

*CS 1674: Intro to Computer Vision*

# Convolutional Neural Networks

Prof. Adriana Kovashka  
University of Pittsburgh  
October 20, 2020

# Plan for the next few lectures

---

Why (convolutional) neural networks?

Neural network basics

- Architecture and biological inspiration
- Loss functions
- Optimization / gradient descent
- Training with backpropagation

Convolutional neural networks (CNNs)

- Special operations
- Common architectures

Practical matters

- Tips and tricks for training
- Transfer learning
- Software packages

Understanding CNNs

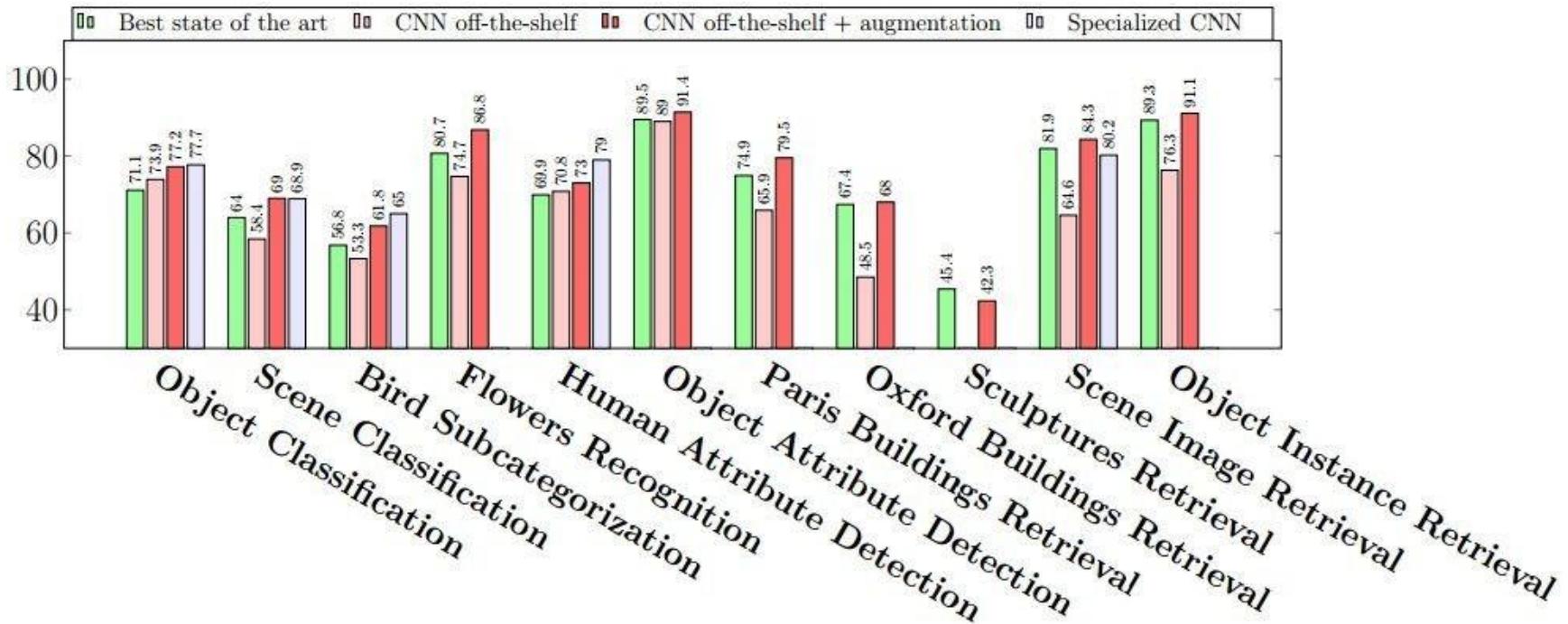
- Visualization
- Synthesis / style transfer
- Breaking CNNs

# Neural network basics



# Why (convolutional) neural networks?

State of the art performance on many problems  
Most papers in recent vision conferences use  
deep neural networks



# ImageNet Challenge 2012

---



[Deng et al. CVPR 2009]

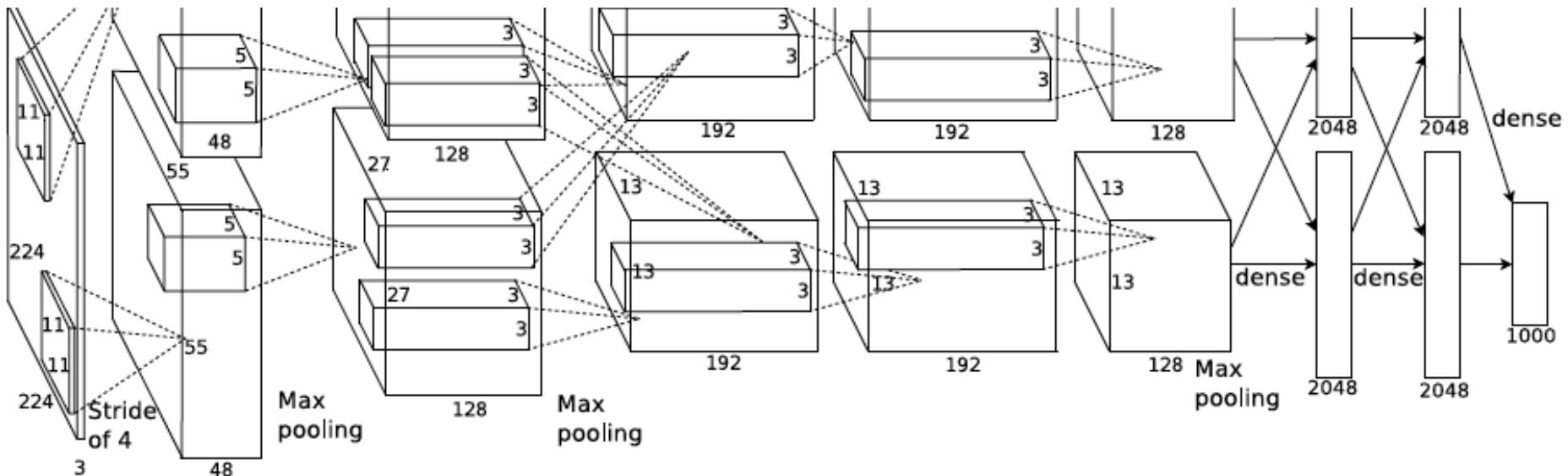
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Mechanical Turk
- Challenge: 1.2 million training images, 1000 classes

A. Krizhevsky, I. Sutskever, and G. Hinton, [ImageNet Classification with Deep Convolutional Neural Networks](#), NIPS 2012

# ImageNet Challenge 2012



- AlexNet: Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data ( $10^6$  vs.  $10^3$  images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Better regularization for training (DropOut)



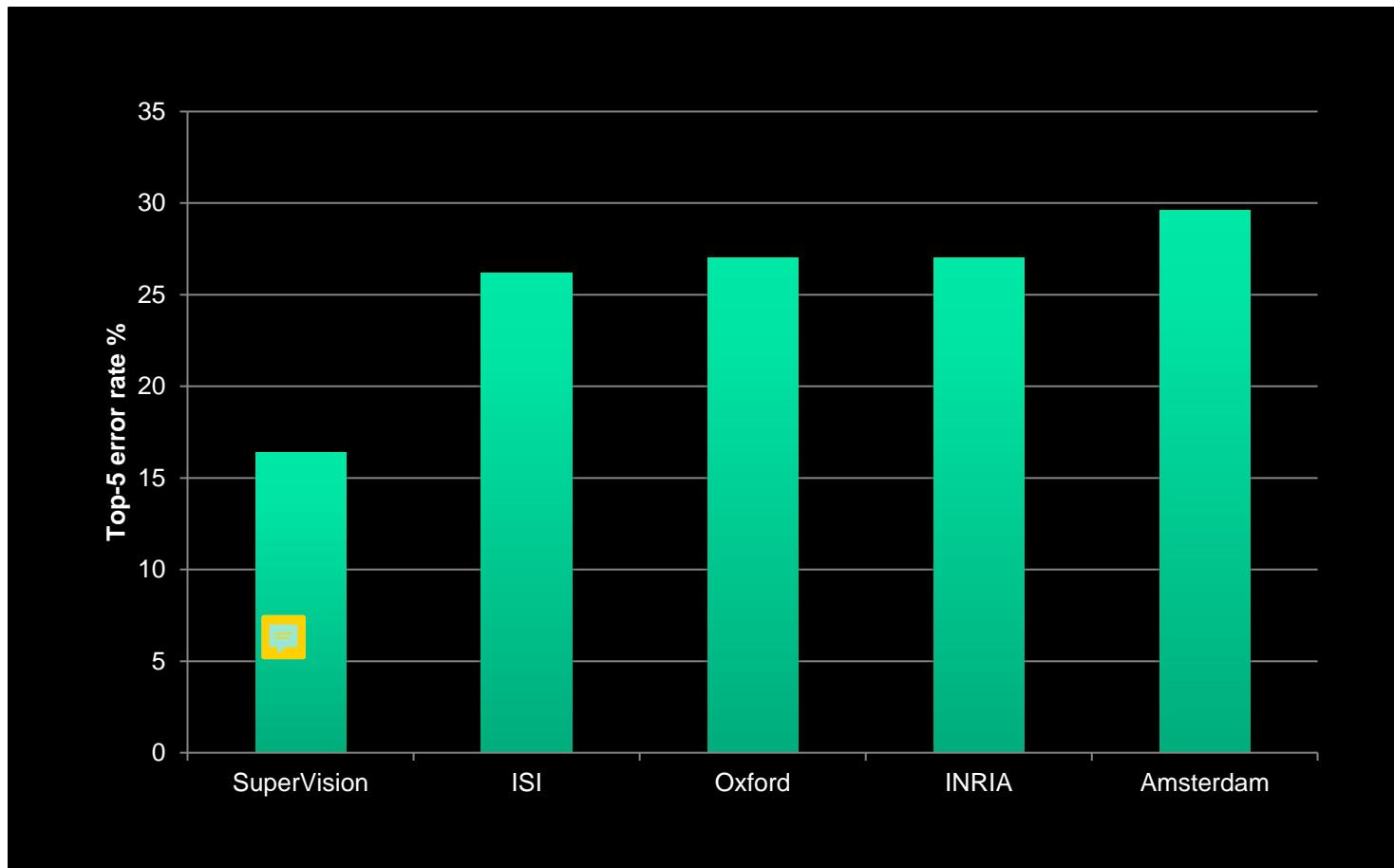
A. Krizhevsky, I. Sutskever, and G. Hinton, [ImageNet Classification with Deep Convolutional Neural Networks](#), NIPS 2012

# ImageNet Challenge 2012

---

Krizhevsky et al. -- **16.4% error** (top-5) 

Next best (non-convnet) – **26.2% error**

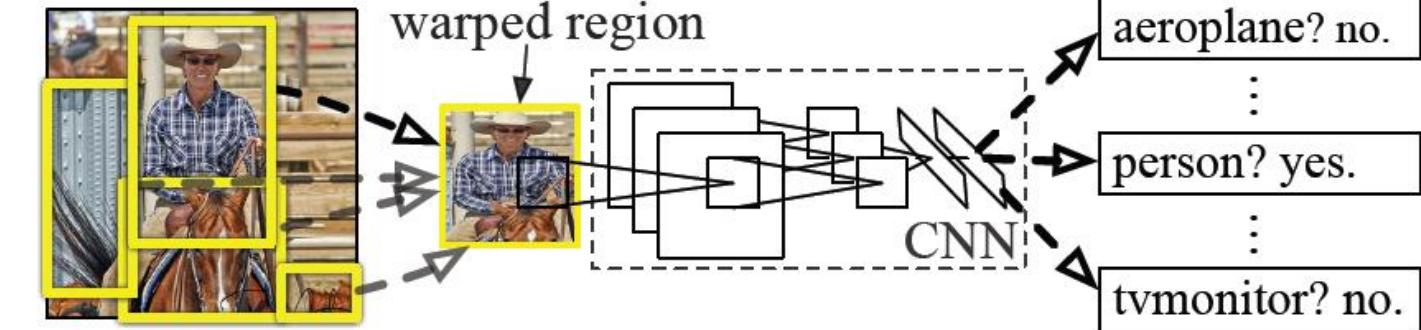


# Easy win, always? Example: detection

## R-CNN: *Regions with CNN features*



1. Input image



2. Extract region proposals (~2k)

3. Compute CNN features

4. Classify regions

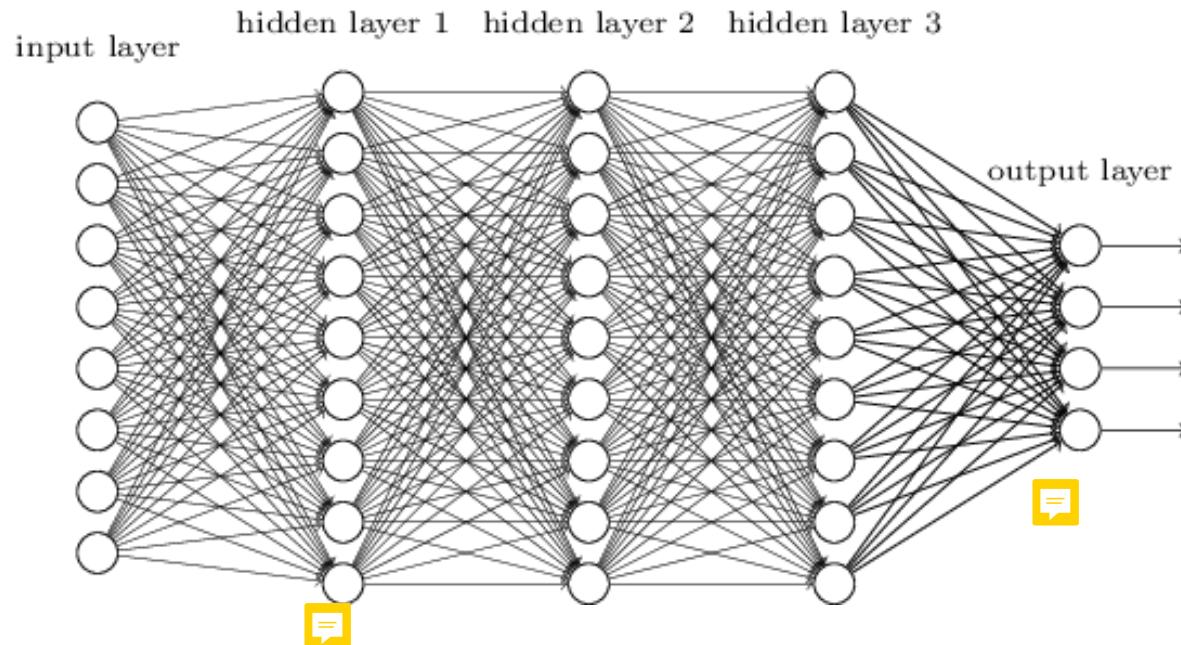
**Object detection system overview.** Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. **R-CNN achieves a mean average precision (mAP) of 53.7% on PASCAL VOC 2010.** For comparison, Uijlings et al. (2013) report 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. **The popular deformable part models perform at 33.4%.**

R. Girshick, J. Donahue, T. Darrell, and J. Malik, [Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation](#), CVPR 2014.

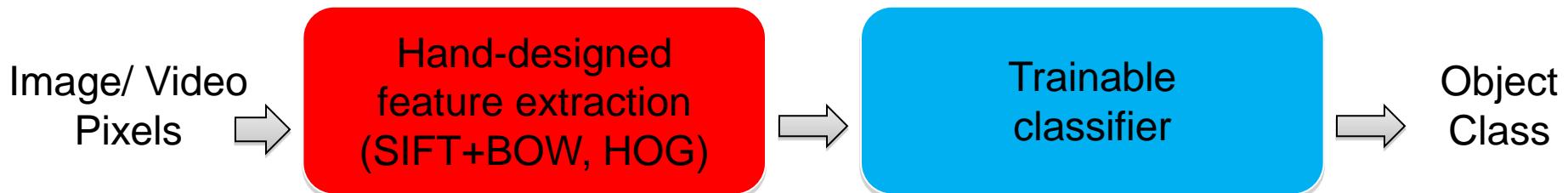
# What are CNNs?

---

- Convolutional neural networks (CNNs) are a type of *neural network* with layers that perform special operations
- Used in vision but also in NLP, biomedical etc.
- Often *deep*, but usually *not* fully connected



# Traditional Recognition Approach



- Features are key to recent progress in recognition, but research shows they're flawed... Where next?



Fig. 1: An image from PASCAL and a high scoring car detection from DPM (Felzenszwalb et al, 2010b). Why did the detector fail?

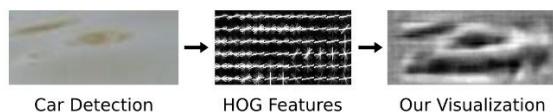
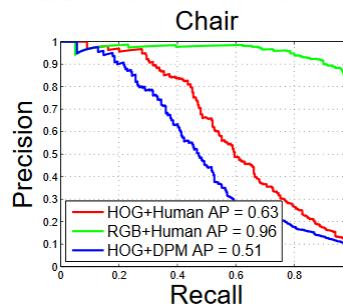
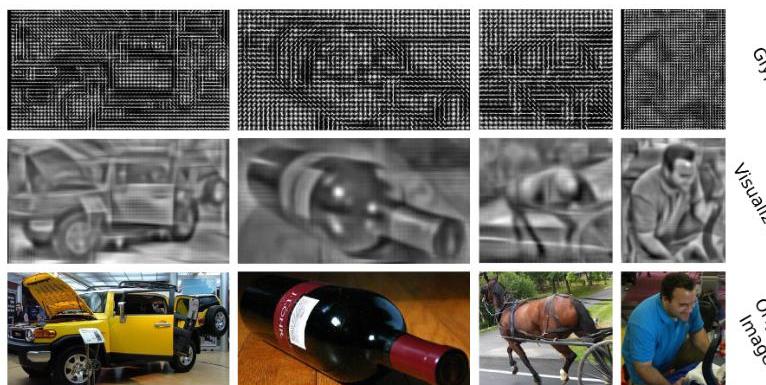


Fig. 2: We show the crop for the false car detection from Figure 1. On the right, we show our visualization of the HOG features for the same patch. Our visualization reveals that this false alarm actually looks like a car in HOG space.



# What about learning the features?



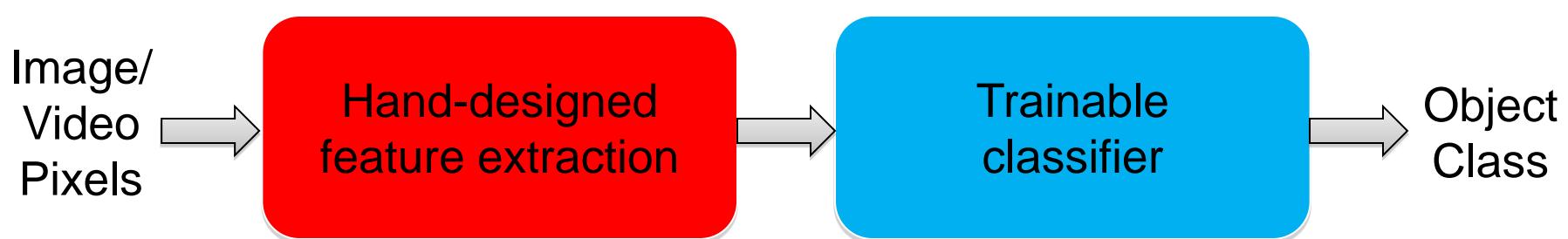
- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



# “Shallow” vs. “deep” architectures

---

Traditional recognition: “Shallow” architecture

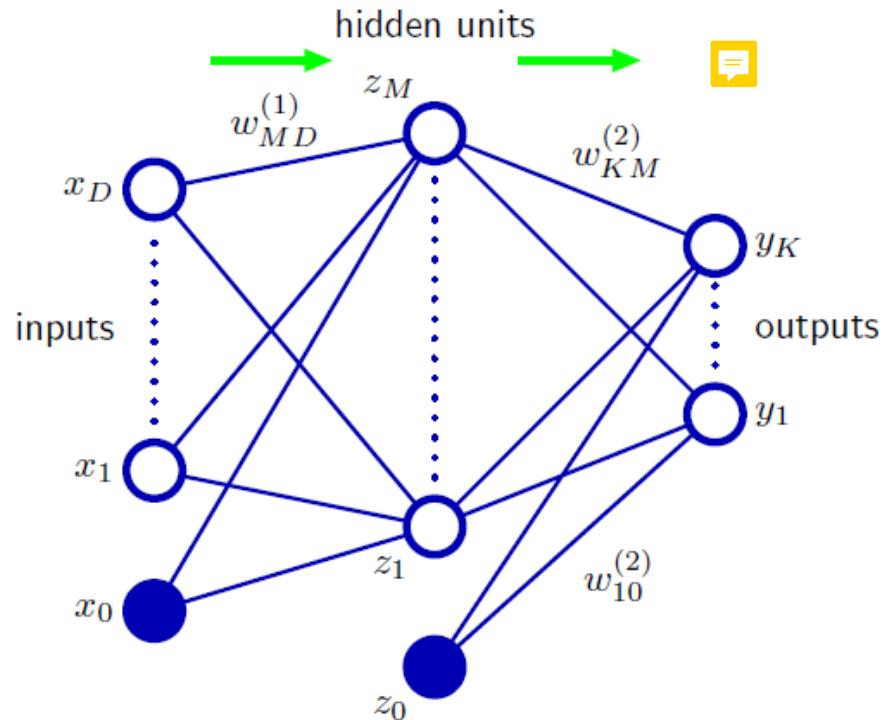


Deep learning: “Deep” architecture



# Neural network definition

**Figure 5.1** Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables  $x_0$  and  $z_0$ . Arrows denote the direction of information flow through the network during forward propagation.

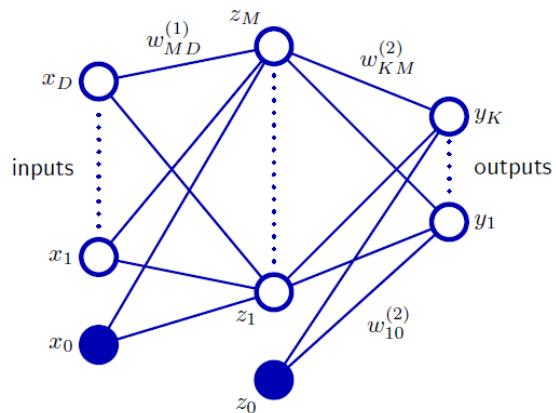


- Activations:  $a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}$  Recall SVM:  
 $w^T x + b$
- Nonlinear activation function  $h$  (e.g. sigmoid, RELU):  $z_j = h(a_j)$

# Neural network definition

- Layer 2

$$a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$



- Layer 3 (final)



$$a_k =$$

- Outputs (e.g. sigmoid/softmax)

(binary)  $y_k = \sigma(a_k) = \frac{1}{1 + \exp(-a_k)}$

(multiclass)  $y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$

- Finally:

(binary)  $y_k(\mathbf{x}, \mathbf{w}) = \sigma \left( \sum_{j=1}^M w_{kj}^{(2)} h \left( \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$

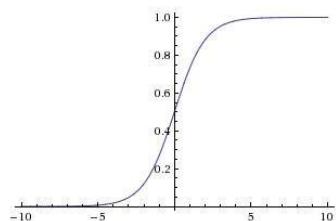
# Activation functions

---

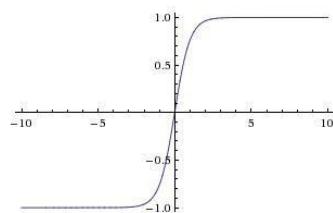
**Sigmoid**



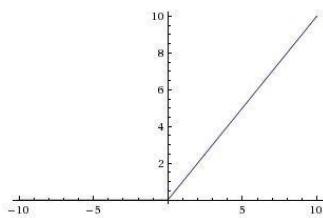
$$\sigma(x) = 1/(1 + e^{-x})$$



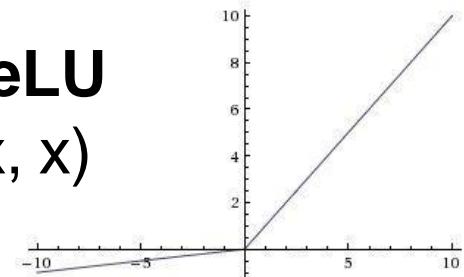
**tanh**     $\tanh(x)$



**ReLU**     $\max(0, x)$



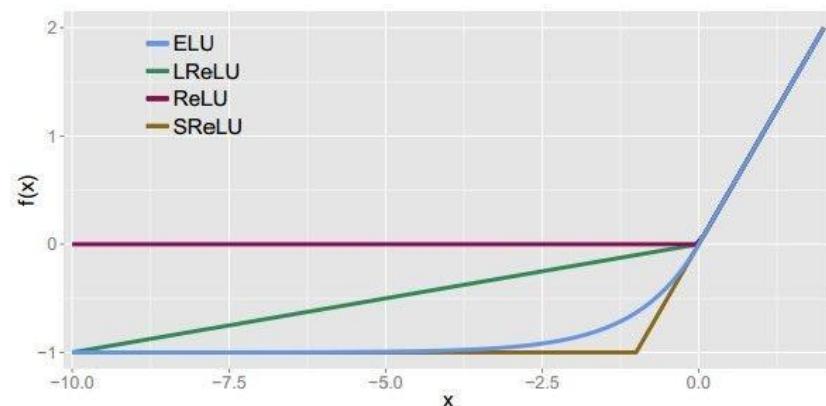
**Leaky ReLU**  
 $\max(0.1x, x)$



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

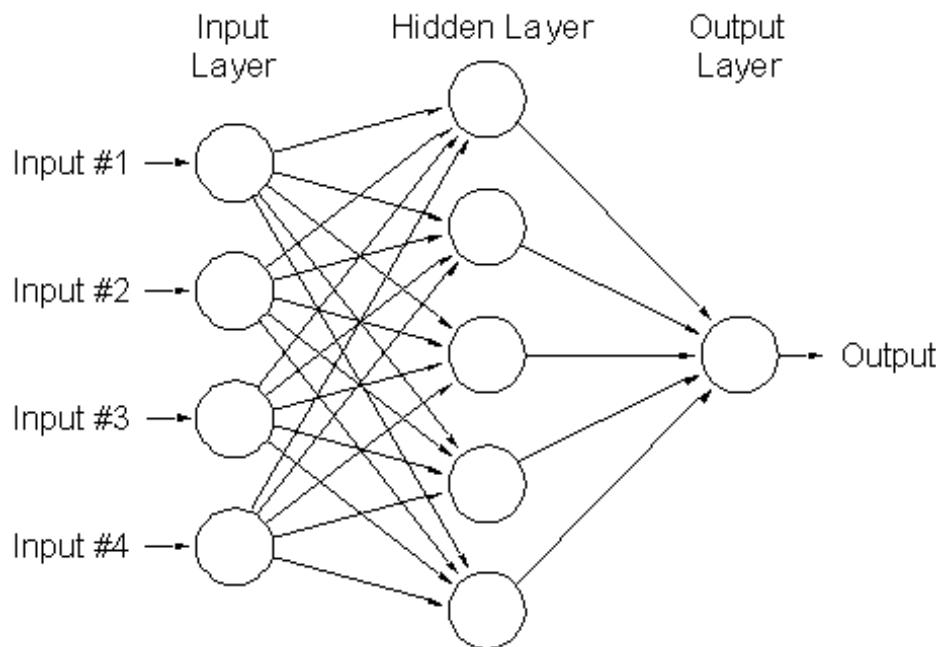
**ELU**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



# A multi-layer neural network

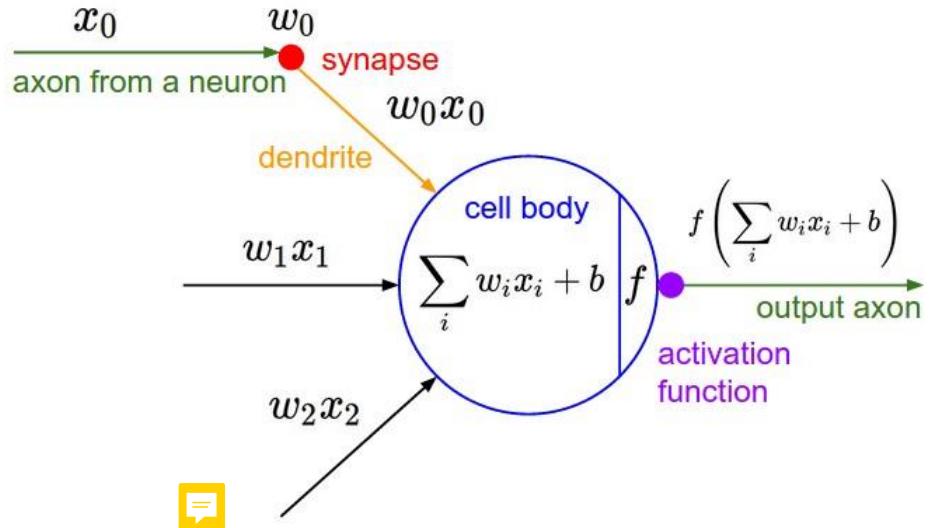
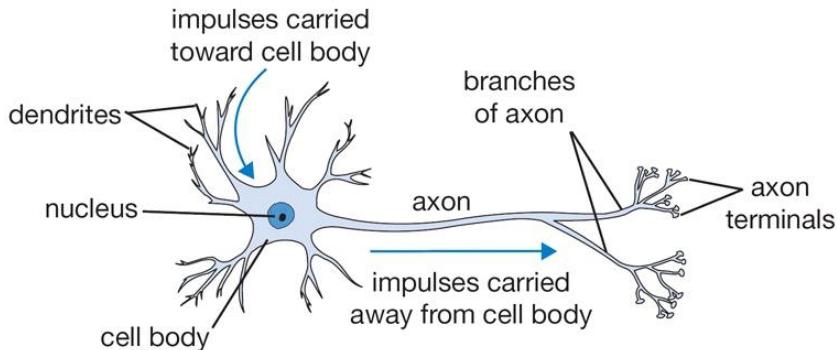
---



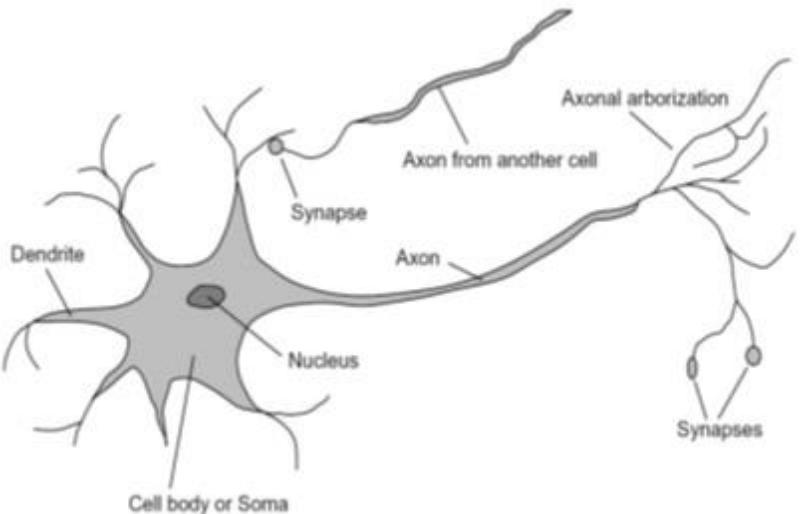
- *Nonlinear classifier* 
- Can approximate any continuous function to arbitrary accuracy given sufficiently many hidden units 

# Inspiration: Neuron cells

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
- If output of function over threshold, neuron “fires”



# Biological analog



A biological neuron

Input

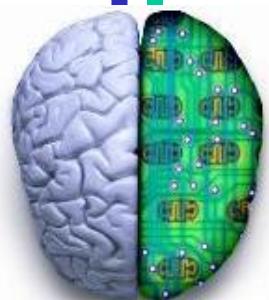
Weights

$$\begin{aligned}x_1 &\rightarrow w_1 \\x_2 &\rightarrow w_2 \\x_3 &\rightarrow w_3 \\&\vdots \\x_d &\rightarrow w_d\end{aligned}$$

Output:  $\sigma(w \cdot x + b)$

Sigmoid function:

