Administrative

- In A1 we caught a small number of questionable cases of similar code. Keep the Honor Code in mind.

- Moving Project milestone to Monday Feb 16 (was Friday Feb 13 before)

- Terminal lets you use faster than \texttt{xlarge}, but don’t use them. OpenBLAS is not configured optimally and you aren’t seeing any improvements, just burning cash. We are fixing this for A3.
Lecture 9:

What makes ConvNets tick & Transfer Learning
Assignment #1 Extra Credit: Cool submissions

- **Enhao**: HOG+Color Histogram+Spatial Pyramid Pooling+PCA, Hierarchical hyperparameter search
- **Amani**: A full Bag-of-words model, and implementation of all SVMs (One vs. All, One vs. One, Structured SVM)
- **Yilun**: LBP features, got 45%, nice!
- **Charles**: Confusion matrix
- **Rohit**: Prewitt filtering preprocessing (improves 43% -> 53%)
- **Pranav**: Similar idea with sobel filters, 50%
- **Vignesh**: Polynomial features \([x, x^{**2}, x^{**3}]\), boost to 51%.
- **John**: patch k-means features, 51%
- **Chris**: Nice CV plots
- **Albert**: F/B segmentation features, 53%
Chain your gradients…

forward pass:

\[
\begin{align*}
    s_1 &= W_1 x + b_1 \\
    a_1 &= \max(0, s_1) \\
    s_2 &= W_2 a_1 + b_2 \\
    L &= f(s_2)
\end{align*}
\]

softmax loss
Chain your gradients…

**forward pass:**
\[ s_1 = W_1 x + b_1 \]
\[ a_1 = \max(0, s_1) \]
\[ s_2 = W_2 a_1 + b_2 \]
\[ L = f(s_2) \]

**backward pass:**
\[ ds_2 = \ldots \]
\[ dW_2 = \ldots \]
\[ db_2 = \ldots \]
\[ da_1 = \ldots \]
\[ ds_1 = \ldots \]
\[ dW_1 = \ldots \]
\[ db_1 = \ldots \]

---

softmax loss

short one-liners that implement small piece of chain rule:

local gradient * chained gradient
Chain your gradients...

forward pass:

\[
\begin{align*}
    s_1 &= W_1 x + b_1 \\
    a_1 &= \max(0, s_1) \\
    s_2 &= W_2 a_1 + b_2 \\
    L &= f(s_2)
\end{align*}
\]

softmax loss

backward pass:

\[
\begin{align*}
    ds_2 &= \ldots \\
    dW_2 &= \ldots \\
    db_2 &= \ldots \\
    da_1 &= \ldots \\
    ds_1 &= \ldots \\
    dW_1 &= \ldots \\
    db_1 &= \ldots
\end{align*}
\]

very similar to assignment #1

short one-liners that implement small piece of chain rule:

local gradient * chained gradient
Where we are...
Question: When does CNN work well and when does it not?
ImageNet (ILSVRC competition) analysis

1. Detecting avocados to zucchinis: what have we done, and where are we going?
2. ImageNet Large Scale Visual Recognition Challenge
[Olga Russakovsky et al.]
(Amount of texture)
Image classification

Easiest classes
- red fox (100)
- hen-of-the-woods (100)
- ibex (100)
- goldfinch (100)
- flat-coated retriever (100)
- tiger (100)
- hamster (100)
- porcupine (100)
- stingray (100)
- Blenheim spaniel (100)

Hardest classes
- muzzle (71)
- hatchet (68)
- water bottle (68)
- velvet (68)
- loupe (66)
- hook (66)
- spotlight (66)
- ladle (65)
- restaurant (64)
- letter opener (59)
CNN vs. Human

[What I learned from competing against a ConvNet on ImageNet]

Try it out yourself: http://cs.stanford.edu/people/karpathy/ilsvrc/
:(
GoogLeNet: 6.7%
Team Human: 5.1% phew...
What makes a ConvNet tick?
Understanding the source of ConvNet performance

Visualizing and Understanding Convolutional Networks
[Zeiler and Fergus, 2013]

