Seminar on Machine Learning, University of Regensburg

Image Segmentation

with Mask R-CNN

Gesina Schwalbe

7th May 2018

Supervisor: Justin Noel

Contents

1 Definition and Goals

1.1 Problem

1.1.1 General Goals

The problem of image segmentation is to find

- bounding boxes
- classification of each box
- pixel-mask for each box

for a given image that contains none, one, or several objects—if possible with more than 1 fps.

Datasets Some common labeled datasets for the image segmentation task, mostly associated with a couple of specific official challenges, are

- COCO

Figure 1: Application of image segmentation: Mapping Challenge solution [7]

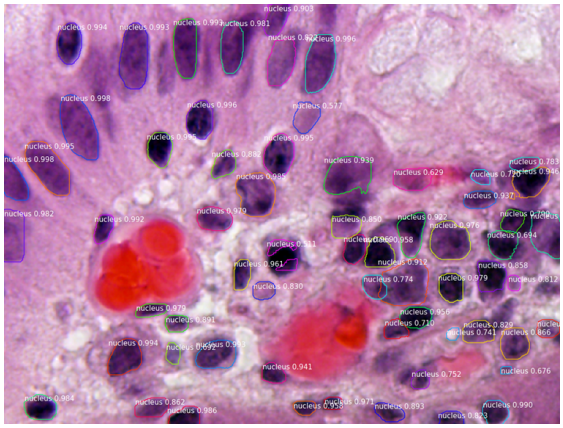


Figure 2: Application of image segmentation: Nucleus segmentation [10]

- ImageNet

Have a look at one of these datasets for further impressions on what the output is expected to look like.

1.1.2 Applications

- Live detection of signs, obstacles in traffic (example video)
- Automatic street map enhancement (see Figure 1)
- Automatic evaluation of microscopy images (see Figure 2)

1.2 Architectual Requirements

- Master natural images: large nets
- Learn and process fast/efficiently (currently best: 32 h wt. 8 GPU learning; 5 fps inference):
  - Special network components instead of only fully connected layers
    (CNN, ResNe(X)t, FPN)
  - Share many features (FPN, RPN)
  - Parallelize tasks/components

2 Architecture Components

2.1 Convolutional Networks

2.1.1 Key Features

The main ingredient to convolutional networks, the convolutional layers, are very good at detecting and summarizing local features in large data-spaces (e.g. corners, edges, patterns in images—i.e. self-learned image-preprocessing). The key features of convolutional neural networks (CNNs) are

- Feedforward neural network with only local connections
- Massive weight sharing
- Translation invariance (see anchor boxes later)
- (Often) Downscaling of feature space:
  - More translation invariance
  - Multiple scale feature-spaces available (see RPN anchor pyramid and FPN later)

2.1.2 Convolution Operation

The input space of a convolution is $\mathbb{R}^{n_1 \times n_2 \times \cdots n_3}$. Each point (image) is a discrete $n_1 \times n_2 \times \cdots n_r$ coordinate grid of values (pixels). The induced measure of distance of coordinates gives a notion of surrounding area for each coordinate.

A convolution in machine learning is the following:
Linear map $\mathbb{R}^{n_1 \times n_2 \times \cdots n_3} \to \mathbb{R}^{n_1 \times n_2 \times \cdots n_3}$ described by

- a fixed sliding window shape i.e. a $k_1 \times \cdots \times k_r$ grid (usually a square)
- a window of that shape with a weight value at each coordinate, called the *kernel*.

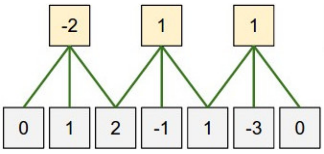


Figure 3: 1D Convolution with stride 2; Note that this is the same as a 1D Convolution without stride and then forgetting every second pixel [8].

The output at a coordinate is given by the scalar product of the values in its surrounding box with the kernel. The border coordinates need special treatment (e.g. continuation by 0).
(Animation)

2.1.3 Convolutional Layer

A convolutional layer is effectively a locally connected layer, i.e. each output pixel receives input only from nearby input pixels, that in addition heavily shares parameters. It consists of several convolution operations in parallel, each followed by an activation function (usually non-linear) that is applied to every pixel.

Convolution Hyperparameters:

input dimension without number of filters; in keras choose e.g. a Conv2D layer for 2D images (the 1–3 channels are the input filters)

size of kernel = weight-window; usually a 3×3- or 5×5-square

padding variant = border treatment; usually "same", i.e. extend the input with zeros s.t. `output_dim` = `input_dim`

number of filters = number of parallel convolutions; perform i convolutions in parallel producing an i-times larger output

stride = downsampling rate; skip all except every ith coordinate (see Figure 3)

dilation = upsampling rate; add i virtual coordinates with fixed value inbetween

activation function e.g. linear, ReLu

Convolution Weights: kernel values, bias

How input filters are handled: If the input has more than one filter, e.g. the color channels for images, then each output filter is made up by

1. Apply a separate kernel to each input filter
2. Sum up the results of all input filters to this one output filter

Thus, the output dimension will be the input dimension plus the number of output filters (and *not* additionally plus the number of input filters).

