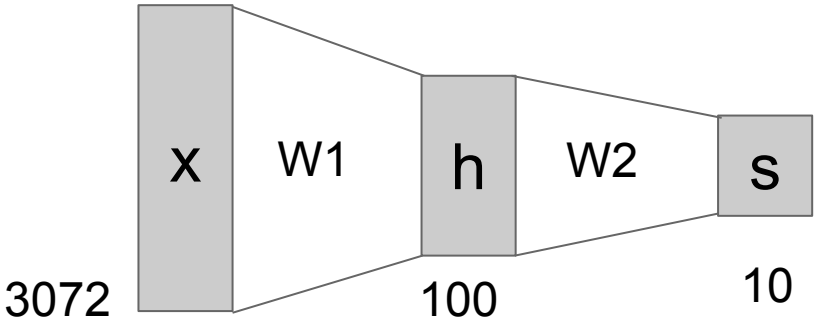
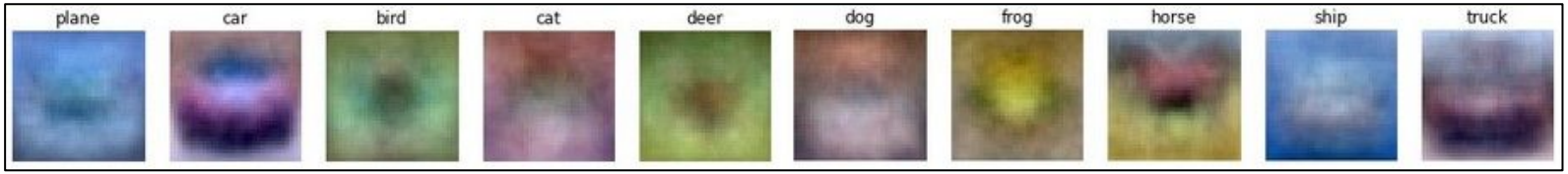
# Lecture 4: Convolutional Neural Networks: Basic operations

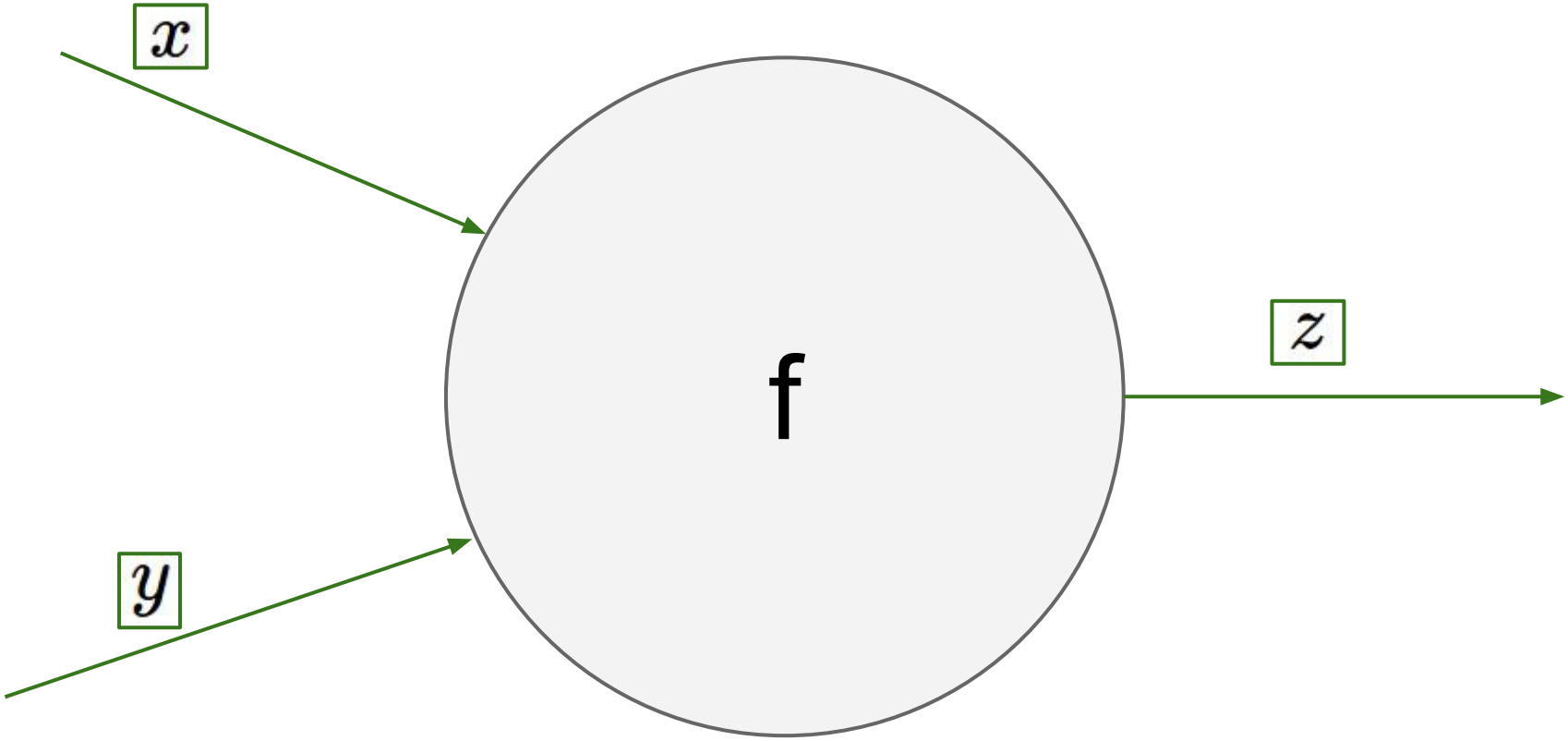
# Last time: Neural Networks

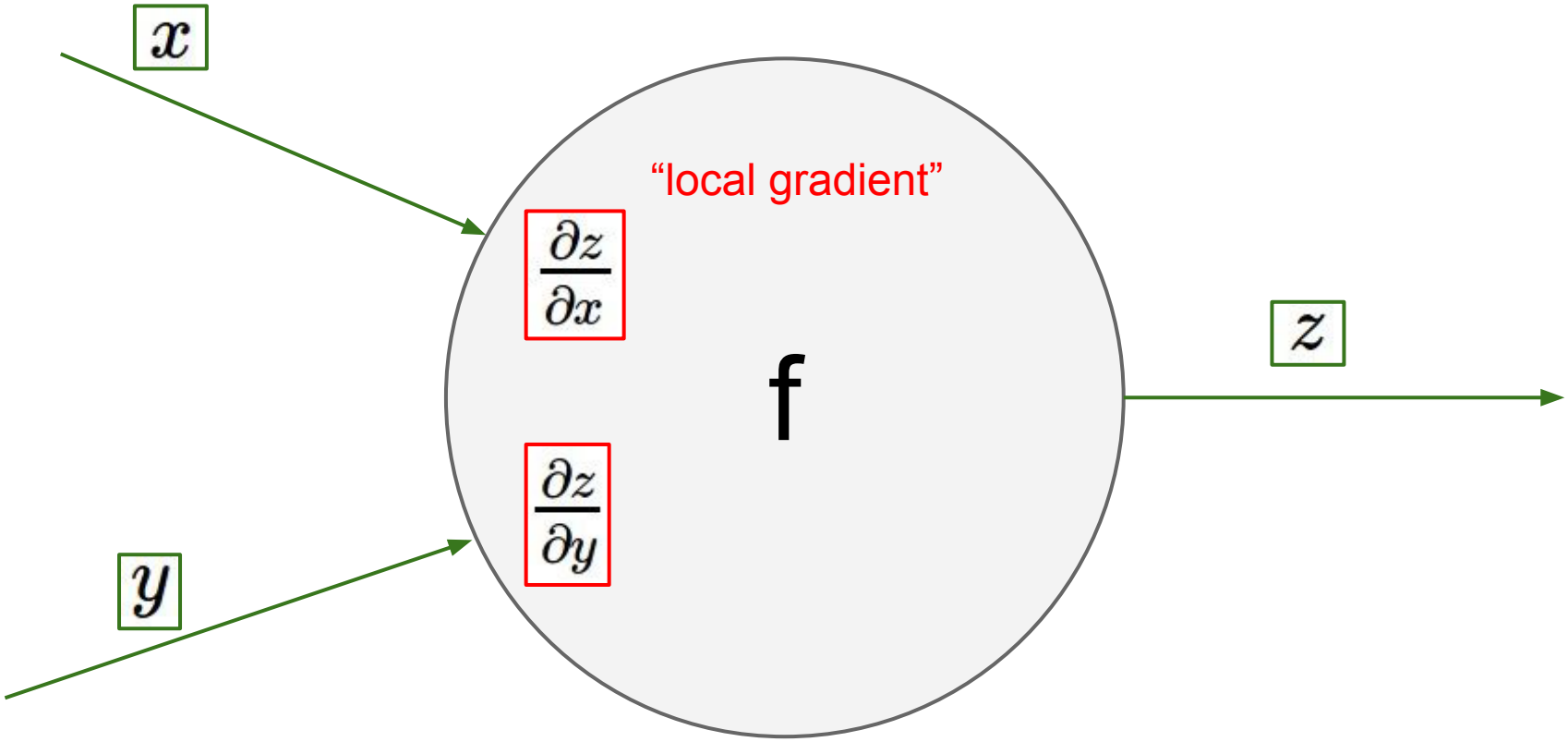
Linear score function: $f = Wx$

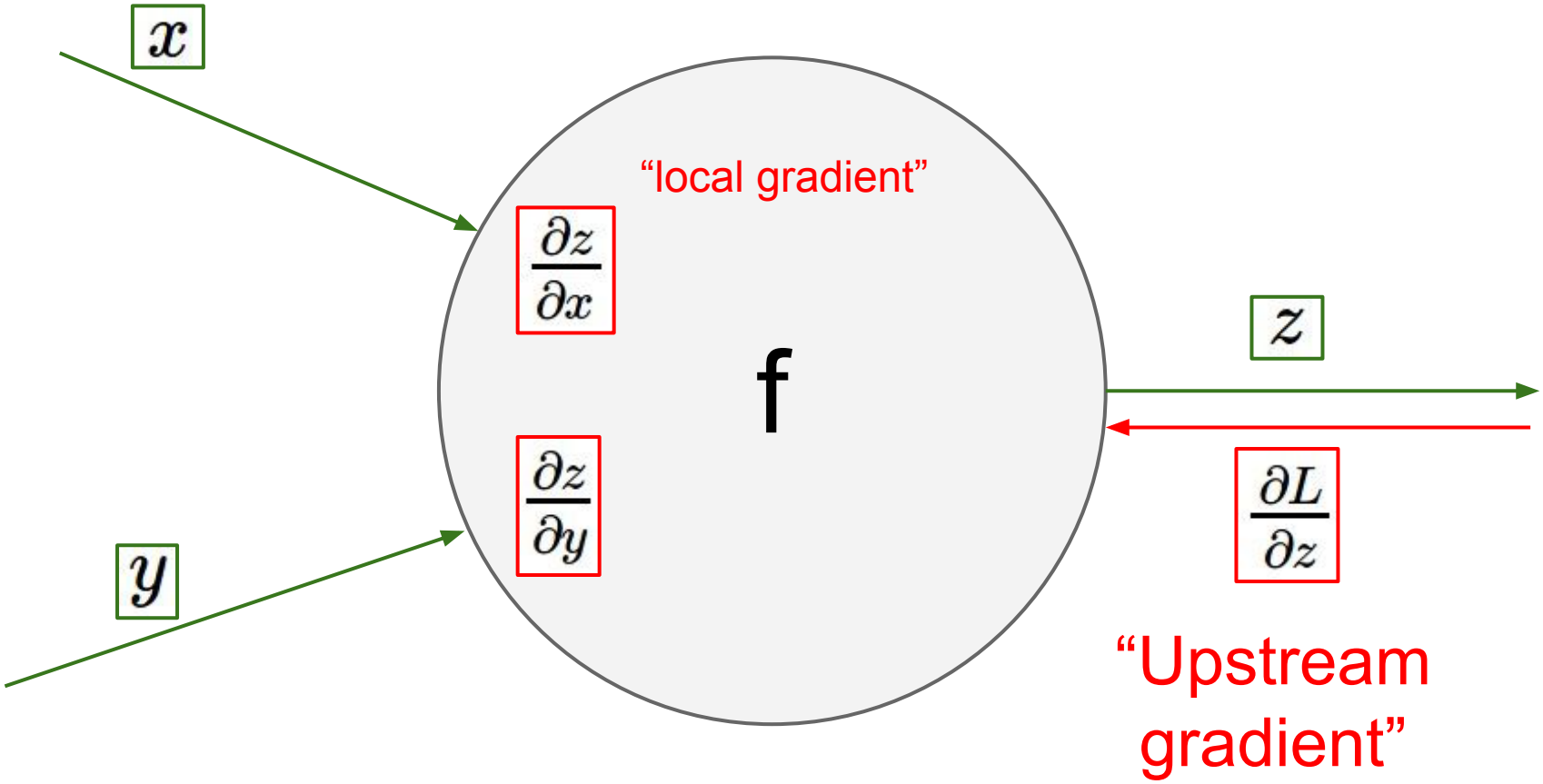
2-layer Neural Network $f = W_2 \max(0, W_1 x)$

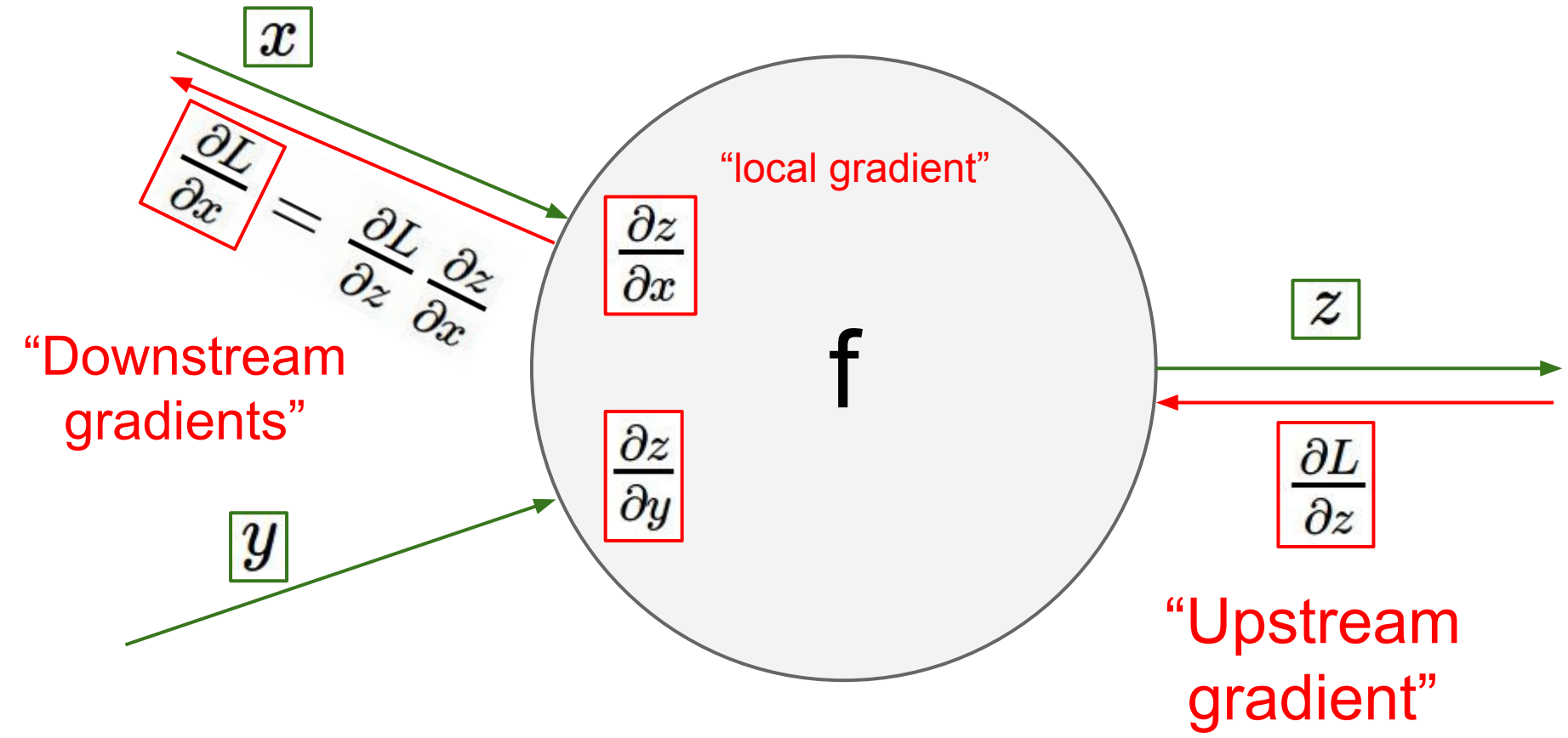


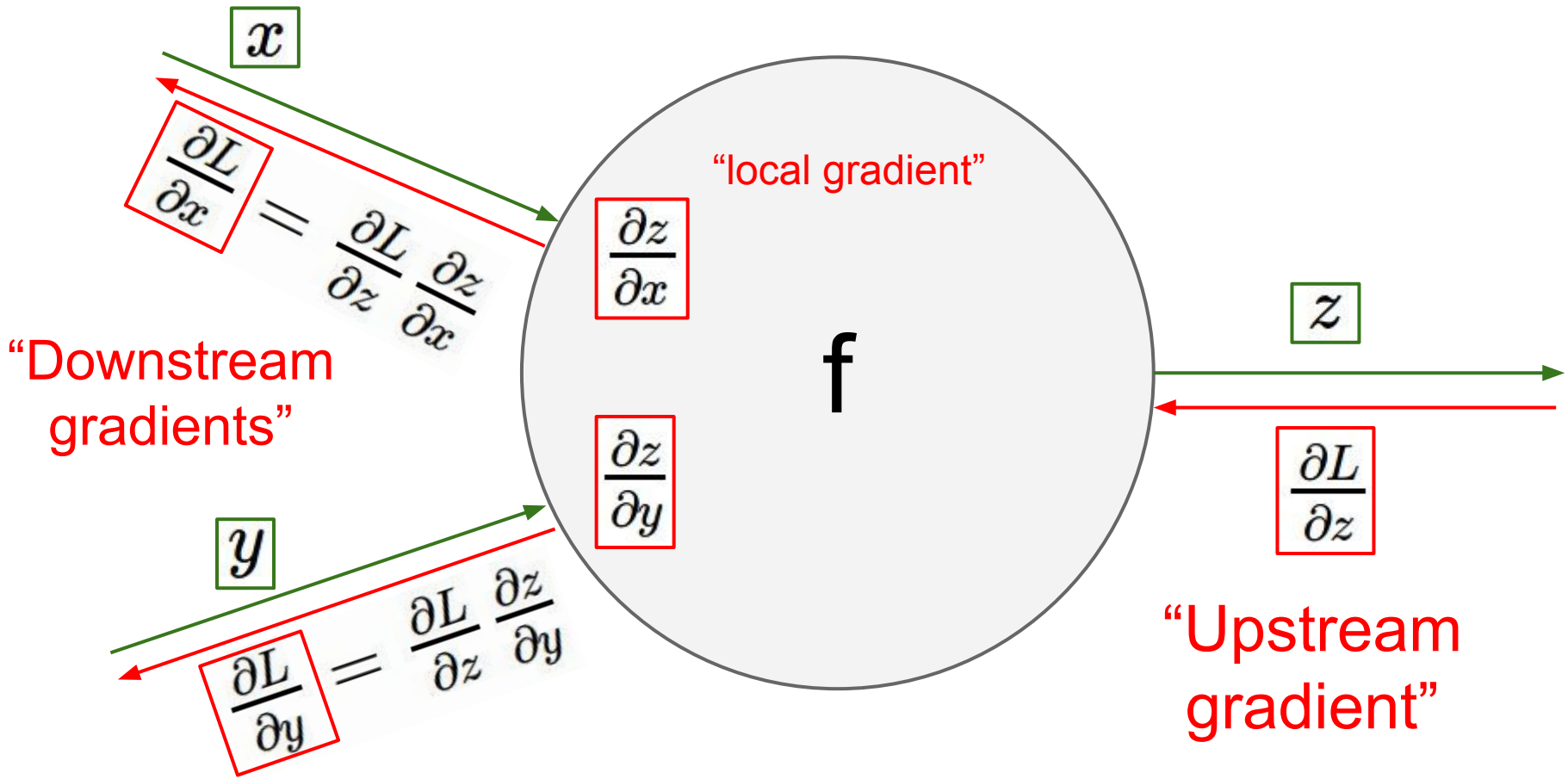



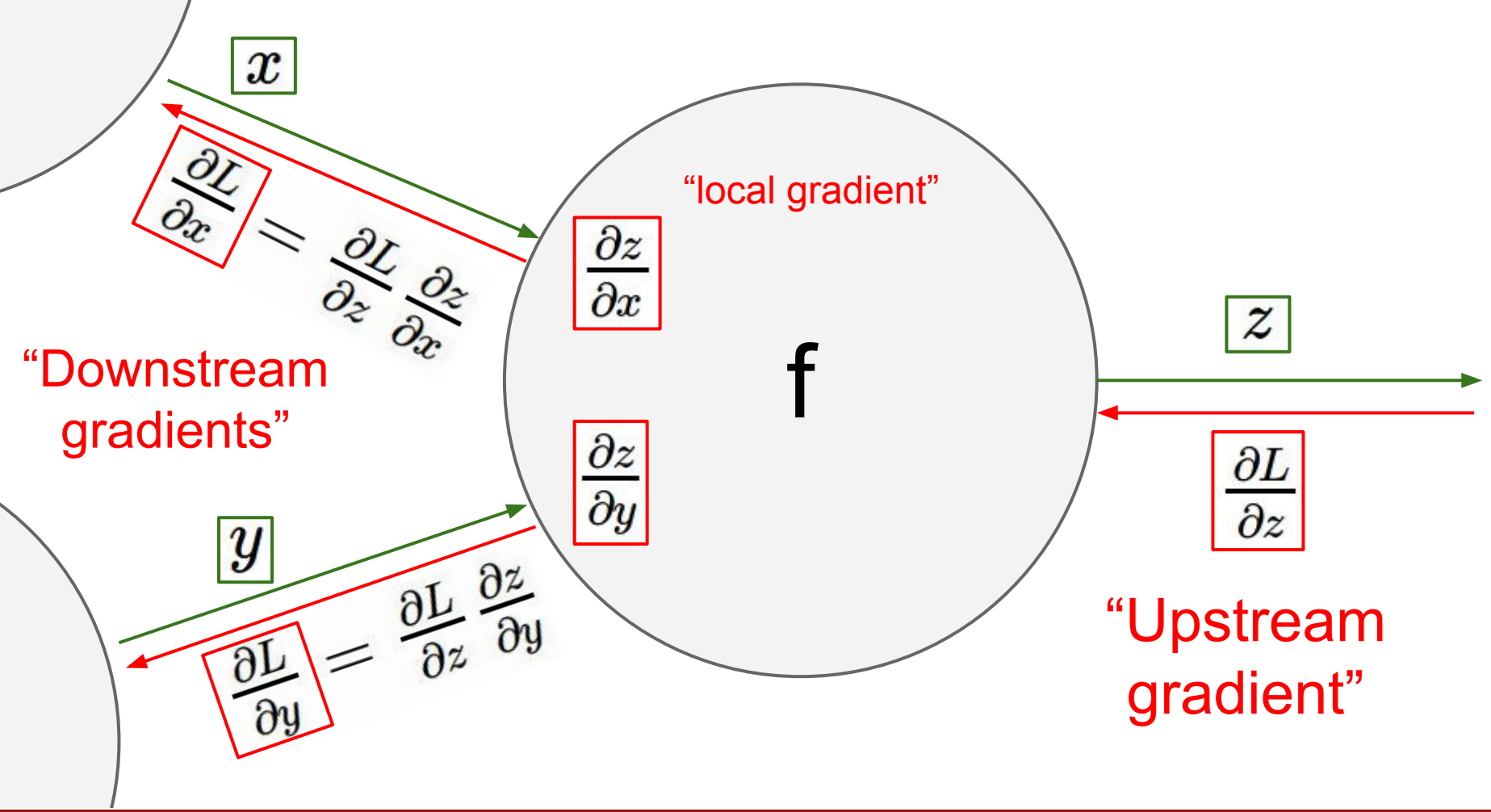

$x$
$y$
f
$z$


$x$
$y$
$z$
f
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
“local gradient”


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


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


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


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

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

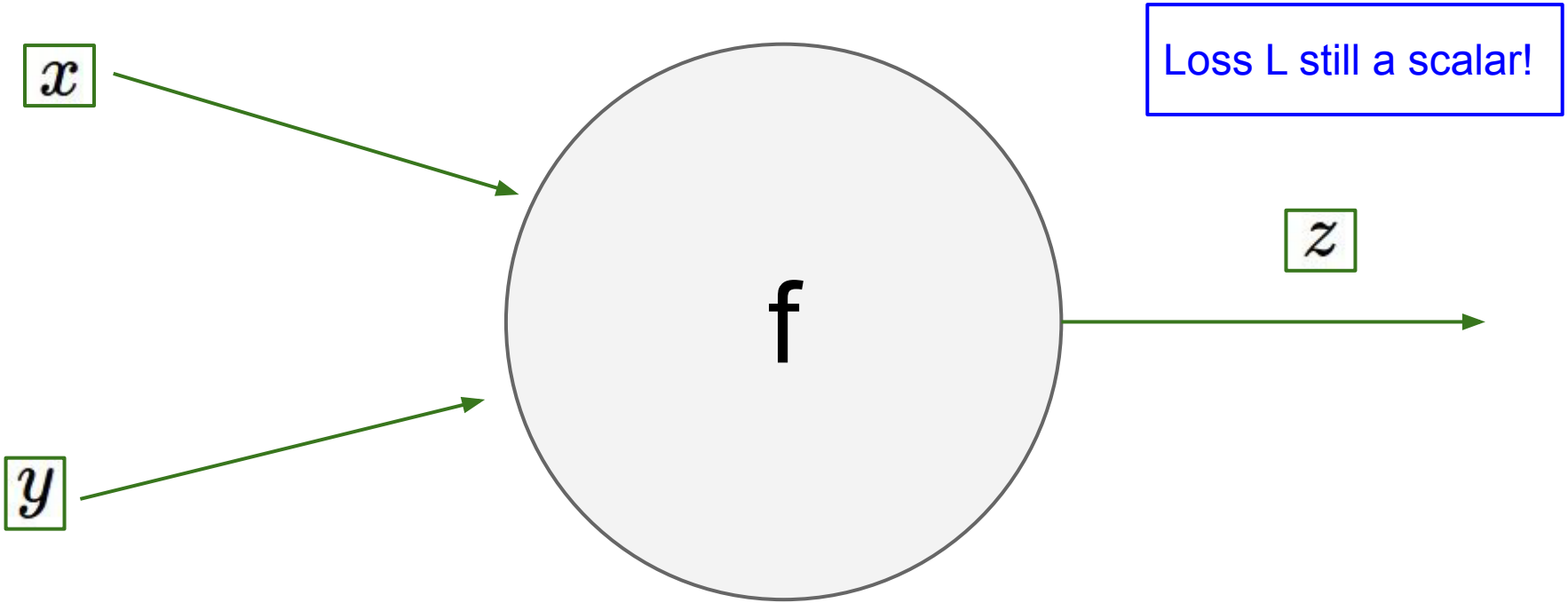
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

