

# Deep Learning for Images: Convolutional Neural Networks

---

**Robin Jia**

USC CSCI 467, Spring 2023

February 23, 2023

# 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**

Learn a classifier whose output is a good feature

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

---

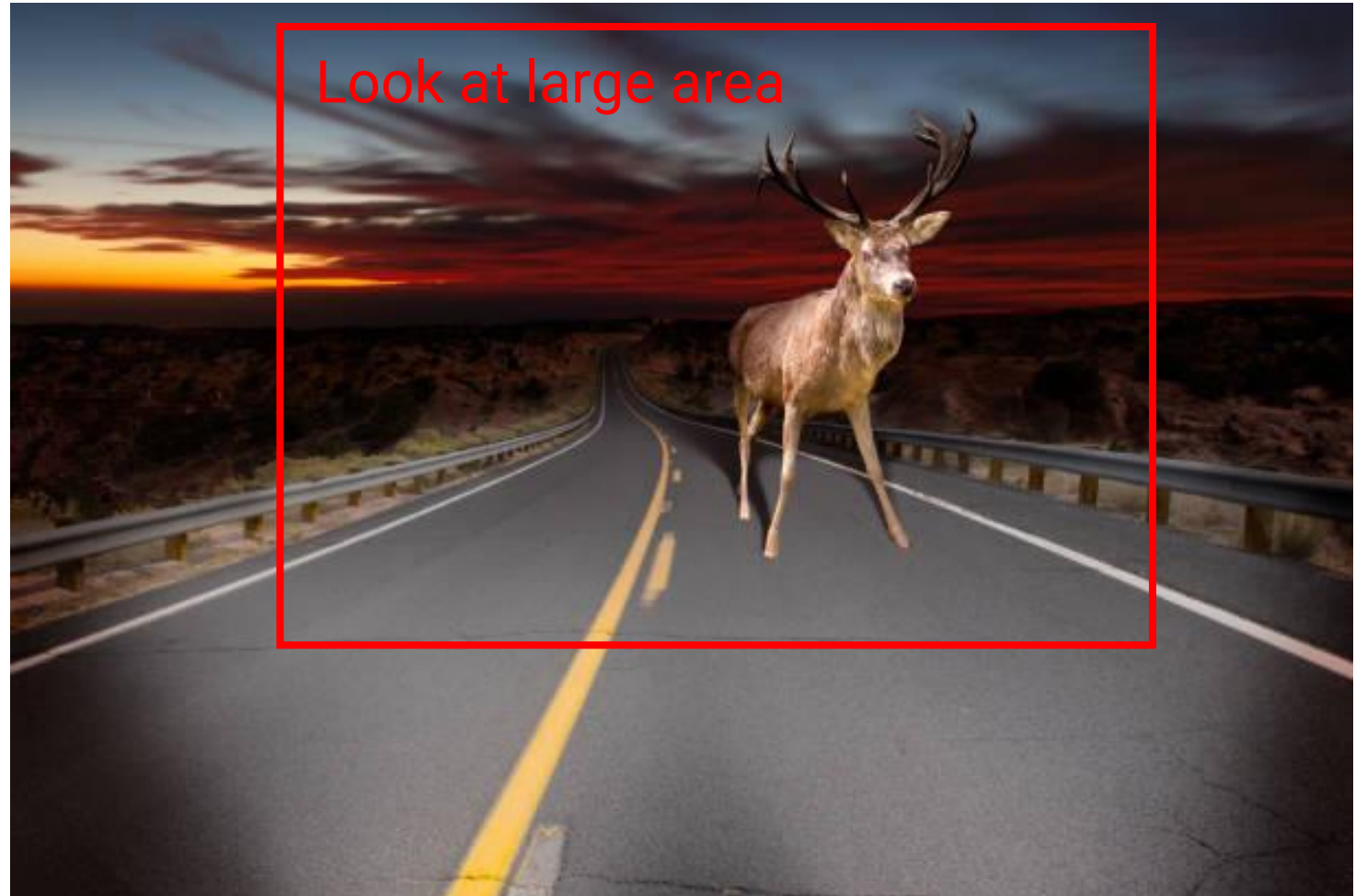
- Turn left?



