Cognitive Services

LM Informatica & Data Science - 2nd semester - 6 CFU

Convolutional Neural Networks for Visual Recognition

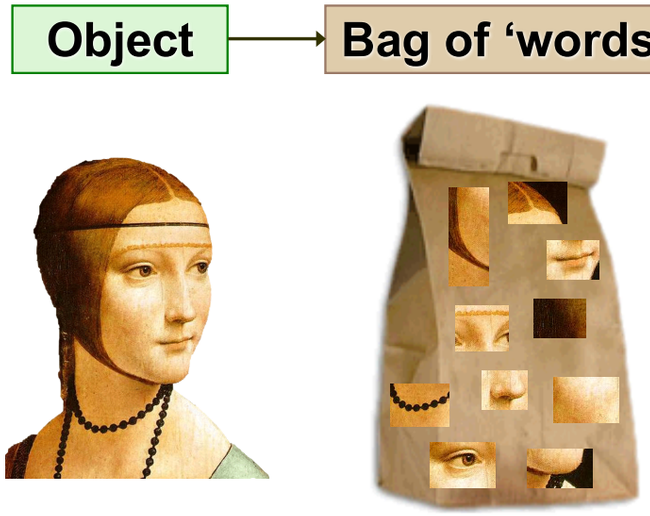
Lamberto Ballan

University of Padova, A.Y. 2018/19 May 13, 2019

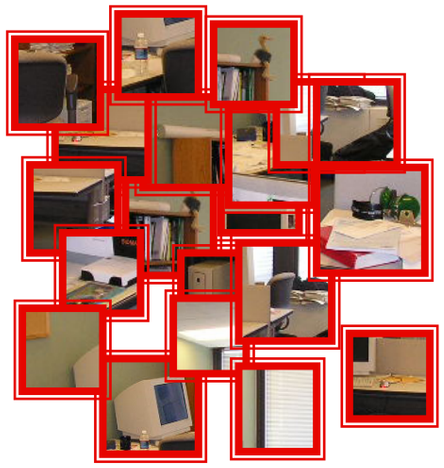
What we learned in the last class

(a brief summary)

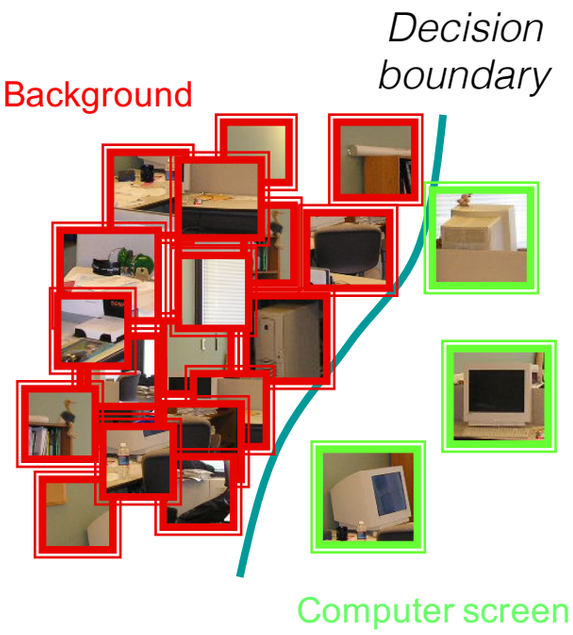
- Introduction to visual (object) recognition
- Image classification
- Bag of (visual) words
- Spatial pyramids



Input image



Bag of image patches



Feature space

Bag of features: pipeline

- Feature extraction:

- Codebook formation:

- Quantize features using visual vocabulary

- Represent images by frequencies of “visual words”:

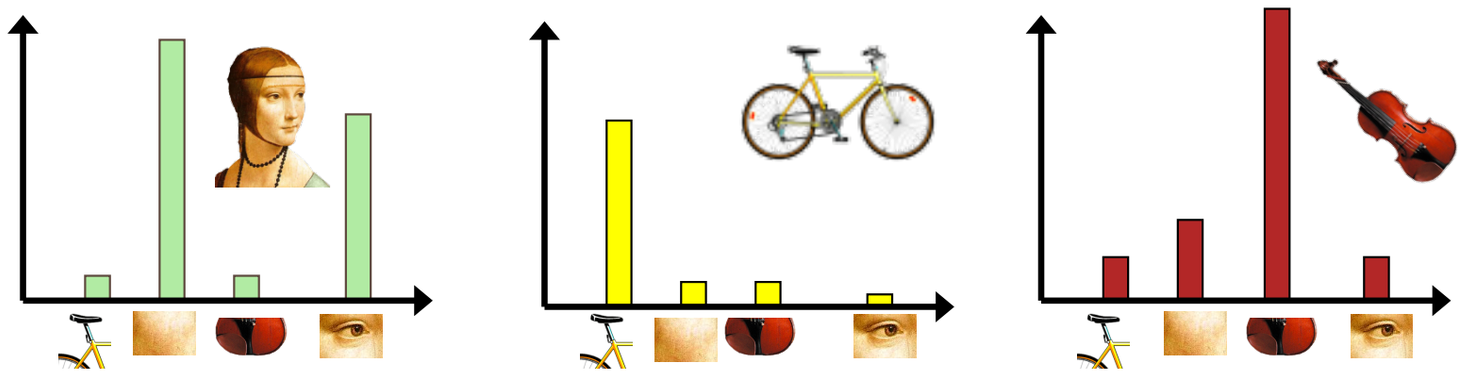
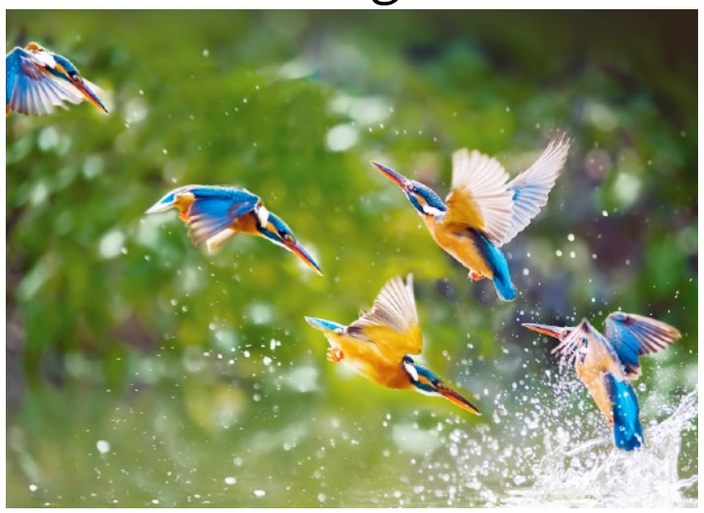


Image classification

- We need two major “ingredients”:
  - a way to *describe* images (e.g bow-histograms)
  - a procedure to *compare* images and *learn* a statistical model for a class (i.e. a classifier: kNN, SVM, etc.)

Image

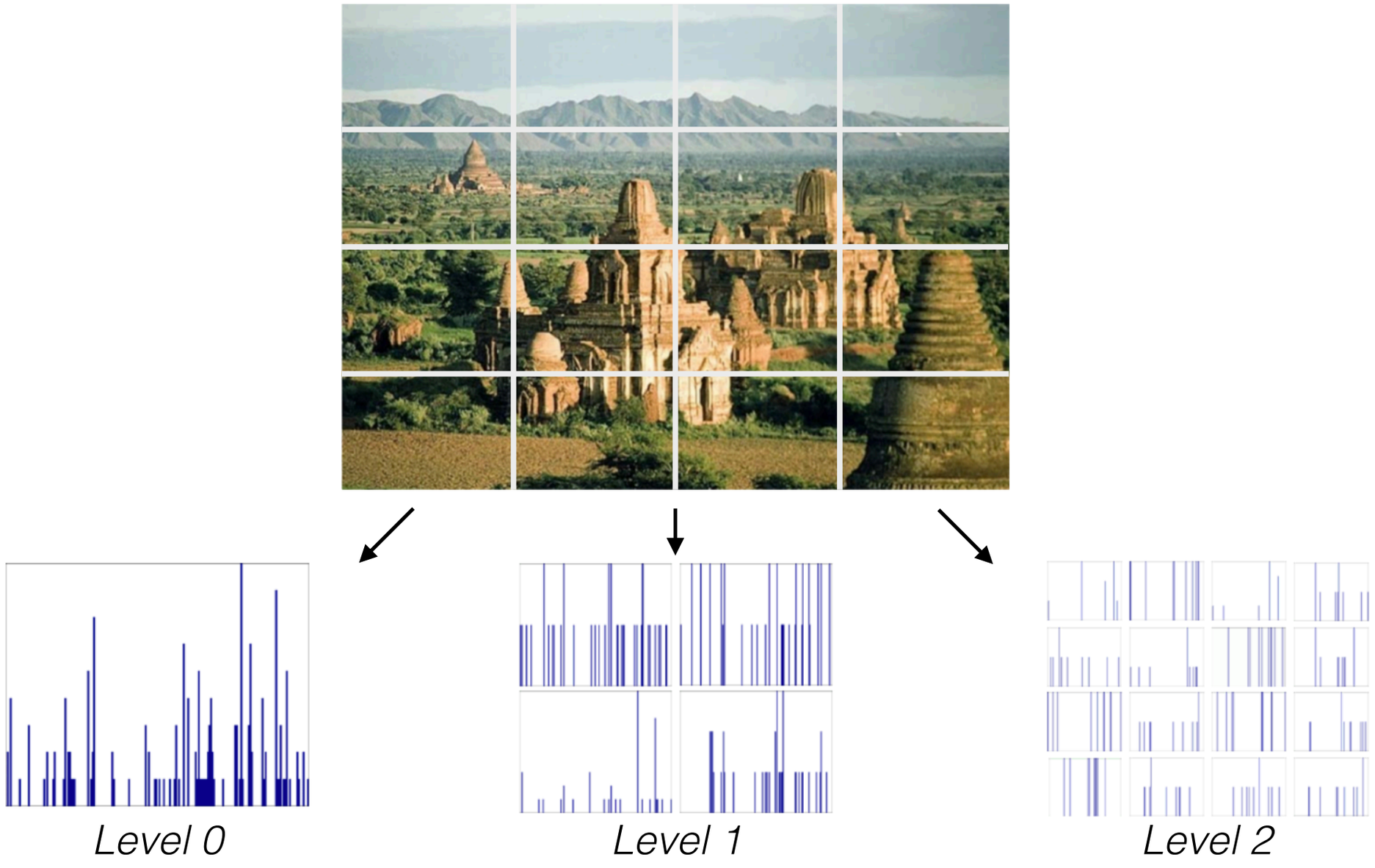


img_feat: $[c_1,\ldots,c_N]$

bird

Bag of features++: spatial pyramids

- Intuition: locally orderless representation at several levels of spatial resolution



Slide credit: S. Lazebnik

What we will learn today?

- From handcrafted features to representation learning
- Intro to Convolutional Neural Networks
- Conv, Pool and Fully Connected Layers; a closer look at spatial dimensions

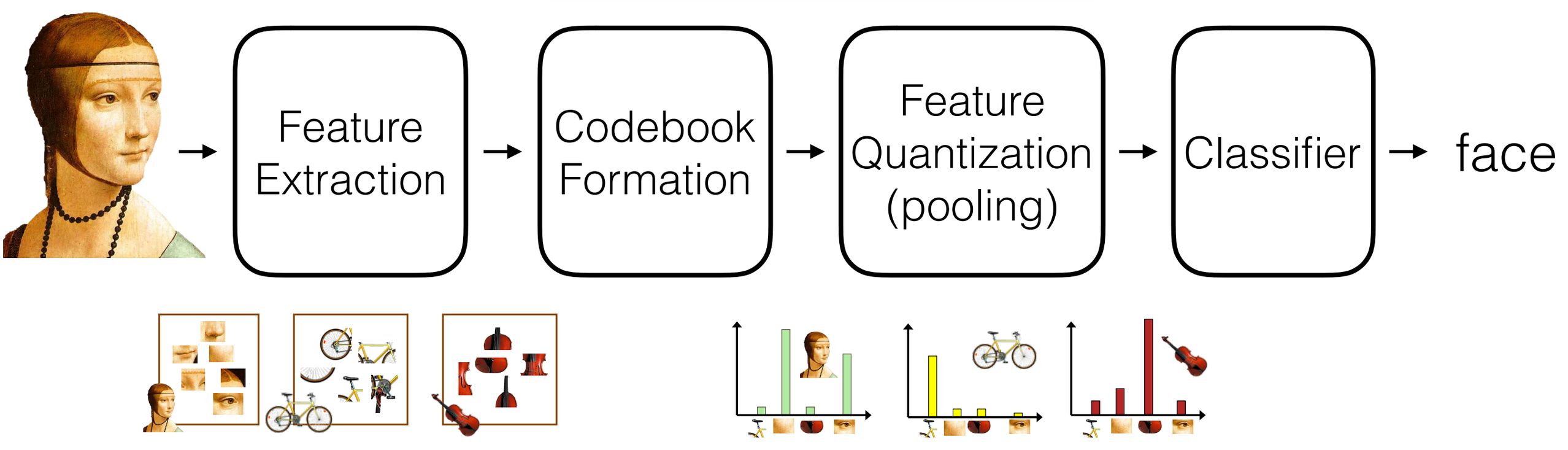
Several slides adapted from Stanford CS231n by L. Fei-Fei, A. Karpathy, J. Johnson, S. Yeung

Resources and background material:

- **http://cs231n.stanford.edu/** (slides, videos)
- **http://cs231n.github.io/** (notes and tutorials)

From hand-crafted representations to…

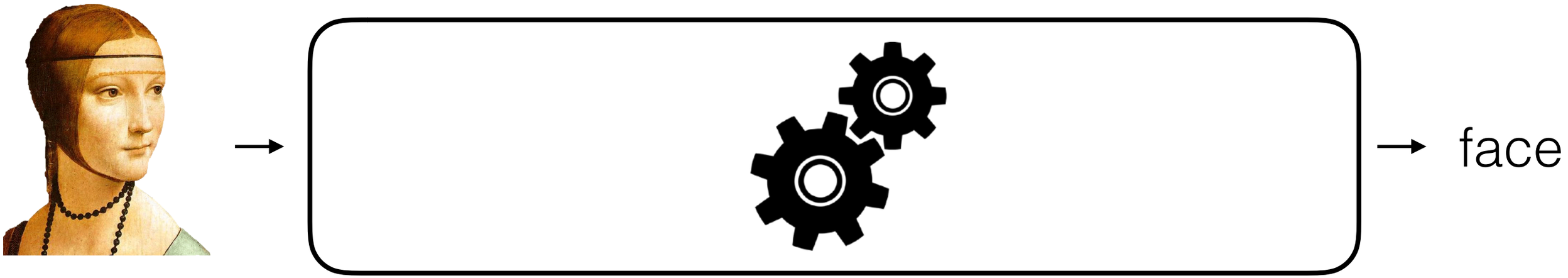
- Since now we have seen hand-crafted feature representations
- Let's take a look (again) at our bow pipeline:



Representation learning

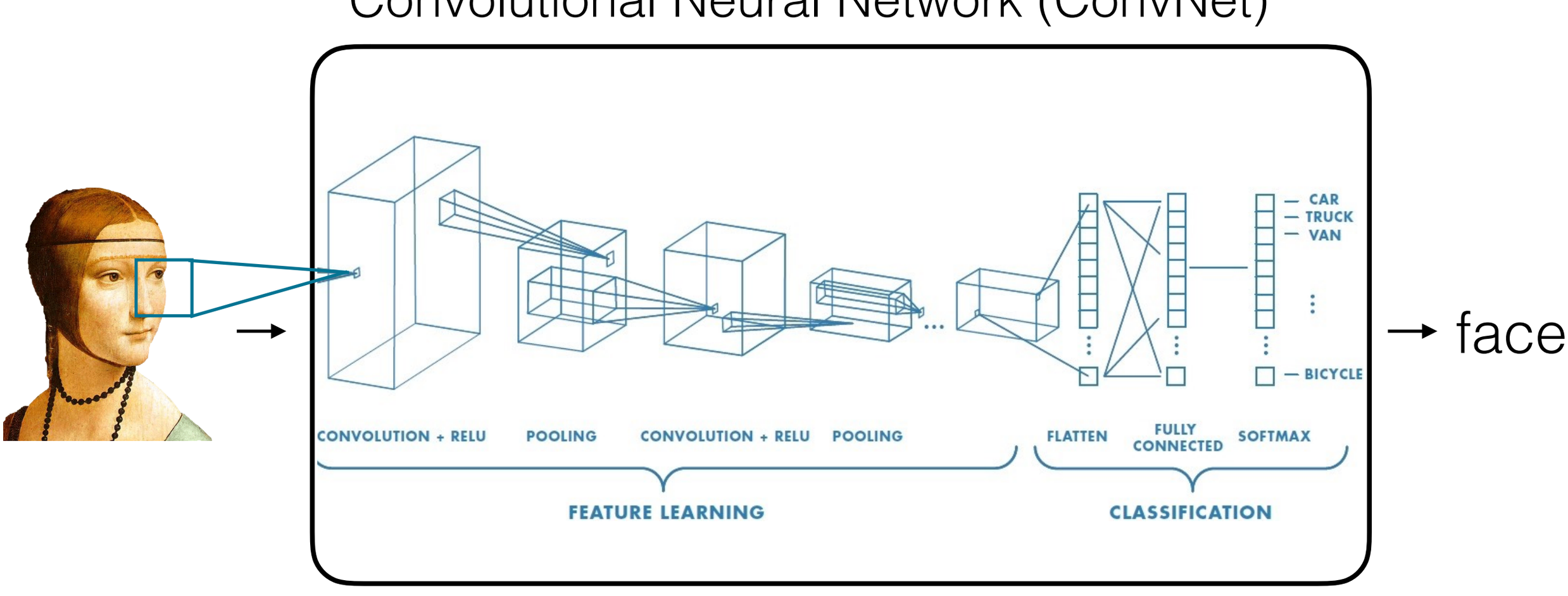
- Intuition: learn also the (image) representation directly from the data (*end-to-end learning)*

Deep Learning can be summarized as learning both the representation and the classifier out of data