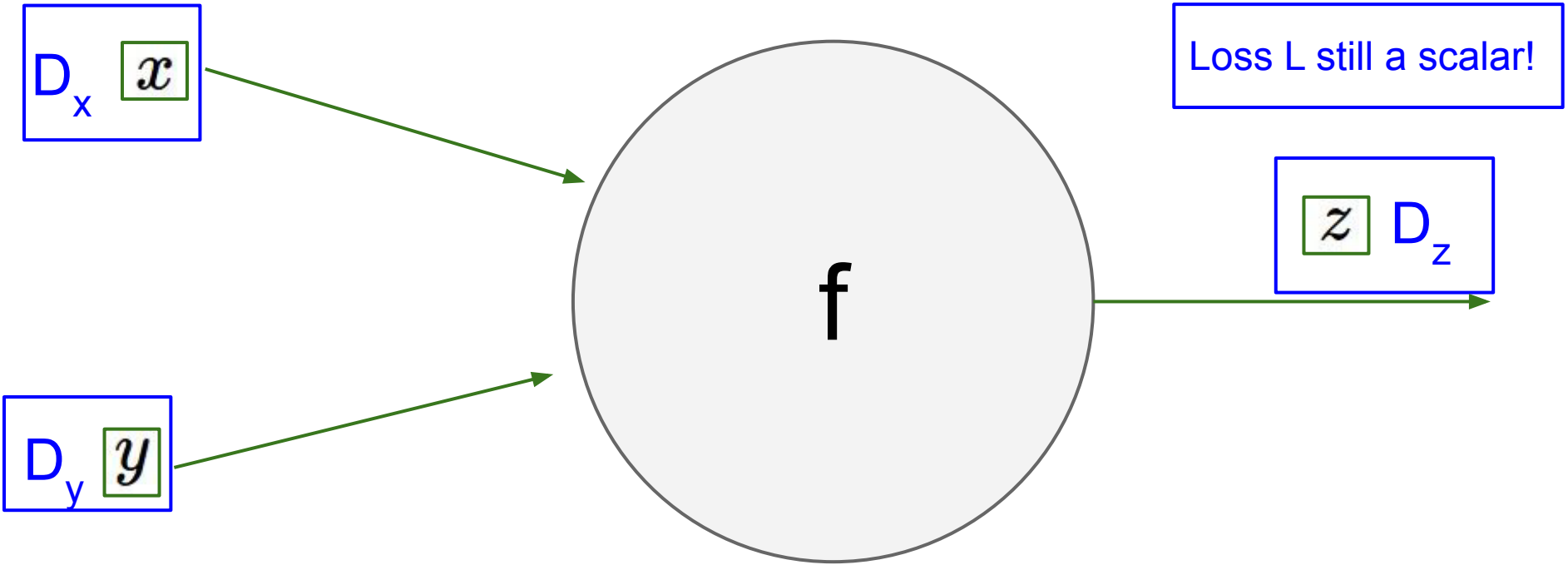
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

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

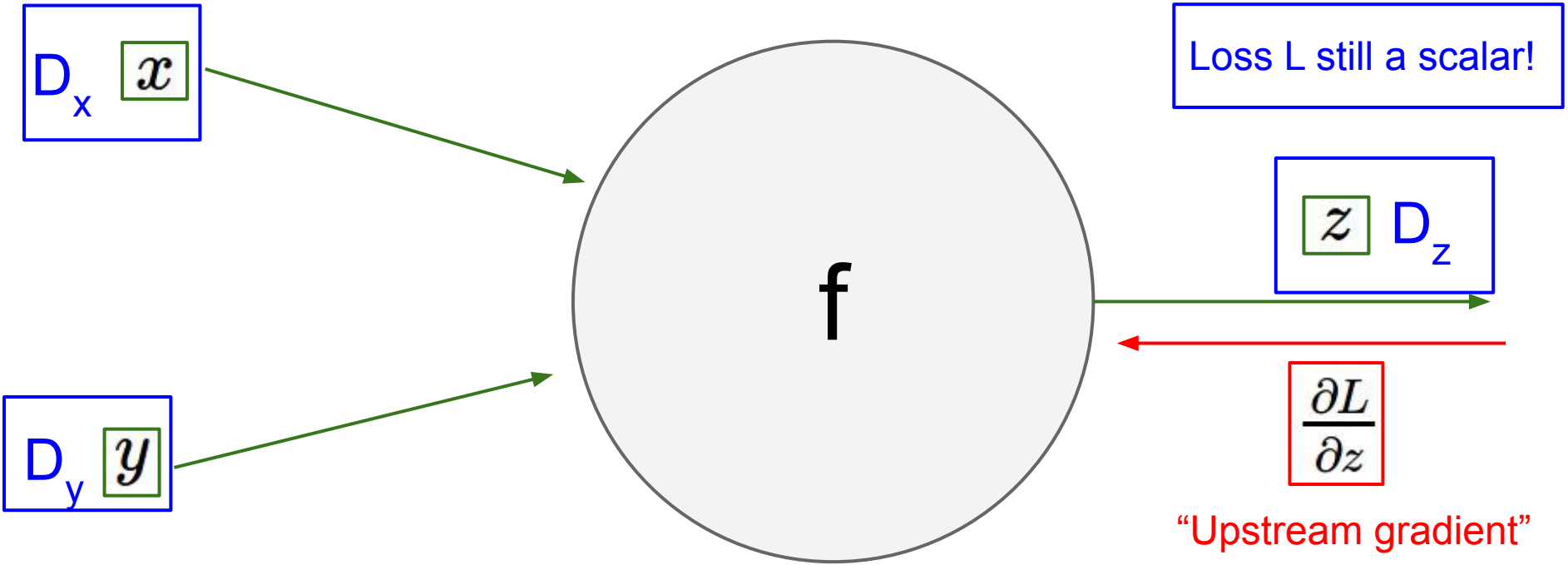
# Backprop with Vectors



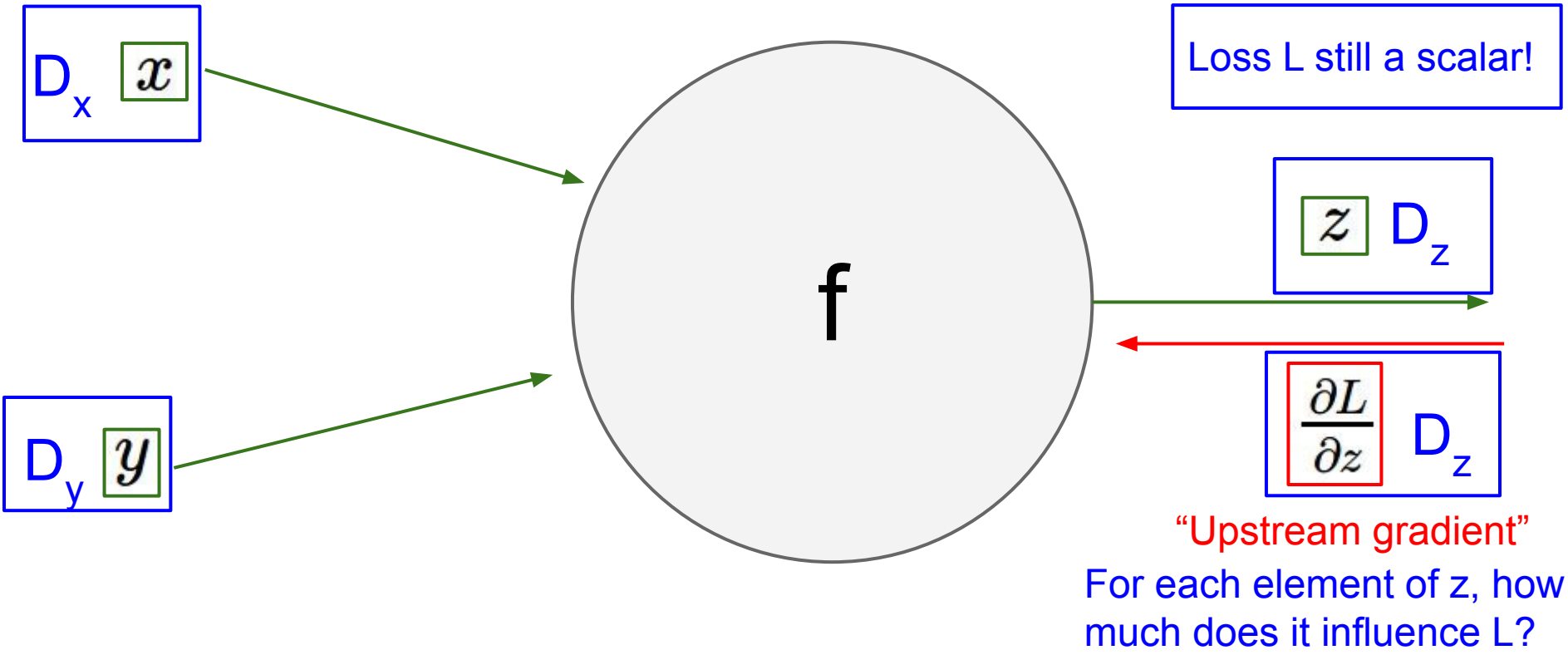
# Backprop with Vectors



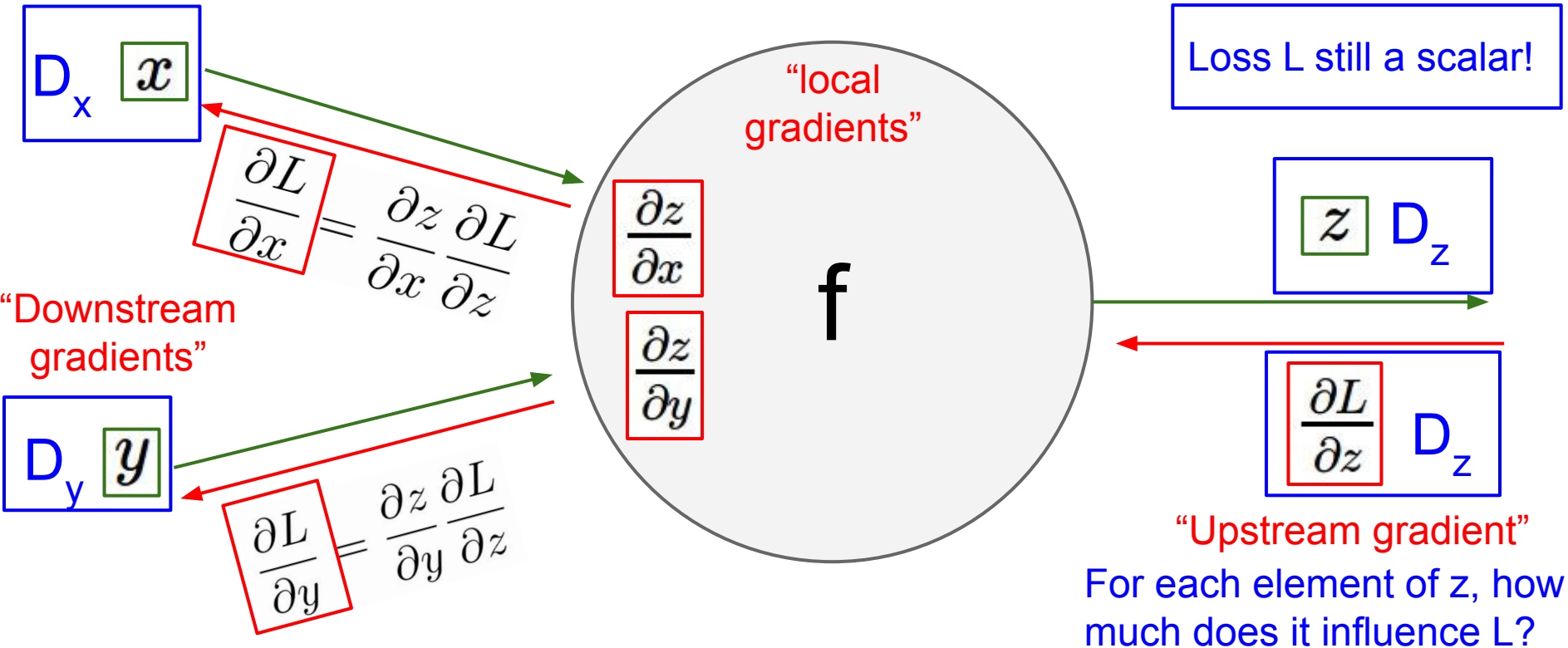
# Backprop with Vectors



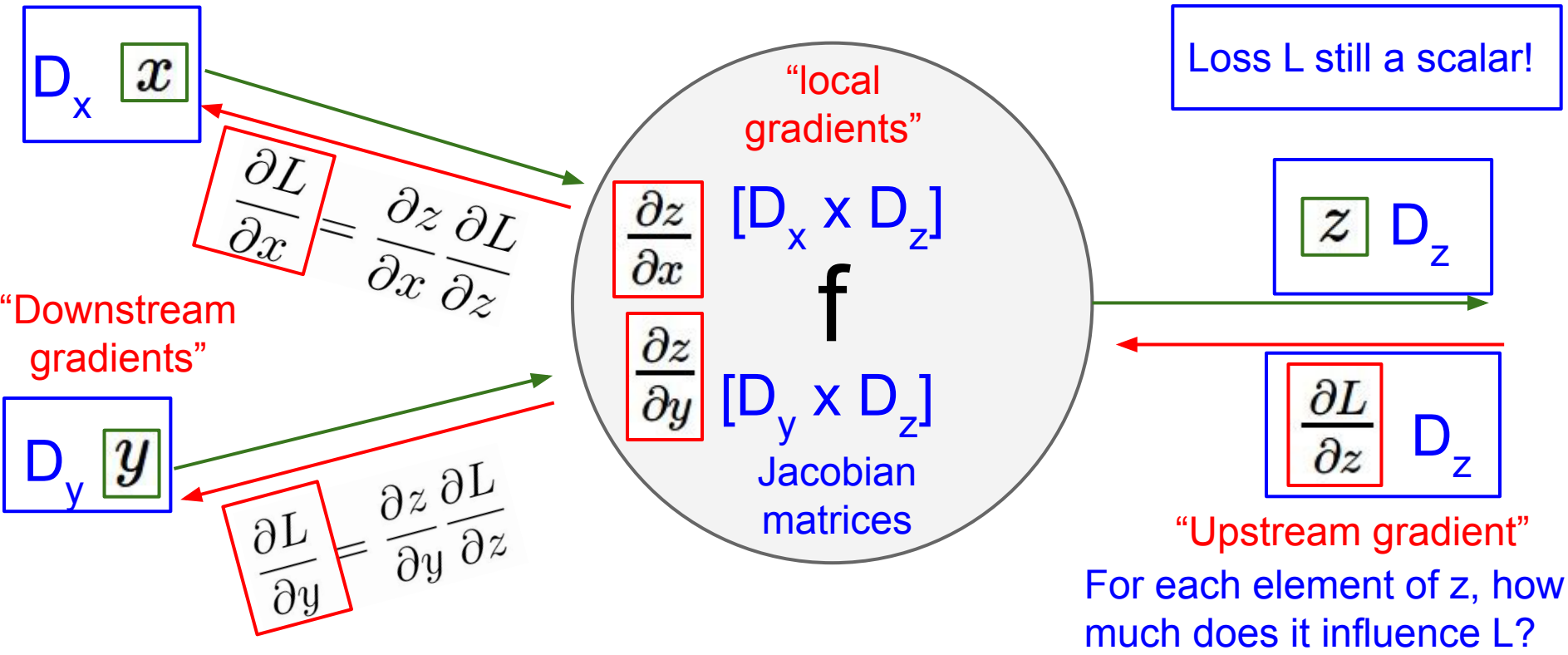
# Backprop with Vectors



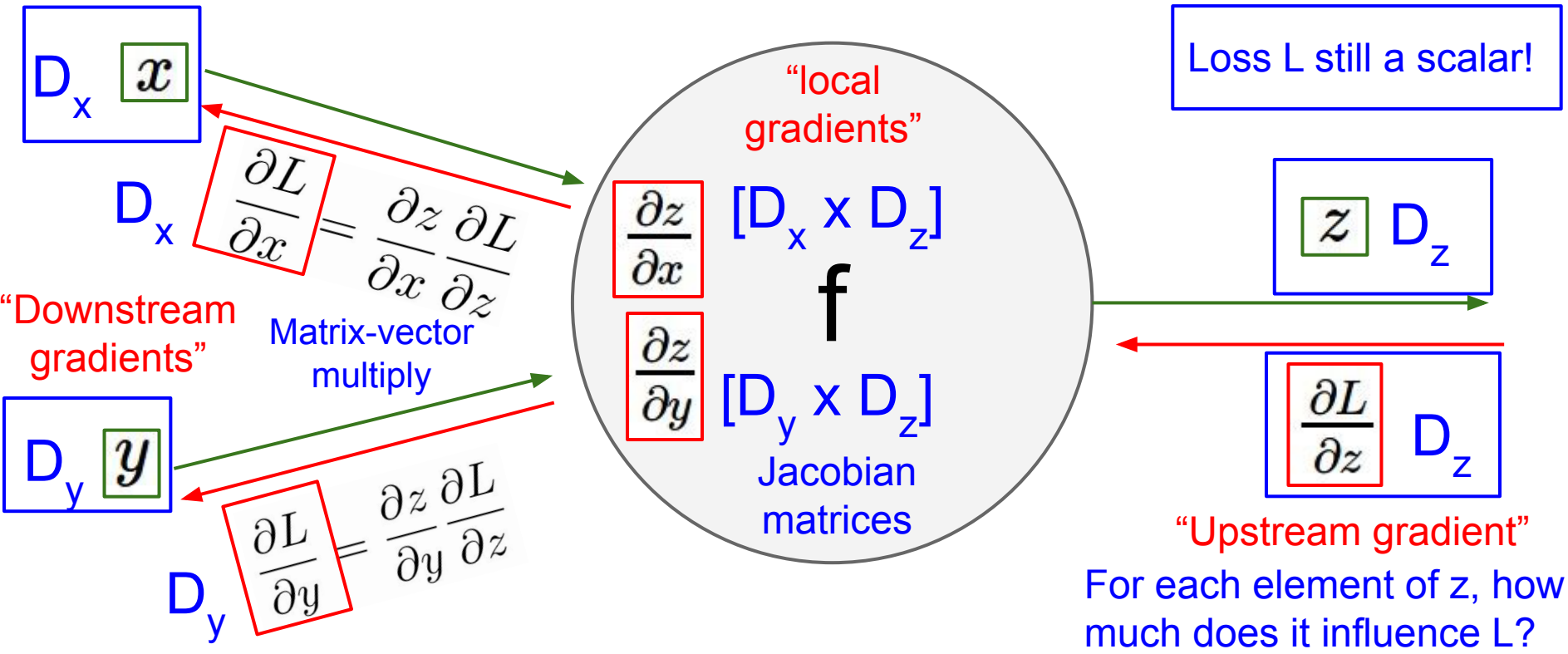
# Backprop with Vectors



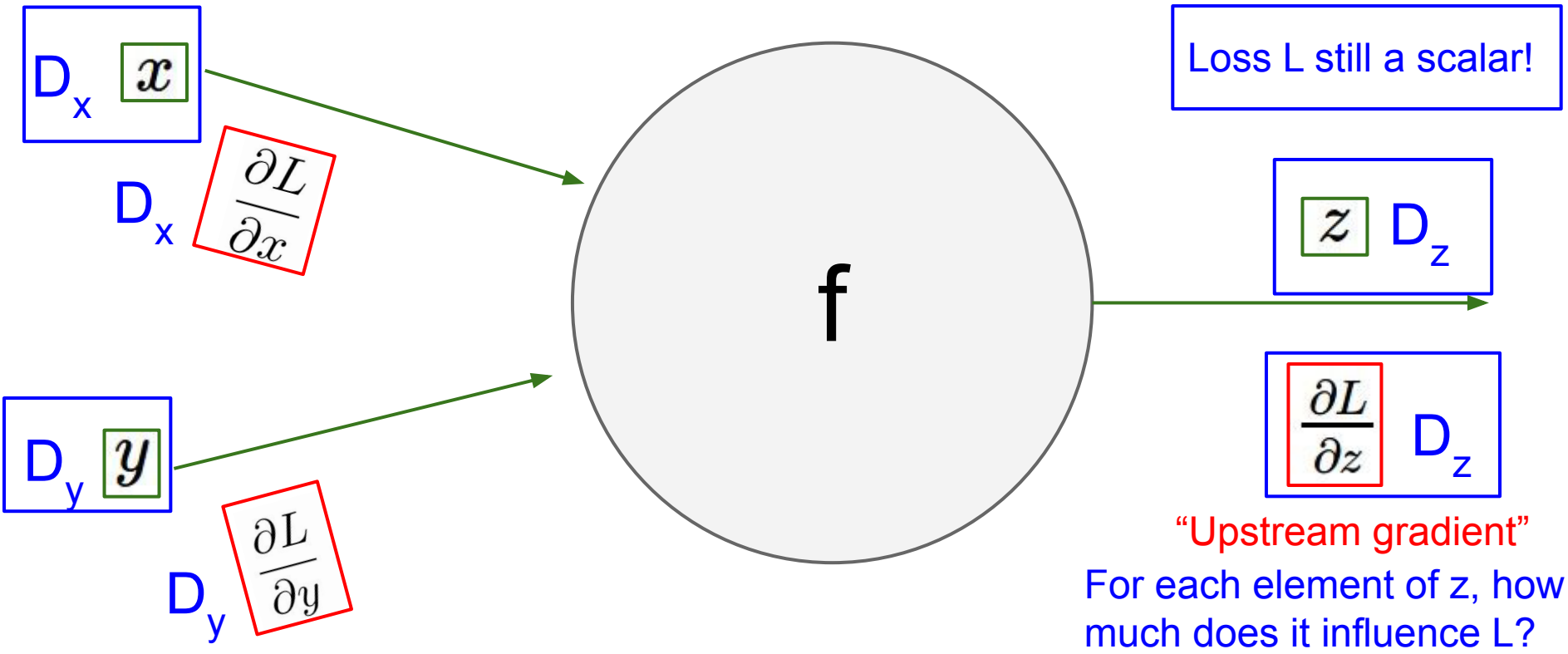
# Backprop with Vectors



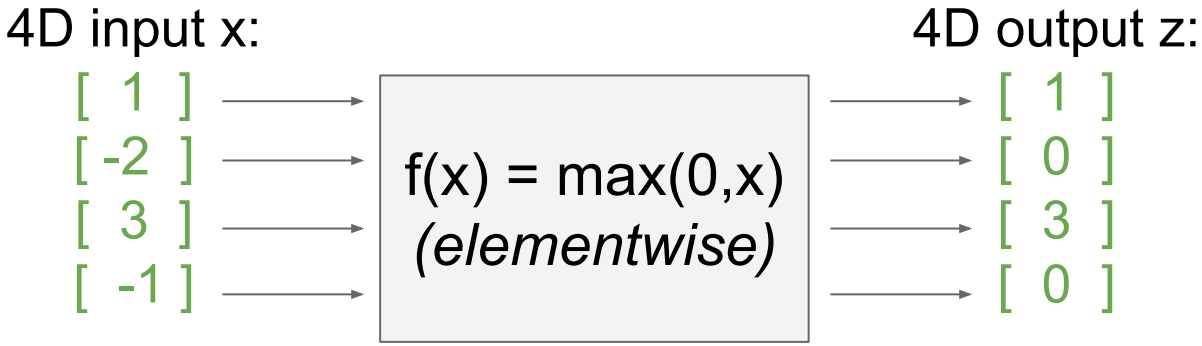
# Backprop with Vectors



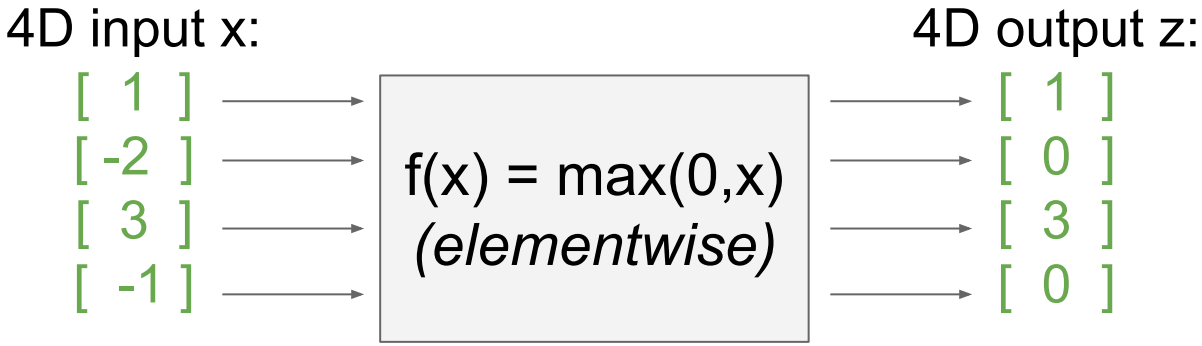
# Gradients of variables wrt loss have same dims as the original variable



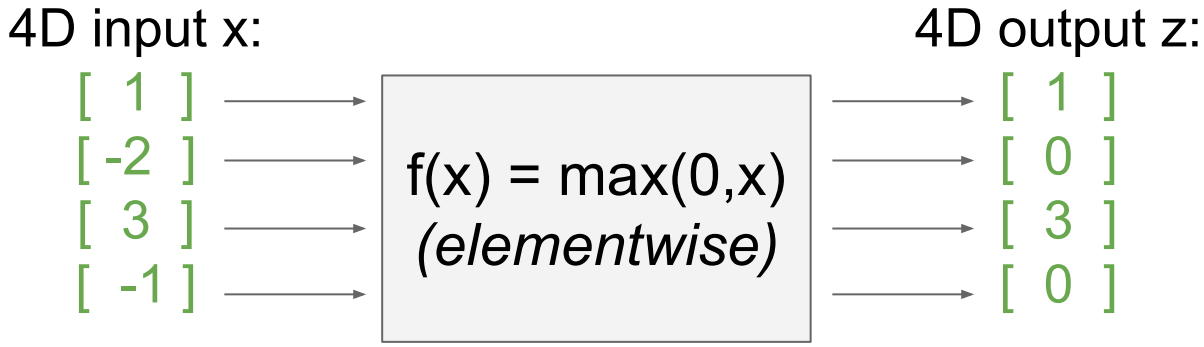
Backprop with Vectors



Backprop with Vectors



Backprop with Vectors




Jacobian dz/dx

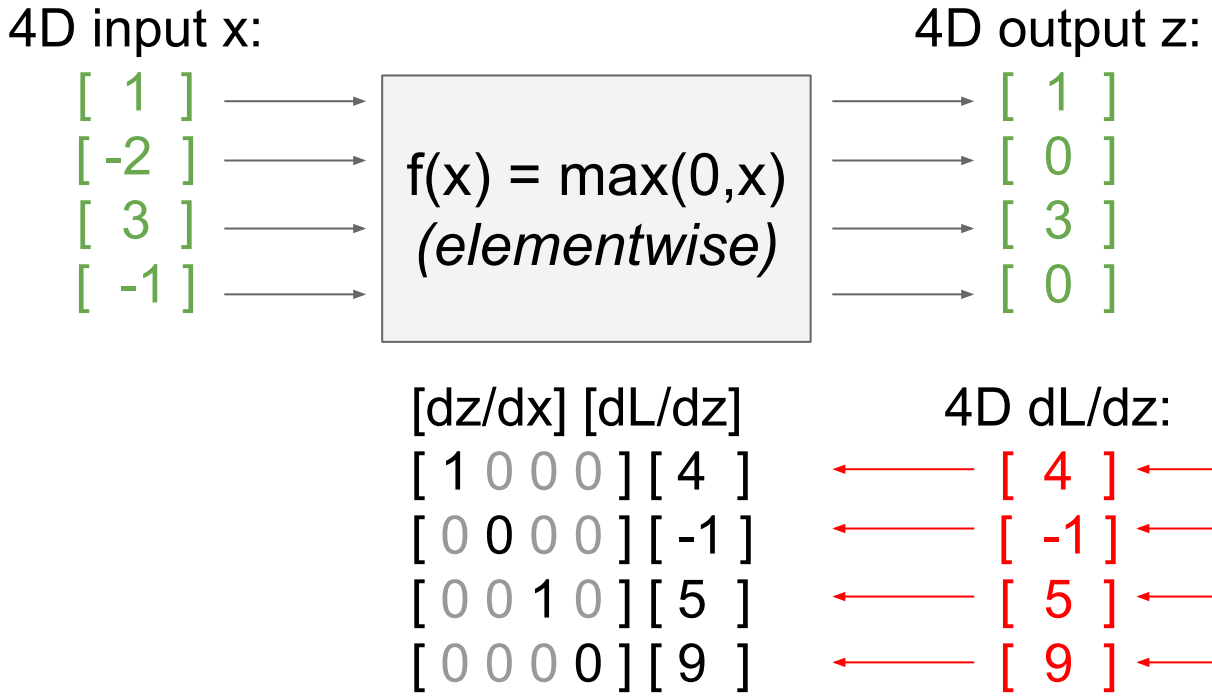
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4D dL/dz:

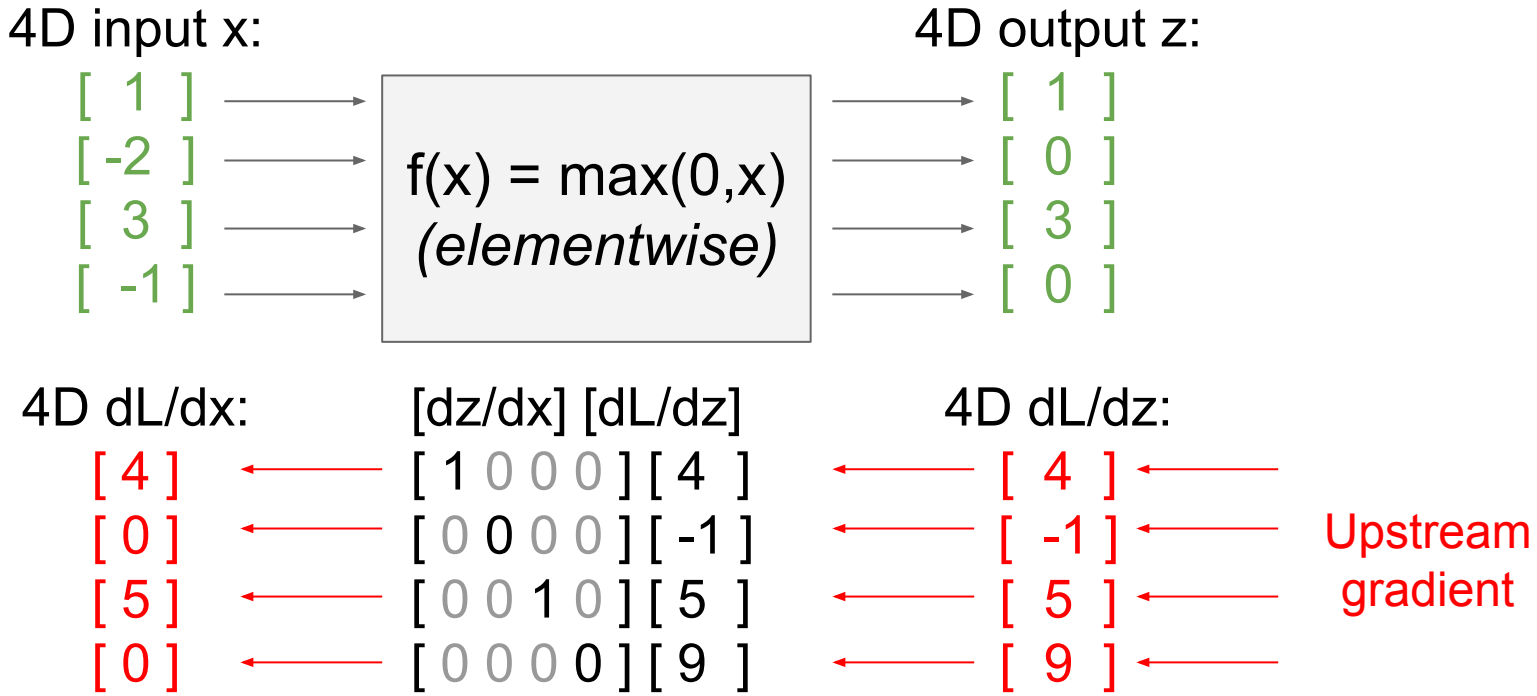
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream gradient

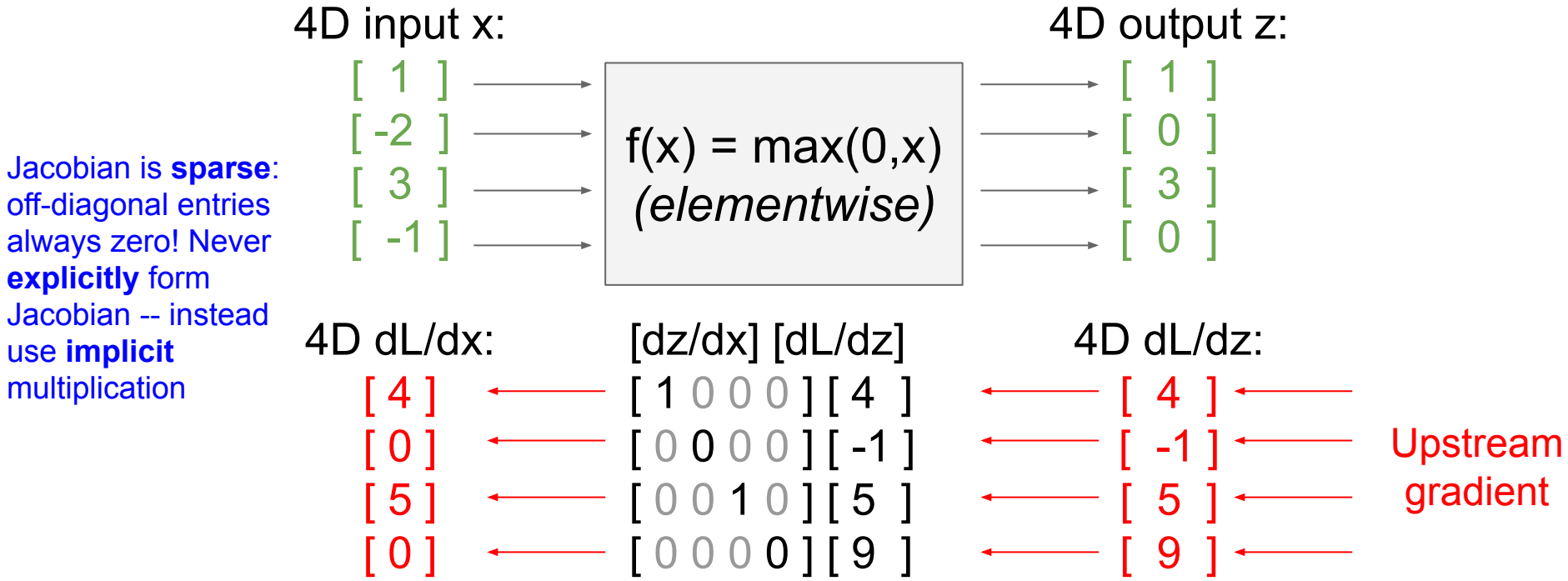
Backprop with Vectors



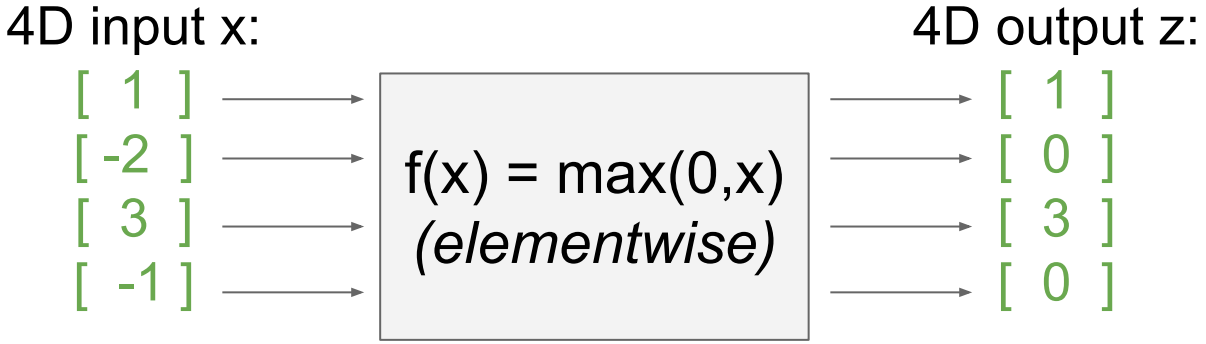
Backprop with Vectors



Backprop with Vectors



Backprop with Vectors




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

4D dL/dx:

[ 4 ]
[ 0 ]
[ 5 ]
[ 0 ]

[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:

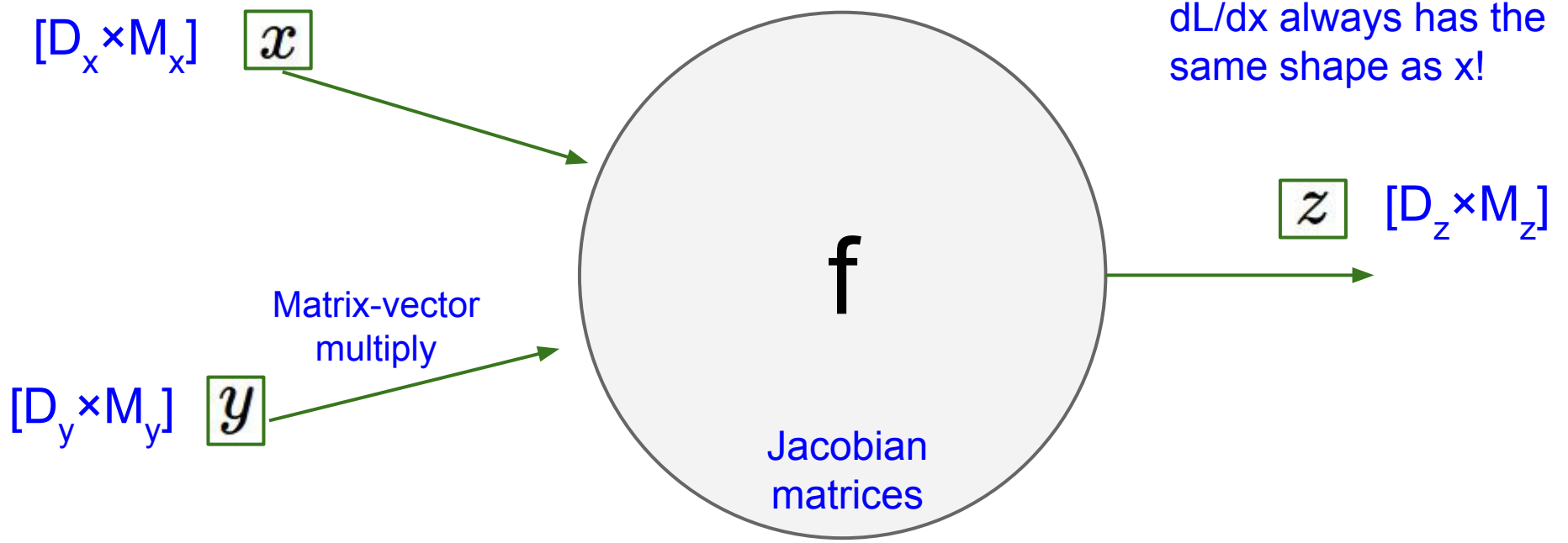
[ 4 ]
[ -1 ]
[ 5 ]
[ 9 ]

Upstream gradient

# Backprop with Matrices (or Tensors)

Loss L still a scalar!

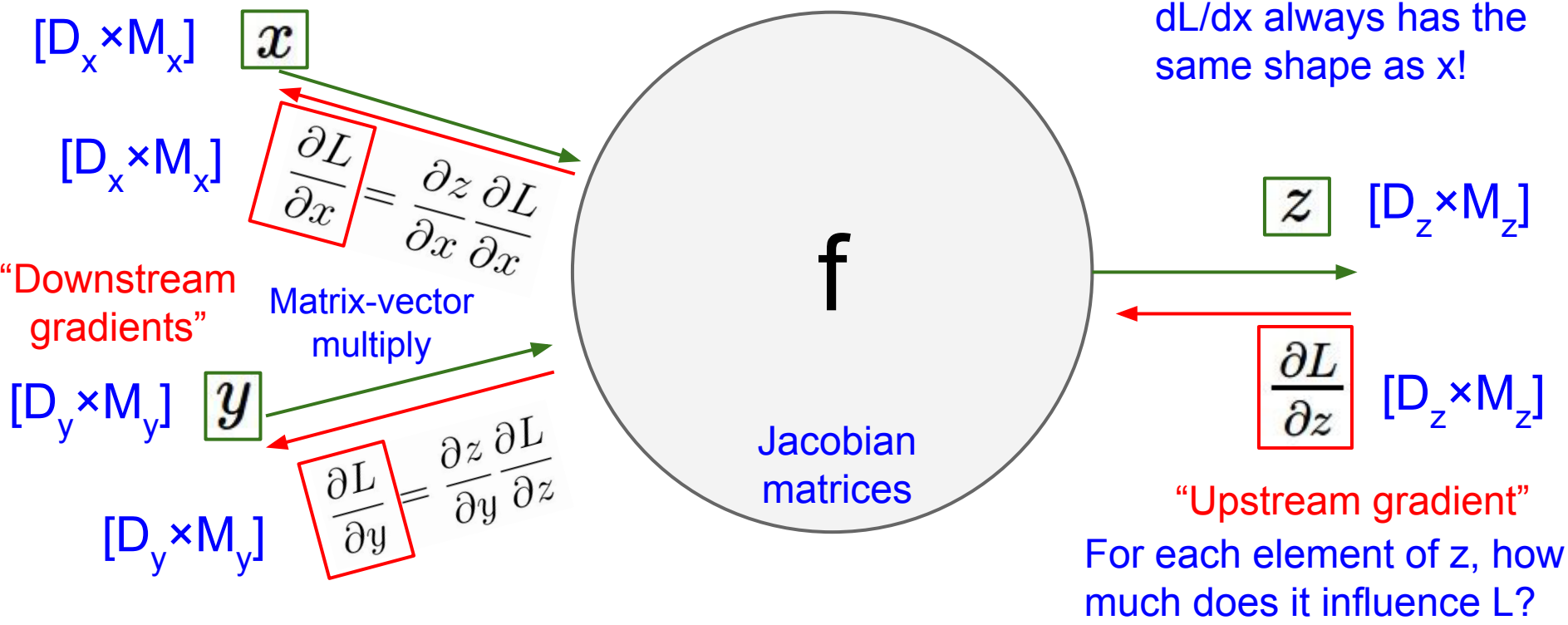
dL/dx always has the same shape as x!



# Backprop with Matrices (or Tensors)

Loss L still a scalar!

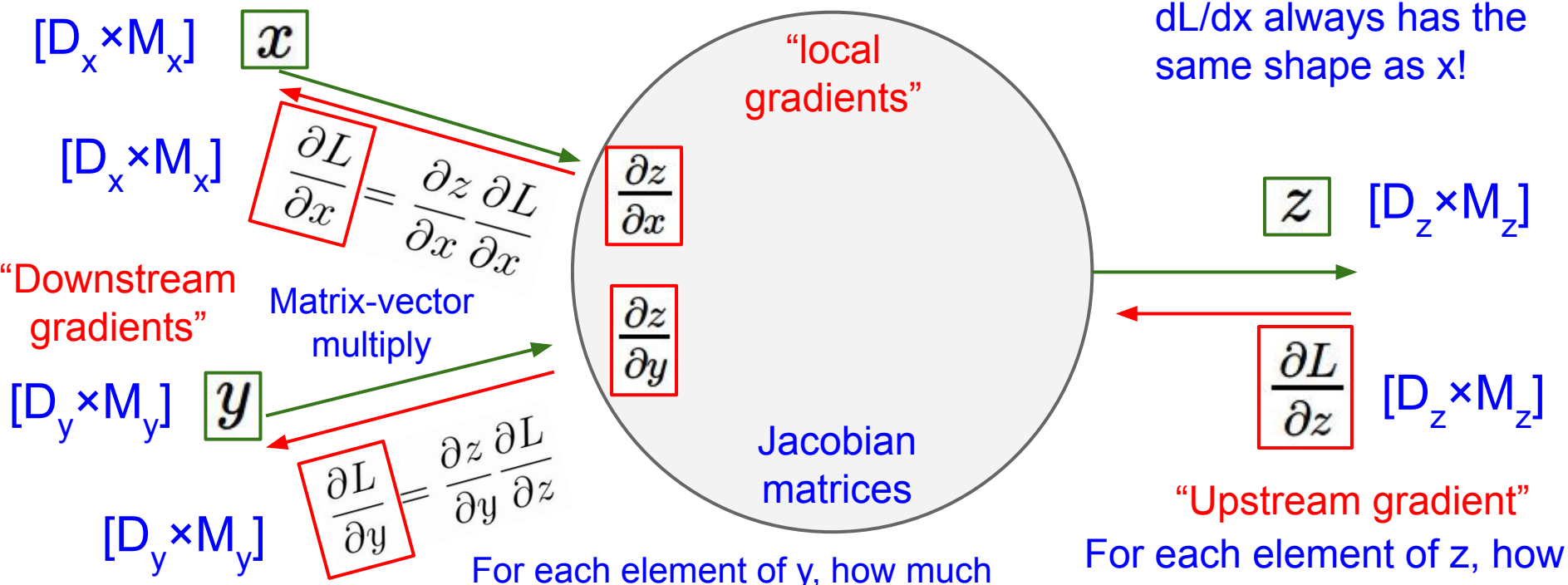
dL/dx always has the same shape as x!



# Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!




"Upstream gradient"

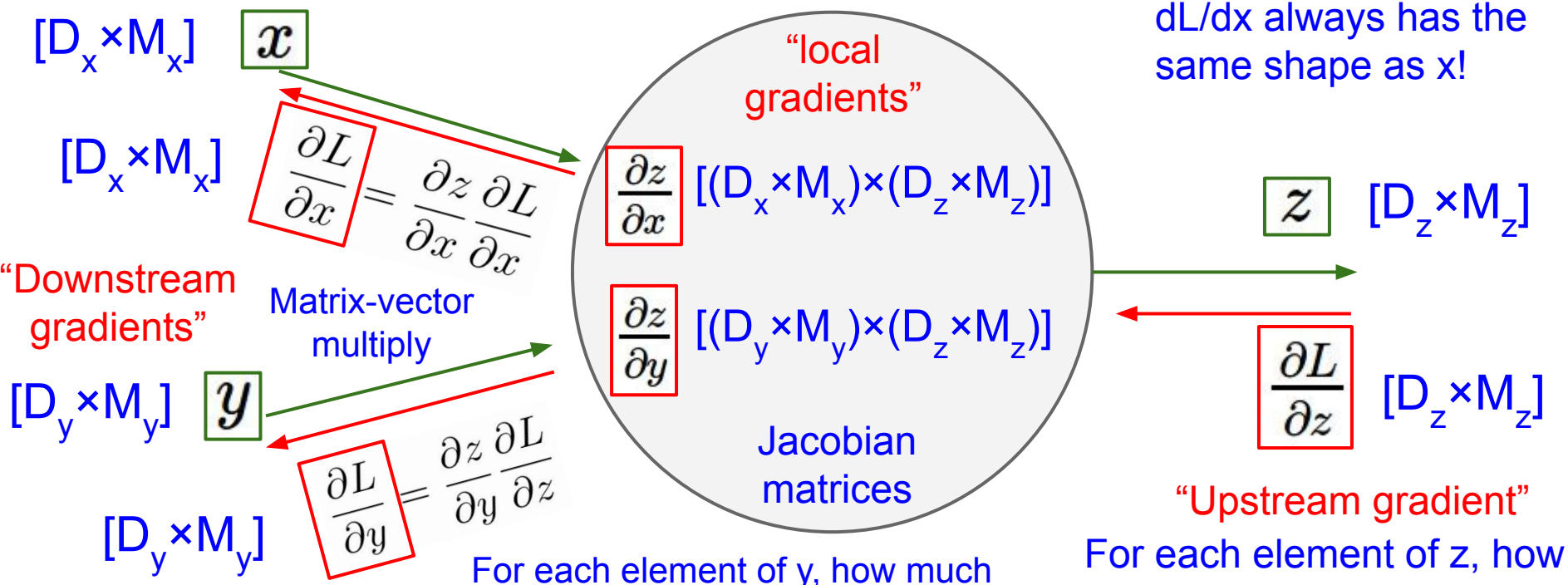
For each element of z, how much does it influence L?

For each element of y, how much does it influence each element of z?

# Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!



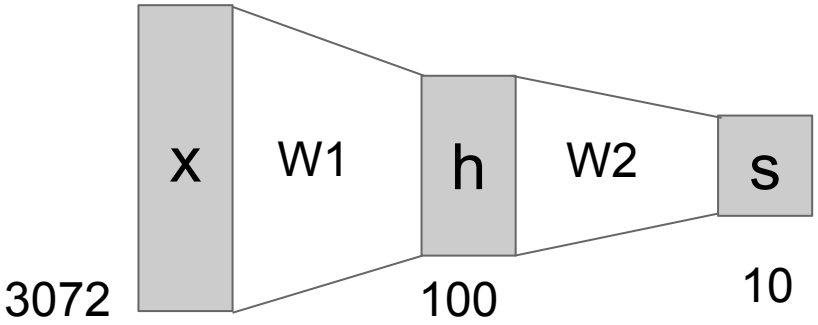

For each element of y, how much does it influence each element of z?

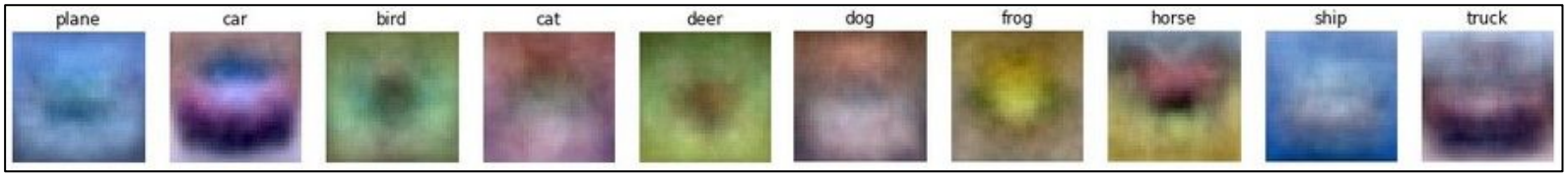
For each element of z, how much does it influence L?

# 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

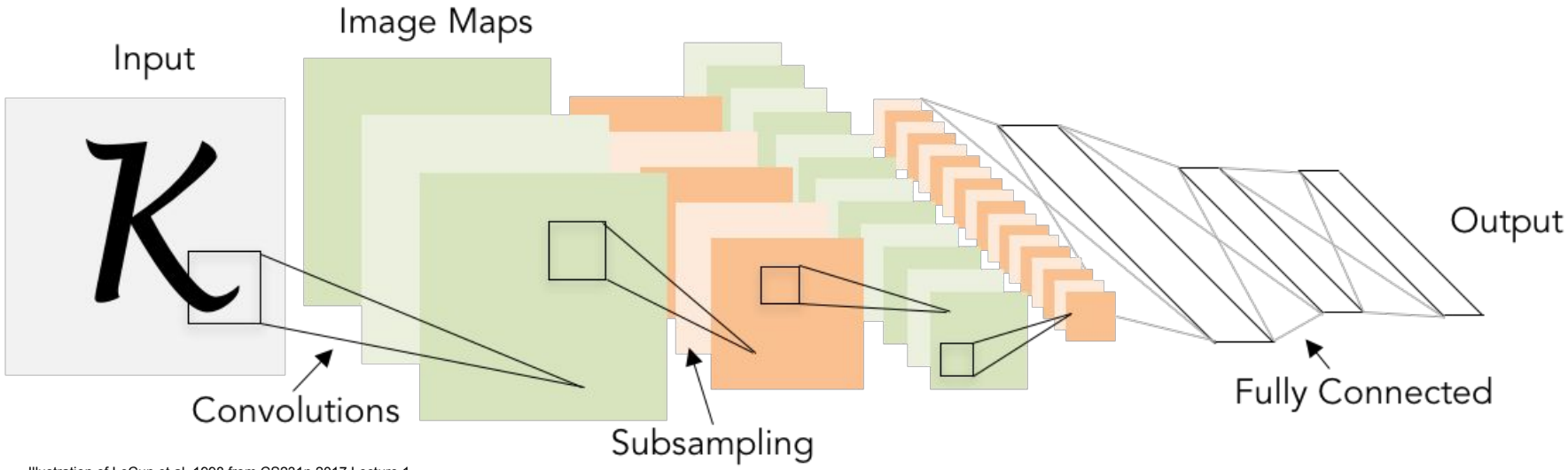



Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

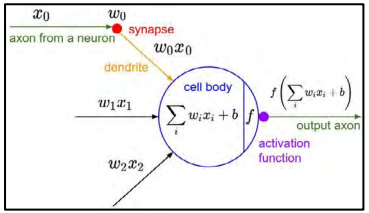
# A bit of history...

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

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



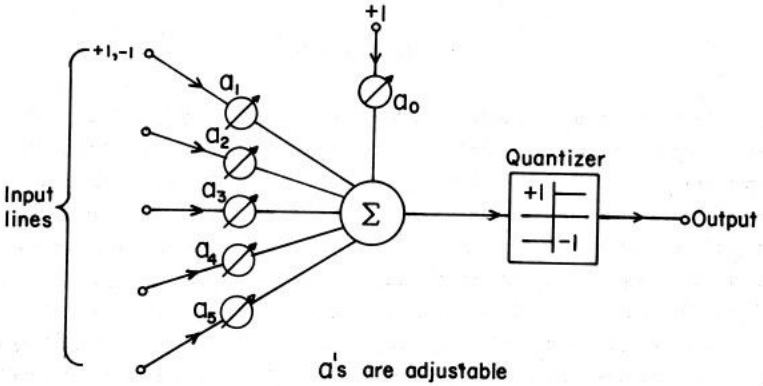

update rule:

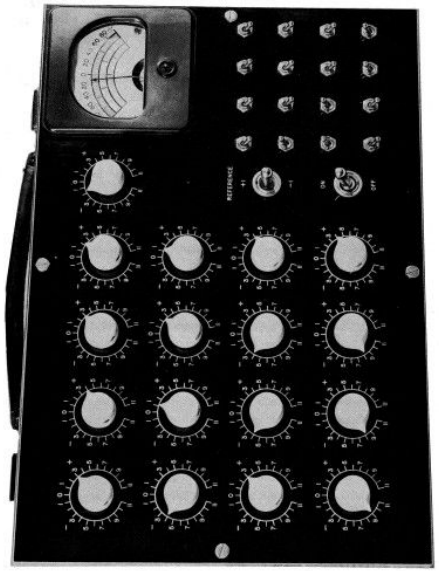
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

