Universität Konstanz

Lecture 7 Deep Learning – Part 2 Convolutional Neural Network

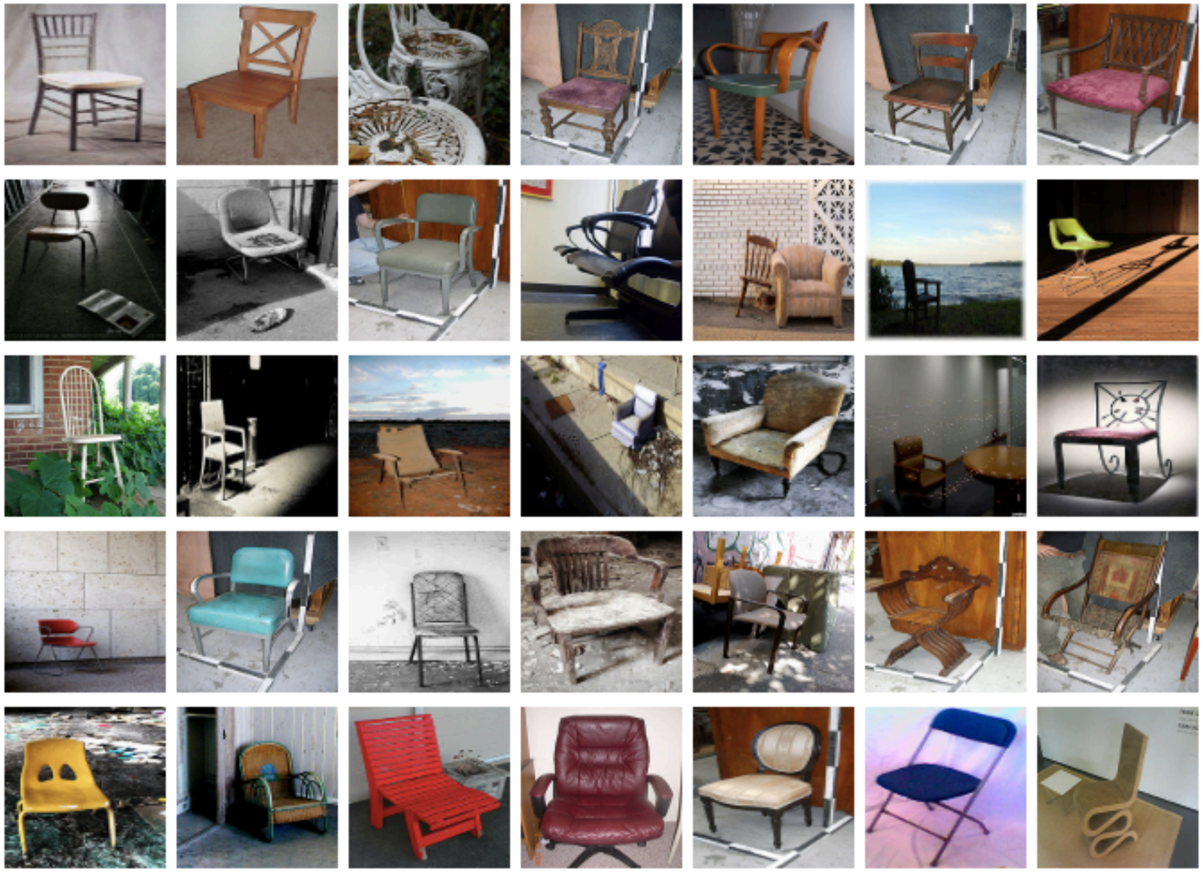
Dr. Hanhe Lin
Dept. of Computer and Information Science
University of Konstanz

Challenges of object recognition

- Occlusion: real scenes are cluttered with other objects:
  - Hard to tell which parts go to which object
  - Part of an object may be hidden behind other objects
- Lighting: the pixel intensities are determined by the lighting on the objects
- Deformation: object can deform in a variety of non-affine ways, for example, written number 7

Challenges of object recognition

- Affordances: object classes are often defined by how they are used, e.g., chairs are designed sitting so they have a variety of shapes



Challenges of object recognition

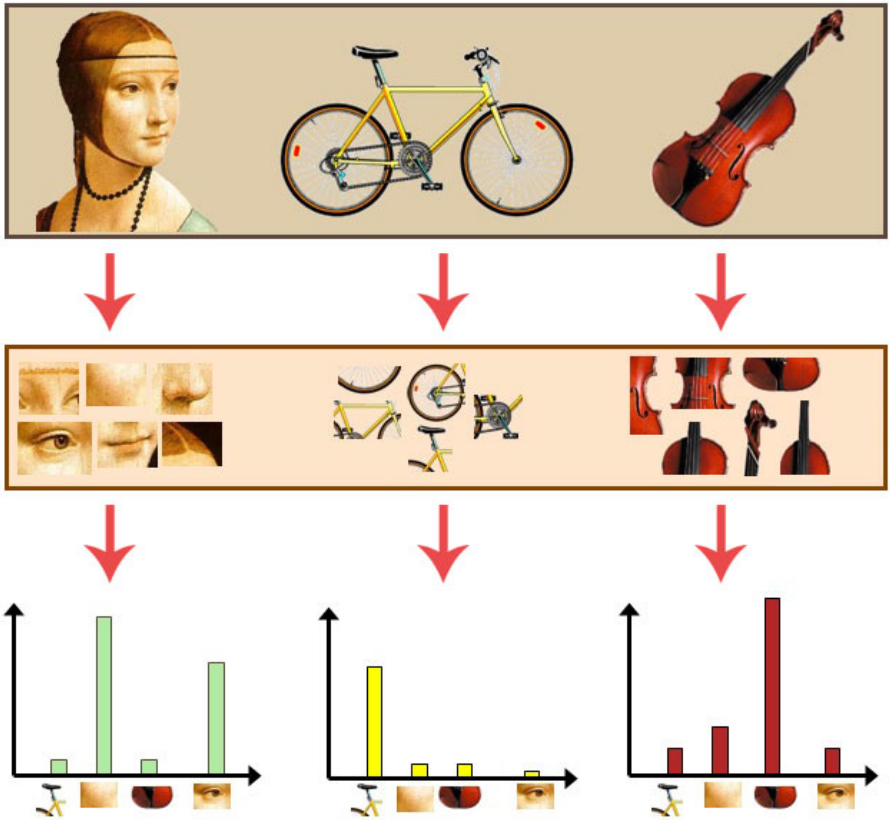
- Viewpoint: changes in viewpoint cause the changes in images that standard learning methods cannot cope with.

Ways to achieve viewpoint invariance

- Use redundant invariant features
- Put a box around the object and use normalized pixels
- Use a hierarchy of parts that have explicit poses relative to the camera
- Use replicated features with pooling, i.e., convolutional neural networks
- . . .

The invariant features approach

- “Local” approach
- Extract a large, redundant set of features that are invariant under transformations, e.g., SIFT
- But for recognition, we must avoid forming same features from parts of different objects
- With enough invariant features, we can represent an object by Bag-of-Words model (BoW)



The brute force normalization approach

- When training the recognizer, use well-segmented, upright images to fit correct box
- At the test time try all possible boxes in a range of positions and scales
  - This approach is widely used for detecting upright things like faces and house numbers in un-segmented images
  - Although it can provide a certain degrees of freedom, you must train multiple recognizers for same class. For example, front face and profile face
  - E.g., HOG for pedestrian detection

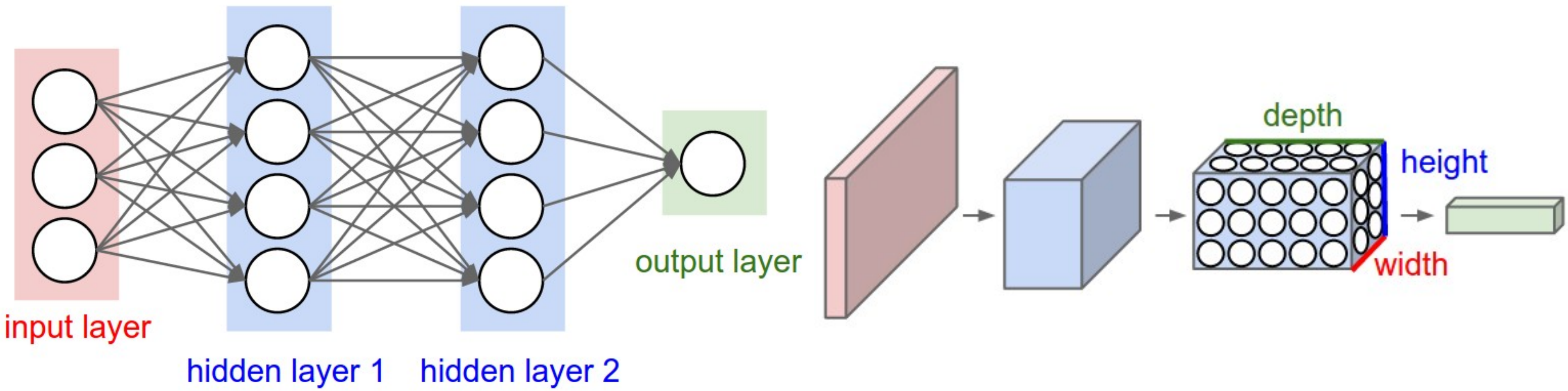
CNNs – Motivation

- Use many different copies of the same feature detector with different positions
  - Could also replicate across scale and orientation, but tricky and expensive
  - Replication greatly reduces the number of free parameters to be learned
- Use several different types of feature detectors, each with its own feature map
  - Allows each patch of image to be represented in several ways