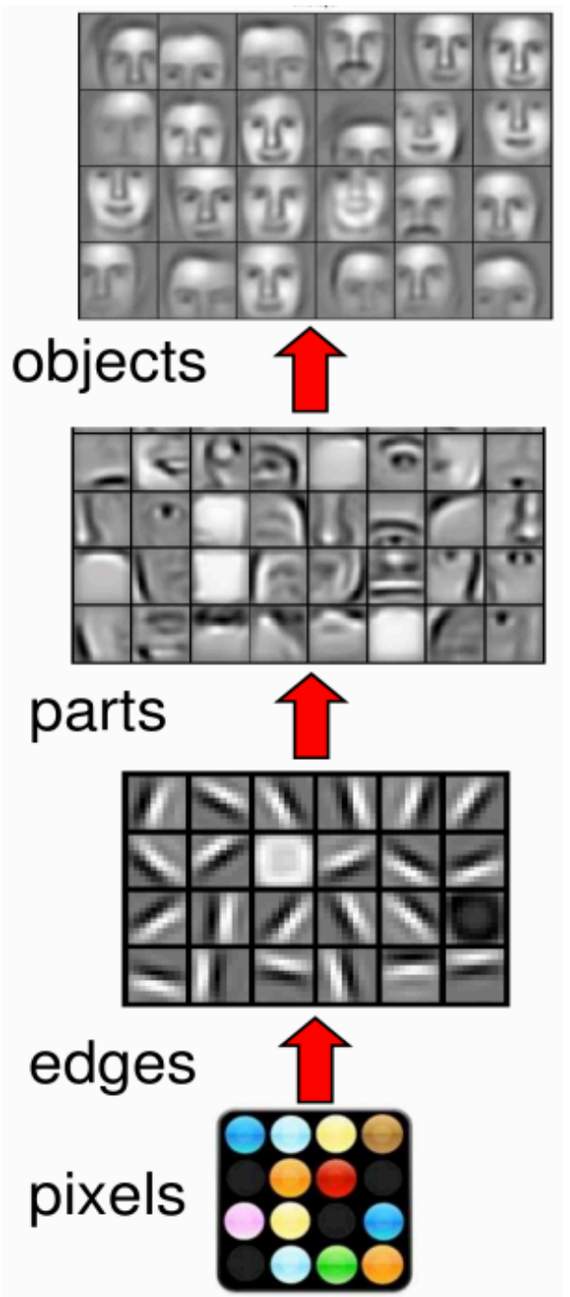
Representation learning

- Intuition: learn also the (image) representation directly from the data



Representation learning

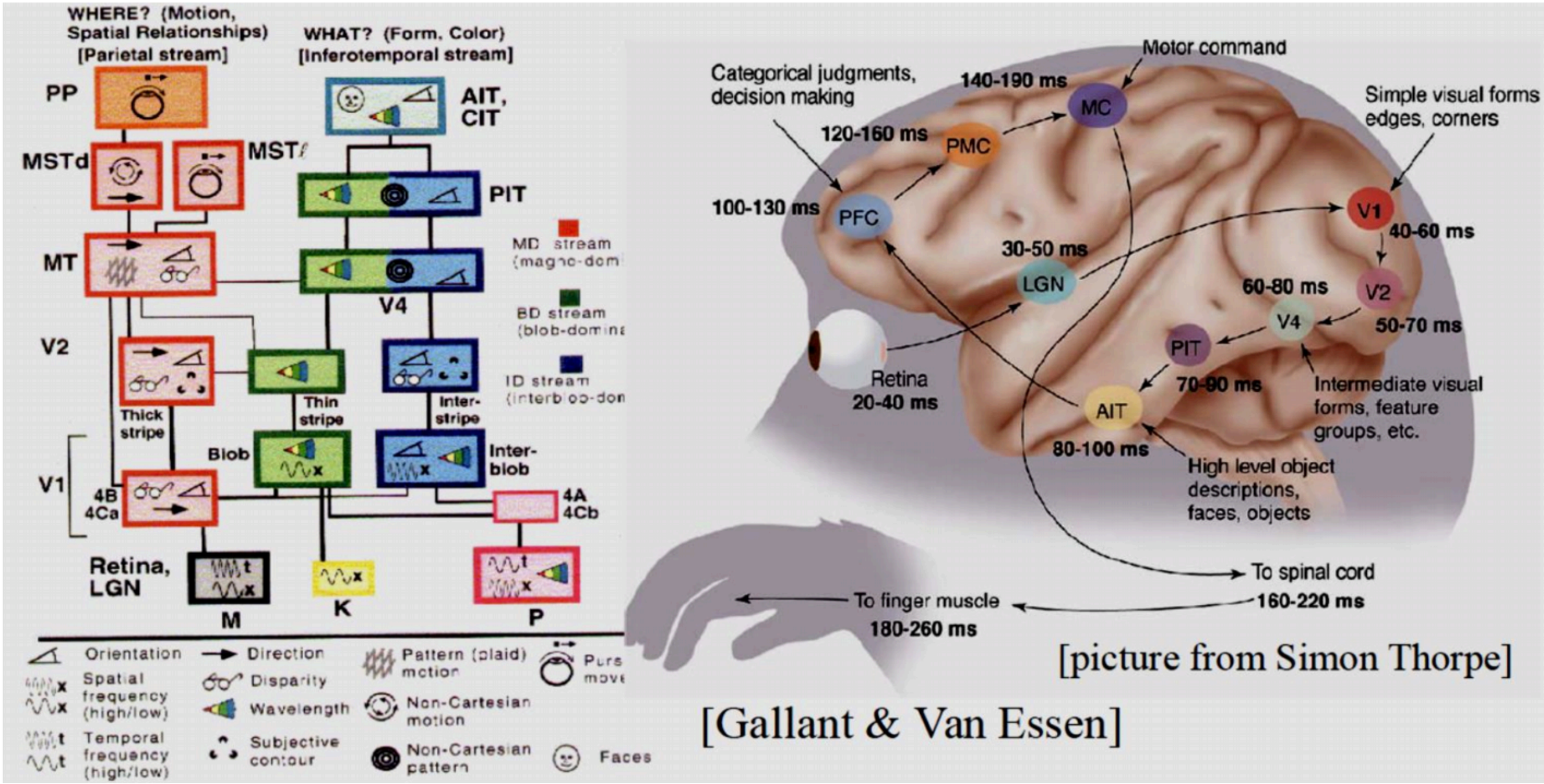
- Learning the representation is a challenging problem for AI/ML, Neuroscience, Cognitive Science
- Cognitive perspective:
  - How can a perceptual system build itself by looking at the external world?
  - How much prior "structure" is needed?
- Neuroscience perspective:
  - Does the cortex run a single general learning algorithm or multiple simpler ones?
- AI/ML perspective:
  - What is the fundamental principle?
  - What are the learning algorithm and architecture?



Slide credit: Y. LeCun

Inspiration / Biological analogy

- The ventral (recognition) pathway in the visual cortex has multiple stages

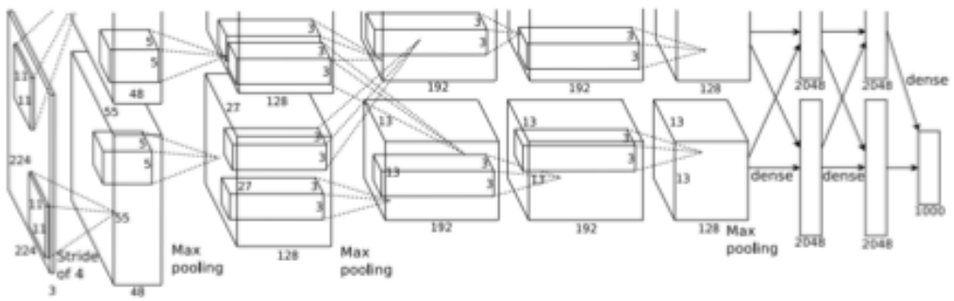


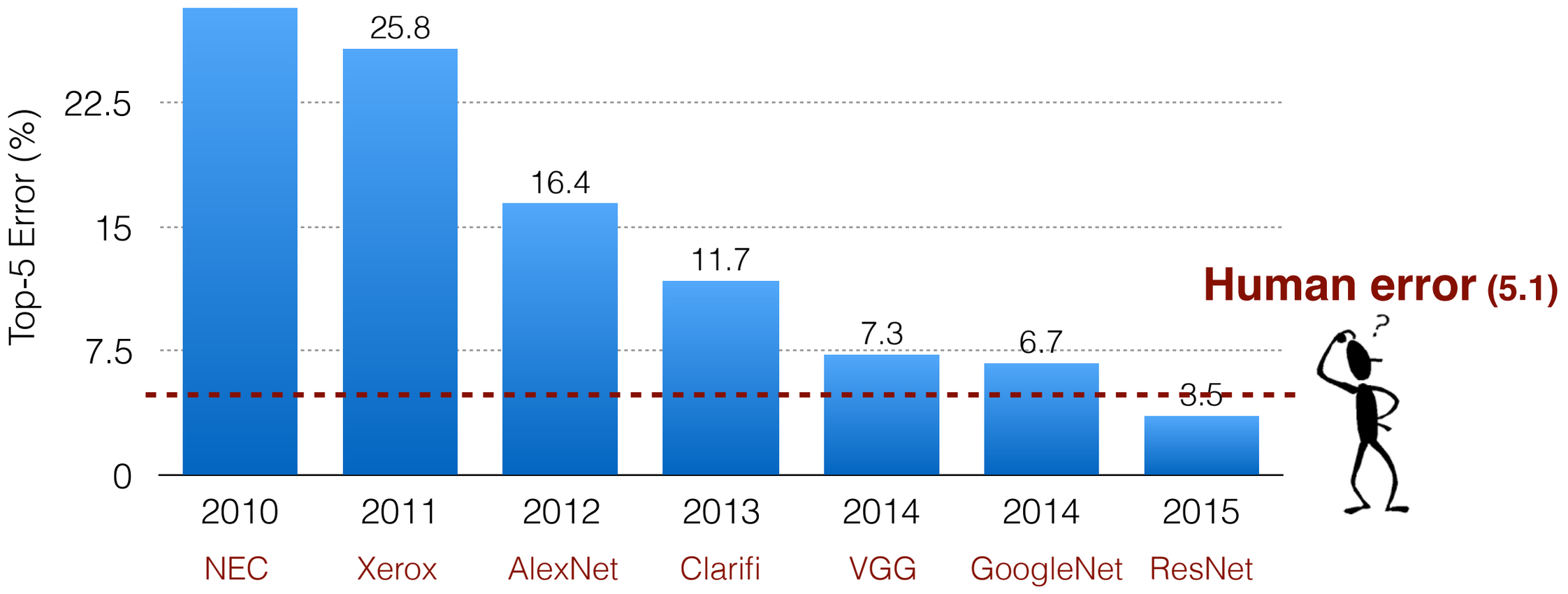
Neural networks are cool again

- Datasets drive computer vision progress

Neural networks are cool again

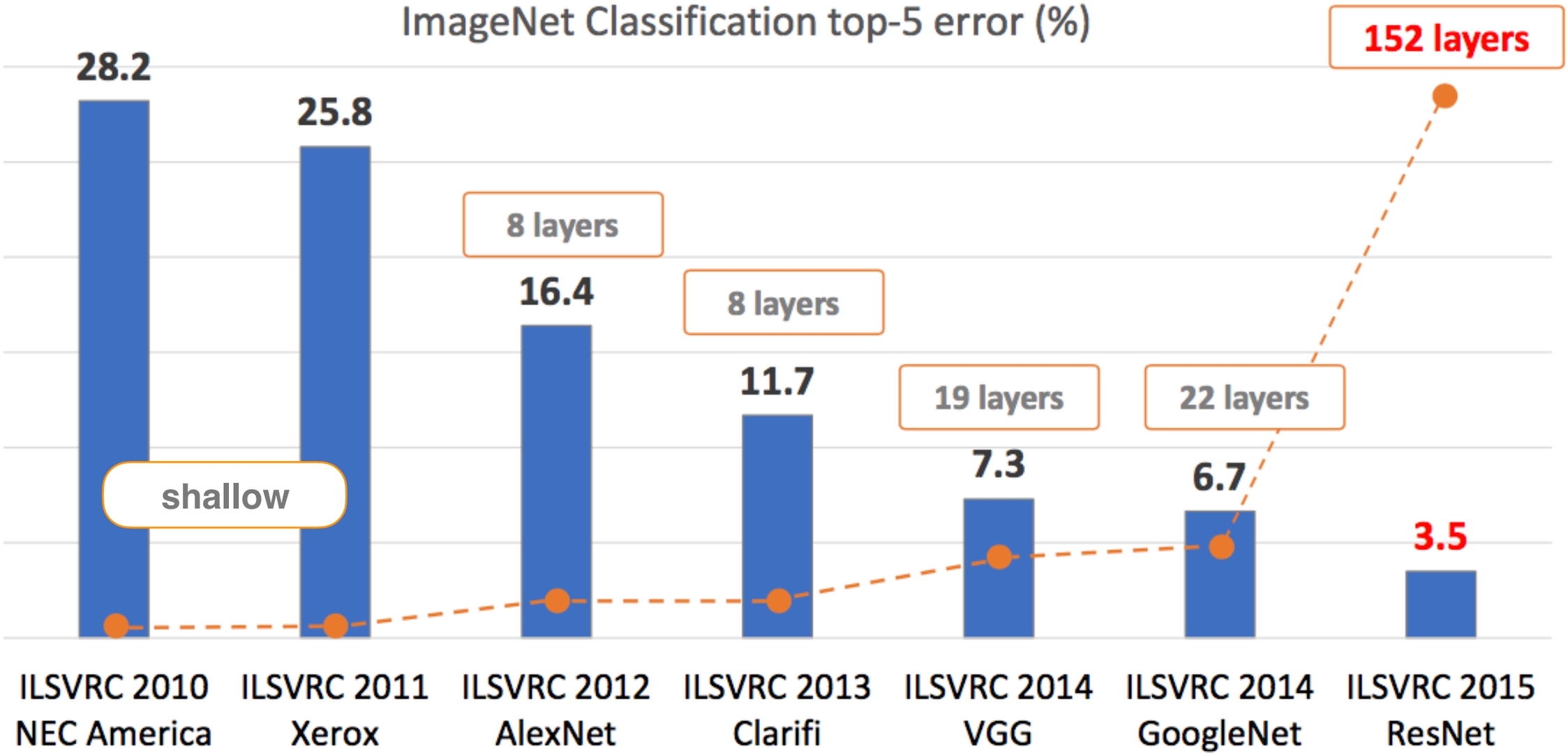
- ImageNet-ILSVRC (classification) over the years





The rise of deep learning

- Revolution of depth: how deep is deep?



Scale drive deep learning progress

Revisiting Unreasonable Effectiveness of Data in Deep Learning Era

Chen Sun[1], Abhinav Shrivastava[1,2], Saurabh Singh[1], and Abhinav Gupta[1,2]

[1]Google Research
[2]Carnegie Mellon University

Abstract

The success of deep learning in vision can be attributed to: (a) models with high capacity; (b) increased computational power; and (c) availability of large-scale labeled data. Since 2012, there have been significant advances in representation capabilities of the models and computational capabilities of GPUs. But the size of the biggest dataset has surprisingly remained constant. What will happen if we increase the dataset size by 10× or 100×? This paper takes a step towards clearing the clouds of mystery surrounding the relationship between 'enormous data' and visual deep learning. By exploiting the JFT-300M dataset which has more than 375M noisy labels for 300M images, we investigate how the performance of current vision tasks would change if this data was used for representation learning. Our paper delivers some surprising (and some expected) findings. First, we find that the performance on vision tasks increases logarithmically based on volume of training data size. Second, we show that representation learning (or pre-training) still holds a lot of promise. One can improve performance on many vision tasks by just training a better base model. Finally, as expected, we present new state-of-the-art results for different vision tasks including image classification, object detection, semantic segmentation and human pose estimation. Our sincere hope is that this inspires vision community to not undervalue the data and develop collective efforts in building larger datasets.

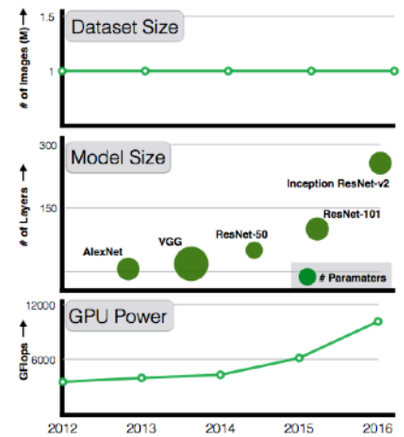


Figure 1. The Curious Case of Vision Datasets: While GPU computation power and model sizes have continued to increase over the last five years, size of the largest training dataset has surprisingly remained constant. Why is that? What would have happened if we have used our resources to increase dataset size as well? This paper provides a sneak-peek into what could be if the dataset sizes are increased dramatically.

ously, while both GPUs and model capacity have continued to grow, datasets to train these models have remained

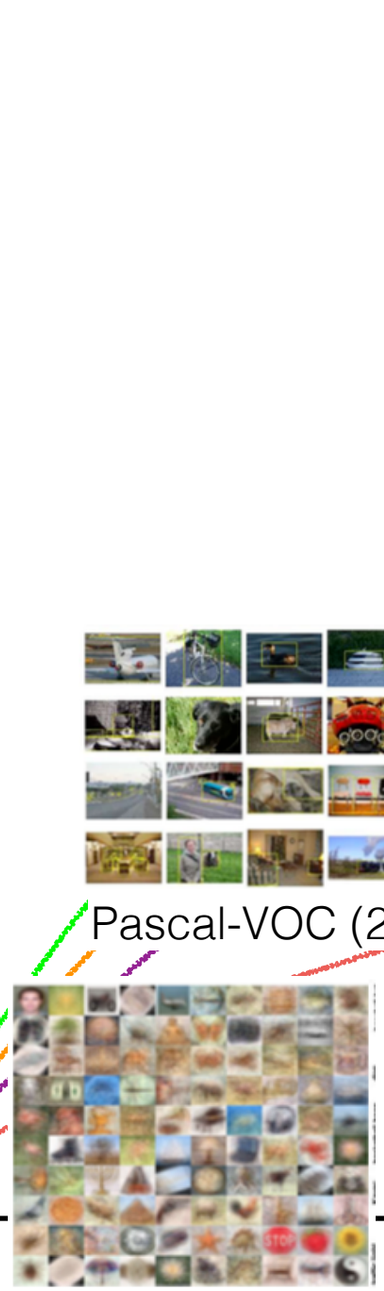
Performance

works

tworks

ks

Pascal-VOC (2007)



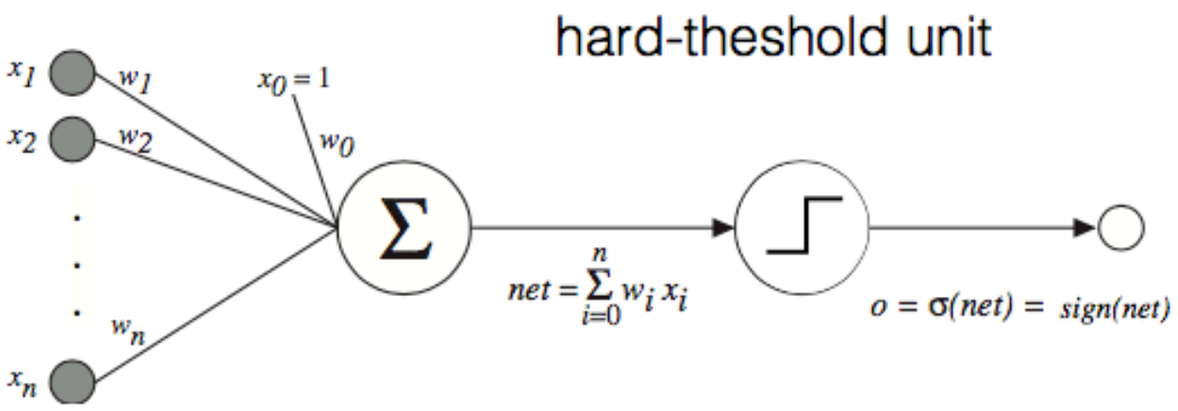
Caltech101 (2005)

