Bachelor's thesis Media Technology

Dwelling Detection on VHR satellite imagery of Refugee/ IDP camps using Faster R-CNN

presented by

Lorenz Wickert

Name 1st examiner: Prof. Dr. Arnulph Fuhrmann (Technische Hochschule Köln)

Name 2nd examiner: Dipl.-Inform. Manfred Bogen, Ph. D. (Fraunhofer IAIS, Sankt Augustin)

September 2019

Fakultät für
Informations-, Medien-
und Elektrotechnik

Technology
Arts Sciences
TH Köln

Bachelor Thesis

Title: Dwelling Detection on VHR satellite imagery of Refugee/ IDP Camps using Faster R-CNN
Reviewers:

- Prof. Dr. Arnulph Fuhrmann
- Dipl.-Inform. Manfred Bogen, Ph.D.

Abstract: This Bachelor Thesis describes a new method for dwelling detection on Very High Resolution (VHR)-Satellite imagery using Faster-RCNN developed for the BMBF-project HUMAN+[1]. HUMAN+ aims to develop a real-time situational awareness application for efficient migration management to guarantee humanitarian security. The described method and a corresponding workflow are used in a remote sensing module in the input layer of the HUMAN+ application. It analyses VHR-satellite images of refugee camps, finds all dwellings on the image and calculates an estimate of the number of tents (i.e. dwellings) and people in the camp. To find the dwellings, a Faster R-CNN is used. R-CNNs build a special neural network on a regular CNN to use them for object detection. This thesis describes the generation of training data, the training of a Faster R-CNN and the utilization of the trained Faster R-CNN in a dwelling detection application.
Keywords: Machine Learning (ML), Faster R-CNN, Remote Sensing (RS), Dwelling Detection, Object Detection
Date: September 9, 2019

[1]https://www.sifo.de/files/Projektumriss_HUMAN-Plus.pdf

Acknowledgements

First, I would like to thank Prof. Dr. Arnulph Fuhrmann for this research and development opportunity, for agreeing to be the supervisor of this thesis, and for assisting and supporting me in completing it.

Second, I would like to thank my second supervisor at Fraunhofer IAIS and project manager in the HUMAN+ project, Dr. Manfred Bogen, for his constant assistance, invaluable expertise, and support during the writing of this thesis.

Special thanks go to all my colleagues at Fraunhofer IAIS ART for their precious ideas and help. In particular, I want to thank Dr. Marvin Richter for the good cooperation in the HUMAN+ project, Dr. Stefan Rilling for giving constant motivation while developing the application, Benjamin Wulff for great support in AI matters and technical issues, and my office mate Fernando Morillo for discussing my thesis with me and for sharing an office.

Finally I want to thank my family and my friends for their ongoing support up to the completion of my bachelor thesis and beyond.

Contents

1. Introduction

1.1. The HUMAN+ Project

This Bachelor Thesis is written in the context of the HUMAN+ project at the Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS[1] in the business area Preventive Security[2]. Fraunhofer IAIS is an institute of the Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. Fraunhofer, Europe's largest application-oriented research organization[3]. The main focus of Fraunhofer IAIS is processing of big data and its analysis using artificial intelligence machine learning[4].

HUMAN+ is a German-Austrian cooperation project[5]. The project is funded by the Bundesministerium für Bildung und Forschung with the Verein Deutscher Ingenieure (VDI) as project management on the German side and by the Bundesministerium für Verkehr, Innovation und Technologie und dem österreichischen Förderungsprogramm für Sicherheitsforschung (KIRAS) on the Austrian side. The aim of the project is to ensure humanitarian safety through real-time situational awareness for efficient management of migration movements.

1.1.1. Project Background

In 2015/16, the biggest migration movements to Germany after the Second World War occurred. The migration movements were mainly caused by the civil war in Syria and the terror organization IS. While being high throughout the whole year, the number of refugees arriving in Germany and Austria intensified in September 2015 after the Dublin Regulation[6] was deferred. On September 4-5, 2015, Syrian refugees who were stranded in Hungary were allowed to come to Germany. At peaks, more than ten thousand refugees were crossing the Austrian-German border a day.

Due to a short reaction time and the high intensity of the migration movements, the humanitarian and administrative challenges were high. Refugees had to be registered, forwarded

[1] https://www.iais.fraunhofer.de/en.html
[2] https://www.iais.fraunhofer.de/en/business-areas/preventive-security.html
[3] https://www.fraunhofer.de/en/about-fraunhofer/profile-structure.html
[4] https://www.iais.fraunhofer.de/en/institute/about-us.html
[5] https://www.iais.fraunhofer.de/de/geschaeftsfelder/praeventive-sicherheit/praeventive-sicherheit/referenzprojekte/HUMANPlus.html
[6] https://ec.europa.eu/home-affairs/what-we-do/policies/asylum/examination-of-applicants_en

and be given supplies and accommodation. Problems were caused by the high number of people as well as by the lack of quickly available information, making humanitarian and administrative operations challenging, expensive and cumbersome.

It is almost certain, that the next migration movement will come. Climate change, unstable political situations, war and poverty make people leave their home to find shelter. In 2018, the United Nations High Commissioner for Refugees (UNHCR) has reported, that "[. . .] the worlds forcibly displaced population remained yet again at a record height" (UNHCR, 2018). The HUMAN+ goal is to be better prepared next time.

1.1.2. The HUMAN+ concept

In HUMAN+, different approaches are used to support humanitarian operations when migration movements are happening. HUMAN+ is going to support decision making by offering a compact situation report consisting of a real-time map and real-time data, short-term forecasts and standardized interfaces. Those tools aim to make humanitarian aid operations better. In particular having data about refugees in a refugee camp near a border which normally is not available to authorities, municipalities, relief organizations, NGOs, as well as the police managing a refugee situation, will ease the decision making in general and the allocation of resources such as medical aid, food, clothes, tents, and transportation in particular. This data is often hard to get in concrete humanitarian operations for those participating in these operations: New camps form quickly and data from the field is often not available.

Therefore new, multisensory data sources are retrieved. HUMAN+ includes data from Social Media, In-Situ optical and thermal images at border stations, remote sensing imagery and input from operational forces working on the ground. Remote sensing and social media data is analyzed using machine learning methods. The different data sources are bundled, processed and shown on a real-time situational awareness map. Further, the received information will be accessible by software such as Vomatec's ARGION® PLUS[7] and CrisCom's Commander[8], - both companies are project partners in the HUMAN+ project -, already used in humanitarian aid operations through standardized interfaces.

The project is driven by the principle of "User centered Design". This means, in short, that the demand carriers and end users of the software are participating in the development of the HUMAN+ applications from the beginning. Further, due to the politically sensitive topic of the project, an ethical board is reviewing the development under ethical and legal aspects.

[7]https://vomatec-innovations.de/produkte/industrie-privatwirtschaft/produkte/arigon-plus.html
[8]https://www.criscom.solutions/produkte?lang=en

1.2. The Remote Sensing Module

Fraunhofer IAIS is responsible in the HUMAN+ project for Remote Sensing analysis, data processing and data supply. This Bachelor thesis describes the development of the Remote Sensing module.

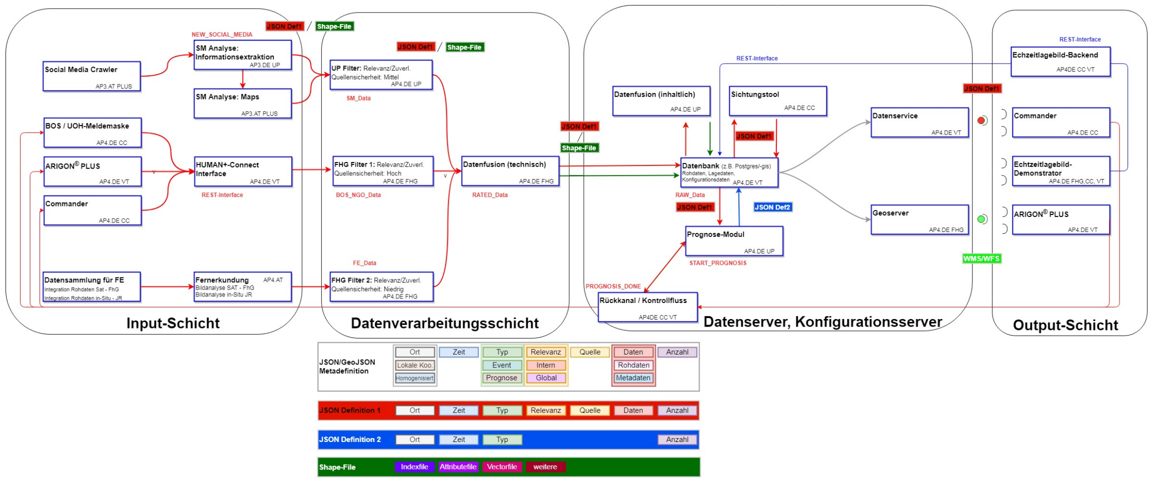

JSON Def1 / Shape-File
NEW_SOCIAL_MEDIA
SM Analyse: Informationsextraktion AP3.DE UP
Social Media Crawler AP3.AT PLUS
SM Analyse: Maps AP3.AT PLUS
BOS / UOH-Meldemaske AP4.DE CC
ARIGON® PLUS AP4.DE VT
HUMAN+-Connect Interface AP4.DE VT
REST-Interface
Commander AP4.DE CC
Datensammlung für FE
Integration Rohdaten Sat - FhG
Integration Rohdaten in-Situ - JR
Fernerkundung AP4.AT
Bildanalyse SAT - FhG
Bildanalyse in-Situ JR
Input-Schicht
JSON Def1 / Shape-File
UP Filter: Relevanz/Zuverl. Quellensicherheit: Mittel AP4.DE UP
SM_Data
FHG Filter 1: Relevanz/Zuverl. Quellensicherheit: Hoch AP4.DE FHG
BOS_NGO_Data
Datenfusion (technisch) AP4.DE FHG
RATED_Data
FE_Data
FHG Filter 2: Relevanz/Zuverl. Quellensicherheit: Niedrig AP4.DE FHG
Datenverarbeitungsschicht
JSON Def1
Shape-File
REST-Interface
Datenfusion (inhaltlich) AP4.DE UP
Sichtungstool AP4.DE CC
JSON Def1
Datenbank (z.B. Postgres/-gis) Rohdaten, Lagedaten, Konfigurationsdaten AP4.DE VT
RAW_Data
JSON Def1
JSON Def2
Prognose-Modul AP4.DE UP
START_PROGNOSIS
PROGNOSIS_DONE
Rückkanal / Kontrollfluss AP4DE CC VT
Datenservice AP4.DE VT
Geoserver AP4.DE FHG
Datenserver, Konfigurationsserver
JSON Def1
WMS/WFS
REST-Interface
Echzeitlagebild-Backend AP4DE CC VT
Commander AP4.DE CC
Echtzeitlagebild-Demonstrator AP4.DE FHG,CC, VT
ARIGON® PLUS AP4.DE VT
Output-Schicht
JSON/GeoJSON Metadefinition
Ort
Lokale Koo.
Homogenisiert
Zeit
Typ
Event
Prognose
Relevanz
Intern
Global
Quelle
Daten
Rohdaten
Metadaten
Anzahl
JSON Definition 1
Ort
Zeit
Typ
Relevanz
Quelle
Daten
Anzahl
JSON Definition 2
Ort
Zeit
Typ
Anzahl
Shape-File
Indexfile
Attributefile
Vectorfile
weitere

Figure 1.1: The HUMAN+ software architecture. The application described in this thesis is located in the module “Fernerkundung” in the “Input-Schicht”.

The aim of the module being a component of the Fernerkundung building block in the HUMAN+-architecture (Figure 1.1) is described in the “Verbundbeschreibung” of the project as following:

> “[. . .] to fix the frequently occurring information deficit with regard to expected persons [. . .] and to enable an improved planning of required resources. [Therefore], data of multisensory terrestrial and airborne sensors are analyzed to deliver qualified contributions to a real time situational awareness map based on terrestrial and airborne sensors as well as derived forecasts to form a basis for current deployment and resource planning[9]”

The first step in developing the module was to translate this request into an idea, how satellite imagery could be used to detect migration movements. The fundamental problem is that humans are not visible on satellite imagery. This is caused on the one hand by a spatial resolution of satellite imagery which is too low to detect people or the other way around: the satellites are too far away from the earth’s surface. The best commercially available satellites offer a spatial resolution up to 30cm per pixel. On the other hand, satellite imagery has a fluctuating image quality. So even if the best satellites could detect humans, it is not said that those images are available. Weather conditions do also play a role in the domain. It is therefore not possible to just count people who walk on the

[9]Translated from German

ground. Concerning the methodology we observed machine learning's huge success in image recognition and computer vision (LeCun et al., 2015). Therefore our working hypothesis is that machine learning should be usable to analyze satellite imagery too. This bachelor thesis is about proving this.

After getting to know the state-of-the-art in machine learning on satellite data by reading systematic reviews on the issue (Cao et al., 2017; Zhang et al., 2016; X. X. Zhu et al., 2017; Cheng et al., 2017), I came to the conclusion, that the best idea would be to do some kind of object detection on satellite imagery. We found the research done by Z_GIS[10], the interdisciplinary Centre of Competence for Geoinformatics at the University of Salzburg. They conducted research on “Dwelling Detection” (Spröhnle et al., 2014). This is an approach to analyze satellite images of refugee camps by finding and counting all dwellings i.e. tents in the camp, and from that estimating the number of people living in the camp. The detection of the dwellings is traditionally done by using classic object-based image analysis (OBIA) workflows (Tiede et al., 2017). Recently in 2018, first approaches (Ghorbanzadeh et al., 2018) of using machine learning for object detection were made. The most promising approach (Quinn et al., 2018) used Mask R-CNNs (He et al., 2017) for dwelling detection.

I decided to develop a dwelling detection workflow for the HUMAN+ remote sensing module. The technical backbone is a Faster R-CNN (Ren et al., 2017), which is based on the same platform as the Mask R-CNN (He et al., 2017) used in (Quinn et al., 2018).

[10]http://zgis.at/

2. Methodology

2.1. Theoretical Background

2.1.1. Geographic Information System

Satellite imagery, as well as remote sensing data in general, is geographic data. This means, that the data is related to the earth and that it can be associated to a certain coordinate on the earth. Technically, that is done by saving some kind of coordinate in the metadata of a dataset. To handle these kinds of data geographic information systems (GIS) are used. I will give a short overview over the main features and contents of a GIS, following (Sutton et al., 2009):

A GIS is a graphical application for working with geographic data. Geographical data consists of two different kinds of information: On the one hand you have measurements of the earth which were taken at certain places on the earth, e.g. satellite imagery. On the other hand, you have map data. This data consists of geometrical objects describing the surface of the earth. Three basic forms exist: Point data, vector data and polygons. Point data describes a single geographic entity – e.g. a tree, a house, or a volcano. Vector data describe lines on the surface of the earth, for example roads or borders. With polygons, things on earth that stretch over a 2-dimensional space, like forests and cities, are described. These geometrical objects are called “geographic features”.

Geographic features in GIS store both above-mentioned kinds of information. They have a description of their geometry in a coordinate reference system (CRS) and they have a so-called attribute table. This table describes what the geometrical object is representing, which can be measurements taken on the earth or basic information like the examples given above.

Anyway, satellite data is not saved in vector objects.

The other important type of data in GIS is so called raster data. This is data where measured intensities are stored on a pixel-based raster. This data can be taken from all kinds of sensors, be it optical, acoustic or radar sensors. The data is georeferenced by specifying a CRS and an affine transformation matrix that maps pixel locations in coordinates to spatial positions (*Rasterio*, 2019). With this, raster data can be mapped into GIS directly.

The last crucial element to make GIS work are the already mentioned Coordinate Reference Systems. CRS try to map the three-dimensional earth on a two-dimensional plane.

This is a very complex operation because mathematically, an accurate projection of a three-dimensional sphere on a two-dimensional plane is not possible. To get a representation as accurate as possible, thousands of projections, which can be sorted in three main categories, have been developed. The EPSG Geodetic Parameter Dataset, "a collection of definitions of coordinate reference systems and coordinate transformations which may be global, regional, national or local in application" (*EPSG*, 2019) lists 17574 entries at the time of writing this thesis (*EPSG Geodetic Parameter Registry*, 2019). Those three categories are cylindrical projections, conic projections and planar projections (see Figure 2.1). It is crucial in every GIS-Project to have a coherent CRS.

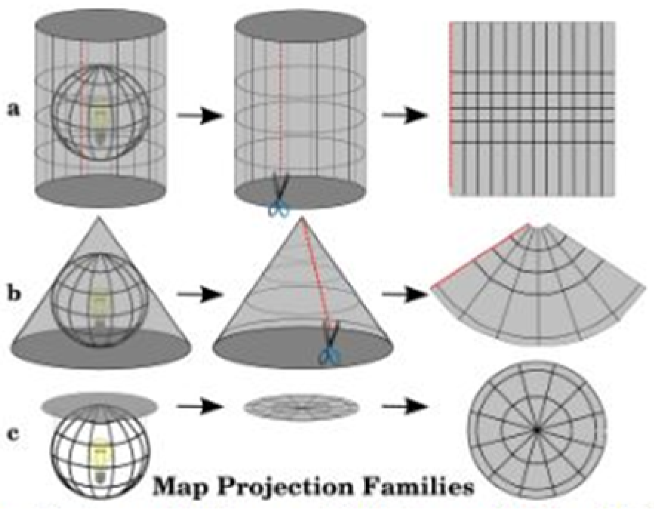


Figure 2.1: An overview over the different types of projections used in Coordinate Reference Systems (Sutton et al., 2009). Those projections are a) cylindrical projections, b) conical projections and c) planar projections.

To put it all together different formats for storing geodata were developed. The most used format for vector data is shapefile[1], originally developed by ESRI Inc. Shapefiles are storing the geometrical information in a main file and the attribute table in a separate database, namely dBase, file. Those two are connected by a separated index file. Raster data is usually stored in GeoTiffs, a special kind of Tiff-images. The image format .tiff was used because of its lossless storing of raster data. The georeferencing – coordinates, projections, CRS – are stored directly in the images metadata.

2.1.2. Remote Sensing Data

Remote Sensing is, in its broadest definition, "the gathering of information at a distance". From a technical perspective this means, that "Remote sensing is the practice of deriving information about the Earth's land and water surface using images acquired from an

[1]https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

overhead perspective, using electro-magnetic radiation in one or more regions of the electromagnetic spectrum, reflected or emitted from the Earth's surface" (Campbell & Wynne, 2011).

The basic workflow of Remote Sensing is described in Figure 2.2: The physical object is the thing that information should be gathered from. This information in raw form is acquired using some kind of sensor outputting sensor data. From that, meaningful information must be extracted. These can be used for different kinds of applications. Classical applications are Land Use, Geology, Hydrology, Vegetation and Soil.

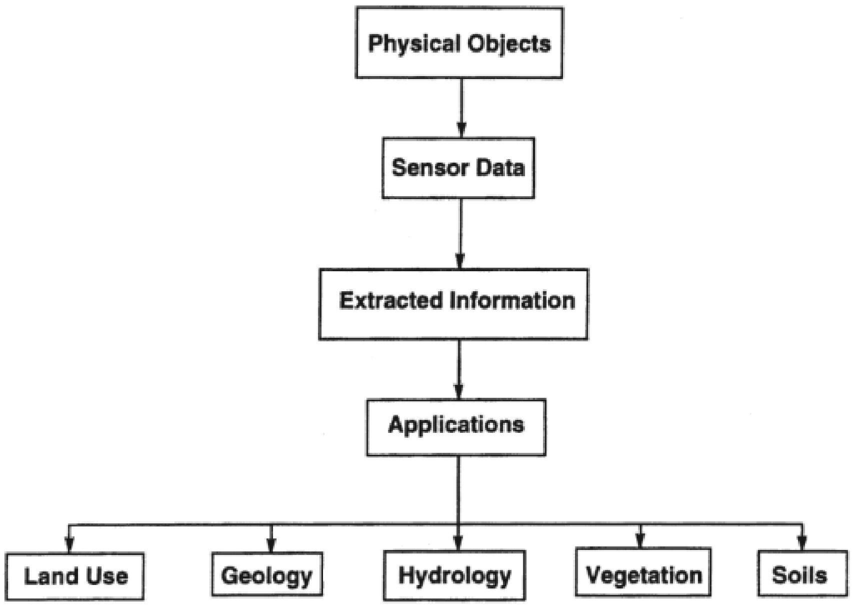


Figure 2.2: A generalized Remote Sensing Workflow (Campbell & Wynne, 2011). Physical objects are measured by sensors producing Sensor Data. From this data information is extracted. This information can be used in different types of applications. These applications can be located in thematic areas like Land Use, Geology, Hydrology, Vegetation and Soil.

