2024-2

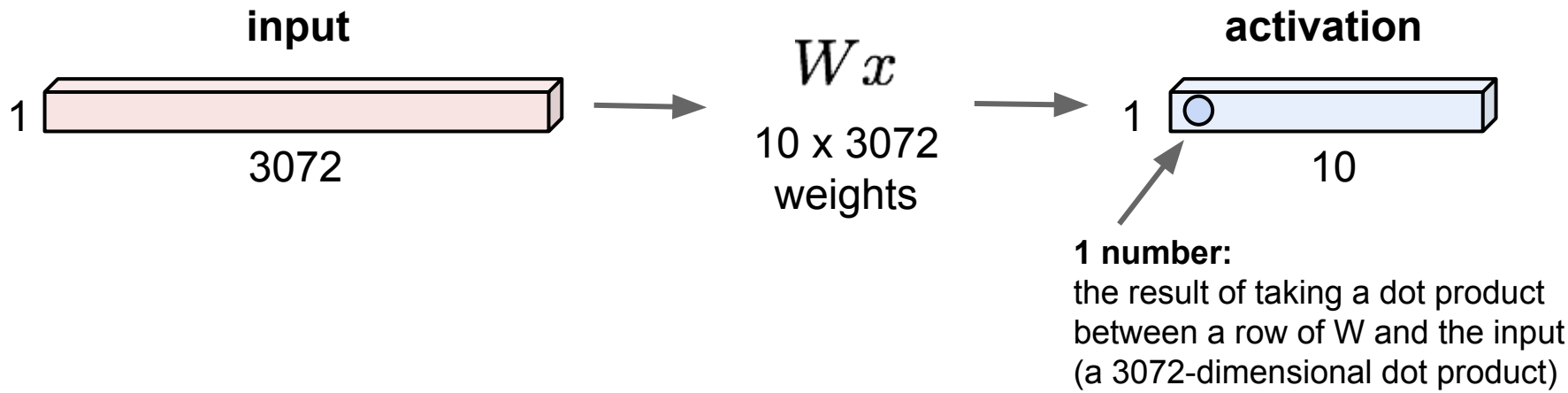
Deep Learning Programming

Lecture 3.2: Convolutional neural networks

sSangryul Jeon
School of Computer Science and Engineering
srjeonn@pusan.ac.kr

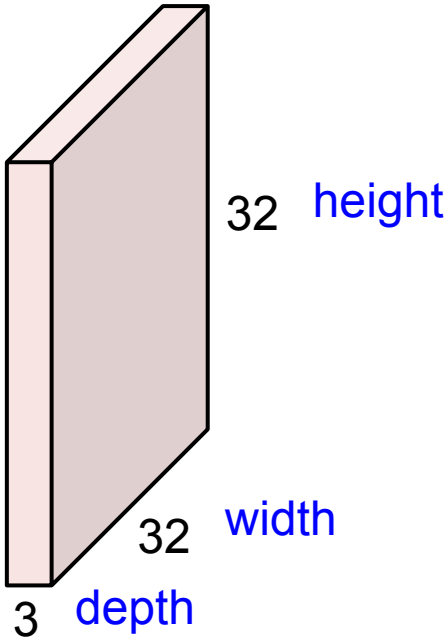
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



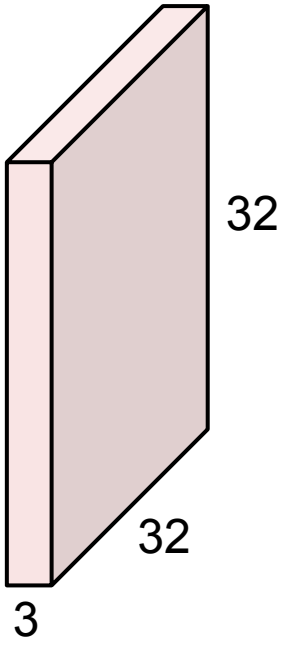
Convolution Layer

32x32x3 image -> preserve spatial structure

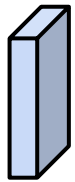


Convolution Layer

32x32x3 image



5x5x3 filter

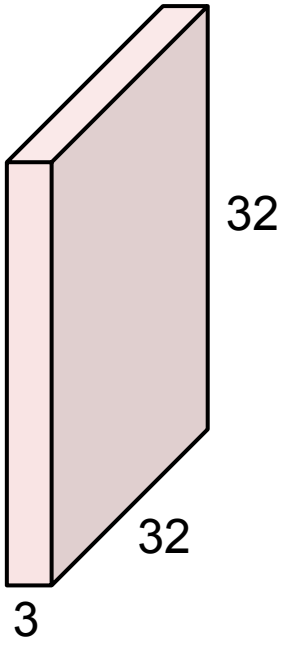


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

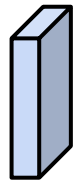
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

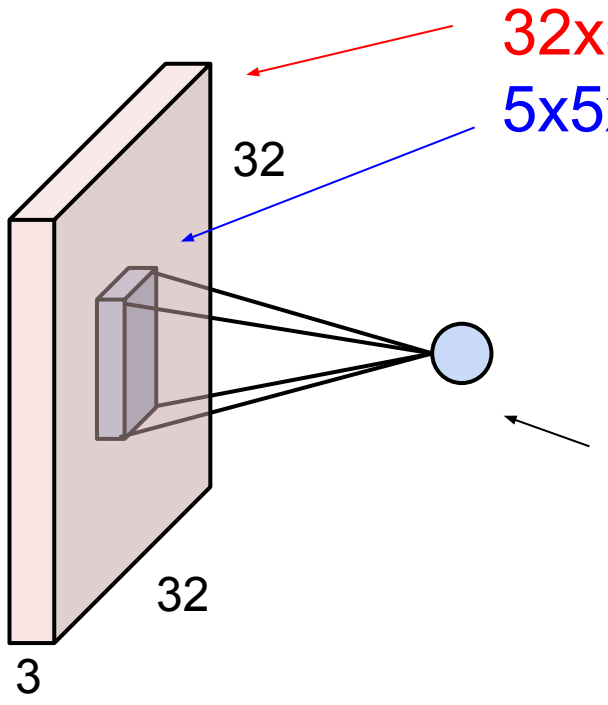


5x5x3 filter



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

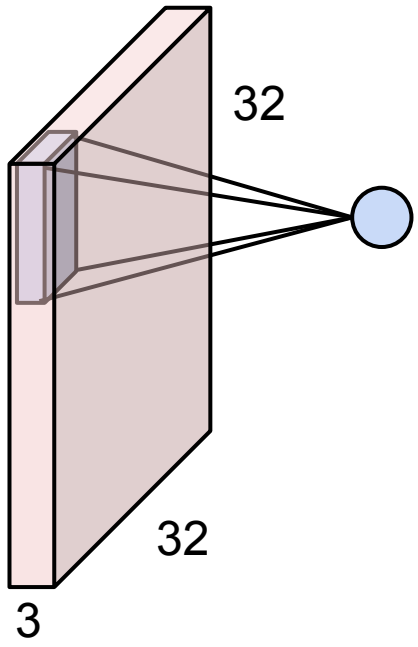
Convolution Layer



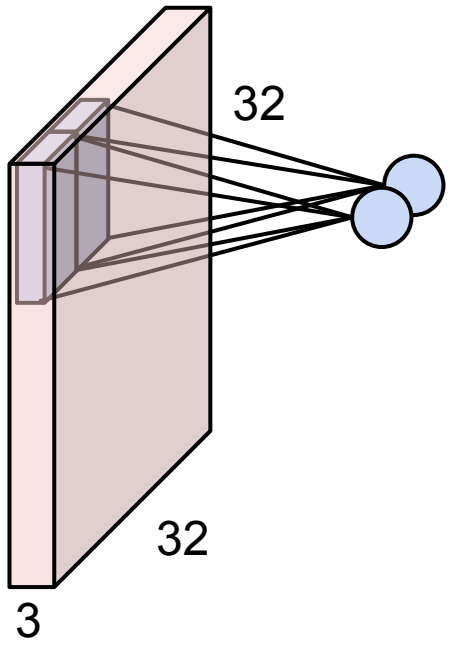
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

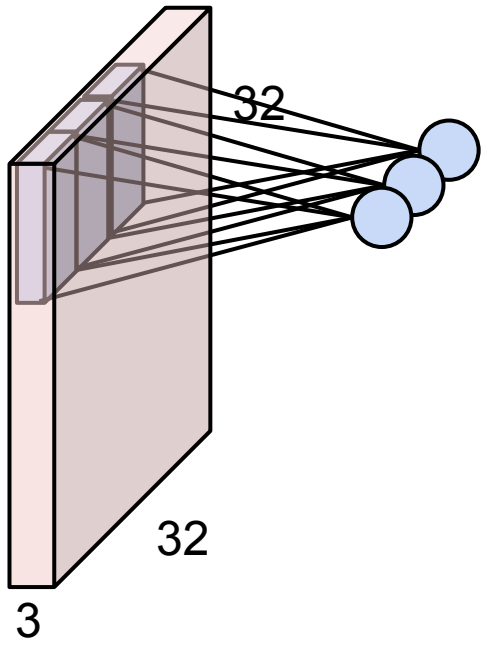
Convolution Layer



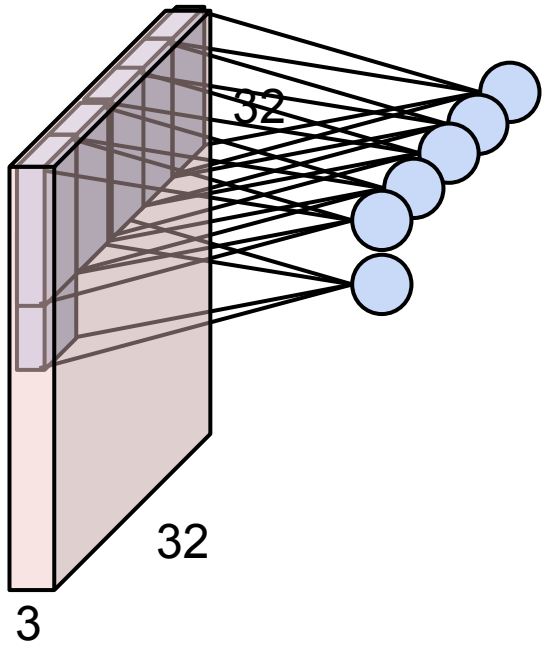
Convolution Layer



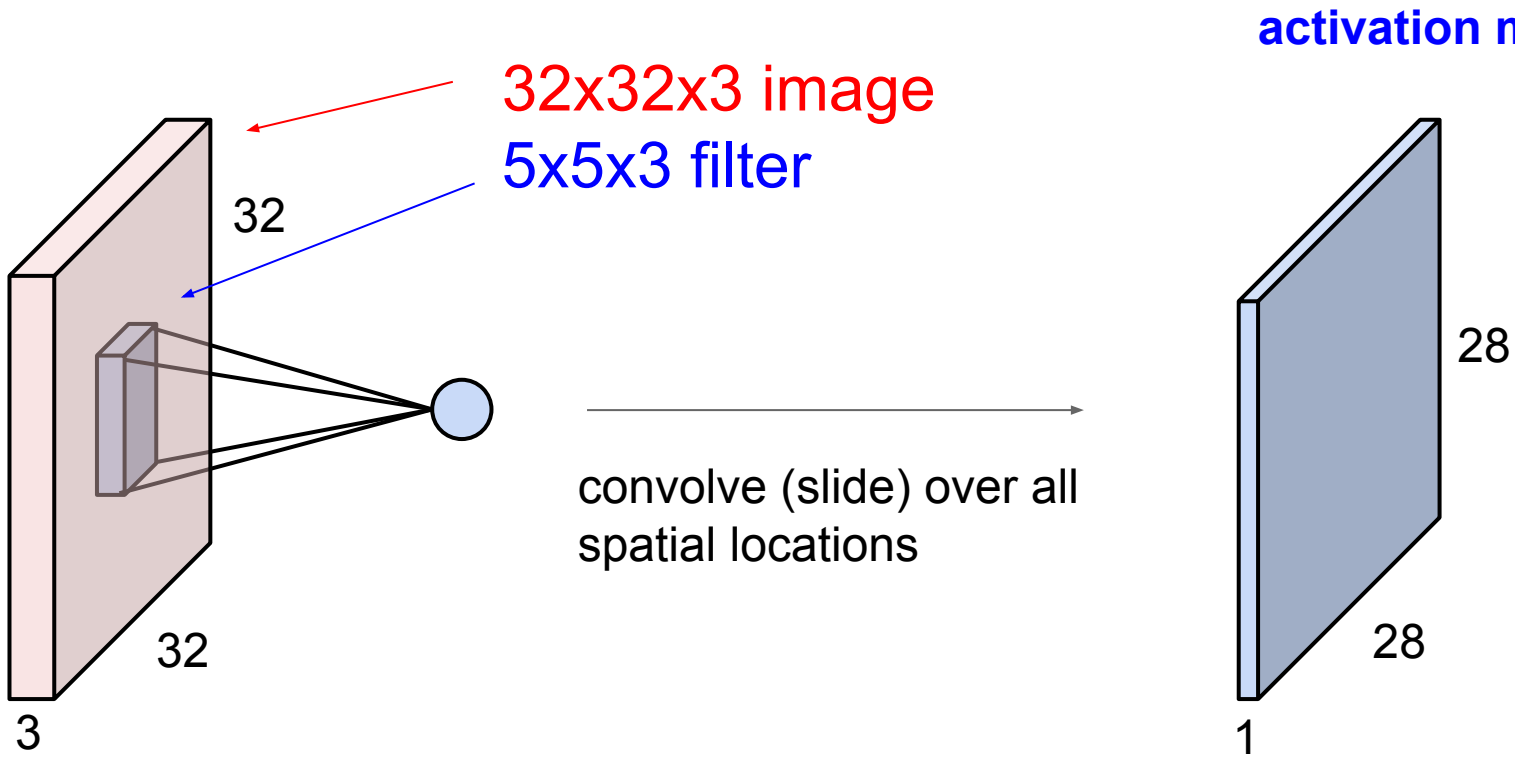
Convolution Layer



Convolution Layer



Convolution Layer



K (3x3 filter)

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}
K_{31}	K_{32}	K_{33}

$*$

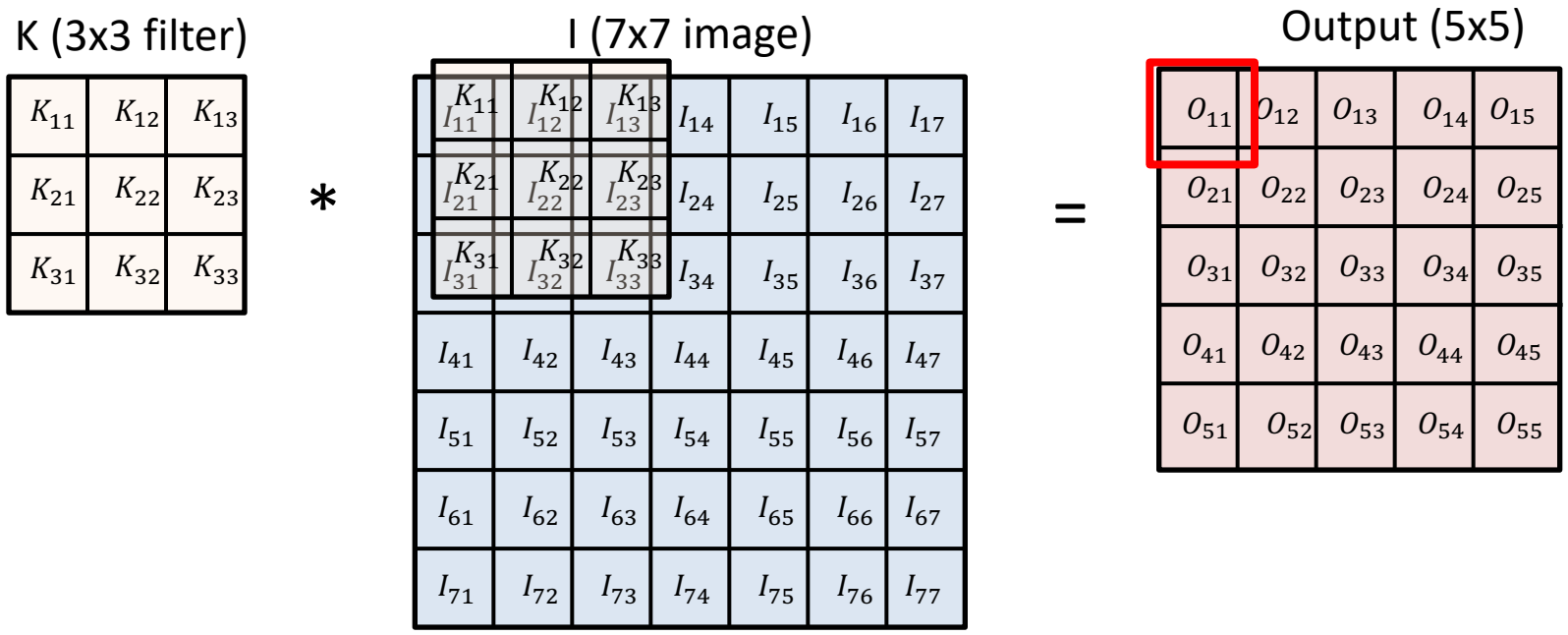
I (7x7 image)

