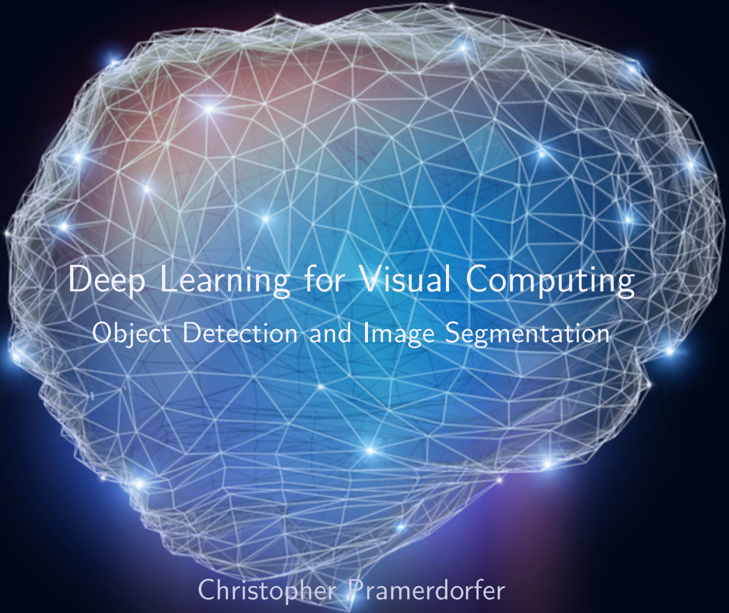


Deep Learning for Visual Computing

Object Detection and Image Segmentation

Christopher Pramerdorfer

Computer Vision Lab, TU Wien

Image by Oracle

Topics

Object detection (part 2)

- Bounding box regression
- Region Proposal Networks

Semantic image segmentation

- Fully Convolutional Networks
- Unpooling and Transposed Convolution layers

Object Detection

Methods covered so far

- Sliding Window Approach (earlier)
- R-CNN (2014)
- Fast R-CNN (2015)

We will now cover more recent / state of the art methods

Fast R-CNN vs. R-CNN

- Around 25 times faster at test time
- Similar (a bit better) performance (joint optimization)

Now finding region proposals is the bottleneck

- Can take up to two seconds depending on algorithm
- This is where Faster R-CNN comes in

Faster R-CNN integrates region proposal computation

- Using a Region Proposal Network (RPN)
- That operates on the output volume

Rest is identical to Fast R-CNN

- Around 10 times faster at test time ($\approx$ 5 fps)

Object Detection
Faster R-CNN

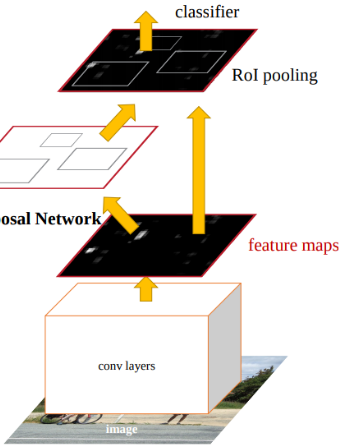


Image from [2]

How could such an RPN look like?

Should operate on the output volume (size $C_u \times H_u \times W_u$)

- Share computation for efficiency

Desired output for each location in output volume

- Is there an object? (binary classifier)
- Bounding box in input image (4D regressor)

Object Detection
Faster R-CNN

Location in output volume gives rough bounding box

- Can map from that location to input (receptive field)
- Similar to Fast R-CNN but in opposite direction

Purpose of regressor is to improve this bounding box

- Predict corrective offsets
- Again similarly to (Fast) R-CNN

Can be implemented using a Sliding Window approach

- Slide a 3×3 window across output volume (with padding)
- Train two MLPs on resulting $C_u \times 3 \times 3$ subvolumes

RPN should be translation invariant

- Objects should be detected no matter where they are
- Means shared parameters across all spatial locations
- Sounds familiar?

