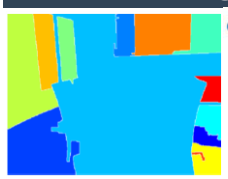
Deep Learning for Computer Vision

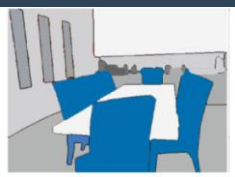
Lukas Bach - lbach@outlook.de - lukasbach.com

1 Grundlagen

1.1 Scene Segmentation



1. Image Segmentation

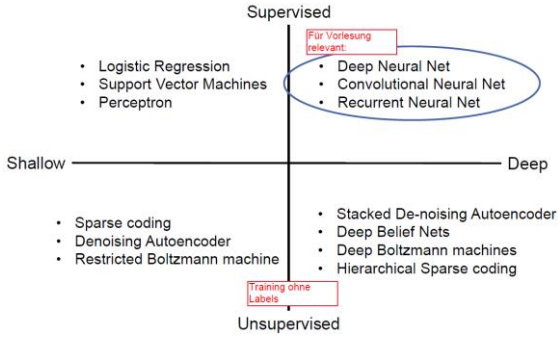


2. Semantic Segmentation

3. Instance Segmentation

2 Neural Network basics

2.1 Feature Learning Taxonomie



Aufteilung in

- Supervised, Unsupervised
- Shallow, Deep
- Generativ, Diskriminativ

2.2 Single-Layer Perceptron

- Needs non-linear activation function. Otherwise the network is a linear regression model
- Non-linear activation function and combining layers enables non-linear seperation.
- Forward propagation:

$$h^{n+1} = f(W^{n+1}h^n + b^n)$$

2.3 Softmax

- Normalize scores as pseudo-probabilities, sum of all probabilities should equal one

$$p(y = c_k|x;\theta) = \frac{e^{\theta_k^T x}}{\sum_j e^{\theta_j^T x}}$$

Describes soft-max prob that feature x belongs to class c_k (output y, under model θ. Class j has score θ_j^T). Exponential function yields nice differential.

2.4 Loss

Negative log-likelyhood commonly used for softmax:

$$L(x, y; \theta) = -\sum_j y_j \log p(c_j|x)$$

[TODO: Wann sehen Werte wie aus?]

[TODO: Why y? y is one-hot-encoded anyway, so only one summand is used]

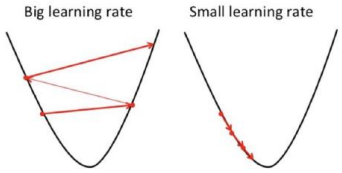
A good model is $\theta^* = \arg\min_\theta \sum_n L(x^n, y^n; \theta)$, i.e. minimizes the loss for all inputs.

When training the model, weights are wiggled by small amounts, and loss is checked, weights are updated based on change in loss.

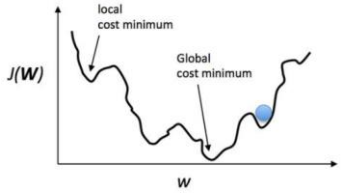
[TODO Backprop, 2-30ff]

2.5 Gradient Descent

- ("Classic") Batch gradient descent: minimize sum of losses of all examples.
  Guaranteed to converge to global minimum for convex error surfaces, to local minimum for non-convex surfaces. Full sum is expensive when N is large!
- Mini-batch gradient descent: Approximate sum with mini-batch of examples (32, 64…)
- Update-rule:
  - $\theta \leftarrow \theta - \eta \frac{\delta L}{\delta \theta}$
  - $\theta \leftarrow \theta + \Delta_t, \quad \Delta_t = \mu\Delta_{t-1} - \eta \frac{\delta L(\theta)}{\delta \theta}$ with Momentum
- Challenges:
  - Choosing the right learning rate η for weight update $\theta \leftarrow \theta - \eta \frac{\delta L}{\delta \theta}$. If too large, fluctuations at minimum or diverging. If too small, very slow convergence.

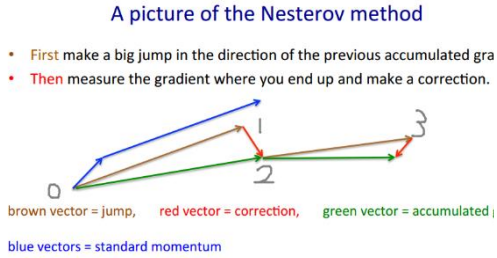


  - Non convex loss functions: Most error functions are highly non-convex!



- Momentum $\mu \cong .9$ accelerates gradients changes in the same direction and reduces updates if gradient changes direction.
  - Update rule: $\theta \leftarrow \theta + \Delta_t, \quad \Delta_t = \mu\Delta_{t-1} - \eta \frac{\delta L(\theta)}{\delta \theta}$, μ is the Momentum hyperparameter

- Nesterov: Instead of computing gradient at current position, calculate at future approximate position

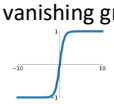


- Adagrad: Adapt learning rate depending on weights. Weights with high gradients ➔ reduced learning rate.
- Adadelta: Less aggressive Adagrad extension, restricts accumulated past gradients to fixed parameter.
- Adam: Most popular, uses average of the seconds moments of the gradients.

2.6 Activation functions

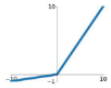
- Non linear, differentiable (backprop)
- Classic functions:
  - Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$. Common in the past, rarely used today.
    Drawbacks: Vanishing gradients, outputs not zero-centered.

  - Tanh $\tanh(x)$. Zero centered (good), but also vanishing gradients (bad).

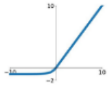


- Modern functions:
  - ReLU $\max(0, x)$: *Simple* and inexpensive. No vanishing gradient. Greatly accelerates ~x6 convergence of stochastic gradient descent compared to tanh.
    Drawback: Dying ReLU: if activation becomes zero, gradient will be zero forever (no learning anymore)

  - Leaky ReLU $\max(\alpha \cdot x, x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \alpha \cong 0.1$: attempts to fix dying ReLU.



  - Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$ (skipped): Fixes dying ReLU, more adaptable, but more parameters.
  - ELU $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



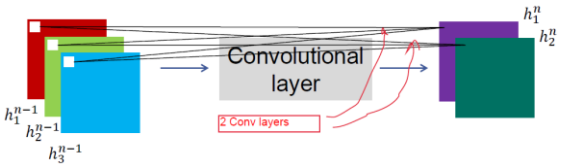
- Practical advice: Never sigmoid. Use ReLU, if not well-adjusted learning rate is a concern, try Leaky ReLU or Maxout. Possibly try tanh, but expect worse results.