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

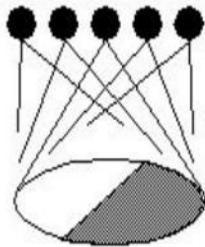


An artificial neuron

# Biological analog

Hubel & Weisel

topographical mapping

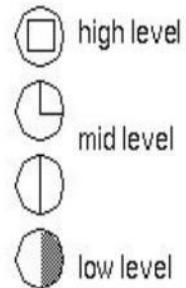
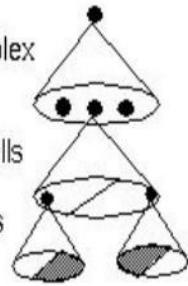


featural hierarchy

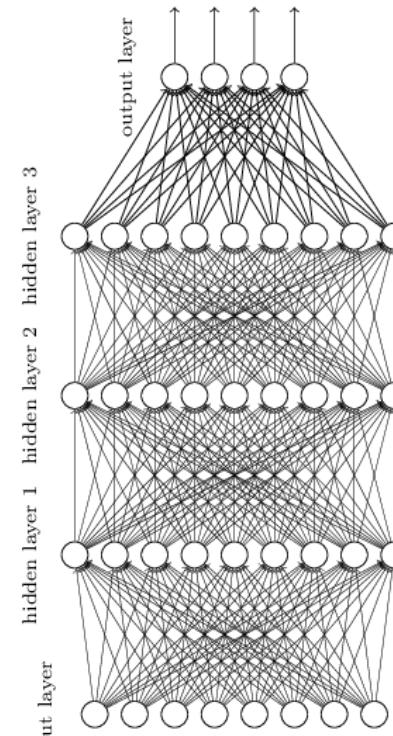
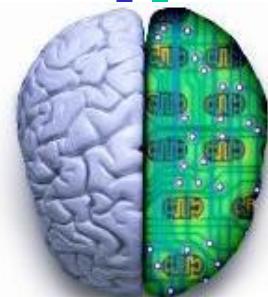
hyper-complex  
cells

complex cells

simple cells



Hubel and Weisel's architecture

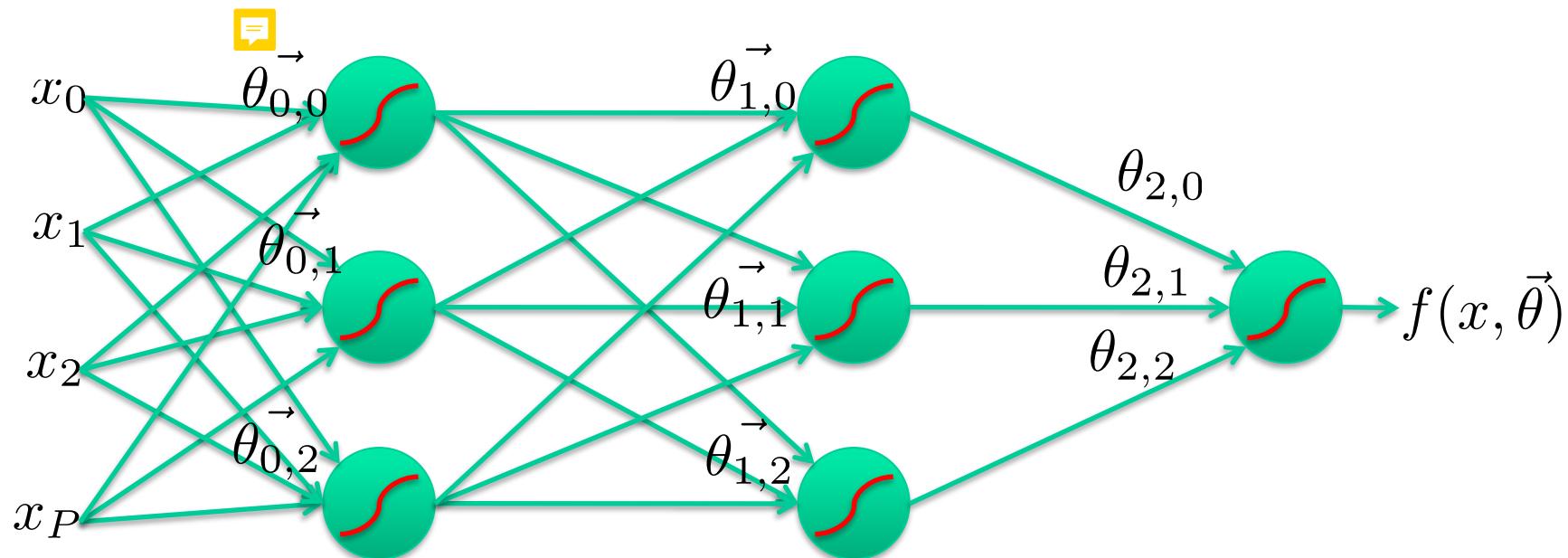


Multi-layer neural network

# Multilayer networks

---

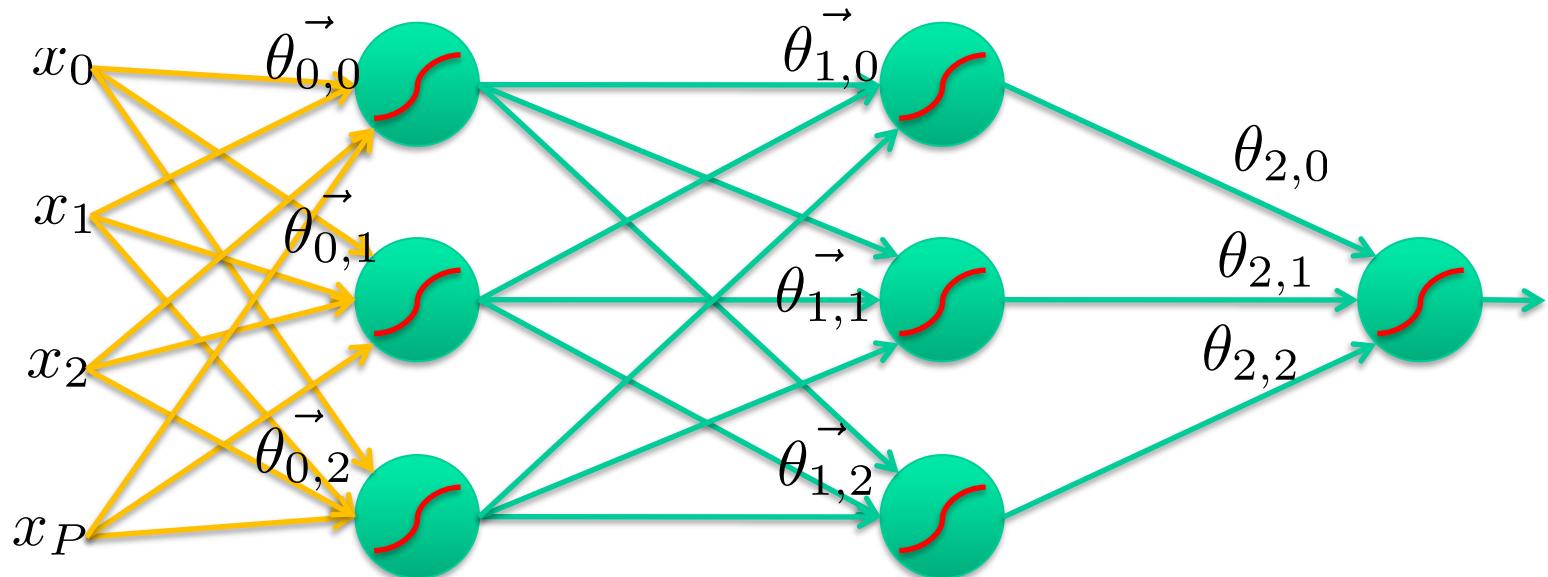
- Cascade neurons together
- Output from one layer is the input to the next
- Each layer has its own sets of weights



# Feed-forward networks

---

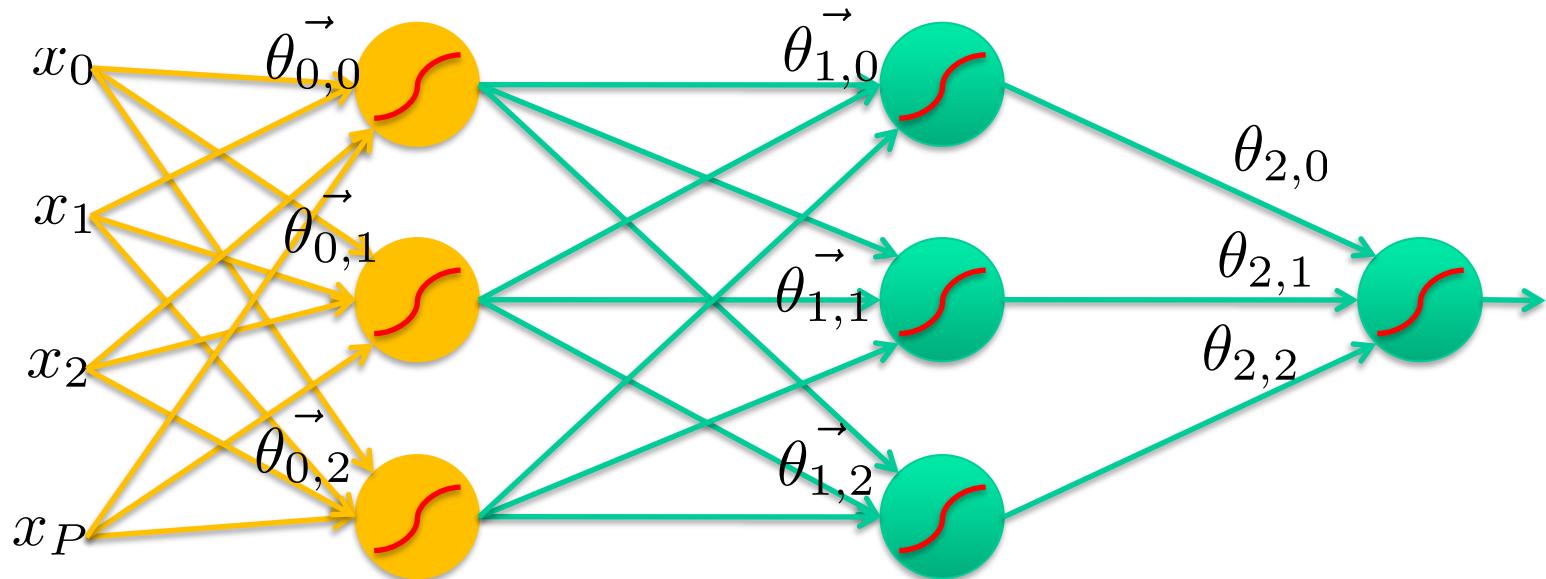
- Predictions are fed forward through the network to classify



# Feed-forward networks

---

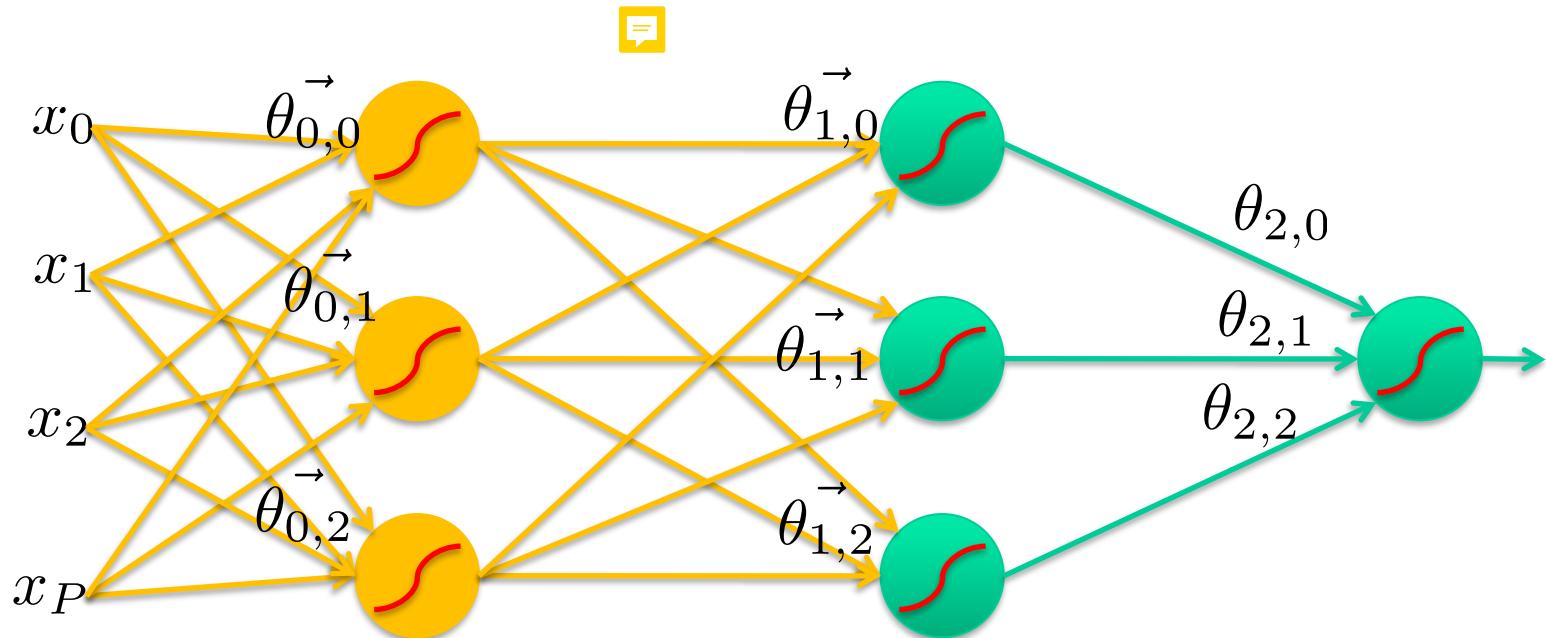
- Predictions are fed forward through the network to classify



# Feed-forward networks

---

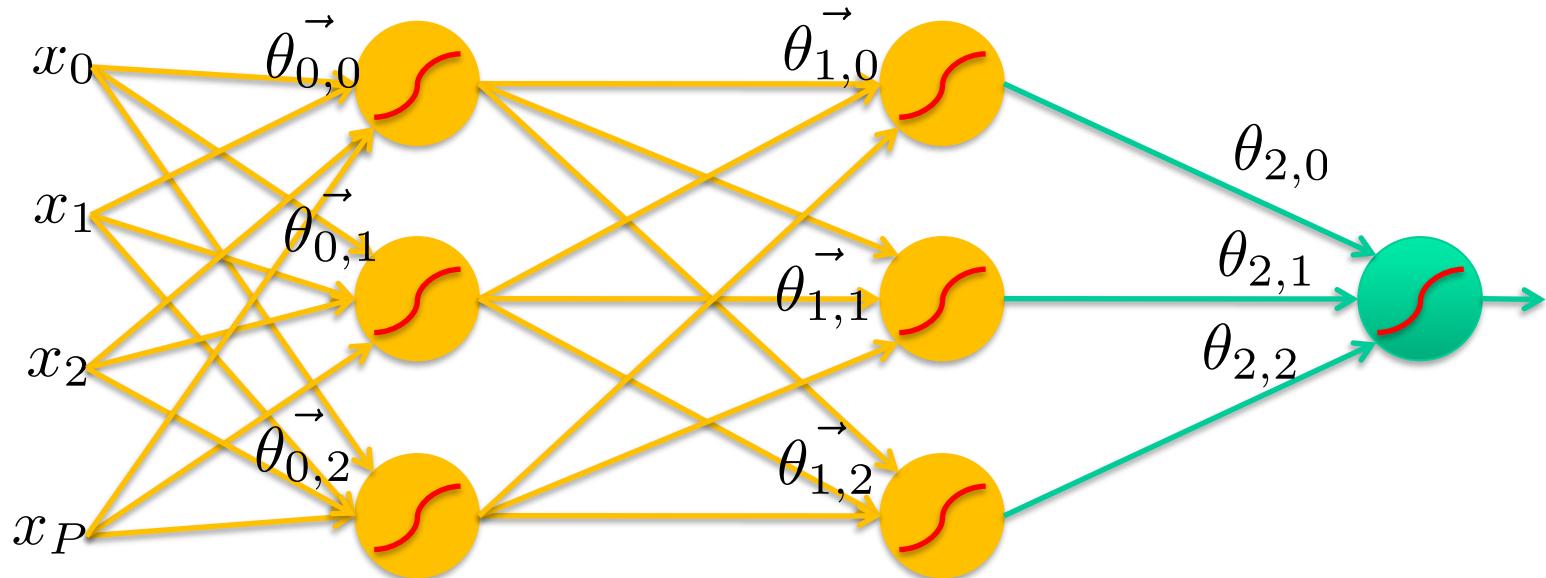
- Predictions are fed forward through the network to classify



# Feed-forward networks

---

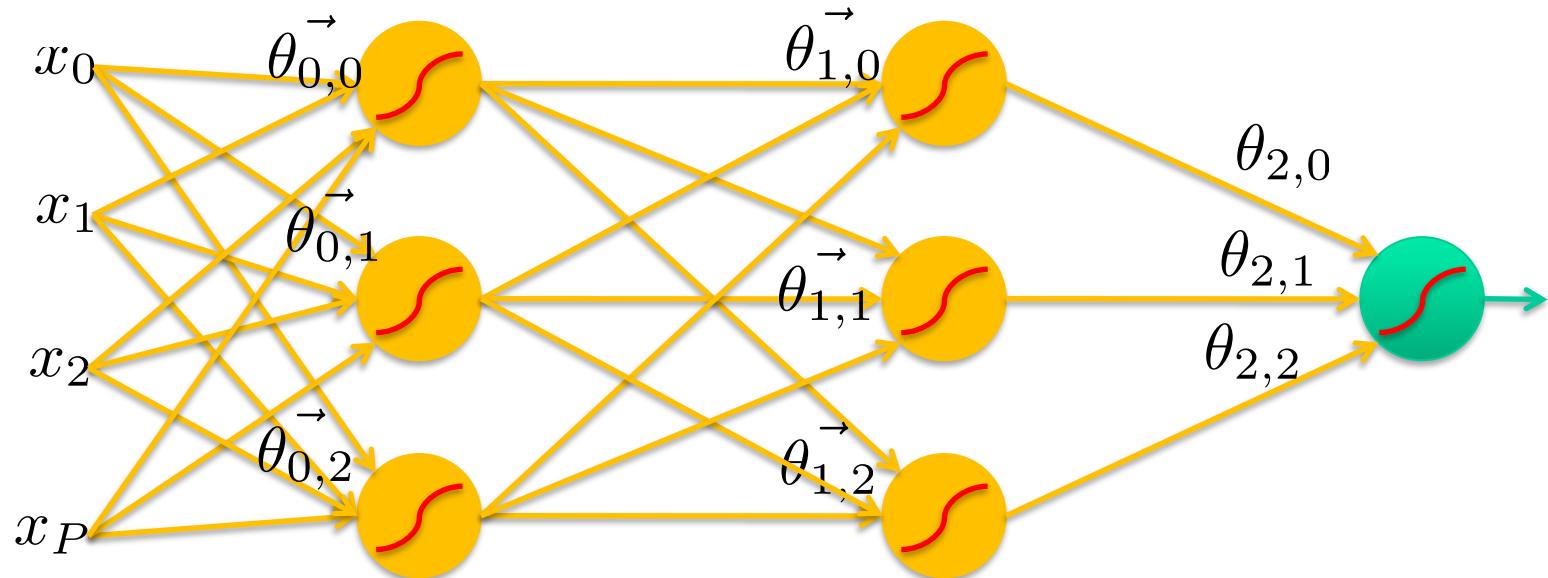
- Predictions are fed forward through the network to classify



# Feed-forward networks

---

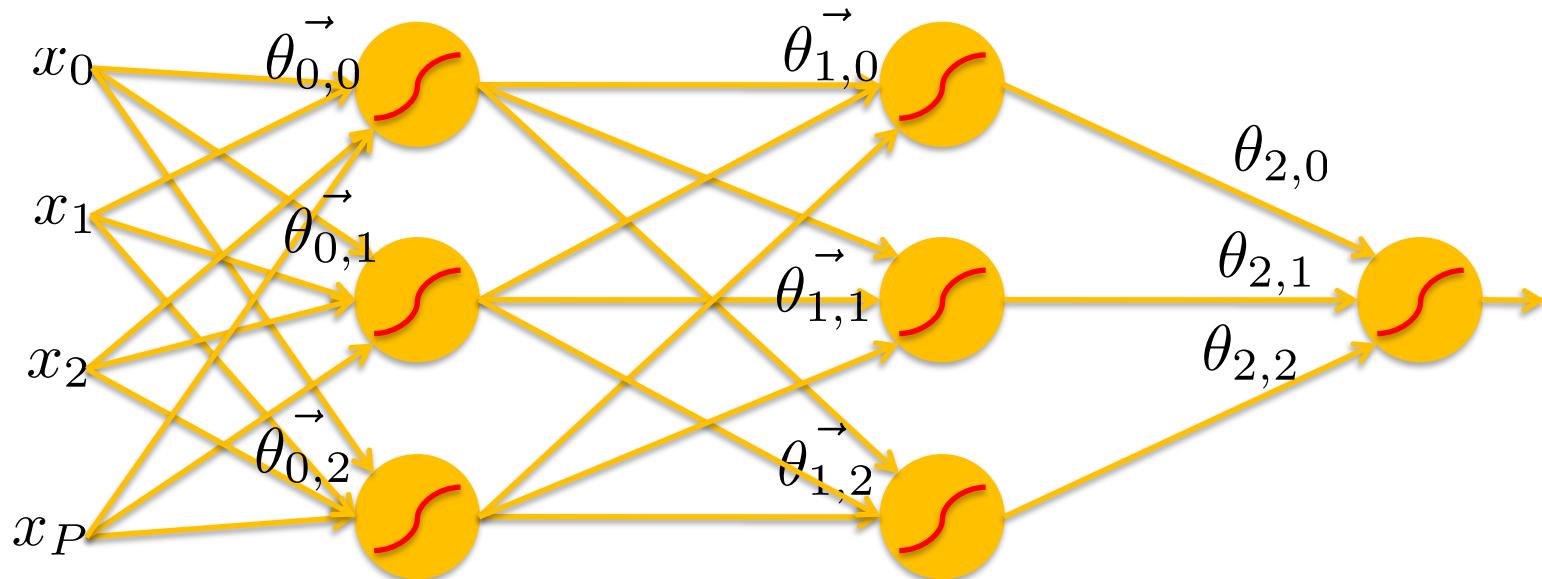
- Predictions are fed forward through the network to classify



# Feed-forward networks

---

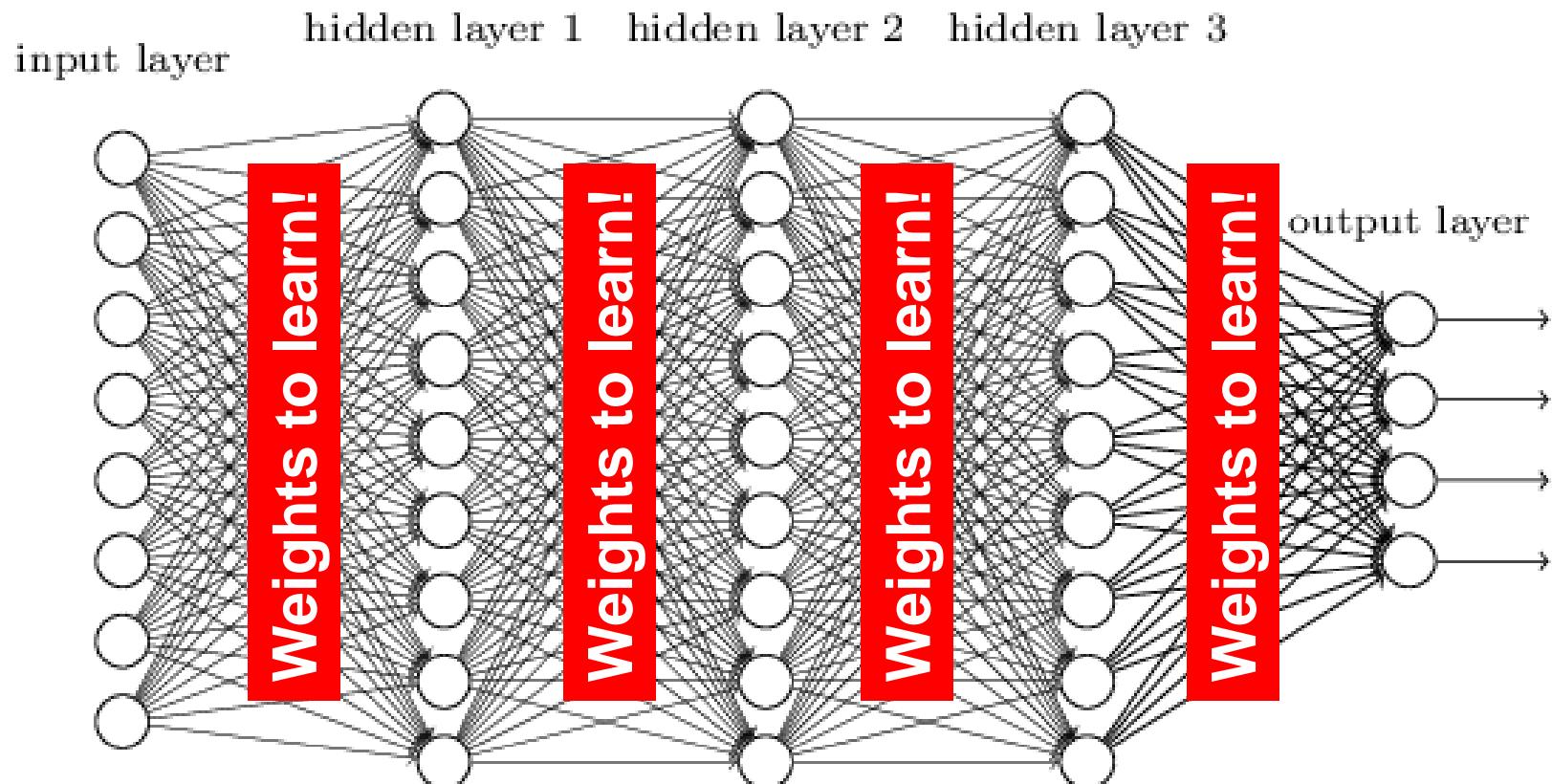
- Predictions are fed forward through the network to classify



# Deep neural networks

---

- Lots of hidden layers
- Depth = power (usually)
- “With great power comes great responsibility”



# How do we train deep neural networks?

---

- The goal is to find such a set of weights that allow the activations/outputs to match the desired output:  $f(\mathbf{W}, \mathbf{x}_i) \sim y_i$
- Unfortunately, no closed-form solution for weights  $\mathbf{W}$ , but we can express our objective
- We want to *minimize a loss function* (a function of the weights in the network), we'll do so iteratively
- For now let's simplify and assume there's a single layer of weights in the network

# Classification goal

---

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



Example dataset: **CIFAR-10**

**10 labels**

**50,000 training images**

each image is **32x32x3**

**10,000 test images**

# Classification scores

---

$$f(x, W) = Wx$$



$$f(\mathbf{x}, \mathbf{W})$$



**10** numbers,  
indicating class  
scores

**[32x32x3]**

array of numbers 0...1  
(3072 numbers total)

# Linear classifier

---



**[32x32x3]**  
array of numbers 0...1

$$f(x, W)$$

**10x1**

$$Wx$$

**10x3072**

**3072x1**

$$(+b)$$

**10x1**



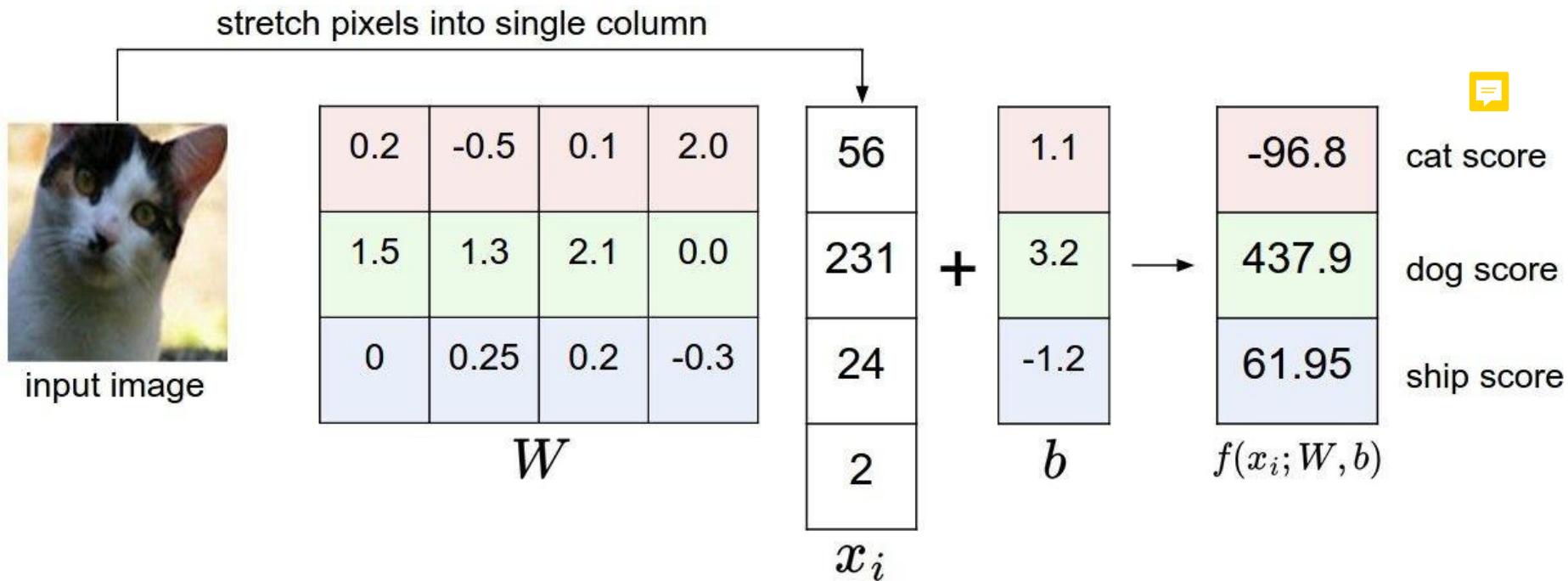
**10** numbers,  
indicating class  
scores

parameters, or “weights”

# Linear classifier

---

Example with an image with 4 pixels, and 3 classes (**cat/dog/ship**)



# Linear classifier

---

Going forward: Loss function/Optimization



cat	<b>3.2</b>	1.3	2.2
car	<b>5.1</b>	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>



TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.
2. Come up with a way of efficiently finding the parameters that minimize the loss function (**optimization**)

# Linear classifier

---

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

# Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## Hinge loss:

Given an example  $(x_i, y_i)$  where  $x_i$  is the image and where  $y_i$  is the (integer) label,

and using the shorthand for the scores vector:  $s = f(x_i, W)$



the loss has the form:

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



Want:  $s_{y_i} \geq s_j + 1$ , for  $j \neq y_i$   
i.e.  $s_j - s_{y_i} + 1 \leq 0$



If true, loss is 0

If false, loss is magnitude of violation

# Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9		

## Hinge loss:

Given an example  $(x_i, y_i)$  where  $x_i$  is the image and where  $y_i$  is the (integer) label,

and using the shorthand for the scores vector:  $s = f(x_i, W)$

the loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \quad \text{💡} \\ &= 2.9 \end{aligned}$$

# Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	

## Hinge loss:

Given an example  $(x_i, y_i)$  where  $x_i$  is the image and where  $y_i$  is the (integer) label,

and using the shorthand for the scores vector:  $s = f(x_i, W)$

the loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

# Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9

## Hinge loss:

Given an example  $(x_i, y_i)$  where  $x_i$  is the image and where  $y_i$  is the (integer) label,

and using the shorthand for the scores vector:  $s = f(x_i, W)$

the loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 2.2 - (-3.1) + 1) \\ &\quad + \max(0, 2.5 - (-3.1) + 1) \\ &= \max(0, 5.3 + 1) \\ &\quad + \max(0, 5.6 + 1) \\ &= 6.3 + 6.6 \\ &= 12.9 \end{aligned}$$

# Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## Hinge loss:

Given an example  $(x_i, y_i)$  where  $x_i$  is the image and where  $y_i$  is the (integer) label,

and using the shorthand for the scores vector:  $s = f(x_i, W)$

the loss has the form:

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

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$\begin{aligned} L &= (2.9 + 0 + 12.9)/3 \\ &= 15.8 / 3 = 5.3 \end{aligned}$$

# Linear classifier: Hinge loss

---



$$f(x, W) = Wx$$

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



# Linear classifier: Hinge loss



## Weight Regularization



$\lambda$  = regularization strength  
(hyperparameter)

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

In common use:

**L2 regularization**

L1 regularization

Dropout (will see later)



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

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

# Another loss: Softmax (cross-entropy)



scores = unnormalized log probabilities of the classes

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$



cat            3.2

where

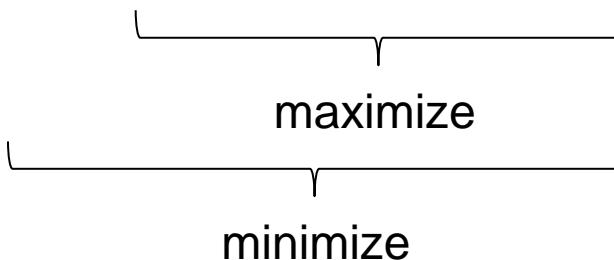
$$s = f(x_i; W)$$

car            5.1

frog          -1.7

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i | X = x_i)$$



# Another loss: Softmax (cross-entropy)



$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat  
car  
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

$$L_i = -\log(0.13) \\ = 0.89$$



unnormalized log probabilities

probabilities

# Other losses

---

- Triplet loss (Schroff, FaceNet)

$$\sum_i^N \left[ \|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$


a denotes anchor  
p denotes positive  
n denotes negative

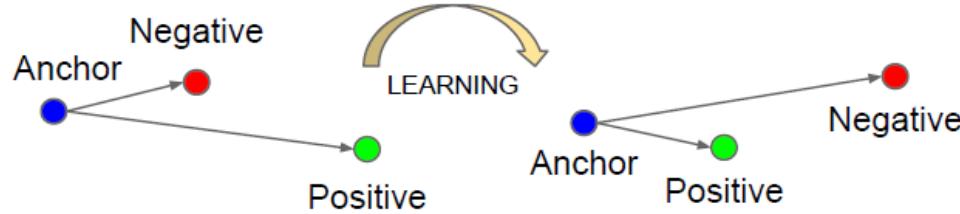
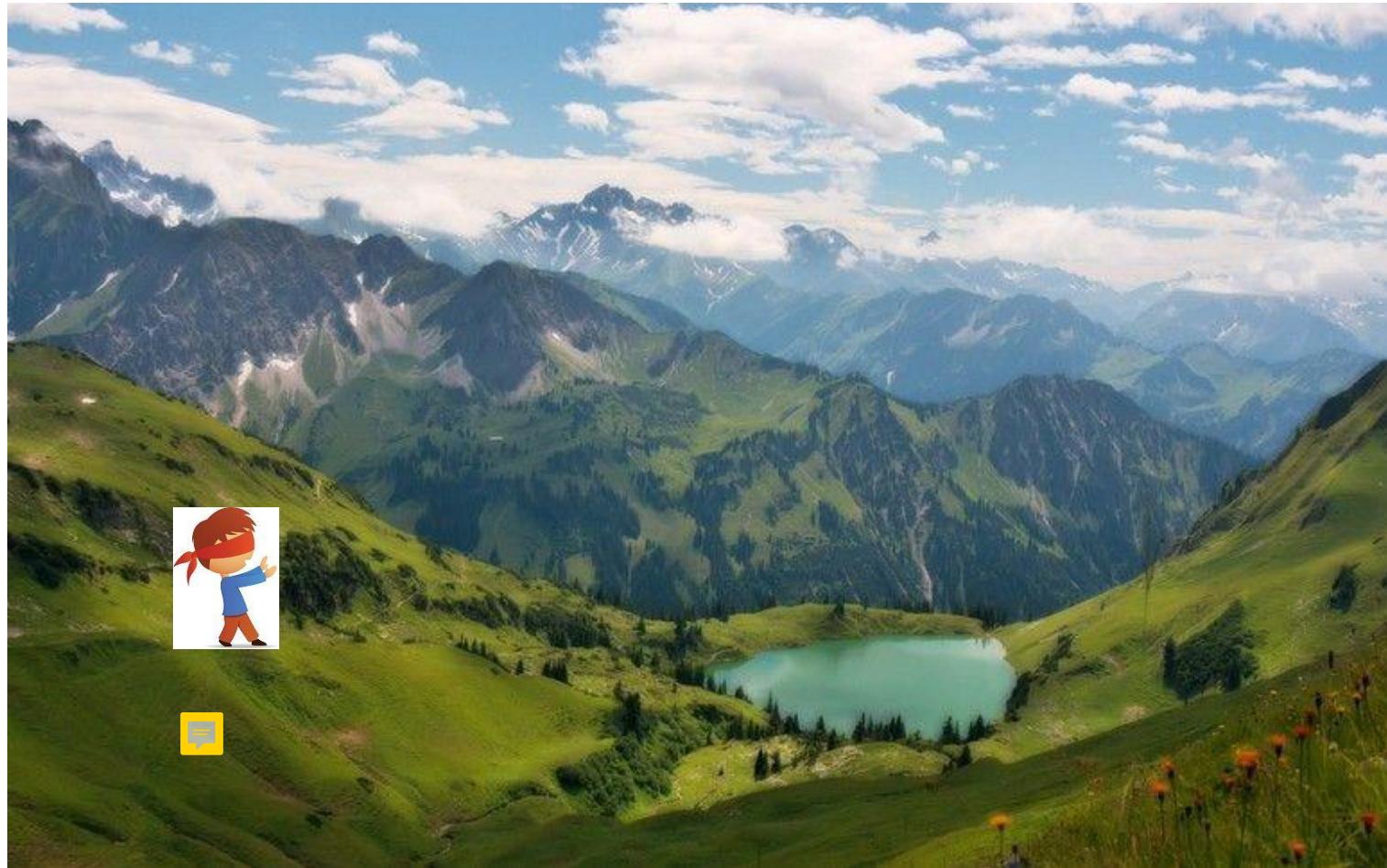


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

- Anything you want!

