# Knowledge Discovery & Data Mining
# – Convolutional Neural Networks –

**Instructor: Yong Zhuang**

***yong.zhuang@gvsu.edu***

Based on the original version at https://cs231n.github.io/convolutional-networks/

# Perceptron

## A bit of history…

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

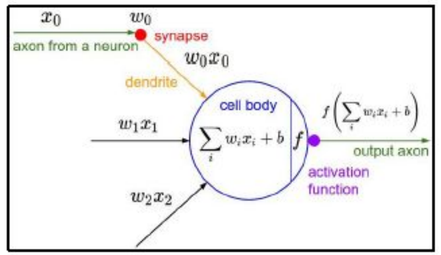
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

update rule:

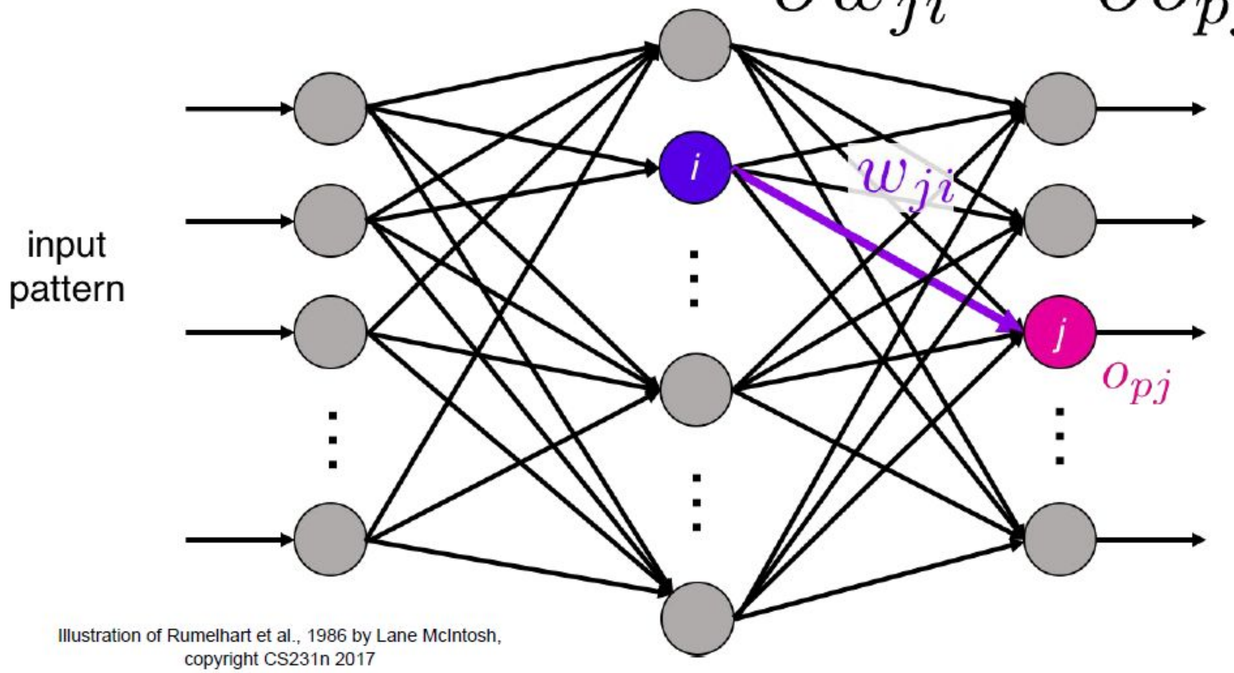
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$




Frank Rosenblatt, ~1957: Perceptron

# Back-propagation

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$




Rumelhart et al., 1986: First time back-propagation became popular

# First Deep Neural Network

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

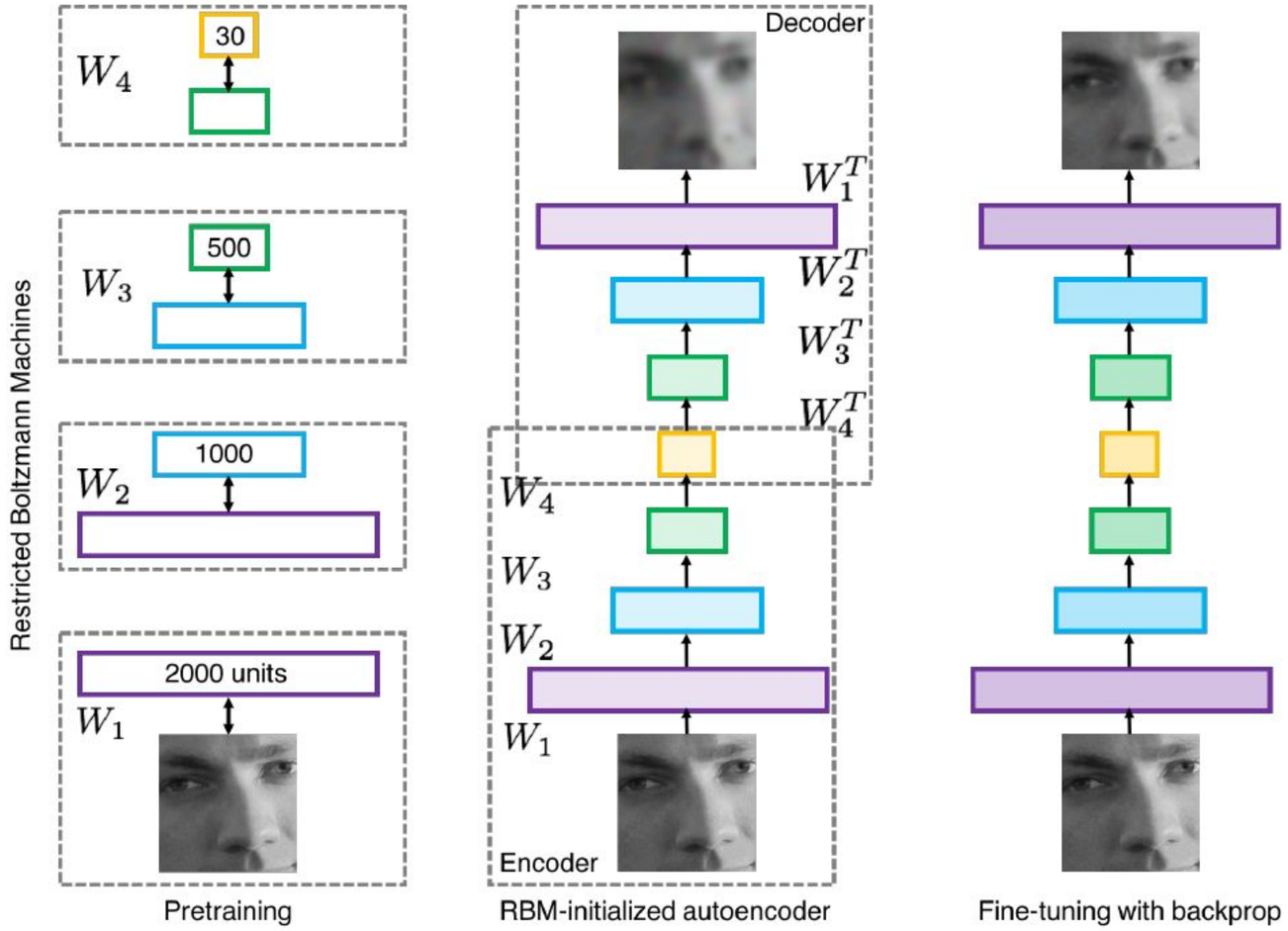



Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

# First Strong Results

***Acoustic Modeling using Deep Belief Networks***
*Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010*
***Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition***
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

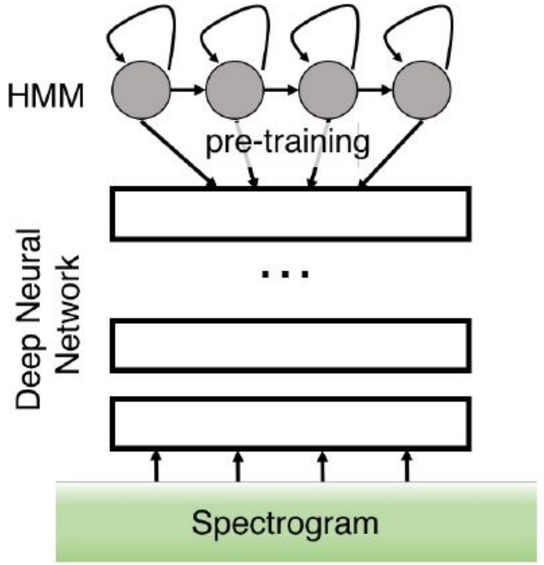
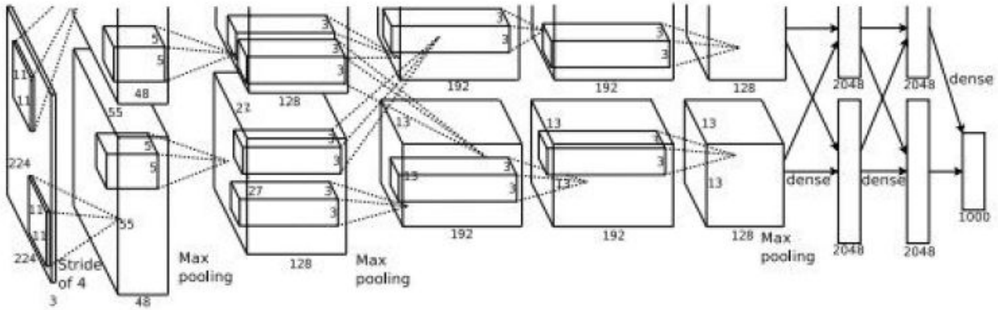



Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

***Imagenet classification with deep convolutional neural networks***
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012




***landmark paper***

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

# What specifically led to the development of convolutional neural networks?

# Hubel & Wiesel's Experiments

**Hubel & Wiesel,**

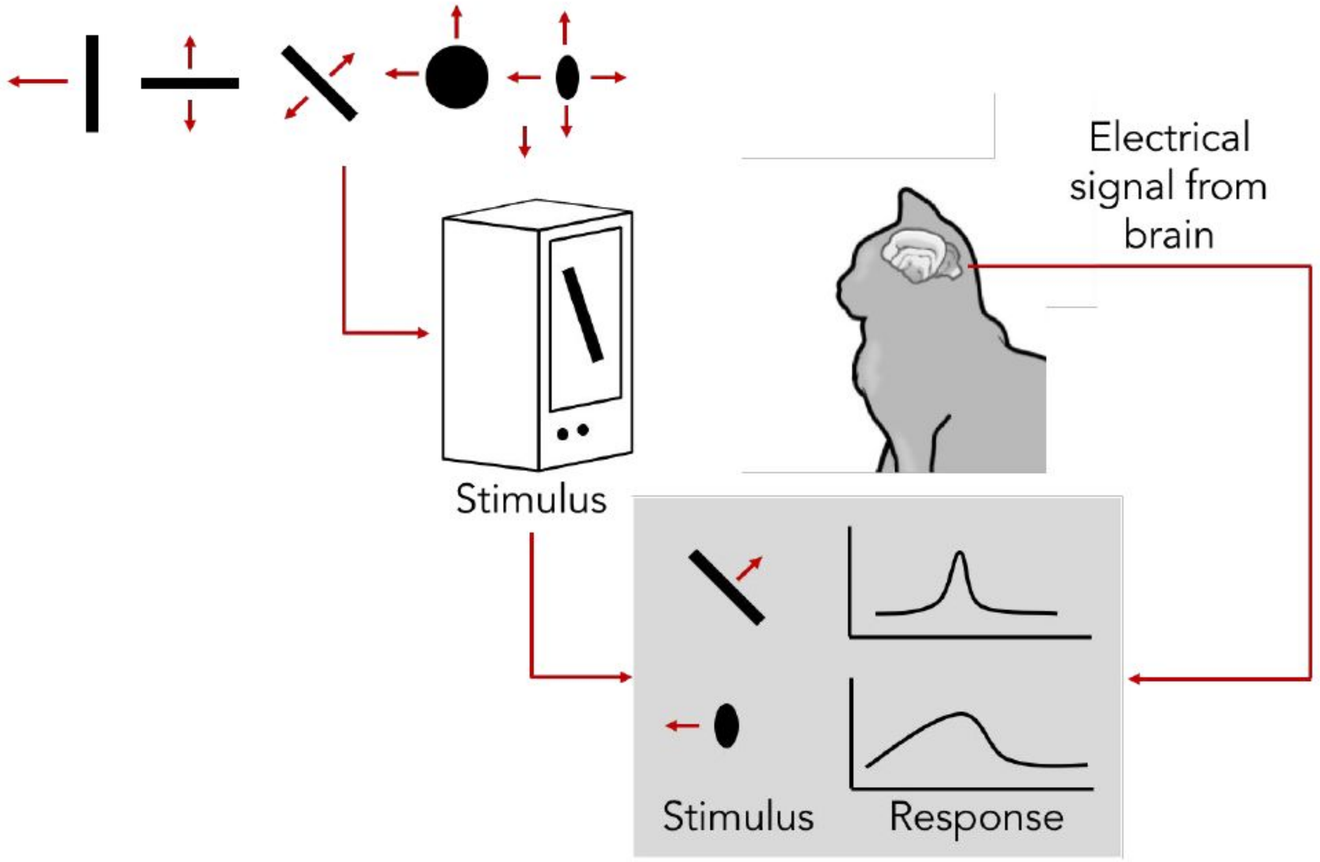
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

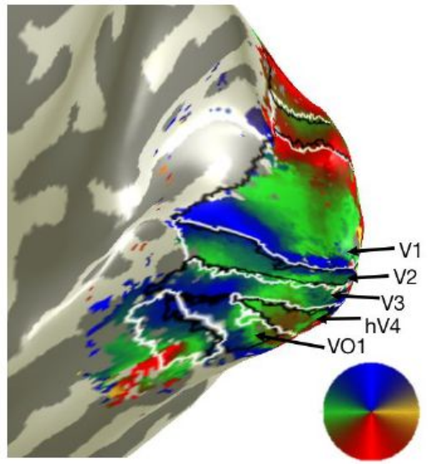
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