Machine Learning on Remote Sensing

In the last years, deep machine learning has been a huge trend and was successful in many different domains. It has been making

> "major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years. [...] In addition to beating

> records in image recognition and speech recognition, it has beaten other machine learning techniques at predicting the activity of potential molecules, analyzing particle accelerator data, reconstructing brain circuits, and predicting the effects of mutations in non-coding DNA on gene expression and disease. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding" (LeCun et al., 2015).

Naturally, deep machine learning has also been used for processing Remote Sensing Data. In recent years, deep learning has been used for as different applications as hyperspectral image analysis, interpretation of synthetic aperture radar (SAR) images, interpretation of high-resolution satellite images, multimodal data fusion of Remote Sensing data with, for example, ground data, and 3D-reconstructions of the earth (X. X. Zhu et al., 2017). All these implementations use different types of sensors, need to extract different kind of information and are used for a huge variety of applications. This variety of tasks leads to a use of a wide variety of machine learning techniques.

Before going into the details of machine learning in Remote Sensing, a general framework for machine learning on remote sensing data (see Figure 2.3) is introduced in (Zhang et al., 2016). Each machine learning algorithm can be narrowed down to three main components: Prepared input data, the core deep network and the expected output data. The input-output pairs are hereby defined by the application. When defined, the core network constructs the “intrinsic and natural relationship” of the input-output pair. This network is usually a “deep architecture composed of multiple levels of nonlinear operations” and can use supervised training algorithms when ground truth data, i.e. data provided by direct observation and evaluation as opposed to data provided solely by inference, exists or unsupervised training algorithms when there is no ground truth. When the network is trained, it can be “employed to predict the expected output data of a given test sample” (Zhang et al., 2016).

Machine Learning on high resolution satellite images

An important field in Remote Sensing is the work with satellite imagery which can be used in a wide range of applications such as “natural hazards detection, LULC[2] determination, geospatial object detection, geographic image retrieval, vegetation mapping, environment monitoring and urban planning”. Due to the increase in spatial resolution of satellite images from the 1970s until now, new and better applications could be developed and the ways of analyzing the data changed. While “in the early 1970s, the spatial resolution of satellite images [...] was low and hence, the pixel sizes were typically coarser than, or at best, similar in size to the objects of interest, [today] the spatial resolution is gradually finer than the typical object of interest and the objects are generally composed of many pixels”. Therefore, fine-grained meaningful information can be derived from satellite imagery. In this millennium, the traditional methods for doing so were Object Based Image

[2]Land Use and Land Cover

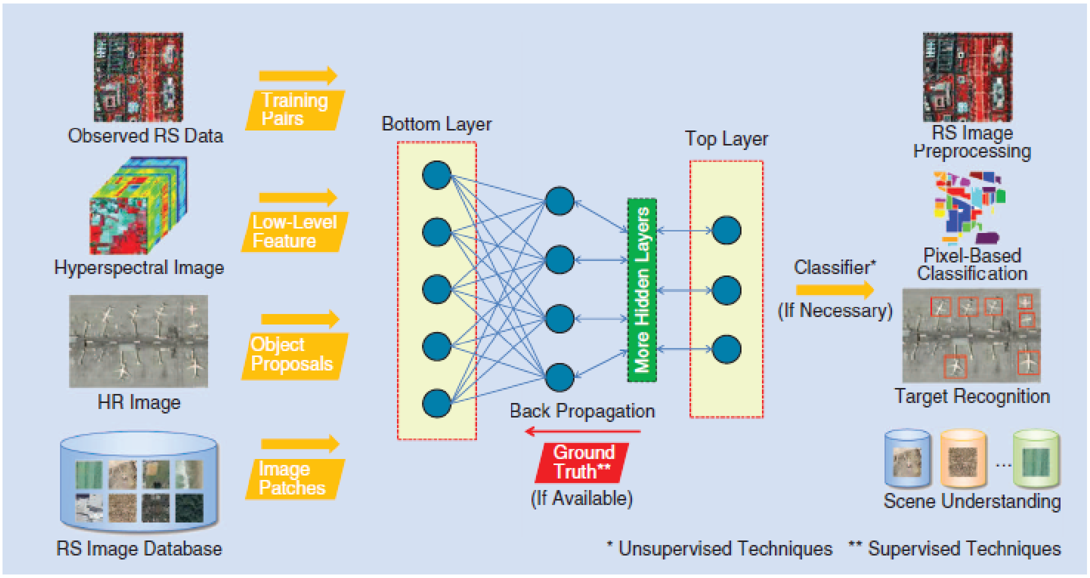


Figure 2.3: A Deep Learning Framework for Remote Sensing introduced by (Zhang et al., 2016). Different kinds of input data are inserted into a deep network. The features extracted from the top layer of the deep network are interpreted by a classifier, outputting data which is in relation to the input data. In supervised learning, this relation is learned via backpropagation of ground truth Data.

Analysis (OBIA) methods. Those aimed to “[produce] a set of non-overlapping segments (or polygons), that is, the partitioning of a scene image into meaningful geographically based objects or superpixels that share relatively homogeneous spectral, color, or texture information” (Cheng et al., 2017).

In the last years, Convolutional Neural Networks (CNNs), trained in a supervised fashion, have emerged as the most powerful method for image classification and object detection on images and started to supersede OBIA methods as the go-to-method. The greatest bottleneck for implementing CNNs is the ground truth data needed for training these networks. On the one hand, a huge amount of high-quality input data must be available; on the other hand, this input data has to be annotated by a human, which can take a lot of time[3]. For Land Use classification, i.e. satellite imagery classification, some open datasets exist (Cheng et al., 2017), for object detection; no datasets specialized on satellite imagery exist. Therefore, it is possible, that, when developing new machine learning applications, you must create your own dataset. (Cheng et al., 2017) define three requirements for a high-quality dataset, which must be thought of when creating a dataset:

– A large number of images/objects per class and a large number of images/objects in the dataset.

[3]The creation of the object detection dataset MS COCO (Lin et al., 2014) utilized over 70.000 worker hours.

- For each class, big variation in image/object translation, spatial resolution, viewpoint, object pose, illumination, background and occlusion.
- A high within class diversity and between class similarity.

Dwelling Detection

I concluded that our working hypothesis, that machine learning is applicable with good results on remote sensing data, could be right after reviewing the state of research on the topic. Still, it was not clear how to concretely transform the working hypothesis into a specific application. A crucial question that needs to be answered to concretize what to develop is what kind of physical objects are going to be analyzed. Problematic on satellite imagery is, that even if we have a fine-grained spatial resolution, humans are not or just rarely visible on satellite images. The best spatial resolution in commercially available satellite imagery today is around 30cm per pixel[4] and therefore a human is lost in the averaging of the input intensities over one pixel.

In papers of the OBIA_Lab at the Z_GIS department of the University of Salzburg[5] a feasible approach can be found. These papers cover research on a technique called "dwelling detection". Dwelling detection aims to analyze the number of people living in a refugee camp using (very) high resolution satellite imagery. In one of the first automated approaches on this kind of analysis, refugee camps were divided into separate areas. On these areas, a statistical linear regression analysis was carried out to find the relationship between area and population. While getting promising results, it is argued that "a more complicated, but potentially more accurate, method of estimating population via remote sensing is through dwelling unit counts and corresponding in situ information on the number of persons occupying each dwelling unit" (Bjorgo, 2000).

This approach has been followed by numerous predecessors. (Giada et al., 2003) were one of the first to adopt a hierarchically segmentation approach for dwelling detection on a camp in Tanzania. (Spröhnle et al., 2014) carried out a broad study comparing semi-automated object based approaches with manual image analysis of a camp in Somalia. They found that the semi-automated object based approach lead to more consistent results than the manual approach while only taking a fraction of the time. (Tiede et al., 2017) introduced an advanced stratified template matching OBIA algorithm with input data from 10 different camp sites taken by 16 satellites. (Aravena Pelizari et al., 2018) developed an advanced OBIA algorithm using multi-sensor feature fusion on data from the Al Zaatari camp, a refugee camp in Jordan.

In 2018, first approaches solving the problem of dwelling detection and extraction with neural networks have been made. At the department of Geoinformatics at the University of Salzburg, first experiments on dwelling extraction on one camp, the Minawao refugee

[4]http://www.ball.com/aerospace/programs/worldview-3
[5]http://obia.zgis.at/

camp in northern Cameroon have been made. (Ghorbanzadeh et al., 2018). They built a CNN and train it on one camp. Further they compare it with an object-based approach using handcrafted features. They note that both approaches detect dwellings with similar accuracies, having F1 scores between 85.2% and 96.3% on three classes. Concerning the CNN, they suspect that its transferability to images from the same or highly similar camps is good, but the transferability to different situations and camp-structures is low. Solving this problem would require a large set of training data, which is a problem concerning refugee camps. Going much further, (Quinn et al., 2018) use a Mask R-CNN, an architecture capable of semantic segmentation building up on Faster R-CNN, to build a more general dwelling detector. They train their model on thirteen camps achieving a mean Average Precision (mAP) over 0.7 for all camps but two when analyzing images of camps not seen by the network while training. Their classifier is not putting out bounding boxes, but segmentations of each dwelling. Given the information which spatial resolution one pixel has, the roof area of one dwelling can be calculated. (Quinn et al., 2018) compare the performance of the net on a satellite image that the net has never seen while training and with an image the net has at least seen some parts of while training. They conclude that training a net on parts of the image can have a good effect when the dwellings in the camp inhabit unique features compared to the other camps the net was trained with. Still, the base that has not seen the image while training is able to yield acceptable results. Further they propose an adaptive, iterative learning strategy to improve the detector.

Refugee Camps

Crucial for a working dwelling detection workflow, especially one relying on machine learning, is to take as many kinds of refugee camps in consideration. Regarding the organization of the camps, the situation they were built in as well as the amount of people in a camp; characteristics of different camps can vary tremendously.

One kind of camps in Europe have been closed refugee camps and detention centers. (Katz, 2016). Similar camps exist outside Europe in the form of processing centers in transit countries to Europe. “Many of these camps are run by private companies [. . .] and located in isolated places, remote from other built environments and from urban centers” (Katz, 2016).

Different from that are huge camps in crisis regions, run by the UNHCR[6]. An example for one of these camps is Zaatari, Jordan. The tents used in these camps are often standardized UNHCR tents, with the classical version sheltering families of five members. The tent structure in those camps depends on how they were built. When built following official planning guidelines, “rigid spatial order in the form of a grid and a clear layout of functions” emerge while a more spontaneously built camp, relying on “squatting practices initiated by the refugee’s themselves” leads to a less controlled camp-structure (see Figure 2.4).

[6]https://www.unhcr.org/

The Zaatari-camp inhabits both kind of structures, also because strict guidelines couldn't be enforced due to a rapid growth of inhabitants[7]. Overall, Zaatari became synonymous with planning failure (A. Dalal et al., 2018).

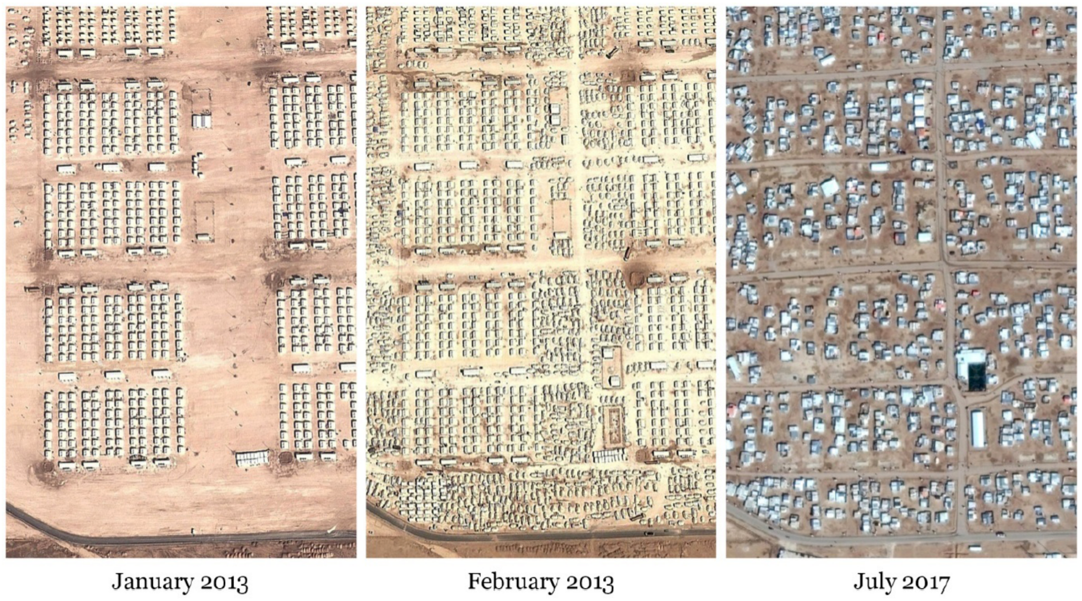


Figure 2.4: The developement of the spatial structure of the Zaatari-Camp (A. Dalal et al., 2018). In January 2013 it was tried to organize the logistics of the camp after guidelines. Following the guidelines became difficult in February 2013 when the number of inhabitants in the camp increased drastically. In July 2017, self-organized structures relying on squatting practices of the inhabitants had formed.

Another kind of camp are so-called makeshift camps. These are created by "forced migrants, [. . .] insisting on continuing the journey to their preferred destination and refusing to stay in camps which are opened by the authorities, [. . .] who are both their residents and their constructors." Location wise, "these makeshift camps are often erected not far from or within existing built environments, nestled in urban centers or in the outskirts of cities" (Katz, 2016). Due to their nature, it cannot be predicted, where these camps are created:

> "The creation of these built spaces seems to be completely arbitrary, since they are constituted in unexpected times and places in relation to various social, economic and political conditions. But where there is an enforced restriction of movement, camps will form" (Katz, 2016).

Extreme forms of makeshift camps are the so called jungles. They often form in front of transit stations at borders, which are the only way to cross otherwise closed borders. In early 2016, two jungles formed at the Klebia-Tompa and Horgos-Röszke transit stations,

[7]The number of inhabitants rose from 50.000 people in January 2013 to 200.000 in May of the same year.

both located at the Serbian-Hungarian border. There “were about 190 people in front of the Kelebia-Tompa transit zone.” These jungles are characterized by appalling living conditions: “There was no sanitation other than one water pipe, and inhabitants told us that insufficient food and non-food items were distributed” (Beznec et al., 2016).

2.1.3. Deep Learning

As mentioned above, deep learning has been widely successful in the last years, marking a shift in the fields of pattern recognition and deep learning. This fundamental change of methods is described by (LeCun et al., 2015) as following:

> “For decades, constructing a pattern recognition or machine learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data [...] into a suitable internal representation of feature vector from which the learning subsystem [...] could detect or classify patterns in the input. [...] Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level [...] into a representation at a higher, slightly more abstract level.”

To get an overview how these achievements are implemented technically I will shortly discuss the most important aspects and techniques of deep learning and especially of Convolutional Neural Networks, following (LeCun et al., 2015):

The basic principle of successful deep learning is combining a neural network (a network of the above mentioned “simple but non-linear modules”, called neurons) with a supervised learning approach. Supervised learning is characterized by training the network with a so called ground truth, which is a classified or labeled dataset, the so called trainset, where the classes in the dataset are those the network is supposed to classify after training.

(LeCun et al., 2015) describe the training process as following:

> “During training, the machine is shown an image and produces an output in the form of a vector of scores, one for each category. [...] We compute an objective function that measures the error (or distance) between the output scores and the desired pattern of scores. The machine then modifies its internal adjustable parameters to reduce this error. These adjustable parameters, often called weights, [...] define the input-output function of the machine. To properly adjust the weight vector, the learning algorithm computes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight were increased by a tiny amount. The weight vector is then adjusted in the opposite direction.”

To compute an output of the network which can be evaluated by the ground truth, the input data must be sent through the layers of the neural network. To conduct this so called forward-pass, the data is passing each layer consecutively. In one layer, the set of neurons compute a weighted sum in one way or another with the output of the previous layer as input and pass the result through a non-linear function to the next layer.

The algorithm for adjusting the internal weights, the backward pass, is called Backpropagation. The basic concept of backpropagation is applying the chain rule of derivatives backwards through the hidden layers of the network. The algorithm begins at the classification layer of the network, calculating the error between the classification and the output which is then used to calculate the gradient for the last hidden layer in the network. This gradient is then used to calculate the next gradient in the hidden layer before the last one. This process is repeated until the input layer is reached.

After training a network in a supervised way, it is commonly tested on a second labeled dataset, the test-set. When testing the network, the forward-pass, meaning the training procedure to the point where the error between output and ground truth is computed, is conducted while the backward-pass is cancelled. Instead, the error is stored for each input-data element and averaged after testing the whole test-set. The obtained average error is a measurement for how good the network is generalizing on data it hasn't seen before.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have been widely successful in the last years and have "recently been widely adopted by the computer-vision community" (LeCun et al., 2015).

CNNs are built on four key ideas: local connections, shared weights, pooling and the use of many layers. Further, they were inspired by "the classic notion of simple cells and complex cells in visual neuroscience" (LeCun et al., 2015; Hubel & Wiesel, 1962). A CNN typically consists of two separate sub-networks: The first network which is computing a feature vector out of the input data and the second network classifying the feature vector. Both networks can be trained with backpropagation.

The first network consists typically of convolutional and pooling layers. A convolutional layer consists of several feature maps, where the input data from the layer before is stored and several filter banks. Those filter banks are sliding window filters, going over the feature maps of the convolutional layer. The peculiarity of these filter banks is, that the weights in the filter are learned by backpropagation. Each of these filter banks is then generating a new feature map in the next layer after the filter output is passed through a non-linearity function. The most common non-linearity today is a rectified linear unit (ReLU). While convolutional layers ought to detect features in images, pooling layers try to "merge semantically similar features into one by reducing the extent of the feature map". The most popular approach, max pooling, takes the maximum from a patch and writes it in the output. The patch is thereby sliding over a feature map with a stride bigger than one, which "[reduces] the dimension of the representation and [creates] an invariance to small shifts and distortions." (LeCun et al., 2015).

The second network is a classifier, classifying the results of the last layer of the first network. How it is built up depends on the task the CNN must solve. In the classic application of image classification, the second part of the network often consists of a few fully connected layers with a softmax-classifier analyzing the output.

The features detected by each layer in the first part of the CNN get more abstract and high-level the deeper they are in the network. The main reason why CNNs are so successful is that natural signals are composed in a similar, hierarchical way:

> "In images, local combinations of edges form motifs, motifs assemble into parts, and parts form objects. Similar hierarchies exist in speech and text from sounds to phones, phonemes, syllables, words and sentences" (LeCun et al., 2015).

CNNs popularity increased drastically after the introduction of AlexNet/SuperVision (Krizhevsky et al., 2012) which won the ImageNet challenge (Russakovsky et al., 2015) by a huge margin compared to its competitors using handcrafted features.