# How to minimize the loss function?

---



# How to minimize the loss function?

---

In 1-dimension, the derivative of a function is:

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives):

That is, for  $f: \mathbf{R}^n \rightarrow \mathbf{R}$ , its gradient  $\nabla f: \mathbf{R}^n \rightarrow \mathbf{R}^n$  is defined at the point  $p = (x_1, \dots, x_n)$  in  $n$ -dimensional space as the vector:<sup>[b]</sup>

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix} \quad \text{💡}$$

The **nabla symbol**  $\nabla$ , written as an upside-down triangle and pronounced "del", denotes the **vector differential operator**.

## Computing the gradient numerically

**current W:**

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25347**

**gradient dW:**

[?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,...]

## Computing the gradient numerically

**current W:**

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25347**

**W + h (first dim):**

[0.34 + 0.0001,   
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25322**

**gradient dW:**

[?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,...]

## Computing the gradient numerically

current W:

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

loss 1.25347

W + h (first dim):

[0.34 + 0.0001,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

loss 1.25322

gradient dW:

[-2.5,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?]

$$(1.25322 - 1.25347)/0.0001  
= -2.5$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

?,  
?,  
?,...]

## Computing the gradient numerically

current W:

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25347**

W + h (second dim):

[0.34,  
-1.11 + **0.0001**,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25353**

gradient dW:

[-2.5,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,...]

## Computing the gradient numerically

current W:

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

loss 1.25347

W + h (second dim):

[0.34,  
-1.11 + 0.0001,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

loss 1.25353

gradient dW:

[-2.5,  
0.6,  
?,  
?,  
?

$$\frac{(1.25353 - 1.25347)}{0.0001} = 0.6$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

?,...]

## Computing the gradient numerically

current W:

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25347**

W + h (third dim):

[0.34,  
-1.11,  
0.78 + **0.0001**,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25347**

gradient dW:

[-2.5,  
0.6,  
?,  
?,  
?,  
?,  
?,  
?,  
?,  
?,...]

## Computing the gradient analytically

The loss is just a function of  $W$ :

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

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

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

want  $\nabla_W L$

## Computing the gradient analytically

The loss is just a function of W:

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

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

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

want  $\nabla_W L$

Calculus

$$\nabla_W L = \dots$$

Computing the gradient analytically



## Computing the gradient analytically

**current W:**

[0.34,  
-1.11,  
0.78,  
0.12,  
0.55,  
2.81,  
-3.1,  
-1.5,  
0.33,...]

**loss 1.25347**

**gradient dW:**

[-2.5,  
0.6,  
0,  
0.2,  
0.7,  
-0.5,  
1.1,  
1.3,  
-2.1,...]

$dW = \dots$   
(some function of  
data and  $W$ )



## Example of gradient calculation

- $f(\mathbf{x}, \mathbf{w}) = \text{dot}(\mathbf{w}, \mathbf{x}) = w_1*x_1 + w_2*x_2 + \dots + w_D*x_D$
- $d f(\mathbf{x}, \mathbf{w}) / d w_1 = ?$
- $d f(\mathbf{x}, \mathbf{w}) / d w_1 = x_1$
- $d f(\mathbf{x}, \mathbf{w}) / d w_2 = x_2$
- ...
- Gradient of  $f(\mathbf{x}, \mathbf{w})$  wrt  $\mathbf{w}$  is  $[x_1 \ x_2 \ \dots \ x_D]$  i.e.  $\underline{\mathbf{x}}$

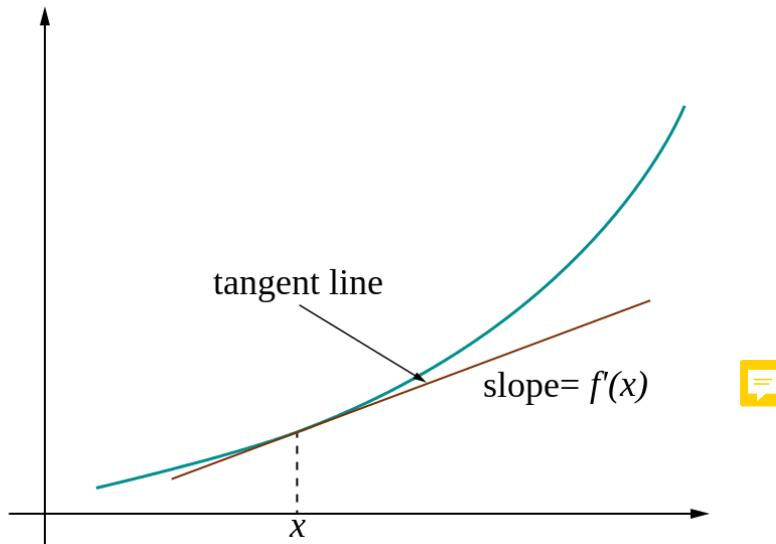
# Loss gradients

---

- Different notations:

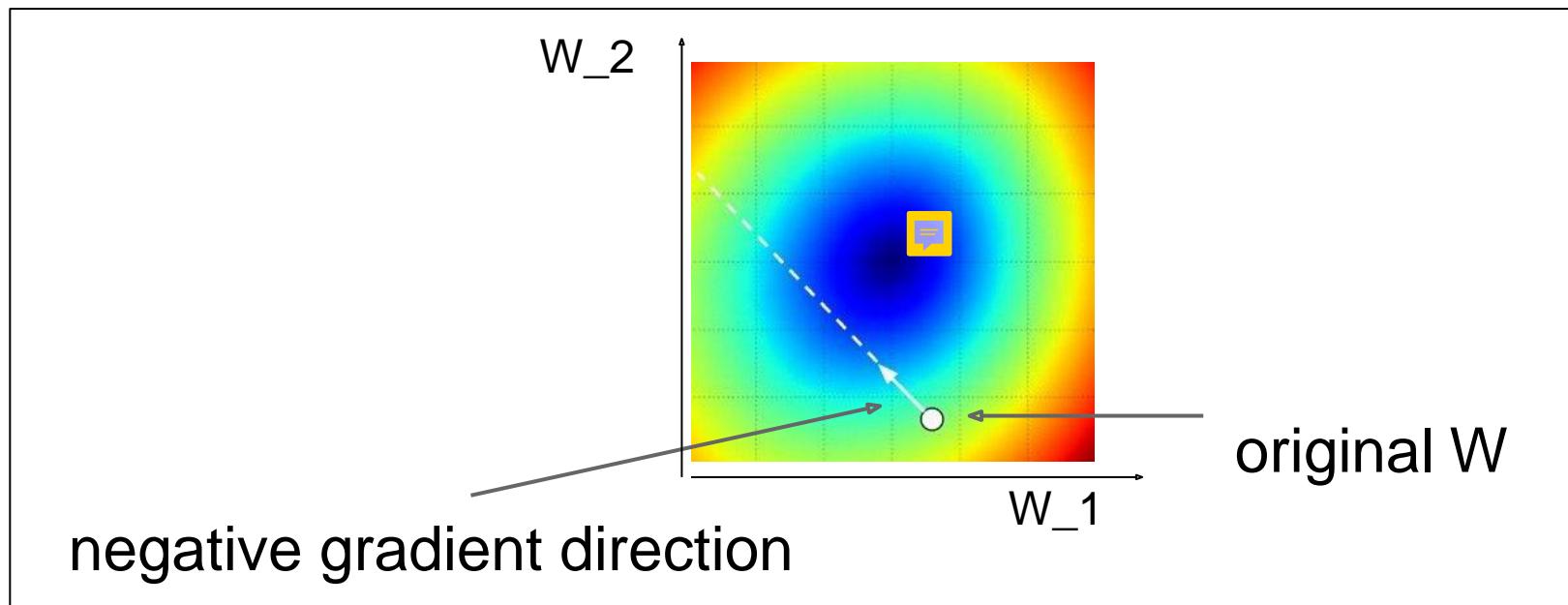
$$\frac{\partial E}{\partial w_{ji}^{(1)}} \quad \nabla_W L$$

- i.e. how does the loss change as a function of the weights
- We want to change weights in a way that makes the loss decrease as fast as possible



# Gradient descent

- We'll update weights
  - Move in direction opposite to gradient:



# Gradient descent

---

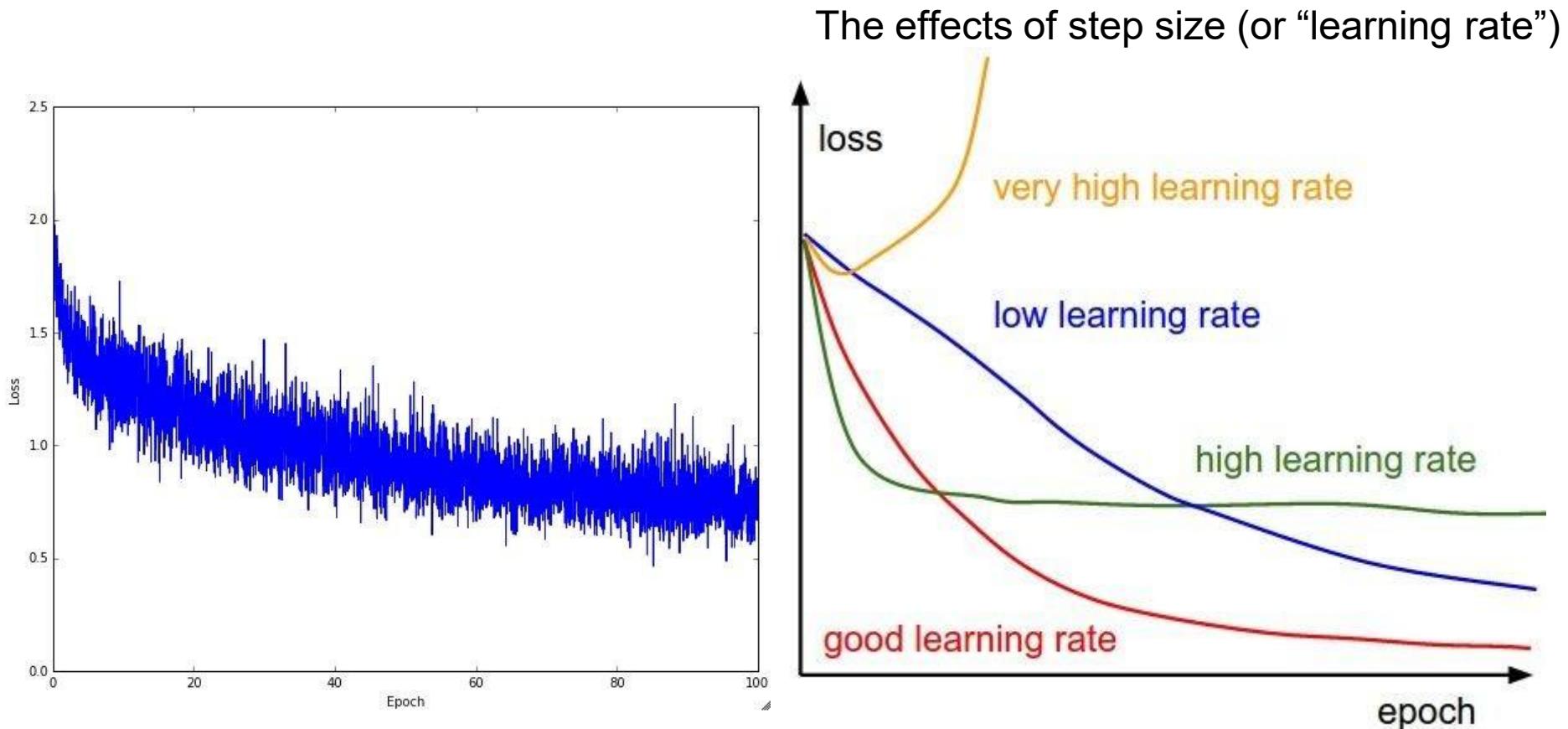
- Iteratively *subtract* the gradient with respect to the model parameters ( $w$ )
- I.e. we're moving in a direction opposite to the gradient of the loss
- I.e. we're moving towards *smaller* loss

# How to compute the loss/gradient?

---

- In classic gradient descent, we compute the gradient from the loss for all training examples
- Mini-batch gradient descent: Only use *some* of the data for each gradient update, cycle through training examples multiple times
  - Each time we've cycled through all of them once is called an ‘epoch’
  - Allows faster training (e.g. on GPUs), parallelization
  - Some benefits for learning due to randomness

# Learning rate selection



# Gradient descent in multi-layer nets

---

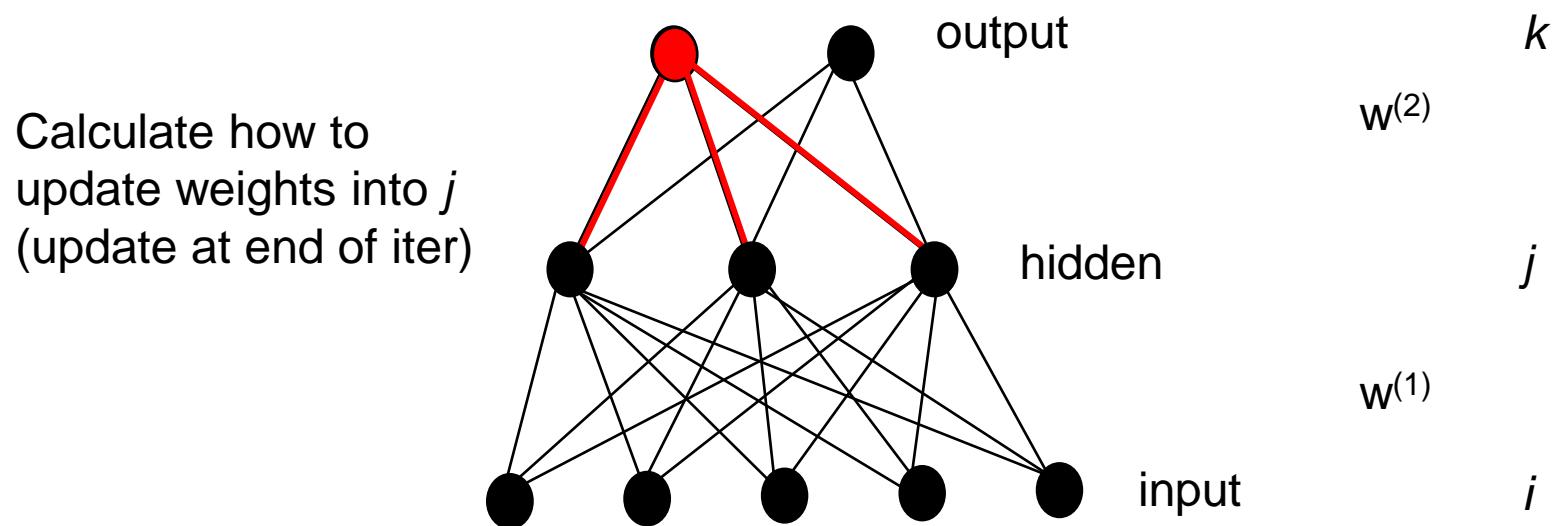
- We'll update weights
- Move in direction opposite to gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

- How to update the weights at all layers?
- Answer: backpropagation of error from higher layers to lower layers

# Backpropagation: Graphic example

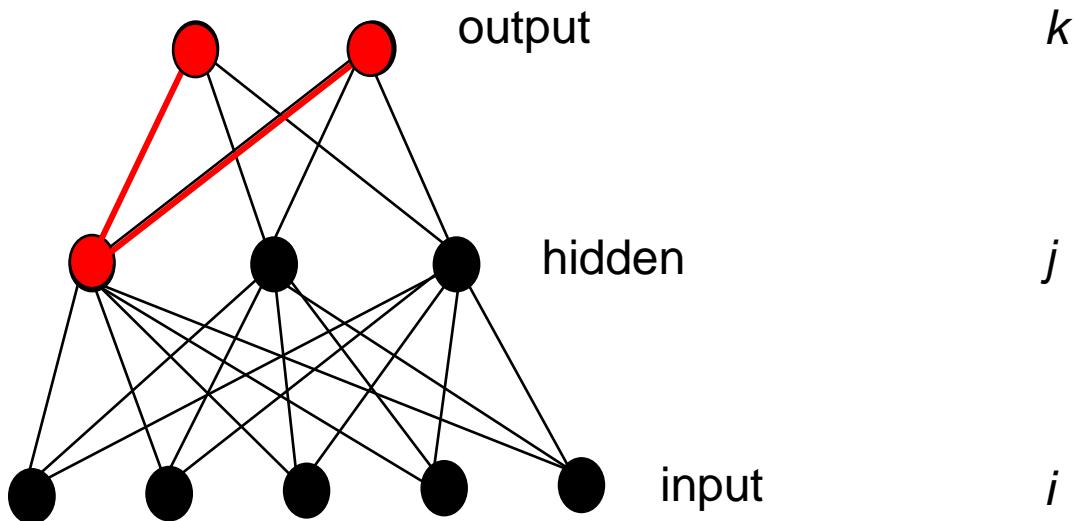
First calculate error of output units and use this to change the top layer of weights.



# Backpropagation: Graphic example

---

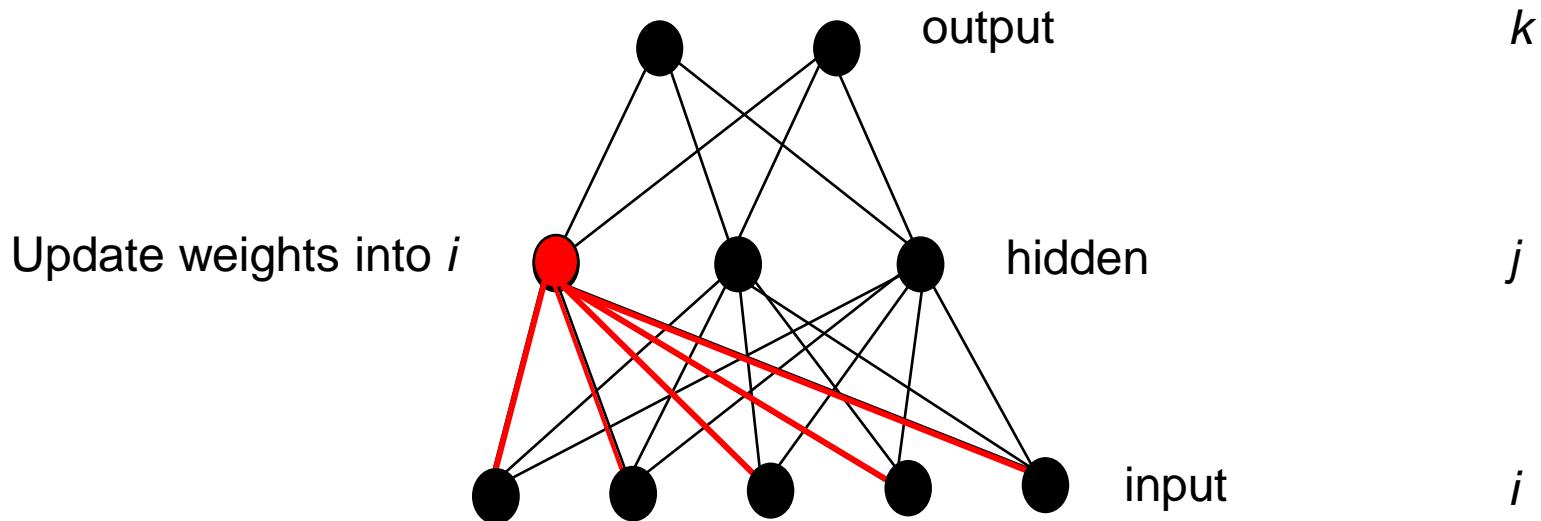
Next calculate error for hidden units based on errors on the output units it feeds into.



# Backpropagation: Graphic example

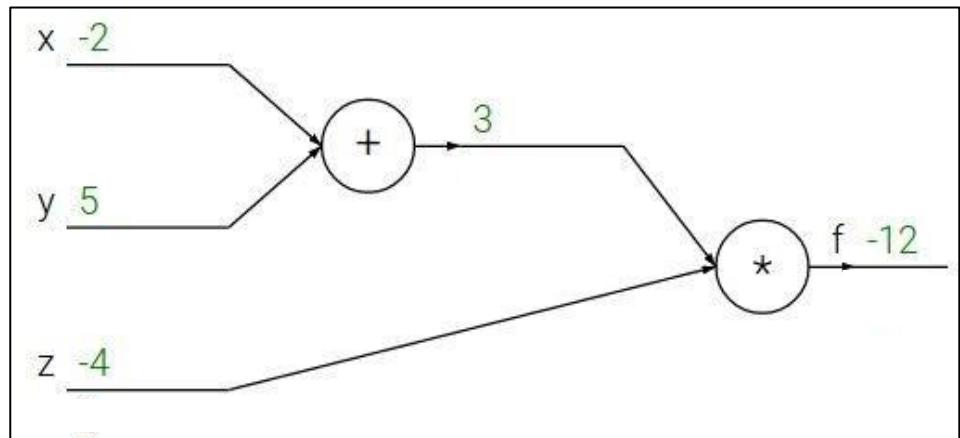
---

Finally update bottom layer of weights based on errors calculated for hidden units.



$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$



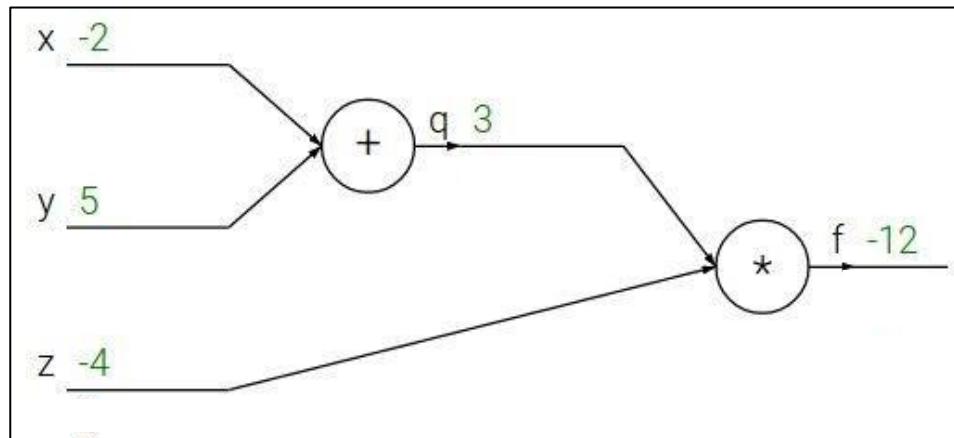
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

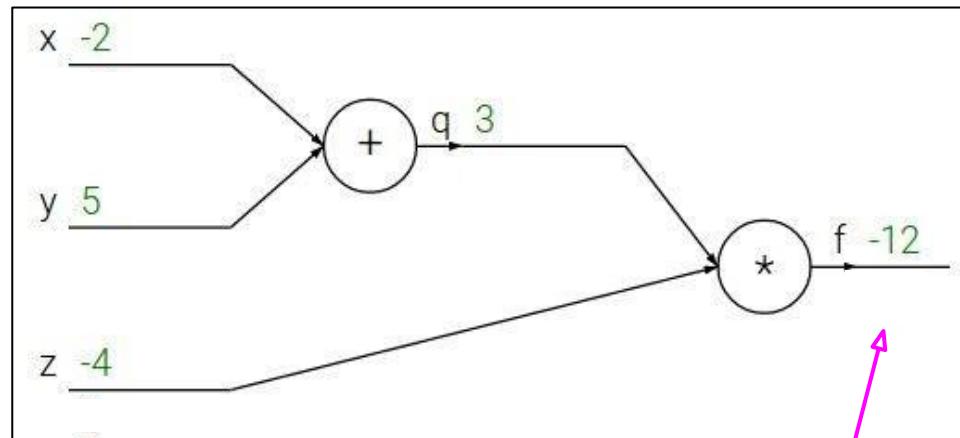


$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

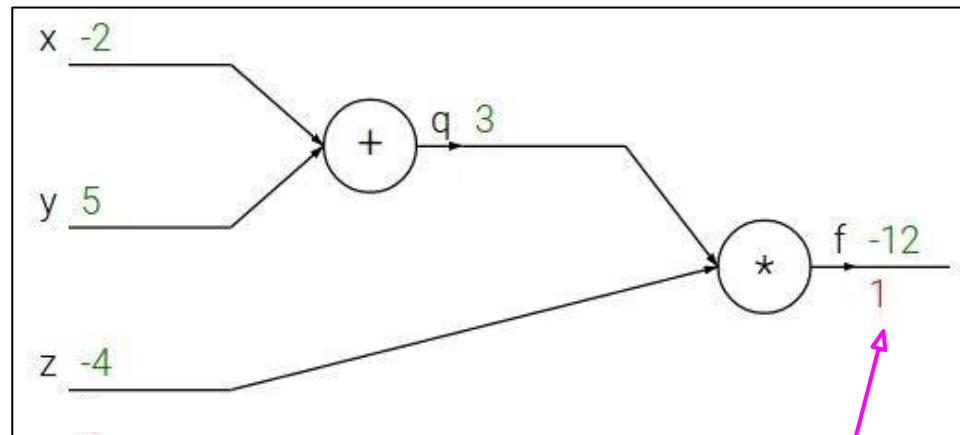
$$\frac{\partial f}{\partial f}$$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

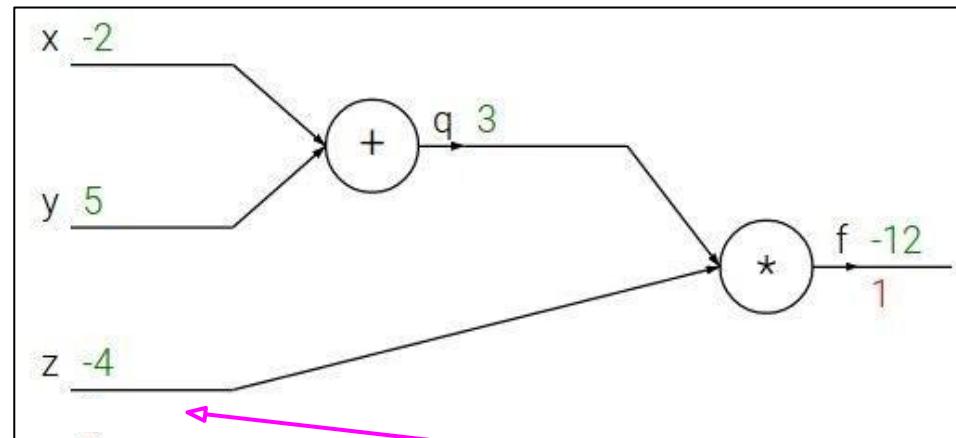
$$\frac{\partial f}{\partial f}$$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

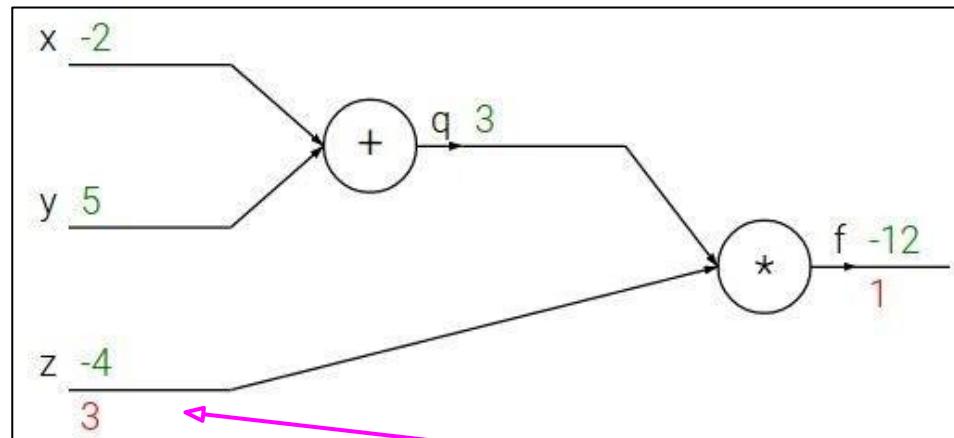
$$\frac{\partial f}{\partial z}$$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

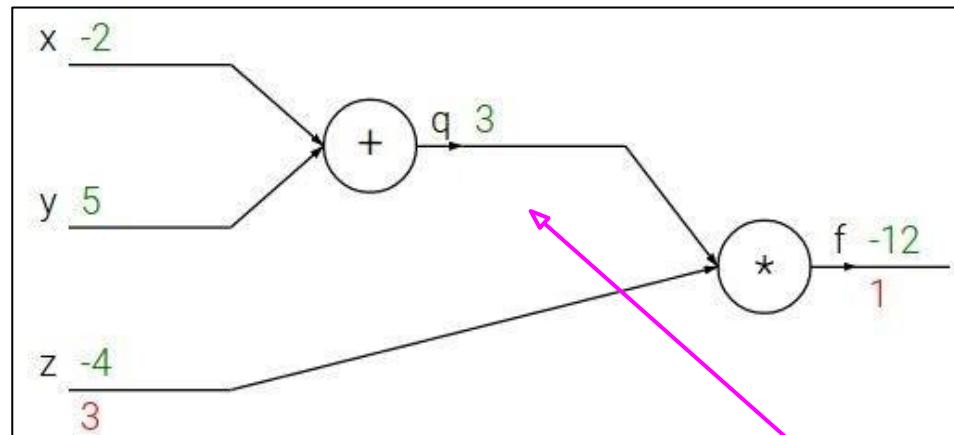
$$\frac{\partial f}{\partial z}$$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

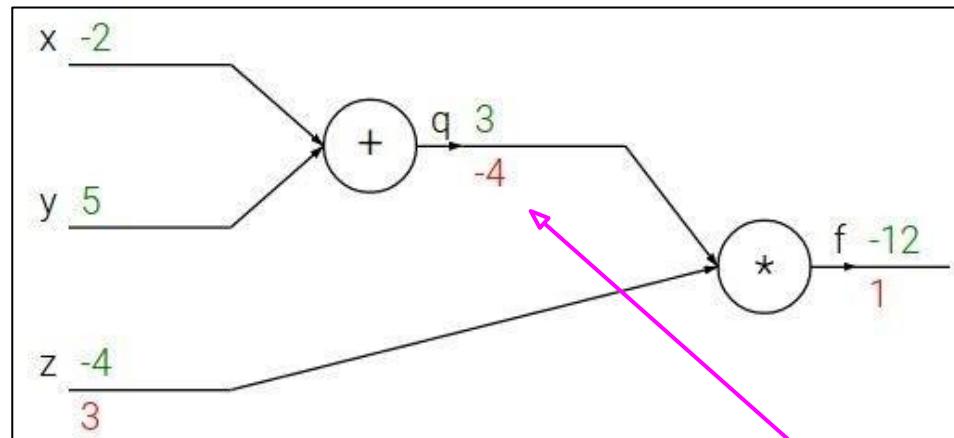
$$\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial q}$$

