Convolutional Networks: Motivation

Sargur Srihari srihari@buffalo.edu

Topics in Convolutional Networks

- Overview
- 1. The Convolution Operation
- 2. Motivation
- 3. Pooling
- 4. Convolution and Pooling as an Infinitely Strong Prior
- 5. Variants of the Basic Convolution Function
- 6. Structured Outputs
- 7. Data Types
- 8. Efficient Convolution Algorithms
- 9. Random or Unsupervised Features
- 10. The Neuroscientific Basis for Convolutional Networks
- 11. Convolutional Networks and the History of Deep Learning

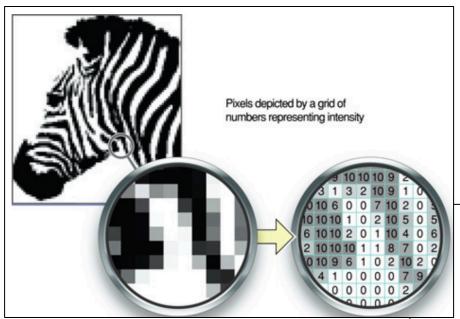
Overview of Convolutional Networks

- Convolutional networks, also known as Convolutional neural networks (CNNs) are a specialized kind of neural network
- It is for processing data that has a known grid-like topology
 - Ex: time-series data, which is a 1-D grid, taking samples at intervals
 - Image data, which are 2-D grid of pixels
- They utilize convolution, which is a specialized kind of linear operation
- The convolution operation is used in place of general matrix multiplication in at least one layer

Motivation for using convolution networks

- 1. Convolution leverages three important ideas to improve ML systems:
 - 1. Sparse interactions
 - 2. Parameter sharing
 - 3. Equivariant representations
- 2. Convolution also allows for working with inputs of variable size

Sparse connectivity due to Image Convolution



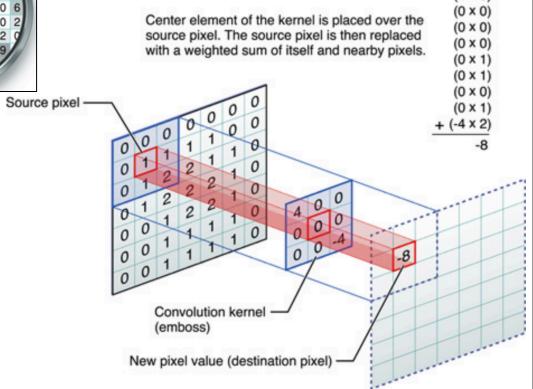
- Input image may have millions of pixels,
- But we can detect edges with kernels of hundreds of pixels
- If we limit no of connections for each input to k
 - we need kxn parameters and $O(k \times n)$ runtime

 (4×0)

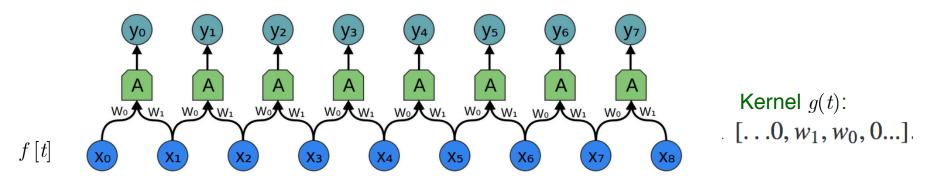
 It is possible to get good performance with k<<n

 Convolutional networks have sparse interactions

 Accomplished by making the kernel smaller than the input Next slide shows graphical depiction



Neural network for 1-D convolution



Equations for outputs of this network:

$$y_0 = \sigma(W_0 x_0 + W_1 x_1 - b)$$
 $y_1 = \sigma(W_0 x_1 + W_1 x_2 - b)$ etc. upto y

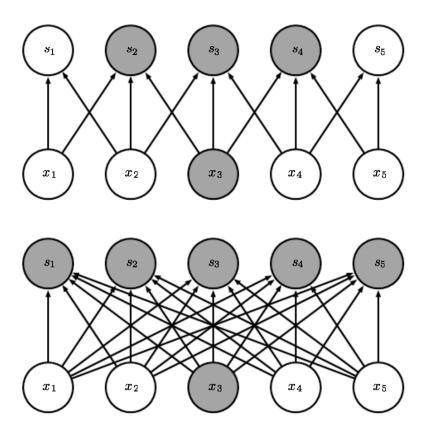
We can also write the equations in terms of elements of a general 8×8 weight matrix W as:

$$y_0 = \sigma(W_{0,0}x_0 + W_{0,1}x_1 + W_{0,2}x_2...)$$

$$y_1 = \sigma(W_{1,0}x_0 + W_{1,1}x_1 + W_{1,2}x_2...)$$
where $W = \begin{bmatrix} w_0 & w_1 & 0 & 0 & ... \\ 0 & w_0 & w_1 & 0 & ... \\ 0 & 0 & w_0 & w_1 & ... \\ 0 & 0 & 0 & w_0 & ... \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & w_0 & ... \end{bmatrix}$

Sparse Connectivity, viewed from below

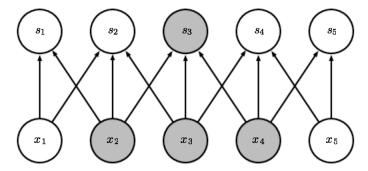
- Highlight one input x_3 and output units s affected by it
- Top: when s is formed by convolution with a kernel of width 3, only three outputs are affected by x₃
- Bottom: when s is formed by matrix multiplication connectivity is no longer sparse
 - So all outputs are affected by x₃



