

Lecture 9: Image Classification with CNNs

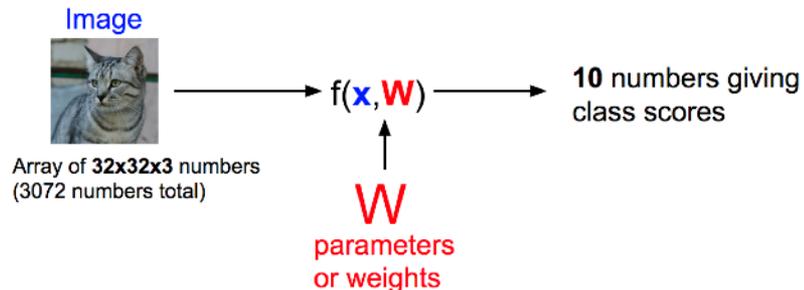
Administrative

Assignment 2 due March 26, 11:55pm

Midterm 1, Tuesday March 10, 4-5:15pm
Syllabus includes topics till lecture 8 (Feb 26)
Practice midterm on piazza

Project discussion Thursday next week
Project proposals due March 31

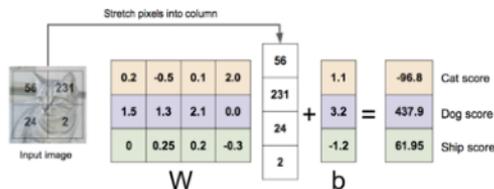
Recap: Image Classification with Linear Classifier



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

Algebraic Viewpoint

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



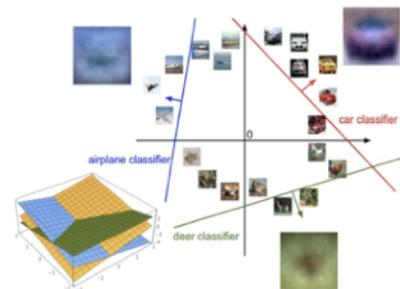
Visual Viewpoint

One template
per class



Geometric Viewpoint

Hyperplanes
cutting up space



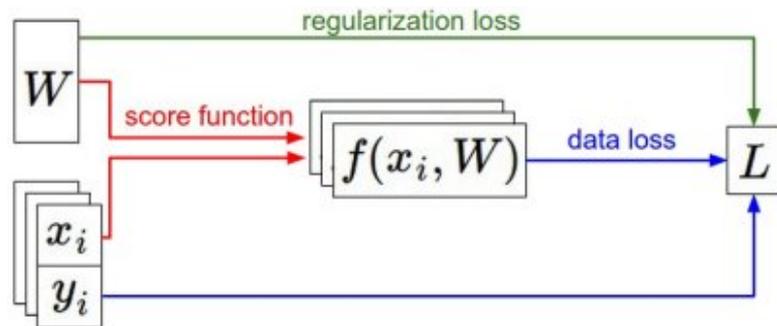
Recap: Loss Function

- We have some dataset of (x,y)
- We have a **score function**: $s = f(x; W) = Wx$
- We have a **loss function**:

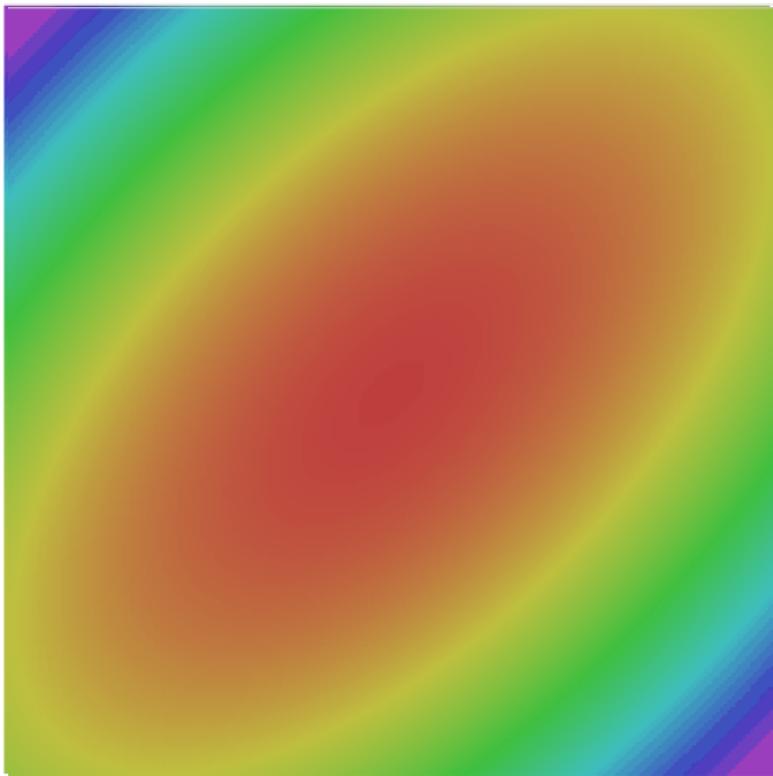
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \text{ Softmax}$$

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

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \text{ Full loss}$$



Recap: Optimization



- SGD
- SGD+Momentum
- RMSProp
- Adam

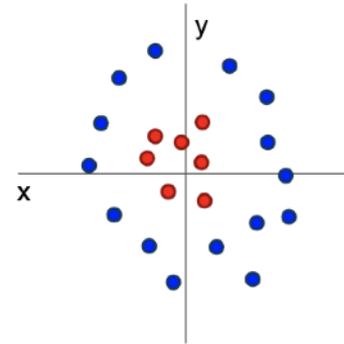
Problem: Linear classifiers are not very powerful

Visual Viewpoint



Linear classifiers learn
one template per class

Geometric Viewpoint



Linear classifiers
can only draw linear
decision boundaries

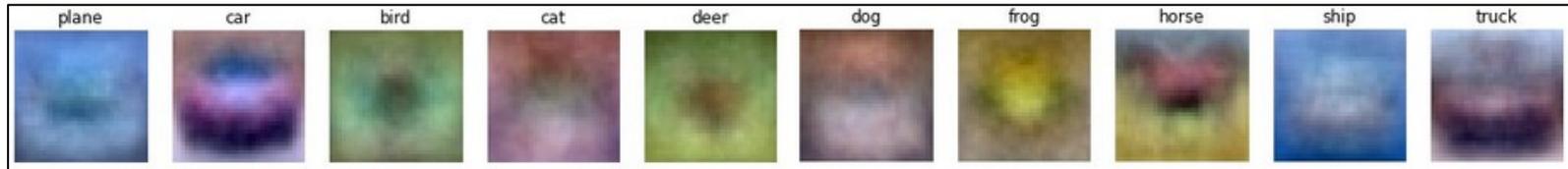
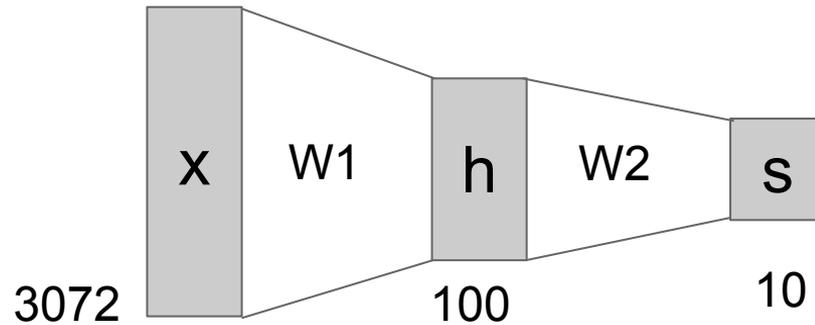
Multi-layer neural networks

Linear score function:

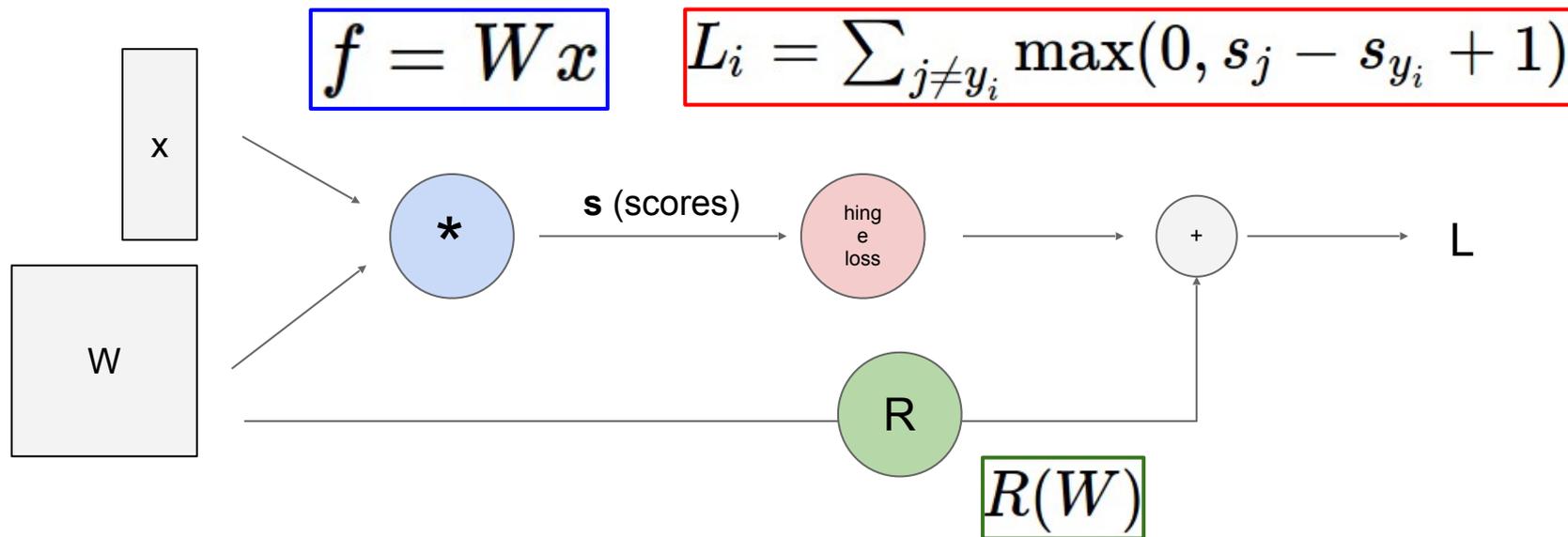
$$f = Wx$$

2-layer Neural Network

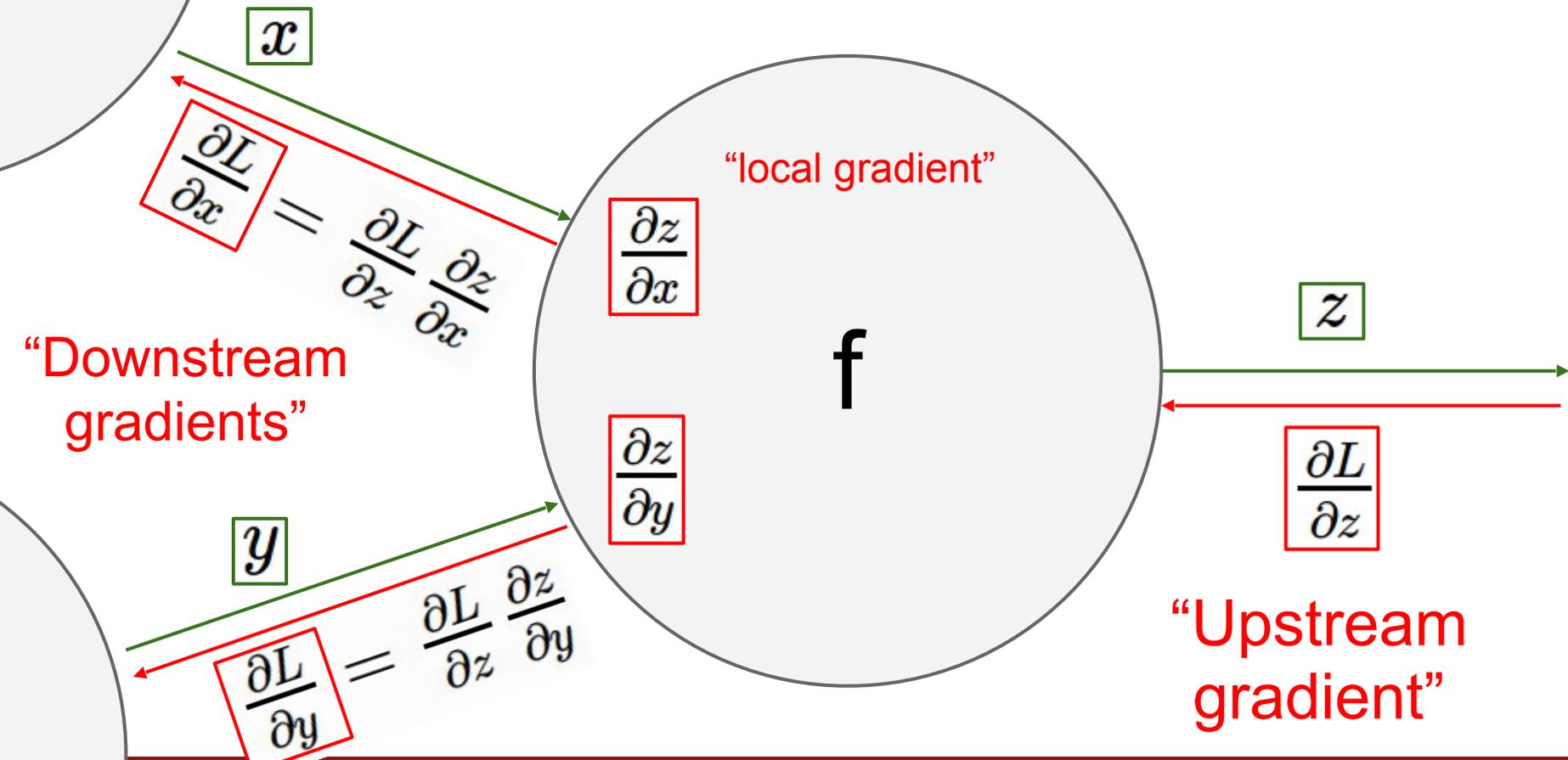
$$f = W_2 \max(0, W_1 x)$$



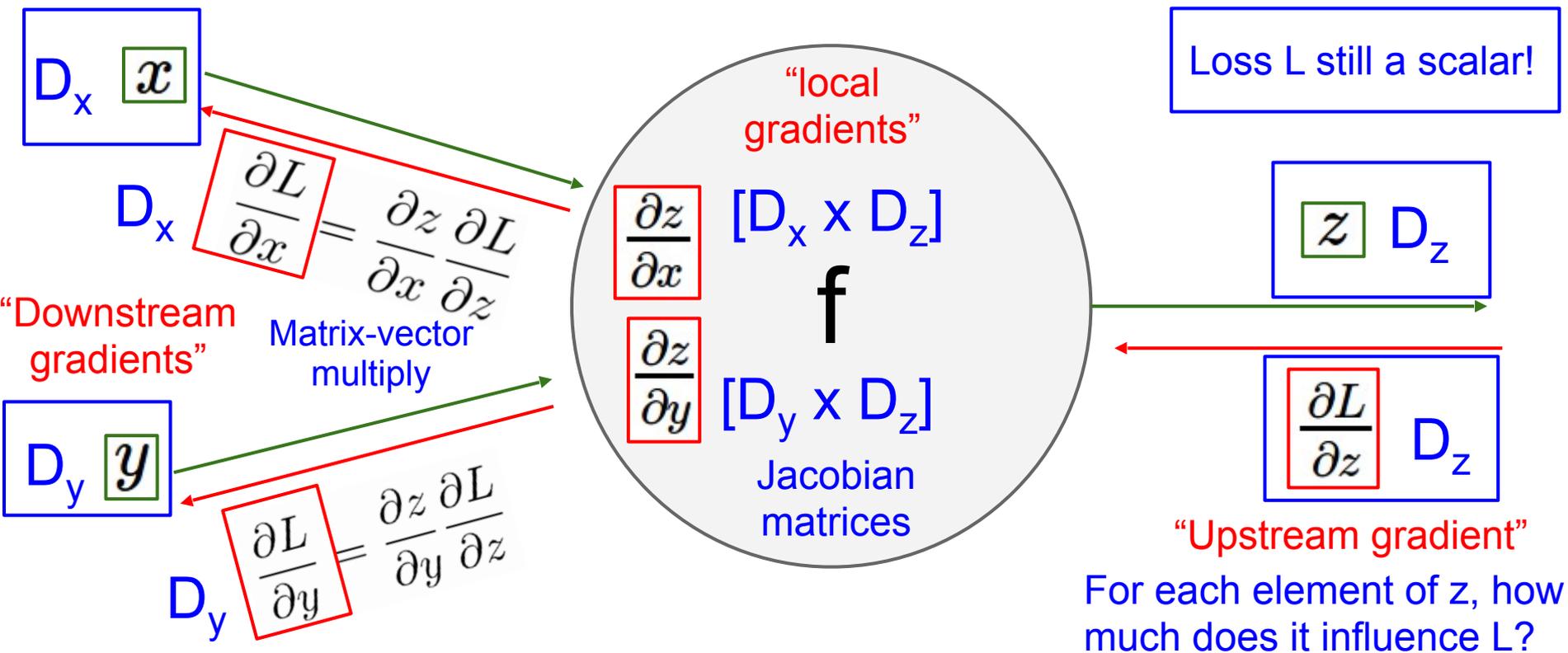
Last time: Computation Graph



Last time: Backpropagation



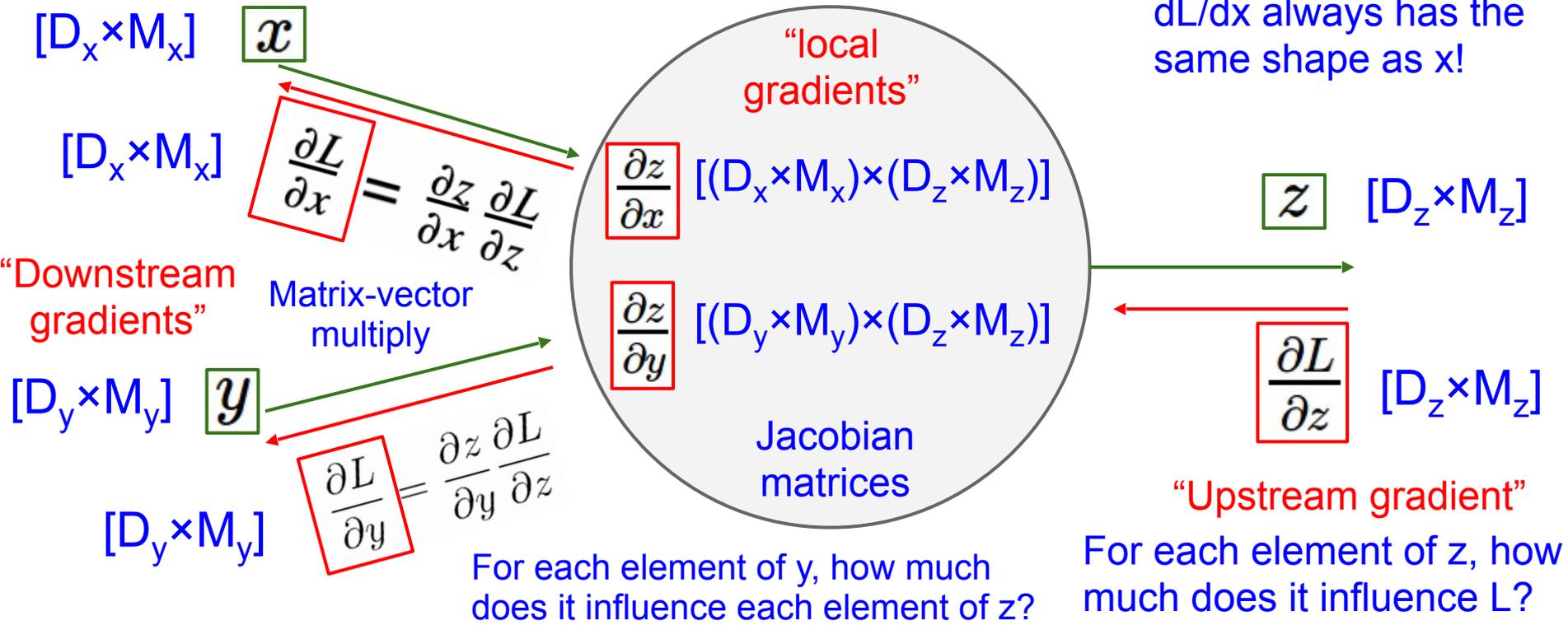
Backprop with Vectors



Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x !



