

# Lecture 5:

# Convolutional Neural Networks

# Administrative

**Assignment 1** due **Friday April 16**, 11:59pm

- Important: tag your solutions with the corresponding hw question in gradescope!

**Assignment 2** will also be released **April 16th**

# Administrative

Project proposal due **Monday Apr 19**, 11:59pm

This Friday's discussion section will discuss how to design a project

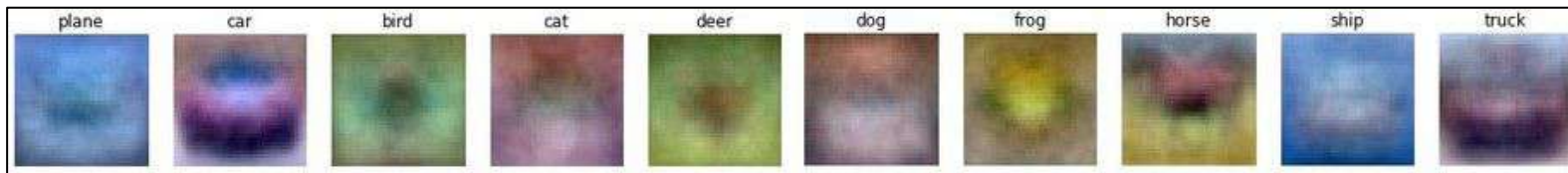
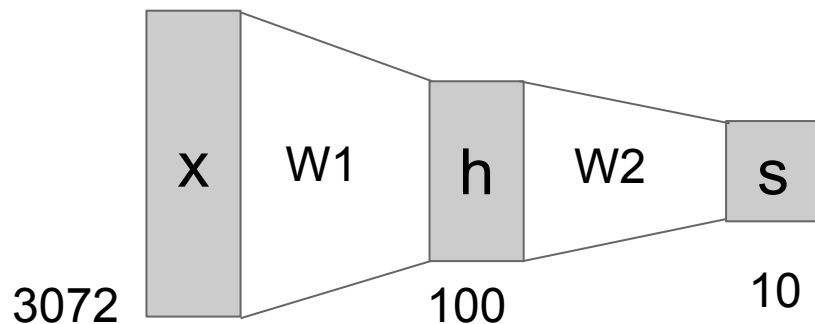
# Last time: Neural Networks

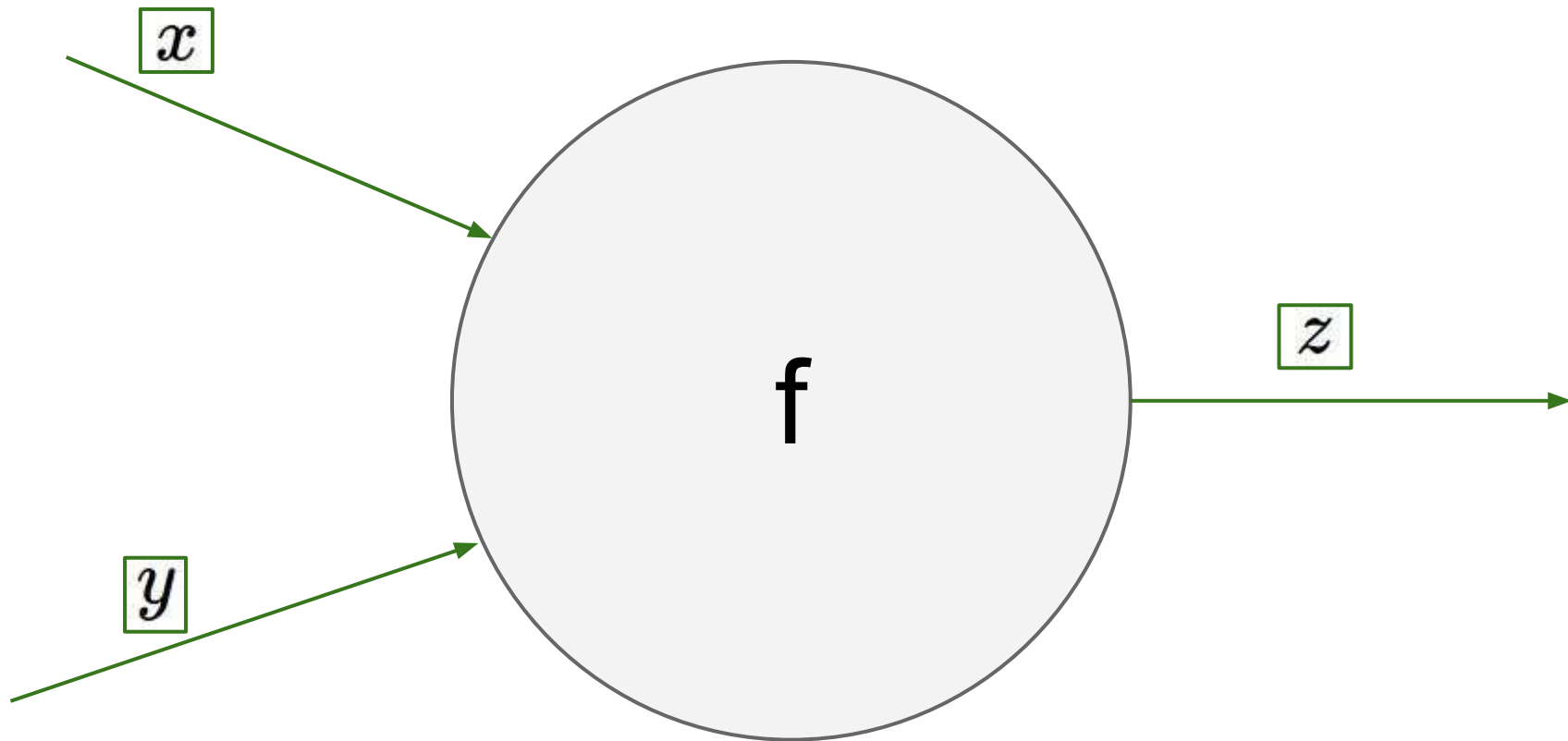
Linear score function:

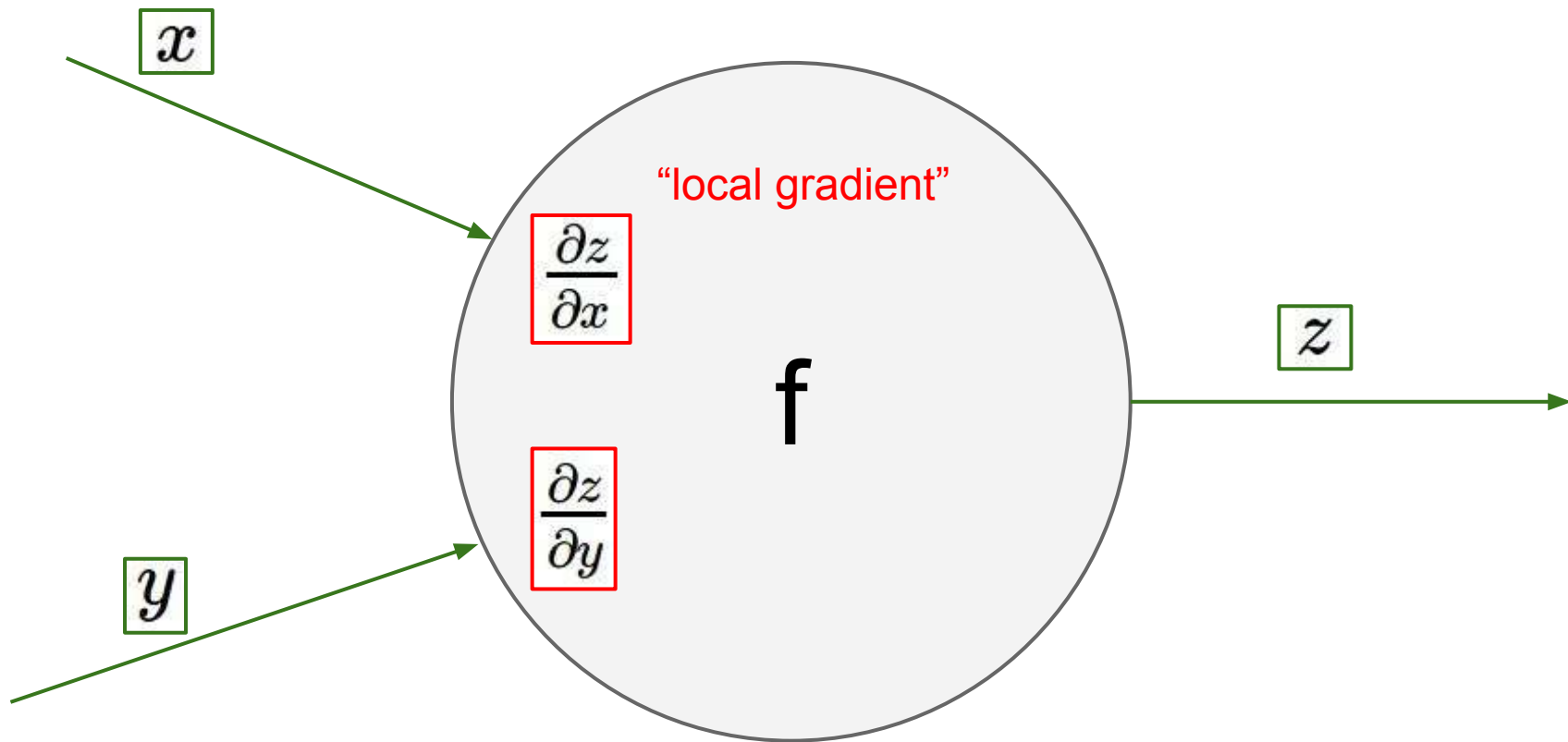
$$f = Wx$$

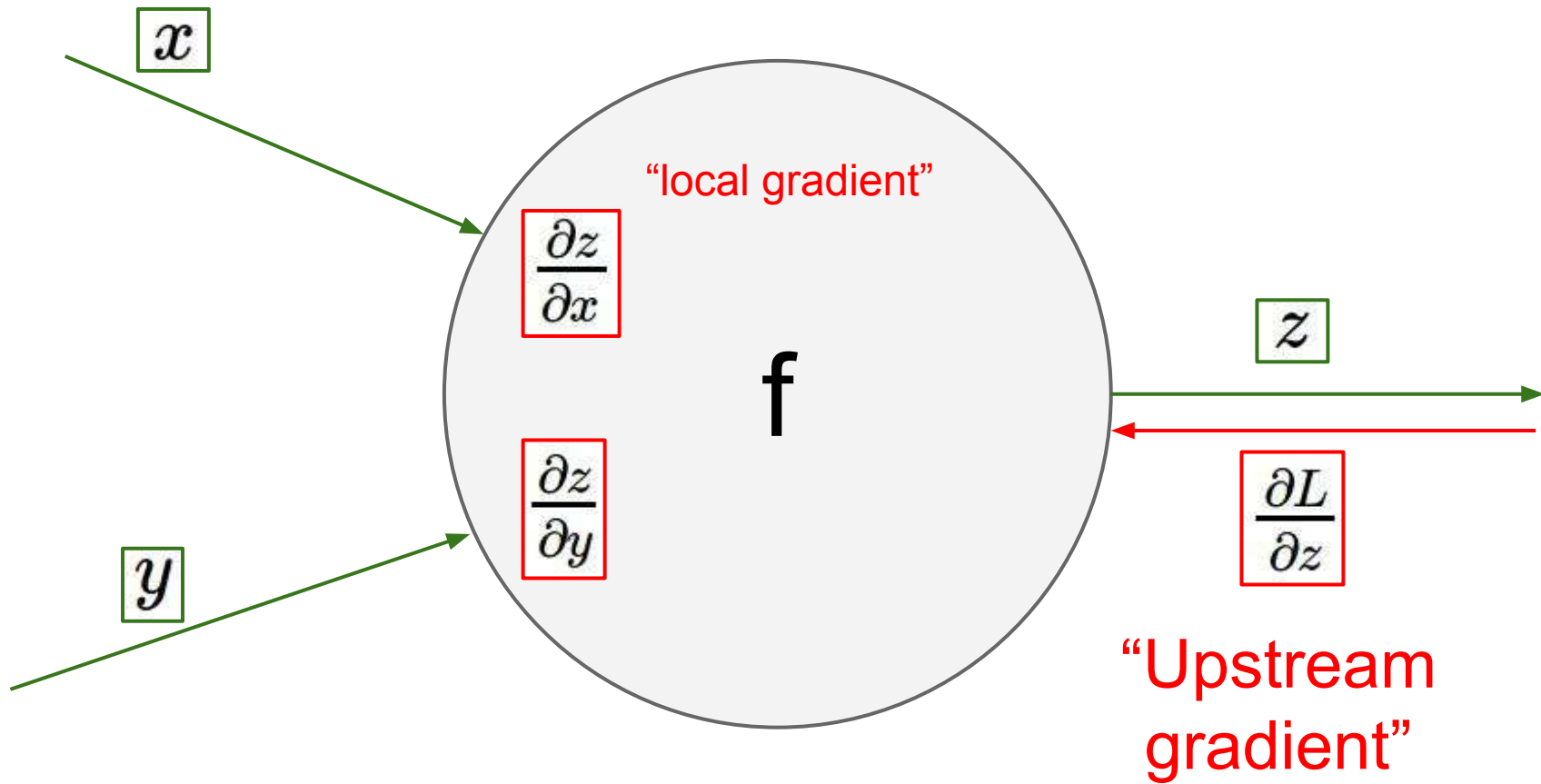
2-layer Neural Network

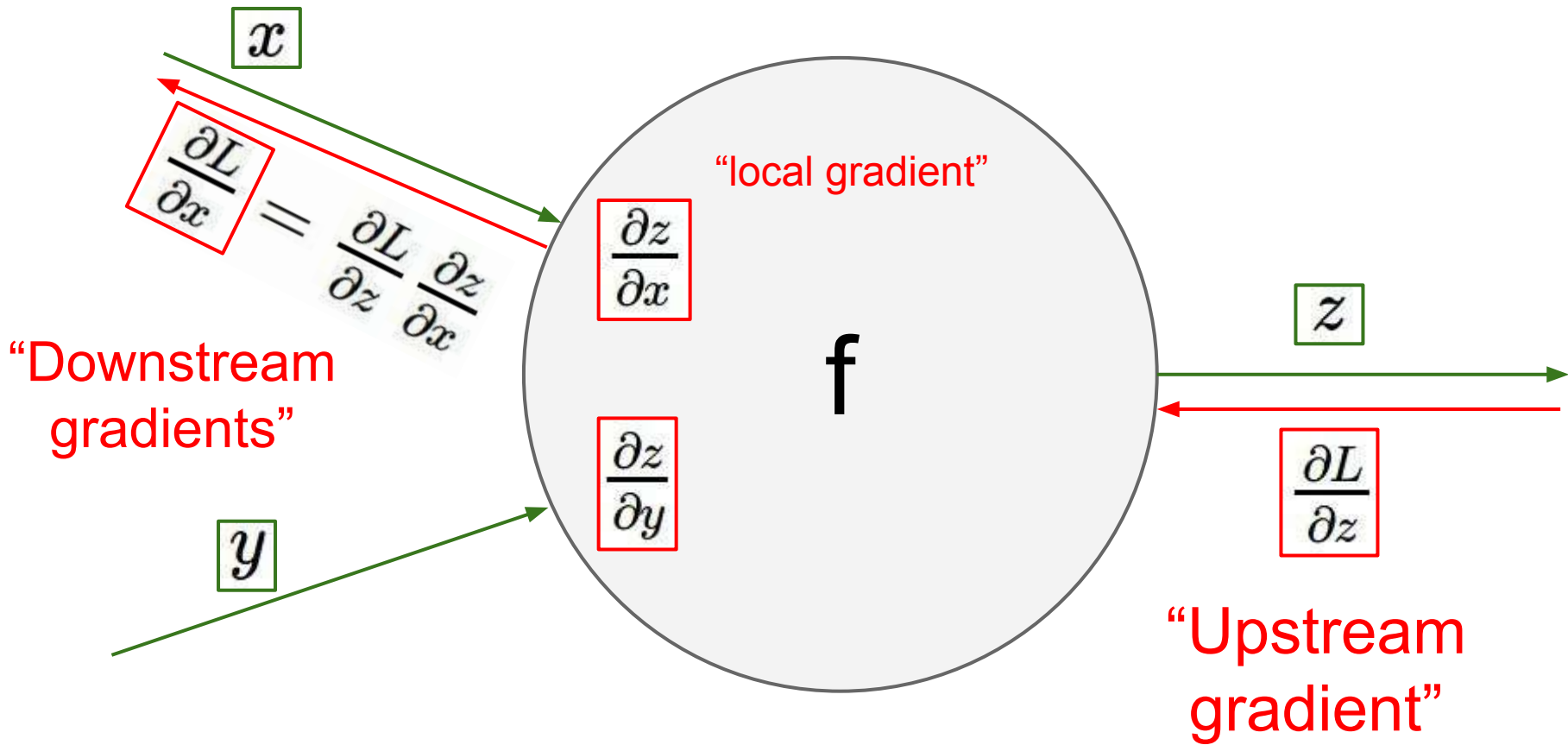
$$f = W_2 \max(0, W_1 x)$$



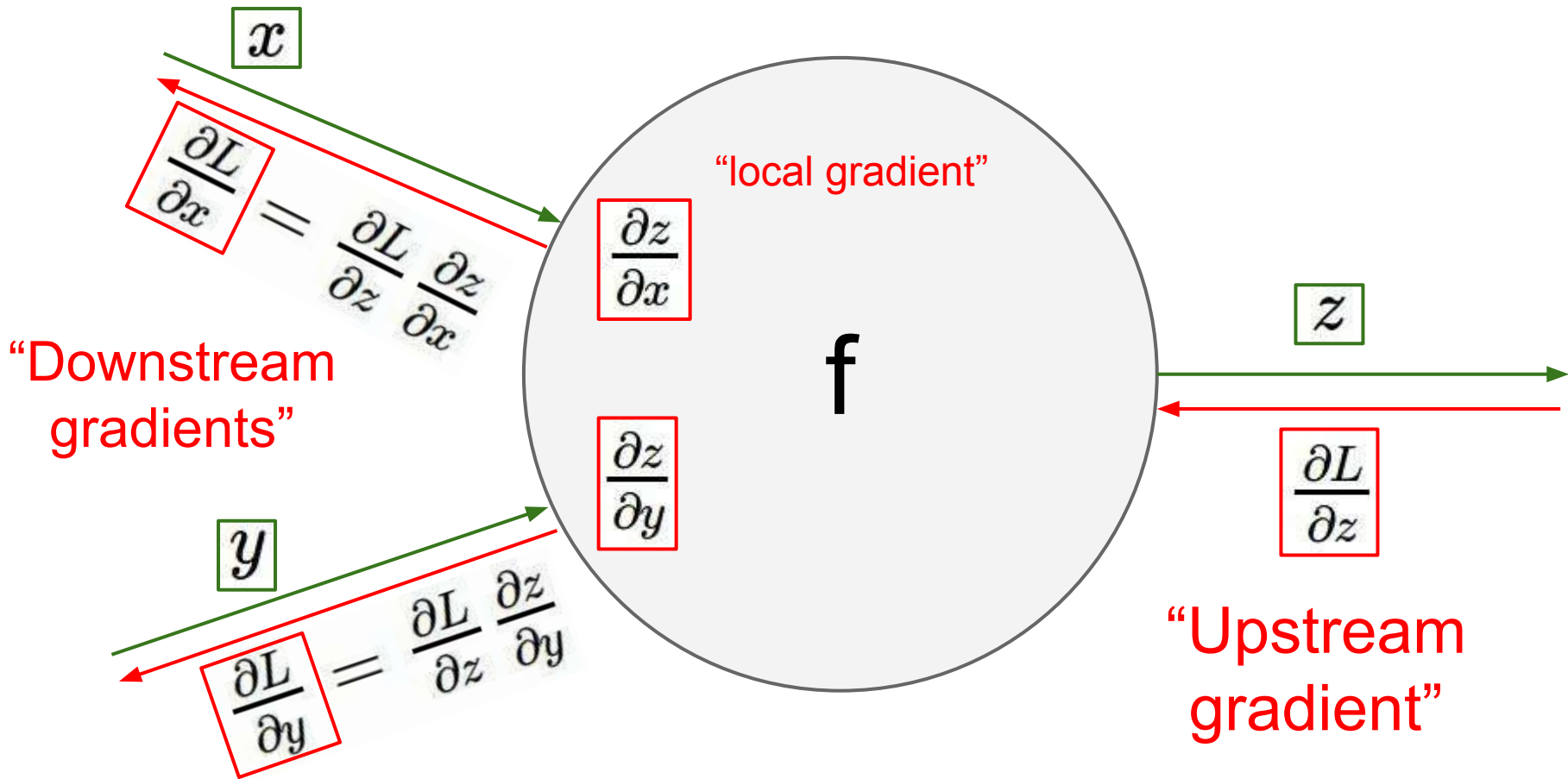


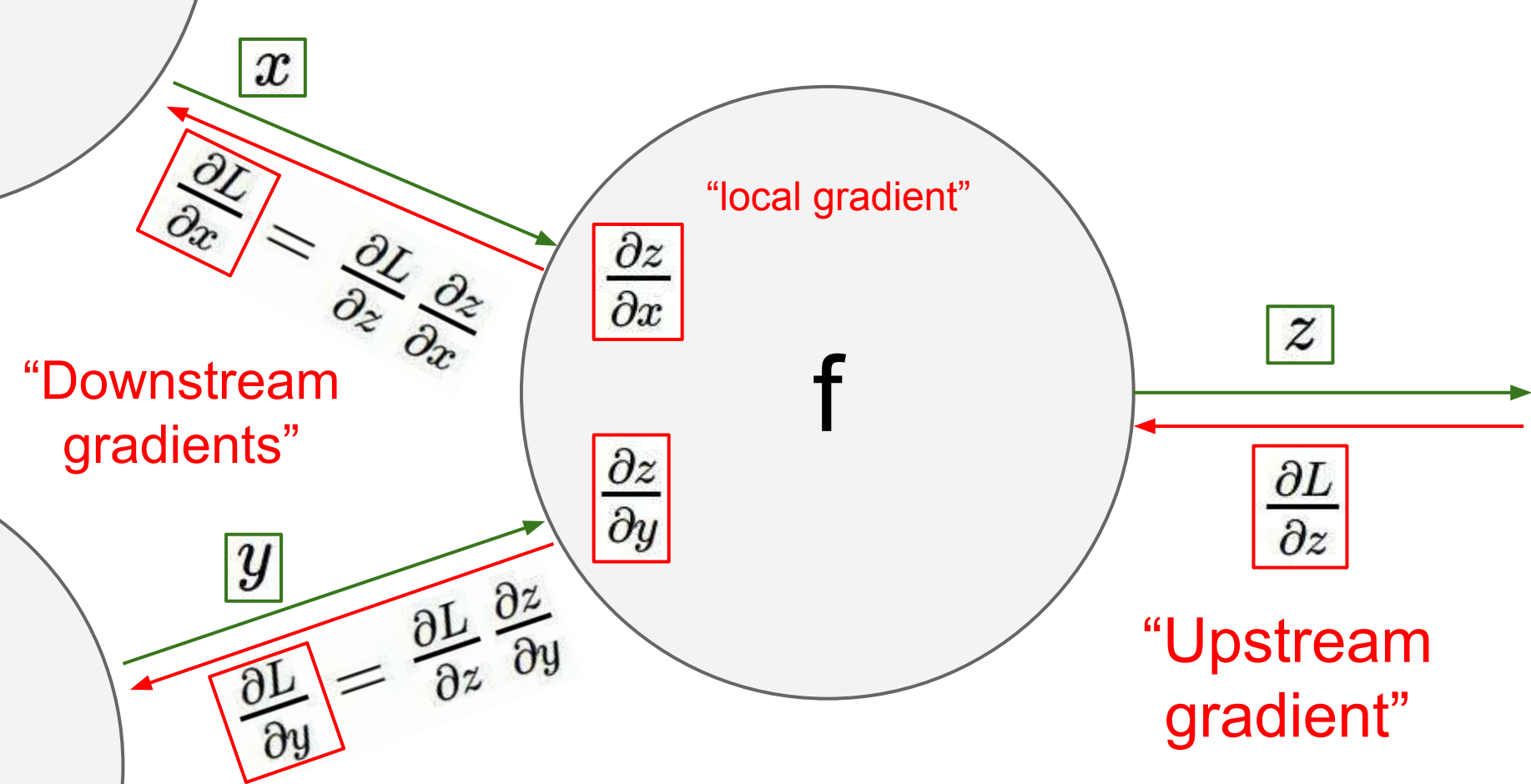












So far: backprop with scalars

What about vector-valued functions?

# Recap: Vector derivatives

## Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If  $x$  changes by a small amount, how much will  $y$  change?

# Recap: Vector derivatives

## Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If  $x$  changes by a small amount, how much will  $y$  change?

## Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will  $y$  change?

# Recap: Vector derivatives

## Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If  $x$  changes by a small amount, how much will  $y$  change?

## Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will  $y$  change?

