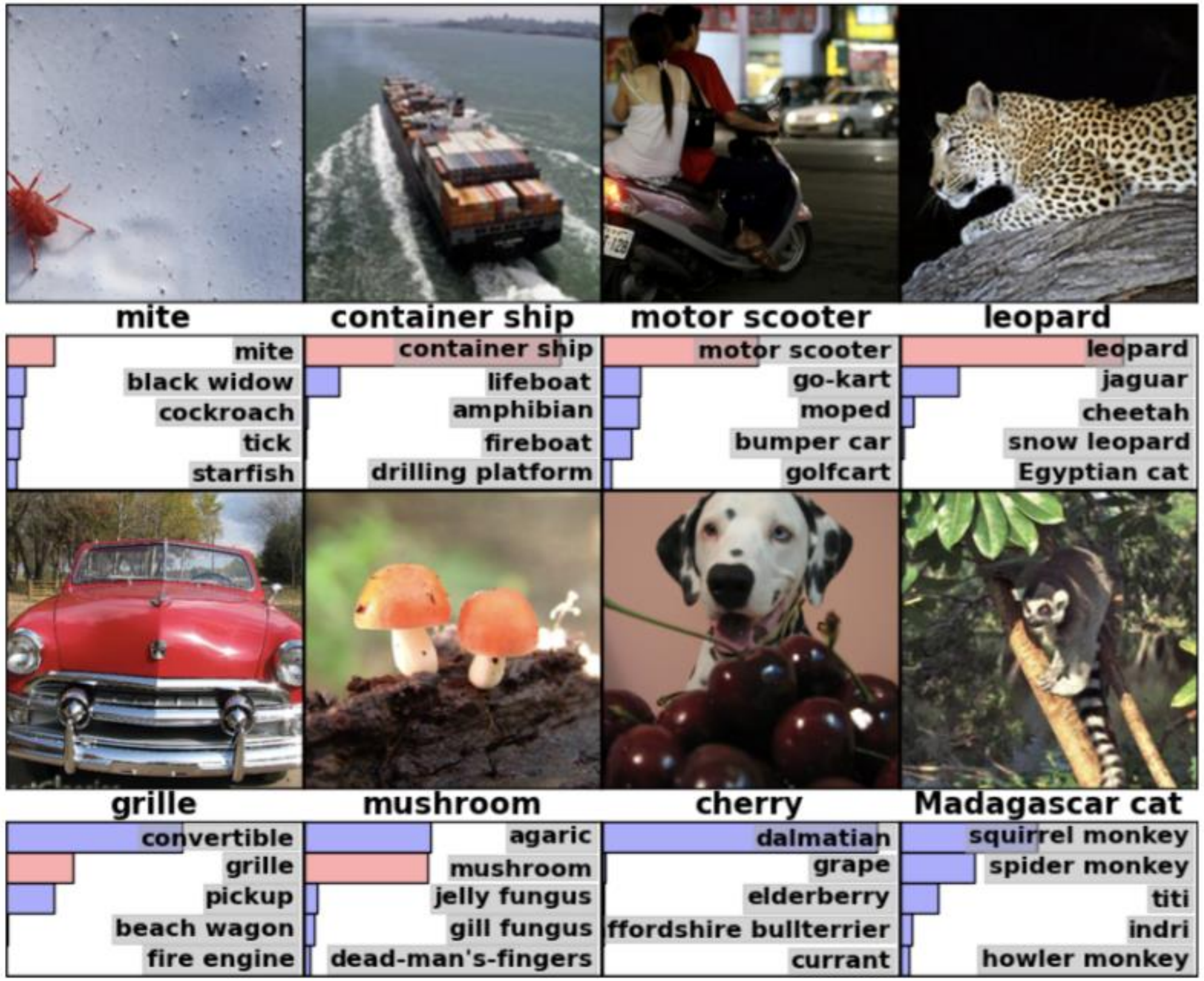
Deep Learning for Images: Convolutional Neural Networks

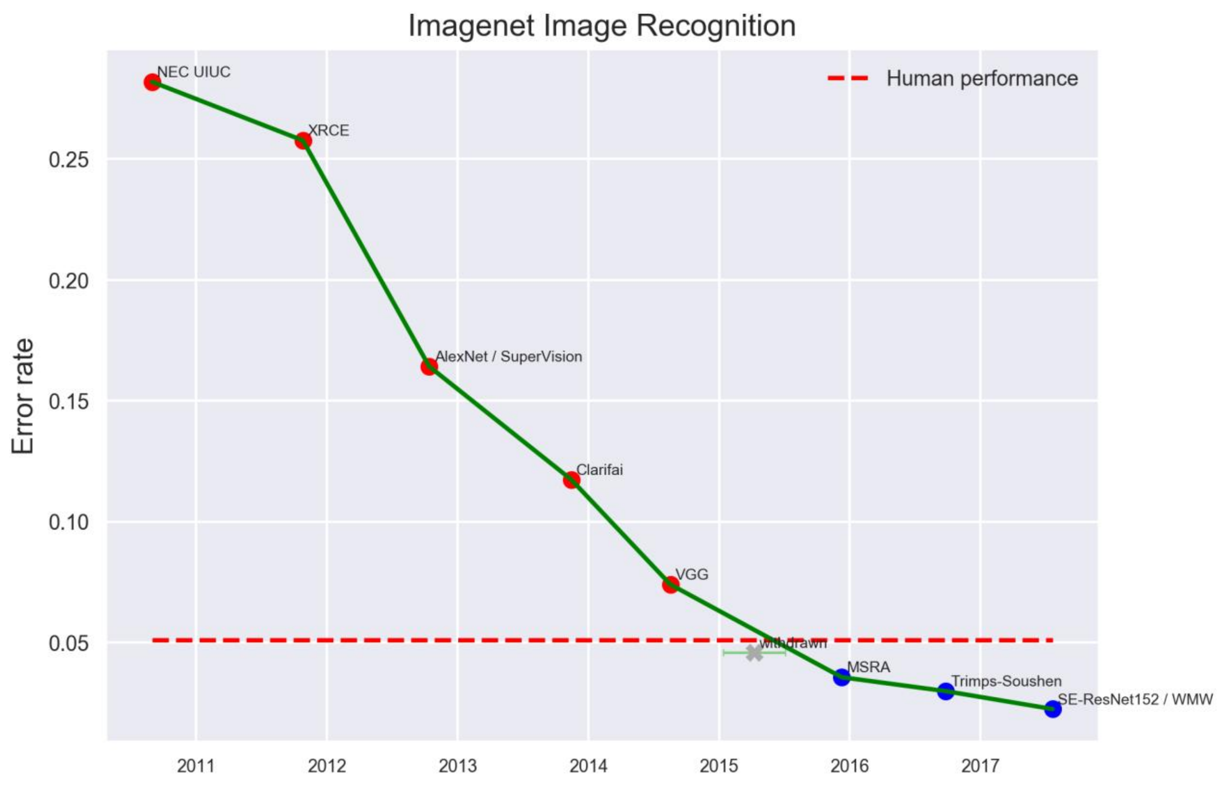
Robin Jia
USC CSCI 467, Fall 2023
September 28, 2023

Image Classification

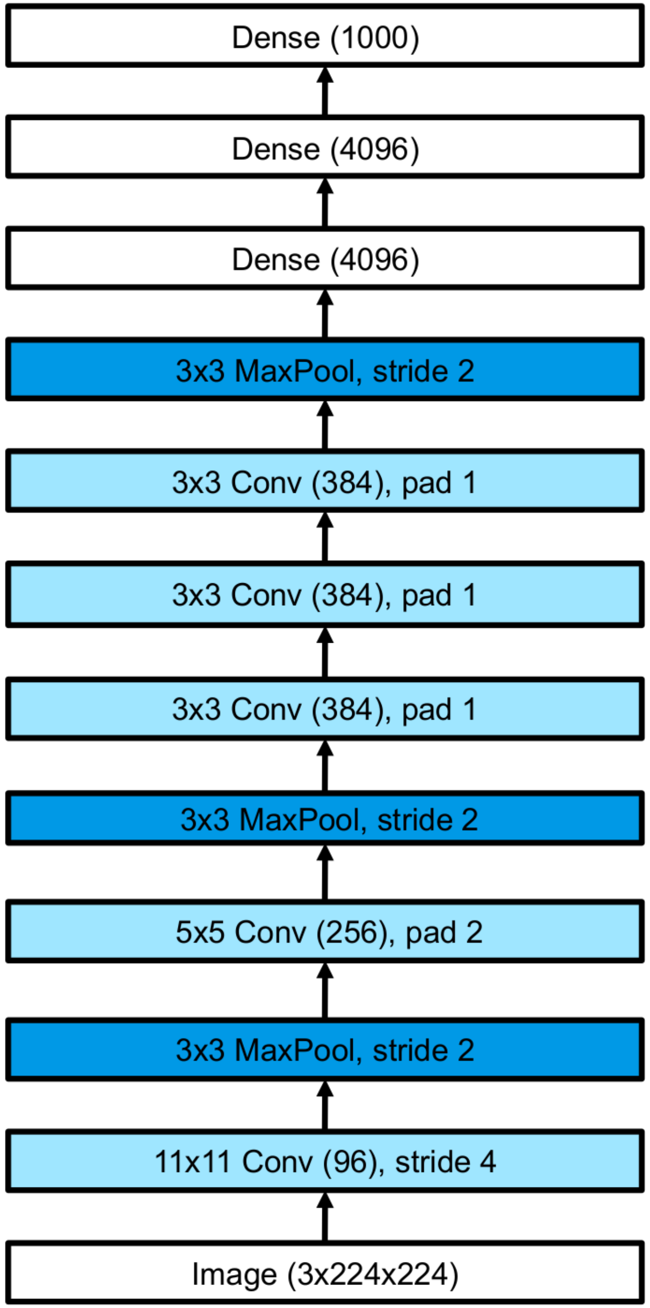


- ImageNet dataset: 14 million images, 1000 labels
- **CNNs do very well at these tasks!**

Progress on ImageNet



- 2012: AlexNet wins ImageNet challenge, marks start of deep learning era **(and is a convolutional neural network)**
- 2016: Machine learning surpasses human accuracy

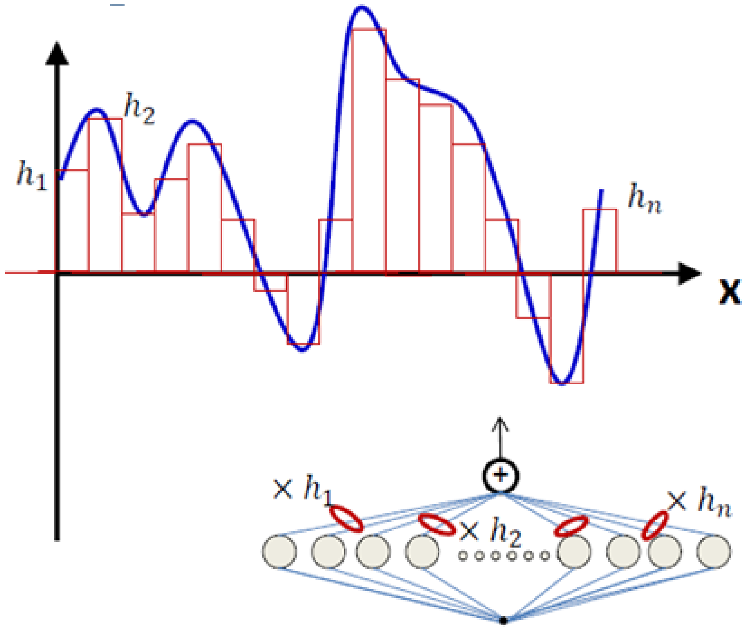


Source: https://www.eff.org/files/AI-progress-metrics.html

Outline

- Regularizing Neural Networks
- Intuition for hierarchical features
- Extracting features with convolutions
- Convolutional Neural Networks

Regularization & Neural Networks



- Recall: Neural networks are universal approximators
- This means they are prone to overfitting!
- How to avoid overfitting too badly?

