Lecture 14:
– Deep Convolutional Networks
Administrative

• **Assignment 3** is due November 30, 2016!

• **Progress reports** are approaching
  – due December 12, 2016!
Last time... Three key ideas

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations

- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extract

- Distributed Representations
  - No single neuron “encodes” everything
  - Groups of neurons work together
Last time… Intro. to Deep Learning

VISION

SIFT/HOG → K-Means/pooling → classifier → “car”

fixed    unsupervised    supervised

SPEECH

MFCC → Mixture of Gaussians → classifier → “d è p”

fixed    unsupervised    supervised

NLP

This burrito place is yummy and fun! → Parse Tree Syntactic → n-grams → classifier → “+”

fixed    unsupervised    supervised

slide by Marc'Aurelio Ranzato, Yann LeCun
Last time... Intro. to Deep Learning

- “Shallow” models

- Deep models
Last time… ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

---

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input Size</th>
<th>Filters</th>
<th>Output Size</th>
<th>Conv &amp; ReLU Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>32</td>
<td></td>
<td>28</td>
<td>CONV, ReLU e.g. 6 5x5x3 filters</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>24</td>
<td>24</td>
<td>CONV, ReLU e.g. 10 5x5x6 filters</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td></td>
<td></td>
<td>CONV, ReLU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⋮</td>
</tr>
</tbody>
</table>

---

Side by Fei-Fei Li, Andrej Karpathy & Justin Johnson
E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume
Last time… Convolutional Neural Networks
Case studies
Case Study: LeNet-5  [LeCun et al., 1998]

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2 
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: \((227-11)/4+1 = 55\)
Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume $[55x55x96]$

Q: What is the total number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Parameters: (11*11*3)*96 = 35K
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: \((55-3)/2+1 = 27\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96

...
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

- **INPUT**: [227x227x3]
- **CONV1**: 96 11x11 filters at stride 4, pad 0
- **MAX POOL1**: 3x3 filters at stride 2
- **NORM1**: Normalization layer
- **CONV2**: 256 5x5 filters at stride 1, pad 2
- **MAX POOL2**: 3x3 filters at stride 2
- **NORM2**: Normalization layer
- **CONV3**: 384 3x3 filters at stride 1, pad 1
- **CONV4**: 384 3x3 filters at stride 1, pad 1
- **CONV5**: 256 3x3 filters at stride 1, pad 1
- **MAX POOL3**: 3x3 filters at stride 2
- **FC6**: 4096 neurons
- **FC7**: 4096 neurons
- **FC8**: 1000 neurons (class scores)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%
Case Study: ZFNet [Zeiler and Fergus, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013

\[ \Rightarrow \]

7.3% top 5 error

---

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>11 weight layers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input (224 x 224 RGB image)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
</tbody>
</table>

| FC-4096 | FC-4096 | FC-1000 | soft-max |

---

Table 2: Number of parameters (in millions).

<table>
<thead>
<tr>
<th>Network</th>
<th>A, A-\LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of parameters</td>
<td>133</td>
<td>133</td>
<td>134</td>
<td>138</td>
<td>144</td>
</tr>
</tbody>
</table>
INPUT: [224x224x3]  memory: 224*224*3=150K  params: 0

CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*64)*64 = 36,864

POOL2: [112x112x64]  memory: 112*112*64=800K  params: 0

CONV3-128: [112x112x128]  memory: 112*112*128=1.6M  params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory: 112*112*128=1.6M  params: (3*3*128)*128 = 147,456

POOL2: [56x56x128]  memory: 56*56*128=400K  params: 0

CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*256)*256 = 589,824

POOL2: [28x28x256]  memory: 28*28*256=200K  params: 0

CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512]  memory: 14*14*512=100K  params: 0

CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512]  memory: 7*7*512=25K  params: 0

FC: [1x1x4096]  memory: 4096  params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory: 4096  params: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory: 1000  params: 4096*1000 = 4,096,000

(not counting biases)
INPUT: [224x224x3] memory: 224*224*3=150K params: 0

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
INPUT: [224x224x3]  memory: 224x224x3=150K  params: 0

CONV3-64: [224x224x64] memory: 224x224x64=3.2M  params: (3x3x3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224x224x64=3.2M  params: (3x3x64)*64 = 36,864