## Vector to Vector

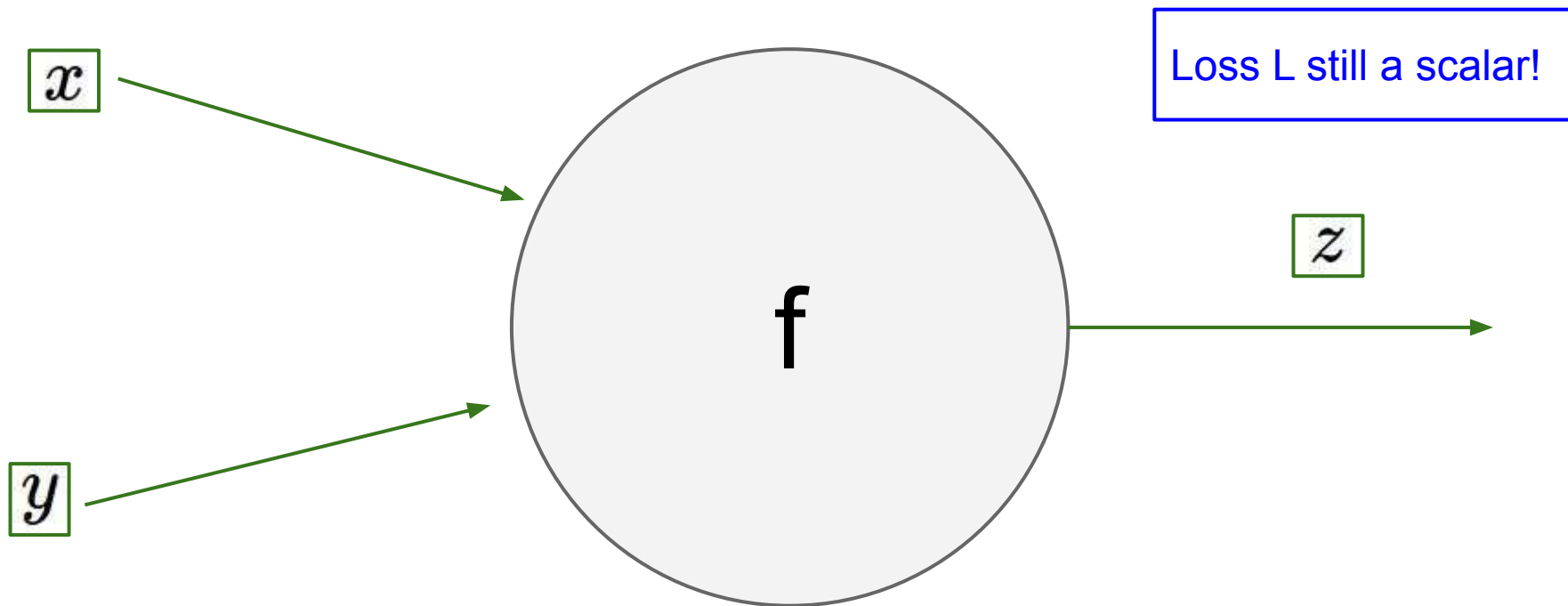
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

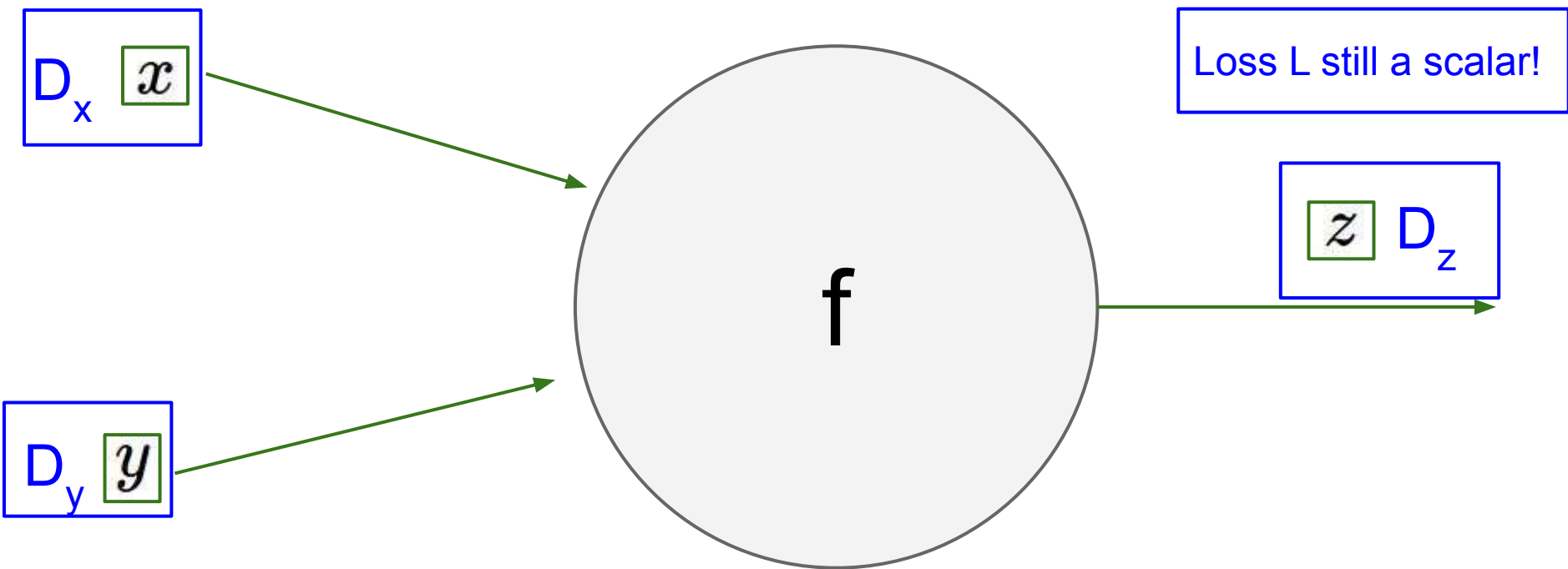
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will each element of  $y$  change?

# Backprop with Vectors

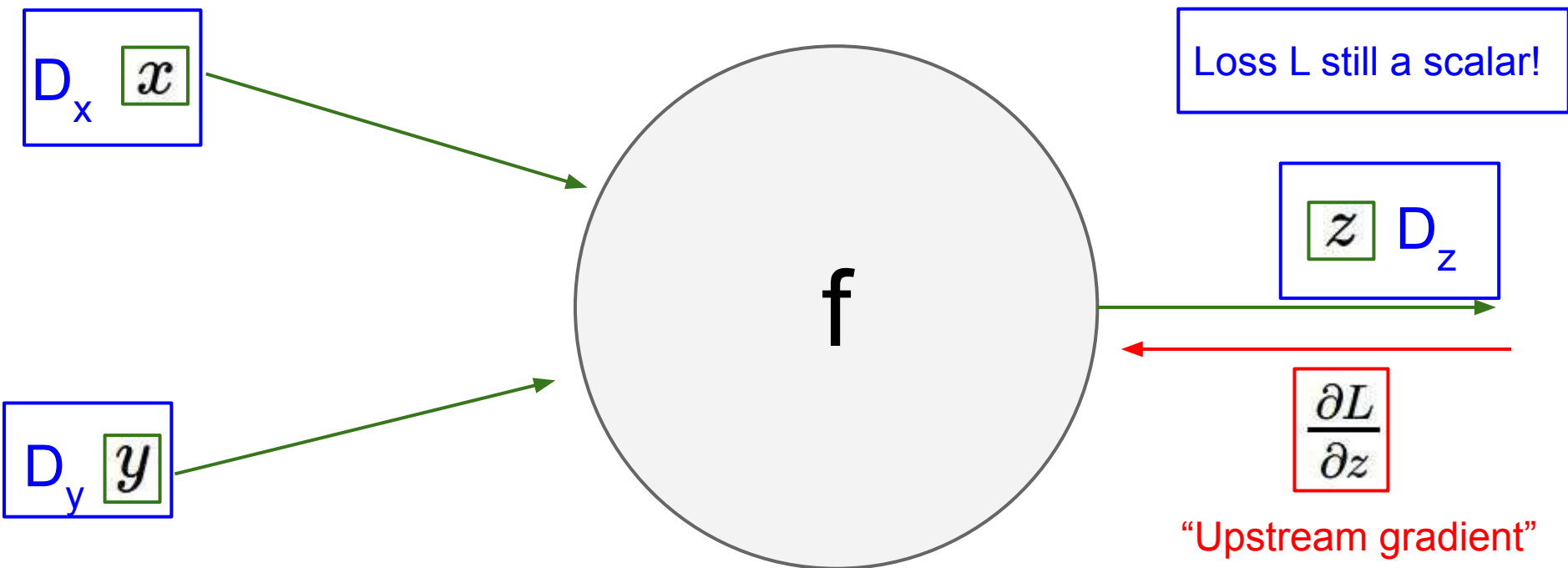


# Backprop with Vectors

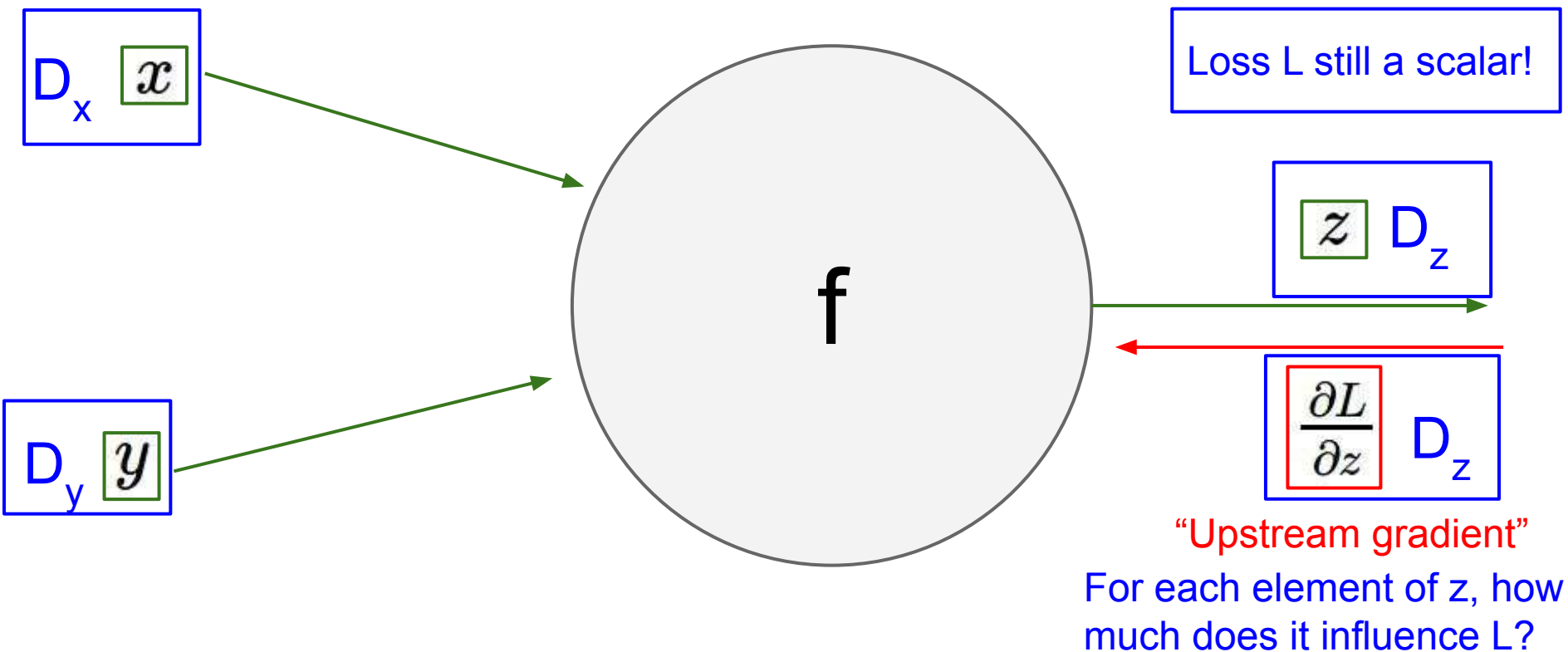




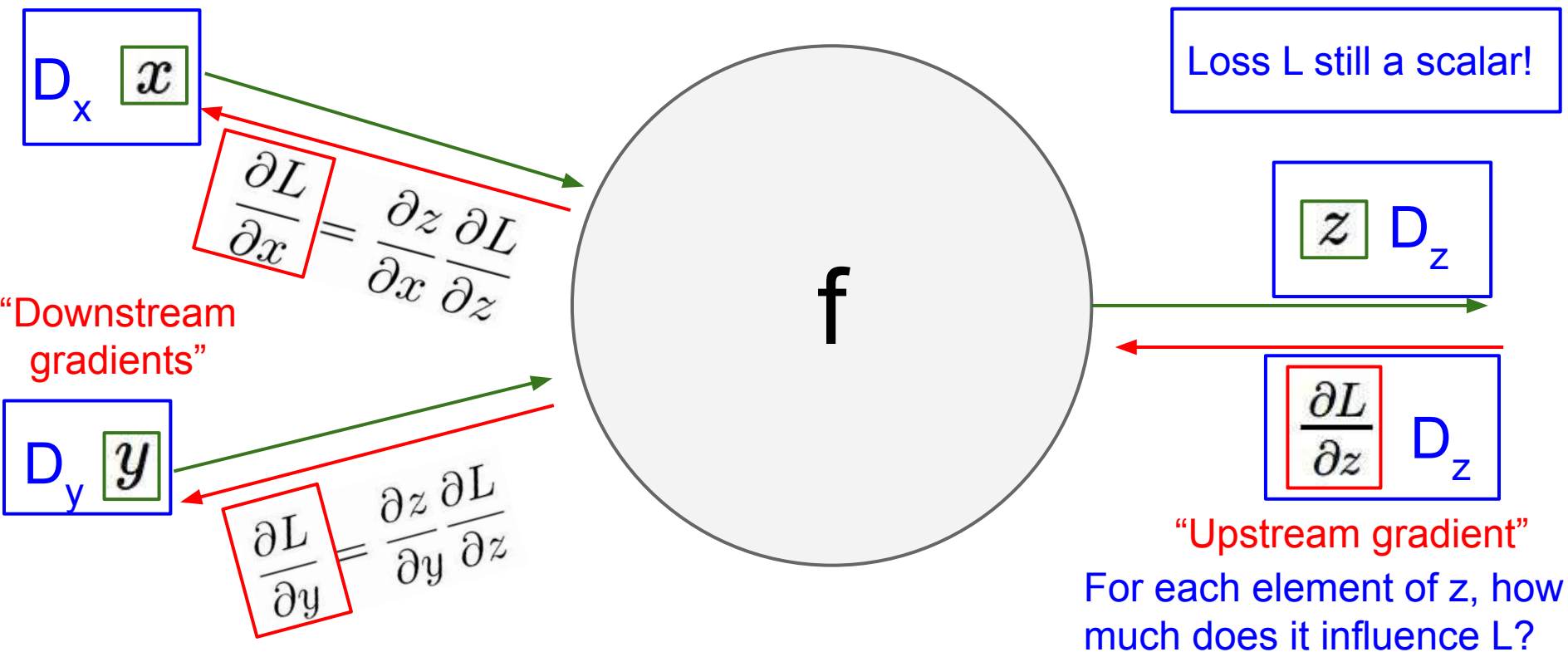
# Backprop with Vectors



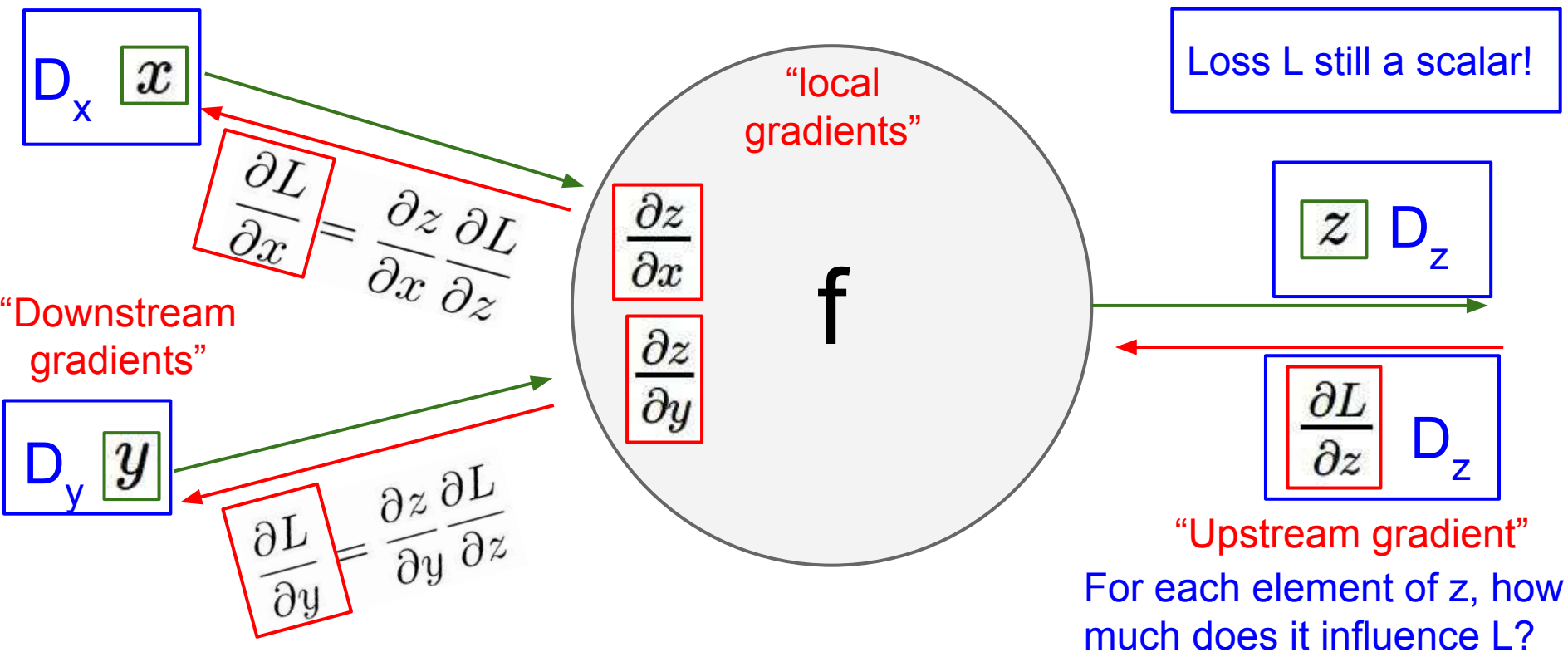
# Backprop with Vectors



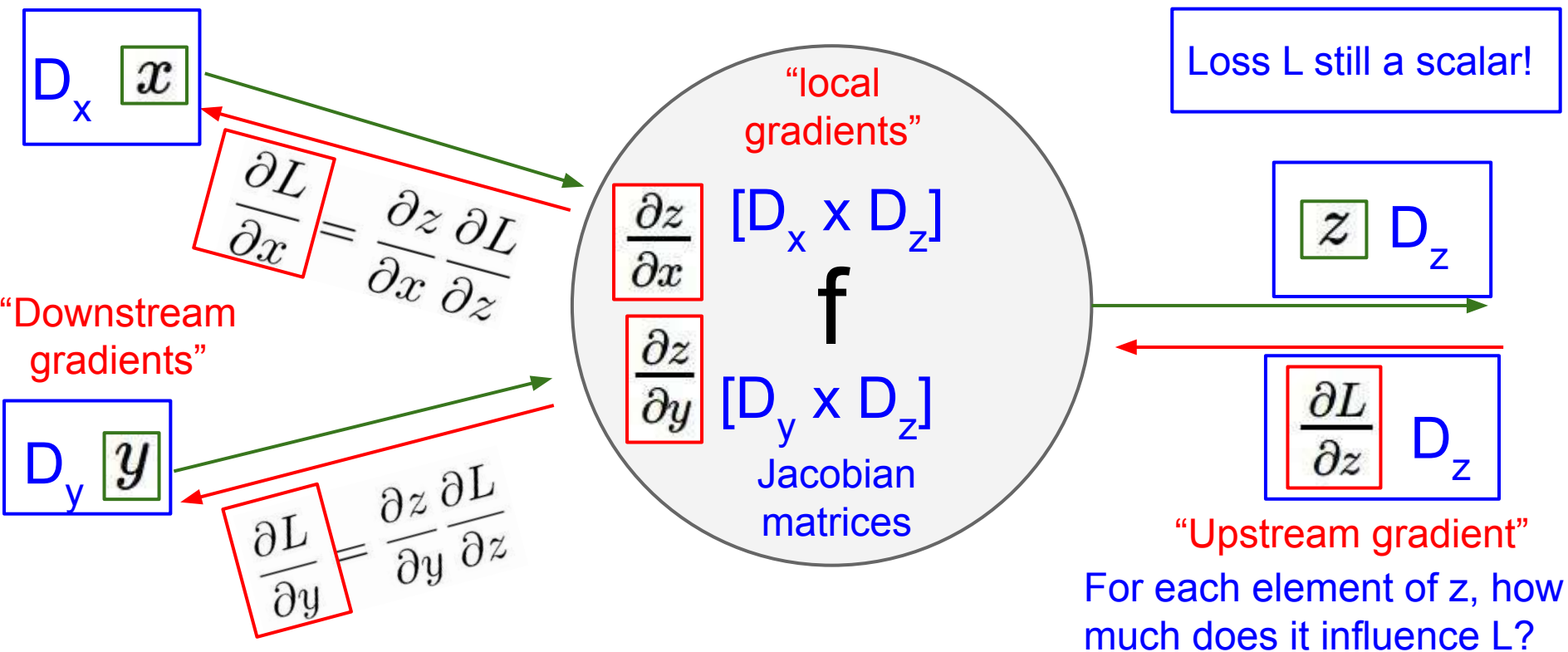
# Backprop with Vectors



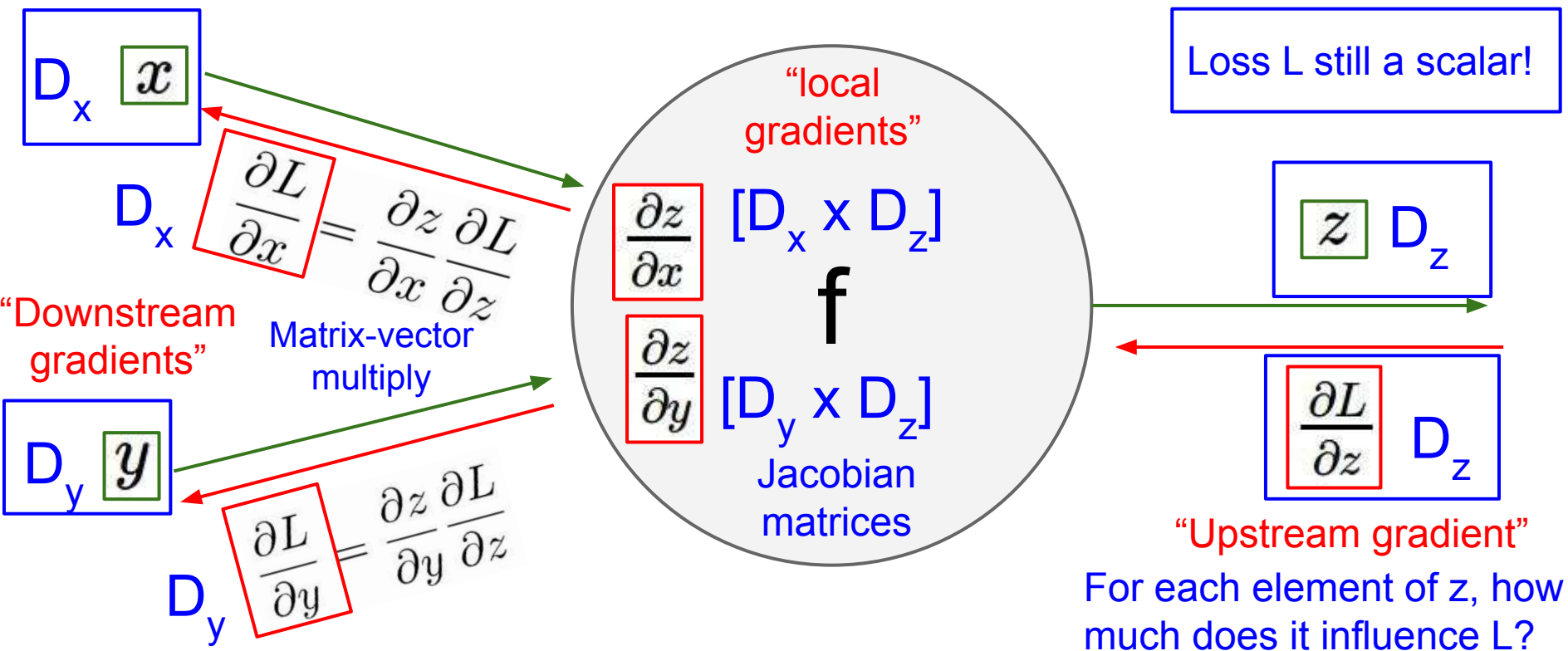
# Backprop with Vectors



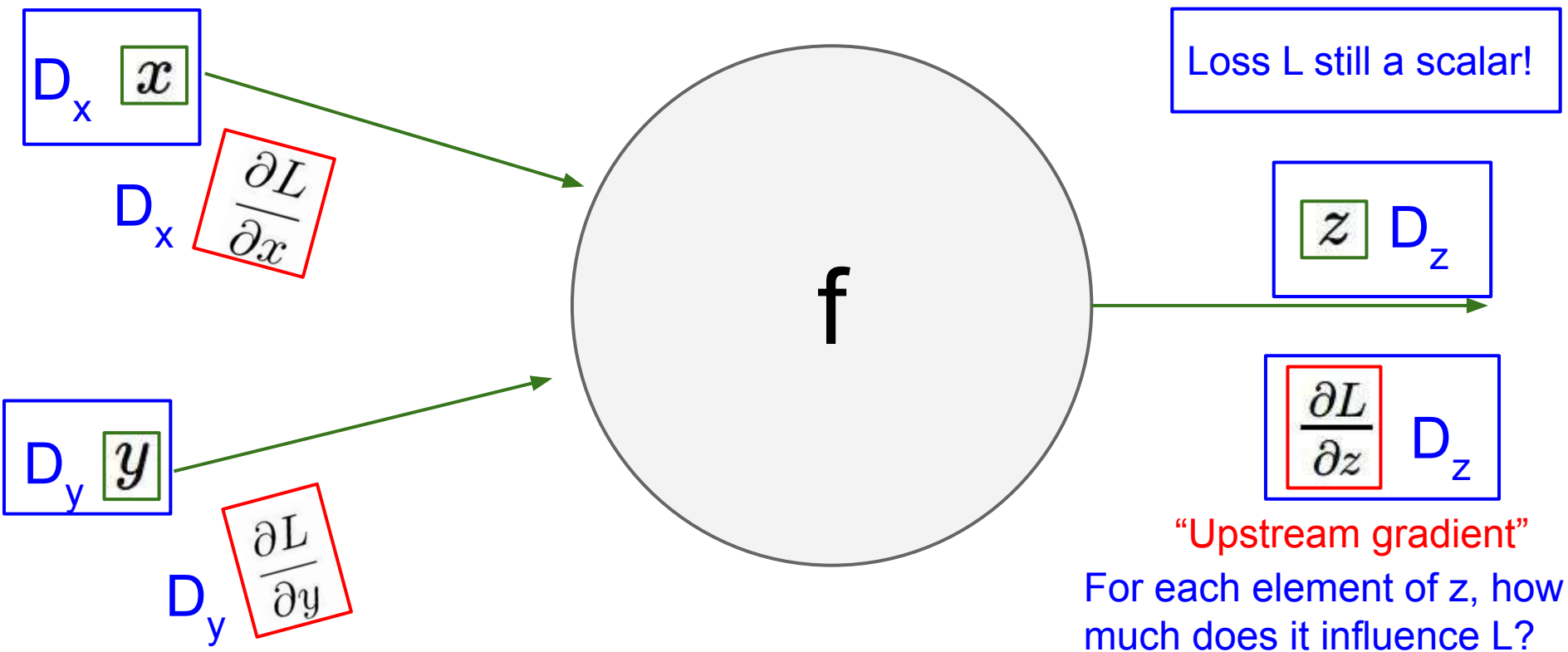
# Backprop with Vectors



# Backprop with Vectors



Gradients of variables wrt loss have same dims as the original variable



## Backprop with Vectors

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

$$f(x) = \max(0, x) \\ (\textit{elementwise})$$

4D output z:

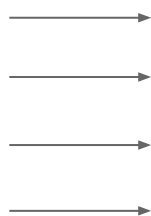
$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$



## Backprop with Vectors

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

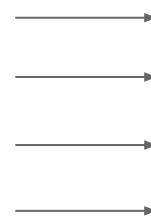


$$f(x) = \max(0, x)$$

*(elementwise)*

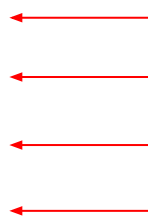
4D output z:

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$



4D dL/dz:

$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$

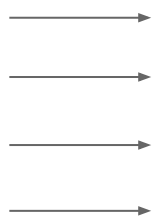


Upstream  
gradient

## Backprop with Vectors

4D input  $x$ :

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

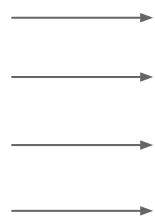


$$f(x) = \max(0, x)$$

*(elementwise)*

4D output  $z$ :