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}
I_{71}	I_{72}	I_{73}	I_{74}	I_{75}	I_{76}	I_{77}

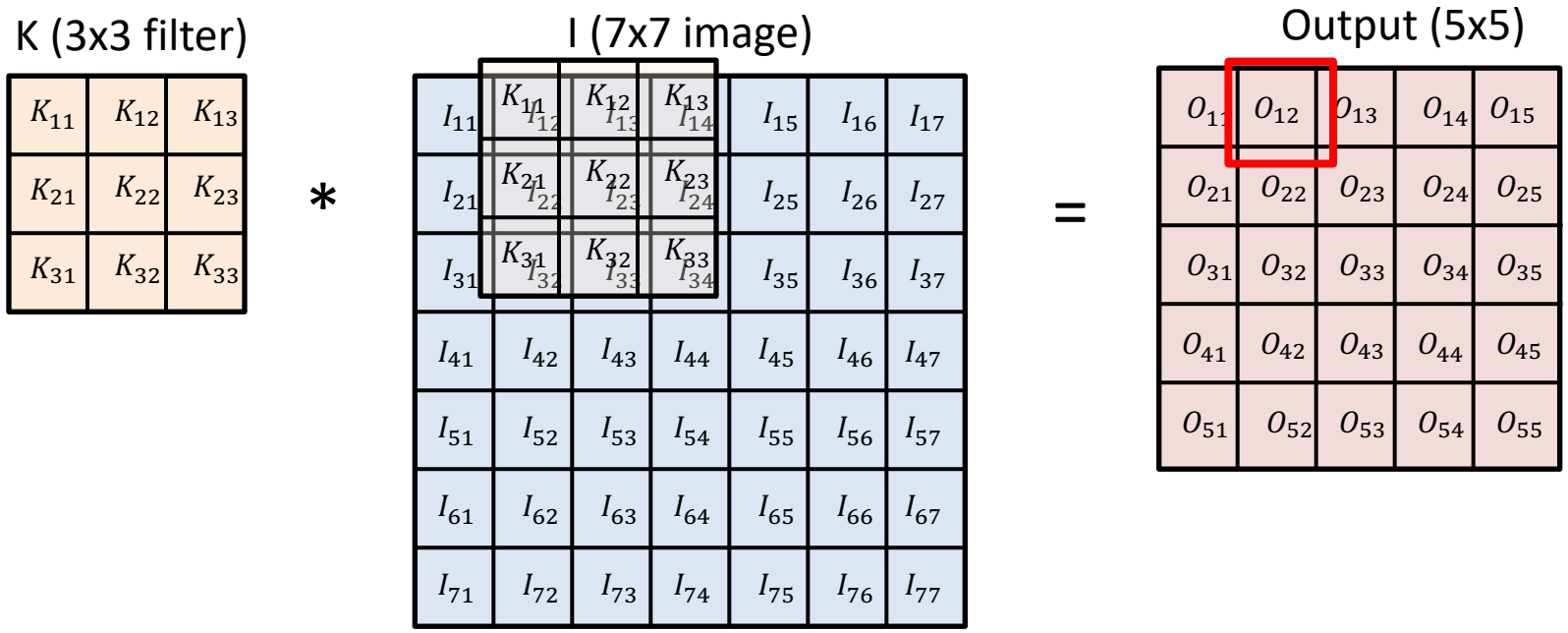
$=$

Output (5x5)

O_{11}	O_{12}	O_{13}	O_{14}	O_{15}
O_{21}	O_{22}	O_{23}	O_{24}	O_{25}
O_{31}	O_{32}	O_{33}	O_{34}	O_{35}
O_{41}	O_{42}	O_{43}	O_{44}	O_{45}
O_{51}	O_{52}	O_{53}	O_{54}	O_{55}

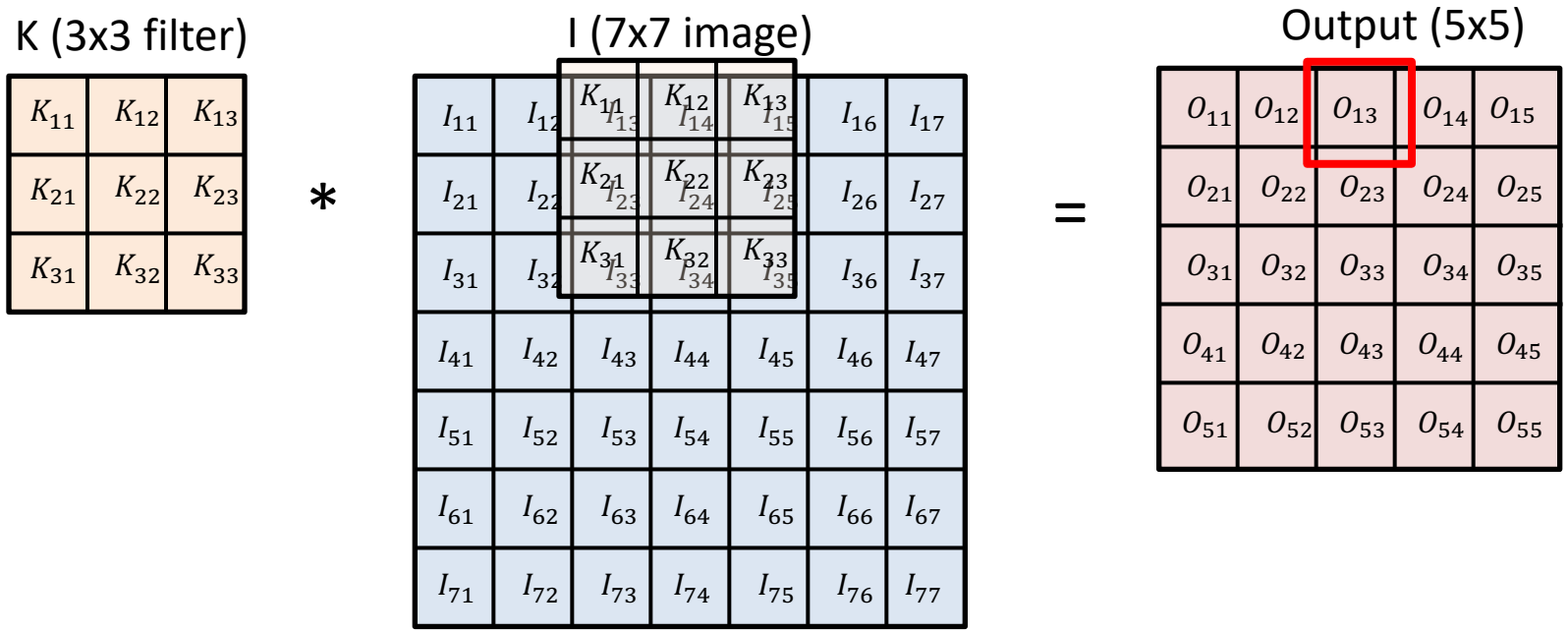


$$O_{11} = I_{11}K_{11}+I_{12}K_{12}+I_{13}K_{13}+I_{21}K_{21}+I_{22}K_{22}+I_{23}K_{23}+I_{31}K_{31}+I_{32}K_{32}+I_{33}K_{33}+bias$$



$$O_{11} = I_{11}K_{11}+I_{12}K_{12}+I_{13}K_{13}+I_{21}K_{21}+I_{22}K_{22}+I_{23}K_{23}+I_{31}K_{31}+I_{32}K_{32}+I_{33}K_{33}+bias$$

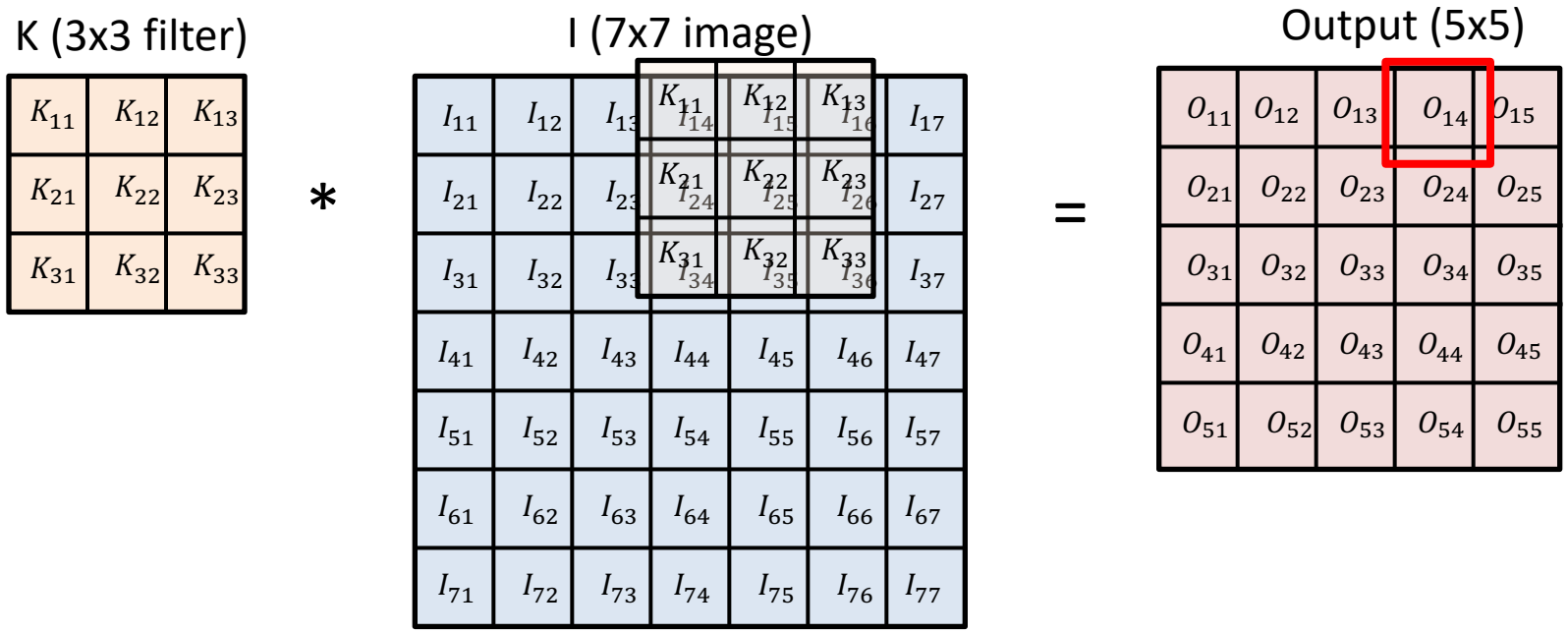
$$O_{12} = I_{12}K_{11}+I_{13}K_{12}+I_{14}K_{13}+I_{22}K_{21}+I_{23}K_{22}+I_{24}K_{23}+I_{32}K_{31}+I_{33}K_{32}+I_{34}K_{33}+bias$$



$$O_{11} = I_{11}K_{11}+I_{12}K_{12}+I_{13}K_{13}+I_{21}K_{21}+I_{22}K_{22}+I_{23}K_{23}+I_{31}K_{31}+I_{32}K_{32}+I_{33}K_{33}+bias$$

$$O_{12} = I_{12}K_{11}+I_{13}K_{12}+I_{14}K_{13}+I_{22}K_{21}+I_{23}K_{22}+I_{24}K_{23}+I_{32}K_{31}+I_{33}K_{32}+I_{34}K_{33}+bias$$

$$O_{13} = I_{13}K_{11}+I_{14}K_{12}+I_{15}K_{13}+I_{23}K_{21}+I_{24}K_{22}+I_{25}K_{23}+I_{33}K_{31}+I_{34}K_{32}+I_{35}K_{33}+bias$$



$$O_{11} = I_{11}K_{11}+I_{12}K_{12}+I_{13}K_{13}+I_{21}K_{21}+I_{22}K_{22}+I_{23}K_{23}+I_{31}K_{31}+I_{32}K_{32}+I_{33}K_{33}+bias$$

$$O_{12} = I_{12}K_{11}+I_{13}K_{12}+I_{14}K_{13}+I_{22}K_{21}+I_{23}K_{22}+I_{24}K_{23}+I_{32}K_{31}+I_{33}K_{32}+I_{34}K_{33}+bias$$

$$O_{13} = I_{13}K_{11}+I_{14}K_{12}+I_{15}K_{13}+I_{23}K_{21}+I_{24}K_{22}+I_{25}K_{23}+I_{33}K_{31}+I_{34}K_{32}+I_{35}K_{33}+bias$$

$$O_{14} = I_{14}K_{11}+I_{15}K_{12}+I_{16}K_{13}+I_{24}K_{21}+I_{25}K_{22}+I_{26}K_{23}+I_{34}K_{31}+I_{35}K_{32}+I_{36}K_{33}+bias$$

- Continuous convolution

$$(f * g)(t) = \int f(\tau)g(t-\tau)d\tau = \int f(t-\tau)g(t)d\tau$$

- Discrete convolution

$$(f * g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(t-i) = \sum_{i=-\infty}^{\infty} f(t-i)g(i)$$

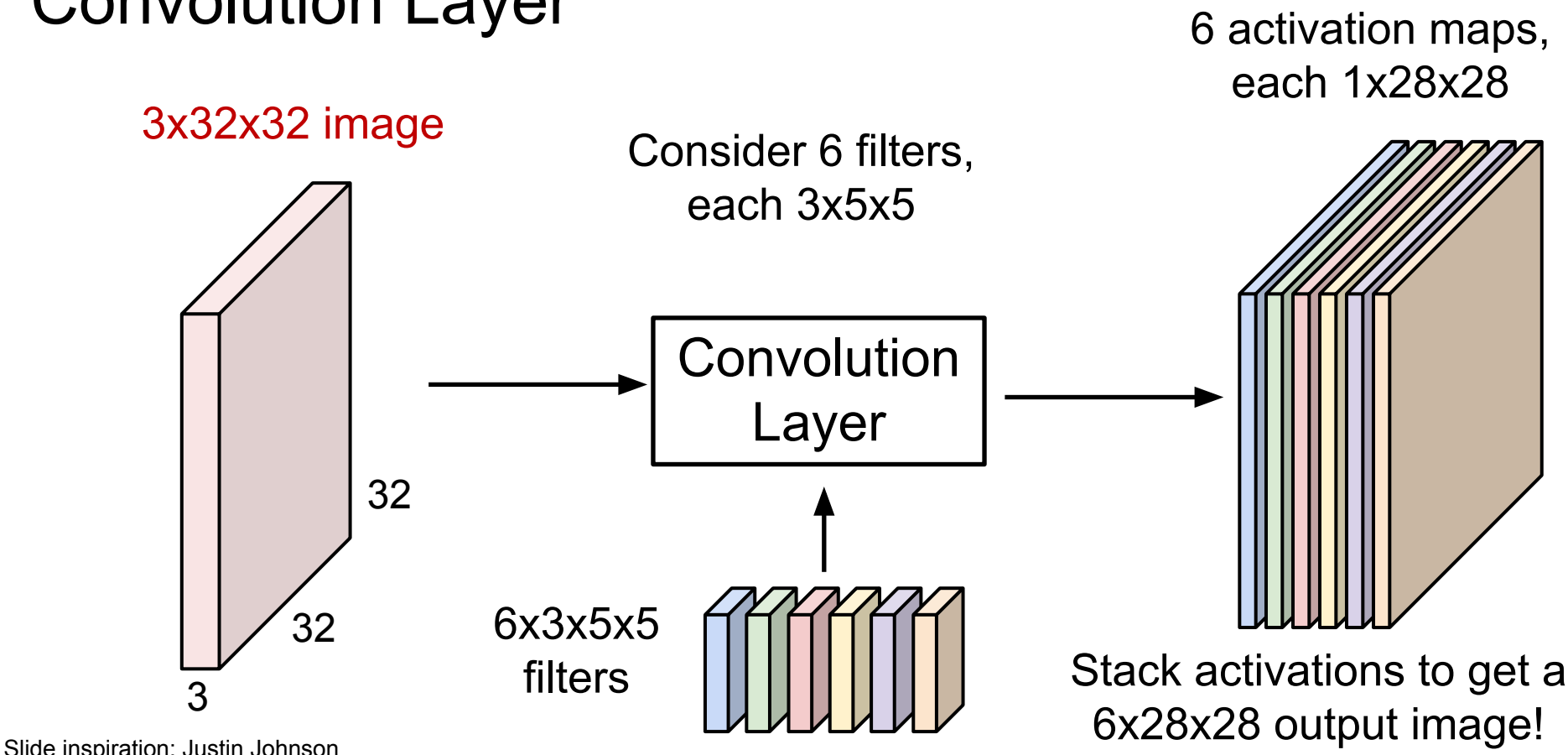
- 2D image convolution

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) = \sum_m \sum_n I(i-m, i-n)K(m,n)$$

