

# Lecture 7

Deep learning  
Section 9.2, 9.3

1

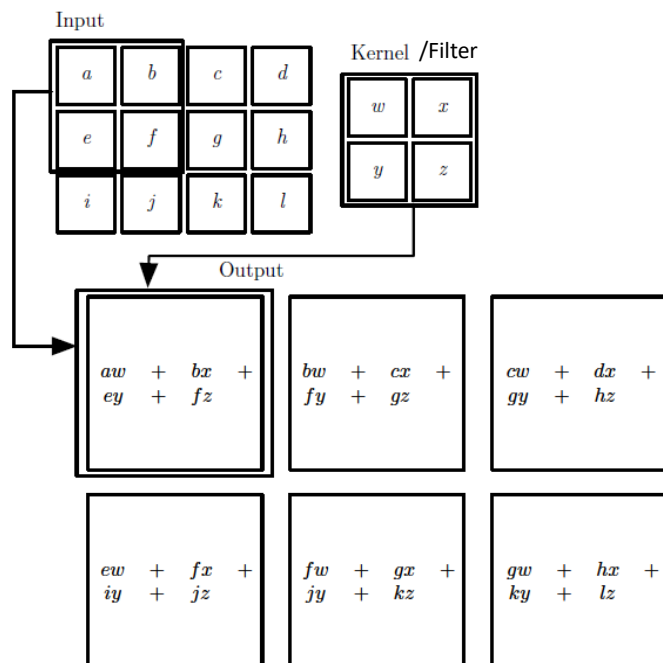
## Convolutional Networks

- Specialized kind of neural network for processing data that has a known, grid-like topology(e.g. image data)
- *Convolutional networks use **convolution** in place of general matrix multiplication in at least one of their layers.*
- In this section
  - What is convolution?
  - What is the motivation behind using convolution in a neural network?
  - What is pooling?
  - How to make convolution more efficient?

Goodfellow, Bengio, Courville 2016

2

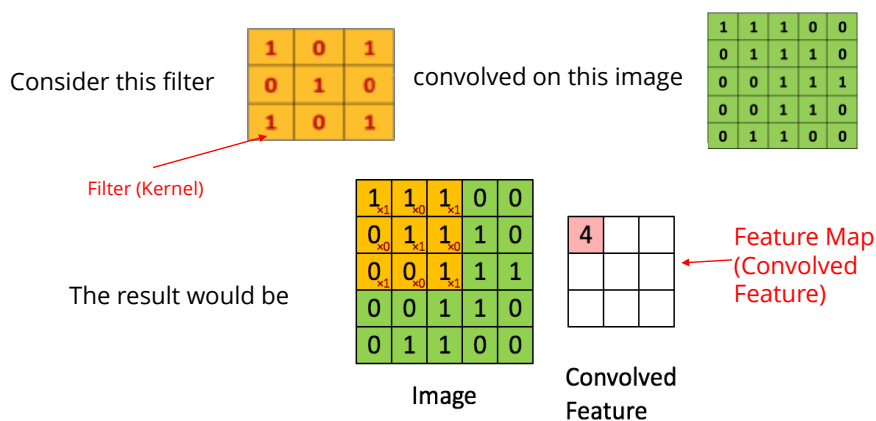
## 2D convolution



3

## Convolution

We slide the orange matrix over our original image (green) by 1 pixel (also called 'stride') and for every position



[http://deeplearning.stanford.edu/wiki/index.php/Feature\\_extraction\\_using\\_convolution/](http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution/)

4

## Motivation

- Convolution leverages three important ideas that can help improve a machine learning system:
  1. Sparse interactions
  2. Parameter sharing
  3. Equivariant representations.

Goodfellow, Bengio, Courville 2016

5

## Sparse Interactions

- 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.
- This means every output unit interacts with every input unit.
- Convolutional networks, however, typically have **sparse interactions** (also referred to as **sparse connectivity** or **sparse weights**).
- This is accomplished by [making the kernel smaller than the input](#).
- For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels.