# A hierarchy of features

---

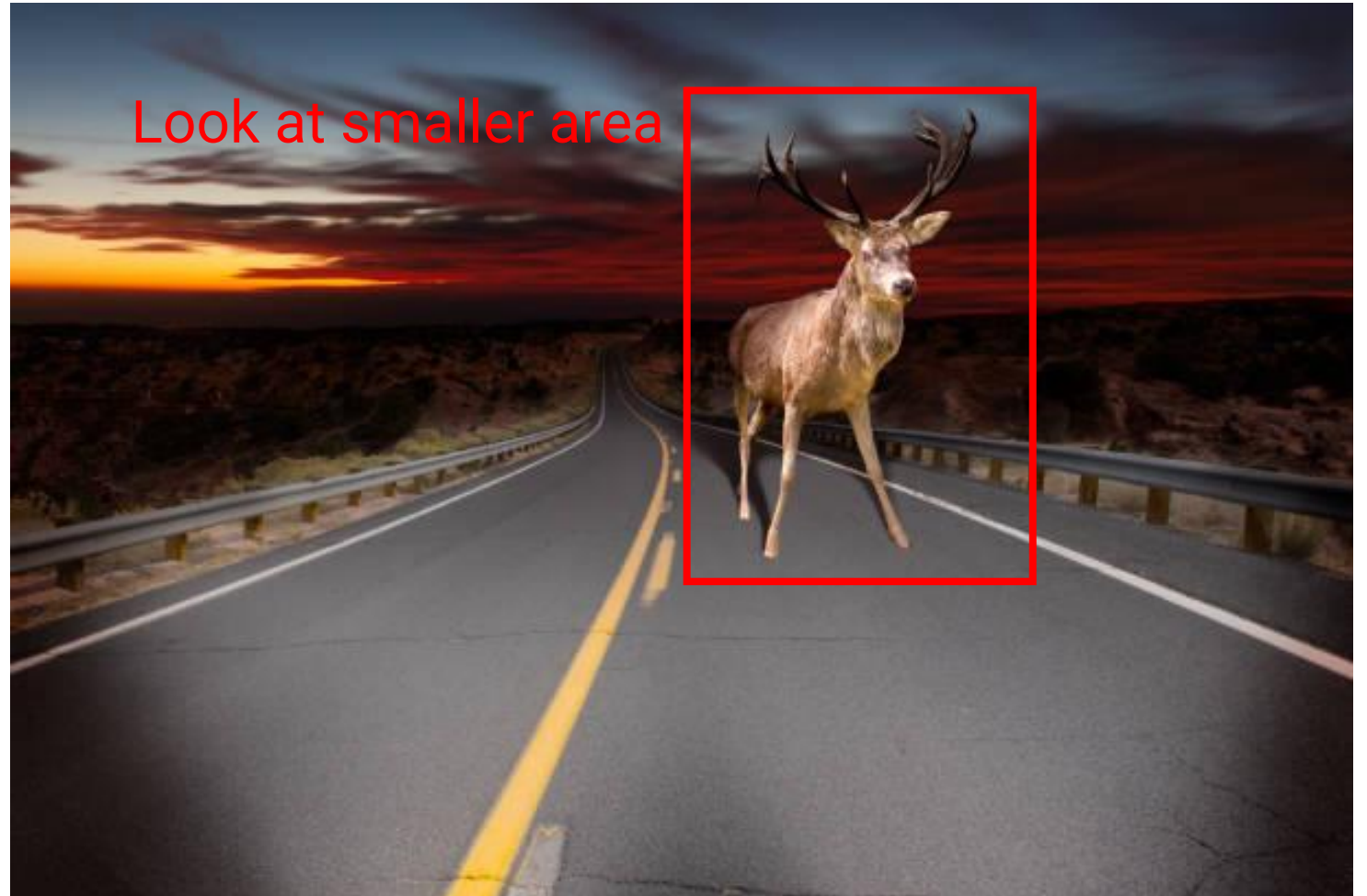
- Turn left?
- **Front is clear?**



# A hierarchy of features

---

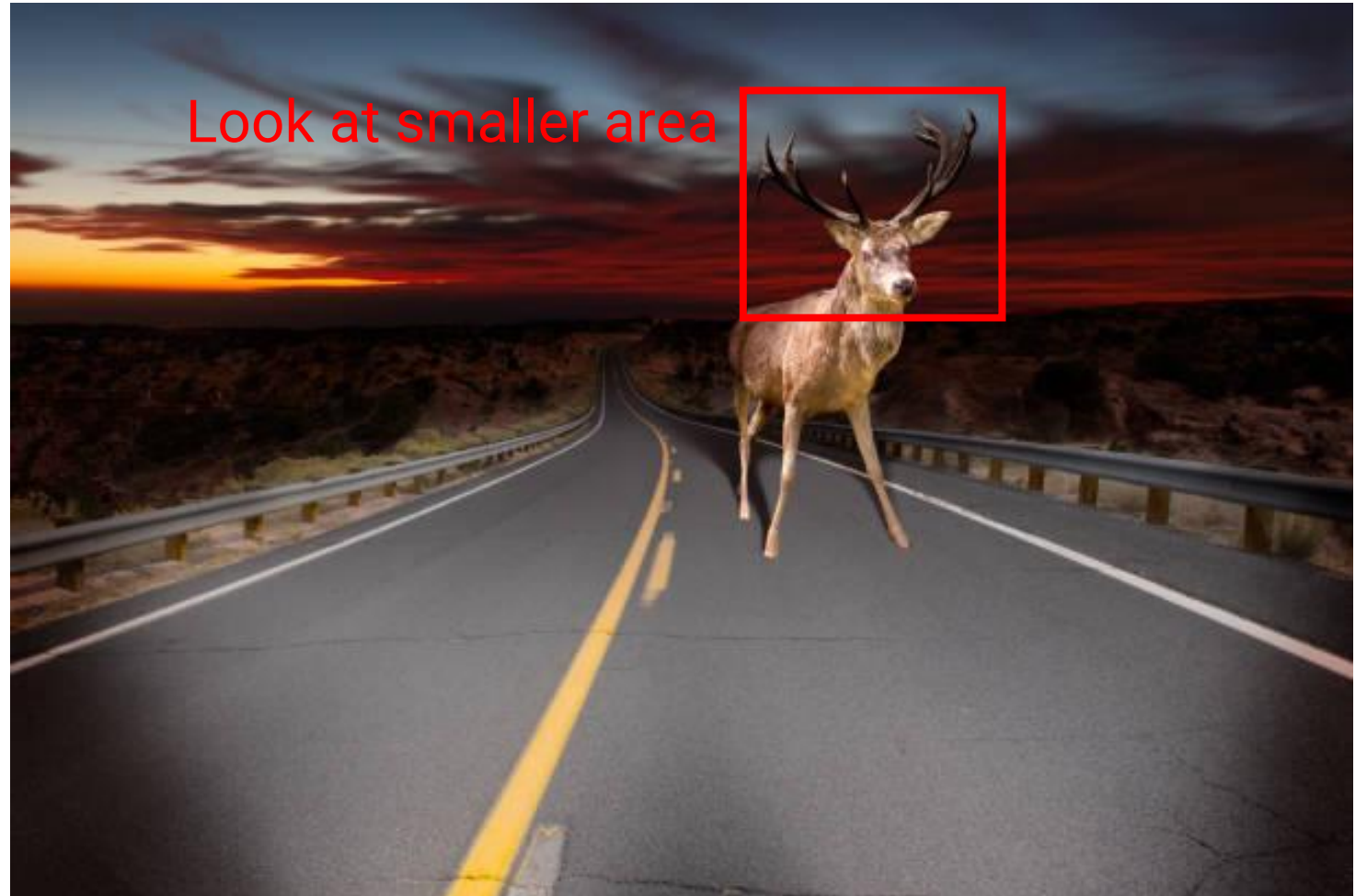
- Turn left?
- Front is clear?
- **Is object a moose?**



# 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?**

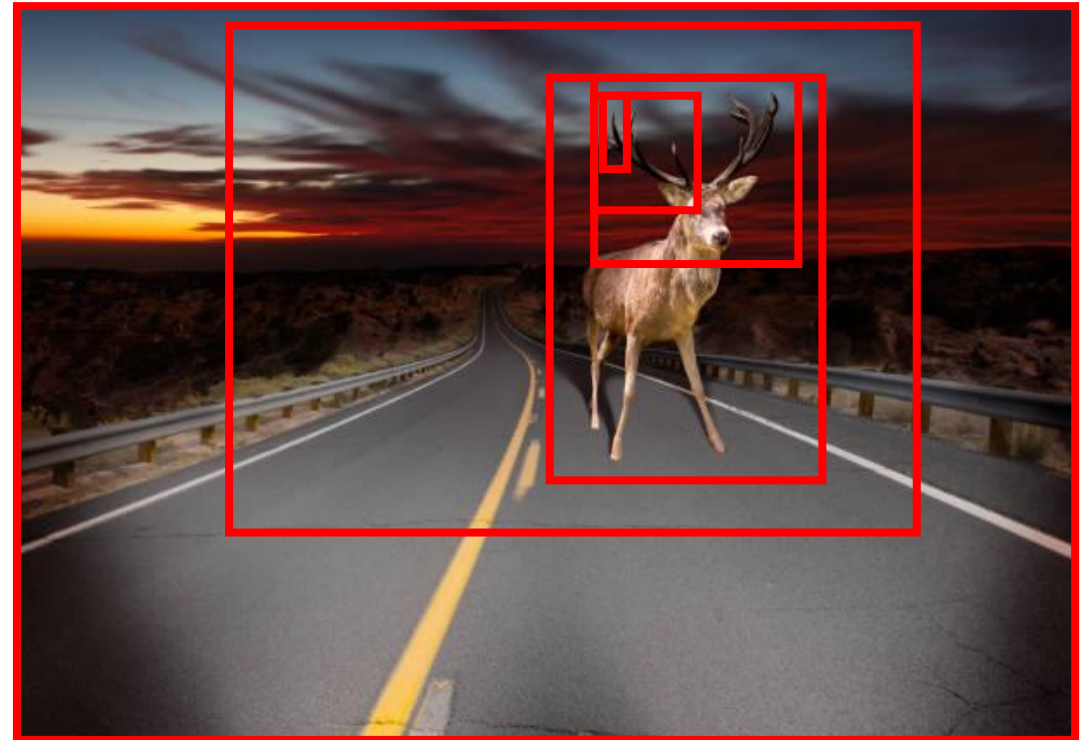




# Learning features hierarchically

---

- 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

---

- Extracting features with convolutions
- Convolutional Neural Networks
- Computer vision tasks