POOL2: [112x112x64] memory: 112x112x64=800K  params: 0

CONV3-128: [112x112x128] memory: 112x112x128=1.6M  params: (3x3x128)*128 = 147,456

CONV3-128: [112x112x128] memory: 112x112x128=1.6M  params: (3x3x128)*128 = 147,456

POOL2: [56x56x128] memory: 56x56x128=400K  params: 0

CONV3-256: [56x56x256] memory: 56x56x256=800K  params: (3x3x128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56x56x256=800K  params: (3x3x256)*128 = 589,824

CONV3-256: [56x56x256] memory: 56x56x256=800K  params: (3x3x256)*256 = 589,824

POOL2: [28x28x256] memory: 28x28x256=200K  params: 0

CONV3-512: [28x28x512] memory: 28x28x512=400K  params: (3x3x256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28x28x512=400K  params: (3x3x512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28x28x512=400K  params: (3x3x512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14x14x512=100K  params: 0

CONV3-512: [14x14x512] memory: 14x14x512=100K  params: (3x3x512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14x14x512=100K  params: (3x3x512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14x14x512=100K  params: (3x3x512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7x7x512=25K  params: 0

FC: [1x1x4096] memory: 4096  params: 7x7x512x4096 = 102,760,448

FC: [1x1x4096] memory: 4096  params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000  params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~ 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Note:
Most memory is in early CONV
Most params are in late FC

(not counting biases)
Case Study: GoogLeNet [Szegedy et al., 2014]

ILSVRC 2014 winner (6.7% top 5 error)

Inception module
Case Study: ResNet [He et al., 2015]
ILSVRC 2015 winner (3.6% top 5 error)

Slide from Kaiming He’s recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w
Case Study: ResNet \[\text{He et al., 2015}\]
ILSVRC 2015 winner
(3.6% top 5 error)

Revolution of Depth

- AlexNet, 8 layers (ILSVRC 2012)
- VGG, 19 layers (ILSVRC 2014)
- ResNet, 152 layers (ILSVRC 2015)

2-3 weeks of training on 8 GPU machine
at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He’s recent presentation)
Case Study: ResNet [He et al., 2015]

34-layer plain

34-layer residual

224x224x3

spatial dimension only 56x56!
Case Study Bonus: DeepMind’s AlphaGo
The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a $23 \times 23$ image, then convolves $k$ filters of kernel size $5 \times 5$ with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a $21 \times 21$ image, then convolves $k$ filters of kernel size $3 \times 3$ with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size $1 \times 1$ with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128, 256$ and 384 filters.

**Policy network:**

[19x19x48] Input
CONV1: 192 5x5 filters, stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] (probability map of promising moves)
Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like

\[(\text{CONV-RELU})*N-\text{POOL}?]*M-(\text{FC-RELU})*K,\text{SOFTMAX}\]

where N is usually up to \(~5\), M is large, \(0 \leq K \leq 2\).

- but recent advances such as ResNet/GoogLeNet challenge this paradigm
Understanding ConvNets
Input Image

RGB Input Image
224 x 224 x 3

96 filters
7x7x3 Convolution
3x3 Max Pooling
Down Sample 4x
55 x 55 x 96

256 filters
5x5x96 Convolution
3x3 Max Pooling
Down Sample 4x
13 x 13 x 256

354 filters
3x3x256 Convolution
13 x 13 x 354

256 filters
3x3x354 Convolution
3x3 Max Pooling
Down Sample 2x
6 x 6 x 256

354 filters
3x3x354 Convolution
13 x 13 x 354

Logistic Regression
≈1000 Classes

Standard Units
4096 Units

Standard Units
4096 Units

3x3x354 Convolution
3x3 Max Pooling
Down Sample 2x
13 x 13 x 354

Visualizing CNN (Layer 1)

Visualizing CNN (Layer 2)

Part that Triggered Filter

Top Image Patches

Visualizing CNN (Layer 3)

Part that Triggered Filter

Top Image Patches

Visualizing CNN (Layer 4)

Part that Triggered Filter

Top Image Patches

Visualizing CNN (Layer 5)

Part that Triggered Filter

Top Image Patches

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis

Jason Yosinski  Jeff Clune  Anh Nguyen  Thomas Fuchs  Hod Lipson

Cornell University  University of Wyoming  NASA  Jet Propulsion Laboratory
Tips and Tricks
• Shuffle the training samples

• Use Dropout and Batch Normalization for regularization
Input representation

• Centered (0-mean) RGB values.

“Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central $256 \times 256$ patch from the resulting image”
Data Augmentation

- Our neural net has 60M real-valued parameters and 650,000 neurons.
- It overfits a lot. Therefore, they train on 224x224 patches extracted randomly from 256x256 images, and also their horizontal reflections.

“This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter-dependent.”

[Krizhevsky et al. 2012]
Data Augmentation

• Alter the intensities of the RGB channels in training images.

“Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1…This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.”

[Krizhevsky et al. 2012]
Data Augmentation

Horizontal flips
Data Augmentation

Get creative!

Random mix/combinaisons of:
- translation
- rotation
- stretching
- shearing,
- lens distortions, ... (go crazy)
Data augmentation improves human learning, not just deep learning

If you're trying to improve your golf swing or master that tricky guitar chord progression, here's some good news from researchers at Johns Hopkins University: You may be able to double how quickly you learn skills like these by introducing subtle variations into your practice routine.

The received wisdom on learning motor skills goes something like this: You need to build up "muscle memory" in order to perform mechanical tasks, like playing musical instruments or sports, quickly and efficiently. And the way you do that is via rote repetition — return hundreds of tennis serves, play that F major scale over and over until your fingers bleed, etc.

The wisdom on this isn't necessarily wrong, but the Hopkins research suggests it's incomplete. Rather than doing the same thing over and over, you might be able to learn things even faster — like, twice as fast — if you change up your routine. Practicing your baseball swing? Change the size and weight of your bat. Trying to nail a 12-bar blues in A major on the guitar? Spend 20 minutes playing the blues in E major, too. Practice your backhand using tennis rackets of varying size and weight.

Transfer Learning with ConvNets

1. Train on Imagenet
Transfer Learning with ConvNets

1. Train on Imagenet

2. Small dataset:
   - feature extractor
   - Freeze these

Train this
Transfer Learning with ConvNets

1. Train on Imagenet

2. Small dataset: feature extractor
   - Freeze these
   - Train this

3. Medium dataset: finetuning
   - more data = retrain more of the network (or all of it)
   - Freeze these
   - Train this
Transfer Learning with ConvNets

1. Train on Imagenet

2. Small dataset: feature extractor
   - Freeze these
   - Train this

3. Medium dataset: finetuning
   - more data = retrain more of the network (or all of it)
   - Freeze these
   - tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers
   - Train this
Today ConvNets are everywhere

Classification

Retrieval

[Krizhevsky 2012]
Today ConvNets are everywhere

Detection

Segmentation

[Faster R-CNN: Ren, He, Girshick, Sun 2015]  [Farabet et al., 2012]
Today ConvNets are everywhere

self-driving cars

NVIDIA Tegra X1
Today ConvNets are everywhere

[Simonyan et al. 2014]

[Goodfellow 2014]
Today ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]
Today ConvNets are everywhere

[Ciresan et al. 2013]

[Sermanet et al. 2011]

[Ciresan et al.]
Today ConvNets are everywhere

[Turaga et al., 2010]

[Denil et al. 2014]
Today ConvNets are everywhere

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010
Today ConvNets are everywhere

[Image Captioning]

[Vinyals et al., 2015]
Today ConvNets are everywhere
Frameworks

• Caffe [http://caffe.berkeleyvision.org/](http://caffe.berkeleyvision.org/) 
  Efficient for convolutional models / images

• Torch [http://torch.ch/](http://torch.ch/) 
  Very efficient. But you must LIKE Lua … 
  Google and Facebook love it

• Theano [http://deeplearning.net/software/theano/](http://deeplearning.net/software/theano/) 
  Compiled from Python. Not as efficient as Torch

• Minerva [https://github.com/dmlc/minerva](https://github.com/dmlc/minerva) 
  Compiler layout of execution on machines

• CXXNet [https://github.com/dmlc/cxxnet](https://github.com/dmlc/cxxnet) 
  Simpler than Caffe. More efficient

• Parameter Server bindings to [https://github.com/dmlc/minerva, Caffe, CXXNet, …](https://github.com/dmlc/minerva)