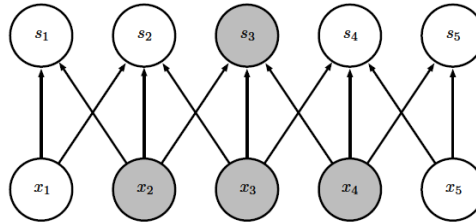
Goodfellow, Bengio, Courville 2016

6

# Sparse Connectivity

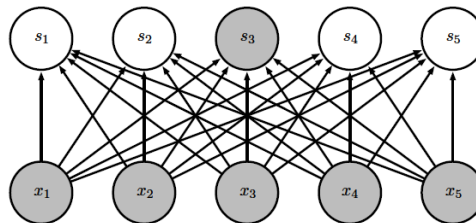
$s$  is formed by convolution with a kernel of width 3 (only three inputs affect  $s_3$ )

Sparse connections  
due to small convolution kernel



$s$  is formed by matrix multiplication

Dense connections

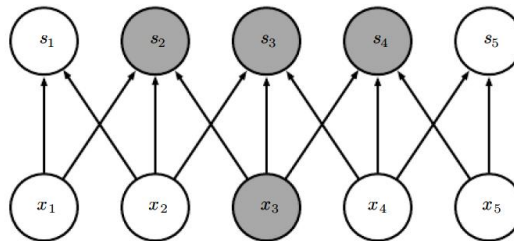


Goodfellow, Bengio, Courville 2016

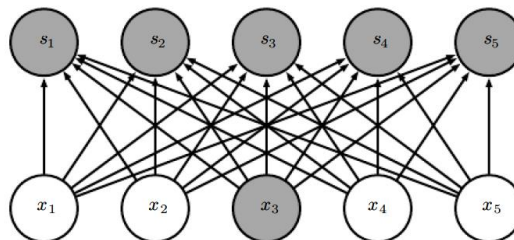
7

# Sparse Connectivity

$s$  is formed by convolution with a kernel of width 3 (only three outputs are affected by  $x_3$ )



$s$  is formed by matrix multiplication (all of the output is affected by  $x_3$ )



Goodfellow, Bengio, Courville 2016

8

## Why sparse connectivity?

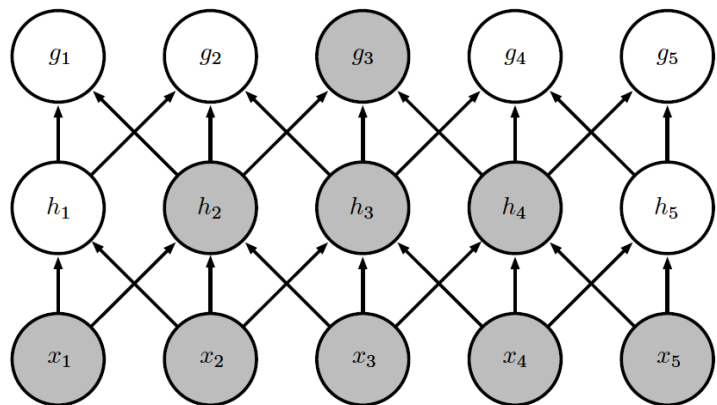
- If there are  $m$  inputs and  $n$  outputs, then matrix multiplication requires  $m \times n$  parameters and the algorithms used in practice have  $O(m \times n)$  runtime (per example).
- If we limit the number of connections each output may have to  $k$  (*several orders of magnitude less than  $m$ .*), then the sparsely connected approach requires only  $k \times n$  parameters and  $O(k \times n)$  runtime.
- Sparse connectivity:
  - Requires fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency.
  - Computing the output requires fewer operations.

Goodfellow, Bengio, Courville 2016

9

## Growing Receptive Fields

- Deeper layers may *indirectly* interact with a larger portion of the input.
- Network can efficiently describe complicated interactions between many variables using simpler building blocks that each describe only sparse interactions.



