Master Thesis

Segmenting Aerial Images into Polygonal Shapes with Deep Learning

Author(s):
Rüegg, Nadine

Publication Date:
2016

Permanent Link:
https://doi.org/10.3929/ethz-a-010870851

Rights / License:
In Copyright - Non-Commercial Use Permitted
Segmenting Aerial Images into Polygonal Shapes with Deep Learning

Master’s Thesis
Photogrammetry and Remote Sensing \ Data Analytics Lab
2016

Nadine Rüegg

Advisors:  Dr. Jan Dirk Wegner, Dr. Aurélien Lucchi, Prof. Martin Jaggi
Supervisors: Prof. Thomas Hofmann, Prof. Konrad Schindler
Abstract

We present two deep learning approaches that pursue the goal of polygon extraction from aerial images. Within this work we focus on buildings. Existing approaches address the problem of online map generation as a pixel-wise segmentation task. We instead frame this problem as direct construction of polygons representing objects.

Our first approach is based on Faster R-CNN, a successful object detection algorithm. The main advantage of this approach is that it allows to efficiently and directly exploit object geometry. Our second approach relies on a combination of a Convolutional Neural Network to detect object corners and a Siamese Network to find matching corners that belong to the same object. Even though this approach consists of several nested steps it shows higher flexibility. The polygonal representation produced by our approaches opens the door to the use of object-level constraints. In the past this has proven to be extremely difficult to model with conventional pixel-based approaches [1, 2, 3] or even combinations of FCN’s with conditional random fields [4] or bilateral filtering [5].

We make use of publicly available very-high resolution (VHR) aerial images and object labels provided by Google Maps and OpenStreetMap (OSM) respectively.
Acknowledgments

First, I wish to thank Dr. Aurélien Lucchi, Dr. Jan Dirk Wegner and Dr. Martin Jaggi for supervising this work, supporting me and contributing with many ideas. I could strongly benefit from their experience and expertise in the broad field of Machine Learning. Moreover a big thank you to Prof. Dr. Thomas Hofmann and Prof. Dr. Konrad Schindler for offering me the opportunity of working on this great project, using the infrastructure of their labs and collaborating with their teams.
# Contents

Abstract iii

Acknowledgments v

Contents vii

1 Introduction 1
   1.1 Related Work ........................................... 2
      1.1.1 Non-Neural Network Methods ......................... 2
      1.1.2 Neural Network Methods .............................. 2
      1.1.3 Neural Network Based Work on Road Images .......... 3

2 Data 7
   2.1 Data Generation ........................................ 7
   2.2 Challenges ............................................ 7
   2.3 Ground Truth Correction ............................... 9

3 Neural Networks 11
   3.1 Neurons ................................................. 11
   3.2 Neural Networks ....................................... 12
      3.2.1 Training ........................................... 12
   3.3 Convolutional Neural Networks .......................... 13
      3.3.1 Pooling Layer ..................................... 14
      3.3.2 Convolutional Layer ................................. 15
      3.3.3 Deconvolutional Layer ............................... 16
      3.3.4 Fully-Connected Layer .............................. 16
      3.3.5 Dropout Layer ..................................... 16
      3.3.6 Loss Layer ......................................... 17
   3.4 Typical Network Structures ............................ 17

4 Faster R-CNN 19
   4.1 Part 1: CNN ............................................. 20
   4.2 Part 2: RPN ............................................. 21
   4.3 Part 3: R-CNN .......................................... 23
   4.4 Network Training ...................................... 23

5 Approach 1: Based on Faster R-CNN 25
   5.1 Implementation ......................................... 26
   5.2 Loss Calculation ....................................... 27
Contents

6 Approach 2: Graph Based  
6.1 Keypoint Detection  
6.1.1 Other Methods Tested  
6.2 Edge Weight Calculation  
6.2.1 Thoughts on Efficiency Enhancement  
6.3 Graph Construction  
6.4 Graph Segmentation  

7 Results  
7.1 Evaluation Metrics  
7.1.1 Recall, Precision and F1-Score  
7.1.2 Receiver Operating Characteristic  
7.2 Approach 1  
7.2.1 Pixel Based Evaluation  
7.2.2 Corner Based Evaluation  
7.2.3 Visualization of the Results  
7.3 Approach 2  
7.3.1 Keypoint Detection  
7.3.2 Edge Weight Calculation  
7.3.3 Graph Segmentation  
7.3.4 Evaluation of the Whole Pipeline and Suggestions of Improvement  
7.4 Comparison of Approach 1 and Approach 2  

8 Conclusion and Future Work  

Appendix  
1 Layers for Approach 1  
1.1 Anchor Target Layer  
1.2 Proposal Layer  
1.3 Proposal Target Layer  
1.4 RoI-Pooling Layer  
2 Triamese Network for Edge Weight Calculation
1 Introduction

The purpose of this work is to extract building polygons from aerial images. We believe that objects such as buildings can be more naturally modeled when we depart from the standard paradigm that labels pixels and aim to directly exploit and learn the geometry of buildings. A polygonal representation comprises much less redundancy than pixel-wise labels and is thus a more compact and useful representation of the objects of interest. Finding methods to leave the grid structure that comes with (aerial) images is a first step in the research direction of work with neural networks on non-grid structured graphs.

The problem of image understanding and segmentation was widely discussed and studied in the past. Great progress was brought about by the success of deep neural networks [6, 7, 8]. In contrast to common computer vision approaches, deep learning methods do not rely on hand-crafted image features but are able to learn them during the training process. On one hand networks for image classification were developed [7, 8], on the other hand networks for semantic segmentation such as Fully Convolutional Networks (FCNs) which can be trained end-to-end, pixels-to-pixels [3, 9, 2] were introduced. Convolutional Neural Networks have already been applied to aerial images [10, 11, 12, 13, 14], but all of the works mentioned aim to perform a semantic segmentation of aerial or satellite images. We however, want to depart from per-pixel labeling and directly learn buildings in their vector-representation that is polygons.

This thesis is organized as follows. First, we review related work (see Section 1.1) and look at non-neural network approaches as well as neural network based methods for aerial image segmentation. In Chapter 3, fundamentals of Convolutional Neural Networks are revisited. The successful object detection algorithm Faster R-CNN is explained in Chapter 4 as it serves as foundation for our first building segmentation approach which we describe in Chapter 5. Our second method for building segmentation is based on building corner extraction that is followed by a siamese network to detect matching corners that belong to the same building. A description of this method can be found in Chapter 6. We evaluate the two approaches in Chapter 7. Conclusions are drawn and future work discussed in the last chapter.
1.1 Related Work

Aerial image segmentation is not a new topic, several methods have been developed during the past decades. There are two main directions of research to distinguish. On one side approaches that mainly rely on hand-crafted features and on the other side methods related to Neural Networks. Both strategies are quickly reviewed within the following sections. In the last section of this Chapter (Section 1.1.3) we present a few approaches for road detection which we have tried but not further examined within the scope of this thesis.

1.1.1 Non-Neural Network Methods

Semantic segmentation is the pixel-wise classification of image content. For accurate classification, knowledge about a pixel’s neighborhood needs to be incorporated. [15] include raw pixel intensity values observed within a local neighborhood. Texture can be incorporated by the use of filters (e.g approximations of Gaussians or Gabor) which are convolved with image channels [16]. The shadow-side of filter banks is that they are often tuned for a specific application and do thus not automatically adapt to various kinds of texture. [17] combine texture filters with several additional features such as Randomized Quasi-Exhaustive (RQE) features calculated via integral images [18] and let a boosting classifier assemble the most discriminant textures. An other way to jointly optimize classification and feature selection for hyperspectral images is shown in [19].

A different strategy to model texture is to work with prior distributions of graphical models as applied by [20] in the context of aerial images. Schindler et al. state that global Random Field models outperform local filtering methods and impose smoothness constraints [21]. A CRF formulation for road labeling where the prior is represented by higher-order cliques connecting superpixels along straight line segments is introduced by [22]. They propose a data-driven sampling strategy in order to find relevant cliques and make inference tractable. 2 years later, in 2015, the assumption of straight line segments is abandoned and [23] extend their model to allow for arbitrary paths. Road extraction is formulated as a minimum cost path problem.

1.1.2 Neural Network Methods

Neural Networks have recently gained a lot of attention and proven to be valuable for many machine learning tasks ranging from medical diagnosis, character recognition, financial applications such as stock market prediction or loan granting to image compression. The main difference between Deep Learning and conventional machine learning techniques it that feature extraction and the actual learning task are fully coupled. This allows a learning algorithm to automatically learn on both in parallel. There is no need for hand-crafted feature generation.

(Convolutional) Neural Networks are particularly successful in the field of image analysis. Image recognition [6, 24, 25] as well as semantic segmentation [3, 9, 2] are tasks where Neural Networks outperform all other state-of-the-art methods. [11, 12, 13] and [26] examine their performance on aerial images and detect buildings and roads with high confidence. Recently, Fully Convolutional Networks (FCNs) have been applied to aerial images as well and proven
1.1 Related Work

to show good performance [26] even in the case of vague ground truth data [14].

FCNs can not predict structured output and impose topological constraints on the output. CNNs have been combined with Conditional Random Fields (CRFs) [27, 28] or Bilateral Filtering [5] in order to overcome that lack. Even though promising, these methods do not allow for long-range constraints. We believe that such constraints could be incorporated best, if the image grid was left. [29, 30] show how to define convolution and pooling operations on arbitrary graphs, but did not yet apply this knowledge to an object detection task.

