RECOD
reasoning for complex data

Machine Learning and Pattern Recognition

A High Level Overview

Prof. Anderson Rocha

(Main bulk of slides kindly provided by **Prof. Sandra Avila**
and based on materials by Fei-Fei Li & Justin Johnson & Serena Yeung)

Institute of Computing (IC/Unicamp)

MC886/MO444

DNNs are everywhere ...

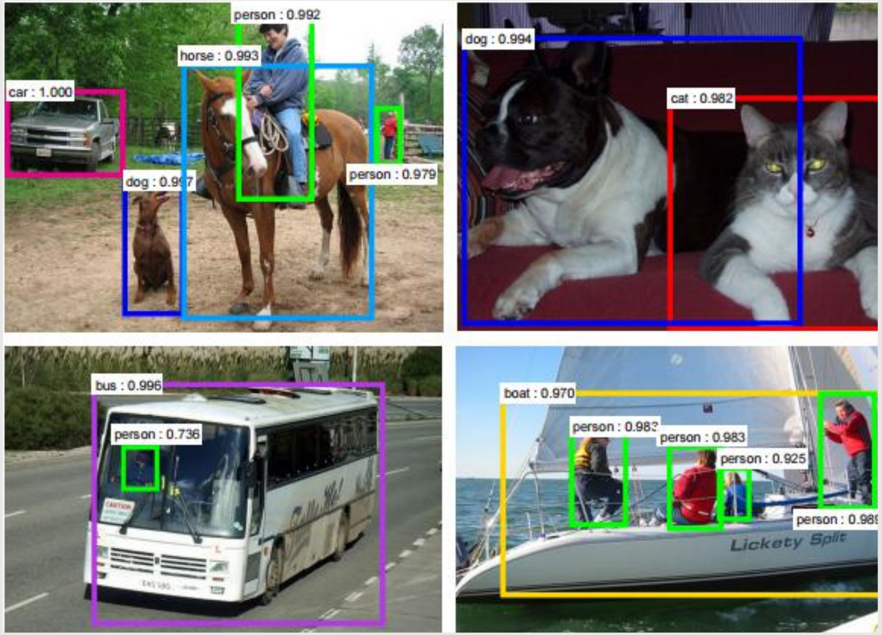
Classification

Retrieval

Credit: Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012

DNNs are everywhere ...

Detection



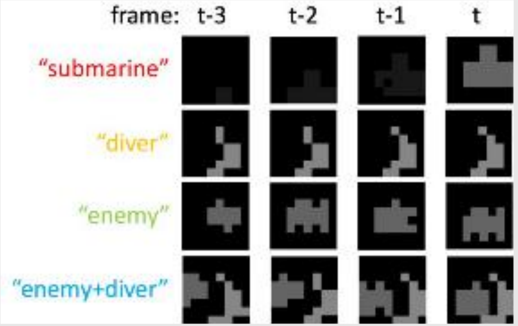
Segmentation

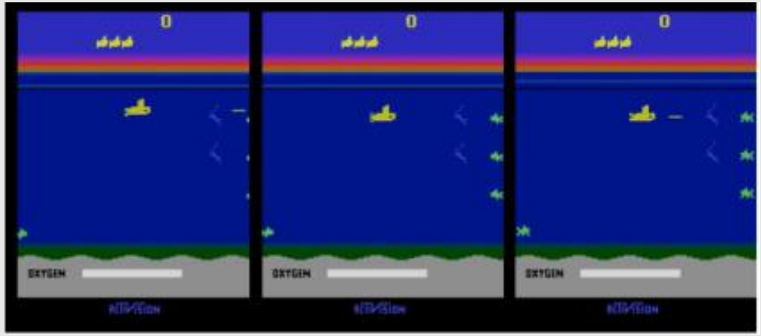
Credit: Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Clement Farabet, 2012.

DNNs are everywhere ...

Pose Estimation

Playing Games





Credit: Toshev & Szegedy 2014. Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014.

DNNs are everywhere ...

Image Captioning

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk.

DNNs are everywhere ...

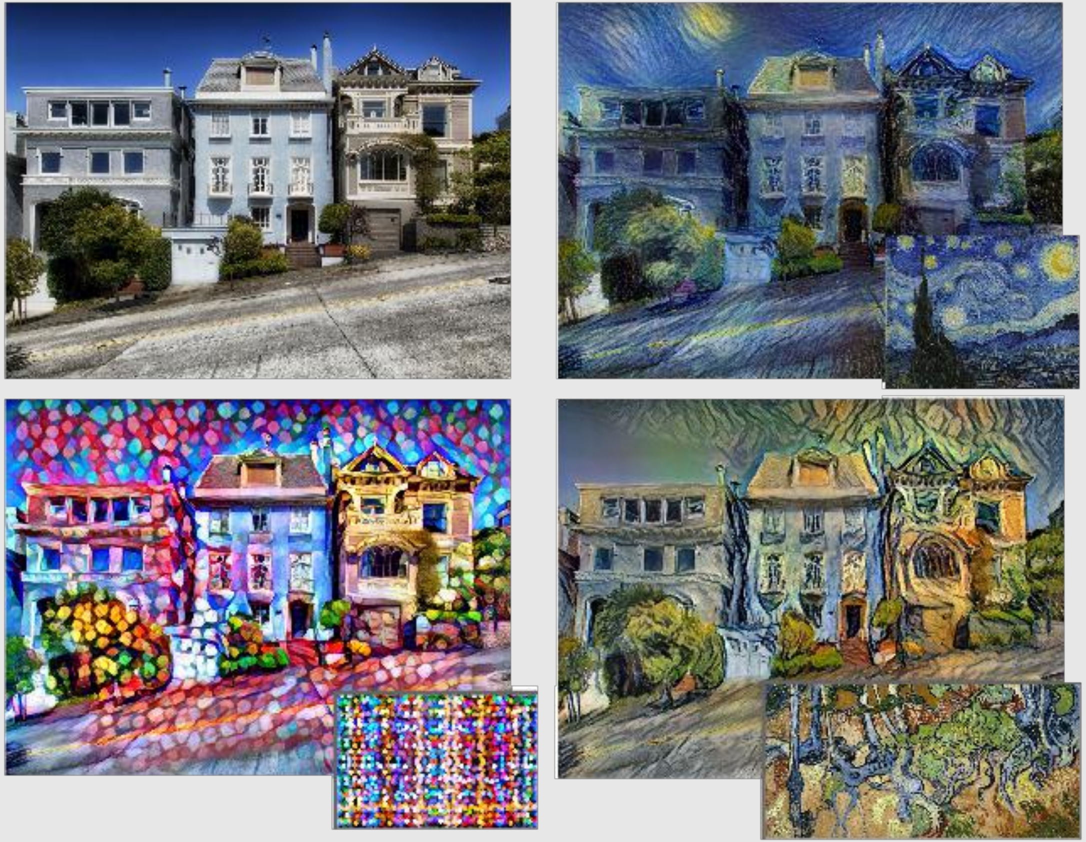
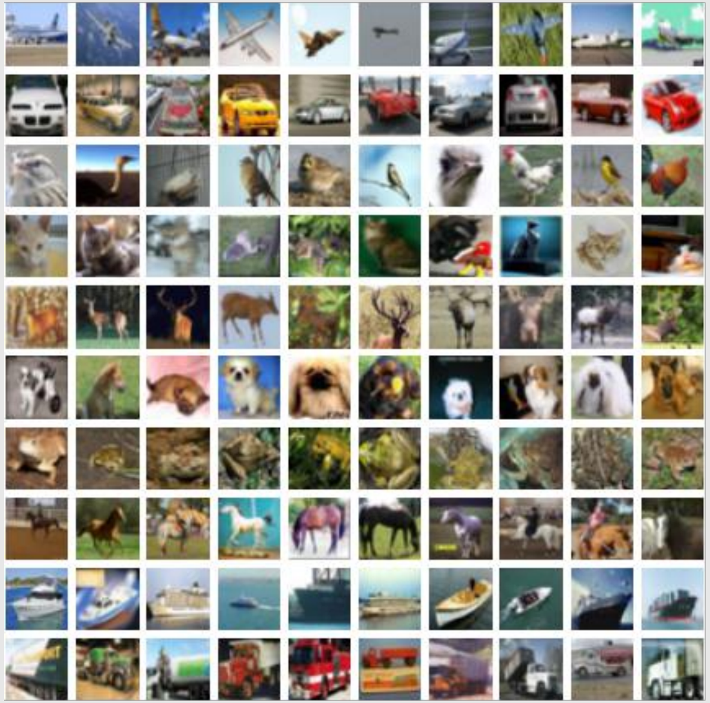


Image Style Transfer

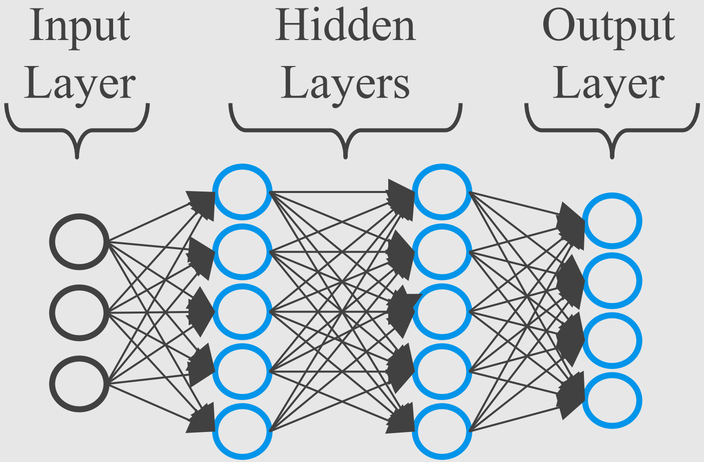
Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016
Gatys et al, “Controlling Perceptual Factors in Neural Style Transfer”, CVPR 2017

Convolutional Neural Networks (CNNs)