Goodfellow, Bengio, Courville 2016

10

## Parameter sharing

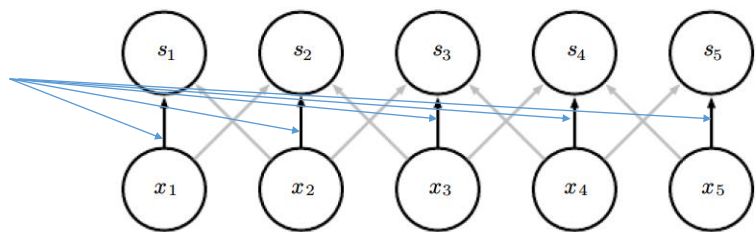
- Refers to the use of the same parameter for more than one function in a model.
- In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer (never revisited for other inputs).
- In a convolutional neural net, each member of the kernel is used at every position of the input.
- Rather than learning a separate set of parameters for every location, we learn only one set (a kernel) for all inputs.
- When the value of the weight applied to one input is tied to the value of a weight applied to another input, the network is said to have **tied weights**.
- This does not affect the runtime of forward propagation (time complexity—it is still  $O(k \times n)$ )—but it does further reduce the storage requirements (space complexity) of the model to  $k$  parameters.

Goodfellow, Bengio, Courville 2016

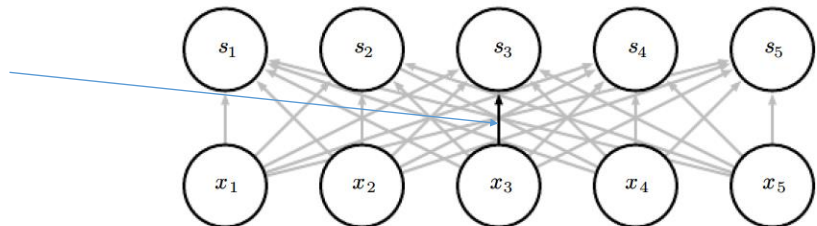
11

## Parameter sharing

Same central element of a 3-element kernel is used at all input locations (parameter sharing)



Model has no parameter sharing. Each parameter is used only once (no parameter sharing)



Goodfellow, Bengio, Courville 2016

12

## Why Parameter sharing?

- Parameter sharing does not affect the runtime of forward propagation (time complexity—it is still  $O(k \times n)$ —but it does further reduce the storage requirements (space complexity) of the model to  $k$  parameters (compare to  $O(m \times n)$ ).
- Since  $m$  and  $n$  are usually roughly the same size,  $k$  is practically insignificant compared to  $m \times n$ .
- Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

Goodfellow, Bengio, Courville 2016

13

## Equivariance

- To say a function is **equivariant** means that if the input changes, the output changes in the same way.
- Specifically, a function  $f(x)$  is equivariant to a function  $g$  if  $f(g(x)) = g(f(x))$ .
- If we move an object in the image, its representation will move the same amount in the output.
- let  $I$  be a function giving image brightness at integer coordinates and  $g$  be a function that maps one image function to another image function such that  $I' = g(I)$  is the image function with  $I'(x, y) = I(x - 1, y)$ .
- If we apply this transformation to  $I$ , then apply convolution, the result will be the same as if we applied convolution to  $I'$ , then applied the transformation  $g$  to the output.

Goodfellow, Bengio, Courville 2016

14



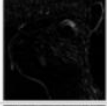

# Equivariance




- In the case of convolution, the particular form of parameter sharing causes the layer to have a property called **equivariance to translation**.
- If we move the object in the input, its representation will move the same amount in the output.
- This is useful if we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations.
- An example is edge detection. it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.
- Convolution is **not** naturally equivariant to some other transformations, such as **changes in the scale** or **rotation** of an image. Other mechanisms are necessary for handling these kinds of transformations.

Goodfellow, Bengio, Courville 2016

15

# Examples of different filter effects

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

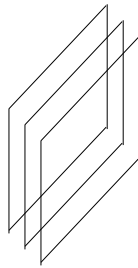
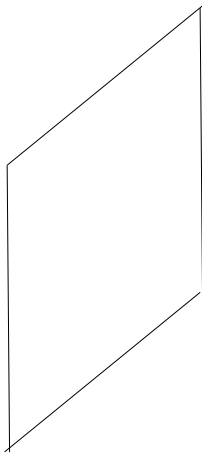
Operation	Filter	Convolved Image
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\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

