Flower detection using object analysis: new ways to quantify plant phenology in a warming tundra biome.

Karol Stanski

MInf Project (Part 1) Report

Master of Informatics
School of Informatics
University of Edinburgh

2018

Abstract

Significant sensitivity to changing temperatures makes phenological data an extremely valuable record for studying the potential influence of global warming on various ecosystems around the world [182]. However, due to the great species richness as well as vast and hardly accessible areas of habitat, currently available methods of data collection are very time-consuming and severely limited to certain locations. In this project, we aim to provide a new fully-automated method for species tracking that allows for the further development of image analysis tools for real-world ecological data collection in any type of flowering ecosystem.

We utilise the most recent deep-learning approaches towards automatic object detection in combination with high spatial resolution imagery collected by unmanned aerial vehicles (UAV) in order to generate the phenology records of *E. vaginatum* species communities in the Arctic tundra. Our final model, based on the very successful Faster R-CNN architecture, utilises a 2-convolutional-block feature extractor in combination with a parametric ReLU activation (PReLU). Such a set-up achieves an impressive AP of 77% on our test set. Moreover, our network indicates almost human-like performance in case of species counting, being able to detect 90% of the *true* number of objects presented within the investigated area. Achieved results demonstrate the potential of deep-learning-based methods for accurate as well as fast detection of small objects, despite using remote sensing drone imagery characterised by a high amount of noise and a limited field of view.

Furthermore, for the purpose of this project, we have generated an extensive dataset of 2160 images containing 34855 annotated *E. vaginatum* flower objects, making it a valuable resource for future studies regarding the phenology of this particular species as well as tundra biome in general.

Acknowledgements

I would like to thank both of my supervisors Chris Lucas (School of Informatics) and Isla Myers-Smith (School of Geosciences) for all their efforts to help me with the successful completion of this project.

I would also like to thank all the Team Shrub members [153], with a special mention to Gergana Daskalova, Jakob Assmann, Jeffrey Kerby and Andrew Cunliffe for dedicating their own time and efforts to collect all the data for this project.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The rapidly rising temperatures caused by global warming are accelerating the phenology and affecting the expansion of many organisms around the world [127]. However, due to the great species richness and vast areas of habitat, it unfeasible to track and assess all the changes when using the standard on-site measuring methods [124]. In our dissertation, we aim to incorporate the most recent deep-learning approaches towards the automated object detection [134, 142], in order to alleviate this issue and deliver a fully-automated tool for species communities detection, tracking and counting in any flowering ecosystem using high spatial resolution drone imagery.

The Arctic, experiencing an average temperature increase of approximately 2°C since 1950, is warming more rapidly than any other biome on the planet [119]. Moreover, its average temperature is predicted to rise further by 6-10°C within the next 100 years [119]. Such a vastly warming environment leads to more extensive snowmelt in the spring and much longer photosynthetic activity of the local plants due to later snowfall in autumn [182]. These patterns suggest lengthening of the tundra's growing season [81] which has been estimated by recent studies to increase further by approximately 4.7 days per decade [121]. However, any further research regarding the changes in vegetation patterns and their extent is significantly limited by a relatively little number of available measuring methods which enable accurate plant age and growth tracking as well as comparison with the snow-free period data obtained through satellite monitoring [182]. Therefore, the exact consequences of warming in tundra plant communities are still uncertain.

Phenology is a study of plant and animal life cycle events including flower and leaf emergence in spring and their decay in autumn [31]. The timing of flowering influences a plant's reproduction capabilities. Its growth and ability to capture energy are determined by the timing of its photosynthetic tissue development [8]. Therefore, fluctuations in phenology are likely to influence the role and distribution of particular species in their communities over time. Furthermore, plant phenology can be influenced by changes in a variety of factors, including temperature [127, 139]. Thus, phenological records are

especially valuable parameters for studying the potential influence of global warming on various ecosystems [182].

One of the most widely used plant phenological records is the time and density of flowering (i.e. the number of individuals per area). The typical way that ecologists have gathered phenological data has been by observing phenological changes on-site in localised plots (i.e. $1m^2$ or along short transects) [124]. Unfortunately, these on-site observations are extremely time-consuming and highly difficult in less accessible areas [127]. However, rapidly developing technology allows for new approaches to the collection of phenology data including proximal remote sensing using drones [32]. The use of unmanned aerial vehicles (UAVs) is a cost-effective way to conduct detailed analysis with high spatial and temporal resolution as well as avoiding destructive sampling of sensitive ecosystems [18].

Various methods had been attempted in recent years in order to achieve fully-automated analysis of ever-growing aerial imagery for community species monitoring. These include simple image data thresholding [5], template matching [3] as well as regression analysis [7] and object-based image analysis (OBIA) [60]. However, none of these approaches successfully provided a fully-automated solution. More successful techniques involving other computer vision methods, such as maximally stable extremal regions (MSER) [82] and UAV imagery, have been used to count seabirds and turtles [100]. However, due to its sensitivity to blur and the tendency to prefer round regions [82], this method still did not enable fully-robust automated analysis of high spatial resolution drone imagery.

In recent years, a rapid development of deep-learning techniques due to the more capable hardware (i.e. GPU) and large-scale datasets gave a rise to numerous outstandingly performing image classification models based on deep convolutional neural networks (CNN) [25] with the most profound being AlexNet [88]. The successful performance of AlexNet in ImageNet Large Scale Visual Recognition Challenge [138] in 2012 proved that the methods based on CNNs are much more robust than the traditional feature extraction techniques such as the histogram of orient gradient (HOG) [171] or scale-invariant feature transform (SIFT) [98].

The most recent advances within the field of deep-learning engender further development of unified frameworks for object detection by combining region proposal extraction and CNN classification networks [25]. Specific architectures involve SPP-Net [67], R-CNN [58] as well as its Fast R-CNN [57] and Faster R-CNN [134] successors which had already proved to be extremely successful in tasks of aerial target tracking [25, 177, 29, 96] and pedestrian detection [94].

However, there were few attempts of direct application of deep-learning-based object detection models to high spatial resolution remote sensing images obtained by the UAVs [25]. Such imagery introduces an additional challenge due to the relatively small object sizes when compared with the background and image distortions caused by the drone movement [19]. Therefore, our project aims to investigate the performance of the most recent object detection Faster R-CNN architectures [134] on the extremely challenging task of flower detection from high spatial resolution UAV imagery in order to deliver a fully-automated tool for real-world ecological data collection.

Our dissertation focuses on the detection and counting of Arctic tundra *Eriophorum vaginatum* cotton-grass flower species [139] in drone imagery. *E. vaginatum* is a widely distributed tundra flower found across the North American and Siberian Arctic which is monitored for plant phenology at sites around the tundra biome [68]. Automatic flower detection and counting could substantially speed up and improve phenology data collection in tundra ecosystems providing landscape-level information on plant responses to global temperature change [68]. Our project aims to provide a new fully-automated method for species tracking and allow for the development of image analysis tools for real-world ecological data collection in tundra as well as other flowering ecosystems (i.e. grasslands, deserts).

Firstly, we will discuss previous work in the field of object detection as well as present the chosen approach used to achieve the project's objectives (Chapter 2). Next, we will describe the data collection strategy and the final dataset used to train our model (Chapter 3). Chapter 4 focuses on the methodology and design choices made to build the best-performing architecture, whereas Chapter 5 describes the training process and experiments performed in order to determine the most optimal model set-up (i.e. both architecture and hyper-parameters). Lastly, in Chapter 6, we provide the reader with a complete evaluation framework to assess the performance of our final model.

1.2 Goals

The following is the overview of the main objectives of a successfully completed project:

- **Data annotation tool**: an intuitive and easily accessible flower annotation tool would allow for comprehensive and robust dataset generation which is crucial for training any deep-learning-based models.
- **Dataset for *E. vaginatum* flower detection**: generation of an extensive dataset including coordinates of each species within an image is essential for achieving the appropriate level of generalisation by our network. Furthermore, such data could potentially be valuable for other phenological studies regarding *E. vaginatum* and the impact of global warming on the tundra biome.
- **Fully-automated and robust model for flower detection and counting**: we consider this as the main objective and primary indicator of the successfully completed project. If well performing, such a model could substantially improve and speed up phenology data collection regarding *E. vaginatum* proving that similar deep-learning-based object detection methods could be applied to other plant or even animal species monitoring.
- **Accurate and reliable evaluation method**: with a complex structure of our detection model and a wide range of available performance metrics, it is necessary to provide the reader with a simple way of assessing our network, not only against its various set-ups (i.e. hyper-parameters) but also against human annotators. Such metric is essential for indicating how close our model is from achieving human-like performance.

1.3 Contributions and achievements

Below, we present our achievements and contributions towards a new method of quantifying plant phenology in a warming tundra biome:

- **Data collection manuscript**: we have proposed a detailed data collection procedure which had been successfully implemented by Team Shrub members [153] resulting in an extensive set of high spatial resolution imagery of the Arctic tundra biome (Chapter 3). The original version of the manuscript has been presented in Appendix A.1.
- **Data annotation tool**: we provided the annotators with a user-friendly web-based tool for *E. vaginatum* flower annotation (Chapter 3).
- ***E. vaginatum* dataset**: in cooperation with other human annotators, we successfully generated a dataset of 2160 tiles (i.e. original image crops) containing 34855 flower objects. Despite the incompleteness of the proposed annotations (Chapter 6), our dataset provides a valuable *E. vaginatum* phenology data and can be utilised in neural networks training process yielding almost human-like performance (Section 6.4).
- **Parametric ReLU layer**: we have successfully incorporated a parametric ReLU (PReLU) activation layer [66] in our final Faster R-CNN architecture improving the performance our model when compared with the equivalent one using standard ReLU activation unit [116].
- **Model for *E. vaginatum* flower detection and counting**: we have successfully designed and trained a neural network based on the Faster R-CNN architecture and its MATLAB implementation (Chapters 4 and 5). Furthermore, our model is capable of detecting even small flower objects despite varying image quality factors and achieving almost human-like performance.
- **Extensive model assessment**: we have investigated various performance metrics and successfully assessed our model against human annotators (Chapter 6). Moreover, we have also proved the incompleteness of our ground-truth annotations as well as its influence on the final assessment results.

Chapter 2

Background

The most fundamental step towards a successful model implementation is choosing a suitable architecture for the considered task. To remind the reader, the goal of our project is to deliver a deep-learning-based detector capable of detecting and counting *E. vaginatum* flower objects of relatively small sizes from high spatial resolution drone imagery. Our model, therefore, needs to characterise with a high robustness to background noise (i.e. other plant species), motion blur as well as varying lighting conditions due to the variable weather conditions. In this chapter, we will present and discuss possible approaches to handle the task of *E. vaginatum* flower detection and counting (Section 2.1) as well as choose the most suitable architecture (Section 2.2).

2.1 Task definition

Neural networks are currently applied to a vast range of tasks (Figure 2.1), the most fundamental being image classification. In this case, each image is classified with an appropriate label based on the dominant object presented within it [88]. More complex tasks involve object localisation denoting the prediction of a region of the image, usually enclosed by a bounding box, where the dominant object is presented and apply classification to determine which class it belongs to [155]. Despite many architectures such as AlexNet [88], VGG-16 [155] or Res-Net [65] being successfully applied in such tasks, these approaches are not suitable to be used on their own in a much more complex *E. vaginatum* flower detection and counting problem due to the multiple objects being present in a single image. By contrast, semantic and instance segmentation, as well as object detection, are the types of tasks which consider multiple objects being included within a single image, thus indicating the potential to be applied in our project.

In semantic segmentation, the task is to understand an image at its pixel-level where each pixel is labelled by an appropriate object class that it belongs to [123]. Early approaches involved texton [150] and random [151] forests based models. However, the introduction of the deep-learning-based methods able to suppress such models by a large

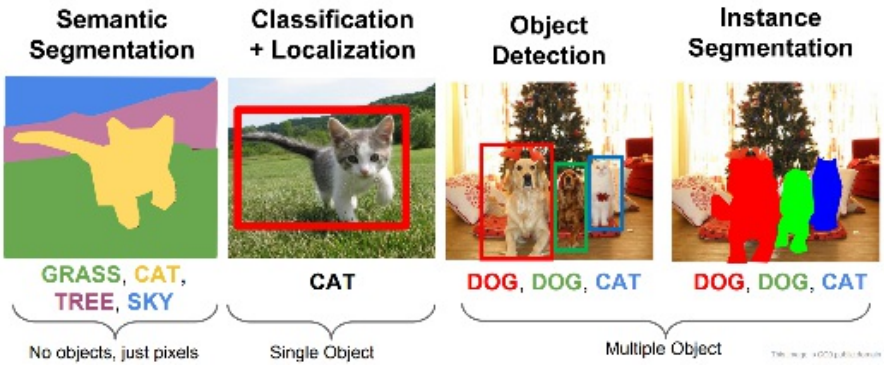


Figure 2.1: Different types of tasks in machine learning [51].

margin in case of both the performance and accuracy allowed for further development within the field.

An early deep-learning approach involved patch classification [30] with further improvement achieved by the fully convolutional networks (FCN) [148] allowing inputs of any size due to the lack of fully-connected layers. U-Net, on the other hand, utilised an idea of encoder-decoder architecture with the former component reducing and the latter recovering the object detail alleviating the issue of losing the localisation information caused by the pooling layers in other models. Further improvements based on the encoder-decoder approach were developed by [122] which introduced very large kernel convolutions.

By contrast, DeepLab network [21] approached the segmentation task by incorporating dilated convolutional layers [178] able to increase the field of view without adding extra parameters to the model also eliminating the need for the pooling layers. Moreover, its performance had further been improved by its later versions of v2 [22] and v3 [23] with the latter achieving an impressive score of 85.7% mAP in the PASCAL VOC 2012 challenge [50] outperforming previously mentioned networks (i.e. in semantic segmentation task).

Despite the rapid improvement of semantic segmentation based architectures, this approach was not the most suitable for our task due to its inability to distinguish between two adjacent instances of the same class (i.e. flowers may overlap with each other). Semantic segmentation simply classifies pixels which might be applicable to studies concerning plant coverage but does not allow for counting the exact number of individual species growing within a specified area.

Instance segmentation goes beyond the semantic segmentation approach, classifying each pixel of an image by the object class as well as determining particular object instance that it belongs to [123]. Recent studies indicate that achieving a robust instance segmentation network is very challenging due to the unknown number of instances and the inability of performed predictions evaluation in a pixel-wise manner as for the semantic segmentation [56]. As a consequence, the instance segmentation task remains a partially unresolved research topic [56].

Most successful instance segmentation architectures include DeepMask [125] and MaskLab [20] which are both build on top of the existing object detectors which are themselves responsible for each instance detection. Therefore, we did not define our task as instance segmentation problem since we could simply use object detector architectures that do not involve additional segmentation step [56] which we find unnecessary in case of flower detection and counting.

Hence, we focused on object detection architectures which aim to localise and classify all objects present in an image. Like in a single object localisation, predicted objects are enclosed within the bounding boxes. We found the object detection approach the most suitable for achieving the objectives of our project. Moreover, the already existing architectures for object detection proved to achieve relatively good performance even on noisy data with a significant number of objects of different classes and scales. Most popular architectures involve Faster R-CNN [134] and Single Shot MultiBox Detector (SSD) [97] which are discussed in more detail in the next section. Furthermore, a successful object detection makes flower counting just a sum of all detected objects within a specified area.

2.2 Object detection methods

Treating object detection as a regression problem where we try to predict the coordinates of the bounding boxes enclosing each object would not be feasible since each input image would yield a different number of outputs. The same applies to the classification approach since the detector would need to be applied to a huge number of image corps of different sizes making it an extremely computationally-expensive solution [133]. Therefore, a more feasible approach is to introduce an efficient method of region proposals extraction (known as region of interest, *RoI*) which are most likely to contain objects and perform classification and regression directly on these proposals.

One of the first approaches was to classify a fixed set of evenly spaced square windows sliding over an image of different scales introduced by OverFeat [142]. Despite the simplicity of this method, OverFeat was able to achieve satisfactory performance at the time, winning the ImageNet Large Scale Visual Recognition Challenge in 2013 (ILSVRC13) [138] and achieving 24.3% mAP (mean average precision) in the detection task on ImageNet dataset. However, many purposed windows contained only parts of the objects leading to good classification results but inaccurate detection.

An alternative was presented by the region-based convolutional neural network (R-CNN) [58] which utilised a separate region extraction algorithm such as selective search in combination with a feature extractor network (CNN). Running a separate method to generate region proposals was more time-consuming, resulting in R-CNN being almost two orders of magnitude slower than the OverFeat network. However, such approach significantly improved the accuracy of the detector to 31.4% mAP on the same ILSVRC13 detection dataset due to much better ability to localise each of the objects. R-CNN originally used 2000 regions proposed by the selective search method which were then rescaled to the size of 227×227 to fit the input dimensions of the AlexNet

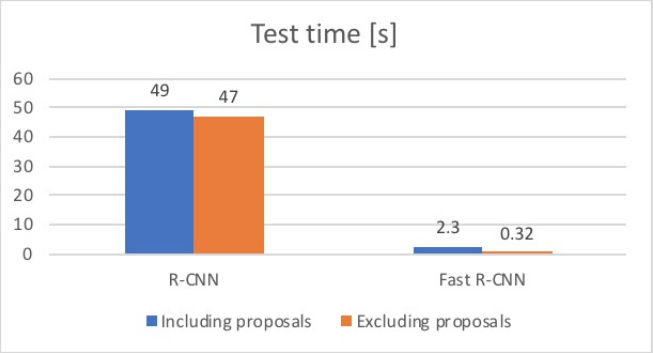


Figure 2.2: R-CNN [58] and Fast R-CNN [57] test time comparison. The benchmarks involve time spent on selecting region proposals and detection.

network used for feature extraction. Each region was then fed into the AlexNet in order to extract 4096 features and passed to an SVM performing classification along with multiple class-specific linear-regressors to refine the bounding boxes. Despite higher accuracy, R-CNN required 2000 separate passes through the convolutional network per image during the training phase which resulted in a very slow detection performance of 47 seconds per image when using VGG-16 [155] network as the feature extractor (instead of AlexNet [88]).

Further improvements were implemented in the Fast R-CNN architecture [57] which also utilised the selective search proposal extraction method, however, instead of feeding each region through the CNN, it required only a single pass of the whole image to extract a global set of feature maps. Fast R-CNN used a region of interest pooling layer which divided the projected proposals into 7×7 grid and performed maximum pooling within each cell, producing a fixed-size output. This was necessary in order to process the input further by the classification and regression steps which used fully-connected layers, thus requiring inputs of pre-defined size. As a result, Fast R-CNN was able to process inputs of any sizes with no need of image rescaling. This approach resulted in a significant 9 and 25 times speed up in training and test times respectively when compared with the original R-CNN. Moreover, the enhancement purposed by the Fast R-CNN, surpassed R-CNN performance yielding 68.8% mAP on the PASCAL VOC 2010 dataset (62.4% mAP for R-CNN; both using VGG-16). However, the biggest bottleneck of the Fast R-CNN was its region extraction method (i.e. selective search), accounting for approximately 80% of the time spent to process a single image (Figure 2.2).

Further architecture enhancement regarding region extraction was proposed by Multi-Box [162], which introduced the concept of region proposal network (RPN), proving that the convolutional networks could be more efficient for region proposal extraction than previously used heuristic approaches. RPN was a small convolutional network whose input was a set of feature maps of each image produced by the last convolutional layer of the base network (i.e. feature extractor) which operated in a sliding-window fashion to generate the region proposals. This idea was utilised in the Faster R-CNN architecture [134], where the combination of the RPN with the Fast R-CNN detector

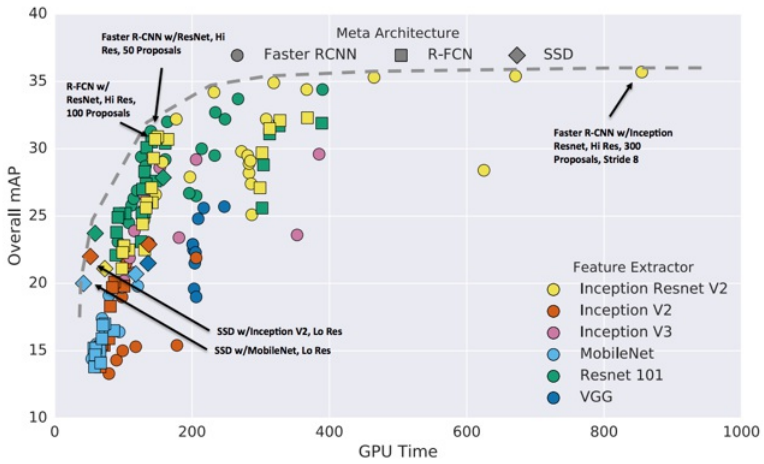


Figure 2.3: Speed/accuracy trade-off between different architectures for object detection [74].

outperformed previous state-of-the-art architectures, yielding 75.9% mAP on PASCAL VOC 2012 dataset (68.4% or the Fast R-CNN). Moreover, due to the lack of the selective search for the proposal generation, Faster R-CNN further reduced testing time to only 0.2 seconds per image.

Faster R-CNN is an example of the refined classification strategy which processes the prior bounding boxes and performs object classification on the features pooled from the set of feature maps produced by last convolutional layer of the base network. This strategy had been challenged by the direct classification approach used by the Single Shot Multibox Detector (SSD) [97] in which prior box regression and object classification were performed simultaneously and directly from the same input region, further reducing the test time. A similar approach was used in a region-based fully convolutional network (R-FCN) yielding an impressive, up to 20 times speedup over its Faster R-CNN counterpart [34]. Other alternatives included *You-Only-Look-Once* (YOLO) [133] network which divided each input image into $N \times N$ grid, directly predicting class scores (i.e. probabilities) along with the bounding boxes within a single pass through the network allowing real-time object detection.

Despite a wide variety of object detection networks, the Faster R-CNN architecture tends to outperform its counterparts including SSD and R-FCN using equivalent feature extractors (i.e. base networks) [74]. Whereas the alternative models tend to be much more computationally efficient, meaning that they require much less time to train and perform the detection, Faster R-CNN achieves better accuracy (i.e. mAP) [74] which is crucial in case of the task of very small *E. vaginatum* flowers detection and counting (Figure 2.2). Therefore, we based our model on the Faster R-CNN architecture in order to deliver a fully-automated method for *E. vaginatum* species tracking in the warming tundra biome.

