CSCE 970 Lecture 4: Convolutional Neural Networks

Stephen Scott and Vinod Variyam

sscott@cse.unl.edu

Introduction

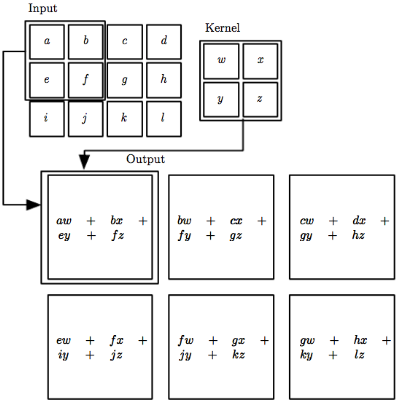
- Good for data with a **grid-like topology**
  - Image data
  - Time-series data
  - We'll focus on images
- Based on the use of **convolutions** and **pooling**
  - Feature extraction
  - Invariance to transformations
- Parallels with biological **primary visual cortex**
  - Arrangement as a **spatial map**
  - Use of **simple cells** for low-level detection
  - Use of **complex cells** for invariance to transformations

Outline

- Convolutions
- CNNs
- Pooling
- Variations
- Completing the network

Convolutions

- A **convolution** is an operation that computes a weighted average of a data point and its neighbors
- Weights provided by a **kernel**



Applications:

- De-noising
- Edge detection
- Image blurring
- Image sharpening

- Define a small, 2-dimensional **kernel** over the image I
- At image pixel $I_{i,j}$, multiply $I_{i-1,j-1}$ by kernel value $K_{1,1}$, and so on, and add to get output $I'_{i,j}$

-1	0	$+1$
-2	0	$+2$
-1	0	$+1$

This kernel measures the **image gradient** in the x direction

Example: **Sobel** operator for edge detection

G_x

-1	0	$+1$
-2	0	$+2$
-1	0	$+1$

G_y

$+1$	$+2$	$+1$
0	0	0
-1	-2	-1

Pass G_x and G_y over image and add gradient results

Convolutions
Example: Image Blurring

A **box blur** kernel computes uniform average of neighbors

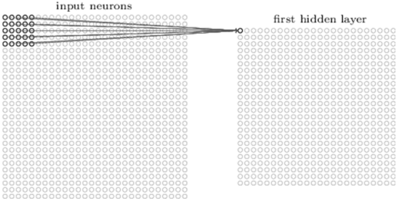
1	1	1
1	1	1
1	1	1

Apply same approach and divide by 9:

- Use of pre-defined kernels has been common in feature extraction for image analysis
- But how do we know if our pre-defined kernels are best for the specific learning task?
- Convolutional nodes in a CNN will allow the network to learn which features are best to extract
- We can also have the network learn which invariances are useful

Basic Convolutional Layer

- Imagine kernel represented as weights into a hidden layer
- Output of a linear unit is exactly the kernel output
- If instead use, e.g., ReLU, get nonlinear transformation of kernel



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

- Note that, unlike other network architectures, do not have complete connectivity

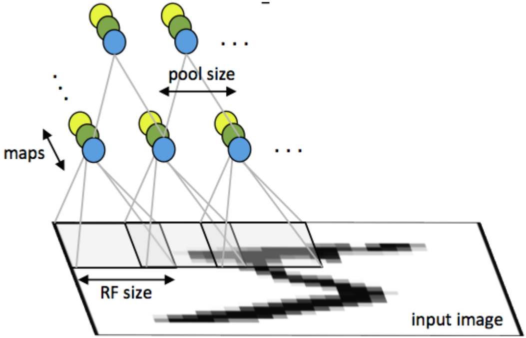
$\Rightarrow$ Many fewer parameters to tune

Basic Convolutional Layer
Parameter Sharing

- Sparse connectivity from input to hidden greatly reduces paramters
- Can further reduce model complexity via **parameter sharing** (aka **weight sharing**)
- E.g., weight $w_{1,1}$ that multiplies the upper-left value of the window is the same for all applications of kernel

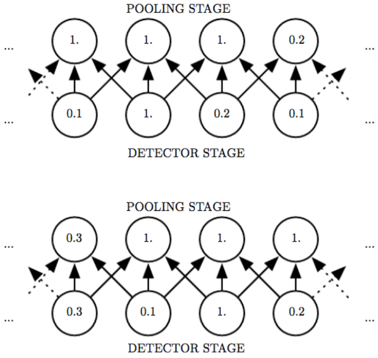
Basic Convolutional Layer
Multiple Sets of Kernels

- Weight sharing forces the convolution layer to learn a specific feature extractor
- To learn multiple extractors simultaneously, can have multiple convolution layers
  - Each is independent of the other
  - Each uses its own weight sharing



Pooling

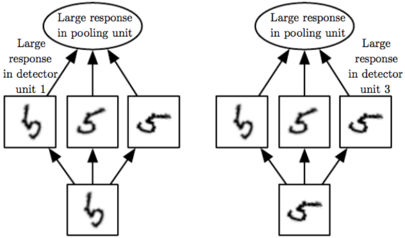
- Often more interested in presence/absence of a feature rather than its exact location
- To help achieve translation invariance, can feed output of neighboring convolution nodes into a **pooling node**
- Pooling function can be average of inputs, max, etc.



Pooling

Other Transformations

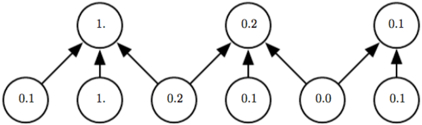
- Pooling on its own won't be invariant to, e.g., rotations
- Can leverage multiple, parallel convolutions feeding into single (max) pooling unit



Pooling

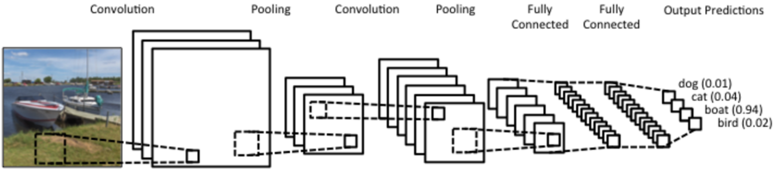
Downsampling

- To further reduce complexity, can space pooled regions at $k > 1$ pixels apart
- Parameters: **window width** (3) and **stride** (2)
- Dynamically adjusting stride can allow for variable-sized inputs



Completing the Network

Can use multiple applications of convolution and pooling layers



Final result of these steps feeds into fully connected subnetworks with, e.g., ReLU and softmax units