[https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

16



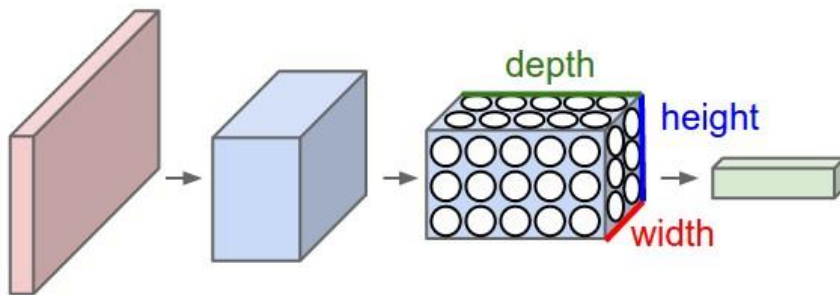
## Multiple Filters



Using 3 filters,  
produces 3 feature  
maps

17

## 3 Dimensions of neurons

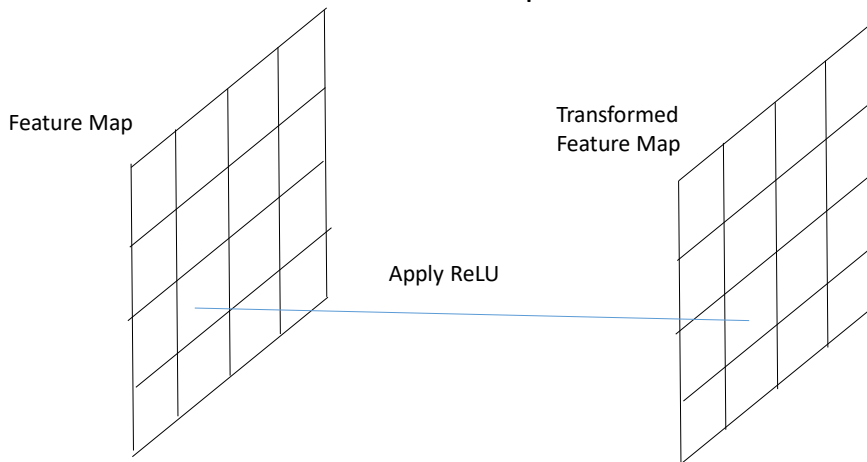


<http://cs231n.github.io/convolutional-networks/>

18

## Non-Linearity

- After obtaining a feature map we apply element-wise non-linear transformation to obtain a transformed feature map



19

## Convolutional Layer Parameters

- The Convolutional layer's parameters consist of a set of learnable filters
- We connect each neuron to only a local region of the input volume (**receptive field/ filter size**).
- Suppose that the input has a size  $[32 \times 32 \times 3]$  (an RGB CIFAR-10 image) and we have 3  $5 \times 5$  filters stride 1
- How many parameters?
- $5 \times 5 \times 3 + 1(\text{bias}) = 76$

20

## Output Size

**N : Input Size**

**K: Kernel Size**

**S: stride**

$$\text{Output Size} = \frac{N-K}{S} + 1$$

**In the previous example, N=5, K =3, S =1 --- output size (2+1=3)**

21

## Zero Padding

- In many situations, you will require the output of the convolution to have the same size as the input. This can be achieved using the zero padding

0	0	0	0	0
0				0
0				0
0				0
0	0	0	0	0

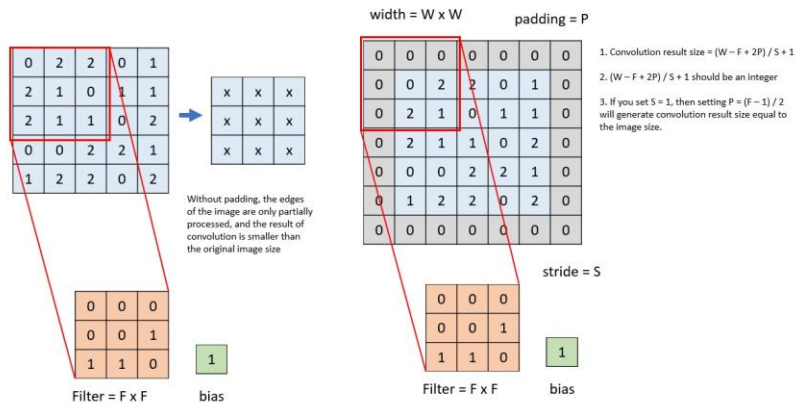
You may use a padding pad of

$$\frac{K-1}{2}$$

If you would like an output of the same size as the input

22

# Zero Padding



<https://jamesmccaffrey.wordpress.com/2018/05/30/convolution-image-size-filter-size-padding-and-stride/>

