# Image Segmentation 2018

**with Mask R-CNN**

Gesina Schwalbe

7th May 2018
Seminar on Machine Learning, University of Regensburg
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 containing none, one, or several objects; if possible with more than 1fps.

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

- COCO
- ImageNet

[t]

Figure 1: Mapping Challenge solution

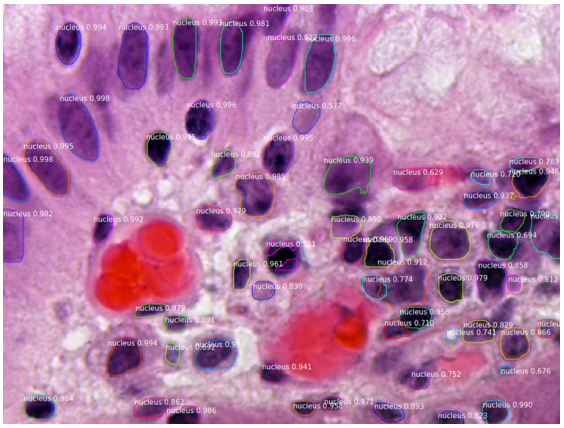


Figure 2: Nucleus segmentation

### 1.1.2 Applications

- live detection of signs, obstacles in traffic (example video)
- automatic street map enhancement
- automatic evaluation of microscopy images

### 1.1.3 Architectual Requirements

- Master natural images: large nets

- Learn and process fast/efficiently (currently best: 32h wt. 8GPU learning; 5fps 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 What

The main ingredient to convolutional networks, 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 are

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

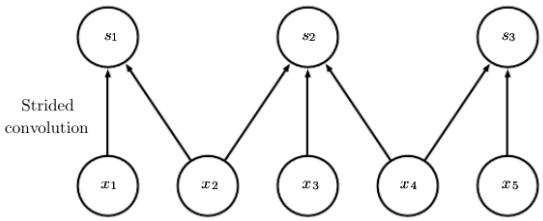
### 2.1.2 Convolution Operation

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 a

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




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

(Animation)

[t]

### 2.1.3 Convolutional Layer

A convolutional layer is effectively a locally connected layer (each output pixel gets input only from nearby input pixels) that 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:**

- size of kernel (= weight-window), usually square 3x3 or 5x5
- padding variant (= border treatment), usually "same", i.e. extend with 0 s.t. output dim = input dim
- number of filters (= number of parallel convolutions); perform $i$ convolutions in parallel producing a $i$-times larger output
- stride (= downsampling rate); skip all except every $i$th coordinate
- dilation (=upsampling rate) ; add $i$ virtual coordinates (wt. fixed value) inbetween
- activation function e.g. linear, ReLu

**Weights:** kernel values, bias

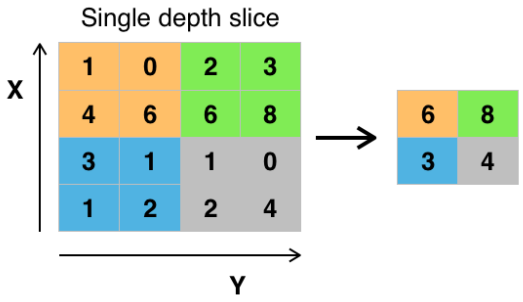



Figure 3: 2D-MaxPooling visualisation

#### 2.1.4 Pooling Layer

Pooling is a way to downscale an image resp. 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) giving one output value per window calculated with a fixed operation, e.g. maximum or average. 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

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

### 2.2 Deep CNNs: ResNet and ResNeXt

For details see [1].

#### 2.2.1 (Deep) CNN

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

**Feature extraction backbone** stack of alternating

- convolutional layers, and
- pooling layers

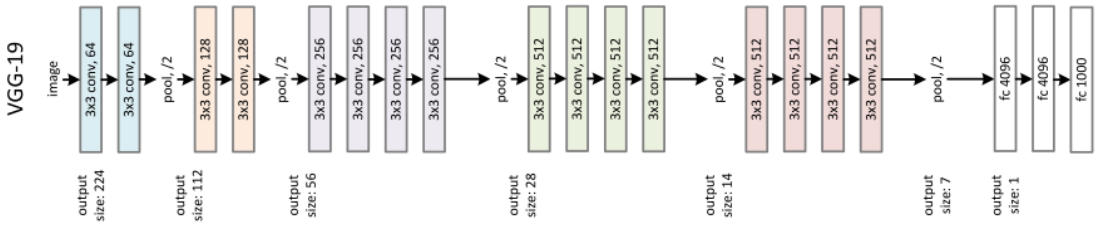



Figure 4: VGG-19, a common ConvNet architecture developed by the Visual Geometry Group at Oxford University and predecessor of ResNet. Mind the alternating convolutional and pooling sections as well as the fully connected layers at the end.