<table>
<thead>
<tr>
<th>Error %</th>
<th>Train Top-1</th>
<th>Val Top-1</th>
<th>Val Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our replication of (Krizhevsky et al., 2012), 1 convnet</td>
<td>35.1</td>
<td>40.5</td>
<td>18.1</td>
</tr>
<tr>
<td>Removed layers 3,4</td>
<td>41.8</td>
<td>45.4</td>
<td>22.1</td>
</tr>
<tr>
<td>Removed layer 7</td>
<td>27.4</td>
<td>40.0</td>
<td>18.4</td>
</tr>
<tr>
<td>Removed layers 6,7</td>
<td>27.4</td>
<td>44.8</td>
<td>22.4</td>
</tr>
<tr>
<td>Removed layer 3,4,6,7</td>
<td>71.1</td>
<td>71.3</td>
<td>50.1</td>
</tr>
<tr>
<td>Adjust layers 6,7: 2048 units</td>
<td>40.3</td>
<td>41.7</td>
<td>18.8</td>
</tr>
<tr>
<td>Adjust layers 6,7: 8192 units</td>
<td>26.8</td>
<td>40.0</td>
<td>18.1</td>
</tr>
</tbody>
</table>
Understanding the source of ConvNet performance
Visualizing and Understanding Convolutional Networks
[Zeiler and Fergus, 2013]

- Remove 2 FC layers (6,7): lose some small performance
- Remove 2 Conv layers (3,4): lose about equal performance
- Remove 2FC 2Conv (3,4,6,7): Very bad (71% error)

=> Depth is important

<table>
<thead>
<tr>
<th>Error %</th>
<th>Train Top-1</th>
<th>Val Top-1</th>
<th>Val Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our replication of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Krizhevsky et al., 2012), 1 convnet</td>
<td>35.1</td>
<td>40.5</td>
<td>18.1</td>
</tr>
<tr>
<td>Removed layers 3,4</td>
<td>41.8</td>
<td>45.4</td>
<td>22.1</td>
</tr>
<tr>
<td>Removed layer 7</td>
<td>27.4</td>
<td>40.0</td>
<td>18.4</td>
</tr>
<tr>
<td>Removed layers 6,7</td>
<td>27.4</td>
<td>44.8</td>
<td>22.4</td>
</tr>
<tr>
<td>Removed layers 3,4,6,7</td>
<td>71.1</td>
<td>71.3</td>
<td>50.1</td>
</tr>
<tr>
<td>Adjust layers 6,7: 2048 units</td>
<td>40.3</td>
<td>41.7</td>
<td>18.8</td>
</tr>
<tr>
<td>Adjust layers 6,7: 8192 units</td>
<td>26.8</td>
<td>40.0</td>
<td>18.1</td>
</tr>
</tbody>
</table>
Understanding the source of ConvNet performance

Visualizing and Understanding Convolutional Networks
[Zeiler and Fergus, 2013]

- Remove 2 FC layers (6,7): lose some small performance
- Remove 2 Conv layers (3,4): lose about equal performance
- Remove 2FC 2Conv (3,4,6,7): Very bad (71% error)

=> Depth is important
- Changing size of FC layers: little to no improvement
- Changing size of Conv layers: reasonable improvement!

<table>
<thead>
<tr>
<th>Error %</th>
<th>Train Top-1</th>
<th>Val Top-1</th>
<th>Val Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our replication of (Krizhevsky et al., 2012), 1 convnet</td>
<td>35.1</td>
<td>40.5</td>
<td>18.1</td>
</tr>
<tr>
<td>Removed layers 3,4</td>
<td>41.8</td>
<td>45.4</td>
<td>22.1</td>
</tr>
<tr>
<td>Removed layer 7</td>
<td>27.4</td>
<td>40.0</td>
<td>18.4</td>
</tr>
<tr>
<td>Removed layers 6,7</td>
<td>27.4</td>
<td>44.8</td>
<td>22.4</td>
</tr>
<tr>
<td>Removed layer 3,4,6,7</td>
<td>71.1</td>
<td>71.3</td>
<td>50.1</td>
</tr>
<tr>
<td>Adjust layers 6,7: 2048 units</td>
<td>40.3</td>
<td>41.7</td>
<td>18.8</td>
</tr>
<tr>
<td>Adjust layers 6,7: 8192 units</td>
<td>26.8</td>
<td>40.0</td>
<td>18.1</td>
</tr>
</tbody>
</table>

Our Model (as per Fig. 3)
- Adjust layers 6,7: 2048 units | 33.1 | 38.4 | 16.5 |
- Adjust layers 6,7: 8192 units | 38.2 | 40.2 | 17.6 |
- Adjust layers 3,4,5: 512,1024,512 maps | 22.0 | 38.8 | 17.0 |
- Adjust layers 6,7: 8192 units and Layers 3,4,5: 512,1024,512 maps | 18.8 | 37.5 | 16.0 |
- Adjust layers 6,7: 8192 units and Layers 3,4,5: 512,1024,512 maps | 10.0 | 38.3 | 16.9 |
Return of the Devil in the Details: Delving Deep into Convolutional Nets [Chatfield et al. 2014]

