

Master Thesis

Image Segmentation using Convolutional Neural Net- works for Change Detection of Landcover

Generating building footprints from orthophotos
and Open Street Map data

Martin Boos
27. April 2018



HSR

HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

This page is intentionally left blank.

Abstract

tbd

Acknowledgments

Put your acknowledgments here.

Prof. Stefan Keller tbd

Contents

1	Examples	1
1.1	Equations	1
1.1.1	some equations	1
1.1.2	Listings	3
2	Management Summary	5
3	Overview	7
3.1	Was ist in welchem Kapitel etc.	7
4	Introduction	9
4.1	Technical Background - PNF / AV	9
5	Image Segmentation with Convolutional Neural Networks	11
5.1	Introduction	11
5.2	Convolution Layers	11
5.3	Pooling Layers	11
5.4	Fully Connected Layer	11
5.5	Mask R-CNN	11
6	Theoretical and Practical Challenges	13
6.1	Training data generation	13
6.2	Building outline regularization	13
6.2.1	Contour extraction	15
6.2.2	Line segmentation	17
6.2.3	Create orientation classes	17
6.2.4	Create neighbourhoods	18
6.2.5	Update line orientation	19
6.2.6	Gap filling	19

7 Theoretical and Experimental Results	21
7.1 Training data	21
7.2 Microsoft COCO	21
7.3 Building detection	21
8 Practical Results	23
8.1 QGIS Plugin	23
8.1.1 Wie es von den Leuten genutzt wird	23
8.1.2 Effizienzsteigerung	23
9 Conclusion	25
Appendices	
List of Figures	29
List of Tables	31
Bibliography	33
Index	35

Examples 1

This chapter is to demonstrate some of the capabilities of this L^AT_EX template. Please take a good look at this chapter and try to follow the guidelines.

GOAL OF THIS
CHAPTER

1.1 Equations

1.1.1 some equations

Equations can easily be written using the *Equation* environment. Inline equations are inserted with $\sqrt{-1} = i$. Equations can also be labeled, so it is possible to reference them. This should be done for all important equations.

EQUATIONS

$$x^2 + y^2 = 1 \tag{1.1}$$

There are several environments for multi line equations. A very useful one is *align*, see equation (1.4).

$$\oint \vec{E} \cdot d\vec{A} = \frac{q}{\epsilon_0} \tag{1.2}$$

$$\oint \vec{B} \cdot d\vec{A} = 0 \tag{1.3}$$

$$\oint \vec{E} \cdot d\vec{s} = -\frac{d\Phi_B}{dt} \tag{1.4}$$

Images are always inserted inside a *figure* environment. If possible, it is advisable to use [tb] as position. Always remember to add a caption and a label, so you can reference the image like this: Fig. 1.1. If possible, images

FIGURES AND
TABLES



FIGURE 1.1 An example image

some	text	is shown	here
there is more	text here	and here	cool.
and	even	more	here.

TABLE 1.1 A sample table

should be inserted as vector graphics, e.g. eps or pdf - or even drawn manually in TikZ.

Tables can be used quite similarly. They are inserted inside a *table* environment, as shown in Tab. 1.1.

Another useful tool is the *tabularx* environment. It lets the user specify the total width of the table, instead of each column. An example is shown in Tab. 1.2.

Please read the documentation of the *booktabs*¹ package to find information on how to create good tables. Always remember the first two guidelines and try also to stick to the other three:

1. Never, ever use vertical rules.
2. Never use double rules.
3. Put the units in the column heading (not in the body of the table).
4. Always precede a decimal point by a digit; thus 0.1 *not* just .1.
5. Do not use „ditto“ signs to repeat a value. In many circumstances a blank will serve just as well. If it won't, then repeat the value.

PARAGRAPHS Note that each paragraph is ended with an empty line. *Never* use `\\` to end paragraphs – this is a new line, not a new paragraph. Also try to keep your source code clean: about 80 characters per line. Using source control makes your life much easier

QUOTES Quotes can easily be made using the „csquotes“ package. Citing text passages is

¹ <http://www.ctan.org/pkg/booktabs>

some	text	is shown here
there is more	text here	and here.
and	even	more here.

TABLE 1.2 Tabularx example

easily done: „First argument: citation, second argument: terminal punctuation!“
(me) Whole block quotes are also easily possible.

Formal requirements in academic writing frequently demand that quotations be embedded in the text if they are short but set oV as a distinct and typically indented paragraph, a so-called block quotation, if they are longer than a certain number of lines or words. In the latter case no quotation marks are inserted.

