Flower Mapping in Grasslands with Drones and Deep Learning

Master's Thesis

Johannes Gallmann

gallmanj@student.ethz.ch

Crop Science Group
D-USYS Institute of Agricultural Sciences
ETH Zürich

Supervisor:
Dr. Helge Aasen

September 5, 2019

Acknowledgements

I would first like to thank my supervisor Dr. Helge Aasen from the crop science group in the D-USYS department at ETH Zürich. He mostly allowed me to work on my thesis independently but was always reachable when I ran into problems or needed advice. His willingness to help out with his group's drone when Agroscope's drone crashed was a life saver. I would also like to thank Prof. Achim Walter for letting me participate in the weekly group meetings of the crop science group and for giving me the opportunity to hold a presentation there. The participation at these meetings gave me great insights into the work of the group and made me better understand the frame of my thesis.

Next I would like to thank all the people from Agroscope that were involved in the thesis. These are Pascal, Bettina and Katja who helped with the data collection. Alex and Jonas who flew the drone were also crucial for this thesis. Special thanks go to Jonas not only for his piloting but also for handling all unforeseen difficulties with the drone in an uncomplicated way. Thanks also to Beatrice from Agroscope who was always reachable if advice was needed.

Finally I want to thank Prof. Roger Wattenhofer for giving me the opportunity to carry out such an interesting thesis outside the scope of his group. Working on the edge of agricultural science and computer science has been a very interesting challenge for me!

Abstract

Assessing the flower abundance in grasslands is a tedious and time consuming process. It involves laying out vegetation squares and counting the flowers by hand. This thesis investigates whether it is possible to determine the flower abundance in grasslands from drone images using a deep learning approach. The central building block is a Faster R-CNN object detection network architecture [1] which is trained and evaluated on aerial image data of five flights and two sites. The results show that the novel method developed throughout the thesis meets or exceeds the accuracy of the hand counted extrapolation method for some flowers. Besides being similarly or more accurate as well as less labor intensive, the drone based abundance determination method allows for generation of spatially explicit maps of flowers which goes beyond what can be done with the traditional abundance assessment method. The results suggest that up to a ground resolution of 5 mm / pixel, good performance can be expected for the well performing flowers, including the ones being smaller than 2 cm in diameter. While the results are very promising for some flowers, other flowers are detected poorly due to various reasons such as lack of enough training data, appearance changes due to seasonal phenology or flowers being too small to be reliably distinguishable on the aerial images.

Contents

Chapter 1

Introduction

1.1 Motivation

The service done by pollinators in farmlands is estimated to value more than 150 Billion Euros a year worldwide [2] and between 205 and 479 Million Swiss Francs in Switzerland [3]. Their declining numbers motivate many ecologists to study their interplay with the environment. This often includes the assessment of flower abundance and distribution, which is an extremely time consuming task.

In the last 10 years, rapid development in sensor technology and robotics have enhanced the capabilities of unmanned aerial vehicles (UAVs) such that today it is both technologically possible and affordable to take ultra-high spatial resolution images of large areas of grassland with UAVs. Additionally deep learning based classification methods have appeared that are able to utilize the details of that data. In this thesis the question is answered whether it is possible to get accurate information about the flower abundance and distribution in grasslands from drone images alone.

1.2 Background

1.2.1 Bees and Biodiversity

Numbers of flying insects have drastically declined in the past few decades. A study conducted in Germany reports a 75 % decline of flying insect biomass over the past 27 years [4]. Due to their importance as pollinators, these alarming findings have drawn the attention of scientists to further study the reasons behind this decline in insects. Pesticides could be a major contributor to increased mortality of honey bee colonies [5]. Agricultural intensification may play a big role as well [6]. Similarly, global warming might affect insect populations [7]. The exact causes however have not been unearthed so far. Therefore a lot of ongoing research is trying to shed light on this matter [8]. A lot of such studies about bees or biodiversity require information about flowering plants within a certain

area [9, 10].

Traditionally, information about plant abundance is acquired by counting the flowers by hand. Since counting all flowers within a large area would be infeasible due to the high workload, so called vegetation squares are distributed inside the area of interest. These vegetation squares are generally 1 by 1 meters wide. Within all such vegetation squares, the numbers of flowers of each species are counted by hand and finally these numbers are extrapolated to the size of the whole area of interest. If the positions of the squares are well chosen, this method gives a good estimate of the abundance of flowers. This thesis investigates if this process can be at least partly automated.

Such an automated approach faces several challenges. The first question that has to be answered is: Are the numbers of flowers visible on the images comparable to the numbers of flowers counted in the field by hand? The hypothesis is that equally many or less flowers are counted on the images because some flowers might be invisible since they are hidden behind other flowers, covered by grass or just too small to be distinguishable. The results in section 4.1 show that this hypothesis proves to be valid for some flowers but not for all of them. Another open question is how well a deep learning algorithm can detect flowers in overhead images with very limited resolution. The lower the drone flies, the higher the resolution gets but the covered area decreases. Flying too low also introduces a downwash onto the flowers caused by the wind of the rotors which is undesirable. Detailed results on the effects of resolution changes are documented in section 4.5.

1.2.2 Computer Vision

Computer Vision and in particular object detection in images has made major progress in the past ten years [11]. [12] states object detection to be the task to find objects in images, assign a class label to it and predict the bounding box around it. This is a very complex task for a computer to do. It basically consists of two subtasks: Firstly, finding the regions in an image that contain potential objects and secondly, assigning a class label to each such region.

For the first task, two approaches exist. The first approach is to use a sliding window approach that suggests a predefined set of bounding boxes and for each of them, the image classification algorithm is run. This approach however is very time and resource consuming because of the huge amount of possible bounding boxes. Since the introduction of MultiBox [13] in 2014, state-of-the-art object detection algorithms use convolutional neural networks (CNNs) to make region proposals where an object is likely to be located. This approach has the benefit that much fewer regions are proposed which improves prediction times by factors of magnitudes [14].

For the second task of image classification, since winning the ImageNet com-

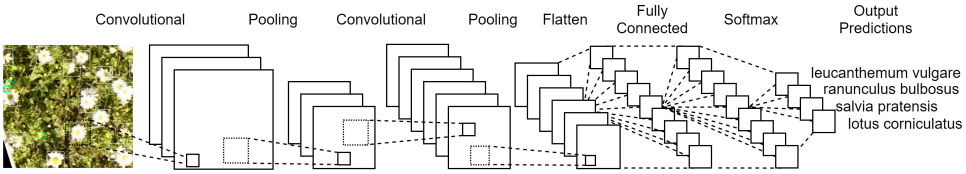


Figure 1.1: Illustration of a Convolution Neural Network. [Diagram created with www.draw.io]

petition [15] in 2012 by a large margin, the standard approach for object classification is to use a deep convolutional neural network [16]. A convolutional neural network is a network architecture with an input layer, an output layer and multiple so called hidden layers in between. For object classification, the network takes the pixels of an image as input and as output predicts the likelihood for each class label. An illustration of this architecture is depicted in Figure 1.1. The hidden layers in CNNs used for object classification are mostly convolution layers or pooling layers. In a convolution layer, a filter of a small size is moved across the image and a convolution operation is applied. In the first few layers, these filters detect low-level features such as edges in different orientations or colors. In the middle layers, more complex shapes are detected by the filters such as combinations of lower level features. In the final layers, whole objects can be detected. Pooling layers (mostly max pooling layers) are used to downsample an image. As the last layer, a fully connected layer is needed. With correctly assigned weights to all edges in the last fully connected layer, each output class can be predicted by combining the relevant features from the second last layer.

In this thesis the Faster-RCNN [1] network architecture is used. The Faster-RCNN architecture improves on the Fast-RCNN [14] network architecture by merging the region proposal network with the image classification network, which has significant speed ups as a consequence.

A convolutional neural network can be trained end to end. This means that the network can be trained by simply providing it with annotated images. With a process called back propagation the weights inside the network are fine tuned automatically such that the filters inside the different convolutional layers extract meaningful features from the images. In order to get good results from a deep convolutional neural network it has to be trained with a large amount of annotated images. This requires huge amounts of computational power. Fortunately, for the Faster-RCNN architecture, a pretrained network configuration can be downloaded from the Tensorflow Github repository. The pretrained model used in this thesis has been trained on thousands of images of the MS COCO dataset [17] and is able to extract a lot of meaningful features from images. By further training it on a custom dataset, the weights can be finetuned to this particular object detection problem.

Chapter 2

Related Work

There are many recent papers that apply advanced machine learning techniques to aerial or satellite images. For example, [18] showed good results in detecting large mammals from aerial imagery in the African savanna. The authors specifically had to deal with a large imbalance of foreground to background. [19] has achieved precision and recall values of over 90 % in detecting shrubs from satellite images.

[20] compares the performance of two different object detection network architectures on aerial images of vehicles: Faster R-CNN [1] and a modified version of YOLO [21]. The authors found that their modified version of YOLO is much faster than the Faster R-CNN architecture. However, the Faster R-CNN architecture provides slightly better results in terms of precision and recall. Similar results have been found by [22] and in a master thesis trying to detect wheat heads in overhead images [23].

Detecting objects of sizes of only a few pixels is challenging. Standard network architectures are not designed to detect such small objects by default as demonstrated by [22]. [24] analyzes the reasons behind this observation. One main cause for the poor performance is that stride sizes are too large such that small objects are highly squeezed and have few features for detection in the internal feature maps. In addition to that, small objects match only a few anchors and are therefore often left undetected which leads to low recall rates. How these problems are tackled in this thesis is described in section 3.3.4. [25] has shown good results on detecting tiny faces of only a few pixels decimeter. The main insights of that paper are that context helps detecting small faces and so does upscaling images by a factor of two. Both techniques are applied in this thesis and are further explained in section 3.3.4.

Chapter 3

Method

3.1 Overview

The main goal is to estimate the abundance of flowers within a field from drone images. To do so a deep convolutional neural network is trained to detect flowers in drone images. To train a neural network, a large amount of training data is necessary.

3.2 Data Collection

Training data in this thesis refers to images taken by a drone and flower annotations made within these images by experts. An overview of the time consuming process of collecting this training data can be obtained from figure 3.1 and is described in more detail in the following subsections.

3.2.1 Preparation and Drone Flights

Firstly, a suitable field has to be selected. In this thesis a field in Linn (canton of Aargau) and a small field in Lindau (canton of Zürich) have been chosen. The former field has been farmed extensively during the last 15 years. This means that the field has never been cut before the 15th of June in any of these years. Also no fertilizers or any other treatments are used in this field, according to the owning farmer. Due to this extensive farming, the biodiversity is very high. 40

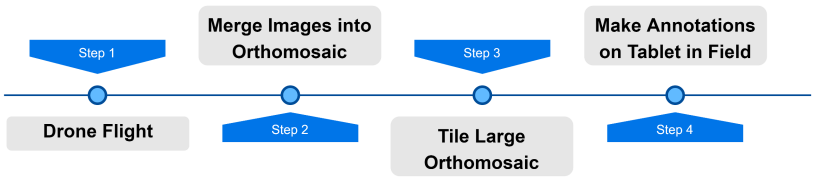


Figure 3.1: Overview of training data collection process.

(a) TransformerUAV drone (b) Vegetation Squares and GPS Marker

Figure 3.2: Photos taken in the test field in Linn.

different types of flowers have been found inside a 30 times 30 meters region of this field between May 23rd and July 3rd 2019. The 25 most common ones are listed in table 4.2.

Before flying the drone, vegetation squares as depicted in Figure 3.2b are distributed across the 30 times 30 meters test region. These vegetation squares are used for the manual counting method of the vegetation information as described in section 1.2.1. In the end, the results of the extrapolation of the number of manually counted flowers are compared to the results of the method developed in this thesis. In addition to the vegetation squares, some ground control points (GCPs) are placed randomly inside the test region. GCPs are small signs with a unique pattern facing upwards so that they can be recognized on the drone images. One such GCP can be seen in Figure 3.2b in the lower right. The exact GPS positions of all these GCPs are collected and later used in the Agisoft software [26] as described in section 3.2.2.

The next step of the training data collection process is to fly the drone over this field with a camera attached that takes aerial images of the field. In this thesis a drone model called TransformerUAV [27] owned by Agroscope is used (Figure 3.2a). Attached to it is a Sony ILCE-7RM2 [28] camera that takes 42.2 Megapixel photos in combination with a Zeiss Batis 1.8/85 telephoto lens [29]. In our experiments, the drone is flown at 19 meters above ground which yields a ground resolution of around 1.5 millimeters per pixel. A flower with diameter of 3 centimeters is therefore around 20 pixels wide on the drone image. The above mentioned drone can be programmed to automatically fly along a predefined route and during the flight take a large amount of overlapping images.

Due to problems with the software of the above mentioned drone, a DJI Matrice 600 PRO [30] is used as an alternative for the last two flights. The camera used on that drone is a Sony ILCE-9 [31] with a 24.2 Megapixel resolution. In order to have the same ground resolution, this drone is flown at 10 meters altitude.

Date	Site	Weather	Annotated Flowers	Additional Annotations	Orthophoto created	Drone
May 23rd	Linn	sunny	1524	0	No	TransformerUAV
June 6th	Linn	sunny	1125	0	No	TransformerUAV
June 14th	Linn	sunny	838	1781	Yes	TransformerUAV
June 29th	Lindau	sunny	2217	0	Yes	DJI Matrice
July 3rd	Linn	sunny	658	1968	Yes	DJI Matrice

Table 3.1: Overview of flights carried out for the thesis.

An overview of all flights is listed in figure 3.1.

3.2.2 Preparation of Images for Annotation

After the flight, the relative positions of the large amount of overlapping aerial images are reconstructed and merged together into a large orthophoto representing the whole area. A software called Agisoft [26] is used to do so. Agisoft takes all aerial images as inputs. It aligns all photos and generates a sparse point cloud model. Optionally this sparse cloud model can be improved into a dense cloud model. Finally the point cloud model can be used to build an orthomosaic. More detailed descriptions of this process can be found in [32] and [33].