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$

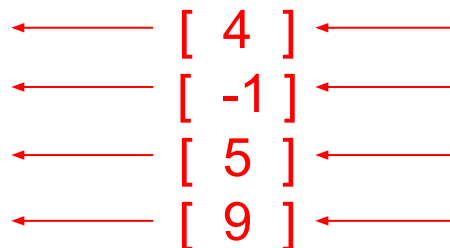


Jacobian  $dz/dx$

$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

4D  $dL/dz$ :

$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$

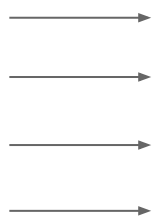


Upstream  
gradient

## Backprop with Vectors

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

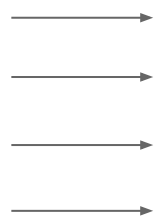


$$f(x) = \max(0, x)$$

(*elementwise*)

4D output z:

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$



$[dz/dx] [dL/dz]$

$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \end{bmatrix}$

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

$\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 5 \end{bmatrix}$

$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 9 \end{bmatrix}$

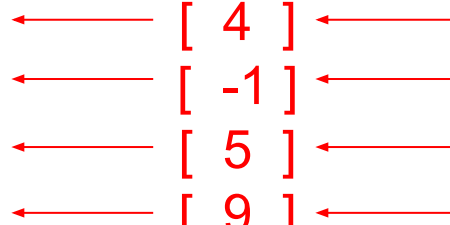
4D dL/dz:

$\begin{bmatrix} 4 \end{bmatrix}$

$\begin{bmatrix} -1 \end{bmatrix}$

$\begin{bmatrix} 5 \end{bmatrix}$

$\begin{bmatrix} 9 \end{bmatrix}$

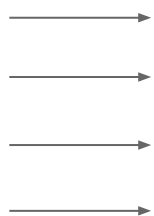


Upstream  
gradient

## Backprop with Vectors

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

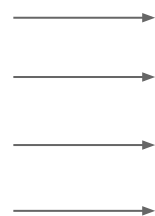


$$f(x) = \max(0, x)$$

(*elementwise*)

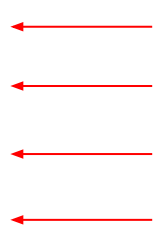
4D output z:

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$



4D dL/dx:

$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix}$

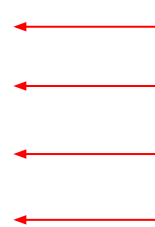


$\begin{bmatrix} dz/dx & dL/dz \end{bmatrix}$

$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$

4D dL/dz:

$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$



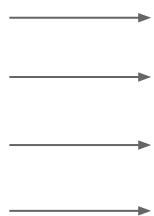
Upstream  
gradient

## Backprop with Vectors

Jacobian is **sparse**:  
off-diagonal entries  
always zero! Never  
**explicitly** form  
Jacobian -- instead  
use **implicit**  
multiplication

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

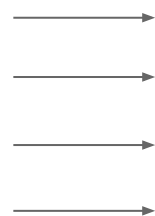


$$f(x) = \max(0, x)$$

(*elementwise*)

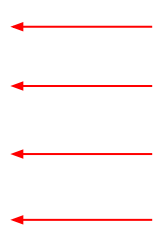
4D output z:

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$



4D dL/dx:

$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix}$

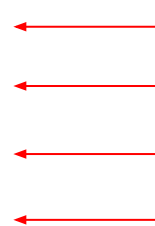


$\begin{bmatrix} dz/dx & dL/dz \end{bmatrix}$

$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$

4D dL/dz:

$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$



Upstream  
gradient

## Backprop with Vectors

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

$$f(x) = \max(0, x) \\ (\textit{elementwise})$$

4D output z:

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$

Jacobian is **sparse**:  
off-diagonal entries  
always zero! Never  
**explicitly** form  
Jacobian -- instead  
use **implicit**  
multiplication

4D  $dL/dx$ :

$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix}$

$[dz/dx] [dL/dz]$

$$\left( \frac{\partial L}{\partial x} \right)_i = \begin{cases} \left( \frac{\partial L}{\partial z} \right)_i & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

4D  $dL/dz$ :

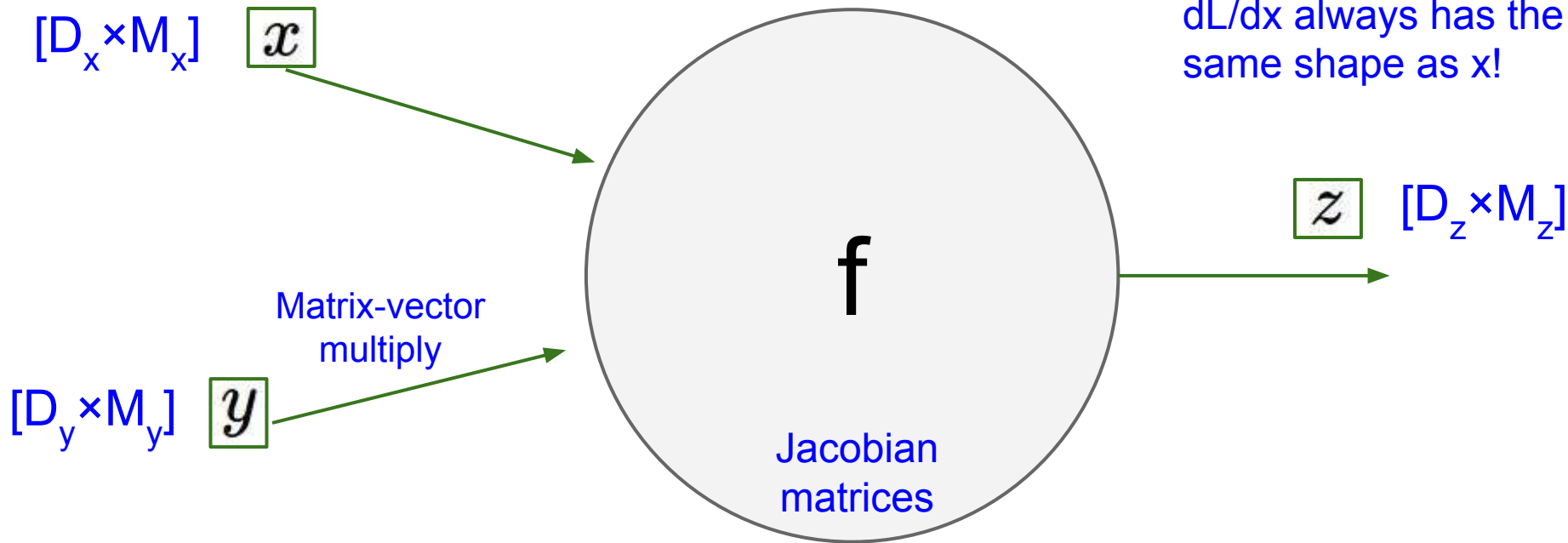
$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$

Upstream  
gradient

# Backprop with Matrices (or Tensors)

Loss  $L$  still a scalar!

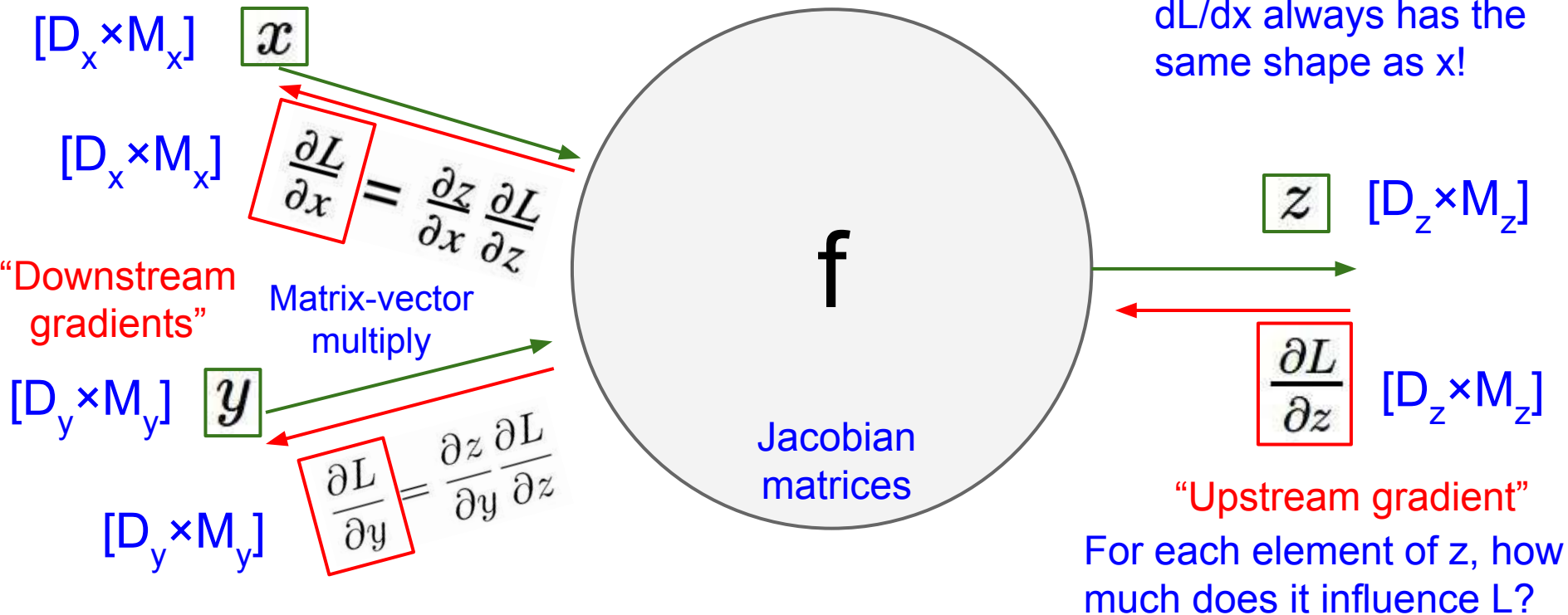
$dL/dx$  always has the same shape as  $x$ !



# Backprop with Matrices (or Tensors)

Loss L still a scalar!

$dL/dx$  always has the same shape as  $x$ !

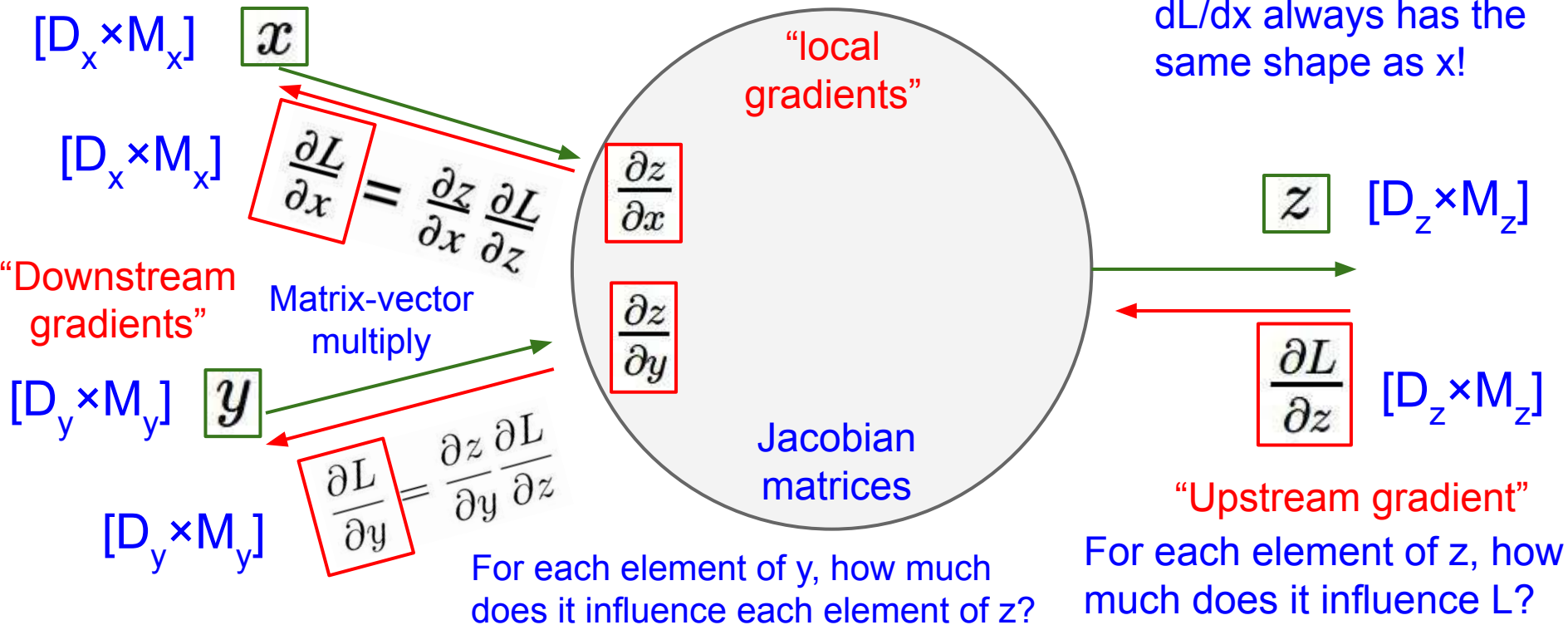




# Backprop with Matrices (or Tensors)

Loss L still a scalar!

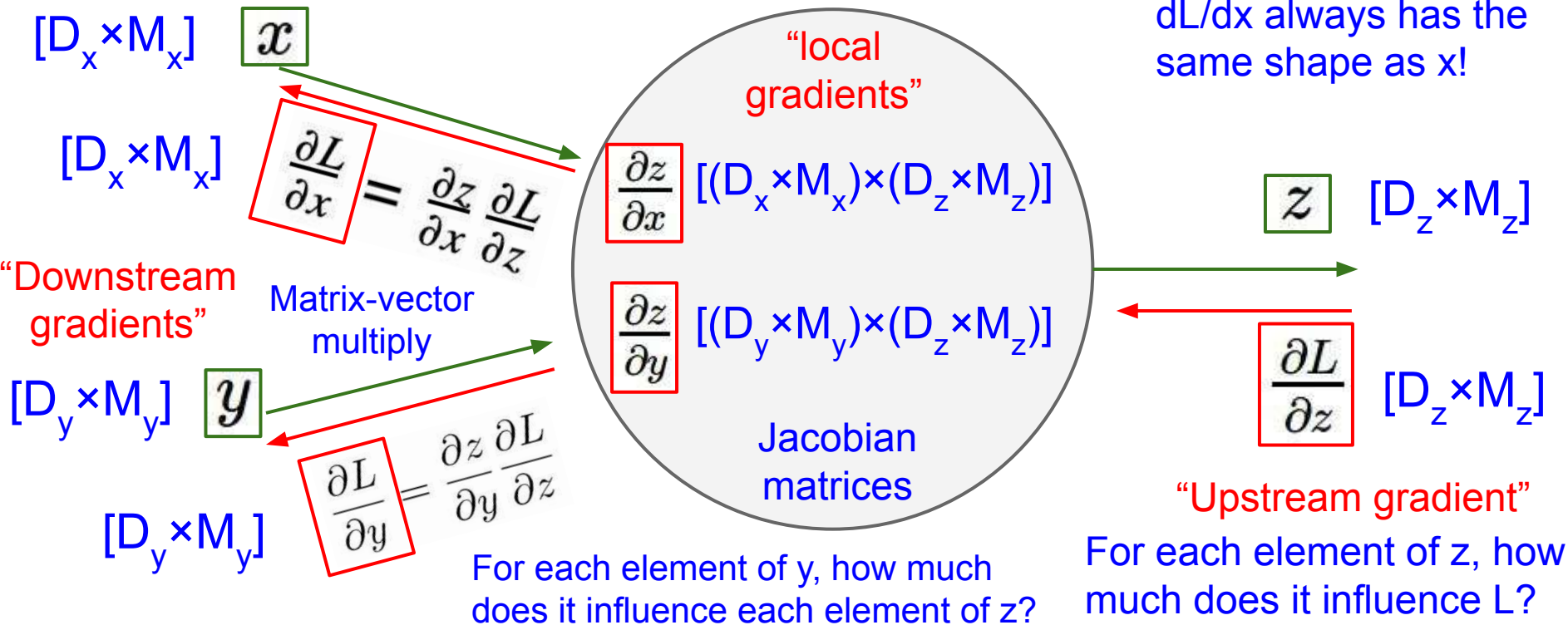
$dL/dx$  always has the same shape as  $x$ !



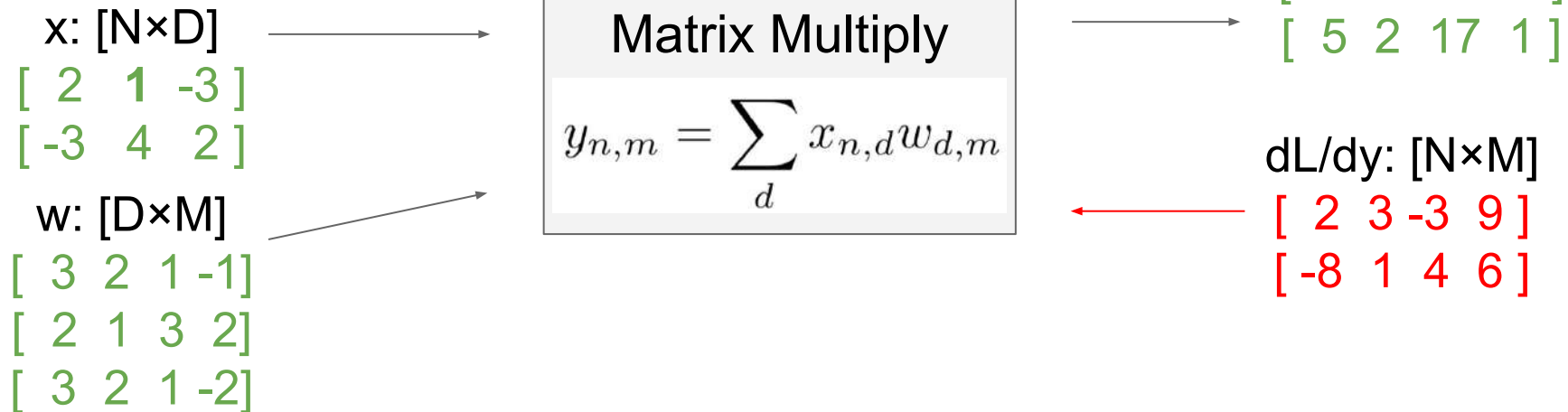
# Backprop with Matrices (or Tensors)

Loss L still a scalar!

$dL/dx$  always has the same shape as  $x$ !



# Backprop with Matrices



Also see derivation in the course notes:

<http://cs231n.stanford.edu/handouts/linear-backprop.pdf>

# Backprop with Matrices

x: [N×D]

[ 2 1 -3 ]

[ -3 4 2 ]

w: [D×M]

[ 3 2 1 -1 ]

[ 2 1 3 2 ]

[ 3 2 1 -2 ]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

**Jacobians:**

dy/dx: [(N×D)×(N×M)]

dy/dw: [(D×M)×(N×M)]

y: [N×M]

[13 9 -2 -6]

[ 5 2 17 1 ]

dL/dy: [N×M]

[ 2 3 -3 9 ]

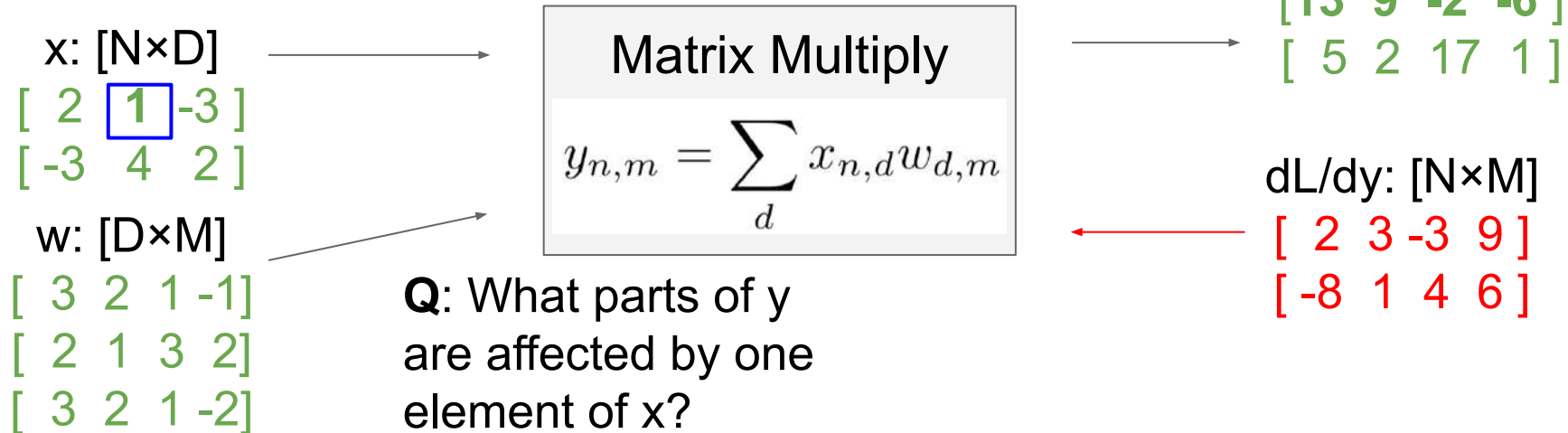
[ -8 1 4 6 ]

For a neural net we may have

N=64, D=M=4096

Each Jacobian takes ~256 GB of  
memory! Must work with them implicitly!

# Backprop with Matrices



# Backprop with Matrices

x: [N×D]  
[ 2 1 -3 ]  
[ -3 4 2 ]

w: [D×M]  
[ 3 2 1 -1 ]  
[ 2 1 3 2 ]  
[ 3 2 1 -2 ]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]  
[ 13 9 -2 -6 ]  
[ 5 2 17 1 ]

dL/dy: [N×M]  
[ 2 3 -3 9 ]  
[ -8 1 4 6 ]

**Q:** What parts of y are affected by one element of x?

**A:**  $x_{n,d}$  affects the whole row  $y_{n,\cdot}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}$$

# Backprop with Matrices

x: [N×D]  
[ 2 1 -3 ]  
[ -3 4 2 ]

w: [D×M]  
[ 3 2 1 -1 ]  
[ 2 1 3 2 ]  
[ 3 2 1 -2 ]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]  
[ 13 9 -2 -6 ]  
[ 5 2 17 1 ]

dL/dy: [N×M]  
[ 2 3 -3 9 ]  
[ -8 1 4 6 ]

**Q:** What parts of y are affected by one element of x?

**A:**  $x_{n,d}$  affects the whole row  $y_{n,\cdot}$

**Q:** How much does  $x_{n,d}$  affect  $y_{n,m}$ ?

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}$$

# Backprop with Matrices

$x: [N \times D]$   
 $\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$

$w: [D \times M]$   
 $\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

$y: [N \times M]$   
 $\begin{bmatrix} \boxed{13} & 9 & \boxed{-2} & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$

$dL/dy: [N \times M]$   
 $\begin{bmatrix} \boxed{2} & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$

**Q:** What parts of  $y$  are affected by one element of  $x$ ?

**A:**  $x_{n,d}$  affects the whole row  $y_{n,\cdot}$ .

**Q:** How much does  $x_{n,d}$  affect  $y_{n,m}$ ?

**A:**  $w_{d,m}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$



# Backprop with Matrices

$$x: [N \times D]$$

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

$$w: [D \times M]$$

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

$$[N \times D] \quad [N \times M] \quad [M \times D]$$

$$\frac{\partial L}{\partial x} = \left( \frac{\partial L}{\partial y} \right) w^T$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

$$y: [N \times M]$$

$$\begin{bmatrix} \boxed{13} & 9 & \boxed{-2} & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

$$dL/dy: [N \times M]$$

$$\begin{bmatrix} \boxed{2} & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

**Q:** What parts of  $y$  are affected by one element of  $x$ ?

**A:**  $x_{n,d}$  affects the whole row  $y_{n,\cdot}$ .

**Q:** How much does  $x_{n,d}$  affect  $y_{n,m}$ ?

**A:**  $w_{d,m}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

# Backprop with Matrices

$$x: [N \times D]$$

$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

$$w: [D \times M]$$

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

$$y: [N \times M]$$

$$\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

$$dL/dy: [N \times M]$$

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

By similar logic:

$$[N \times D] \quad [N \times M] \quad [M \times D]$$

$$\frac{\partial L}{\partial x} = \left( \frac{\partial L}{\partial y} \right) w^T$$

$$[D \times M] \quad [D \times N] \quad [N \times M]$$

$$\frac{\partial L}{\partial w} = x^T \left( \frac{\partial L}{\partial y} \right)$$

These formulas are easy to remember: they are the only way to make shapes match up!

