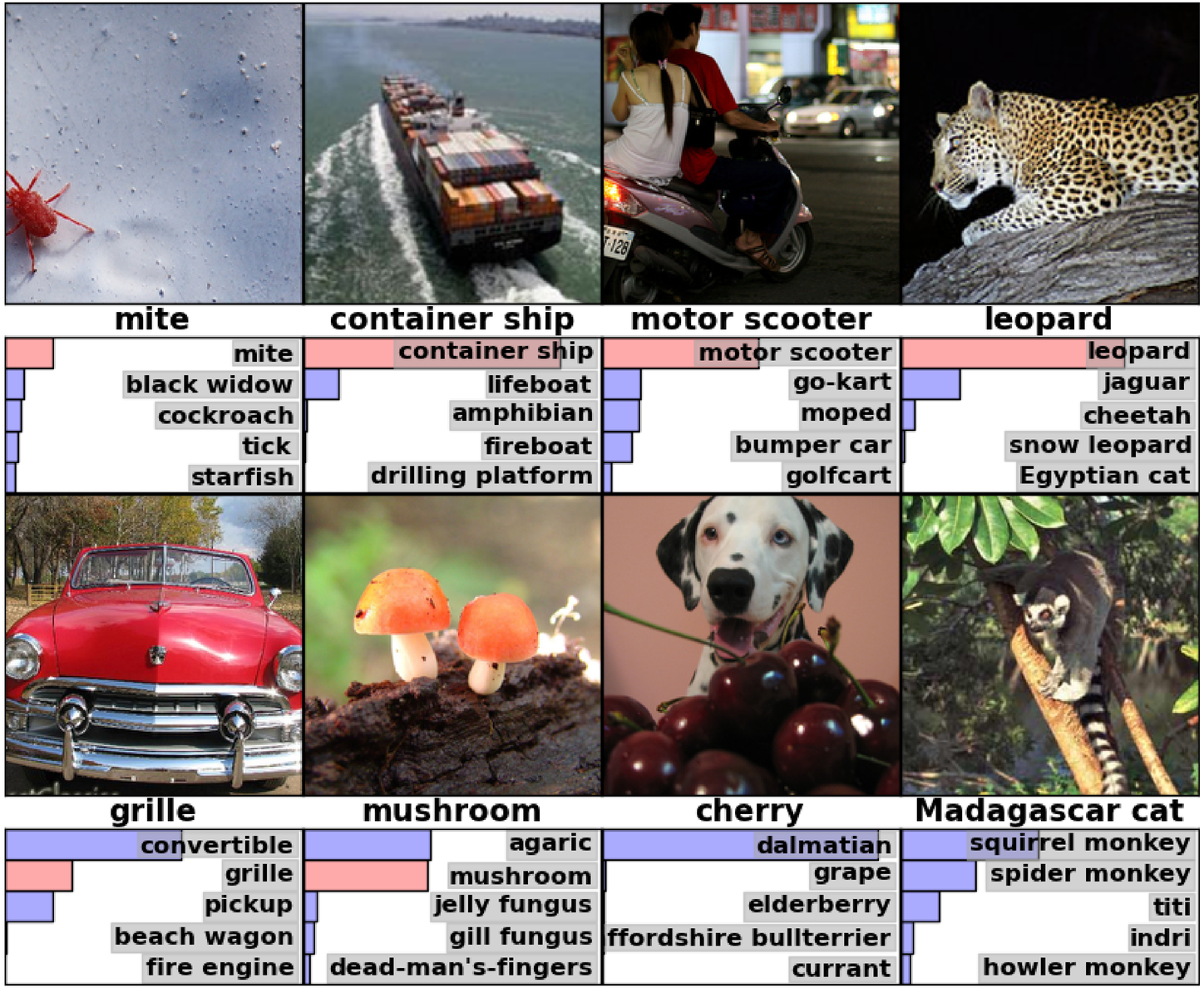


# IMAGENET Large Scale Visual Recognition Challenge

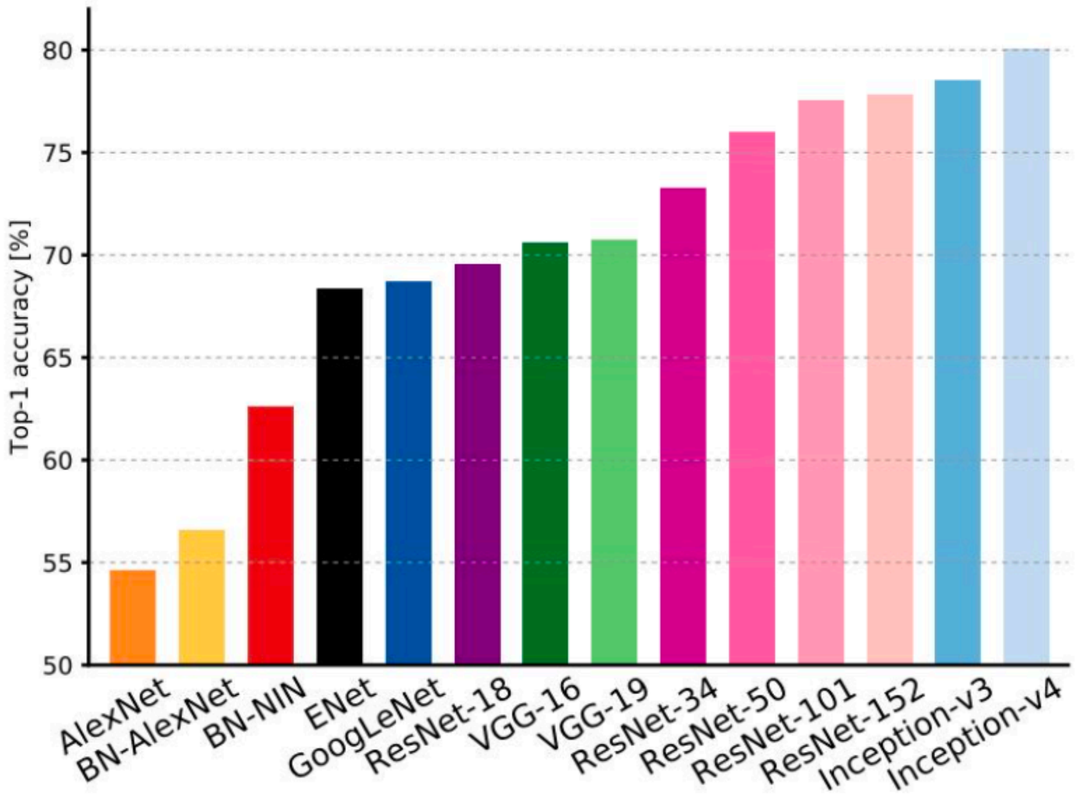


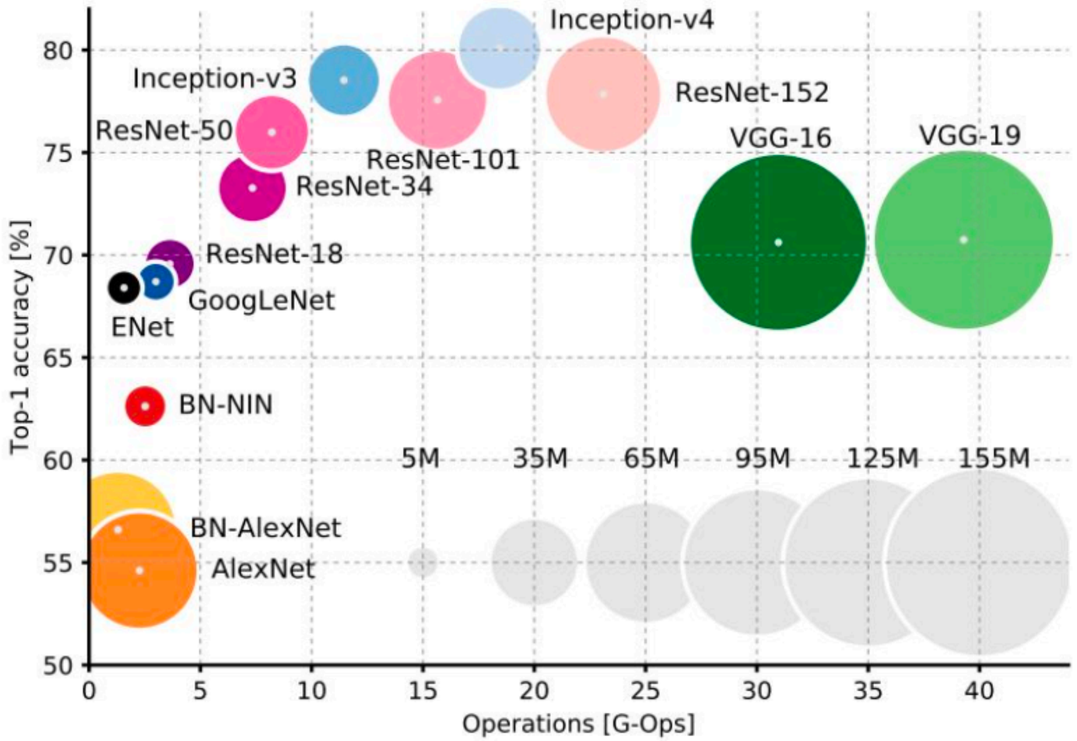



# AlexNet on ImageNet



# Comparing **Complexity**




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

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