Fully Connected Layer

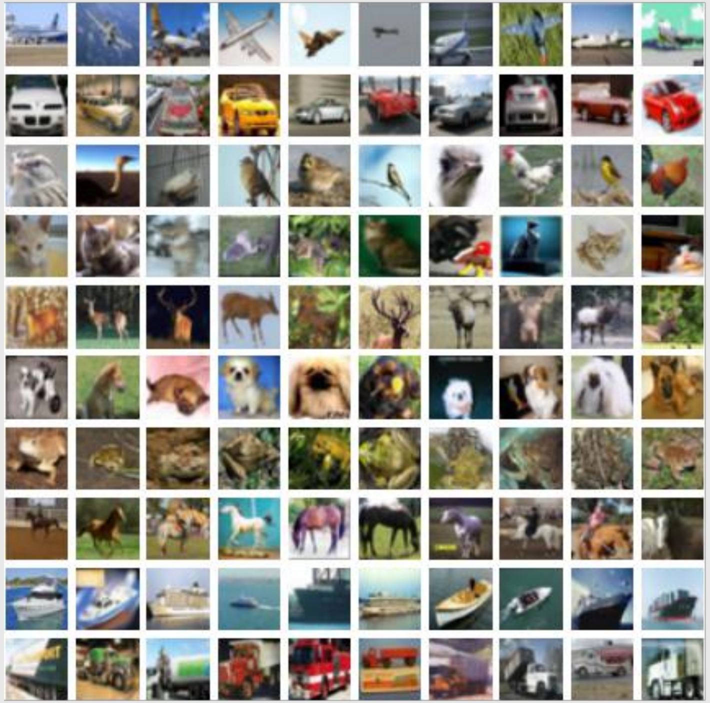


CIFAR-10

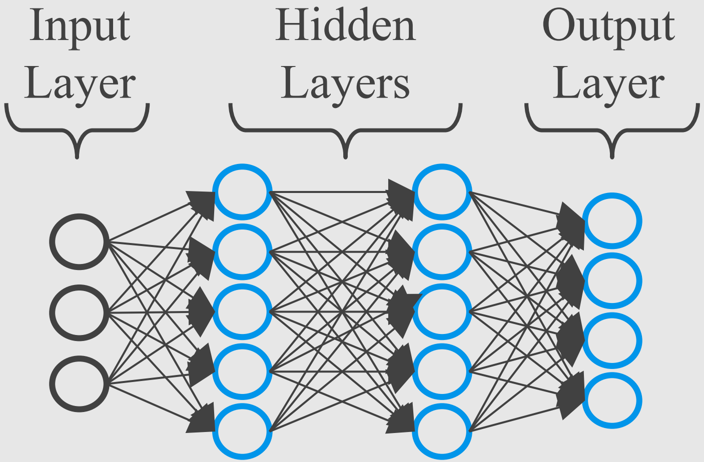


$32 \times 32 \times 3$ image $\Rightarrow$ stretch to 3072×1

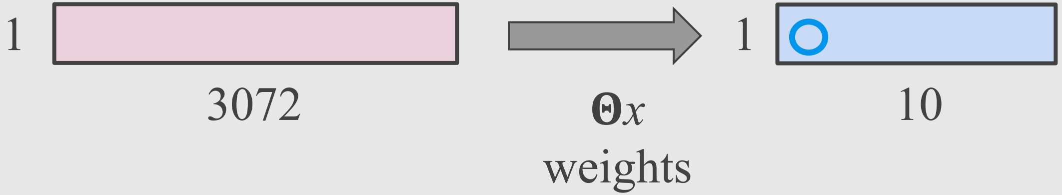
Fully Connected Layer



CIFAR-10

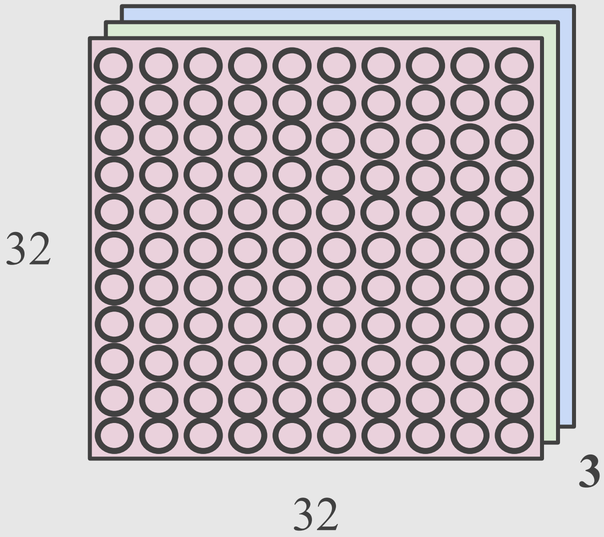


$32 \times 32 \times 3$ image $\Rightarrow$ stretch to 3072×1



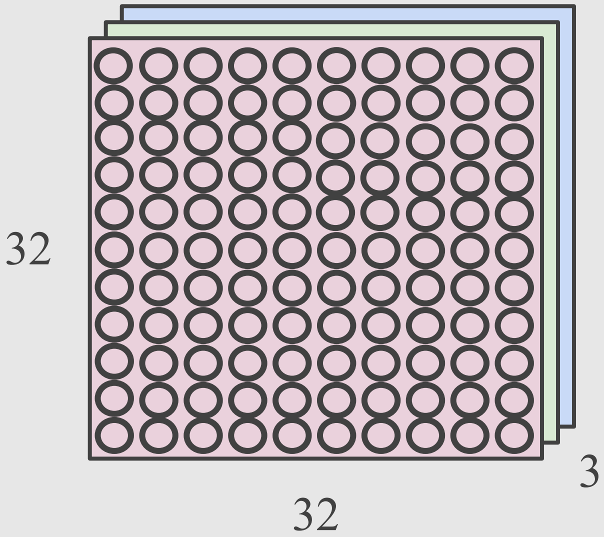
Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



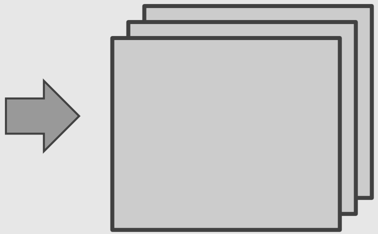
Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



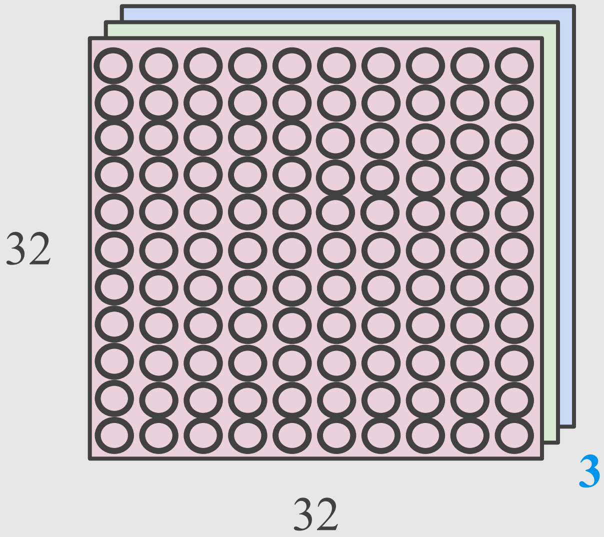
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

$5 \times 5 \times 3$ filter



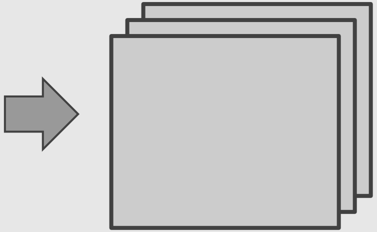
Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



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

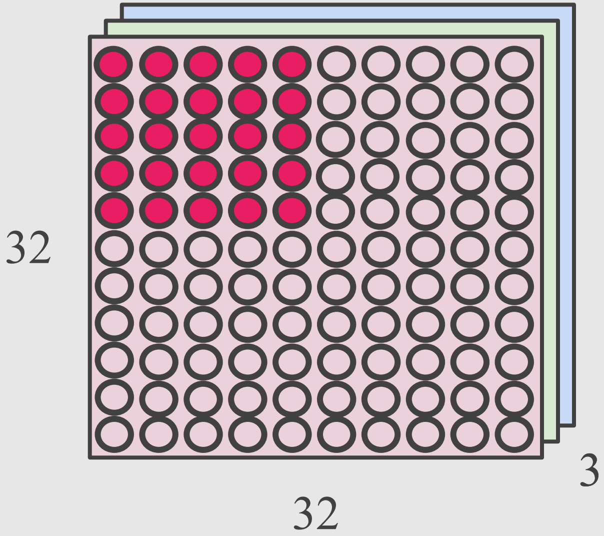
$5 \times 5 \times \mathbf{3}$ filter



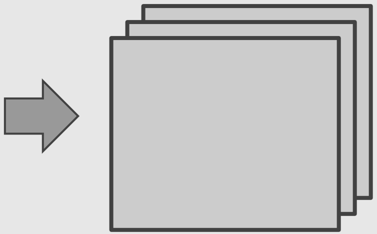
Filters always extend the full depth of the input volume

Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure

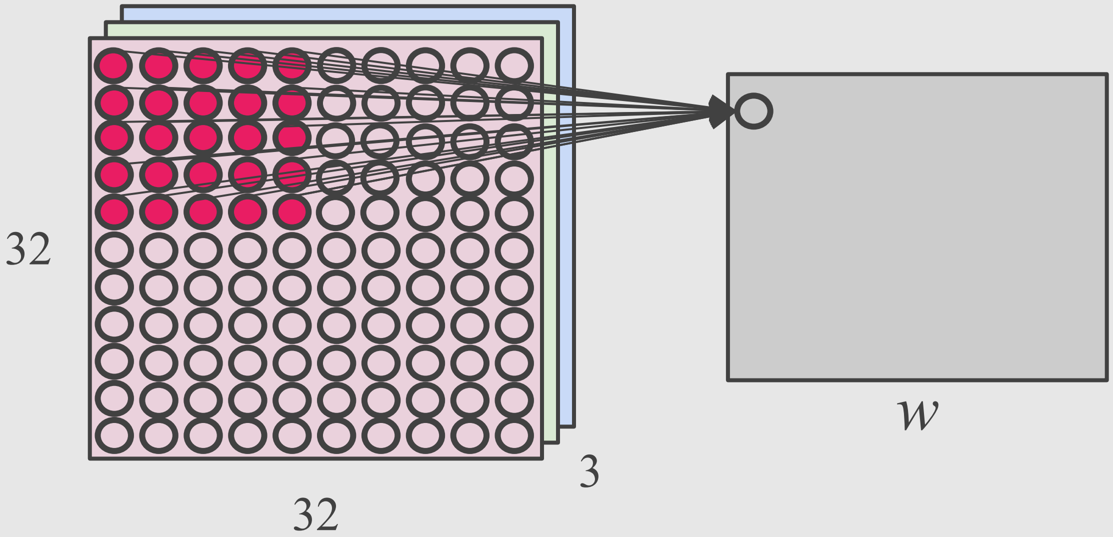


$5 \times 5 \times 3$ filter



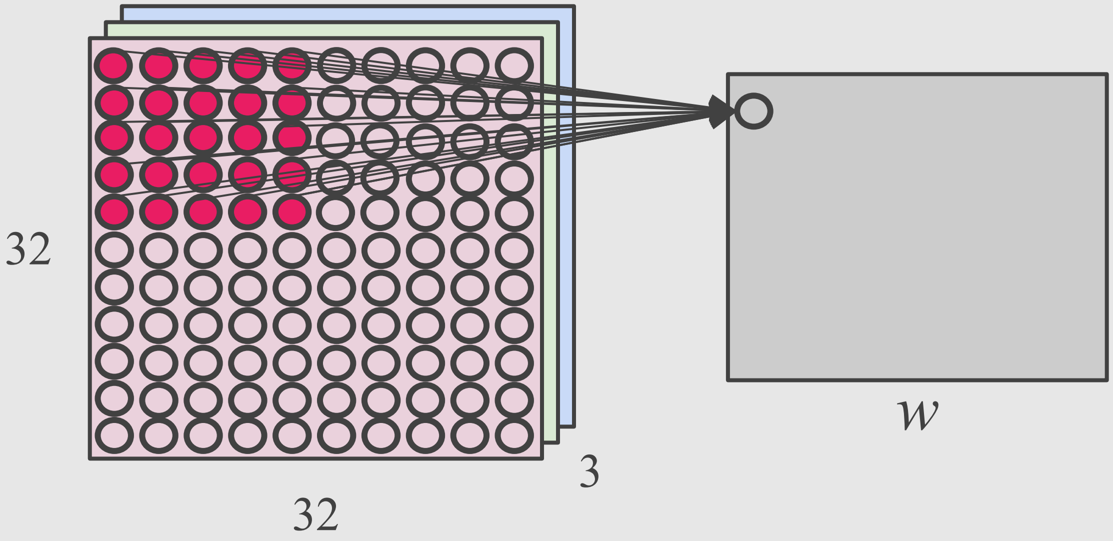
Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



1 number:

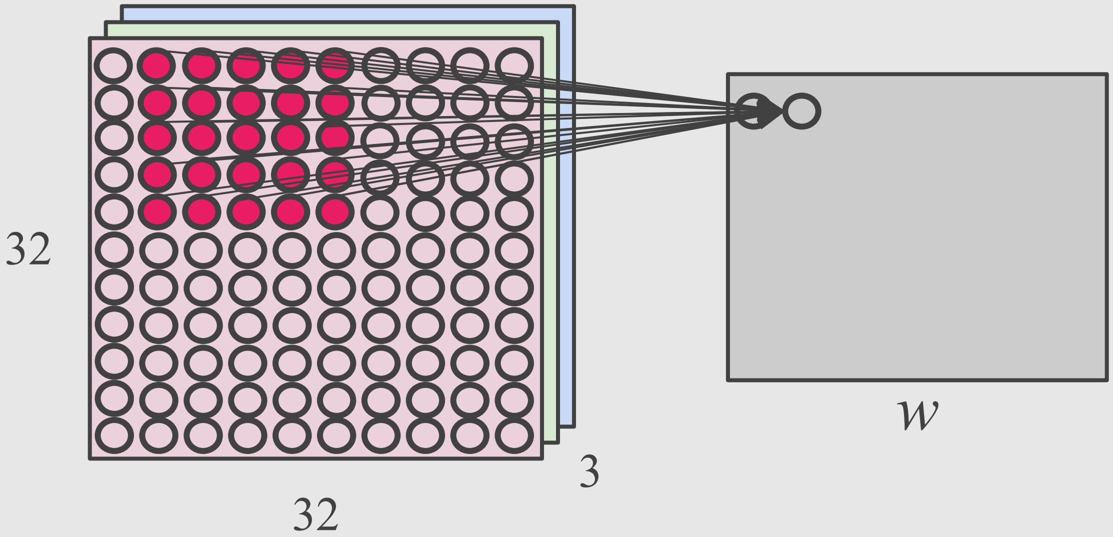
5*5*3 = 75-dimensional

dot product + bias)

