Data Mining & Machine Learning

CS37300
Purdue University

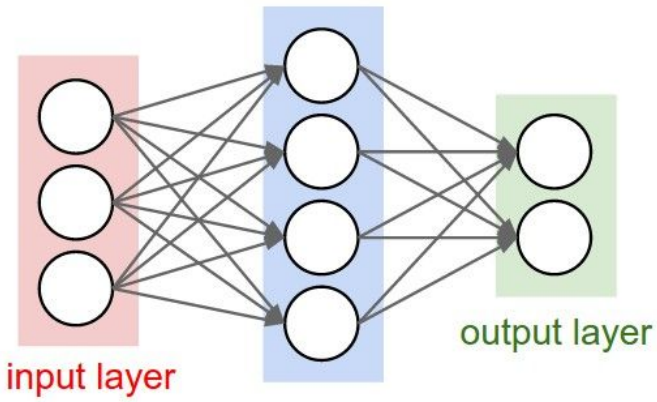
Oct 25, 2023

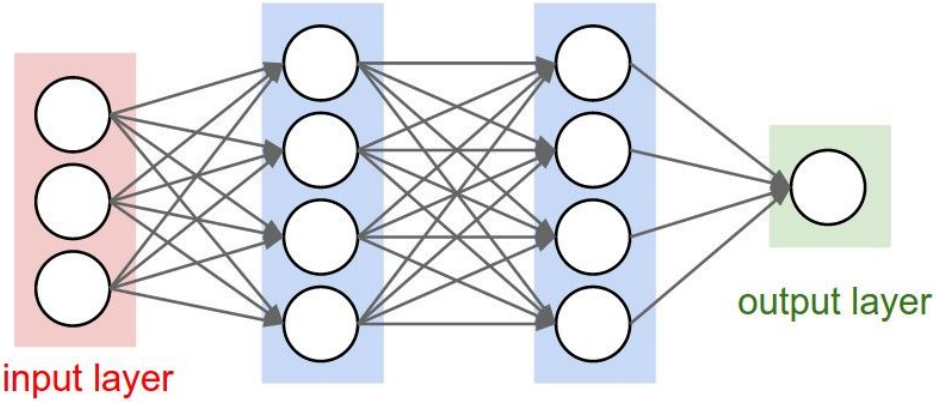
Convolutional Neural Networks

Overview

- Deep Learning Architectures
    - Feedforward neural networks
    - Convolutional neural networks
        - Filters

Feedforward Neural Networks





"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Ack: Fei-Fei Li & Andrej Karpathy & Justin Johnson

- Layers do not need to be fully connected
- Size of layer (number of units) is another hyperparameter

Convolutional Neural Networks (Motivation)

- Feedforward networks has input as a vector

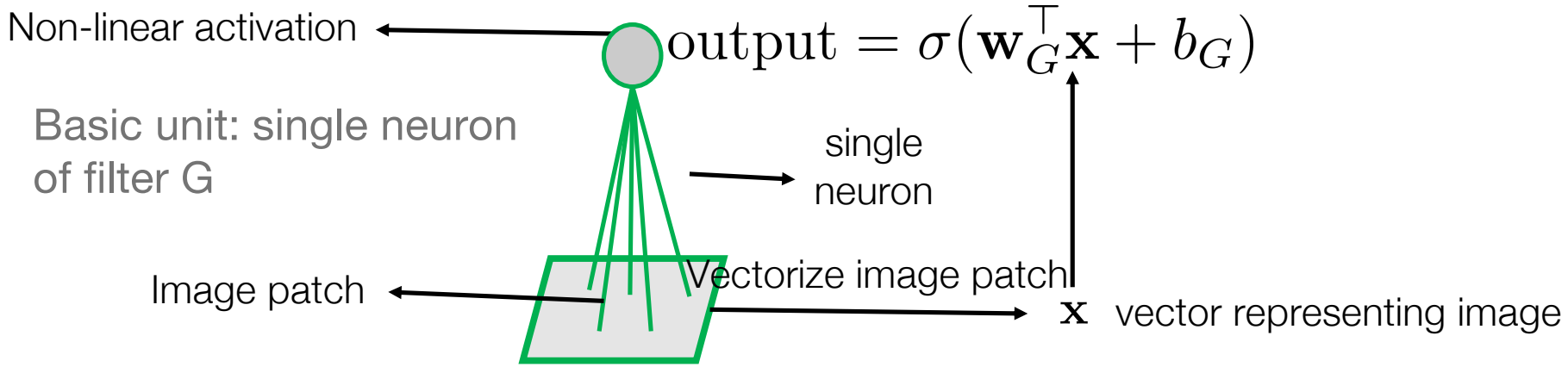
- From image to vector… hard to account for spatial correlations in the vector representation (which pixels are next to each other?)

Convolution function

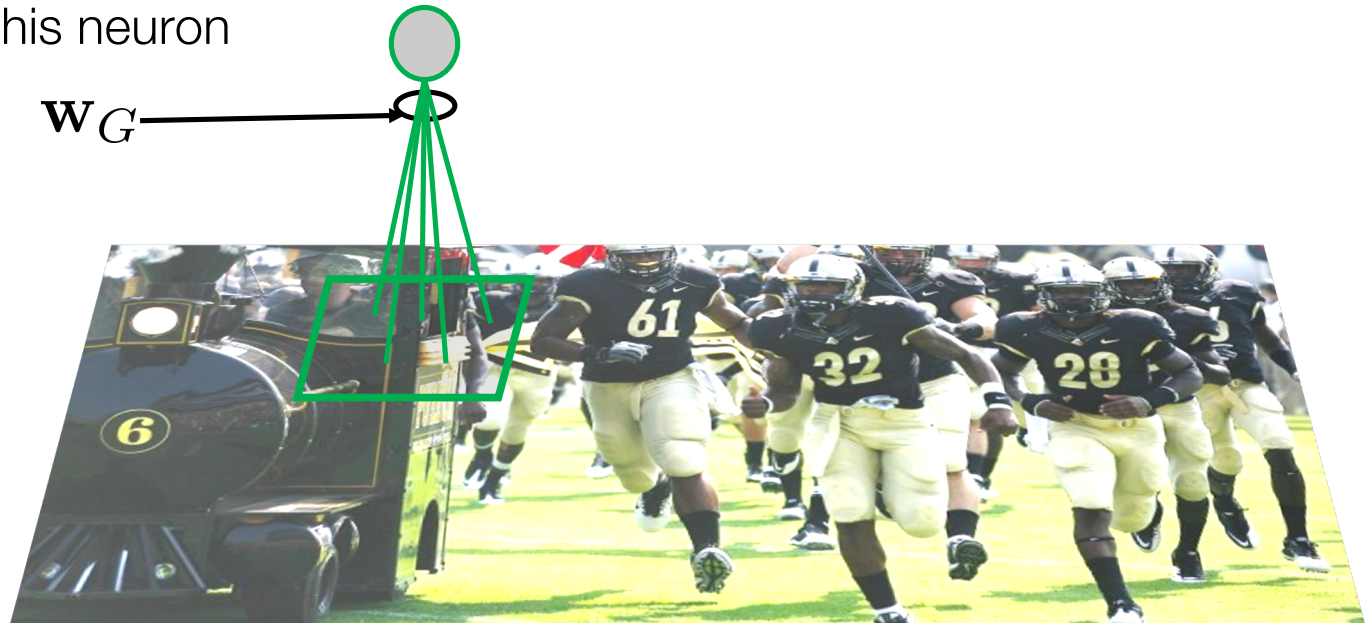
- On the real line: $s(t) = \int x(a)w(t-a)da$,
  - Averaged measurement across several t
  - x -> input
  - w -> kernel
  - s -> feature map
- Discrete variant: $s(t) = \sum_{a=l}^{a=u} x(a)w(t-a)$
- Easy to generalize to multidimensional x

Convolutional Neural Network (CNN)