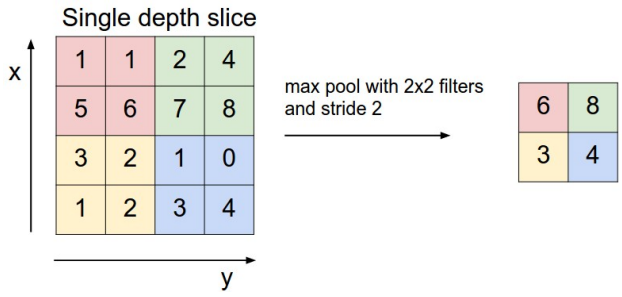


Figure 4: 2D-MaxPooling visualisation [8]

2.1.4 Pooling Layer

Pooling is a way to downscale an image respectively summarize local sets of features. Given a window size, the input to the pooling layer is disected into windows of this size (with or without overlap). Those are then processed using a fixed operation, e.g. maximum or average, to yield one output value per window. This

- reduces the number of features drastically,
- makes the network translation invariant up to the window size, and
- prevents overfitting to some extend.

Usual pooling functions:

- Average Pooling (linear) gives the average value of each window.
- MaxPooling (non-linear) gives the maximum value of each window (see Figure 4).

  MaxPooling cheaply introduces one more non-linearity to the convolutional setup, and usually performs better than average pooling (according to Wikipedia).

2.2 Deep CNNs: ResNet and ResNeXt

For details see [1].

2.2.1 (Deep) CNN

A deep convolutional neural network (DCNN, or simply CNN or ConvNet) is a feedforward neural network consisting of:

Feature extraction backbone a stack of alternating

- convolutional layers, and
- pooling layers

Feature interpretation a stack of fully connected neural network layers usually ending in a softmax layer for classification

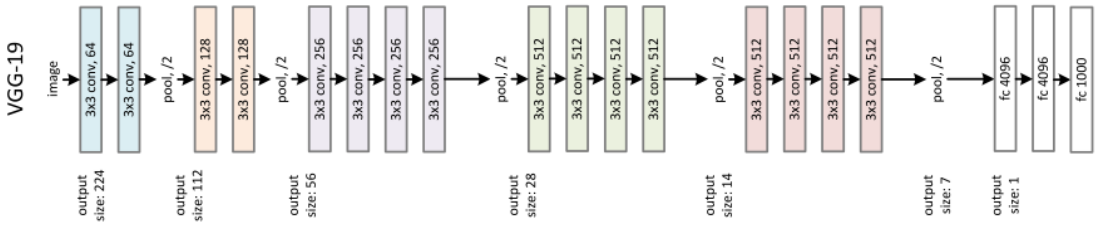


Figure 5: VGG-19, a common ConvNet architecture developed by the Visual Geometry Group at Oxford University and predecessor of ResNet. Mind the alternating convolutional and pooling sections as well as the fully connected layers at the end. [1]

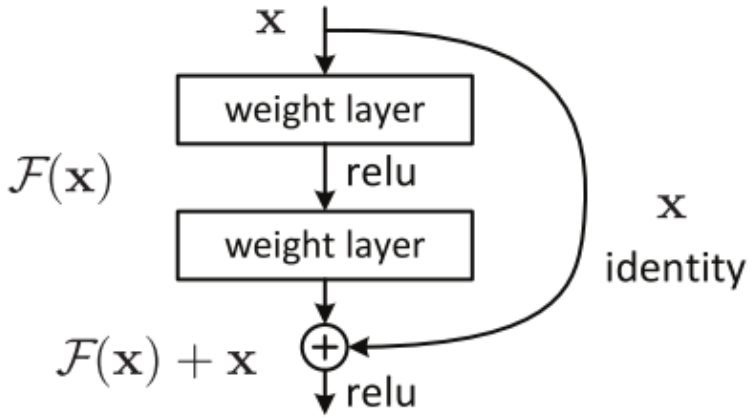


Figure 6: Residual component of a RCNN network. Up to two layers get a parallel identity shortcut [1].