2.7 Loss Functions

- If continuous multplie outputs y_j
  - L1 loss (Mean Avg Err): $L(x, y) = \sum_j (y_j - x_j)$
  - L2 loss (Mean Sq Err): $L(x, y) = \sum_j (y_j - x_j)^2$
  - x_j are predicted values, y_j are true values.
  - For regression tasks, e.g. continuous labels are learned, e.g. bbs

3 CNNs

- In traditional CV, features are not learnt (instead, manual edge detection…). Today, end-to-end learning, hierarchical representation of data is learnt.
- Convolutions are used to compress data (FCs early one not doable, too many parameters).
- Filter depth always extends to the full depth of the input volume (224x224x*3* image ➔ 5x5x*3* filter needed)
- Filter slides over image with specific stride, compute dot product at each location
- Output size depends on size, stride, padding
  - E.g. 224x224x3, 5x5x3 filter, stride 1 ➔ 220x220x1
  - Using 6 filters would result in output depth 6 (6 activation/feature maps)



- Formula: (FM=Feature Map)

$$\underbrace{h_j^n}_{output\ FM} = \max\left(0, \sum_{k=1}^{K} \overbrace{h_k^{n-1}}^{input\ FM} * \underbrace{w_{kj}^n}_{kernel}\right)$$

- CNNs layers: $[conv, pool]^+ FC$
  - *Typical CNN block*:
    - Convolve (filter)
    - Non-linearity (rectified linear)
    - Pooling (local max)
    - Normalization (contrast)

3.1 Conv Layer Sizes

M input-channels of size DxD, N filter kernels of size KxK.

Input size = DxD, Output size = $\left(\frac{D-K+2 \cdot padding}{stride}\right) + 1$.
$\#parameters = N \cdot M \cdot K^2$.

Output has usually more feature maps than input. Filter size depends on to-be-detected pattern.

3.2 Pooling

- Sum/Average pooling or Max pooling
- Introduces invariance, improves robustness to noise/clutter
- Overlapping pooling:
  - Stride s, patch z. Patch is diameter of pool.
  - Overlapping pooling: $s < z$

3.3 Notable networks

- LeNet: 1989 for handwritten digit classifgication on MNIST. Conv-Pool-Conv-Pool-FC-FC
- AlexNet: 2012. Many conv-pool repetitions and high strides causes high compression. Final classification with 2 dense and a soft-max layer. E2E trained.
  - Conv-MaxPool-Norm-Conv-MaxPool-Norm-Conv-Conv-Conv-MaxPool-FC-FC-FC
  - Norm-layers uncommon nowadays

  Tricks used in AlexNet:
  - Data augmentation: Data synthesizing viatranslations, reflections, intensity variations
  - Dropout: Randomly set neuron output to 0. Reduces dependencies between neurons, each neuron is forced to learn.
    Dropout rate = 0.5 ➔ a neuron is active half the time
    Introduces noise, increases generalizability.
  - Batch normalization (commonly used today): output distribution is highly coupled with learning rate. Goal: activations should have mean=0, variance=1. Normalize using batch statistics.
    Normalize each data batch by adjusting values by their mean and values.
    Each feature/channel is normalized using their mean/variance computed from batch. Implemented as layers after FC/Conv, before nonlinearity.
    Problem: Can limit expressive power of the Net
    ⇨ Learn parameters γ, β that rescale/shift normalized output to desired values $y_k = \gamma_k x_k + \beta_k$

    Benefits:
    - Higher learning rates
    - Hidden unit values don't shift around as much
    - Less sensitive to weight initialization
    - Reduces overfitting (adds noise)

  Parallelism during implementation:
  - Data paralellism: Workers train the same model on different data-subsets. Share gradients and adjust weighgts. Efficient when gradients are sparse.
  - Model parallelism: Workers train modelparts on the same data, share neuron activations.
- ZFNet: 2013, AlexNet with changes: Smaller filter & stride at first convolution (less params), more filters at later convolutions.
  Small adjustments, but large performance increase.
- Going Deeper: Only 3x3 filters, few weights.
- VGGNet: Many layers (16-19, compared to 8 from AlexNet), many parameters in late FC-layers.

Take-home messages: 03-68

4 CNNs (2)

4.1 CNNs as feature extractors

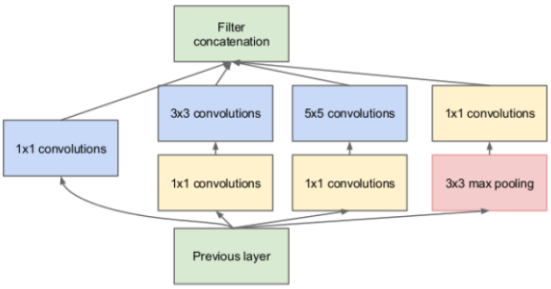
Example network: DeCaf aka : A **De**ep **C**onvolutional **A**ctivation **F**eature for Generic Visual Recognition. Idea:

- Train image-classification network end-to-end
- Switch last FC-layers for other tasks (e.g. scene recog)
- Remove last FC-layers for feature extraction (hidden unit values as features)

4.2 Other notable networks

Tendencies: much more layers, but far less params.

- GoogleNet
  - Very deep (22 layers)
  - Efficient "Inception" module



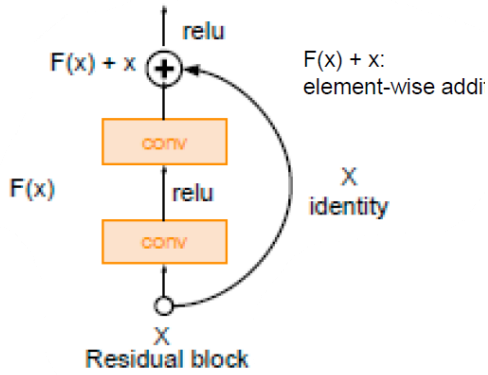
  - No FC-layers (linear layer at the end instead)
  - 12x less parameters than AlexNet, 5mil
  - Note that pooling has stride 1, so image sizes are not reduced, during concatenation all sizes are equal to original size

  Inception module: good local network topology. Multiple conv sizes and pooling, all outputs are concatenated. Output sizes are the same after each module.
  Problem: High computation complexity per module. Solution: Bottleneck-layers (1x1 convolutions) reduce feature depth.