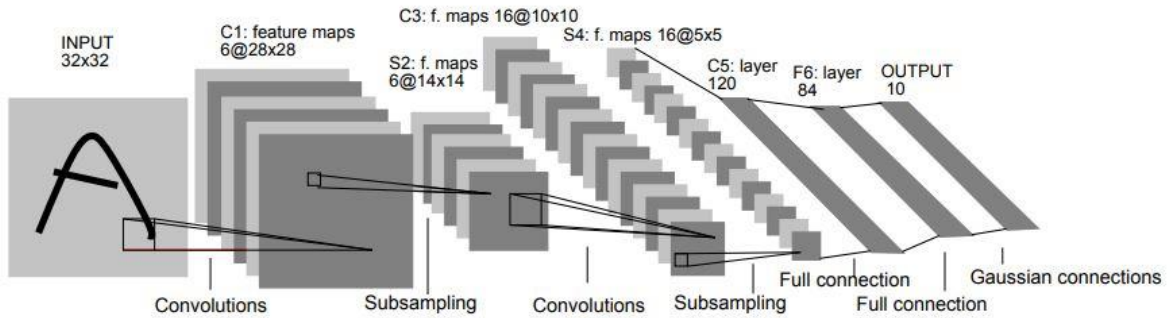
23

# Hyper-parameters

- **Depth:** number of filters we would like to use
- **Stride:** how we slide the filter (typically 1 or 2)
- Size of **zero-padding:**

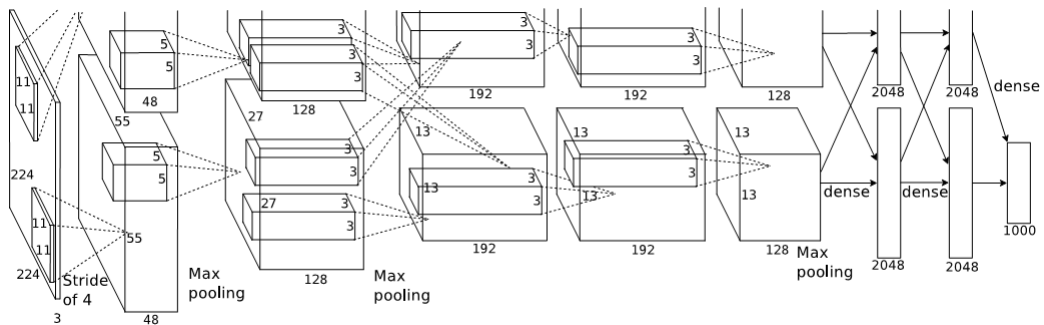
24

## LeNet-5 (LeCun et al., 1998)



25

## AlexNet (Alex Krizhevsky, Ilya Sutskever, Hinton 2012))



26

## Variants of the Basic Convolution Function

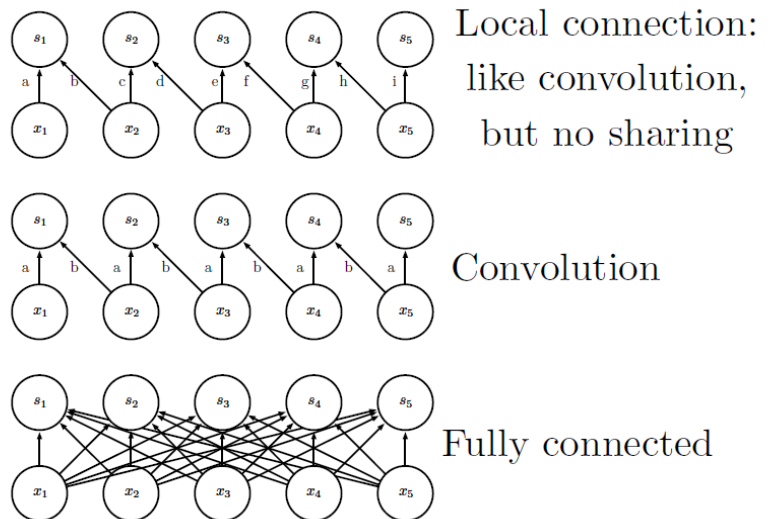
- Locally Connected Layers (unshared convolution)
- Tiled Convolution

