Computer Vision – Lecture 13

Deep Learning IV

18.06.2019

Bastian Leibe
Visual Computing Institute
RWTH Aachen University
http://www.vision.rwth-aachen.de/

leibe@vision.rwth-aachen.de
Course Outline

• Image Processing Basics
• Segmentation & Grouping
• Object Recognition & Categorization
  ➢ Sliding Window based Object Detection
• Local Features & Matching
• Deep Learning
  ➢ Convolutional Neural Networks (CNNs)
  ➢ Deep Learning Background
  ➢ CNNs for Object Detection
  ➢ CNNs for Semantic Segmentation
  ➢ CNNs for Matching
• 3D Reconstruction
Recap: R-CNN for Object Detection

Slide credit: Ross Girshick
Recap: Faster R-CNN

- One network, four losses
  - Remove dependence on external region proposal algorithm.
  - Instead, infer region proposals from same CNN.
  - Feature sharing
  - Joint training
  => Object detection in a single pass becomes possible.
Recap: Mask R-CNN


Slide credit: FeiFei Li
Recap: YOLO / SSD

- Idea: Directly go from image to detection scores
- Within each grid cell
  - Start from a set of anchor boxes
  - Regress from each of the B anchor boxes to a final box
  - Predict scores for each of C classes (including background)
Topics of This Lecture

• Practical Advice on CNN training
  - Data Augmentation
  - Initialization
  - Batch Normalization
  - Dropout
  - Learning Rate Schedules

• CNNs for Segmentation
  - Fully Convolutional Networks (FCN)
  - Encoder-Decoder architecture
  - Transpose convolutions
  - Skip connections

• CNNs for Human Body Pose Estimation
Data Augmentation

• Idea
  - Augment original data with synthetic variations to reduce overfitting

• Example augmentations for images
  - Cropping
  - Zooming
  - Flipping
  - Color PCA
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
Glorot Initialization

• 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 $Y$.
  - *We want the variance of the input and output of a unit to be the same,* therefore $n \ Var(W_i)$ should be 1. This means
    \[
    Var(W_i) = \frac{1}{n} = \frac{1}{n_{in}}
    \]
  - Or for the backpropagated gradient
    \[
    Var(W_i) = \frac{1}{n_{out}}
    \]
  - As a compromise, Glorot & Bengio propose to use
    \[
    Var(W) = \frac{2}{n_{in} + n_{out}}
    \]
  \Rightarrow Randomly sample the initial weights with this variance.
He Initialization

- Extension of Glorot Initialization to ReLU units
  - Use Rectified Linear Units (ReLU)
    \[ g(a) = \max\{0, a\} \]
  - Effect: gradient is propagated with a constant factor
    \[ \frac{\partial g(a)}{\partial a} = \begin{cases} 
    1, & a > 0 \\
    0, & \text{else} 
  \end{cases} \]
- Same basic idea: Output should have the input variance
  - However, the Glorot derivation was based on \( \tanh \) units, linearity assumption around zero does not hold for \( \text{ReLU} \).
  - He et al. made the derivations, proposed to use instead
    \[ \text{Var}(W) = \frac{2}{n_{\text{in}}} \]
Practical Advice

• Initializing the weights
  - Draw them randomly from a zero-mean distribution.
  - Common choices in practice: Gaussian or uniform.
  - Common trick: add a small positive bias (+$\varepsilon$) to avoid units with ReLu nonlinearities getting stuck-at-zero.

• When sampling weights from a uniform distribution $[a,b]$
  - Keep in mind that the standard deviation is computed as
    \[ \sigma^2 = \frac{1}{12} (b - a)^2 \]
  - Glorot initialization with uniform distribution
    \[ W \sim U \left[ -\frac{\sqrt{6}}{\sqrt{n_{in}} + n_{out}}, \frac{\sqrt{6}}{\sqrt{n_{in}} + n_{out}} \right] \]
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

B. Leibe
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
Choosing the Right Learning Rate

- Behavior for different learning rates

![Graphs showing the effect of learning rates on a loss function](image-url)
Learning Rate vs. Training Error

![Graph showing the relationship between learning rate and training error. The graph highlights a point and notes not to go beyond this point.]

Do not go beyond this point!

Image source: Goodfellow & Bengio book
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
Summary

• Deep multi-layer networks are very powerful.