1.1.3 Neural Network Based Work on Road Images

We quickly examine a few approaches for road segmentation. They are all based on a Neural Network for pixel-wise classification. This network is very shallow and thought as tool to achieve first results on a Laptop without GPU in a short time.

Max-Flow / Min-Cut Algorithm

We examine graph cut as a post-processing step. The energy function to minimize is

$$E(x) = \sum_i D_i(x_i) + \sum_{i,j} K|x_i - x_j|$$  \hspace{1cm} (1.1)

where $K$ denotes the regularization strength. The higher its value, the smoother the output. $x \in \{0, 1\}$ represent values of the smoothed output. The unary term $D_i(0)$ is the penalty for assigning value 0 to the $i$-th output of the neural network $p_i$, $D_i(1)$ the penalty for assigning value 1 to it [31, 32]:

$$D_i = \begin{cases} 
  p_i & \text{for } x_i = 0 \\
  1 - p_i & \text{for } x_i = 1 
\end{cases}$$ \hspace{1cm} (1.2)

Figure 1.1 illustrates in (a) a graph with source, sink and gray nodes representing the
described function. In (b) an output of the neural network and in (c) the corresponding solution after Min-Cut.

**Figure 1.1:** In (a) a graph with source, sink and nodes representing predictions for pixels. In (b) and (c) output of a very shallow classification network and corresponding solution after Min-Cut. Predicted streets are depicted in white.
the energy function introduced in 1.1 is minimized. Subfigures (b) and (c) show for an example image the classification obtained by our neural network and the resulting labeling after applying the Min-Cut algorithm respectively.

**Network Stack**

The performance of our shallow Convolutional Neural Network is not overwhelming but we find that adding a second network on top of it leads to an improvement. We train a Neural Network on the output of the first network. By doing so we increase the overall receptive field and enhance the consistency between labels of pixels in a neighborhood. Figure 1.2 demonstrates the effect of the idea. This method was first applied to road images by [33] and developed by [34] who state that this kind of Neural Networks is mathematically related to Markov Random Field (MRF) approaches and that special cases of Convolutional Networks correspond to mean field inference for a MRF.

**Figure 1.2:** Input image followed by the output of the first and second neural network respectively

**Patch-Wise Classification**

A different method investigated is based on image patches. Similarly as before we start with a Neural Network to perform semantic segmentation. In a second step, the output is patch-wise replaced by items (pixel-wise labeled patches) out of a library. In order to construct a patch library we extract ground truth samples and cluster them either by mean shift or k-means clustering. Each cluster is represented by one descriptive patch. We perform experiments with different classifiers such as SVM, Random Forest or Logistic Regression to find for each patch within the neural network output the corresponding cluster. The results obtained are not satisfying and one of the main issues is that neighboring patches do often not fit well. This problem is tackled by combining the patch probability prediction from classification with a score describing the probability that two patterns are found next to each other. This score is calculated based on the matching quality of the pixels on the two aligned patch fronts. A probabilistic model is built and solved by use of the Max-Product algorithm. Unfortunately, we do not yet achieve convincing results for this approach.
1.1 Related Work

Hough Transform

An other method of post processing is to perform a Hough transform \[35, 36\] on the neural network’s output. Hough transform is a method for parameter estimation. We let each Neural Network output pixel vote for lines (streets) that it may originate from. Votes from points with higher score get more weight. A line in 2D can be parametrized by an angle $\alpha$ and a distance $d$ to the origin of the coordinate system as follows:

$$x \cos \alpha + y \sin \alpha = d \quad (1.3)$$

Each pixel could theoretically belong to many lines and thus it is mapped to a sinusoid in parameter space. For an example image we illustrate the corresponding parameter space in Figure 1.3(a). By looking at the input to the Hough transform (subfigure (b)), we recognize four streets. The parameter space shows 5 peaks, but by having a closer look it becomes clear, that the bright point on top ($\alpha = \pi, d = d_{\text{top}}$) and the bottom one ($\alpha = -\pi, d = -d_{\text{top}}$) parametrize the same line. Applying non-maximum suppression (as well as some line refinement) leads to the result depicted in Figure 1.3(d).

Hough transform is not directly feasible for more complex pictures where streets are either not straight or do not cross the whole image. In order to adapt this idea to more interesting situations we could divide images into sub patches and work not only with parameterizations for straight lines but curves as well. \[36\] show how the method can be adapted to more general curve fitting.

Figure 1.3: In (a) parameter space with distance $d$ on x-axis and angle $\alpha$ on the y-axis. In (b-d) original image, output of the neural network and output of the hough transform respectively.
2 Data

Within this chapter we first quickly present our method for ground truth generation. In a second step we discuss its quality and drawbacks. Last, we briefly present a method to refine data. Even though this method is not used further on, we believe it could serve as a starting point for further ground truth quality enhancement.

2.1 Data Generation

All data used within this thesis is online available. We follow a semi-automatic approach to generate a new ground truth dataset of very-high resolution (VHR) aerial images with corresponding object labels that was presented in [14]. Aerial images in RGB format are downloaded from Google Maps via the Google Static Maps API\(^1\), geographic coordinates of building corners are retrieved via the OpenStreetMap API\(^2\).

Within the scope of this thesis we consider data from Chicago only. Each of our images (∼ 3000 × 2500 pixels) covers an area of approximately 0.08 km\(^2\).

2.2 Challenges

[14] report that data sets generated the way described above suffer from the angle issue. The angle issue describes the effect that within aerial images not all parts show the captured area from a truly orthogonal perspective, as the example in Figure 2.1 illustrates. This is due to the fact that airplanes fly with a limited distance to earth surface. This issue complicates...

---

\(^1\)https://developers.google.com/maps/documentation/static-maps/
\(^2\)http://wiki.openstreetmap.org/wiki/API_v0.6
(a) Inaccurate corner positions, unlisted buildings
(b) Near neighboring buildings
(c) Buildings with many small corners
(d) Difficult perspective, occlusions
(e) High variance in building size and shapes
(f) Shadows

Figure 2.2: Challenges arising with our data, red circles are centered on ground truth corner positions
not only the interpretation of the images, but makes it also difficult to define how buildings should be labeled. Buildings need to be defined by their floor plane in order to construct meaningful maps, on the other hand, most of the time complete roofs are visible whereas the floor plan is not.

But the angle issue is not the only challenge that arises with data acquired from Google Maps and OSM. Several cases of difficult data and inaccurate labels are portrayed in Figure 2.2. Ground truth building corner positions are in all sub figures tagged by red circles. Images (a)-(d) illustrate cases where labels are not accurate, house tags are missed or illusory added, buildings are occluded or corners lay very near together such that it’s hard to distinguish between individual units. Images (e) and (f) show problems that arise independently of the ground truth generation method. Intra class variation can be large and images captured do always a bit differ depending on the immediate sunlight.

The big advantage of our data generation method is that a nearly infinite amount of training data from various cities and villages can be acquired. But unfortunately we find that in some cases the bad data accuracy does harm the performance of city structure learning algorithms.

### 2.3 Ground Truth Correction

OSM ground truth is often shifted with respect to the aerial images. Not all buildings are shifted in the same direction and most of the building positions are not precisely known. We find that a sobel edge detector is able to detect many building edges and try to correct our ground truth using such an edge detector. For each building the ground truth outline

![Output of the sobel edge detector](image1.png)

![Some shifted building outlines, original edges in blue, corrected ones in red](image2.png)

**Figure 2.3:** Pixel-wise score of the sobel edge detector and corrected outlines for some randomly chosen buildings
(thickness of 3 pixels) is shifted in x and y direction. We consider all displacements up to at most 8 pixels in each direction. For each possible outline displacement all sobel scores in the 3 pixel thick building outline are added up. The outline shift which leads to the best score is chosen to be the corrected building outline. Figure 2.3(b) shows for some buildings the old position in blue and the new position in red. For many buildings which do not suffer much from the angle-issue, it is possible to correct the outline. Nevertheless, the method described here is not always reliable, not elegant and computationally expensive. Due to this reasons we decide to not apply this method to our data.
3 Neural Networks

This chapter gives an overview of Neural Networks and learning methods based on them. We introduce the concept of neurons in Section 3.1 and pass to Neural Networks and the special case of Convolutional Neural Networks in Sections 3.2 and 3.3, respectively. VGG-16 and U-Net are presented as example network structures in the last section.

3.1 Neurons

The basic computational unit of a Neural Network is a neuron. The operating principle of a neuron is illustrated in Figure 3.1 Given inputs $x_i$, the neuron calculates a weighted sum and adds a bias $b$. The weights $w_i$ are learnable and control the strength of influence of corresponding inputs $x_i$, $b$ can be adjusted as well. A neuron does furthermore apply a non-linearity, called activation function $f$, to its output.

![Figure 3.1: Schematic representation of a neuron](image1)

When constructing a Neural Network, the type of activation function is chosen in advance. A historically often used example is the sigmoid activation function

$$f(x) = \frac{1}{1 + e^{-x}}.$$  \hspace{1cm} (3.1)

It can be interpreted as a squeezing of all real numbers into the range between 0 and 1. This activation function has the undesirable property, that when the neuron’s activation saturates at either tail, the gradient is almost zero what makes weight optimization nearly impossible. An activation function that has become very popular is the ReLU activation [37]

$$f(x) = \max(0, x) \quad ,$$  \hspace{1cm} (3.2)

which can be interpreted as a simple output thresholding at zero. ReLU greatly accelerates the convergence of gradient descent and does not involve expensive computations [38]. Both activation functions presented are illustrated by Figure 3.2.