Goodfellow, Bengio, Courville 2016

27

## Locally Connected Layers (unshared convolution)

- Similar to discrete convolution but without sharing parameters across locations.
- Locally connected layers are useful when we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space.
- For example, if we want to tell if an image is a picture of a face, we only need to look for the mouth in the bottom half of the image.

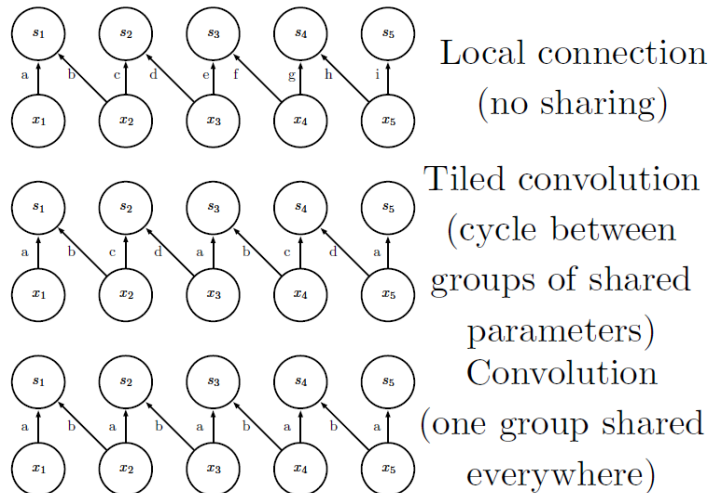


Goodfellow, Bengio, Courville 2016

28

## Tiled convolution

- offers a compromise between a convolutional layer and a locally connected layer.
- Rather than learning a separate set of weights at *every* spatial location, we learn a set of kernels that we rotate through as we move through space.
- This means that immediately neighboring locations will have different filters, like in a locally connected layer, but the memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels, rather than the size of the entire output feature map.