# A moose detector

---

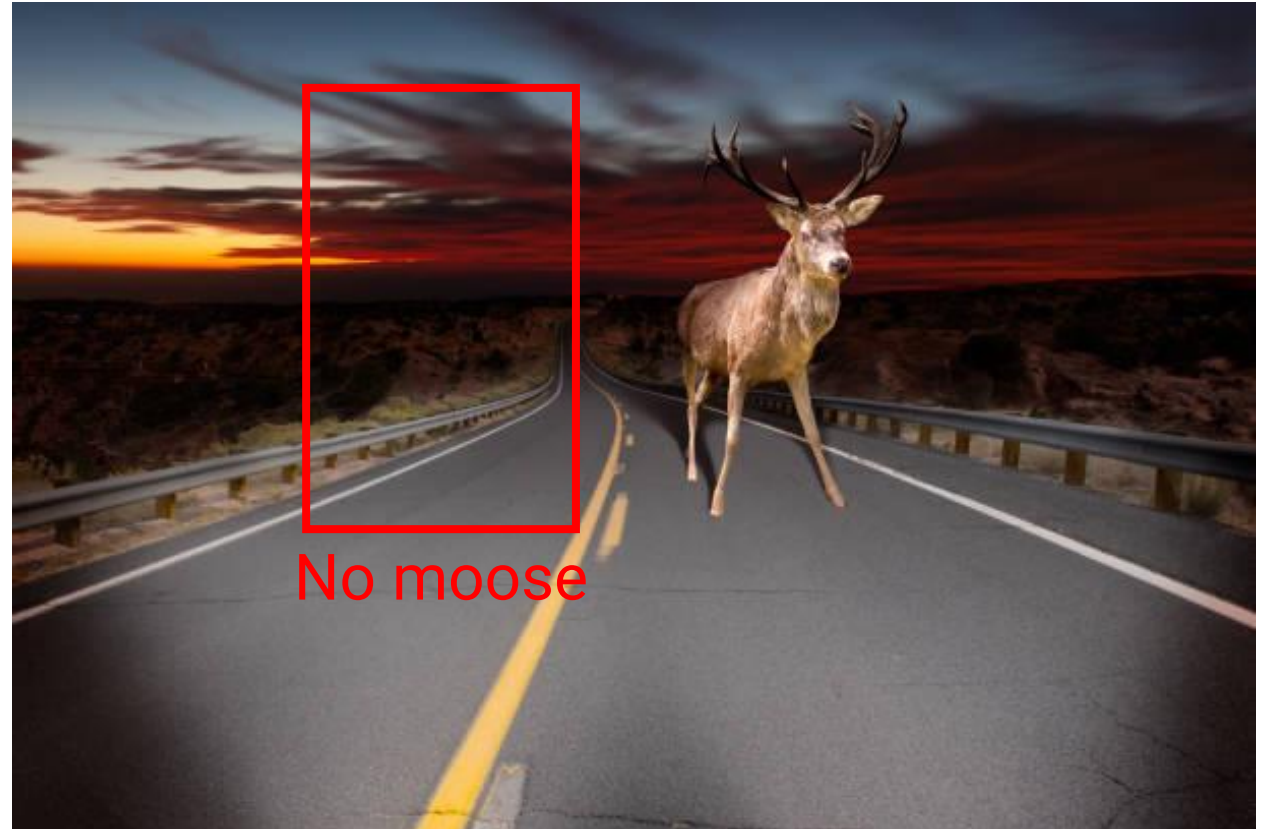
- Suppose you have a classifier that can tell if a region has a moose
- How to use it to create a useful feature vector?
- Slide it over each region and check if there's a moose there!



# A moose detector

---

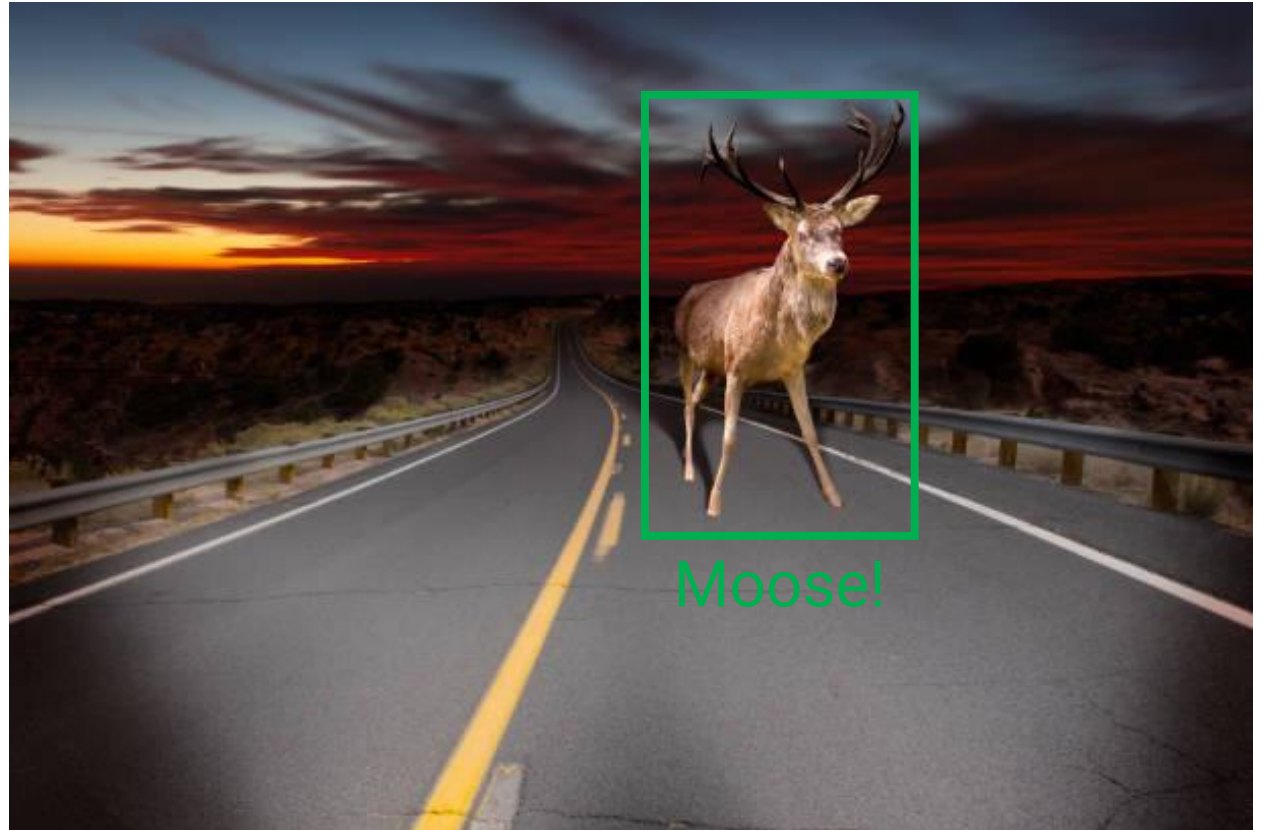
- Suppose you have a classifier that can tell if a region has a moose
- How to use it to create a useful feature vector?
- Slide it over each region and check if there's a moose there!



# A moose detector

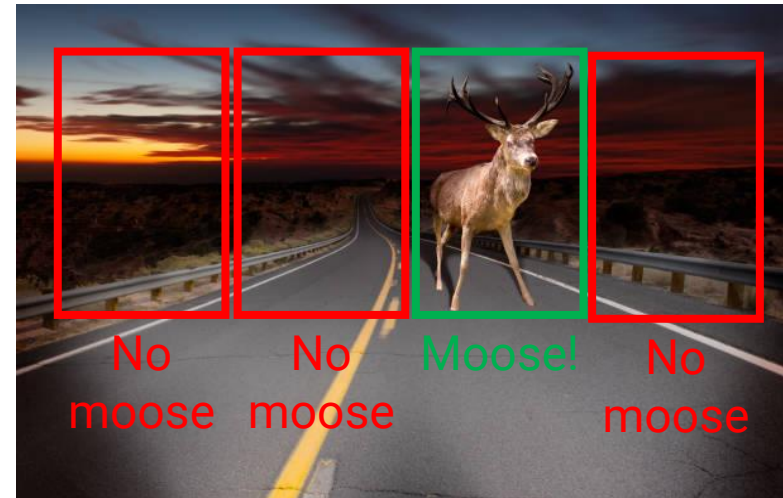
---

- 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!



Learned features  $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \end{pmatrix}$

- Moose in far left?
- Moose in center left?
- Moose in center right?
- Moose in far right?