• But training them is hard!
  ➢ Complex, non-convex learning problem
  ➢ Local optimization with stochastic gradient descent

• Main issue: getting good gradient updates for the lower layers of the network
  ➢ Many seemingly small details matter!
  ➢ Weight initialization, normalization, data augmentation, choice of nonlinearities, choice of learning rate, choice of optimizer,…

⇒ Exercise 5 will guide you through those steps. Take advantage of it!
Topics of This Lecture

• Practical Advice on CNN training
  ➢ Data Augmentation
  ➢ Initialization
  ➢ Batch Normalization
  ➢ Dropout
  ➢ Learning Rate Schedules

• **CNNs for Segmentation**
  ➢ Fully Convolutional Networks (FCN)
  ➢ Encoder-Decoder architecture
  ➢ Transpose convolutions
  ➢ Skip connections

• **CNNs for Human Body Pose Estimation**
Semantic Segmentation

- **Semantic Segmentation**
  - Label each pixel in the image with a category label
  - Don’t differentiate instances, only care about pixels

- **Instance segmentation**
  - Also give an instance label per pixel
Segmentation Idea: Sliding Window

- Problem
  - Very inefficient
  - No reuse of features between shared patches

Full image

Extract patch

Classify center pixel with CNN

Cow

Cow

Grass

(e.g., AlexNet)

Slide adapted from FeiFei Li
Segmentation Idea: Fully-Convolutional Nets

- Design a network as a sequence of convolutional layers
  - To make predictions for all pixels at once
  - **Fully Convolutional Networks** (FCNs)
    - All operations formulated as convolutions
    - Fully-connected layers become $1 \times 1$ convolutions
    - Advantage: can process arbitrarily sized images
CNNs vs. FCNs

• CNN

• FCN

• Intuition
  - Think of FCNs as performing a sliding-window classification, producing a heatmap of output scores for each class
  - But: more efficient, since computations are reused between windows

Image source: Long, Shelhamer, Darrell
Segmentation Idea: Fully-Convolutional Nets

- Design a network as a sequence of convolutional layers
  - To make predictions for all pixels at once

- Problem
  - Convolutions at original image resolution will be very expensive!

Slide adapted from FeiFei Li
Segmentation Idea: Fully-Convolutional Nets

- Design a network as a sequence of convolutional layers
  - With downsampling and upsampling inside the network!
  - Downsampling
    - Pooling, strided convolution
  - Upsampling
    - ???

Slide credit: FeiFei Li
In-Network Upsampling: “Unpooling”

- Nearest-Neighbor
  - Simplest version
  - Problem: blocky output structure

- “Bed of Nails”
  - Preserve fine-grained structure of the output
  - Problem: fixed location for upsampled stimuli

Slide credit: FeiFei Li
In-Network Upsampling: “Max Unpooling”

- **Max Pooling**
  - Remember which element was max!

- **Max Unpooling**
  - Use positions from pooling layer
  - Use corresponding pairs of downsampling and upsampling layers together
  - Remember which elements were max

Slide credit: FeiFei Li
Learnable Upsampling: Transpose Convolution

• Recall: Normal convolution, stride 2, pad 1

• Effect
  - Filter moves 2 pixels in the input for every one pixel in the output
  - Stride gives ration between movement in input and output

Slide credit: FeiFei Li
Learnable Upsampling: Transpose Convolution

- Recall: Normal convolution, \textit{stride 2} pad 1

- Effect
  - Filter moves 2 pixels in the input for every one pixel in the output
  - Stride gives ration between movement in input and output

Slide credit: FeiFei Li
Learnable Upsampling: Transpose Convolution

• Recall: Normal convolution, \textit{stride 2} pad 1

![](image)

• **Effect**
  - Filter moves 2 pixels in the input for every one pixel in the output
  - Stride gives ratio between movement in input and output

Slide credit: FeiFei Li
Learnable Upsampling: Transpose Convolution

• Now: 3x3 transpose convolution, stride 2 pad 1
Learnable Upsampling: Transpose Convolution

- Now: 3x3 transpose convolution, stride 2 pad 1

Input gives weight for filter

Input: 2 x 2

Output: 4 x 4

Slide credit: FeiFei Li
Learnable Upsampling: Transpose Convolution

• Now: 3x3 transpose convolution, **stride 2** pad 1

- Effect
  - Filter moves 2 pixels in the *output* for every one pixel in the *input*
  - Stride gives ration between movement in output and input

Slide credit: FeiFei Li
Learnable Upsampling: Transpose Convolution

• Now: 3x3 transpose convolution, stride 2 pad 1

Other names
- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

Sum where output overlaps

Input gives weight for filter

Input: 2 x 2

Output: 4 x 4

Slide credit: FeiFei Li
Learnable Upsampling: 1D Example

- Observations
  - Output contains copies of the filter weighted by the input, summing overlaps in the output
  - Need to crop one pixel from output to make output exactly 2x input

Slide credit: FeiFei Li
Convolution as Matrix Multiplication (1D Example)

• Express convolution in terms of matrix multiplication
  - Example:
    - 1D conv
    - Kernel size = 3
    - Stride 1, padding = 1

\[ \vec{x} \ast \vec{a} = X \vec{a} \]

\[
\begin{bmatrix}
    x & y & z & 0 & 0 & 0 \\
    0 & x & y & z & 0 & 0 \\
    0 & 0 & x & y & z & 0 \\
    0 & 0 & 0 & x & y & z \\
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
    c \\
    d \\
\end{bmatrix} =
\begin{bmatrix}
    ay + bz \\
    ax + by + cz \\
    bx + cy + dz \\
    cx + dy \\
\end{bmatrix}
\]

