

# Indian Institute of Information Technology, Allahabad



## Convolutional Neural Networks

By

**Dr. Satish Kumar Singh & Dr. Shiv Ram Dubey**

Computer Vision and Biometrics Lab

Department of Information Technology

Indian Institute of Information Technology, Allahabad



# TEAM

**Computer Vision and Biometrics Lab (CVBL)**

**Department of Information Technology**

**Indian Institute of Information Technology Allahabad**

## **Course Instructors**

Dr. Satish Kumar Singh, Associate Professor, IIIT Allahabad (Email: [sk.singh@iiita.ac.in](mailto:sk.singh@iiita.ac.in))

Dr. Shiv Ram Dubey, Assistant Professor, IIIT Allahabad (Email: [srdubey@iiita.ac.in](mailto:srdubey@iiita.ac.in))



# DISCLAIMER

**The content (text, image, and graphics) used in this slide are adopted from many sources for Academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.**



# Outline

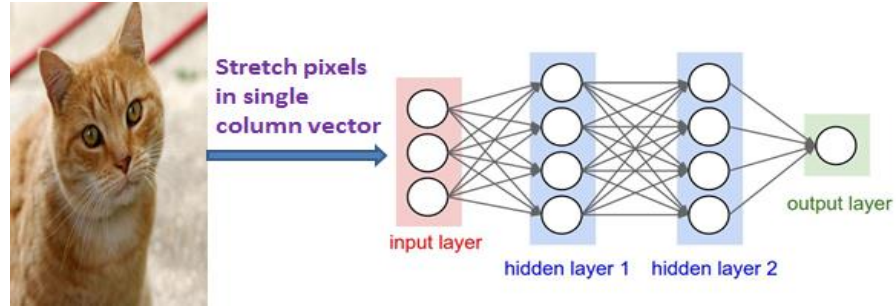
---

- Building blocks
  - Convolutional layers and backprop rules
  - Pooling layers and nonlinearities
- Architectures:
  - 2012: AlexNet
  - 2013: ZFNet
  - 2014: VGGNet, GoogLeNet
  - 2015: ResNet
  - 2016: ResNeXt, DenseNet
  - etc.

# This Class

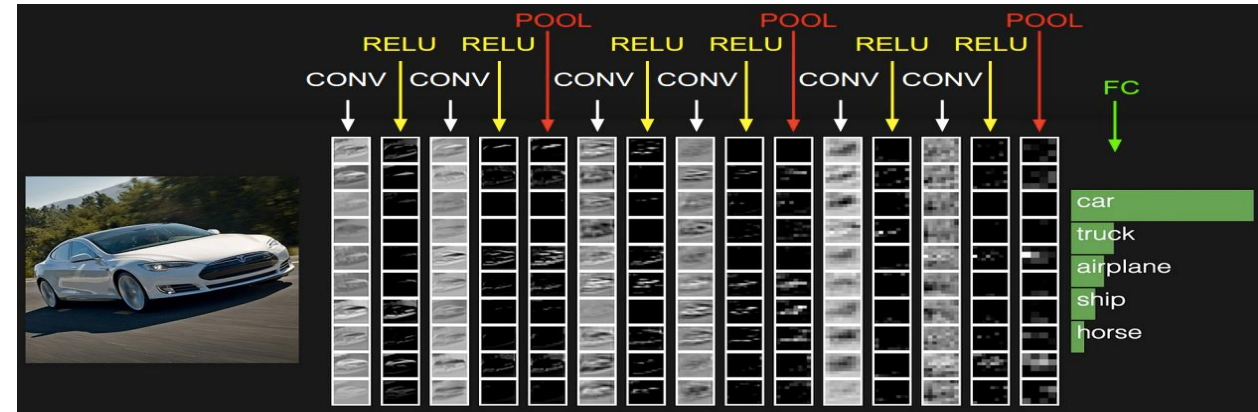
## Neural Network and Image

- Dimensionality
- Local relationship



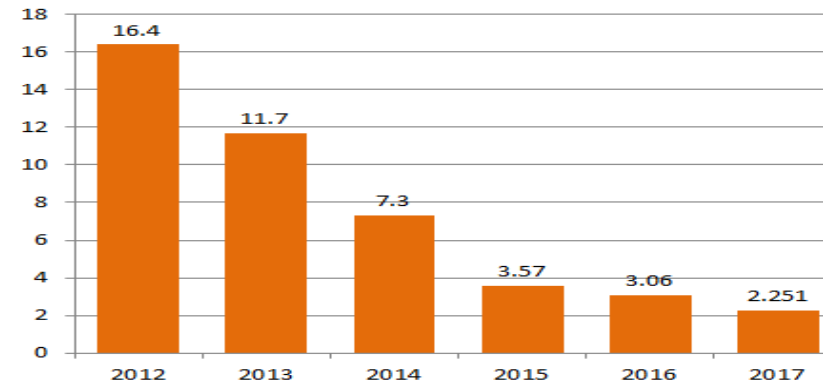
## Convolutional Neural Network (CNN)

- Convolution Layer
- Non-linearity Layer
- Pooling Layer
- Fully Connected Layer
- Classification Layer



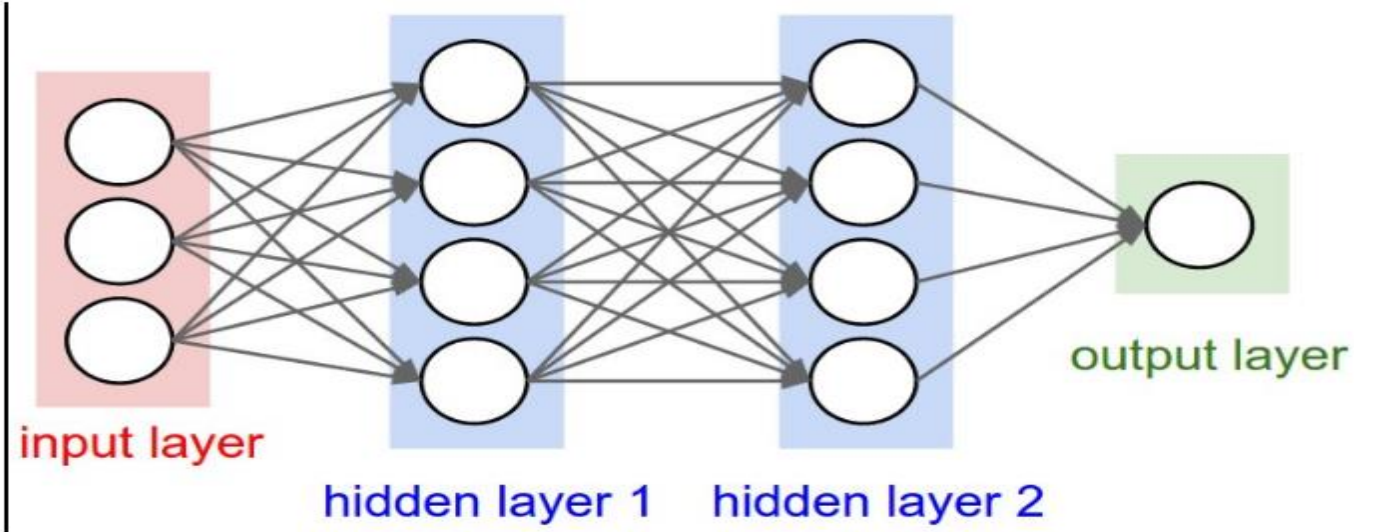
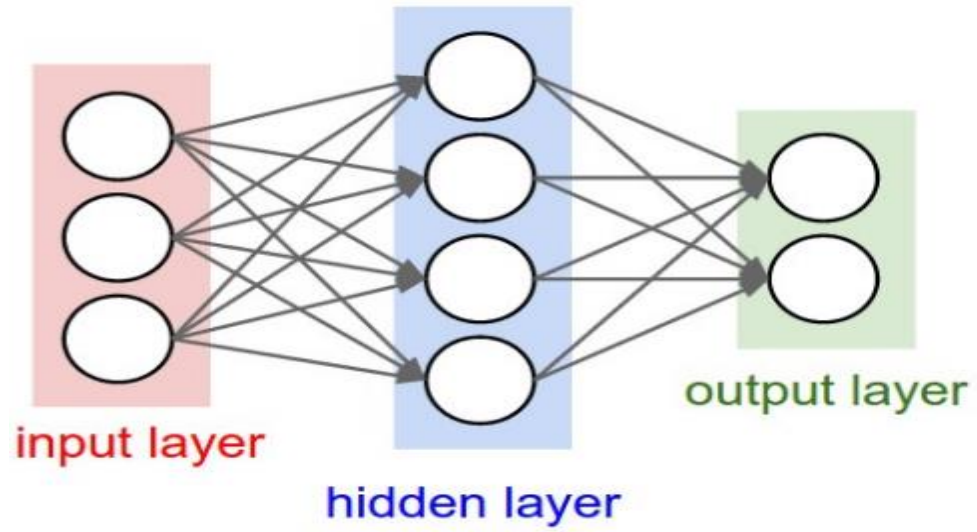
## ImageNet Challenge