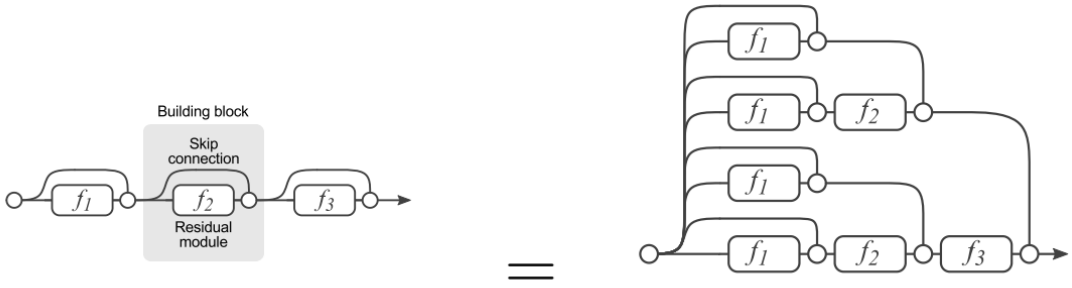
See e.g. Figure 5 for a common CNN architecture. Neural networks consisting mainly or solely of CNN parts are sometimes called fully convolutional neural networks (FCNN).

2.2.2 Residual CNNs: ResNet

Residual convolutional networks introduce identity shortcuts to circumvent the vanishing gradient problem of deep networks, see Figure 6.

The actual trick is to provide alternative shorter "paths" for all weights (see Figure 7). Intuitively this means that features detected in any stage are available in any other stage.

A nice description of this network architecture using shortcuts is, that not every layer introduces new features, but the first layer already roughly correctly describes the needed features, and any proceeding layer adds a little correction to that. The later the layer, the smaller the corrections of the feature map will have to be. This is contrary to the usual



(a) Conventional 3-block residual network (b) Unraveled view of (a)

Figure 7: Impact of residual blocks within a network on paths through the network [1].

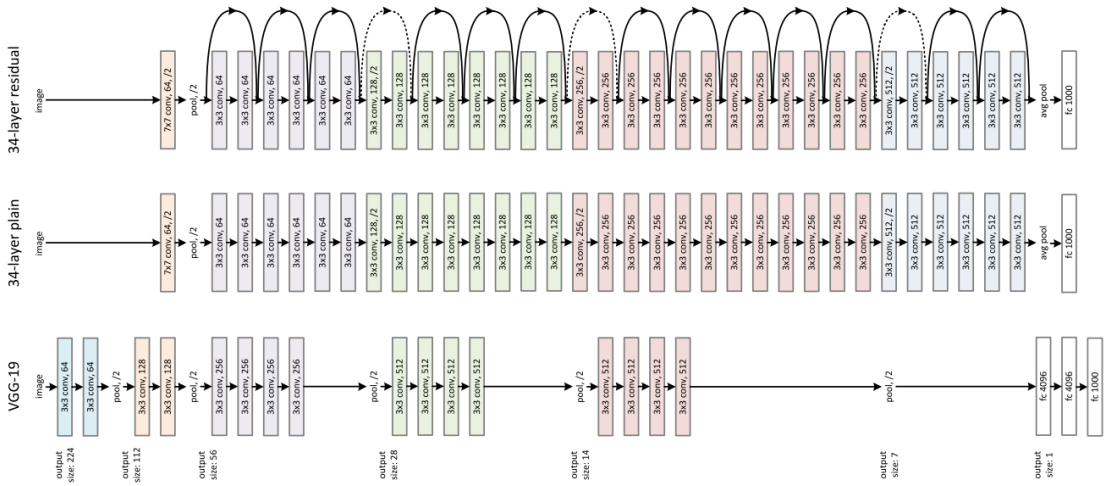


Figure 8: Comparison of VGG and ResNet: Residuals enable buiding much deeper convolutinal networks [1].

architectures, where in every stage the features get processed to yield some completely new set of features.

Best performance for residual modules was found for shortcutting two layers.

ResNet is an example of a specific such network (usually 50 or 101 convolutional layers). See Figure 8 for a comparison of ResNet with the previously discussed VGG network.

ResNeXt splits the shortcutted parts into several parallel parts (Figure 9) to get even better performance without further gradient vanishing.

2.2.3 More Feature sharing: FPNs

Feature Pyramid Networks do not only use residuals to make earlier features more accessible in the final prediction block, but apply a prediction block at every feature stage (see Figure 10 and Figure 11). This makes the network more invariant to the scale of

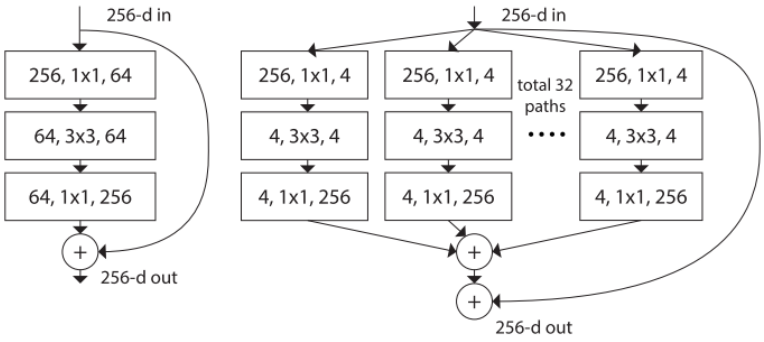


Figure 9: Comparison of a normal residual block with one or ResNeXt. ResNeXt shortcuts many parallel branches [1].