• Convolution transpose multiplies by the transpose of the same matrix
  - When stride = 1, convolution transpose is just a regular convolution (with different padding rules)

\[ \vec{x} \ast^T \vec{a} = X^T \vec{a} \]

\[
\begin{bmatrix}
    x & 0 & 0 & 0 \\
    y & x & 0 & 0 \\
    z & y & x & 0 \\
    0 & z & y & x \\
    0 & 0 & z & y \\
    0 & 0 & 0 & z \\
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
    c \\
    d \\
\end{bmatrix} =
\begin{bmatrix}
    ax \\
    ay + bx \\
    az + by + cx \\
    bz + cy + dx \\
    cz + dy \\
    dz \\
\end{bmatrix}
\]
Convolution as Matrix Multiplication (1D Example)

- Express convolution in terms of matrix multiplication
  - Example:
    - 1D conv
    - Kernel size = 3
    - Stride 2, padding = 1

\[ \vec{x} \ast \vec{a} = X \vec{a} \]

\[
\begin{bmatrix}
    x & y & z & 0 & 0 & 0 \\
    0 & 0 & x & y & z & 0 \\
    a & b & c & d \\
    0 
\end{bmatrix}
\begin{bmatrix}
    0 \\
    a \\
    b \\
    c \\
    d \\
    0 
\end{bmatrix}
= \begin{bmatrix}
    ay + bz \\
    bx + cy + dz
\end{bmatrix}
\]

- Convolution transpose multiplies by the transpose of the same matrix
  - When stride > 1, convolution transpose is no longer a normal convolution!

\[ \vec{x} \ast T \vec{a} = X^T \vec{a} \]

\[
\begin{bmatrix}
    x & 0 \\
    y & 0 \\
    z & x \\
    0 & y \\
    0 & z \\
    0 & 0 
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
    c \\
    d \\
    0 \\
    0 
\end{bmatrix}
= \begin{bmatrix}
    ax \\
    ay \\
    az + bx \\
    by \\
    bz \\
    0 
\end{bmatrix}
\]

Slide credit: FeiFei Li
Segmentation Idea: Fully-Convolutional Nets

- Design a network as a sequence of convolutional layers
  - With downampling and upampling inside the network!
  - **Downsampling**
    - Pooling, strided convolution
  - **Upsampling**
    - Unpooling or strided transpose convolution

Slide credit: FeiFei Li
Extension: Skip Connections

- Encoder-Decoder Architecture with skip connections
  - Problem: downsampling loses high-resolution information
  - Use skip connections to preserve this higher-resolution information

Image source: Newell et al.
Example: SegNet

- **SegNet**
  - Encoder-Decoder architecture with skip connections
  - Encoder based on VGG-16
  - Decoder using Max Unpooling
  - Output with K-class Softmax classification

Example: U-Net

- U-Net
  - Similar idea, popular in biomedical image processing
  - Encoder-Decoder architecture with skip connections

O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015
Semantic Segmentation

- Recent results
  - Based on an extension of ResNets for high-resolution segmentation

[Pohlen, Hermans, Mathias, Leibe, CVPR 2017]
Topics of This Lecture

• Practical Advice on CNN training
  - Data Augmentation
  - Initialization
  - Batch Normalization
  - Dropout
  - Learning Rate Schedules

• CNNs for Segmentation
  - Fully Convolutional Networks (FCN)
  - Encoder-Decoder architecture
  - Transpose convolutions
  - Skip connections

• CNNs for Human Body Pose Estimation
FCNs for Human Pose Estimation

- **Input data**
  - Image
  - Keypoints
  - Labels

- **Task setup**
  - Annotate images with keypoints for skeleton joints
  - Define a target disk around each keypoint with radius $r$
  - Set the ground-truth label to 1 within each such disk
  - Infer heatmaps for the joints as in semantic segmentation

Slide adapted from Georgia Gkioxari
Heat Map Predictions from FCN

Test Image

<table>
<thead>
<tr>
<th>Right Ankle</th>
<th>Right Knee</th>
<th>Right Hip</th>
<th>Right Wrist</th>
<th>Right Elbow</th>
<th>Right Shoulder</th>
</tr>
</thead>
</table>

Slide adapted from Georgia Gkioxari
Example Results: Human Pose Estimation

[Rafi, Gall, Leibe, BMVC 2016]
More Recently: Parts Affinity Fields

- [https://www.youtube.com/watch?v=pW6nZXeWlGM](https://www.youtube.com/watch?v=pW6nZXeWlGM)
References

• ReLu

• Initialization
References and Further Reading

- **Batch Normalization**

- **Dropout**
References: Computer Vision Tasks

• Semantic Segmentation
References: Computer Vision Tasks

• Human Body Pose Estimation
  - S.E. Wei, V. Ramakrishna, T. Kanade, Y. Sheikh, Convolutional Pose Machines, CVPR 2016.
  - B. Xiao, H. Wu, Y. Wei, Simple Baselines for Human Pose Estimation and Tracking, ECCV 2018. (Code)