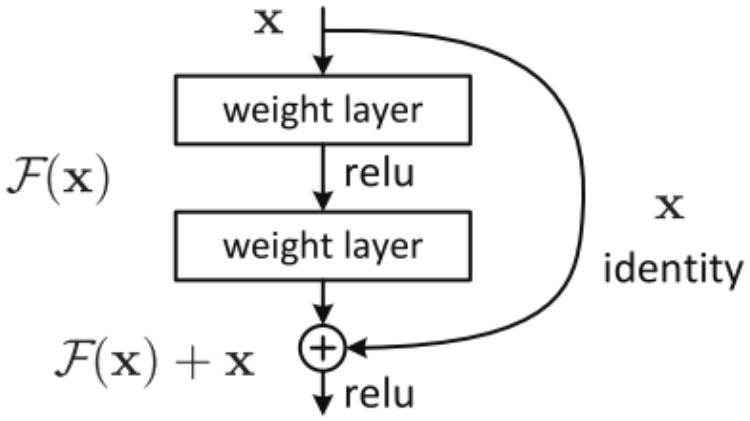



Figure 5: Residual component of a RCNN network. Up to two layers get a parallel identity shortcut [1].

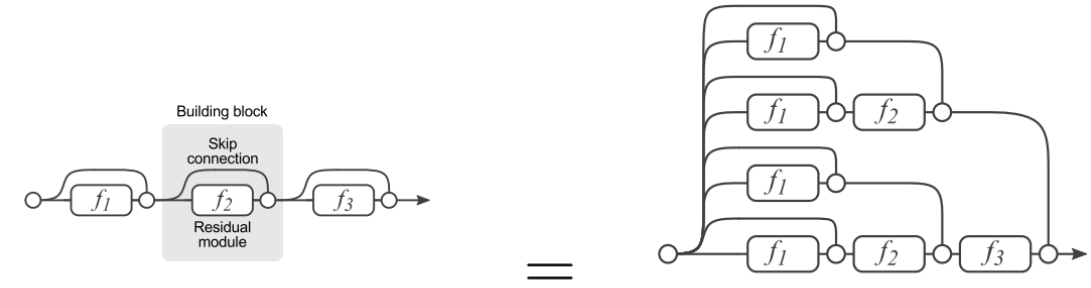
**Feature interpretation** stack of fc neural network layers usually ending in a softmax layer for classification

### 2.2.2 Residual CNNs: ResNet

Residual convolutional networks introduce identity shortcuts to circumvent the vanishing gradient problem of deep networks.

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

Best performance for residual modules was found for shortcutting 2 stacks of linearity




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

Figure 6: Impact of residual blocks within a network on paths through the network [1].

