Convolutional Neural Networks I

CS 4391 Introduction Computer Vision
Professor Yu Xiang
The University of Texas at Dallas

Some slides of this lecture are courtesy Stanford CS231n
Visual Perception vs. Computational Perception

- Depth
- Motion
- Object classes
- Object poses
- Etc.
Mathematic Models

• Try to model the human brain with computational models, e.g., neural networks
Mathematic Models

• What is the form of the function $f(x)$?
  • No idea!
  • Concatenate simple functions (neurons)

$x \rightarrow f(x) \rightarrow y \in \{+1, -1\}$

Dog
Neural Network: Concatenation of functions

Linear score function: \( f = Wx \)

2-layer Neural Network

\[
f = f_2(f_1(x)) = W_2 \max(0, W_1 x)
\]

Non-linearity

\[
h = f_1(X) \quad s = f_2(h)
\]

Need to learn the weights!
Frank Rosenblatt’s Perceptron

Schematic of a biological neuron.

\[
\sigma(w^T x + b) = \begin{cases} 
1 & \text{if } w^T x + b \geq 0, \\
0 & \text{otherwise.}
\end{cases}
\]

Frank Rosenblatt (1928-1971)
Activation Functions

2-layer Neural Network

\[ f = f_2(f_1(x)) = W_2 \max(0, W_1 x) \]

Rectified Linear Unit (ReLU)

\[ \max(0, x) \]

Introduce non-linearity to the network
Activation Functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**
\[ \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
**Exponential Linear Unit**
\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
Fully Connected Layer

\[ y = Wx \]

- \( x : n \times 1 \)
- \( W : m \times n \)
- \( y : m \times 1 \)
Fully Connected Layer

• What is the drawback of only using fully connected layers?

\[ y = Wx \]

• Consider an image with 640 x 480
  • x is with dimension 307,200
  • The weight matrix of the fully connect layer is too large
Convolutional Layers

• Consist of convolutional filters

• Share weights among different image locations

\[
g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

Gaussian Filter
Convolutional Neural Networks

Input image

Convolutional layer (translation invariant)

ReLU layer

Pooling layer

Fully connected layer

Output vector
Convolutional Neural Networks

[LeNet-5, LeCun 1980]
Convolutional Layer

32x32x3 image

- width: 32
- height: 32
- depth: 3
Convolutional Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolutional Layer

$32 \times 32 \times 3$ image

$5 \times 5 \times 3$ filter $w$

1 number:
the result of taking a dot product between the filter and a small $5 \times 5 \times 3$ chunk of the image
(i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$w^T x + b$
Convolutional Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
Convolutional Layer

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
Convolutional Layer

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1
Convolutional Layer

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

=> 5x5 output
Convolutional Layer

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 2
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, \textbf{with stride 2}
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, **with stride 2**

=> 3x3 output!
Convolutional Layer

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, **with stride 3**
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 3

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
\[(N - F) / \text{stride} + 1\]

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

\[
\begin{align*}
\text{stride 1} & \Rightarrow (7 - 3)/1 + 1 = 5 \\
\text{stride 2} & \Rightarrow (7 - 3)/2 + 1 = 3 \\
\text{stride 3} & \Rightarrow (7 - 3)/3 + 1 = 2.33
\end{align*}
\]
In practice: Common to zero pad the border

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

(recall:)
(N - F) / stride + 1

<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>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In practice: Common to zero pad the border

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

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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Convolutional Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolutional Layer

Consider a second, green filter.

Convolve (slide) over all spatial locations.

32x32x3 image
5x5x3 filter

Activation maps
Convolutional Layer

For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps.

We stack these up to get a “new image” of size 28x28x6!
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
Convolutional Neural Networks

Input image → Convolutional layer → ReLU layer → Pooling layer → Fully connected layer → Output vector
Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:
Max Pooling Layer

Single depth slice

```
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
```

max pool with 2x2 filters and stride 2

```
6 8
3 4
```
Further Reading

• Stanford CS231n, lecture 5, Convolutional Neural Networks
  http://cs231n.stanford.edu/schedule.html

• Deep learning with PyTorch
  https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

• AlexNet (2012):
  https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