(Finetuning on PASCAL VOC 2007)

<table>
<thead>
<tr>
<th></th>
<th>number of neurons in last FC layer</th>
<th>mAP (high = good)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p)</td>
<td>CNN M</td>
<td>(C)</td>
</tr>
<tr>
<td>(x)</td>
<td>CNN M 2048</td>
<td>(C)</td>
</tr>
<tr>
<td>(y)</td>
<td>CNN M 1024</td>
<td>(C)</td>
</tr>
<tr>
<td>(z)</td>
<td>CNN M 128</td>
<td>(C)</td>
</tr>
</tbody>
</table>

surprisingly small drop for 128 dimensions
Normalization doesn’t do anything

<table>
<thead>
<tr>
<th>ConvNet config. (Table 1)</th>
<th>smallest image side</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>256</td>
<td>29.6</td>
<td>10.4</td>
</tr>
<tr>
<td>A-LRN</td>
<td>256</td>
<td>29.7</td>
<td>10.5</td>
</tr>
<tr>
<td>B</td>
<td>256</td>
<td>28.7</td>
<td>9.9</td>
</tr>
<tr>
<td>C</td>
<td>256</td>
<td>28.1</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>28.1</td>
<td>9.3</td>
</tr>
<tr>
<td></td>
<td>[256;512]</td>
<td>27.3</td>
<td>8.8</td>
</tr>
<tr>
<td>D</td>
<td>256</td>
<td>27.0</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>26.8</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td>[256;512]</td>
<td>25.6</td>
<td>8.1</td>
</tr>
<tr>
<td>E</td>
<td>256</td>
<td>27.3</td>
<td>9.0</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>26.9</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td>[256;512]</td>
<td>25.5</td>
<td>8.0</td>
</tr>
</tbody>
</table>

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>
More depth = better performance

well almost, looks like D with 16 weight layers is a sweet spot
[Going deeper with convolutions, Szegedy et al., 2014]

**GoogLeNet**
12x less params than Krizhevsky et al.
=> ~5M params

Q: How to reduce the number of parameters?
**GoogLeNet**

12x less params than Krizhevsky et al.

=> ~5M params

Q: How to reduce the number of parameters?
A: Throw away the FC layers (only part of their answer (Inception modules))

After last pooling layer, volume is of size \[7 \times 7 \times 1024\]

Normally you would place the first 4096-D FC layer here (Many M params)

Instead: use Average pooling in each depth slice:

=> \[1 \times 1 \times 1024\]

performance actually improves 0.6% (less overfitting?)
[Going deeper with convolutions, Szegedy et al., 2014]

**GoogLeNet**

Fun feature: multiple Softmaxes along the way
(Very experimental!)
Summary: What makes ConvNets tick?

- depth
- small filter sizes
- Conv layers > FC layers
“You need a lot of data if you want to train/use CNNs”
Transfer Learning

“You need a lot of data if you want to train/use CNNs”
Transfer Learning with CNNs

1. Train on Imagenet
Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end
Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier
   i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, “finetune” instead: use the old weights as initialization, train the full network or only some of the higher layers
   retrain bigger portion of the network, or even all of it.
Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier
   i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, “finetune” instead: use the old weights as initialization, train the full network or only some of the higher layers

   retrain bigger portion of the network, or even all of it.

   tip: use only ~1/10th of the original learning rate in finetuning to player, and ~1/100th on intermediate layers
CNN Features off-the-shelf: an Astounding Baseline for Recognition
[Razavian et al, 2014]
CNN Features off-the-shelf: an Astounding Baseline for Recognition
[Razavian et al, 2014]

DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition
[Donahue*, Jia*, et al., 2013]

<table>
<thead>
<tr>
<th></th>
<th>DeCAF_0</th>
<th>DeCAF_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogReg</td>
<td>40.94 ± 0.3</td>
<td>40.84 ± 0.3</td>
</tr>
<tr>
<td>SVM</td>
<td>39.36 ± 0.3</td>
<td>40.66 ± 0.3</td>
</tr>
</tbody>
</table>

Xiao et al. (2010) 38.0
Visualizing and Understanding Convolutional Networks
[Zeiler and Fergus, 2013]