$w^T x + b$

Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



1 number:

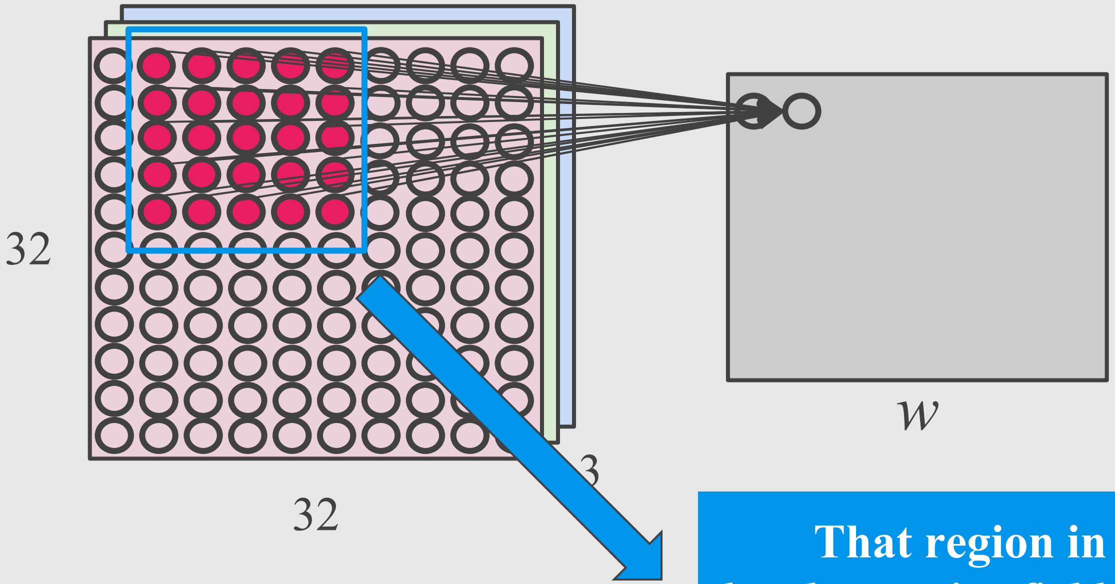
5*5*3 = 75-dimensional

dot product + bias)

$w^{T}x + b$

Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



1 number:

5*5*3 = 75-dimensional

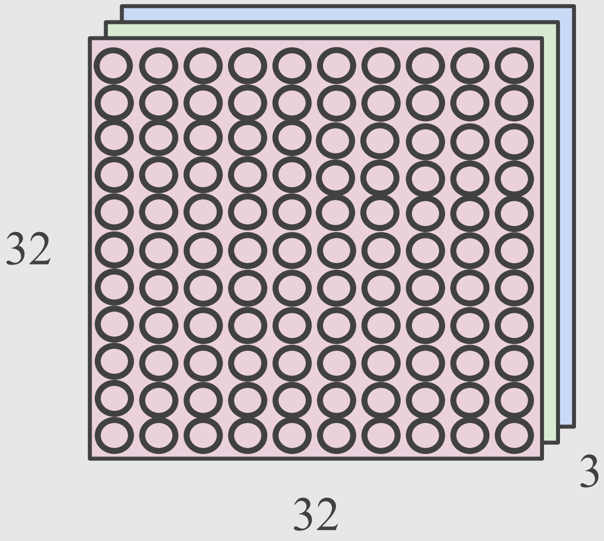
dot product + bias)

$w^T x + b$

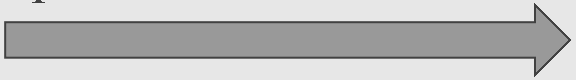
That region in the input image is called the *local receptive field* for the hidden neuron.

Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure

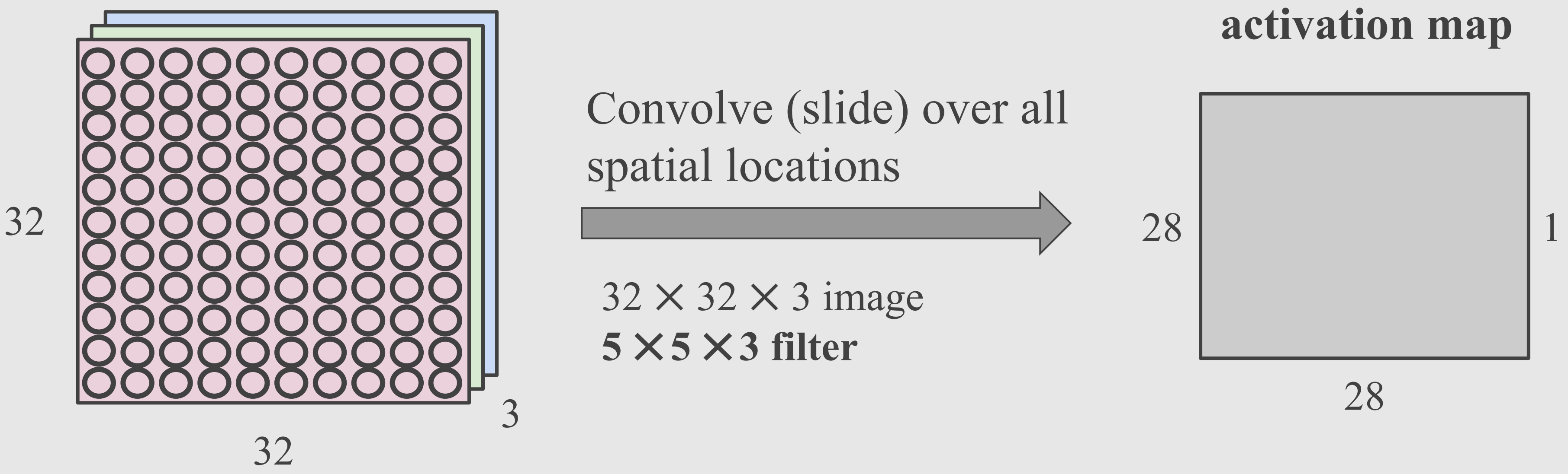


Convolve (slide) over all spatial locations



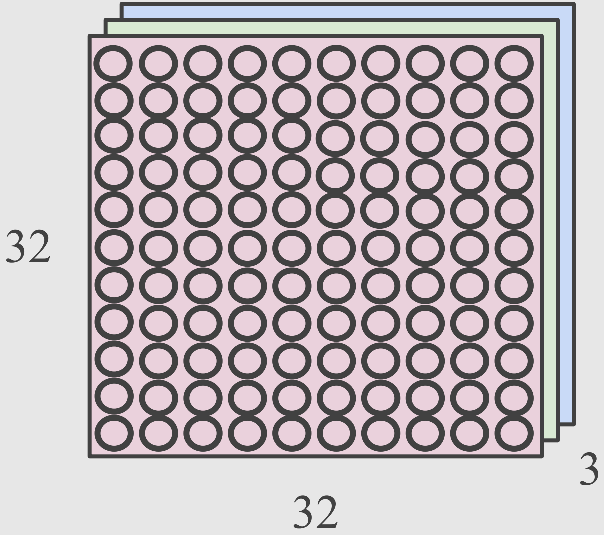
Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure

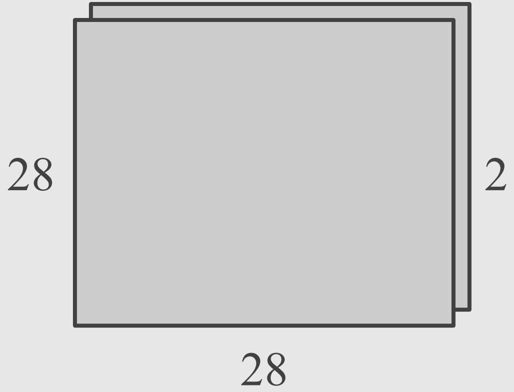


Convolve (slide) over all spatial locations

$32 \times 32 \times 3$ image
$5 \times 5 \times 3$ filter

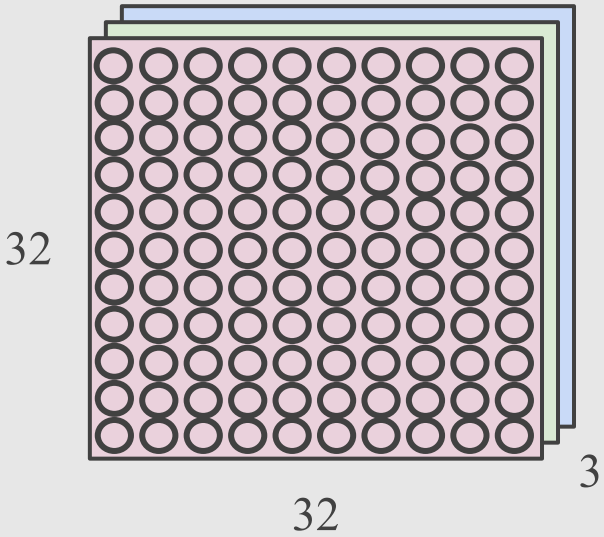
(considering a second filter)

activation maps



Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



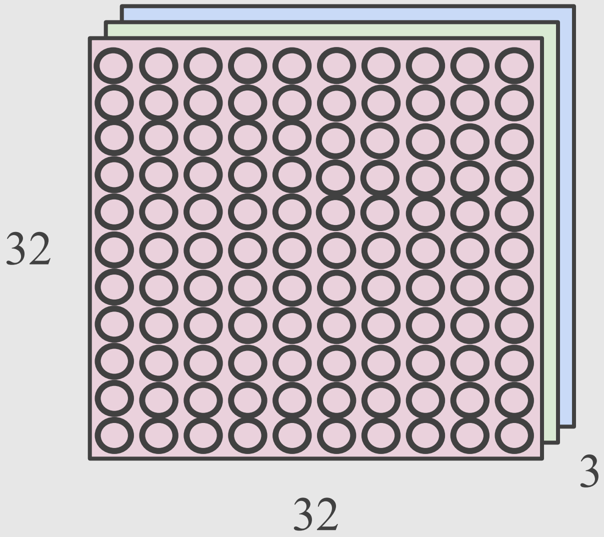
Convolve (slide) over all spatial locations

$32 \times 32 \times 3$ image
$5 \times 5 \times 3$ filter

If we had 6 filters $5 \times 5 \times 3$...

Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



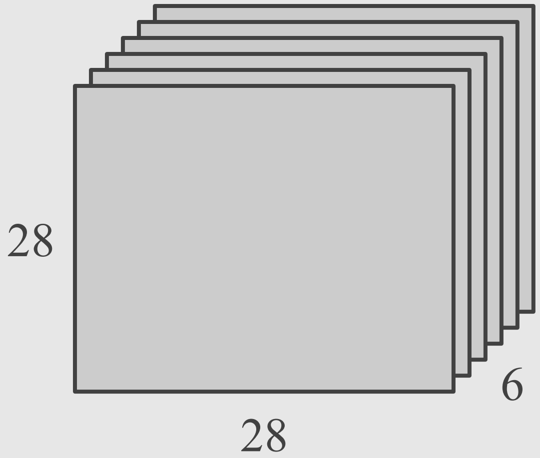
Convolve (slide) over all spatial locations

$32 \times 32 \times 3$ image

$5 \times 5 \times 3$ filter

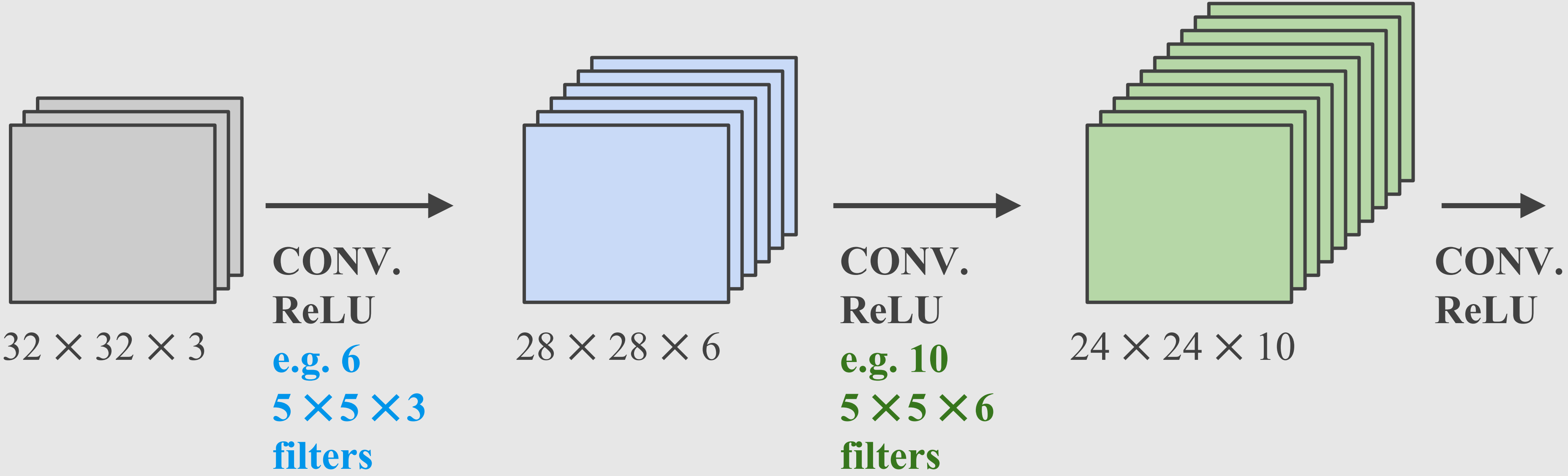
If we had 6 filters $5 \times 5 \times 3$...

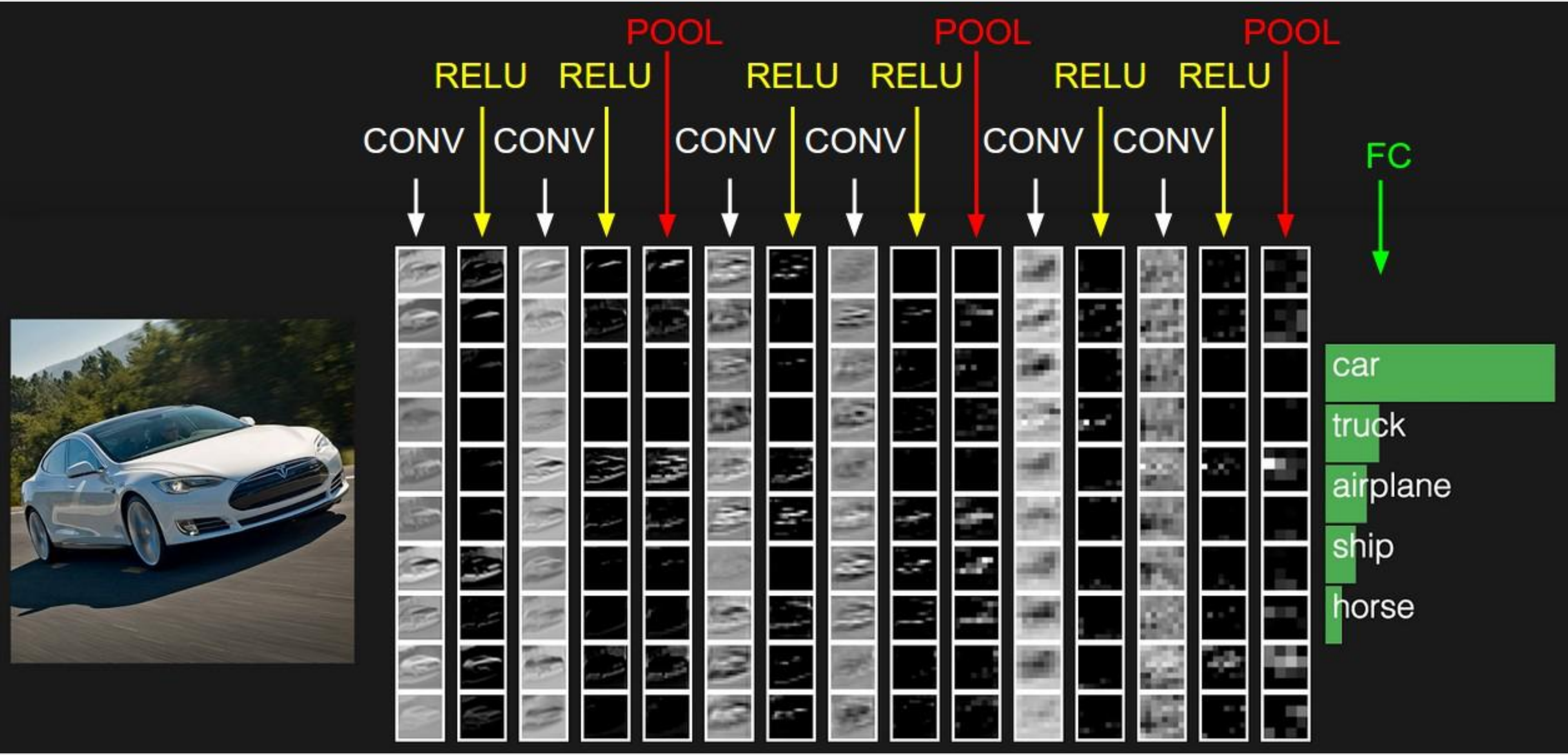
6 activation maps



Convolutional Networks

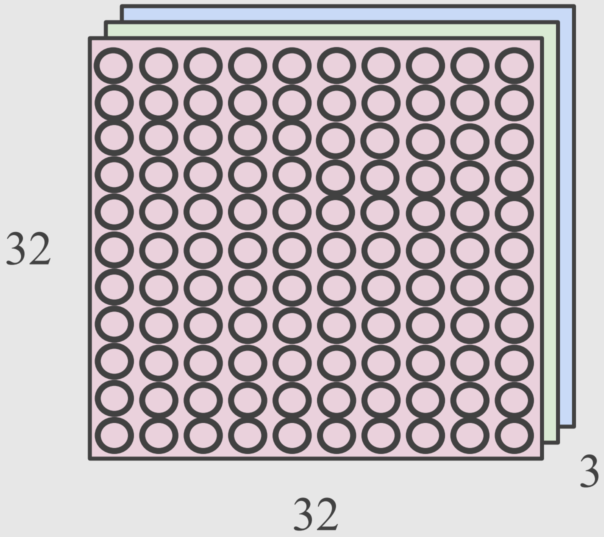
Sequence of Convolutional Layers, interspersed with activation functions.





Credit: http://cs231n.github.io/convolutional-networks/

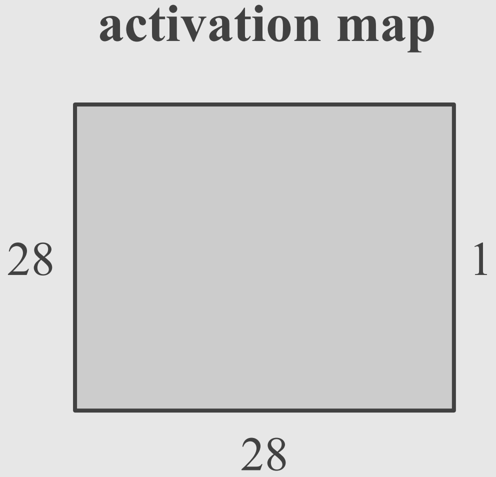
A Closer Look at Spatial Dimensions



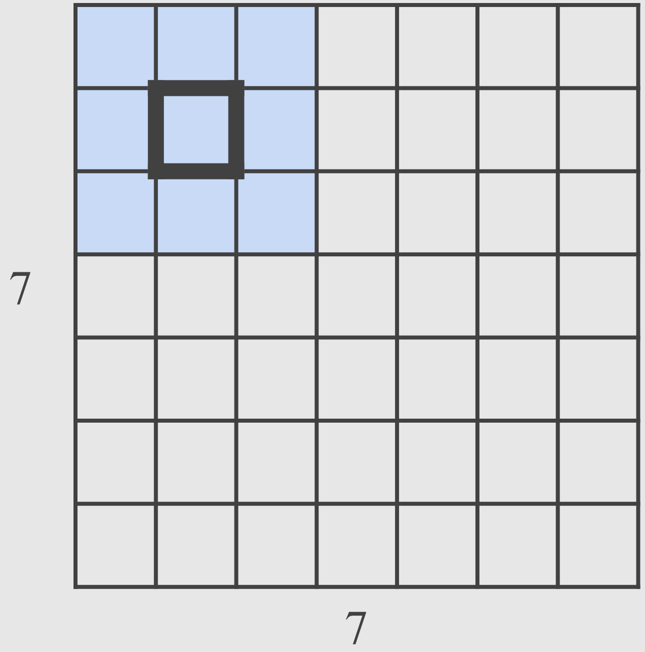
Convolve (slide) over all spatial locations

