CSCE 496/896 Lecture 4: Convolutional Neural Networks

Stephen Scott

sscott@cse.unl.edu

Introduction

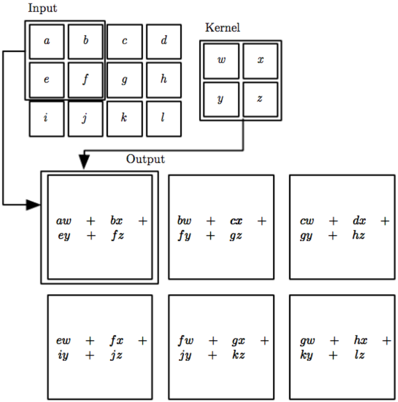
- Good for data with a **grid-like topology**
  - Image data
  - Time-series data
  - We'll focus on images
- Based on the use of **convolutions** and **pooling**
  - Feature extraction
  - Invariance to transformations
  - Parameter-efficient
- Parallels with biological **primary visual cortex**
  - Use of **simple cells** for low-level detection
    - Each has a **local receptive field** covering a small region of the visual field
    - Each tends to respond to **specific patterns**, e.g., vertical lines
  - Use of **complex cells** for invariance to transformations

Outline

- Convolutions
- CNNs
- Pooling
- Completing the network
- Example architectures

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

Convolutions

Example [Image from Kenneth Dwain Harrelson]

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

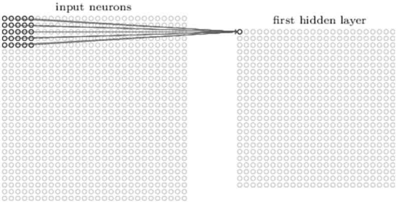
1	1	1
1	1	1
1	1	1

Apply same approach and divide by 9:

- Use of pre-defined kernels has been common in feature extraction for image analysis
  - User specified kernels, applied them to input image, and processed results into features for learning algorithm
- 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

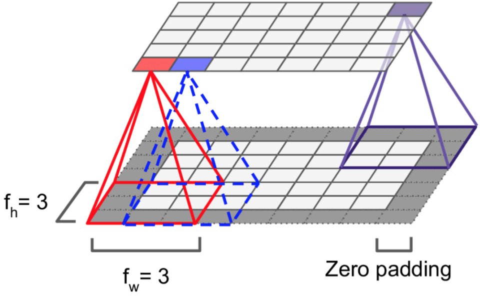


Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

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

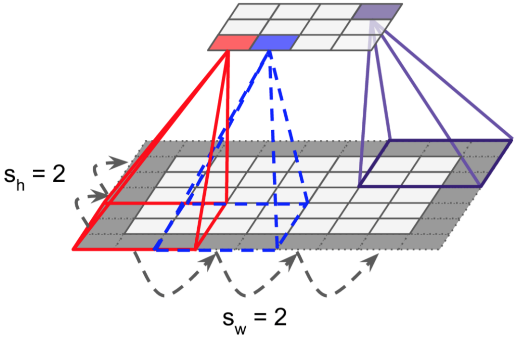
$\Rightarrow$ Many fewer parameters to tune

Neuron at row i, column j connects to previous layer's rows i to $i + f_h - 1$ and columns j to $j + f_w - 1$



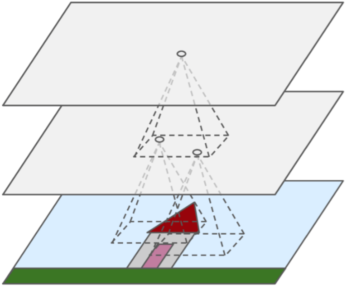
Apply **zero padding** at boundary

Can reduce size of layers by **downsampling** with a **stride** parameter



Neuron at row i, column j connects to previous layer's rows is_h to $is_h + f_h - 1$ and columns js_w to $js_w + f_w - 1$