CNNs – Definition

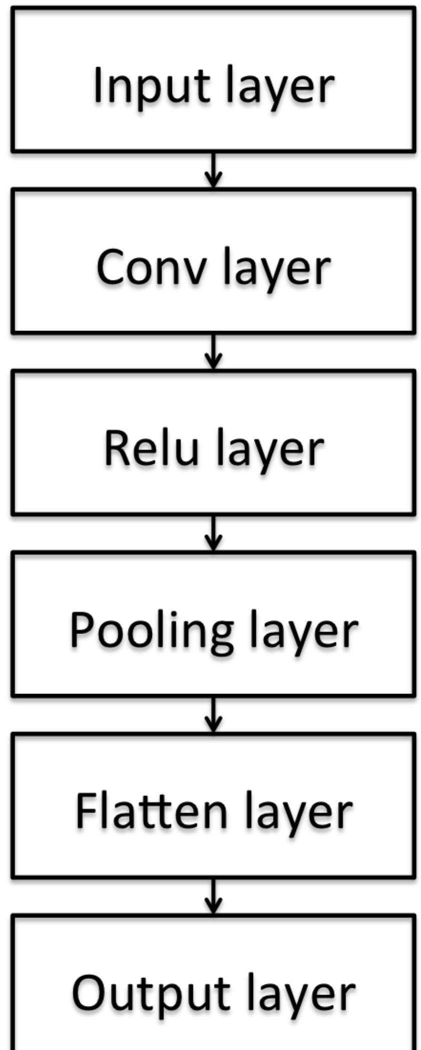
- Convolutional neural networks (CNNs), also known as convolutional networks, ConvNets
- It is a specialized kind of neural network for processing data that has a know grid-like topology
- Examples:
  - 1-D grid - time-series data
  - 2-D grid - gray-scale image
  - 3-D grid - color image or gray-scale video
  - 4-D grid - color video
- CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers

Convolution vs. Feedforward



- Full-connected networks have many parameters, prone to overfitting
- CNN arranges neurons in 3 dimensions: **width**, **height**, **depth**

Overview: a simple CNN



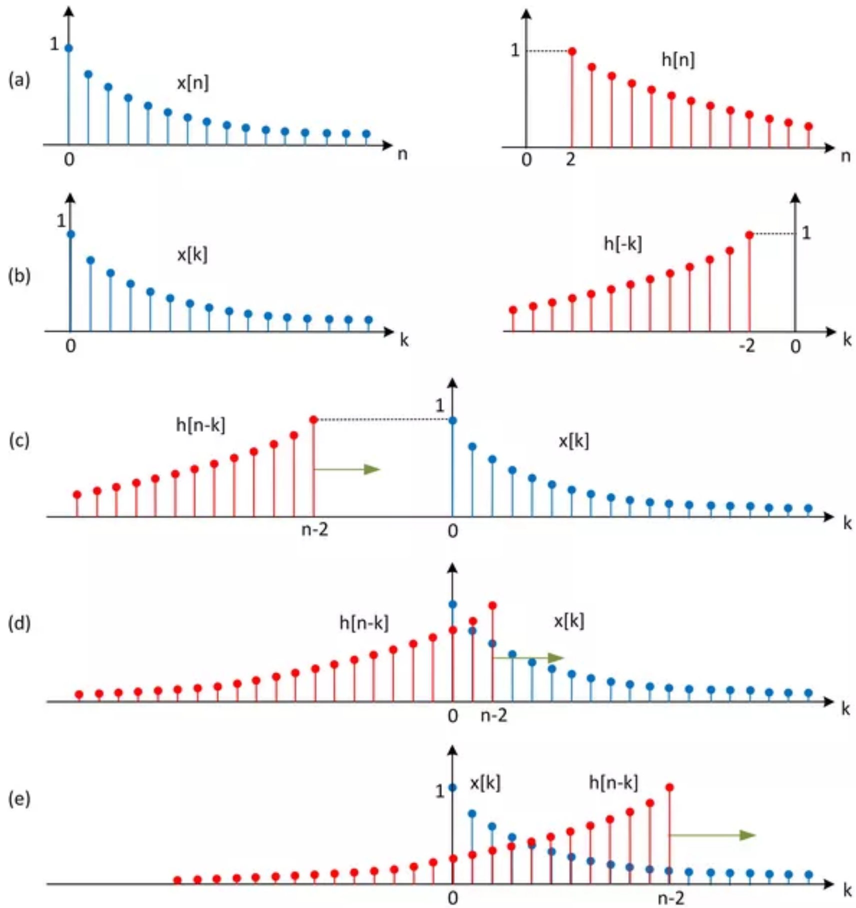
The convolution operation

The convolution operation takes two time-series sequences $x[n]$ and $h[n]$, produce a third sequence $y[n]$, defined as:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k], \quad \forall n$$

In digital signal processing, $x[n]$, $h[n]$, and $y[n]$ are referred to as input, filter, and output respectively.

In CNN terminology, $x[n]$, $h[n]$, and $y[n]$ are referred to as input, kernel, and feature map.



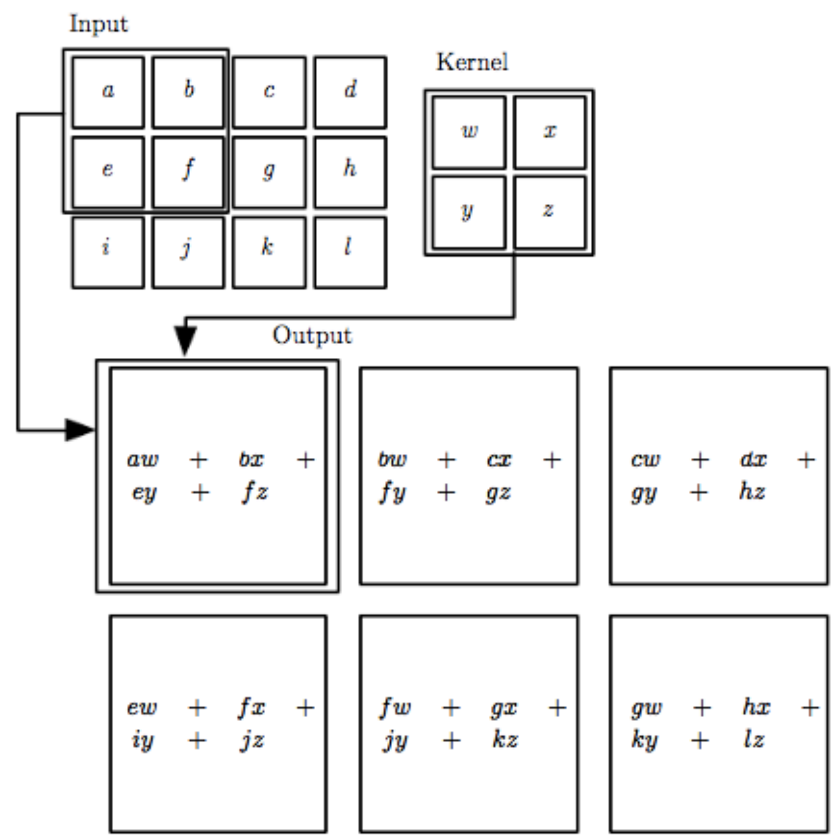
The convolution operation in CNNs

If we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j)=(I*K)(i,j)=\sum_m\sum_n I(i+m,j+n)K(m,n)$$

When an image is three-dimensional, the kernel should also be three-dimensional

What is the difference between these two convoluion operations?



The convolution operation in CNNs

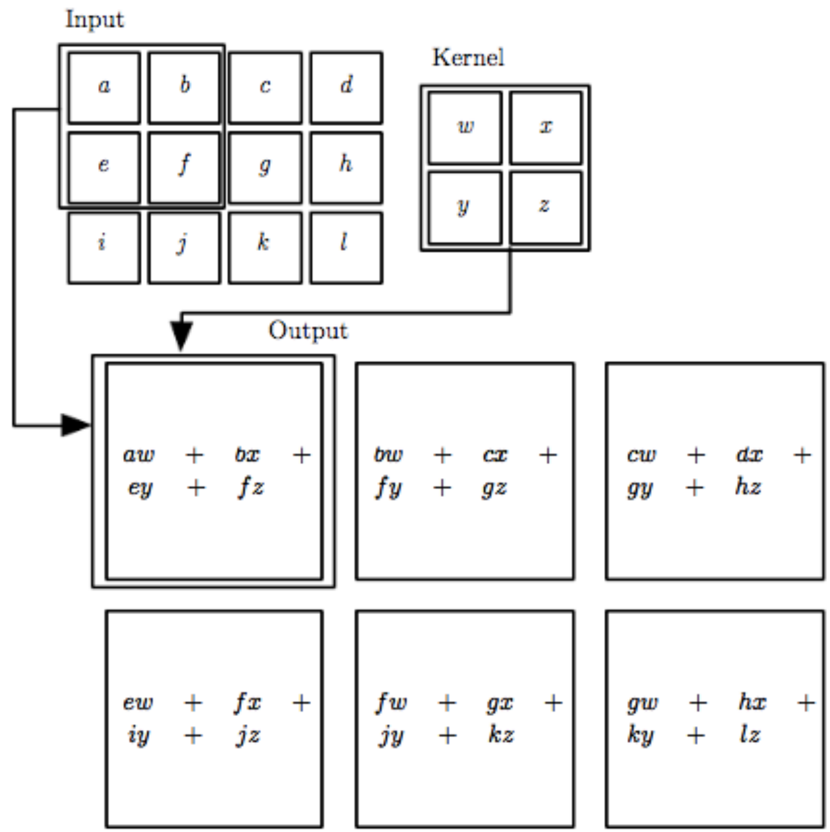
If we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