Any numbers and units should be typed using the siunitx package. Numbers are written as 1234.345×10^{-5} , 1, 2 and 4 to 30 10° $5^\circ 3' 2''$. Units are written with kg m/s² or 14 F⁴. Of course also possible is 10 m, 40 m and 12 m or -40°C to 125°C . Almost every unit you could possibly think of is implemented!

SI UNITS

The bibliography is created using *Bibtex*. The standard format is set to `ieeetr`, which is the IEEE Standard. There are example entries for different types of in the separate bibliography file **article book booklet conference inbook incollection manual mastersthesis misc phdthesis proceedings techreport unpublished**.

BIBLIOGRAPHY

All glossary entries are made in the separate file `glossary.tex`. They can then be used with Equation. Acronyms are defined as shown there and used similarly. The first time, it will be *support vector machine (SVM)*. The second time: *SVM*. The glossary has to be created manually by invoking `makeindex -s doku.ist -t doku.glg -o doku.gls doku.glo`. The index is simply created by using `index{text}`. It is generated automatically.

INDEX & GLOSSARY

1.1.2 Listings

Listings are created by the `lstlistings` package.

This is a simple ToDo note

ToDo

This is a small note [Citation needed]

ToDo
ToDo

Management Summary **2**

Overview **3**

3.1 Was ist in welchem Kapitel etc.

Introduction 4

4.1 Technical Background - PNF / AV

Image Segmentation with Convolutional Neural Networks

5

5.1 Introduction

5.2 Convolution Layers

5.3 Pooling Layers

5.4 Fully Connected Layer

5.5 Mask R-CNN

Theoretical and Practical Challenges **6**

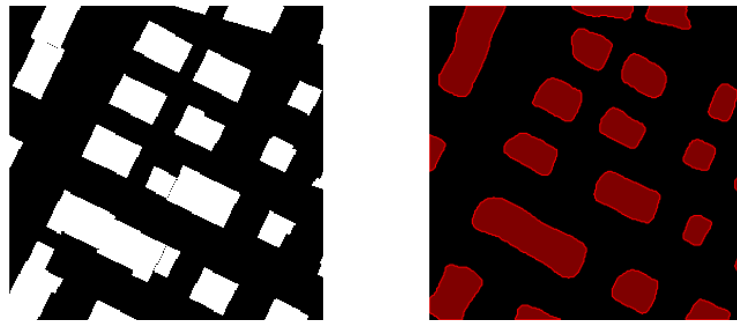
6.1 Training data generation

In order to train the neural network a data set was required, which could then be used for training and validation. Due to this, a tool [1] has been developed which uses publicly available vector data from OpenStreetMap and satellite imagery Microsoft Bing.

tbd

6.2 Building outline regularization

Once the network has been trained, it can be used to make predictions on images, it has never "seen" before. However, there are situations in which the predictions are far from perfect, especially then if the building is partially covered from trees or has unclear outlines. This can be seen in Fig. 6.1.



(A) Actual ground truth

(B) Predicted building masks

FIGURE 6.1 Predicted masks and actual ground truth

As a result of the inaccuracies in the predicted building masks the contours of these predictions can not directly be used to create the vectorized outlines. Instead, the predictions have to be regularized. The approach used is similar to [2] and described in Fig. 6.2.

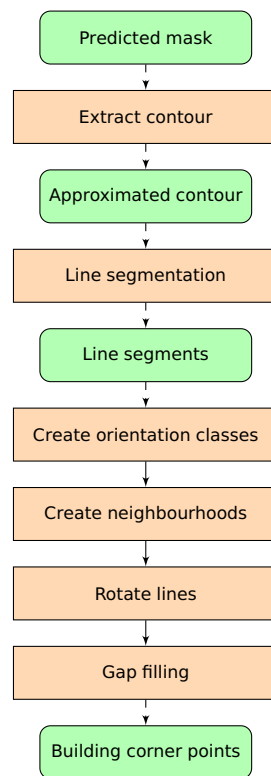


FIGURE 6.2 Rectangularization procedure

The single steps are described in detail in the following sections.