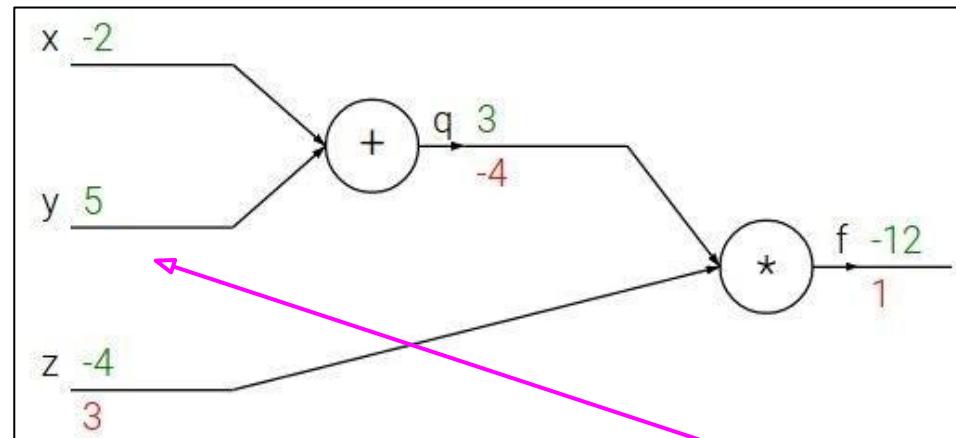
Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial y}$$

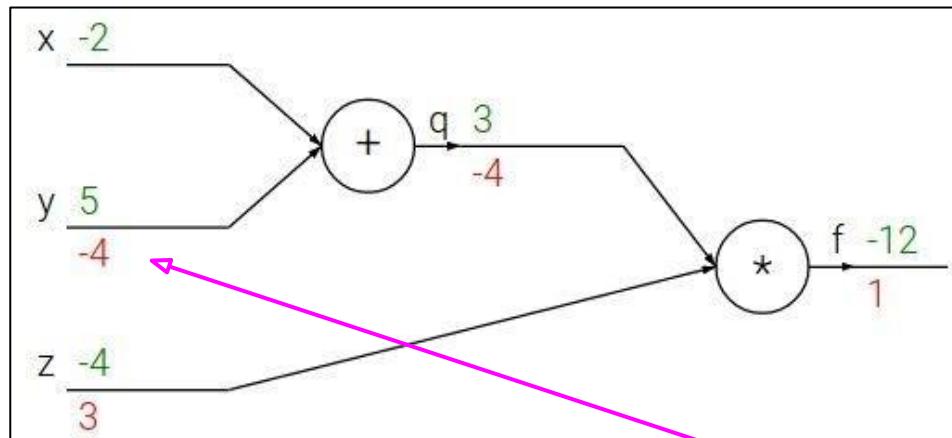
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

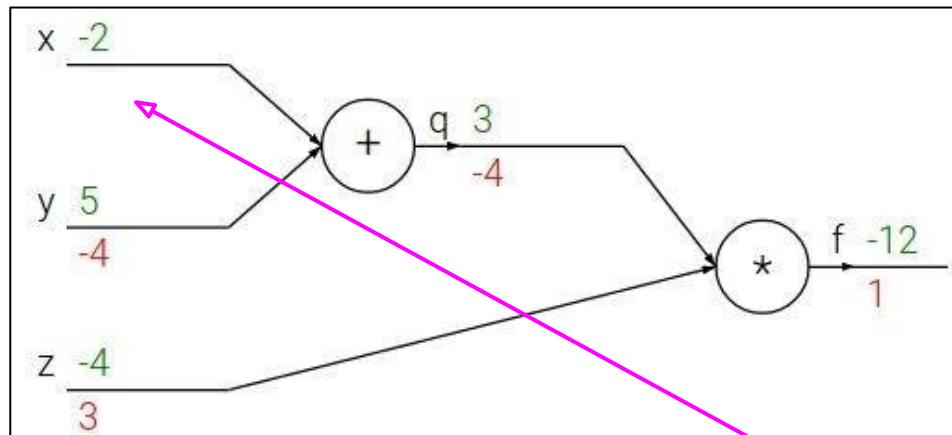
$$\boxed{\frac{\partial f}{\partial y}}$$

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial x}$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

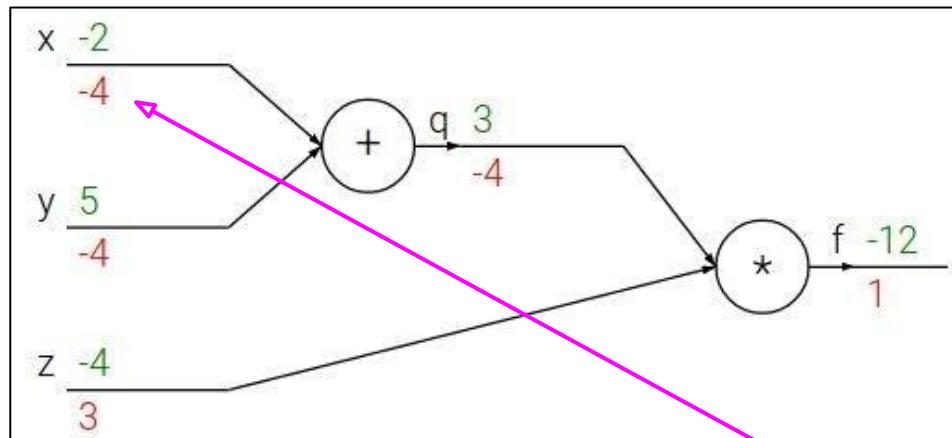
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

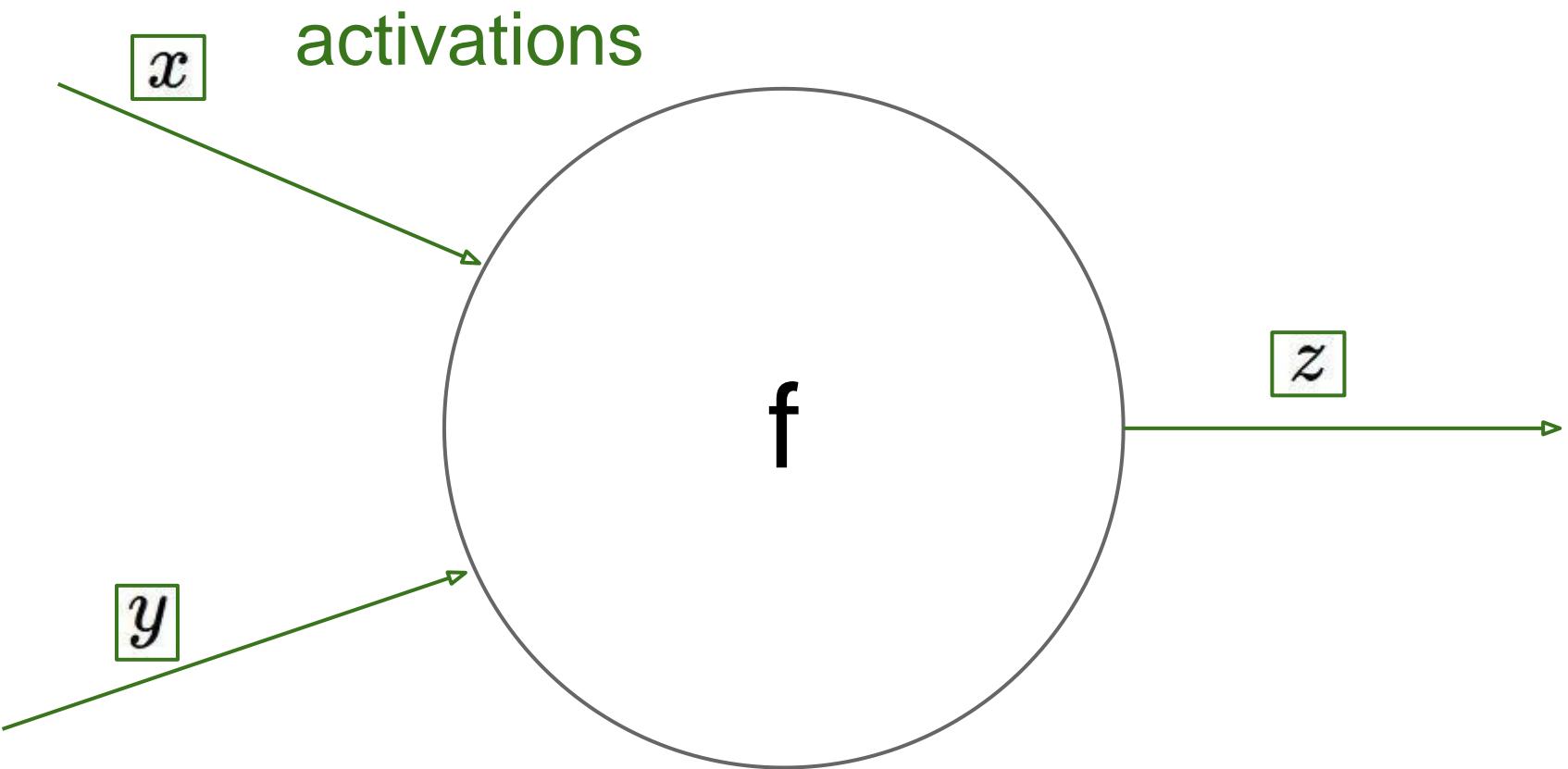
Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

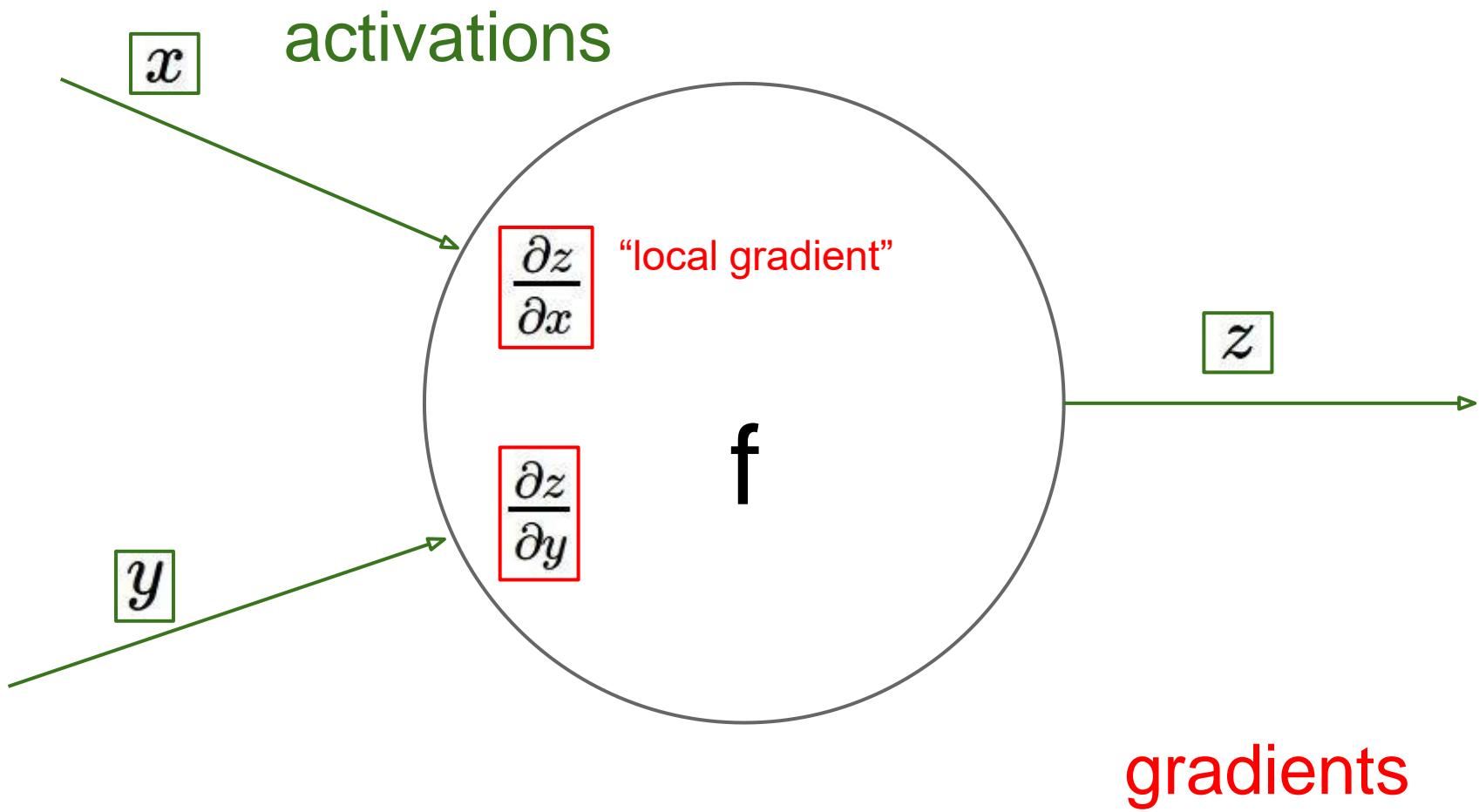


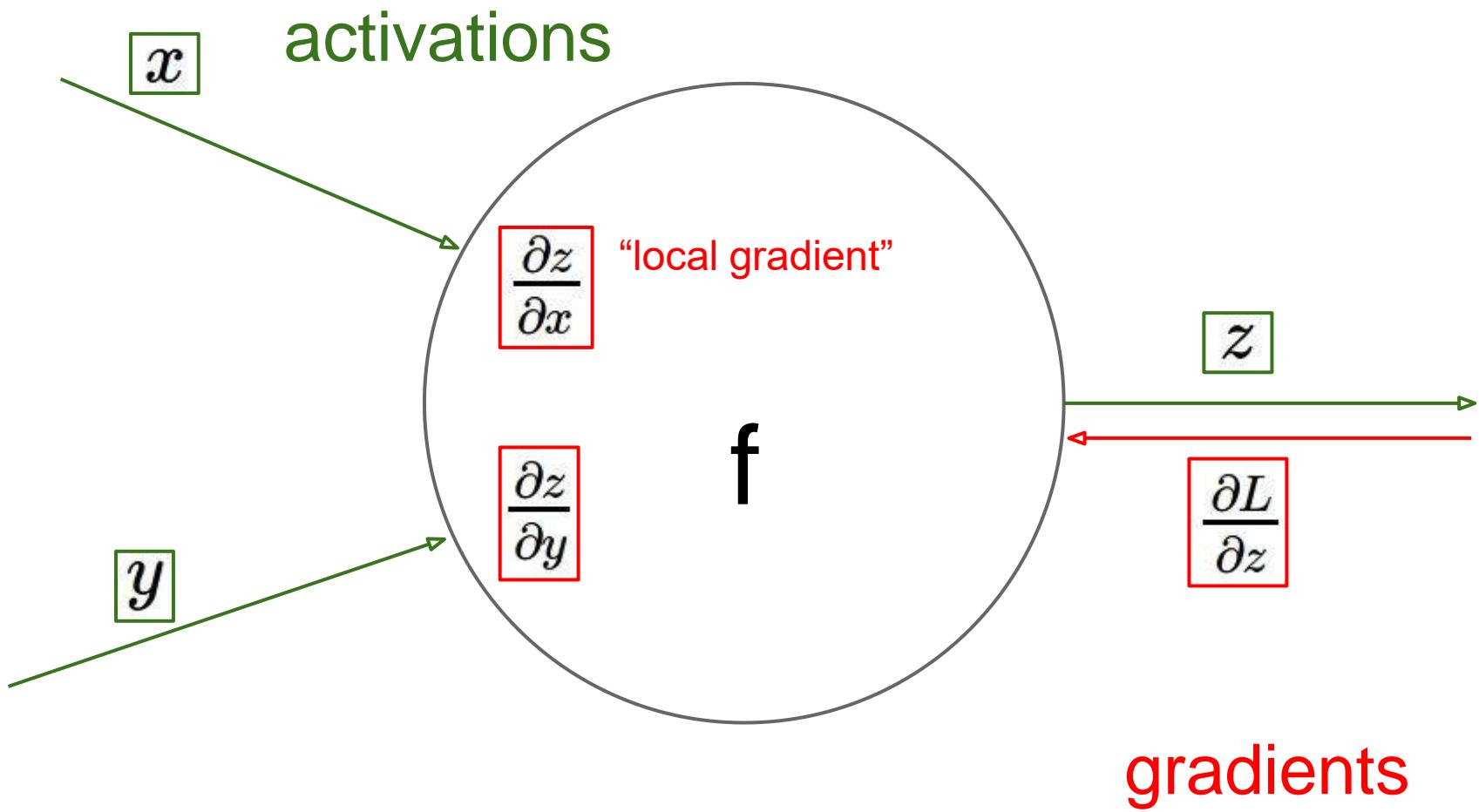
Chain rule:

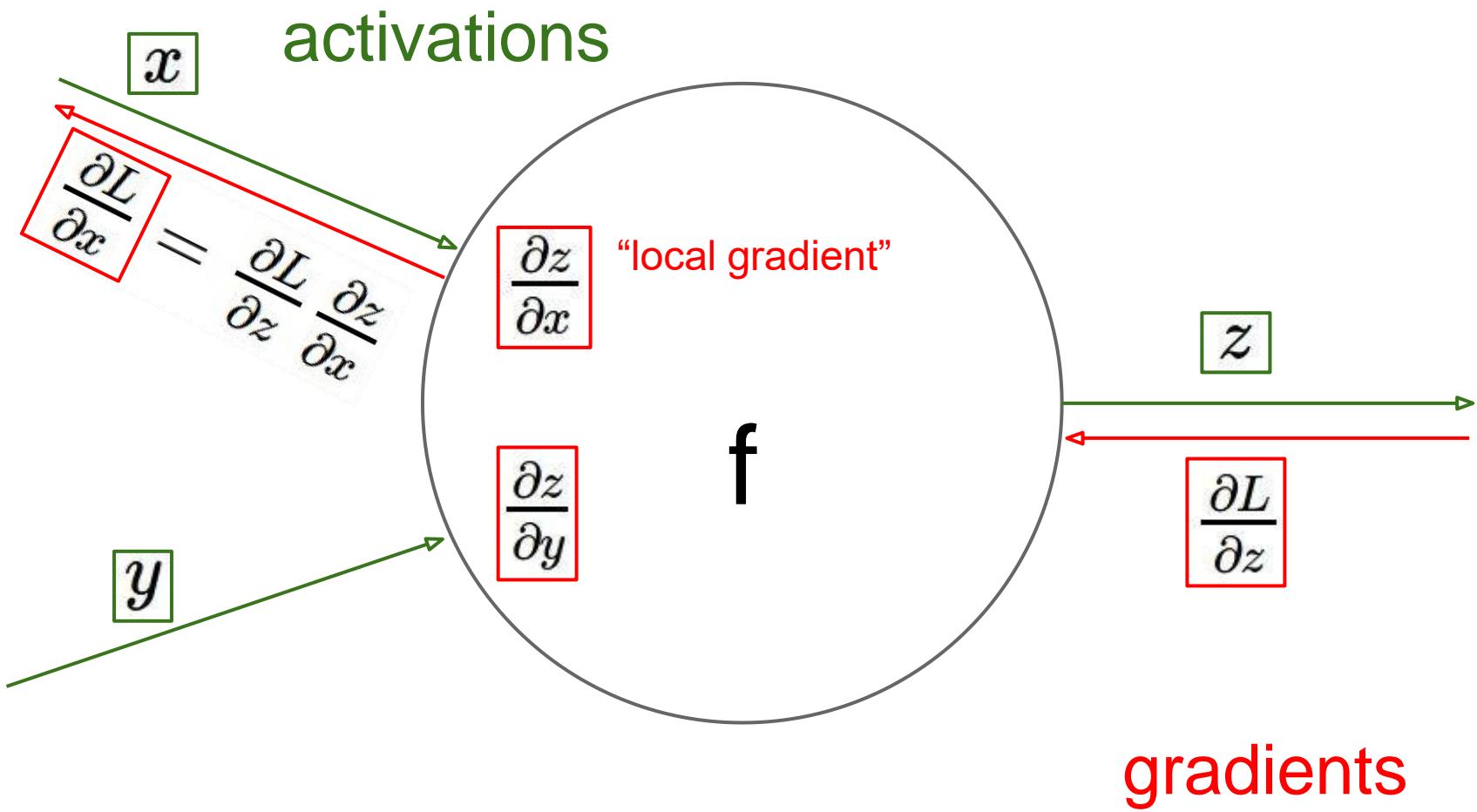
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

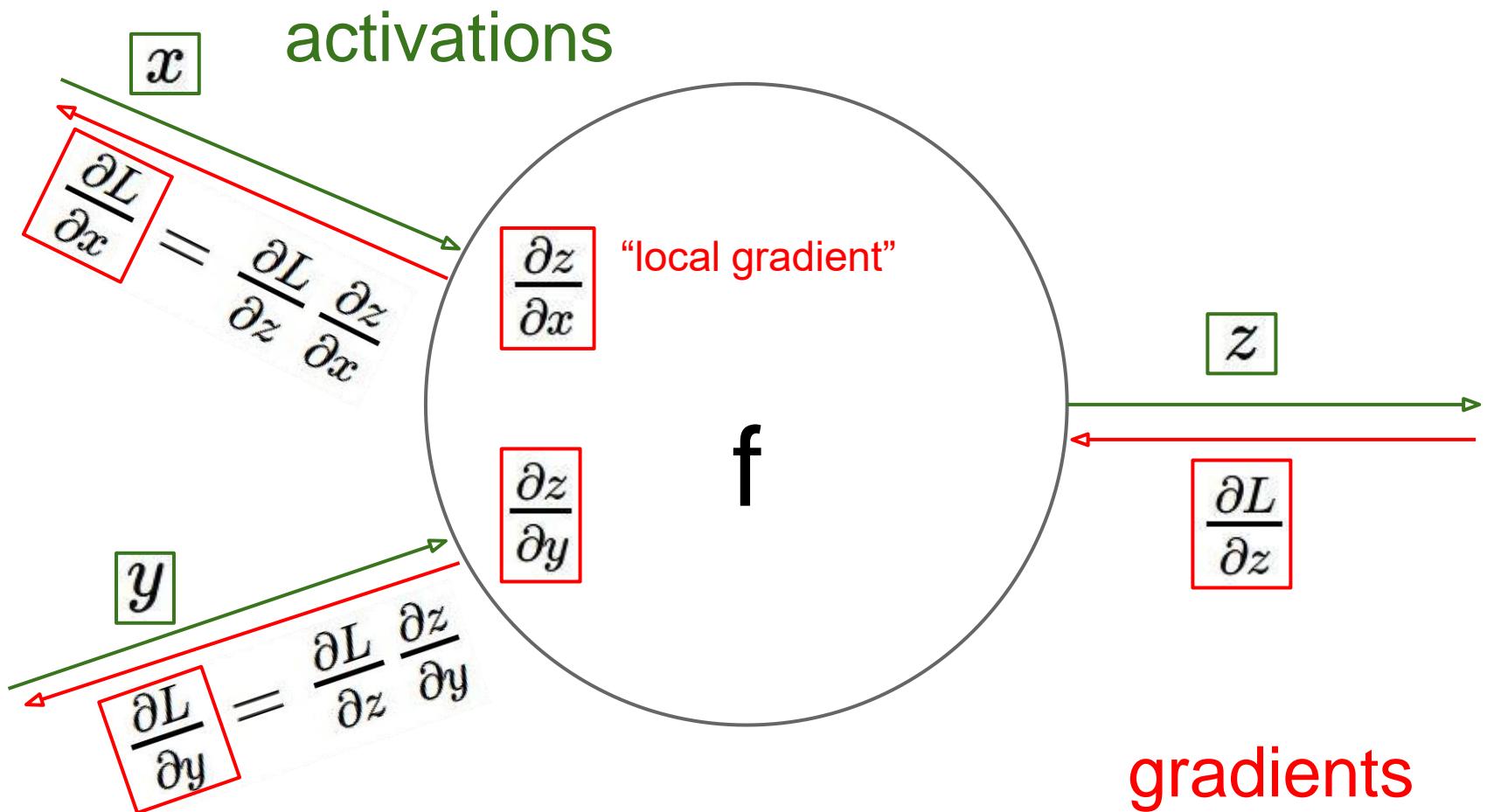
$$\frac{\partial f}{\partial x}$$









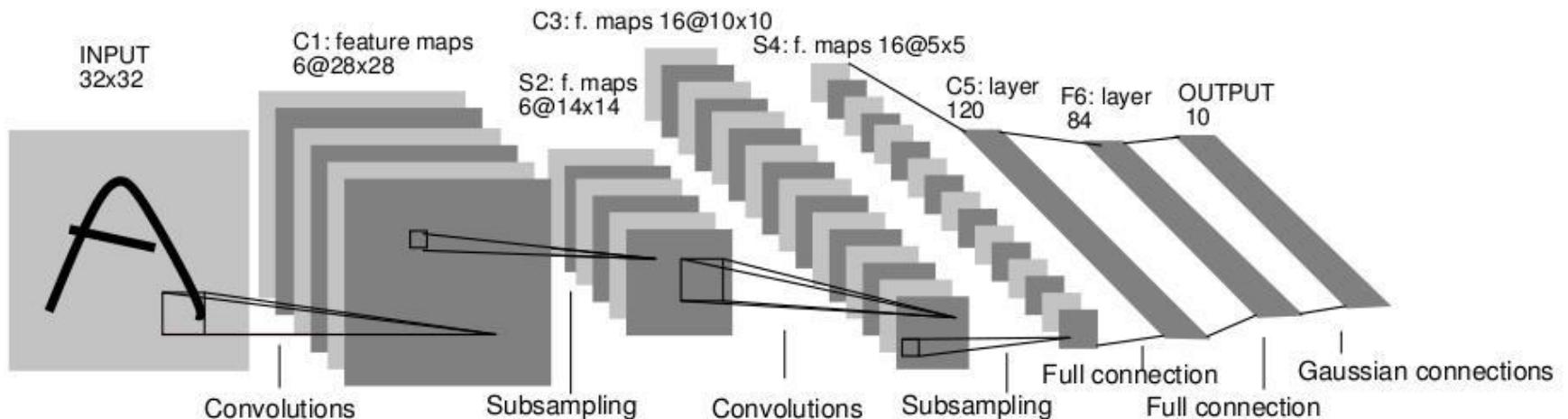
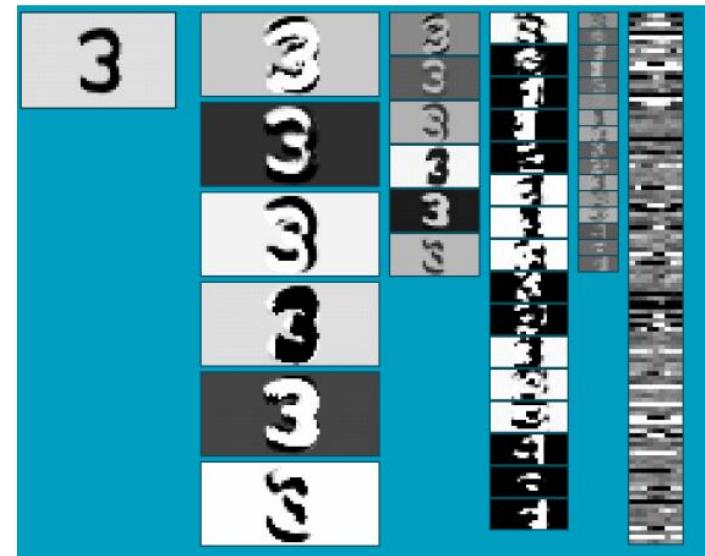


See hidden slides for more details.

# Convolutional neural networks

# Convolutional Neural Networks (CNN)

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant, *more abstract* features
- Classification layer at the end

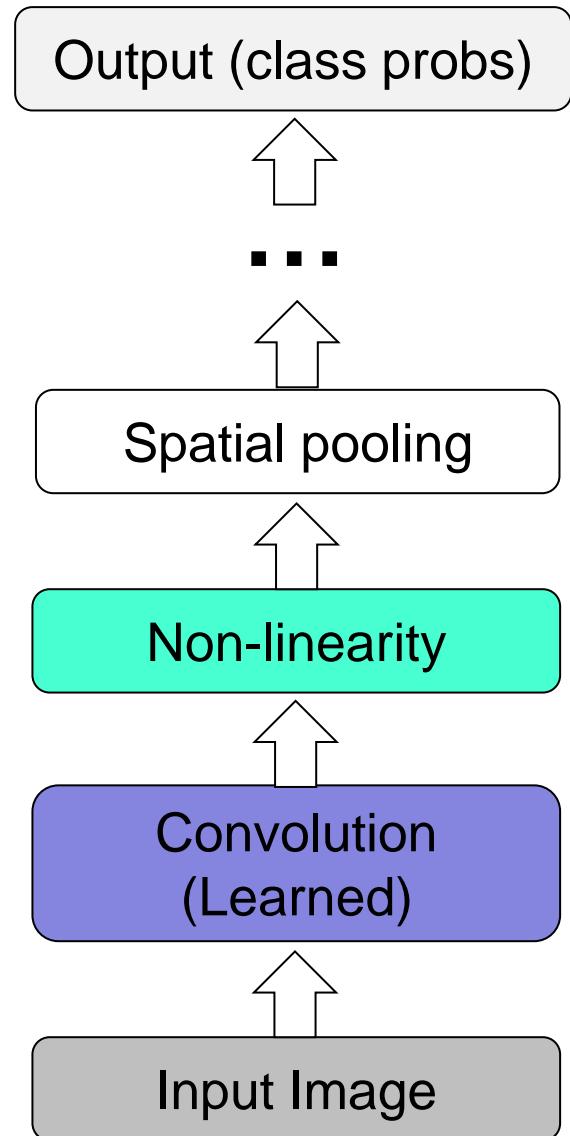


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, [Gradient-based learning applied to document recognition](#), Proceedings of the IEEE 86(11): 2278–2324, 1998.

