Convolutional Neural Networks (CNNs)

Keith L. Downing

The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
keithd@idi.ntnu.no

February 9, 2020

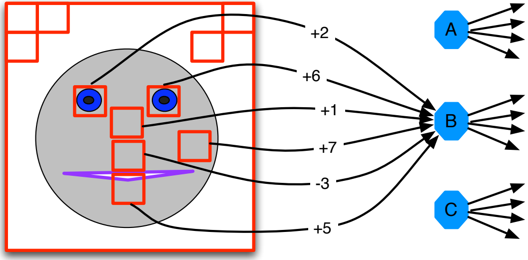
Key Elements of Recent Deep-Learning Success

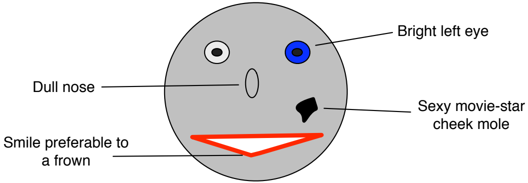
1. Rectified Linear Units (ReLU)
2. Dropout - different random subsets of neurons are temporarily silenced on different training cases.
3. Fast Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs).
4. Lots of data!!
5. Convolution Nets

4 Key Properties of Convolution Nets

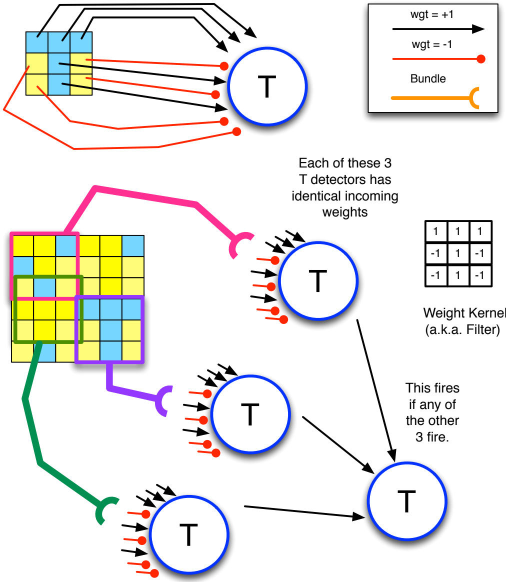
1. Local connections between layers - reduces weights to tune.
2. Shared weights among filters (a.k.a. kernels) - further reduces weights to tune.
3. Pooling - detects invariant patterns.
4. Multiple layers - hierarchical feature detection (a la brains).

Neurons as Pattern Detectors





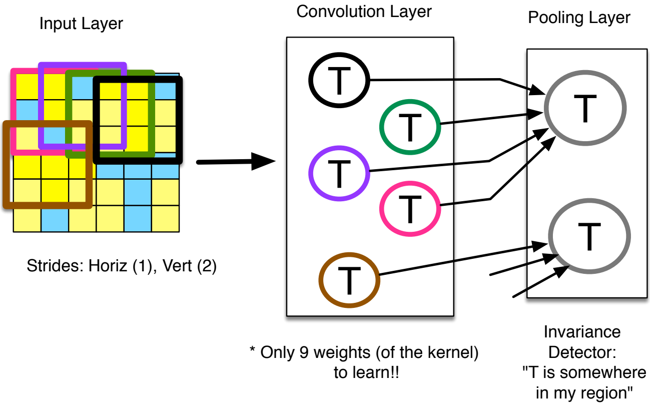
Spatially Invariant Patterns: Detect T anywhere.



1	1	1
-1	1	-1
-1	1	-1

Weight Kernel
(a.k.a. Filter)

Convolution and Pooling



Terminology varies: Neuron groups = feature maps, while combos of convolving (or pooling) kernels + in and out feature maps = convolution (or pooling) layers.

Convolution Networks (CNNs)

Convolution Nets = NNs that use convolution instead of matrix multiplication in at least one layer. (*Deep Learning*, Bengio, et. al., 2016, pg. 321)

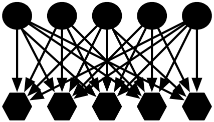
- Neocognitron (Fukushima, 1980) - deep, hierarchical, locally-connected, weight-sharing and pooling multilayered NNs based on mammalian visual system. Trained by unsupervised (competitive Hebbian) methods and a specialized form of supervised learning.
- Le Cun (1989) enhanced Neocognitron with backpropagation, multiple channels, more general pooling and a more flexible architecture.
- Le Cun's group had first practical app of CNNs in 1998: Optical Character Recognition (OCR).