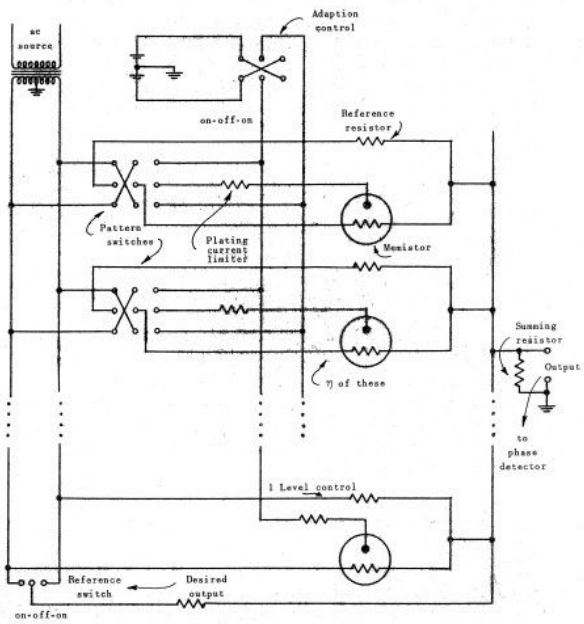
Frank Rosenblatt, ~1957: Perceptron

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...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

recognizable math

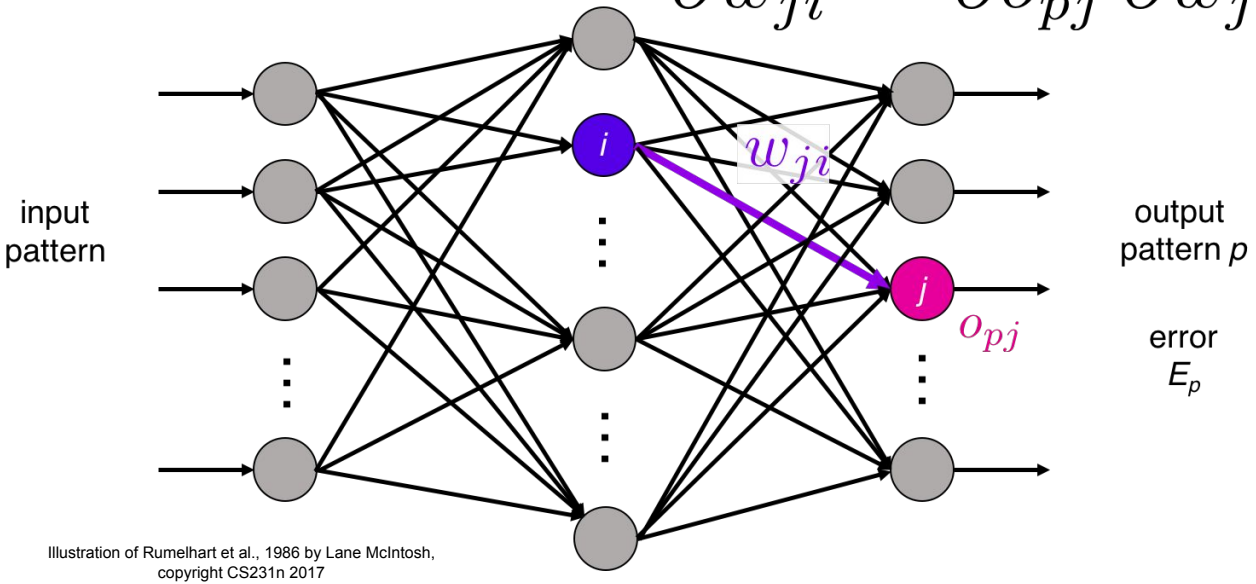



Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

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

# A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

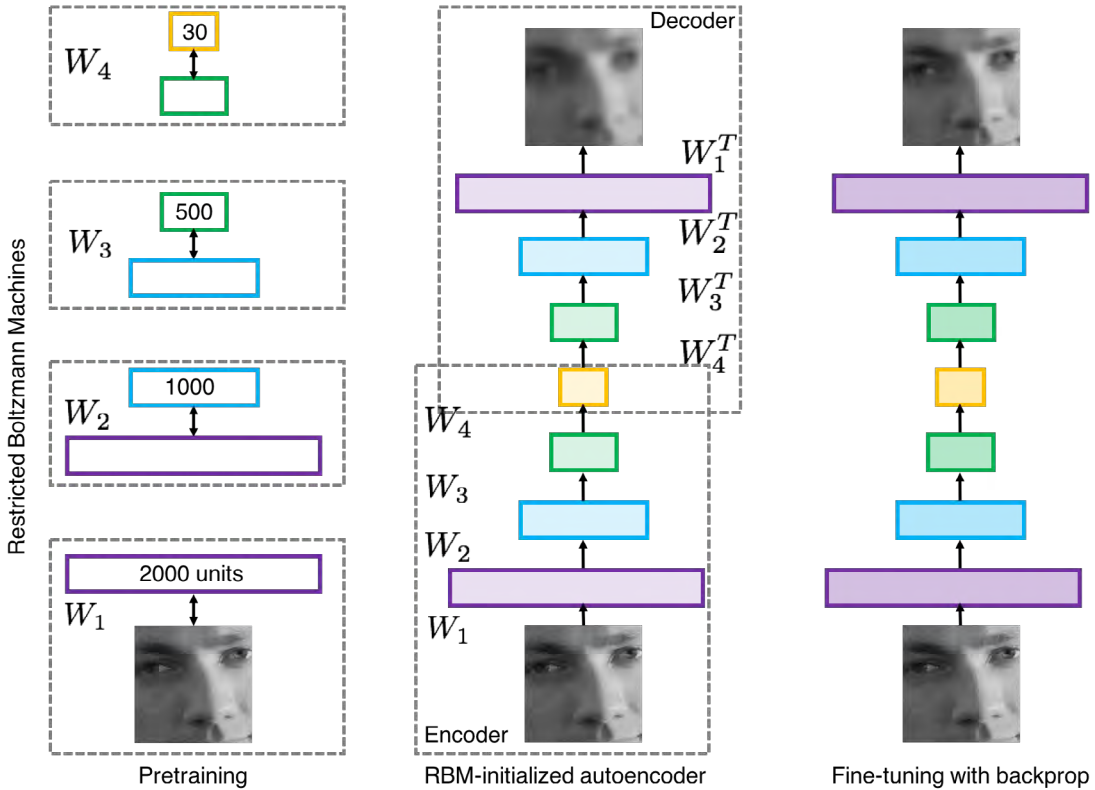



Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

# 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

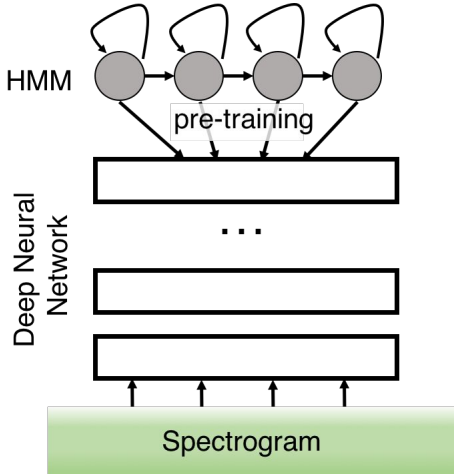
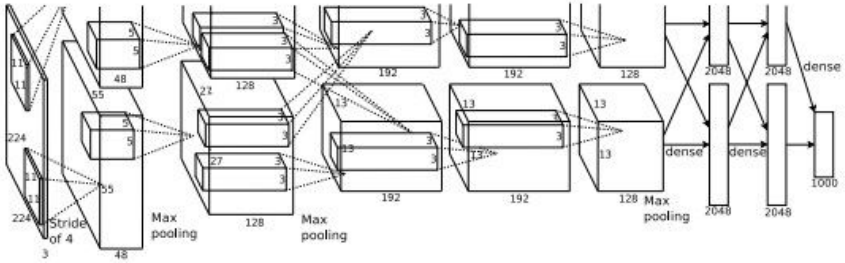
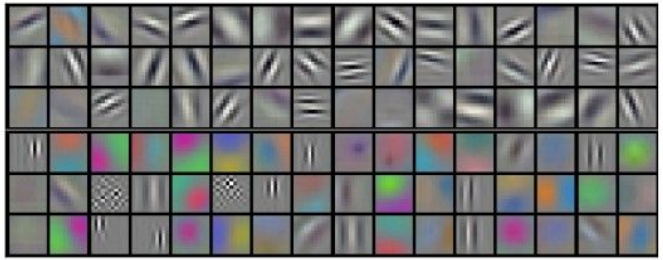



Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

***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,**

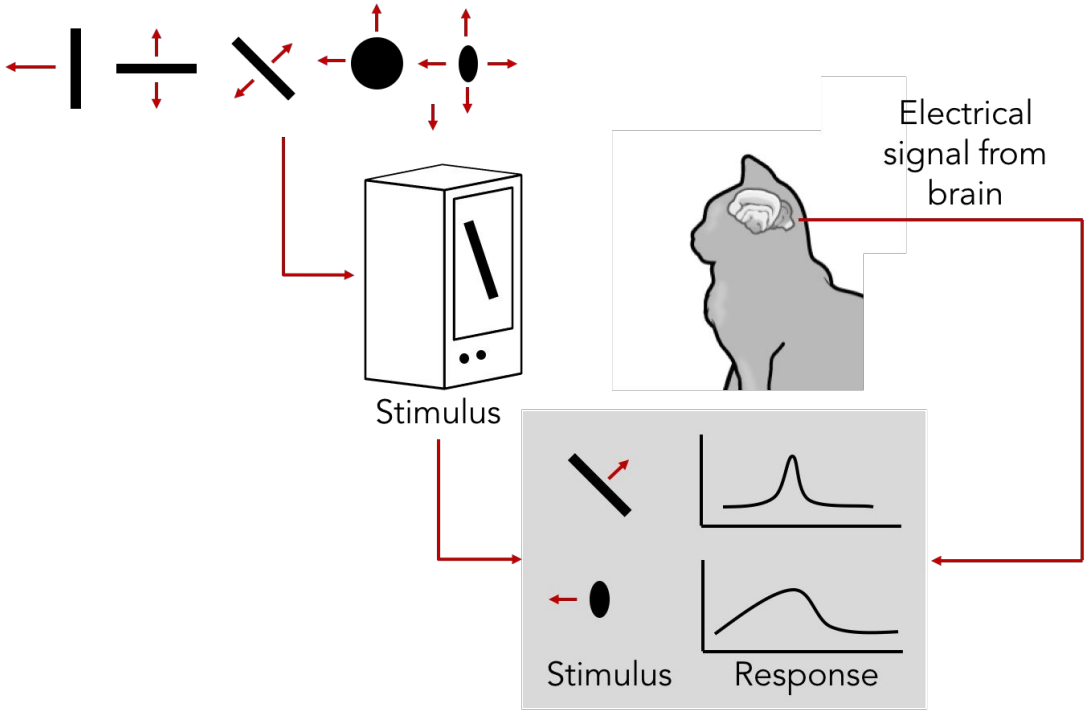
1959

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

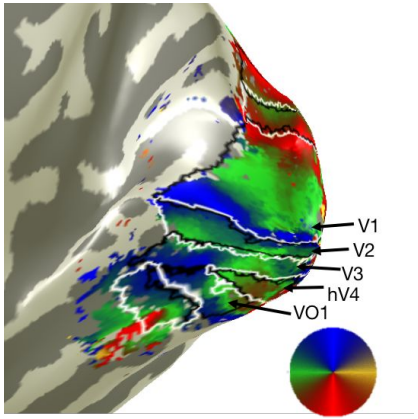
1968...

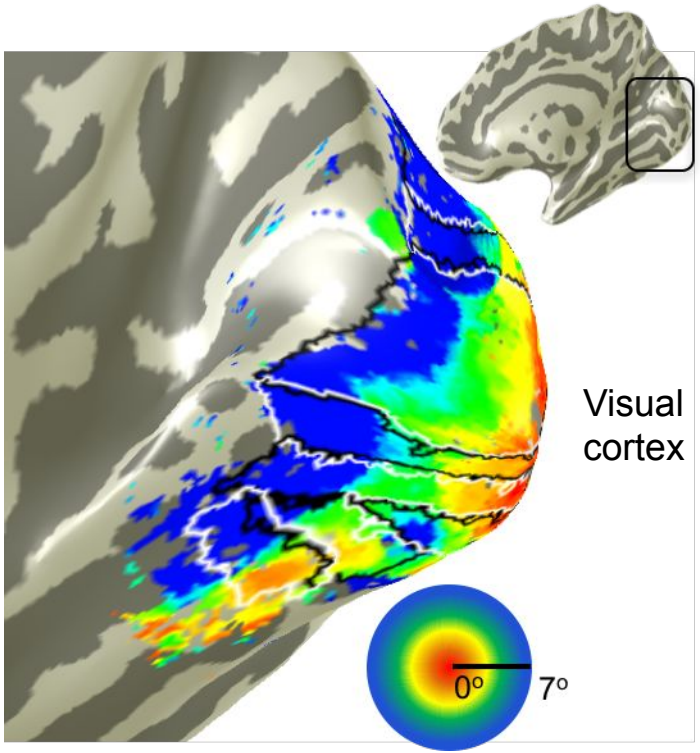



Cat image by CNX OpenStax is licensed under CC BY 4.0; changes made

# 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

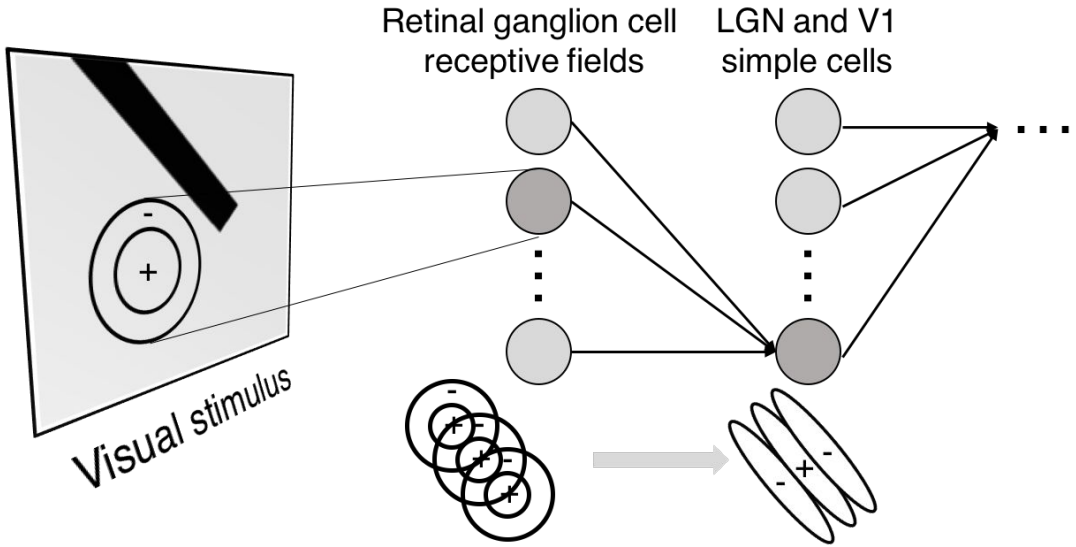
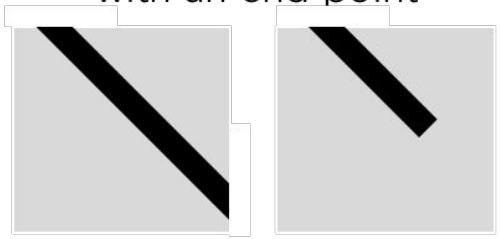



Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

**Simple cells:**
Response to light orientation

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

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



No response

Response (end point)

# A bit of history

Rainer Goebel

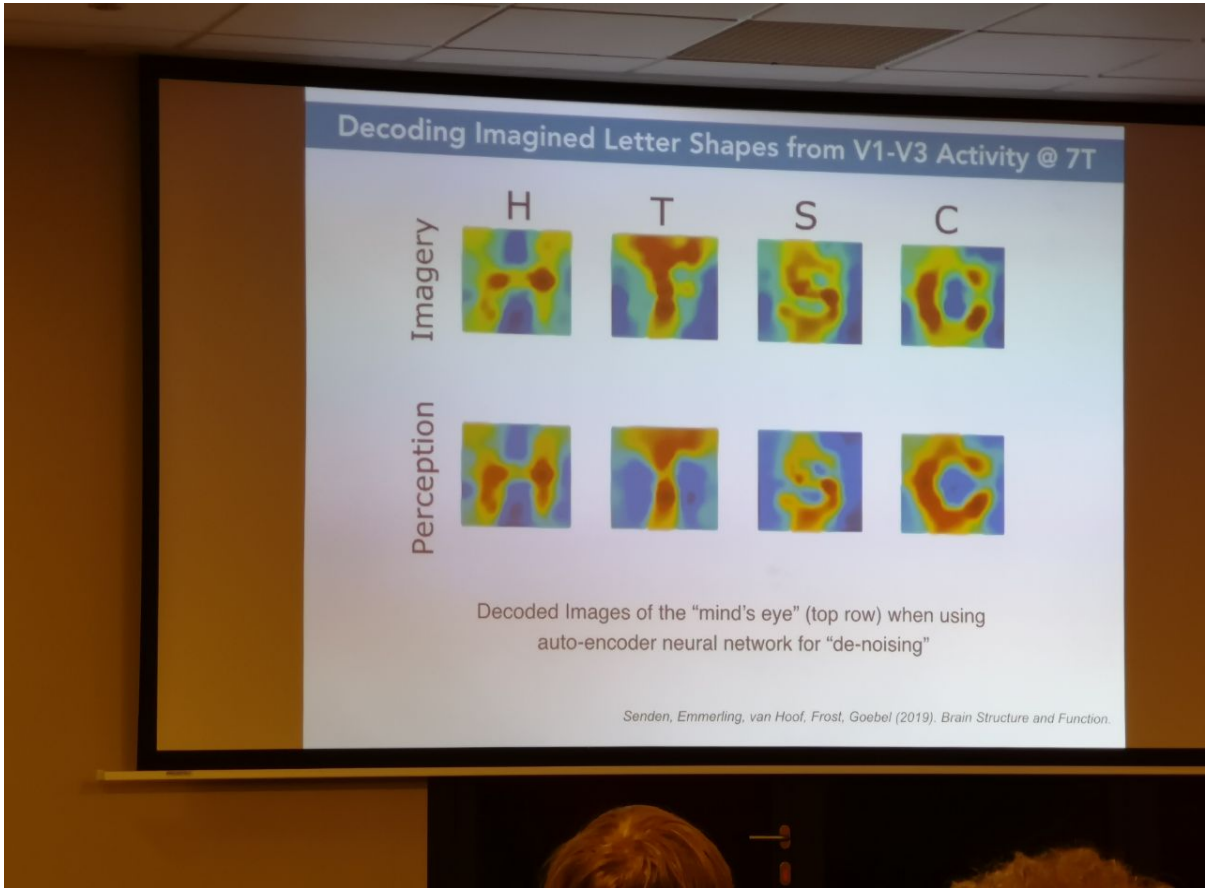
Professor of Cognitive Neuroscience, Maastricht University
Verified email at maastrichtuniversity.nl - Homepage

Ultra-High Field fMRI  Human Brain Research  Cognitive Neuroscience
Neural Network Modelling  Brain-Computer Interfaces

ARTICLES  CITED BY  CO-AUTHORS

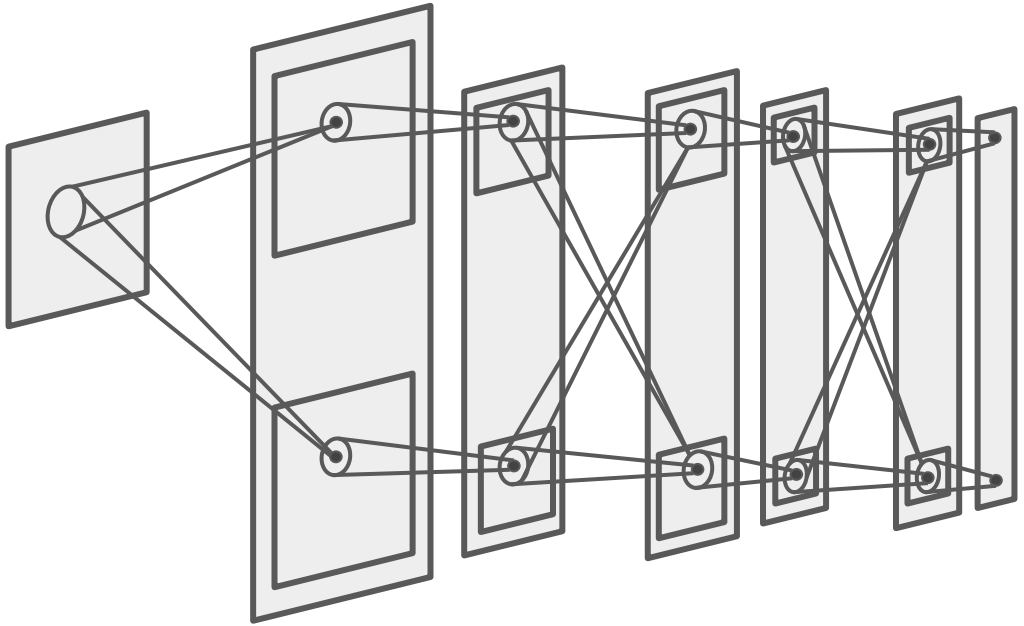
| | All | Since 2015 |
|---|---|---|
| Citations | 34458 | 15380 |
| h-index | 101 | 67 |
| i10-index | 274 | 241 |

Создатель Turbo BrainVoyajer
Наша альтернатива для TBV:
http://opennft.org/



# A bit of history:

**Neocognitron**
*[Fukushima 1980]*

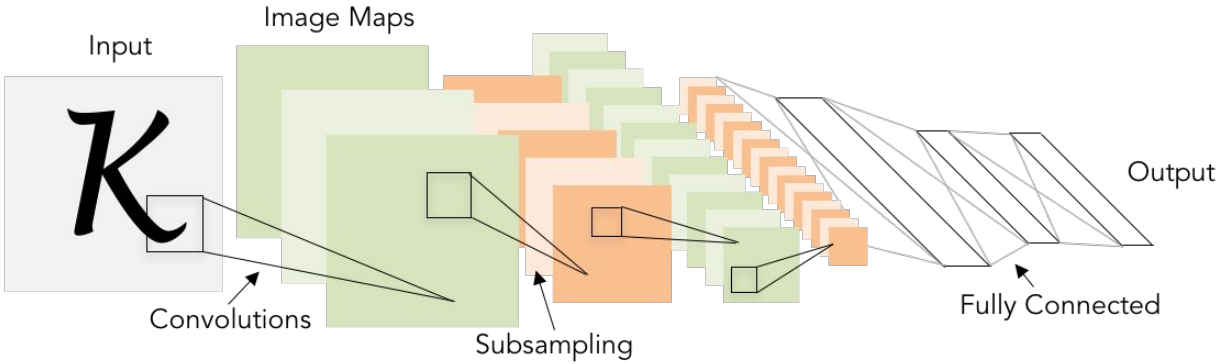


"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

# A bit of history:

**Gradient-based learning applied to document recognition**

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

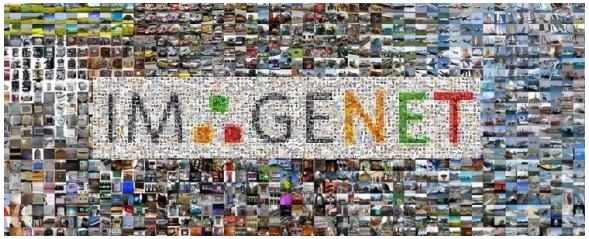



LeNet-5

# A bit of history:

## ImageNet Classification with Deep Convolutional Neural Networks

*[Krizhevsky, Sutskever, Hinton, 2012]*



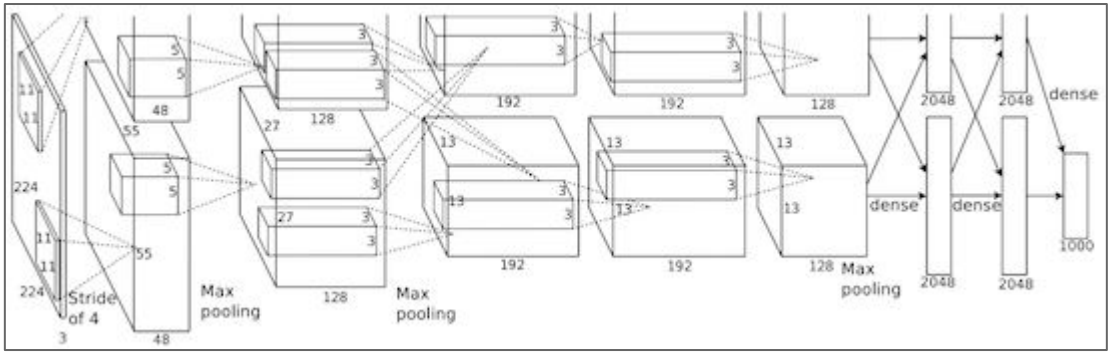




Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

# A bit of history

1989 G Cybenko
Теорема об универсальной аппроксимации

Approximation by superpositions of a sigmoidal function - Springer Link
https://link.springer.com/article/10.1007/BF02551274 - Перевести эту страницу
автор: G Cybenko - 1989 - Цитируется: 10688 - Похожие статьи

1998 Yann LeCun
сверточные сети

ieeexplore.ieee.org › document - Перевести эту страницу
Gradient-based learning applied to document recognition ...
**Gradient-based learning applied to document recognition**. ... A new **learning** paradigm, called graph transformer networks (GTN), allows such multimodule systems to be trained globally using **gradient-based** methods so as to minimize an overall performance measure. Two systems for online handwriting **recognition** are described.
автор: Y Lecun - 1998 - Цитируется: 28105 - Похожие статьи

2007 – Выход NVIDIA CUDA,

2009 – Google отказывается от нейронных сетей

2012 – AlexNet

[PDF] ImageNet Classification with Deep Convolutional Neural Networks
https://papers.nips.cc/.../4824-imagenet-classification-with-de... ▾ Перевести эту страницу
автор: A Krizhevsky - 2012 - Цитируется: 34232 - Похожие статьи

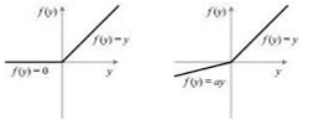


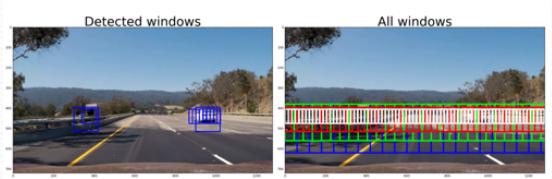

Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

Delving Deep into Rectifiers: Surpassing Human-Level Performance .
https://arxiv.org › cs ▾ Перевести эту страницу
автор: K He - 2015 - Цитируется: 3856 - Похожие статьи

IEEE CVPR Cite Score: 3.23 (2012), 6.19 (2015), 18.18 (2018)

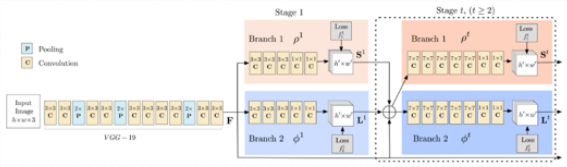
# Fast-forward to today: ConvNets are everywhere

## Fast Detection

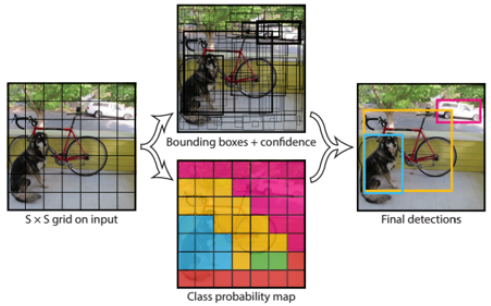



Sliding window detection -NO

## Tracking




Real time pose tracking, CVPR 17




YOLO - You Only Look Once – Yes!

Openpose

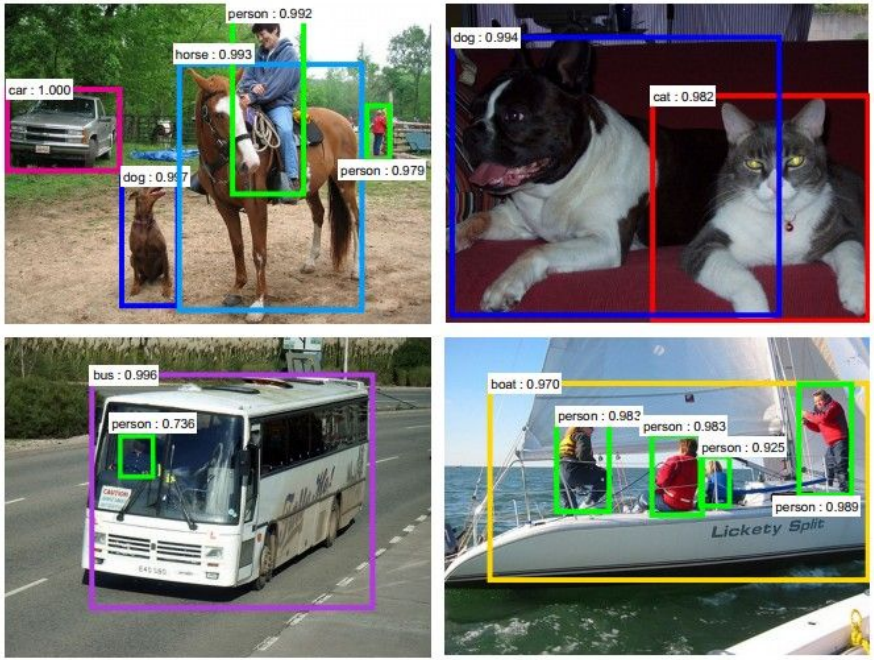
Вычислительная фотография –

- Стекинг фото
- Сверхразрешение
- Съемка в темноте

Light.co

# Fast-forward to today: ConvNets are everywhere

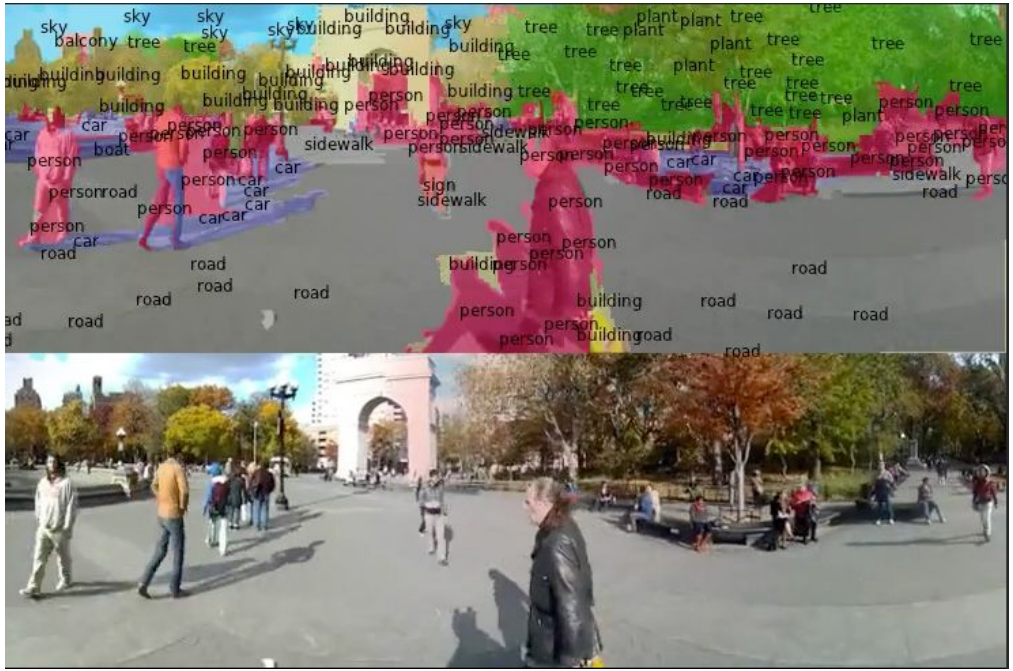
Detection




Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

*[Faster R-CNN: Ren, He, Girshick, Sun 2015]*

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

*[Farabet et al., 2012]*

# Fast-forward to today: ConvNets are everywhere

Photo by Lane McIntosh. Copyright CS231n 2017.