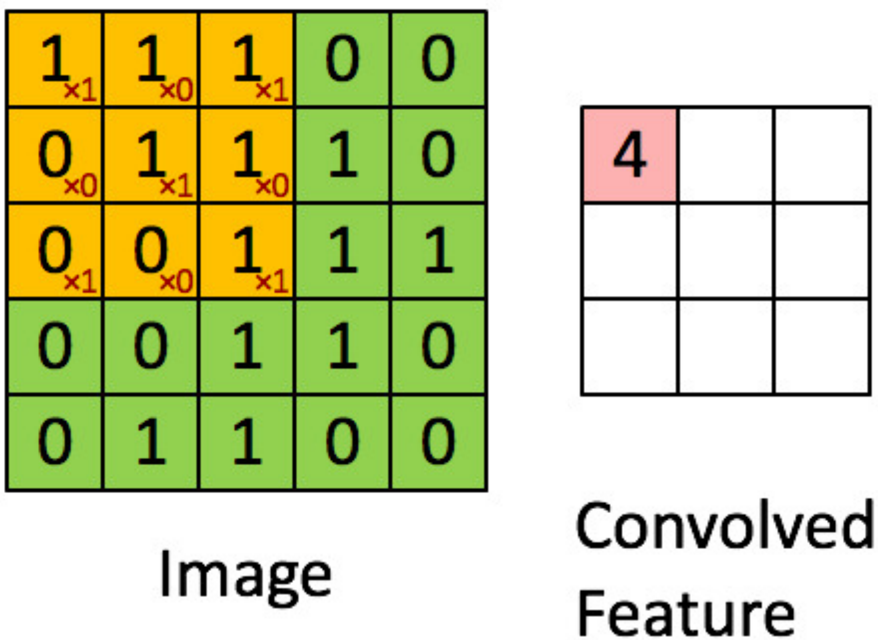
When an image is three-dimensional, the kernel should also be three-dimensional

What is the difference between these
 two convoluion operations?

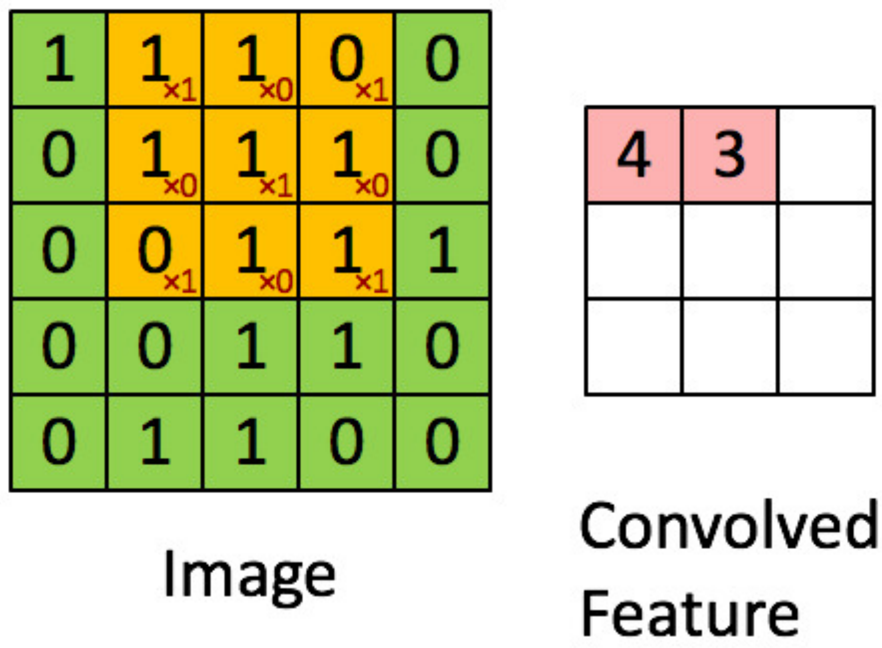
- Without flipped the kernel!
- Many deep learning libraries implement cross-correlation but call it convolution