Convolution Layer

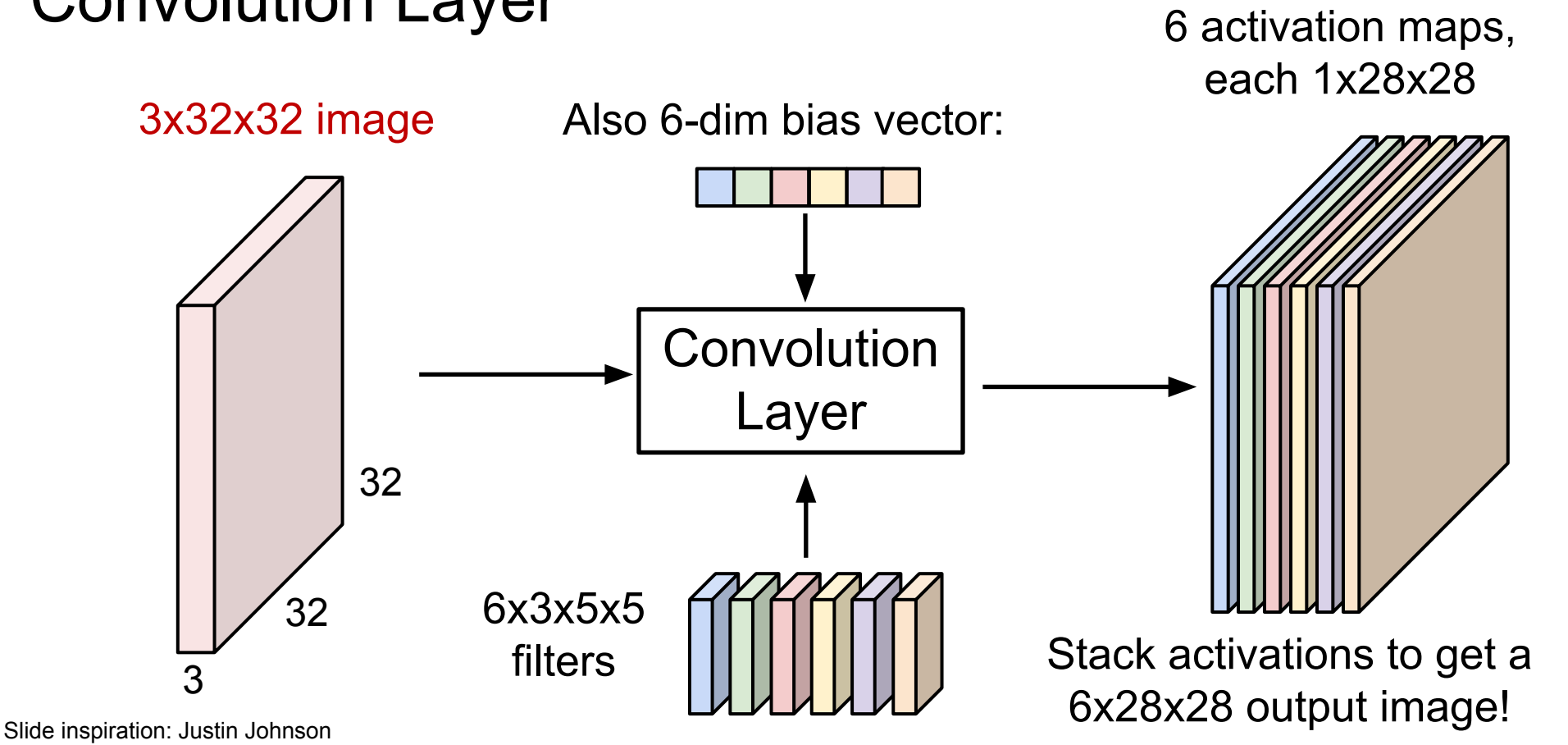
consider a second, green filter

Convolution Layer



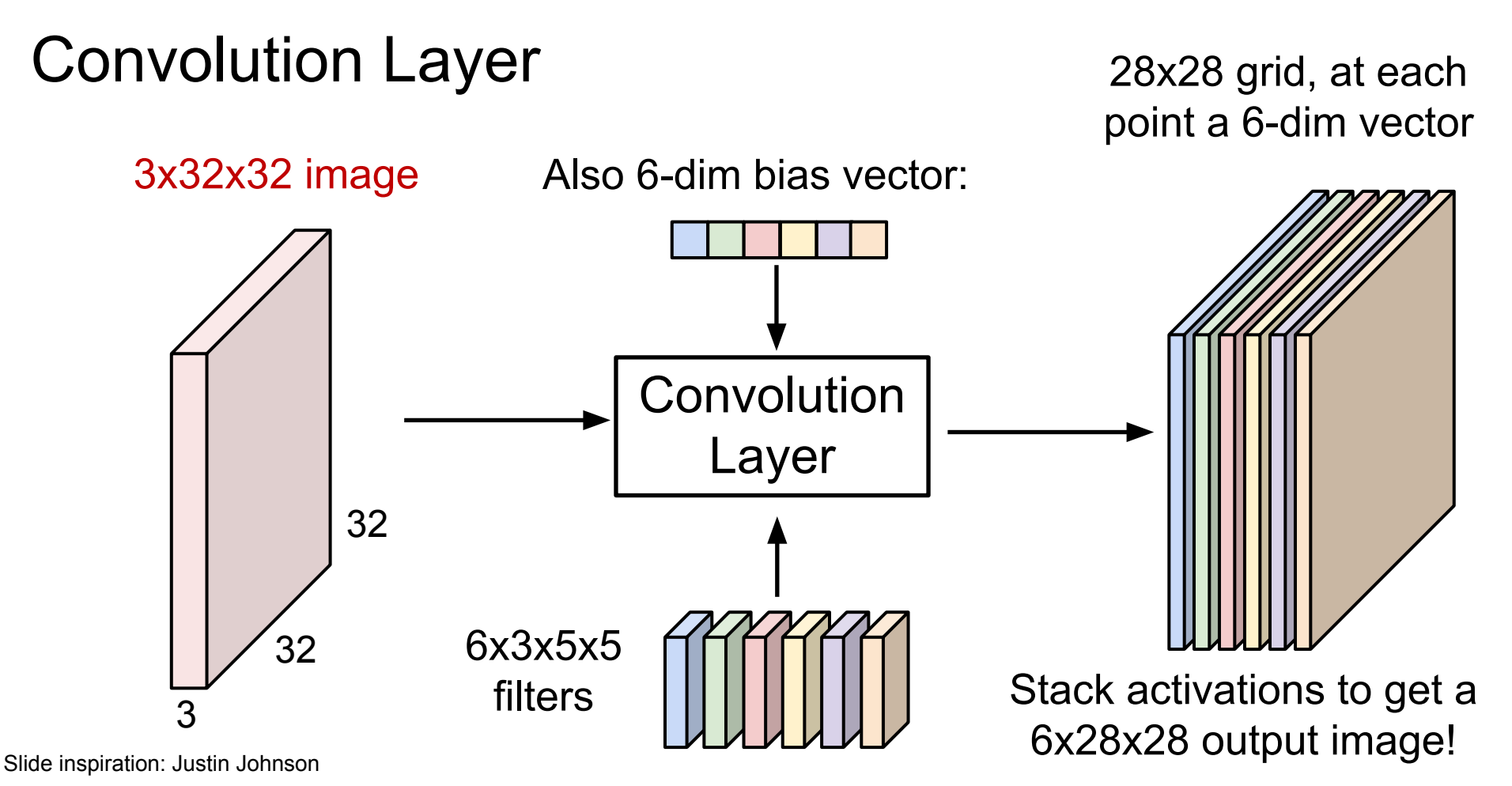
Slide inspiration: Justin Johnson

Convolution Layer



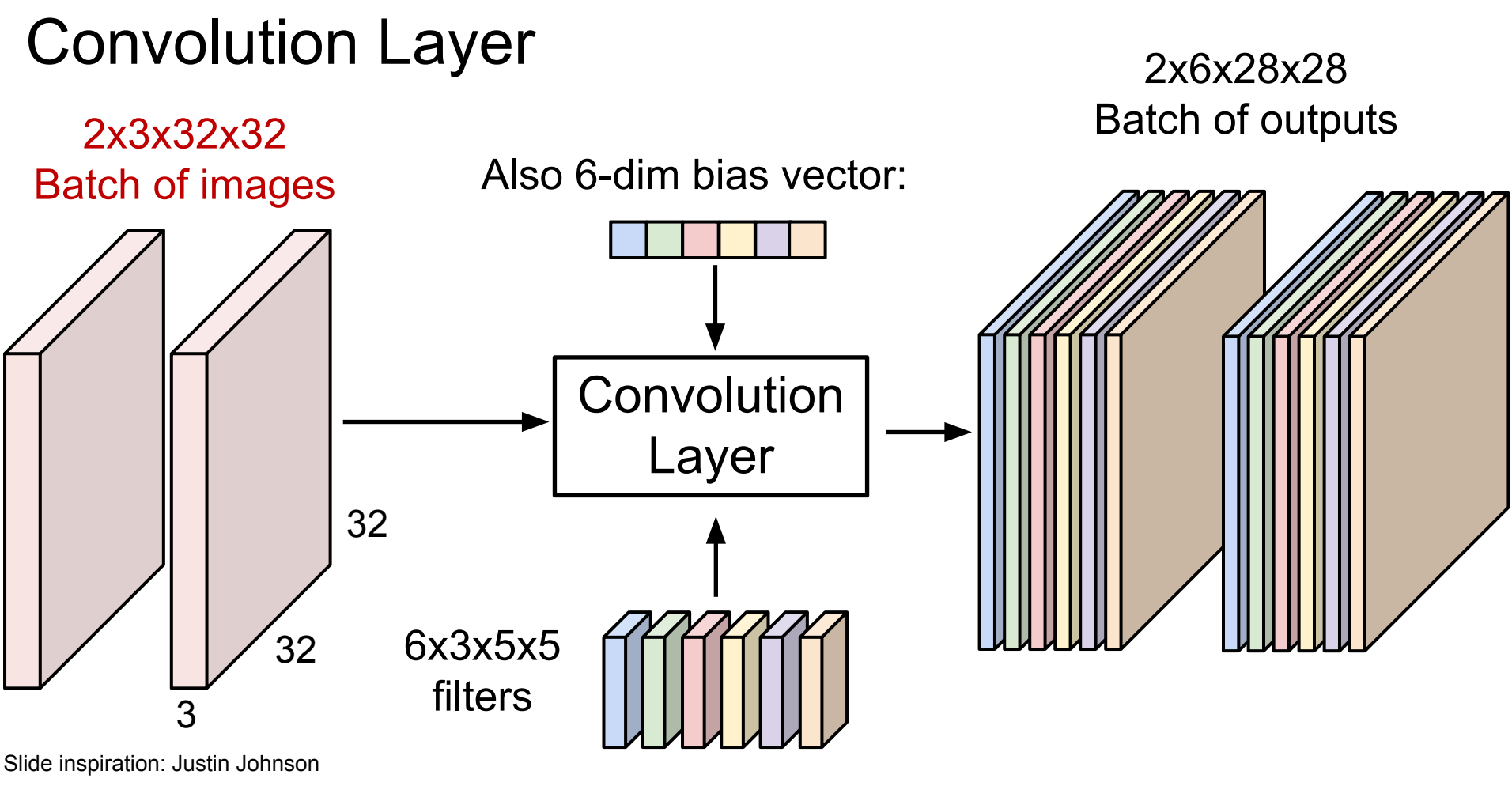
Slide inspiration: Justin Johnson

Convolution Layer



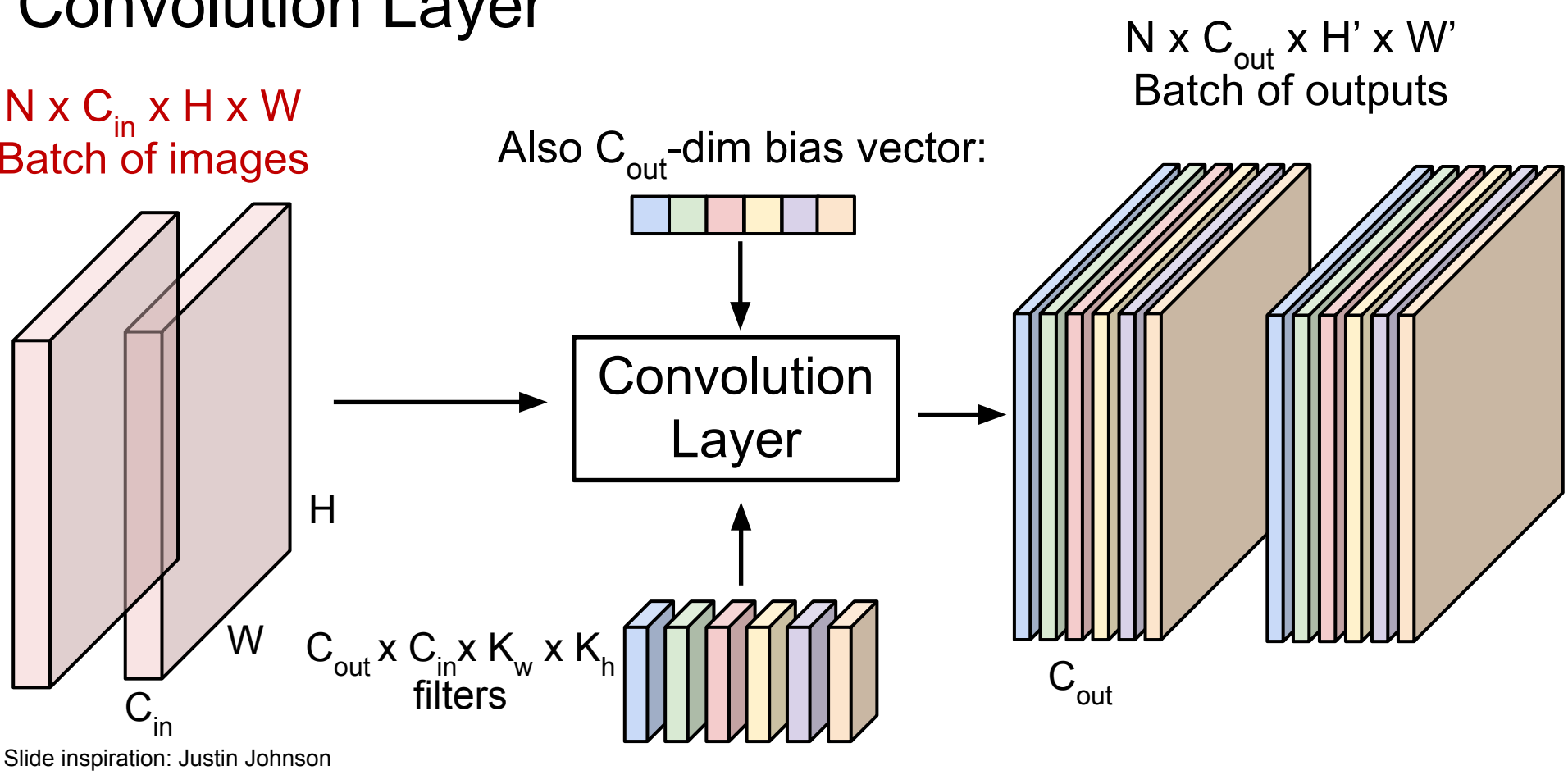
Slide inspiration: Justin Johnson

Convolution Layer



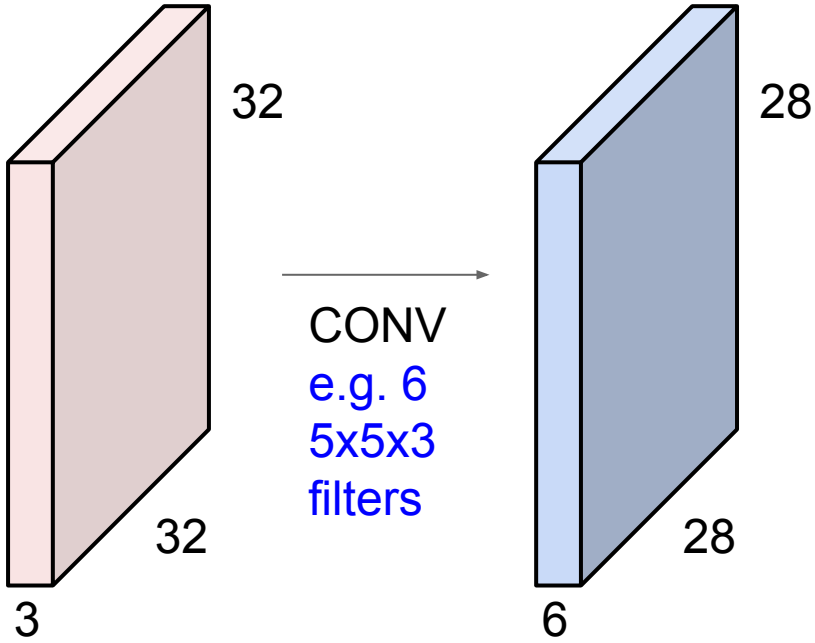
Slide inspiration: Justin Johnson

Convolution Layer

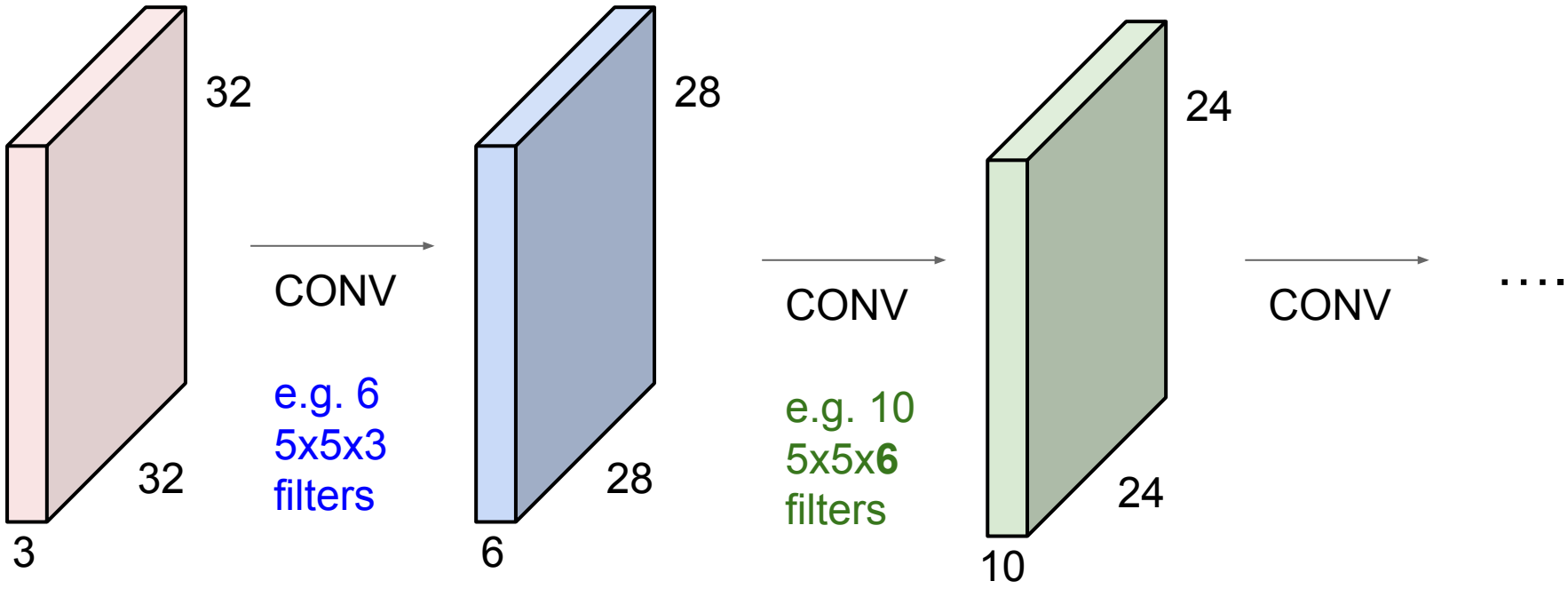


Slide inspiration: Justin Johnson

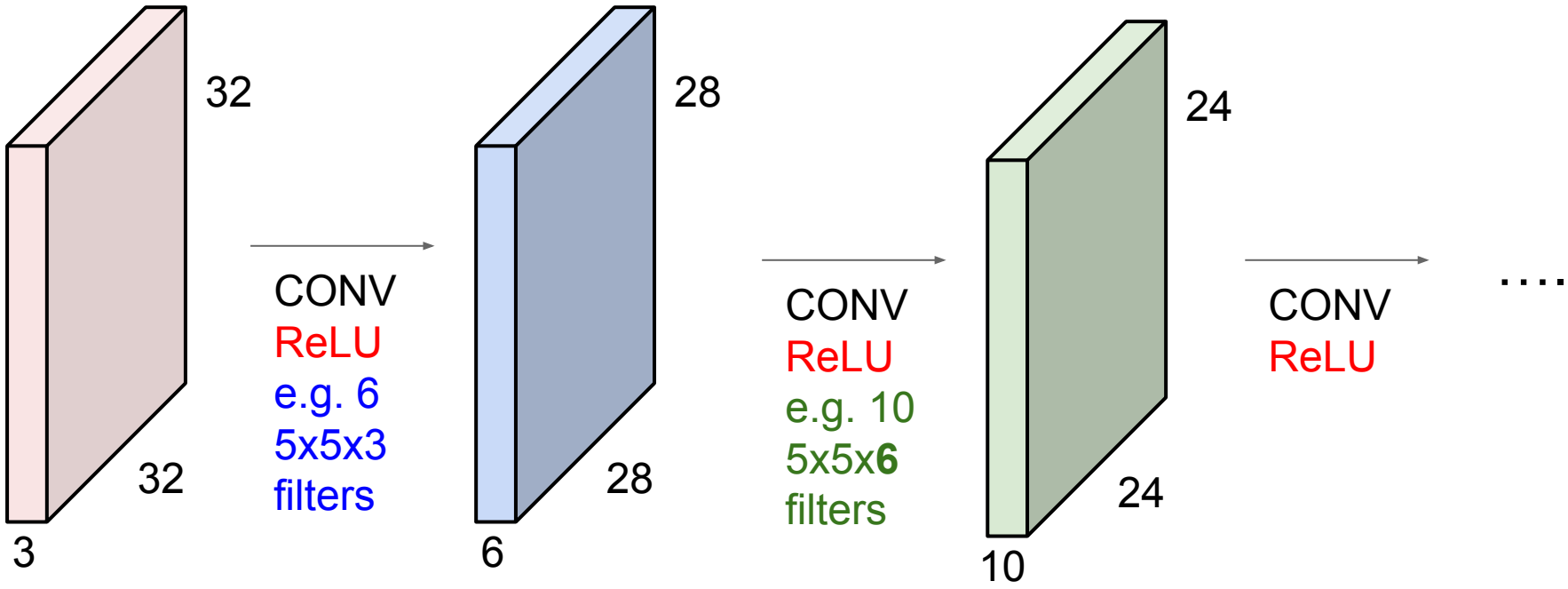