$32 \times 32 \times 3$ image

$5 \times 5 \times 3$ filter

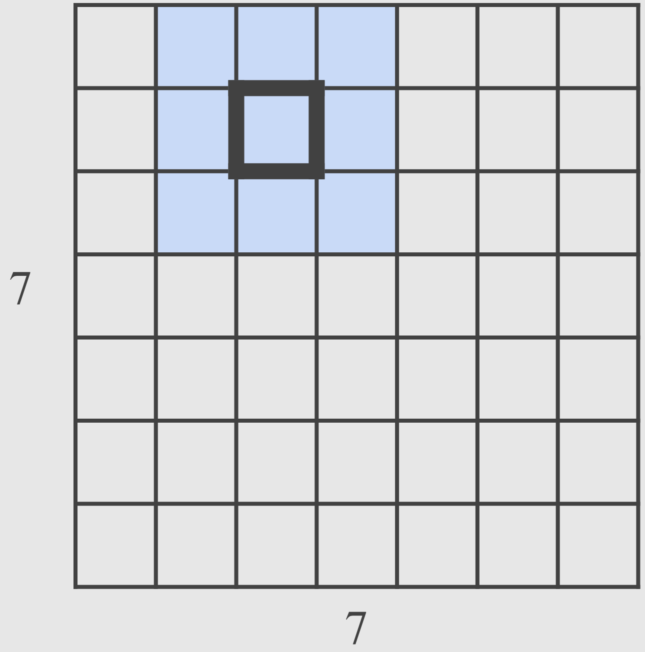


A Closer Look at Spatial Dimensions



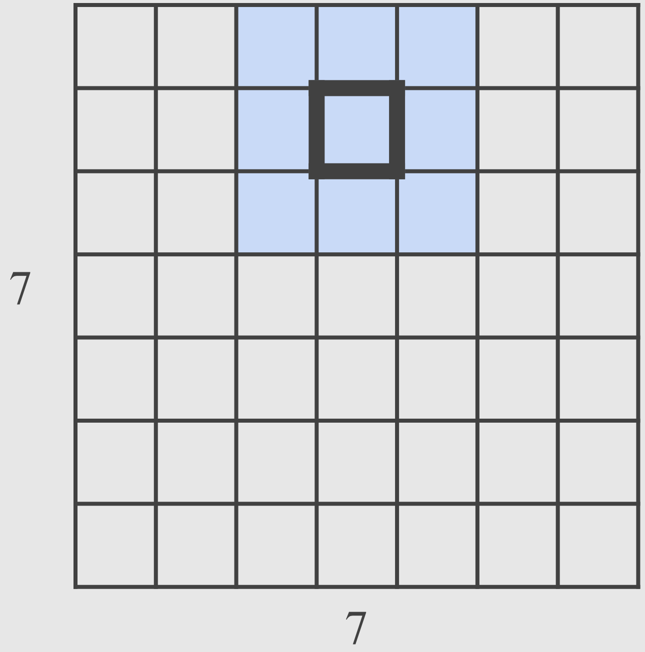
7×7 input (spatially)
assume 3×3 filter

A Closer Look at Spatial Dimensions



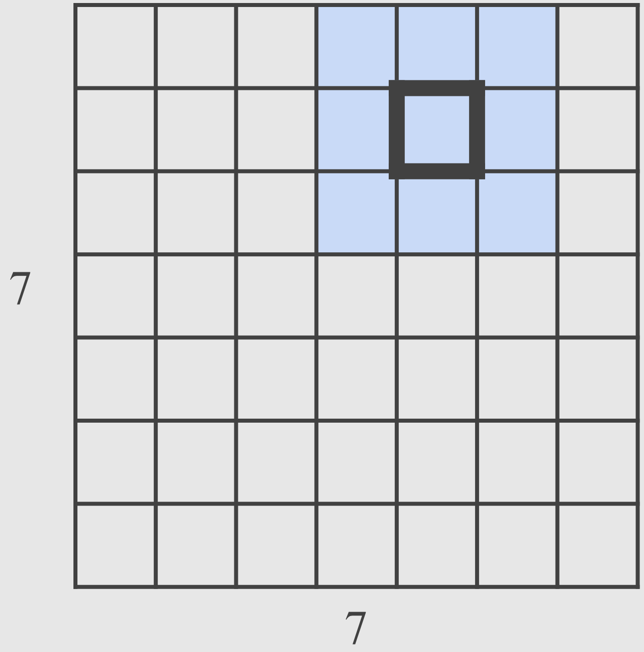
7×7 input (spatially)
assume 3×3 filter

A Closer Look at Spatial Dimensions



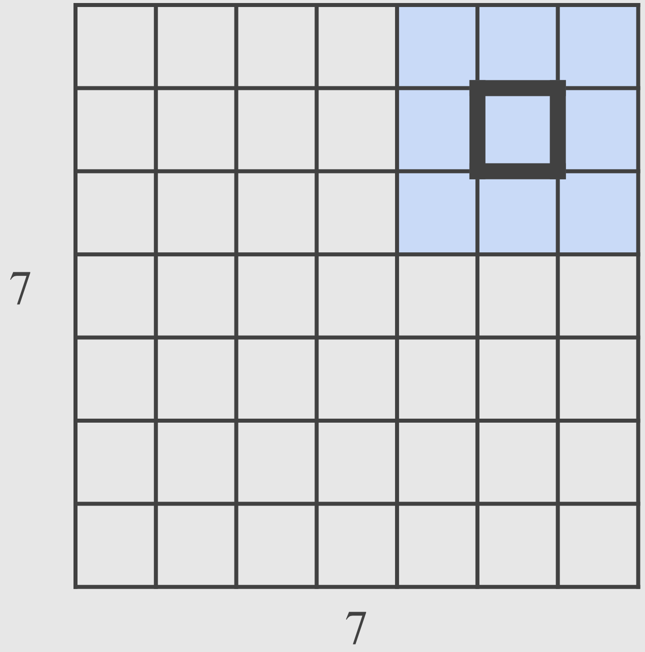
7×7 input (spatially)
assume 3×3 filter

A Closer Look at Spatial Dimensions



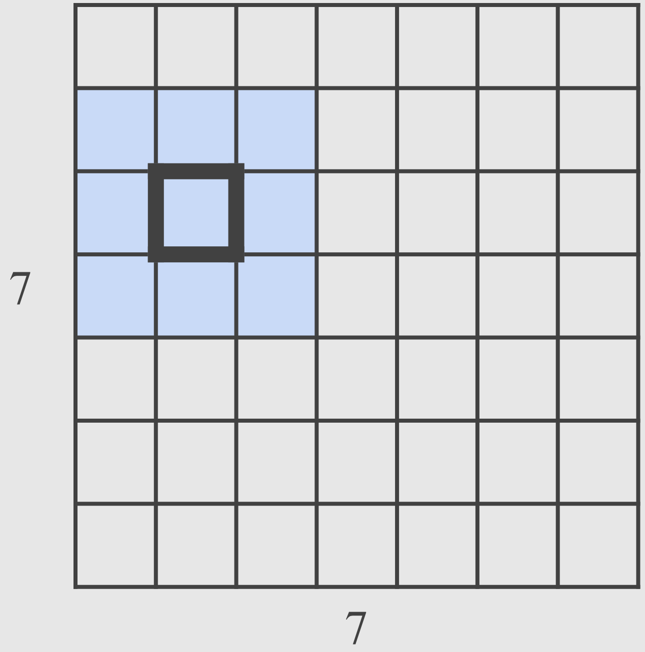
7×7 input (spatially)
assume 3×3 filter

A Closer Look at Spatial Dimensions



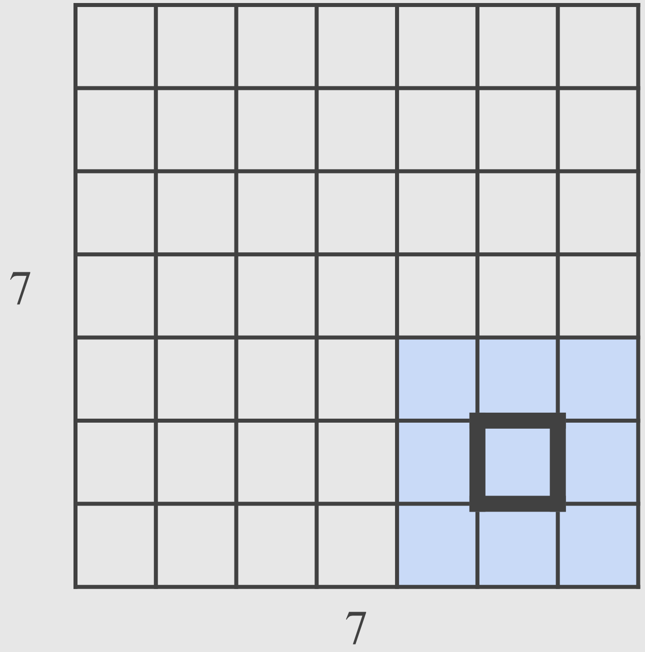
7×7 input (spatially)
assume 3×3 filter

A Closer Look at Spatial Dimensions



7×7 input (spatially)
assume 3×3 filter

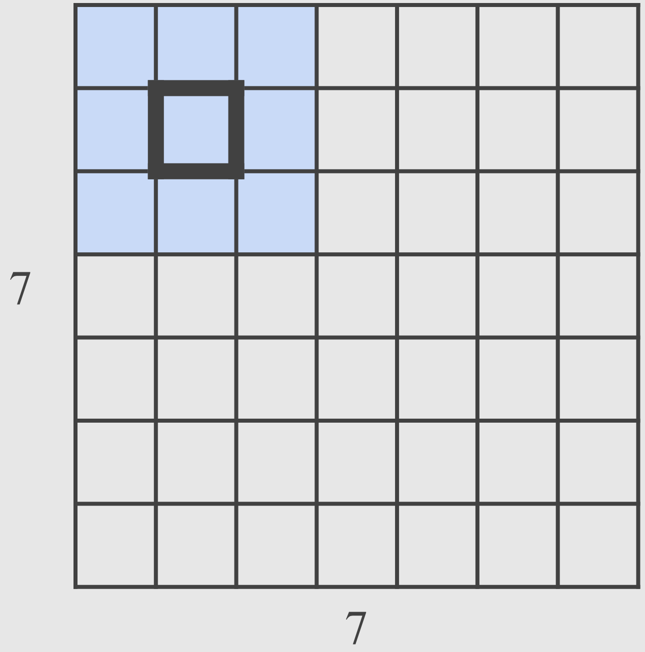
A Closer Look at Spatial Dimensions



7×7 input (spatially)
assume 3×3 filter

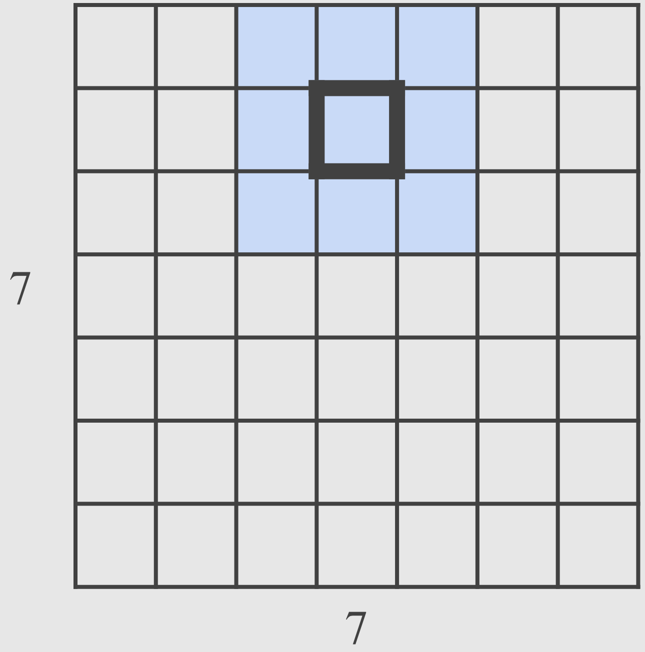
$\Rightarrow$ **5×5 output**

A Closer Look at Spatial Dimensions



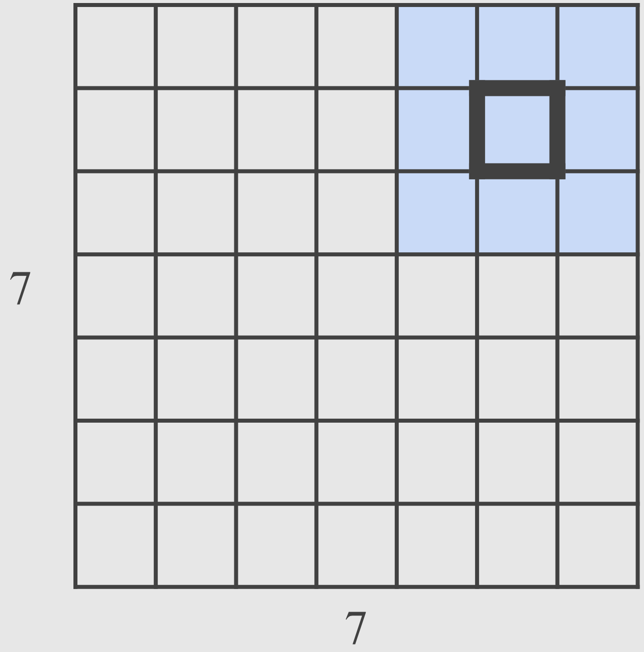
7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