Some Recent Success Stories

- ImageNet Competition (2012) - millions of images, thousands of classes; Deep CNNs dominated the competition.
- Deep Face (Facebook) - locally connected, but no weight sharing.
- Google Deep Mind
  - NN + RL for playing 49 different Atari Games.
  - AlphaGo - NN+RL+ MC search for world-class Go play.
- Deep Dreaming via Inceptionism (Google)

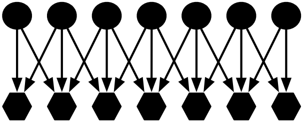
Sparse Connections and Parameter Sharing

Dense Connections



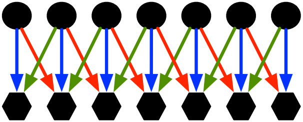
Full connection between layers. All weights are different

Sparse Connections



All weights are different

Shared Connections



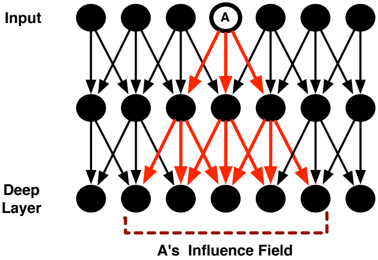
Only 3 (shared) weights

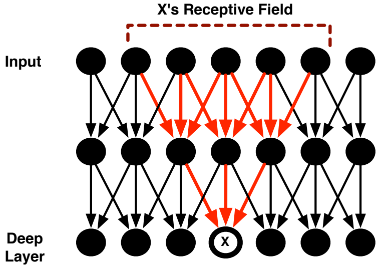
Why are dense, shared connections not a practical option?

Advantages of Convolution Networks

- **Sparse Connections**
  - Reduces number of weights to learn.
  - Can produce local receptive fields for neurons, and hierarchies of receptors with deeper neurons having wider fields but dealing with more abstract data.
- **Parameter Sharing**
  - Further reduces number of weights to learn.
  - Pattern Invariance - the network can learn to detect the same pattern in multiple locations (receptive fields), but it does not need to learn it multiple times, just once.
  - So patterns are invariant to translation, but not necessarily to rotation nor scaling.
- **Variable-Sized Inputs** - Since parameters are shared, the same CNN can often handle diverse input sizes, as long as those inputs have the same type of information, e.g. photos, MRI scans, audio time series...

Hierarchy of Receptive Fields





Convolution

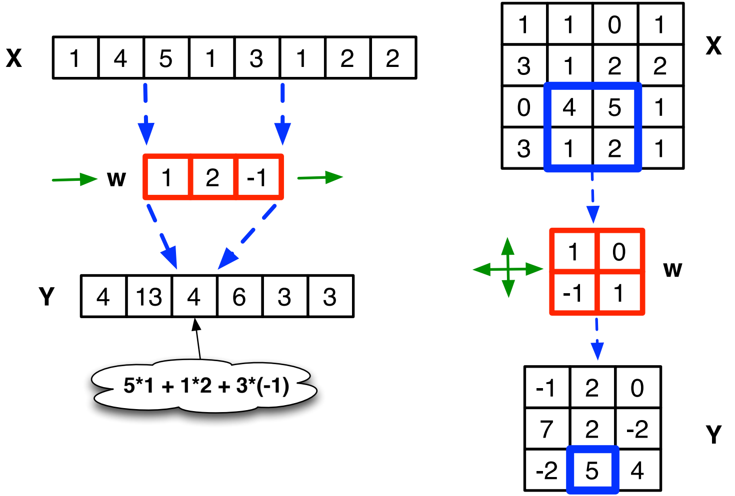
$$Y(k) = (X * w)(k) = \sum_{\delta=-d}^{d} X(k+\delta)w(\delta)$$

where:

- X = upstream layer or convolution *input*
- Y = downstream layer or convolution *feature map*
- w = the kernel or filter = a tensor of weights that normally has the same number of dimensions as X but is smaller in scope. Here, scope = 2d + 1.

- The convolution operation applies to the activations of one layer (i.e. neuron vector) to produce activations of next downstream layer.
- But some CNN definitions view a single *convolution layer* as the two neuron vectors plus the convolution(s) that connect them.