Last time: Training details

- **One time setup:** activation functions, preprocessing, weight initialization, regularization, gradient checking
- **Training dynamics:** babysitting the learning process, parameter updates, data augmentation, hyperparameter optimization
- **Evaluation:** model ensembles

Image Classification: A core task in Computer Vision



This image by Nikita is licensed under [CC-BY 2.0](https://creativecommons.org/licenses/by/2.0/)

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



cat
dog
bird
deer
truck

Pixel space

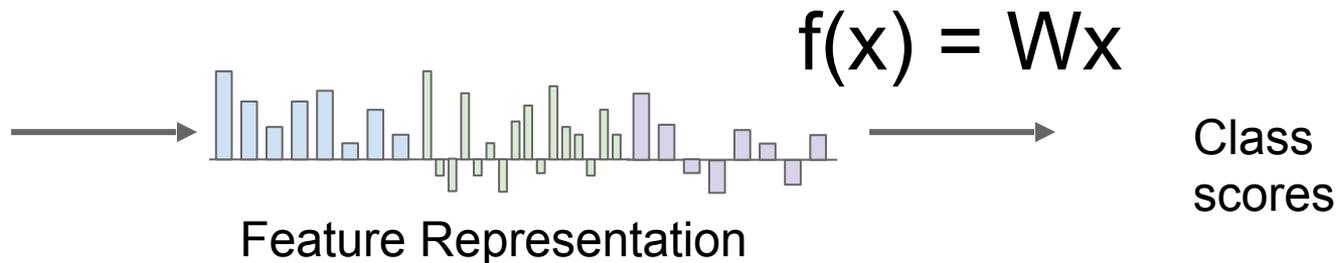


$$f(x) = Wx$$

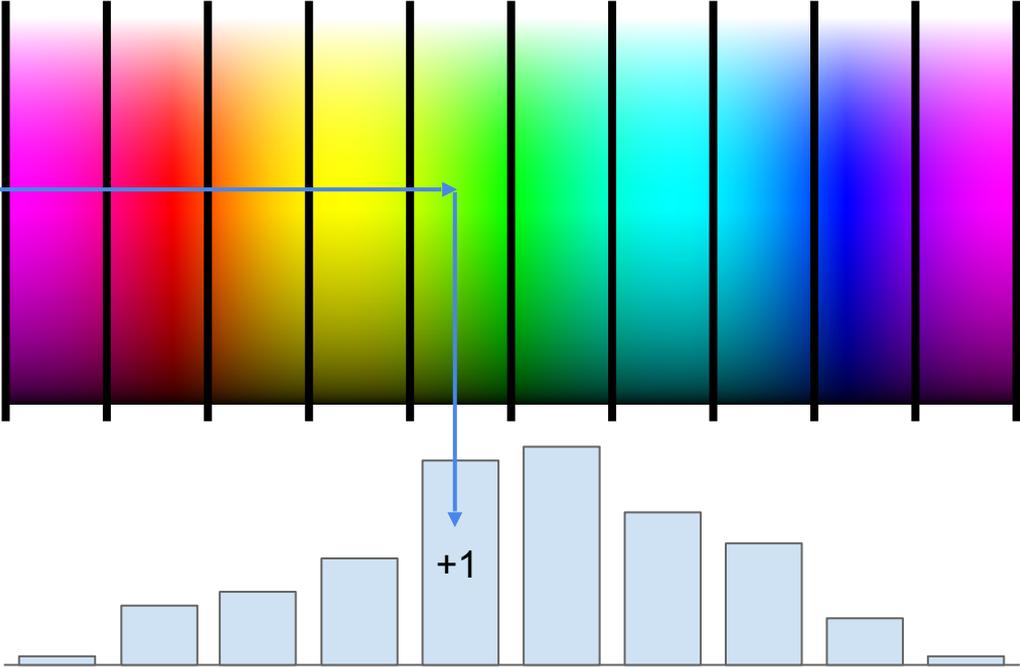
Class scores



Image features



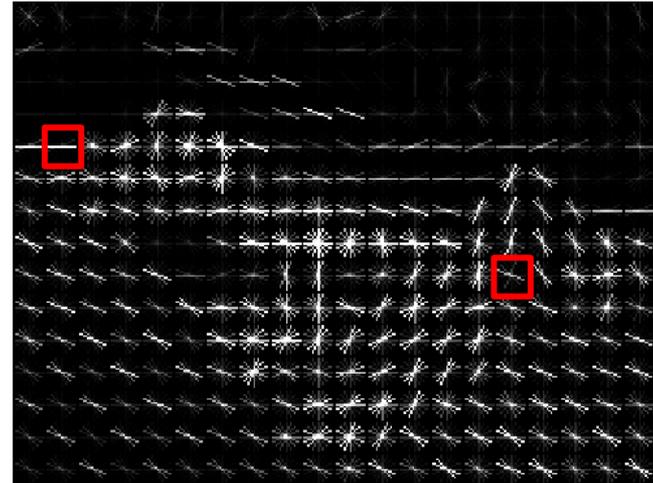
Example: Color Histogram



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
 $30 \times 40 \times 9 = 10,800$ numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Example: Bag of Words

Step 1: Build codebook



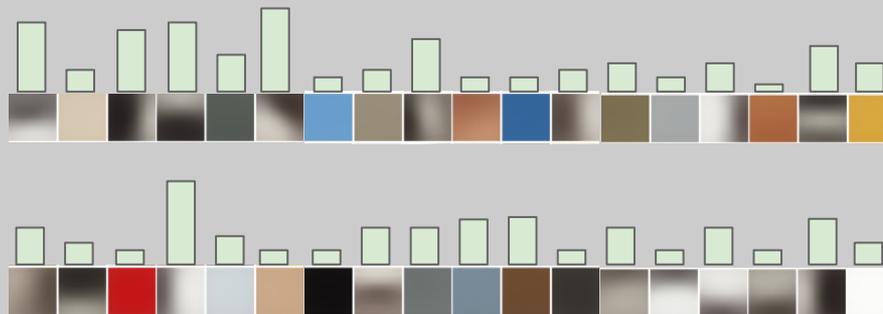
Extract random patches



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

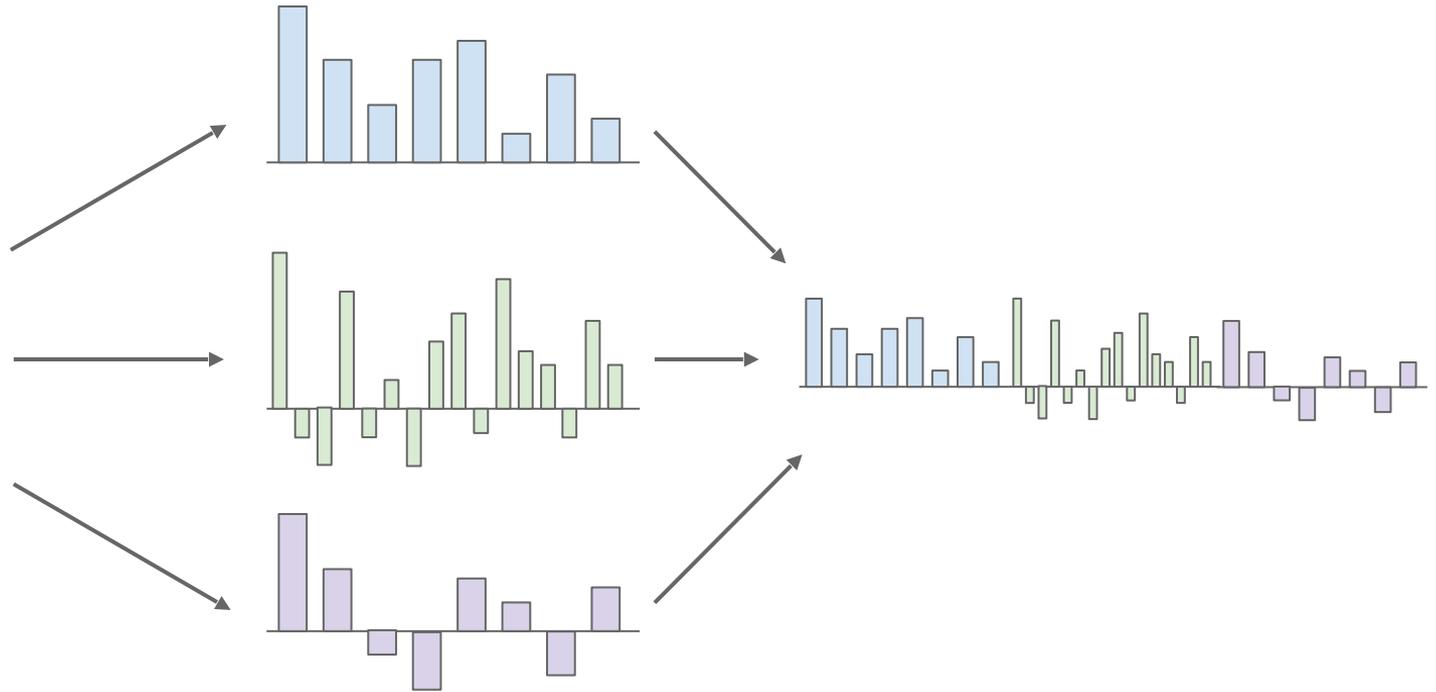
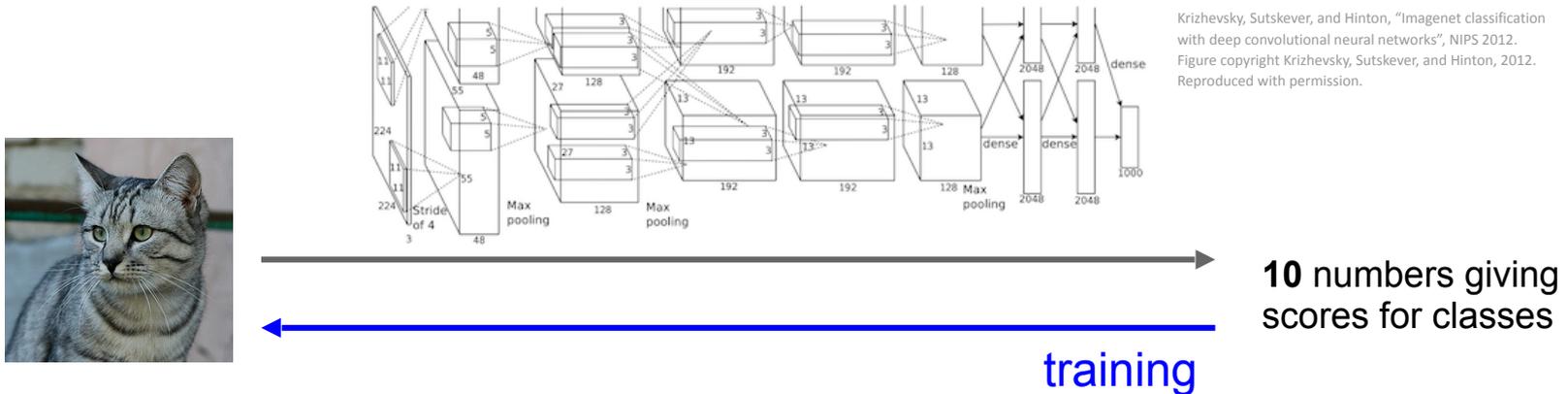
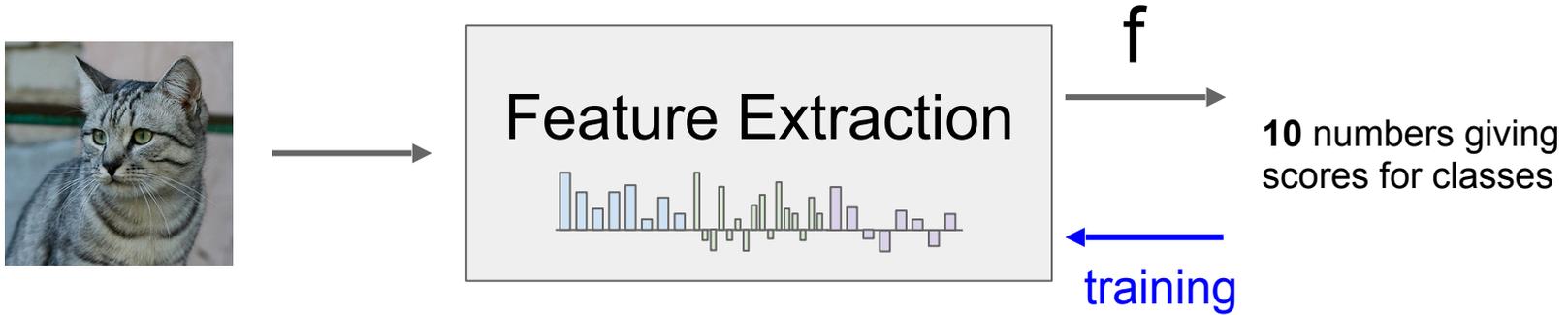


Image features vs. ConvNets



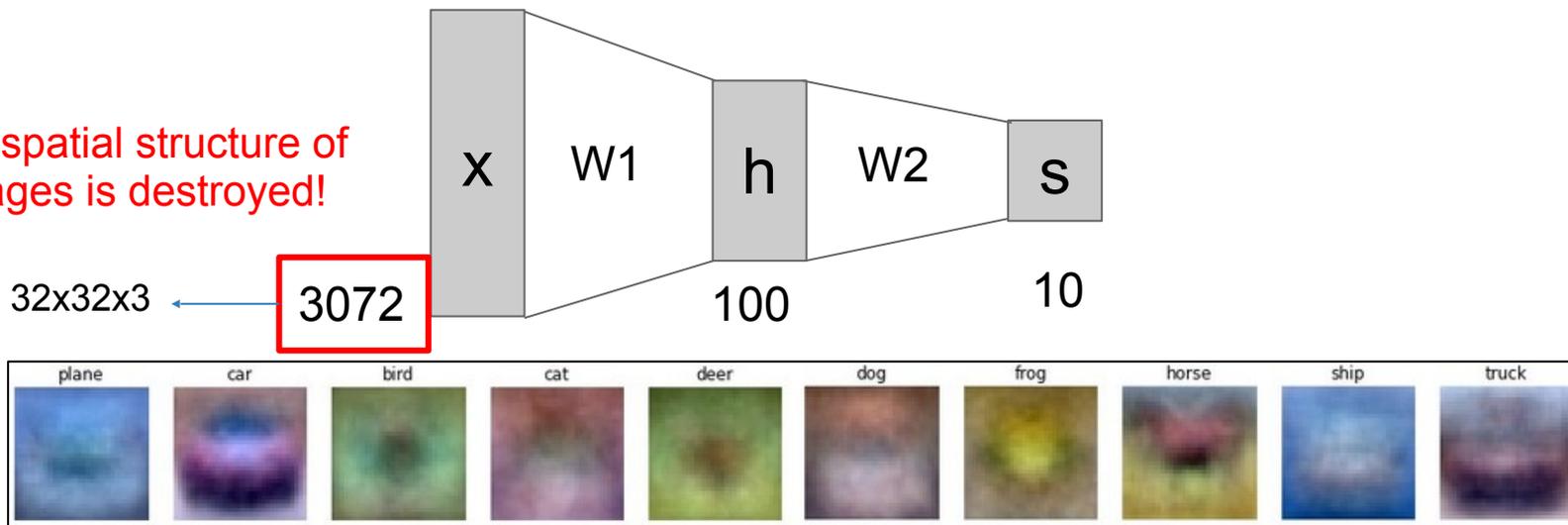
Linear score function:

$$f = Wx$$

2-layer Neural Network

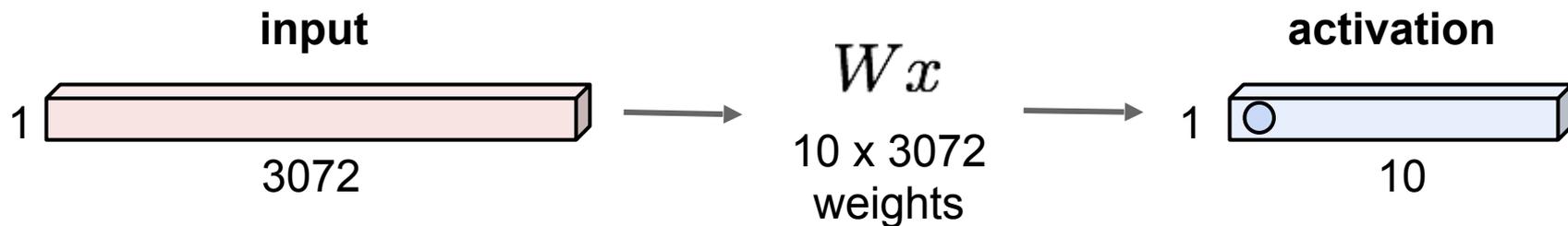
$$f = W_2 \max(0, W_1 x)$$

The spatial structure of images is destroyed!