Goodfellow, Bengio, Courville 2016

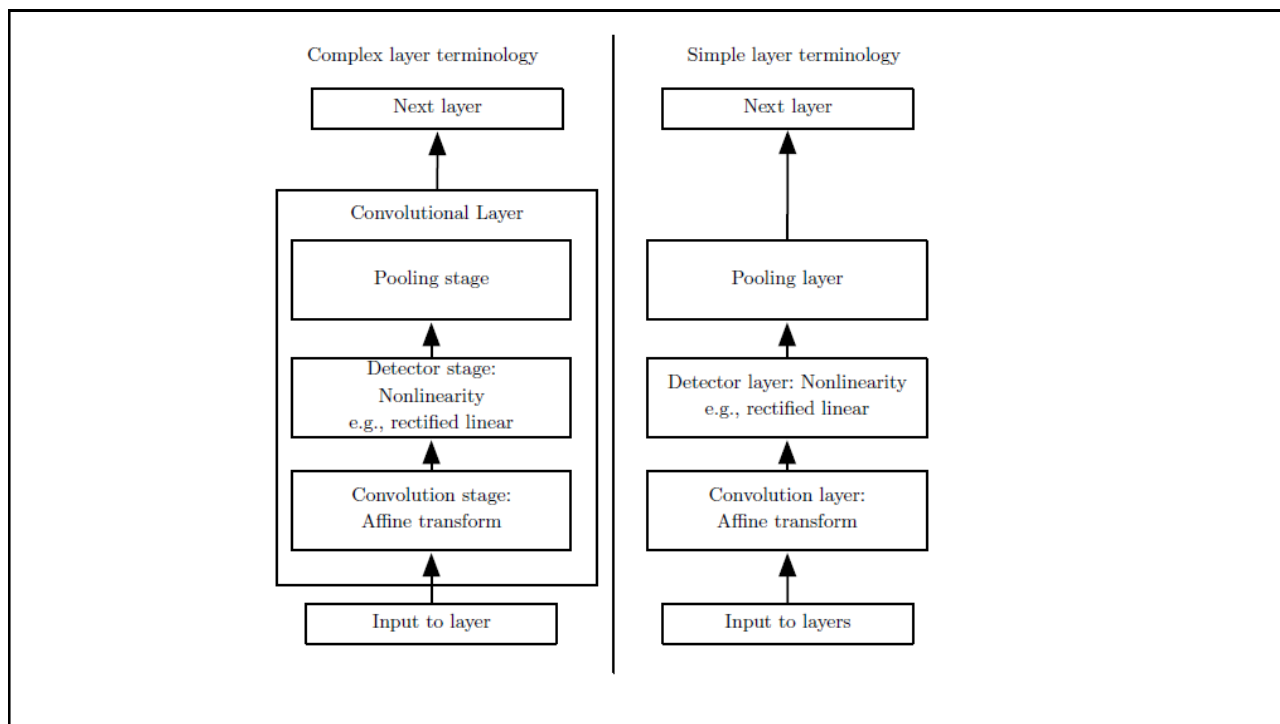
29

## Pooling

- A typical layer of a convolutional network consists of three stages:
  1. Performs several convolutions in parallel to produce a set of linear activations.
  2. Each linear activation is run through a nonlinear activation function, such as ReLU
  3. A pooling function to modify the output of the layer.
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- For example, the **max pooling** (Zhou and Chellappa, 1988) operation reports the *maximum output* within a rectangular neighborhood
- Other popular pooling functions include the *average* of a rectangular neighborhood, the  *$L_2$  norm* of a rectangular neighborhood, or a *weighted average* based on the distance from the central pixel.

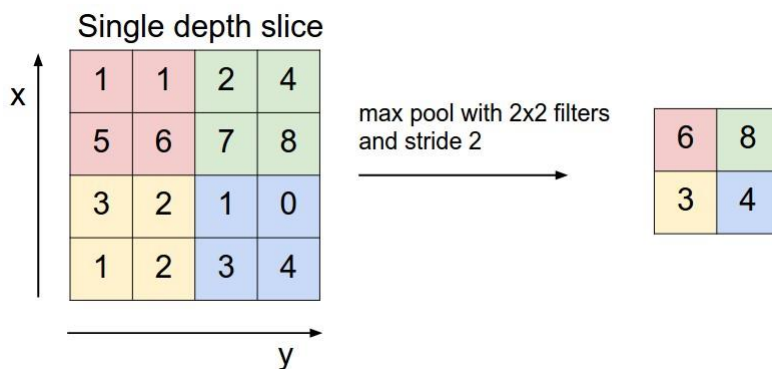
Goodfellow, Bengio, Courville 2016

30



31

## Example - Pooling



<http://cs231n.github.io/convolutional-networks/>

32



## Why Pooling? Invariant to translation

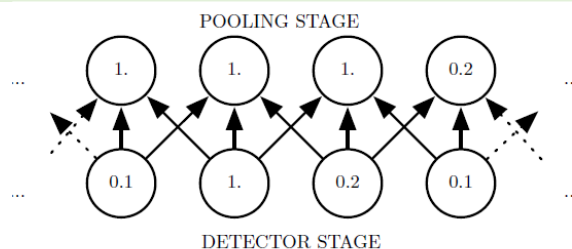
- Pooling helps to make the representation become **approximately invariant** to small translations of the input (if we translate the input by a small amount, the values of most of the pooled outputs do not change.).
- Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.
- For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face.

Goodfellow, Bengio, Courville 2016

33

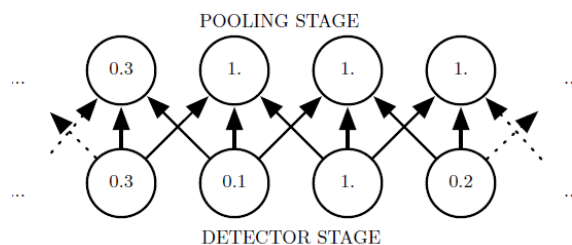
## Max pooling introduces invariance

The outputs of max pooling, with a stride of one and a pooling width of three.



