Convolutional Neural Networks

CSE 151 Winter 2019

Taylor Berg-Kirkpatrick

Slide credits: Fei-Fei Li & Justin Johnson & Serena Yeung

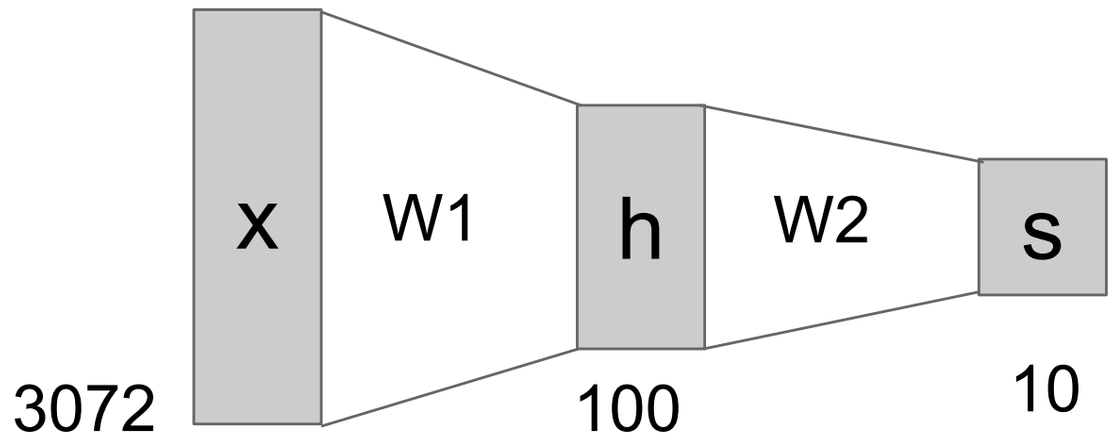
Last time: Neural Networks

Linear score function: $f = Wx$

Last time: Neural Networks

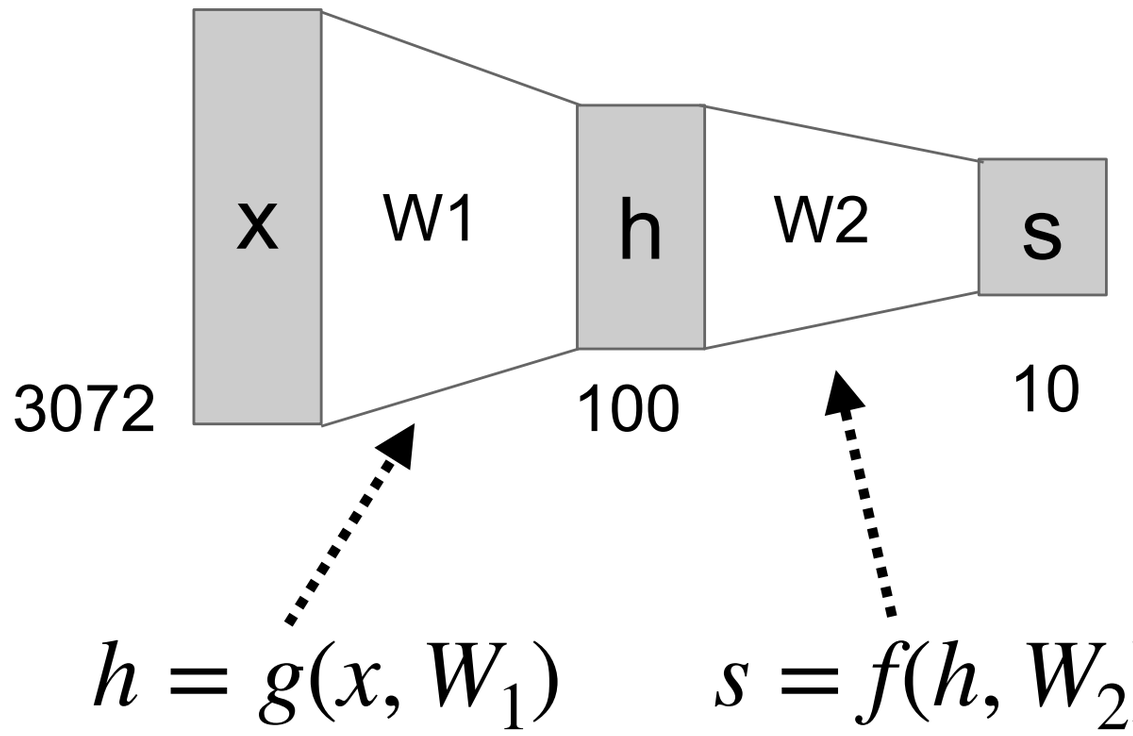
Linear score function: $f = Wx$

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



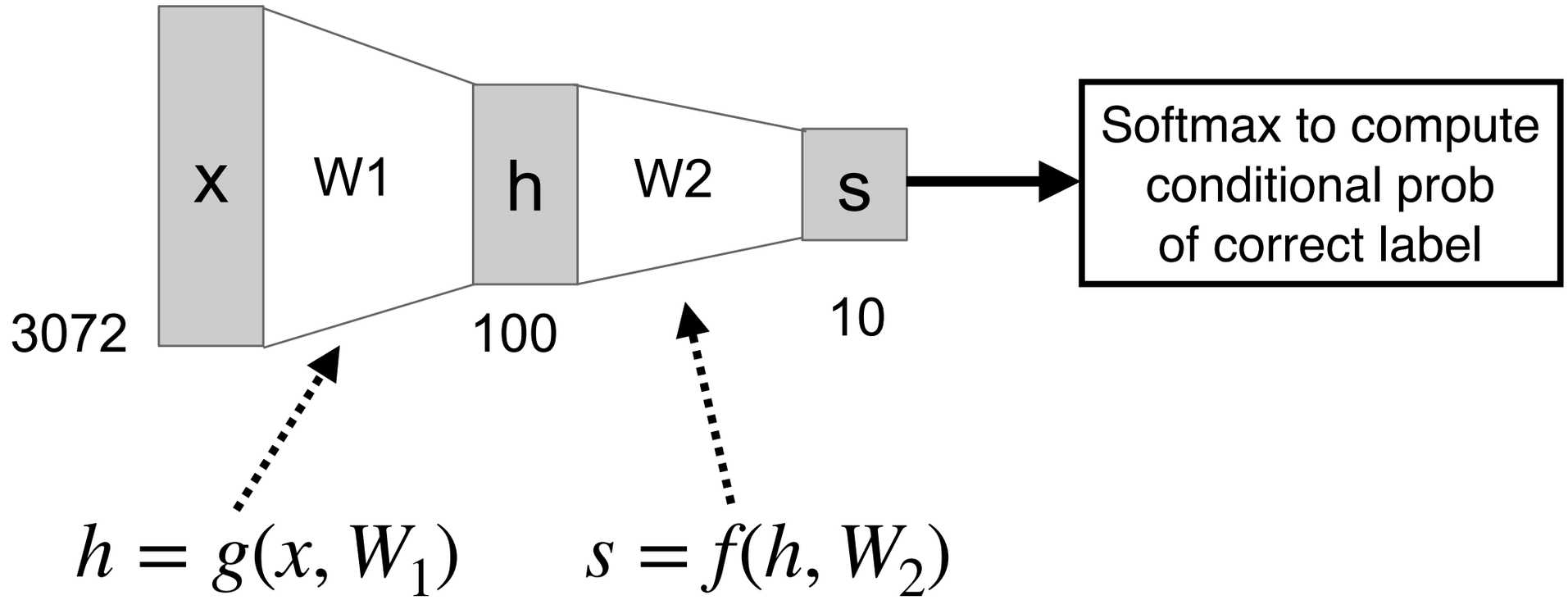
Last time: Neural Networks

2-layer Neural Network



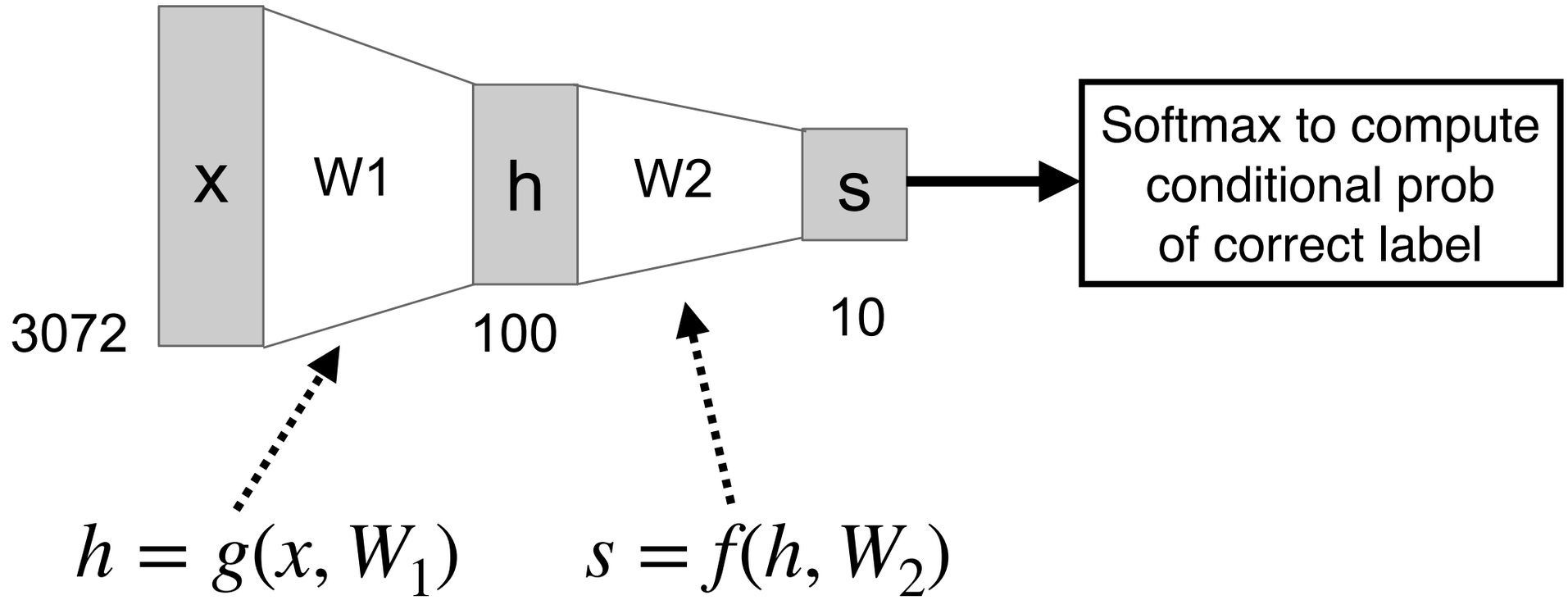
Last time: Neural Networks

2-layer Neural Network



Last time: Neural Networks

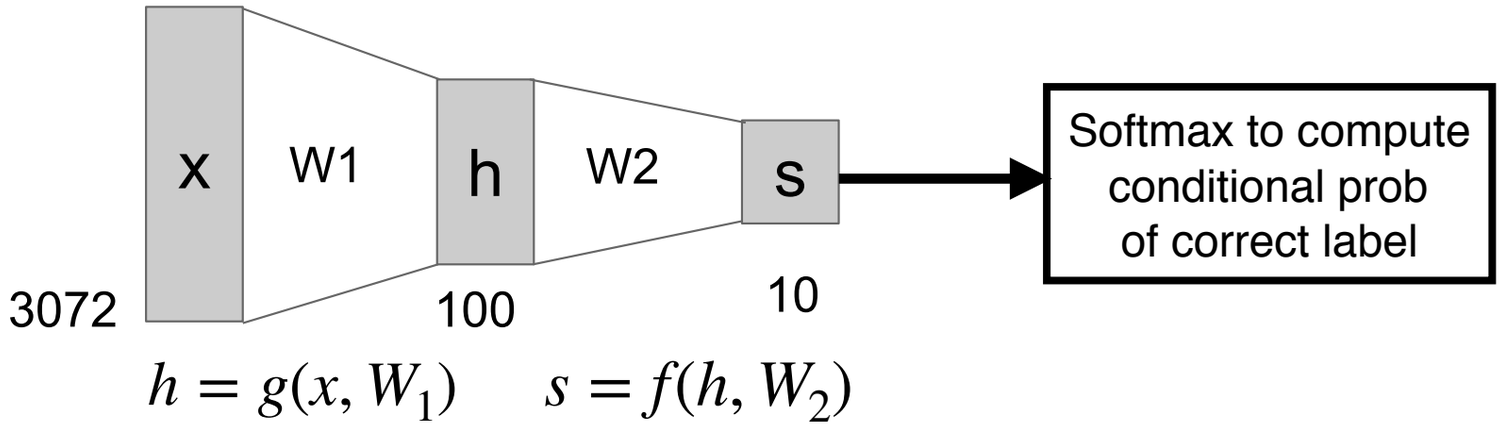
2-layer Neural Network



Training loss:

$$L(W_1, W_2) = \ell\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)$$

2-layer Neural Network

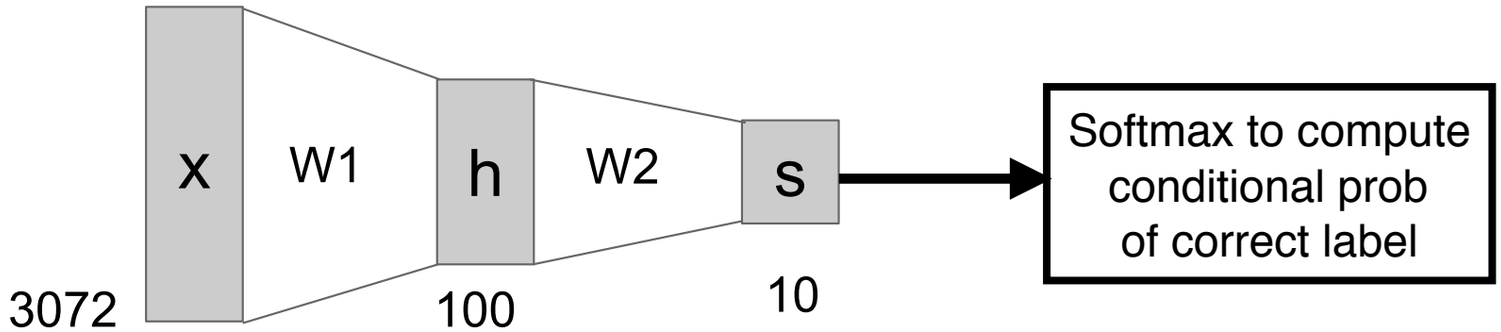


$$h = g(x, W_1) \qquad s = f(h, W_2)$$

Backpropagation (i.e. computing gradient via chain rule):

$$L(W_1, W_2) = \ell\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)$$

2-layer Neural Network



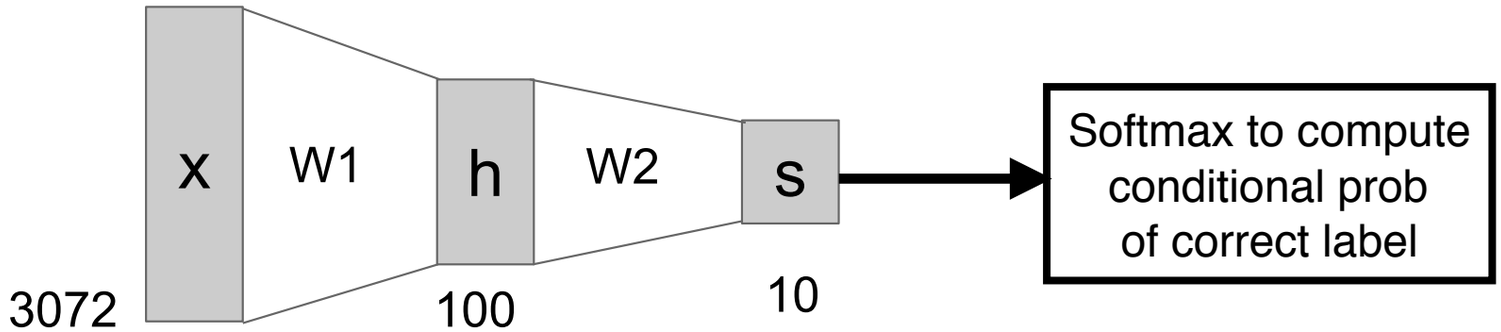
$$h = g(x, W_1) \qquad s = f(h, W_2)$$

Backpropagation (i.e. computing gradient via chain rule):

$$L(W_1, W_2) = \ell\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)$$

$$\nabla_{W_2} L(W_1, W_2) = \frac{\delta \ell}{\delta f}\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)\frac{\delta f}{\delta W_2}\Big(g\big(x, W_1\big), W_2\Big)$$

