CSC2515 Lecture 6: Convolutional Networks

Roger Grosse

University of Toronto
Midterm Test

- Wednesday, Oct. 30, from 4:10-5:40pm
- Health Sciences building, room 610
- Covers all lectures up through Lecture 6 (i.e. this one)
- You’re only responsible for things covered in lecture, but we might ask harder questions about things you got to practice in homeworks and tutorials.
- Practice tests will be posted on the course web site.
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]
Why is it a Problem?

- Tons of classes

~10,000 to 30,000

[Biederman]
Neural Nets for Object Recognition

- 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
The object can be anywhere

[Slide: Y. Zhu]
The object can be anywhere
The object can be anywhere
The Invariance Problem

- Our perceptual systems are very good at dealing with *invariances*
  - translation, rotation, scaling
  - deformation, contrast, lighting

- We are so good at this that it’s hard to appreciate how difficult it is
  - It’s one of the main difficulties in making computers perceive
  - We still don’t have generally accepted solutions
Locally Connected Layer

**STATIONARITY?** Statistics is similar at different locations

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).

Ranzato
The replicated feature approach

The red connections all have the same weight.

- 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.
Convolutional Neural Net

- 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

Ranzato
Convolution

- Convolution layers are named after the convolution operation.
- If $a$ and $b$ are two arrays,

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

Method 1: translate-and-scale

\[
\begin{array}{c}
\begin{array}{c}
2 \quad -1 \quad 1 \\
\downarrow \quad \downarrow \\
\end{array} \\
\times \\
\begin{array}{c}
1 \\
\downarrow \\
\end{array} \\
\end{array}
\begin{array}{c}
= \\
+ \\
+ \\
\end{array}
\begin{array}{c}
1 \\
\downarrow \\
\end{array}
\begin{array}{c}
1 \\
\downarrow \\
\end{array}
\begin{array}{c}
1 \\
\downarrow \\
\end{array}
\begin{array}{c}
1 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\end{array}
\begin{array}{c}
= \\
+ \\
= \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
1 \\
\downarrow \\
\end{array}
\begin{array}{c}
1 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
4 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
2 \\
\downarrow \\
\end{array}
\begin{array}{c}
-1 \\
\downarrow \\
\end{array}
\end{array}
\]
Method 2: flip-and-filter
Convolution can also be viewed as matrix multiplication:

\[
(2, -1, 1) * (1, 1, 2) = \begin{pmatrix}
1 & 1 \\
2 & 1 & 1 \\
2 & 1 \\
2 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
2 \\
-1 \\
1
\end{pmatrix}
\]

*Aside:* This is how convolution is typically implemented. (More efficient than the fast Fourier transform (FFT) for modern conv nets on GPUs!)
Some properties of convolution:

- **Commutativity**
  \[ a \ast b = b \ast a \]

- **Linearity**
  \[ a \ast (\lambda_1 b + \lambda_2 c) = \lambda_1 a \ast b + \lambda_2 a \ast c \]
2-D convolution is defined analogously to 1-D convolution.

If $A$ and $B$ are two 2-D arrays, then:

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

Method 1: Translate-and-Scale

\[
\begin{array}{c}
\begin{array}{ccc}
1 & 3 & 1 \\
0 & -1 & 1 \\
2 & 2 & -1 \\
\end{array}
\end{array} \ast \begin{array}{c}
\begin{array}{c}
1 \\
2 \\
0 \\
\end{array}
\end{array} = 1 \times \begin{array}{c}
\begin{array}{ccc}
1 & 3 & 1 \\
0 & -1 & 1 \\
2 & 2 & -1 \\
\end{array}
\end{array} + 2 \times \begin{array}{c}
\begin{array}{ccc}
1 & 3 & 1 \\
0 & -1 & 1 \\
2 & 2 & -1 \\
\end{array}
\end{array} = \begin{array}{c}
\begin{array}{cccc}
1 & 5 & 7 & 2 \\
0 & -2 & -4 & 1 \\
2 & 6 & 4 & -3 \\
0 & -2 & -2 & 1 \\
\end{array}
\end{array} + (-1) \times \begin{array}{c}
\begin{array}{ccc}
1 & 3 & 1 \\
0 & -1 & 1 \\
2 & 2 & -1 \\
\end{array}
\end{array}
\]
2-D Convolution