For the purpose of this thesis, the sparse point cloud is sufficient and has even advantages over the dense point cloud generation. The problem of the dense point cloud generation algorithm is that it distorts certain regions in the image in order to have better alignments of the single photos. This is not desirable since detecting distorted flowers in images is not the goal. Since the dense point cloud generation is also computationally expensive and can take hours or days for a large amount of overlapping images, it is not suitable because the day after the drone flight, annotations must be made in the field.

Agisoft automatically detects the unique pattern on the GCPs to map the GPS coordinates to each of them. The advantage of providing the positions of the GCPs in the field is that Agisoft creates an image that is orthorectified and georeferenced. Georeferencing of the images is later needed to display the user's position in the android annotation application as well as to be able to copy annotations to other images that are georeferenced (c.f. 3.2.3 and 3.3 for further reading).

Once Agisoft has created the orthophoto of the whole area of interest, this orthophoto is to be loaded onto an Android tablet for the flower annotations. However, Android tablets are not capable of handling such large orthophotos (around 50000 times 50000 pixels for a 30 times 30 meters area). Therefore the orthophoto is tiled into small chunks of 256 times 256 pixels in various zoom levels. For this purpose a small image preprocessing tool has been developed with a user interface as depicted in Figure 3.3. The user can select the large

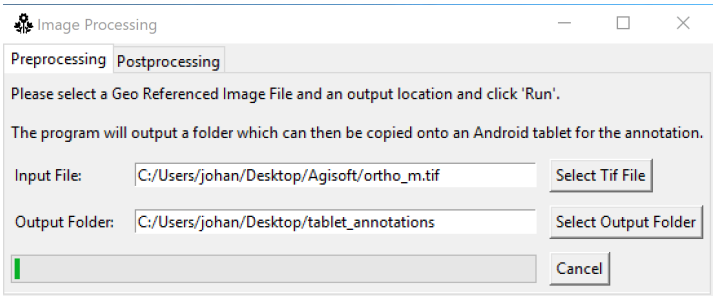


Figure 3.3: Image Preprocessing Tool used to tile large orthophotos before they are annotated on a tablet.

orthophoto created by Agisoft as well as an output folder location, where the tiled version of the orthophoto should be saved to. When clicking on 'Run', the tool will tile the orthophoto into suitable chunks utilizing the gdal library [34]. Along with the image tiles, the tool generates a small json file that contains some metadata such as the dimensions and the geo information of the upper left and the lower right corner of the original orthophoto. The tool makes sure that this geo information is in the WGS84 coordinate system that is used by the worldwide GNSS system. As an alternative, this conversion script can be executed using the command line interface introduced in section 3.3.

3.2.3 Annotations on Android Tablet

For the annotations, an Android tablet application called PhenoAnnotator has been developed. It can be downloaded from the Google Playstore. The advantage of being able to make the annotations on a tablet is that they can be made directly in the field. This is necessary because some flowers can be very hard to distinguish in the image alone. If one can compare the image to the actual flowers on site, the quality of the training data can be improved and it is made sure that the number of false annotations is minimized.

A screenshot of the main window of the annotation application can be seen in figure 3.4. The output folder created by the image preprocessing tool mentioned in the previous subsection can be copied onto the Android tablet and imported into the annotation application (1). The orthophoto is then displayed to the user. If the geo information is included in the metadata file, the user's GPS location is indicated on top of the image (2). This helps the user navigate through the field. The displayed image can be zoomed up to a level where the individual pixels are visible. If the user clicks on any location in the image, the annotation settings on the right appear. The user can select the type of flower from the list (3). Next to each flower in the list, the number of already recorded occurrences are indicated in brackets. If necessary, the position of an annotation can be fine tuned by using the four buttons on the bottom (4). The user can dismiss the annotation

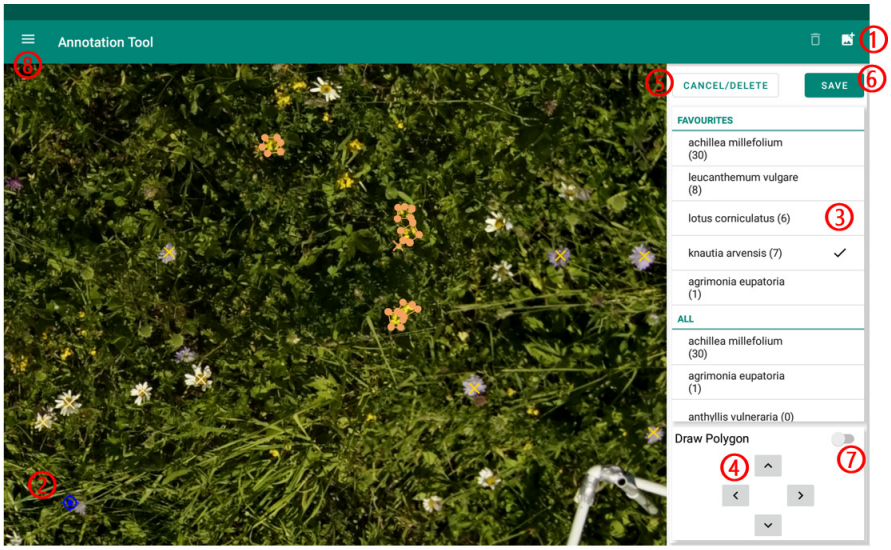


Figure 3.4: Screenshot of the main window of the PhenoAnnotator application for Android.

in processing by clicking on 'cancel/delete' (5) or save it by clicking on 'save' (6). Optionally, instead of a point annotation, a polygon can be drawn around a region. To do so, the switch at the bottom (7) has to be activated. Already saved annotations can also be edited or deleted again by simply clicking on them inside the image. By clicking on the menu button on the upper left, the settings screen can be opened. There the user can edit the list of flowers either manually or by importing a predefined list from a csv file. Also an export of the flower list to a csv file is possible. Furthermore, some zoom settings such as the maximal zoom level or the zoom level at which the annotations should be displayed to the user can be set. The application continuously saves the annotations to a json file in the project folder. The application is programmed in Kotlin.

3.3 Training a Deep Convolutional Neural Network

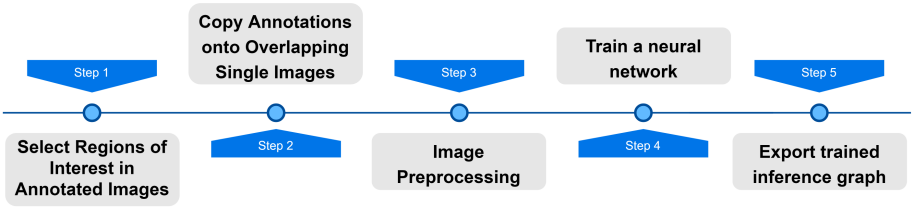
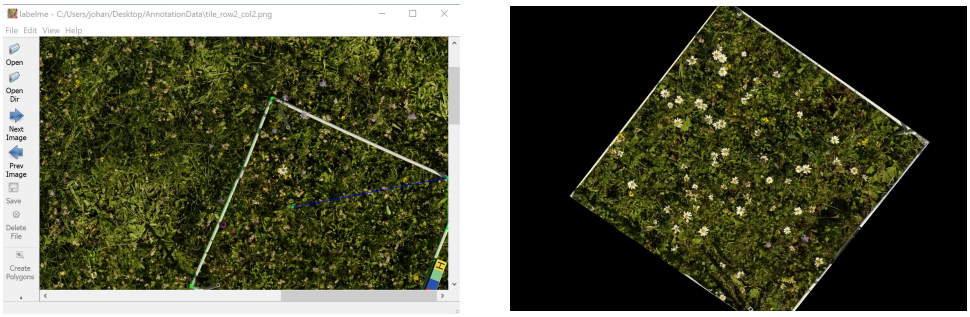


Figure 3.5: Overview of the training process.

Five iterations of the training data collection process as summarized in figure 3.1 are carried out for this thesis. An overview of all flights can be obtained from table 3.1 Before being able to actually train a neural network with this data, it needs to be preprocessed. Steps one to three of the overview in figure 3.5 are carried out before training the neural network. They are the direct continuation of the training data collection process described in the previous section. These three steps are explained in the following three subsections. A python command line interface as depicted in figure 3.7 developed during the thesis can be used to carry out each of the five steps from the overview in figure 3.5.

3.3.1 Selection of Regions of Interest in Annotated Images

Since the annotation process takes a considerable amount of time, it is not possible to annotate all flowers within the 30 times 30 meters test region for every iteration. Instead, all flowers within the 15 vegetation squares are annotated. The flower detection model should only be trained on the image data within these vegetation squares. Consequently a python script has been developed that allows the user to cut out certain regions (polygon shaped) from the images. Only the image pixels within these selected regions are kept while the rest of the image pixels are overridden with black as in figure 3.6b. This ensures that the tensorflow model does not learn non-annotated flowers as the background class. For the polygon selection, the script utilizes the labelme program [35] as depicted in figure 3.6a.



(a) LabelMe application (b) Result of the region selection script

Figure 3.6: Screenshots of the region selection script.

3.3.2 Leveraging Overlapping Images

Since the camera attached to the drone captures a large amount of highly overlapping images, the idea is to use these overlapping images as additional training data. Since the flowers are pictured from a slightly different angle on each image

and the background changes from image to image, this provides valuable additional training data. All single images can be exported as georeferenced images in Agisoft. Since the images on which the annotations have been made on are georeferenced as well, the locations of all annotations are also known within the single images. Results of this annotation copying process show almost perfect overlaps on images of 2018. When applying the same copying script to the new images collected during this thesis, the results are not satisfying. The reason is that the grass is very high in the new images which is not the case in the images of 2018. The high grass moves markedly even in weak wind. Therefore the flowers' locations are not static but move around and inevitably turn out to be different in every image. Since the copied annotations are mostly not in the right position but a few centimeters shifted, a script lets the user view and adjust all annotations in the LabelMe application. These slight adjustments of the annotations take significantly less time than annotating new data.

3.3.3 Image Conversion into Tensorflow Readable Format

The training data consisting of image files alongside with json files containing the annotations has to be converted into a format that is supported by Tensorflow [36]. Running the `python cli.py image-preprocessing` command carries out all steps necessary to prepare the data for the training.

First of all, because the Faster RCNN Network architecture takes at most 1000 times 1000 pixel images as inputs, the script tiles the images. Having 1000 times 1000 pixel tiles overwhelms the NVIDIA GeForce GTX 1080 GPU's [37] memory capacities. Therefore the default tile size is set to 450 times 450 pixels. These image tiles are then upscaled to 900 times 900 pixel tiles as suggested in [25] and justified in section 4.2.

Note that the annotations are stored as points or polygons so far. For the training of the neural network however, the annotations have to be converted to

(a) Main page of CLI (b) Example of command description

Figure 3.7: Screenshots of the command line interface developed throughout the thesis.

Ranunculus	**Lotus corniculatus**	**Galium mollugo**	**Crepis biennis**	**Centaurea jacea**
-Ranunculus bulbosus -Ranunculus friesianus -Ranunculus acris	-Lotus corniculatus -Lathyrus pratensis	-Galium mollugo -Achillea millefolium -Daucus carota -Carum carvi	-Crepis biennis -Leontodon hispidus -Tragopogon pratensis -Picris hieracioides	-Centaurea jacea -Lychnis flos cuculi

Table 3.2: Flowers that are combined into one super class.

bounding boxes. The polygon annotations are easy to convert. As the bounding box simply the surrounding rectangle is taken. If the annotation is a point, a standard size in pixels for that particular flower is taken.

The number of training examples varies heavily from class to class. For some flowers, less than 50 instances are annotated. 50 instances are very few by deep learning standards. By default all classes that have less than 50 annotated instances are not included in the training. This value can be adjusted with the `--min-instances INT` option. Some flowers are combined into groups because they have few annotated instances and they look similar to other flowers. Table 3.2 lists these combinations.

Once all training images are processed into the desired tile size, the images are split up into train, test and validation set. For each input folder the user can define, what portion of that training data should be used for testing and validation. It can also be chosen between random and deterministic splitting. Deterministic splitting is designed such that running the splitting function multiple times yields the same splitting result each time. This is useful for the comparison of different configurations of certain parameters.

The script carrying out the whole conversion takes some additional optional parameters such as `--overlap INT`. The integer parameter defines how many pixels of overlap the image tiles should have. The idea is that the flowers positioned on the edge of two tiles are not lost as training data but are always present as a whole in at least one tile. The Faster R-CNN paper states that annotations crossing the image bounds do not influence the loss function. The flag `--tile-size INT` defines the tile size to be used as Tensorflow input. It can be defined multiple times which has the consequence of multiple tile sizes being used as Tensorflow input. The idea of varying tile sizes is to obtain a more robust network that trains more on the shape of the flowers rather than the size. Note that by default each tile will be rescaled to 900 times 900 pixels before the actual training. Therefore varying the tile size has actually the consequence of the network being presented with different sizes of the flowers (with more or less resolution details).

3.3.4 Faster R-CNN Model Training

Running the `train` command in the command line interface starts the training of a model. As the underlying network model, the Faster R-CNN architecture is used. This architecture requires more compute power than other architectures but it has been shown that it performs better on aerial images than other architectures [20, 22, 23]. For the training, the default configuration of the Faster R-CNN architecture is used with a few adjustments. The `first_stage_features_stride` variable is set to the minimum of 8. Default is 16. It defines the output stride of the extracted region proposal network feature map. Additionally the `height_stride` and the `width_stride` variables of the `grid_anchor_generator` are set to the minimum of 8 as well. Default for these variables would be 16 as well. These changes slow down the training and the prediction process significantly but it is necessary because it improves the performance on small flowers that are just a few pixels wide. A smaller `first_stage_features_stride` value has the consequence that the region proposal network of the Faster R-CNN architecture outputs a feature map with a higher resolution. The `height_stride` and the `width_stride` variables control the distance in pixels of two consecutive anchors. An anchor is a location within the image from which various sizes of possible bounding boxes to be evaluated are spanned. Having a large distance between two such anchors might cause the network to miss flowers that are placed in between two such anchors. These changes are suggested in the Faster R-CNN paper itself [1] and specifically for small objects in [24].

In contrast, the default anchor size of 256 pixels is not changed. This might seem surprising because there is no flower that is 256 pixels in diameter. However changing this default configuration slightly decreases the accuracy of the model, as the results in section 4.2 confirm. As mentioned in section 2, a paper trying to detect tiny faces [25] has reported similar results. It is likely that more context does also help in detecting flowers just as it helps to detect faces.