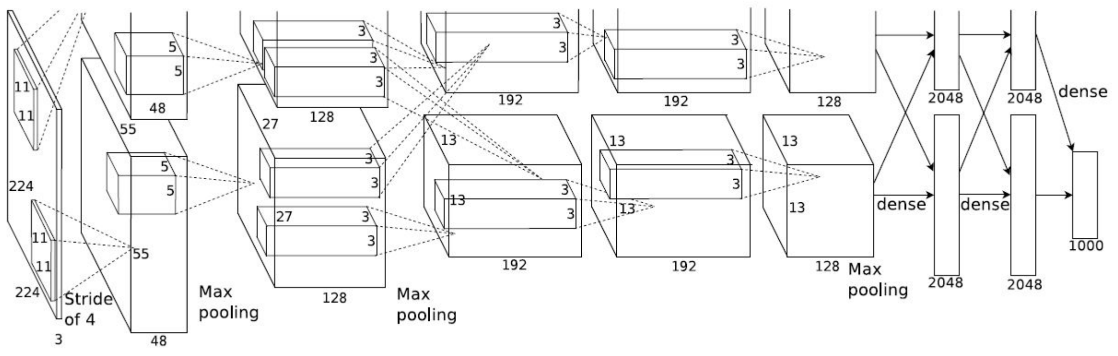


Figure 2.5: Architecture of the AlexNet, which won the ImageNet challenge in 2012. A sequence of alternating convolution- and pooling-layer with fully connected layers at the end of the network can be observed (Krizhevsky et al., 2012).

Regional Convolutional Neural Networks

Object Detection on images is a long and broadly researched topic. It consists of two tasks: Finding an object in an image and classifying the found object. The output of the task is an image with bounding boxes framing objects in the image and class scores determining the object class of the object in the bounding box.

The image classification task was revolutionized when AlexNet was introduced in 2012 (Krizhevsky et al., 2012) and superseded hand-crafted feature detectors as the go-to-technique for image classification. The same shift can be seen in object detection: In 2010 to 2012, the winning entries of the PASCAL VOC challenge, the benchmark challenge in the object detection community (Everingham et al., 2010), used handcrafted features based on the techniques SIFT (Lowe, 2004) and HOG (N. Dalal & Triggs, 2005). In 2014 R-CNN was introduced, which uses a Convolutional Neural Network and supervised training for object detection and improved the results of the PASCAL VOC 2012 challenge by more than 30% (Girshick et al., 2014).

(Girshick et al., 2014) define two fundamental problems that need to be solved to transfer CNNs for object detection: The localization of objects and a scarce amount of training data. Localization in object detection is traditionally done with a sliding window approach. This is problematic for a CNN because deeper convolutional layers have, due to pooling in-between convolutional layers, a large receptive field and stride on the original image, making localization inaccurate. R-CNNs solve that problem by analyzing pre-computed region proposals for objects on the image. The second problem, scarce training data, is solved by pre-training the net on a large, auxiliary dataset and then fine-tuning it on a smaller, domain-specific dataset.

The R-CNN in (Girshick et al., 2014) consists of three modules (see Figure 2.6): A category independent region proposal module, a large convolutional neural network and a set of class-specific linear SVMs (Hearst et al., 1998), classifying the output feature-vector of the CNN.

The first module utilizes a state-of-the-art algorithm for extracting category-independent region proposals. An R-CNN can use any algorithm that offers these proposals. In their paper, (Girshick et al., 2014) use the Selective Search (Uijlings et al., 2013) algorithm. The second module is an implementation of the CNN introduced in (Krizhevsky et al., 2012). Each region is forward through the network separately. The network requires a fixed input size; therefore each region has to be warped to that size first. The output of the module is a 4096-dimensional feature vector. This feature vector is analyzed by a set SVM's, each trained for one class. The results for each class are processed with greedy non-maximum suppression.

The R-CNN is trained in two stages: The first stage is a supervised pre-training with a large dataset for an auxiliary task. The dataset only needs image-level annotations, meaning no bounding box labels. After initially training the network on an image dataset, the architecture has to be rebuild and a second stage, domain-specific fine-tuning, is carried out. To adapt the network, the classification output layer for the auxiliary network has to be replaced with an output layer fitting to the classes of the new dataset. Further, it has to be defined when an analyzed proposal is treated as a positive. (Girshick et al., 2014) treat all regions, which have an IoU[8] >= 0.5 with a ground truth box as a positive for the highest-scoring class in the bounding box. Further a learning rate has to be set and

[8]IoU: Intersection over Union

an stochastic gradient descent (SGD) mini-batch, containing positive class windows and background windows, has to be defined. In addition to the output feature vector, classified by the set of SVM's, a bounding box regressor is trained. Using a linear regression model, a new detection window is trained, refining the Selective Search object proposals.

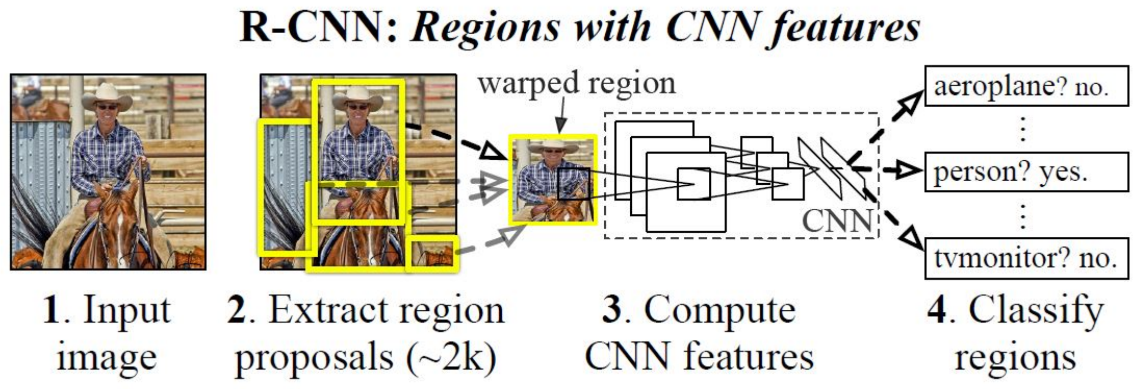


Figure 2.6: The architecture of an R-CNN. Region proposals are warped into the input size of the CNN. The CNN computes class securities for each proposal (Girshick et al., 2014).

Fast R-CNN

While the R-CNN was a huge leap forward for deep object detection, it has some major drawbacks in its architecture, described in (Girshick, 2015). The first drawback concerns the training algorithm which is multi-staged: First, the CNN is trained using log-loss, then the SVMs are fitted to the CNN and finally the bounding box regressors are learned. The second drawback is that the training is furthermore slow and expensive. All features extracted from object proposals, extracted from the initial image, are written to disk. This takes a lot of time and a lot of disk space. The third major drawback is that object detection is slow. The reason for that is that every object proposal has to go through the forward-pass of the network.

Fast R-CNN (Girshick, 2015), building on the architecture of the original R-CNN, is fixing these problems by adjusting R-CNNs architecture where it's necessary. This results in an architecture which has higher detection quality than its predecessor, has a single-stage training workflow which can update all network layers using a multi-task loss and requires no disk storage for feature caching.

The Fast R-CNN workflow consists of four steps, utilizing different modules inside the network architecture (see Figure 2.7). A Fast R-CNN takes an image and a set of region proposals as input. The first module of a Fast R-CNN is a CNN. The CNN takes the image to be analysed and calculates a feature vector as its output. In the second step, the object proposals are utilized. For each proposal, a special Region of Interest (RoI) pooling layer extracts a fixed length feature vector associated with the proposal from the input image from the CNNs feature vector. This object proposal specific feature vector is then fed in the third step into two fully connected layers, outputting a RoI feature vector. This RoI feature vector is fed in the fourth step into two separate output layers: A softmax probabilities estimator outputting class-probabilities over all trained classes K plus a background-class and a bounding box regressor outputting four offsets for each class for each bounding box resulting in 4*K values.

A Fast R-CNN is, like R-CNN, relying on a network pre-trained on a large, auxiliary dataset. To convert a pre-trained network into a Fast R-CNN, some adjustments have to be made: First, the last pooling layer of the pre-trained network is replaced with a RoI pooling layer which is compatible to the first fully connected layer. Second, the last fully connected layer has to be replaced by the two sibling layers described above. Third, the input has to be modified so that images and object proposals are accepted.

The RoI pooling layer is a special layer developed for Fast R-CNN. It converts the features insid any valid region of interest into a small feature map with the fixed spatial extends H x W. A RoI is defined as a rectangular window into a convolutional feature map. Pooling works by max-pooling a h x w RoI into a H x W grid consisting of h/H x w/W sub-windows. Pooling is done on every feature map separately.

The fine-tune training of the Fast R-CNN takes advantage of feature sharing through the different tasks during training. Both networks, the softmax classifier as well as the bounding box regressor are trained simultaneously using backpropagation. The backpropagation algorithm is adapted so that backpropagation through the RoI pooling layer is possible. The joint optimization of both outputs is achieved through a multi-task loss term. The loss term ads the log-loss for the softmax classifier and the L1 regression loss. The regression-loss term is only evaluated when the proposal is a positive sample and not a background sample.

Shared computation is also thought of when building a SGD mini-batch. Region proposals in a mini-batch are sampled from one or two images. Therefore, for training with all proposals, only one, respectively two forward passes for the input images have to be carried out. In one mini-batch, 75% of all proposals are negative background proposals (IoU with ground truth box < 0.5) and 25% are positive samples, representing an object.

When used with real data, one image and a list of object proposals are used as input of the network. The image is processed once through the first CNN until the RoI pooling layer is reached. For each region proposal, the RoI extracts the associated feature vector out of the input images feature vector. The extracted feature vector is then analysed by the sibling networks. The output of the Fast R-CNN is for every RoI one class probability distribution and a set of predicted bounding-box offsets relative to the proposals coordinates.

During experiments (Girshick, 2015) found out that Fast R-CNN achieves state-of-the-art mAP, is faster in training and testing than its predecessors and that fine-tuning the backbone CNN improves mAP. Further findings are that multi-task training improves performance, that deep CNNs directly learn scale invariant features from the data, that larger training sets slightly improve performance, that softmax classifiers yield better results than SVMs and that fewer but high quality object proposals improve the networks accuracy.

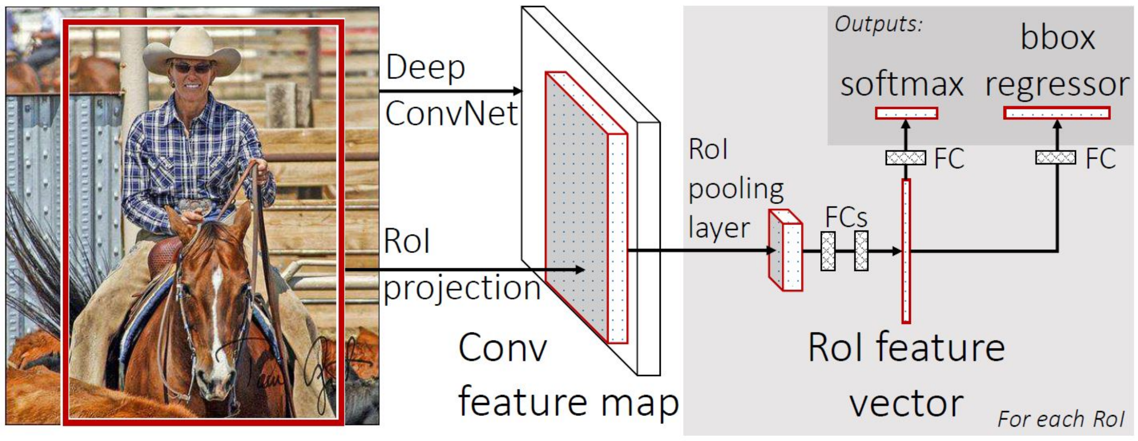


Figure 2.7: The architecture of a Fast R-CNN. The image is first processed by the CNN backbone of the network. Then region proposals are pooled into a classification network consisting of two fully connected layers using a RoI pooling layer. The fully connected layers are outputting class securities using a softmax classifier and bounding box offsets using a regression (Girshick, 2015).

Faster R-CNN

Fast R-CNN showed that CNNs for object detection are widely successful and have a fast detection time. Detection on one image using Fast R-CNN takes roughly two seconds. Interestingly, computing the convolutional features happens in near-real time while computing the object proposals using classic computer vision algorithms takes the biggest amount of computation time, making this step the bottleneck regarding detection performance. Faster R-CNN (Ren et al., 2017) solves that problem by introducing a Region Proposal Network (RPN).

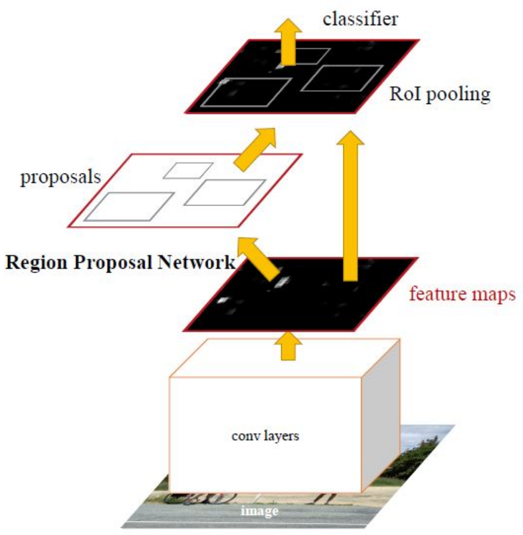
Faster R-CNN is a Fast R-CNN that obtains the required region proposals by the RPN (see Figure 2.8). The RPN is located between the feature vector outputted by Fast R-CNNs backbone CNN and Fast R-CNNs RoI pooling layer. That means that RPN shares convolutional layers, the backbone CNN, with the initial Fast R-CNN which enables nearly cost-free proposal computations. The feature maps used for region-based detectors can also be used for region-based proposals. RPN and Fast R-CNN form a unified network: Faster R-CNN. Looked at from another perspective, Faster R-CNN is a “Neural Network with attention” (Ren et al., 2017), with the RPN telling Fast R-CNN where to look.

A RPN is a fully convolutional network. That means that the output has the same dimension as the input. The input of the RPN is, when considering the whole network including the shared CNN backbone layers, the input image to be analyzed. When considering only the RPN-head stacked on the backbone CNN, the inputs are the feature maps produced by the last convolutional layer of the backbone. The RPN outputs a set of rectangular object proposals, each with an objectness score. Objectness is defined for a proposal as belonging to a set of object classes or belonging to the background, not encapsulating an object.

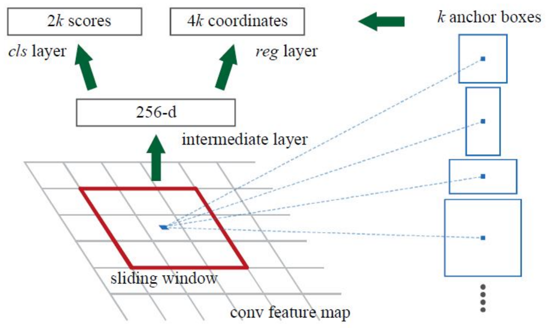
The RPN architecture consists of the backbone CNN and the RPN head. The backbone CNN can be any CNN pre-trained for image classification. The RPN head consists of a layer sliding a window over the last feature maps of the backbone CNN and two output layers, a box-regression layer *reg* and a box-classification layer *cls*. The sliding window layer calculates a set of k so called anchors on every location it visits. Anchors are rectangular boxes which are analyzed if they could be proposals. Typically, nine anchors are calculated. For calculating anchors, three anchor-sizes and three anchor-ratios which are all combined with each other are used. These anchors are supposed to cover all possible objects in an image. The resulting convolutional map has the size W x H and has nine anchor definitions on each location. Each anchor is then fed into the two sibling output layers. The *cls*-layer calculates if the anchor is an object and the *reg*-layer refines its coordinates. The anchors with the best objectness scores can then be used for the further Fast R-CNN workflow. Using anchors has two main benefits: Anchors are translation-invariant and anchors fit multiple scales of objects.

For training, anchors are labelled with a binary label – being an object or not - using the ground truth boxes of the input image. An anchor is labelled positive when it either has the highest IoU overlap with a ground truth-box compared to all other anchors or when it has an IoU overlap higher than 0.7 with any ground-truth box. Anchors are labelled as negative or background when the highest IoU overlap to a bounding box is lower than 0.3. All anchors not falling in those categories do not contribute to the training. The loss function for training is a multi-task loss, combining the classification and regression losses. Regression is performed for one anchor to the nearest ground truth bounding box. The RPN-head-weights are initialized with a zero-mean Gaussian distribution with standard deviation 0.01. For training the standard backpropagation and stochastic gradient descent algorithms are used. For one SGD mini-batch, 256 anchors are sampled from one image with a positive to negative ratio of 1:1 if possible.

For training the whole network, the RPN and the Fast R-CNN should be trained independently because both training procedures modify the network in different ways. Different training approaches can be used: When using alternating training the RPN is trained first. Then the Fast R-CNN is trained using the proposals generated by the RPN. This process can be iterated. Approximate joint training merges the RPN and the Fast R-CNN into one network during training. In the forward pass of the training, proposals are generated by the RPN which are then used by the Fast R-CNN. During backpropagation, the RPN loss and



(a) Faster R-CNN Architecture.



(b) RPN Architecture.

Figure 2.8: The architecture of a Faster R-CNN and the RPN head. The RPN is located between the last convolutional layer of the CNN and the Fast R-CNN head. In the RPN, an anchor is set for each location on the feature map of the last convolutional layer of the CNN. The RPN is outputting if the anchor is encapsulating an object as class security and bounding box offsets to the anchor (Ren et al., 2017).

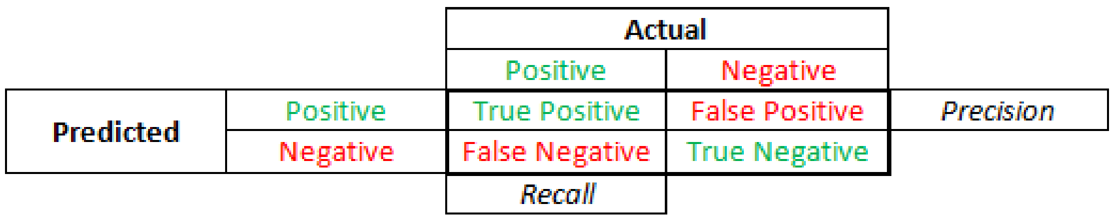
the Fast R-CNN loss are combined when training the shared layers. This training workflow produces results close to Alternating Training while reducing training time by about 25-50%. In (Ren et al., 2017) an alternating training process is used. First, the RPN is trained and after that the Fast R-CNN. Then, the RPN-specific layers are fine-tuned followed by fine-tuning the Fast R-CNN specific layers.

After experiments, (Ren et al., 2017) conclude that Faster R-CNN yields results as good as Fast R-CNN using precomputed proposals while being much faster than Fast R-CNN. Sharing convolutional layers between RPN and Fast R-CNN even improves the performance. A reason for that could be that fewer inconsistencies between training and testing proposals exist. Furthermore, the top-ranked RPN proposals can be seen as accurate. Another advantage is that using deeper networks for RPN and Fast R-CNN improves the performance.

Object Detection Evaluation Metrics

Convolutional Neural Networks are usually evaluated using three different ground truth datasets derived from an initial ground truth dataset: the training set, the validation set and the test set. The initial ground truth dataset is split into two sets: The training set is the dataset the CNN is trained on during training process. It is the bigger subset of the initial dataset. The validation set is a subset of the training set. It is used to test the performance of a CNN during the training process. It can therefore give information about the training history. The test set is the other subset of the initial data. It is used after the training for evaluating the obtained classifier. The training set should not be seen by the classifier before (Goodfellow et al., 2016, p.118).

For object detection, several metrics for evaluating an object detector and classifier exist. Object Detectors can yield for different kinds of results: True Positives TP, False Positives FP, True Negatives TN and False Negatives FN. These types of results can be visualized in a confusion matrix (Figure 2.9).