- Progress
- Human Level Performance



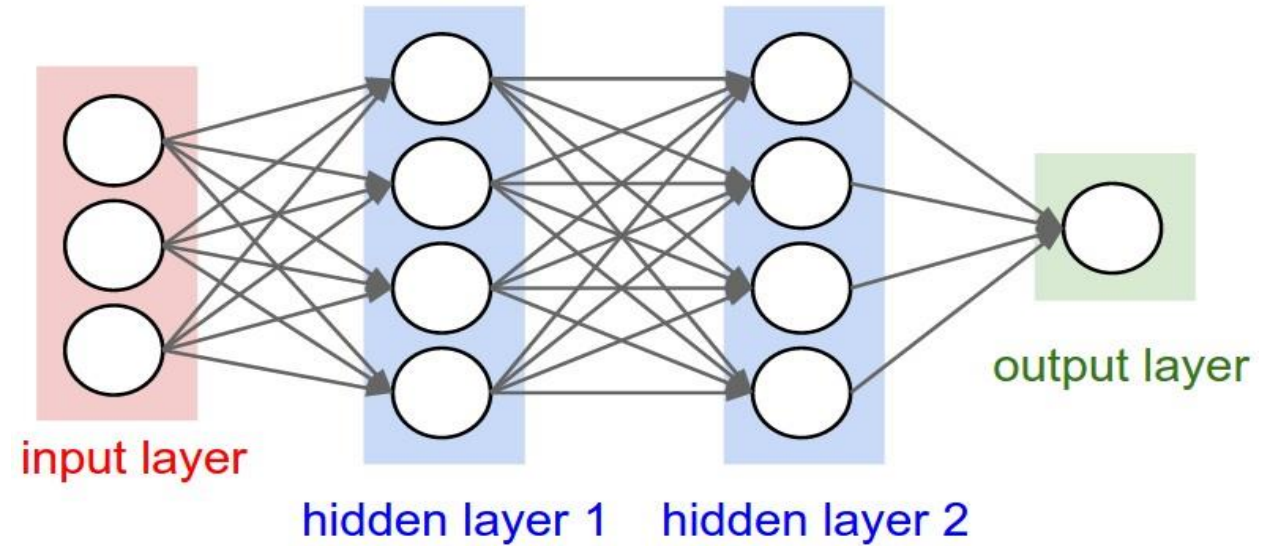
# Neural Networks

---



# Multi-layer Neural Network & Image

---



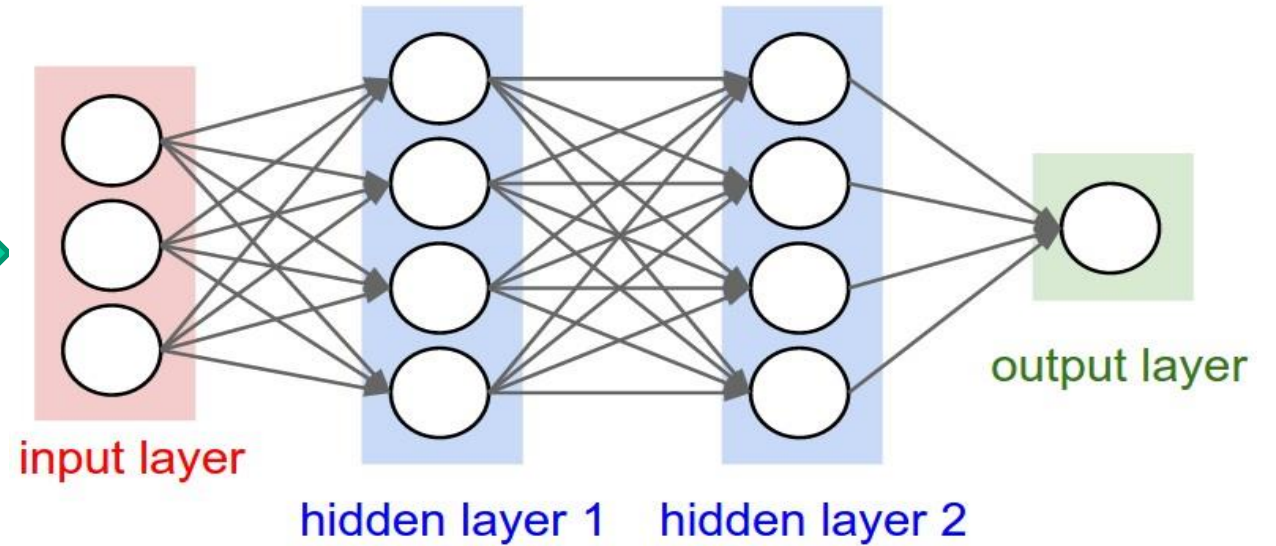
**How to apply NN over Image?**



# Multi-layer Neural Network & Image



Stretch pixels in  
single column  
vector

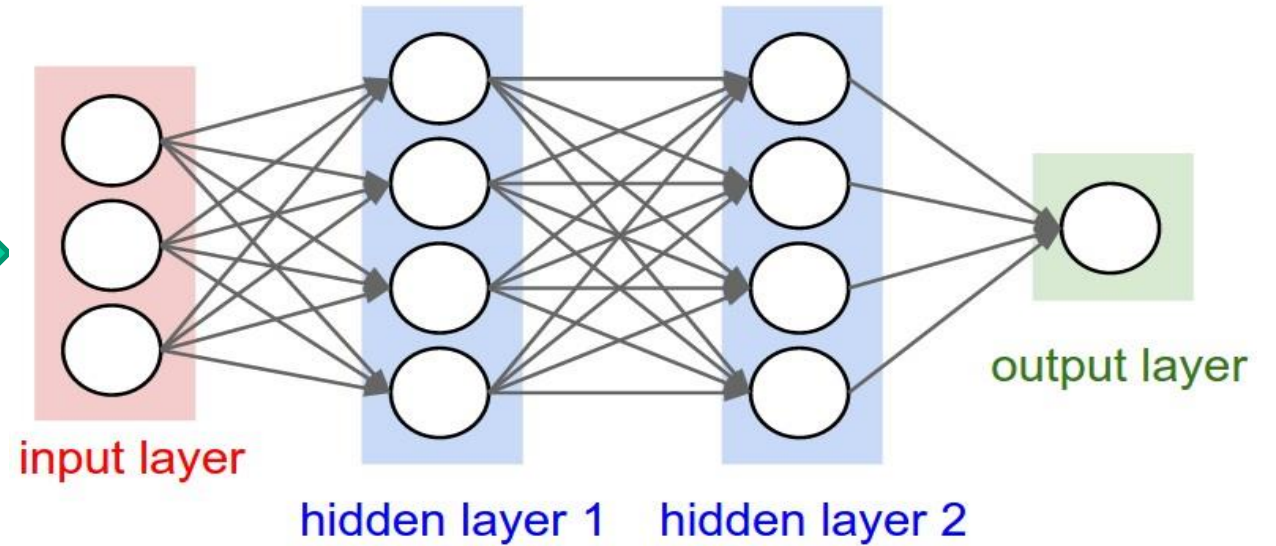




# Multi-layer Neural Network & Image

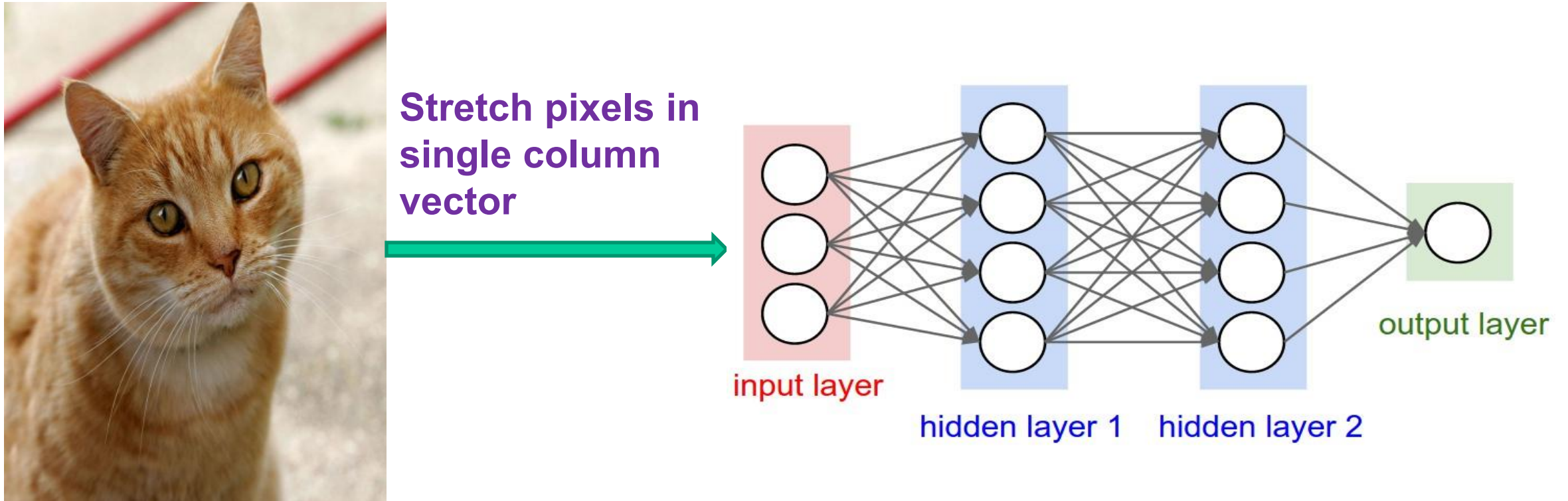


Stretch pixels in  
single column  
vector



**Problems ?**

# Multi-layer Neural Network & Image

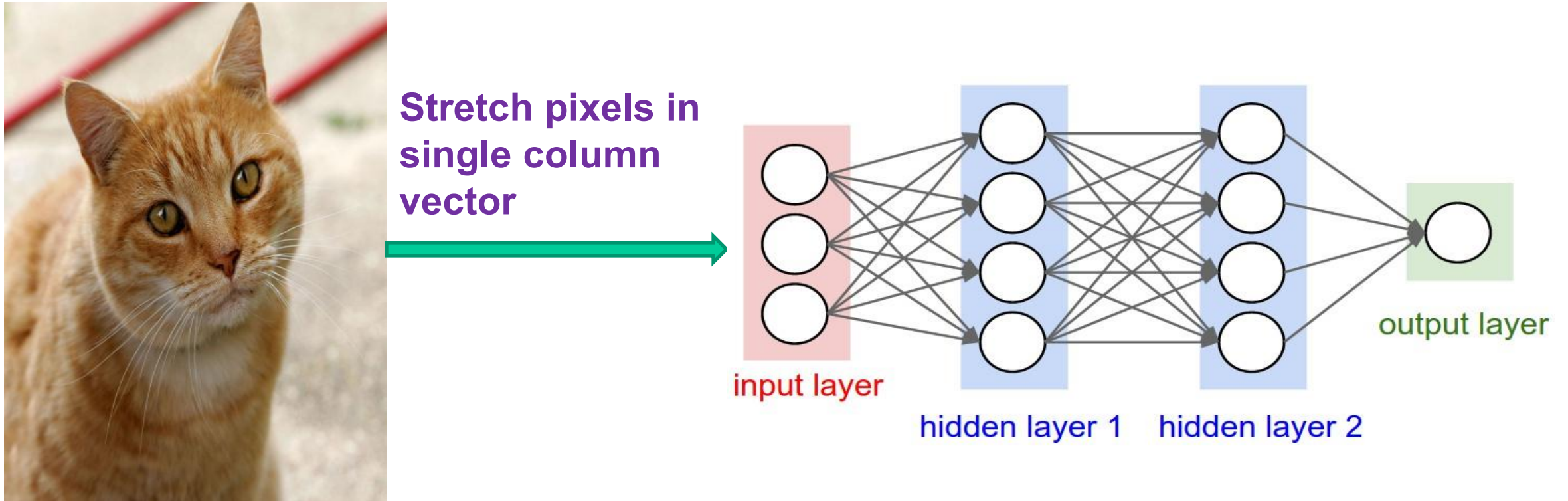


## Problems:

High dimensionality

Local relationship

# Multi-layer Neural Network & Image



**Problems:**

High dimensionality

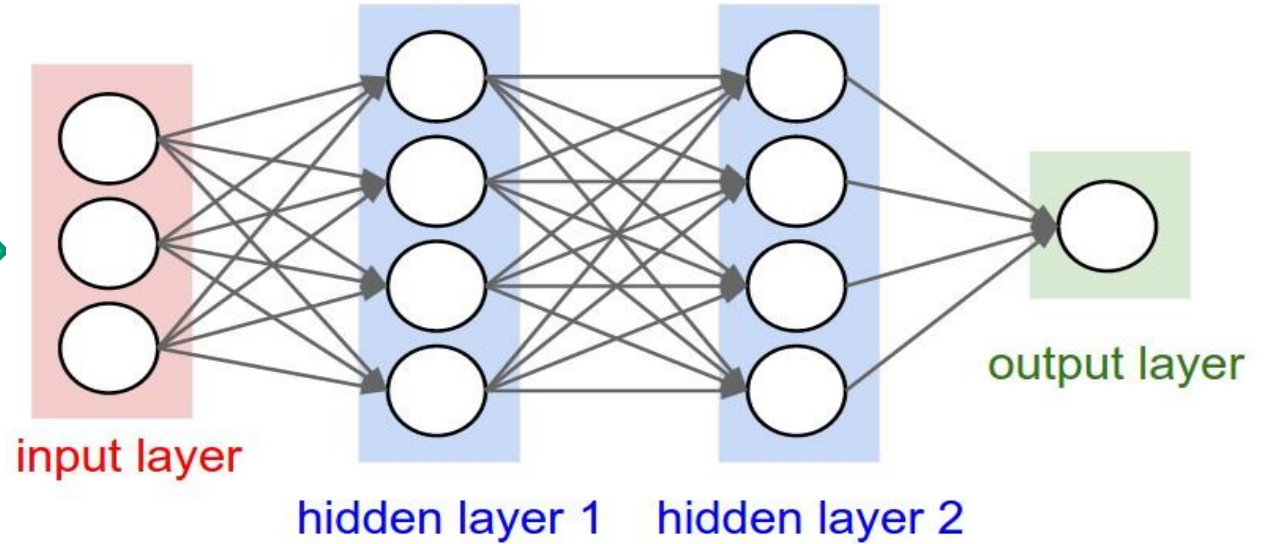
Local relationship

**Solution ?**

# Multi-layer Neural Network & Image



Stretch pixels in  
single column  
vector



## Problems:

High dimensionality

Local relationship

## Solution:

Convolutional Neural Network

# Convolutional Neural Networks

---

Also known as

CNN,

ConvNet,

DCN

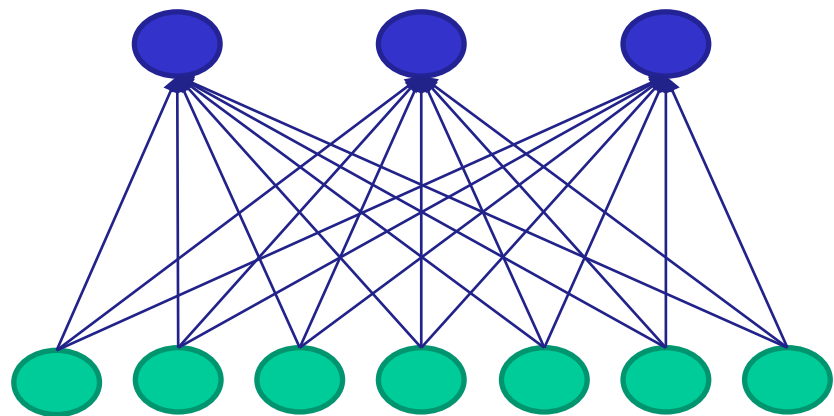
CNN = a multi-layer neural network with

1. Local connectivity
2. Weight sharing



# CNN: Local Connectivity

---



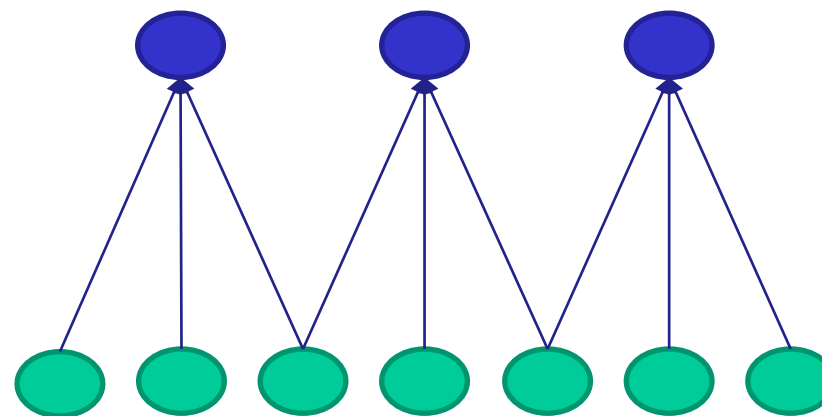
**Global connectivity**

# input units (neurons): 7

# hidden units: 3

Hidden layer

Input layer

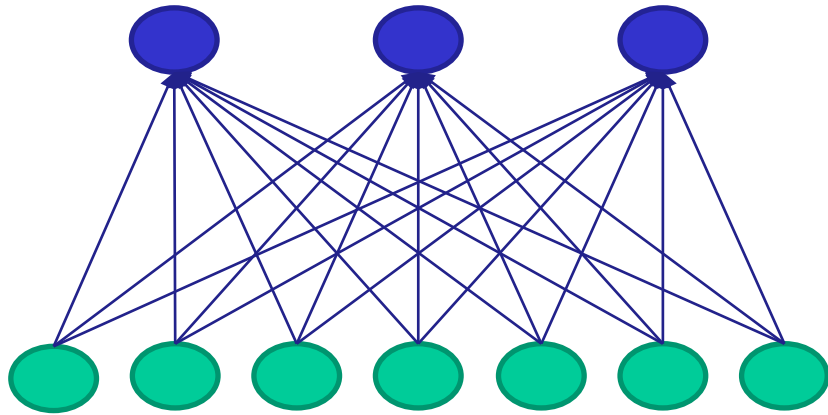


**Local connectivity**



# CNN: Local Connectivity

---



Hidden layer

Input layer

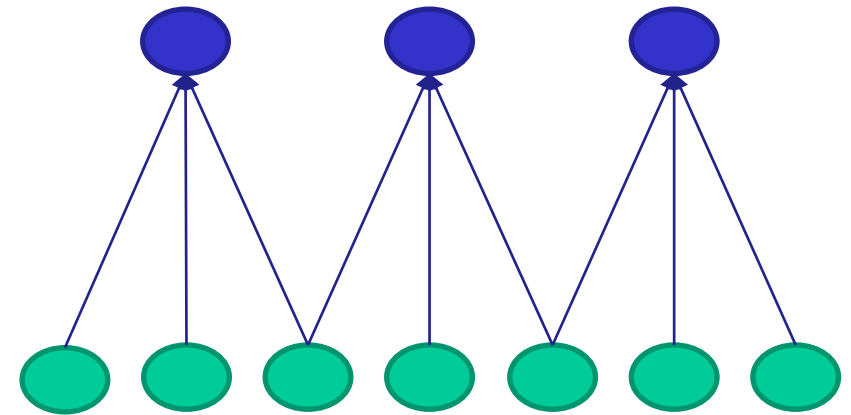
## Global connectivity

# input units (neurons): 7

# hidden units: 3

## Number of parameters

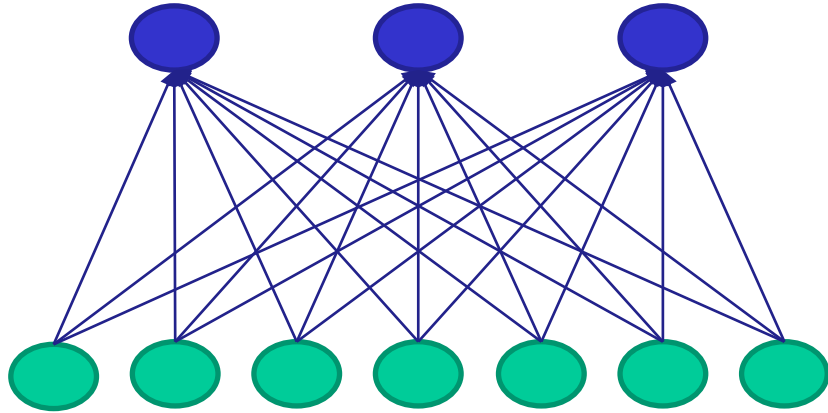
- Global connectivity: ?
- Local connectivity: ?



## Local connectivity

# CNN: Local Connectivity

---



Hidden layer

Input layer

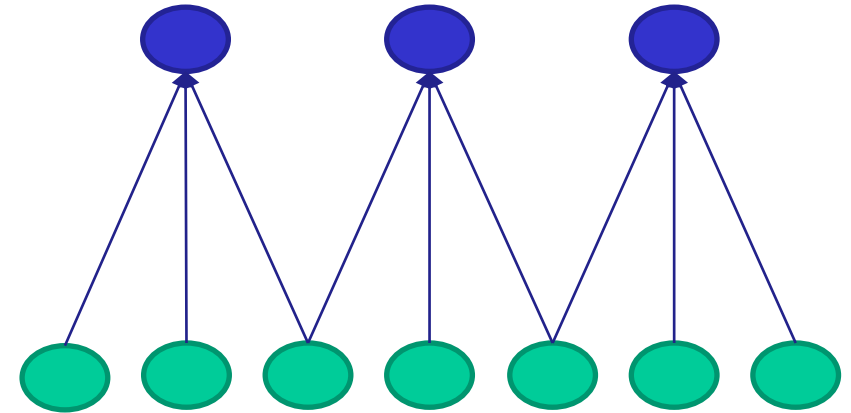
## Global connectivity

# input units (neurons): 7

# hidden units: 3

## Number of parameters

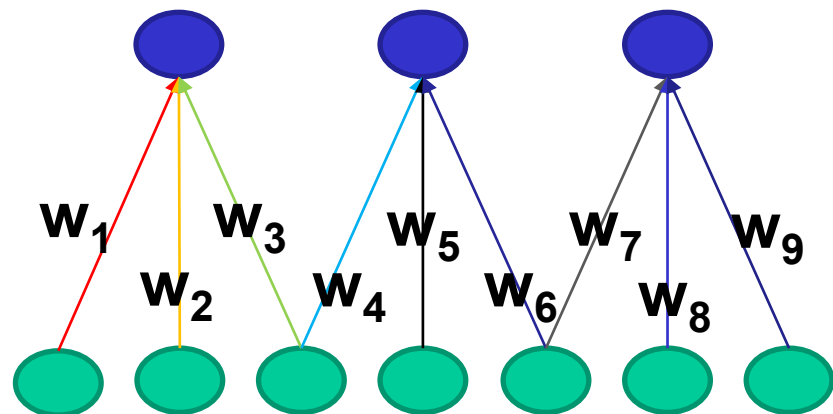
- Global connectivity:  $3 \times 7 = 21$
- Local connectivity:  $3 \times 3 = 9$



## Local connectivity

# CNN: Weight Sharing

---

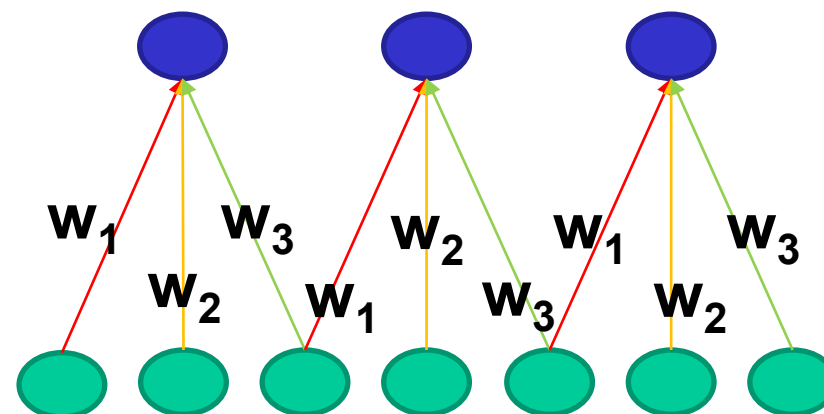


Without weight sharing

- # input units (neurons): 7
- # hidden units: 3

Hidden layer

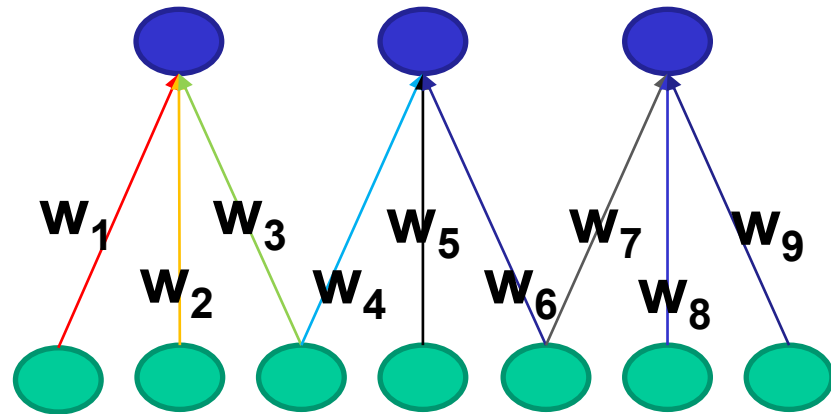
Input layer



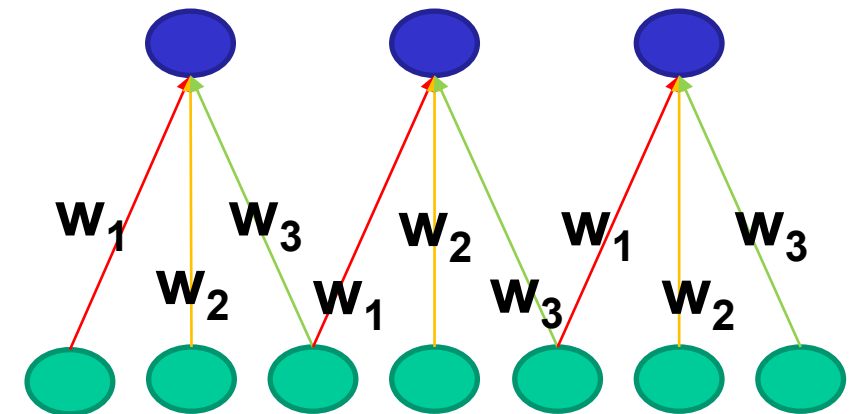
With weight sharing

# CNN: Weight Sharing

---



Without weight sharing

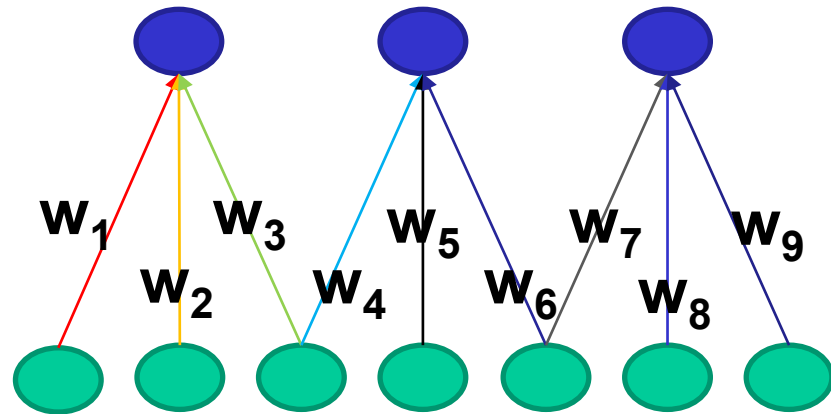


With weight sharing

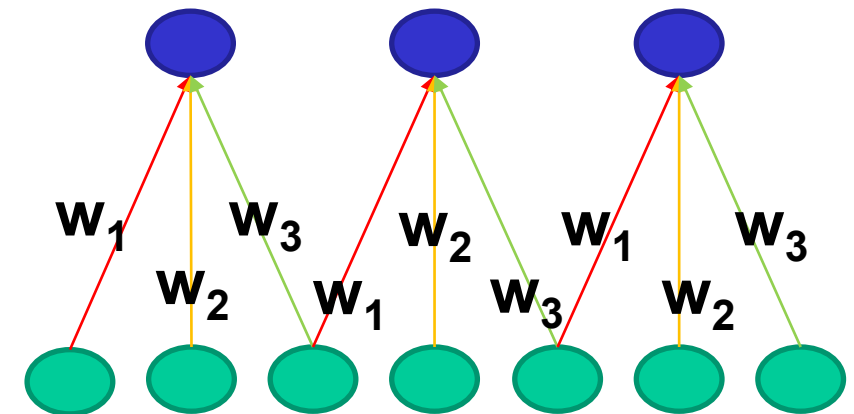
- # input units (neurons): 7
- # hidden units: 3
- **Number of parameters**
  - Without weight sharing: ?
  - With weight sharing : ?

# CNN: Weight Sharing

---



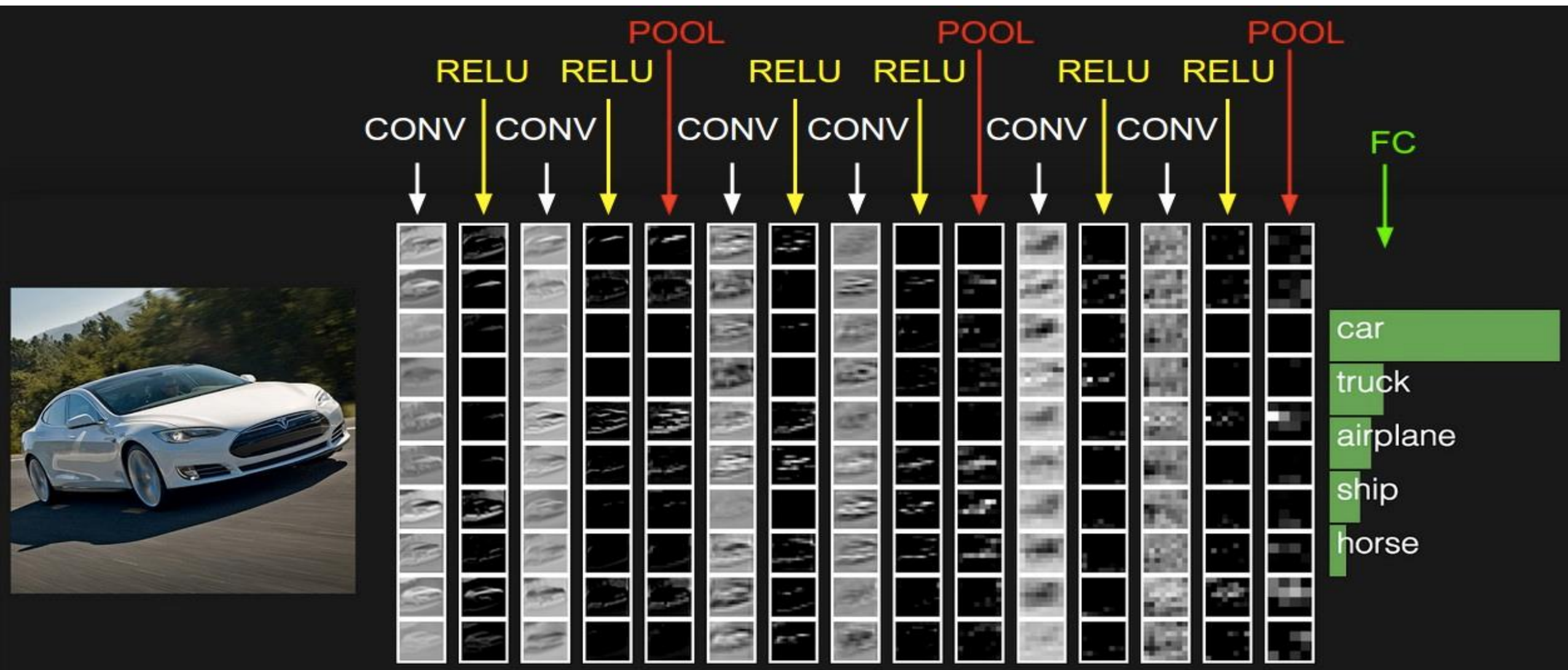
Without weight sharing



With weight sharing

- # input units (neurons): 7
- # hidden units: 3
- **Number of parameters**
  - Without weight sharing:  $3 \times 3 = 9$
  - With weight sharing :  $3 \times 1 = 3$

# Convolutional Neural Networks





# Layers used to build ConvNets

---

Input Layer (Input image)

Convolutional Layer

Non-linearity Layer (such as Sigmoid, Tanh, ReLU, PReLU, ELU, Swish, etc.)

