Administrivia

• If you need a team-mate for your project, email me. I’m matching people.
Previously – Backpropagation

\[ f(x) = (-x + 3)^2 \]

Forward pass: compute function
Backward pass: compute derivative of all parts of the function
Setting Up A Neural Net

Input | Hidden | Output

\[ x_1 \] | \[ h_1 \] | \[ y_1 \]

\[ x_2 \] | \[ h_2 \] | \[ y_2 \]

\[ h_3 \] | \[ h_4 \] | \[ y_3 \]
Setting Up A Neural Net

Input  Hidden 1  Hidden 2  Output
Each neuron connects to each neuron in the previous layer
Define New Block: “Linear Layer”

(Ok technically it’s Affine)

\[ L(n) = Wn + b \]

Can get gradient with respect to all the inputs
Convolutional Layer

New Block: 2D Convoluiton

\[ C(n) = n \times W + b \]
Convolution Layer

\[ b + \sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^{c} F_{i,j,k} \ast I_{y+i,x+j,c} \]
Convolutional Neural Network (CNN)
Convert HxW image into a F-dimensional vector

- What’s the probability this image is a cat (F=1)
- Which of 1000 categories is this image? (F=1000)
- At what GPS coord was this image taken? (F=2)
- Identify the X,Y coordinates of 28 body joints of an image of a human (F=56)
Today’s Running Example: Classification

Running example: image classification

\[
P(\text{image is class \#1})
\]
\[
P(\text{image is class \#2})
\]
\[
P(\text{image is class \#F})
\]
Today’s Running Example: Classification

Loss function

\[- \log \left( \frac{\exp((Wx)_y)}{\sum_k \exp((Wx)_k)} \right)\]

“Hippo”
Today’s Running Example: Classification

“Baboon”

Loss function

\[-\log \left( \frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)} \right)\]
Model For Your Head

- Provide:
  - Examples of images and desired outputs
  - Sequence of layers producing a 1x1xF output
  - A loss function that measures success
- Train the network -> network figures out the parameters that makes this work
Layer Collection

You can construct functions out of layers. The only requirement is the layers “fit” together. Optimization figures out what the parameters of the layers are.

Image credit: lego.com
Review – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel

Max-pool 2x2 Filter Stride 2

Slide credit: Karpathy and Fei-Fei
Review – Pooling

Max-pool
2x2 Filter
Stride 2
Other Layers – Fully Connected

1x1xC → 1x1xF

Map C-dimensional feature to F-dimensional feature using linear transformation 
$W_{FxC}$ matrix + $b_{Fx1}$ vector

How can we write this as a convolution?
Everything’s a Convolution

1x1xC → 1x1xF

Set $F_h=1$, $F_w=1$

1x1 Convolution with F Filters

\[ b + \sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^{c} F_{i,j,k} \ast I_{y+i,x+j,c} \rightarrow b + \sum_{k=1}^{c} F_k \ast I_c \]
Converting to a Vector

HxWxC  \rightarrow  1x1xF

How can we do this?
Converting to a Vector* – Pool

HxWxC → 1x1xF

Avg Pool
HxW Filter
Stride 1

3.1

*(If F == C)
Converting to a Vector – Convolve

HxWxC  \rightarrow  1x1xF

HxW Convolution with F Filters

Single value Per-filter
Looking At Networks

• We’ll look at 3 landmark networks, each trained to solve a 1000-way classification output (Imagenet)
  • Alexnet (2012)
  • VGG-16 (2014)
  • Resnet (2015)
AlexNet

<table>
<thead>
<tr>
<th>Layer</th>
<th>Shape</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>227x227x3</td>
<td>1</td>
</tr>
<tr>
<td>Conv 1</td>
<td>55x55x96</td>
<td>1</td>
</tr>
<tr>
<td>Conv 2</td>
<td>27x27x256</td>
<td>1</td>
</tr>
<tr>
<td>Conv 3</td>
<td>13x13x384</td>
<td>1</td>
</tr>
<tr>
<td>Conv 4</td>
<td>13x13x384</td>
<td>1</td>
</tr>
<tr>
<td>Conv 5</td>
<td>13x13x256</td>
<td>1</td>
</tr>
<tr>
<td>FC 6</td>
<td>1x1x4096</td>
<td>1</td>
</tr>
<tr>
<td>FC 7</td>
<td>1x1x4096</td>
<td>1</td>
</tr>
<tr>
<td>Output</td>
<td>1x1x1000</td>
<td>1</td>
</tr>
</tbody>
</table>