2-layer Neural Network



$$h = g(x, W_1) \qquad s = f(h, W_2)$$

Backpropagation (i.e. computing gradient via chain rule):

$$L(W_1, W_2) = \ell\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)$$

$$\nabla_{W_2} L(W_1, W_2) = \frac{\delta \ell}{\delta f}\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)\frac{\delta f}{\delta W_2}\Big(g\big(x, W_1\big), W_2\Big)$$

$$\nabla_{W_1} L(W_1, W_2) = \frac{\delta \ell}{\delta f}\Big(f\Big(g\big(x, W_1\big), W_2\Big)\Big)\frac{\delta f}{\delta g}\Big(g\big(x, W_1\big), W_2\Big)\frac{\delta g}{\delta W_1}\big(x, W_1\big)$$

Next: Convolutional Neural Networks

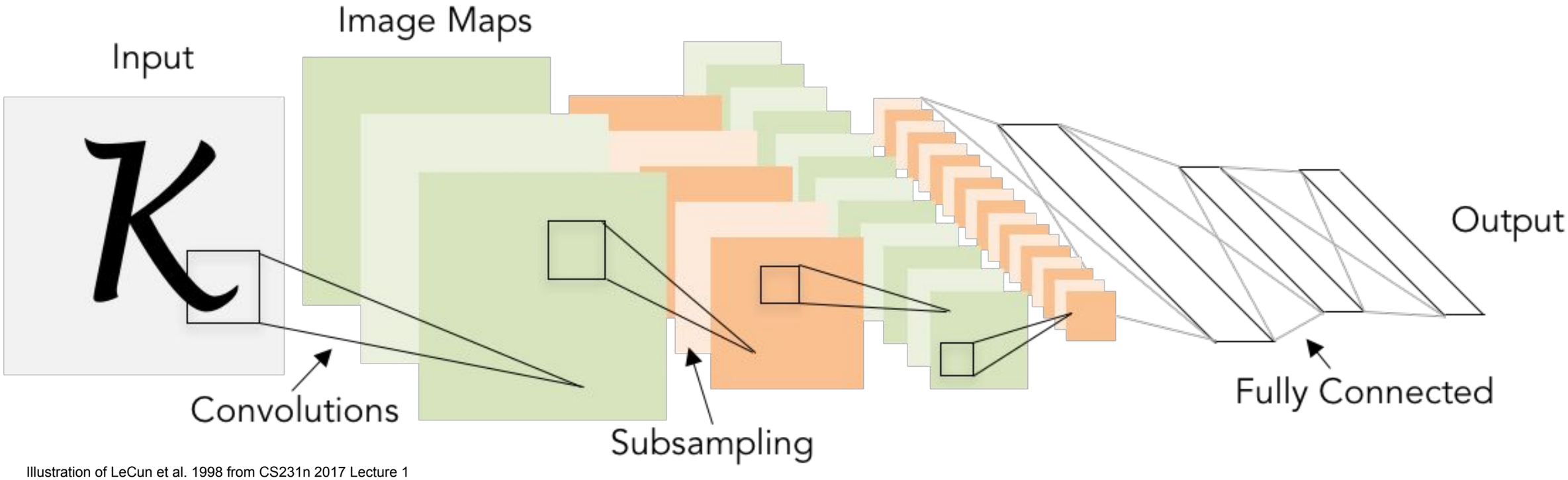


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Change the form of these functions (overall loss equation unchanged)

$$L(W_1, W_2) = \ell\left(f\left(g\left(x, W_1\right), W_2\right)\right)$$

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

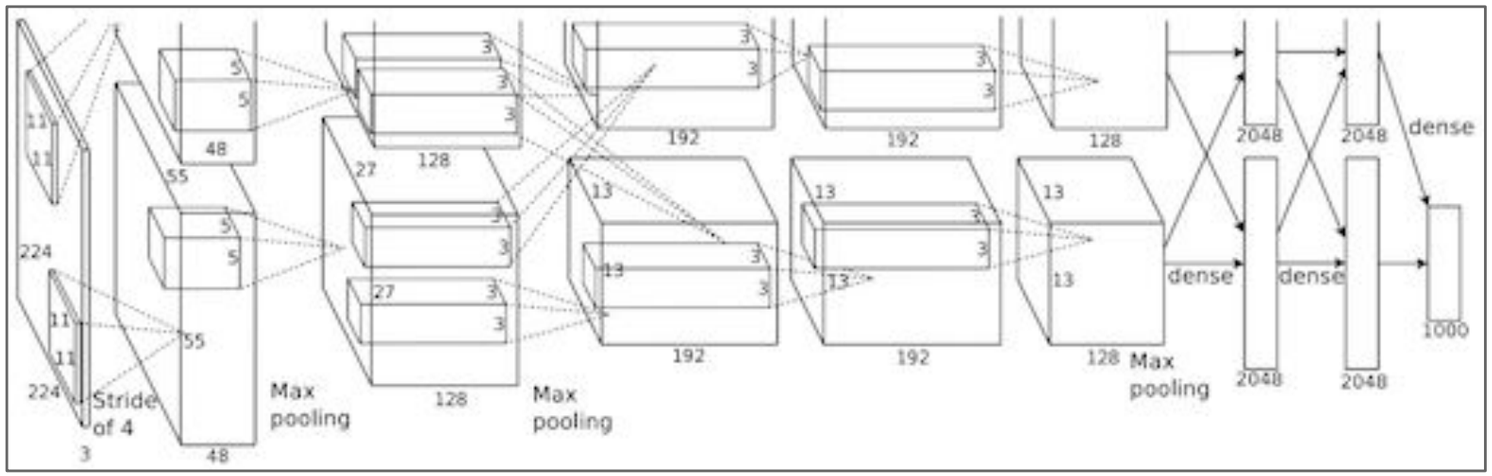


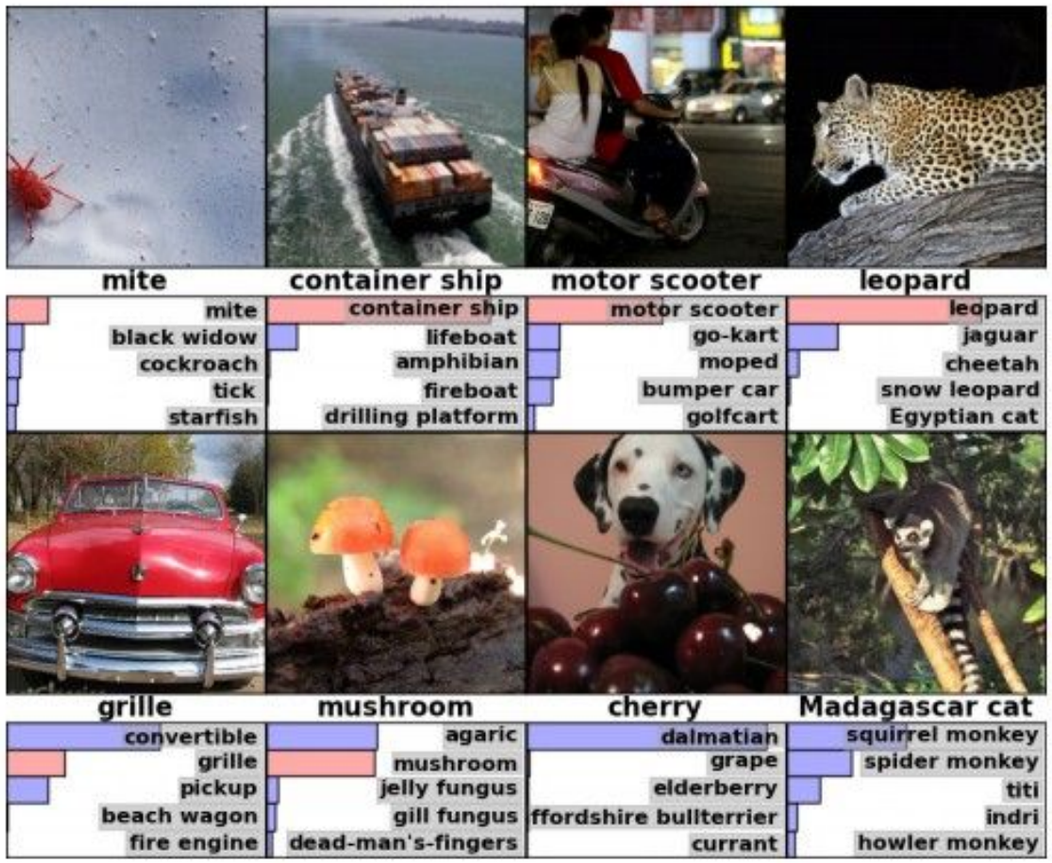
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

Fast-forward to today: ConvNets are everywhere

Classification

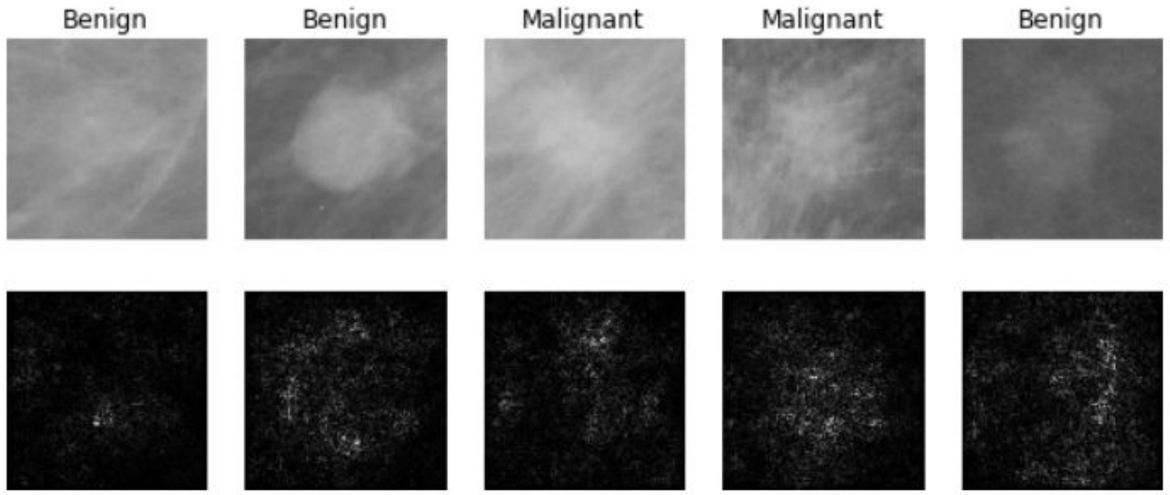
Retrieval





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

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

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



[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

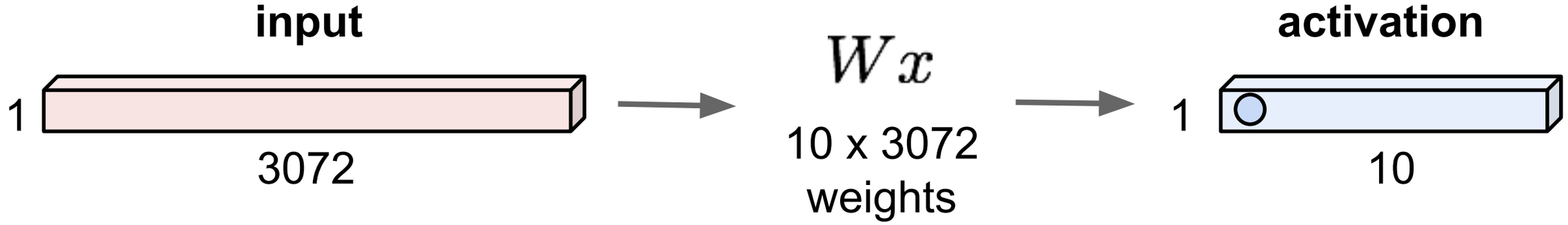


[Sermanet et al. 2011]
[Ciresan et al.]

Photos by Lane McIntosh.
Copyright CS231n 2017.

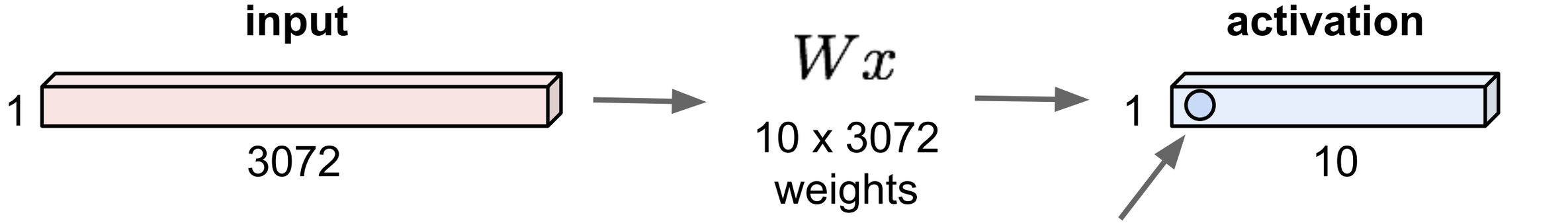
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



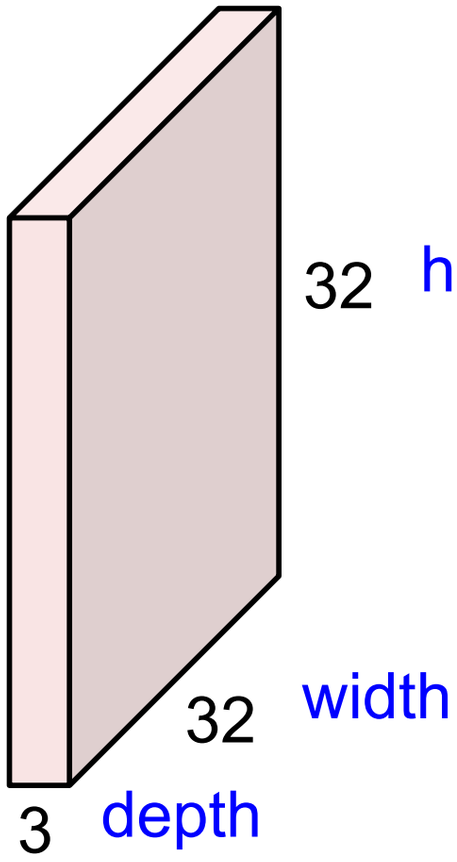
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



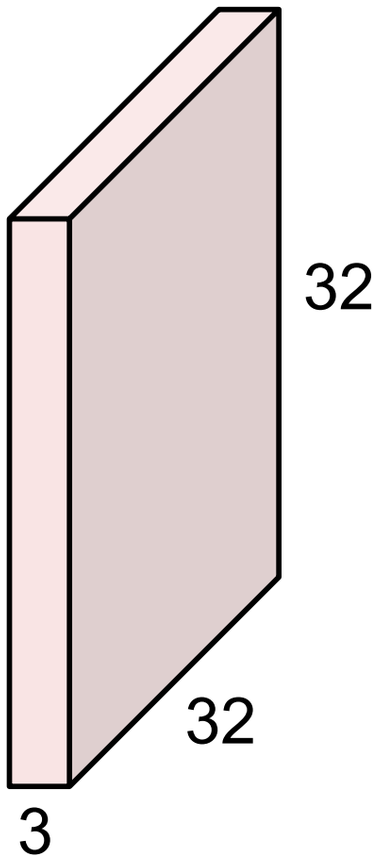
Convolution Layer

32x32x3 image -> preserve spatial structure

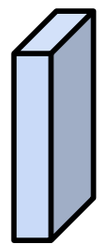


Convolution Layer

32x32x3 image



5x5x3 filter

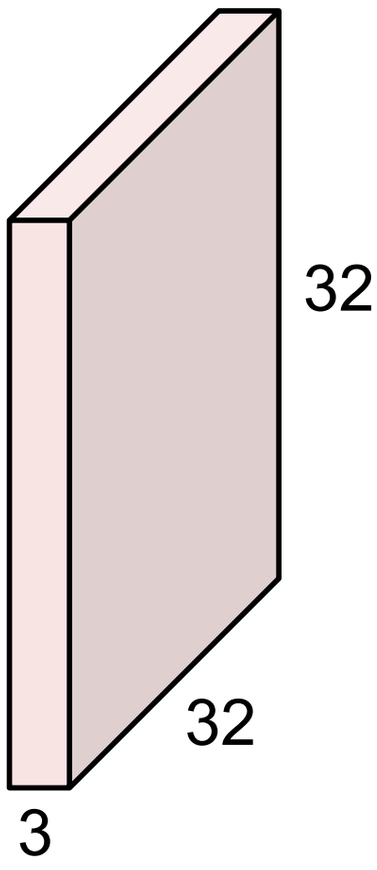


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

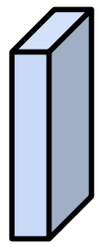
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