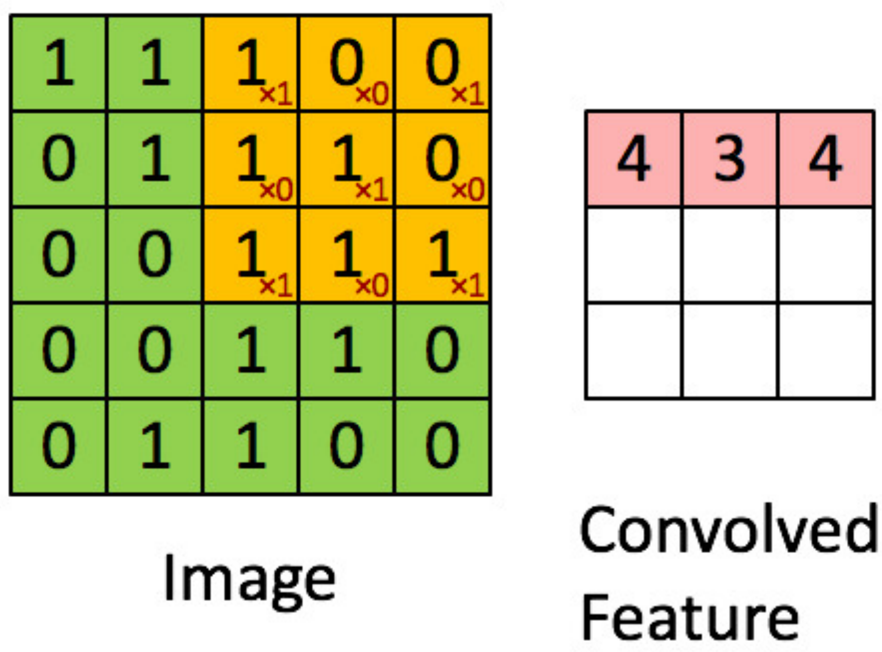
Example: convolution operation



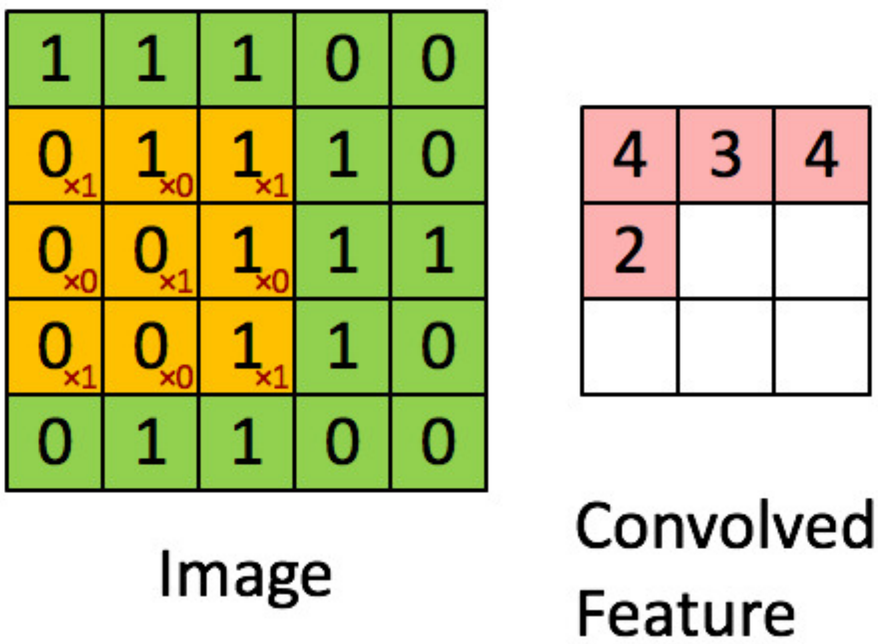
Example: convolution operation



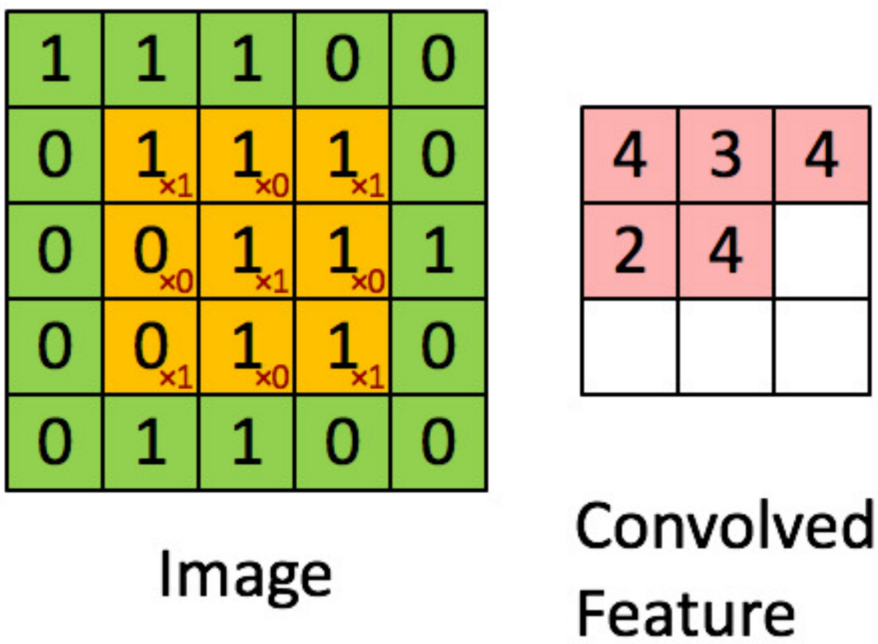
Example: convolution operation



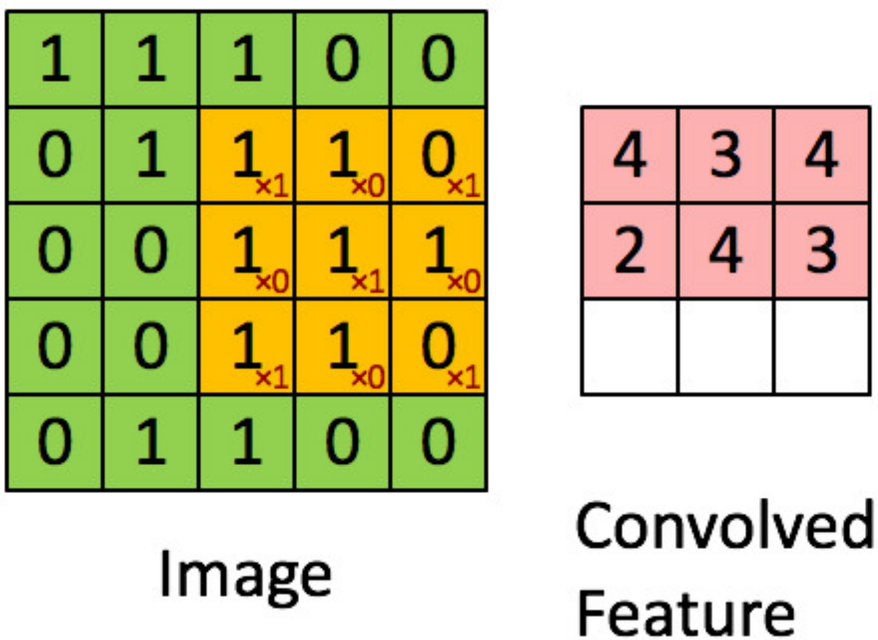
Example: convolution operation



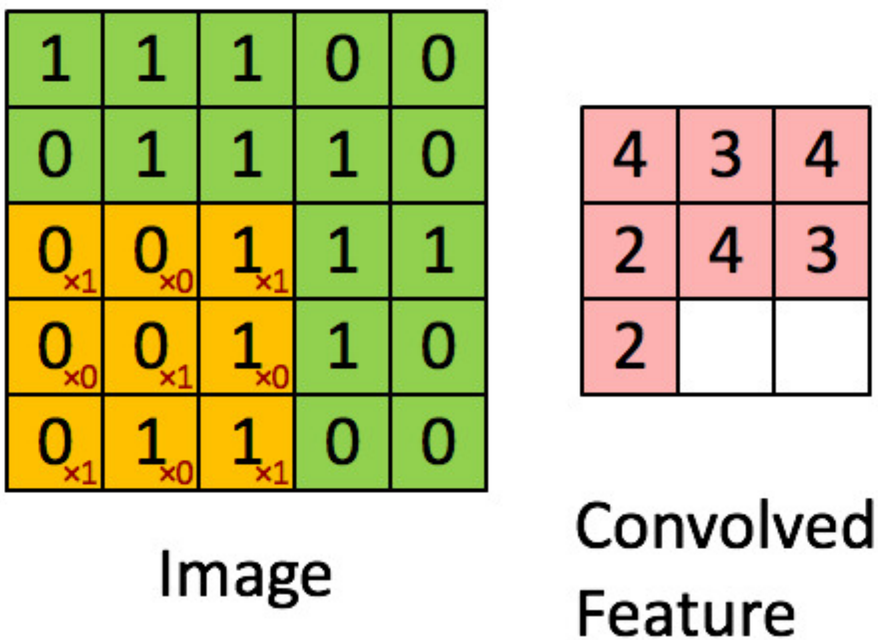
Example: convolution operation



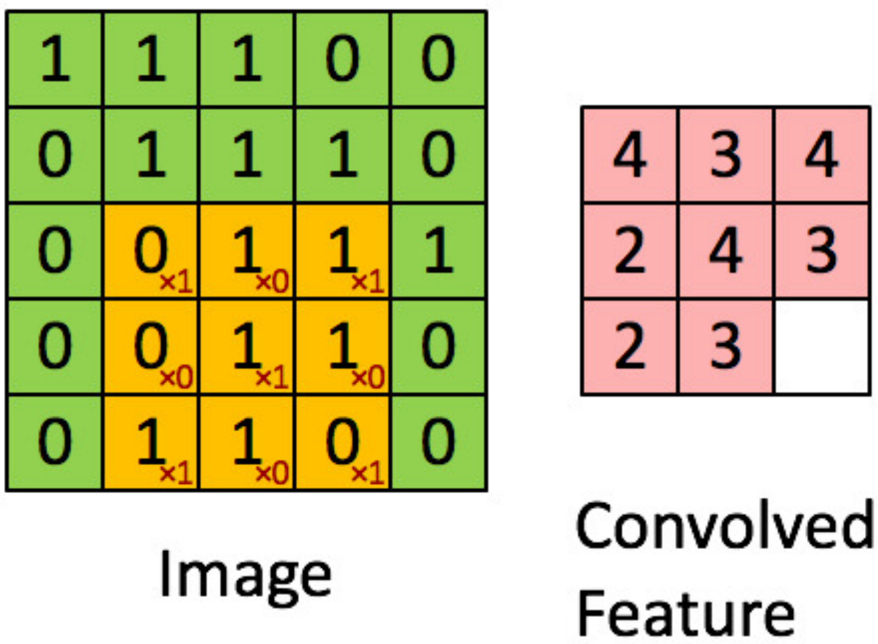
Example: convolution operation



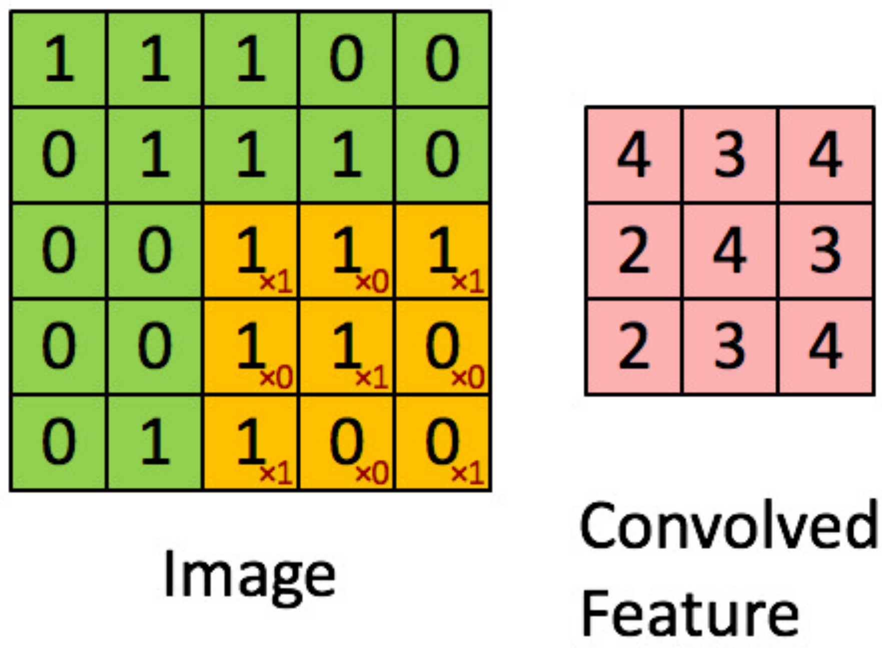
Example: convolution operation



Example: convolution operation



Example: convolution operation

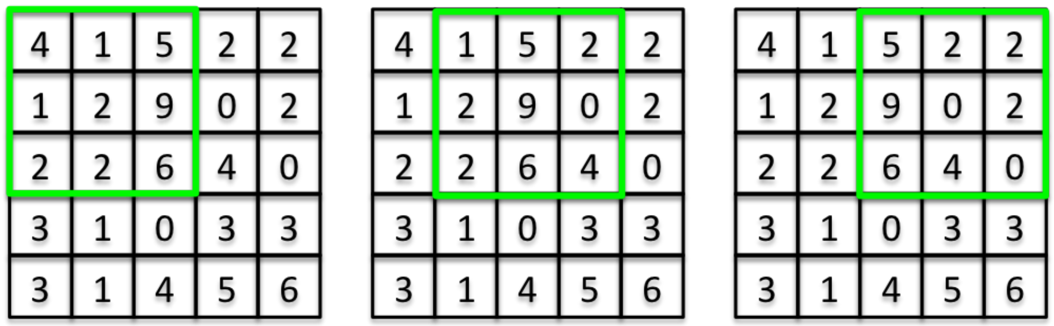


Hyper-parameters in conv layer

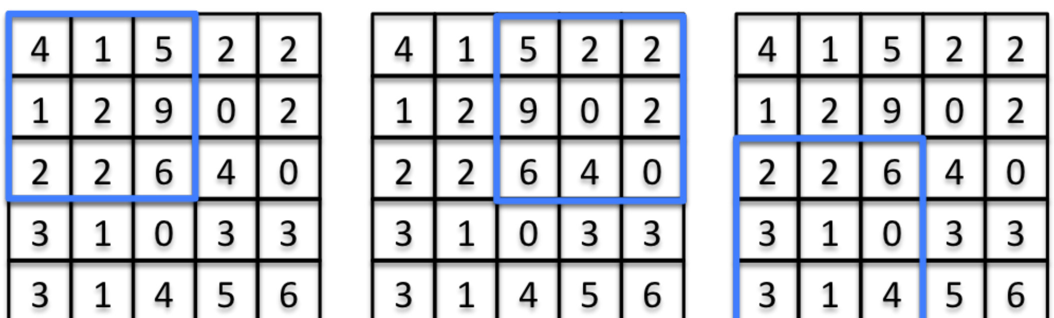
- Depth: corresponds to the number of kernels we would like to use, each learns to look for something different in the input
- Kernel size: large kernel size increases computation
- Stride
- Zero-padding