Often use multiple convolutional layers in a **convolutional stack**



Extends a higher-layer node's receptive field

Basic Convolutional Layer

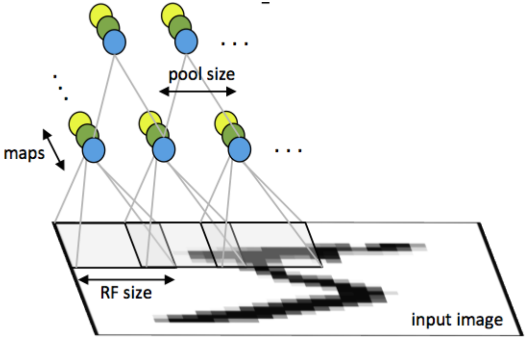
Parameter Sharing

- 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

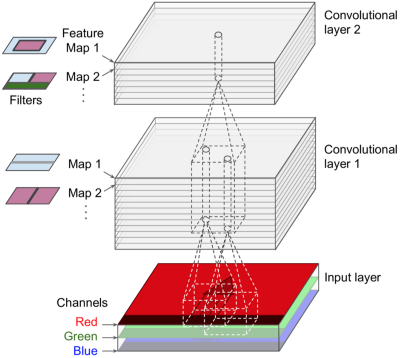
Multiple Sets of Kernels

- 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



Can also span multiple **channels** (e.g., color planes)

- A neuron's receptive field now spans all feature maps of previous layer
- Neuron at row i, column j of feature map k of layer ℓ connects to layer $(\ell - 1)$'s rows is_h to $is_h + f_h - 1$ and columns js_w to $js_w + f_w - 1$, spanning all feature maps of layer $\ell - 1$



Basic Convolutional Layer

Multiple Sets of Kernels

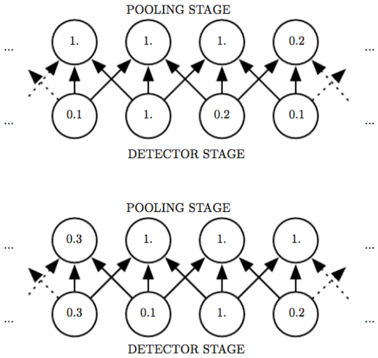
- Let z_{ijk} be output of node at row i, column j, feature map k of current layer ℓ
- Let s_h and s_w be strides, receptive field be $f_h \times f_w$, and let $f_{n'}$ be number of feature maps in layer $\ell - 1$
- Let $x_{i'j'k'}$ be output of layer-$(\ell - 1)$ node in row i', column j', feature map (channel) k'
- Let b_k be bias term for feature map k and $w_{uvk'k}$ be weight connecting any node in feature map k', position (u, v), layer $\ell - 1$, to feature map k in layer ℓ

$$z_{ijk} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i'j'k'} w_{uvk'k}$$

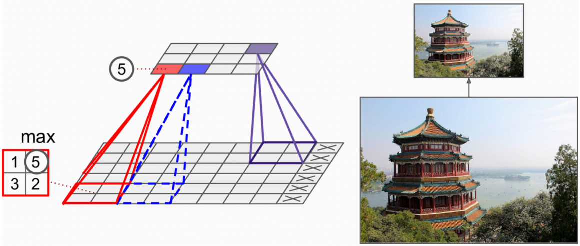
where $i' = is_h + u$ and $j' = js_w + v$

Pooling

- To help achieve translation invariance and reduce complexity, can feed output of neighboring convolution nodes into a **pooling node**
- Pooling function typically unweighted max or average of inputs



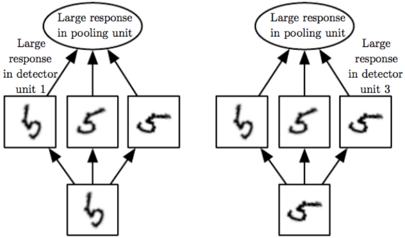
Pooling



Typically pool each channel independently (reduce dimension, not depth), but can also pool over depth and keep dimension fixed

