Convolutional Neural Networks

M. Soleymani

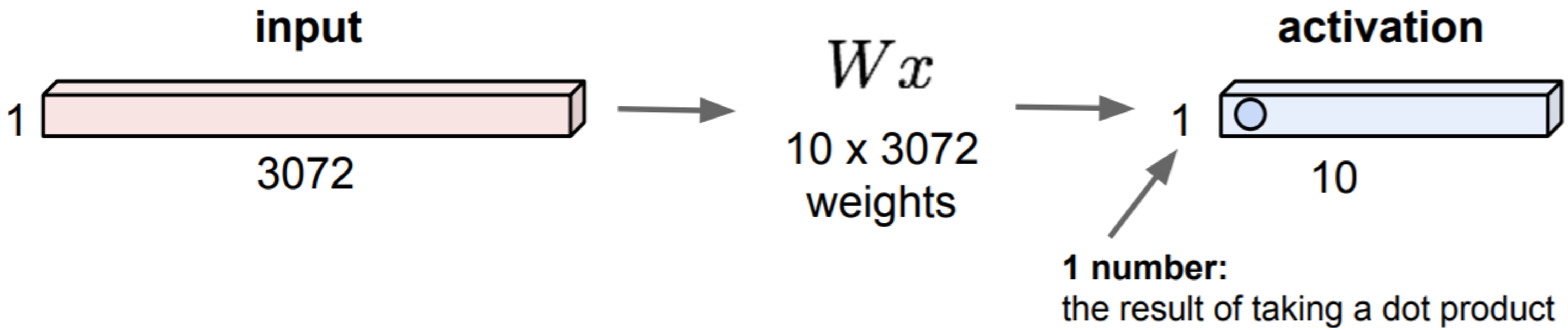
Sharif University of Technology

Fall 2017

Slides have been adopted from Fei Fei Li and colleagues lectures and notes, cs231n, Stanford 2017.

Fully connected layer

32x32x3 image -> stretch to 3072 x 1



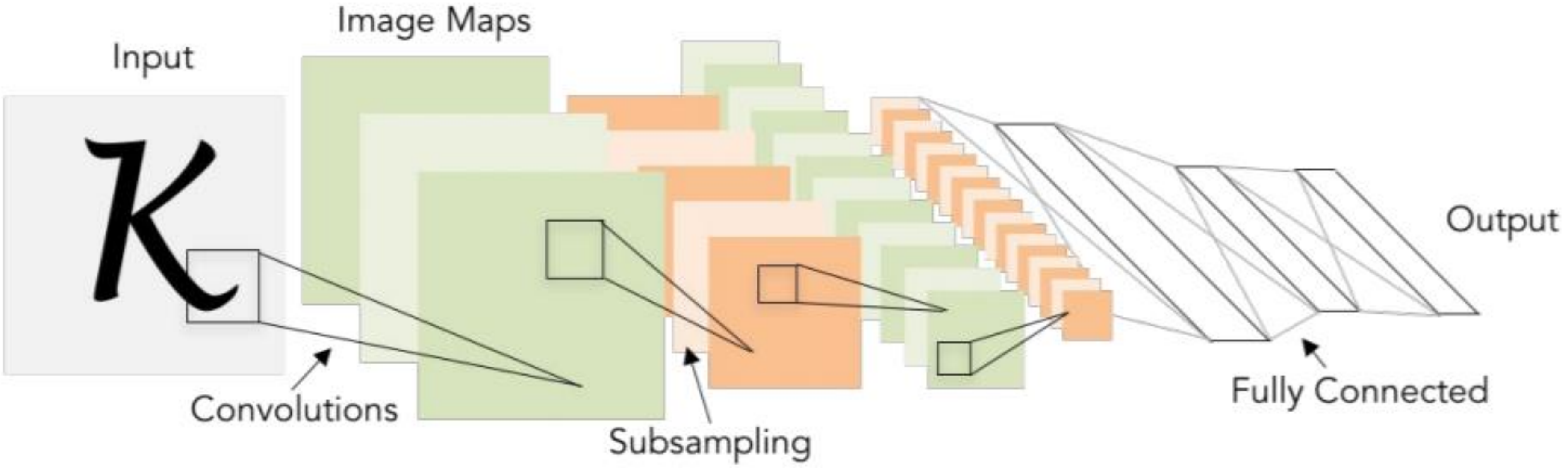
Fully connected layers

- Neurons in a single layer function completely independently and do not share any connections.

- Regular Neural Nets don’t scale well to full images
  - parameters would add up quickly!
  - full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

LeNet

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

AlexNet

[Krizhevsky, Sutskever, Hinton, 2012]

- ImageNet Classification with Deep Convolutional Neural Networks

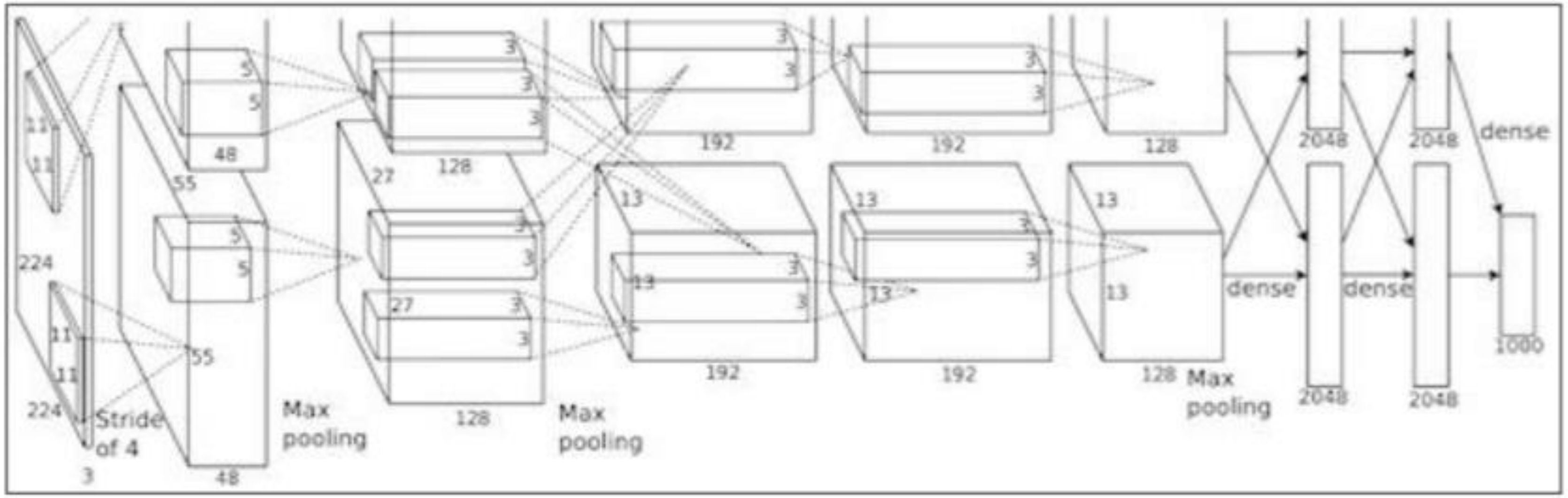


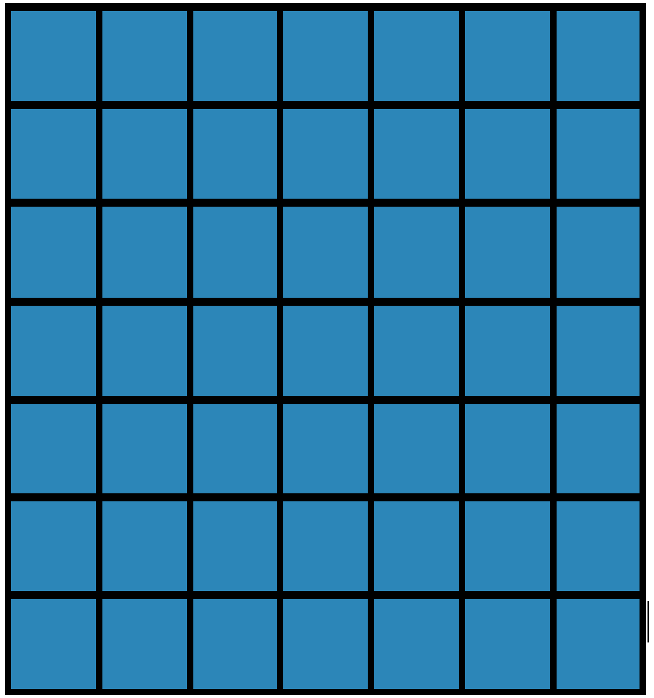
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Layers used to build ConvNets

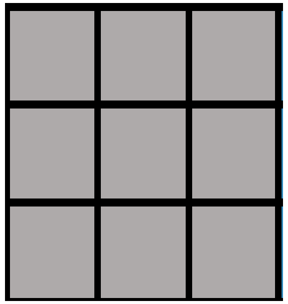
- Three main types of layers
  - **Convolutional Layer**
    - output of neurons are connected to local regions in the input
    - applying the same filter on the whole image
    - CONV layer’s parameters consist of a set of learnable filters.
  - **Pooling Layer**
    - perform a downsampling operation along the spatial dimensions
  - **Fully-Connected Layer**

Convolutional filter

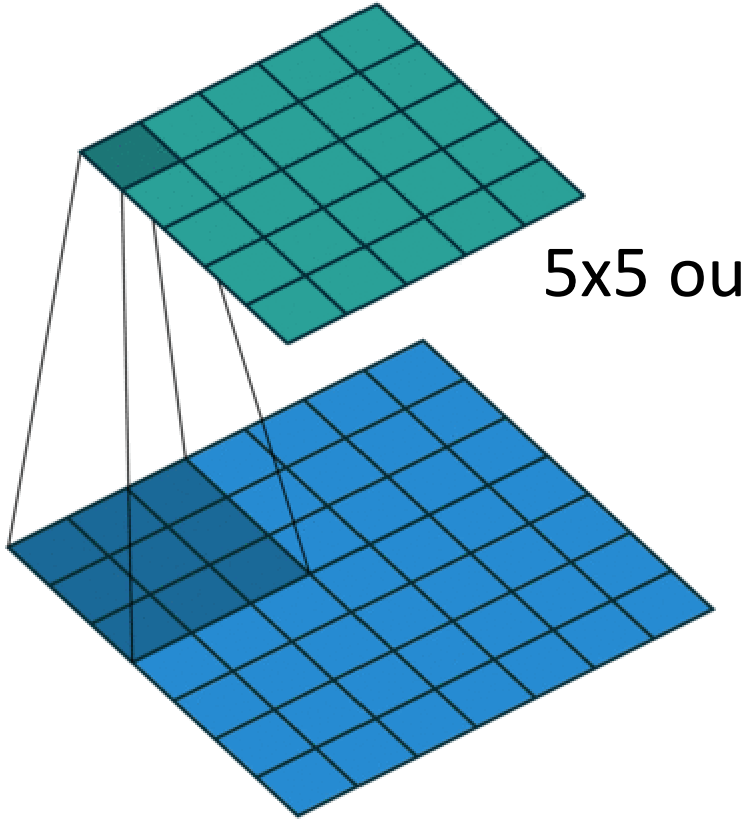
Gives the responses of that filter at every spatial position



7x7 input



3x3 filter

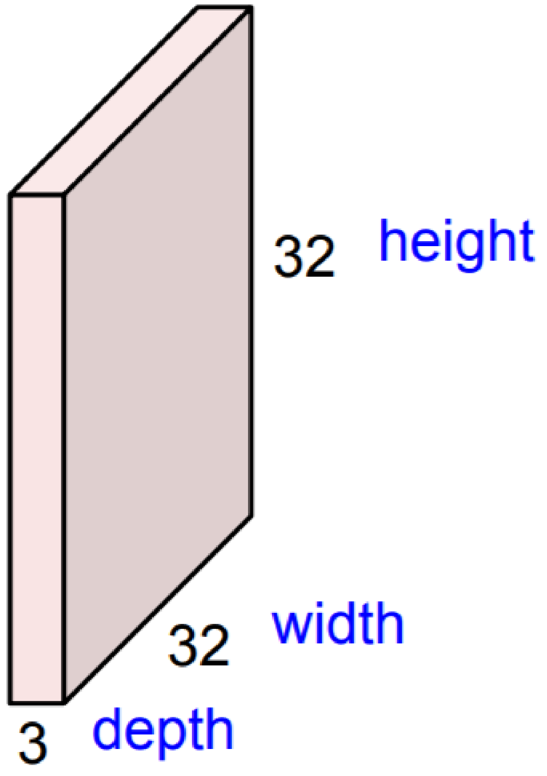


5x5 output

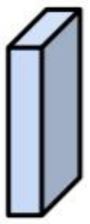
Source:
http://iamaaditya.github.io/2016/03/one-by-one-convolution/

