COMP 5630/6630:Machine Learning

Lecture 10: CNN, Demo CNN

Slide Adapted: Drs. Dhruv Batra, Marc'Aurelio Ranzato, Aakur

Convolutional Neural Networks

Convolutional Nets: Intuition

- Basic Idea
  - On board
  - Assumptions:
    - Local Receptive Fields
    - Weight Sharing / Translational Invariance / Stationarity
  - Each layer is just a convolution!

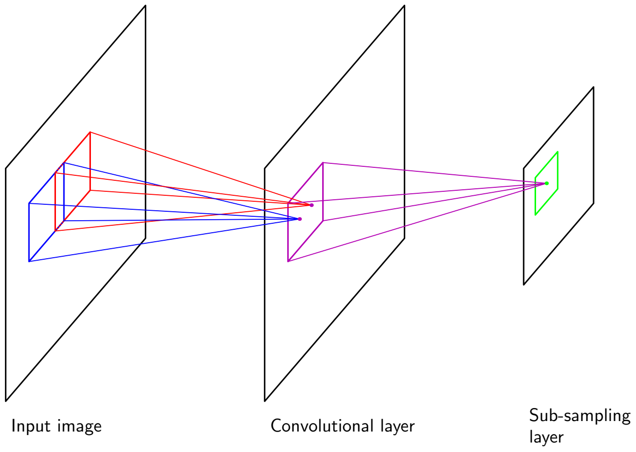
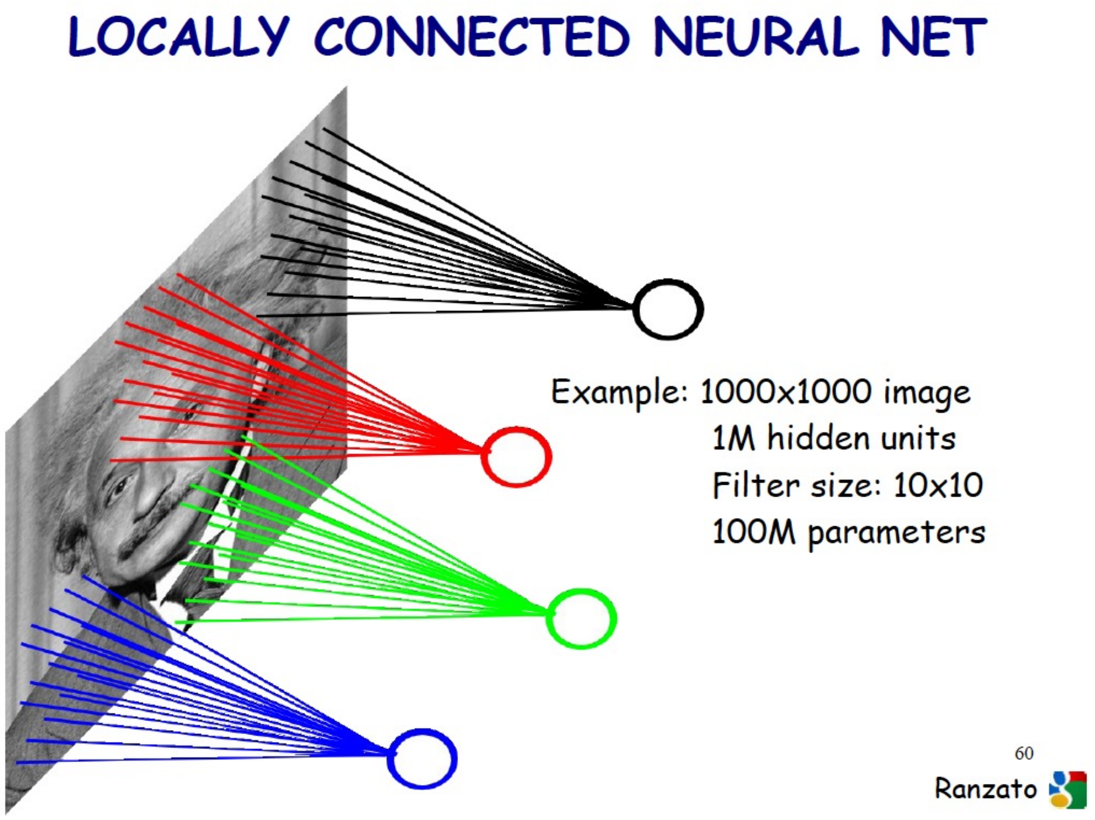


Image Credit: Chris Bishop

Convolutional Nets: Assumption1

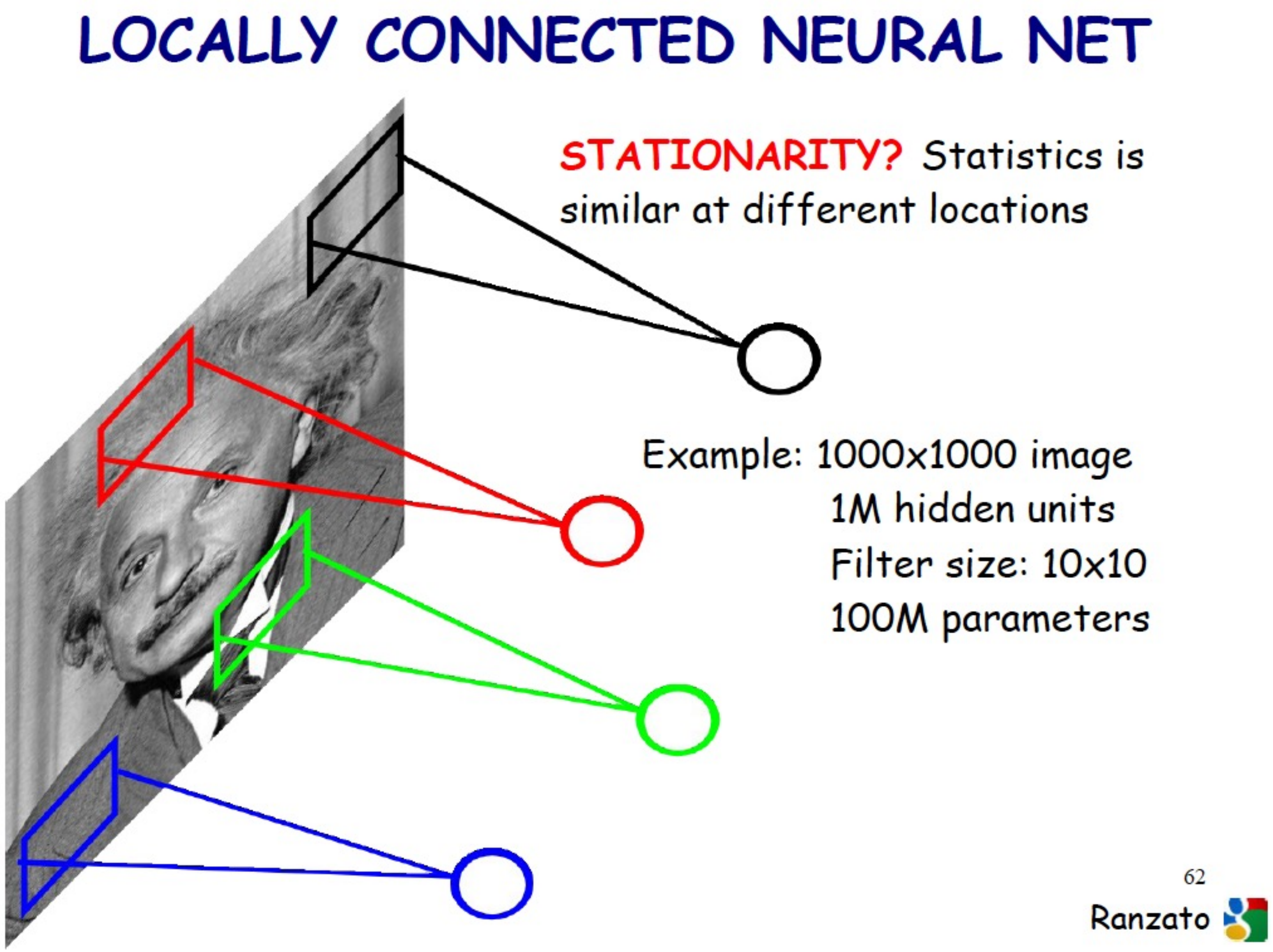


Flatten Image (X) 1000x 1000 =10^6
Hidden unit/neuron (W) = 10^6

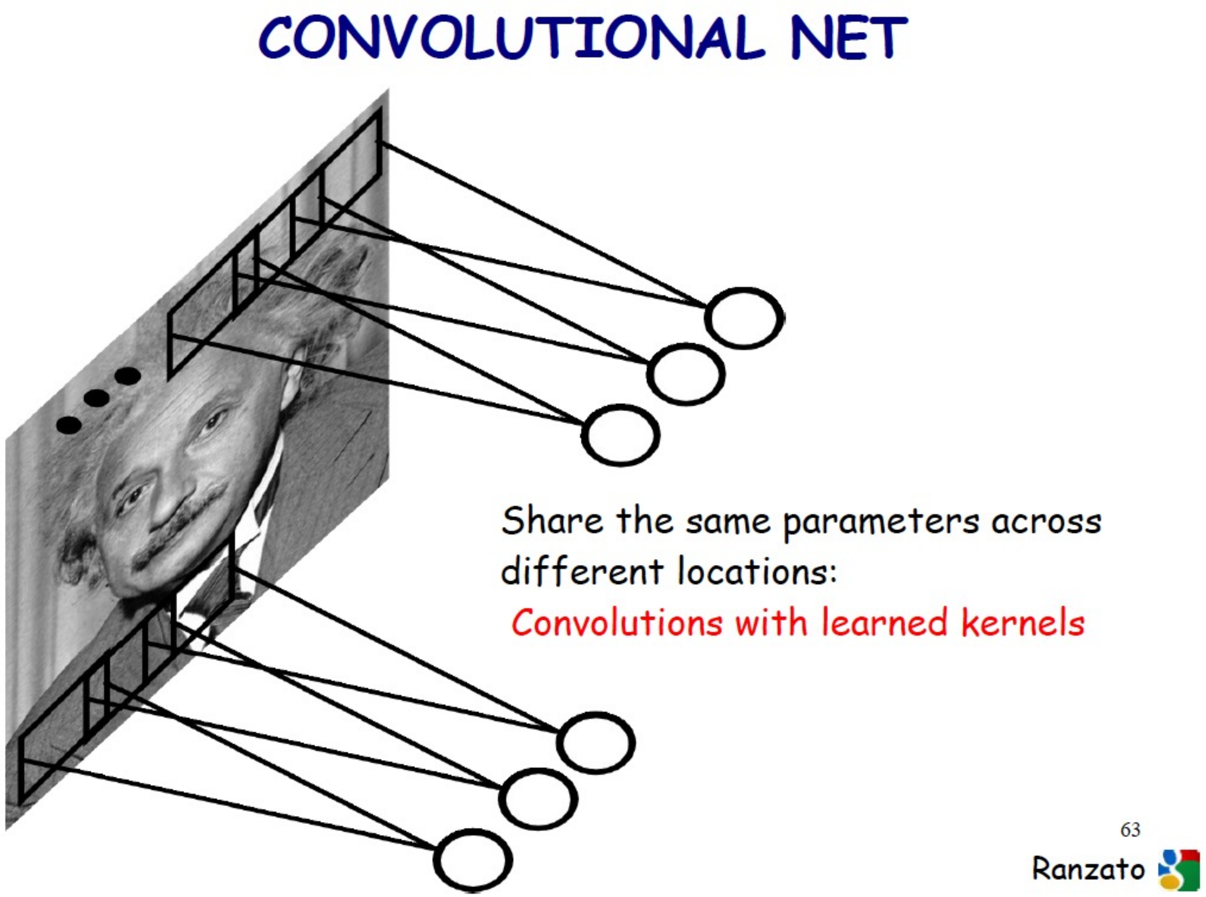
Each neuron is connected to only 10x10 units (Filter size)

Total number of parameters = 10^6 x 100 = 100M

Convolutional Nets: Assumption 1



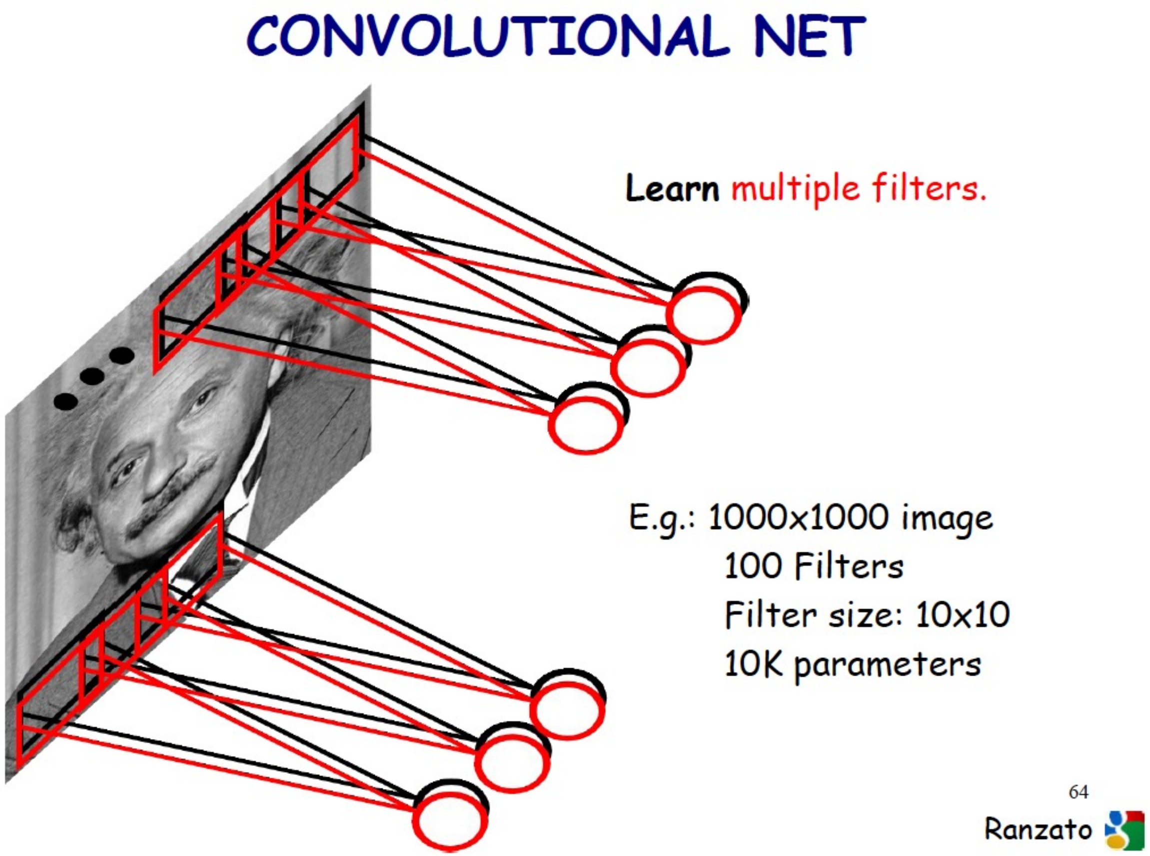
Convolutional Nets: Assumption 2



Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels/filters

Convolutional Nets: Assumption 2

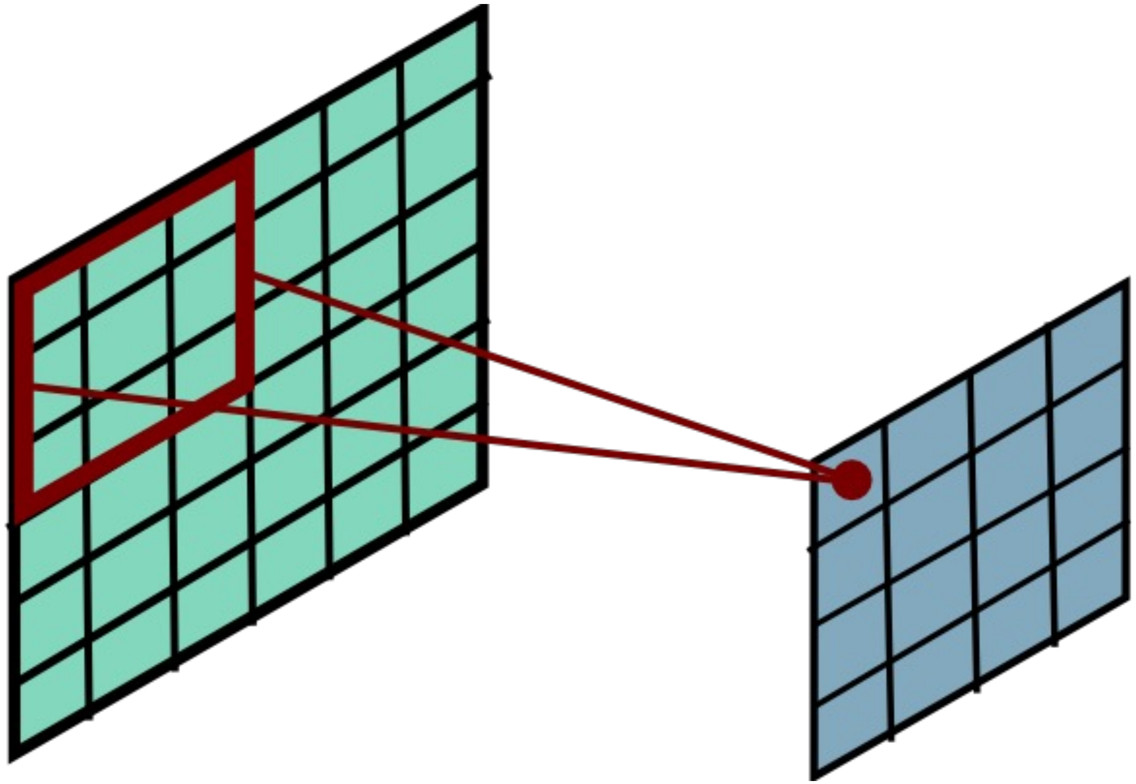


- Now we have two filters:
  1. One **black** color
  2. One **red** color

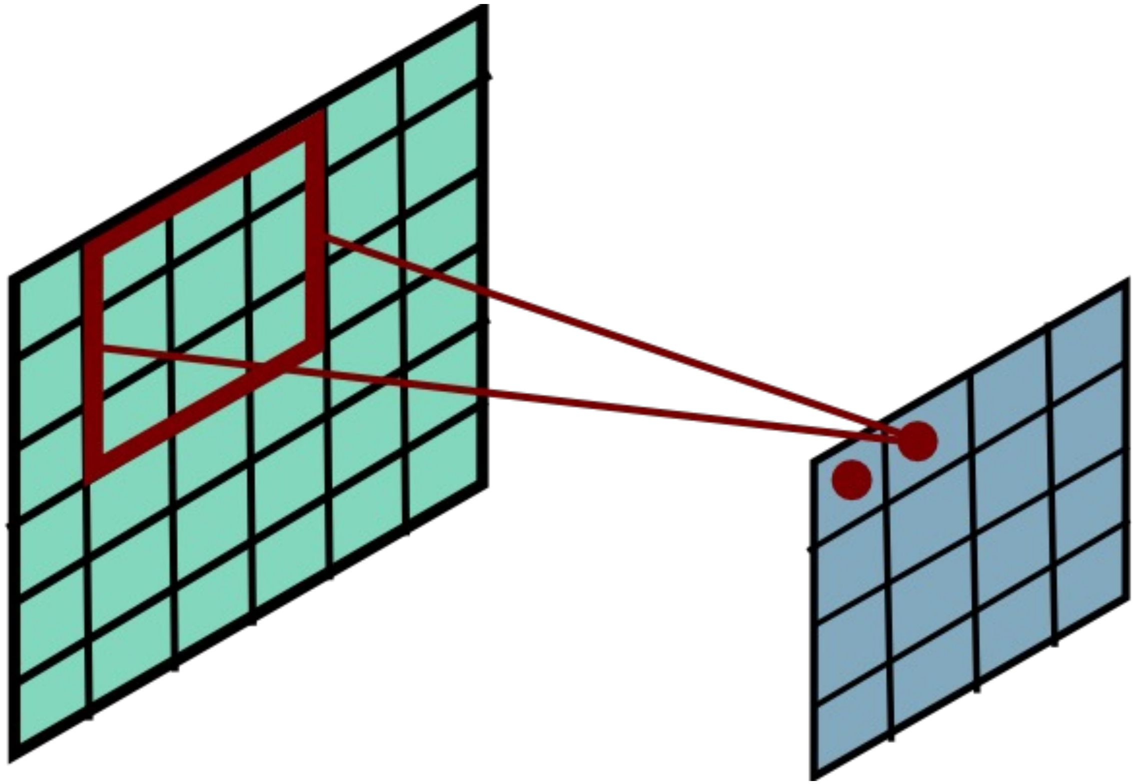
- Each filter will slide over the entire image
- Each filter will learn different local properties