"Filter" Transforms large image patch into one output



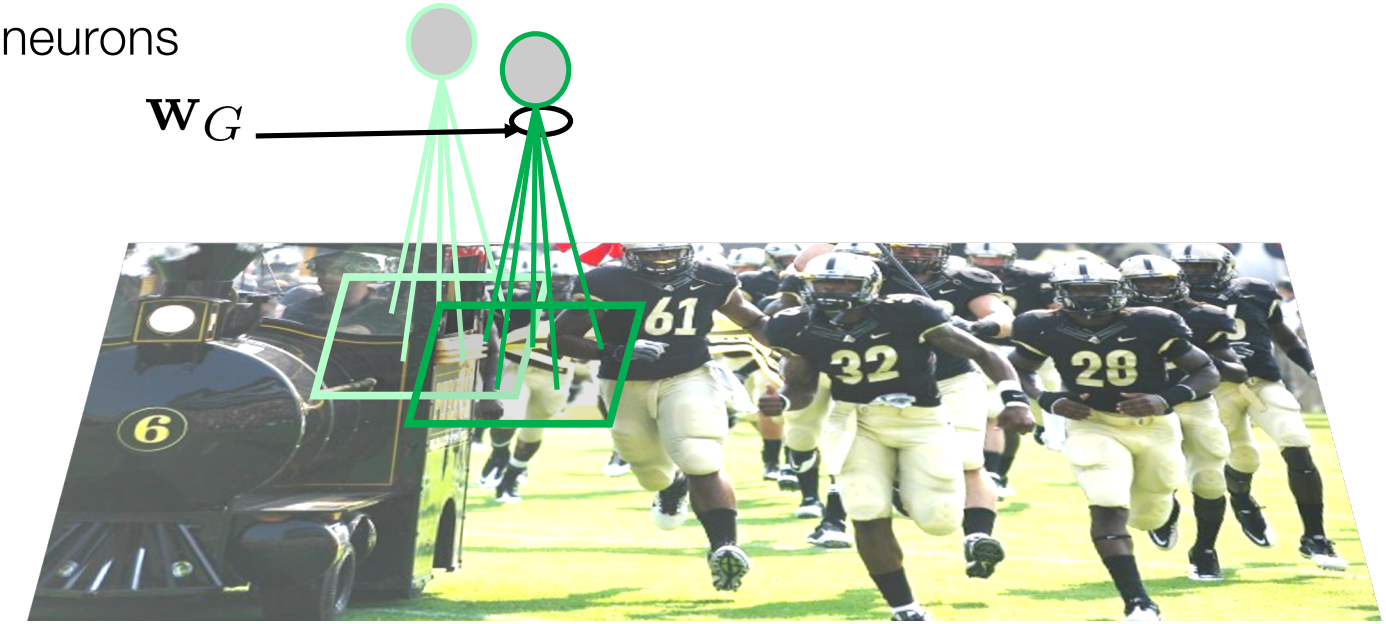
Weights used by this neuron



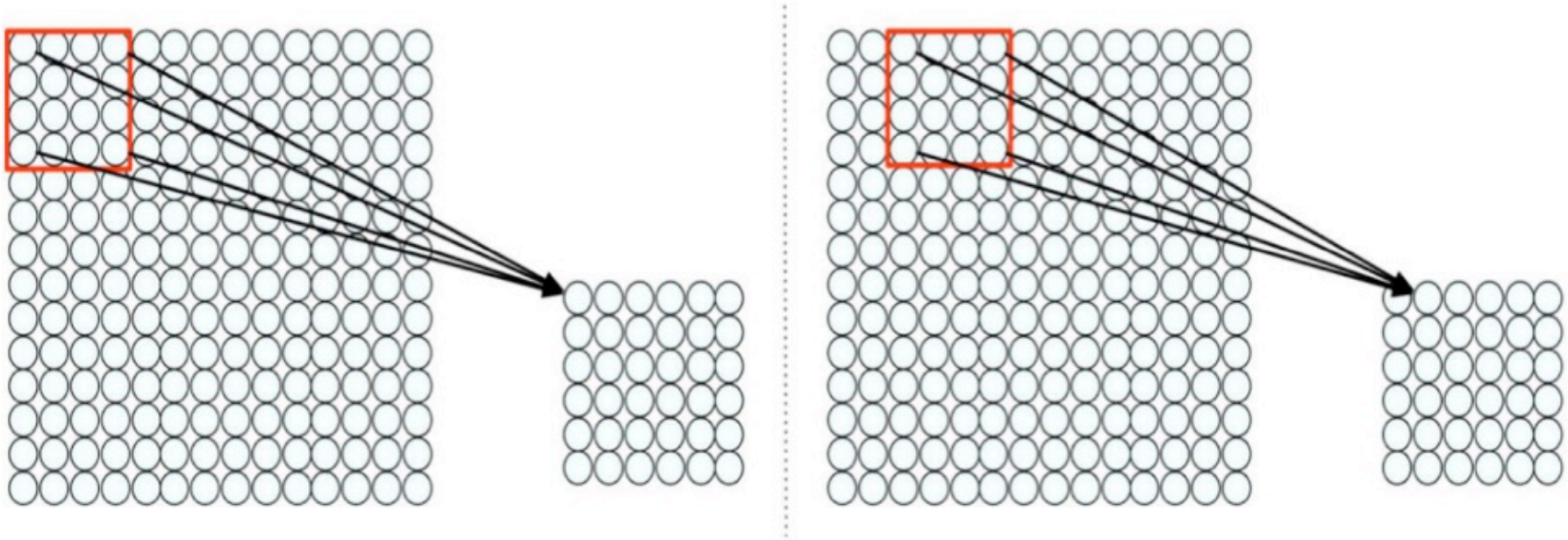
Convolutional Neural Network (CNN)

Cover the rest of the image by sliding the filter

Keep the same weights,
just slide the filter around
to make different neurons



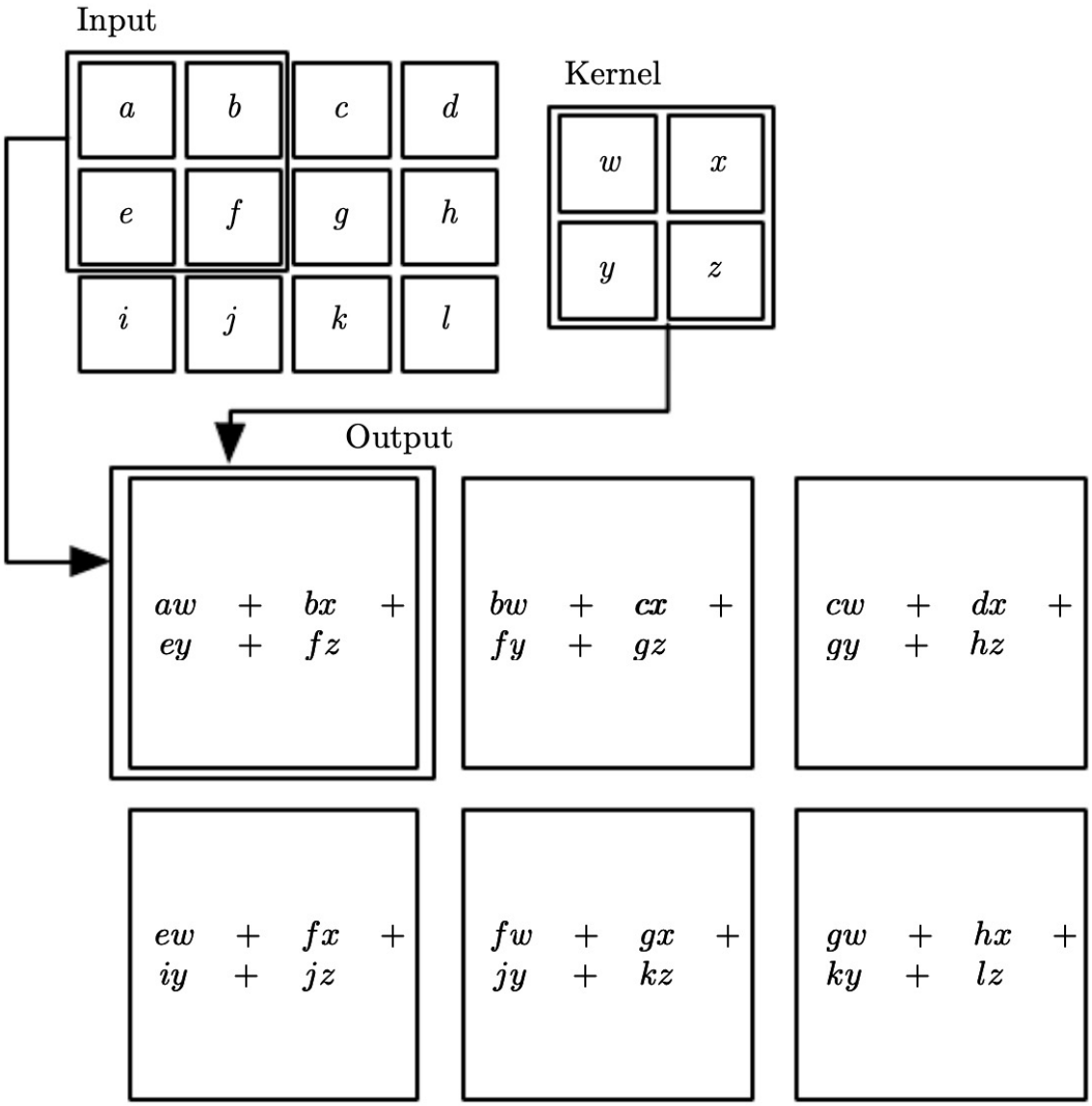
Convolutional Neural Network (CNN)