Training can be carried out in two modes. Mode one trains the network with a fixed number of steps and a preset schedule at which steps to reduce the learning rate. Mode two uses the validation set to decide when to change the learning rate and when to stop training. The advantage of the first method is that no valuable training data is lost to the validation set and it is slightly faster since no resources are spent on evaluating the performance on the validation data. The advantage of the second approach is that the best time to stop training does not have to be guessed but can be decided based on the prediction performance on the validation set.

In mode two, every 2500 steps the training is paused and the prediction followed by the evaluation algorithm is run on the validation set. The learning rate is adjusted if for the last 15000 steps no further improvements were made. After adjusting the learning rate two times from 3×10^{-4} to 3×10^{-5} and from 3×10^{-5} to 3×10^{-6}, the training is stopped if for 15000 steps again no im-

provement on the performance has been made. The number of 15000 steps is chosen empirically. Reducing the learning rate twice by a factor of 10 is directly adapted from the Faster R-CNN default configuration. The evaluation metric can be chosen as either the f1 score or the mean average precision (mAP). Section 3.4 further explains the prediction and evaluation process.

The number of training examples varies heavily from class to class. Therefore each flower is assigned a weight. The weight is inversely proportional to the number of training examples and influences the loss function during training. This ensures that the network does not just optimize to detect the most common flowers. Each mistake on a less common flower has a much higher penalty to the loss function as a consequence.

Once a network is fully trained it can be exported as an inference graph with the `export-inference-graph` command of the command line tool. This exported inference graph is then used by the prediction and evaluation scripts described in section 3.4.

3.4 Usage of Trained Network

3.4.1 Predictions

Once a network is trained and the trained inference graph is exported, it can be used to make predictions on unseen aerial imagery. The overview of the prediction process can be obtained from figure 3.8. The first two steps are identical to the training data collection process. The drone has to be flown over a field and the aerial images have to be merged together with Agisoft. Once these two steps are done, the region of interest has to be selected from the resulting orthophoto. This is done in the same manner as it is done for the region selection during the training process described in the last section and visualized in figure 3.6. Because the LabelMe application is not able to handle large orthophotos, the script decreases the resolution of the image before forwarding it to the LabelMe application if necessary. Once the user has selected the region of interest of the decreased resolution orthophoto, the script applies this selection to the original orthophoto to remove the pixels that are not of interest.

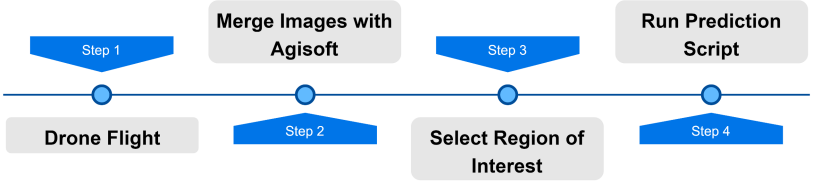


Figure 3.8: Overview of prediction process.

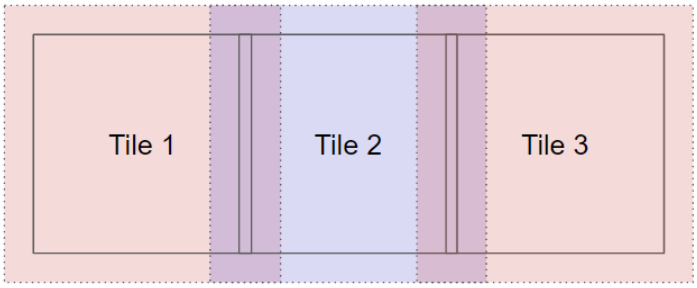


Figure 3.9: Illustration how the images are tiled for the prediction process.

Having a large image containing only the region of interest after these first three steps, the prediction python script can be run. It draws the bounding boxes of all found flowers onto the image and saves the statistics about the flower abundance within the image to a json file. As in the training process, the image is split up into tiles similar as it is done in [38]. By default 450 times 450 pixel tiles are used which are scaled up to 900 times 900 pixel tiles before the prediction algorithm is run on them. If a tile consists of only one color, the tile is skipped without running the prediction algorithm.

To improve the prediction accuracy, the tiles have an overlap of 100 pixels by default. This ensures that as long as a flower is not larger than 100 pixels in diameter, it is fully visible on at least one tile of 450 times 450 pixels. Figure 3.9 illustrates how exactly the tiling is done. Each dotted square corresponds to one tile. Each tile has a core which is marked by the solid line. First of all, the predictions close to the edges of a tile are discarded. These are the regions outside the core of a tile. In these regions the predictions are often error prone because flowers might be only partly visible. All predicted bounding boxes whose centers are within the core of a tile are kept for the final result. As can be seen in the illustration, also these core regions have a small overlap. The problem without this overlap is that a flower located at the border of two cores could be discarded in both adjacent tiles because in both tiles the center of the predicted bounding box could be slightly shifted towards the other tile. The overlap of the cores ensures that no predictions are lost for this reason. Having this overlap introduces a new problem of duplicate predictions. This is mitigated by applying non maximum suppression with an intersection-over-union threshold of 0.3. Meaning that for all predictions that have an overlap of more than 30 %, only the one with the highest confidence score is kept. 30 % might seem as a small intersection area, however empirical tests have shown that using 0.3 achieves the best results. This is in line with [38], where 0.25 is used as a threshold.

3.4.2 Evaluations

To evaluate the performance of a model, the predictions on the test set are compared to the ground truth of the test set. The main metrics of interest are precision and recall. To compute precision and recall values, the true positive (TP), false positive (FP) and false negative (FN) predictions have to be known. In order to obtain these values the following algorithm is carried out:

Firstly the predictions are sorted by their confidence. Then it is looped through all the predictions and for each of them it is compared to all ground truth bounding boxes of the same label. To compare two bounding boxes the intersection-over-union (*iou*) formula is used:

$$iou = \frac{intersection\ area}{area\ of\ union}$$

If the greatest *iou* value is greater than some threshold (default of 0.3), the corresponding ground truth box is marked as used and the prediction is market as true positive. If the largest *iou* value is lower than the threshold, the prediction is marked as false positive. After this process is done for each prediction, all ground truth entries that are not marked as used are counted as false negatives. Having the TP, FP and FN numbers, the precision and recall values can easily be calculated using the following formulas:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

A good way to rate the performance of a model is to compute the F1 score. The F1 score is calculated as follows:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The better the precision and recall values are, the better the F1 score gets. It rates precision and recall equally and reaches its maximum of 1 at perfect precision and recall. As an alternative to the F1 score, the mean average precision (mAP) as defined in the PASCAL VOC Challenge Development Kit [39] can be used to rate a model's performance. The `evaluate` command of the command line interface computes all mentioned metrics. Optionally it also computes the confusion matrix.

3.4.3 Visualizations

Several options for visualizations exist in the command line tool. Firstly, the `visualize` command which takes an input folder and an output folder as arguments paints all annotations onto the images in the input folder and saves the modified images to the output folder. The command understands all annotation formats used throughout the thesis. These can be the json files created by the android app, json files created by LabelMe application or the xml files created during the `image-preprocessing` command. This makes the command a handy tool for confirming the annotations at any stage of the process.

As a second option for visualization, the `--visualize-predictions` and `--visualize-groundtruth` flags of the `predict` command can be set to `True`. In addition to the json files containing the predictions and groundtruth annotations, the script then generates copies of the input images and draws the bounding boxes onto them. A similar option named `generate-visualizations` exists for the `evaluate` command. It paints all erroneous predictions onto the input images. This includes the FN and FP predictions. False negatives are painted pink, FP are painted red. If there is a FP and a FN at the same location, this means that there is a confusion between two flowers which causes the script to paint an orange bounding box in this case.

The third option is to generate heatmaps from predictions with a command called `generate-heatmaps`. Given an image and the corresponding json file containing the predictions, it draws a heatmap onto a copy of the image which indicates the distribution and density of the flowers. The heatmaps can be generated for individual flowers or for all flowers. If the input images are georeferenced, there is the option to generate one heatmap from a collection of images. If the images are overlapping, the heatmap indicates the average number of flowers found at a particular position. A handy feature is the `--window` flag. The user can provide the coordinates (in the swiss coordinate system CH1903+ / LV95 [40]) of the upper left and lower right corner of the desired output region. The script will then output a heatmap of exactly that region. This allows for time series generations. Example results of such time series generations can be viewed in section 4.4. Comparisons between heatmaps generated from predictions on an orthophoto and heatmaps generated from predictions of single overlapping images are discussed in the same section.

Chapter 4

Results

4.1 Manual Counting vs Drone Image Based Tablet Annotations

Since the exact same areas are annotated on the tablet as they are counted by hand, we are able to directly compare the numbers of flowers annotated on the drone images to the numbers of flowers counted by hand. Table 4.1 lists a subset of all flowers found within the test fields. Some interesting observations can be made from it.

To begin with, some flowers are hardly visible on the drone images and therefore significantly less instances are counted in the tablet annotations compared to the hand counted data. Onobrychis viciifolia, medicago lupulina and to some extent trifolium pratense fall under this category. The blooms of medicago lupulina are simply too small to be reliably identifiable on the drone images. In the case of trifolium pratense and onobrychis viciifolia, the blooms would be big enough but often they are hardly distinguishable from the background. Refer to table 4.2 for visualizations of 25 flowers found within the test fields. For many flowers (leucanthenum vulgare, ranunculus, knautia arvensis and centaurea jacea) there are more flowers annotated on the tablet than counted by hand.

Flower	**Hand Counted**	**Tablet Annotations**
leucanthemum vulgare	724	960
onobrychis viciifolia	483	105
lotus corniculatus	1943	748
salvia pratensis	142	127
ranunculus	431	474
knautia arvensis	371	471
trifolium pratense	129	72
medicago lupulina	117	5
centaurea jacea	25	28

Table 4.1: Comparison of hand counted results to tablet annotations.

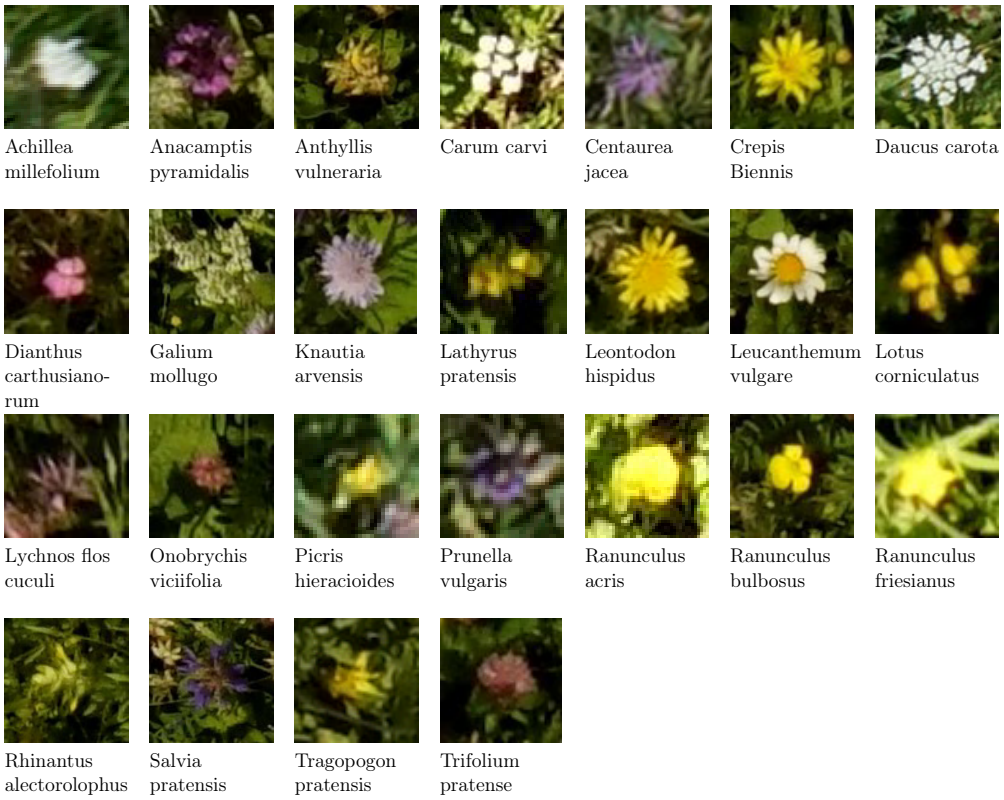


Table 4.2: Excerpts from aerial images of the most common flowers.

It should be noted that it is expected that the two data sets have slight differences due to the fact that the hand counted data has always been collected one day after the flight. In theory, the flower populations could have changed during that time. However, the impact of the time shift should be negligible.

4.2 Parameter Tuning

Using the default configuration of the Faster R-CNN architecture does not generate satisfying results. Table 4.3 shows the effects of certain parameter choices. For each configuration in table 4.3, a network is trained and evaluated. Each configuration is trained and evaluated on the same data. All results are weighted average values across all flower species meaning that all TP, FP and FN values are summed up across all flower species and then the precision, recall and f1 score are calculated using these summed up values. All predictions having a confidence score below 0.2 are ignored for the evaluation. This threshold is chosen empirically.

There is no standard train/test/validation set splitting rule in object detection

problems. The MS COCO 2017 Detection Challenge [17] uses 25 % of the images for testing and 3 % for validating while the Pascal Voc Challenge 2012 [41] uses around one fourth of the images for testing and validating each. The ILSVRC 2017 Challenge [15] uses 4 % of the images for validating and 8 % for testing. All these mentioned public challenges have massive amounts of data available. Having significantly less data available, the decision is made to use 70 % of the images of each flight for training, 20 % for testing and 10 % for validating for the experiments in this section. This splitting strategy ensures that the majority of the data is used for training but still large enough portions are available for validation and testing.

As the first column of table 4.3 shows, using the original image size mainly hurts the recall metric. The low recall value indicates that many flowers are not detected by the trained network. Upscaling the images mitigates this problem as can be seen from the other columns of table 4.3. In all configurations marked with *Upscaled Image*, the training data is tiled into 450 times 450 pixel tiles. Each such tile is then scaled up to a 900 times 900 pixel tile. The drawback is that this slows down the training and prediction process.