Convolution

32x32x3 image -> preserve spatial structure



5x5x3 filter

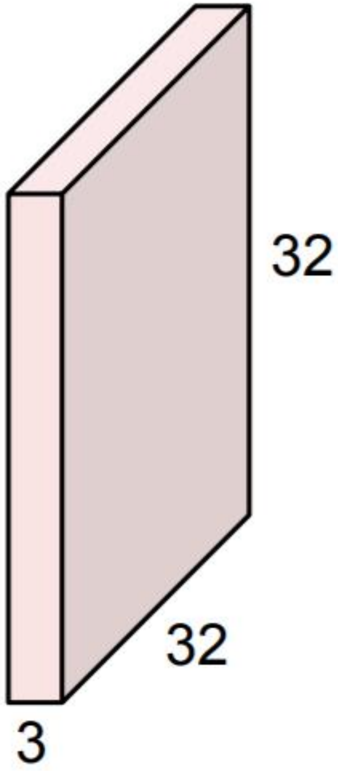


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

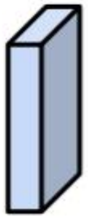
Convolution

Filters always extend the full depth of the input volume

32x32x3 image

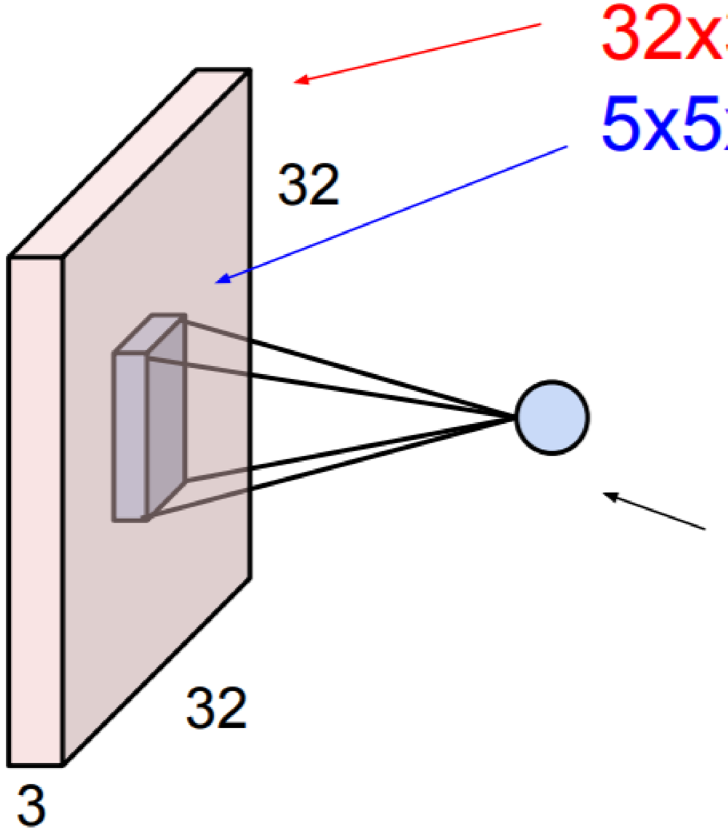


5x5x3 filter



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

Convolution

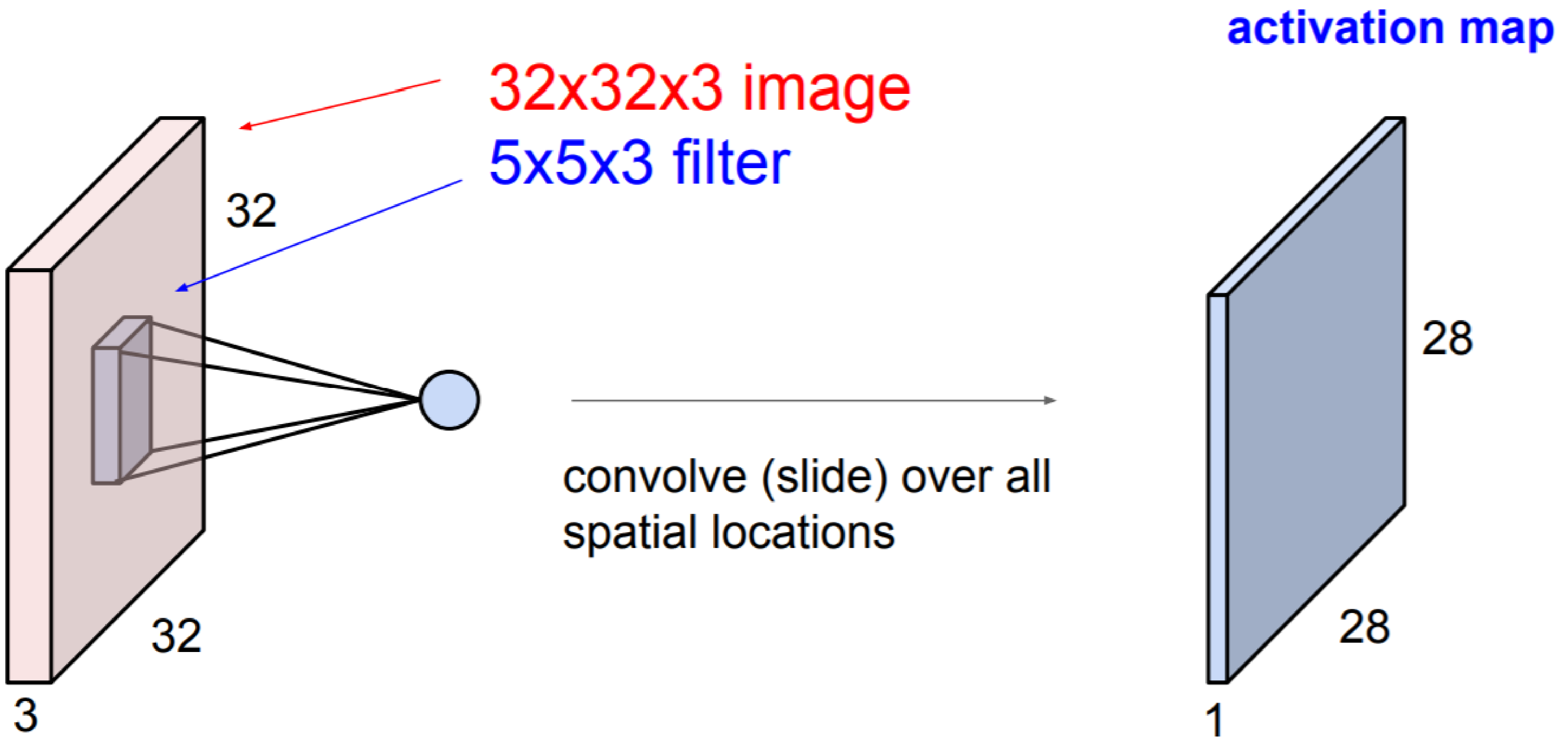


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

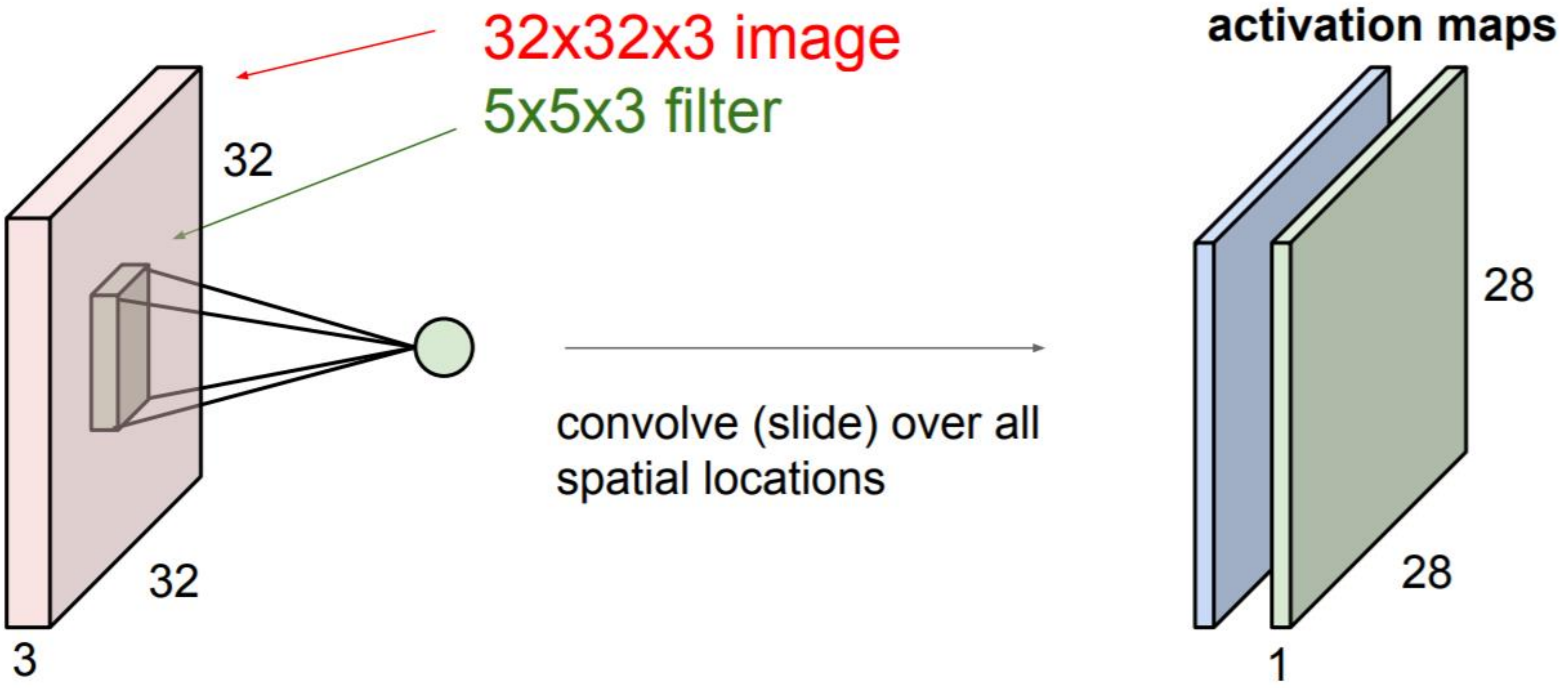
Local connections spatially but full along the entire depth of the input volume.

Convolution



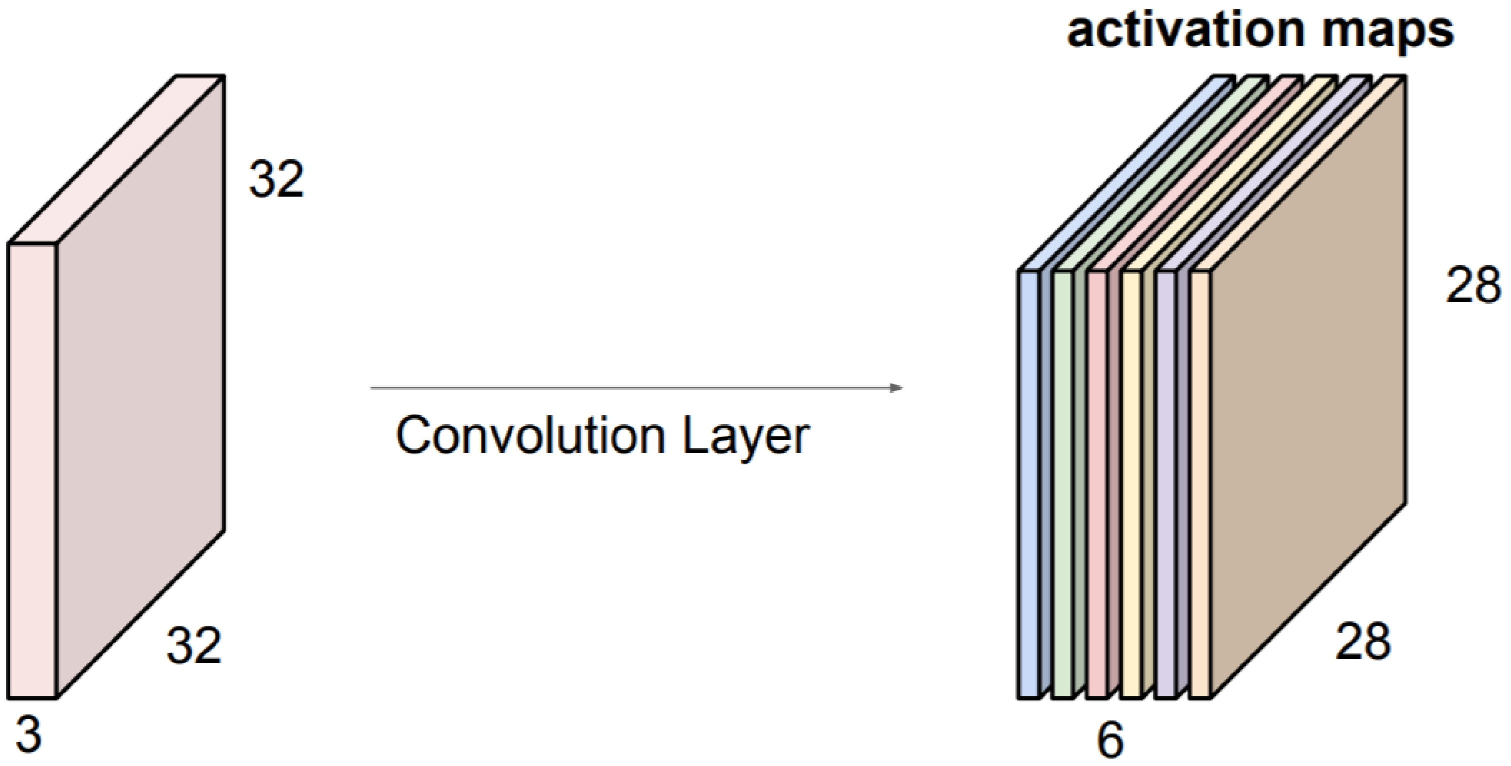
Convolution: Feature maps or activation maps

consider a second, green filter



Convolution: Feature maps or activation maps

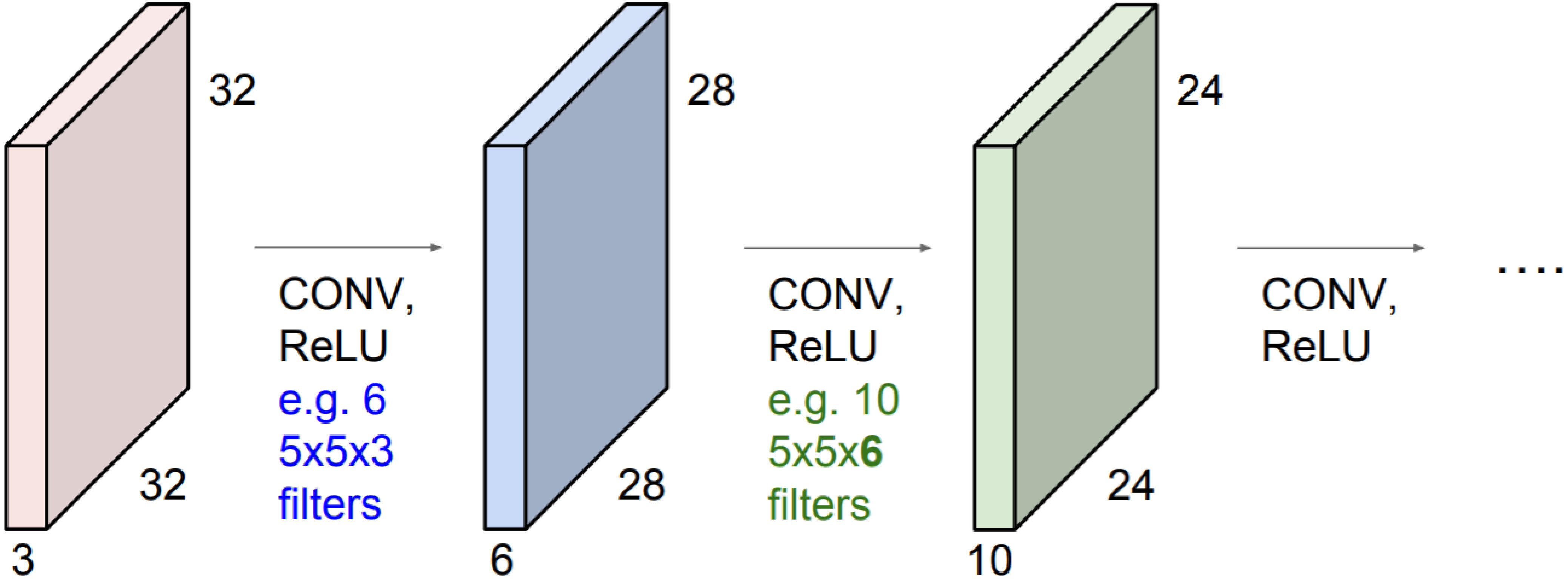
- If we had 6 5x5 filters, we'll get 6 separate activation maps:



- We stack these up to get a "new image" of size 28x28x6!
  - **depth** of the output volume equals to the number of filters

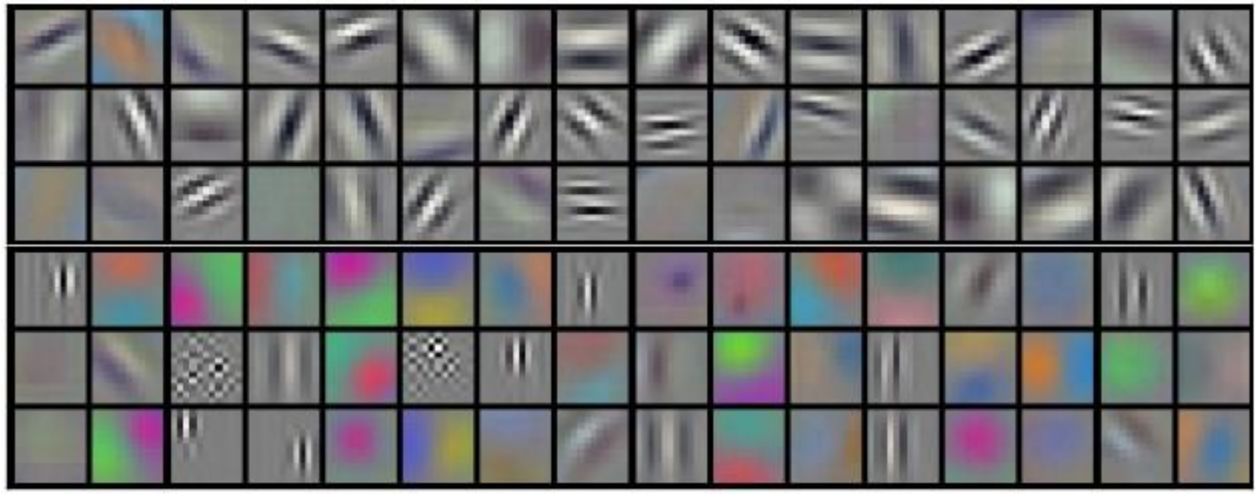
ConvNet

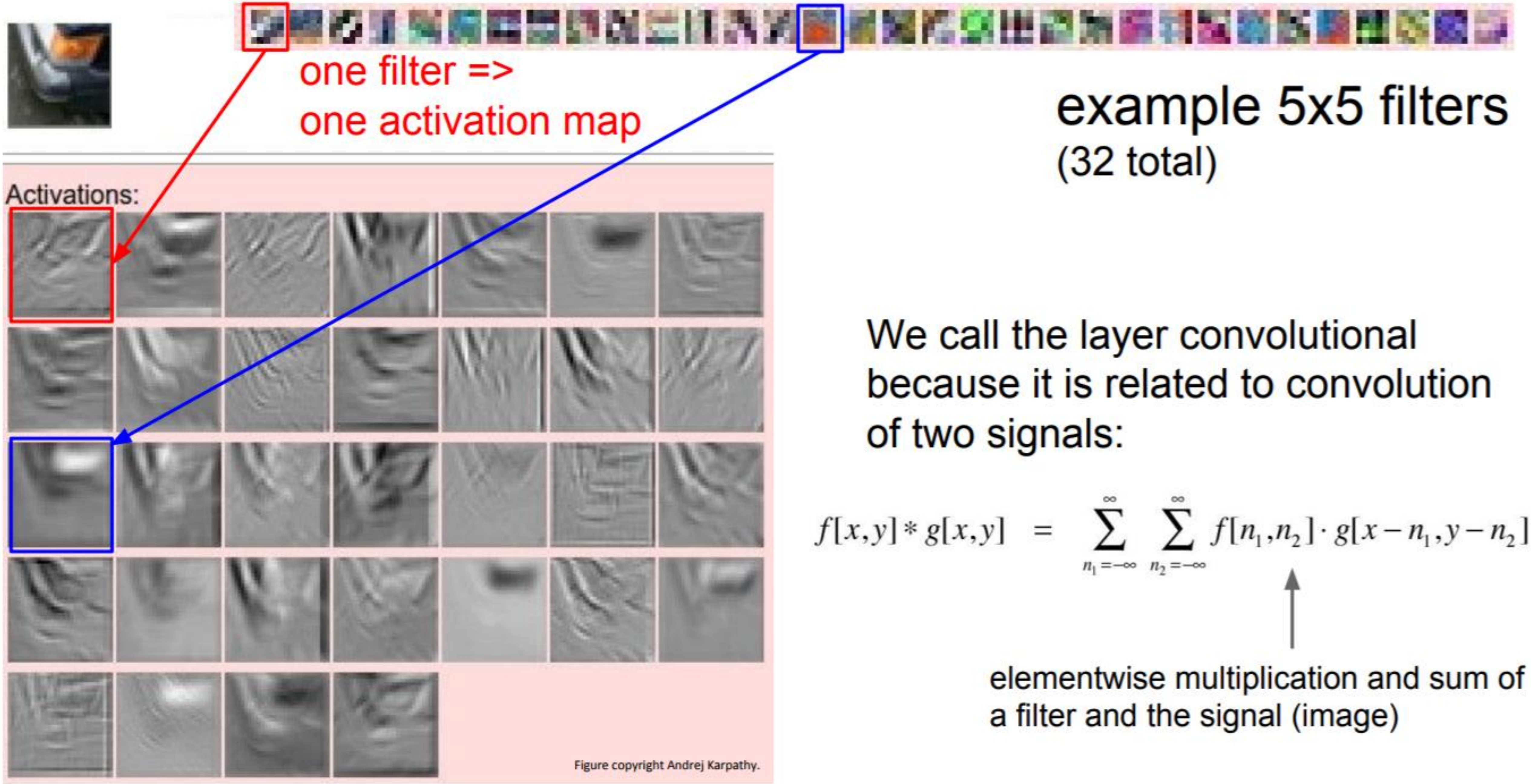
- Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Alexnet: the first layer filters

- filters learned by Krizhevsky et al.
  - Each of the 96 filters shown here is of size [11x11x3]
  - and each one is shared by the 55*55 neurons in one depth slice



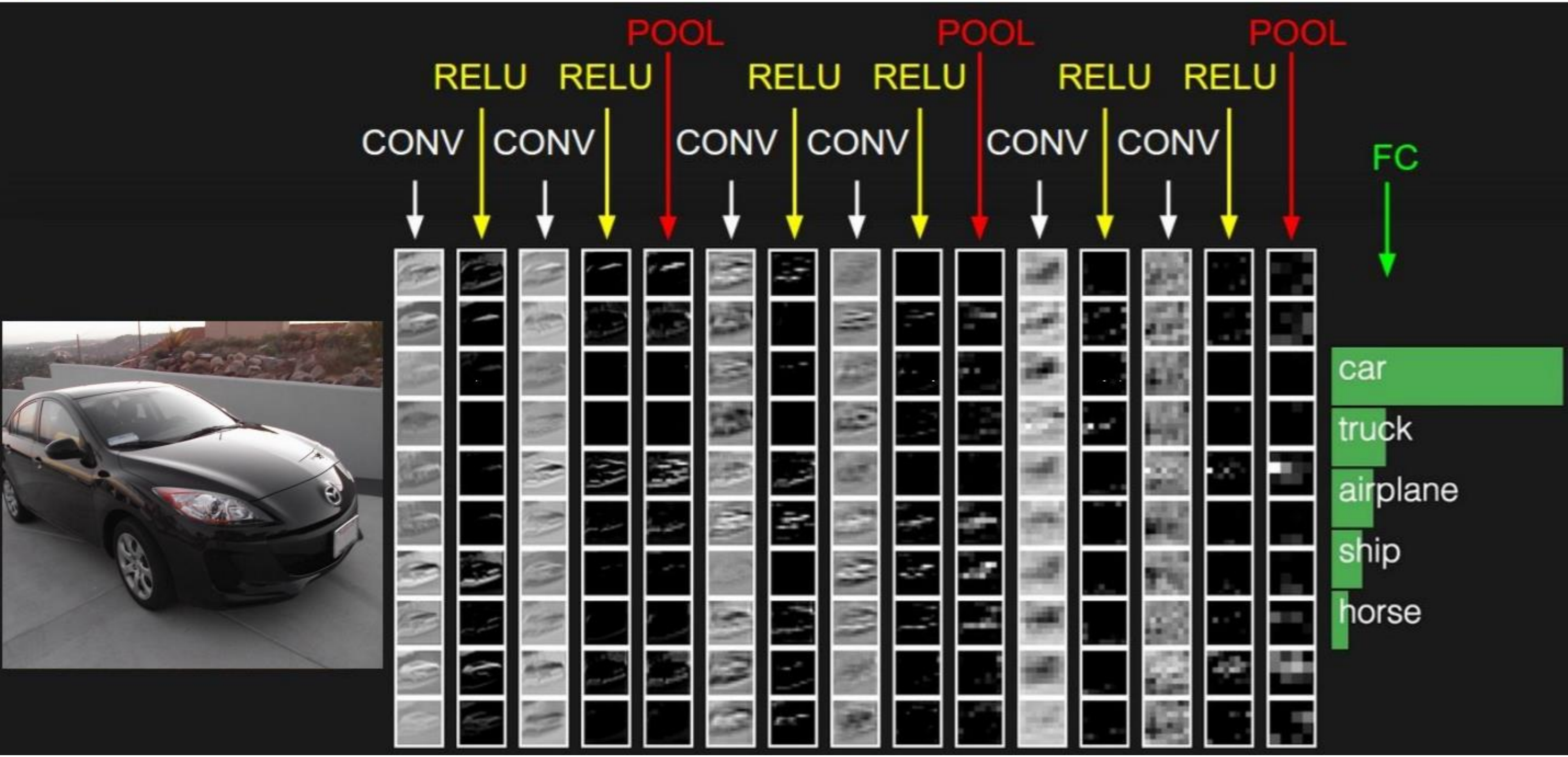


example 5x5 filters
(32 total)

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

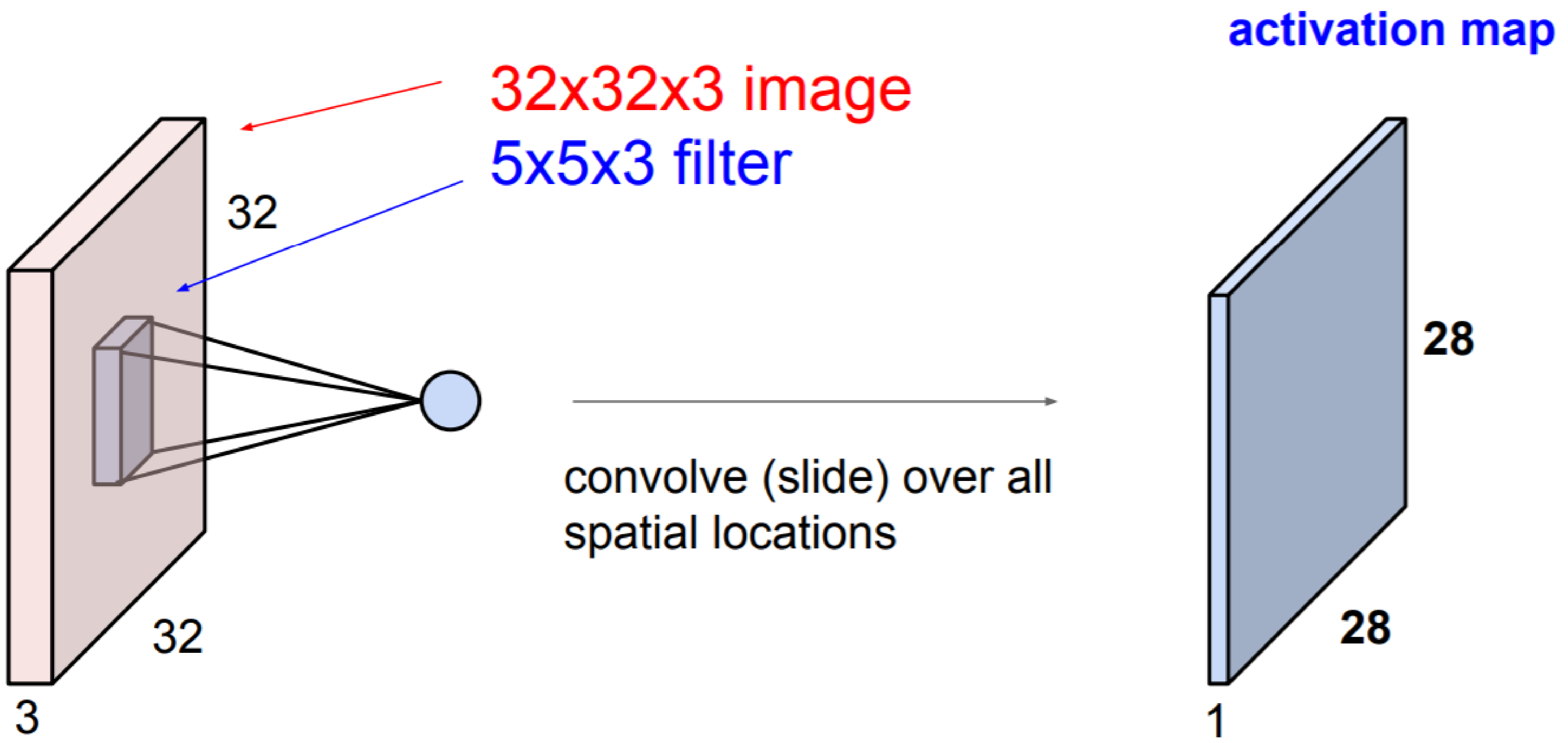
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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


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

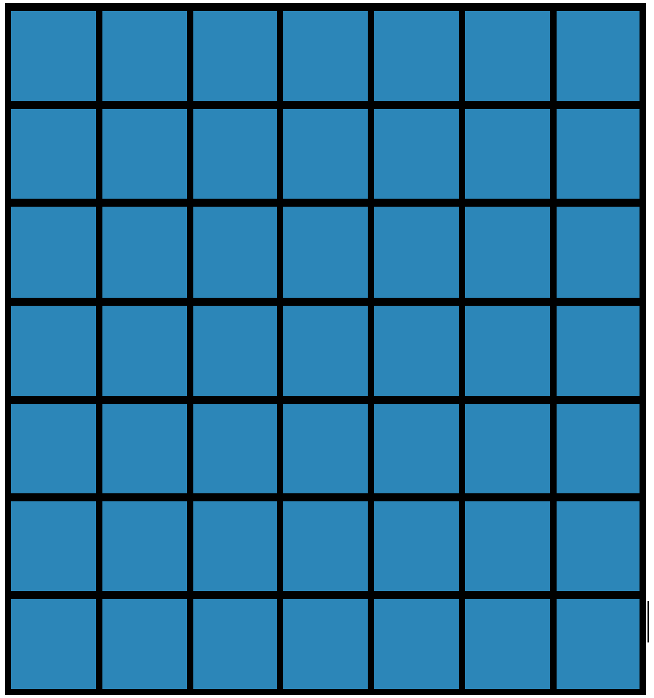
Convolutional layer

- A closer look at spatial dimensions:

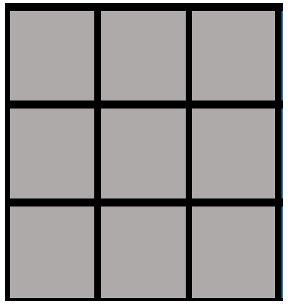


Convolutional filter

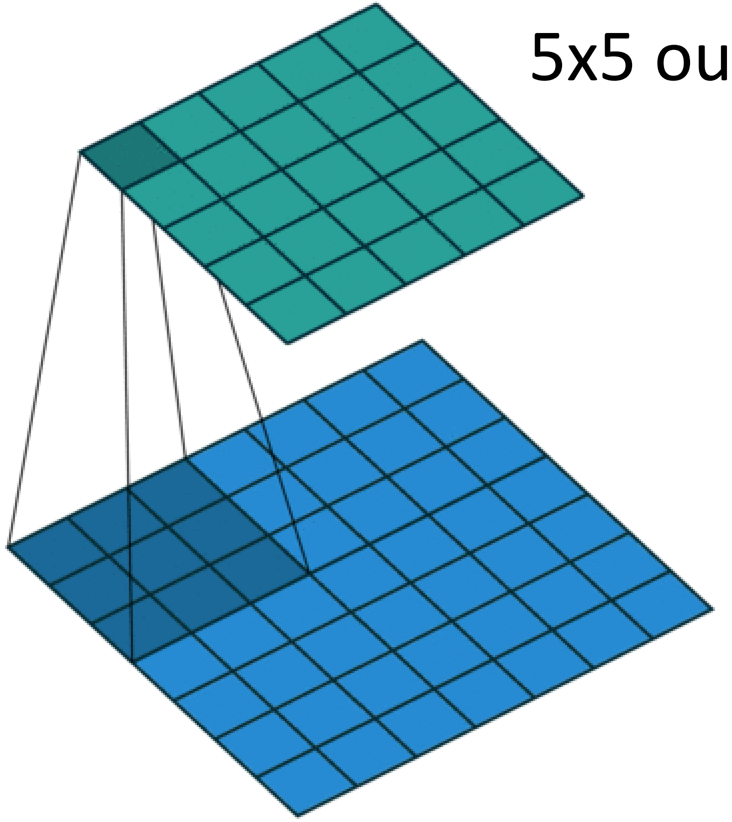
gives the responses of that filter at every spatial position



7x7 input



3x3 filter



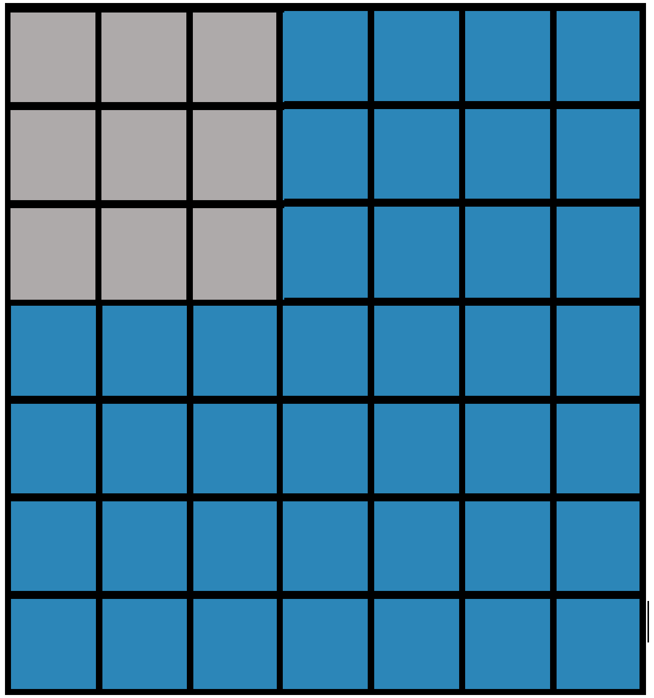
5x5 output

Source: http://iamaaditya.github.io/2016/03/one-by-one-convolution/

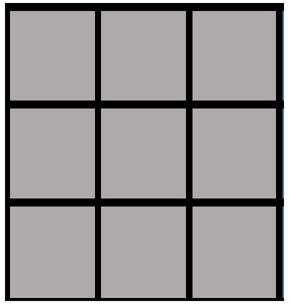
computing a dot product between their weights and a small region they are connected to in the input volume.