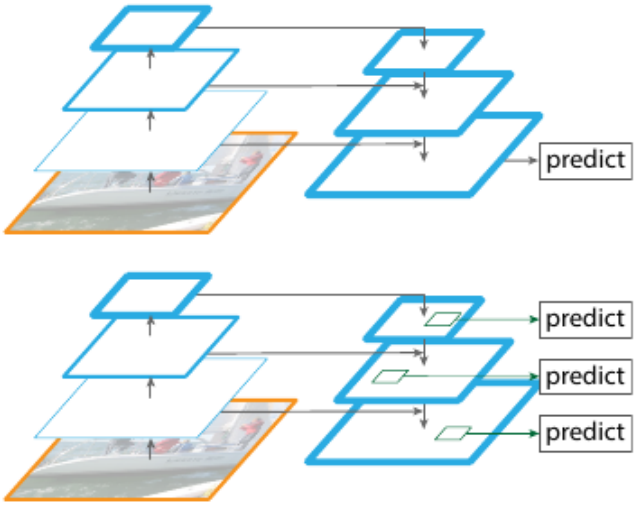


Figure 10: FPN setup principle [12]

the feature to be detected.

3 Model

3.1 Overview

3.1.1 Predecessor Problems

Object Classification Assume one object per image to classify. The key point here was to use sufficiently powerful architectures (FCNNs, ResNets, FPNs) to master the complexity of natural image features.

Object Detection

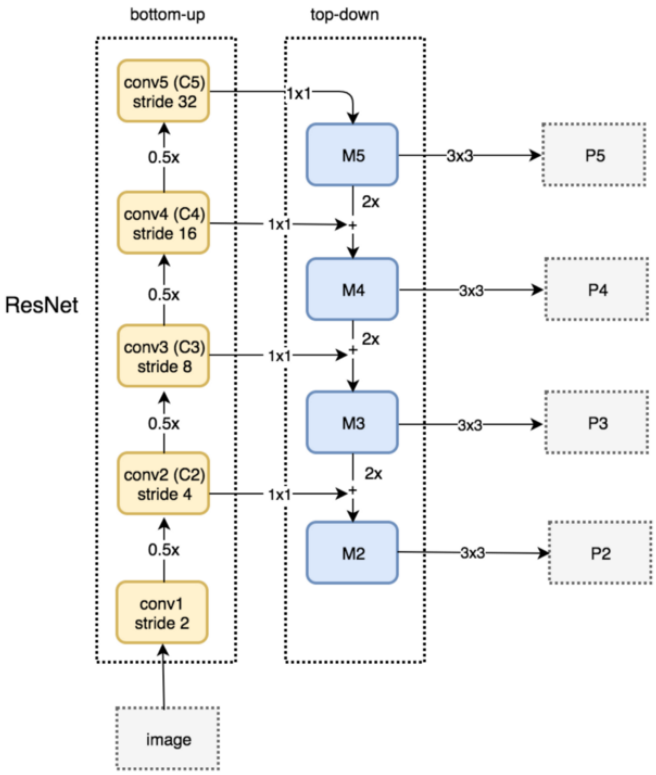


Figure 11: FPN with ResNet [18]

- (intelligently) choose windows of the image
- classify each window (including class "no obj")
- maybe enhance the window selection

The key point of Object Detection was to understand, that it can be done as a window-wise version of Object Classification. Main advances were how to effectively and efficiently choose the windows and how to share feature detection between overlapping windows.

3.1.2 Components

1. Convolutional backbone: extract important features
2. Region proposal; *in parallel:*
   - Window objectness scoring
   - Window correction
3. Frontend; *in parallel:*
   - Classification
   - Masking
   - Bounding Box Optimization

3.2 Convolutional Backbone

Goal extract features

Architecture The architecture should be chosen as for a convolutional backbone of an analogous Object Detection task, e.g. ResNet with FPN for natural image processing. See 4.2.

3.3 Region Proposal Network

3.3.1 Predecessors/Alternatives

See [6] for a more detailed overview.

Pixel Merging (e.g. SelectiveSearch) Merge pixels with neighbors to objects by manual or learned rules, starting at more or less intelligently chosen starting pixels. The computation is usually very costful.

Window scoring (e.g. Objectness) Obtain a fixed or preselected set of candidate windows and learn/apply an objectness scoring algorithm. The localisation accuracy is usually low and the cost is proportional to the number of window candidates. Scale invariance is usually ensured by conducting the process several times for differently sized windows/image resize scales.

