Machine Learning – Lecture 15

Convolutional Neural Networks

05.12.2019

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de
Course Outline

• Fundamentals
  ➢ Bayes Decision Theory
  ➢ Probability Density Estimation

• Classification Approaches
  ➢ Linear Discriminants
  ➢ Support Vector Machines
  ➢ Ensemble Methods & Boosting
  ➢ Random Forests

• Deep Learning
  ➢ Foundations
  ➢ Convolutional Neural Networks
  ➢ Recurrent Neural Networks
Topics of This Lecture

• Recap: Tricks of the Trade
  - Initialization
  - Dropout
  - Batch Normalization

• Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

• CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
Recap: Reducing the Learning Rate

- Final improvement step after convergence is reached
  - Reduce learning rate by a factor of 10.
  - Continue training for a few epochs.
  - Do this 1-3 times, then stop training.

- Effect
  - Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches.

- Be careful: *Do not turn down the learning rate too soon!*
  - Further progress will be much slower/impossible after that.

Slide adapted from Geoff Hinton
Recap: Data Augmentation

• Effect
  - Much larger training set
  - Robustness against expected variations

• During testing
  - When cropping was used during training, need to again apply crops to get same image size.
  - Beneficial to also apply flipping during test.
  - Applying several ColorPCA variations can bring another ~1% improvement, but at a significantly increased runtime.
Recap: Normalizing the Inputs

• Convergence is fastest if
  - The mean of each input variable over the training set is zero.
  - The inputs are scaled such that all have the same covariance.
  - Input variables are uncorrelated if possible.

• Advisable normalization steps (for MLPs only, not for CNNs)
  - Normalize all inputs that an input unit sees to zero-mean, unit covariance.
  - If possible, try to decorrelate them using PCA (also known as Karhunen-Loeve expansion).

Recap: Commonly Used Nonlinearities

- **Sigmoid**
  \[ g(a) = \sigma(a) = \frac{1}{1 + \exp\{-a\}} \]

- **Hyperbolic tangent**
  \[ g(a) = \tanh(a) = 2\sigma(2a) - 1 \]

- **Softmax**
  \[ g(a) = \frac{\exp\{-a_i\}}{\sum_j \exp\{-a_j\}} \]
Extension: ReLU

- Another improvement for learning deep models
  - Use Rectified Linear Units (ReLU)
    \[ g(a) = \max\{0, a\} \]
  - Effect: gradient is propagated with a constant factor
    \[ \frac{\partial g(a)}{\partial a} = \begin{cases} 1 , & a > 0 \\ 0 , & \text{else} \end{cases} \]

- Advantages
  - Much easier to propagate gradients through deep networks.
  - We do not need to store the ReLU output separately
    - Reduction of the required memory by half compared to tanh!

\[ \Rightarrow \text{ReLU has become the de-facto standard for deep networks.} \]
Extension: ReLU

• Another improvement for learning deep models
  ➢ Use Rectified Linear Units (ReLU)
    \[ g(a) = \max \{0, a\} \]
  ➢ Effect: gradient is propagated with a constant factor
    \[ \frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases} \]

• Disadvantages / Limitations
  ➢ A certain fraction of units will remain “stuck at zero”.
    – If the initial weights are chosen such that the ReLU output is 0 for the entire training set, the unit will never pass through a gradient to change those weights.
  ➢ ReLU has an offset bias, since its outputs will always be positive
Further Extensions

- **Rectified linear unit (ReLU)**
  \[ g(a) = \max\{0, a\} \]

- **Leaky ReLU**
  \[ g(a) = \max\{\beta a, a\} \]
  - Avoids stuck-at-zero units
  - Weaker offset bias

- **ELU**
  \[ g(a) = \begin{cases} 
  a, & x < 0 \\ 
  e^a - 1, & x \geq 0 
  \end{cases} \]
  - No offset bias anymore
  - BUT: need to store activations
Topics of This Lecture

• Recap: Tricks of the Trade
  - Initialization
  - Dropout
  - Batch Normalization

• Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

• CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
Initializing the Weights

• Motivation
  - The starting values of the weights can have a significant effect on the training process.
  - Weights should be chosen randomly, but in a way that the sigmoid is primarily activated in its linear region.

