中山大學
SUN YAT-SEN UNIVERSITY

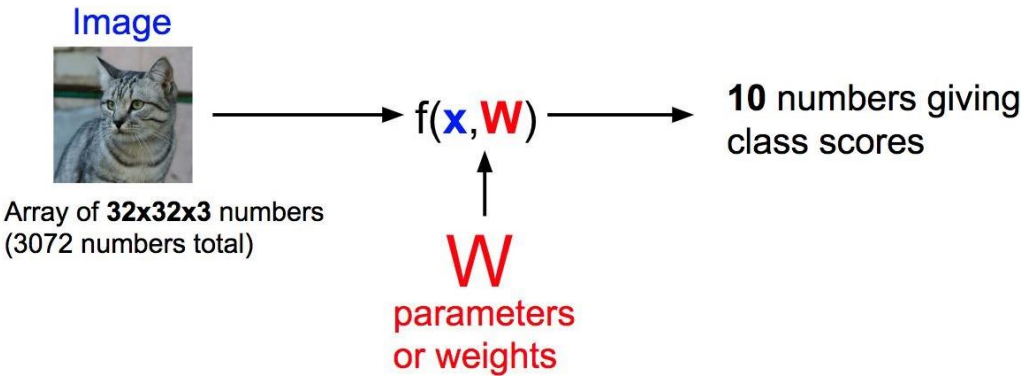
Lecture 19. Image Classification with CNNs

Pattern Recognition and Computer Vision

Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

扫码签到

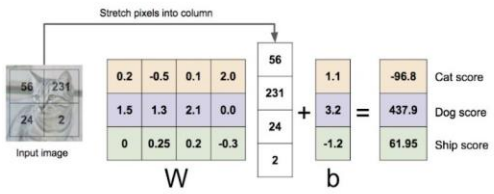
Recap: Image Classification with Linear Classifier



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

Algebraic Viewpoint

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

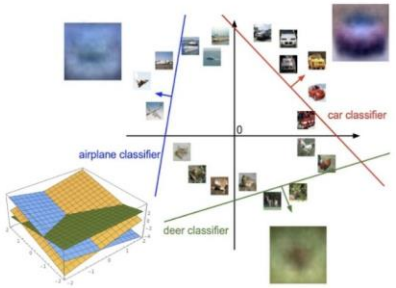


Visual Viewpoint

One template per class

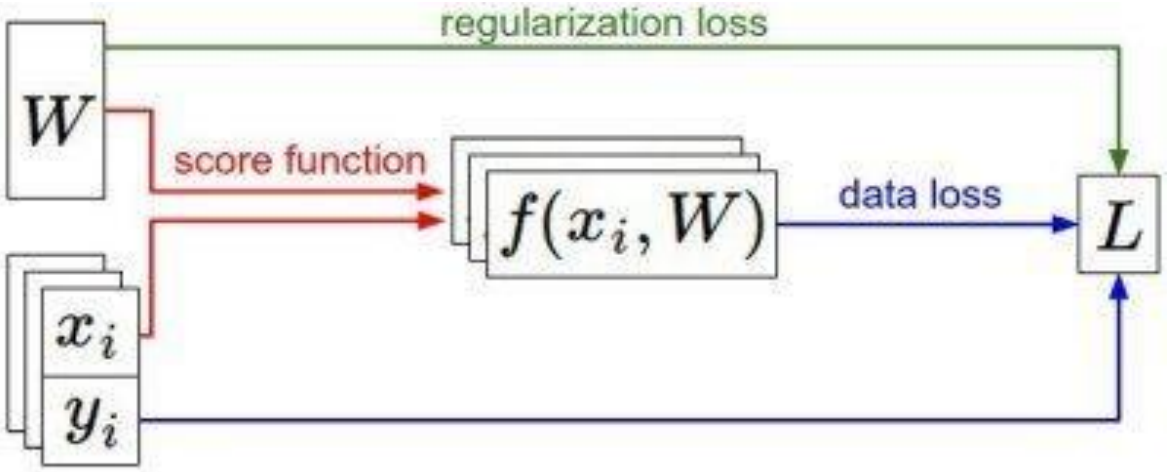
Geometric Viewpoint

Hyperplanes cutting up space

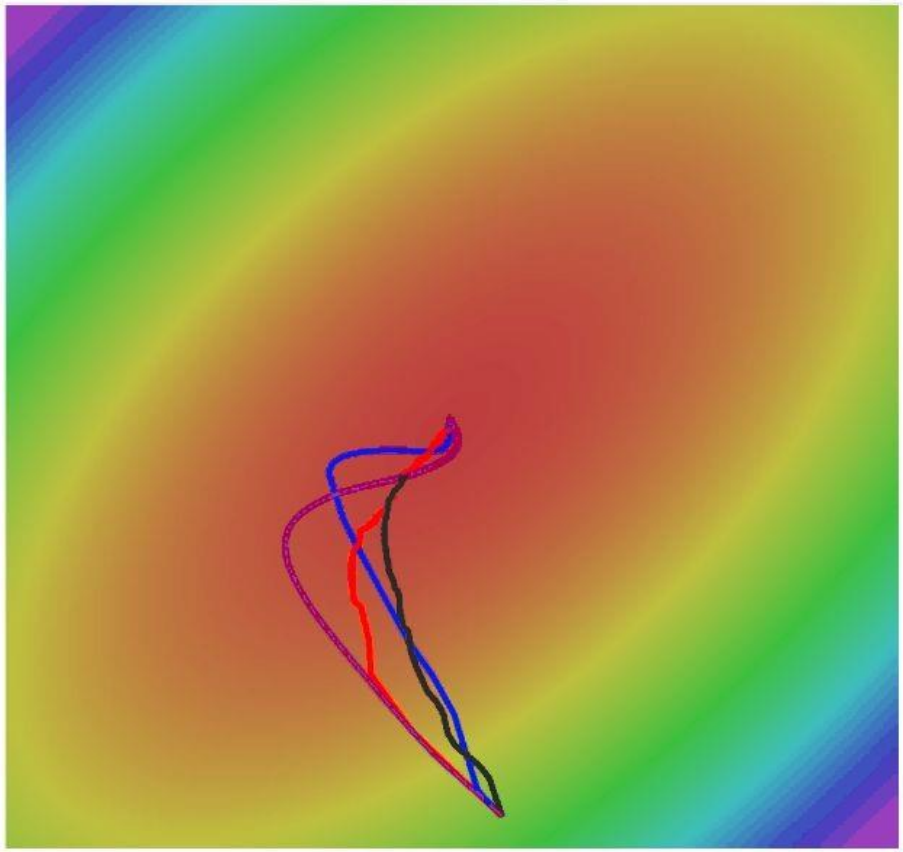


Recap: Loss Function

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) = Wx$
- We have a **loss function:**
- **Softmax:** $L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$
- **SVM:** $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
- **Full loss:** $L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$



Recap: Optimization



SGD

SGD+Momentum

RMSProp

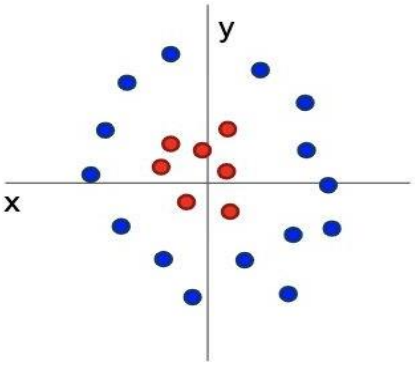
Adam

Problem: Linear Classifiers are not very powerful

Visual Viewpoint

Linear classifiers learn one template per class

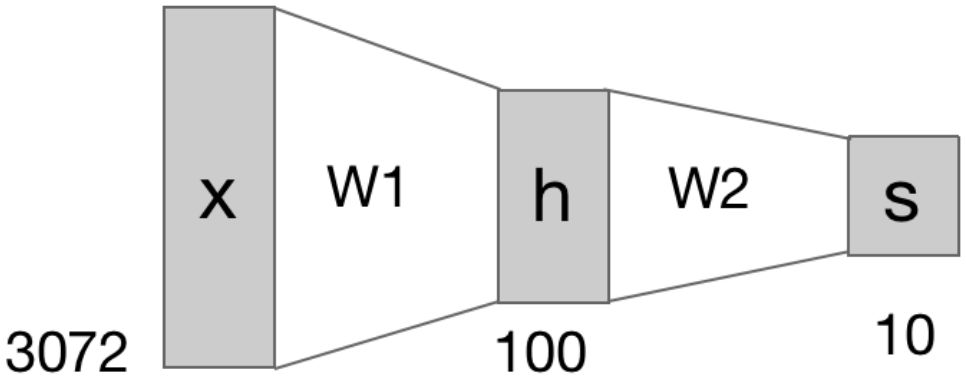
Geometric Viewpoint

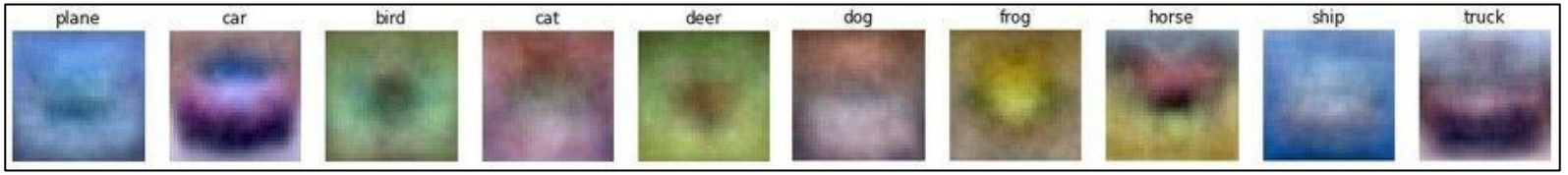


Linear classifiers can only draw linear decision boundaries

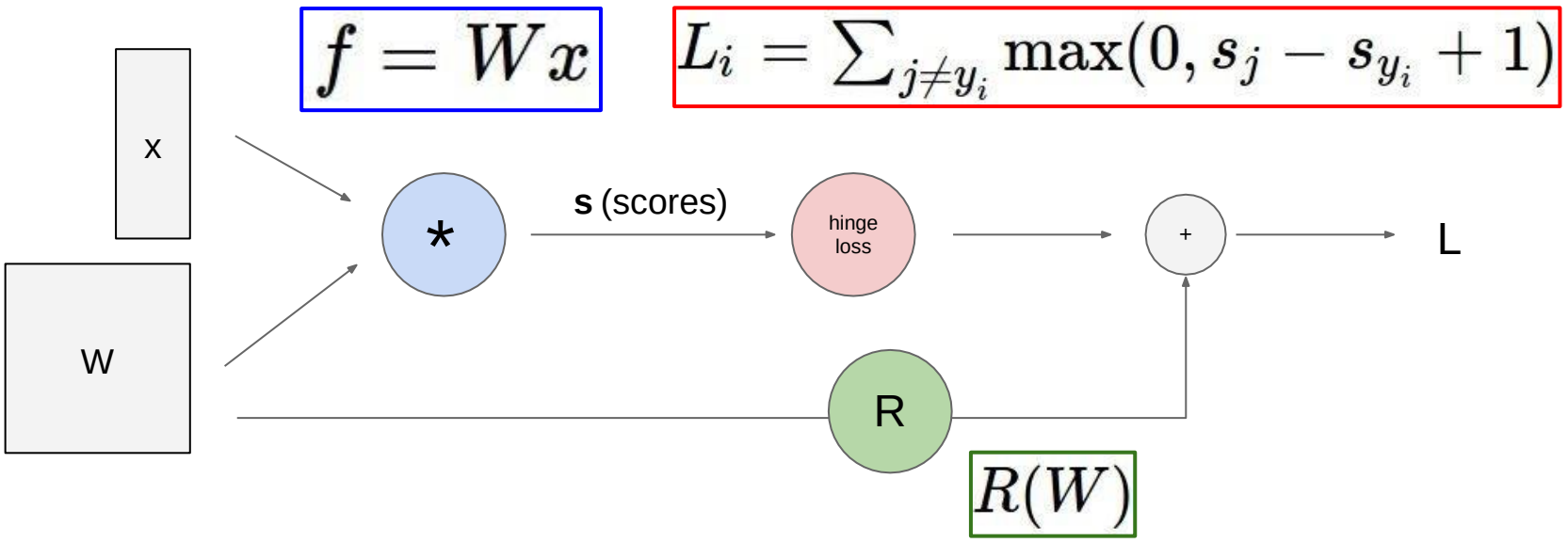
Last time: Neural Networks

- Linear score function: $f = Wx$
- 2-layer Neural Network: $f = W_2 \max(0, W_1 x)$

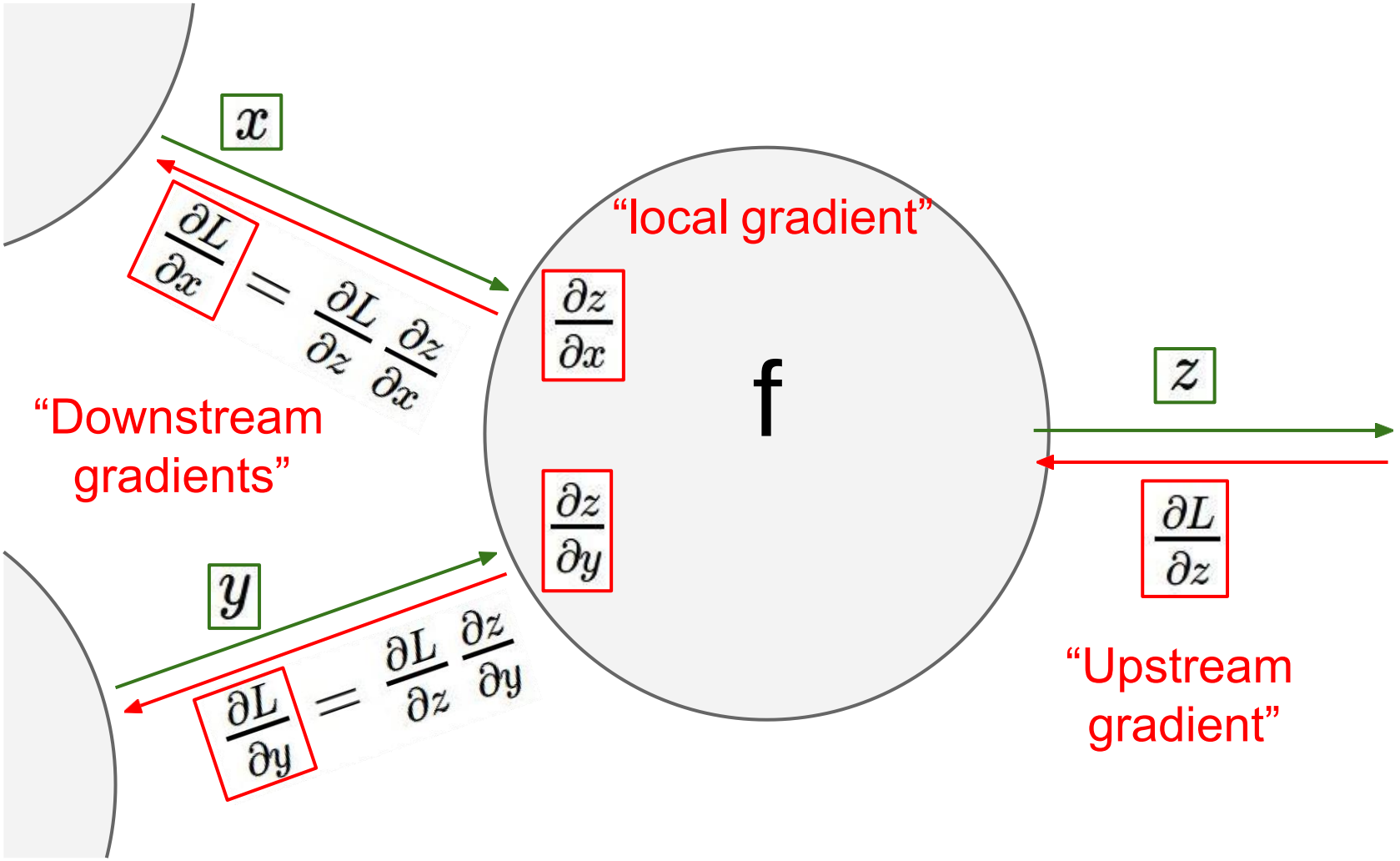




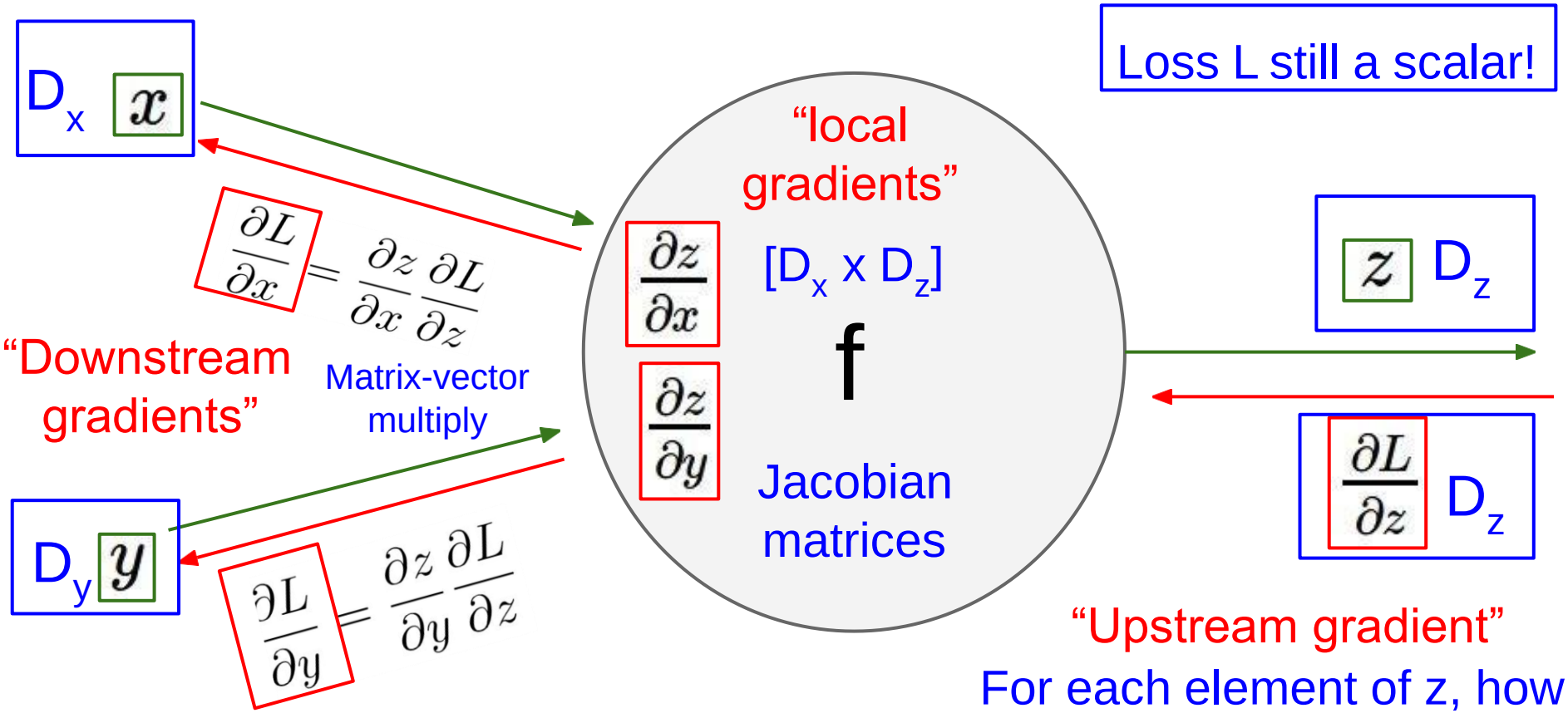
Last time: Computation Graph



Last time: Backpropagation



Backprop with Vectors



Backprop with Matrices (or Tensors)