The effect of adding a train overlap is not clear. While the recall, f1 score and mAP slightly drop compared to the configuration with only upscaled images (column 3 vs. column 2 in table 4.3), in combination with data augmentation options, the performance of these three metrics is better with a train overlap (column 5 vs column 4). The opposite happens with the precision metric. The idea of a train overlap is to not loose any training data because of the tiling process. The Faster R-CNN architecture ignores all cross boundary bounding boxes during training.

Using data augmentation techniques and a train overlap within the training pipeline results in the best performance in terms of recall, f1 score and mAP. The precision metric is only insignificantly lower than in other configurations. The following data augmentation options are used: random horizontal and vertical flips, random brightness adjustments, random contrast adjustments, random saturation adjustments and random box jittering. In all further experiments in the

	Original Image	Upscaled Image						Various Image Scales	
Train Overlap			Yes	Yes		Yes	Yes	Yes	Yes
Data Augmentation				Yes	Yes	Yes	Yes	Yes	Yes
Anchor Size 128						Yes			
Stride 16							Yes		
Multiple Scales									Yes
Precision	81.7 %	82.1 %	82.3 %	84.7 %	85.2 %	85.2 %	86.1 %	85.7 %	73.4 %
Recall	79.5 %	83.9 %	81.8 %	**84.3** %	83.8 %	83.6 %	81.7 %	76.6 %	70.9 %
F1 Score	0.806	0.830	0.820	**0.845**	0.845	0.844	0.838	0.809	0.721
mAP	0.575	0.668	0.634	**0.686**	0.680	0.671	0.619	0.588	0.485

Table 4.3: Performance of various parameter configurations.

subsequent sections the configuration with data augmentation, train overlap and a minimum confidence score of 0.2 is used.

Three more configurations can be found in table 4.3. The fourth last column contains the results with the base anchor size set to 128 pixels instead of default value of 256. Recall, F1 score and mAP are slightly worse than with a base anchor size of 256 pixels. This might seem surprising because none of the flowers have a size of 256 pixels. The authors of [25] have found that context helps to detect tiny faces. This is likely to be true for tiny flowers as well. Nevertheless, the effect is not very significant. The third last column shows the performance with the `first_stage_features_stride`, `height_stride` and `width_stride` parameters set to 16. As hypothesized in section 3.3.4, the performance is worse than with these values set to the minimum of 8. The recall value is notably lower than in the configuration with the stride values set to 8. This is an indication that some flowers are missed by the prediction network due to the lower coverage of anchors and the lower resolution of the feature map. In the second last experiment, the training images are tiled into stripes of various tile sizes (225, 300, 450 and 600 square pixels). The smaller the tile size the more tiles are generated of that size such that the flowers are more or less equally distributed among the various tile sizes. Each tile is again upscaled to 900 times 900 pixels and therefore the network presented with variable flower sizes. The idea is that the network should learn to detect flowers based on their shape and color rather than their size. Running the prediction algorithm with 450 square pixel tiles however shows that the recall value decreases compared to fixed sized tiles.

In the last experiment, all images are tiled into many tile sizes (300, 350, 400, 450, 500, 550, 600) and all tiles are used as training images. Therefore each flower is presented in various sizes to the network. This experiment resulted in very bad performance on the test images. Namely 73.4 % precision and 70.9 % recall. When investigating the results it can be seen that for many flowers there are multiple predictions from which all but the one with the highest score are discarded by the non maximum suppression function. Quite often the prediction with the highest score is not the correct one. Especially flowers of the same color such as crepis biennis, lotus corniculatus and ranunculus are frequently confused. For these flowers the size of the bloom seems to be an important feature learned by the network. On the other hand, the prediction performance of leucanthemum vulgare drops only marginally with a network trained on images with various tile sizes. Since there are no flowers present in the two test sites with a similar appearance, the size of the flower is not important to the prediction network. Appendix A.3 contains some examples images of predictions.

4.3 Application to Grassland Dataset

The idea of the experiments in this section is to simulate an as realistic as possible situation. The data of the flight on June 14th is not used for training and validation but is only used as test data. 90 % of the data of all other flights is used as training data and 10 % as validation data. The decision of taking the data of the flight on June 14th as the test set is made based on the fact that all flowers present in the field on June 14th are present in at least one other data set, either in the flights before or after. It would be even better to apply a trained model to an unseen field. An obvious choice would be to use the field in Lindau as test set. However, since the vegetation of that field is substantially less diverse than the vegetation in Linn, this option discarded.

4.3.1 Performance inside Vegetation Squares

The prediction performance for each flower can be obtained from table 4.4. A prediction is considered if its confidence is greater than 0.2. With the overall precision and recall being 87 % and 84.2 % respectively, the results are better than in the previous section. The vast majority of the flowers present in the test data of June 14th are knautia arvensis, leucanthemum vulgare and lotus corniculatus. These three flowers perform well and therefore the good overall score is mainly determined by these three flowers. All the other flowers perform worse than the overall performance indicates.

Table 4.5 shows the confusion matrix of this experiment. It is striking that

Flower	**Train Instances**	**Test Instances**	**Precision**	**Recall**	**mAP**	**F1 Score**
anthyllis v.	196	6	20.0 %	16.7 %	0.056	0.182
centaurea j.	742	53	73.0 %	50.9 %	0.382	0.6
crepis b.	124	21	36.8 %	66.7 %	0.325	0.475
dianthus c.	20	34	100.0 %	23.5 %	0.235	0.381
galium m.	546	16	19.5 %	50.0 %	0.1	0.281
knautia a.	429	438	89.6 %	94.1 %	0.879	0.918
leucanthemum v.	928	1022	96.5 %	88.6 %	0.861	0.924
lotus c.	2153	1026	87.4 %	85.5 %	0.772	0.864
onobrychis v.	92	95	77.6 %	47.4 %	0.407	0.588
rhinanthus a.	23	26	61.5 %	30.8 %	0.218	0.41
salvia p.	133	15	50.0 %	80.0 %	0.436	0.615
trifolium p.	109	7	9.5 %	57.1 %	0.104	0.163
Overall	5495	2759	87.0 %	84.2 %	0.398	0.855

Table 4.4: Performance of the prediction algorithm on all flower species present in the field on June 14th. The numbers in the *Train Instances* and *Test Instances* columns refer to the ground truth annotations. The overall scores of the performance metrics are weighted means.

	Anthyllis v.	Centaurea j.	Crepis b.	Dianthus c.	Galium m.	Knautia a.	Leucanthemum v.	Lotus c.	Onobrychis v.	Prunella v.	Ranunculus	Rhinanthus a.	Salvia p.	Trifolium p.	False Negatives
Anthyllis v.	1	-	-	-	-	-	-	3	-	-	-	-	-	-	2
Centaurea j.	-	27	-	-	-	17	-	-	-	1	-	-	3	3	2
Crepis b.	-	-	14	-	-	-	-	5	-	-	-	-	-	-	2
Dianthus c.	-	3	-	8	-	1	-	-	10	-	-	-	-	6	6
Galium m.	-	-	-	-	8	-	-	-	-	-	-	-	-	-	8
Knautia a.	-	-	-	-	-	412	1	-	-	-	-	-	2	-	23
Leucanthemum v.	-	-	1	-	1	4	906	-	-	-	-	-	1	-	109
Lotus c.	-	-	6	-	-	-	1	877	-	-	-	-	-	-	142
Onobrychis v.	-	1	-	-	-	1	-	-	45	-	-	-	-	11	37
Prunella v.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Ranunculus	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Rhinanthus a.	-	-	-	-	-	-	-	1	-	-	-	8	-	-	17
Salvia p.	-	-	-	-	-	-	-	-	-	-	-	-	12	-	3
Trifolium p.	-	-	-	-	-	1	-	-	-	-	-	-	-	4	2
False Positives	4	6	17	-	32	24	31	117	3	-	1	5	6	18	-

Table 4.5: The table shows the confusion matrix. The columns represent what the model predicted and the rows represent what the model should have predicted (the ground truth). The green, red, orange and brown numbers denote TP, FP, FN and confusions between two flowers respectively.

there are only a few confusions between different flowers (brown). The much more common cases are that flowers are predicted where there are none (red) and flowers are not predicted where they should be (orange). The green entries denote the correctly predicted flowers. Section A.1 in appendix A contains the same confusion matrix but with relative values.

During the training phase each class is assigned a weight which is inversely proportional to the number of instances present in the training data. This is to make sure that the network does not only optimize on the frequent flowers. Table 4.4 shows that the flowers with little training data tend to not perform well. The question is whether this is due to the lack of enough training data or because assigning a weight to each class is not sufficient to regularize the loss function. Therefore a seperate network is trained in which the three best performing flowers (leucanthemum vulgare, lotus corniculatus and knautia arvensis) are ignored and treated as background. With the mAP rising from 25.2 % to 31.5 % (f1 score improves from 47 % to 51.3 %) a certain improvement can be seen but the performance is still significantly below what is satisfactory. Therefore the possibility of leveraging two separately trained networks is not futher evaluated.

When looking at the predictions, there are various sources of errors apparent. Some examples can be seen in table 4.6. For leucanthemum vulgare, a typical

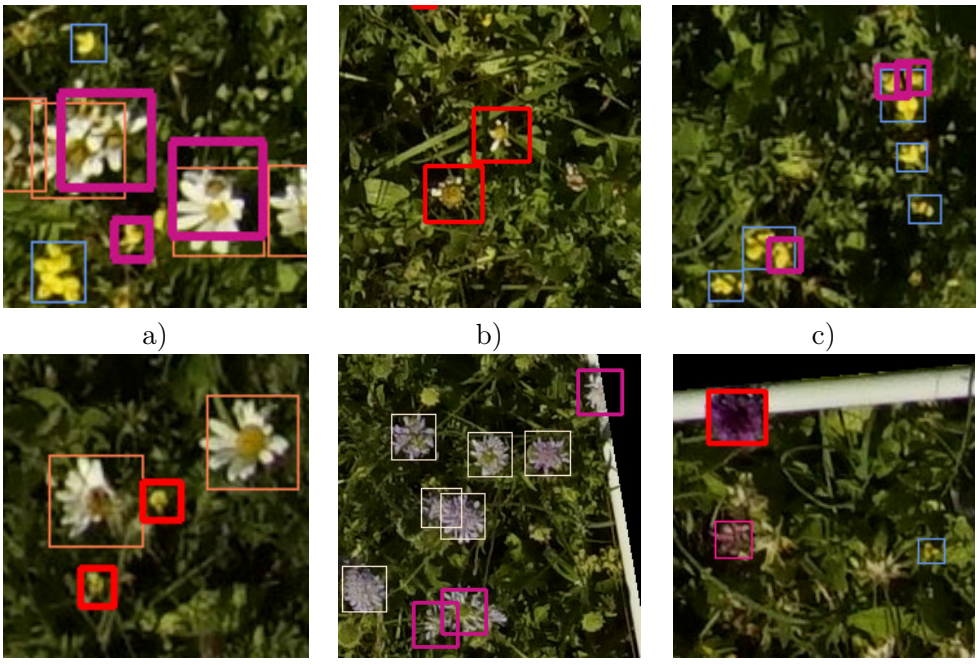


Table 4.6: Selection of typical mispredictions. All thin bounding boxes are correct predictions. The bold red bounding boxes denote false positive and the bold violet bounding boxes denote false negative predictions. There are various explanations for the mispredictions: Overlapping flowers (a), partially faded flowers (b and e), collections of flowers (c), missing ground truth annotations (d) and flowers that are missing in the training data (f).

error occurs when two instances are very close to each other as in image a). In that case often only one of the two flowers is detected. The missing annotation is not caused by the non maximum suppression algorithm as a closer look discloses. Another typical source of errors are flowers that are on the verge of fading. In the case of image b) two flowers are detected that are not annotated in the ground truth because the expert considered the flowers to be faded already. Even when counting the flowers by hand it is sometimes difficult to decide if a flower should be counted or not because of the seamless transition from blooming to faded. Two main problems exist for lotus corniculatus. Firstly, the blooms of lotus corniculatus are often arranged as small collections of blooms as visible in image a) in the bottom left or in image c). In some cases the network predicts the blooms of a collection as individual instances while in the ground truth the whole collection is annotated as one instance. The opposite case is common as well. The second problem of lotus corniculatus are false positive predictions caused by missing ground truth annotations (as in image d)). These problems are further discussed in section 5.0.1. The main error source of knautia arvensis is again

blooms that look different because they are wilting as for example in image e). In image f) the model erroneously predicts a knautia arvensis where there is a anacamptis pyramidalis. Anacamptis pyramidalis is not included in the training because too few training instances exist.

4.3.2 Performance outside Vegetation Squares

It is difficult to make statements about the performance of the model on the full test field since there are no ground truth annotations present outside the annotated vegetation squares. The only possibility is to compare the predictions of the model to the extrapolations of the manually counted flowers. The numbers of manually counted flowers are extrapolated to the size of the whole field which is 730 square meters. Table 4.7 lists all flowers that perform reasonably well inside the vegetation squares. For each flower species the number of detections in the whole field is listed as well as the number of flowers predicted by the extrapolation of the manual counting.

For centaurea jacea, knautia arvensis and lotus corniculatus the number of drone based predictions is very similar to the extrapolation of the manually counted number of flowers. The results are within 11 %, 3 % and 2 % respectively. According to heatmaps generated from the drone based predictions (c.f. section 4.4), these are also the flowers that have a relatively even distribution. The extrapolation of the manually counted number of leucanthemum vulgare is 53 % higher than the number of drone based predictions. The question is, which prediction is more accurate. Assuming that the performance of the prediction algorithm is similar on the whole field as it is inside the annotated vegetation squares, the extrapolation of the manual counting must be inaccurate. Even when adding 8 % to the number of drone based predictions to compensate for the low recall value of leucanthemum vulgare, the results still have a 47 % gap.

Flower	**Drone Based Prediction**	**Extrapolation of Manual Counting**	**Relative Difference**
centaurea jacea	456	505	10.7 %
knautia arvensis	8059	8308	3.1 %
leucanthemum vulgare	7044	10778	53.0 %
lotus corniculatus	50365*	51139	1.5 %
onobrychis viciifolia	595	3761	532 %
salvia pratensis	209	673	222 %

Table 4.7: Predictions on the whole field of 730 square meters. The 50365 predicted lotus corniculatus are calculated as the multiplicative of the actual predictions of the network (19389) and a ratio of 2.6. The numbers in table 4.1 suggest that there are 2.6 blooms per prediction on average.