• Guideline (from [LeCun et al., 1998] book chapter)
  - Assuming that
    - The training set has been normalized
    - The recommended sigmoid \( f(x) = 1.7159 \tanh \left( \frac{2}{3} x \right) \) is used
  the initial weights should be randomly drawn from a distribution (e.g., uniform or Normal) with mean zero and variance
  \[ \sigma_w^2 = \frac{1}{n_{in}} \]
  where \( n_{in} \) is the fan-in (#connections into the node).
Historical Sidenote

- Apparently, this guideline was either little known or misunderstood for a long time
  - A popular heuristic (also the standard in Torch) was to use
    \[ W \sim U \left[ -\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}} \right] \]
    - This looks almost like LeCun’s rule. However…

- When sampling weights from a uniform distribution \([a, b]\)
  - Keep in mind that the standard deviation is computed as
    \[ \sigma^2 = \frac{1}{12} (b - a)^2 \]
  - If we do that for the above formula, we obtain
    \[ \sigma^2 = \frac{1}{12} \left( \frac{2}{\sqrt{n_{in}}} \right)^2 = \frac{1}{3} \frac{1}{n_{in}} \]
    \[ \Rightarrow \text{Activations & gradients will be attenuated with each layer! (bad)} \]
Glorot Initialization

• Breakthrough results
  - In 2010, Xavier Glorot published an analysis of what went wrong in the initialization and derived a more general method for automatic initialization.
  - This new initialization massively improved results and made direct learning of deep networks possible overnight.
  - Let’s look at his analysis in more detail...

Analysis

• Variance of neuron activations

  ➢ Suppose we have an input $X$ with $n$ components and a linear neuron with random weights $W$ that spits out a number $Y$.
  ➢ What is the variance of $Y$?

  $$Y = W_1X_1 + W_2X_2 + \cdots + W_nX_n$$

  ➢ If inputs and outputs have both mean 0, the variance is

  $$Var(W_iX_i) = E[X_i]^2Var(W_i) + E[W_i]^2Var(X_i) + Var(W_i)Var(X_i)$$

  $$= Var(W_i)Var(X_i)$$

  ➢ If the $X_i$ and $W_i$ are all i.i.d, then

  $$Var(Y) = Var(W_1X_1 + W_2X_2 + \cdots + W_nX_n) = nVar(W_i)Var(X_i)$$

  $\Rightarrow$ The variance of the output is the variance of the input, but scaled by $n \ Var(W_i)$. 
Analysis (cont’d)

- Variance of neuron activations
  - if we want the variance of the input and output of a unit to be the same, then \( n \, \text{Var}(W_i) \) should be 1. This means
    \[
    \text{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}
    \]
  - If we do the same for the backpropagated gradient, we get
    \[
    \text{Var}(W_i) = \frac{1}{n_{\text{out}}}
    \]
  - As a compromise, Glorot & Bengio proposed to use
    \[
    \text{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}
    \]
  ⇒ Randomly sample the weights with this variance. That’s it.
Sidenote

• When sampling weights from a uniform distribution \([a, b]\)  
  
  ➢ Again keep in mind that the standard deviation is computed as 
  \[
  \sigma^2 = \frac{1}{12} (b - a)^2
  \]

  ➢ Glorot initialization with uniform distribution
  \[
  W \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right]
  \]

  ➢ Or when only taking into account the fan-in
  \[
  W \sim U \left[-\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}}\right]
  \]

  ➢ If this had been implemented correctly in Torch from the beginning, the Deep Learning revolution might have happened a few years earlier…
Extension to ReLU

• Important for learning deep models
  ➢ Rectified Linear Units (ReLU)
    \[ g(a) = \max\{0, a\} \]
  ➢ Effect: gradient is propagated with a constant factor
    \[ \frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases} \]

• We can also improve them with proper initialization
  ➢ However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
  ➢ He et al. made the derivations, derived to use instead
    \[ \text{Var}(W) = \frac{2}{n_{\text{in}}} \]
Topics of This Lecture