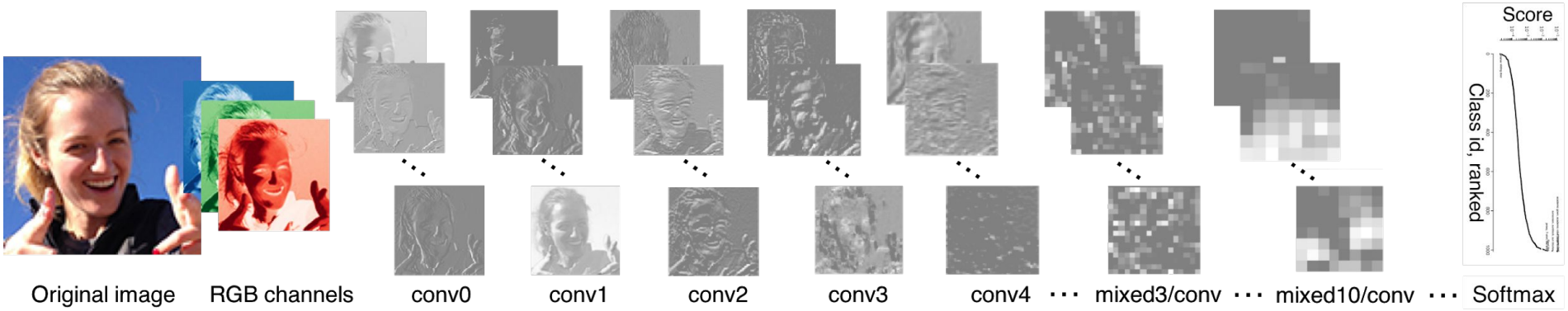
self-driving cars

This image by GBPublic_PR is licensed under CC-BY 2.0

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

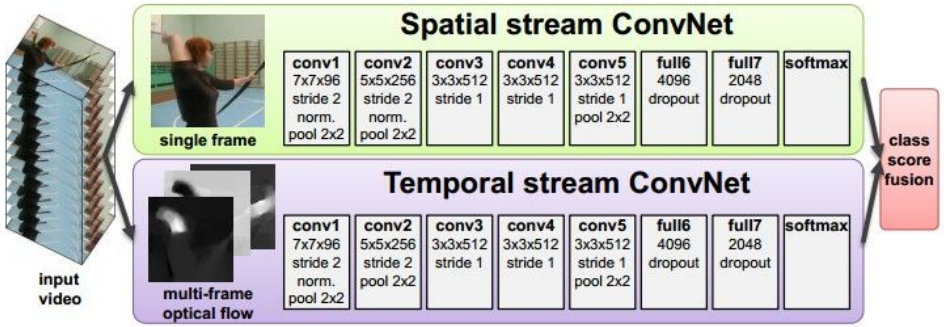
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

# 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.

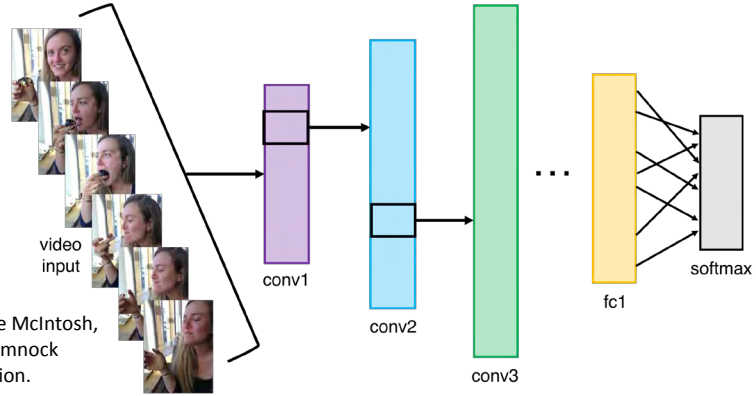
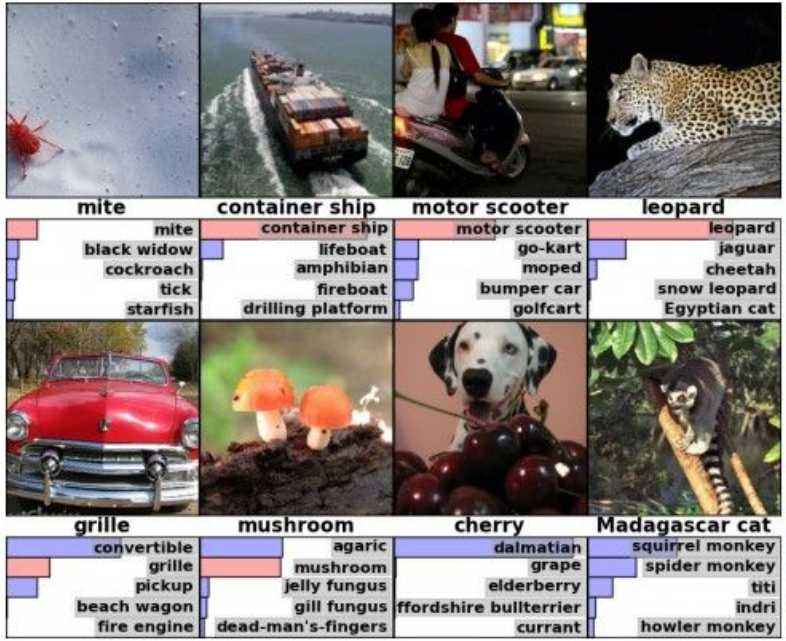



Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

# Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

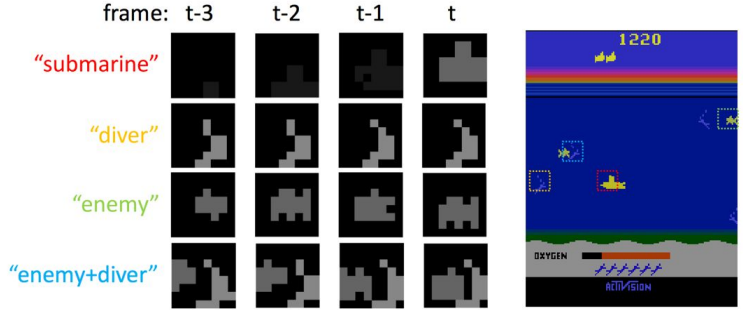


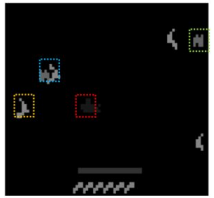
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

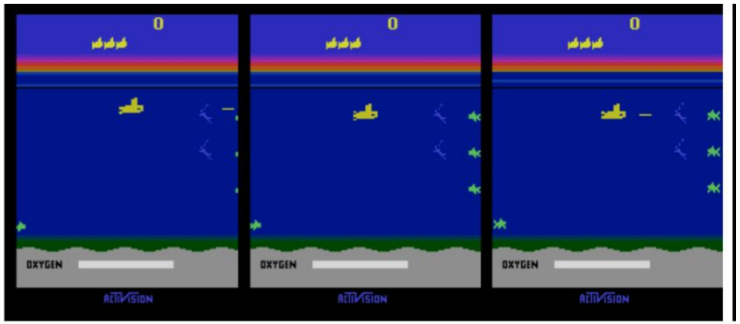
# Fast-forward to today: ConvNets are everywhere

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

*[Toshev, Szegedy 2014]*



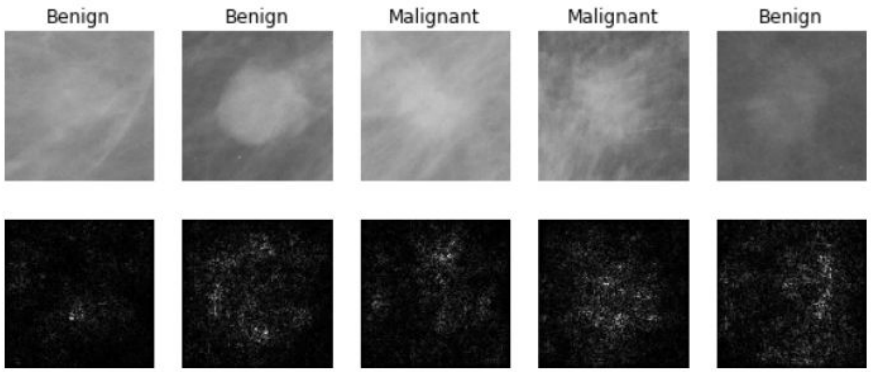






*[Guo et al. 2014]*

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

# 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*

# Image Captioning

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

No errors

*A white teddy bear sitting in the grass*

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

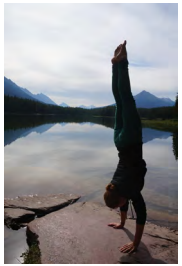
Minor errors

*A man in a baseball uniform throwing a ball*

*A cat sitting on a suitcase on the floor*

Somewhat related

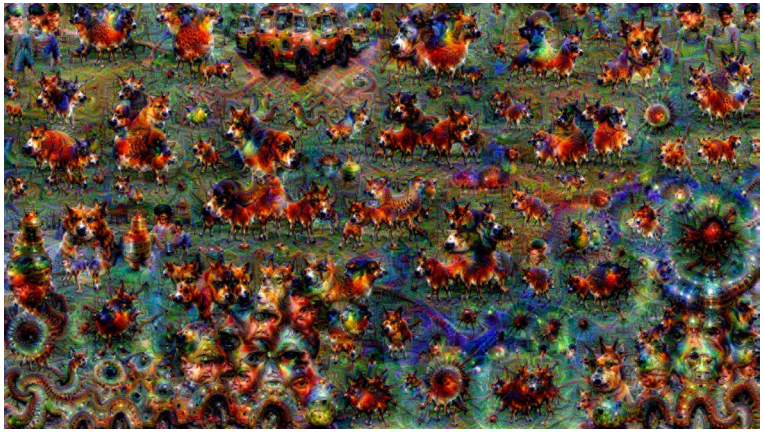
*A woman is holding a cat in her hand*

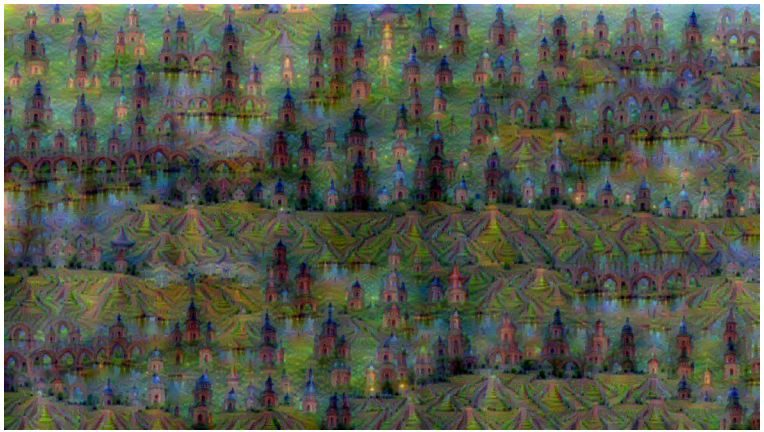


*A woman standing on a beach holding a surfboard*

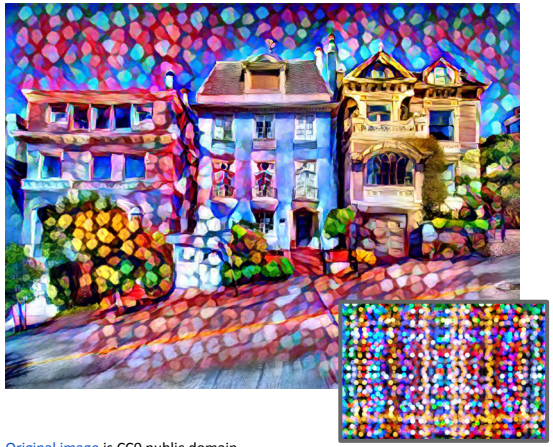
All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
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.



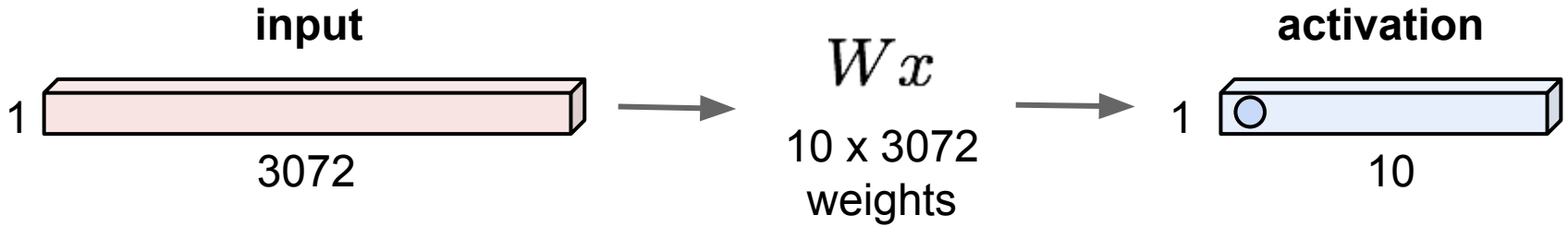
Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

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

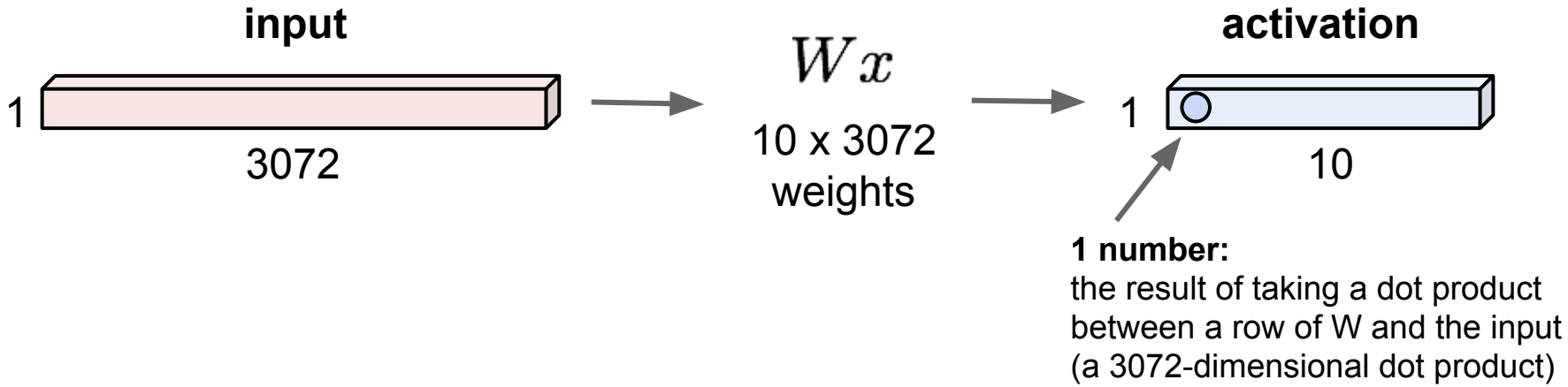
# Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



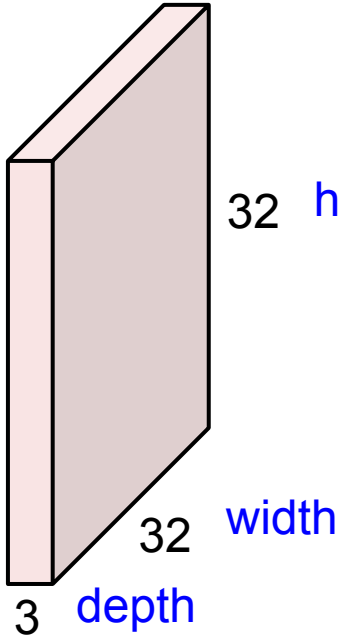
# Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



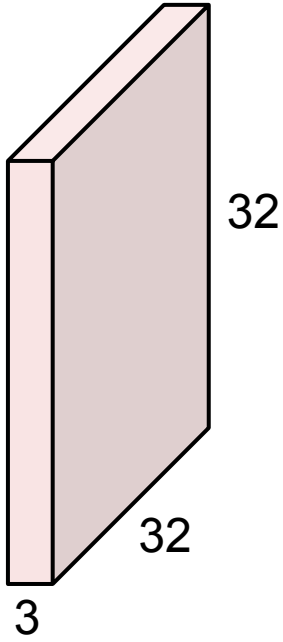
# Convolution Layer

32x32x3 image -> preserve spatial structure

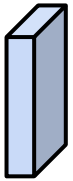


# Convolution Layer

32x32x3 image




5x5x3 filter

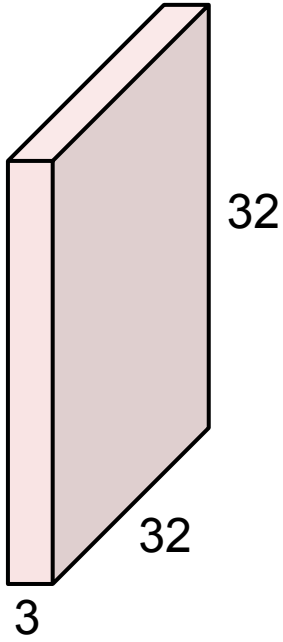


**Convolve** the filter with the image
i.e. “slide over the image spatially,
computing dot products”

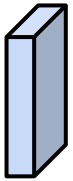
# Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

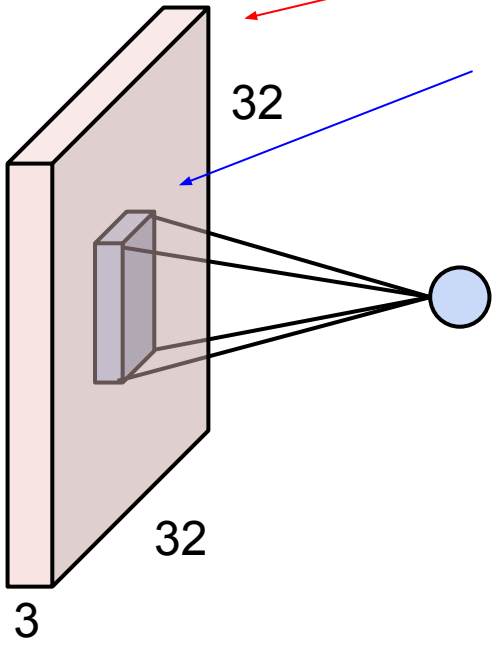



5x5x3 filter



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

# Convolution Layer



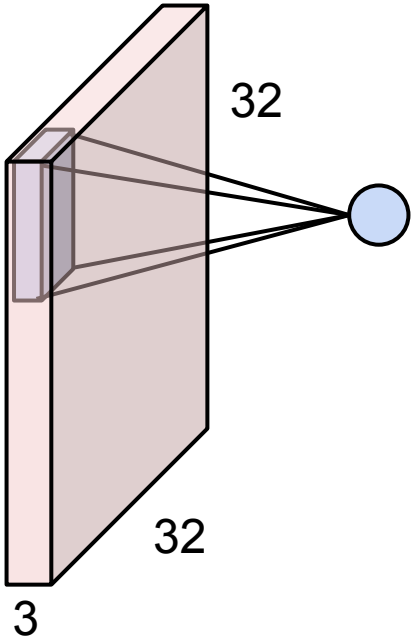

32x32x3 image

5x5x3 filter $w$

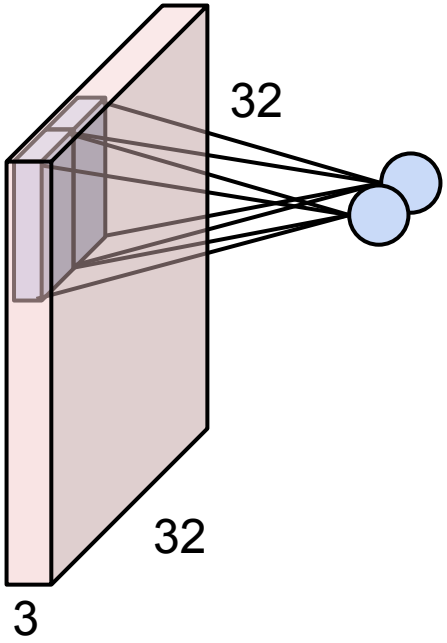
**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