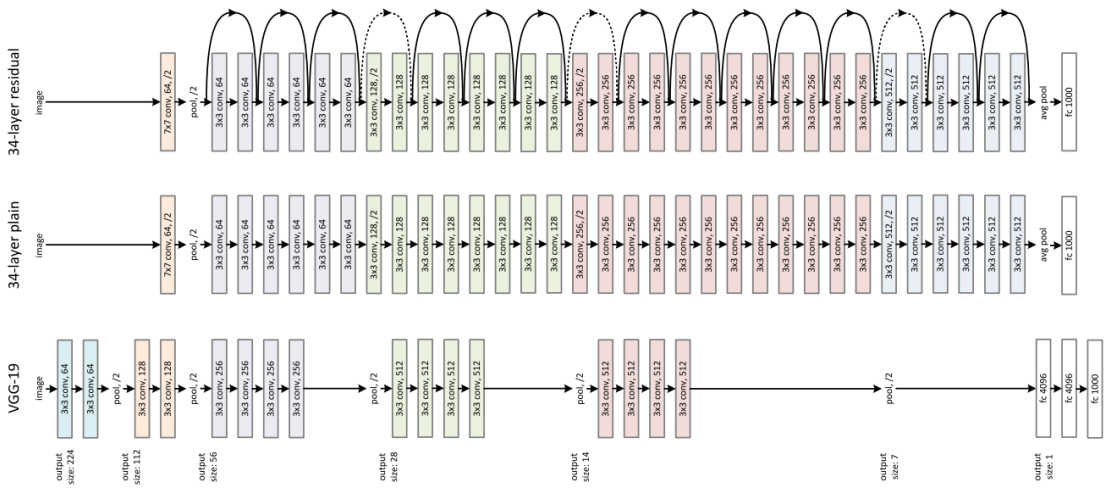



Figure 7: Comparison of VGG and ResNet: Residuals enable buiding much deeper convolutinal networks [1].

and non-linearity. ResNet is an example of a specific such network (usually 50 or 101 convolutional layers).

ResNeXt splits the shortcutted parts into several parallel parts to get even more 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. This makes the network more invariant to the scale of the feature to be detected.

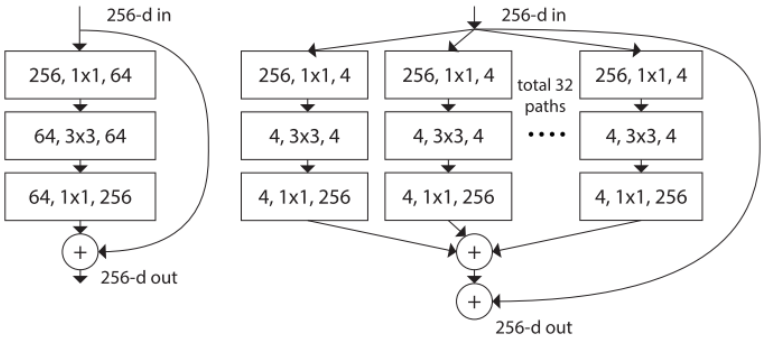



Figure 8: Comparison of a normal residual block with one or ResNeXt. ResNeXt shortcuts many parallel branches [1].

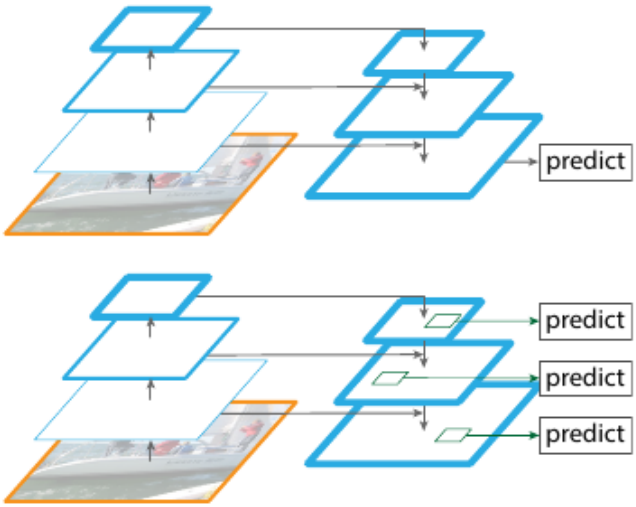



Figure 9: FPN setup principle

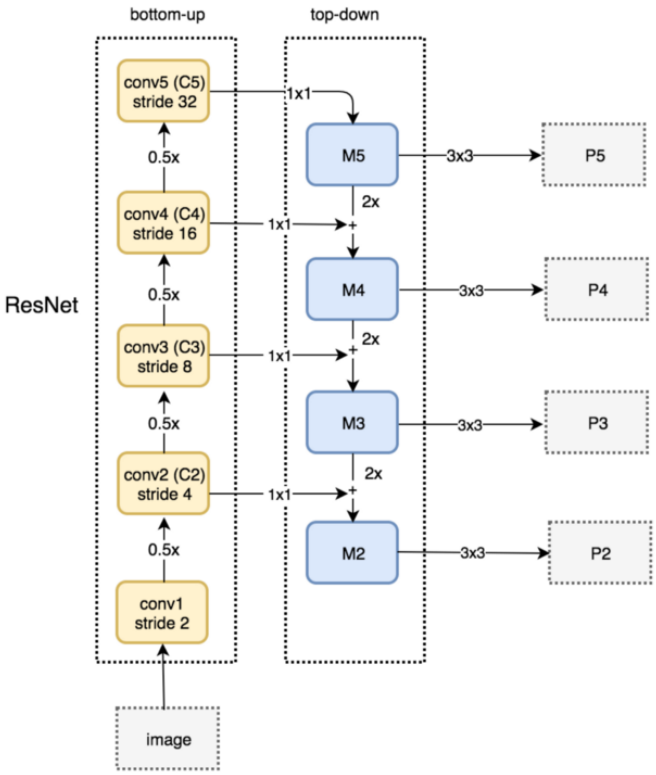



Figure 10: FPN with ResNet [14]

# 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**

- (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.