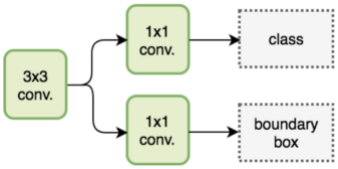


Figure 12: RPN components [18]

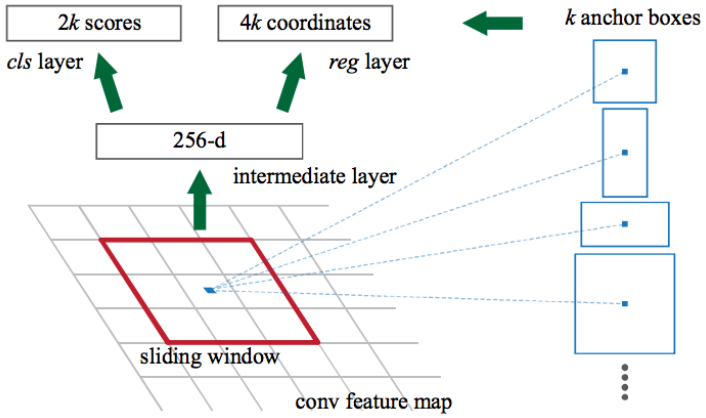


Figure 13: Anchor shapes per center point i.e. feature map pixel [15].

Separate NN (e.g. Multibox) Separately train a complete neural network for region proposals. Usually very costful to train.

3.3.2 Main Ideas of the Mask R-CNN/Faster R-CNN Approach

Window scoring with enhancements:

Decoupling of classification and window proposals

All Scales at once using different candidate window shapes

Bounding box correction in *parallel* to scoring

Excessive weight sharing amongst same shapes

RoI-Pooling i.e. use convolutional features from backbone and pool each proposal window to fixed size

3.3.3 Architecture Overview

Shared Conv Layer Sliding window on feature space with fixed set of anchor shapes per window (=base box shape proposals), see Figure 13

cls*, *reg Per window and anchor shape do in parallel (see Figure 12)

objectness score (*cls*) This is analogous to the window scoring method and yields for each region a score on how likely it is that this region contains an object. The main difference is the *high weight sharing*: Anchors of the same shape share their scoring algorithm, thus ensuring high translation invariance (it does not matter, in which corner of the image the object is) and high scale invariance (choose anchor shapes of different scale).

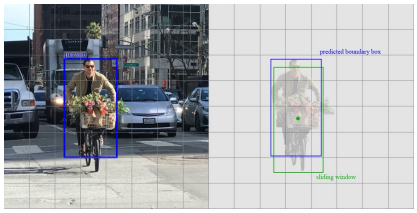


Figure 14: Example of a bounding box correction by the *reg* part of a region propsal network. [18]

coordinate correction (*reg*) This part tries to overcome the low localisation accuracy of usual window scoring and thus enables to use much less candidate windows (anchors), as the shape only has to fit very roughly. See Figure 14.

The main idea behind this parallel approach is, that with a well enough feature space output of the backend the questions

- *Is* there an object of roughly that shape?
- If it was an object, what *exact shape* would it have?

can actually be answered independently and also independent of the later object classification (class-agnostic region proposals).

Proposal Layer Select best proposals

3.3.4 Coordinate Encoding

All box coordinates are normalized, i.e. $x = \frac{x_{\text{abs}}}{\texttt{image\_width}}$, $y = \frac{y_{\text{abs}}}{\texttt{image\_height}}$.

Box coordinate (= Proposal Layer output) encoding, all normalized:

$$\begin{pmatrix} x_1, y_1 & \texttt{\# upper left corner} \\ x_2, y_2 \end{pmatrix} \texttt{\# lower right corner}$$

Coordinate correction (=*reg* output) encoding:

$$\begin{pmatrix} dx, & \texttt{\# } \frac{\texttt{centerpoint x-difference}}{\texttt{anchor width}} \\ dy, & \texttt{\# } \frac{\texttt{centerpoint y-difference}}{\texttt{anchor height}} \\ \log(dw), & \texttt{\# } \log\left(\frac{\texttt{ground-truth box width}}{\texttt{anchor width}}\right) \\ \log(dh) \end{pmatrix} \texttt{\# } \log\left(\frac{\texttt{ground-truth box height}}{\texttt{anchor height}}\right)$$