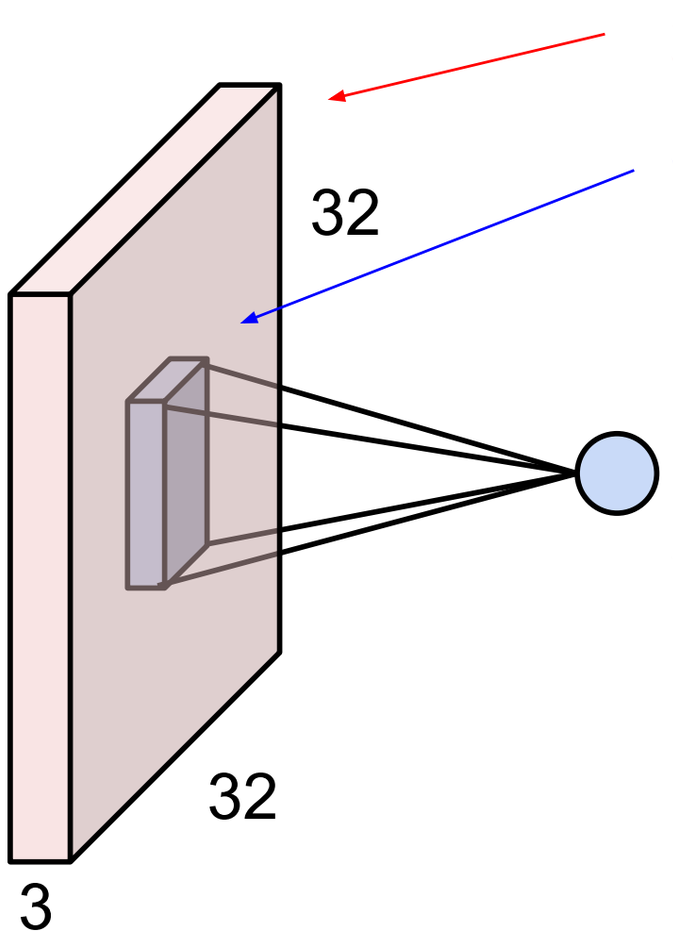


5x5x3 filter



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

Convolution Layer



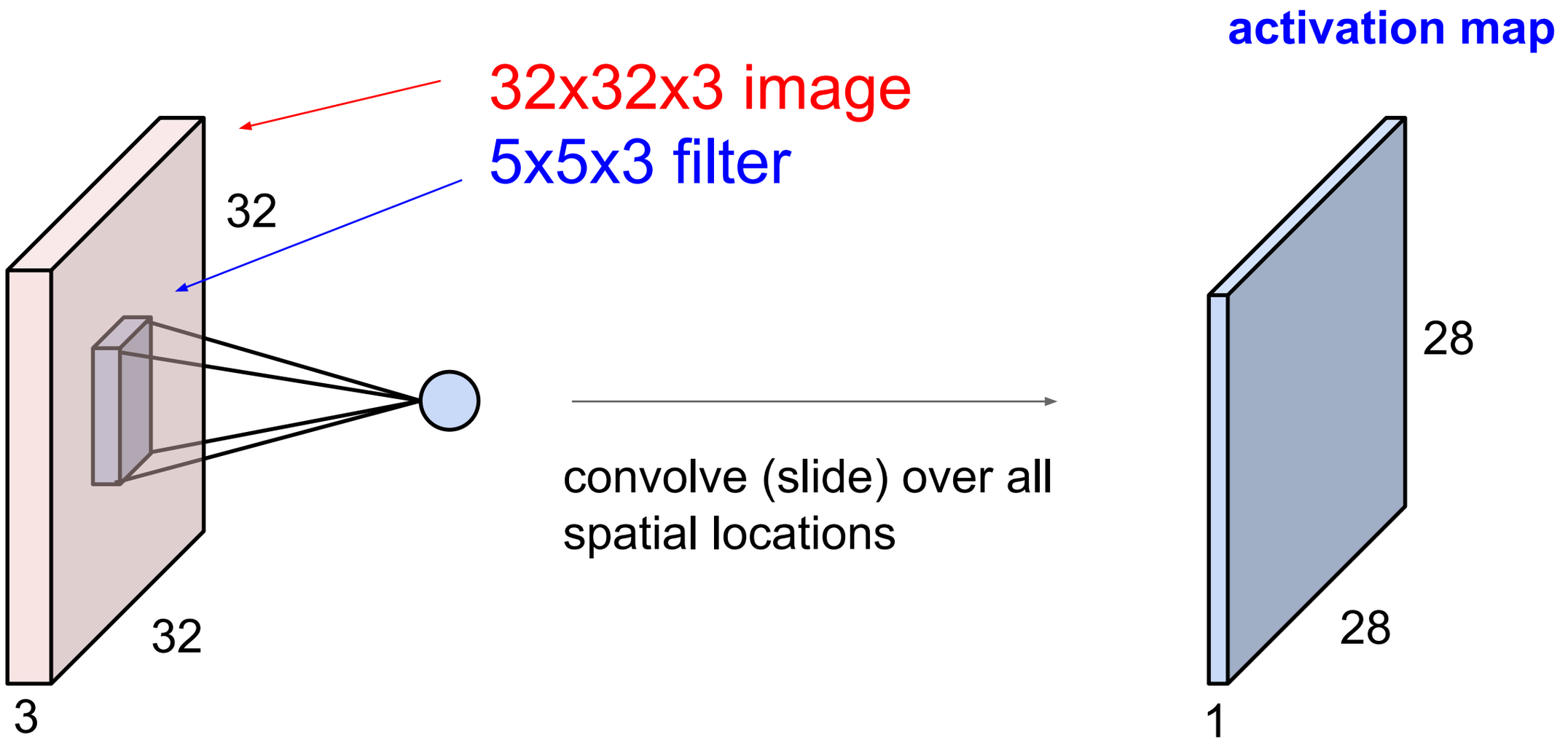
32x32x3 image

5x5x3 filter w

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

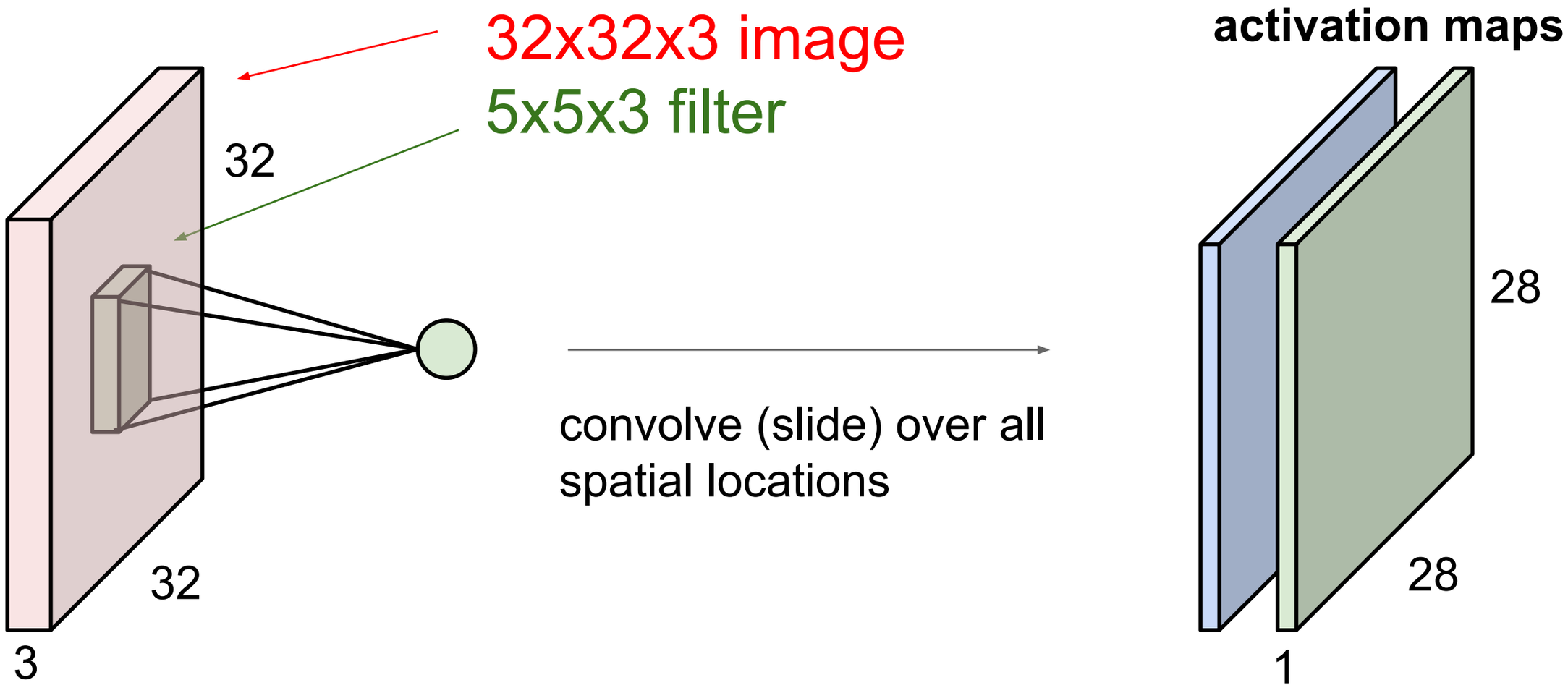
$$w^T x + b$$

Convolution Layer

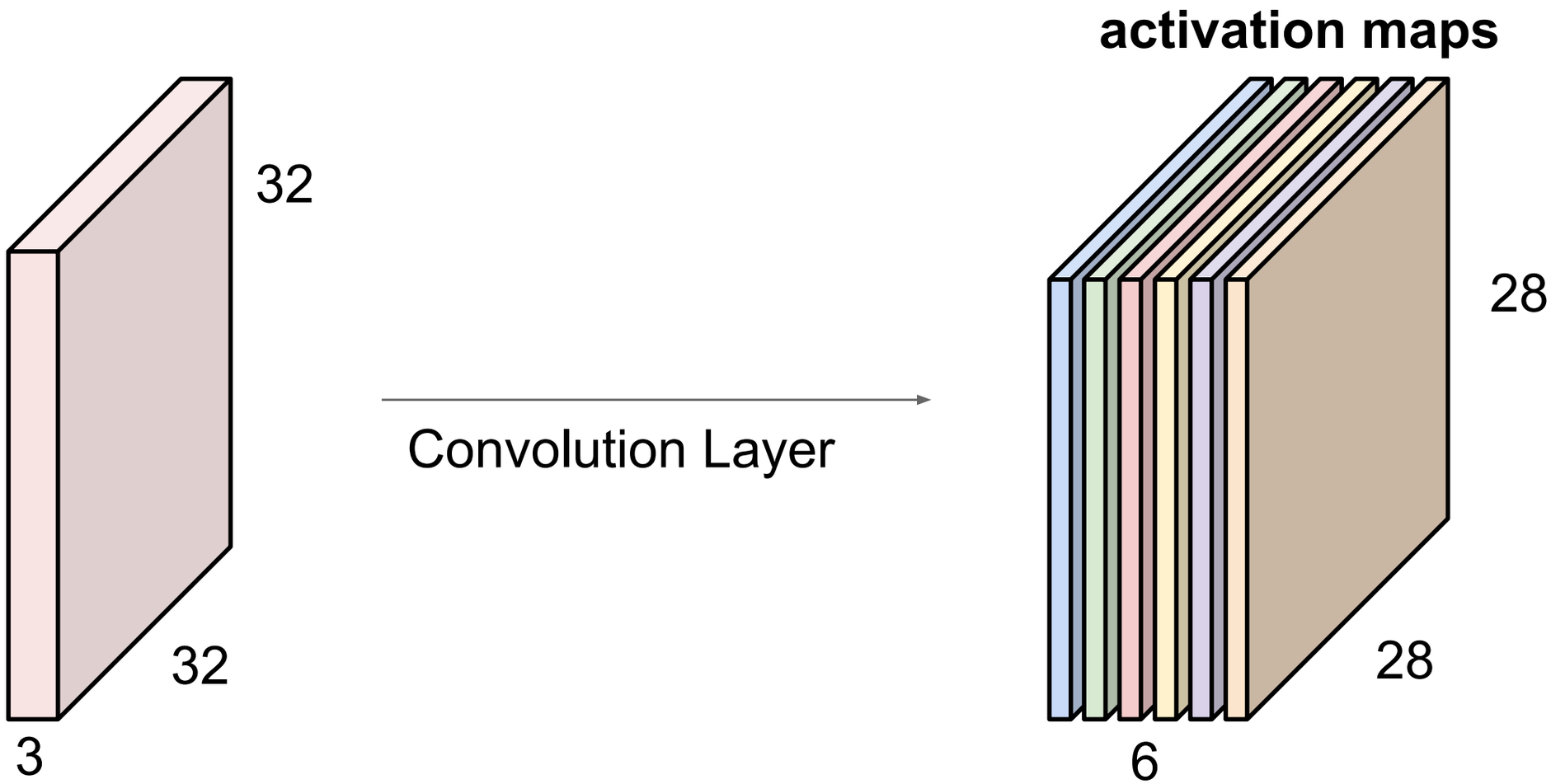


Convolution Layer

consider a second, green filter

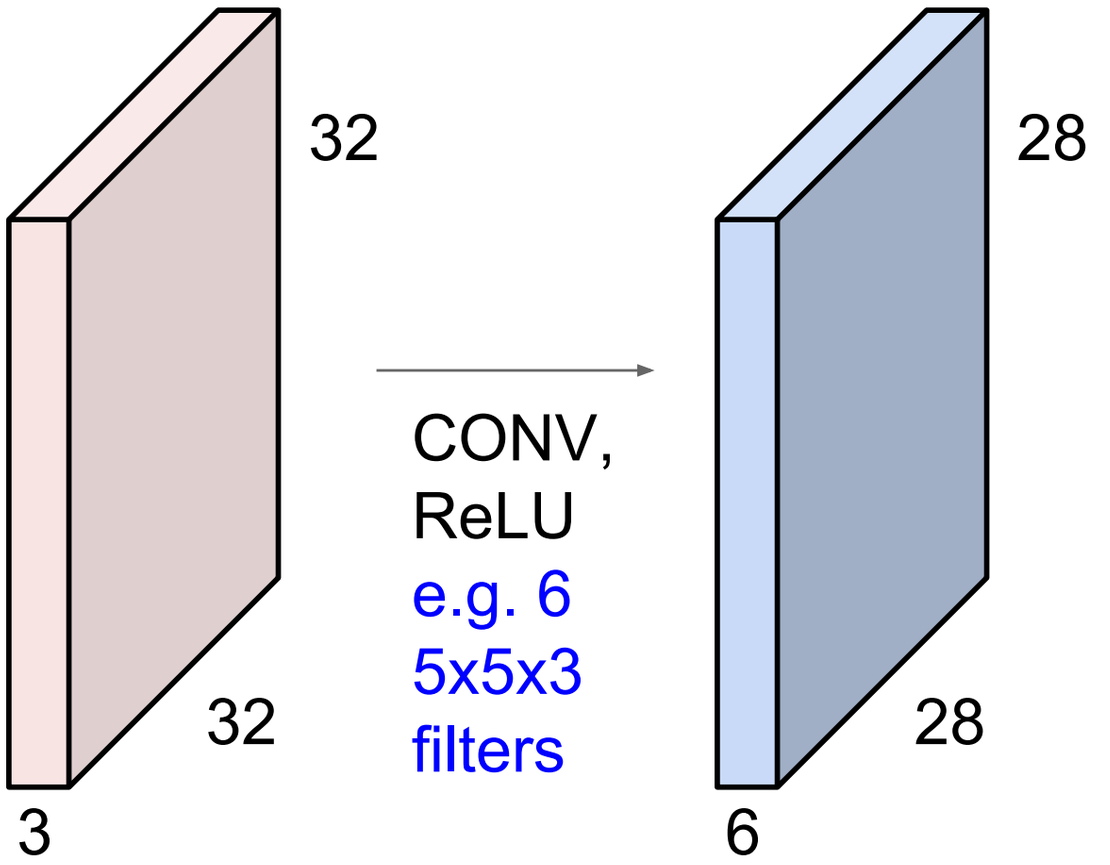


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

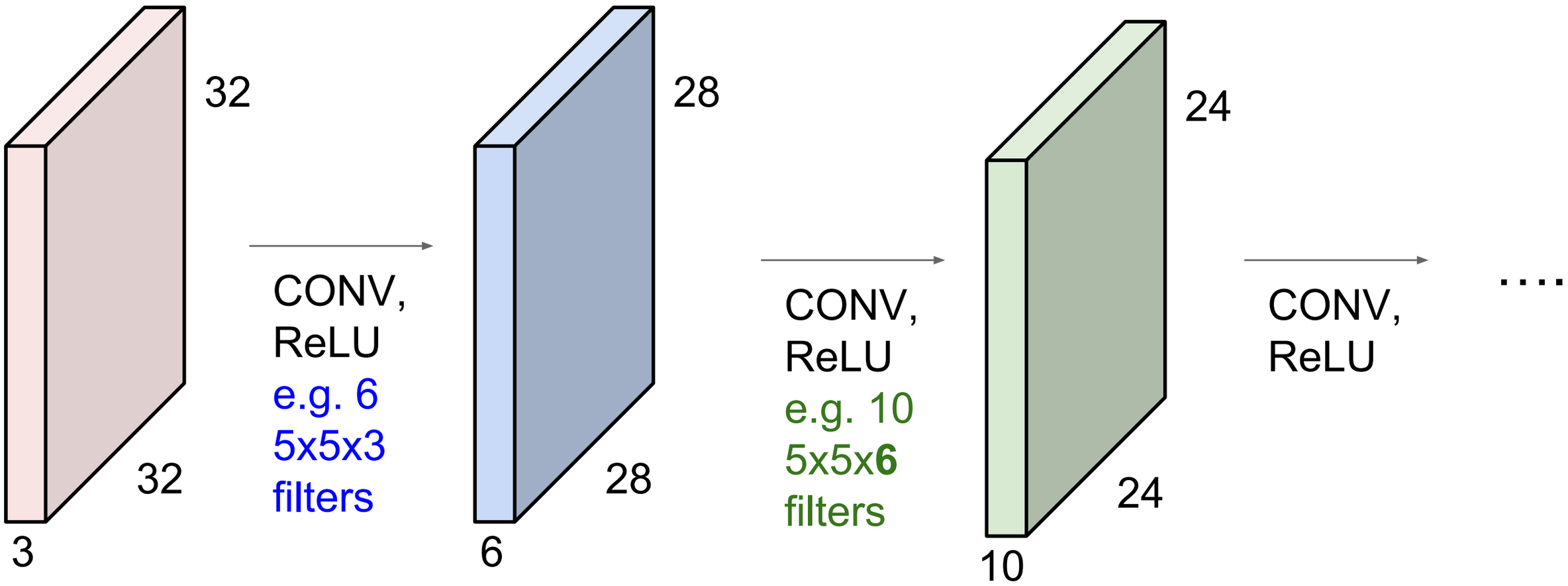


We stack these up to get a "new image" of size 28x28x6!

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



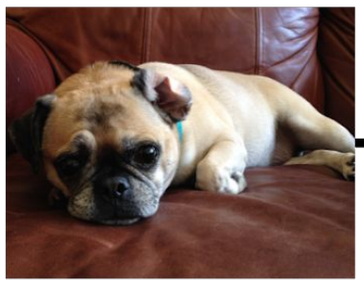
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview

[Zeiler and Fergus 2013]