- If we have 100 such filter, each with 10x10 size,
  - Total parameters to learn = 100 x 10 x 10 = 10k

Convolutional Layer

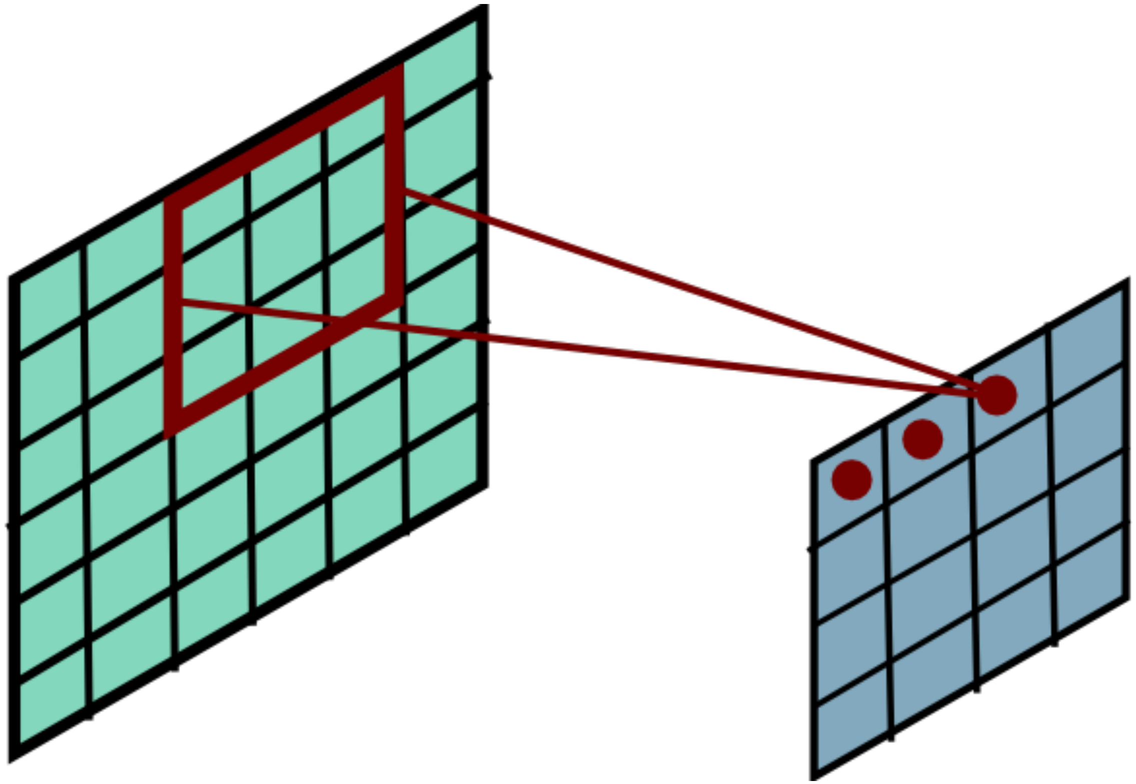


Convolutional Layer

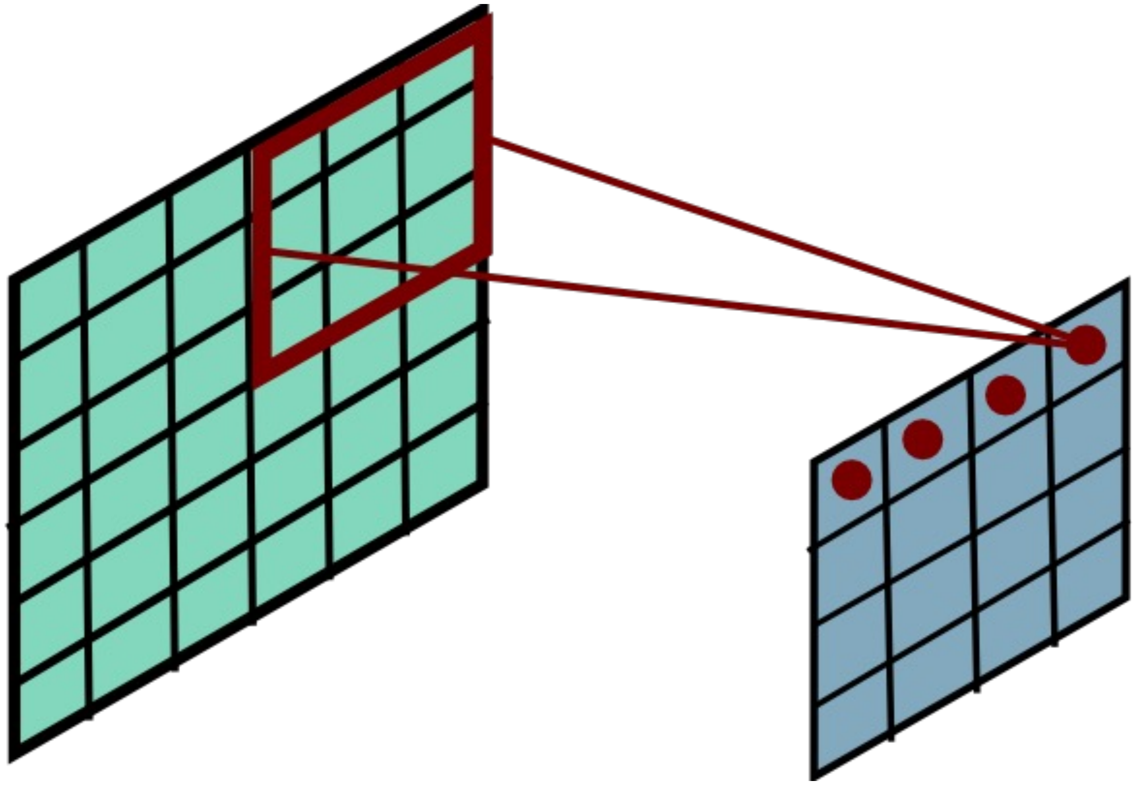


Slide Credit: Dhruv Batra, Marc'Aurelio Ranzato

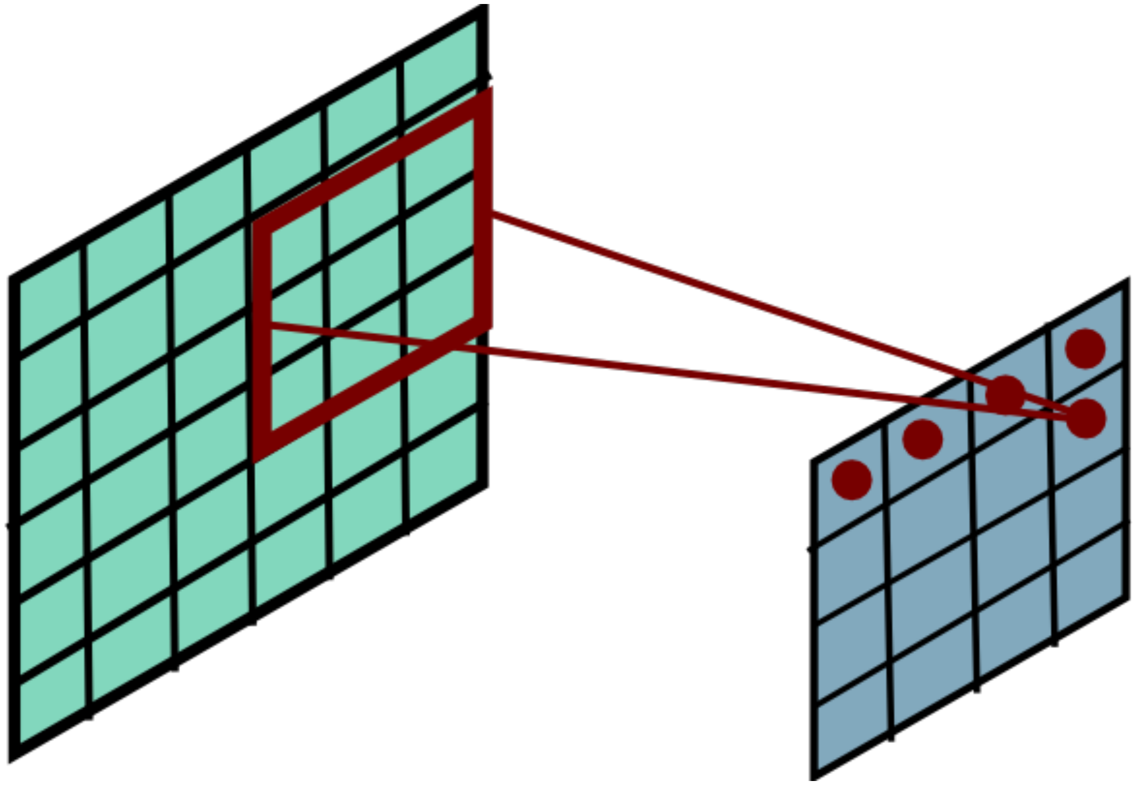
Convolutional Layer



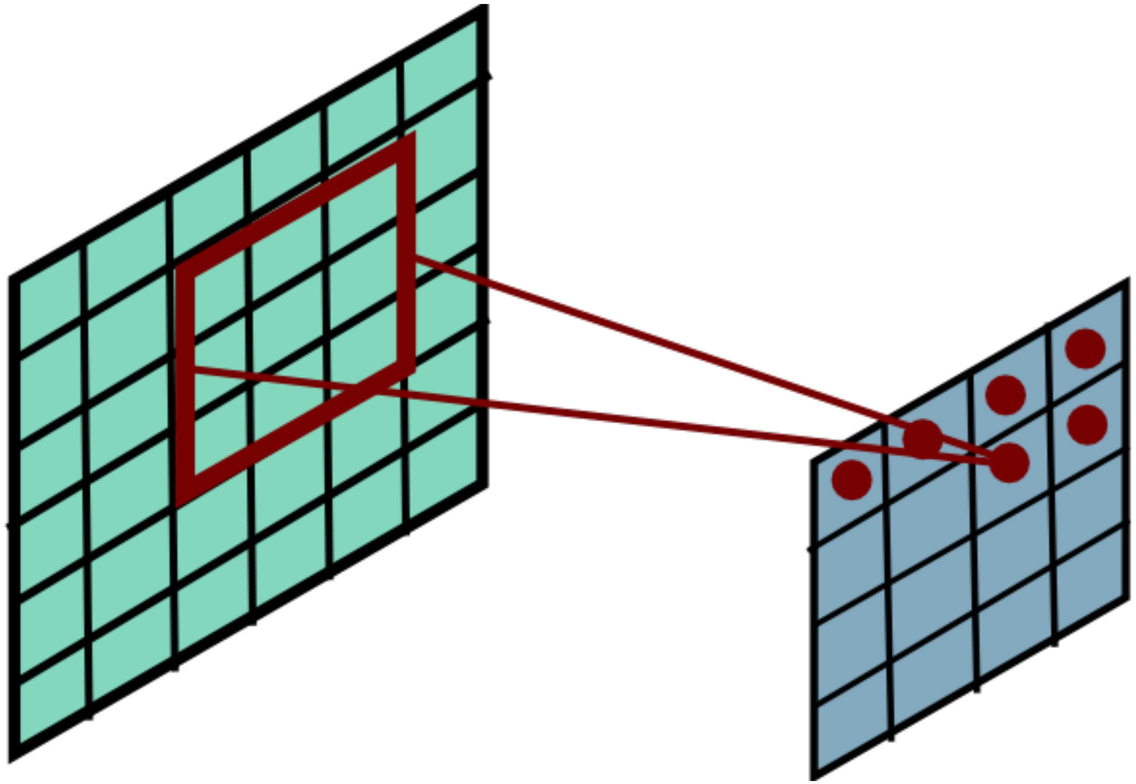
Convolutional Layer



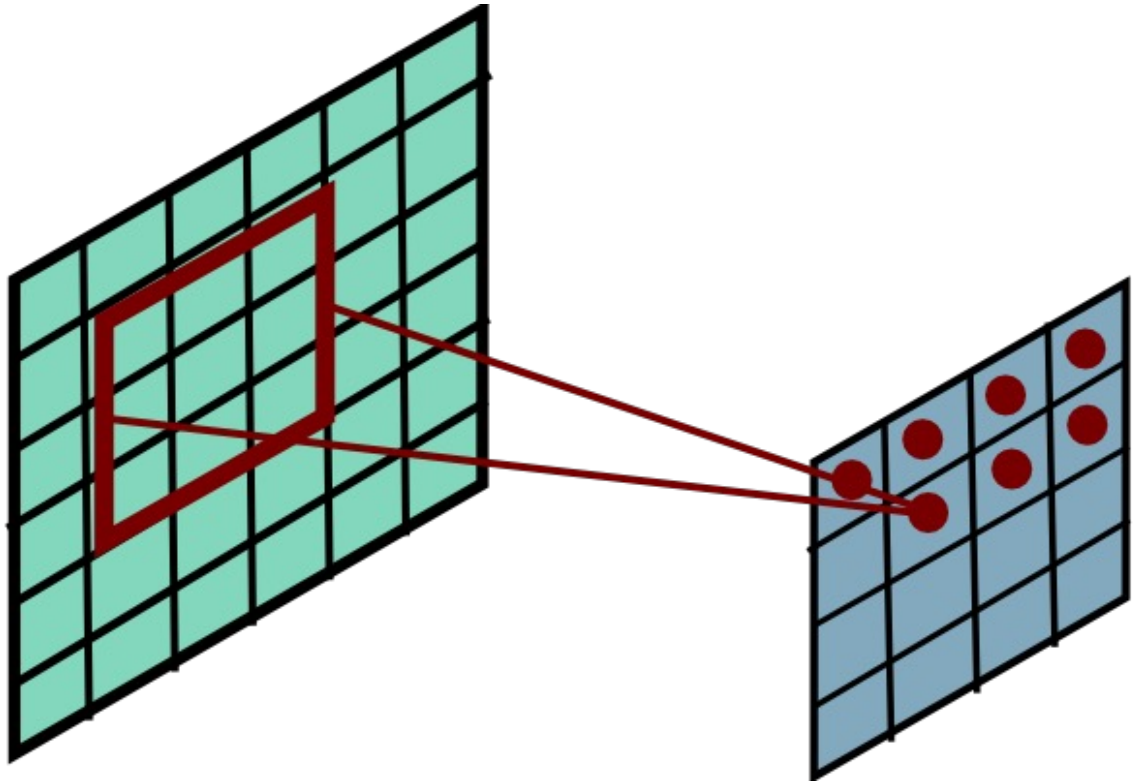
Convolutional Layer



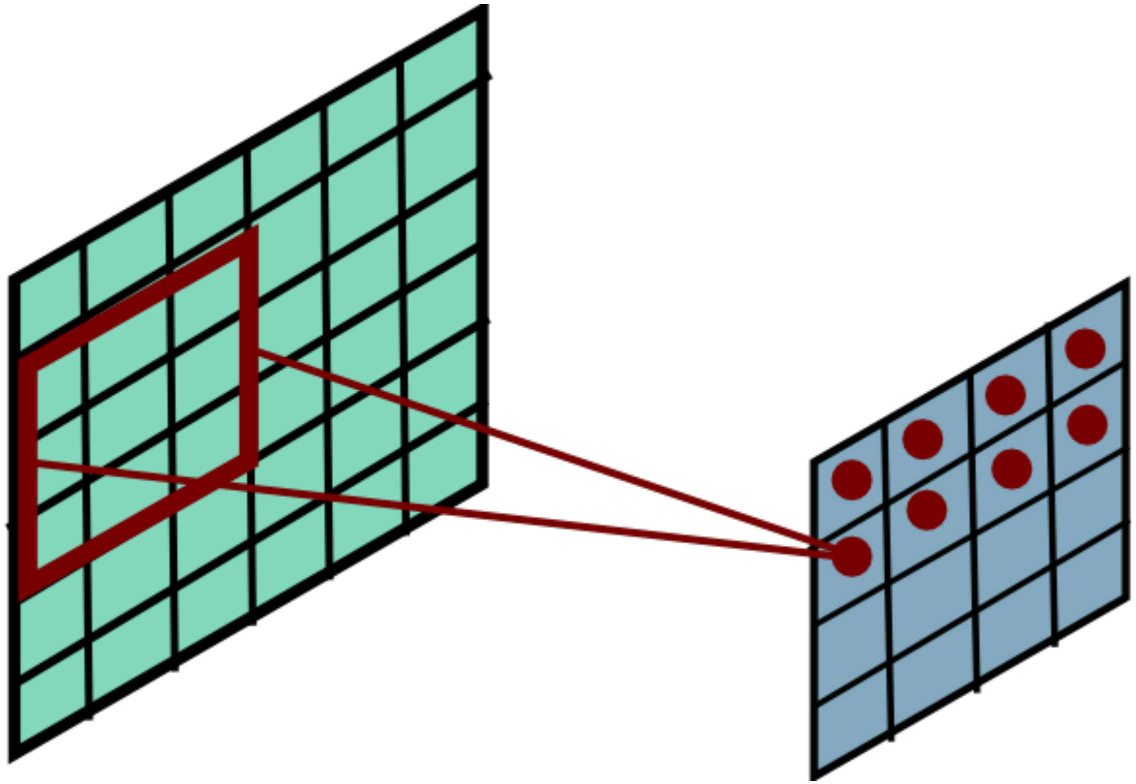
Convolutional Layer



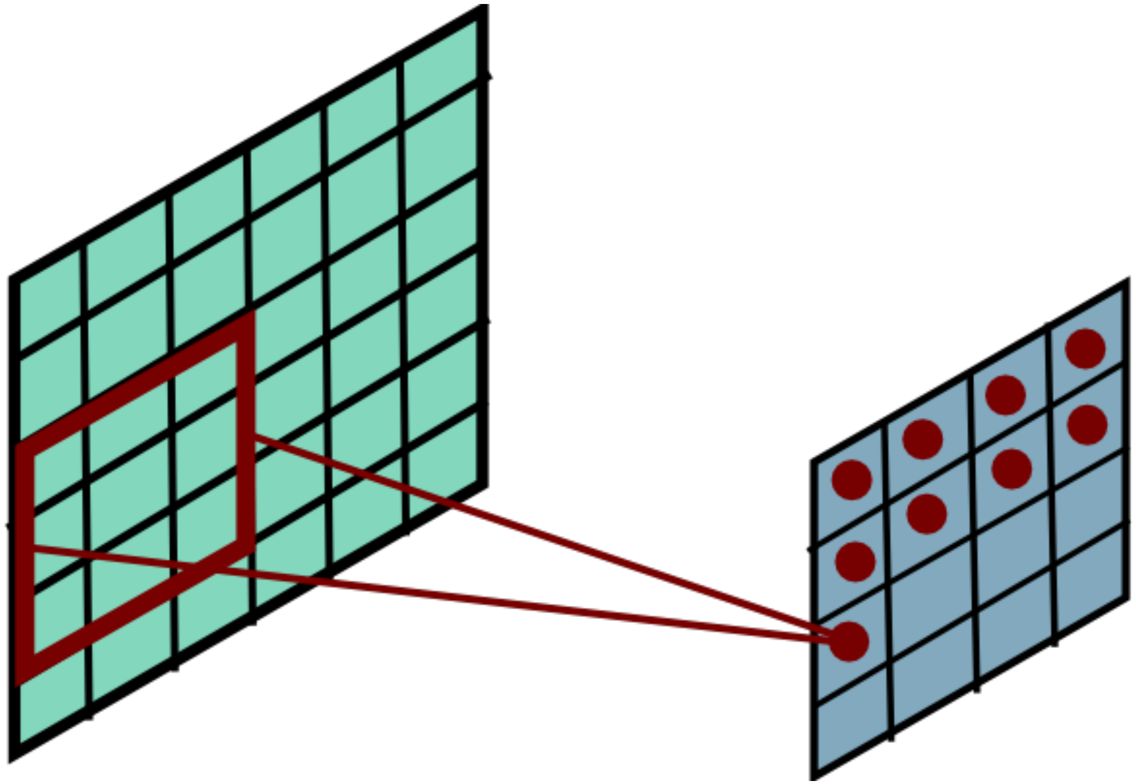
Convolutional Layer



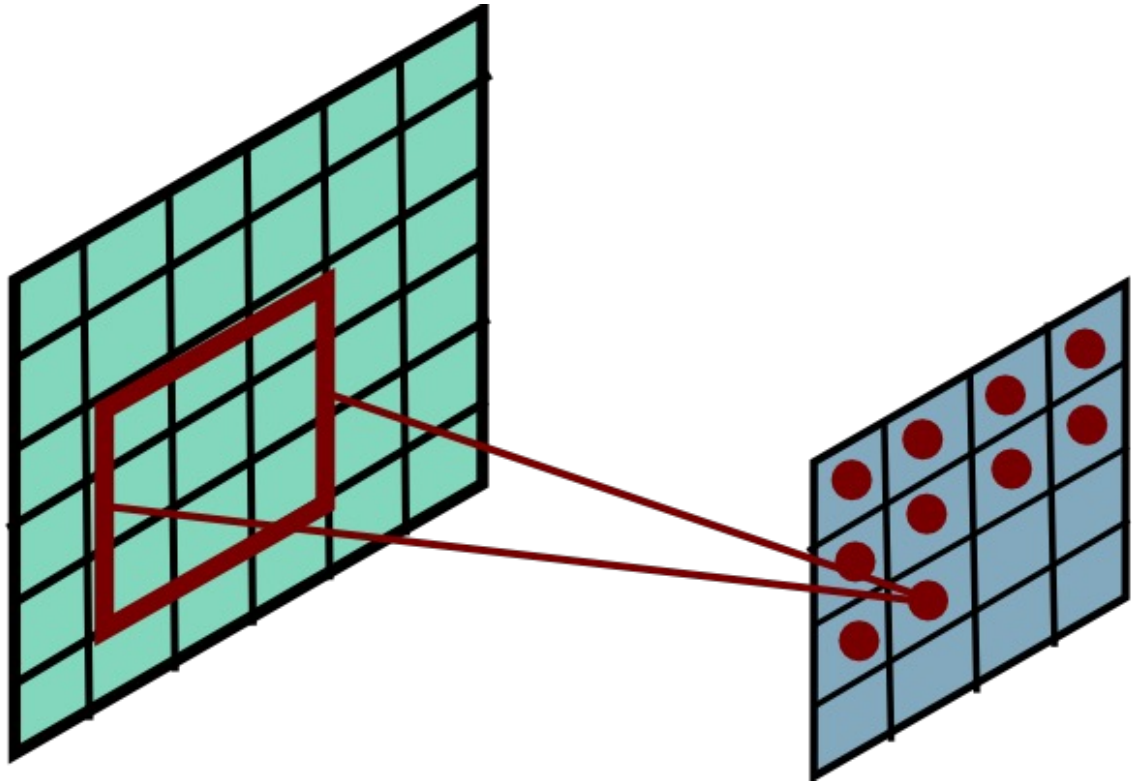
Convolutional Layer