Method 2: Flip-and-Filter

\[
\begin{array}{ccc}
1 & 3 & 1 \\
0 & -1 & 1 \\
2 & 2 & -1 \\
\end{array} \quad \ast \quad \begin{array}{cc}
1 & 2 \\
0 & -1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 & 1 \\
0 & -1 & 1 \\
2 & 2 & -1 \\
\end{array} \times \begin{array}{ccc}
-1 & 0 \\
2 & 1 \\
\end{array} \quad \begin{array}{cccc}
1 & 5 & 7 & 2 \\
0 & -2 & -4 & 1 \\
2 & 6 & 4 & -3 \\
0 & -2 & -2 & 1 \\
\end{array}
\]
2-D Convolution

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

What does this filter do?
2-D Convolution

What does this filter do?

\[
\begin{array}{ccc}
0 & -1 & 0 \\
-1 & 8 & -1 \\
0 & -1 & 0 \\
\end{array}
\]
2-D Convolution

What does this filter do?

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

What does this filter do?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

![Image of a person]

![Image of a filtered face]
Learn multiple filters.

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

Hyperparameters of a convolutional layer:

- The number of filters (controls the **depth** of the output volume)
- The **stride**: how many units apart do we apply a filter spatially (this controls the spatial size of the output volume)
- The size $w \times h$ of the filters

Figure: **Left**: CNN, **right**: Each neuron computes a linear and activation function

[http://cs231n.github.io/convolutional-networks/]
By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Pooling Options

- **Max Pooling**: return the maximal argument
- **Average Pooling**: return the average of the arguments
- Other types of pooling exist.
Pooling

Hyperparameters of a pooling layer:

- The spatial extent $F$
- The stride

[http://cs231n.github.io/convolutional-networks/]
The backprop procedure from last lecture can be applied directly to conv nets.

This is covered in csc2516.

As a user, you don’t need to worry about the details, since they’re handled by automatic differentiation packages.
MNIST Dataset

- MNIST dataset of handwritten digits
  - **Categories:** 10 digit classes
  - **Source:** Scans of handwritten zip codes from envelopes
  - **Size:** 60,000 training images and 10,000 test images, grayscale, of size $28 \times 28$
  - **Normalization:** centered within in the image, scaled to a consistent size
    - The assumption is that the digit recognizer would be part of a larger pipeline that segments and normalizes images.

- In 1998, Yann LeCun and colleagues built a conv net called **LeNet** which was able to classify digits with 98.9% test accuracy.
  - It was good enough to be used in a system for automatically reading numbers on checks.
Here’s the LeNet architecture, which was applied to handwritten digit recognition on MNIST in 1998:
Ways to measure the size of a network:

- **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).
- **Number of weights.** This is important because the weights need to be stored in memory, and because the number of parameters determines the amount of overfitting.
- **Number of connections.** This is important because there are approximately 3 add-multiply operations per connection (1 for the forward pass, 2 for the backward pass).

We saw that a fully connected layer with $M$ input units and $N$ output units has $MN$ connections and $MN$ weights.

The story for conv nets is more complicated.
Size of a Conv Net

I output maps

kernel dimension K

J input maps

height H

width W

fully connected layer

convolution layer

# output units

WHI

# weights

W^2H^2IJ

# connections

W^2H^2IJ

WHI

K^2IJ

WHK^2IJ
## Sizes of layers in LeNet:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th># units</th>
<th># connections</th>
<th># weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>convolution</td>
<td>4704</td>
<td>117,600</td>
<td>150</td>
</tr>
<tr>
<td>S2</td>
<td>pooling</td>
<td>1176</td>
<td>4704</td>
<td>0</td>
</tr>
<tr>
<td>C3</td>
<td>convolution</td>
<td>1600</td>
<td>240,000</td>
<td>2400</td>
</tr>
<tr>
<td>S4</td>
<td>pooling</td>
<td>400</td>
<td>1600</td>
<td>0</td>
</tr>
<tr>
<td>F5</td>
<td>fully connected</td>
<td>120</td>
<td>48,000</td>
<td>48,000</td>
</tr>
<tr>
<td>F6</td>
<td>fully connected</td>
<td>84</td>
<td>10,080</td>
<td>10,080</td>
</tr>
<tr>
<td>output</td>
<td>fully connected</td>
<td>10</td>
<td>840</td>
<td>840</td>
</tr>
</tbody>
</table>

**Conclusions?**
Size of a Conv Net

- Rules of thumb:
  - Most of the units and connections are in the convolution layers.
  - Most of the weights are in the fully connected layers.
- If you try to make layers larger, you’ll run up against various resource limitations (i.e. computation time, memory)
- You’ll repeat this exercise for AlexNet for homework.
  - Conv nets have gotten a LOT larger since 1998!
ImageNet is the modern object recognition benchmark dataset. It was introduced in 2009, and has led to amazing progress in object recognition since then.
ImageNet