Convolving with the Kernel



Convolution in Two Dimensions

$$Y(j,k) = (X * w)(j,k) = \sum_{\gamma=-c}^{c} \sum_{\delta=-d}^{d} X(j+\gamma, k+\delta) w(\gamma, \delta)$$

- Begin in the upper left corner of X.
- Apply the kernel to create the upper-left entry of Y.
- Move the kernel horizontally along X, one stride at a time, applying it and producing a new entry for Y, in the corresponding row.
- After completing a row of X, return to the row start and shift the kernel down one stride and begin a new row in Y.
- Continue until the kernel is in the bottom right corner of X.

  Note: stride ≥ 1 and horizontal and vertical strides may differ.

Kernels as Pattern Detectors

1	1	1	0	0
1	1	1	0	0
1	1	1	0	0
0	1	1	1	0
0	1	1	1	1

Kernel

1	1	-1
1	1	-1
1	1	-1

Detects Right Edge of a Thick Object

3	6	3
2	5	4
1	4	4

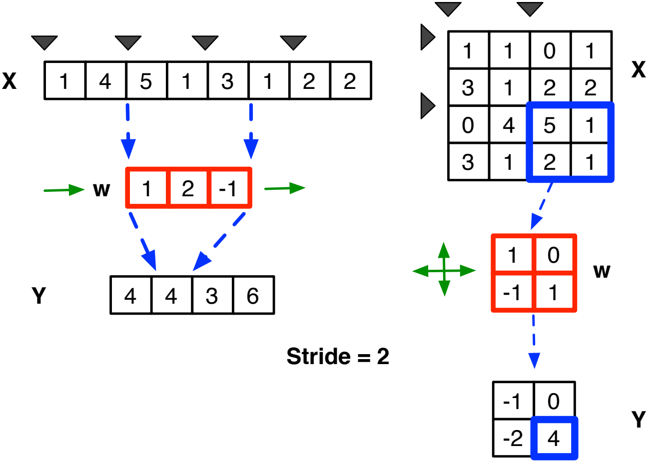
Zero Padding

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	1	1	1	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	1	1	1	1	0
0	0	0	0	0	0	0

0	2	4	2	0
0	3	6	3	0
-1	2	5	4	1
-2	1	4	4	3
-2	0	2	3	3

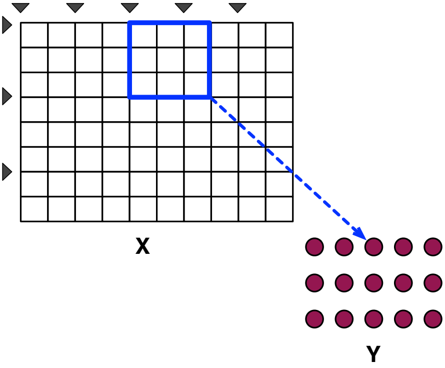
Zero padding combats shrinking layer sizes, which are not always desired.

Strides



The larger the strides, the greater the difference: size(X) - size(Y)