A Closer Look at Spatial Dimensions



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

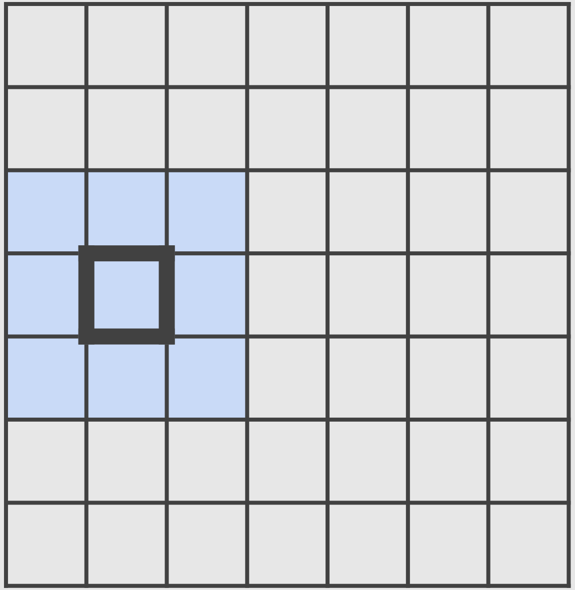
A Closer Look at Spatial Dimensions



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

A Closer Look at Spatial Dimensions

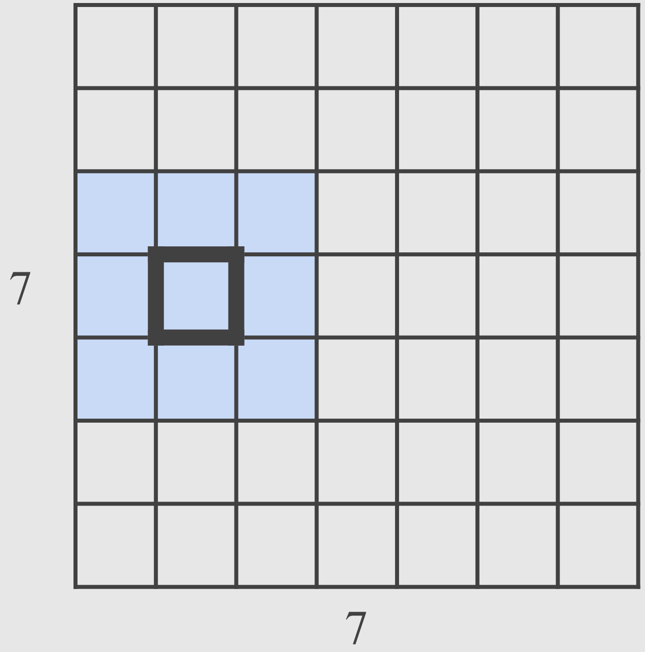
7



7

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

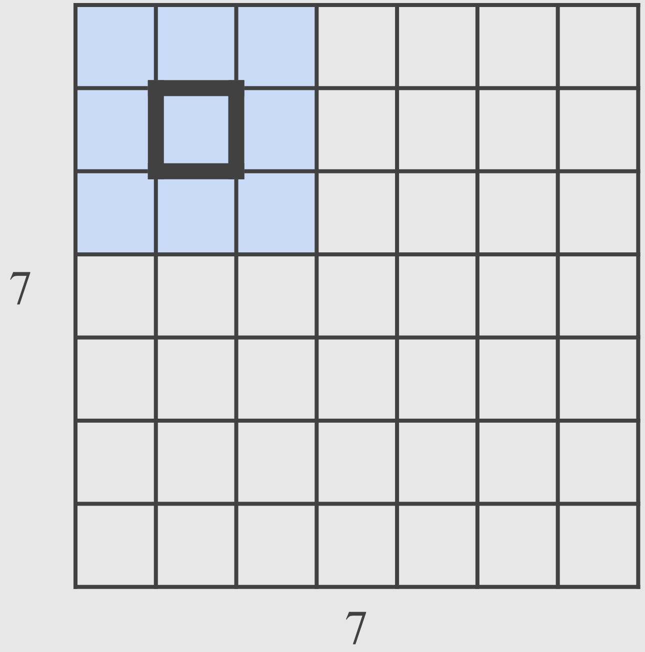
A Closer Look at Spatial Dimensions



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

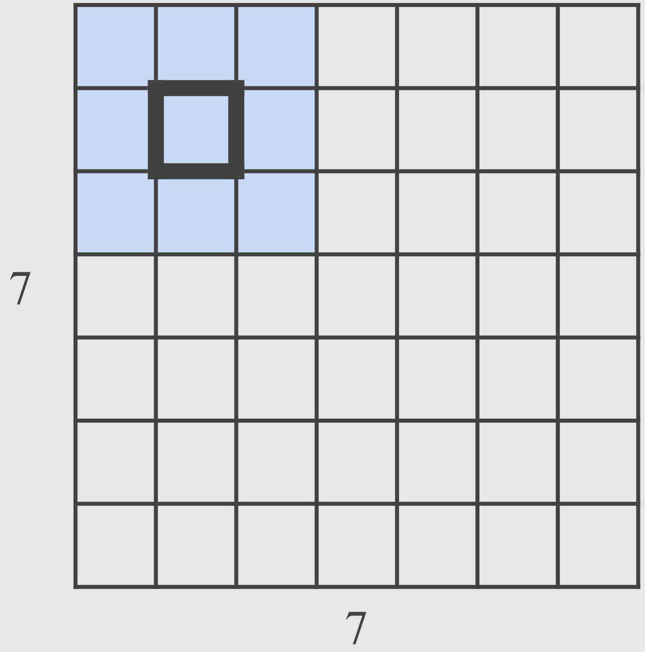
$\Rightarrow$ 3×3 output

A Closer Look at Spatial Dimensions



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

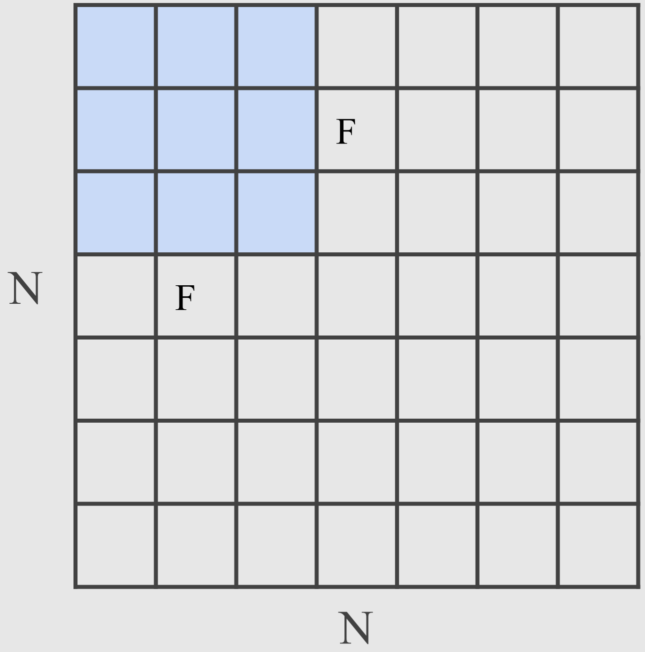
A Closer Look at Spatial Dimensions



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

Doesn't fit!
cannot apply 3×3 filter on 7×7 input with stride 3.

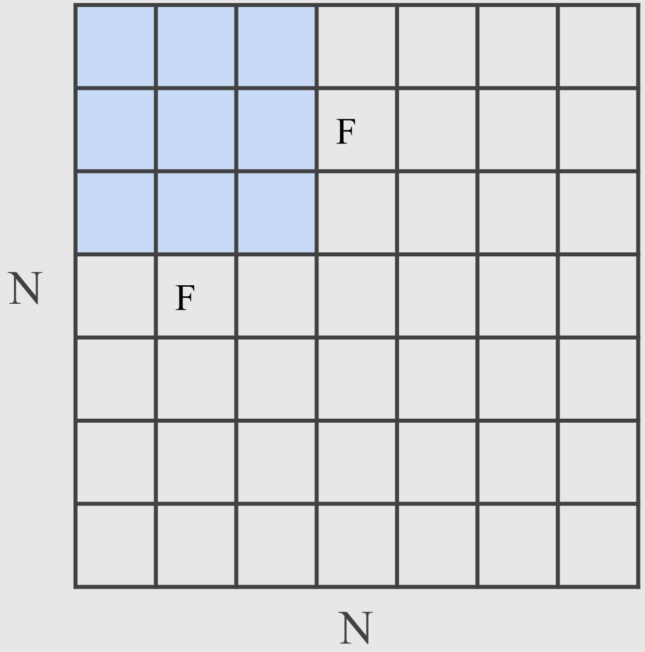
A Closer Look at Spatial Dimensions



Output size:

(N - F) / stride + 1

A Closer Look at Spatial Dimensions



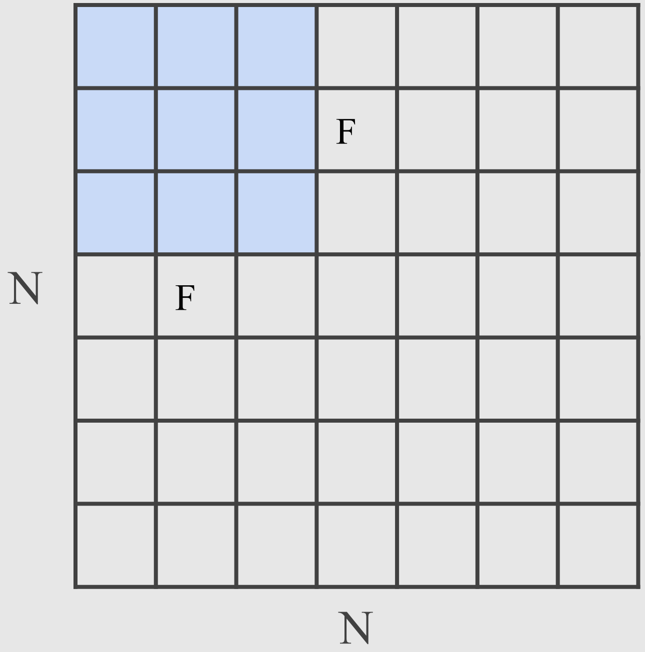
Output size:

(N - F) / stride + 1

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

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

A Closer Look at Spatial Dimensions



Output size:

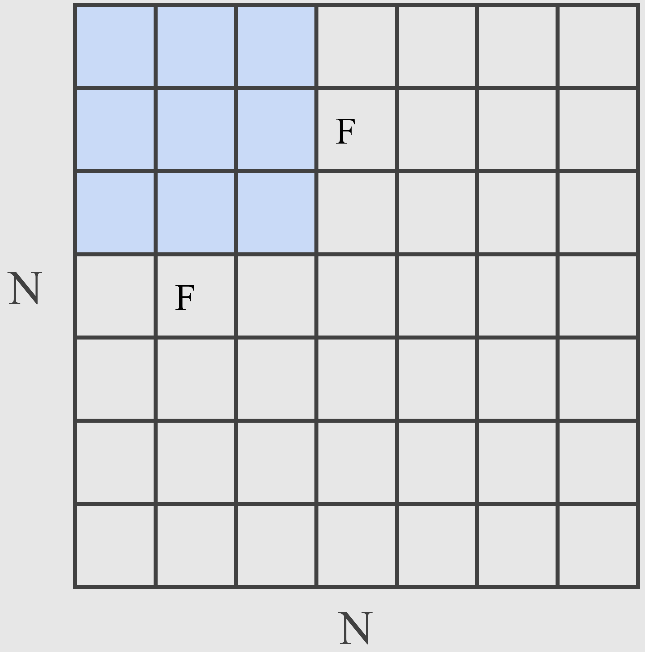
(N - F) / stride + 1

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

stride 1 ⇒ (7 - 3)/1 + 1 = 5

stride 2 ⇒ (7 - 3)/2 + 1 = 3

A Closer Look at Spatial Dimensions



Output size:

$(N - F)$ / stride + 1

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

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

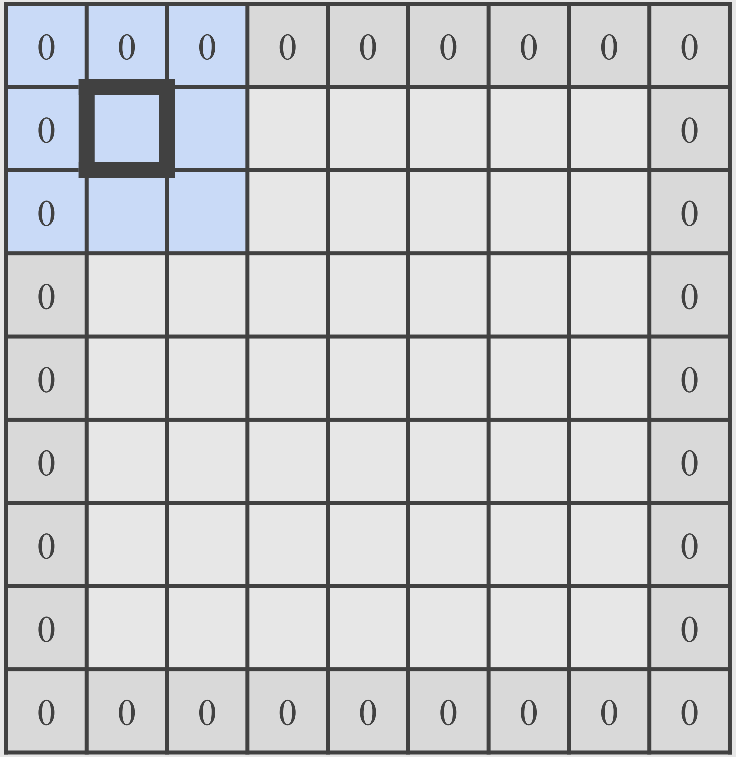
stride 2 $\Rightarrow (7 - 3)/2 + 1 = 3$

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

In Practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

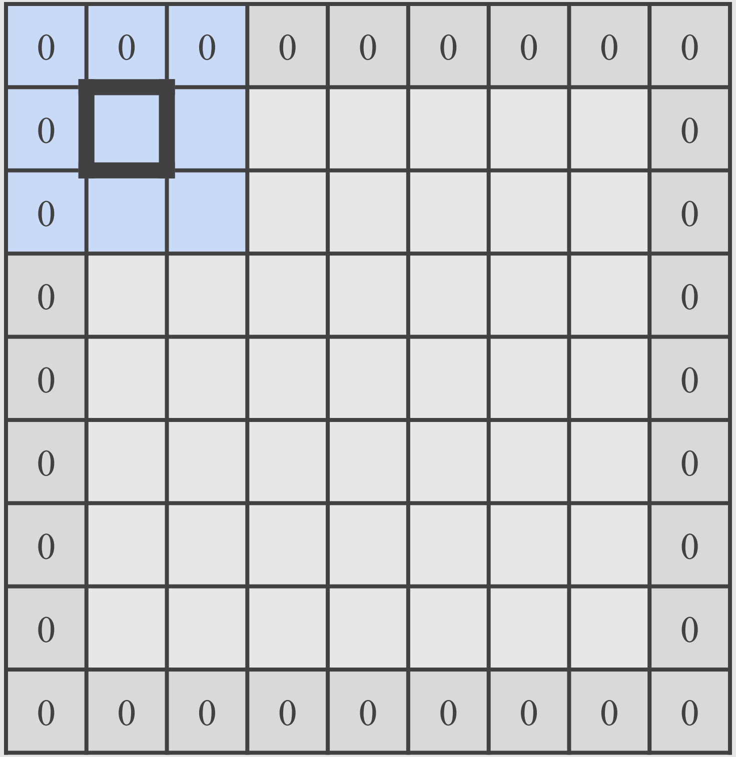
In Practice: Common to zero pad the border



7×7 input,
3×3 filter applied
with **stride 1 with pad 1**

What is the output?