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

d-dimensional convolution

- Inputs are given, need to learn w,x,y,z



Convolutional Neural Network (CNN)

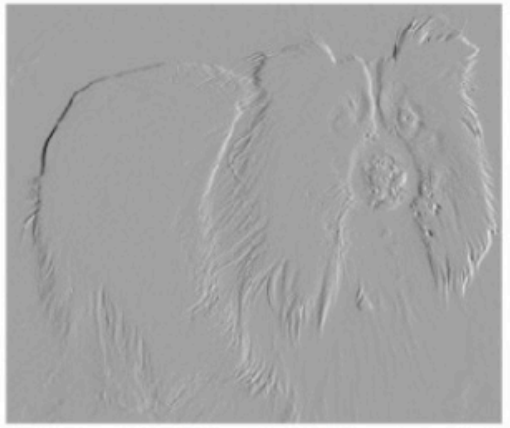
A simple edge detector with convolutions

Input

1x2 patch,
Weights =
(b=0)

1	-1

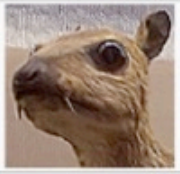
Filter



Output

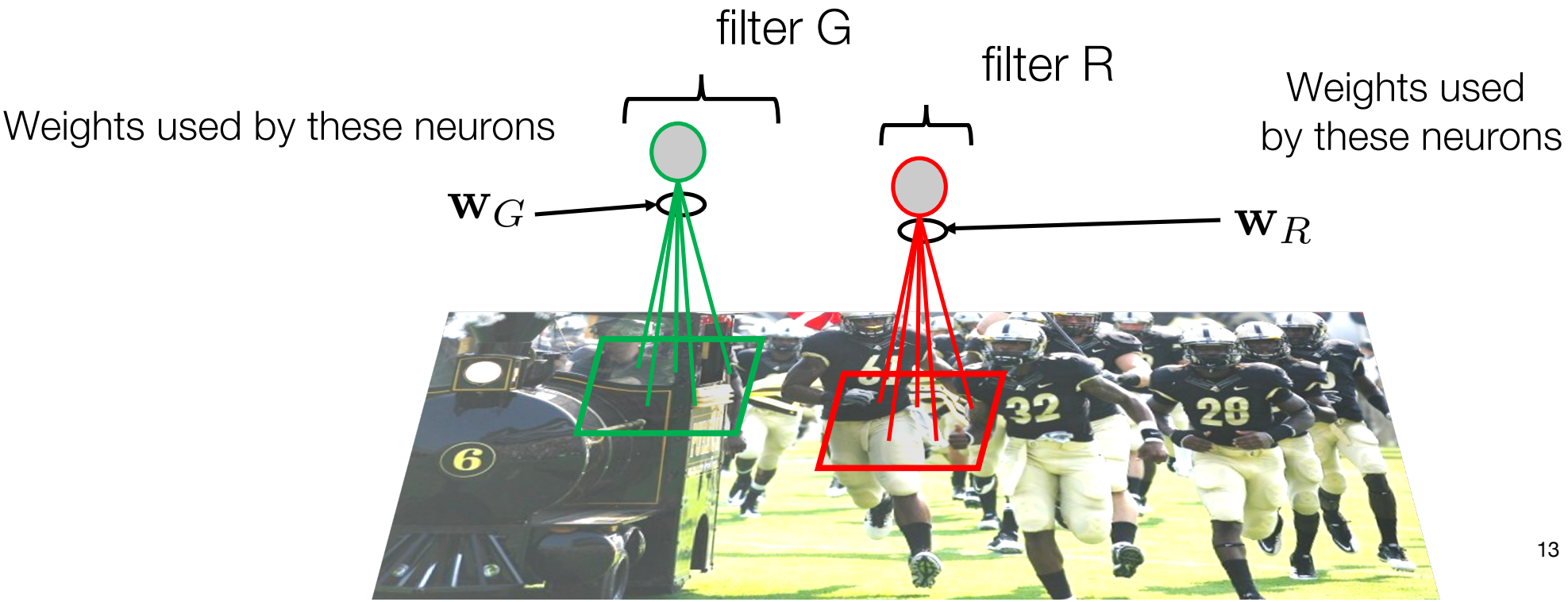
Convolutional Neural Network (CNN)

A few other examples

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolutional Neural Network (CNN)

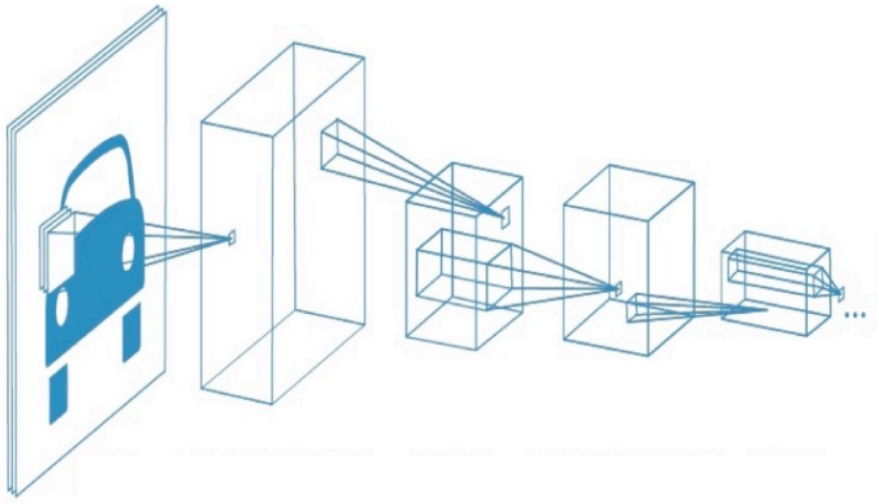
- Extracting more than one kind of feature: We can use more than one filter to create the next layer



Convolutional Neural Network (CNN)