# 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


**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**

→  
Dot product  
kernel &  
each image  
patch

3	-1	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

**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

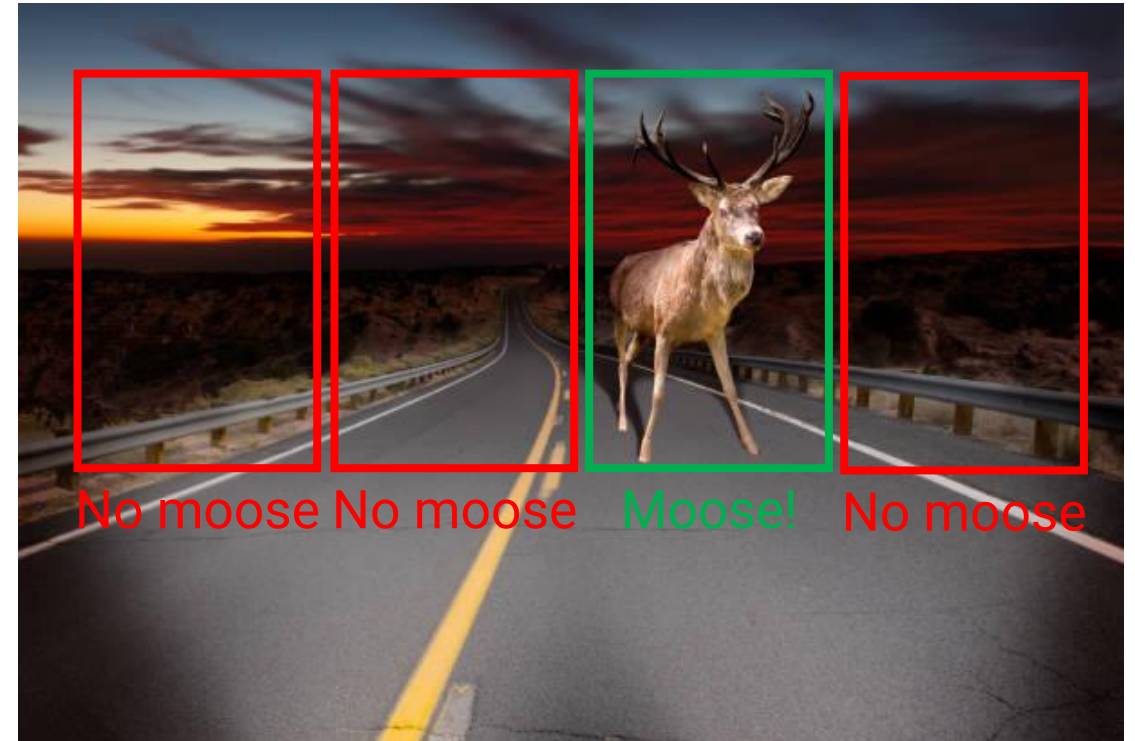
---

- 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



# 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

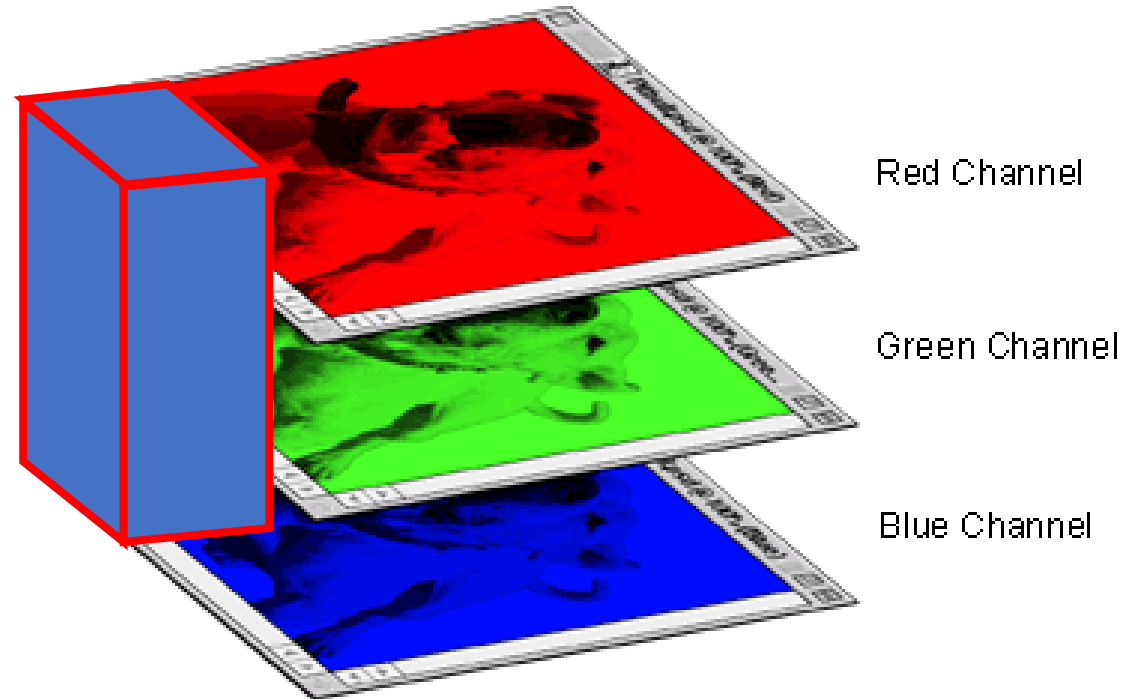
**Output**  
(size 12)

- Let's count parameters needed
  - Convolutional layer with  $K=3$ 
    - Kernel =  $3 \times 3 = 9$  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

---

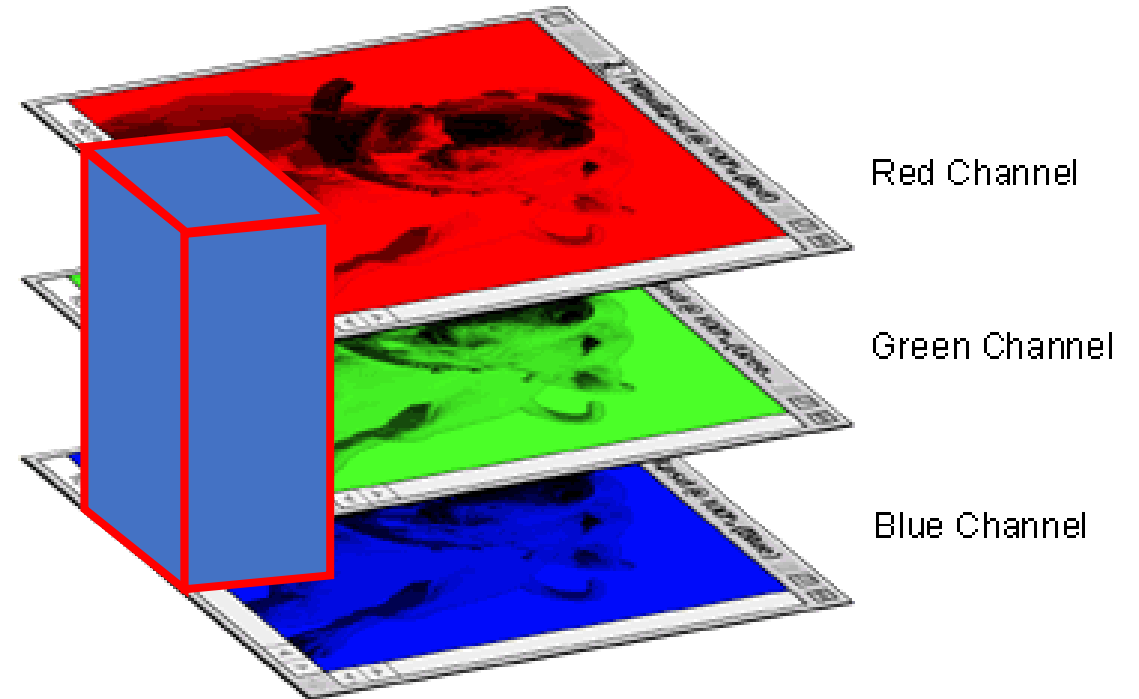
- Input may have multiple input channels
  - Color image has 3 “channels” for red/green/blue
  - Input is actually  $3 \times W \times H$
  - Solution: Kernel must be of size  $C_{in} \times K \times K$