6.2.1 Contour extraction

The first step of the building outline regularization procedure consists of getting the contour from the predicted mask, which covers the whole building. The extraction is done using the marching squares algorithm [3]. In this algorithm, a square consisting of four cells is moved (marched) along the contour in such a way, that at least one cell always covers the object to be contoured. Additionally, the square always has a state, which is derived from the content of its cells, according to (6.1). The cells are traversed in counter clockwise order.

$$\begin{aligned}
s &= \sum_{i=0}^3 2^i f(c_i) \\
&= 2^0 * f(c_0) + 2^1 * f(c_1) + 2^2 * f(c_2) + 2^3 * f(c_3) \\
&= f(c_{bottomLeft}) + 2 * f(c_{bottomRight}) + 4 * f(c_{topRight}) + 8 * f(c_{topLeft})
\end{aligned} \tag{6.1}$$

where:

c_i : The value of the cell i

c_0 : The bottom left cell

and

$$f(c_i) = \begin{cases} 0 & \text{if } c_i \leq 0 \\ 1 & \text{if } c_i > 0 \end{cases}$$

As soon as the contour has been extracted, its number of points will be reduced using a Douglas-Peucker algorithm [4]. The reason for this is, that the contour has pixel accuracy. That means, there may be several points on the same horizontal or vertical line, even though, the startpoint and endpoint of each such line would be enough, to represent the line. Additionally, the lower the number of points per contour is, the faster the following processing will be.

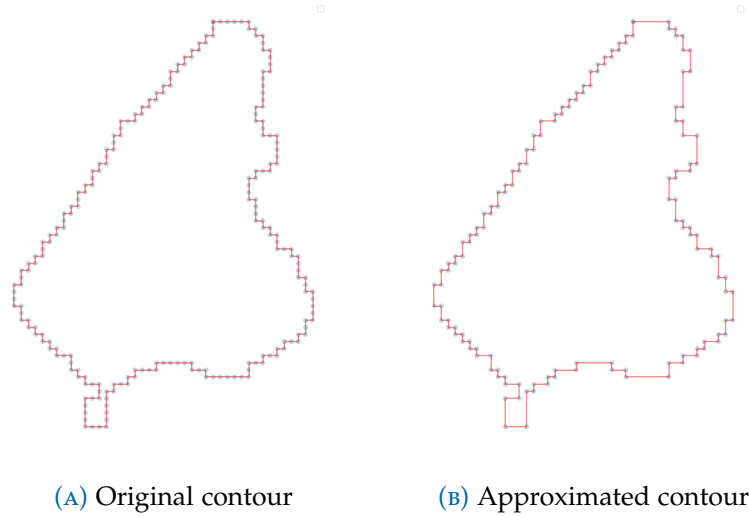


FIGURE 6.3 Contour before and after approximation

6.2.2 Line segmentation

Once the contour has been extracted, it is split into multiple line segments. For this, the main direction of the building is determined using the Hough lines algorithm. The result of this is the line which hits the greatest number of points. The angle between this line and a horizontal line gives the main building orientation.

Once the main building direction is known, the line segmentation starts at the point which has the smallest distance to this line. The whole procedure is depicted in algorithm 1.

Data: Contour points, startpoint

Result: Lines

rearrange points so that startpoint == points[0]

lines = []

while *any points left* **do**

 segment = remove first 3 elements from points

while *points is not empty* **do**

 p = points.first()

 err = root mean square error of distance between segment.last() and

 p

if *err > threshold* **then**

 break

end

 segment.append(p)

 points.remove(p)

end

if *segment.length() >= 3* **then**

 line = fit line to points of segment

 lines.append(line)

end

end

Algorithm 1: Line segmentation

6.2.3 Create orientation classes

After the line segmentation, an orientation will be assigned to each line. Generally, most of the buildings consist of mainly right angles. However, there are still buildings, for which this assumption is not true. Due to this, orthogonality

will be preferred, but other angles will still be possible. Algorithm 2 shows the procedure, which assigns a main orientation class to each line and defines the lines parallelity / orthogonality to the longest line of the same orientation class.