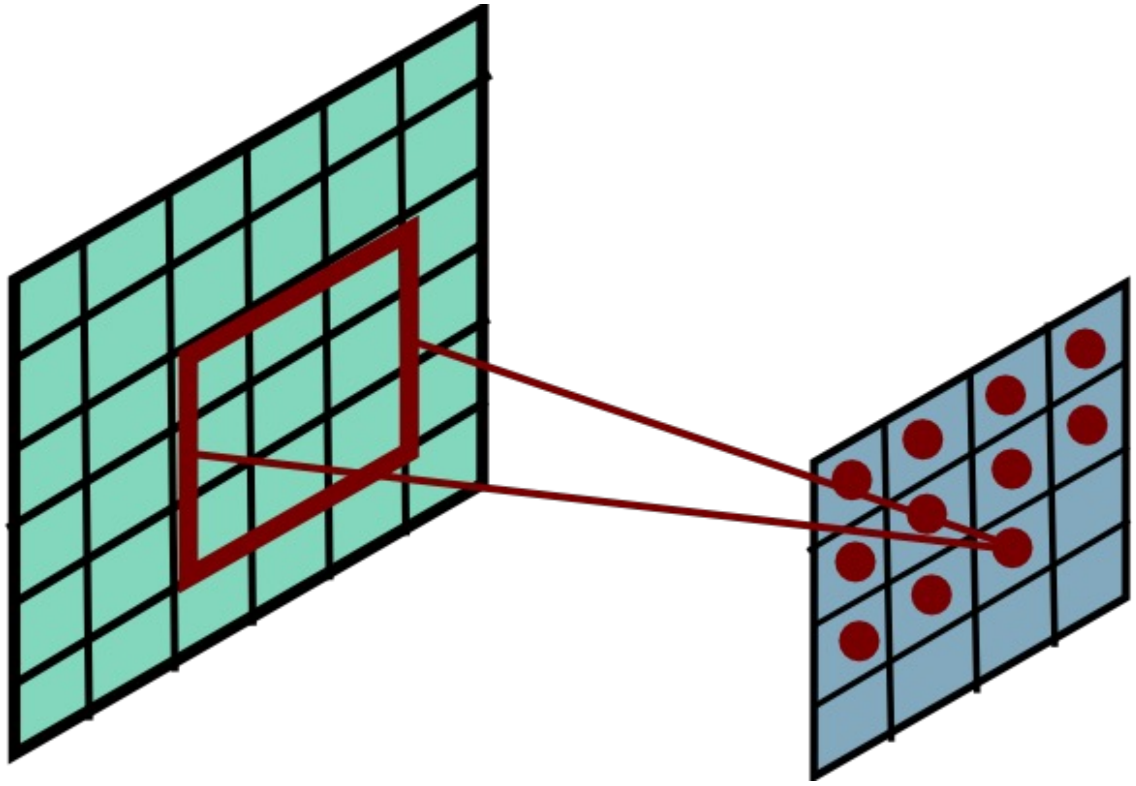
Convolutional Layer



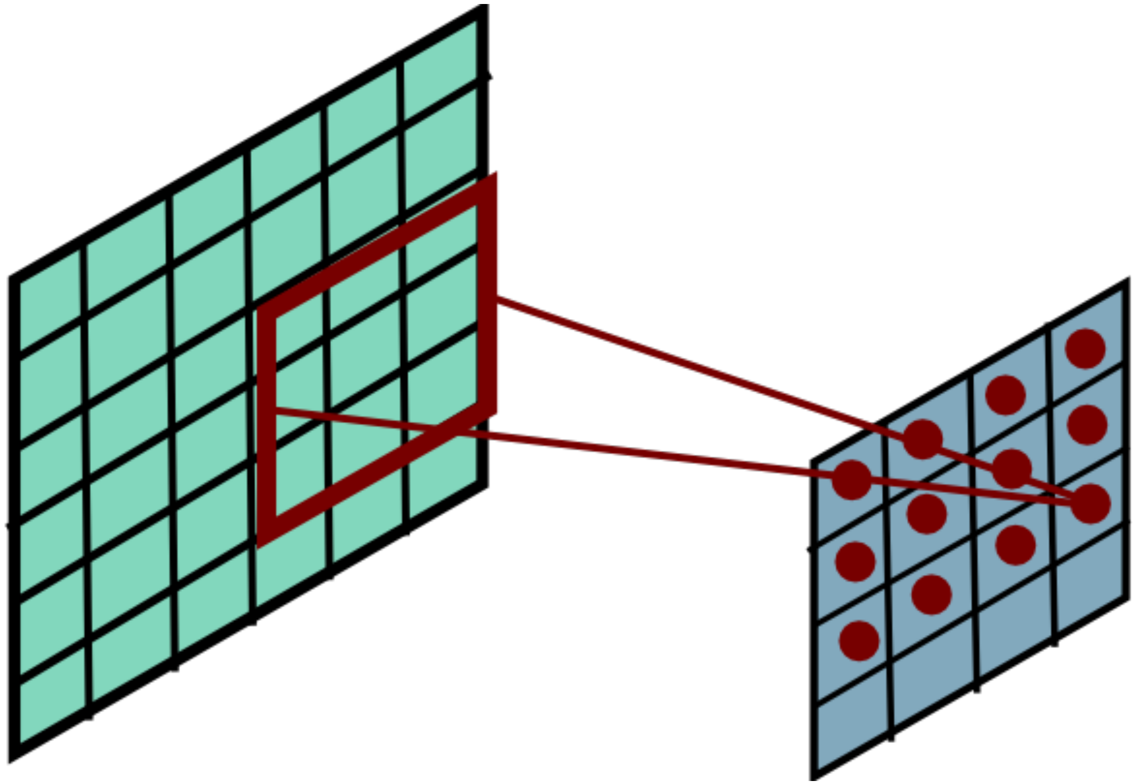
Convolutional Layer



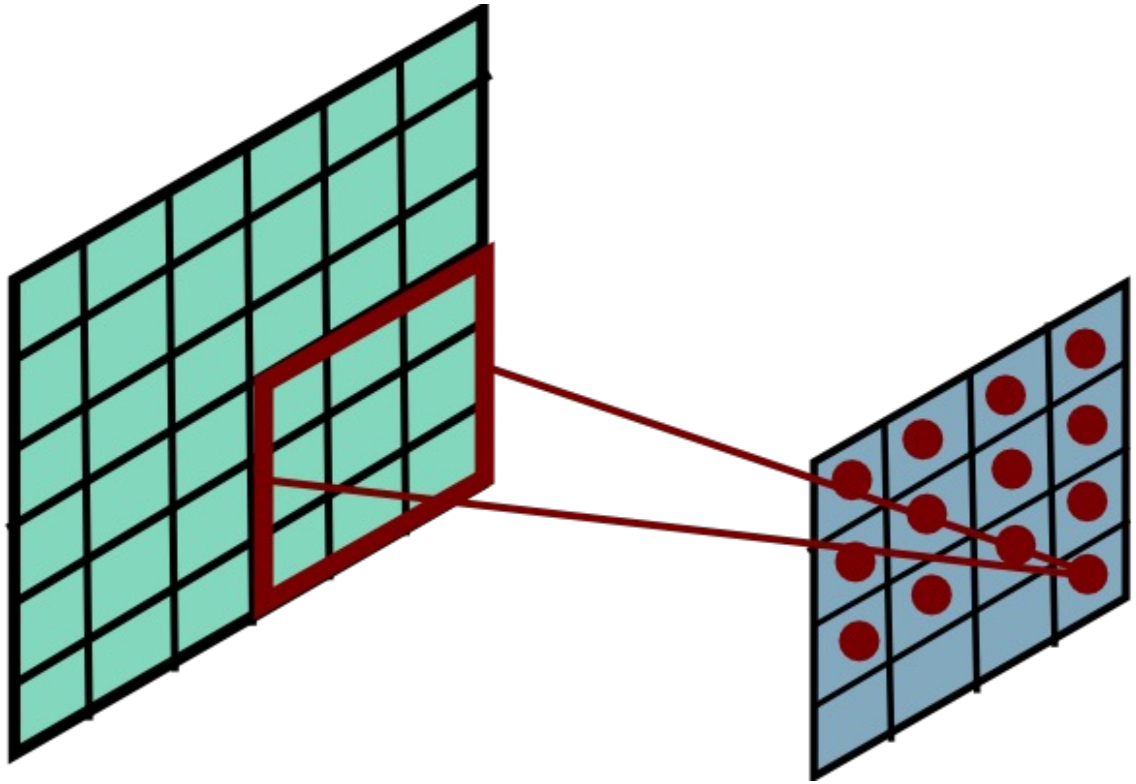
Convolutional Layer



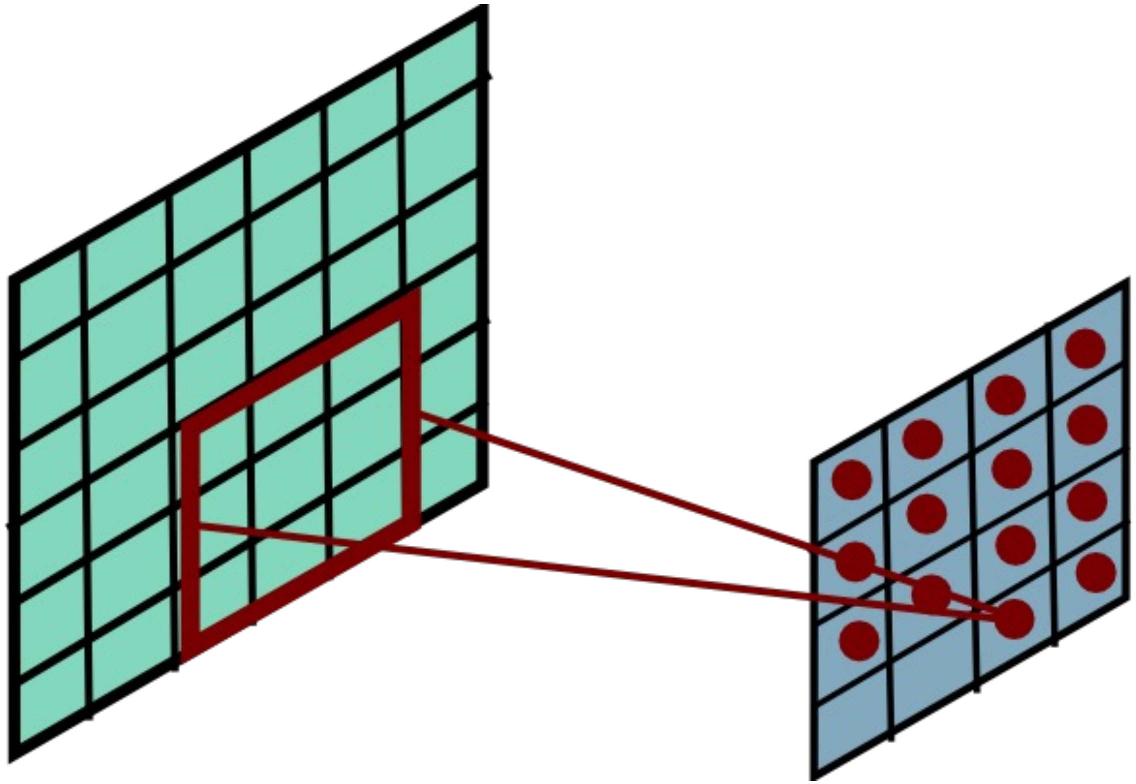
Convolutional Layer



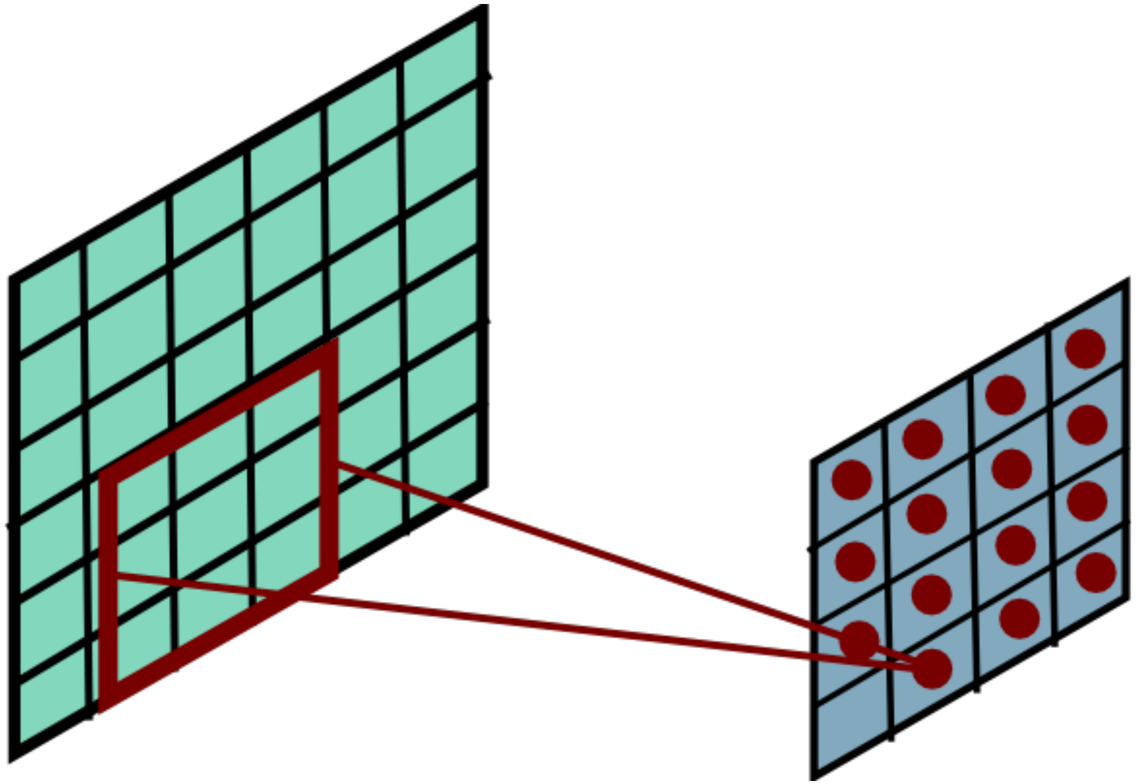
Convolutional Layer



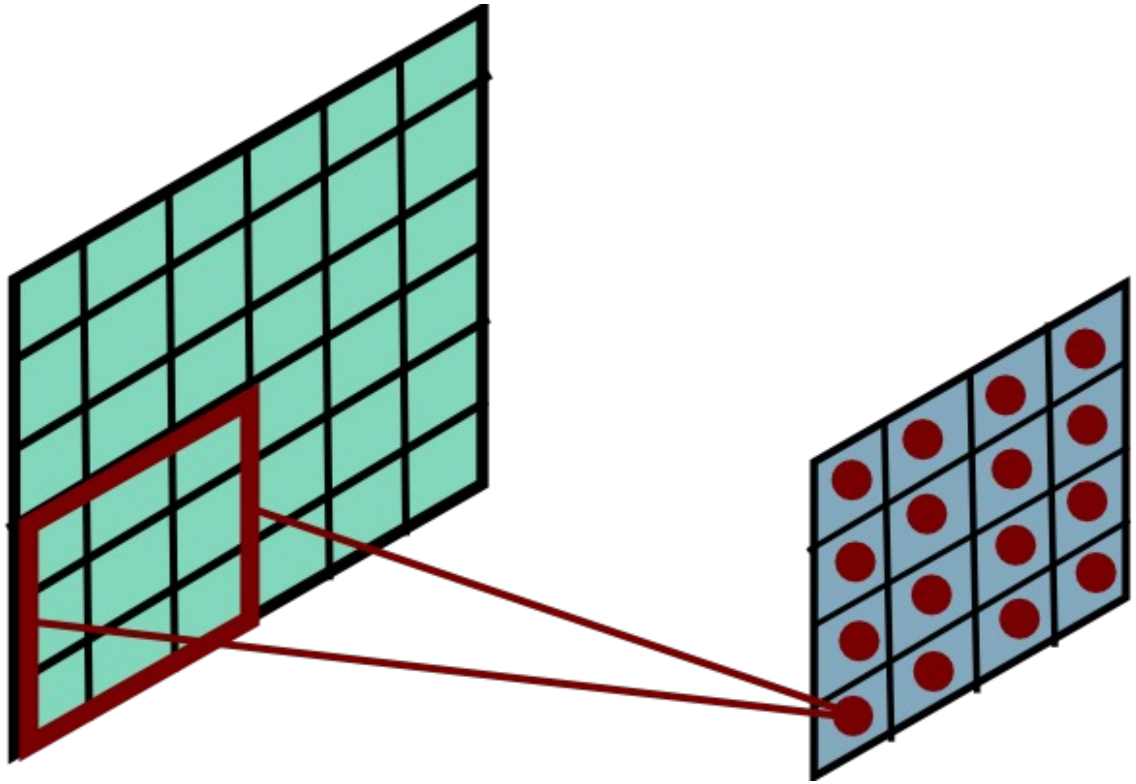
Convolutional Layer



Convolutional Layer



Convolutional Layer



Convolution Explained

- http://setosa.io/ev/image-kernels/

- https://github.com/bruckner/deepViz

Key Ideas

- Take advantage of properties of natural signals
  - Local connections
  - Shared weights
  - Pooling
  - Use of many layers