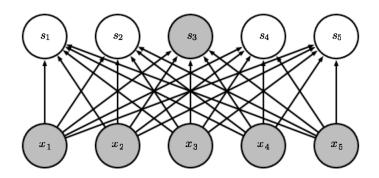
Sparse Connectivity, viewed from above

- Highlight one output s_3 and inputs x that affect this unit
 - These units are known as the receptive field of s_3

When s_3 is formed by convolution with a kernel of width 3

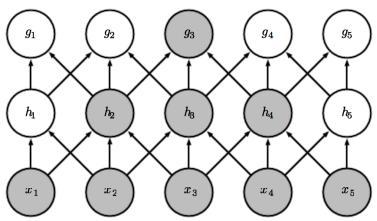


When s_3 is formed by matrix multiplication



Keeping up performance with reduced connections

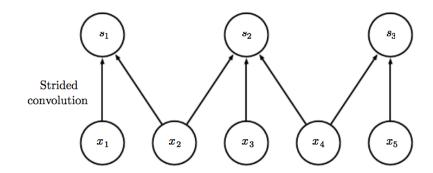
- It is possible to obtain good performance while keeping k several magnitudes lower than m
- In a deep neural network, units in deeper layers may indirectly interact with a larger portion of the input
 - Receptive Field in Deeper layers is larger than the receptive field of units in shallow layers

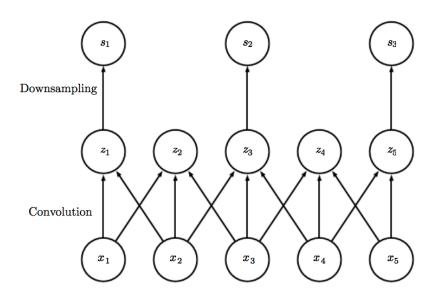


 This allows the network to efficiently describe complicated interactions between many variables from simple building blocks that only describe sparse interactions

Convolution with a stride

- Receptive Field in Deeper layers is larger than the receptive field of units in shallow layers
- This effect increases if the network includes architectural features like strided convolution or pooling





Parameter Sharing

- Parameter sharing refers to using 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
 - It is multiplied by one element of the input and never revisited
- Parameter sharing is synonymous with tied weights
 - Value of the weight applied to one input is tied to a weight applied elsewhere
- In a Convolutional net, each member of the kernel is used in every position of the input (except at the boundary subject to design decisions)

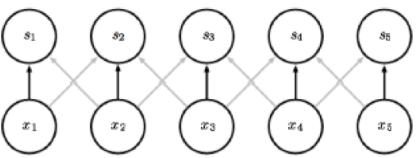
Efficiency of Parameter Sharing

- Parameter sharing by convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set
- This does not affect runtime of forward propagation—which is still $O(k \times n)$
- But further reduces the storage requirements to k parameters
 - *k* is orders of magnitude less than *m*
 - Since m and n are roughly the same size k is much smaller than $m \times n$

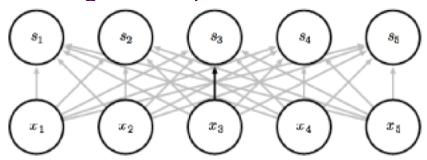
How parameter sharing works

- Black arrrows: connections that use a particular parameter
 - 1. Convolutional model: Black arrows indicate uses of the central element

of a 3-element kernel



- 2. Fully connected model: Single black arrow indicates use of the central element of the weight matrix
 - Model has no parameter sharing, so the parameter is used only once



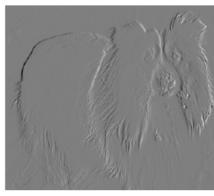
 How sparse connectivity and parameter sharing can dramatically improve efficiency of image edge detection is¹³ shown in next slide

Efficiency of Convolution for Edge Detection

- Image on right formed by taking each pixel of input image and subtracting the value of its neighboring pixel on the left
 - This is a measure of all the vertically oriented edges in input image which is useful for object detection

Input image





Both images are 280 pixels tall Input image is 320 pixels wide Output image is 319 pixels wide

- Transformation can be described by a convolution kernel containing two elements and requires $319 \times 320 \times 3 = 267,960$ flops (2 mpys, one add)
- Same transformation would require $320 \times 280 \times 319 \times 280$, i.e., 8 billion entries in the matrix
- Convolution is 4 billion times more efficient

Equivariance of Convolution to Translation

- The particular form of parameter sharing leads to equivariance to translation
- Equivariant means that if the input changes, the output changes in the same way
- A function f(x) is equivariant to a function g if f(g(x)) = g(f(x))
- If g is a function that translates the input, i.e., that shifts it, then the convolution function is equivariant to g
 - I(x,y) is image brightness at point (x,y)
 - I'=g(I) is image function with I'(x,y)=I(x-1,y), i.e., shifts every pixel of I one unit to the right
 - If we apply g to I and then apply convolution, the output will be the same as if we applied convolution to I, then applied transformation g to the output

Example of equivariance

- With 2D images convolution creates a map where certain features appear in the input
- If we move the object in the input, the representation will move the same amount in the output
- It is useful to detect edges in first layer of convolutional network
- Same edges appear everywhere in image, so it is practical to share parameters across entire image

Absence of equivariance

- In some cases, we may not wish to share parameters across entire image
 - If image is cropped to be centered on a face, we may want different features from different parts of the face
 - Part of the network processing the top of the face looks for eyebrows
 - Part of the network processing the bottom of the face looks for the chin
- Certain image operations such as scale and rotation are not equivariant to convolution
 - Other mechanisms are needed for such transformations