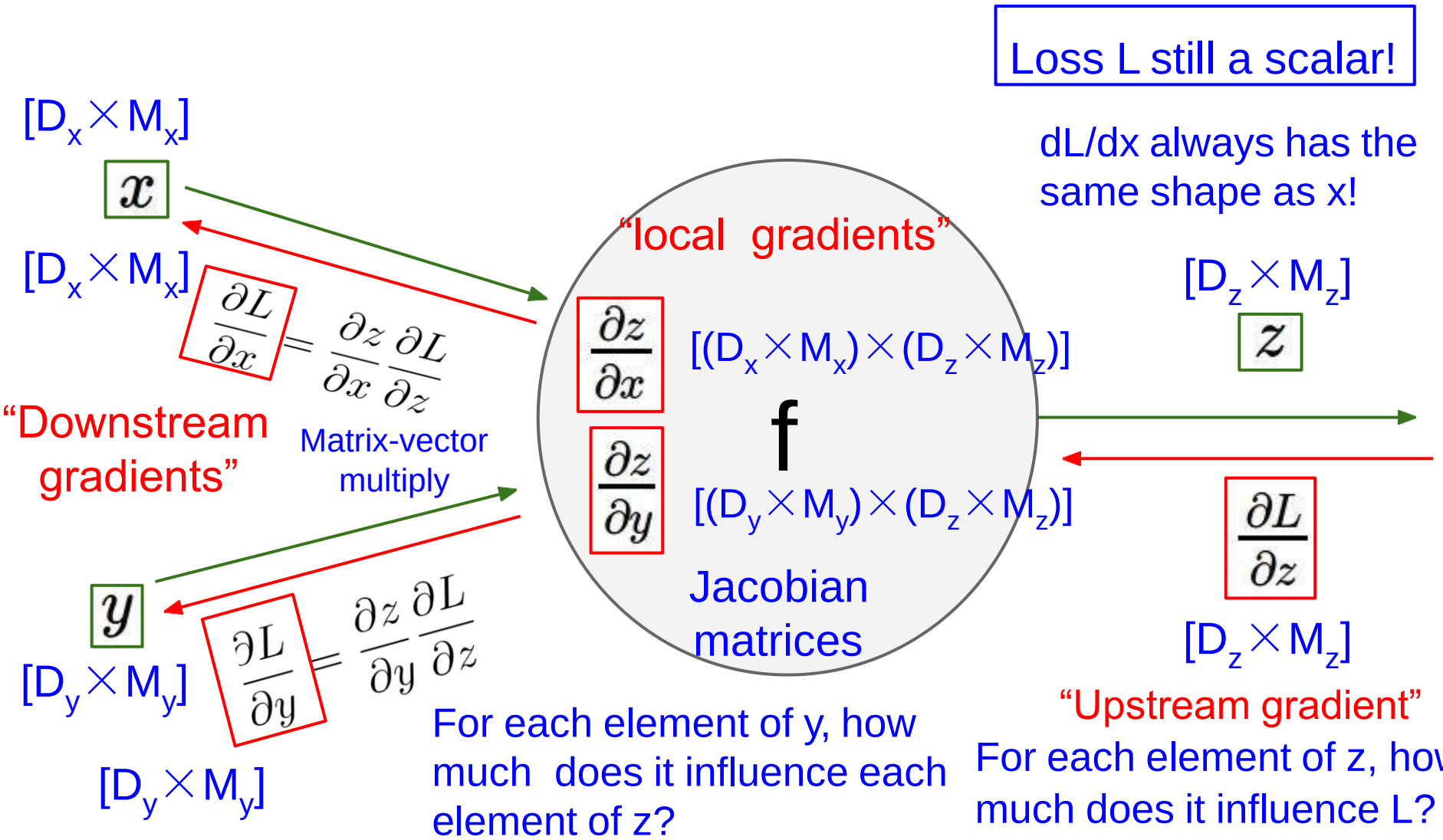


Image Classification: A core task in Computer Vision

(assume given a set of labels)
{dog, cat, truck, plane, ...}

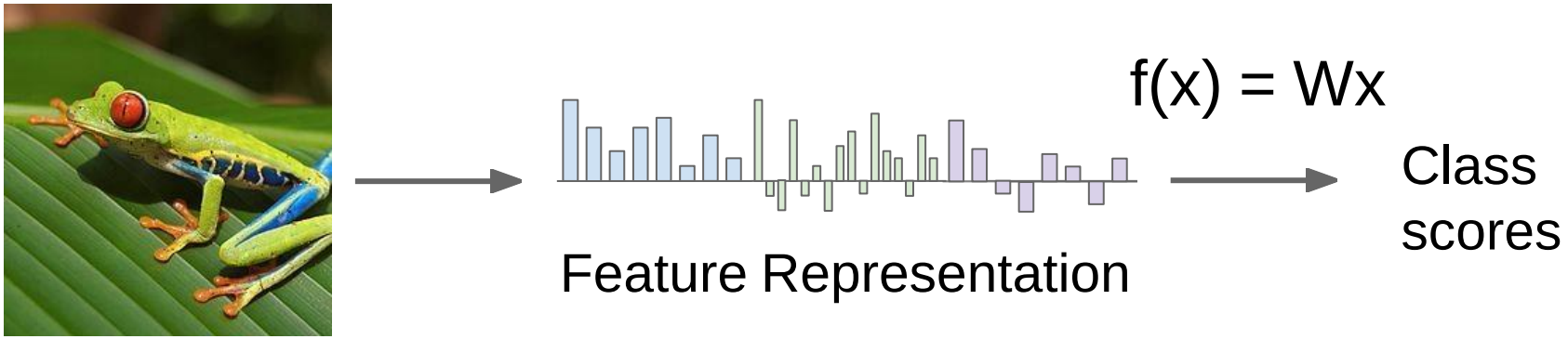
cat
dog
bird
deer
truck

Pixel space

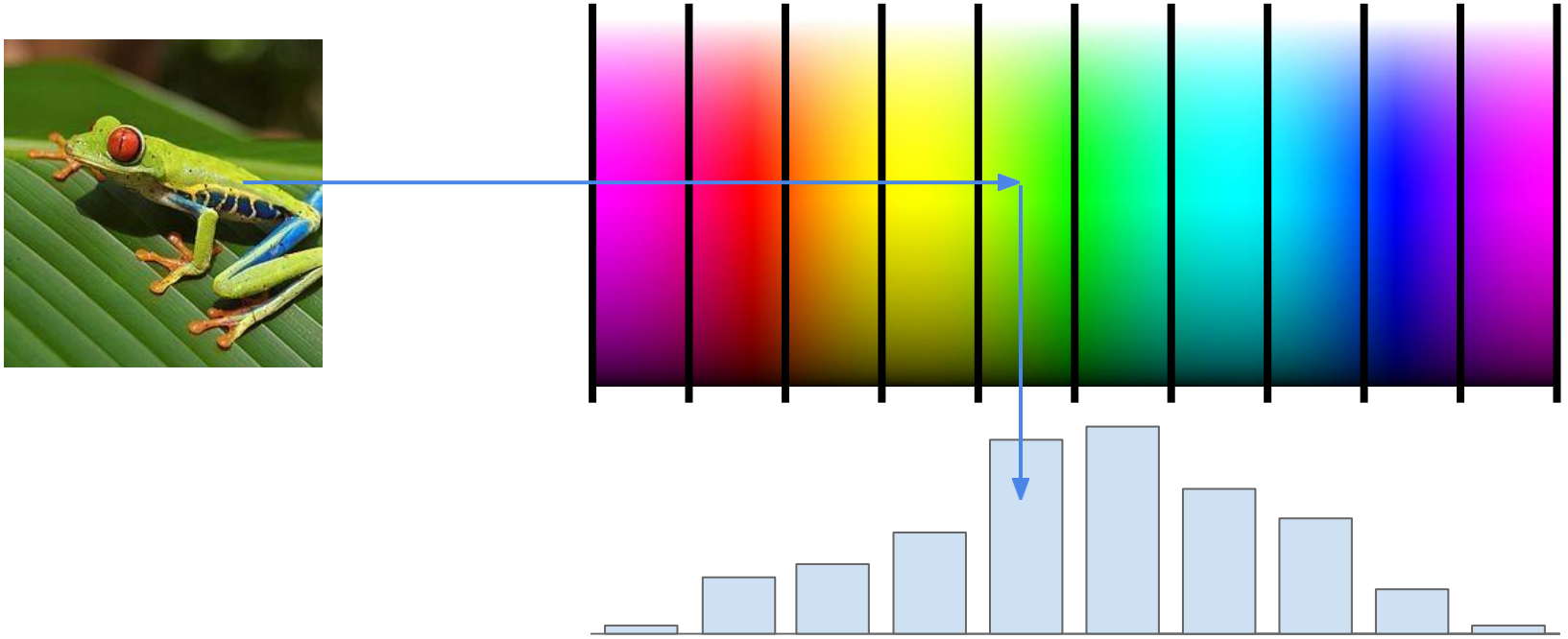
Class scores

$$f(x) = Wx$$

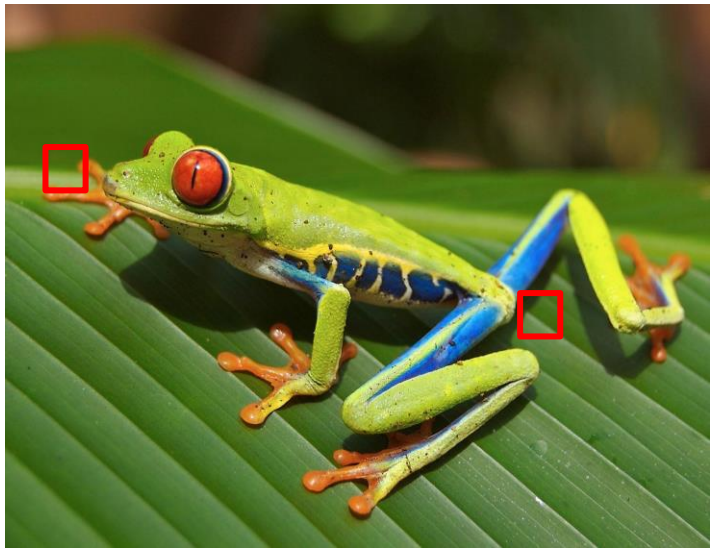
Image features

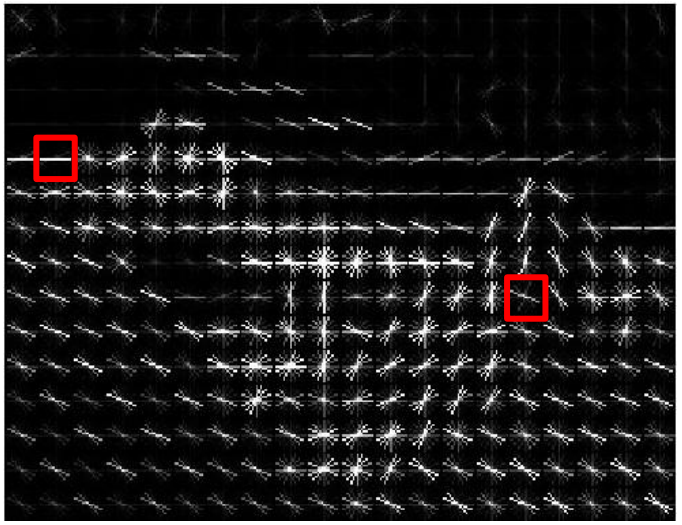


Example: Color Histogram



Example: Histogram of Oriented Gradients (HoG)





Divide image into 8x8 pixel regions Within each region quantize edge direction into 9 bins

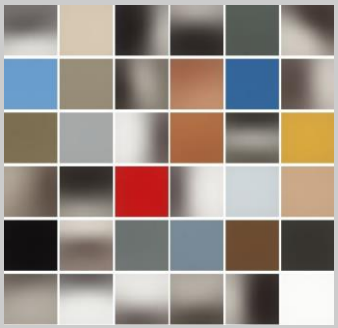
Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Example: Bag of Words

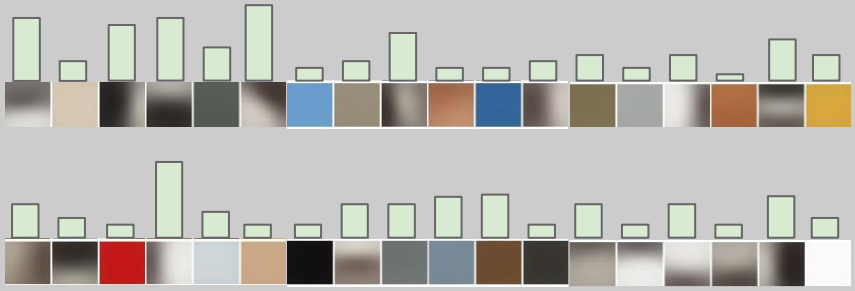
Step 1: Build codebook

Extract random patches

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



Step 2: Encode images



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

Image Features

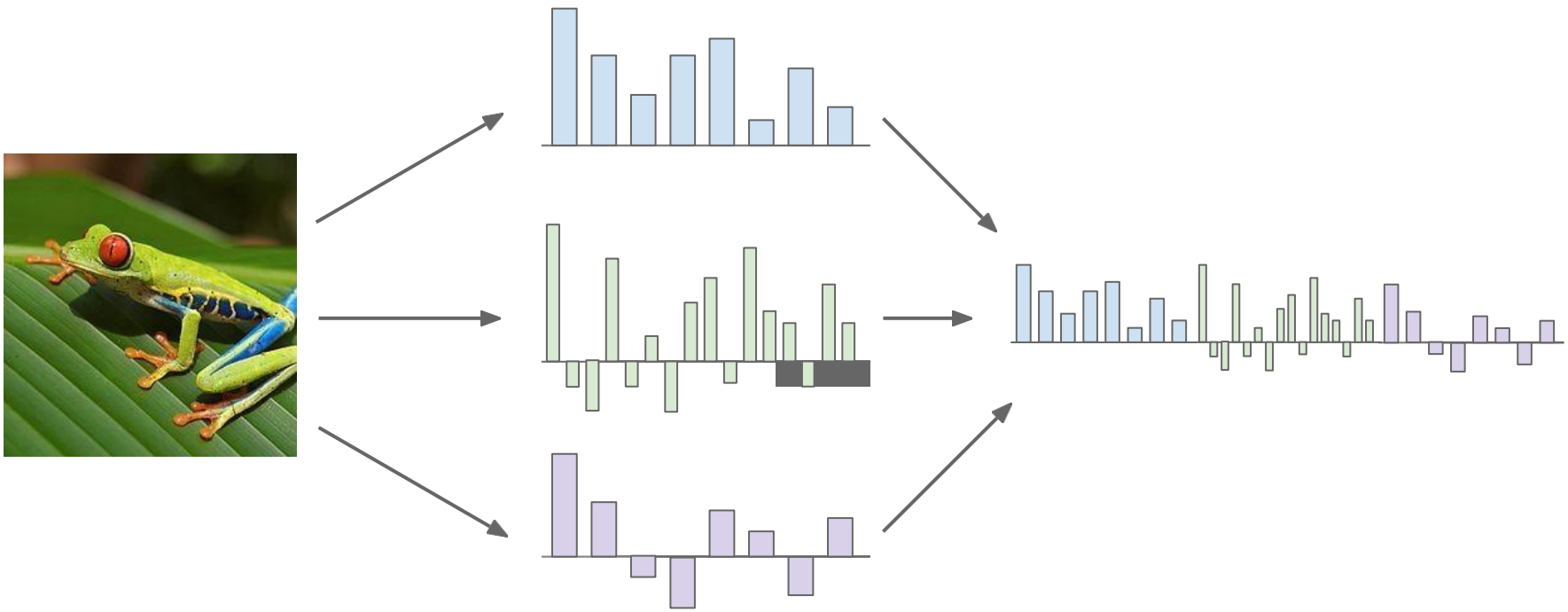
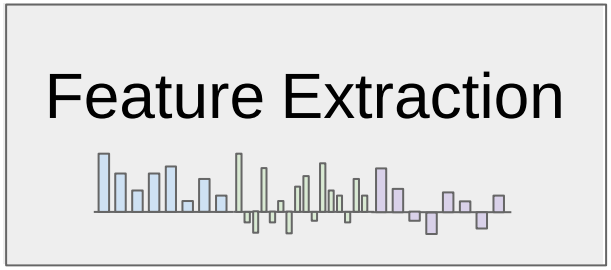


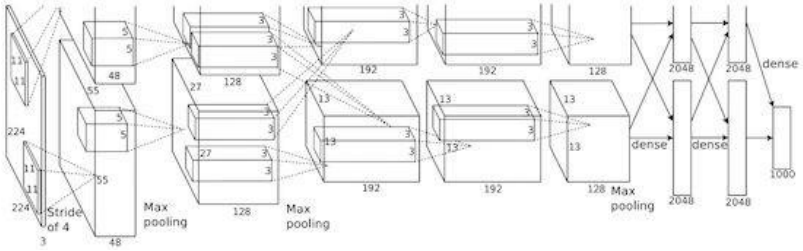
Image features vs. ConvNets



f

10 numbers giving scores for classes

training



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

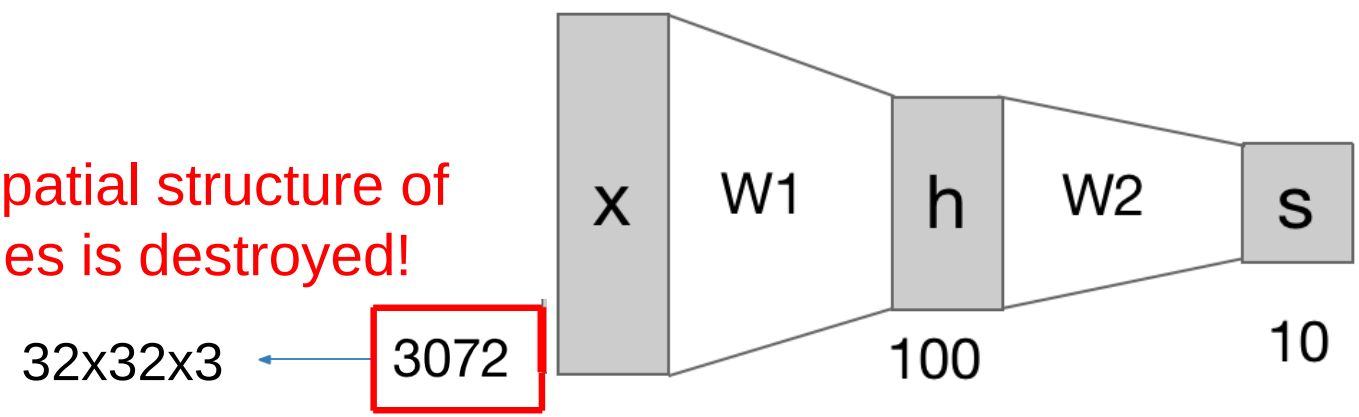
10 numbers giving scores for classes

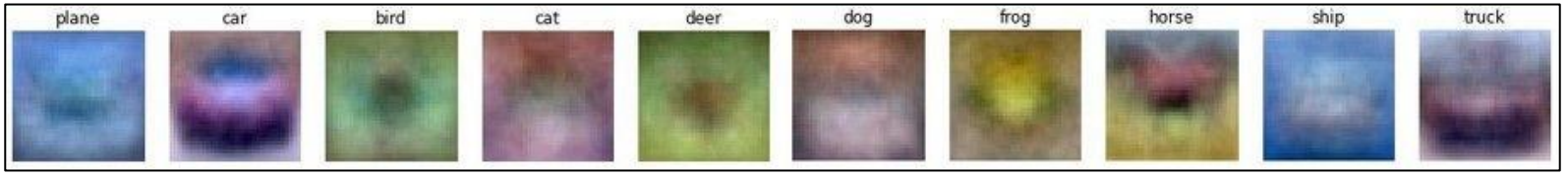
training

Last time: Neural Networks

- Linear score function: $f = Wx$
- 2-layer Neural Network: $f = W_2 \max(0, W_1 x)$

The spatial structure of images is destroyed!





Next: Convolutional Neural Networks

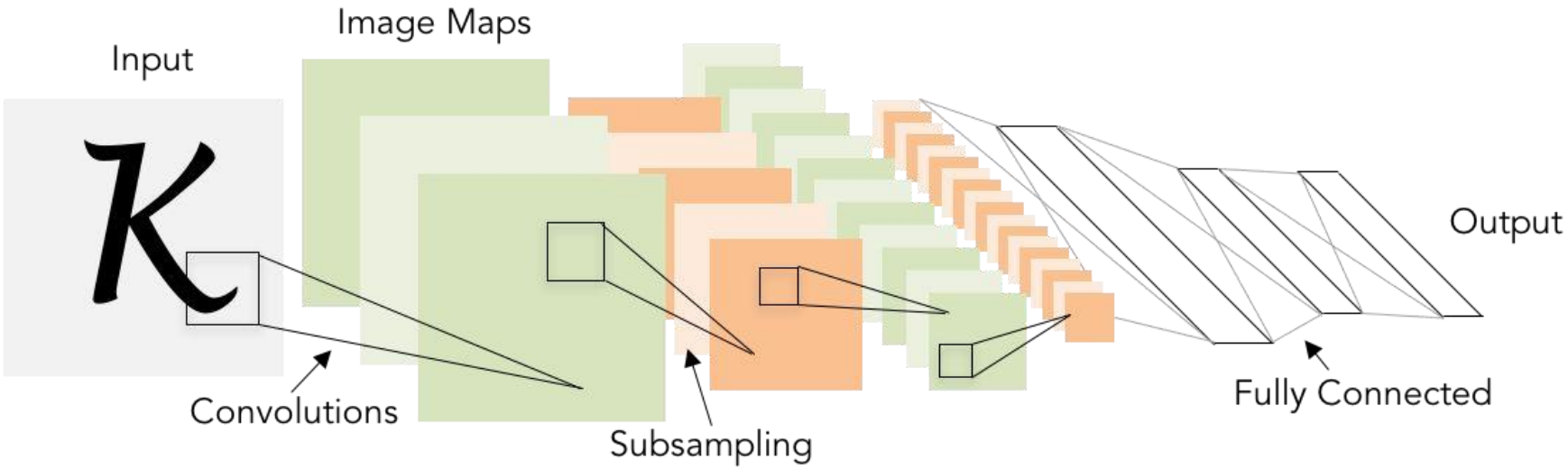


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history...

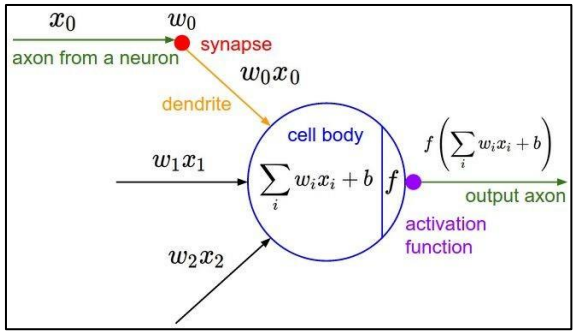
The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells（硫化镉光电管）to produce a 400-pixel image.

recognized letters of the alphabet

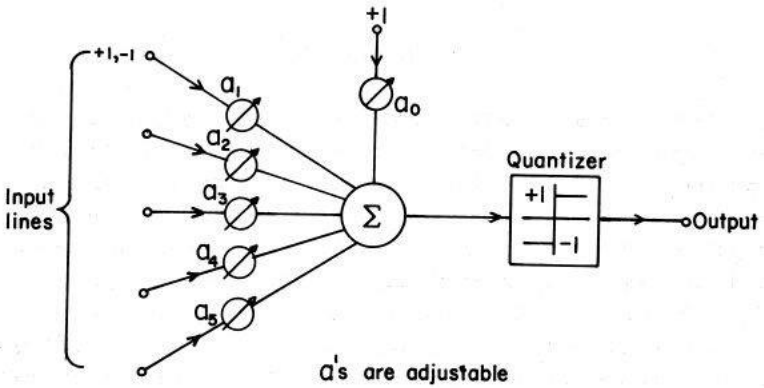
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

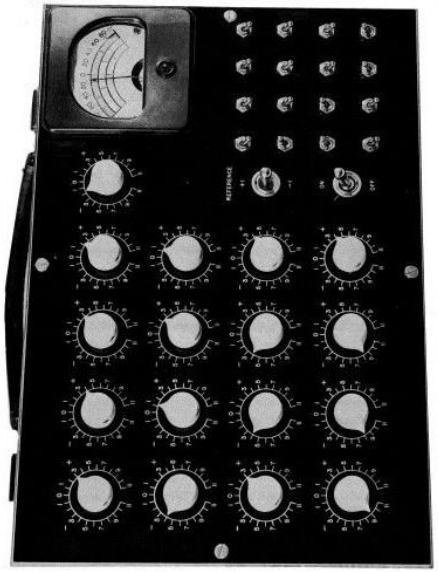
update rule:

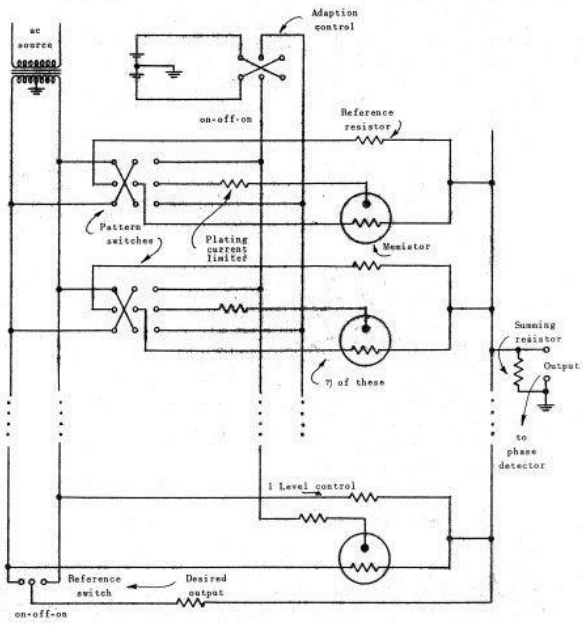


Frank Rosenblatt, ~1957: Perceptron

A bit of history...





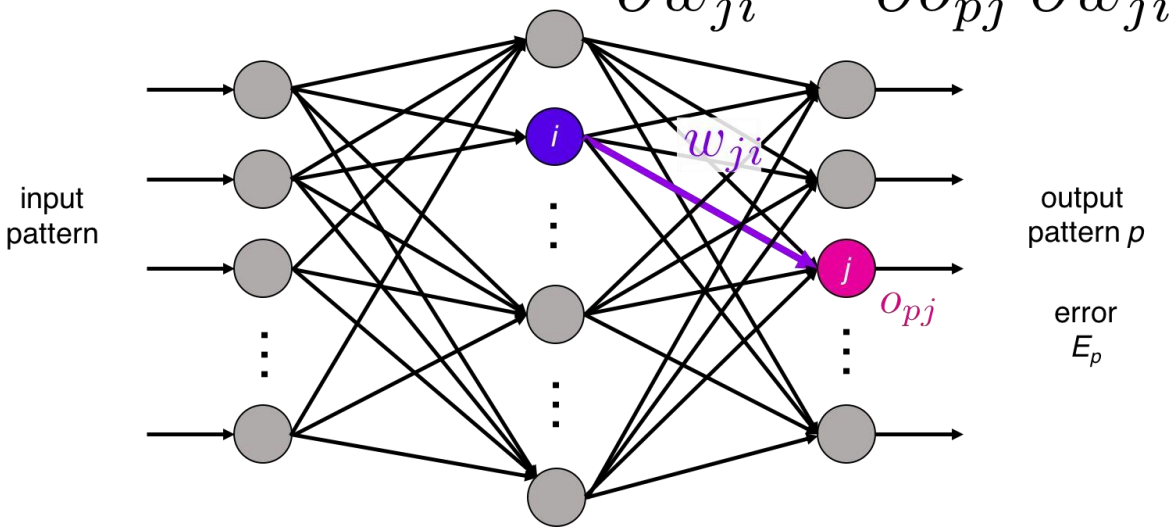


These figures are reproduced from Widrow 1960, Stanford Electronics Laboratories Technical Report with permission from Stanford University Special Collections.