# Multiple Input Channels

---

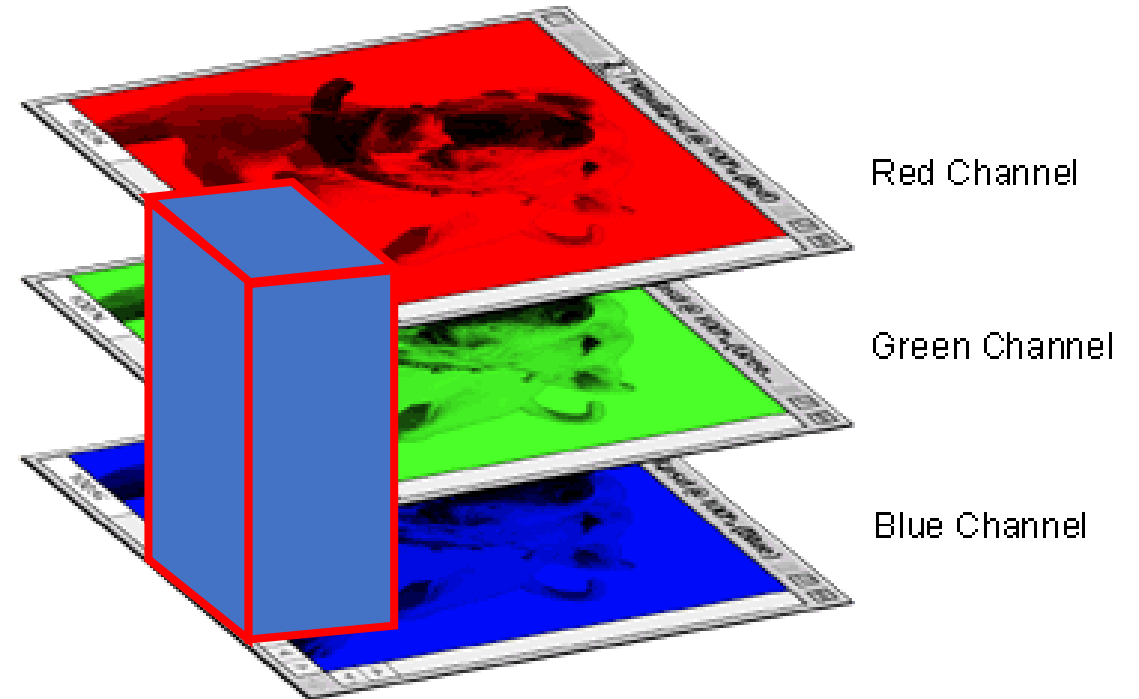
- Input may have multiple input channels
  - Color image has 3 “channels” for red/green/blue
  - Input is actually  $3 \times W \times H$
  - Solution: Kernel must be of size  $C_{in} \times K \times K$



# Multiple Input Channels

---

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



# 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  $C_{\text{out}} \times C_{\text{in}} \times K \times K$

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

Kernel[0,0,:,:]

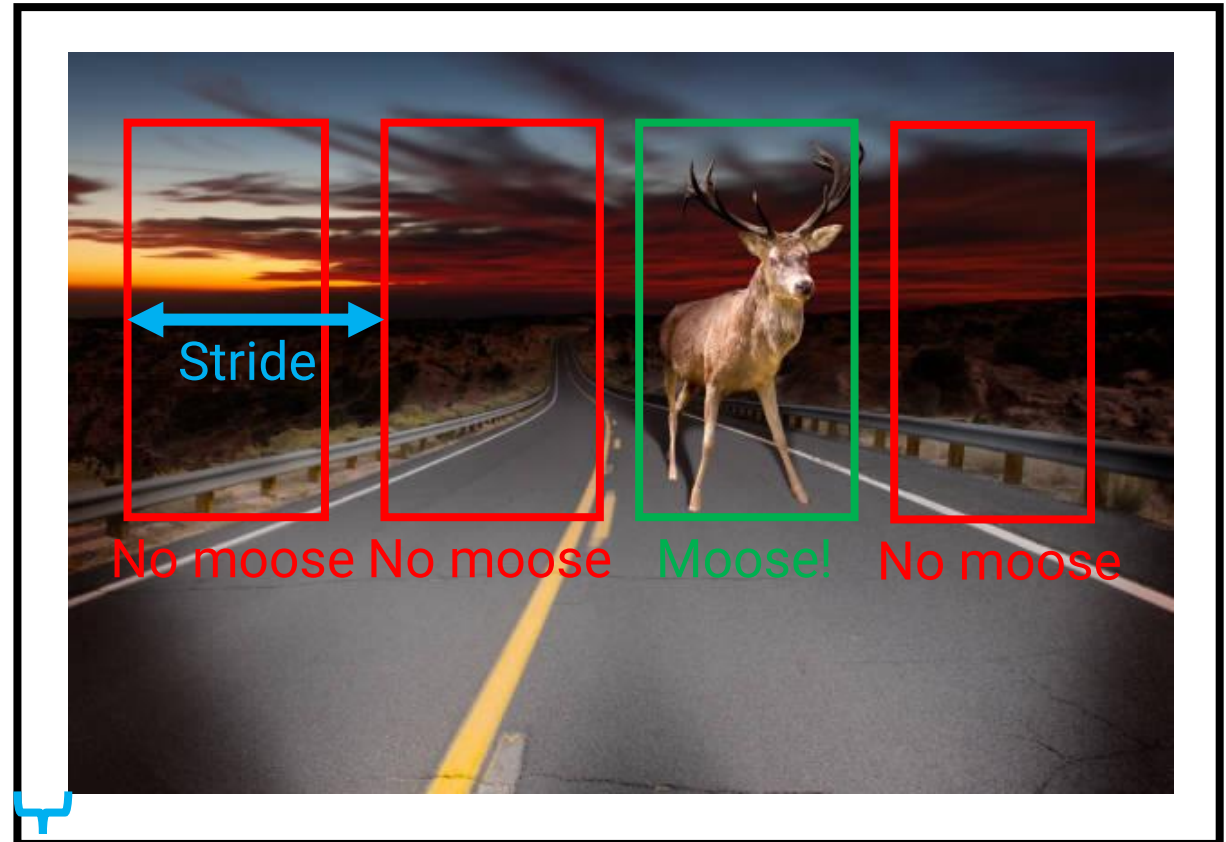
-1	-1	-1
2	2	2
-1	-1	-1

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



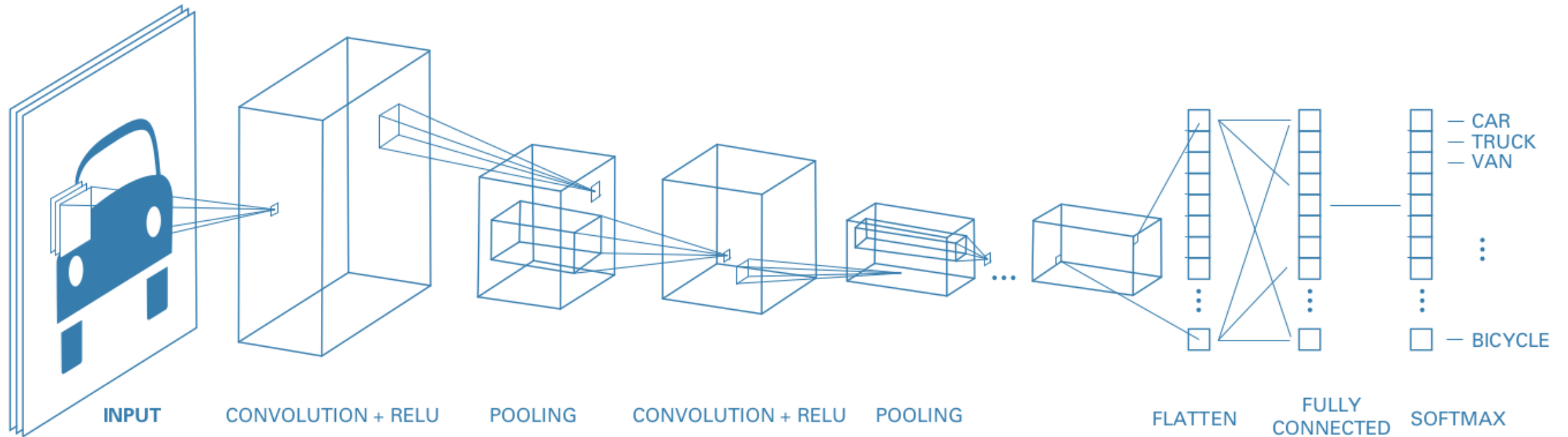
Padding

# Outline

---

- Extracting features with convolutions
- **Convolutional Neural Networks**
- Computer vision tasks