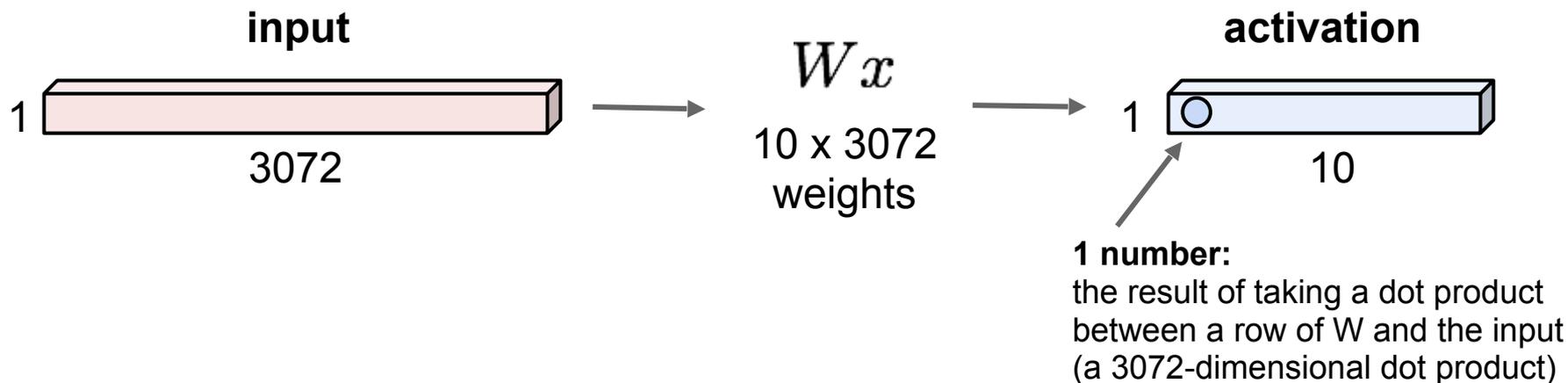
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



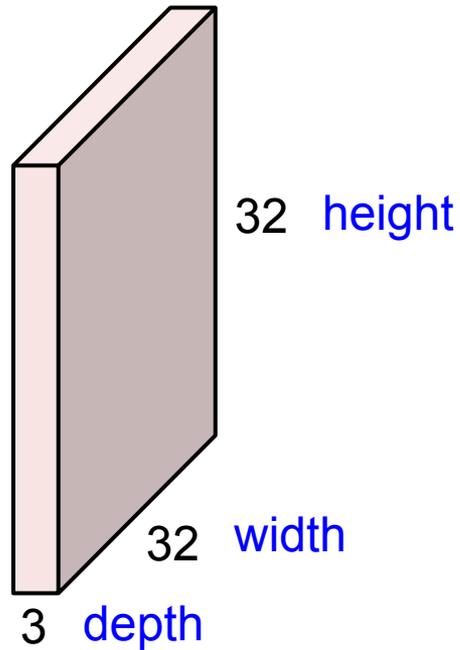
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



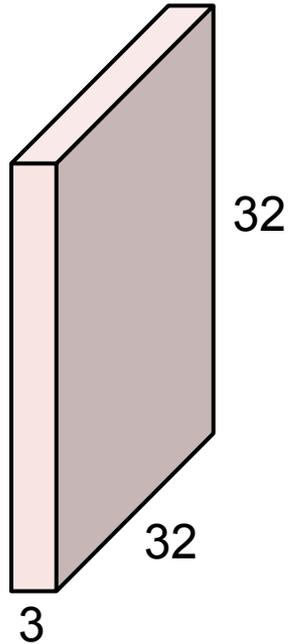
Convolution Layer

32x32x3 image -> preserve spatial structure

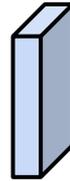


Convolution Layer

32x32x3 image



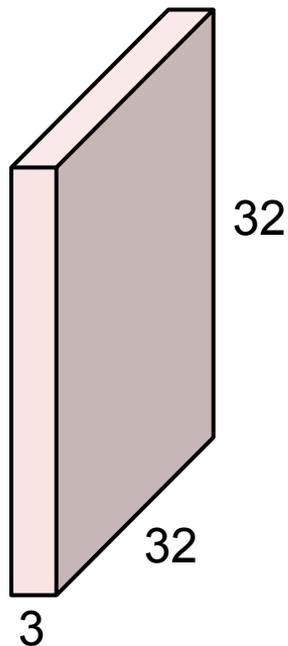
5x5x3 filter



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

Convolution Layer

32x32x3 image



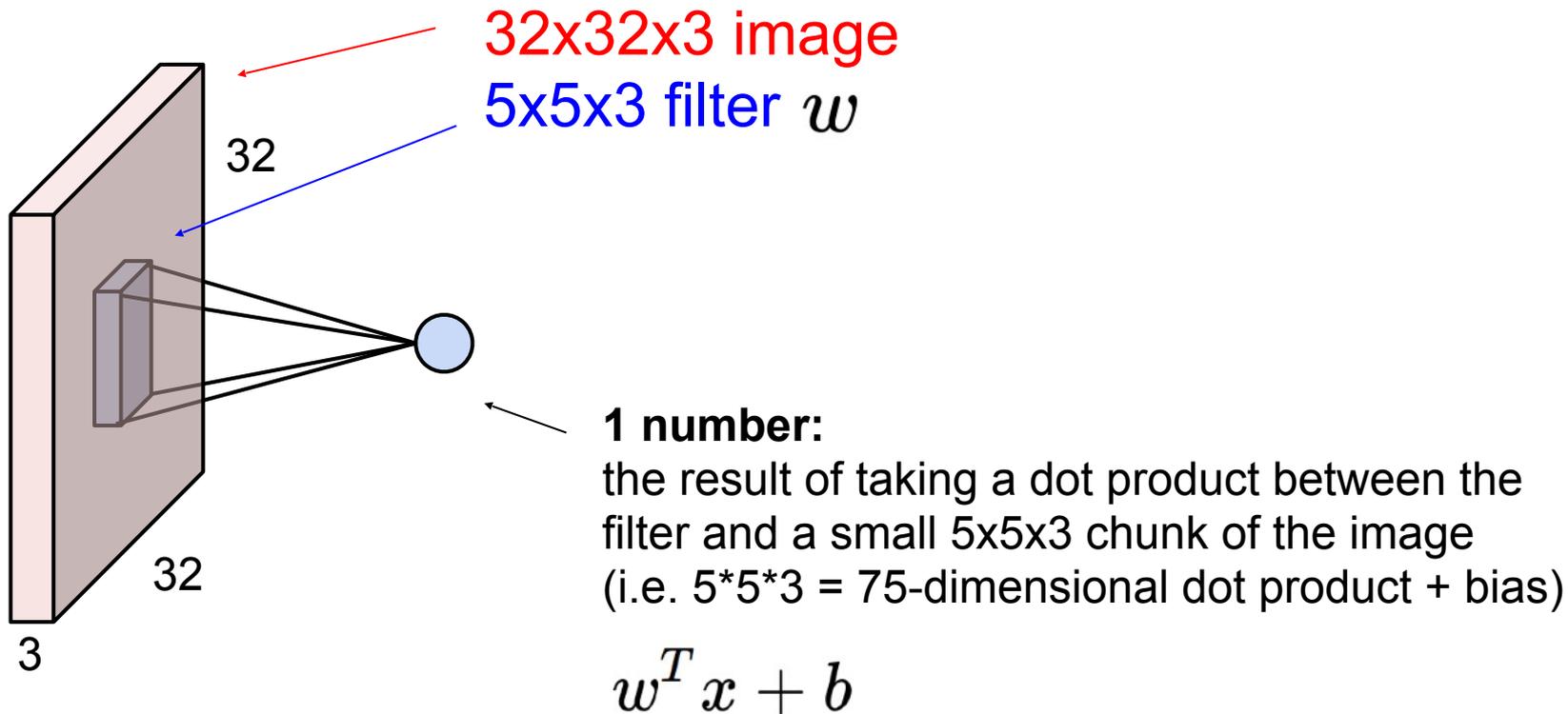
Filters always extend the full depth of the input volume

5x5x3 filter

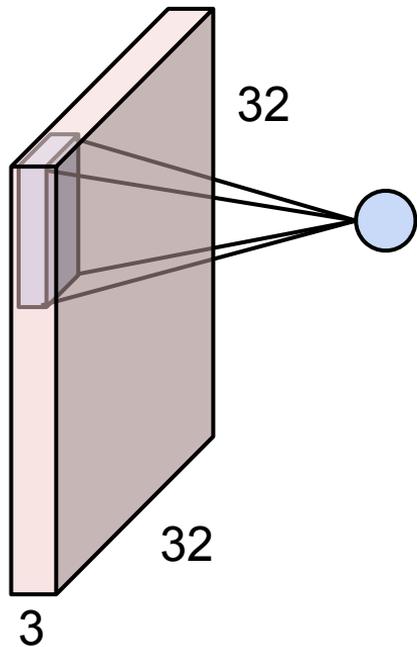


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

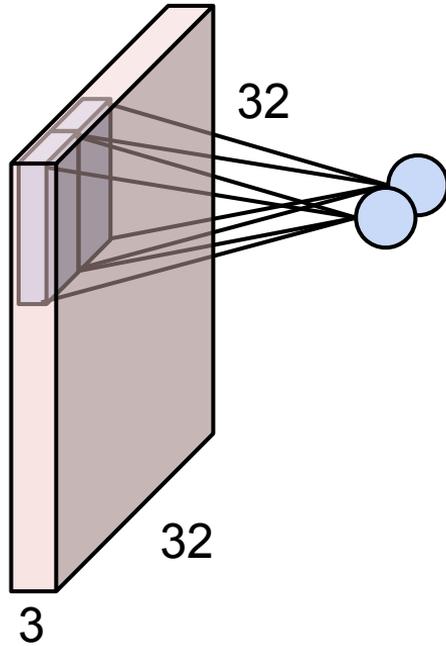
Convolution Layer



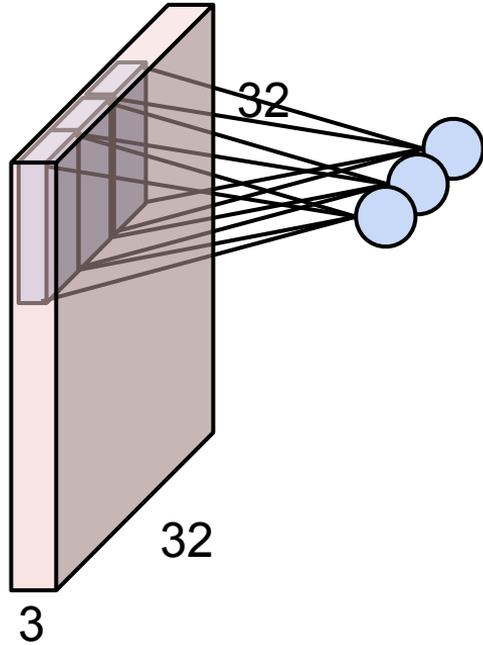
Convolution Layer



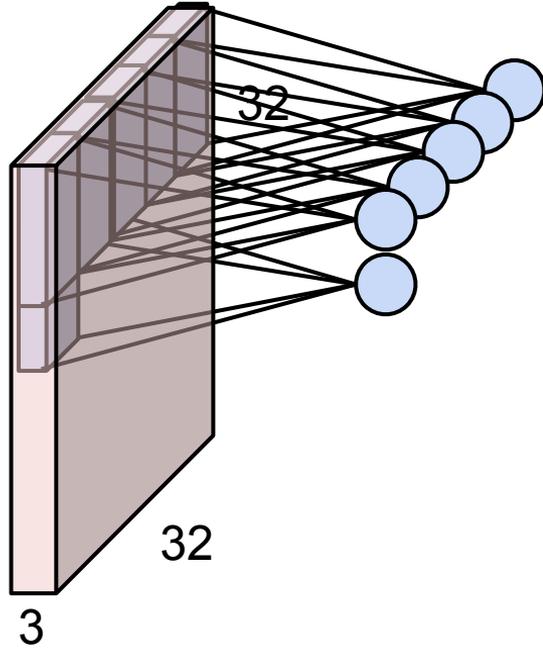
Convolution Layer



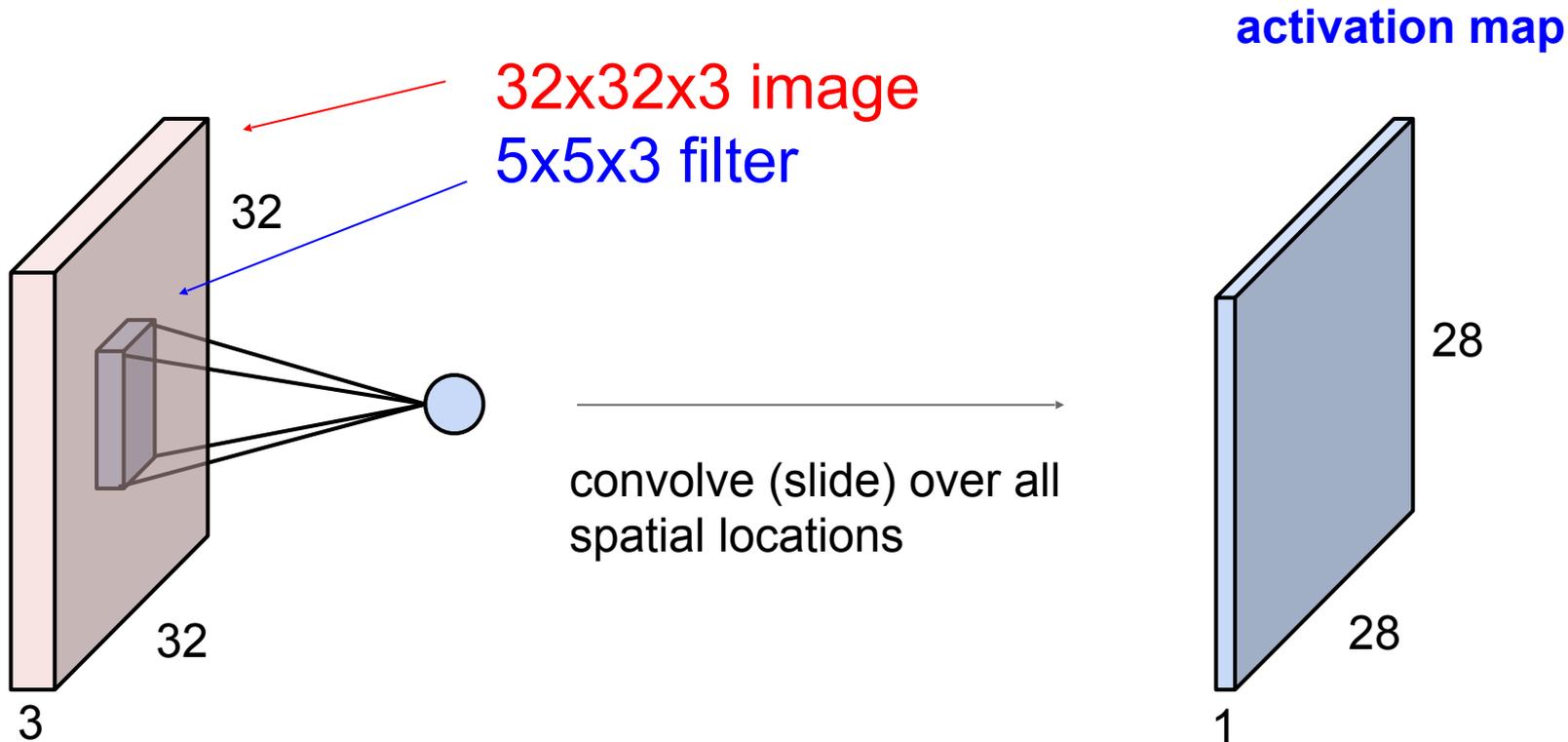
Convolution Layer



Convolution Layer

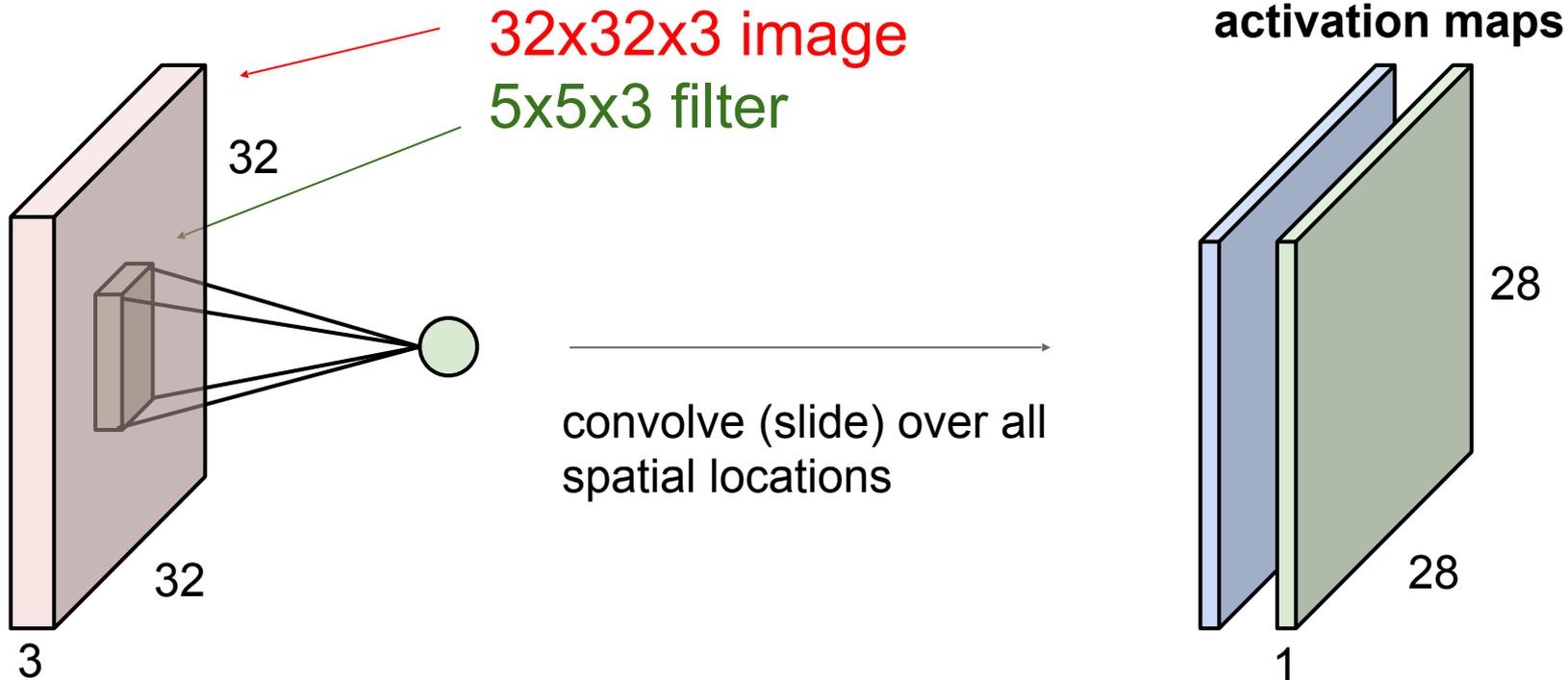


Convolution Layer



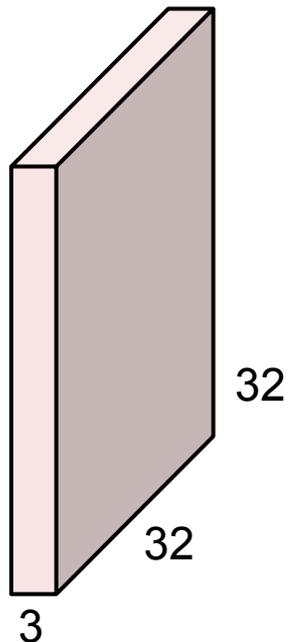
Convolution Layer

consider a second, **green** filter

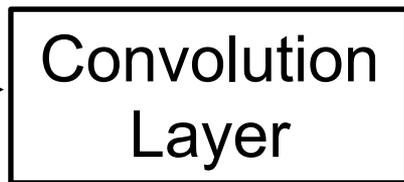


Convolution Layer

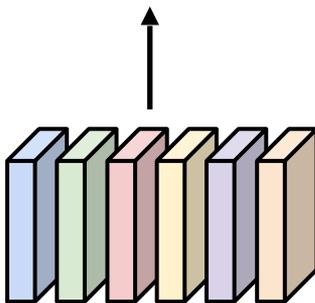
3x32x32 image



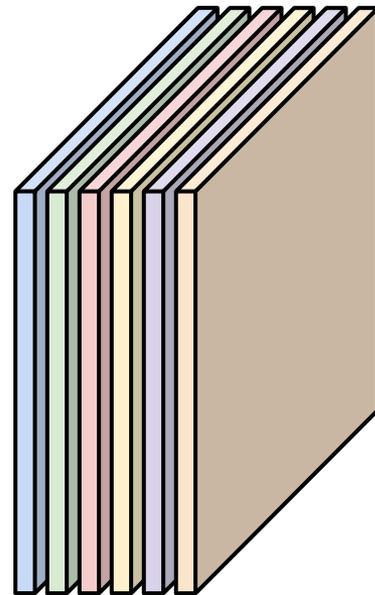
Consider 6 filters,
each 3x5x5



6x3x5x5
filters



6 activation maps,
each 1x28x28



Stack activations to get a
6x28x28 output image!