Each block is a HxWxC volume.
You transform one volume to another with convolution.
### CNN Terminology

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

Each entry is called an “activation”/“neuron”/“feature”
## AlexNet

<table>
<thead>
<tr>
<th></th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
</tr>
</tbody>
</table>
AlexNet

Input: 227x227, 3

Conv 1:
- 55x55, 96

227x227 -> 55x55

ReLU

11x11 filter, stride of 4
(227-11)/4+1 = 55
AlexNet

Input | Conv 1 | Conv 2 | Conv 3 | Conv 4 | Conv 5 | FC 6 | FC 7 | Output
--- | --- | --- | --- | --- | --- | --- | --- | ---
227x227 | 55x55 | 27x27 | 13x13 | 13x13 | 13x13 | 1x1 | 1x1 | 1x1
3 | 96 | 256 | 384 | 384 | 256 | 4096 | 4096 | 1000

All layers followed by ReLU

Red layers are followed by maxpool

Early layers have “normalization”
<table>
<thead>
<tr>
<th>Layer</th>
<th>Input Size</th>
<th>C:</th>
<th>P:</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>227x227x3</td>
<td>C:11</td>
<td>P:3</td>
<td>C: 1x1</td>
</tr>
<tr>
<td>Conv 1</td>
<td>55x55x96</td>
<td>C:5</td>
<td>P:3</td>
<td>C: 1x1</td>
</tr>
<tr>
<td>Conv 2</td>
<td>27x27x256</td>
<td>C:3</td>
<td></td>
<td>C: 1x1</td>
</tr>
<tr>
<td>Conv 3</td>
<td>13x13x384</td>
<td>C:3</td>
<td></td>
<td>C: 1x1</td>
</tr>
<tr>
<td>Conv 4</td>
<td>13x13x384</td>
<td>C:3</td>
<td></td>
<td>C: 1x1</td>
</tr>
<tr>
<td>Conv 5</td>
<td>13x13x256</td>
<td>C:3</td>
<td></td>
<td>C: 1x1</td>
</tr>
<tr>
<td>FC 6</td>
<td>1x1x4096</td>
<td></td>
<td></td>
<td>C: 1x1</td>
</tr>
<tr>
<td>FC 7</td>
<td>1x1x4096</td>
<td></td>
<td></td>
<td>C: 1x1</td>
</tr>
<tr>
<td>Output</td>
<td>1x1x1000</td>
<td></td>
<td></td>
<td>C: 1x1</td>
</tr>
</tbody>
</table>

C: Size of conv
P: Size of pool
**AlexNet**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

13x13 Input, 1x1 output. How?
## Alexnet – How Many Parameters?

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>
Alexnet – How Many Parameters?

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

- **96 11x11** filters on 3-channel input
- **11x11x3x96+96 = 34,944**
**Alexnet – How Many Parameters?**

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

Note: max pool to 6x6.

4096 6x6 filters on 256-channel input

6x6x256x4096+4096 = 38 million
Alexnet – How Many Parameters?

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

4096 1x1 filters on 4096-channel input

1x1x4096x4096+4096 = 17 million
Alexnet – How Many Parameters

How long would it take you to list the parameters of Alexnet at 4s / parameter?

1 year? 4 years? 8 years? 16 years?

- 62.4 million parameters
- Vast majority in fully connected layers
- But... paper notes that removing the convolutions is disastrous for performance.
Dataset – ILSVRC

• Imagenet LargeScale Visual Recognition Challenge
• 1.4M images
• 1000 Categories, often ridiculously precise
Dataset – ILSVRC

Figure Credit: O. Russakovsky
Visualizing Filters

Conv 1 Filters

- Q. How many input dimensions?
  - A: 3

- What does the input mean?
What’s Learned

First layer filters of a network trained to distinguish 1000 categories of objects

Remember these filters go over color.

Figure Credit: Karpathy and Fei-Fei
## Visualizing Later Filters

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
</tr>
</tbody>
</table>

### Conv 2 Filters

- **Q. How many input dimensions?**
- **A: 96…. hmmm**
- **What does the input mean?**
- **Uh, the uh, previous slide**
Visualizing Later Filters

• Understanding the meaning of the later filters \textit{from their values} is typically impossible: too many input dimensions, not even clear what the input means.
Understanding Later Filters

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

CNN that extracts a 13x13x256 output

2-hidden layer Neural network
Understanding Later Filters

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>256</td>
<td>384</td>
<td>384</td>
<td>256</td>
<td>4096</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

CNN that extracts a 1x1x4096 feature

1-hidden layer NN
Understanding Later Filters

CNN that extracts a 13x13x256 output
Understanding Later Filters

Feed an image in, see what score the filter gives it. A more pleasant version of a real neuroscience procedure.

