Recap: standard ANN

\[ P_j = \sigma \left( \sum_i y_{ji} x_i \right) \]
\[ q_k = \sigma \left( \sum_j p_{jk} q_j \right) \]
\[ Y_l = \sigma \left( \sum_k \alpha_{lk} q_k \right) \]

weights that must be trained: \( Y \in \mathbb{R}^{M \times N} \), \( \beta \in \mathbb{R}^{N \times N} \), \( \alpha \in \mathbb{R}^{M \times M} \)

training examples: \( \{ x^{(1)}, \ldots, x^{(t)}, y^{(1)}, \ldots, y^{(F)} \} \)

why doesn’t this work for images? (specifically classification tasks)

- too many weights! 256 x 256 x 3 images, one fully-connected layer is 38.7 billion weights! Even with \( \sim 1000 \) neurons in first layer we would need \( \sim 200 \) million weights. Even if we could train something this large, we would overfit.

- does not take advantage of / exploit spatial invariance. i.e. a cat is a cat! doesn’t matter whether it’s in the top-left of the image or the bottom-right!
basic building blocks of CNN (convolutional neural network)

- convolution step
- subsampling step (pooling)
- optional extra filter or nonlinearity.

Standard CNN:

```
\[ \text{input} \rightarrow \text{convolutions} \rightarrow \text{pooling} \rightarrow \text{nonlinearity} \rightarrow \ldots \rightarrow \text{output} \]
```

one layer. more layers! standard fully-connected ANN.

**What is a "convolution"?**

It's a sliding window that essentially computes a dot-product.

Take a 1-D image for example: (pad with zeros),

\[
\begin{array}{ccccccc}
0 & x_1 & x_2 & x_3 & x_4 & x_5 & 0 \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
& w_1 & w_0 & w_1 & & & \\
\end{array}
\]

```
\[ \text{image} \rightarrow \text{conv. kernel} \rightarrow \text{output:} \]
\[
y_3 = w_1 x_2 + w_0 x_3 + w_1 x_4 \\
\]

Similarly: \( y_k = \sum_{i=-1}^{1} w_i x_{k+i} \) for all \( k \).

The filter \( w \) is like a small image!

Example: \( w = [1 \ 0 \ -1] \) is an edge detector.

Example: \( w = [0.06 0.24 0.40 0.24 0.06] \) is a smoothing filter.
Convolution also make sense in 2-D:

\[
\begin{align*}
  \text{output is an image } Y \in \mathbb{R}^{M \times N} \\
  \text{where: } \\
  Y_{ij} = \sum_{p=-D}^{D} \sum_{q=-D}^{D} W_{pq} X_{i+p, j+q}
\end{align*}
\]

If image is in color, there are three channels, i.e. \( X \in \mathbb{R}^{M \times N \times 3} \). Then filter is also \( W \in \mathbb{R}^{(2D+1) \times (2D+1) \times 3} \) and it slides over \( M \times N \) spatial map. Outputs are summed so \( Y \in \mathbb{R}^{M \times N} \) still.

Show demo of edge detector and smoothing operation in Matlab.

Can also use stride, i.e., do not apply convolution at every pixel, instead skip by \( s \) (\( s = 1 \) above). Using stride \( s \),

\[
Y_{ij} = \sum_{p=-D}^{D} \sum_{q=-D}^{D} W_{pq} X_{i+sp, j+sq}
\]

Output image is smaller: \( Y \in \mathbb{R}^{\frac{M}{s} \times \frac{N}{s}} \) (roughly).
What is pooling?

It's a subsampling operation that makes image smaller.

- Simplest: actually subsample; this might miss something!
- "Max pooling": choose largest value in same window, then slide window with some stride.

Example: max pooling 3x3 with stride 2:

Each cell is the max of a 3x3 cell in original image.

YeIR 7x7

Putting it together (e.g., 2 layers):

256x256x3

W_i ∈ IR^{5x5x3}
i = 1, ..., 100

100 convolution,
each is 5x5x3

256x256x100

max pooling

128x128x100

50 conv.
each 3x3x100

128x128x50

max pooling

w ∈ IR

3x3x100

i = 1, ..., 50

Train using back-propagation!
ImageNet: huge database of images (~15M images - 22k categories)
labeled using mechanical turk.

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge).

\[ \{ \sim 1000 \text{ images/category, } 1000 \text{ categories. Total: } 1.2 \text{M images} \]

for training and 150k images for testing. \]

Metric: "top-5" err → correct if one of top-5 labels predicted as most likely by algorithm is the correct label.

<table>
<thead>
<tr>
<th>Year</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>71.8%</td>
</tr>
<tr>
<td>2011</td>
<td>74.3%</td>
</tr>
<tr>
<td>2012</td>
<td>83.6%</td>
</tr>
<tr>
<td>2013</td>
<td>88.3%</td>
</tr>
<tr>
<td>2014</td>
<td>93.3%</td>
</tr>
</tbody>
</table>

→ Show Krizhevsky CNN architecture from their paper.

→ Lots of params! (2M for conv, 58M for fully-connected part).

→ Overcoming a real danger! Use tricks for data augmentation
  - reflections, random patches
  - brightness modulation

→ Show architecture from GoogleNet 2014 entry.
  Has only ~5M params, better performance. More depth
  (9 layers vs. 5). BUT it's less flexible. (weaker on
  other tasks besides image classification).