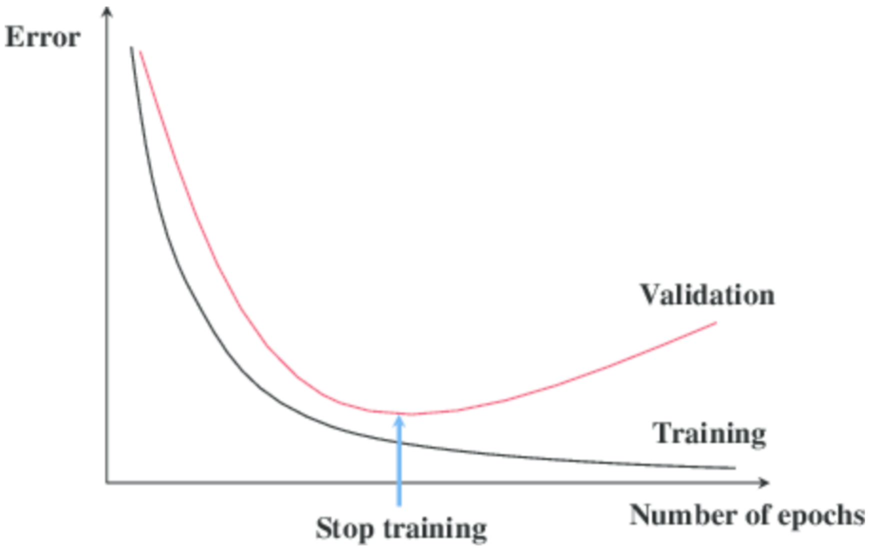
Weight decay (AKA L2 Regularization)

- L2 regularization is a valid strategy!
- Often called “weight decay” when used with neural networks
  - Because every gradient step, you add the update

$$\theta \leftarrow \theta - \eta \cdot \lambda \cdot \theta$$

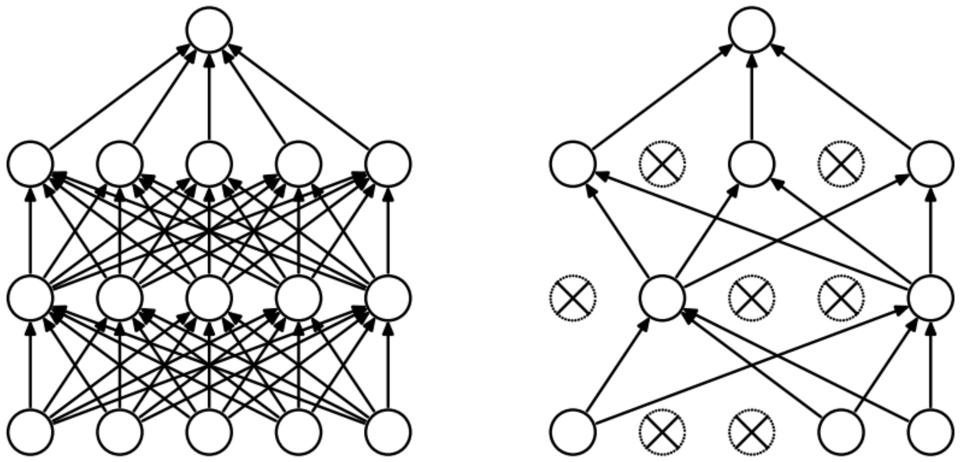
Weights literally “decay” by factor of $(1 - \eta\lambda)$

Early stopping



- Idea: Prevent overfitting by stopping training (gradient descent) before you overfit too much
- How it works
  - Every so often during gradient descent, save "checkpoint" of model parameters and evaluate development loss
  - Remember which checkpoint had best development loss
  - If development loss keeps going up, stop training
- Can be used for any model, but especially common for neural networks
  - For linear models, also common to train all the way to convergence

Dropout



(a) Standard Neural Net (b) After applying dropout.

- Idea: During training you randomly "drop out" some neurons by setting their value to 0
  - Drop each out with probability p
  - To compensate, scale the other neurons up by 1/p
  - During testing, don't do dropout
- Why?
  - Standard intuition is about "co-adaptation" of neurons
  - My personal intuition: Making the problem harder during training is good practice

Outline

- Regularizing Neural Networks
- Intuition for hierarchical features
- Extracting features with convolutions
- Convolutional Neural Networks

Review: Neural networks as feature learners

Input x

Classifier 1:
Is front clear?

Classifier 2:
Is left clear?

Classifier 3:
Is right clear?

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Classifier 4:
Where to go?

Output y
Turn left

<u>**Learn a classifier whose output is a good feature**</u>
We don't tell the model what classifier to learn
Model must learn that "is front clear" is a useful concept

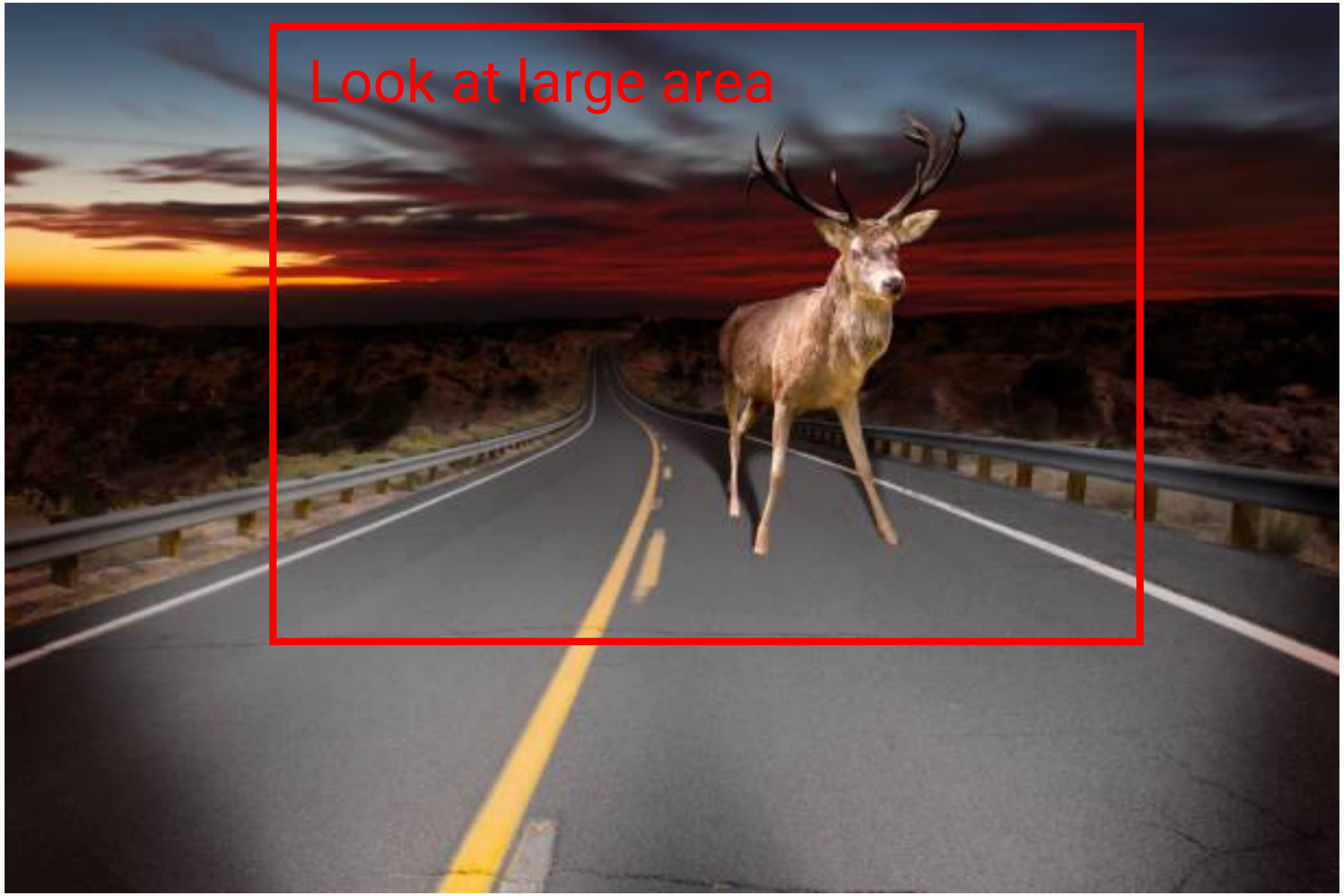
Learn to classify based on features
(same as linear model)

A hierarchy of features

- Turn left?