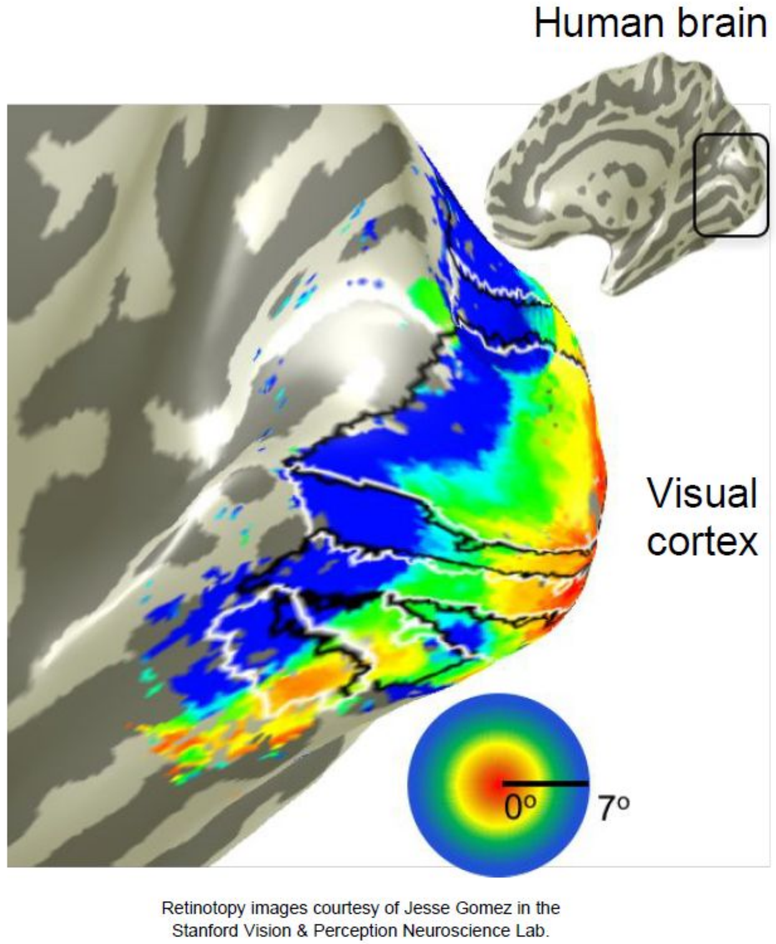
1968…



# Topographical mapping in the cortex

nearby cells in cortex represent
nearby regions in the visual field




Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

# Hierarchical Organization

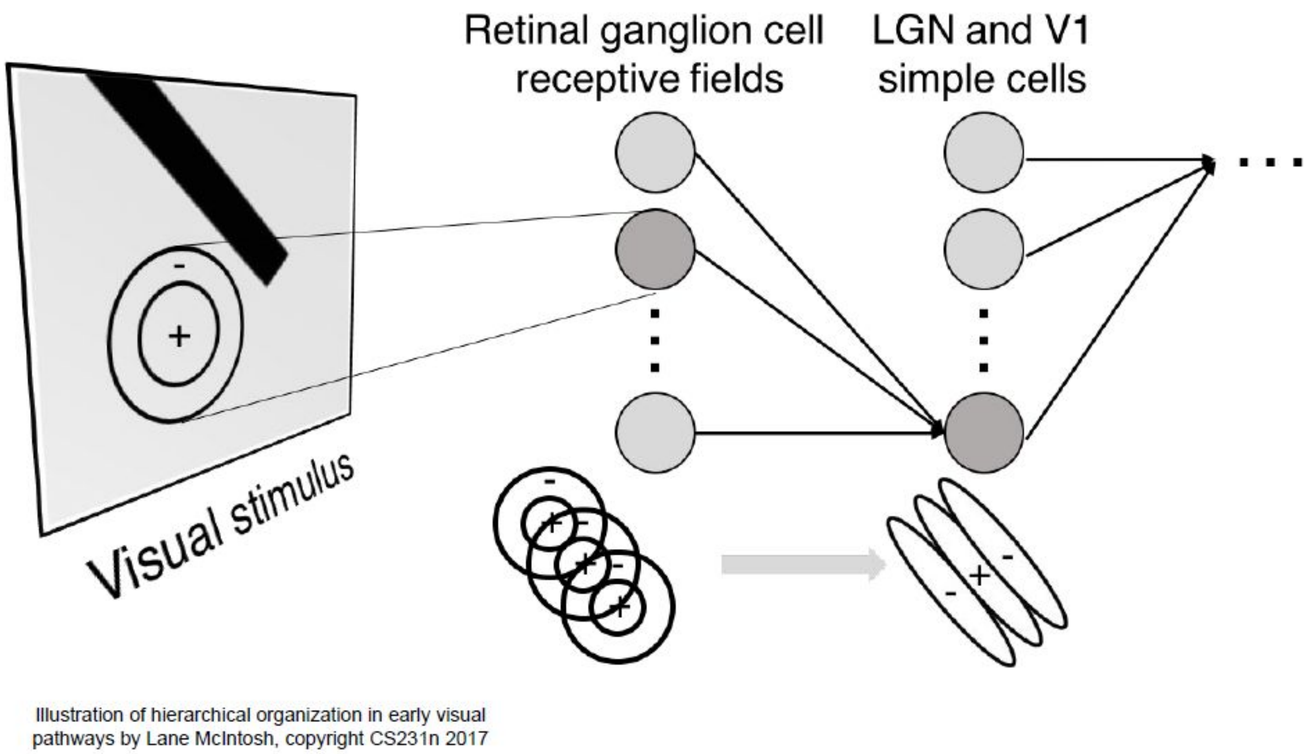
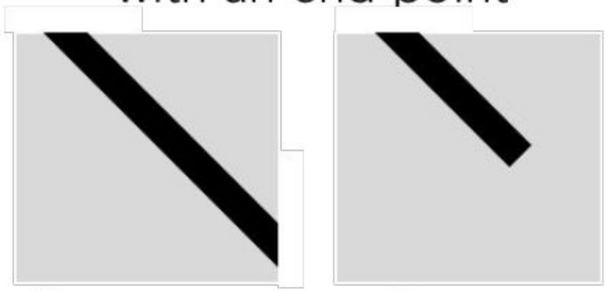



Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

**Simple cells:**
Response to light orientation

**Complex cells:**
Response to light orientation and movement

**Hypercomplex cells:**
response to movement with an end point

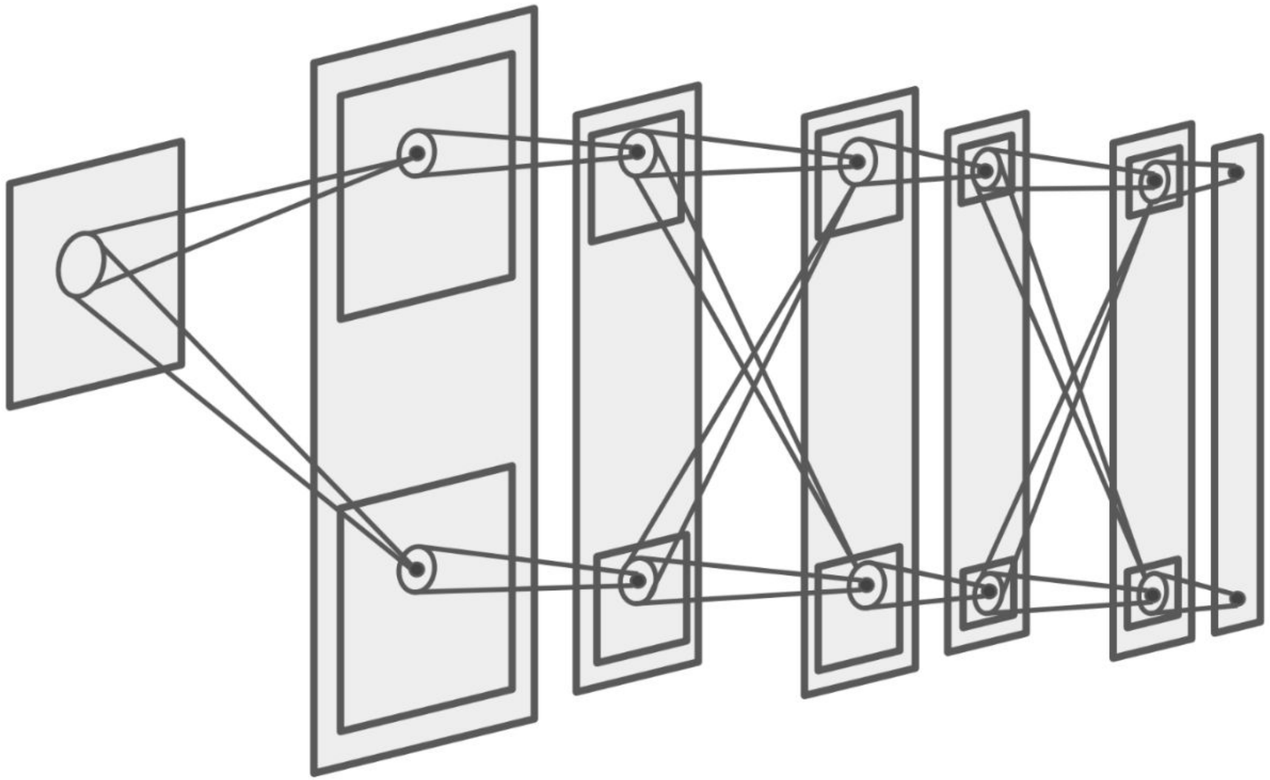


No response

Response (end point)

# Neurocognitron

**Neocognitron**
*[Fukushima 1980]*

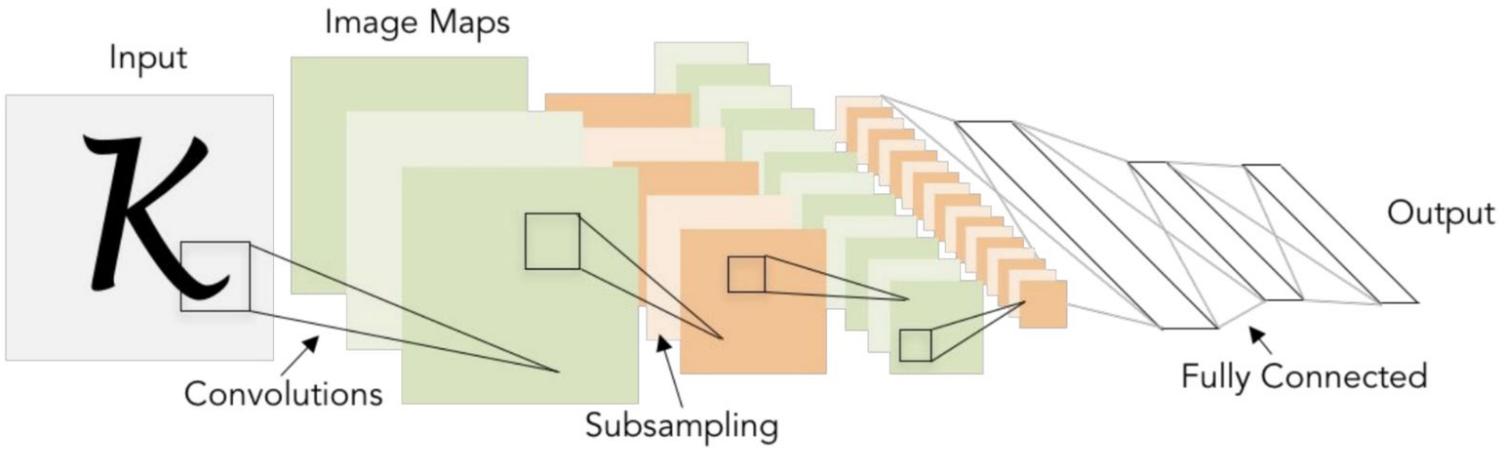


"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

# History

**Gradient-based learning applied to document recognition**

*[LeCun, Bottou, Bengio, Haffner 1998]*




LeNet-5

# History

## ImageNet Classification with Deep Convolutional Neural Networks

*[Krizhevsky, Sutskever, Hinton, 2012]*

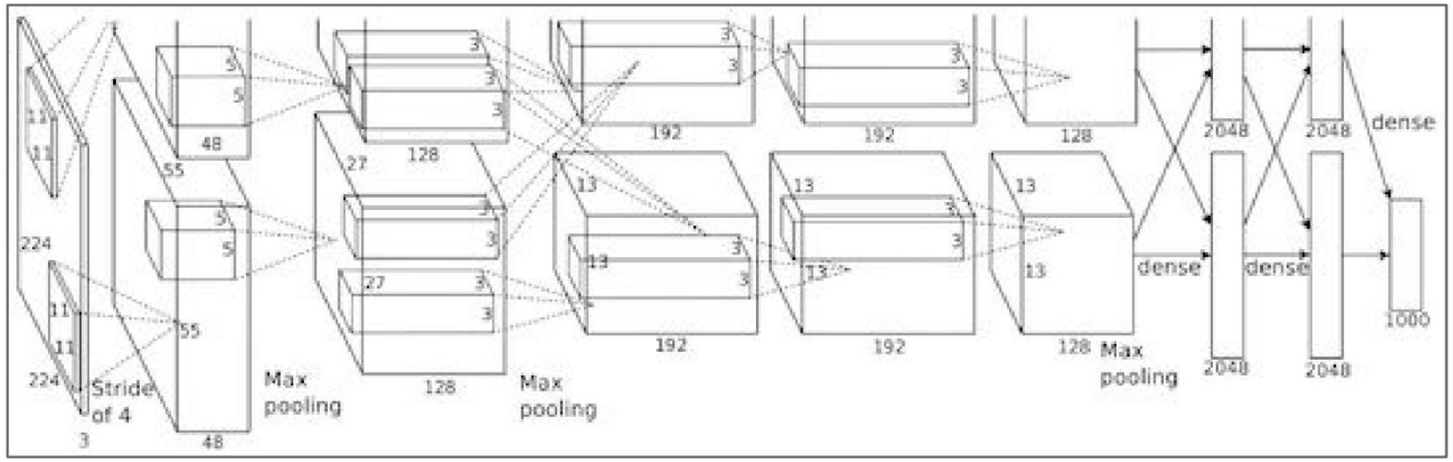
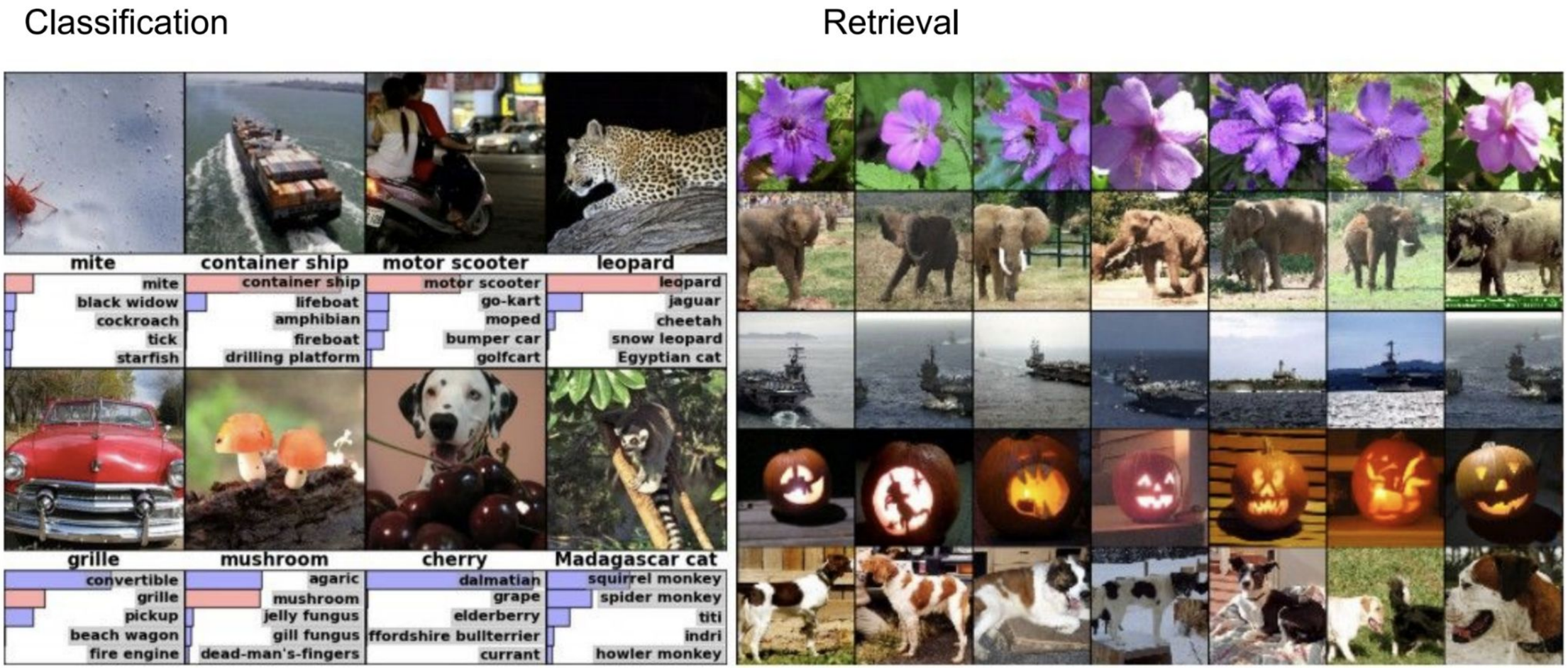



Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

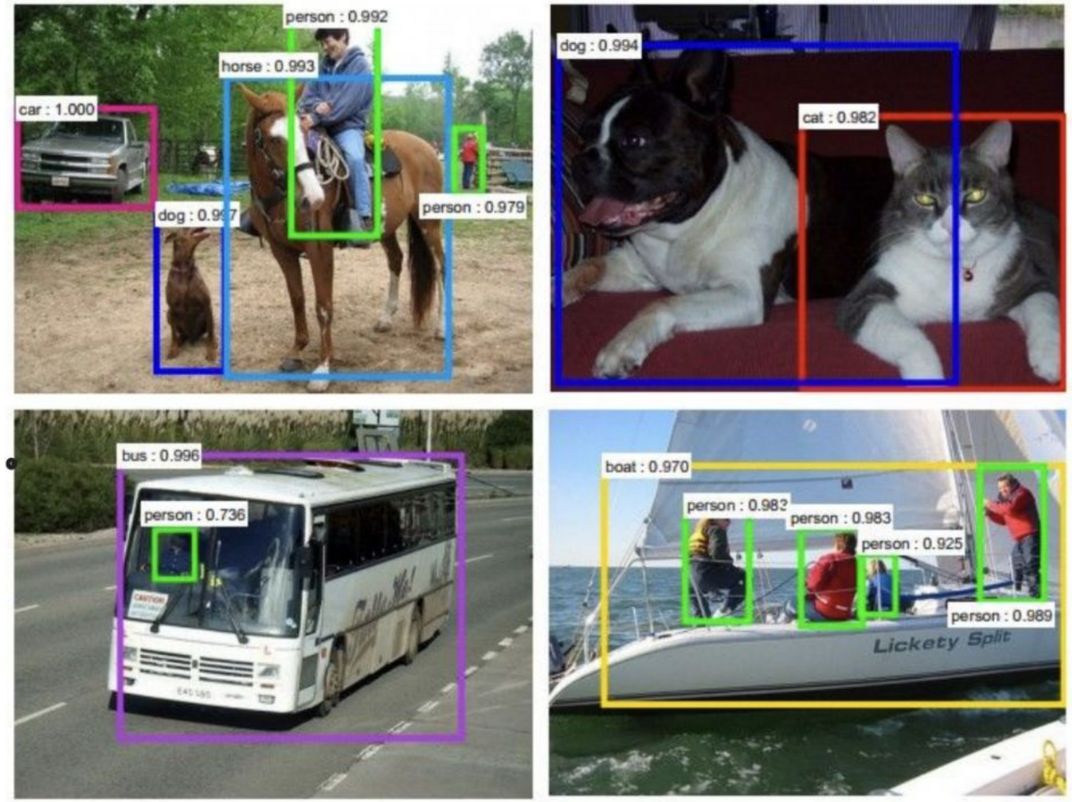
# Today: CNNs are everywhere




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

# Today: ConvNets are everywhere

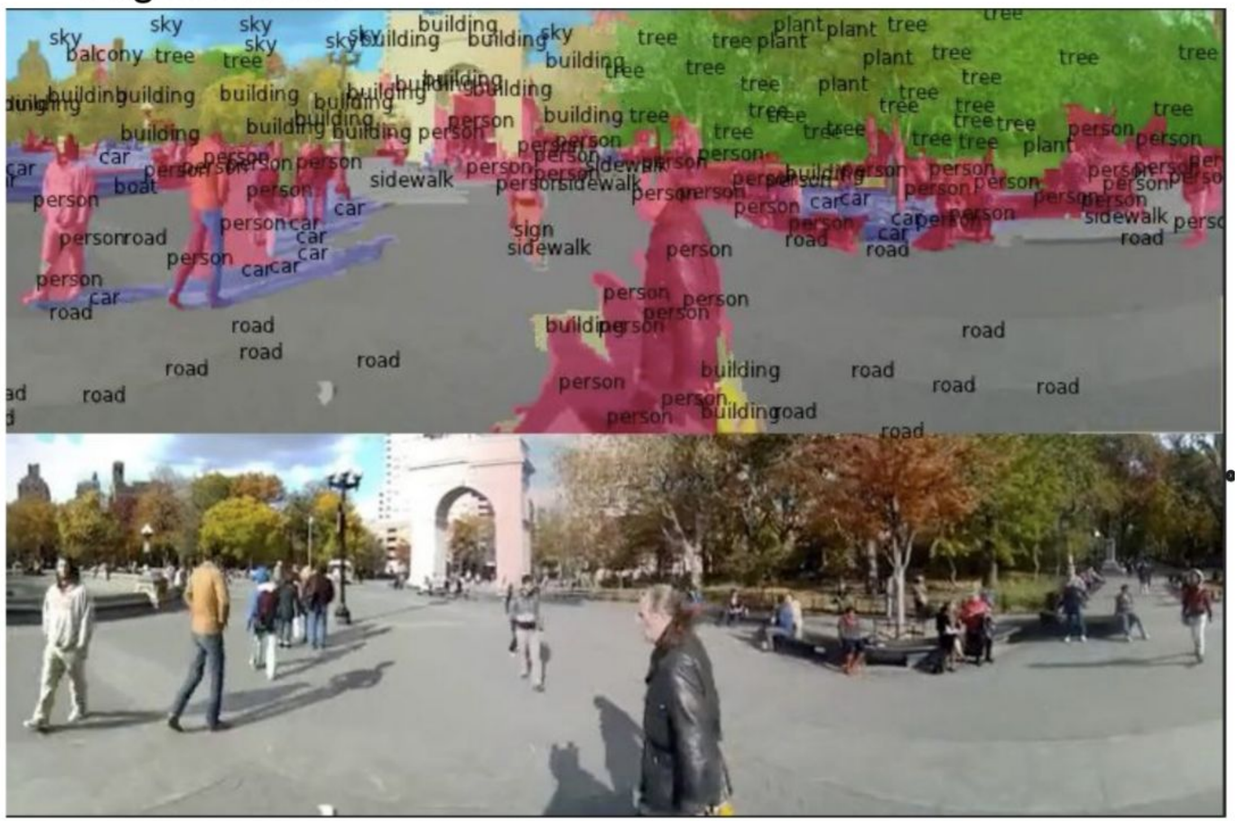
Detection




Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

*[Faster R-CNN: Ren, He, Girshick, Sun 2015]*

Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

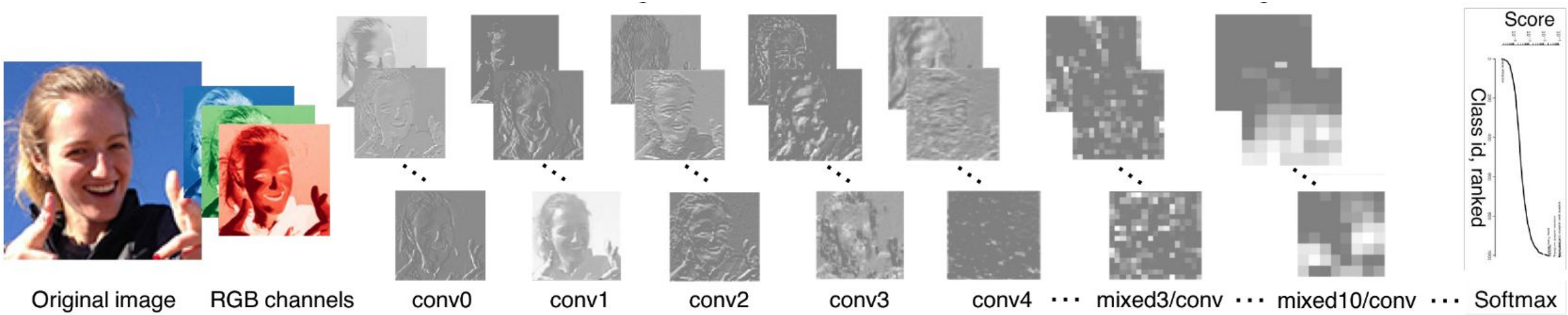
*[Farabet et al., 2012]*

# Self-Driving Cars


Photo by Lane McIntosh. Copyright CS231n 2017.

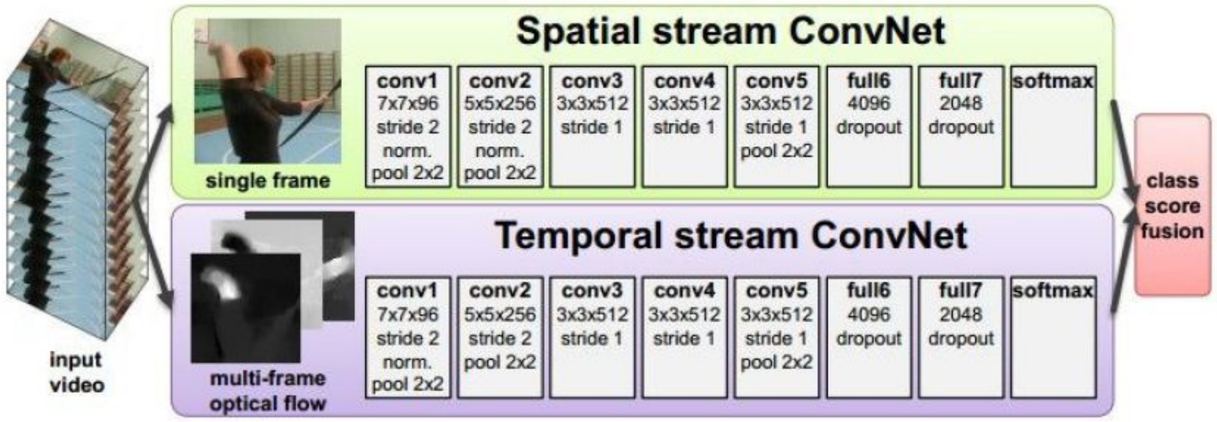
self-driving cars

# Face-Recognition & Video Classification




*[Taigman et al. 2014]*

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.




*[Simonyan et al. 2014]*

Figures copyright Simonyan et al., 2014. Reproduced with permission.

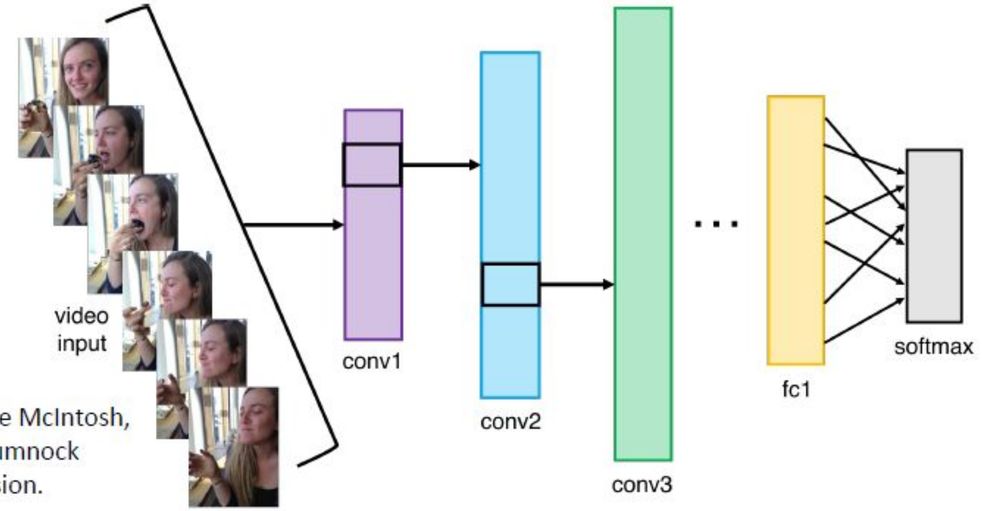
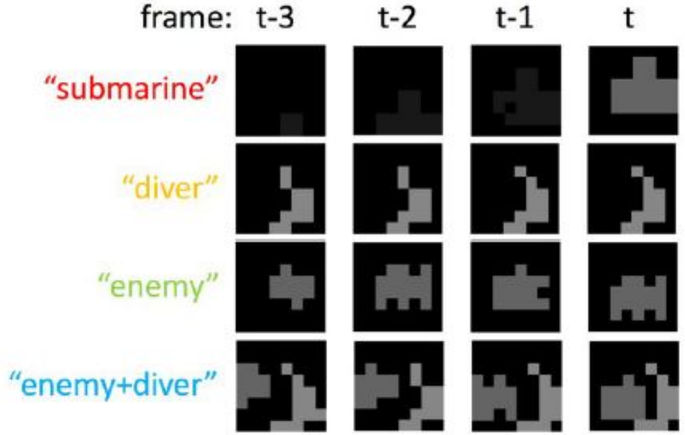
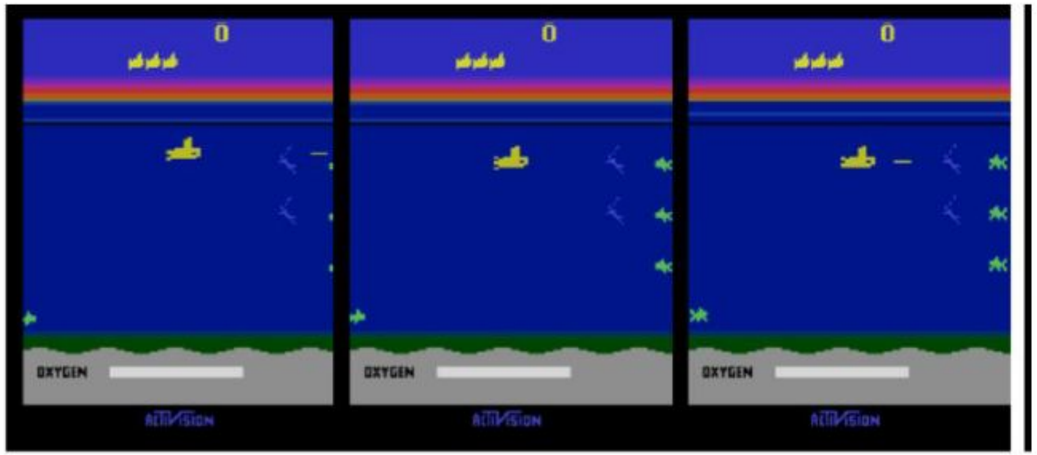



Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

# Reinforcement Learning

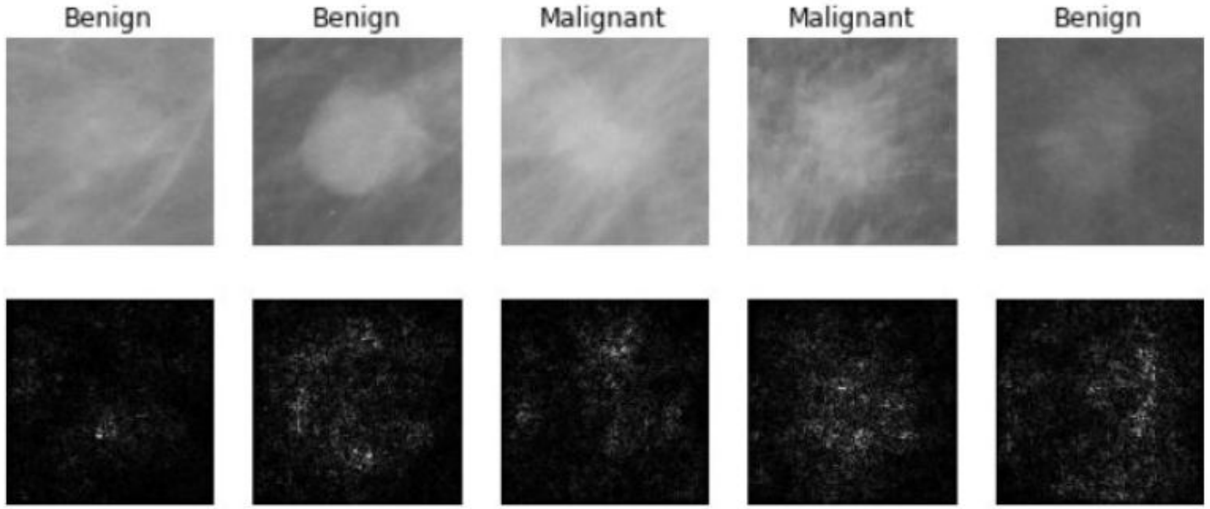







*[Guo et al. 2014]*

# More Applications




*[Levy et al. 2016]*



*[Dieleman et al. 2014]*

# More Applications

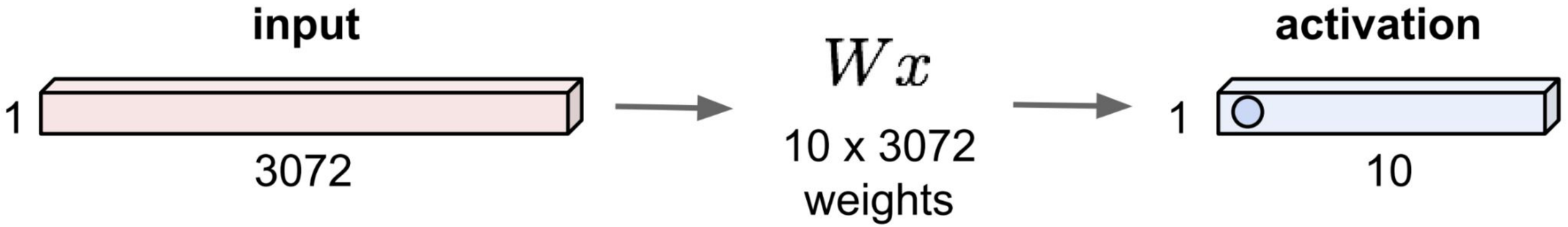
Image Style Transfer Using Convolutional Neural Networks

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge

# Convolutional Neural Networks

# Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

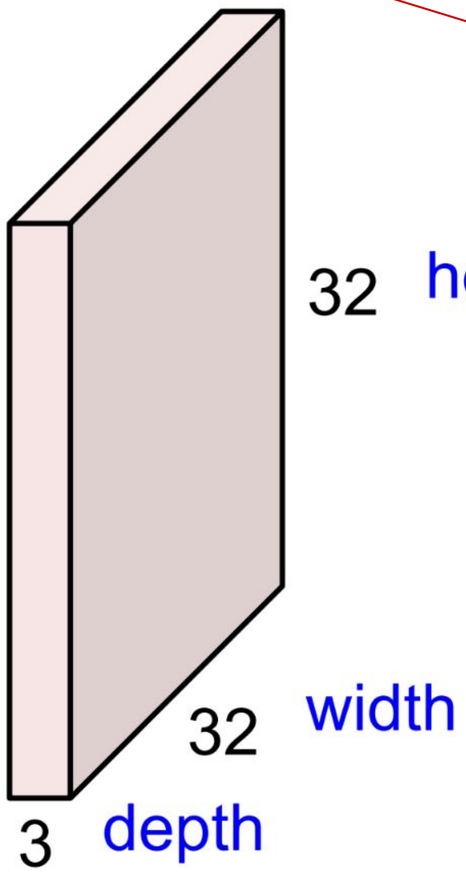



***1 number:***
*the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)*

# Convolution Layer

32x32x3 image -> preserve spatial structure

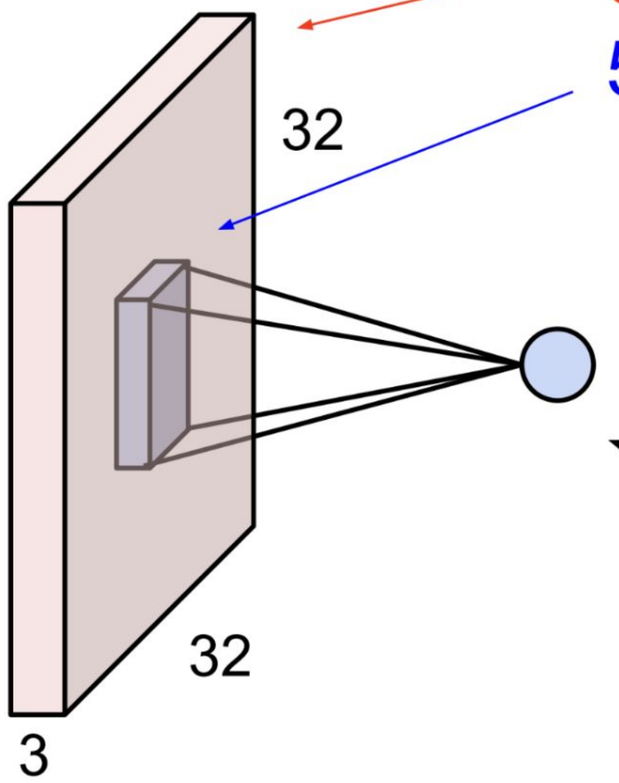
*Filters always extend the full depth of the input volume*




5x5x3 filter

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

# Convolution Layer



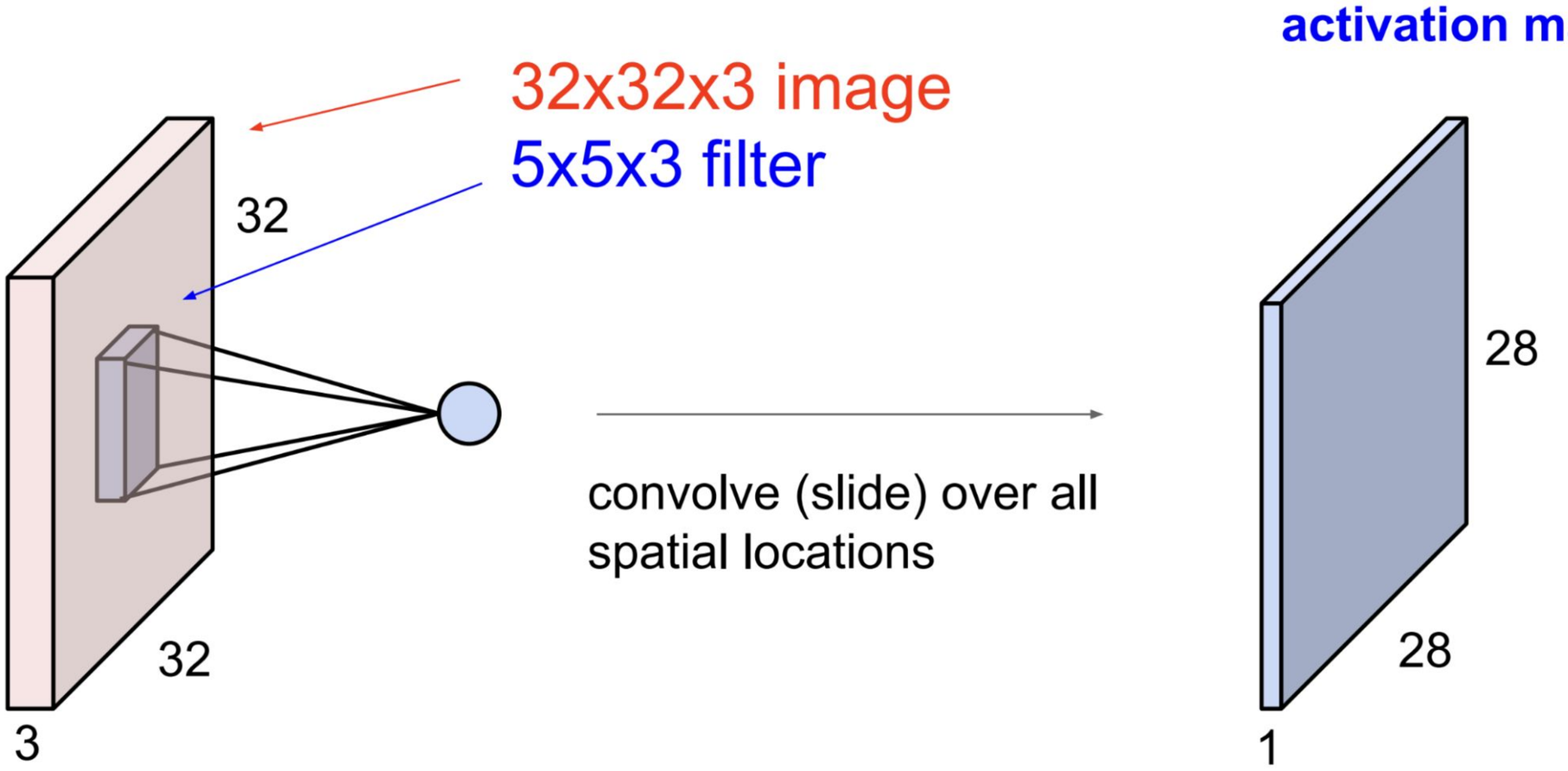

32x32x3 image

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)