  - Stem network: First part of the network before modules, conv-pool-conv-conv-pool
  - Classifier output: Last part after modules, AvgPooling+LinearLayer
  - *Auxiliar loss layers*: injects additional gradient at levels next to modules, AvgPool-1x1Conv-

Fc-Fc-Softmax. Fights vanishing/exploding gradient.

- ResNet
  - Very very deep (152 layers)
  - Residual connections ("Skip-…") over at most two layers.
  - Observation: Deeper models generally perform worse due to being hard to optimize. Solution: residual mappings, allows gradient to propagate through



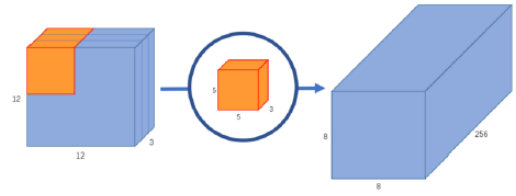
  - Architecture: Stacked *residual blocks*, each consisting of two 3x3-conv layers, periodical double #filters+downsample. Additional conv-layer at the start, global avg pooling after last conv and then only one 1000-FC.

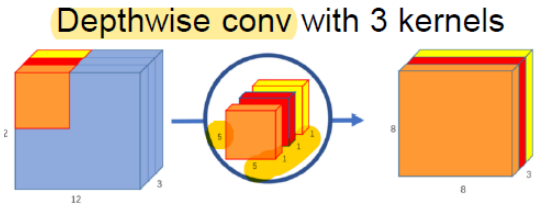
Results:

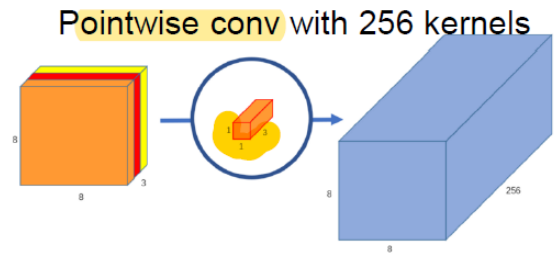
- VGG inefficient, but good accuracy
- GoogleNet efficient, good accuracy
- AlexNet efficient, low accuracy
- ResNet moderately efficient, highest acc

Improvements:

- Improved ResNet block design, add Batch Normalization and in blocks between skips
- Wide Residual Networks: ? TODO?
- ResNeXt: wider ResNet blocks similar to inception modules
- DenseNet: skips between pretty much all layers. Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- MobileNetv1: smaller model size&complexity. Fewer params via depthwise- and pointwise conv using the same kernel.







Take-home messages: 04-67.

5 Object Detection

Object detection: What and where?

Problem: Variable sized outputs (no idea how many bounding boxes are in one image)

Naïve approach: Sliding window, detect each class for each window position. Apply window at different scales of image.

Problem: Very time consuming. Can be solved by using fast classifiers, or using region proposals.

5.1 Region Proposals

Objectness approach, very fast (1k proposals at 300fps)

Selective Search: Split image based on color, merge regions based on similarity.

5.2 R-CNN

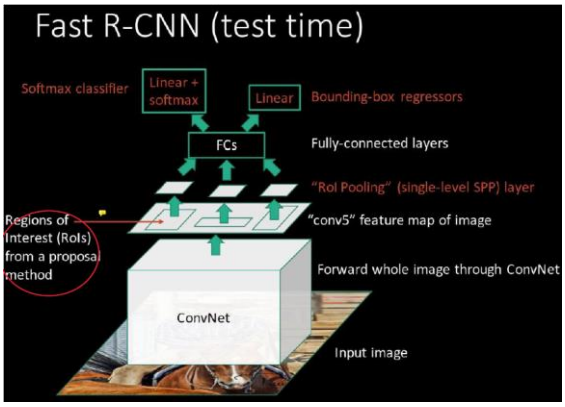
aka Slow-R-CNN. Feed proposed regions into RCNN, embedded AlexNet classifies the regions. Last (FC) layers are reinitialized to output class scores, then for each class a SVM is trained.

Region proposals are improved using a regression model (results are "Regression targets", describe relative changes to bounding boxes)

Precision/Recall/F1: **TODO**

5.3 Fast R-CNN

Image is inputted only once for all proposals. Feature map is computed only once for all proposals.



ROI pooling: image size is mapped to fixed size for FC (?)

Entire network is trained e2e, instead of separate SVM/regressor training as in RCNN.

Significantly faster than RCNN (8.8x training speed, 146x inference speed, +1% mean accuracy)

5.4 Faster R-CNN

- Region Proposal Network (RPN): Gets feature map from larger conv network as input, outputs list of p proposals and "objectness" score of size $p \cdot 6$.
  - Anchors: Initial boxes that are adapted during inference
  - Each anchor is labelled based on IoU with groundtruth to generate loss
- Faster-R-CNN is mostly trained alltogether with four losses: objectness classification, anchor regression, object class classification, detection regression.
- x10 speed without accuracy costs.

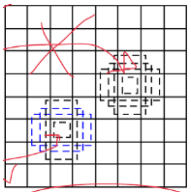
	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Pascal Voc and COCO as example data sets.

5.5 SDD Detector

Single Shot MultiBox Detector. Avoids resampling of regions to speedup the process.

Idea: Don't detect box positions, instead use *fixed default boxes* and classify object class/box regression for each default box.



Much fewer parameters (no FCs), only one forward-pass ("single shot"). Non-maximum suppression to reduce false-positives/overlapping boxes.

As accurate as Faster-RCCN and 6x as fast, or more accurate and 3x as fast.

5.6 Mask R-CNN

Adds Segmentation stage to R-CNN.

5.7 YOLO-9000

You Only Look Once, single-shot model.

Take-Home Messages: 04-73

6 Segmentation

Find coherent regions in image, without classifying the regions.

Problem: Hard to say how many regions one wants. Edge detection can for example find various regions based on thresholds.

6.1 Simple approaches via Clustering

- Agglomerative clustering (merging): one cluster per pixel, then continue to merge
- Divisive clustering (splitting):

Problems are choice of inter-cluster distances (TODO?), and stopping criterion (stop at how many clusters?).

Other naïve approach: Reduce color depth to get regions. But then no semantic info is preserved!