![Figure 3.2: Activation functions](image2)
3.2 Neural Networks

A Neural Network is modeled as a collection of neurons that are connected in an acyclic graph. They are often organized into distinct layers. The outputs of some neurons become inputs to other neurons. A common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected. Usually, there are no connections between neurons within a layer. Figure 3.3 shows a network with two fully connected layers that contain a total of 6 neurons. This network has 36 learnable parameters ($3 \times 4 + 4 \times 2 = 20$ weights, $4 + 2 = 6$ biases). Unlike all other layers in a Neural Network, the output layer neurons most commonly have the identity function as activation function.

Neural Networks with fully connected layers define a family of functions parameterized by their weights and biases. [39] show that given a continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x)$ with one hidden layer that can approximate this function $f(x)$ such that $\forall x, |f(x) - g(x)| < \epsilon$. But even though in theory a network with only one hidden layer is able to approximate any continuous function, in practice we find that such shallow networks are harder to train with local methods such as Gradient Descent. Even though deep networks have more local loss minima than shallow networks, these minima turn out to be much better in terms of their actual loss [38].

3.2.1 Training

We consider a Neural network that accepts an input $x$ and produces an output $\hat{y}$, where $x$ and $\hat{y}$ do not necessarily need to be scalars. During the forward pass, input $x$ provides the initial information that then propagates up to all hidden neurons until it produces $y$. During training time, the quality of the network is measured by a loss $L(y, \hat{y})$ which compares the produced output $\hat{y}$ to some desired output $y$. One way to minimize this loss is to update all learnable parameters (such as weights $w_i$ and biases $b_i$) according to the local gradient of the loss function. During a so-called backward pass, gradients get back-propagated through the Neural Network. The backpropagation algorithm numerically calculates local gradients by applying the chain rule multiple times.

Suppose that $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$ and $f$ from $\mathbb{R}^n$ to $\mathbb{R}$. If $z = f(y)$ and
3.3 Convolutional Neural Networks

\[ y = g(x), \text{ the chain rule states [40]}: \]

\[ \frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \] (3.3)

**Back-Propagation Example**

Assume we have a function \( L(x_1, x_2, x_3) = (2x_1 - x_2) \cdot x_3 \) and would like to calculate the local gradients at position \( x_1 = 1, x_2 = 5 \) and \( x_3 = -4 \). The loss function can be broken down into the three parts \( g = 2x_1, h = g - x_2 \) and \( L = h \cdot x_3 \).

**Forward pass:**

\[
\begin{align*}
g & = 2x_1 = 2 \cdot 1 = 2 \\
h & = g - x_2 = 2 - 5 = -3 \\
L & = h \cdot x_3 = -3 \cdot -4 = 12
\end{align*}
\]

**Backward pass:**

\[
\begin{align*}
\frac{\partial L}{\partial g} & = x_3 = -4 \\
\frac{\partial L}{\partial h} & = h = -3 \\
\frac{\partial L}{\partial x_2} & = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial x_2} = -3 \cdot -1 = 3 \\
\frac{\partial L}{\partial x_1} & = \frac{\partial L}{\partial g} \cdot \frac{\partial g}{\partial x_1} = -4 \cdot 2 = -8
\end{align*}
\]

Figure 3.4: Visual representation of forward and backward pass calculations

Figure 3.4 shows the visual representation of the above stated example problem. It is illustrated in green, how the forward pass computes all values starting with the inputs \( x_i \). Back-propagation then starts at the end and the chain rule is recursively applied in order to calculate all gradients [38].

**Updating the Network**

By simply calculating the gradients with respect to all learnable weights, the network is not yet trained. The learnable parameters need to be updated. The simplest and most intuitive update rule for a trainable parameter \( x \) is

\[ x \leftarrow x - \lambda \cdot dx \] (3.4)

where \( dx \) denotes the gradient of the loss function with respect to parameter \( x \) and \( \lambda \) a hyperparameter called learning rate. This update rule can be seen as taking a small step of size \( \lambda \) along the direction of the strongest loss decrease.

There exist several more advanced methods such as for example Momentum update or ADAM optimizer [41].

### 3.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN, ConvNet) makes the explicit assumption that its input is grid structured (most of the time an image). Unlike a regular Neural Network, a ConvNet layer has its neurons arranged as three-dimensional arrays of size \( W \times H \times D \), where \( W \) and \( H \) are spatial dimensions and \( D \) represents the depth (number of feature channels).
Figure 3.5 illustrates this typical neuron configuration. The first layer of a ConvNet is the input image itself and D equal to the number of color channels.

![Convolutional Neural Network, [38]](image)

A ConvNet is a sequence of layers, where every layer transforms one volume of activations to another through a differentiable function. For example in a visual recognition tasks, the network transforms the input image layer by layer from pixel values to class scores.

Fully-connected layers as explained in the context of general Neural Networks can be found in CNNs, too. But in addition convolutional layers and pooling layers are introduced. For so called Fully Convolutional Neural Networks (FCNs, see section 3.4 for an example), deconvolutional layers are a further important building block. A brief description of different layer types follows in the next sections. For reference see [38].

### 3.3.1 Pooling Layer

Pooling operations perform non-linear down sampling. This kind of layer is used to reduce the spatial resolution $W \times H$ of the data, decrease the amount of computation within a network and help to prevent over-fitting.

- **Working Principle:** A rectangular pooling filter of size $f_w \times f_h$ is slid over the input and outputs the maximum value for each spatial region. Channels are treated separately. A pooling filter could also perform other operations such as average or L2, but max is the most commonly used. The interval at which the pooling filter is applied to the input is called stride $s$.

- **Learnable Parameters:** The hyperparameters $s$, $f_h$ and $f_w$ need to be fixed in advance, there are no learnable parameters.

- **Size of the Output Volume:**
  
  \[
  D_2 = D_1 \\
  W_2 = (W_1 - f_w)/(s + 1) \\
  H_2 = (H_1 - f_h)/(s + 1)
  \]

  Indices 1 and 2 tag input and output quantities respectively. As pooling is applied to each channel separately, the volume depth does not change.
3.3 Convolutional Neural Networks

3.3.2 Convolutional Layer

A convolutional layer consists of a set of learnable filters. When a filter is slid over the input data, it produces a 2-dimensional activation map that records the filter response at every spatial position. The network learns filters that activate when their input shows some kind of visual feature. Filters in the beginning of the network might learn small simple local features such as corners or edges of certain orientation, while filters in the end (that get all preceding filter’s output as input) learn bigger structures as for example a wheel.

Figure 3.6: Convolution with filter size $f = 3$, stride $s = 1$ and padding $p = 0$, [42]

Figure 3.7: Convolution with filter size $f = 3$, stride $s = 2$ and padding $p = 1$, [42]

• Working Principle: All $k$ filters of a convolutional layer have similar shape $f_w \times f_h \times D_1$ and are slid (convolved) across the whole height $H_1$ and width $W_1$ of the convolutional layer’s input. By convolution we describe the dot product between the elements of the filter and the respective region in the input data. In order to obtain a desired output size, one sometimes needs to add zeros to the border of the input. The amount of such ‘pixels’ is specified by padding parameters $p_w$ and $p_h$. Stride parameter $s$ defines the interval at which filters are applied.

A different way to see a convolution is as a volume of neurons. Every neuron is only connected to a local region of the input volume. The extent of the connectivity along the depth axis is equal to the depth of the input volume, the spatial extent equal to the filter size $f_w \times f_h$ and called receptive field. See Figure 3.1 for a schematic representation of a neuron. Each neuron multiplies its inputs with corresponding weights $w_i$, adds a bias $b_i$ and feeds the result to an activation function.

• Learnable Parameters: Stride $s$, filter size $f_w \times f_h$ and pooling parameters $p_w, p_h$ are hyperparameters to be fixed in advance. The filter weights $w_i$ and biases $b_i$ build the set of learnable parameters. In total, $k$ biases and $k f_w f_h$ weights need to be learned.

• Size of the Output Volume:

\[
D_2 = k \\
W_2 = \lfloor \frac{W_1 - f_w + 2p_w}{s} \rfloor + 1 \\
H_2 = \lfloor \frac{H_1 - f_h + 2p_h}{s} \rfloor + 1
\]

$\lfloor \cdot \rfloor$ denotes the floor function. Each filter is responsible for one 2-dimensional output filter map. These maps are stuck along the depth dimension and thus the output volume is equal to the amount of filters $k$.
3.3.3 Deconvolutional Layer

By the use of deconvolutional layers, it is possible to upsample data back to a higher resolution. A deconvolutional layer multiplies its inputs by a filter element-wise and sums over the resulting output windows to produce an upsampled output. Deconvolution is sometimes also called backward convolution, fractionally strided convolution or transposed convolution.

![Deconvolution with filter size \( f = 3 \), unit stride and no zero padding, no up-sampling effect, [42]](image1)

![Deconvolution that can be used to up-sample data, white pixels indicate zeros that are inserted between the input, [42]](image2)

- **Working Principle:** The deconvolution operation just reverses the forward and backward passes of a convolutional layer. Because of this deconvolution can be described by a set of parameters similar to the ones used to describe a convolution. These are the filter size \( f_w \times f_h \) as well as zero padding parameters \( p_w \times p_h \) and a stride parameter \( s \).

- **Learnable Parameters:** All filter weights \( w_i \) as well as biases \( b \) need to be trained.

3.3.4 Fully-Connected Layer

Neurons of a fully connected layer are connected to all neurons of the previous layer. All spatial context information gets lost. After inserting a fully connected layer it is not possible to go back to (de)convolutional or pooling layers. Such layers are often used in the end of the CNN to combine information from different filters and various locations. They are responsible for the high-level reasoning. Neurons of a fully-connected layer do not share any weights. The output can be seen as a volume of shape \( 1 \times 1 \times n \), where \( n \) denotes the freely selectable number of neurons.