Widrow and Hoff, ~1960: Adaline/Madaline

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

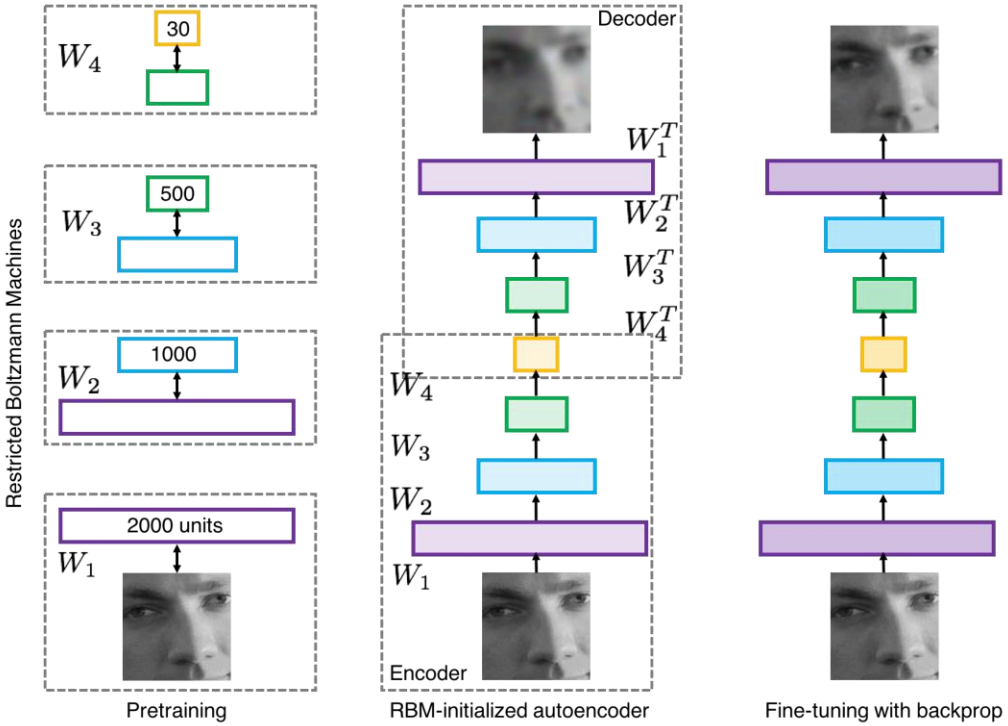
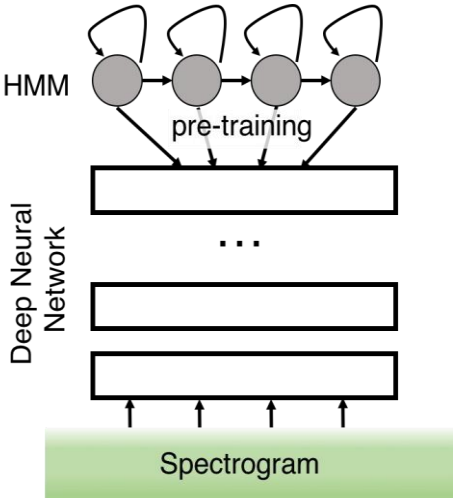


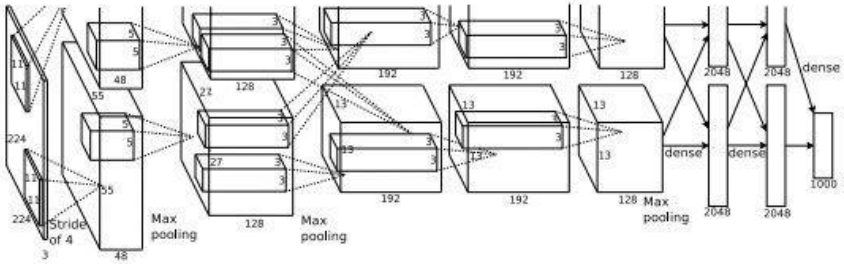
Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

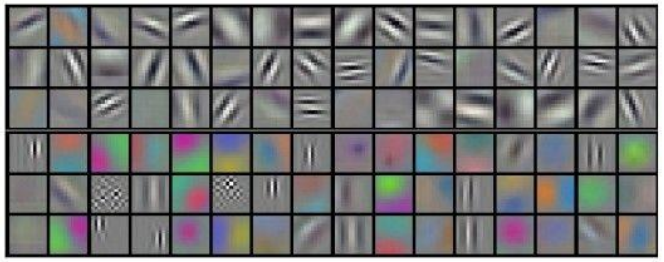
First strong results

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



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:

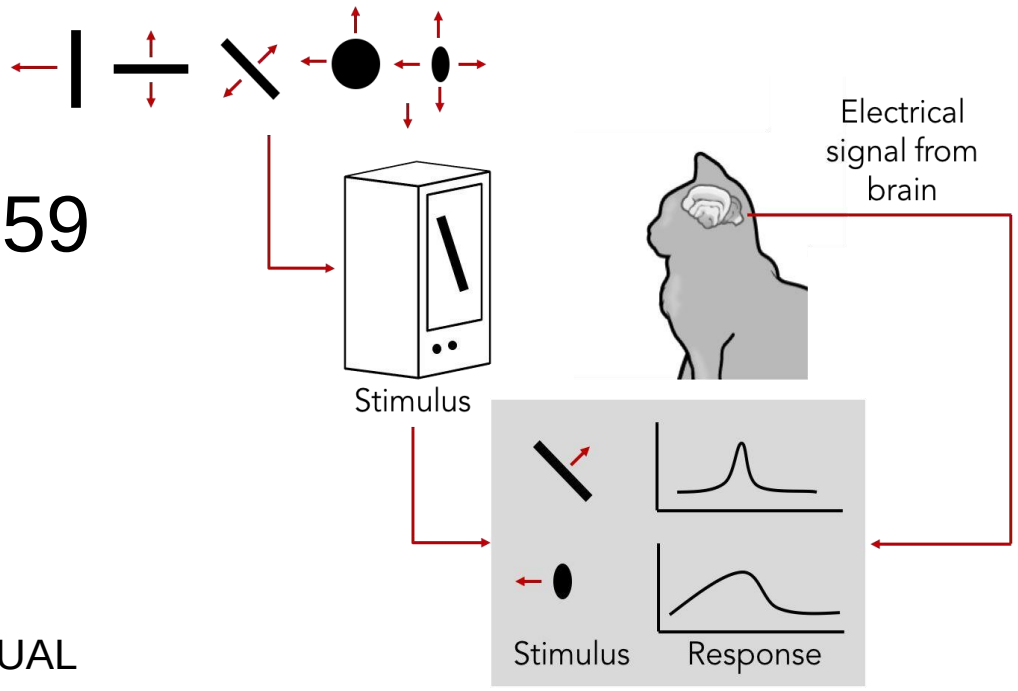
Hubel & Wiesel, 1959

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

1962

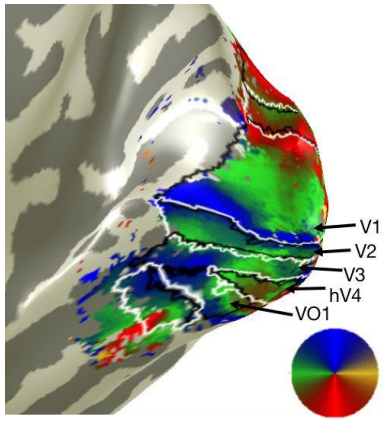
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...

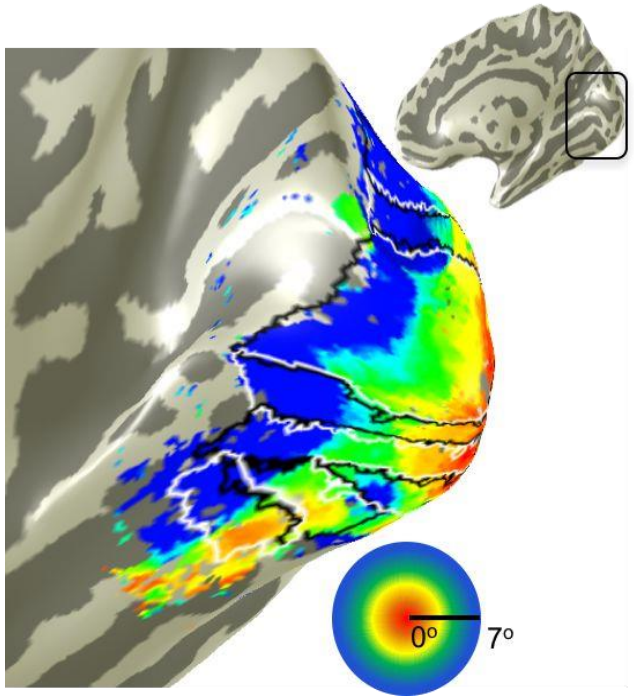


A bit of history:

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



Human brain



Visual cortex

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

Hierarchical organization

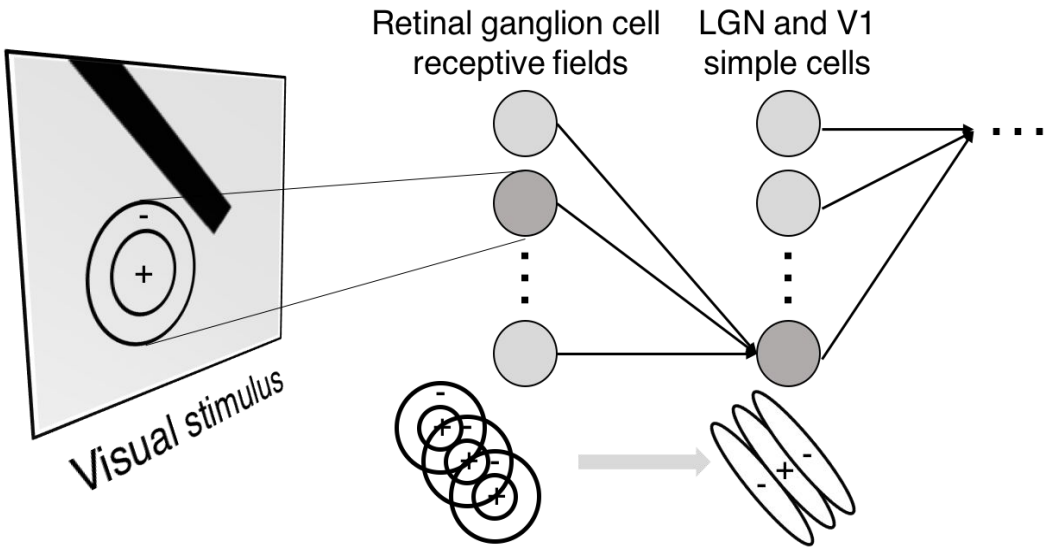
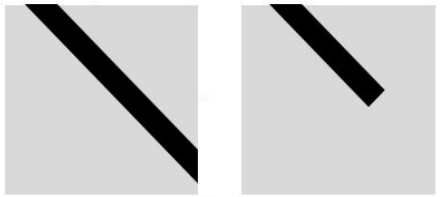


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

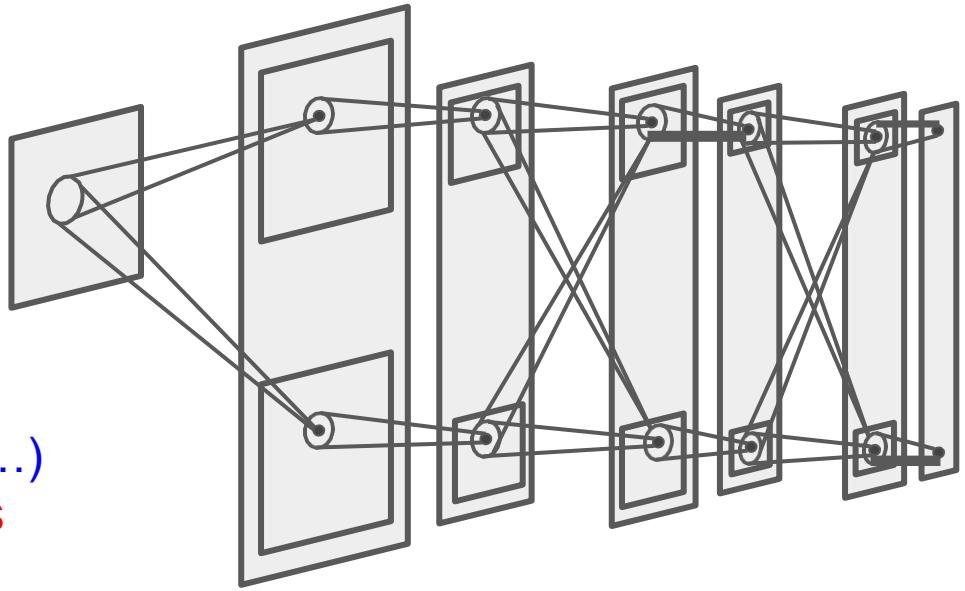


No response

Response (end point)

A bit of history:

Neocognitron
[Fukushima 1980]

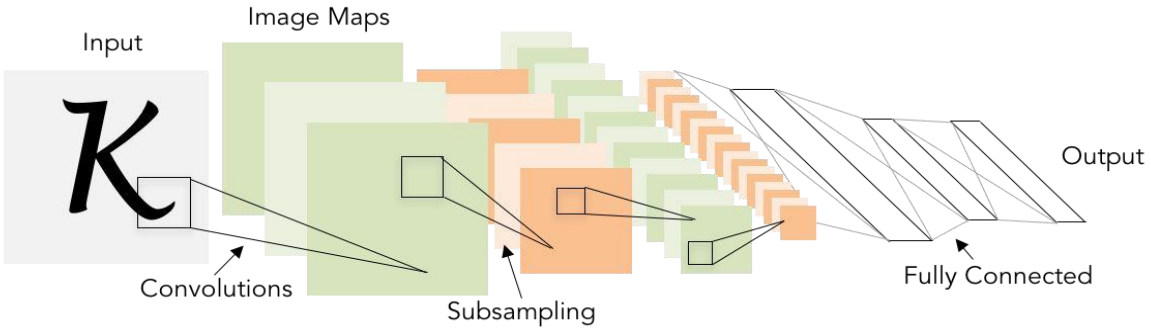


"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history:

Gradient-based learning applied to document recognition

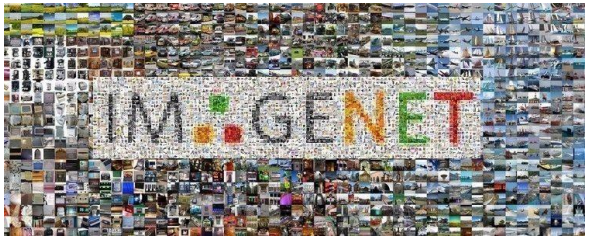
[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]



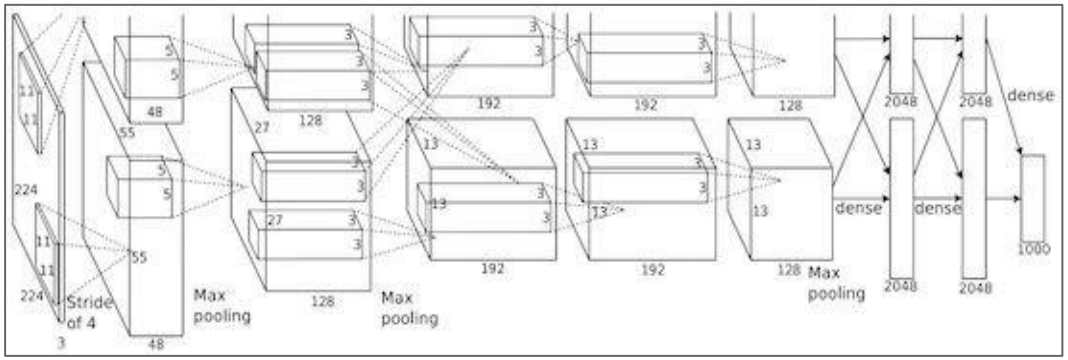


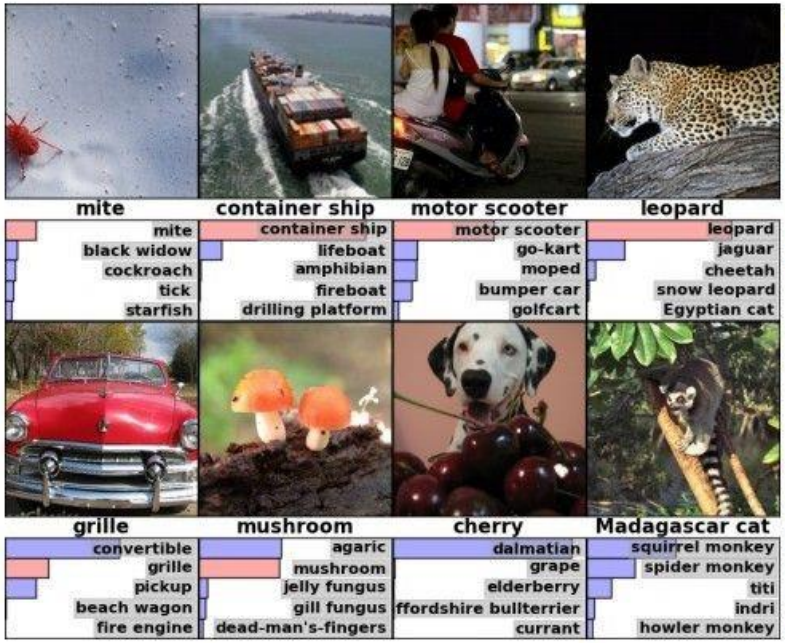
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

Fast-forward to today: ConvNets are everywhere

Classification

Retrieval



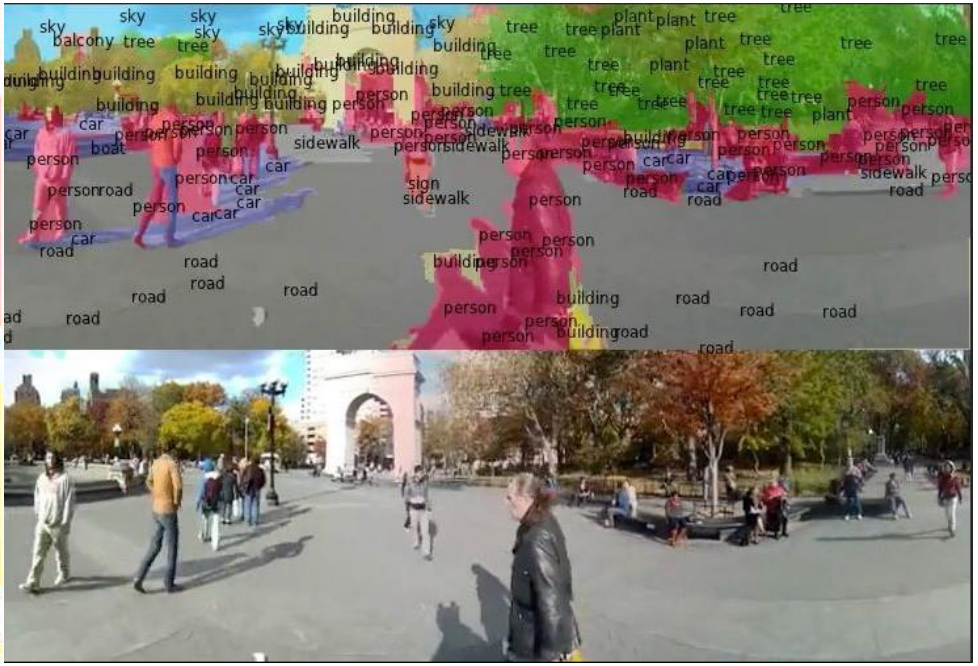
Fast-forward to today: ConvNets are everywhere

Detection

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

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

Segmentation



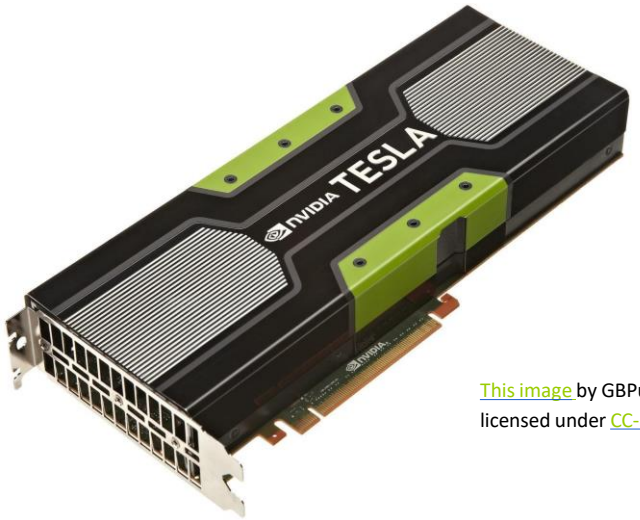
Figures copyright Clement Farabet, 2012.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

Photo by Lane McIntosh. Copyright CS231n 2017.

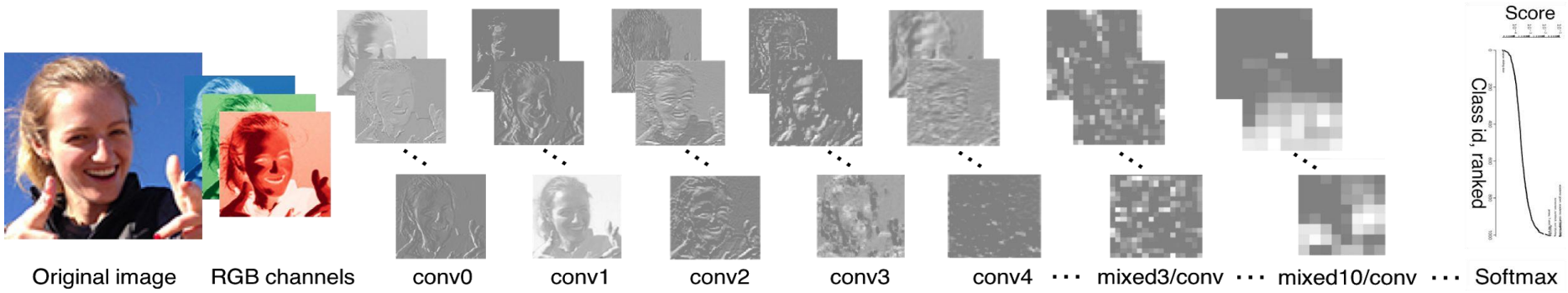
self-driving cars



NVIDIA Tesla line

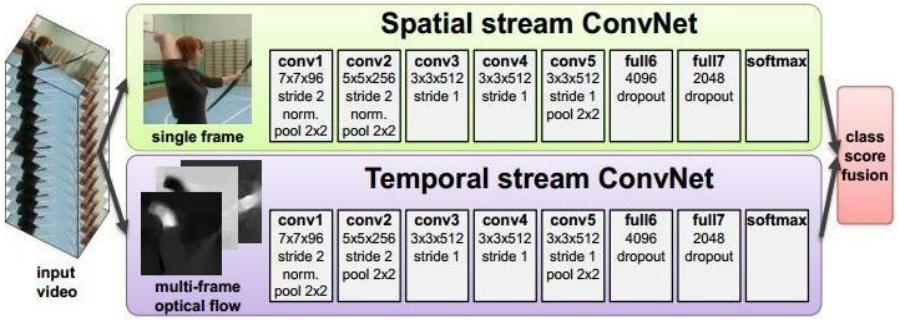
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

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.

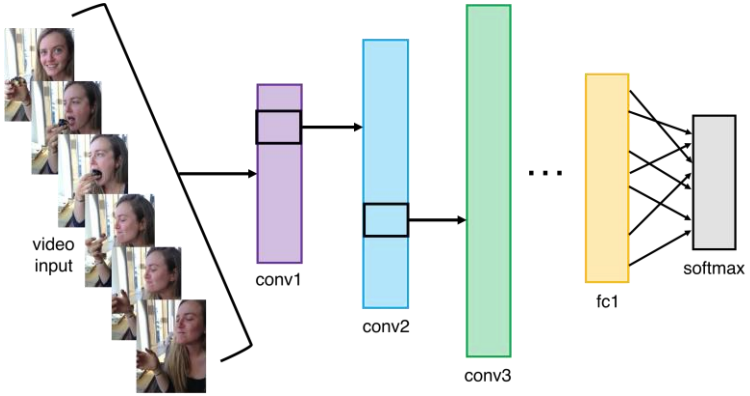
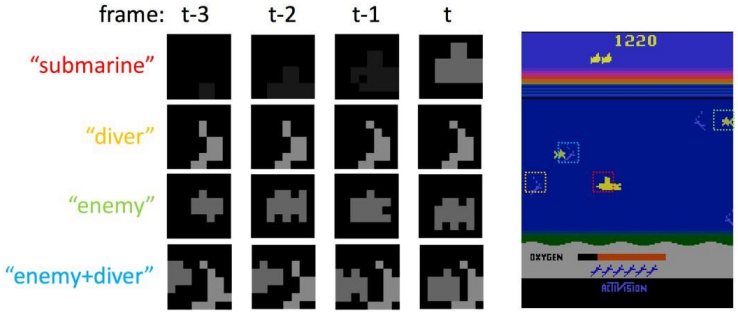


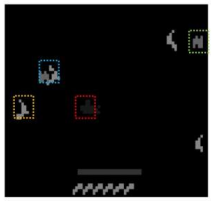
Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

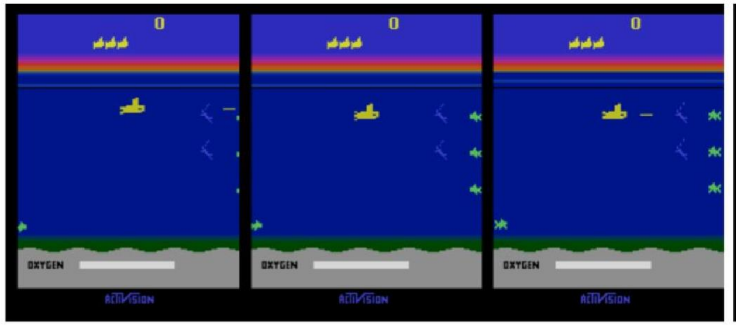
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

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



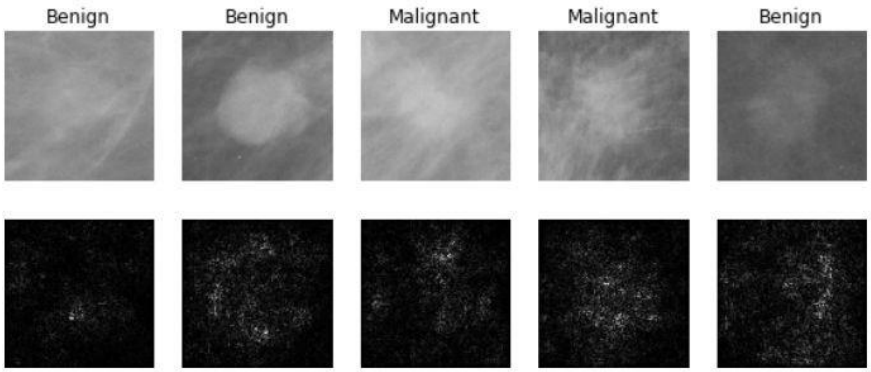




[Guo et al. 2014]

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]

[Dieleman et al. 2014]

