Convolutional Neural Networks (CNNs)
- convolutional filters
- CNN architecture
- LeNet 5
Why CNNs

- Appropriate for computer vision tasks:
  - Bounding boxes for objects
  - Background subtraction
  - Image classification

(Also used for problems from other domains)

- Key to good performance: Choice of feature map

\[ \phi : \mathbb{R}^{k \times k} \rightarrow \mathbb{R}^D \] (implemented as a NN)

These feature maps can be learned on one dataset and used on others.

- So far in this class, to do image classification:
  - \( \phi \) from flattening image into a vector
  - \( \phi \) from autoencoder (use encoder as feature map)
  - \( \phi \) from MLP

Then use logistic regression.
CNNs: another, biologically inspired, approach to learning good feature maps for vision problems.

Use convolutional filters

\[
\begin{array}{cccc}
1 & 2 & 1 & 0 \\
2 & 1 & 1 & 2 \\
0 & 0 & 1 & 2 \\
2 & 1 & 1 & 1 \\
0 & 1 & 2 & 0 \\
\end{array}
\] \times
\begin{array}{cccccc}
-1 & 0 & -1 \\
0 & 4 & 0 \\
-1 & 0 & -1 \\
\end{array}
= \begin{array}{cccc}
1 & 0 & 5 \\
-6 & -1 & 6 \\
1 & 1 & -2 \\
\end{array}
\]

\( I \ast K \)
- range may be different
- \( I \ast K \) is smaller than \( I \)

Our basic CNN layer takes as input an image \( I \), convolves it with filter \( K \), and returns \( \sigma(I \ast K) \)
Note that the filters always have size $k_1 \times k_2$ where $k_1$ & $k_2$ are odd

If the input image is size $d_1 \times d_2$, the output image is size $d_1 - (k_1 - 1) \times d_2 - (k_2 - 1)$

Formula for Convolutions

$$(I * K)_{i,j} = \sum_{m=-\frac{(k_1-1)}{2}}^{\frac{(k_1-1)}{2}} \sum_{n=-\frac{(k_2-1)}{2}}^{\frac{(k_2-1)}{2}} I_{i+m, j+n} K_{mn}$$

indices into $K$
Idea of CNNs: use convolutional filters to build our nonlinear feature map

Q: Why?  A: it has very desirable inductive biases
  - we expect that vision is translational invariant
  - intuition that image features should be local
    low-level!
  - cheap compared to MLP layer
    $\left( k_1 \cdot k_2 \text{ parameters} \right.$ vs $\left. (d_1-k_1+1)(d_2-k_2+1) \times d_1d_2 \right)$
  - easier to optimize b/c parameter sharing
Convolutional layers

Input: $d_1 \times d_2$
Filter: $k_1 \times k_2$
Output: $(d_1-k_1+1) \times (d_2-k_2+1)$
(Stride: 1)

(assuming $k_1 = k_2 = 3$)

$o(I*K)$ - learn $K$ with backprop

Stride: 2 in each direction.

In general, with a stride of $s_1 \times s_2$, we evaluate the filter at $x$-intervals of $s_1$ and $y$-intervals of $s_2$.

- Equivalent to computing $I*K$ and then throwing away the unused outputs.
Multichannel Images
-arise as RGB images or if we use multiple convolutional filters in the previous layer

Given an m channel image \([I_1, \ldots, I_m]\), one convolutional layer takes the form

\(\sigma(I_1 * K_1 + I_2 * K_2 + \ldots + I_m * K_m)\)
CNNs:
- combine convolutional layers with fully-connected layers

Image (RGB) → conv → t₁ conv features → t₂ conv features → t₃ conv feature

→ flatten → FCLayer 1 → FCLayer 2 → Softmax
Pooling

Pooling reduces the dimensionality of a convolutional layer by aggregating locally.

Typical pooling types:
- max pooling
- average pooling

Pooling’s advantages:
- decreases computation downstream
- makes model more robust to shifts in location of features
**Example of a CNN: LeNet-5 (1997)**

5-layer CNN for 10 class classification

<table>
<thead>
<tr>
<th>Layer</th>
<th>Features</th>
<th>Filler Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv2D</td>
<td>6</td>
<td>5x5</td>
<td>1</td>
<td>tanh</td>
</tr>
<tr>
<td>Aug Pooling</td>
<td>6</td>
<td>2x2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Conv2D</td>
<td>16</td>
<td>5x5</td>
<td>1</td>
<td>tanh</td>
</tr>
<tr>
<td>Aug Pooling</td>
<td>16</td>
<td>2x2</td>
<td>2</td>
<td>tanh</td>
</tr>
<tr>
<td>Conv2D</td>
<td>120</td>
<td>5x5</td>
<td>1</td>
<td>tanh</td>
</tr>
<tr>
<td>FC</td>
<td>84</td>
<td></td>
<td></td>
<td>softmax</td>
</tr>
<tr>
<td>FC</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>