Neural Nets for Visual Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it

- Why is it difficult?
Why is it a Problem?

- Difficult scene conditions

[From: Grauman & Leibe]
Why is it a Problem?

- Huge within-class variations. Recognition is mainly about modeling variation.

[Pic from: S. Lazebnik]
Tons of classes

~10,000 to 30,000

[Biederman]
People are very good at recognizing objects
- Intrinsically difficult, computers are bad at it

Some reasons why it is difficult:
- **Segmentation**: Real scenes are cluttered
- **Invariances**: We are very good at ignoring all sorts of variations that do not affect class
- **Deformations**: Natural object classes allow variations (faces, letters, chairs)
- A huge amount of computation is required
How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $x$ is very high dimensional
- How many parameters do I have?
- Prohibitive to have fully-connected layers
- What can we do?
- We can use a locally connected layer
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
When Will this Work?

- This is good when the input is (roughly) registered
General Images

- The object can be anywhere
The object can be anywhere

[Slide: Y. Zhu]
The object can be anywhere
The replicated feature approach

Adopt approach apparently used in monkey visual systems

Use many different copies of the same feature detector.
  - Copies have slightly different positions.
  - Could also replicate across scale and orientation.
    - Tricky and expensive
  - Replication reduces the number of free parameters to be learned.

Use several different feature types, each with its own replicated pool of detectors.
  - Allows each patch of image to be represented in several ways.
Idea: statistics are similar at different locations (Lecun 1998)

Connect each hidden unit to a small input patch and share the weight across space

This is called a **convolution layer** and the network is a **convolutional network**

Share the same parameters across different locations (assuming input is stationary):
**Convolutions with learned kernels**
Convolution layers are named after the convolution operation.

If $a$ and $b$ are two (possibly infinite) 1-D arrays, $a \ast b$ is another 1-D array:

$$(a \ast b)_t = \sum_{\tau} a_{\tau} b_{t-\tau}.$$ 

Can think of $a$ as a signal living on a one dimensional line

Normally $a$ finite so $a_t = 0$ for $t \not\in \{1, \ldots, d\}$
Convolution

“Flip and Filter” interpretation:

\[
\begin{array}{c}
\begin{array}{c}
2 \uparrow \\
-1 \downarrow \\
\end{array} & \ast & \begin{array}{c}
1 \uparrow \\
1 \downarrow \\
2 \uparrow \\
\end{array} & = & \begin{array}{c}
4 \uparrow \\
2 \downarrow \\
2 \uparrow \\
\end{array} \\
\end{array}
\]

...
2-D Convolution

2-D convolution is analogous:

\[(A \ast B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}.\]
2-D Convolution

The thing we convolve by is called a kernel, or filter.

What does this convolution kernel do?
What does this convolution kernel do?

\[
\begin{array}{ccc}
0 & -1 & 0 \\
-1 & 8 & -1 \\
0 & -1 & 0 \\
\end{array}
\]
What does this convolution kernel do?

\[
\begin{array}{ccc}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0 \\
\end{array}
\]
2-D Convolution

What does this convolution kernel do?

\[
\begin{array}{c}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{array}
\]
Convolutional Layer

Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolutional Filter

Convolving one filter $F$ with an image $I$ of size $C \times H \times W$ yields an activation map $A$ of size $H' \times W'$

$$I \ast F = A$$

- $H'$ and $W'$ depend on:
  - the stride: how many units apart do we apply a filter spatially
  - the size of the filter
  - These are hyperparameters!
We convolve with many filters and stack the resulting activation maps depthwise.

- This will be the “image” we convolve over in the next layer.

- This operation is called a **convolutional layer**.

- The number of filters in a layer is a hyperparameter!
Pooling

Figure: Left: Pooling, right: max pooling example

By pooling filter responses at different locations we gain robustness to the exact spatial location of our features

Hyperparameters of a pooling layer:

- The spatial extent $F$
- The stride

[http://cs231n.github.io/convolutional-networks/]
Pooling Options

- Max Pooling: return the maximal argument
- Average Pooling: return the average of the arguments
- Other types of pooling exist.
The backprop procedure from last lecture can be applied directly to conv nets.

This is covered in csc421.

As a user, you don’t need to worry about the details, since they’re handled by automatic differentiation packages.
Here’s the LeNet architecture, which was applied to handwritten digit recognition on MNIST in 1998:
Imagenet, biggest dataset for object classification: http://image-net.org/

1000 classes, 1.2M training images, 150K for test
AlexNet

- AlexNet, 2012. 8 weight layers. 16.4% top-5 error (i.e. the network gets 5 tries to guess the right category).
- Closest competitor: 26.1%

(Krizhevsky et al., 2012)

The two processing pathways correspond to 2 GPUs. (At the time, the network couldn’t fit on one GPU.)

AlexNet’s stunning performance on the ILSVRC is what set off the deep learning boom of the last 6 years.
Networks are now at 150 layers

They use a skip connections with special form

In fact, they don’t fit on this screen

Amazing performance!

A lot of “mistakes” are due to wrong ground-truth

\[
F(x) \rightarrow \text{relu} \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow H(x) = F(x) + x \rightarrow \text{identity} \rightarrow x
\]

Results: Object Classification

ImageNet Classification top-5 error (%)

- ILSVRC'15: ResNet, 3.57%
- ILSVRC'14: GoogleNet, 6.7%
- ILSVRC'14: VGG, 7.3%
- ILSVRC'13: 11.7%
- ILSVRC'12: AlexNet, 16.4%
- ILSVRC'11: Shallow, 25.8%
- ILSVRC'10: Shallow, 28.2%

Revolution of Depth

152 layers

Results: Object Detection

Revolution of Depth

Engines of visual recognition

HOG, DPM

AlexNet (RCNN)

VGG (RCNN)

ResNet (Faster RCNN)*

34

58

66

101 layers

86

shallow

8 layers

16 layers

Results: Object Detection

Results: Object Detection

Results: Object Detection

What do CNNs Learn?

Figure: Filters in the first convolutional layer of Krizhevsky et al
What do CNNs Learn?

Figure: Filters in the second layer

What do CNNs Learn?

Figure: Filters in the third layer

What do CNNs Learn?

Great course dedicated to NN: http://cs231n.stanford.edu

Open source frameworks:
  ▶ Pytorch http://pytorch.org/
  ▶ Tensorflow https://www.tensorflow.org/
  ▶ Caffe http://caffe.berkeleyvision.org/

Most cited NN papers:
https://github.com/terryum/awesome-deep-learning-papers