$$w^T x + b$$

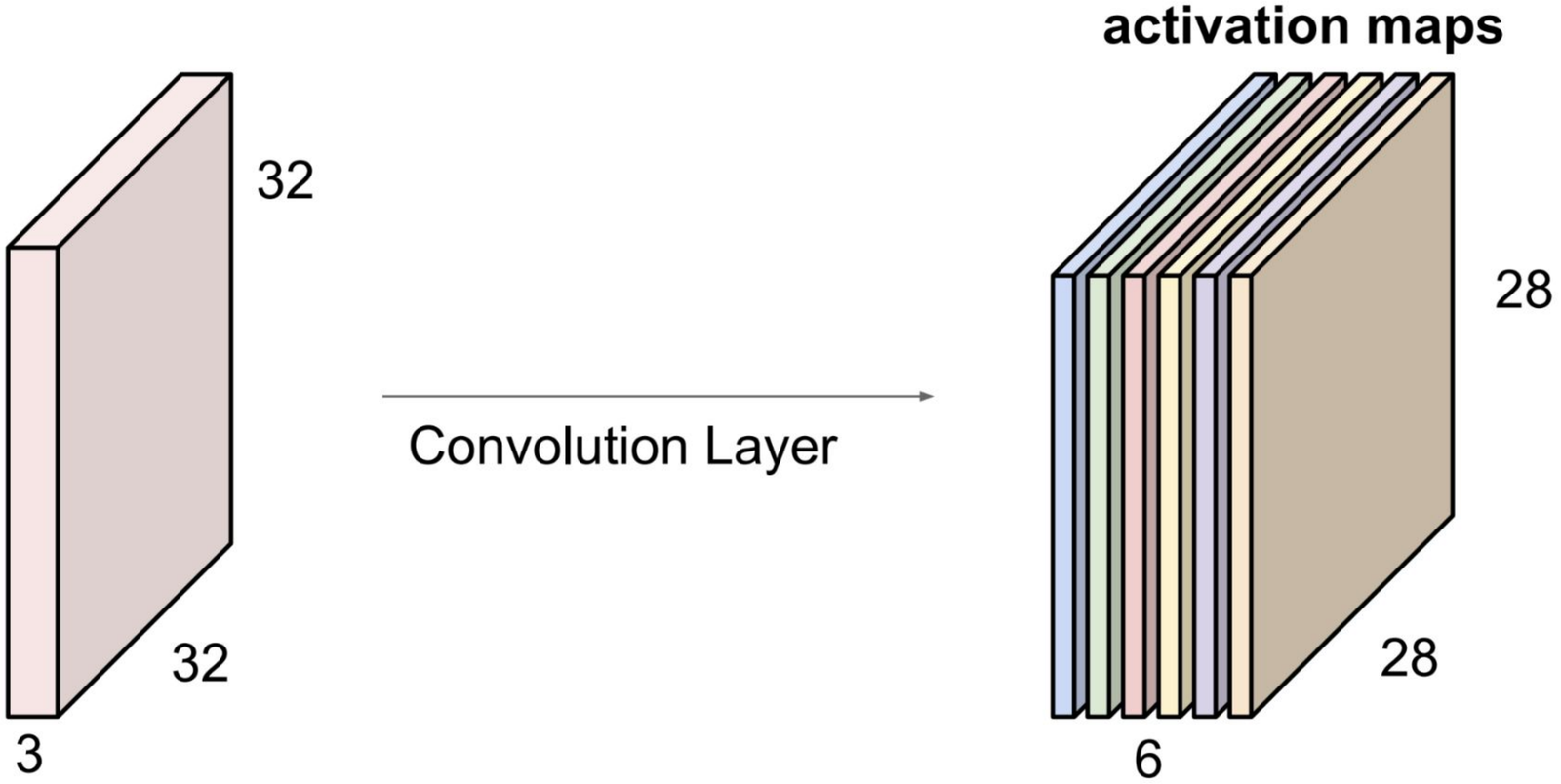
# Convolution Layer



# Convolution Layer

# Convolution Layer

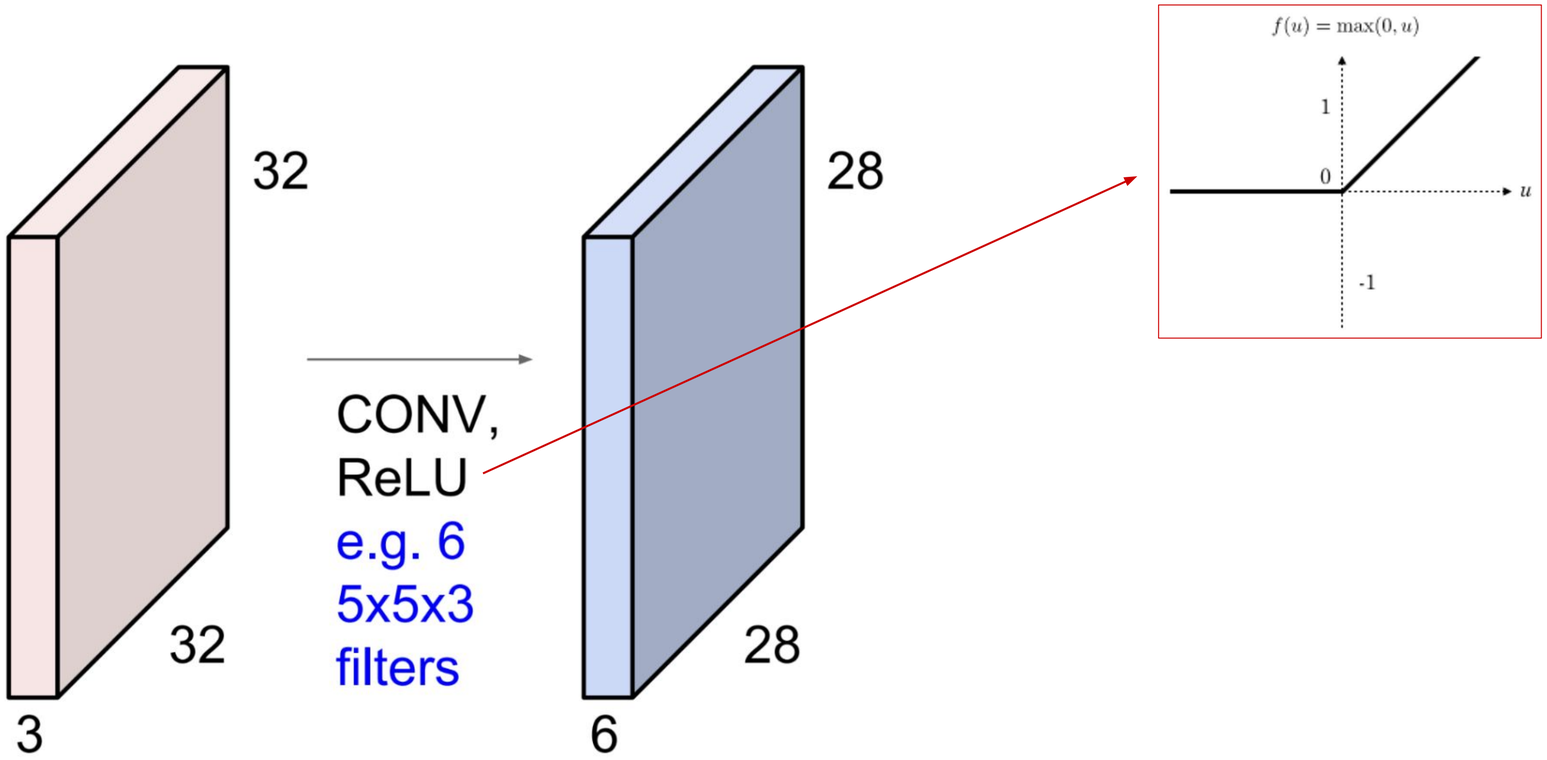
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:




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

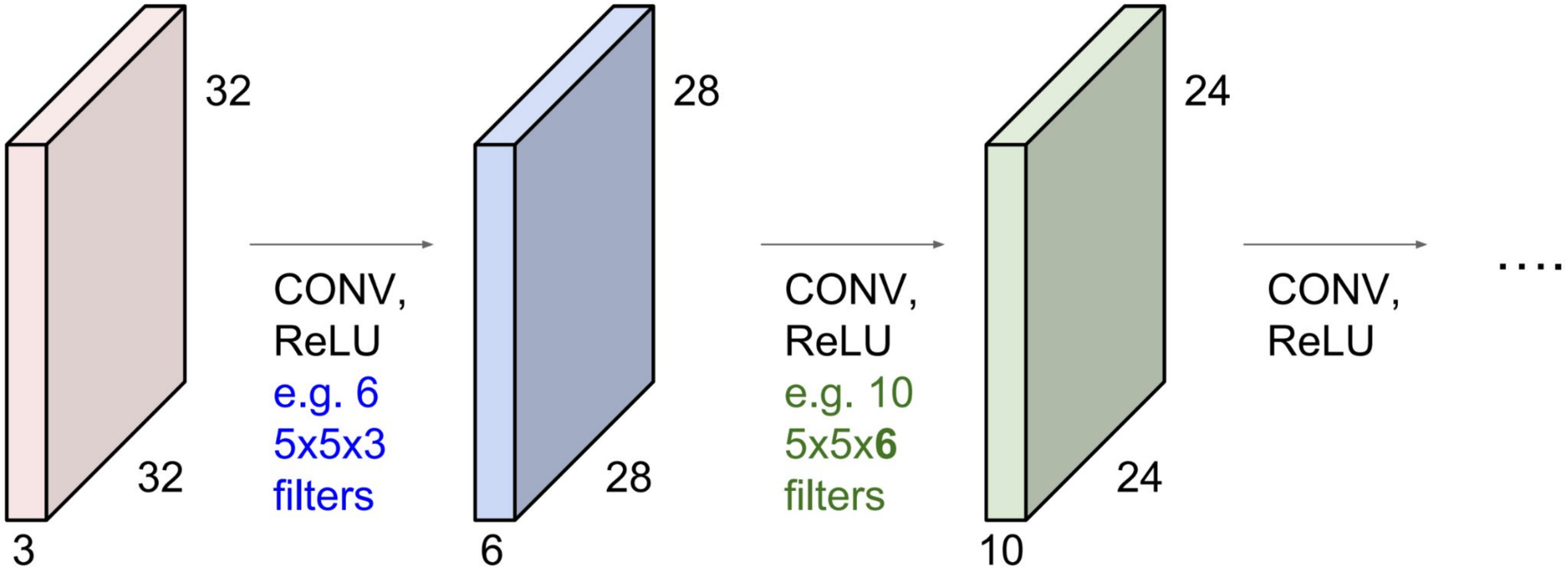
# Convolutional Nets

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

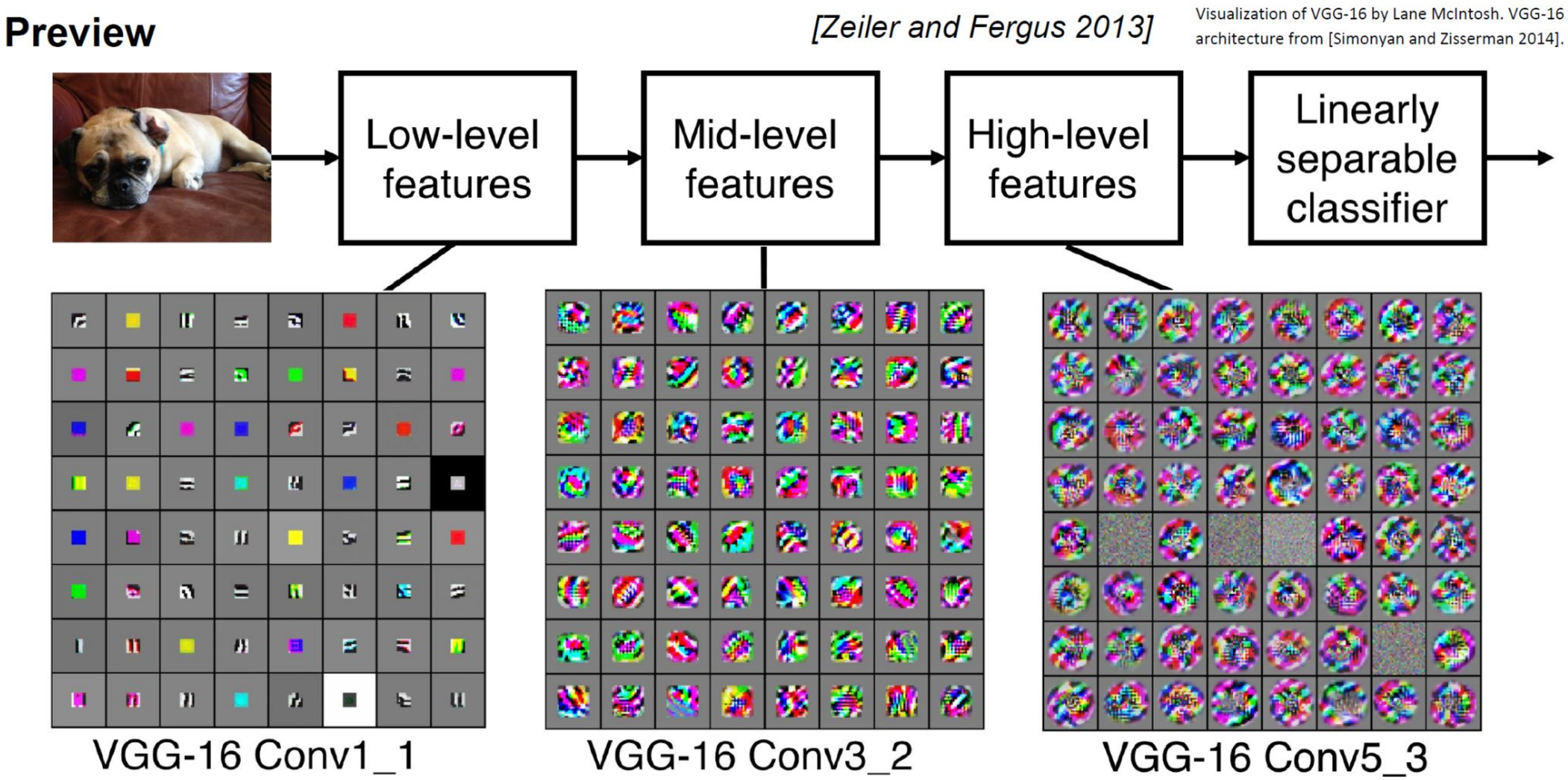


# Convolutional Nets

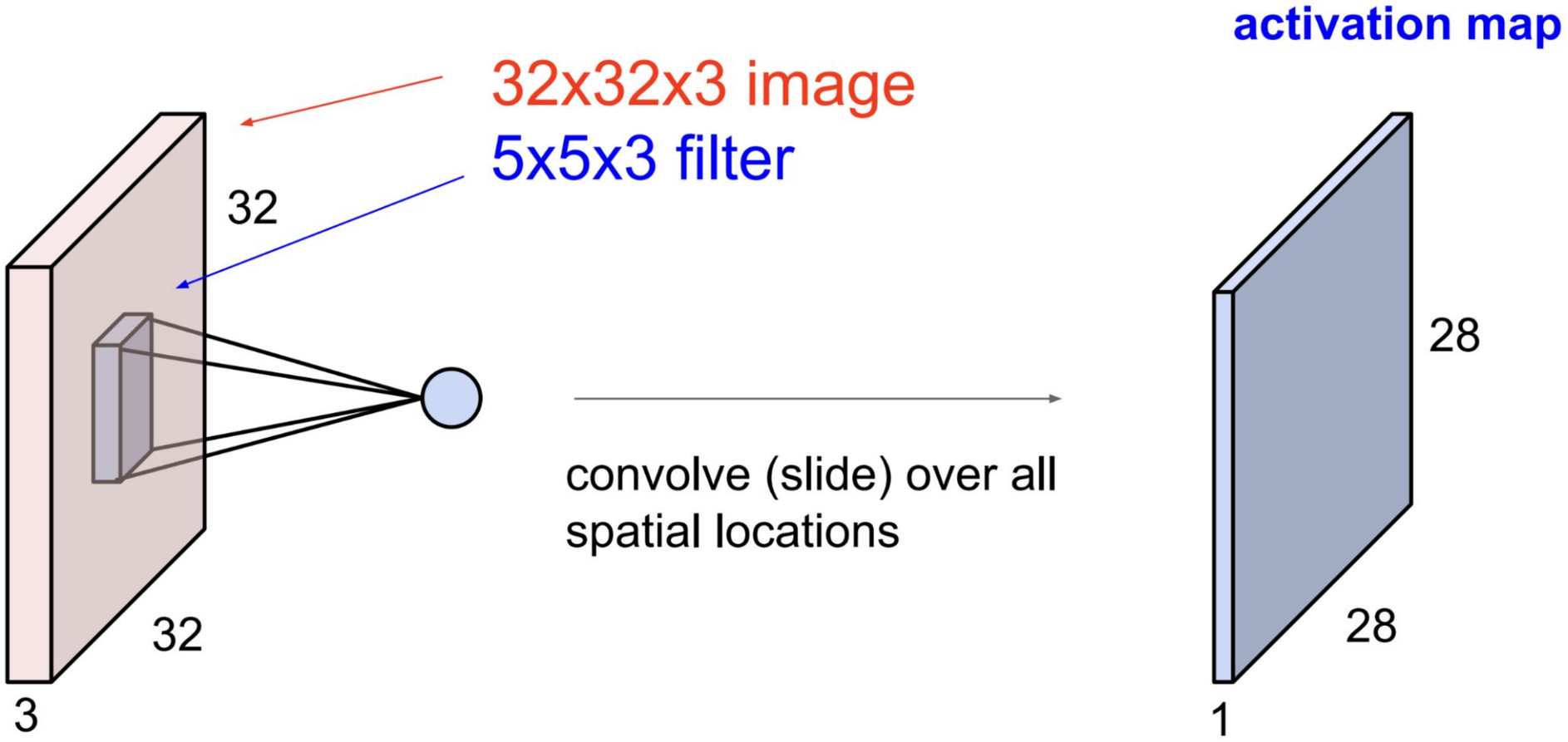
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