		Actual		
		Positive	Negative	
Predicted	Positive	True Positive	False Positive	*Precision*
	Negative	False Negative	True Negative	
		Recall		

Figure 2.9: Confusion Matrix showing all possible results of a classification. Recall is calculated using True Positives and False Negatives, Precision is measured using True and False Positives.

Out of these kinds of results, the three mostly used basic measurements can be calculated (Sokolova et al., 2006):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

Accuracy is therefore the ratio of all right detections to all detection. It is a commonly used metric but gives deceiving results when working on an imbalanced classification problem. It is therefore not recommendable for reliable evaluations of a classifier. Precision is a value indicating to the ability of a classifier to find only the relevant points in a dataset. The value lowers when classifying an element as positive when it's negative. Recall is giving a measurement of how many of the positive elements in the dataset are found by the classifier. It is increasing with every True Positive found. Precision and Recall are widely used in object detection.

When trying to optimize recall, it is a good strategy to try to classify as much data points as possible as positives. Because False Positives are not in the Recall-equation they don't harm the value. On the other hand, that strategy lowers the precision significantly. To optimize precision, it is a good strategy to try to classify as few as possible data points that are negative as positive. That can be done by setting a high threshold for classifying something as positive. This strategy strengthens precision but lowers recall. For having a better value to be optimized, the F1 score can be calculated. This is the harmonic mean of Precision and Recall.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.4}$$

Another way of interpreting precision and recall is he Precision-Recall-Curve. (Everingham et al., 2010) The PR-Curve is important for calculating the most important metric for object detection: Average Precision (AP). The basic assumption of the curve is that Recall increases with every true positive found and that for every Recall value a Precision value can be calculated. On the PR-Curve, the recall values between 0 and 1 are on the x-axis, the associated precision values are on the y-axis. It can be seen that the curve has the form of a zig-zag curve because every True Positive elevates the Precision while Recall can also be elevated by False Negatives. The Area under Curve of the Precision-Recall-Curve is the Average Precision (AP) of the classifier. Average Precision is an important value for the evaluation of object classifiers.

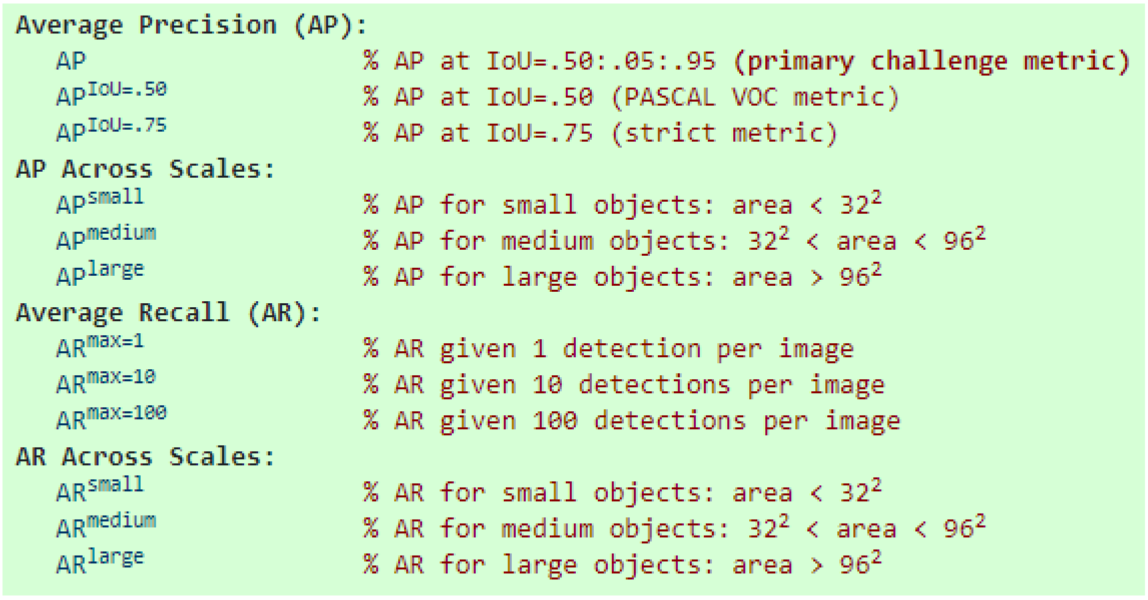
To calculate Average Precision, the zig-zag curve is often smoothed, taking the local maxima of the curve and replacing each precision value with the local maxima to the right. To measure the Area under Curve, the Average Precision, different approaches are used. The Pascal VOC challenge uses an interpolated AP, meaning that the Precision is measured at 11 recall values (0, 1) and calculating the mean under each point. For the COCO challenge a similar but stricter calculation is used by setting a 101-point interpolated AP

Average Precision is mostly not used directly for evaluating an object classifier but many AP values that are calculated on the classifier are summarized as mean Average Precision (mAP). (Everingham et al., 2010) The APs can be calculated for each class and for different IoU thresholds for the overlap of a finding with a ground truth box. The measurements defined for the COCO dataset can be found in Figure 2.10.

Machine Learning Frameworks

To utilize a Faster R-CNN I use the Facebook AI Research (FAIR)[9] software Detectron. Detectron is part of facebooks AI environment (Joulin & Paris, 2015). Detectron is built on the Deep Learning Framework Caffe2, which was merged with Facebooks Deep Learning Framework PyTorch on 02. March 2019 (Caffe2, 2018). Detectron was developed by, among others, Ross Girshick, Georgia Gkioxari, Piort Doll and Kaiming He who all have worked as author or co-author on various R-CNN papers mentioned above. It features the implementations presented in these papers.

[9]https://ai.facebook.com/research/



Average Precision (AP):

AP	% AP at IoU=.50:.05:.95 (primary challenge metric)
$AP^{IoU=.50}$	% AP at IoU=.50 (PASCAL VOC metric)
$AP^{IoU=.75}$	% AP at IoU=.75 (strict metric)

AP Across Scales:

AP^{small}	% AP for small objects: area < 32^2
AP^{medium}	% AP for medium objects: 32^2 < area < 96^2
AP^{large}	% AP for large objects: area > 96^2

Average Recall (AR):

$AR^{max=1}$	% AR given 1 detection per image
$AR^{max=10}$	% AR given 10 detections per image
$AR^{max=100}$	% AR given 100 detections per image

AR Across Scales:

AR^{small}	% AR for small objects: area < 32^2
AR^{medium}	% AR for medium objects: 32^2 < area < 96^2
AR^{large}	% AR for large objects: area > 96^2

Figure 2.10: Evaluation metrics used for the COCO dataset. AP and AR are calculated as mean Average Precision and mean Average Recall over different levels of IoU and over different Bounding Box sizes (Consortium, 2019)

2.2. Implementation

For the HUMAN+ project, the described Dwelling Detection module was built from scratch. The module consists of two parts: a submodule to create the training data called *human-plus_satprocessing* and a submodule to carry out training and analysis of VHR-satellite imagery using a Faster R-CNN called *humanplus_satAI*.

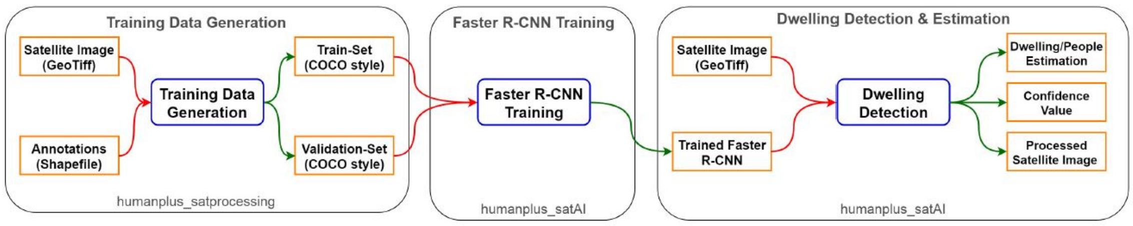


Figure 2.11: The architecture and the dataflow of the HUMAN+ SAT module, described in the following chapter. The training data generation is described in 2.2.1, Faster R-CNN training is described in 2.2.2 and Dwelling Detection is described in 2.2.3

2.2.1. Training Data Generation

One of the highest obstacles in every Machine and Deep Learning Project is acquiring the data for training and testing the network. In supervised learning, the burden of creating meaningful ground truth to the gathered data is added.

Getting the training data was the bottleneck for developing a successful application that was the problem that had to be fixed first. Faster R-CNN requires an image where ground truth bounding boxes are placed around the objects that are supposed to be trained. Both types of data are hard to find.

For researching prices for satellite images from Digital Globe/Maxar[10], resellers like European Space Imaging[11] were contacted. It showed that prices for satellite images are too high to buy them for the HUMAN+ application. Further the Copernicus Emergency Management Service[12] of the European Union was contacted through the Bundesamt für Bevölkerungsschutz und Katastrophenhilfe[13]. It was found out that Copernicus EMS can only be activated when a crisis takes place. Last the Center for Satellite Based Crisis Information (ZKI)[14] at the German Aerospace Center (DLR)[15] was contacted. They communicated that they can't give away satellite imagery due to license issues.

Therefore the solution for the training data problem followed two goals:

1. Using already existing labeled ground truth data and publicly available satellite images.
2. Automating the process of labeling and processing ground truth data as much as possible.

An approach for solving that problem can be found in (Quinn et al., 2018) who use a Mask R-CNN, which is the successor architecture of Faster R-CNN and works in most aspects like Faster R-CNN.

For training and testing, they use VHR-satellite data from camps, where freely available dwelling detection annotations, created by experts on the field, exist[16]. This data was supposed to be used in HUMAN+ too. The problem is that the annotations are free, but the underlying satellite images are not. Unfortunately, they are very expensive and couldn't be bought in the HUMAN+ project as discussed above.

Because the official satellite images couldn't be acquired, fitting satellite imagery was searched for using Google Earth Pro. Google Earth offers for locations of the earth a

[10]https://www.maxar.com/

[11]https://www.euspaceimaging.com/

[12]https://www.copernicus.eu/en/services/emergency

[13]https://www.bbk.bund.de/DE/AufgabenundAusstattung/Krisenmanagement/GMLZ/Copernicus/Copernicus_node.html

[14]https://www.dlr.de/eoc/en/desktopdefault.aspx/tabid-12797/gallery/30434

[15]https://www.dlr.de/dlr/en/desktopdefault.aspx/tabid-10002/

[16]http://www.unitar.org/unosat/maps, and https://data.humdata.org/organization/un-operational-satellite-appplications-programme-unosat

set of satellite images via a time-slider in the GUI. To fit the annotation data found to the satellite image they were made on, the attribute table of the annotations was checked for information about the satellite image they were taken on. The annotation data is in the form of Shapefiles, so the attribute table can be read using the open source GIS QGIS (QGIS Development Team, 2009). In it, the Sensor_ID, specifying which satellite took the image and the Sensor_Dat, specifying when the image was taken, could be found. For one of the annotation datasets, dwellings detected on the Juba camp in South Sudan, the fitting satellite image could be found.

After having found a satellite image with fitting annotations on which a workflow could be developed, the next problem occurred: Satellite imagery is not downloadable in a georeferenced format in Google Earth but only savable in standard image formats. To work around that problem, the image was saved in the highest possible resolution (4800 x 3600 pix), and georeferenced using QGIS.

In QGIS, the shapefile annotations of the Juba camp were loaded and a tool called Georeferencer (see Figure 2.12) was used to fit the satellite image taken from Google Earth on the annotation. In the Georeferencer, one selects points on the raster/the image and specifies a location they are equivalent to. The easiest way to select the location is to select a location on the georeferenced QGIS map canvas, the main window on the program when having a map or other objects giving orientation loaded in the canvas. The points set on the raster are then used as an input for a function projecting the image according to the specified locations into the coordinate reference system defined in the QGIS project. After precise georeferencing a georeferenced image of the Juba camp with fitting annotations of its dwellings was received.

Processing of Geodata

After acquiring a minimal working dataset for developing and training a Faster R-CNN based dwelling detector, the dataset had to be transformed in a form that the Faster R-CNN implementation can read and use. This workflow for the training data generation is implemented in the *humanplus_satprocessing* module and controlled in the *create_Training_ Data.py* script.

The first step is to map the annotations, whose coordinates are stored in geolocations, to their associated pixel locations on the satellite image. The satellite image contains an affirmative translation matrix which calculates the geolocation for one of its pixels, defined in the target coordinate reference system also stored in the satellite images metadata. To map the annotations to their associated pixel locations, the inverse of the affirmative matrix, stored in the satellite image, is calculated and each location is multiplied with the inverse matrix. As a kind of visual debugger, each annotation is mapped on the image by drawing a point on the pixel the annotation points at and the processed image is saved in an image file. The pixel locations of the annotations show to be fitting.

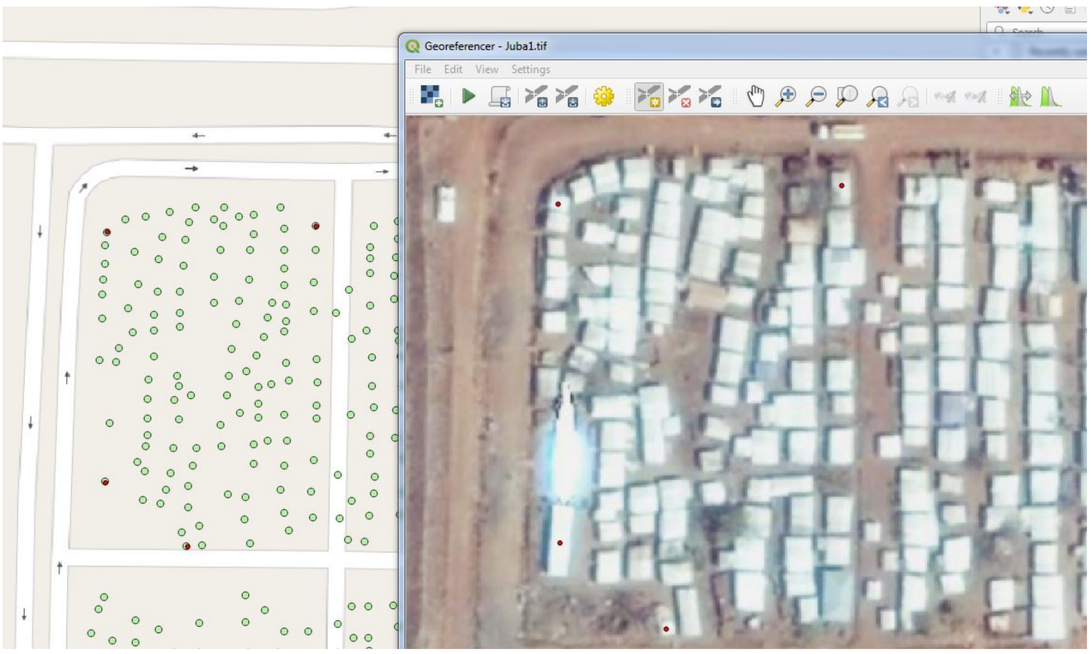


Figure 2.12: The QGIS Georeferencer Tool. The satellite image of the Juba-Camp is loaded in the Georeferencer opened in the window. In the background, the georeferenced QGIS canvas can be seen. In it, the Open Street Map and the UNHCR-annotations for the Juba-camp are loaded. The red points connect the annotations with the raster.

Satellite images often have a very huge size while CNNs work best with small input image sizes. When working with larger images, a huge amount of operations and data must be handled during training, making the training process really slow. To address that problem, the satellite image is cut in several tiles. As tile size 300 x 300 pixels is chosen, assuming the same tile size as (Quinn et al., 2018).

The image is then cut in tiles and the annotations are mapped on the tile they are onto. To cut the image into tiles, a padded image whose height and width can be divided by 300 is created and the original image is written into the padded one. After that a 300x300 slice is cut out of the padded image and saved as a tile in an image file. To map the annotations on the right tiles, two things have to be done: First, it has to be figured out, on which tile the current annotation is situated on. That is done by dividing the x- and y-location of the tile on the original image with 300 and rounding off the result. This gives the tile number in x- and y-direction starting with tile (0,0) in the upper left corner. To find the exact location on the found tile, the x- and y-value on the original image is subtracted by 300 multiplied with the found tile number in its respective direction. For every newly created tile, the corresponding annotations are saved in a separate file. In a last step, the tiles whose corresponding annotation file is empty, meaning that there are no annotations on the tile, are deleted to speed up the following steps.

Finding Bounding Boxes

After loading the annotations, cutting the input satellite image into tiles and adjusting the annotations to the tiles, the annotations must be fitted for a Faster R-CNN. A Faster R-CNN is trained on bounding boxes. Therefore, the point-annotations have to be transformed to bounding boxes encapsulating the annotated dwellings.

The simplest solution would be to just draw a square around each annotation-point. While this could work, it would be very inaccurate. Dwellings are often rectangular, there are different types of tents which have different sizes and one tent can have a different size than the same tent on another image photographed from a different height or with a different sensor. Therefore a region-fill algorithm is used to find the extent of each tent, assuming that tents are mostly uniform in its appearance and that they are distinguishable from the background of the satellite image (see Figure 2.13).

The algorithm initially calculates the mean value of a region in the image. This region is defined by the pixel an annotation is pointing at and its 4-neighbourhoud. All pixels in the region are written in a priority queue called *regionPoints*. Each point in *regionPoints* is then loaded in the Seed Growth algorithm. There, each pixel of the points 4-neighbourhood is analyzed. First it is analyzed if the point was already visited. Therefore the visited array, an array of the size of the input image where it is marked if the pixel was already assigned to a region, is checked. If that is not the case, the Euclidian Distance (2.5) of a pixel to the mean of the region is checked. If this distance is below a threshold, the pixel is marked as belonging to the region in the *visited* array and pushed in the *regionPoints* queue, so that its neighbors can be checked for membership in the region too. The threshold is one of four parameters to control the training data generation process.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.5}$$

After that, the found regions are processed so that they fit meaningful bounding boxes which can be used as training data.

The first step is to create Bounding Boxes, using the function *createBoundingBox(visited, nRegions, offset)*. The function is looking at every region found except the background-region with the label 0 in the *visited* array. The minimal and the maximal x- and y-values are found, and offset is added to them. The smallest x- and y-value form the coordinate for the top-left-coordinate of the bounding box, the biggest x- and y-value form the coordinate for the bottom-right-coordinate. The offset is the second of four parameters to control the training data generation process. There are two reasons for adding it: On the one hand it adds some context to a bounding box. The idea is that it will give the Faster R-CNN more information about the features at the edges of tents. On the other hand, it clears up issues where the seed growing algorithm is not accurate enough and just finds parts

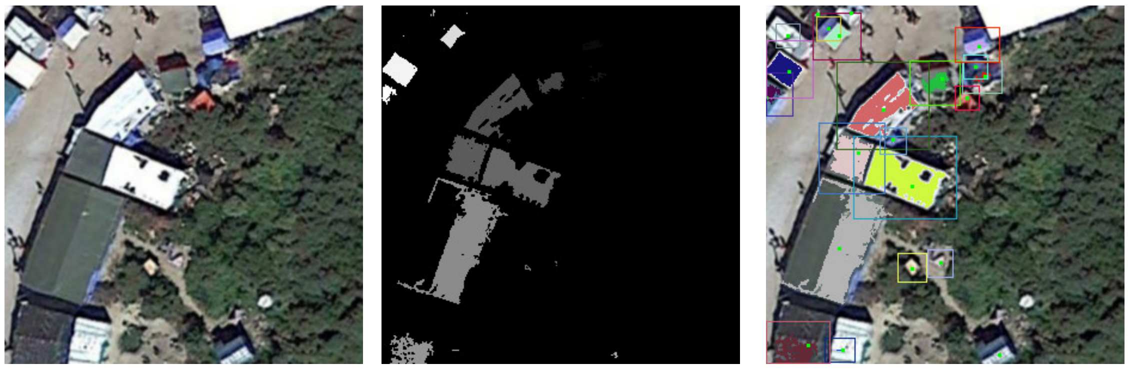

(a) Initial Tile (b) Found Regions (c) Processed Tile

Figure 2.13: Bounding Box Workflow for a Tile from Calais Camp. On a), the original tile can be seen. b) shows the found regions. On c), the segmented regions are colored in random colors. The bounding boxes are drawn around the regions. The green point shows the initial annotation.

of the tent. Adding an offset can help fitting a box that includes the whole dwelling. The found bounding boxes are appended to the list *boundingBoxes*.