Chapter 3

Dataset

Unlike the images used in most of the object classification and detection tasks such as ImageNet Large Scale Visual Recognition Challenge [138] or PASCAL VOC [50], high-spectral drone imagery is characterised by even greater complexity. This includes a very limited field of view variation due to the images being taken from the top-down perspective which severely limits the available object characteristics. Moreover, the object's dimensions are significantly smaller when compared with the background making the potential detector extremely vulnerable to noise. Nonetheless, due to the defined ground spatial resolution (Section 3.1) we can easily estimate the size of *E. vaginatum* flower objects which can greatly help with the network parameter adjustment. In this chapter, we focus on the dataset generation procedure, specifically its most challenging aspects and our approaches to tackling them with the available resources.

3.1 Original images

In our dissertation research, we considered a set of 2625 high spatial resolution drone images taken across four hectares on Qikiqtaruk - Herschel Island ($69°N$, $139°W$) in the Canadian Arctic (Figure 3.1). The data was collected during the summer of 2017 (i.e. June-August) by the Team Shrub research group from the School of Geosciences at the University of Edinburgh [153]. The drone platform used in this study was Phantom 4 Advanced Pro equipped with a 20-megapixel CMOS sensor [40]. The data collection protocol was created in cooperation with the field data collection team at the end of May 2017.

In order to improve the robustness of our model, we used the images from four $100m \times 100m$ separate tundra phenology sites ($PS1$, $PS2$, $PS3$, $PS4$) with varying types of terrain that were taken on different days, in various times of day and under different weather conditions (i.e. wind, cloud coverage). This procedure allowed us to generate a representative set of images of the investigated area in variable light conditions and including artefacts such as motion blur.

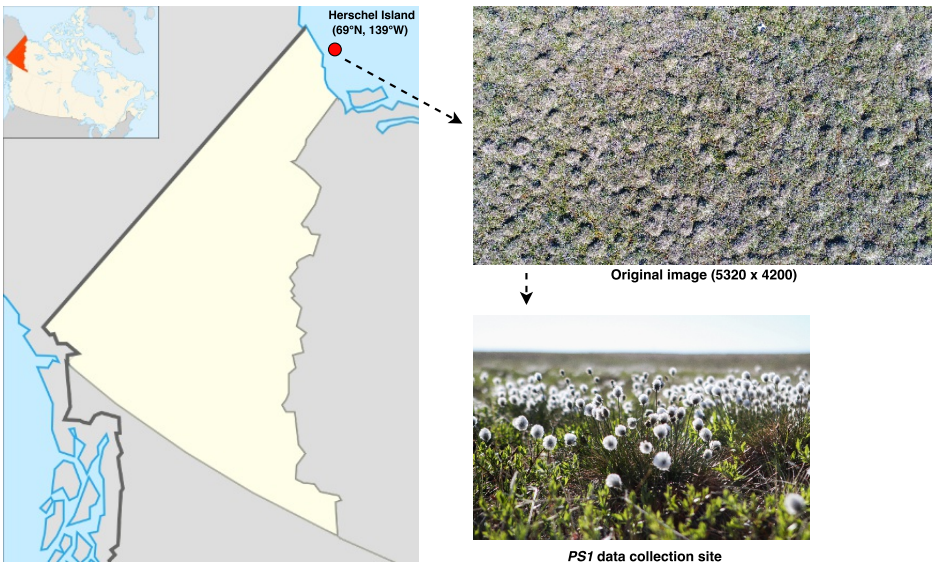


Figure 3.1: Herschel Island ($69°N$, $139°W$) location and an example drone imagery.

Altitude [m]	Forelap [%]	Sidelap [%]	GSR [mm]	Margin [m]	Number of images
12.2	64	74	3	2	1112
23.5	80	84	6	8.5	775
50	90	90	14	8.5	427
100	88	89	28	15	311

Table 3.1: Image parameters according to the flight altitude.

In order to provide a possibility of orthomosaic generation of the whole area and to avoid cases where a flower would be on the edge of two adjacent images, a sufficient overlap ($\sim 75\%$ front and $\sim 50\%$ side overlaps) between images was provided (Table 3.1). Moreover, for the purpose of scale-invariance detection, every flight covered each area four times to collect data at different flight altitudes (Table 3.1). As a result, due to the altitude difference, each set of images was characterised by a different ground spatial resolution (GSR) represented by the formula in Equation 3.1. This simple metric allowed us to easily determine what area corresponds to a single pixel in the image and therefore, the size of the overlap at each altitude.

$$GSR = Pixel\ size \times Altitude \times Focal\ length\ of\ the\ sensor\ [mm] \tag{3.1}$$

The visibility of flowers in drone imagery could be affected by light reflection, lens distortion as well as wind conditions which might greatly affect the annotators' judgement on presence or absence of a particular flower object. Therefore, the protocol included quadrant counts in randomly selected $1m \times 1m$ sections of the areas in order to obtain ground-counts for the final model evaluation (Section 6.3). The exact coordinates of each corner of the quadrant were marked and recorded to allow for the area extraction from the original imagery.

Due to the complexity of the task involving very small object detection, we decided to use only images taken from the lowest altitude of $12.2m$ above the ground level

Area	Date	Time (start)	Wind speed [m/s]	Sky conditions	Number of images
PS1	12/07/2017	19:00:00	4.5	1	132
PS1	14/07/2017	20:55:40	2	3,7	129
PS2	12/07/2017	17:51:00	5	1	131
PS2	14/07/2017	17:47:30	5	1	137
PS3	12/07/2017	14:41:00	3	2	133
PS3	14/07/2017	14:40:40	6	1	141
PS4	12/07/2017	16:16:00	4.5	1	155
PS4	14/07/2017	16:09:50	5	1	154

Table 3.2: Parameters of all expeditions at the flight altitude of $12.2m$ for each survey plot. Scale used to describe sky conditions had been provided in Appendix A.2.

collected on two different days (11^{th} and 14^{th} of July). This decision was made based on individual *E. vaginatum* flower average dimensions which were approximately 3-5cm [139]. This translates to just 10-16 pixels (i.e. in diameter) in even the highest resolution imagery (i.e. lowest altitude), making the detection extremely challenging.

Table 3.2 presents every single expedition at $12.2m$ altitude in more detail regarding the start time and weather conditions. Each expedition consisted of three flights starting from a different point on the area grid allowing to obtain images of the same area from different angles. Collecting images with multiple angles was extremely valuable since the visibility of some of the flowers was heavily affected by the wind conditions and motion blur in the imagery. The duration of each flight was approximately 3-4min. The difference in the number of images collected on each day was a result of slight variations in altitude, overlap and the coverage of the outskirts of the considered area, although not preventing the samples from being comparable.

After the successful data collection conducted by the field team members, the database consisted of 2650 high spatial resolution drone images (i.e. 5320×4200), gathered from the four survey plots in various weather conditions and at different altitudes. From this set, we chose 1112 images collected at a flight altitude of $12.2m$ for further processing.

Each original image was saved in JPEG format. This was a slight variation from our original protocol which proposed using both RAW and JPEG format for the most optimal detail retrieval. However, due to the limited disk space and time required to save each RAW formatted image during the flight, we decided to save the original data as JPEG only.

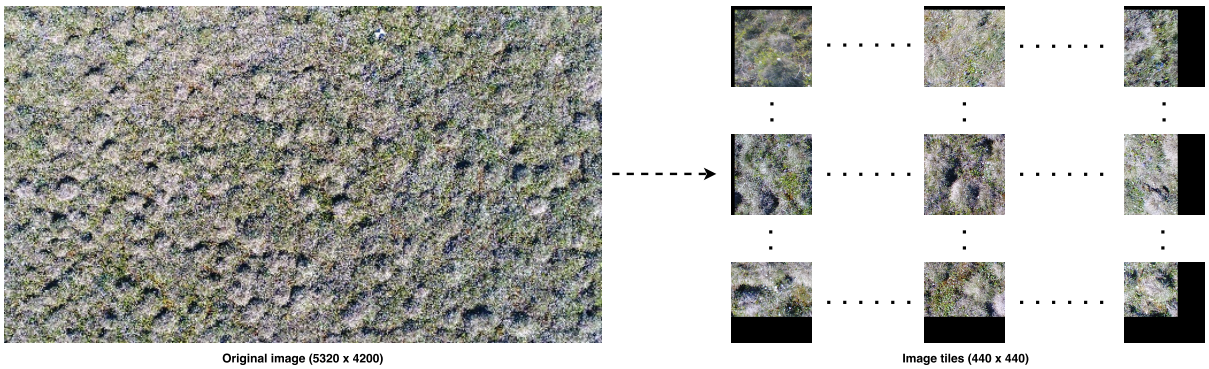


Figure 3.2: Due to relatively big dimensions of the original RGB imagery (5320×4200), we decided to split each image into 440×440 tiles.

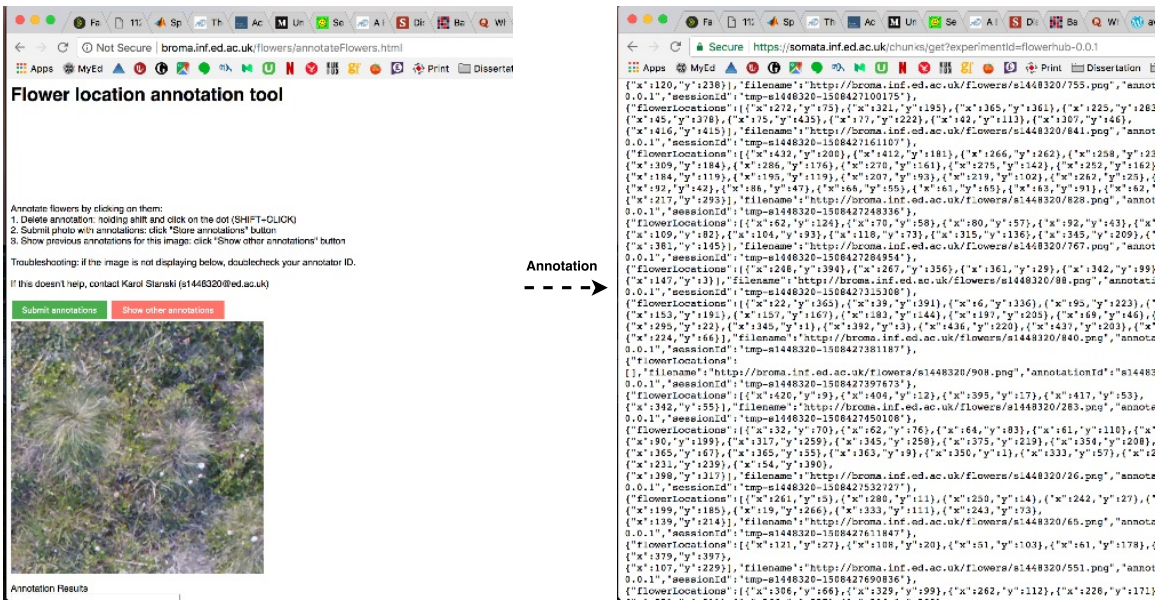
3.2 Data processing and annotation

After selecting the appropriate set of imagery, we divided chosen data into 24 groups (*folders*) to distinguish images from three flights on two different days, covering the four survey plots. The number of images in each folder varied between 43 to 52, depending on the flight. This procedure was necessary to provide the detector with evenly distributed and representative data among the flights to aid with its robustness.

Due to relatively big dimensions of the original RGB imagery (5320×4200), making it extremely inconvenient to annotate a single photograph in one sitting as well as requiring substantial amount of memory to process each image by the model, we decided to split each image into 440×440 tiles and save them in PNG format to provide lossless compression (Figure 3.2). The tiles provided a 20-pixel overlap from each size in order to avoid cases where a flower is missed due to its location on the edge of two adjacent tiles and to allow easy reconstruction of the original image. Therefore, each tile's *active* coverage area is equal to 400×400 pixels. Moreover, in order to preserve the dimension consistency among the data, we provided black pixel padding (i.e. the value of 0 per each RGB channel) for the outermost tiles.

As a result of the image cropping, the number of data-points increased to approximately 150000, making the future annotation process very time-consuming. Therefore, we decided to drastically reduce the number of tiles being used for annotation. Moreover, to achieve good generalisation of our model, we randomly selected 8% of tiles from 10 random images within each of the folders. This procedure resulted in a dataset of 2160 randomly selected tiles, evenly spread across images of different areas under various weather conditions.

In order to provide an intuitive way to annotate the tiles by other people, we needed to implement a user-friendly and easily accessible tool which did not require an installation of any additional software. Despite the availability of complementary online image annotation tools [78, 141], these were mostly applicable for easier tasks of image classification rather than object detection. Therefore, we implemented a simple web-based tool (Figure 3.3) allowing the users to automatically load a random, unannotated



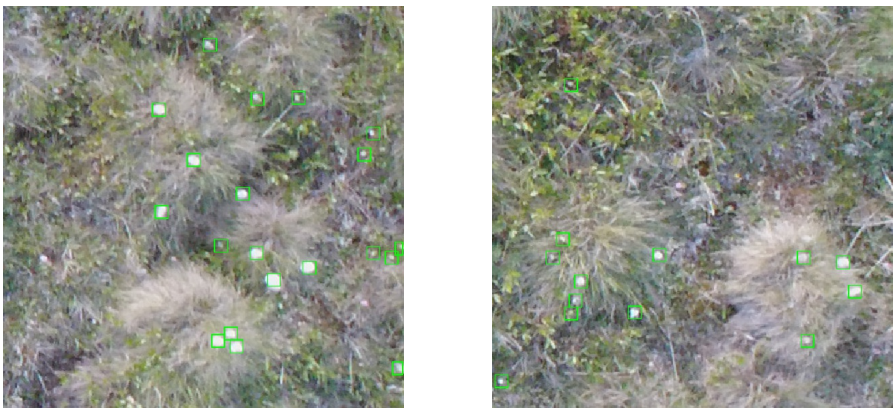
Data annotation tool (broma.inf.ed.ac.uk) JSON with saved annotations (somata.inf.ed.ac.uk)

Figure 3.3: Screen views of the annotation tool (left) and the saved flower coordinates annotated by the users (right).

tile after providing their username. The usernames were necessary to distinguish the annotators between each other and provide them with previously unannotated tiles as well as to obtain double-coverage of some of the images (i.e. annotation by two different people). This approach allowed us to investigate how various people annotated the same set of tiles and also improve the annotation quality in cases when some users missed a particular flower which resulted in a *collective* annotation process (i.e. if the annotations from two different people had IoU < 0.5, both of the bounding boxes were included).

The annotation tool, along with the dataset of 2160 tiles, were hosted on one of the servers at the University of Edinburgh School of Informatics (*broma.inf.ed.ac.uk*). Each user was able to annotate each flower by clicking on the area of the image which they recognised as a *E. vaginatum* flower with a distinct red dot being placed to indicate their choice. If the annotator made a mistake and wished to undo the annotation, they could easily do so by holding *Shift* key and clicking on the incorrect annotation. After completing a tile, each annotator submitted their choices by clicking the appropriate green button and automatically loading the next unannotated tile. All flower locations along with the annotator username, tile name and timestamp were then saved onto Somata server (*somata.inf.ed.ac.uk*) in an easily interpretable JSON format (Figure 3.3).

Moreover, each flower annotation was recorded as a single pair of *X*-*Y* coordinates within the tile and considered as a centre point of each flower. We are aware that such a simplified assumption might not always be valid due to the small pixel and flower sizes making it almost unfeasible to accurately click on the exact centre of a flower. However, that is why we provided double coverage for most of the tiles, refining the final annotations among various users.



(a) (b)

Figure 3.4: Example annotated tiles. Each flower is marked as a green 14×14 bounding box.

3.3 Final dataset

Due to the specificity of the object detection task requiring bounding box as the form of object annotation, we had to process the coordinates of flower centres and generate the bounding boxes. Therefore, after successfully completing the annotation process of the selected tiles, we extracted and processed the annotations from the JSON file saved on the *Somata* server. As a result, the data was saved as a table containing 2160 rows and two columns. Each row corresponded to a single image, whereas the first column contained a file path of each tile and the second column being a cell array consisting of varying number of integer arrays of length four corresponding to flower location (i.e. bounding box). First two entries of the integer array represented top-left coordinates of the bounding box with their height and width specified by the last two entries respectively.

The original Faster R-CNN architecture [134] used centre coordinates to determine the bounding box location rather than its top-left corner. However, we chose to follow the MATLAB interpretation of the bounding box annotation determined by the upper-left corner [110]. Hence, we adjusted the original annotations by simply shifting the centroids by an appropriate number of pixels (i.e. entries in the matrix representing each tile) depending on the flower dimensions. We estimated the height and width of each bounding box to be 14 pixels each due to the regular circular shape of *E. vaginatum* flowers, thus shifting each centroid by seven pixels in a top-left direction. Despite the severe oversimplification, our approach resulted in reasonable fit for the vast majority of the flowers within the purposed bounding boxes (Figure 3.4). This was possible due to the uniform ground spatial resolution of 3*mm* for all of the tiles which allowed us to estimate a typical size in pixels of an *E. vaginatum* flower based on its average diameter being approximately 3-5*cm* [139].

We are fully aware of potential disadvantages leading to inaccuracies of the ground-truth annotations such as bounding box misalignment. Hence, we consider a further improvement of the flower annotation process in the second part of this project. The

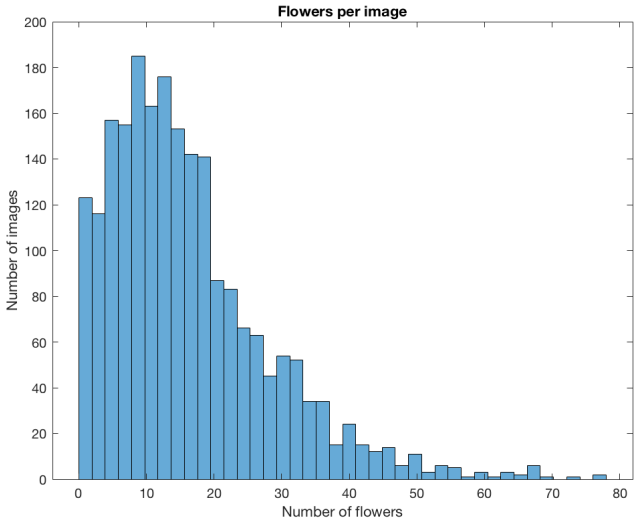


Figure 3.5: Histogram of the distribution of flowers among the selected tiles suggests a great imbalance between the number of annotated objects and the number of background examples.

annotation process using a single click and recording a single pixel as a flower's centroid was the most feasible approach at the time of implementation making the annotation process easy and understandable for the users. By contrast, drawing an appropriate bounding box around each flower would make the process more time-consuming, and vulnerable to inaccuracies due to the very small size of the objects.

Finally, after successful bounding box generation, we divided the modified dataset into training and testing sets. An appropriate ratio between those divisions was crucial for model generalisation ability. The training dataset is used to fit the parameters (i.e. weights of the connections between nodes of the neural network and bias terms) whereas the testing dataset allows us to evaluate the final network fit for the training data [47]. The division ratio between these sets greatly depends on the complexity of the task as well as the dataset itself [15]. The the most commonly used division ratios greatly differ between studies and range from 90%-10% to even 60%-40% for training and testing respectively [61]. Due to the lack of universal method for dataset division, we followed *Pareto principle* [167] and randomly divided our dataset into training and testing sets with ratio of 80%-20% with a possibility of extending the testing set with newly annotated part of the remaining tiles which had not been selected in the procedure described in Section 3.2. As a result of the split, the final training and testing sets consisted of 1728 and 432 tiles respectively.

It is worth to note that we have not used any tiles to form a validation set. The validation set denotes a separate set of datapoints which is used to provide an unbiased evaluation of the network fit on the training set enabling hyper-parameter tuning [16]. However, we were unable to take advantage of the validation set due to the lack of such option provided by the `trainFasterRCNNObjectDetector` method at the time of the implementation. We consider incorporating the validation set to our training process as a future enhancement which might help with combating network overfitting (see Section 5.1.1).

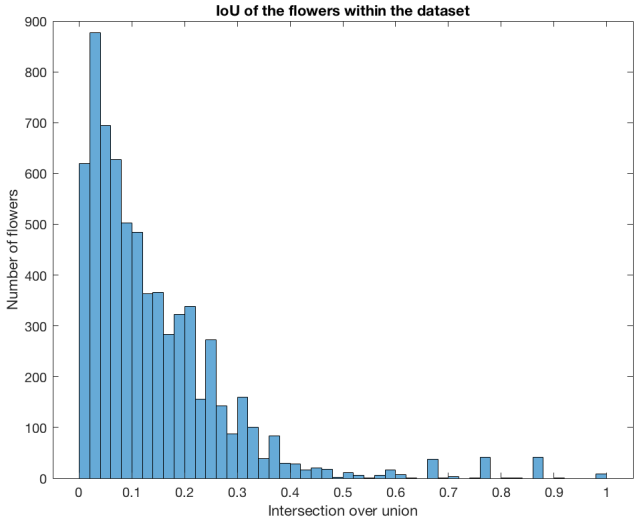


Figure 3.6: Histogram of the amount of intersection between annotated flower bounding boxes (datapoints with no intersection not included) suggests that our final dataset does not include many cases of highly-clustered groups of flower objects.

Our final dataset contained 34855 annotated flowers (6933 in testing and 27922 in training set) within 2160 tiles yielding approximately 16 flowers per tile on average. Figure 3.5 indicates a relatively uneven distribution of flowers among the images with almost 72% of tiles containing 20 or less annotated objects. This suggests a great imbalance between the number of annotated objects and the number of background examples making it extremely challenging for the network to fill the mini-batch with the appropriate amount of positive example proposals (Section 4.2). However, as discussed in Section 6.3, these statistics are very likely to be underestimated due to the poor quality of the annotations which do not include all flower objects present within each survey plot.

Furthermore, in order to investigate how sparsely the flowers were distributed within each tile, we plotted the histogram showing the extent of intersection between flowers in Figure 3.6. As a measure of bounding box intersection we used the notion of intersection over union (IoU), also known as Jaccard index, which is defined as the size of the intersection between two bounding boxes (in pixels) divided by the size of their union [131] yielding a score of 1 if the boxes perfectly overlap and 0 if there is no overlap between them (Section 4.2.2).