Pooling Layer (such as Max Pooling, Average Pooling, etc.)

Fully-Connected Layer

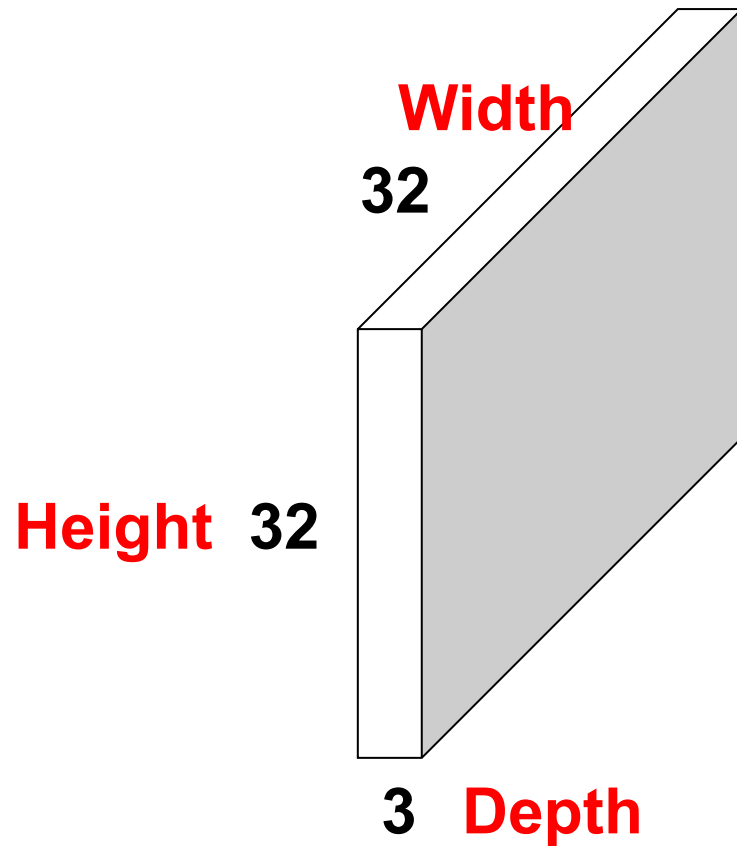
Classification Layer (Softmax, etc.)

# Convolutional Layer

---

**32×32×3 Image**

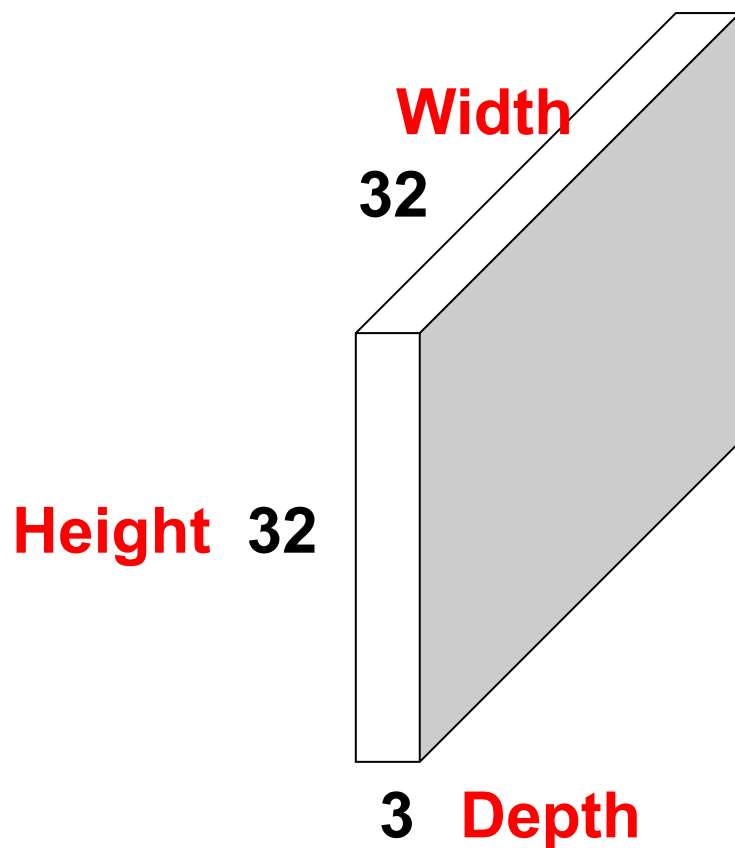
-> preserve spatial structure



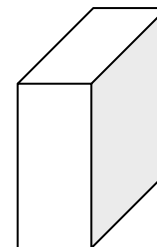
# Convolutional Layer

---

**32×32×3 Image**



**5×5×3 Filter**



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

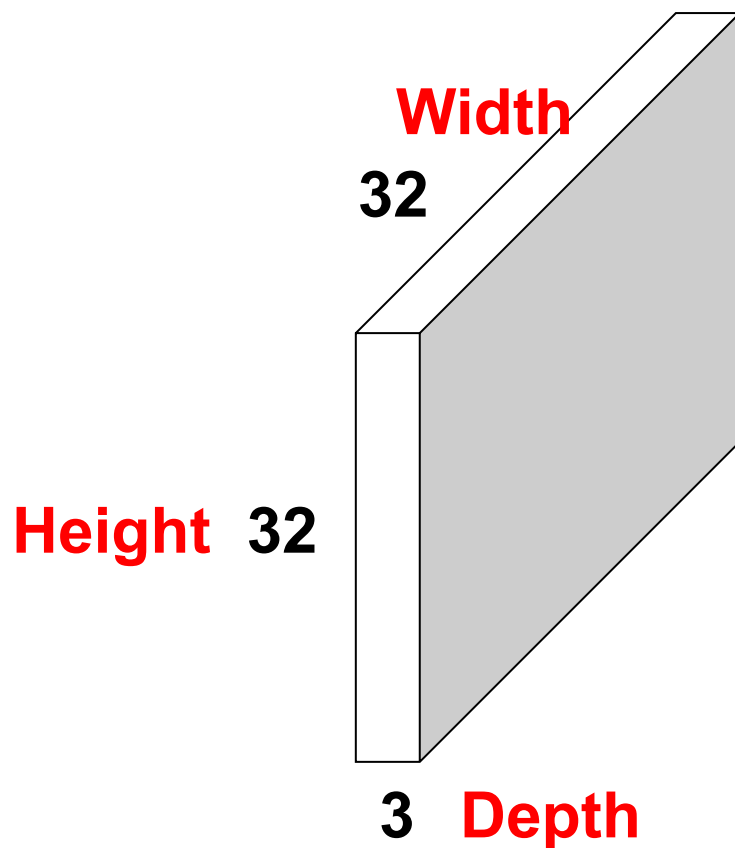
# Convolutional Layer

---

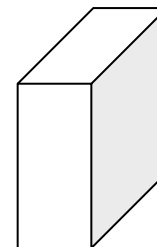
## Handling multiple input channels

Filters always extend the full depth of the input volume

32×32×3 Image



5×5×3 Filter

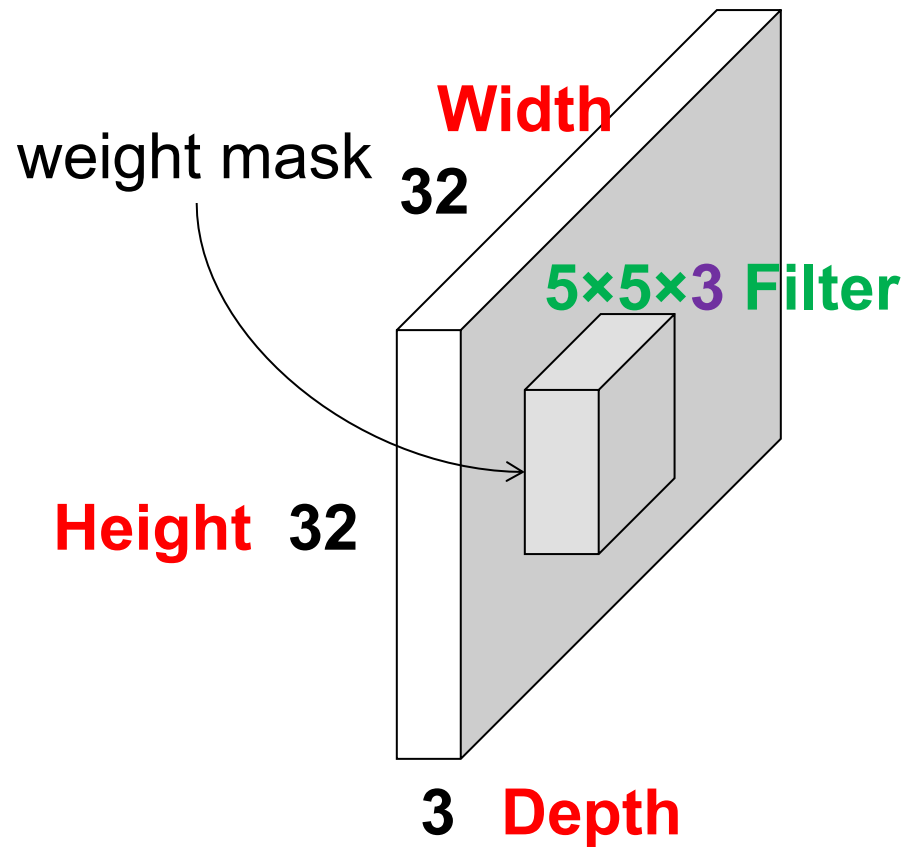


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

# Convolutional Layer

---

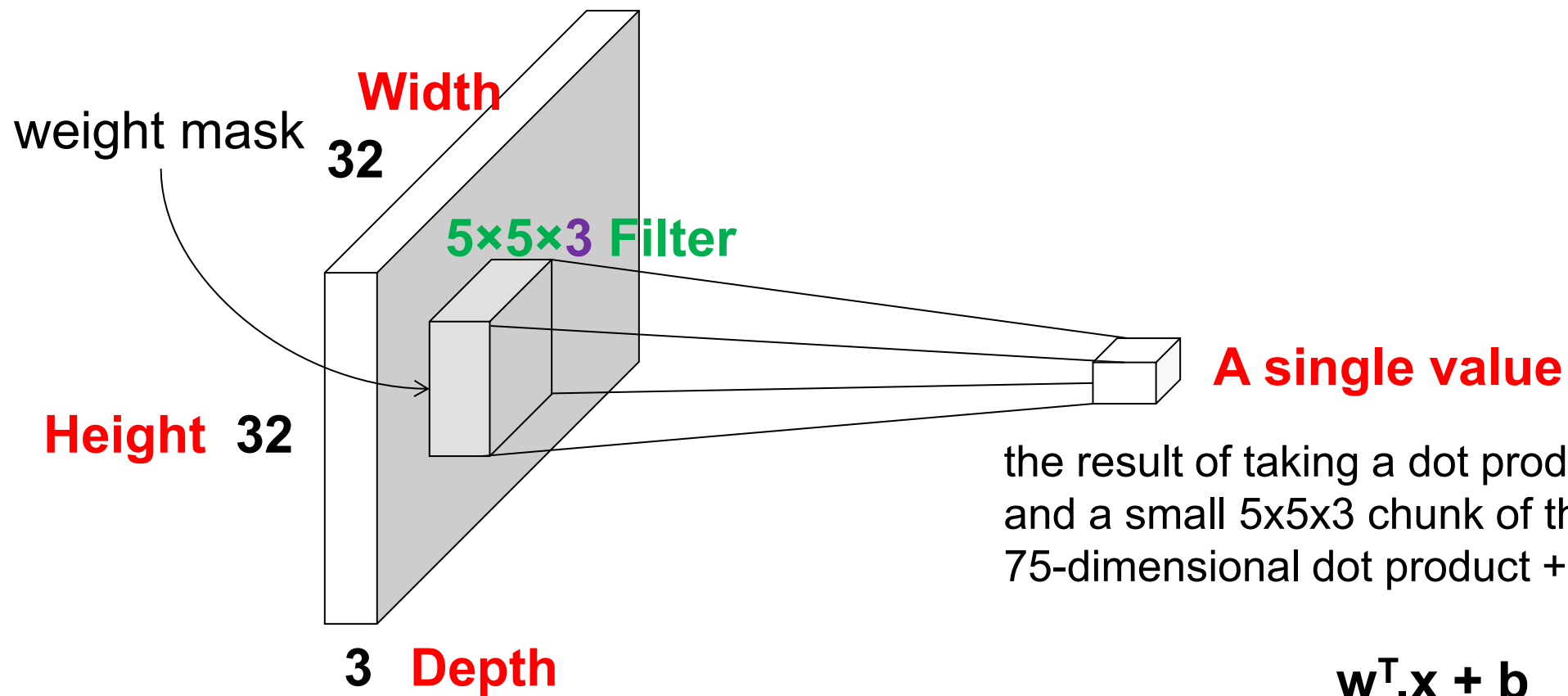
**32×32×3 Image**



# Convolutional Layer

---

**32×32×3 Image**

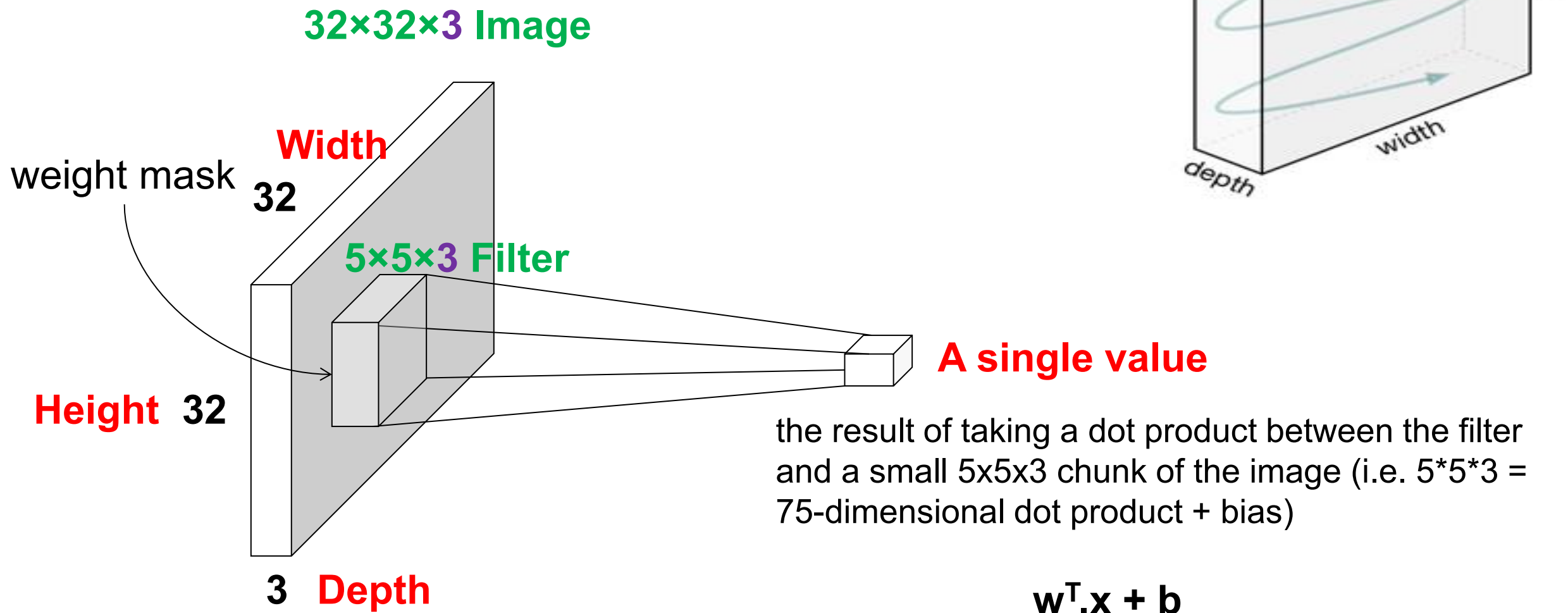


the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5\*5\*3 = 75-dimensional dot product + bias)