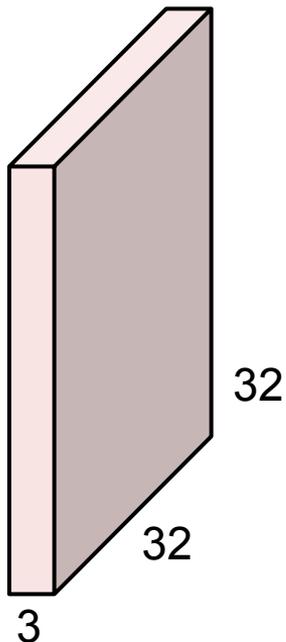
Slide inspiration: Justin Johnson

Subhansu Maji and TAs

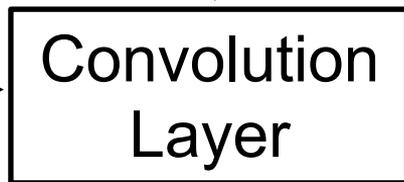
Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

Convolution Layer

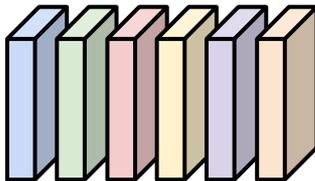
3x32x32 image



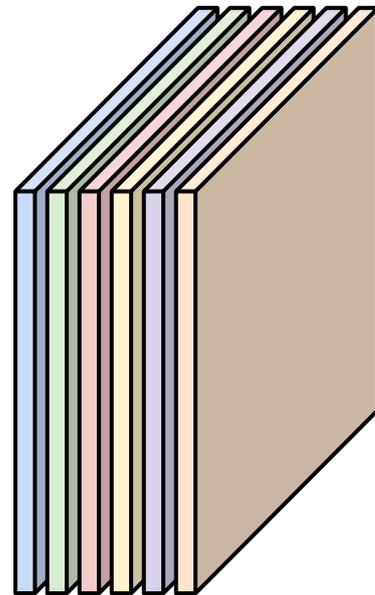
Also 6-dim bias vector:



6x3x5x5 filters



6 activation maps,
each 1x28x28



Stack activations to get a
6x28x28 output image!

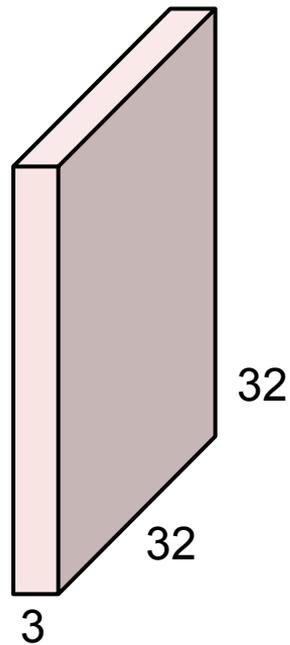
Slide inspiration: Justin Johnson

Subhansu Maji and TAs

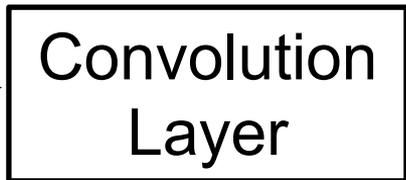
Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

Convolution Layer

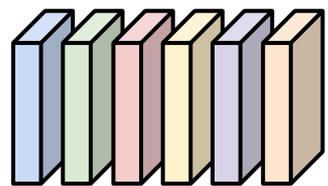
3x32x32 image



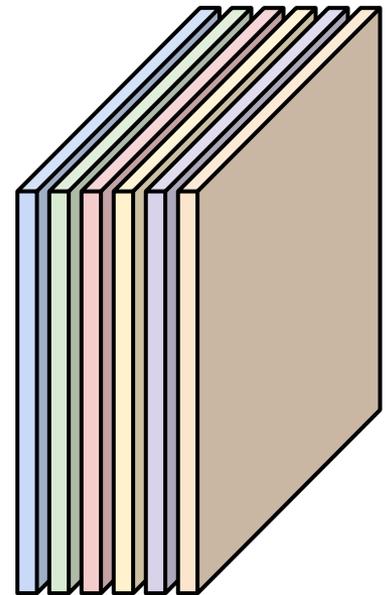
Also 6-dim bias vector:



6x3x5x5 filters



28x28 grid, at each point a 6-dim vector

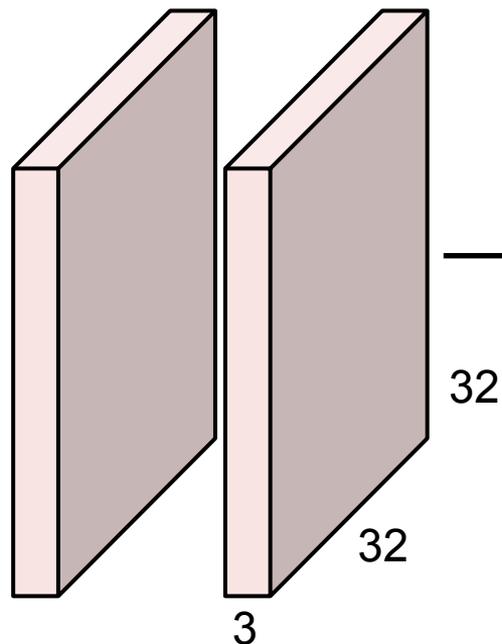


Stack activations to get a 6x28x28 output image!

Slide inspiration: Justin Johnson

Convolution Layer

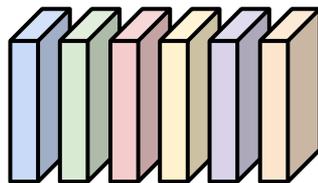
2x3x32x32
Batch of images



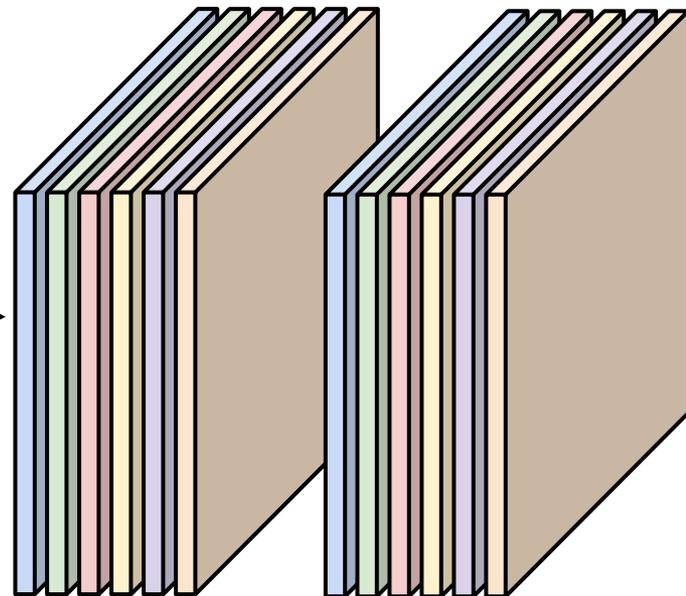
Also 6-dim bias vector:



6x3x5x5
filters



2x6x28x28
Batch of outputs



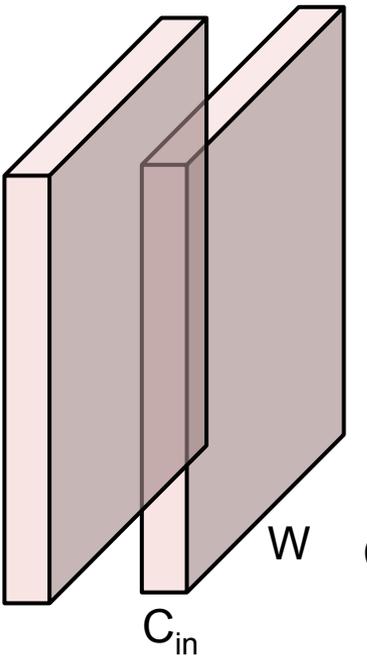
Slide inspiration: Justin Johnson

Subhransu Maji and TAs

Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

Convolution Layer

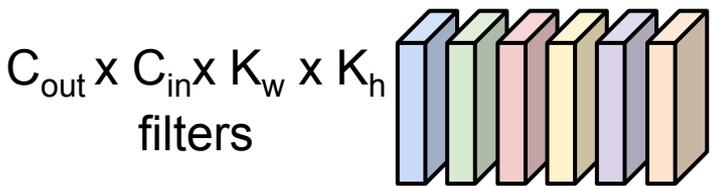
$N \times C_{in} \times H \times W$
Batch of images



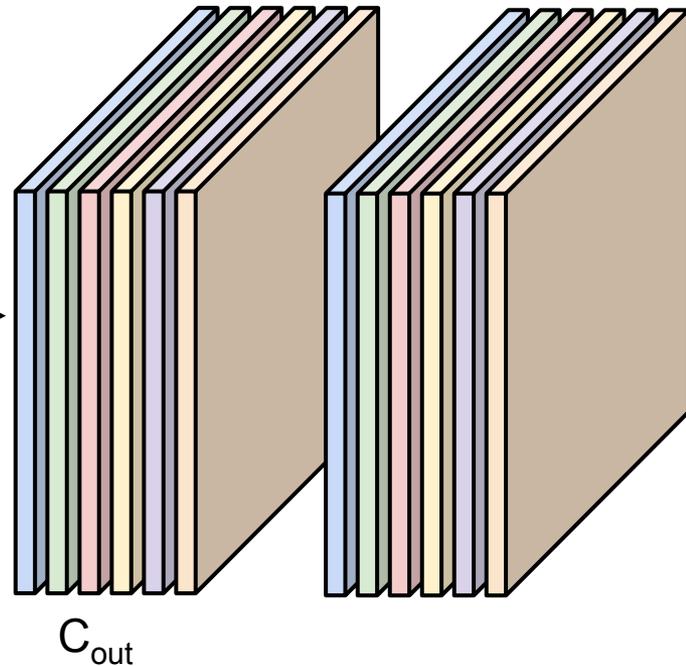
Also C_{out} -dim bias vector:



Convolution Layer



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

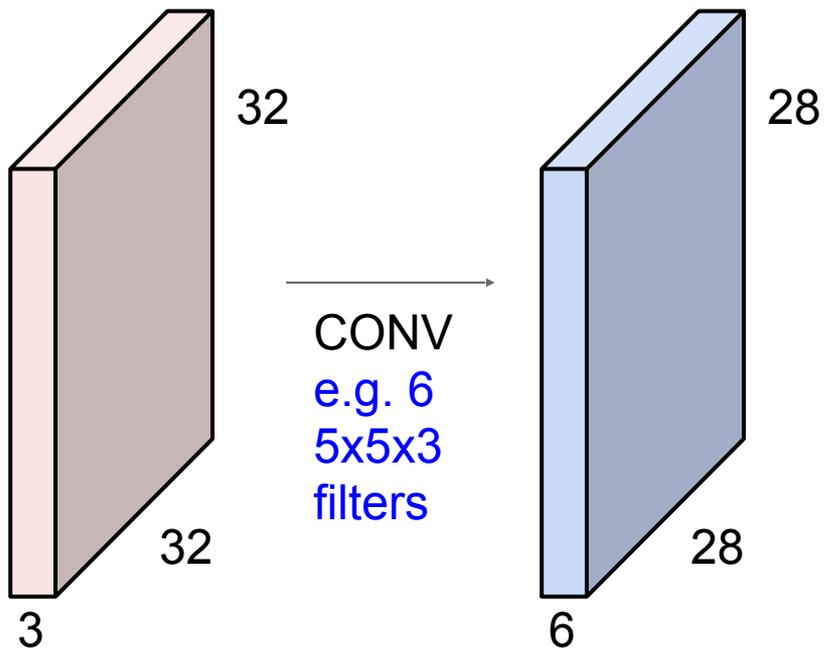


Slide inspiration: Justin Johnson

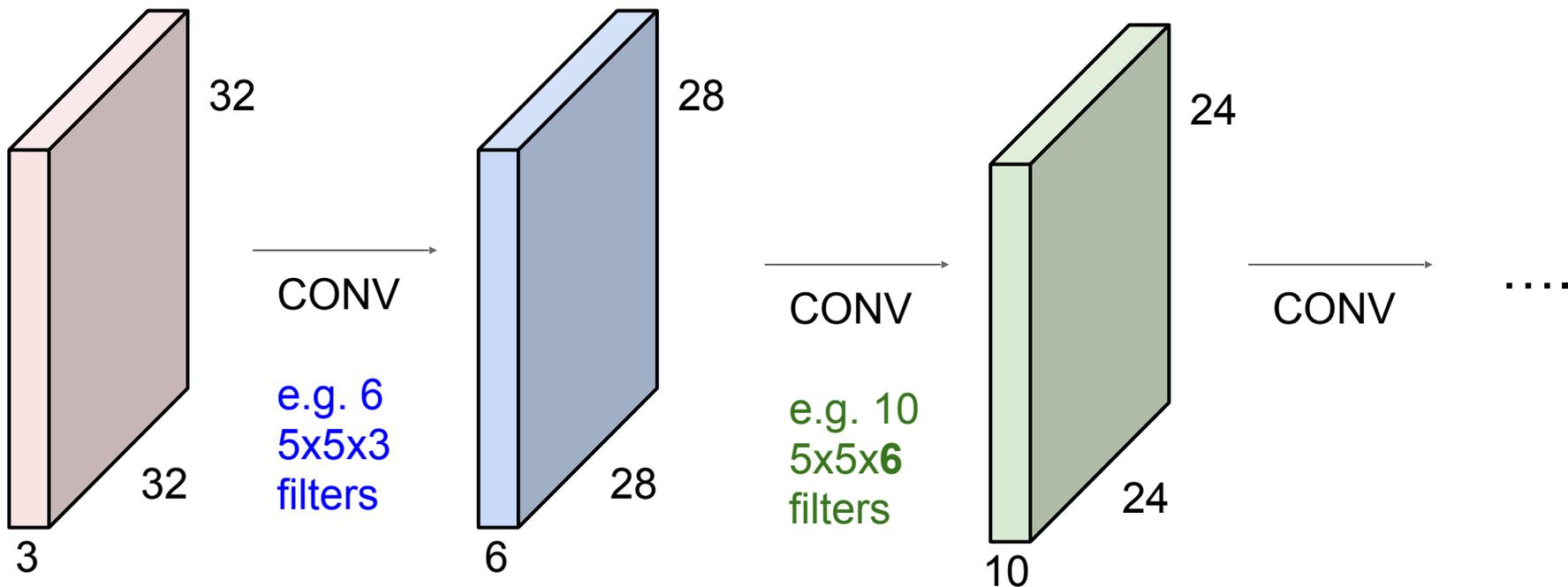
Subhransu Maji and TAs

Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

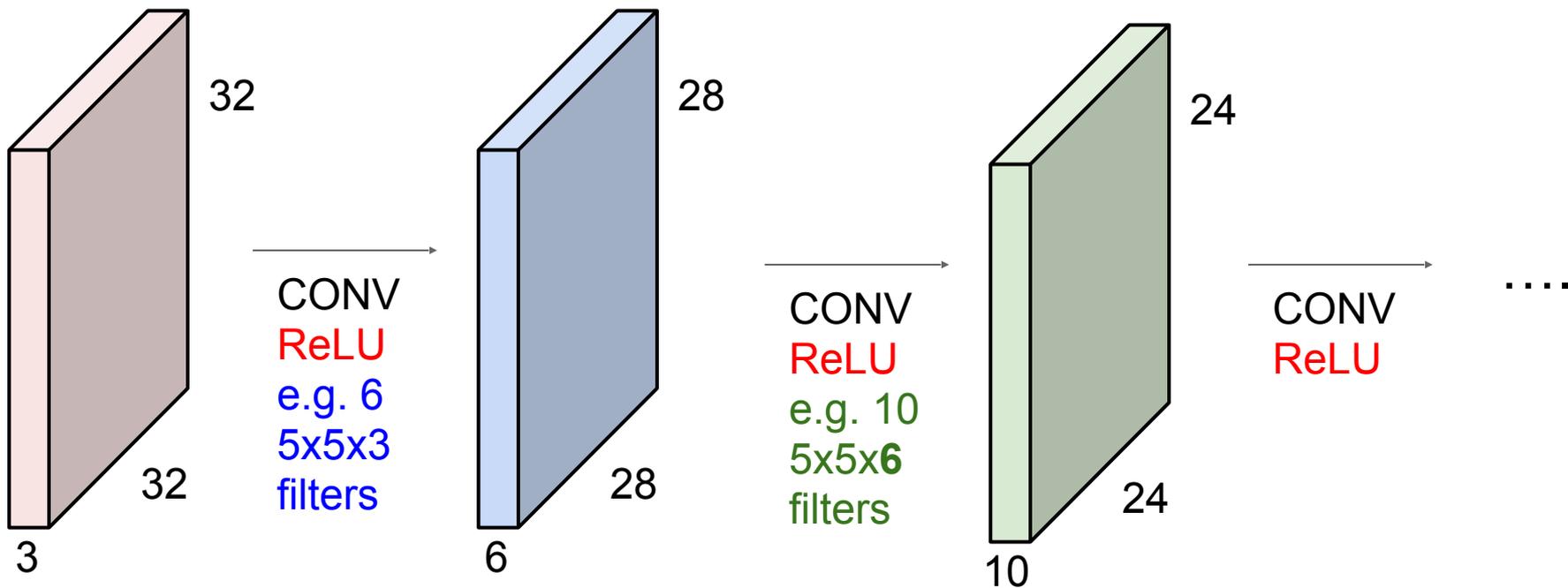
Preview: ConvNet is a sequence of Convolution Layers