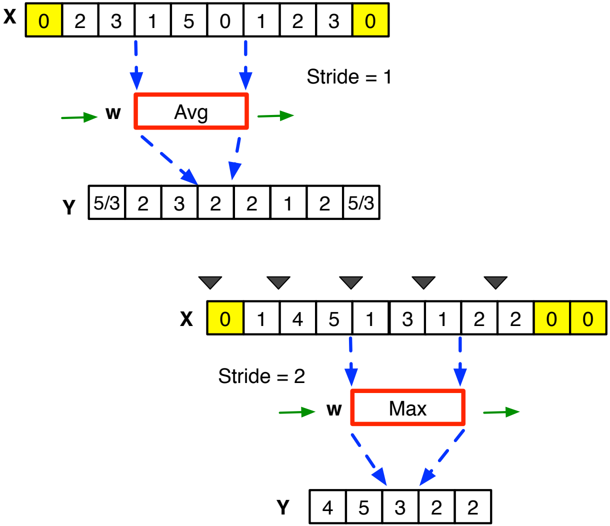
Strides > 1 → Size Reduction



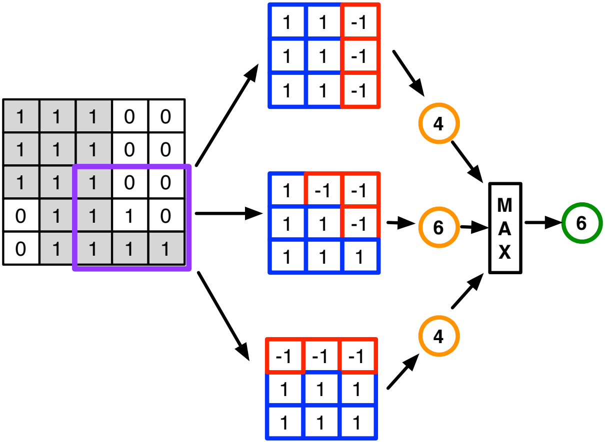
- $Size(Y) = \lceil \frac{R}{s_v} \rceil \times \lceil \frac{C}{s_h} \rceil$ (Assuming zero-padding)
- R = rows(X), C = columns(X), s_h = horizontal stride, s_v = vertical stride
- R = 8, C = 10, s_h = 2, s_v = 3
- $Size(Y) = \lceil \frac{8}{3} \rceil \times \lceil \frac{10}{2} \rceil = 15$; whereas Size(X) = 8 x 10 = 80

Pooling

Downstream layer computes statistical summaries of its upstream neighbor. This may or may not involve a layer-size reduction.



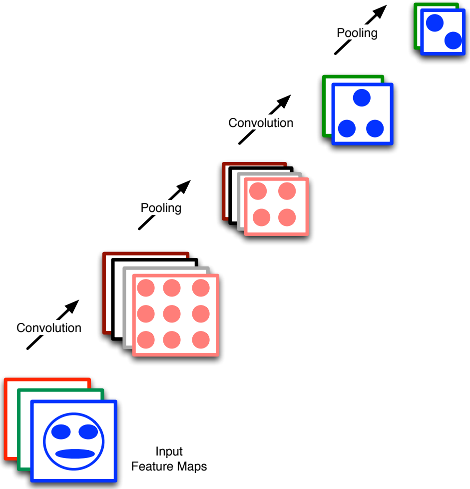
Pooling Across Different Kernels



Kernels = **Different Detectors for the Edge of a Thick Object**

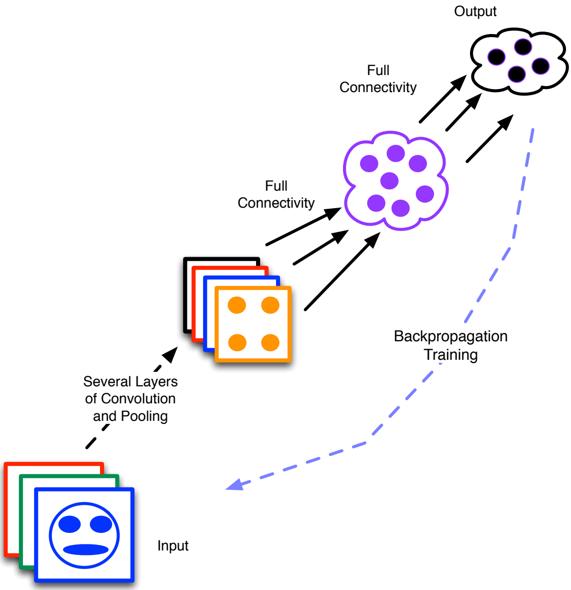
- Pooling can occur across different kernels and/or channels.
- Detect patterns invariant to scaling and rotation (not just translation).

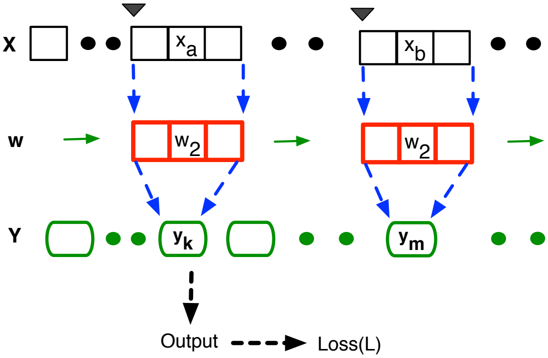
Combining Convolution and Pooling



- Pooling is not always necessary.
- Though not shown here, the number of feature maps often increases deeper into the net, to detect the multitude of higher-level patterns.

A Complete Convolution Network





- $\frac{\partial L}{\partial w_2} = \cdots + x_a \frac{\partial L}{\partial sum(y_k)} + \cdots + x_b \frac{\partial L}{\partial sum(y_m)} + \cdots$
- $\frac{\partial L}{\partial w_j} = \sum_{c \in M} \sum_{s \in S} \frac{\partial L}{\partial w_j}\Big|_{c,s}$

  where M = minibatch, S = locations in X where w is applied.