Due to the clarity of the graph, we included only flowers whose bounding boxes intersected by at least one pixel (i.e. IoU > 0) which applies to only 19.5% of all the annotated flowers. Furthermore, only 8% of all the bounding boxes had IoU of over 0.1 suggesting that our final dataset did not include many cases of highly-clustered groups of flower objects which could further complicate the detection process for the model as well as human annotators.

Chapter 4

Methodology

The architecture of Faster R-CNN is a complex structure consisting of three distinct parts (i.e. sub-networks) and trained in four separate stages (Section 5.1) [134]. The inputs to the network are the images containing objects that we aim to learn the detection of and the ground-truth bounding boxes indicating the exact location for each of the objects. In case of our network the input data is the training set described in Section 3.3 consisting of 1726 RGB images (i.e. tiles), each represented as a $440 \times 440 \times 3$ matrix. These tiles are processed by the following sub-networks within the Faster R-CNN pipeline:

- **Convolutional neural network (base network)**: extracts features of the input tiles (Section 4.1).
- **Region proposal network (RPN)**: finds a predefined number of region proposals, using the features from the base network, which are likely to contain relevant objects (Section 4.2).
- **Region-based convolutional neural network detector (Fast-CNN)**: uses the outputs of two previous components (feature maps from the base network and region proposals from the RPN) to refine the proposed bounding boxes, classify the objects inside them and output the final detection results (Section 4.3).

After the completed training process, the network outputs bounding boxes of detected objects, labels assigned to objects indicating the class of the object (i.e. *flower*) and the probability scores of each bounding box belonging to that class.

The diagram of the complete Faster R-CNN architecture can be found in Appendix B. Our implementation of the described model had been based on the `fasterRCNNObject Detector` MATLAB class [106] and can be found in `fasterRCNN.m` file.

4.1 Base network

The base network is a backbone of the Faster R-CNN architecture. It is used to extract a set of features from each input tile (i.e. $440 \times 440 \times 3$ matrix) which is outputted by its last convolutional layer (Figure 4.1). These features are essential for the region proposal network (RPN; Section 4.2) and Fast R-CNN detector (Section 4.3) to extract region proposals and further classify and refine the bounding boxes.

We based our base network on the VGG-16 architecture [155] which introduced convolutional blocks with increasing effective receptive fields of each block. This property allowed us to extract more complex features using smaller kernels, thus reducing training and testing times as well as yielding superior performance over other networks in equivalent tasks [155]. VGG-16 had been incorporated in the original version of the Faster R-CNN architecture and since then became the most widely used base network within this pipeline with slight variations including ZF-Net [179, 134] or Res-Net [65, 74].

We decided to reduce the number of convolutional blocks within the base network, making it shallower compared to the original VGG-16 and, as a result, much more appropriate for small object detection [45]. As proved by other studies, using fewer blocks results in the more suitable receptive field and larger feature maps due to the smaller amount of pooling layers, hence reducing the risk of information loss regarding smaller objects [62, 45]. We discussed this aspect in more detail in Section 5.2.2. Therefore, our baseline network contained 2 blocks, each consisting of 2 convolutional layers complemented by activation units and followed by a pooling layer (Figure 4.1). Specific details regarding the choices made for each part of the network will be discussed in the following subsections.

4.1.1 Input layer

The first component of every neural network is a passive input layer meaning that no computations are performed in any of its nodes [158]. It simply passes an image, represented by a $440 \times 440 \times 3$ matrix, to the hidden layers of the network (Figure 4.1). A big advantage of Faster R-CNN is its ability to process images of arbitrary sizes due to the lack of fully-connected layers in its RPN component (Section 4.2) and a special RoI-pooling layer in the Fast R-CNN detector part (Section 4.3) which adjusts region features to a fixed-size representation. Hence, the maximum size of the inputted image is determined by the amount of available memory to store the image and its proposals as well as available time to perform the computations.

Unlike in most implementations [134, 94, 29], we did not rescale the input images. This is because the downscaling of the original tiles would make the flower objects even smaller, whereas the upscaling would significantly lengthen the time needed to complete the training process due to the increased amount of computations within the model (i.e. bigger input).

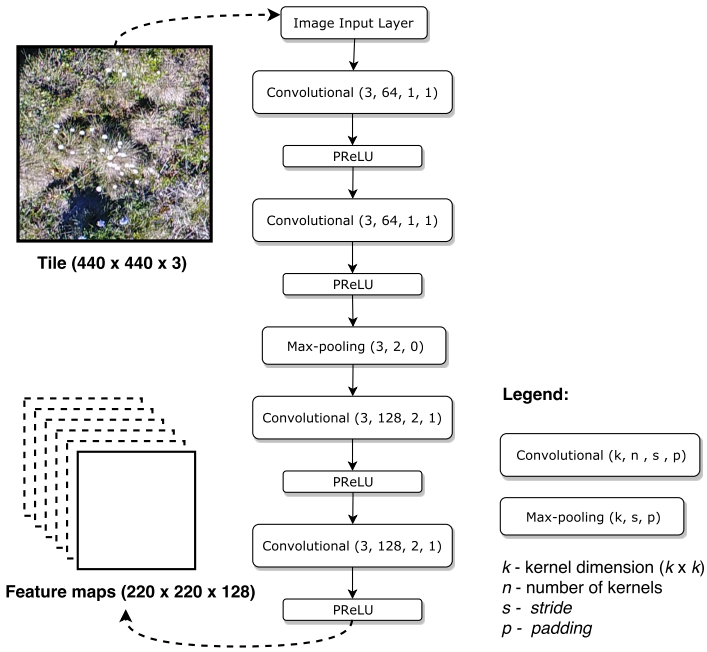


Figure 4.1: Base network architecture. Parameters k, n, s, p correspond to square kernel dimensions, number of kernels, stride and padding respectively.

We also decided to use MATLAB implementation of the `imageInputLayer` object [108] and apply a normalisation step of zero-centering the data. Zero-centering subtracts a mean image of the training set, centering the data towards the origin [85]. This operation is applied every time an image is forward propagated through the input layer resulting in faster network convergence by removal of possible bias in the gradients [33]. Furthermore, zero-centering makes the network more robust to images of various contrast and illumination [140] which is a crucial aspect in case of changing weather conditions within the considered area. Our experiments indicated noticeable performance benefits of applying zero-centering within the input layer (average precision increased from 0.7275 to 0.7719 when using zero-centering; Section 5.2.4).

4.1.2 Convolutional layer

Once the image has been zero-centred by passing through the input layer, the next step is feature extraction performed by the convolutional layer. Unlike fully-connected layer which treats the input as a one-dimensional vector, convolutional layer preserves spatial relationship between the pixels by considering small patches of the image at a time of the size determined by the kernel [91].

Kernel, also known as the filter, is a small $k \times k$ matrix (usually of size 3×3 or 5×5) which slides along the image computing the dot-product of its weights and the input pixels within its receptive field and adding the bias term. The size of the step by which the kernel is moving across the image is defined by the stride (s) which also determines the overlap between the subsequent operations if the stride is smaller than the size of

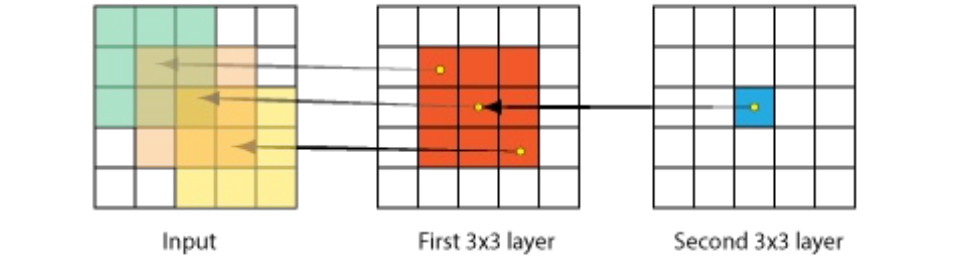


Figure 4.2: The effective receptive field (ERF) of 5 for 2 adjacent convolutional layers with 3×3 kernel, stride and padding of 1 [42].

the kernel (i.e. $k > s$). Moreover, we used zero-padding (p) in order to preserve the dimensionality of the outputted feature map (o) and prevent the network from losing part of the information in the case when the kernel is not fitting entirely within the input (Equation 4.1) [44].

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \tag{4.1}$$

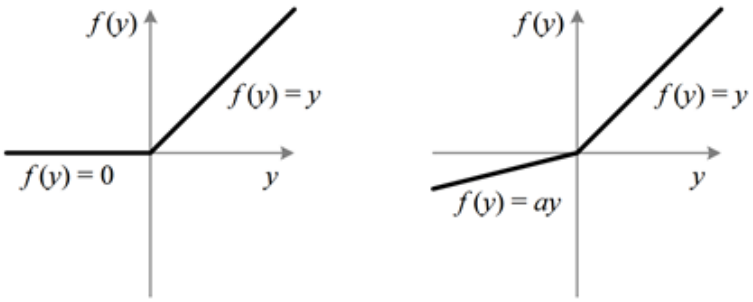
The result of the convolutional step is a set of two-dimensional matrices containing features of the image known as feature maps. Due to the same sets of weights and biases (i.e. weight sharing) within each kernel being applied on the input, all the neurons in a given convolutional layer respond to the same set of features within their specific response field [145]. Applying multiple filers on the same input enables deeper layers to combine simpler features from the previous layers (i.e. curves, edges) to extract more complex structures (i.e. eyes, faces) [160].

The effective receptive field (ERF) in the j_{th} layer represents the overall area of the input that influences the activation of a single node in j_{th} layer, and is represented by the formula in Equation 4.2 [29].

$$ERF_j = F_1 + \sum_{i=1}^{j} ((F_i - 1) \times S_{i-1}) \tag{4.2}$$

Here, ERF_j, F_i, S_{i-1} represent ERF of the j_{th} layer, receptive field of the i_{th} layer and the stride of the $(i-1)_{th}$ layer respectively. This implies that 2 adjacent convolutional layers with 3×3 kernel, stride and padding of 1, has the ERF of 5 (Figure 4.2).

We have chosen 3×3 kernels which allow us to achieve a higher effective receptive field (ERF) with additional introduction of non-linearity between each convolutional operation due to the activation unit (Section 4.1.3). This approach proved to yield significant improvement over other architectures using bigger kernels [88, 179], allowing the network to increase its depth and extract more complex representations of the data [155] as well as aiding the network with becoming more translation-invariant [79].



(a) ReLU (b) PReLU

Figure 4.3: Graphs of the considered activation functions [66].

4.1.3 Activation layer

The next component of the majority deep-learning-based architectures is an activation layer, also known as activation unit which introduces a non-linearity, enabling the network to extract more complex representations of the inputted data. Without the activation layer, every neural network would become a linear regression model making it equivalent to a single-layer network able to perform a linear regression, no matter its depth since the combination of linear functions is still a linear function. Hence, a non-linear activation function allows us to take the advantage of a multi-layer network to generate an output that cannot be reproduced from a linear combination of its inputs. This allows us to extract more complex features from the input images [132] tackling more complex real-life tasks which are usually non-linear [147].

To further understand the purpose of the activation function, let us consider a single node within the network. Each of the nodes calculates a *weighted sum* of its inputs x and weights w, and adds a bias term b (Equation 4.3). The output y can take arbitrary values (i.e. negative or positive) and it is determined by the activation function if the node is considered as *activated* by its outside connections.

$$y = \sum_i (w_i x_i) + b \tag{4.3}$$

Despite a wide variety of available activation functions, rectified linear unit (ReLU) ([116]) proved to be the most popular choice in most of the state-of-the-art architectures, including VGG-16 [155] used in the original Faster R-CNN [134]. As seen in Figure 4.3(a), ReLU introduces a non-linearity by outputting y for all positive inputs y and 0 otherwise (Equation 4.4). This operation proves to be very computationally efficient by simply thresholding the matrix of activations at 0. As a result, the activation is more sparse and less calculation is being performed. Moreover, ReLU allows deeper networks to converge significantly faster (factor of 6 compared with an equivalent network with *tanh* neurons [88]) making them easier to optimise when compared with other functions such as sigmoid or *tanh* [116, 132]. This is because the gradients are able to flow when the input to the ReLU function is positive 4.4.

$$relu(y) = max(0, y) \tag{4.4}$$

Unfortunately, due to the *negative* part of ReLU being a horizontal line, this introduces a significant risk of *dying* units [175]. This is due to all negative inputs y becoming 0 (i.e. horizontal line) along with their gradient which means that such neutrons will stop responding to variations in error/input and *die* making a segment of the network *passive* [66]. This may significantly limit the ability of the network to properly train from the inputted data [66]. Moreover, ReLU activation can be very sensitive to higher learning rate values [175].

In order to alleviate the effect of vanishing gradients on the network performance, many alternatives including *leaky* ReLU [99] introduce a *leakage* parameter to the horizontal part of the ReLU graph (i.e. 0.01 for the Leaky ReLU). However, according to the studies [175, 99], a constant parameter value for *leaky* ReLU has a marginal impact on improving network performance when compared with the equivalent architecture using ReLU.

Therefore, we decided to extend the idea of *leakage* parameter and incorporated parametric rectified linear unit (PReLU) [66] which progressively learns such parameter for each input channel (α_i) yielding higher accuracy with a marginal extra computational cost. Due to the parameter adaptation for each channel, PReLU eliminates the vanishing gradients problem as well as reduces the risk of overfitting (Section 5.1.1) due to its randomness [175]. As a result, PReLU tends to outperform the equivalent architectures with ReLU unit in the most popular tasks such as CIFAR-10, CIFAR-100 [175] and ImageNet [66], significantly reducing testing error.

$$prelu(y_i) = max(0, y_i) + \alpha_i min(0, y_i) = \begin{cases} y_i & \text{if } y_i > 0 \\ \alpha_i y_i & \text{if } y_i \leqslant 0. \end{cases} \tag{4.5}$$

The exact formula for PReLU is presented in Equation 4.5, with the leaking parameter α_i controlling the slope of the negative half of its graph (Figure 4.3(b)) and subscript i indicating that the activation varies across different channels. In particular cases where $a_i = 0$, PReLU becomes a simple ReLU unit, whereas a fixed $a_i = 0.01$ denotes Leaky ReLU. Moreover, it is worth noting that PReLU introduces only a very small additional number of learnable parameters compared with their overall amount within the model, and is equal to the total number of channels which does not impact the length of the training process significantly [66].

$$\frac{\partial E}{\partial \alpha_i} = \sum_{y_i} \frac{\partial E}{\partial f(y_i)} \frac{\partial f(y_i)}{\partial \alpha_i} \tag{4.6}$$

PReLU is trained the same way as all other network layers using back-propagation. The formula for the α_i parameter update for one layer is presented in Equation 4.6 where E denotes the objective function and $\frac{\partial E}{\partial f(y_i)}$ being the gradient propagated from the deeper layer. As we can observe, it is simply derived from the chain rule requiring a marginal amount of additional computations. Moreover, we did not find any studies investigating PReLU's potential benefits to performance when used within the Faster R-CNN architecture which indicates the novelty of our research project. Our experiments showed PReLU's advantage, improving our detector's average precision on the test set by nearly 5% which is a significant boost in performance (Section 5.2.1).

4.1.4 Pooling layer

After a series of convolutional and activation layers applied on the input data and its resulting feature maps, the next step is to perform downsampling using a pooling layer. Downsampling reduces the number of connections to the following layers, hence decreasing the number of parameters within the network. This results in the computation speed-up and helps with controlling the overfitting [93].

The pooling operation is applied separately on each feature map and combines the input features within a small patch (i.e. $k \times k$ pooling region). The size of the pooling region varies with a possibility of an overlap between the regions depending on the stride (s) similarly as in the convolutional layer. The dimensions of the reduced feature maps generated by the pooling layer can be determined the formula in Equation 4.7 [44]. Therefore, for the non-overlapping pooling regions (i.e. $s = k$) of size $k \times k$, the dimensions of each $i \times i$ input feature map is downsampled to $\frac{i}{k} \times \frac{i}{k}$ for each channel [115].

$$o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1 \tag{4.7}$$

The most widely used types of pooling operations include taking the maximum value within each of the regions (i.e. max-pooling) or averaging across them (i.e. average-pooling) [14]. The potential disadvantage of the average-pooling comes from the risk of devaluating high activations since many low activations are averagely included [173]. Therefore, based on the original Faster R-CNN architecture and recent studies indicating max-pooling being the most optimal choice [115], we decided to choose this method as part of our final model. Based on the VGG-16 architecture [155], our choice was to reduce both dimensions of the feature maps by the factor of 2 after every pass through the pooling layer. This operation effectively reduces the number of parameters by 75% preserving the most relevant information within each pooling region (i.e. taking the maximum value) and is defined by the pooling size of 2×2 and stride of 2.

The main advantage of the pooling layer is that it greatly helps with controlling the overfitting by reducing the number of parameters within the model [93]. The smaller number of parameters also means that fewer computations need to be performed within the network resulting in a significant speed-up of the training process [115]. Furthermore, pooling helps the model to arrive with an almost scale-invariant representation of the input image [170] as well as becoming robust to small distortions and transformations within the input image (i.e. translation invariance). This is because we are taking the maximum value within a local neighbourhood (i.e. region) of the image and small variations of the input are unlikely to affect the output [79].

4.1.5 Dropout

One of the additional layers considered within our architecture was dropout [159]. Dropout is one of the most widely used methods for network regularisation. It randomly drops the defined fraction (with the fixed probability of 1-p) of activations at each

iteration to regularize the network. Dropout can be viewed as a combination of multiple models trained on different subsets of data [120]. Despite it being so successful, especially when combined with a fully-connected layer (in Fast R-CNN component), we did not include any dropout layer in our final implementation due to the following reasons regarding the stated layer combinations:

- **Convolutional layer**: is less likely to suffer from overfitting than other layers because the number of parameters for this layer is small relative to the number of activations [120].
- **Max-pooling layer**: dropout is inferior to the downsampling operation provided by the max-pooling layer, therefore, we did not consider combining it with any of the pooling layers [173].
- **Fully-connected layer**: this layer is part of the Fast R-CNN component described in Section 4.3. We chose to motivate our choices regarding dropout in this section for the sake of clarity of the report. Combining dropout with the fully-connected layer is the most widely used combinations, however, our experiments in Section 5.2.3 did not indicate any benefits to network performance even after increasing the learning rate by the factor of 10 as recommended by [159].

4.1.6 Batch normalisation

Another potential enhancement of the base network could be the introduction of the batch normalisation layer [77] which allowed many state-of-the-art architectures to improve their final performance [65]. Batch normalisation forces all network activations to take the unit Gaussian distribution throughout the entire training process by shifting its inputs to zero-mean and unit variance per each mini-batch, making the data comparable across all the features [117].

It is worth noting that a similar effect could be achieved by a data pre-processing step of shifting and scaling all the inputs before feeding them into the model. However, batch normalisation allows adjusting the data multiple times within the network after each pass through the batch normalisation layer [65]. This step supports the gradients in becoming more independent from the initial values of inputs and the scale of the hyper-parameters, thus making the network less sensitive to parameter initialisation and, as a result, easier to optimise, allowing for higher learning rates and faster training [77].

Due to the batch normalisation being invented over a year after the VGG-16 network implementation, the authors of its original architecture did not incorporate this method within their model [155]. However, as described in Sections 4.2 and 4.3, in case of the Faster R-CNN architecture [134], each mini-batch is formed from the proposals extracted from a single input image which undermines the original idea of per-batch normalisation (where the inputs are images of different characteristics). Some studies include a pre-trained batch normalisation layer to further improve its performance [74]. However, as explained in Section 5.1.1, due to the uniqueness of our task involving objects of very different characteristics we were not able to take the advantage of any batch normalisation layer pre-trained on other datasets.

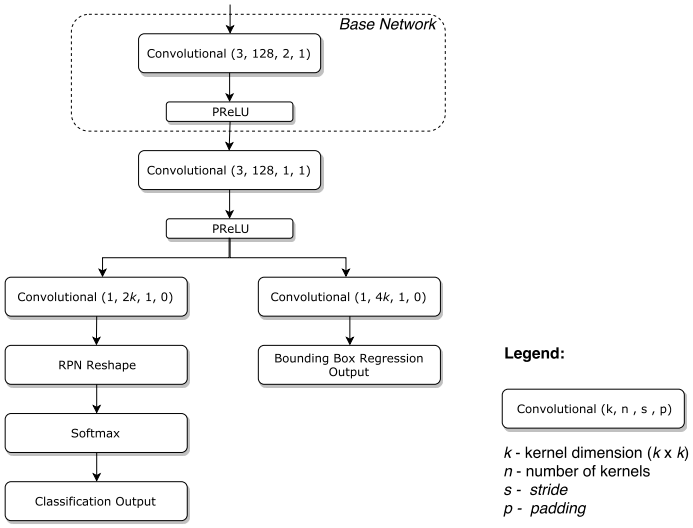


Figure 4.4: Region proposal network (RPN) architecture.

A possible solution could include training a separate base network on a classification task using crops of images containing single flower objects to adjust the parameters within the batch normalisation layer [74]. However, such approach would involve adding an extra stage to already complex training process (5.1), making the whole training procedure very time-consuming. Therefore, due to the limited computational resources as well as time constraints, we did not use batch normalisation within any of the Faster R-CNN components. Nevertheless, we aim to alleviate the effect of lack of batch normalisation layer to some extent by zero-centering the inputs (Section 4.1.1) and using per-epoch input shuffling (Section 5.1.1).

4.2 Region Proposal Network

Instead of using a separate method of region proposal extraction such as the selective search [169] used in Fast R-CNN [57], Faster R-CNN introduces a fully convolutional region proposal network (RPN). This gives the model a huge advantage over its predecessor resulting in 10 times detection speed-up per single frame. Like selective search, RPN aims to find a predefined number of regions (i.e. bounding boxes), which are the most likely to contain objects. At this stage, the network is not concerned about the specific class of an object since further classification is performed in the next stages within the framework, namely Fast R-CNN detector component (Section 4.3).

Another property of the RPN which gives the model a substantial advantage over the selective search method is the introduction of translation invariance allowing the network to detect the same object regardless of its scale and position within the image. This is because predicting raw coordinates of each object within an image might be greatly affected by the changing aspect ratio and size of the image [134]. Instead, RPN predicts the coordinates by learning to predict offsets (Δ) from reference boxes known as anchors [134].