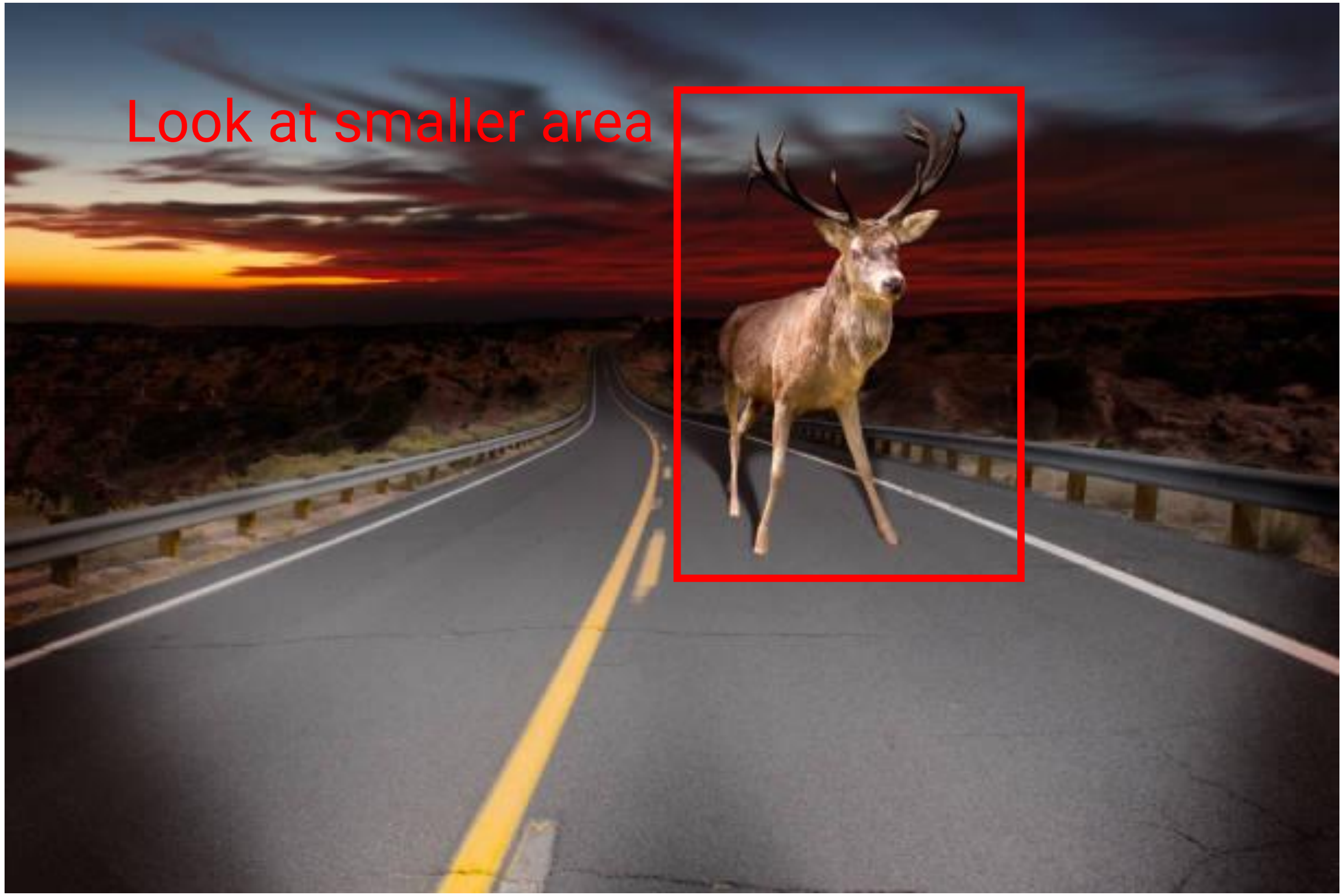
A hierarchy of features

- Turn left?
- **Front is clear?**



A hierarchy of features

- Turn left?
- Front is clear?
- **Is object a moose?**



A hierarchy of features

- Turn left?
- Front is clear?
- Is object a moose?
- **Is this a head?**

A hierarchy of features

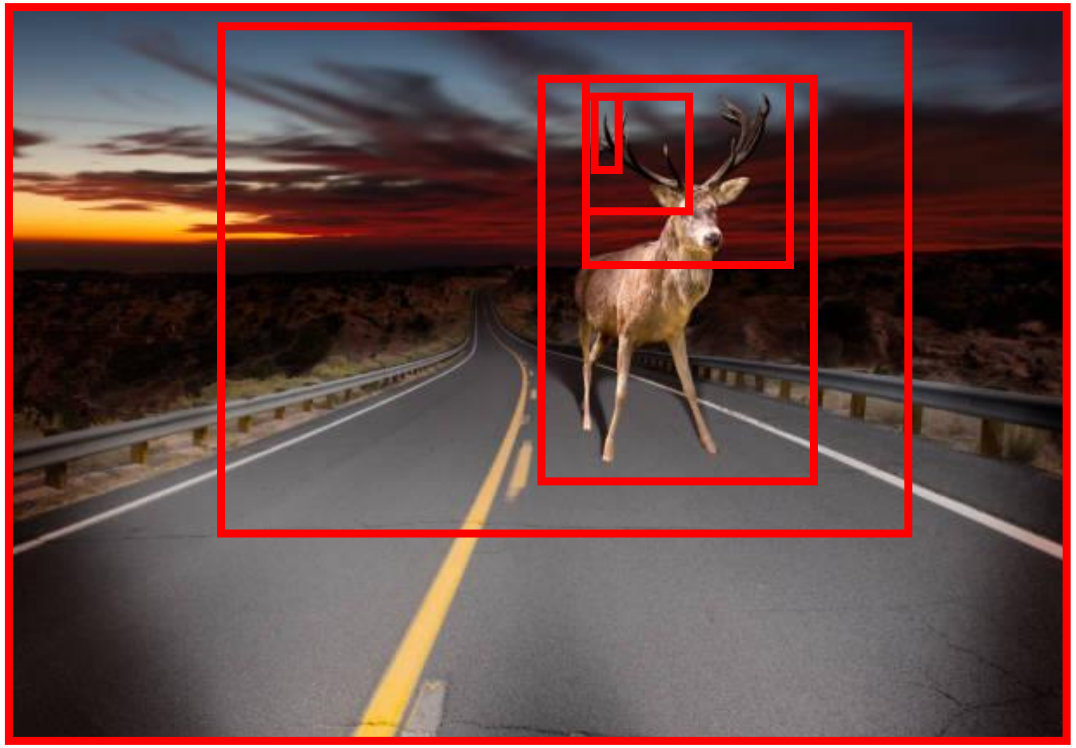
- Turn left?
- Front is clear?
- Is object a moose?
- Is this a head?
- **Is this an antler?**

A hierarchy of features

- Turn left?
- Front is clear?
- Is object a moose?
- Is this a head?
- Is this an antler?
- **Is this a line?**

Learning features hierarchically

- Today: **Process images by learning features hierarchically**
- Start with most basic features on smallest patches (e.g., a line)
- Based on those, identify more complex features (e.g., a moose)

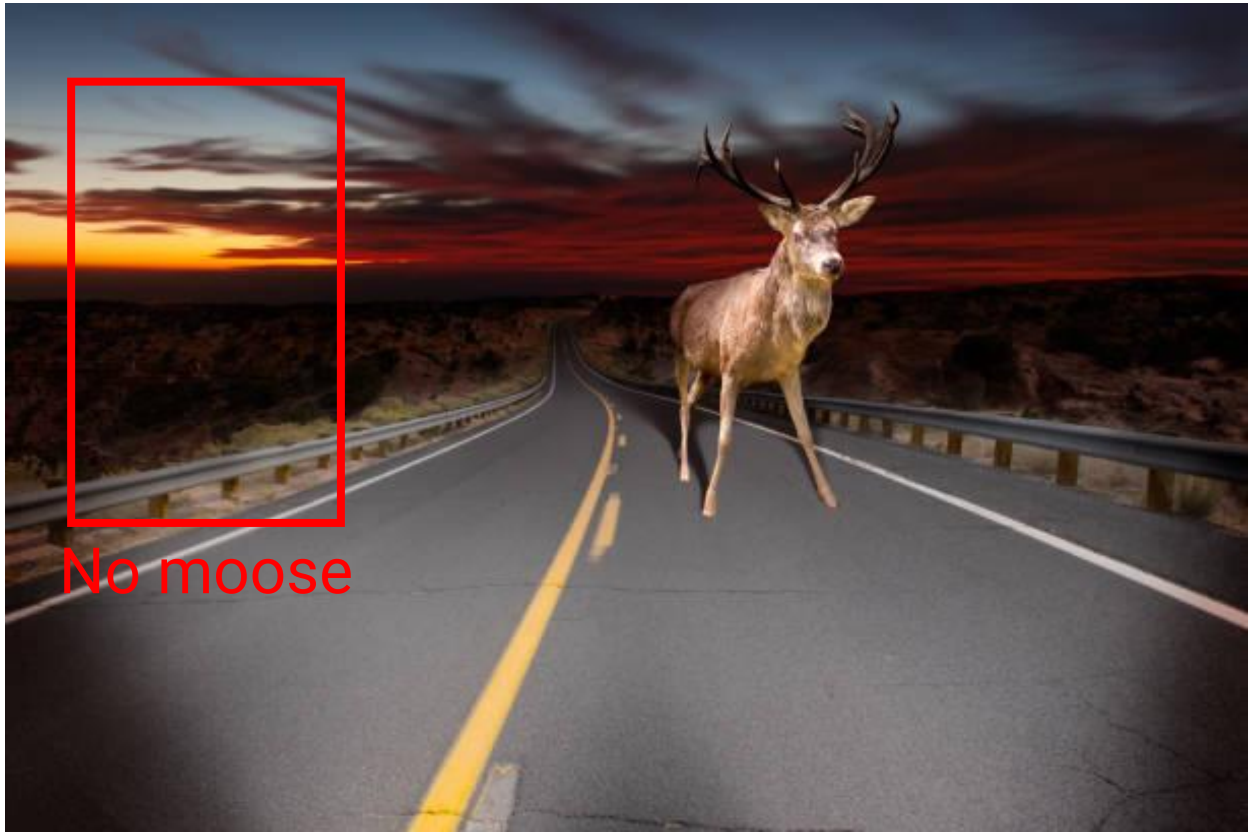


Outline

- Regularizing Neural Networks
- Intuition for hierarchical features
- Extracting features with convolutions
- Convolutional Neural Networks