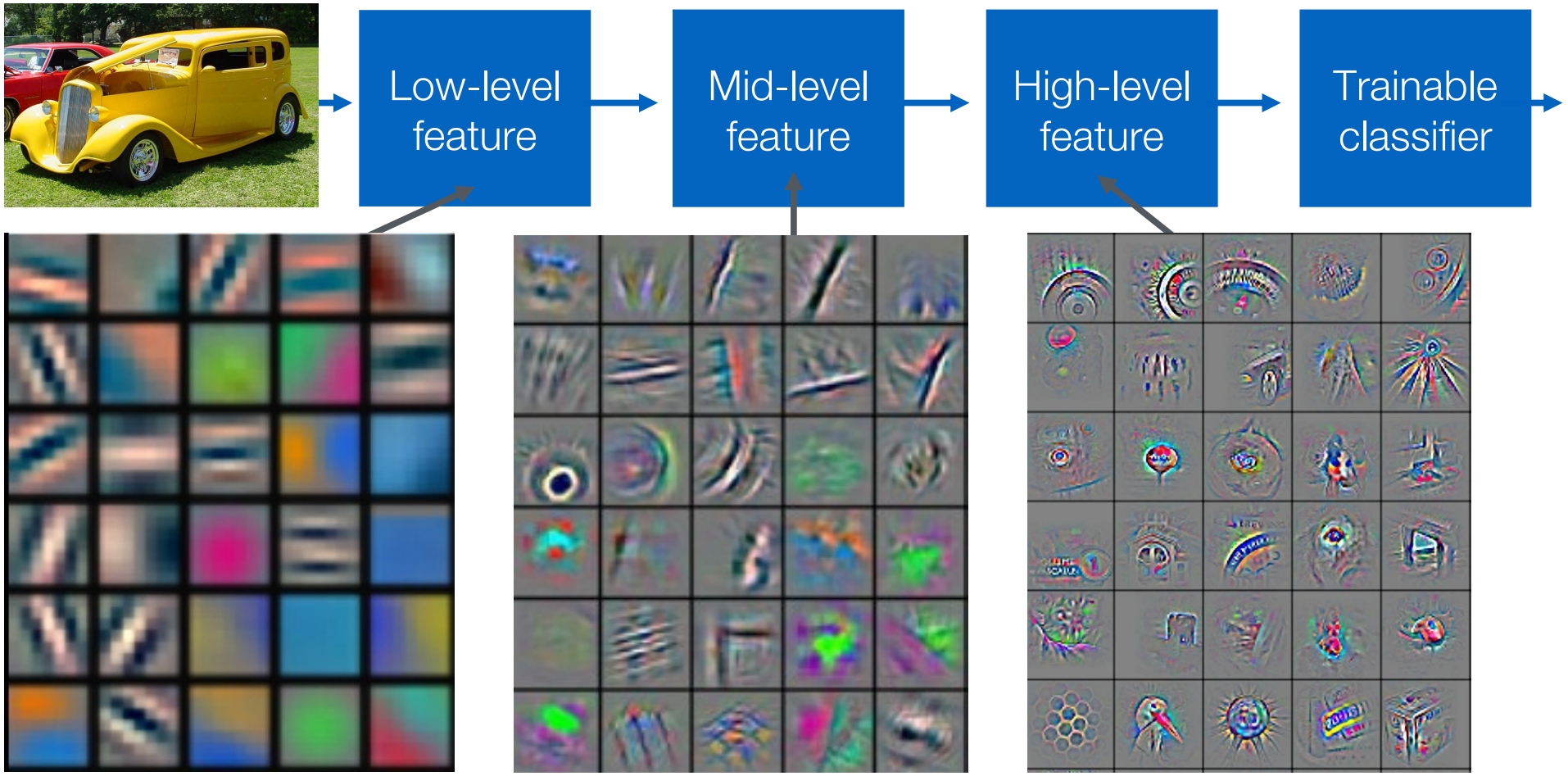
We can also “stack” convolutional layers.

With multiple filters per layer, this can create complex pattern detectors



Deep learning = learning hierarchical representations

It's deep if it has more than one stage of non-linear feature transformation



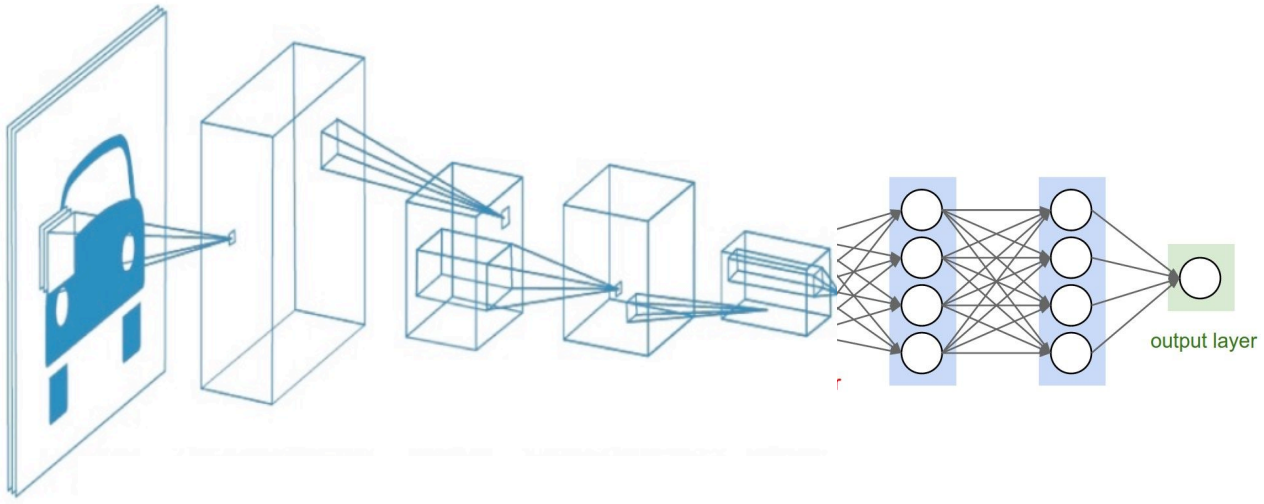
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Credit: Y. LeCun, Facebook

Convolutional Neural Network (CNN)

Typically, we would have some initial convolutional layers, and the last few layers are fully-connected (as in the previous lecture)

Intuition:
The convolutional layers output informative features for the classifier at the end



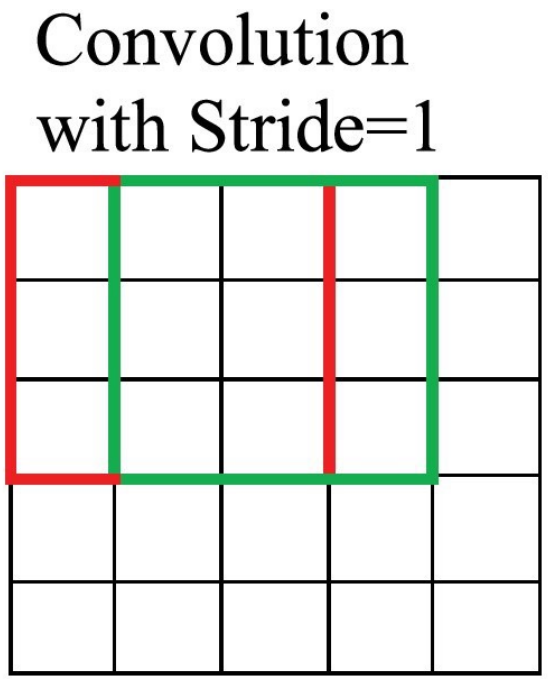
Stride

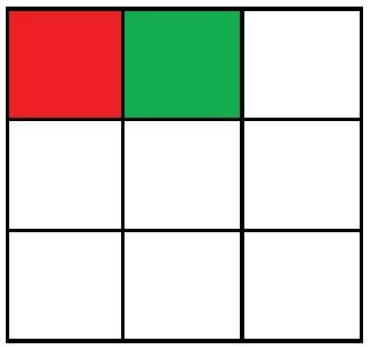
To reduce size of the next layer, can use “stride” greater than 1.

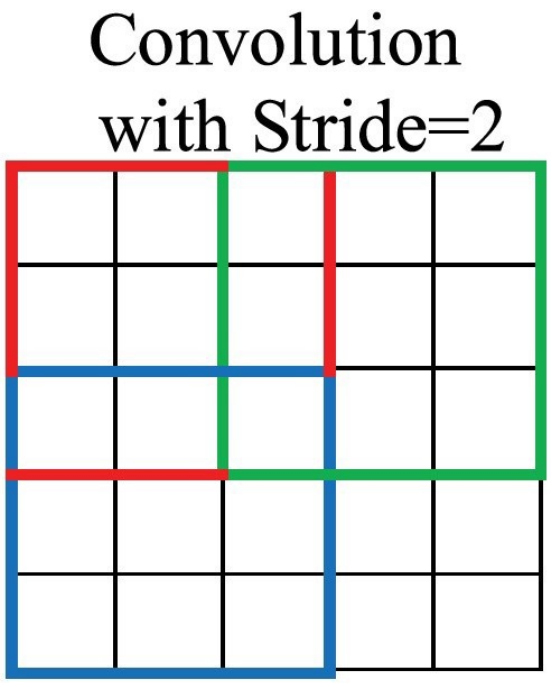
Stride = 1 means we shift the filter around in increments of 1 pixel

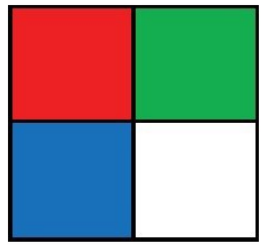
Stride = 2 means we shift by 2 pixels each time instead

Etc.





