Computer Vision
Neural Network:

(Before) Linear score function:

(Now) 2-layer Neural Network

\[ f = W_1 x \]

\[ f = W_2 \max(0, W_1 x) \]
Classification
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
ImageNet

Figure 1: A snapshot of two root-to-leaf branches of ImageNet: the top row is from the mammal subtree; the bottom row is from the vehicle subtree. For each synset, 9 randomly sampled images are presented.
Revolution of Depth

152 layers

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

(slide from Kaiming He’s recent presentation)
Working with CNNs in practice:

- Data augmentation
- Transfer learning
- Autoencoders
Data Augmentation
Classification
Data Augmentation

Load image and label

“cat”

CNN

Compute loss
Data Augmentation

1. Load image and label
2. "cat"
3. Transform image
4. CNN
5. Compute loss
Data Augmentation

- Change the pixels without changing the label
- Train on transformed data
- VERY widely used

What the computer sees
Data Augmentation

1. Horizontal flips
Data Augmentation

2. Random crops/scales

**Training**: sample random crops / scales
Data Augmentation

2. Random crops/scales

**Training:** sample random crops / scales

ResNet:

1. Pick random $L$ in range $[256, 480]$  
2. Resize training image, short side = $L$  
3. Sample random $224 \times 224$ patch
Data Augmentation

2. Random crops/scales

**Training**: sample random crops / scales

ResNet:
1. Pick random $L$ in range $[256, 480]$
2. Resize training image, short side = $L$
3. Sample random $224 \times 224$ patch

**Testing**: average a fixed set of crops
Data Augmentation

2. Random crops/scales

**Training**: sample random crops / scales

ResNet:
1. Pick random $L$ in range $[256, 480]$
2. Resize training image, short side = $L$
3. Sample random 224 x 224 patch

**Testing**: average a fixed set of crops

ResNet:
1. Resize image at 5 scales: $\{224, 256, 384, 480, 640\}$
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips
Data Augmentation

3. Color jitter

**Simple:**
Randomly jitter contrast
Data Augmentation
3. Color jitter

**Simple:**
Randomly jitter contrast

**Complex:**
1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
1. Add offset to all pixels of a training image
   (As seen in [Krizhevsky et al. 2012], ResNet, etc)
Data Augmentation

4. Get creative!

Random mix/combinations of:
- translation
- rotation
- stretching
- shearing,
- lens distortions, ... (go crazy)
Data Augmentation: Takeaway

• Simple to implement, use it
• Especially useful for small datasets
• Fits into framework of noise / marginalization
Transfer Learning

“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. Small dataset: feature extractor

Freeze these

Train this
Transfer Learning with CNNs

1. Train on Imagenet
   - conv-64
   - conv-64
   - maxpool
   - conv-128
   - conv-128
   - maxpool
   - conv-256
   - conv-256
   - maxpool
   - conv-512
   - conv-512
   - maxpool
   - conv-512
   - conv-512
   - maxpool
   - FC-4096
   - FC-4096
   - FC-1000
   - softmax

2. Small dataset: feature extractor
   - conv-64
   - conv-64
   - maxpool
   - conv-128
   - conv-128
   - maxpool
   - conv-256
   - conv-256
   - maxpool
   - conv-512
   - conv-512
   - maxpool
   - conv-512
   - conv-512
   - maxpool
   - FC-4096
   - FC-4096
   - FC-1000
   - softmax

   Freeze these

   Train this

3. Medium dataset: finetuning
   - conv-64
   - conv-64
   - maxpool
   - conv-128
   - conv-128
   - maxpool
   - conv-256
   - conv-256
   - maxpool
   - conv-512
   - conv-512
   - maxpool
   - FC-4096
   - FC-4096
   - FC-1000
   - softmax

   Freeze these

   Train this

more data = retrain more of the network (or all of it)
Transfer Learning with CNNs

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)
   - Freezing these
   - Tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers

Tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers
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.]

<table>
<thead>
<tr>
<th></th>
<th>DeCAF0</th>
<th>DeCAF2</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
<table>
<thead>
<tr>
<th>more generic</th>
<th>more specific</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very similar dataset</strong></td>
<td><strong>very different dataset</strong></td>
</tr>
<tr>
<td>very little data</td>
<td>?</td>
</tr>
<tr>
<td>quite a lot of data</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>very similar dataset</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------</td>
</tr>
<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>
</tbody>
</table>

- More generic models can handle very similar datasets better.
- More specific models can handle very different datasets better.