Adapted from Rob Fergus

# Convolutional Neural Networks (CNN)

- Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Apply non-linearity 
  3. **Spatial pooling (downsample)**
- Supervised training of convolutional filters by back-propagating classification error



# 1. Convolution

---

- Apply learned filter weights
- One feature map per filter
-  Stride can be greater than 1 (faster, less memory)



Input

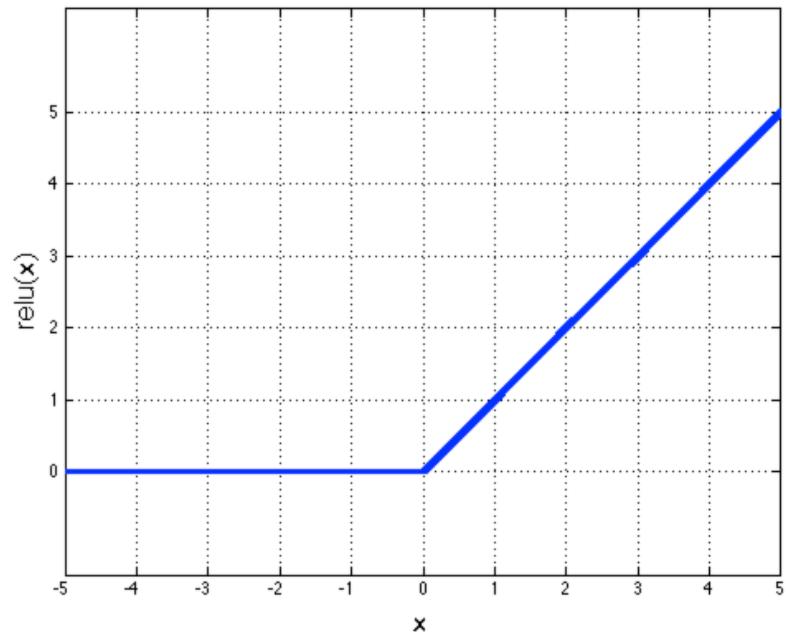
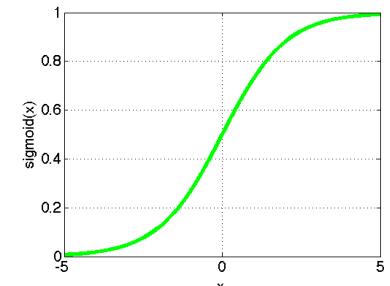
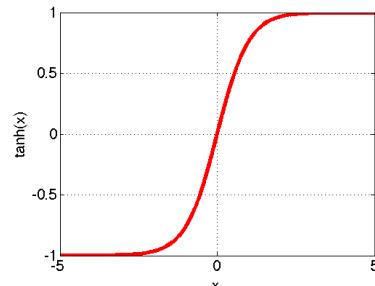


Feature Map

## 2. Non-Linearity

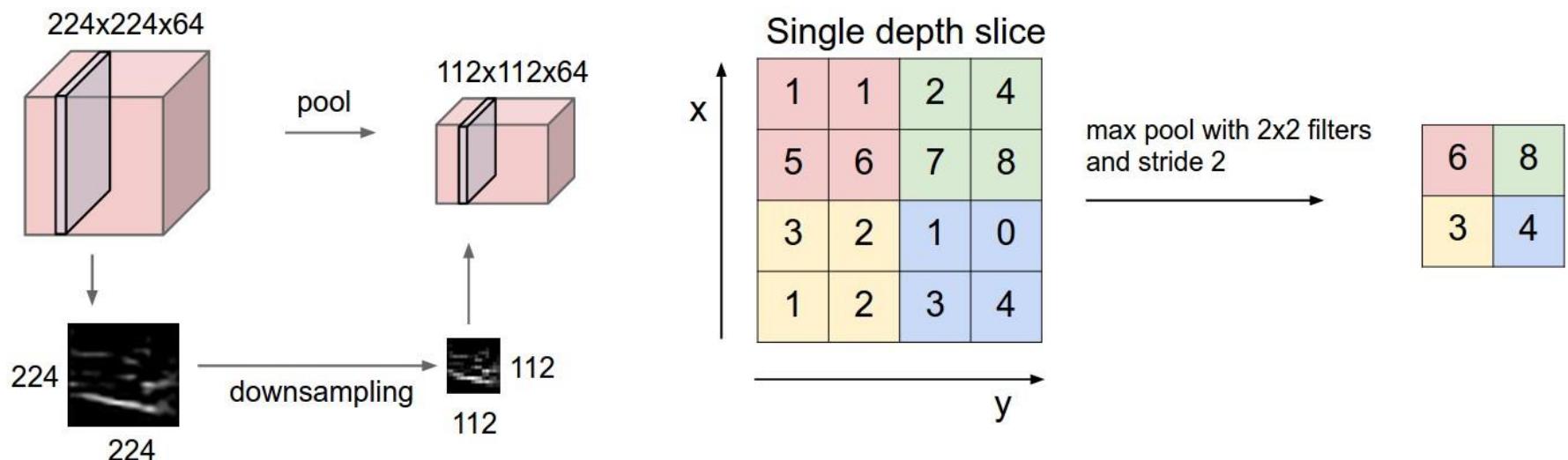
---

- Per-element (independent)
- Some options:
  - Tanh
  - Sigmoid:  $1/(1+\exp(-x))$
  - Rectified linear unit (ReLU)
    - Avoids saturation issues



### 3. Spatial Pooling

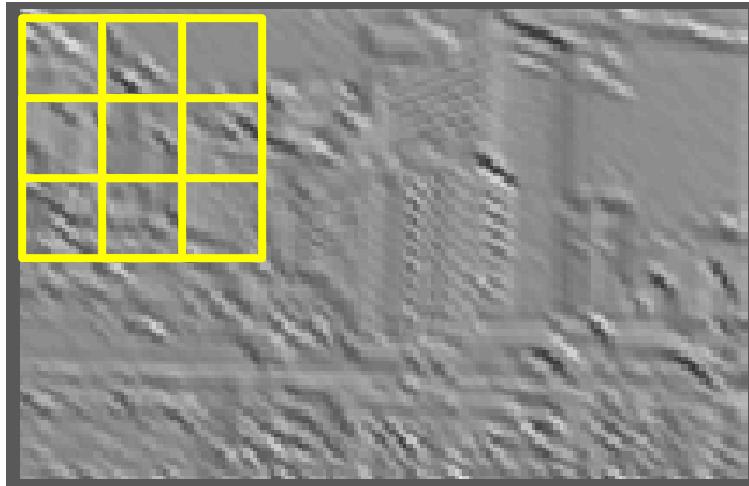
- Sum or max over non-overlapping / overlapping regions



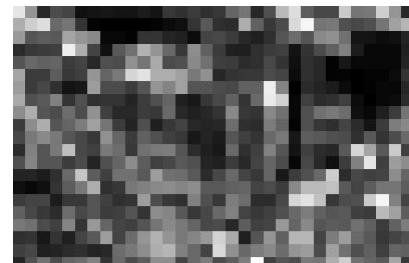
### 3. Spatial Pooling

---

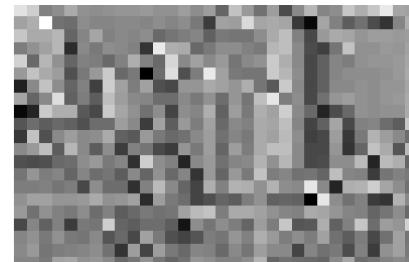
- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)



**Max**

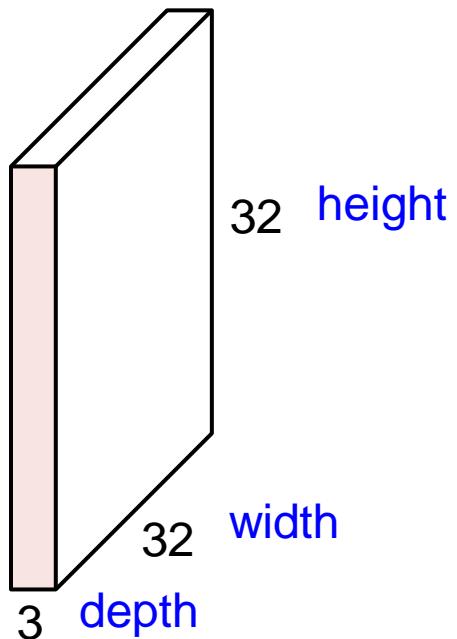


**Sum**



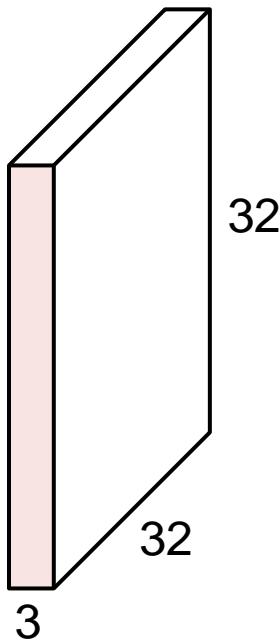
# Convolutions: More detail

32x32x3 image

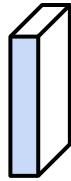


# Convolutions: More detail

32x32x3 image



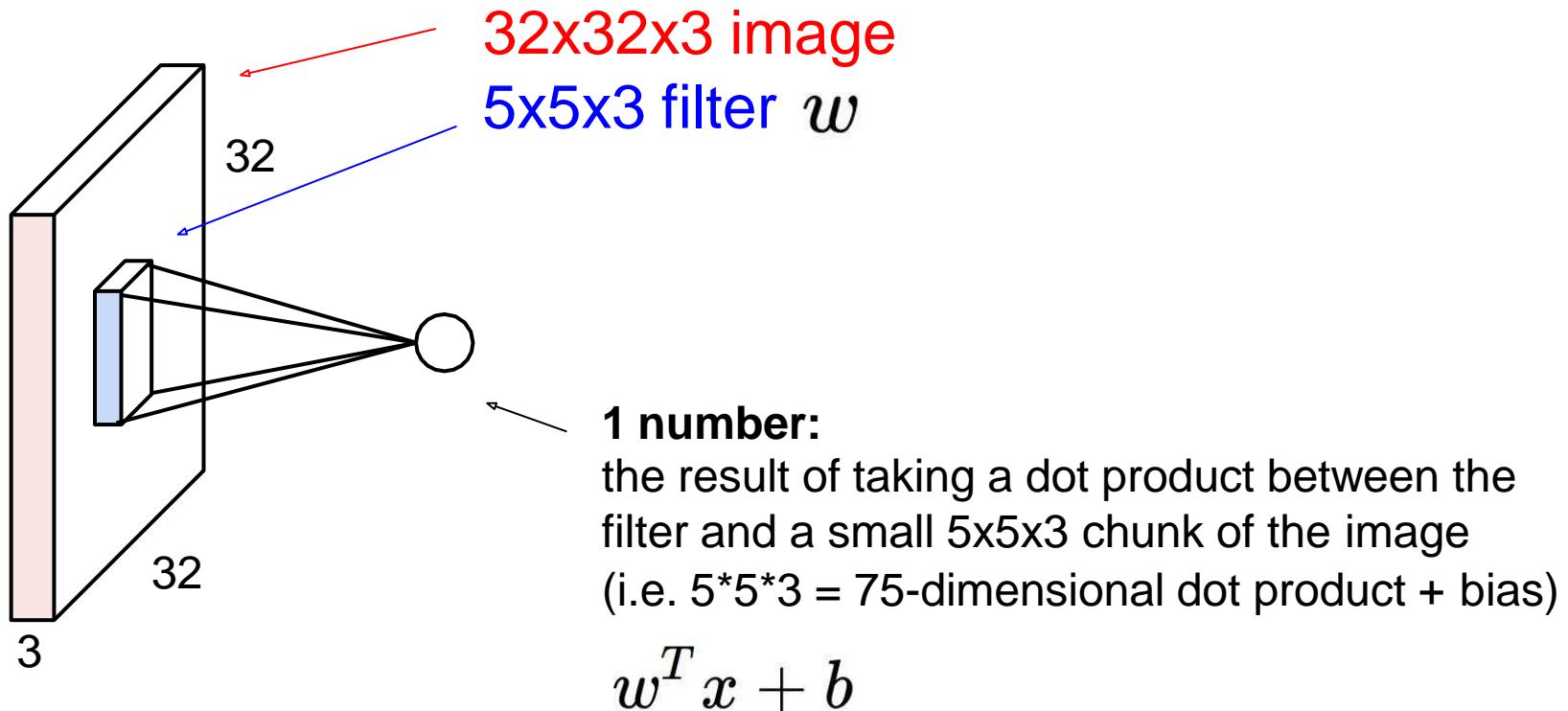
5x5x3 filter



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

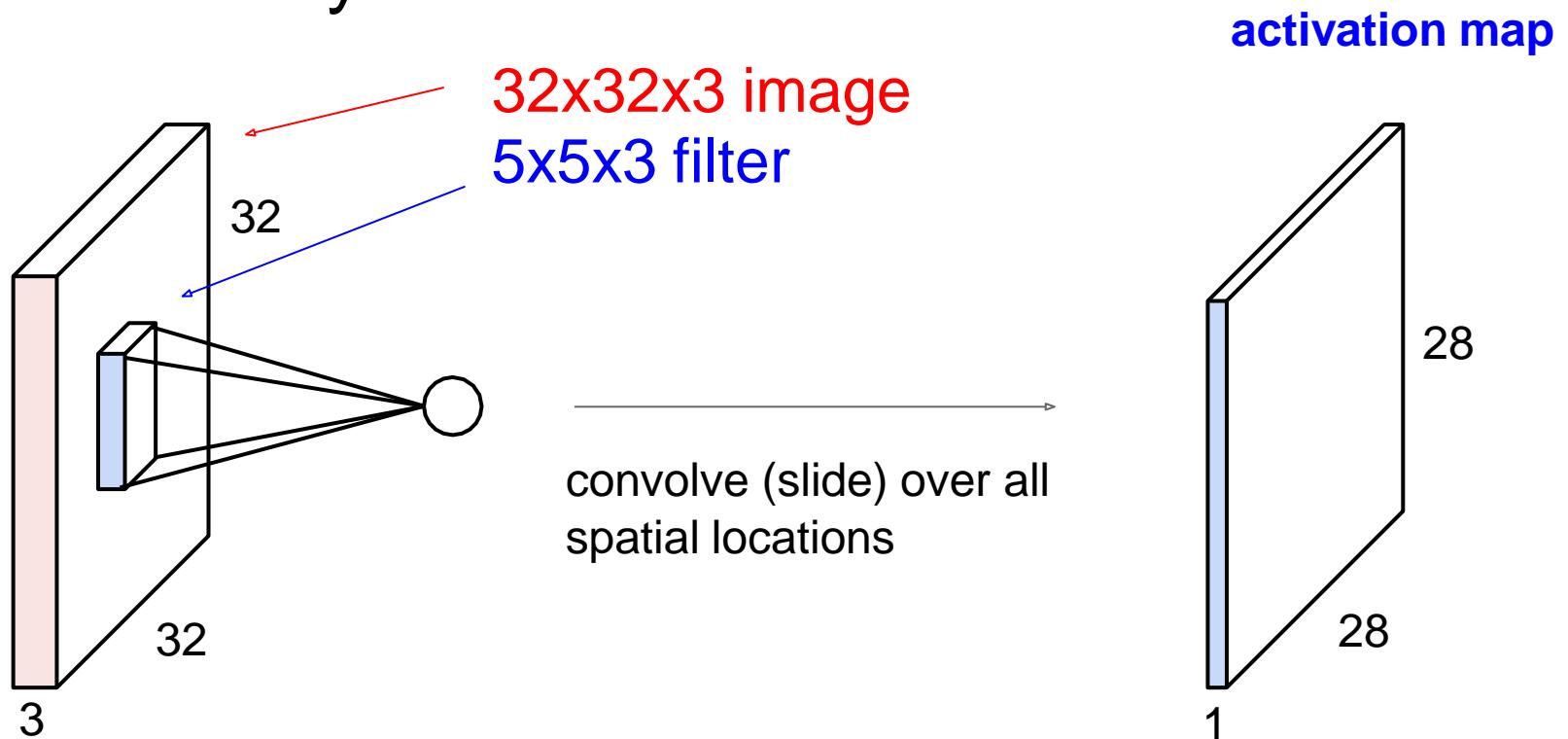
# Convolutions: More detail

## Convolution Layer



# Convolutions: More detail

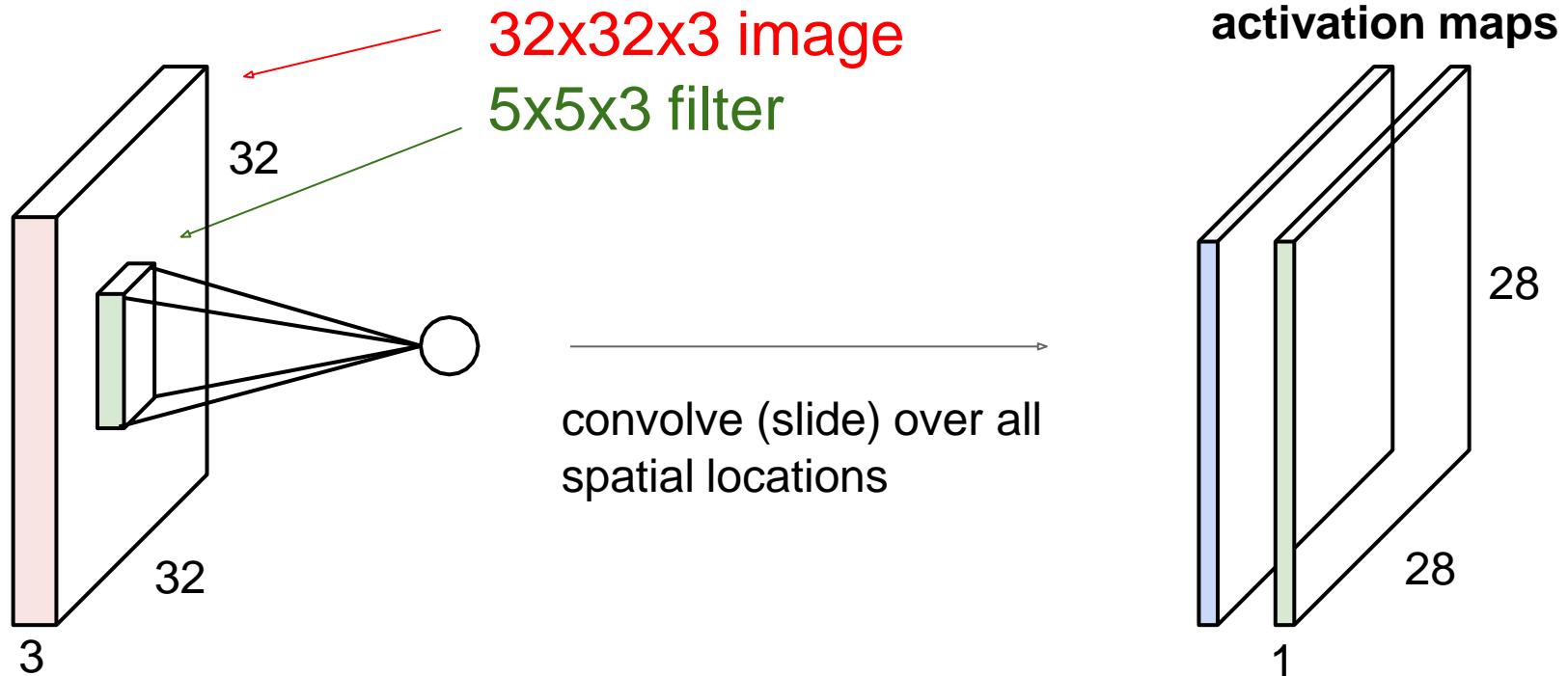
## Convolution Layer



# Convolutions: More detail

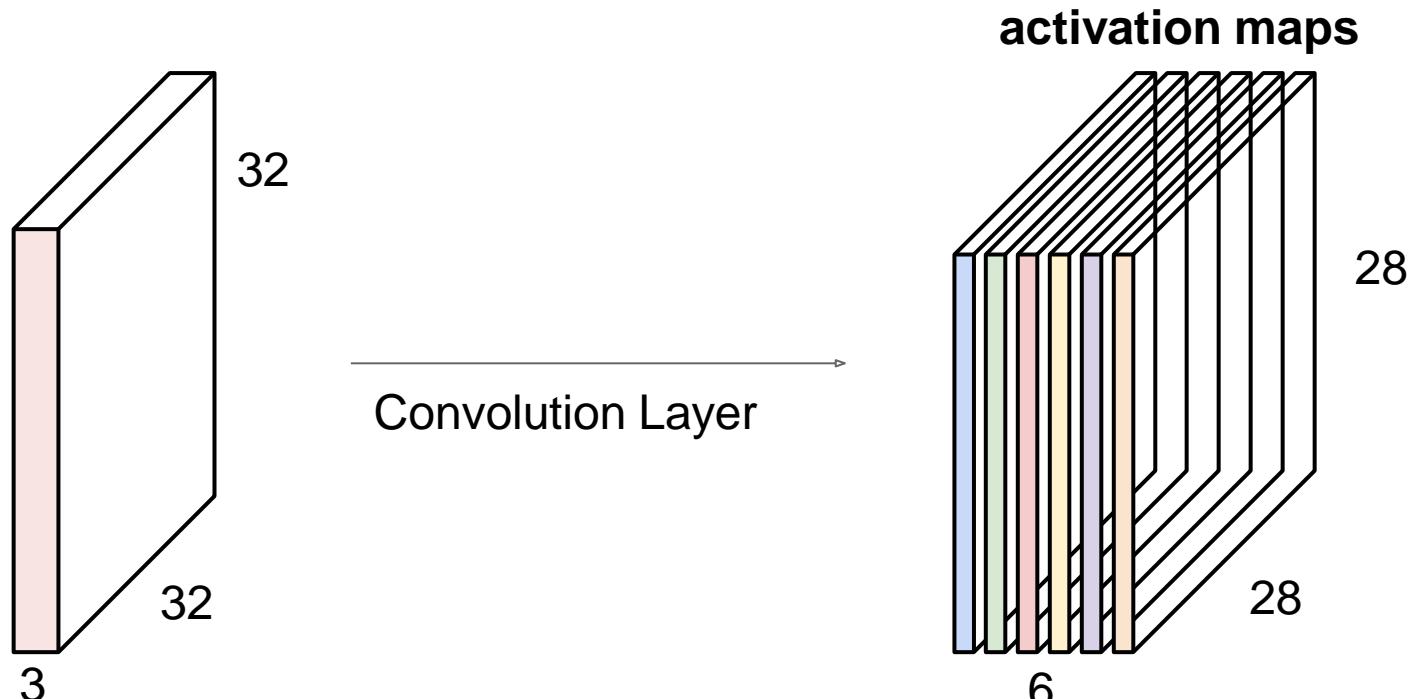
## Convolution Layer

consider a second, green filter



# Convolutions: More detail

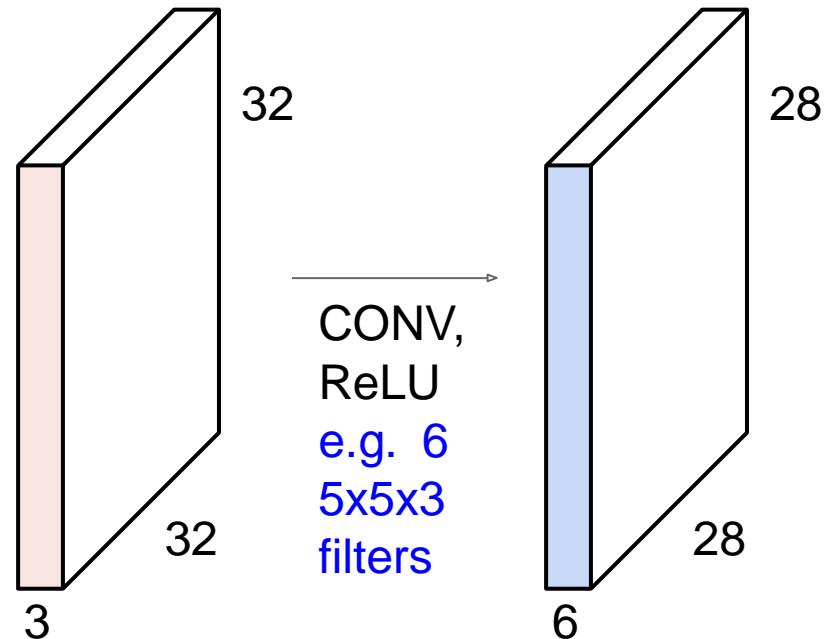
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size  $28 \times 28 \times 6$ !

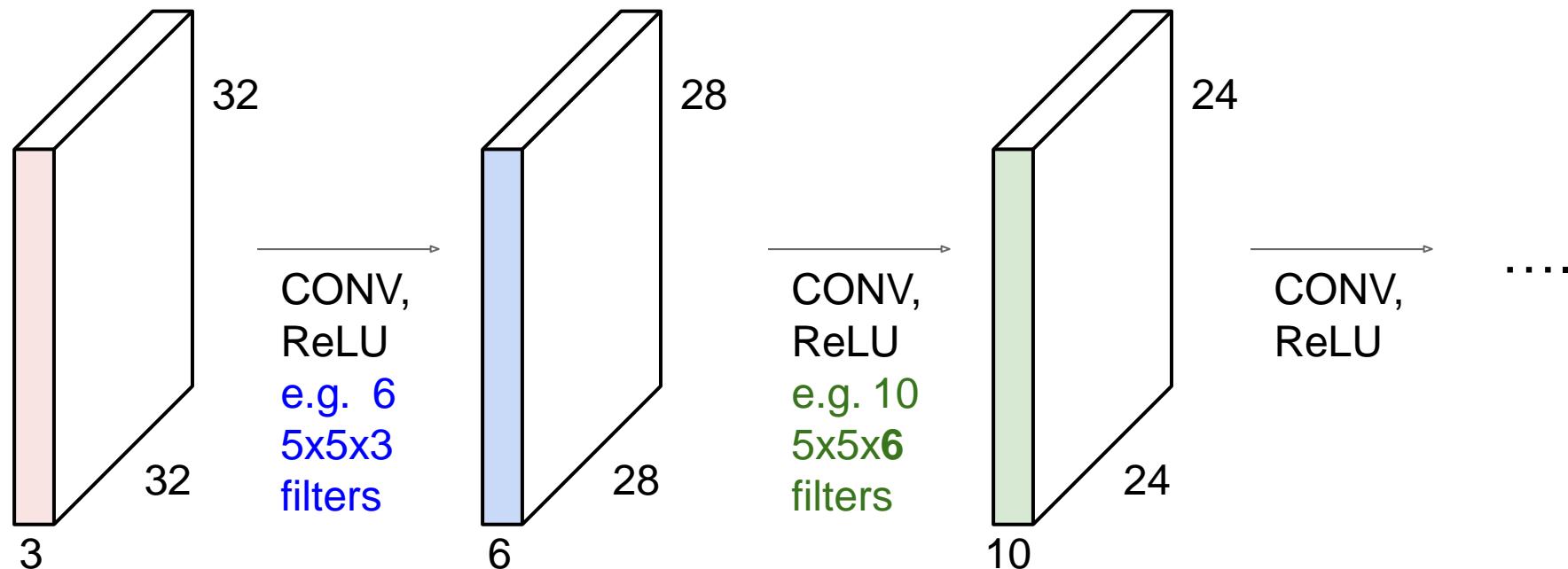
# Convolutions: More detail

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



# Convolutions: More detail

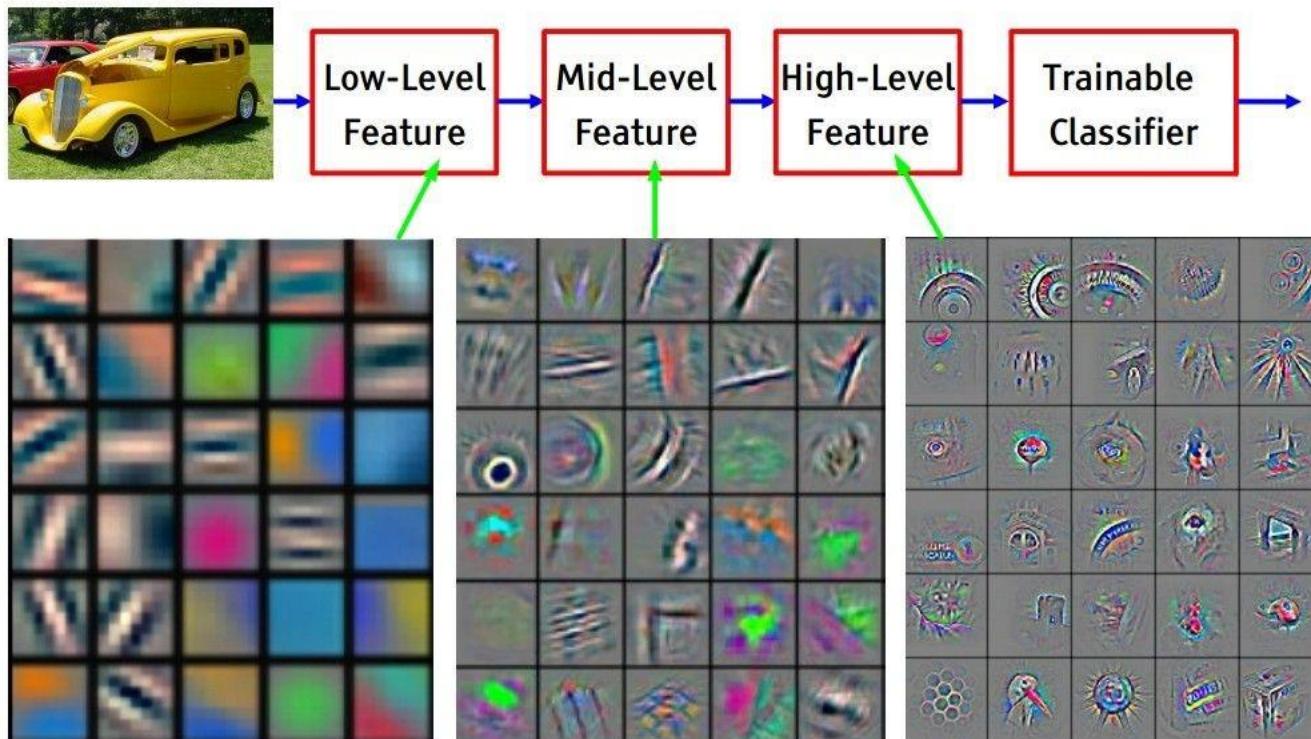
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



# Convolutions: More detail

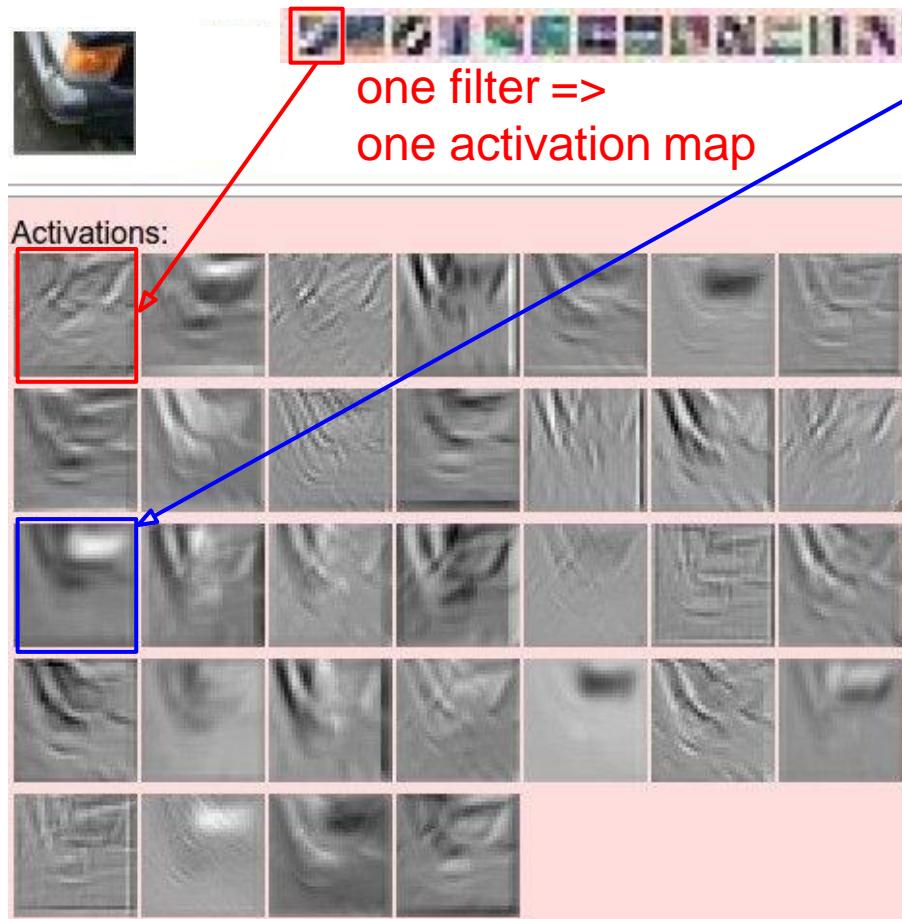
## Preview

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Convolutions: More detail



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

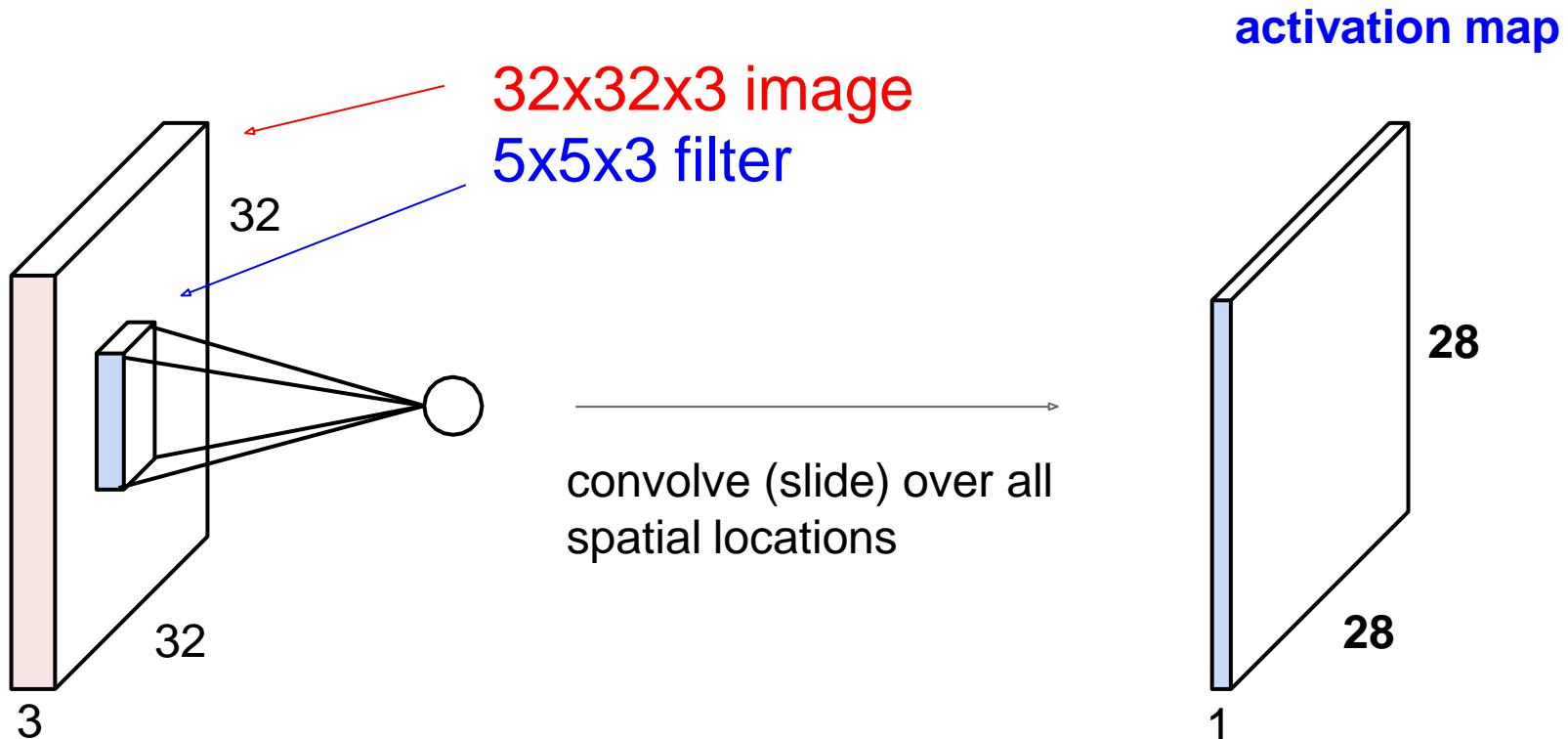
$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v]F[i + u, j + v]$$



Element-wise multiplication and sum of a filter and the signal (image)