Such "MLPs" can be implemented with conv layers

Hidden layer : $C_u \times 3 \times 3$

- Sliding Window with shared parameters equals convolution
- Layer can be shared to increase efficiency

Output layers : $C_o \times 1 \times 1$ conv layers

- $C_o = 2$ for classification layer (object, background)
- $C_o = 4$ for regression layer (x, y, w, h)

Resulting RPN with $C_u = 256$

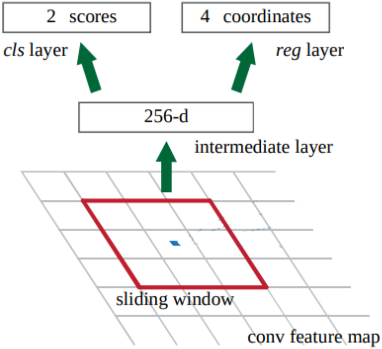


Image adapted from [2]

We have just designed a Fully Convolutional Network (FCN)

- No linear layers so independent of input size

Produces output volumes of size $C_o \times H_o \times W_o$

- H_o and W_o vary with input size
- In our case $2 \times H_o \times W_o$ and $4 \times H_o \times W_o$

Powerful architecture (more examples later)

- Performs predictions on grid over input

Object Detection
Faster R-CNN – Fully Convolutional Networks

We have "converted" linear to conv layers

- Not always sensible (image classification)
- But great for e.g. efficient object detection

Assuming $C \times H \times W$ input the following are identical

- Linear layer with n neurons (after vector conversion)
- $n \times H \times W$ conv layer (output $n \times 1 \times 1$)

Our RPN can predict only one proposal per spatial location

Faster R-CNN addresses this with anchor boxes

- k fixed bounding box configurations (size, aspect ratio)
- Align with current location and map to input

Classification and regression for each box

- $C_o = 2k$ (classifier) and $C_o = 4k$ (regressor)

Resulting RPN with $C_u = 256$

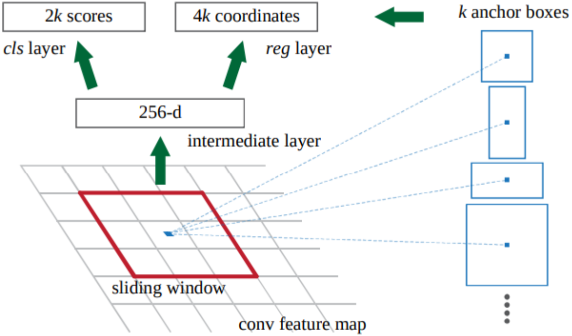


Image from [2]

How can we train this thing?

- Dataset with images, bounding boxes, and class labels
- RPN predicts $k \times H_o \times W_o$ boxes and class scores per image

We have to

- Assign ground-truth class labels to predicted boxes
- Define a bounding box regression loss
- Minimize two losses concurrently

Object Detection
Faster R-CNN

Assign ground-truth class labels based on IoU

- Maximum Intersection over (divided by) Union
- Between any ground-truth bounding box and prediction

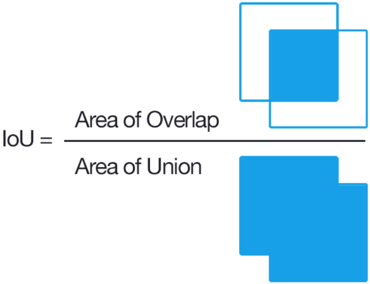


Image from Wikipedia

Assign ground-truth class labels to predicted boxes

- Object if IoU > 0.7, background if IoU < 0.3

Now for a given image (and epoch)

- Randomly sample a minibatch of 256 predicted boxes
- Compute classification loss via softmax and cross-entropy
- Compute regression loss (smooth L1) only for object boxes

Smooth L1 loss: $\ell_h(\mathbf{w}, \mathbf{x}') = \sum_i h(w_i - w_i')$ with Huber loss h

- L2 loss close to 0, L1 loss otherwise

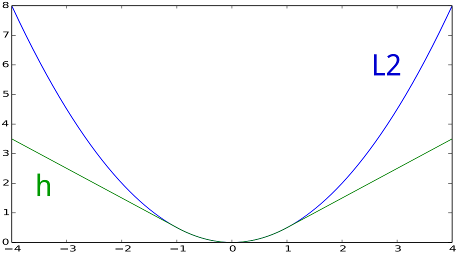


Image adapted from Wikipedia

Having defined both losses, how can we combine them?