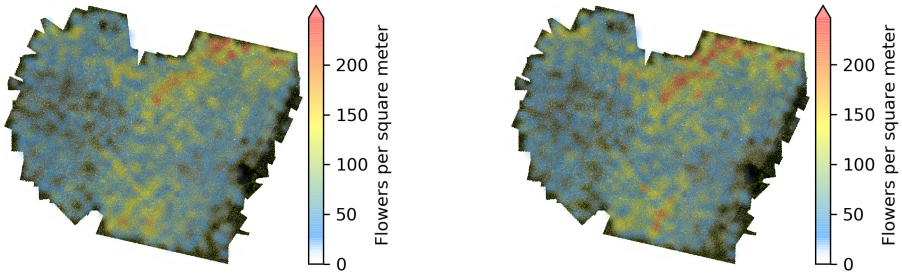
To make matters worse, the extrapolation is based on the manually counted number of flowers which is lower than the number of tablet based annotations within the vegetation squares as pointed out in section 4.1. If the tablet based numbers were taken, the result of the extrapolation would be an additional 51 % higher making them a total of 131 % higher than the drone based predictions.

The main reason for the bad results of onobrychis viciifolia is that it is very hard to distinguish on the drone images. The most probable reason for the unsatisfactory results of salvia pratensis is that the amount of training data is too low to accomplish good results. A likely additional reason is an unrepresentative choice of vegetation square locations for these flowers.

4.4 Density Distribution Maps

Figure 4.1 shows the predicted density distribution of all flowers in the field in Linn on June 14th. The image in figure 4.1a is generated from the predictions on the orthophoto covering the whole area. The image in figure 4.1b is generated from the overlapping orthorectified and georeferenced single images. It is apparent that the results are very similar. The total amount of flowers predicted on the orthophoto is 2.8 % lower than the averaged number of predicted flowers on the single images. In the case of leucanthemum vulgare it is 1.4 % more predictions on the ortho image, for knautia arvensis and lotus corniculatus it 2.5 % and 3.4 % less respectively. These results suggest that there is not much accuracy to gain with a lot of overlapping images. The only advantage is that per heatmap pixel an averaged value can be calculated which smoothes false predictions. The main disadvantage of predictions on the overlapping single images is that it takes substantially more compute time. In addition to the heatmap in figure 4.1a illustrating the overall abundance of all predicted flowers, the heatmaps in table 4.8 depict the abundance of some selected individual flowers in the field in Linn on June 14th. The three heatmaps for leucanthemum vulgare, lotus corniculatus and knautia arvensis are generated from the ortho photo. The last heatmap shows the single image coverage of the field in Linn.

Table 4.9 contains a time series of an excerpt of the field in Linn. It illustrates the difference of the abundance evolution of leucanthemum vulgare and lotus corniculatus. It is conspicuous that lotus corniculatus is much more evenly distributed than leucanthemum vulgare. While leucanthemum vulgare has a peak population on June 6th and on July 3rd the population is almost completely faded, the peak population of lotus corniculatus is much less pronounced.



(a) Generated from ortho image. (b) Generated from overlapping images

Figure 4.1: Density distribution maps of all flowers on June 14th.

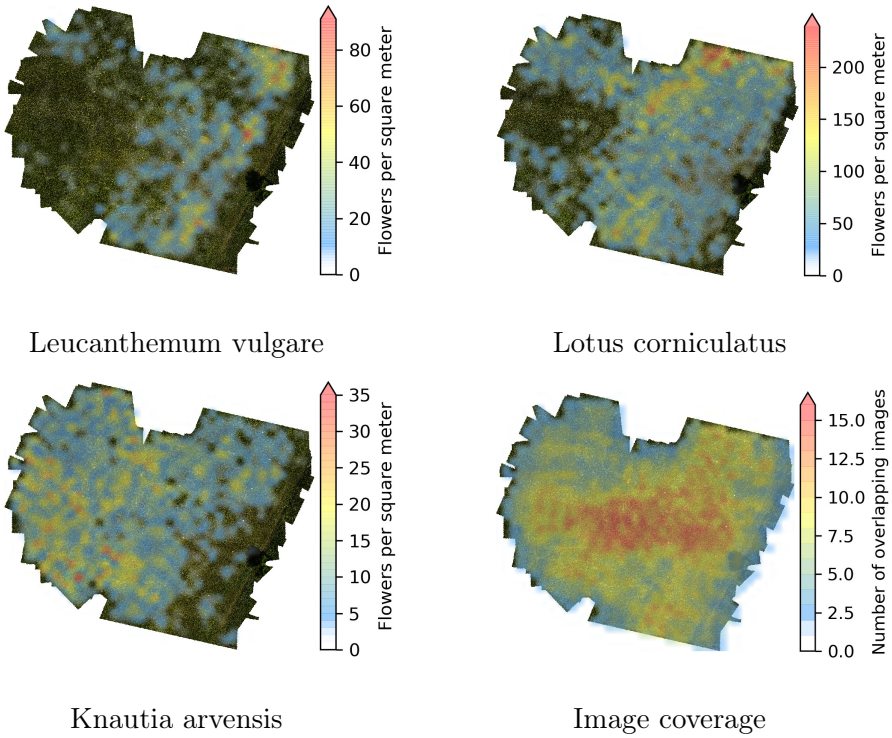


Table 4.8: Heatmaps of the field in Linn for various flower species. The last image depicts the image coverage of the field.

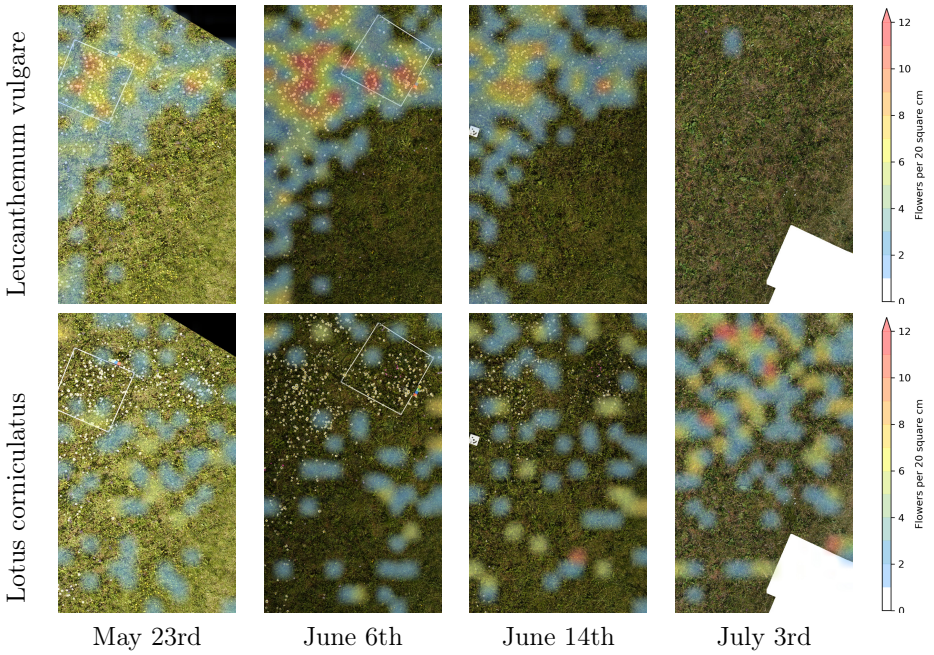


Table 4.9: Timeseries of the distribution of leucanthemum vulgare and lotus corniculatus in the field in Linn.

4.5 Impact of Image Spatial Resolution

4.5.1 Predictions on Simulated Resolutions

The higher the drone can fly the more area can be covered with a single drone flight. Table 4.10 demonstrates the effect of decreasing ground resolution on an example excerpt of an aerial image containing a leucanthemum vulgare flower and a lotus corniculatus collection of blooms. Figure 4.2a and 4.2b illustrate the effect of decreasing ground resolution on the F1 score and the mAP respectively. Both figures show that down to a ground resolution of 5 mm per pixel, there is just a marginal decrease in prediction performance. Further decreasing the ground resolution to 10 mm and 20 mm per pixel however has noticeable

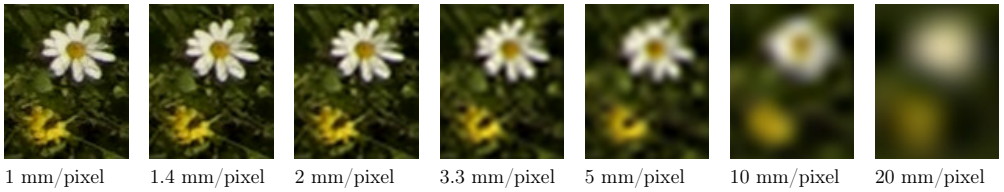
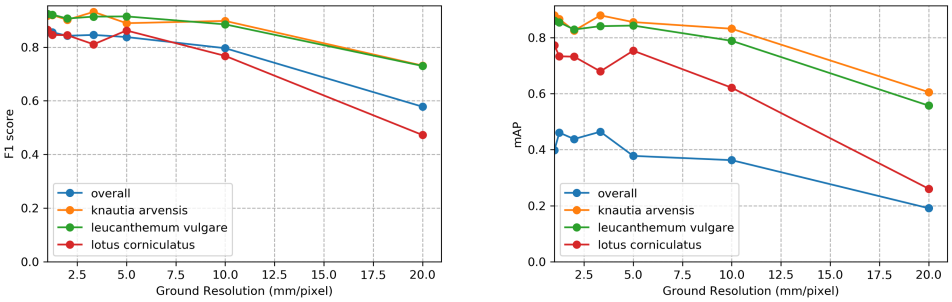


Table 4.10: Resolution degradation on an except of an aerial image.



(a) Evolution of F1 score.

(b) Evolution of mAP.

Figure 4.2: Evolution of performance metrics over various simulated ground resolutions.

negative effects on the model's performance. As expected, the performance of small flowers such as lotus corniculatus decreases disproportionately because at a certain ground resolution they simply get indistinguishable. The average size of a lotus corniculatus bloom is around 16 mm. The performance of larger flowers such as leucanthemum vulgare (40 mm) and knautia arvensis (34 mm) degrades notably slower. The graphs for the precision and recall metrics are omitted since the trends are equivalent to the trends of the F1 score and the mAP metric.

Each training, test and validation image is first scaled down to the desired ground resolution and then scaled up again. After upscaling, all datasets have the same ground sampling distance (pixel size) as the original images again. This ensures that the flower's sizes (in image pixels) are large enough to be detectable by the faster R-CNN network architecture and prevents performance losses caused by this problem as described in [25]. For each ground resolution a network is trained and evaluated with the processed training images. In an additional experiment, the resolution degradation is only applied to the test images but not to the training and validation images. The results of this experiment can be found in appendix A in section A.2.

4.5.2 Predictions on Low Resolution Flight

The results in the previous subsection suggest that a model trained and evaluated on imagery with 2 mm/pixel ground resolution performs similarly well as with 1 mm/pixel ground resolution. The question is whether this holds true when predicting on flight data that is originally captured with 2 mm/pixel ground resolution. There are images of three flights on June 29th in Lindau. The first flight is captured from 20 meters height with a ground resolution of around 2 mm/pixel. The second and third flight are captured from 11 meters altitude and have a ground resolution of around 1 mm/pixel. The image data of the second

	Ground Truth (n)	Prediction on Flight 2 (1 mm/pixel)	Prediction on flight 3 (1 mm/pixel)	Prediction on flight 3 (simulated 2 mm/pixel)	Prediction on flight 1 (2 mm/pixel)
achillea millefolium	201	+1 %	+8 %	-14 %	-10 %
centaurea jacea	587	-3 %	+6 %	+12 %	-16 %
centaurea jacea faded	1116	+10 %	+1 %	-29 %	-82 %
lotus corniculatus	292	+6 %	-33 %	-35 %	-40 %
leucanthemum vulgare	15	-27 %	-27 %	-27 %	-72 %

Table 4.11: Prediction result variation for flights with different resolutions in Eschikon, Lindau. The image data of flight 2 is included in the training and validation data.

flight is annotated and included in the training/validation data. It is listed in the *Ground Truth* column of table 4.11.

When using the model trained for the experiments in section 4.3 for predictions on flight two of June 29th, the results are very good as the numbers in table 4.11 show. This is not surprising since these images are included in the training data that is used to train the model. Also when predicting on flight 3, which has 1 mm/pixel ground resolution as well, the numbers of predicted flowers are very similar to the ground truth numbers except for lotus corniculatus which has 33 % less predictions.

The last two columns of table 4.11 are predicted with the model trained on the simulated 2 mm/pixel ground resolution. When scaling down the image of flight three by a factor of two and running the prediction algorithm, the accuracy further drops but except for faded centaurea jacea not remarkably. The results on the flight data which is captured with a 2 mm/pixel ground resolution are considerably worse.

4.6 Time Measurements

In this section several time measurements are listed to give an idea about the compute power needed for the different steps in the work process.

The duration of the image preprocessing script is mainly determined by the amount of annotated input images. Another factor is the speed of the internet connection which is used to download the pretrained model from the tensorflow website. Preprocessing 410 mainly large (larger than 25 Megapixel) annotated images containing around 8500 annotations in total takes around 20 minutes. Two minutes of which are needed to download the pretrained Faster R-CNN model configuration.

Training a model is the most time consuming process. On the computer architecture used in the thesis, it takes roughly a day to completely train a well

performing model. Although, already after a few hours the model performance is within a few percent of the final model performance. Predicting the flower abundance in the ortho image of June 14th takes 90 minutes on a GeForce GTX 1080 GPU [37]. The area covered on that ortho image is around 730 square meters. Therefore prediction time for one square meter is around 7.4 seconds.