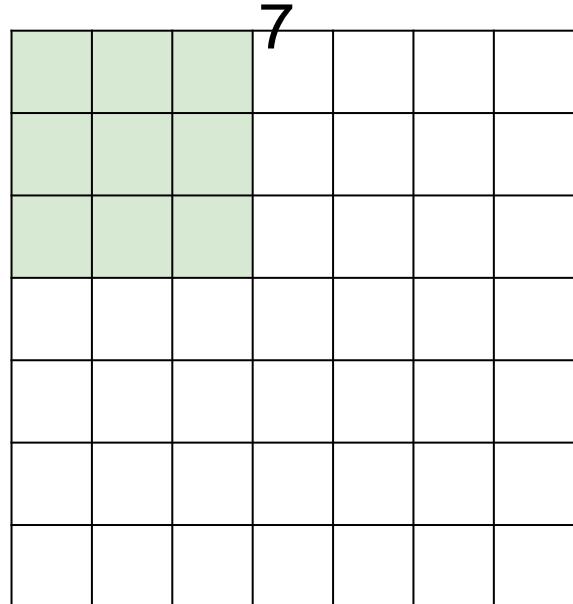
# Convolutions: More detail

A closer look at spatial dimensions:



# Convolutions: More detail

A closer look at spatial dimensions:

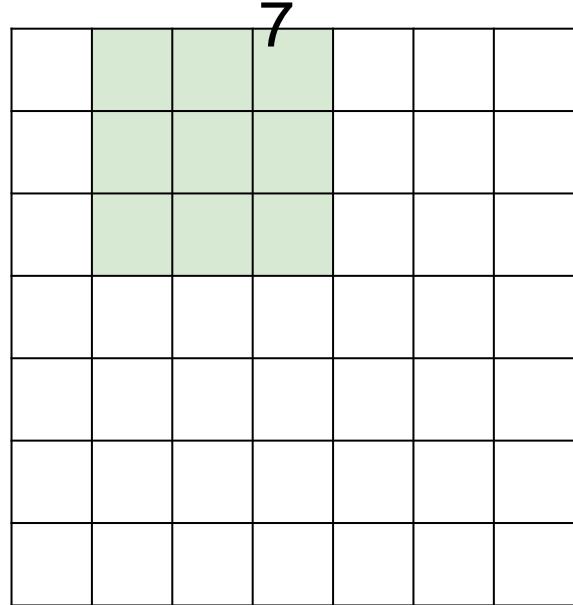


7x7 input (spatially)  
assume 3x3 filter

7

# Convolutions: More detail

A closer look at spatial dimensions:

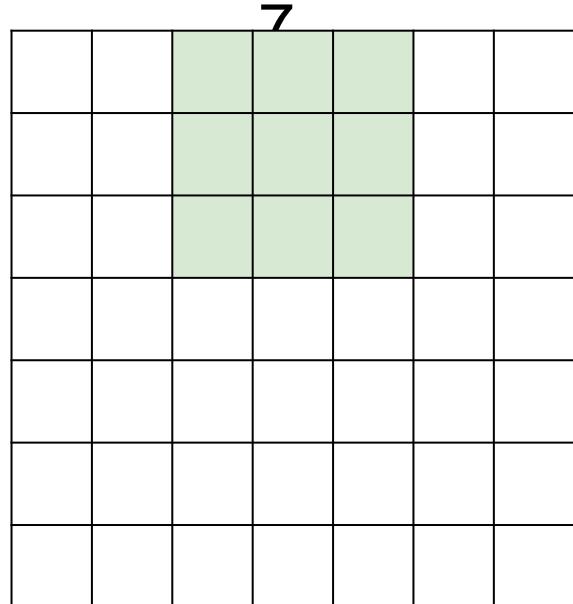


7x7 input (spatially)  
assume 3x3 filter

7

# Convolutions: More detail

A closer look at spatial dimensions:

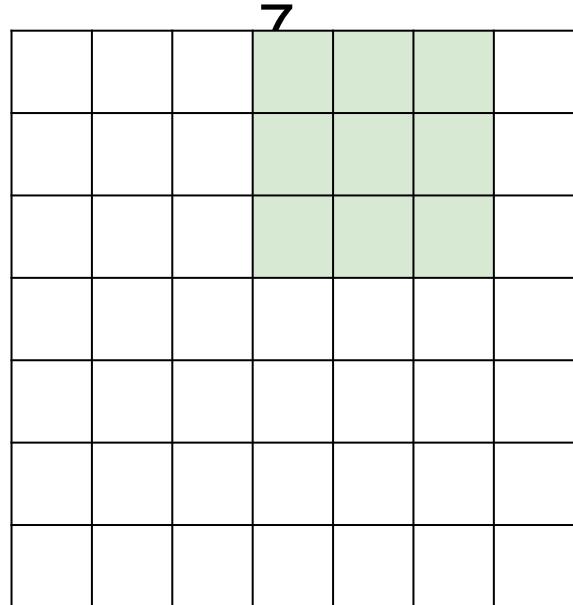


7x7 input (spatially)  
assume 3x3 filter

7

# Convolutions: More detail

A closer look at spatial dimensions:

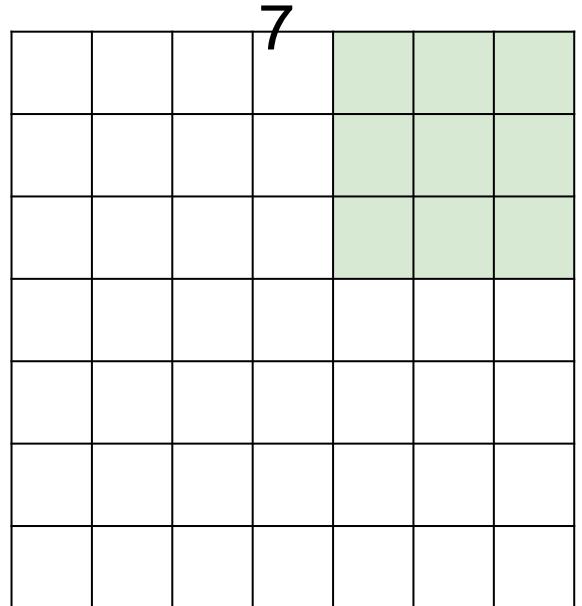


7x7 input (spatially)  
assume 3x3 filter

7

# Convolutions: More detail

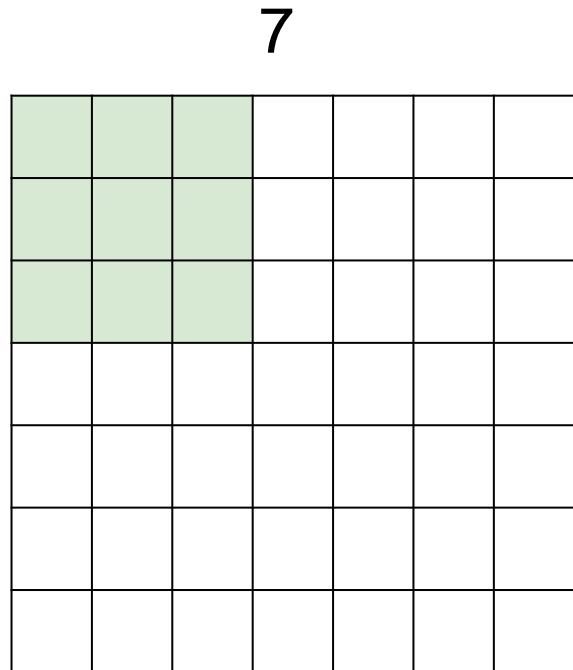
A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter  
**=> 5x5 output**

# Convolutions: More detail

A closer look at spatial dimensions:

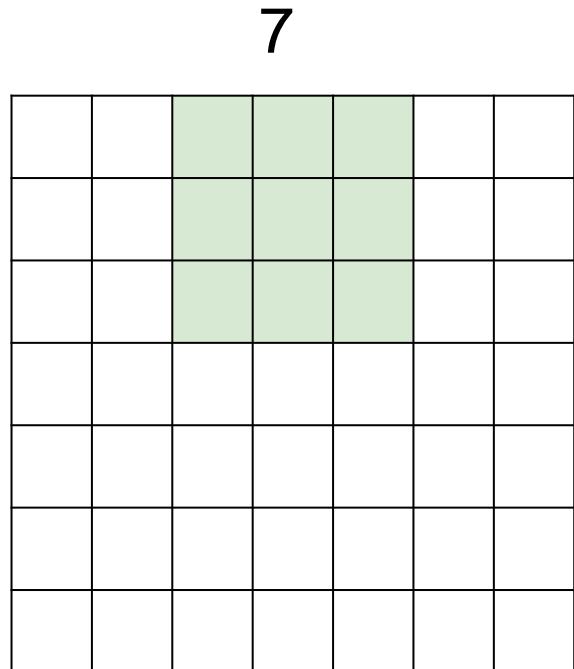


7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

7

# Convolutions: More detail

A closer look at spatial dimensions:

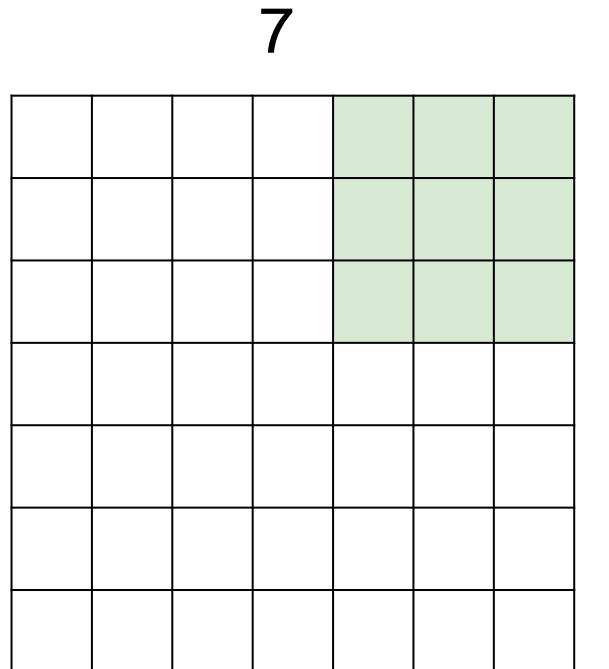


7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

7

# Convolutions: More detail

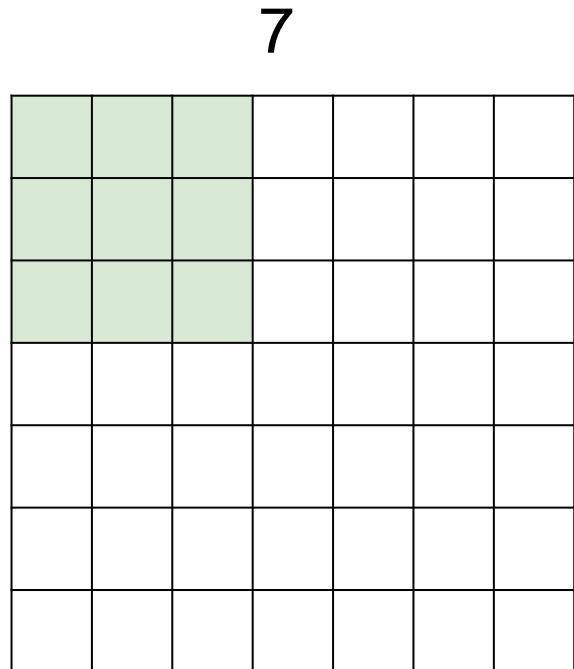
A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**  
**=> 3x3 output!**

# Convolutions: More detail

A closer look at spatial dimensions:

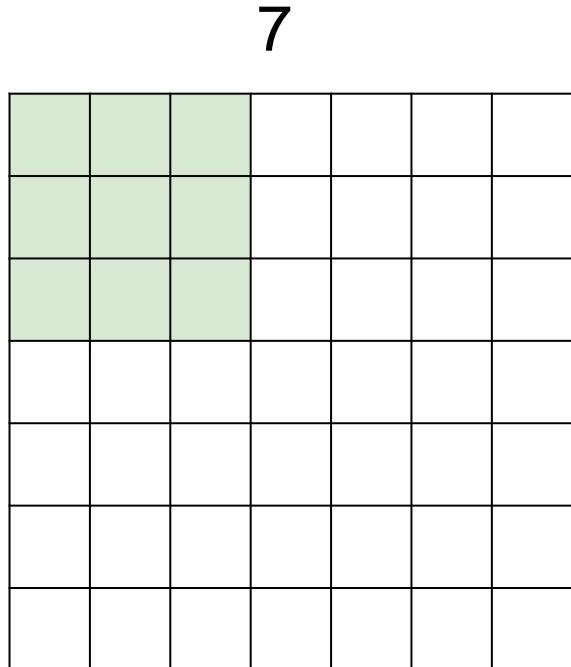


7

7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 3?**

# Convolutions: More detail

A closer look at spatial dimensions:



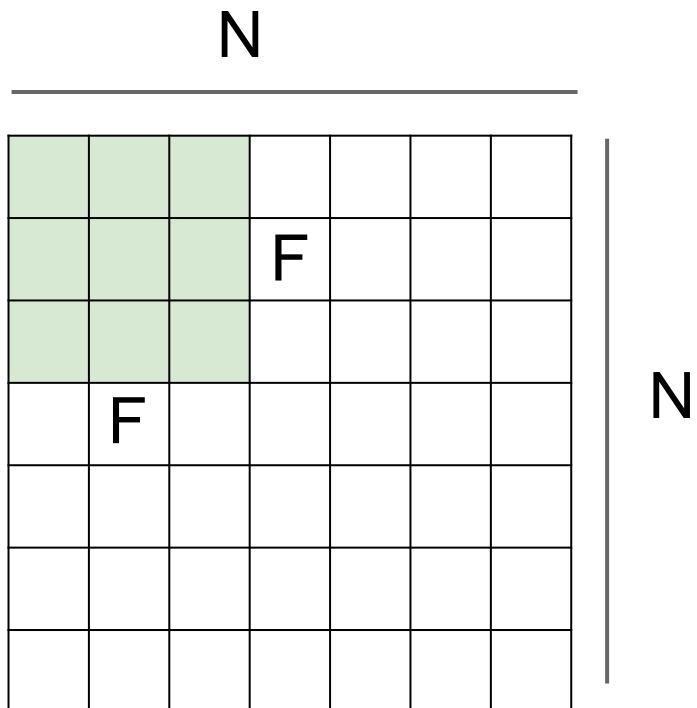
7

7

7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 3?**

**doesn't fit!**  
cannot apply 3x3 filter on  
7x7 input with stride 3.

# Convolutions: More detail



Output size:  
**(N - F) / stride + 1**

e.g.  $N = 7$ ,  $F = 3$ :  
stride 1 =>  $(7 - 3)/1 + 1 = 5$   
stride 2 =>  $(7 - 3)/2 + 1 = 3$   
stride 3 =>  $(7 - 3)/3 + 1 = 2.33$  :\

# Convolutions: More detail

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3** filter, applied with **stride 1**

**pad with 1 pixel border => what is the output?**

(recall:)

$$(N - F) / \text{stride} + 1$$

# Convolutions: More detail

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

# Convolutions: More detail

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

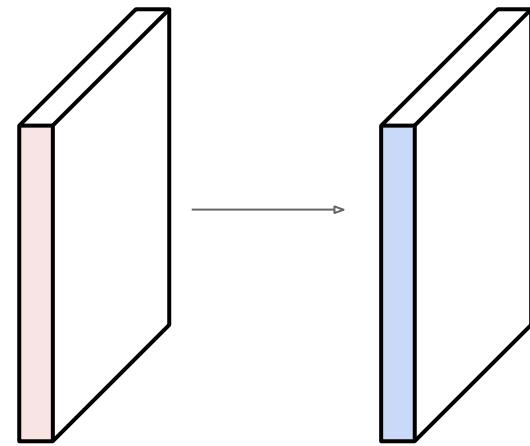
$$(N + 2 * \text{padding} - F) / \text{stride} + 1$$

# Convolutions: More detail

Examples time:

Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2



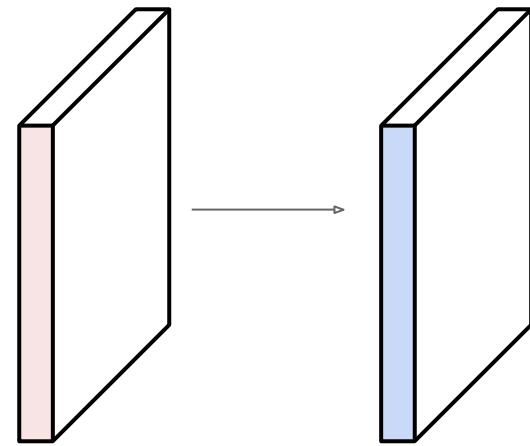
Output volume size: ?

# Convolutions: More detail

Examples time:

Input volume: **32x32x3**

**10 5x5x3** filters with stride 1, pad 2



Output volume size:

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

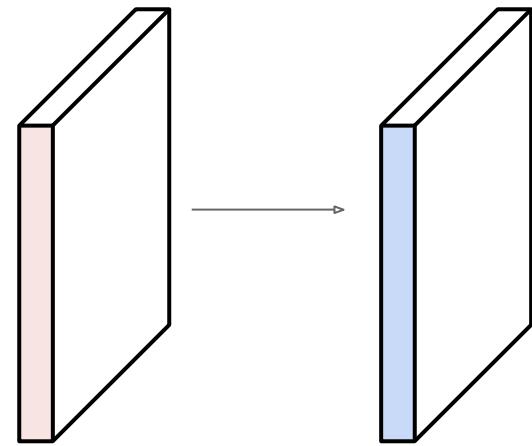
**32x32x10**

# Convolutions: More detail

Examples time:

Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2



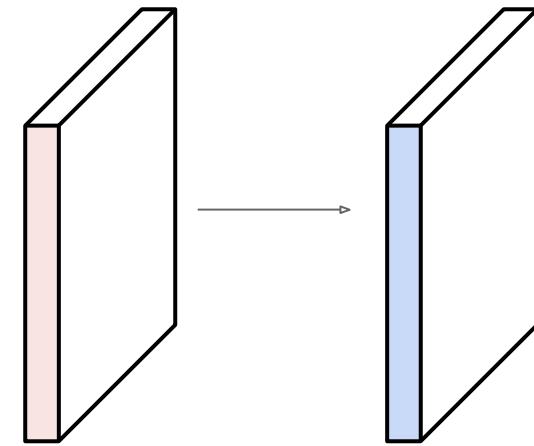
Number of parameters in this layer?

# Convolutions: More detail

Examples time:

Input volume: **32x32x3**

**10 5x5x3** filters with stride 1, pad 2



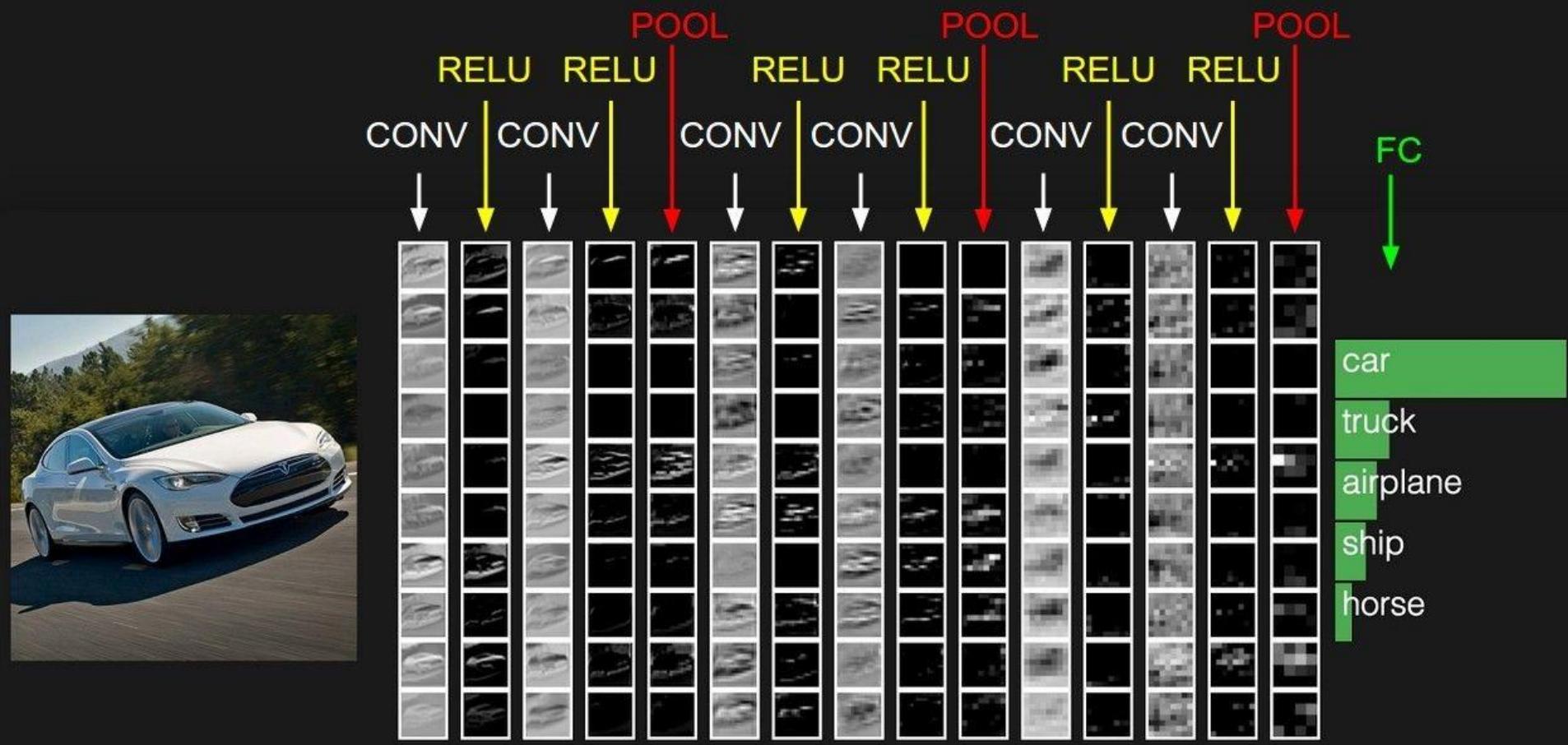
Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params

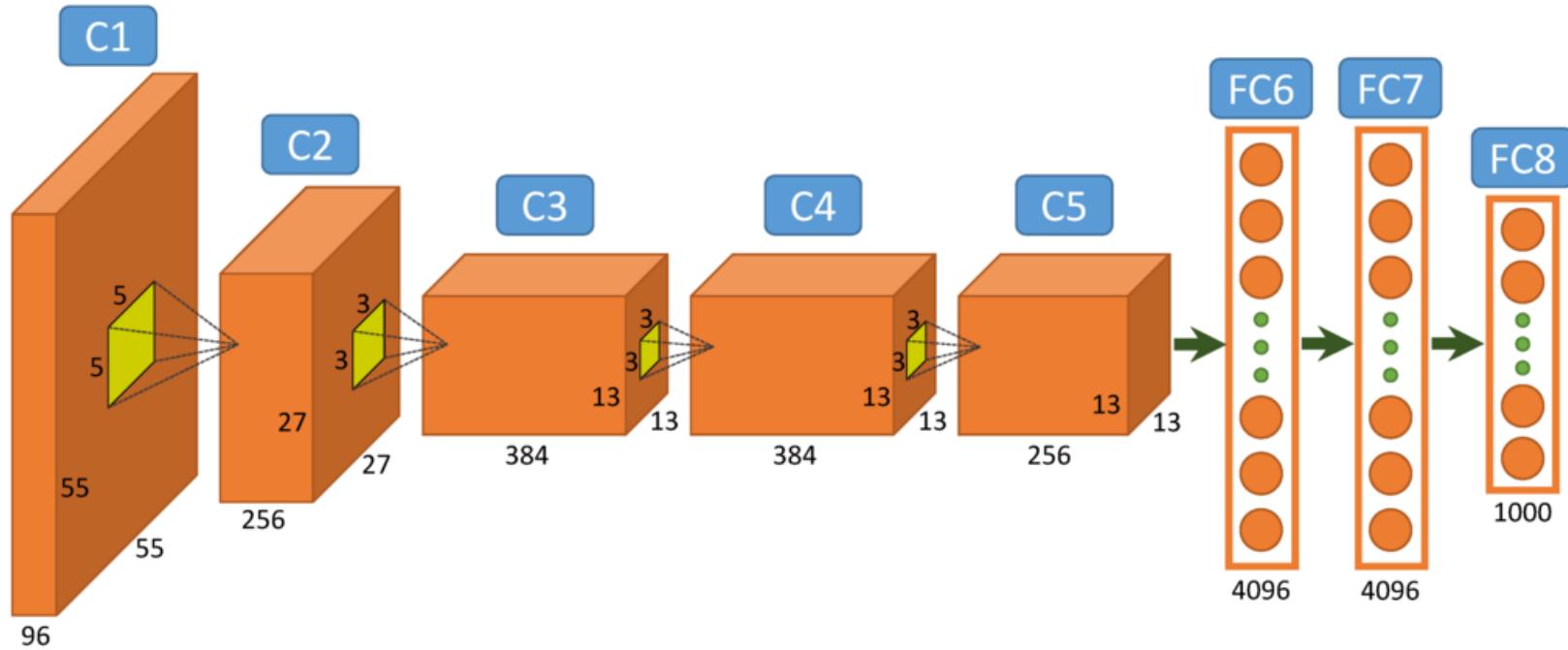
(+1 for bias)

$$\Rightarrow 76 * 10 = 760$$

# Putting it all together



# A Common Architecture: AlexNet



# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1  
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error

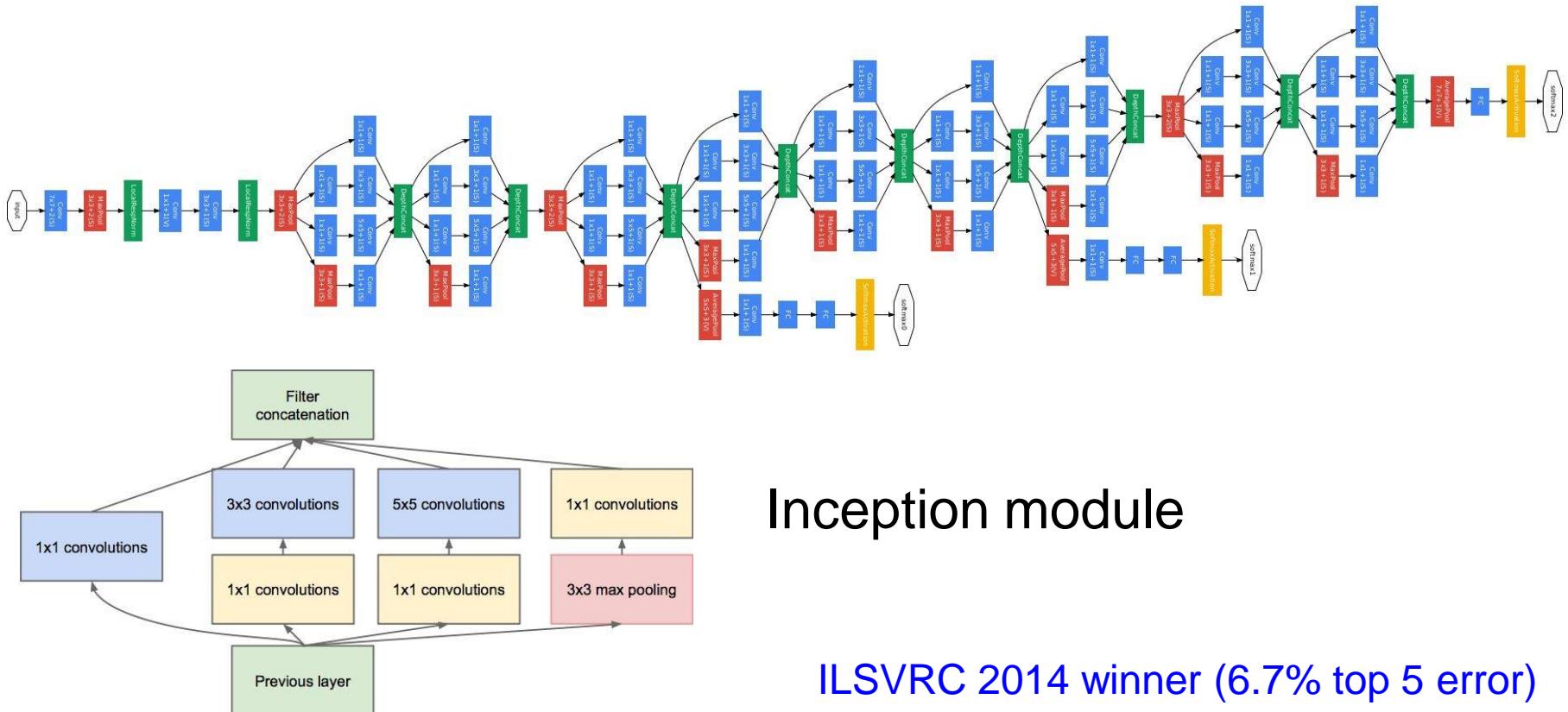
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 <b>conv3-256</b> <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

# Case Study: GoogLeNet

[Szegedy et al., 2014]



# Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer nets**
  - ImageNet Detection: **16%** better than 2nd
  - ImageNet Localization: **27%** better than 2nd
  - COCO Detection: **11%** better than 2nd
  - COCO Segmentation: **12%** better than 2nd

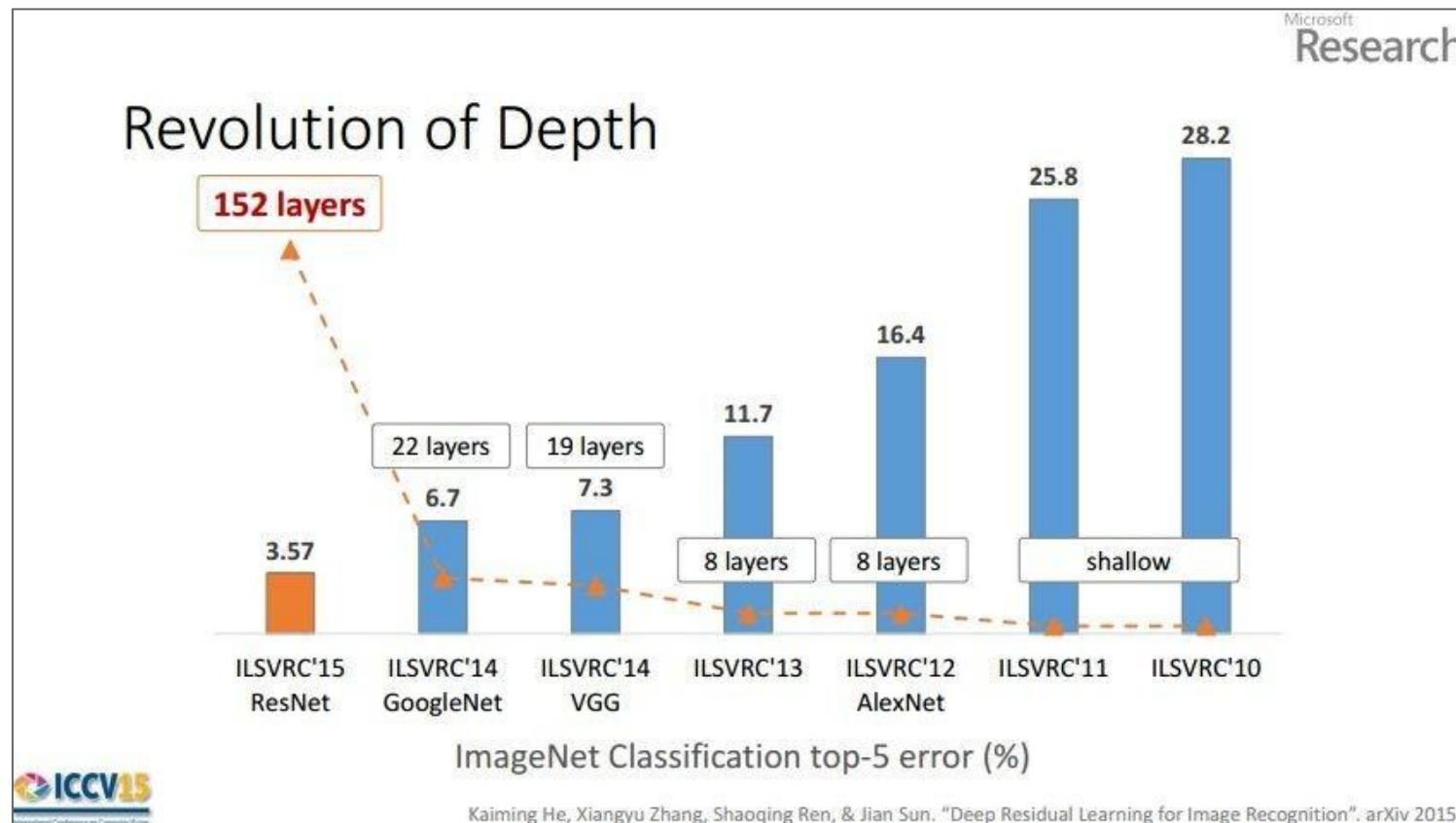
\*improvements are relative numbers

ICCV15  
International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

Slide from Kaiming He's presentation <https://www.youtube.com/watch?v=1PGLj-uKT1w>

# Case Study: ResNet



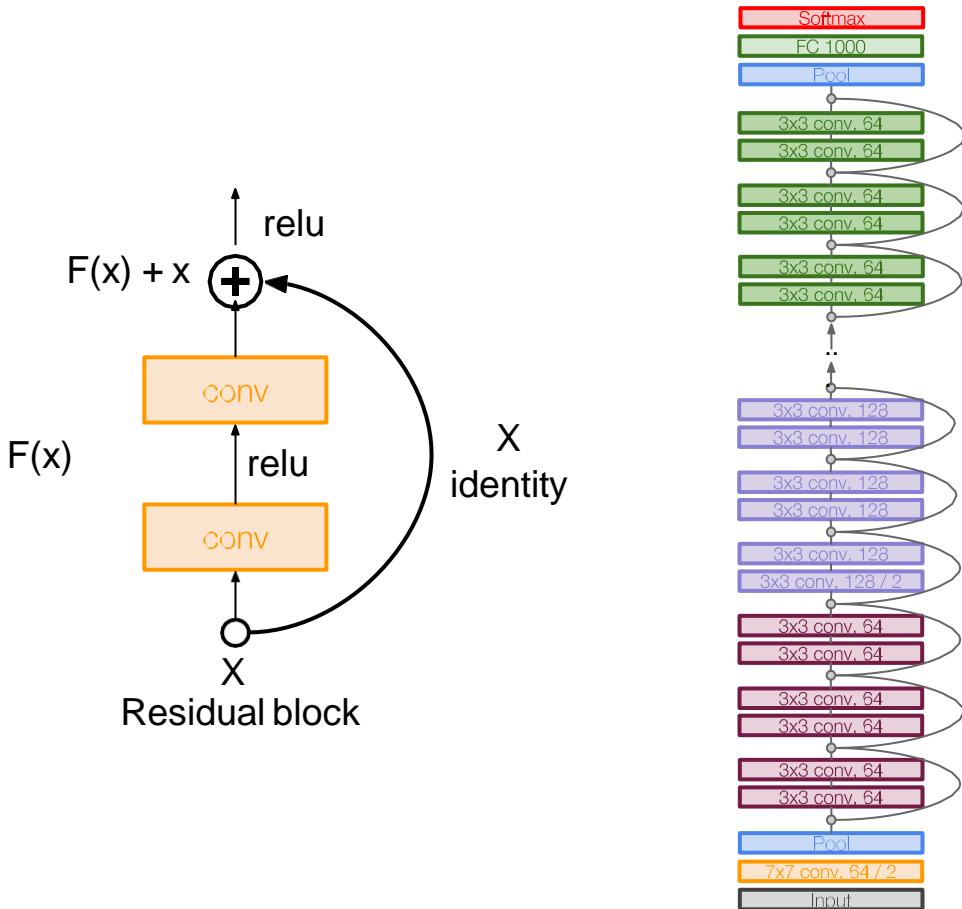
(slide from Kaiming He's presentation)

# Case Study: ResNet

[He et al., 2016]

Very deep networks using residual connections

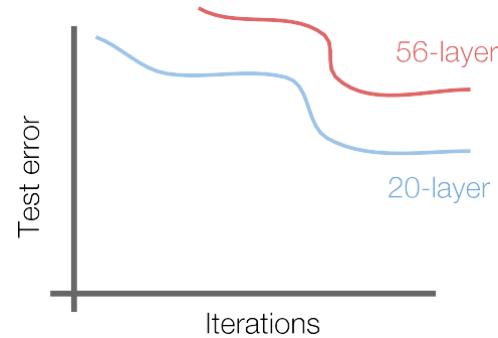
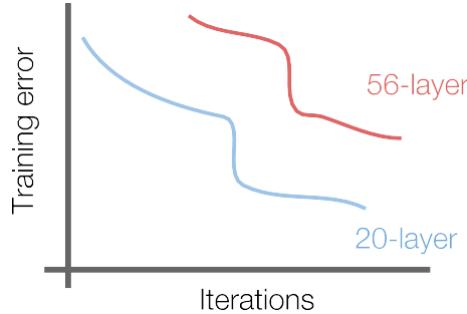
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



# Case Study: ResNet

[He et al., 2016]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



Q: What's strange about these training and test curves?