The created bounding boxes are then evaluated and filtered in *evaluateBoundingBox(offset)*. Especially in images where there is a low contrast between dwellings and background, a huge inhomogeneity on the features of tents or a generally bad image quality, the seed growth algorithm can yield bad results. These cases are supposed to be filtered out by checking, if the sides of the bounding box are smaller than a minBox or longer than a maxBox parameter. If one of these cases occurs, the bounding boxes are marked as corrupted and not used for training. The length of the sides is evaluated without the offset-parameter. The minBox and the maxBox parameter are the third and the fourth of four parameters to control the training data generation process.

After determining the found bounding boxes, the bounding boxes itself and the region is drawn on a copy of the input tile using the methods *drawRegions()* and *drawBoundingBox()*. Valid regions are drawn in a random color with the bounding box around them; corrupt regions are drawn grey without a bounding box around them. Further the initial annotation is marked on that tile with a red pixel, using the method *printAnnotations()*. Finally, the visited array is saved as a separated image, using the method *printVisited()*. Both output images can be used to check if the algorithms are working properly.

Formatting Training Data

The last step to make the data useable for training a Faster R-CNN in the Detectron implementation is writing it in the Microsoft COCO (Common Objects in Context) annotation format (Lin et al., 2014). The COCO dataset is a benchmark object detection dataset and offers ground truth data for object detection tasks including object detection with bounding boxes, image segmentation, image captioning and keypoint detection. For the dataset, its own annotation format was developed which today is widely used for object detection tasks.

The COCO annotation format is written in JavaScript Object Notation (JSON). On the first level it contains five JSON objects: *info*, *licenses*, *images*, *annotations* and *categories*. The *info*-object contains an array of objects describing the dataset with a *description*, a *url*, a *version*, a *year*, a *contributor* and a *date_created* object. The *license* object allows specifying various image licenses. They are characterized by a *url*, an *id* and a *name*. The *categories*-object specifies all categories that are in the dataset. The categories are each stored in a JSON array containing the objects *supercategory*, *id* and *name*. The *id*-object is referenced by annotations to add ground truth, the *name*-object giving context to the *id*. The *supercategory*-object is used to organize datasets with a huge number of categories.

The *info*, *license* and *categories* objects specify information concerning the whole dataset. Therefore, they can be filled before processing the training data whereas the *images* and *annotations* objects refer to the tiles and bounding boxes created by processing the input satellite image and shapefile annotations. This makes it necessary to fill them after the training data procession.

First, the *images*-object is filled. Every image-tile is stored in a separate JSON array, containing the objects *license*, *file_name*, *height*, *width* and *id*. The *license*-object is storing the license id specified for the related license in the *license* object. *id* is a unique identifier for each image, the other three objects store further meta-data. Before adding a tile to the JSON file, it is checked if the id of the tile is already stored in the *images* object of the file to avoid duplicate entries. For the HUMAN+ dataset the ID for one tile is build the following:

Tile Id = <Id of image, int><y-value of tl[17] *in orig. image><x value of tl in orig. image>*

After adding the image in the image object to the JSON output file, the annotations made on the tile are added to the annotation object of the JSON file. The annotation object consists of one array for each annotation, containing the objects *segmentation*, *area*, *iscrowd*, *image_id*, *bbox*, *category_id* and *id*. *Segmentation* is a field which is necessary for semantic segmentation tasks and is filled with ground truth data. Due to the Detectron implementation it needs to be filled so it is filled with the corner points of the bounding box. The *area*-object is filled by multiplying width and height of the bounding box. *Is crowded* is a parameter that can be checked for images where a lot of objects of the same category overlap and are annotated by one bounding box. This is not the case in the HUMAN+ dataset. The *image_id*-object refers to the id specified for the image the annotation is made on. The *bbox* stores the bounding box coordinates by storing the y- and x-coordinates of the top-left corner, the width and the height. *category_id* refers to

[17]Top left corner

the id specified for the associated category in the *categories* object. The id-object stores the id of the annotation which is built as following:

Annotation Id = <Tile Id><Id for specific annotation>

2.2.2. Faster R-CNN Training

Training Data Creation

To create the HUMAN+ dataset used for training a network data from more than one camp is put together in a dataset. After analyzing the first camp, the variable *EXPAND_DATASET* has to be set True to add data to an inchoate dataset. If this variable is set False, a new JSON file for storing the training data in the COCO annotation format is created and already existing files are deleted.

Eight images of camps (see Appendix A.2) located on the Balkan route between Syria and northern France have been used for testing the dwelling detection application. All camps but Rukban have been used for training the Faster R-CNN. Rukban wasn't included because it was offered by a HUMAN+ project partner after the other images were annotated. These camps have a huge variety in the surrounding landscape (from the Rukban camp located in a desert to the Basroch camp located on a clearing in a forest), the organization form (from a planned and organized structure to the unorganized "jungle" of Calais), the size (from a city like structure in Zaatari inhabited by ten-thousands to a few hundred camping in front of a border in Horgos) and the image quality and pixel resolution (from nearly visible people on the Calais image to tents floating into each other in the Basroch image).

Each image was acquired from Google Earth Pro and dwelling annotations on the image were created using QGIS following the same procedure as described in 2.2.1. Fortunately, the georeferencing could be done much coarser because it didn't have to fit to already existing data and was only necessary for mapping the annotations with the image. Creating the annotations was easy from a technical point using QGIS but often proofed difficult on image with lower image quality. It is often hard, even for a human observer to figure out what one tent is, how much tents there are in one structure, where it seems that tents flow into each other, and how many tents are in one of these structures. These difficulties are documented in Figure 2.14 on an input image from the camp Horgos. When annotations for all images are available they are loaded in the training data generation workflow. To get the best results and due to the different characteristics in each image, the four parameters mentioned above have to be adjusted for each image. The parameters are adjusted by visually analyzing the debug images outputted in the workflow. The used parameters can be seen in the Table below. It can be seen that images with a good contrast require a higher threshold than images with a low contrast. A higher threshold makes it harder for a pixel to be added to a region. On images with lower contrast and image quality, the threshold has to be set lower due to the inhomogeneity of regions caused by the worse

Table 2.1: Parameter for the satellite images used for training data generation

Image Name	ID	Threshold	Offset (px)	MinBox (px)	MaxBox (px)
Basroch	1	10	13	0	70
Calais	2	17	10	0	90
Horgos	3	5	12	0	55
Idomeni	4	7	10	1	25
Juba	5	8	7	1	44
Kara Tepe	6	9	12	20	130
Zaatari	7	7	7	0	30
Moria	8	4	15	4	150
Tabanovce	9	5	15	8	150

image quality. Often on lower quality images not the whole dwelling is found. To encapsulate the whole tent it is a good idea to choose a higher threshold even if some information about the form could be lost. The minBox parameter controls the minimal Box extend. It can be used to either filter regions where only the initial pixel is added to the region (or not filter them if necessary) or to filter small results when it is known, that tents have a minimal size in one camp (see Kara Tepe). The maxBox filter is best for filtering out regions where several dwellings were detected. The used parameters for every image are shown in Table 2.1

As a final step, the found ground truth bounding boxes are sorted into a training- and a validation-set. The boxes are sorted regarding their y-coordinate on the input image, making a cut at 70% of the top of the image going into the training set, the rest in the validation set. The exact split-coordinate is calculated in regard to the tiles created out of the input image, ensuring that a tile is either fully in the training or in the validation-set. If the ground truth bounding box is on a tile in the upper 70% of the image, it is added to the training-set, if it is located on a tile in the remaining 30% of the image, it is sorted into the test-set.

Faster R-CNN Training

When a finished training dataset is created the training of the Faster R-CNN can be started. Detectron supplies various out-of-the-box functions for creating, training and testing networks from the R-CNN family. Training is started by *train.py* which is an adaptation of a Detectron script. As an input it takes a config file and the output-directory where the trained weights are supposed to be saved to. The script outputs the trained network weights saved in a .pkl file, a file called *net.pbtxt* describing the trained network and a file called *param_init_net.pbtxt*, describing how the net has to be initialized.

Creation, training and testing are controlled by detectron config values that can be found in *detectron/detectron/core/config.py*. For an individual network, an additional yaml-config file has to be written. There, the network-specific configuration parameters are set. Detectron contains various configuration files for baseline-networks trained at FAIR. For HUMAN+ the Detectron configuration:

detectron/configs/12_2017_baselines/e2e_faster_rcnn_R-101-FPN_2x.yaml

was used as a basis for the HUMAN+ configuration file:

detectron/configs/human_plus/e2e_faster_rcnn_R-50_h+_long.yaml

In the first section of the file called *MODEL*, a CNN backbone, a ResNet50, a state of the art CNN-architecture developed in (He, Zhang, Ren, Sun, 2016) and pre-trained on ImageNet, the number of classes the Faster R-CNN is trained on and that the model is built as a Faster R-CNN is defined. The parameter *NUM_GPUs* specifies if multi-GPU training should be done. The next section *SOLVER* specifies the training algorithm by defining different parameters concerning the learning rate and the length of the training. The values in the detectron-baseline are kept for HUMAN+, only the training iterations are set to a fifth of the original length. In the next section, five anchor-sizes for the RPN network are specified and the detectron-intern building parameters for the RPN are set.

The last two modules specify the train and the test procedure. For training, a link to the weights for a CNN pre-trained on ImageNet is specified. These weights are offered by FAIR. Further, the datasets which the network is going to be trained and tested on are specified, the maximum input sizes of the network as well as the SGD batch sizes. The train- and validation-dataset has to be registered in the detectron software in *detectron/detectron/datasets/dataset_catalog.py* and the actual data has to be copied into *detectron/detectron/datasets/data* using a symlink. The parameters for testing are similar. First the test dataset is specified, then input scales follow. Specific for training are some post-processing parameters, filtering the bounding boxes found by the Faster R-CNN.

2.2.3. Dwelling Detection Workflow

When a Faster R-CNN is trained, it can be used for the main application which is doing the dwelling detection. The workflow for analyzing a new satellite image of a refugee camp is controlled in the *analyze_sat.py* script.

Loading and Processing Satellite image

For running a trained Detectron net like the Faster R-CNN trained before, Detectron offers a sample script on images in an input folder. This script is adapted for the application and can be found in *interference.py*. Before utilizing the Faster R-CNN the input satellite image has to be processed to a form that fits in the network. The workflow for doing so begins by loading the new raster with rasterio. This allows collecting the metadata regarding georeferencing in the case that the input image is georeferenced. The geoinformation are needed to visualize the dwelling detection results in the software developed in the HUMAN+ project. So, the image is cut into tiles, following the workflow used in the training data generation, described in 2.2.1. Further, for each tile a JSON file is created, storing the results after analysis.

Interference

After processing the input image, the interference workflow is started. First, the parameters of the workflow have to be set. The basic testing configurations are set in the .yaml file that was already used for training the network. Further the weights for the network defined in the .yaml file have to be specified. The weights are the output of the training process, stored in a .pkl file. In addition, the folder containing the images to be analyzed, the image extension of the images that are going to be analyzed and the output folder, storing the visual representation of the analysis has to be set. When all parameters are set, the interference process is started.

The interference script in *interference.py* resembles mostly the script offered by Detectron. The way the input parameters are handed over is changed to fit it into the applications workflow. Further a method to write the found objects and their class scores into a JSON file is added. The script first loads all parameters: the Faster R-CNN weights, the Faster R-CNN initialized with the loaded weights, the dataset definition and a list all the images it has to test. The dataset definition contains the classes that are tested. Then interference is carried out for every image specified to be tested using *detectron.core.test_engine*. The results of the interference are stored in the variable *cls_boxes*, which are written in the tile-specific JSON file created before. Finally, the script visualizes the found objects in a tile and saves the processed tile as an image in the output folder specified. This image can be used for debugging purpose.

Analyzing Tiles

The bounding boxes obtained by Detectron have class securities from 0.05 to nearly 1. A class security of 0.05 denotes that the Faster R-CNN is 5% sure that the object contained in the bounding box really is a tent and, due to the fact that only one class was trained, 95% sure that the object in the bounding box is not a tent but background or another object. This is unsuitable for a reliable prediction. Bounding boxes with a class security lower than 0.5 can be identified as negative result because the network is more than 50% sure that the bounding box is not encapsulating a tent. Therefore it is important to set a threshold for which bounding boxes to count as real findings. As in (Ghorbanzadeh et al., 2018) I set the threshold for a bounding box being a dwelling or not to 0.85. All accepted bounding boxes are then written in a new JSON file. Before writing the coordinates into the JSON file, they are transferred from being coordinates on the tile to coordinates on the actual input image, using the position of the tile in the input image, stored in its filename.

The accepted bounding boxes on the original input image are further drawn on the input image and the processed image is saved.

Besides the actual found dwellings and the calculated number of residents living in them, it is from great importance in a real life application, to give a measurement how good the estimation in general is on the whole image. This confidence metric can be calculated using the class security values outputted by the Faster R-CNN and has to be image-specific. Image-specific information is important, because the detector works differently on differing images. Detection quality depends on many factors: The image quality, the camp structure and how close the image is to training data are some. One option for estimating the per-image quality of the detection is the processed input image, outputted after evaluating the bounding boxes. Still, in HUMAN+ there should also be a numeric, more concrete and easy to interpret value for the quality of the detection on one image. One option for a confidence value would be to just give out the classification threshold used for filtering positive and negative bounding boxes. Problematic with this approach is, that it only gives a coarse and little meaningful estimate. It gives a measurement that the network is sufficiently sure that the found boxes are dwellings. Unfortunately, this is just a statement about how the detector is working in general. It gives no image-specific information about the findings.

To get a measurement, how sure the detector is on a distinct image, the bounding boxes with a class security between 0.5 and 0.85 are considered. These values are chosen because for bounding boxes that have a class security bigger than 0.5 it can be said, that the network is surer that the content in the box is a dwelling than that it is something different or belonging to the background while 0.85 is the threshold where bounding boxes are classified as positives. The considered bounding boxes can therefore be seen as bounding boxes, where the detector is in doubt about how to classify the content of the box. The confidence score is calculated by dividing the number of bounding boxes with class securities bigger than 0.85 through the number of bounding boxes with class securities bigger than 0.5. The idea is that the more bounding boxes have class securities on the interval between 0.5 and 0.85 the more uncertain is the detector.

Processing Results

After having trained a working Faster R-CNN which yields acceptable to great results on all test-images of camps the ability of the net to perform its core task was tested. The goal of dwelling detection is to calculate the number of people living in a camp from a satellite image.

A common approach is to count the number of dwellings in a camp and to multiply that number with a coefficient describing how many people are in a tent in a camp in average [Watkins 1985]. A more precise way is to take the size of the dwellings, an information the Faster R-CNN is outputting in pixel-values, into account.

The first idea to take this information into account was to define three categories: Large, medium and small tents. Each of these categories had a parameter stating how many people were living in a dwelling of the respective categories. This approach has two problems: First, the distinction of different tent sizes is quite coarse; in reality there are more than three tent sizes. Second, and more important, the way each dwelling is categorized is that its size is calculated and if the size is under a threshold, the bounding box is categorized to one of the categories. Therefore, this method is dependent of the absolute number of pixels in a bounding box. This is a huge problem when looking at satellite images shot from a different height or with a different pixel resolution. The same tent can have a different size on different satellite images. In conclusion, it is more guessing a number than carrying out a reliable calculation.

Therefore, a better approach for calculating the number of people in a camp from the results of the faster R-CNN regarding the different characteristics of satellite images was implemented:

The implemented approach is independent from further additional information than the output of the Faster R-CNN and works solely on the data outputted of the network. The key assumption is, that there is a variance of tent sizes with different numbers of people in them in one camp. Therefore, to the smallest tent size and the largest tent size the number of people possibly living in those tents is assigned. For the tent sizes between those anchor points, the number of people living in a tent is interpolated linearly. The calculated number is not dependent of the absolute number of pixels in a bounding box but of the sizes of the bounding boxes in one image in relation to each other. This approach is implemented with promising results 2.3.2.

2.2.4. Implementation Details

The application is run in a Docker-container, running on Nvidia-Docker 2. The container contains the HUMAN+ application described in this thesis and the Detectron software. The software uses the libraries cocoapi, Cython, detectron, Fiona, future, matplotlib, mock, numpy, opencv-python, protobuf, pyproj, python3-gdal, pytorch, queuelib, rasterio, scikit-image, scipy, six and setuptools.

Some minor modifications were made to the Detectron software to fit it into the HUMAN+ workflow.

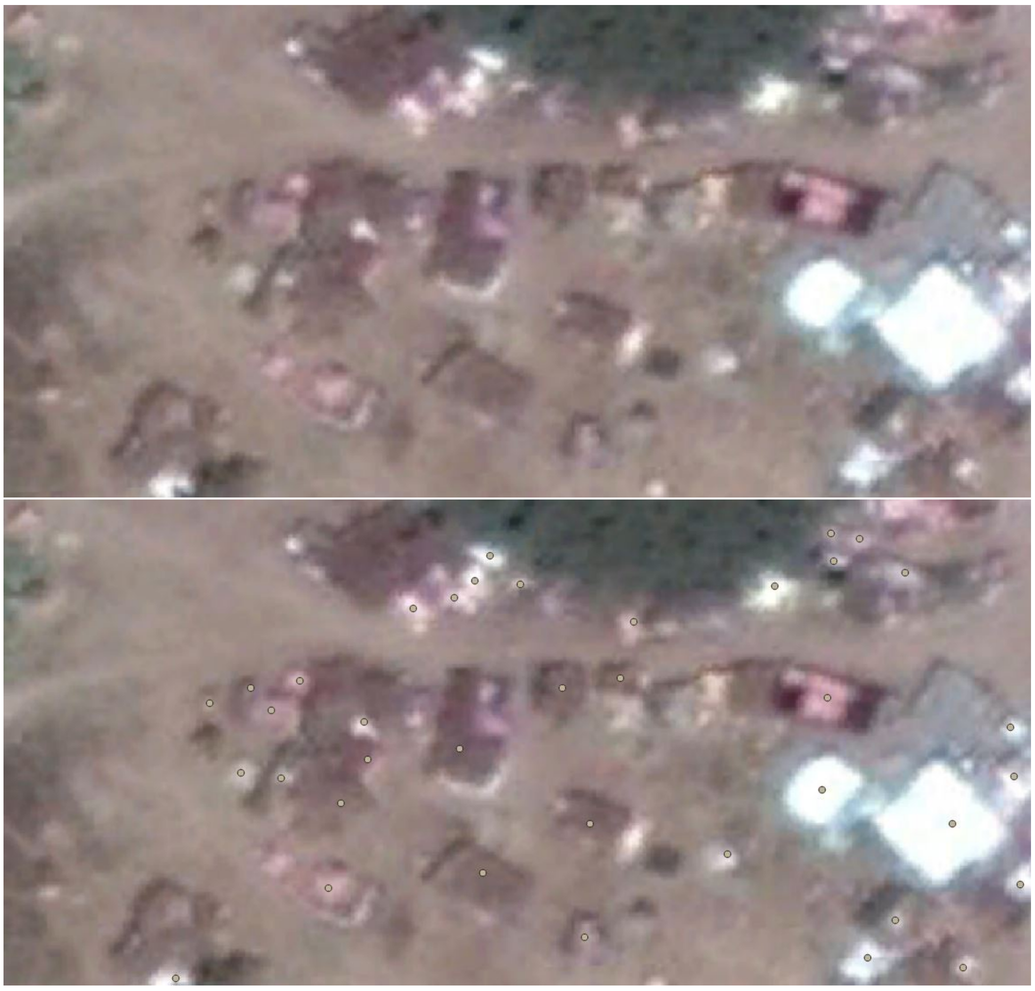


Figure 2.14: Input Data creation on a satellite image of the camp Horgos: Due to the bad image quality and the low contrast between dwellings and background, the annotation is complicated.