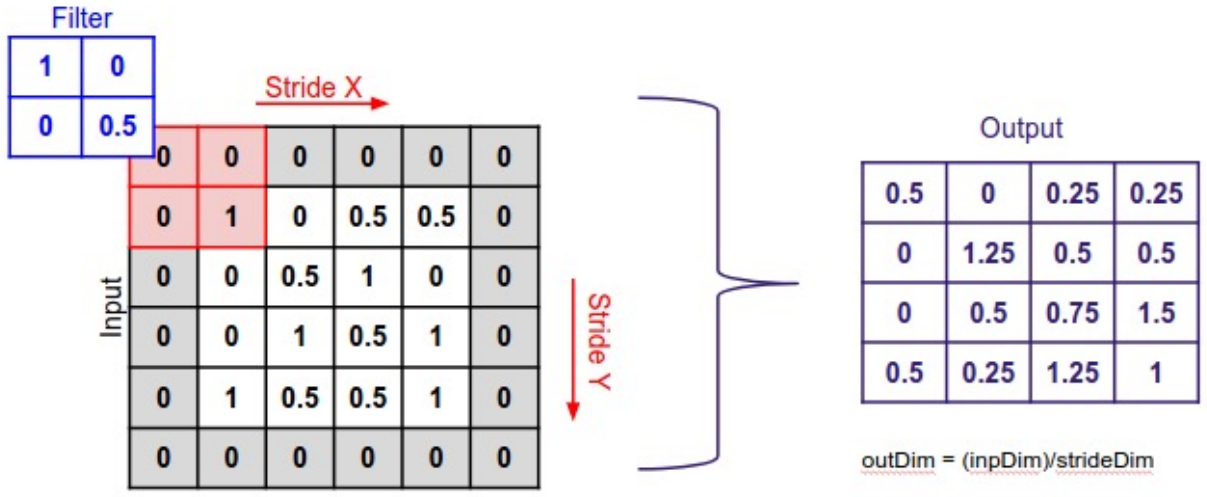




Padding

The edges are tricky.

(optionally) Sometimes we might add “padding”: a constant value all around

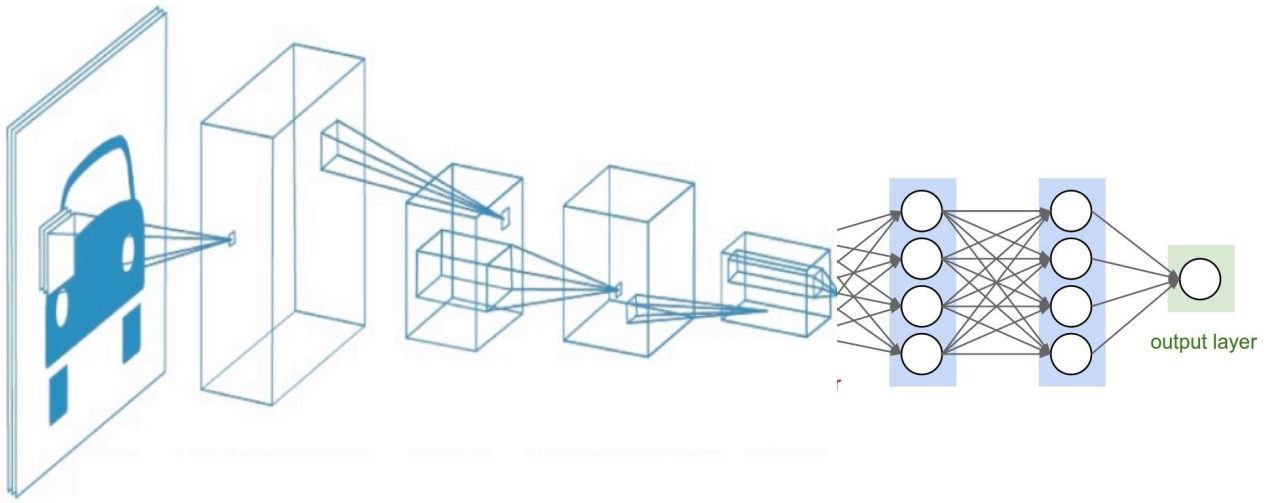


Training

Training a CNN

We want to **learn** the filters

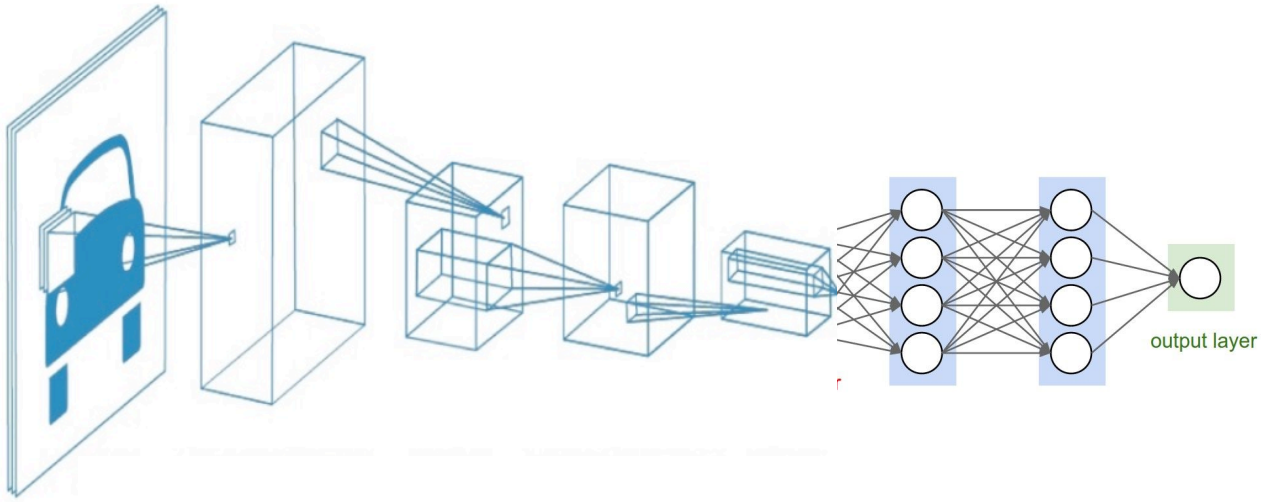
How can we learn them? Suggestions?