• Recap: Tricks of the Trade
  ➢ Initialization
  ➢ Dropout
  ➢ Batch Normalization

• Convolutional Neural Networks
  ➢ Neural Networks for Computer Vision
  ➢ Convolutional Layers
  ➢ Pooling Layers

• CNN Architectures
  ➢ LeNet
  ➢ AlexNet
  ➢ VGGNet
  ➢ GoogLeNet
Batch Normalization [Ioffe & Szegedy ’14]

- Motivation
  - Optimization works best if all inputs of a layer are normalized.

- Idea
  - Introduce intermediate layer that centers the activations of the previous layer per minibatch.
  - I.e., perform transformations on all activations and undo those transformations when backpropagating gradients
  - Complication: centering + normalization also needs to be done at test time, but minibatches are no longer available at that point.
    - Learn the normalization parameters to compensate for the expected bias of the previous layer (usually a simple moving average)

- Effect
  - Much improved convergence (but parameter values are important!)
  - Widely used in practice
Dropout

[Perceptual and Sensory Augmented Computing, Machine Learning, Winter '19]

[Srivastava, Hinton ’12]

• Idea
  - Randomly switch off units during training (a form of regularization).
  - Change network architecture for each minibatch, effectively training many different variants of the network.
  - When applying the trained network, multiply activations with the probability that the unit was set to zero during training.
  
⇒ Greatly improved performance
Topics of This Lecture

• Recap: Tricks of the Trade

• **Convolutional Neural Networks**
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

• **CNN Architectures**
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
Neural Networks for Computer Vision

• How should we approach vision problems?

  ➢ Input is 2D ➞ 2D layers of units
  ➢ No pre-segmentation ➞ Need robustness to misalignments
  ➢ Vision is hierarchical ➞ Hierarchical multi-layered structure
  ➢ Vision is difficult ➞ Network should be deep

→ Face Y/N?
Why Hierarchical Multi-Layered Models?

• Motivation 1: Visual scenes are hierarchically organized

Object

↑

Object parts

↑

Primitive features

Input image

↑

Face

↑

Eyes, nose, ...

↑

Oriented edges

Face image

Slide adapted from Richard Turner

B. Leibe
Why Hierarchical Multi-Layered Models?

- Motivation 2: *Biological vision* is hierarchical, too

```
Object
  \uparrow
Object parts
  \uparrow
Primitive features
  \uparrow
Input image

Face
  \uparrow
Eyes, nose, ...
  \uparrow
Oriented edges
  \uparrow
Face image

Inferotemporal
cortex

V4: different
textures

V1: simple and
complex cells

Photoreceptors,
retina
```

Slide adapted from Richard Turner

B. Leibe
Hubel/Wiesel Architecture

  - Visual cortex consists of a hierarchy of *simple*, *complex*, and *hyper-complex* cells
Why Hierarchical Multi-Layered Models?

- Motivation 3: Shallow architectures are inefficient at representing complex functions.

An MLP with 1 hidden layer can implement *any* function (universal approximator).

However, if the function is deep, a very large hidden layer may be required.
What’s Wrong With Standard Neural Networks?

• Complexity analysis
  - How many parameters does this network have?
    \[ |\theta| = 3D^2 + D \]
  - For a small $32 \times 32$ image
    \[ |\theta| = 3 \cdot 32^4 + 32^2 \approx 3 \cdot 10^6 \]

• Consequences
  - Hard to train
  - Need to initialize carefully
  - *Convolutional nets reduce the number of parameters!*
Convolutional Neural Networks (CNN, ConvNet)

- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end


Slide credit: Svetlana Lazebnik
Convolutional Networks: Intuition

- Fully connected network
  - E.g. $1000 \times 1000$ image
    - 1M hidden units
    - $\Rightarrow$ 1T parameters!

- Ideas to improve this
  - Spatial correlation is local

Slide adapted from Marc'Aurelio Ranzato

Image source: Yann LeCun
Convolutional Networks: Intuition

• Locally connected net
  ➢ E.g. 1000×1000 image
    1M hidden units
    10×10 receptive fields
  ⇒ 100M parameters!