**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-RCNN 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 = Feature sharing** i.e. use convolutional features from backbone and pool each proposal window to fixed size

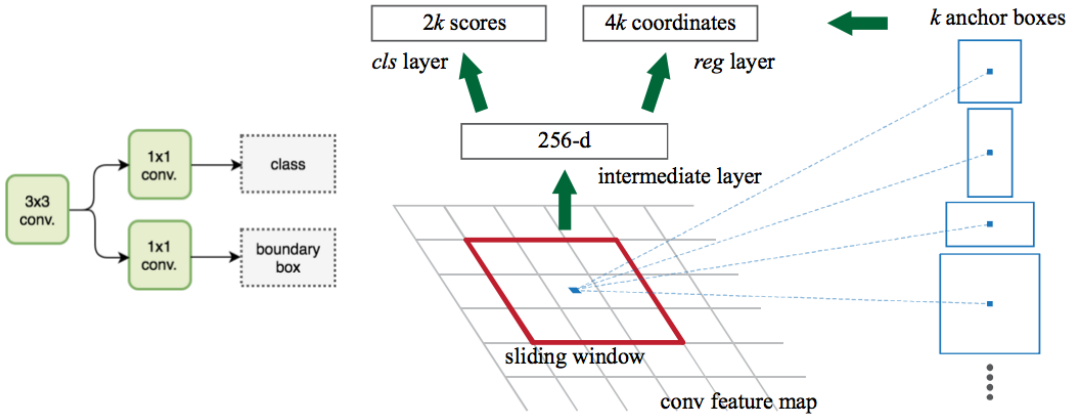



Figure 11: Left: RPN components [14]; Right: Anchor shapes per center point i.e. feature map pixel [11].

### 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)

***cls*, *reg*** Per window and anchor shape do in parallel

**objectness score** (*cls*) This is analogous to the window scoring method and provides the rough region proposals. 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).

**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 only the aspect ration has to fit very roughly.

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

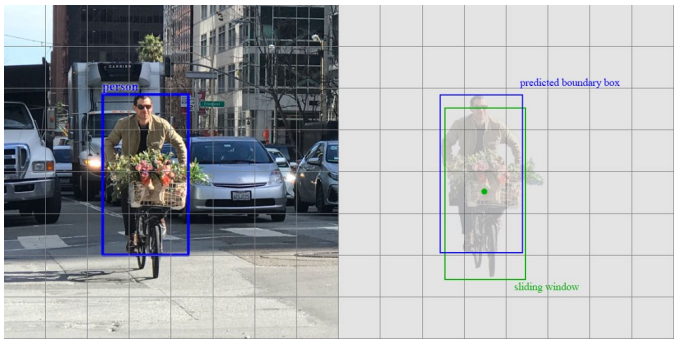



Figure 12: Example of a bounding box correction by the *reg* part of a region propsal network. [14]