---

- Linear classifier on top layer.
- Finetune a few layers.
<table>
<thead>
<tr>
<th>More Generic</th>
<th>More Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>More General</td>
<td>More Specific</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>very little data</td>
<td>Use Linear Classifier on top layer</td>
</tr>
<tr>
<td>quite a lot of data</td>
<td>Finetune a few layers</td>
</tr>
</tbody>
</table>
## Overview

<table>
<thead>
<tr>
<th></th>
<th>Caffe</th>
<th>Torch</th>
<th>Theano</th>
<th>TensorFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>C++, Python</td>
<td>Lua</td>
<td>Python</td>
<td>Python</td>
</tr>
<tr>
<td><strong>Pretrained</strong></td>
<td>Yes ++</td>
<td>Yes ++</td>
<td>Yes (Lasagne)</td>
<td>Inception</td>
</tr>
<tr>
<td><strong>Multi-GPU:</strong> Data parallel</td>
<td>Yes</td>
<td>Yes cunn.DataParallelTable</td>
<td>Yes platoon</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Multi-GPU:</strong> Model parallel</td>
<td>No</td>
<td>Yes fbcunn.ModelParallel</td>
<td>Experimental</td>
<td>Yes (best)</td>
</tr>
<tr>
<td><strong>Readable source code</strong></td>
<td>Yes (C++)</td>
<td>Yes (Lua)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Good at RNN</strong></td>
<td>No</td>
<td>Mediocre</td>
<td>Yes</td>
<td>Yes (best)</td>
</tr>
</tbody>
</table>
Supervised vs Unsupervised

Supervised Learning

**Data:** \((x, y)\)

\(x\) is data, \(y\) is label

**Goal:** Learn a *function* to map

\(x \rightarrow y\)

**Examples:** Classification, regression, object detection, semantic segmentation, image captioning, etc
Supervised vs Unsupervised

**Supervised Learning**

**Data:** (x, y)

x is data, y is label

**Goal:** Learn a *function* to map

x -> y

**Examples:** Classification, regression, object detection, etc.

---

**Unsupervised Learning**

**Data:** x

Just data, no labels!

**Goal:** Learn some *structure* of the data

**Examples:** Clustering, dimensionality reduction, feature learning, etc.
Unsupervised Learning

• Autoencoders
  • Traditional: feature learning
Autoencoders

\[\text{Input data} \xrightarrow{\text{Encoder}} \mathbf{z} \xrightarrow{\text{Features}} \mathbf{x}\]
Autoencoders

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN
Autoencoders

$z$ usually smaller than $x$ (dimensionality reduction)

*Originally*: Linear + nonlinearity (sigmoid)
*Later*: Deep, fully-connected
*Later*: ReLU CNN
Autoencoders

Reconstructed input data

Decoder

Features

z

Encoder

Input data

x

Reconstructed input data

Encoder

Input data

z

Decoder

x
Autoencoders

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Encoder: 4-layer conv
Decoder: 4-layer upconv
Autoencoders

Encoder / decoder sometimes share weights

\[ \text{Example:} \]
\[ \dim(x) = D \]
\[ \dim(z) = H \]
\[ w_e: H \times D \]
\[ w_d: D \times H = w_e^T \]

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Train for reconstruction with no labels!
Autoencoders

Input data $X$ -> Encoder $Z$ -> Decoder $X$ -> Reconstructed input data

Loss function (Often L2)

Train for reconstruction with no labels!
Autoencoders

After training, throw away decoder!

Input data $x$ → Encoder $z$ → Features $z$ → Decoder $x$ → Reconstructed input data
Autoencoders

Use encoder to initialize a \textit{supervised} model

Loss function (Softmax, etc)

Predicted Label

\( y \)

Features

\( z \)

Encoder

Input data

\( x \)

Classifier

Fine-tune encoder jointly with classifier

Train for final task (sometimes with small data)

bird  plan
dog  deer  truck
Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
\[
R(W) = \sum_k \sum_l W_{k,l}^2 \\
R(W) = \sum_k \sum_l |W_{k,l}|
\]

\[
\text{minimize} \quad \frac{\lambda}{m} \sum_{i=1}^{m} \left\| W^T W x^{(i)} - x^{(i)} \right\|_2^2
\]

\[
\text{minimize} \quad \frac{\lambda}{m} \sum_{i=1}^{m} \left\| \sigma(W^T \sigma(W x^{(i)} + b) + c) - x^{(i)} \right\|_2^2
\]