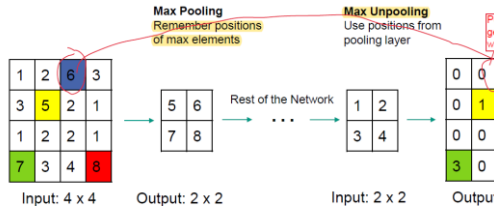
6.1.1 Segmentation via Kmeans

- Choose initial mean values for k regions
- Assign pixels to their closest mean
- Recompute the means by the average of samples in their new classes
- Stop when means don't change anymore

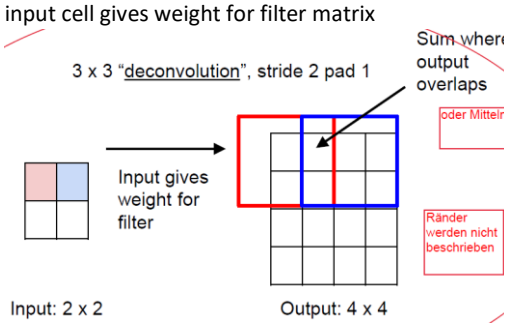
Idea: Switch between "assuming" that cluster centers are known and pixel allocations are known, each time recalculating the other one.

6.2 Semantic Segmentation

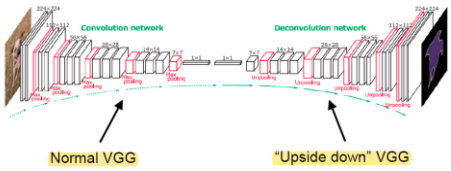
- Sliding window: slide window over image, classify each window. But: Way too expensive
- Fully Convolutional Neural Networks:
  Convolution & Pooling → Upsampling
  Upsampling to preserve original image size
  Upsampling methods
  - Nearest Neighbor, Bilinear interpolation
  - Max Unpooling (remember positions of max elements from before pooling, zeros on other locations)



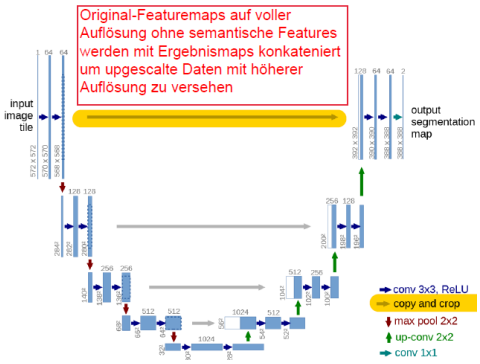
  - Transposed Convolution (Deconvolution): input cell gives weight for filter matrix



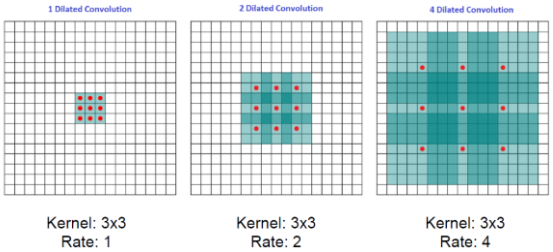
    TODO further lookup
    - In der Praxis wird zuerst vortrainiertes VGG mit Max Pooling benutzt, danach gespiegeltes VGG mit Max Unpooling:

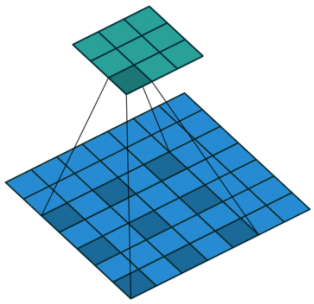


    - Skip Connections are added to retain information
  - U-Net
    - Problem: Pretrained net not applicable for every domain
    - Solution: Copy initial features to later parts of the network with concatenations



- Atrous ("Dilated") Conolution
  We want bigger receptive field, few parameters, and stride of one.





  In the example, all filters are 3x3 and "cover" the green area, but only take the red dots into account.
  "Rate"-parameter
  In normal conv nets, we reduce the resolution in each step. In atrous conv nets, we can retain resolution and just increase dilation.
- Atrous Spatial Pyramid Pooling
  Multiple atrous conv operations simultanously with varying rates.
  Note that, $rate \cong resolution$ causes filters to degenerate to 1x1 filters. Solution: Global Average Pooling (TODO)

6.3 Evaluation

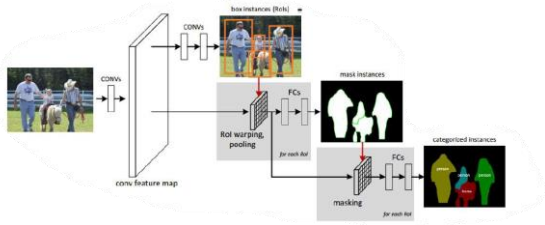
Use pixel-wise accuracy (calculate TP, FP, … for each pixel). Can be inaccurate for small object regions, as TN is large even for bad segmentations.

Most commonly used: Intersection over Union (IoU, Jaccard Index): $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP+FP+FN}$. Calculate that for ewach class, then avergae results (→ Mean Intersection over Union)

6.4 Instance Segmentation

Not only should areas be assign classes, but class instances should be assigned to areas independently.

- Multi-task Network Cascades
  - Boxes are regressed (box location + object/non-object)
  - Mask instances are regressed (single bbox is inputted into net, 14x14-features are outputted. Outputs are class-agnostic
  - Instances are categorized (outputs category-scores for each mask. Uses convolutional features, masked by the mask instances from previous step)

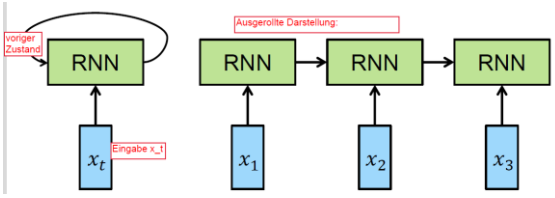


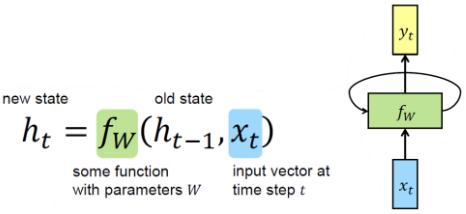
- Mask R-CNN
  - Extends Faster-R-CNN with mask prediction module.
  - Uses ROI Align instead of ROI Pooling (TODO, 06-58ff)
  - Losses: Sum of
    - classification loss (sigm cross entropy)
    - box loss (diff between GT & output coords)
    - mask loss (sigm cross entropy between GT and prediction)