Comparison with Regular NNs

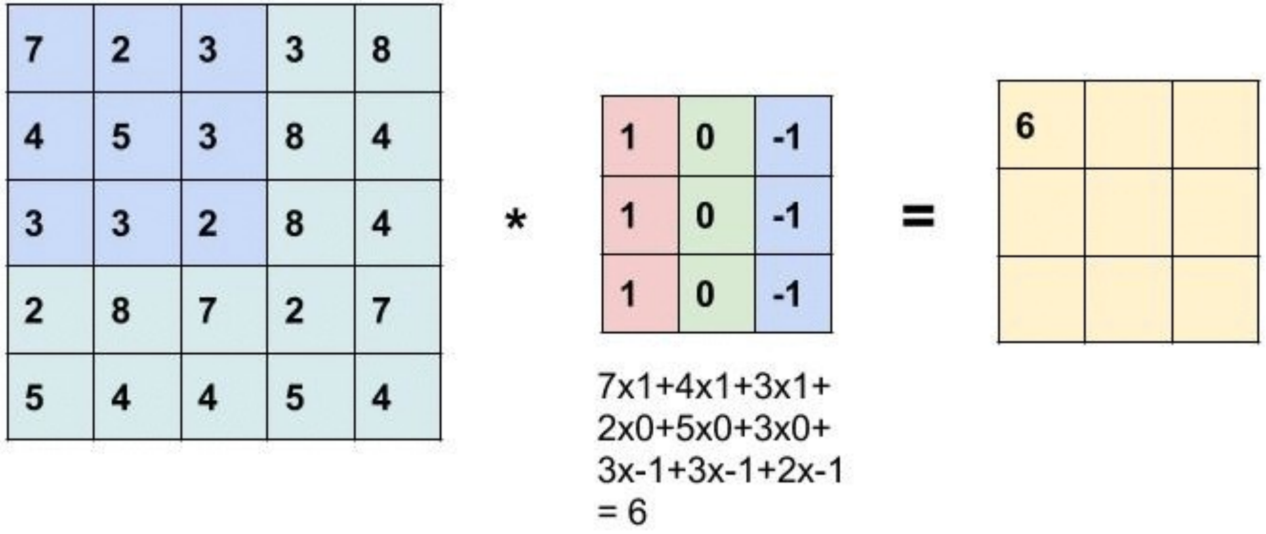
- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**

Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**
- **Exploitation of Local Properties!**
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image

Basic Mechanisms in CNNs

- Three Mechanisms of Convolutional Neural Networks:
  - Convolution Operation
  - Local Receptive Fields
  - Subsampling
  - Weight (Parameter) Sharing

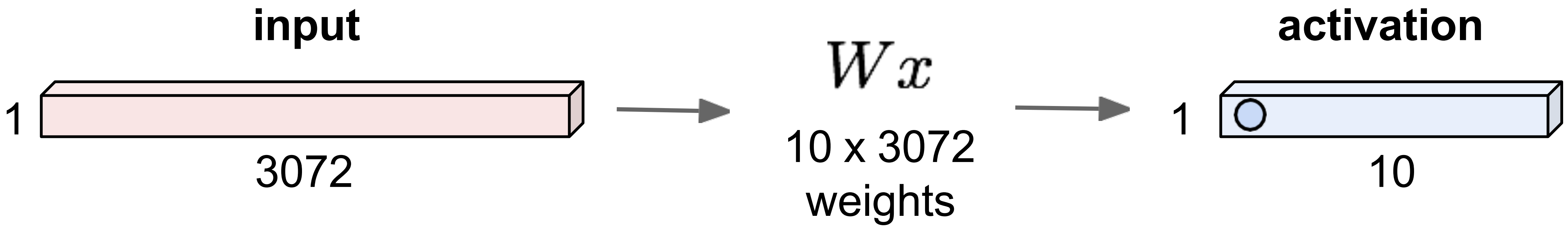


Visualization of convolution: https://github.com/effat/MLP-Demo/blob/main/Convolution_gif.gif

https://medium.com/@Tms43/understanding-padding-strides-in-convolutional-neural-networks-cnn-for-effective-image-feature-1b0756a52918

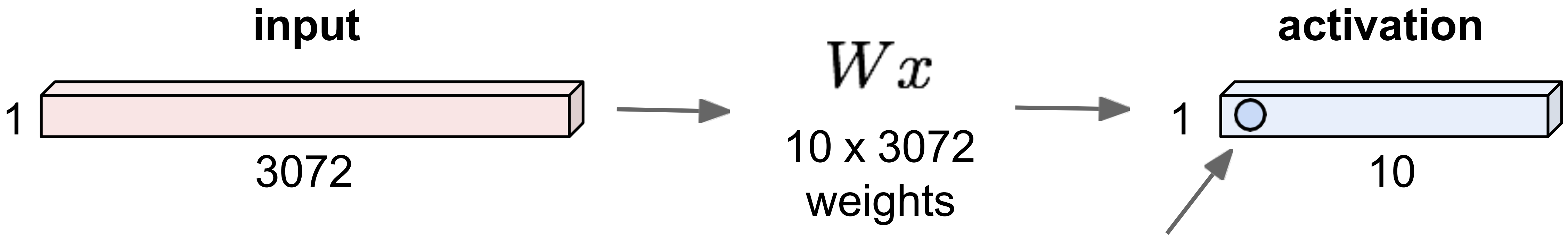
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

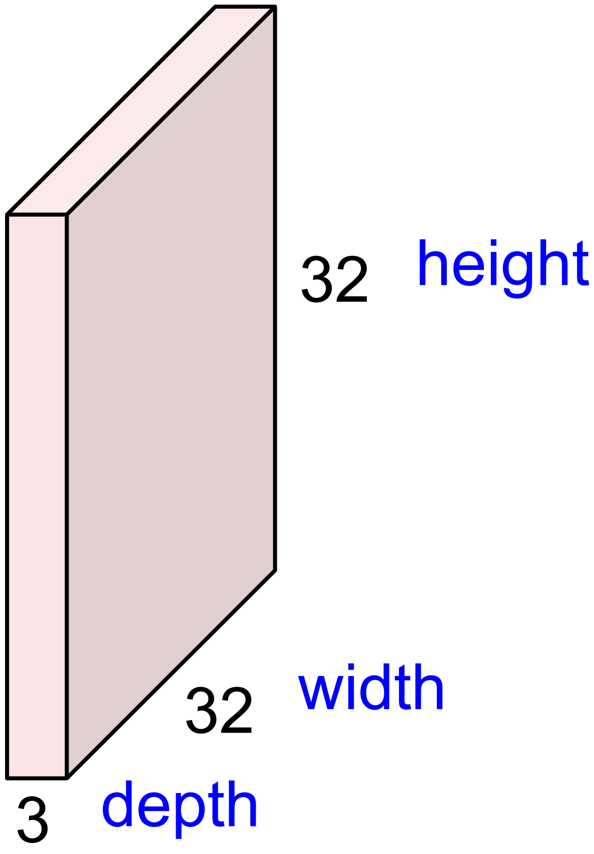
32x32x3 image -> stretch to 3072 x 1



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

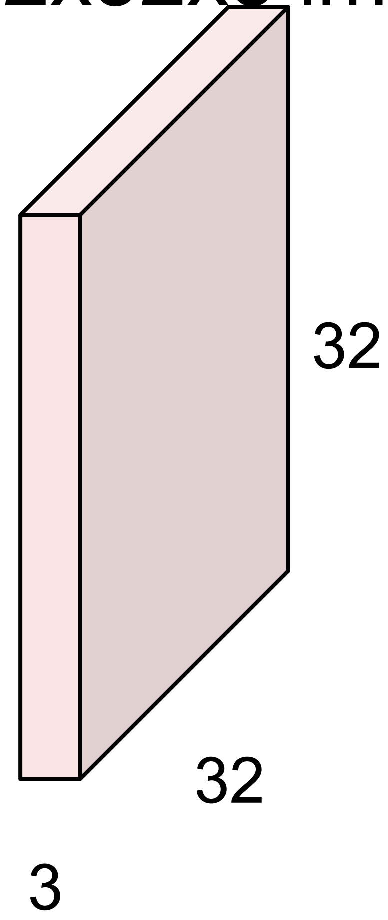
Convolution Layer

32x32x3 image -> preserve spatial structure

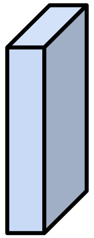


Convolution Layer

32x32x3 image



5x5x3 filter

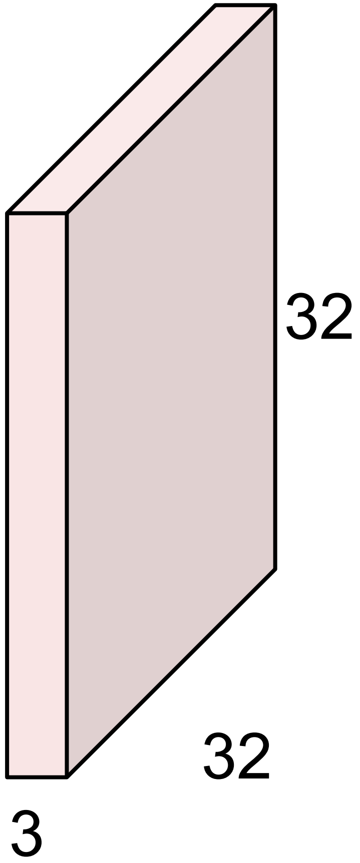


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

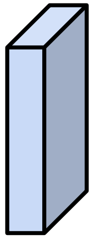
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

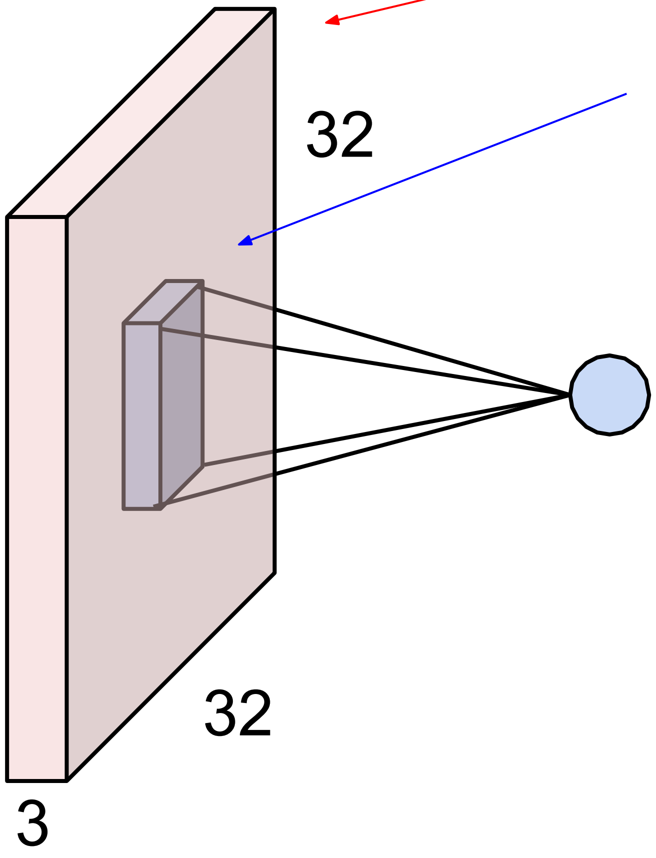


Convolve the filter with the image i.e. “slide over the image spatially, computing
dot products”

Convolution Layer

32x32x3 image

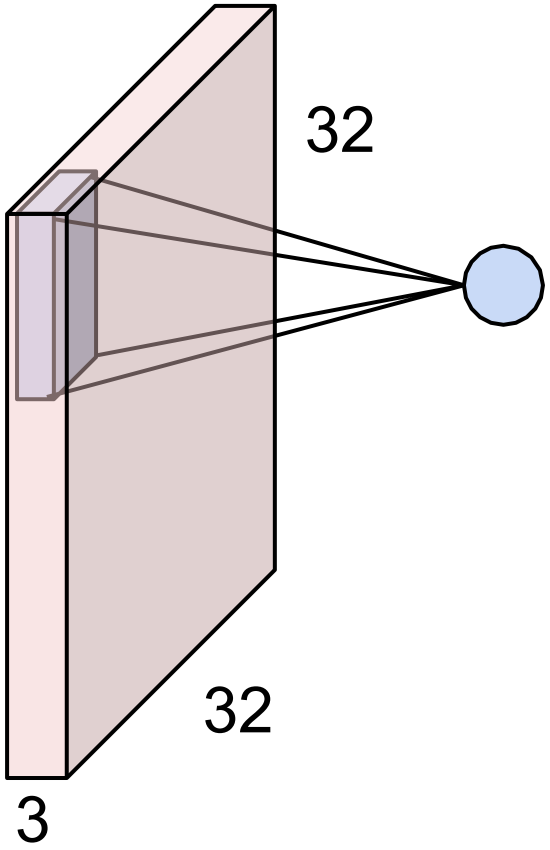
5x5x3 filter w



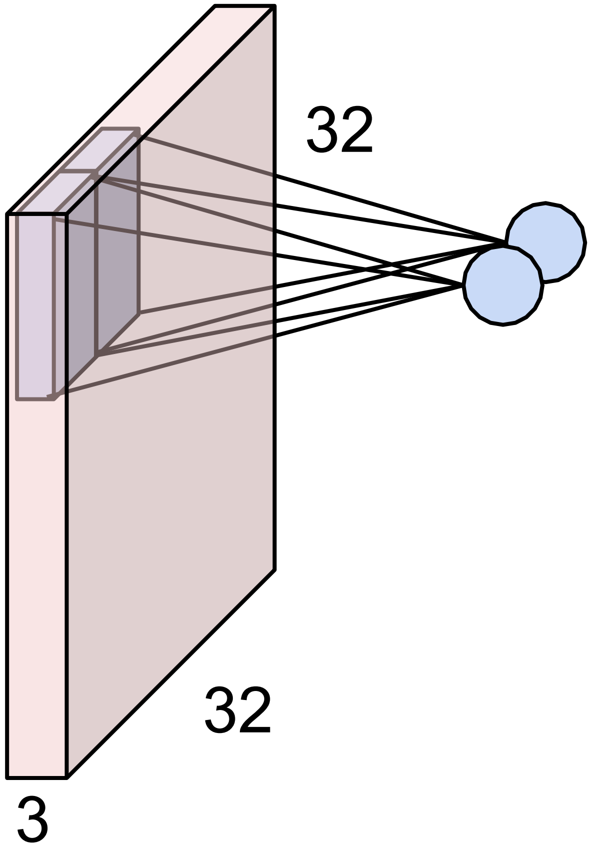
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