3.3.5 Dropout Layer

Especially fully connected layers contain many parameters that need to be trained and are thus prone to over-fitting. Dropout layers randomly omit part of the feature detectors on each training case. During training time, input neurons of a dropout layer are dropped out with probability \( p \). For neurons that are dropped, all incoming and outgoing edges are removed as well. Before the next training step, everything is inserted again and a new set of neurons dropped. At testing time, there are no neurons removed, but instead all outputs get weighted by a factor \( p \), what makes the expected output of a neuron at testing time being the same as at training time. Dropout layers prevent over-fitting and lead to an improvement of training speed [43].
3.3.6 Loss Layer

The loss layer is usually the last layer of a CNN. Within this layer, the loss (a scalar) is calculated. There exist many different loss functions and it’s important to choose an appropriate one. The loss functions used within this work are discussed at the time they are applied.

3.4 Typical Network Structures

Within this section we present two exemplary networks. First a CNN for object classification called VGG-16 and secondly a U-Net structured FCN, that performs well on semantic segmentation tasks.

VGG-16

VGG-16 is a successful CNN model for object classification proposed by K. Simonyan and A. Zisserman [25]. The architecture consists of the repeated application of two or three convolutions followed by a max pooling operation for down-sampling. Having multiple successive convolutional layers supports the development of complex features and the overall receptive field is increased. Figure 4.4 shows how an image of size $224 \times 224 \times 3$ is merged spatially to a small size while the depth (number of activation maps describing it) is increased until the output has shape $7 \times 7 \times 512$. Three fully-connected layers follow, the last of them has a depth equal to the number of object classes (1000). The output of that last fully-connected layer is interpreted as a vector containing unnormalized log-probabilities for all object classes. The softmax layer transforms them to probabilities.

$$y_c = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}},$$  \hspace{1cm} (3.5)
where $z_c$ is the log-probability of class $c$. It’s common to use a cross-entropy loss function $L_{ce}$ to train such a network.

$$L_{ce} = - \sum_{i=1}^{N} \sum_{c=1}^{C} t_{ic} \log(y_{ic})$$

$t_{ic}$ is 1 if and only if sample $i$ belongs to class $c$ and $y_{ic}$ denotes the corresponding predicted probability [45, 46].

**U-Net**

U-Net is a network for image segmentation, it calculates pixel-wise class labels. The network structure (see Figure 3.11) was developed by O. Ronneberger et al. [3] and is based on earlier work such as [9]. The architecture consists of a contracting path (left side) and a symmetric expanding path (right side). The contracting path has an architecture similar to typical convolutional networks, it stacks a few (unpadded) convolutional layers, follows them by pooling layers and repeats this pattern. At each down-sampling step the number of feature-channels is doubled. While the contracting path captures context, the expansive path is responsible for precise localization. In the expanding path, the output of a deconvolutional layer (called up-conv within Figure 3.11) is always concatenated with the correspondingly cropped feature map from the contracting path. This concatenation allows to fuse local appearance information from a fine layer (contracting path) with semantic information from a coarse layer (expanding path) to produce accurate and detailed segmentations. Two convolutions are applied before the next up-sampling step follows.

In the end a feature map with $C$ channels corresponding to $C$ classes is produced. This map does not have the exact same size as the input image due to the loss of border pixels in every convolution. The loss function for such a network is calculated by summing up some pixel-wise loss.

---

**Figure 3.11:** U-Net Architecture, [3]
4 Faster R-CNN

S. Ren et al. propose the object detection algorithm *Faster R-CNN* [47] which is described in this chapter and serves as basis for our work. Previous work divides object detection in two parts. On one side, regions, that with high probability contain an object, need to be sampled efficiently. Selective Search [48] is a popular method based on super-pixel merging, EdgeBoxes [49] selects boxes that show a high number of edges existing within the box and a low amount of box boundary overlapping contours. On the other side, the spatial location of such object proposals needs to be refined and the objects classified. Region-based CNNs as originally developed by [50] celebrate considerable success but process each proposal individually and are thus slow. Fast R-CNN [51] on contrary shares computations between regions of the same image. Further, the concept of RoI-pooling is introduced, thanks to which preceding convolutional layers can be updated. Fast R-CNN serves a basis for Faster R-CNN.

![Faster R-CNN network structure](image)

**Figure 4.1:** Faster R-CNN network structure, [52]

The question of how to efficiently combine Fast R-CNN with region proposal generation is not yet answered and the main contribution of [47]. They introduce a Region Proposal Network (RPN) which shares convolutional layers with a network similar to Fast R-CNN and obtain nearly cost-free proposal computations given the detection network’s computations. Figure 4.1 shows the structure of a ‘Faster R-CNN’. It can be divided in three parts:
1. **Convolutional Neural Network (CNN):** The CNN takes an image of arbitrary size as input and outputs a feature map. Even though this map has a lower resolution than the image, it can be seen as a map providing a feature vector for each location on a grid.

2. **Region Proposal Network (RPN):** Input is the feature map created by the CNN. A small window is slid over this map and at each location simultaneously for \( k \) anchor boxes a regression as well as a score calculated. This score encodes the probability, that the current anchor box encloses an object of interest. A set of boxes with high scores (Regions of Interests, RoIs), is output. Figure 4.2 illustrates two anchor boxes at a single position as well as the regressed box for an anchor with high score.

3. **Region based CNN (R-CNN):** The R-CNN extracts for each box the corresponding part of the feature map. In order to get a small square of fixed shape for each RoI, a special kind of max-pooling, called RoI-pooling, is applied. These new evenly shaped feature maps serve as input for a neural network consisting of three fully connected layers. Class scores as well as further bounding box regressors are predicted with respect to each RoI. Figure 4.3 shows RoIs as proposed by the RPN as well as one of the RoIs after bounding box regression.

In the following sections all three parts are described in more detail.

### 4.1 Part 1: CNN

In principle any Convolutional Neural Network could be used as long as it is deep enough and the total stride not too big. [47] investigate the Zeiler and Fergus model [53] (ZF) and the
4.2 Part 2: RPN

Simonyan and Zisserman model [25] (VGG-16). The second one is deeper and thus used in the following. VGG-16 (see Figure 4.4) has 13 convolutional layers (filter size: $3 \times 3$, stride: 0) and four max-pooling layers (filter size: $2 \times 2$, stride: 2) what leads to a total stride of 16. The depth of the last layer is 512. Instead of randomly initializing the VGG-16 weights, it is possible to use pretrained values, for example from training on a PASCAL VOC dataset. For more information regarding this network see Chapter 3.4

![Figure 4.4](image)

**Figure 4.4:** Part of the VGG-16 that is used as ConvNet for the Faster R-CNN, [52]

4.2 Part 2: RPN

To generate region proposals, a small window of size $3 \times 3$ is slid over the feature map created by the CNN. Each sliding window is mapped to a 512-dimensional feature vector and this feature vector fed into two fully-connected layers – a box-regression layer and a box-classification layer. Their output is of shape $4k$ and $2k$ respectively.

![Figure 4.5](image)

**Figure 4.5:** Concept of anchor boxes, [47]

For each of the $k$ anchor boxes width, height and position need to be regressed. The parametrization form [50] is adapted:

$$
t_x = (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a
$$

$$
t_w = \log(w/w_a), \quad t_h = \log(h/h_a)
$$

(4.1)
where $x$, $y$, $w$ and $h$ denote the box’s center coordinates and its width and height. All four parametrized coordinates of the predicted bounding box are represented by a vector $t_i$, the coordinates of the corresponding ground truth box by $t_i^*$. Variable $x$ belongs to the predicted box, $x_a$ to the anchor box and $x^*$ to the ground truth box (likewise for $y$, $w$ and $h$).

The predicted probability of an anchor $i$ being an object is denoted by $p_i$ and the ground truth label by $p_i^*$, where

$$p_i^* = \begin{cases} 
1, & \text{if the anchor has an IoU > 0.7 with any object} \\
0, & \text{if the anchor has an IoU < 0.3 with all objects}
\end{cases} \quad (4.3)$$

The loss function for an image is defined as

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad , \quad (4.4)$$

The two terms are normalized by the minibatch size ($N_{cls}$) and the number of anchor locations ($N_{reg}$) and weighted by a balancing parameter $\lambda$. Observe that boxes with overlap between 0.3 and 0.7 are not considered when calculating the loss and only anchor boxes with $p_i^* = 1$ contribute to the regression loss $L_{reg}$.

For $L_{reg}$ the robust loss function (smooth $L_1$) as defined by [51] is used:

$$L_{reg}(t_i, t_i^*) = \sum_{j \in \{x,y,w,h\}} L_{1\text{smooth}}(t_{i,j} - t_{i,j}^*) \quad (4.5)$$

in which

$$L_{1\text{smooth}}(x) = \begin{cases} 
0.5x^2 & \text{if } \|x\| < 1 \\
\|x\| - 0.5 & \text{otherwise}
\end{cases} \quad (4.6)$$

This loss is less sensitive to outliers than a $L_2$ loss. The classification loss $L_{cls}$ is defined as log loss over two classes (object vs. background).

At every position, the regressors for the $k$ anchor boxes do not share weights! Each regressor is responsible for one scale and aspect ratio – that of the corresponding anchor box. Thanks to this multi-scale design based on anchors, one can simply use the convolutional features computed on a single-scale image.