2.3. Results

2.3.1. Machine Learning Metrics

To get the best Faster R-CNN for the dwelling detection task, various training passes with different parameters, specified in Table 2.2, were conducted.

The parameters changed were the amounts of camps that were included in the training data and the training-iterations used for training the network. For every new training set containing a new set of camps, a new validation set was created as described in 2.2.2. Each trained net was tested on the validation set created with the last training set. This validation set contains all images trained on. It has the full diversity in structure, organization, localization and image quality that is considered in this thesis. The mAP is measured using the COCO-evaluation metrics. The results of the training can be seen in Table 2.2.

The first four training passes have been first experiments to check if the method is working in general. The first training and testing were done on Juba, the only camp with available UNHCR annotations. In Juba, two separate camps where annotated, the main camp, Juba1, and a smaller peripheral camp, Juba2. Alas, the image quality and the training data generation quality of Juba2 are poor, so Juba2 was dropped from training and testing from training four on. Training three and four were conducted after acquiring two more satellite images and creating training data on them. At this point, no manual adjustments on the training data process using the four parameters introduced in 2.2.2 were made. This low level on adjustments and optimizations can explain the generally low mean Average Precision (mAP) around 20%.

The last three training passes were conducted after optimizing the training process, fully defining the four training data generation parameters. Further they include an enhanced set of camps in the training set. In training five, seven images of different camps are used for training the network. Further, optimized parameters have been used for creating the training data set. The parameters can be found in Table 2.1. The enhanced training set leads to a huge jump in detection performance of the network on the full validation set with an increase of 28,5% in mAP compared with the earlier experiments. The increase in mAP denotes that the Faster R-CNN finds more of the ground truth boxes defined in the validation set. Still, a visual analysis on the output of the Faster R-CNN when using the available images of camps as input showed that there were still dwellings that were not found and a quite high number of False Positives. Therefore the network was trained two times as long as before, using 36.000 iterations, which is a fifth of the training time used for the baseline training of the COCO dataset, conducted by FAIR. This leads to a quite high increase in performance of the network, increasing mAP for around 7.4%. This shows that a longer and finer training leads to better results. The training iterations were not increased after that to minimize the risk of overfitting the network. Overfitting is

Table 2.2: Parameter and mAP for all training passes

Training	**Camps**	**Training Iteration**	**mAP (COCO) in %**
1	Juba1	18.000	19,9
2	Juba1, Juba2	18.000	19,4
3	Juba1, Juba2, Idomeni	18.000	20,6
4	Juba1, Zaatari, Idomeni	18.000	20,2
5	Basroch, Calais, Horgos, Idomeni, Juba1 Kara Tepe, Zaatari	18.000	49,1
6	Basroch, Calais, Horgos, Idomeni, Juba1, Kara Tepe, Zaatari	36.000	56,5
7	Basroch, Calais, Horgos, Idomeni, Juba1, Kara Tepe, Moria, Tabanovce, Zaatari	36.000	57,1

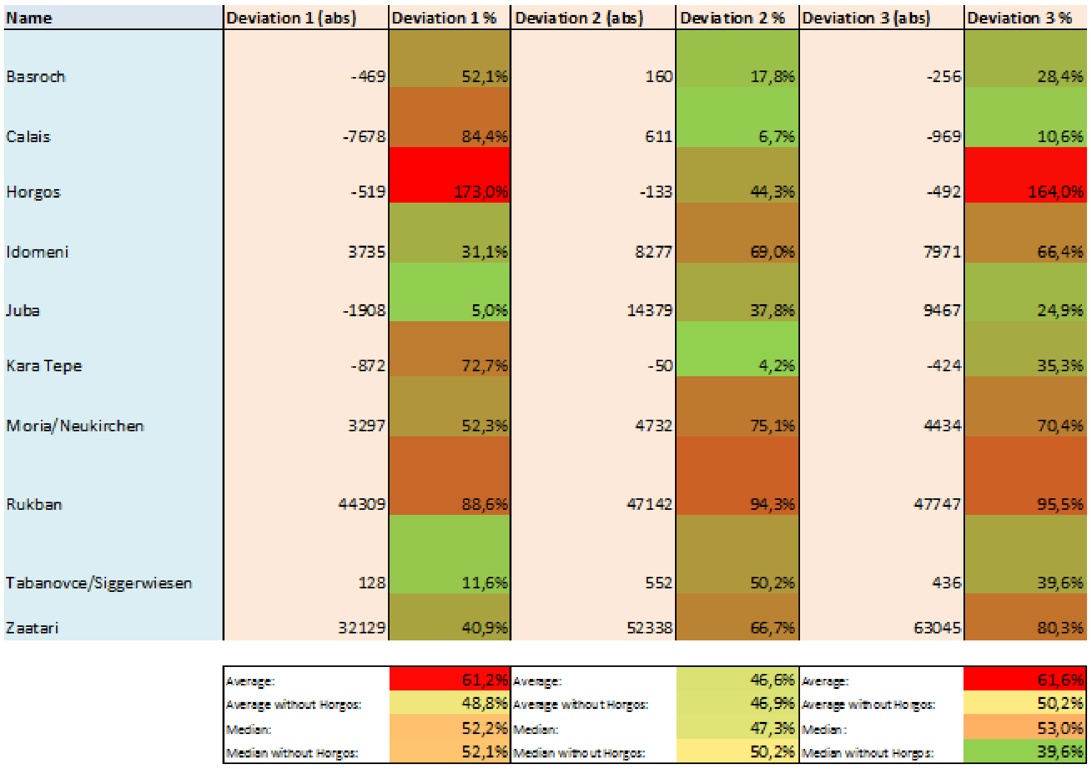
a phenomena which occurs when “standard backpropagation learning builds up brittle co-adaptions that work for the training data but do not generalize to unseen data” (Srivastava et al., 2014)

For the last training pass, the camps Moria and Tabanovce were included in the training data after achieving unsatisfactory results when testing the network trained in Training 6 on those images. Including those two images increases the mAP for 0,6% and yields better results for the two added images. This is explainable with the observation that both Moria and Tabanove show dwellings with structures not existent in the other camps.

2.3.2. Validity of Dwelling Detection

After having validated the performance of the trained Faster R-CNN, experiments on the dwelling detection algorithm were conducted. To test its validity, the number of people living in the camps at the time the related images were taken were researched (see Appendix A.1).

The analysis routine was run once with the initial algorithm classifying dwellings after their actual pixel size (Deviation 1) and twice with the linear interpolation approach (Deviation 2/3), using different parameter each iteration. For the first iteration the smallest tent was defined to have 3 inhabitants and the largest to have 8. For the second iteration the smallest tent was defined to have 3 inhabitants and the largest to have 12 inhabitants. Concerning the threshold, when a bounding box was considered as a dwelling, the dwelling detection workflows were run with a variable threshold fitted for each camp in the first two approaches and with a fixed threshold of 0.85 in the third approach. The threshold was set to a fixed value after developing the confidence metric honoring the image-specific aspects of each input image. The absolute numbers of found tents and estimated people



Name	Deviation 1 (abs)	Deviation 1 %	Deviation 2 (abs)	Deviation 2 %	Deviation 3 (abs)	Deviation 3 %
Basroch	-469	52,1%	160	17,8%	-256	28,4%
Calais	-7678	84,4%	611	6,7%	-969	10,6%
Horgos	-519	173,0%	-133	44,3%	-492	164,0%
Idomeni	3735	31,1%	8277	69,0%	7971	66,4%
Juba	-1908	5,0%	14379	37,8%	9467	24,9%
Kara Tepe	-872	72,7%	-50	4,2%	-424	35,3%
Moria/Neukirchen	3297	52,3%	4732	75,1%	4434	70,4%
Rukban	44309	88,6%	47142	94,3%	47747	95,5%
Tabanovce/Siggerwiesen	128	11,6%	552	50,2%	436	39,6%
Zaatari	32129	40,9%	52338	66,7%	63045	80,3%

Average:	61,2%	Average:	46,6%	Average:	61,6%
Average without Horgos:	48,8%	Average without Horgos:	46,9%	Average without Horgos:	50,2%
Median:	52,2%	Median:	47,3%	Median:	53,0%
Median without Horgos:	52,1%	Median without Horgos:	50,2%	Median without Horgos:	39,6%

Figure 2.15: Deviations of the actual and the estimated numbers per camp.

as well as the official number of inhabitants for each camp can be found in Appendix A.1, Figure A.1 and Figure A.2.

To evaluate the results, the deviation of the estimated and the actual inhabitants per camp were calculated in absolute numbers and in percentage of the deviation to the actual number (Figure 2.15). After that, the mean and the median over the deviations of all camps were calculated for each method.

Concerning the absolute numbers the estimations were devided from the official. Therefore, a negative value indicates that the estimation is too high; a positive number indicates that the estimation is too low. The percentage values are calculated using absolute values. It can be seen, that the deviation in percentage is higher for camps with a low number of actual residents living in it and therefore small deviations have a higher impact on the resulting number. Taking a first look at the results, it falls into the eye that the camp of Horgos has a high deviation in percentage and can be seen as an outlier. This can be explained with the low absolute number of inhabitants in the camp. Further the source could be inaccurate. Therefore the averages and mean values were calculated with and without the camp Horgos.

Looking at the average and mean values, Approach 1 has the middle results in the categories Average, Average without Horgos and Median with approach 3 being 0.4% to 1.4%

worse results. It is important to mention that the parameters (small, medium and large tent, threshold) were highly tuned to achieve fitting results, while no fine-tuning on the parameters were applied to Approach 3. In the real HUMAN+ application it is not anticipated and further not possible that users fine-tune the algorithm parameters for each camp. Anyhow, fine tuning is not possible because the values towards which the parameters have to be optimized are not known. Therefore, Approach 1 is not suitable for a dwelling detection application.

Approach 2 is winning in the categories *Average*, *Average without Horgos* and *Median*. Approach 2 delivers generally well results because it estimates the number of people lower and more conservative than the other two approaches. Therefore, in camps with fewer actual inhabitants, where Approach 1 & 3 estimate a number of inhabitants too high resulting in bad percentage values, Approach 2 is near the official values. Further, in camps with a very high number of inhabitants, the lower estimated values doesn't fall that much into weight. It also needs to be remarked that Approach 2 is using the optimized threshold values for each camps that Approach 1 is using.

Still, Approach 3 shows it strength in the category *Median without Horgos*, having a result which is 10.6% better than the second best value. The median is showing the main emphasis of a distribution. The high value in the last category indicates that eliminating the outliners, Approach 3 is giving the most stable estimates compared to the other two. It further is the most realistic approach because it relies on no fine-tuning of parameters.

3. Outcome

3.1. Discussion

In this Bachelor Thesis I developed a Dwelling Detection application using a Faster R-CNN. To achieve this, I showed the various steps, challenges and considerations I had to take to develop this application. I could show that it is possible to develop such an application using open software and data. Therefore, the work hypothesis that machine learning should be usable to analyze satellite imagery could be confirmed. Still, much work can be further done on this project and on this topic.

First of all, it is beyond the scope of this thesis to detect refugee camps in satellite images.

Then it is out of scope to develop the best possible algorithm for each of the different parts of the software. This thesis relies on standard software and methods, each aspect of the application can be enhanced in itself. Enhancements can be considered beginning with the input data itself, satellite images extracted from Google Earth. They were extracted by zooming to a zoom level where the whole camp could be seen and then extracted at once. Extracting multiple parts of one image and stitching them together or buying the original satellite images could greatly enhance the quality of the results. Buying the original images would also make georeferencing each image by hand, which led to minor distortions of the input image, unnecessary.

The next step, the finding of the regions and the generation of the bounding boxes can be enhanced. The used algorithm is a pretty basic and easy to implement algorithm. A more complex image segmentation algorithm using seed points could yield drastically more exact results. (H. Zhu et al., 2016) offer a detailed review over state-of-the-art algorithms in that field.

It has to be noted, that this thesis is not a review of modern machine learning architectures but a study how to best make use of already existing and tested techniques. Therefore, the focus of this thesis was not on the optimization of a CNN architecture and its super parameters. Regarding the choice of the right architecture, (Han et al., 2018) offer a detailed study about modern deep object detection techniques. Still, it is of great importance to understand how the underlying techniques work to develop and evaluate an application built with them. First approaches in this direction were taken at Fraunhofer IAIS in the project DeepGeo, where a visualizer for the architecture and learned features of a CNN was developed (Landenberg, 2019).

An interesting and concrete next step following this thesis would be to change the Deep Learning Architecture from a Faster R-CNN to a Mask R-CNN (He et al., 2017) like (Quinn et al., 2018) did in their approach. Mask-RCNN enables semantic segmentation making it possible to detect a dwelling on pixel-basis. When the spatial resolution of a pixel in the satellite image and a parameter, how much space one inhabitant of a refugee camp is occupying is available, very accurate results are possible.

Further, some specific problems and approaches worth investigating occurred during the development and writing of this thesis.

When looking at the input into the Dwelling Detection classifier: A problem with satellite imagery derived from optical sensors is that it can be heavily distorted by cloud cover or atmospheric haze. A solution proposed by (Braun & Hochschild, 2017) is to further use radar satellite data to analyze refugee camps. Radar data is not affected by atmospheric conditions.

Regarding the output of the Faster R-CNN, it occurred that when analyzing not seen satellite images, the network outputs worse results as suspected when looking at the metrics derived when testing it with the test set. The main reason for that is that the test set is actually the validation set. It is created from the same images the train set is created on and was used for training. This is useful for checking if the network learned well on the images it was trained on but gives no information how it handles new situations, be it different backgrounds, different dwellings, different image qualities or different spatial resolutions. Creating a unique test set containing not seen dwellings from the camps that training was conducted on and adding images of camps that were not used in the train set could give better performance estimates.

A reason for worse results on images not seen by the network while training than on images seen by the network while training can be overfitting as described in 2.3.1. This can be prevented by adding additional data augmentation before fitting the trained images in the network for training. (Quinn et al., 2018) vary the brightness and scale and randomly rotate and flip training images. This could be a useful addition to the training process.

Processing the results of the Faster R-CNN is an important step in the Dwelling Detection workflow: Only here the actual information looked for can be derived. In that stage, two problems occurred.

The first problem concerns the estimation of inhabitants of one camp. In the current implementation, the sizes of dwellings only considered after interference of the Faster R-CNN. When defined carefully under consideration of special domain knowledge, different tent sizes and usage forms of dwellings can be defined in training data and therefore be learned by the neural network. Each dwelling type would be its own trainable class. (Ghorbanzadeh et al., 2018) use such a multiple class approach in their CNN implementation.

Another approach for estimating the inhabitants of a camp including the sizes of tents relies on the spatial resolution of one pixel of a satellite image. Given a parameter, how much space one refugee is occupying in a tent, the footprint of one dwelling can be multiplied with this parameter. The footprint of the dwelling has to be calculated from the output of the detection network. The approach is used in (Quinn et al., 2018) using dwelling segmentations to calculate the footprint of a dwelling.

The second problem concerns the improvement of an already trained dwelling detector by domain experts. Due to the high variation in satellite imagery of the earth, certain structures on one image can strongly resemble tents on other. This becomes apparent when looking at satellite images of desert-like environments like the Rukban-image, where here and there white spaces appear on the ground. These strongly resemble white dwellings on other satellite images and often cause false positives. (Quinn et al., 2018) suggest an assisted mapping process to minimize the number of false positives. They propose letting experts adjust the output of the trained detector and train the net with the adjusted ground truth data. A further enhancement of this method could be to create a negative background class. In Addition to just delete false-positives, they could be declared to be background and therefore the wrong solution. This method is called “bootstrapping” or “hard negative mining” (Sung, 1996). The technique is already used in the Faster R-CNN training process but could be enhanced by manual input.

3.2. Conclusion

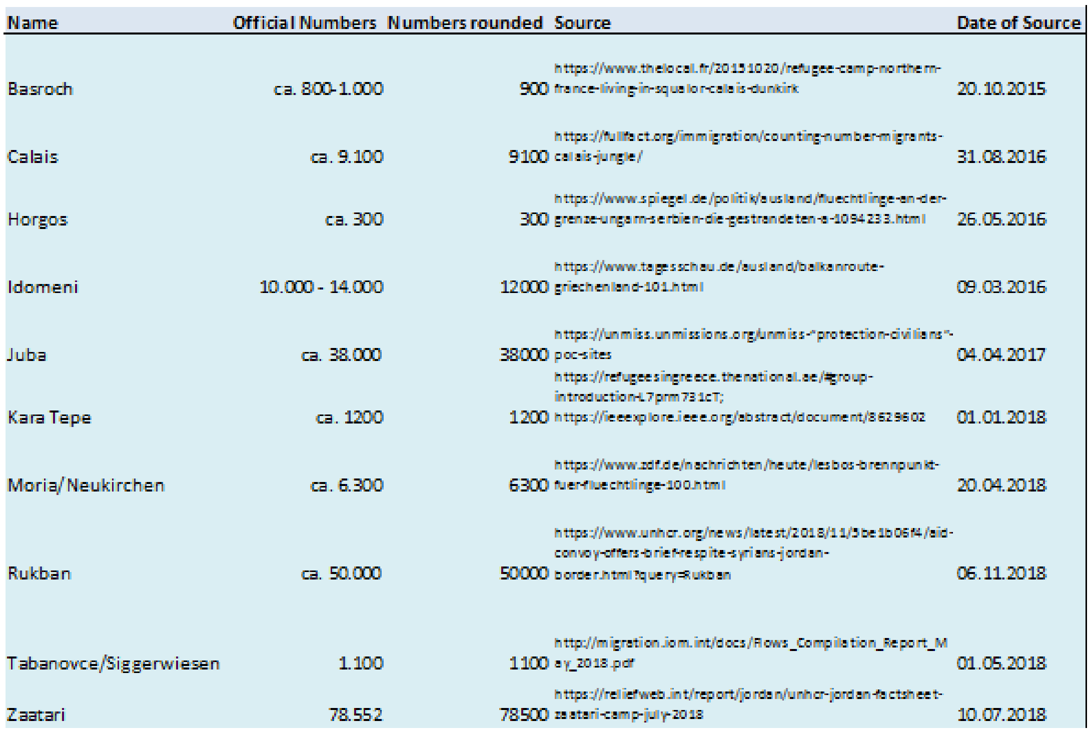
In this thesis I could show that my assumption, that it is possible to build an object detector and classificator for dwelling extraction using a Faster R-CNN is correct. My classifier is able to detect the dwellings it was trained on and also finds objects in images it has not seen before. Further, false positives are not as frequent as I feared. False Positives could have been a problem regarding the diverse structures and manifestations of the different camps. A huge part in holding off False Positives is the post-detection filtering mechanism created.

To achieve these results, I had to build up a machine learning environment. It has been shown that for implementing a successful machine learning classifier it is not enough to just build a sophisticated architecture. A huge importance lies on the data: the data quality, choosing the right data, creating training data and storing them in the right format. Especially writing your data in the Microsoft COCO format is an important step in every object detection machine learning project. Further it is from great importance when using already existing software to adapt them correctly to one’s own problem. Even when Facebooks Detectron software is a powerful and sophisticated tool, a great deal of adaption has to be done to fit it into one’s own workflow.

Building up a Faster R-CNN workflow is a great achievement. Anyway, it is clear that building on these results still a lot of improvements can be made. More in-detail experiments on the trained Faster R-CNN followed by a reevaluation of the networks architecture could improve the detection results. Further tackling the problem of image segmentation using a Mask R-CNN would be the logical next step when thinking about new directions to go. This would involve improving the region detection algorithm while creating the training data. Also the method to estimate the number of people from the found dwellings could be enhanced.