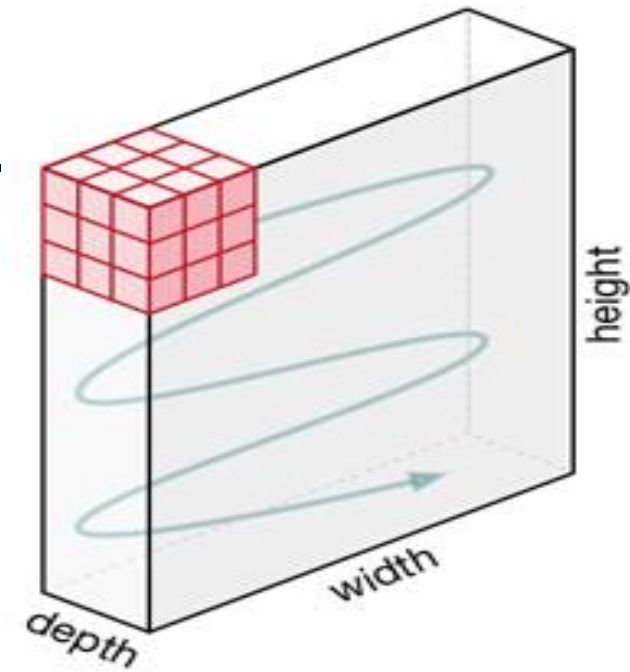
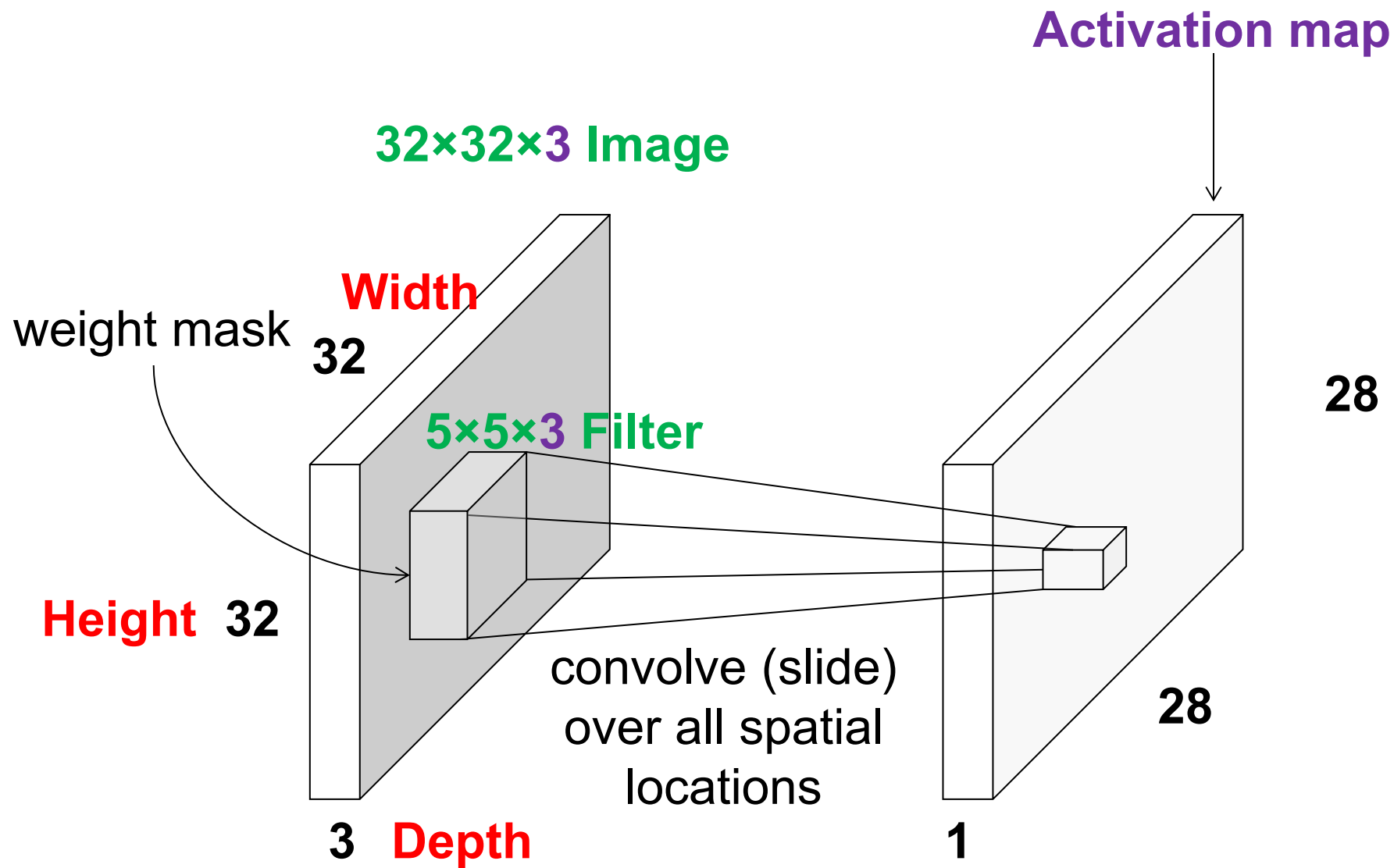
$$w^T \cdot x + b$$



# Convolutional Layer



# Convolutional Layer



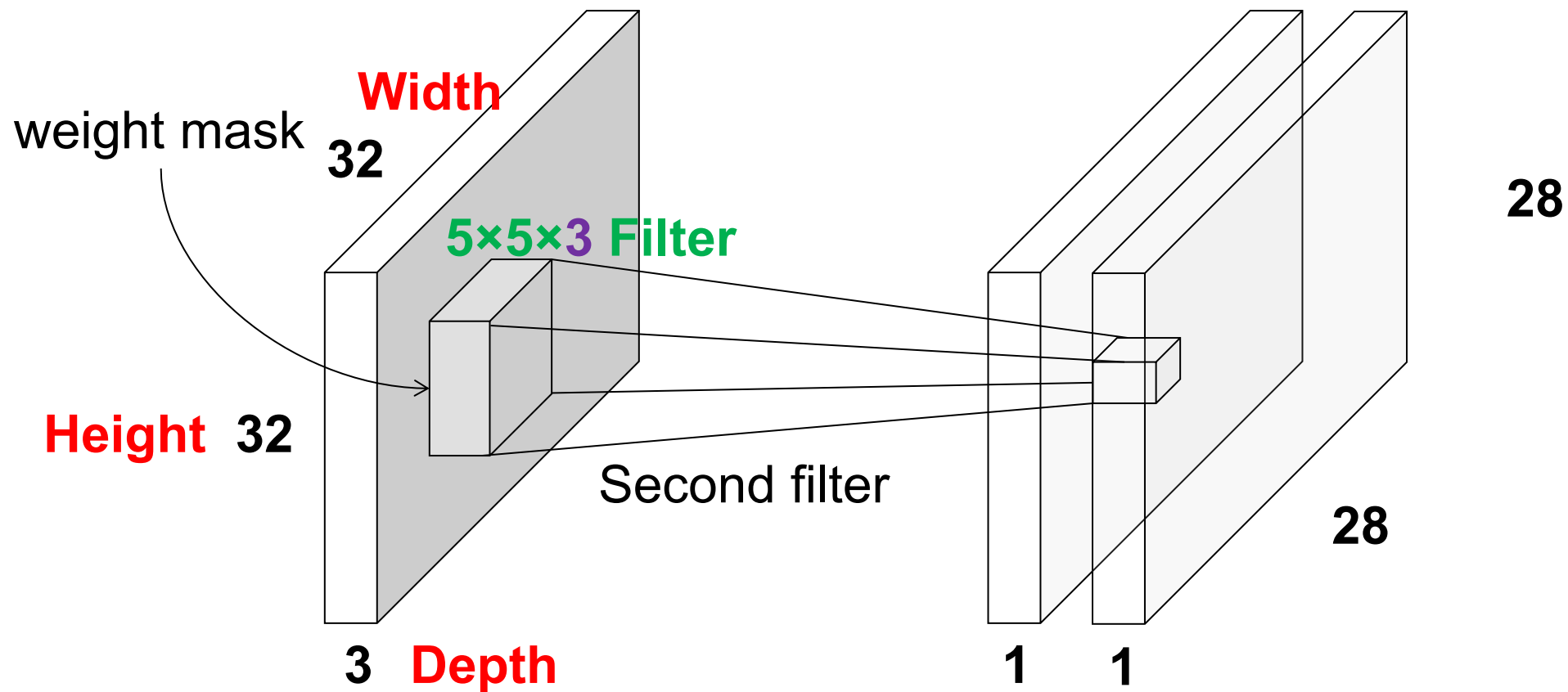
# Convolutional Layer

---

## Handling multiple output maps

**32×32×3 Image**

**Activation maps**



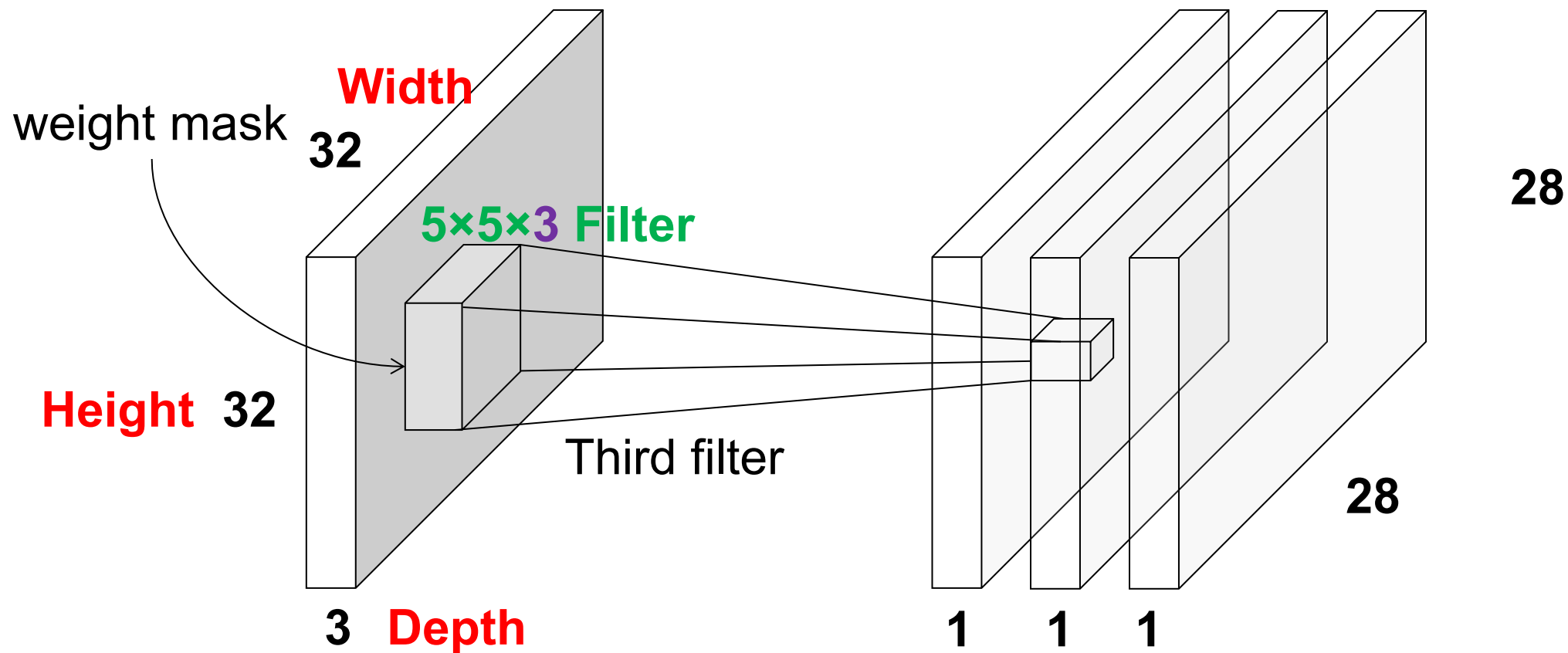
# Convolutional Layer

---

Handling multiple output maps

32×32×3 Image

Activation maps



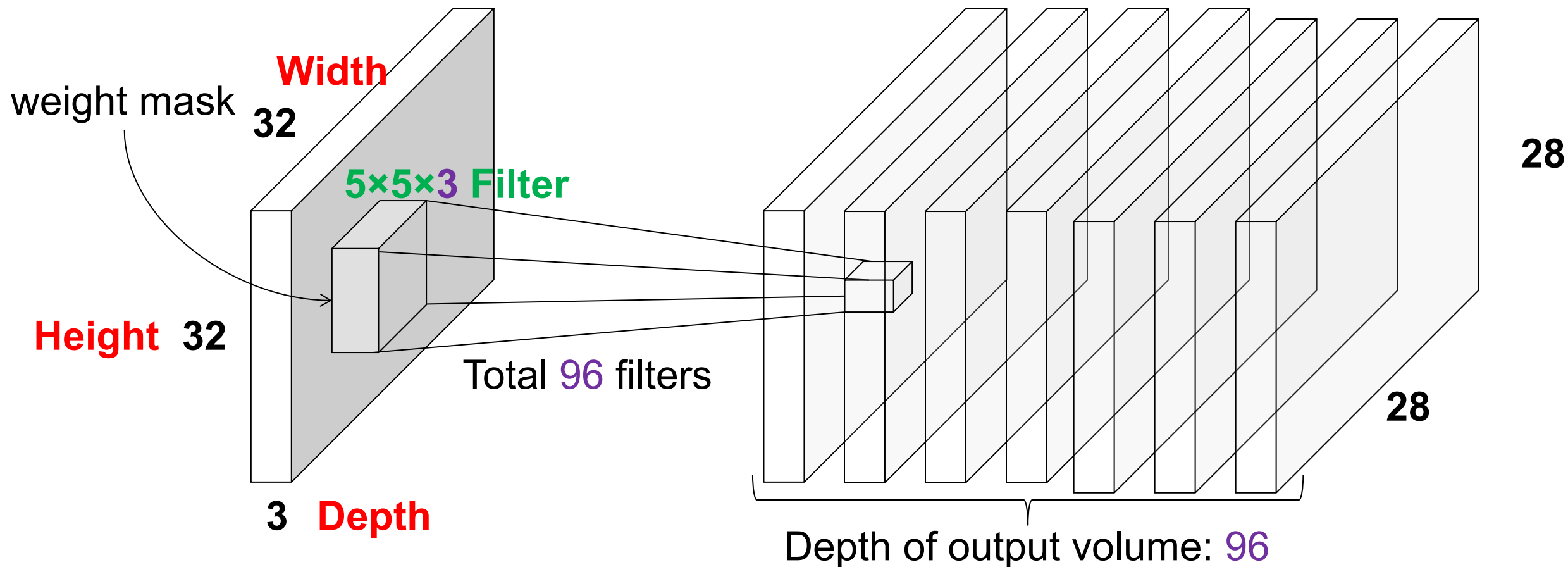
# Convolutional Layer

---

## Handling multiple output maps

32×32×3 Image

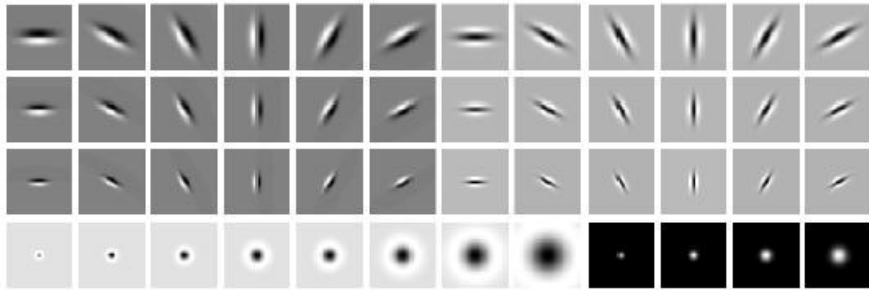
Activation maps



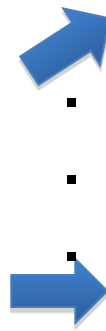
# Convolution and traditional feature extraction

---

bank of  $K$  filters



$K$  feature maps



image

feature map



Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$

0	0	0	0	0	0	0
0	0	0	2	0	2	0
0	1	2	2	2	0	0
0	0	2	1	0	0	0
0	0	2	0	2	1	0
0	1	0	2	2	0	0
0	0	0	0	0	0	0

$x[:, :, 1]$

0	0	0	0	0	0	0
0	1	0	0	2	0	0
0	1	0	1	0	1	0
0	0	0	2	1	2	0
0	1	2	1	0	0	0
0	1	2	2	0	2	0
0	0	0	0	0	0	0

$x[:, :, 2]$

0	0	0	0	0	0	0
0	0	2	2	1	0	0
0	0	2	0	0	1	0
0	1	1	1	2	1	0
0	2	0	1	2	2	0
0	2	1	1	0	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$

-1	0	-1
1	-1	1
-1	0	-1

$w0[:, :, 1]$

0	-1	1
1	1	1
-1	1	-1

$w0[:, :, 2]$

0	0	0
-1	1	-1
1	1	-1

Bias  $b0$  (1x1x1)

$b0[:, :, 0]$

1
---

Filter W1 (3x3x3)

$w1[:, :, 0]$

0	1	1
-1	-1	-1
1	-1	1

$w1[:, :, 1]$

-1	0	-1
1	1	1
-1	1	-1

$w1[:, :, 2]$

0	1	0
0	0	-1
1	-1	0

Bias  $b1$  (1x1x1)

$b1[:, :, 0]$

0
---

Output Volume (3x3x2)

$o[:, :, 0]$

-3	-1	0
-1	-8	2
3	0	3

$o[:, :, 1]$

1	4	2
-1	4	5
3	1	3

toggle movement

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$

0	0	0	0	0	0	0
0	0	0	2	0	2	0
0	1	2	2	2	0	0
0	0	2	1	0	0	0
0	0	2	0	2	1	0
0	1	0	2	2	0	0
0	0	0	0	0	0	0

$x[:, :, 1]$

0	0	0	0	0	0	0
0	1	0	0	2	0	0
0	1	0	1	0	1	0
0	0	0	2	1	2	0
0	1	2	1	0	0	0
0	1	2	2	0	2	0
0	0	0	0	0	0	0

$x[:, :, 2]$

0	0	0	0	0	0	0
0	0	2	2	1	0	0
0	0	2	0	0	1	0
0	1	1	1	2	1	0
0	2	0	1	2	2	0
0	2	1	1	0	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$

-1	0	-1
1	-1	1
-1	0	-1

$w0[:, :, 1]$

0	-1	1
1	1	1
-1	1	-1

$w0[:, :, 2]$

0	0	0
-1	1	-1
1	1	-1

Bias  $b0$  (1x1x1)

$b0[:, :, 0]$

1
---

Filter W1 (3x3x3)

$w1[:, :, 0]$

0	1	1
-1	-1	-1
1	-1	1

$w1[:, :, 1]$

-1	0	-1
1	1	1
-1	1	-1

$w1[:, :, 2]$

0	1	0
0	0	-1
1	-1	0

Bias  $b1$  (1x1x1)

$b1[:, :, 0]$

0
---

Output Volume (3x3x2)