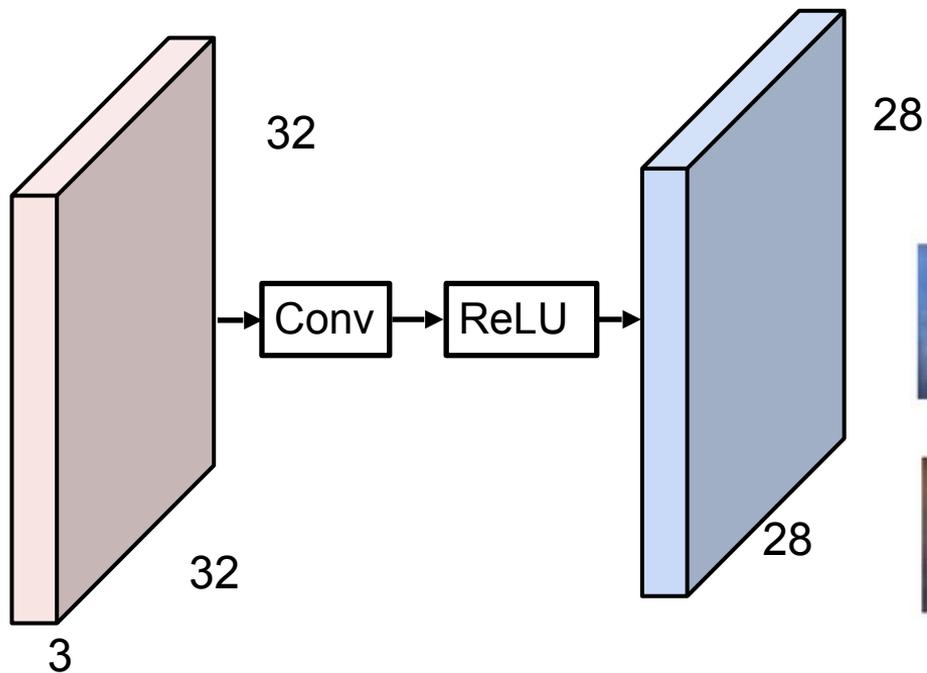


Preview: ConvNet is a sequence of Convolution Layers



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

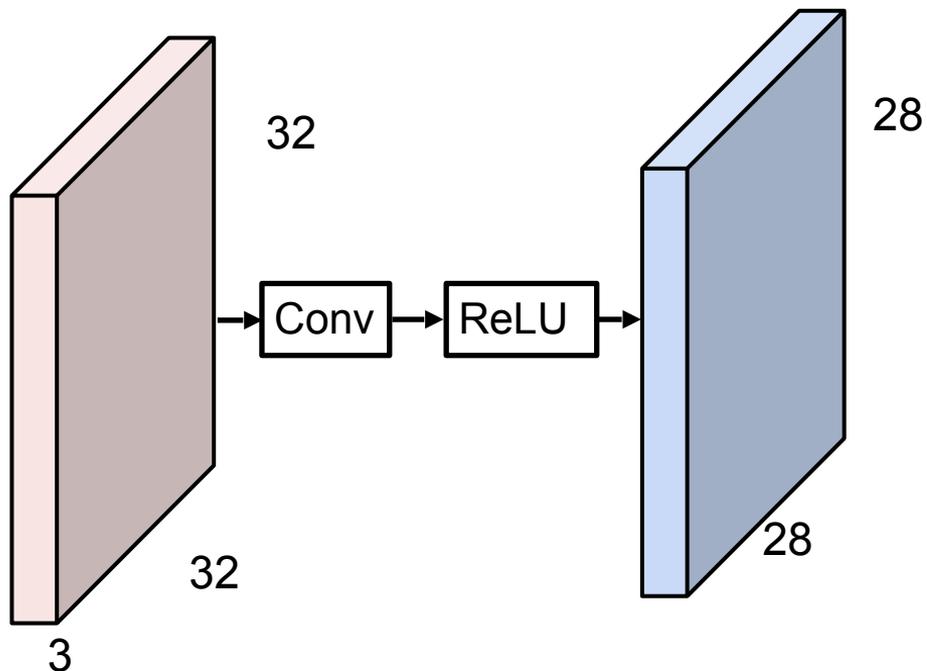




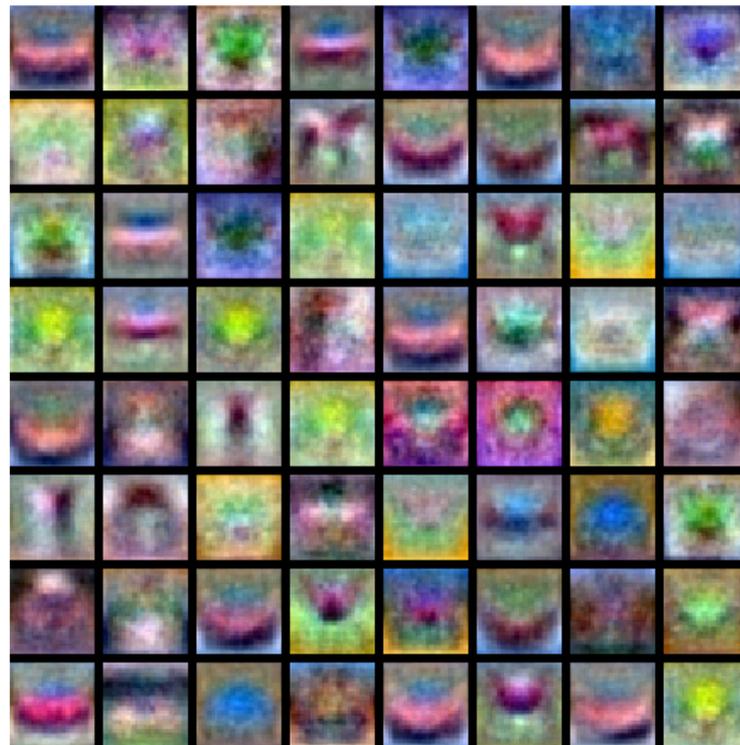
Linear classifier: One template per class

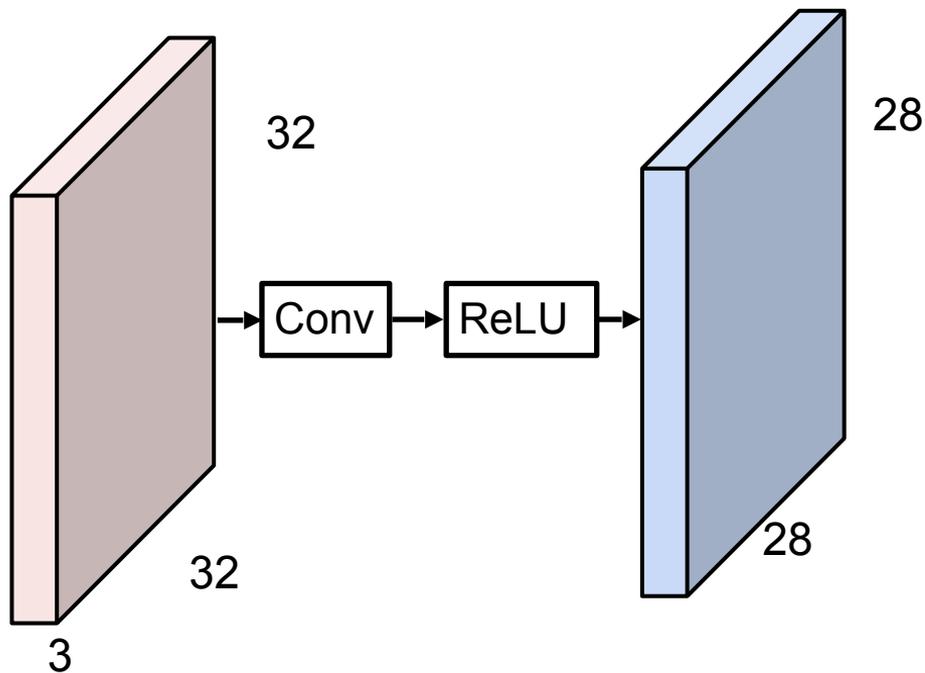


Preview: What do convolutional filters learn?

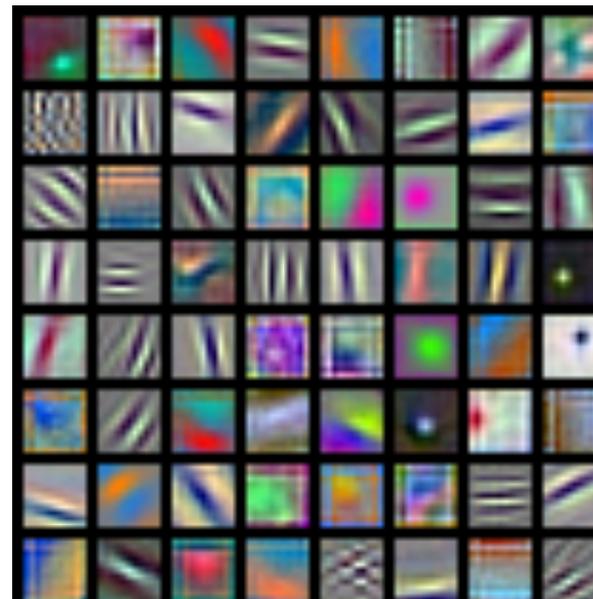


MLP: Bank of whole-image templates





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



AlexNet: 64 filters, each 3x11x11



one filter =>
one activation map

example 5x5 filters
(32 total)

Activations:

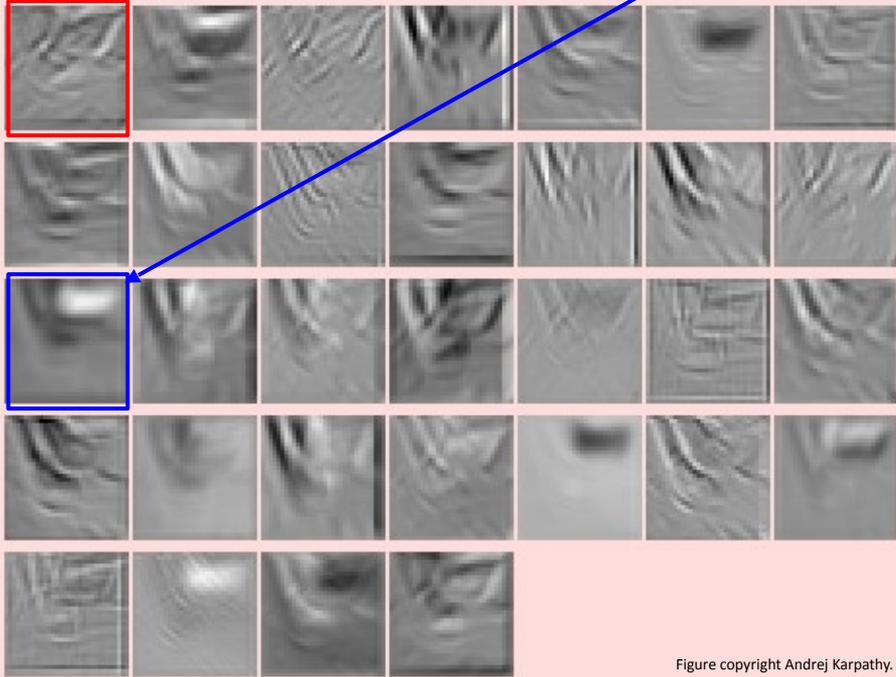


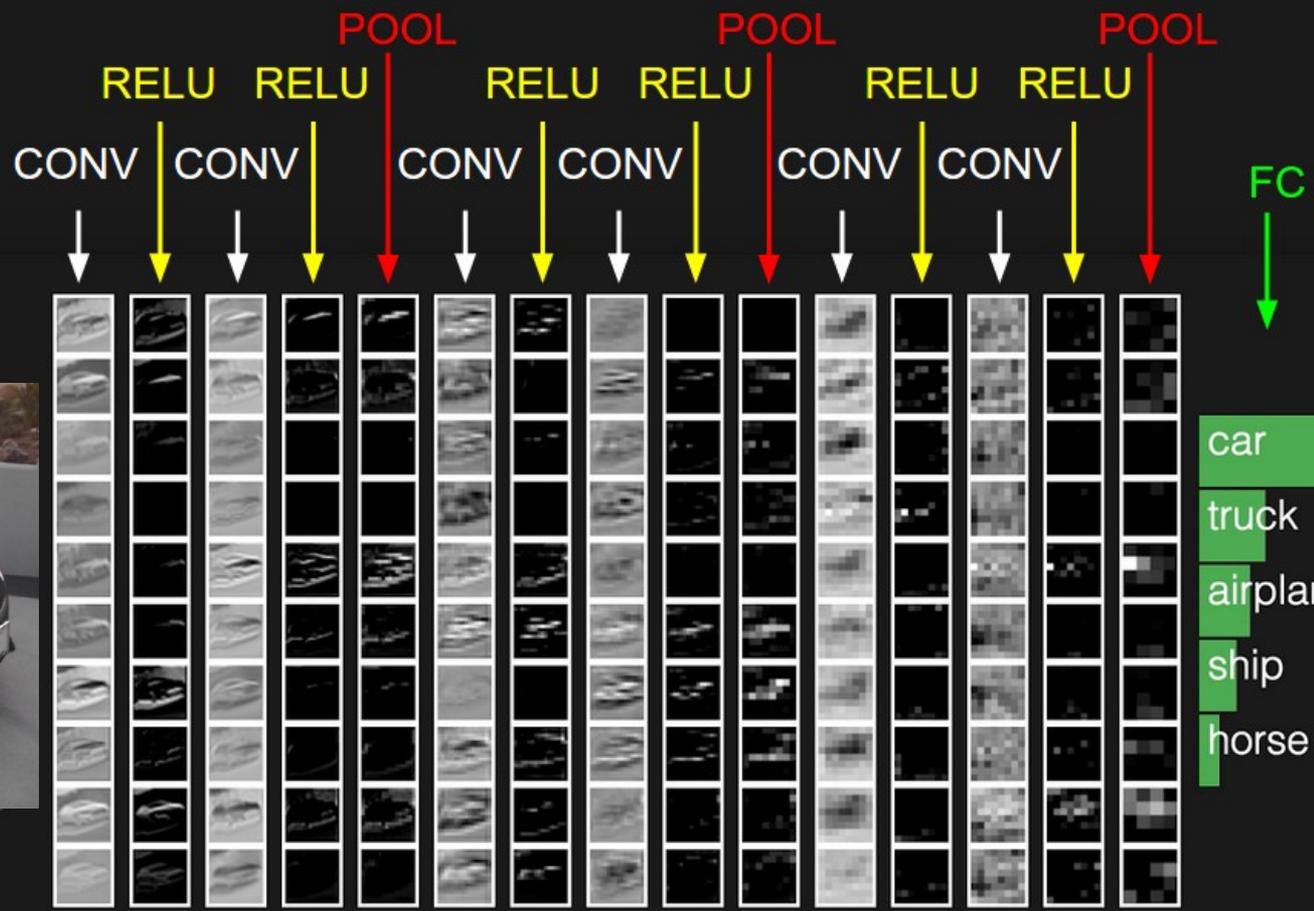
Figure copyright Andrej Karpathy.

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

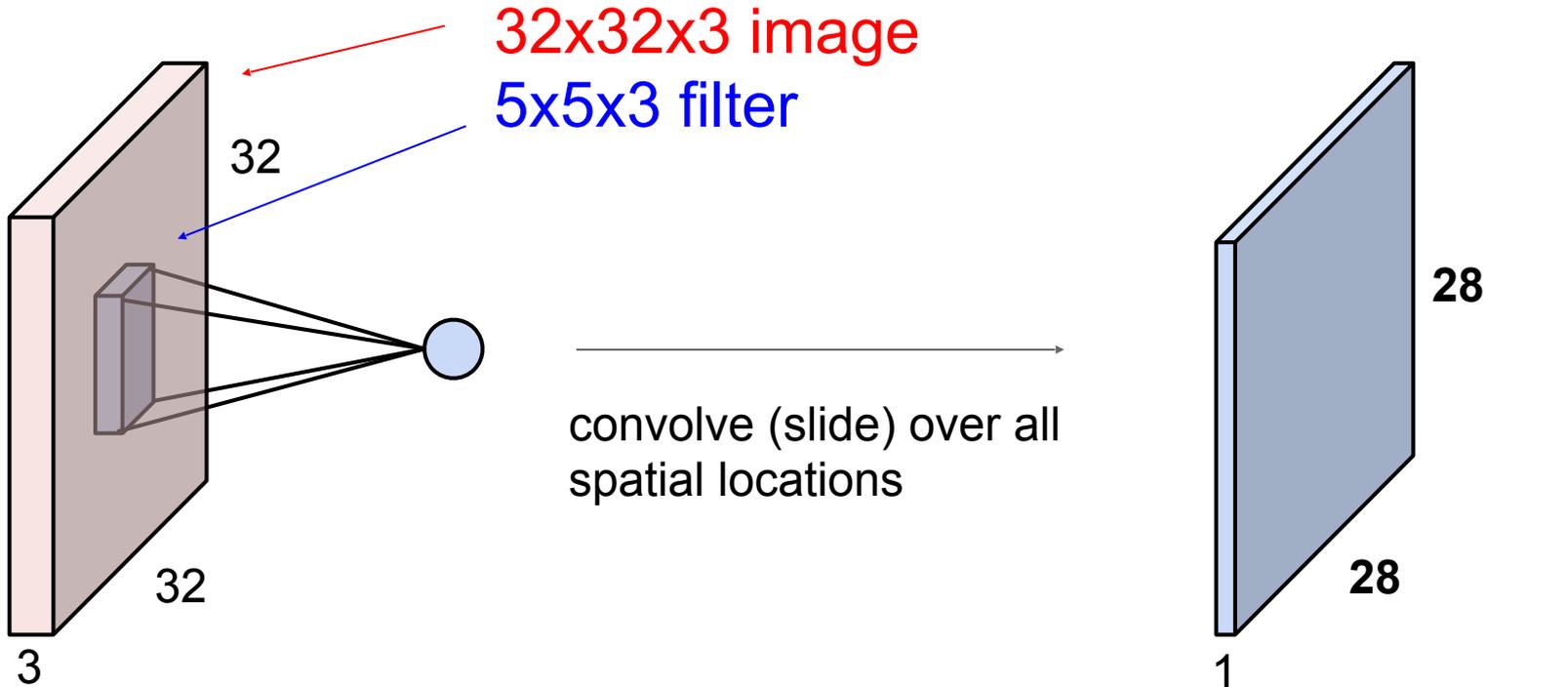
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

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

preview:

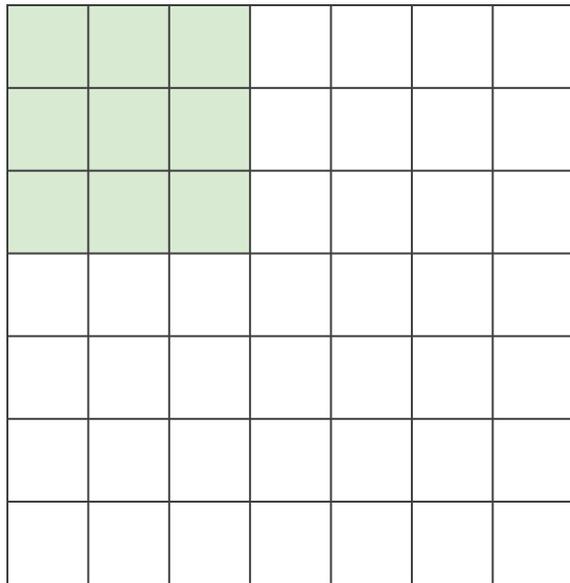


A closer look at spatial dimensions:



A closer look at spatial dimensions:

7

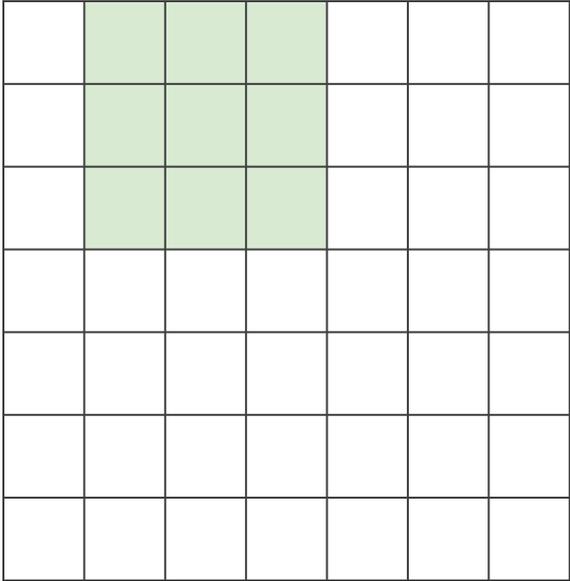


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

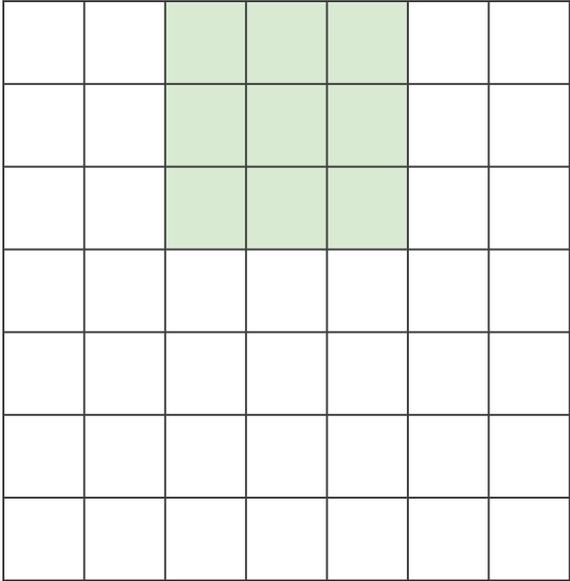


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

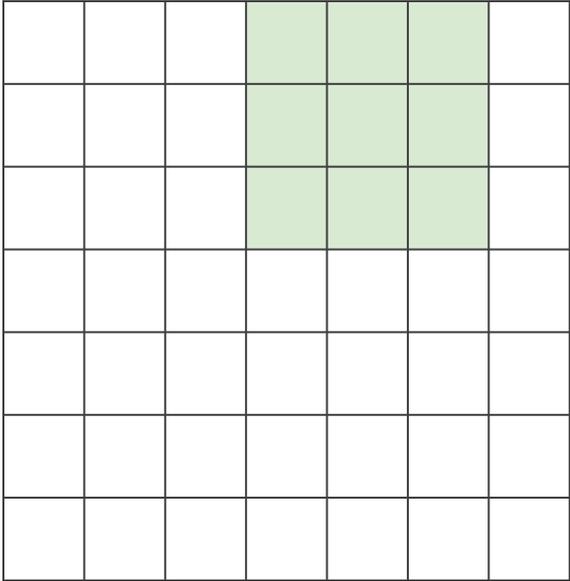


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

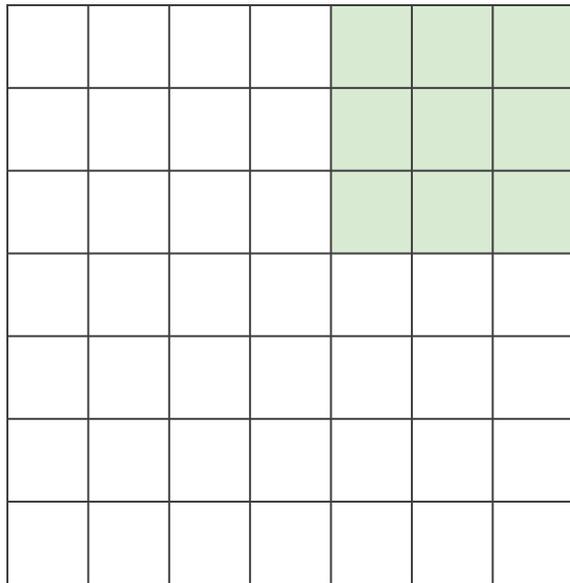


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7



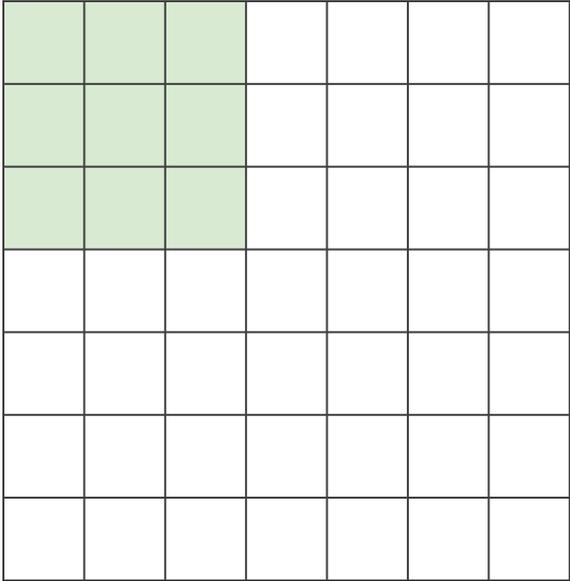
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

7

A closer look at spatial dimensions:

7

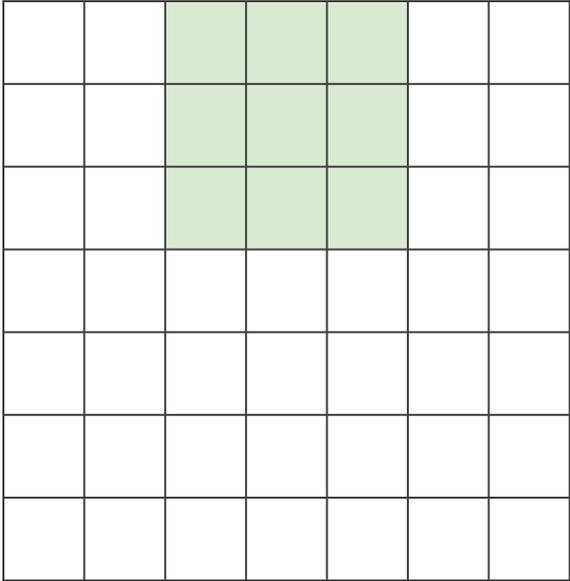


7

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

A closer look at spatial dimensions:

7

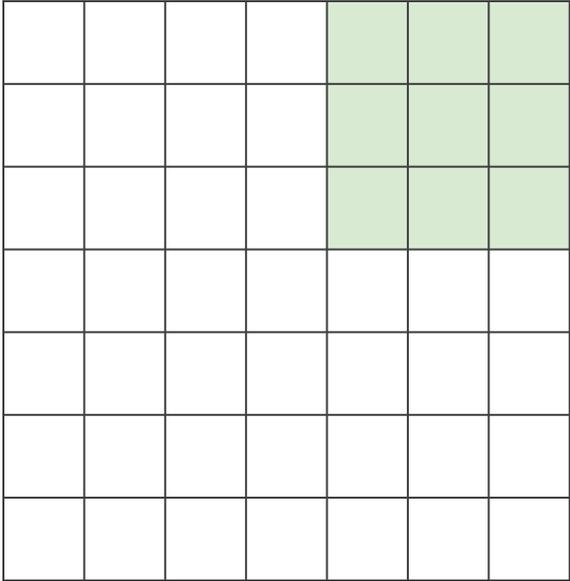


7

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

A closer look at spatial dimensions:

7

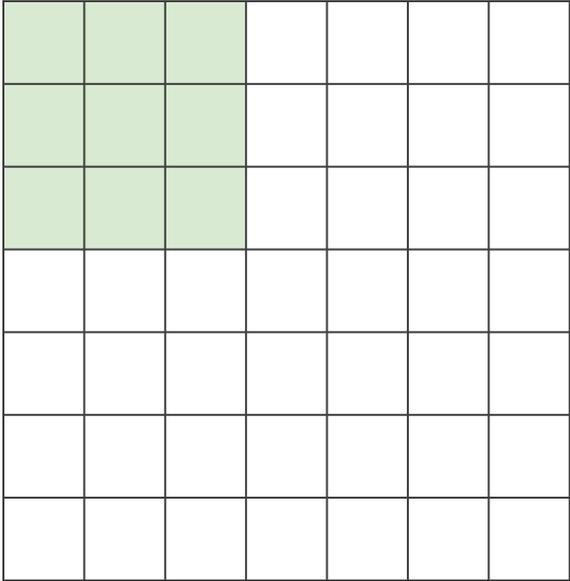


7

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

A closer look at spatial dimensions:

7

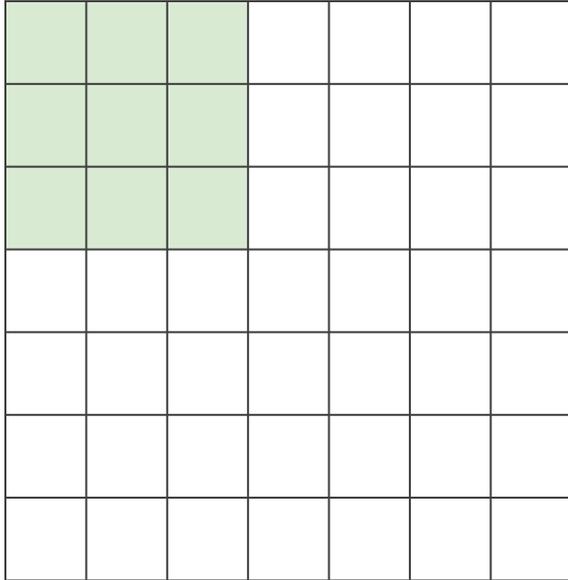


7

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

A closer look at spatial dimensions:

7

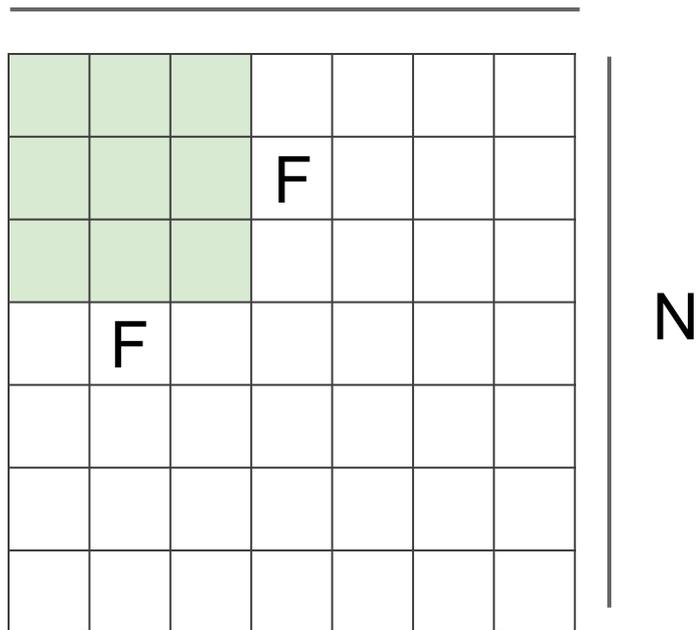


7

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

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

N



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

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

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

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

stride 3 $\Rightarrow (7 - 3) / 3 + 1 = 2.33 \text{ :}\backslash$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

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

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

(recall:)

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

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

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

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

7x7 output!

(recall:)

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

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

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

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

7x7 output!

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

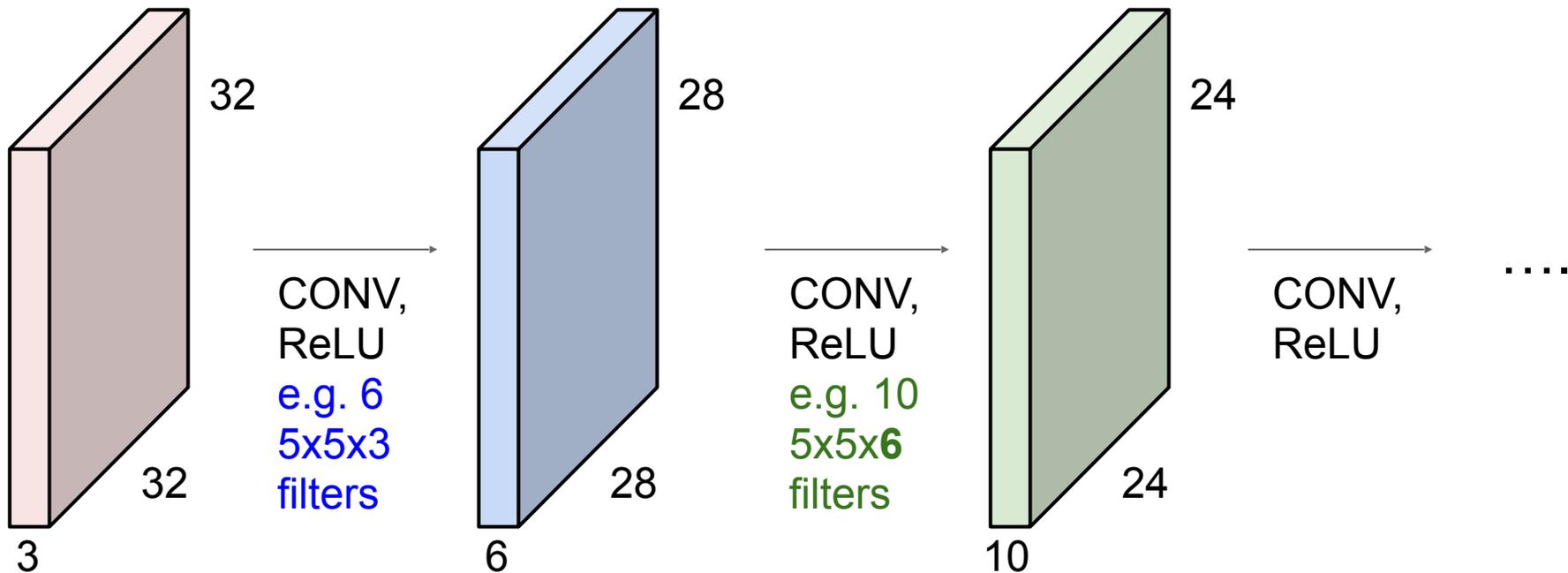
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Remember back to...

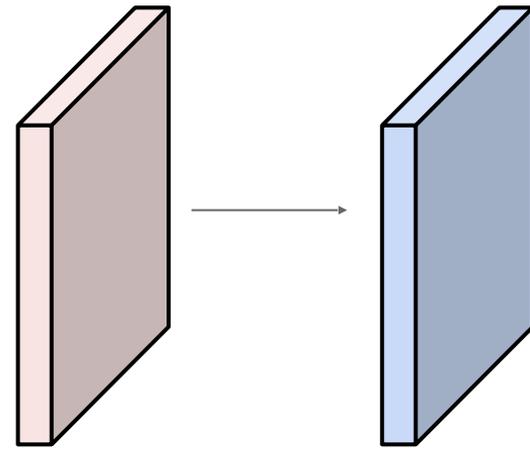
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 → 28 → 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time:

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

Output volume size: ?



Examples time:

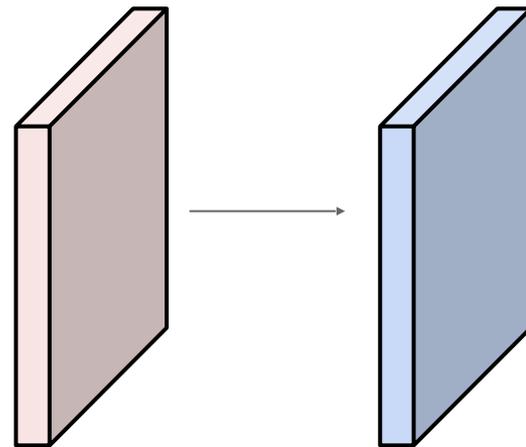
Input volume: **32x32x3**

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

Output volume size:

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

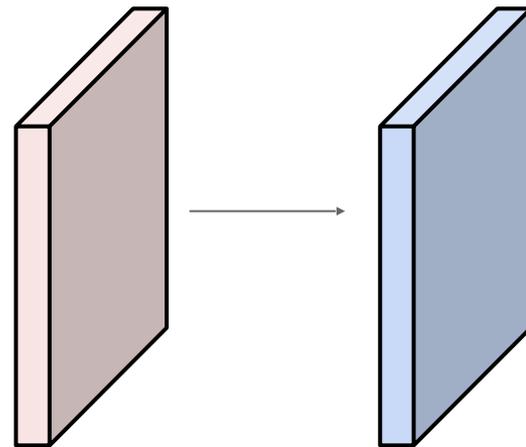
32x32x10



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

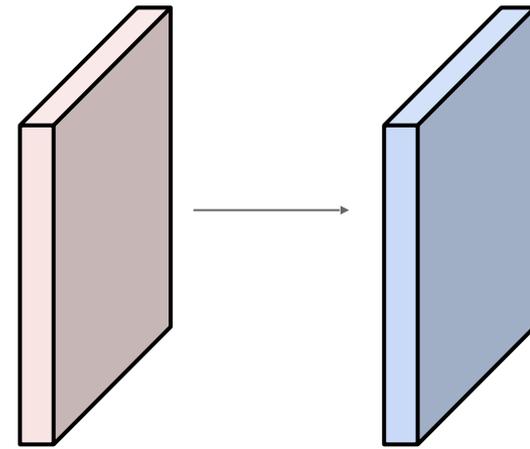


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

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



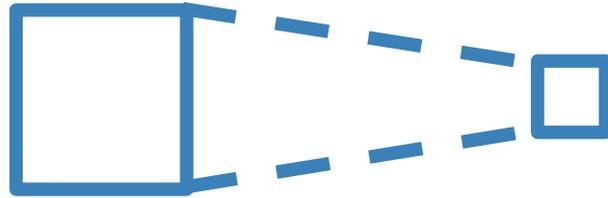
Number of parameters in this layer?