Stride

- Stride is the number of pixels to slide the kernel, where large stride produce smaller output spatially



Kernel size of 3, stride 1



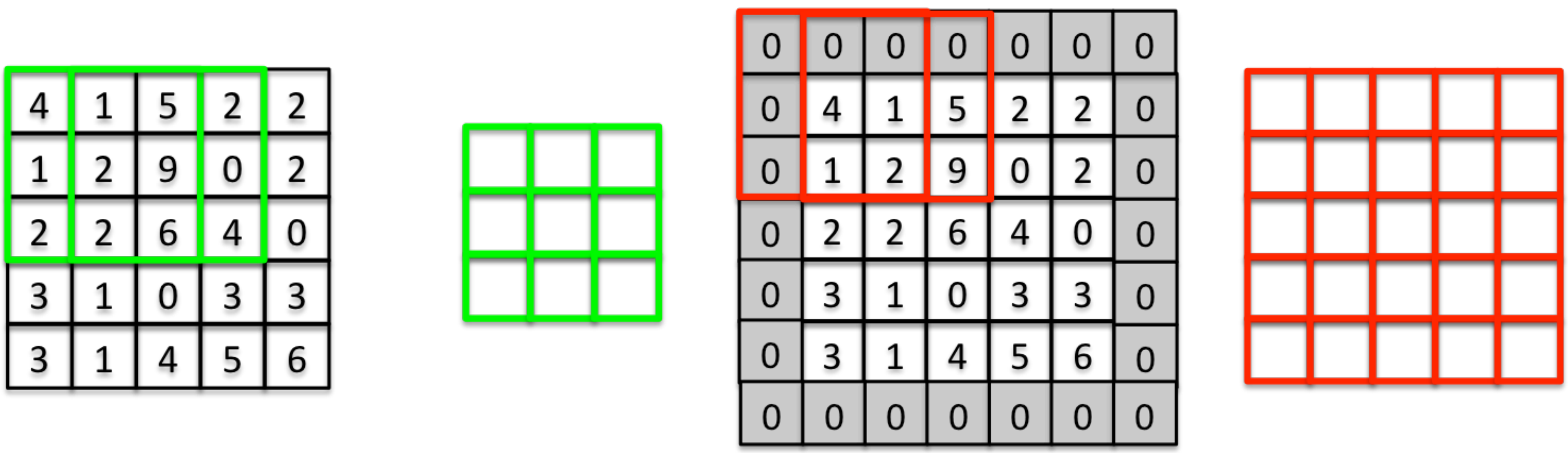
Kernel size of 3, stride 2

Zero-padding

4	1	5	2	2
1	2	9	0	2
2	2	6	4	0
3	1	0	3	3
3	1	4	5	6

Zero-padding

- Pad the input volume with zeros around the border so that the input and output width and height are the same



Conv layer input and output

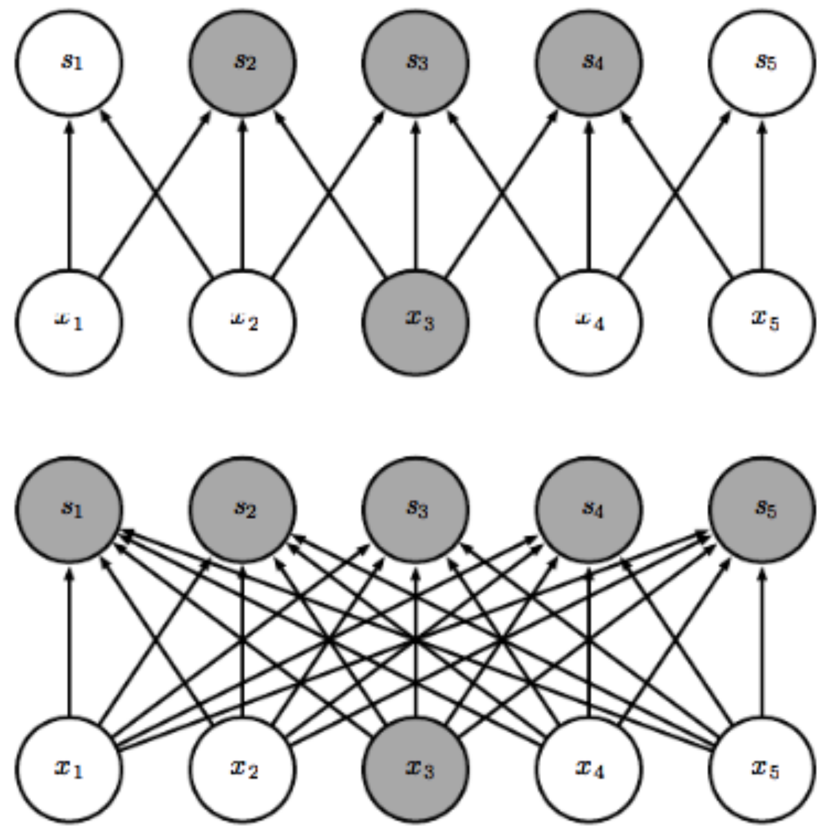
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyper-parameters:
  - Number of kernels K
  - Kernel size F
  - Stride S
  - The amount of zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- With parameter sharing, it introduces FFD_1 weights per filter, for a total of $(FFD_1)K$ weights and K biases.

What does convolution achieve?

- Convolution operation leverages three important ideas:
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
- Convolution provides a means for working with inputs of variable size

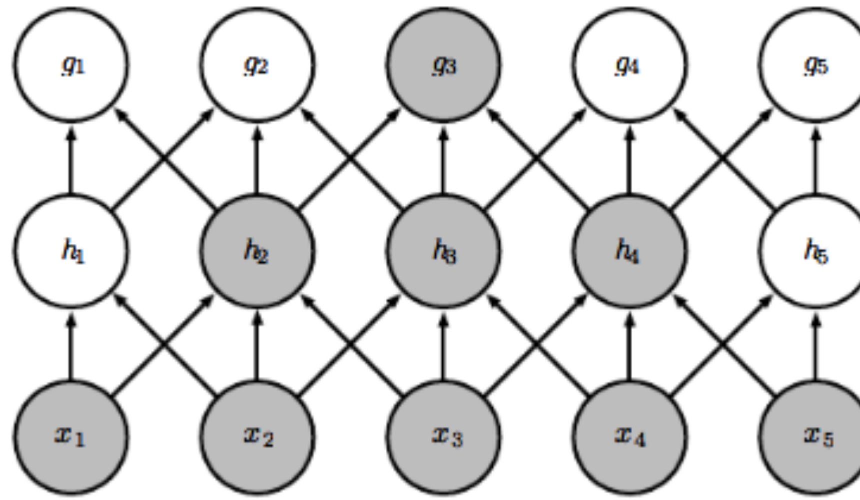
Sparse interactions

- Also referred to as **sparse connectivity** or **sparse weights**
- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit
- Making the kernel smaller than the input ➔ fewer parameters ➔ less memory requirement



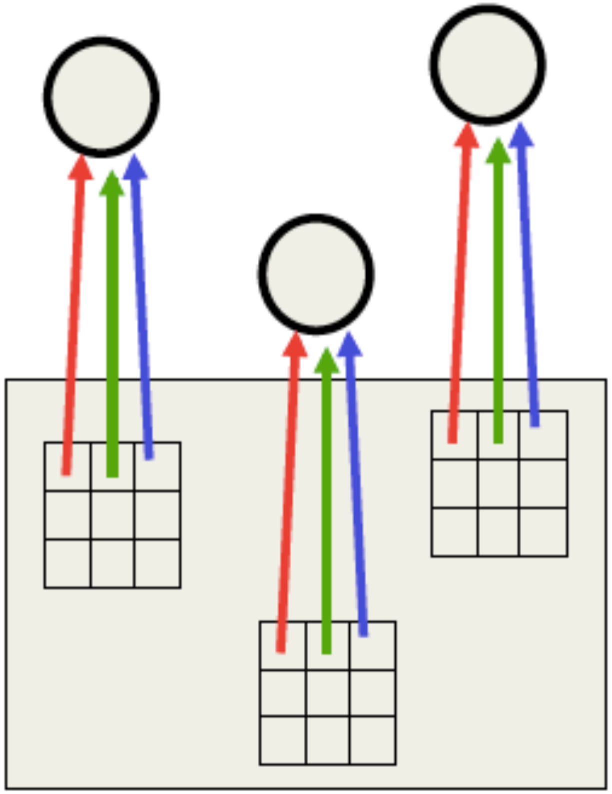
Sparse interaction

- In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input ➔ allow complicated interactions between many variables