Preview: ConvNet is a sequence of Convolution Layers



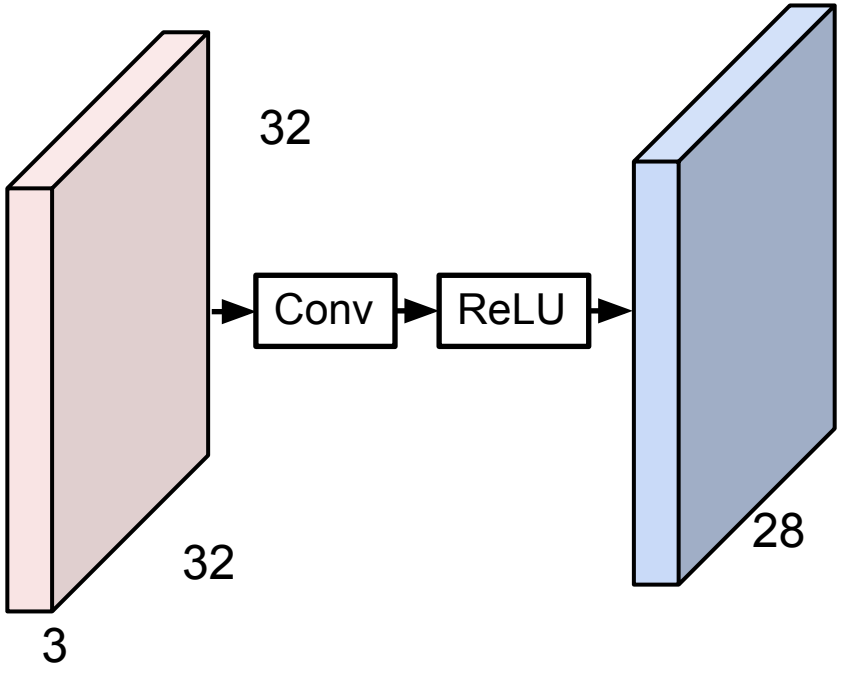
Preview: ConvNet is a sequence of Convolution Layers



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

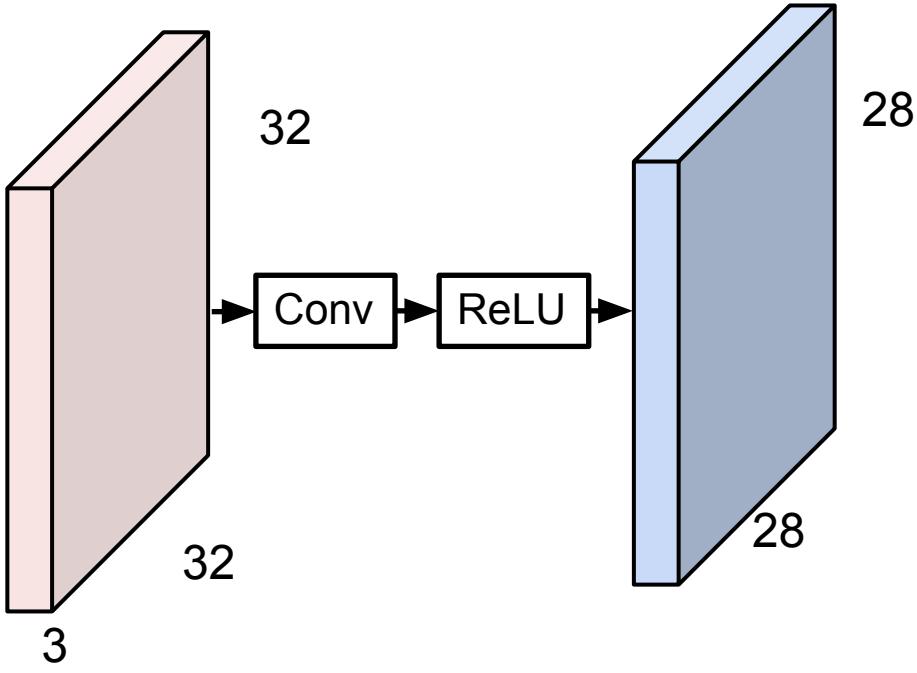


Preview: What do convolutional filters learn?



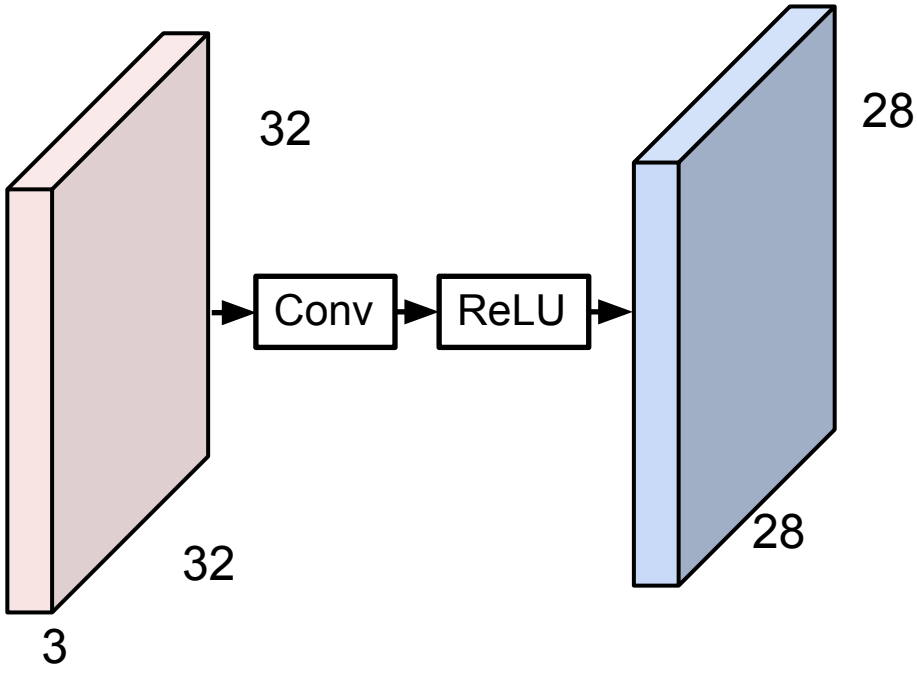
Linear classifier: One template per class

Preview: What do convolutional filters learn?

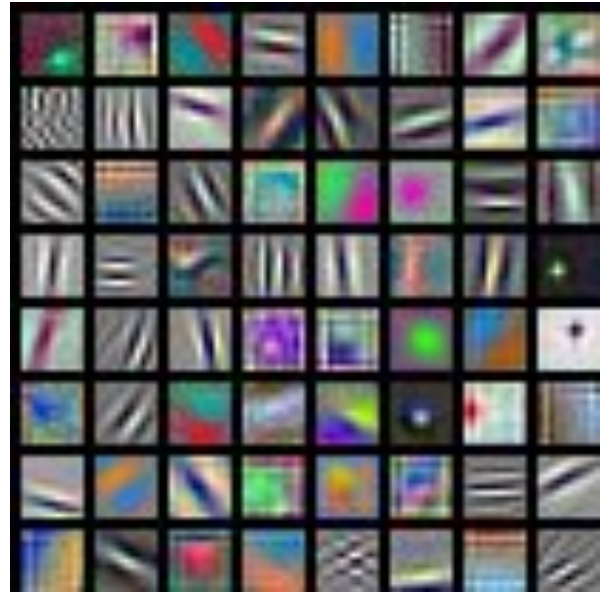


MLP: Bank of whole-image templates

Preview: What do convolutional filters learn?



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)