Take-home messages: 06-64

7 Recurrent NNs

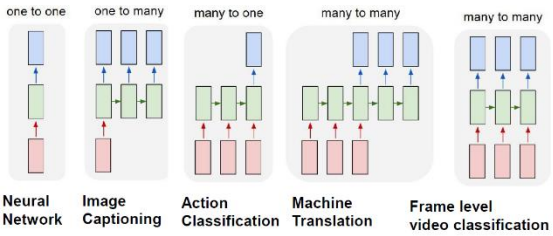
Idea: We want to process a stream of data, e.g. Image/Video Captioning.





$$h_t = f_W(h_{t-1}, x_t) \overset{e.g.}{=} \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$



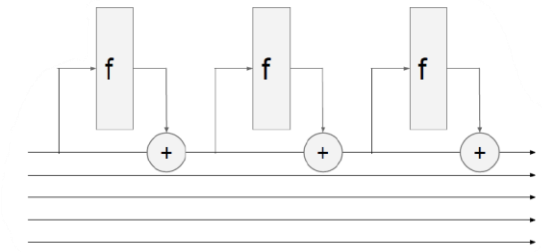
Training of net is done by "unrolling" past actions and performing backpropagation on unrolled net.

- Initialize parameters, draw weights randomly from gaussian, biases = 0.
- Calculate Loss via:
  - Unroll network
  - Forward pass, share weights across recurrence
  - Compute prediction scores
  - Soft-max
  - Negative log-likelihood at every input
- Backpropagate gradients *from every output*, update model parameters

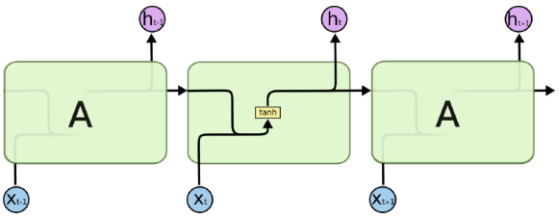
Gradient can become unstable after many unrollments ($\geq \sim 20$), and will go towards 0 (vanishing grad) or infinity (exploding grad).

7.1 LSTMs

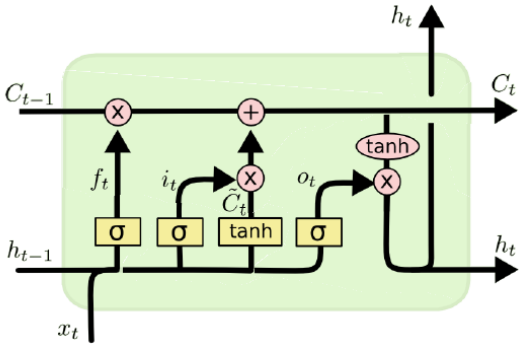
State now goes through additive instead of multiplicative changes, making it more stable (called "gating").



Vanilla RNN:



LSTM:



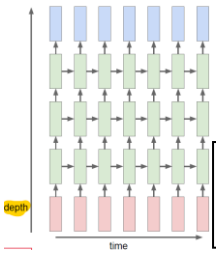
Components (see 07-24ff):

- Forget gate (σ): Reset or keep cell state. σ normalizes amount of reset
$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
- Input gate (σ): What to include from new input, similar to forget gate, but different weight matrix
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
- New cell state ($tanh$): Same input as input gate but in $[-1,1]$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
- Update cell state: how much to forget from old state (use forget gate), add modulated info from new input.
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
- Output: how much to pass on to next HS?
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

Because gradients are added and not multiplied, problem of expl./vanish. Gradient is reduced.

Relatively many parameters (4Hx2H if $|x_t| = |h_t| = H$ TODO?).

Hidden states can then be stacked on top of each other:



Results get besser with more layers

7.2 GRUs

Gated Recurrent Units: Similar to LSTM (similar performance), less parameters (3Hx2H), also additive.

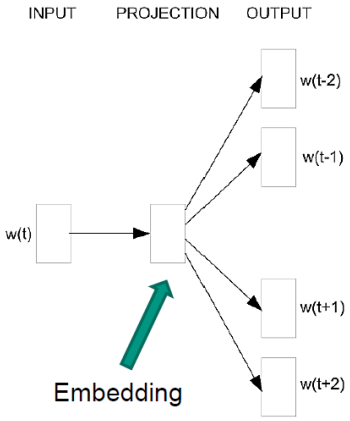
7.3 Machine Translation

Many-to-Many RNN models. Words are encoded as embeddings (captures semantics, not a too large dimensions as one-hot encoded vectors have).

Word2Vec Training:

- Supervised: Word similarity labels manually encoded
- Unsupervised: labels learnt from how they appear in text documents.

Skip-gram Models: FC-feed-forward-network, learns word vector representations, trained on billions of words. Learns to detect closest words as output for one input word.



Skip Thoughts: Word2Vec for sentences, detects closest sentences.

Take-home messages: 07-55

8 Image and Video Captioning

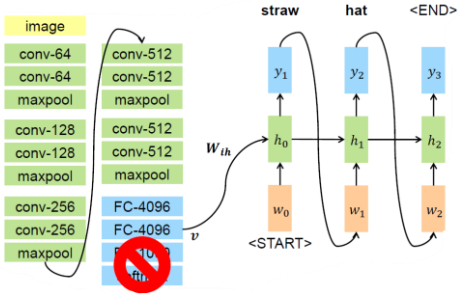
"Meaning Space" used as intermediate representation between Image Space and Sentence Space.

Fazit: Image Captioning = RNN+CNN+Attention

8.1 CNN+RNN approach

Use CNN as feature extractor, RNN to describe.

- VGG can be used for image classification (feature extr)
  - Trained on ImageNet, simple architecture
  - 1k classes, softmax at the end
  - Discard final classification (FC/output-) layer!
- Replace final FC layer with RNN hidden state



Wichtig: $x_i = y_{i-1}$

Can be trained on Microsoft COCO (Common Objects in Context) dataset.

8.2 Evaluation

- BLEU: BiLingual Evaluation Understudy
  Sentence matching precision. Good quality and used for evaluating generated sentences, but short partial

sentences with high matches also get high scores even though relevant parts are missing.
- METEOR: Metric for Evaluation of Translation with Explicit ORdering
  Measures recall instead of precision.

BLEU favors shorter sentences, METEOR favors longer sentences. Best idea to report both.