**RoI Selection**

A set of regression parameters as well as a score is predicted for each anchor. Given an image of size $600 \times 600$, there are roughly $13'000$ ($\approx \frac{600}{16} \times \frac{600}{16} \times 9$) anchors, but many anchors are ignored at training time, as they cross the image boundary. At test time, Non-Maximum Suppression (NMS) is applied and only top proposals selected (for details check Appendix 1.2). Each of them defines a location within the image and the corresponding feature map (see Section 4.2). These regions within the feature map are called Regions of Interest (RoIs). In order to calculate the loss of the RPN we do not use the same set of RoIs as we submit to the next part of the network. For selection of RoIs used for loss calculation see Appendix 1.1. More information about the RoIs used as training samples for the R-CNN part of the network can be found in Appendices 1.2 and 1.3.
4.3 Part 3: R-CNN

This part of Faster R-CNN is adapted from Fast R-CNN [51]. Figure 4.6 illustrates well how an example RoI gets pooled and object class as well as bounding box regressors are predicted.

**Figure 4.6:** Fast R-CNN architecture which is similar to the last part of the Faster R-CNN architecture, [51]

**Figure 4.7:** RoI pooling

**RoI-Pooling**

RoI pooling converts features from any rectangular region of interest (RoI) into a small feature map with fixed spatial extent $H \times W$ [51]. RoI max pooling divides the $h \times w$ RoI window in a grid of sub-windows of approximate size $h/H \times w/W$ and max-pools the values within such a sub-window into the corresponding output grid cell. Figure 4.7 illustrates the RoI max pooling method.

It is not uncommon that $(w \mod W) \neq 0$. In this case the width of sub-windows is slightly increased and thus a feature could be mapped to more than one output cell. The same can be observed if $H$ is no divider of $h$.

**From Pooled RoIs to Predictions**

The by now equally shaped RoIs of size $7 \times 7$ are fed into two fully connected layers. Two sibling output layers follow, one outputting a discrete probability distribution, the second bounding-box regression offsets (per RoI). The loss function is similarly structured as the one for the RPN, details can be found in [51].

4.4 Network Training

The region proposal network (RPN) and the region-based CNN share convolutional layers. Both, if trained independently, modify their convolutional layers in different ways. [47] propose three different techniques that allow for sharing convolutional layers:

- alternating training
- approximate joint training
- non-approximate joint training
We are most interested in approximate joint training and describe this quickly. RPN and R-CNN are merged into one network during training. In each stochastic gradient iteration, region proposals are generated by the forward pass and treated as fixed when training the R-CNN. During the backward pass, signals from RPN and R-CNN loss are combined. This method ignores derivatives with respect to proposal box coordinates, so is approximate.
5 Approach 1: Based on Faster R-CNN

The goal of this thesis is to find a compact representation for buildings. The approach presented in this chapter exploits the geometry of these objects directly. Many buildings have corners with angles of 90 degrees and looking at their outline it is common to see very simple geometric shapes such as rectangles or compositions of rectangles. Due to that reason we adopt an approach similar to Faster R-CNN [47] and design a neural network that looks for objects with rectangular shape.

The successful object detection algorithm Faster R-CNN is described in Chapter 4. It is designed such that objects of various classes are found within an image and simultaneously bounding boxes enclosing these objects can be regressed. Unfortunately, bounding boxes need to be aligned with the x- and y-axis of the image. But buildings might be arbitrarily rotated and it is thus not possible to directly apply Faster R-CNN to our problem. While

![Network structure](image)

**Figure 5.1:** Network structure

Faster R-CNN serves as starting point for our research, we implement an approach capable of predicting rectangles of any desired rotation.

The structure of our network is illustrated in Figure 5.1, changes with respect to Faster R-CNN highlighted in green. We introduce a regression parameter for the orientation of each anchor box and submit rotated boxes to the RoI pooling and R-CNN part of the network. Within RoI-pooling all proposal boxes are reshaped, such that for the last part of the network (R-CNN), all input boxes have similar shape and can be processed by the same network structure. Among other changes we adapt the RoI-pooling forward and backward pass. For
the reshaped boxes (output of RoI pooling step) we do again perform a regression of the box itself (position and side lengths) as well as a regression of the orientation. We predict the rotation $\alpha$ of a box as the angle that this box is rotated with respect to an x-, y- axis aligned box ($-\frac{\pi}{4} \leq \alpha < \frac{\pi}{4}$). We believe that orientation is a feature that can be learned and encoded directly within the feature map. Experiments show that the angle computed in the end is more accurate than the preliminary estimation from the RPN network.

5.1 Implementation

We can fall back on existing code for the Faster R-CNN approach [54] and use this as a basis for our implementation. Figure 5.2 shows the structure of our network training code. Purple

![Figure 5.2: Structure of our code, network for training](image)

denotes network input which consists of ground truth boxes (height, width, x-position and y-position), orientation of these boxes and the image itself. All loss terms which are in the end summed up and jointly minimized are colored orange. Layers in between input and loss are either colored blue or green. Blue indicates, that a layer is not changed with respect to the original version of the code, green highlights changes.

During network training, an image serves as input to a VGG structured CNN. Its output can be seen as a feature map. For nine anchor boxes per location on the feature map, regression parameters and the likelihood that this anchor contains a building are predicted. The RPN loss is calculated using only a subset of all predicted boxes. For more details regarding the sampling process check out the explanation of the anchor target layer in Appendix 1.1. A different subset of boxes is submitted to the second part of the network. For information about the selection of these boxes (called RoIs) see explanations on proposal layer and proposal target layer (Appendices 1.2 and 1.3, respectively). Roi pooling is applied to each of the candidate RoIs and then every RoI separately fed to a stack of fully connected layers that output regression parameters and as well as a score. The loss of the second part of the network (called R-CNN loss) is calculated.

The network that is used for predictions (see Figure 5.3) is smaller than the training network.
5.2 Loss Calculation

Anchor target layer and proposal target layer are removed. The output of the proposal layer is directly fed to the RoI pooling layer.

5.2 Loss Calculation

The loss terms for the boxes’ scores and regression parameters for side lengths as well as position do not change with respect to the original Faster R-CNN network. For their specification see Chapters 4.2 and 4.3. In addition we introduce two loss terms for the orientation. One for the orientation estimated within the RPN part of the network and one for the estimation within the R-CNN part. For both orientation estimates we implement an $L_2$ loss function. 0.2 is found to be a good weight for each of the rotation loss terms.
6 Approach 2: Graph Based

The approach presented within this chapter is designed such that the image grid structure is left very soon. Buildings are seen as polygons defined by building corners and edges. The workflow is depicted in Figure 6.1. In a first step corners are detected, probabilities that different corners belong together calculated after that. Corners and potential edges are then assembled to a graph and in a last step individual objects can be segmented. We aim to train the algorithm end-to-end and construct it in a way that this is possible. Nevertheless this step is left as future work.

6.1 Keypoint Detection

The goal of this step is to find building corners. Several methods are examined (see Section 6.1.1) but building corner extraction using an U-Net structured network works best. In order to train such a network, we do first need to label our data pixel-wise. A small circle is drawn around ground truth corner positions and all pixels within these circles labeled as corner pixels. For general information about U-Nets see Chapter 3.4. Our network is not as deep as the one shown in Figure 3.11. We perform down-sampling and corresponding up-sampling twice instead of 4 times. We do further use 16 channels in the beginning instead of 64. Dropout rate is set to 0.5.

The network is trained with gradient descent and an ADAM optimizer. A good choice of the learning rate (in our case 0.0001) is crucial. Further it is important that the class weight for corner pixels is much higher than for background pixels (we work with 10:1), as the total number of corner pixels is rather small. One thing that makes corner detection difficult is the inaccuracy of our ground truth. Especially for buildings that lay near together it can happen that ground truth corner points do not even lay on the correct building. This vagueness of our data prevents us from choosing very small circles around corner positions and thus it becomes difficult to distinguish between different corners within our result. We apply Non-Maximum Suppression (NMS) to the U-Net’s output and many corners are correctly found but it still happens that near corner points are wrongly merged.
6.1.1 Other Methods Tested

We test other methods for corner detection which we quickly describe in this section, but detection based on U-Net shows the best performance.

**Harris Corners**

The Harris Corner detector is designed such that regions in the image with large variation in intensity in all the directions are detected [55]. Assuming that building corners are such points we try to detect Harris Corners and learn afterwards which of the detected points are building corners. First experiments already reveal, that this method does not lead to the desired success. Figures 6.2 and 6.3 show an example image and Harris Corners as detected for different discrimination thresholds. It can be seen that many wrong points are detected and correct corners missed. This method does further have the disadvantage that there is no chance to learn to detect occluded building corners.

![Figure 6.2: Harris corners, threshold 0.1](image1)
![Figure 6.3: Harris corners, threshold 0.001](image2)

**SIFT Descriptor in Combination with Random Forrest Classifier**

An other idea that we quickly try is to extract a SIFT descriptor [56] for each pixel and learn a Random Forrest Classifier [57] to distinguish between background pixels and pixels that lay near a building corner. Unfortunately we do not achieve good results for this algorithm as well.

6.2 Edge Weight Calculation

After corner detection, the next step is to calculate probabilities that corners belong to the same building. This calculation is done for all potential corner pairs separately by adapting the idea of siamese networks as described in [58, 59] or [60]. We work with patches showing the two corners and a third patch from in between. The third patch is rotated and responsible for information concerning the relative position between the corners. Figure 6.5 illustrates how patches are extracted. Each of the patches is fed to a network that consists of convolutional and pooling layers with weights shared between all three networks. The concatenation of the outputs is followed by a few fully connected layers as well as a softmax layer. We minimize
6.3 Graph Construction

A fully-connected graph $G = (V, E)$ is constructed. All corners detected within an image serve as vertices $V$. The weight $\omega_{ij}$ of an edge from node $i \in V$ to node $j \in V$ is equal to the corresponding score calculated by the triamese network presented in section 6.2 if the spatial distance between the two corners is smaller than a certain threshold. If the distance is bigger, we do not run the triamese network on this corner pair and set the corresponding edge weight to zero.