Convolutional filter

Stride = 2



7x7 input

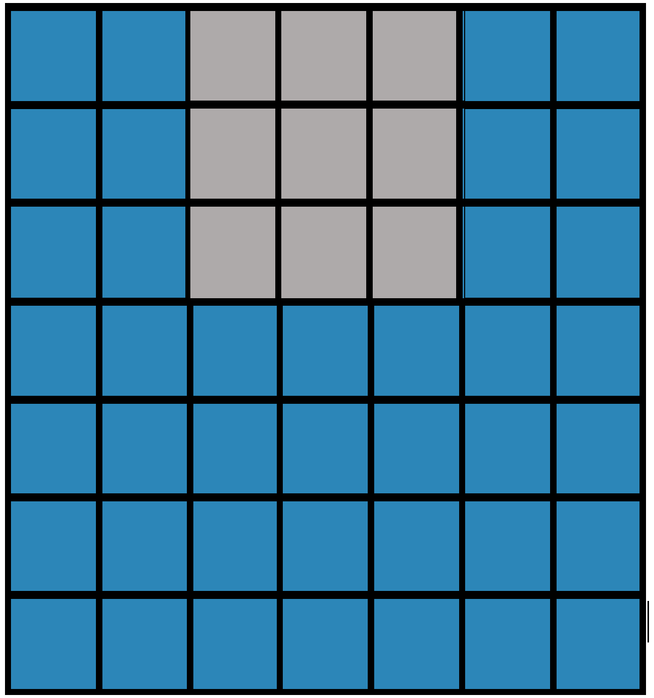


3x3 filter

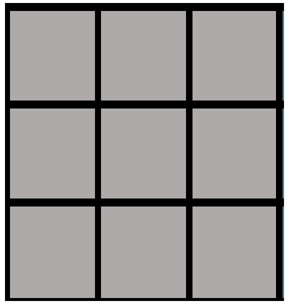
Convolutional filter

Stride = 2

filters jump 2 pixels at a time as we slide them around



7x7 input

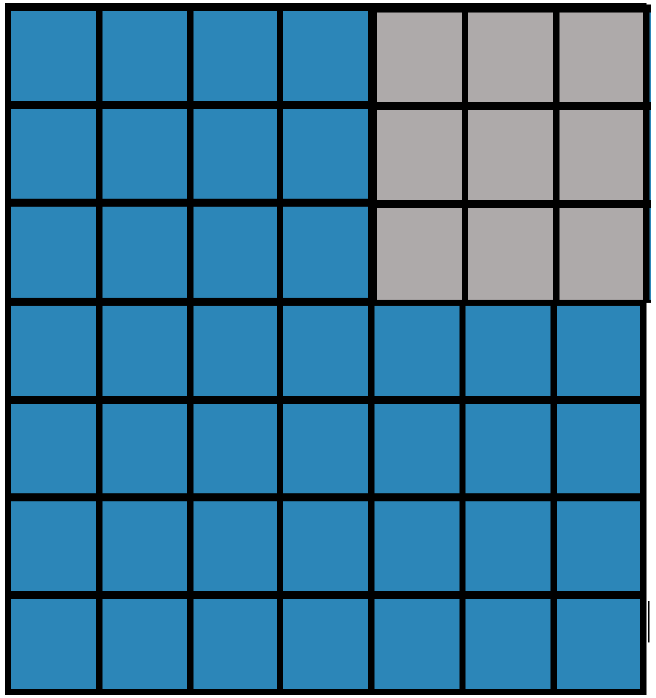


3x3 filter

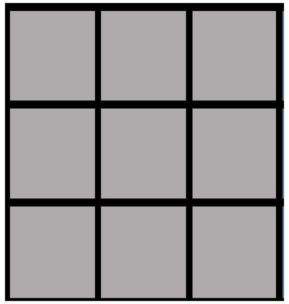
Convolutional filter

Stride = 2

filters jump 2 pixels at a time as we slide them around



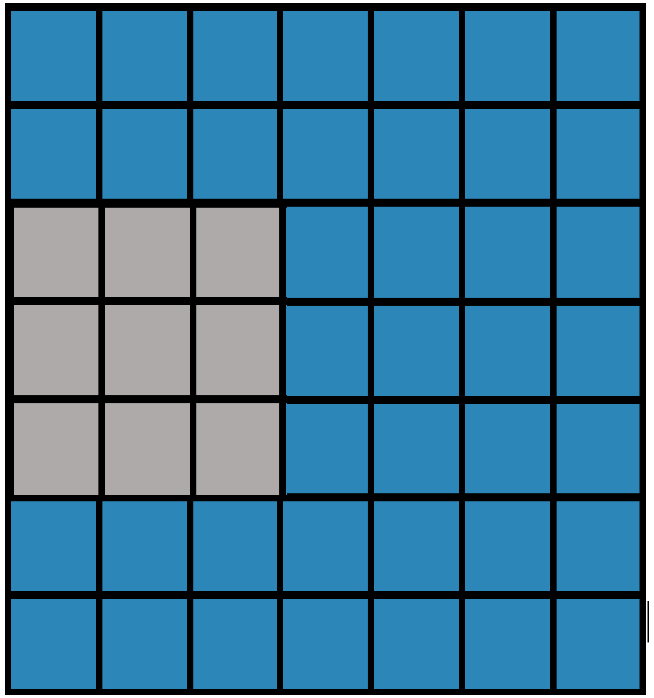
7x7 input



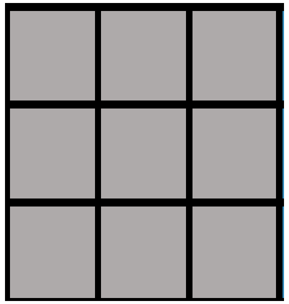
3x3 filter

Convolutional filter

Stride = 2



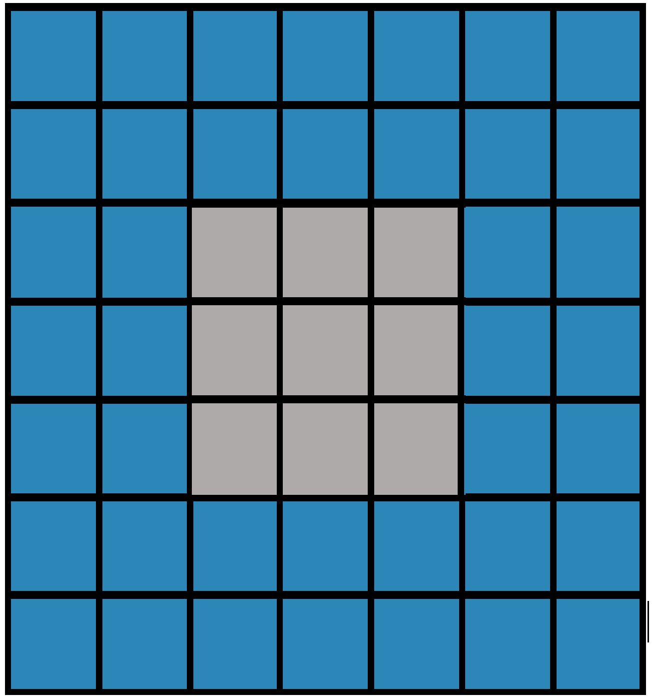
7x7 input



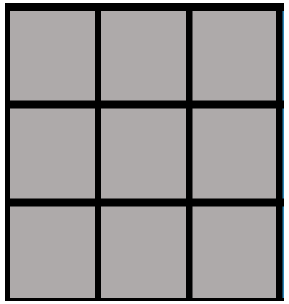
3x3 filter

Convolutional filter

Stride = 2



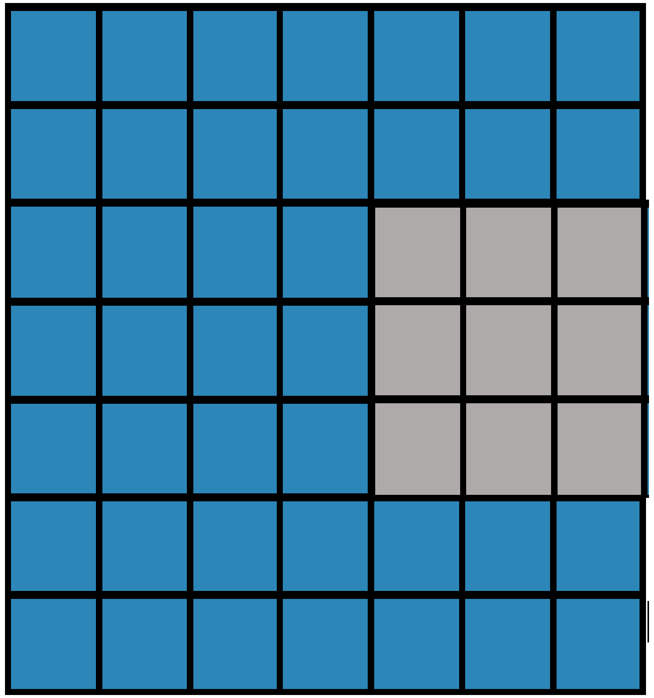
7x7 input



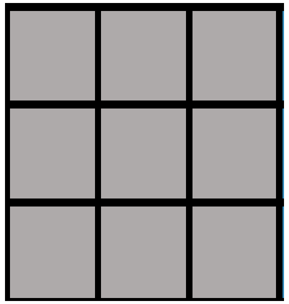
3x3 filter

Convolutional filter

Stride = 2



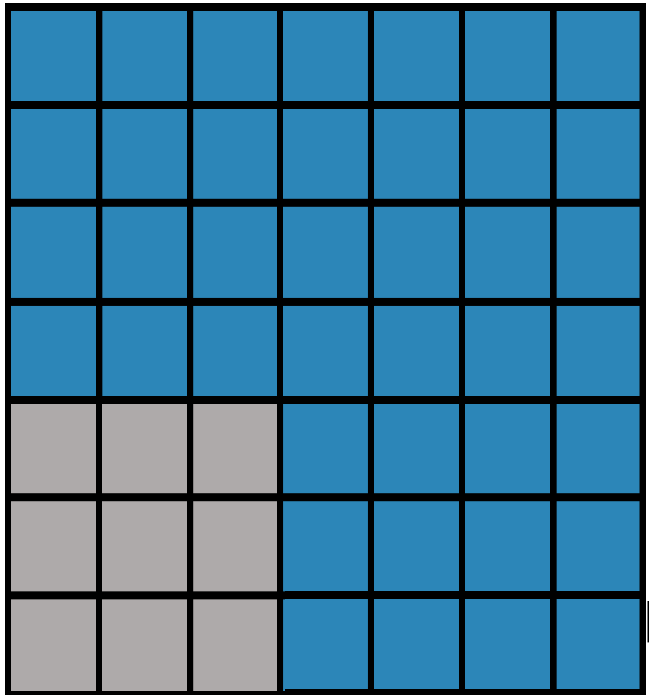
7x7 input



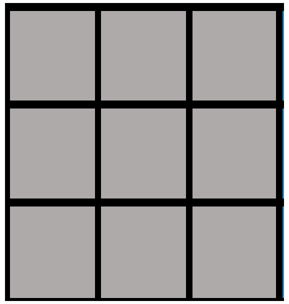
3x3 filter

Convolutional filter

Stride = 2



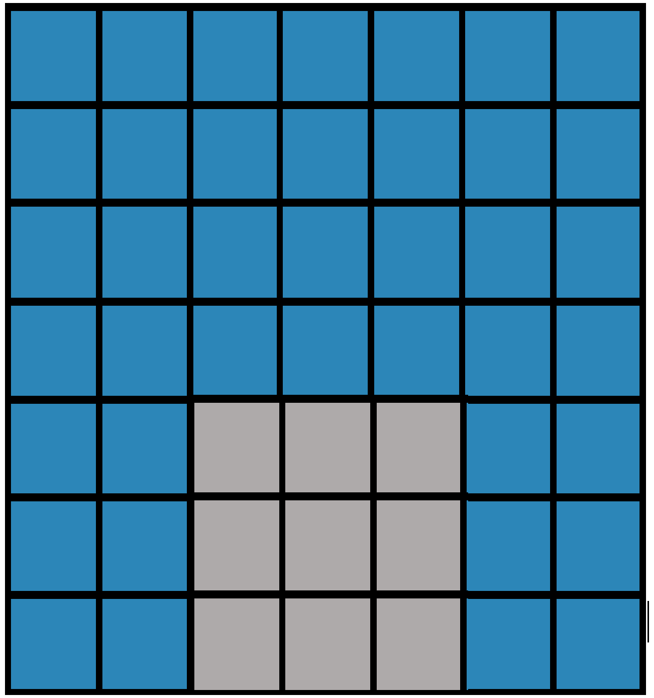
7x7 input



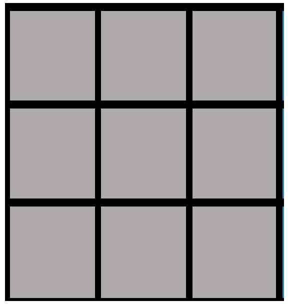
3x3 filter

Convolutional filter

Stride = 2



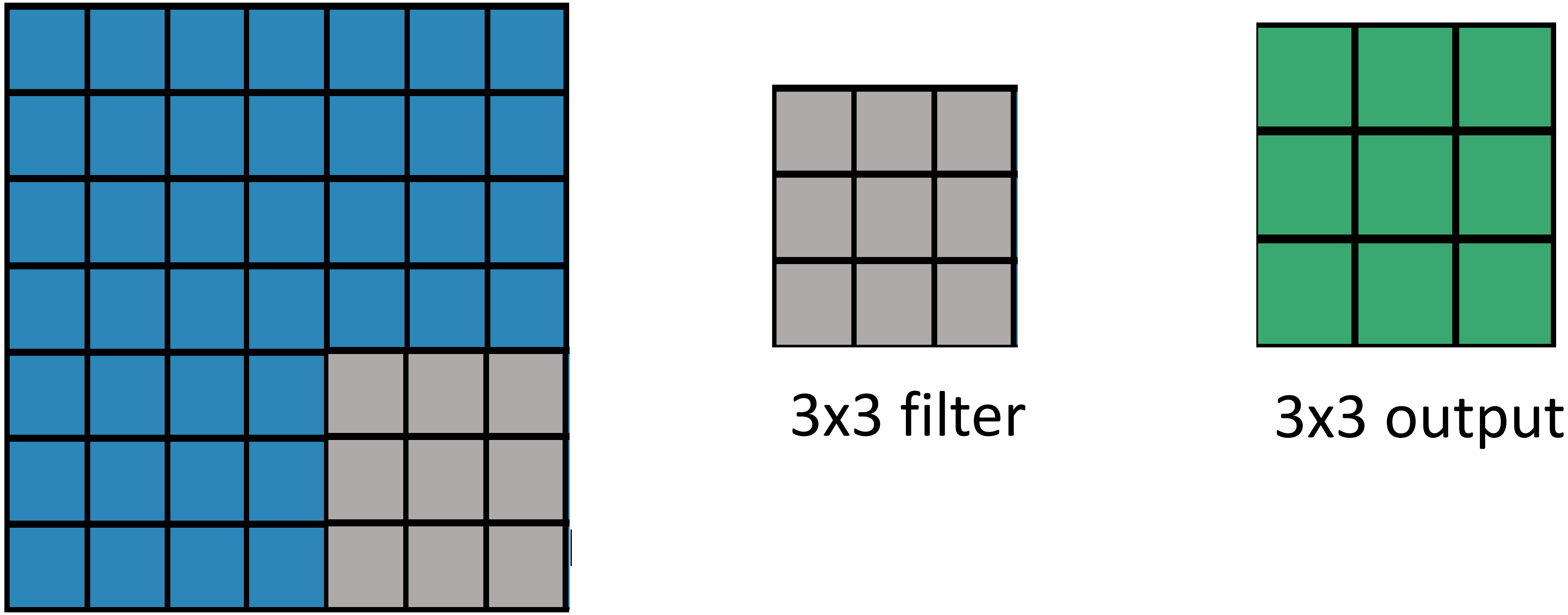
7x7 input



3x3 filter

Convolutional filter

Stride = 2



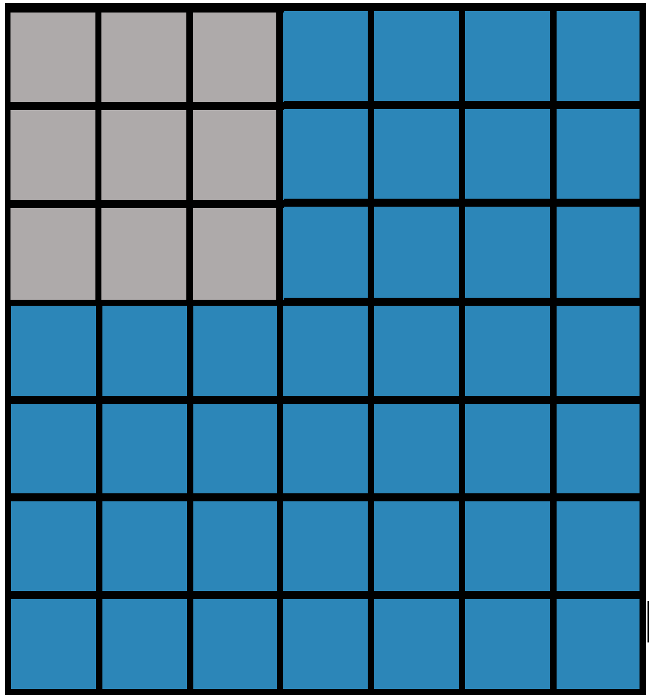
7x7 input

3x3 filter

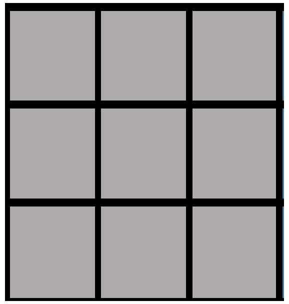
3x3 output

Convolutional filter

Stride = 3



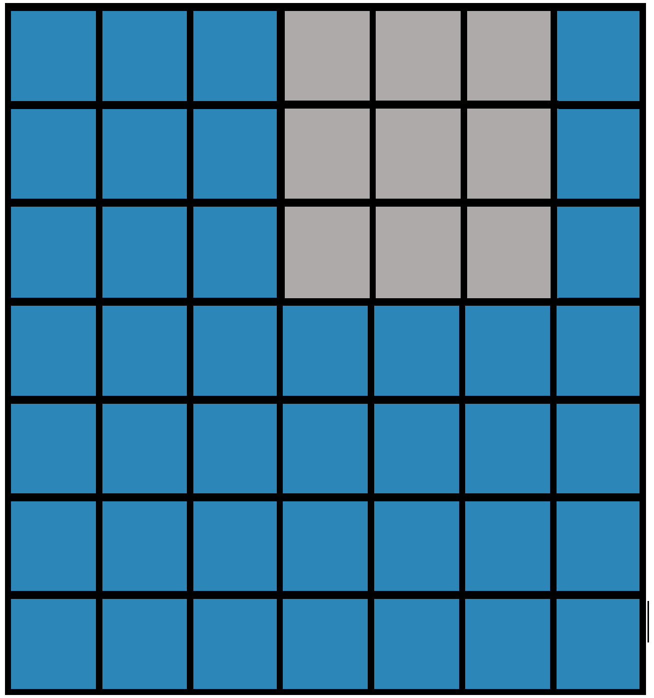
7x7 input



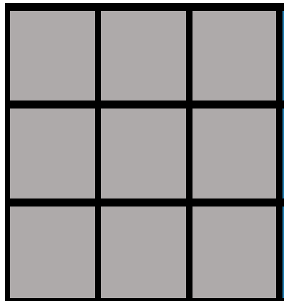
3x3 filter

Convolutional filter

Stride = 3



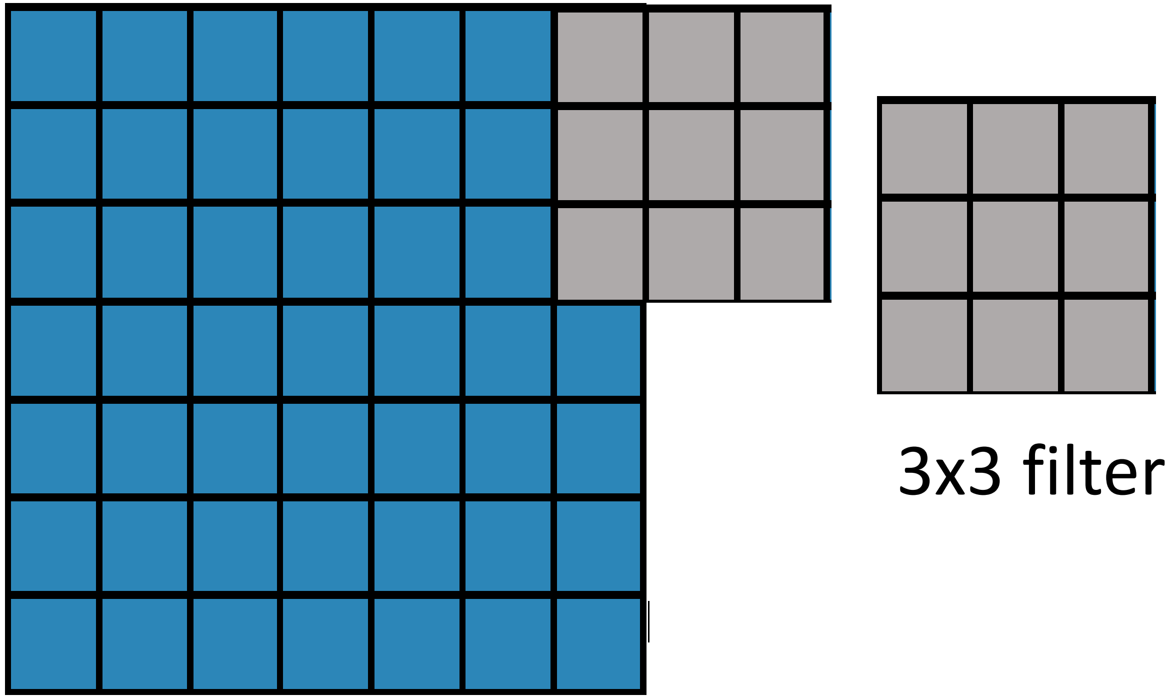
7x7 input



3x3 filter

Convolutional filter

Stride = 3



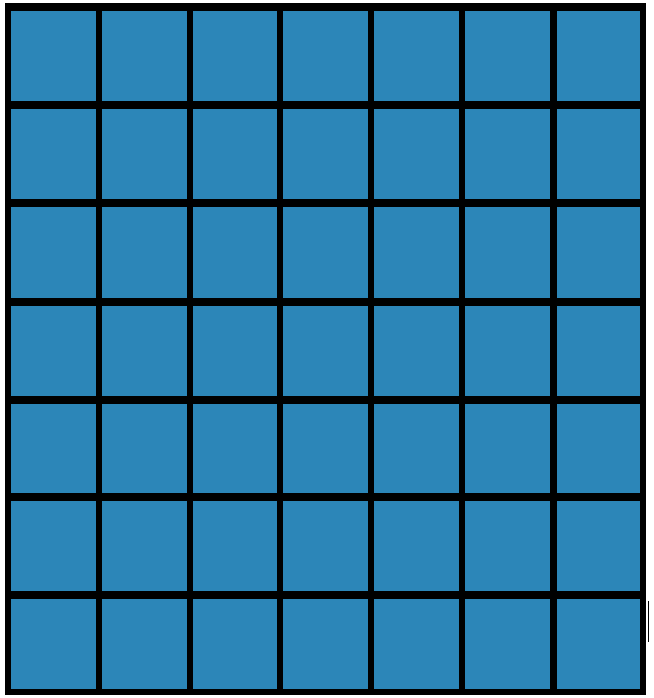
7x7 input

3x3 filter

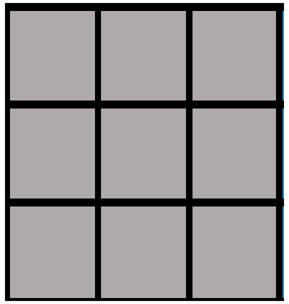
cannot apply 3x3 filter on 7x7 input with stride 3.