• Ideas to improve this
  ➢ Spatial correlation is local
  ➢ Want translation invariance
Convolutional Networks: Intuition

- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels

Slide adapted from Marc'Aurelio Ranzato
Convolutional Networks: Intuition

- **Convolutional net**
  - Share the same parameters across different locations
  - Convolutions with learned kernels

- **Learn multiple filters**
  - E.g. $1000 \times 1000$ image
    - 100 filters
    - $10 \times 10$ filter size
  - $\Rightarrow 10k$ parameters

- **Result: Response map**
  - size: $1000 \times 1000 \times 100$
  - Only memory, not params!
Important Conceptual Shift

• Before

• Now:
Convolution Layers

Example
image: $32 \times 32 \times 3$ volume

**Before**: Full connectivity
$32 \times 32 \times 3$ weights

**Now**: Local connectivity
One neuron connects to, e.g., $5 \times 5 \times 3$ region.
$\Rightarrow$ Only $5 \times 5 \times 3$ shared weights.

• **Note**: Connectivity is
  - Local in space (5×5 inside 32×32)
  - But full in depth (all 3 depth channels)
Convolution Layers

- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth

Slide adapted from FeiFei Li, Andrej Karpathy
Convolution Layers

- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth
  - Form a single $[1 \times 1 \times \text{depth}]$ depth column in output volume.

Naming convention:

Slide credit: FeiFei Li, Andrej Karpathy
Convolution Layers

Example:
7×7 input
assume 3×3 connectivity
stride 1

• Replicate this column of hidden neurons across space, with some **stride**.
Convolution Layers

- Replicate this column of hidden neurons across space, with some *stride*.

Example:
- 7×7 input
- assume 3×3 connectivity
- stride 1

Example:

<table>
<thead>
<tr>
<th>7×7 input</th>
<th>3×3 connectivity</th>
<th>stride 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Convolution Layers

- Replicate this column of hidden neurons across space, with some stride.

Example:
7×7 input
assume 3×3 connectivity
stride 1
Convolution Layers

- Replicate this column of hidden neurons across space, with some **stride**.

Example:
7×7 input
assume 3×3 connectivity
stride 1
Convolution Layers

- Replicate this column of hidden neurons across space, with some *stride*.

Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output
Convolution Layers

- Replicate this column of hidden neurons across space, with some **stride**.

Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output

What about stride 2?
Convolution Layers

- Replicate this column of hidden neurons across space, with some stride.

Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output

What about stride 2?
Convolution Layers

- Replicate this column of hidden neurons across space, with some **stride**.

Example:
- 7×7 input
- assume 3×3 connectivity
- stride 1
  - ⇒ 5×5 output

What about stride 2?
  - ⇒ 3×3 output
Convolution Layers

- Replicate this column of hidden neurons across space, with some stride.
- In practice, common to zero-pad the border.
  - Preserves the size of the input spatially.

Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output

What about stride 2?
⇒ 3×3 output

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Activation Maps of Convolutional Filters

Each activation map is a depth slice through the output volume.

Slide adapted from FeiFei Li, Andrej Karpathy

B. Leibe
Effect of Multiple Convolution Layers

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Convolutional Networks: Intuition

- Let’s assume the filter is an eye detector
  - How can we make the detection robust to the exact location of the eye?
Convolutional Networks: Intuition

• Let’s assume the filter is an eye detector
  - How can we make the detection robust to the exact location of the eye?

• Solution:
  - By pooling (e.g., max or avg) filter responses at different spatial locations, we gain robustness to the exact spatial location of the eye.
Max Pooling

- Effect:
  - Make the representation smaller without losing too much information
  - Achieve robustness to translations
Max Pooling

- **Note**
  - Pooling happens independently across each slice, preserving the number of slices.