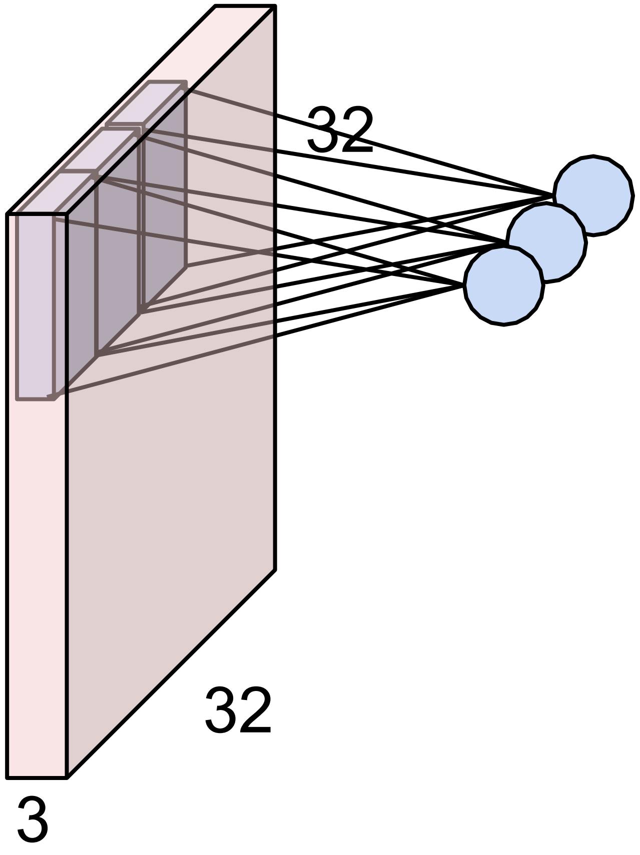
Convolution Layer



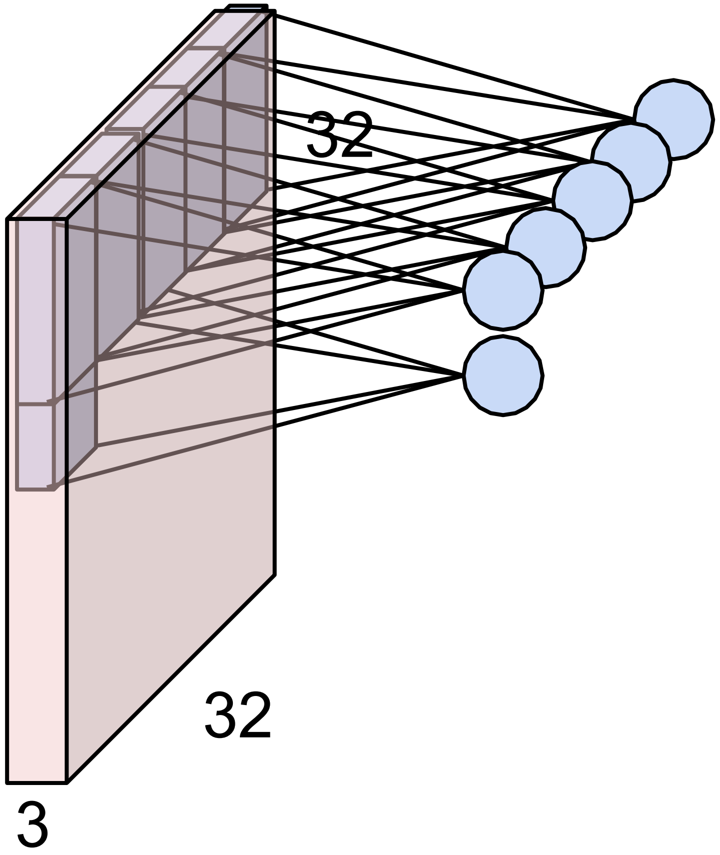
Convolution Layer



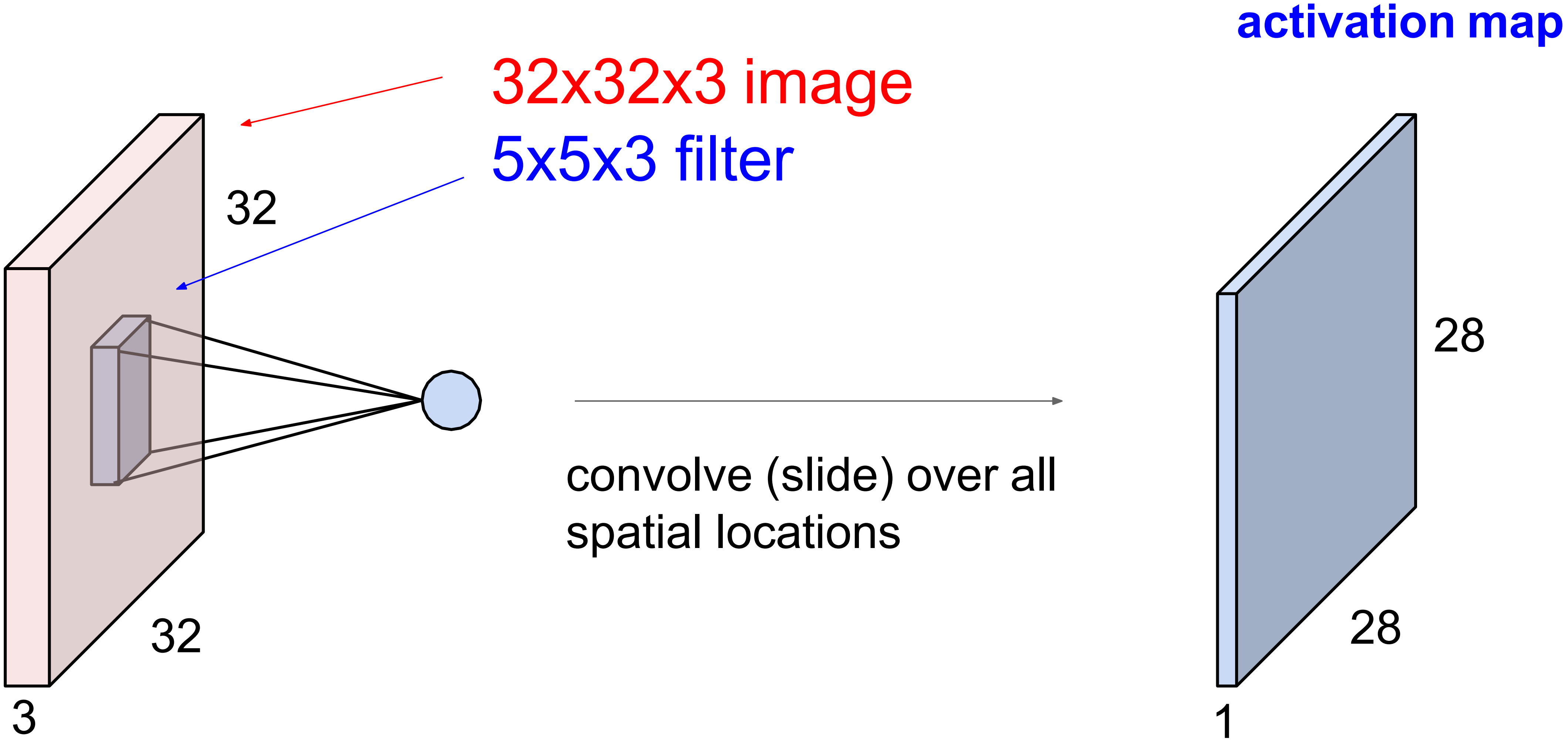
Convolution Layer



Convolution Layer

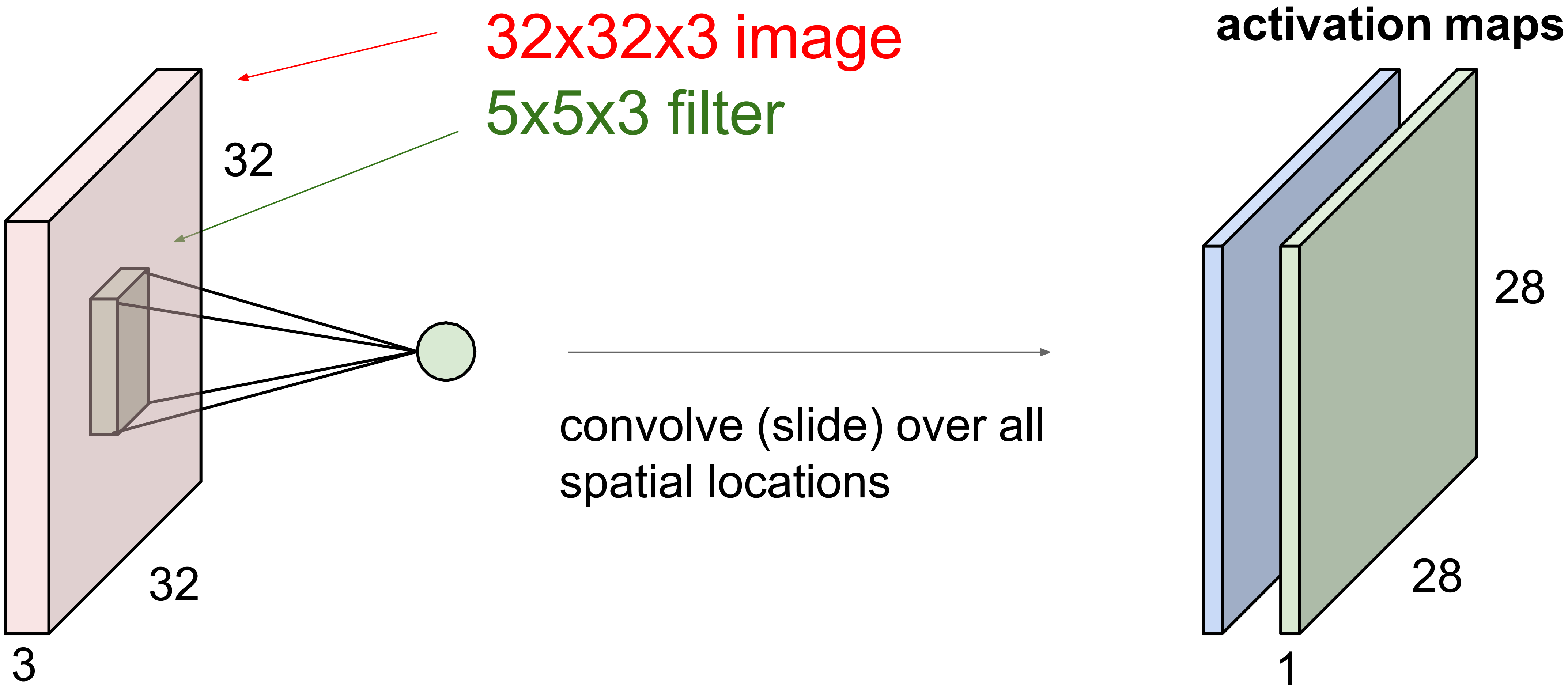


Convolution Layer

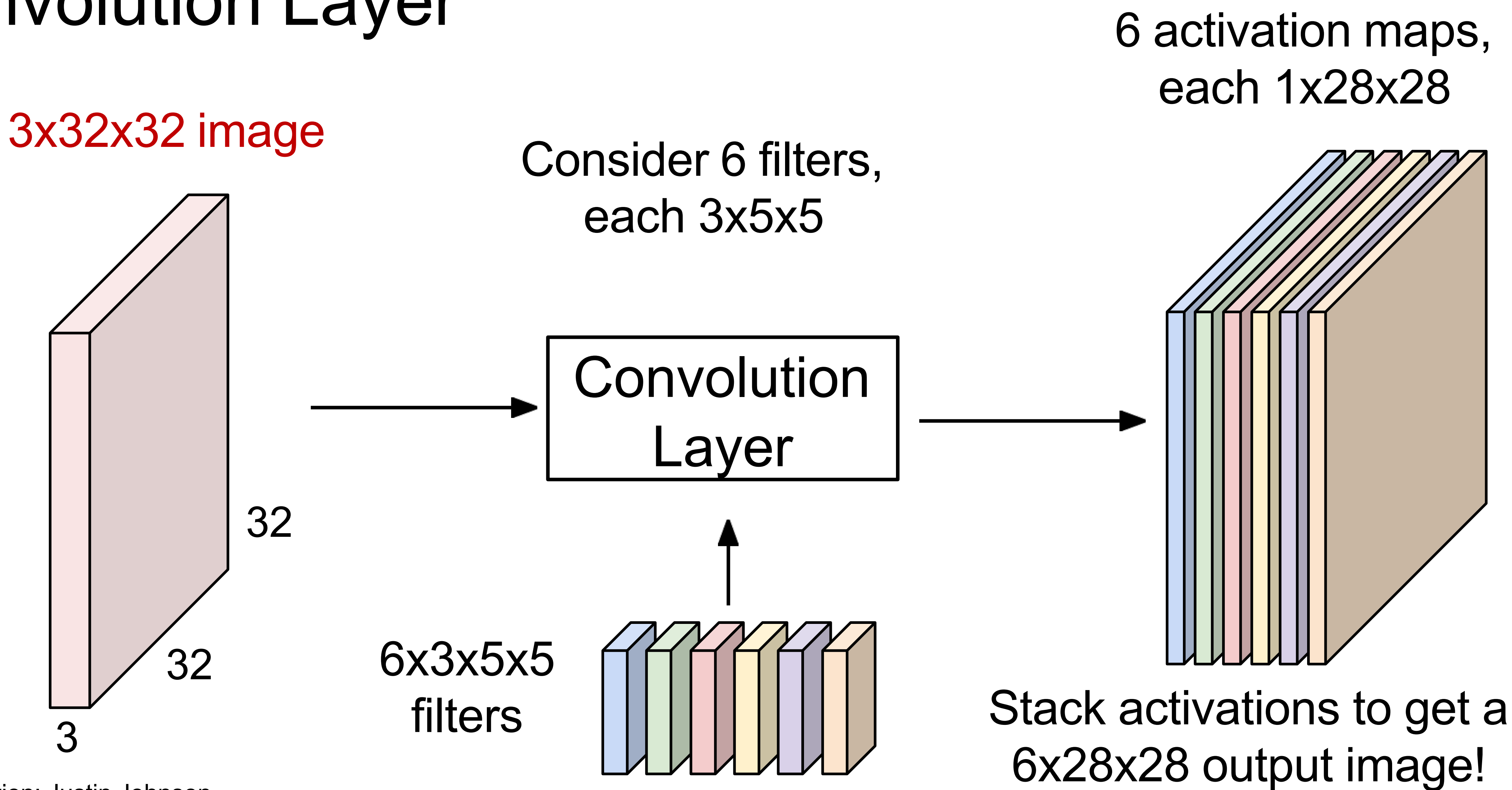


Convolution Layer

consider a second, green filter

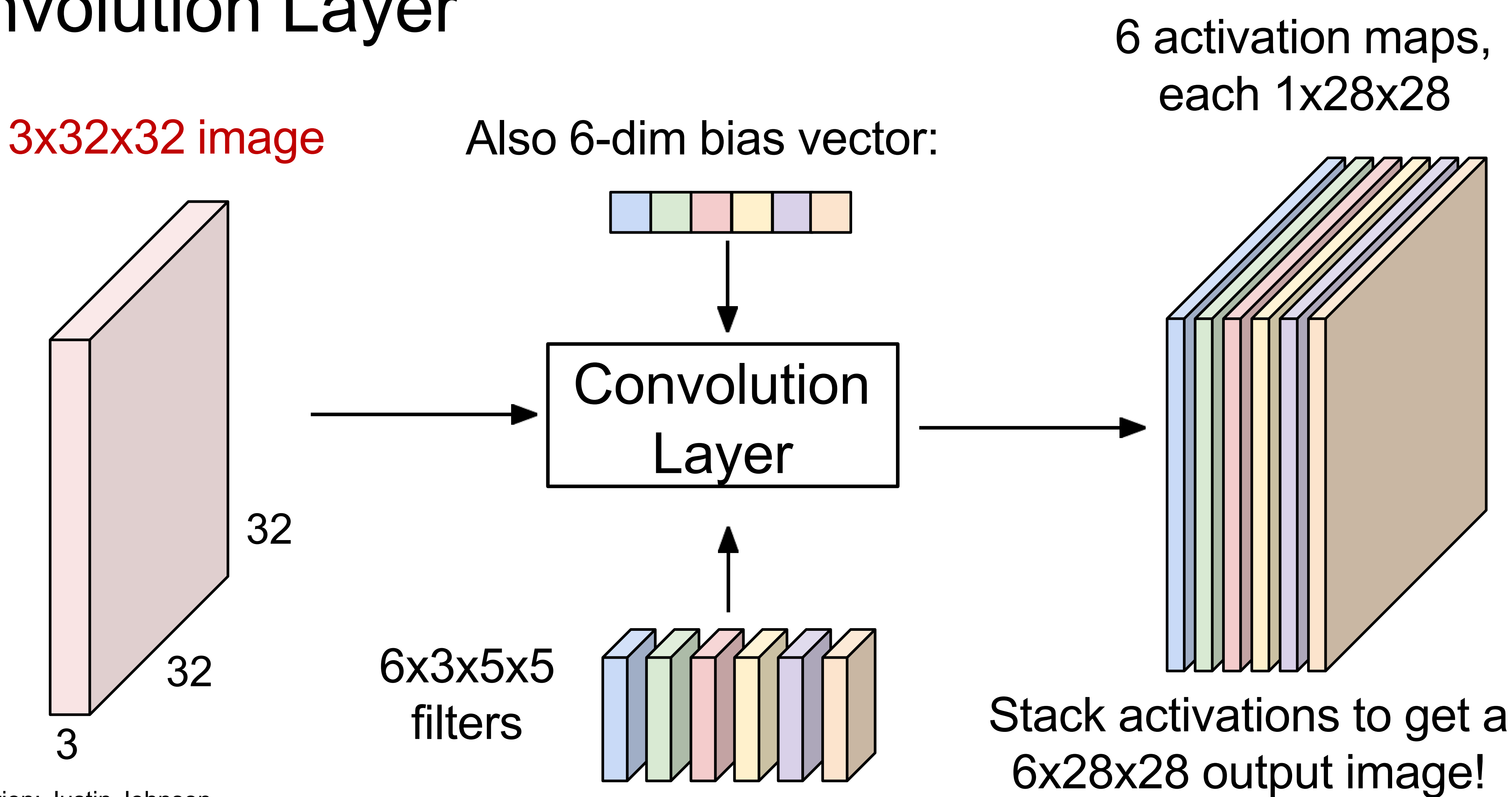


Convolution Layer



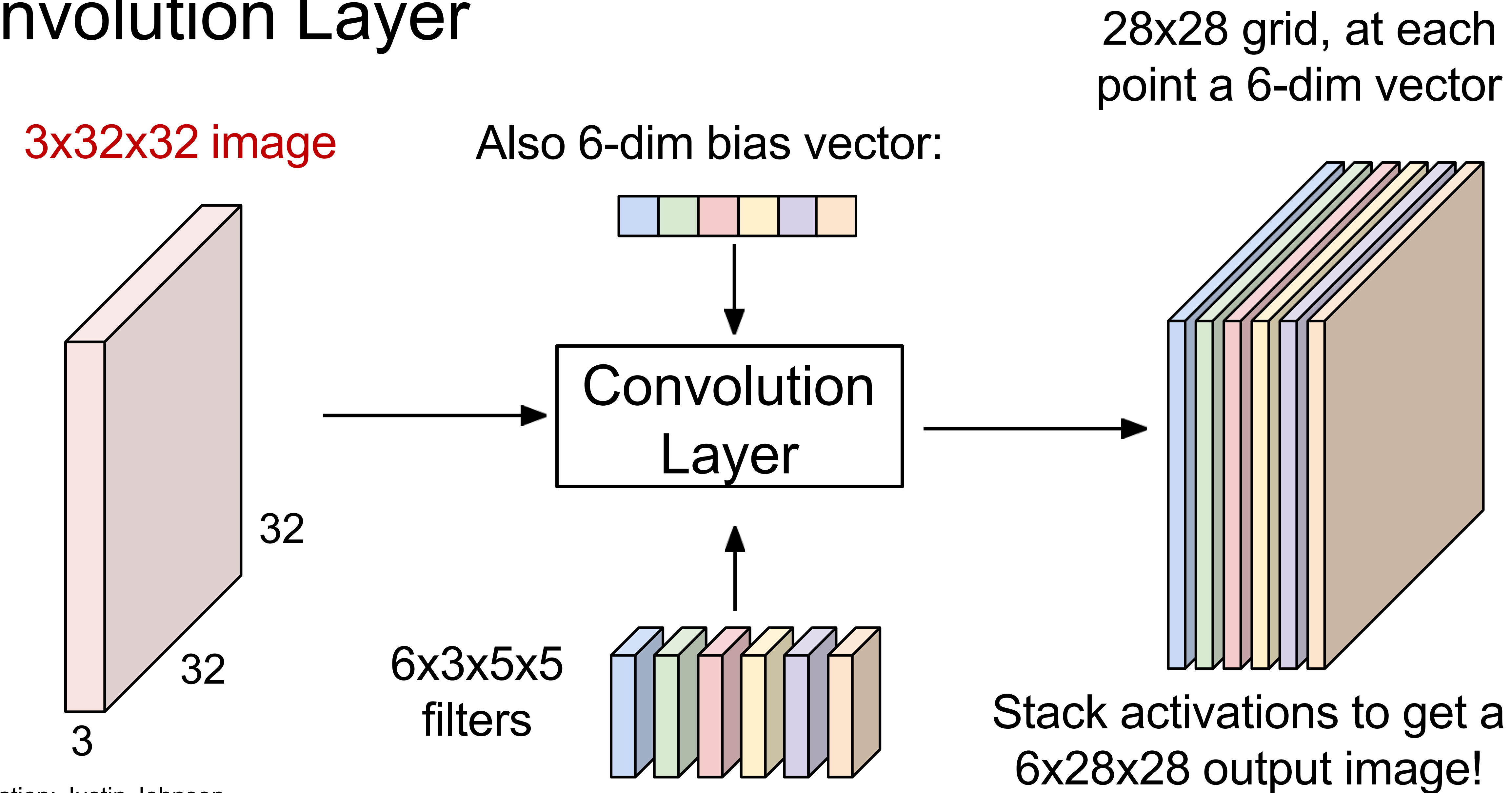
Slide inspiration: Justin Johnson

Convolution Layer



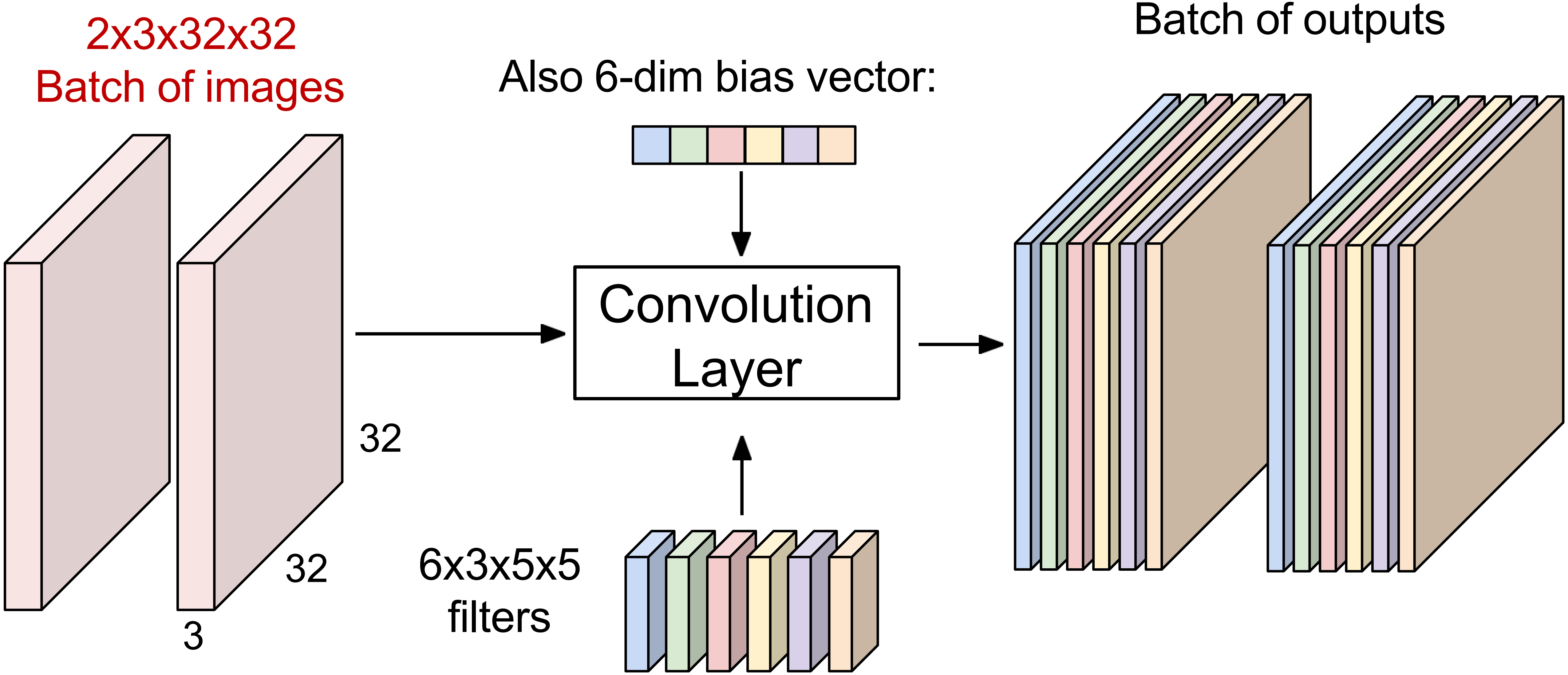
Slide inspiration: Justin Johnson

Convolution Layer



