Advanced Machine Learning
Lecture 15
Convolutional Neural Networks
11.01.2016

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de/
leibe@vision.rwth-aachen.de
This Lecture: **Advanced Machine Learning**

- **Regression Approaches**
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Gaussian Processes

- **Learning with Latent Variables**
  - Prob. Distributions & Approx. Inference
  - Mixture Models
  - EM and Generalizations

- **Deep Learning**
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & **Optimization**
  - CNNs, RNNs, RBMs, etc.
Topics of This Lecture

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

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

• CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
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.

Augmented training data (from one original image)

Image source: Lucas Beyer
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)
  - 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: Choosing the Right Learning Rate

- **Convergence of Gradient Descent**
  - Simple 1D example
    \[ W^{(\tau-1)} = W^{(\tau)} - \eta \frac{dE(W)}{dW} \]
  - What is the optimal learning rate \( \eta_{\text{opt}} \)?
  - If \( E \) is quadratic, the optimal learning rate is given by the inverse of the Hessian
    \[ \eta_{\text{opt}} = \left( \frac{d^2E(W^{(\tau)})}{dW^2} \right)^{-1} \]
  - Advanced optimization techniques try to approximate the Hessian by a simplified form.
  - *If we exceed the optimal learning rate, bad things happen!*

Recap: Advanced Optimization Techniques

- **Momentum**
  - *Instead of using the gradient to change the position of the weight “particle”, use it to change the velocity.*
  - Effect: dampen oscillations in directions of high curvature
  - Nesterov-Momentum: Small variation in the implementation

- **RMS-Prop**
  - *Separate learning rate for each weight: Divide the gradient by a running average of its recent magnitude.*

- **AdaGrad**
- **AdaDelta**
- **Adam**

Some more recent techniques, work better for some problems. Try them.
Trick: Patience

- Saddle points dominate in high-dimensional spaces!

\[ \Rightarrow \text{Learning often doesn’t get stuck, you just may have to wait...} \]
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 after that.
Topics of This Lecture

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

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

• CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
Batch Normalization

[loffe & 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.

• Effect
  - Much improved convergence
Dropout

[Srivastava, Hinton ’12]

• Idea
  - Randomly switch off units during training.
  - Change network architecture for each data point, 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.

⇒ Greatly improved performance
Topics of This Lecture

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

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

- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

B. Leibe
Neural Networks for Computer Vision

• How should we approach vision problems?

• Architectural considerations
  - 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

- Object parts
  - Primitive features
    - Input image

Face

- Eyes, nose, ...
  - Oriented edges
    - Face image

Inferotemporal cortex

- V4: different textures

V1: simple and complex cells

Photoreceptors, retina

Slide adapted from Richard Turner
Inspiration: Neuron Cells
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

**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
Convolutional Networks: Intuition

- Locally connected net
  - E.g. $1000 \times 1000$ image
  - $1$M hidden units
  - $10 \times 10$ receptive fields
  $\Rightarrow$ $100$M 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

Image source: Yann LeCun

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. 1000x1000 image
  - 100 filters
  - 10x10 filter size
  - \( \Rightarrow 10k \) parameters

- **Result: Response map**
  - size: 1000x1000x100
  - Only memory, not params!

Image source: Yann LeCun

Slide adapted from Marc’Aurelio Ranzato
Important Conceptual Shift

• Before

• Now:
Convolution Layers

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

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.
⇒ Only $5 \times 5 \times 3$ shared weights.
**Convolution Layers**

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

*before:* “hidden layer of 200 neurons”  
*near:* “output volume of depth 200”
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:
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

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
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

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

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

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

What about stride 2?

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

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

What about stride 2?

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

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

What about stride 2?
⇒ 3×3 output

- Replicate this column of hidden neurons across space, with some \textit{stride}.
### Convolution Layers

<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>

**Example:**
- 7×7 input
- Assume 3×3 connectivity
- Stride 1
- ⇒ 5×5 output

**What about stride 2?**
- ⇒ 3×3 output

- 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.
Activation Maps of Convolutional Filters

Activations:

one filter = one depth slice (or activation map)

Activation maps

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

$5 \times 5$ filters

Slide adapted from FeiFei Li, Andrej Karpathy
Effect of Multiple Convolution Layers

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Yann LeCun
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 features.

Image source: Yann LeCun

Slide adapted from Marc’Aurelio Ranzato
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.

Slide adapted from FeiFei Li, Andrej Karpathy
CNNs: Implication for Back-Propagation

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

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

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

• CNN Architectures
  ➢ LeNet
  ➢ AlexNet
  ➢ VGGNet
  ➢ GoogLeNet
**CNN Architectures: LeNet (1998)**

- 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
AlexNet Results

AlexNet Results

Test image

Retrieved images

CNN Architectures: VGGNet (2015)

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

---

**ConvNet Configuration**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers</td>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
<td>19 weight layers</td>
</tr>
<tr>
<td>Input</td>
<td>(224 x 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>LRN</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
<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>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>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>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>FC-4096</td>
<td>FC-4096</td>
<td>FC-1000</td>
<td>soft-max</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Mainly used

Image source: Simonyan & Zisserman
Comparison to AlexNet

K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

Image source: Hirokatsu Kataoka
CNN Architectures: GoogLeNet (2014)

Main ideas

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

GoogLeNet Visualization
## 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>
References and Further Reading

• LeNet

• AlexNet

• VGGNet

• GoogLeNet