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

(+1 for bias)

$\Rightarrow 76*10 = 760$

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

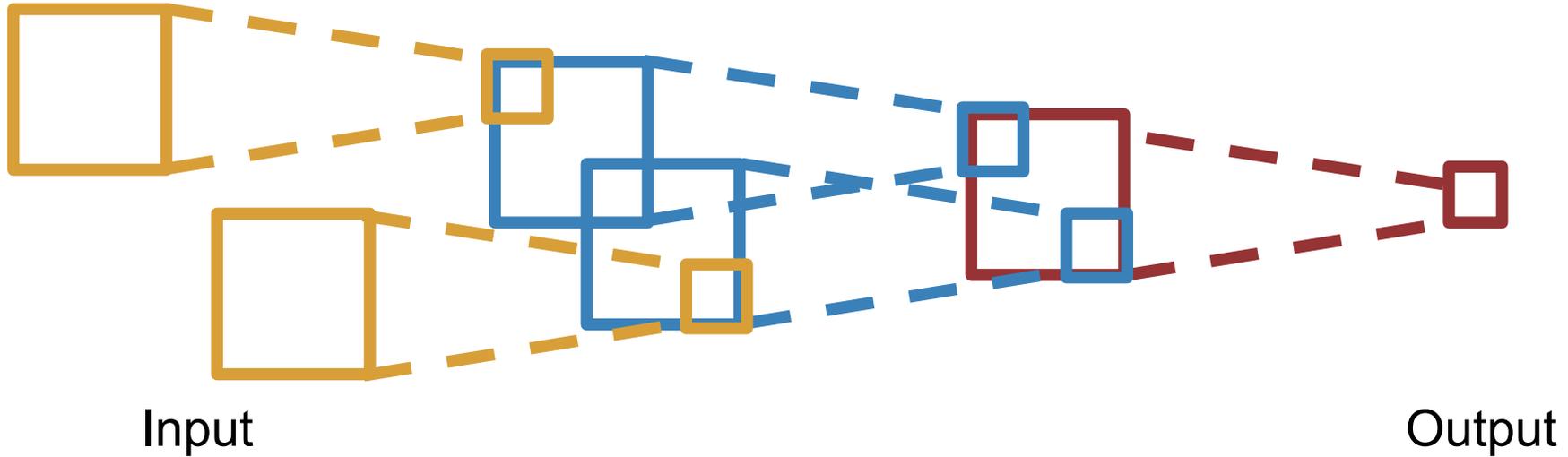


Input

Output

Slide inspiration: Justin Johnson

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

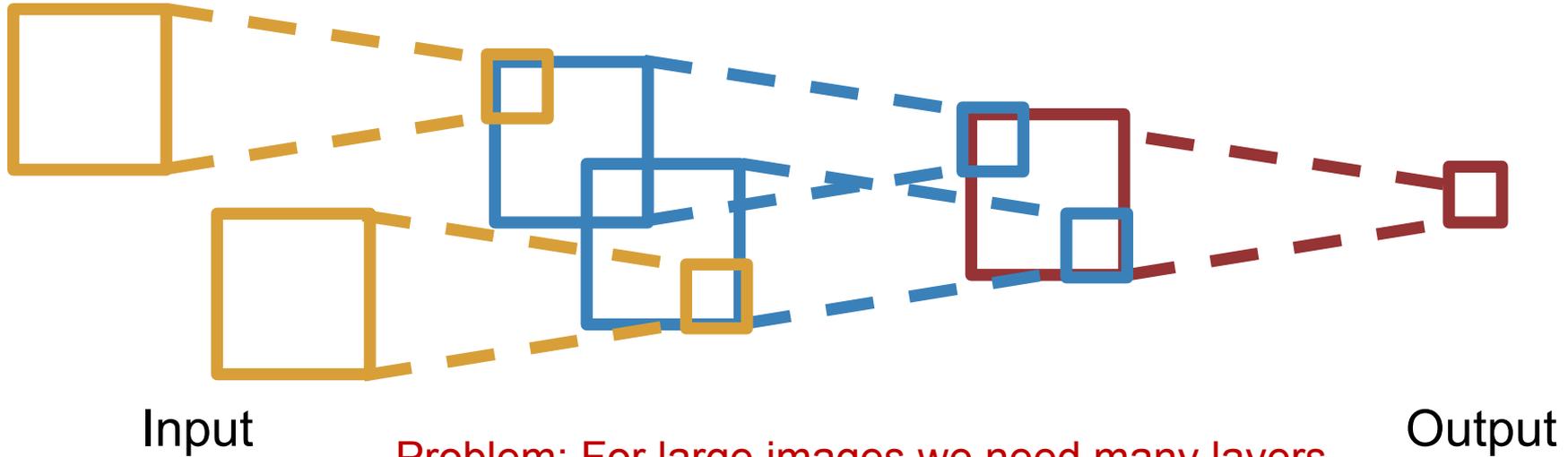


Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

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 $1 + L * (K - 1)$

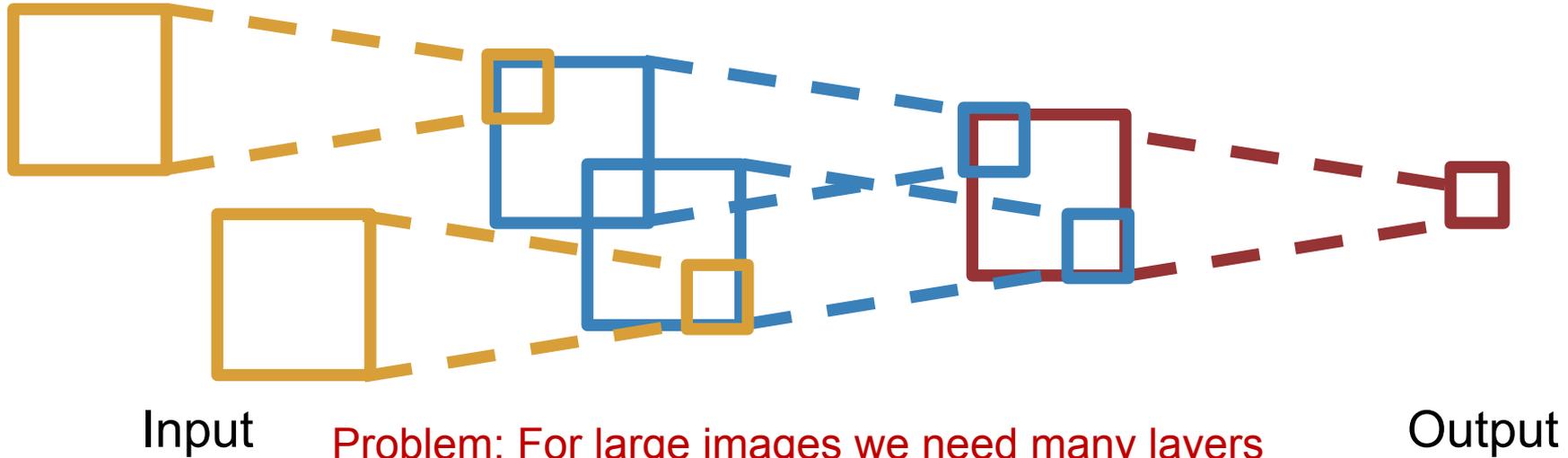


Problem: For large images we need many layers for each output to “see” the whole image

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 $1 + L * (K - 1)$

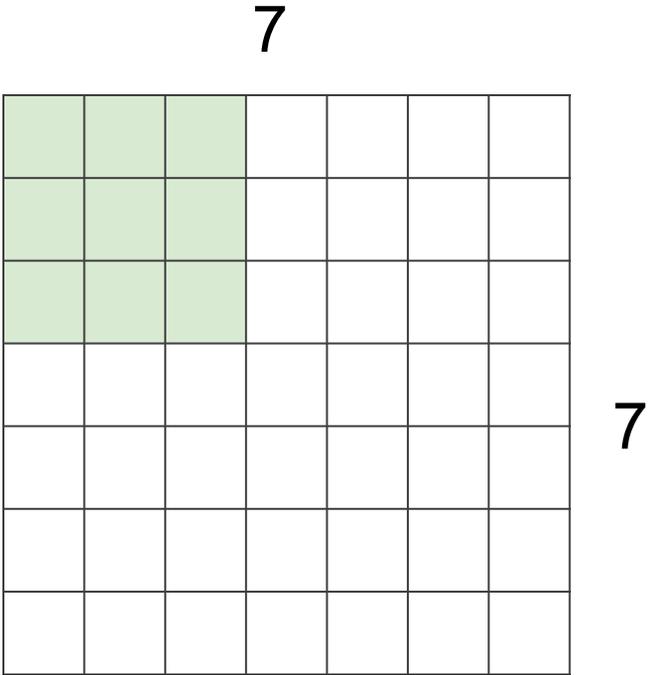


Problem: For large images we need many layers for each output to “see” the whole image

Solution: Downsample inside the network

Slide inspiration: Justin Johnson

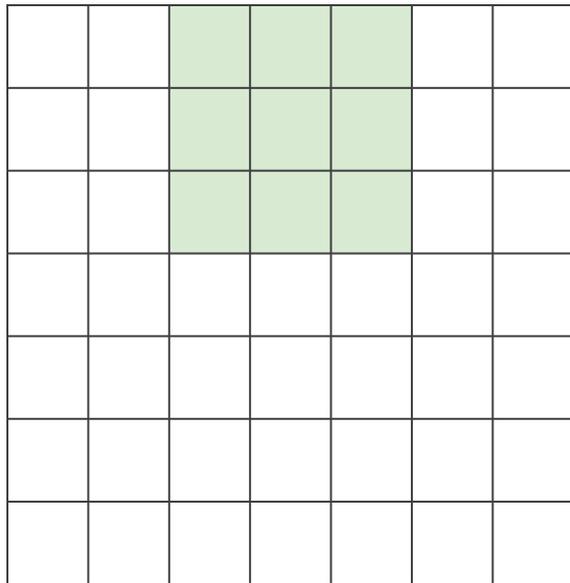
Solution: **Strided** Convolution



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

Solution: **Strided** Convolution

7



7

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

=> 3x3 output!

Convolution layer: summary

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

Conv layer needs 4 hyperparameters:

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

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

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

Number of parameters: F^2CK and K biases

Convolution layer: summary

Common settings:

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

Conv layer needs 4 hyperparameters:

$K =$ (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)

- $F = 1, S = 1, P = 0$

- Number of filters **K**

- The filter size **F**

- The stride **S**

- The zero padding **P**

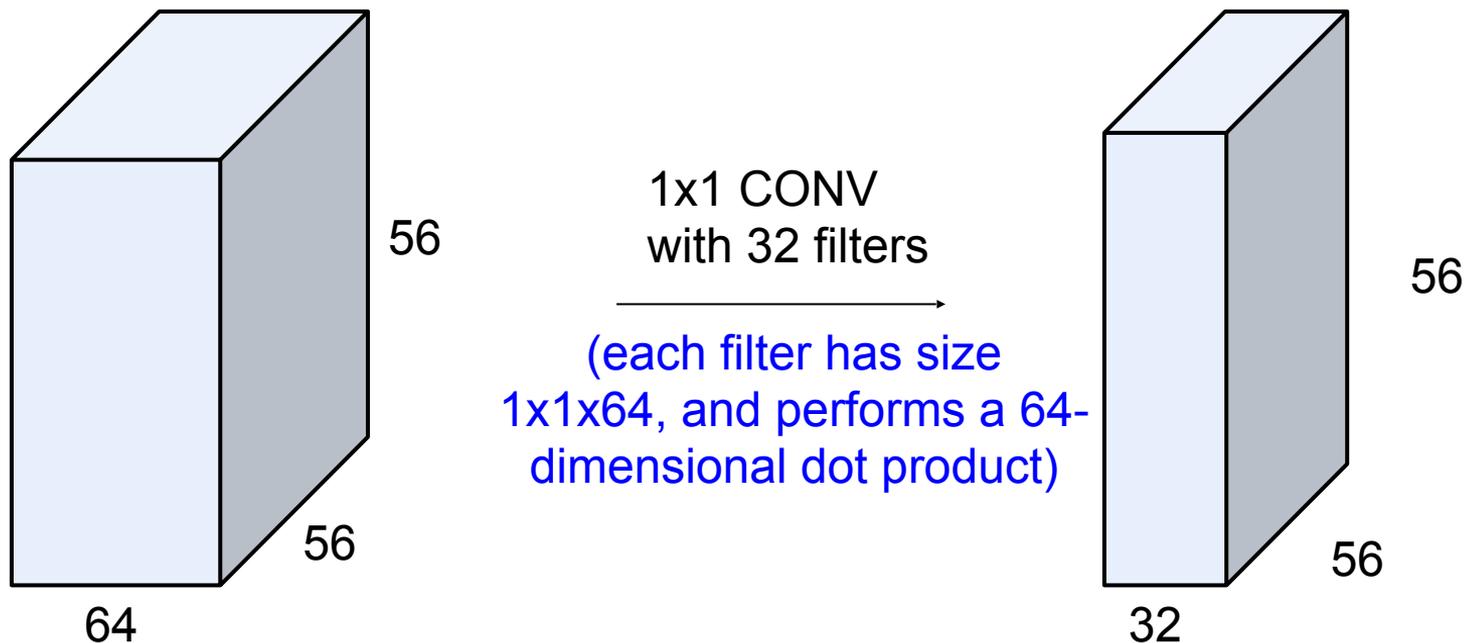
This will produce an output of $W_2 \times H_2 \times K$ where:

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

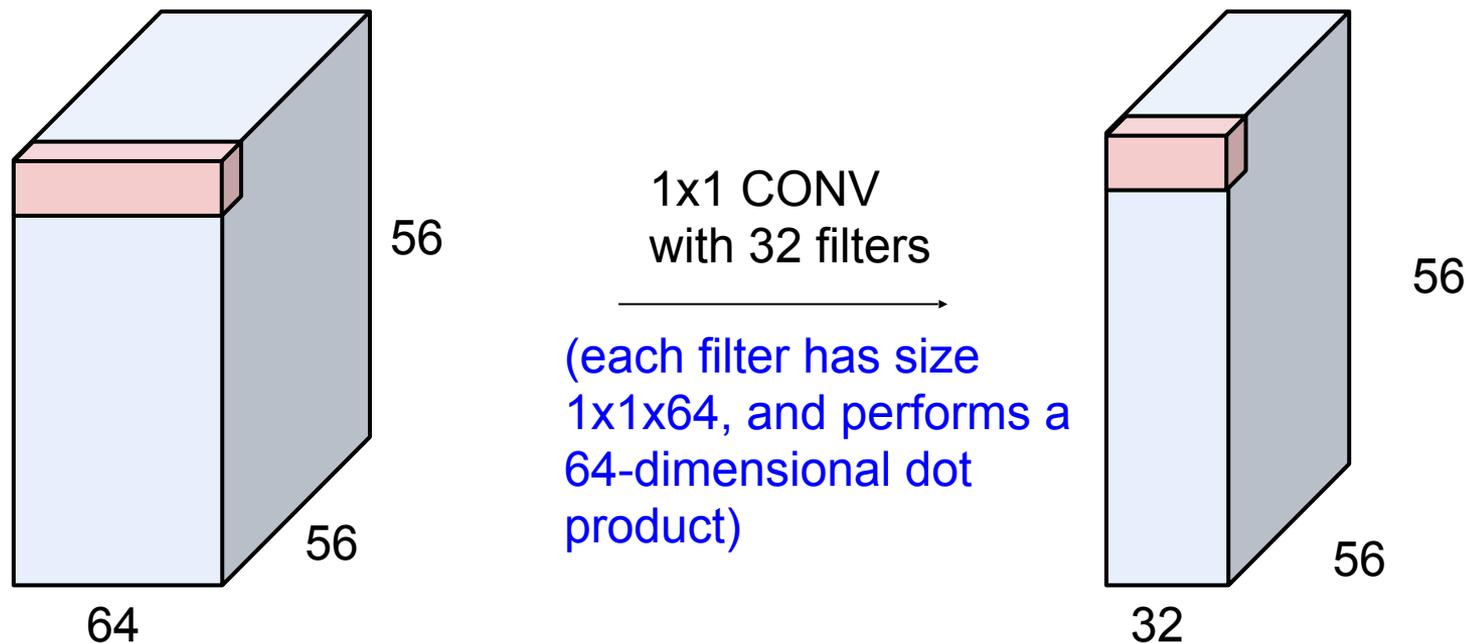
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

(btw, 1x1 convolution layers make perfect sense)



(btw, 1x1 convolution layers make perfect sense)



Example: CONV layer in PyTorch

Conv2d

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

[SOURCE]

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

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

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

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: $\begin{bmatrix} C_{out} \\ C_{in} \end{bmatrix}$.

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

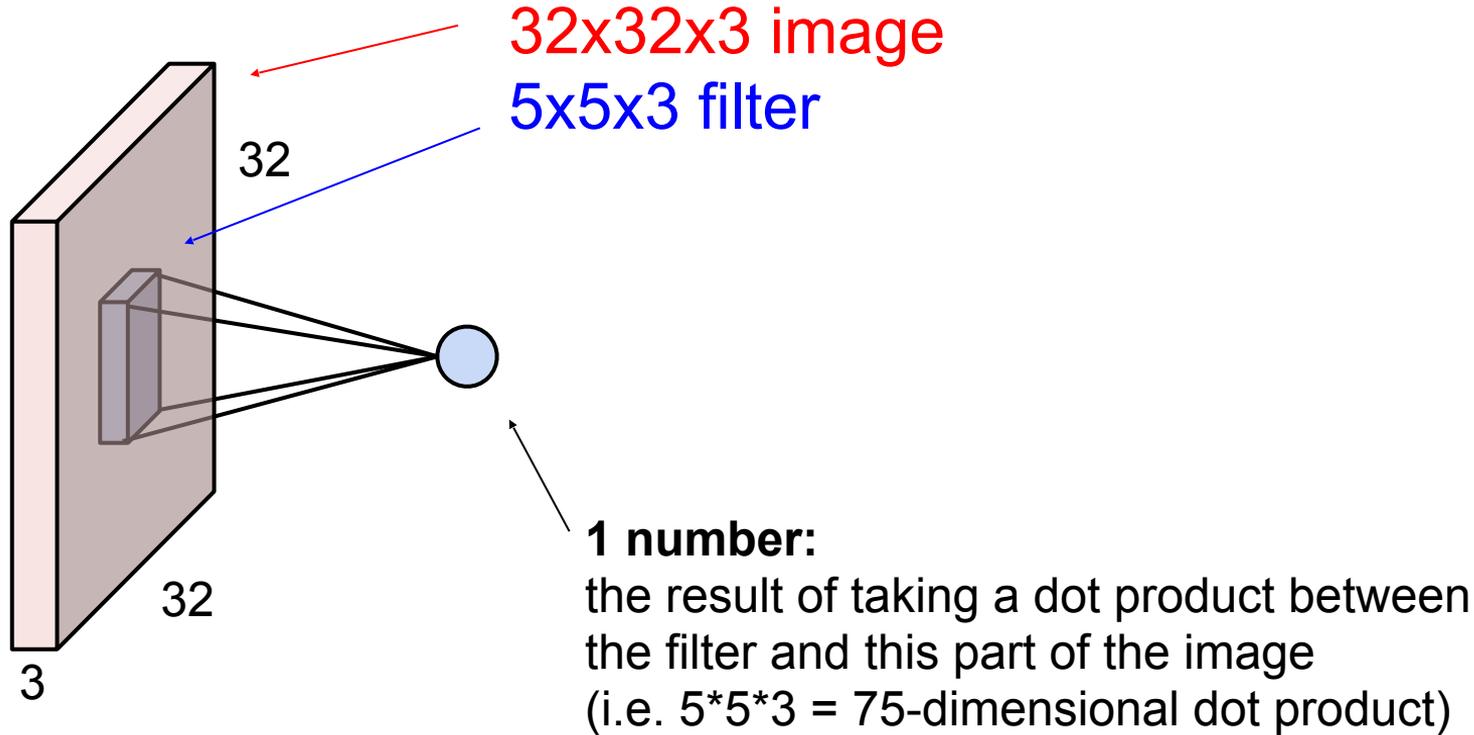
- a single `int` - in which case the same value is used for the height and width dimension
- a `tuple` of two ints - in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