---

**Single depth slice**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
CNNs: Implication for Back-Propagation

- Convolutional layers
  - Filter weights are shared between locations
  - Gradients are added for each filter location.
Topics of This Lecture

• Recap: Tricks of the Trade

• Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

• CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

- Early convolutional architecture
  - 2 Convolutional layers, 2 pooling layers
  - Fully-connected NN layers for classification
  - Successfully used for handwritten digit recognition (MNIST)


Slide credit: Svetlana Lazebnik
ImageNet Challenge 2012

- **ImageNet**
  - ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk

- **Challenge (ILSVRC)**
  - 1.2 million training images
  - 1000 classes
  - Goal: Predict ground-truth class within top-5 responses
  - Currently one of the top benchmarks in Computer Vision

[Deng et al., CVPR’09]
CNN Architectures: AlexNet (2012)

- Similar framework as LeNet, but
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data ($10^6$ images instead of $10^3$)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

ILSVRC 2012 Results

- AlexNet almost halved the error rate
  - 16.4% error (top-5) vs. 26.2% for the next best approach
  - A revolution in Computer Vision
  - Acquired by Google in Jan ‘13, deployed in Google+ in May ‘13
CNN Architectures: VGGNet (2014/15)


Image source: Hirokatsu Kataoka
CNN Architectures: VGGNet (2014/15)

- **Main ideas**
  - Deeper network
  - Stacked convolutional layers with smaller filters (+ nonlinearity)
  - Detailed evaluation of all components

- **Results**
  - Improved ILSVRC top-5 error rate to 6.7%.

---

### CNN Architectures: VGGNet (2014/15)

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>input (224 × 224 RGB image)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-4096</td>
<td>FC-1000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mainly used

---

*Image source: Simonyan & Zisserman*
Comparison: AlexNet vs. VGGNet

• Receptive fields in the first layer
  - AlexNet: $11 \times 11$, stride 4
  - Zeiler & Fergus: $7 \times 7$, stride 2
  - VGGNet: $3 \times 3$, stride 1

• Why that?
  - If you stack a $3 \times 3$ on top of another $3 \times 3$ layer, you effectively get a $5 \times 5$ receptive field.
  - With three $3 \times 3$ layers, the receptive field is already $7 \times 7$.
  - But much fewer parameters: $3 \cdot 3^2 = 27$ instead of $7^2 = 49$.
  - In addition, non-linearities in-between $3 \times 3$ layers for additional discriminative ability.

- Main ideas
  - “Inception” module as modular component
  - Learns filters at several scales within each module

GoogLeNet Visualization

Inception module + copies

Auxiliary classification outputs for training the lower layers (deprecated)

Convolution Pooling Softmax Other

B. Leibe
### Results on ILSVRC

<table>
<thead>
<tr>
<th>Method</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
<th>top-5 test error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG (2 nets, multi-crop &amp; dense eval.)</td>
<td>23.7</td>
<td>6.8</td>
<td>6.8</td>
</tr>
<tr>
<td>VGG (1 net, multi-crop &amp; dense eval.)</td>
<td>24.4</td>
<td>7.1</td>
<td>7.0</td>
</tr>
<tr>
<td>VGG (ILSVRC submission, 7 nets, dense eval.)</td>
<td>24.7</td>
<td>7.5</td>
<td>7.3</td>
</tr>
<tr>
<td>GoogLeNet (Szegedy et al., 2014) (1 net)</td>
<td>-</td>
<td>-</td>
<td>7.9</td>
</tr>
<tr>
<td>GoogLeNet (Szegedy et al., 2014) (7 nets)</td>
<td>-</td>
<td>-</td>
<td>6.7</td>
</tr>
<tr>
<td>MSRA (He et al., 2014) (11 nets)</td>
<td>-</td>
<td>-</td>
<td>8.1</td>
</tr>
<tr>
<td>MSRA (He et al., 2014) (1 net)</td>
<td>27.9</td>
<td>9.1</td>
<td>9.1</td>
</tr>
<tr>
<td>Clarifai (Russakovsky et al., 2014) (multiple nets)</td>
<td>-</td>
<td>-</td>
<td>11.7</td>
</tr>
<tr>
<td>Clarifai (Russakovsky et al., 2014) (1 net)</td>
<td>-</td>
<td>-</td>
<td>12.5</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus (Zeiler &amp; Fergus, 2013) (6 nets)</td>
<td>36.0</td>
<td>14.7</td>
<td>14.8</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus (Zeiler &amp; Fergus, 2013) (1 net)</td>
<td>37.5</td>
<td>16.0</td>
<td>16.1</td>
</tr>
<tr>
<td>OverFeat (Sermanet et al., 2014) (7 nets)</td>
<td>34.0</td>
<td>13.2</td>
<td>13.6</td>
</tr>
<tr>
<td>OverFeat (Sermanet et al., 2014) (1 net)</td>
<td>35.7</td>
<td>14.2</td>
<td>-</td>
</tr>
<tr>
<td>Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)</td>
<td>38.1</td>
<td>16.4</td>
<td>16.4</td>
</tr>
<tr>
<td>Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)</td>
<td>40.7</td>
<td>18.2</td>
<td>-</td>
</tr>
</tbody>
</table>