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

Fast-forward to today: ConvNets are everywhere

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

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

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

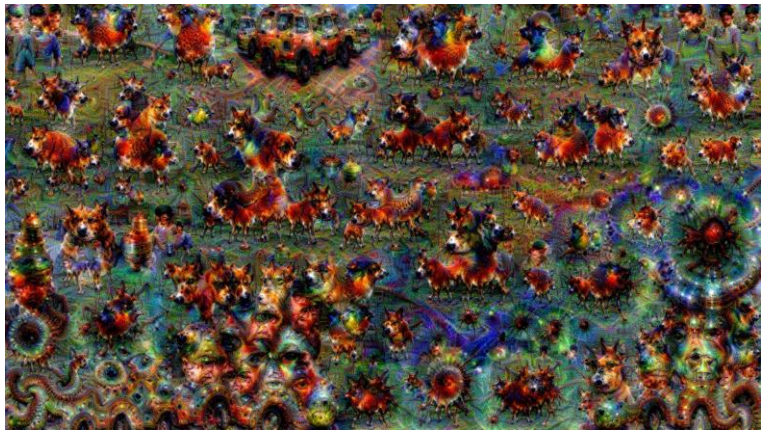
A woman standing on a beach holding a surfboard

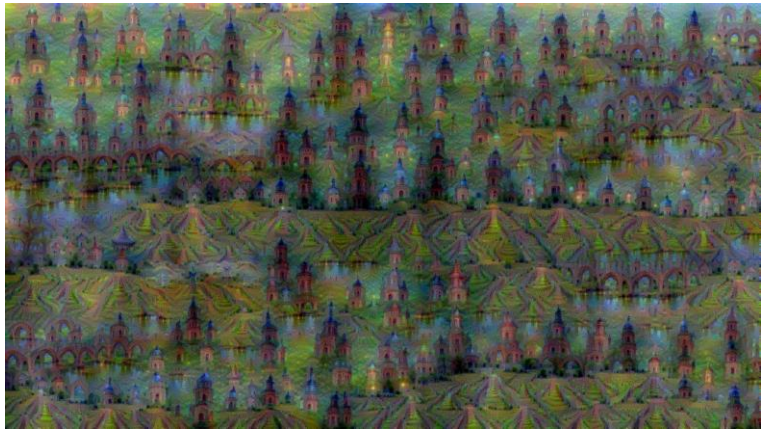
All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

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

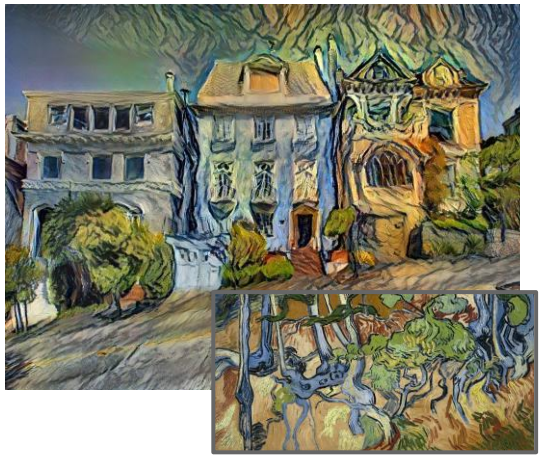
Fast-forward to today: ConvNets are everywhere





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

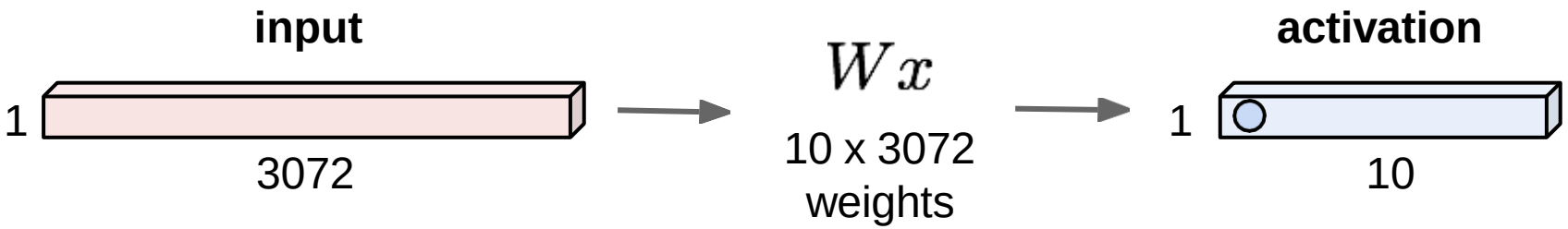


Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Convolutional Neural Networks

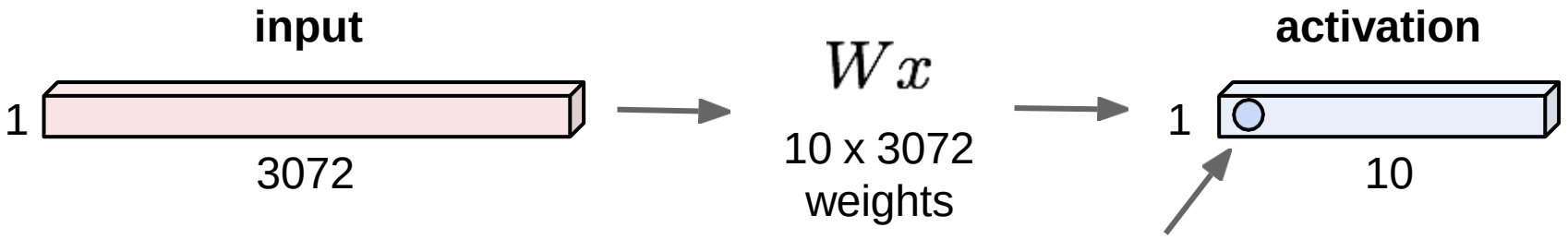
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

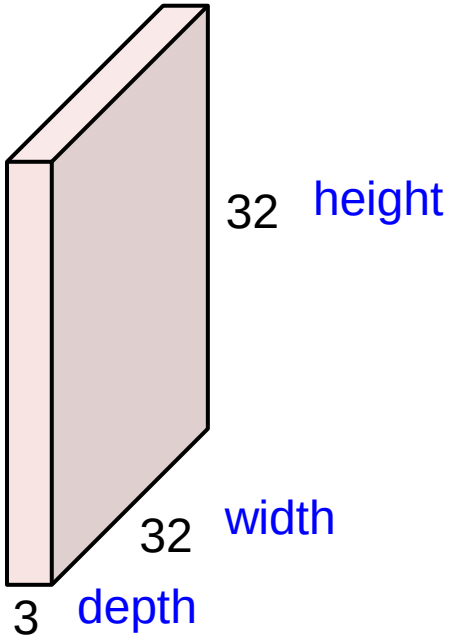
32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

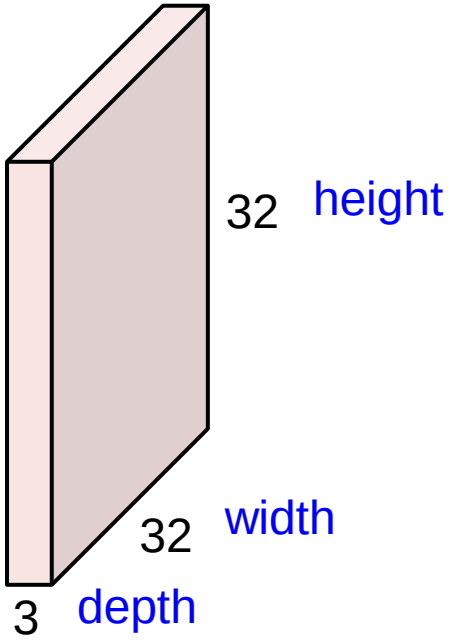
Convolution Layer

32x32x3 image -> preserve spatial structure

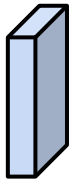


Convolution Layer

32x32x3 image



5x5x3 filter

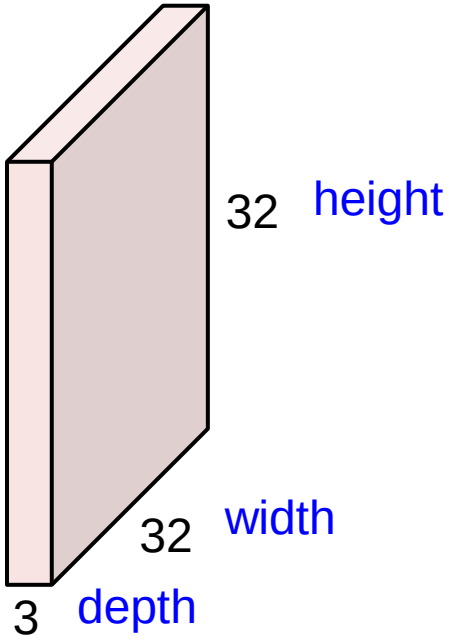


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

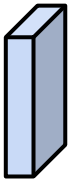
Convolution Layer

32x32x3 image

Filters always extend the full depth of the input volume

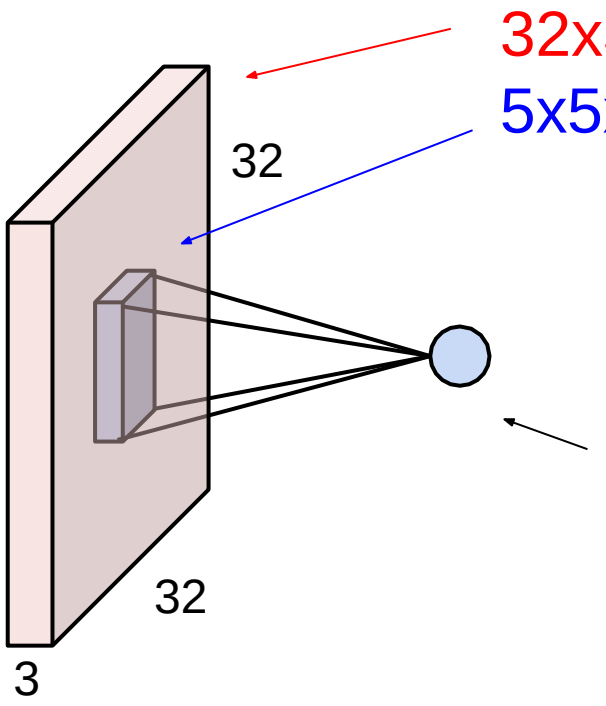


5x5x3 filter



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

Convolution Layer

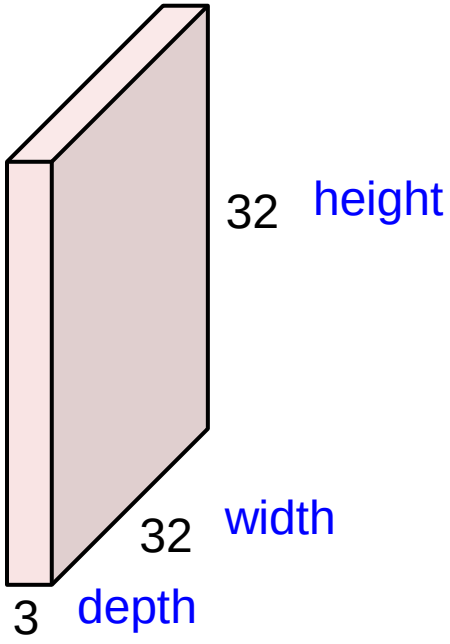


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

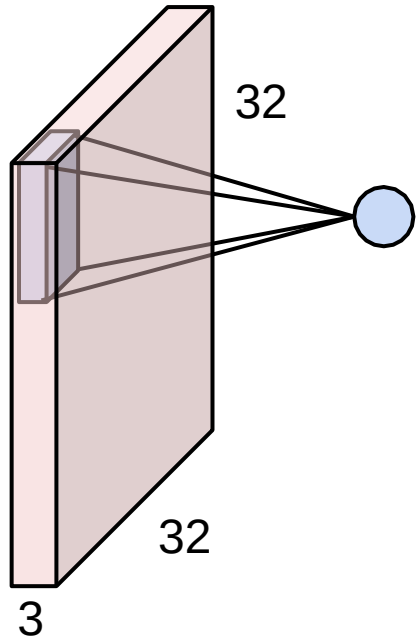
$$w^T x + b$$

Convolution Layer

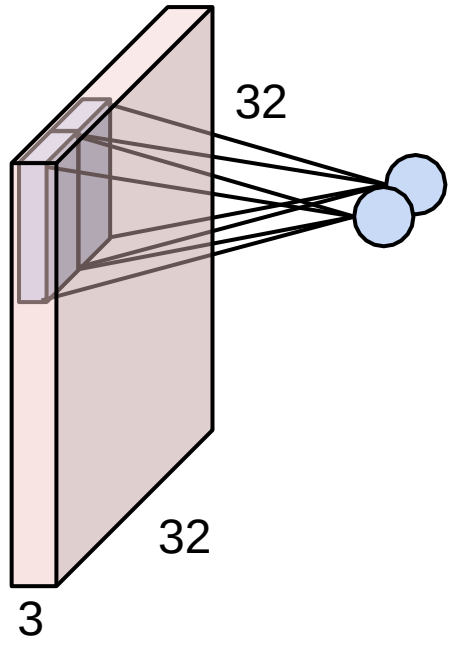
32x32x3 image -> preserve spatial structure