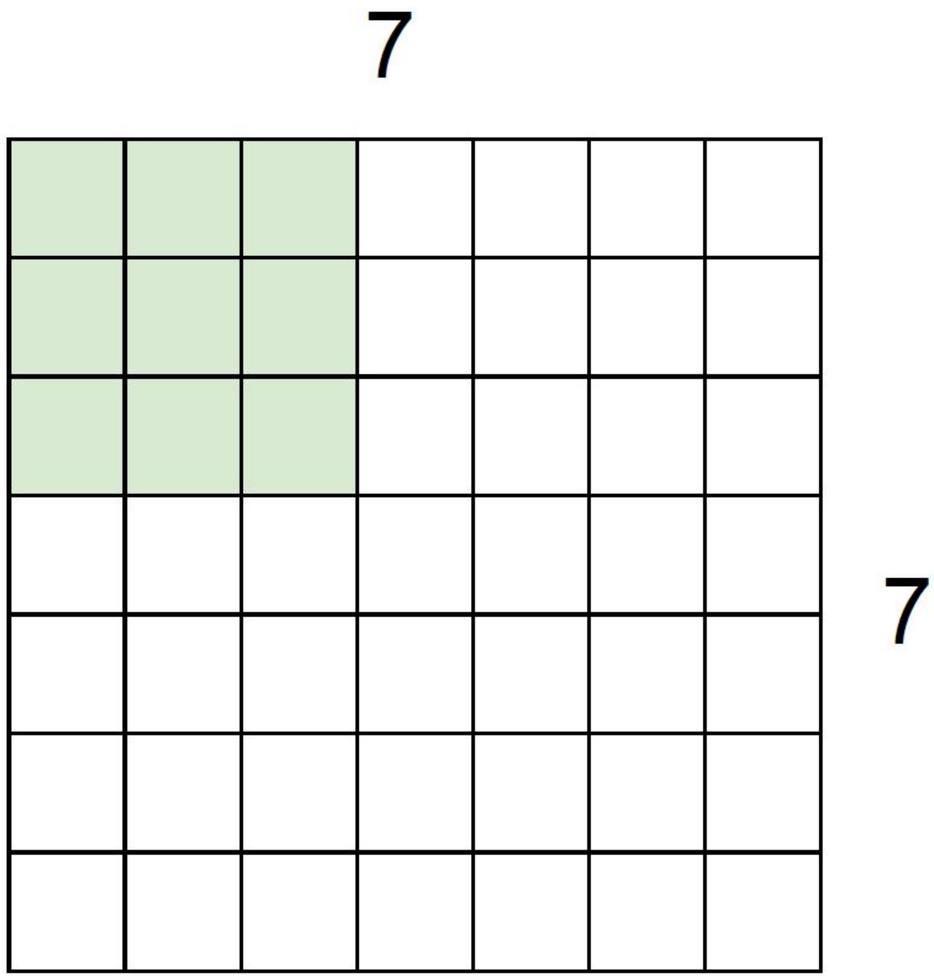
# Convolutional Nets



# A closer look at spatial dimensions

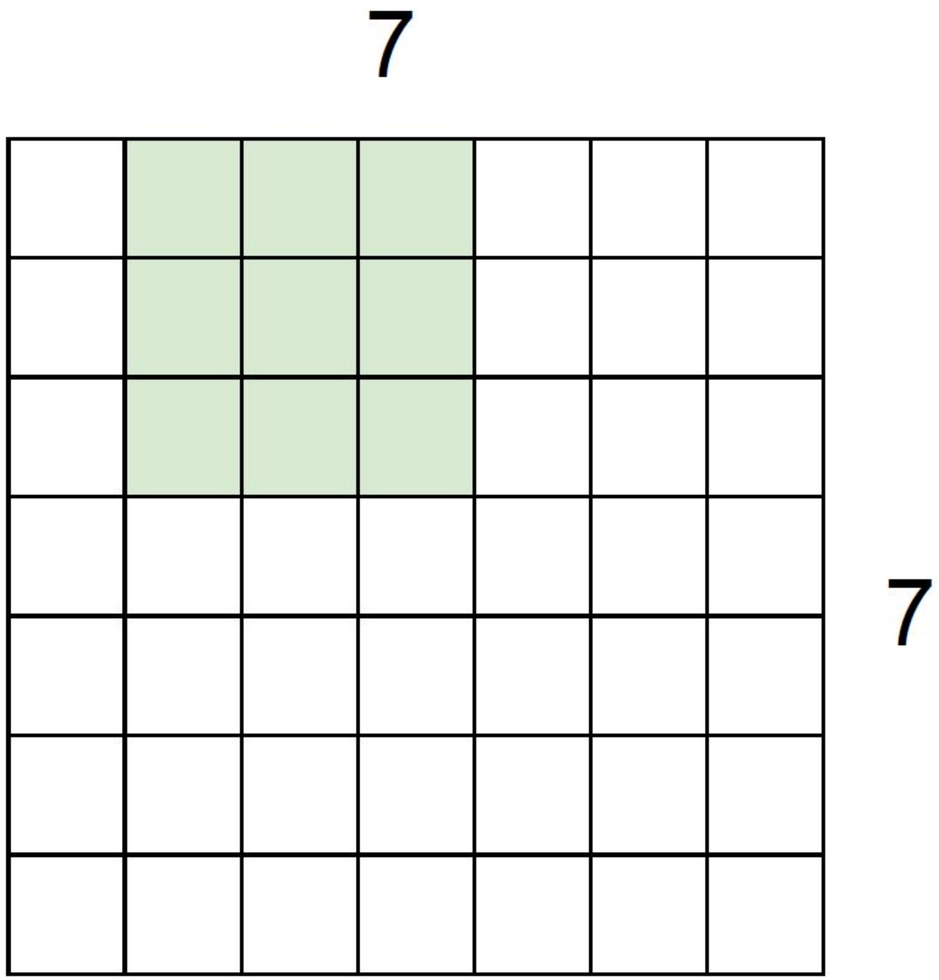


# A closer look at spatial dimensions



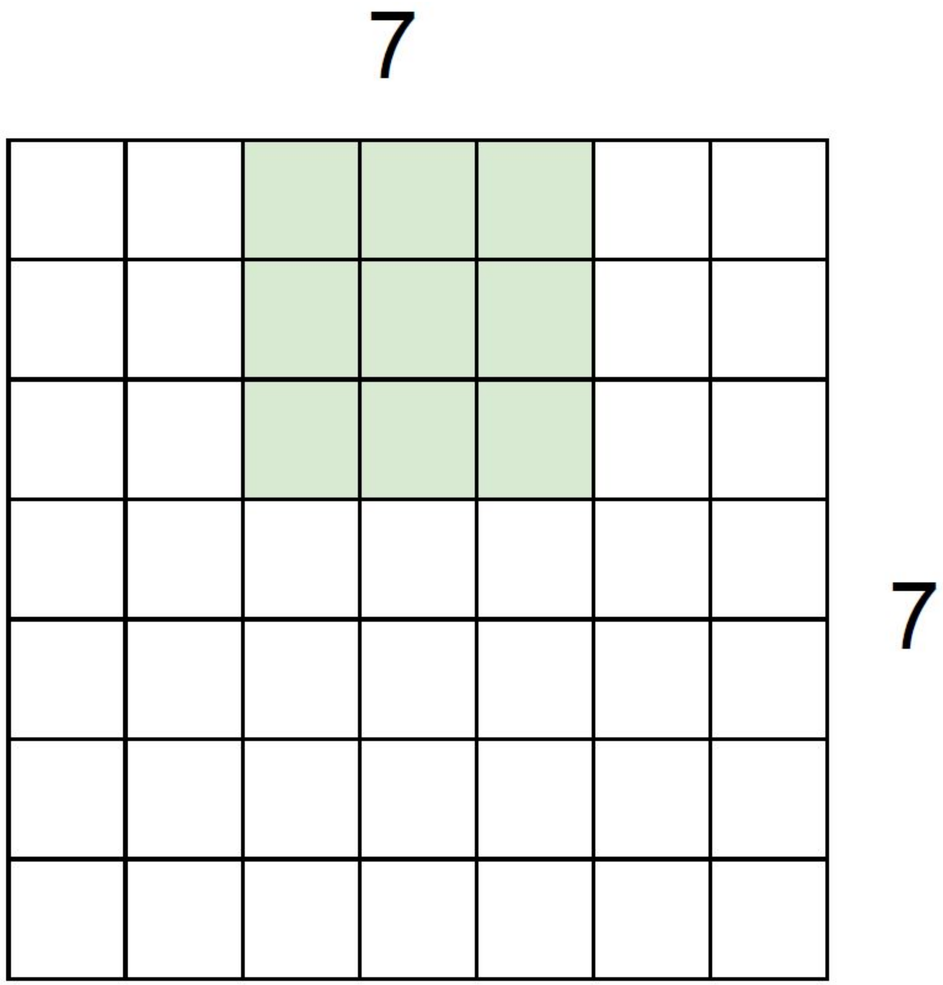

7x7 input (spatially)
assume 3x3 filter

# A closer look at spatial dimensions



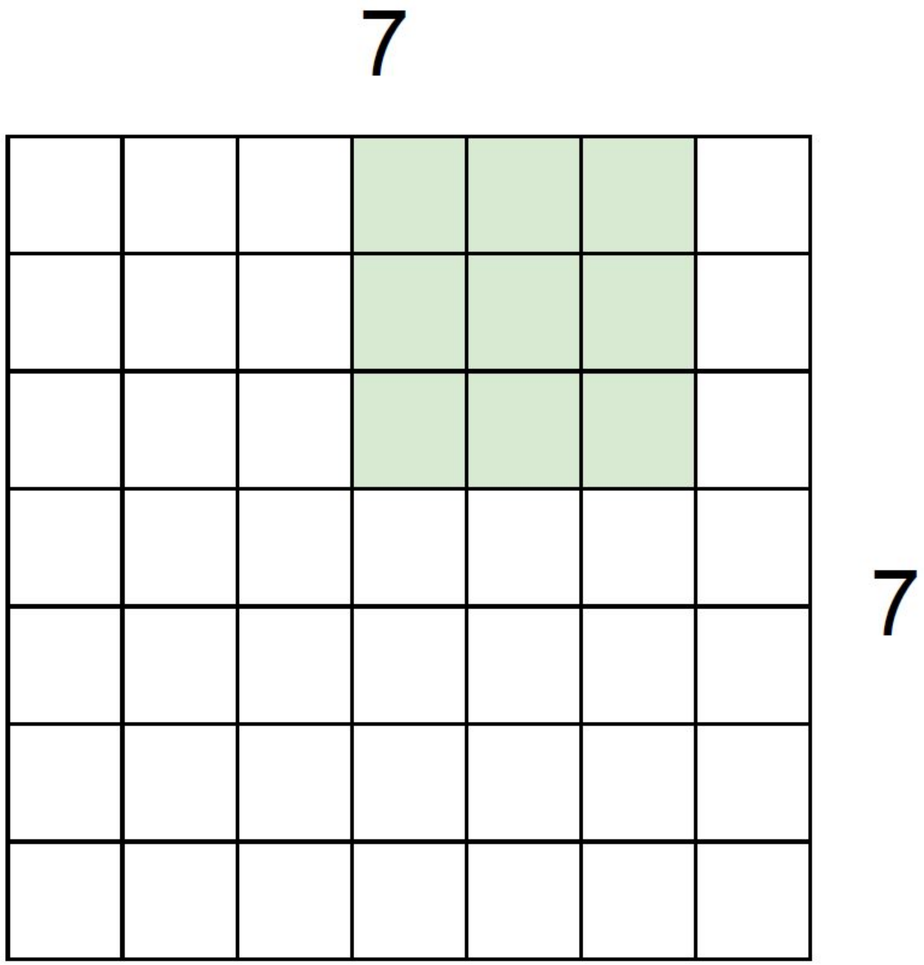

7x7 input (spatially)
assume 3x3 filter

# A closer look at spatial dimensions



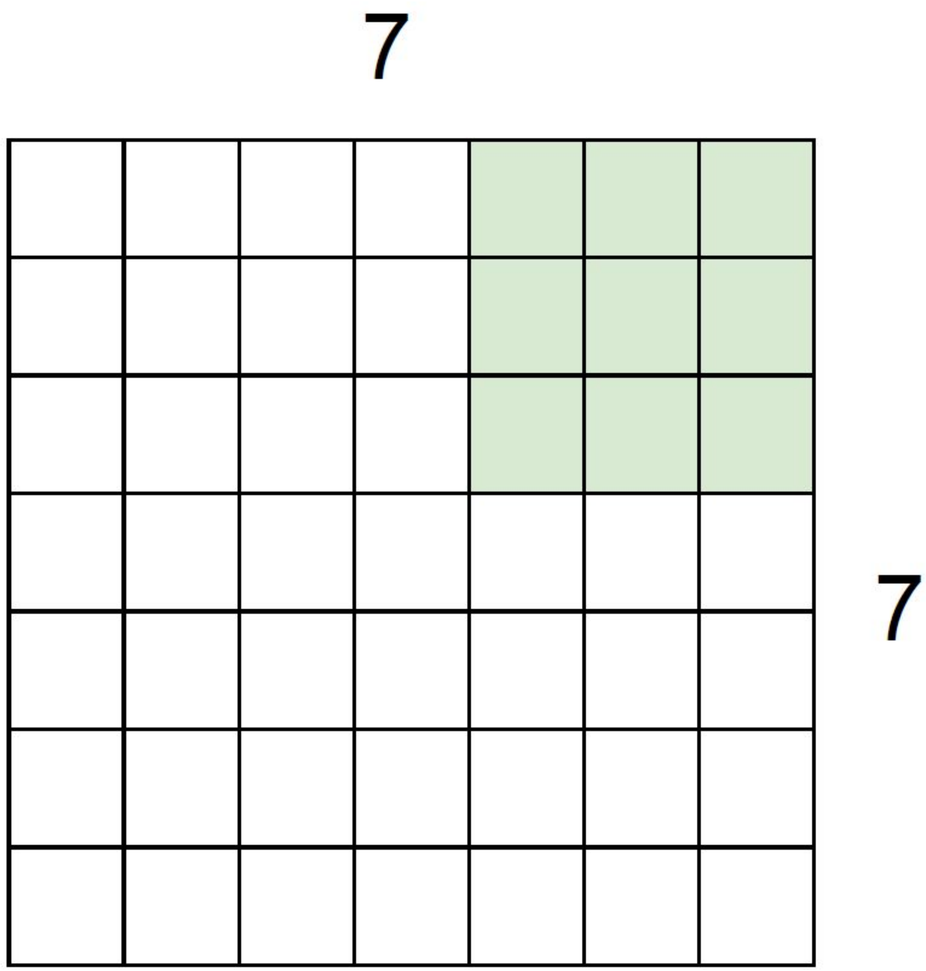

7x7 input (spatially)
assume 3x3 filter

# A closer look at spatial dimensions




7x7 input (spatially)
assume 3x3 filter

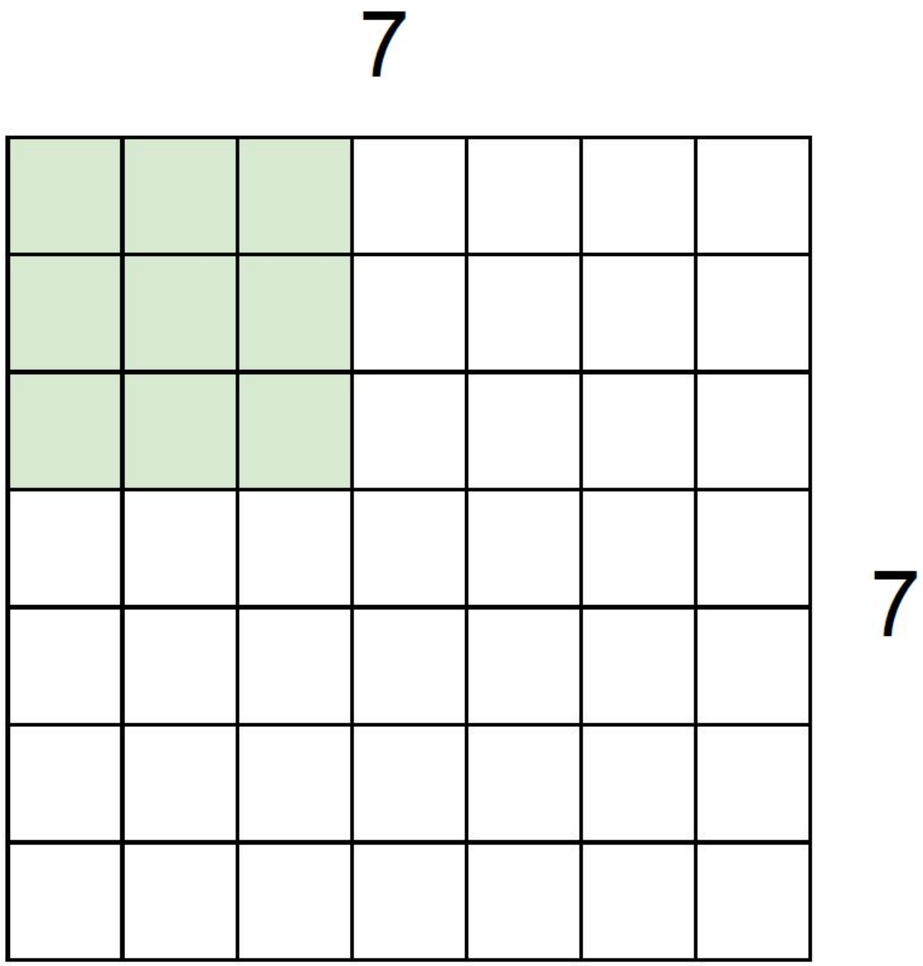
# A closer look at spatial dimensions




7x7 input (spatially)
assume 3x3 filter

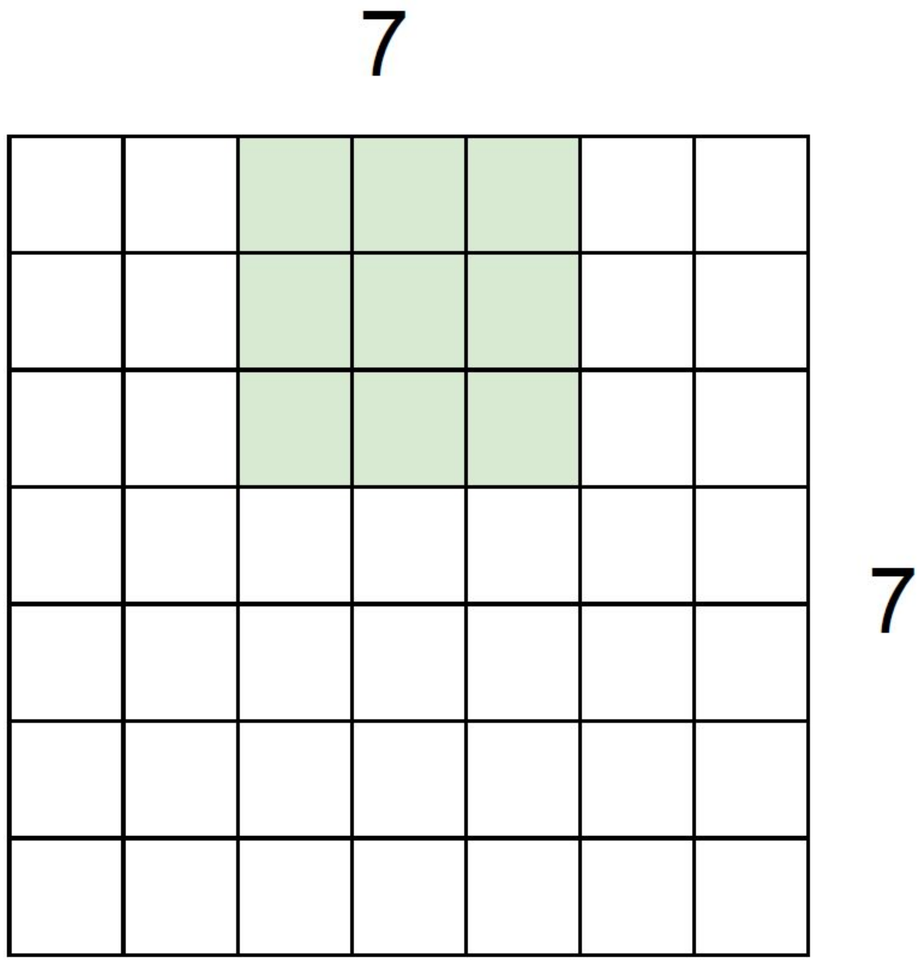
**=> 5x5 output**

# A closer look at spatial dimensions



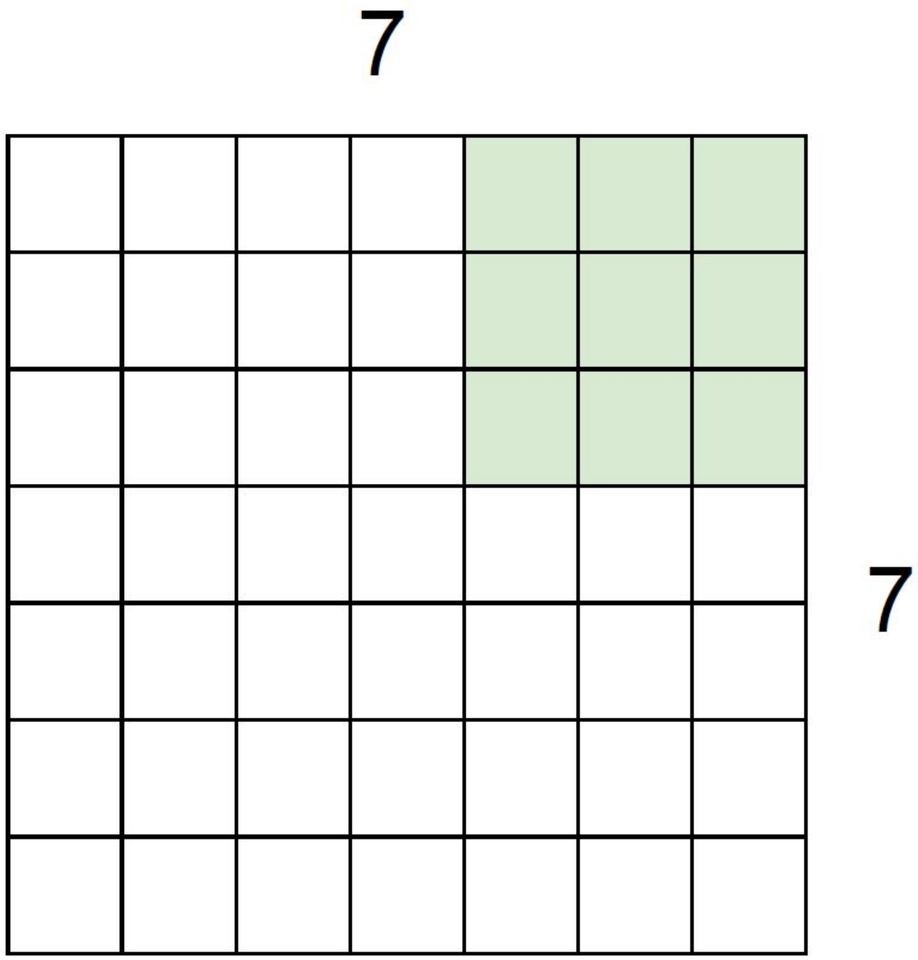

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

# A closer look at spatial dimensions



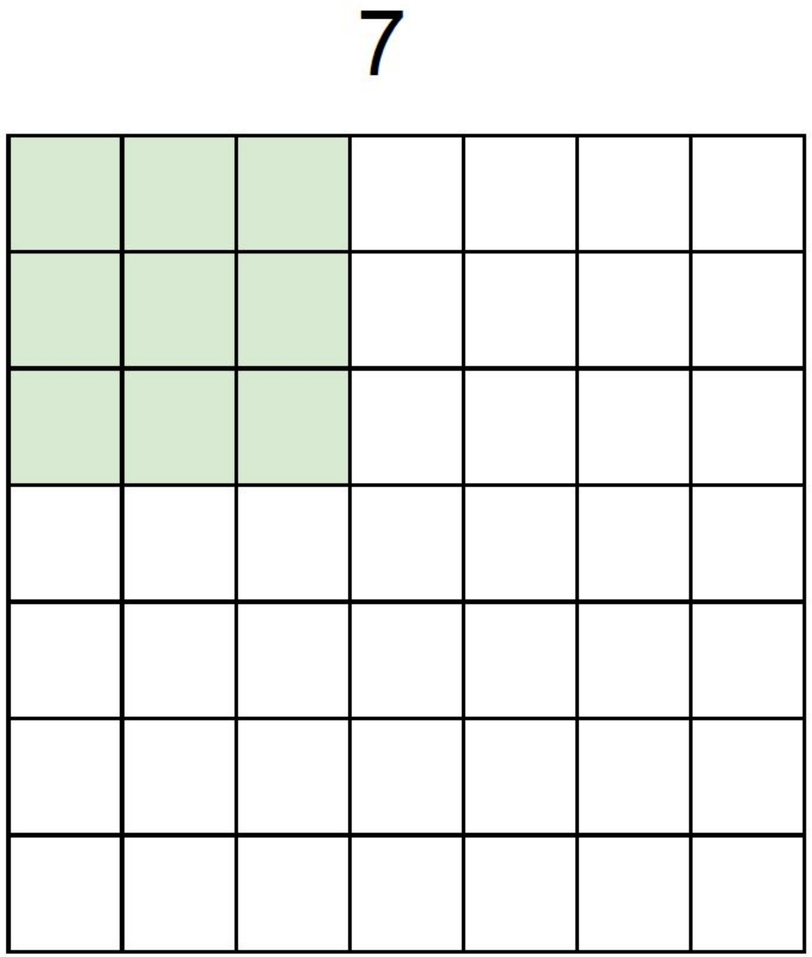

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