6.2.1 Thoughts on Efficiency Enhancement

So far each corner pair is considered separately. This leads to many repeated computations as each corner is tested against many other corners. During training it would make sense to calculate the forward pass for all patches of one image (or part of an image) in parallel, then calculate all fully connected layer forward passes. This would mean that the batch size is not constant but it enables calculating the loss for all potential corner pairs at the same time. The network update step is straightforward.

The mid patch might cause memory issues if it needs to be calculated for all corner combinations at the same time and we would propose to forgo it. If the mid patch is skipped, it is even possible to train a network on top of the original image, let this network output a feature map and directly extract patches from this map. By doing so we can build a deep network and avoid separate calculations on distinct patches. Only the fully connected layers’ computations would need to be performed for potential corner pairs individually.

6.3 Graph Construction

The cross-entropy loss. As the network has three inputs we call it a triamese network. For details concerning its structure see Appendix 2.
6.4 Graph Segmentation

The last step is a segmentation of our graph $G = (V, E)$ into individual buildings. We keep in mind that later on our approach should be end-to-end trainable and use normalized cuts [61, 62].

The criterion that is minimized using this approach is called normalized cut criterion and measures both the total dissimilarity between the different groups as well as the total similarity within the groups. Assuming we partition $G$ into two disjoint sets $A, B$, $A \cup B = V$, $A \cap B = \emptyset$, the $Ncut$ criterion is defined as follows:

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)},$$

where $cut(A, B) = \sum_{u \in A, t \in B} \omega_{ut}$ denotes as the total weight of edges that are removed. $assoc(A, V) = \sum_{u \in A, t \in V} \omega_{ut}$ is the total connection from nodes in $A$ to all nodes of the graph, $assoc(B, V)$ calculated analogously.

Assume graph $G$ has $n$ nodes. Let $d(i) = \sum_j \omega_{ij}$ be the total connection from node $i$ to all other nodes. Let further $D$ be an $n \times n$ diagonal matrix with $d$ on its diagonal and $W$ an $n \times n$ symmetrical matrix with $W(i, j) = \omega_{ij}$. An approximate discrete solution for the $Ncut$ minimization can be found by solving the generalized eigenvalue problem

$$(D - W)y = \lambda Dy.$$  

We can use the eigenvector $y$ with the second smallest eigenvalue $\lambda$ to bipartition the graph. For more details on the derivation of Equation 6.2 see [61].

We do not only cut our graph once, but multiple times until all buildings are separated from each other. At the moment we do still assume the amount of buildings is known in advance. Later on (but not part of this thesis) it is planned to cross-validate a threshold for the $Ncut$ criterion and cut the graph as many times as possible without exceeding this threshold.
7 Results

First we introduce evaluation metrics used within the scope of this work (Section 7.1). For results from Approach 1 check out Section 7.2, for results belonging to the second approach Section 7.3. The two methods are compared in Section 7.4.

7.1 Evaluation Metrics

For binary classification, a data point is predicted to be positive (e.g. belongs to a building) or negative (e.g. background). This prediction is either true or false. It follows that four different types of outcomes ($TP, TN, FP, FN$) are possible, as shown in Figure 7.1.

![Coincidence matrix for a two class classifier](image)

Figure 7.1: Coincidence matrix for a two class classifier

7.1.1 Recall, Precision and F1-Score

Recall denotes the amount of correctly classified positives divided by the total positive count, precision as the amount of correctly classified positives divided by the total amount of positive ground truth labels.

\[
Precision = \frac{TP}{TP + FP} \tag{7.1}
\]

\[
Recall = \frac{TP}{TP + FN} \tag{7.2}
\]

A measure that combines recall and precision and is widely used to describe the quality of a classifier is the $F_1$-score [63]. It is calculated as the harmonic mean of the two.

\[
F_1 \text{score} = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{7.3}
\]

For many classification task we have a classification threshold that can be varied. The choice of this discrimination threshold is usually a trade of between precision and recall and could be formulated as a maximization of the $F_1$-score.
7.1.2 Receiver Operating Characteristic

A receiver operating characteristic (ROC) curve illustrates the performance of a binary classifier as its discrimination threshold is varied. The ROC curve is defined by the true positive rate ($TPR$, recall) plotted against the false positive rate ($FPR$), where

$$TPR = \frac{TP}{TP + FN}$$  \hspace{1cm} (7.4)

$$FPR = \frac{FP}{TN + FP}$$  \hspace{1cm} (7.5)

As illustrated by Figure 7.2, a good classifier is characterized by a large area between curve and x-axis.

Figure 7.2: Three ROC curves for classifiers of different quality
7.2 Approach 1

The evaluation of Approach 1 is established in two ways. First, we look at the predicted buildings, project them into the images and evaluate the performance of our method based on pixel-wise classification results. Second, we wish to analyze the accuracy of predicted boxes. This is achieved by an evaluation based on their corner positions relative to the ground truth boxes’ corners.

We are working with 7592 images of size 800 × 900 originating from around 250 big aerial images of size 3000 × 2500 pixels. The small images can show arbitrary rotation and are resized within our algorithm, such that their shorter side has length 600. Images originating from every fifth of the big images serve as test images, all others as training samples. All evaluation metrics presented in the following are calculated with respect to this test set if not stated otherwise.

7.2.1 Pixel Based Evaluation

The pixel-based evaluation of this approach is straight forward. Each predicted box comes with an associated score representing the sureness that a building is enclosed. All pixels that lay within one of the predicted boxes with a score exceeding a discrimination threshold are labeled as building, all remaining pixels as background. Figure 7.3 shows the corresponding precision-recall curve. It is remarkable that a recall level of 100 cannot be achieved. This is due to the fact that a limited amount of proposal boxes is passed through the roi-pooling step and even all together might not always cover the whole image. Figure 7.4 illustrates the F1-Score achieved in function of the discrimination threshold. Many bounding boxes that do not show a building get an extremely low score of almost zero. But from scores of around 0.01 on, boxes that cover buildings start to appear. Unfortunately, it seems to be hard to predict accurate box scores. Some examples are illustrated in Figure 7.7, where

![Figure 7.3: Precision vs. recall for pixel-wise evaluation](image1)

![Figure 7.4: F1-Score in function of the discrimination threshold, for pixel-wise evaluation](image2)
blue boxes highlight predictions and the numbers in white the corresponding scores. F1-score as well as intersection-over-union reach their maximum of 0.7 and 0.54 respectively, for a discrimination threshold of 0.01. But this result has to be treated with caution. When having a look at resulting images it becomes clear that boxes which partially cover multiple buildings often occur with low score. If evaluated pixel-wise, they do not harm the evaluation score badly, even though their prediction is not desired. So in order to get more building map like predictions it can make sense to slightly raise the discrimination threshold.

7.2.2 Corner Based Evaluation

The pixel based evaluation described above does not always punish bad alignment of the predicted boxes with buildings. It can for example happen that one box covering multiple neighboring buildings instead of one produces a high score when evaluated as described above. One way to tackle this problem is to look at the corner positions of the ground truth boxes and compare them to the corner positions of the predicted boxes.

On one hand all four corners of each predicted box are extracted and build the set of positive samples, on the other hand all corners belonging to ground truth box are our target points. Each predicted corner is assigned to the nearest target corner and counts as 'true positive', if such a target exists within a certain range and if no other predicted point is nearer to this target. It is not a priori clear how many 'true negative' points there are, as the output size of the neural network can vary. Recall and precision as well as F1-score are independent of the of the 'true negatives' and thus we are still able to plot a recall-precision curve (see Figure 7.5) as well as the F1-score in function of the discrimination threshold (see Figure 7.6). The F1-score calculated based on a pixel-wise evaluation supports our claim that a threshold which is around 0.03 and thus slightly higher than 0.01 leads to better results.

![Figure 7.5: Precision vs. recall for corner-wise evaluation](image)

![Figure 7.6: F1-Score in function of the discrimination threshold, for corner-wise evaluation](image)
7.2.3 Visualization of the Results

As we work with images within this thesis, a good way to get a feel for the quality of predictions is to visualize results. Figure 7.7 shows a few example images overlayed by predictions in blue. The score (between 0 and 1) of each predicted box is denoted in white. We aim to show a balanced set of examples where weaknesses are not hidden. The third and sixth image show weak prediction quality and missing buildings, but predicted boxes with high score are reliable and almost always accurate.

The quality of predictions does strongly depend on the image. Especially narrow long buildings which lay near together seem to be hard examples. So far all anchor boxes have a side length ratio of 1:1 or 1:2. We believe that the performance of this approach could be improved if anchor boxes with an aspect ratio of 1:3 or 1:4 were introduced. A further challenge of slim boxes is that slightly wrong orientation estimates lead to a bad overlap between the region proposals submitted to the RoI-pooling step and the true image regions covering corresponding buildings. Such inaccurate proposals can harm the algorithm’s performance.
Figure 7.7: Predicted buildings with associated score
7.3 Approach 2

Approach 2 consists of several nested steps. We evaluate the main steps separately in order to highlight weak points and strengths of the method. We start with keypoint detection, step over to the triamese network and finally end up at the normalized cuts step.