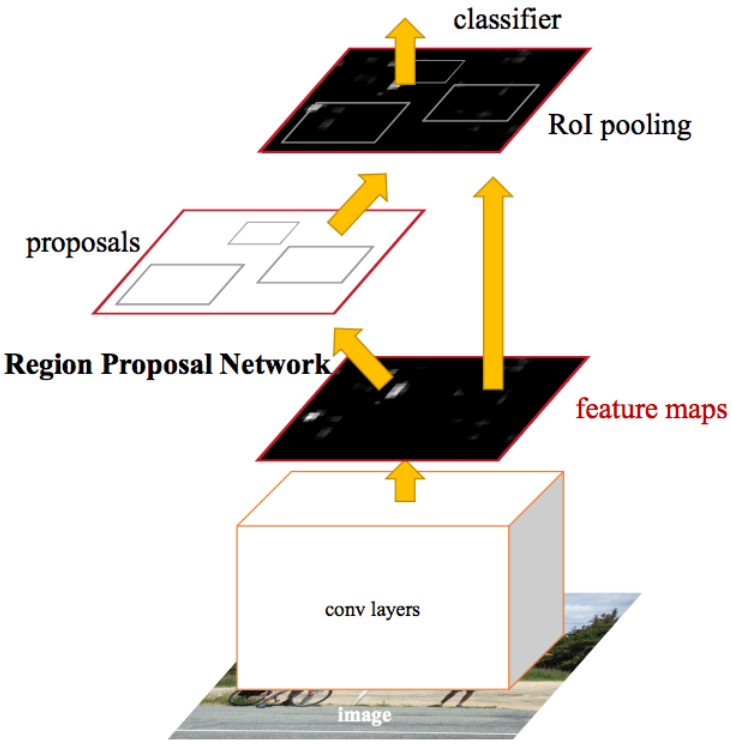


Figure 15: Location of the RPN component within the network [15]

3.3.5 Metrics

The metric for measuring how well a proposal window fits for a real bounding box is the *intersection over union ratio* (IoU). For two measurable subsets A, B of $\mathbb{R}^2$ this is defined as

$$\mathbf{IoU}(A, B) \coloneqq \frac{\text{Intersection Area}}{\text{Union Area}}$$

3.3.6 Labels

For each anchor we assign an objectness label, and if this is positive, a corresponding ground-truth bounding box (else the ground-truth bounding box is set to all 0):

1: object with ground-truth box b if

- it has best **IoU** for b, else if
- **IoU** for $b \geq 0.7$

-1: no object if not positive and **IoU** ≤ 0.3

0: neutral else (excluded from training)

All neutral boxes get completely excluded from training respectively from the loss calculation.

Balance positive and negative boxes for training! This is important, as there usually will be by far more non-object anchor boxes than positive ones. Balancing can e.g. be done by setting most of the actually negative anchors to neutral s.t. they get excluded from training.

3.3.7 Architecture Details

Sliding window This is implemented by a shared convolutional layer that naturally gives one value per sliding window (= kernel) location. The padding is "valid" to exclude anchors crossing image borders (see Coordinate Correction details below and *reg*-Training).

Objectness classification Conv layer with

- 1×1-sized kernel
- 2-class softmax activation: (non-object score, object score)

Loss: crossentropy for non-neutral anchors

Coordinate correction Conv layer with

- 1×1-sized kernel

- $4\times$(number of anchors) filters: (dx, dy, dw, dh) coordinate correction for each anchor

Loss: smooth L_1-loss[1]

The border-crossing anchors should be excluded because:

> "[They would] introduce large, difficult to correct error terms in the objective, and training does not converge." [15]

This exclusion can simply implemented by a "valid"-padding for the sliding window as described above; or alternatively set their ground-truth objectness score to neutral (thus they are ignored during training).

Proposal selection

1. Trim to N best-object-scored anchors.
2. Apply coordinate correction.
3. Clip boxes. This is necessary, because the coordinate corrections may well exceed the sliding window size, in which the original anchor lies. This makes sense, because one can suggest the size of an object without seeing it—respectively its surroundings—completely. However, anchor boxes exceeding the image bounds by this need to be clipped.
4. Non-maximum suppression:
   a) Sort by score.
   b) Reject boxes which have high IoU with better scored ones.
   c) Trim to best `num_proposals` proposals.

3.4 RoI-Pooling

Region of Interest (RoI) pooling is the process of resizing the output of a proposed window to the frontend's input shape. This is done by uniformly splitting the window into rectangular regions, one for each desired output pixel, and pooling each rectangle to this one pixel. As this means downscaling, all of the window outputs have to be larger than the frontend's input shape.

3.4.1 Standard RoI-Pooling

The usual RoI-Pooling method respects pixel bounds for selecting the rectangles, thus the size of the rectangles may vary by one pixel in width or height in the end, see Figure 16.

[1] The smooth L_1-loss takes the smoothed out absolute value

$$|d|_{\text{smooth}} := \begin{cases} 0.5 \cdot d^2 & \text{d=1} \\ |d|_1 & \text{d>1} \end{cases}$$

Other than the usual absolute value this is differentiable and the loss is more robust to outliers than L_2.