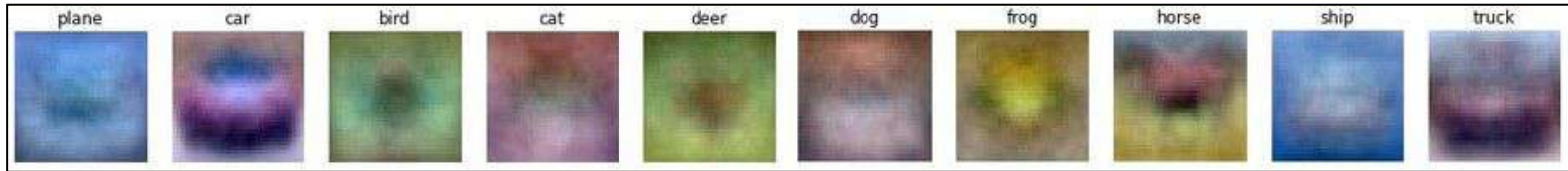
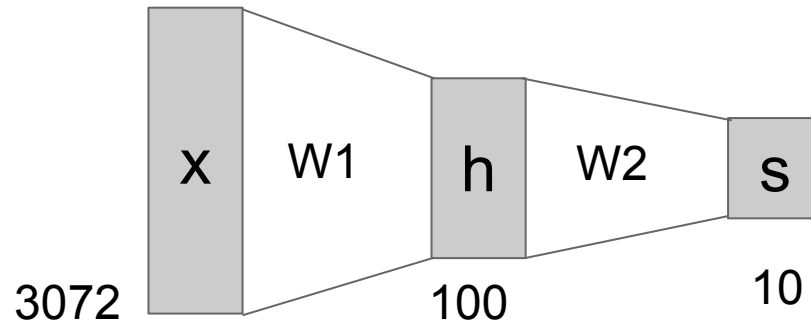
# Wrapping up: Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network

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



# Next: Convolutional Neural Networks

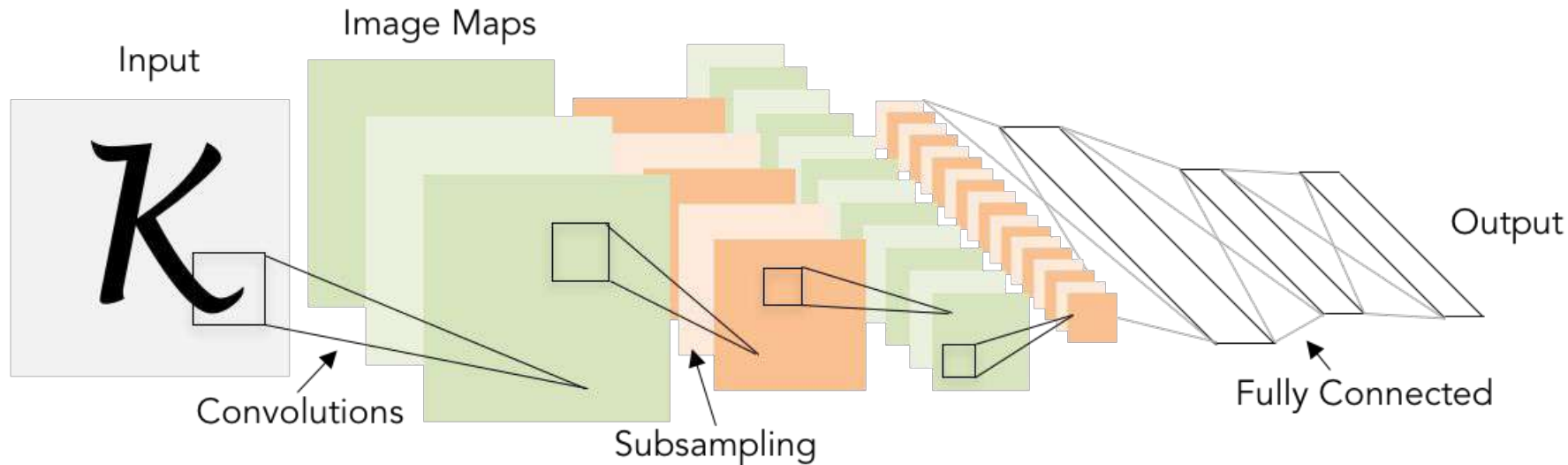


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

# A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

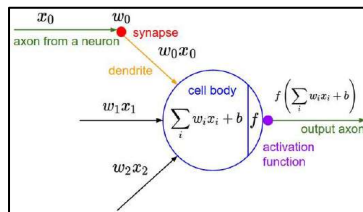
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized  
letters of the alphabet

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

update rule:

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

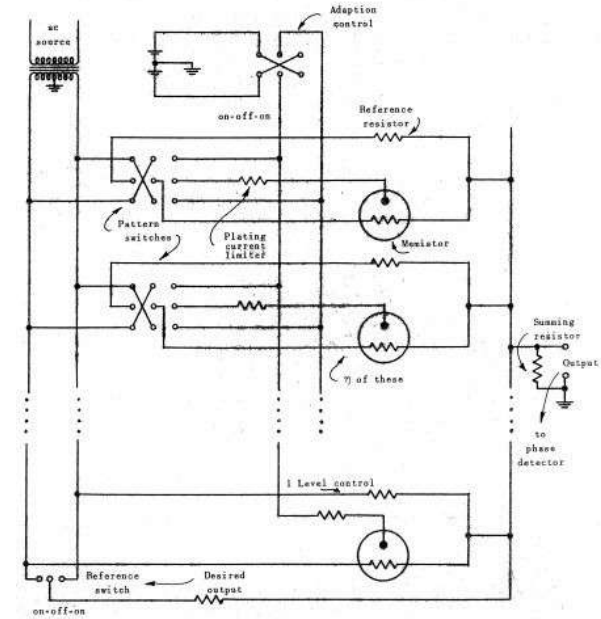
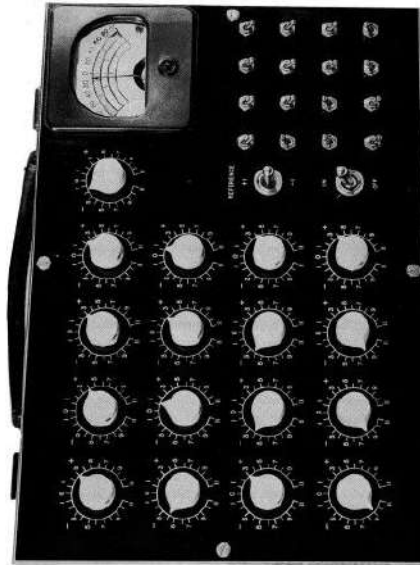
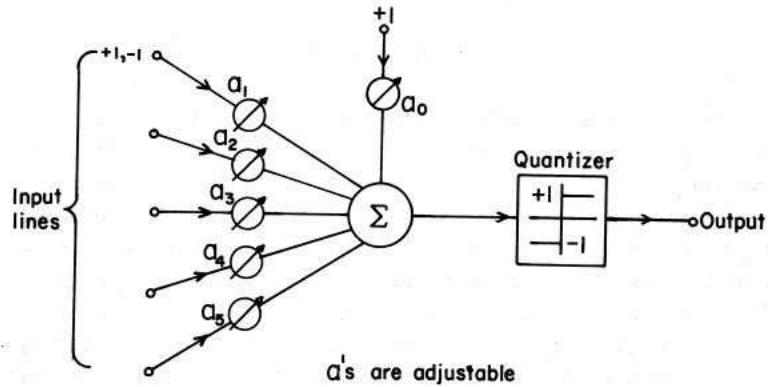


Frank Rosenblatt, ~1957: Perceptron



[This image](#) by Rocky Acosta is licensed under [CC-BY 3.0](#)

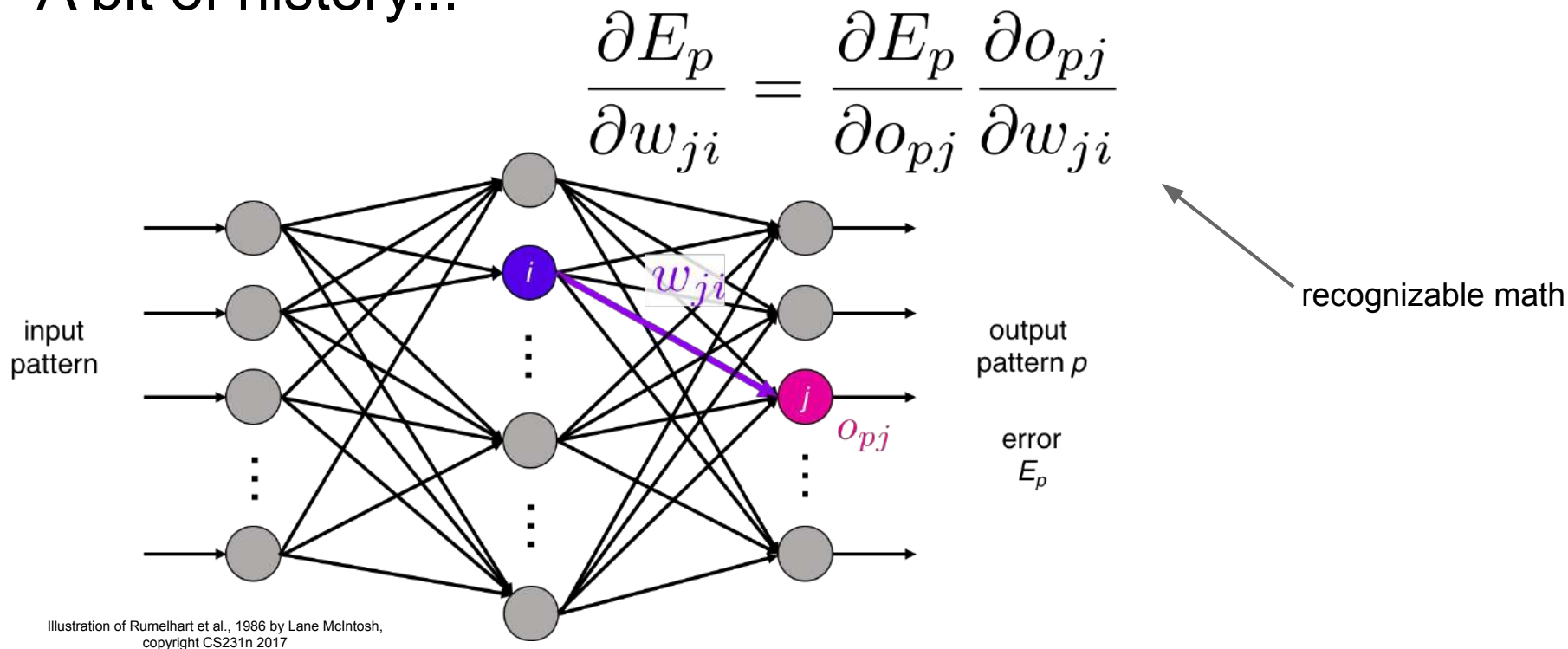
# A bit of history...



Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from [Widrow 1960, Stanford Electronics Laboratories Technical Report](#) with permission from [Stanford University Special Collections](#).

# A bit of history...



Rumelhart et al., 1986: First time back-propagation became popular

# A bit of history...

[Hinton and Salakhutdinov 2006]

## Reinvigorated research in Deep Learning

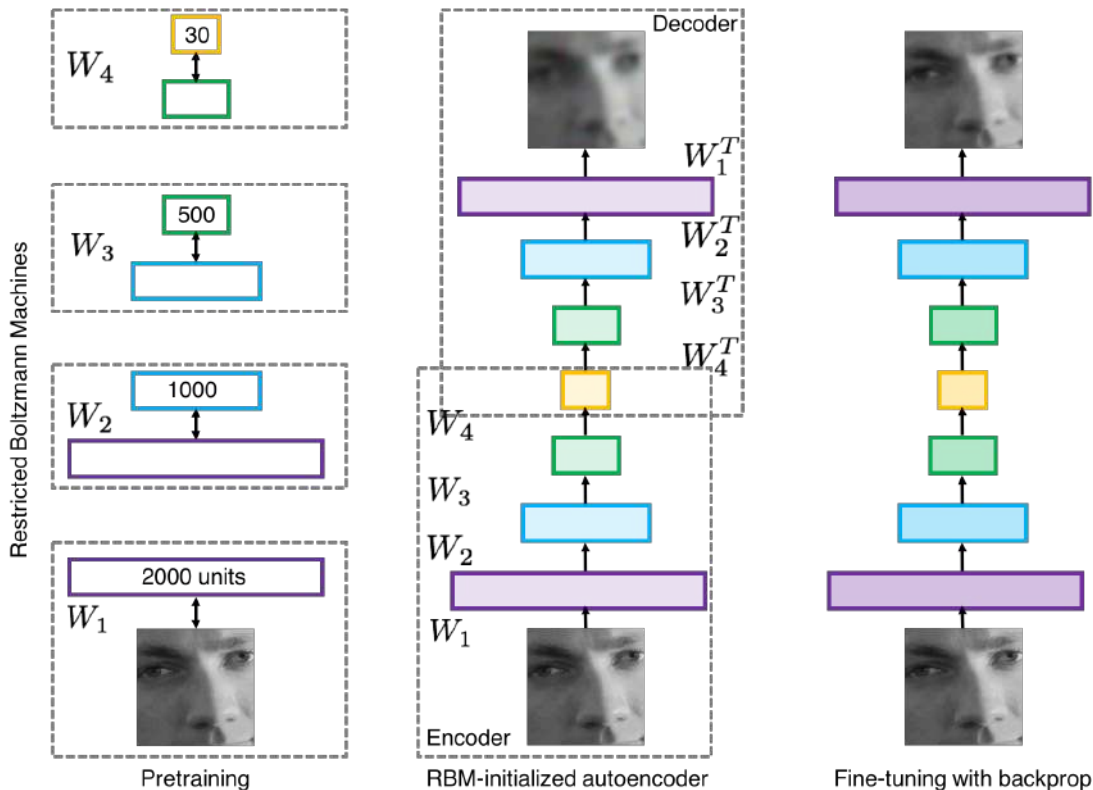


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017



# First strong results

## ***Acoustic Modeling using Deep Belief Networks***

Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

## ***Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition***

George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

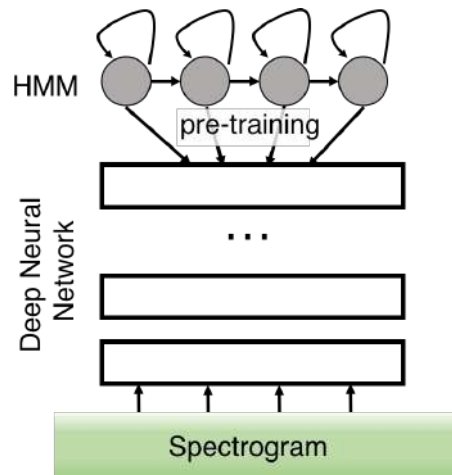
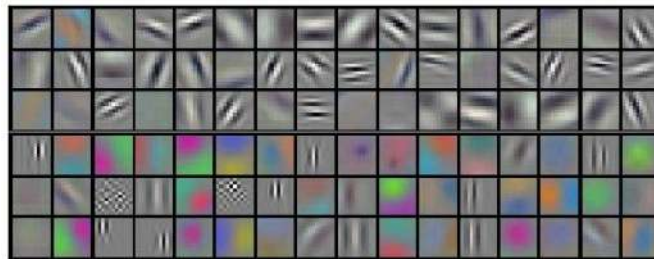
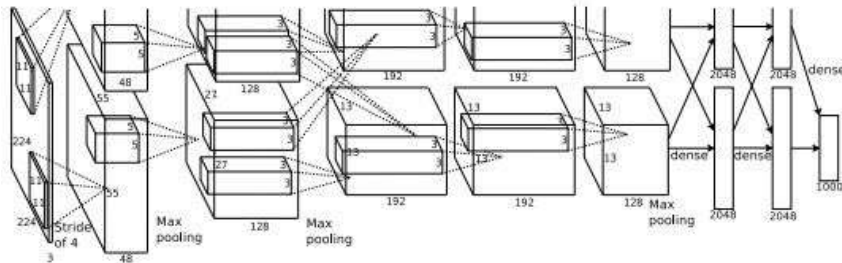


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

## ***Imagenet classification with deep convolutional neural networks***

Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

A bit of history:

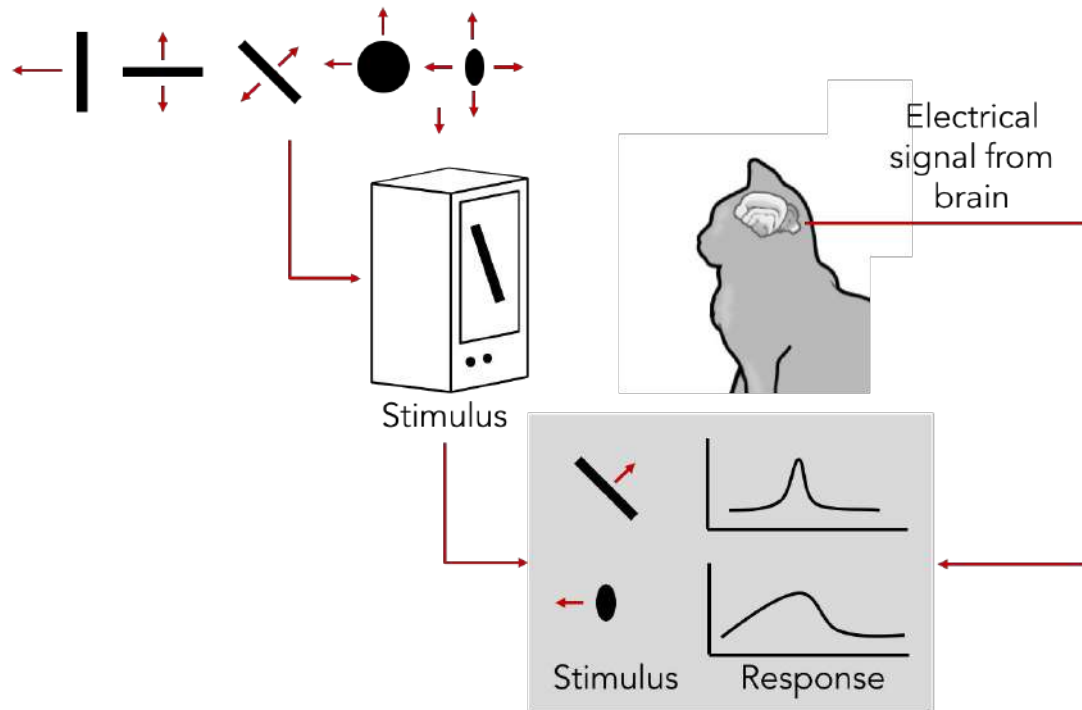
## Hubel & Wiesel, 1959

RECEPTIVE FIELDS OF SINGLE  
NEURONES IN  
THE CAT'S STRIATE CORTEX

## 1962

RECEPTIVE FIELDS, BINOCULAR  
INTERACTION  
AND FUNCTIONAL ARCHITECTURE IN  
THE CAT'S VISUAL CORTEX

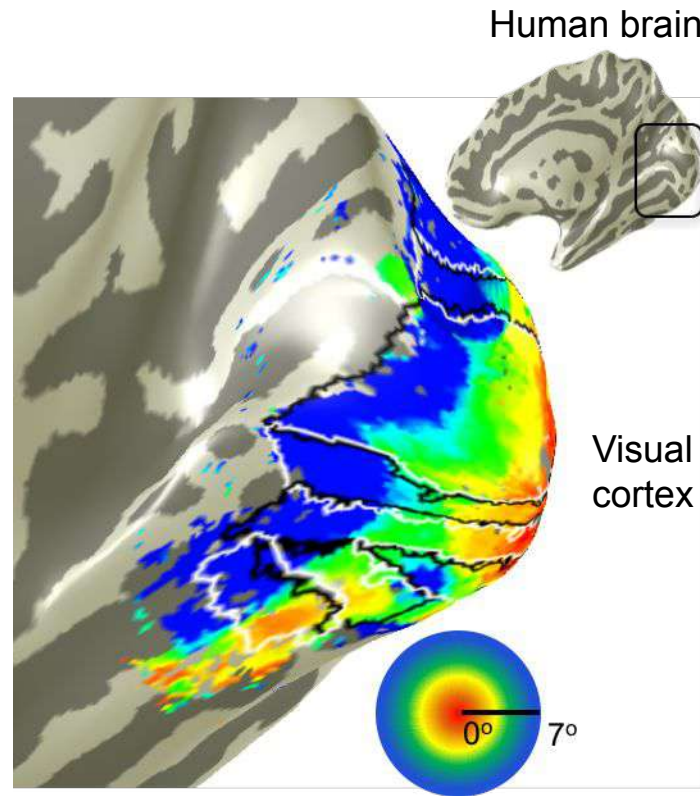
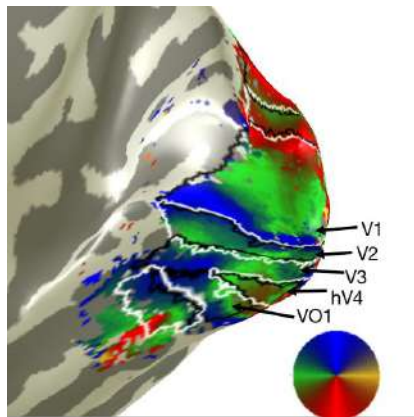
## 1968...



[Cat image](#) by CNX OpenStax is licensed under CC BY 4.0; changes made

# A bit of history

**Topographical mapping in the cortex:**  
nearby cells in cortex represent  
nearby regions in the visual field



Retinotopy images courtesy of Jesse Gomez in the  
Stanford Vision & Perception Neuroscience Lab.

# Hierarchical organization

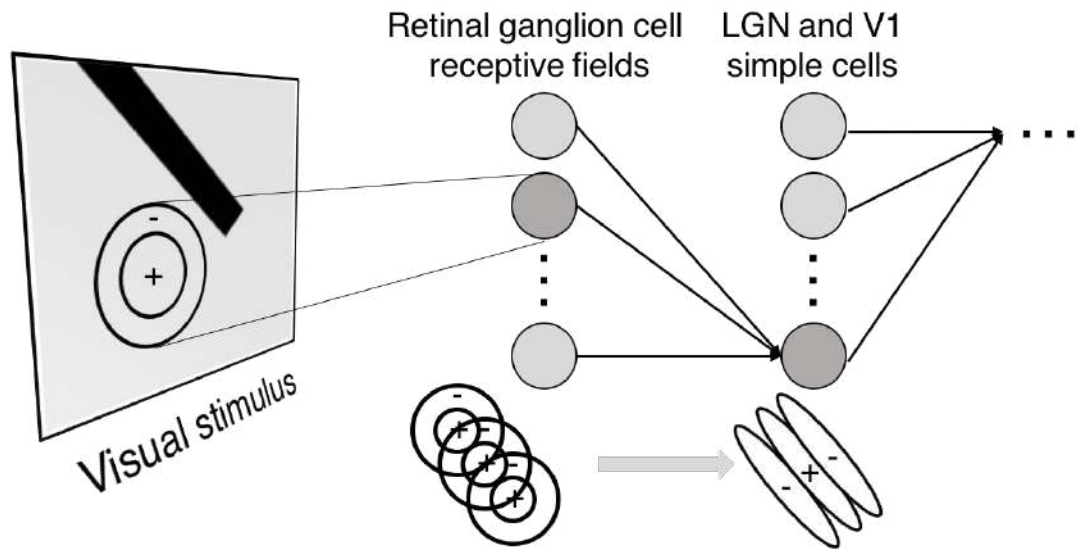
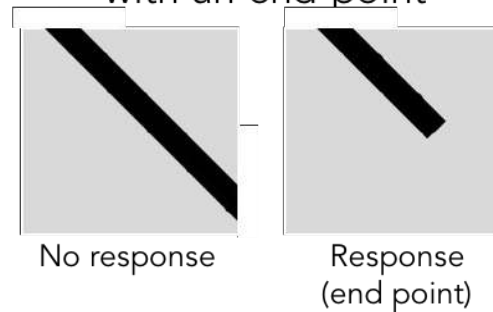


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

**Simple cells:**  
Response to light  
orientation

**Complex cells:**  
Response to light  
orientation and movement

**Hypercomplex cells:**  
response to movement  
with an end point

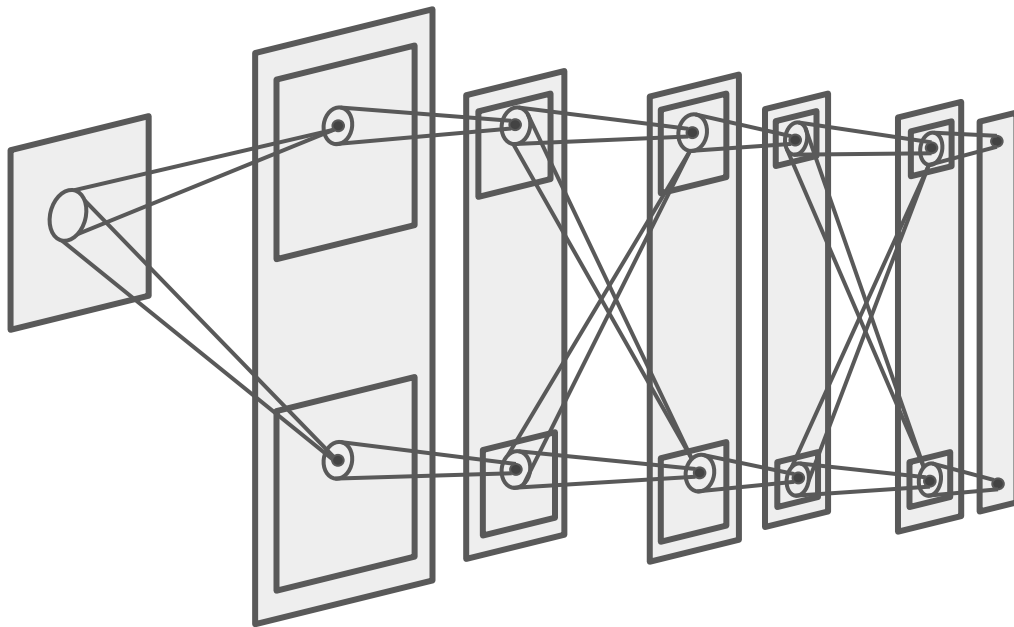


A bit of history:

# Neocognitron

*[Fukushima 1980]*

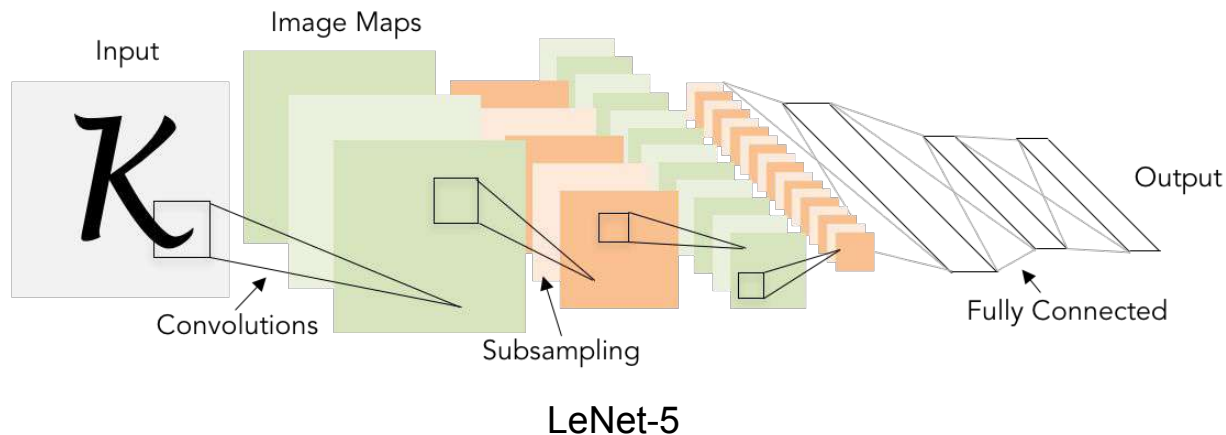
“sandwich” architecture (SCSCSC...)  
simple cells: modifiable parameters  
complex cells: perform pooling



A bit of history:

## Gradient-based learning applied to document recognition

*[LeCun, Bottou, Bengio, Haffner 1998]*



# A bit of history:

## ImageNet Classification with Deep Convolutional Neural Networks

*[Krizhevsky, Sutskever, Hinton, 2012]*

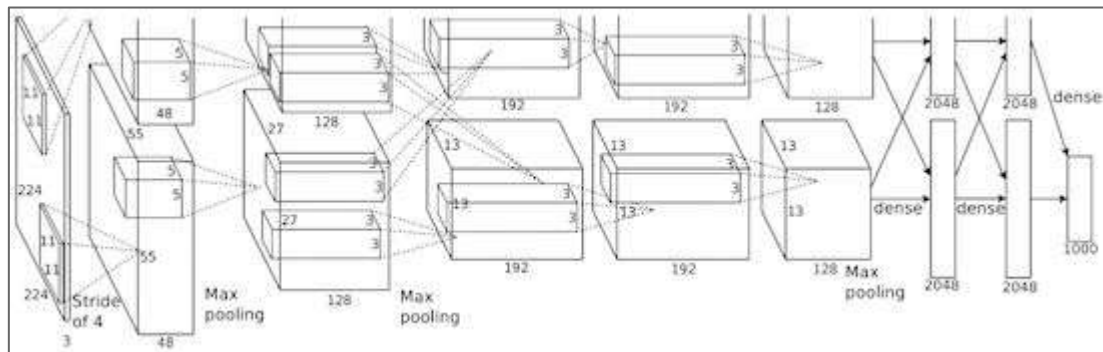


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”



# Fast-forward to today: ConvNets are everywhere

Classification



Retrieval

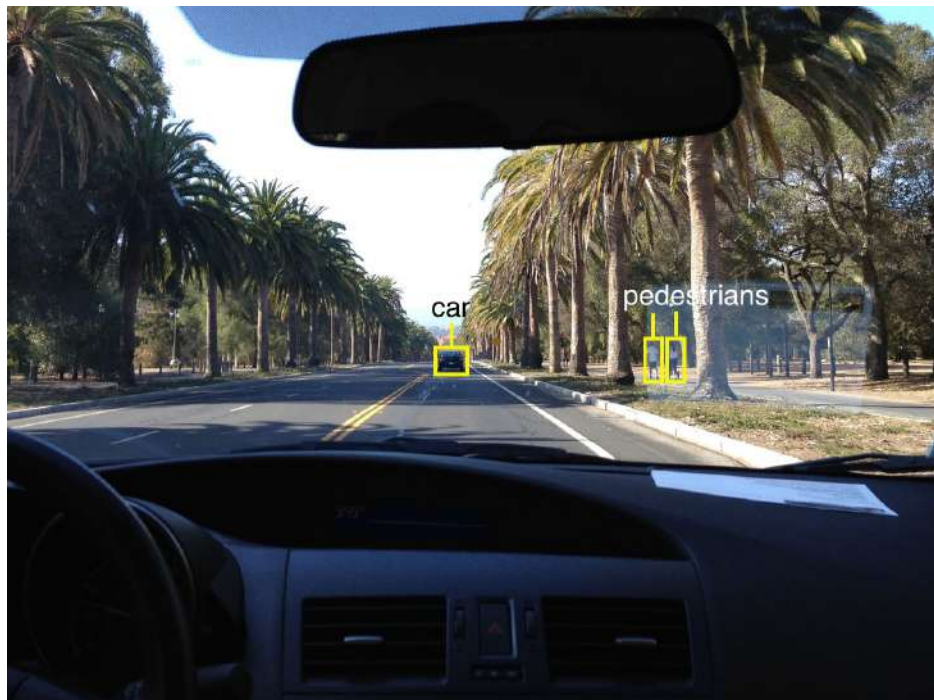


Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.