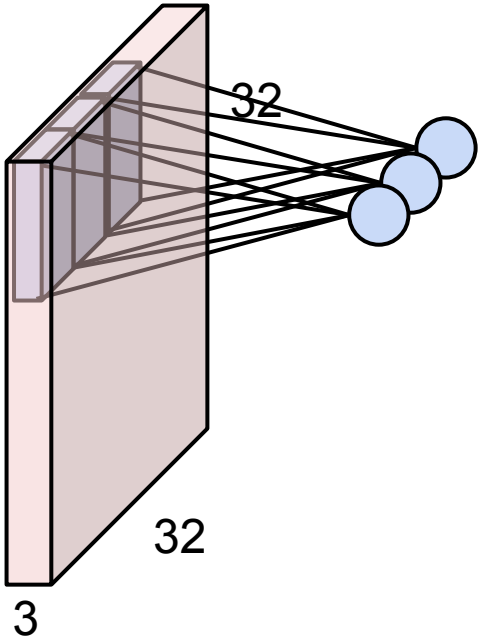
# Convolution Layer



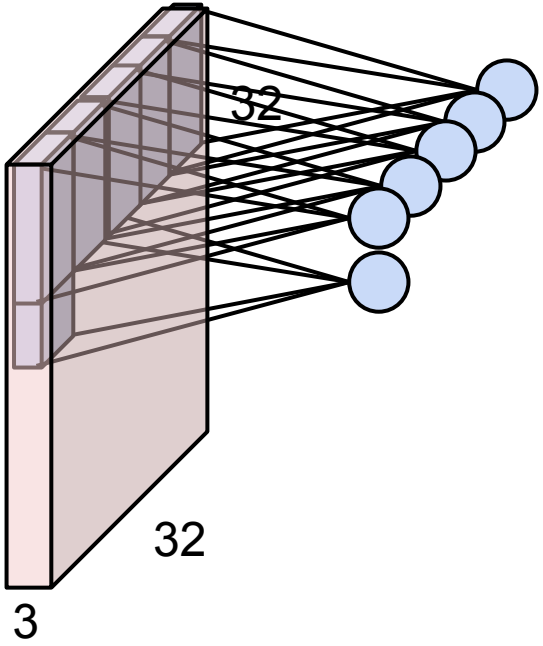
# Convolution Layer



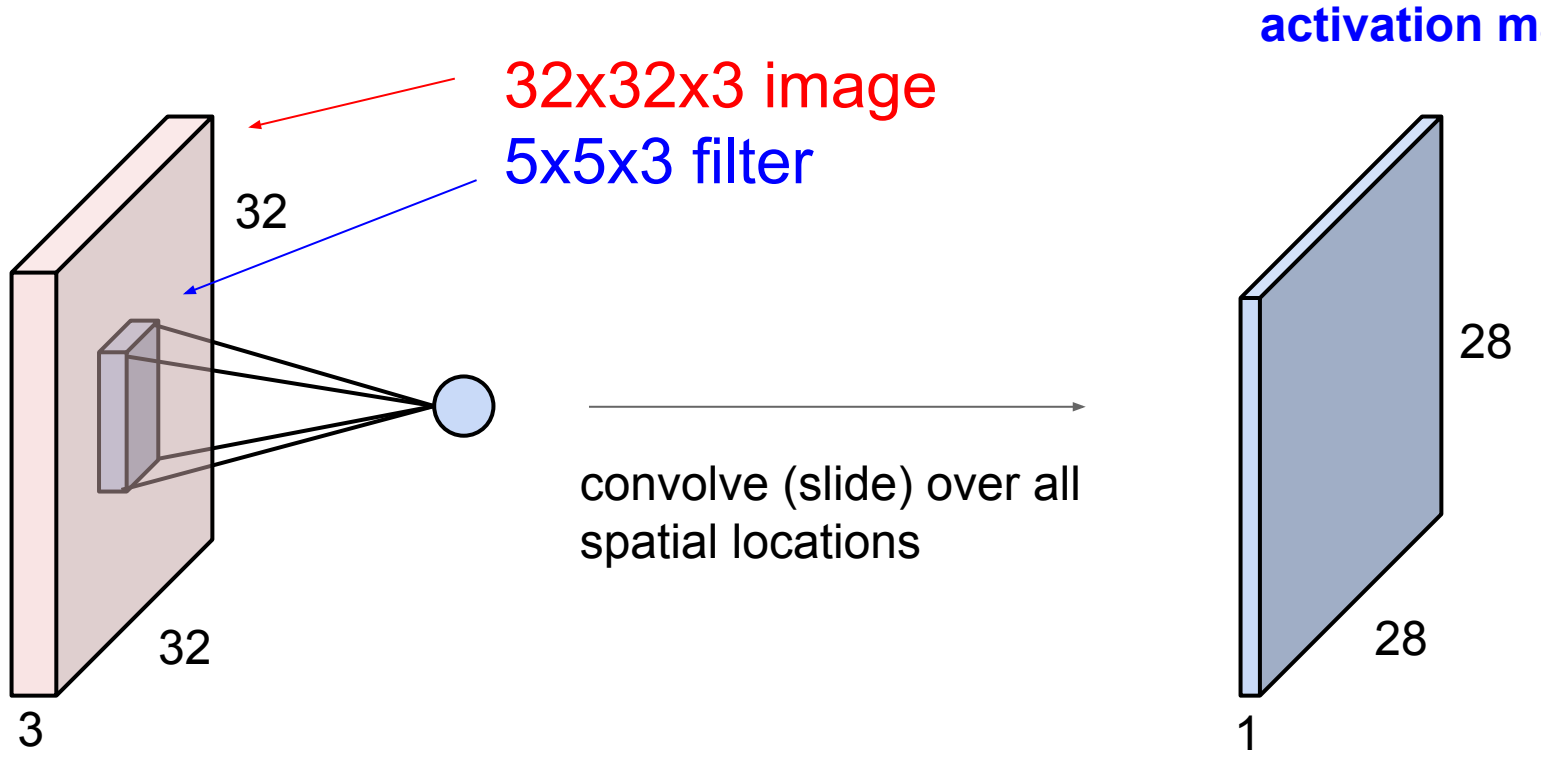
# Convolution Layer



# Convolution Layer

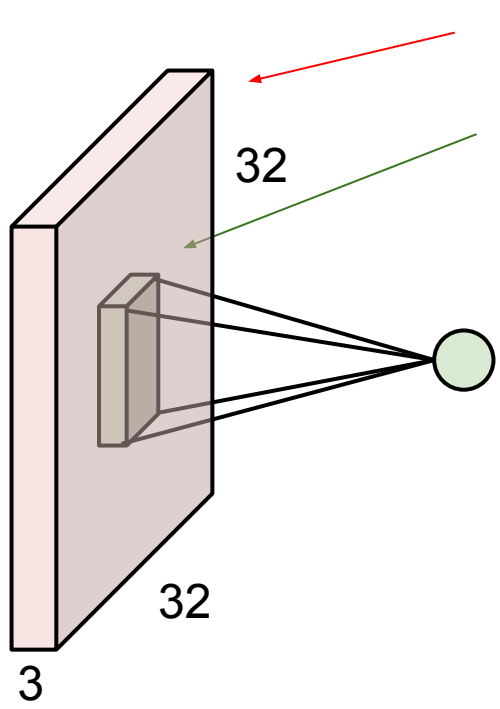


# Convolution Layer

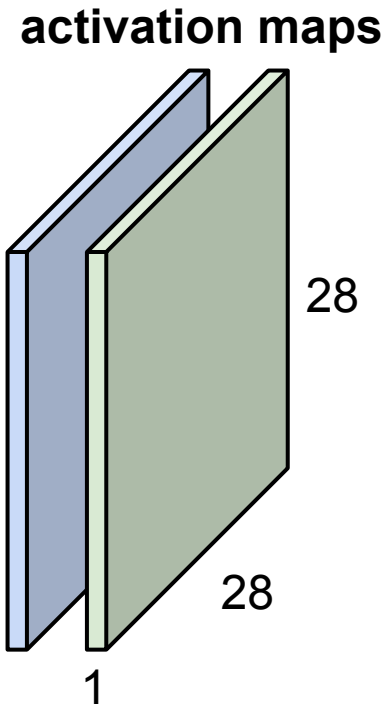


# Convolution Layer

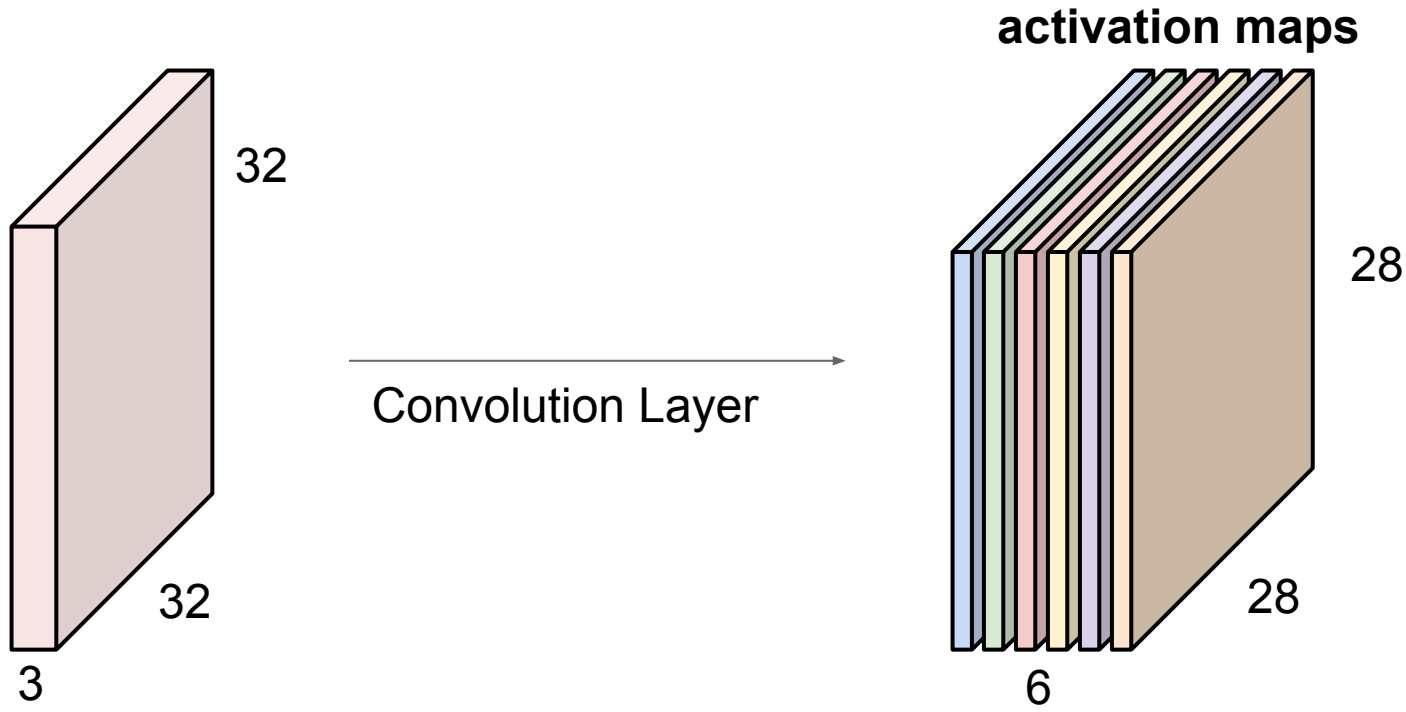
consider a second, green filter




convolve (slide) over all spatial locations

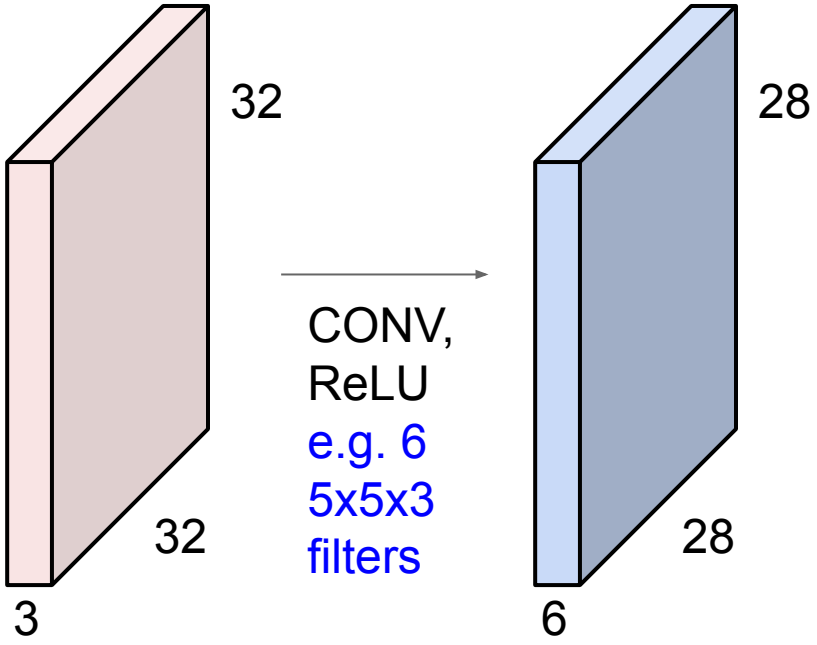


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

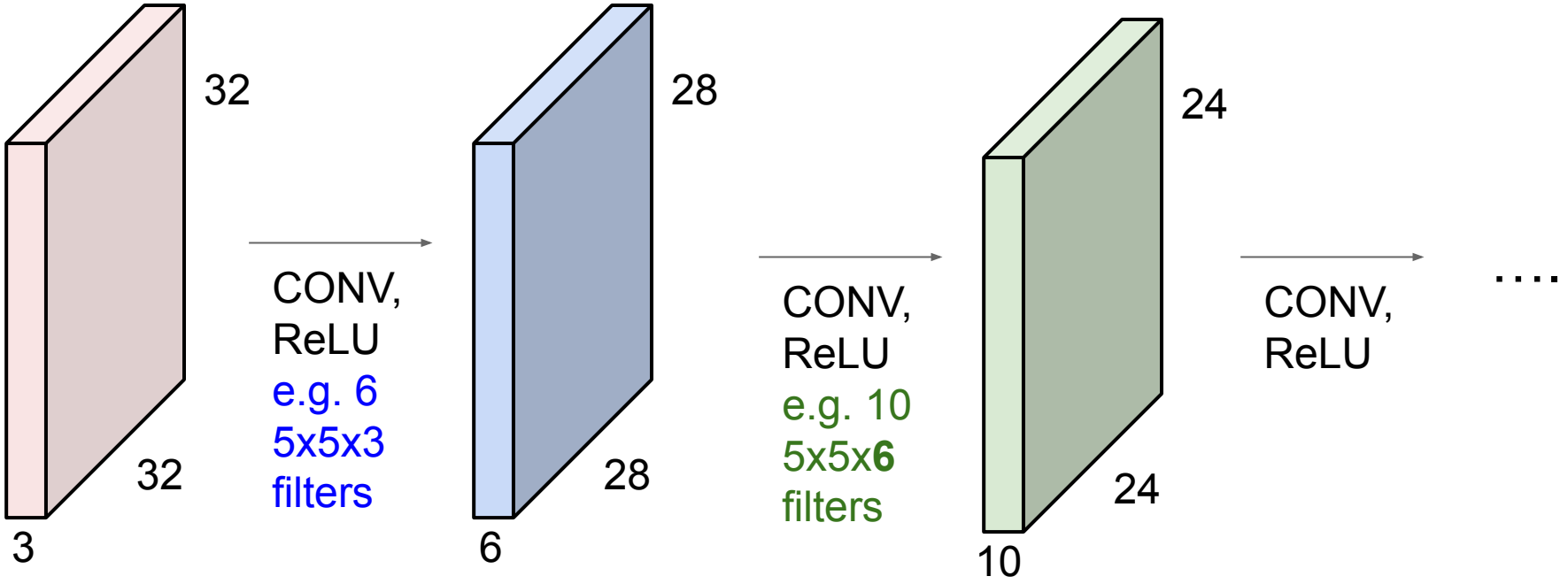



We stack these up to get a "new image" of size 28x28x6!

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



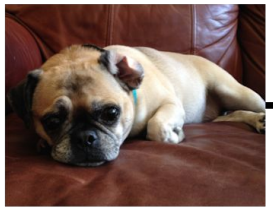
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions



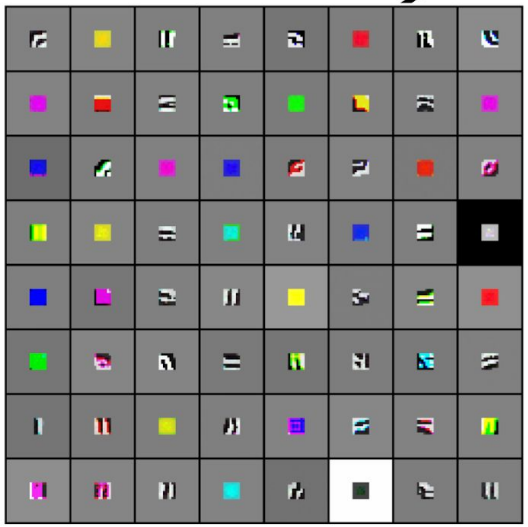
# Preview

*[Zeiler and Fergus 2013]*

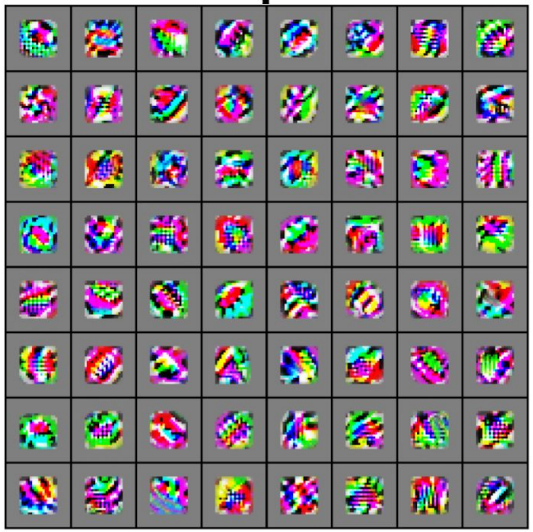
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



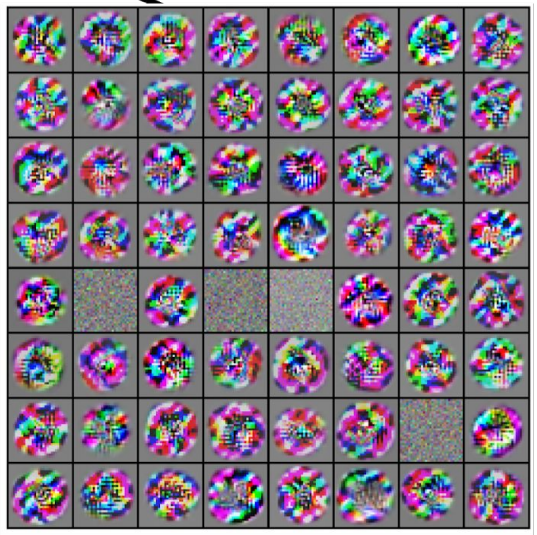
Low-level features → Mid-level features → High-level features → Linearly separable classifier



VGG-16 Conv1_1

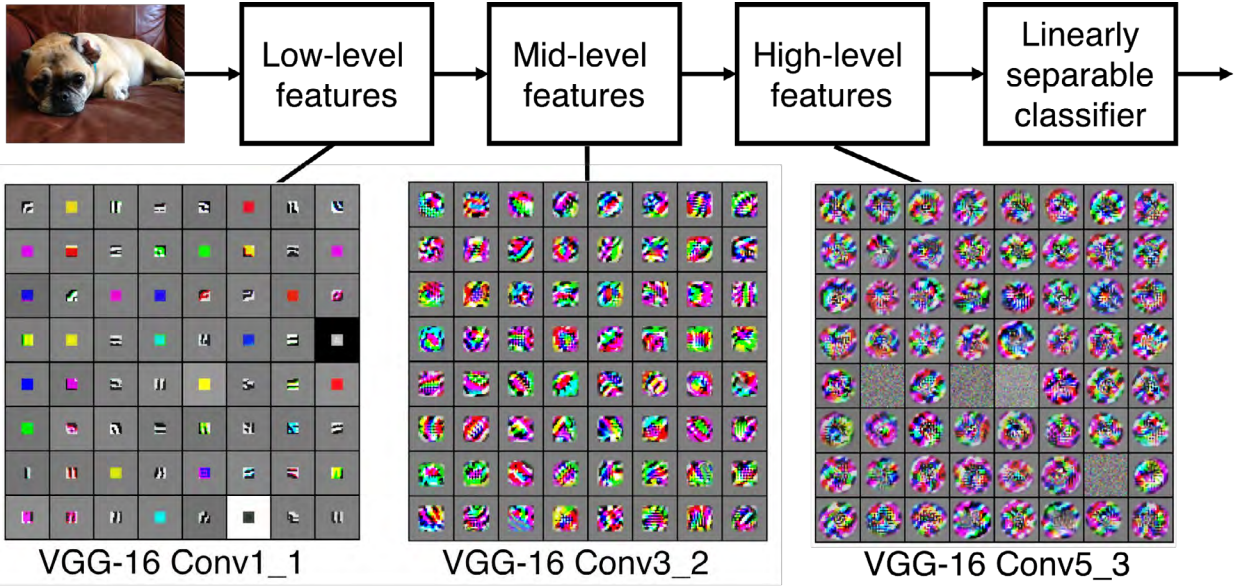


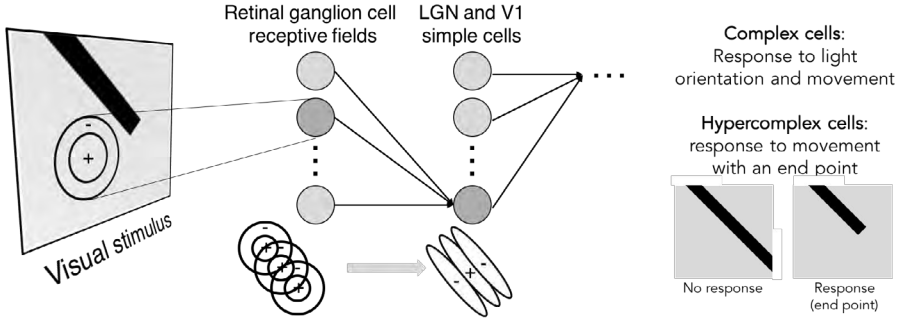
VGG-16 Conv3_2



VGG-16 Conv5_3

# Preview


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3


Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells:
Response to light orientation and movement
Hypercomplex cells:
response to movement with an end point
No response
Response (end point)

one filter =>
one activation map

example 5x5 filters
(32 total)

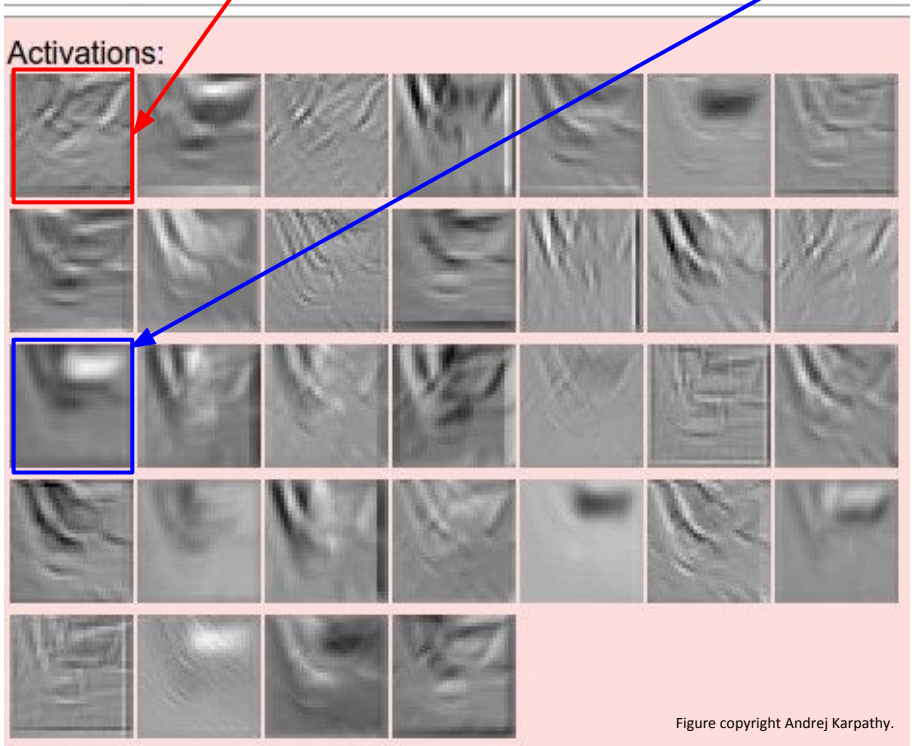


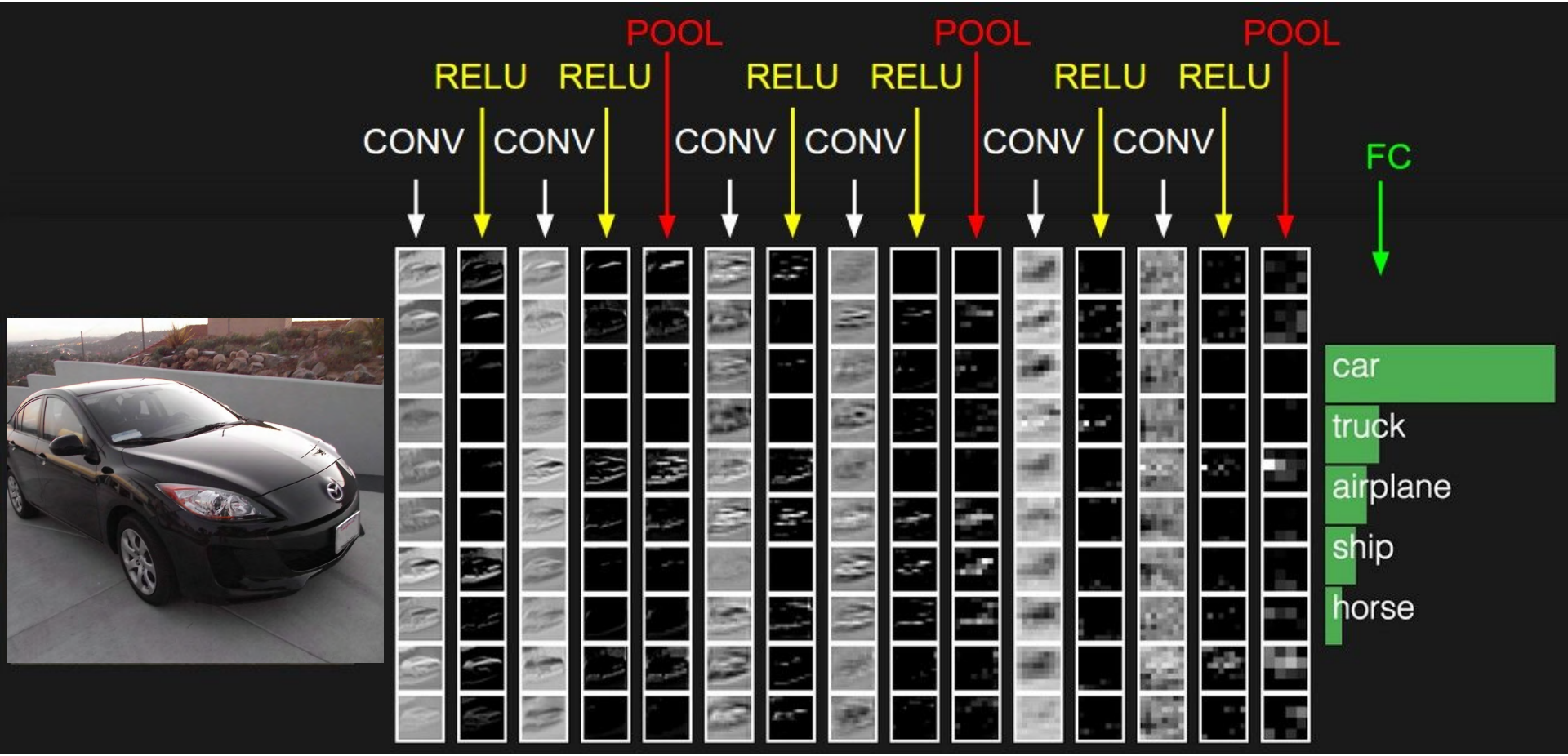

Figure copyright Andrej Karpathy.

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

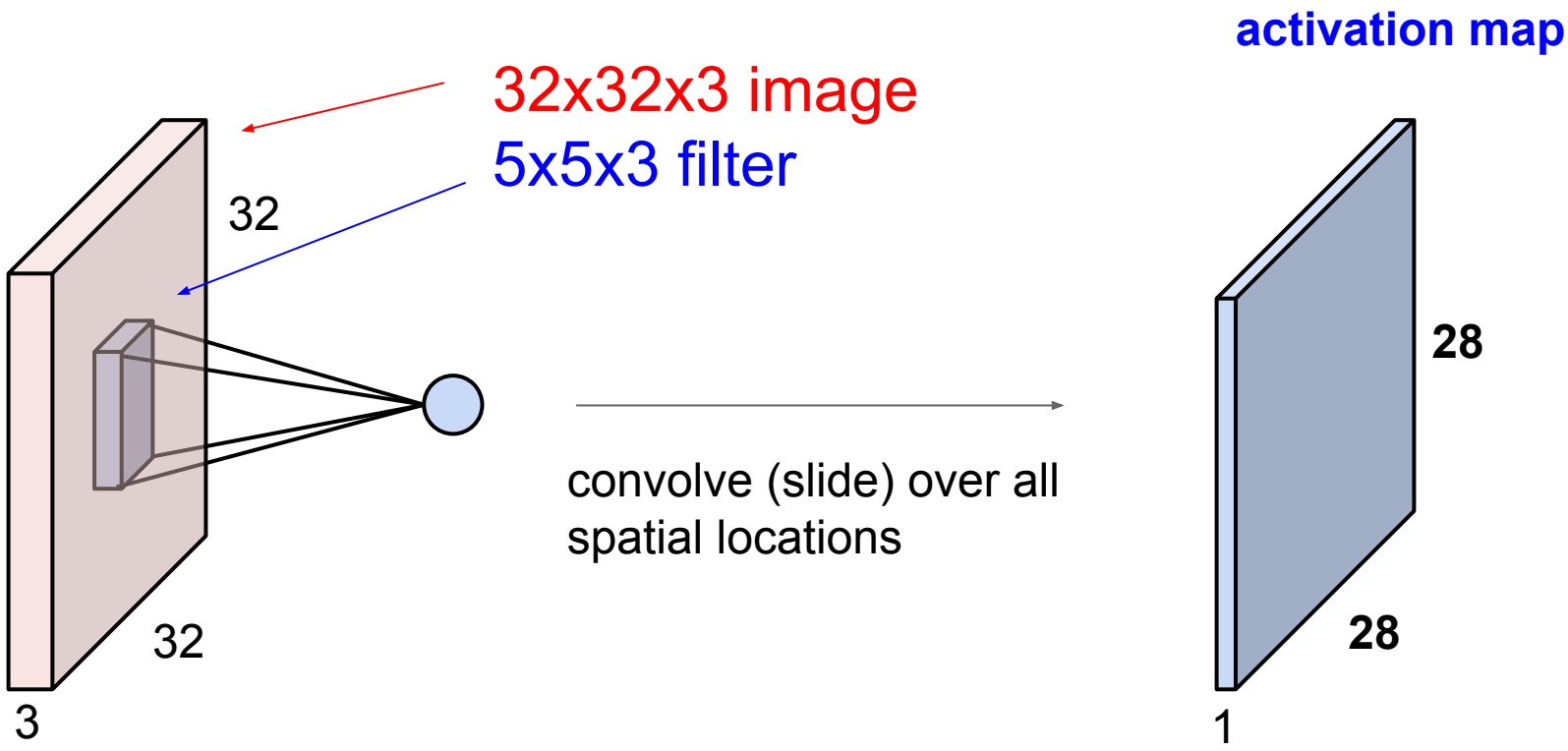
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

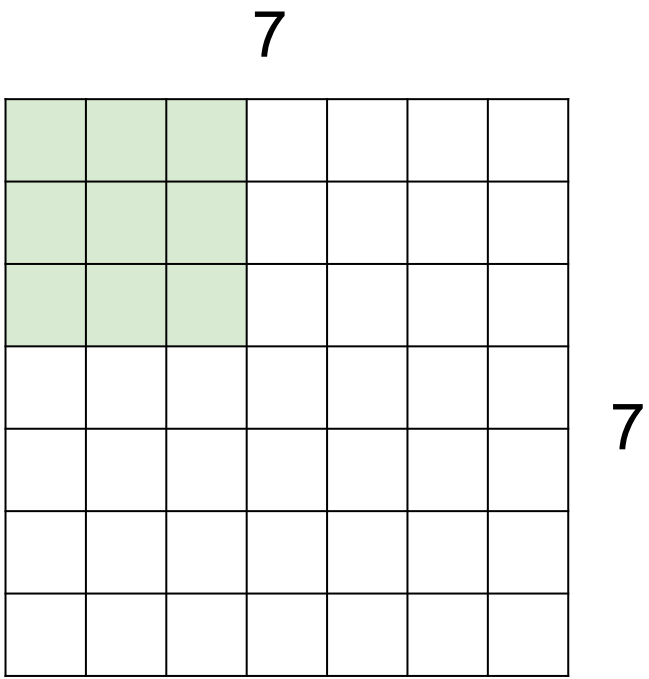
preview:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