# A closer look at spatial dimensions




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

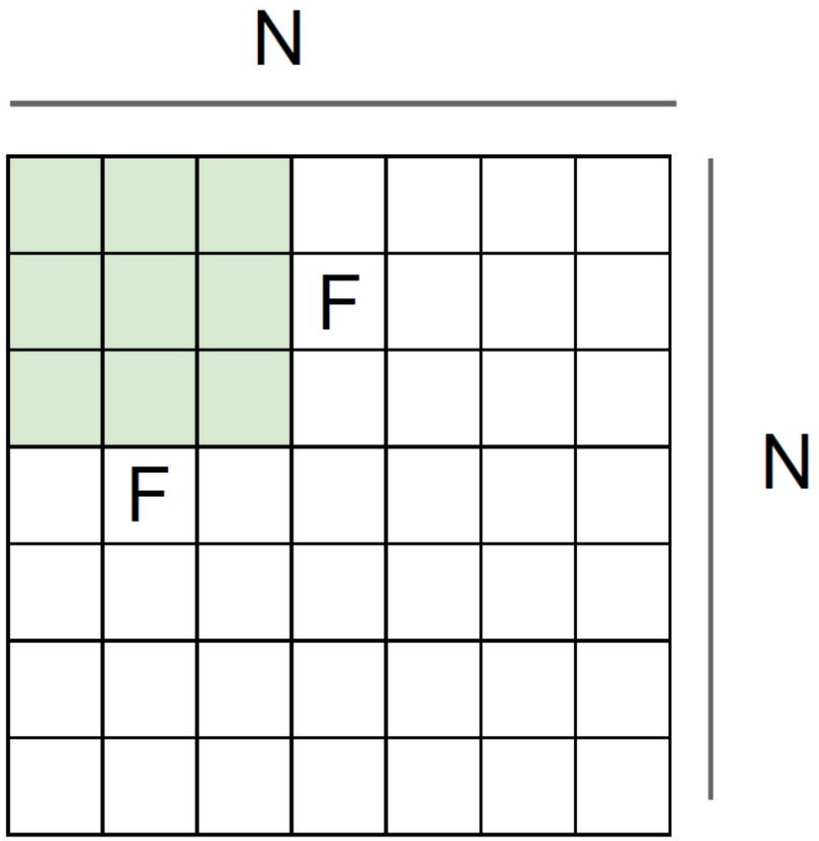
# A closer look at spatial dimensions




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

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

# A closer look at spatial dimensions




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

# A closer look at spatial dimensions

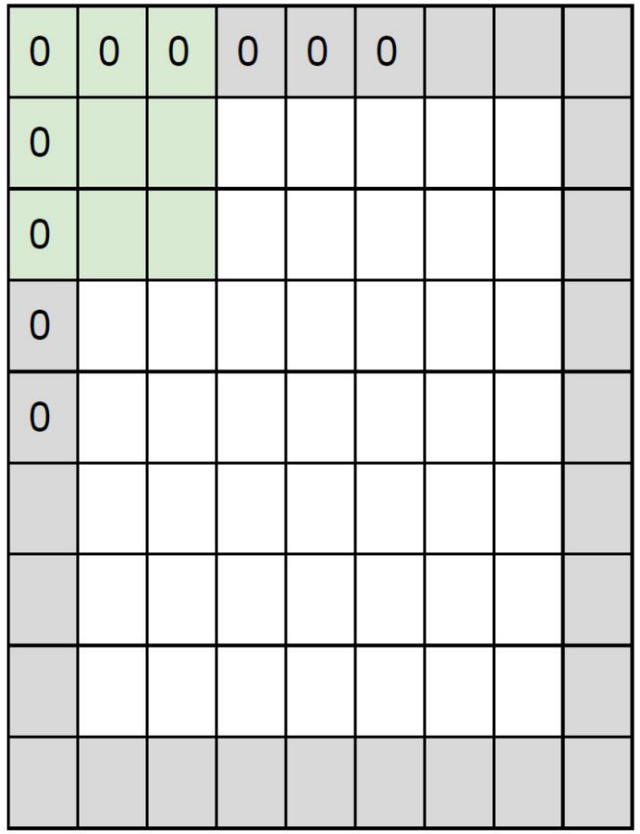
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

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

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

# A closer look at spatial dimensions

## In practice: Common to zero pad the border



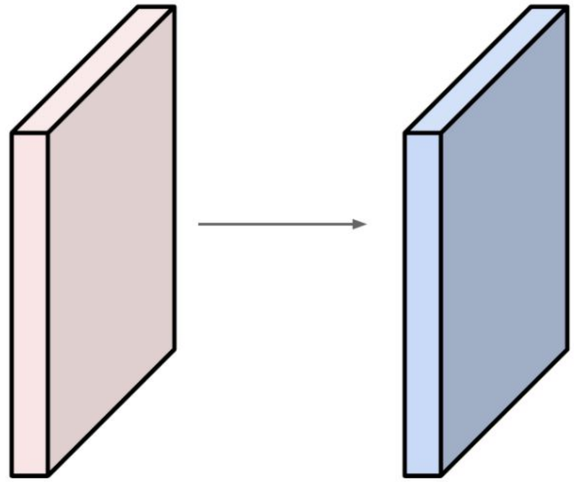

e.g. input 7x7
**3x3** filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

**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

# A closer look at spatial dimensions

Examples time:

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

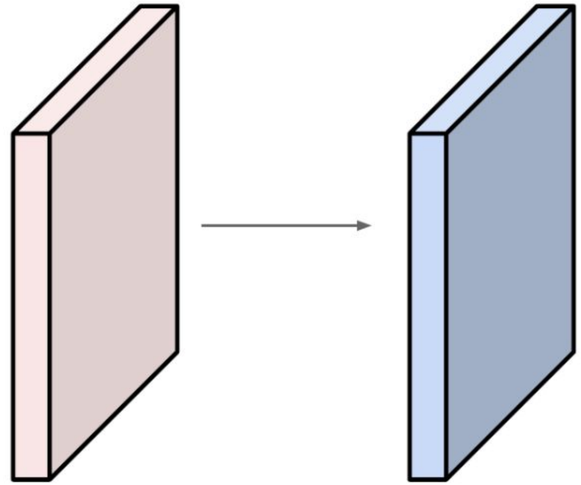


Output volume size: ?