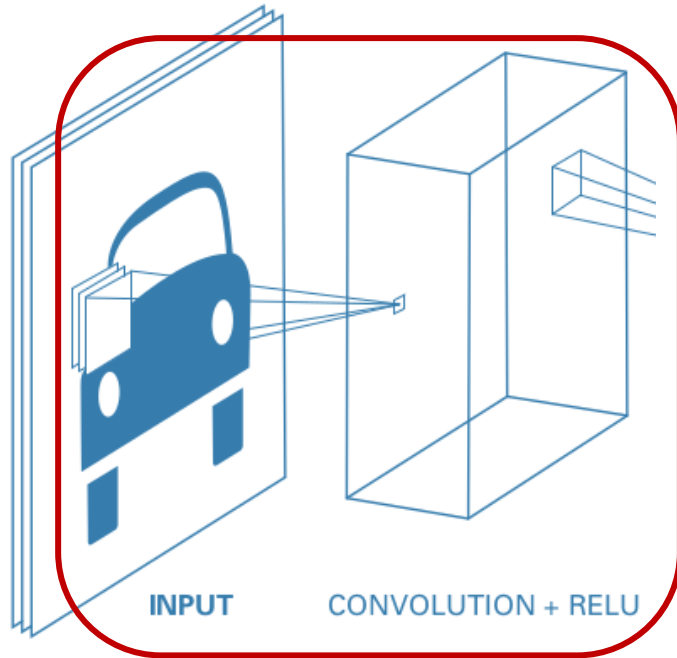
# Convolutional Neural Networks (CNNs)



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

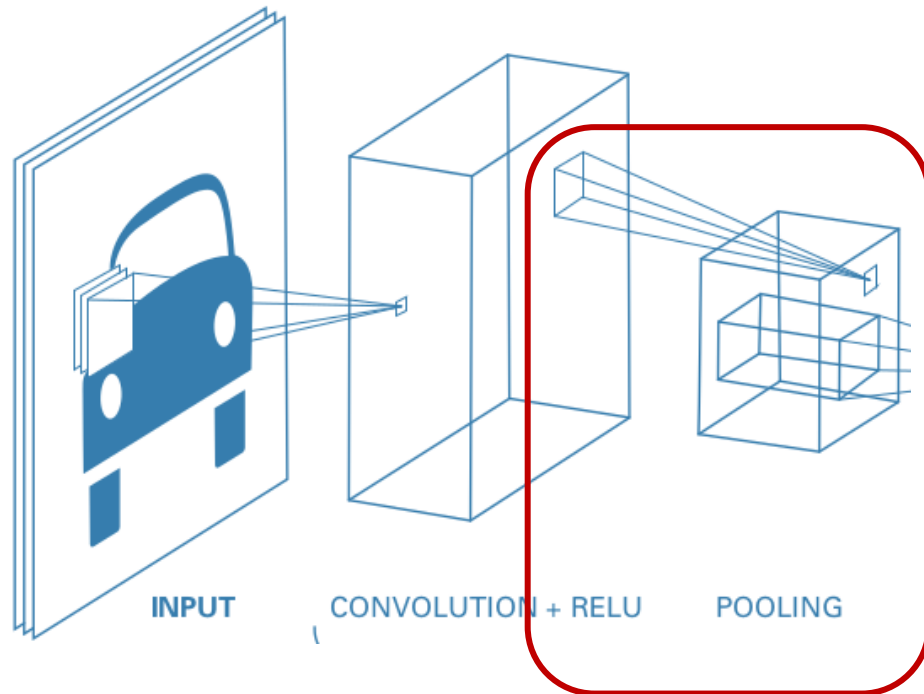
# Convolutional Layers

---



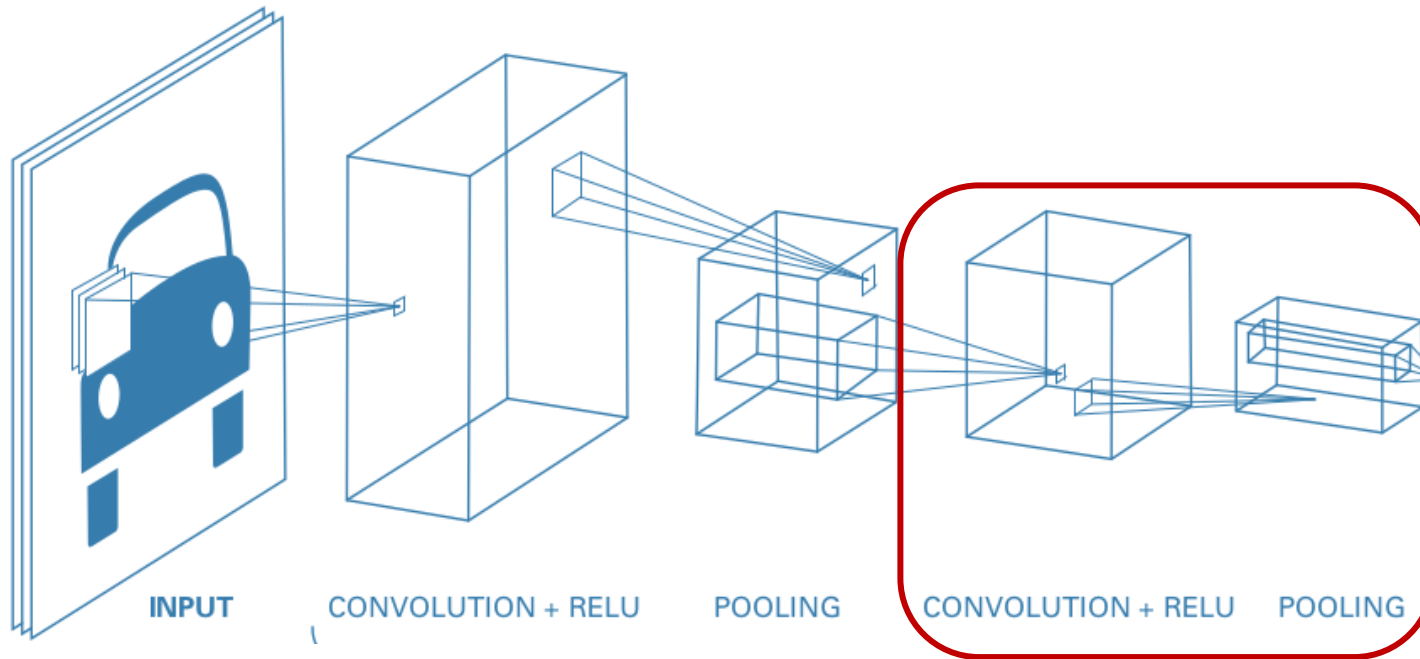
- First step: Convolutional Layer + ReLU
- Analogous to Linear layer + ReLU
  - Convolutional layer is just a special type of linear layer with local receptive fields & weight sharing!
  - So we again want to apply a non-linearity after the linear operation
- ReLU is standard for CNNs