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

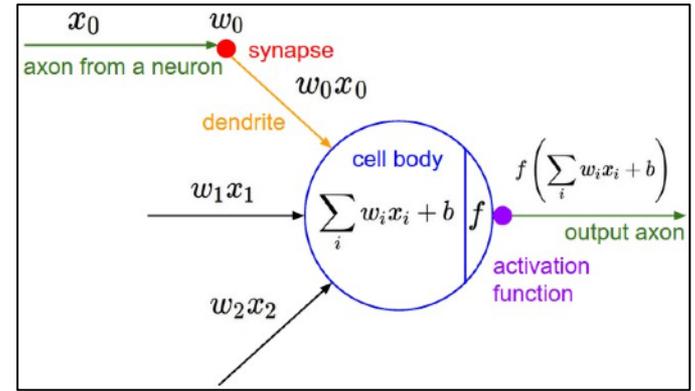
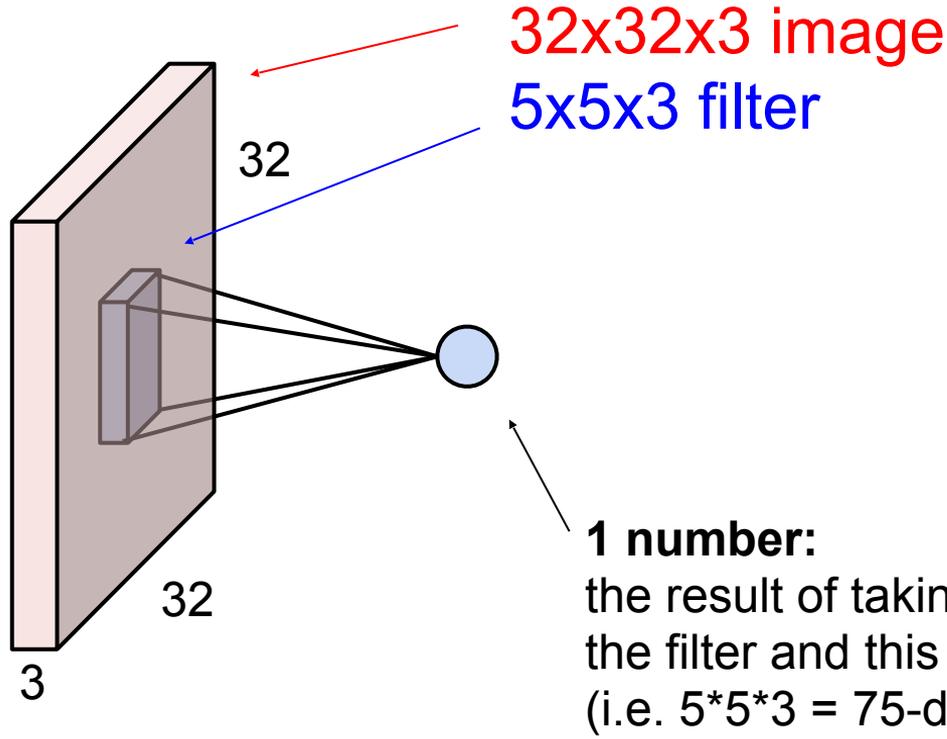
Conv layer needs 4 hyperparameters:

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

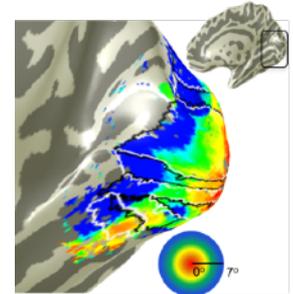
The brain/neuron view of CONV Layer



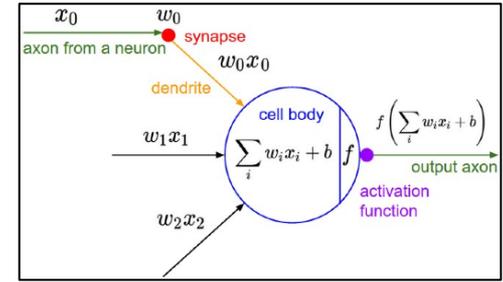
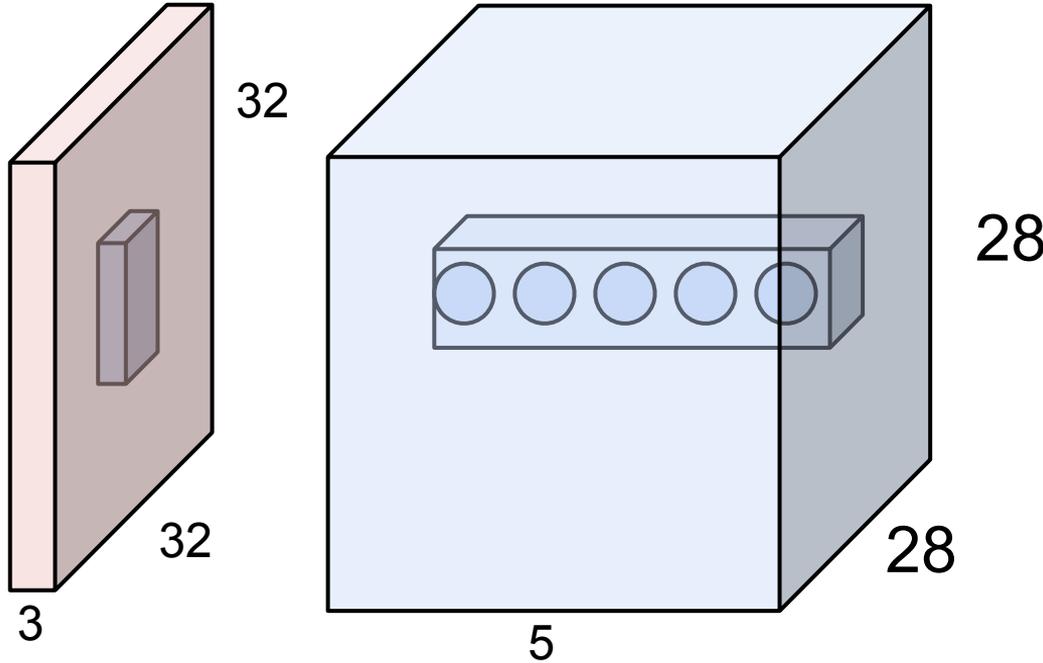
The brain/neuron view of CONV Layer



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



The brain/neuron view of CONV Layer



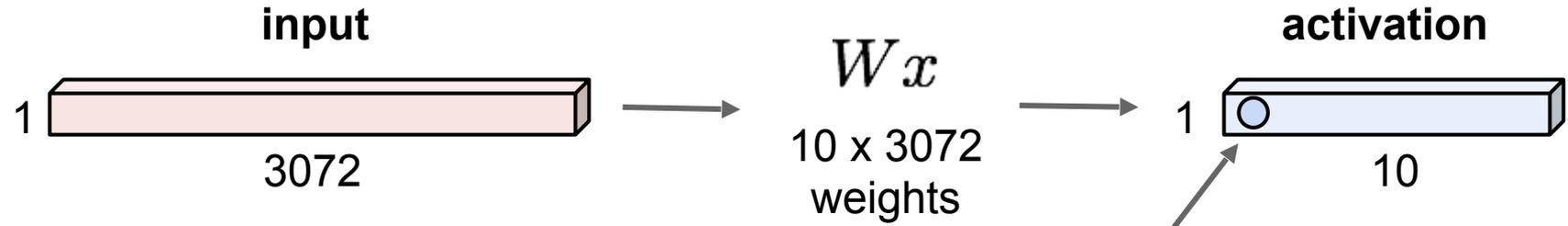
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

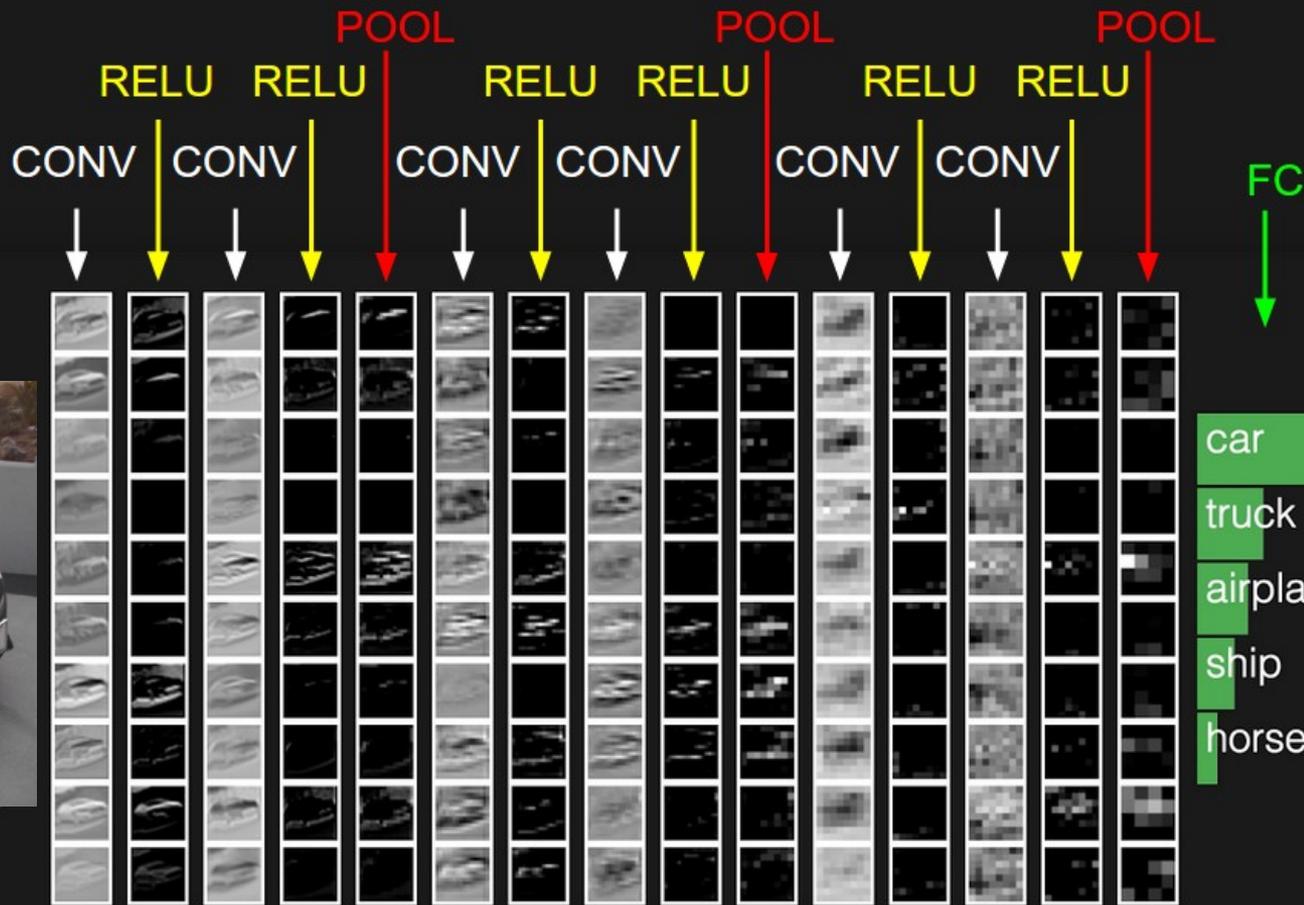
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

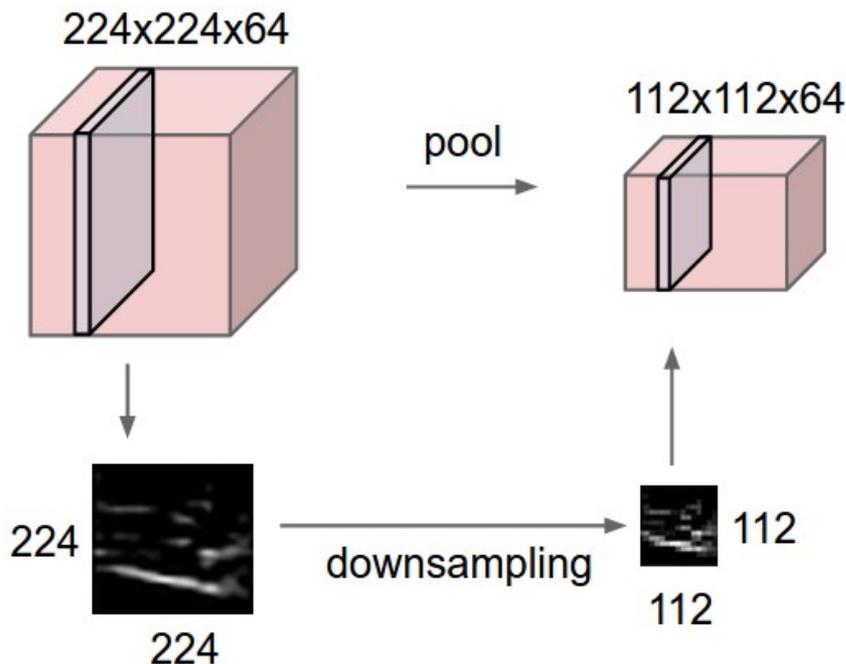


1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)



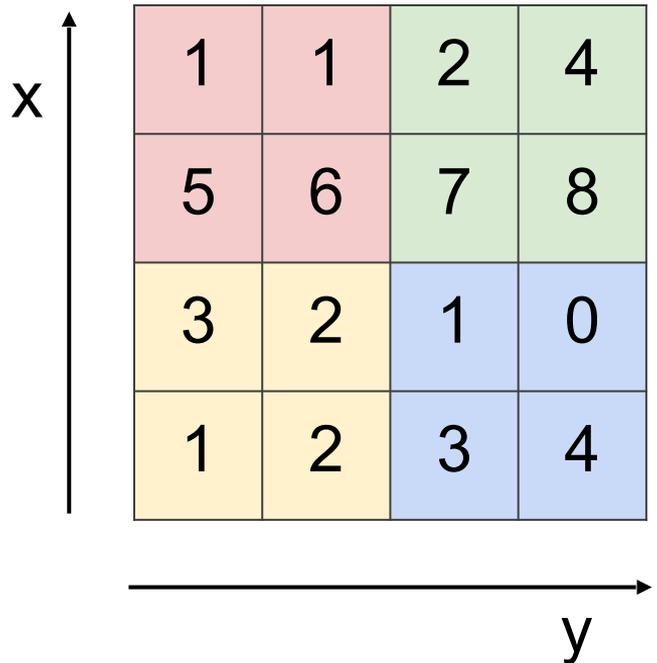
Pooling layer

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

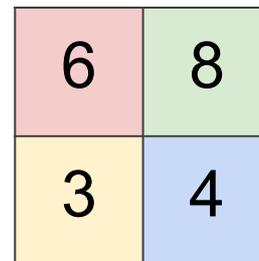


MAX POOLING

Single depth slice



max pool with 2x2 filters
and stride 2



MAX POOLING

Single depth slice

x ↑

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→ y

max pool with 2x2 filters
and stride 2



6	8
3	4

- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

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

Conv layer needs 2 hyperparameters:

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

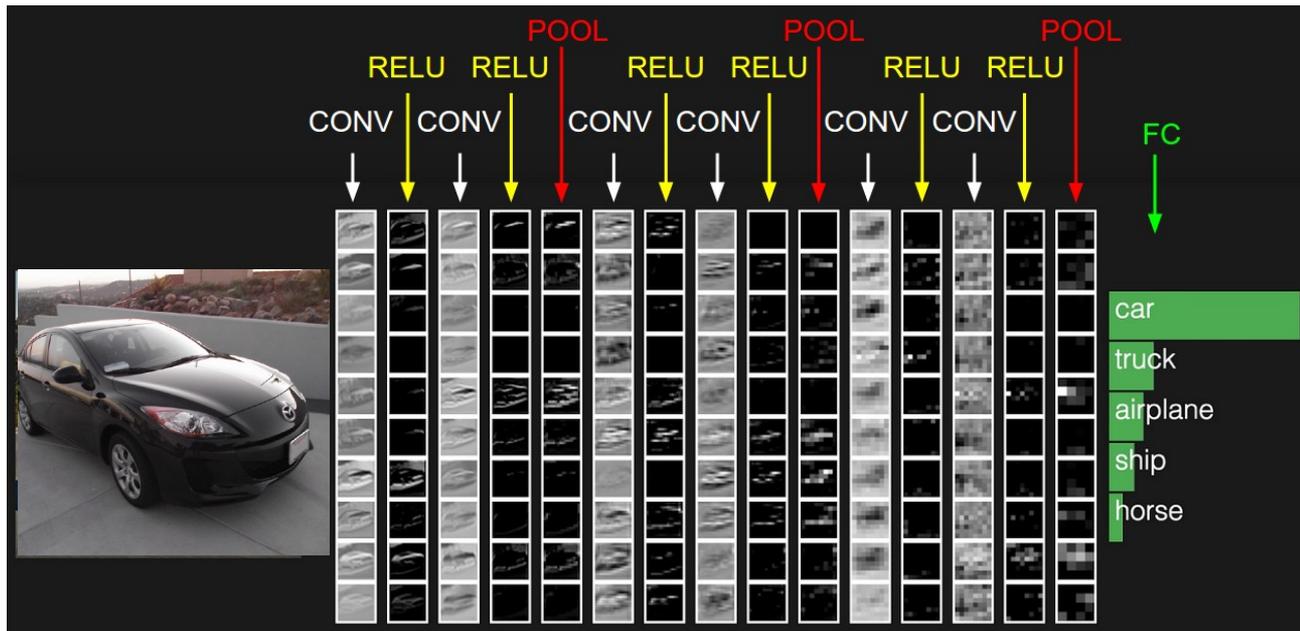
This will produce an output of $W_2 \times H_2 \times C$ where:

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

Number of parameters: 0

Fully Connected Layer (FC layer)

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



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

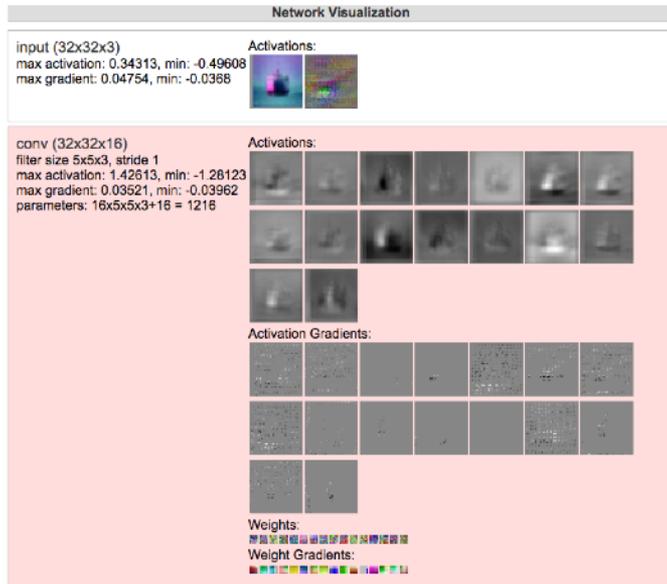
Description

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

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

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

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



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

Summary

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