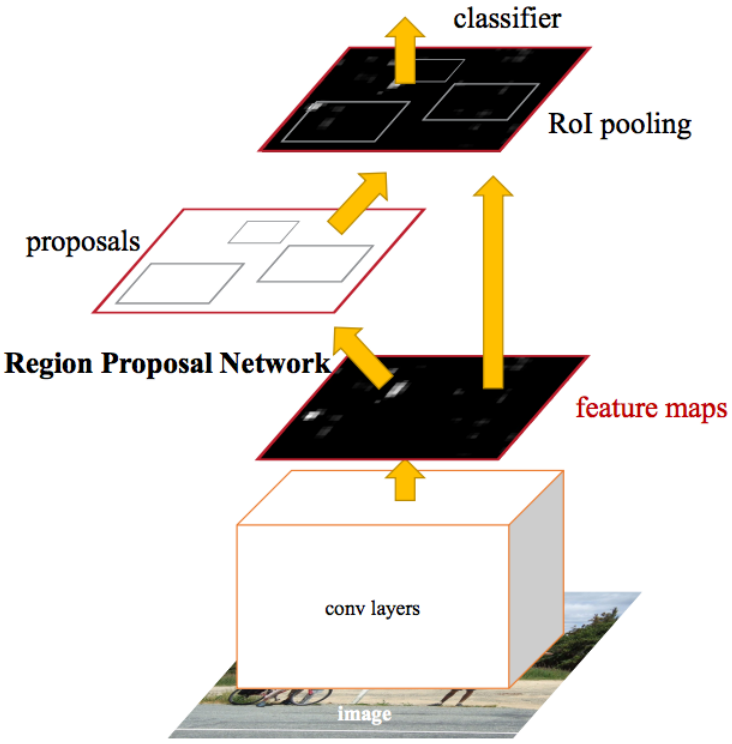



Figure 13: Location of the RPN component within the network

### 3.3.4 Coordinate and RPN *reg* output encoding

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

**Box coordinate** encoding (all normalized):

$(x_1, y_1$ `# upper left corner`

$x_2, y_2)$ `# lower right corner`

**Coordinate correction** (=*reg* output) encoding:

$\Big(dx,$ `#` $\frac{\texttt{centerpoint x-difference}}{\texttt{anchor width}}$

$dy,$ `#` $\frac{\texttt{centerpoint y-difference}}{\texttt{anchor height}}$

$\log(dw),$ `#` $\frac{\texttt{ground-truth box width}}{\texttt{anchor width}}$

$\log(dh)\Big)$ `#` $\frac{\texttt{ground-truth box height}}{\texttt{anchor height}}$

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

$$\textbf{IoU}(A, B) := \frac{\text{Intersection Area}}{\text{Union Area}}$$

### 3.3.5 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):

**object=1** with ground-truth box $b$ if

- it has the

  best **IoU** for $b$, else if
- **IoU** with $b \geq 0.3$

**no object=-1** if not pos. and **IoU** $\leq 0.3$

**neutral=0** else (excluded from training)

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

Balance positive and negative boxes for training! This can e.g. be done by setting most of the actually negative anchors to neutral to be excluded from training.

### 3.3.6 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 *reg*-Training).

**Objectness classification** Conv layer with

- $1 \times 1$-sized kernel
- 2-class softmax activation: (non-object score, object score)

*Loss:* crossentropy for non-neutral anchors

**Coordinate correction** Conv layer with

- $1 \times 1$-sized kernel
- $4 \times$(number of anchors) filters: $(dx, dy, dw, dh)$ coordinate correction for each anchor

*Loss:* smooth $L_1$-loss[1]

for positive anchors that do not cross image bounds

The border-crossing anchors are excluded because:

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

This is simply implemented by a "valid"-padding for the sliding window; or alternatively setting their ground-truth objectness score to neutral (thus they are ignored during training, see *cls*-Training).

**Proposal selection**

1. Trim to $N$ best-object-scored anchors.
2. Apply coordinate correction.

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

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

This is differentiable and the loss is more robust to outliers than $L_2$.

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 resp. its surroundings completely. However, anchors at the
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*

## 3.4 RoI-Pooling

Region of Interest (RoI) pooling is the process of resizing the output of a proposed window to the frontends 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 RoI-Align

RoI-align suggests to avoid any cropping/rounding when disecting the windows output into rectangles for pooling. This can be done by calculating the rectangle values via bilinear interpolation instead of cropping to full pixels followed by e.g. average pooling. Using RoI-Align, the feature maps from the bounding boxes evaluated by the frontend are more accurately aligned with the original image and masks get better.