# Fast-forward to today: ConvNets are everywhere



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



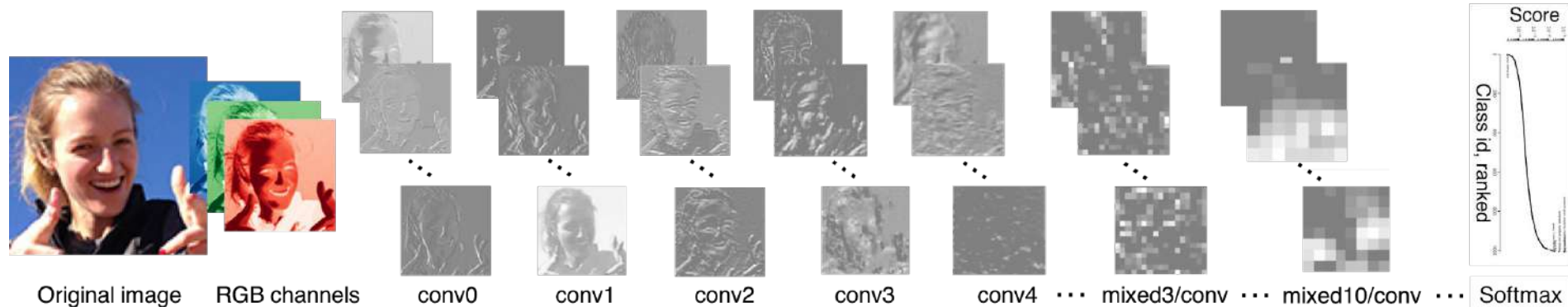
[This image](#) by GBPublic\_PR is licensed under [CC-BY 2.0](#)

## NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

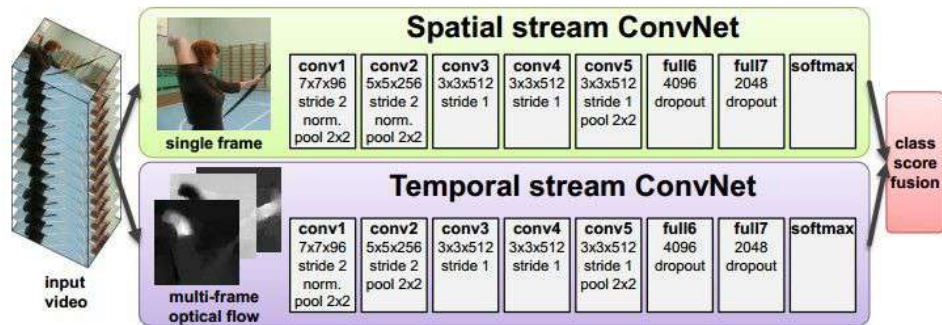
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

# Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]

Activations of [inception-v3 architecture](#) [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

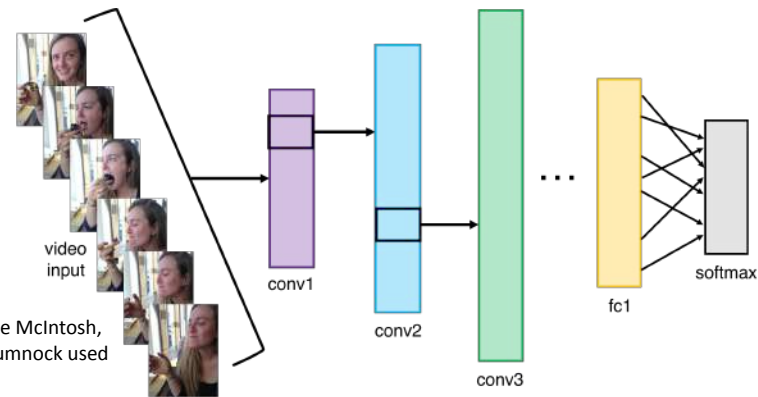


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

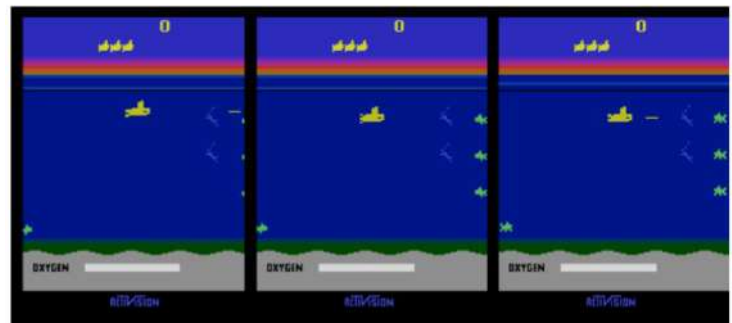
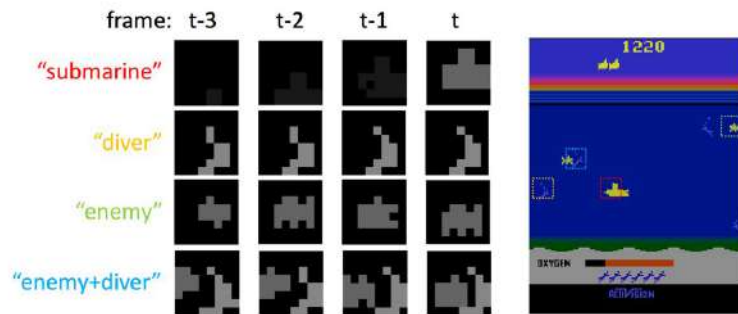


# Fast-forward to today: ConvNets are everywhere



Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

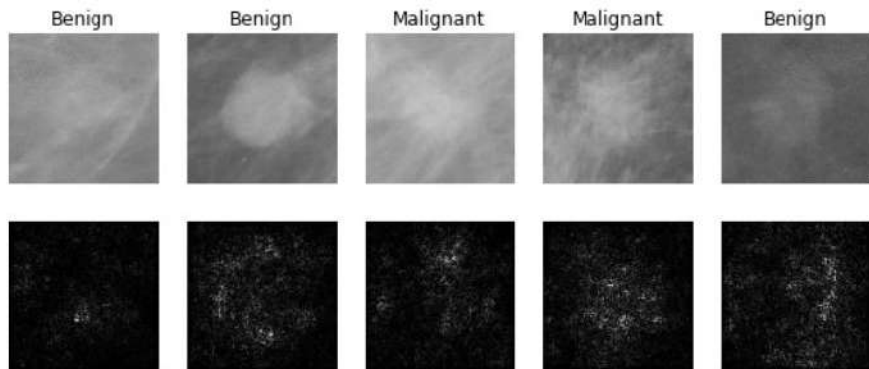
[Toshev, Szegedy 2014]



Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

[Guo et al. 2014]

# Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

Figure copyright Levy et al. 2016.  
Reproduced with permission.



[Dieleman et al. 2014]

From left to right: [public domain by NASA](#), usage [permitted](#) by ESA/Hubble, [public domain by NASA](#), and [public domain](#).



[Sermanet et al. 2011]  
[Ciresan et al.]

Photos by Lane McIntosh.  
Copyright CS231n 2017.

[This image](#) by Christin Khan is in the public domain and originally came from the U.S. NOAA.



*Whale recognition, Kaggle Challenge*

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.



*Mnih and Hinton, 2010*



No errors



*A white teddy bear sitting in the grass*

Minor errors



*A man in a baseball uniform throwing a ball*

Somewhat related



*A woman is holding a cat in her hand*

# Image Captioning

*[Vinyals et al., 2015]  
[Karpathy and Fei-Fei, 2015]*



*A man riding a wave on top of a surfboard*



*A cat sitting on a suitcase on the floor*

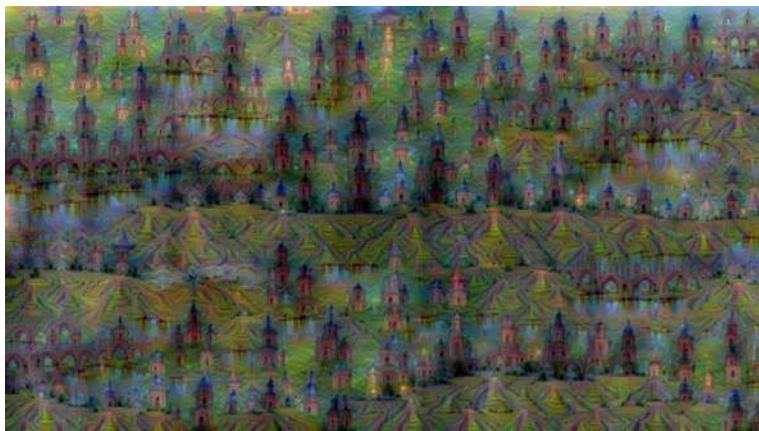
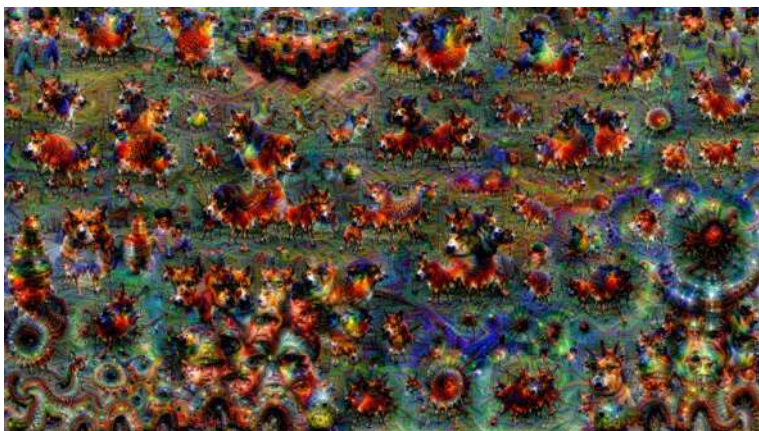


*A woman standing on a beach holding a surfboard*

All images are CC0 Public domain:

<https://pixabay.com/en/luggage-antique-cat-1643010/>  
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>  
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>  
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>  
<https://pixabay.com/en/handstand-lake-meditation-496008/>  
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [NeuralTalk2](#)



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a [blog post](#) by Google Research.



[Original image](#) is CC0 public domain  
[Starry Night](#) and [Tree Roots](#) by Van Gogh are in the public domain  
[Bokeh image](#) is in the public domain  
 Stylized images copyright Justin Johnson, 2017;  
 reproduced with permission



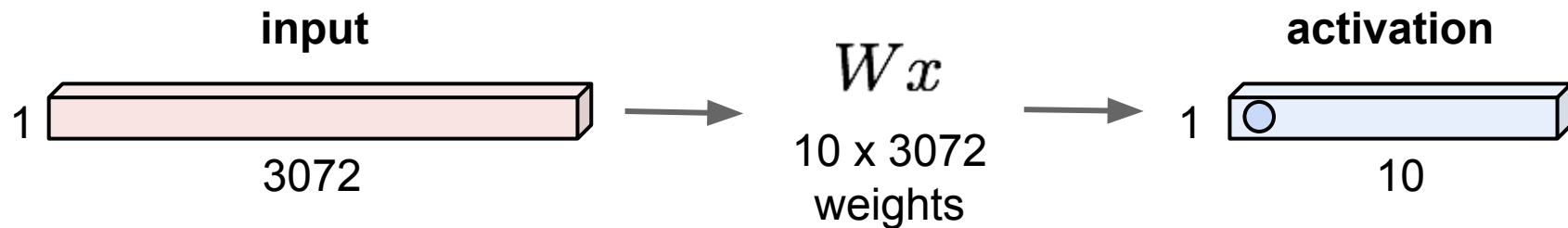
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016  
 Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017



# Convolutional Neural Networks

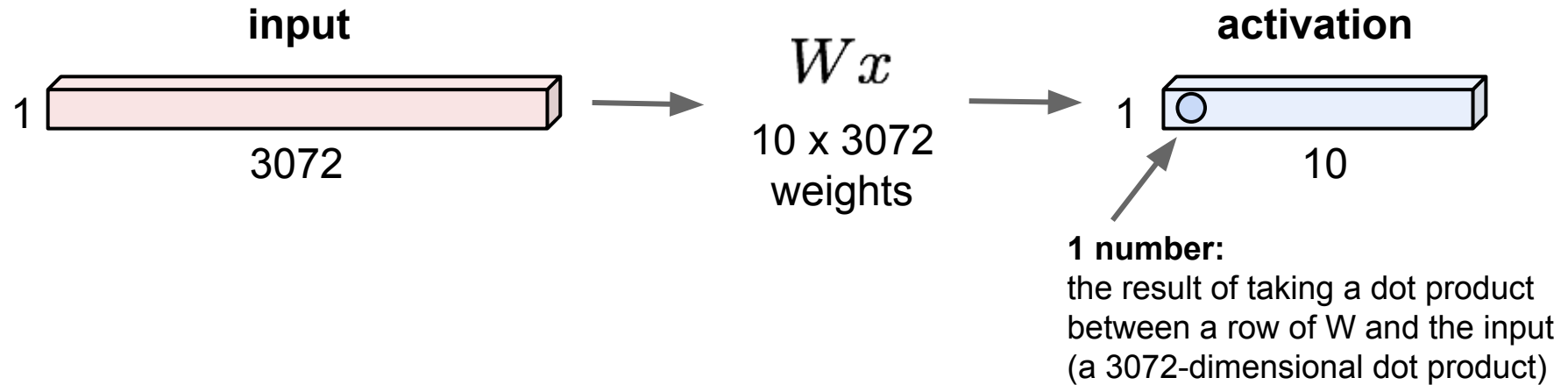
# Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



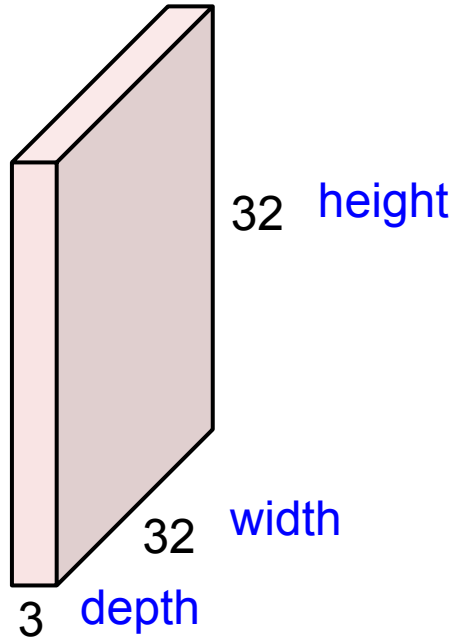
# Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



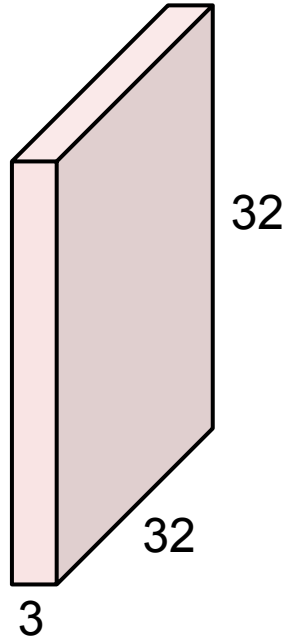
# Convolution Layer

32x32x3 image -> preserve spatial structure



# Convolution Layer

32x32x3 image



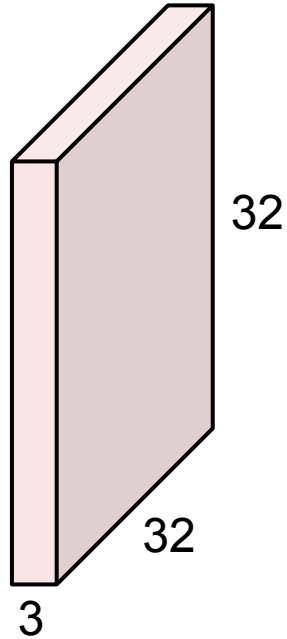
5x5x3 filter



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

# Convolution Layer

32x32x3 image



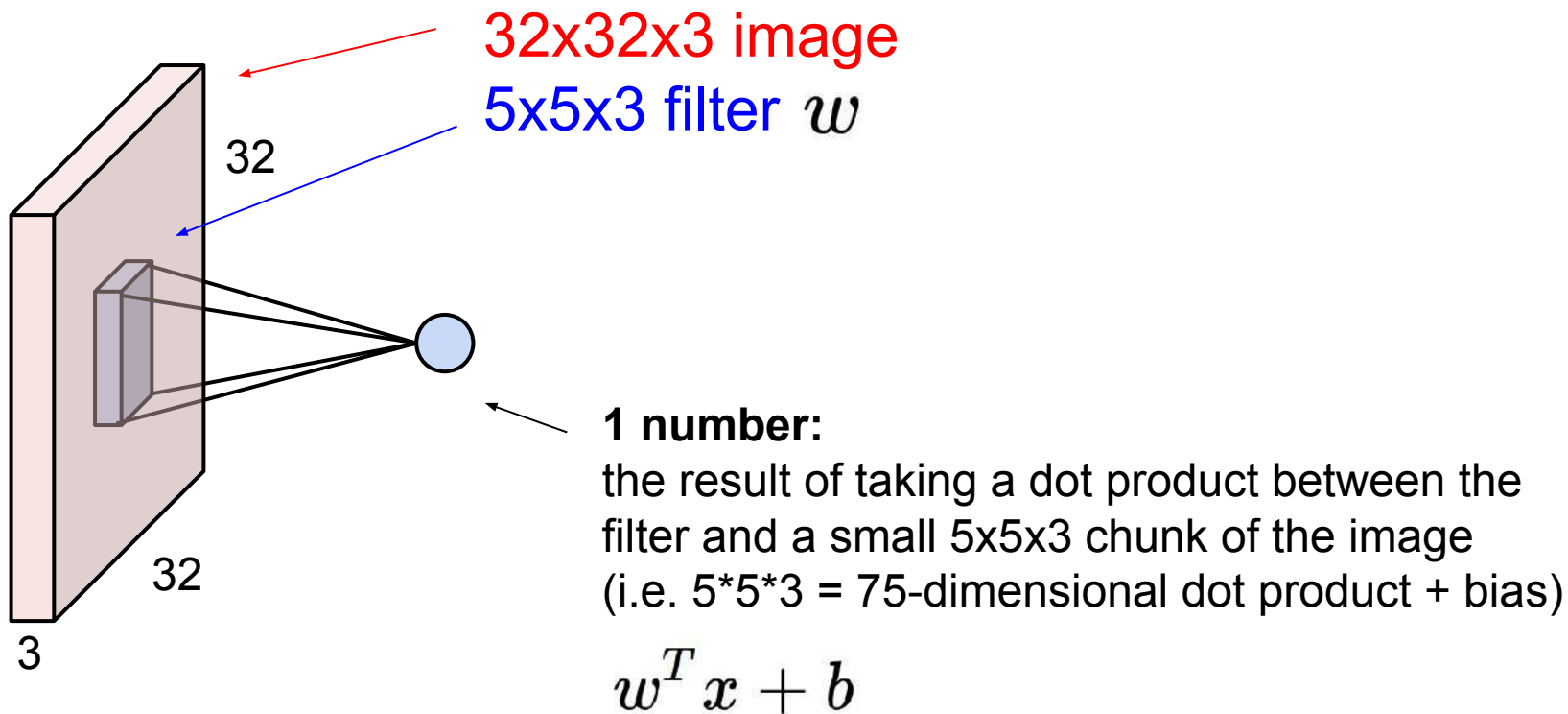
Filters always extend the full depth of the input volume