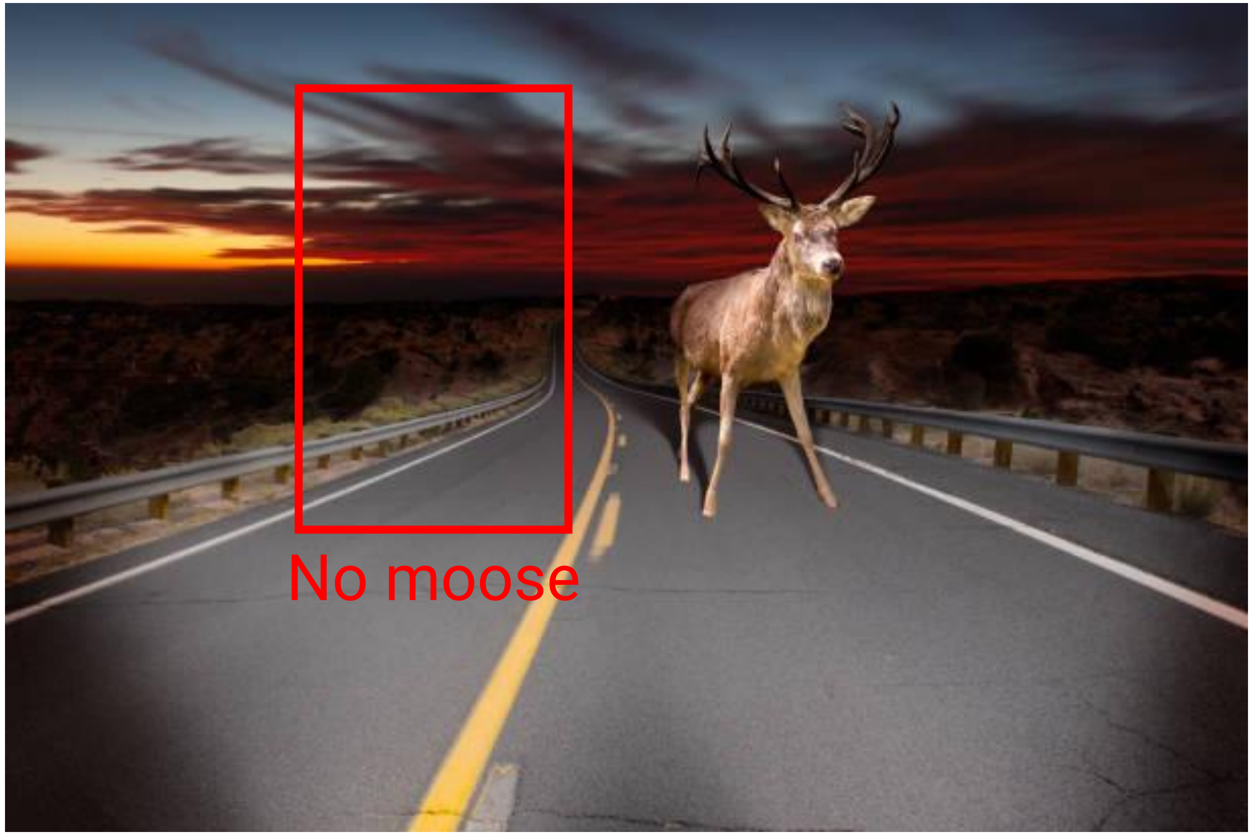
A moose detector

- Suppose you have a classifier that can tell if a region has a moose
- How to use it to create a useful feature vector?
- Slide it over each region and check if there's a moose there!



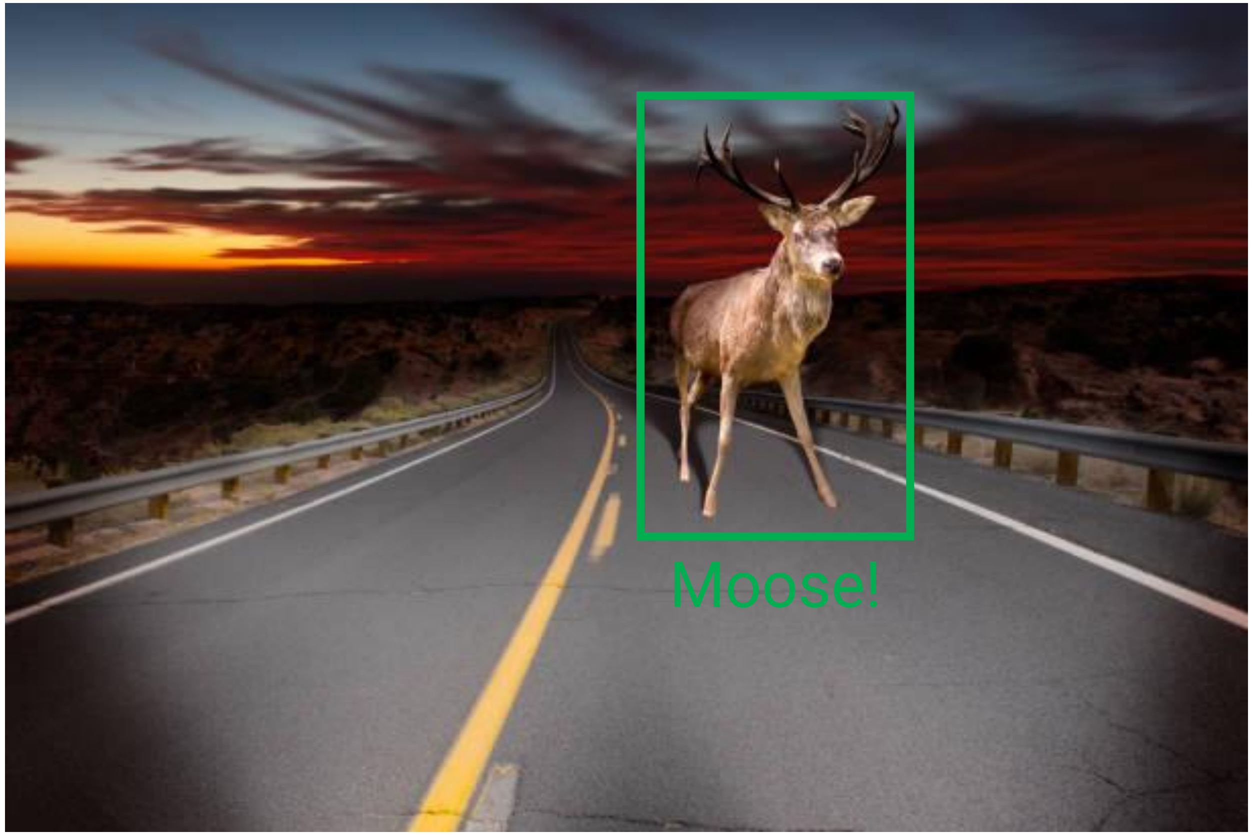
A moose detector

- Suppose you have a classifier that can tell if a region has a moose
- How to use it to create a useful feature vector?
- Slide it over each region and check if there's a moose there!



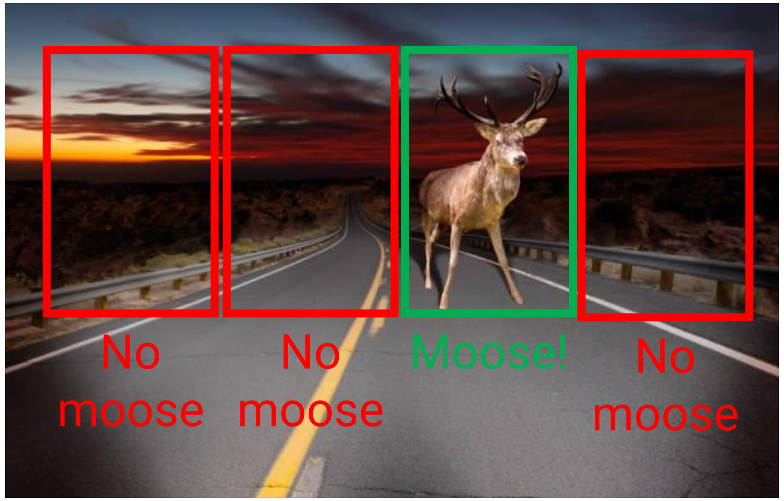
A moose detector

- Suppose you have a classifier that can tell if a region has a moose
- How to use it to create a useful feature vector?
- Slide it over each region and check if there's a moose there!



A moose detector

- Suppose you have a classifier that can tell if a region has a moose
- How to use it to create a useful feature vector?
- Slide it over each region and check if there's a moose there!
- We just did a convolution!



$$\text{Learned features} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \ldots \end{pmatrix} \begin{array}{l} \text{Moose in far left?} \\ \text{Moose in center left?} \\ \text{Moose in center right?} \\ \text{Moose in far right?} \\ \end{array}$$

An edge detector

Let’s start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve $\longrightarrow$

Dot product kernel & each image patch

Output
3x4 matrix

An edge detector

Let’s start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →

Dot product kernel & each image patch

3			

Output
3x4 matrix

An edge detector

Let's start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →
Dot product kernel & each image patch

3	-1		

Output
3x4 matrix

An edge detector

Let’s start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve $\longrightarrow$

Dot product kernel & each image patch

3	-1	0	

Output
3x4 matrix

An edge detector

<u>Let’s start a little less ambitiously…can we detect a vertical line?</u>

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →

Dot product kernel & each image patch

3	-1	0	0

Output
3x4 matrix

An edge detector

Let’s start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →
Dot product kernel & each image patch

3	-1	0	0
5			

Output
3x4 matrix

An edge detector

Let's start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →
Dot product kernel & each image patch

3	-1	0	0
5	-2		

Output
3x4 matrix

An edge detector

Let's start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →

Dot product kernel & each image patch

3	-1	0	0
5	-2	0	

Output
3x4 matrix

An edge detector

Let's start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →
Dot product kernel & each image patch

3	-1	0	0
5	-2	0	0

Output
3x4 matrix

An edge detector

Let's start a little less ambitiously…can we detect a vertical line?

-1	2	-1
-1	2	-1
-1	2	-1

(Convolutional) Kernel
3x3 matrix

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input image
5x6 matrix

Convolve →

Dot product kernel & each image patch

3	-1	0	0
5	-2	0	0
3	-1	0	0

Output
3x4 matrix

"is there vertical edge in top left?"

"is there vertical edge in bottom right?"

Each extracted feature looks for the same thing in different location

Convolutions

-1	2	-1
-1	2	-1
-1	2	-1

Kernel
(K=3)

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input
(5 x 6)

input[1:4,2:5]

3	-1	0	0
5	-2	0	0
3	-1	0	0

Output
(5-3+1 x 6-3+1)
=(3 x 4)

(1, 2)-th
element

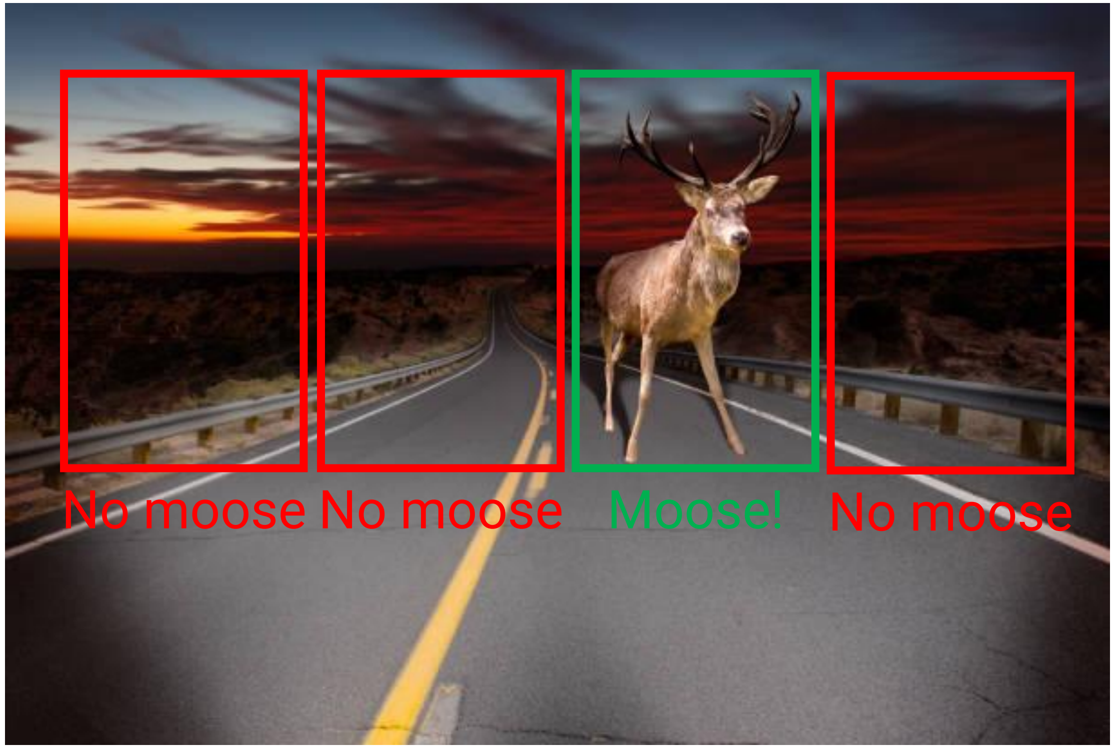
- Convolution is an operation that takes in two matrices:
  - Kernel: K x K matrix (e.g., K=3)
  - Input: W x H matrix
- Output: (W-K+1) x (H-K+1) matrix
  - ij-th element of output is dot product of kernel & input[i:i+K,j:j+K]
  - (I'm 0-indexing in these slides)
- Convolutional Layer: Kernel is our weight/parameter, use convolution to extract features
- Note: Convolution is a **linear** operation!