<table>
<thead>
<tr>
<th>Layer</th>
<th>Cal-101 (30/class)</th>
<th>Cal-256 (60/class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM (1)</td>
<td>44.8 ± 0.7</td>
<td>24.6 ± 0.4</td>
</tr>
<tr>
<td>SVM (2)</td>
<td>66.2 ± 0.5</td>
<td>39.6 ± 0.3</td>
</tr>
<tr>
<td>SVM (3)</td>
<td>72.3 ± 0.4</td>
<td>46.0 ± 0.3</td>
</tr>
<tr>
<td>SVM (4)</td>
<td>76.6 ± 0.4</td>
<td>51.3 ± 0.1</td>
</tr>
<tr>
<td>SVM (5)</td>
<td>86.2 ± 0.8</td>
<td>65.6 ± 0.3</td>
</tr>
<tr>
<td>SVM (7)</td>
<td>85.5 ± 0.4</td>
<td>71.7 ± 0.2</td>
</tr>
<tr>
<td>Softmax (5)</td>
<td>82.9 ± 0.4</td>
<td>65.7 ± 0.5</td>
</tr>
<tr>
<td>Softmax (7)</td>
<td>85.4 ± 0.4</td>
<td>72.6 ± 0.1</td>
</tr>
</tbody>
</table>

Q: If we were to use only one layer, which layer should we transfer from?
How transferable are features in deep neural networks? [Yosinski et al., 2014]

Split ImageNet classes in half to two sets: A/B.

Train on A, fix the first $n$ layers, reinit layers $n+$, train on B, test on B val.

$=>$ performance degrades because representation higher up is too A-specific
How transferable are features in deep neural networks?  
[Yosinski et al., 2014]

Split ImageNet classes in half to two sets: A/B.

Train on A, reinit layers n+, train on B, test on B val.

=> the information from once seeing data from A seems to linger, gives better generalization
How transferable are features in deep neural networks?

[Yosinski et al., 2014]

Split ImageNet classes in half to two sets: A/B.

Train on B, fix the first n layers, re-initialize layers n+, train on B again and test on B val

=> performance degrades when n = 4. wat.
How transferable are features in deep neural networks? [Yosinski et al., 2014]

Split ImageNet classes in half to two sets: A/B.

Train on B, reinitialize layers n+, train on B again and test on B val.

=> performance doesn’t degrade anymore
<table>
<thead>
<tr>
<th></th>
<th>More Generic</th>
<th>More Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dataset</strong></td>
<td>Very Similar Dataset</td>
<td>Very Different Dataset</td>
</tr>
<tr>
<td>Very Little Data</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Quite a Lot of Data</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>more generic</td>
<td>more specific</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>conv-64</td>
<td>conv-64</td>
<td></td>
</tr>
<tr>
<td>conv-64</td>
<td>conv-64</td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
</tr>
<tr>
<td>conv-128</td>
<td>conv-128</td>
<td></td>
</tr>
<tr>
<td>conv-128</td>
<td>conv-128</td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
</tr>
<tr>
<td>conv-256</td>
<td>conv-256</td>
<td></td>
</tr>
<tr>
<td>conv-256</td>
<td>conv-256</td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
</tr>
<tr>
<td>conv-512</td>
<td>conv-512</td>
<td></td>
</tr>
<tr>
<td>conv-512</td>
<td>conv-512</td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-4096</td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-4096</td>
<td></td>
</tr>
<tr>
<td>FC-1000</td>
<td>softmax</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>Use Linear Classifier on top layer</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>Finetune a few layers</td>
</tr>
<tr>
<td>more generic</td>
<td>more specific</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>very little data</strong></td>
<td><strong>very similar dataset</strong></td>
</tr>
<tr>
<td>Use Linear Classifier on top layer</td>
<td>You’re in trouble… Try linear classifier from different stages</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td><strong>very different dataset</strong></td>
</tr>
<tr>
<td>Finetune a few layers</td>
<td>Finetune a larger number of layers</td>
</tr>
</tbody>
</table>
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

pretrained with ImageNet, as is everything else

word representations pretrained from word2vec
Takeaway for your projects/beyond:
Have some dataset of interest but it has $< ~1M$ images?

1. Find a very large dataset that has similar data, train a big ConvNet there.
2. Transfer learn to your dataset

Caffe ConvNet library has a “Model Zoo” of pretrained models:
https://github.com/BVLC/caffe/wiki/Model-Zoo
Summary:
- A lot of CNN’s power comes from middle Conv layers, and depth seems to be important.
- Transfer Learning can be very helpful with small-medium data
Next Lecture:

CNN Tips/Tricks: squeezing out the last few percent