Generating heatmaps from predictions of a single image is quick. In contrast, generating heatmaps from multiple overlapping images can take a considerable amount of time. It depends on several factors. The prominent one is the number of single images. The more single images are used to generate the heatmap, the longer it takes. The second factor is the resolution of the heatmap. High resolution heatmaps take longer because for every single heatmap pixel the coverage has to be determined. Generating a heatmap from 228 single images and a heatmap resolution of 44 times 40 pixels takes around four minutes. Generating a heatmap from the same 228 single images but with a heatmap resolution of 1000 times 910 pixels takes 20 minutes.

Chapter 5

Discussion

5.0.1 Flower Counting

Section 4.1 shows that for some flowers there are more flowers annotated on the tablet than are counted by hand. This should not be the case and can only be explained by the fact that counting flowers by hand is a very tedious process. Mistakes happen very easily if a lot of flowers are present within a small area. Annotating on an image on a tablet has the advantage that flowers are marked and therefore the risk of counting twice or forgetting to count a flower is minimized. When combining these falsely counted numbers with non optimally chosen vegetation square locations, the extrapolations of the manually counted flowers have the potential to be very inaccurate. This finding is a big motivation for this thesis since with a reliable flower detection model, the results can be much more accurate than with the extrapolation from the manual counting. Moreover, the drone based approach has other advantages. The potential to have spatially explicit maps of flowers goes beyond what can be done with the traditional approach of extrapolating the manually counted numbers of flowers within the vegetation squares. Also the workload is considerably lower once a trained and reliable network is available.

Whether it is possible to get reliable predictions for a certain flower depends on several factors. First, enough training data of the flower in question needs to be available. The results suggest that with a few hundred instances good performance can be achieved. Second, also the morphology of the flower has an impact. Flowers such as galium mollugo are difficult for an object detection network to predict reliably. The cause seems to be that this flower can sometimes be very small and in other cases multiple instances of the same flower cover a large area in which it is difficult to separate the single instances. In such cases it would be interesting to see how an image segmentation network such as Mask R-CNN [42], which predicts regions (pixels) that belong to a certain class, would perform. Third, the size of a flower should span a certain minimum amount of pixels. The results in section 4.5.1 state that with a 5 mm/pixel ground resolution still good performance can be achieved for lotus corniculatus. This corresponds to a flower diameter of around three to four pixels. The good results of lotus

corniculatus are likely to be caused by its distinct color and the strong contrast to the background. Other flowers of similar size such as onobrychis viciifolia or trifolium pratense perform significantly worse. These flowers are much harder to distinguish from the background. It is evident that distinguishability / contrast is the fourth main factor which determines the prediction performance of a network for a particular flower.

When taking a closer look at the results, a substantial portion of mispredictions that negatively influences the model performance scores such as mAP and F1 score is not fatal. This includes for example false positives that are in fact missing annotations in the ground truth such as the examples in table 4.6. False positive predictions of flowers that are on the verge of fading fall under this category as well. The mispredictions caused by the confusion between single blooms and collections of blooms of lotus corniculatus as described in section 4.3.1 are not severe either. If such mispredictions were ignored, the performance scores would be better.

These mispredictions exemplify the challenges that exist for the training data collection. Even when being able to directly compare the image on the tablet to the flowers on site, it is sometimes not clear how to annotate a flower. Lotus corniculatus is a good example. They are often arranged as collections of blooms. It is not uncommon however that there are single blooms that do not belong to a collection. Since it is often not possible to distinguish the single blooms within a collection, the whole collection is annotated as one instance. Unfortunately there are border cases in which a single bloom very close to a collection is annotated as a separate instance in the ground truth but the prediction algorithm includes that flower in the collection and predicts only one bounding box. This results in false negative predictions for the single blooms very close to the collection as the examples in image c) in table 4.6 show. The opposite case that single blooms are predicted which are annotated as a collection is common as well. The second big problem for lotus corniculatus is that some instances are hardly visible on the images because they are very small. Sometimes they are partly hidden by other vegetation and sometimes weak motion blurr is present which makes it even harder to distinguish between flower and background. This problem also manifests itself in false positive and false negative predictions. False positive predictions are mainly caused by background areas that look similar to a blurred bloom and real blooms which are not present in the ground truth annotations (as in image d)). The false negative predictions are often blooms that are small and hardly distinguishable.

The issue of two overlapping blooms of leucanthemum vulgare being predicted as one instance as in figure 4.6 a) could possibly be prevented by annotating flowers with more accurate bounding boxes in the training data. The vast majority of leucanthemum vulgare flowers is annotated with a dot in the center and the standard size bounding box is used for training the network. For two overlapping

flowers, each bounding box covers a large portion of the other flower as well. Therefore the network learns to predict such overlapping flowers as one instance. Furthermore, by using a standard size bounding box, the network learns to predict exactly this size of bounding box whereas the bounding boxes of overlapping flowers should often be smaller. The network trained on various tile sizes seems to predict overlapping flowers more reliably. The reason is that the network has learned various bounding box sizes for the same flower and is therefore more robust. Unfortunately, that network configuration has other drawbacks as described in section 4.2. In a nutshell, there is a tradeoff between accuracy and expenditure of time. Annotating flowers with polygons or bounding boxes of various sizes takes considerably more time than annotating the flowers with a single dot in the center. In cases of very high density of flowers, the extra effort is likely to pay off with an increased accuracy.

Due to malfunctioning of the drone's software, there are no overlapping images available for the first two flights on May 23rd and June 6th. If the drone's software worked from the beginning, there would be substantially more training data available from the overlapping images.

For the bee studies carried out by Agroscope the number of single blooms is of interest. A lot of flowers are arranged and predicted as collections of blooms however. As demonstrated on the example of lotus corniculatus in section 4.3.2, it seems to be a good idea to calculate an average number of blooms per annotation from the hand counted data and multiply the prediction result with that ratio to get the number of single blooms. Alternatively, an average number of blooms per pixel could be calculated from the training data and hand counted data. This value can then be multiplied with the total amount of pixels inside the bounding boxes of the predictions to get the number of blooms. This technique has the advantage that the bounding box sizes influence the predicted number of blooms. This approach would be more robust if different fields have different average amounts of blooms per collection.

5.0.2 Influences of the network configuration and image resolution

The prediction results on the data with 2 mm/pixel ground resolution in section 4.5.2 are unsatisfactory. Especially because the results on a simulated 2 mm/pixel ground resolution are much better and comparable to the results with 1 mm/pixel ground resolution. It is difficult to identify the cause for this bad performance. One reason might be that in the artificially scaled images, there is more information contained than in the images captured at a higher flight level. To create the artificial 2 mm/pixel resolution a lanczos filter is used. According to [43] a lanczos filter offers the best compromise in terms of reduction of aliasing, sharpness, and minimal ringing compared to other image scaling techniques.

The more likely reason however is that the image data of the 2 mm/pixel flight is slightly blurred and not perfectly focused.

As documented in [25] it is advised to scale images up to improve the performance of a network. The paper suggests to scale up all images with objects that are smaller than 40 pixels in diameter by a factor of two. This is the case for the vast majority of flowers dealt with in this thesis. The Faster R-CNN architecture is not designed to detect very small objects such as flowers of just a few pixels in decimeter [22, 24]. Therefore scaling up the images is an appropriate counter measure which helped to improve the results in this thesis.

Data Augmentation options are a convenient way of artificially increasing the amount of training data. One should be careful with applying too many augmentation options. Since the flowers do not span a large number of pixels, they are predicted based on minuscule details. Changing these details too much might be counterproductive. Flips and random box jittering can be applied without hesitation. They do not alter the important details but only the orientation or the position of the bounding box. Brightness, contrast and saturation adjustment should be applied moderately. In this thesis the maximal change is a delta of 25 %.

5.0.3 Practical considerations

The main test field in Linn is around 30 times 30 meters large. In order to have enough overlapping images to generate an orthophoto of this area, a drone has to fly over the field for about 20 minutes. This means that it is difficult to scale this approach to larger areas. A way of overcoming this problem is to take sample pictures at random locations of a larger field. Knowing the flight height and the lens angle of the camera, one can calculate the covered area of the image. Running the prediction algorithm on these sample images and extrapolating the numbers of predictions to the size of the whole field can still achieve very good results. The advantage over the manual counting flower abundance determination approach is that a much larger sample size can be collected in very little time and effort. What remains to be evaluated is whether the prediction algorithm generates similar results close to the edges of an image compared to the center. The viewing angle close to the edges is different than in the center of an image. Consequently there could be a degradation in prediction performance. The orthophotos are created only from the center regions of the single images.

Throughout the thesis various metrics are used to describe a model's performance. Precision, recall, F1 score and mAP all describe a certain aspect of a model's performance. It depends on the application case, which metric is most important. If one tries to find a weed or invasive species within a field, it might be better to predict too many than too few instances. In a precision agriculture setting, the drone might directly forward the image information to a ground robot

which navigates to the locations of the predicted weeds, validates the drone based predictions and precisely sprays a minimal amount of pesticide onto the weed. In this case a high recall value is desirable. Precision would not be very important because the ground robot validates the drone based predictions with more detailed information from close up. It is more relevant that as many instances are detected as possible. In other cases it might not be necessary to detect as many plants as possible but detecting one or a few plants is sufficient. In this occasion a high precision is advantageous in order to minimize the occurrences of false alarms (false positives). A possible example could be the detection of jacobea vulgaris. Jacobea vulgaris is a weed that is unwanted by farmers due to its poisonous nature. It is advised to fight this weed uncompromisingly once it is found within a field [44]. The seeds of jacobea vulgaris can fly large distances and therefore it can show up anywhere. A drone could help to detect this weed in new locations. Because the plants have to be fought carefully by hand, it is not necessary for the drone to detect all instances but just detecting the presence of such a plant within a certain area would be helpful. It could give a farmer the hint that he should check if an intervention is necessary in that area. Since false alarms would be annoying, a high precision is desirable in such a scenario.

Precision and recall can easily be controlled with the minimum confidence parameter. The higher the minimum confidence parameter of the prediction script is set, the higher the precision gets. Lowering the minimum confidence score increases the recall. For the abundance determination use case as studied in this thesis a balanced precision/recall ratio is advantageous, because false negatives and false positives are likely to cancel each other out and therefore a good estimate of the abundance can be given. The F1 score is mainly determined by precision and recall. The higher these two values are, the higher is the F1 score. A balanced ratio of precision and recall rewards the score even more. Consequently, the F1 score is a good indicator of a model's performance.

In none of the experiments any sort of cross validation is used. This is due to the fact that training a model takes a lot of time and compute power. Time and computational resource restrictions have made it impossible to include cross validation into the experiments. Some spikes in figure 4.2 might be caused by the lack of cross validation. Nonetheless, independent train, evaluation and test sets are used. Therefore the results are not expected to be remarkably different with other data splits. Apart from that, all aerial imagery collected for this thesis is captured in sunny weather conditions. Future work should investigate the effects of other weather conditions (e.g. cloudy diffuse) on the prediction performance.

The method developed in this thesis opens a wide range of imaginable use cases beyond the substitution of manual flower counting. Weed control could be realized as in the two examples given above. The multitemporal abundance maps have the potential to map flowering dynamics quantitatively and spatially assess co-occurrence of different flowers. By detecting certain indicator species, conclu-

sions may be drawn about the soil properties. The presence of leucanthemum vulgare for example is an indicator for lean meadows. In the context of quality assessment of meadows in connection with direct payments by the state or canton, drone usage is imaginable. Detecting invasive neophyte plants in difficult-to-access terrain such as reed terrain could replace manual checks. For some use cases it might be beneficial to have real time detections. The method developed in this thesis is not designed for that. By using the default configuration of the Faster R-CNN architecture without upscaling the images, the prediction algorithm can be sped up by a factor of four at least. The drawback is that the accuracy is lower. Nevertheless, for some use cases this might be acceptable. Using a more light weight object detection network design such as the SSD architecture [45] can deliver further speed ups. However, the accuracy is expected to be lower than with Faster R-CNN.

The suite of tools developed throughout the thesis is easy to install (c.f. appendix B.1) and can be applied to any sort of object detection problem on aerial images. The time consuming task of training data collection by annotating aerial images can be carried out on the PhenoAnnotator application for Android or with the widely used LabelMe application for desktop operating systems. The script that copies annotations onto overlapping images can be a powerful way of increasing the amount of training data without major efforts.

During the thesis, several lessons have been learned. Choosing a test field that is a fourty minute drive away from the Agroscope center in Reckenholz has not been optimal. In a few occasions a closer location would have been more efficient. Second, if from the beginning the more popular and well-engineered DJI drone was used, the troubles caused by the TransformerUAV drone could have been prevented. More training data would be available as well as orthophotos of all four flight dates. Third, using a Linux operating system from the beginning would have prevented several complications with the Tensorflow object detection framework. Tensorflow is widely used on Linux and therefore it is likely to be better optimized and documented for the Linux operating system. Finally, implementing the possibility to draw bounding box annotations within the PhenoAnnotator application in addition to point and polygon annotations would have been a good idea. As mentioned above, more accurate bounding boxes could have been annotated this way and the model performance could have possibly been improved for some flowers.

Figure 5.1: Typical prediction example.

Figure 5.2: Aerial photograph of the test site in Linn.

Chapter 6

Conclusion

To conclude, the thesis shows that there is great potential in combining very high-resolution UAV remote sensing with object detection in an ecological context. It has been shown that determining the abundance of flowers in grasslands can be done with drones and object detection if certain requirements are fulfilled. The most important requirement is that a lot of training data is available. Furthermore, the flowers have to be distinguishable on the RGB images meaning that they should be large enough to spread across multiple pixels on an image. Last but not least, the flowers should not be a plant that expands over large areas in which the individual instances are difficult to keep apart.

If these requirements are fulfilled, the drone based abundance determination approach described in this thesis achieves similar results as the extrapolations of the manual flower counting and for some flowers significantly improves the accuracy of the abundance prediction. The drone based approach minimizes the risk of mispredictions caused by inadequately placed vegetation squares or mistakes during the manual counting of the flowers.

While for some flowers the results are very good, for others this is not the case. Therefore this approach is not suitable to determine the complete flower abundance in grasslands with a rich bio diversity but only part of it. This could be improved in the future with more training data and a higher ground sampling distance. Apart from that, the thesis opens the door to a broad range of imaginable use cases. The whole work process described in the thesis can be applied to any other aerial object detection task. All tools are well documented and easy to install and reuse. (See installation instructions in appendix B.1 and user manual in appendix C.)