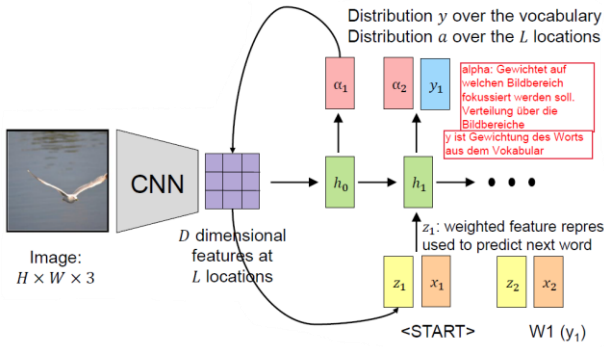
8.3 Alignments

Idea: align boxes of interest in image to captions. Use RCNN to detect regions, and RNN to generate captions.

8.4 Attention

In long sentences, early words may be forgotten. Idea: Weight words based on relevance, important earlier words are more likely to be respected during later generation.

Attention in image captioning: Apply softmax indication which location in the image to "attend" (α_i):



Soft attention: Weighted combination of softmax over all locations.

Hard attention: Picks only highest scoring location, not nice to train e2e

8.5 Video Captioning

Harder, because motions and actions need to be analyzed.

TODO

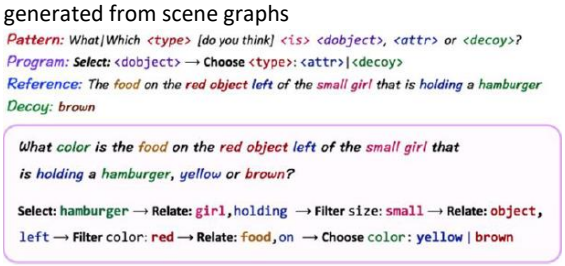
Take-home messages: 08-40

9 Visual Question Answering

Question + Image → Answer

Many datasets, partially synthetic. Mostly just annotated images.

- First attempt of training (Visual Madlibs): Leave key parts of sentences out and train model to predict those parts given the image (e.g. Person A wears a ~~hat~~)
- VGA dataset: image/question data, crowd-sourced
  - Multiple-Choice and Open-ended questions (where answer is single word of fixed vocab)
- CLEVR dataset: synthetic data, geometric forms in 3D
- QGA dataset: compositional QA, questions are generated from scene graphs



9.1 Approaches

9.1.1 Global Embedding

- Image embedded with CNN
- question embedded with LSTM
  - final question representation is concatenation of last hidden state and last cell state).
- Embeddings are fused (e.g. multiplicated)
- FC for classification of output class.

9.1.2 Attention based

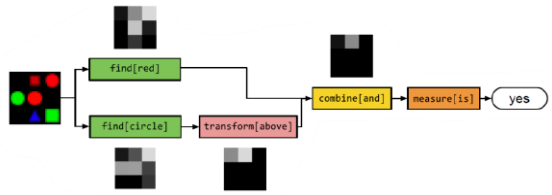
SAN: Stacked Attention Network

Again, image embedded with CNN, question embedded with LSTM or CNN.

- Image embedded with CNN on pretrained VGG net. FCs removed, 3D tensor is output.
- Question embedded either via LSTM, or CNN (1-3 dimensional convolutions, then maxpooling over time)
- Multiple passes with question vector and weighted image to refine attention
- Entirely learnt e2e, including attention weights

9.1.3 Compositional Models

Neural Module Networks



Decompose question into modular parts, train NNs for these parts.

"Is there a circle next to the square?" → $is(circle, nextto(square))$

Modules:

- Find: $image \mapsto attention.$
- Transform: $attention \mapsto attention.$
- Combine: $attention \times attention \mapsto attention.$
- Describe: $image \times attention \mapsto label$
- Measure: $attention \mapsto label$

Training:

- Per question example, build neural module chains, then train e2e
- Share weights betw similar function/argument pairs
- Only annotations needed are question/answers, no manual “here is a circle” needed

Only small performance improvements.

9.1.4 Dynamic Memory Nets (DMN)

Store facts, lookup during questions. Nets are dynamic because factbase can be adapted during runtime.

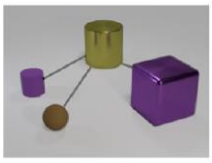
- Facts are re-embedded using GRU
- Hidden state is used as embedding of a fact
- Memory is initialized with question representation
- Read memory: Use soft attention to extract infos
  - Attention learned from facts
- Write memory: function on question and information extracted from memory (TODO?)

9.1.5 Graph Neural Networks

For relational questions.

Relational question:

Are there any rubber things that have the same size as the yellow metallic cylinder?



- Question encoded with LSTM
- Objects in image detected with CNN
- Each object is concatenated with the (same) question
- These representations are re-embedded using MLP
- Those are added together
- Final FC layer generates answer-vector

Exceeds human performance on CLEVR.

9.1.6 Video Question Answering

Problem: strong dependencies between frames. Which frames are relevant?

MovieQA dataset: Annotated movies, subtitles, scripts, …

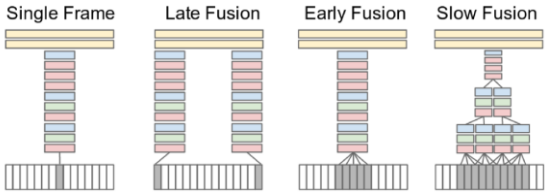
Memory network: TODO? Probably not relevant?

Take-home messages

10 Action & Activity Recognition

How to adapt CNNs to video classification w.r.t. activity?

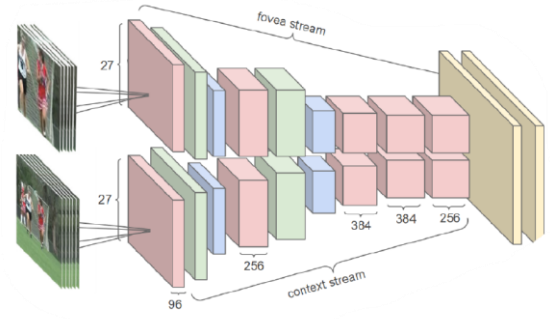
10.1 Feature aggregation over time



- Components:
  - Yellow top: FCs for classification
  - Inbetween: CNN
  - White/greys: individual image frames
- Single frame: Network sees one frame at a time, no temporal data
- Late fusion: Network sees two frames seperated by $F = 15$ frames, only last FC layer has access to temporal info
- Early fusion: Have $11 \times 11 \times 3 \times T$ filters, with T being the temporal con text ($T = 10$ images at a time)
- Slow fusion: Combination of early/late fusion. Fuse temporal data during various conv stages. *Layers higher in hierarchy have access to larger temporal context*.