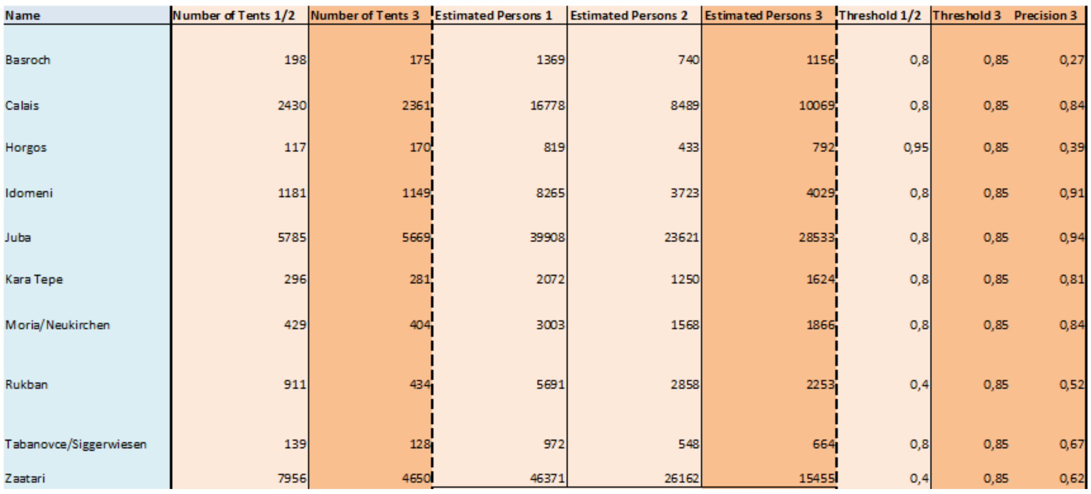
A. Appendix

A.1. Sources for the inhabitants of the found Refugee Camps



Name	Official Numbers	Numbers rounded	Source	Date of Source
Basroch	ca. 800-1.000	900	https://www.thelocal.fr/20151020/refugee-camp-northern-france-living-in-squalor-calais-dunkirk	20.10.2015
Calais	ca. 9.100	9100	https://fullfact.org/immigration/counting-number-migrants-calais-jungle/	31.08.2016
Horgos	ca. 300	300	https://www.spiegel.de/politik/ausland/fluechtlinge-an-der-grenze-ungarn-serbien-die-gestrandeten-a-1094233.html	26.05.2016
Idomeni	10.000 - 14.000	12000	https://www.tagesschau.de/ausland/balkanroute-griechenland-101.html	09.03.2016
Juba	ca. 38.000	38000	https://unmiss.unmissions.org/unmiss-"protection-civilians"-poc-sites	04.04.2017
Kara Tepe	ca. 1200	1200	https://refugeesingreece.thenational.ae/#group-introduction-L7prm731cT; https://ieeexplore.ieee.org/abstract/document/8629602	01.01.2018
Moria/Neukirchen	ca. 6.300	6300	https://www.zdf.de/nachrichten/heute/lesbos-brennpunkt-fuer-fluechtlinge-100.html	20.04.2018
Rukban	ca. 50.000	50000	https://www.unhcr.org/news/latest/2018/11/5be1b06f4/aid-convoy-offers-brief-respite-syrians-jordan-border.html?query=Rukban	06.11.2018
Tabanovce/Siggerwiesen	1.100	1100	http://migration.iom.int/docs/Flows_Compilation_Report_May_2018.pdf	01.05.2018
Zaatari	78.552	78500	https://reliefweb.int/report/jordan/unhcr-jordan-factsheet-zaatari-camp-july-2018	10.07.2018

Figure A.1: Official Numbers of Inhabitants in one camp at the time of the recording of the satellite image



Name	Number of Tents 1/2	Number of Tents 3	Estimated Persons 1	Estimated Persons 2	Estimated Persons 3	Threshold 1/2	Threshold 3	Precision 3
Basroch	198	175	1369	740	1156	0,8	0,85	0,27
Calais	2430	2361	16778	8489	10069	0,8	0,85	0,84
Horgos	117	170	819	433	792	0,95	0,85	0,39
Idomeni	1181	1149	8265	3723	4029	0,8	0,85	0,91
Juba	5785	5669	39908	23621	28533	0,8	0,85	0,94
Kara Tepe	296	281	2072	1250	1624	0,8	0,85	0,81
Moria/Neukirchen	429	404	3003	1568	1866	0,8	0,85	0,84
Rukban	911	434	5691	2858	2253	0,4	0,85	0,52
Tabanovce/Siggerwiesen	139	128	972	548	664	0,8	0,85	0,67
Zaatari	7956	4650	46371	26162	15455	0,4	0,85	0,62

Figure A.2: Results of the tests of the Faster R-CNN and the estimation algorithms