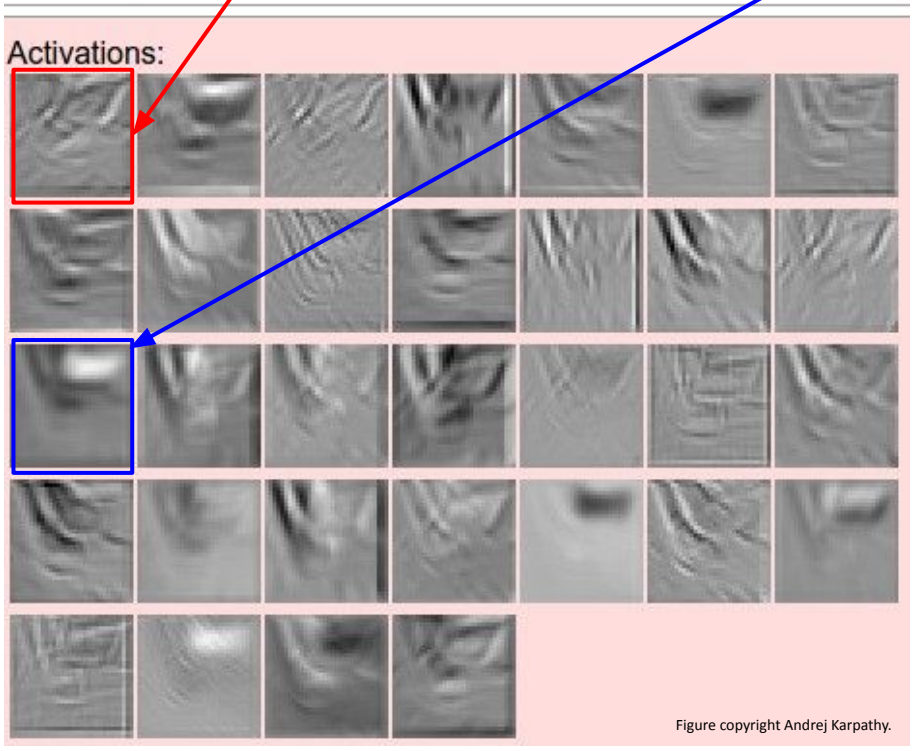


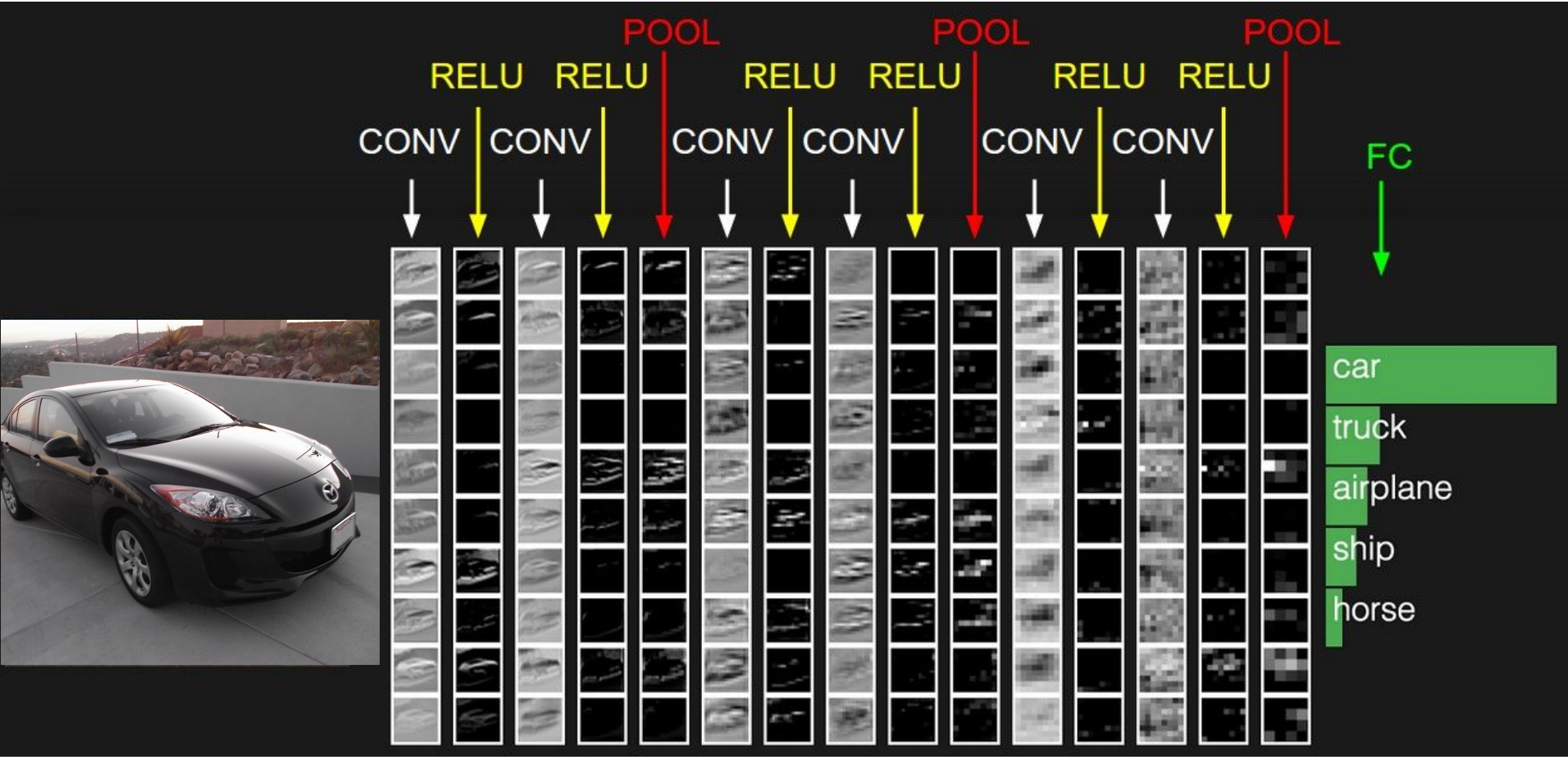
Figure copyright Andrej Karpathy.

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

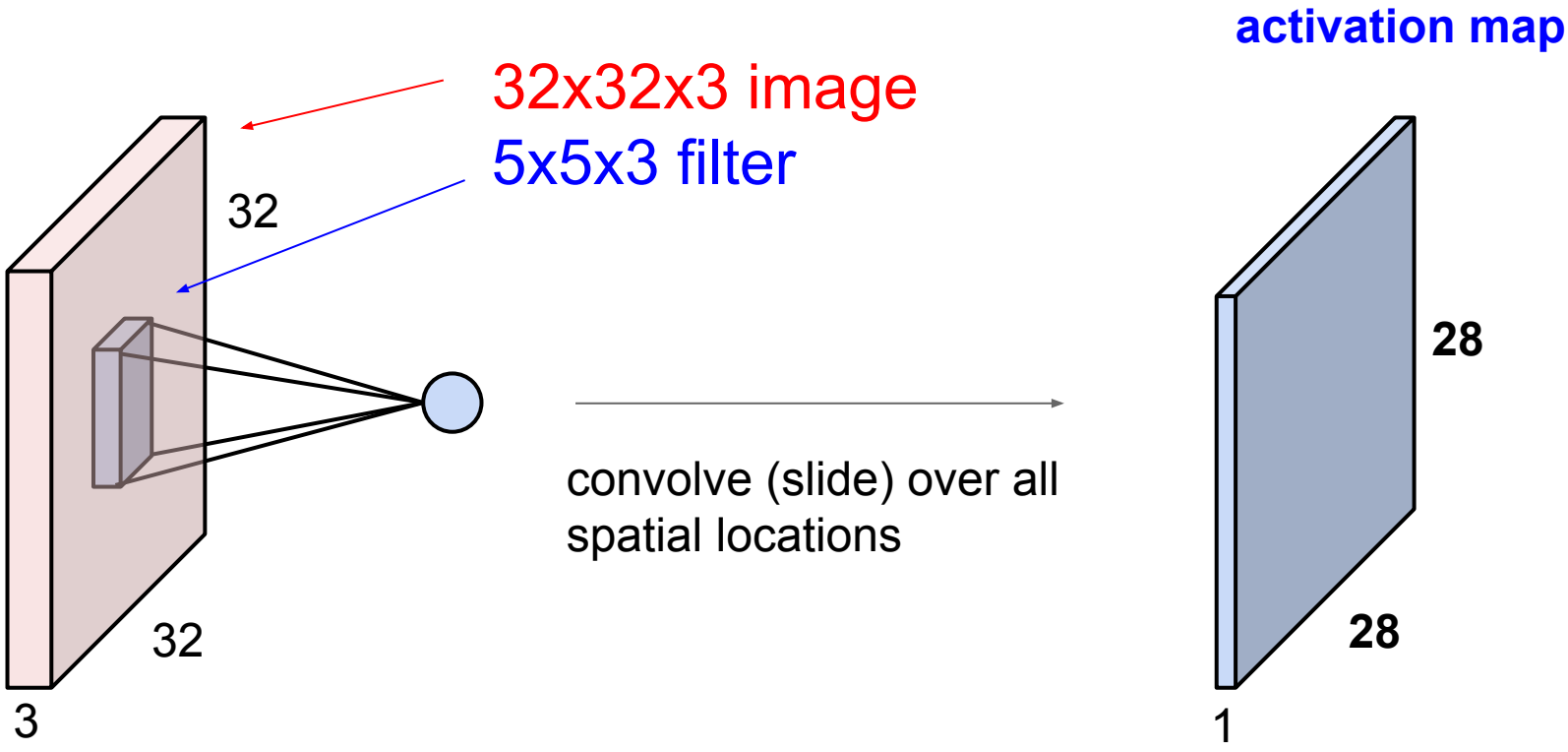
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

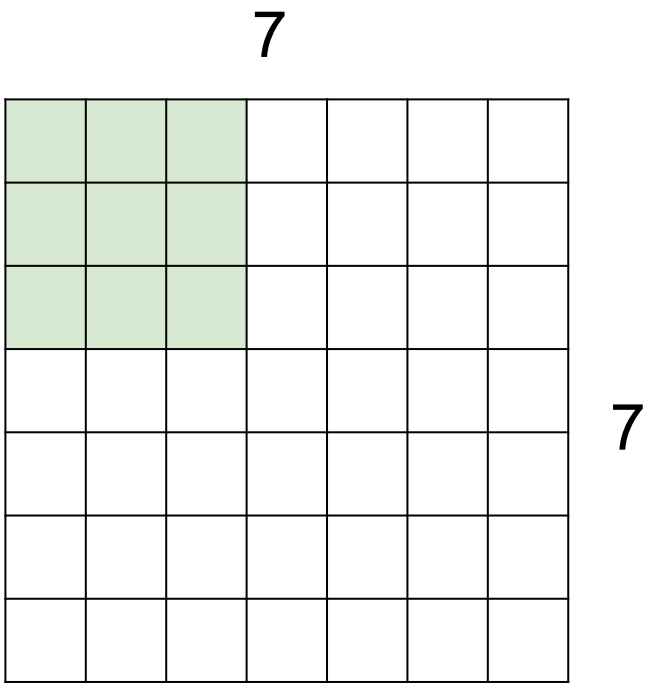
preview:


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

A closer look at spatial dimensions:

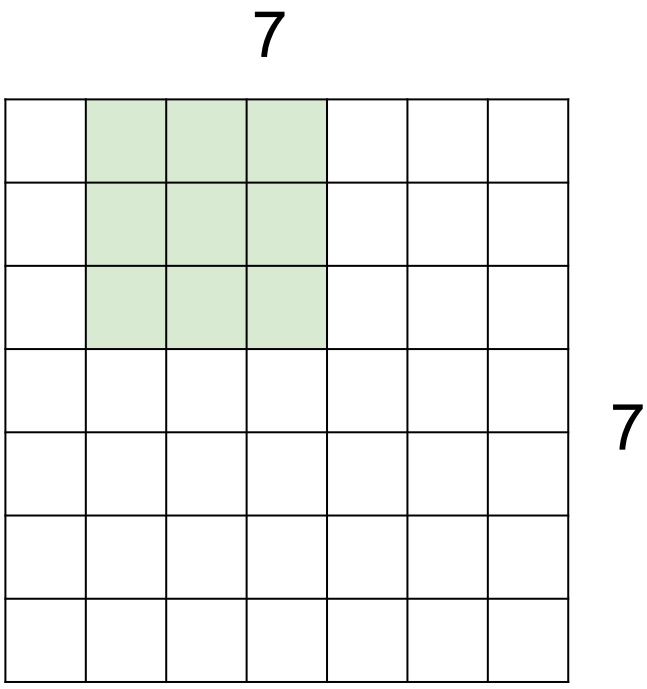


A closer look at spatial dimensions:



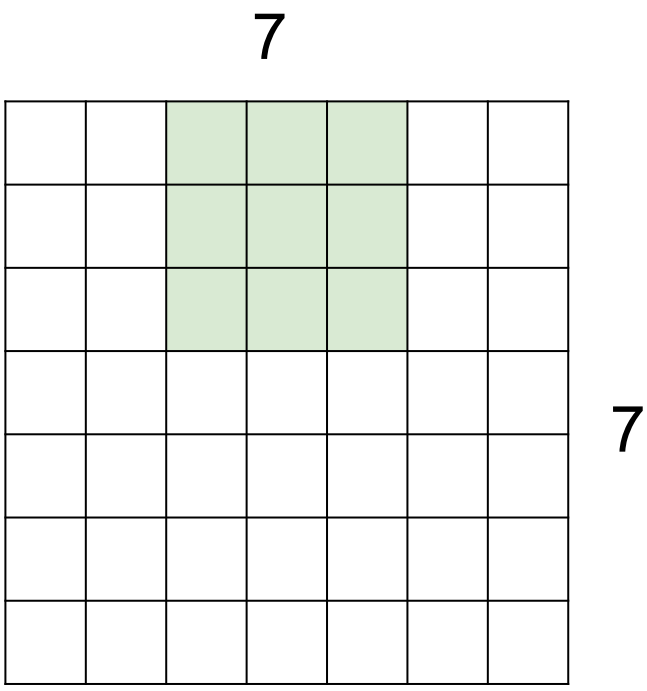
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



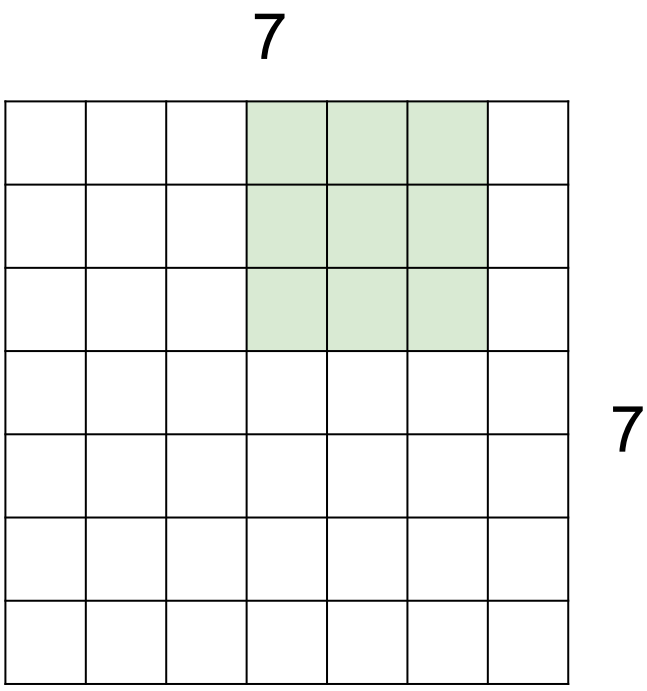
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



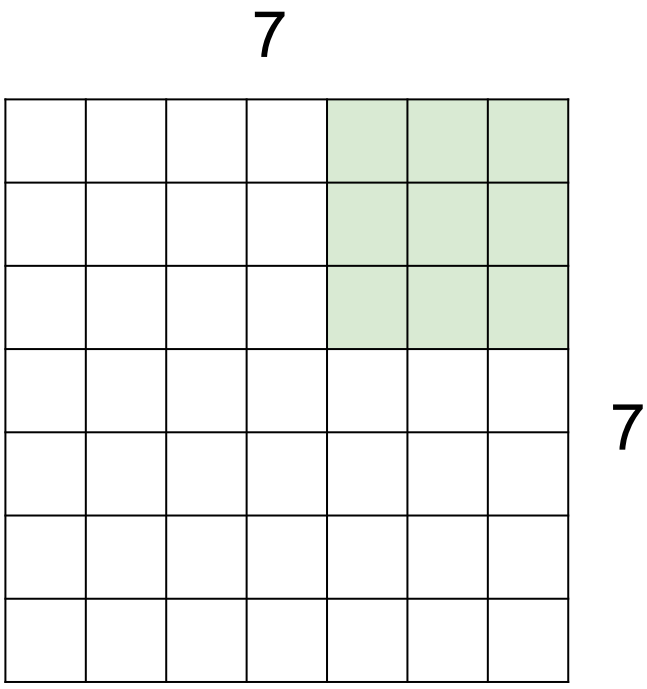
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

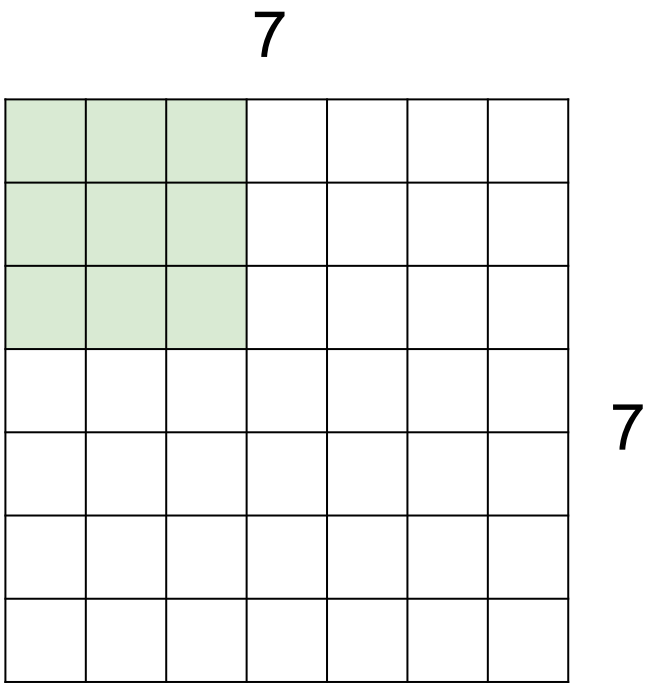
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

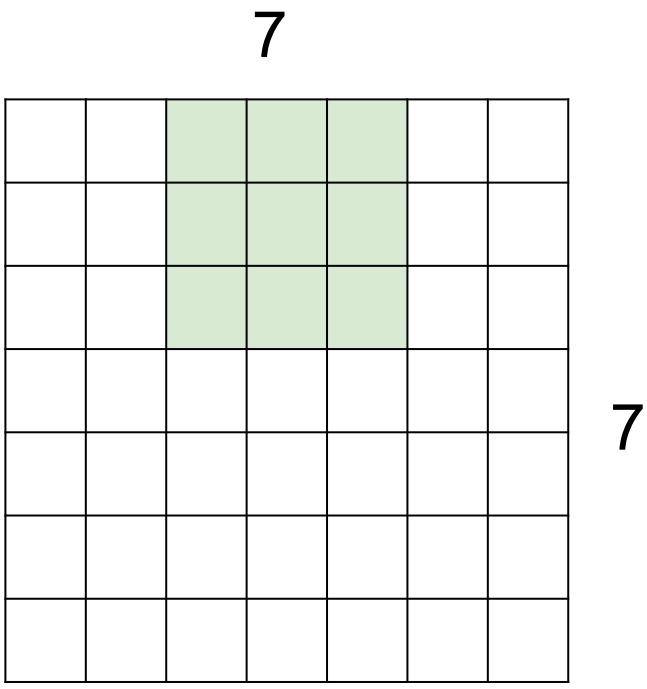
=> 5x5 output

A closer look at spatial dimensions:



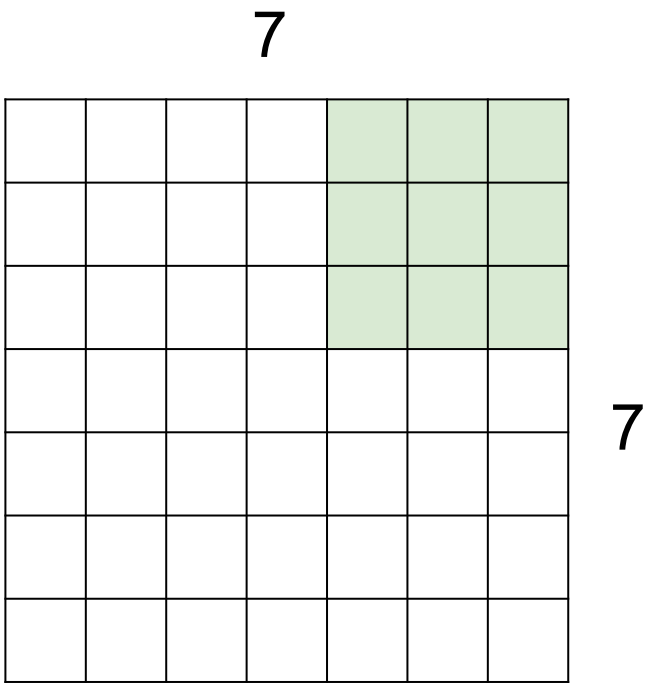
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



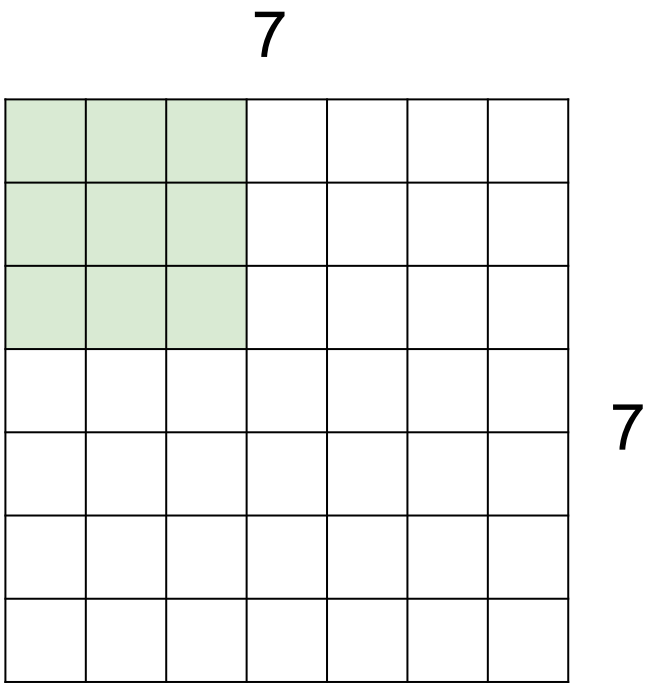
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



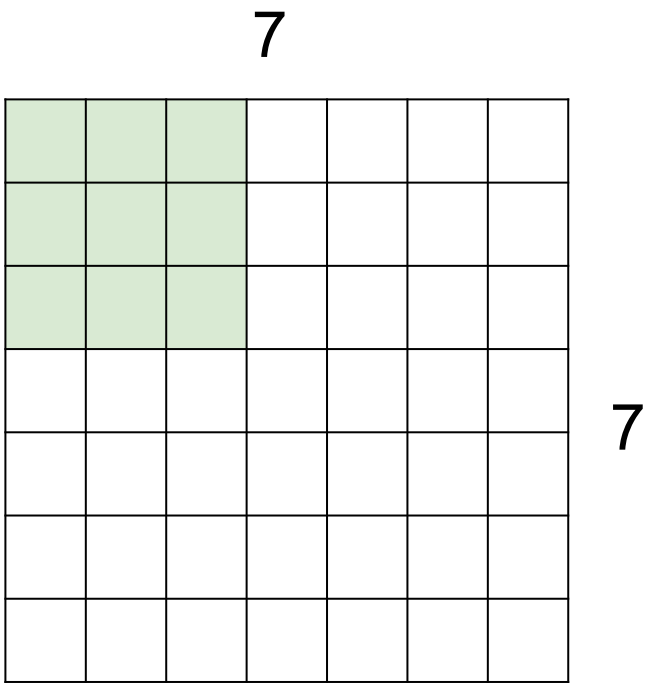
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



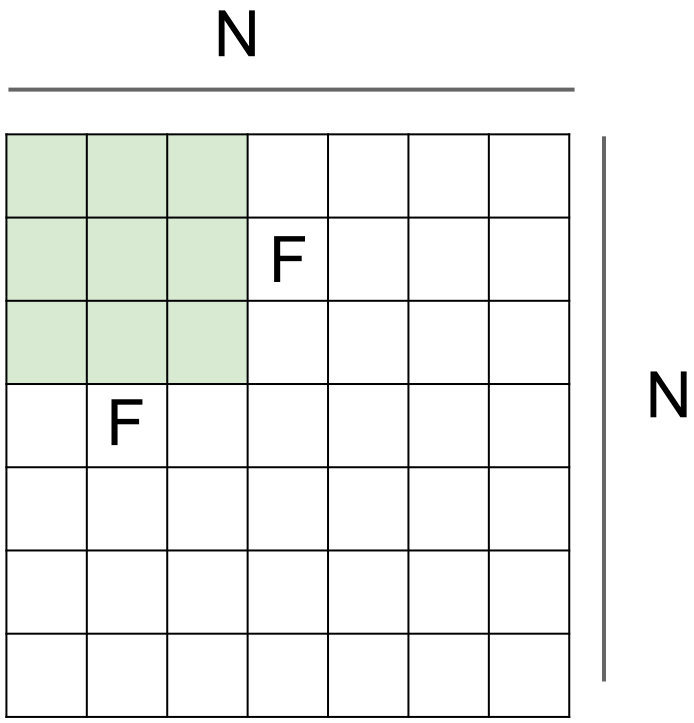
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

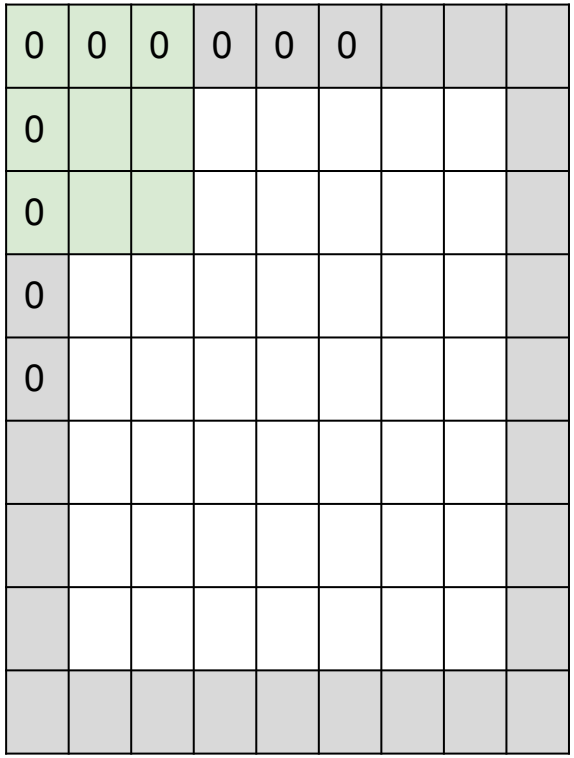
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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

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

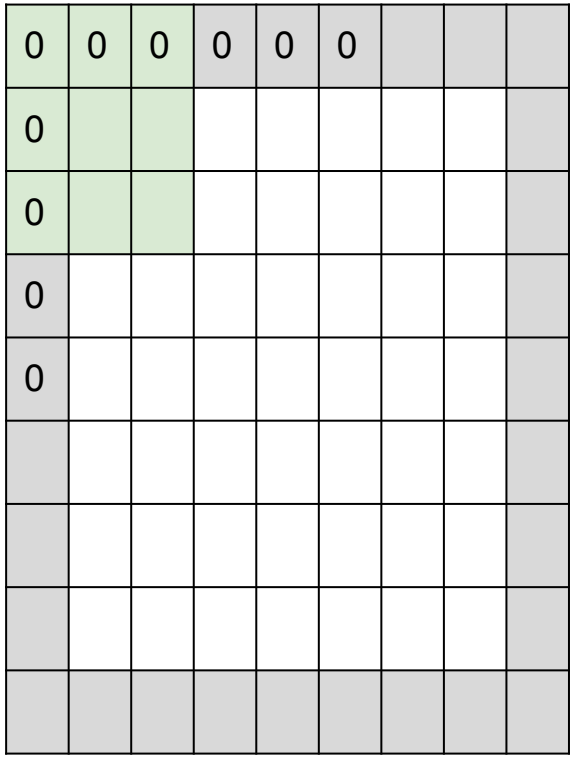
In practice: Common to zero pad the border



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

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

In practice: Common to zero pad the border

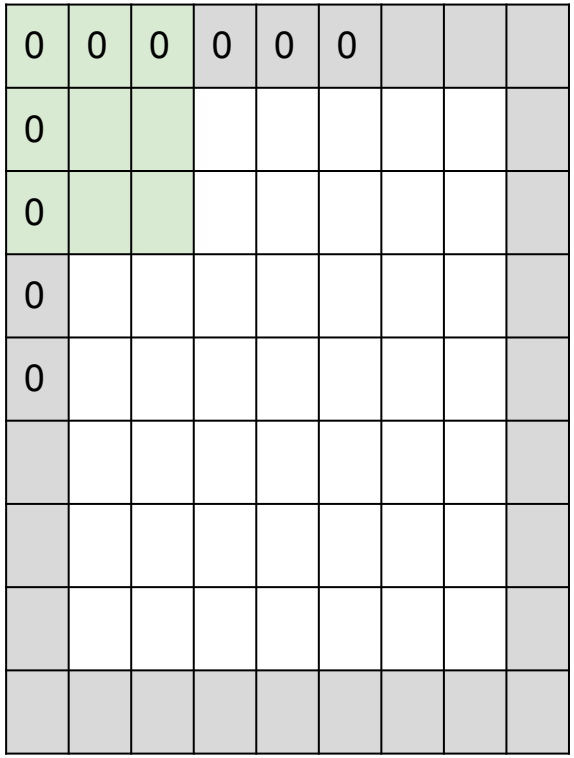


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

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

In practice: Common to zero pad the border

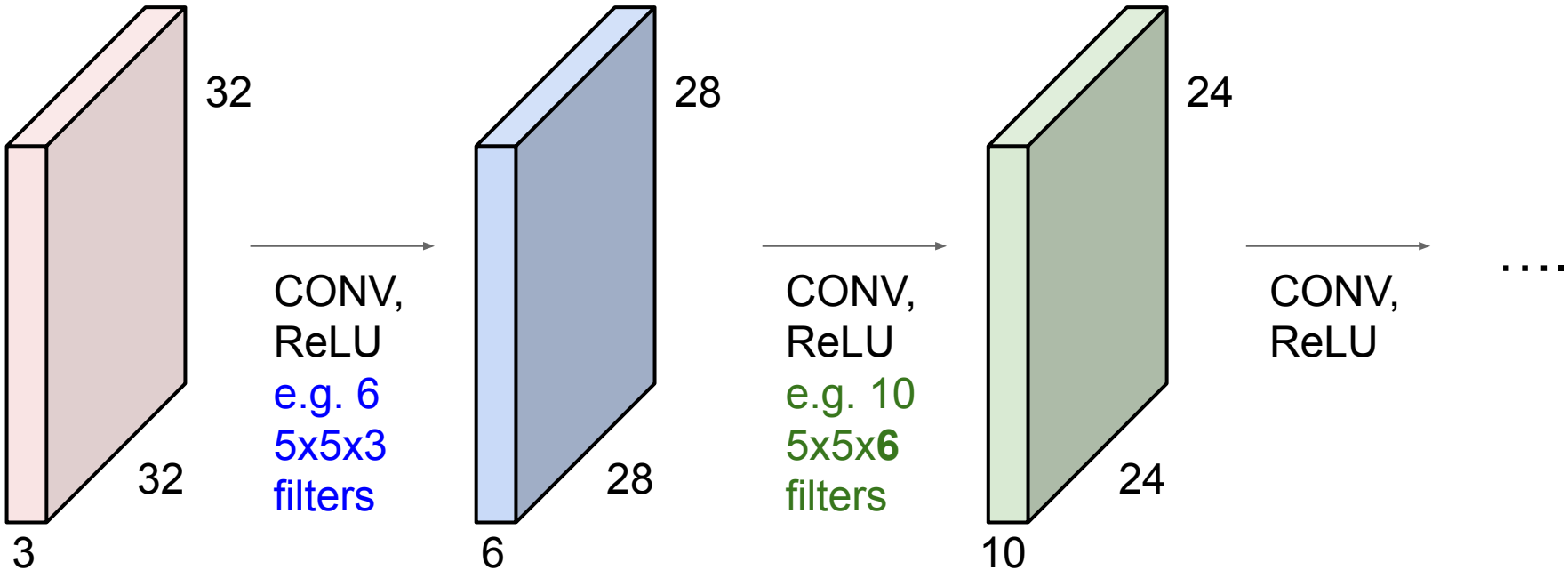


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

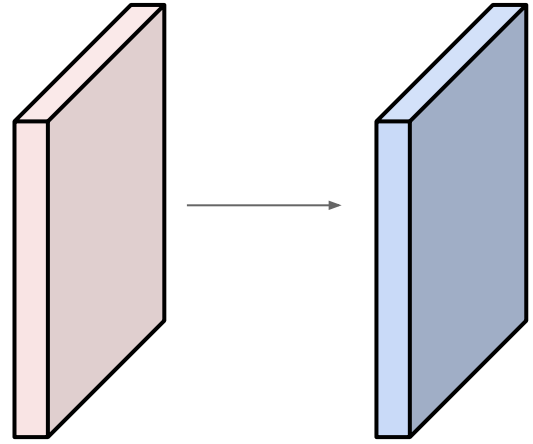
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to…

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



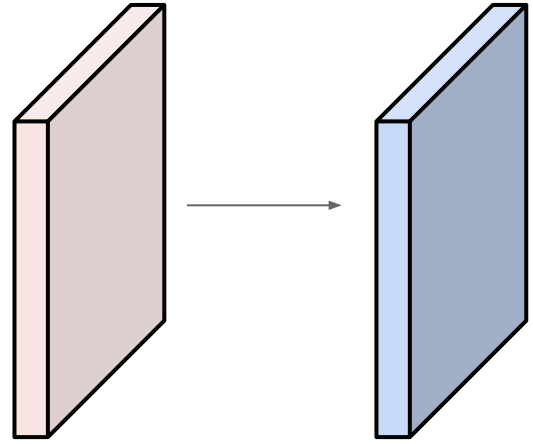
Examples time:



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

Output volume size: ?

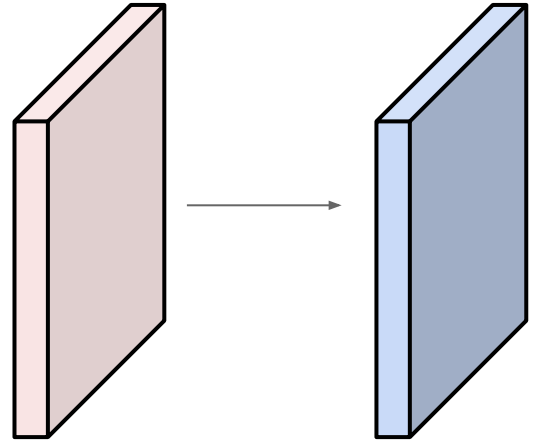
Examples time:



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

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

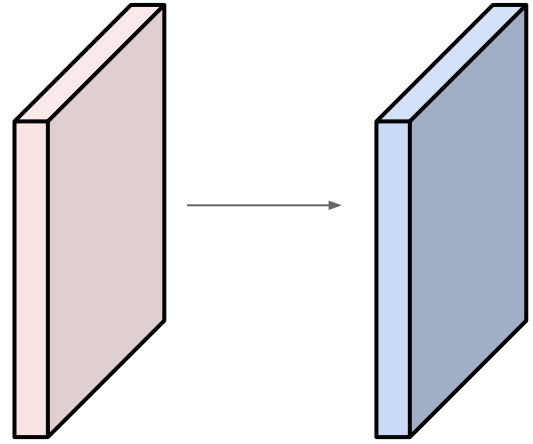
Examples time:



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

Number of parameters in this layer?