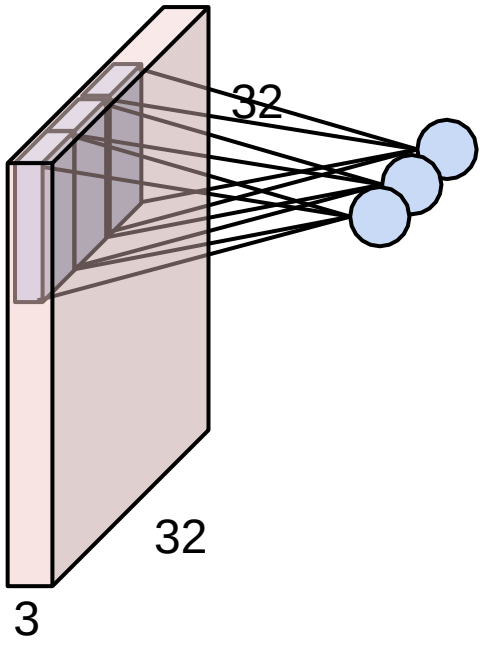
Convolution Layer



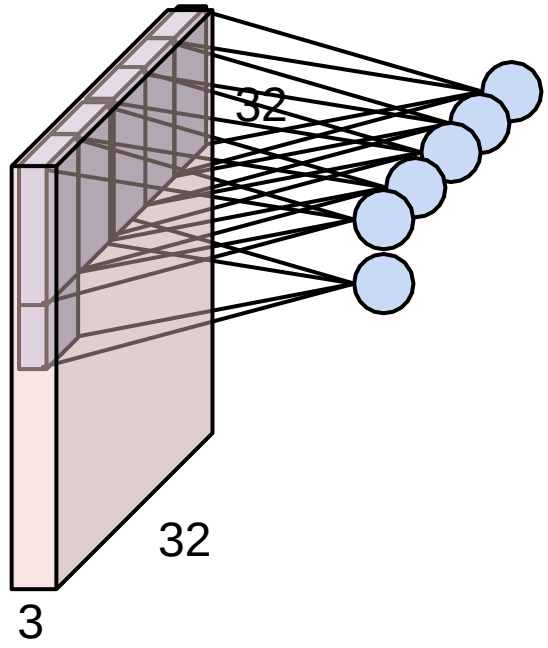
Convolution Layer



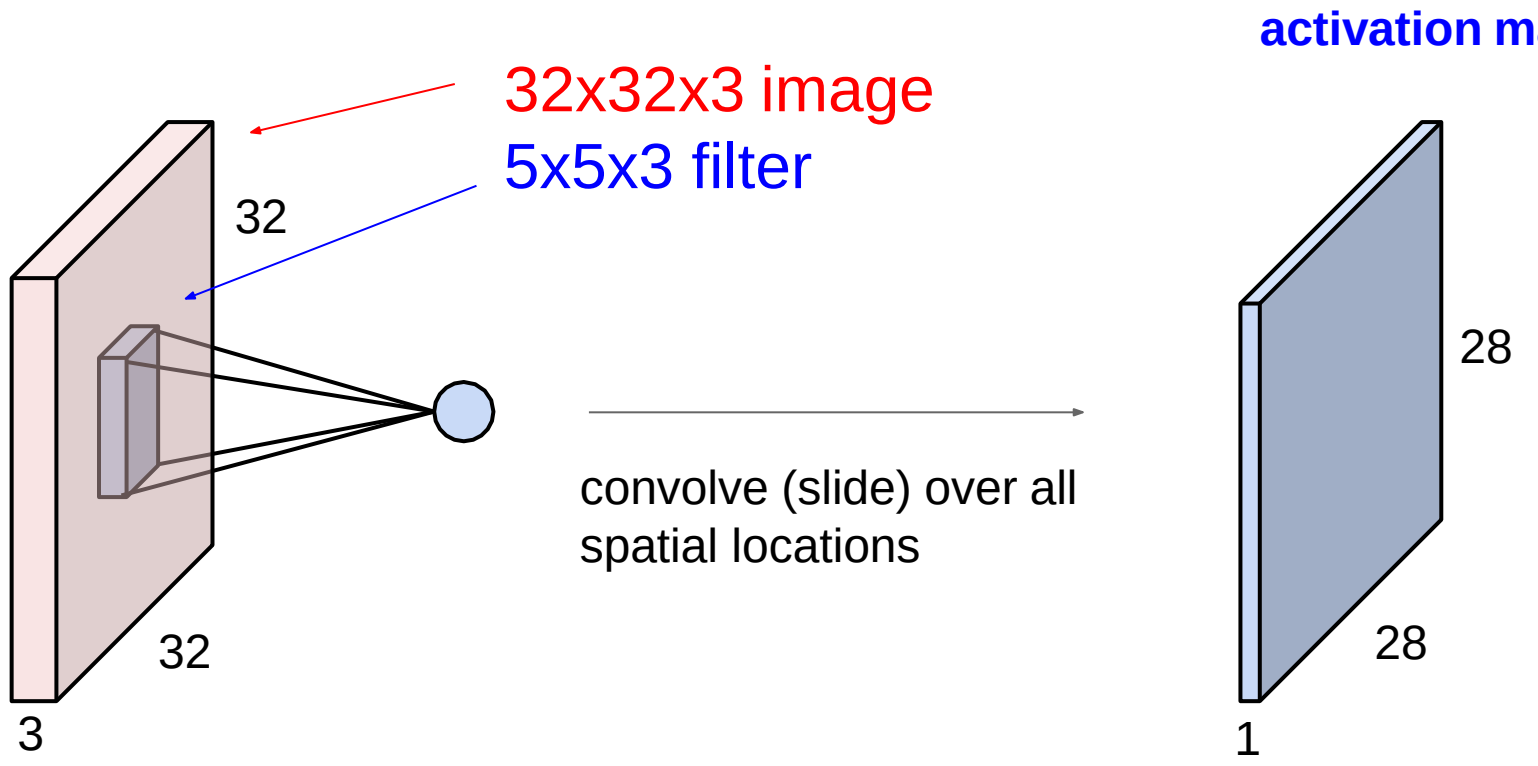
Convolution Layer



Convolution Layer



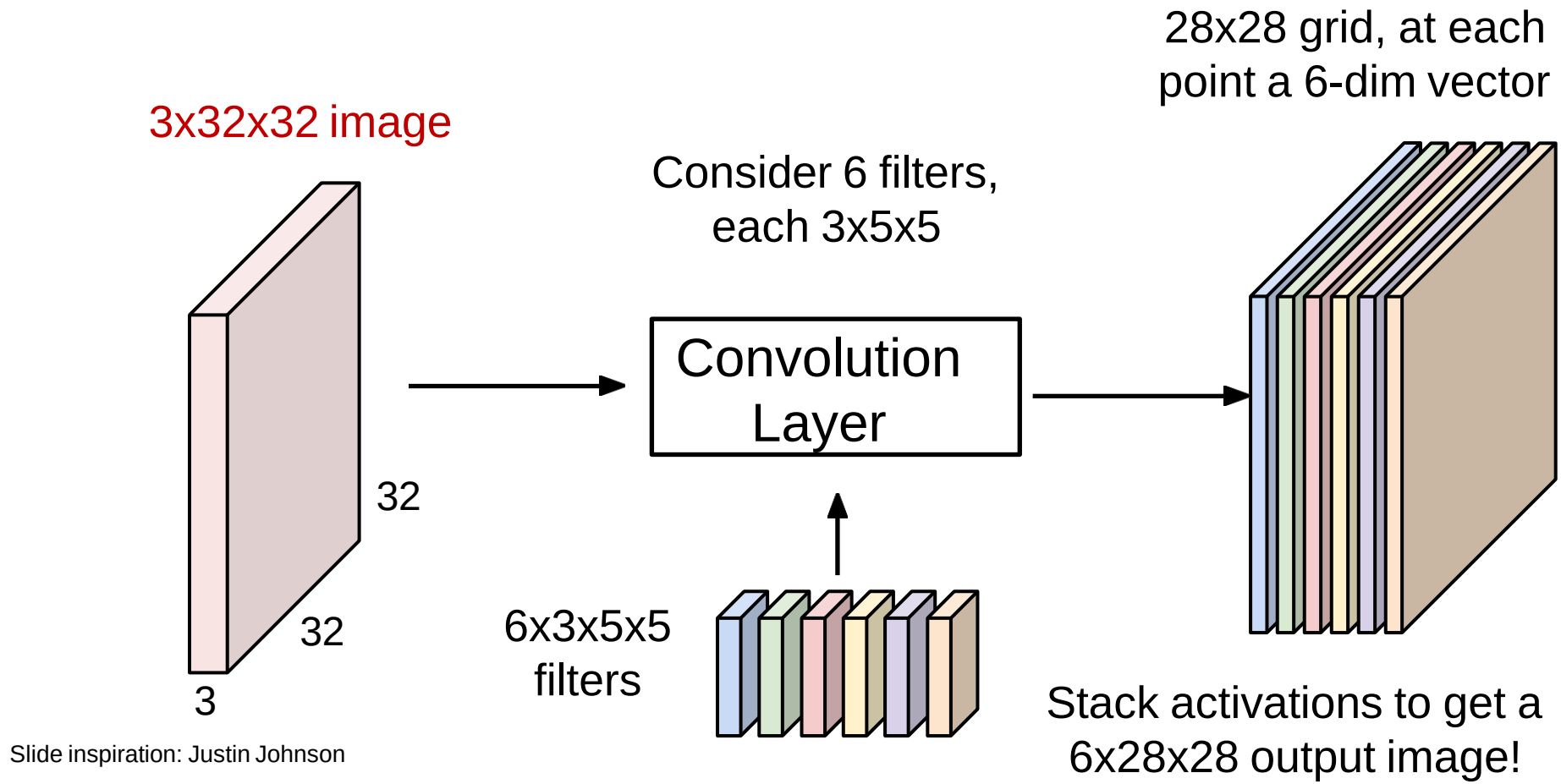
Convolution Layer



Convolution Layer

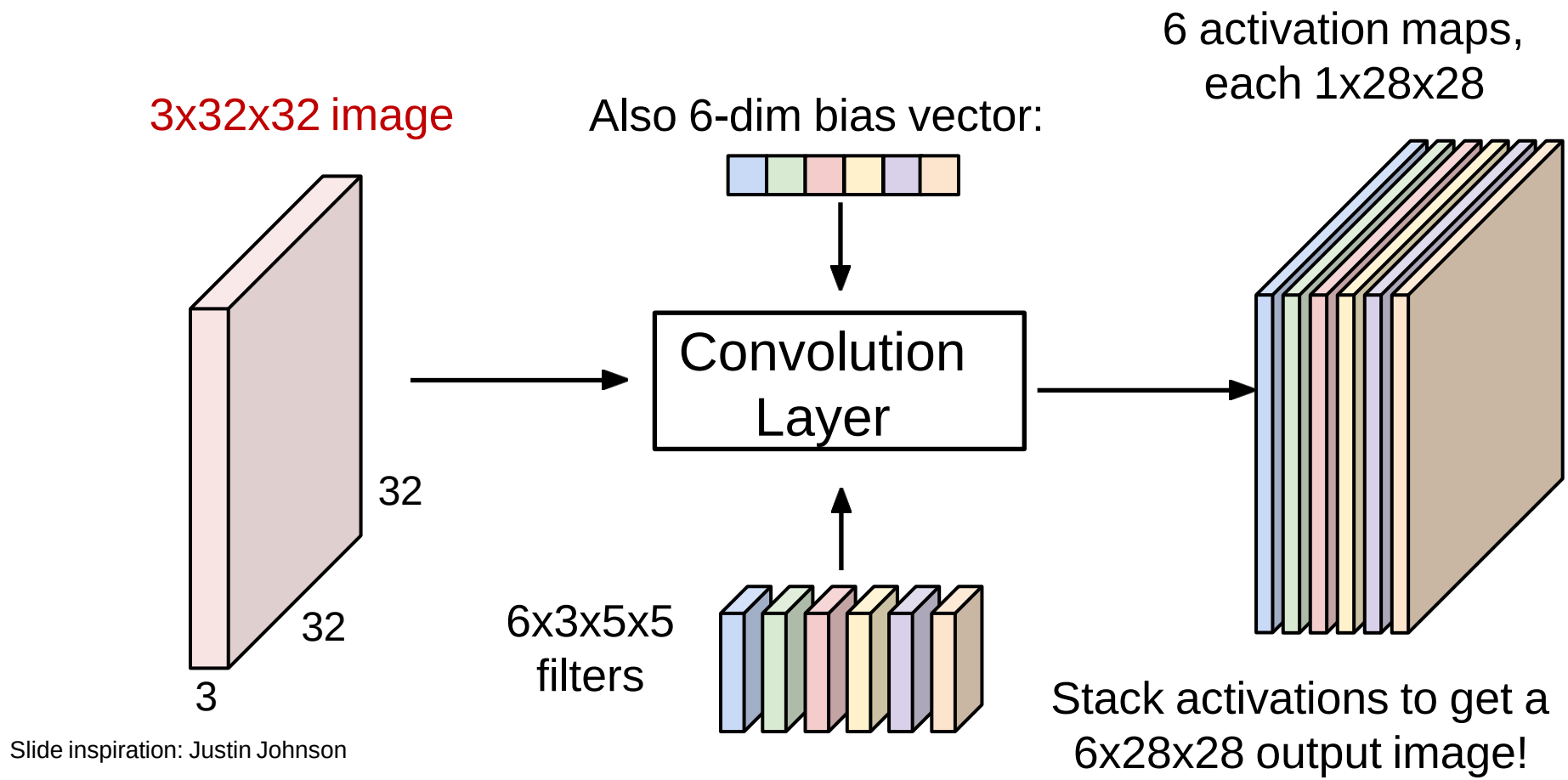
consider a second, green filter

Convolution Layer



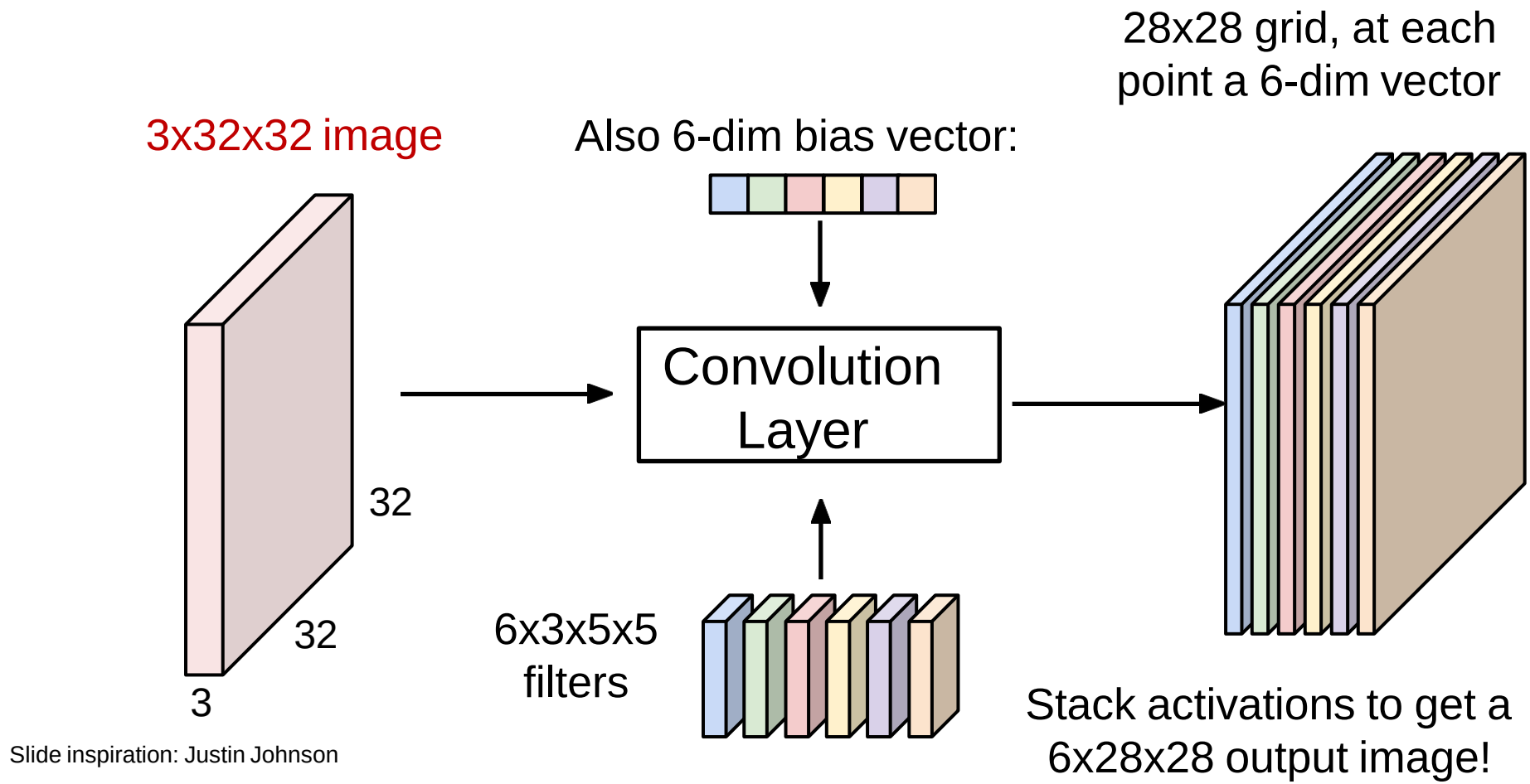
Slide inspiration: Justin Johnson

Convolution Layer



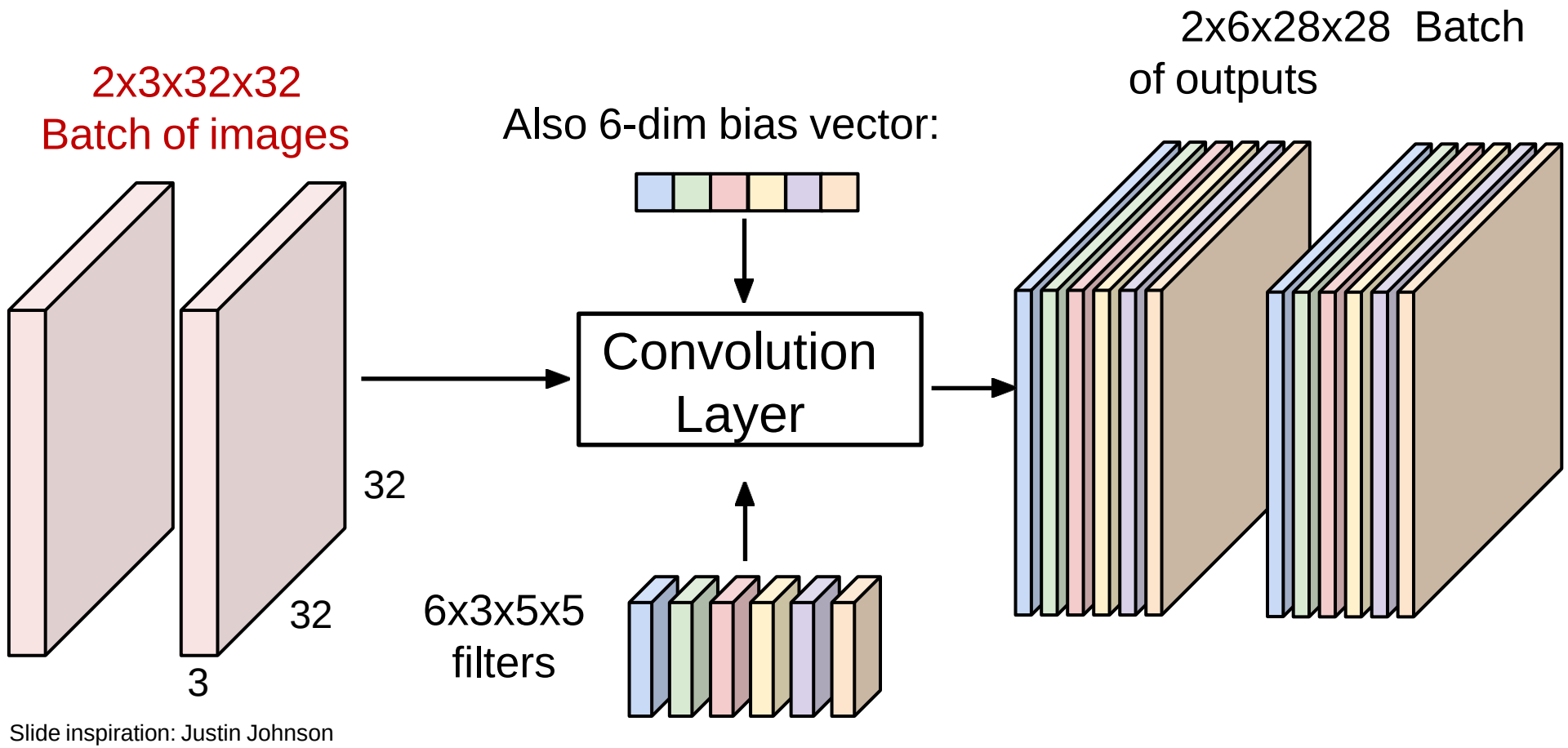
Slide inspiration: Justin Johnson

Convolution Layer



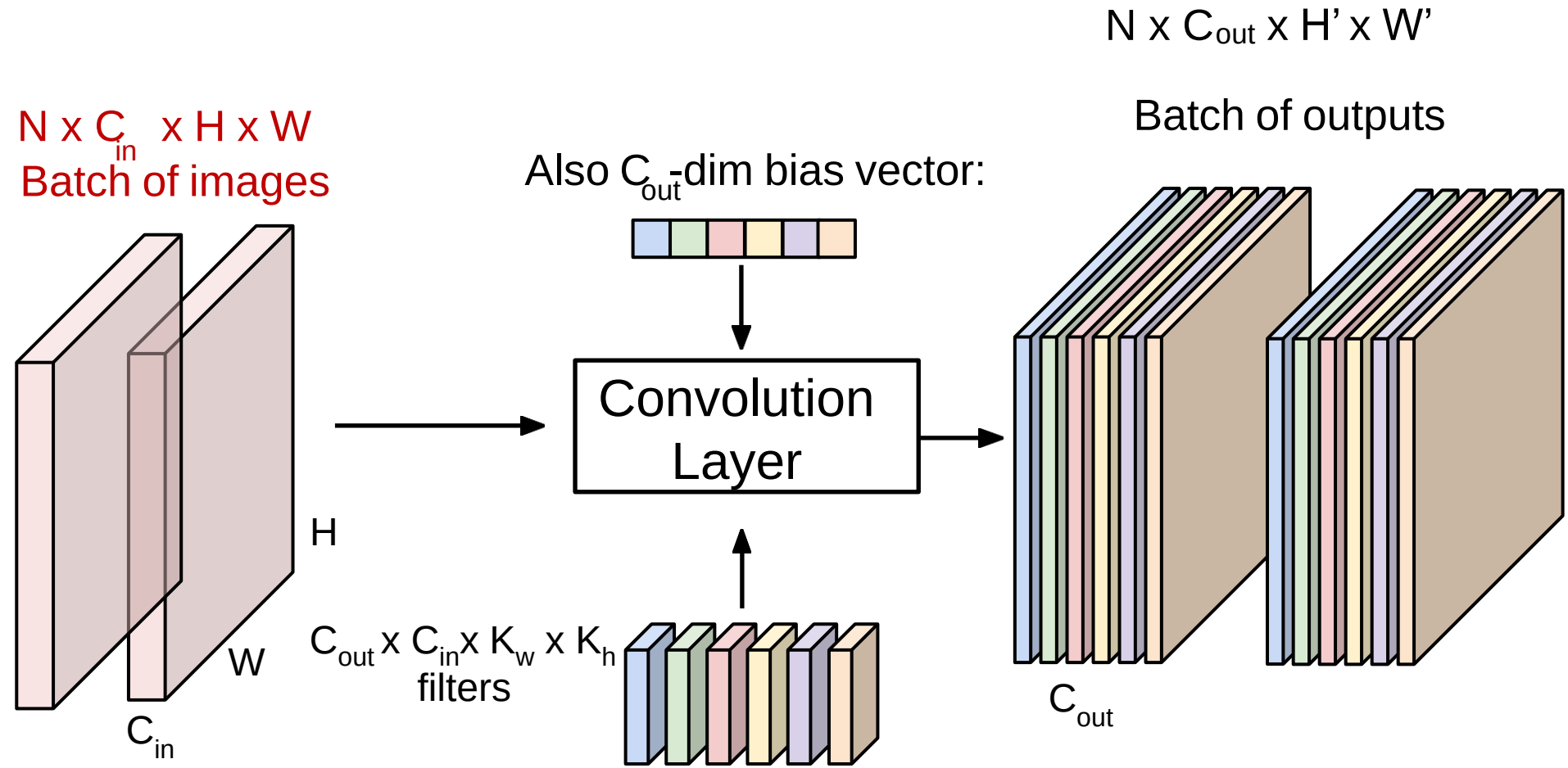
Slide inspiration: Justin Johnson

Convolution Layer



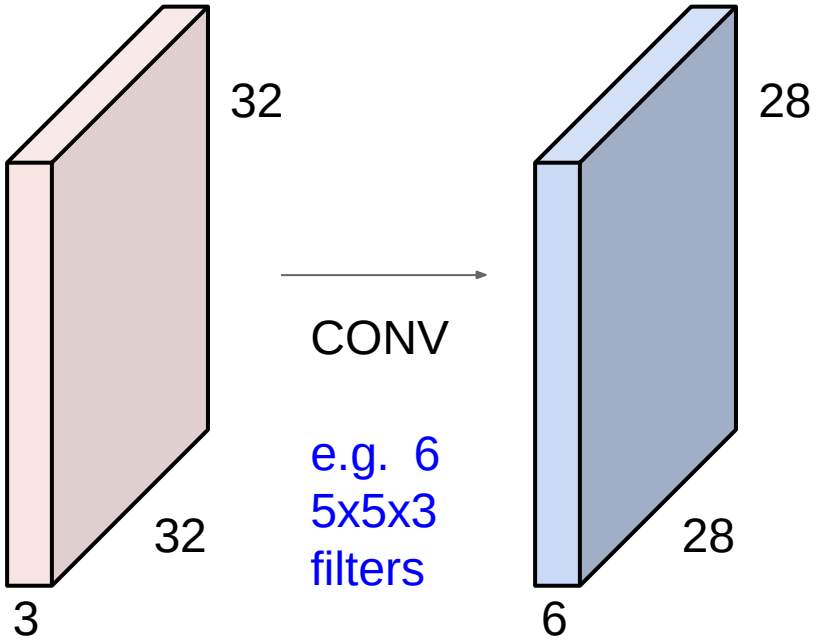
Slide inspiration: Justin Johnson

Convolution Layer

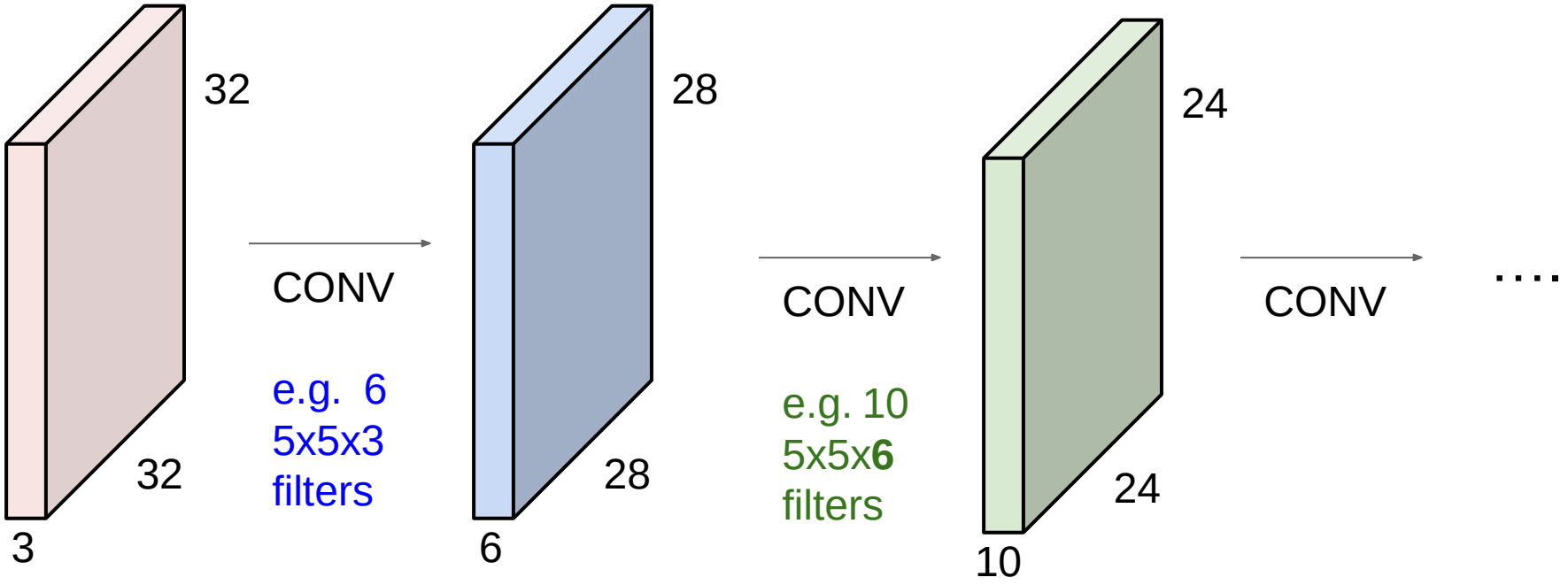


Slide inspiration: Justin Johnson

Preview: ConvNet is a sequence of Convolution Layers

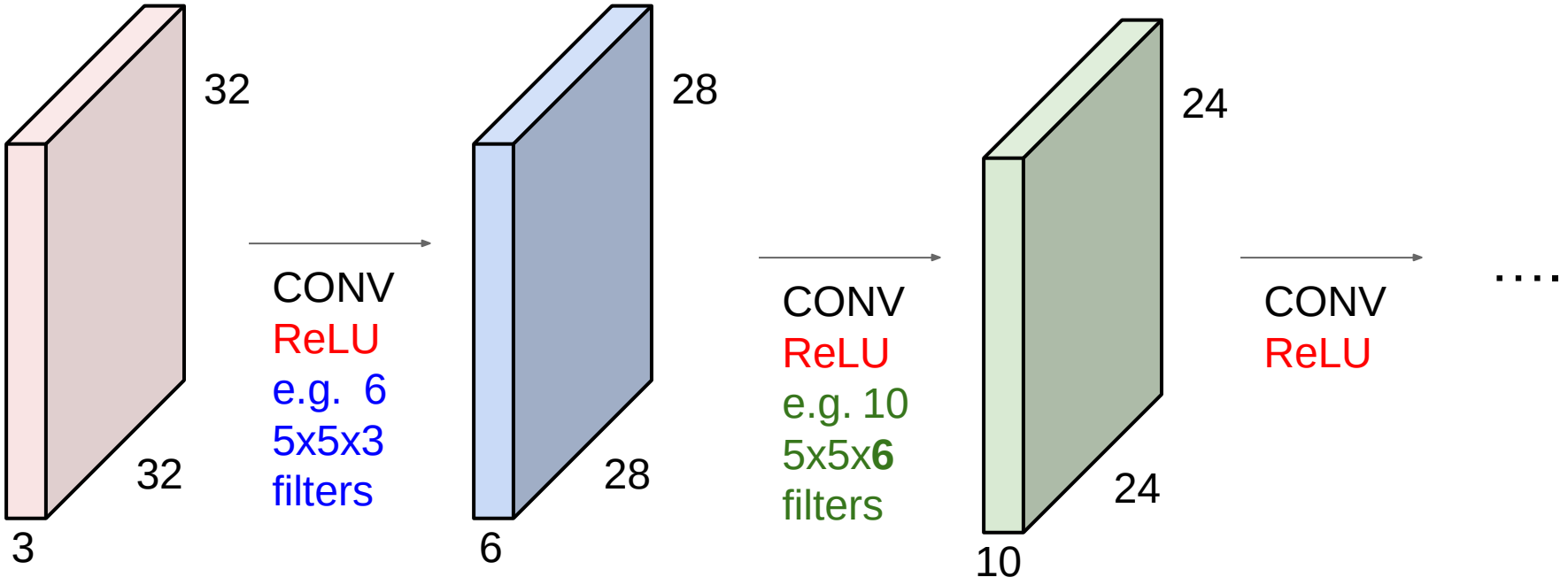


Preview: ConvNet is a sequence of Convolution Layers

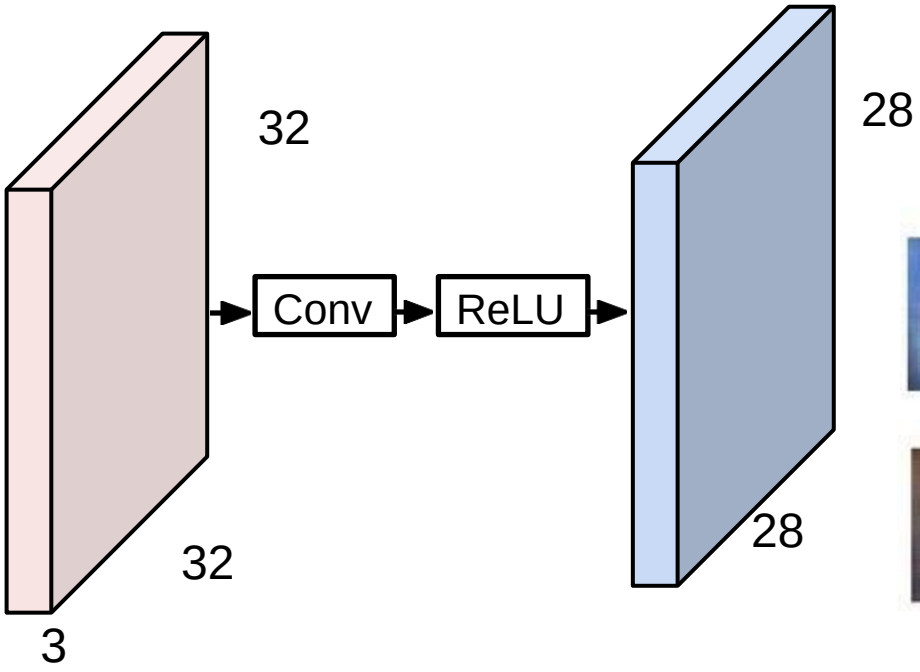


Preview: ConvNet is a sequence of Convolution Layers

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

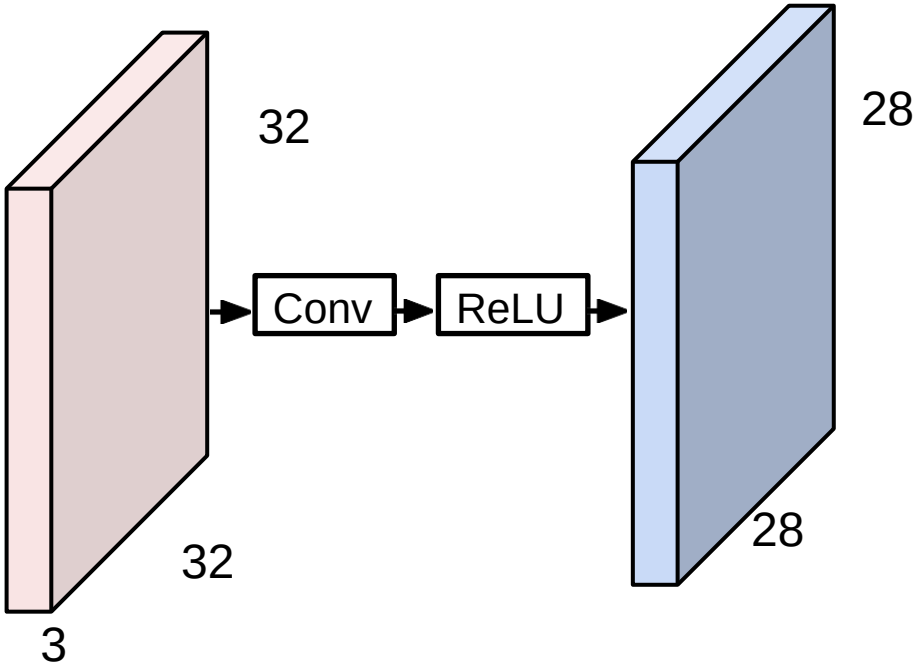


Preview: What do convolutional filters learn?

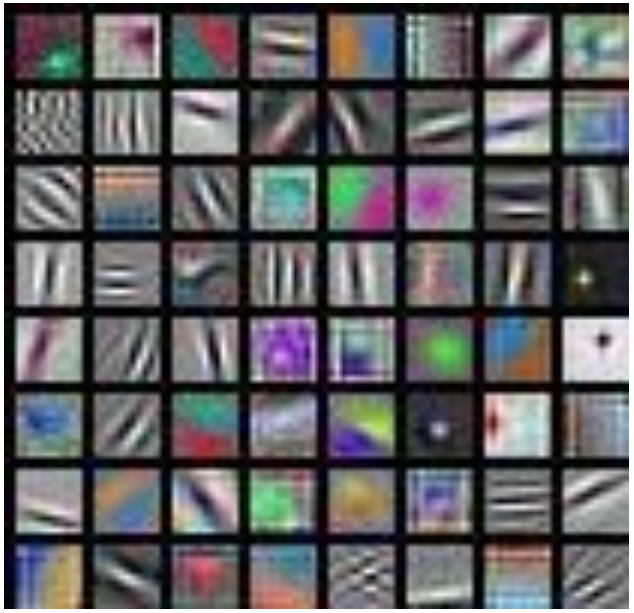


Linear classifier: One template per class

Preview: What do convolutional filters learn?