# A closer look at spatial dimensions:

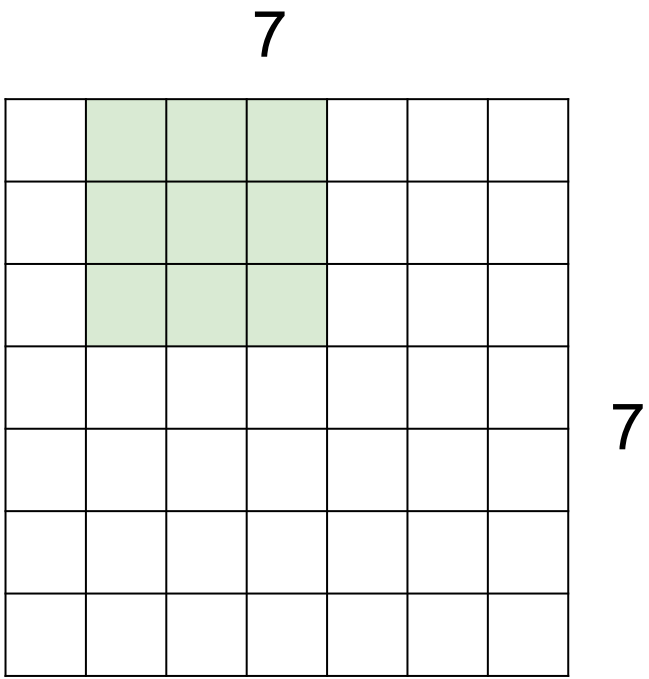


# A closer look at spatial dimensions:



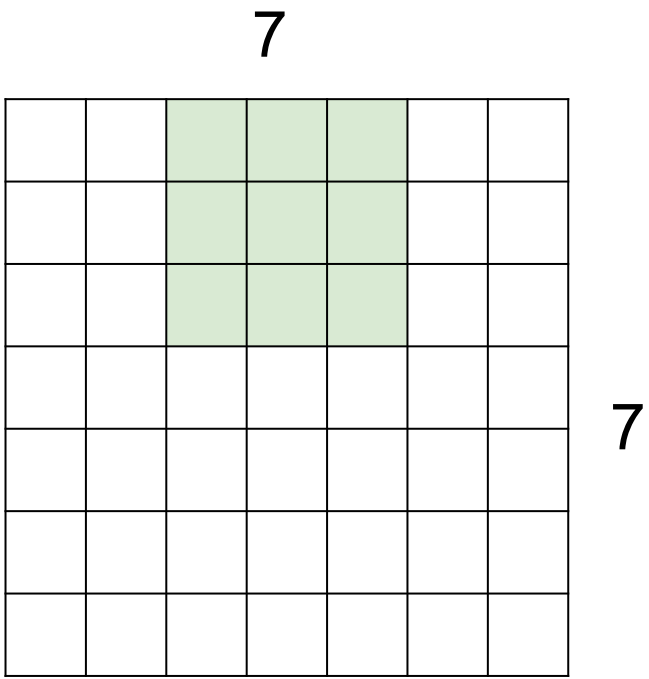

7x7 input (spatially)
assume 3x3 filter

# A closer look at spatial dimensions:



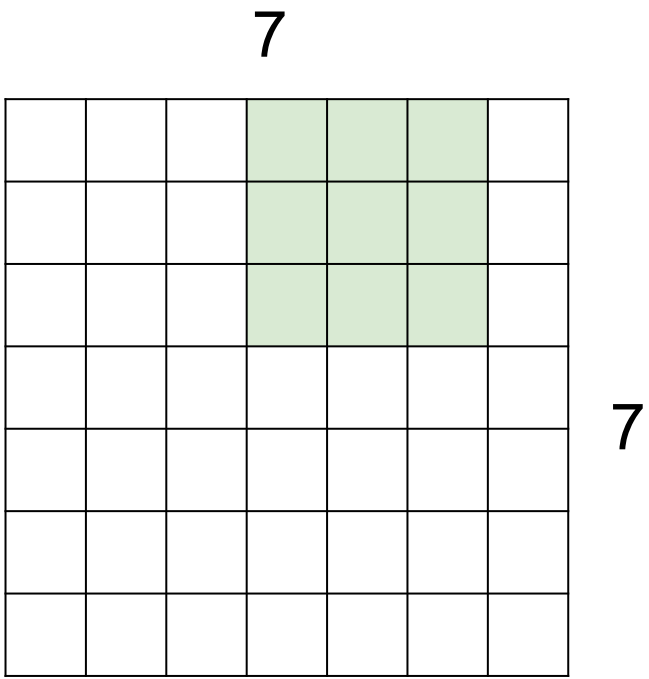

7x7 input (spatially)
assume 3x3 filter

# A closer look at spatial dimensions:



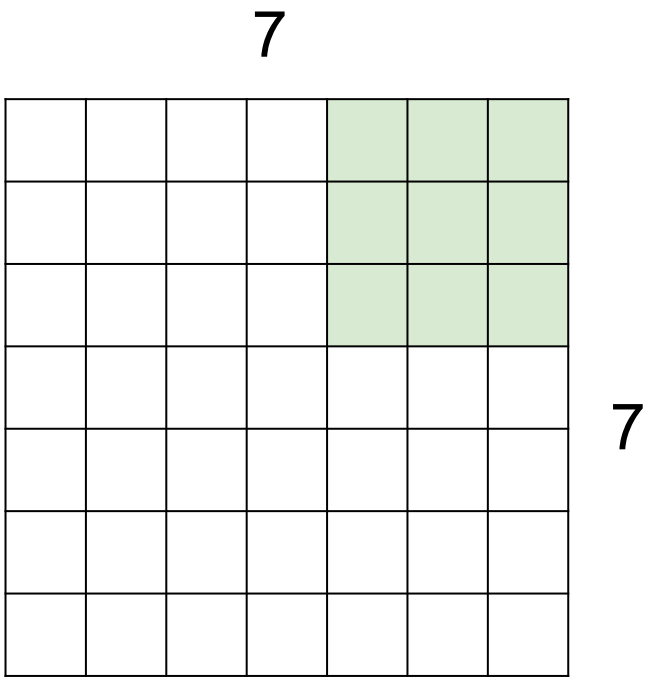

7x7 input (spatially)
assume 3x3 filter

# A closer look at spatial dimensions:




7x7 input (spatially)
assume 3x3 filter

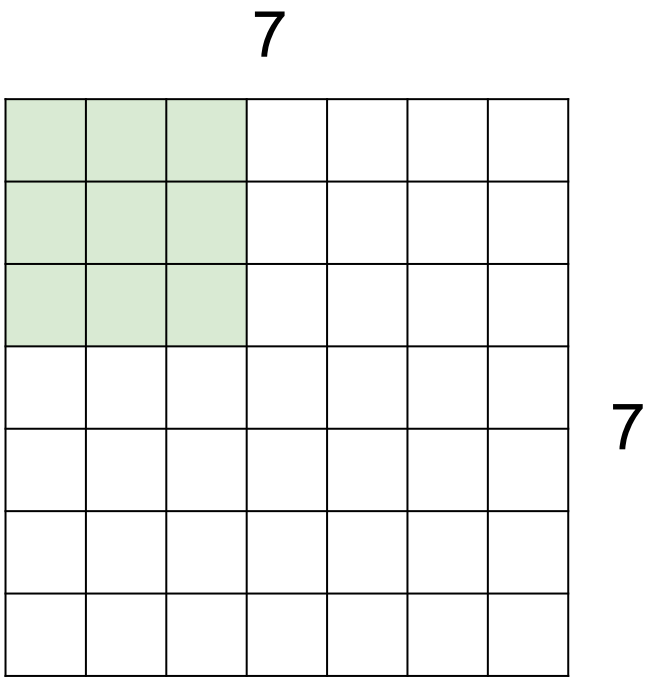
# A closer look at spatial dimensions:




7x7 input (spatially)
assume 3x3 filter

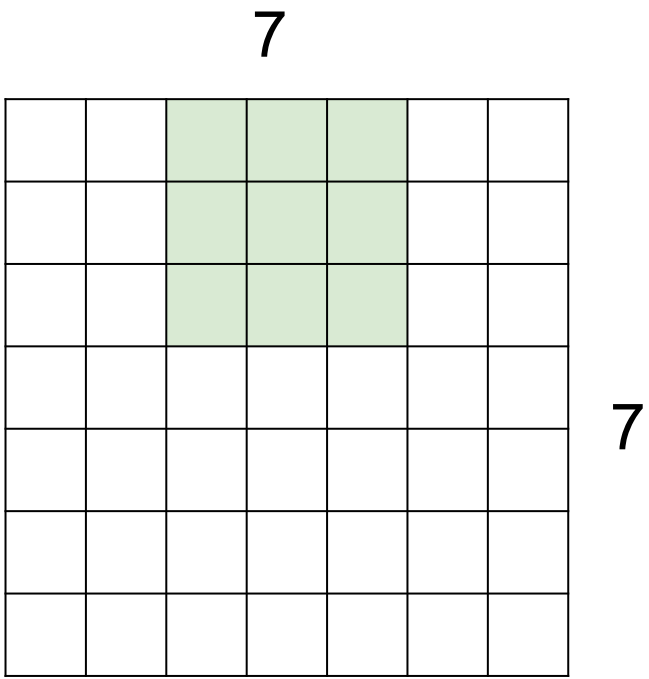
**=> 5x5 output**

# A closer look at spatial dimensions:



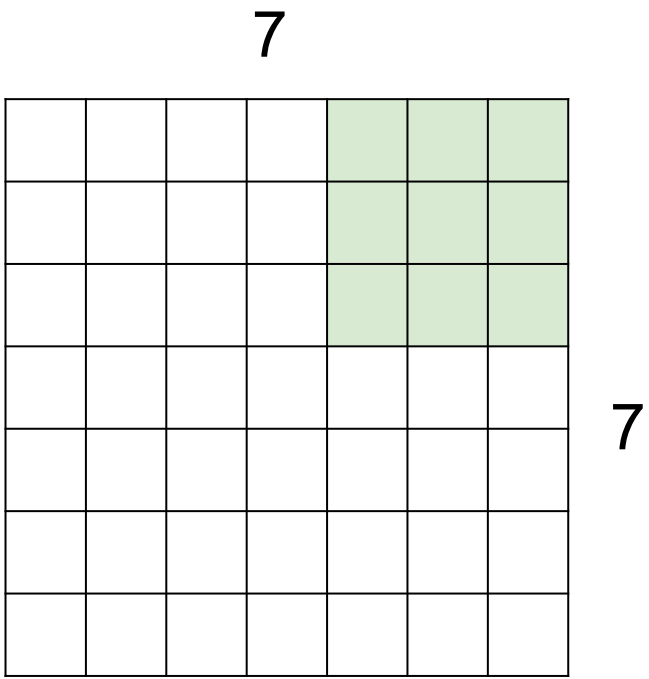

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

# A closer look at spatial dimensions:



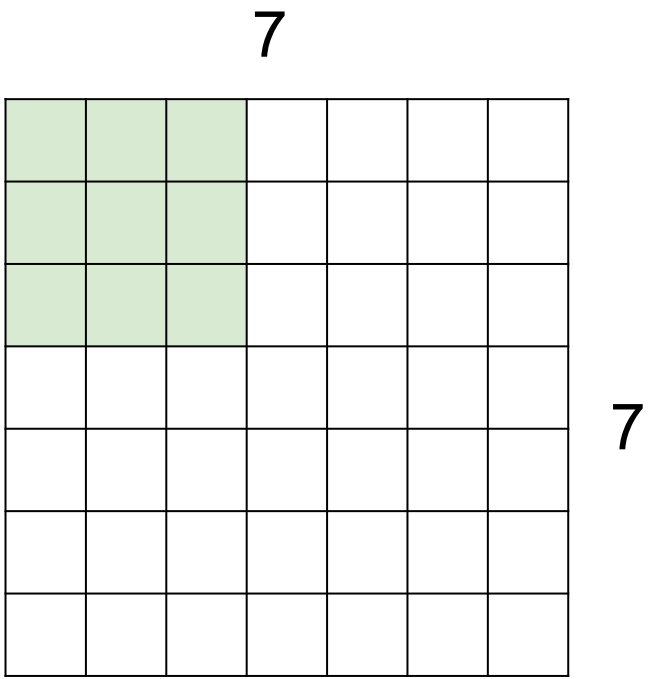

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

# A closer look at spatial dimensions:



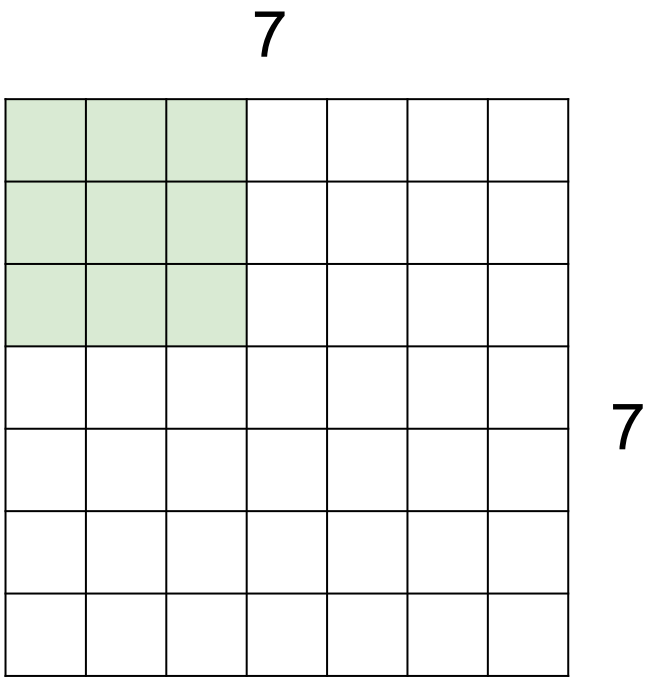

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
**=> 3x3 output!**

# A closer look at spatial dimensions:



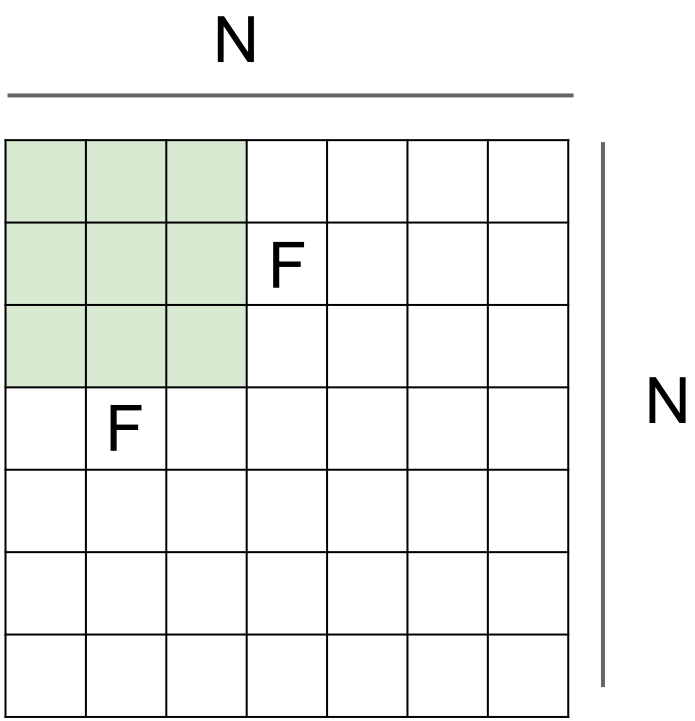

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

# A closer look at spatial dimensions:




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

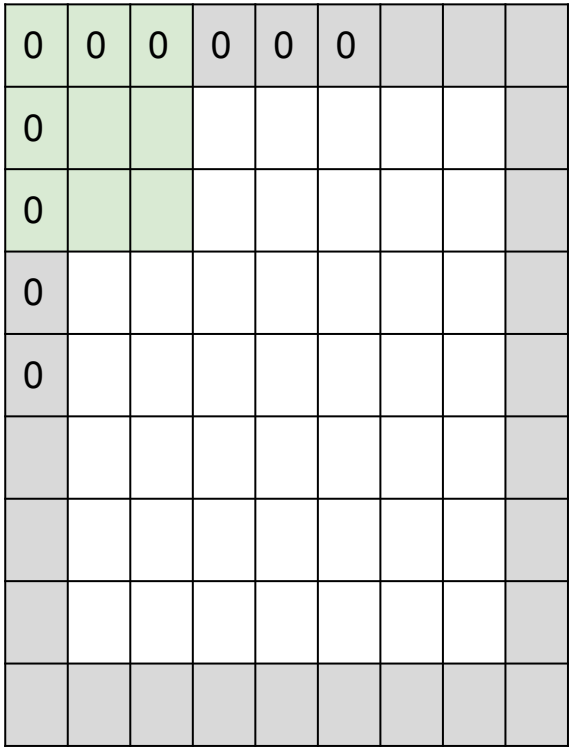
**doesn't fit!**
cannot apply 3x3 filter on
7x7 input with stride 3.




Output size:
**(N - F) / 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 :\

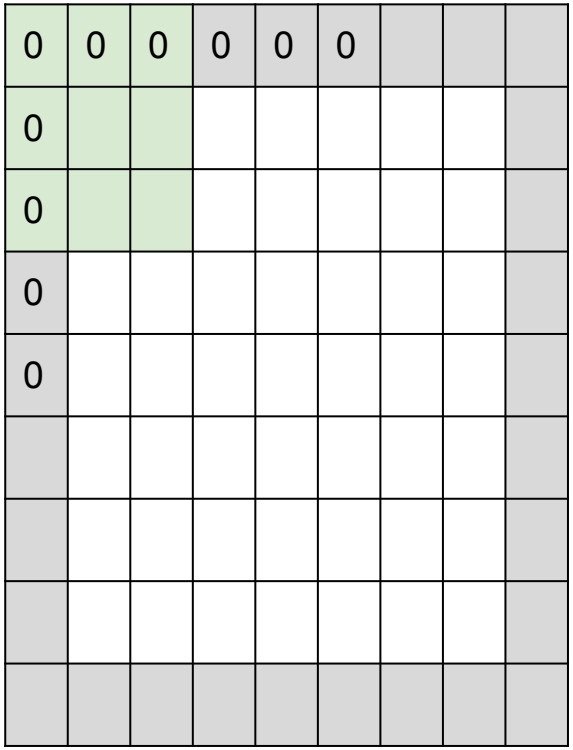
# In practice: Common to zero pad the border




e.g. input 7x7
**3x3** filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

(recall:)
(N - F) / stride + 1

# In practice: Common to zero pad the border

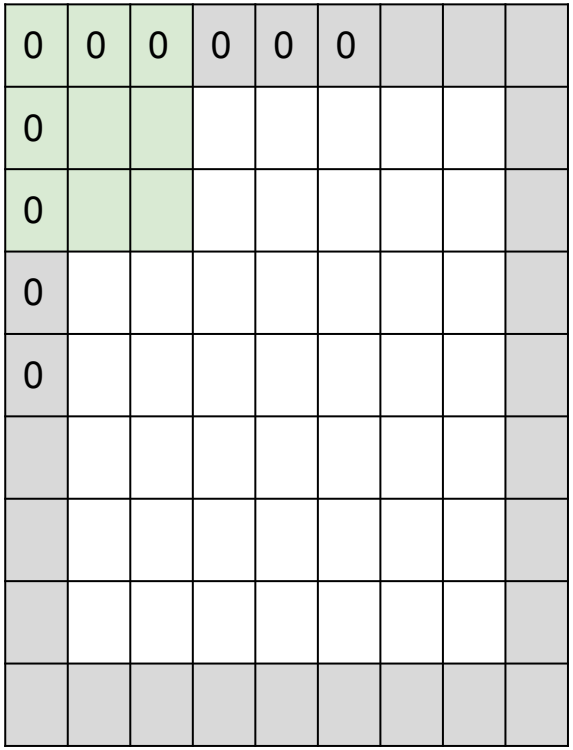



e.g. input 7x7
**3x3** filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

**7x7 output!**

(recall:)
(N + 2P - F) / stride + 1

# In practice: Common to zero pad the border

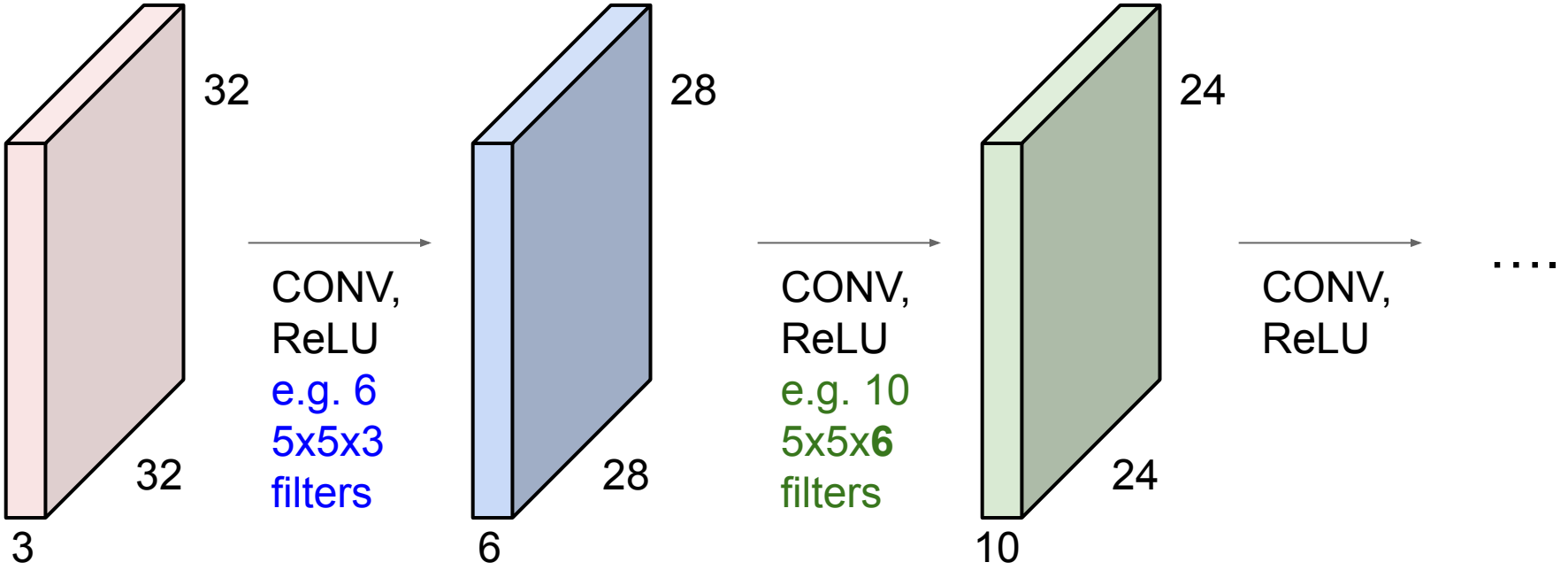



e.g. input 7x7
**3x3** filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

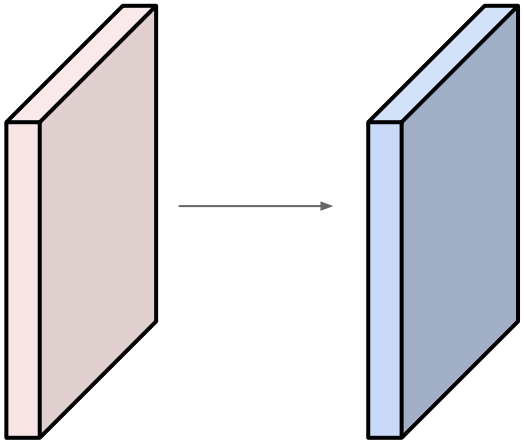
**7x7 output!**
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

**Remember back to…**

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



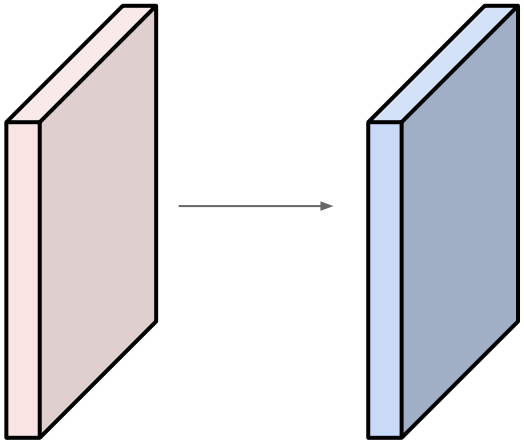
Examples time:



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

Output volume size: ?

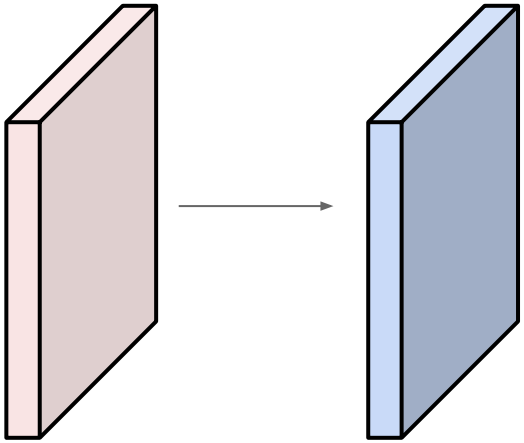
Examples time:



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

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
**32x32x10**

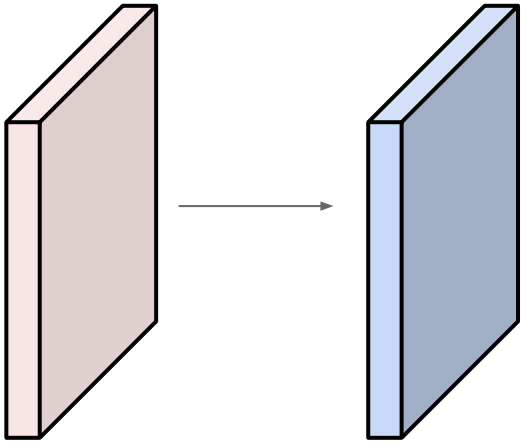
Examples time:



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 $W_1$ x $H_1$ x 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 $W_2$ x $H_2$ x 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

Let's assume input is $W_1$ x $H_1$ x 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 $W_2$ x $H_2$ x K
where:

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

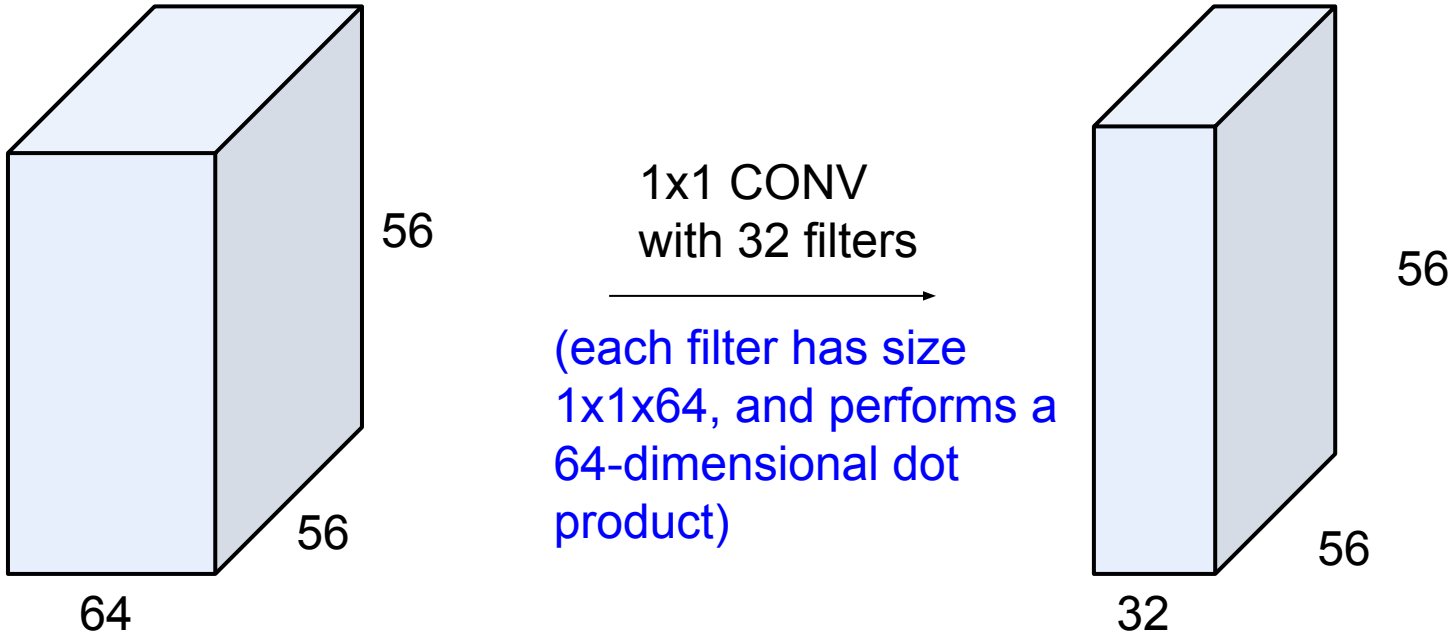
Number of parameters: $F^2CK$ and K biases

Common settings:

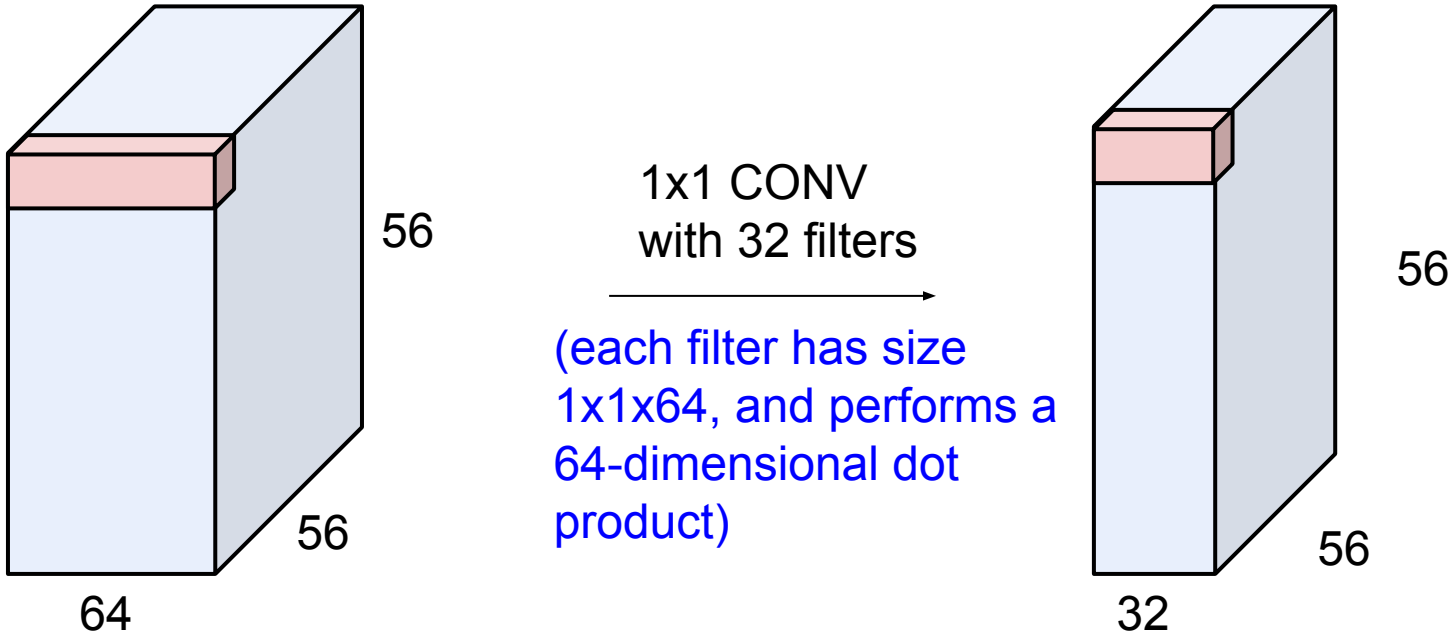
K = (powers of 2, e.g. 32, 64, 128, 512)

- 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

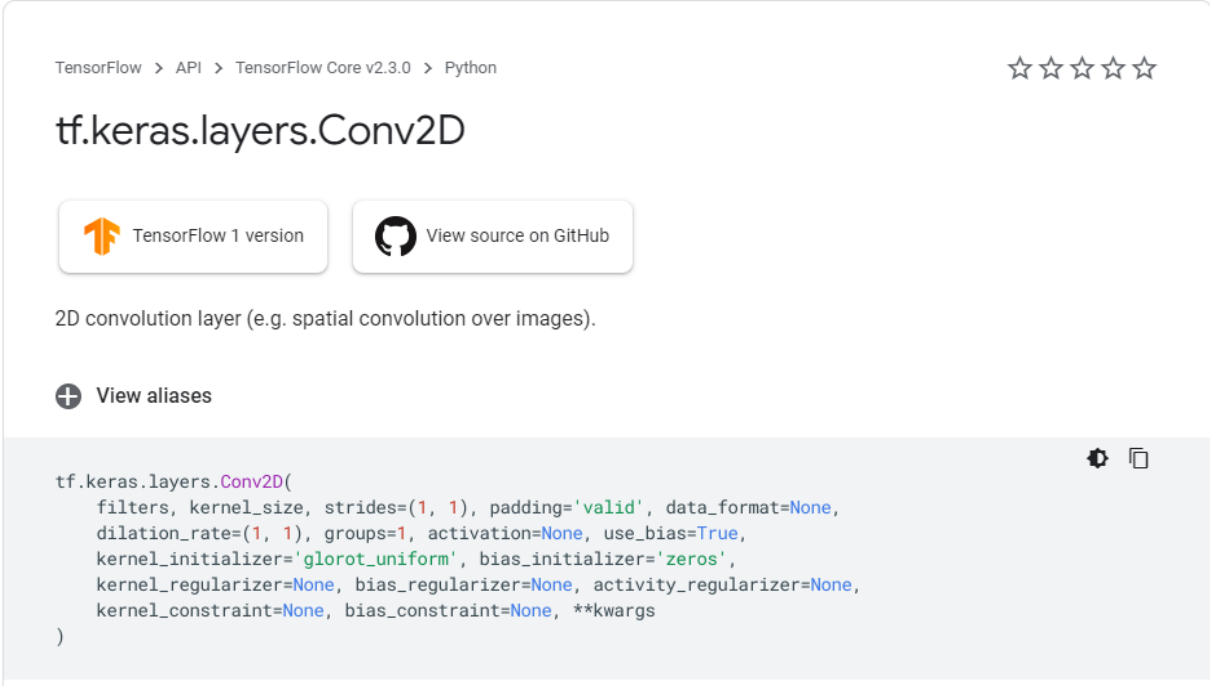
# (btw, 1x1 convolution layers make perfect sense)



# (btw, 1x1 convolution layers make perfect sense)



# Example: CONV layer in TF



TensorFlow > API > TensorFlow Core v2.3.0 > Python

tf.keras.layers.Conv2D

TensorFlow 1 version | View source on GitHub

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

View aliases

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, bias_constraint=None, **kwargs
)
```

Conv layer needs 4 hyperparameters:

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

Keras is licensed under the MIT license.

# Example: CONV layer in Keras

Conv layer needs 4 hyperparameters:

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

## 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 != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 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.

# Example: CONV layer in PyTorch

Conv layer needs 4 hyperparameters:

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

## 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 $(N, C_{\text{in}}, H, W)$ and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

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.

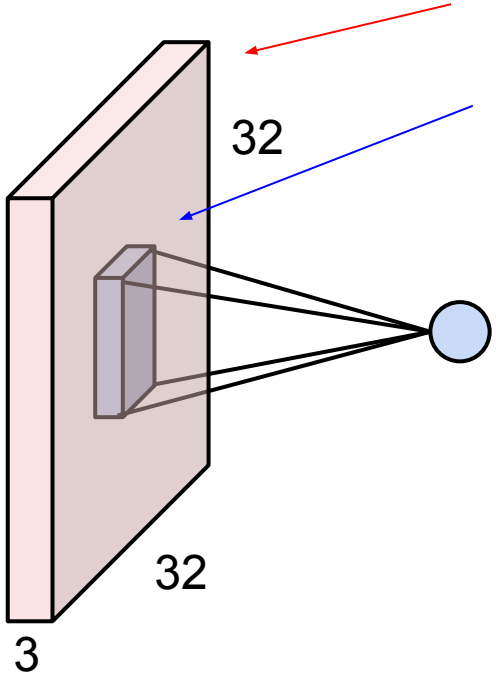
- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. 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: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

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.

# The brain/neuron view of CONV Layer

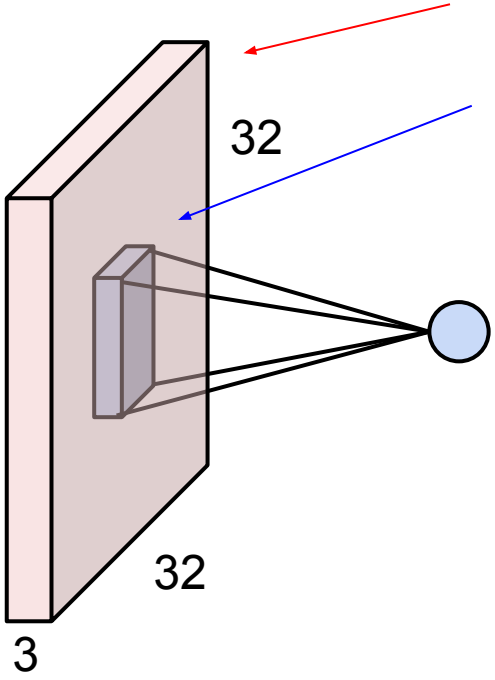



32x32x3 image

5x5x3 filter

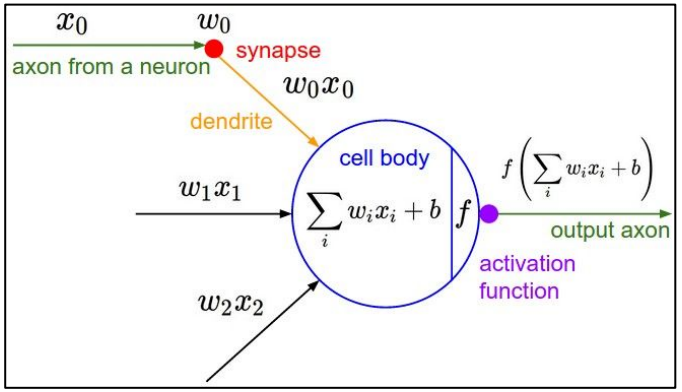
**1 number:**
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

# The brain/neuron view of CONV Layer



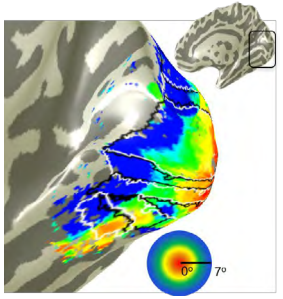

32x32x3 image

5x5x3 filter

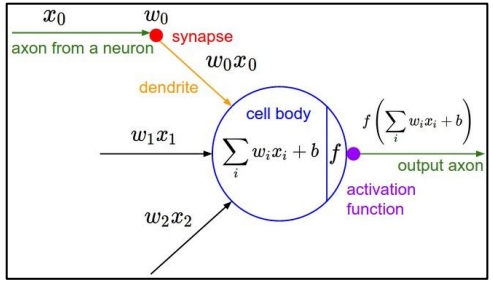


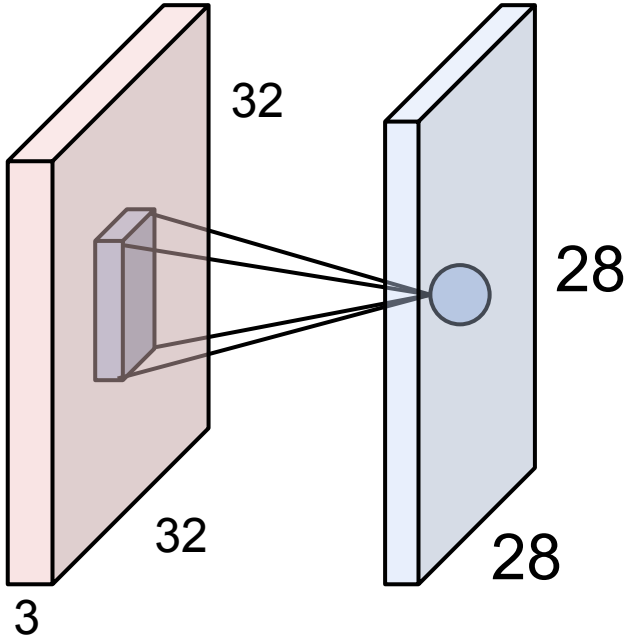

It's just a neuron with local connectivity...

**1 number:**
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



# The brain/neuron view of CONV Layer



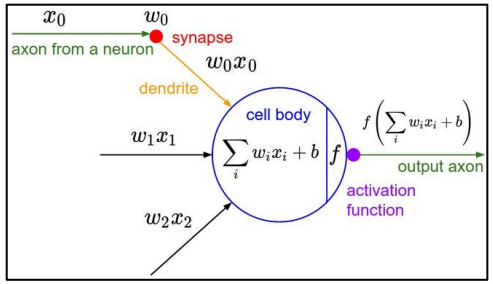



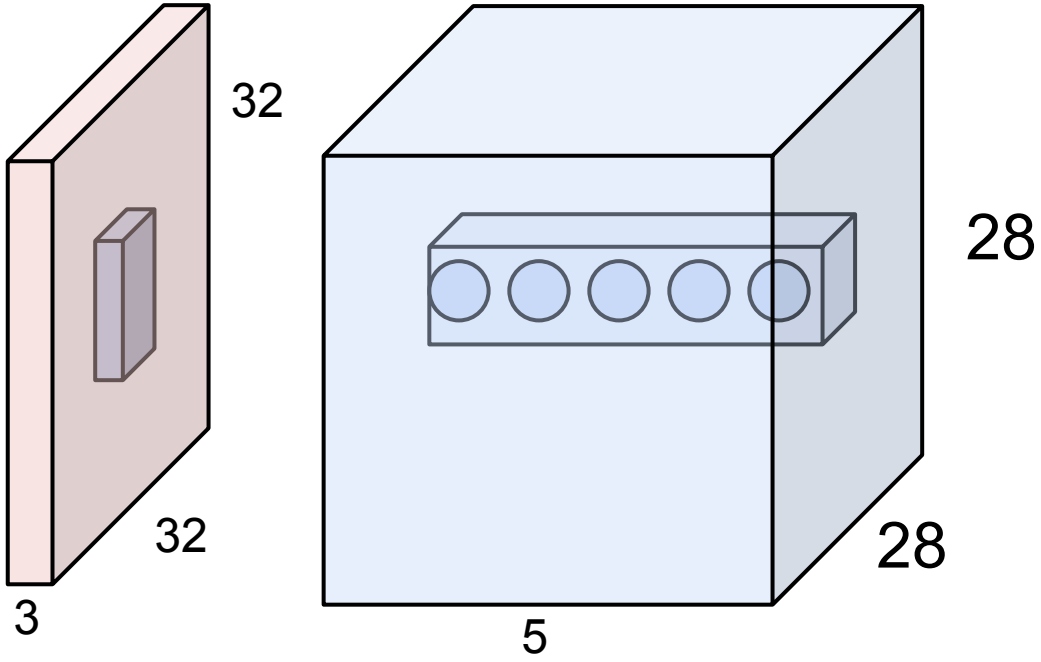

An activation map is a 28x28 sheet of neuron outputs:

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



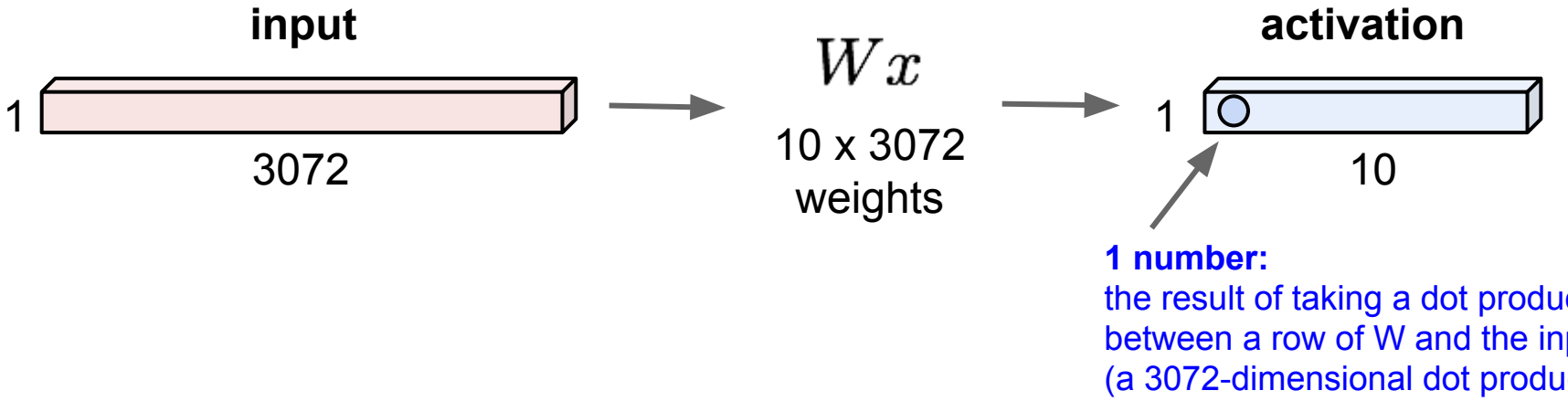

E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

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

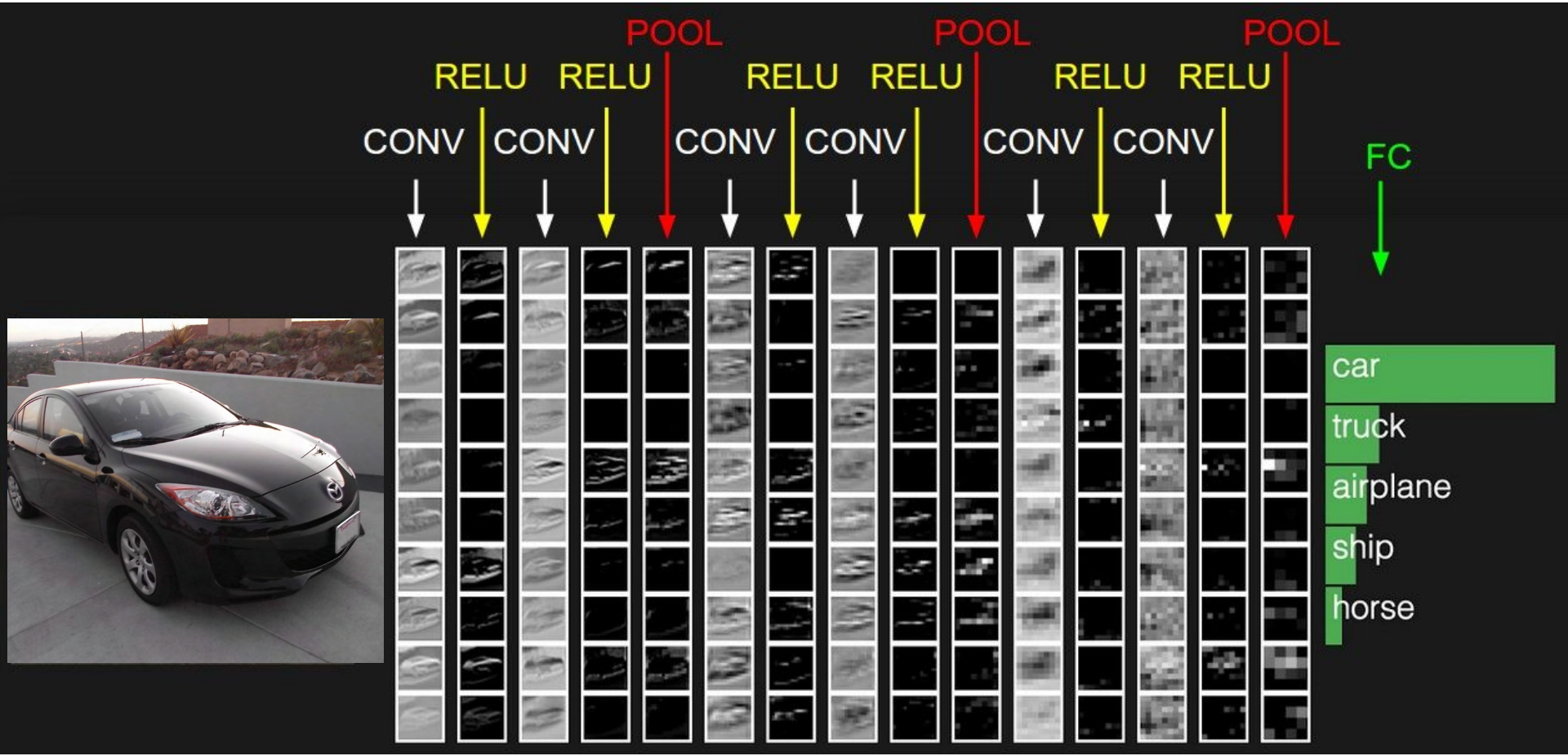
# Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

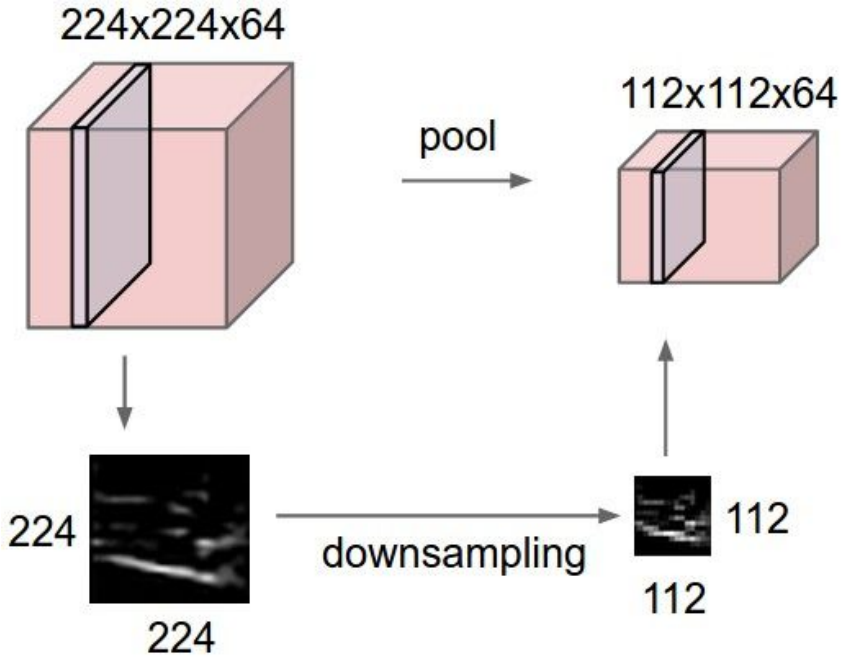


two more layers to go: POOL/FC

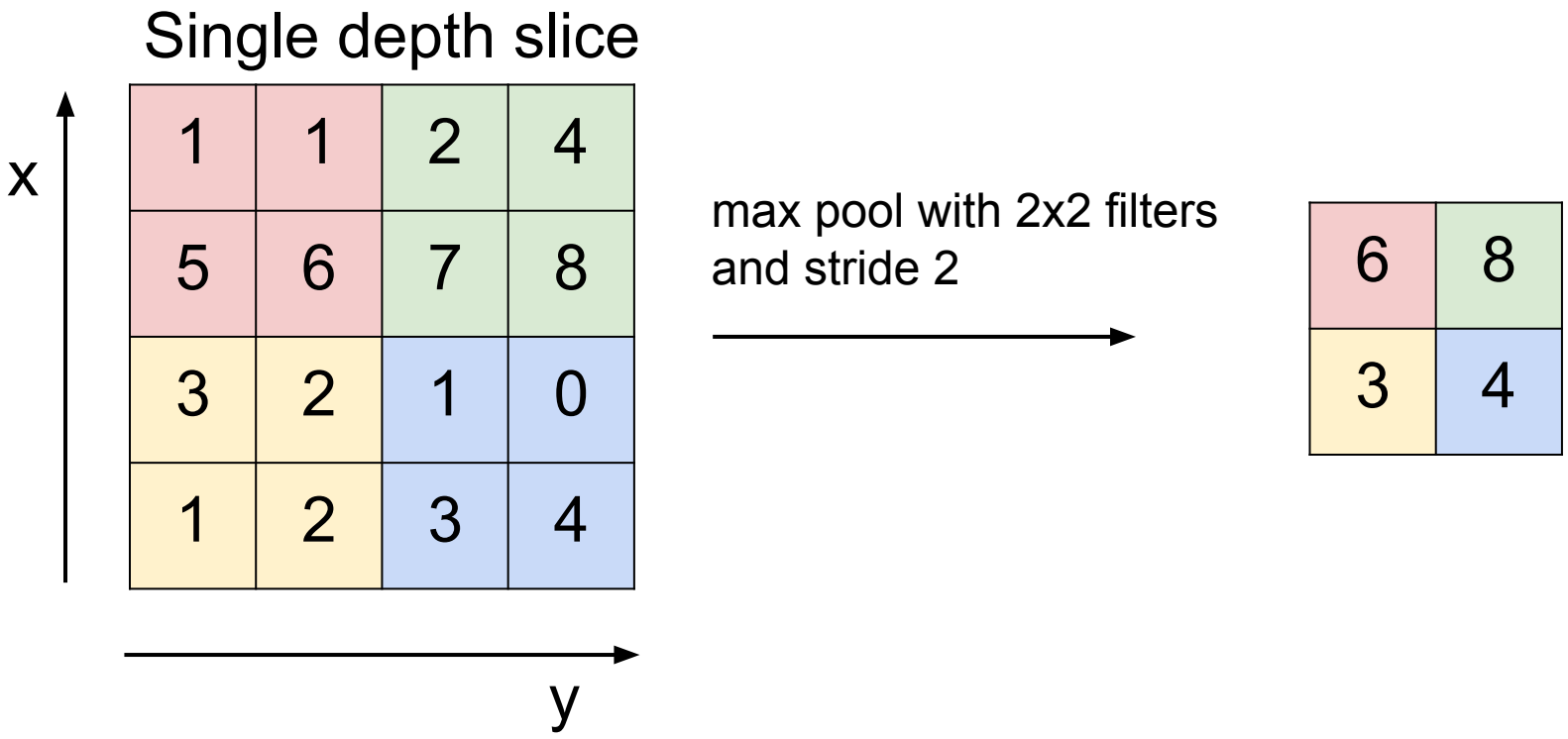


# Pooling layer

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



# MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

# Pooling layer: summary

Let's assume input is $W_1$ x $H_1$ x C
Conv layer needs 2 hyperparameters:

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

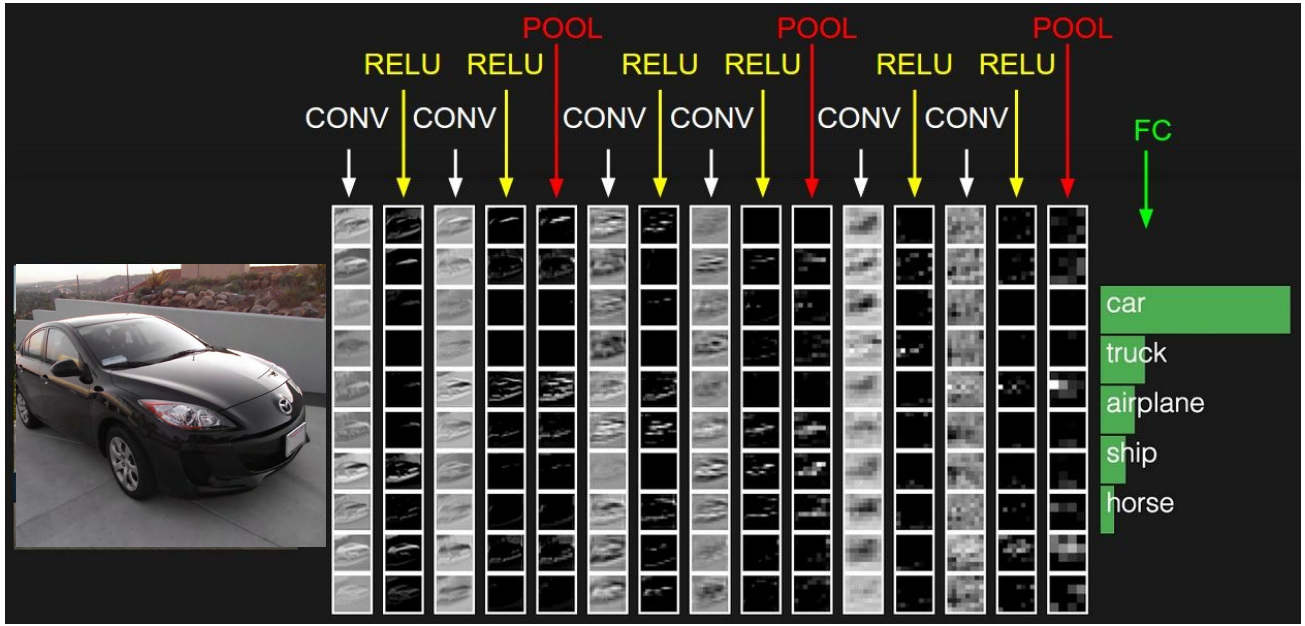
This will produce an output of $W_2$ x $H_2$ x C where:

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

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 verically.

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

  **[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX**

  where N is usually up to ~5, M is large, 0 <= K <= 2.
  - but recent advances such as ResNet/GoogLeNet have challenged this paradigm

# Задача на дом:

Входное изображение:

[1 2 3 4 5]
[2 2 1 1 1]
[3 2 1 1 1]
[4 1 1 1 1]
[5 1 1 1 1]

CONV фильтр:

[0 -1 0]
[1  1 1]
[0 -1 0]

Посчитать выход сети: conv(depth=1, stride=2) -> ReLU -> MaxPool