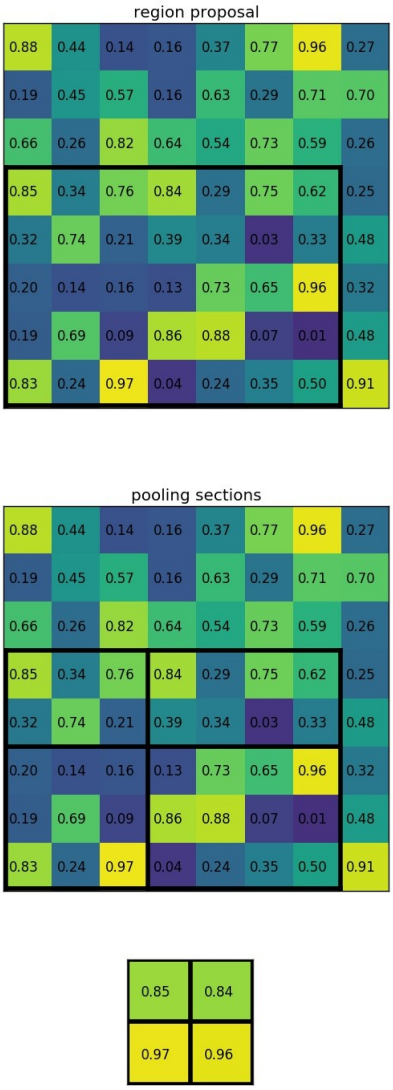


Figure 16: Visualization of Region of Interest Pooling. From top to bottom: Selection of a window; splitting of the window into roughly equal rectangular sections (round half width/height to full pixels); Max-Pooling output of the segmentation (Wikimedia Commons)

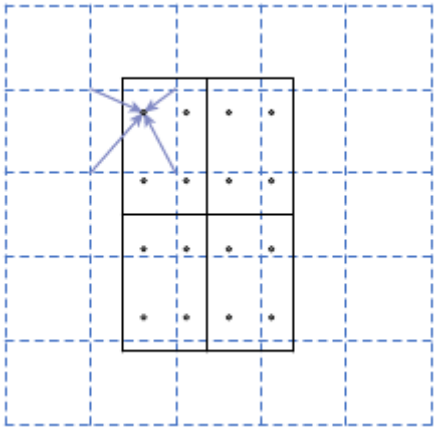


Figure 17: Visualization of RoI-Align: A pixel on the output feature map is bilinearly interpolated from nearby pixels in the input feature map. [4]

3.4.2 RoI-Align

RoI-Align suggests to avoid any cropping/rounding when disecting the windows output into rectangles for pooling. This then results in equally shaped rectangle pooling regions. This can be done by calculating the rectangle values via bilinear interpolation instead of cropping to full pixels (see Figure 17). Using RoI-Align, the feature maps from the bounding boxes evaluated by the frontend are more accurately aligned with the original image and the masks get better [4].

3.5 Frontend

3.5.1 Main Ideas

Decouple

- classification, bounding box optimization, and masks;
- mask predictions for the different classes

3.5.2 Architecture

See Figure 18 for a quick overview.

Classification fully connected layers ending in softmax (include class “no object”)

The network should look like the network head used for the corresponding object detection task, see Backbone Pretraining.

Loss: multinomial crossentropy

Mask generation

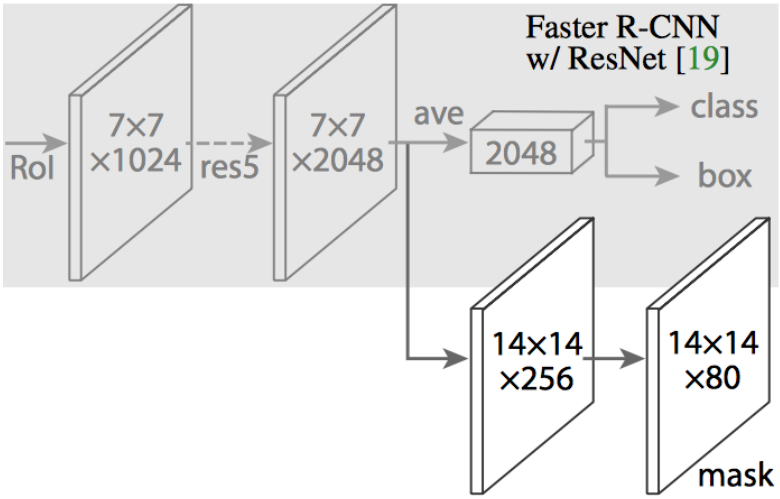


Figure 18: Frontend architecture overview; in this case a ResNet-FPN backend is assumed. Note the three mostly independent prediction branches. [4]

1. Few (1–3) Conv Layers, maybe with upscaling parts
2. Conv Layer with
   - a filter for each class
   - sigmoid activation

   This means, as for the RPN, classification and mask (= per pixel scoring) are handled independently. Tests showed, this works much better than sharing filters for all classes and apply a softmax with multinomial cross-entropy loss, because here mask results from several classes influence each other [4].