- **VGGNet and GoogLeNet perform at similar level**
  - Comparison: human performance \( \sim 5\% \) [Karpathy]

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/
Newer Developments: Residual Networks

- **AlexNet**, 8 layers (ILSVRC 2012)
  - 11x11 conv, 96, /4, pool/2
  - 5x5 conv, 256, pool/2
  - 3x3 conv, 384
  - 3x3 conv, 384
  - 3x3 conv, 256, pool/2
  - fc, 4096
  - fc, 4096
  - fc, 1000

- **VGG**, 19 layers (ILSVRC 2014)
  - 3x3 conv, 64
  - 3x3 conv, 64, pool/2
  - 3x3 conv, 128
  - 3x3 conv, 128, pool/2
  - 3x3 conv, 256
  - 3x3 conv, 256
  - 3x3 conv, 256
  - 3x3 conv, 256
  - 3x3 conv, 256, pool/2
  - 3x3 conv, 512
  - 3x3 conv, 512
  - 3x3 conv, 512
  - 3x3 conv, 512
  - 3x3 conv, 512
  - 3x3 conv, 512
  - 3x3 conv, 512
  - 3x3 conv, 512, pool/2
  - fc, 4096
  - fc, 4096
  - fc, 1000

- **GoogleNet**, 22 layers (ILSVRC 2014)
Newer Developments: Residual Networks

- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers
  - We’ll analyze this mechanism in more detail later…
ImageNet Performance

ResNet | 152 layers
GoogleNet | 22 layers
VGG | 19 layers
AlexNet | 8 layers
ILSVRC'13 | 11.7%
ILSVRC'12 | 16.4%
ILSVRC'11 | shallow
ILSVRC'10 | 28.2%

ImageNet Classification top-5 error (%)
Understanding the ILSVRC Challenge

• Imagine the scope of the problem!
  - 1000 categories
  - 1.2M training images
  - 50k validation images

• This means...
  - Speaking out the list of category names at 1 word/s...
    ...takes 15mins.
  - Watching a slideshow of the validation images at 2s/image...
    ...takes a full day (24h+).
  - Watching a slideshow of the training images at 2s/image...
    ...takes a full month.
Perceptual and Sensory Augmented Computing
Machine Learning
Winter '19
More Finegrained Classes

PASCAL

- birds
  - bird

- cats
  - cat

- dogs
  - dog

ILSVRC

- flamingo
- cock
- ruffed grouse
- quail
- partridge

- Egyptian cat
- Persian cat
- Siamese cat
- tabby
- lynx

- dalmatian
- keeshond
- miniature schnauzer
- standard schnauzer
- giant schnauzer

Image source: O. Russakovsky et al.
Quirks and Limitations of the Data Set

• Generated from WordNet ontology
  ➢ Some animal categories are overrepresented
  ➢ E.g., 120 subcategories of dog breeds

⇒ 6.7% top-5 error looks all the more impressive
References and Further Reading

• LeNet

• AlexNet

• VGGNet

• GoogLeNet
References and Further Reading

• ResNet