Motivation #1: Local Receptive Fields

- Motivation #1: Each neuron should only look at a small patch of input
- Why? Local textures/shapes are useful
- First understand local patterns, build up to global understanding

Motivation #2: Weight Sharing

- Motivation #2: In each local receptive field, the same types of features are useful
  - Basic: Detecting edges
  - More advanced: Detecting moose
- So, **share the same kernel** (i.e. weights) for all image patches
- Convolutions encode **translation equivariance**
  - If your image gets shifted, convolution outputs just get shifted too



Announcements

- HW2 due next Thursday, October 5
- Midterm exam Tuesday, October 10
  - In-class, 80 minutes, one double-sided 8.5x11 sheet of notes
  - Room assignments (also on Piazza)
    - Last name A-O: LVL 17 (this room)
    - Last name P-Z: THH 116
- Section tomorrow: Pytorch tutorial
  - Important for problem 4 of homework!

Convolutional vs. Fully Connected Layers

-1	2	-1
-1	2	-1
-1	2	-1

Kernel
(size 9)

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input
(size 30)

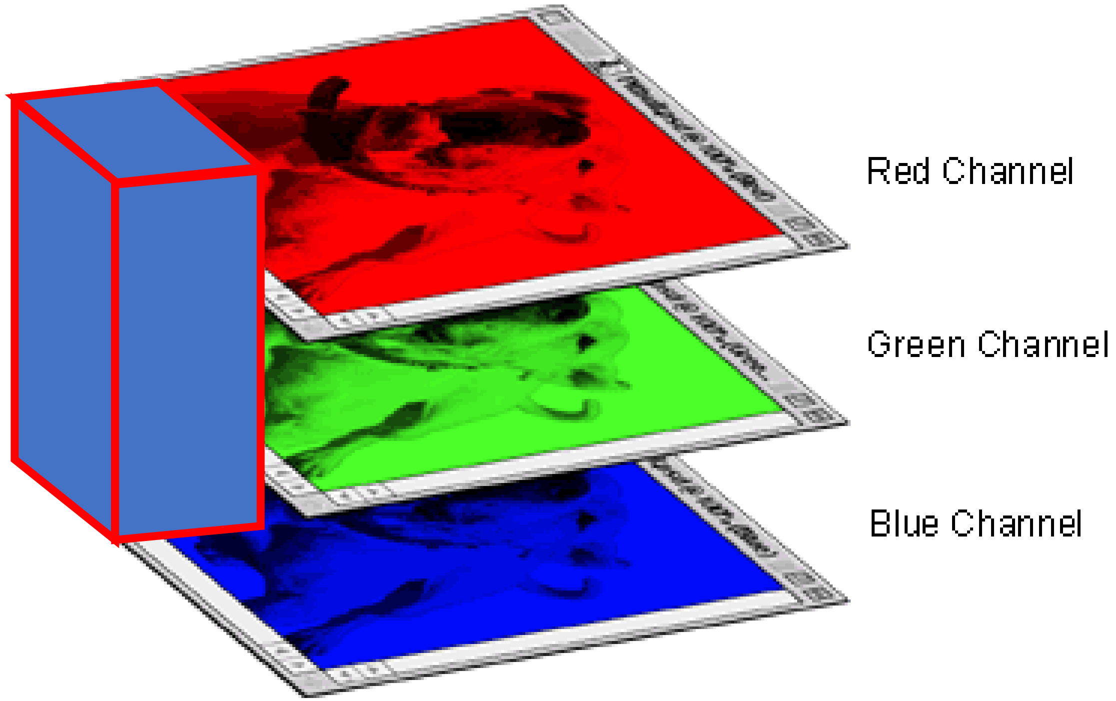
3	-1	0	0
5	-2	0	0
3	-1	0	0

Output
(size 12)

- Let's count parameters needed
  - Convolutional layer with K=3
    - Kernel = 3 x 3 = 9 parameters
    - Add a bias term = **10 parameters**
  - Fully connected layer with 30-dim input, 12-dim output needs
    - W: 30 * 12 = 360 parameters
    - b: 12 parameters
    - Total: **372 parameters**
- Fewer parameters = need less data to learn useful features
- FC would have to learn to detect the same feature (e.g., an edge) over and over again at different locations

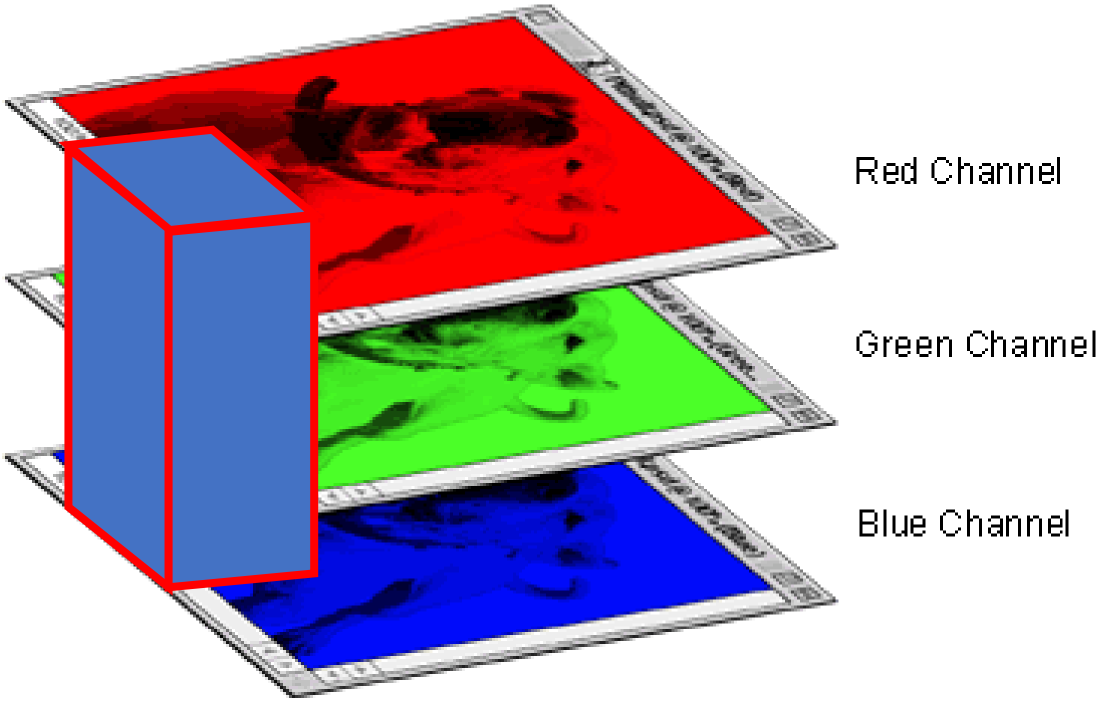
Multiple Input Channels

- Input may have multiple input channels
  - Color image has 3 “channels” for red/green/blue
  - Input is actually 3 x W x H
  - Solution: Kernel must be of size C_{in} x K x K
    - Where C_{in} is number of input channels



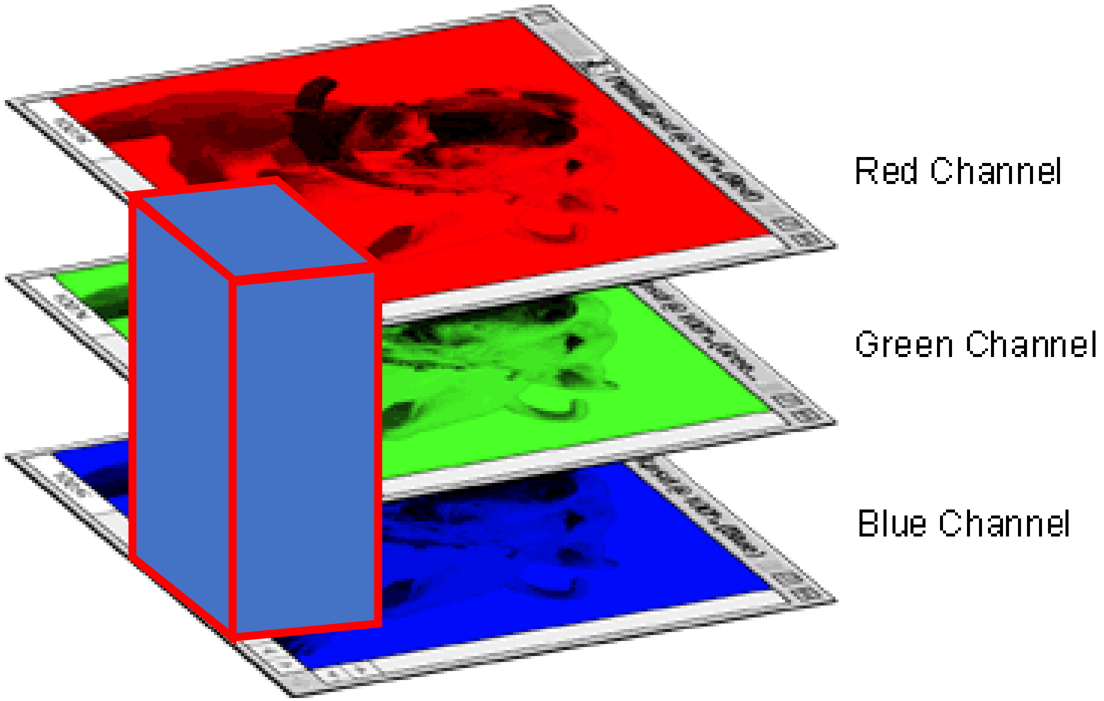
Multiple Input Channels

- Input may have multiple input channels
  - Color image has 3 “channels” for red/green/blue
  - Input is actually 3 x W x H
  - Solution: Kernel must be of size C_{in} x K x K
    - Where C_{in} is number of input channels