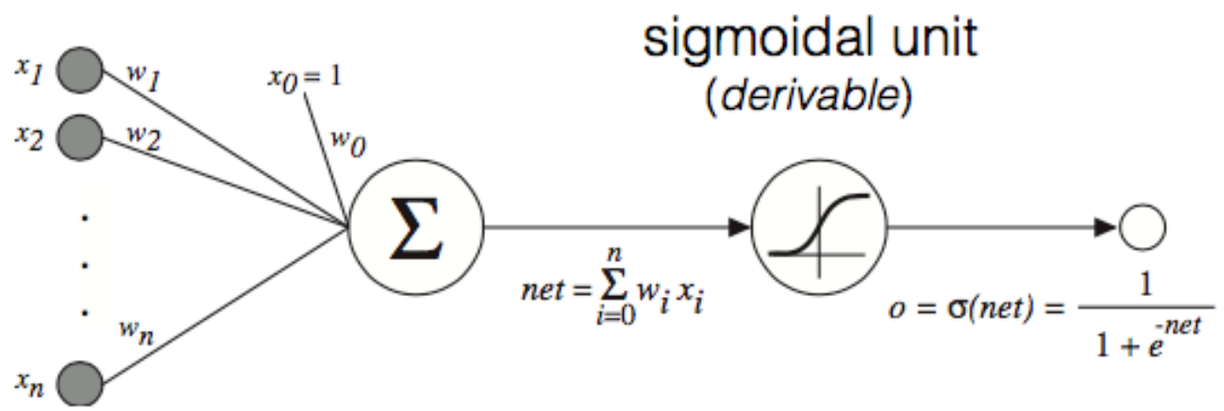
Amount of data

Adapted from slide by A. Ng

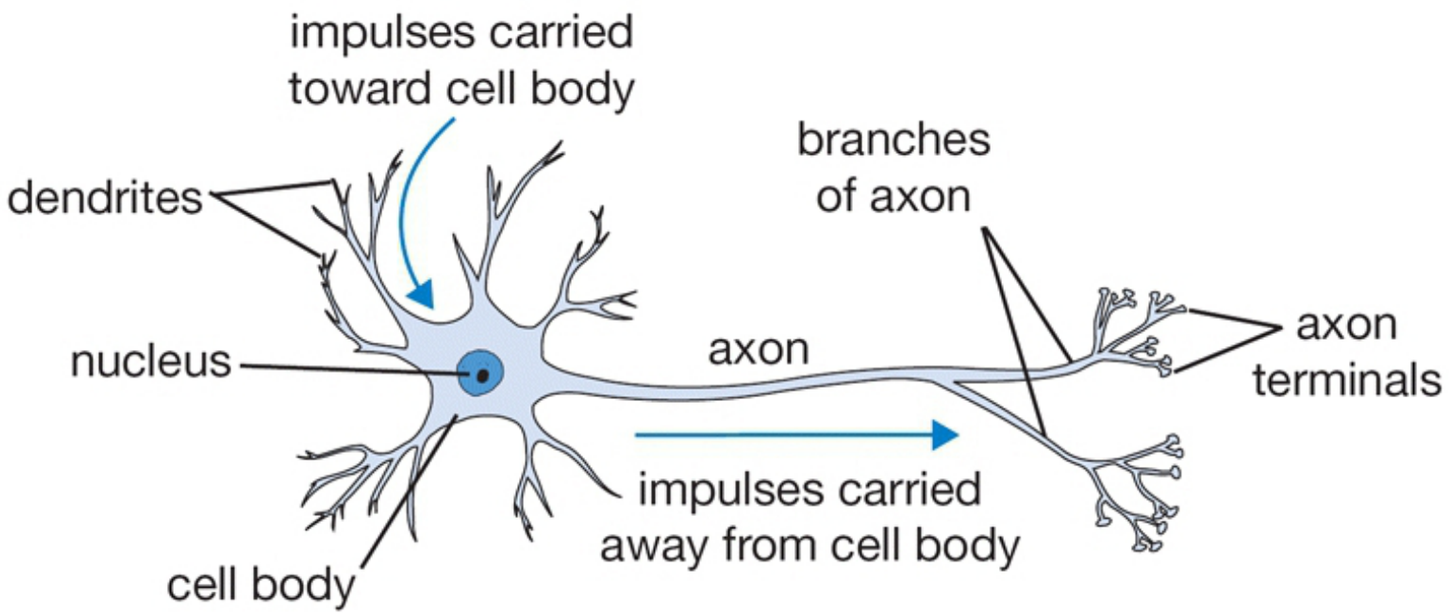
Neural Networks recap

- Artificial Neural Network: it is a system consisting of interconnected units that compute *nonlinear functions*
  - *input* units represent input variables
  - *output* units represent output variables
  - *hidden* units (if present) represent internal variables that codify (after learning) correlations among input and desired output variables
  - *weights* are associated to connections among units



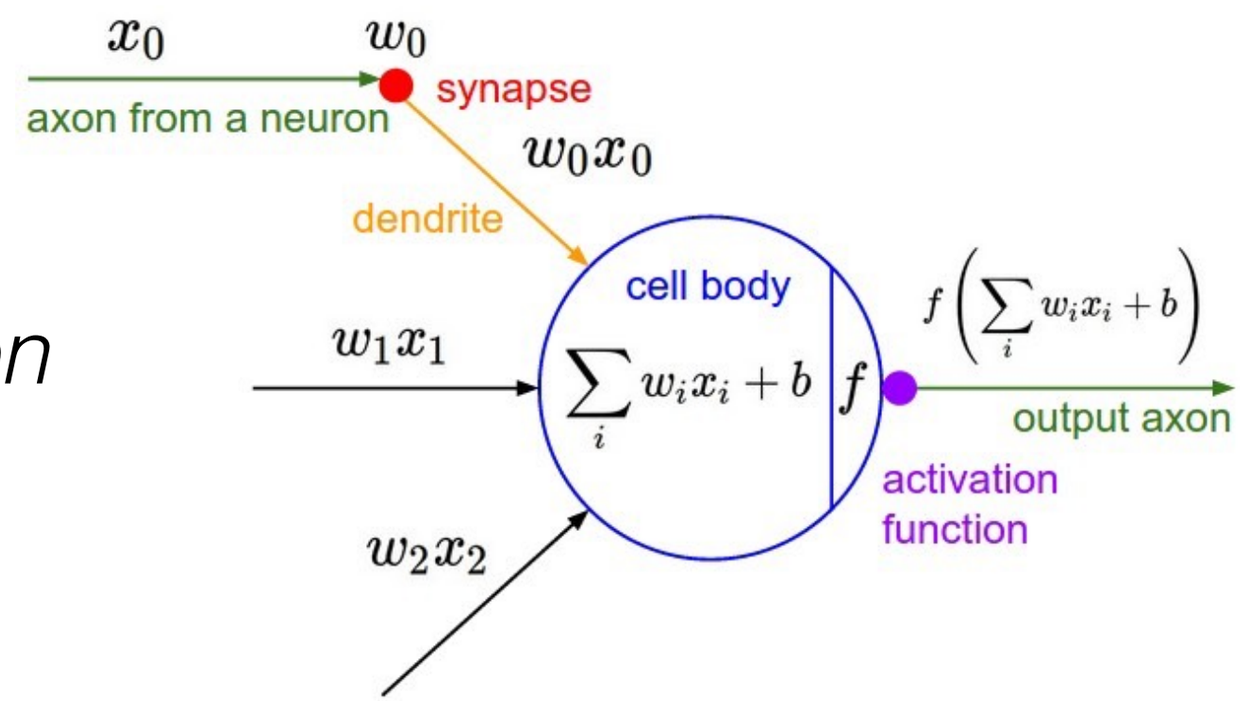


Neural Networks recap: biological analogy



Biological Neuron

A simple Artificial Neuron (perceptron)



e.g. sigmoid activation

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

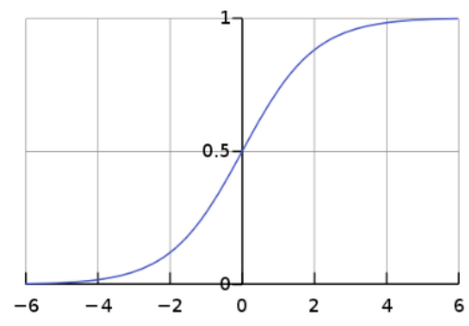


Image credit: Fei-Fei, Karpathy, Johnson

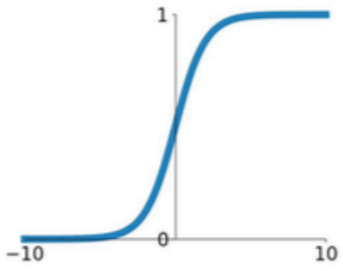
Neural Networks recap: activation functions

- **Feed-forward**: each neuron performs a dot product with the input and its weights, adds the bias and applies the activation function (or *non-linearity*)

- There are several activation functions you may encounter:

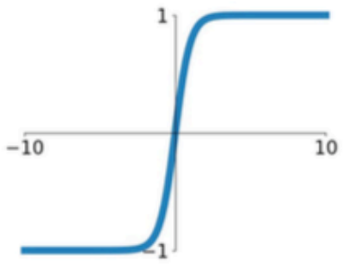
Sigmoid

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



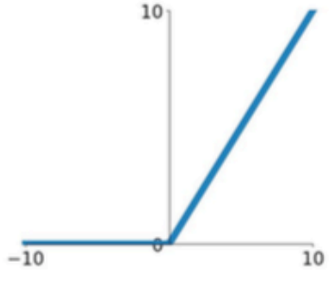
tanh

$$\tanh(x)$$



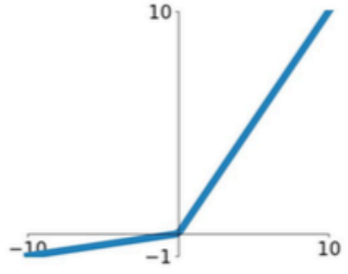
ReLU

$$\max(0, x)$$



Leaky ReLU

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

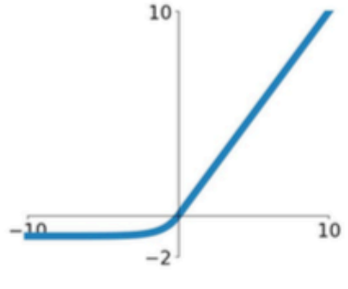


Maxout

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

ELU

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

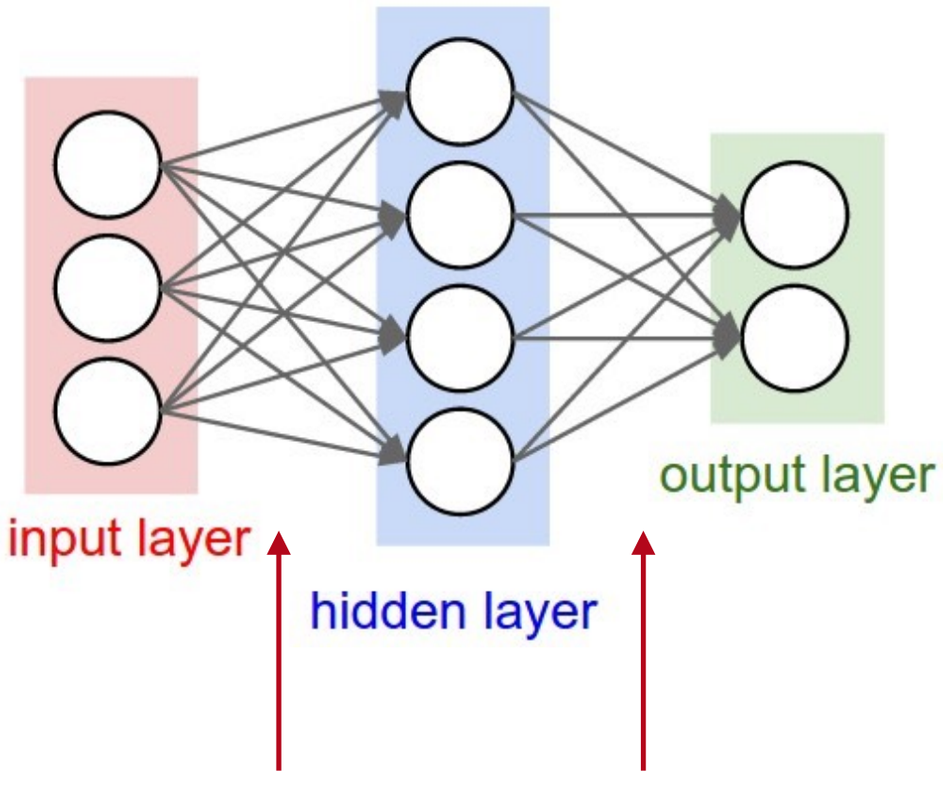


Neural Networks recap: architectures

- Note: when we say N-layer neural network, we do not count the input layer

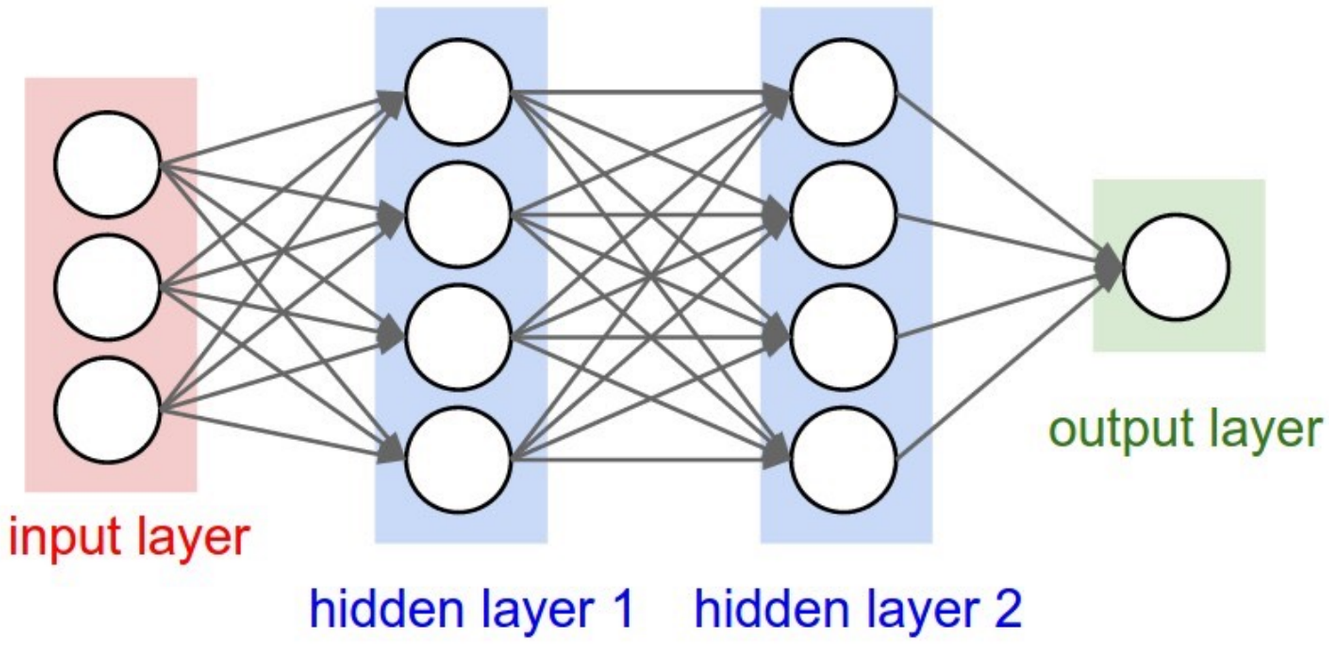
2-layer neural network

or 1-hidden-layer neural net



3-layer neural network

or 2-hidden-layer neural net



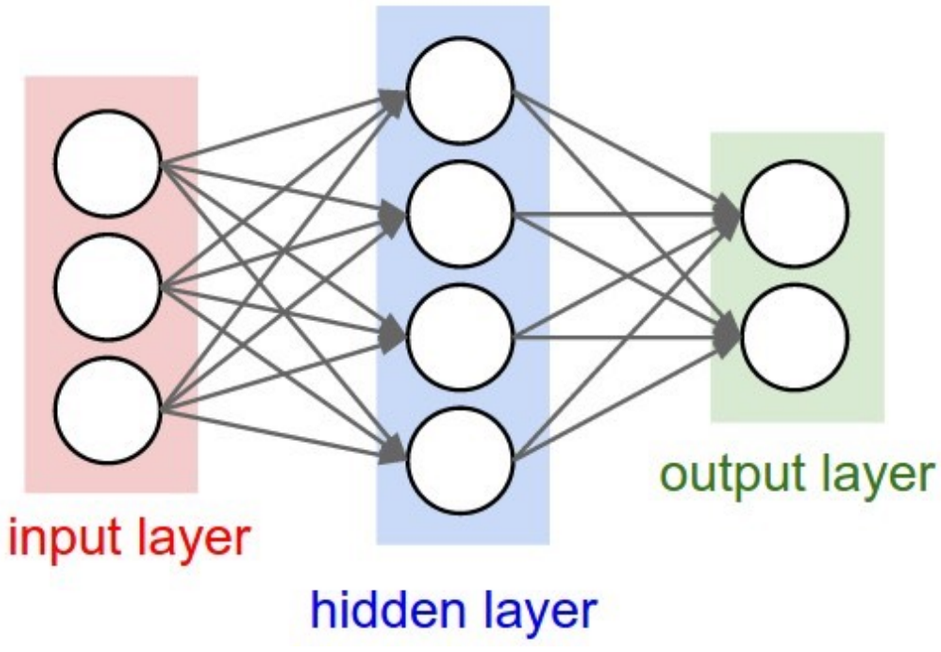
Fully Connected layers

Image credit: Fei-Fei, Karpathy, Johnson

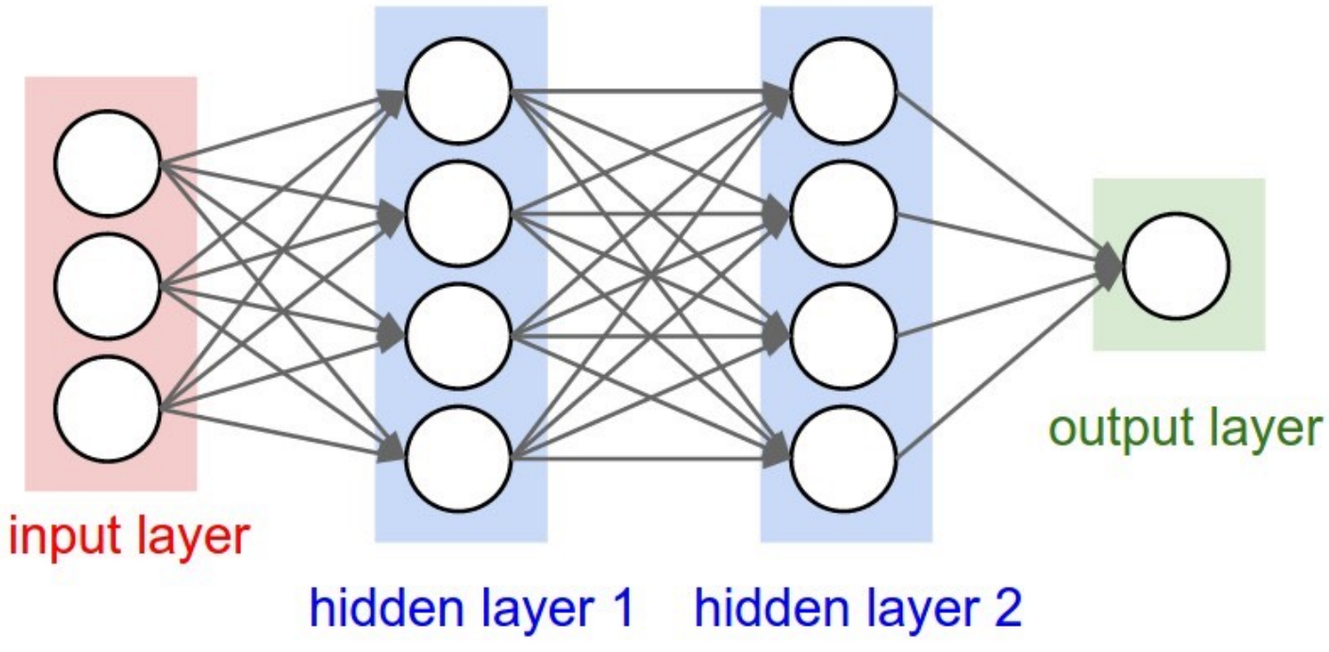
Neural Networks recap: architectures

- **Sizing neural networks:** the two metrics that are commonly used to measure the size of a NeuralNet are the #neurons or (more commonly) the #parameters

(a) **2-layer neural network**



(b) **3-layer neural network**

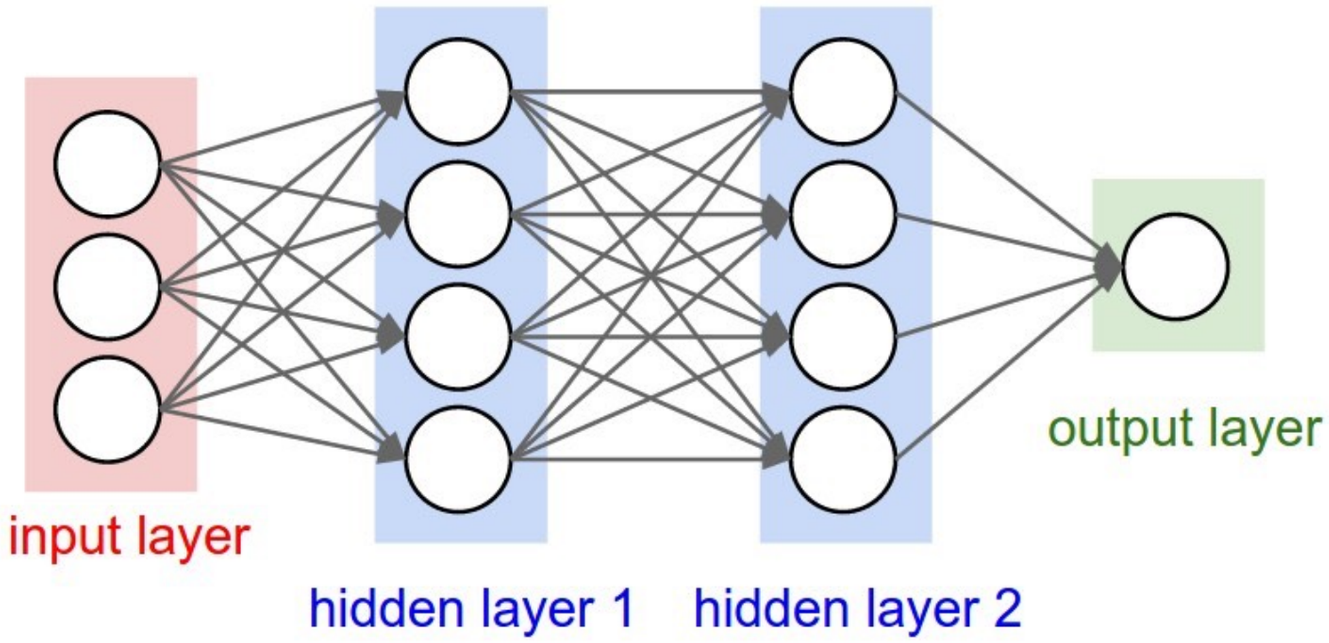


(b) has 4+4+1 neurons; how many (learnable) parameters?