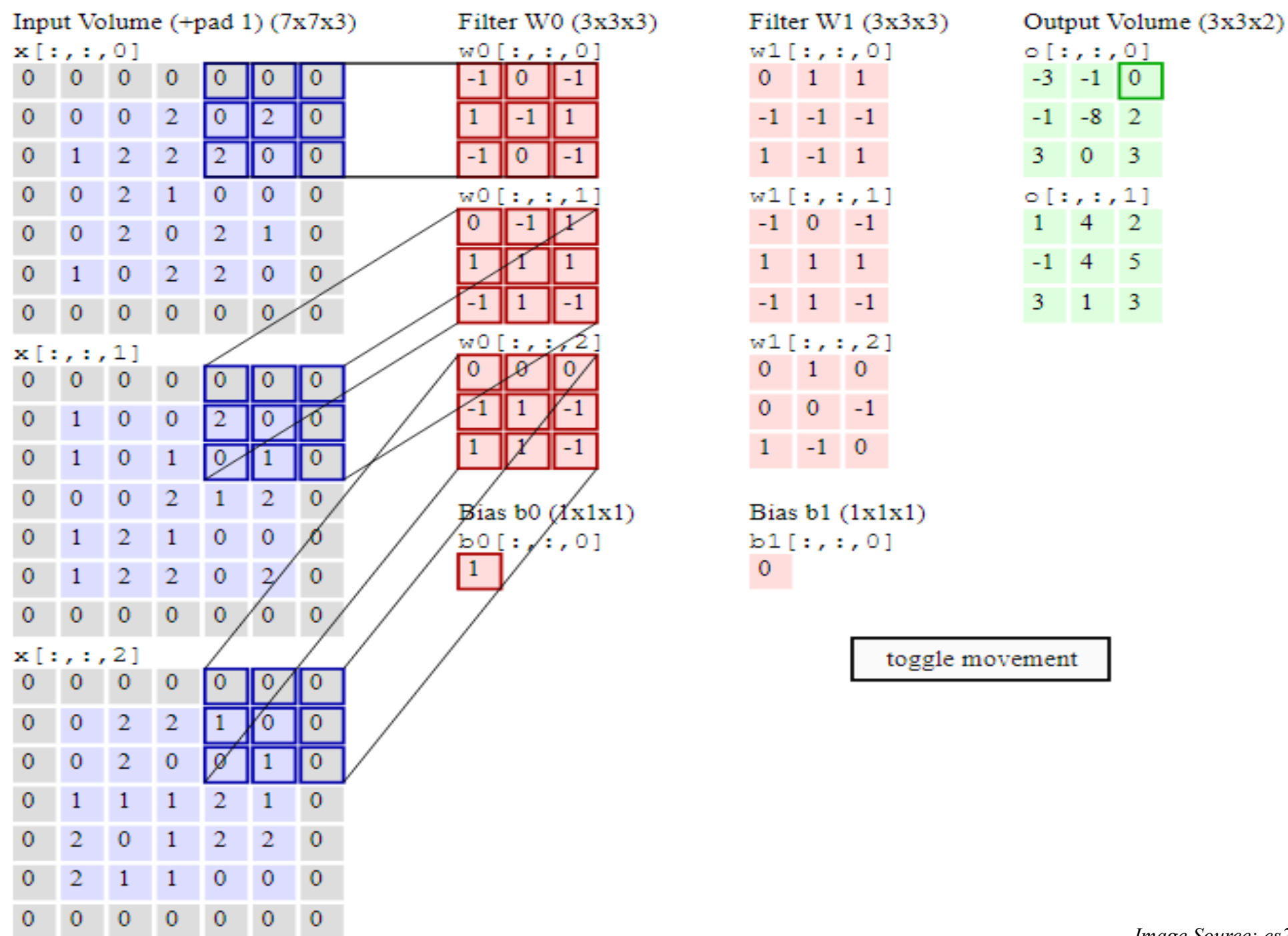
$o[:, :, 0]$

-3	-1	0
-1	-8	2
3	0	3

$o[:, :, 1]$

1	4	2
-1	4	5
3	1	3

toggle movement



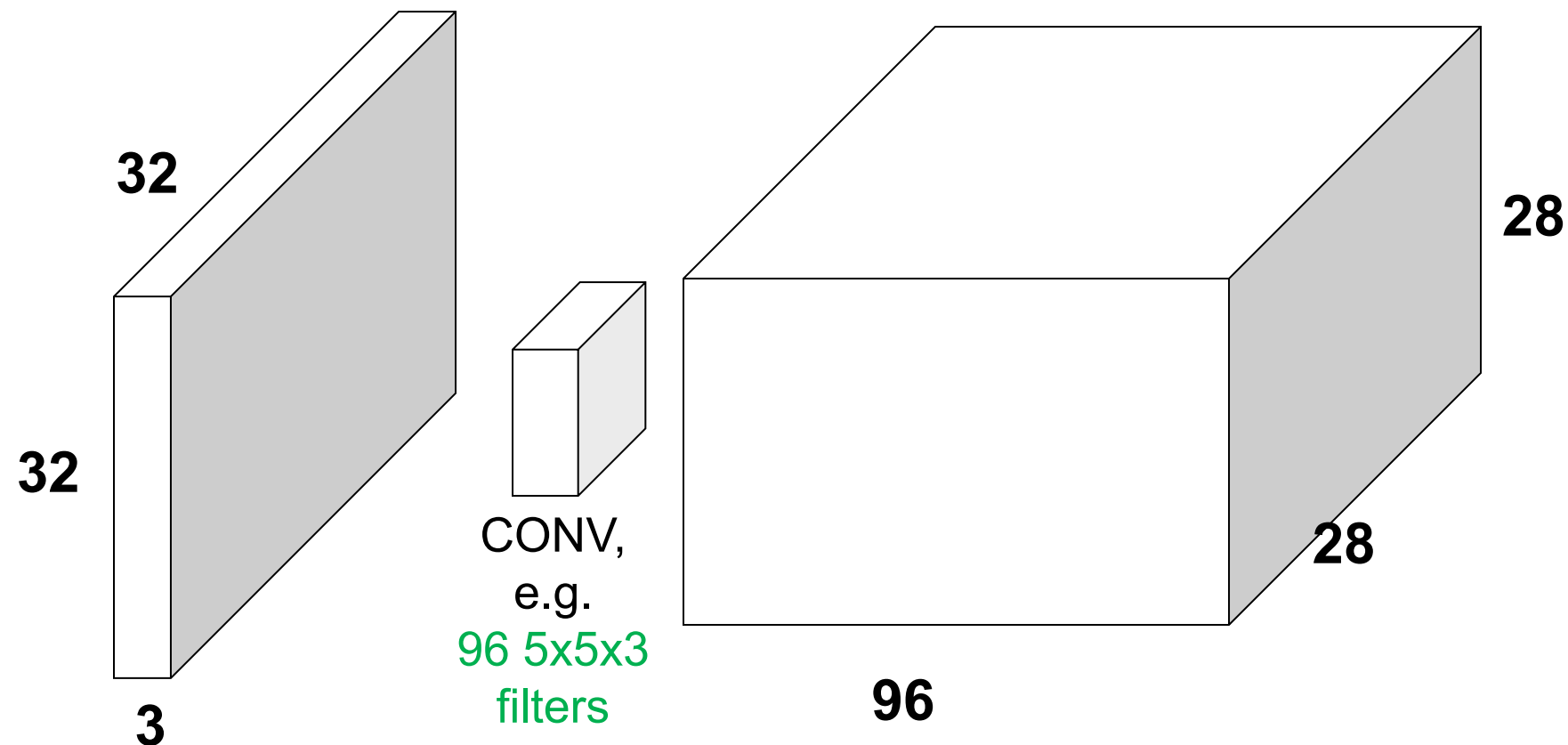
# Convolutional Layer

---

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

**32×32×3 Image**

**Activation maps**

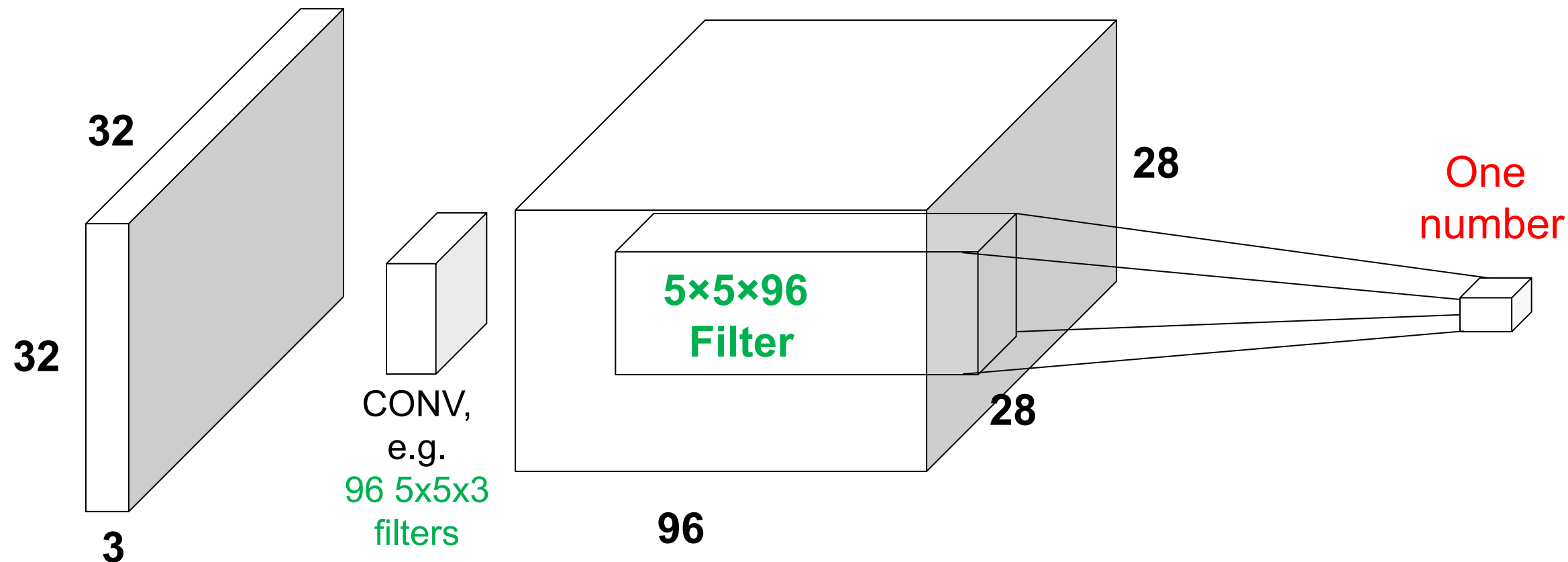


# Convolutional Layer

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

**32×32×3 Image**

**Activation maps**



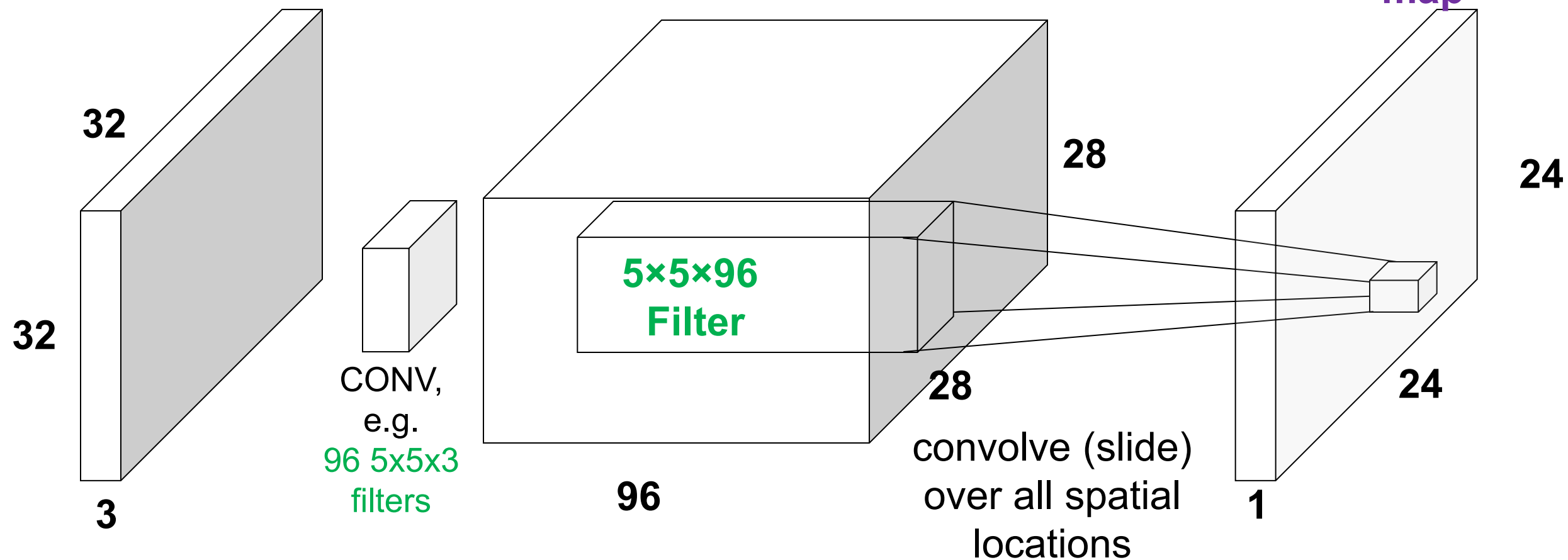
# Convolutional Layer

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

**32×32×3 Image**

**Activation maps**

**Deeper activation map**



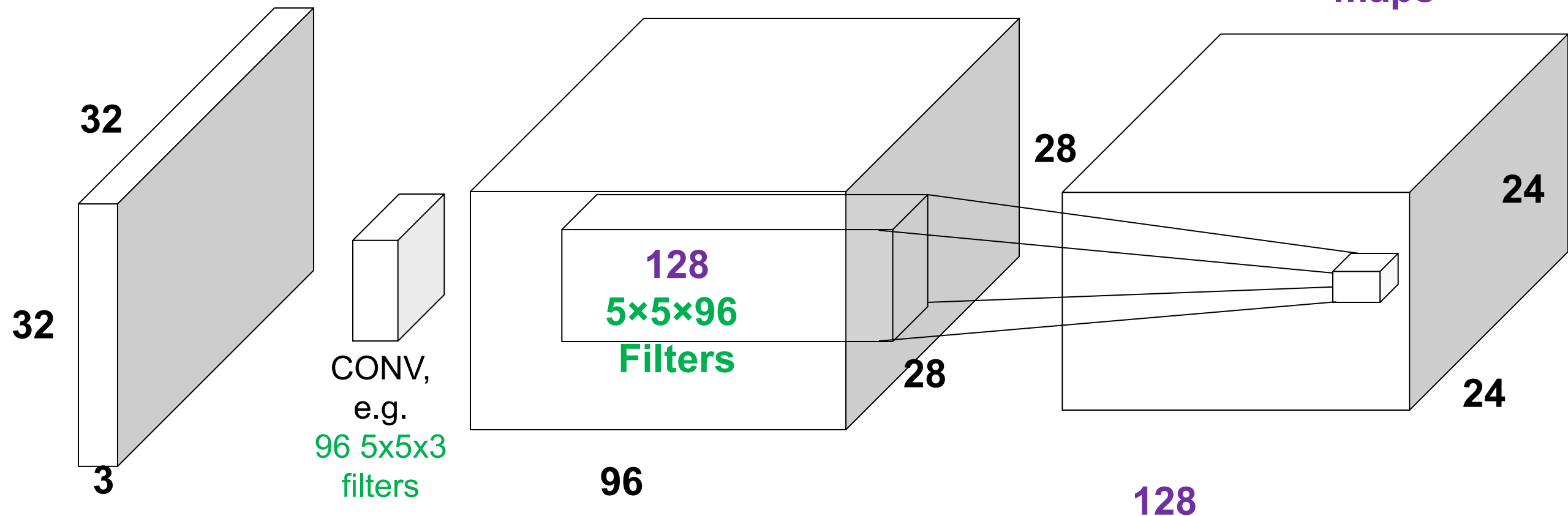
# Convolutional Layer

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

**32×32×3 Image**

**Activation maps**

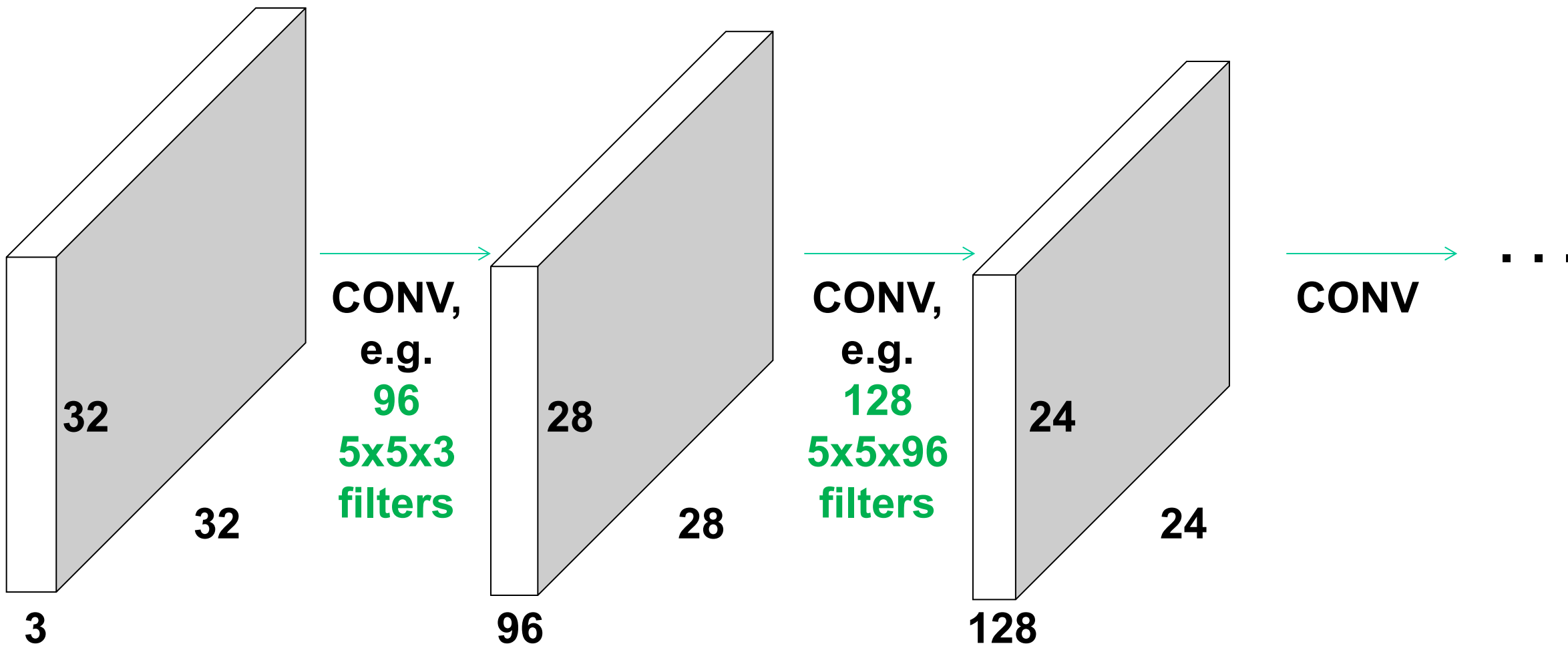
**Deeper activation maps**



# Multilayer Convolution

---

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





# Any Convolution Layer

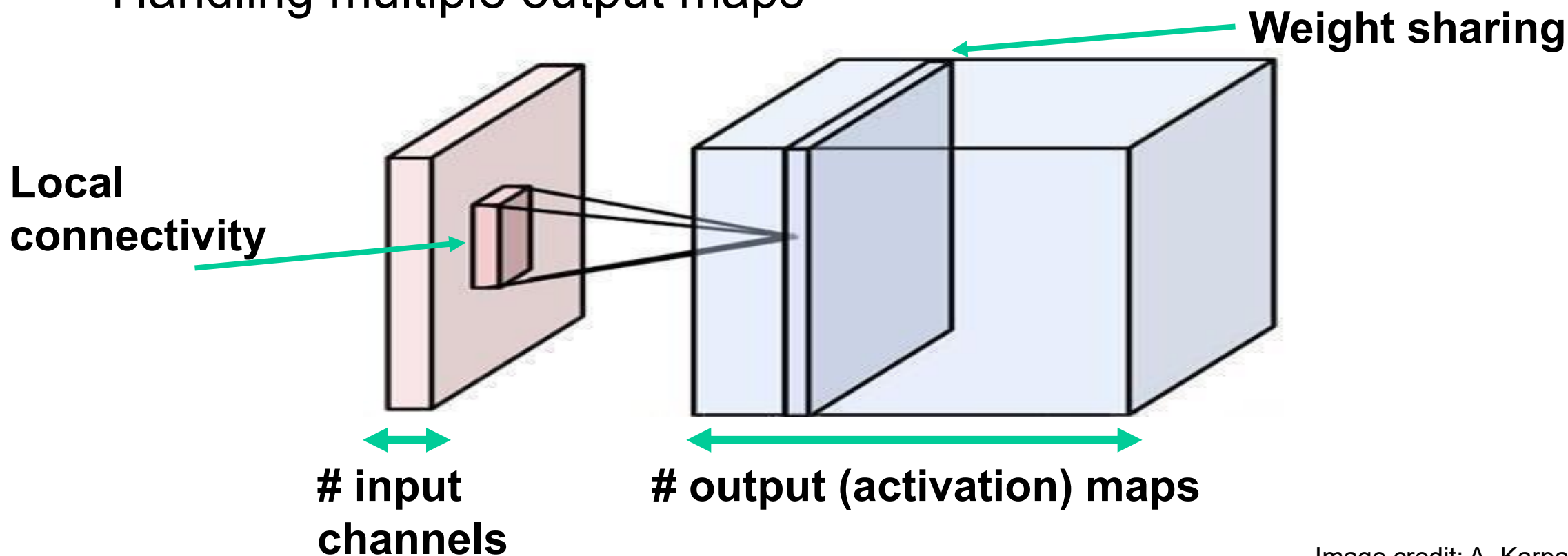
---

Local connectivity

Weight sharing

Handling multiple input channels

Handling multiple output maps



# A closer look at spatial dimensions

---

7

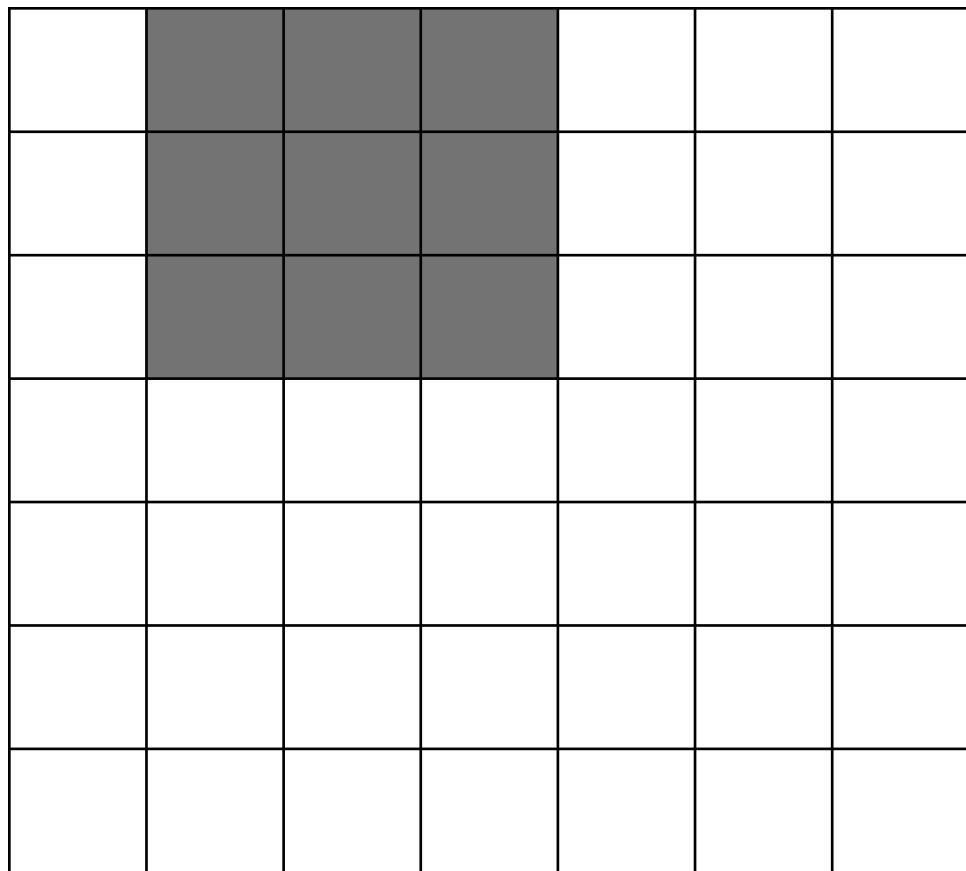

7

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

# A closer look at spatial dimensions

---

7



7

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

# A closer look at spatial dimensions

---

7


7

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

# A closer look at spatial dimensions

---

7

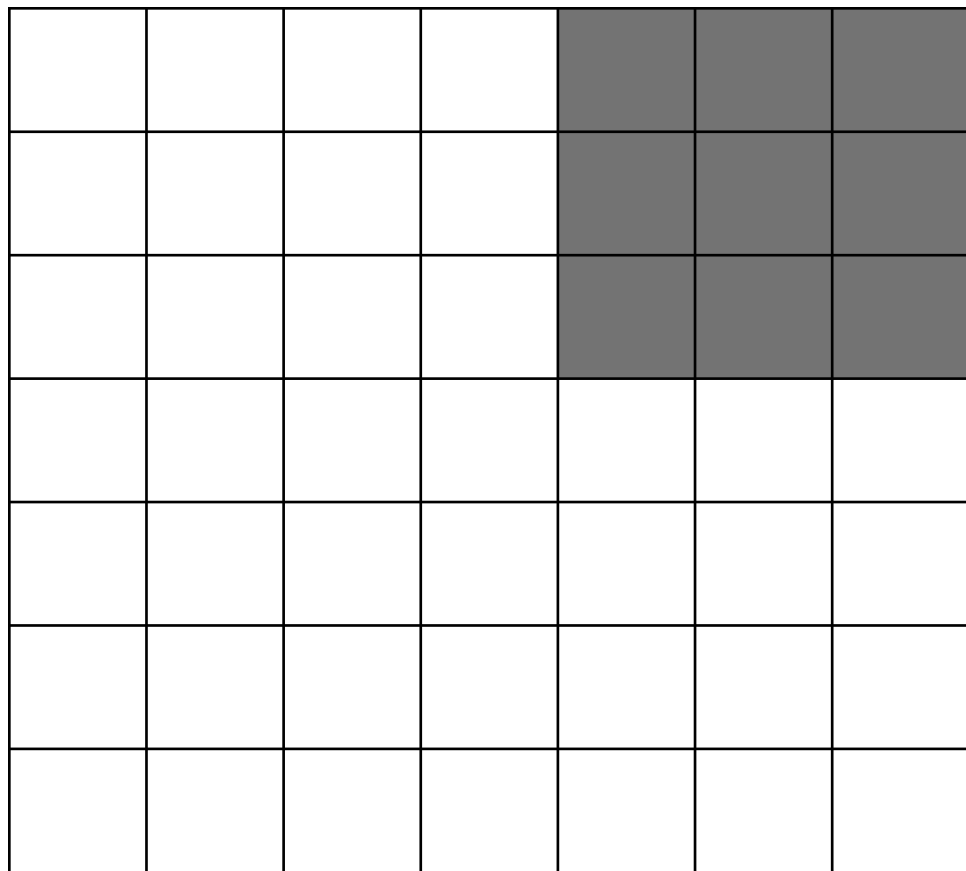

7

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

# A closer look at spatial dimensions

---

7



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

7



5×5 output

# A closer look at spatial dimensions

---

7


7

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

# A closer look at spatial dimensions

---

7


7

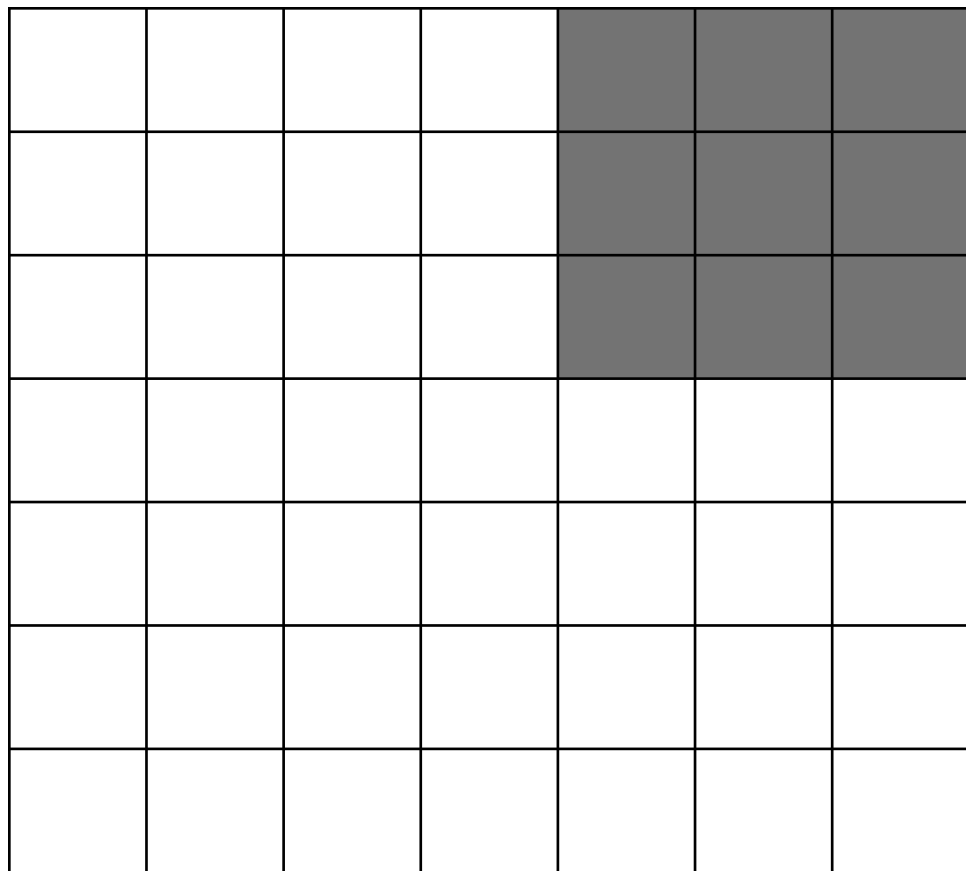
7×7 input (spatially)  
assume 3×3 filter  
applied with stride 2



# A closer look at spatial dimensions

---

7



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

7



3×3 output

# A closer look at spatial dimensions

---

7


7

