Overview

Topics covered in the lecture:

- Deep Architectures
- Convolutional Neural Networks (CNNs)
- Transfer learning
Why Deep Architectures?

- Is it better to use deep architectures rather than the shallow ones for complex nonlinear mappings?
- We know that deep architectures evolved in Nature (e.g., cortex)
- Universal approximation theorem: one layer is enough so why to bother with more layers?
  - deep neural networks can have exponentially less units than shallow networks for learning the same function
  - functions such as those realized by current deep convolutional neural networks are considered
- Handcrafted features vs. automatic extraction
- Gradually increasing complexity, intermediate representations: each successive layer brings higher abstraction
Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top activation signals and outputs for features maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g., eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form. The compression artifacts are a consequence of the 30Mb submission limit, not the reconstruction algorithm itself.
Layer 2
Layer 1
Layer 3
Layer 4
Layer 5
Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top activations in a network and our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form. The compression artifacts are a consequence of the 30Mb submission limit, not the reconstruction algorithm itself.
Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top activations for a random subset of features from the validation set, projected down to pixel space using our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form. The compression artifacts are a consequence of the 30Mb submission limit, not the reconstruction algorithm itself.
Processing Images

- **Input**: grayscale image $32 \times 32$ pixels
- **Output**: layer of $32 \times 32$ features
- **How many parameters do we need when input and output is fully connected?**
Processing Images

- Input: grayscale image $32 \times 32$ pixels
- Output: layer of $32 \times 32$ features
- How many parameters do we need when input and output is fully connected?

$$32^2 \times \left( \frac{32^2}{\text{inputs}} + 1 \right) \approx 1\text{M}$$
Locally Connected Layer

- Motivation: topographical mapping in the visual cortex - nearby cells process nearby regions in the visual field
- Each neuron has a **receptive field** of $3 \times 3$ pixels
- It is fully connected only to the corresponding set of 9 inputs
- How many parameters do we need now?
Locally Connected Layer

- Motivation: topographical mapping in the visual cortex - nearby cells process nearby regions in the visual field
- Each neuron has a **receptive field** of $3 \times 3$ pixels
- It is fully connected only to the corresponding set of 9 inputs
- How many parameters do we need now?
  \[
  30^2 \times (3^2 + 1) = 9k
  \]
Multiple Input Channels

- We can have more input channels, e.g., colors, depth map, . . .
- Now the input is defined by width, height and depth: $32 \times 32 \times 3$
- The number of parameters is $30^2 \times (3 \times 3^2 + 1) \approx 25k$
Sharing Parameters

- We can further reduce the number of parameters by sharing weights.
- Use the same set of weights and bias for all outputs, define a *filter*.
- The number of parameters drops to \(3 \times 3^2 + 1 = 28\) inputs + bias.
- Translation *equivariance*.
Multiple Output Channels

- Extract multiple different of features
- Use multiple filters to get more feature maps
- For 4 filters we have \(4 \times \left(3 \times 3^2 + 1\right) = 112\) parameters
- This is the convolutional layer
- Processes volume into volume
# Convolution Applied to an Image

<table>
<thead>
<tr>
<th>Identity</th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Identity Image" /></th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Identity Image" /></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Sharpen</th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Sharpen Image" /></th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Sharpen Image" /></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Box blur (normalized)</th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Image" /></th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel" /></td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel Image" /></td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel Image" /></td>
</tr>
<tr>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel" /></td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel Image" /></td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel Image" /></td>
</tr>
<tr>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel" /></td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel Image" /></td>
<td><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Box blur (normalized) Kernel Image" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gaussian blur (approximation)</th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Gaussian blur (approximation) Image" /></th>
<th><img src="https://en.wikipedia.org/wiki/Kernel_(image_processing)" alt="Gaussian blur (approximation) Image" /></th>
</tr>
</thead>
</table>

https://en.wikipedia.org/wiki/Kernel_(image_processing)
Convolution in 2D: Forward Message

\[ z_{kld} = f_{kld}(\mathbf{x}, \mathbf{w}, \mathbf{b}) = b_d + \sum_{i=1}^{F} \sum_{j=1}^{F} \sum_{c=1}^{C} x_{k+i-1, l+j-1, c} w_{ijcd} \]
Stride

- Stride hyper parameter, typically $S \in \{1, 2\}$
- Higher stride produces smaller output volumes spatially
**Stride**

- Stride hyper parameter, typically $S \in \{1, 2\}$
- Higher stride produces smaller output volumes spatially
Zero Padding

- Convolutional layer reduces the spatial size of the output w.r.t. the input.
- For many layers this might be a problem.
- This is often fixed by zero padding the input.
- The size of the zero padding is denoted $P$.

$$P = 1, \quad S = 1$$

![Diagram showing zero padding](image)
Convolutional Layer Summary

- **Input volume:** $W_{\text{input}} \times H_{\text{input}} \times C$

- **Output volume:** $W_{\text{output}} \times H_{\text{output}} \times D$

- **Having $D$ filters:**
  - receptive field of $F \times F$ units,
  - stride $S$
  - zero padding $P$

\[
W_{\text{output}} = (W_{\text{input}} - F + 2P)/S + 1 \\
H_{\text{output}} = (H_{\text{input}} - F + 2P)/S + 1
\]

- **Needs $F^2CD$ weights and $D$ biases**

- **The number of activations and $\delta$s to store:** $W_{\text{output}} \times H_{\text{output}} \times D$
Convolutional Layer: Nonlinearities