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

# A closer look at spatial dimensions

Examples time:

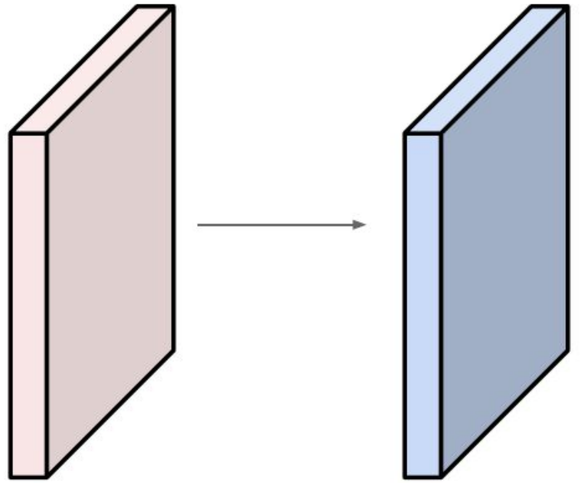


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

# A closer look at spatial dimensions

Examples time:

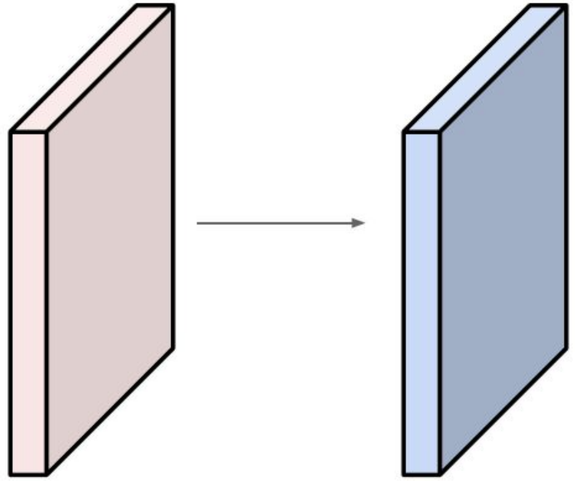


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

# A closer look at spatial dimensions

Examples time:

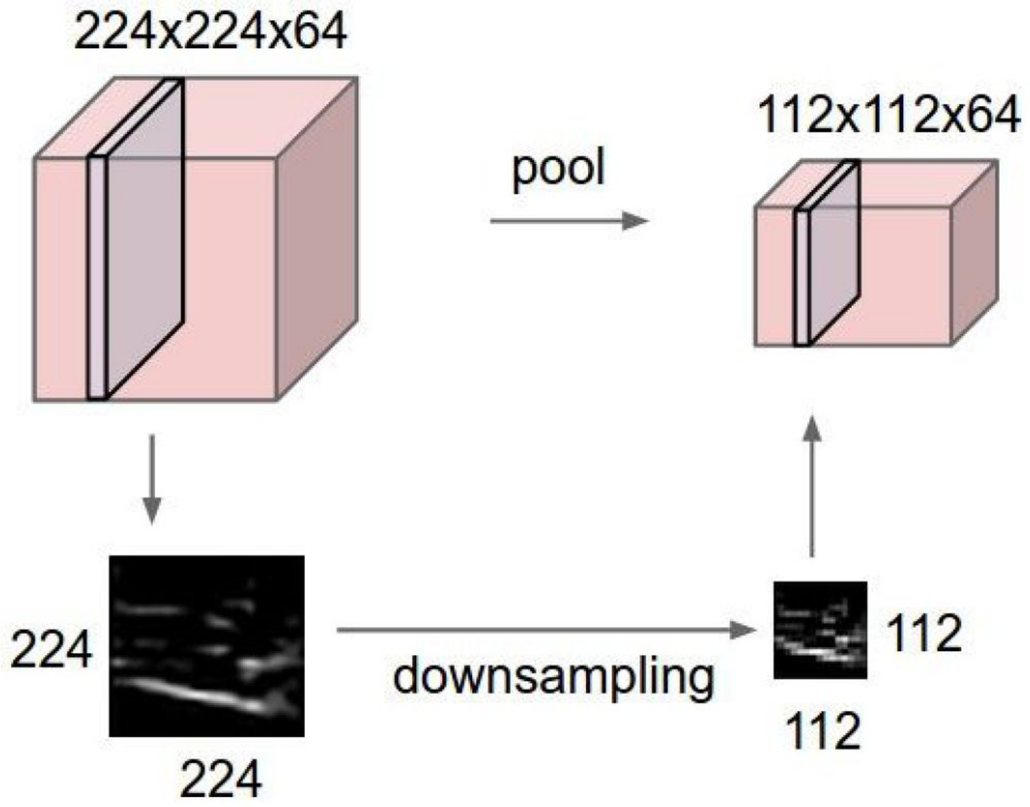


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

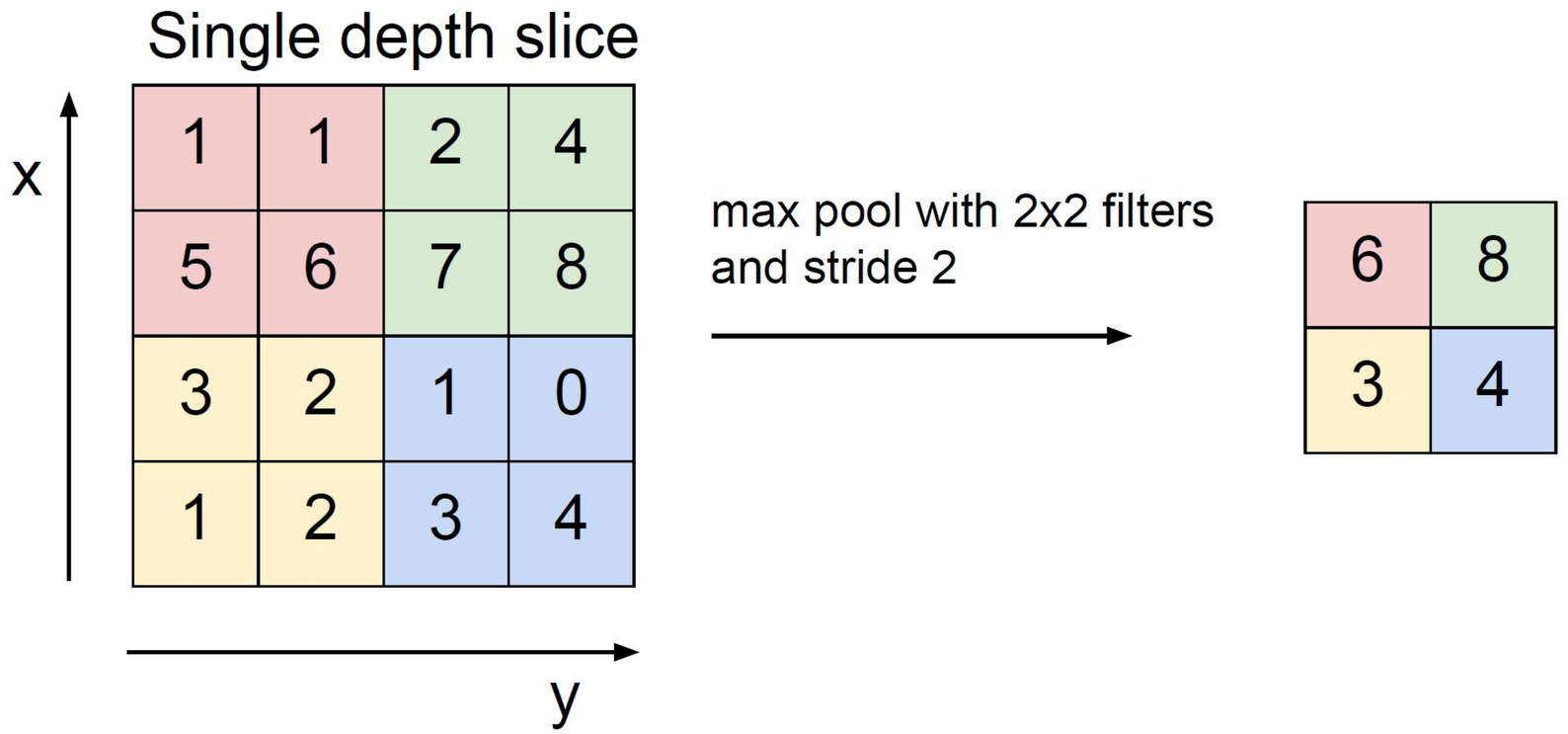
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

# Pooling layer

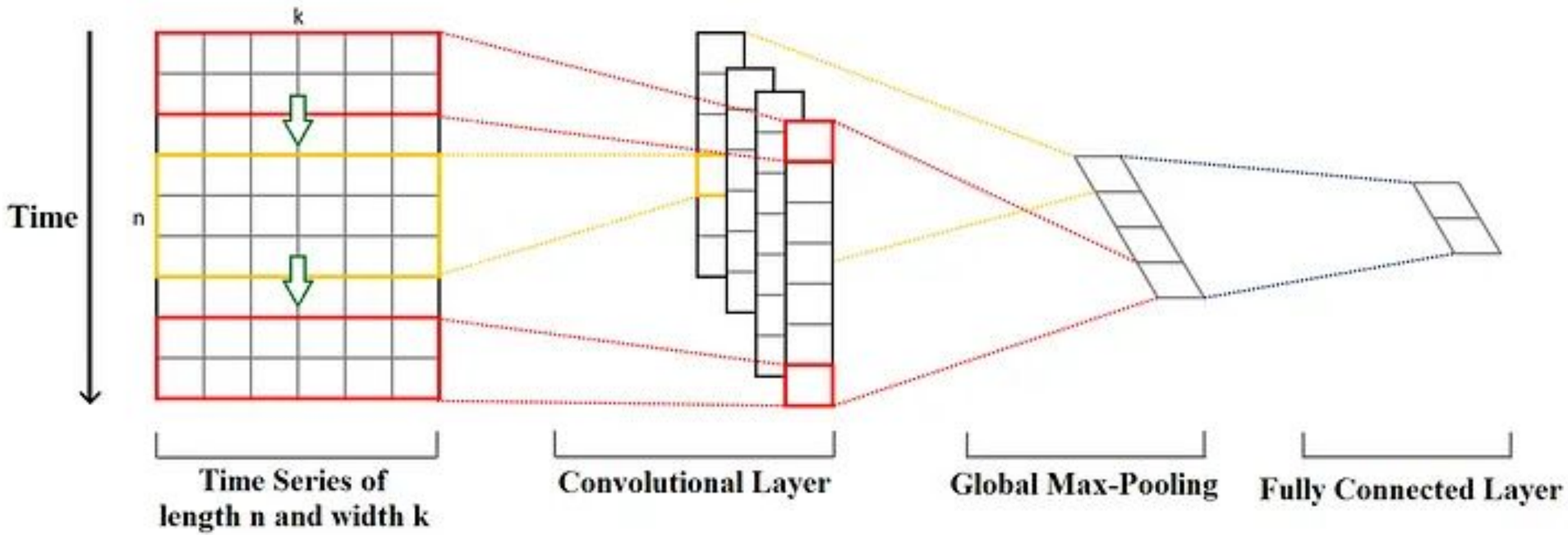
- makes the representations smaller and more manageable
- operates over each activation map independently:



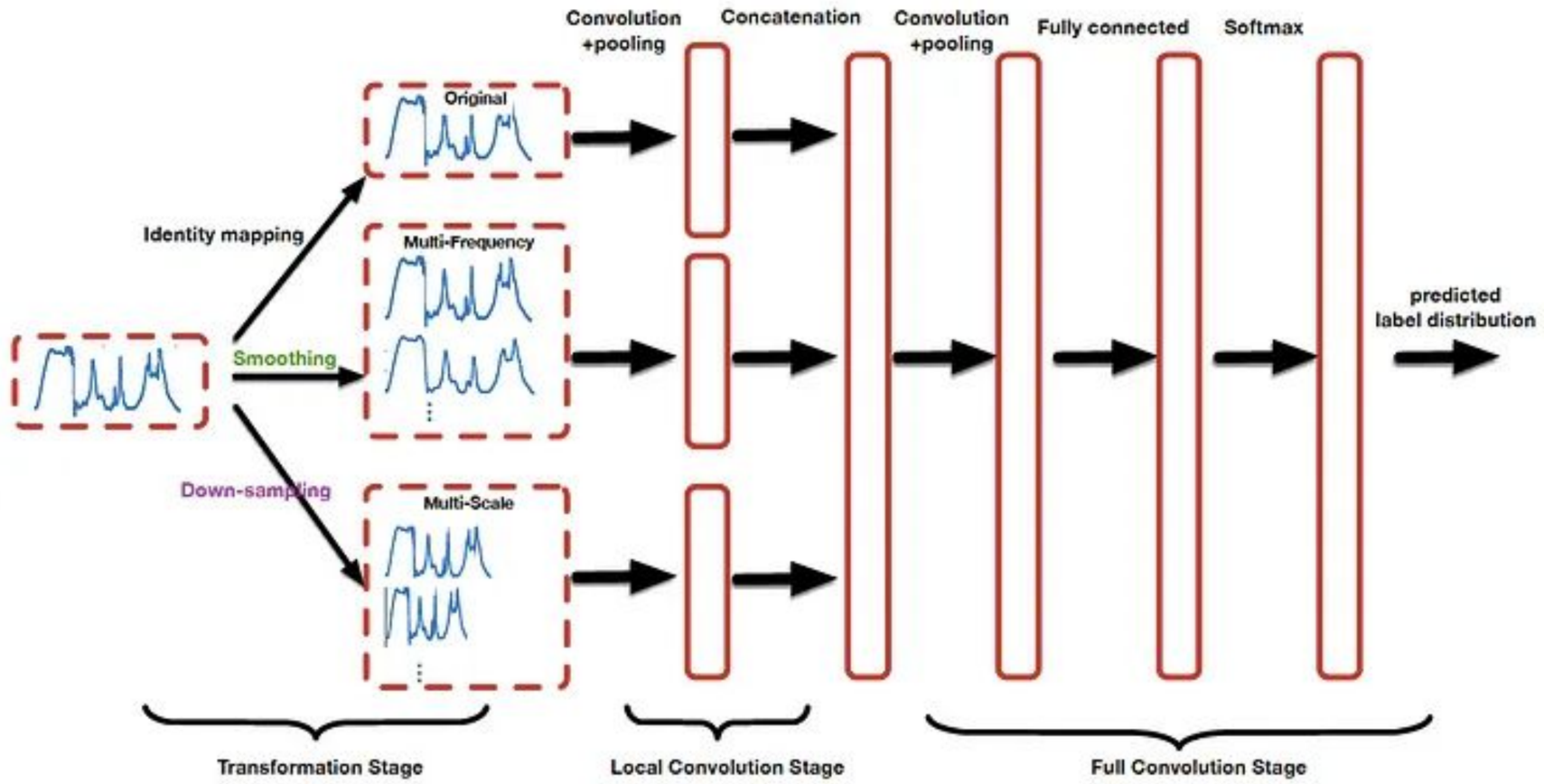
# Max Pooling


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

# 1-D Convolution for Time Series



# 1-D Convolution for Time Series



# Summary

- Convolutional Neural Networks
  - History
  - Applications
  - Layers
    - Fully connected layer
    - Convolutional layer
    - Pooling layer
  - 1-D Convolution for Time Series