(b) #parameters: 3x4 + 4x4 + 4x1 = 32 + 9 *(biases)* = 41

Neural Networks recap: computation

- Example feed-forward computation of a NeuralNet:



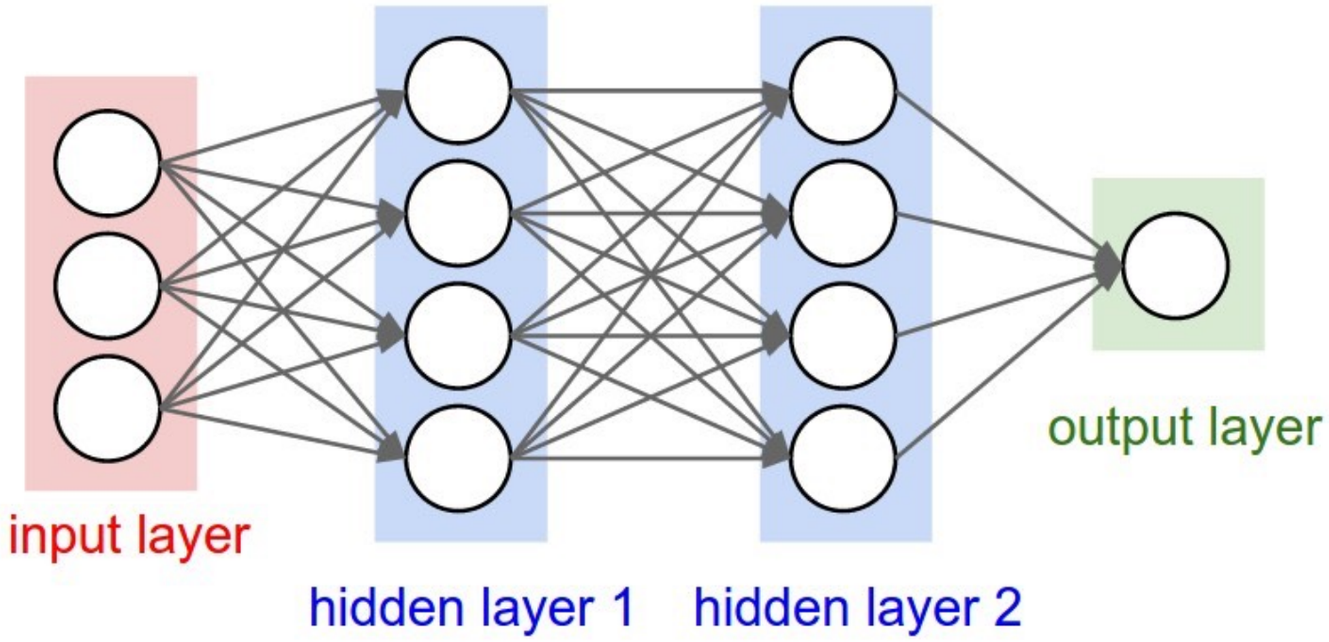
We can efficiently evaluate an entire layer of neurons

```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```


Slide credit: Fei-Fei, Karpathy, Johnson

Neural Networks recap: computation

- Example feed-forward computation of a NeuralNet:



We can efficiently evaluate an entire layer of neurons

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```


Slide credit: Fei-Fei, Karpathy, Johnson

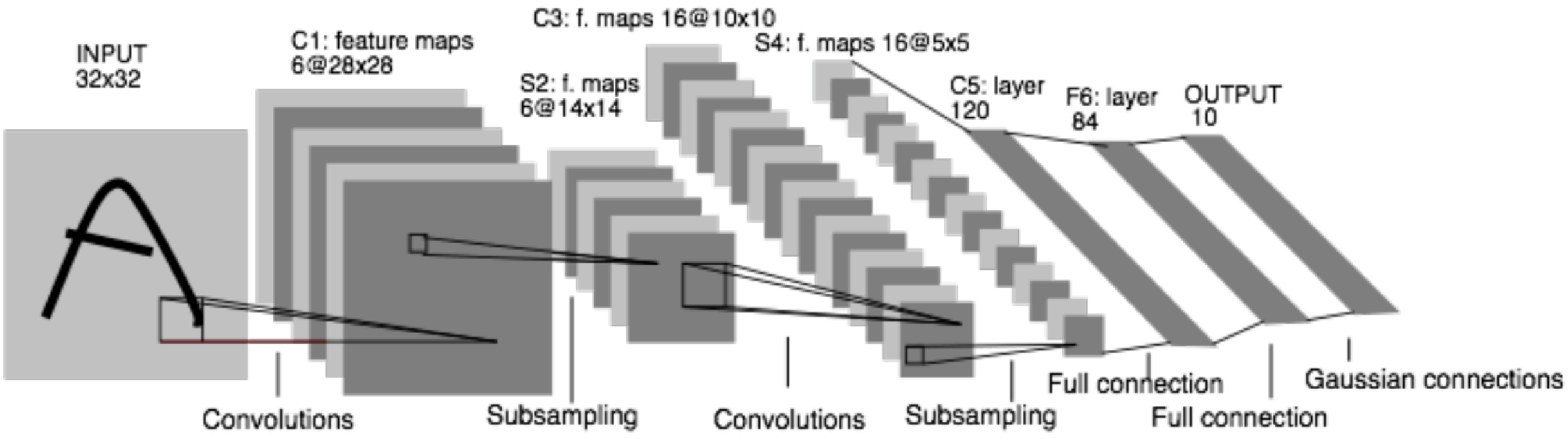
Neural Networks recap

- Architectures: we arrange neurons (units) into Fully Connected layers
  - Note: the abstraction of a layer has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- The forward pass of a FC layer corresponds to one matrix multiplication followed by a bias offset and an activation function
- Representational power: a neural network can approximate any continuous function

Convolutional Neural Networks

- CNNs: the assumption that the inputs are images allows us to encode certain local properties into the architecture

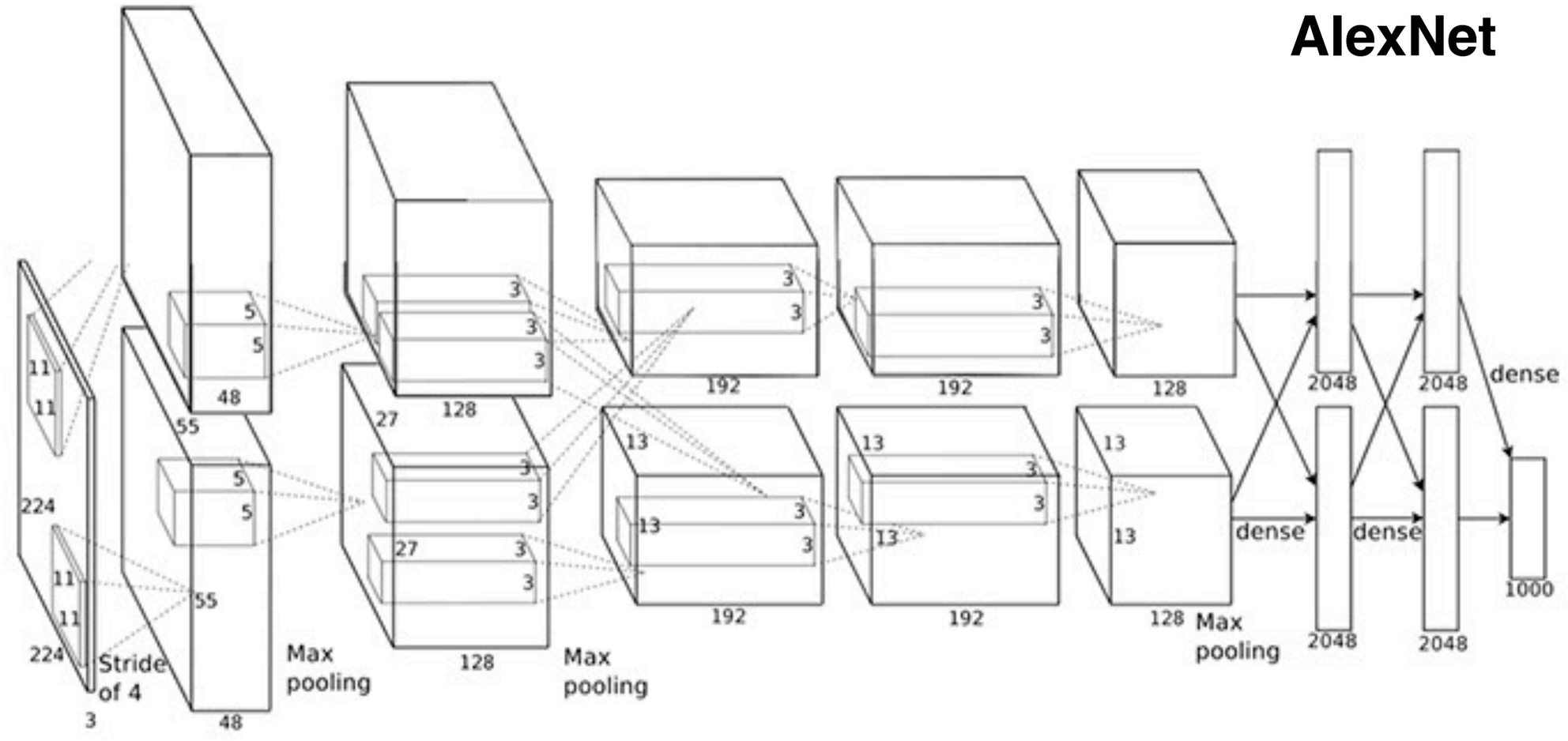
LeNet-5



Y.LeCun, L.Bottou, Y.Bengio, P.Haffner, “Gradient-based learning applied to document recognition”, Proc.IEEE 1998

Convolutional Neural Networks

- Return of the CNN: first strong “modern” results



A.Krizhevsky, I.Sutskever, G.Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012

Convolutional Neural Networks

- A ConvNet is a list of layers that transform the input image (volume) to the final class scores

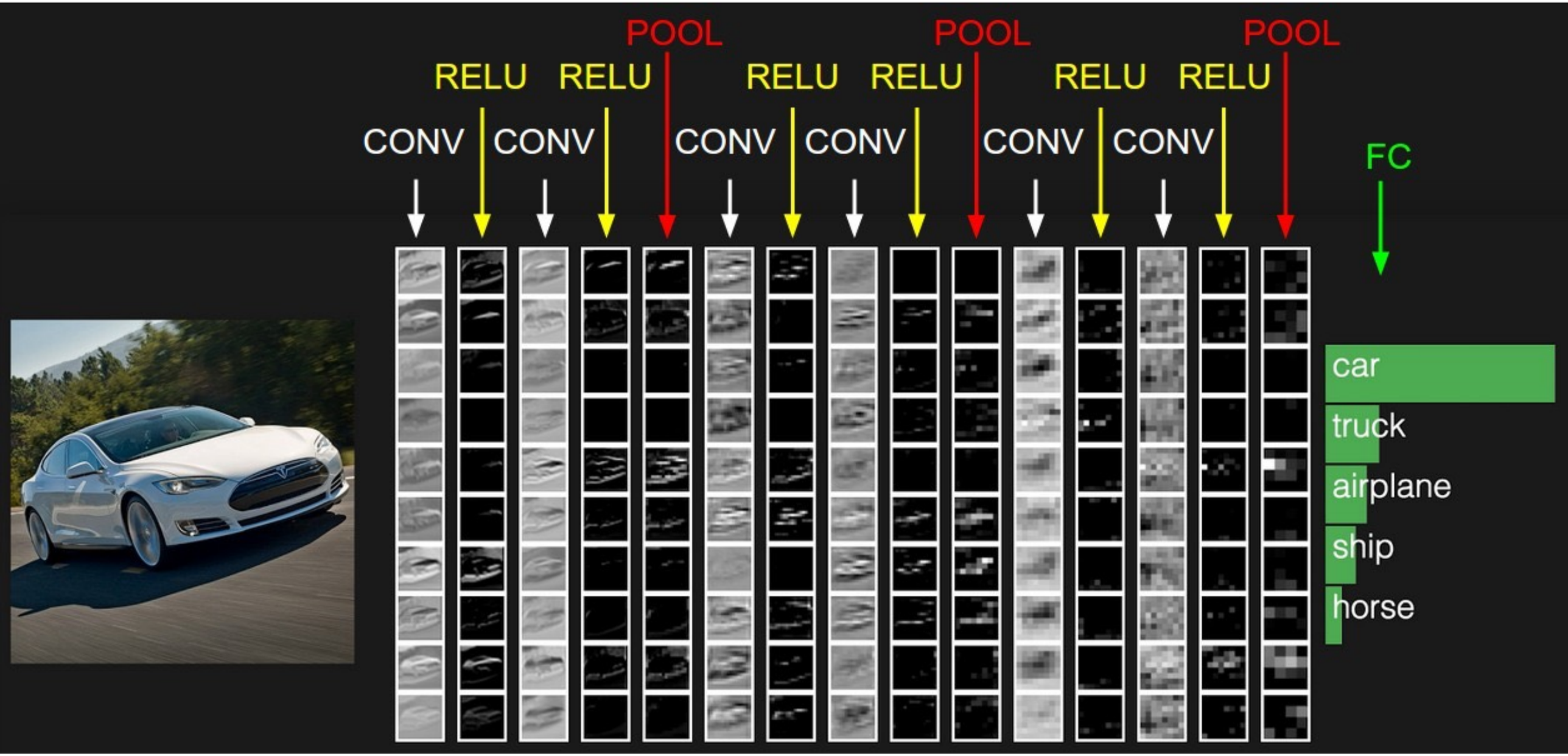


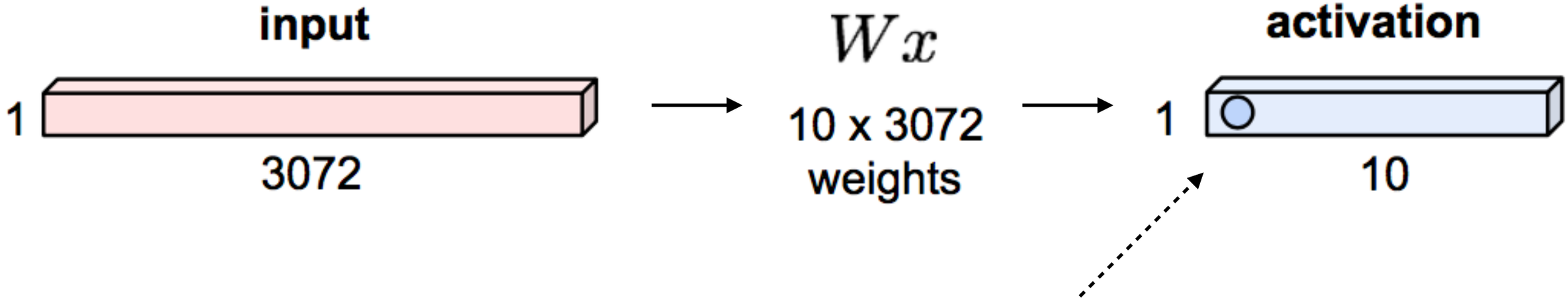
Image credit: Fei-Fei, Karpathy, Johnson

Convolutional Neural Networks

