Introduction to Convolutional Networks

[LeNet-5, LeCun 1980]
Computing Features

• Idea: each neuron on the higher layer is detecting the same feature, but in different locations on the lower layer
  • Detecting=the output is high if the feature is present

• It’s the same feature because the weights are the same

• Note: each neuron is only connected with non-zero weights to a small area in the input

The red connections all have the same weight.
Feature Detection

• The weights of each unit in the upper layer can be represented as a 2D array.

• To compute the input to each neuron in the upper layer, we are computing the dot product between the 2D array (called *kernel*) and the area of the lower layer to which the neuron is connected (called the *receptive field*).

• The operation of computing the feature layer from the lower layer is called *convolution* (technically, “cross-correlation,” but the differences between convolution and cross-correlation is unimportant here.)
Convolution Example: Sobel Filter

Vertical Edge (absolute value)
Convolution Example: Sobel Filter

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

* Horizontal Edge (absolute value)
**Convolution Example: Blob Detection**

\[
\begin{pmatrix}
0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\
0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\
3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\
2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\
2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\
2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\
3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\
0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\
0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0
\end{pmatrix}
\]

*
7x7 input (spatially) 
assume 3x3 filter
7x7 input (spatially)
assume 3x3 filter
7x7 input (spatially) assume 3x3 filter
7x7 input (spatially) assume 3x3 filter
7x7 input (spatially) assume 3x3 filter

=> 5x5 output
7x7 input (spatially) assume 3x3 filter applied with stride 2
7x7 input (spatially) assume 3x3 filter applied with stride 2
7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):

- stride 1 \(\Rightarrow \frac{7 - 3}{1} + 1 = 5\)
- stride 2 \(\Rightarrow \frac{7 - 3}{2} + 1 = 3\)
- stride 3 \(\Rightarrow \frac{7 - 3}{3} + 1 = 2.33 \downarrow\)
In practice: Common to zero pad the border

![Image of a pad matrix]

- e.g. input 7x7
- 3x3 filter, applied with **stride 1**
- **pad with 1 pixel** border => what is the output?

(recall:)

\[(N - F) / \text{stride} + 1\]
In practice: Common to zero pad the border

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- e.g. input 7x7
- 3x3 filter, applied with **stride 1**
- **pad with 1 pixel** border => what is the output?

**7x7 output!**
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

- e.g. F = 3 => zero pad with 1
- F = 5 => zero pad with 2
- F = 7 => zero pad with 3
Pooling Features ("subsampling")

• The job of complex cells

• Max Pooling
  • Is there a diagonal edge somewhere in an area of the image?
  • Take the maximum over the responses to the feature detector in the area

• Average Pooling
  • Is there a blobs pattern in an area of the image?
  • Take the average over the responses to the feature detectors in the area

• Max Pooling generally works better
Max Pooling as Hierarchical Invariance

- At each level of the hierarchy, we use an “or” to get features that are invariant across a bigger range of transformations.
- (Average Pooling is a little bit like an “AND”)
Putting it All Together

- Different types of layers: convolution and subsampling.
- Convolution layers compute features maps: the response to multiple feature detectors on a grid in the lower layer.
- Subsampling layers pool the features from a lower layer into a smaller feature map.
Why Convolutional Nets

• It’s possible to compute the same outputs in a fully connected neural network, but
  • The network is much harder to learn
  • There is more danger of overfitting if we try it with a really big network
    • A convolutional network has fewer parameters due to weight sharing*

• It makes sense to detect features and then combine them
  • That’s what the brain seems to be doing

* Small fully connected networks can work very well, but are hard to train
Learning Convolutional Nets: Replicated Weights

• \( v = g(Wu_1 + Wu_2) \)

• \( \frac{\partial v}{\partial W} = (u_1 + u_2)g'(Wu_1 + Wu_2) = u_1g'(Wu_1 + Wu_2) + u_2g'(Wu_1 + Wu_2) \)

• Note: if \( u_1 \) is positive but \( u_2 \) is negative, \( W \) will be “pulled” in different directions by the two
Learning Convolutional Nets: Max Pooling

\[ \frac{\partial v}{\partial u_i} = \begin{cases} 1, & u_i > u_j, \forall j \neq i \\ 0, & otherwise \end{cases} \]

- The u’s are real, so let’s not worry about them being equal.
- The gradient only flows to the unit that’s responsible for the value of \( v \).
  - Makes sense! The other ones aren’t likely detecting any patterns.

\[ v = \max(u_1, u_2) \]
LeNet:

[LeNet-5, LeCun 1980]
A Brute Force Approach

• Convolutional Networks architectures use knowledge about invariances to design the network architecture/weight constraints

• But it’s much simpler to incorporate knowledge of invariances by just creating extra training data:
  • for each training image, produce new training data by applying all of the transformations we want to be insensitive to (Le Net can benefit from this too)
  • Then train a large, dumb net on a fast computer.
  • This works surprisingly well if the transformations are not too big