4.2.1 Workflow

Anchors are fixed size bounding boxes of different sizes and ratios used as a reference during the prediction. Although anchors are defined based on the set of feature maps, the final object proposals reference to the original image [134]. When processing the input, anchors are placed throughout the feature maps implemented as the sliding-window. At each point, we simultaneously predict k region proposals by considering k different anchors with their centres being defined by the centre of the sliding-window. This method builds on the *pyramid of anchors* approach [134] which allows the network to rely on the images, feature maps and sliding-windows of a single scale being much more cost-efficient as opposed to using multiple-scale images and their feature maps per each scale and size as in OverFeat [142].

Relative sizes and ratios of the anchors greatly depend on the object's characteristics that we aim to detect. In the original Faster R-CNN implementation, the authors proposed an *anchor pyramid* of 3 different sizes and scales yielding 9 distinct combinations per window step donated by k (i.e. $k = 9$). However, due to the specificity of the task, our implementation introduces anchors of 5 sizes corresponding to very small flowers in the original image (i.e. 8-16 pixels in width and height). Such a change was motivated by the recent studies indicating the beneficial influence of smaller anchor sizes on the small object detection quality [45]. Moreover, we only considered anchors of a single $1:1$ ratio due to the circular shape of the flowers which fits within a square-shaped bounding box in most of the cases. We believe that this choice would not affect the results greatly but will significantly reduce time spent on proposal extraction due to fewer anchors being considered at each sliding-window step (i.e. k reduced to 5).

The sliding-window method has been implemented using a convolutional layer with 3×3 kernel, stride 1 and the zero-padding of 1 in order to prevent downsampling of the feature maps. After sliding over each point of the feature map stack, each window is passed through an activation unit and mapped onto a lower-dimensional feature (i.e. 128-d) within classification and regression part of the network in parallel (Figure 4.4). The classification part of the RPN aims to determine if the anchor contains an object outputting the *objectiveness score* which is later used to determine *bad* bounding boxes. Moreover, at this stage, the network is not concerned about the specific object classes in case of multi-class detection but rather aims to determine if an object of any class other than *background* is present within the bounding box.

During the classification stage, each window is mapped onto a lower-dimensional feature using a convolutional layer with 1×1 kernel. Such configuration acts as a fully-connected layer, making the RPN a fully-convolutional network and allowing inputs of any dimensions to be processed by it [148]. In the next step, the tensor is reshaped in order to compute softmax probabilities of the scores generated by the `Softmax` and `RPNClassicationLayer` layers. The output of the classification part of the RPN are $2k$ scores corresponding to the *background* and *foreground* probabilities per each of the k anchors of different sizes.

The regression part of the RPN is used to predict and adjust the offsets relative to the anchor coordinates corresponding to the actual position of each object within the image [134]. This part also contains a convolutional layer with 1×1 kernel whose output is passed through the `BoundingBoxRegressionOutputLayer` resulting in $4k$ outputs per each size of the anchor. Each of the four predictions generated at this stage correspond to offsets regarding coordinates of the top-left corner, height and width of the bounding box respectively (t_x, t_y, t_h, t_w in Equation 4.12). This, combined with the objective scores from the classification part, gives us a set of possible region proposals which can be further processed by the final part of the Faster R-CNN architecture to complete the detection (Section 4.3).

4.2.2 Post-processing

Due to the possibly very high number of all predicted region proposals, with most of them highly overlapping between each other, we need to post-process the extracted bounding boxes. Therefore, we remove all cross-boundary anchors which denote the boxes that do not fully fit within the image. Their inclusion would prevent the training from convergence since that might introduce large error terms to the objective function [134]. However, this step is applied only during the training phase because the objects that we wish to detect may not be fully included within the image.

In the next step, non-maximum suppression (NMS) is applied on the remaining proposals in order to remove highly overlapping proposals and merge those that belong to the same object [12, 73]. NMS algorithm takes $(proposal, score)$ pairs sorted in the ascending order according to their scores (generated by the classification part of RPN) and discards the proposals whose intersection over union (IoU) is greater than 0.7 with a proposal of a higher score.

IoU, also known as Jaccard index, is defined as the size of the intersection between two bounding boxes (A and B) divided by the size of their union [131] yielding a score of 1 if the boxes overlap perfectly and 0 if there is no overlap between them (Equation 4.8). Moreover, we chose the threshold of 0.7 based on the original implementation of the Faster R-CNN [134] which is a suitable compromise between missing too many proposals for objects (i.e. small value) and not eliminating enough overlapping proposals (i.e. high value).

$$IoU(A,B) = \frac{|A \bigcup B|}{|A \bigcap B|} \tag{4.8}$$

The final post-processing step involves post-NMS sorting of the proposals and choosing N of them with the highest scores. The specific value of N introduces a trade-off between the quality and time spent on the detection. Our choice of $N = 2000$ was based on recent studies concerning small object detection indicating the efficiency of such approach [45]. Despite other experiments conducted in [134], indicating that even the value of N set to 300 may yield satisfying results, we decided not to choose such a small value in order to assure high accuracy of our final model.

4.2.3 Loss function

Before calculating the loss function for each of the images, the proposals are divided into *positive* and *negative* samples based on the IoU threshold with the ground-truth boxes. We used the default values for both thresholds meaning that all proposals of the IoU with the ground-truth lower than 0.3 are considered as the *negatives* and those with the IoU higher than 0.7 as *positives*. All other regions are not considered in calculating the loss function to assure that the network does not learn on ambiguous cases [134].

As a next stage, a randomly selected set of proposals is chosen to form a mini-batch used to calculate the loss for each of the parts. The ratio of 1 : 1 between the positives and negatives is used to ensure that the network is not biased towards the negative examples due to their much higher presence within the set of all generated proposals. However, if there are not enough positives, the mini-batch is supplemented with the leftover proposals with the highest IoU with the ground-truth. Although we realise that this is not a perfect solution due to a potential risk of selecting very *weak* regions, this approach allows us to always generate positive samples and yields satisfying results [134].

A crucial aspect of every network is its loss function which we aim to optimise. RPN loss function is represented by Equation 4.9 and consists of two terms (L_{cls} and L_{reg}) corresponding to classification and regression losses respectively. These losses are calculated within each part of the RPN.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i L_{reg}(t_i, t_i^*) \tag{4.9}$$

After differentiating the proposals, we use both types to calculate log-loss (L_{cls}) over two classes denoting *object* and *not object*. In the equation 4.10, i represents the index of an anchor within the mini-batch and the predicted probability p_i of anchor i being an object. Moreover, p_i^* is the ground-truth label which is equal to 1 for positive and 0 for negative anchors.

$$L(p_i) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) \tag{4.10}$$

Equation 4.11 represents regression loss function for which only positive proposals are considered. Here, t_i represents a vector containing four parametrised coordinates (t_x, t_y, t_h, t_w) of the predicted bounding box whereas t_i^* is a set of ground-truth bounding box coordinates for the positive example [58]. The parametrisations of the four coordinates are described by the equation 4.12 in which t specifies a scale-invariant translation and log-space height/width shift relative to an object proposal [57].

$$L(t_i) = \lambda\frac{1}{N_{reg}}\sum_i L_{reg}(t_i, t_i^*) = \lambda\frac{1}{N_{reg}}\sum_i smooth_{L1}(t_i - t_i^*) \tag{4.11}$$

$$\begin{gathered} t_x = \frac{x - x_a}{w_a},\ t_y = \frac{y - y_a}{h_a}, \\ t_w = \log(\frac{w}{w_a}),\ t_h = \log(\frac{h}{h_a}) \end{gathered} \tag{4.12}$$

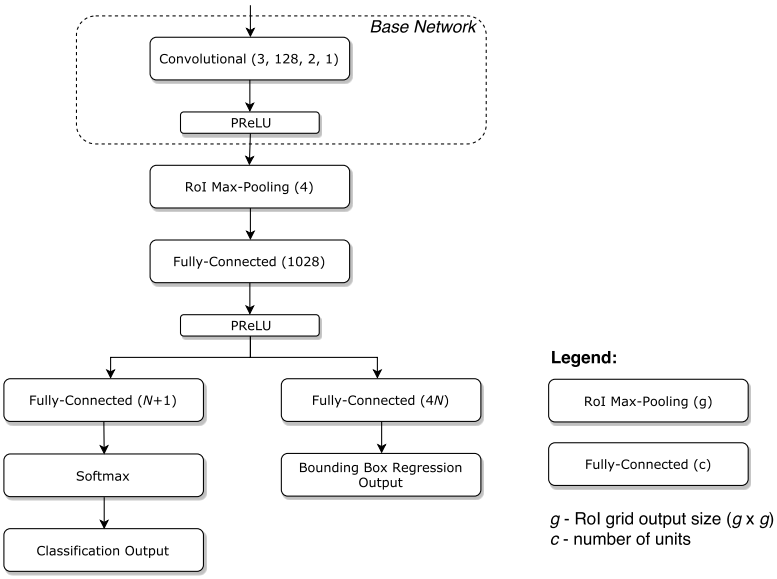


Figure 4.5: Fast R-CNN detector component architecture.

Moreover, *smooth* L_1 loss function defined in equation 4.13, is used for the regression. This particular type of loss tends to be more robust and less sensitive to outliers than the L_2 loss used in the R-CNN [58] and SPPnet [67]. L_2 loss is much more sensitive to higher learning rate values resulting in exploding gradients [26]. Moreover, when combined into a single loss function, both classification and regression terms are normalised by the mini-batch size (N_{cls}) and the overall number of anchor locations (N_{reg}) respectively. Additionally, the regression loss is weighted by a λ parameter set to 10. This ensures that both terms are equally weighted within the loss function.

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{4.13}$$

4.3 Fast R-CNN

The last component of the Faster R-CNN architecture is the Fast R-CNN detector [57]. Once all region proposals (region of interest; RoIs) had been extracted and pre-processed, they can be now fed into the Fast R-CNN for further classification and bounding box refinement [62].

4.3.1 Workflow

One possible approach to region proposals classification would be to use them to crop segments of the image, pass each crop through the base network and apply an image classifier. However, such procedure would be very computationally expensive due to multiple passes through the network. Instead, Fast R-CNN makes the process more

efficient by reusing the feature maps of the whole image extracted by the base network [26]. Hence, the inputs to the Faster R-CNN are RoIs generated by the RPN and a set of feature maps from the base network. The proposals are needed to extract specific parts of the feature map containing an object.

The next step is fixed-sized feature map extraction for each proposal, using region of interest pooling (ROI-pooling layer) [134]. This step is essential since the proposals are of the different sizes which cannot be processed by the fully connected layer in the next stages. ROI-pooling uses the same principle as max-pooling, extracting the maximum value within each of the sub-windows being applied independently to each feature map channel [34]. However, the essential difference is that ROI-pooling outputs a feature map of a fixed spatial extent of $h \times w$, with h and w being layer hyper-parameters independent of any particular RoI.

RoI pooling layer is a special case of the spatial pyramid pooling layer used in SPP architectures [67] consisting of only one pyramid level which is more computationally efficient [134]. Furthermore, due to a very small size of the objects that we aim to detect, we reduced the default 7×7 dimensions of the grid produced by this layer to 4×4. This is determined by the `InputSize` within `ImageInputLayer` object which we set to `[16 16 3]` accounting for two pooling operations within the network (i.e. max-pooling and RoI-pooling), yielding the final size of the grid of 4×4 [108]. This input size is a balance between processing time and the amount of spatial detail the CNN needs to resolve. The resulting fixed-sized features for all of the proposals are then flattened to produce a feature vector using a fully connected layer followed by the activation unit. This allows us to proceed to the final steps of classification and regression.

The aim of the classification part of the Fast R-CNN is to further classify the proposals into one of the classes and remove those classified as *background*. It consists of a single fully-connected layer with $K + 1$ units followed by the softmax and classification output layer which outputs the probabilities of the proposal belonging to each of the classes. K donates the number of classes and is equal to one in this case, with the additional class representing the *background*.

Regression is applied on the bounding boxes for their further refinement. In order to do so, we need to take into account the class with the highest probability for each proposal [57]. Moreover, at this stage, the proposals with the highest probability of belonging to the *background* class are discarded. The regression part of Fast R-CNN consists of a fully connected layer with $4K$ units followed by the regression output layer. Each of the four units correspond to the offsets (t_x, t_y, t_h, t_w; Equation 4.12) for each of the K classes ($K = 1$) similarly as in the RPN.

4.3.2 Post-processing

Like in the RPN, the final proposals need to be post-processed in order to ignore those with a very high overlap, which most likely belong to the same object. This is done by sorting the proposals by their probability in ascending order (grouped by class) and applying NMS with IoU threshold of 0.7. Furthermore, the regions with the probability score lower than 0.5 are discarded. This is done to ensure that only the most probable objects are being detected. The specific value of 0.5 follows widely used standards applied in other detection tasks such as PASCAL VOC [50].

4.3.3 Loss Function

The loss function for the Fast R-CNN component follows the equations for the RPN where we use a multi-task loss on each labelled RoI to jointly train for classification and bounding-box regression (Equation 4.9). The usual difference between Fast R-CNN and RPN losses (Section 4.2.3) is that the former distinguish between different classes (i.e. multi-class log loss for classification) which is not the case in our task (i.e. only *flower* class), hence, the equations do not change.

Another difference is the values of IoU thresholds with the ground-truth boxes to distinguish between the positive and negative samples. Originally, their values were set to 0.5 and 0.1 respectively, meaning that the samples with the IoU $\geqslant 0.5$ were considered as positives and those with the IoU between 0.1 and 0.5 as negatives. This appears to act as a heuristic for hard example mining [52], helping the network to learn on harder cases. However, according to other studies [152], the lower 0.1 is suboptimal because it ignores some infrequent, but important, difficult background regions. Therefore, in order to improve the detection further, we decided to set the lower threshold to 0, meaning that the proposed boxes of the IoU over 0.5 are considered as *positive* and the remaining boxes as *negative*.

Moreover, in case of the Fast R-CNN component, the mini-batch consists of 25% positive (i.e. IoU $\geqslant 0.5$) and 75% negative (i.e. IoU < 0.5) proposals. This relative positive/negative ratio imbalance allows the network to learn from more ambiguous examples [152]. Furthermore, the samples used to form the mini-batch belong to a single image and follow MATLAB implementation of `fastRCNNObjectDetector` class [107]. This is more a computationally efficient approach than using samples from two images per mini-batch [152] since gradient accumulation across multiple batches is slower and requires additional operations [24].

Chapter 5

Training and Experiments

The purpose of this chapter is to familiarise the reader with the Faster R-CNN complex training procedure as well as explain and justify our choices regarding various hyper-parameter values (Section 5.1). In Section 5.2 we will discuss the results of our experiments using different set-ups (i.e. architectures) in order to further support our design choices.

5.1 Training

As explained in Chapter 4, Faster R-CNN consists of multiple independent components, each responsible for a different task, making the overall training process more complex when compared with the standard sequential architectures [155, 88, 39]. Therefore, we will describe and explain the training process of Faster R-CNN and motivate our choice of hyper-parameters.

Rather than using a one-stage class-specific detection strategy presented in architectures such as OverFeat [142], Faster R-CNN utilises a two-stage pipeline based on the class-agnostic proposals and class-specific detections. Unlike the Faster R-CNN, OverFeat performs simultaneous proposal extraction and classification. Although both architectures use sliding-window approach to extract region-wise features, Faster R-CNN separates the tasks of region proposal extraction and bounding box refinement into two subsequent stages carried out by its distinct units (i.e. RPN, Fast R-CNN). This results in the separate components specialising in each of the tasks and allowing the region-wise features to be adaptively pooled (i.e. RoI pooling layer) from proposal boxes that cover the features of each region more conscientiously.

Despite the longer training time required by the networks following the two-stage approach, such models tend to yield considerably better results with a 5% mAP increase (53.8% to 58.7%) compared to their one-stage counterparts trained on the same dataset [134, 74]. Although region proposal network (RPN) and Fast R-CNN detector focus on different tasks, namely region proposal (i.e. RoI) extraction and further classification and bounding box adjustment, they both require the set of feature maps extracted by the

base network. As a result, while trained separately, both of these components modify the convolutional layers of the base network differently [134]. Therefore, it is essential to develop a technique that allows them to share the base network layers rather than training two separate models which would result in increased number of learnable parameters and consequently longer training [26].

Our solution, inspired by the original Faster R-CNN paper [134], introduces an alternating training strategy. In this method, we first trained the base network with the RPN and used the generated proposals to train a newly initialised base network with the Fast R-CNN. The convolutional layers tuned in the previous step were then used to initialise the RPN and the process iterated (see stages below). We followed the 4-step training process due to the negligible improvements and a notable increase in training time achieved after more iterations [134].

In the first two iterations, we trained the combination of base and RPN networks (Stage 1) and used the generated proposals to train the merged base and Fast R-CNN networks (Stage 2). Up to this point, these two components (i.e. RPN, Fast R-CNN) had not been sharing the base network layers which were initialised at the beginning of each stage. This way we trained both components independently allowing them to modify the parameters within the base network layers accordingly. Next, we combined the previously created networks in order to construct a single pipeline. More precisely, we used the fixed base network layers trained in the previous step to fine-tune the RPN layers (Stage 3) which resulted in a unified architecture (i.e. shared base network). Lastly, the region proposals generated by the updated RPN were used to fine-tune (i.e. re-train) Fast R-CNN detector layers (Stage 4). This way we ensured that only good quality proposals were used by the Fast R-CNN for further classification and bounding-box regression [180]. Therefore, the Faster R-CNN training procedure consisted of the following four stages:

- Stage 1: Training a region proposal network (RPN)
- Stage 2: Training a Fast R-CNN Network using the RPN from step 1
- Stage 3: Re-training RPN using weight sharing with Fast R-CNN
- Stage 4: Re-training Fast R-CNN using updated RPN

Due to the distinct purpose of each training stage, respective networks are likely to have different convergence rates [6]. A very common approach indicates using lower learning rates for the last two stages of training since we only fine-tune the model [183]. This allows the network weights to be modified more slowly, reaching the optimal point configuration in a more controllable manner. More details regarding specific hyper-parameter set-ups are provided in Sections 5.1.1 - 5.1.4.

5.1.1 Stage 1: Training a region proposal network

In the first stage of the Faster R-CNN training process, we combined the base network and the RPN in order to extract object proposals which are later used by the Fast R-CNN component to perform the detection. This is achieved by extracting feature maps from

each of the images by the base network and feeding them into the region proposal network (RPN) for the region of interest extraction.

Because the original Faster R-CNN model was used to detect standard objects (cats, cars or people), the authors took the advantage of the VGG-16 architecture [155] pre-trained on the ImageNet dataset [138] as the base network. This approach, also known as transfer learning, allows an already trained network to be used on similar tasks, making the process less computationally expensive since the network only needs to be fine-tuned rather than trained from scratch to adjust its parameters to the new task [58]. However, due to the *E. vaginatum* flower objects being significantly smaller than any items considered by the standard datasets, we were unable to incorporate any pre-trained models as feature extractors. Therefore, we trained the combination base and RPN networks from scratch, randomly initialising all layers by drawing weights from a Gaussian distribution with standard deviation of 0.01 and zero mean.

Appropriate weights initialisation is critical for the model convergence and its final quality [89]. If not done properly, the deeper layers may receive inputs with small variances, which in turn slows down back-propagation, consequently obstructing the overall convergence process [174]. An intuitive approach would be to initialise all the values to zero. However, this would make the neurons to compute the same output resulting in the same gradients during back-propagation and passing the exact same information preventing the network from learning anything from the input data [86]. In order alleviate such scenario, a common method suggests to randomly sample the weight vector form multi-dimensional Gaussian distribution making the neurons to point in a random direction of the input space, hence preventing vanishing gradients [89].

We initialised weights of each layer by drawing from a zero-mean Gaussian distribution with standard deviation 0.01 which is a very a widely used configuration, successfully applied in many popular architectures [88, 57, 142, 66] including the original Faster R-CNN [134]. We are aware of other initialisation methods such as *Xavier* which adopts a properly scaled uniform distribution for initialisation [59], however, we consider such methods to be out of the scope of this project. We also decided to initialise all biases to zero [66, 134]. Such approach can be justified by the necessary asymmetry between the neurons being already provided by the weights initialisation to small random numbers [70].

The network was trained using mini-batch gradient descent algorithm with the momentum of 0.9. In case of the standard gradient descent algorithm (GD), network parameters (i.e. biases and weights) are updated using a subset (mini-batch) or the entire training set (batch) in order to minimise the loss function (Section 4.2.3) by taking minute steps towards the negative gradient of the loss at each iteration [161]. The size of the step is determined by the learning rate parameter α. The GD algorithm is defined in Equation 5.1 where θ_i and $\nabla E(\theta_i)$ denote the parameter vector at iteration i and gradient of the loss function $E(\theta)$ respectively [136].

$$\theta_{i+1} = \theta_i - \alpha \nabla E(\theta_i) \tag{5.1}$$

GD algorithm might experience significant oscillations along its path towards the optimum point due to the negative gradient always pointing to the steepest sides rather than along the most optimal direction [129]. GD with momentum (Equation 5.2), on the other hand, helps with accelerating the gradient vectors in the appropriate direction by considering the contribution of the previous gradient step to the current iteration determined by the momentum parameter γ, thus leading to the faster model convergence [136]. This property makes it one of the most popular optimisation algorithms adopted in numerous of state-of-the-art architectures [161, 57].

Despite many existing alternatives including *AdaGrad* [43], *RMSProp* [36] or *Adam* [84], we decided to follow the choice suggested by the authors of the original Faster R-CNN architecture [134] and adapt the GD with momentum optimisation algorithms in our network. We also set the γ parameter to 0.9 which is the most standard and, in the majority of the cases, most optimal value [136].

$$\theta_{i+1} = \theta_i - \alpha \nabla E(\theta_i) + \gamma(\theta_i - \theta_{i-1}) \tag{5.2}$$

In the standard GD algorithm, mini-batch is defined as a subset used to update the network parameters. Recent approaches include considering a batch consisting of the whole training set (i.e. batch gradient descent). However, such approach might slow down the training or be completely unfeasible for bigger datasets which do not fit in the memory [136]. Moreover, this way we perform many redundant computations due to recomputing gradients for very similar examples before each update [136].

Stochastic GD, on the other hand, updates parameters for each example (i.e. mini-batch size = 1) which allows the model to avoid premature convergence (i.e. being stuck in the local minimum) due to the frequent updates [13]. However, the increased amount of steps may ultimately complicate the convergence to the optimum point as the stochastic approach keeps overshooting. Furthermore, very frequent updates make the training process computationally more expensive, thus increasing the amount of time needed to train the network [95].