Multiple Input Channels

- Input may have multiple input channels
  - Color image has 3 “channels” for red/green/blue
  - Input is actually 3 x W x H
  - Solution: Kernel must be of size C_{in} x K x K
    - Where C_{in} is number of input channels



Multiple Output Channels

- What if you want more than one kernel?
  - Can have multiple kernels, each to detect a different thing
  - One for vertical lines, one for horizontal lines, etc.
  - So the total size of kernel tensor is C_{out} x C_{in} x K x K

-1	2	-1
-1	2	-1
-1	2	-1

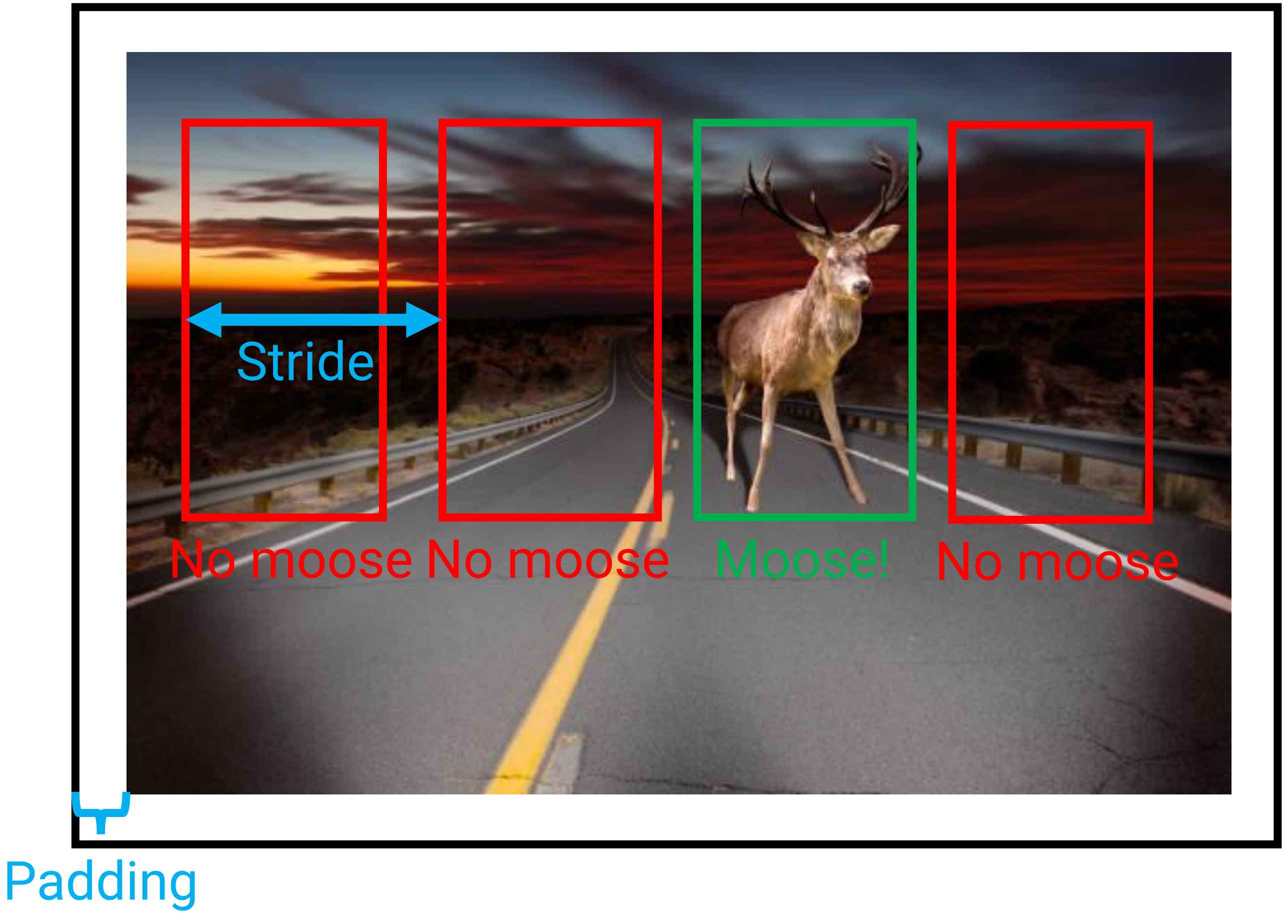
Kernel[0,0,:,:]

-1	-1	-1
2	2	2
-1	-1	-1

Kernel[1,0,:,:]

Stride and Padding

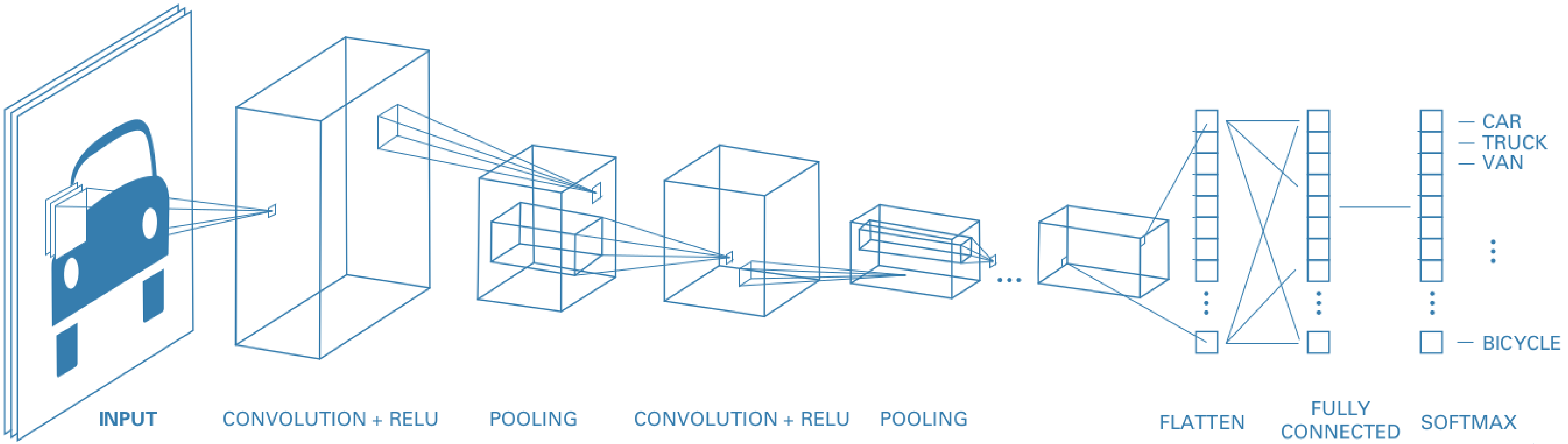
- Stride: As you slide across image, how big of a step do you take?
  - Default: stride=1 pixel
  - Can choose larger stride to reduce dimensionality
- Padding: Can pad the edges of images with 0's
  - For K=3 and no padding, width/height shrink by 2 each time
  - Adding width-1 padding on each side prevents this
  - For K=5, pad by 2, etc.
  - Default: No padding



Outline

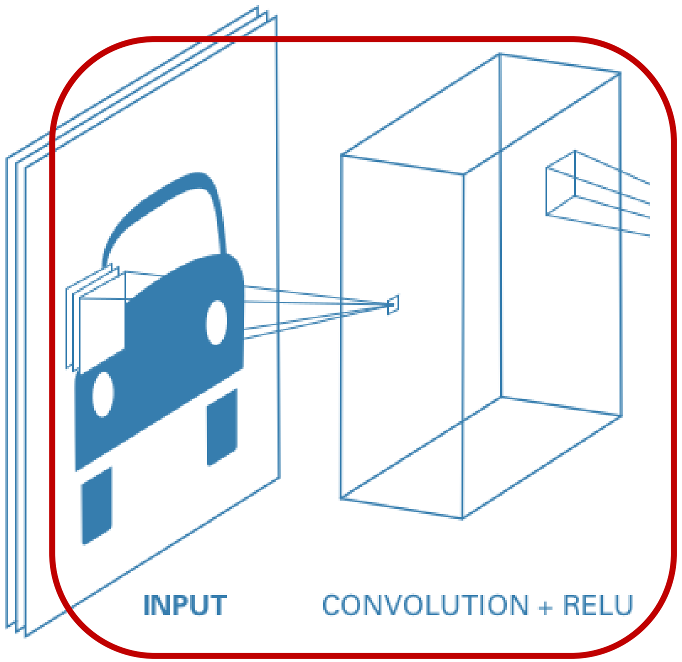
- Regularizing Neural Networks
- Intuition for hierarchical features
- Extracting features with convolutions
- Convolutional Neural Networks

Convolutional Neural Networks (CNNs)



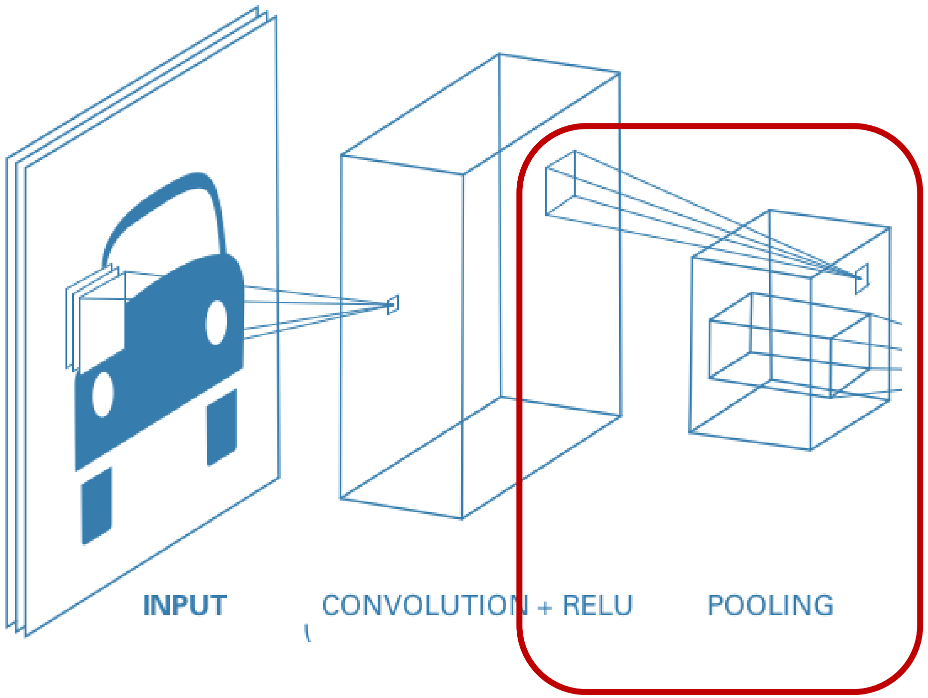
- How to incorporate convolutions into a full model?
- Basic idea: Use convolutions at beginning, then fully connected layer at end