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

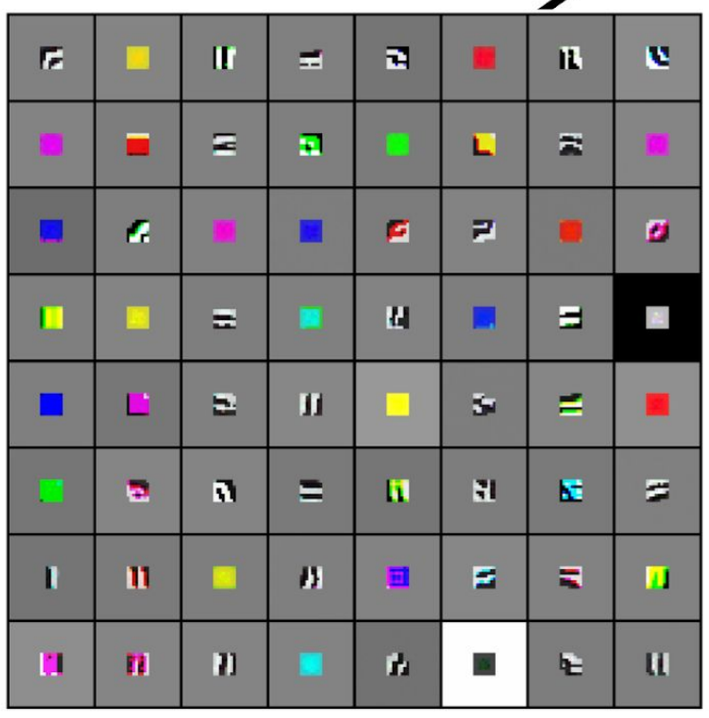


Low-level features

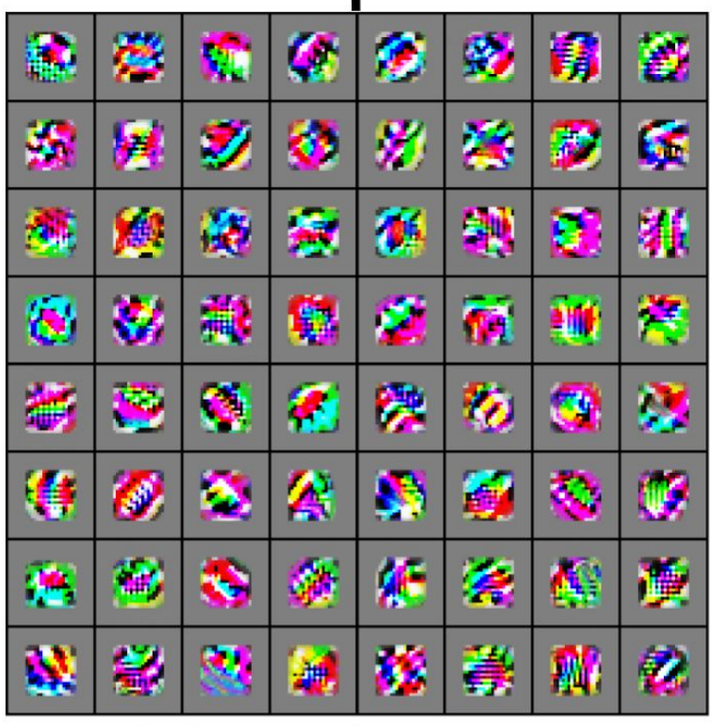
Mid-level features

High-level features

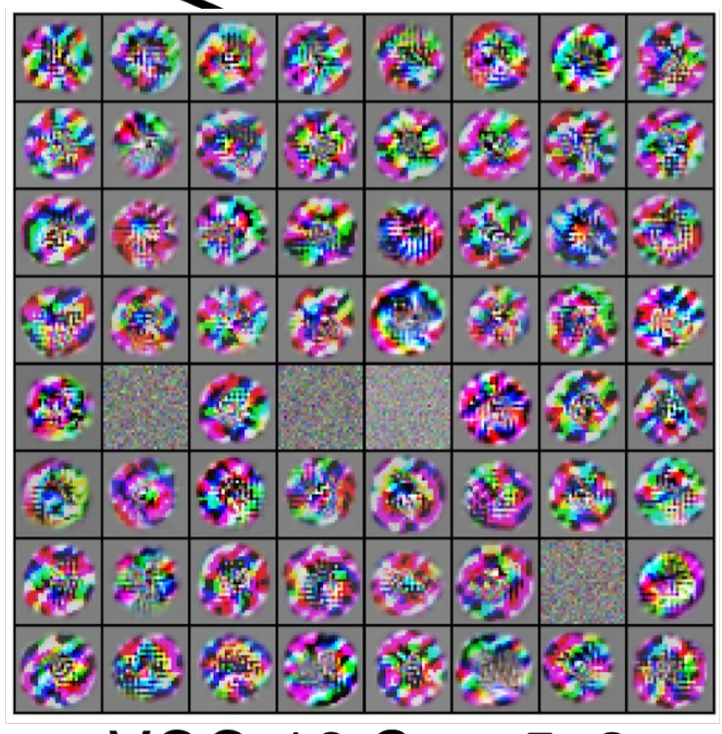
Linearly separable classifier



VGG-16 Conv1_1

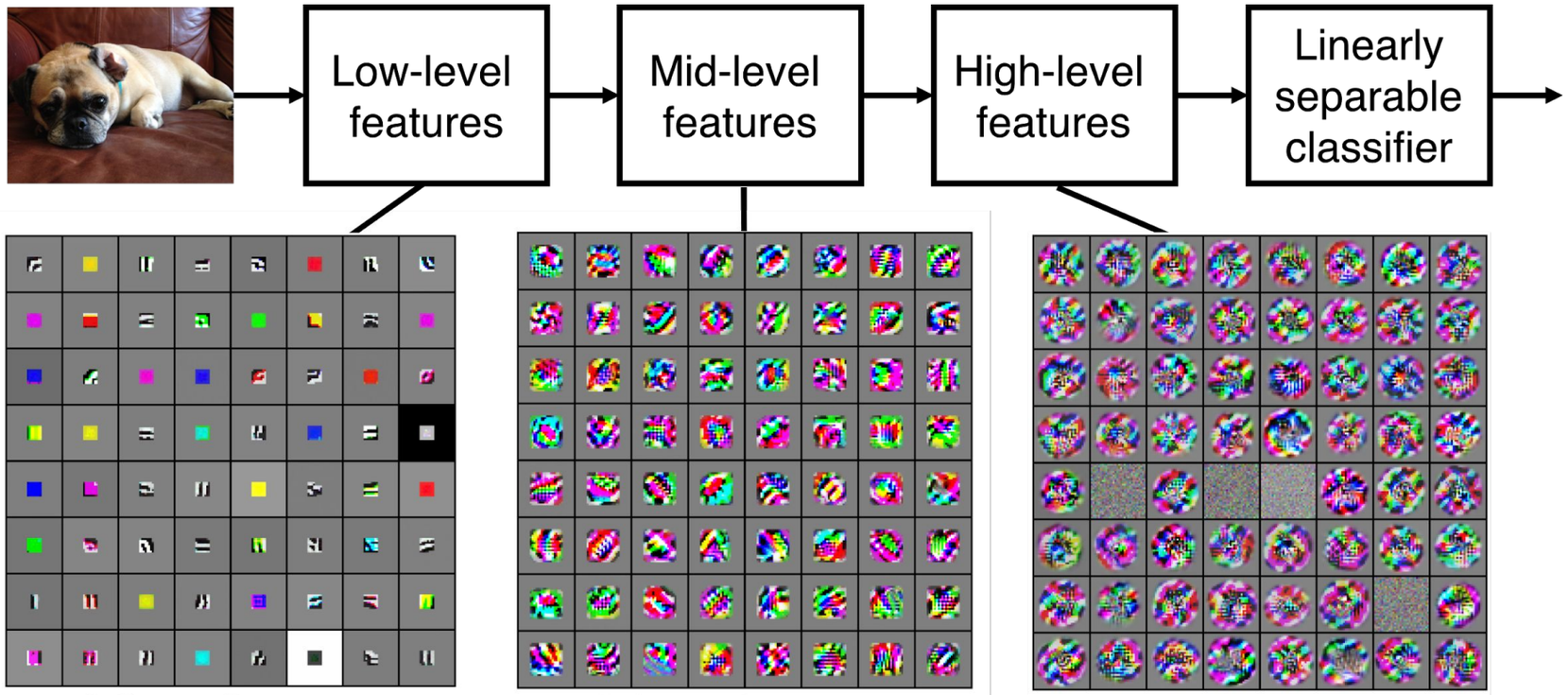


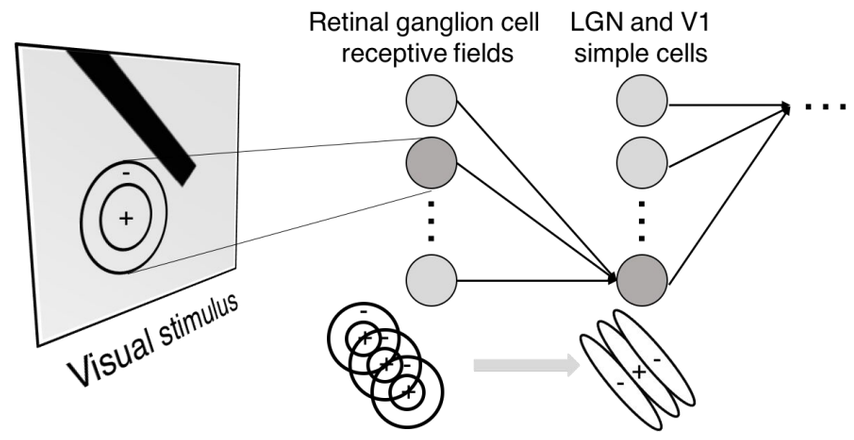
VGG-16 Conv3_2



VGG-16 Conv5_3

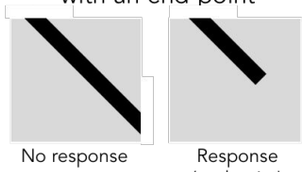
Preview

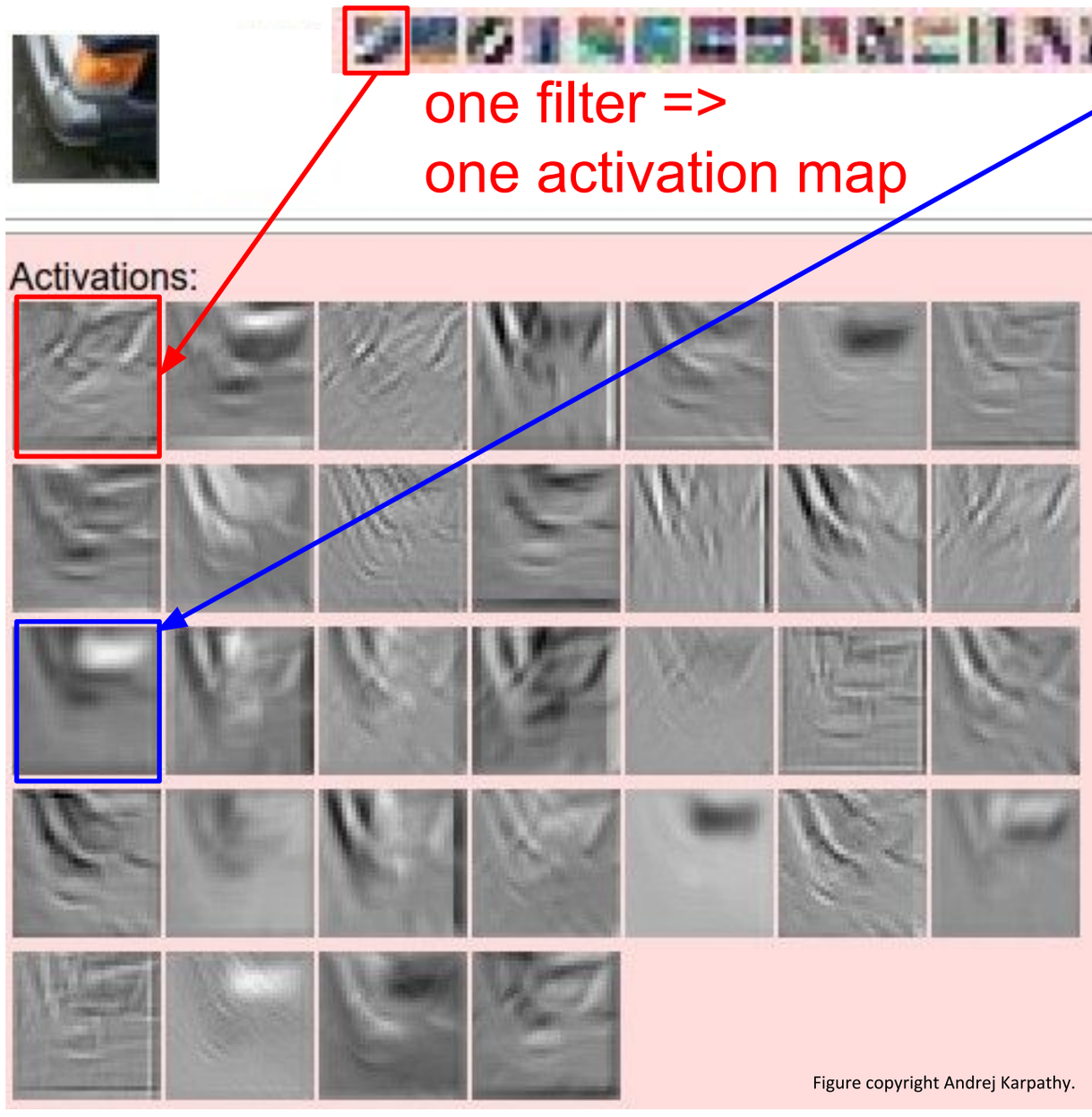




Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point