# Pooling



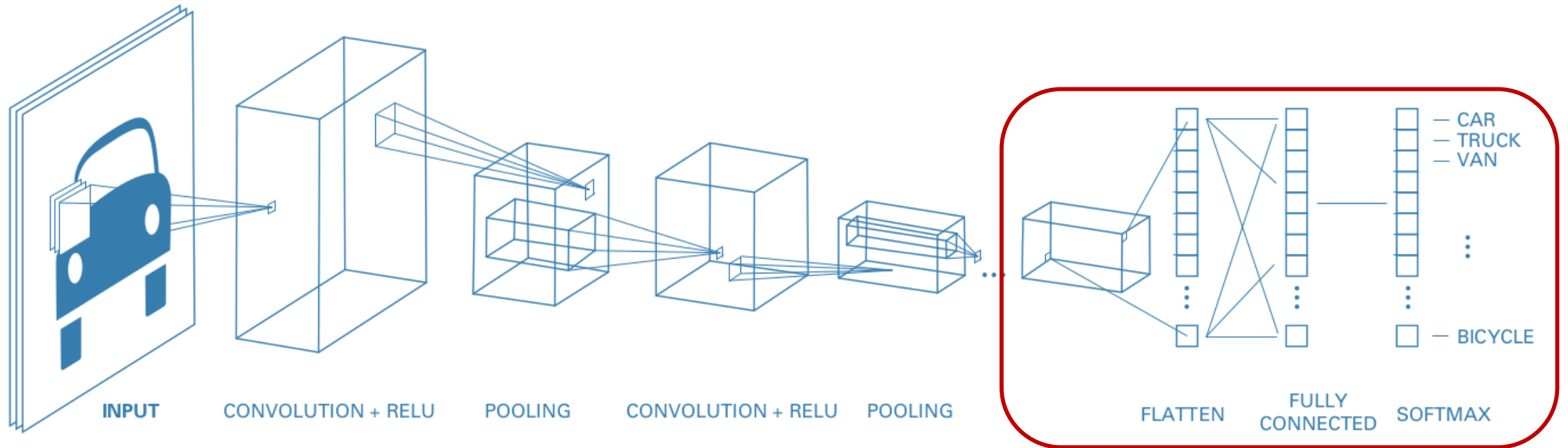
- Goal: Make receptive field bigger as we process the image
  - Early: Look for edges (small patch)
  - Later: Look for moose (larger patch)
- How do we do this? Pooling!
- Effectively we reduce resolution of input by a factor of  $P$  (often  $P=2$ )
  - Average pool: Average in each  $2 \times 2$ 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

# More Conv + ReLU + Pool



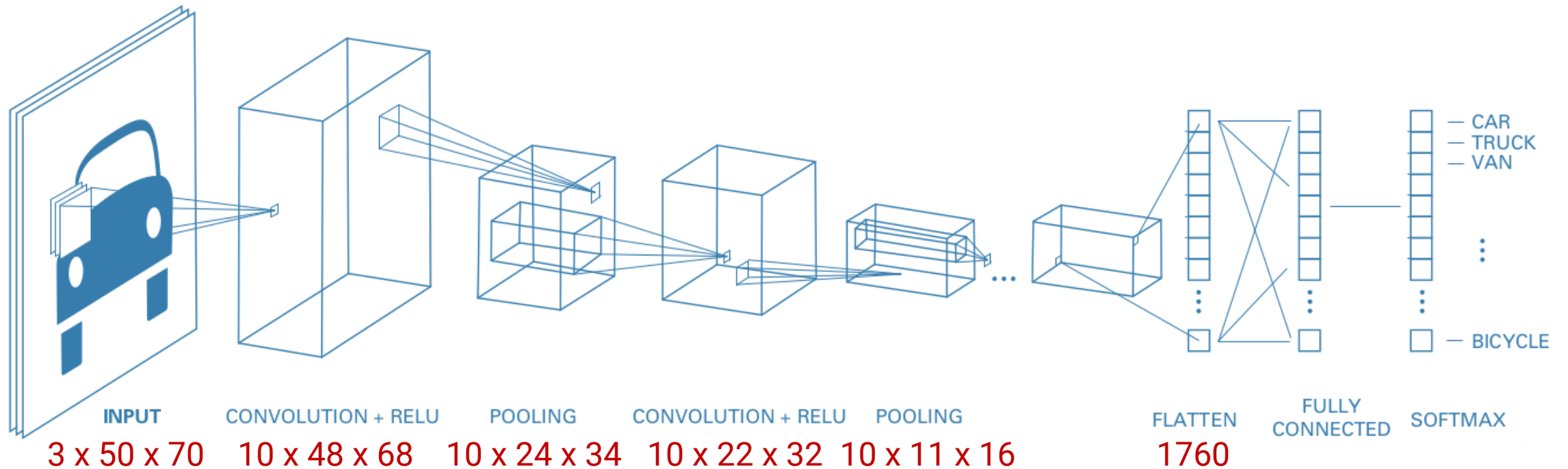
- Can stack multiple Conv + ReLU + pool blocks
- Similar to increasing number of hidden layers in MLP
- Deeper layers convolutional layers have larger effective receptive field
  - Can learn higher-level concepts

# 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



# Outline

---

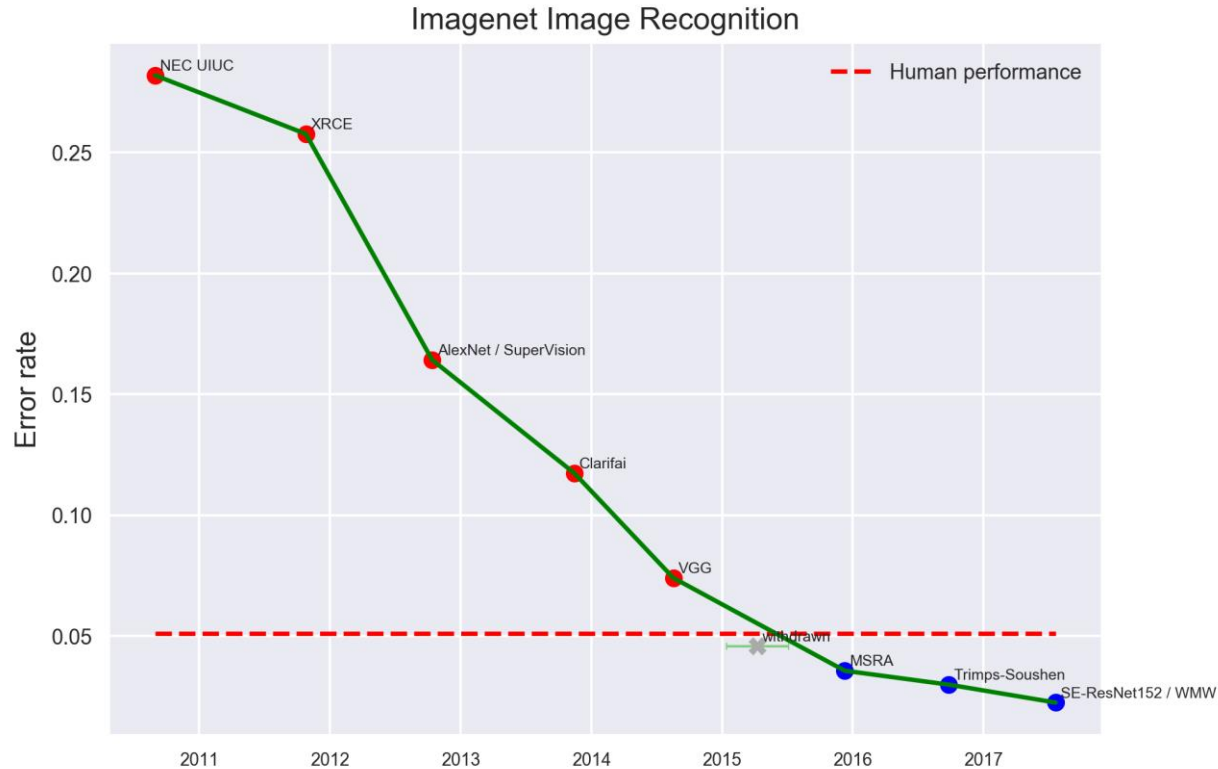
- Extracting features with convolutions
- Convolutional Neural Networks
- **Computer vision tasks**