Mini-batch GD compromises two previous approaches by dividing the training set into smaller subsets (i.e. mini-batches) which are then used for error calculation and parameter update [13]. The batched approach provides a more computationally efficient strategy for parameter update than its stochastic counterpart due to the decreased amount of steps (i.e. updates) [11]. Moreover, mini-batch GD significantly reduces the parameter updates variance resulting in more stable convergence as well as does not require to store the whole data in the memory as for the batch GD [95].

Mini-batch GD algorithm introduces a new hyper-parameter determining the size of the mini-batch. Mini-batch sizes vary between 50 to even 256 for most state-of-the-art networks, depending on the architecture and data used [57, 67, 65]. Larger values of the batch size result in the slower learning process, which could be partially alleviated by the parallelisation across CPU/GPU cores, leading to a more accurate estimate of error gradient. Small values, on the other hand, speed-up the convergence compromising the stability of the training [11]. Hence, the mini-batch size hyper-parameter introduces a trade-off between the accuracy (i.e. relevance of the proposals) and the time required

to complete the training. Therefore, we set its value to 256 as in the original Faster R-CNN architecture [134]. We did not observe any substantial benefits of increased batch size other than longer training time. Hence, we consider this value as optimal.

However, the standard definition of the mini-batch differs for the Faster R-CNN architecture [134]. In case of the RPN, mini-batch consists of many positive and negative example anchors obtained from a single image (Section 4.2). The ratio between positive and negative samples is set to 1 : 1 in order to avoid the bias towards negative samples which are dominant in number. However, if the number of *positives* is not sufficient, the mini-batch is supplemented with the leftover proposals of the highest IoU with the ground-truth (Section 4.2.3).

The next considered hyper-parameter was the number of epochs determining the length of the training process which is greatly dependent on the network architecture (i.e. depth, number of parameters, etc.) [11]. A single epoch is defined as a full pass of the algorithm through the entire training set [157]. Due to the limited size of our training set, one epoch was not enough to deliver a well-trained network able to generalise to new examples (i.e. underfitting) [11]. An increased number of epochs results in more parameter updates, thus leading to a finer trained model with a risk of overfitting when the amount of epochs is too high [2].

Overfitting is a phenomenon where a network is unable to capture the underlying structure of the data, simply memorising previously seen examples (i.e. training set) [2]. One of the methods to combat overfitting is to introduce a validation set. Validation set consists of examples separate from the training and testing sets and is used to provide an unbiased evaluation of the network fit on the training set enabling hyper-parameter tuning [16]. This set can be used in one of the regularisation methods such as early stopping [11] where the training is stopped as soon as the error on the validation set starts to increase signifying network overfitting.

However, we were unable to take advantage of the validation set due to the lack of such option provided by the `trainFasterRCNNObjectDetector` method at the time of the network implementation. We consider incorporating the validation set to our training process as a potential future enhancement which might help with combating the potential network overfitting. Nonetheless, based on our initial training run of length of 100 epochs (Appendix C), we chose to train the RPN for 60 epochs due to deteriorating network performance when trained for longer (i.e. 0.04 drop in AP on test set for 100 epochs), indicating data overfitting.

Learning rate determines the size of the steps taken to reach the minimum point of the objective function and is often considered as the single most important hyper-parameter [157]. If set too high, such learning rate may prevent the training from converging since the parameter changes can become extremely big, risking the possibility of overshooting by the optimiser and increasing the value of the loss. A very low value, on the other hand, can significantly slow down the training process due to the very small steps taken towards the minimum of the loss function [136]. Therefore, we considered determining the optimal learning rate value as crucial for our network final performance.

The original Faster R-CNN architecture [134] had been trained using the initial learning rate of 0.001, however, we found this value too high for our network. This was indicated by very high mini-batch loss, eventually leading to extreme values that could not be represented in a single-precision floating-point format (i.e. `NaN` value) suggesting optimiser overshooting [11]. Hence, we set the initial learning rate to 0.0001 as this value indicated fast and relatively stable network convergence (Figure 5.1). Much lower value of this parameter, when compared with the original Faster R-CNN set-up, might be the result of lack of dropout layer in our architecture [159]. Dropout introduces additional noise in the gradient when compared to the standard gradient descent algorithm resulting in gradients cancelling with each other. In order to alleviate the effects of cancelling gradients, it is recommended to increase the optimal learning rate (i.e. when no dropout used) by the factor of 10 or even 100 [159] (Section 5.2.3).

Moreover, we further decreased the initial learning rate by the factor of 10 after 40 epoch of training. This strategy, used in many state-of-the-art models, allowed for more stable network convergence [57, 65]. As the learning rate decays, the optimiser takes smaller steps, hence reducing the risk of overshooting and eventually settling into a minimum [11]. The exact value of the *decay step* (i.e. 40 epoch) was determined by the negligible training progress of the RPN after 40 epochs. Despite other learning rate decay strategies involving adaptive learning rate methods [11], our simplified approach yielded satisfactory results which made us integrate it in our final architecture.

We have also introduced a weight decay parameter ($L2$ regularisation) of 0.0005 [159]. The regularisation term is added to the loss function (i.e. $E(\theta)$) in order to limit freedom of the model by penalising large weights [11]. Weight decay limits the number of free parameters, thus preventing the network from overfitting. It also suppresses some of the effects of static noise on the targets improving generalisation ability of the network [114]. Therefore, weight decay helps the network with better generalisation, limiting the need for the dropout layer [69]. Equation 5.3 presents this concept in more detail, introducing the notion of regularisation factor λ and the regularisation function $\Omega(w)$ (Equation 5.4) with w being the weight vector. We chose the specific parameter value (i.e. 0.0005) based on the widely used choices incorporated in other state-of-the-art architectures [88, 134].

$$E_R(\theta) = E(\theta) + \lambda\Omega(w) \tag{5.3}$$

$$\Omega(w) = \frac{1}{2}w^T w \tag{5.4}$$

Furthermore, we decided to shuffle the input tiles after each completed epoch of training in order to further minimise the risk of data overfitting. This way we prevented the network from learning on the exact same sequence of tiles at each epoch aiding to achieve better network generalisation [92].

Figure 5.1 presents the training process expressed in the accuracy plot of the RPN in the first stage with the specific hyper-parameter values discussed above. We provided the accuracy plots during all four stages for the same network trained for 100 (stages 1 and 2) and 50 (stages 3 and 4) epochs in the Appendix C. The accuracy plot in Figure 5.1 indicates that our RPN combined a newly initialised base network for feature extraction achieves a very satisfactory level of performance of nearly 98% on the training set

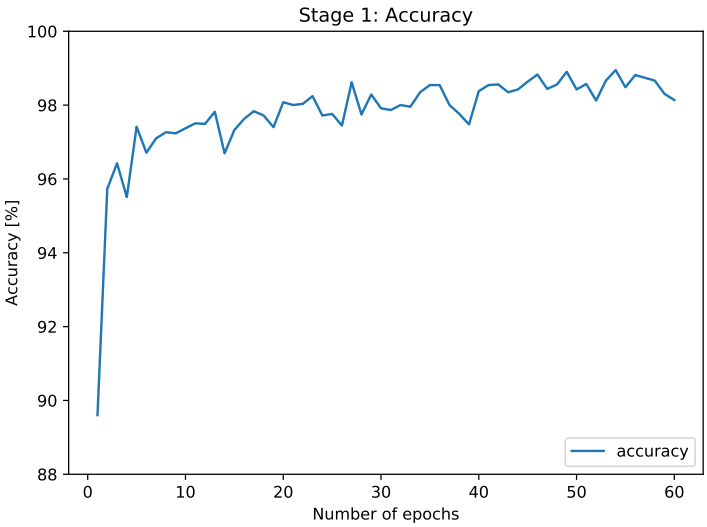


Figure 5.1: Training process expressed as an accuracy plot (i.e. averaged mini-batch accuracies for each epoch) of the RPN in the first training stage.

within just 20-30 epochs suggesting the appropriate hyper-parameter and architectural choices. We do not observe any further improvement in region proposal extraction performance after more number of epochs which has been expressed by the plateau and temporary fluctuations in accuracy.

Nonetheless, we can see a slight impact of the initial learning rate decay after 40 epochs (i.e. dropped by the factor of 10), resulting in further improvement in accuracy to up to almost 99% between 40 and 60 epochs. This can be attributed to the optimiser taking smaller steps (i.e. lower learning rate), hence reducing the risk of overshooting and allowing further training progress. Such a high accuracy achieved by our region proposal network at this stage of training suggests its ability to extract good quality proposals which are crucial for the Fast R-CNN detector component in the next stage.

5.1.2 Stage 2: Training Fast R-CNN using the RPN from step 1

The next stage of the training process was to train the combined base and Fast R-CNN components to perform the detection using the region proposals extracted by the RPN in the previous stage [134]. The Fast R-CNN component aims to further classify the proposals into specific classes (i.e. only *flower* class in our task), removing ones which represent the background as well as refine each of the bounding boxes.

The original Faster R-CNN architecture [134] used a newly initialised VGG-16 [155] architecture, pre-trained on the ImageNet dataset [138] as the base network. The reason for not using the base network layers trained in the previous stage is that when trained independently, RPN and Fast R-CNN modify these layers differently since they concentrate on different tasks. As in the previous stage, we were not able to take advantage of any pre-trained architecture due to the specificity of our task. Therefore,

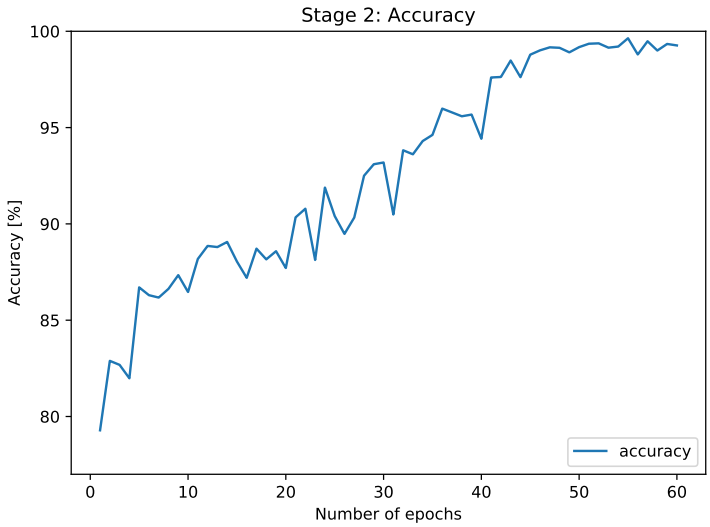


Figure 5.2: Accuracy plot illustrating the training progress of the Fast R-CNN component during the second stage of training.

we initialised all base network's parameters by drawing weights from a Gaussian distribution with standard deviation of 0.01 and zero mean and equalled all bias terms to 0. The same strategy has been applied to all Fast R-CNN detector's parameters (i.e. weights and biases) which follows the training regime of the original architecture [57].

We trained our detector component following the strategy from the preceding stage-using the mini-batch gradient descent algorithm with the momentum of 0.9, weight decay of 0.0005 and the mini-batch of size 256. The mini-batch denotes the number of proposals extracted from each input image to perform the detection. As described in Section 4.3.3, in case of our implementation, the proposals used to form a mini-batch belong to a single tile, which tends to be more computationally efficient than using samples from two tiles [57] due to the gradient accumulation across multiple batches being slow and requiring additional operations [24].

Therefore, following the recommendations in [24], we kept the original mini-batch size of 256 consisting of proposals extracted from a single tile. Such approach results in a relatively stable and efficient training process as indicated in Figure 5.2. Finally, due to the very similar training patterns for both components (i.e. RPN, Fast R-CNN), we kept the same epoch number of 60 and the initial learning rate of 0.0001 decaying by the factor of 10 after 40 epochs. These values combined with previously described hyper-parameters yield the training process depicted in Figure 5.2.

As indicated by the accuracy plot in Figure 5.2, Fast R-CNN detector component was able to progressively learn throughout the training process suggesting appropriate architectural and hyper-parameter choices. The detector's accuracy tends to improve more steadily than in case of the RPN in the previous stage which experienced a rapid performance rise at the very beginning of training. We can also see that the Fast R-CNN took more epochs to achieve the same level of accuracy as the RPN which might demonstrate higher complexity of the detection task compared with proposal extraction.

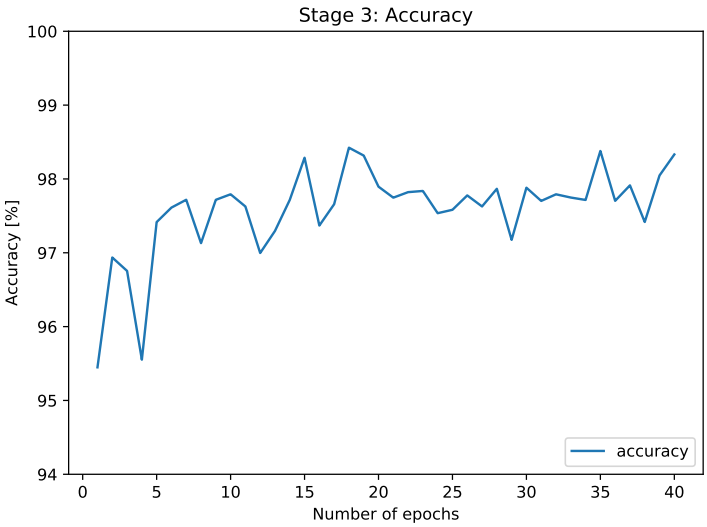


Figure 5.3: Accuracy plot illustrating the training progress of the fine-tuned RPN during the third stage of training.

Moreover, the plot shows slight fluctuations in the network's accuracy. Substantial drops at some epochs (i.e. 23, 26, 30) might indicate training instability caused by the learning rate being too high (i.e. optimiser overshooting). Nonetheless, the overall trend is constantly increasing with a noticeable jump in accuracy after 40 epochs attributed to the initial learning rate decay allowing the learning process to progress further and eventually settle very close to the optimum point after about 45-50 epochs. At this point, the detector component achieved nearly 100% accuracy suggesting excellent performance and ability to correctly classify and refine the bounding boxes.

5.1.3 Stage 3: Re-training RPN using weight sharing with Fast R-CNN

After separate training of RPN and Fast R-CNN components using newly initialised base network's layers, all parts were finally combined into a unified object detection network. Therefore, during the third stage of the training process, we used the base network with fixed parameters obtained in the previous stage (i.e. shared base network) in order to fine-tune the unique layers of the RPN resulting in further quality improvement of the proposals generated by this component. Moreover, this stage was crucial for obtaining a unified detection network, enabling end-to-end training through parameter sharing of the base network layers.

Similarly to all previous stages, we re-trained the RPN using the mini-batch gradient descent algorithm with the momentum of 0.9, weight decay of 0.0005 and the mini-batch of size 256. During this stage we only fine-tuned a pre-trained RPN network, thus assuming that its parameters have already been learned which enabled the network to extract good quality proposals. Therefore, we decreased the initial learning rate by the

factor of 10 allowing the network parameters to be modified more slowly, reaching the optimal configuration in a more controllable manner.

Moreover, we kept a similar learning rate decay strategy as in the previous training stages (i.e. this time decaying after 30 epochs) in order to allow the network to steadily proceed towards the optimal point [183]. We also decreased the number of epochs to 40 due to the fine-tuning purpose of this training stage and no further gains in network's performance observed after training for a higher amount of epochs which was also likely to prevent the network from further overfitting (Appendix C).

As presented in Figure 5.3, the accuracy graph for the fine-tuned RPN is relatively stable and does not indicate any substantial growth as recorded in the previous stages. That can be explained by the fact that, at this stage, we only fine-tuned the RPN to adjust it to the fixed base network trained in the preceding stage. Moreover, a relatively high initial accuracy of approximately 95% suggests that the fixed shared convolutional layers of the base network were capable of high-quality feature extraction allowing the RPN to generate relevant proposals. The lack of further improvement in accuracy supports our initial decision to decrease the number of epochs for the fine-tuning stages (i.e. reduction from 60 to 40), limiting the risk of data overfitting. Furthermore, the training accuracy plot does not show any noticeable contribution of decaying the learning rate after 30 epochs. Nonetheless, the training was still able to progress marginally.

5.1.4 Stage 4: Re-training Fast R-CNN using updated RPN

The last stage of the Faster R-CNN training process involved fine-tuning the layers belonging to the Fast R-CNN detector component. We used the same fixed base network as in Section 5.1.3 combined with the Fast R-CNN part and region proposals generated by the fine-tuned RPN. This way we ensured that only good quality proposals were used by the detector for further classification and bounding-box regression [180]. Moreover, we kept the same values of all hyper-parameters as in the preceding stage due to the stability and efficiency of the training process as well as very similar network's behaviour during the fine-tuning step (Figure 5.4).

As indicated by the training plot in Figure 5.4, as in stage 2, the network constantly improved throughout the fine-tuning process achieving the highest accuracy of almost 98% after 38 epochs with no further improvement when trained for longer (Appendix C). Such a high accuracy suggests very good network performance on the training set. Furthermore, the initial network accuracy was significantly higher (i.e. 89.3% compared with 78.6%) than in the second stage when training the network from scratch. This shows that the Fast R-CNN was already well trained in the second stage and only needed to be fine-tuned on the new proposals extracted by the improved RPN from the previous stage. Moreover, we did not observe any immediate rise in accuracy after decaying the learning at 30 epoch which might be attributed to its extremely small value of 0.00001 after the decay resulting in minute steps. Nonetheless, the training kept progressing indicating appropriate hyper-parameter set-up and training strategy.

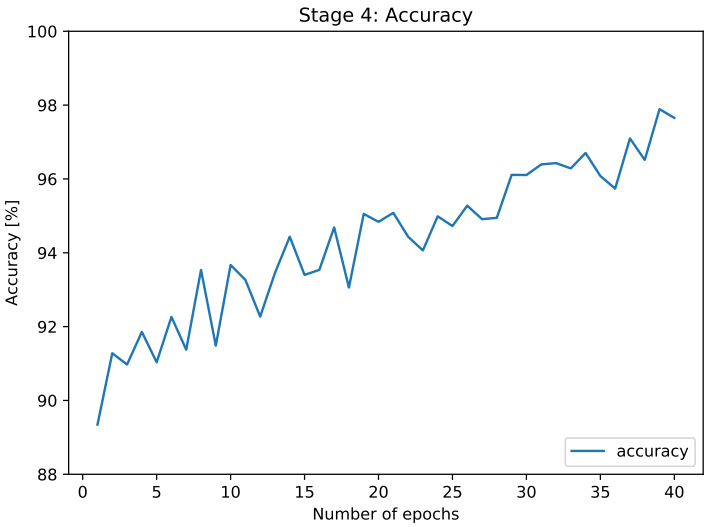


Figure 5.4: Accuracy plot illustrating the training progress of the fine-tuned Fast R-CNN detector component during the fourth stage of training.

5.1.5 Requirements

Our network was implemented in MATLAB [109]. We have used 2017*b* MATLAB release. However, our code is compatible with any MATLAB version starting from 2017*a* [109]. This is due to the `fasterRCNNObjectDetector` MATLAB class [106] being available from the 2017*a* realise. Moreover, we trained our network using the *Super Shrub* workstation located at the School of Geosciences at the University of Edinburgh (co-owned by the Team Shrub research group [153]), with the following specifications:

- 2 Nvidia Geforce GTX 1080 TI GPUs with 11GB of memory and compute capability of 6.1 [118]
- Intel Xeon E5-2640 v4 2.40 GHz processor [76]
- 256 GB of RAM memory
- 2 TB of disk space
- 64-bit Microsoft Windows 10 Pro operating system [113]

Due to our code being fully GPU-compatible, we were able to achieve an extreme computational speed-up when compared with training the network on a single or multiple CPU cores. However, we could not take the advantage of running our model on multiple GPUs due to the lack of such option for MATLAB `trainFasterRCNNObjectDetector` method [110] at the time of the implementation. We consider introducing such feature as a potential enhancement in the next part of our project (i.e. year 2) which would allow us to significantly reduce the time required to complete the training process of our network.

5.2 Experiments

In this section, we present and discuss the results of our experiments using different architecture set-ups in order to justify our design choices. For the evaluation between networks, we used precision-recall curves and the average precision (AP) performance metric denoting the area under the curve. AP is calculated by averaging the precision values of the precision-recall curve across all values of recall between 0 and 1, thus being equivalent to the area under the curve [184]. Hence, the graph of the precision-recall curve indicates the trade-off between precision and recall for different thresholds with a bigger area under the curve indicating both higher recall and precision. We provide a detailed explanation of the precision-recall curve and AP metric in Section 6.1.2.

Although AP is not suitable for evaluating human-to-model performance due to the lack of detection confidence scores for human annotators (see Section 6.1.2), it is an appropriate metric to compare the performance between models since all the scores are generated, allowing AP to summarise both precision and recall at all thresholds [184]. Moreover, we used AP due to its popularity in evaluating other object detection tasks [50, 57, 134, 19].

5.2.1 Activation function

In this experiment, we aimed to investigate potential benefits of incorporating PReLU in place of normal ReLU activation layer within our final model. We wanted to examine to what extent the issue of vanishing gradients affects our detector and if the introduction of a learnable *leakage* parameter per each channel may aid with potential performance enhancement. For the purpose of this experiment we used the same architecture as described in Chapter 4 and the same values of hyper-parameters (Section 5.1) with the only difference being the type of activation layer which made the experiments comparable between each other since the networks were equivalent except the investigated component.

Figure 5.5 presents the precision-recall curves for the equivalent models using PReLU and ReLU activation units respectively. As we can observe, the network using PReLU unit recorded a higher precision value than its counterpart using ReLU at every recall threshold. This is also summarised by the average precision being almost 5% higher when using PReLU and being equal to 0.7719 compared with 0.7377 for ReLU. Such performance gain is in line with other studies investigating the influence of different activation units [66, 175] and indicates the benefit of introducing a new parameter to control the slope for the negative inputs per channel, alleviating the effect of vanishing gradients on the final performance.

Furthermore, the higher AP score for PReLU might imply its potential benefit of combating the overfitting. However, due to the implementation limitations (Section 5.1.1), we were unable to support this claim. Achieved results also prove that ReLU is very likely to suffer from vanishing gradients affecting the overall network performance.