A.2. Used satellite images

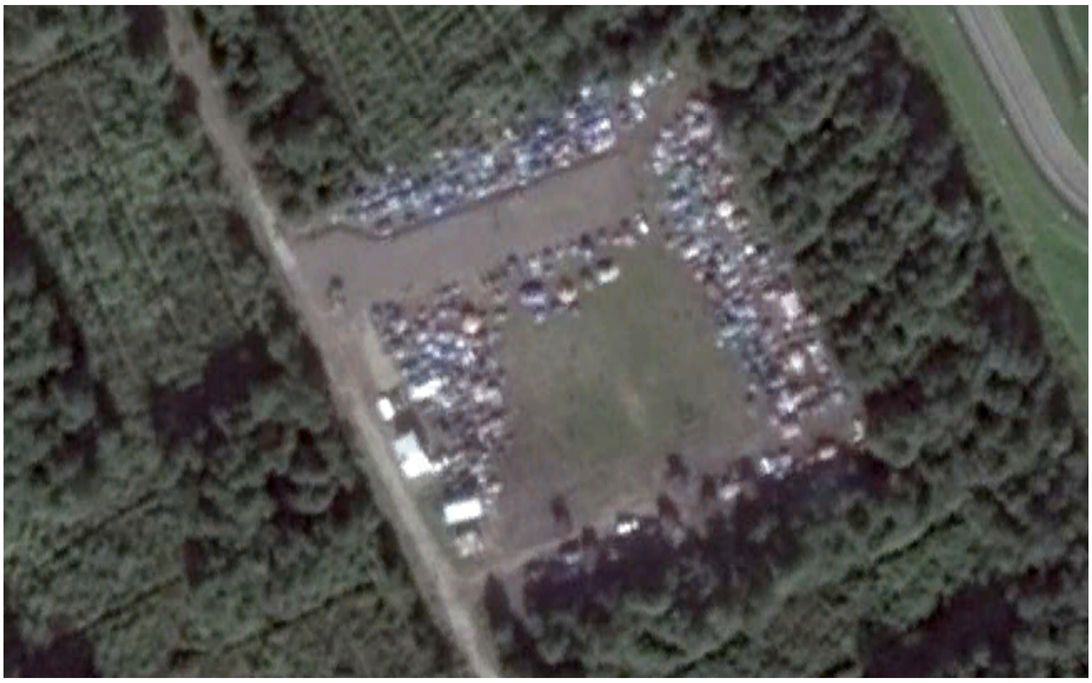


Figure A.3: Basroch

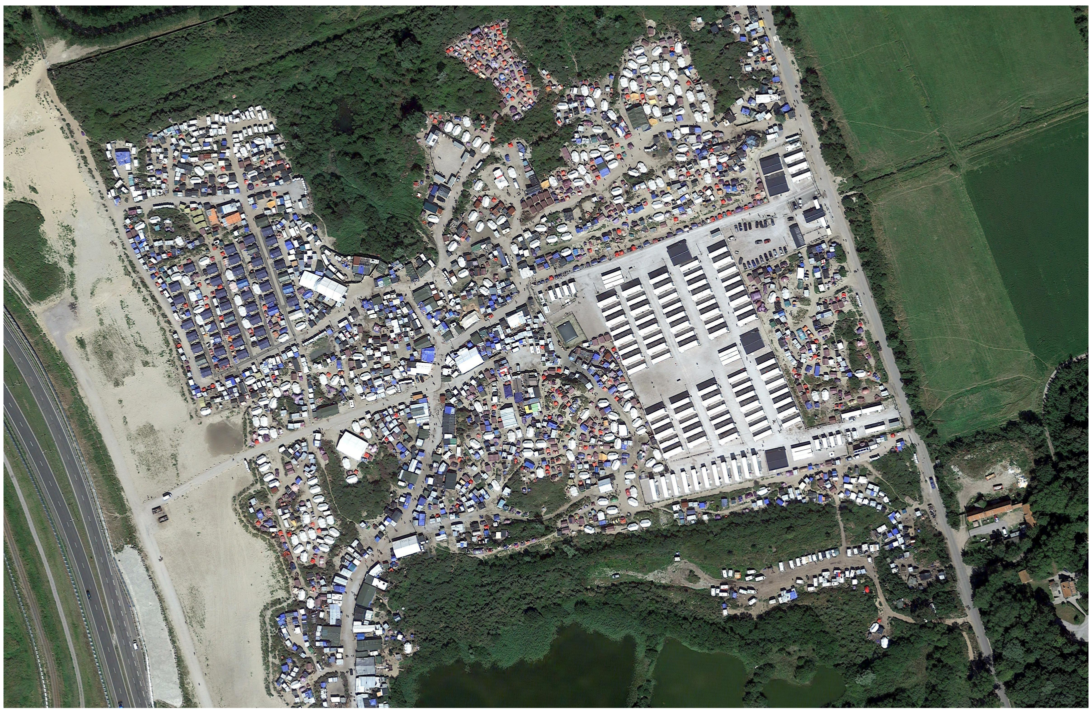


Figure A.4: Calais

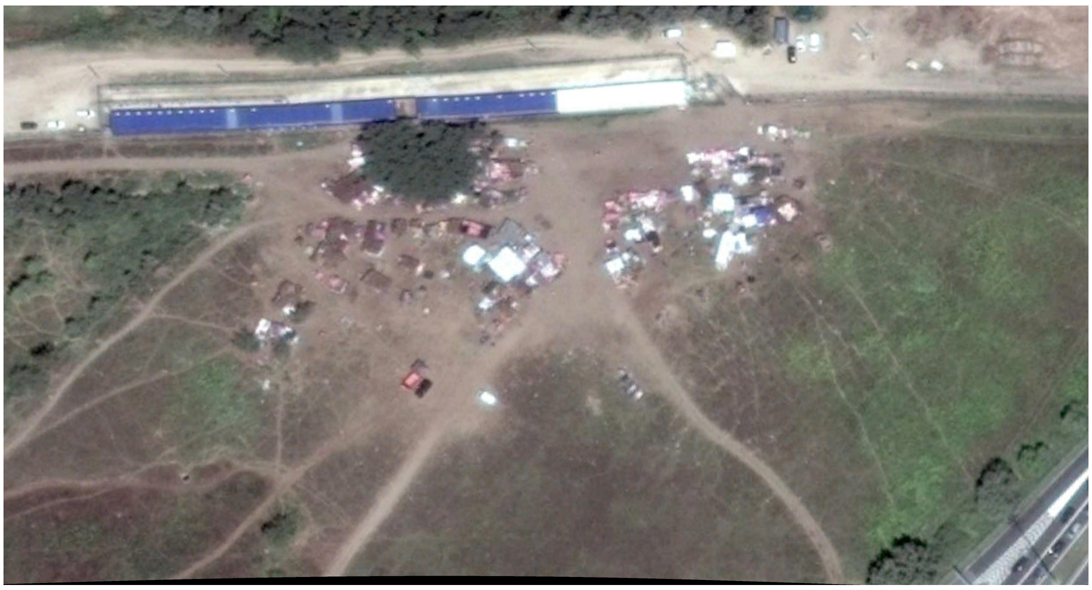


Figure A.5: Horgos

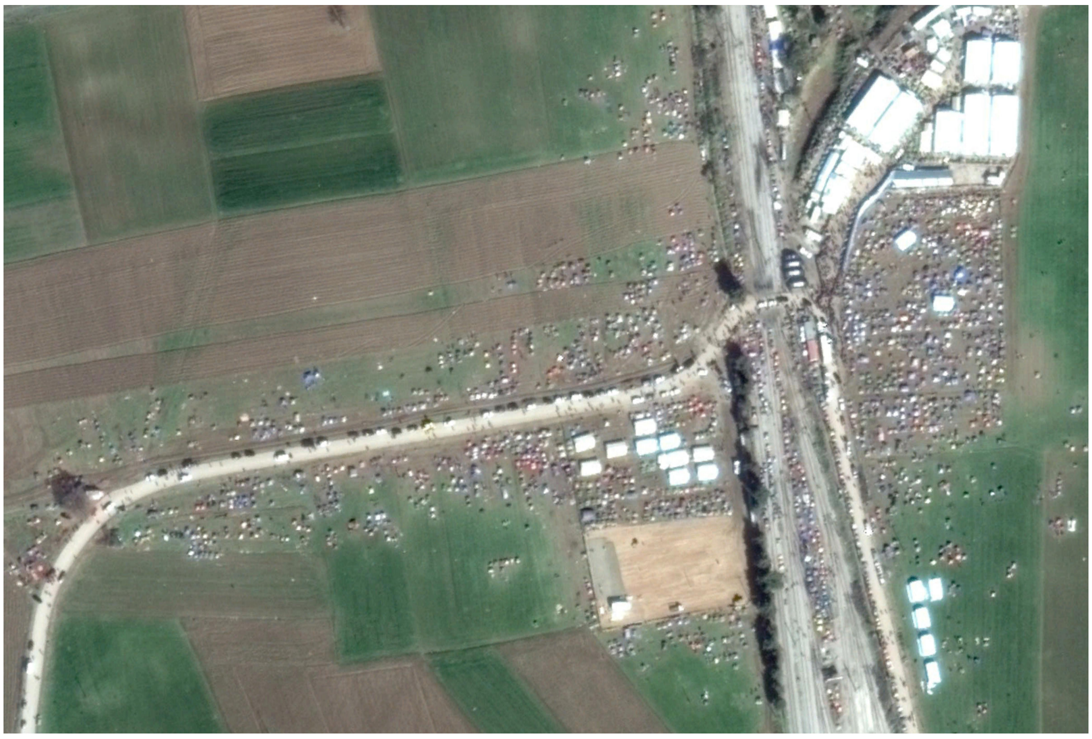


Figure A.6: Calais

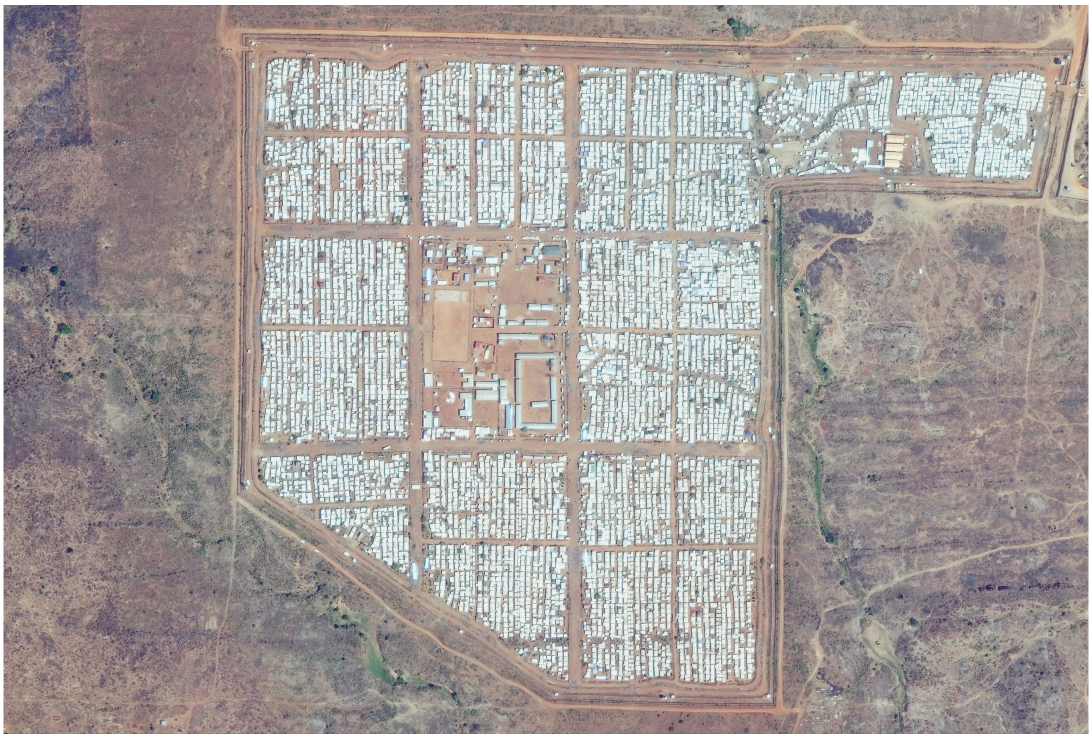


Figure A.7: Juba

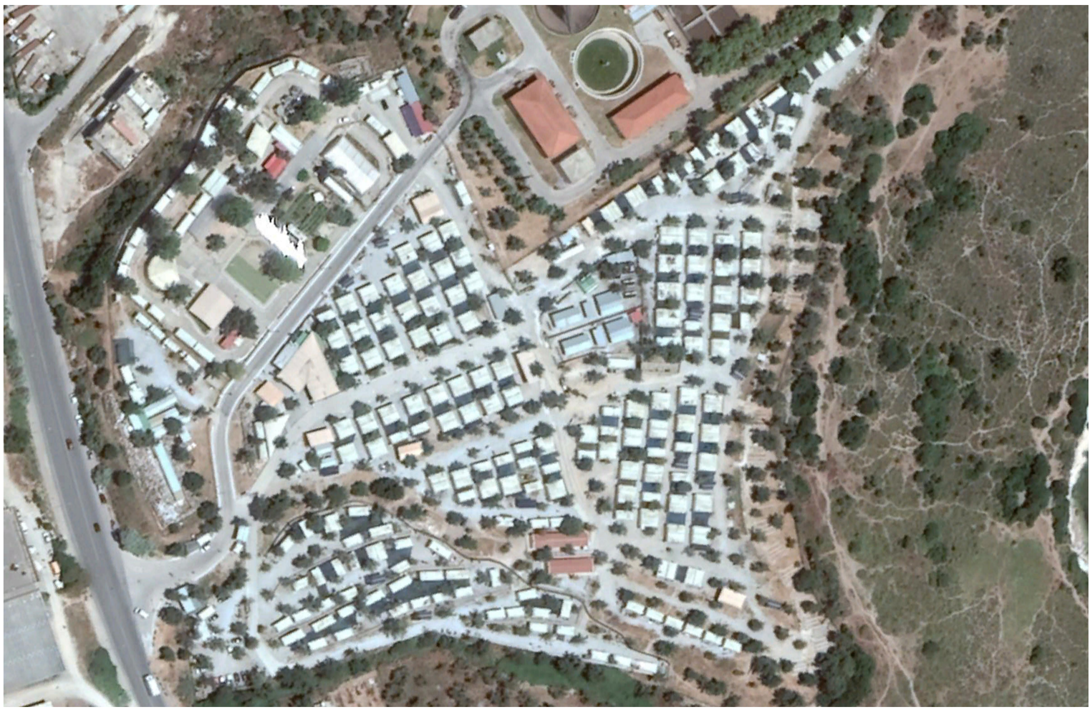


Figure A.8: Kara Tepe

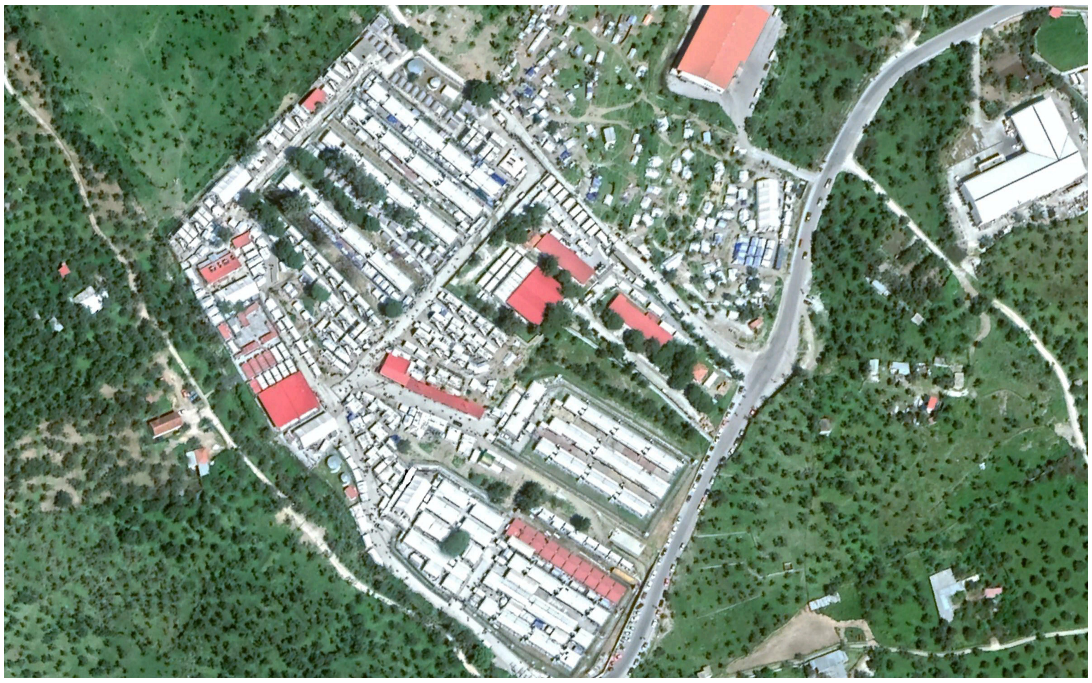


Figure A.9: Moria

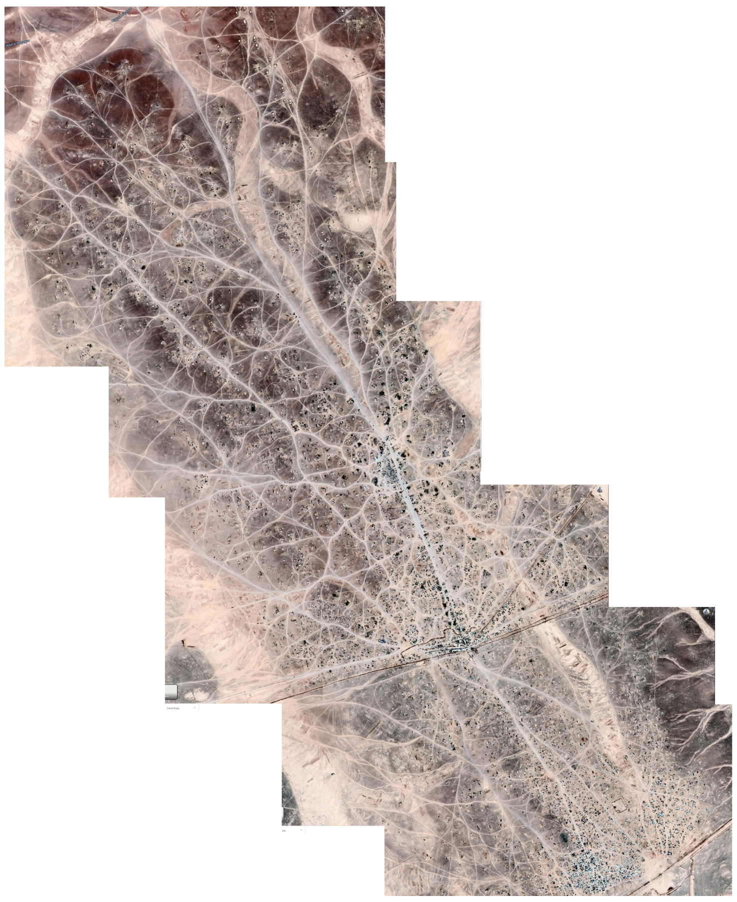


Figure A.10: Rukban

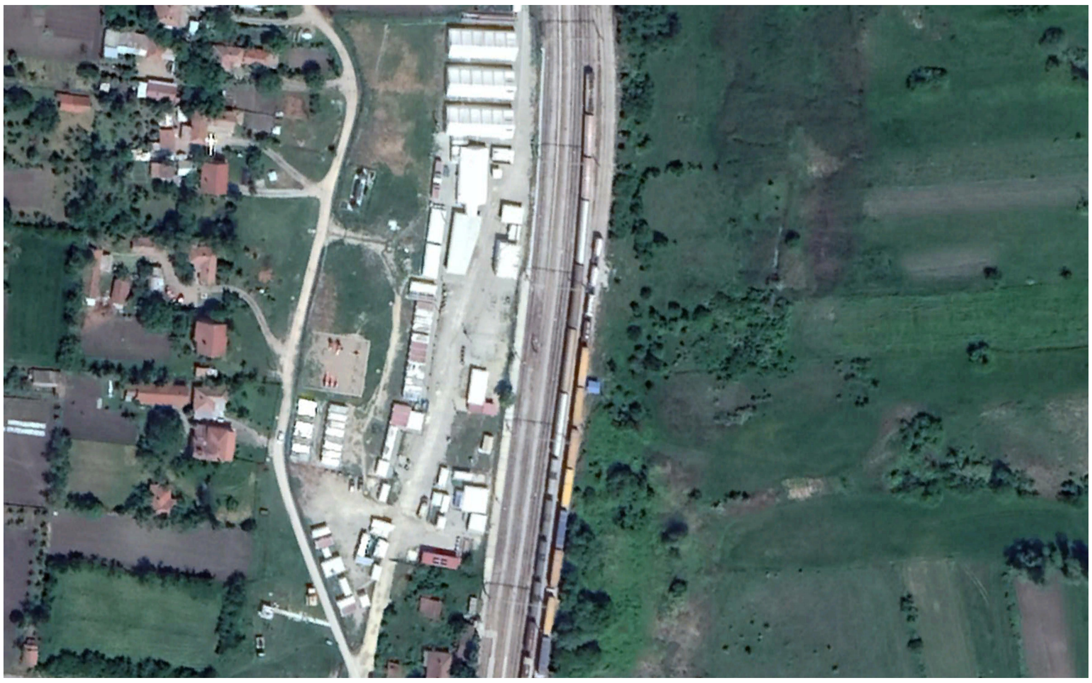


Figure A.11: Tabanovce

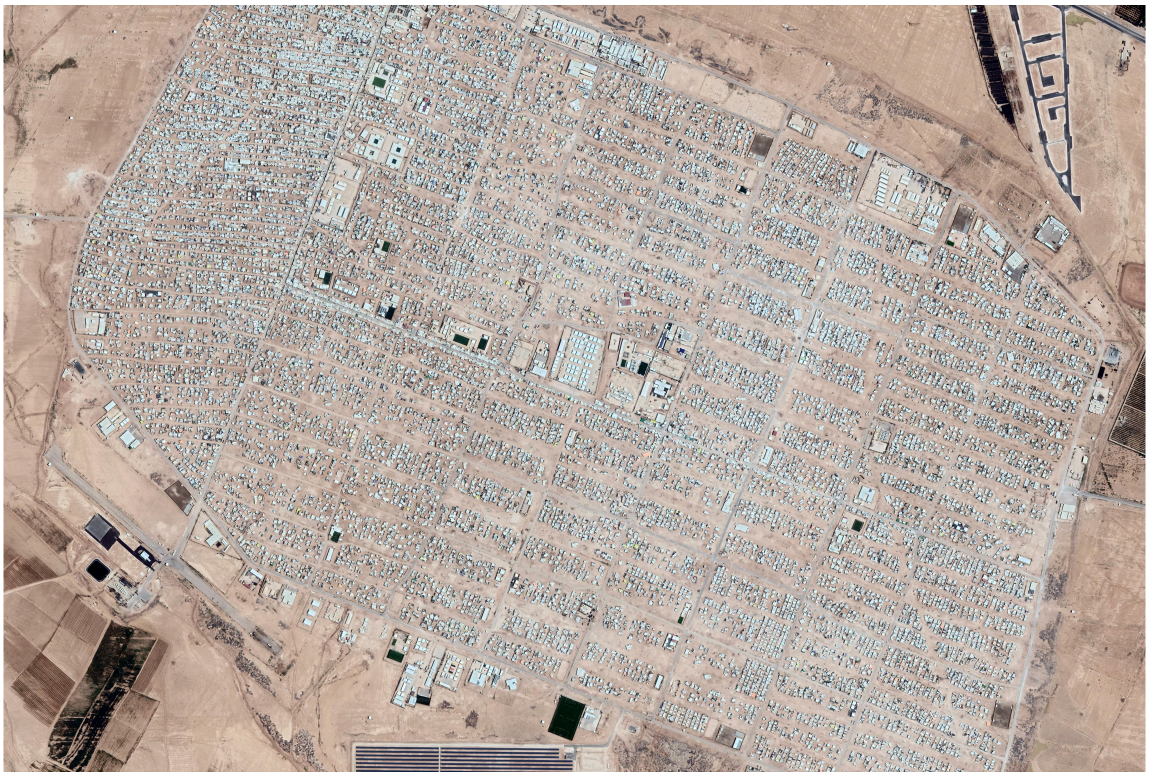


Figure A.12: Zaatari

B. Bibliography

References

Aravena Pelizari, P., Spröhnle, K., Geiß, C., Schoepfer, E., Plank, S., & Taubenböck, H. (2018, May). Multi-sensor feature fusion for very high spatial resolution built-up area extraction in temporary settlements. *Remote Sensing of Environment*, *209*, 793–807. Retrieved 2019-04-17, from `https://linkinghub.elsevier.com/retrieve/pii/S0034425718300312` doi: 10.1016/j.rse.2018.02.025

Beznec, B., Speer, M., & Mitrović, M. S. (2016). *Governing the Balkan route: Macedonia, Serbia and the European border regime*. Rosa Luxemburg Stiftung Southeast Europe.

Bjorgo, E. (2000, January). Using very high spatial resolution multispectral satellite sensor imagery to monitor refugee camps. *International Journal of Remote Sensing*, *21*(3), 611–616. Retrieved 2019-08-12, from `https://doi.org/10.1080/014311600210786` doi: 10.1080/014311600210786

Braun, A., & Hochschild, V. (2017). Potential and Limitations of Radar Remote Sensing for Humanitarian Operations. *GI_Forum*, *1*, 228–243. Retrieved 2019-04-17, from `http://hw.oeaw.ac.at?arp=0x00369d99` doi: 10.1553/giscience2017_01_s228

Caffe2. (2018, May). *Caffe2 and PyTorch join forces to create a Research + Production platform PyTorch 1.0.* Retrieved 2019-08-18, from `http://caffe2.ai/blog/2018/05/02/Caffe2_PyTorch_1_0.html`

Campbell, J. B., & Wynne, R. H. (2011). *Introduction to Remote Sensing, Fifth Edition*. Guilford Press. (Google-Books-ID: NkLmDjSS8TsC)

Cao, G., Wang, B., Xavier, H.-C., Yang, D., & Southworth, J. (2017, December). A new difference image creation method based on deep neural networks for change detection in remote-sensing images. *International Journal of Remote Sensing*, *38*(23), 7161–7175. Retrieved 2019-04-17, from `https://www.tandfonline.com/doi/full/10.1080/01431161.2017.1371861` doi: 10.1080/01431161.2017.1371861

Cheng, G., Han, J., & Lu, X. (2017, October). Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proceedings of the IEEE*, *105*(10), 1865–1883. Retrieved 2019-04-17, from `http://ieeexplore.ieee.org/document/7891544/` doi: 10.1109/JPROC.2017.2675998

Consortium, T. C. (2019). *Common Objects in Context.* Retrieved from `http://cocodataset.org/#detection-eval`

Dalal, A., Darweesh, A., Misselwitz, P., & Steigemann, A. (2018, December). Planning the Ideal Refugee Camp? A Critical Interrogation of Recent Planning Innovations in Jordan and Germany. *Urban Planning*, *3*(4), 64. Retrieved 2019-05-28, from https://www.cogitatiopress.com/urbanplanning/article/view/1726 doi: 10.17645/up.v3i4.1726

Dalal, N., & Triggs, B. (2005, June). Histograms of Oriented Gradients for Human Detection. In C. Schmid, S. Soatto, & C. Tomasi (Eds.), *International Conference on Computer Vision & Pattern Recognition (CVPR '05)* (Vol. 1, pp. 886–893). San Diego, United States: IEEE Computer Society. Retrieved from https://hal.inria.fr/inria-00548512 doi: 10.1109/CVPR.2005.177

EPSG. (2019). Retrieved 2019-08-07, from http://www.epsg.org/EPSGhome.aspx

EPSG Geodetic Parameter Registry. (2019). Retrieved 2019-08-07, from http://www.epsg-registry.org/

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010, June). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, *88*(2), 303–338. Retrieved 2019-08-22, from https://doi.org/10.1007/s11263-009-0275-4 doi: 10.1007/s11263-009-0275-4

Ghorbanzadeh, O., Tiede, D., Dabiri, Z., Sudmanns, M., & Lang, S. (2018, September). DWELLING EXTRACTION IN REFUGEE CAMPS USING CNN – FIRST EXPERIENCES AND LESSONS LEARNT. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-1*, 161–166. Retrieved 2019-04-17, from https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-1/161/2018/ doi: 10.5194/isprs-archives-XLII-1-161-2018

Giada, S., Groeve, T. D., Ehrlich, D., & Soille, P. (2003, January). Information extraction from very high resolution satellite imagery over Lukole refugee camp, Tanzania. *International Journal of Remote Sensing*, *24*(22), 4251–4266. Retrieved 2019-08-12, from https://doi.org/10.1080/0143116021000035021 doi: 10.1080/0143116021000035021

Girshick, R. (2015). Fast R-CNN. In (pp. 1440–1448). Retrieved 2019-04-17, from http://openaccess.thecvf.com/content_iccv_2015/html/Girshick_Fast_R-CNN_ICCV_2015_paper.html

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In (pp. 580–587). Retrieved 2019-04-17, from http://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. (Google-Books-ID: omivDQAAQBAJ)

Han, J., Zhang, D., Cheng, G., Liu, N., & Xu, D. (2018, January). Advanced Deep-Learning Techniques for Salient and Category-Specific Object Detection: A Survey. *IEEE Signal Processing Magazine*, *35*(1), 84–100. Retrieved 2019-09-01, from http://ieeexplore.ieee.org/document/8253582/ doi: 10.1109/MSP.2017.2749125

He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. In (pp. 2961–2969). Retrieved 2019-07-31, from http://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., & Scholkopf, B. (1998, July). Support vector machines. *IEEE Intelligent Systems and their Applications*, *13*(4), 18–28. doi: 10.1109/5254.708428

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, *160*(1), 106–154.

Joulin, A., & Paris, F. (2015). Facebook AI Research. *Learning Visual Features from Large Weakly Supervised Data*.

Katz, I. (2016). A network of camps on the way to Europe. *Forced Migration Review*(51), 17.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Landenberg, A. v. (2019). *Understanding Convolutional Neural Networks for Seismic Features by Visualization* (Master's thesis, The University of Koblenz-Landau). Retrieved from `http://publica.fraunhofer.de/dokumente/N-531812.html`

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May). Deep learning. *Nature*, *521*(7553), 436–444. Retrieved 2019-04-17, from `http://www.nature.com/articles/nature14539` doi: 10.1038/nature14539

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014* (Vol. 8693, pp. 740–755). Cham: Springer International Publishing. Retrieved 2019-04-17, from `http://link.springer.com/10.1007/978-3-319-10602-1_48` doi: 10.1007/978-3-319-10602-1_48

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91–110.

QGIS Development Team. (2009). *QGIS Geographic Information System*. Open Source Geospatial Foundation. Retrieved from `http://qgis.org`

Quinn, J. A., Nyhan, M. M., Navarro, C., Coluccia, D., Bromley, L., & Luengo-Oroz, M. (2018, September). Humanitarian applications of machine learning with remote-sensing data: review and case study in refugee settlement mapping. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *376*(2128), 20170363. Retrieved 2019-04-17, from `http://rsta.royalsocietypublishing.org/lookup/doi/10.1098/rsta.2017.0363` doi: 10.1098/rsta.2017.0363

Rasterio. (2019). Retrieved 2019-08-07, from `https://rasterio.readthedocs.io/en/stable/index.html`

Ren, S., He, K., Girshick, R., & Sun, J. (2017, June). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. Retrieved 2019-04-17, from `http://ieeexplore.ieee.org/document/7485869/` doi: 10.1109/TPAMI.2016.2577031

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Fei-Fei, L. (2015, December). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. Retrieved 2019-09-03, from https://doi.org/10.1007/s11263-015-0816-y doi: 10.1007/s11263-015-0816-y

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence* (pp. 1015–1021). Springer.

Spröhnle, K., Tiede, D., Schoepfer, E., Füreder, P., Svanberg, A., & Rost, T. (2014, September). Earth Observation-Based Dwelling Detection Approaches in a Highly Complex Refugee Camp Environment — A Comparative Study. *Remote Sensing*, *6*(10), 9277–9297. Retrieved 2019-04-17, from http://www.mdpi.com/2072-4292/6/10/9277 doi: 10.3390/rs6109277

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Sung, K.-K. (1996, March). Learning and Example Selection for Object and Pattern Detection. Retrieved 2019-08-20, from https://dspace.mit.edu/handle/1721.1/6774

Sutton, T., Dassau, O., & Sutton, M. (2009). A gentle introduction to GIS. *Chief Directorate: Spatial Planning & Information, Department of Land Affairs, Eastern Cape, South Africa*.

Tiede, D., Krafft, P., Füreder, P., & Lang, S. (2017, March). Stratified Template Matching to Support Refugee Camp Analysis in OBIA Workflows. *Remote Sensing*, *9*(4), 326. Retrieved 2019-04-17, from http://www.mdpi.com/2072-4292/9/4/326 doi: 10.3390/rs9040326

Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, *104*(2), 154–171.

UNHCR (Ed.). (2018). *Global Trends - Forced Displacement in 2018* (Tech. Rep. No. 2018). Switzerland: UNHCR. Retrieved 2019-07-30, from https://www.unhcr.org/statistics/unhcrstats/5d08d7ee7/unhcr-global-trends-2018.html

Zhang, L., Zhang, L., & Du, B. (2016, June). Deep Learning for Remote Sensing Data: A Technical Tutorial on the State of the Art. *IEEE Geoscience and Remote Sensing Magazine*, *4*(2), 22–40. Retrieved 2019-04-17, from http://ieeexplore.ieee.org/document/7486259/ doi: 10.1109/MGRS.2016.2540798

Zhu, H., Meng, F., Cai, J., & Lu, S. (2016, January). Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation. *Journal of Visual Communication and Image Representation*, *34*, 12–27. Retrieved 2019-09-01, from https://linkinghub.elsevier.com/retrieve/pii/S1047320315002035 doi: 10.1016/j.jvcir.2015.10.012

Zhu, X. X., Tuia, D., Mou, L., Xia, G.-S., Zhang, L., Xu, F., & Fraundorfer, F. (2017, December). Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources. *IEEE Geoscience and Remote Sensing Magazine*, *5*(4), 8–36. Retrieved 2019-04-17, from http://ieeexplore.ieee.org/document/8113128/ doi:

10.1109/MGRS.2017.2762307

C. Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Köln, den 09.09.2019

Unterschrift
Lorenz Wickert

TH Köln
Gustav-Heinemann-Ufer 54
50968 Köln
www.th-koeln.de

Technology
Arts Sciences
TH Köln