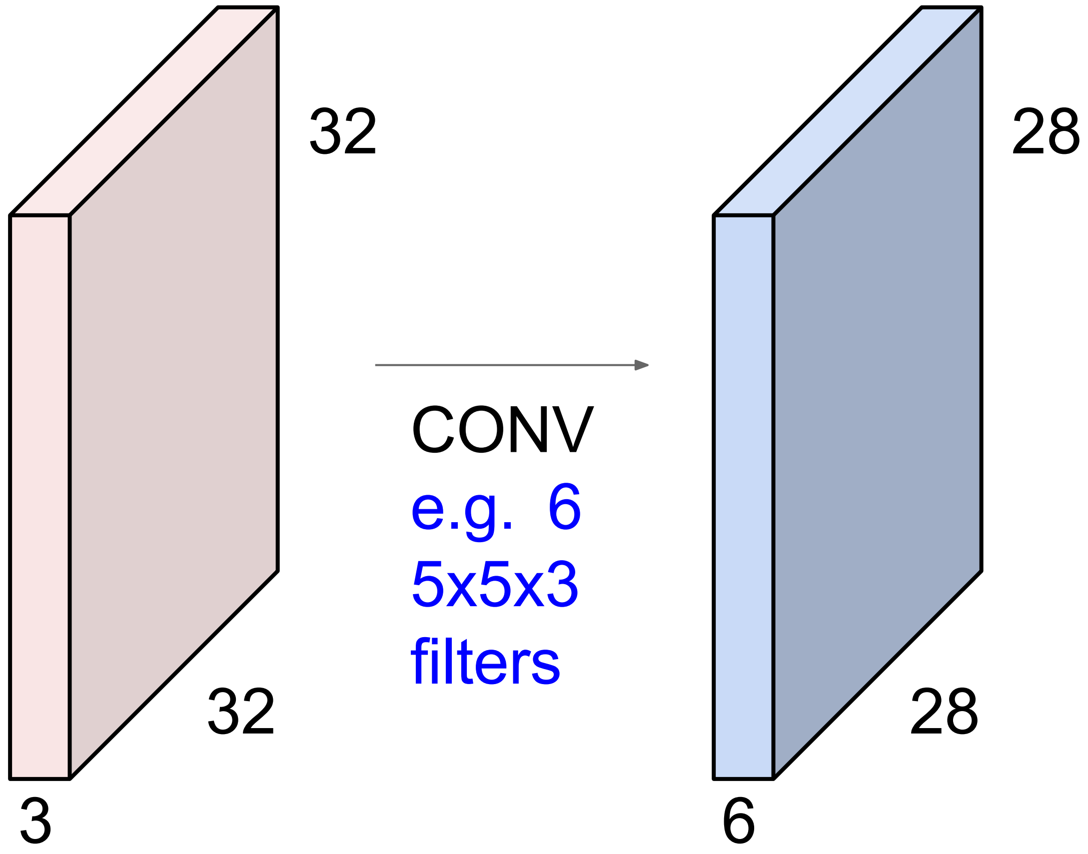
Slide inspiration: Justin Johnson

Convolution Layer

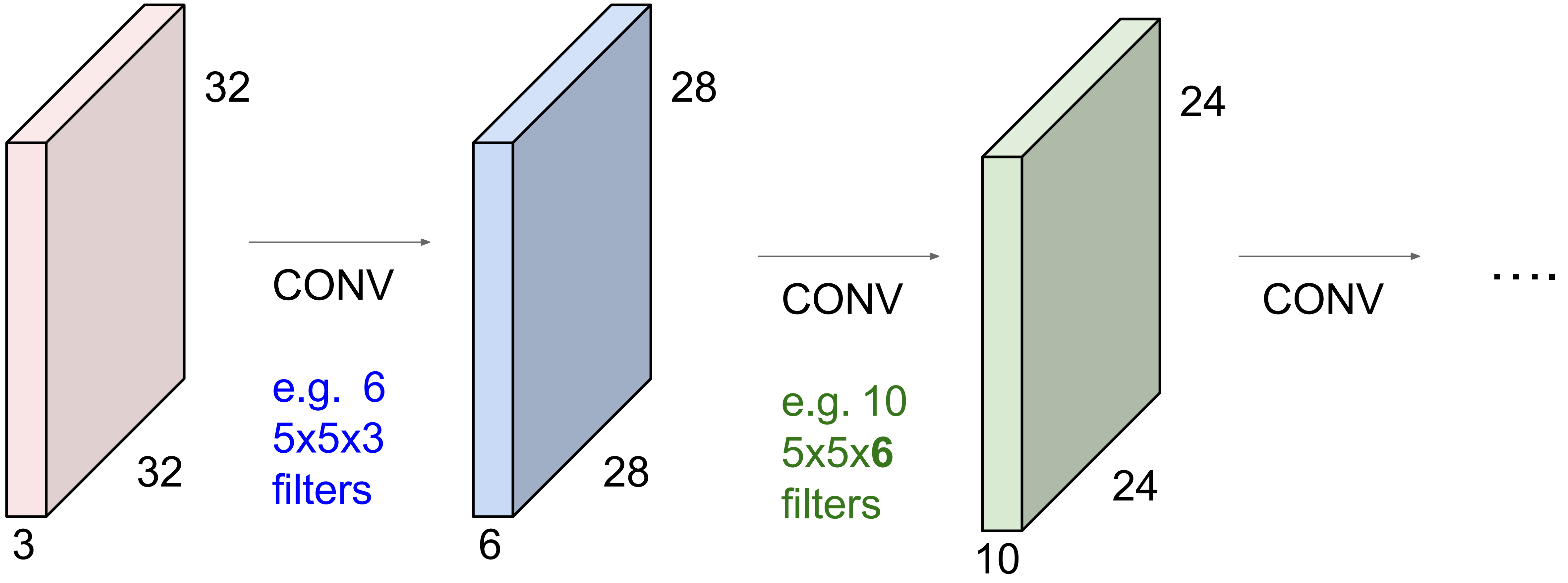


Slide inspiration: Justin Johnson

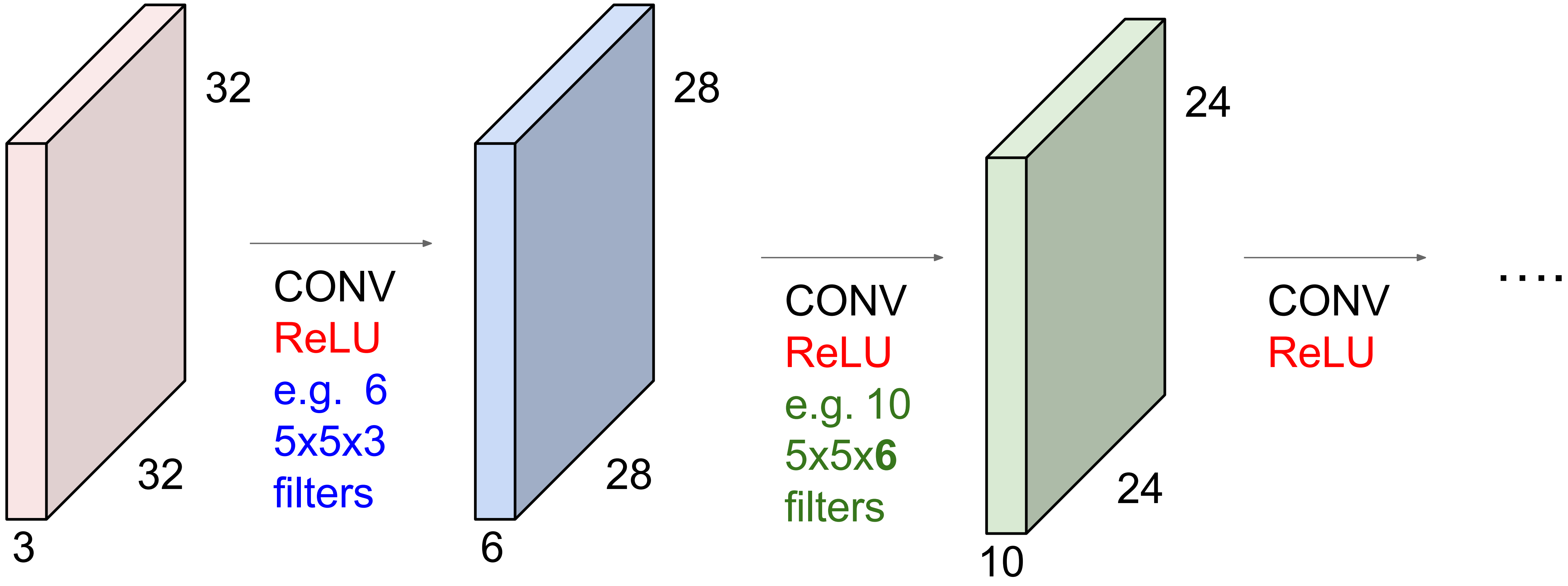
Preview: ConvNet is a sequence of Convolution Layers



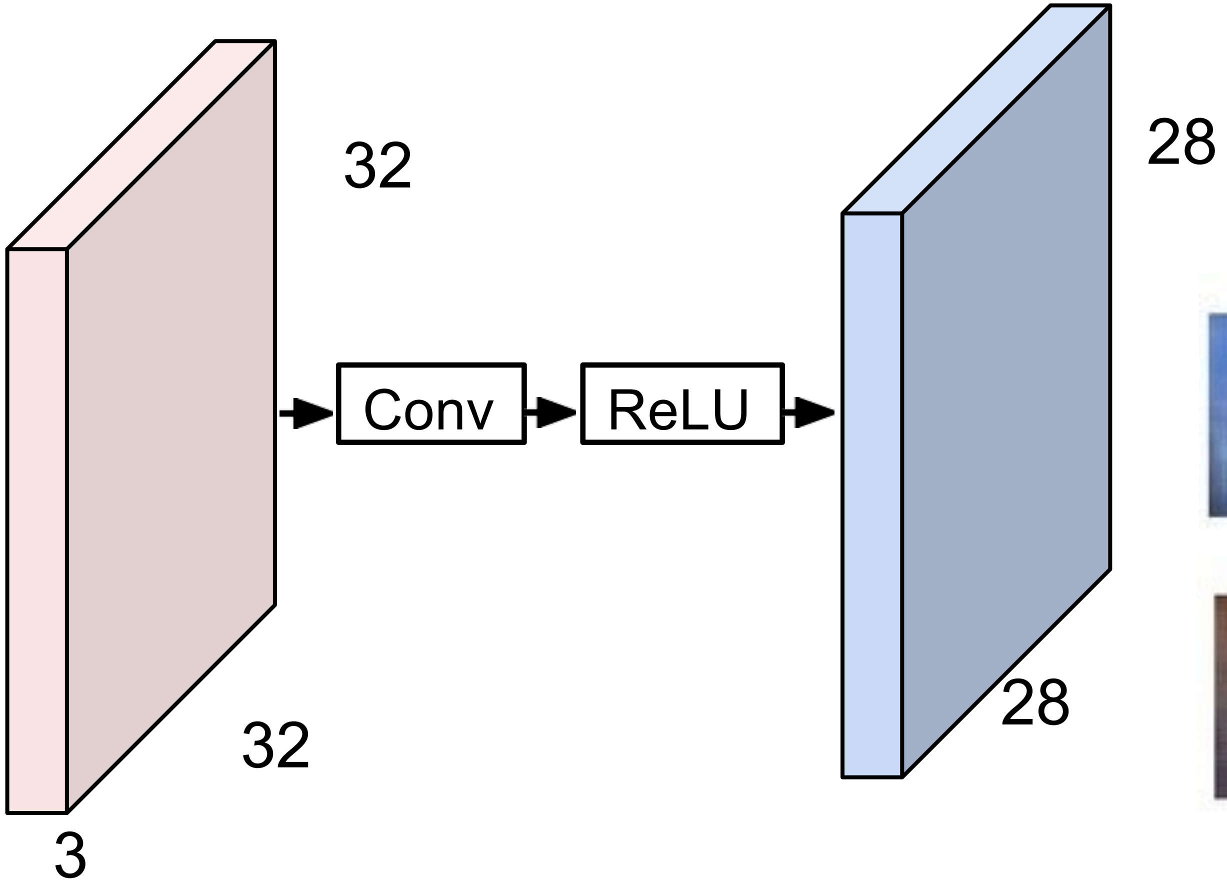
Preview: ConvNet is a sequence of Convolution Layers



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

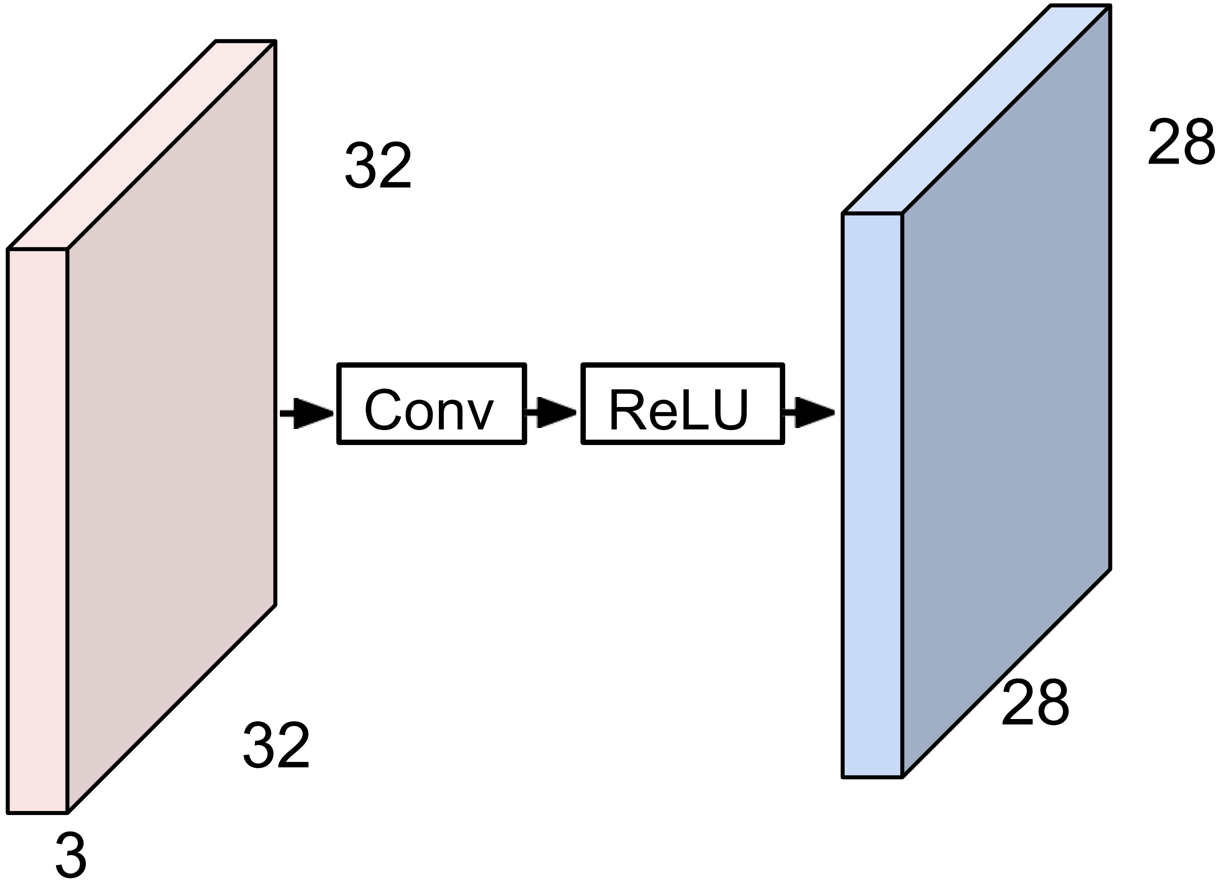


Preview: What do convolutional filters learn?



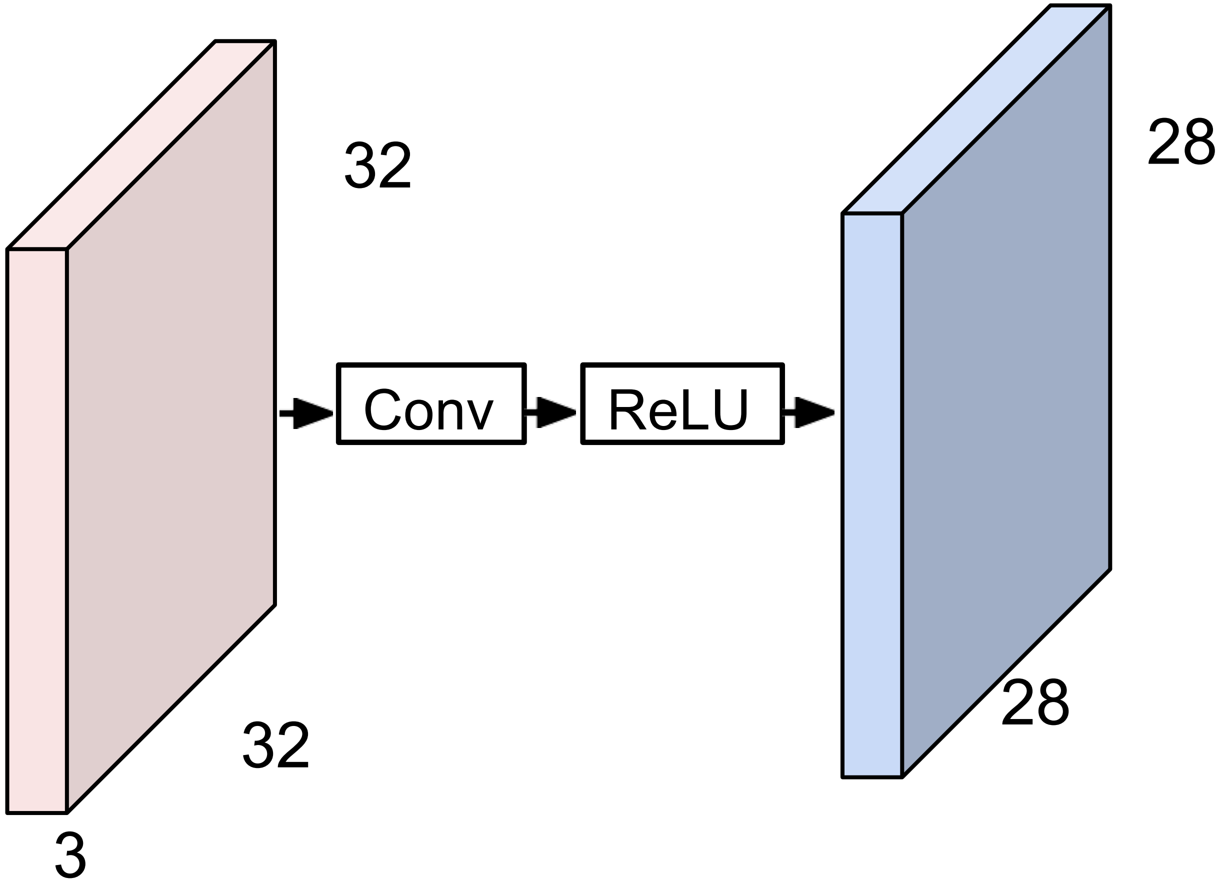
Linear classifier: One template per class

Preview: What do convolutional filters learn?