- In most cases a nonlinearity (sigmoid, tanh, ReLU) is applied to the outputs of the convolutional layer

- Example: ReLU units

**Diagram:**
- **Input feature map**
  - Black = negative; white = positive values
- **Output feature map**
  - Only non-negative values
Max Pooling

- Reduces spatial resolution → less parameters → helps with overfitting
- Introduces translation invariance and invariance to small rotations
- Depth is not affected

\[ F = 2, S = 2 \]
Convolutional Neural Networks (CNNs)
Simonyan, Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014

- Lowering filter spatial resolution ($F = 3$, $S = 1$, $P = 1$), increasing depth
- A sequence of $3 \times 3$ filters can emulate a single large one
- Top five error 7.3%, 6.8% for an ensemble of 2 CNNs
Convolutional vs. Fully-Connected Layers

- Convolutional layer can be simply transformed to a Fully-connected layer → sparse weight matrix

- The other direction is also possible:
  FC layer of $D$ units following a $F \times F \times C$ convolutional layer can be replaced by a $1 \times 1 \times D$ convolutional layer using $F \times F$ filters ($P = 0$, $S = 1$)

- In both cases you do not have to recompute the weights, you just rearrange them
Fully-Connected Layer to Convolutional Example

input

CONV, MP layers

224 × 224 × 3

7 × 7 × 512

4096

1000

softmax
Fully-Connected Layer to Convolutional Example

input

CONV, MP layers

224 × 224 × 3

7 × 7 × 512

1 × 1 × 4096

1 × 1 × 4096

1 × 1 × 1000

F = 7

F = 1

softmax

F = 1
Fully-Connected Layer to Convolutional Example

input

CONV, MP layers

F = 7
F = 1
F = 1

384 ⇥ 384 ⇥ 3
12 ⇥ 12 ⇥ 512
6 ⇥ 6 ⇥ 4096
6 ⇥ 6 ⇥ 4096
6 ⇥ 6 ⇥ 1000
softmax

6 × 6 × 1000
F = 1
Transfer Learning

- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization
Transfer Learning

- Idea: use an existing model as a base to solve a similar problem
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization
Transfer Learning

- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization
Transfer Learning

- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization
- You can:
  - cut the original network at various layers,
  - fix or not the weights of the original network or use different learning rates
  - use different type of model instead of the output layers, e.g., linear SVM
Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top activations in a neural network model and the corresponding image patches. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form. The compression artifacts are a consequence of the 30Mb submission limit, not the reconstruction algorithm itself.
Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a neuron and the corresponding feature map across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form. The compression artifacts are a consequence of the 30Mb submission limit, not the reconstruction algorithm itself.
Fig. 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation set, projected down to pixel space using our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form. The compression artifacts are a consequence of the 30Mb submission limit, not the reconstruction algorithm itself.
<table>
<thead>
<tr>
<th>Identity</th>
<th><img src="image1.png" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identity</strong></td>
<td></td>
</tr>
<tr>
<td><img src="image2.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image3.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image4.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image5.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image6.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image7.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image8.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image9.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image10.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image11.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image12.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image13.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image14.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image15.png" alt="Image" /></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge detection</th>
<th><img src="image16.png" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge detection</strong></td>
<td></td>
</tr>
<tr>
<td><img src="image17.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image18.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image19.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image20.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image21.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image22.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image23.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image24.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><img src="image25.png" alt="Image" /></td>
<td></td>
</tr>
</tbody>
</table>

### Identity Matrix

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

### Edge Detection Matrices

1. **Sobel Filter for X Direction**

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

2. **Sobel Filter for Y Direction**

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

3. **Prewitt Filter for X Direction**

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

4. **Prewitt Filter for Y Direction**

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]
<table>
<thead>
<tr>
<th>Method</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharpen</td>
<td>$\begin{bmatrix} 0 &amp; -1 &amp; 0 \ -1 &amp; 5 &amp; -1 \ 0 &amp; -1 &amp; 0 \end{bmatrix}$</td>
</tr>
<tr>
<td>Box blur (normalized)</td>
<td>$\frac{1}{9} \begin{bmatrix} 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>Gaussian blur (approximation)</td>
<td>$\frac{1}{16} \begin{bmatrix} 1 &amp; 2 &amp; 1 \ 2 &amp; 4 &amp; 2 \ 1 &amp; 2 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>
$S = 1$

$S = 2$
$S = 1$

$S = 2$
$P = 1, S = 1$
Non-Linearity

Rectified linear function
– Applied per-pixel
– output = max(0, input)

Input feature map
Black = negative; white = positive values

Output feature map
Only non-negative values
\[ F = 2, \quad S = 2 \]

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

\[
\begin{array}{ccc}
2 & 5 & 4 \\
5 & 5 & 4 \\
3 & 4 & 5 \\
\end{array}
\]
input

CONV, MP layers

224 \times 224 \times 3

softmax

input

CONV, MP layers

224 \times 224 \times 3

1000

7 \times 7 \times 512

FC

4096

FC

4096

FC

softmax

1000
CONV, MP layers

224 × 224 × 3

7 × 7 × 512

1 × 1 × 4096

F = 1

1 × 1 × 4096

F = 1

1 × 1 × 1000

F = 1

softmax
CONV, MP layers

input

384 × 384 × 3

CONV, MP

12 × 12 × 512

6 × 6 × 4096

6 × 6 × 4096

6 × 6 × 1000

softmax

F = 7

F = 1

F = 1