Loss: binary cross-entropy

Bounding box regression Linear regression

4 Training

4.1 Overview

Even though the training could be conducted on the complete network at once, this is not advisable as the different components would influence each other heavily. Thus training is splittet into several steps:

1. Backbone
2. RPN
3. Alternating training of RPN and Frontend

4.2 Backbone Pretraining

To train the backbone, a separate ConvNet model is set up with the convolutional backbone to train as backbone and a fully connected frontend.

Separate ConvNet Training Model

1. Backbone (Conv & Pooling)
2. Dense Layers
3. Classification Softmax-Layer where the "no object" class should already be respected and trained.

Training Input

inputs one-object images (ca. the size of later bounding boxes)

labels the single objects' labels

4.3 Alternating Training

One further special thing about image segmentation with a region proposal network is the preferable training routine, where the RPN and the frontend are trained alternately. The proposed order is:
(after RPN is pretrained)

1. Frontend (RPN fixed, backbone not shared)
2. RPN (frontend fixed, shared backbone fixed)
3. Frontend (RPN fixed, shared backbone fixed)
4. *... not much further improvement with more repetitions*

5 Implementation Review

5.1 Source Code

5.1.1 Example Sources

- Caffe2 reference implementation by facebookresearch: [9]
- Keras implementation by matterport: [10]
- Example for handwritten number detection using autogenerated data based on MNIST: github

5.1.2 Keras Implementation Specialties

- Use the functional API!
- Custom layers:

  Loss Layers custom losses

  Proposal Layer select RPN proposals

  RoI-Pooling Layer reshape proposals and mask labels (e.g. use tensorflow function `tf.crop_and_resize()`)

- Separate models for training and inference: Since in keras a specified loss will be applied to all outputs, one needs to use the more internal `model.add_loss()` function as a workaround. This function accepts a layer whose output will be added to the overall loss. As the loss layers need the ground-truth labels as inputs, one has to define separate models for training and inference and initialize the inference model with the trained weights of the other one.

5.2 Lessions learned

- Always double-check and note down tensor/array dimensions; mind the padding for convolutions.
- Always have a look at *all* input and output data:
  - Do they roughly make sense, e.g. do positive classes have different probability output than negative ones?)
  - Are there error patterns?

  Visualization is your friend. But it will definitely contain bugs, too.
- Always double-check and test your algorithms.
- Always double-check your parameters: Do they fit for your data?
- Convolution is very geometric: Depict your sliding windows, anchor shapes, and downscaling factor(s) to check whether they make sense with your data/object sizes.
- Directly document and update all input and output formats of functions—with a good IDE this makes your life much easier. (And, of course, follow the master rule: Keep your code clean and understandable.)
- Mind your RAM when optimizing data generation/tagging.
- Check your loss and validation values; Does the model actually get better?
- The deeper the network, the *much* more time training will take on CPU.

- Make data as easy (small) as possible; if a human can still classify the data in that format, the network can do so, too. E.g. grayscale instead of several color channels.

References

[1] Vincent Fung. *An Overview of ResNet and its Variants*. URL: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

[2] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). eprint: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). eprint: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[5] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. "Learning non-maximum suppression". In: *CoRR* abs/1705.02950 (2017). arXiv: 1705.02950. URL: http://arxiv.org/abs/1705.02950.

[6] Jan Hendrik Hosang et al. "What makes for effective detection proposals?" In: *CoRR* abs/1502.05082 (2015). arXiv: 1502.05082. URL: http://arxiv.org/abs/1502.05082.

[7] https://github.com/crowdAI. *Mask-RCNN baseline for the crowdAI Mapping Challenge*. URL: https://github.com/crowdAI/crowdai-mapping-challenge-mask-rcnn.

[8] https://github.com/cs231n. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/.

[9] https://github.com/facebookresearch. *FAIR's research platform for object detection research, implementing popular algorithms like Mask R-CNN and RetinaNet.* URL: https://github.com/facebookresearch/Detectron.

[10] https://github.com/matterport. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. URL: https://github.com/matterport/Mask_RCNN.

[11] https://github.com/unsky. *Feature Pyramid Networks for Object Detection*. URL: https://github.com/unsky/FPN.

[12] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.

[13] Dhruv Parthasarathy. *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. 2017. URL: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4.

[14] *Region of interest pooling explained*. 2017. URL: https://blog.deepsense.ai/region-of-interest-pooling-explained/.

[15] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). eprint: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[16] Dominik Scherer, Andreas Müller, and Sven Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". In: *Artificial Neural Networks – ICANN 2010*. Ed. by Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101. ISBN: 978-3-642-15825-4. URL: http://ais.uni-bonn.de/papers/icann2010_maxpool.pdf.

[17] *tf.non_max_suppression*. URL: https://www.tensorflow.org/api_docs/python/tf/image/non_max_suppression.

[18] *What do we learn from single shot object detectors (SSD, YOLO), FPN and Focal loss?* URL: https://mc.ai/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss/.