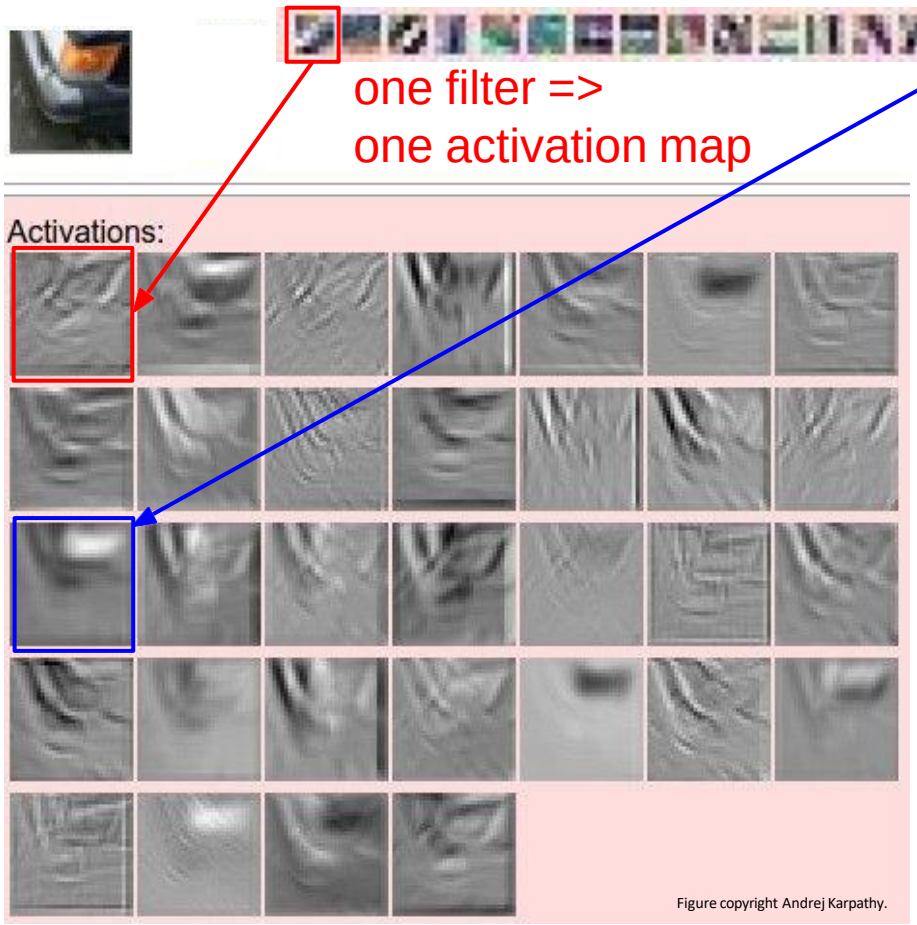


First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11

Preview



example 5x5 filters
(32 total)

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

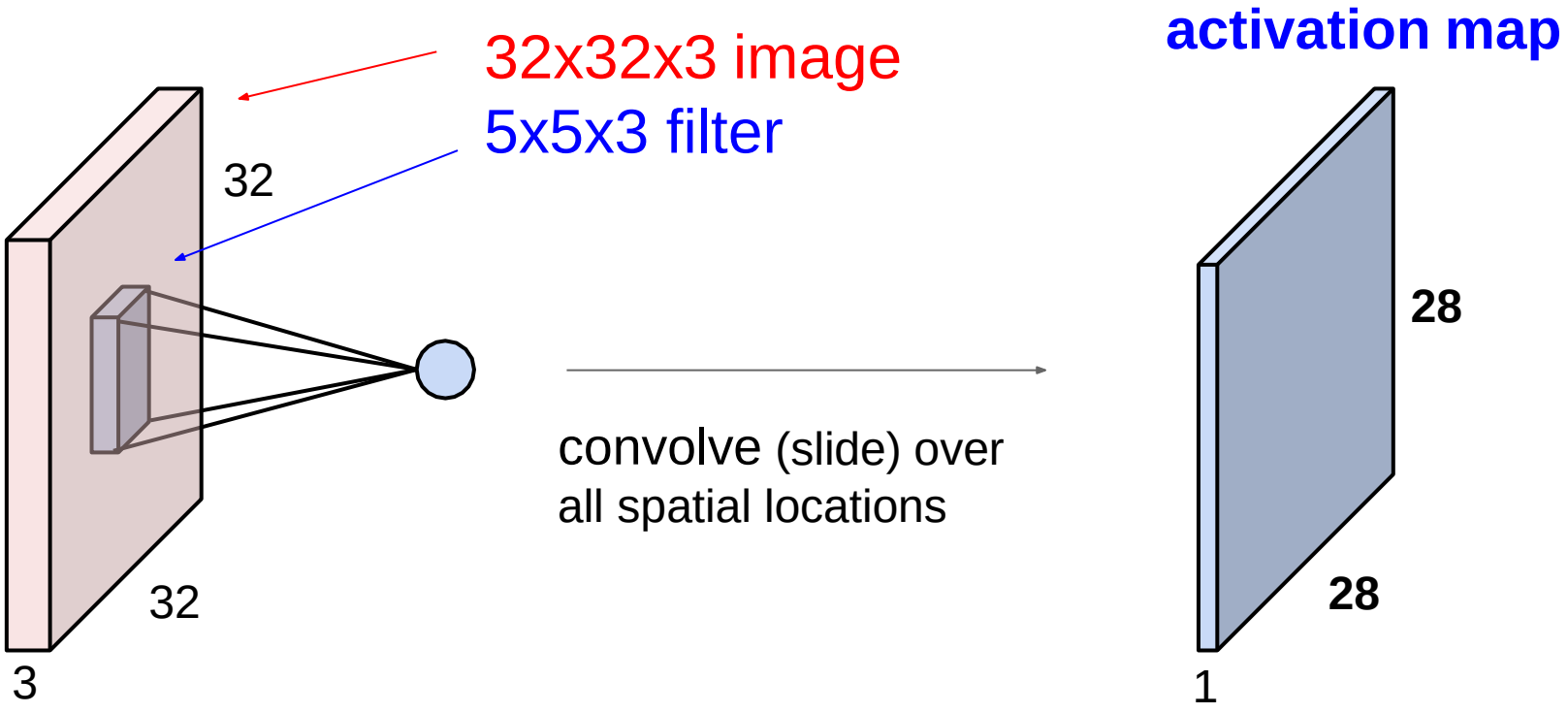
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

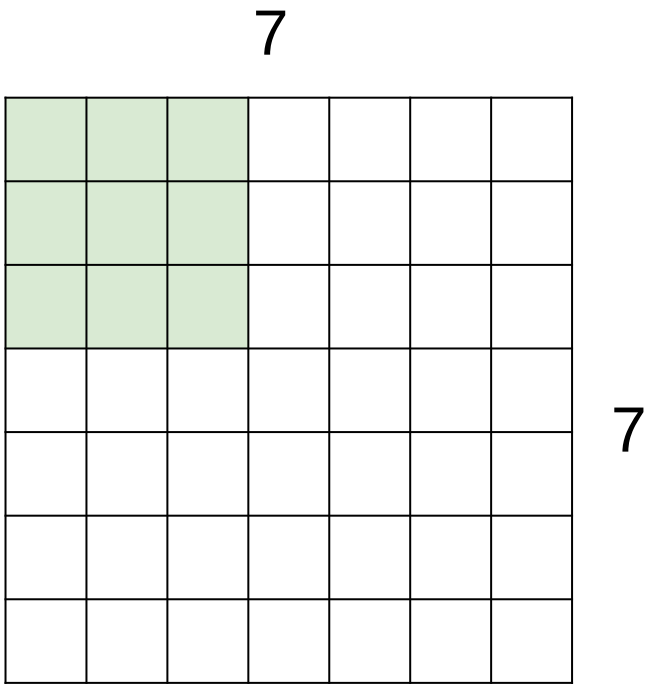
Preview


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:

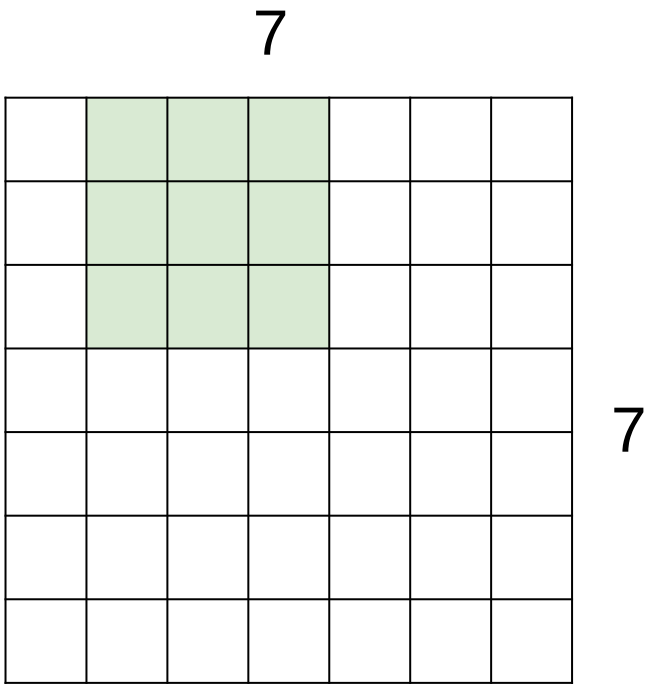


A closer look at spatial dimensions:



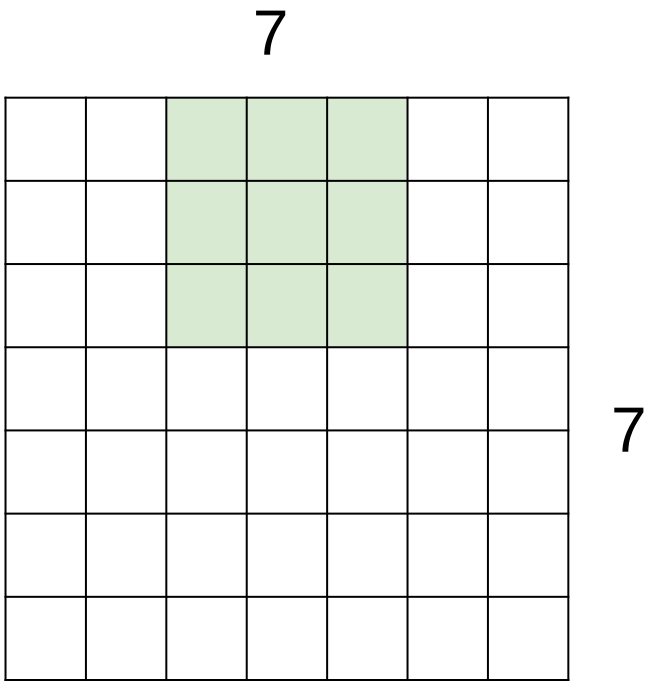
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



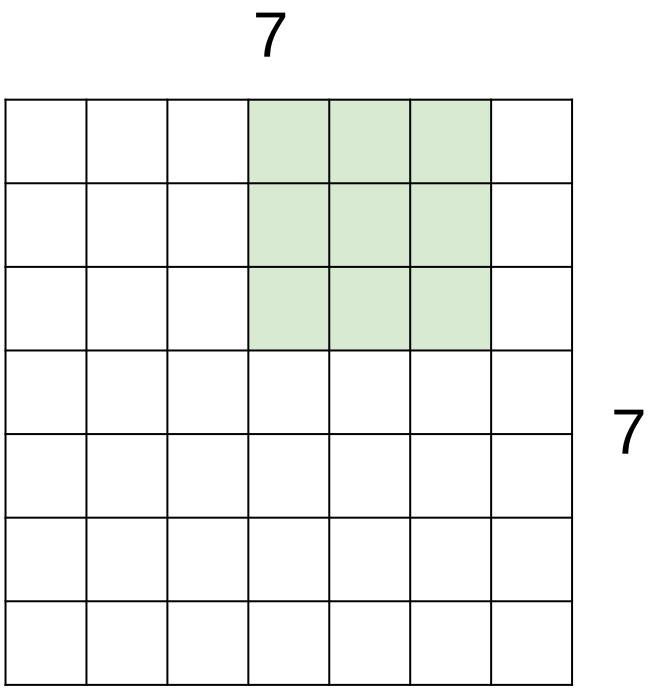
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



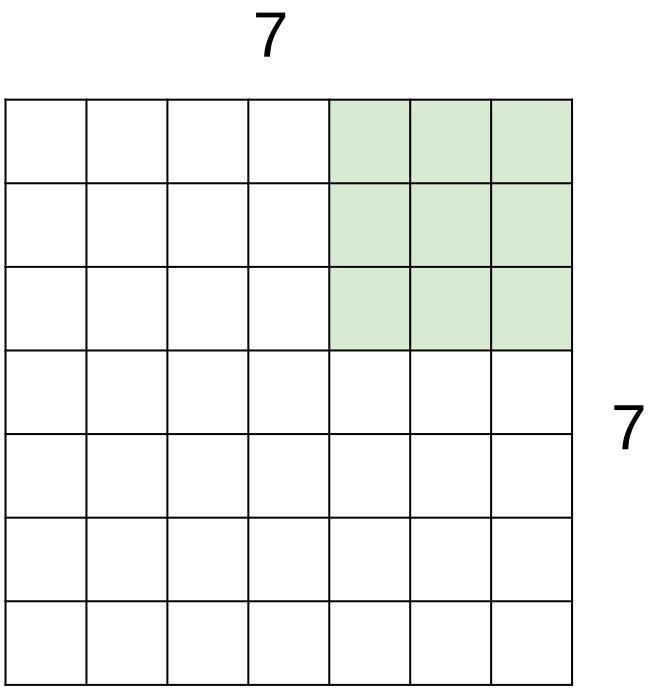
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

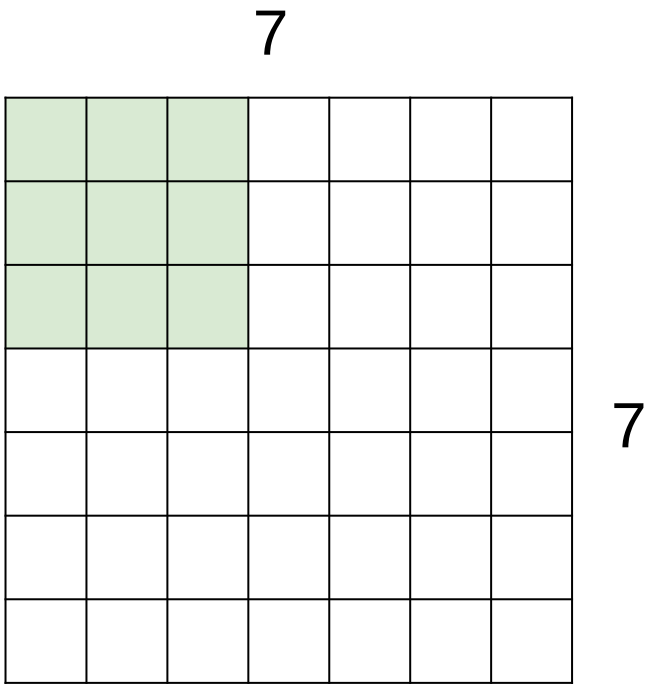
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

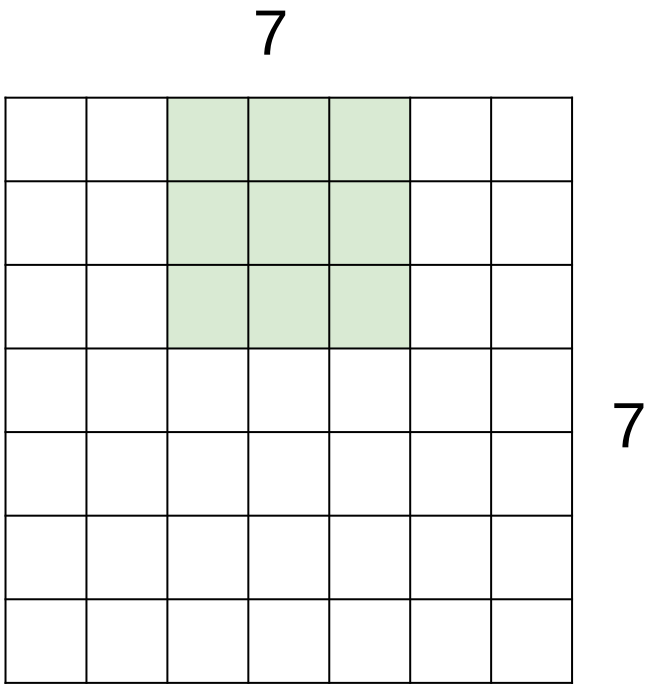
=> 5x5 output

A closer look at spatial dimensions:



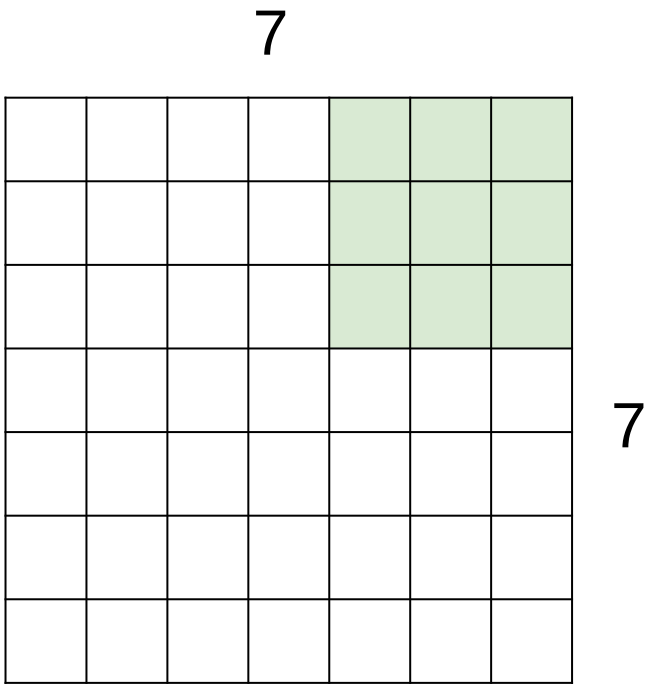
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



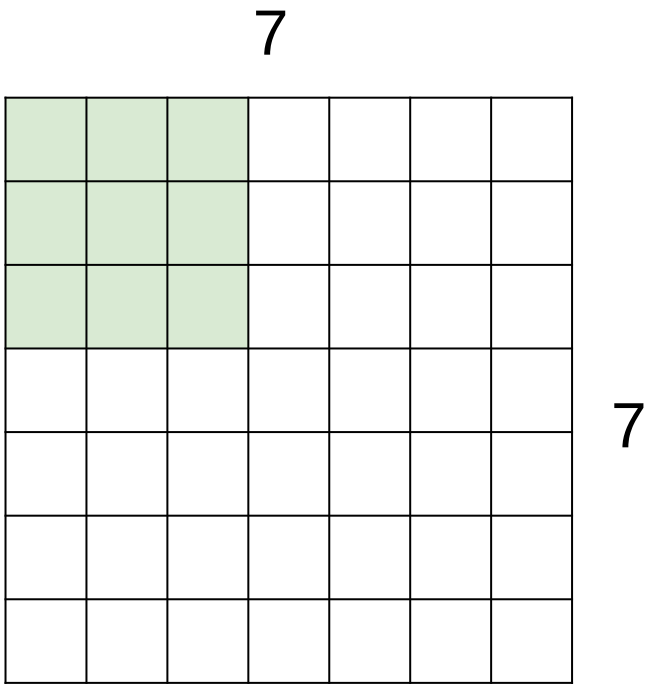
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



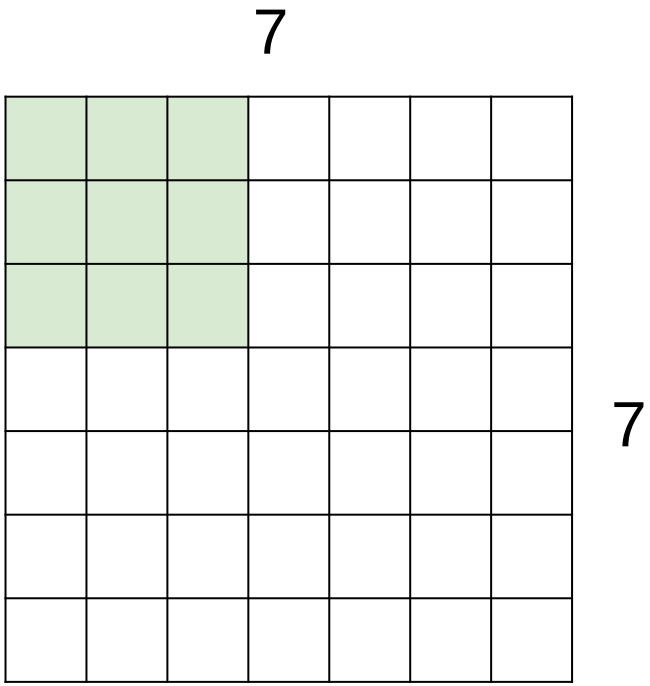
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

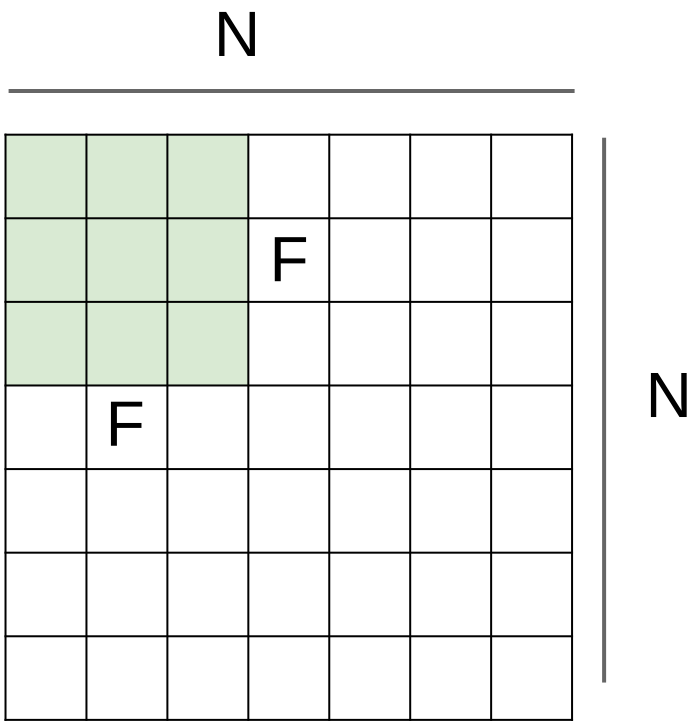
A closer look at spatial dimensions:



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

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

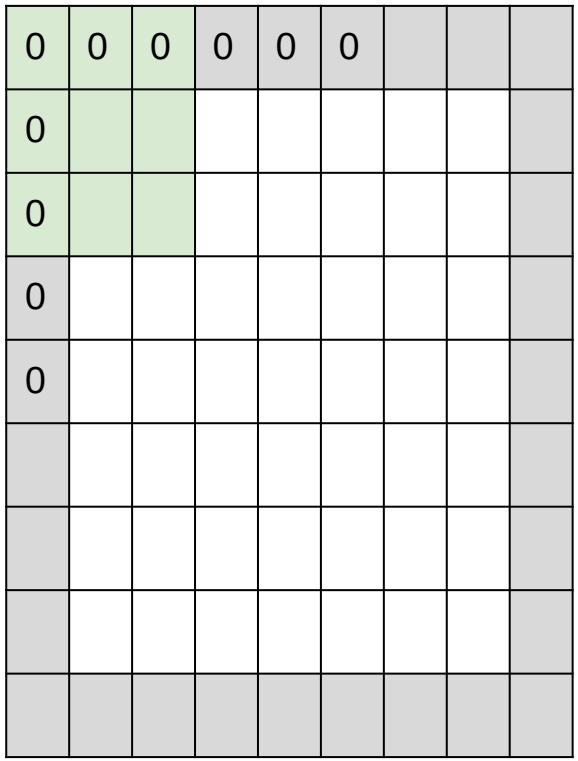
A closer look at spatial dimensions:



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 :\

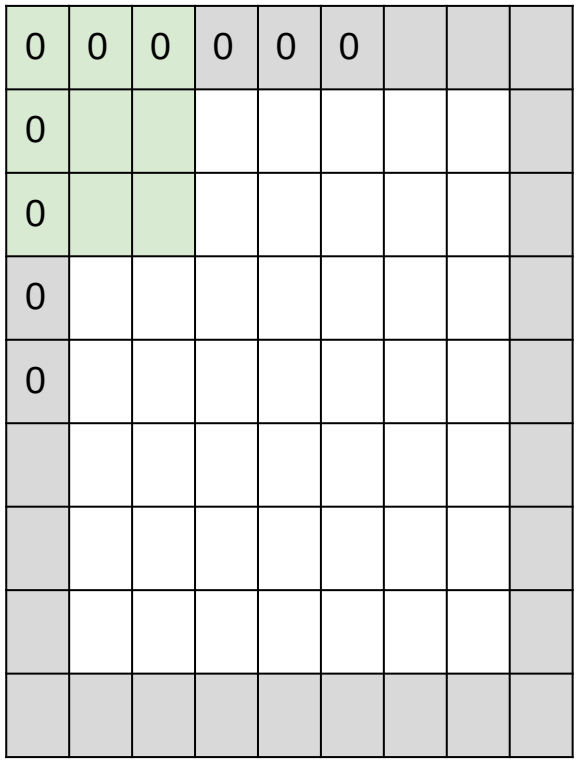
In practice: Common to zero pad the border



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

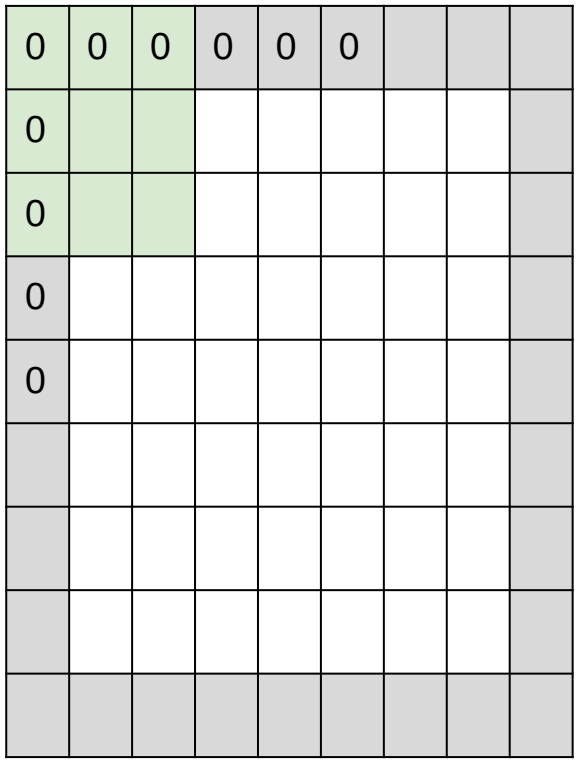


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border =>
what is the output?