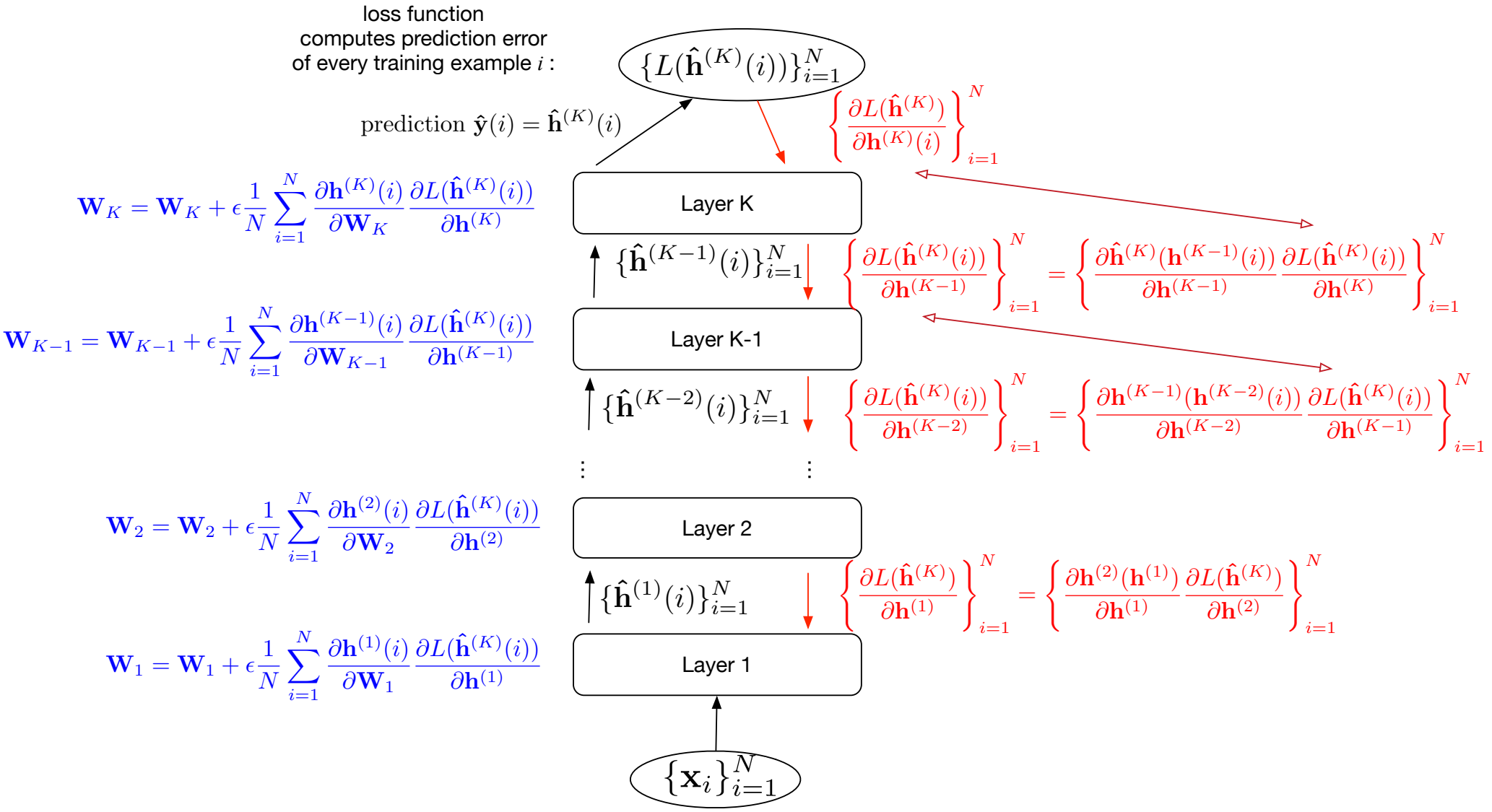
Training a CNN

We want to **learn** the filters

How can we learn them? Use **backpropagation!**



Backpropagation



Backpropagation

Q: What part needs to be modified for CNN?

loss function computes prediction error of every training example i :

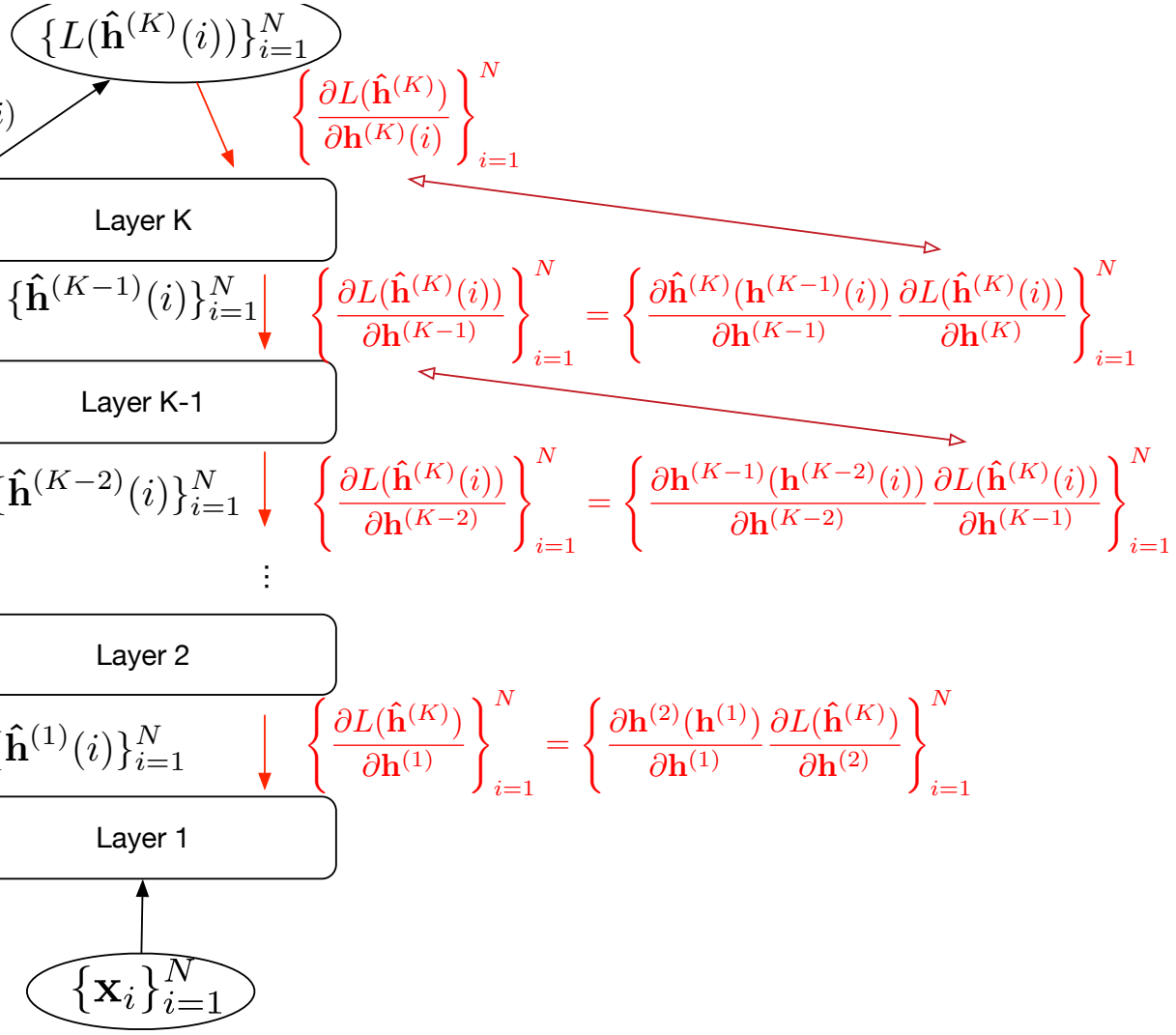
prediction $\hat{\mathbf{y}}(i) = \hat{\mathbf{h}}^{(K)}(i)$

$$\mathbf{W}_K = \mathbf{W}_K + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K)}(i)}{\partial \mathbf{W}_K} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}$$

$$\mathbf{W}_{K-1} = \mathbf{W}_{K-1} + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K-1)}(i)}{\partial \mathbf{W}_{K-1}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(2)}(i)}{\partial \mathbf{W}_2} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(2)}}$$

$$\mathbf{W}_1 = \mathbf{W}_1 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(1)}(i)}{\partial \mathbf{W}_1} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(1)}}$$



Backpropagation

Q: What part needs to be modified for CNN?

loss function computes prediction error of every training example i :