- ConvNet (CNN): a sequence of layers where each layer transforms one volume of activations to another through a *differentiable* function
- We have three main types of layers:
  - Convolutional Layer
  - Pooling Layer
  - Fully Connected Layer (exactly as in “regular” ANN)
- We stack these layers to form a full CNN architecture

Fully Connected layer

- input: 32x32x3 image -> *stretch to* 3072x1



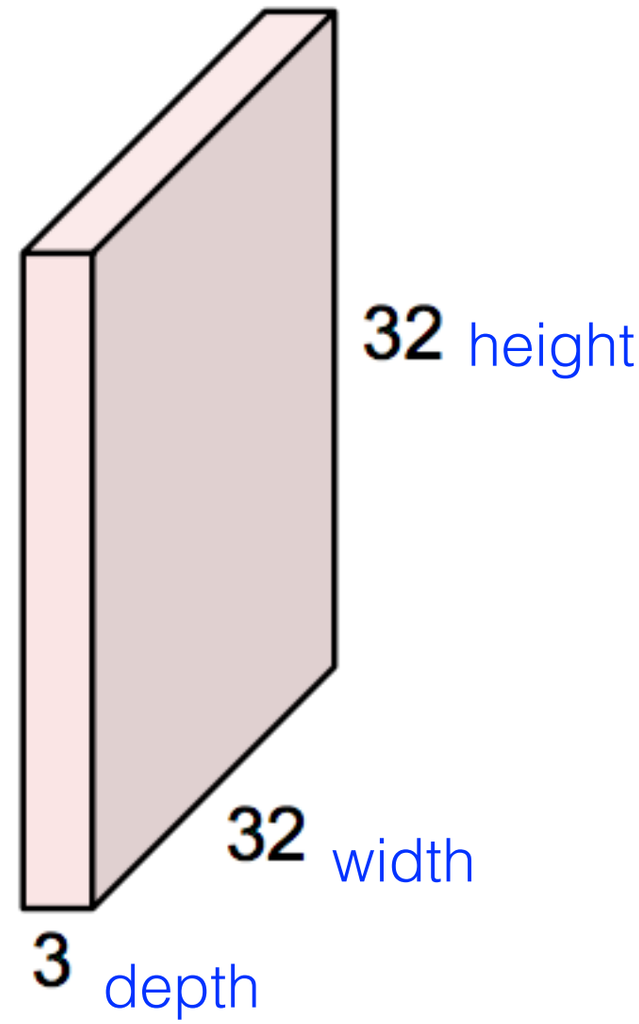
activation: *1 number, i.e. the result of taking a dot product between a row of W and the input (a 3072-d dot product)*

Slide credit: Fei-Fei, Karpathy, Johnson

Convolution layer

- input: 32x32x3 image -> *preserve spatial structure*

32x32x3 image



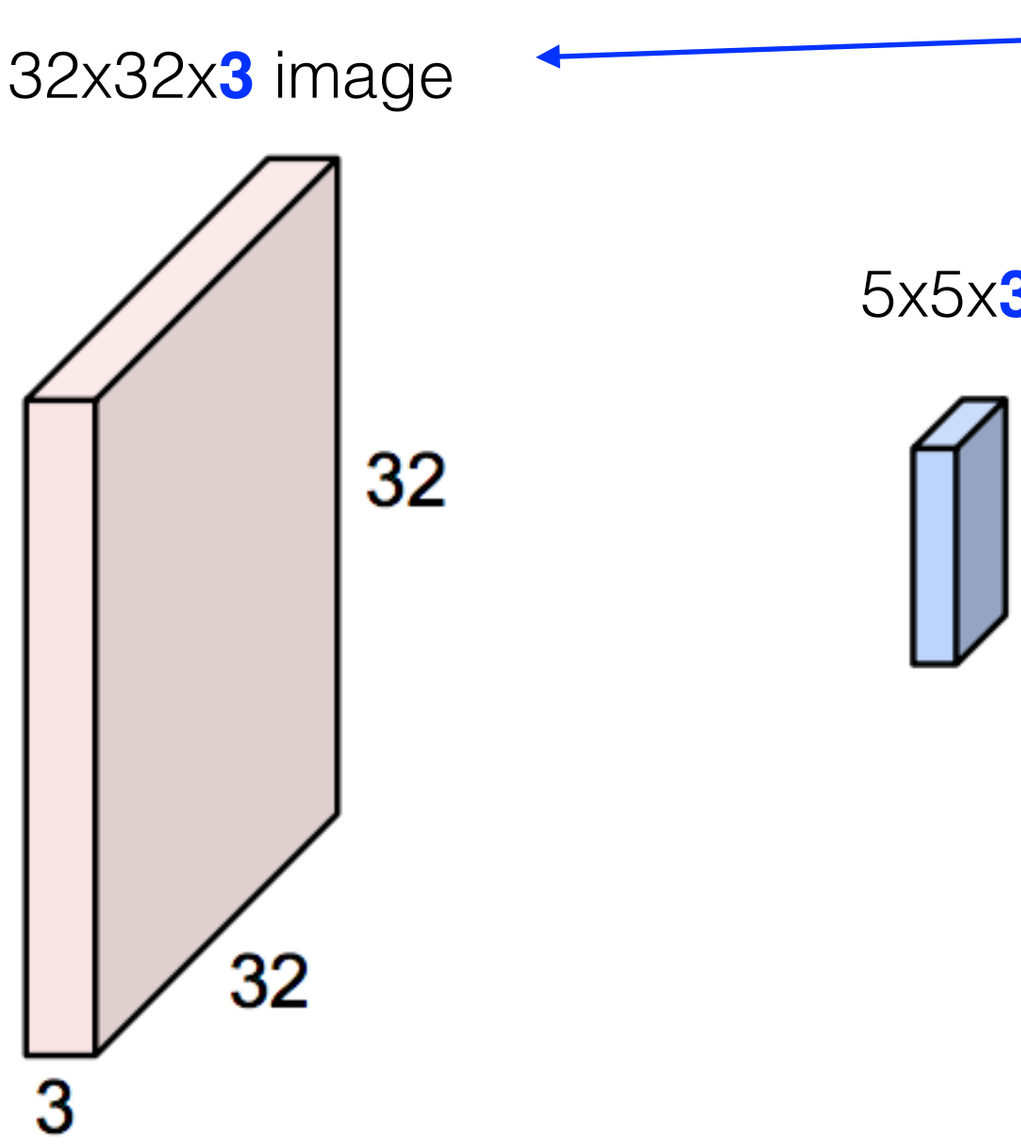
5x5x3 filter

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

Slide credit: Fei-Fei, Karpathy, Johnson

Convolution layer

- input: 32x32x3 image -> *preserve spatial structure*



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

Slide credit: Fei-Fei, Karpathy, Johnson

Recall: image filtering and convolution

- Image filtering example: Gaussian smoothing

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

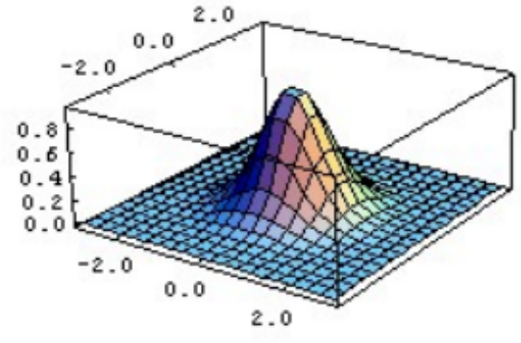
$F[x, y]$

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

$H[x, y]$

This kernel is an approximation of a 2D Gaussian function

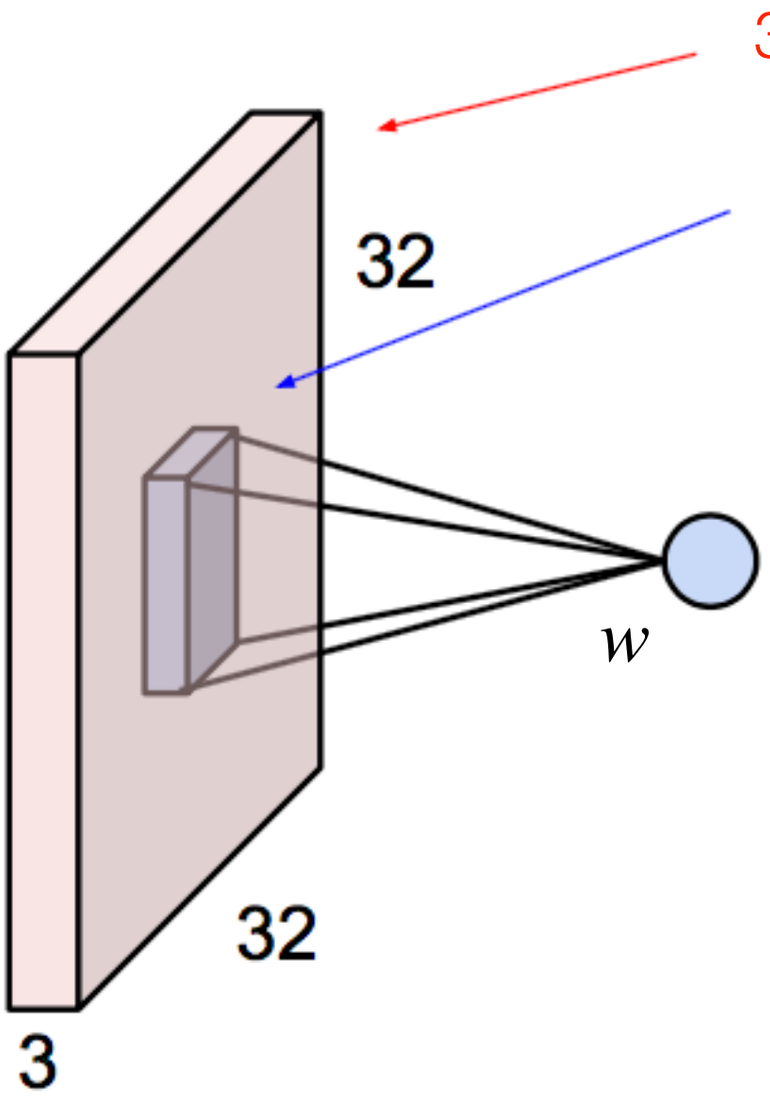
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Convolution: $$(f * h)[n, m] = \sum_{k, l} f[k, l]\, h[n\text{-}k, m\text{-}l]$$

Convolution layer

- input: 32x32x3 image -> *preserve spatial structure*



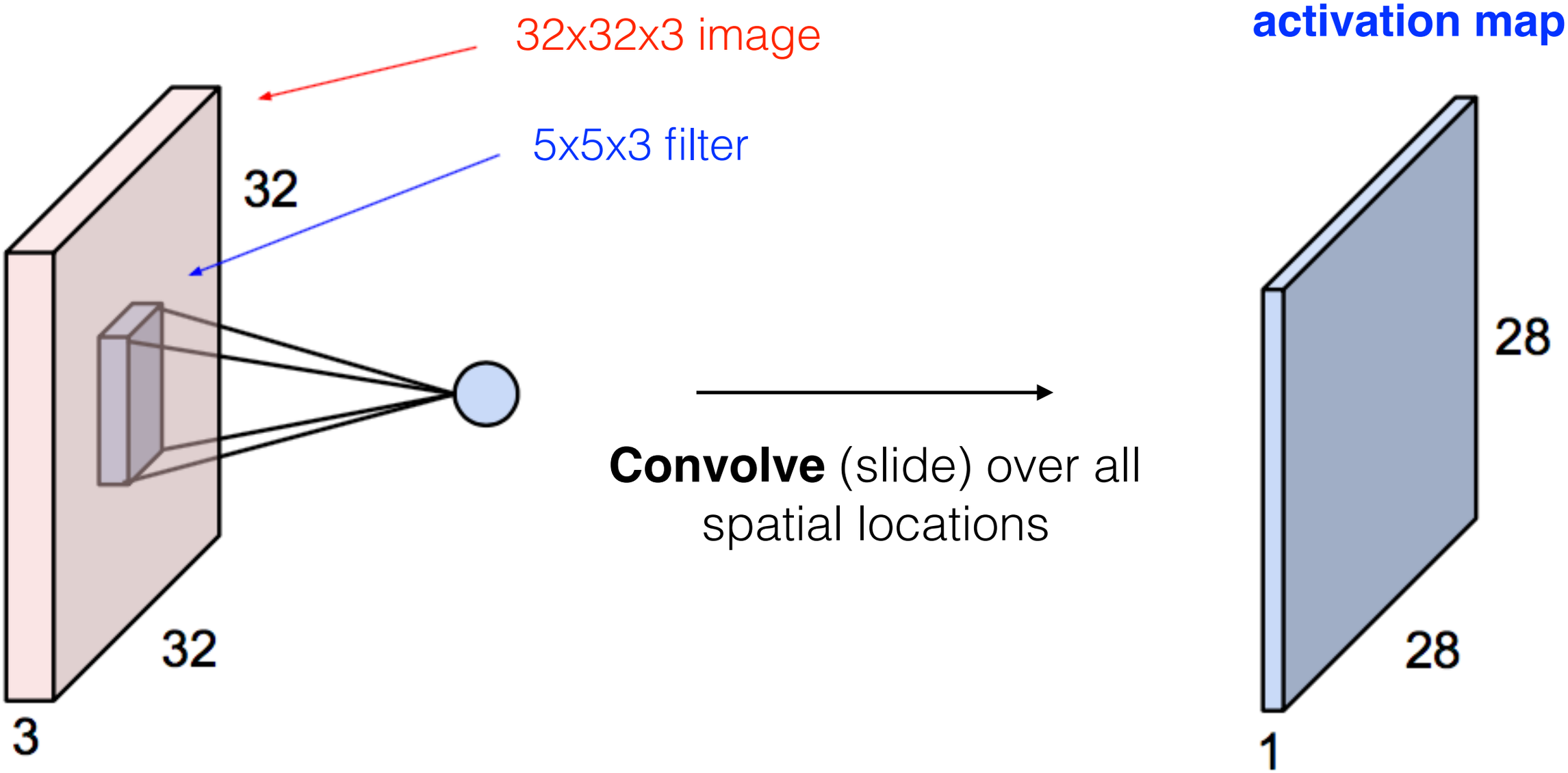
activation: *1 number, i.e. the result of taking a dot product between the filter and a 5x5x3 chunk of the image*

*(i.e. 5*5*3=75-d dot product + bias)*

$$w^T x + b$$

Convolution layer

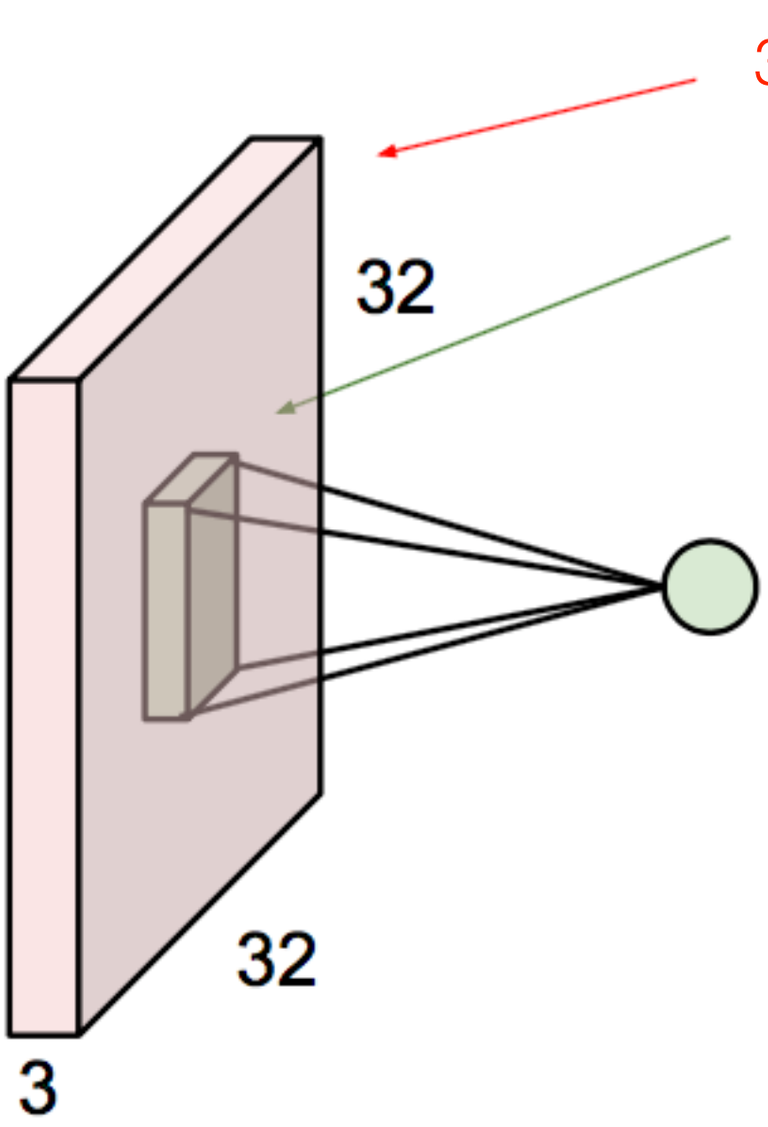