The same network, after the input has been shifted to the right by one pixel.

Every value in the bottom row has changed, but only half of the values in the top row have changed.



Goodfellow, Bengio, Courville 2016

34

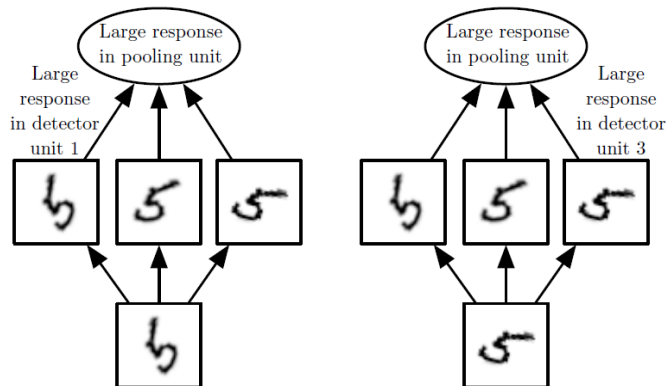
## Learned invariances

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

In this example, we use 3 filters to detect a slightly different orientation of the 5.

When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit.

The max pooling unit then has a large activation regardless of which detector unit was activated.



Goodfellow, Bengio, Courville 2016

35

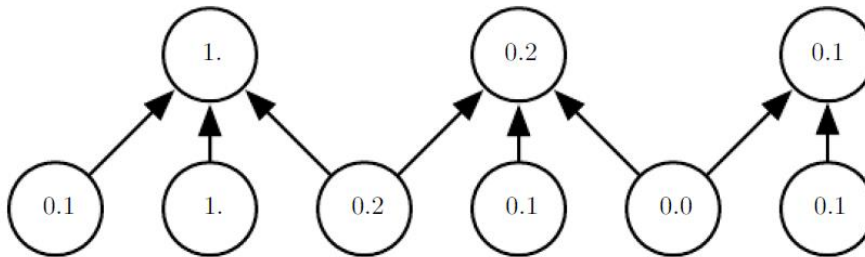
## Why Pooling? Down sampling

- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced  $k$  pixels apart rather than 1 pixel apart.
- This improves the computational efficiency of the network because the next layer has roughly  $k$  times fewer inputs to process.
- Reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.

Goodfellow, Bengio, Courville 2016

36

## Why Pooling? Down sampling



Max-pooling with a pool width of three and a stride between pools of two. This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer.

Goodfellow, Bengio, Courville 2016

37

## Problem

Consider a CNN that has the following layers:

- Input layer: one channel of size 32x32
  - Convolutional layer: 6 filters with size 5x5 each. Padding is 2  $((5-1)/2)$  and stride is 1
  - Pooling layer: a 2x2 max-pooling layer with stride 2 and padding 0
  - Fully connected layer: with N output neurons
- a. Compute the dimensions and number of the parameters at both the convolutional and pooling layers.
  - b. Determine the number of weights in the fully-connected layer.

Goodfellow, Bengio, Courville 2016

38

## Solution - Convolutional Layer

Size after padding =  $32 + 2 * 2 = 36$  (padding adds 4 rows and 4 columns to the input)

*Size after convlution layer* =  $\left(\frac{36-K}{s} + 1\right) = 32$  (same as input)

Dimension =  $32 * 32 * 6$

Number of parameters =  $6 * (5 * 5 + 1 \text{ (bias)}) = 156$

Goodfellow, Bengio, Courville 2016

39

## Solution - Pooling Layer

Size after pooling =  $32/2 = 16$

Dimension =  $16 * 16 * 6$

Number of parameters = 0 (no parameters in the pooling layer)

Goodfellow, Bengio, Courville 2016

40

## Solution - Fully Connected

Number of weights =  $(16*16*6+1(\text{bias}))*N$

Goodfellow, Bengio, Courville 2016