# Image classification

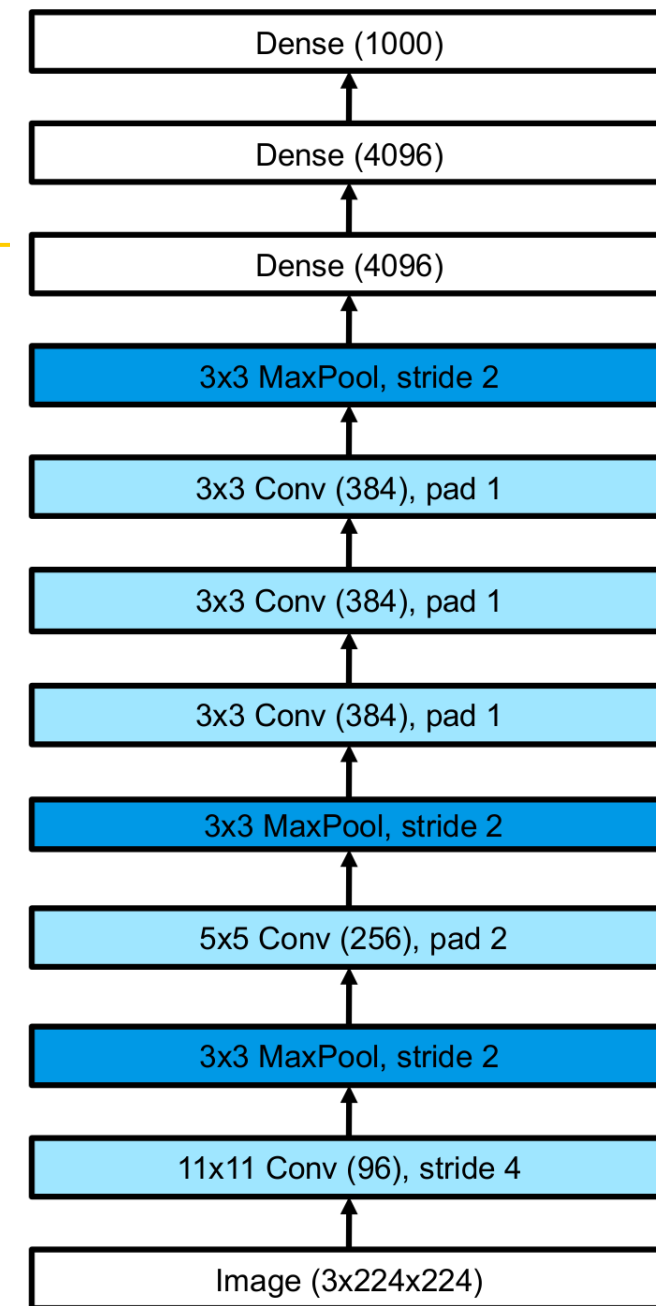


- ImageNet dataset: 14M images, 1000 labels
- **CNNs do very well at these tasks!**

# 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



# Object Detection

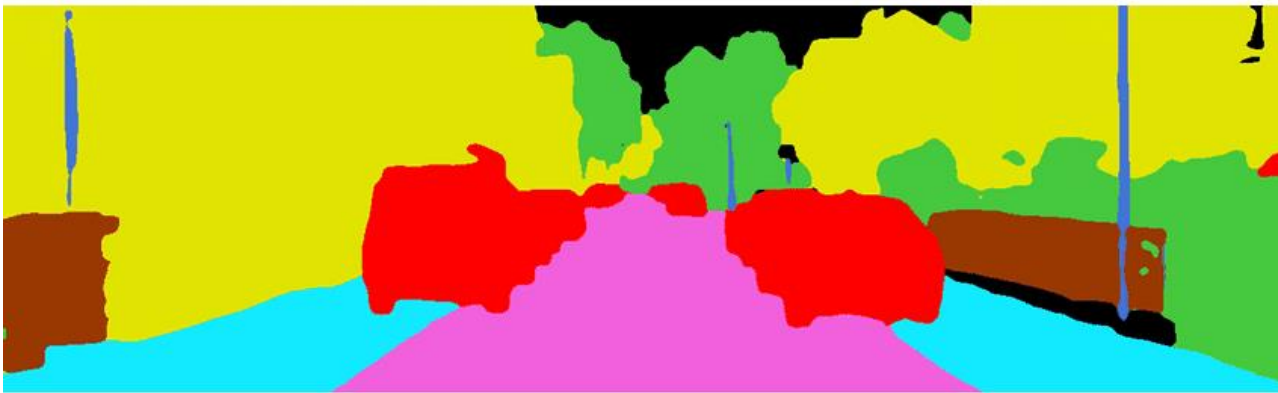










- Task: Identify objects, provide bounding boxes, and label them
- One strategy: Propose candidate bounding boxes, then classify each box (possibly as nothing)

# Semantic Segmentation

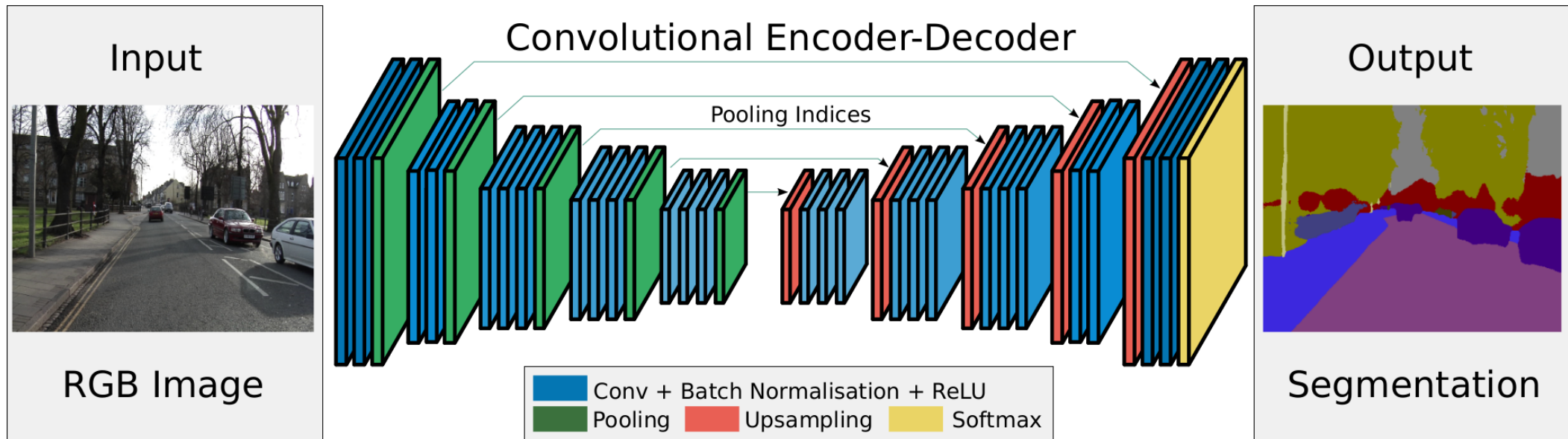


- Task: Predict a class label for each pixel



 Road	 Sidewalk	 Building	 Fence
 Pole	 Vegetation	 Vehicle	 Unlabel

# Semantic Segmentation



- One strategy: Encoder-Decoder
  - First do conv + ReLU + pooling as before
  - Then do upsampling + conv + ReLU to generate an output of original size

# 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**