5x5x3 filter

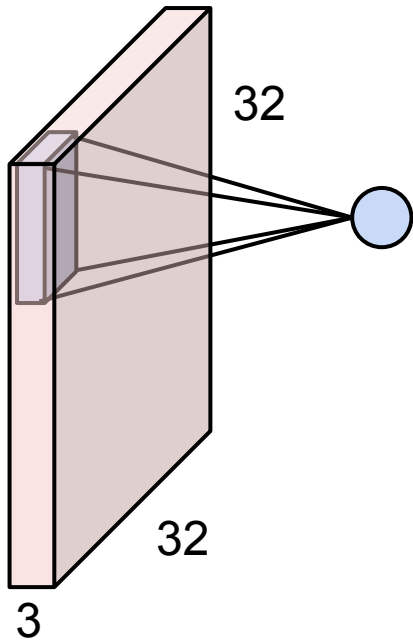


**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

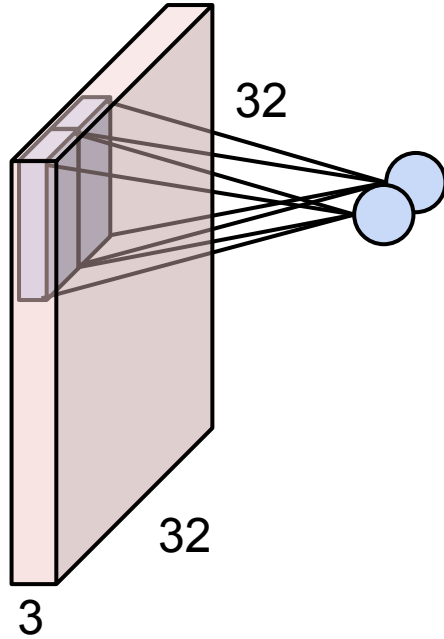


# Convolution Layer

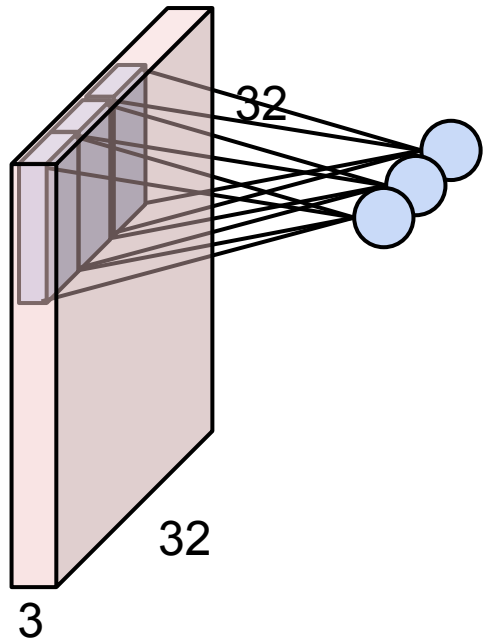




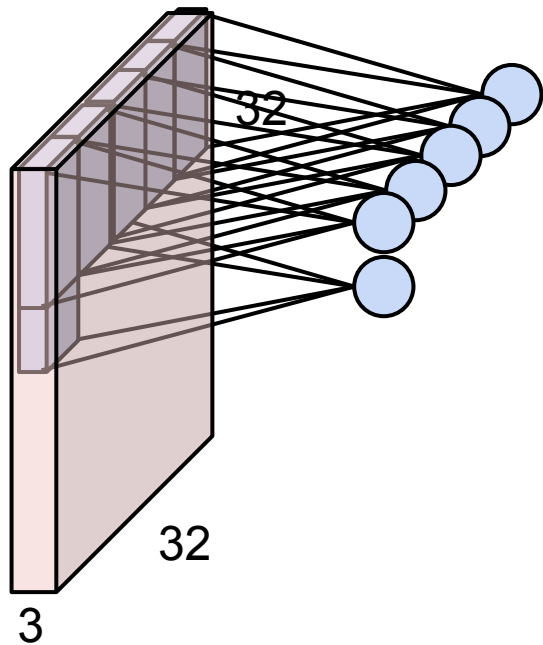
# Convolution Layer



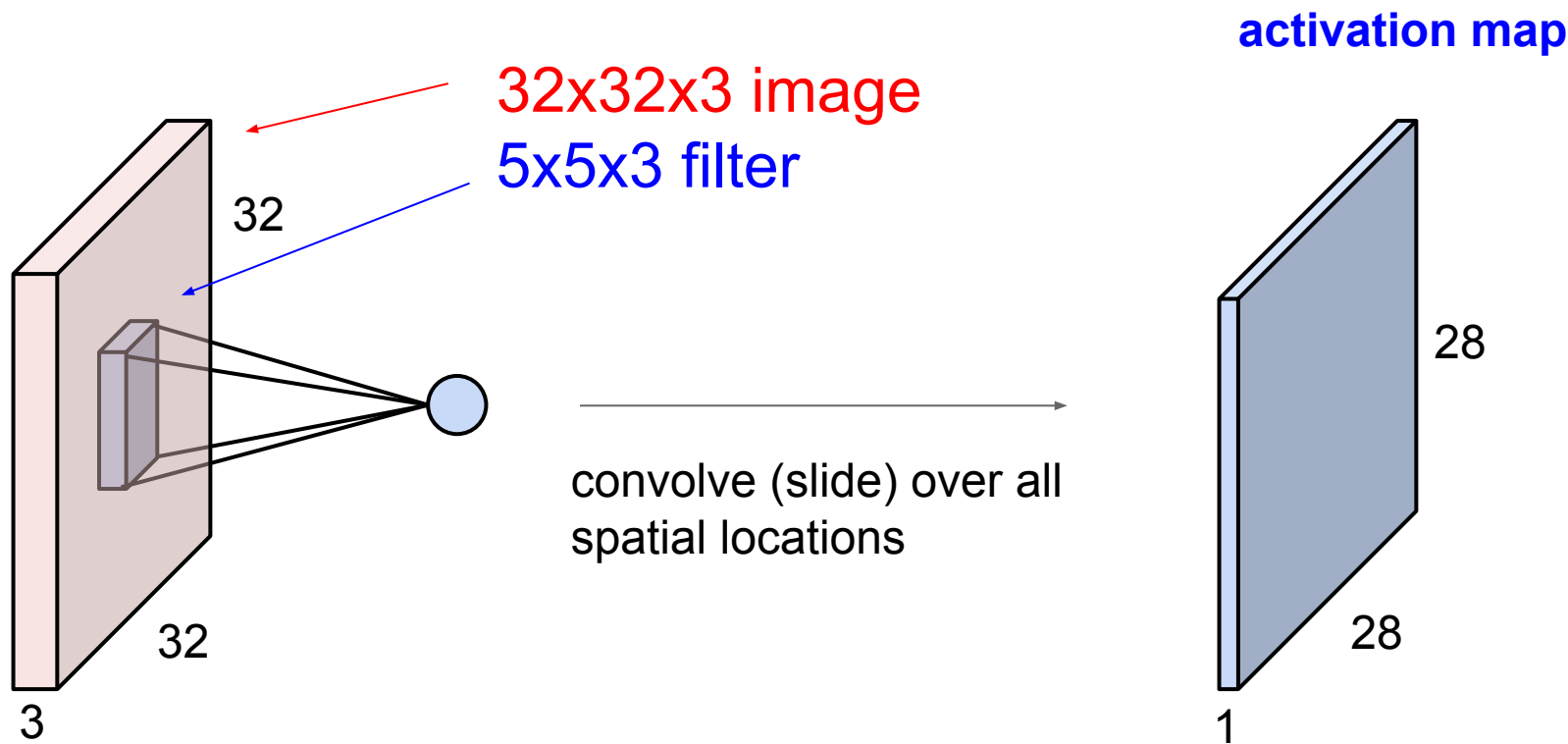
# Convolution Layer



# Convolution Layer

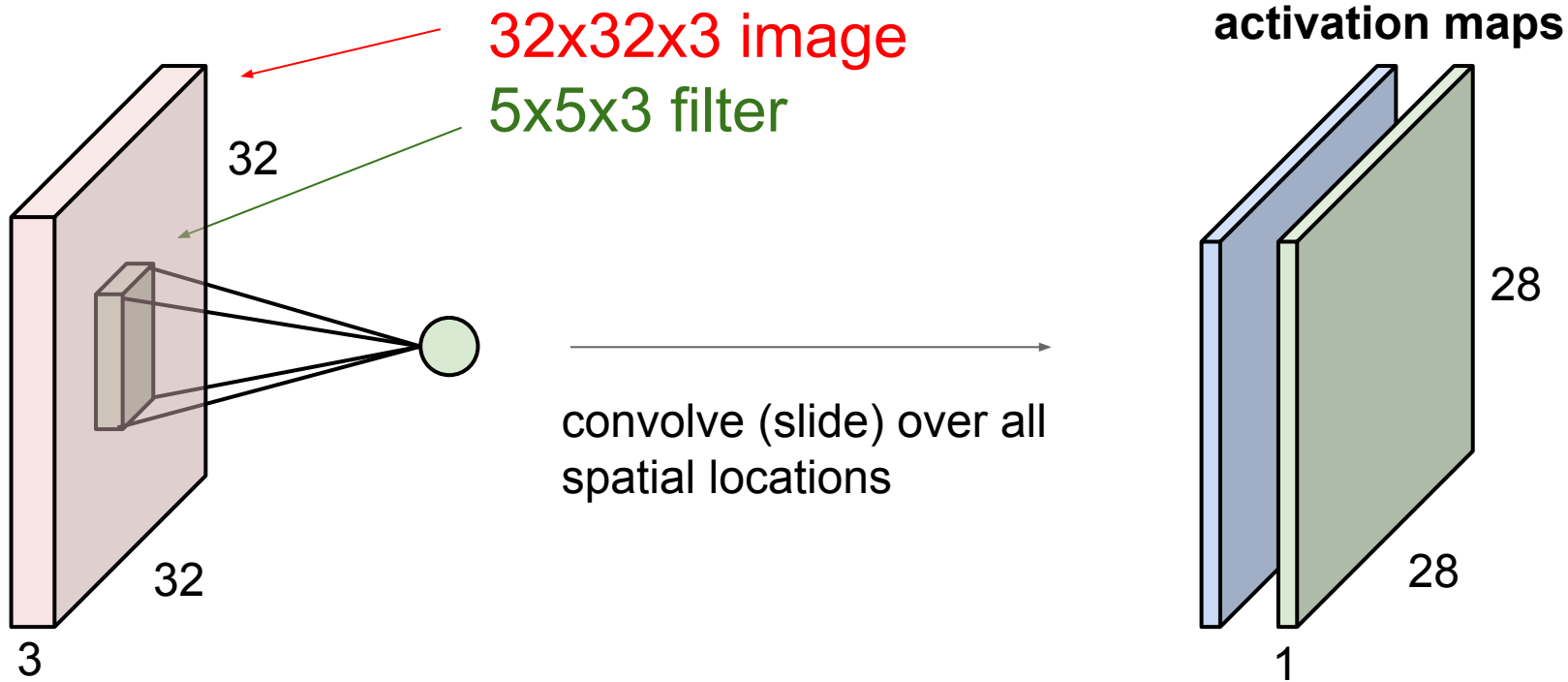


# Convolution Layer

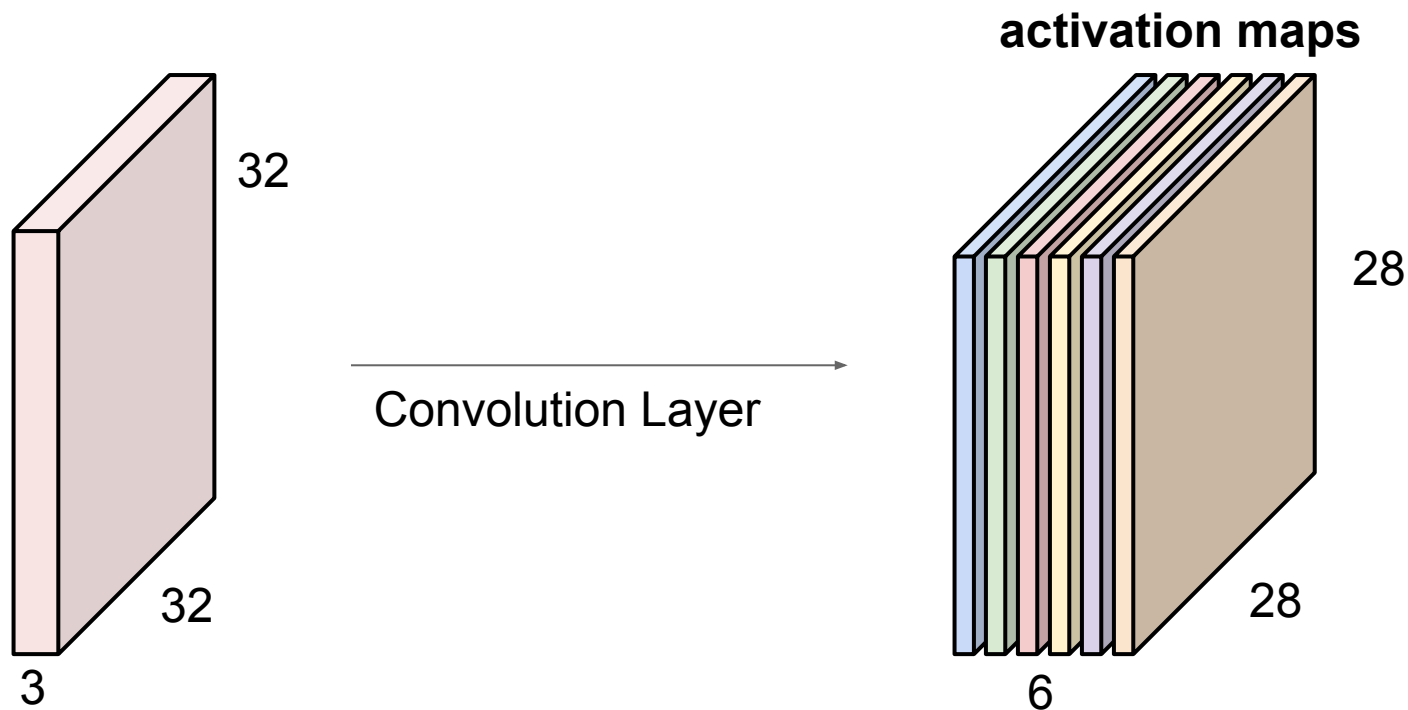


# Convolution Layer

consider a second, **green** filter

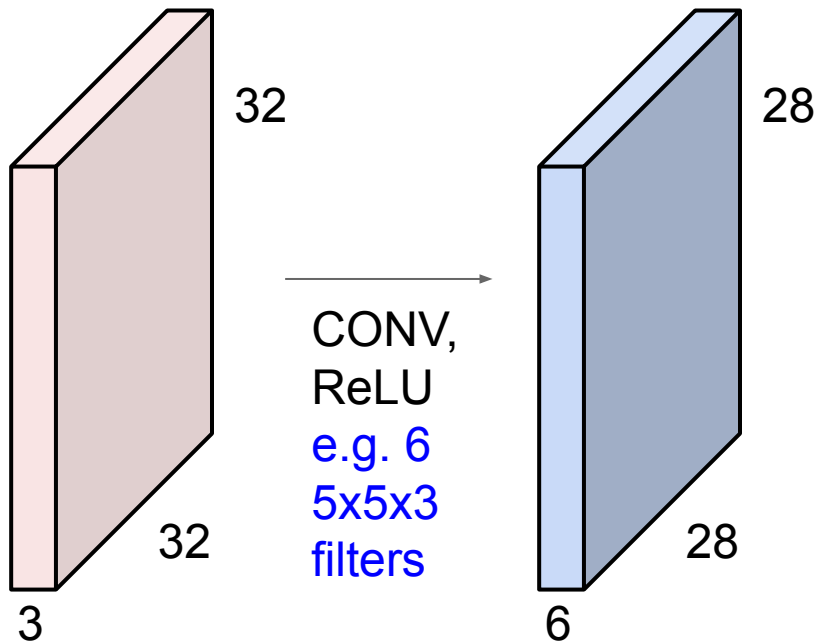


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

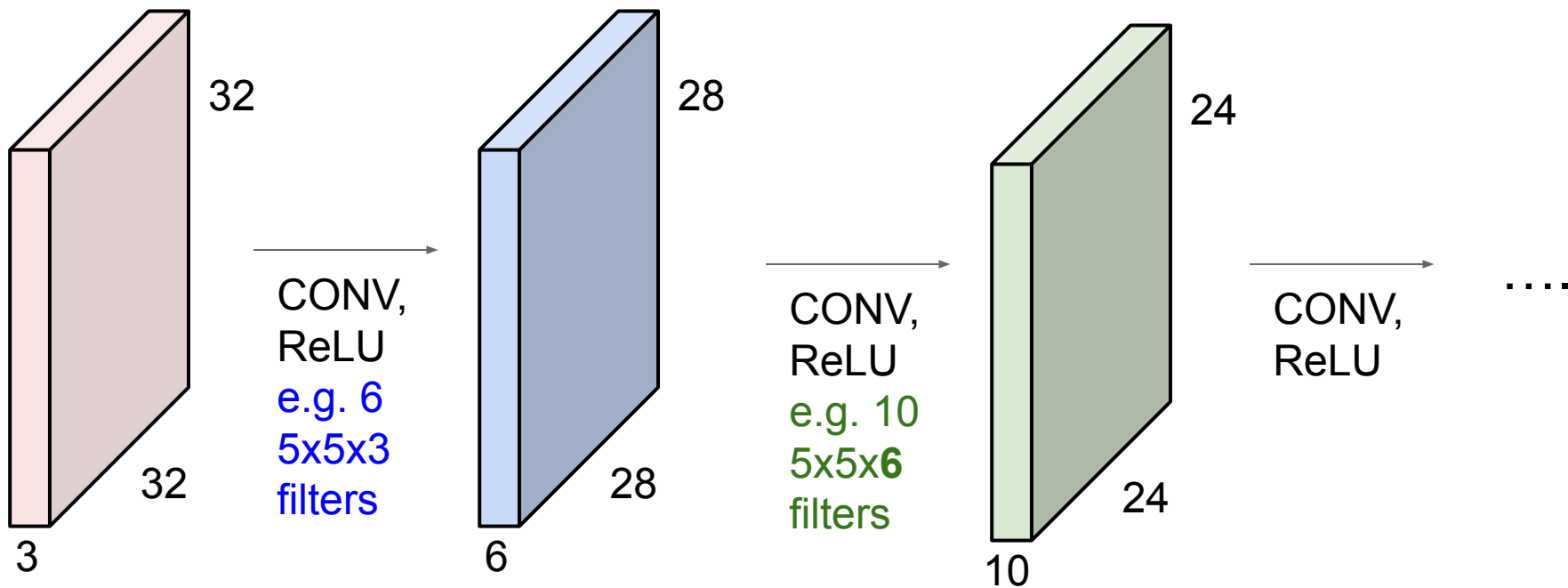


We stack these up to get a “new image” of size 28x28x6!

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



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

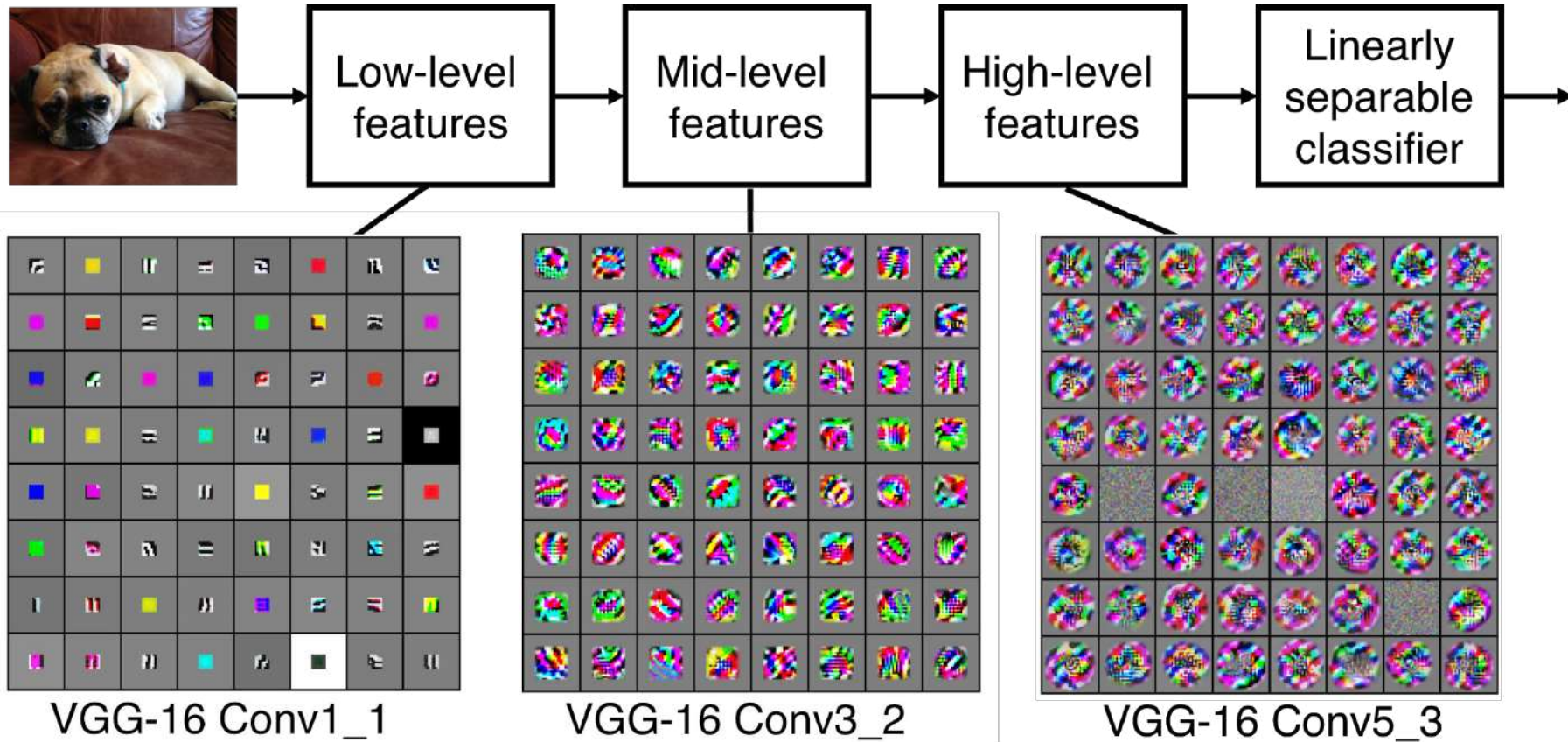




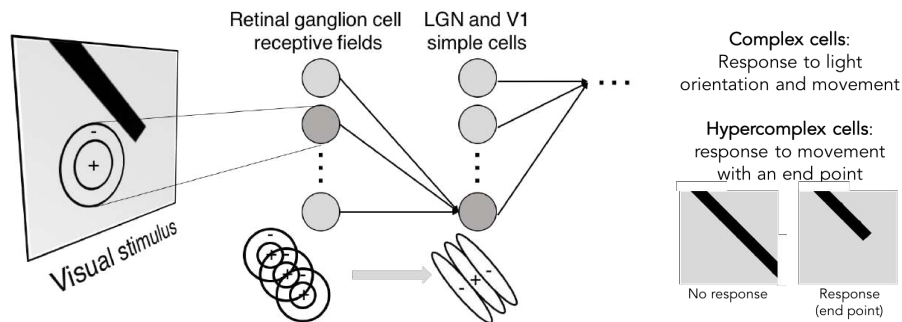
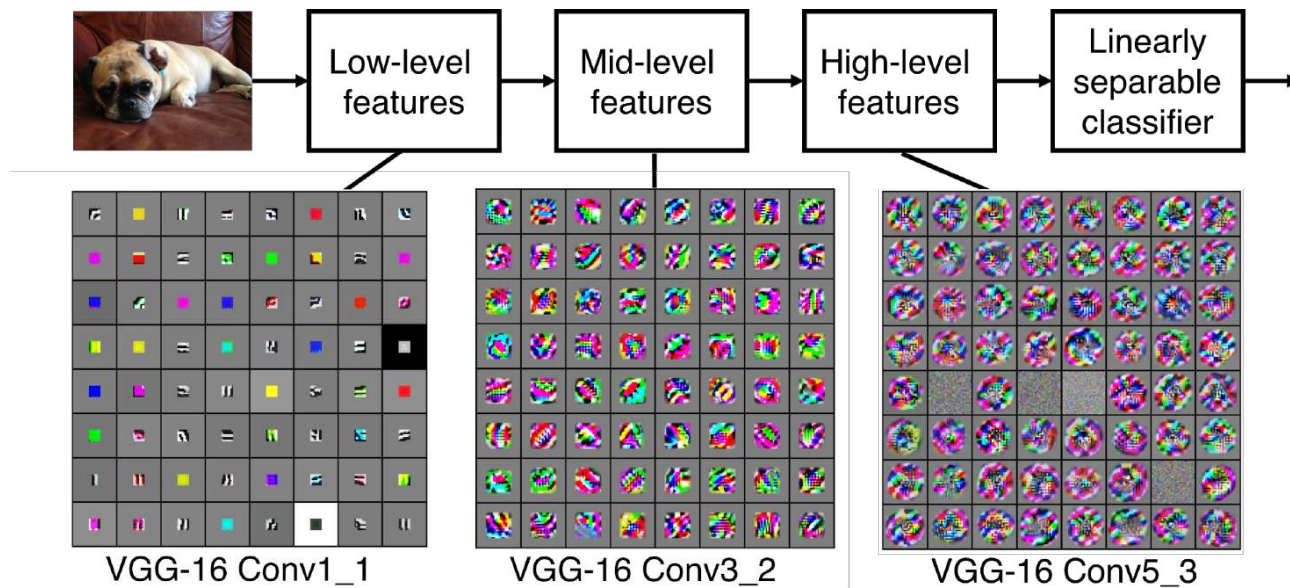
# Preview

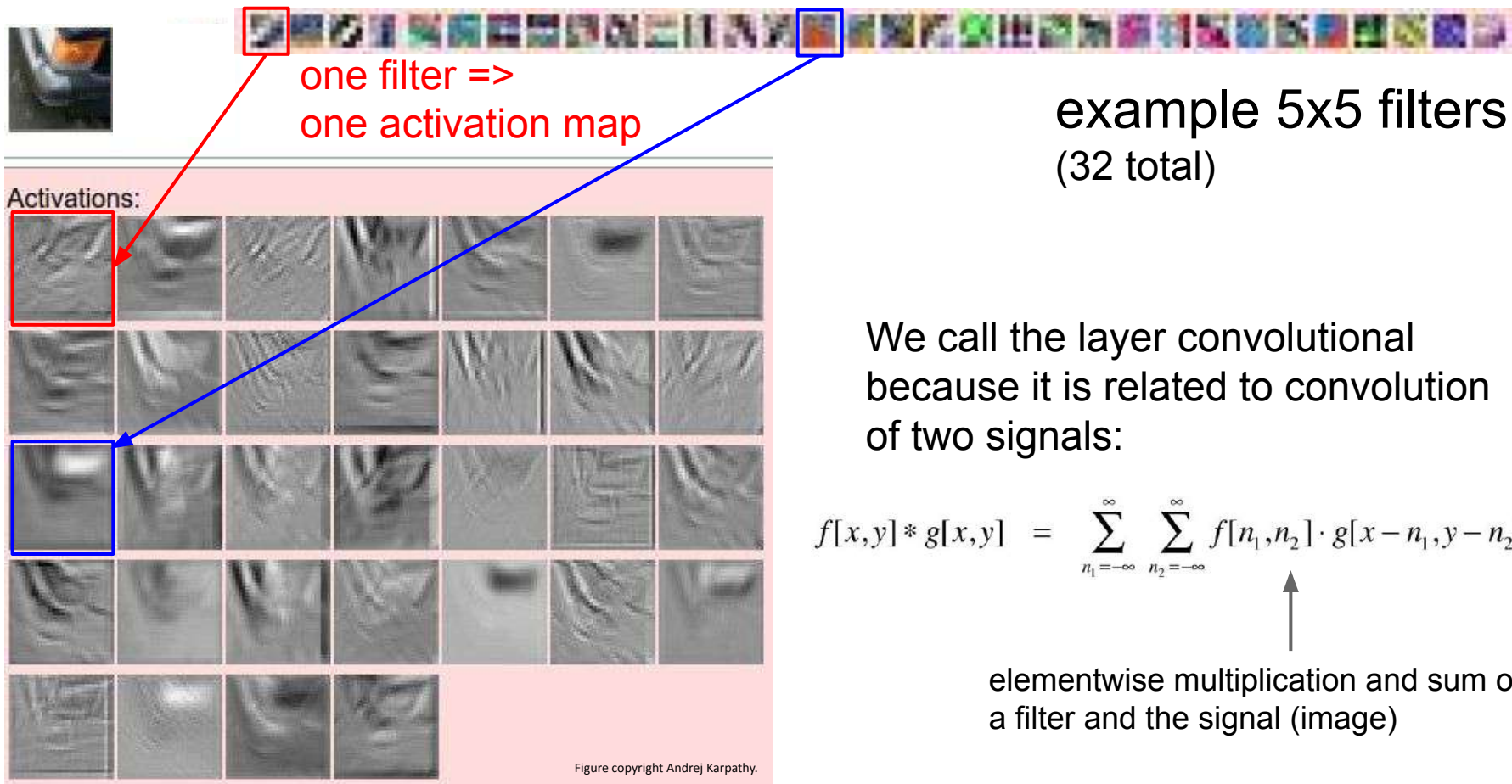
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



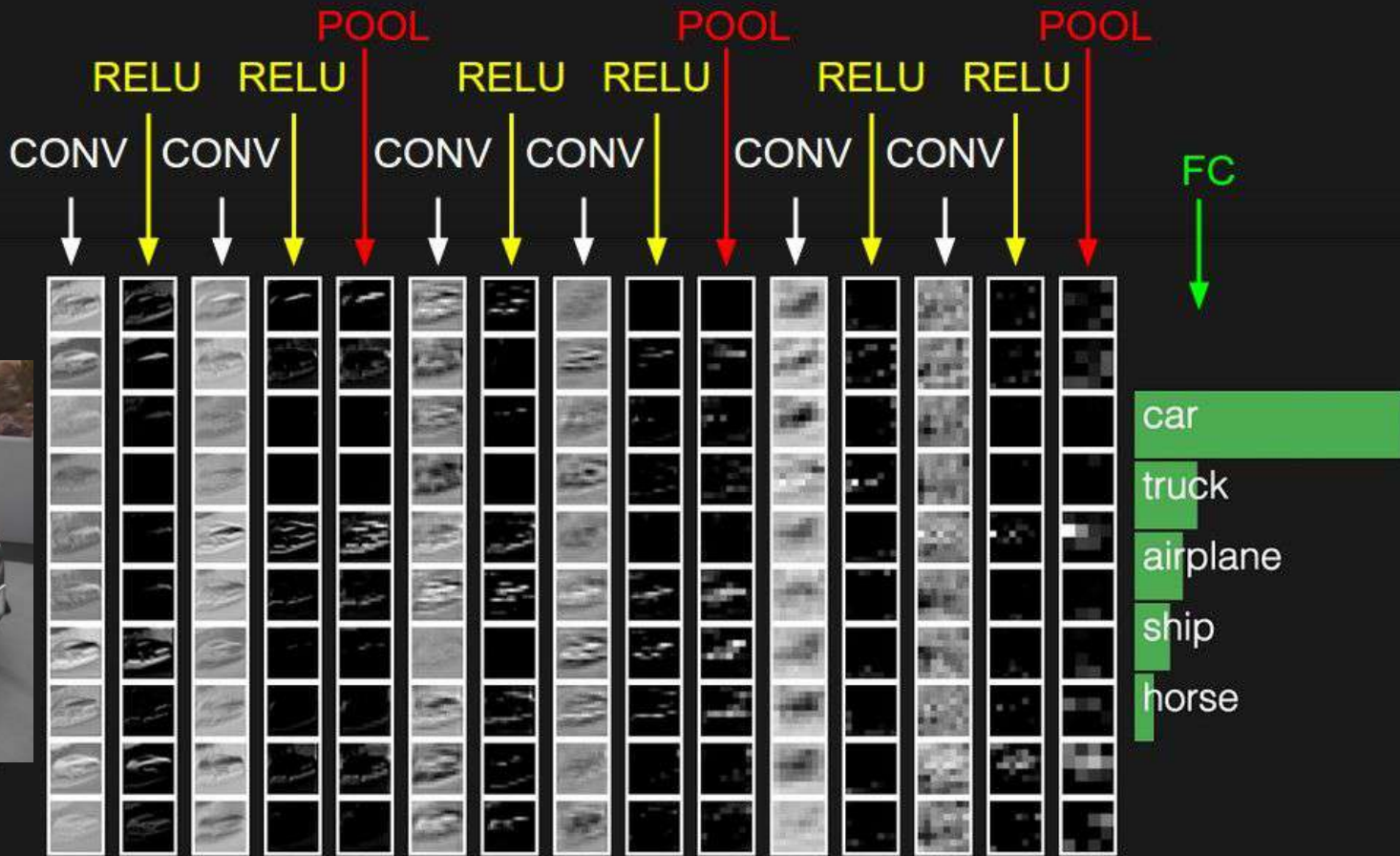
# Preview



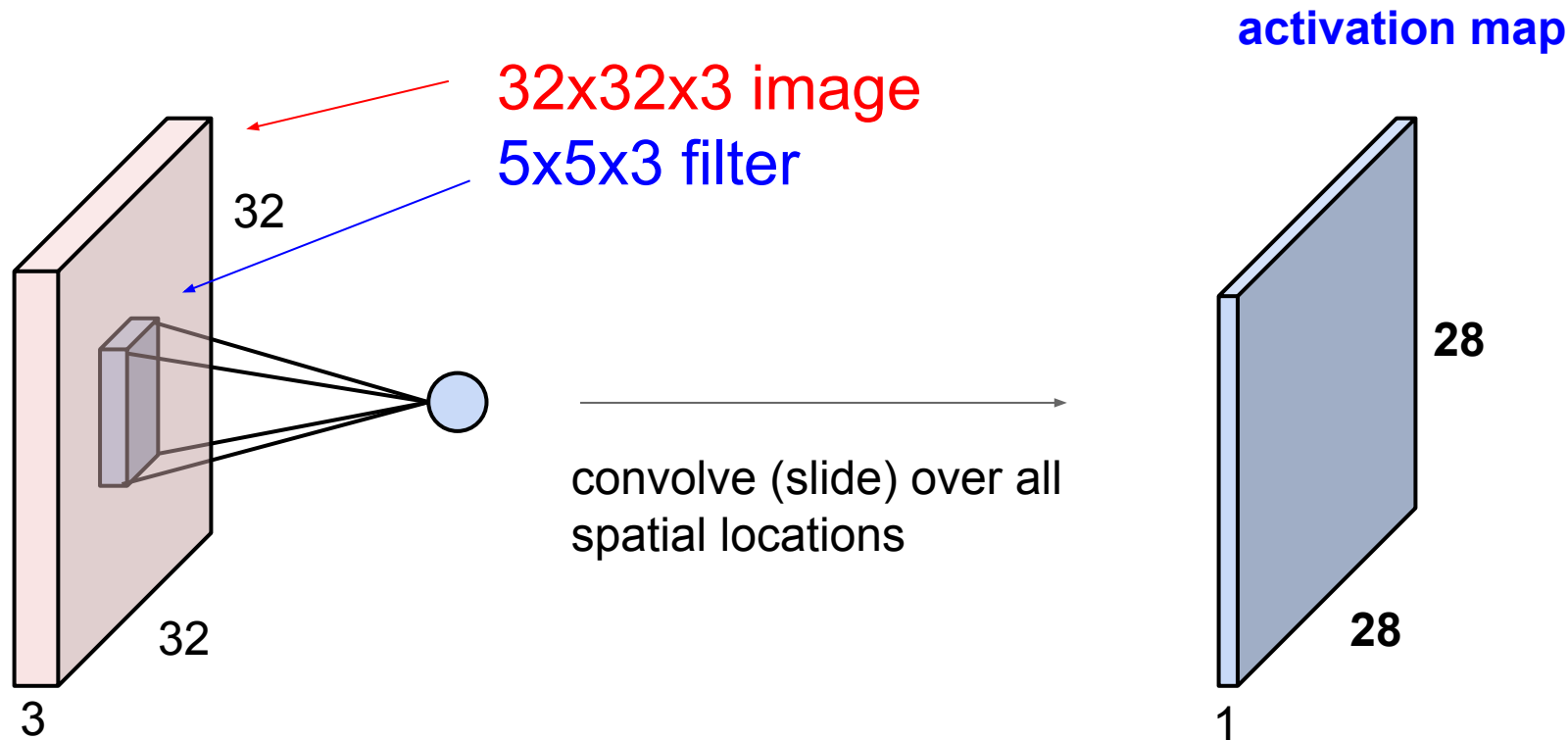




preview:



## A closer look at spatial dimensions:



## A closer look at spatial dimensions:

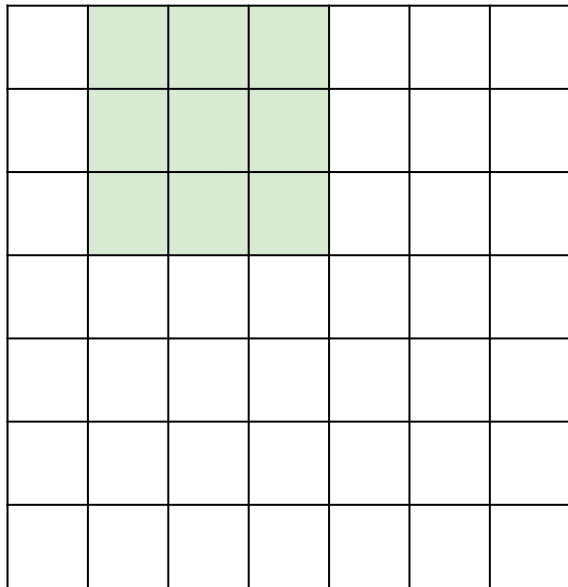
7


7

7x7 input (spatially)  
assume 3x3 filter

## A closer look at spatial dimensions:

7

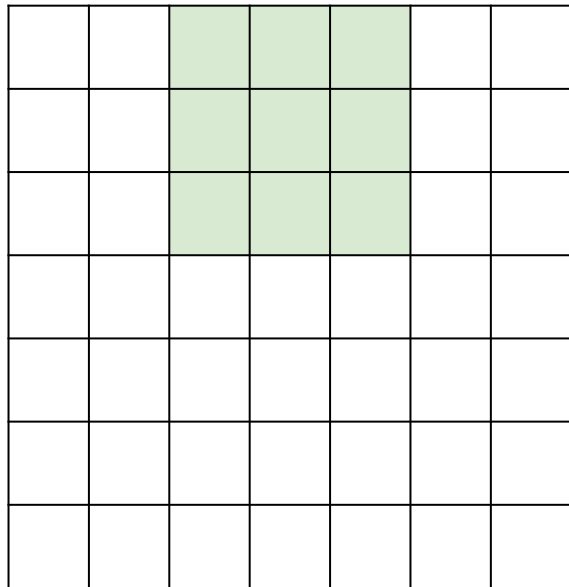


7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

7



7

7x7 input (spatially)  
assume 3x3 filter



## A closer look at spatial dimensions:

7


7

7x7 input (spatially)  
assume 3x3 filter

## A closer look at spatial dimensions:

7

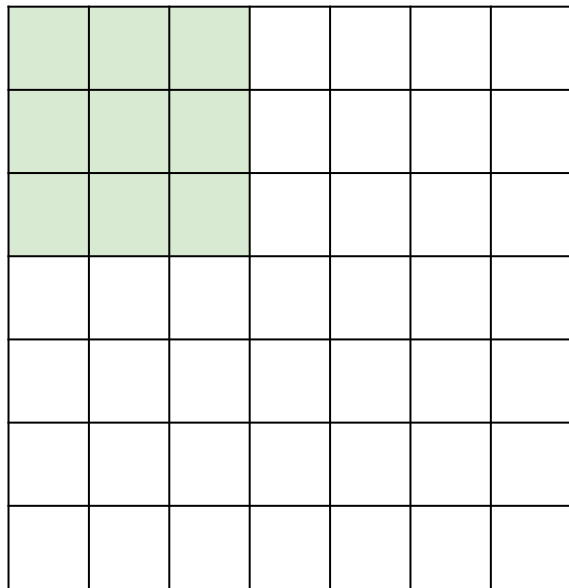

7

7x7 input (spatially)  
assume 3x3 filter

**=> 5x5 output**

## A closer look at spatial dimensions:

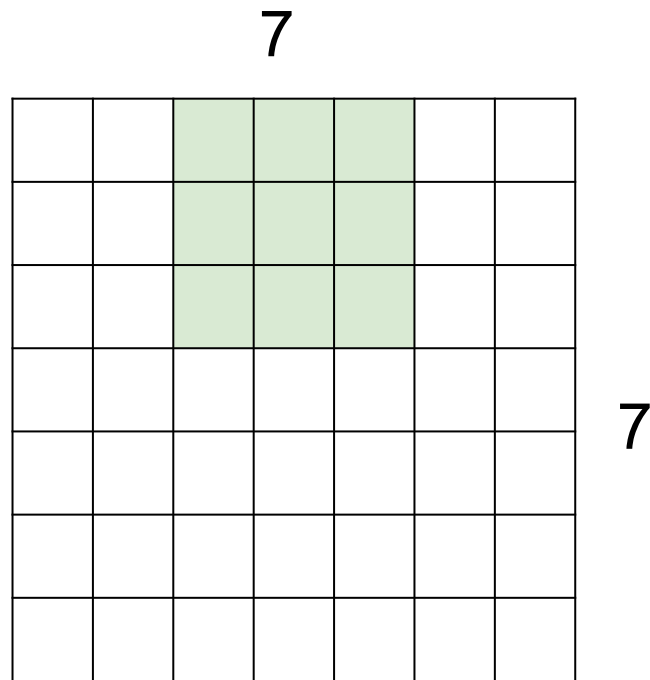
7



7

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

## A closer look at spatial dimensions:



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

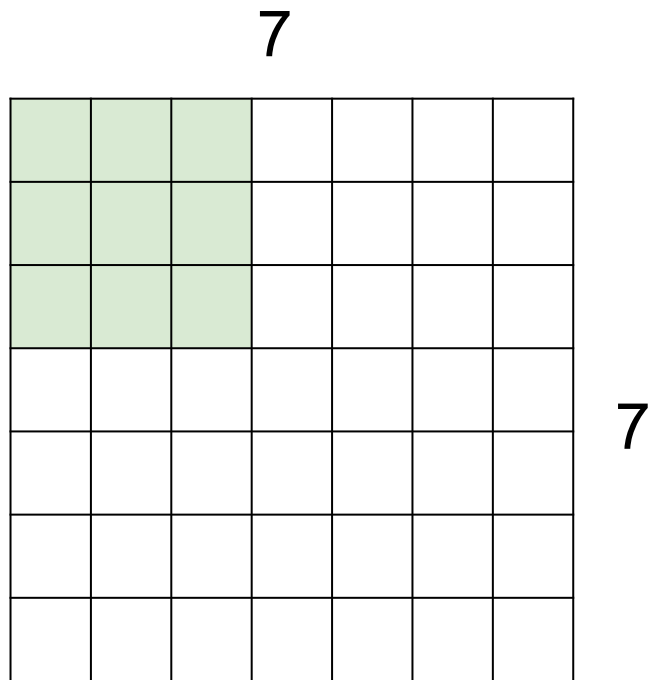
## A closer look at spatial dimensions:

7


7

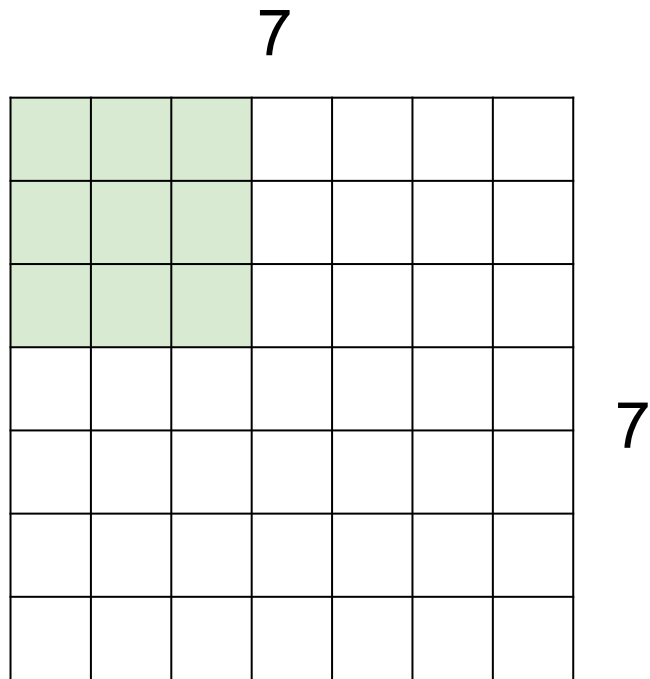
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**  
**=> 3x3 output!**

A closer look at spatial dimensions:



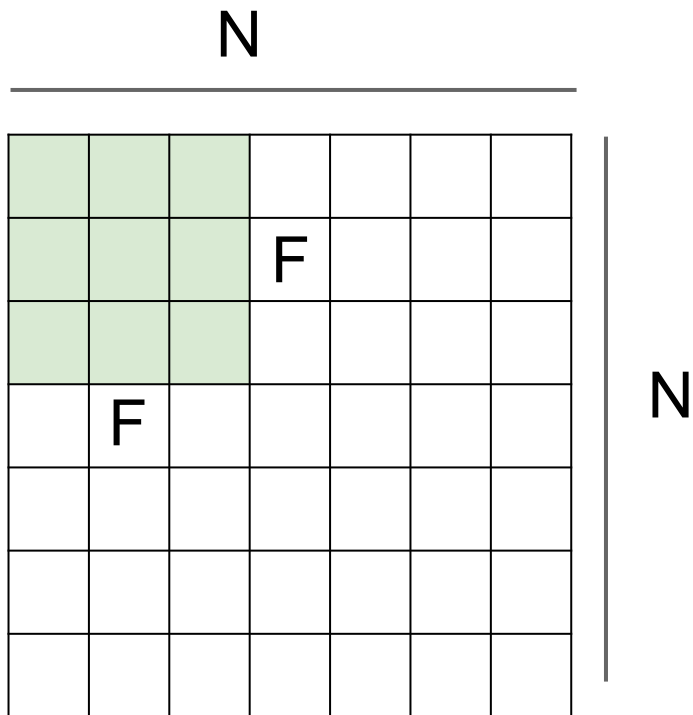
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 3?**

## A closer look at spatial dimensions:



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

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



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

e.g.  $N = 7, F = 3$ :

stride 1  $\Rightarrow (7 - 3) / 1 + 1 = 5$

stride 2  $\Rightarrow (7 - 3) / 2 + 1 = 3$

stride 3  $\Rightarrow (7 - 3) / 3 + 1 = 2.33 \therefore \backslash$



# 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:)

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

# 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!**

(recall:)

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

# 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  $F \times F$ , and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

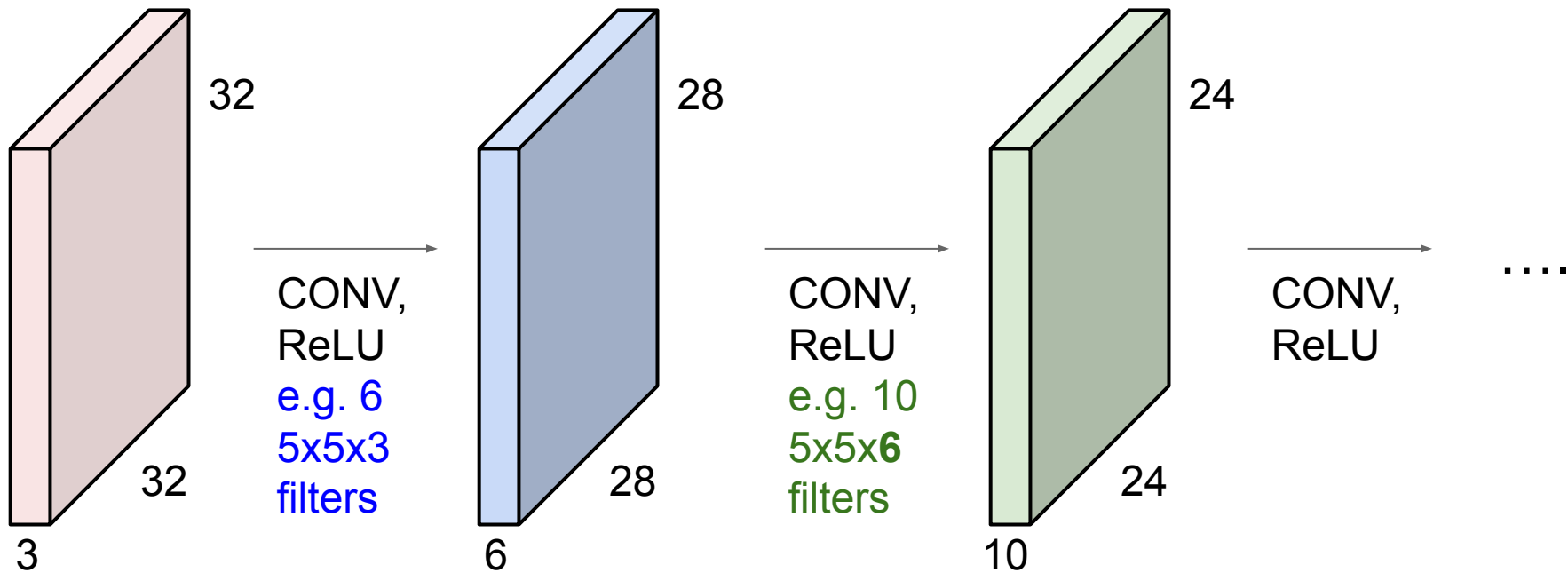
e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

## Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32  $\rightarrow$  28  $\rightarrow$  24 ...). Shrinking too fast is not good, doesn't work well.

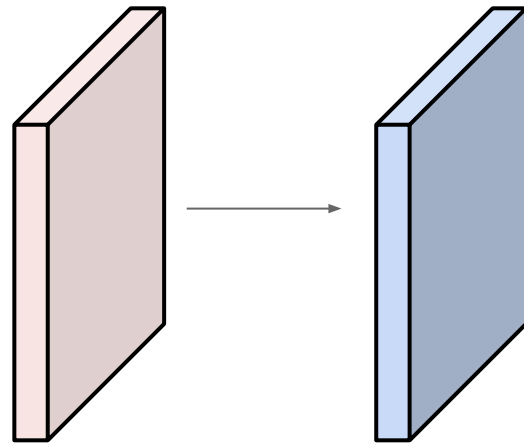


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:

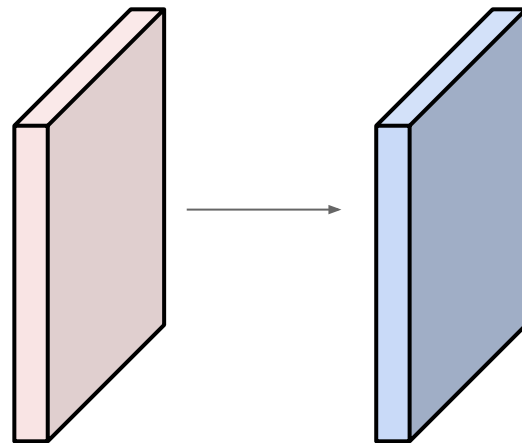
Input volume: **32x32x3**

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

Output volume size:

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

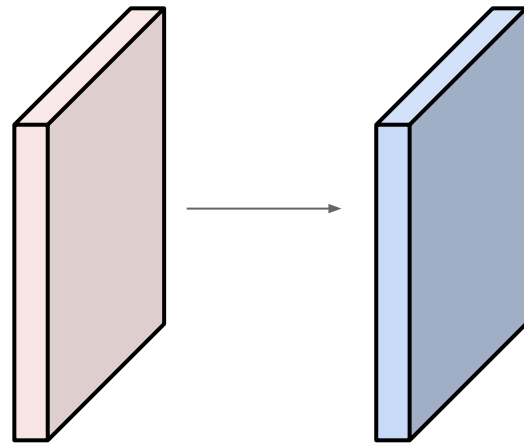
**32x32x10**



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

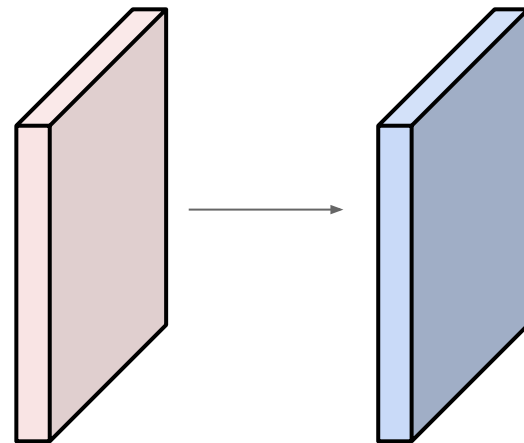


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

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



Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params (+1 for bias)

=>  $76*10 = 760$



# Convolution layer: summary

Let's assume input is  $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of  $W_2 \times H_2 \times K$   
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters:  $F^2CK$  and  $K$  biases

# Convolution layer: summary

Common settings:

Let's assume input is  $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

**K** = (powers of 2, e.g. 32, 64, 128, 512)

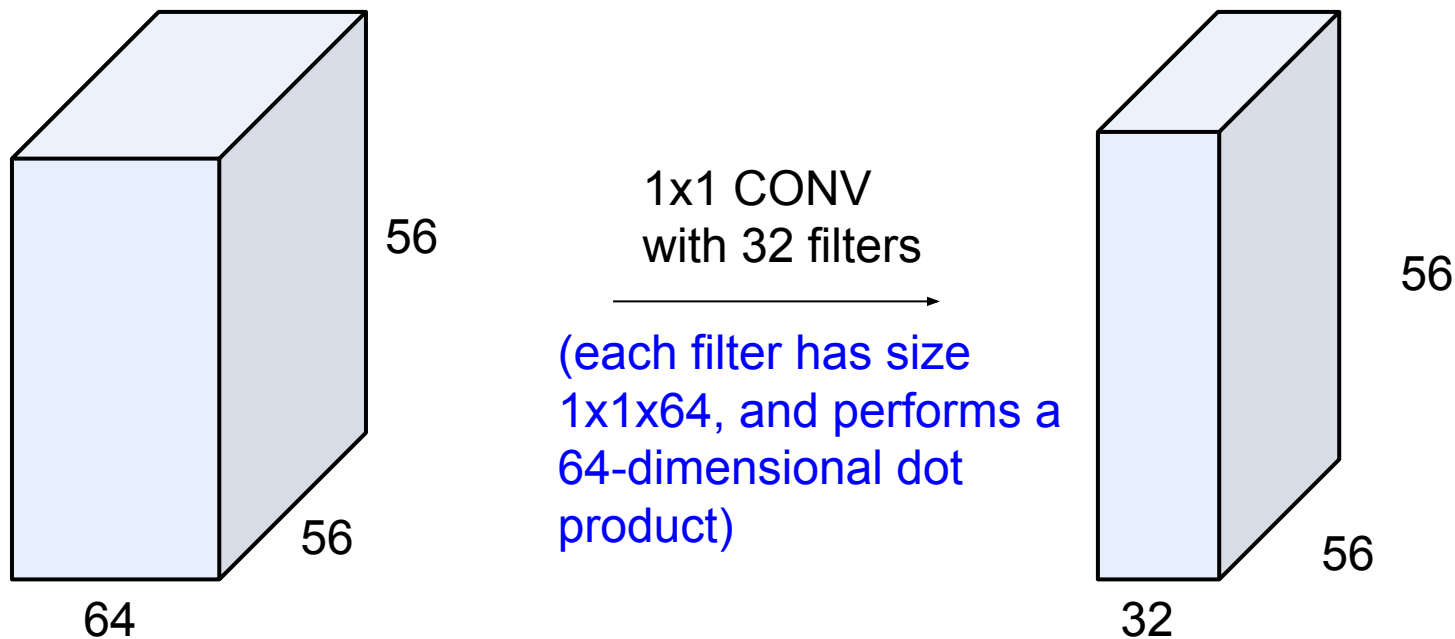
- **F** = 3, **S** = 1, **P** = 1
- **F** = 5, **S** = 1, **P** = 2
- **F** = 5, **S** = 2, **P** = ? (whatever fits)
- **F** = 1, **S** = 1, **P** = 0

This will produce an output of  $W_2 \times H_2 \times K$   
where:

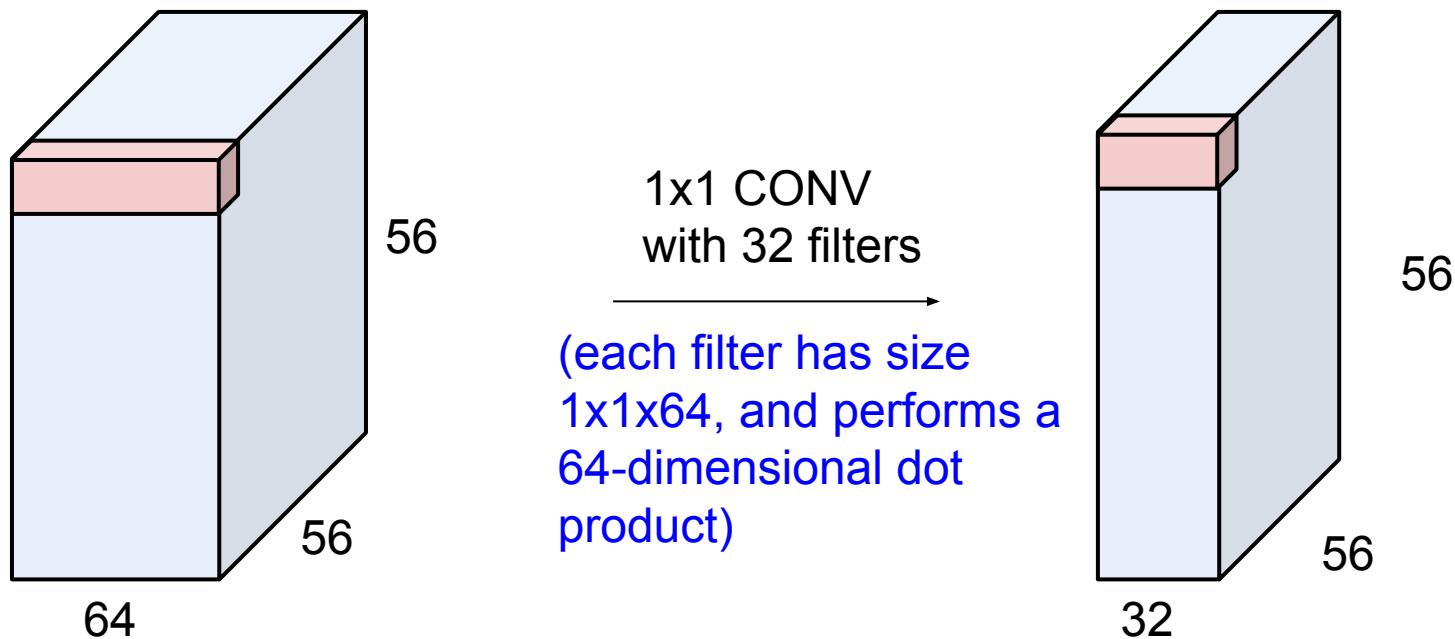
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters:  $F^2CK$  and  $K$  biases

(btw, 1x1 convolution layers make perfect sense)



(btw, 1x1 convolution layers make perfect sense)



# Example: CONV layer in PyTorch

## Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
```

[SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C_{in}, H, W)$  and output  $(N, C_{out}, H_{out}, W_{out})$  can be precisely described as:

$$\text{out}(N, C_{out_i}) = \text{bias}(C_{out_i}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_i}, k) \star \text{input}(N, k)$$

where  $\star$  is the valid 2D **cross-correlation** operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but [this link](#) has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At `groups=1`, all inputs are convolved to all outputs.
  - At `groups=2`, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At `groups= in_channels`, each input channel is convolved with its own set of filters, of size:  $\begin{bmatrix} C_{out} \\ C_{in} \end{bmatrix}$ .

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

PyTorch is licensed under [BSD 3-clause](#).

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

# Example: CONV layer in Keras

## Conv2D

[\[source\]](#)