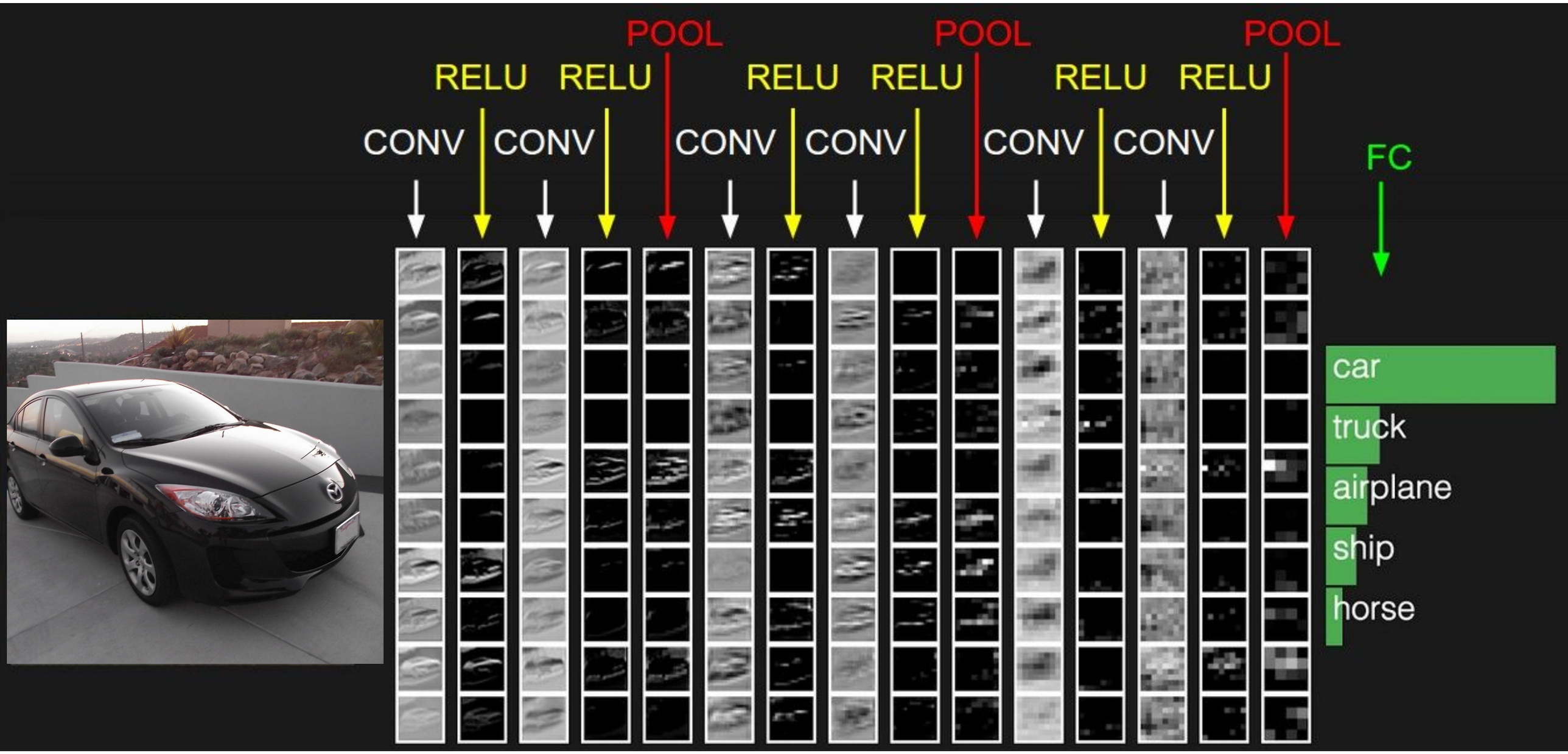
example 5x5 filters
(32 total)

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

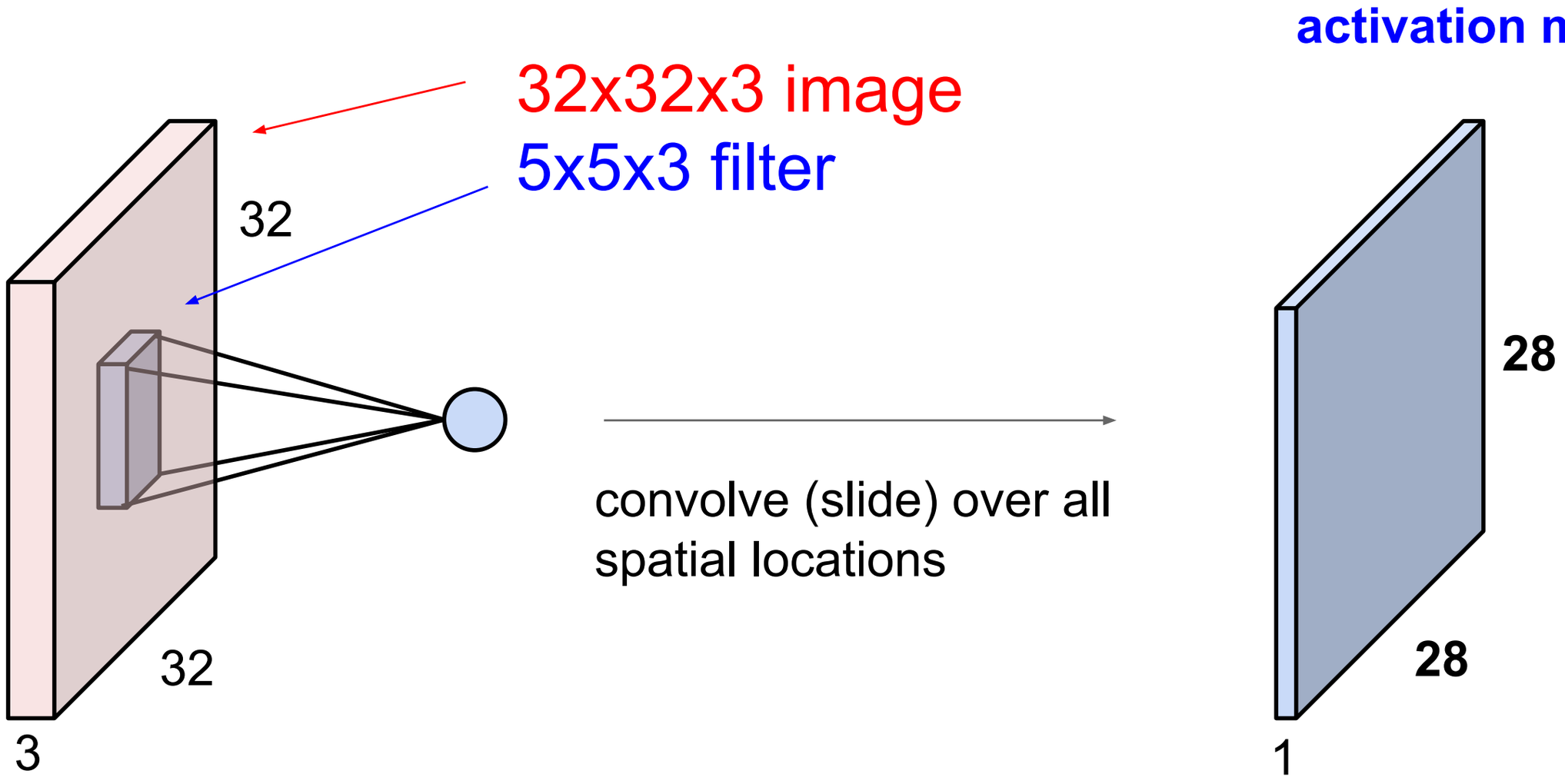
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

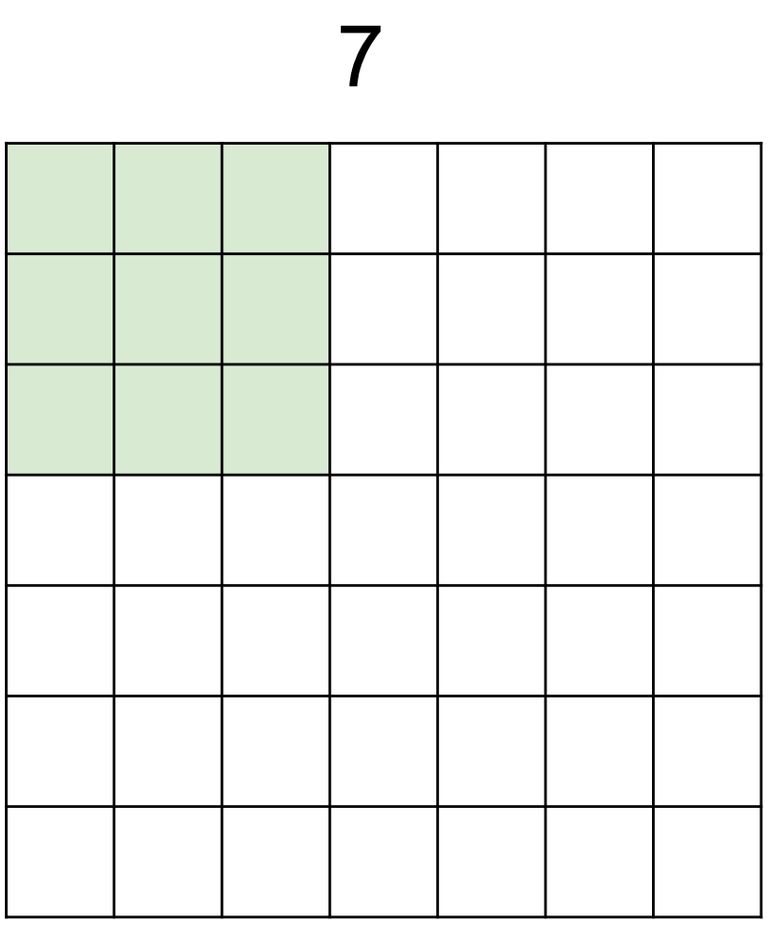
preview:


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:

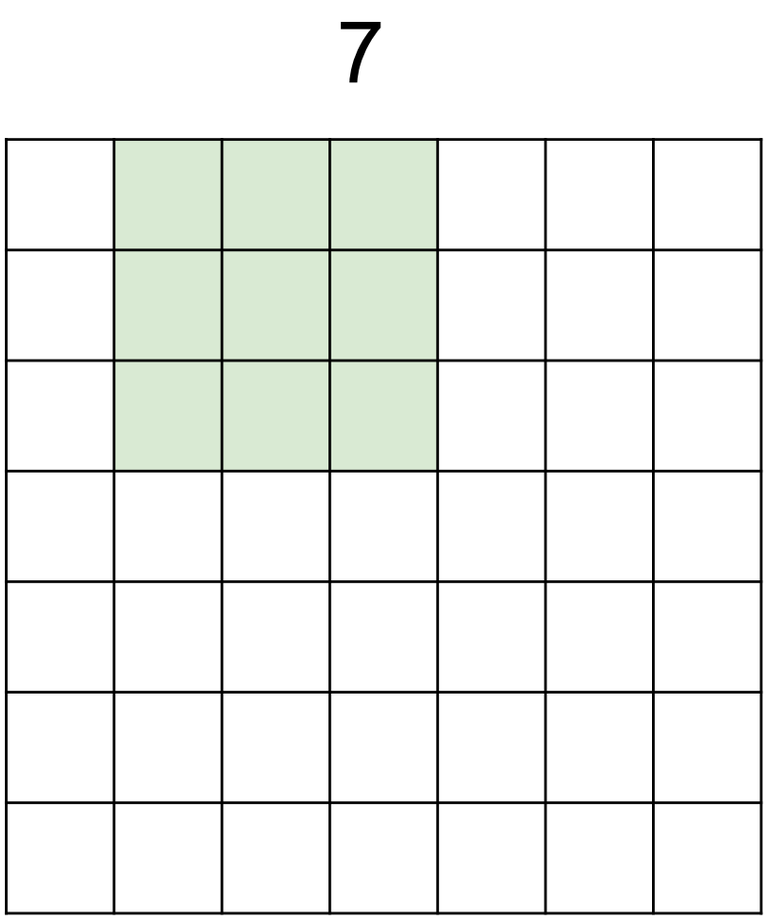


A closer look at spatial dimensions:



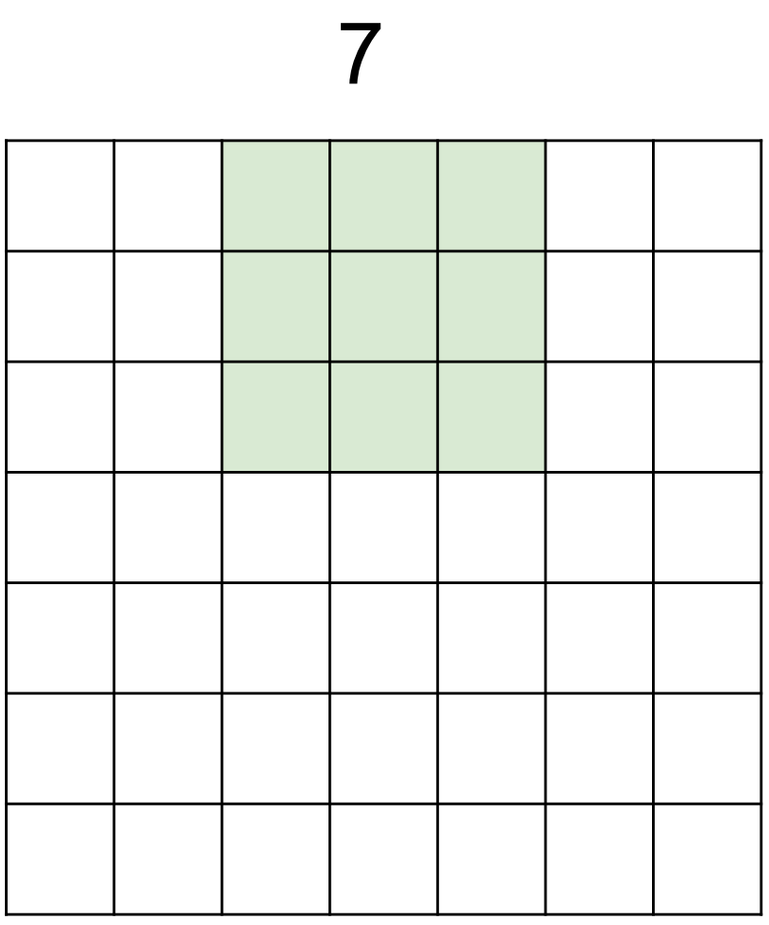
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