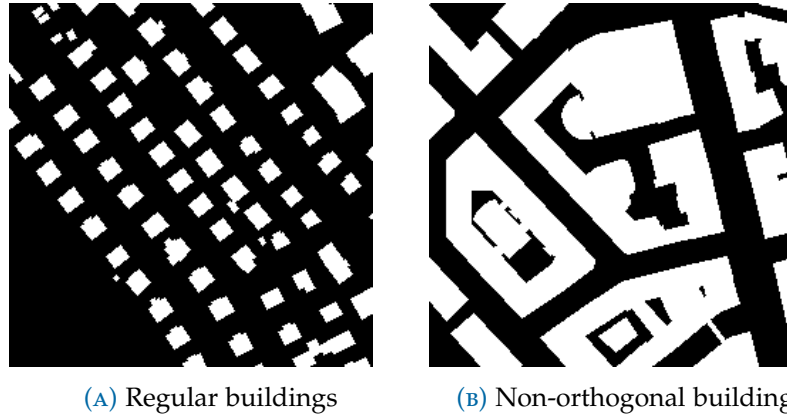


FIGURE 6.4 Different kinds of buildings with regard to their corner angles

```

Data: Lines, angleThreshold
while any line without orientation do
    line = longest of unprocessed lines
    line.orientation = angle between line and horizontal-line
    foreach line li without orientation do
        a = angle between line and li
        if  $a \leq \text{angleThreshold}$  then
            li.orientation = line.orientation li.orthogonal =
            line.orthogonalTo(line)
        end
    end
end

```

Algorithm 2: Orientation assignment

6.2.4 Create neighbourhoods

At this point, each line segment belongs to a orientation class and the parallelity / orthogonality of each line to the orientation classes main line is known. However, the spatial location of each line has not been taken into account yet, which is done in this step. Clusters of neighbouring lines are created within each orientation class. As a result of this, it is now possible to find

lines, which may be better placed in an other orientation class. This is based in the assumption, that it is improbable, that a line k of the orientation class x is surrounded by lines of the orientation class y . In this case, the line k will be assigned to the orientation class y .

6.2.5 Update line orientation

Finally, the lines will be adjusted to their orientation class with respect to each lines parallelity / orthogonality. The result of such an adjustment, can be seen in Fig. 6.5, which shows a single orientation class and lines, which have been adjusted either parallel or orthogonal to the orientation class.



FIGURE 6.5 Adjusted lines

6.2.6 Gap filling

In order to create the final building outline, the only thing left to do is to fill the gaps between the line segments.

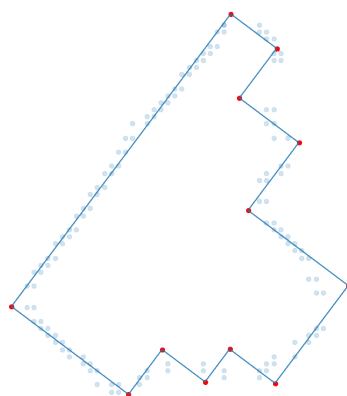


FIGURE 6.6 Final building outline

Theoretical and Experimental Results **7**

7.1 Training data

7.2 Microsoft COCO

7.3 Building detection

Practical Results **8**

8.1 QGIS Plugin

8.1.1 Wie es von den Leuten genutzt wird

8.1.2 Effizienzsteigerung

Conclusion 9

Appendices

List of Figures

1.1	An example image	2
6.1	Predicted masks and actual ground truth	14
6.2	Rectangularization procedure	15
6.3	Contour before and after approximation	16
6.4	Different kinds of buildings with regard to their corner angles .	18
6.5	Adjusted lines	19
6.6	Final building outline	20

List of Tables

1.1	A sample table	2
1.2	Tabularx example	3

Bibliography

- [1] M. Boos, *Airtiler*, <https://github.com/mnboos/airtiler>, [Online; accessed 23-March-2018], 2018.
- [2] T. Partovi, R. Bahmanyar, T. Kraus, and P. Reinartz, „Building outline extraction using a heuristic approach based on generalization of line segments“, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 3, pp. 933–947, 2017, ISSN: 1939-1404. DOI: [10.1109/JSTARS.2016.2611861](https://doi.org/10.1109/JSTARS.2016.2611861).
- [3] C. Maple, „Geometric design and space planning using the marching squares and marching cube algorithms“, in *2003 international conference on geometric modeling and graphics*, E. e. Banissi and M. Sarfraz, Eds., IEEE Comput. Soc, 2003, pp. 90–95, ISBN: 0-7695-1985-7. DOI: [10.1109/GMAG.2003.1219671](https://doi.org/10.1109/GMAG.2003.1219671).
- [4] D. H. DOUGLAS and T. K. PEUCKER, „Algorithms for the reduction of the number of points required to represent a digitized line or its caricature“, *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973, ISSN: 0317-7173. DOI: [10.3138/FM57-6770-U75U-7727](https://doi.org/10.3138/FM57-6770-U75U-7727).

Index

Equation, 1

Image, 1

Table, 2

tabular, 2

tabularx, 2