- input: 32x32x3 image -> *preserve spatial structure*



Slide credit: Fei-Fei, Karpathy, Johnson

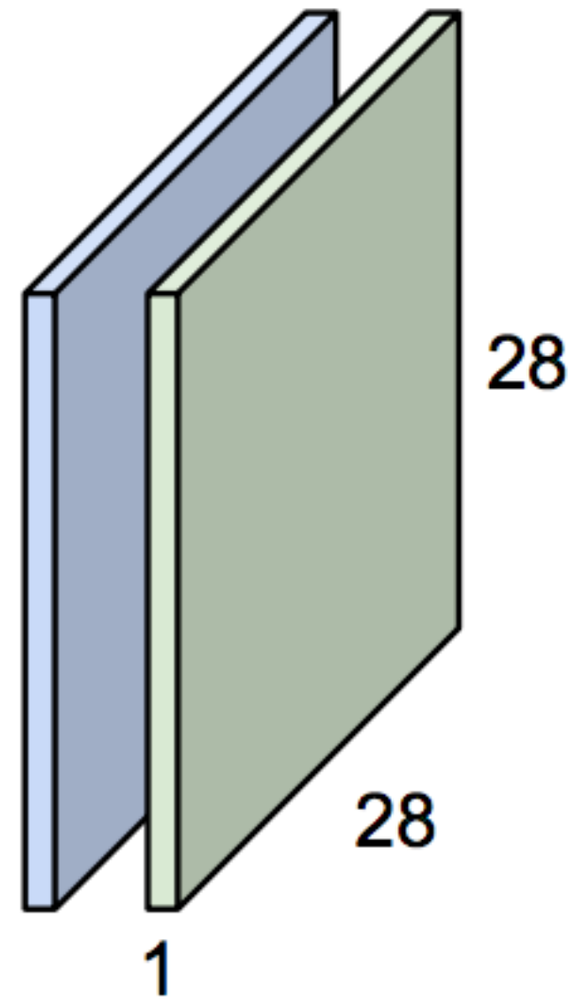
Convolution layer

- Consider a second, green filter



Convolve (slide) over all spatial locations

activation maps



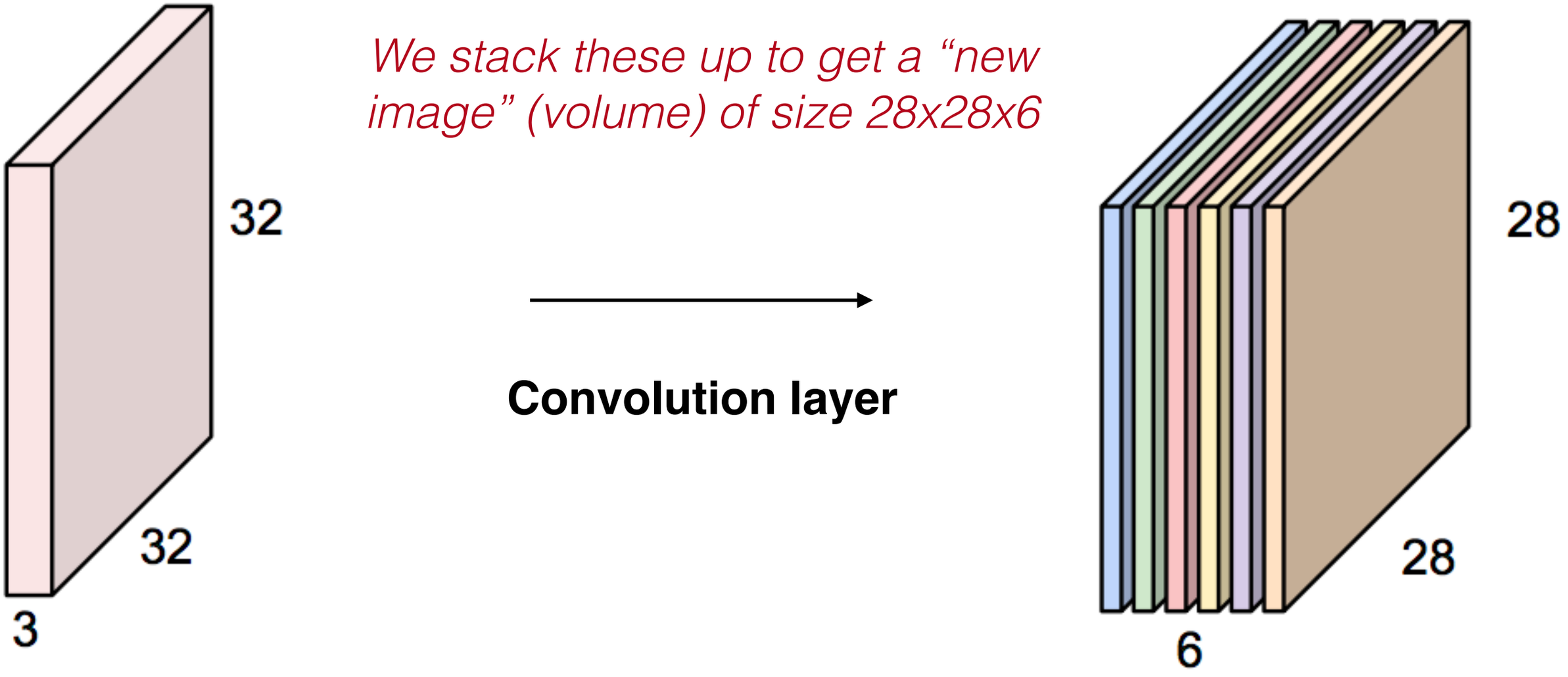
Slide credit: Fei-Fei, Karpathy, Johnson

Convolution layer

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

activation maps

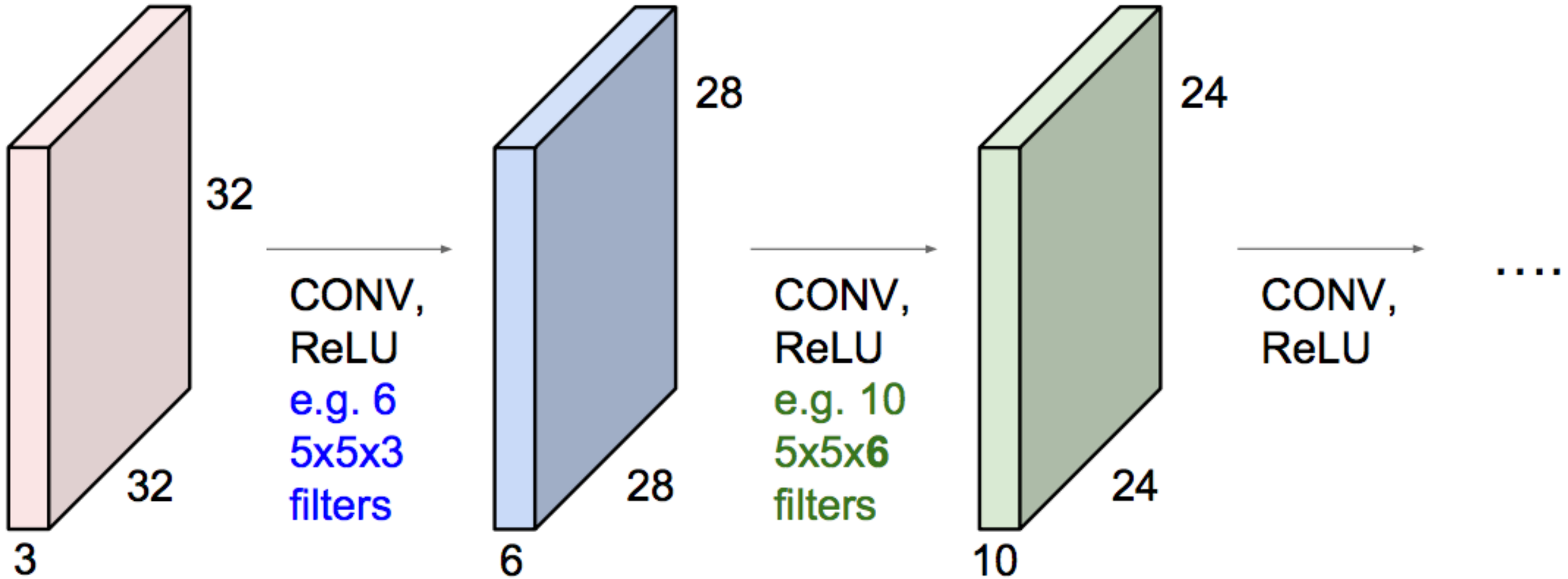
We stack these up to get a "new image" (volume) of size 28x28x6



Slide credit: Fei-Fei, Karpathy, Johnson

Convolutional Neural Networks

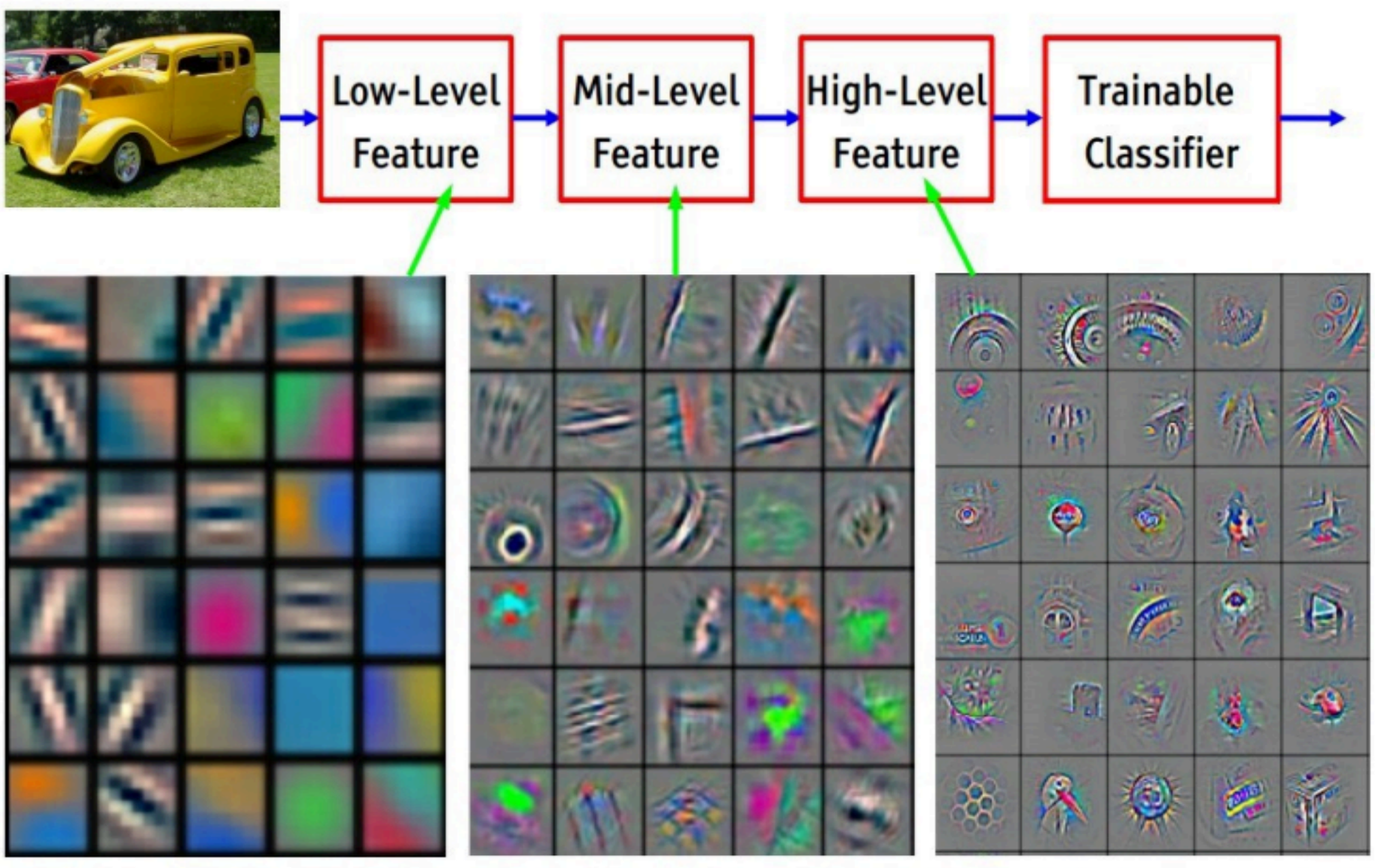
- A ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Slide credit: Fei-Fei, Karpathy, Johnson

Convolutional Neural Networks

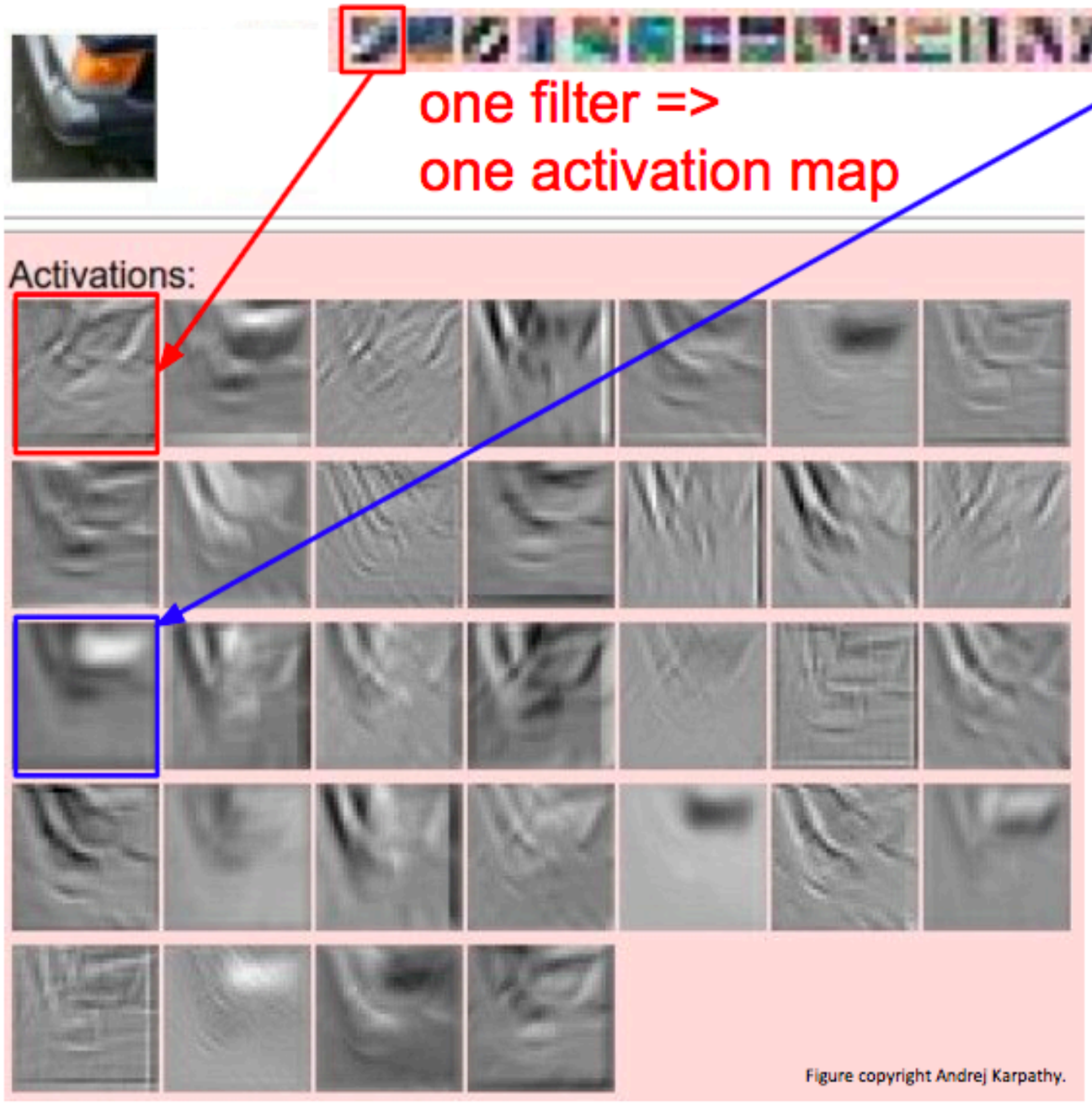
- Preview: feature visualization of CNN trained on ImageNet



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Y. LeCun

Convolutional Neural Networks



example 5x5 filters
(32 total)

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

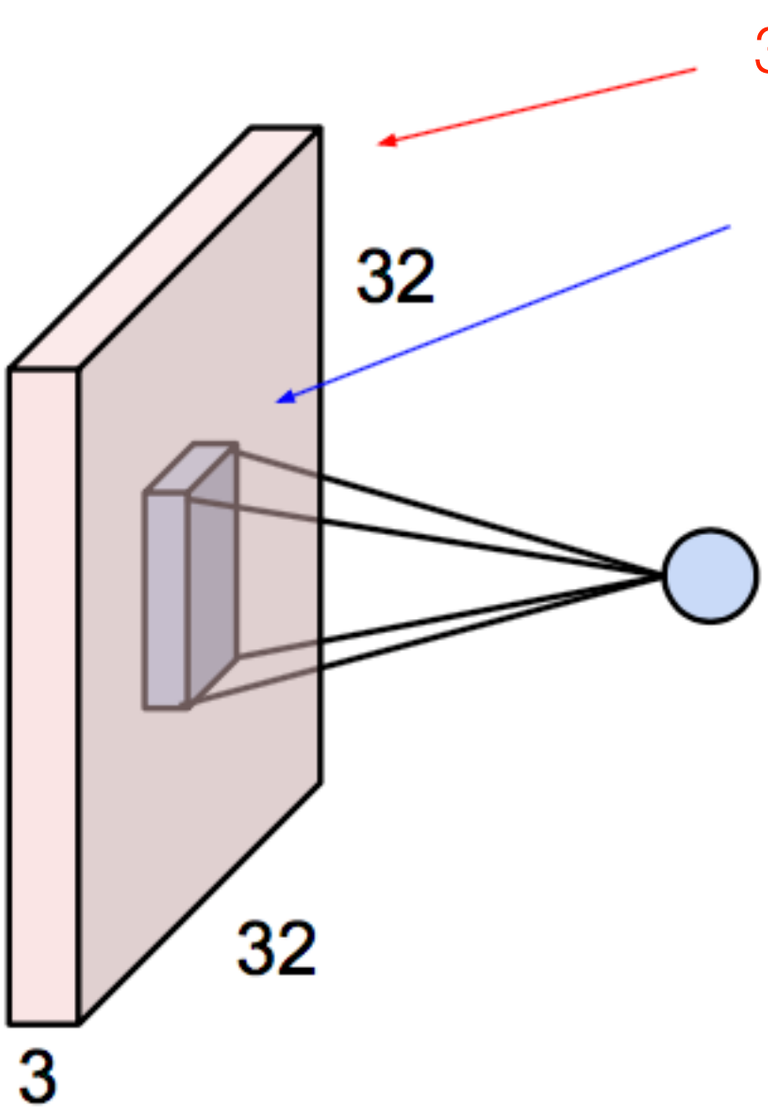
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

Slide credit: Fei-Fei, Karpathy, Johnson

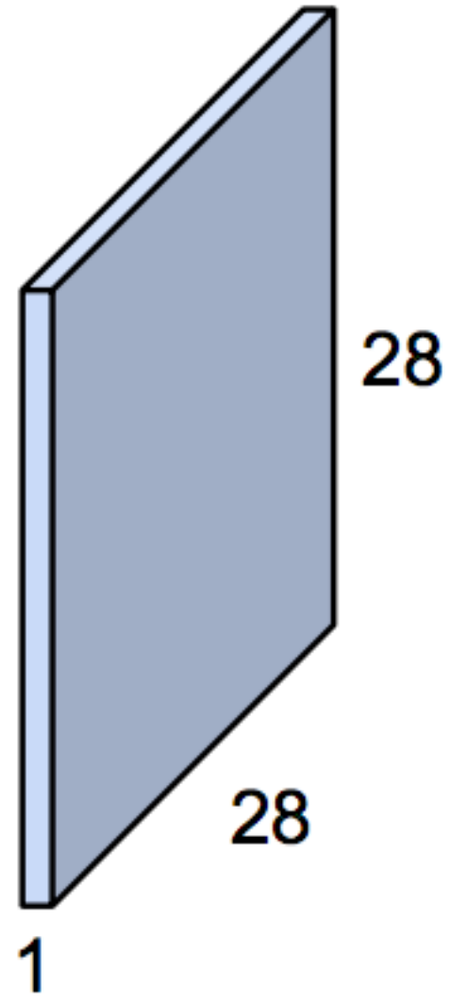
Convolution layer

- A closer look at spatial dimensions:



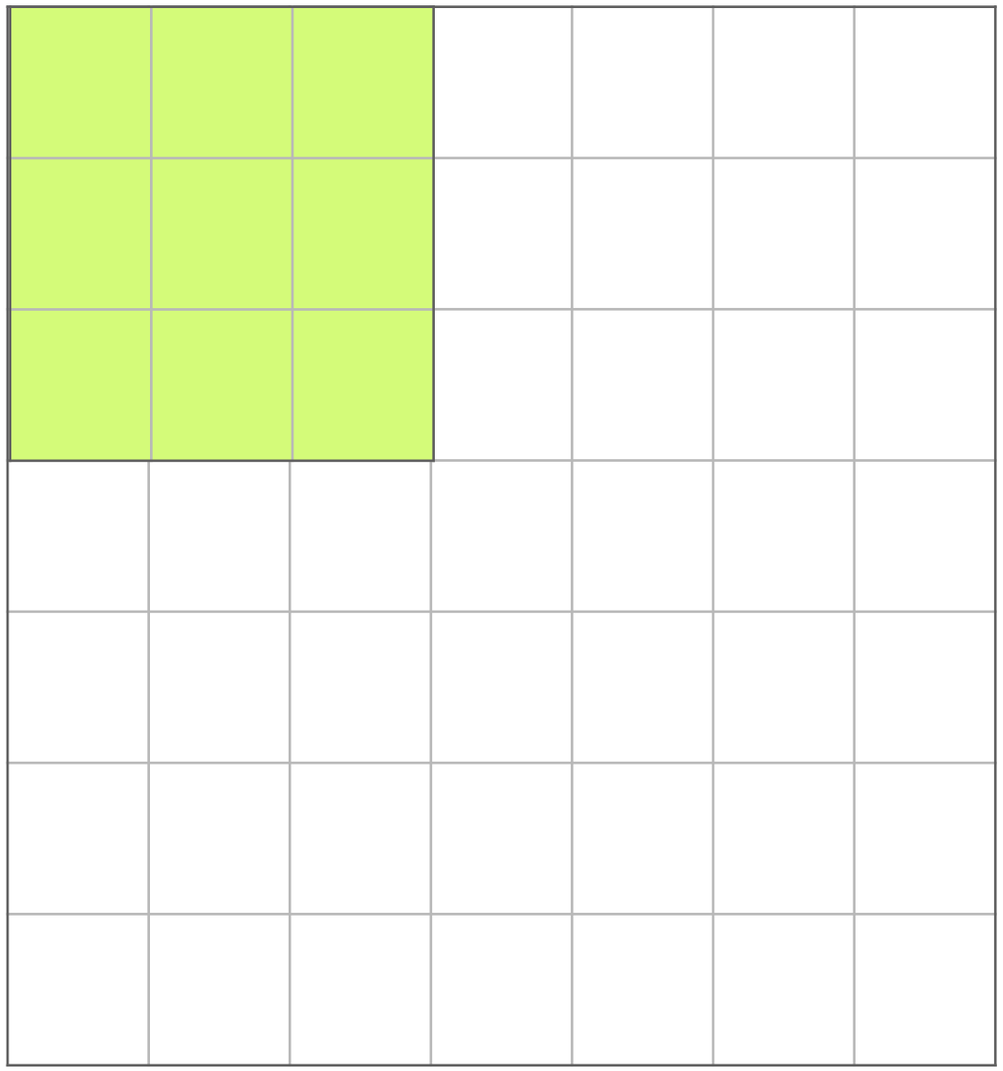
Convolve (slide) over all spatial locations

activation map



Convolution layer

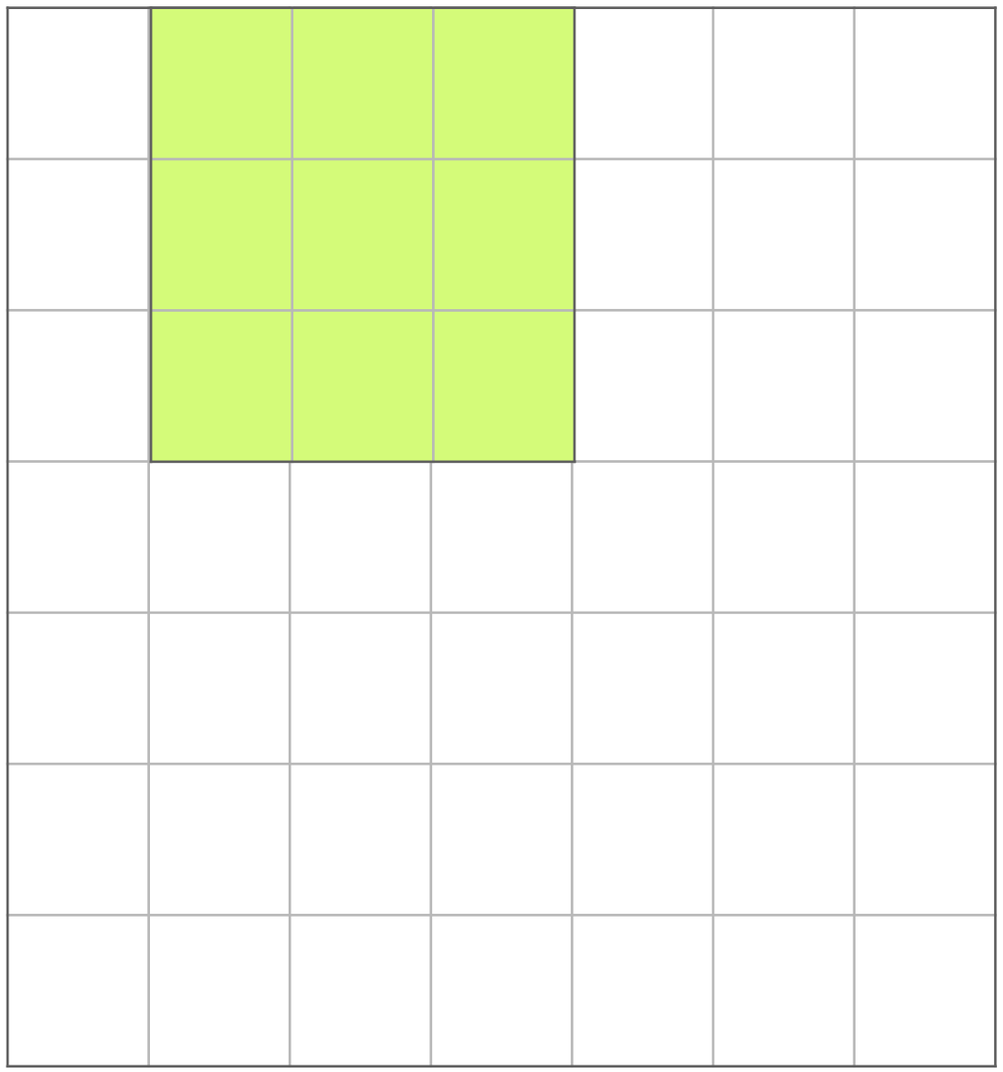
- A closer look at spatial dimensions:



7x7 input (spatially)
Assume 3x3 filter

Convolution layer

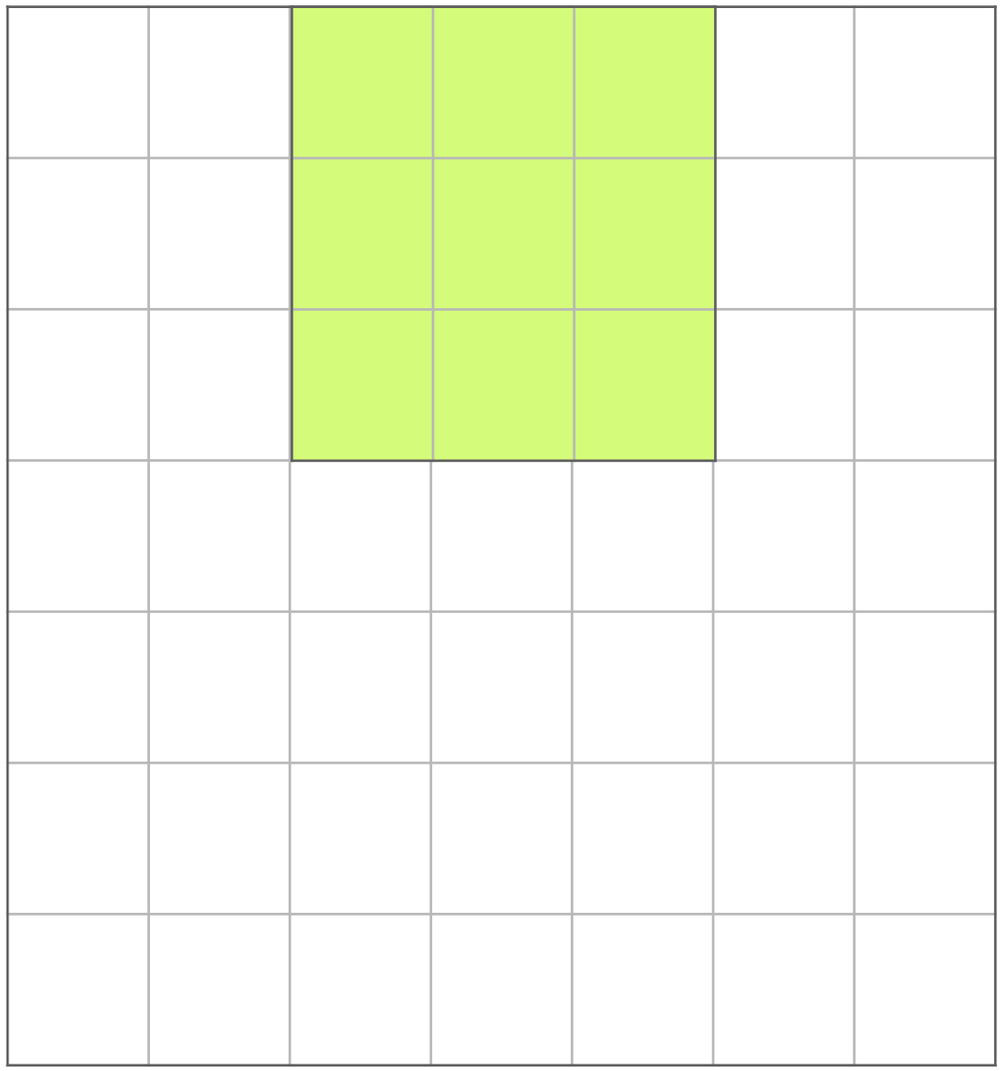
- A closer look at spatial dimensions:



7x7 input (spatially)
Assume 3x3 filter

Convolution layer

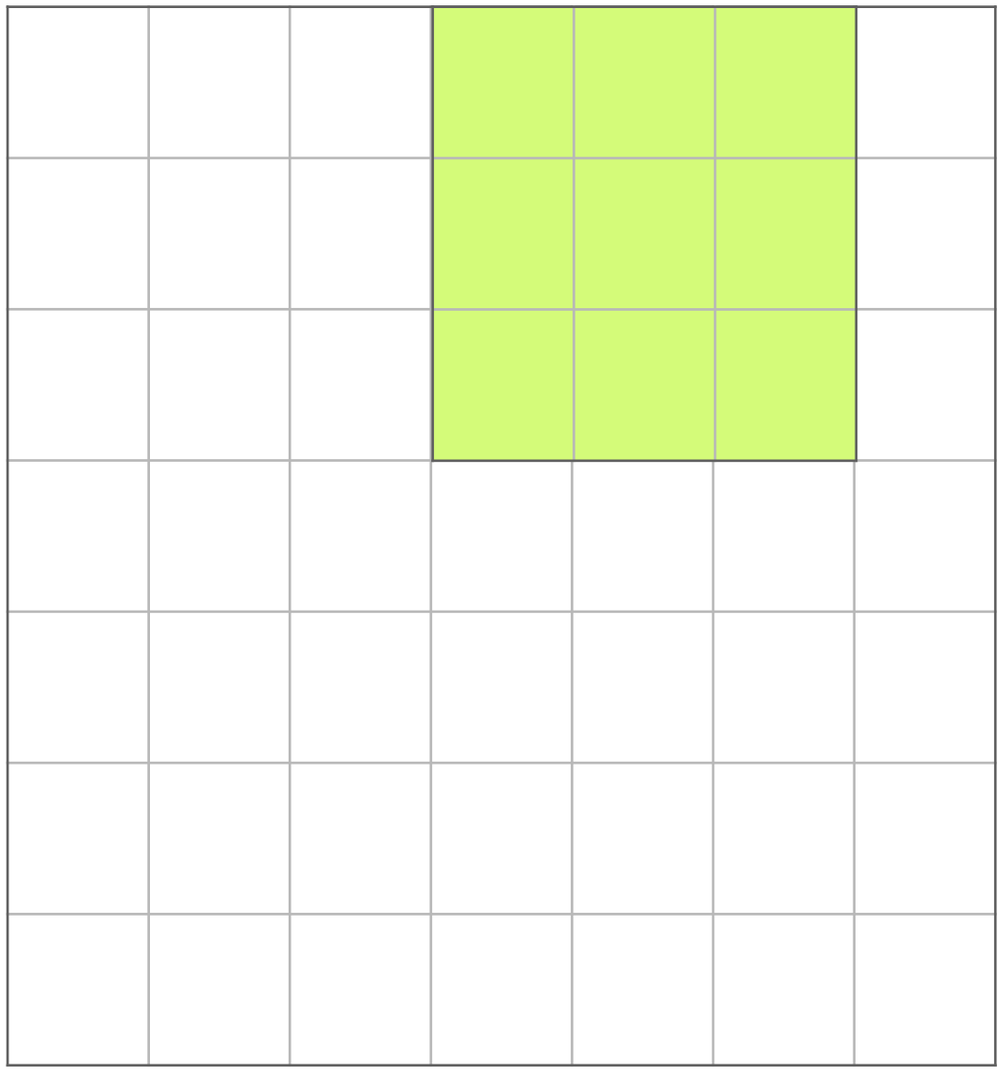
- A closer look at spatial dimensions:



7x7 input (spatially)
Assume 3x3 filter

Convolution layer