- Simply “connect” their outputs using a summation
- And apply standard back-propagation

This works for any number of losses

- Usually with weights to balance their impact

Now we know how to train RPNs

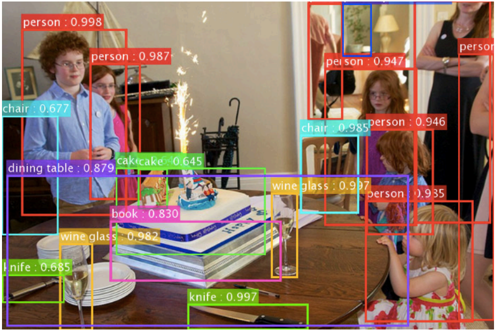
- Useful for class-agnostic object detection
- Part of several state of the art architectures

Original Faster R-CNN trains RPN and object detector in turns

- Again starting with pre-trained ImageNet classifier
- But can also be trained together

State of the art image detection performance

- With ResNet as RPN backbone



State of the art image detection performance

- With ResNet as RPN backbone

link

Faster R-CNN is quite fast ($\approx$ 5 fps)

- Still insufficient for some real-time applications

YOLO is a popular alternative

- Similar performance to Faster R-CNN
- But at least 10 times faster

We will cover YOLO v2

R-CNN family is based on image classification

- Classify region proposals

YOLO is based on regression

- Predict vectors $\mathbf{w}$ of object bounding boxes and class scores

Both share concept of k anchor boxes

- $\mathbf{w}$ have dimension $k \cdot (5 + C)$
- p, x, y, w, h (p is IoU with object)
- C class scores (no background class needed)

YOLO operates on $S \times S$ grid over input image

- Output is $k \cdot (5 + C) \times S \times S$ (one $\mathbf{w}$ per grid cell)
- Achieved using a FCN (so S may vary)

Output similar to RPN above

- But class-specific and combining both output volumes
- Only regression (L2) losses

Object Detection
YOLO – Training

Assuming $S = 3$, $k = 2$ and $C = 3$

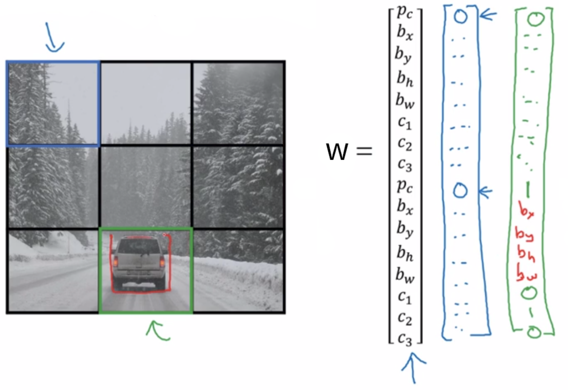


Image adapted from Youtube

Weighted sum of three L2 losses per anchor box

- Confidence loss (loss on p)
- Classification loss (loss on class scores) *
- Localization loss (loss on x, y, w, h) *

(*) Only if IoU with ground-truth box is sufficient

Object Detection
YOLO – Inference

$k \cdot S \cdot S$ detections

- Perform per-class non-maximum suppression

Image adapted from Youtube

We have used FCNs for prediction on grids

- Low grid resolution ($S = 13$ in YOLO paper)

Can we increase or even maintain the image resolution?

- Would enable per-pixel prediction
- Seems ideal for image segmentation tasks

Semantic Segmentation

- Assign class labels to pixels

Image from cocodataset.org

Can we maintain the original resolution?

Obvious solution is to avoid pooling

- But we already established this is too slow
- Also receptive field increases slowly so missing context

Any other ideas?

Given an images of size $H_0 \times W_0$ we

- Downsample using conv and pooling layers
- Then upsample again to $H_0 \times W_0$

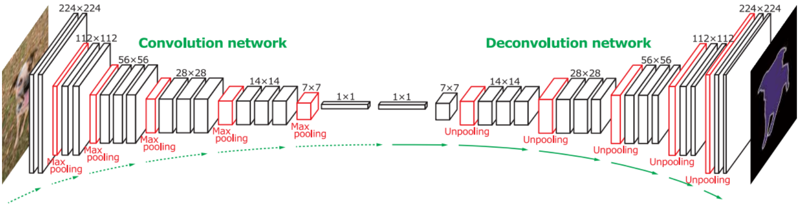


Image from [1]

Different kinds of upsampling layers exist, including

- Unpooling layers
- Transposed convolution layers

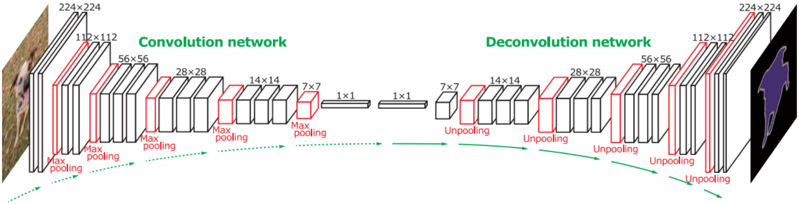


Image from [1]