[Hint: look at the order of the curves]

56-layer model performs worse on both training and test error

-> The deeper model performs worse, but it's not caused by overfitting!

# Case Study: ResNet

[He et al., 2016]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

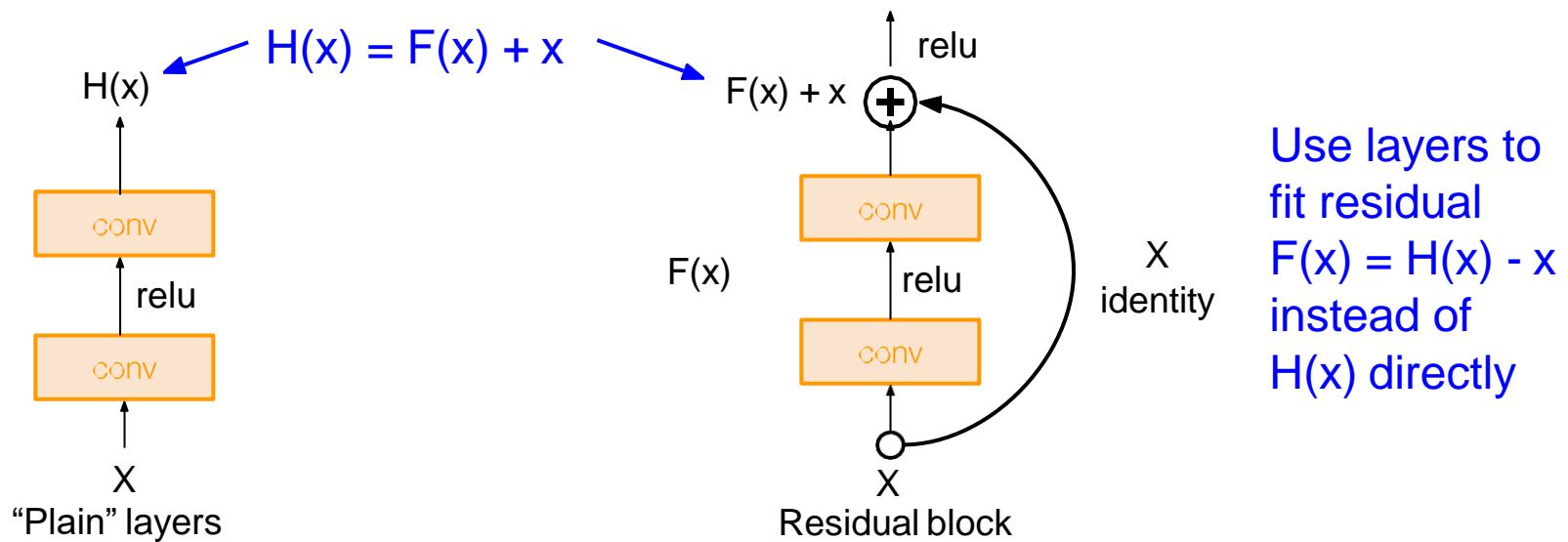
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

# Case Study: ResNet

[He et al., 2016]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

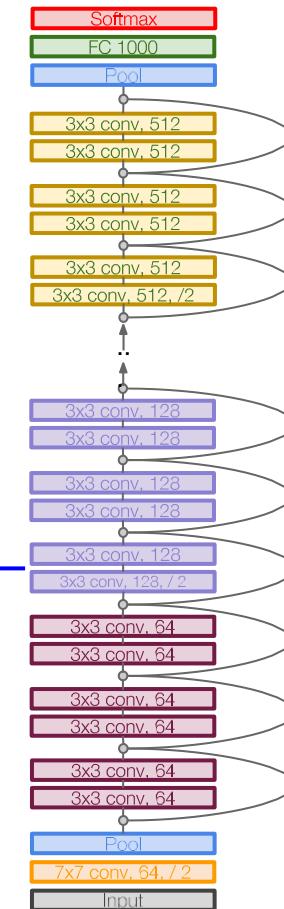
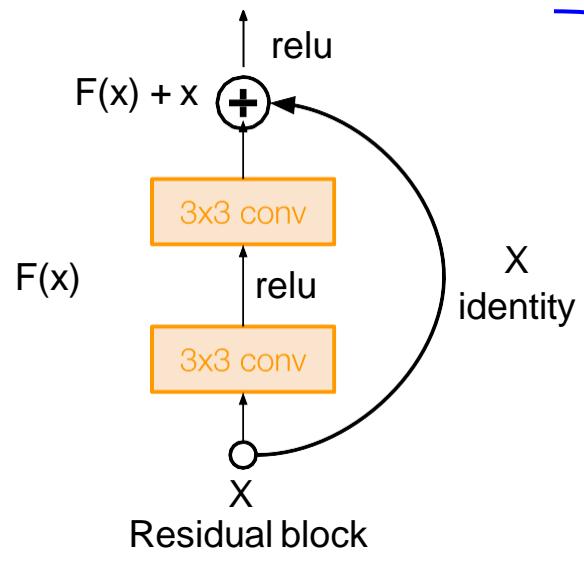


# Case Study: ResNet

[He et al., 2016]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers



# Practical matters

# Comments on training algorithm

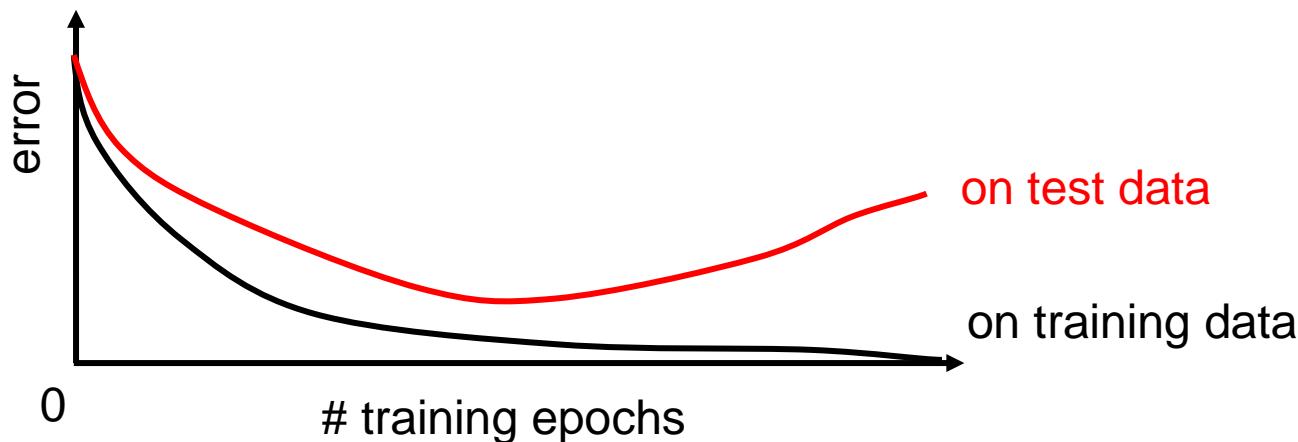
---

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data, with good choice of hyperparameters (e.g. learning rate).
- Thousands of epochs (epoch = network sees all training data once) may be required, hours or days to train.
- To avoid local-minima problems, run several trials starting with different random weights (*random restarts*), and take results of trial with lowest training set error, use transfer learning, regularization, ...
- May be hard to set learning rate and to select number of hidden units and layers.
- Neural networks had fallen out of fashion in 90s, early 2000s; back with a new name and improved performance (**deep networks trained with dropout and lots of data**).

# Over-training prevention

---

- Running too many epochs can result in over-fitting.



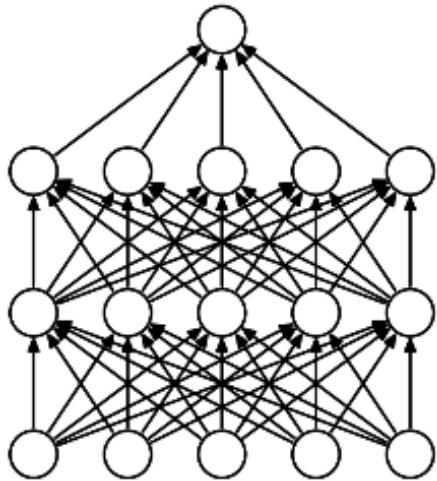
- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.

# Training: Best practices

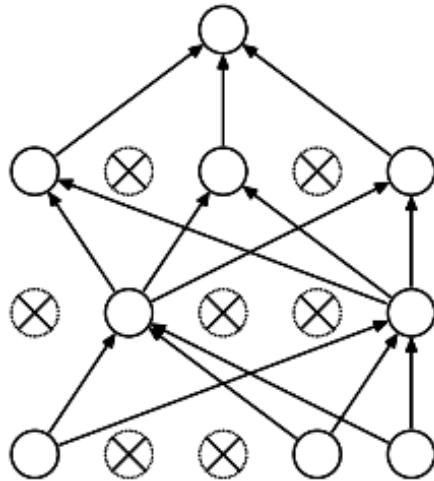
---

- Data
  - Center (subtract mean from) your data
  - Use data augmentation
  - Use mini-batch
- Weights/activations
  - To initialize weights, use “Xavier initialization”
  - Use regularization
  - Use RELU (most common), don’t use sigmoid
- Hyperparameters:
  - Learning rate: too high? Too low?
  - Use cross-validation to pick

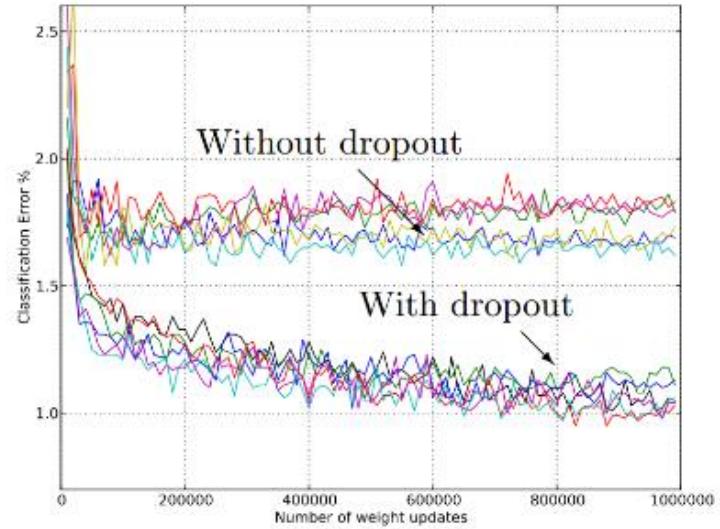
# Regularization: Dropout



(a) Standard Neural Net



(b) After applying dropout.



- Randomly turn off some neurons
- Allows individual neurons to independently be responsible for performance

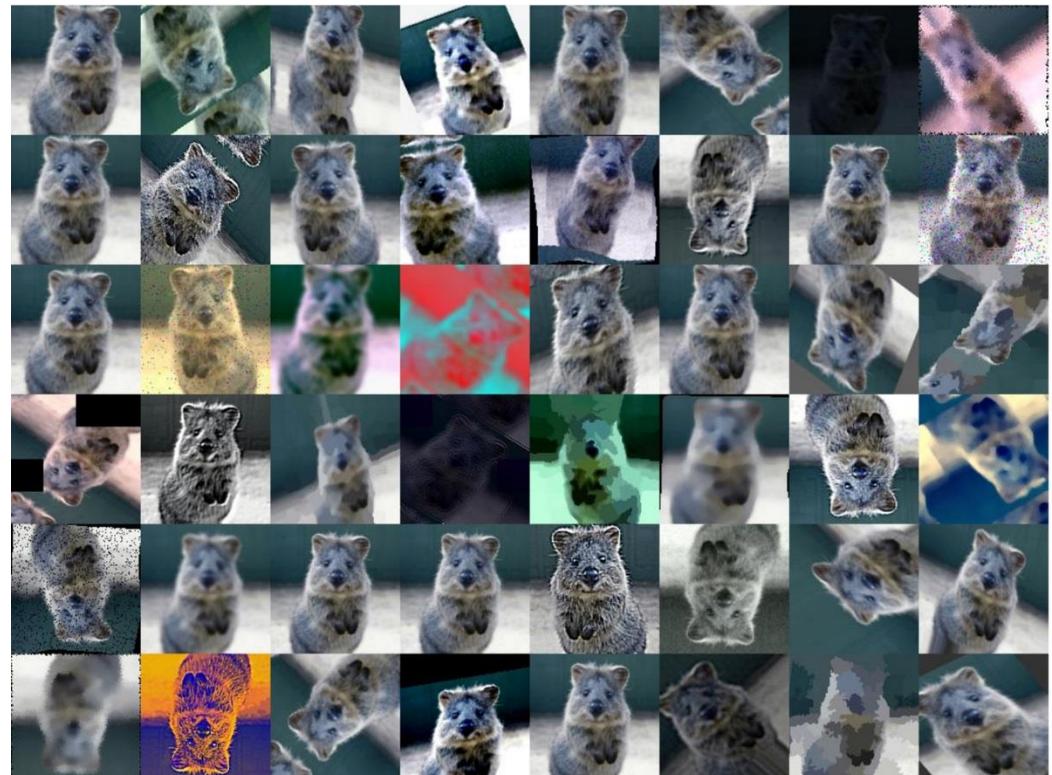
Dropout: A simple way to prevent neural networks from overfitting [[Srivastava JMLR 2014](#)]

# Data Augmentation (Jittering)

---

Create *virtual* training samples

- Horizontal flip
- Random crop
- Color casting
- Geometric distortion



---

# Transfer Learning

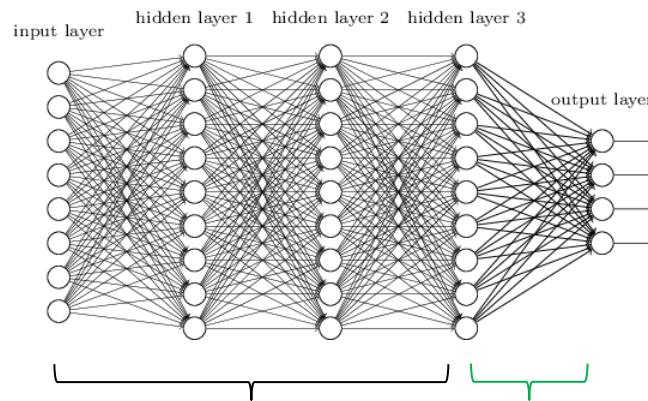
“You need a lot of data if you want to  
train/ use CNNs”

BUSTED

# Transfer Learning with CNNs

---

- The more weights you need to learn, the more data you need
- That's why with a deeper network, you need more data to train than for a shallower net
- One possible solution:



Set these to the already learned  
weights from another network

Learn these on your own task

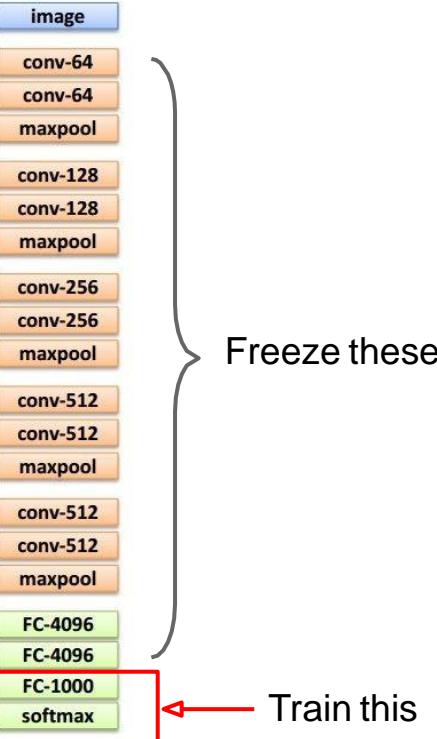
# Transfer Learning with CNNs

Source: classification on ImageNet

1. Train on  
ImageNet



2. Small dataset:



Freeze these

Train this

3. Medium dataset:  
**finetuning**

more data = retrain more of  
the network (or all of it)

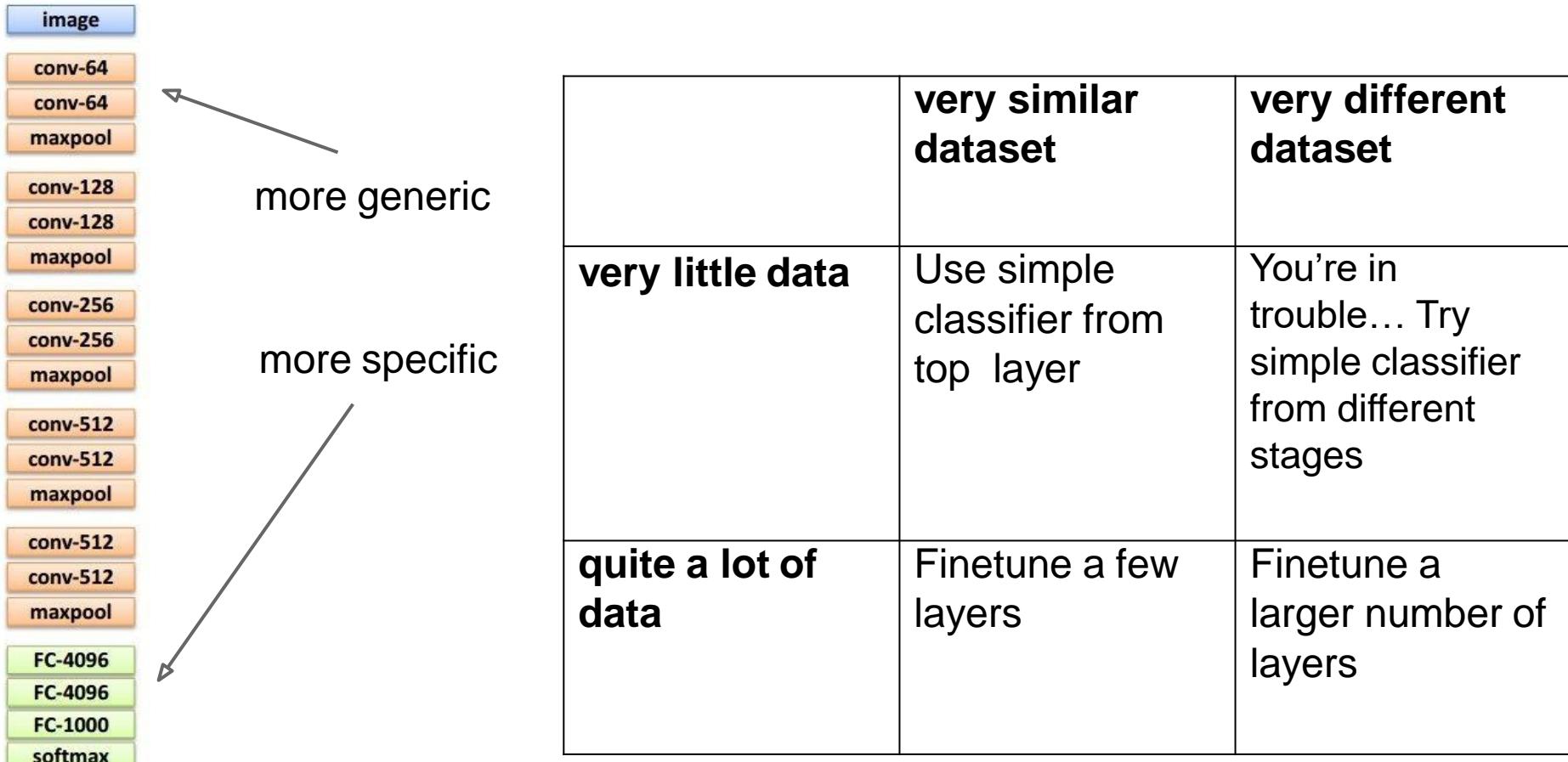
Freeze these



Train this

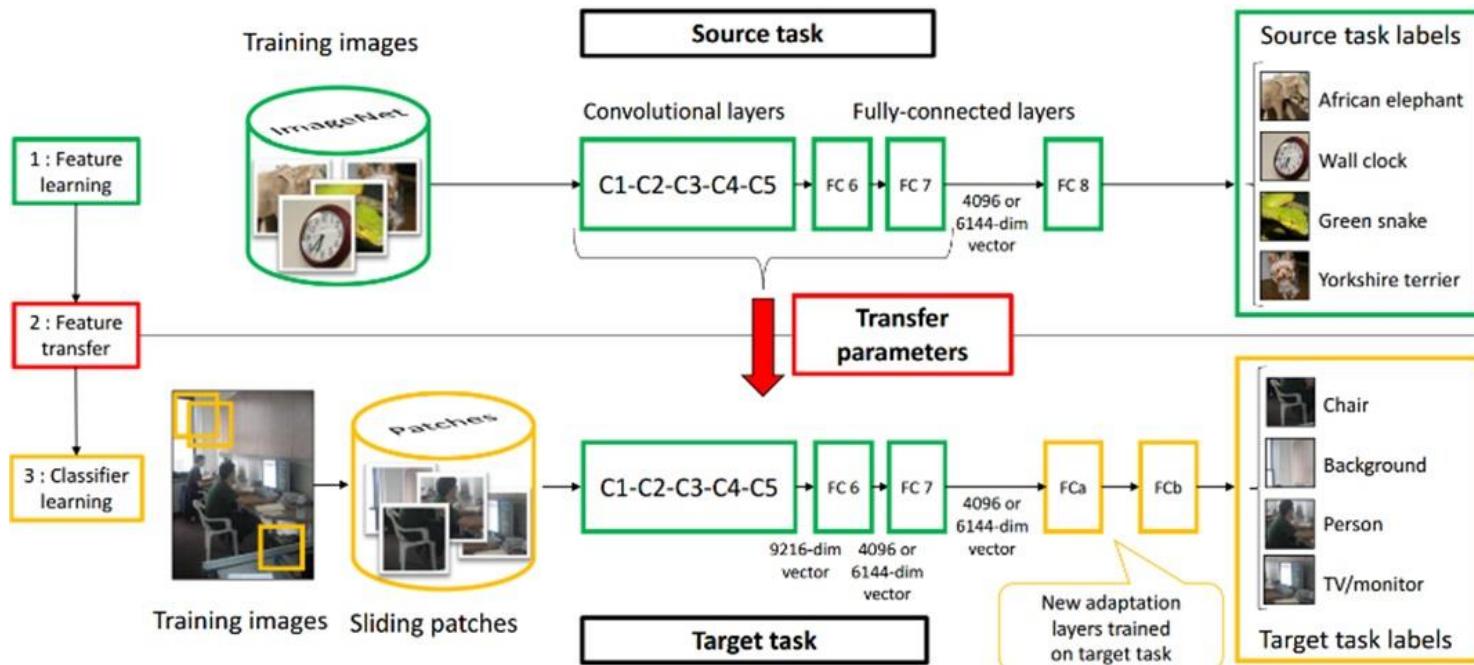
Another option: use network as feature extractor,  
train SVM on extracted features for target task

# Transfer Learning with CNNs



# Pre-training on ImageNet

- Have a source domain and target domain
- Train a network to classify ImageNet classes
  - Coarse classes and ones with fine distinctions (dog breeds)
- Remove last layers and train layers to replace them, that predict target classes



# Transfer learning with CNNs is pervasive...

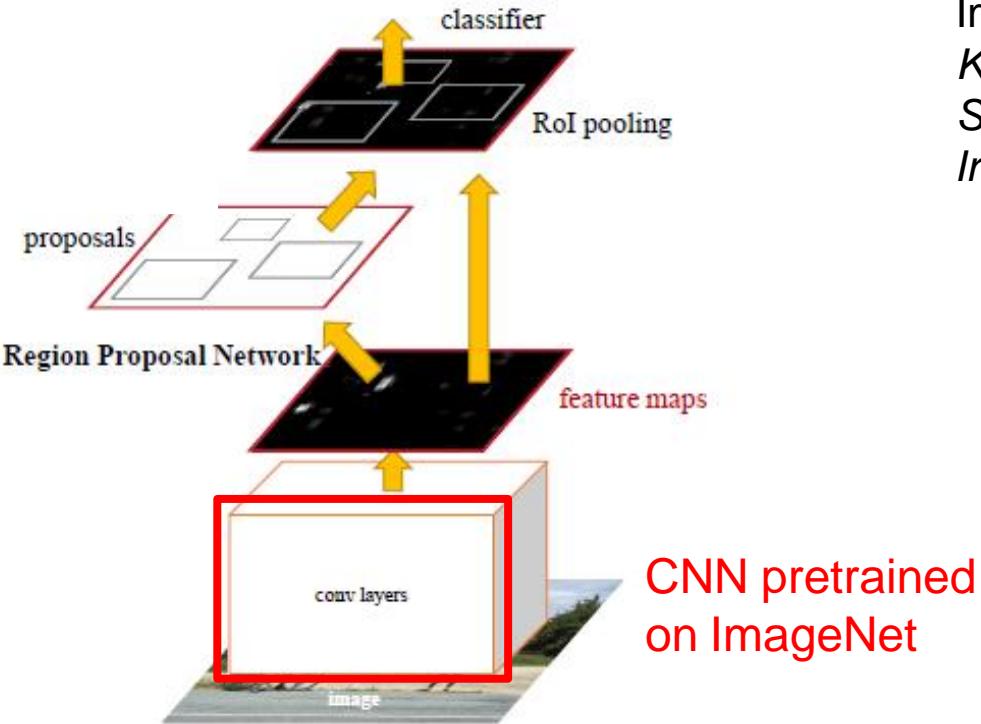
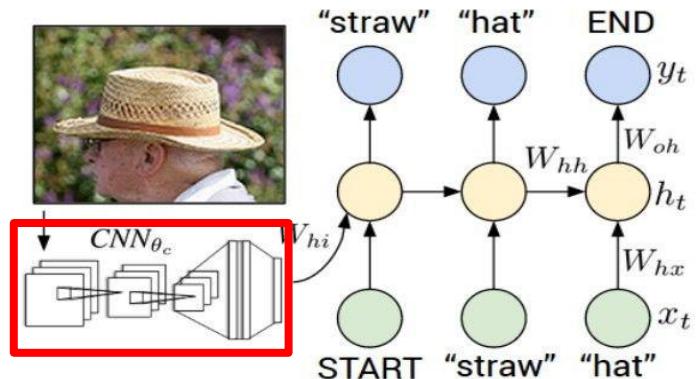


Image Captioning

*Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015*

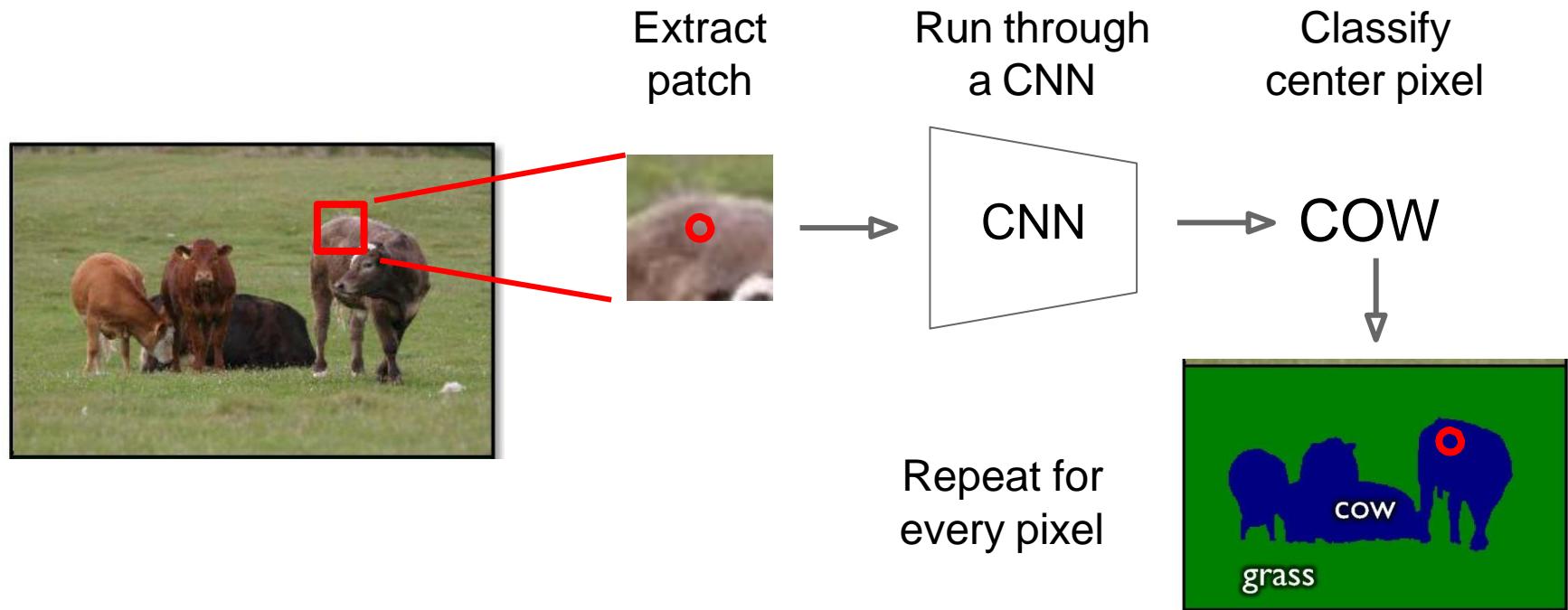


Object Detection

*Ren et al., "Faster R-CNN", NIPS 2015*

# Semantic segmentation

---



# Photographer identification

---

Who took this photograph?