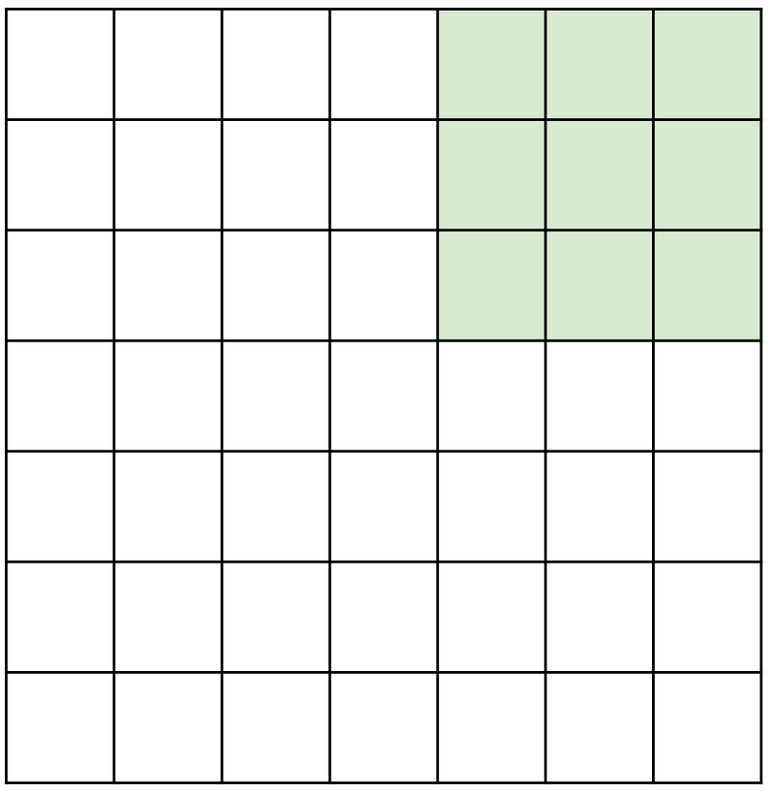
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

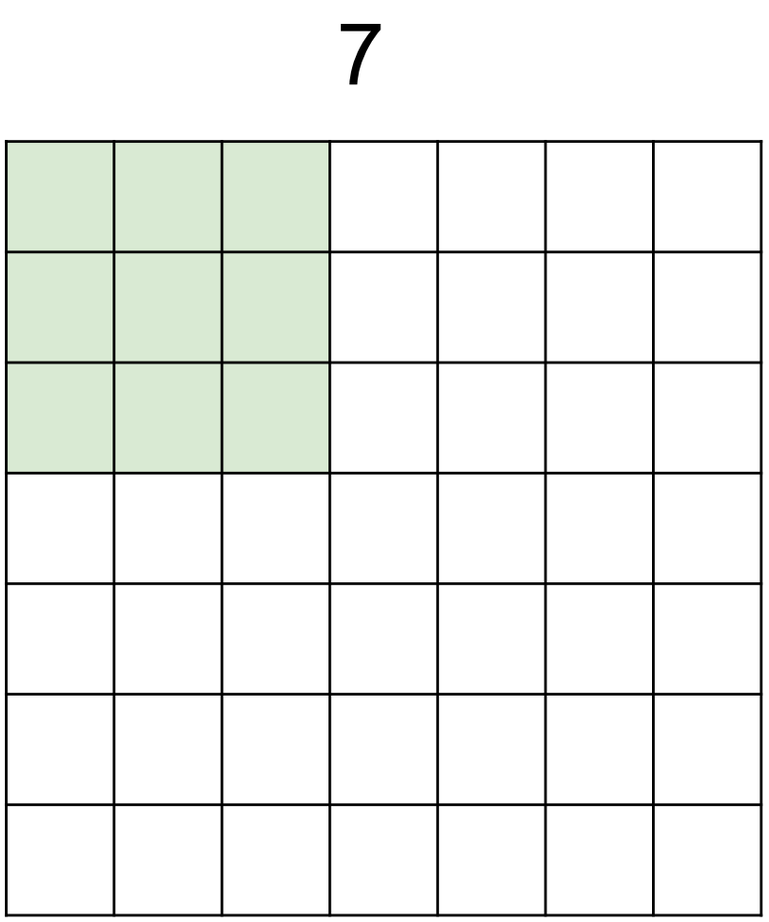
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

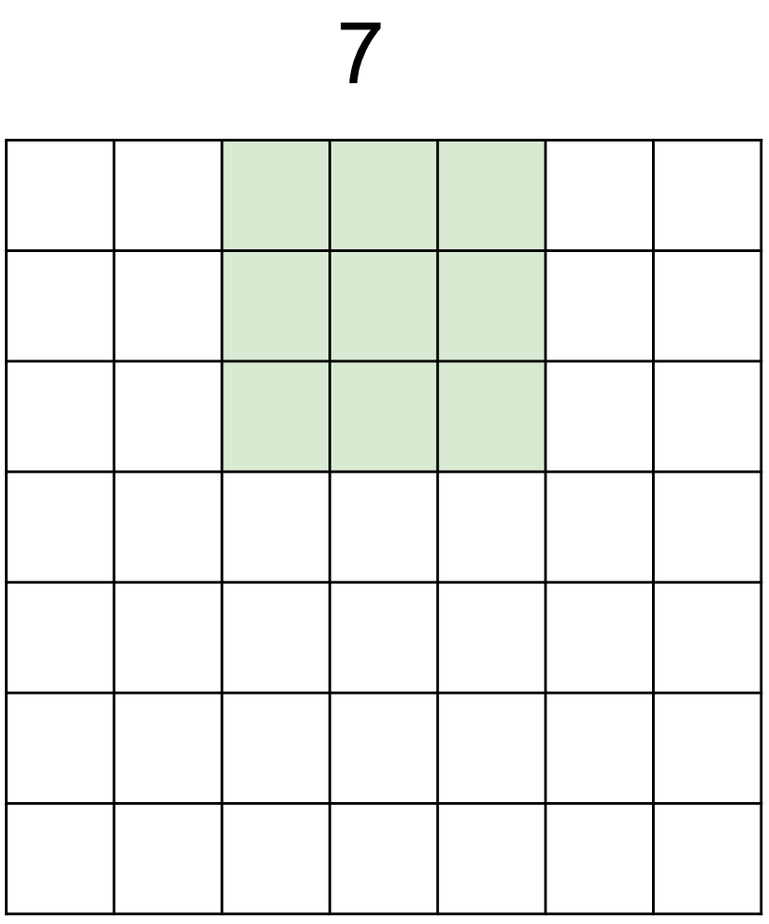
=> 5x5 output

A closer look at spatial dimensions:



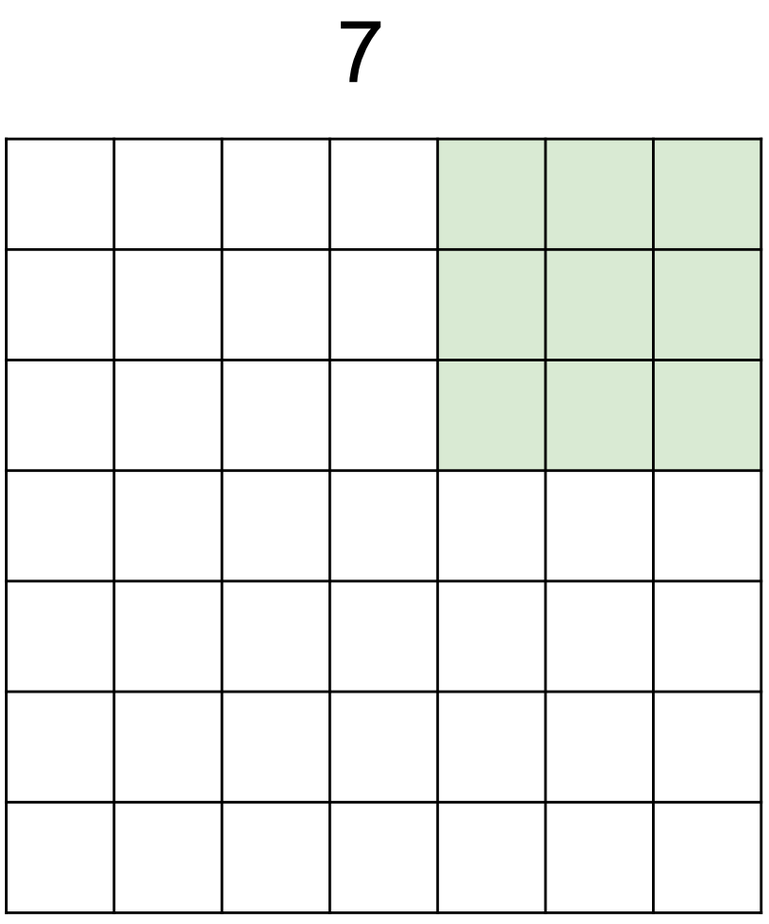
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



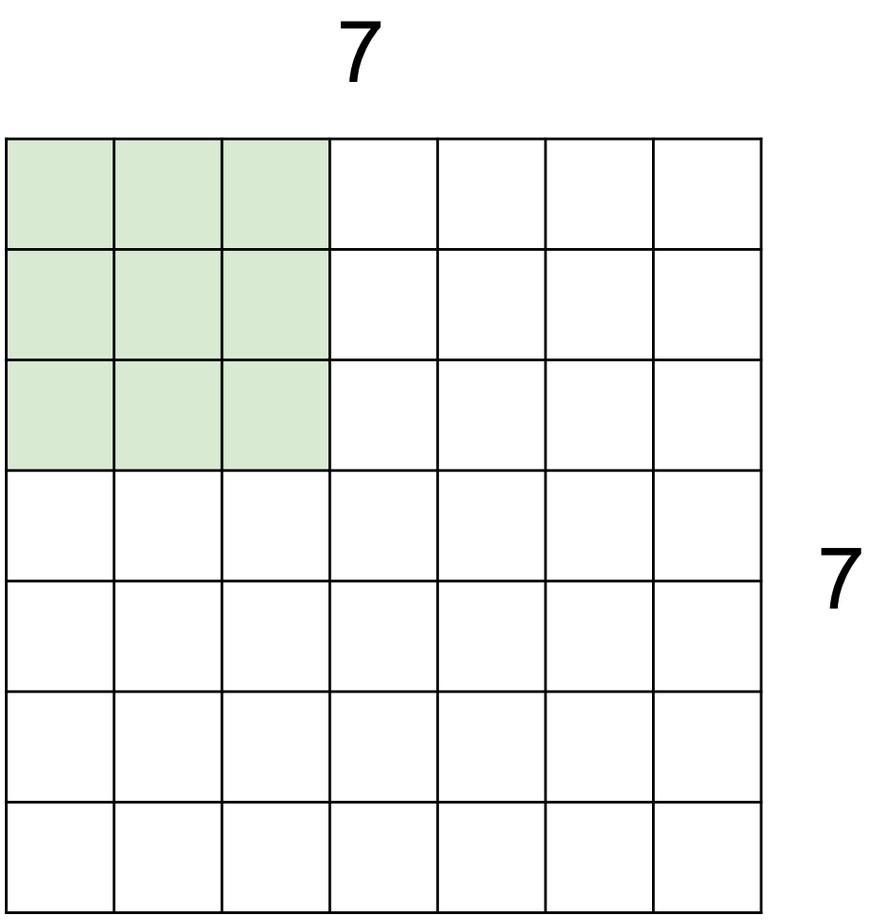
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



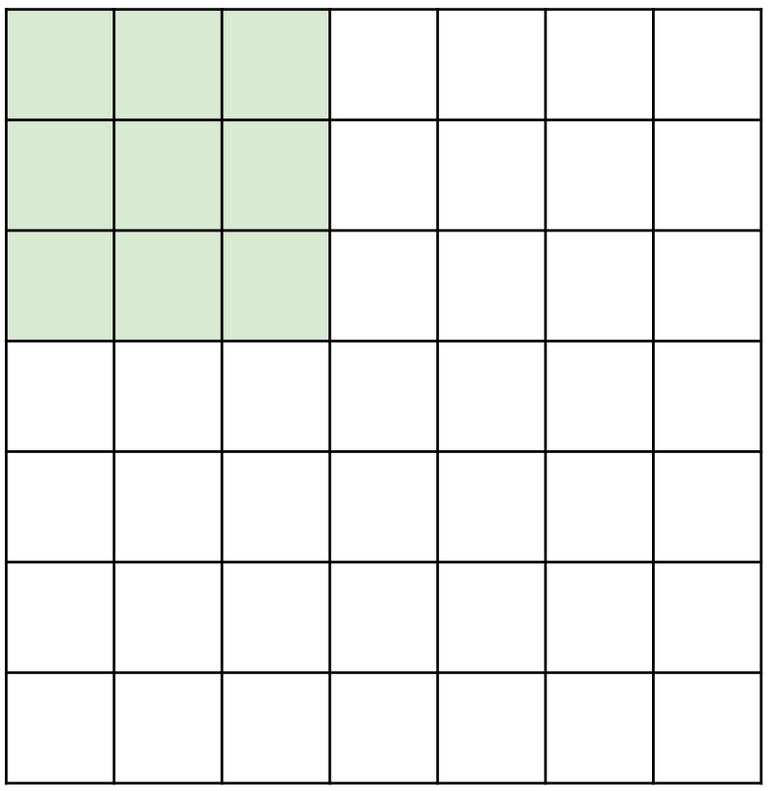
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



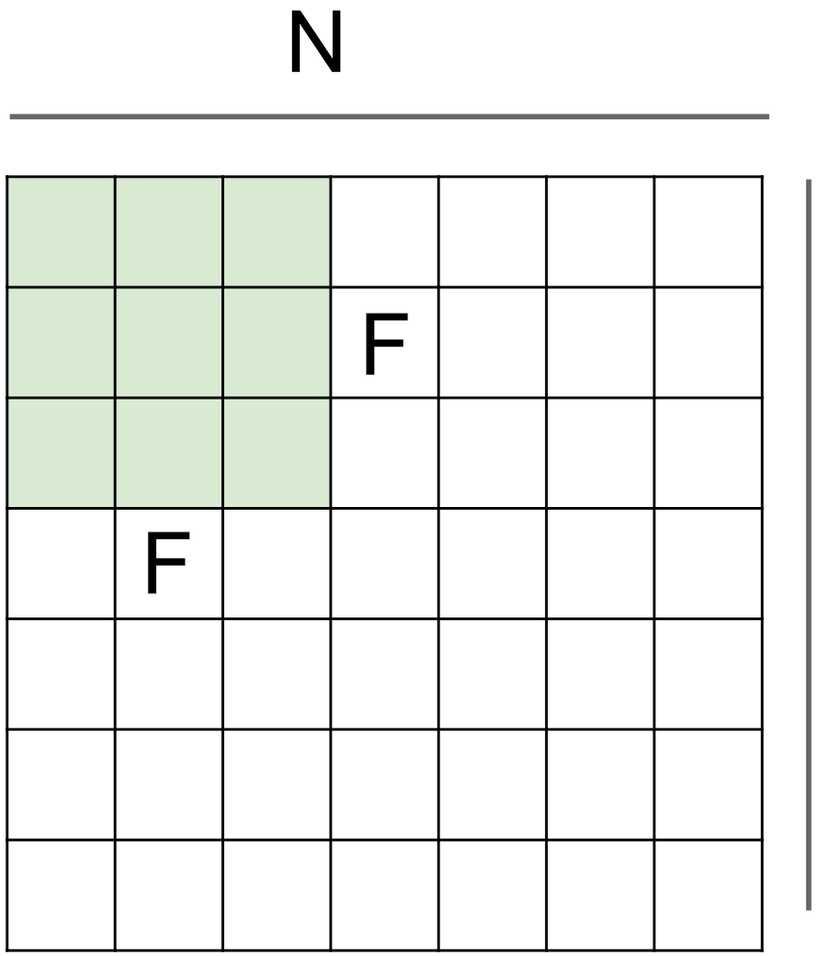
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