Which one’s bigger? What image makes the output biggest?
Figure Credit: Girshick et al. CVPR 2014.
What's Up With the White Boxes?
Due to convolution, each later layer's value depends on / "sees" only a fraction of the input image.
Can use receptive fields to see where the network is “looking” to make its decisions

A very active area of research (lots of great work done by Bolei Zhou, MIT → CUHK)

Classic Recognition

Input

227x227
3
Recall: can compute a descriptor based on histograms of image gradients. Do it densely (at each pixel).
Can do bag-of-words-like techniques on SIFT, taking into consideration spatial location.
Classic Recognition

Input: 227x227, 3

SIFT: 227x227, 128

Bag of Words: HxW, #codewords

Output: 1x1, 1000

Dense SIFT (a few layers) -> BOW -> Classifier
Classic Recognition

Input

227x227
3

SIFT

227x227
128

Bag of Words

HxW
#codewords

Output

1x1
1000

Dense SIFT (a few layers)

BOW

Classifier
Classic vs Deep Recognition

Classic
Pipeline of hand-engineered steps

Deep
Pipeline of learned convolutions + simple operations

What are some differences?
The classic steps don’t talk to each other or have many parameters that are learned from data.
<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Size</th>
<th>Stride</th>
<th>Filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>227x227</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Conv 1</td>
<td>55x55</td>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>Conv 2</td>
<td>27x27</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>Conv 3</td>
<td>13x13</td>
<td>1</td>
<td>384</td>
</tr>
<tr>
<td>Conv 4</td>
<td>13x13</td>
<td>1</td>
<td>384</td>
</tr>
<tr>
<td>Conv 5</td>
<td>13x13</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>FC 6</td>
<td>1x1</td>
<td>1</td>
<td>4096</td>
</tr>
<tr>
<td>FC 7</td>
<td>1x1</td>
<td>1</td>
<td>4096</td>
</tr>
<tr>
<td>Output</td>
<td>1x1</td>
<td>1</td>
<td>1000</td>
</tr>
</tbody>
</table>
3 Key Developments Since Alexnet

• 3x3 Filters
• Batch Normalization
• Residual Learning
Key Idea – 3x3 Filters

3x3 filter followed by 3x3 filter

→

Filter with 5x5 receptive field
Key Idea – 3x3 Filters

3x3 filter followed by 3x3 filter followed by 3x3 filter

→ Filter with 7x7 receptive field
Why Does This Make A Difference?

Empirically, repeated 3x3 filters do better compared to a 7x7 filter.

Why?
Key Idea – 3x3 Filters

Receptive Field: 7x7 pixels
Parameters/channel: 49
Number of ReLUs: 1

Receptive Field: 7x7 pixels
Parameters/channel: 3x3x3=27
Number of ReLUs: 3
We Want More Non-linearity!

Can they implement xor?

No

Yes
VGG16

All filters 3x3
All filters followed by ReLU
Training Deeper Networks

Why not just stack continuously?

What will happen to gradient going back?
Backprop

Every backpropagation step multiplies the gradient by the local gradient

\[ 1 \times d \times d \times d \ldots \times d = d^{n-1} \]

What if \( d \ll 1 \), \( n \) big?

Vanishing Gradients
Backprop

Every backpropagation step multiplies the gradient by the local gradient

\[ 1 \times d \times d \times d \ldots \times d = d^{n-1} \]

What if \( d \gg 1 \), \( n \) big?

Exploding Gradients
Solution 1 – Batch Normalization

Learning algorithms work far better when data looks like the right as opposed to the left.

Mean(x) \neq Mean(Y) \neq 0
Var(x) \neq Var(y) \neq 0
Cov(x,y) \neq 0

Mean(x) = Mean(Y) = 0
Var(x) = Var(y) = 1
Cov(x,y) = 0
Solution 1 – Batch Normalization

Data

$\text{Mean}(x) = \text{Mean}(Y) = 0$
$\text{Var}(x) = \text{Var}(y) = 1$

Idea: make layer (Batch Norm) that normalizes things going through it based on estimates of $\text{Var}(x_i)$ in each batch. Stick in between other layers

There exists vs. We Can Find

- Still can’t fit models to the data: **Deeper model** fits worse than **shallower model** on the training data.
- There exists a deeper model that’s identical to the shallow model. Why?