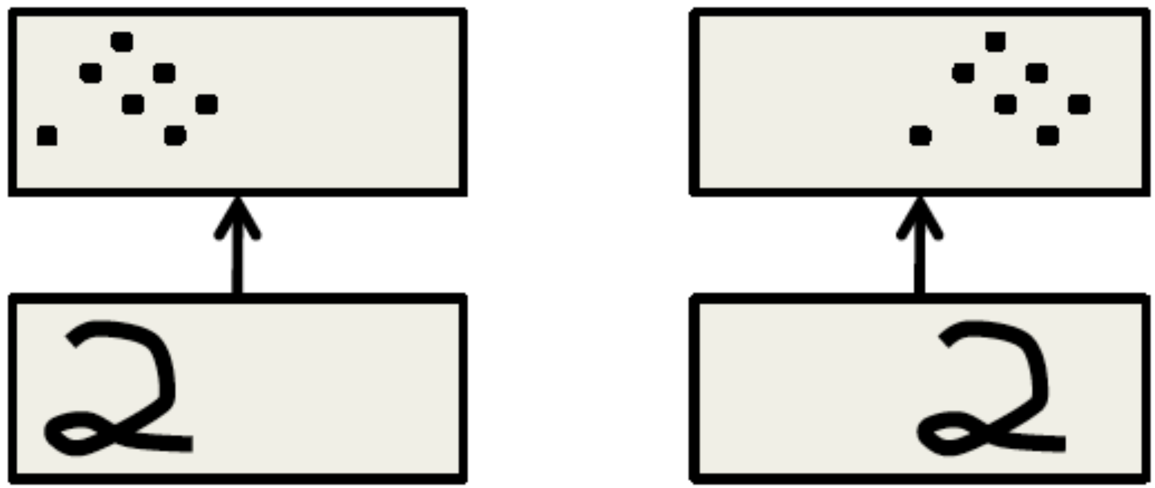


Parameter sharing

- Same parameter for more than one function in a model
- In a convolutional neural network, each member of the kernel is used at every position of the input
- Less computation and less parameters to learn



Equivariant representation



- Equivariant: If the input changes, the output changes in same way
- If we move the object in the input, its representation will move the same amount in the feature map
- Invariant knowledge: If a feature is useful in some locations during training, kernels for that feature will be available in all locations during testing.

Conv layer: Forward propagation

$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}}_{\text{Input}} * \underbrace{\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}}_{\text{Kernel}} = \underbrace{\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}}_{\text{Feature map}}$$

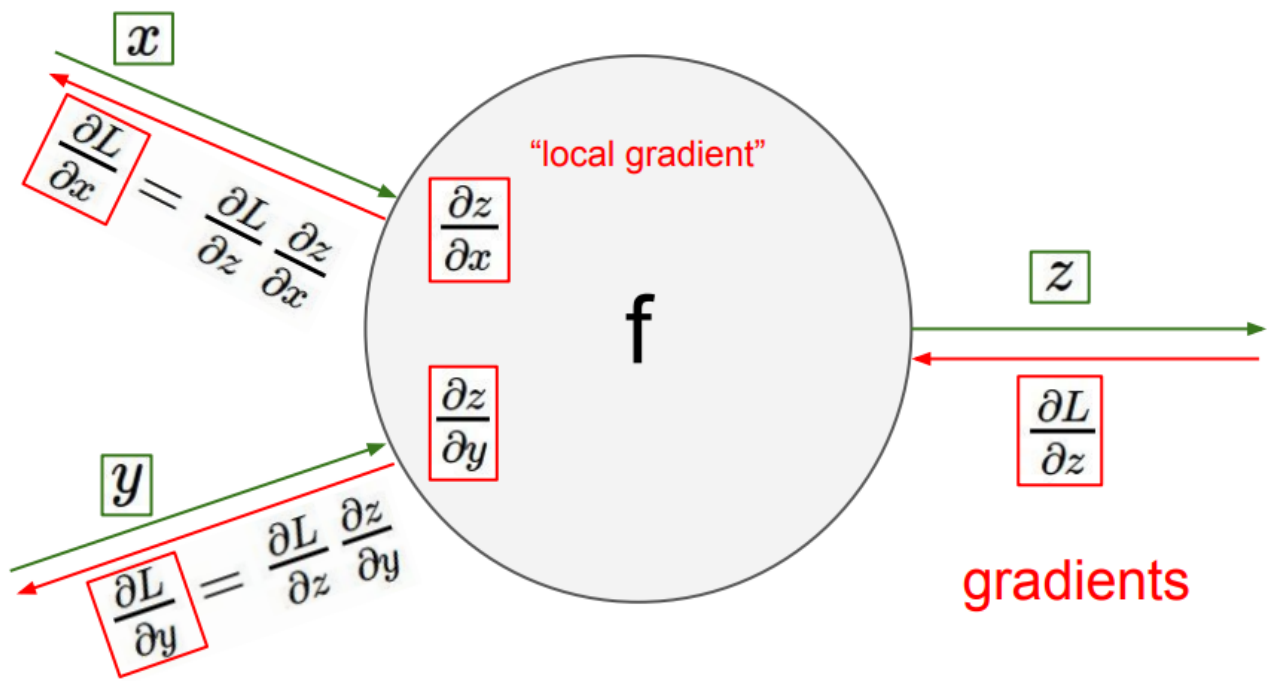
$$z_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$z_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$z_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$

$$z_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

Conv layer: Backward propagation



Conv layer: Backward propagation

- Apply chain rule to calculate the gradient of error L w.r.t kernel w:

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial z_{11}}\frac{\partial z_{11}}{\partial w_{11}} + \frac{\partial L}{\partial z_{12}}\frac{\partial z_{12}}{\partial w_{11}} + \frac{\partial L}{\partial z_{21}}\frac{\partial z_{21}}{\partial w_{11}} + \frac{\partial L}{\partial z_{22}}\frac{\partial z_{22}}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial z_{11}}\frac{\partial z_{11}}{\partial w_{12}} + \frac{\partial L}{\partial z_{12}}\frac{\partial z_{12}}{\partial w_{12}} + \frac{\partial L}{\partial z_{21}}\frac{\partial z_{21}}{\partial w_{12}} + \frac{\partial L}{\partial z_{22}}\frac{\partial z_{22}}{\partial w_{12}}$$

$$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial z_{11}}\frac{\partial z_{11}}{\partial w_{21}} + \frac{\partial L}{\partial z_{12}}\frac{\partial z_{12}}{\partial w_{21}} + \frac{\partial L}{\partial z_{21}}\frac{\partial z_{21}}{\partial w_{21}} + \frac{\partial L}{\partial z_{22}}\frac{\partial z_{22}}{\partial w_{21}}$$

$$\frac{\partial L}{\partial w_{22}} = \frac{\partial L}{\partial z_{11}}\frac{\partial z_{11}}{\partial w_{22}} + \frac{\partial L}{\partial z_{12}}\frac{\partial z_{12}}{\partial w_{22}} + \frac{\partial L}{\partial z_{21}}\frac{\partial z_{21}}{\partial w_{22}} + \frac{\partial L}{\partial z_{22}}\frac{\partial z_{22}}{\partial w_{22}}$$

Conv layer: Backward propagation

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial z_{11}} x_{11} + \frac{\partial L}{\partial z_{12}} x_{12} + \frac{\partial L}{\partial z_{21}} x_{21} + \frac{\partial L}{\partial z_{22}} x_{22}$$

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial z_{11}} x_{12} + \frac{\partial L}{\partial z_{12}} x_{13} + \frac{\partial L}{\partial z_{21}} x_{22} + \frac{\partial L}{\partial z_{22}} x_{23}$$

$$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial z_{11}} x_{21} + \frac{\partial L}{\partial z_{12}} x_{22} + \frac{\partial L}{\partial z_{21}} x_{31} + \frac{\partial L}{\partial z_{22}} x_{32}$$

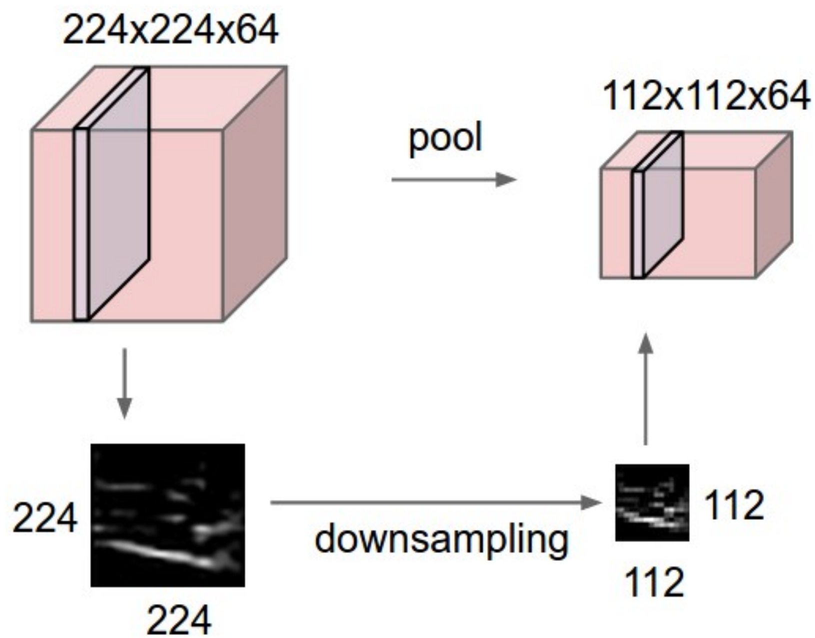
$$\frac{\partial L}{\partial w_{22}} = \frac{\partial L}{\partial z_{11}} x_{22} + \frac{\partial L}{\partial z_{12}} x_{23} + \frac{\partial L}{\partial z_{21}} x_{32} + \frac{\partial L}{\partial z_{22}} x_{33}$$