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

# A closer look at spatial dimensions

---

7


7

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

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

# A closer look at spatial dimensions

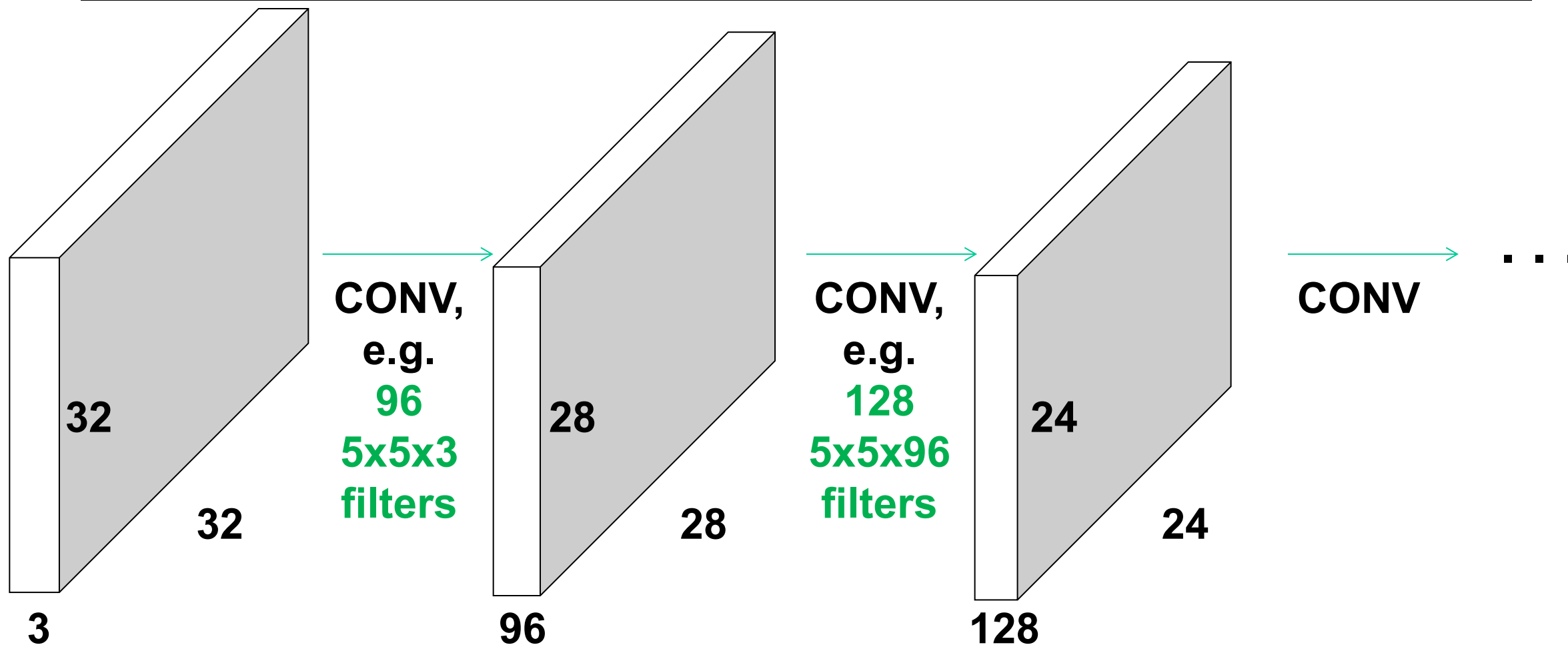
---

N						
			F			
	F					

Output size  
 $(N - F) / \text{stride} + 1$

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

# A closer look at spatial dimensions



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.

# In practice: common to zero pad

---

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

e.g. input 7×7 (spatially)

3×3 filter, applied with stride 1

pad with 1 pixel border

What is the output dimension?

# In practice: common to zero pad

---

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

e.g. input 7×7 (spatially)  
3×3 filter, applied with stride 1  
pad with 1 pixel border

7×7 Output

# In practice: common to zero pad

---

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

e.g. input 7×7 (spatially)

3×3 filter, applied with stride 1

pad with 1 pixel border

**7×7 Output**

in general, common to see CONV layers with stride 1, filters of size  $F \times F$ , and zero-padding with

$(F-1)/2$ . (will preserve size spatially)

e.g.

$F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

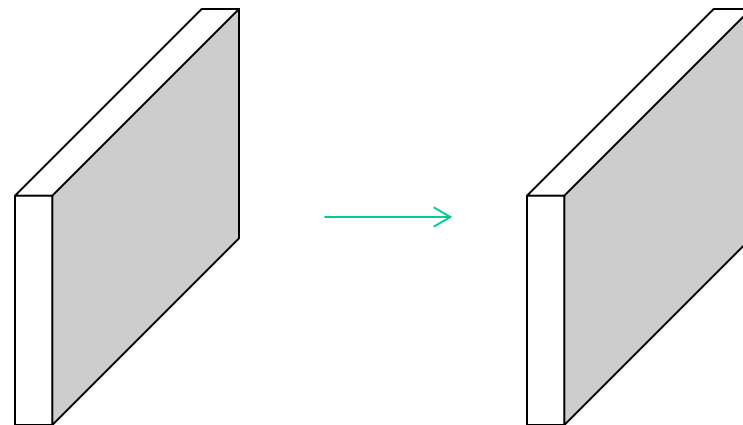


# Example

---

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

Output volume size: ?

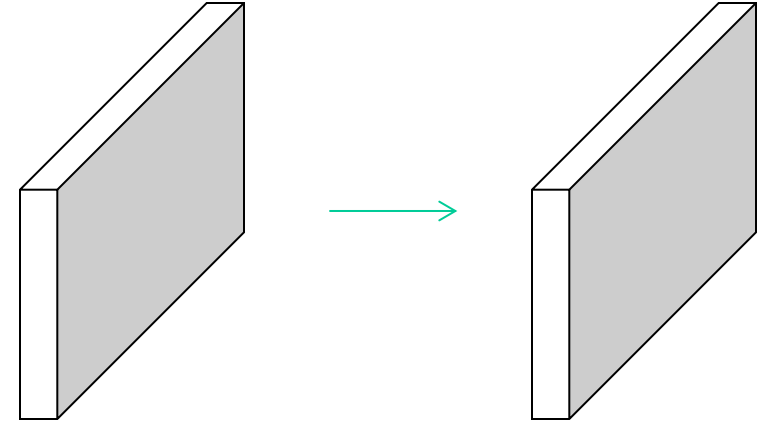


# Example

---

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

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



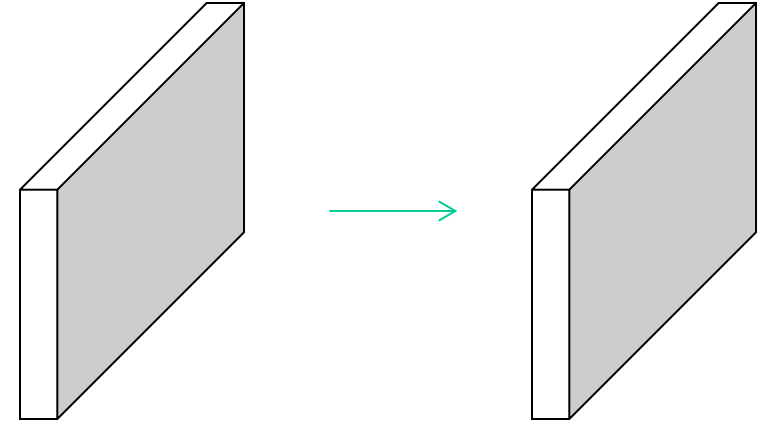
# Example

---

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?



# Example

---

Input volume: 32x32x3

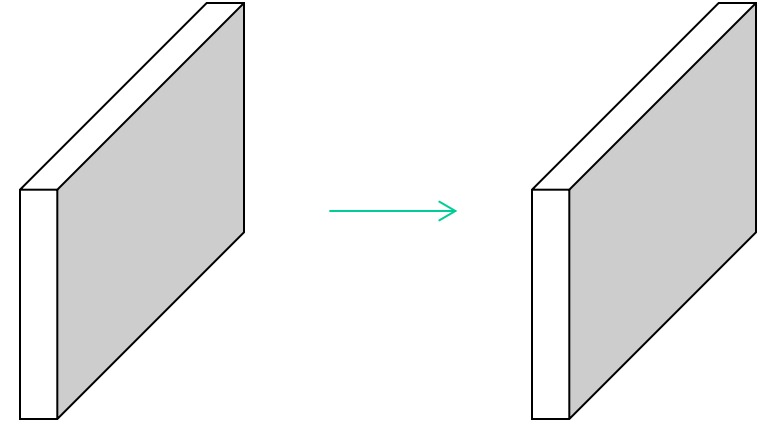
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has

$5*5*3 + 1 = 76$  params (+1 for bias)

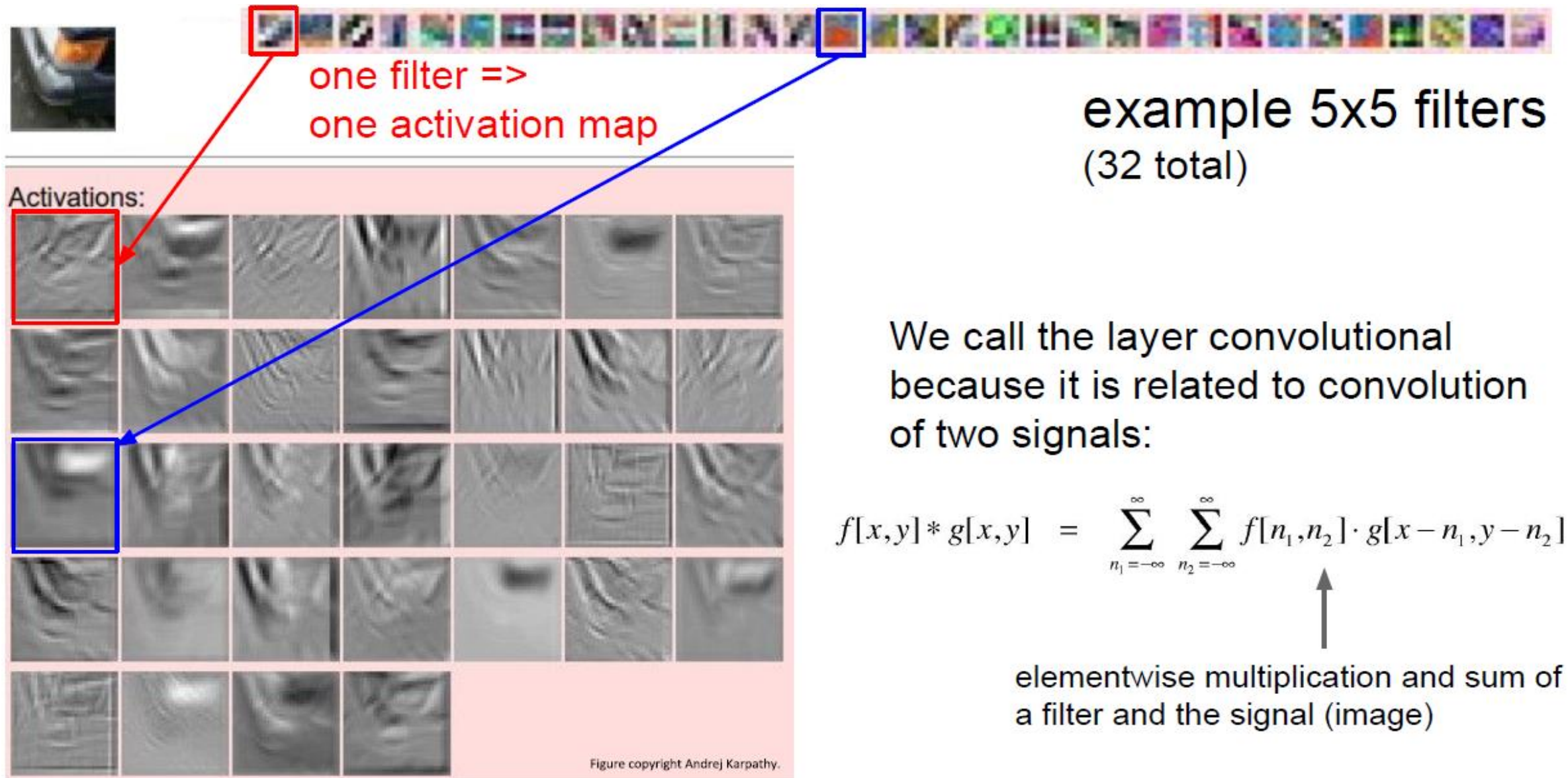
$\Rightarrow 76*10 = 760$



**Summary.** To summarize, the Conv Layer:

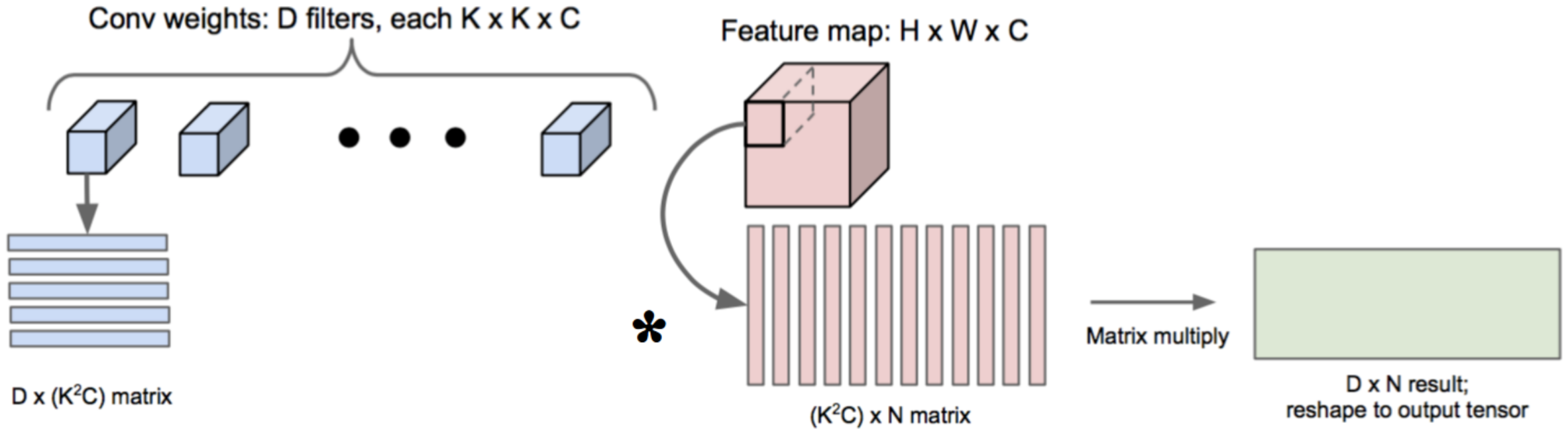
- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (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.