- Used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual benchmark competition for object recognition algorithms

- Design decisions

  - **Categories**: Taken from a lexical database called WordNet
    - WordNet consists of “synsets”, or sets of synonymous words
    - They tried to use as many of these as possible; almost 22,000 as of 2010
    - Of these, they chose the 1000 most common for the ILSVRC
    - The categories are really specific, e.g. hundreds of kinds of dogs

  - **Size**: 1.2 million full-sized images for the ILSVRC

  - **Source**: Results from image search engines, hand-labeled by Mechanical Turkers

    - Labeling such specific categories was challenging; annotators had to be given the WordNet hierarchy, Wikipedia, etc.

  - **Normalization**: none, although the contestants are free to do preprocessing
Images and object categories vary on a lot of dimensions

Russakovsky et al.
ImageNet

Size on disk:

MNIST
60 MB

ImageNet
50 GB
AlexNet

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

- 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.
Inception, 2014. ("We need to go deeper!")

22 weight layers

Fully convolutional (no fully connected layers)

Convolutions are broken down into a bunch of smaller convolutions

6.6% test error on ImageNet

(Szegedy et al., 2014)
They were really aggressive about cutting the number of parameters.

- Motivation: train the network on a large cluster, run it on a cell phone
  - Memory at test time is the big constraint.
  - Having lots of units is OK, since the activations only need to be stored at training time (for backpropagation).
  - Parameters need to be stored both at training and test time, so these are the memory bottleneck.

- How they did it
  - No fully connected layers (remember, these have most of the weights)
  - Break down convolutions into multiple smaller convolutions (since this requires fewer parameters total)

- Inception has “only” 2 million parameters, compared with 60 million for AlexNet
- This turned out to improve generalization as well. (Overfitting can still be a problem, even with over a million images!)
150 Layers!

- 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

\[ F(x) \]

\[ H(x) = F(x) + x \]

Results: Object Classification

Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

3.57
6.7
7.3
11.7
16.4
25.8
28.2

152 layers
22 layers
19 layers
8 layers
8 layers
shallow

Results: Object Detection

Revolution of Depth

Engines of visual recognition

HOG, DPM
AlexNet (RCNN)
VGG (RCNN)
ResNet (Faster RCNN)*

34
58
66
86
shallow
8 layers
16 layers
101 layers

Results: Object Detection

Results: Object Detection


person: 0.989
refrigerator: 0.979
knife: 0.739
bottle: 0.79623
oven: 0.969
bowl: 0.927
cup: 0.881
spoon: 0.727381
bowl: 0.969
cup: 0.971
Results: Object Detection

What Do Networks Learn?

- Recall: we can understand what first-layer features are doing by visualizing the weight matrices.

![Fully connected (MNIST)](image)

![Convolutional (ImageNet)](image)

- Higher-level weight matrices are hard to interpret.
- The better the input matches these weights, the more the feature activates.
  - Obvious generalization: visualize higher-level features by seeing what inputs activate them.
One way to formalize: pick the images in the training set which activate a unit most strongly.

Here’s the visualization for layer 1:
Layer 3:
What Do Networks Learn?

- Layer 4:
What Do Networks Learn?

- Layer 5:
Higher layers seem to pick up more abstract, high-level information.

Problems?

- Can’t tell what the unit is actually responding to in the image.
- We may read too much into the results, e.g. a unit may detect red, and the images that maximize its activation will all be stop signs.

Can use input gradients to diagnose what the unit is responding to.

- Optimize an image from scratch to increase a unit’s activation
Recall the computation graph:

From this graph, you could compute $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$, but we never made use of this.
Optimizing the Image

- Can do gradient ascent on an image to maximize the activation of a given neuron.
- Requires a few tricks to make this work; see https://distill.pub/2017/feature-visualization/

Starting from random noise, we optimize an image to activate a particular neuron (layer mixed4a, unit 11).

Step 0  →  Step 4  →  Step 48  →  Step 2048
**Dataset Examples** show us what neurons respond to in practice.

**Optimization** isolates the causes of behavior from mere correlations. A neuron may not be detecting what you initially thought.

Baseball—or stripes?  
*mixed4a, Unit 6*

Animal faces—or snouts?  
*mixed4a, Unit 240*

Clouds—or fluffiness?  
*mixed4a, Unit 453*

Buildings—or sky?  
*mixed4a, Unit 492*
Optimizing the Image

- Higher layers in the network often learn higher-level, more interpretable representations.

https://distill.pub/2017/feature-visualization/
Higher layers in the network often learn higher-level, more interpretable representations.