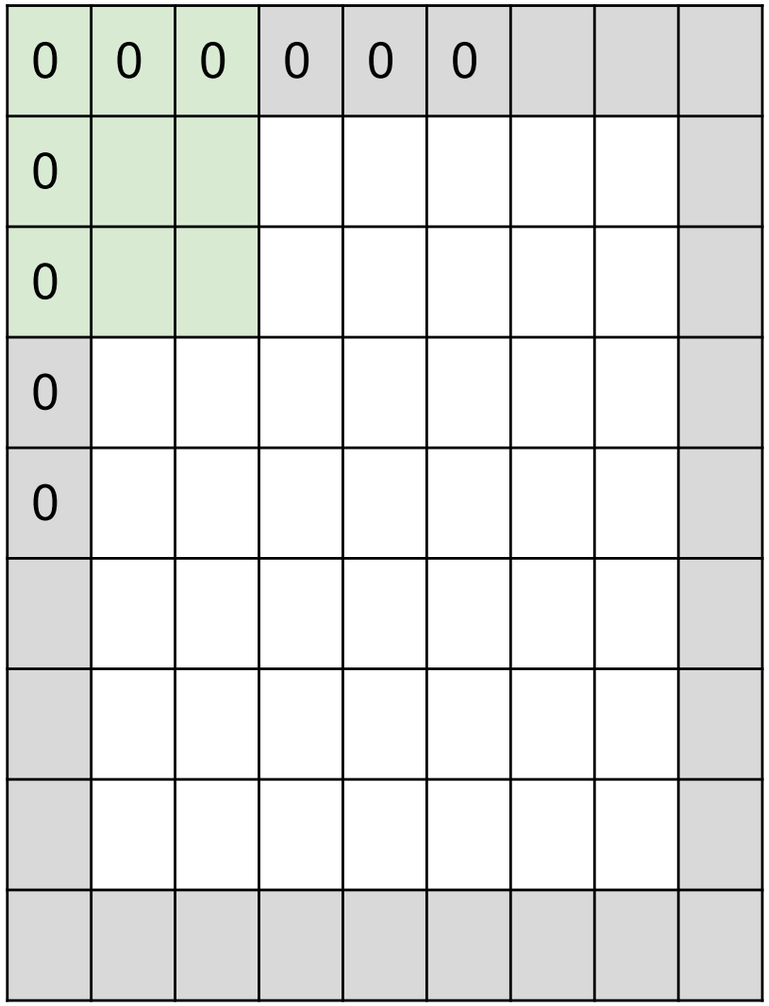
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

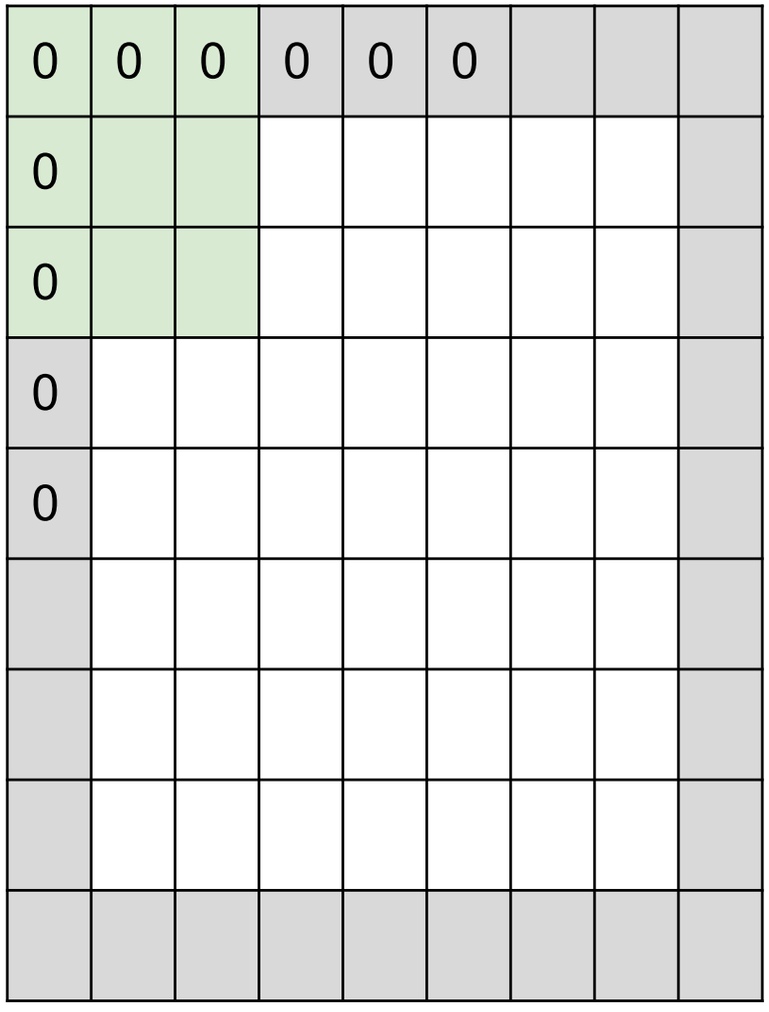
In practice: Common to zero pad the border



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

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

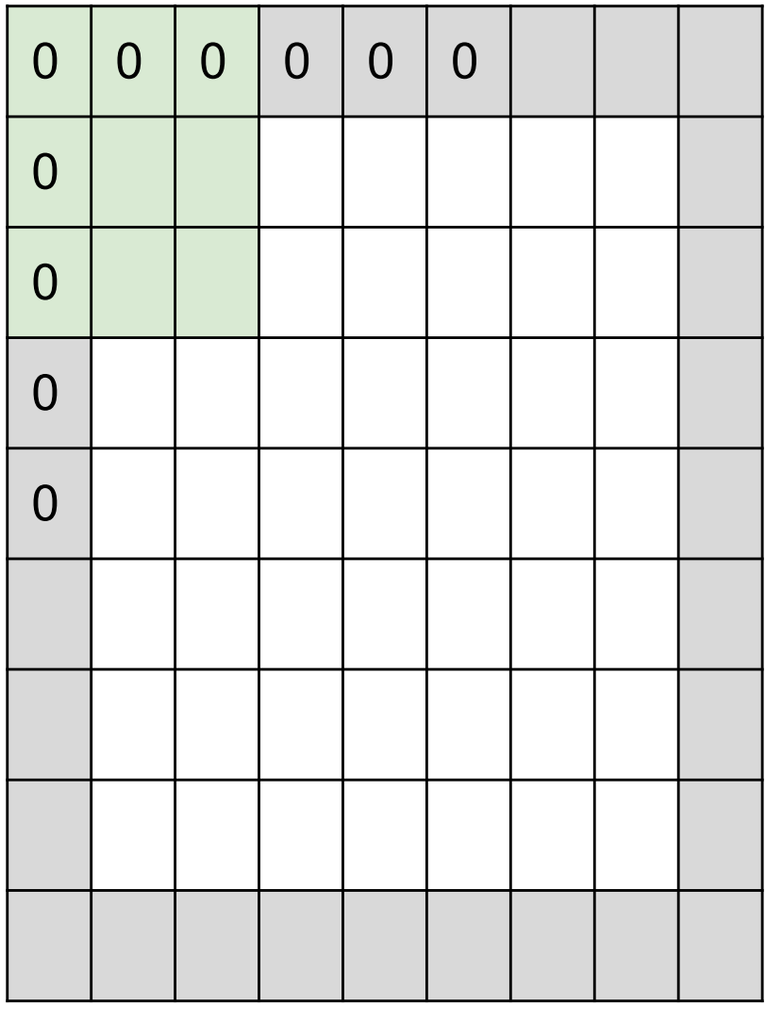
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

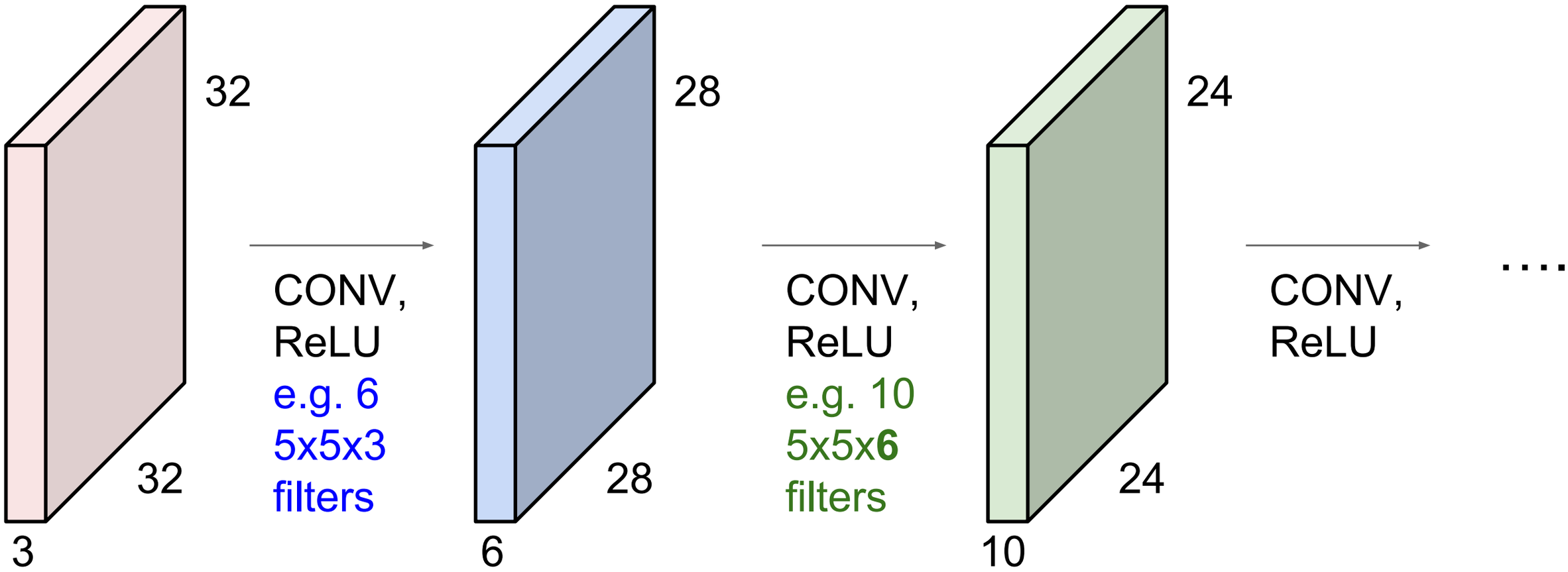


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

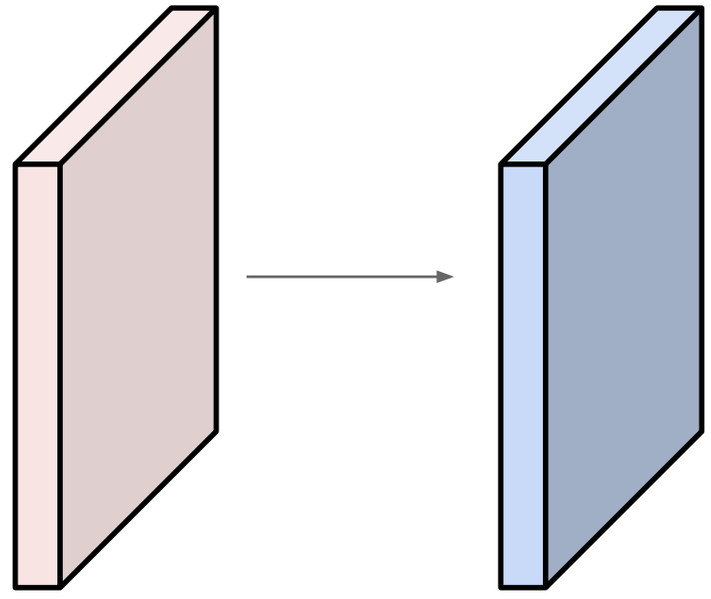
Remember back to…

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



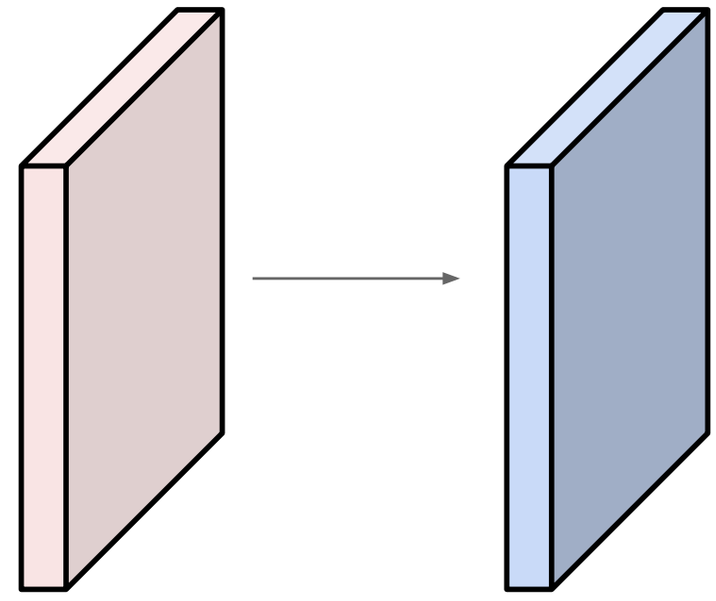
Examples time:

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



Output volume size: ?

Examples time:

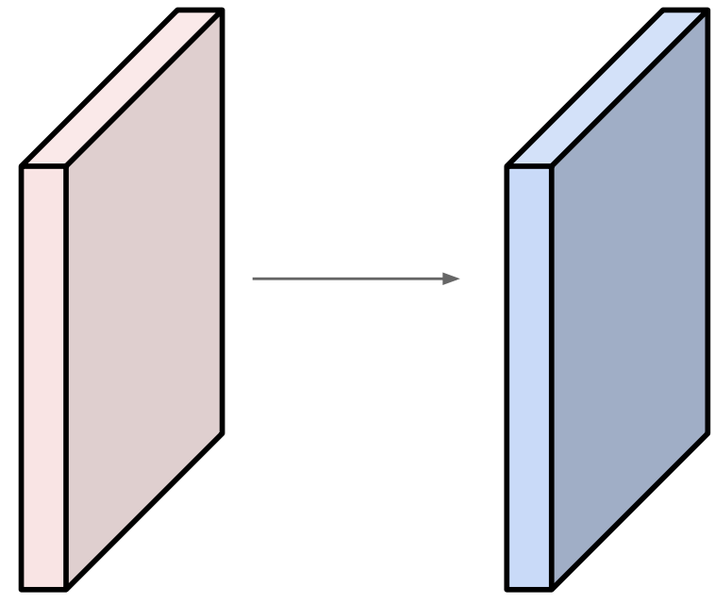


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

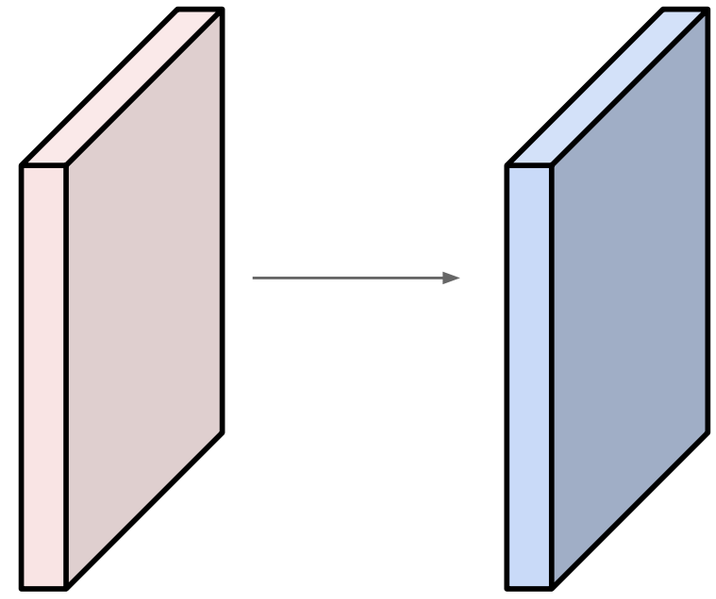
Examples time:

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



Number of parameters in this layer?