Convolutional Layers



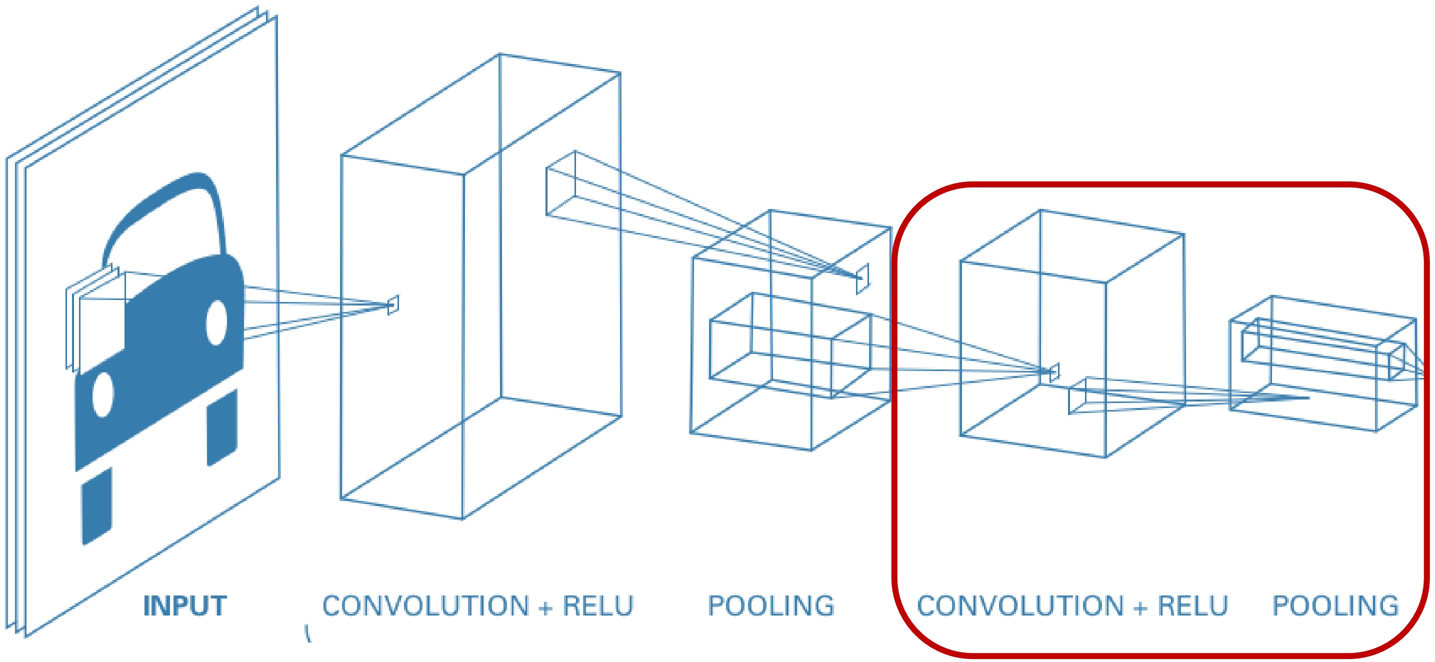
- First step: Convolutional Layer + ReLU
- Analogous to Linear layer + ReLU
  - Convolutional layer is just a special type of linear layer with local receptive fields & weight sharing!
  - So we again want to apply a non-linearity after the linear operation
- ReLU is standard for CNNs

Pooling



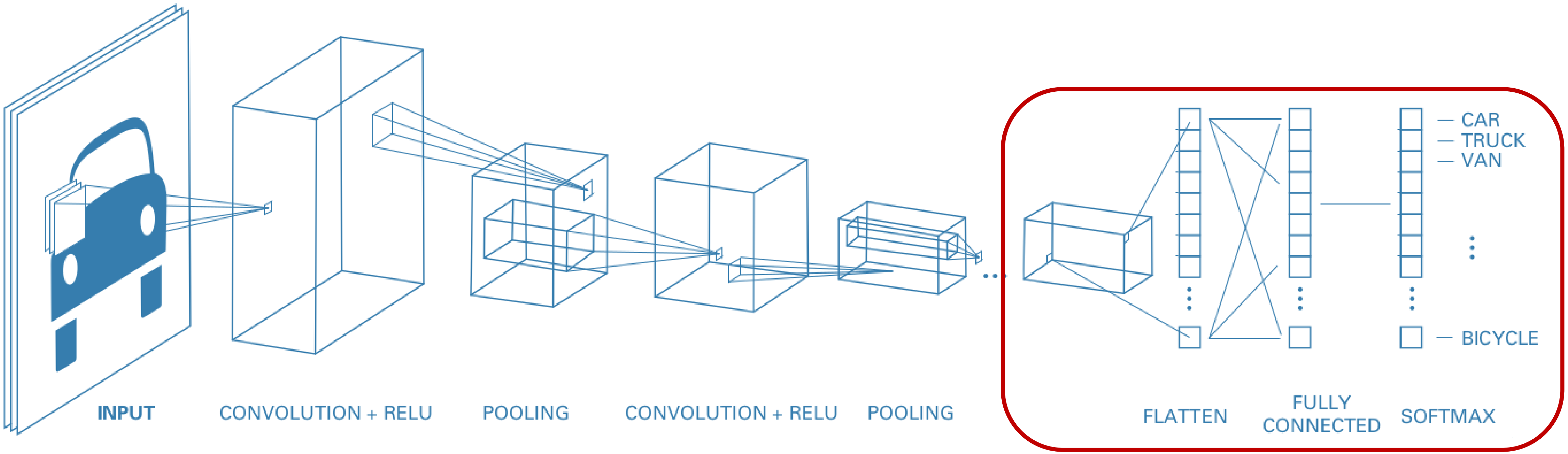
- Goal: Make receptive field bigger as we process the image
  - Early: Look for edges (small patch)
  - Later: Look for moose (larger patch)
- How do we do this? Pooling!
- Effectively we reduce resolution of input by a factor of P (often P=2)
  - Average pool: Average in each 2x2 patch
  - Max pool: Max in each 2x2 patch

More Conv + ReLU + Pool



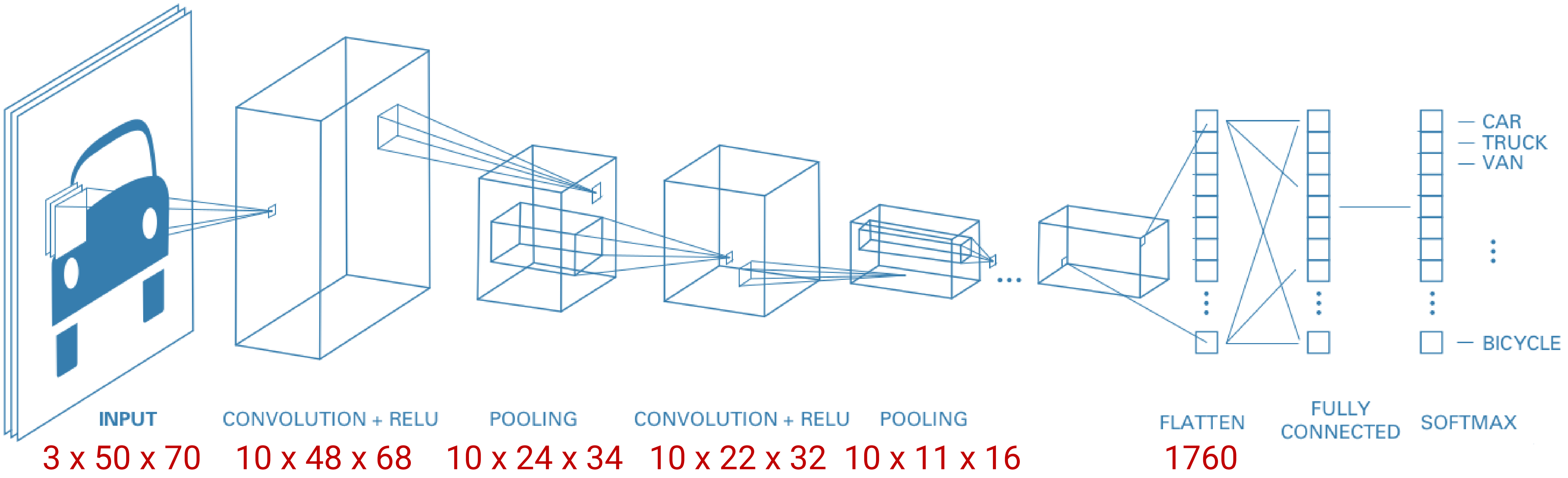
- Can stack multiple Conv + ReLU + pool blocks
- Similar to increasing number of hidden layers in MLP
- Deeper layers convolutional layers have larger effective receptive field
  - Can learn higher-level concepts

Fully connected layers



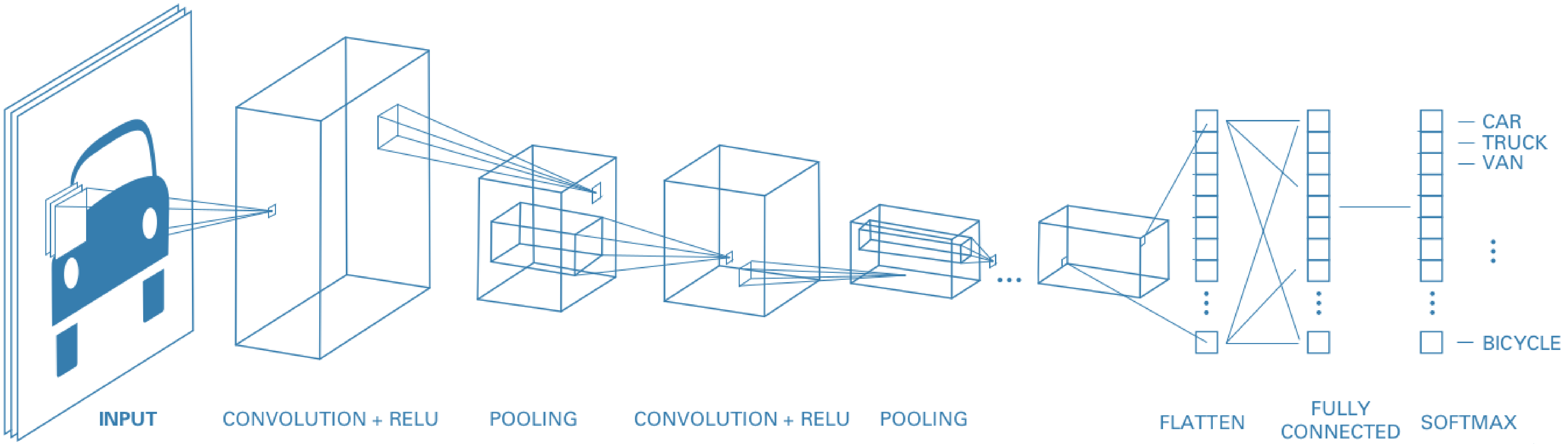
- At the very end, we want fully global processing
- Fully connected layers are good at this!
- First flatten from [channels x width x height] to a flat vector
- Then do a MLP (e.g., 2-layer neural network) on top