Bibliography

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv e-prints*, p. arXiv:1506.01497, Jun 2015.

[2] N. Gallai, J.-M. Salles, J. Settele, and B. E. Vaissière, "Economic valuation of the vulnerability of world agriculture confronted with pollinator decline," *Ecological Economics*, vol. 68, no. 3, pp. 810 – 821, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921800908002942

[3] L. Sutter, F. Herzog, V. Dietemann, J.-D. Charrière, and M. Albrecht, "Nachfrage, angebot und wert der insekten-bestäubung in der schweizer landwirtschaft," *Agrarforschung Schweiz*, vol. 8, no. 9, pp. 332–339, 2017.

[4] C. A. Hallmann, M. Sorg, E. Jongejans, H. Siepel, N. Hofland, H. Schwan, W. Stenmans, A. Müller, H. Sumser, T. Hörren *et al.*, "More than 75 percent decline over 27 years in total flying insect biomass in protected areas," *PloS one*, vol. 12, no. 10, p. e0185809, 2017.

[5] J. S. Pettis, J. Johnson, G. Dively *et al.*, "Pesticide exposure in honey bees results in increased levels of the gut pathogen nosema," *Naturwissenschaften*, vol. 99, no. 2, pp. 153–158, 2012.

[6] F. Hendrickx, J.-P. MAELFAIT, W. Van Wingerden, O. Schweiger, M. Speelmans, S. Aviron, I. Augenstein, R. Billeter, D. Bailey, R. Bukacek *et al.*, "How landscape structure, land-use intensity and habitat diversity affect components of total arthropod diversity in agricultural landscapes," *Journal of Applied Ecology*, vol. 44, no. 2, pp. 340–351, 2007.

[7] R. Menéndez, "How are insects responding to global warming?" *Tijdschrift voor Entomologie*, vol. 150, no. 2, p. 355, 2007.

[8] F. Sánchez-Bayo and K. A. Wyckhuys, "Worldwide decline of the entomofauna: A review of its drivers," *Biological conservation*, vol. 232, pp. 8–27, 2019.

[9] L. Sutter, P. Jeanneret, A. M. Bartual, G. Bocci, and M. Albrecht, "Enhancing plant diversity in agricultural landscapes promotes both rare bees and dominant crop-pollinating bees through complementary increase in key floral resources," *Journal of applied ecology*, vol. 54, no. 6, pp. 1856–1864, 2017.

[10] L. M. Blackmore and D. Goulson, "Evaluating the effectiveness of wildflower seed mixes for boosting floral diversity and bumblebee and hoverfly abundance in urban areas," *Insect Conservation and Diversity*, vol. 7, no. 5, pp. 480–484, 2014.

[11] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, 2019.

[12] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[13] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 2155–2162. [Online]. Available: https://doi.org/10.1109/CVPR.2014.276

[14] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[18] B. Kellenberger, D. Marcos, and D. Tuia, "Detecting mammals in uav images: Best practices to address a substantially imbalanced dataset with deep learning," *Remote Sensing of Environment*, vol. 216, pp. 139 – 153, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425718303067

[19] E. Guirado, S. Tabik, D. Alcaraz-Segura, J. Cabello, and F. Herrera, "Deep-learning versus obia for scattered shrub detection with google earth imagery: Ziziphus lotus as case study," *Remote Sensing*, vol. 9, no. 12, 2017. [Online]. Available: http://www.mdpi.com/2072-4292/9/12/1220

[20] J. Carlet and B. Abayowa, "Fast vehicle detection in aerial imagery," *arXiv preprint arXiv:1709.08666*, 2017.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[22] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7310–7311.

[23] K. Keller, "Deep Learning Object Detection for Wheat Head Estimation Using RGB Images," Master's thesis, ETH Zürich, 2019.

[24] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li, "S^3fd: Single shot scale-invariant face detector," *CoRR*, vol. abs/1708.05237, 2017. [Online]. Available: http://arxiv.org/abs/1708.05237

[25] P. Hu and D. Ramanan, "Finding tiny faces," *CoRR*, vol. abs/1612.04402, 2016. [Online]. Available: http://arxiv.org/abs/1612.04402

[26] L. Agisoft, "Agisoft metashape user manual," *Professional Edition, Version 1.5*, vol. 1, p. 71, 2019. [Online]. Available: https://www.agisoft.com/pdf/metashape-pro_1_5_en.pdf

[27] C. GmbH, "Transformeruav," https://copting.de/index.php?option=com_quix&view=page&id=12&Itemid=629&lang=en, 2017.

[28] S. Corporation, "Sony ilce-7rm2 user manual," https://www.sony.com/electronics/support/res/manuals/W001/W0014549M.pdf, 2015.

[29] C. Z. AG, "Zeiss batis 1.8/85, technische daten/technical specifications," https://www.zeiss.com/content/dam/camera-lenses/files/service/download-center/datasheets/batis-lenses/datasheet-zeiss-batis-1885.pdf, 2017.

[30] L. SZ DJI Technology Co., "Dji matrice 600 pro user manual," https://dl.djicdn.com/downloads/m600%20pro/20180417/Matrice_600_Pro_User_Manual_v1.0_EN.pdf, 2018.

[31] S. Corporation, "Sony ilce-9 user manual," https://www.sony.com/electronics/support/res/manuals/d830/9e418e206969550a21e784c254221119/d8301001M.pdf, 2017.

[32] H. Aasen, "The acquisition of hyperspectral digital surface models of crops from uav snapshot cameras," Ph.D. dissertation, Universität zu Köln, 2016.

[33] S. Harwin and A. Lucieer, "Assessing the accuracy of georeferenced point clouds produced via multi-view stereopsis from unmanned aerial vehicle (uav) imagery," *Remote Sensing*, vol. 4, no. 6, pp. 1573–1599, 2012.

[34] GDAL/OGR contributors, *GDAL/OGR Geospatial Data Abstraction software Library*, Open Source Geospatial Foundation, 2019. [Online]. Available: https://gdal.org

[35] K. Wada, "labelme: Image Polygonal Annotation with Python," https://github.com/wkentaro/labelme, 2016.

[36] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[37] *NVIDIA GeForce GTX 1080*, Nvidia Corporation, 2016. [Online]. Available: https://www.nvidia.com/content/geforce-gtx/GTX_1080_User_Guide.pdf

[38] F. Ozge Unel, B. O. Ozkalayci, and C. Cigla, "The power of tiling for small object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[39] M. Everingham and J. Winn, "The pascal visual object classes challenge 2012 (voc2012) development kit," *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 2011.

[40] *Formulas and constants for the calculation of the Swiss conformal cylindrical projection and for the transformation between coordinate systems*, Federal Office of Topography swisstopo, 2016. [Online]. Available: https://www.swisstopo.admin.ch/content/swisstopo-internet/en/online/calculation-services/_jcr_content/contentPar/tabs/items/documents_publicatio/tabPar/downloadlist/downloadItems/20_1467104436749.download/refsys_e.pdf

[41] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[42] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[43] K. Turkowski, "Filters for common resampling tasks," in *Graphics gems*. Academic Press Professional, Inc., 1990, pp. 147–165.

[44] A. Bosshard, J. Joshi, A. Lüscher, and U. Schaffner, "Jakobs-und andere kreuzkraut-arten: eine standortbestimmung," *Agrarforschung*, vol. 10, no. 6, pp. 231–235, 2003.

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

Appendix A

Additional Results

A.1 Confusion Matrices

In this section the same confusion matrices are displayed as the one in table 4.5 but with relative values. Table A.1 focuses on recall. Therefore all values are percentage values relative to the number of ground truth annotations. Table A.2 focuses on precision and therefore all numbers are percentage values relative to the number of predictions made by the network for a specific flower.

	Anthyllis v.	Centaurea j.	Crepis b.	Dianthus c.	Galium m.	Knautia a.	Leucanthemum v.	Lotus c.	Onobrychis v.	Prunella v.	Ranunculus	Rhinanthus a.	Salvia p.	Trifolium p.	False Negatives
Anthyllis v.	16.7	-	-	-	-	-	-	50.0	-	-	-	-	-	-	33.3
Centaurea j.	-	50.9	-	-	-	32.1	-	-	-	1.9	-	-	5.7	5.7	3.8
crepis b.	-	-	66.7	-	-	-	-	23.8	-	-	-	-	-	-	9.5
Dianthus c.	-	8.8	-	23.5	-	2.9	-	-	29.4	-	-	-	-	17.6	17.6
Galium m.	-	-	-	-	50.0	-	-	-	-	-	-	-	-	-	50.0
Knautia a.	-	-	-	-	-	94.1	0.2	-	-	-	-	-	0.5	-	5.3
Leucanthemum v.	-	-	0.1	-	0.1	0.4	88.6	-	-	-	-	-	0.1	-	10.7
Lotus c.	-	-	0.6	-	-	-	0.1	85.5	-	-	-	-	-	-	13.8
Onobrychis v.	-	1.1	-	-	-	1.1	-	-	47.4	-	-	-	-	11.6	38.9
Prunella v.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Ranunculus	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Rhinanthus a.	-	-	-	-	-	-	-	3.8	-	-	-	30.8	-	-	65.4
Salvia p.	-	-	-	-	-	-	-	-	-	-	-	-	80.0	-	20.0
Trifolium p.	-	-	-	-	-	14.3	-	-	-	-	-	-	-	57.1	28.6

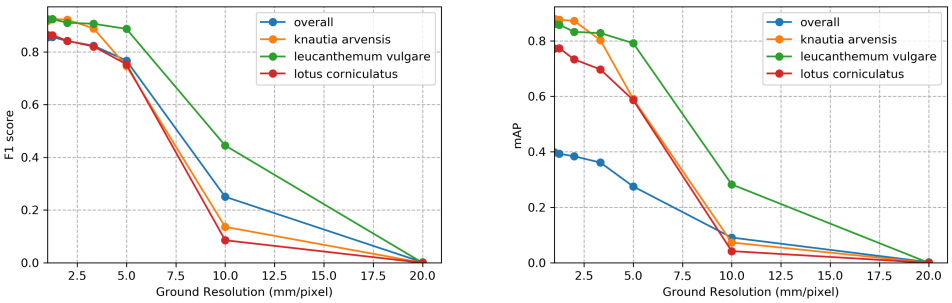
Table A.1: The table shows the confusion matrix. The columns represent what the model predicted and the rows represent what the model should have predicted (the ground truth). The green, orange and brown numbers denote TP, FN and confusions between two flowers respectively. The numbers are relative to the number of ground truth annotations of a specific flower.

	Anthyllis v.	Centaurea j.	Crepis b.	Dianthus c.	Galium m.	Knautia a.	Leucanthemum v.	Lotus c.	Onobrychis v.	Prunella v.	Ranunculus	Rhinanthus a.	Salvia p.	Trifolium p.
Anthyllis v.	20.0	-	-	-	-	-	-	0.3	-	-	-	-	-	-
Centaurea j.	-	73.0	-	-	-	3.7	-	-	-	100	-	-	12.5	7.1
crepis b.	-	-	36.8	-	-	-	-	0.5	-	-	-	-	-	-
Dianthus c.	-	8.1	-	100	-	0.2	-	-	17.2	-	-	-	-	14.3
Galium m.	-	-	-	-	19.5	-	-	-	-	-	-	-	-	-
Knautia a.	-	-	-	-	-	89.6	0.1	-	-	-	-	-	8.3	-
Leucanthemum v.	-	-	2.6	-	2.4	0.9	96.5	-	-	-	-	-	4.2	-
Lotus c.	-	-	15.8	-	-	-	0.1	87.4	-	-	-	-	-	-
Onobrychis v.	-	2.7	-	-	-	0.2	-	-	77.6	-	-	-	-	26.2
Prunella v.	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Ranunculus	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Rhinanthus a.	-	-	-	-	-	-	-	0.1	-	-	-	61.5	-	-
Salvia p.	-	-	-	-	-	-	-	-	-	-	-	-	50.0	-
Trifolium p.	-	-	-	-	-	0.2	-	-	-	-	-	-	-	9.5
False Positives	80.0	16.2	44.7	-	78.0	5.2	3.3	11.7	5.2	-	100	38.5	25.0	42.9

Table A.2: The table shows the confusion matrix. The columns represent what the model predicted and the rows represent what the model should have predicted (the ground truth). The green, red, orange and brown numbers denote TP, FP, FN and confusions between two flowers respectively. The numbers are relative to the number of predictions of a specific made by the network.

A.2 Predictions on Simulated Resolutions

Section 4.5.1 presents the effect of decreasing the image resolution. When applying the resolution degradation only to the test images but leaving the training images and validation images at the full resolution to train a network, the performance drops much more rapidly as can be seen in figure A.1b and A.1a.



(a) Evolution of F1 score.

(b) Evolution of mAP.

Figure A.1: Evolution of performance metrics over various simulated ground resolutions with a model trained only on the full resolution images (1 mm/pixel ground resolution).

Considering the example images in table 4.10, this makes sense because the images at full resolution are distinctly different from the images with lower ground resolution. This means that it is not possible to train a network with high ground resolution images and using it to run the prediction algorithm on lower ground resolution images.

A.3 Mispredictions of Network Trained on Multiple Ground Resolutions

The network trained on multiple tile sizes (last column in table 4.3) generates unsatisfying results, especially for flowers with the same color. Figure A.2a depicts a typical source of errors. All yellow flowers in the image are ranunculus. However, the network cannot differentiate between the various yellow flowers that it is trained on. For one instance it predicts crepis biennis (violet bounding box) and ranunculus (green bounding box). Because the confidence score of crepis biennis is higher than the one of ranunculus, the ranunculus prediction is discarded by the non maximum suppression function. Two more flowers are falsely predicted as crepis biennis and the last instance on the right is predicted as lotus corniculatus (blue bounding box).

The problems for overlapping leucanthemum vulgare described in section 4.3.1 seem to be less severe with a network trained on multiple tile sizes. Figure A.2b illustrates the prediction performance on leucanthemum vulgare. It can be obtained that all overlapping flowers are correctly predicted. The reason appears to be that the network uses varying bounding box sizes to predict the flowers instead of just one.