Examples time:



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

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary. To summarize, the Conv Layer:

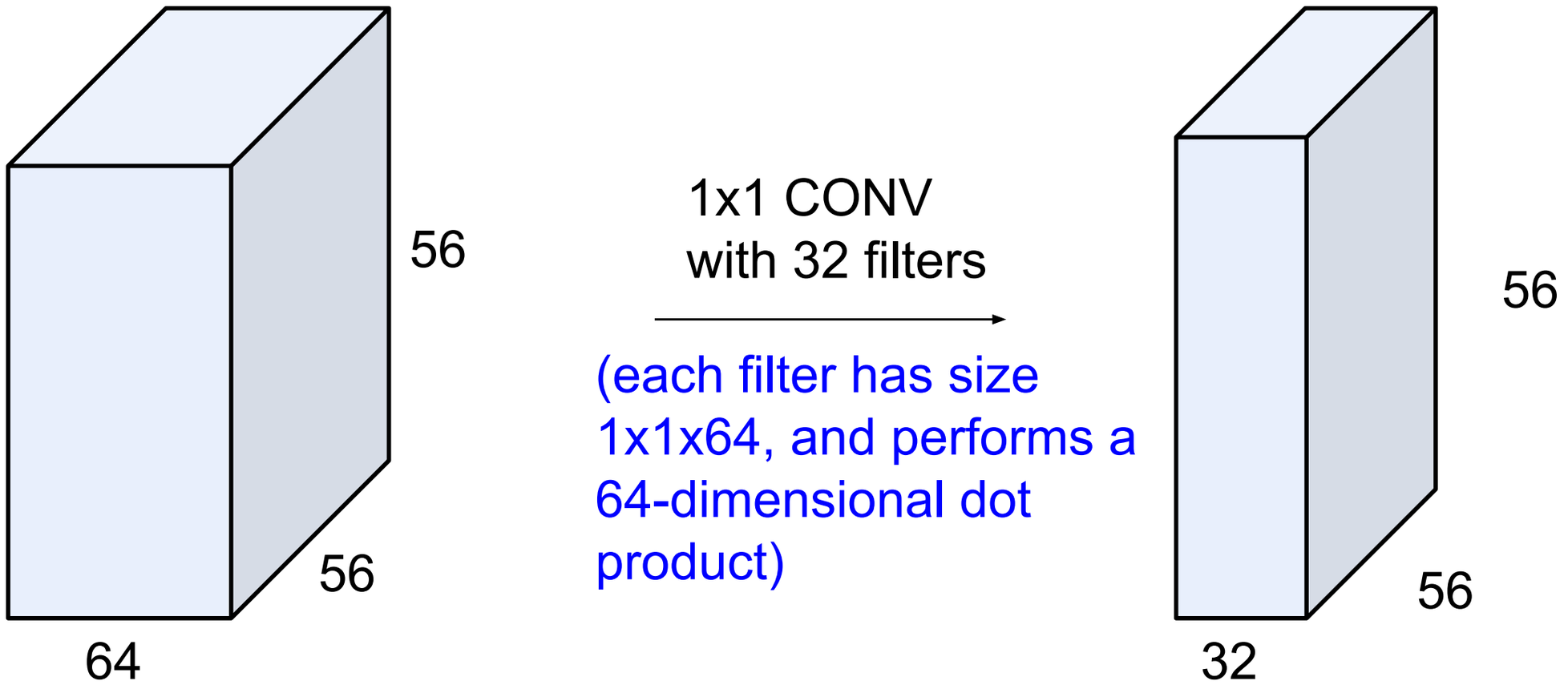
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

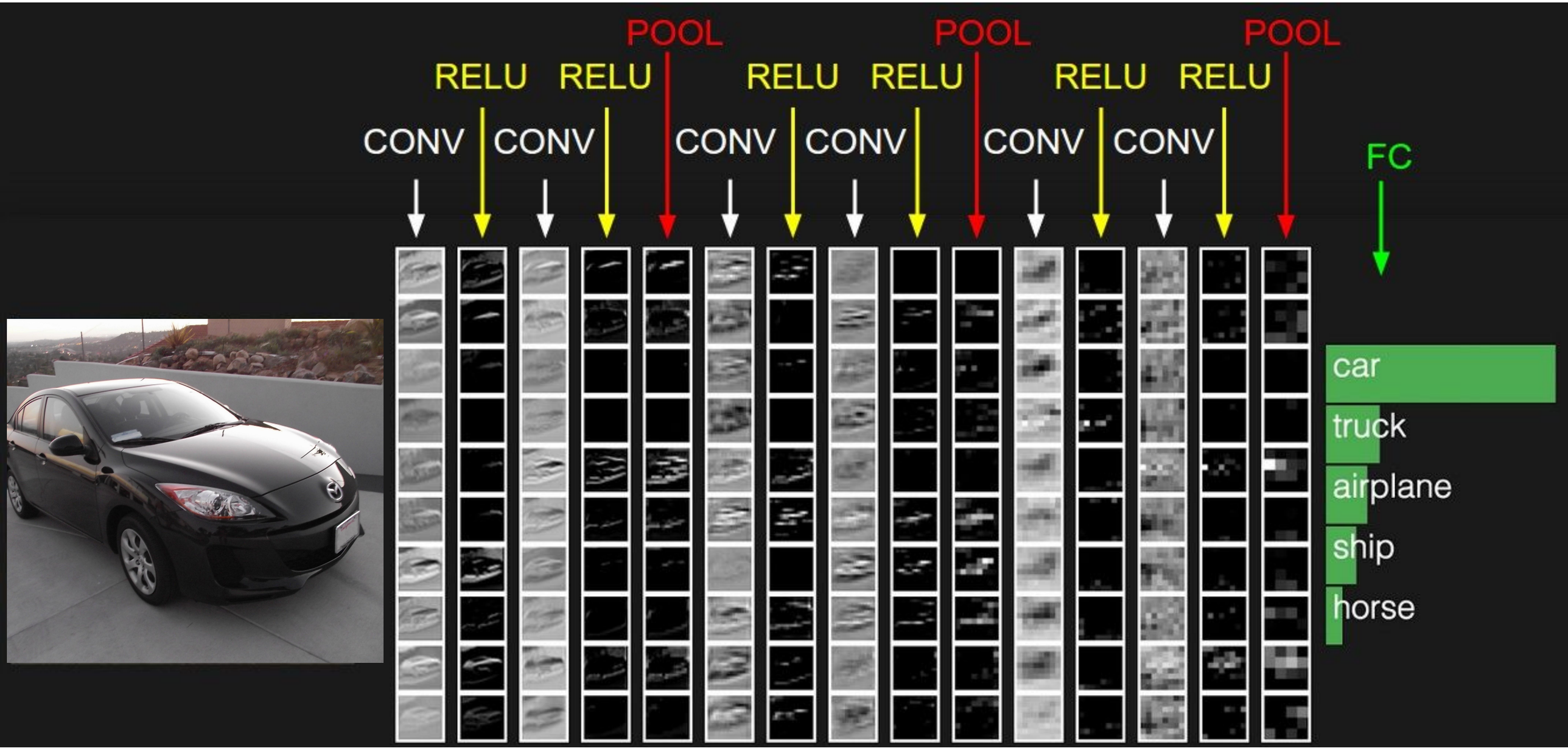
K = (powers of 2, e.g. 32, 64, 128, 512)

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

(btw, 1x1 convolution layers make perfect sense)

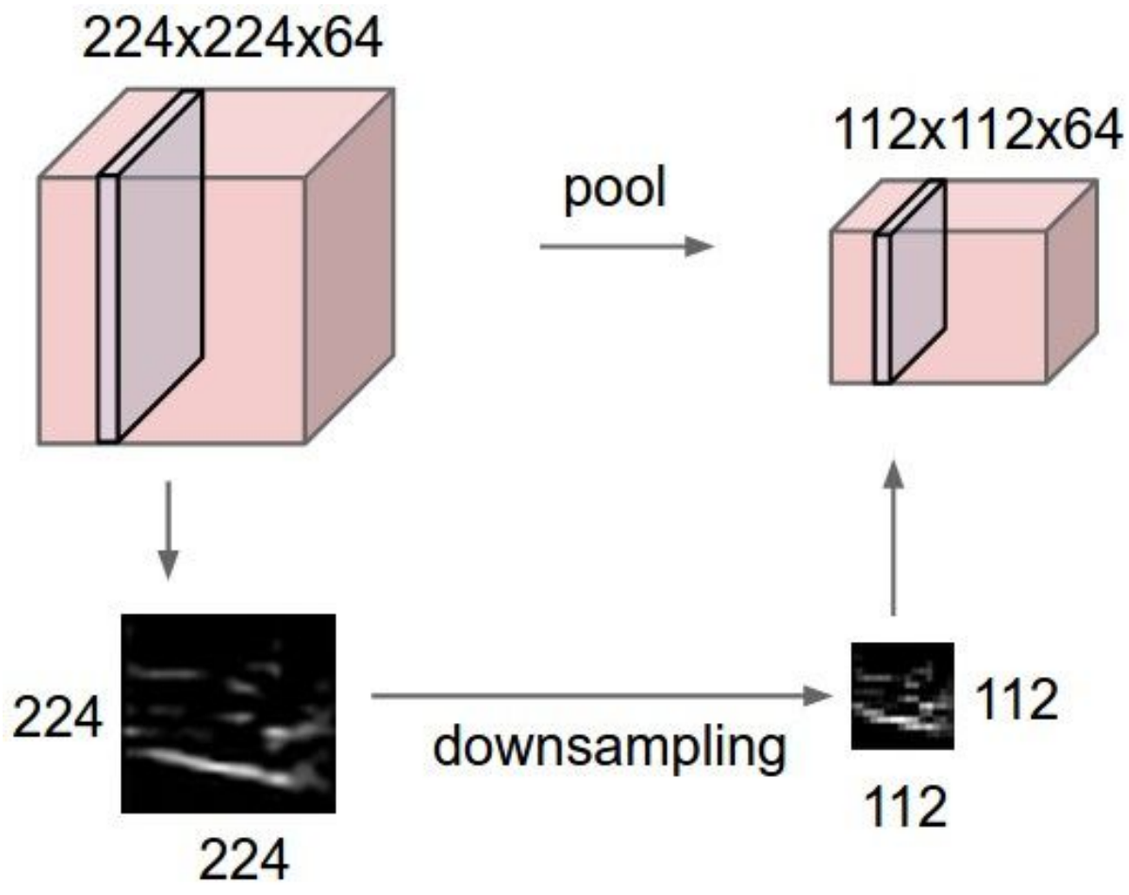


two more layers to go: POOL/FC



Pooling layer

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



MAX POOLING

Single depth slice

x

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

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

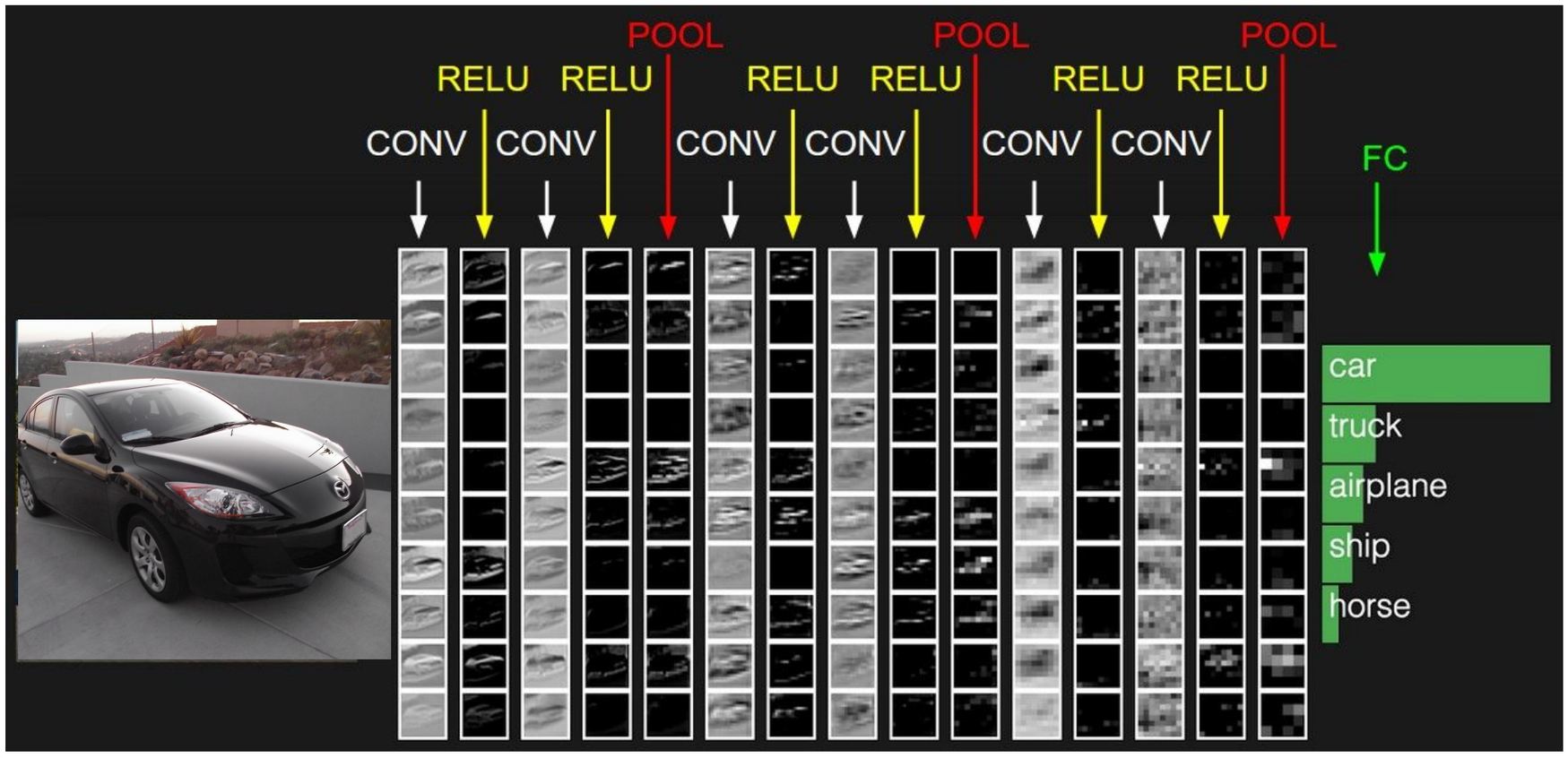
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

Fully Connected Layer (FC layer)

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



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

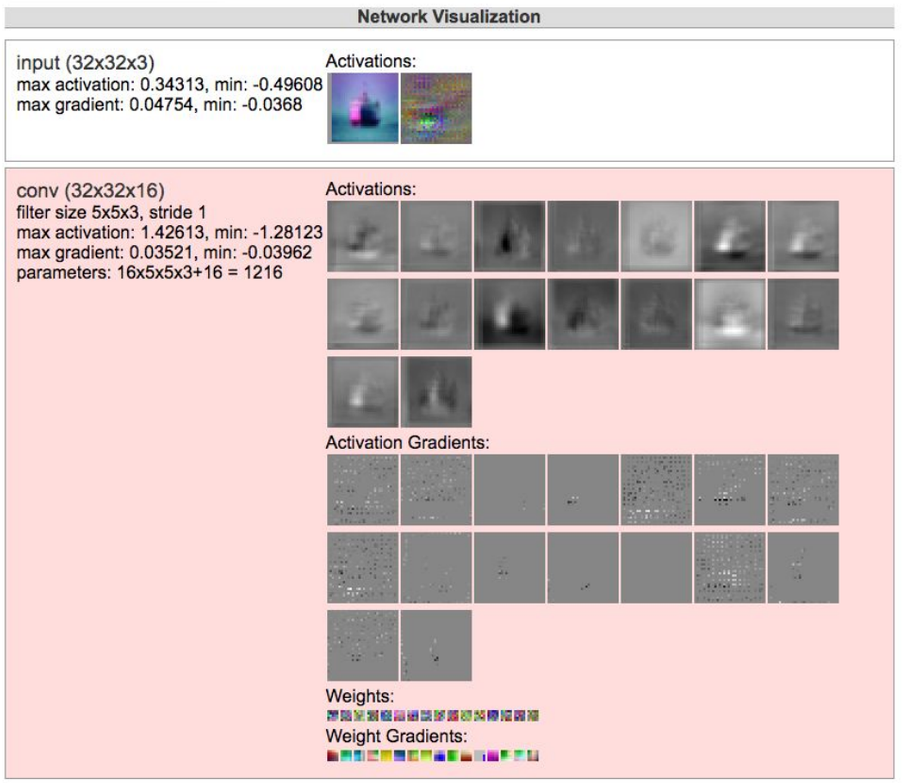
Description

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

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

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

Report questions/bugs/suggestions to @karpathy.



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

Summary

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

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

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