7x7 output!

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

In practice: Common to zero pad the border



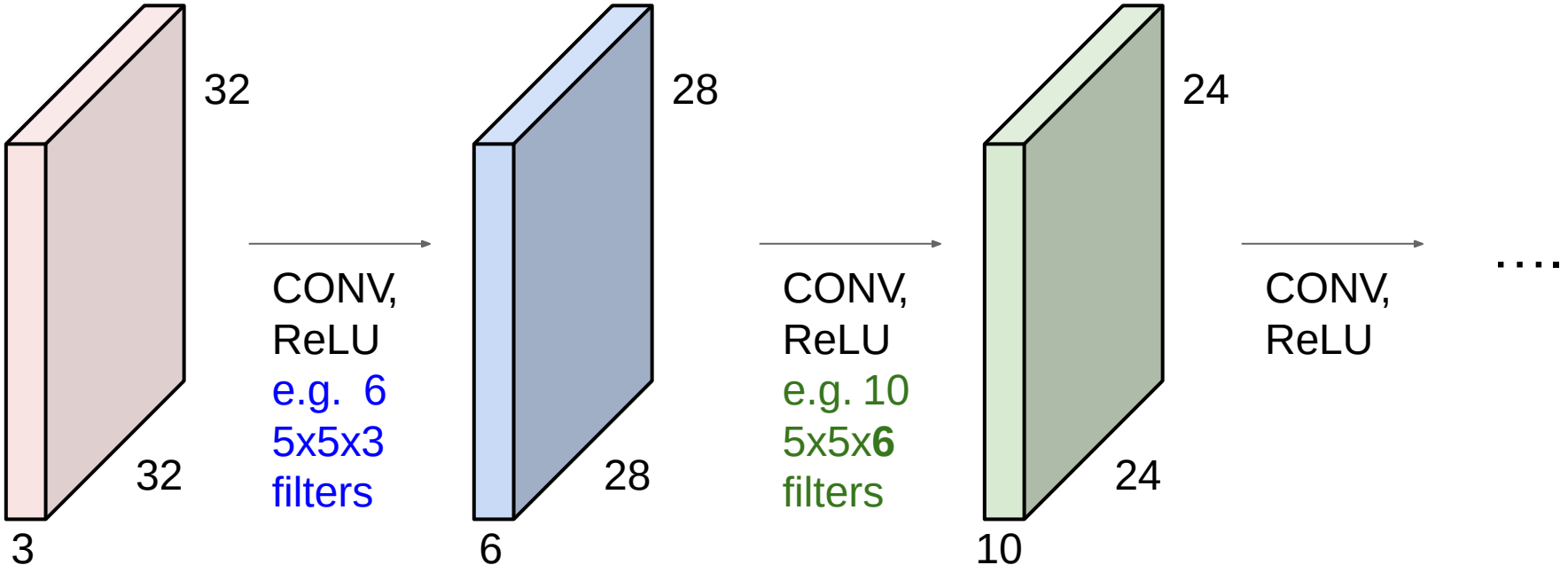
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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.

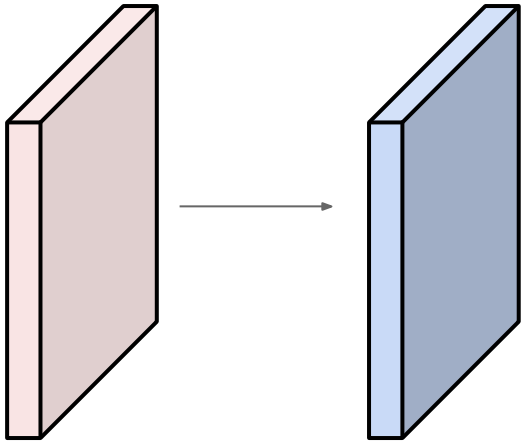


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

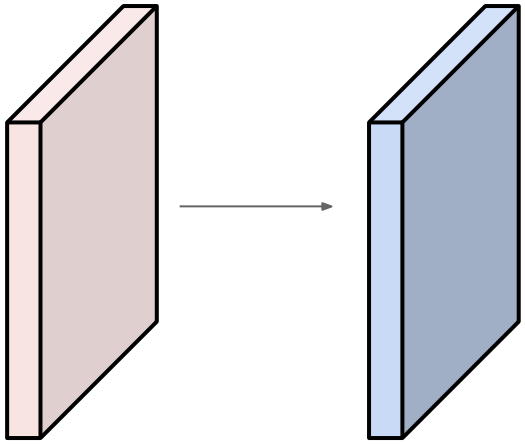
Output volume size: ?



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Output volume size:

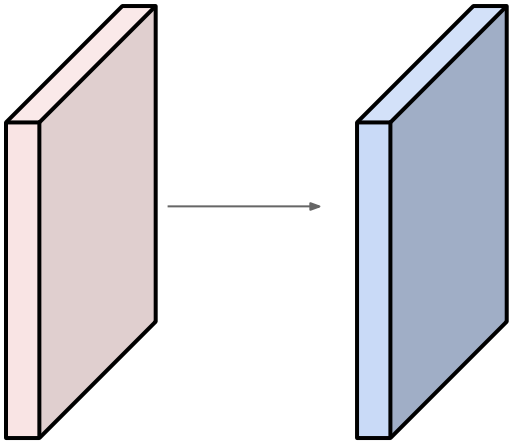
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

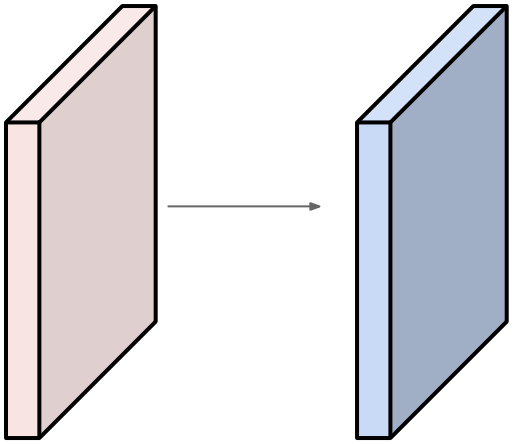
Number of parameters in this layer?



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

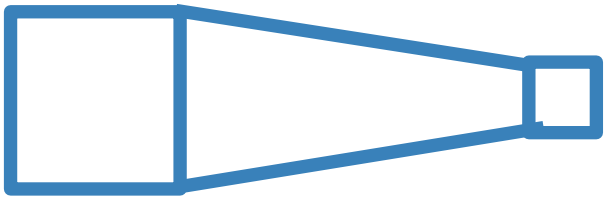


Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params
=> 76*10 = **760**

Receptive Fields

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input

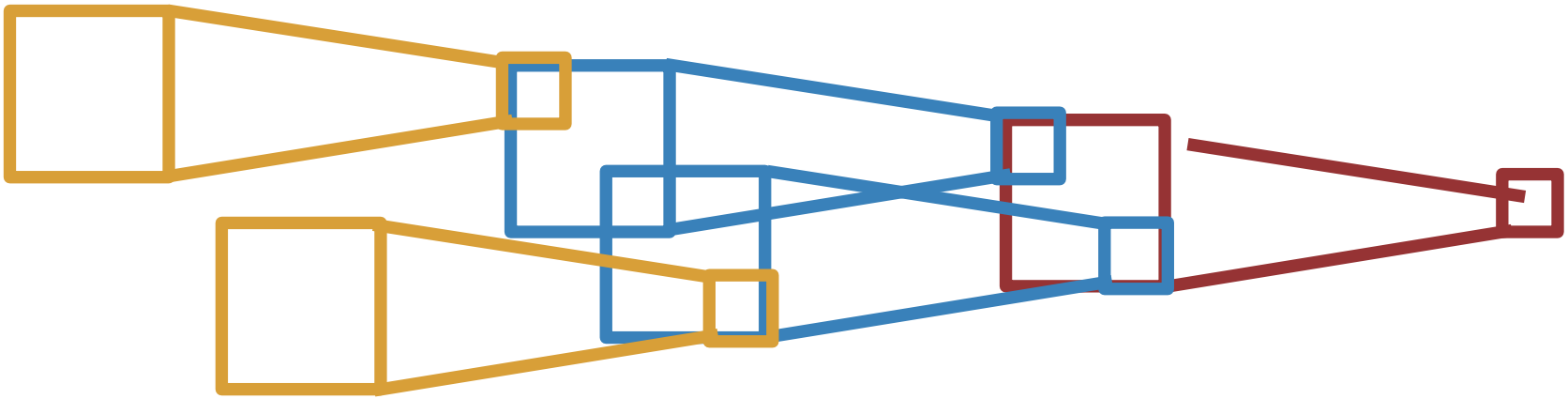


Input Output

Slide inspiration: Justin Johnson

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size With L layers the receptive field size is 1 + L * (K – 1)



Input

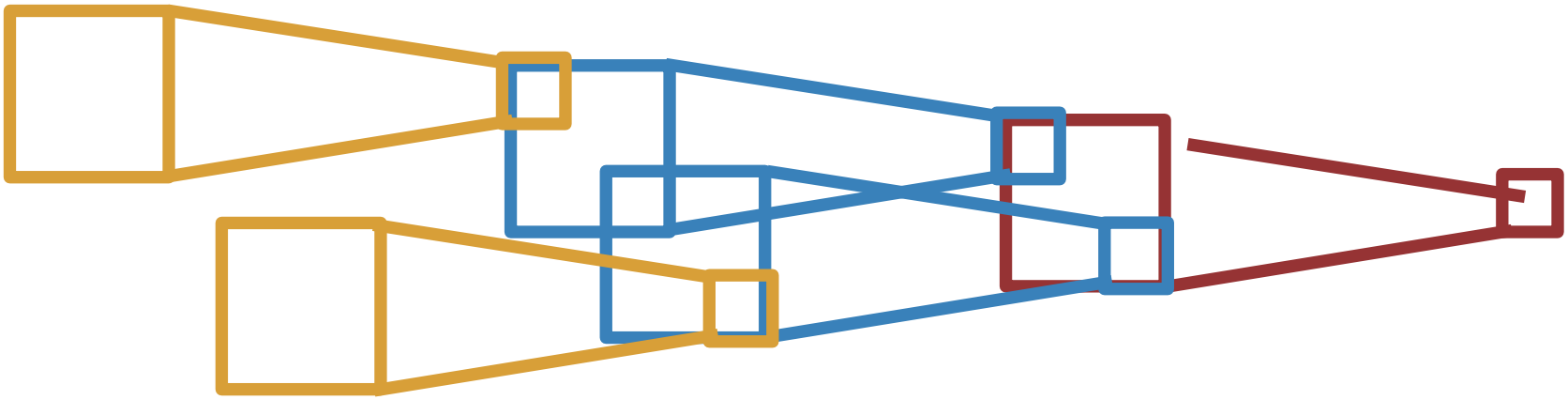
Output

Be careful – ”receptive field in the input” vs. “receptive field in the previous layer”

Slide inspiration: Justin Johnson

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Input

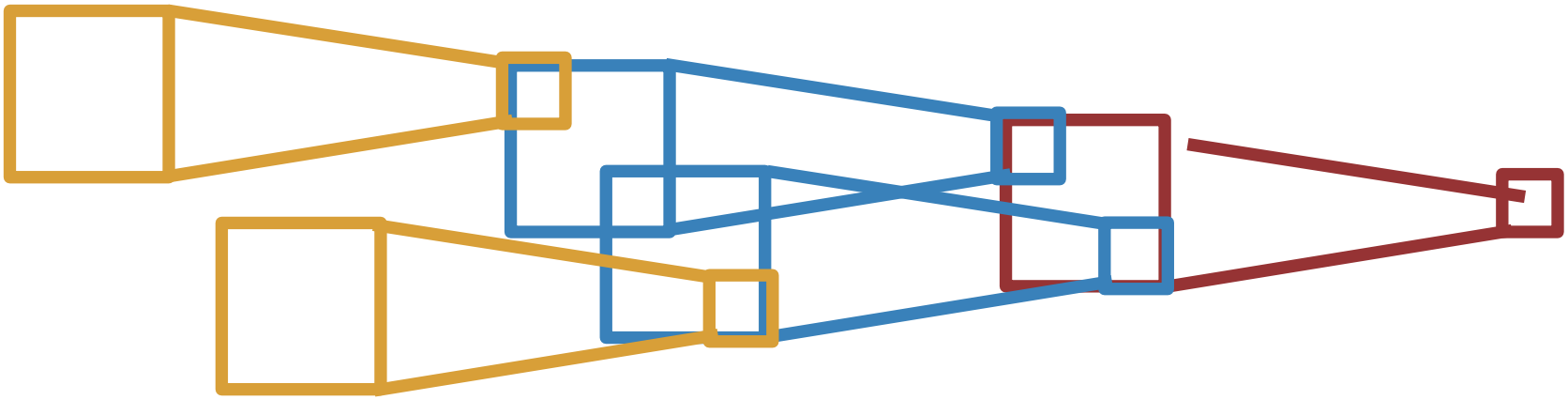
Output

Problem: For large images we need many layers for each output to “see” the whole image image

Slide inspiration: Justin Johnson

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size With L layers the receptive field size is 1 + L * (K – 1)



Input

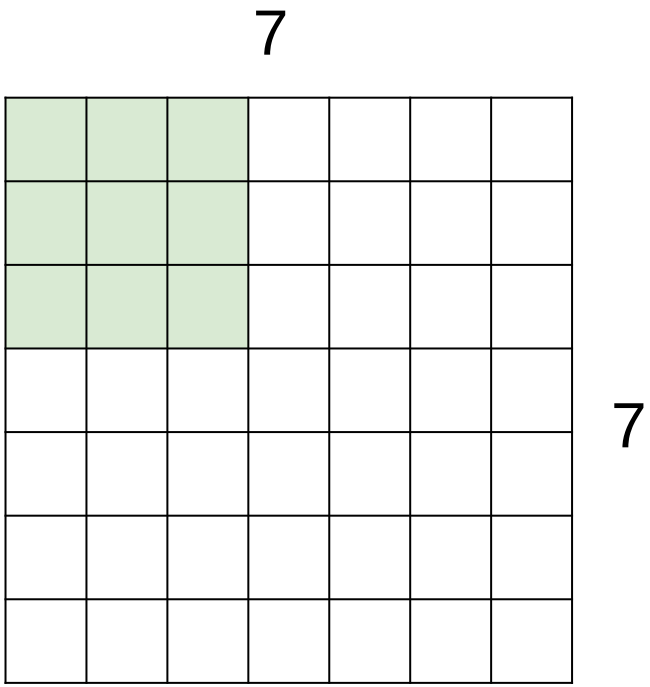
Output

Problem: For large images we need many layers for each output to “see” the whole image image

Solution: Downsample inside the network

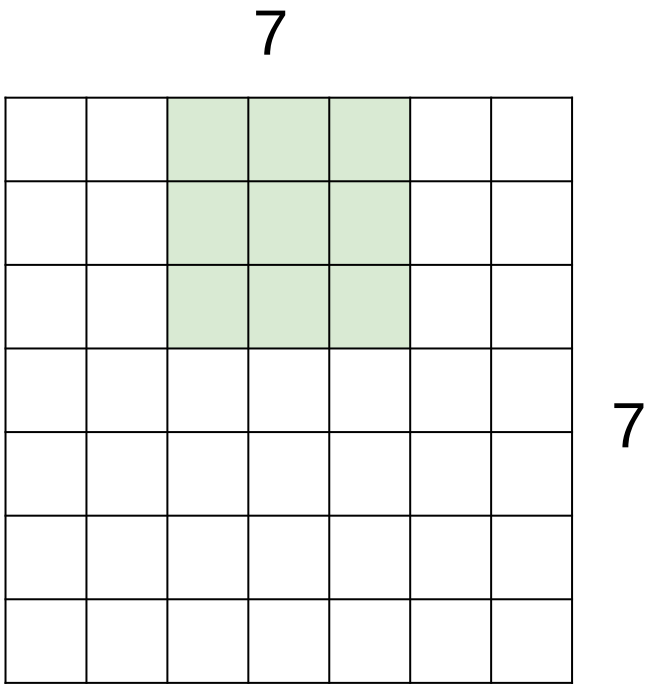
Slide inspiration: Justin Johnson

Solution: Strided Convolution



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

Solution: Strided Convolution



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

=> 3x3 output!

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**

Common settings:

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

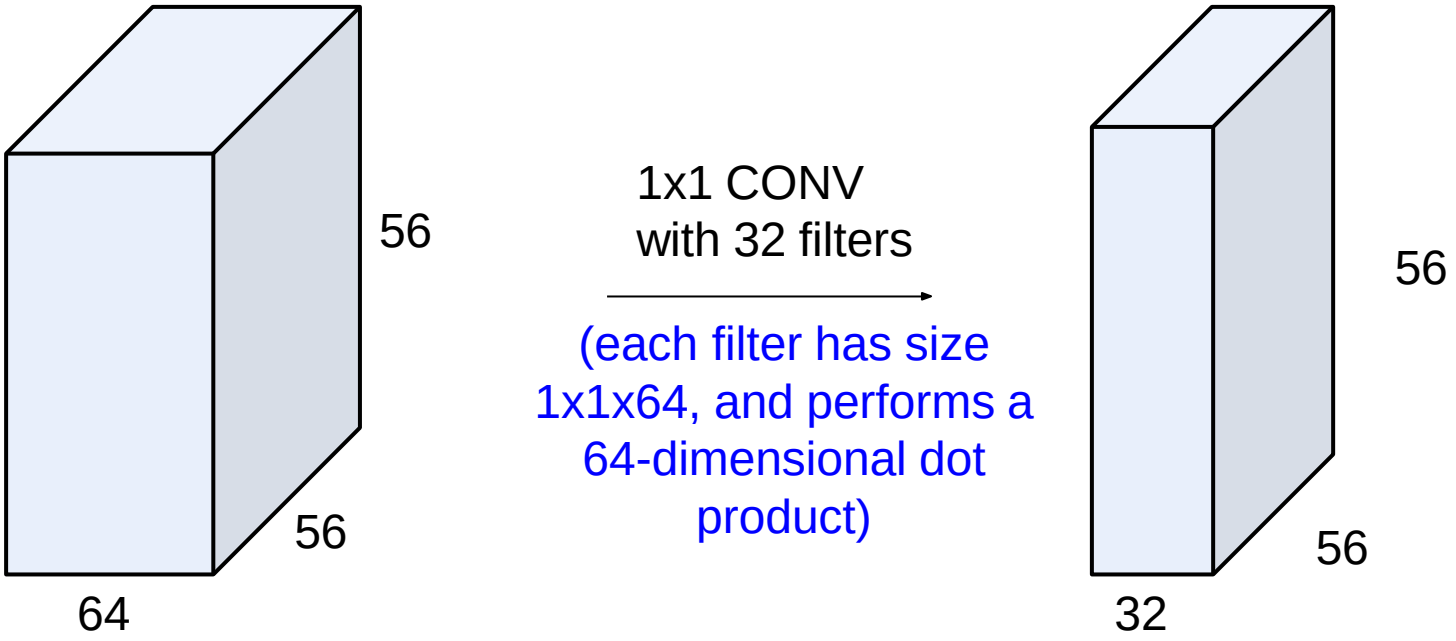
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

This will produce an output of W_2 x H_2 x K where:

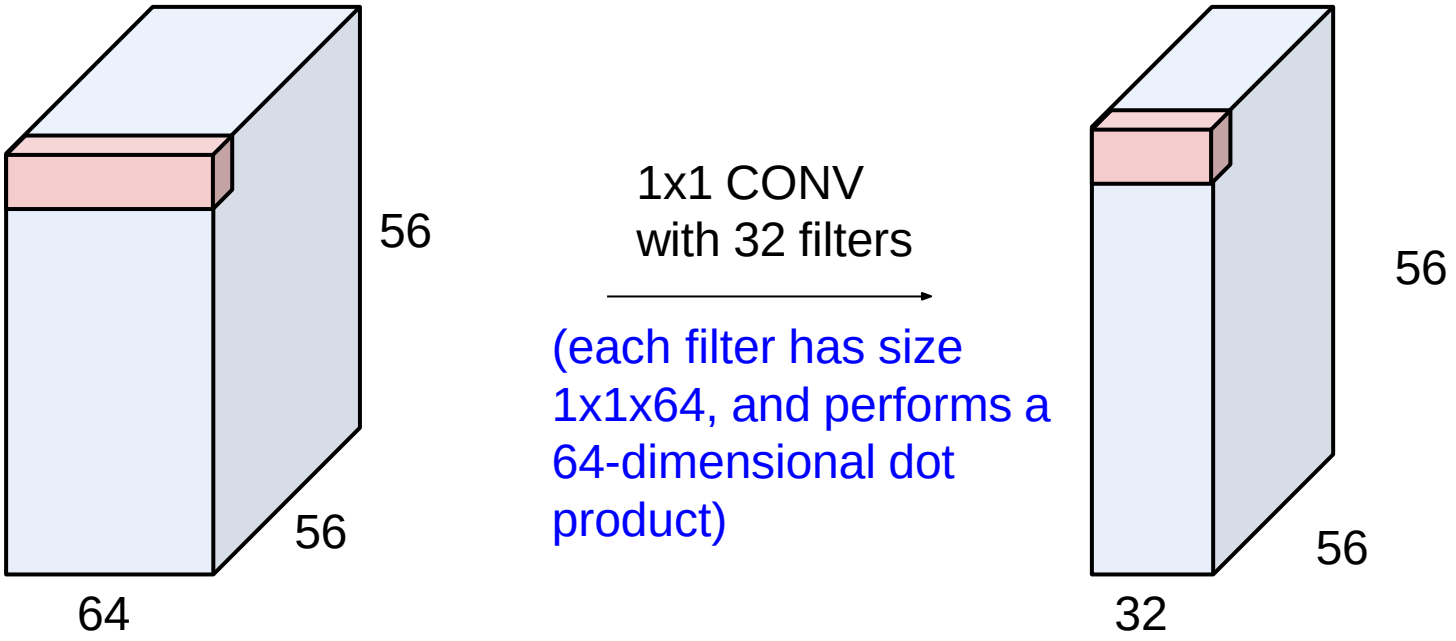
- $W_2 = (W_1 - F + 2P) / S + 1$
- $H_2 = (H_1 - F + 2P) / S + 1$

Number of parameters: F^2CK and K biases

1x1 convolution layers make perfect sense



1x1 convolution layers make perfect sense



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

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

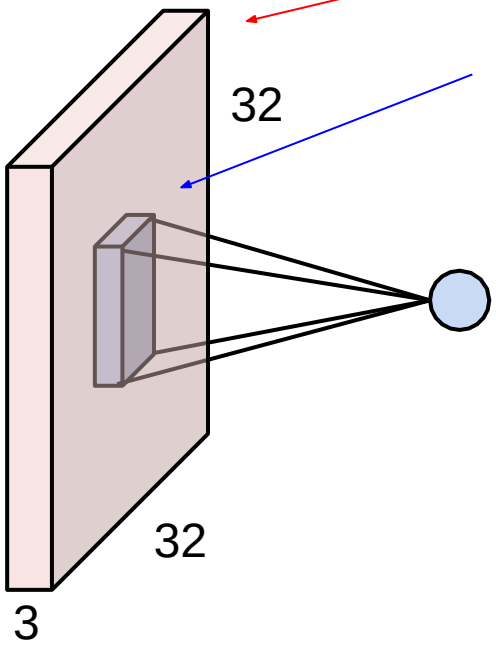
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `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:

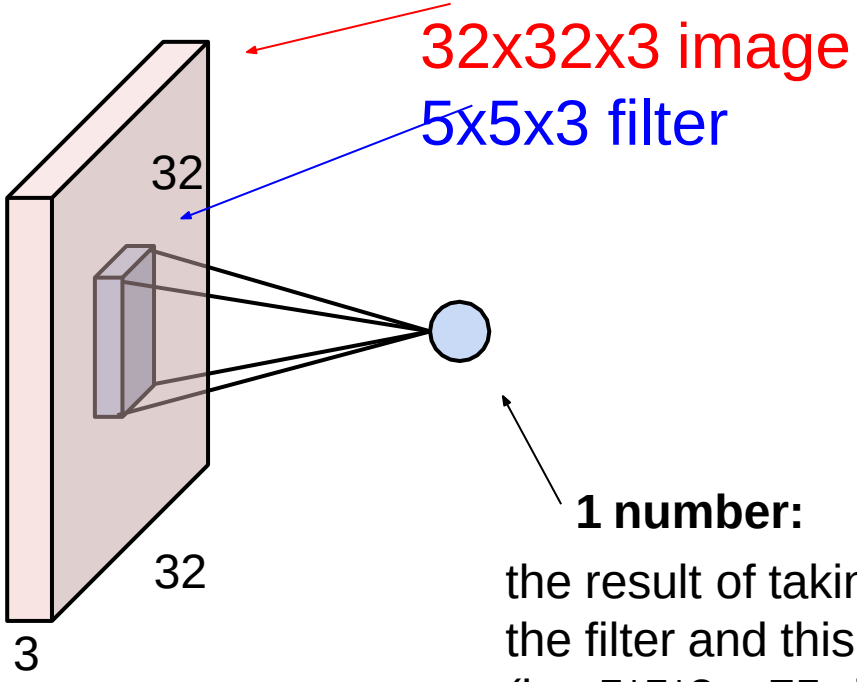
- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

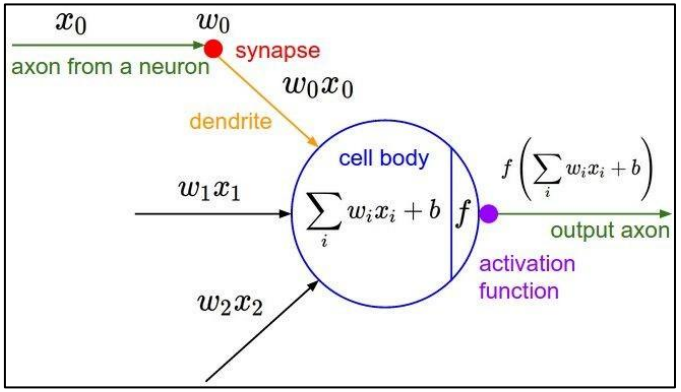
The brain/neuron view of CONV Layer



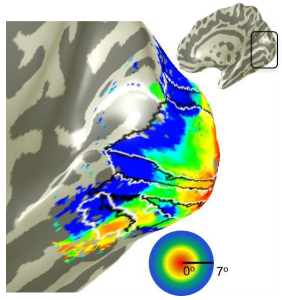
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer

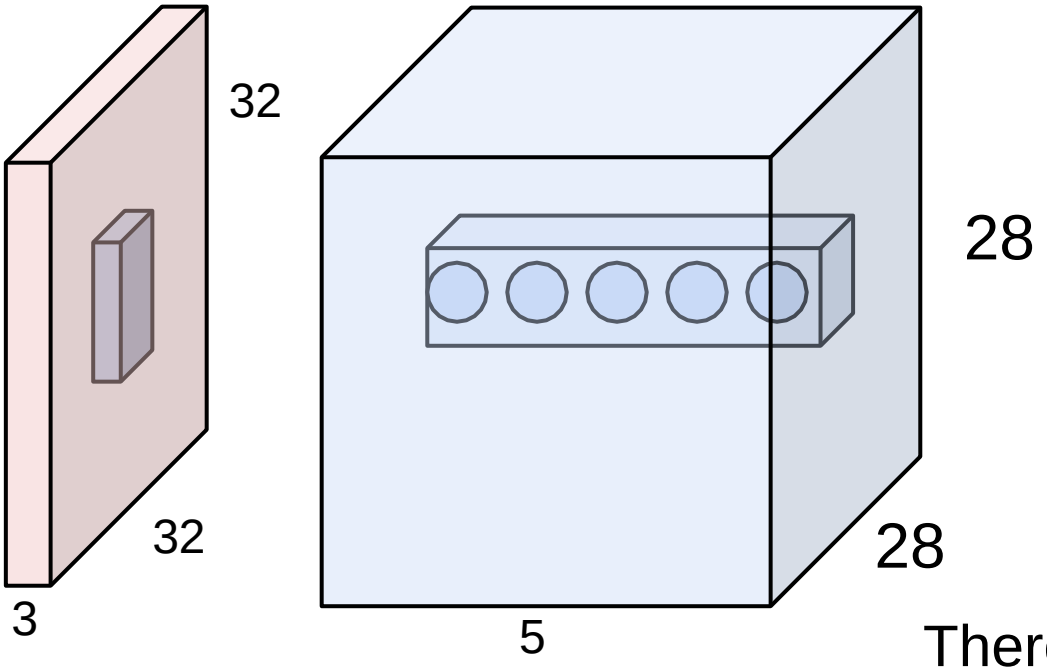


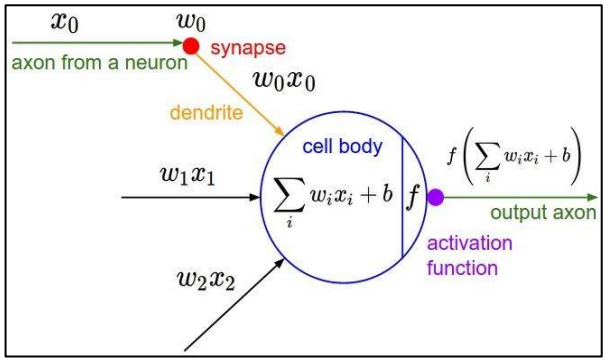


It's just a neuron with local connectivity...



The brain/neuron view of CONV Layer





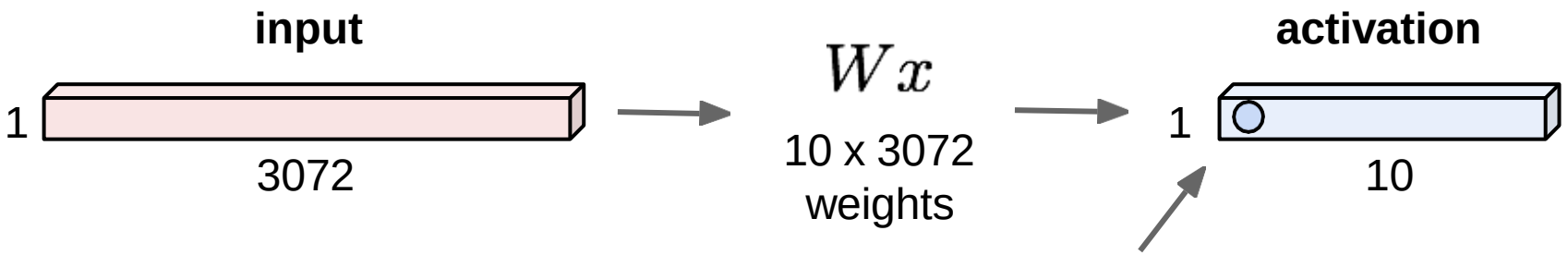
E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

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

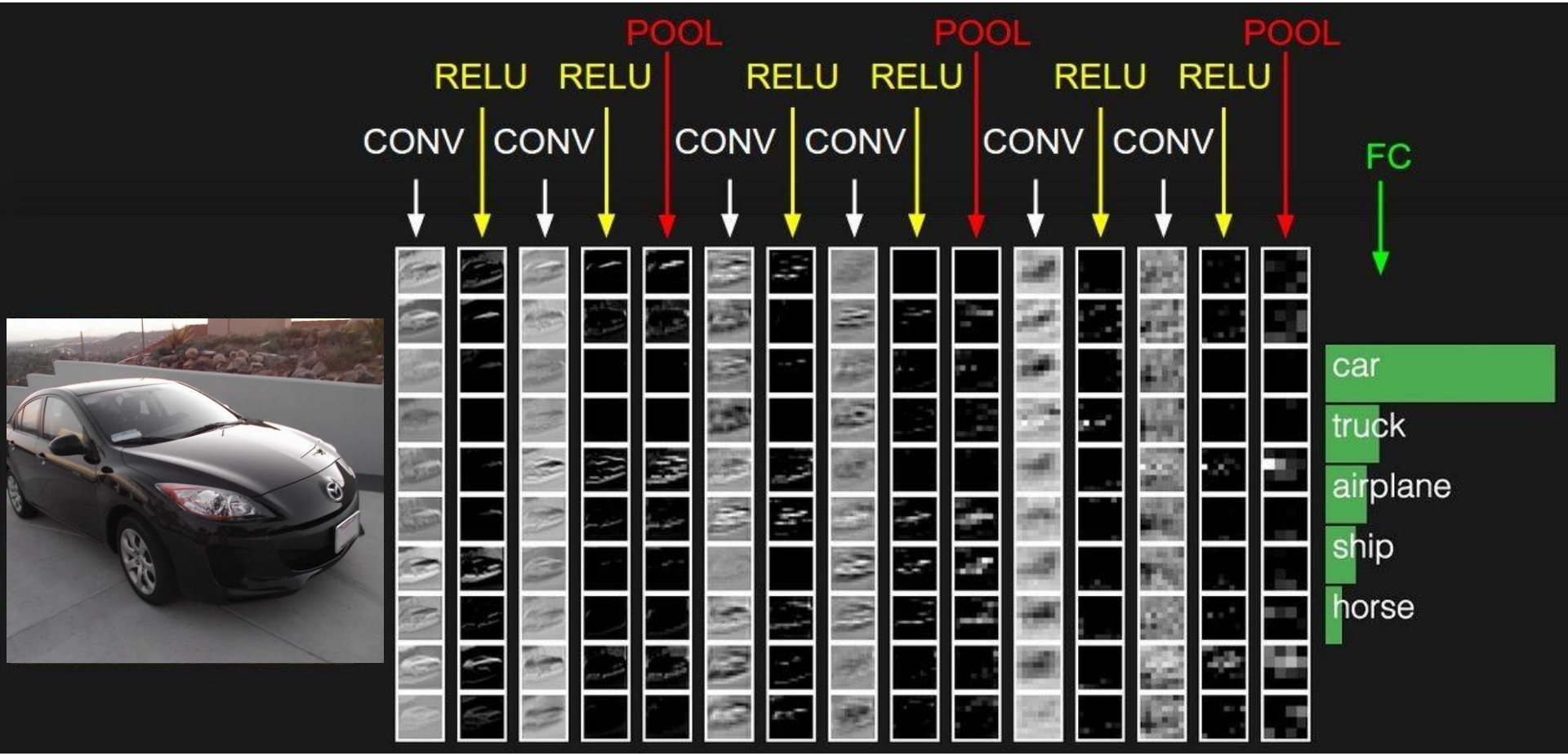
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

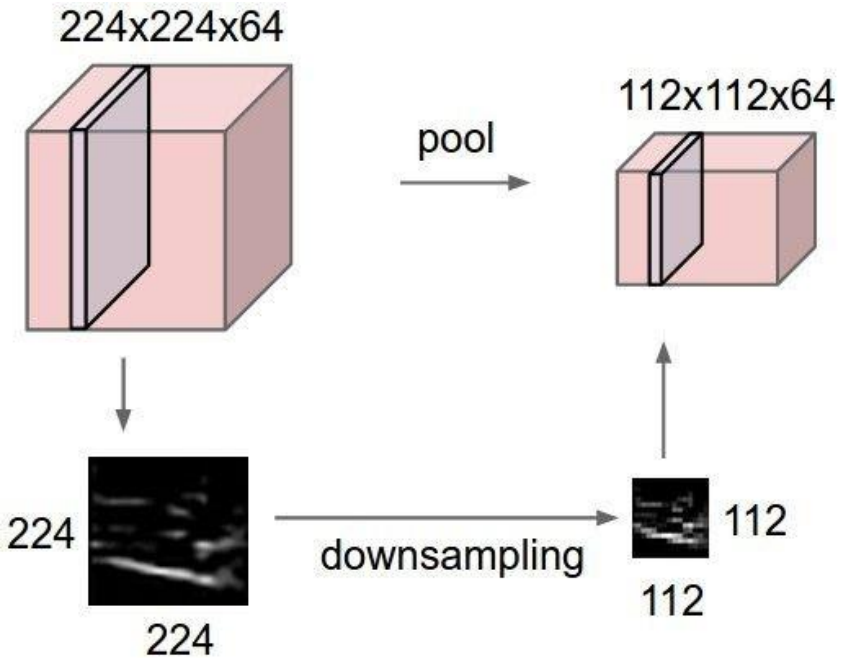


1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

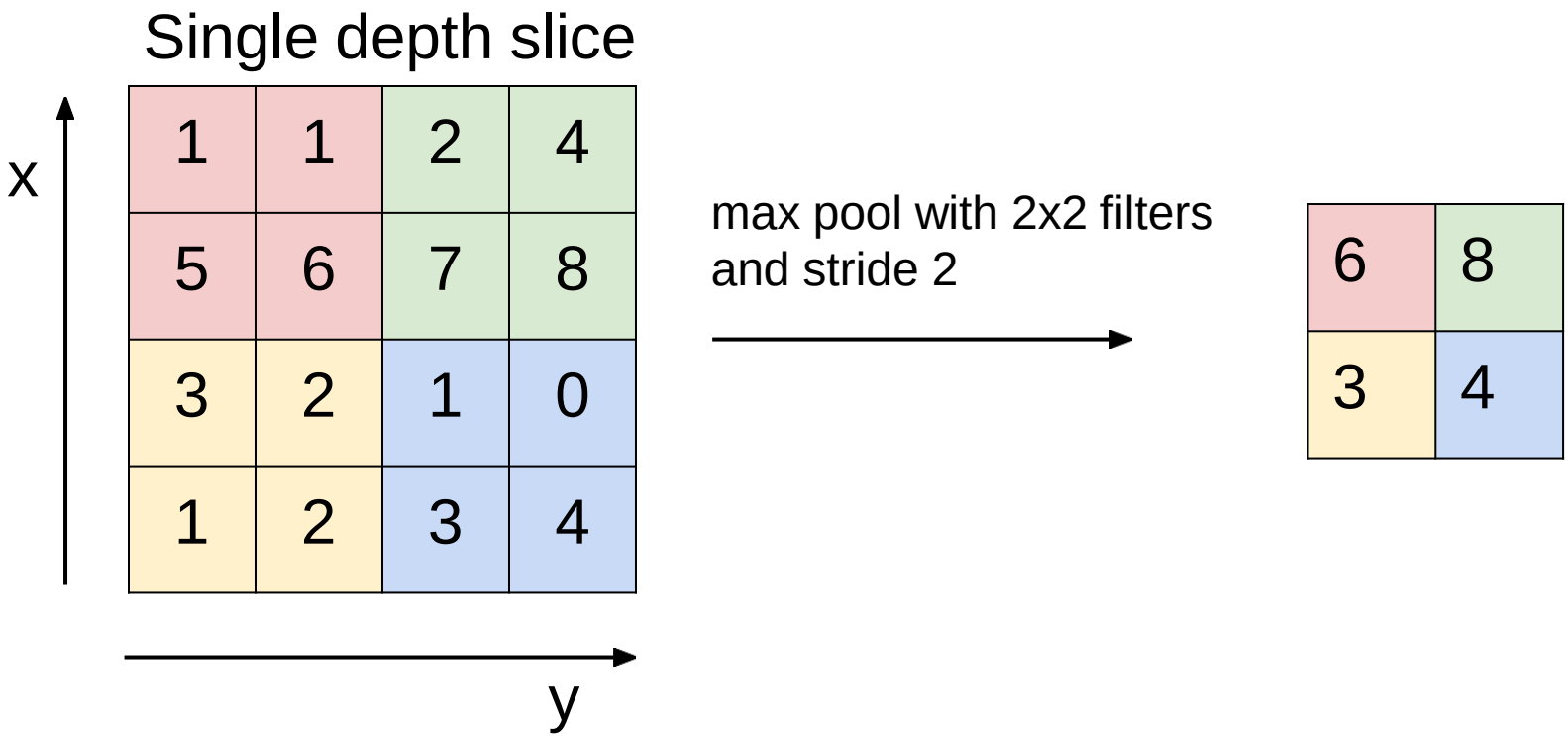

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Pooling layer

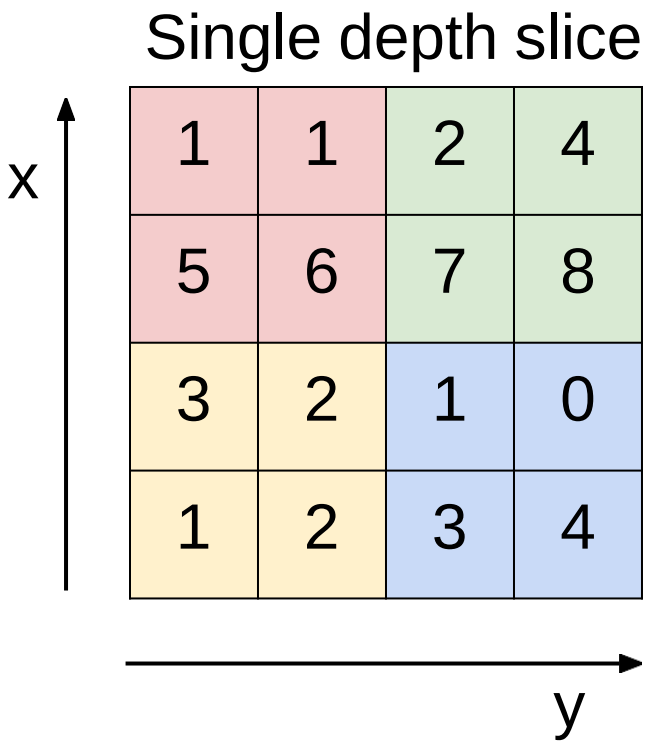
- makes the representations smaller and more manageable
- operates over each activation map independently



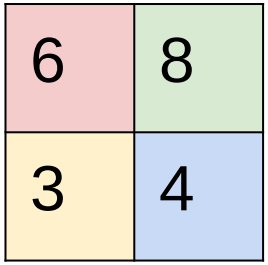
MAX POOLING


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

MAX POOLING



max pool with 2x2 filters
and stride 2



- No learnable parameters
- Introduces spatial invariance

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**

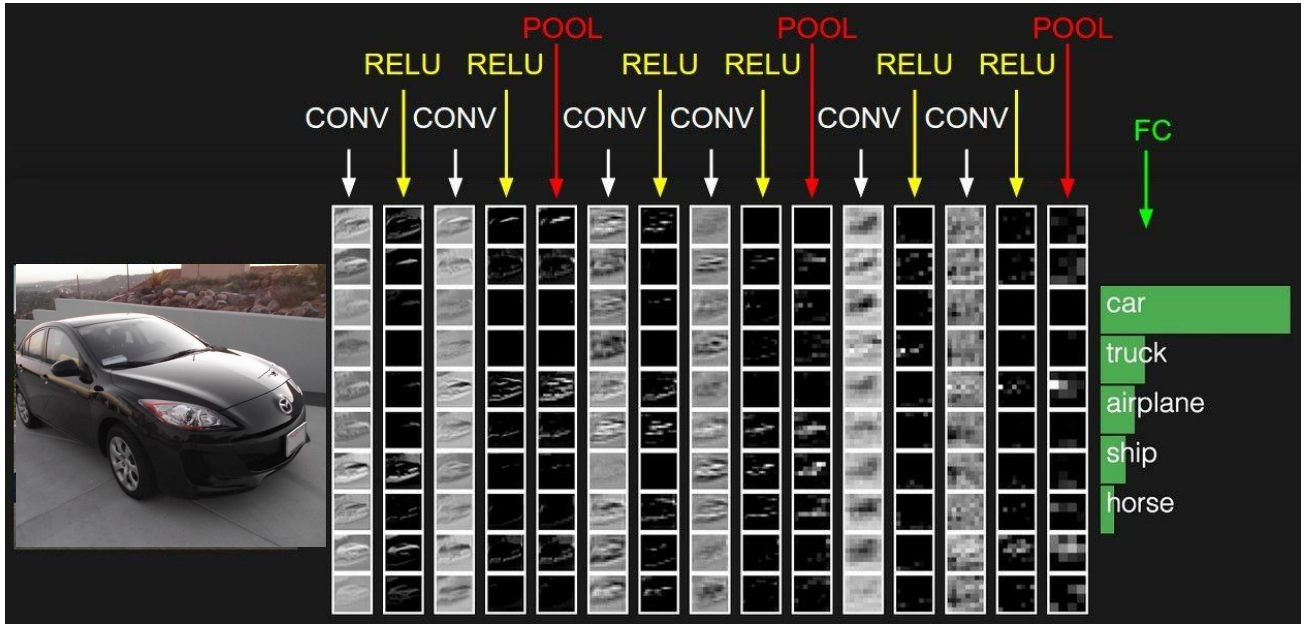
This will produce an output of W_2 x H_2 x C where:

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

Number of parameters: 0

Fully Connected Layer (FC layer)

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



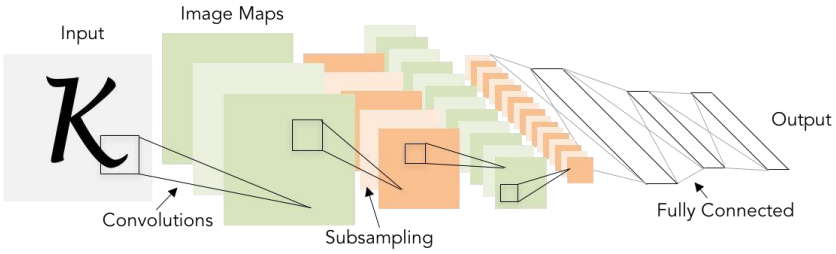
Summary

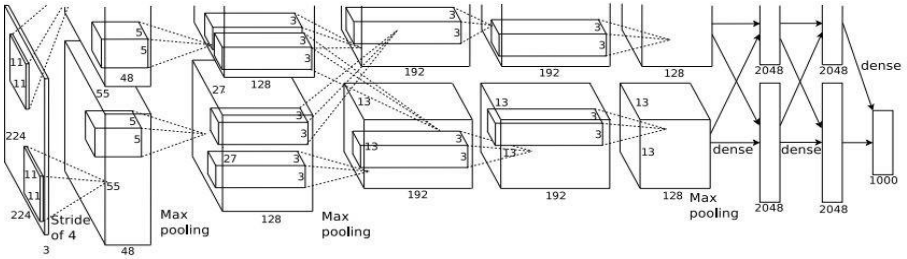
- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)

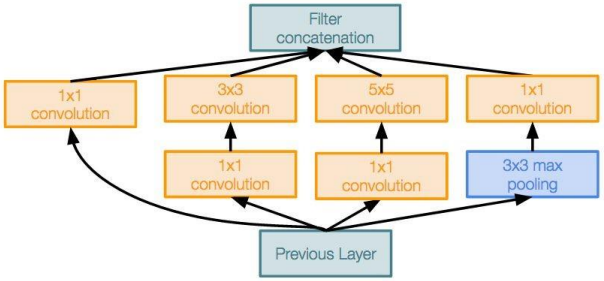
- Historically architectures looked like
  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K, SOFTMAX
  where **N** is usually up to ~5, **M** is large, 0 <= **K** <= 2.

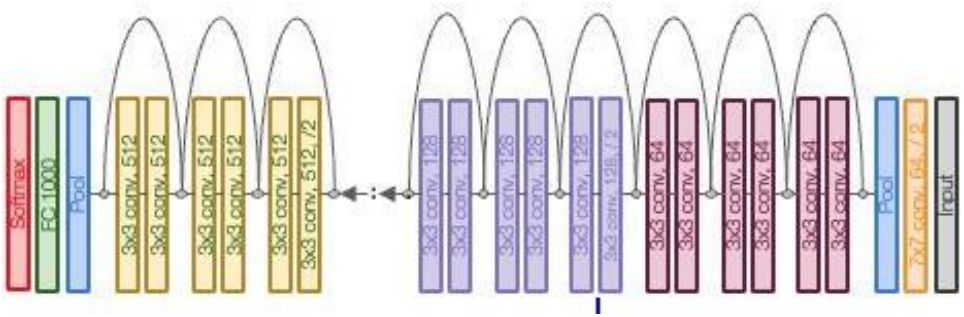
-But recent advances have challenged this paradigm

Next time: CNN Architectures









中山大學
SUN YAT-SEN UNIVERSITY

Next time:

CNN Architectures

Pattern Recognition and Computer Vision

Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

Most of our slides have been borrowed from the courses of Stanford CS131 and CS231N, Thanks!