## 3.5 Frontend

### 3.5.1 Main Ideas

Decouple

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

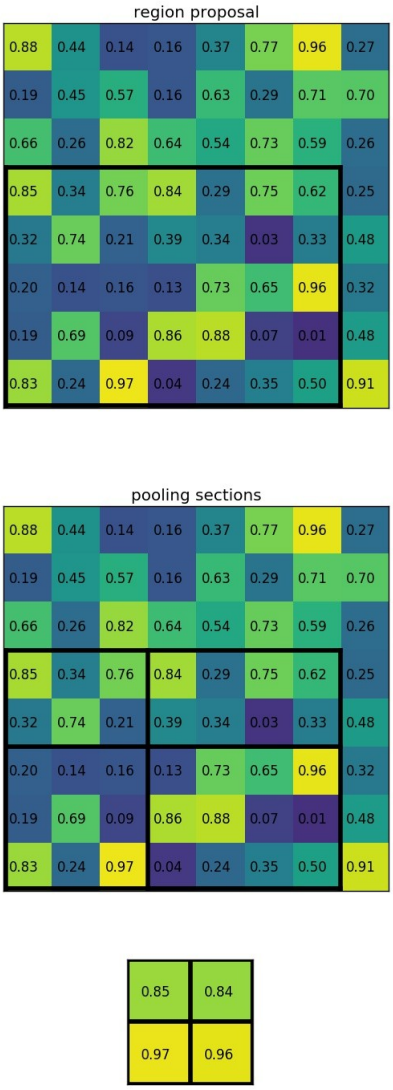



Figure 14: Visualization of Region of Interest Pooling. From top to bottom: Selection of a window; splitting of the window into roughly equal rectangular sections (round half width/height to full pixels); Max-Pooling output of the segmentation

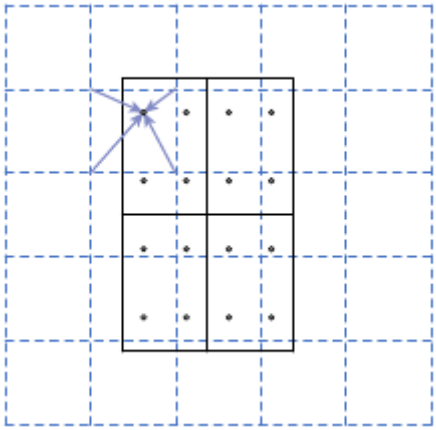


Figure 15: Visualization of RoI-Align: A pixel on the output feature map is bilinearly interpolated from nearby pixels in the input feature map.

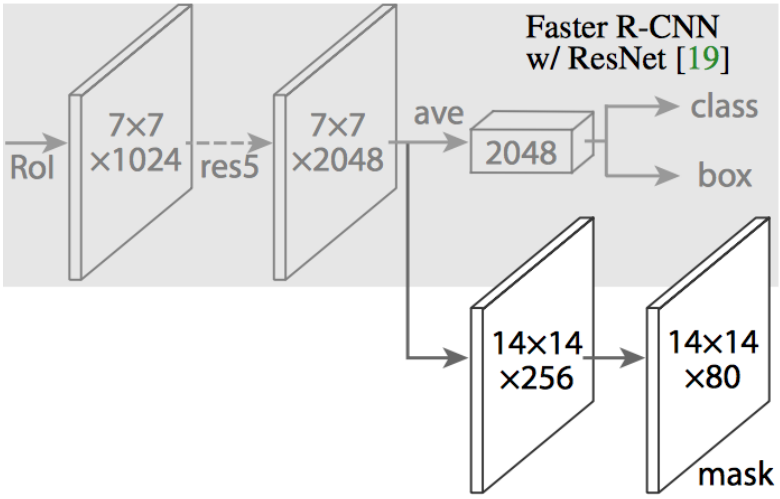



Figure 16: Frontend architecture overview; in this case a ResNet-FPN backend is assumed. Note the three mostly independent prediction branches.

### 3.5.2 Architecture

**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**

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 in several steps:

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

## 4.2 Backbone Pretraining

[t] 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 Here the "No object" class should already be respected and trained.

**Training Input**

**inputs** one-object images (ca. 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 region proposal network and the frontend are trained alternately. The proposed order is:

(after RPN is trained)

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)

## Implementation Review

### 4.4 Source Code

- Keras implementation by matterport: [7]
- Easy example for handwritten number detection using autogenerated data based on MNIST: github

### 4.4.1 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 (can use `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.

## 4.5 Lessions learned

- Always double-check and note down tensor/array dimensions; mind the padding for convolutions.
- Always double-check and test your algorithms.
- 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 contain bugs, too.
- Directly document and update all input and output formats of functions. Resp. in general: Keep your code VERY clean and understandable.
- Convolution is very geometric: Depict your sliding windows and downscaling factor(s) to check whether they make sense with your data/object sizes.
- Mind your RAM when optimizing data generation/tagging
- Check your loss and validation values; Does the model actually get better?
- The deeper, the much more time;
- Make data as easy (small) as possible; if you can still classify, the network can do, too (e.g. grayscale instead of several color channels).

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

[14] *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/.