Convolutional Neural Networks (CNNs)

Keith L. Downing

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

February 16, 2021
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

Convolutional Neural Networks (CNNs)

Keith L. Downing
Spatially Invariant Patterns: Detect T anywhere.

Each of these 3 T detectors has identical incoming weights.
This fires if any of the other 3 fire.

Weight Kernel (a.k.a. Filter)

Keith L. Downing
Convolutional Neural Networks (CNNs)
**Convolution and Pooling**

**Input Layer**

**Convolution Layer**

**Pooling Layer**

* Only 9 weights (of the kernel) to learn!!

* Invariance Detector:
  "T is somewhere in my region"

* Strides: Horiz (1), Vert (2)

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

Keith L. Downing  
Convolutional Neural Networks (CNNs)
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

Input

A's Influence Field

Deep Layer

X's Receptive Field

Input

Deep Layer

Convolutional Neural Networks (CNNs)
Convolution

\[ Y(k) = (X \ast w)(k) = \sum_{\delta=-d}^{d} X(k + \delta)w(\delta) \]

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

\[ w = \begin{bmatrix} 1 & 2 & -1 \end{bmatrix} \]

\[ X = \begin{bmatrix} 1 & 4 & 5 & 1 & 3 & 1 & 2 & 2 \end{bmatrix} \]

\[ Y = \begin{bmatrix} 4 & 13 & 4 & 6 & 3 & 3 \end{bmatrix} \]

\[ 5*1 + 1*2 + 3*(-1) \]

Keith L. Downing
Convolutional Neural Networks (CNNs)
Convolution in Two Dimensions

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

- 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: \( \text{stride} \geq 1 \) and horizontal and vertical strides may differ.
Kernels as Pattern Detectors

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Detected Right Edge of a Thick Object

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Kernel

<table>
<thead>
<tr>
<th>3</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Zero Padding

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Keith L. Downing

Convolutional Neural Networks (CNNs)
The larger the strides, the greater the difference: size(X) - size(Y)
Strides $> 1 \rightarrow$ Size Reduction

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

For multiple dimensions, treat each dimension independently: each has its own mode.
Pooling

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

Stride = 1

Stride = 2
Pooling can also occur across different kernels.
Detect patterns invariant to scaling and rotation (not just translation).
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

Several Layers of Convolution and Pooling

Full Connectivity

Output

Full Connectivity

Backpropagation Training

Input

Keith L. Downing
Convolutional Neural Networks (CNNs)
Layers, Feature Maps and Kernel Stacks

Layer = Many Feature Maps

Higher-Level Feature Maps

Each Layer has its own stack of 2d kernels: one per upstream map.

Each node in a layer uses same kernel stack, but applies it to a different region of each upstream map.
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
- I = 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.
Backpropagation over Convolution Layers

2 Input Channels
4 Output Channels
Minibatch Size = 3

\[ S(Y) = \text{sum wgt'd inputs to } Y \]

These are hard to represent explicitly as single tensors

Each minibatch case contains all channels

W = Kernel weights

4 filters

Minibatch

Input Channels

Output Channels (one per kernel)

Keith L. Downing
Convolutional Neural Networks (CNNs)
Gradients for Kernel Weights

\[ \frac{\partial L}{\partial w_2} = \cdots + x_a \frac{\partial L}{\partial \text{sum}(y_k)} + \cdots + x_b \frac{\partial L}{\partial \text{sum}(y_m)} + \cdots \]

\[ \frac{\partial L}{\partial w_j} = \sum_{c \in M} \sum_{s \in S} \frac{\partial L}{\partial w_j}|_{c,s} \]

where \( M = \text{minibatch} \), \( S = \text{locations in X where w is applied} \).

Note: Combining \( \frac{\partial L}{\partial w_j} \) over all \( c \in M \) as basis for computing \( \Delta w_j \)
Gradients for Input Activations

\[
\frac{\partial L}{\partial x_j} = \sum_{q \in Q} \sum_{w \in W} \frac{\partial L}{\partial x_j} \left|_{q,w} \right.
\]

where \( Q = \) kernels, \( W = \) weights within any kernel that apply to \( x_j \).

Separate \( \frac{\partial L}{\partial x_j} \) computed \( \forall c \in M \) (the minibatch), and passed upstream.
Kernel Gradient Combination (1-D)

- 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

∀ kernel (W): ∀ upstream feature map (X) do:
- for xloc = -pad to len(X)-1 by s:
  - for j = 0 to len(W)-1 by 1:
    - i = xloc + j
    - if 0 ≤ i < len(X):
      - \( \frac{\partial \text{Loss}}{\partial w_j} = \frac{\partial \text{Loss}}{\partial w_j} + x_i \times \frac{\partial \text{Loss}}{\partial \text{Sum}(y_k)} \)
  - \( k \leftarrow k + 1 \)

This Jacobian, \( J^L_W \), is used to modify the kernel.
Input Gradient Combination (1-D)

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

\[ \forall \ \text{upstream feature map } (X): \forall \ \text{kernel } (W), \text{ do:} \]

for \( \text{xloc} = -\text{pad} \) to \( \text{len}(X) - 1 \) by \( s \):
  - for \( j = 0 \) to \( \text{len}(W) - 1 \) by 1:
    - \( i = \text{xloc} + j \)
    - if \( 0 \leq i < \text{len}(X) \):
      \[ \frac{\partial \text{Loss}}{\partial x_i} = \frac{\partial \text{Loss}}{\partial x_i} + w_j \times \frac{\partial \text{Loss}}{\partial \text{Sum}(y_k)} \]
      
      \( k \leftarrow k + 1 \)

This Jacobian, \( J^L_X \), is sent back upstream. Minibatch dimension not shown.
- Layer X (size M) connects to Layer Y (size N), so weight matrix, W, is M x N (M rows, N columns).
- \(Y_s = \text{sum weighted inputs to } Y\). Assume it’s a row vector.
- \(\delta_Y = \text{Delta Jacobian for } Y\); \(J_X^L = \text{output Jacobian for } X\).

**Forward Pass**

- \(Y_s = X \cdot W\)
- \(Y_s = \text{np.einsum}('i,ij->j',X,W)\)
- Matching i’s on left-hand-side of einsum rule entails element-wise multiplication of items along the matched dimensions.
- Absence of i on right-hand side of the einsum rule entails addition of all pairwise multiplications.

**Backward Pass**

- \(J_X^L = \delta_Y \cdot W\)
- \(J_X = \text{np.einsum}('j,ij->i',\delta_Y,W)\)
Layer X outputs C channels of size M x N to Layer Y (of size P).
- Weight matrix, W, of size C x M x N x P
- \( Y_s \) = sum weighted inputs to Y. Assume it’s a row vector.
- \( \delta_Y \) = Delta Jacobian for Y; \( J^X_L \) = output Jacobian for X.

**Forward Pass**
- \( Y_s = X \cdot W \)
- \( Y_s = \text{np.einsum}(\text{ijkl}->l',X,W) \)
- Multiply and add along 3 different dimensions.

**Backward Pass**
- \( J^X_L = \delta_Y \cdot W \)
- \( J_\cdot X = \text{np.einsum}(\text{l,ijkl}->ijk',\text{delta}_Y,W) \)
- This creates a C x M x N Jacobian that the convolution layer can use to create its own delta Jacobian and continue backpropagating.
## Data Formats for Applied Convolution Nets

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

*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.
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.