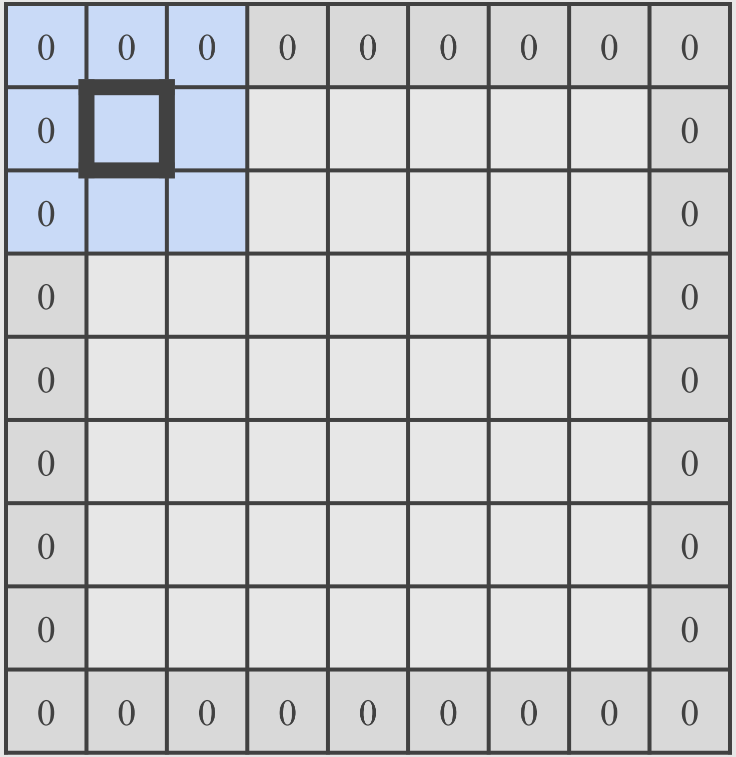
In Practice: Common to zero pad the border



7×7 input,
3×3 filter applied
with **stride 1 with pad 1**

What is the output?
7×7 output

In Practice: Common to zero pad the border



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)**.

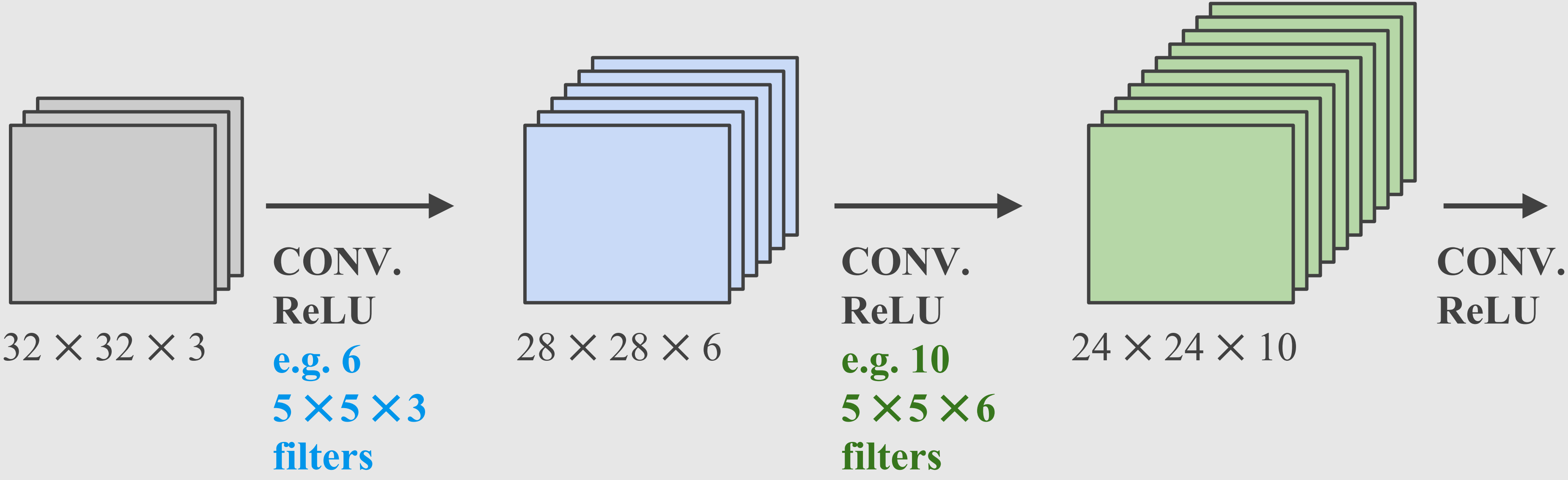
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Shrinking too fast is not good, doesn't work well.

$32 \rightarrow 28 \rightarrow 24 \rightarrow \ldots$



Number of Parameters

Input volume: $\mathbf{32 \times 32 \times 3}$

10 5×5 filters with stride 1, pad 2

Number of parameters in this layer?

Number of Parameters

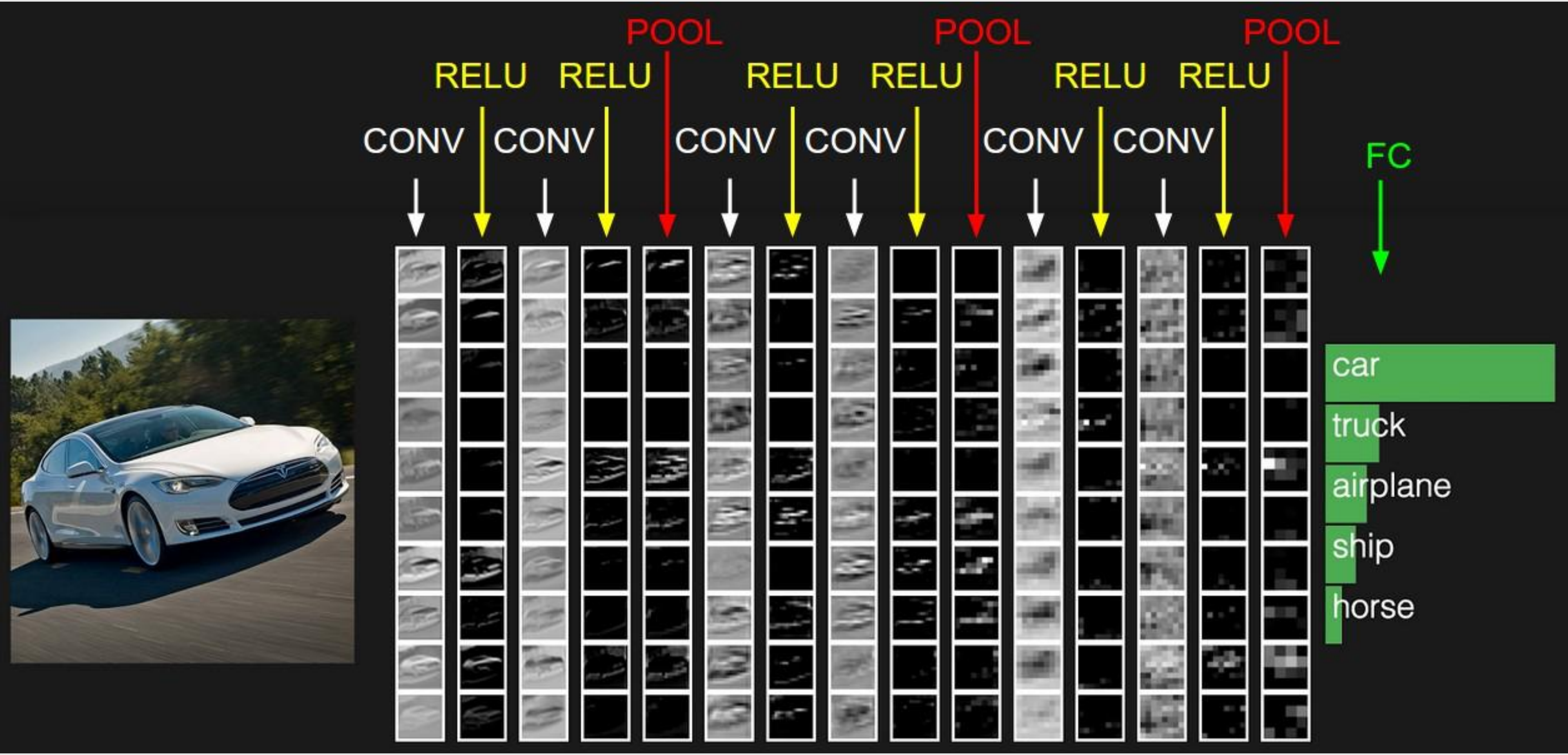
Input volume: $\mathbf{32 \times 32 \times 3}$

$\mathbf{10}$ $\mathbf{5 \times 5}$ filters with stride 1, pad 2

Number of parameters in this layer?

Each filter has $5*5*3 + 1 = \mathbf{76}$ parameters (+1 for bias)

$\Rightarrow \mathbf{76}*\mathbf{10} = 760$



Credit: http://cs231n.github.io/convolutional-networks/

Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently

Pooling Layer

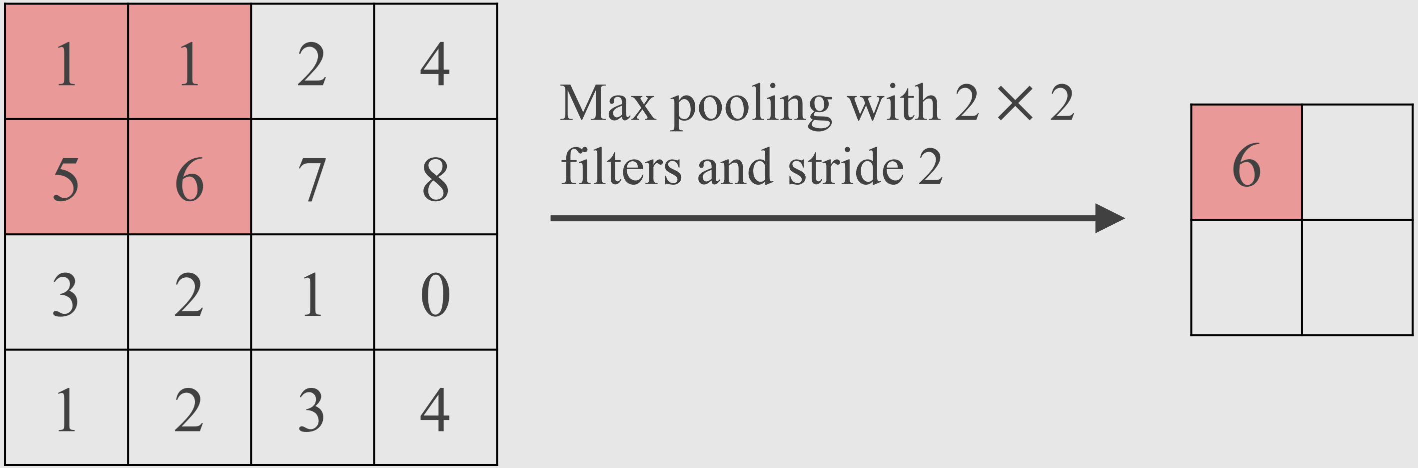
- Makes the representations smaller and more manageable
- Operates over each activation map independently