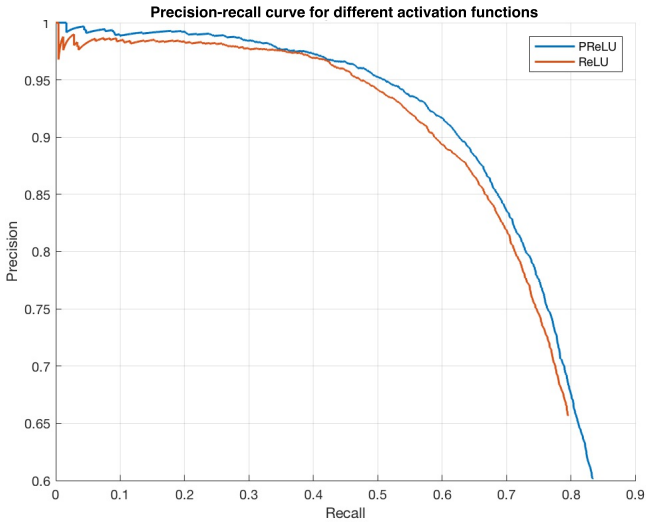


Figure 5.5: Precision-recall curves for the equivalent models using PReLU and ReLU activation units.

The time required to complete the training process did not increase noticeably when using PReLU and was less than 10% higher when compared with ReLU suggesting that the introduction of 1408 additional learnable parameters did not significantly affect the overall number of computations within the network. Such a small but noticeable increase is still acceptable due to the substantial increase in performance. Therefore, due to the presented benefits in network performance and no significant increase in training time, we chose to incorporate PReLU in our final Faster R-CNN architecture.

5.2.2 Base network depth

The purpose of this experiment was to determine the most optimal depth of our base network in order to extract appropriate features from the inputted tiles and allow good-quality proposal generation along with further refinement by the RPN and Fast R-CNN components respectively. The VGG-16 base network [155] used in the original Faster R-CNN architecture [134] consisted of 5 convolutional blocks of 2-3 convolutional layers followed by the pooling layer. However, due to the very small dimensions of the flower objects (i.e. approximately 14×14) being greatly downsampled after each pass through the pooling layer, we were not able to utilise as many blocks. Having more convolutional layers allows for feature extraction of greater complexity, however, these layers need to be occasionally separated by the pooling layer in order to downsample the incoming tensor thus reducing the risk of overfitting and allowing the training process to complete in a reasonable amount of time [115].

In this experiment, we investigated the performance of two equivalent architectures of different depths (i.e. the number of convolutional layers in the base network). The first network represented the same architecture with 2 convolutional blocks presented in Chapter 4, whereas the other one included an additional block of 2 convolutional

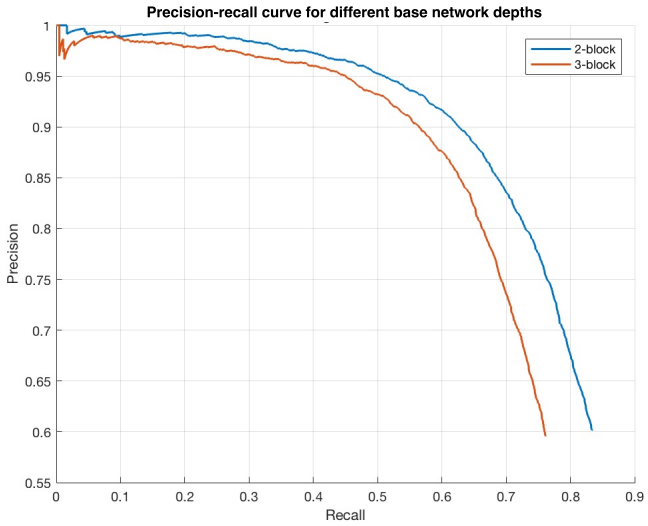


Figure 5.6: Precision-recall curves for the equivalent models using 2 and 3 convolutional blocks within their base network respectively.

layers with 256 feature maps each. We did not examine any deeper networks due to the downsampling significantly reducing the input dimensionality (i.e. factor of 2 per dimension) risking the information loss from the smaller flower objects after the complete pass through the base network.

As presented in Figure 5.6, the shallower architecture recorded higher values of precision for the same level of recall when compared with its 3-block counterpart. Hence, the average precision achieved by the base network with only 2 convolutional blocks was 0.7719 which was significantly higher than for the deeper equivalent achieving only 0.6993. Such a big difference in performance indicates that incorporating more pooling layers (3-blocks) resulted in a highly downsampled input and reduced network's ability to preserve appropriate features belonging to smaller objects. Our results followed the outcomes of similar studies [45] concerning small object detection, where the shallower blocks of the VGG-16 network tend to be more suitable for detecting smaller objects.

Moreover, our results might suggest that more complex feature might not be necessary to improve small flower detection, at the same time making the network more prone to overfitting due to the increased number of parameters when compared with its shallower counterpart. However, due to significantly different network capacities (i.e. increased number of parameters) and the same hyper-parameter set-up used to train both architectures, we cannot undoubtedly state if the addition of another block and more complex feature extraction disables our model to successfully detect smaller objects. That is because networks with different characteristics might archive their optimal performance in different training set-ups (i.e. learning rate, number of epochs). We were unable to investigate this aspect in more detail due to the time constraints. Nonetheless, the shallower base network achieved much better performance in the current training setting, therefore we used only 2 convolutional blocks in our final base network architecture.

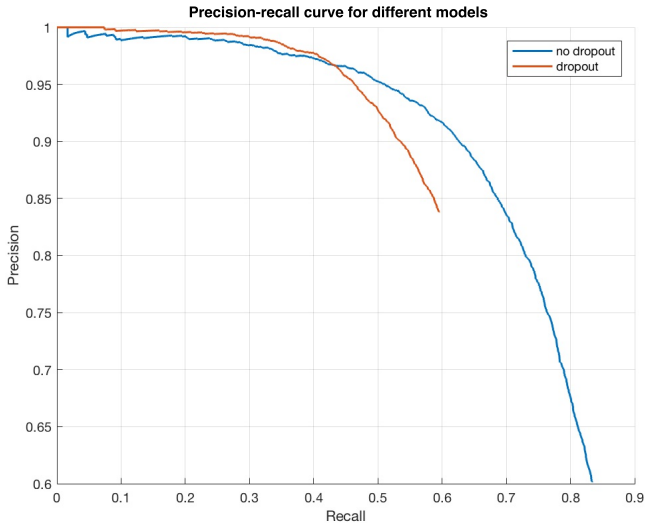


Figure 5.7: Precision-recall curves for the equivalent models with and without dropout layer used in their Fast R-CNN detector component.

5.2.3 Dropout

Our next experiment aimed to investigate the potential benefits of incorporating the dropout layer in combination with the fully-connected layers within the Fast R-CNN detector component of our model. As discussed in Section 4.1.5, dropout is a widely used method of regularisation which aids in combating network overfitting by preventing too complex co-adaptations on the training data. This is achieved by randomly dropping (with a fixed probability 1-p) the fraction of activations at each iteration. Hence, a network using dropout can be viewed as a combination of multiple architectures trained on different subsets of data [120].

For the purpose of this experiment, we used equivalent networks (Chapter 4) with the only difference between the two being the addition of dropout layer in combination with every fully-connected layer in the Fast R-CNN component. This is the most widely used sequence also incorporated in the original Faster R-CNN architecture [134]. Other studies also used dropout with max-pooling and convolutional layers, however, such approaches did not indicate any significant performance gains (Section 4.1.5).

We followed the original Faster R-CNN architecture and used the dropout layer only with the fully connected layers with the fixed probability p of retaining units equal to 0.7. This value was slightly higher than the one used in the original architecture due to the risk of the high amount of noise being introduced to the network inputs, decreasing its ability to detect smaller objects [120]. We also increased the learning rate by the factor of 10 for the architecture utilising dropout as recommended by [159] in order to alleviate the effect of cancelling gradients introduced by the additional noise when using dropout (Section 5.1.1).

As we can observe in Figure 5.7, the precision-recall curve for the architecture utilising dropout is slightly higher when compared with the equivalent architecture without it

until the recall threshold of approximately 0.4. Such result indicates that the addition of dropout helped to increase networks precision for the lower values of recall where the model detects a smaller number of objects, with the great majority of which being *real* flowers. However, the trend reverses for the higher values of recall (i.e. > 0.4) where the *no dropout* architecture achieves noticeably greater precision. This is also reflected in much lower AP score for the *dropout* network of 0.6039 which is significantly smaller than the result achieved by its alternative (i.e. 0.7719). This might suggest that the potential lower risk of overfitting was outweighed by the high amount noise introduced within the network, disabling it to perform correct detection of smaller objects.

Despite achieving noticeably lower AP, dropout indicates the potential of improving network performance as suggested by higher precision scores for lower recall values. According to its authors [159], adding dropout results in the network which might take approximately 2-3 time longer to train than the equivalent architecture without it due to the noisy parameter updates. That is because at each pass, we are training different random combination of units (i.e. dropping parameter p) in each layer meaning that the gradients generated during training are not the gradients of the final network that will be used at test time, hence requiring longer amount of epochs to successfully complete the training process [120]. However, despite its potential benefits to network performance, we did not include the dropout layer in our final architecture since that would result in lengthening the training process by a significant amount of time (2-3 fold) which would further strain our limited time and computational resources. Nonetheless, we consider dropout as a viable network enhancement and plan to investigate it further in the second part of the project (i.e. year 2).

5.2.4 Zero-centering

Our last experiment focused on investigating the potential benefits of zero-centering the input data before feeding it through the network using the `zerocenter` property of the MATLAB `imageInputLayer` implementation. As discussed in Section 4.1.1, zero-centering is a normalisation method which subtracts a mean image from the training set, centering the data towards the origin [85]. This operation results in faster network convergence due to the removal of possible bias in the gradients [46]. Moreover, zero-centering makes the network more robust to images of various contrast and illumination [140] which is a desirable property considering the varying quality of the drone imagery used in this project (Section 6.2). We used the same architecture as presented in Chapter 4 with the only difference being the lack of the `zerocenter` property for one of the models within the input layer of its base network (i.e. `imageInputLayer` in MATLAB).

As presented in Figure 5.8, the *zero-center* network achieved noticeably higher precision for all values of recall (including very small rates of < 0.1) when compared with *no zero-centering* model. This indicates a great benefit of zero-centering method on achieving better network convergence, thus increasing the overall performance of the model. Furthermore, the advantage of using this technique was also expressed by a noticeably higher value of AP. The *no zero-centering* network recorded AP of 0.7275 which is

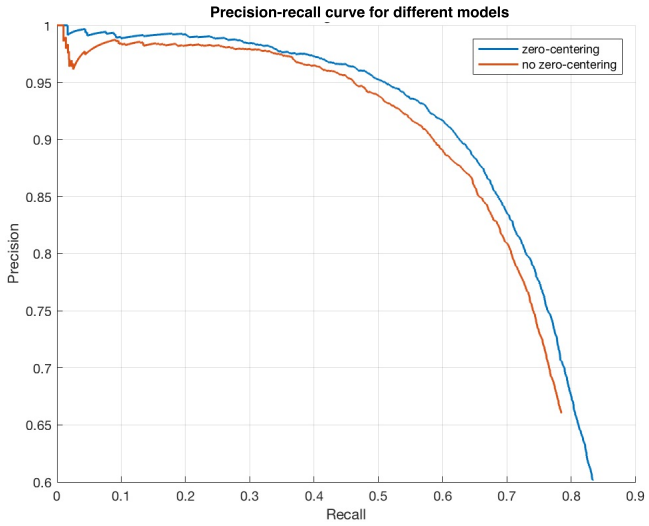


Figure 5.8: Precision-recall curves for the equivalent models with and without input zero-centering used within the input layer of their base network.

significantly less than the network utilising this method (i.e. 0.7719). Such results are in line with other studies which proved that zero-centering the inputs results in faster convergence when training neural networks with gradient descent due to alleviating the effect of randomly-directed gradients form the non-centred inputs [46].

We could have extended the idea of input normalisation further by dividing the input tiles by their standard deviation which would result in scaled inputs (standardisation). Input scaling speeds up the rate at which the weights connected to the preceding node are learned [92], further aiding network convergence. However, due to the lack of standardisation option for the MATLAB input layer implementation and time constraints, we did not investigate this concept further with a possibility of incorporating it in the second part our project (i.e. year 2). Nevertheless, our network clearly benefits from the zero-centering method, therefore we included this technique within our final model.

Chapter 6

Evaluation

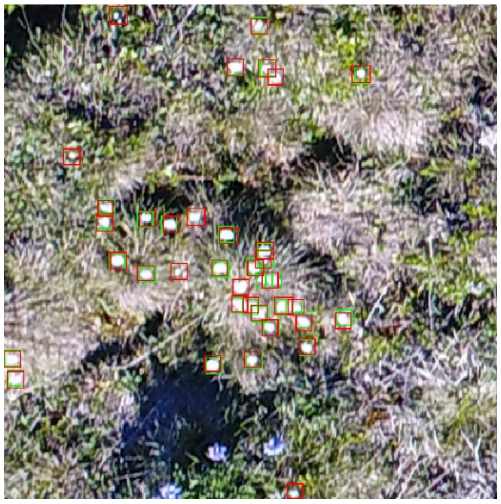
In this chapter, we will evaluate the final version of our network against human annotators. We will present, discuss and select the most appropriate measure for performance evaluation allowing both model-to-model and human-to-model comparison (Section 6.1). Moreover, due to the ambiguity of the data (i.e. partly covered or adjacent flowers) resulting in inconsistent annotators' judgement, we will analyse how the quality of the annotations used for the evaluation influenced the final assessment results (Section 6.1.5).

To assess both network's and humans' robustness to images collected in varying weather conditions, Section 6.2 examines how different image quality factors are likely to influence their detection ability. Section 6.3 presents the evaluation based on the *raw* counts collected on the ground allowing us to assess our network on the closest available measure of the *true* number of the flowers being present within each investigated area.

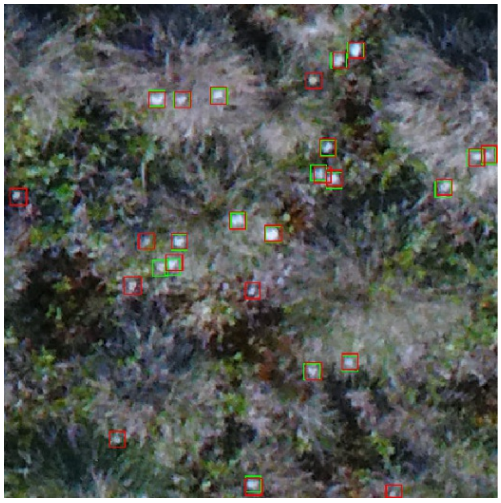
6.1 Standard measures

The decision on whether the object is considered as *real* flower is not trivial. This is determined by the amount of overlap (i.e. IoU) between the detection and ground-truth bounding boxes. We set the overlap threshold to 0.5 which follows the standard values in most of the object detection challenges such as PASCAL VOC [50]. Figure 6.1 indicates the following concepts for bounding box classification:

- True positives (TP): *true* flowers detected by the model (i.e. overlapping green and red bounding boxes with the IoU $\geqslant 0.5$).
- False negatives (FN): *true* flowers that have not been detected by the model (i.e. green bounding boxes overlapping with red boxes with the IoU < 0.5).
- False positives (FP): objects detected by the model, which are not included in the ground-truth (i.e. red bounding boxes overlapping with green boxes with the IoU < 0.5).



(a) Example model detections aligning with ground-truth.



(b) Example of detection where model's FP are very likely to be *true* flowers.

Figure 6.1: Example of the object detection on test set tiles (green and red bounding boxes donating ground-truth and model detections respectively).

It is worth noting that the definition of *real* flower is based purely on the *ground-truth* bounding boxes provided by the human annotators used for training (i.e. training set) and evaluation of the model (i.e. test set) where the flowers marked by two different people were combined by incorporating both bounding boxes if their IoU was less than 0.5. These annotations might be influenced by numerous of factors discussed in Chapter 3 resulting in the annotators' inability to spot every single flower object within each tile leading to the misinterpretation of some of the model's detections as *false* flowers (FPs). The quality of the provided annotations and its influence on the final results are investigated in Section 6.1.5. Nonetheless, in the first part of this section, we will follow the definition of *true* flower based on the human detections to find the most appropriate evaluation metric for the network and compare it to human performance.

6.1.1 Precision and recall

In the field of information retrieval as well as object detection, precision and recall are widely used metrics of system performance [72]. Precision is defined as a ratio of correctly predicted positive observations (TP) to the total predicted positive observations ($TP + FP$) (Equation 6.1). High precision relates to the low false positive (FP) rate. In case of our task, precision measures the proportion of the *true* flowers (TP) out of all flower objects detected by the model ($TP + FP$).

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

Recall, also known as sensitivity in case of binary classification, is a ratio of correctly predicted positive observations (TP) to the all observations in the positive class ($TP +$

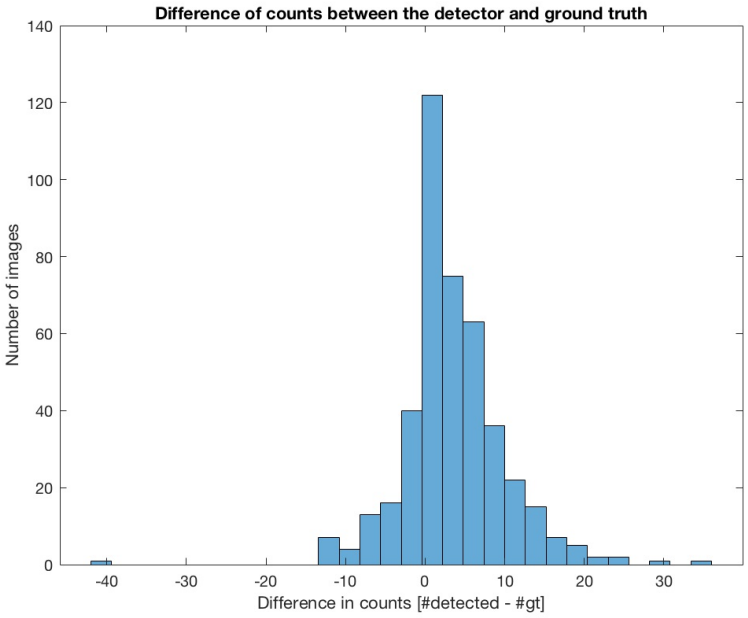


Figure 6.2: Histogram of count difference between model detections and the ground-truth (i.e. human detections).

FN) (Equation 6.2). For our task, recall answers the question of what proportion of all *true* flower objects ($TP + FN$) has been detected by our model (TP).

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

Having a model with a low precision and a high recall would not meet the expectations the potential users (i.e. researchers) since the network would return many detections, most of which not being the *true* flower objects. On the other hand, a model achieving high precision but a very low recall would return very few detections, most of which being real flowers. Such models are very likely to generate incomplete statistics (i.e. misestimated number of flowers), greatly affecting the studies of the influence of global warming on the *E. vaginatum* population. Therefore, our goal is to deliver a system scoring highly on both metrics (i.e. precision and recall), delivering numerous of objects most of which being *true* flowers.

Test set

Both precision and recall had been calculated based on all the metrics (i.e. TP, FP, FN) being summed over all tiles within the test set. When evaluated on the test set (i.e. 432 tiles), our network achieved much higher recall (0.8340) than precision (0.6014) which indicates that it was very likely to output many detections, the majority of which not matching the ground-truth annotations. The same can be observed from the histogram depicting the difference of flower counts between network's detections and the ground-truth (Figure 6.2). For almost 74% of the images, the network detected more objects

		Result							
		Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Ground-truth	Model
Target	Annotator A	1.0000	0.9387	0.7897	0.9305	0.7390	0.8341	0.6718	0.6169
	Annotator B	0.8057	1.0000	0.7336	0.8663	0.6441	0.7788	0.6043	0.5577
	Annotator C	0.6842	0.7406	1.0000	0.7968	0.5525	0.7143	0.5399	0.4704
	Annotator D	0.7045	0.7642	0.6963	1.0000	0.6034	0.7696	0.5399	0.4901
	Annotator E	0.8826	0.8962	0.7617	0.9519	1.0000	0.8341	0.6718	0.6225
	Annotator F	0.7328	0.7972	0.7243	0.8930	0.6102	1.0000	0.5859	0.5042
	Ground-truth	0.8866	0.9292	0.8224	0.9412	0.7424	0.8802	1.0000	0.7155
	Model	0.8866	0.9340	0.7804	0.9305	0.7492	0.8249	0.7791	1.0000

(a) Precision scores

		Result							
		Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Ground-truth	Model
Target	Annotator A	1.0000	0.8057	0.6842	0.7045	0.8826	0.7328	0.8866	0.8866
	Annotator B	0.9387	1.0000	0.7406	0.7642	0.8962	0.7972	0.9292	0.9340
	Annotator C	0.7897	0.7336	1.0000	0.6963	0.7617	0.7243	0.8224	0.7804
	Annotator D	0.9305	0.8663	0.7968	1.0000	0.9519	0.8930	0.9412	0.9305
	Annotator E	0.7390	0.6441	0.5525	0.6034	1.0000	0.6136	0.7424	0.7492
	Annotator F	0.8341	0.7788	0.7143	0.7696	0.8295	1.0000	0.8802	0.8249
	Ground-truth	0.6718	0.6043	0.5399	0.5399	0.6718	0.5859	1.0000	0.7791
	Model	0.6169	0.5577	0.4704	0.4901	0.6225	0.5042	0.7155	1.0000

(b) Recall scores

Figure 6.3: Precision and recall scores for each individual annotator and the model evaluated against each other on the set of 15 images (target/ground-truth and result/generated annotations are represented in rows and columns respectively).

than indicated by the ground-truth (i.e. human detections). This suggests a high number of the false positives (FP) generated by the model.

However, it was not certain that the human annotations included all possible flowers within each tile due to the inconsistency of the annotators and other factors described in Chapter 3. Hence, we cannot be entirely certain if our network detected the instances of *E. vaginatum* flowers that the human annotators had missed or these were just irrelevant objects (i.e. light reflections, grass). This is also indicated by Figure 6.1(b), where part of the model's FP detections (i.e. red bounding boxes) resemble flowers. Nonetheless, the scores for both measures show model's potential to deliver satisfactory results and the need for further investigation of its false positives which in fact might be the *real* flowers.