10.2 Architectures

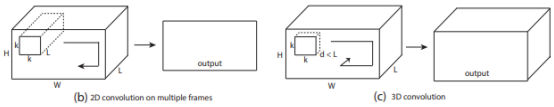
Multiresolution CNN: Input image with original resolution, and with center cut out and enlarged.



Two-stream ConvNets: one conv net for a single frame, one for optical flow vectors describing the temporal flow, merging at the end after FCs.

P-CNN: Pose based CNN. Detect human poses, increases accuracy.

C3D: 3D Convolutional networks: stack multiple video frames, perform 3D convolution within. Kernel has 3 dimensions, thus convolutes through time. Note than 2D convolution on image or multiple images always yields an image, hence don’t respect temporal information. Only 3D conv through time preserves that info, and *results in a 3D output volume*.



10.3 Summary of choices

- Type of conv operators
  - 2D kernels (image-based)
  - 3D kernels (video-based)
- Input streams
  - RGB spatial stream, usually for single-steam nets
  - Precomputed optical flow (temporal stream)
  - Other streams
- Fusion strategy across multiple frames
  - Feature aggreg over time (Single frame, …)
  - Recurrent layers (LSTM)

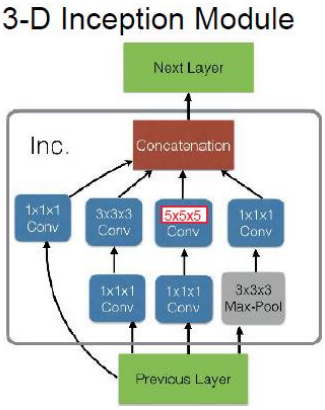
10.4 Evaluation of Architectures

Compared LSTM, 3D-ConvNet and Two-Stream. Two-Stream worked best. LSTM had lowest parameters. 3D-ConvNet had very many parameters and performed poorly.

10.5 I3D: Inflated 3D-CNN

Two-stream inflated 3D ConvNet. Idea: transfer knowledge from image recog tasks.

Inception-Module architecture extended to 3D. Filter and pooling kernels are inflated with time dimension ($N \times N$ becomes $N \times N \times N$). Weights of 2D filters can be pretrained on ImageNet.



Start with Convolutions, then Inception modules, with Pooling from time to time.

Pretraining on large image datasets is crucial.

10.6 Self-Supervised Learning with NNs

Problem: (x_i, y_i) used for training, but y_i can be hard to obtain. Huge amounts of unlabeled data is available!

➔ Use inherent structure of data to label itself (clustering?)

Can be used for pretraining activity recognition nets where data is hard to obtain.

- Reconstruction: Similar to word2Vec, find similarities. Encoder/Decoder structure with bottleneck, train net s.t. output is very similar to input, then the bottleneck contains a very compact representation
  - Every input is a label, no setup
  - Difficult for larger images with small details, often leading to trivial outputs
- Colorization
  - Use grayscale image as input, original as output. Network learns to color image.
- Context and Positon
  - Crop image into pieces, learn to assemble pieces
  - Avoid trivial solutions (line continuation) through context abuse, insert gaps or wobble area locations
- Temporal Structure
  - Mine clips with high motion, generate positive (ordered) and negative (out of order) image tuples.
  - Can improve other nets leveraging temp. str.

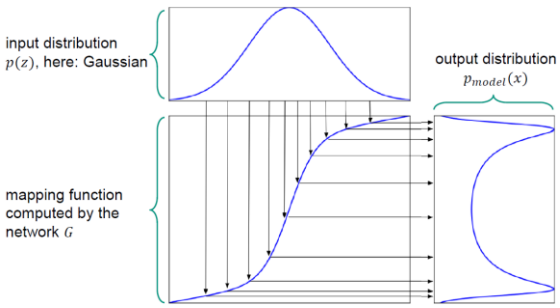
Take home messages: 10-46, 10-62

11 GANs

GAN: Generative Adversarial Networks

11.1 Generative Models

- Density estimation
  - Explicit density estimation: Explicitly define and solve distribution
  - Implicit density estimation: Implicitly learn model that can sample from distribution without explicitly defining it
- Try to train network to model distribution $p_{model}(x)$ that closely reselmbles target distribution $p_{data}(x)$.
- Sampling from network to get new images
- Implicit Generative Models
  - Start by sampling noise vector from fixed distribution.
  - Generative network computes differentiable function, mapping vector to instance in data space
    Example for mapping function:



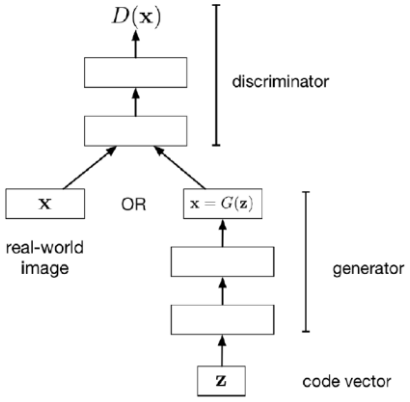
11.2 GANs

Idea: Train two different NNs:

- Generator network G tries to produce realistic samples

- Discriminator network D tries to figure out if sample is real or by G

G tries to fool D. Both networks trained in sync.



11.2.1 Training GANs

Train jointly in minimax game:

$$\min_{\theta_g} \max_{\theta_d} \left[E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

- Discriminator D_{θ_d} tries to maximize objective s.t. $D_{\theta_d}(x)$ (real samples) is close to 1, and $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ (fake samples) is close to 0.
- Generator G_{θ_g} tries to minimize objective s.t. $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ is close to 1 (thus fooling discriminator)

Alternate between gradient descent for both networks.

Saturation problem: Realistically, initial fake samples are so poor that D saturates, thus loss for G is in the exponential tail of D's sigmoid, and G will receive zero gradients.

Solution: Use different gradient descent for generator. Don't minimize discriminator correctness, maximize likelihood that discriminator is incorrect. → higher gradient signal for bad samples.

Training algorithm: TODO (11-24)

11.2.2 Evaluation

Properties:

- Sample quality (real looking images?)
- Diverse samples (more than one image generated?)
- Generalization (not only memorizing images?)

Scores:

- Inception score: Use pretrained classifier to classify samples, compare label distribution with original dataset.
  - Captures sample diversity and quality, but not diversity (11-28)
- Fréchet Inception Distance: Compare feature distribution instead of labels
- Nearest Neighbor: Search nearest neighbors of generated image among training images

11.3 Deep Convolution GAN

(DCGAN).