Examples time:

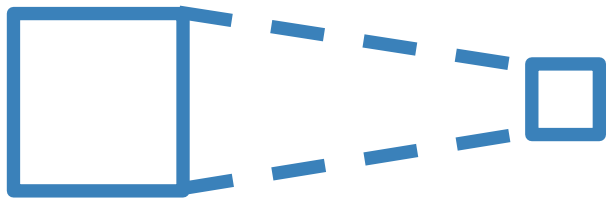


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

Receptive Fields

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

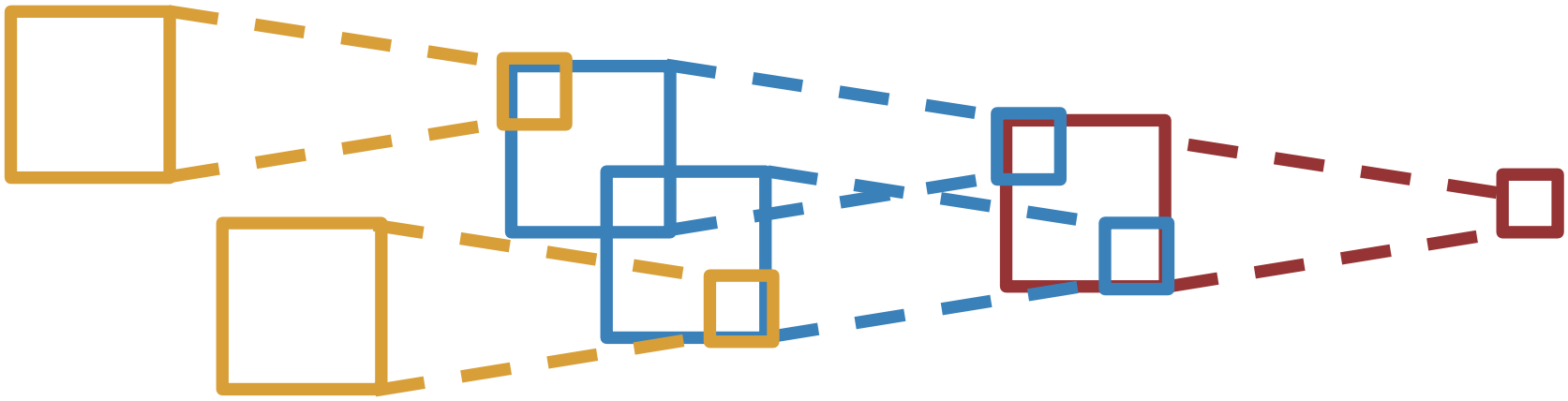


Input Output

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)

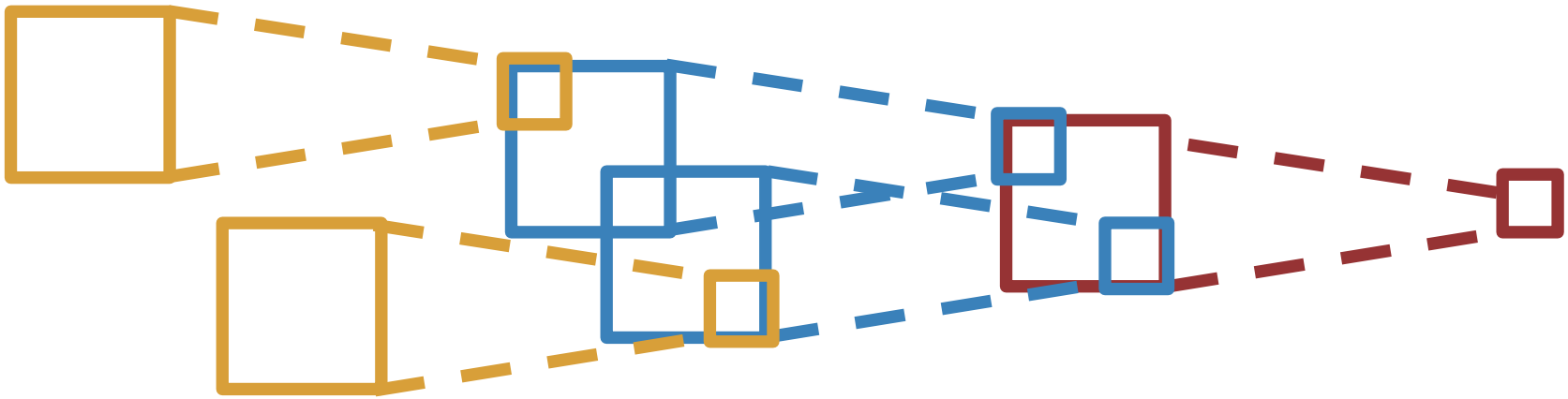


Be careful – ”receptive field in the input” vs. “receptive field in the previous layer”

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)

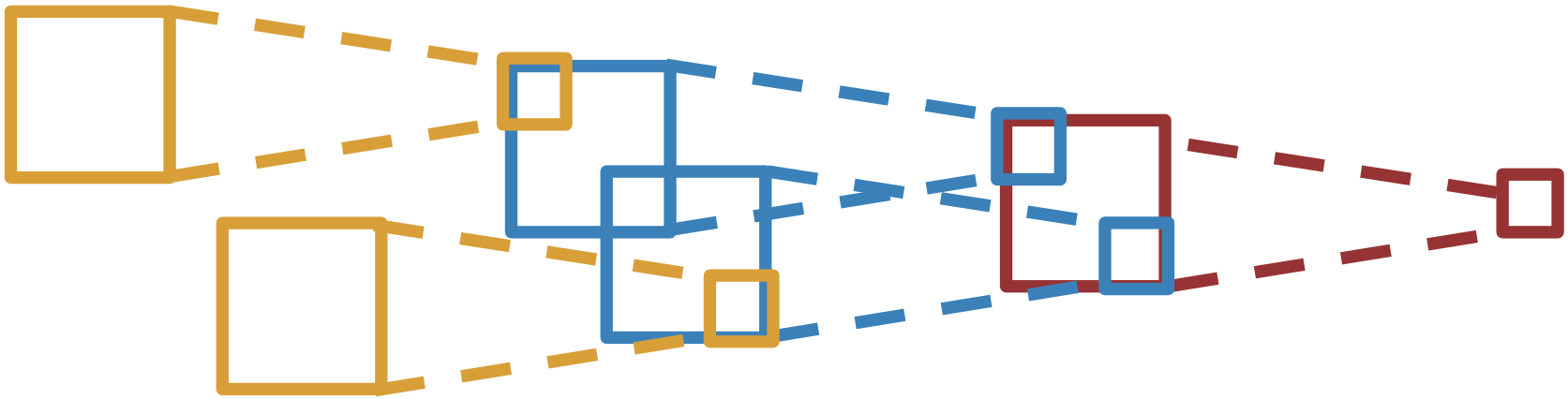


Problem: For large images we need many layers for each output to “see” the whole image image

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

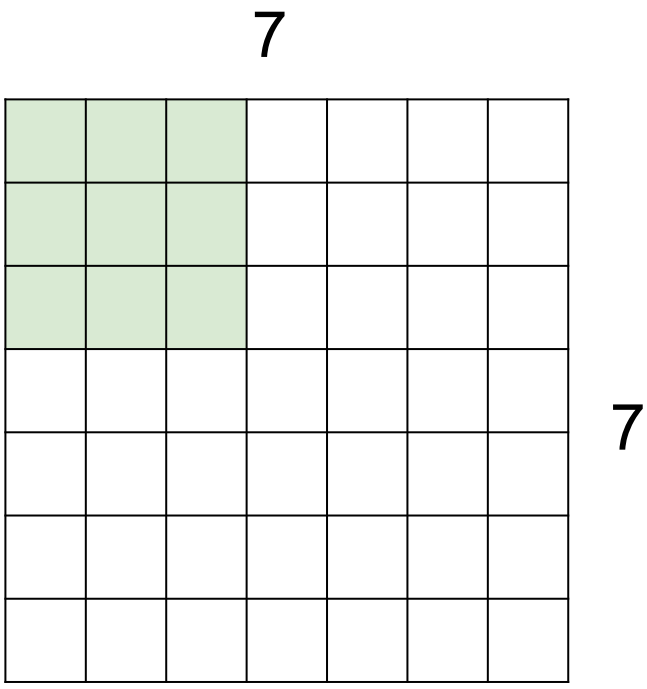
Output

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

Solution: Downsample inside the network

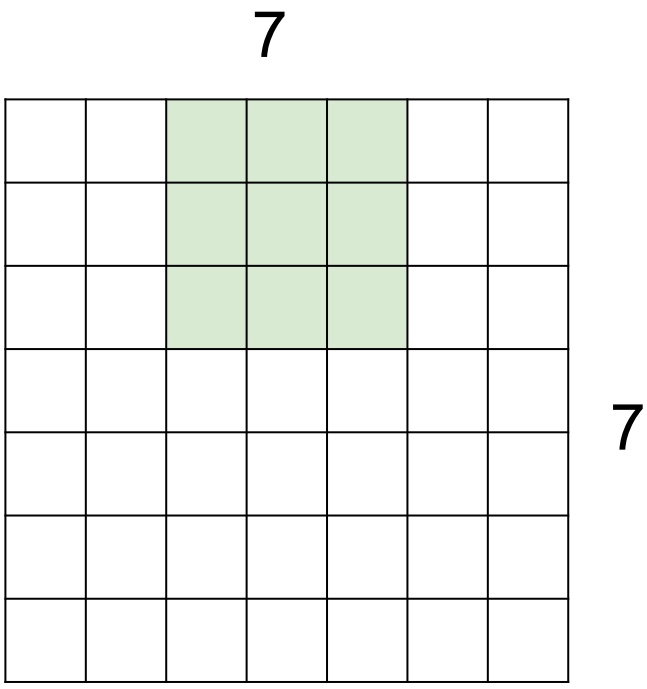
Slide inspiration: Justin Johnson

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**

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

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

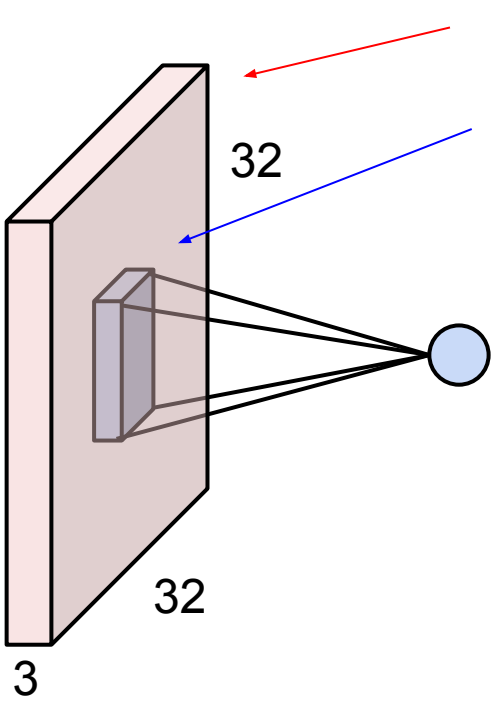
Number of parameters: F^2CK and K biases

Common settings:

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

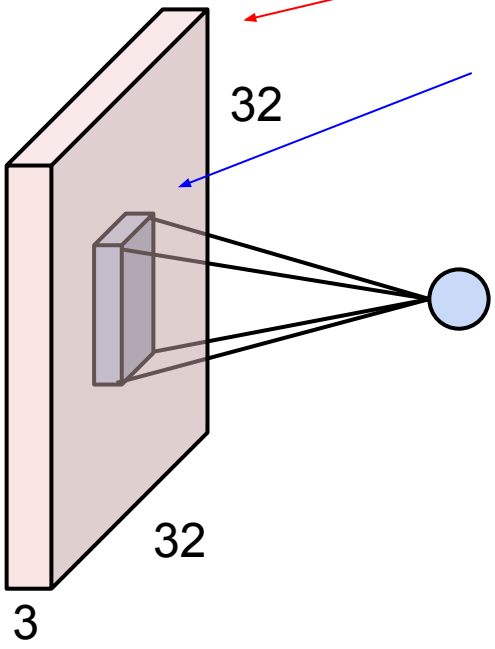
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

The brain/neuron view of CONV Layer



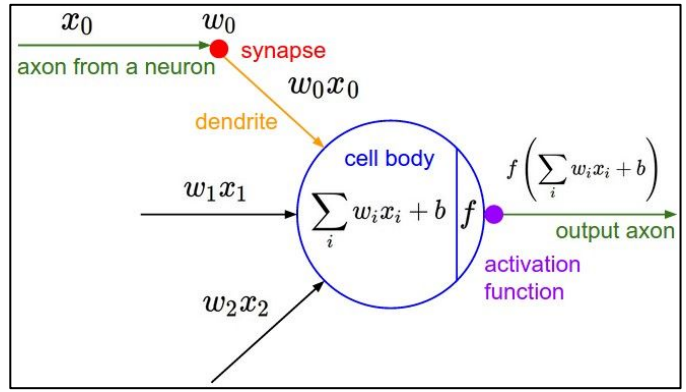
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer



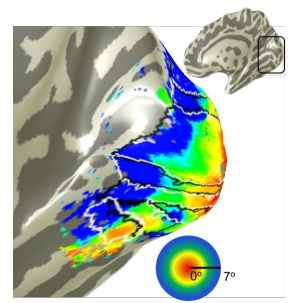
32x32x3 image

5x5x3 filter

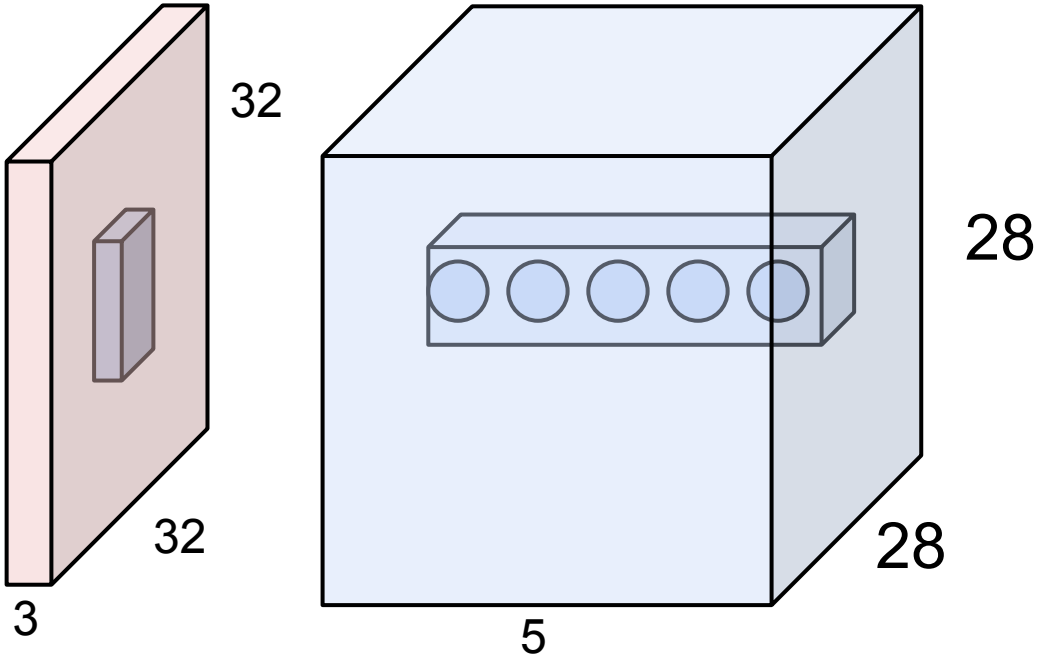


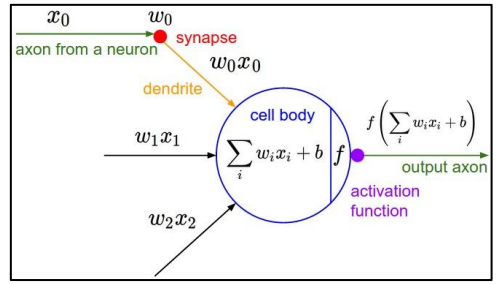
It's just a neuron with local connectivity...