Gradient Combination for 1-D Convolution

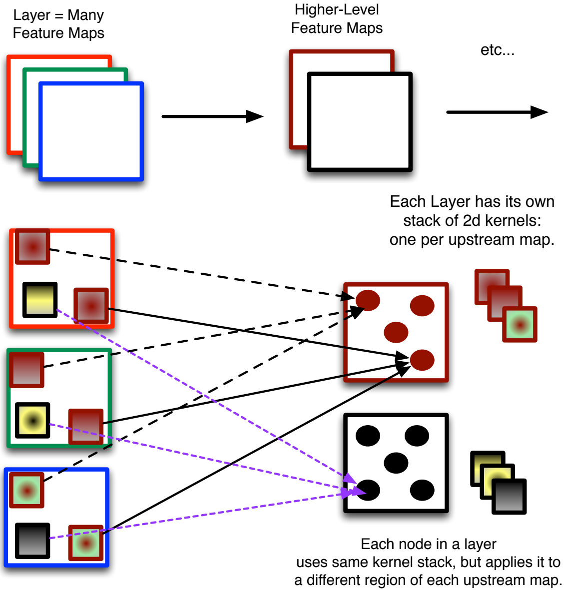
- i = index into upstream feature map X
- W = convolution weights
- j = index into W
- k = 0, index into downstream feature map Y
- pad= number of 0's on the left (and right); s = stride

- for xloc = -pad to len(X)-1 by s:
  - for j = 0 to len(W)-1 by 1:
    - i = xloc + j
    - if $0 \leq i < len(X)$:

$$\frac{\partial Loss}{\partial w_j} = \frac{\partial Loss}{\partial w_j} + x_i \times \frac{\partial Loss}{\partial Sum(y_k)}$$

  - $k \leftarrow k+1$

Layers, Feature Maps and Kernel Stacks



Formal Description of Layer-Kernel Relationships

- X = Upstream Layer; Y = Downstream Layer
- Both X and Y may contain many feature maps (a.k.a. *channels*)
- i = output channel; i.e. feature map of Y
- l = input channel, i.e. feature map of X
- j,k = 2-d coordinates in any layer
- K = stack of (m x n) kernels connecting X to Y

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m,k+n} \times K_{i,l,m,n}$$

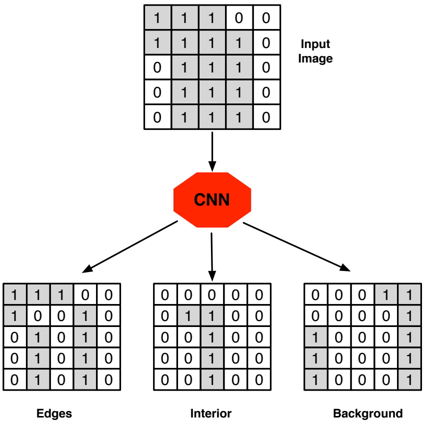
- Each value in an output channel is based on **all** values in the same **m x n window** of **some or all** input channels.

Data Formats for Applied Convolution Nets

	Single Channel	MultiChannel
1-D	Audio Time Series; values = amplitudes; convolve over time	Skeleton animations: time series of joint angles, one joint per channel.
2-D	Audio data: Fourier Series. rows = frequencies, colums = time points. Convolve over time or frequency to find invariants in either dimension.	Color images: 2-d coordinates + 3 channels (red, green, blue)
3-D	Volumetric Data; e.g. CT and MRI scans	Color Video: axes = (width, height, time); channels = (Red, Green, Blue)

Deep Learning(2016), Goodfellow et. al., pg. 349

Structured Outputs



- Multi-dimensional output tensor; one 2-d plane per **class**.
- Individual classification for each input pixel.
- Output values are binary for readability only; normally floats.
- Example: Classifying individual pixels in aerial photos as road, river, house, etc.

Final Words from the Masters

CNNs are a good example of an idea inspired by biology that resulted in competitive engineering solutions that compare favorably with other methods...Although applying CNNs to image recognition removes the need for a separate hand-crafted feature extractor, normalizing the images for size and orientation (if only approximately) is still required. Shared weights and subsampling (pooling) bring invariance with respect to small geometric transformations or distortions, but fully invariant recognition is still beyond reach. Radically new architectural ideas, possibly suggested by biology, will be required for a fully neural image or speech recognition system. .... **Bengio and Le Cun**, *The Handbook of Brain Theory and Neural Networks*, 2nd Edition, Arbib (2003), pp. 276-279.