(a) Multiple predictions for one flower. (b) Overlapping leucanthemum vulgare

Figure A.2: Example results of the network trained on multiple tile sizes.

Appendix B

Installation

B.1 Installation instructions

1. Download the MasterThesis Github repository and extract it. This can either be done by downloading the repository as a zip file from https://github.com/tschutli/MasterThesis and extracting it or by cloning the repository using the following command:

   ```
   git clone https://github.com/tschutli/MasterThesis
   ```

2. Download Anaconda with Python 3.7 from https://www.anaconda.com/distribution/ and install it.

3. Open the Anaconda Prompt Program, change to the previously downloaded MasterThesis folder and run the following command:

   ```
   conda env create -f environment.yml
   ```

   This will install all python dependencies including the Tensorflow library with GPU support.

4. Run the following command to install the LabelMe application:

   ```
   pip install labelme
   ```

5. Inside the Anaconda Prompt change directory to Tensorflow and run the following two commands:

   ```
   conda activate tf_gpu
   python cli.py
   ```

6. Done! You should now be able to use the command line interface developed during this thesis. Just follow the instructions displayed by the above command or watch the tutorial videos located in `Masterthesis/Documentation/TutorialVideos`.

B.2 Remarks

- You should not need any administrator rights.
- The described installation installs a pre-compiled version of Tensorflow. This version is not optimized for certain special CPU instructions. If you have a CPU that supports special CPU instructions such as the ones from the AVX or AVX2 set, you can build Tensorflow from source. This is however a very tedious process. If you run the code on a GPU, using an optimized version of Tensorflow does not improve the performance significantly since it only optimizes the execution on the CPU.
- The standard pixel size of each flower can be defined in the `flower_info.py` file. If you annotate images on the PhenoAnnotator tablet with point annotations, add the standard pixel sizes of your classes to this file.
- The PhenoAnnotator tablet app can be downloaded from the Google Playstore.

Appendix C

User Manual For Command Line Tool

The command line tool developed during the thesis is used as follows:

```
Usage: python cli.py [OPTIONS] COMMAND [ARGS]
```

The following list of commands is available:

- `annotate` Annotate images or adjust existing annotations.
- `copy-annotations` Copy Annotations to geo referenced images.
- `evaluate` Evaluate Predictions.
- `export-annotations` Export annotations to shape files.
- `export-inference-graph` Export the trained inference graph.
- `generate-heatmaps` Generate heatmaps from predictions.
- `image-preprocessing` Prepare a folder with annotated images for training.
- `predict` Run Prediction.
- `prepare-for-tablet` Prepare an image for the Android Annotation App.
- `train` Train a network.
- `visualize` Visualize Bounding Boxes.

Each command is described in more detail in the following sections. Along with a brief description of what the command does, all arguments and options are described.

C.1 annotate

Running this command will open the LabelMe Application with which all images in the input-folder can be annotated. If the images in the input folder are already annotated, these annotations can be viewed and adjusted.

```
python cli.py annotate [OPTIONS] INPUT_FOLDER
```

Arguments:

`INPUT_FOLDER`
: Input folder path containing images.

Options:

`-roi-strip BOOLEAN`
: If the roi-strip flag is set to True, the user can select Regions of Interest (RoI) in the images. To do so, the user has to draw one or multiple polygons around the region(s) of interest and label them 'roi'. After the user has labelled all images accordingly, only the Regions of Interest (RoI) are kept in the images. The rest of the pixels are overriden with black. [default: False]

`-h, -help`
: Show help and exit.

C.2 copy-annotations

If you have one folder with annotated images that are geo referenced, with this command you can copy the annotations from that folder to other images that are also georeferenced. An image can be georeferenced in the standard geotif format in any coordinate system. Alternatively, it can be a normal png or jpg image with a json file called imagename_geoinfo.json in the same folder. The imagename_geoinfo.json file must contain the following information in the WGS84 coordinate system: {"lr_lon": a, "lr_lat": b, "ul_lon": c, "ul_lat": d} With CTRL+C the execution of the script can be interrupted. In the one-by-one mode, the execution can later be continued.

```
python cli.py copy-annotations [OPTIONS] ANNOTATED_FOLDER
TO_BE_ANNOTATED_FOLDER OUTPUT_FOLDER
```

Arguments:

`ANNOTATED_FOLDER`
: Folder path containing georeferenced annotated images.

`TO_BE_ANNOTATED _FOLDER`
: Folder path containing georeferenced images.

`OUTPUT_FOLDER`
: Folder path where the results should be saved to.

Options:

`-one-by-one BOOLEAN`
: If True, the annotations will be applied to one image in the 'to-be-annotated-folder' at a time. It will be shown to the user in the LabelMe Application such that the user can check and adjust the copied annotations. [default: True]

`-h, -help`
: Show help and exit.

C.3 evaluate

If the images on which the predictions algorithm was run on had groundtruth information, this command will evaluate the performance of the prediction algorithm on these images. (Evaluation of Precision, Recall, mAP and F1 score)

```
python cli.py evaluate [OPTIONS]
```

Options:

`--project-dir PATH`
: Provide the project folder that was used for the predictions.

`--predictions-folder PATH`
: The folder where the predictions were saved to.

`--evaluations-folder PATH`
: The folder where the evaluation results should be saved to.

`--iou-threshold FLOAT RANGE`
: Defines what is the minimum IoU (Intersection over Union) overlap to count a prediction as a True Positive. [default: 0.3]

`--generate-visualizations BOOLEAN`
: If True, the erroneous predictions will be printed onto the images and saved to the evaluations-folder [default: False]

`--print-confusion-matrix BOOLEAN`
: If True, the confusion matrix will be printed to the console in latex table format. [default: False]

`--min-score FLOAT`
: The minimum score a prediction must have to be included in the evaluation [default: 0.5]

`--visualize-info BOOLEAN`
: If True, in addition to the bounding boxes, info about the mispredictions is painted above the boxes. [default: False]

`-h, --help`
: Show help and exit.

C.4 export-annotations

Exports the trained network to a format that can then be used to make predictions.

```
python cli.py export-annotations ANNOTATION_FOLDER OUTPUT_FOLDER
```

Arguments:

`ANNOTATION_FOLDER`
: Folder path containing annotated images.

`OUTPUT_FOLDER`
: The desired output folder path where the shape files should be saved to.

C.5 export-inference-graph

Exports the trained network to a format that can then be used to make predictions.

```
python cli.py export-inference-graph [OPTIONS]
```

Options:

`--project-dir PATH`
: Provide the project folder that was also used for the training.

`--model-selection-criterion [mAP|f1]`
: If the train command was executed with the '–with-validation True' flag, the model with the best performance on the validation set is exported (in terms of either mAP or f1 score). [default: f1]

`--checkpoint INTEGER`
: If a specific checkpoint should be exported, the checkpoint id can be provided with this flag.

`-h, --help`
: Show help and exit.

C.6 generate-heatmaps

```
python cli.py generate-heatmaps [OPTIONS]
PREDICTIONS_FOLDER OUTPUT_FOLDER
```

Arguments:

`PREDICTIONS_FOLDER`
: The folder containing the predictions.

`OUTPUT_FOLDER`
: The folder where the heatmaps should be saved to.

Options:

`--heatmap-width INTEGER`
: Defines the the number of pixels the heatmap will have on the x axis. The height of the heatmap is chosen such that the width/height ratio is preserved. This heatmap will finally be resized to the size of the input (or background) image. [default: 100]

`--max-val INTEGER`
: If defined, it denotes the maximum value of the heatmap, meaning that all values in the heatmap that are larger than this max_val will be painted as red.

`--flower TEXT`
: For which class the heatmap should be generated. If None is provided, only the overall heatmap for all classes is generated. This flag can be defined multiple times.

`--min-score FLOAT`
: The minimum score a prediction must have to be included in the heatmap. [default: 0.5]

`--overlay BOOLEAN`
: If True, the heatmap is drawn onto a copy of the input image. Otherwise it is drawn without any background. [default: True]

`--output-image-width INTEGER`
: The width of the output image, the height is resized such that the width/height ratio is preserved. [default: 1000]

`--generate-from-multiple BOOLEAN`
: If True, the script takes all predictions in the input folder and generates one heatmap from all of them. For this option, the input folder needs to contain georeferenced images and the background-image option has to be set. [default: False]

`--background-image PATH`
: The path to the image that should be used as background for the heatmap. (The background can still be deactivated with the –overlay flag but it needs to be provided as a frame for the heatmap.) If generate-from-multiple is set to False, this option is ignored.

`---window FLOAT...`
: Four float values indicating the [ulx, uly, lrx, lry] coordinates in the swiss coordinate system LV95+ of the area that should be used for the heatmap.

`--with-colorbar BOOLEAN`
: If True, a colorbar will be printed onto the output image. [default: True]

`-h, --help`
: Show help and exit.

C.7 image-preprocessing

Running this command converts one or multiple input folders containing annotated images into a format that is readable for the Tensorflow library. The input folders must contain images (jpg, png or tif) and along with each image a json file containing the annotations. These json files can either be created with the widely used LabelMe Application or with the AnnotationApp available for Android Tablets.

```
python cli.py image-preprocessing [OPTIONS]
```

Options:

`-i, --input-folder PATH`
: Input Folder Path (can be defined multiple times)

`-t, --test-split FLOAT RANGE`
: Float between 0 and 1 indicating what portion should be used for the test set (must be the same number as input folders -> one number for each folder)

`-v, --validation-split FLOAT RANGE`
: Float between 0 and 1 indicating what portion should be used for the validation set (must be the same number as input folders -> one number for each folder)

`--project-folder PATH`
: This project directory will be filled with various subfolders used during the training or evaluation process.

`--tile-size INTEGER`
: Tile size to use as tensorflow input (squared tiles). Can be more than one! [default: 450]

`--split-mode [random|deterministic]`
: Test set / Train set splitting technique. Deterministic mode ensures that an input directory is split the same way if this command is executed multiple times. [default: random]

`--min-instances INTEGER`
: Minimum instances of one class to include it in the training [default: 50]

`--overlap INTEGER`
: The image tiles are generated with an overlap to better cover flowers on the edges of the tiles. Define the overlap in pixels with this flag. [default: 50]

`--model-link TEXT`
: The link from where the pretrained model can be downloaded.

`-h, --help`
: Show help and exit.

C.8 predict

Runs the prediction algorithm on images (png, jpg and tif) of any size.

```
python cli.py predict [OPTIONS]
```

Options:

`--project-dir PATH`
: Provide the project folder that was used for the training.

`--images-to-predict PATH`
: Path to a folder containing images on which the prediction algorithm should be run.

`--predictions-folder PATH`
: Path to a folder where the prediction results should be saved to.

`--tile-size INTEGER`
: Image Tile Size that should be used as Tensorflow input. [default: 450]

`--prediction-overlap INTEGER`
: The image tiles are predicted with an overlap to improve the results on the tile edges. Define the overlap in pixels with this flag. [default: 50]

`--min-score FLOAT`
: Float between 0 and 1 indicating the minimum confidence a prediction must have to be considered. [default: 0.5]

`--visualize-predictions BOOLEAN`
: If True, the prediction bounding boxes are painted onto copies of the input images and are saved to the predictions-folder. [default: True]

`--visualize-groundtruth BOOLEAN`
: If True, the groundtruth bounding boxes are painted onto copies of the input images and are saved to the predictions-folder. [default: False]

`--visualize-score BOOLEAN`
: If true, the score is printed above each bounding box. [default: False]

`--visualize-name BOOLEAN`
: If true, the class name is printed above each bounding box. [default: False]

`--max-iou FLOAT`
: Float between 0 and 1 indicating the maximal iou for the non maximum suppression algorithm. For all predictions with an iou greater than max-iou, only the one with the better score is kept. [default: 0.3]

`-h, --help`
: Show help and exit.

C.9 prepare-for-tablet

Given an input-image (any format) and an output-folder, the command tiles the input-image into tiles of suitable size for an android tablet. If the input image is georeferenced (can be georeferenced tif or other image format with a imagename.imageformat.aux.xml file in the same folder(=>see gdal)), the script generates additional files with geo information that are read and used by the tablet app for displaying the user location. An additional advantage of using a georeferenced image as input is that after annotating on the tablet, all annotations can be copied onto other georeferenced images with the copy-annotations commmand.

```
python cli.py prepare-for-tablet [OPTIONS] INPUT_IMAGE
OUTPUT_FOLDER
```

Arguments:

`INPUT_IMAGE`
: Path of the input image that should be tiled to be usable on the tablet.

`OUTPUT_FOLDER`
: The path of the desired output folder.

Options:

`--tile-size INTEGER`
: Tile size to use for tablet. Too large tile sizes can cause the app to crash. [default: 256]

`-h, --help`
: Show help and exit.

C.10 train

Trains a network. Pressing CTRL+C during the training process interrupts the training. Running the train command again will resume the training. If you want to start training from the beginning, make sure that the contents of the <path to project-folder>/training folder are all deleted.

```
python cli.py train [OPTIONS]
```

Options:

`--project-dir PATH`
: Provide the project folder that was also used for the image-preprocessing command.

`--max-steps INTEGER`
: Max Training steps to carry out. [default: 130000]

`--with-validation BOOLEAN`
: If true, the training process is carried out as long as the validation error decreases. If false, the training is carried out until max-steps is reached. [default: True]

`--stopping-criterion [mAP|f1]`
: If the train command was executed with the '–with-validation True' flag, the training is stopped once either the mAP or the f1 score stop improving. [default: f1]

`-h, --help`
: Show help and exit.

C.11 visualize

Draws the bounding boxes on each image in the input folder. The input folder therefore needs to contain images (png, jpg or tif) and annotation files (LabelMe json, AnnotationApp json or Tensorflow xml format)

```
python cli.py visualize [OPTIONS] INPUT_FOLDER OUTPUT_FOLDER
```

Arguments:

`INPUT_FOLDER`
: The folder containing images and annotation files.

`OUTPUT_FOLDER`
: The folder where the visualizations should be saved to.

Options:

`--with-name-info BOOLEAN`
: If True, the name will be printed on top of each bounding box. If False, only the bounding boxes will be drawn. [default: True]

`--clean-output-folder BOOLEAN`
: If True, all contents of the output folder will be deleted before the execution of the command. [default: True]

`-h, --help`
: Show help and exit.