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



The brain/neuron view of CONV Layer





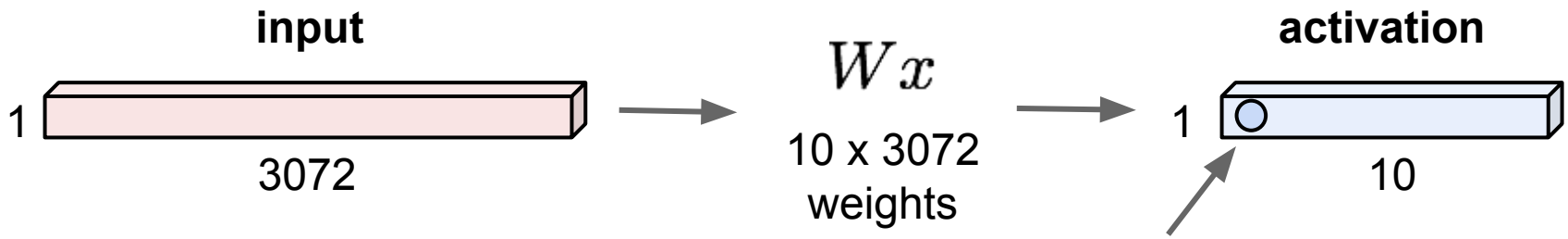
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

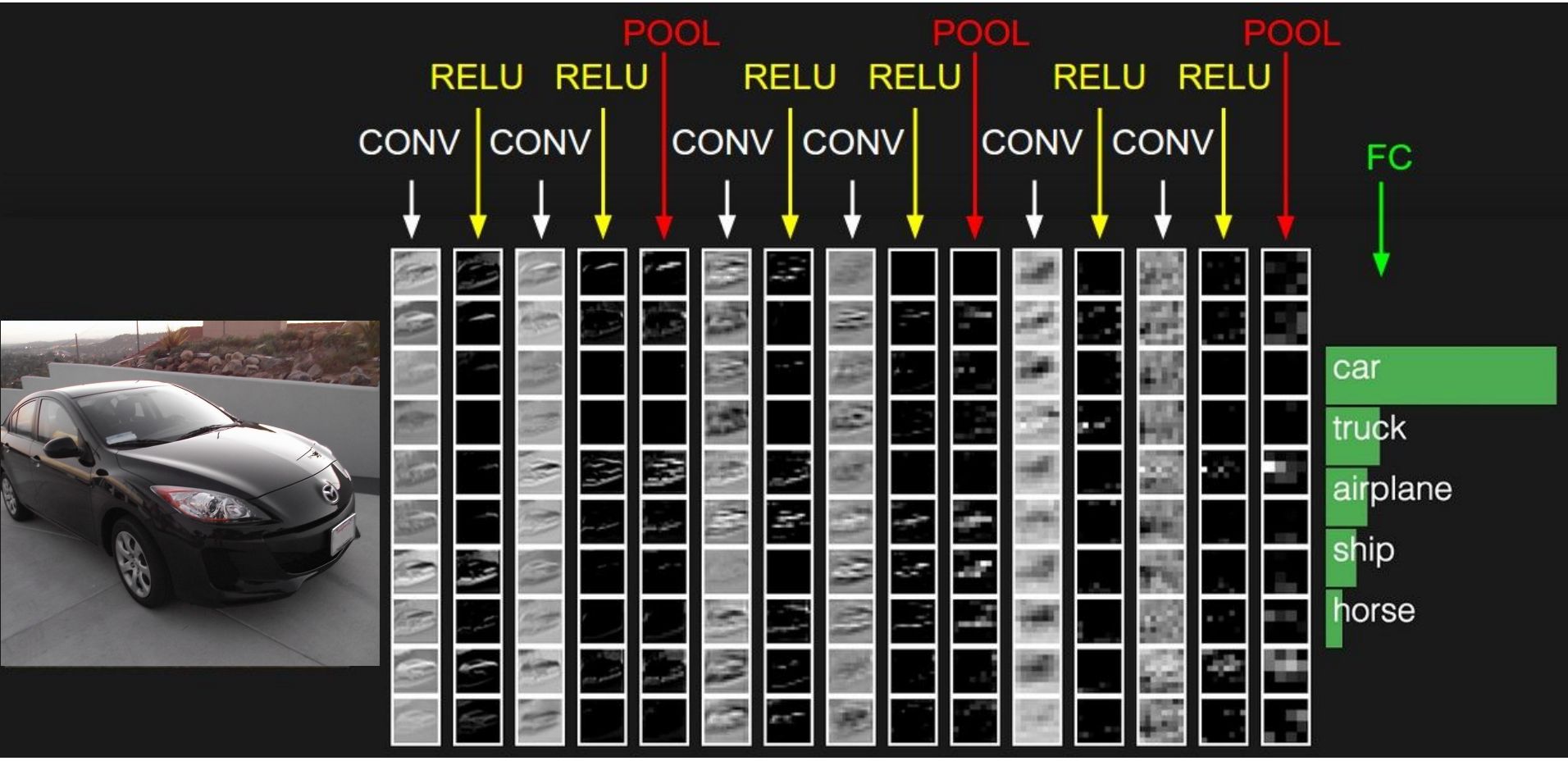
There will be 5 different neurons all looking at the same region in the input volume

Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

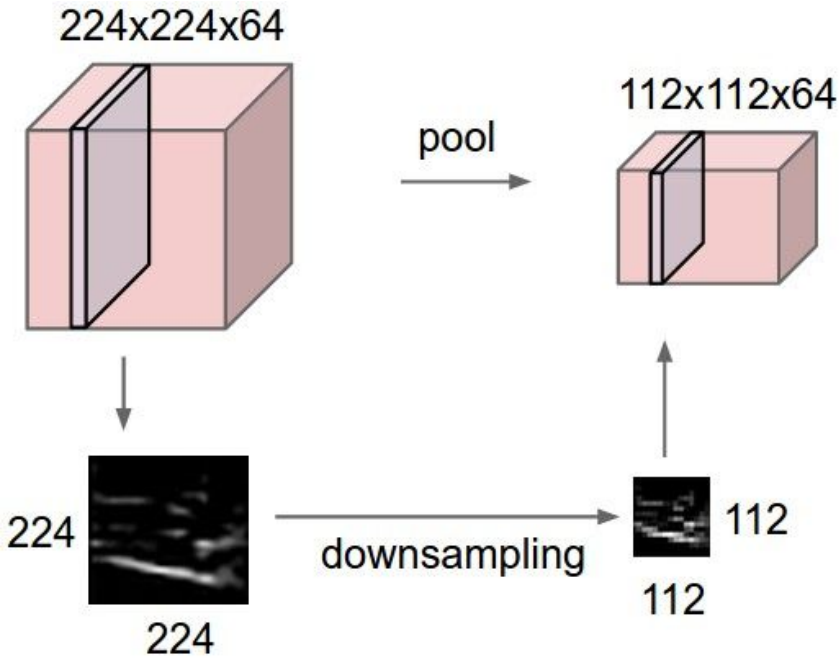
Each neuron looks at the full input volume



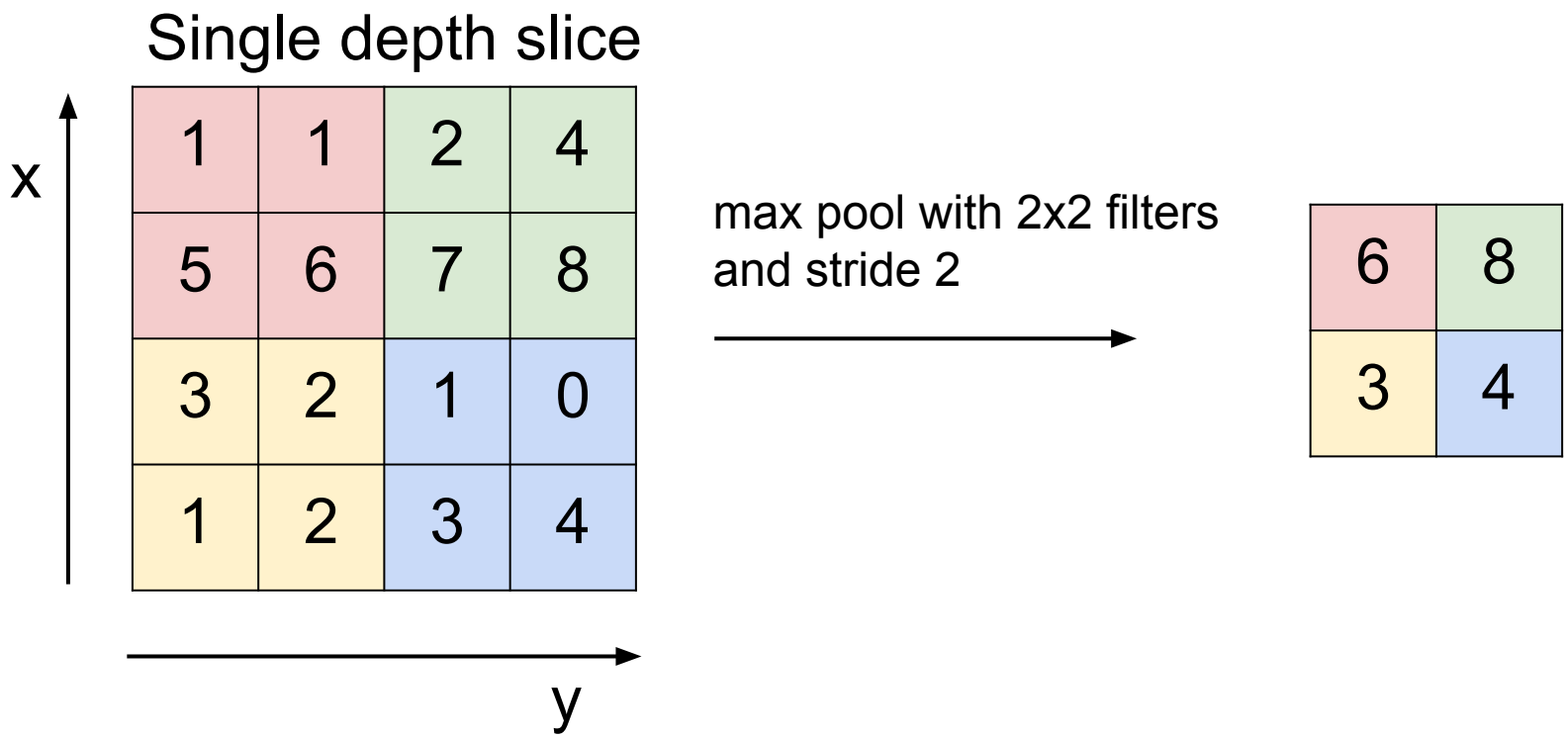

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Pooling layer

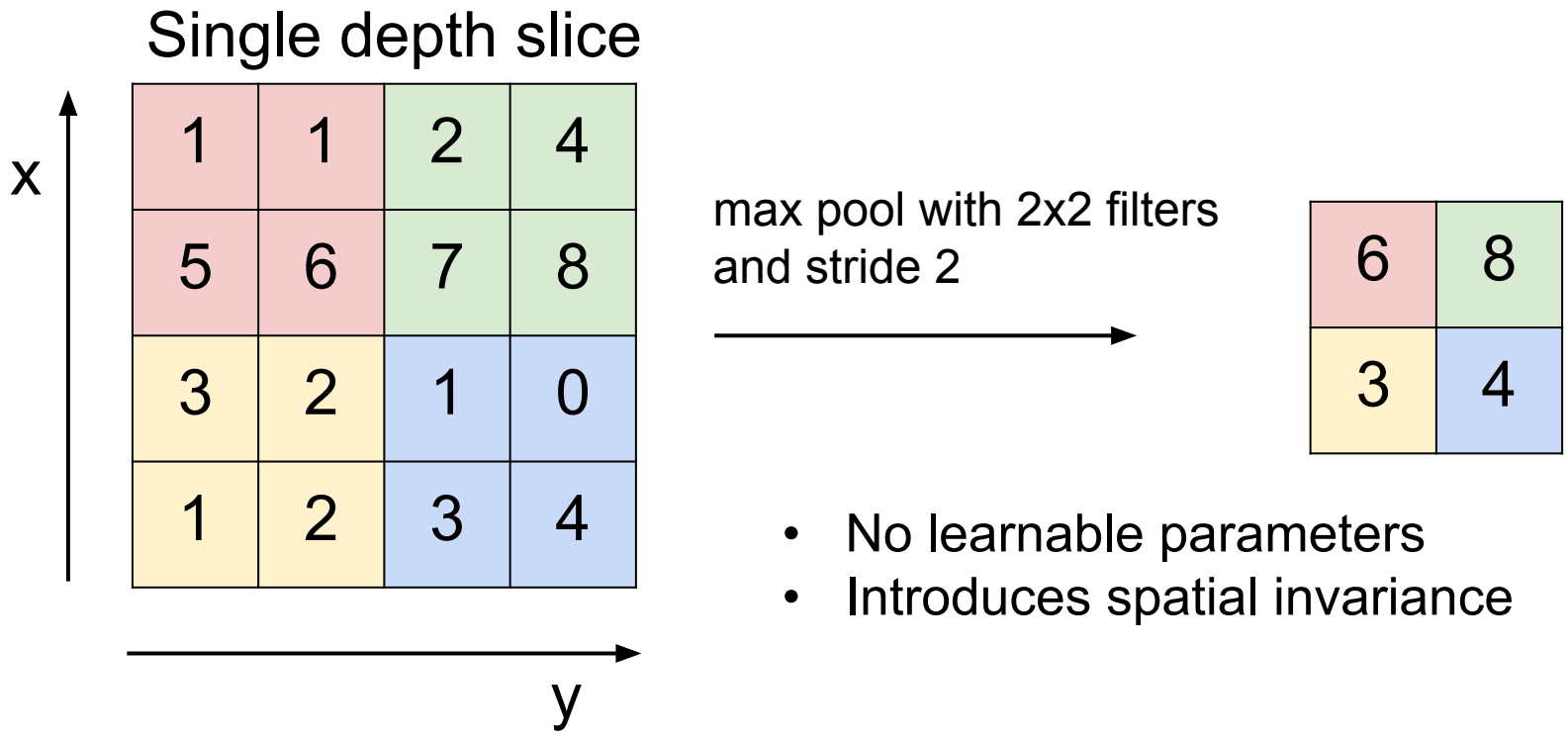
- makes the representations smaller and more manageable
- operates over each activation map independently



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



- 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**

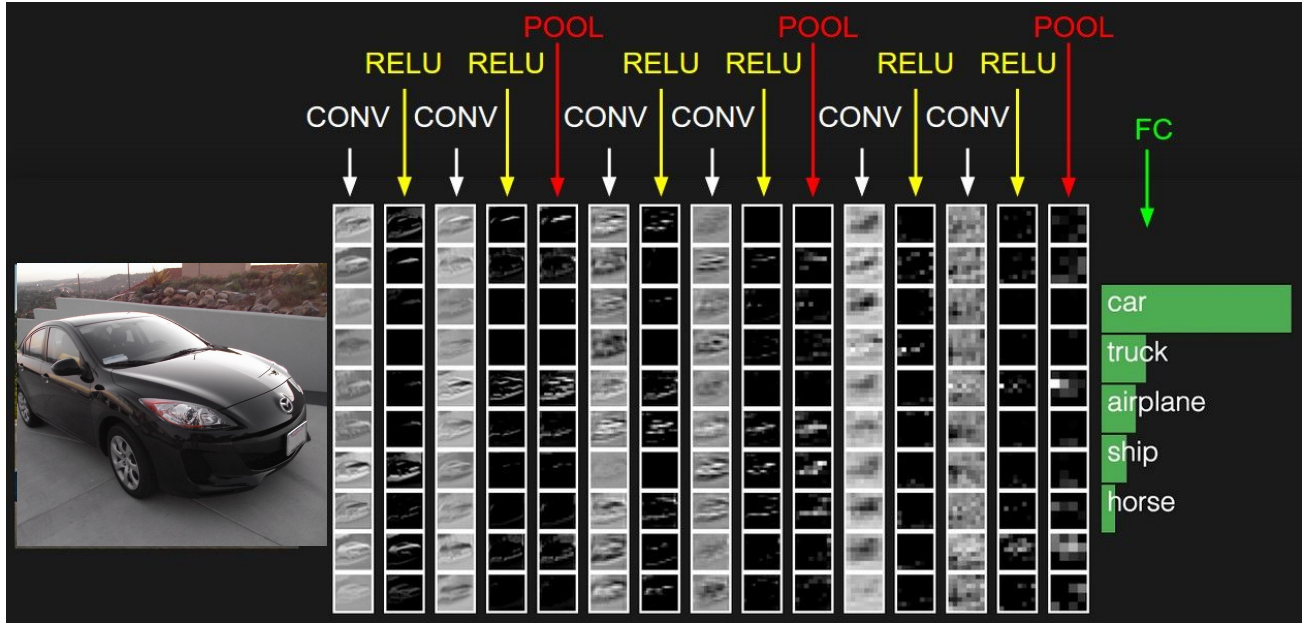
This will produce an output of W_2 x H_2 x C where:

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

Number of parameters: 0

Fully Connected Layer (FC layer)

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



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

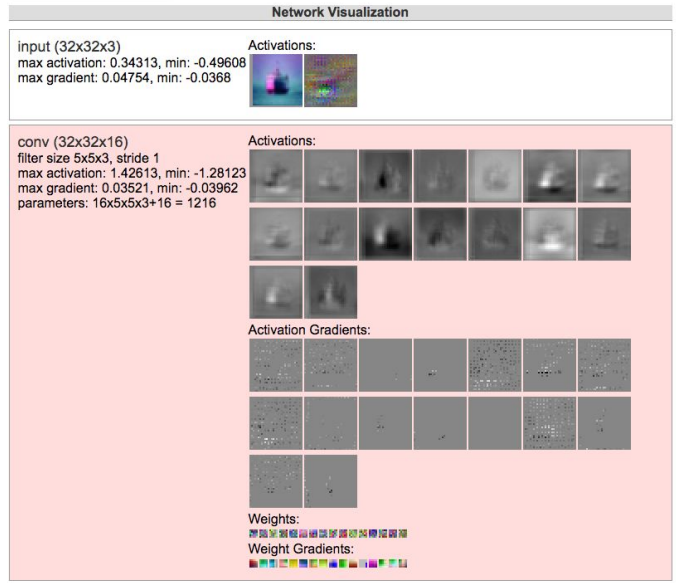
Description

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

Thank you

Prof. Jeon, Sangryul

Computer Vision Lab.

Pusan National University, Korea

Tel: +82-51-510-2423

Web: http://sr-jeon.github.io/

E-mail: srjeonn@pusan.ac.kr