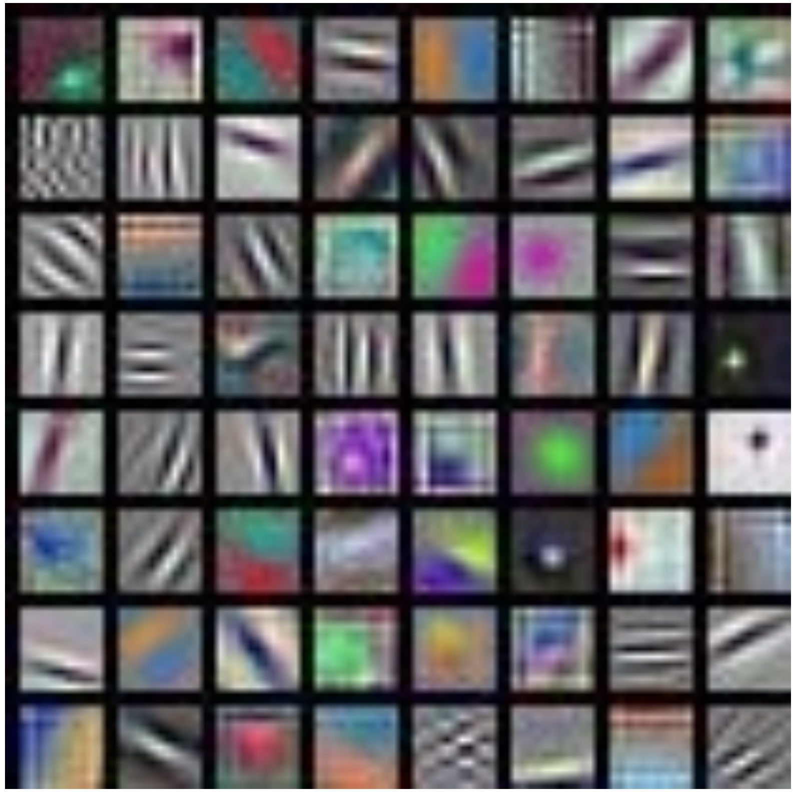


MLP: Bank of whole-image templates

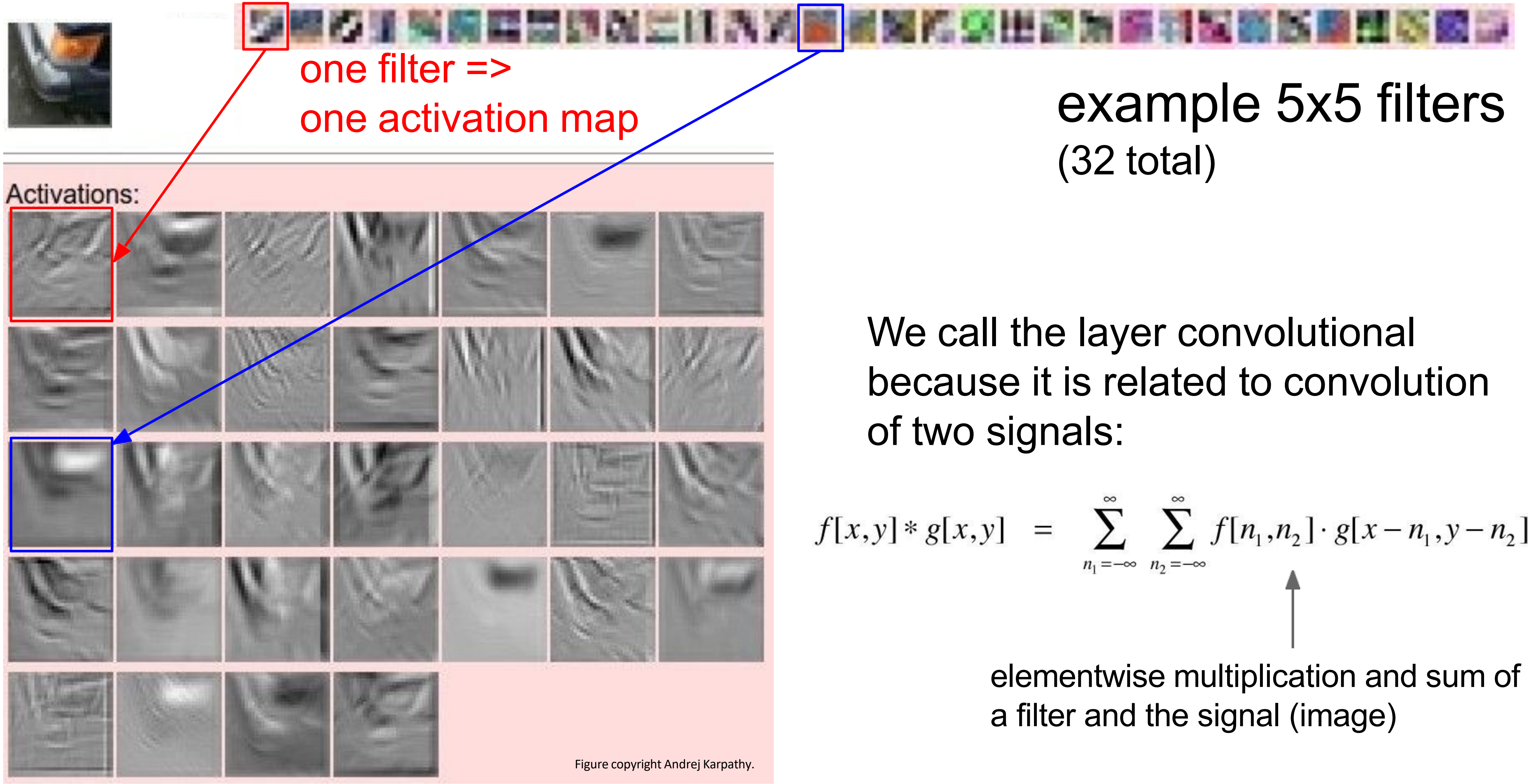
Preview: What do convolutional filters learn?



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



AlexNet: 64 filters, each 3x11x11

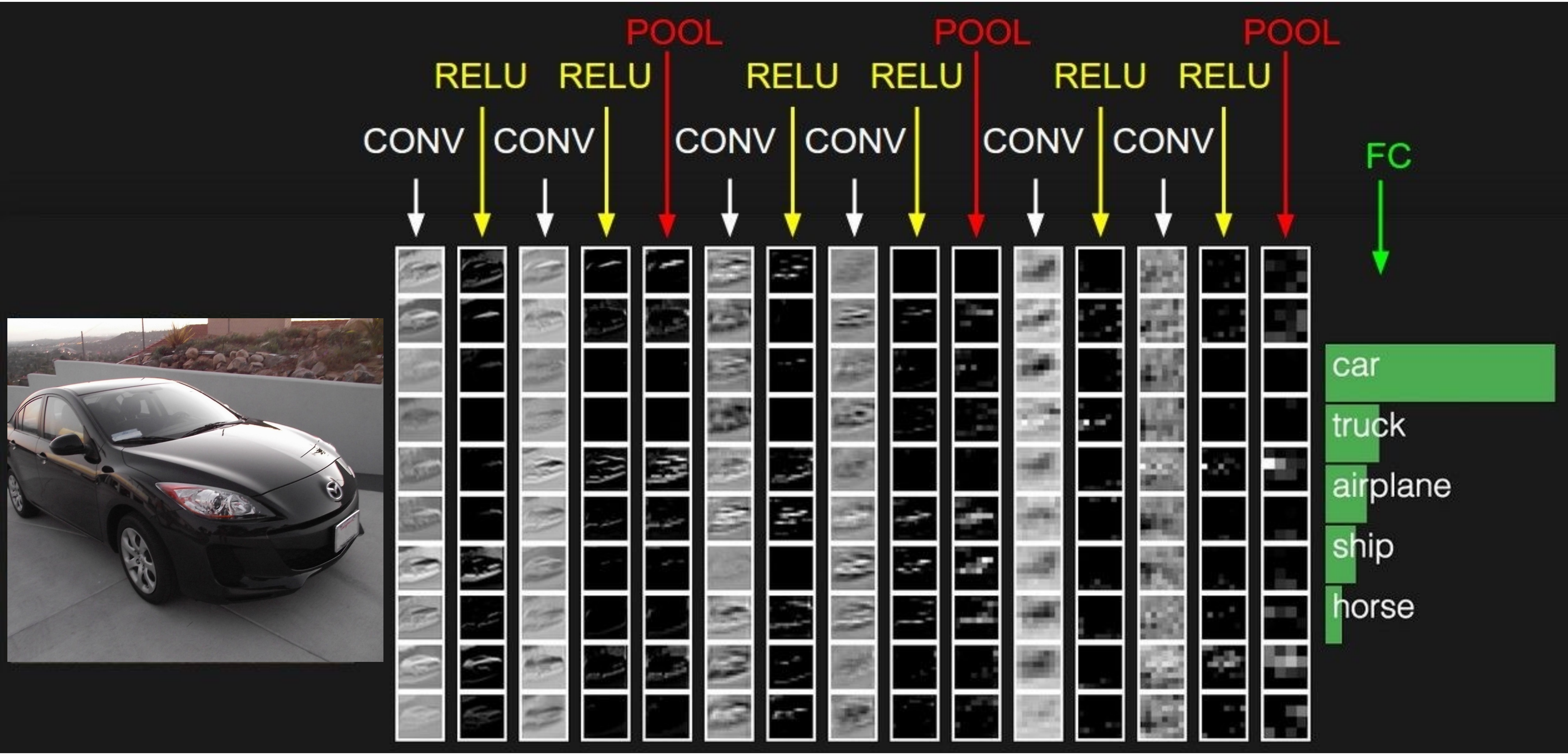


example 5x5 filters
(32 total)

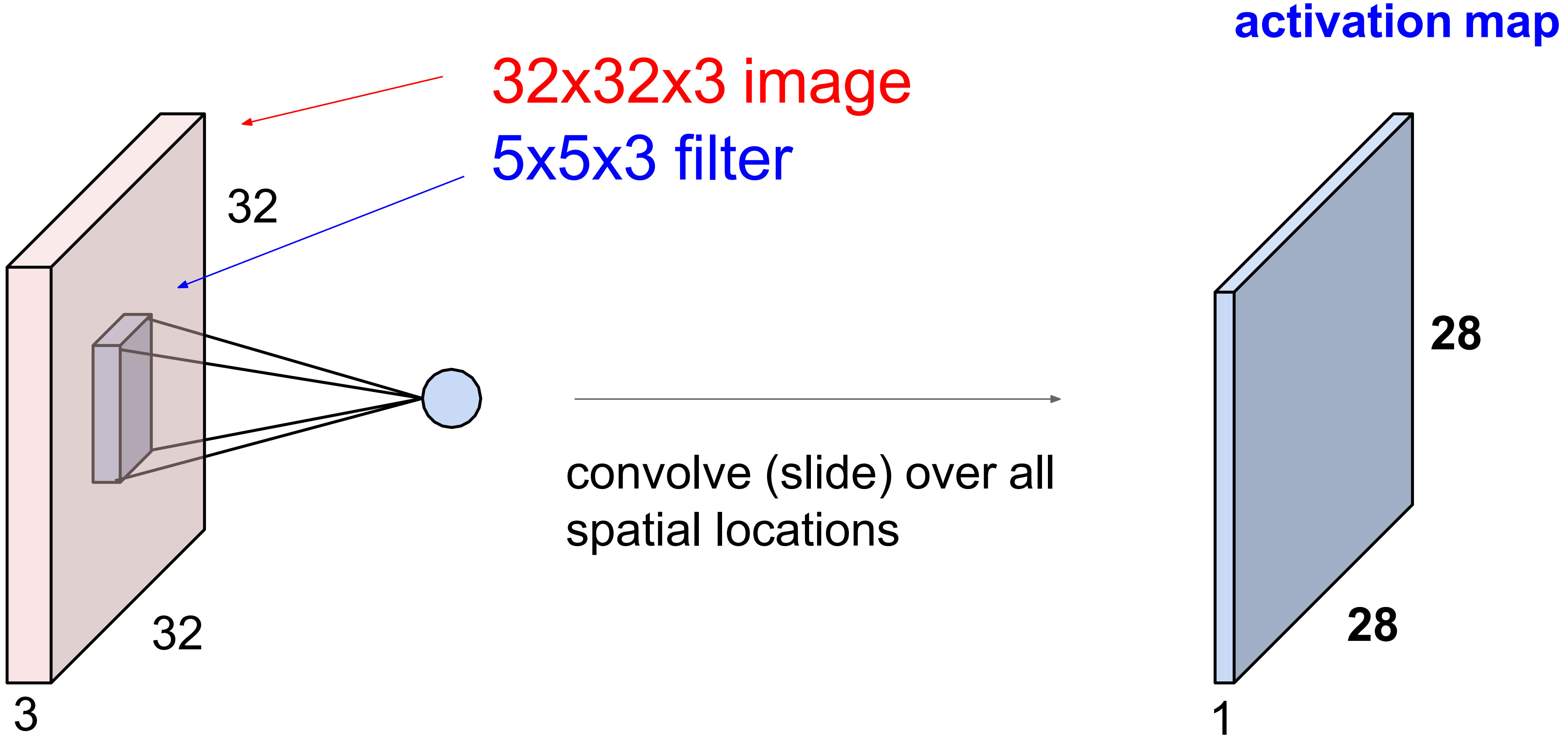
We call the layer convolutional because it is related to convolution of two signals:

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

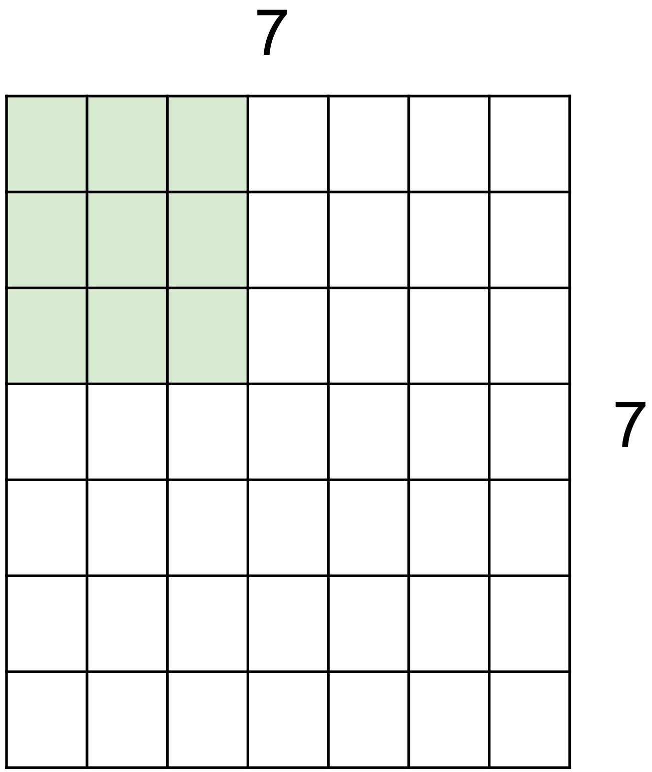
elementwise multiplication and sum of a filter and the signal (image) preview:


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:

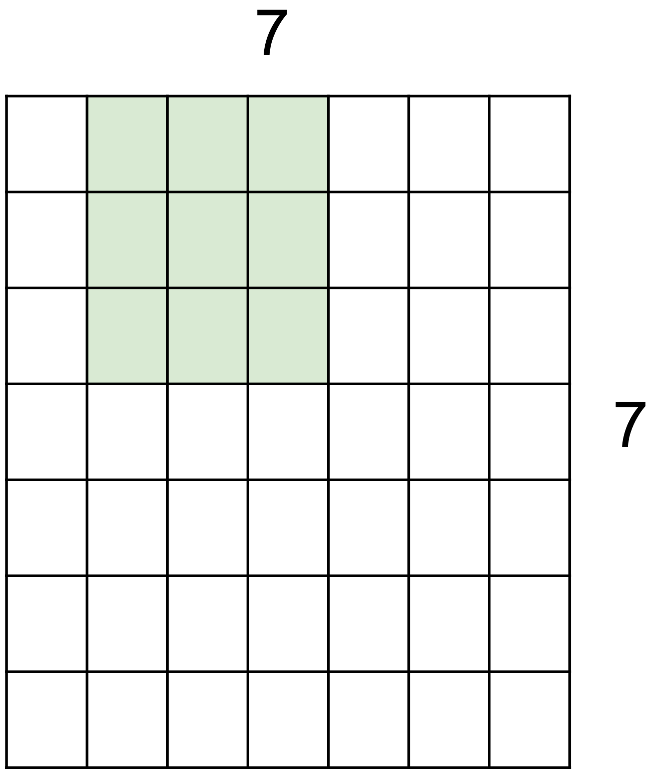


A closer look at spatial dimensions:



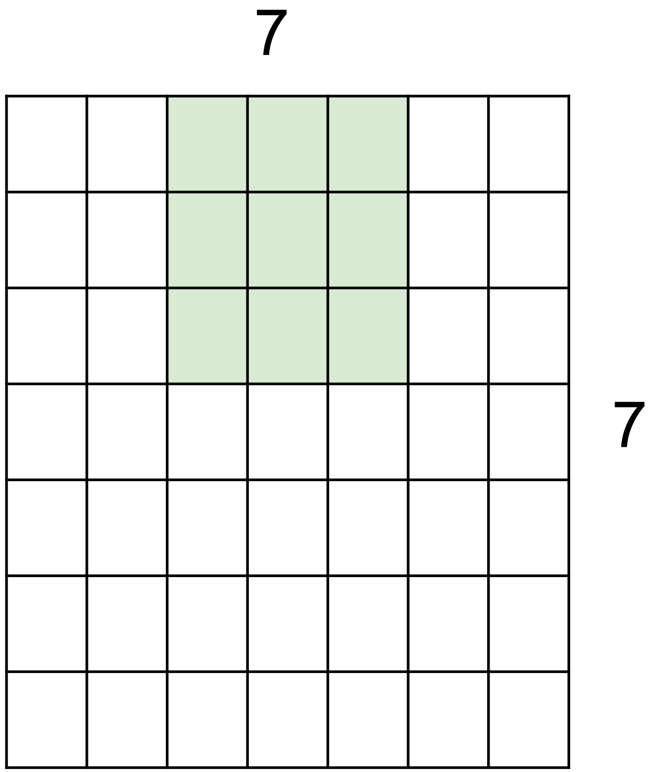
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



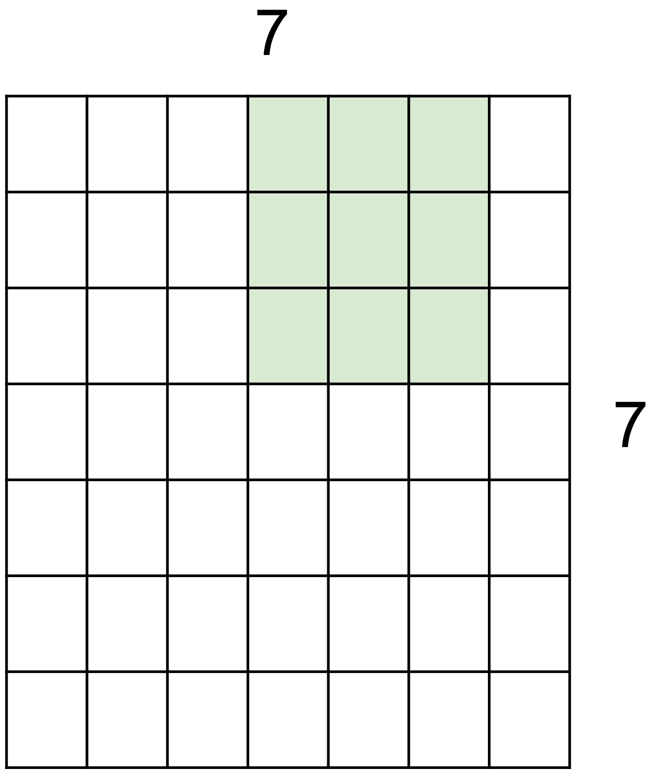
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