# More on Neural Networks

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


Hans could get 89% of the math questions right

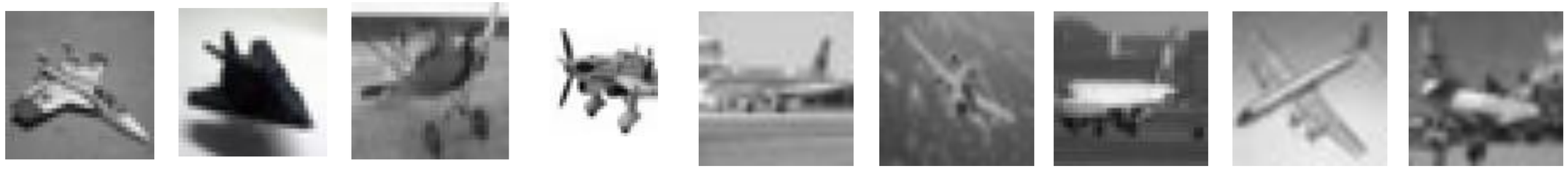
# Training Neural Nets: **Clever** Hans

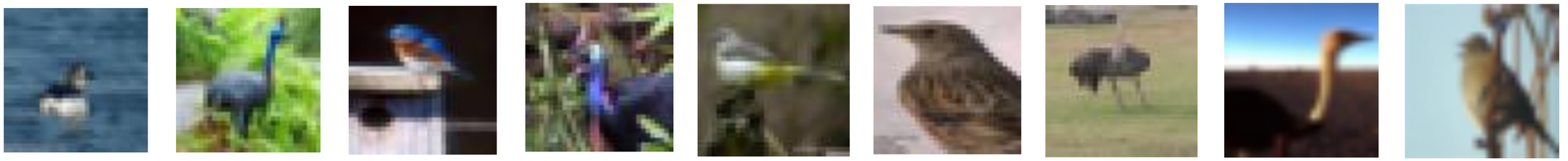

The course was **smart**, just not in the way van Osten thought!

Hans could get 89% of the math questions right

# Training Neural Nets

- Suppose we build a neural net classifier for the following dataset

plane 

bird 

- How should the network classify this image? Query: $\mathbf{x}_q =$ 

Please fill out

**Student Evaluations**

(on Canvas)

# 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