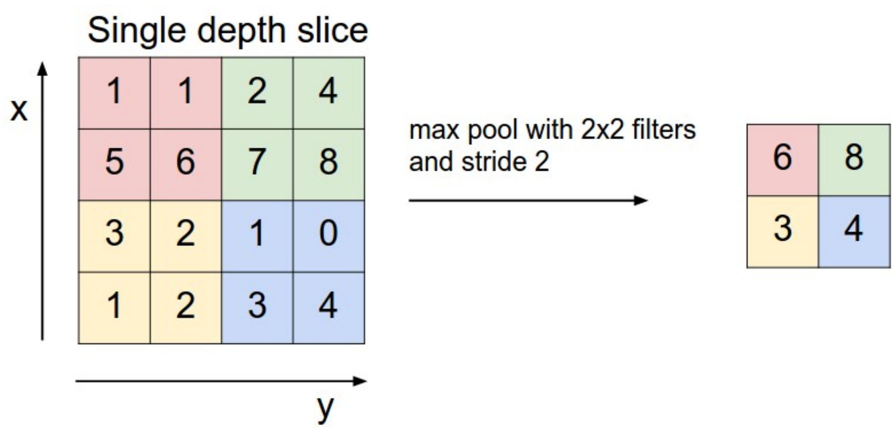
$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}}_{\text{Input}} * \begin{bmatrix} \frac{\partial L}{\partial z_{11}} & \frac{\partial L}{\partial z_{12}} \\ \frac{\partial L}{\partial z_{21}} & \frac{\partial L}{\partial z_{22}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} \end{bmatrix}$$

Pool layer

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs
- Functions include average, weighted average, L2 norm, maximum, . . .
- In all cases, pooling helps to make the representation approximately invariant to small translations of the input

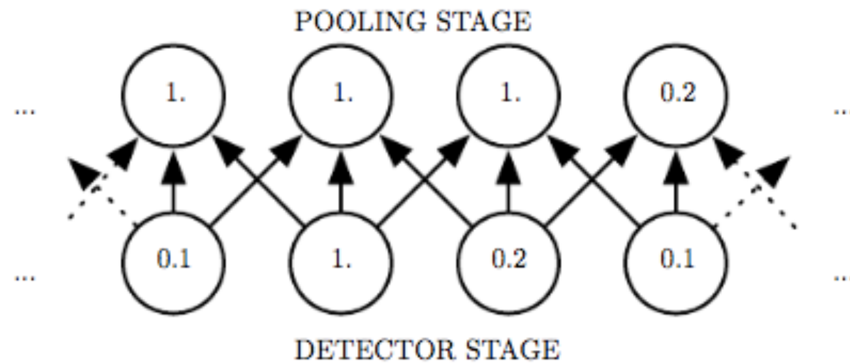
Max pooling: Forward propagation

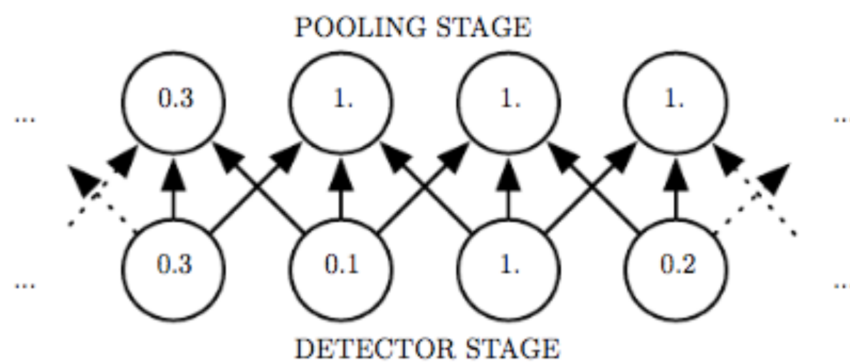




Max pooling

- The max pooling operation, which reports the maximum output within a rectangular neighborhood (usually 2x2), is preferred because it worked slightly better
- Pros:
  - This reduces the number of inputs to the next layer of feature extraction, thus allowing us to have many more different feature maps
  - Invariant to small translations of the input
- Cons: after several levels of pooling, we have lost information about the precise positions of things



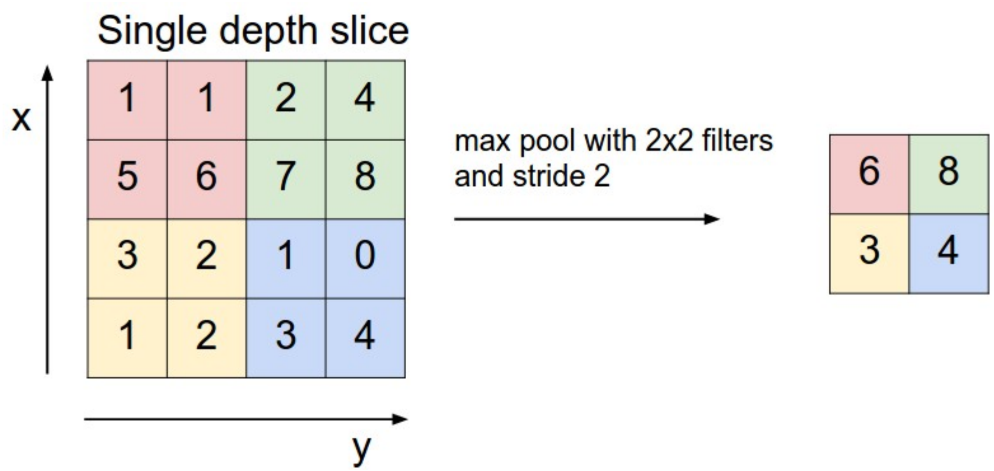


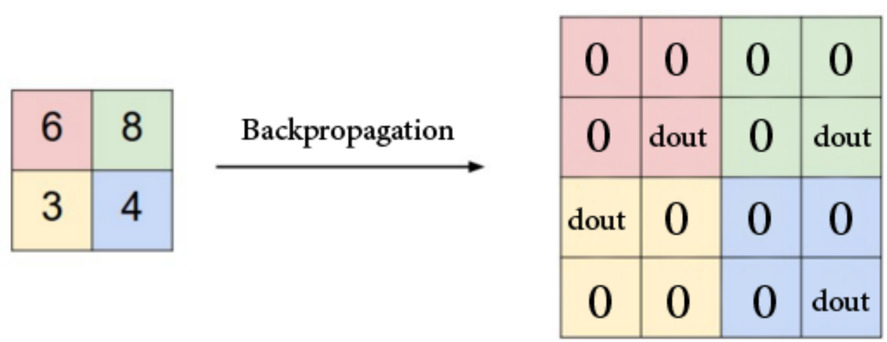
Max pooling: Input and output

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyper-parameters:
  - Spatial extend F
  - Stride S
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduce zero parameter since it computes a fixed function of the input

Max pooling: Backward propagation

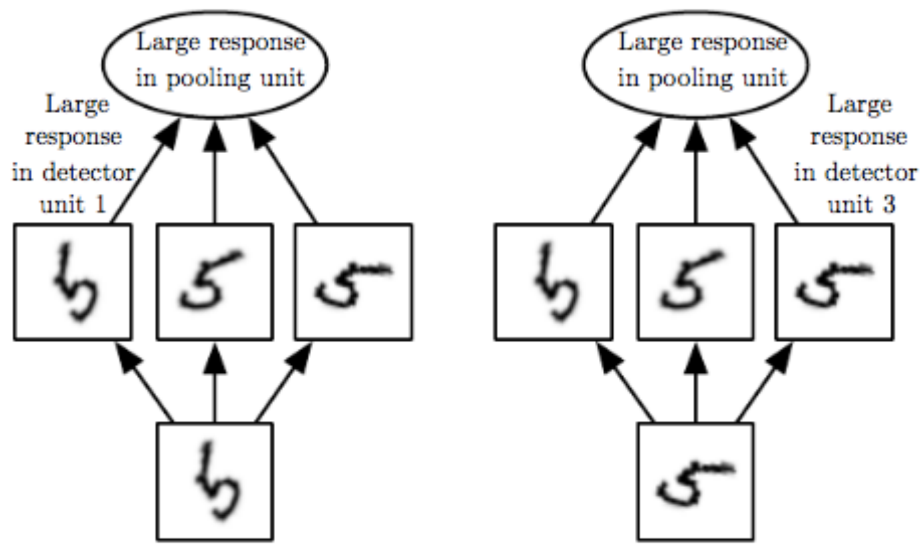
- Route the gradient to the input that has the highest value in the forward pass