# Convolution as feature extraction



# Efficient implementation of convolutions

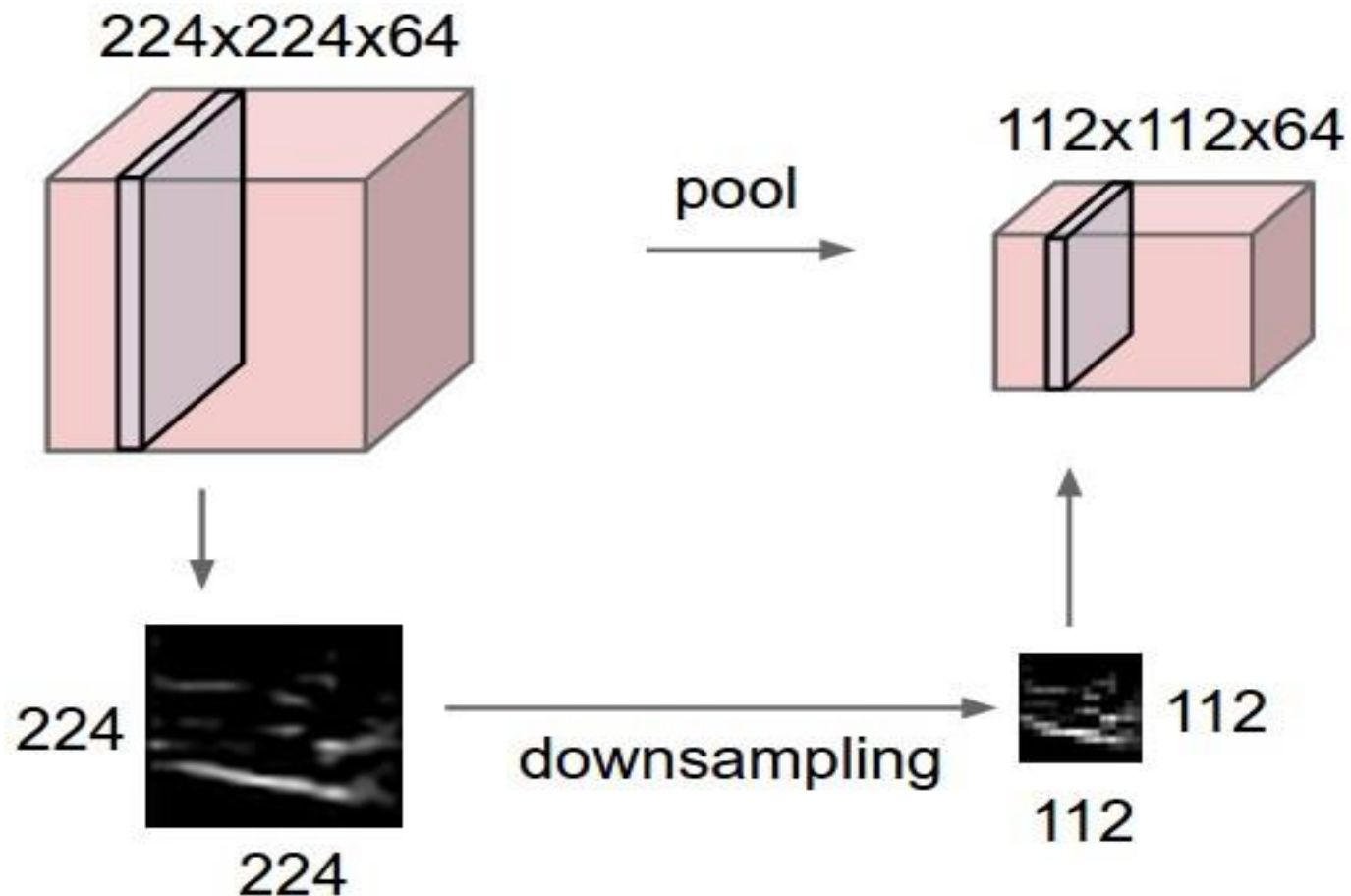
- Reshape all image neighborhoods into columns (im2col operation), do matrix-vector multiplication



# Pooling Layer

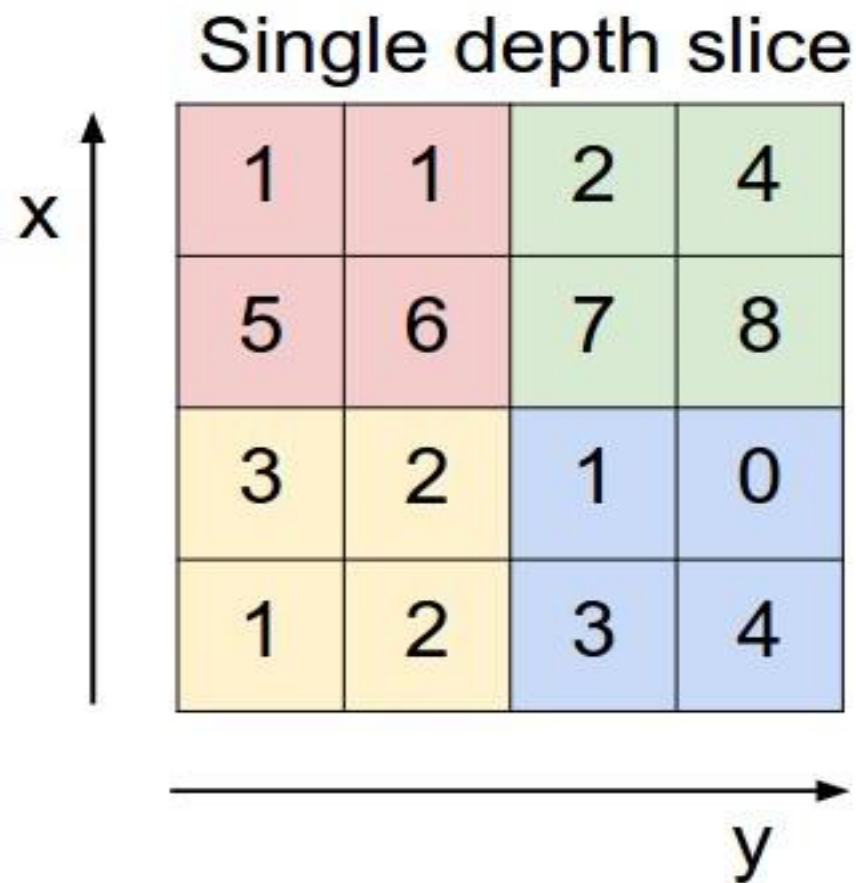
---

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

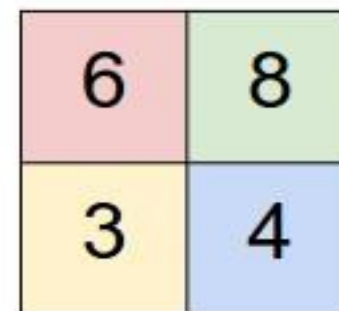




# Max Pooling



max pool with 2x2 filters  
and stride 2



Backward pass: upstream  
gradient is passed back only to  
the unit with max value

# Pooling Layer

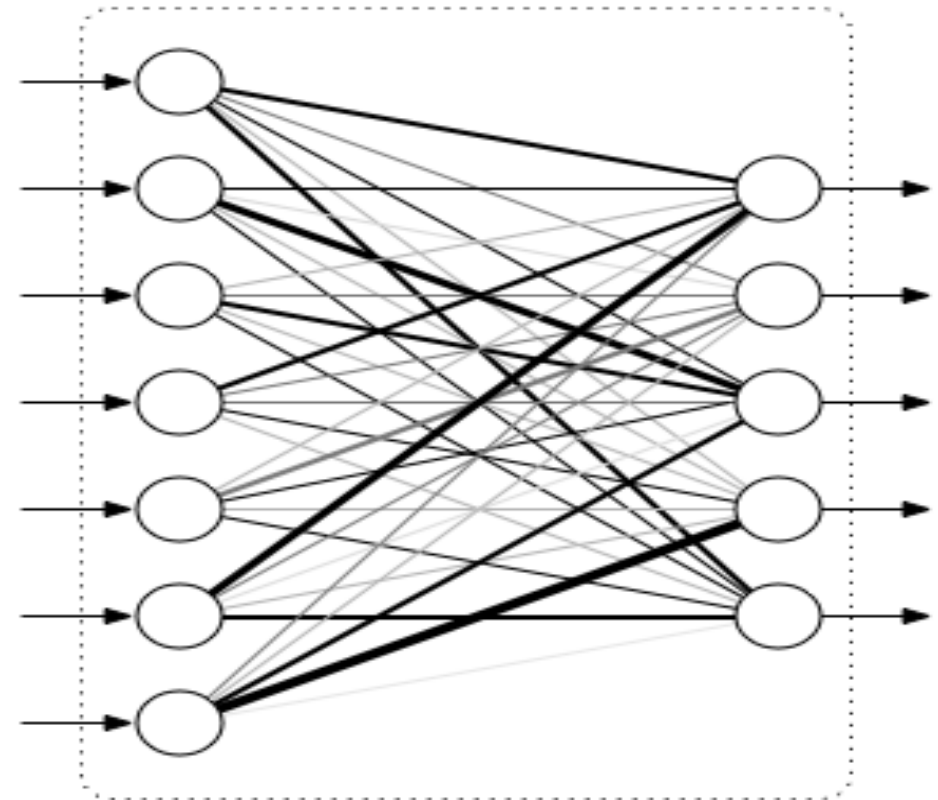
---

- 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

# Fully Connected Layer

---

- **Connect every neuron in one layer to every neuron in another layer**
- **Same as the traditional multi-layer perceptron neural network**

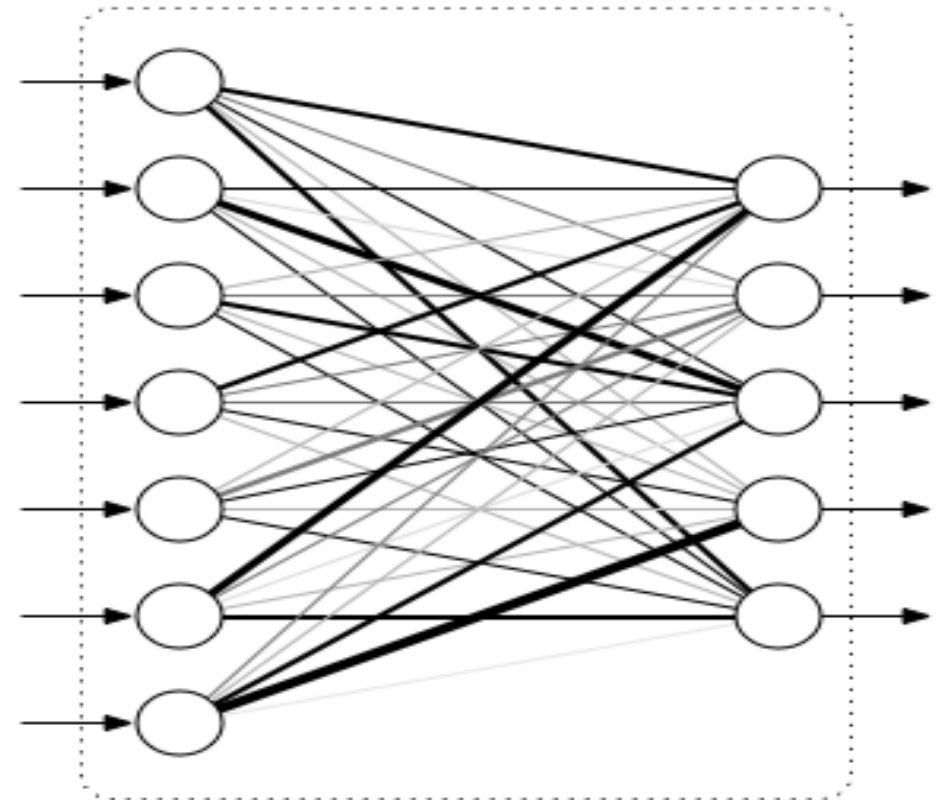


# Fully Connected Layer

---

- **Connect every neuron in one layer to every neuron in another layer**
- **Same as the traditional multi-layer perceptron neural network**

**No. of Neurons (Last FC)  
= No. of classes**



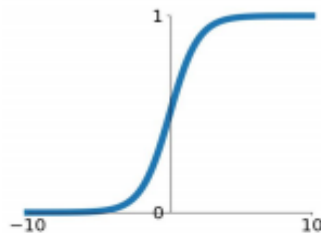
# Non-linearity Layer

---

## Activation Functions

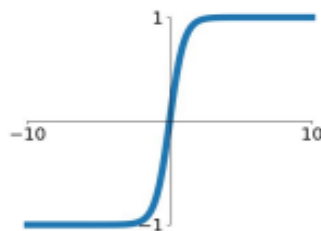
### Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