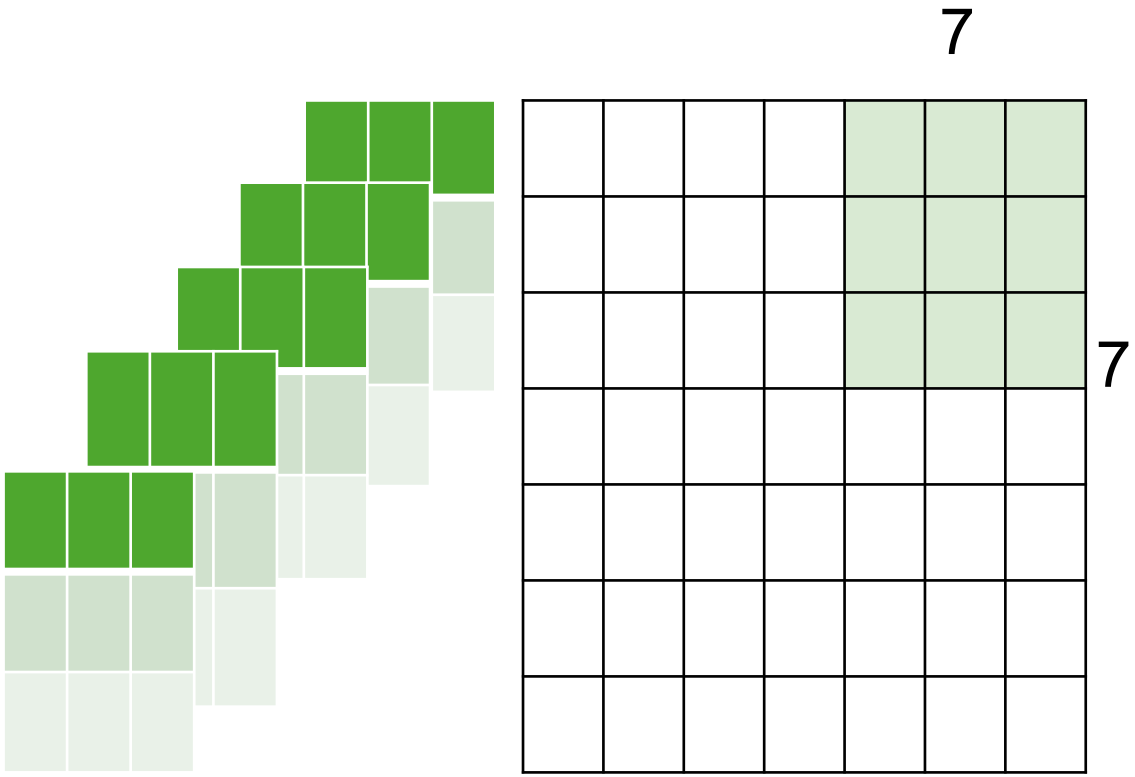
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

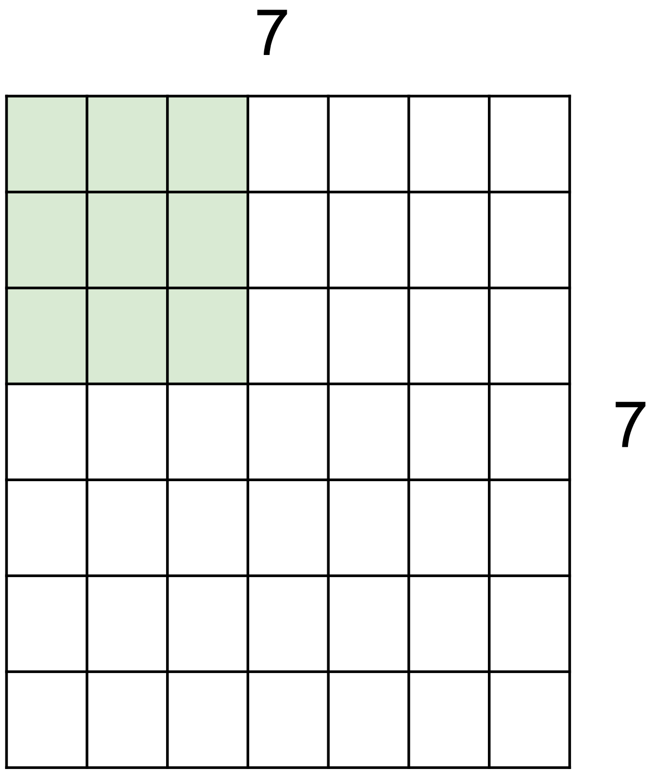
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

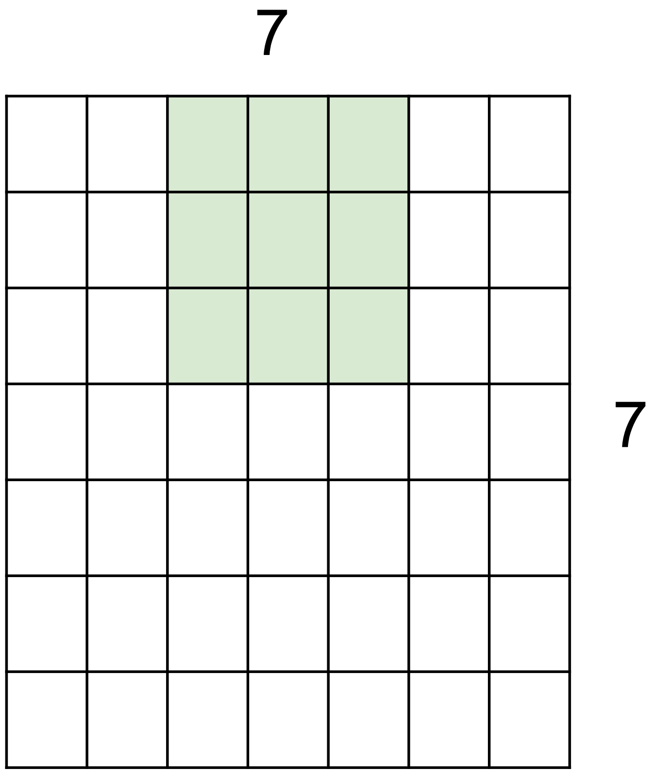
⇒ **5x5 output**
⇒ **With stride =1, takes 5 times sliding to traverse the width**
⇒ **3 x 3 filter will need 5 times sliding to cover the height with stride = 1**

A closer look at spatial dimensions:



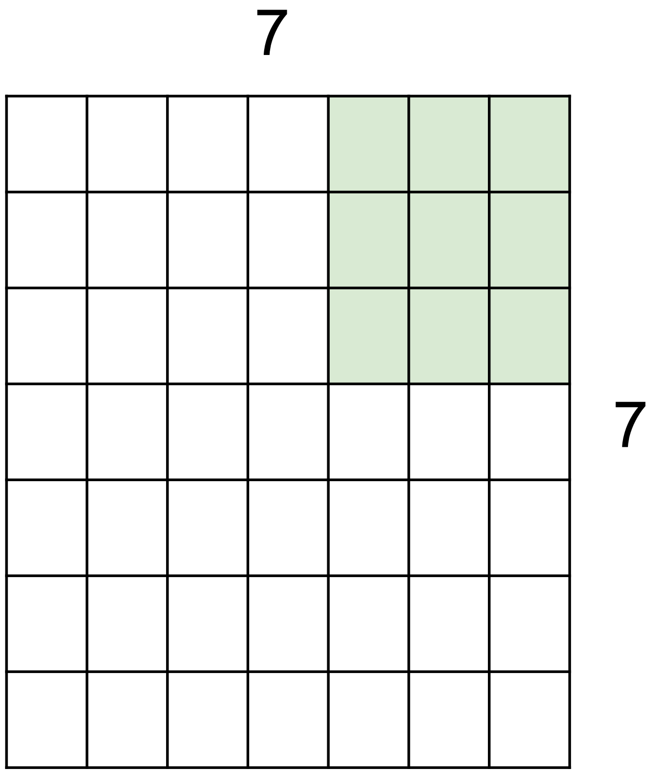
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



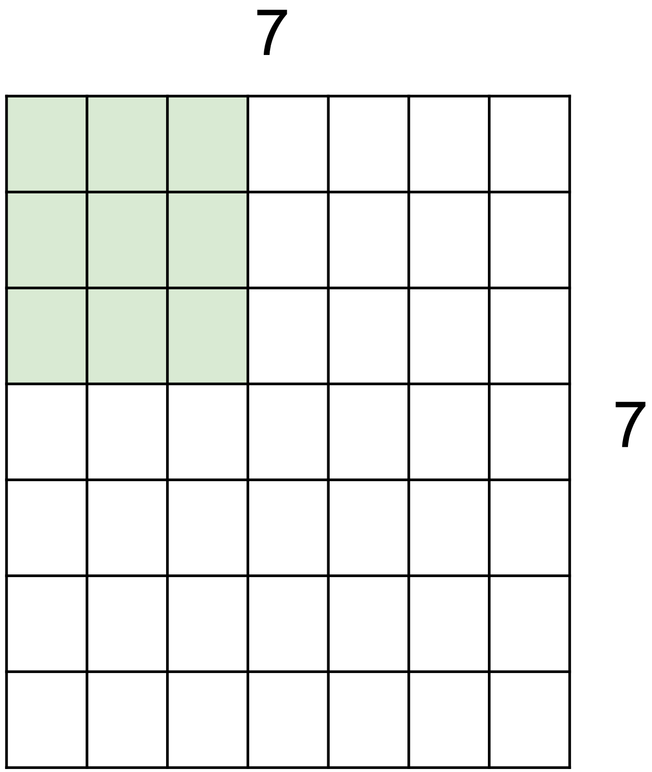
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



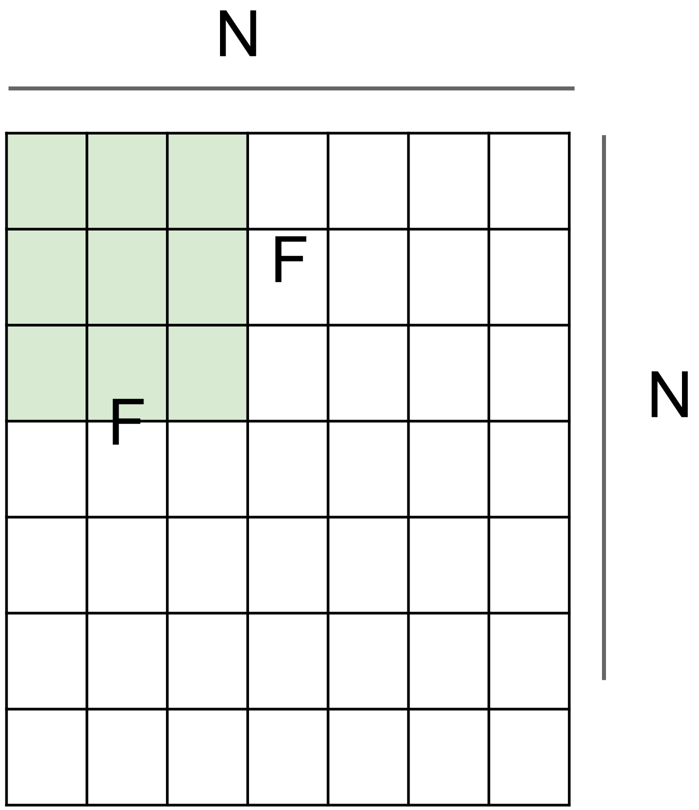
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

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



Assume input image height (== width) = N
Applied stride's height(== width) = F

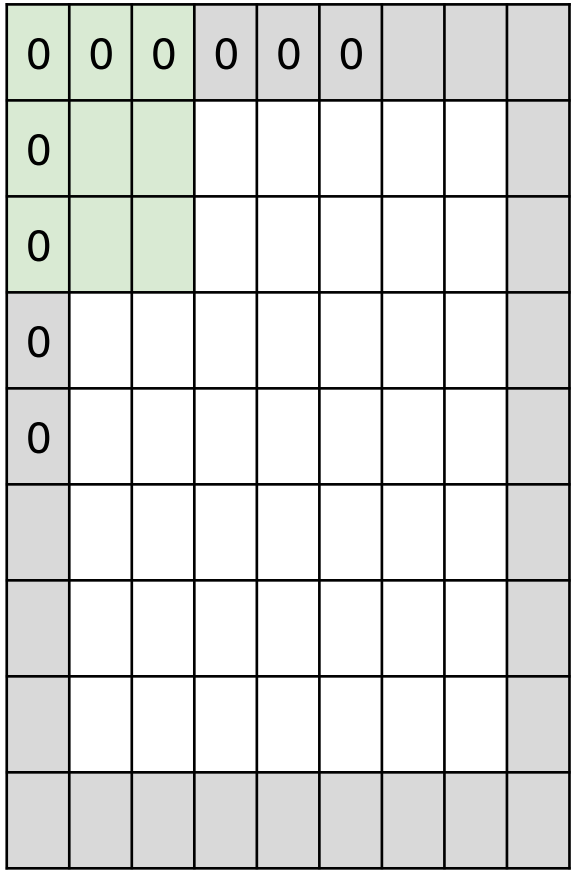
Output width (==height)
(N - F) / stride + 1

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 :\

Padding the Input Before Convolution

- Solution
  - Add extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image.
  - Set values of extra pixels with zeros

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

CNN Vocabulary

- Padding
- Stride
- Pooling

Stride

- Number of rows and columns traversed per slide as the *stride*
- Previous example
- Stride = 1 for both height and width

Stride Limitations

1. The output feature map is smaller (3x3) than the input image (5x5).
    1. Applying more convolutional layers → the spatial dimensions decreases→ loss of information.

2. The pixels at the borders of receives fewer convolutional operations compared to the center pixels.
    1. Can impact model's performance if important features are present at the edges.

Stride, Padding, and the Output Volume

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.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Stride, Padding, and the Output Volume

Common settings:

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

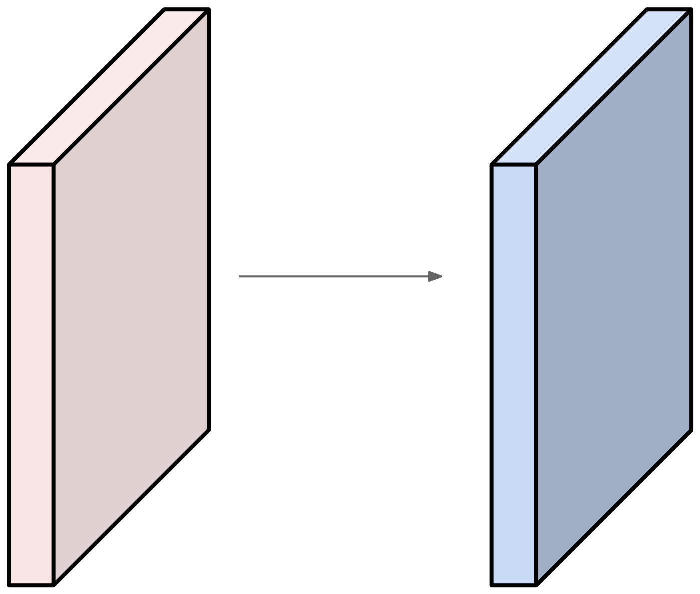
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

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.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

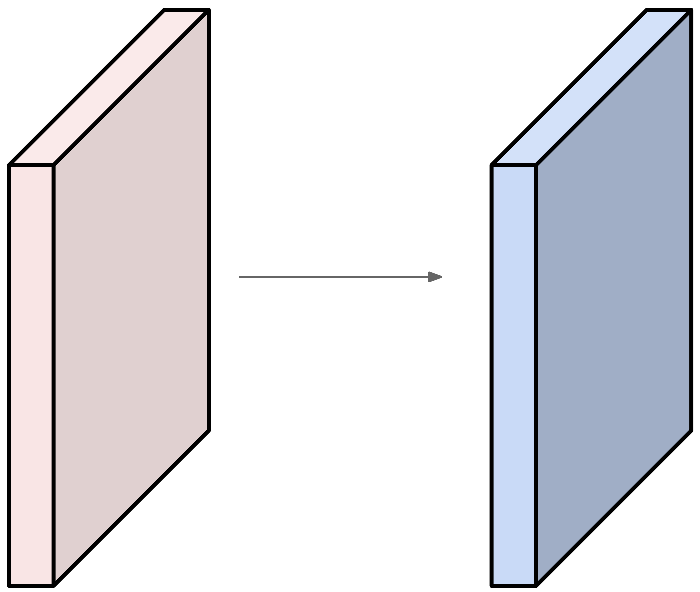
Examples time:



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

Output volume size: ?

Examples time:



Input volume: **32x32x3**

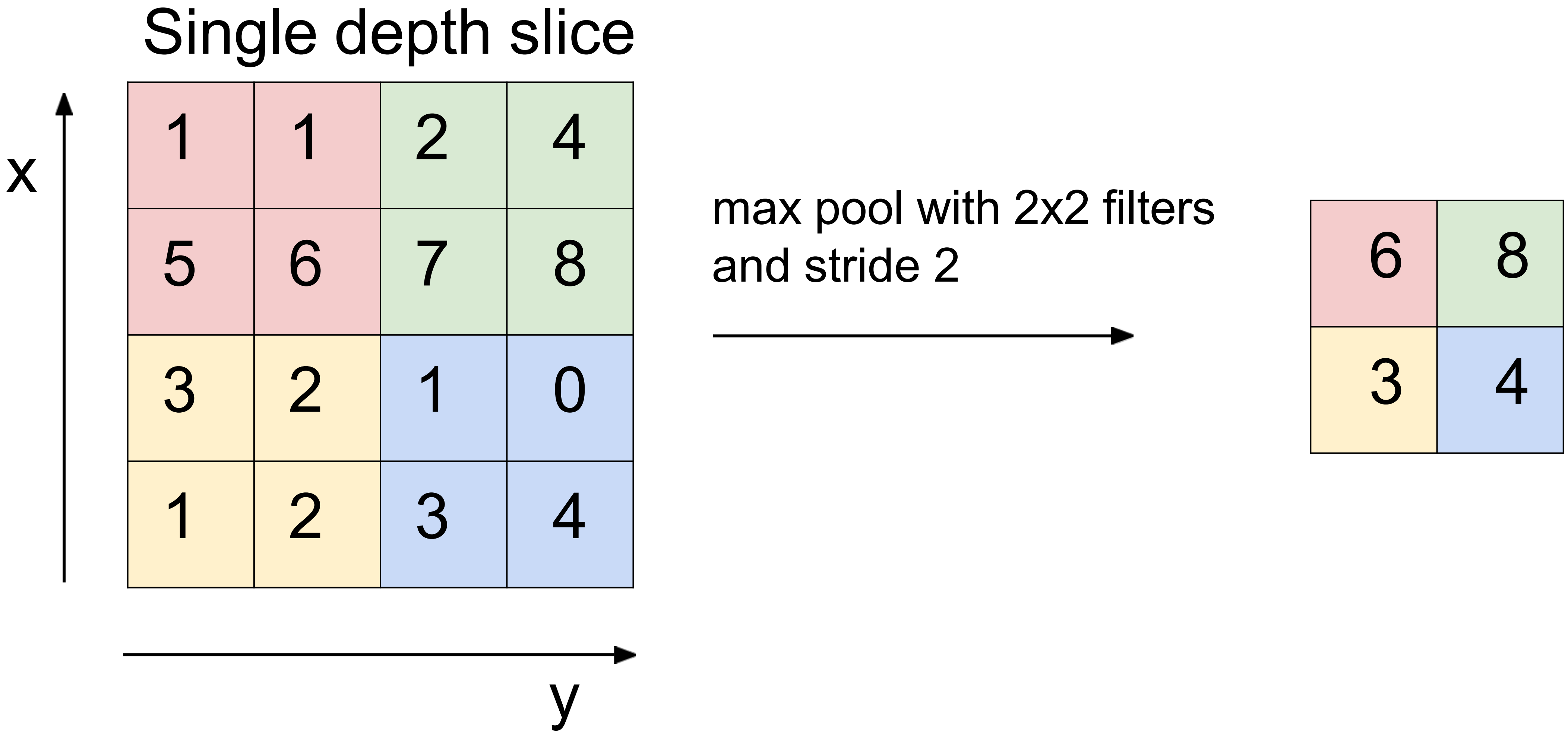
10 5x5 filters with stride 1, pad 2

Output volume size:

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

32x32x10

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

Single depth slice

x

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

y

max pool with 2x2 filters
and stride 2

6	8
3	4

- No learnable parameters
- Introduces spatial invariance

Pooling layer: Summary

Let's assume input is W_1 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

Pooling Continued

The three types of pooling operations are:

1. Max pooling: The maximum pixel value of the batch is selected.
2. Min pooling: The minimum pixel value of the batch is selected.
3. Average pooling: The average value of all the pixels in the batch is selected.

CNN Demo

https://github.com/effat/MLP-Demo/blob/main/CNN_Demo.ipynb