Residual Learning

New Building Block: \( x + F(x) \)

Lets you train networks with 100s of layers.
Evaluating Results

At training time, we minimize: \(-\log\left(\frac{\exp((Wx)_{yi}}{\sum_k \exp((Wx)_k)})\right)\)

At test time, we evaluate, given predicted class \(\hat{y}_i\):

Accuracy: \(\frac{1}{n} \sum_{i=1}^{n} 1(y_i = \hat{y}_i)\)
Evaluating Many Categories

Does this image depict a cat or a dog?

To avoid penalizing ambiguous images, many challenges let you make five guesses (top-5 accuracy):

Your prediction is correct if one of the guesses is right.

Image credit: Coco dataset
## Accuracy over the Years

<table>
<thead>
<tr>
<th>Model</th>
<th>Top 1 Error</th>
<th>Top 5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Pre-Deep</td>
<td>-</td>
<td>26.2%*</td>
</tr>
<tr>
<td>Alexnet</td>
<td>43.5%</td>
<td>20.9%</td>
</tr>
<tr>
<td>VGG-16</td>
<td>28.4%</td>
<td>9.6%</td>
</tr>
<tr>
<td>VGG-16 + Batch Norm</td>
<td>26.6%</td>
<td>8.5%</td>
</tr>
<tr>
<td>Resnet-152</td>
<td>21.7%</td>
<td>5.9%</td>
</tr>
<tr>
<td>Human*</td>
<td>-</td>
<td>5.1%</td>
</tr>
</tbody>
</table>
A Practical Aside

- People usually use hardware specialized for matrix multiplies (the card below does 13.4T flops if it’s matrix multiplies).
- The real answer to why we love homogeneous coordinates?
  - Makes rendering matrix multiplies →
  - leads to matrix multiplication hardware →
  - deep learning.
Training a CNN

- Download a big dataset
- Initialize network weights randomly
- for epoch in range(epochs):
  - Shuffle dataset
  - for each minibatch in dataset:
    - Put data on GPU
    - Compute gradient
    - Update gradient with SGD
Training a CNN from Scratch

Need to start \( \mathbf{w} \) somewhere

- AlexNet: weights \( \sim \text{Normal}(0,0.01) \), bias = 1
- “Xavier” initialization: Uniform\( \left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right) \) where \( n \) is the number of neurons
- “Kaiming” initialization: Normal\( (0,\sqrt{2/n}) \)

Take-home: important, but use defaults
Training a ConvNet

• Convnets typically have millions of parameters:
  • AlexNet: 62 million
  • VGG16: 138 million

• Convnets typically fit on ~1.2 million images

• Remember least squares: if we have fewer data points than parameters, we’re in trouble

• Solution: need regularization / more data
Training a CNN – Weight Decay

SGD Update

\[ w_{t+1} = w_t - \epsilon \frac{\partial L}{\partial w_t} \]

+ Weight Decay

\[ w_{t+1} = w_t - \eta \epsilon w_t + \epsilon \frac{\partial L}{\partial w_t} \]

**What does this remind you of?**

Weight decay is very similar to regularization but might not be the same for more complex optimization techniques.
Quick Quiz

Raise your hand if it’s a hippo

Horizontal Flip
Color Jitter
Image Cropping
Training a CNN – Augmentation

- Apply transformations that don’t affect the output
- Produces more data but you have to be careful that it doesn’t change the meaning of the output
Training a CNN – Fine-tuning

• What if you don’t have data?
Fine-Tuning: Pre-trained Features

1. Extract some layer from an existing network
2. Use as your new feature.
3. Learn a linear model.

Surprisingly effective

Convolutions that extract a 1x1x4096 feature (*Fixed/Frozen/Locked*)

\[ Wx + b \]
Fine-Tuning: Transfer Learning

• Rather than initialize from random weights, initialize from some “pre-trained” model that does something else.
• Most common model is trained on ImageNet.
• Other pretraining tasks exist but are less popular.
Fine-Tuning: Transfer Learning

Why should this work?
Transferring from objects (dog) to scenes (waterfall)

Recommendations

• <10K images: features
• **Always** try fine-tuning
• >100K images: consider trying from scratch
Summary

• We learned about converting an image into a vector output (e.g., which of K classes is this image, or predict K continuous outputs)
• We learned about some building blocks for doing this