Problem with GAN training: Either oscillations or mode collapse (generator only focuses on small subset of samples).

Various architecture rules were defined to fight that (11-37)

11.4 Conditional GAN

Sampling from high-dimension distribution only useful for synthesis. Conditional distribution more suited for e.g. next frame prediction, image in-painting, style transfer or image-to-image translation.

11.5 Image-to-Image Translation

"Translate" images pixel-to-pixel (Pix2Pix), calculate realistic photos from edges, gray-scale images to color, …

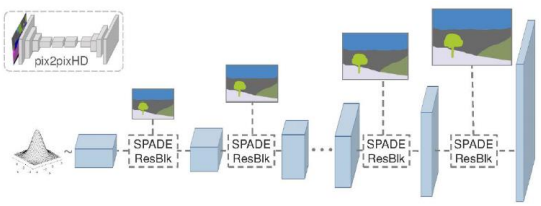
Pix2Pix: Generator uses DCGAN architecture, but with skip-connections over bottleneck, Discriminator is a regular ConvNet. Drawback: Requires pairs of samples with pixel-to-pixel correspondence.

Cycle GAN: We don't have samplepairs with pixel-to-pixel correspondence, but have two densities that we want to switch between (image of apples, image of oranges).

- Train 2 different generators, one for $A \mapsto B$ and one for $B \mapsto A$.
- Use discriminator nets to make sure samples suit their class
- Generates need to be cycle-consistent: Mapping an image to B and back to A should yield the original.

11.6 Advanced GANs

- BigGAN: Larger models to get performance boost on large image batches. Becomes unstable at later iterations
- SPADE: Synthesize image that follows underlying semantic mapping. Similar to Pix2Pix, but no bottleneck and no encoder, instead inject semantic mask at various scales into generator.



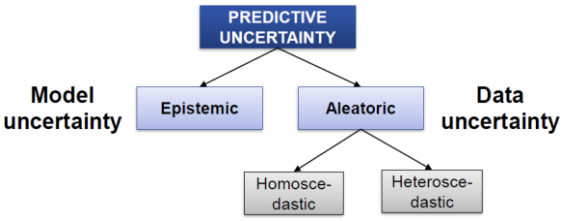
Take-home messages: 11-81

12 Advanced Topics

12.1 Uncertainty

We don't want catastrophic results if classification outcome is uncertain.

Caused by *noisy data, imprecise measurements or unforeseen data*.

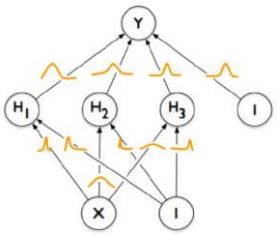


- Aleatoric uncertainty: Caused by data. Irreducable, but can be learnt!
  - Homoscedastic: Same for all inputs, e.g. imprecise sensor
  - Heteroscedastic: Uncertainty varies with input, e.g. noisy data
- Epistemic uncertainty: Caused by model, reducable. Unforeseen data.
  Approaches: Bayesian NN (use distribution), Ensemble

Note: Softmax causes overly optimistic accuracies (outputs are adapted to sum up to one).

12.1.1 Bayesian Neural Networks

Weights have distribution associated, output of the net is another distribution instead of pseudo-probability.Every neuron then pulls a probability of the distribution.



Inference is much more complicated, thus approximations are used.

12.1.2 Dropout and BNNs

During training, ignore random fraction of nodes per iteration.

Approximates Bayesian Neural Networks and can quantify epistemic uncertainty. (TODO how?)

Output statistics: Mean used for prediction, Variance used for epistemic uncertainty.

12.2 Zero-Shot Learning

Task: recognize classes that were never encountered.

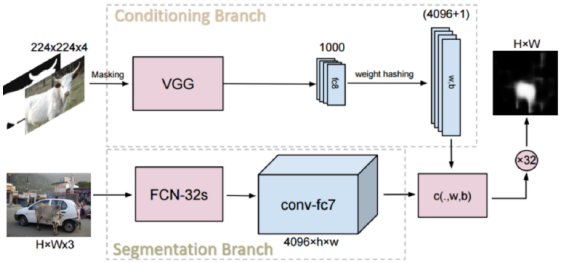
Idea: Use semantic embedding, e.g. attribute descriptions (has_stripes=1, has hooves=1 => Zebra. Has_hooves=0 => Tiger).

Attribute discovery: Mine online articles (wikipedia) that describe attributes (e.g. small tail) and match those with instances (e.g. wombat).

12.3 Few-Shot Learning

Task covered was semantic segmentation. Large training dataset of segmentation-mask pairs, we want good segmentations on unseen classes.

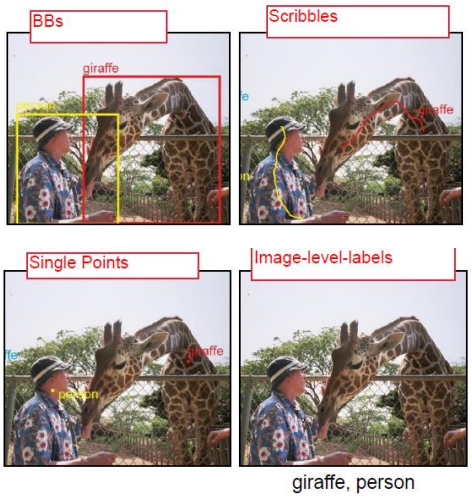
Use two branches, one for segmentation, and one for conditioning (handling new classes with few instances).



12.4 Weak Supervision

We want semantic segmentation, but don't want to spend as much effort for pixelwise annotating examples.

Idea: Rough annotations



giraffe, person

Net iteratively adapts segmentations to get proper pixelwise segmentations.

12.5 Transfer

Train model on images in one domain, that can solve the task for images of the same class, but a different domain.

Example: Train on 3D generated synthetic examples, infer on realistic photos.

Domain gap has to be adressed. Use adversarial network for discriminating between source-/target features, train domain adaption module s.t. discriminator cannot tell the difference.

Take-home Messages: 12-04 (Challenges of CV), 12-28 (Uncertainty), []

Fragestellungen auf Folien:

- 02-23
- 03-35
- 03-60
- 04-18
- 04-64ff!
- 05-62
- 08-11

Leseempfehlung:

- GoogleNet?
- 08-07

Typische Architektur für Semantic Segmentation malen und Ausgabevolumen erklären