- Deep net features achieve 74% accuracy
  - Chance is less than 3%, human performance is 47%
- Method learns what proto-objects + scenes authors shoot

# Analysis of pre-training on ImageNet

---

- Source:
  - distinguish 1000 ImageNet categories (incl. many dog breeds)
- Target tasks:
  - object detection and action recognition on PASCAL
  - scene recognition on SUN
- Pre-training with 500 images per class is about as good as pre-training with 1000
- Pre-training with 127 classes is about as good as pre-training with 1000
- Pre-training with (fewer classes, more images per class) > (more classes, fewer images)
- Small drop in if classes with fine-grained distinctions removed from pre-training set

# Packages

---

[TensorFlow](#)

[Torch / PyTorch](#)

[Keras](#)

[Caffe](#) and [Caffe Model Zoo](#)

# Some Learning Resources

---

<http://deeplearning.net/>

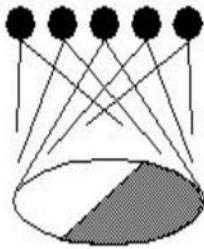
<http://cs231n.stanford.edu>

# Understanding CNNs

# Recall: Biological analog

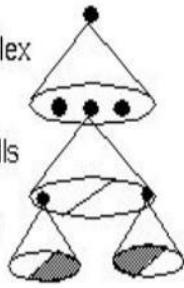
Hubel & Weisel

topographical mapping



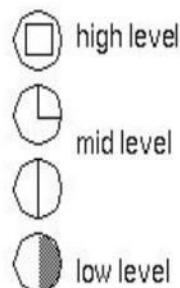
featural hierarchy

hyper-complex  
cells

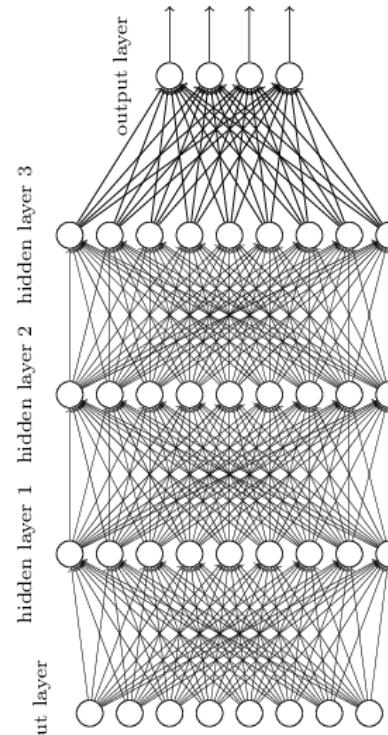
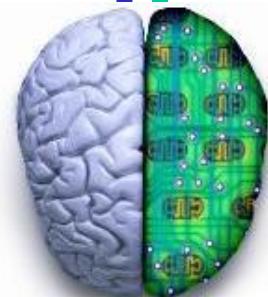


complex cells

simple cells

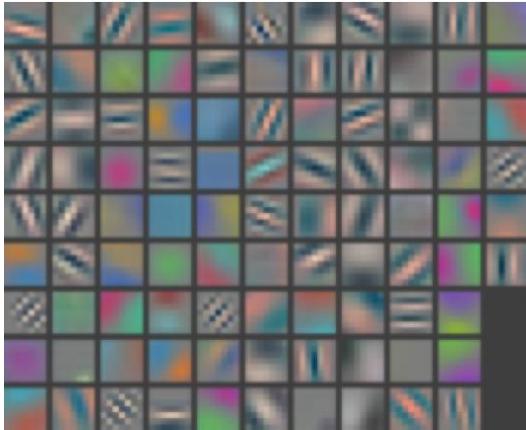


Hubel and Weisel's architecture



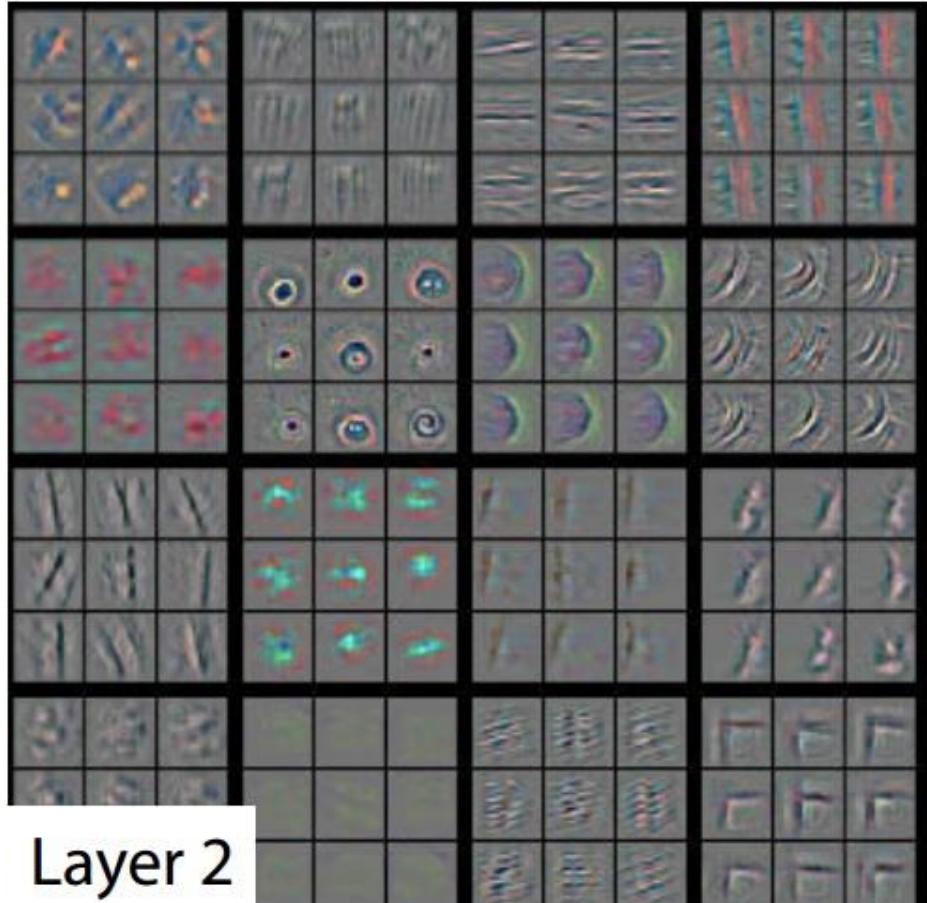
Multi-layer neural network

# Layer 1



# Layer 2

---



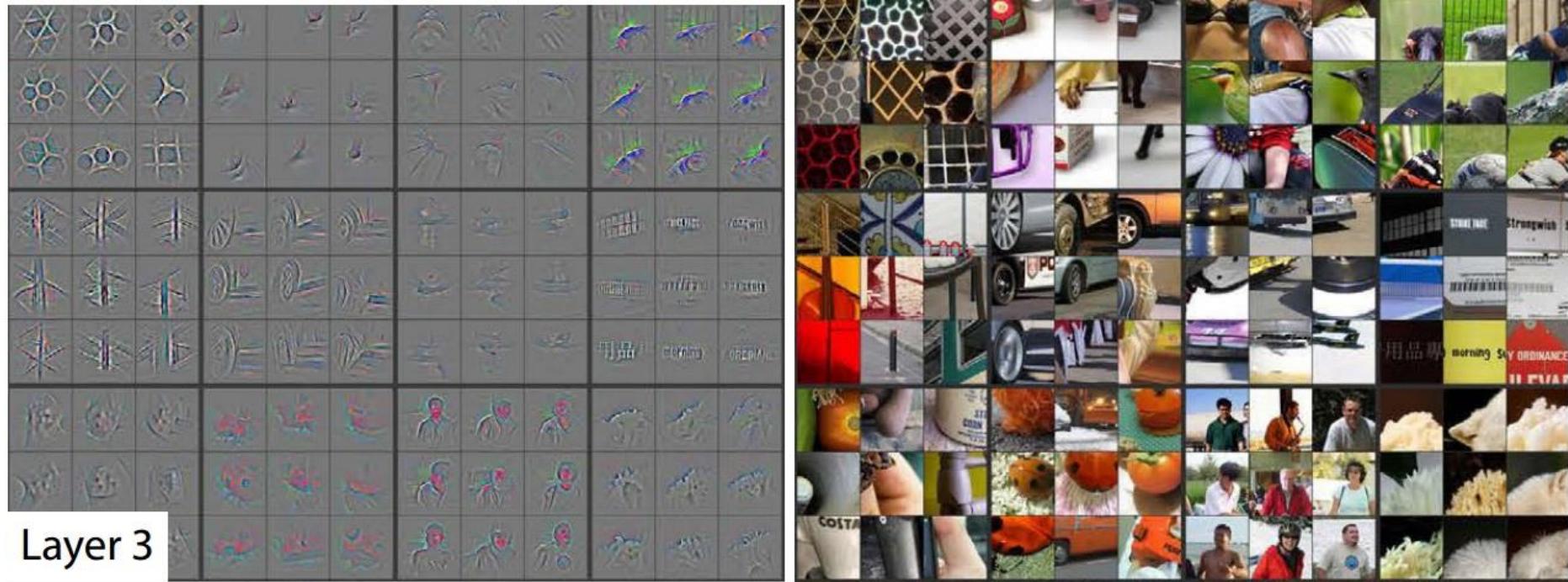
Layer 2



- Activations projected down to pixel level via deconvolution
- Patches from validation images that give maximal activation of a given feature map

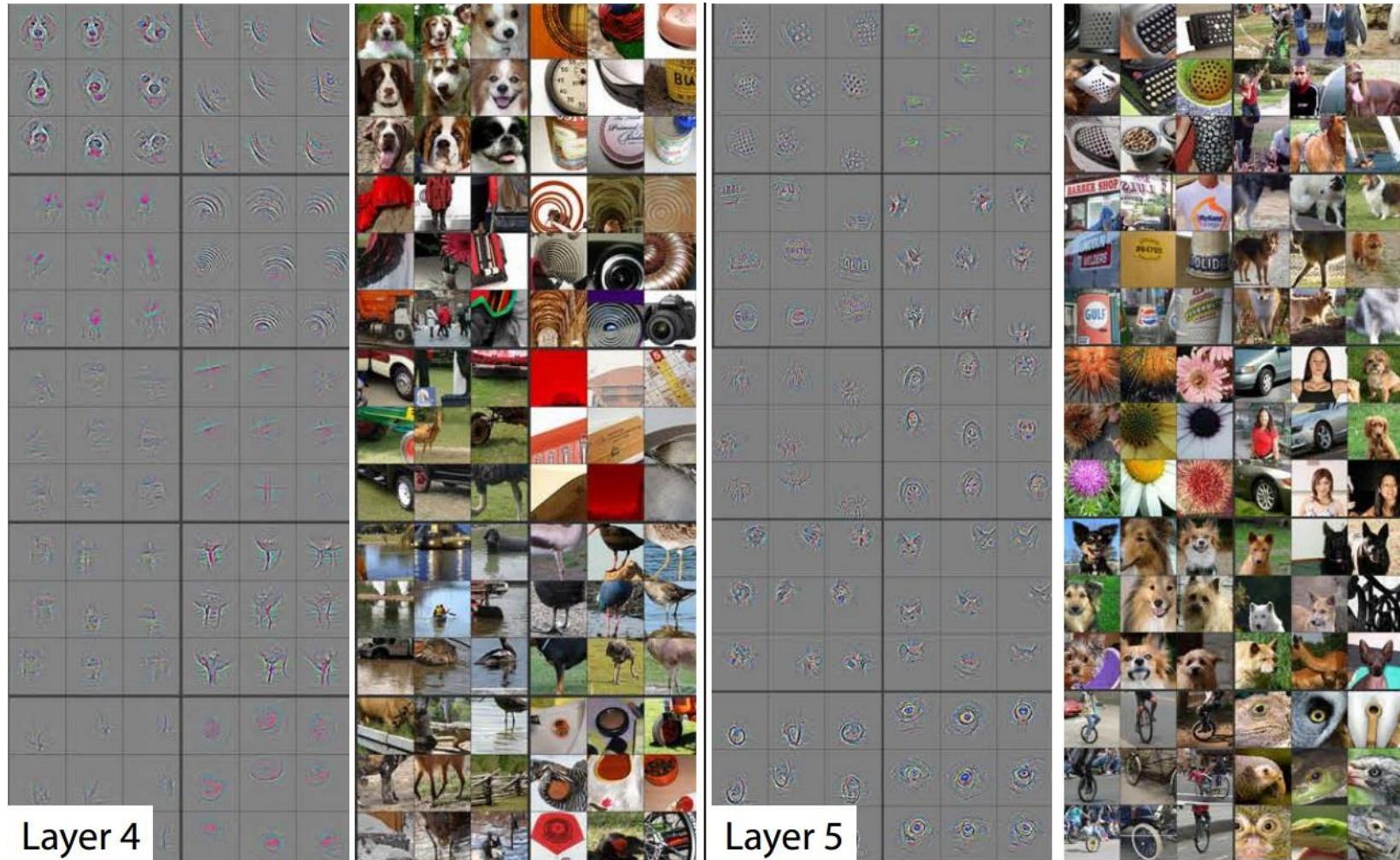
# Layer 3

---



# Layer 4 and 5

---

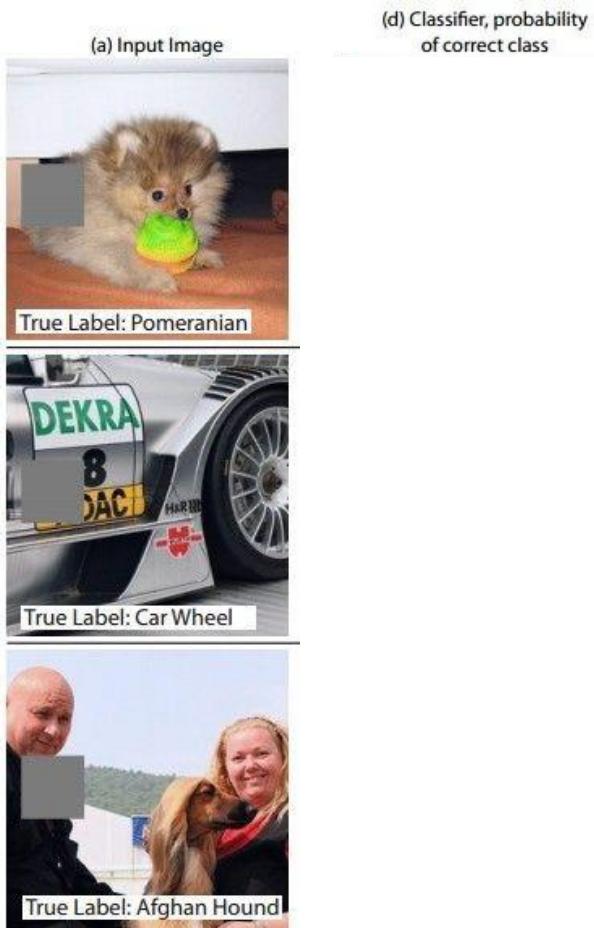


Layer 4

Layer 5

# Occlusion experiments

---



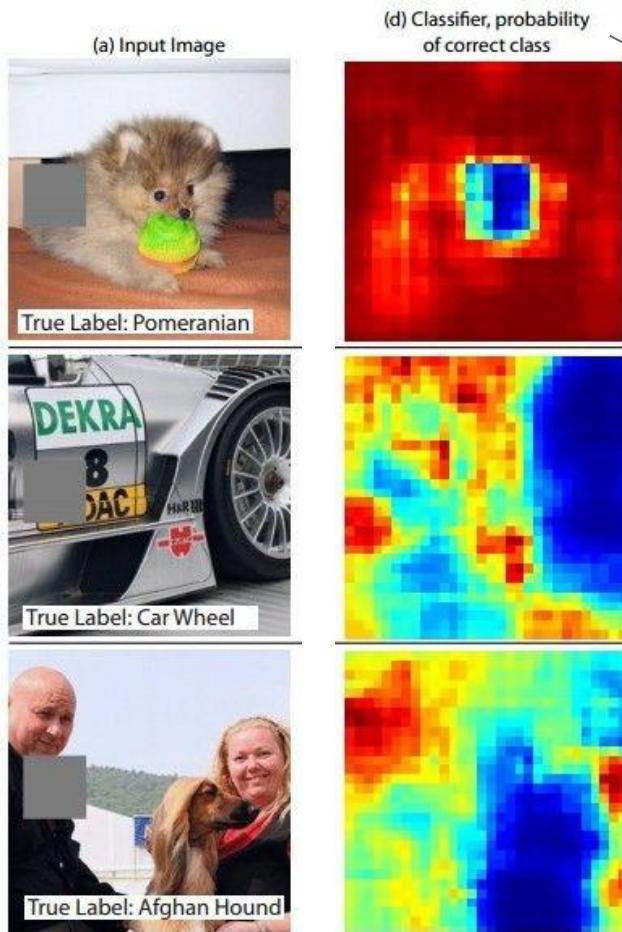
(d) Classifier, probability  
of correct class

(as a function of the  
position of the  
square of zeros in  
the original image)

[Zeiler & Fergus 2014]

# Occlusion experiments

---

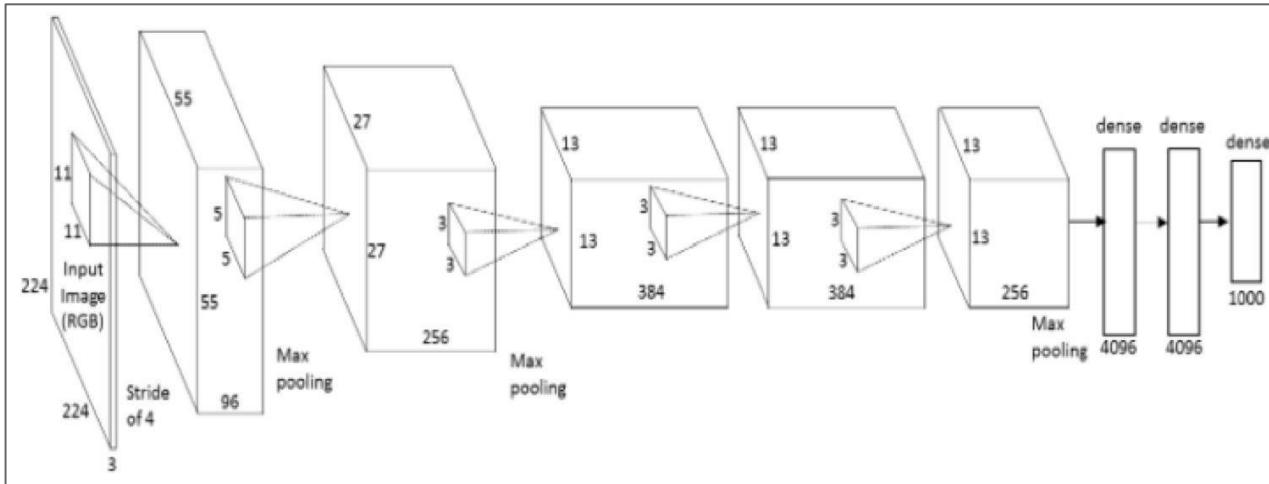


(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]

# What image maximizes a class score?

---



**Repeat:**

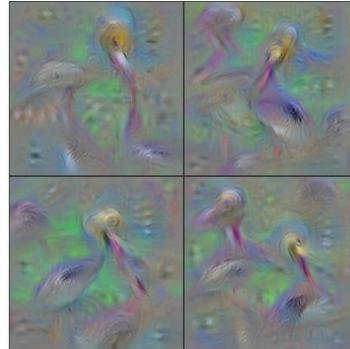
1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an “image update”

# What image maximizes a class score?

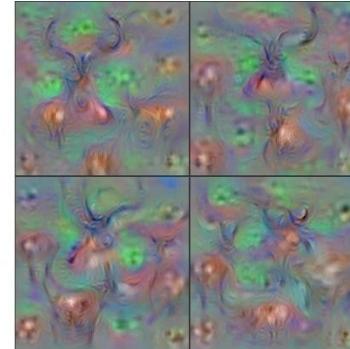
---



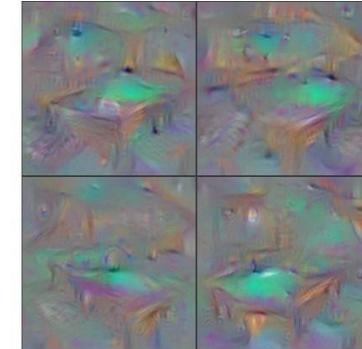
Flamingo



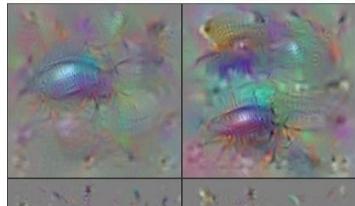
Pelican



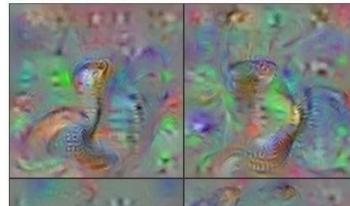
Hartebeest



Billiard Table



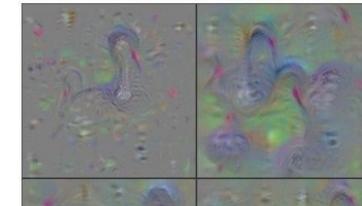
Ground Beetle



Indian Cobra



Station Wagon



Black Swan

[*Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015*]

<http://yosinski.com/deepvis>

# What image maximizes a class score?

---



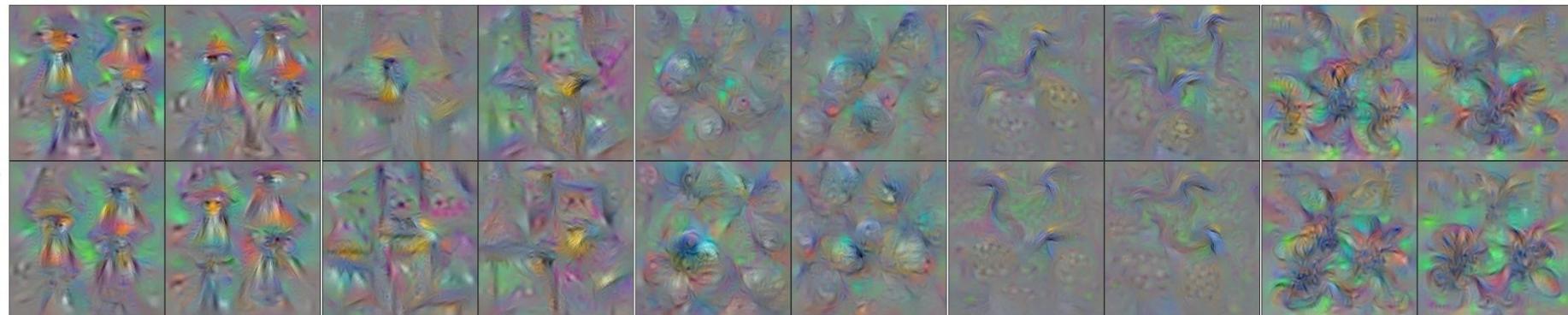
Pirate Ship

Rocking Chair

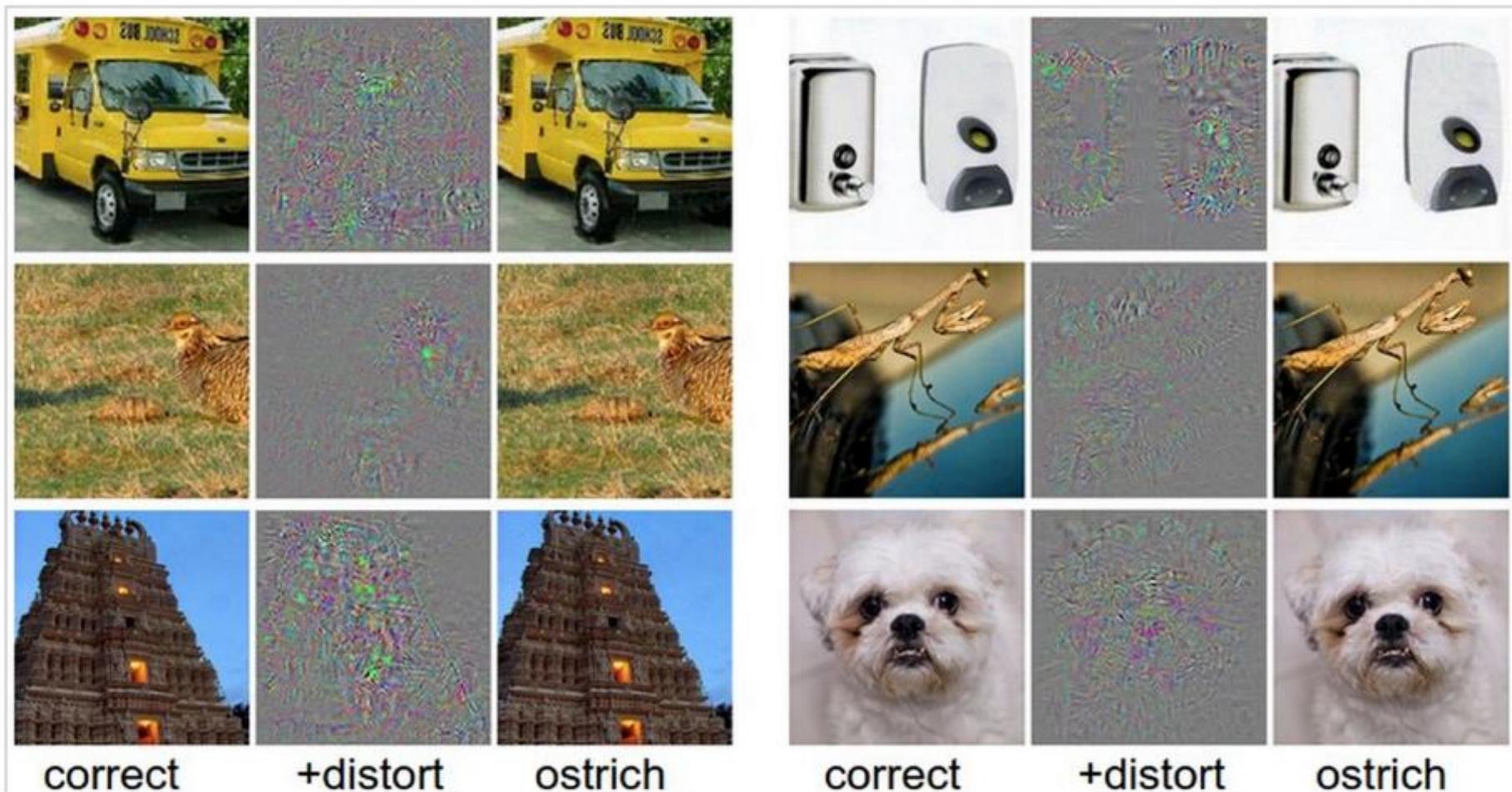
Teddy Bear

Windsor Tie

Pitcher



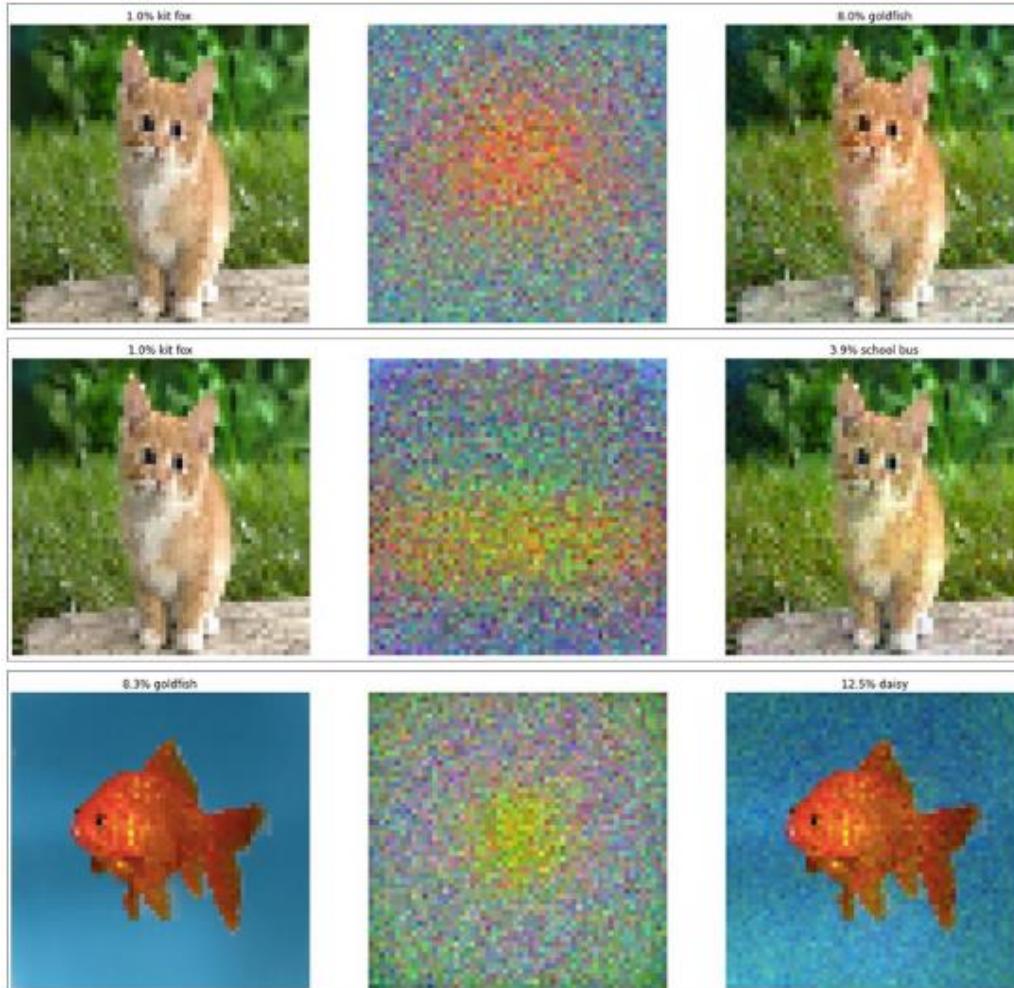
# Breaking CNNs



Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

Intriguing properties of neural networks [[Szegedy ICLR 2014](#)]

# Fooling a linear classifier



Fooled linear classifier: The starting image (left) is classified as a kit fox. That's incorrect, but then what can you expect from a linear classifier? However, if we add a small amount 'goldfish' weights to the image (top row, middle), suddenly the classifier is convinced that it's looking at one with high confidence. We can distort it with the school bus template instead if we wanted to.

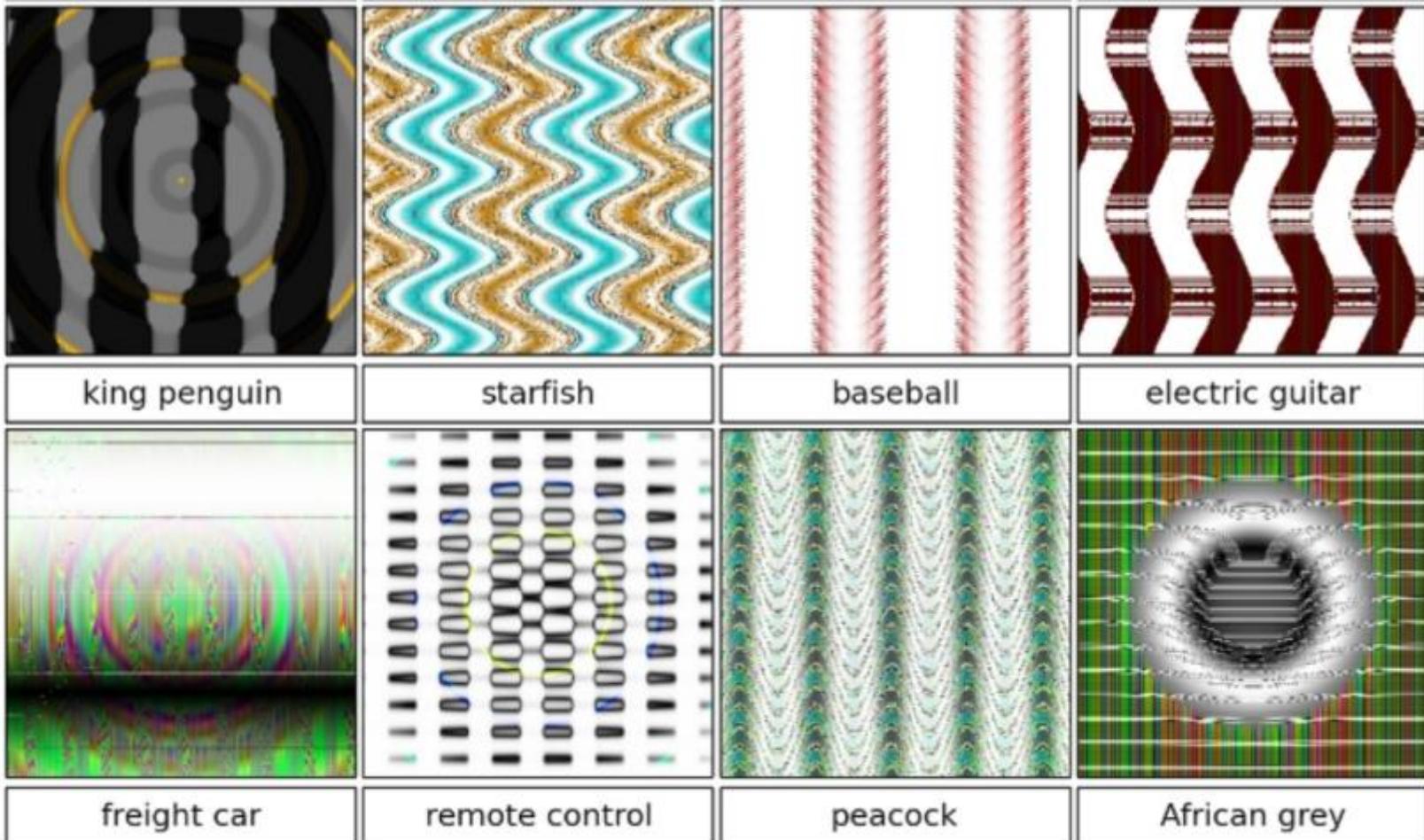
How to fool?

Add a small multiple of the weight vector to the training example:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{w}$$

# Breaking CNNs

---



Deep Neural Networks are Easily Fooled: High Confidence Predictions for  
Unrecognizable Images [[Nguyen et al. CVPR 2015](#)]

# Summary

---

- We use deep neural networks because of their strong performance in practice
- Convolutional neural network (CNN)
  - Special operations: Convolution, max pooling
- Training deep neural nets
  - We need an objective function that measures and guides us towards good performance
  - We need a way to minimize the loss function: stochastic gradient descent
  - We need backpropagation to propagate error from end of net towards all layers and change weights at those layers
- Practices for preventing overfitting
  - Dropout; data augmentation; transfer learning