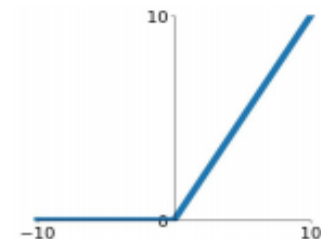
### tanh

$$\tanh(x)$$



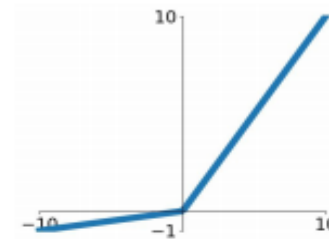
### ReLU

$$\max(0, x)$$



### Leaky ReLU

$$\max(0.1x, x)$$

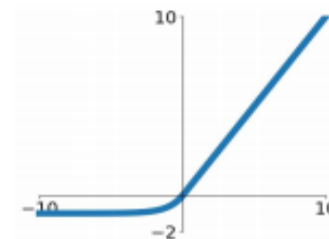


### Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

### ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Classification/Loss Layer

---

## SVM Classifier

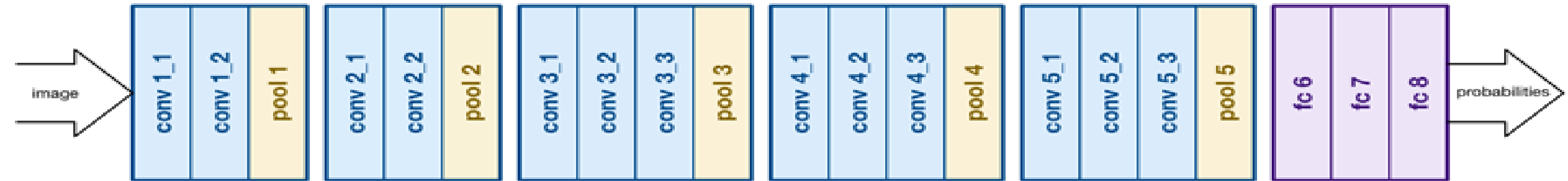
SVM Loss/Hinge Loss/Max-margin Loss

## Softmax Classifier

Softmax Loss/Cross-entropy Loss

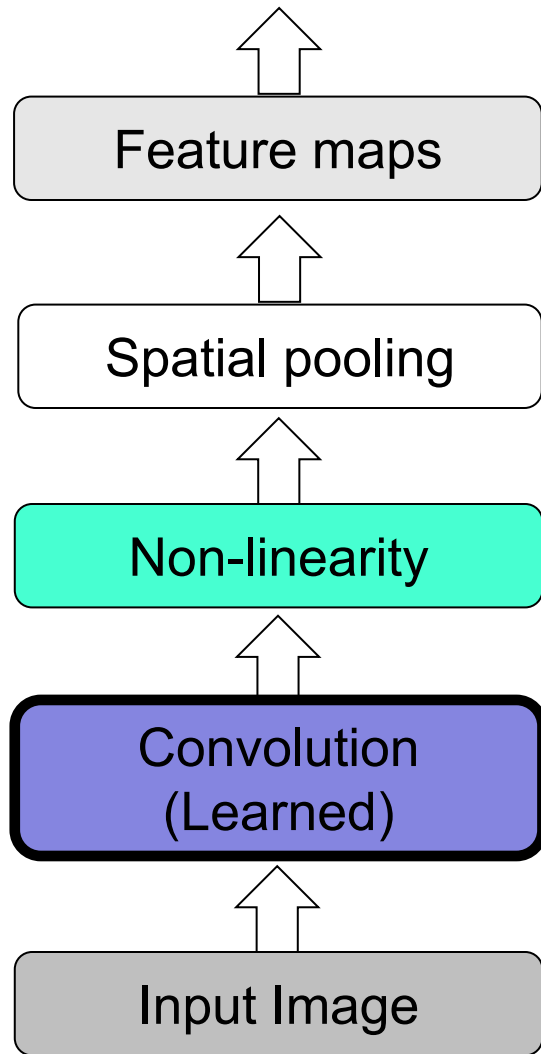
# A typical CNN structure

---



# Summary: CNN pipeline

---



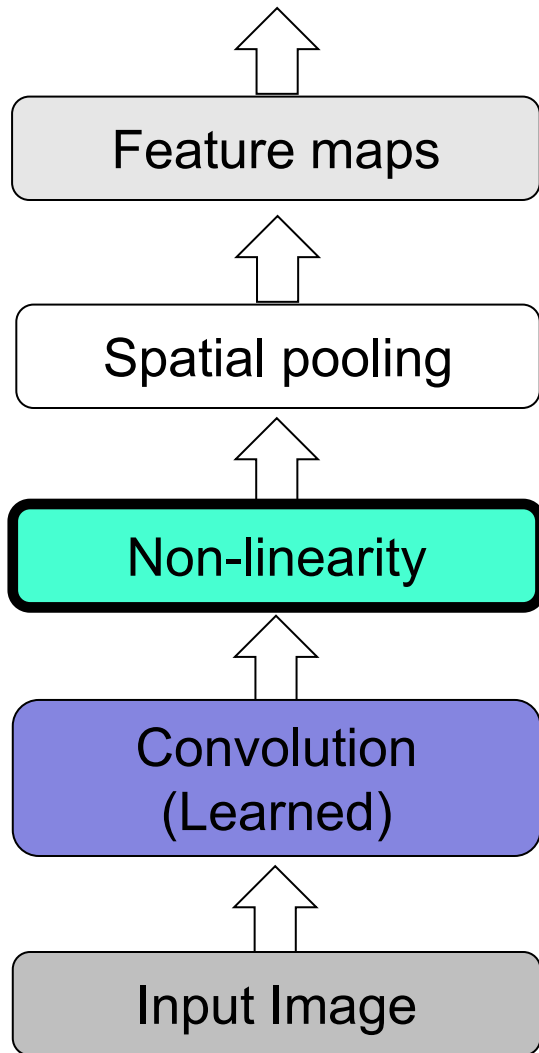
Input



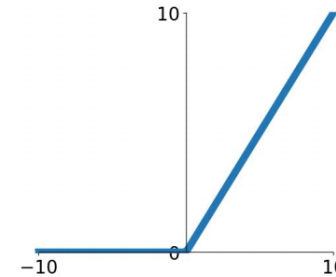
Feature Map



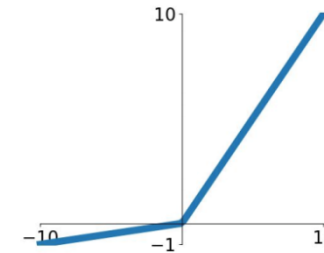
# Summary: CNN pipeline



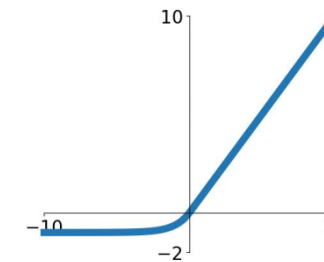
**ReLU**  
 $\max(0, x)$



**Leaky ReLU**  
 $\max(0.1x, x)$

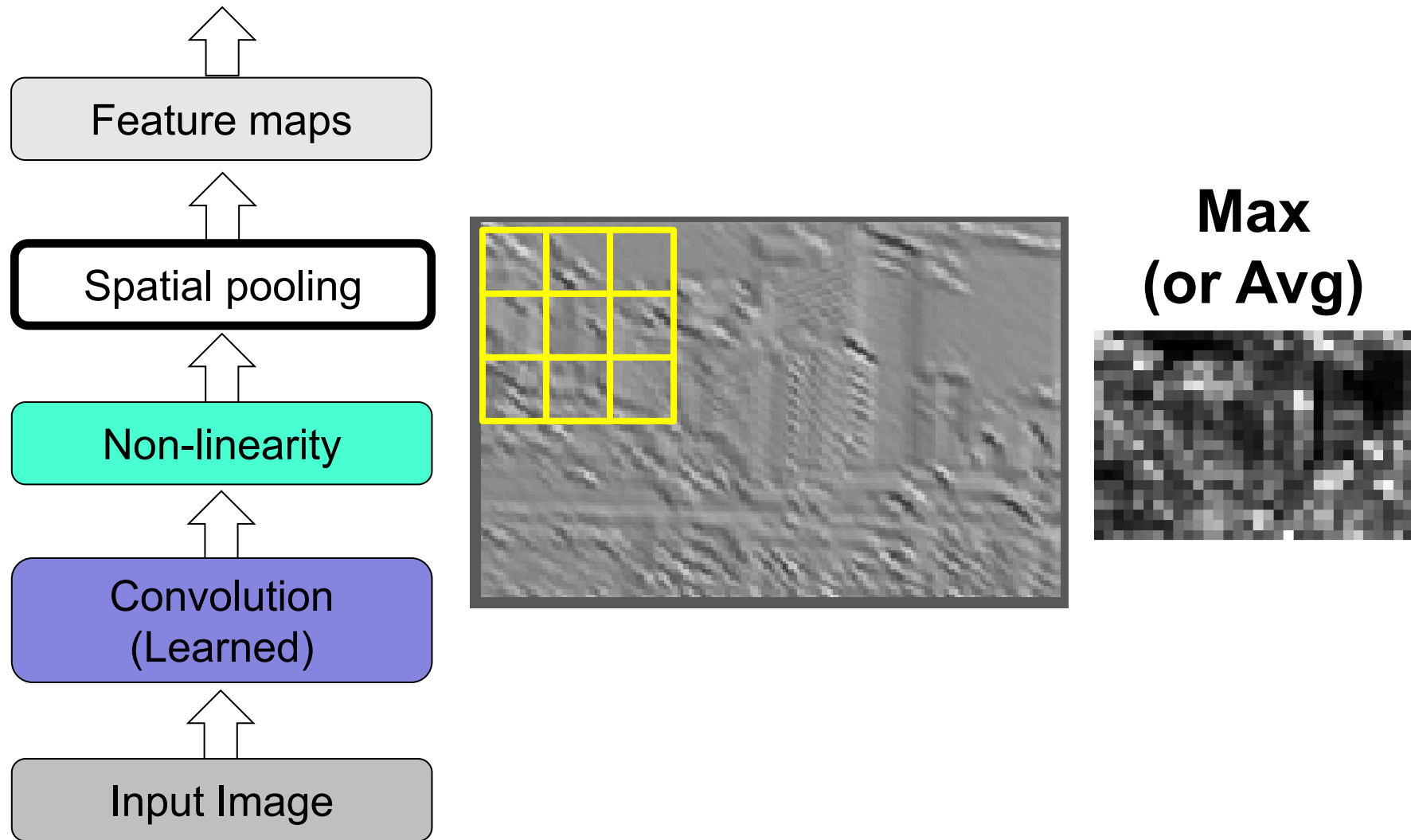


**ELU**  
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



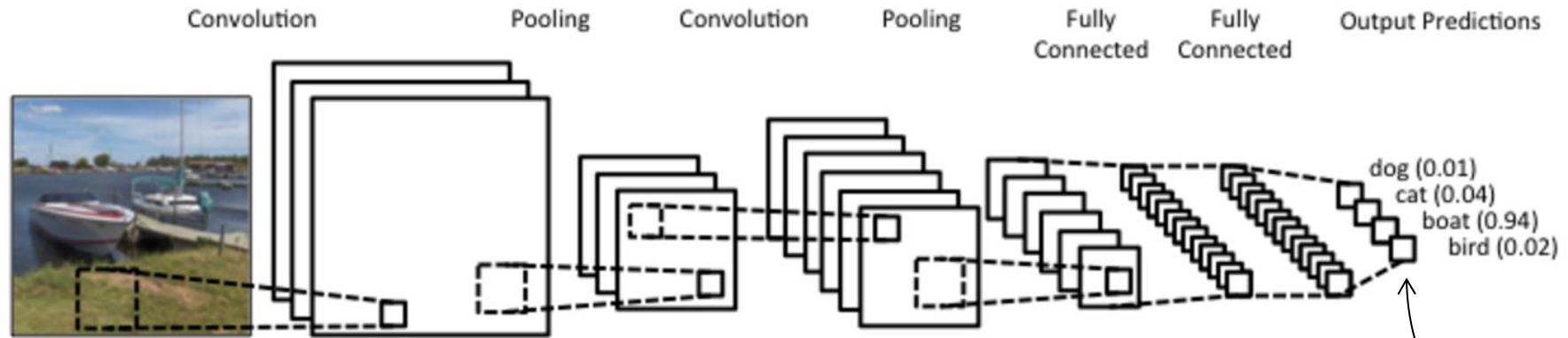
# Summary: CNN pipeline

---



# Summary: CNN pipeline

---

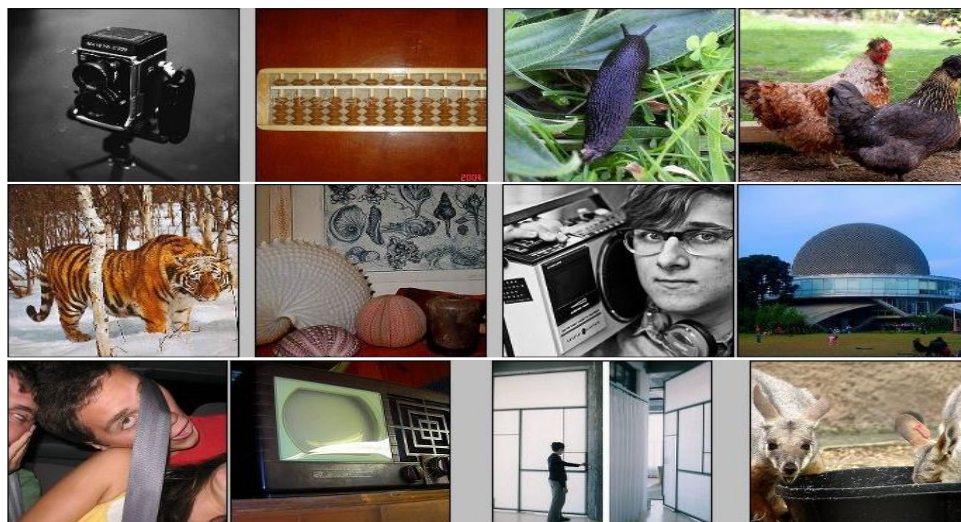


Softmax layer:

# ImageNet Challenge

---

IMAGENET



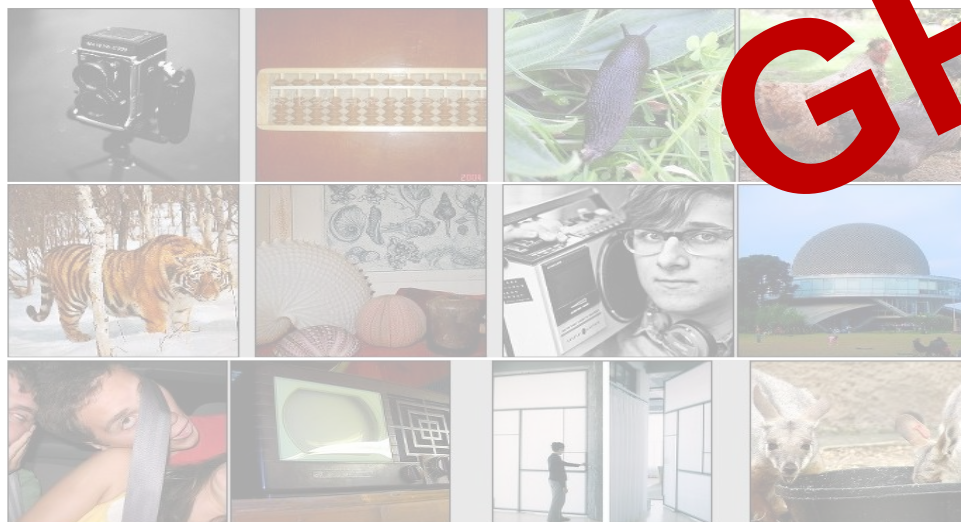
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- Challenge: 1.2 million training images, 1000 classes

[www.image-net.org/challenges/LSVRC/](http://www.image-net.org/challenges/LSVRC/)

# ImageNet Challenge

---

IMAGENET



**GPUS  
+  
Data**

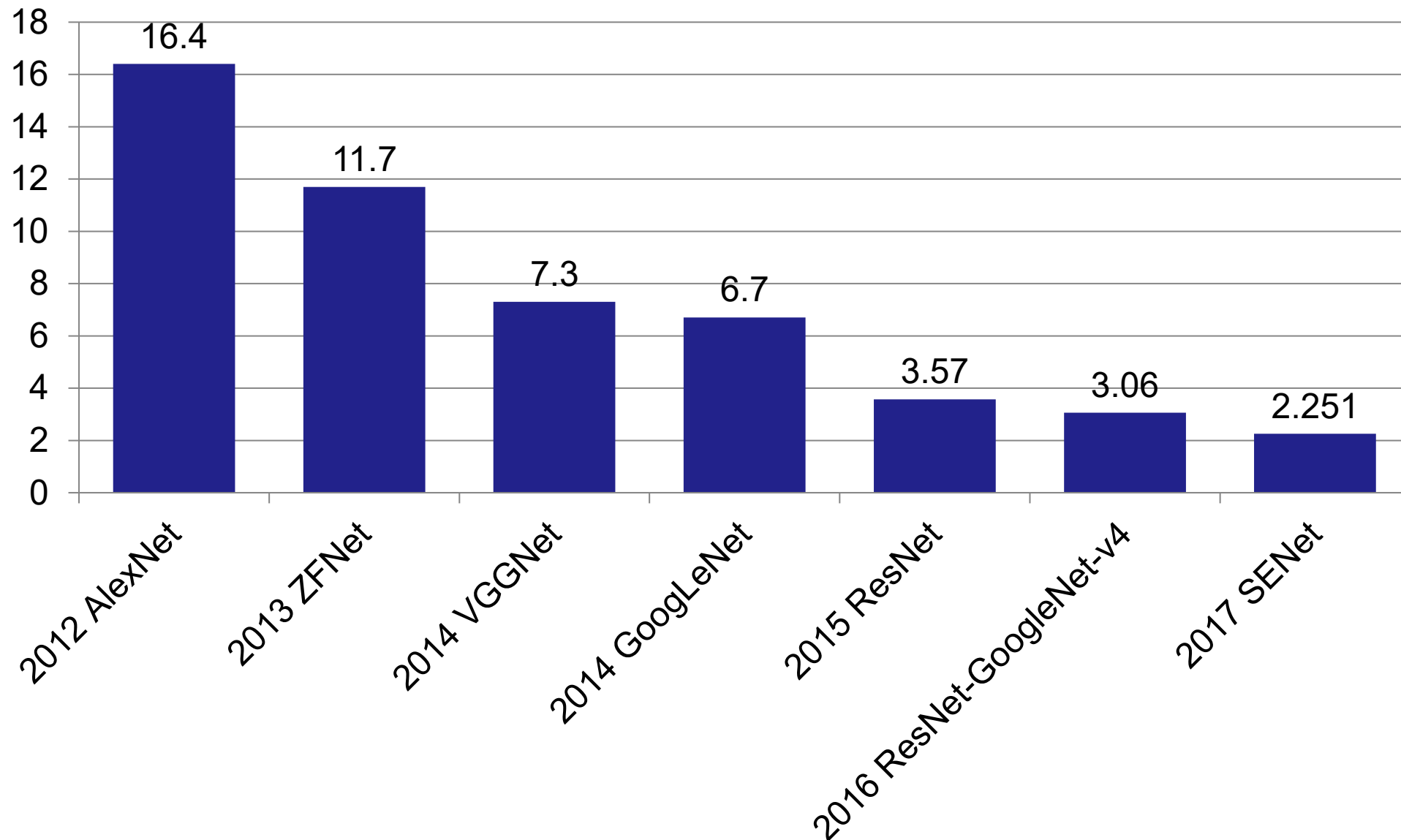
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- Challenge: 1.2 million training images, 1000 classes

[www.image-net.org/challenges/LSVRC/](http://www.image-net.org/challenges/LSVRC/)

# Progress on ImageNet Challenge

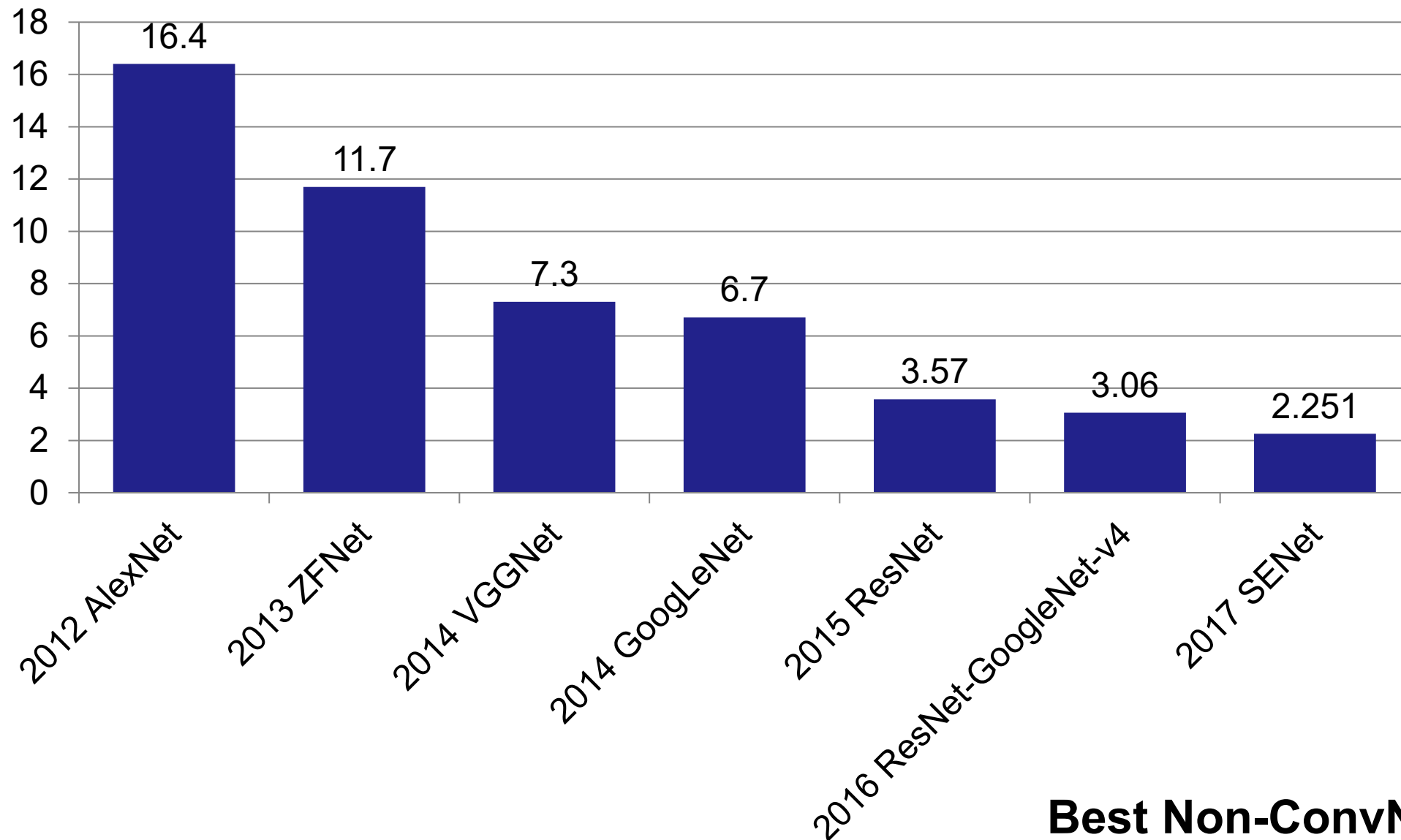
---

ImageNet Image Classification Top5 Error



# Progress on ImageNet Challenge

ImageNet Image Classification Top5 Error



**Best Non-ConvNet in 2012: 26.2%**

# Things to remember

---

## Neural network and Image

- Neuroscience, Perceptron, Problems due to High Dimensionality and Local Relationship

## Convolutional neural network (CNN)

- Convolution Layer,
- Nonlinearity Layer,
- Pooling Layer,
- Fully Connected Layer,
- Loss/Classification Layer

## Progress on ImageNet challenge

- Latest SENet, Winner 2017



# ACKNOWLEDGEMENT

Thanks to the following courses and corresponding researchers for making their teaching/research material online

- Convolutional Neural Networks for Visual Recognition, Stanford University
- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Natural Language Processing with Deep Learning, Stanford University
- And Many More Publicly Available Resources .....



# Questions?