Unpooling is the "inverse" of max-pooling

- Pooling layer stores $\arg\max$ for back-propagation
- Associated unpooling layer uses this information

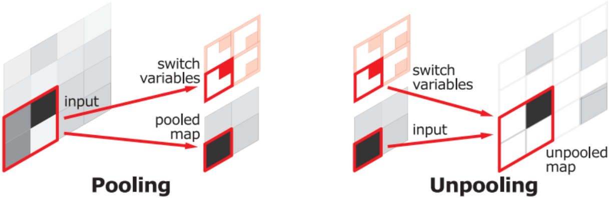


Image from [1]

Downsides of unpooling

- Requires corresponding pooling layers so less flexible
- Results in sparse reconstruction (result has holes)

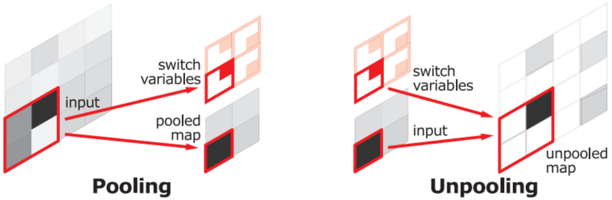


Image from [1]

Transposed convolution layers are more elegant

- Like convolution but with stride $1/s$
- Also called deconvolution layers

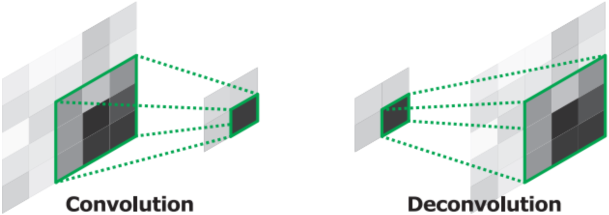
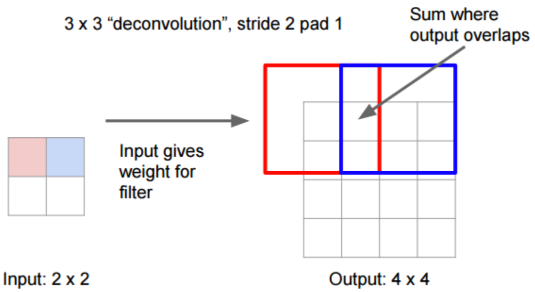


Image from [1]

Assuming $C_{l-1} = C_l = 1$ for simplicity

- Input/output channels handled as in conv layer



Adapted from Stanford slides

Essentially we

- Align top-left input and top-right of kernel
- Multiply kernel with input value (output is 3×3)
- Move kernel by 1 in input and s in output and repeat
- Sum values where filters overlap
- Finally perform "inverted padding" (cropping)

This is identical to backward pass of conv layer

- Also applies in the other direction

Now we can design FCNs for pixel-level prediction

- Downsampling part (what happens in image?)
- Upsampling part (where does it happen?)

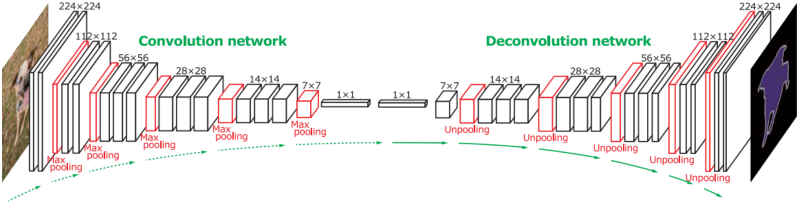


Image from [1]

Popular FCNs include skip-connections

- Between downsampling and upsampling layers
- Sum or concatenation along channels (e.g. UNet)
- Improves resolution of predictions (less upsampling required)

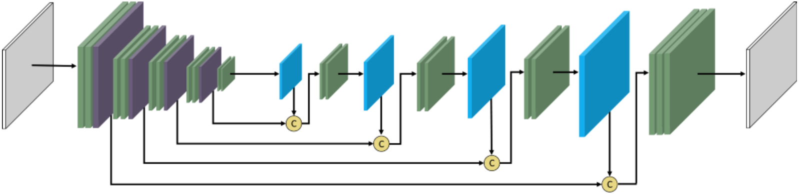


Image from medium.com

We already know how to train such networks

- Just define a suitable loss (e.g. per-pixel cross-entropy)

Image from cocodataset.org

Example application by a colleague

Image from remove.bg

Bibliography

[1] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. *Learning Deconvolution Network for Semantic Segmentation*. ICCV (2016).

[2] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. CVPR (2016).