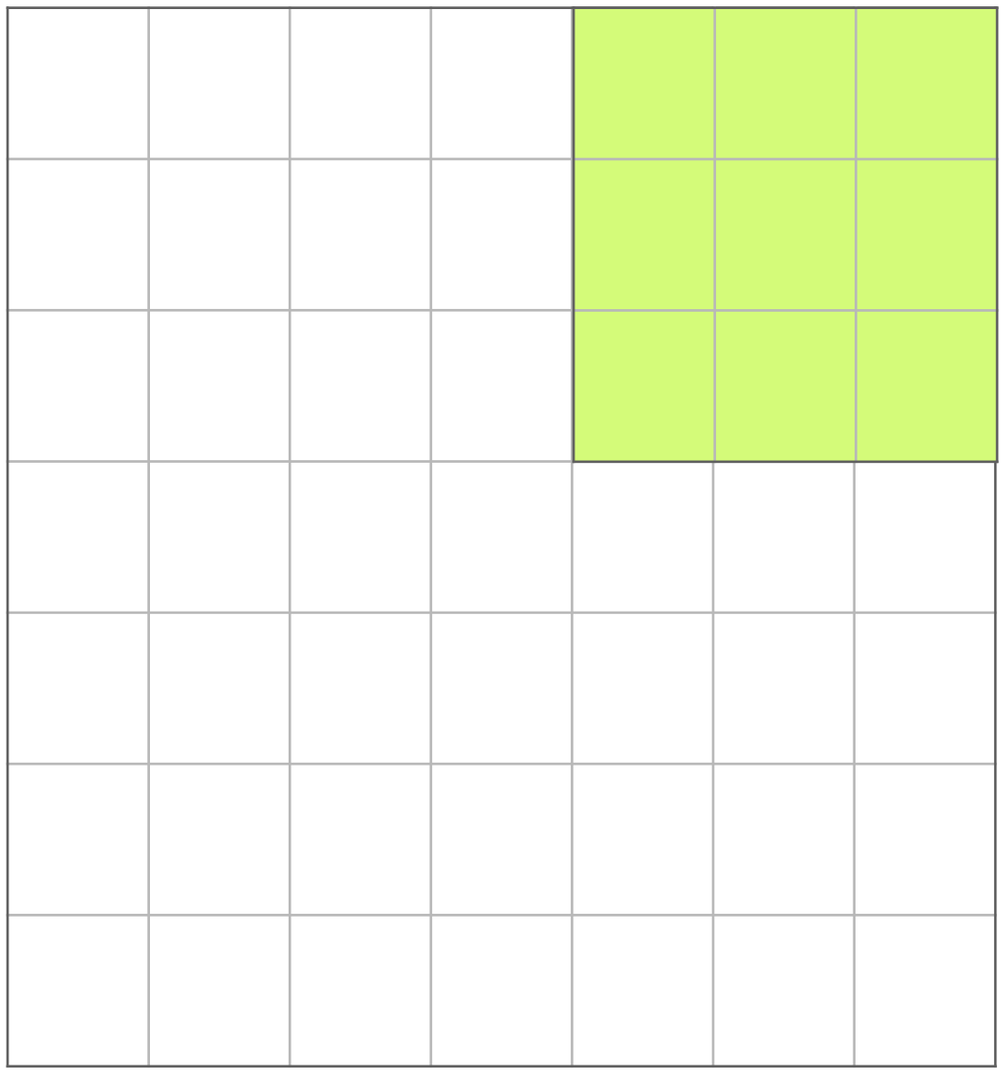
- A closer look at spatial dimensions:



7x7 input (spatially)
Assume 3x3 filter

Convolution layer

- A closer look at spatial dimensions:

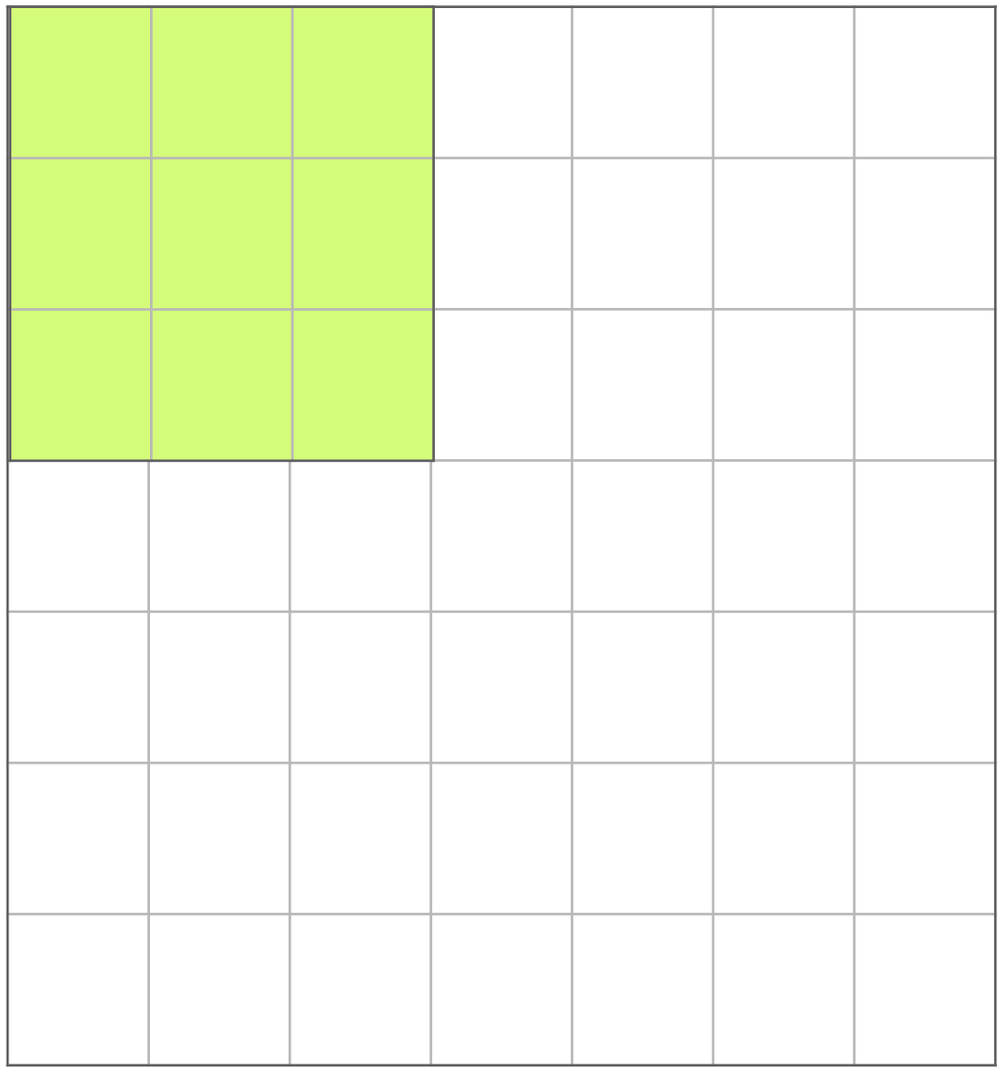


7x7 input (spatially)
Assume 3x3 filter

→ **5x5 output**

Convolution layer

- A closer look at spatial dimensions:



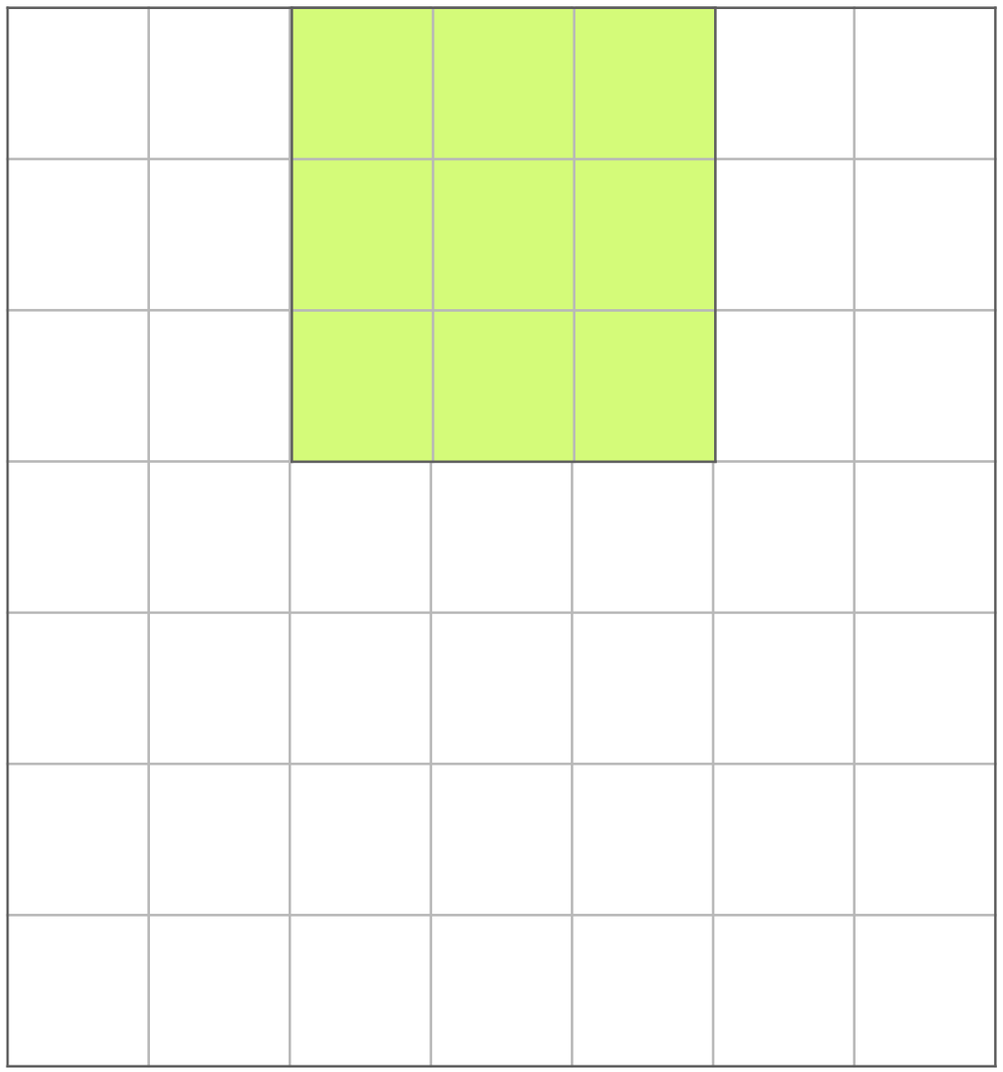
7x7 input (spatially)

Assume 3x3 filter

applied with **stride 2**

Convolution layer

- A closer look at spatial dimensions:

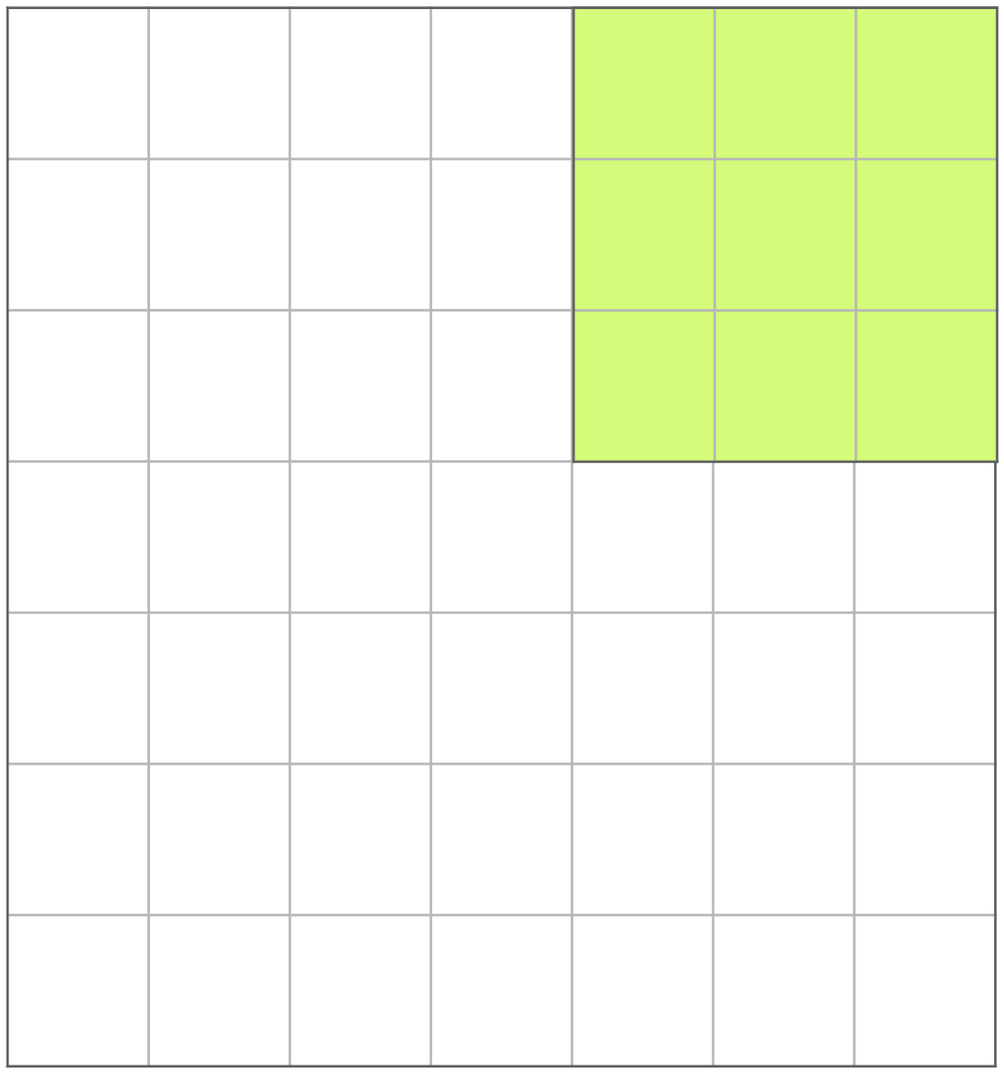


7x7 input (spatially)

Assume 3x3 filter applied with **stride 2**

Convolution layer

- A closer look at spatial dimensions:



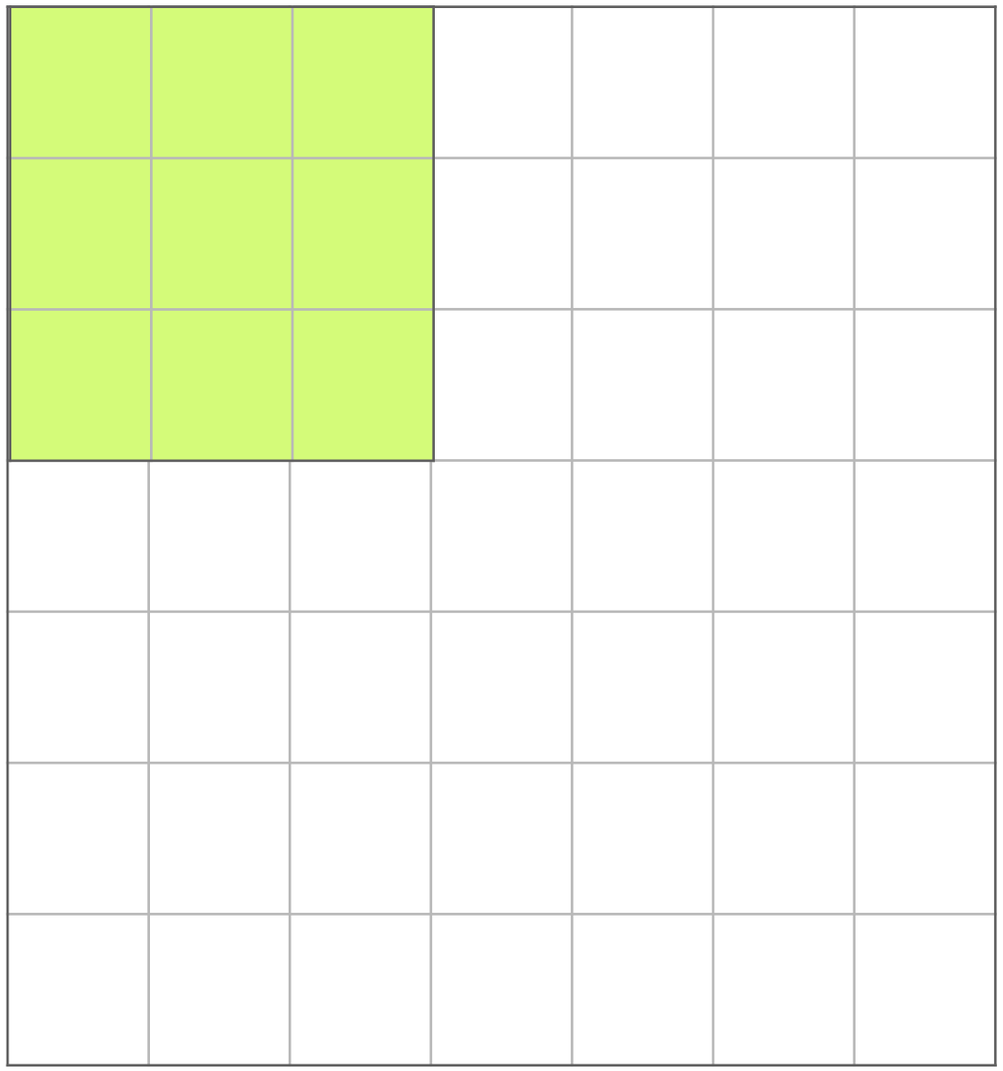
7x7 input (spatially)

Assume 3x3 filter applied with **stride 2**

→ **3x3 output**

Convolution layer

- A closer look at spatial dimensions:



7x7 input (spatially)

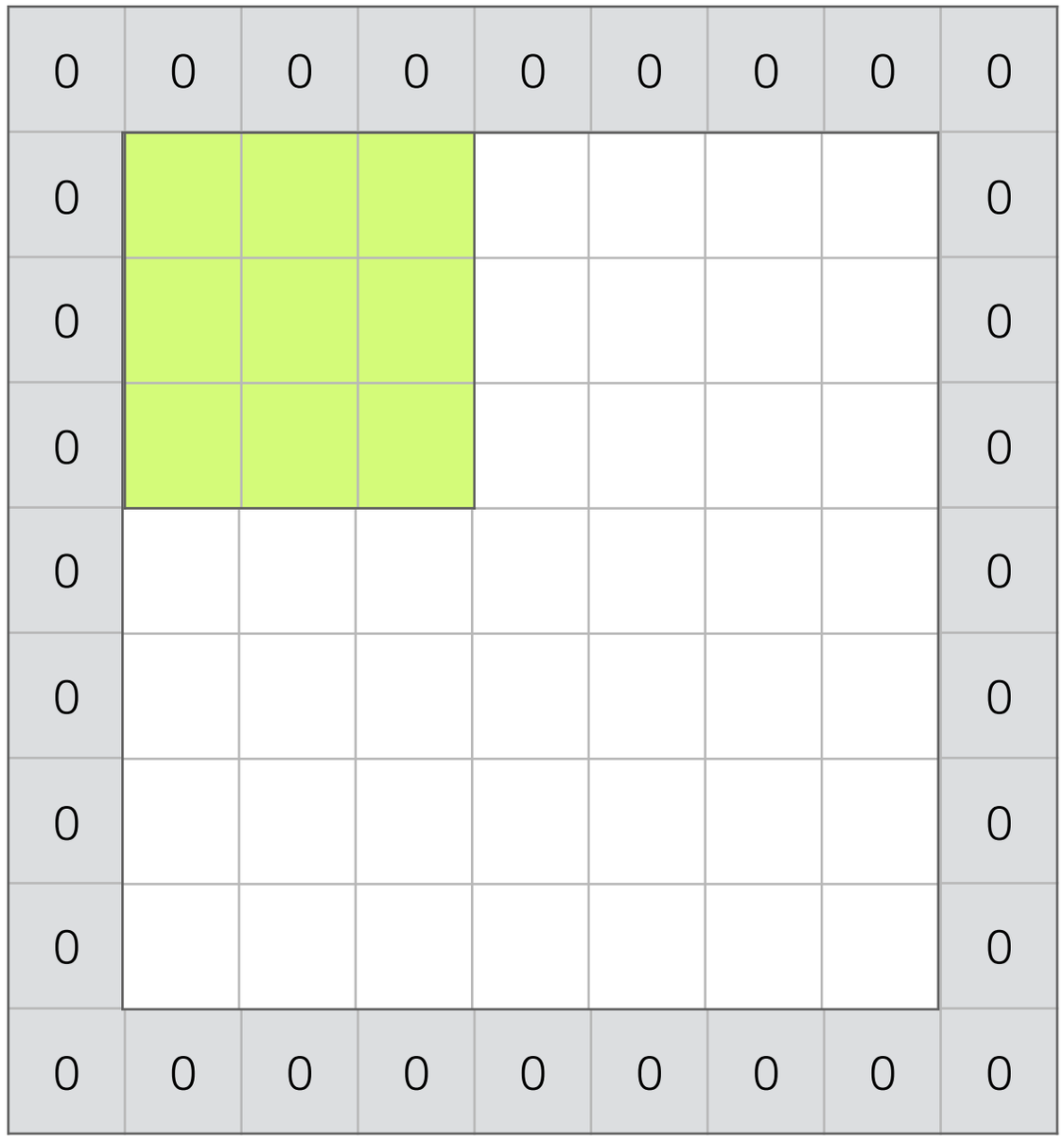
Assume 3x3 filter
applied with **stride 3?**

It doesn't fit!

Cannot apply 3x3 filter
on 7x7 input with stride 3

Convolution layer

- Padding: it's common to zero pad the border



7x7 input (spatially)

Assume 3x3 filter applied with **stride 3**

Pad with 1 pixel border; what is the output?

→ **3x3 output**

In general is common to see CNNs with stride 1, filters of size FxF, and zero-padding with (F-1)/2 (this will preserve size spatially)

Convolution layer: spatial arrangement

- Three hyper-parameters control the size of the output volume: the *depth*, *stride* and *zero-padding*

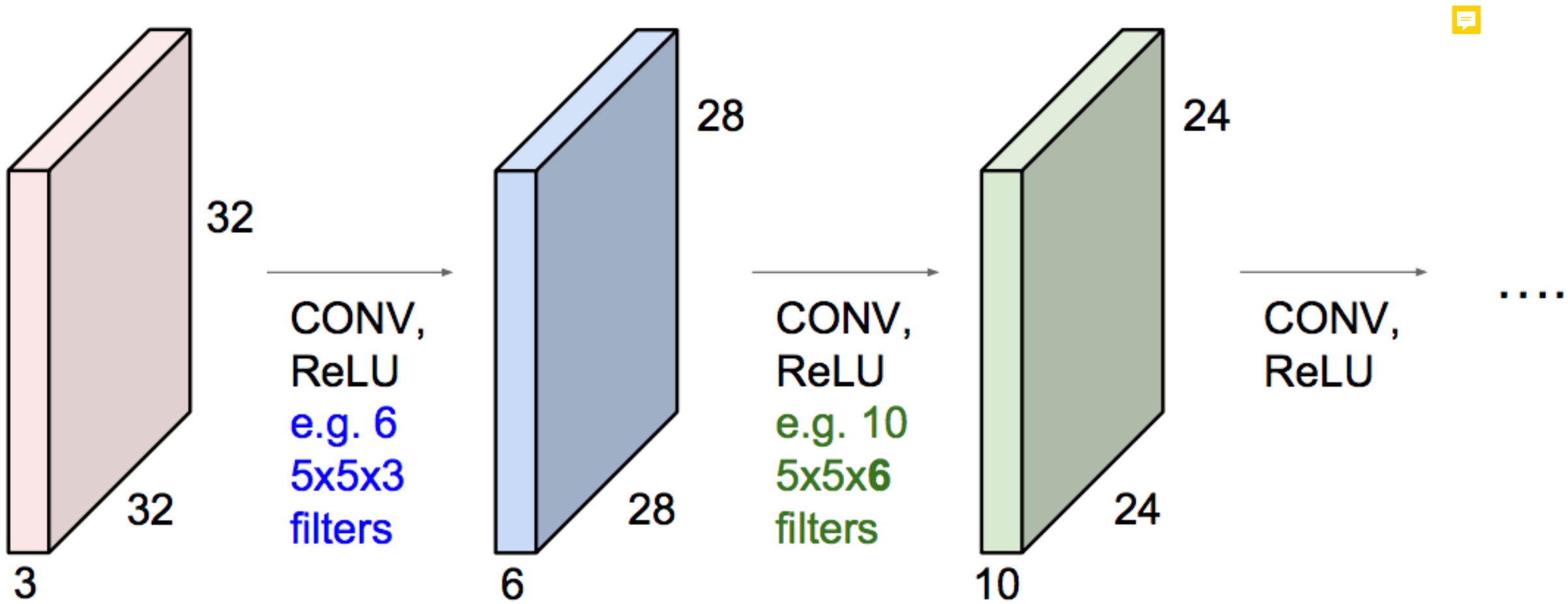
- Output volume size: *(W-F+2*P)/S+1*
  - *W*: input volume size
  - *F*: filter size (commonly referred as to *receptive field*)
  - *S*: the stride with which filter(s) are applied
  - *P*: the amount of zero padding used on the border

Previous example:

*7x7 input, 3x3 filter, stride 1 and pad 0 => 5x5 output, i.e. (7-3+2*0)/1+1=5*

Convolutional Neural Networks

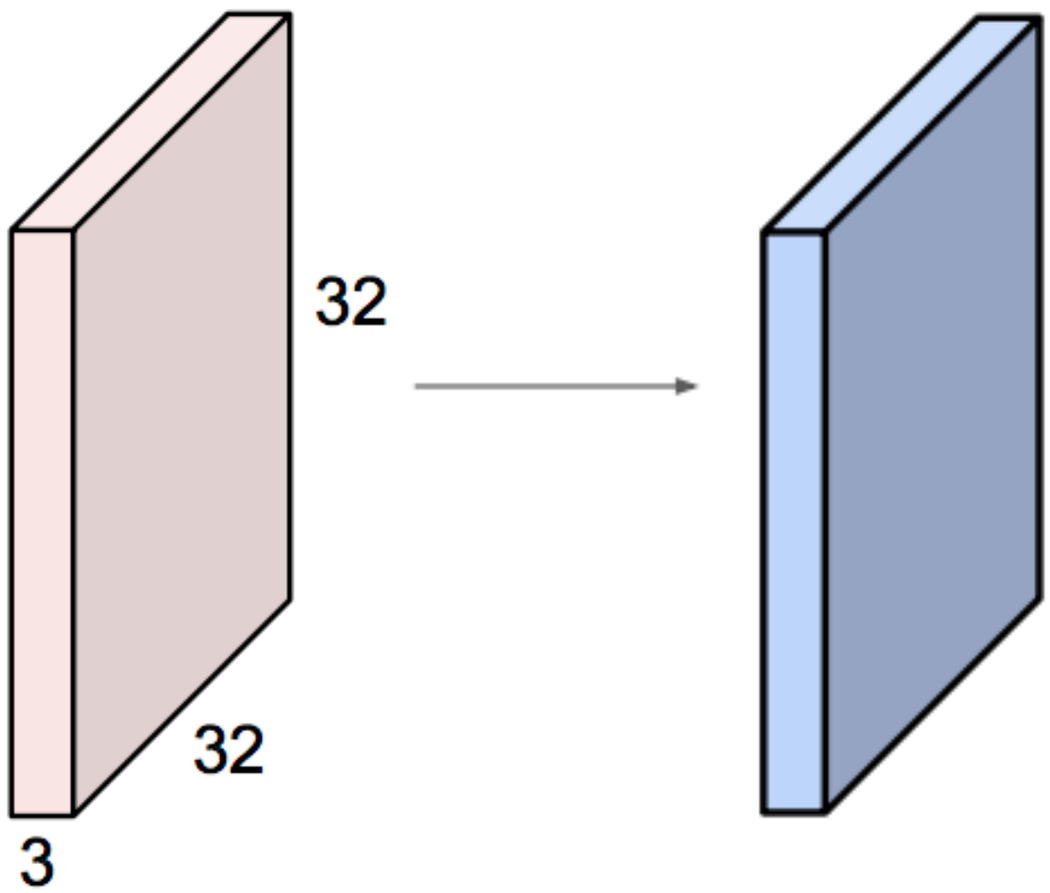
- Remember: 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially (i.e. 32, 28, 24, …)



Shrinking too fast is bad. It doesn't work well.

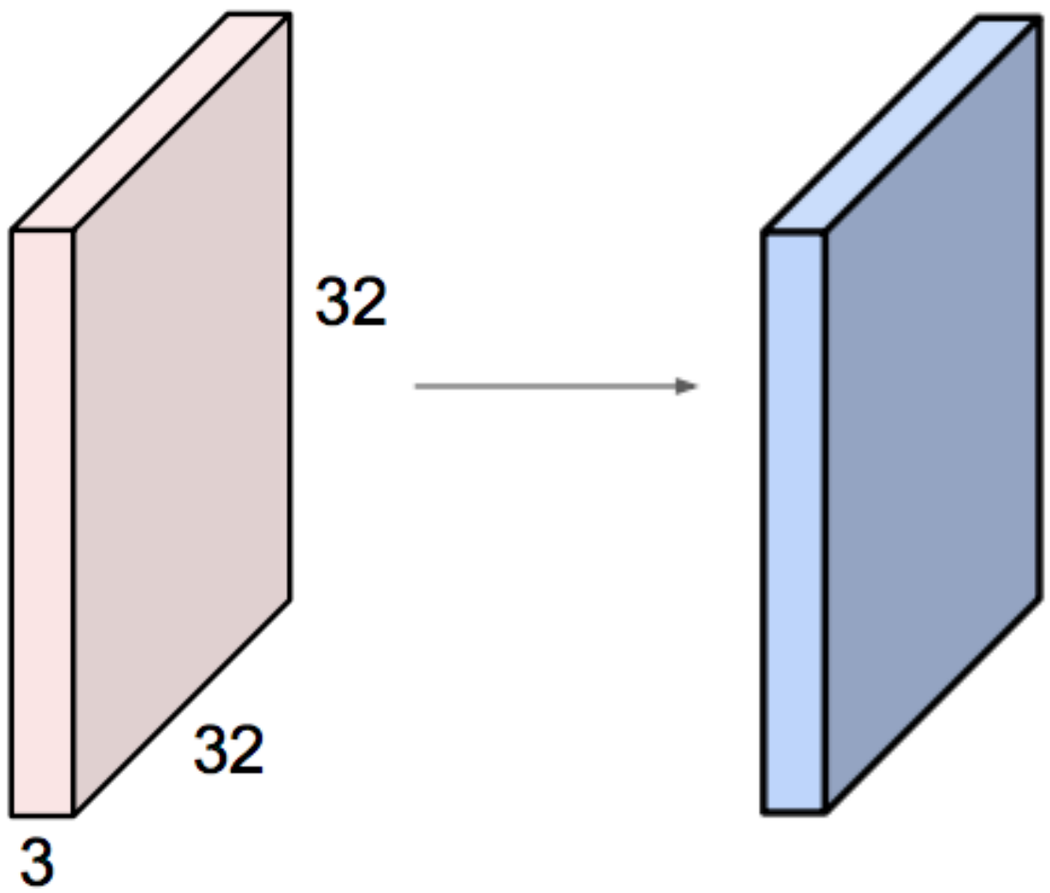
Examples

- Input volume: **32x32x3**; 10 5x5(x3) filters with stride 1, pad 2
- Output volume: **?**



Examples

- Input volume: **32x32x3**; 10 5x5(x3) filters with stride 1, pad 2
- Output volume: (32+2*2-5)/1+1 = 32 spatially => **32x32x10**



How many learnable parameters in this layer?

Each filter has: 5*5*3 + 1(bias) = 76 params
=> 76*10 = **760**

Examples

- A conv layer in TensorFlow:

Convolutional Layer #1

In our first convolutional layer, we want to apply 32 5x5 filters to the input layer, with a ReLU activation function. We can use the `conv2d()` method in the `layers` module to create this layer as follows:

```
conv1 = tf.layers.conv2d(
    inputs=input_layer,
    filters=32,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
```

The `inputs` argument specifies our input tensor, which must have the shape `[batch_size, image_height, image_width, channels]`. Here, we're connecting our first convolutional layer to `input_layer`, which has the shape `[batch_size, 28, 28, 1]`.

> **Note:** `conv2d()` will instead accept a shape of `[batch_size, channels, image_height, image_width]` when passed the argument `data_format=channels_first`.

The `filters` argument specifies the number of filters to apply (here, 32), and `kernel_size` specifies the dimensions of the filters as `[height, width]` (here, `[5, 5]`).

Examples

- A conv layer in Caffe:

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Coming up

- **Tuesday, May 14** (next lecture)**:**
  - Convolutional Neural Networks - Part 2