Output size

N



F



Output size:

(N - F) / stride + 1

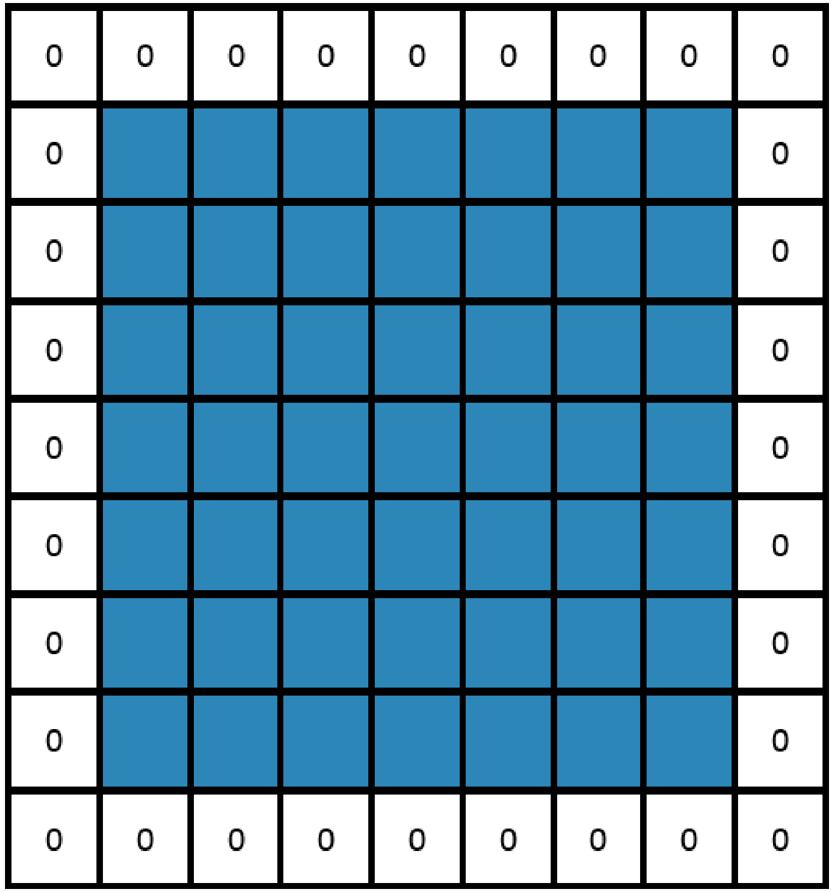
Example:

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

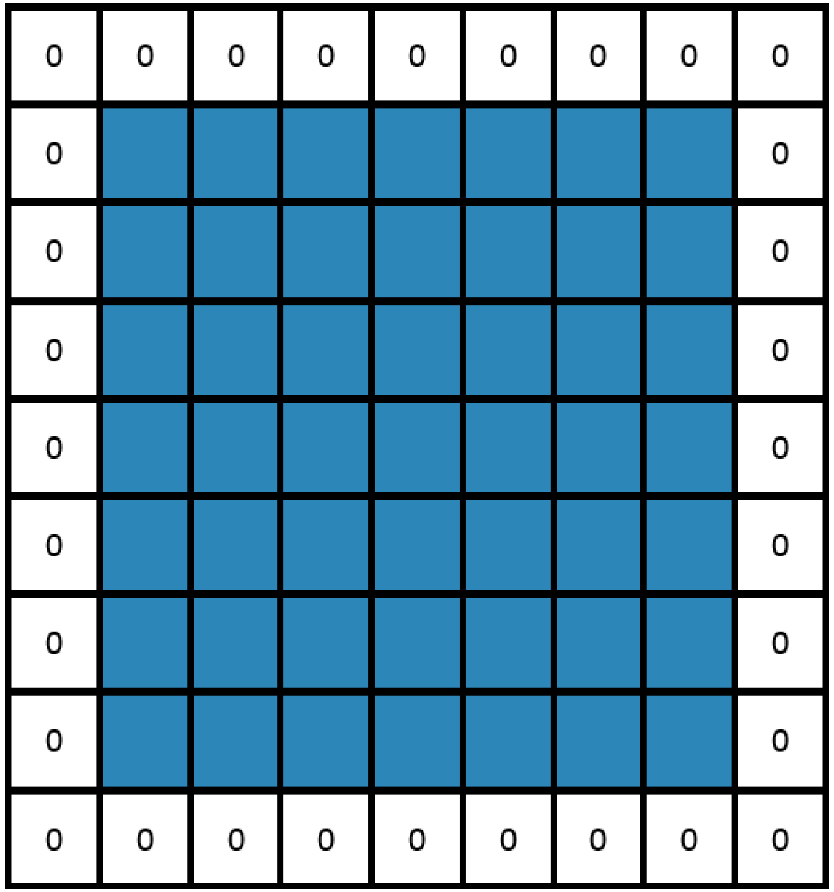


input 7x7
Filter 3x3
stride 1
zero pad with 1 pixel border

=> output= 7x7

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

In practice: Common to zero pad the border



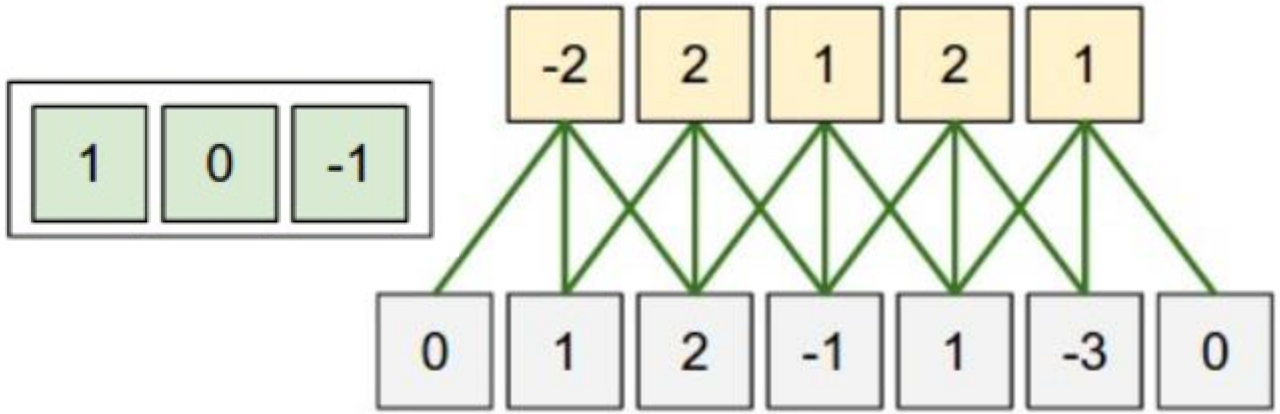
Common in practice:
filters FxF
stride 1
zero-padding with (F-1)/2

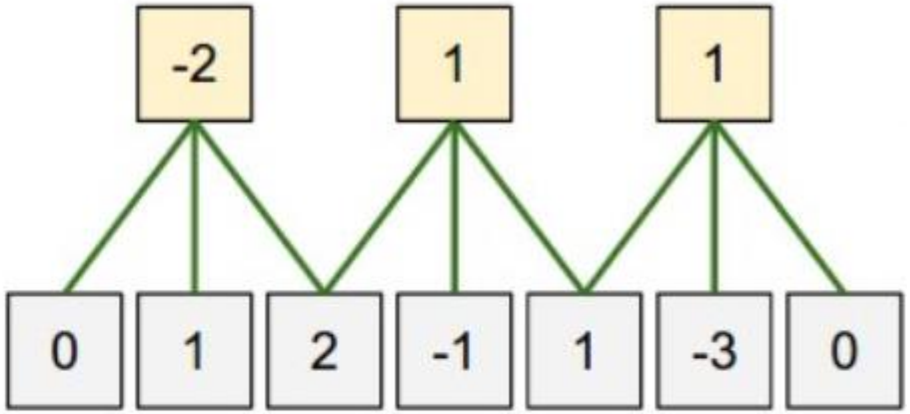
=> will preserve size

e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

zero padding allows us to control the spatial size of the output volumes

1D example



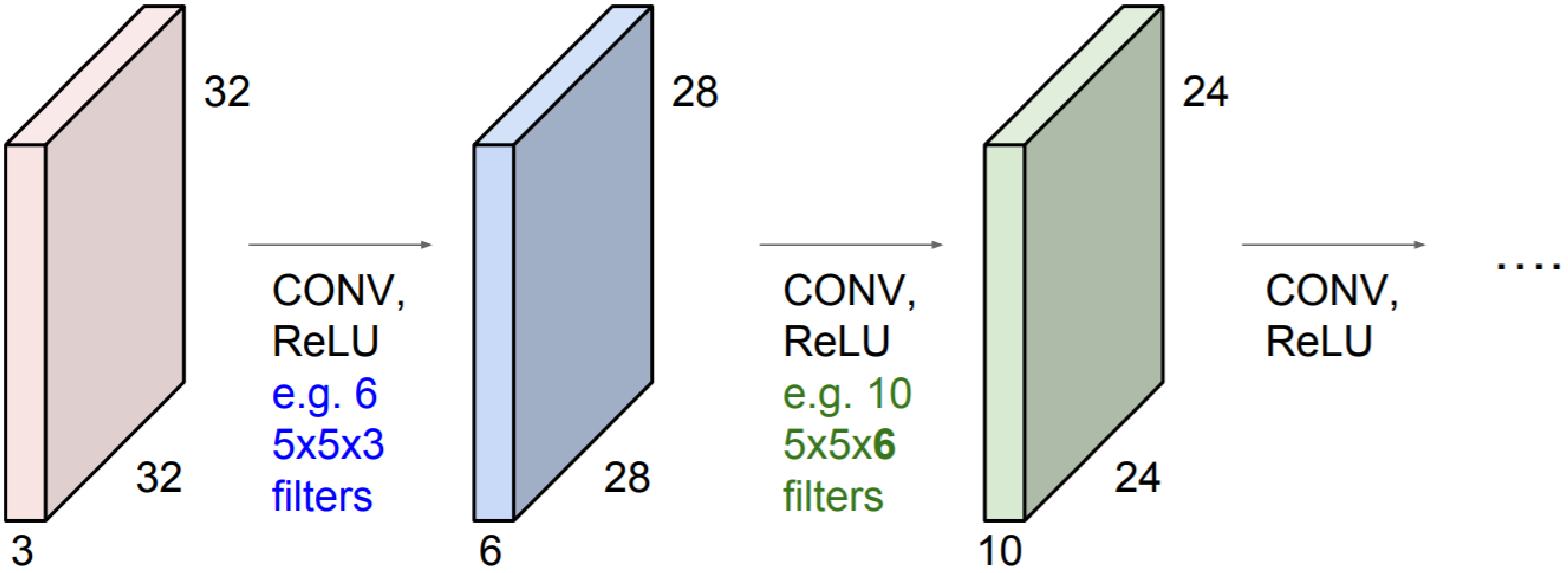


N = 5
F = 3
P = 1
S = 1
Output = (5 - 3 + 2)/1+1 = 5.

N = 5
F = 3
P = 1
S = 2
Output = (5 - 3 + 2)/2+1 = 3.

We want to maintain the input size

- (32 -> 28 -> 24 ...).
- Shrinking too fast is not good, doesn’t work well.



Example

- Input: 32x32x3
- Filters: 10   5x5x3 filters
- Stride: 1
- Pad: 2

- Output size: 32x32x10

Example

- Input: 32x32x3
- Filters: 10   5x5x3 filters
- 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)
  - $D_2 = K$

Common settings:

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

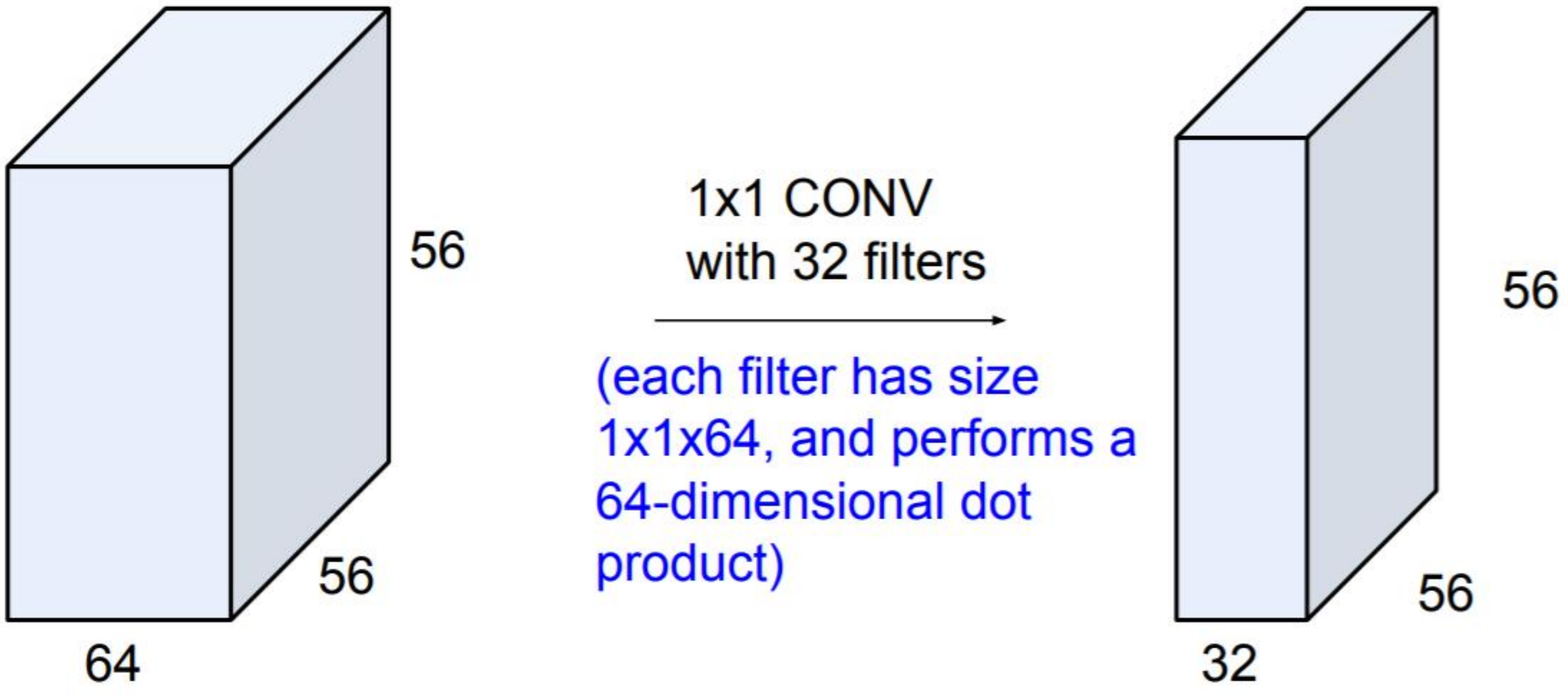
F = 3, S = 1, P = 1

F = 5, S = 1, P = 2

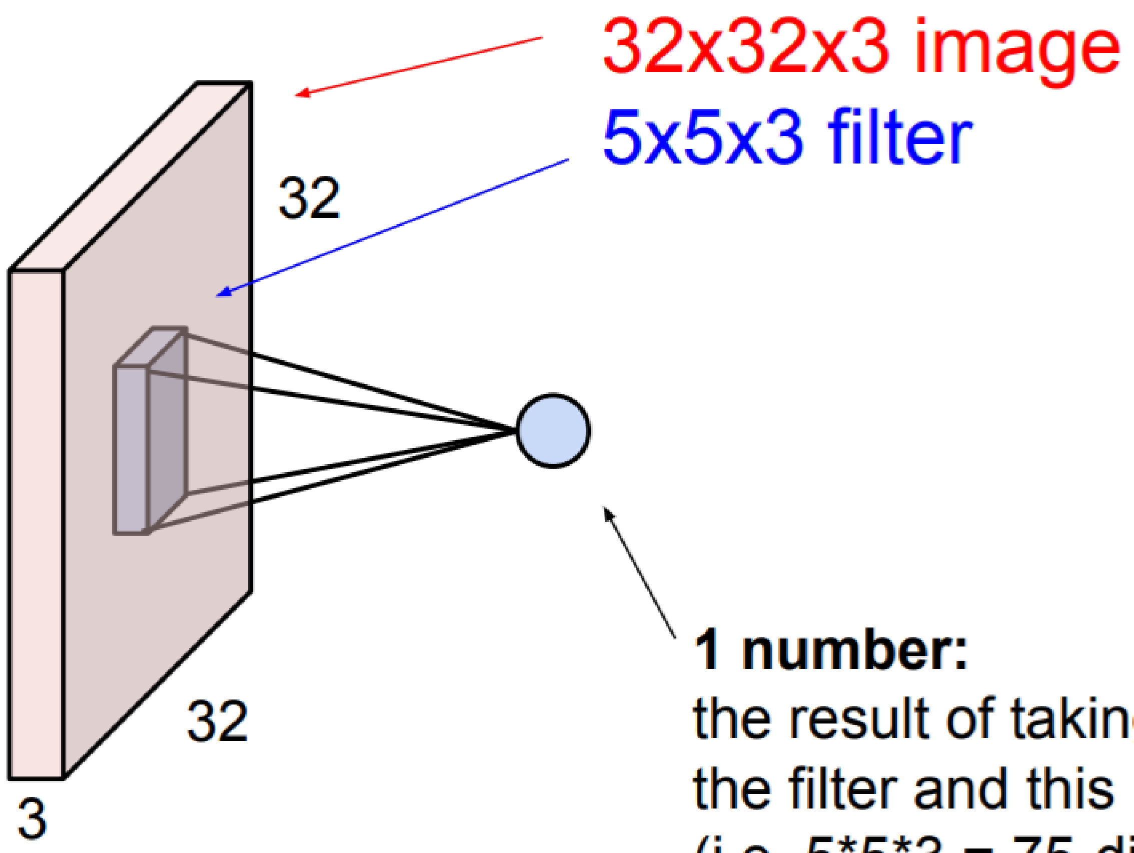
F = 5, S = 2, P = ? (whatever fits)

F = 1, S = 1, P = 0

Example

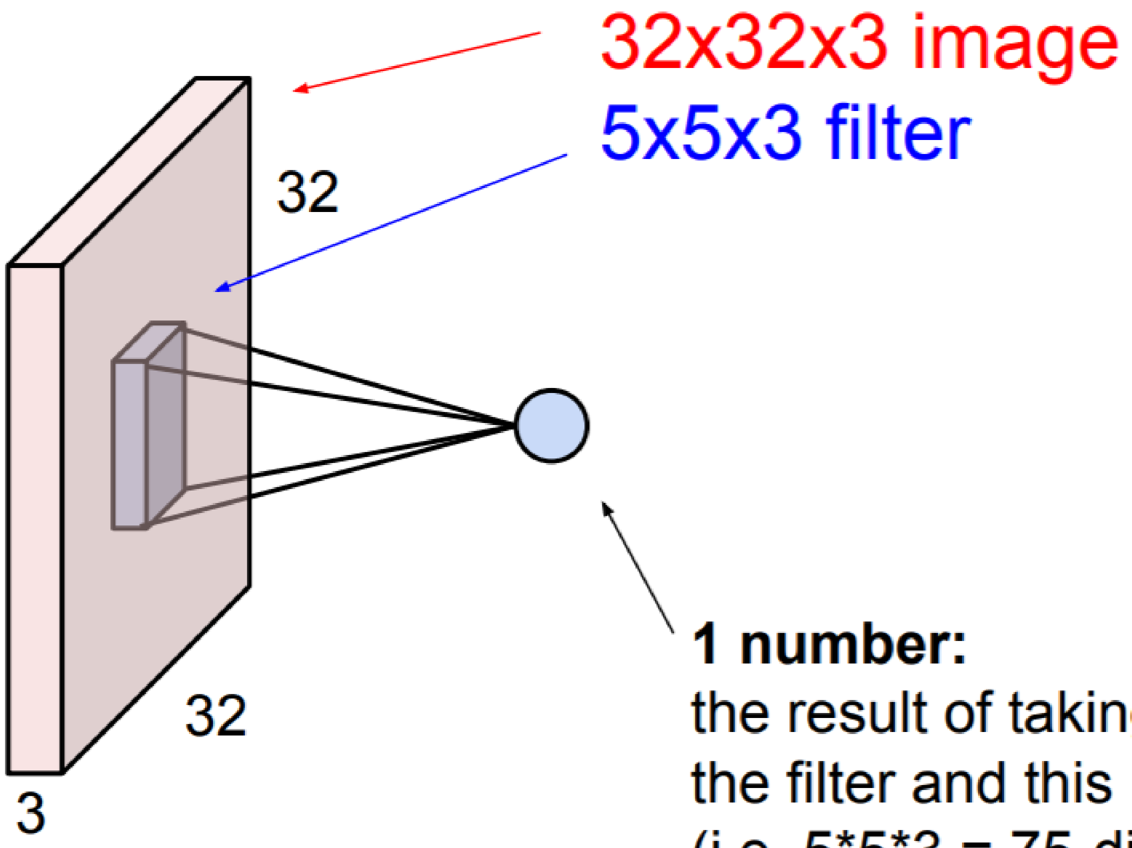

56
56
64
1x1 CONV
with 32 filters
(each filter has size
1x1x64, and performs a
64-dimensional dot
product)
56
56
32

Convolutional layer: neural view



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)

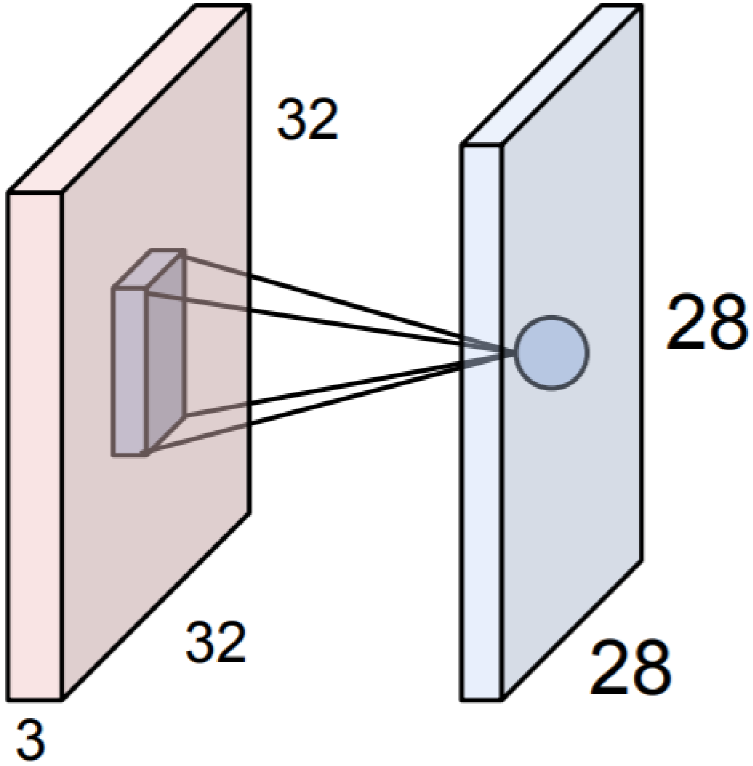
Convolutional layer: neural view



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)

Convolutional layer: neural view



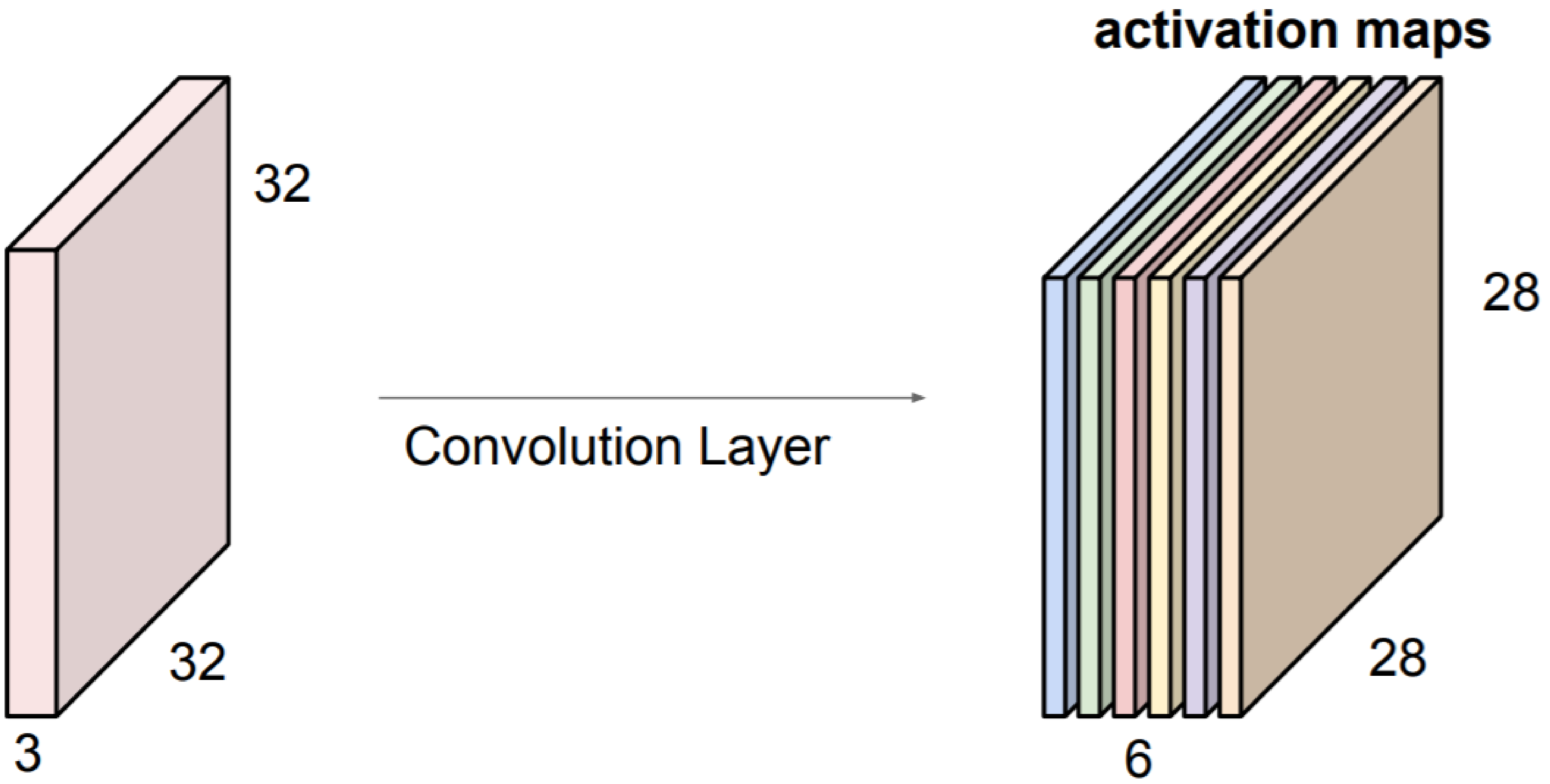
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters “5x5x3”

“5x5 filter” => “5x5 receptive field for each neuron”

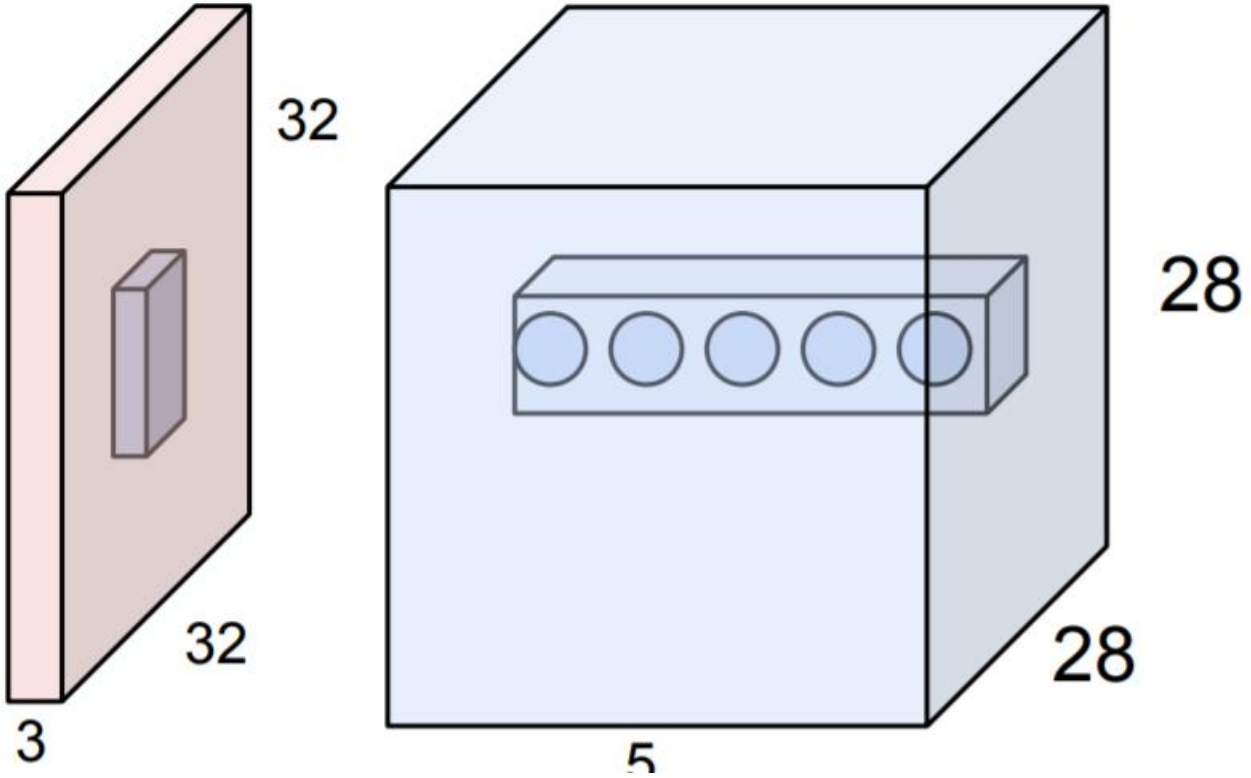
Convolutional layer: neural view

- If we had 6 “5x5 filters”, we’ll get 6 separate activation maps:



There will be 6 different neurons all looking at the same region in the input volume
constrain the neurons in each depth slice to use the same weights and bias

Convolutional layer: neural view



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

set of neurons that are all looking at the
same region of the input as a **depth column**

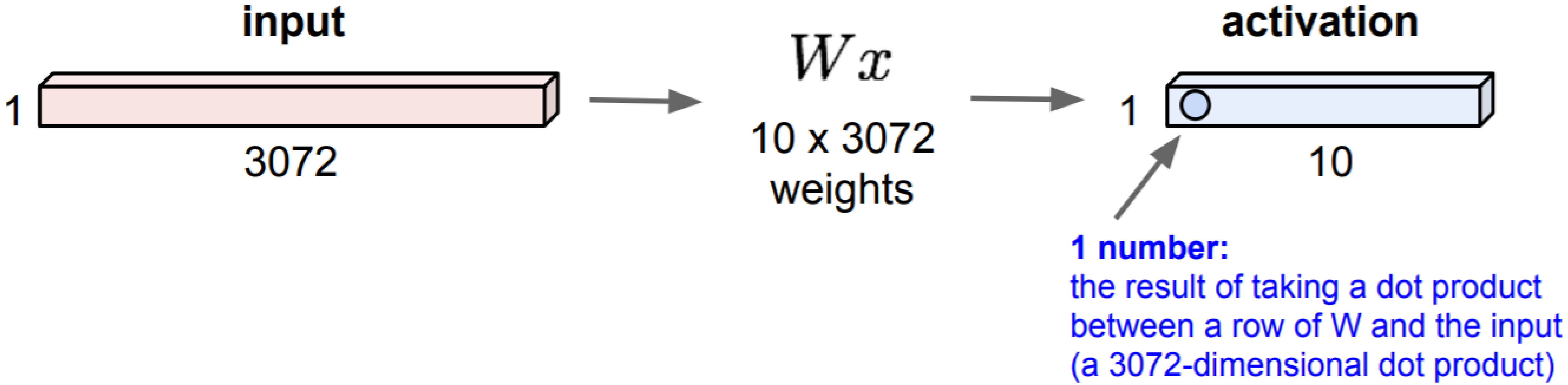
Convolutional layer

- **Local Connectivity**
  - each neuron is connected to only a local region of the previous layer outputs.
    - receptive field (or the filter size)
  - The connections are local in space (along width and height)
- **Parameter Sharing**
  - if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2)

Fully connected layer

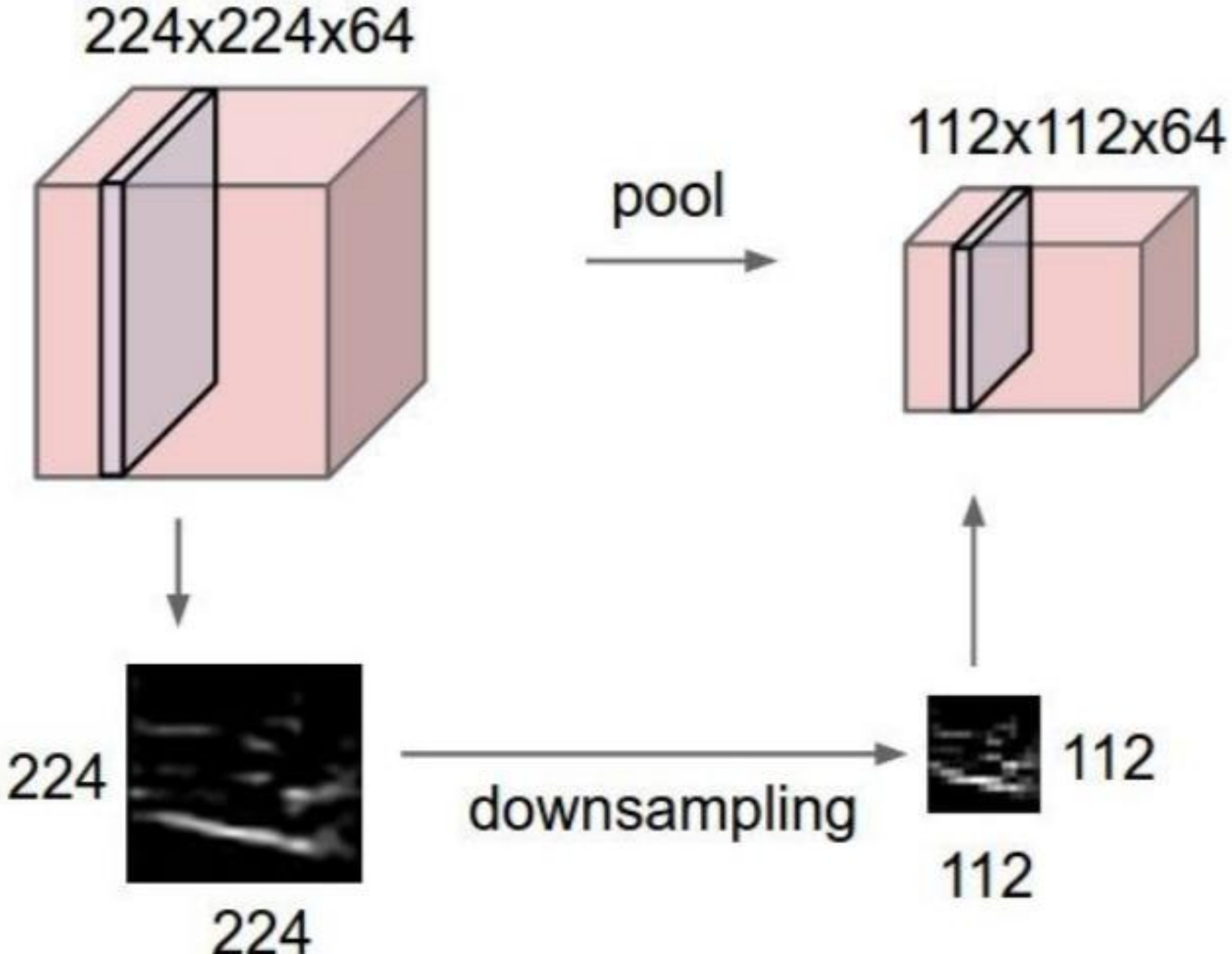
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

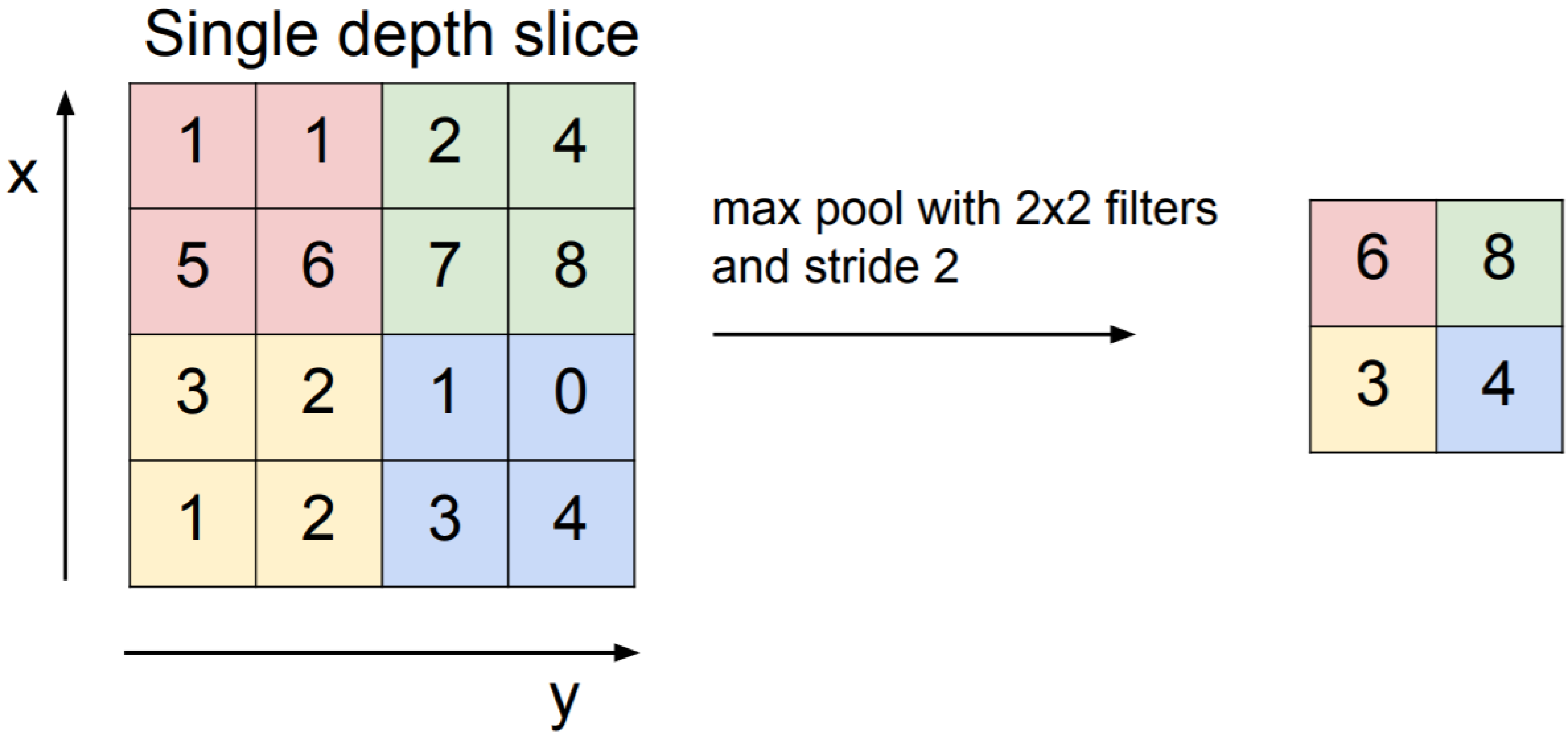


Pooling layer

- 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

Pooling

- reduce the spatial size of the representation
  - to reduce the amount of parameters and computation in the network
  - to control overfitting
- operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

Pooling

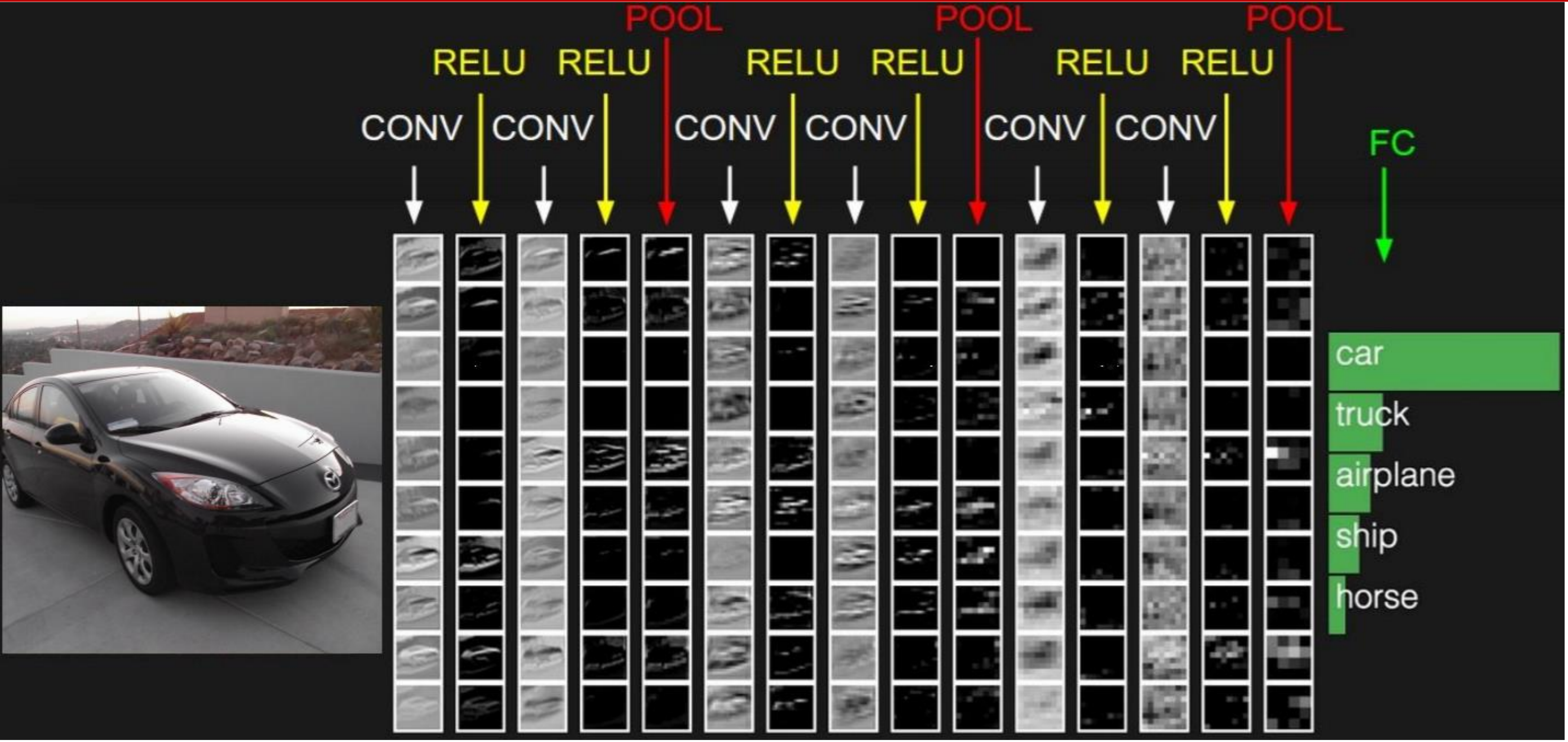
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since 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)



n

- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)

Demo

- 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

Resources

- Deep Learning Book, Chapter 9.
- Please see the following note:
  - http://cs231n.github.io/convolutional-networks/