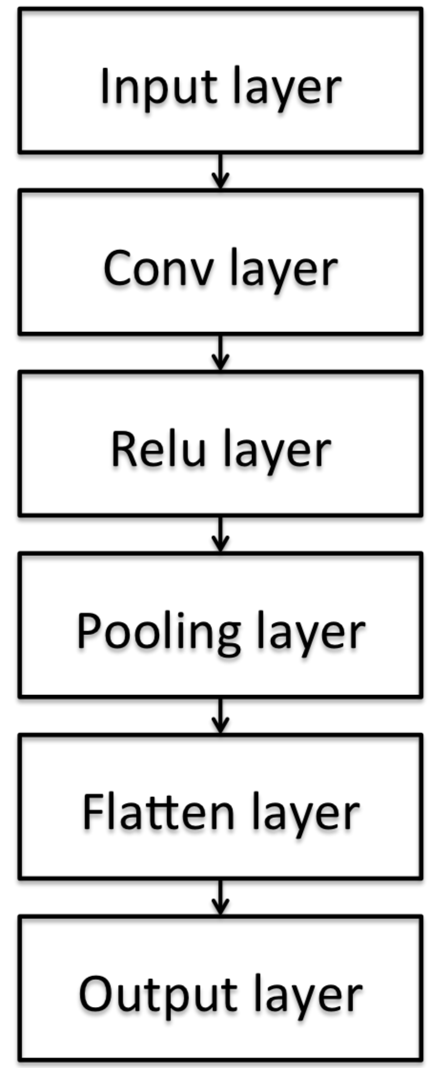
Transformation invariant

- If we pool over the outputs of separately parameterized convolutions, the features can learn which transformations to become invariant

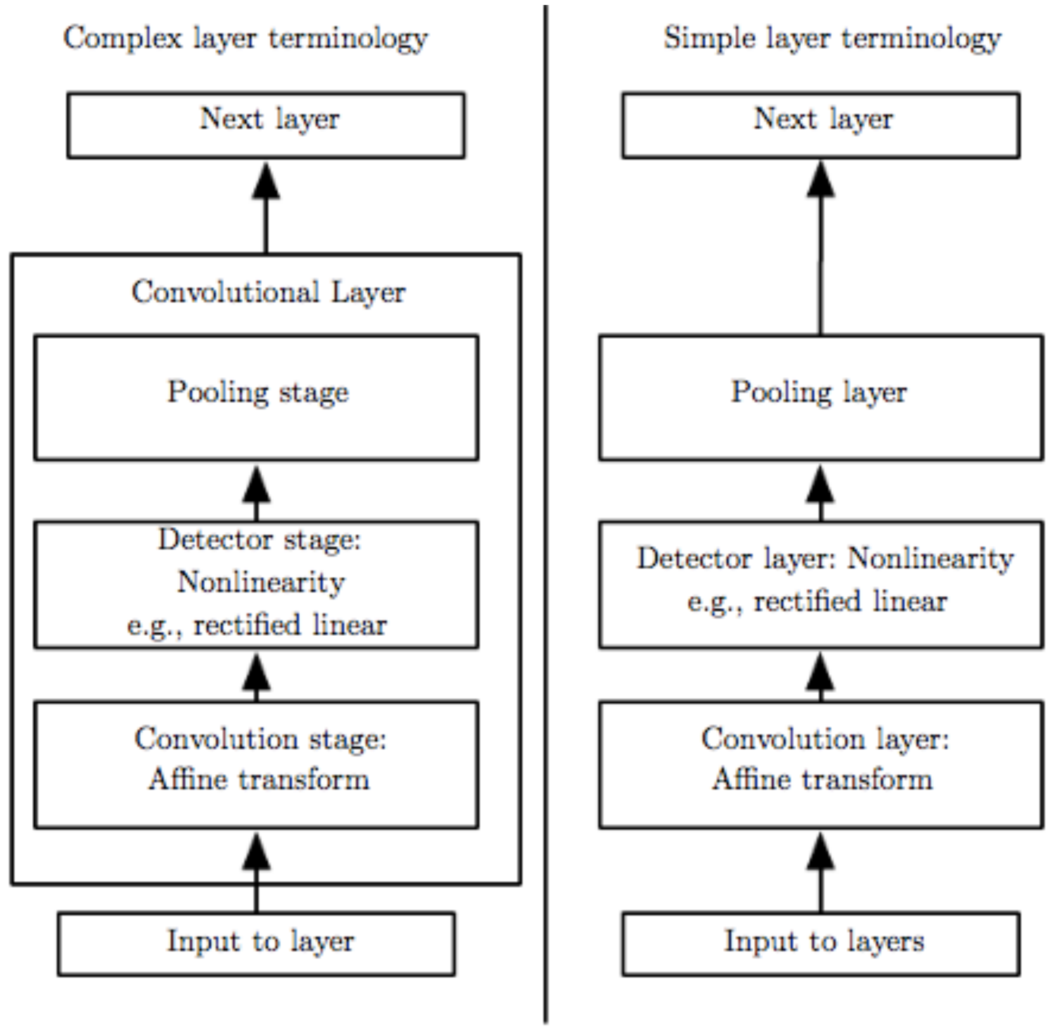


A simple CNN revisited

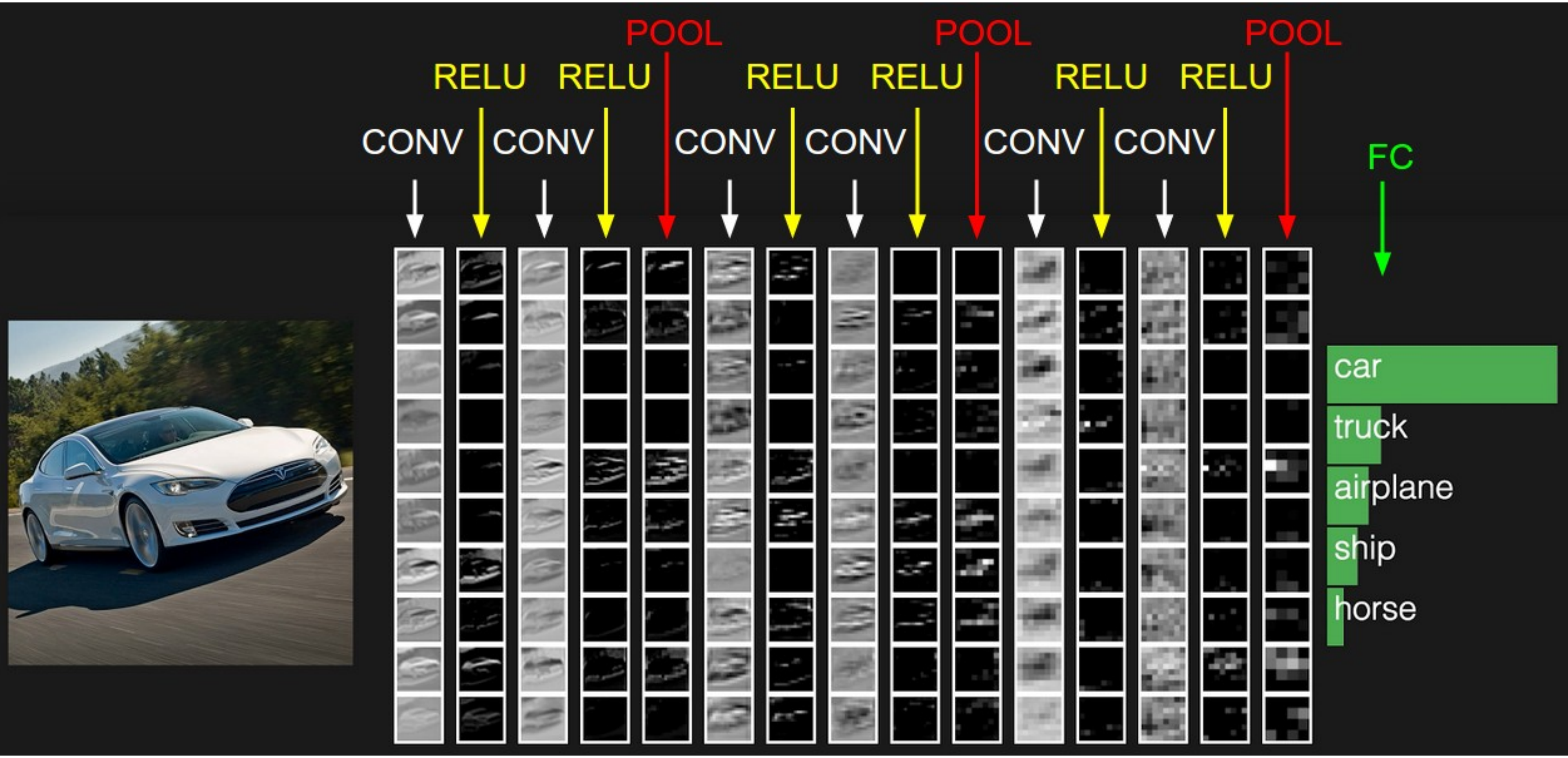
- There are a few distinct types of Layers
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters?
- Each Layer may or may not have additional hyper-parameters?



Two different terminology for CNN



Deep CNN: example



Summary

- CNNs is a specialized kind of neural network to process data a grid-like topology
- A typical convolutional layer includes convolution, ReLU, and max pooling operations
- Convolution operation has three important features, including sparse interactions, parameter sharing, and equivariant representations
- Max pooling reduces the number of inputs to the next layer and invariant to small translations of the input