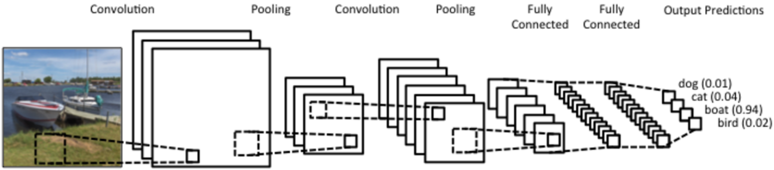
Pooling
Other Transformations

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



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

Considerations

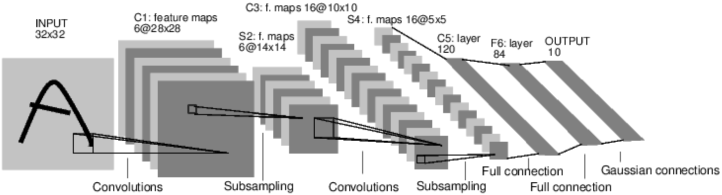
- CNNs are very flexible and very powerful, but:
  - Many hyperparameters to tune (number of filters, f_h, f_w, strides, etc.)
  - Training requires remembering all intermediate values computed (memory-intensive)
    - E.g., using filters of size 5×5, 200 feature maps each sized 150×100, stride 1, and inputs are 150×100 RGB images
    - Number of parameters is only 15200 (vs 675M for fully connected)
    - But to store all intermediate computations, need 11.4MB per instance
  - Need to keep these in mind when setting things up, and adjust architecture, mini-batch size, etc.

Example Architectures

- Performance of state-of-the-art systems often measured in **ILSVRC Image Net Challenge**
  - Large images, many classes, tough to distinguish
  - **Top-5 error rate:** Fraction of test images not in a system's top 5 predictions
- Notable systems:
  - LeNet-5
  - AlexNet
  - GoogLeNet
  - ResNet

Example Architectures

LeNet-5 (LeCun et al., 1998)

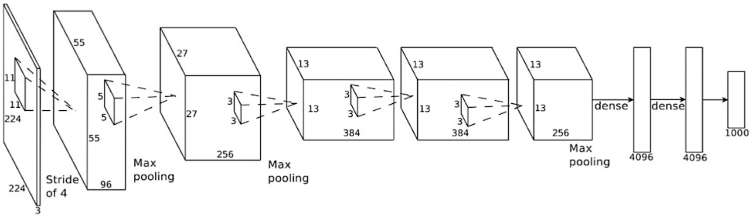


Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

- Output is **radial basis function**, one function per class

Example Architectures

AlexNet (Krizhevsky et al., 2012): 17% top-5 error rate

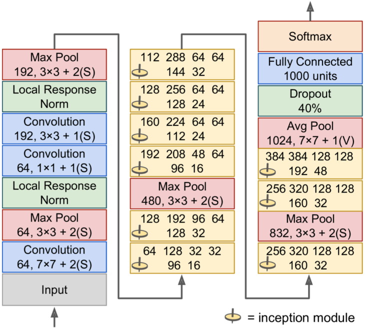


Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

- Didn’t strictly alternate convolutional and pooling layers
- **Local response normalization:** strong response at (i, j) inhibits same location in other feature maps

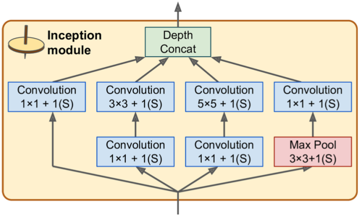
Example Architectures

GoogLeNet (Szegedy et al., 2014): 7% top-5 error rate



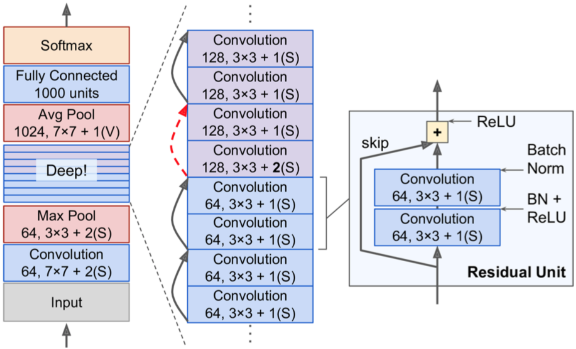
- **Inception modules** nest convolutions and pooling

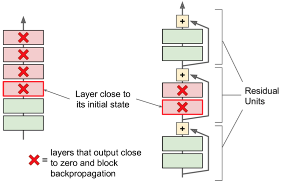
- Different kernel sizes capture features at different scales



Example Architectures

ResNet (Kaiming He et al., 2015): 3.6% top-5 error rate





- **Residual units** use **skip connections** to speed learning
  - Initial wts $\approx 0 \Rightarrow$ outputs $\approx 0 \Rightarrow$ depress error signal
  - Skip connections allow error signal to propagate faster