The training data for this approach consists of 100 aerial images ($\sim 3000 \times 2500$ pixels) resized by a factor 4. The test set contains around 50 images. All evaluation metrics presented in the following are calculated with respect to this test set if not stated otherwise.

7.3.1 Keypoint Detection

The U-Net for corner detection is trained for 750 epochs. First, we evaluate the performance of the predictions pixel-wise. A receiver operating curve is drawn in Figure 7.9. The area under this curve suggests that performance is ok but not outstanding. Our objects, circles around building corners, are not very well localized. Figure 7.8 shows F1-scores calculated pixel-wise on the U-Net prediction for all test images, while the corner threshold is varied. The best results are obtained by choosing a low threshold (see Figure 7.10(c) for an example).

![Figure 7.8: F1-score of prediction with respect to a corner classification threshold, pixel based evaluation](image1)

![Figure 7.9: Receiver operating characteristic (ROC) curve for pixel-wise predictions](image2)

For the performance of the complete building polygon construction pipeline, not pixel-wise but corner-wise performance is important. Corners are extracted from the U-Net prediction by means of Non-Maximum Suppression. We evaluate the quality of the resulting set of corners as follows: Each corner found is assigned to the nearest ground truth corner if there is one within a certain range (8 pixels) and if this ground truth corner point does not belong to an other predicted point which is nearer. If for a predicted corner no such corresponding point is found, it is counted as ’false positive’ ($FP$), otherwise as ’true positive’ ($TP$). Ground truth corner points that do not get any predicted point assigned count as ’false negative’ ($FN$). For the test set we achieve:
As an illustration we show in Figure 7.10 (a) a test image with corresponding corner ground truth labels in red, (b) raw prediction results, where brighter pixels symbolize higher score, (c) predictions thresholded at a value of 0.1 superimposed onto the input image, (d) resulting corner positions found by applying Non-Maximum Suppression.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.47</td>
</tr>
<tr>
<td>Precision</td>
<td>0.70</td>
</tr>
<tr>
<td>IoU</td>
<td>0.39</td>
</tr>
<tr>
<td>F1-Score</td>
<td>0.56</td>
</tr>
</tbody>
</table>

**Figure 7.10:** Ground truth and result for keypoint detection (test image)
7.3 Approach 2

It is often hard to distinguish between corners of near neighboring buildings. This is mainly caused by the inaccuracy of the ground truth data (see Chapter 2.2) which prohibits the use of smaller circles around the corner points. The corner detection performance is crucial, as we do so far not have a method to re-detect lost building corners at a later stage.

We have quickly tried to formulate corner detection as a multi class classification problem where corners of different orientations belong to distinct classes. We hope that by doing so it is possible to prevent merger of neighboring corners. So far it is hard to train a U-Net and achieve good results - further examination of this idea is left as future work.

7.3.2 Edge Weight Calculation

Edge Weights are calculated by a triamese network. This network is trained and evaluated on predictions for edges between ground truth corner positions. The training set consists of all edges between corners that are not farther than 55 pixels away from each other. There are more negative samples (corners that do not belong to the same building) than positive samples. Training works best, if balanced data is used, so at each iteration step a batch with 50% positive and 50% negative samples is considered. For evaluation we use the complete test set - from each test image all potential pairs of corners with a distance up to 55 pixels.

The F1-score is plotted in Figure 7.11. The curve is rather flat, indicating that the results are not very sensitive with respect to the classification threshold. The corresponding ROC curve is illustrated in Figure 7.12. The area below the curve is large and predictions pretty good. Two example results are shown in Figure 7.13. White lines represent corner connections with predicted score above 0.5. This visualization does thus not exactly represent the best solution for this step which according to the F1-score plot (Figure 7.11) would be achieved for a threshold of around 0.6. But we are going to submit all scores to the Normalized Cuts step, so we are not directly interested in what the best classification threshold would be.

![Figure 7.11: F1-score of prediction with respect to a corner classification threshold](image1)

![Figure 7.12: ROC curve for triamese network](image2)
By looking at the example results it becomes clear, that the maximum distance between corners of the same building may exceed 55 pixels. So far we ignore these cases due to computational effort which gets higher the longer the maximum distance considered. For small buildings we get most of the time reliable predictions, except in some cases where corners from neighboring buildings lay near together.

7.3.3 Graph Segmentation

**Evaluation of Graph Segmentation for Ground Truth Corner Points**

First, we evaluate graph segmentation in combination with edge weight calculation separately, which means we use ground truth corner positions, calculate edge weights between such points and feed these weights to Normalized Cuts. Normalized Cuts emits several clusters. For each cluster we check from which ground truth building it contains the most points and assign that building to the cluster. All points within the cluster which belong to this building get the label 'true positive', points that are not part of the building 'false positive'. Points that belong to the ground truth building but are not included in the cluster are 'false negatives'. 'True positives', 'false positives' and 'false negatives' are calculated for each cluster and all test images and summed up. By the end we achieve the following values for recall, precision and F1-score:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.61</td>
</tr>
<tr>
<td>Precision</td>
<td>0.87</td>
</tr>
<tr>
<td>F1-Score</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Two image patches with corners belonging to the same cluster colored alike are shown in Figure 7.14. Given accurate corner positions, N-cut does often work well. On top of the left
image we see buildings which are rather big. The score between some of their corners might have been set to zero and not calculated, as described in 7.3.2, what leads to errors here.

![Figure 7.14: Two example results for the graph segmentation step. Corners that are clustered together are colored similarly](image)

**Evaluation of Graph Segmentation for U-Net Corner Points**

The evaluation for corner points that are detected using the U-Net is slightly different. A priori it is not known, which points should be clustered together. In order to find the correct building for a point, we seek the nearest ground truth corner and assign the point to the same building. Points whose distance to all ground truth corners is larger than a predefined threshold (10 pixel) are labeled as background points and thus 'false positives', independently on what cluster they later on get assigned to. Our newly adjusted ground truth buildings do all contain some of the predicted corners (original corner positions are removed for evaluation), but it may happen that there are incomplete buildings (e.g. only three corners detected). The amount of clusters is set to be the number of candidate points divided by four. First, we cluster all candidate corner points using normalized cuts and proceed similar as before when we evaluated the algorithm based on corners given from ground truth. We achieve:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.80</td>
</tr>
<tr>
<td>Precision</td>
<td>0.55</td>
</tr>
<tr>
<td>F1-Score</td>
<td>0.65</td>
</tr>
</tbody>
</table>

Second, we want to take care of points that are not found during corner detection and thus also not part of the clustering. We add them to the 'false negatives' and obtain lower recall and F1-score:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.30</td>
</tr>
<tr>
<td>Precision</td>
<td>0.55</td>
</tr>
<tr>
<td>F1-Score</td>
<td>0.39</td>
</tr>
</tbody>
</table>
We think that this second way to calculate the F1-score better represents the overall quality of the algorithm. Nevertheless, our evaluation method shows some drawbacks. We do not work on the ground truth buildings directly and if there is no cluster assigned to a certain building, this object does only enter the evaluation through 'false positives' of other buildings. The evaluation method proposed should be seen as an attempt to quantify the quality of Approach 2, but interpreted with caution.

Again, we show two image patches with corners that are clustered together in similar color (see Figure 7.15). The bad quality of predicted corners degrades the results’ quality visibly. For example on the right top corner of the right image there are buildings which are just ignored. Further it may happen that objects are merged.

![Figure 7.15](image)

Figure 7.15: Two example results for the graph segmentation step. Corners that are clustered together are colored similarly

We can finally conclude that the triamese network and N-cut perform well but only under the condition that input corner positions are accurate.

7.3.4 Evaluation of the Whole Pipeline and Suggestions of Improvement

This approach consists of several nested steps and we observe error accumulation. Figure 7.16 depicts the whole pipeline. For a test image we show first the input image, second U-Net scores for corner detection. After that we split the illustration in two rows. On top we draw corners found by Non-Maximum Suppression, followed by edge weight calculation where edges with a score above 0.5 are shown in white. The image on the top right shows the output after normalized cuts and thus the result after running the whole pipeline. The second row of the illustration depicts results that are obtained assuming given corner positions. It becomes clear that the degree of details drops, once we work with predicted corner positions. The problem of near corners that are merged can not only be observed within buildings but also in between distinct objects. The first step where corners are detected seems to be the most
We believe that better ground truth data would allow for smaller circles around the corners and thus more precise predictions. Further it might make sense to work with images in a higher resolution (we have them) as well as to partition corners into different classes corresponding to their orientation. If corners were classified depending on the orientation it would be much easier to distinguish between neighboring corners.

The second step, edge weight calculation, leads to satisfactory results. What should be improved, is computational efficiency. A way to do so is described in Chapter 6.2.1. We propose to replace the convolutions of the triamese network by a network working on the image as a whole. This network might even share weights with the U-net for corner detection.

The third step, normalized cuts, works as desired but it is not possible to learn any constraints on building shapes. To incorporate such constraints we might have to replace normalized cuts by a neural network which works on building corners and edges or maybe just on potential building edges. Different ways to perform pooling and convolution operations on non-grid structured graphs have been proposed [30, 29], but there exists not yet much work for networks on arbitrary graph. We believe that this is an interesting and promising research direction.