Keeping the dimensions straight



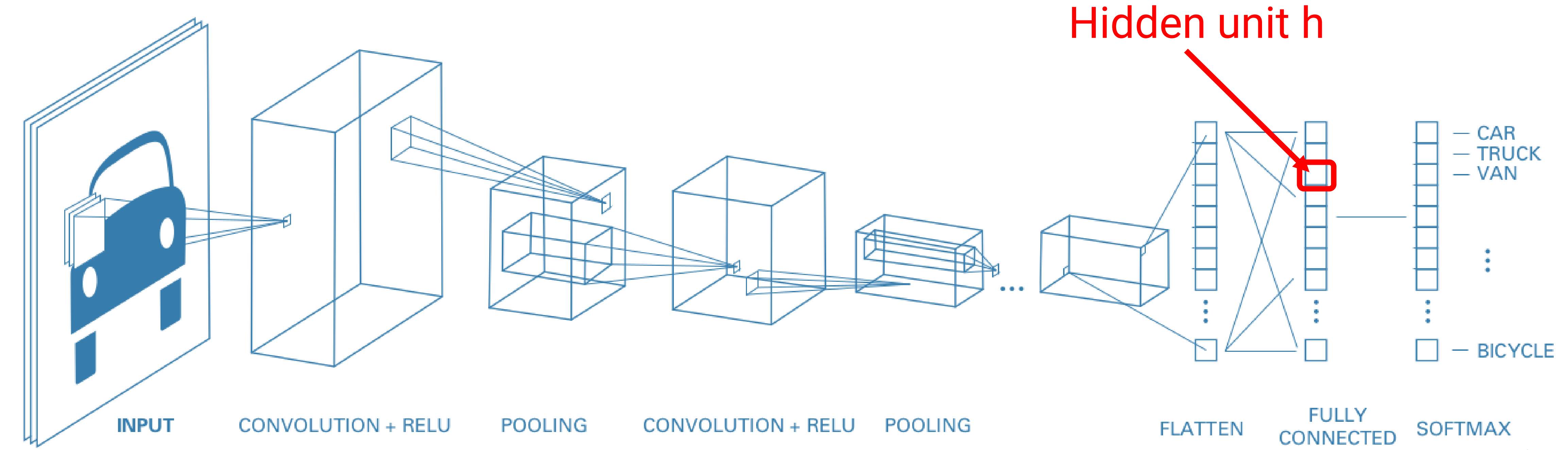
- Suppose convolution kernels are 3x3, 10 output channels, pooling is 2x2, no padding, stride=1
  - Each convolution operation loses 3-1=2 in width and height
- In code, also a “batch” dimension because we process all examples in batch together

How does backprop learn features?



- Every convolution & fully connected layer has (many) parameters
  - Convolutional: Kernel with #outChannels x (#inChannels x K x K + 1) params
  - Fully connected: #outDimensions x (#inDimensions + 1) params
- These all have to get learned by backprop + gradient descent on the loss

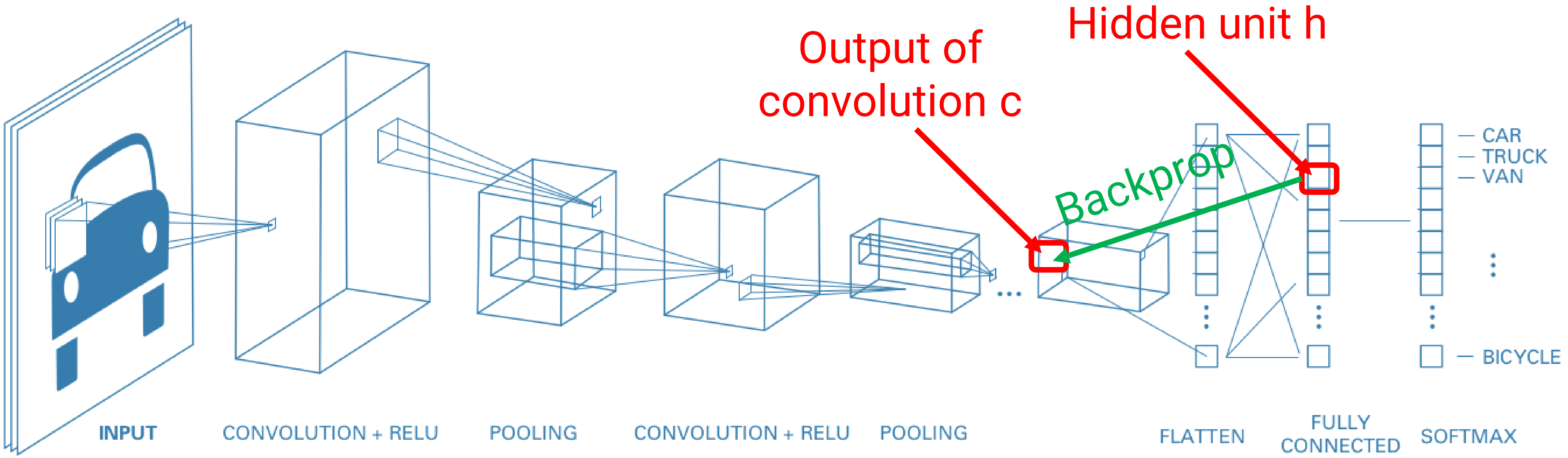
How does backprop learn features?



- Training example $(x^{(1)}, y^{(1)})$: $\partial(\text{Loss})/\partial(h) > 0$
  - Means that making h **smaller** leads to lower loss
- Training example $(x^{(2)}, y^{(2)})$: $\partial(\text{Loss})/\partial(h) < 0$
  - Means that making h **larger** leads to lower loss

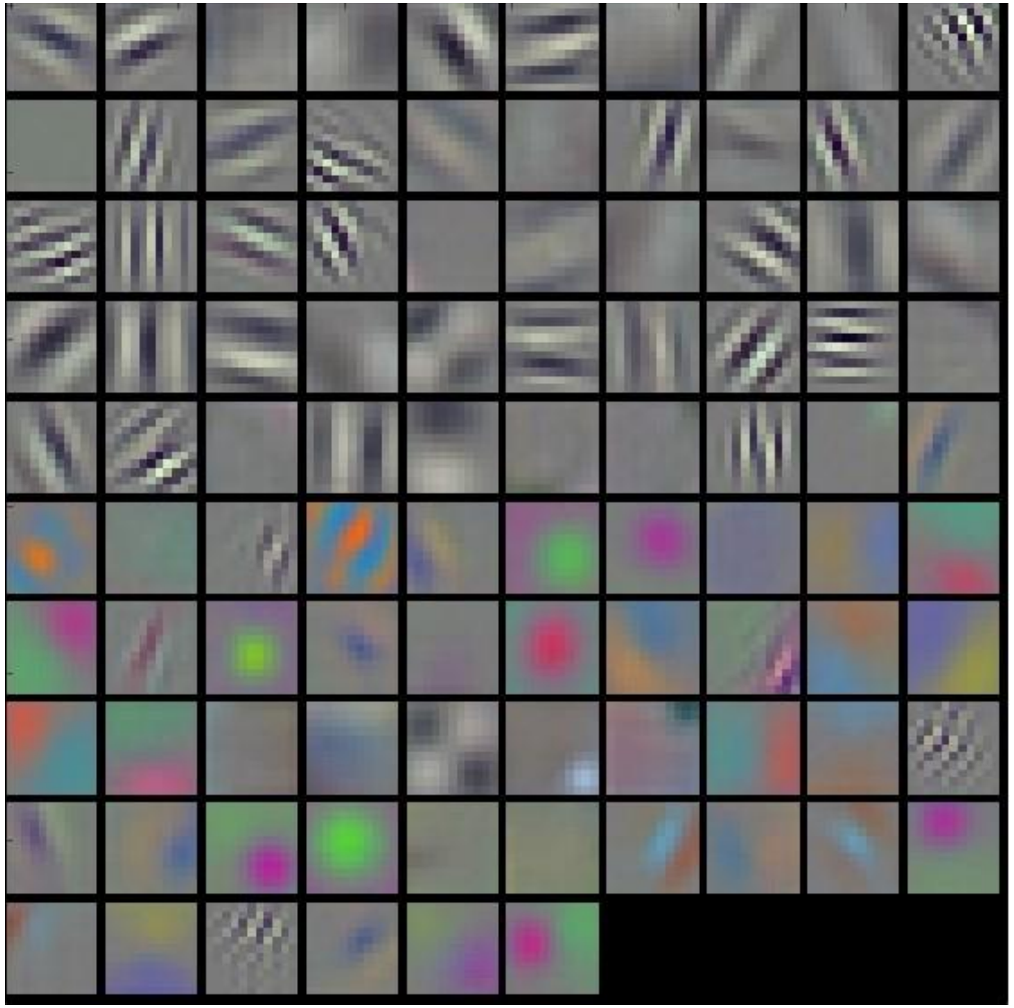
- h is output of "classifier"
- Gradient tunes classifier parameters to make output larger on some examples, smaller on others

How does backprop learn features?



- Backpropagation: Does making c bigger change h in good or bad way?
- Sum up these considerations over all hidden units that depend on c
- Train convolutional kernel parameters so that value of c leads to [values of h's that lead to good outputs]
- And so on for earlier layers…

What features do CNNs learn?



- Kernels of AlexNet first layer
  - Each one is 3 (for RGB) x 11 x 11
- What is learned?
  - Edge detectors in different directions and widths
  - Patches of various colors

What features do CNNs learn?

Faces

Dogs (eyes?)

Red ornaments/
flowers

Text (years?)

Houses

Lens flare?

Each Row: Images that activate a different neuron in 5th POOL layer of AlexNet

Conclusion

- Convolution: Restricted linear operation parameterized by a small kernel
- Convolutional layers extract useful features for images
  - Motivation #1: Local Receptive Fields
  - Motivation #2: Weight Sharing
- Standard CNN architecture
  - Start: Convolutional layer + ReLU + Max Pooling
  - End: Fully connected layer

-1	2	-1
-1	2	-1
-1	2	-1

Kernel
(K=3)

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0

Input

3	-1	0	0
5	-2	0	0
3	-1	0	0

Output