Deep Learning 236606
Lecture 6 (part 2)
Convolutional layers and networks
• Motivation: image processing and machine vision
• Convolutional filters
• Strided convolutions and padding
• Convolution on volumes
• The convolution layer
• Pooling layer
• Convolution network
• Transfer learning (on a nut shell)
CNNs
Can we process images with NNs?

- We would like to feed images to NNs
- Small images are no problem:

\[ h = 128^2 \times 3 \approx 49,000 \]

\[ n^2 \approx 2 \times 10^9 \]
• We would like to feed images to NNs
• Big images: problematic

\[ n = (1024)^2 \times 3 \]
\[ n^2 \approx 1 \times 10^{12} \]
\[ 10^{13} \]
• Discrete averaging (box) filter

\[
\begin{align*}
\frac{4}{3} + \frac{3}{3} + \frac{2}{3} &= 3 \\
\frac{1}{3} + \frac{2}{3} + \frac{2}{3} &= 3 \\
\frac{1}{3} + \frac{2}{3} &= 0 \\
\frac{1}{3} &= 0
\end{align*}
\]
- **Discrete averaging (box) filter**

\[
\begin{array}{c}
  f \\
  4 & 3 & 2 & -5 & 3 & 5 & 2 & 5 & 5 & 5 & 6 \\
  g \\
  1/3 & 1/3 & 1/3 \\
  f \ast g \\
  3 & 0 & 0 & 1 & 10/3 & 4 & 4 & 16/3 \\
\end{array}
\]
2D Convolutional filter

Image 5x5

-5  3  2  -5  3
-4  3 -1  2  1 -3
 1 -1  1  3  3  5
-2  0  1  4  4
5  6  7  9 -1

\[-3 - 4 + 5 \cdot 3 - 2 = 6\]
2D Convolutional filter

Image 5x5

<table>
<thead>
<tr>
<th>-5</th>
<th>3</th>
<th>2</th>
<th>-5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>-1</td>
</tr>
</tbody>
</table>

Output 3x3

<table>
<thead>
<tr>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
0 -1 0
-1 5 -1
0 -1 0
```
2D Convolutional filter

```
<table>
<thead>
<tr>
<th>-5</th>
<th>3</th>
<th>2</th>
<th>-5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>-1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
2D Convolutional filter

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>3</td>
<td>2</td>
<td>-5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>-1</td>
</tr>
</tbody>
</table>

Image 5x5

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>-7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2D Convolutional filter

![Image of 2D Convolutional filter with a 5x5 image and a 3x3 filter applied](image)

<table>
<thead>
<tr>
<th>-5</th>
<th>3</th>
<th>2</th>
<th>-5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
2D Convolutional filter

Image 5x5

-5  3   2  -5  3
4   3   2   1  -3
1   0   3   3   5
-2  0   1   4   4
 5   6   7   9  -1

0  -1  0
-1  5  -1
 0 -1  0

6  1   8
-7  9   2
2D Convolutional filter

Image 5x5

\[
\begin{array}{cccccc}
-5 & 3 & 2 & -5 & 3 \\
4 & 3 & 2 & 1 & -3 \\
1 & 0 & 3 & 3 & 5 \\
-2 & 0 & 1 & 4 & 4 \\
5 & 6 & 7 & 9 & -1 \\
\end{array}
\]

\[
\begin{array}{ccc}
6 & 1 & 8 \\
-7 & 9 & 2 \\
-4 & & \\
\end{array}
\]
### 2D Convolutional Filter

**Image 5x5**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>3</td>
<td>2</td>
<td>-5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

The 2D convolutional filter is applied to the image 5x5. The filter is highlighted in red and the computed values are shown in the right table.
2D Convolutional filter

Image 5x5

<table>
<thead>
<tr>
<th>-5</th>
<th>3</th>
<th>2</th>
<th>-5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>-4</td>
<td>-9</td>
<td>3</td>
</tr>
</tbody>
</table>
Many useful convolutional filters

Original image and the filter are convolved to get the filtering result

**Identity (Original unfiltered image)**

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

**Sharpening**

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 9 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]

**Smoothing**

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 2 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

**Edge detection**

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Edge detection

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 9 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Vertical edge detection alone

Vertical edge detector

\[
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix}
\]

=
Image processing research yielded many types of interesting edge detection filters. For example:

Which filter is best?
Convolutional networks (CNNs): learn filter from data per your task!
• What about the boundaries?

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
<th>-5</th>
<th>3</th>
<th>5</th>
<th>2</th>
<th>5</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>??</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10/3</td>
<td>4</td>
<td>4</td>
<td>16/3</td>
<td>??</td>
</tr>
</tbody>
</table>

• In 2D:

Padding
Padding

- Size loss:

\[ N \times N \quad \ast \quad f \times f \quad = \quad (N-f+1) \times (N-f+1) \]

image \hspace{1cm} \text{filter} \hspace{1cm} \text{resulting image}

\[ 6 \times 6 \quad 3 \times 3 \quad 4 \times 4 \]
• What’s the problem?

• **Image shrinks**: we cannot repeat this many times

• **Information loss**: pixels in the middle contribute to convolution values more than pixels on the edges
• The solution: **padding**

Suppose we pad with $p$ pixels

**“Same”** padding: resulting image is of same size ("valid" padding = no padding)

Filter size $f$ is typically **odd** and "same" padding requires: $p = (f-1)/2$
• Stride = 2
• Stride = 2
Strided convolution

- Stride = 2
Strided convolution

- Stride = 2
Why use strides > 1?