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

Max pooling with 2×2 filters and stride 2

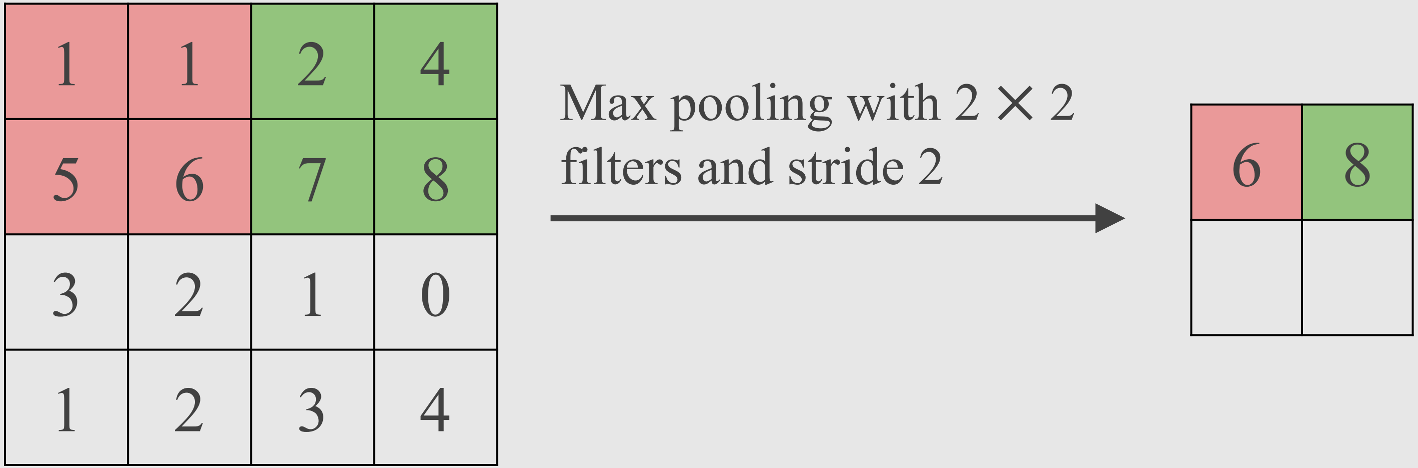
Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



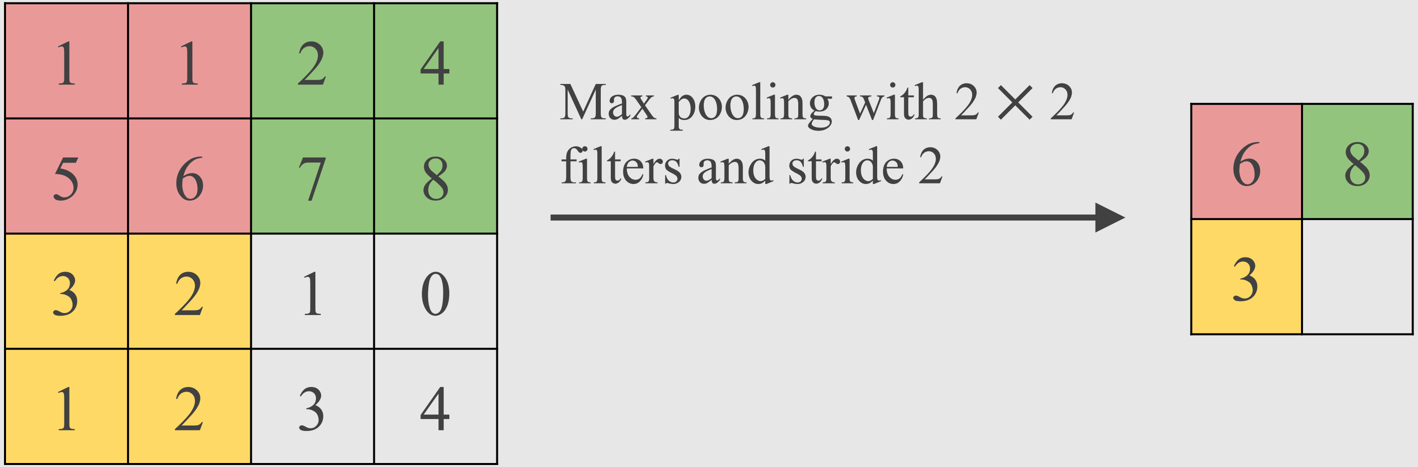
Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



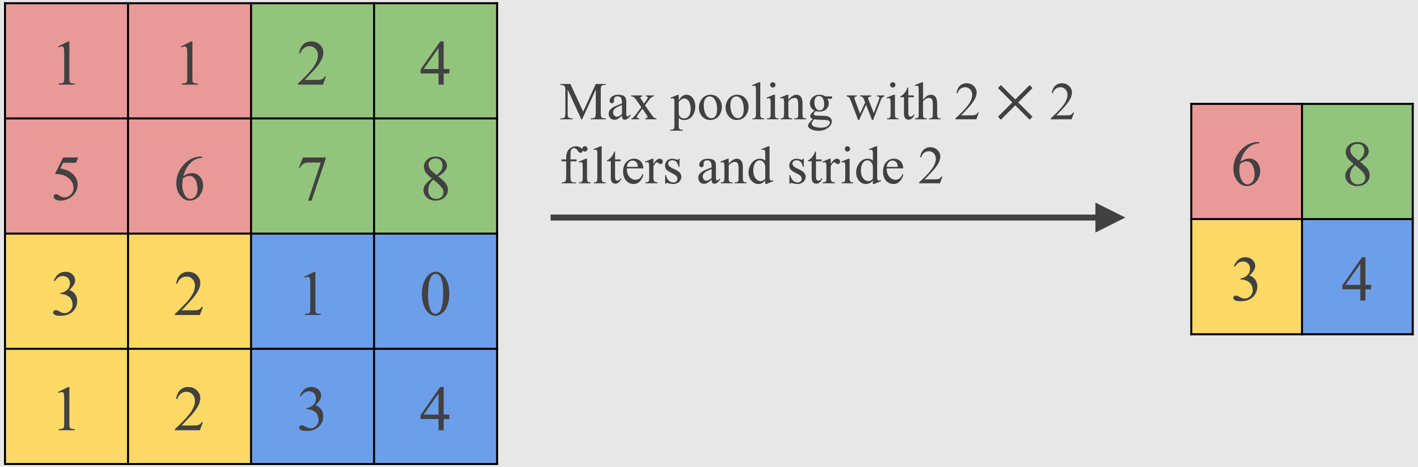
Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



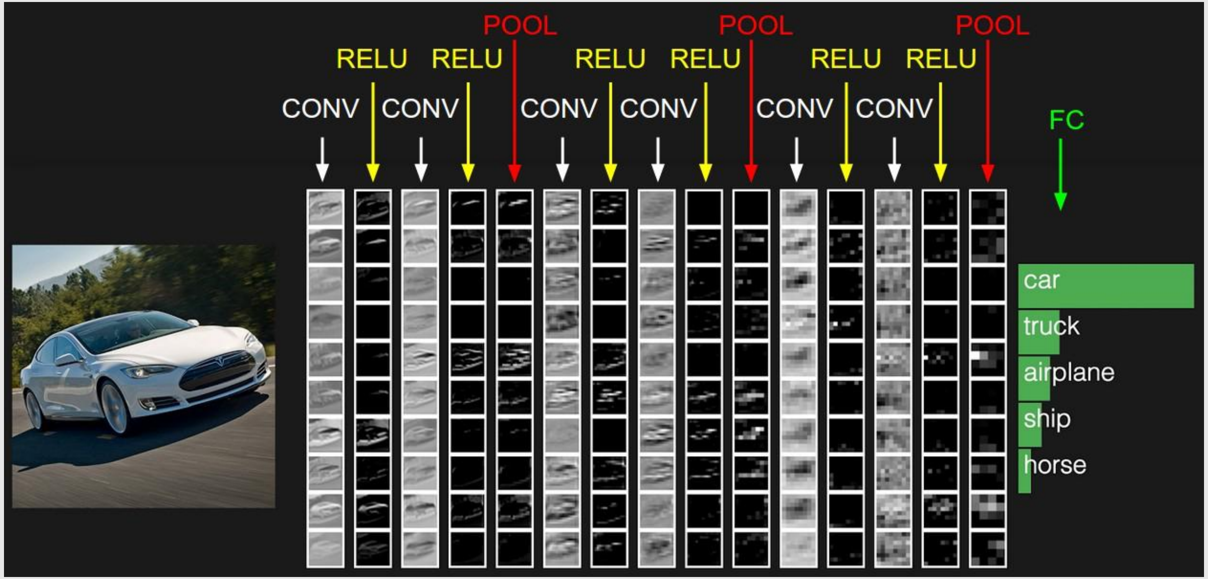
Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



Fully Connected Layer

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



Credit: http://cs231n.github.io/convolutional-networks/

http://neuralnetworksanddeeplearning.com/chap6.html#final_conv

neuralnetworksanddeeplearning.com/chap6.html#final_conv

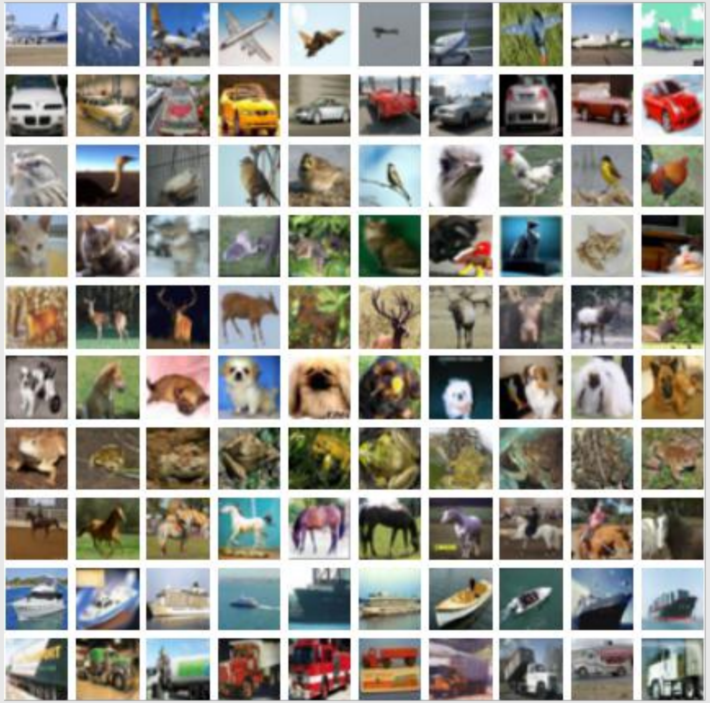
Search

Convolutional neural networks in practice

We've now seen the core ideas behind convolutional neural networks. Let's look at how they work in practice, by implementing some convolutional networks, and applying them to the MNIST digit classification problem. The program we'll use to do this is called `network3.py`, and it's an improved version of the programs `network.py` and `network2.py` developed in earlier chapters*. If you wish to follow along, the code is available on GitHub. Note that we'll work through the code for `network3.py` itself in the next section. In this section, we'll use `network3.py` as a library to build convolutional networks.

*Note also that `network3.py` incorporates ideas from the Theano library's documentation on convolutional neural nets (notably the implementation of LeNet-5), from Misha Denil's implementation of dropout, and from Chris Olah.

ConvNetJS demo: Training on CIFAR-10



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.htm

DNNs Architectures

DNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

To be continued ...

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 11 & 13

Machine Learning Courses

- https://www.coursera.org/learn/neural-networks
- “The 3 popular courses on Deep Learning”: https://medium.com/towards-data-science/the-3-popular-courses-for-deeplearning-ai-ac37d4433bd