```
keras.layers.Conv2D(filters, kernel_size, strides=[1, 1], padding='valid', data_format=None, d
```

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the batch axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

### Arguments

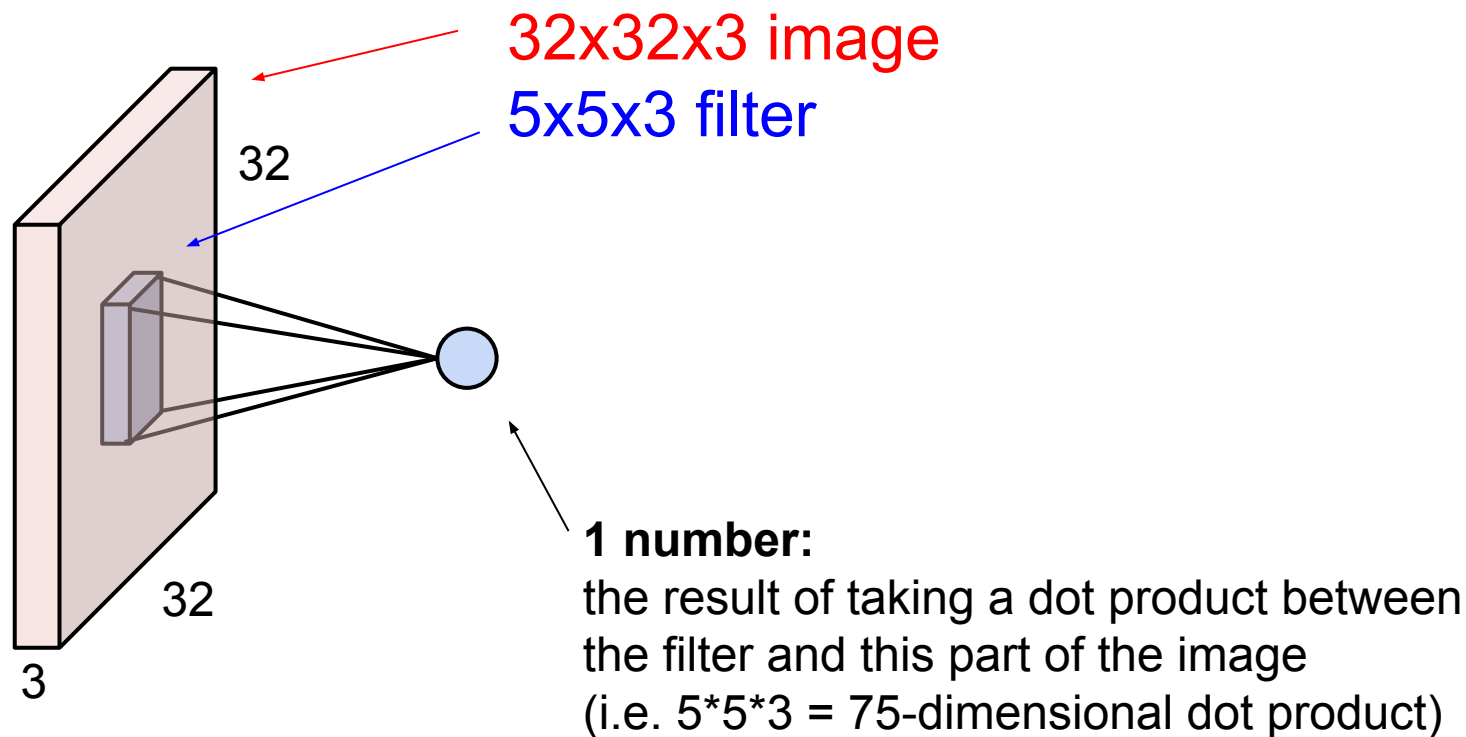
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel\_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value  $\neq 1$  is incompatible with specifying any `dilation_rate` value  $\neq 1$ .
- **padding**: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with `strides`  $\neq 1$ , as described [here](#).
- **data\_format**: A string, one of "channels\_last" or "channels\_first". The ordering of the dimensions in the inputs. "channels\_last" corresponds to inputs with shape (batch, height, width, channels) while "channels\_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels\_last".

[Keras](#) is licensed under the [MIT license](#).

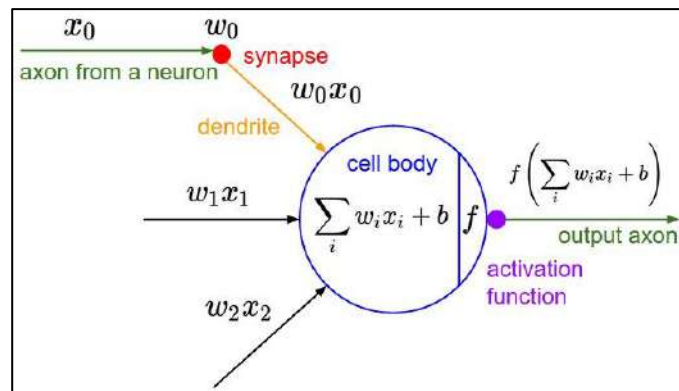
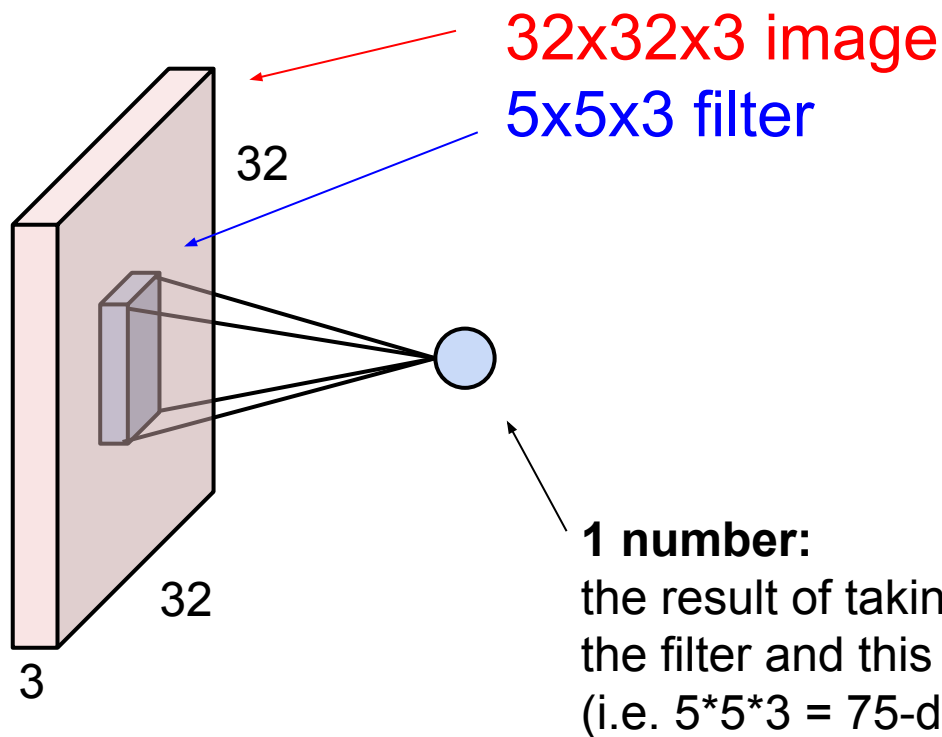
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

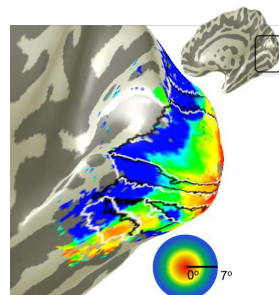
# The brain/neuron view of CONV Layer



# The brain/neuron view of CONV Layer

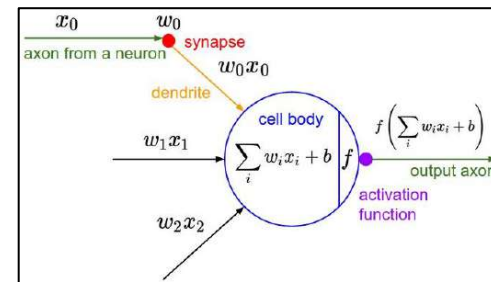
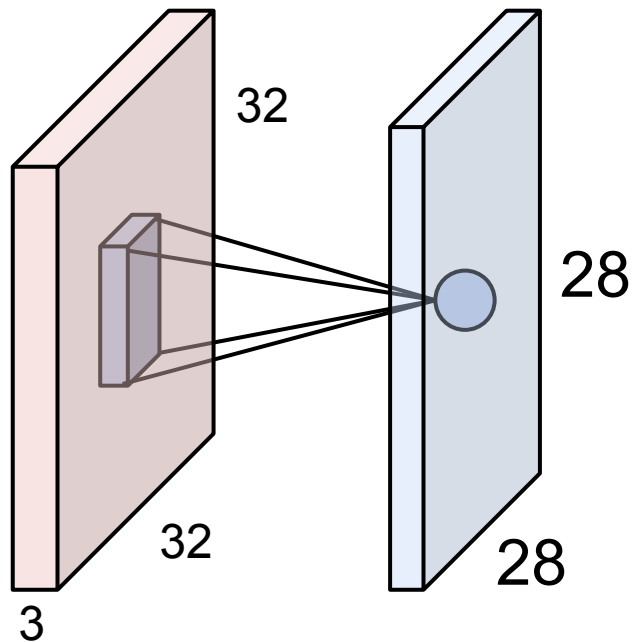


It's just a neuron with local connectivity...





# Receptive field

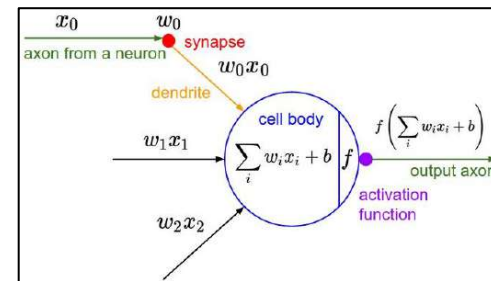
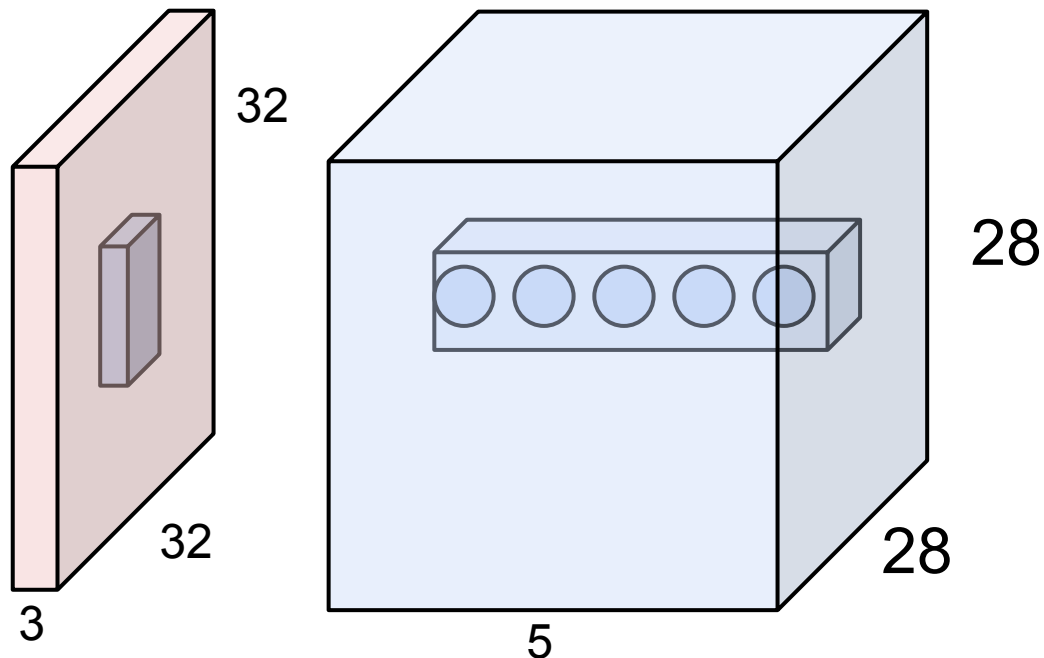


An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”

# The brain/neuron view of CONV Layer



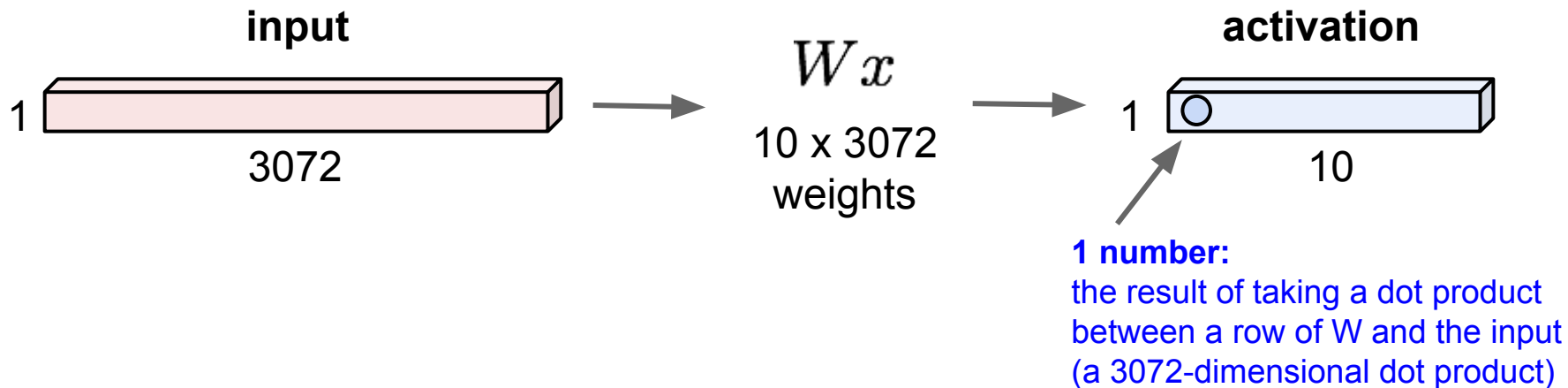
E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
(28x28x5)

There will be 5 different  
neurons all looking at the same  
region in the input volume

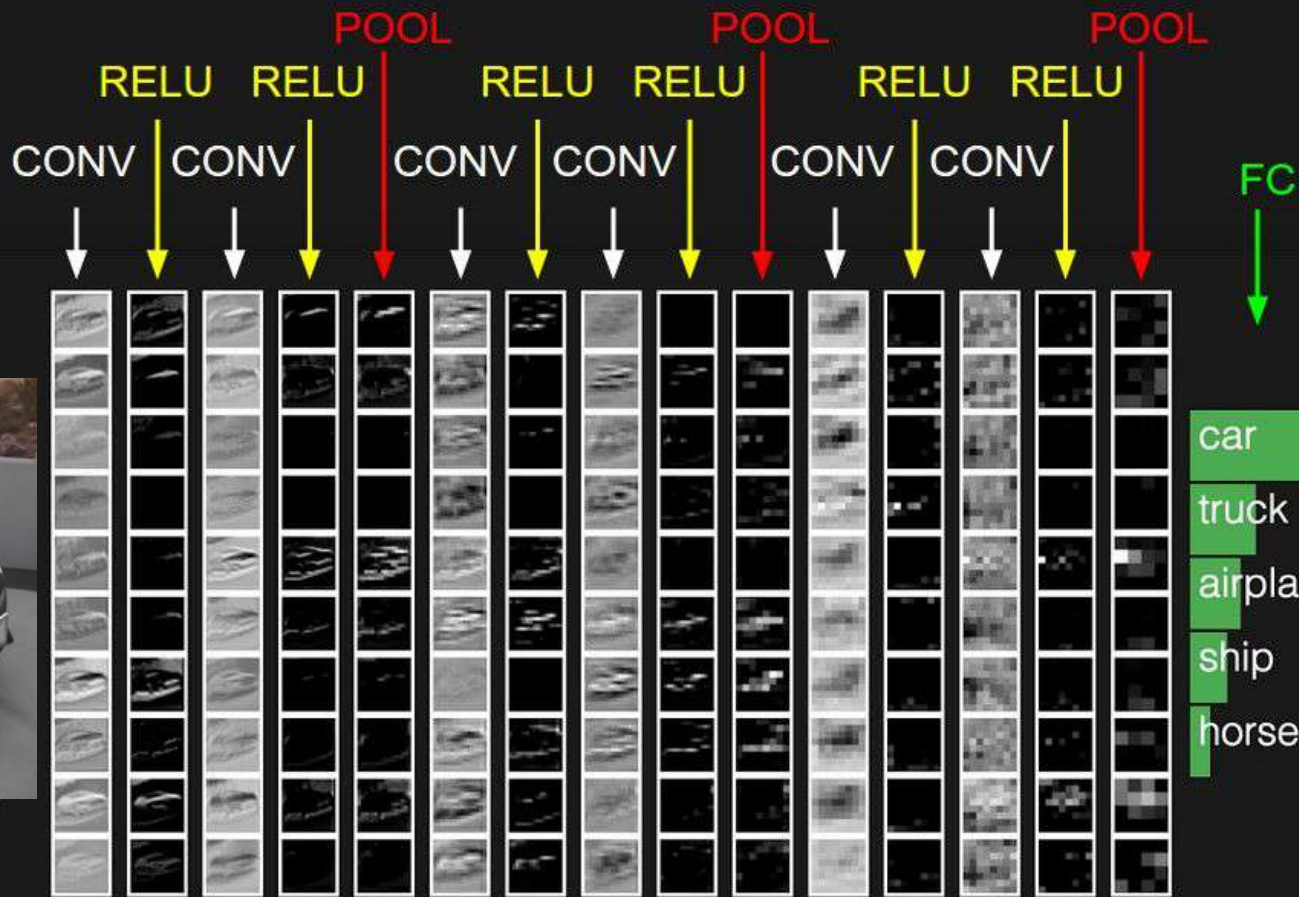
# Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron  
looks at the full  
input volume

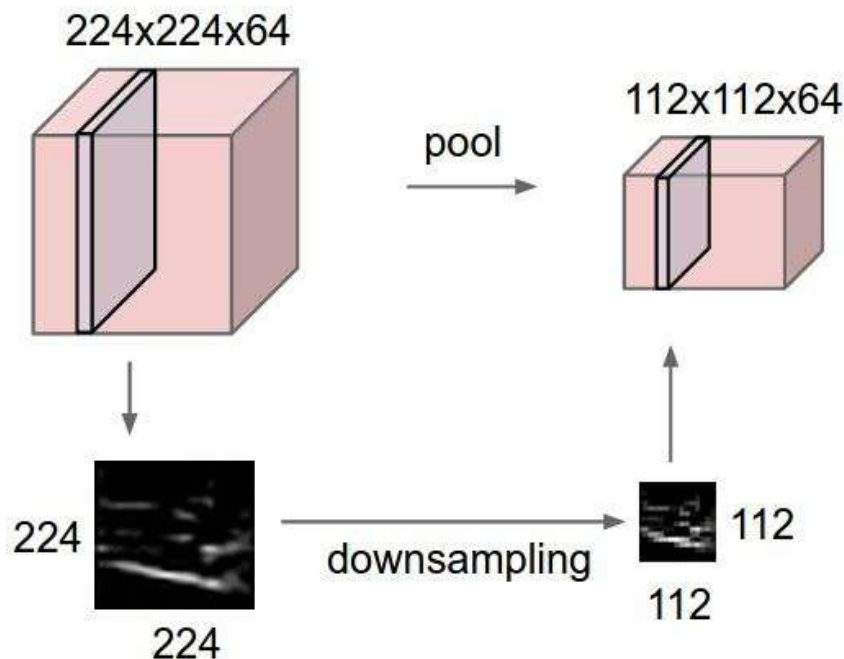


two more layers to go: POOL/FC

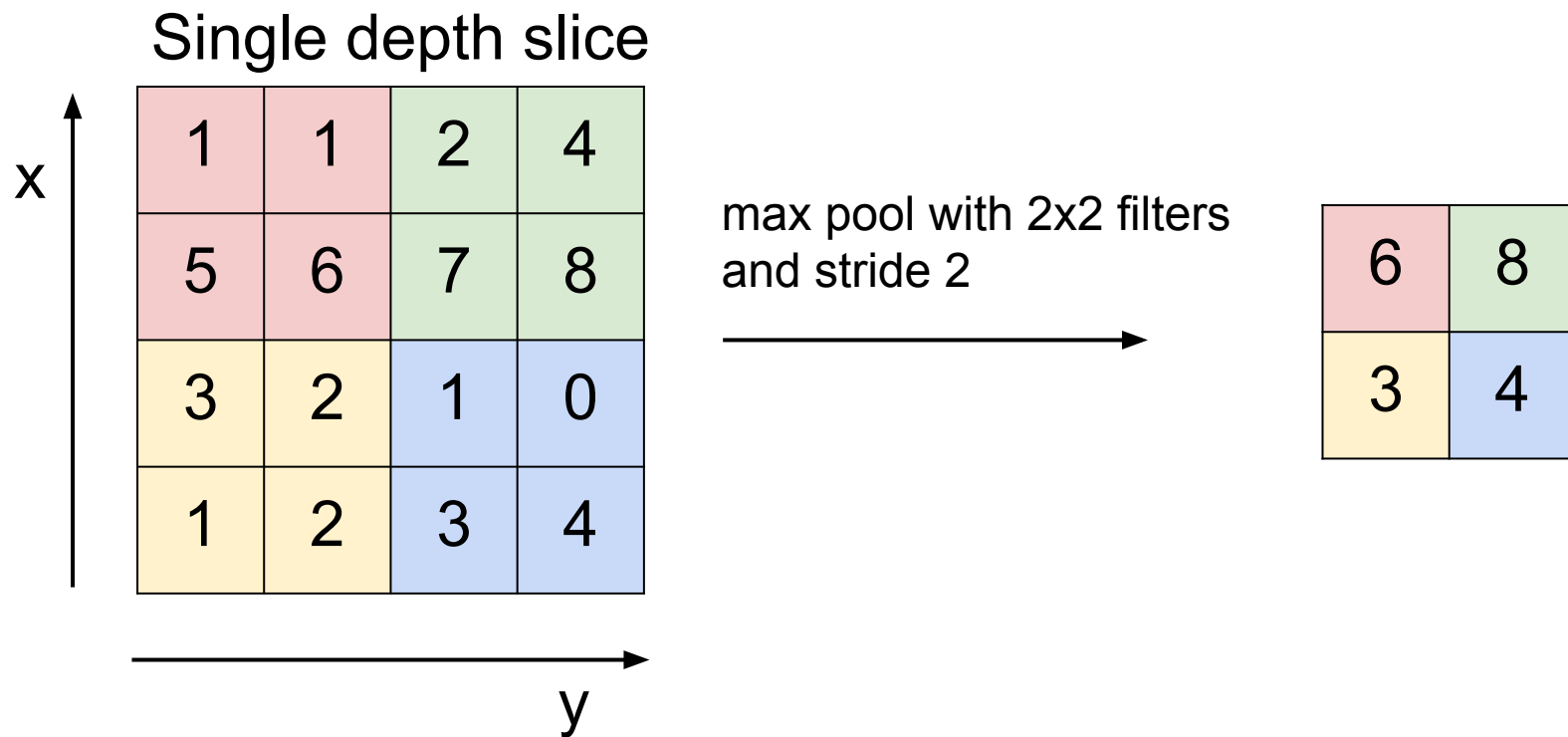


# Pooling layer

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



# MAX POOLING



# Pooling layer: summary

Let's assume input is  $W_1 \times H_1 \times C$

Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

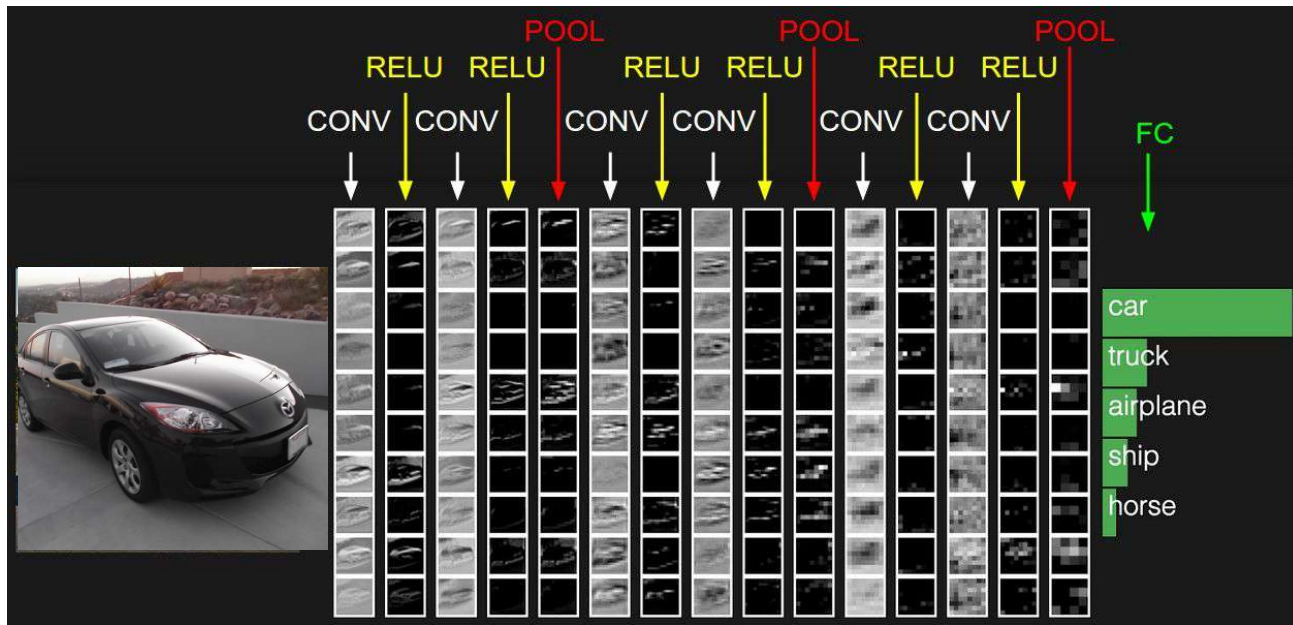
This will produce an output of  $W_2 \times H_2 \times C$  where:

- $W_2 = (W_1 - F) / S + 1$
- $H_2 = (H_1 - F) / S + 1$

Number of parameters: 0

# Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks





# [ConvNetJS demo: training on CIFAR-10]

## ConvNetJS CIFAR-10 demo

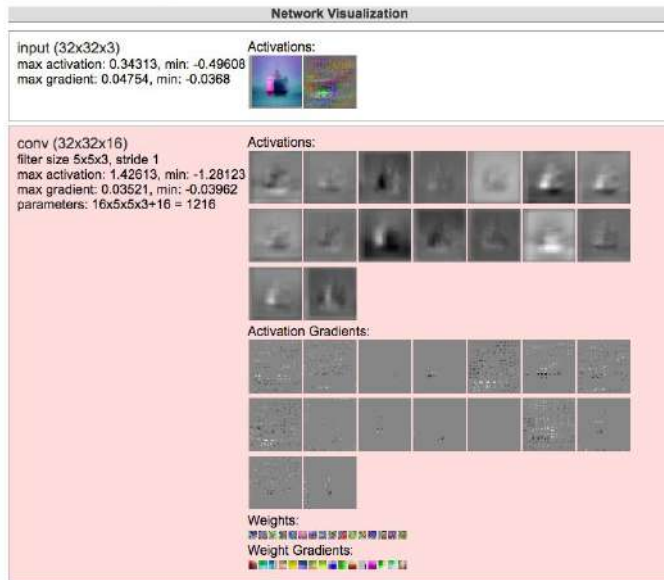
### Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).



<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

# Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like  
 **$[(\text{CONV-RELU}) * N - \text{POOL?}] * M - (\text{FC-RELU}) * K, \text{SOFTMAX}$**   
where N is usually up to ~5, M is large,  $0 \leq K \leq 2$ .
  - but recent advances such as ResNet/GoogLeNet have challenged this paradigm