The probably most important improvement would be to combine all steps and train them end-to-end. Back-propagation through Non-Maximum Suppression of the corner detection step as well as through Normalized Cuts constitute difficulties. [47] interrupt back-propagation for NMS within their algorithm, but this is for sure not the most elegant solution. Back-propagation through Normalized Cuts has been studied and solved in [62] (for an extended
version see [64]).
7.4 Comparison of Approach 1 and Approach 2

Approach 1 achieves a F1-score of around 0.58 if evaluated corner based, Approach 2 0.39. This indicates that Approach 1 is more accurate, what is consistent with our observations. Nevertheless, the situation is not exactly the same for both methods. Approach 1 would get a perfect F1-score of 1 if all rectangular boxes were correctly predicted - 4 bounding box corners per building. The evaluation of Approach 2 on the other hand, requires that every single building corner is correctly detected and assigned to the corresponding building.

Approach 1 has the advantages that it is end-to-end trainable, building geometry is exploit directly and performance does not that much suffer if buildings are partially occluded. On the other hand this approach is not flexible and only able to predict rectangular shapes. The probability that a building is correctly detected (4 corners that build an outline), is higher for this method, but the localization is much better for Approach 2. Given that building corners are found, they do usually match well with the ground truth positions. By construction, Approach 1 can not handle complex shapes and in addition it has a weakness when it comes to detection of long narrow buildings. We assume that this is due to the fact that all anchor boxes have a side length aspect ratio of 1 : 1 or 1 : 2. Approach 2 is very flexible but due to computational costs edge weights are only evaluated within a neighborhood of 55 pixels. This prevents the algorithm from working on large objects.
8 Conclusion and Future Work

We present two methods for building polygon extraction from aerial images. Approach 1 is designed such that the image grid is kept as long as possible. This confers a benefit on simplicity as there are many known tools to work efficiently on grid structured graphs. Building geometry is exploit directly and object proposals do always show a plausible shape. On the other hand this approach is not capable of predicting shapes different to rectangles. Our second approach is much more flexible but not yet very stable in terms of prediction quality. It consists of several nested steps and shows the disadvantage of error accumulation. Corner detection is the first step and the weak point. Unfortunately, this has an influence on the algorithm’s performance. Nevertheless our results show a proof of concept for each of the methods and for some images very accurate building polygons.

Considering Approach 1, we believe that training on even more data and adding anchor boxes of different aspect ratio would lead to further improvements. The feature map created does have a limited resolution which could be enhanced by addition of deconvolutional layers. In other words, the VGG part could be replaced by an U-Net [3]. The downside is the increase of computational effort for network training. Our method is capable of predicting arbitrary rotated rectangular shapes. More complex shapes could be treated as compositions of rectangles or the library of anchor boxes extended. An enlargement of this library would involve a new interpretation of the RoI pooling. We believe that this is possible for some but not arbitrary shapes.

Especially the corner detection step of Approach 2 suffers from the inaccuracy of our data. Precise target locations could facilitate the distinction between corners that lay near together. It might be worth to review the approach on a hand labeled data set. As mentioned earlier, we would further propose to improve corner detection by enhancement of the input image resolution and differentiation between corners of different orientation. The second part of the algorithm where edge weights are calculated works reliably, but we wish to improve efficiency. For ideas on how to do this check out Chapter 6.2.1. We would further like to train Approach 2 end-to-end and hope to enhance its performance through this. Non-Maximum Suppression (NMS) as applied for corner extraction as well as Normalized Cuts preformed for instance segmentation are seen as critical points when it comes to end-to-end training. We have not yet derived back propagation through NMS but a solution for Normalized Cuts can be implemented as proposed by [64] who discuss the training of deep networks with structured layers.

It is conceivable to combine the two algorithms. One possibility is to link object detection as performed by Approach 1 with corner detection and polygon construction developed in scope of the second approach. A different course of action is prediction of rectangular shapes and connection of belonging together objects by use of ideas presented in Approach 2. The probability that rectangles originate from the same building could be calculated by a
siamese network acting on the two RoI pooling output vectors.

We propose two algorithms for building polygon extraction. In both cases we leave the image grid structure, but at different time. We hand-craft many steps in order to let the networks learn how to start with an image and end up with a vector representation of our objects. It would be of great interest to develop techniques that facilitate work on non-grid structured graphs. [30, 29] show how to define convolution and pooling operations on arbitrary graphs, but the problem of automatic graph generation is not yet solved. Our long-term goal is to develop algorithms capable of starting with any graph as input and produce a different output graph of previously unknown structure. We are convinced that this type of work can have a broad influence not only in the field of computer vision.
Bibliography


Appendix

1 Layers for Approach 1

In the following we describe anchor target layer, proposal layer, proposal target layer and RoI pooling layer. All layers are explained for Faster R-CNN adapted to our problem where rotation is estimated as well.

1.1 Anchor Target Layer

- **Goal:** Assign anchors to ground truth targets. Produce anchor regression targets classification labels (0: background, 1: object, -1: don’t care and is not used for evaluation).

- **Algorithm:**

  - **Input:** ground truth boxes
  - **Output:** classification labels, target boxes for all anchors, weights for evaluation

  for each \((H,W)\) location \(i\) generate 9 anchor boxes centered on cell \(i\);
  - throw away anchor boxes that cross the image border;
  - initialize all remaining labels to \(-1\);
  - check intersection over union (IoU) overlaps between all anchors and ground truth boxes;
  - **for each ground truth box do**
    - set label of anchor box with maximum overlap to 1
  - **end**
  - **for each anchor do**
    - find the 'best ground truth box' (highest IoU) and corresponding 'max overlap';
      - if \(\text{max overlap} > 0.7\) then
        - set label to 1
      - end
      - if \(\text{max overlap} < 0.3\) then
        - set label to 0
      - end
  - **end**

  subsample 128 anchors with label 0 and 128 anchors with label 1 (they are later on used for the RPN loss evaluation) and set all remaining labels to -1;
  - return labels, anchor box targets with corresponding rotations and a weight tensor with non-zero entries for anchor boxes with label 1 only;

**Remark:** The weight tensor can later on be multiplied with differences between ground truth and predicted anchor regressions → that way only positive anchor boxes count to the bounding box regression loss function.
1.2 Proposal Layer

- **Goal:** Find the best proposals by looking at their predicted score.

- **Algorithm:**

  **Input:** predicted score and regression parameters for all anchor boxes  
  **Output:** best proposed boxes  
  for each \((H, W)\) location \(i\) generate 9 anchor boxes centered on cell \(i\) and apply predicted regression parameters for side lengths and position;  
  **for each box do**  
  clip box to image;  
  **if** height < threshold or width < threshold **then**  
  remove this box  
  **end**  
  **end**  
  sort all (proposal, score) pairs;  
  apply non-maximum suppression (NMS) and choose a given amount of remaining boxes with highest scores;  
  return these boxes with corresponding orientations and scores;

1.3 Proposal Target Layer

- **Goal:** Assign object proposals (RoIs) to ground truth targets. Produce classification labels and bounding box regression targets.

- **Algorithm:**

  **Input:** candidate RoIs as selected within Proposal Layer  
  **Output:** RoIs, labels, targets for regression (side lengths, position, orientation)  
  include ground truth boxes in the set of candidate RoIs;  
  calculate overlaps between candidate RoIs and ground truth boxes;  
  **for each candidate RoI do**  
  find ground truth box with highest IoU overlap score;  
  calculate target regression parameters for the RoI;  
  **if** score > foreground_threshold **then**  
  keep this RoI as foreground RoI  
  **end**  
  **if** background_threshold_low < score < background_threshold_high **then**  
  keep this RoI as background RoI  
  **end**  
  **end**  
  randomly sample foreground and background RoIs;  
  return sampled RoIs with corresponding target regression parameters (including rotation) and a weight matrix (that has zero entries for all background RoIs);  
  **Remark:** The weight tensor can later on be multiplied with differences between ground truth and predicted regressions.
1.4 RoI-Pooling Layer

- **Goal:** Reshape the region proposals such that all of them have similar shape.

- **Algorithm:**

  **Input:** RoIs, orientations, feature map  
  **Output:** reshaped boxes which represent that part of the feature map which lies within the corresponding regions of interest

  \[
  \text{for each RoI do} \quad \text{perform RoI-pooling as described in section 4.3 but instead of pooling elements of an x- and y-axis aligned rectangle, pool the elements corresponding to a rectangle that is rotated.} \\
  \text{end} \\
  \text{return all reshaped boxes}
  \]

2 Triamese Network for Edge Weight Calculation

The triamise network consists of three convolutional neural networks acting on the patches in parallel and a second network part consisting of fully connected layers. The convolutional networks share weights and are of the following structure:

<table>
<thead>
<tr>
<th>shared layers</th>
<th>type</th>
<th>filter size</th>
<th>stride</th>
<th>depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td>5 × 5</td>
<td>1</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>pool</td>
<td>2 × 2</td>
<td>2</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>5 × 5</td>
<td>1</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>pool</td>
<td>2 × 2</td>
<td>2</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>5 × 5</td>
<td>1</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>pool</td>
<td>2 × 2</td>
<td>2</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>5 × 5</td>
<td>1</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

The outputs calculated based on the three patches (each of size \( p \times p = 40 \times 40 \)) are concatenated and fed to a stack of fully connected layers:

<table>
<thead>
<tr>
<th>layers after concatenation</th>
<th>type</th>
<th>input depth</th>
<th>output depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>fully</td>
<td>150×3 × ( \frac{p}{8} \times \frac{p}{8} )</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>fully</td>
<td>200</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>fully</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>fully</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>fully</td>
<td>100</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>fully</td>
<td>50</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

The final output is a vector of shape of 2, where the two numbers are interpreted as log probabilities.

All fully connected layers except the last one are trained with dropout. In order to prevent over-fitting we choose a high drop out rate and thus a keep probability of 0.3.