**Strided convolution with padding**

- Image: $N \times N$
- Filter: $f \times f$
- Padding: $p$
- Stride: $s$

**Resulting image:** \[
\left( \frac{N+2p-f}{s} + 1 \right) \times \left( \frac{N+2p-f}{s} + 1 \right)
\]
2D convolutions on volumes
2D convolutions on volumes

\[ 6 \times 6 \times 3 \]

\[ \ast \]

\[ 3 \times 3 \times 3 = 27 \]

\[ \text{func} \]

\[ = \]
2D convolutions on volumes
2D convolutions on volumes
2D convolutions on volumes
2D convolutions on volumes
2D convolutions on volumes
2D convolutions on volumes
2D convolutions on volumes
2D convolutions on volumes
Channels/depth

\[ \begin{array}{c}
\text{Channels} \\
\text{Depth}
\end{array} \]
Many channels
Another filter?
Many filters
Many filters

Result can be is stacked
But not yet!
We need non-linearity
Many filters
A CNN Layer

\[
\begin{align*}
\left( \left( \begin{array}{*{20}c}
\end{array} \right) \ast \left( \begin{array}{*{20}c}
\end{array} \right) + b_1 \right) &= \left( \left( \begin{array}{*{20}c}
\end{array} \right) + b_1 \right) \\
\left( \left( \begin{array}{*{20}c}
\end{array} \right) \ast \left( \begin{array}{*{20}c}
\end{array} \right) + b_2 \right) &= \left( \left( \begin{array}{*{20}c}
\end{array} \right) + b_2 \right) \\
\left( \left( \begin{array}{*{20}c}
\end{array} \right) \ast \left( \begin{array}{*{20}c}
\end{array} \right) + b_3 \right) &= \left( \left( \begin{array}{*{20}c}
\end{array} \right) + b_3 \right) \\
\left( \left( \begin{array}{*{20}c}
\end{array} \right) \ast \left( \begin{array}{*{20}c}
\end{array} \right) + b_4 \right) &= \left( \left( \begin{array}{*{20}c}
\end{array} \right) + b_4 \right) \\
\vdots &= \vdots \\
\left( \left( \begin{array}{*{20}c}
\end{array} \right) \ast \left( \begin{array}{*{20}c}
\end{array} \right) + b_k \right) &= \left( \left( \begin{array}{*{20}c}
\end{array} \right) + b_k \right)
\end{align*}
\]
A CNN Layer

ReLU

( ) + b1 = ReLU

ReLU

( ) + b2 = ReLU

ReLU

( ) + b3 = ReLU

ReLU

( ) + b4 = ReLU

\ldots = \ldots

ReLU

( ) + bk = ReLU

( ) + bk
A CNN Layer

ReLU

\( \begin{pmatrix} \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \end{pmatrix} \) \( \begin{pmatrix} \ast \\ \ast \\ \ast \\ \ast \\ \ast \end{pmatrix} \begin{pmatrix} \text{+} \\ \text{+} \\ \text{+} \\ \text{+} \\ \text{+} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_k \end{pmatrix} \) = \( \begin{pmatrix} \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \end{pmatrix} \) \( \begin{pmatrix} \text{+} \\ \text{+} \\ \text{+} \\ \text{+} \\ \text{+} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_k \end{pmatrix} \)

Result is stacked

Ready for a subsequent
Multiple convolution
Parameter arithmetic example: the first CNN layer in a network has 100 3x3x3 filters. How many parameters does it have?
### Pooling layers

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Pooling layers

Max pooling
With 2x2 filters
Stride 2
Pooling layers

• When pooling volumes: each slice (channel) has its own pooling image:

• Common pooling types: max and average
• Pooling has hyper-parameters, but no learnable parameters!

• Pooling layer size:
• Why do pooling?
CNN: is about weight sharing!
A simple CNN architecture

- Image
- Conv
- Max Pool
- Conv
- Max Pool
- Conv
- Max Pool
- Conv
- Max Pool
- Conv
- Fully connected
- Fully connected
- Fully connected
Transfer learning

• Suppose we have lots of data from source domain S
• We need to build a model for a related target domain T
• We have none, or very few labeled instances for T
• What could we do?
Transfer learning

Train on S

- Decision
  - Max Pool
  - Conv
  - Conv
- Object parts
  - Max Pool
  - Conv
  - Conv
- Simple geometrical patterns
  - Max Pool
  - Conv
  - Conv
- Edges
  - Max Pool
  - Conv
  - Conv
- Image

Edges

Simple geometrical patterns

Object parts

Decision

Train on S
Transfer learning

Train on S

Decision

Object parts

Simple geometrical patterns

Edges

Transfer to T

Fully connected

Max Pool

Conv

Conv

Max Pool

Conv

Conv

Max Pool

Conv

Conv

Max Pool

Conv

Conv

Image

Image
Transfer learning

Train on S

- Image
- Conv
- Conv
- Max Pool
- Conv
- Conv
- Max Pool
- Conv
- Conv
- Max Pool
- Conv
- Conv
- Max Pool
- Conv
- Conv

- Fully connected
- Fully connected
- Fully connected

Decision

Object parts

Simple geometrical patterns

Edges

Transfer to T

- Image
- Conv
- Conv
- Max Pool
- Conv
- Conv
- Max Pool
- Conv
- Conv
- Max Pool
- Conv
- Conv

- Fully connected
- Fully connected
- Fully connected

Retrain

Freeze

- Freeze