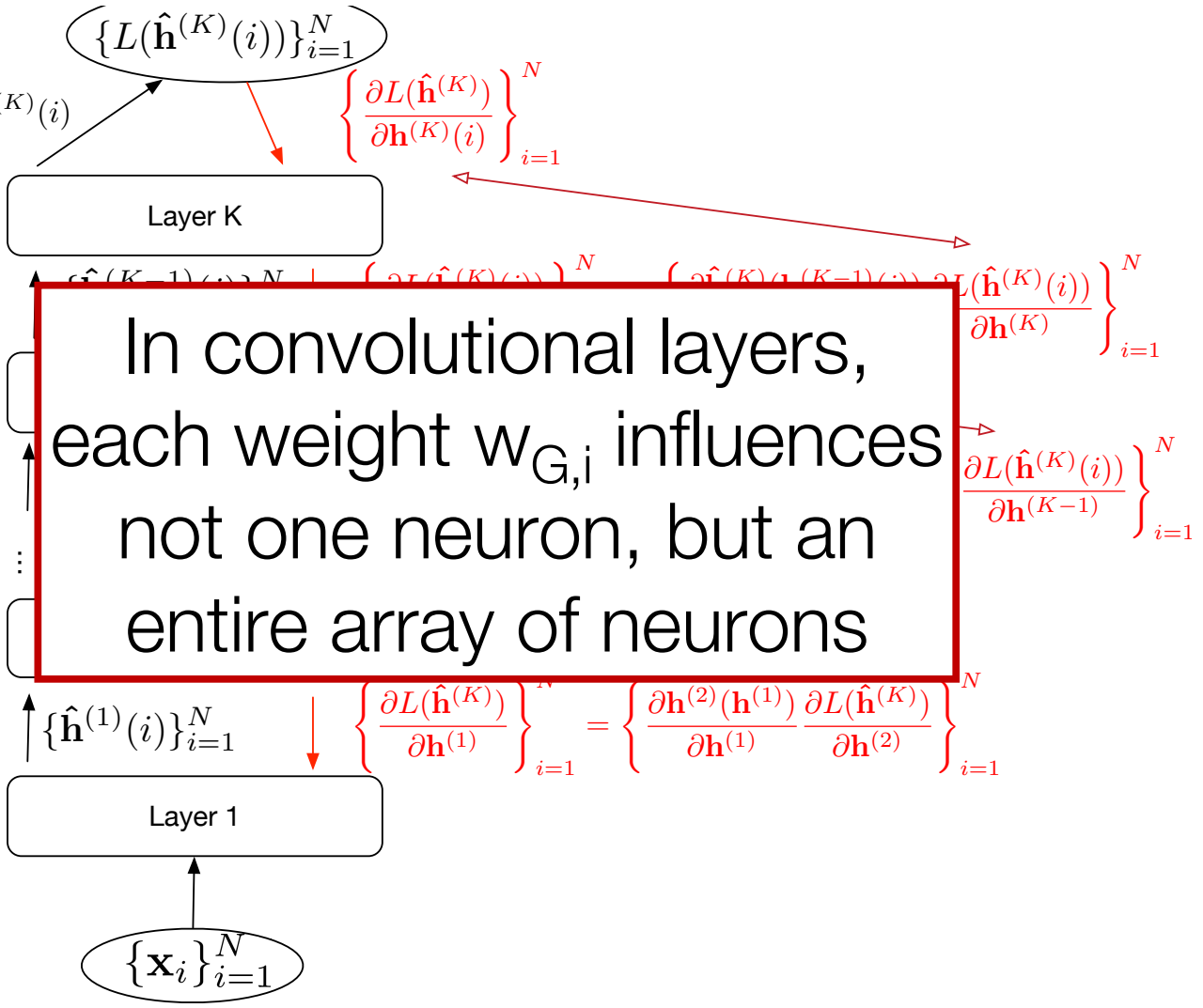
prediction $\hat{\mathbf{y}}(i) = \hat{\mathbf{h}}^{(K)}(i)$

$$\mathbf{W}_K = \mathbf{W}_K + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K)}(i)}{\partial \mathbf{W}_K} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}$$

$$\mathbf{W}_{K-1} = \mathbf{W}_{K-1} + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K-1)}(i)}{\partial \mathbf{W}_{K-1}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(2)}(i)}{\partial \mathbf{W}_2} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(2)}}$$

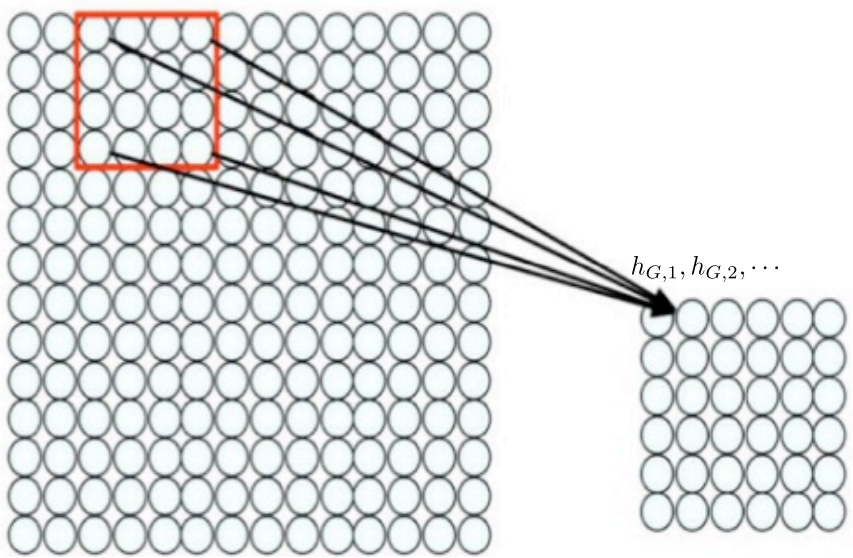
$$\mathbf{W}_1 = \mathbf{W}_1 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(1)}(i)}{\partial \mathbf{W}_1} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(1)}}$$



In convolutional layers, each weight $w_{G,i}$ influences not one neuron, but an entire array of neurons

Backpropagation for CNN

A weight $w_{G,i}$ influences a whole array of neurons at the next level



So we compute (inductively, via backpropagation) a derivative of the final loss L wrt each of these: $\frac{\partial L}{\partial h_{G,j}}$

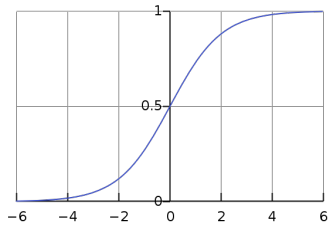
Then the derivative wrt weight $w_{G,i}$ from filter G is:

$$\frac{\partial L(x,y)}{\partial w_{G,i}} = \sum_j \frac{\partial h_{G,j}(x)}{\partial w_{G,i}} \frac{\partial L(x,y)}{\partial h_{G,j}(x)}$$

Activations and other tricks

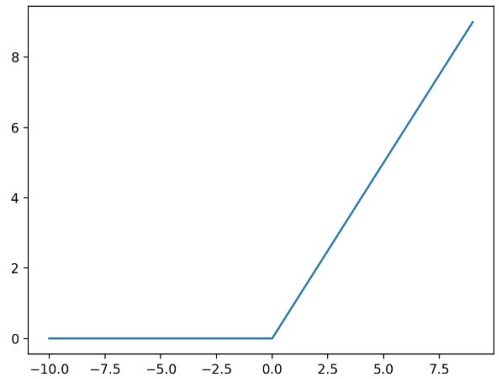
Different activation functions are useful

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



Rectified Linear Unit (ReLU):

$$\sigma(x) = \max\{0, x\}$$



Typically the convolution layers use ReLU activation function or “leaky ReLU” (see previous lecture)

(has advantages when the network is very deep, signal doesn’t diminish as it propagates backward)

Other Tricks

Neural networks notoriously require a ton of tricks/hacks to get them to work

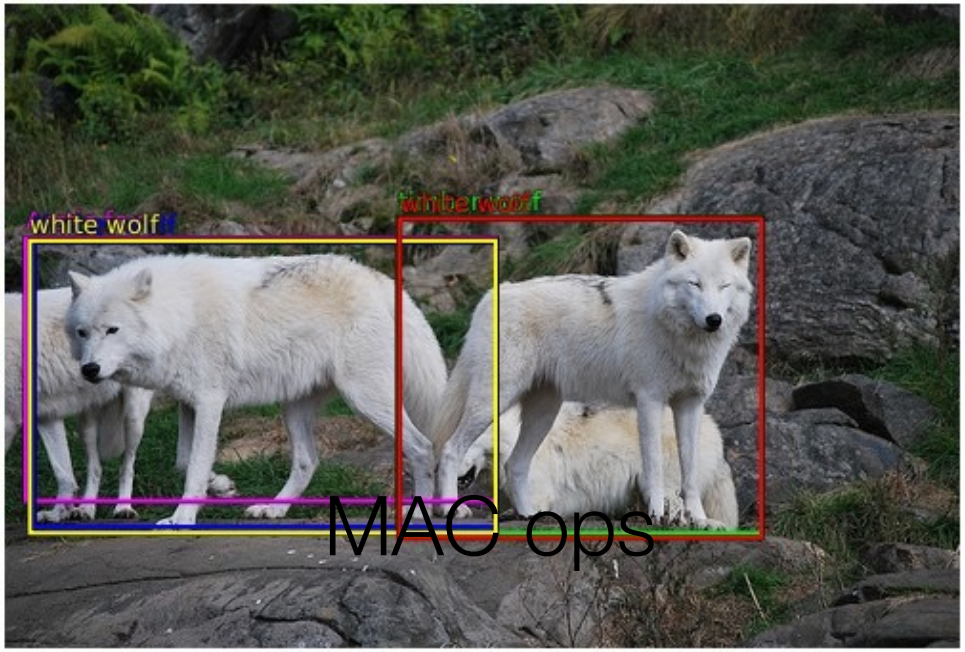
Some common tricks include:

- Pooling
- Dropout
- Weight initialization strategies

We won't get into all these details, but there are many online sources if you want to learn more about these tricks

Example Applications

Object Recognition



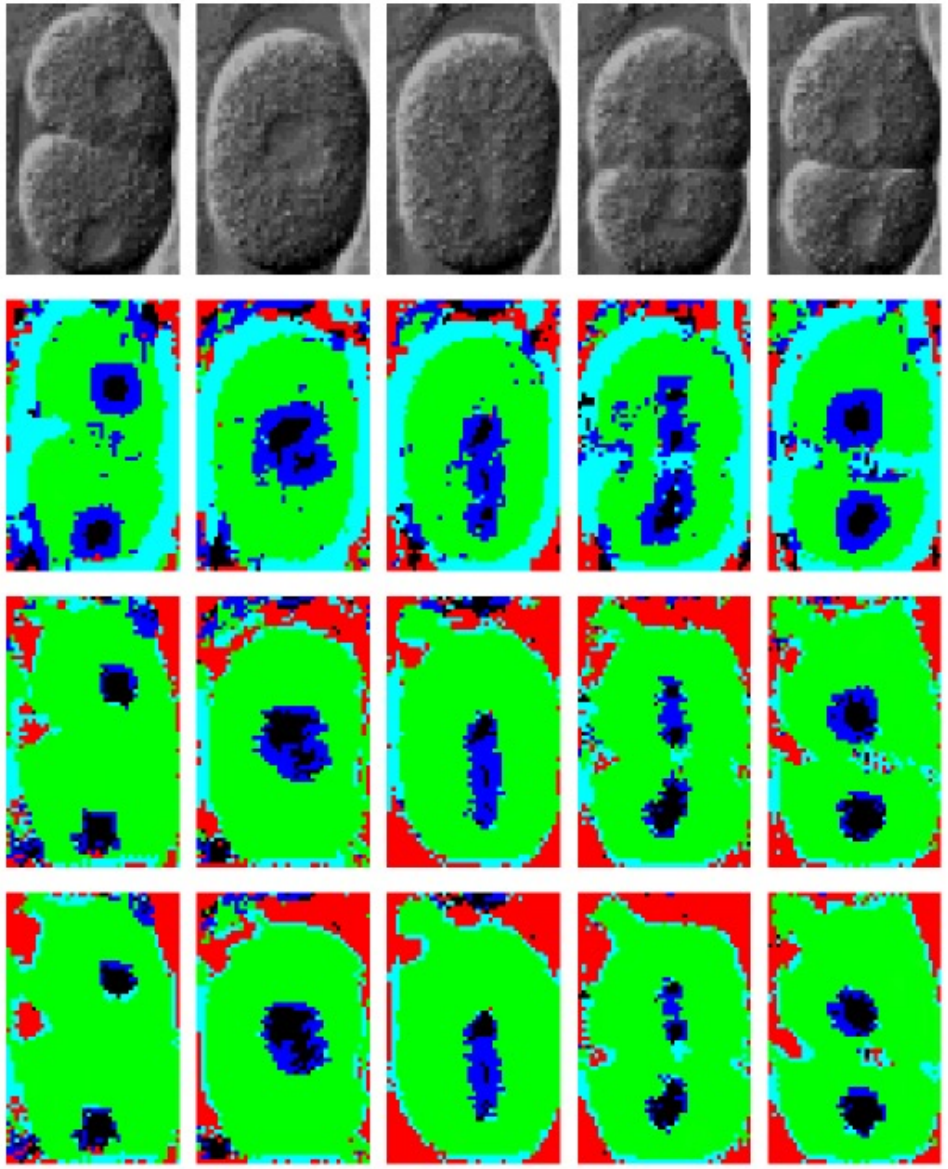
Top 5:
white wolf
white wolf
timber wolf
timber wolf
Arctic fox

ILSVRC2012_val_00000027.JPEG

Credit: Y. LeCun, Facebook

ConvNets for image segmentation

- Biological Image Segmentation
  - [Ning et al. IEEE-TIP 2005]
- Pixel labeling with large context using a convnet
- ConvNet takes a window of pixels and produces a label for the central pixel

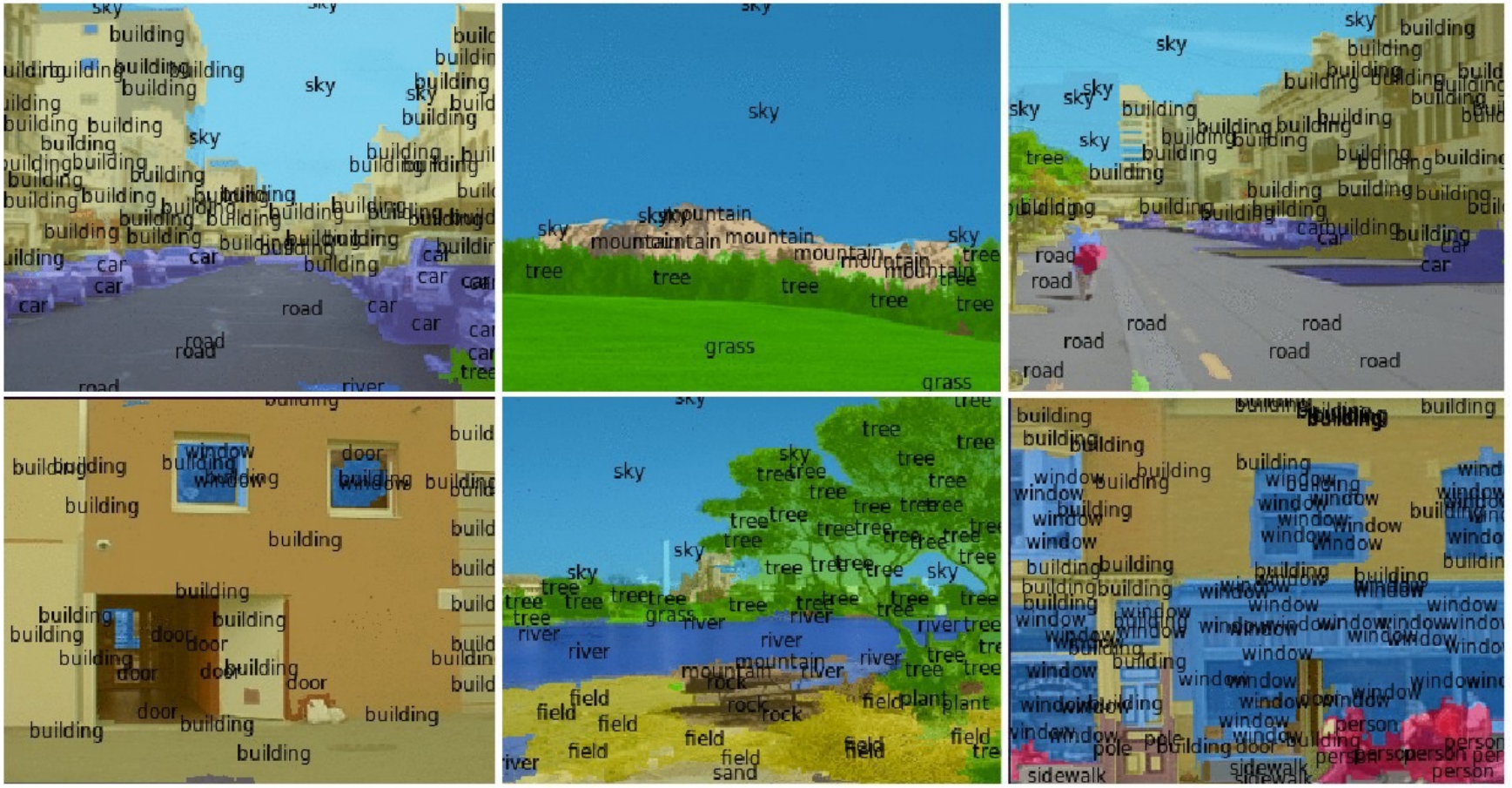


Credit: Y. LeCun, Facebook

Semantic labeling / scene parsing:

Labeling every pixel with the object it belongs to

– Would help identify obstacles, targets, landing sites, dangerous areas
Would help line up depth map with edge maps



[Farabet et al. ICML 2012, PAMI 2013]

Credit: Y. LeCun, Facebook