Validation subset

Another aspect of our task was to provide users with a system which mimics human performance or even outperforms people by generating better quality detections. We wanted to investigate the consistency among human annotators and its influence on the evaluation results. Therefore, the next step of our evaluation process was to compare our network to human annotators directly on the exact same set of annotated tiles.

For this part of the evaluation, we randomly selected 15 images from the test set and asked 6 individuals to perform the annotations based on their own judgement. We decided to create new *evaluation subset* of tiles due to the ground-truth annotations being generated from a combination of tiles annotated by many different people with no common subset being covered by all of the participants. This suggests that the ground-truth might not be suitable for direct evaluation against each individual person and assessment of the influence of their annotations on the final results. We are also aware that due to its very limited size, the subset might not be fully representative and

sensitive to very small differences in the annotations among people. However, we were not able to provide a bigger set for the evaluation purposes due to the limited time and resources. Expanding this set is considered as one of the improvements for the second part of the project (Chapter 8).

We used new annotations in combination with the human and model detections to evaluate them against each other, meaning that we used each of them as target *ground-truth* (i.e. row) to evaluate the rest (i.e. column). The results for precision and recall of those comparisons are presented in Figure 6.3(a) and Figure 6.3(b) respectively and might suggest the possible but not mutually exclusive explanations:

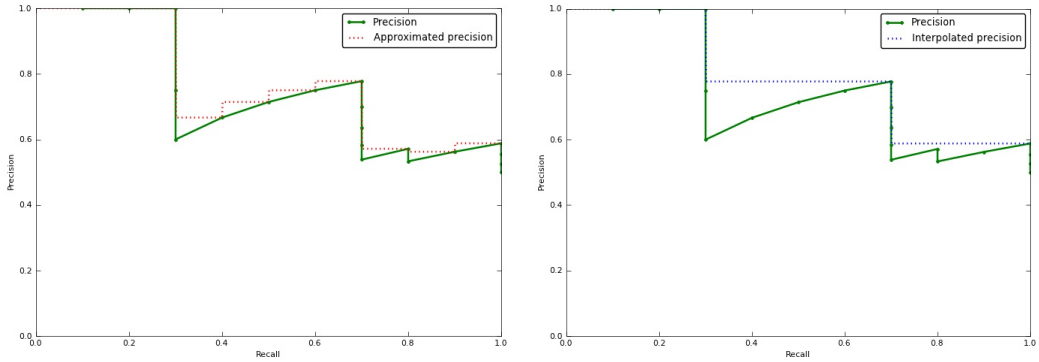
- **Inconsistency among annotators**: the values for both measures vary greatly among annotators showing how ambiguous and non-trivial both the annotation and, as a result, the evaluation tasks actually are. The inconsistency of the annotators' decisions suggests that the ground-truth might indeed not include all possible flowers within each of the tiles, altering the number of false positives generated by the model.

- **Poor recall of the annotators**: humans tend to achieve lower recall than the network, especially while evaluated on the ground-truth predictions as the target (i.e. second row from the bottom). Such low value might be the result of humans' self-restraint in annotating the objects which do not clearly resemble flowers (i.e. humans' low TP) or the model detecting many objects which in fact are not flowers (i.e. model's high $TP + FN$). The results suggest that flower detection is potentially non-trivial visual research task, and people might not have noticed all (or even most) of the flowers presented within each tile.

- **Poor precision of the model**: the model tends to achieve lower precision compared to most of the human annotators, which has previously been noticed in case of the test set. This might be caused either by the inability of the network to extract the appropriate features and misinterpreting other similar looking objects as flowers (i.e. network's high FP) or human inability to notice all the flowers within a tile (i.e. humans' low $TP + FP$).

Although, precision and recall are extremely useful for interpreting networks behaviour, having a single measure for network evaluation would be much more convenient and make the performance easily interpretable. Therefore, in the next sections, we will strive to find the most optimal measure that enables us to combine both precision and recall metrics.

Previous studies indicate the quality of the annotations having a significant influence on models' performance and their evaluation [181]. This was also confirmed by our comparisons on each individual annotator showing that the ground-truth used as a test set might not contain all flowers present within each tile, potentially limiting network's ability to generate good detections (i.e. high FP). This needs further analysis, which we had conducted and presented in Section 6.1.5.



(a) Approximation method for AP.

(b) Interpolation method for AP.

Figure 6.4: Comparison between approximation and interpolation of the average precision [112] (AP).

6.1.2 Average precision

One of the most popular metrics of model performance is average precision (AP) [1, 45]. AP is calculated by averaging the precision values of the precision-recall curve across all values of recall between 0 and 1 (Equation 6.3), which is equivalent to the area under the curve [184].

$$AP = \int_0^1 p(r)dr \tag{6.3}$$

We have previously used precision-recall curves to compare different parameter and architecture set-ups of the network (Section 5.2). In this section, we are going to explain this method in more detail. The graph of the precision-recall curve shows the trade-off between precision and recall for different thresholds. A big area under the curve indicates both high recall and high precision (Figure 6.5). In practice, the integral representing the average precision in Equation 6.3 is approximated by a sum over the precision scores at every possible recall threshold multiplied by the change in recall (Equation 6.4).

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k) \tag{6.4}$$

However, some of the studies [102] present alternative approach of interpolated average precision to approximate the $p(r)$ function and reduce the impact of *wiggles* in the precision-recall curve which might be caused by the small variants in the ranked examples [102]. The formula in Equation 6.5 represents 11-point interpolated average precision where samples from 11 equidistant points between 0 and 1 ({0, 0.1, 0.2, ..., 0.9, 1.0}) are used for the calculations. This method has also been used in Pascal VOC competition as a standard measure to evaluate the results of each model [50].

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{interp}(r) \tag{6.5}$$

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geqslant r} p(\tilde{r}) \tag{6.6}$$

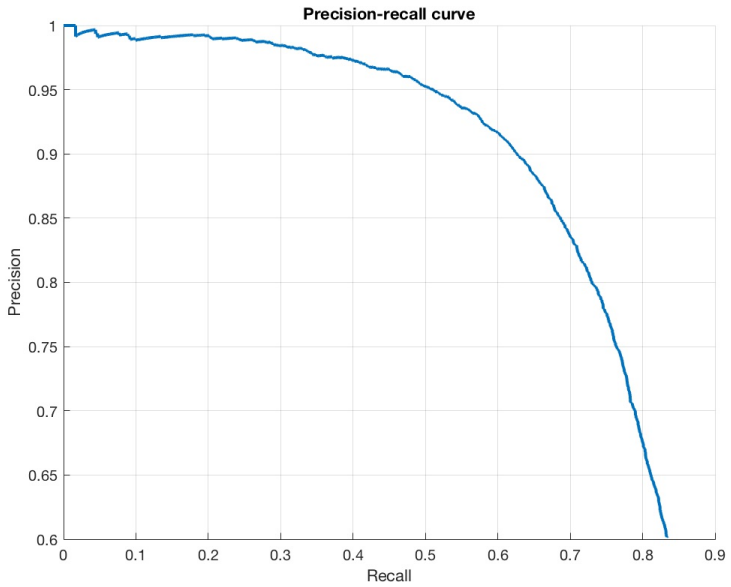


Figure 6.5: Precision-recall curve for the test set.

This method of linear interpolation of points on the precision-recall curve tends to yield overly-optimistic measures of model performance [53] due to the cases of possible non-linear change in precision as the value of recall varies [38]. Differences between interpolation and approximation methods are presented in Figure 6.4. Therefore, we used the formula presented in Equation 6.4 which also follows MATLAB implementation of the `evaluateDetectionPrecision` method for object detection evaluation [105].

Test set

We have provided the precision-recall curve for the test set in Figure 6.5. When applied on the test set, our network achieved the average precision of 0.7719 which indicates a satisfactory performance in the case of small object detection [45] with some networks scoring up to 0.66 mAP in different tasks involving small objects (i.e. mean-AP in multi-class tasks). Despite the significant differences between those tasks, both of them involve very small objects making their results relatively comparable.

While being a good measure for comparison between models, AP is not suitable for comparison between human annotators. This is due to the lack of ranked list according to the increasing confidence or any measure to estimate people's annotation confidence which would allow us to rank the annotations according to the scores, plot the precision-recall curve and estimate the enclosed area under the curve (i.e. AP) [4]. It is worth noting that, along with the bounding boxes, our network also returns confidence scores of each box containing a flower making it possible to plot precision-recall curve.

Validation subset

Basing the AP on just one precision-recall pair for each person would mean a significant simplification and a high risk of underestimation of the true value of AP [102]. Introduction of confidence scores while annotating each individual flower would significantly increase the amount of time required to complete the annotation process, influencing

the quality of the annotations. Moreover, the confidence scores might be the case of a subjective judgement of each individual annotator, which could also influence the final results. Therefore, we decided not to ask people to rank their annotations based on their confidence.

Despite the lack of the confidence scores, we have provided the reader with the AP scores on the set of 15 images for each individual annotator and the model evaluated against each other just like in the previous section (Section 6.1.1). All annotations had been given the confidence score of 1 indicating that each person was certain about their decisions which is a substantial simplification in this case. The table of the scores can be found in the Appendix D.1 and indicates better performance of the network compared to any of the annotators. However, as mentioned previously, the AP scores for humans are very likely to be underestimated due to the lack of their confidence scores.

The above discussion demonstrates that the average precision provides a concise and reliable metric to summarise the precision-recall curve for the model, making it a suitable metric for model-to-model performance comparison. However, AP is not appropriate for human-to-model evaluation due to the lack of human confidence scores of each annotation increasing the risk of underestimation. Therefore, in next sections, we aim to find a single, universal evaluation metric allowing reliable human-to-model comparison.

6.1.3 Miss-rate

Another commonly used way of presenting model's performance is a plot of miss-rate against the number of false-positives per image (i.e. $FPPI$) by varying the threshold on the detection confidence [10] with both values plotted on log axes (Figure 6.6). Such method of presenting the results is especially popular in the task of pedestrian detection [149, 180] due to the upper limit on the acceptable $FPPI$ rate independent of pedestrian density [181].

The plot can be summarised by the aggregate performance score of log-average miss rate (i.e. $LAMR$) computed by averaging miss rate at nine equidistant $FPPI$ data points in log-space in the range of 10^{-2} to 10^{0} [75] (Equation 6.7; the minus sign yields positive values of the measure).

$$LAMR = \frac{-1}{n} \sum_{i \in [10^{-2}..10^{0}]} \log(mr_i) \tag{6.7}$$

$$mr_i = \frac{FN_i}{FN_i + TP_i} \tag{6.8}$$

Some alternatives include considering a broader range of log-space, namely 10^{-4} to 10^{0}, reflecting improvements in localisation errors and yielding complete evaluation metric [181]. However, we have not recorded any significant differences in the values when applied the broader range. Therefore, we have followed the standard MATLAB implementation of the `evaluateDetectionMissRate` method [104].

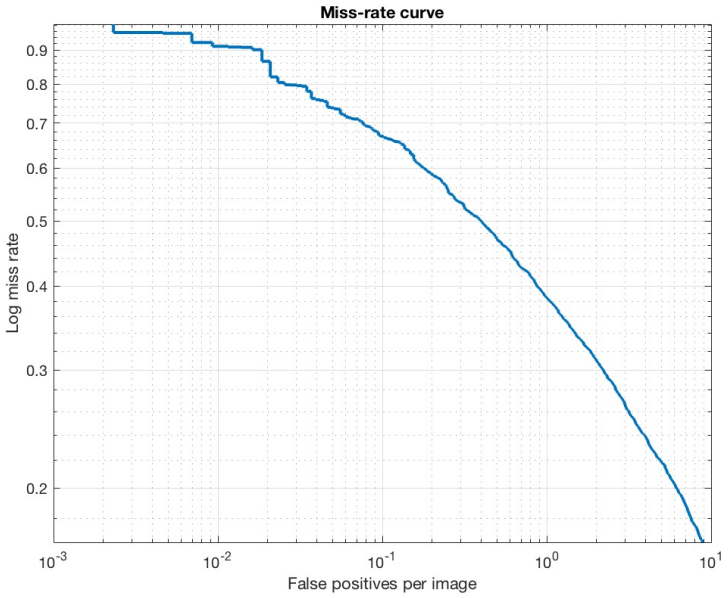


Figure 6.6: Miss-rate curve for the test set.

Test set

The miss-rate plot is shown in Figure 6.6 with our network achieving a log-average miss rate of 0.6382 when evaluated on the test set. This value indicates relatively poor performance when compared to other Faster R-CNN based detectors [180] used for pedestrian detection tasks, capable of achieving *LAMR* of 0.3020 and 0.0690 on ETH [49] and INRIA [35] datasets respectively. However, it is worth noting that these sets include much bigger objects (i.e. pixel area) making the results on those tasks highly incomparable. Nonetheless, the value of 0.6382 is relatively high (the maximum of 1) and might be influenced by the poor quality of the annotations used in the ground-truth which we are going to analyse in Section 6.1.5.

Validation subset

We have provided the tables of the *LAMR* scores for each annotator and the network evaluated against each other in Appendix D.2. The model tends to achieve lower *LAMR* scores than the humans which are very likely to be underestimated as in the case of AP due to the lack of confidence scores. Like *AP*, *LAMR* represents a comprehensive measure for representing the entire curve by a single value. However, similarly to precision-recall curve, the detections also need to be ranked according to their confidence scores making *LAMR* unsuitable measure for human-to-model evaluation [41].

Moreover, *LAMR* does not take into account the value of precision, recording only the miss rate (i.e. $1 - recall$) which makes this measure more suitable for evaluating pedestrian detection, where avoiding the crash with a human is prioritised over delivering precise detections [96]. This is different from the task of flower counting where both precision and recall are vital to yield reliable counts. Hence, log-average miss rate is not a suitable measure for evaluation both model-to-model and human-to-model performance.

6.1.4 F-score

The last considered evaluation metric was a weighted harmonic mean of precision and recall known as *F-score* [130]. The formula for the *F-score* is presented in Equation 6.9 and introduces a new parameter β. In this case, *F-score* measures the effectiveness of the retrieval depending on the importance of the recall compared to the precision, controlled by the parameter β (i.e. recall being β times more important than precision).

$$F_\beta = \frac{(1+\beta^2)\cdot(precision\cdot recall)}{(\beta^2\cdot precision + recall)} \tag{6.9}$$

It is worth noting that the Equation 6.9 uses harmonic rather than the arithmetic mean. This gives the *F-score* the advantage of minimising the impact of large outliers over the small ones, privileging balanced systems [143]. This being said, the *F-score* tends to be a good measure to combine both precision and recall. If one of them excels, *F-score* will reflect it [102]. Since we value both precision and recall equally, we decided to set the value of β to 1 which yielded, so-called, F_1 score. The formula for this measure has been presented in Equation 6.10.

$$F_1 = \frac{2\cdot precision\cdot recall}{(precision + recall)} \tag{6.10}$$

A possible disadvantage of the *F-score* measure is that it does not take true negatives into account (i.e. TN) [54] which might be appropriate for assessing the performance of a binary classifier, hence, it is often replaced by other measures such as Cohen's kappa [80] or Matthews correlation coefficient [126]. However, this is not the concern in case of the object detection due to TNs not being suitable for assessing the detector's performance since it only defines all the pixels that have not been enclosed in any of the boxes (model detection or ground-truth).

Test set

Our network achieved F_1 score of 0.6989 on the test set which is still significantly less than the maximum value of 1. This score is very likely to be influenced by the poorer value of precision caused by the incompleteness of annotations used as the ground-truth (i.e. some of the network's FPs might be the actual flowers; TP). However, due to the big amount of images in the test set (i.e. 432) we were unable to verify each tile and update the values of precision and recall accordingly. Nonetheless, we performed this analysis in Section 6.1.5 on the subset of 15 images was previously used for the comparison with individual annotators.

Validation subset

Since calculating the overall precision and recall does not involve taking any confidence scores into account, F_1 score is a suitable measure for human-to-model comparison. Moreover, the scores on the subset of 15 images are presented in Figure 6.7 and indicate poorer performance of the network against human annotators. However, as mentioned previously, this is caused by the network's lower precision scores and requires further analysis of both humans and network annotations in order to make sure whether the

		Result							
		Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Ground-truth	Model
Target	Annotator A	1.0000	0.8671	0.7332	0.8018	0.8044	0.7802	0.7644	0.7276
	Annotator B	0.8671	1.0000	0.7371	0.8120	0.7495	0.7879	0.7323	0.6984
	Annotator C	0.7332	0.7371	1.0000	0.7431	0.6405	0.7193	0.6519	0.5870
	Annotator D	0.8018	0.8120	0.7431	1.0000	0.7386	0.8267	0.6862	0.6421
	Annotator E	0.8044	0.7495	0.6405	0.7386	1.0000	0.7070	0.7053	0.6800
	Annotator F	0.7802	0.7879	0.7193	0.8267	0.7031	1.0000	0.7035	0.6259
	Ground-truth	0.7644	0.7323	0.6519	0.6862	0.7053	0.7035	1.0000	0.7460
	Model	0.7276	0.6984	0.5870	0.6421	0.6800	0.6259	0.7460	1.0000

Figure 6.7: F_1 scores achieved on the subset of 15 images. The results indicate poorer performance of the network against human annotators.

model detected a high number of objects that do not represent flowers (i.e. high number of TN+FP). The results of this analysis will be presented in the next section (Section 6.1.5).

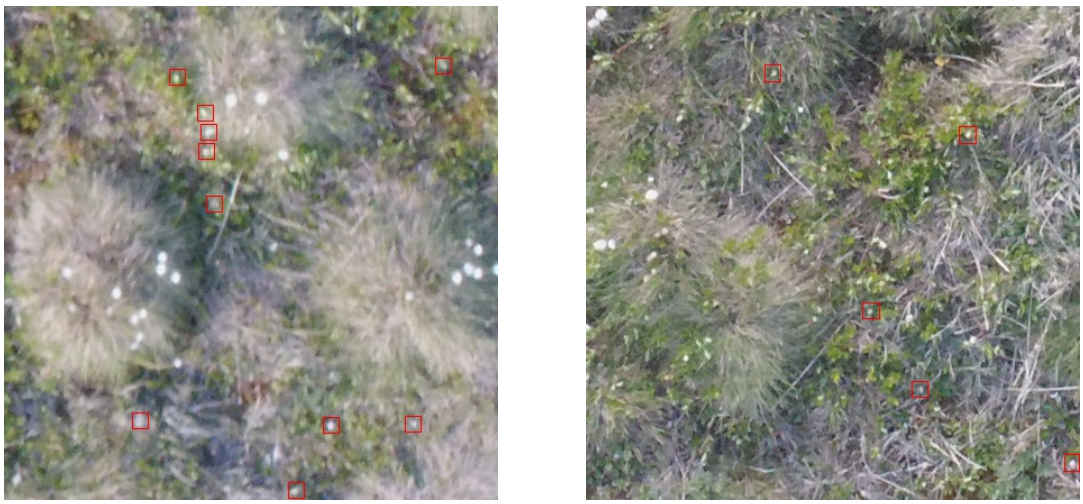
Despite its values being influenced by the quality of the annotations, *F-score* combines both precision and recall in a controllable manner (i.e. β parameter) and, unlike average precision and log-average miss rate, does not require confidence scores for its true value estimation. This makes *F-score* suitable for both model-to-model and human-to-model evaluation and that is why we decided to base our final evaluation on this metric.

6.1.5 Annotation analysis

After analysing network's performance on the previously described set of 15 images used for human-to-model comparison, we noticed network's low precision scores when compared with human annotators. This was likely to be caused by the poor annotations quality provided as a ground-truth as explained in Section 6.1.1.

While analysing 15 images used as the ground-truth, we noticed that all annotated objects indeed resembled *real* flowers. That being said, we considered the TPs for both model and humans as not worth analysing due to the fact that all detected objects with sufficient overlap (i.e. $IoU \geqslant 0.5$) had been correctly classified as detector's TP. This is because the ground-truth contains only *real* flowers, both network's and annotators' TPs had been correctly classified and its value might only change due to some of the FPs being missed by the ground-truth.

However, our ground-truth analysis indicated that some of the *real* flowers were missing and had not been annotated as shown in Figure 6.8. That is why we decided to investigate detections classified as FPs to assess if the network and the annotators successfully detected those objects that had been missed by the ground-truth. For this purpose, we generated sets of tiles with annotated FPs (i.e. red bounding boxes) for each of the annotators and the model. These tiles were presented to the annotators who analysed if each of the boxes indeed contained objects which should have been classified as flowers (i.e. TP). We used the same annotation tool as previously (Section 3.2) with a single click within a bounding box indicating that it had been incorrectly classified as FP and indeed contained a *real* flower. The annotators did not analyse their own annotations



(a) (b)

Figure 6.8: Tiles representing FPs generated by the network (i.e. red bounding boxes). We can see that the network is detects even very small and blurry objects resembling flowers in their fading stage suggesting the ability of the model to detect the flowers in any stage of their growth.

and they were not aware whose annotations they were investigating in order to avoid any bias.

The analysed detections allowed us to recalculate the values of FPs and TPs (i.e. objects previously misclassified as FPs which are *real* flowers became TPs) which we used to update precision, recall and F_1 scores for both network and the annotators. We have presented the updated results in Figure 6.9 which indicate an significant changes in both precision and recall values for both model and human detections. It is worth noting that 68% of the model's FP detections contained *real* flowers and were reclassified as TPs. Such a high percentage of misclassification signifies a great incompleteness of the ground-truth annotations and, as a result, underestimation of the previously recorded values of precision (Figure 6.3(a)), recall (Figure 6.3(b)) and F_1 (Figure 6.7).

Figure 6.9(a) presents values of precision for both the annotators and the network before (i.e. top row) and after (i.e. bottom row) the parameter update. The greatest increase of 27% (i.e. change from 0.7155 to 0.9099), had been recorded for the network's precision value. The second highest increase of 17% has been achieved by the annotator E. Such significant changes in precision scores indicate many *real* flower objects not being included in the original ground-truth annotations causing many detections to be classified as FPs. This also suggests that the previously recorded network's test set precision of 0.6549 (Section 6.1.1) was very likely to be underestimated. Furthermore, despite the value of the updated network's precision not being the highest when compared with human annotators, the precision of 0.9099 indeed suggests a good performance of our model with most of its detections being *real* flowers.

	Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Model
Before	0.8866	0.9292	0.8224	0.9412	0.7424	0.8802	0.7155
After	0.9514	0.9764	0.9065	0.9786	0.8678	0.9539	0.9099

(a) Updated precision

	Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Model
Before	0.6718	0.6043	0.5399	0.5399	0.6718	0.5859	0.7791
After	0.6871	0.6161	0.5640	0.5495	0.7052	0.6053	0.8177

(b) Updated recall

	Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Model
Before	0.7644	0.7323	0.6519	0.6862	0.7053	0.7035	0.7460
After	0.7980	0.7555	0.6953	0.7038	0.7781	0.7406	0.8613

(c) Updated F_1 score

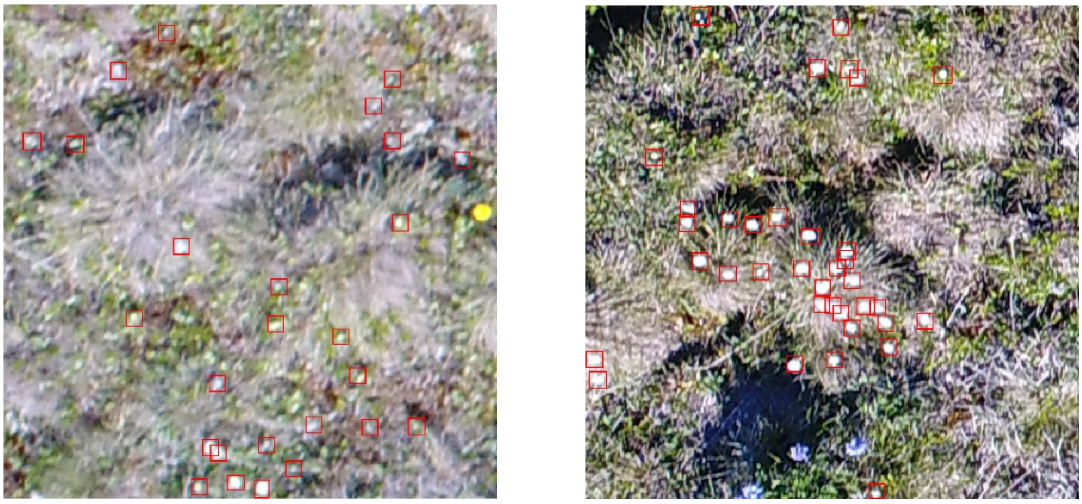
Figure 6.9: Updated values of precision, recall and F_1 score respectively. Our network recorded a significant increase of precision resulting in the highest F_1 score and indicating its very good performance.

Figure 6.9(b) shows the updated values of recall with the highest increase of 5% being recorded for our network. Smaller changes in the scores for recall compared to precision are caused by the values of FNs being unaffected by our analysis since all of the ground-truth annotations contained *real* flowers (i.e. denominator only affected by the updated number of TPs). Moreover, lower recall values for human annotators suggest that some people were not able to detect as many flowers as the network.

Updated F_1 scores presented in Figure 6.9(c) reflect previously discussed changes in both precision and recall values. Our network recorded the highest score of 0.8613, with annotator A achieving second best result of just 0.7980. Such a high value of F_1 demonstrates our network's excellent performance, disproving previous claims of it returning very many detections most of which being random objects other than *E. vaginatum* flowers.

It is worth noting that we have not analysed the whole test set (i.e. 432 tiles) initially used for the network evaluation but rather a small randomly selected subset of it containing just 15 tiles used for human-to-model comparison. This was caused by the limited resources (i.e. annotators availability) as well as time constraints. Nonetheless, the significant changes of both precision and recall suggest that both of these measures were likely to be underestimated if the whole test set would be considered. Moreover, mentioned changes also show us how the quality of the annotations is likely to affect the overall evaluation of any model which had previously been indicated by other studies [181].

The updated scores still vary greatly among the annotators indicating people's lack of consistency in flower annotation which is not the case for the network detections since the detector's decisions are based purely on the features extracted from an image and learned parameters. This consistency in decision combined with the higher value of F_1 score suggest that our network is indeed able to perform as good as human annotators or even outperform them in some cases.



(a) Tile with the relative sharpness of 2.459 (b) Tile with the relative sharpness of 9.115

Figure 6.10: Despite extreme variety of tile sharpness, our network is still able to perform accurate detection.

6.2 Other factors

Apart from yielding human-like performance, any model ought to present the highest possible robustness to various factors affecting image quality. Our dataset introduces a great challenge to any detector since its tiles represent samples of extremely varying quality including very blurred (Figure 6.10(a)) or dark (Figure 6.13(a)) tiles. Therefore, in this section, we will investigate our network's robustness to two main factors which are the most likely to affect the quality of detection such as relative image sharpness and luminance [166, 144].

6.2.1 Sharpness

Sharpness determines the amount of detail an image can reproduce which makes it one the most important image quality factors [9]. Image sharpness can be affected by the aperture used (i.e. focal length, distance from image centre) as well as other factors including weather conditions such as wind (i.e. camera shake, object movement) or rain (i.e. lens cleanliness) [64].

Due to the extremely diverse and relatively unpredictable weather conditions within the studied area (i.e. Arctic), we expected the collected images to present a wide degree of detail. Indeed, as shown in Figure 6.10, the tiles used as the training and test sets significantly differ in quality with Figure 6.10(a) representing a very blurred tile and Figure 6.10(b) being able to preserve much more detail due to its higher sharpness.

Lost sharpness can be restored using a variety of methods such as unsharp masking [168] or deconvolution [164]. However, these techniques are limited and not always universal in respect to images of a wide range of quality making artifacts such as

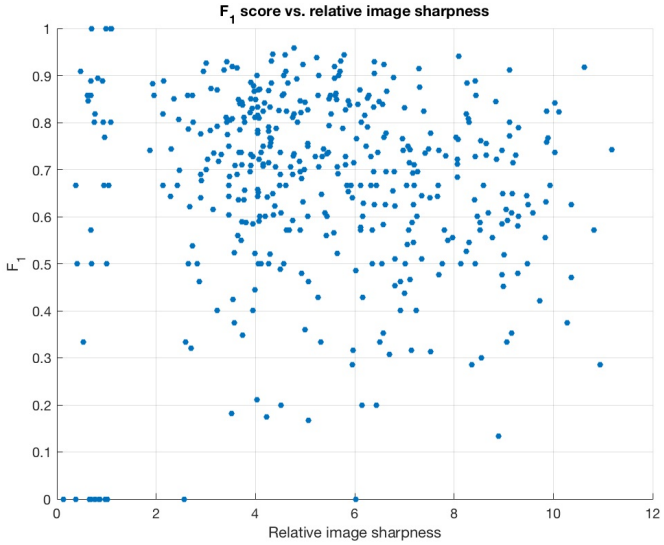


Figure 6.11: Scatter plot of F_1 scores and relative sharpness for test set indicates a high degree of variance in image sharpness and network's robustness to this factor.

amplified noise become even more apparent [164]. That is why we did not decide to use any image augmentation steps to sharpen the tiles due to the risk of affecting important flower features.

Varying image sharpness is extremely likely to affect the quality of detections made by both human and model. We would expect very blurred images to be completely unsuitable for reliable flower detection, however, this needs to be closely analysed based on the quantitive measures. Therefore, our next step involves defining a metric for the relative image sharpness and investigate our network's performance on the test set based on this measure.

$$f_{grad,n} = \frac{1}{MN} \sum_{x=1}^{M-n} \sum_{y=1}^{N-n} \sqrt{[i(x+n,y) - i(x,y)]^2 + [i(x,y+n) - i(x,y)]^2} \tag{6.11}$$

Recent studies suggest many different metrics to estimate image sharpness including power spectrum [27], variance [48] and histogram-based [103] methods. However, since our analysis involves only assessing relative differences in sharpness between the tiles (i.e. how much sharper a tile is compared to the other), we do not see the necessity to follow the most accurate technique which is also likely to increase the number of computations and complexity of our assessment. Hence, we decided to apply a very simple measure based on image gradients [37] which has been presented by Equation 6.11 where $i(x,y)$ represents an image and $n = 1$ since we are considering per-pixel change. This technique had been widely used in the field of electron microscopy[71] as well as other studies [37] due to the ease of implementation and satisfactory accuracy.

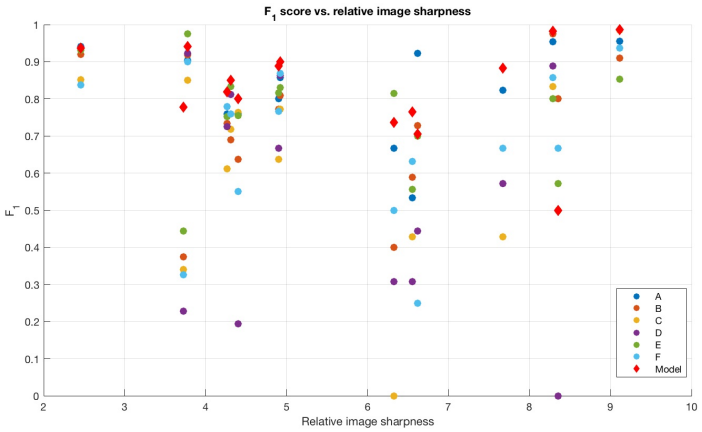


Figure 6.12: Scatter plot of F_1 scores and relative image sharpness for the set of 15 images. Extreme variety in image sharpness does not significantly affect neither human nor network's performance.

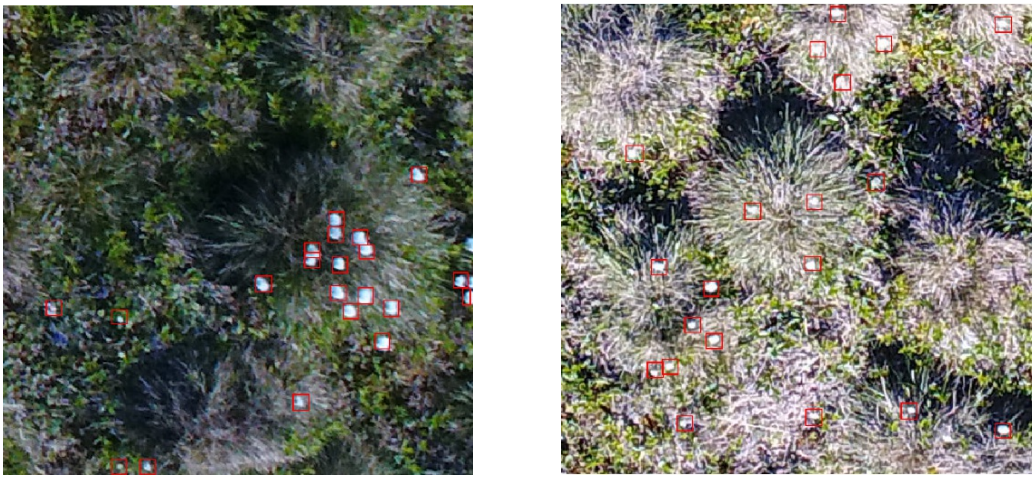
Test set

The results presented in Figure 6.11 confirm a wide range of tiles' sharpness within the test set which is very likely to affect the detector's performance. However, except a small number of outliers caused by a very poor tile quality and random objects such as markers or black plastic foils used to protect the equipment, the scatterplot does not indicate that the network performs poorer on more blurred images. Our model did not achieve significantly higher F_1 scores when applied on sharper images either, which might suggest that it is robust to this image factor and it is able to extract essential flower features to perform the detection despite poorer image quality.

Validation subset

It is worth noting that the discussed F_1 scores are highly likely to be underestimated due to the poor quality of the annotations used as the ground-truth within the test set which we had proven in Section 6.1.5. That is why we decided to conduct a similar image quality analysis on the set of 15 images with updated F_1 scores. Moreover, this analysis allowed us to assess how the relative image sharpness may affect human annotators' judgements on how many flowers they can notice within each tile. The results have been presented in Figure 6.12 and represent each of the 15 tiles (i.e. points forming a vertical line), their sharpness and F_1 scores for each individual annotator as well as the model. Furthermore, despite its limited size, the investigated set consists of tiles with a wide degree of sharpness.

Similarly, as for the test set, there is no clear pattern observed which might suggest that the quality of the provided tiles was not a limiting factor for neither human nor the network performance. None of the tiles, however, were extremely blurred or over-sharpen which would likely affect the detection results. Nonetheless, the network tends to be unaffected by the quality of the provided tiles, in some cases outperforming human annotators.



(a) Tile with relative luminance of 0.3192 (b) Tile with relative luminance of 0.6060

Figure 6.13: An extremely good performance on both dim and bright tiles suggest network's robustness to image relative luminance.

6.2.2 Luminance

The next very crucial image quality factor is luminance which defines the luminous intensity detected by the human eye while looking at a given area from a given angle [55]. Luminance describes the amount of light which is emitted and reflected from a particular area which makes it a good indicator of image brightness [90]. However, it is important to distinguish between image luminance and brightness where the latter is a subjective attribute of light which cannot be measured objectively. Nonetheless, luminance is a widely used measure to indicate image brightness [83] and we are going to use those terms interchangeably for the sake of simplicity.

Due to the extremely varied weather conditions and times of the day of the data collection, our dataset consisted of tiles with a wide degree of brightness as shown in Figure 6.13. Figure 6.13(a) depicts a dimmed tile collected during the late afternoon, whereas Figure 6.13(a) indicates much brighter tile photographed at noon. There are many techniques for improving image brightness such as histogram equalisation [17, 101], however, similarly as for sharpness, we did not decide to apply any image enhancement techniques prior to network training [111].

In order to estimate the brightness of each of the tiles, we used the relative luminance which for RGB spaces can be calculated using linear RGB components (i.e. RGB values normalised to 1 for a reference white) applied within the formula in Equation 6.12. It is worth noting that the components are not equally weighted. This is because cone cells responsible for colour vision in the human eye are most sensitive to green light and less to red and blue [137]. That was the reason why we had not simply used the average of channel values per pixel which would be the intuitive approach when trying to access image luminance. However, such an approach would introduce a significant bias towards the blue component.

$$Luminance = 0.2126R + 0.7152G + 0.0722B \tag{6.12}$$

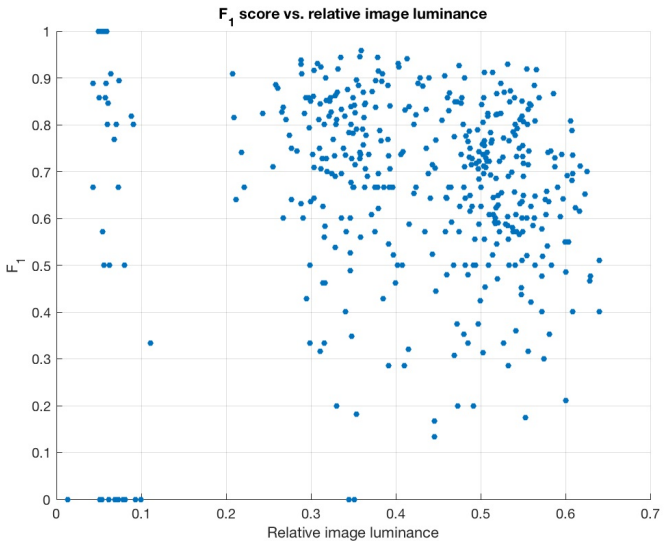


Figure 6.14: Scatter plot of F_1 scores and relative luminance for test set indicates a high degree of variance in image brightness and network's robustness to this factor.

Test set

The scatter plot presented in Figure 6.14 indicates a wide range of luminance of the tiles used as the test set. The tiles did not cover the spectrum of relative luminance of 0.7 and higher, nonetheless, the range was still significantly wide introducing a great challenge for potential detectors and testing their robustness. Our network achieved similar F_1 scores for both dim (i.e. $L < 0.4$) and bright (i.e. $L > 0.4$) tiles which suggests that relative image luminance did not affect network's ability to perform correct detection. Based on the provided scatterplot, we can deduce that the network is fairly robust to varying image brightness.

Validation subset

As mentioned in Section 6.2.1, recorded F_1 scores are likely to be underestimated due to annotation quality causing some of the model's and annotators' detections to be misclassified as FPs. Therefore, we provided the reader with another image analysis involving updated F_1 scores for the set of 15 images in order to assess the influence of the discussed image factor on both human and model performance (Figure 6.15). Yet again, considered set does not provide extreme cases of dim (i.e. $L < 0.3$) or bright (i.e. $L > 0.7$) tiles, nonetheless the variety is still significant.

The scatter plot presented in Figure 6.15 does not indicate any patterns suggesting that image brightness was a limiting factor for neither the network nor human performance. In most of the cases, our network yielded very similar or even better performance when compared to all six individuals.

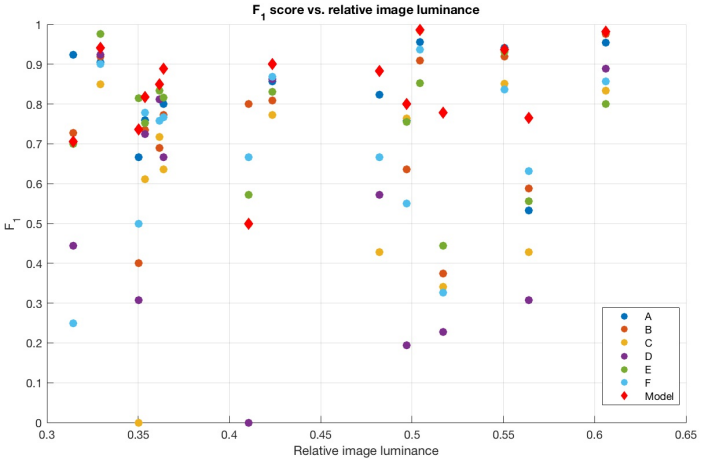


Figure 6.15: Scatter plot of F_1 scores and relative image luminance for the set of 15 images. Extreme variety in image brightness does not significantly affect neither human nor network's performance.

6.3 Ground counts

After assessing our network against human annotators (Section 6.1) and demonstrating its robustness to various image factors (Section 6.2), the last step of our evaluation process is the assessment based on the *raw* counts. It is important to note that the accurate object detection yields counts which are very close to the *real* value. The prime focus of our network's users are the *raw* counts rather than the accurately adjusted bounding boxes. Therefore, we consider the assessment based on the counts conducted on the ground, which are the closest to the *real* value, as a crucial part of our evaluation process.

When investigating our network's precision and recall (Section 6.1.1), we have shown (Figure 6.2) that it tends to overestimate the number of flower objects compared to the values indicated by the ground-truth (i.e. human annotations). However, as discussed in Section 6.1.5, the ground-truth annotations are very likely to be incomplete, missing a substantial amount of flowers present within each tile. This is another reason why we have decided to include ground counts in our data collection manuscript. These counts allow us to prove if the network indeed overestimates the number of flowers present within a particular area or if humans were unable to spot all the flowers being present within each tile.

Flower counting had been conducted by one of the Team Shrub members just before the image collection on the 14^{th} of July in all of the four data collection sites. Flowers were counted within $2m \times 2m$ area, enclosed by special orange markers at each of its corners which is presented in Figure 6.16. The GPS coordinates of each of the markers had also been recorded for the future tile extraction. We initially used coordinates of the markers along with the *ExifTool* [63] to extract the exact areas of the *validation sites* from the whole images. However, due to a slight coordinates inaccuracy caused by poor camera calibration, the extracted tiles were not well aligned with the orange

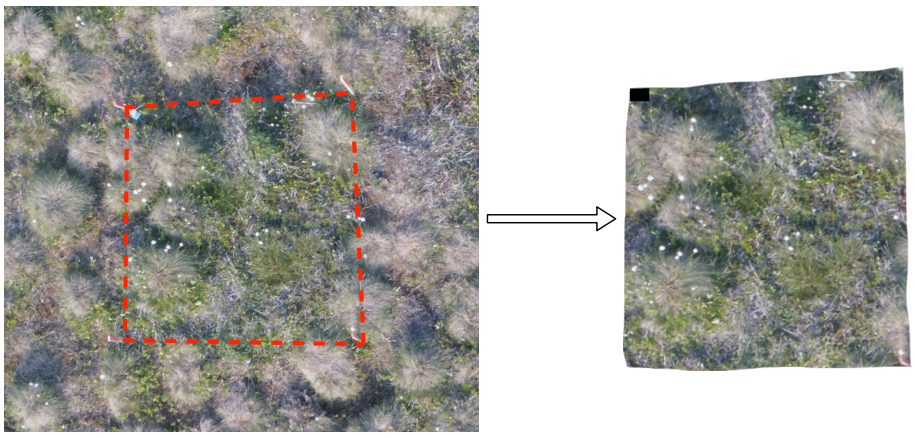


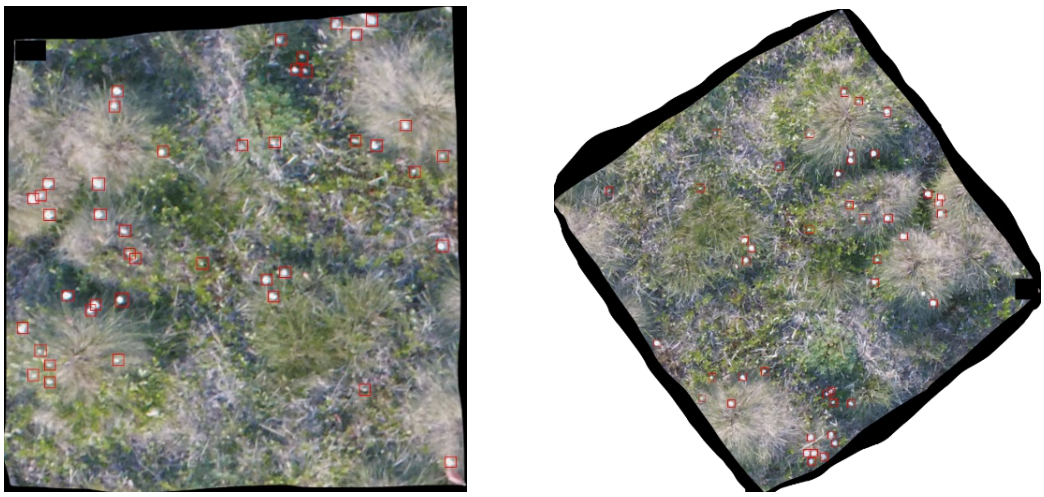
Figure 6.16: Outline on the *validation site* extraction process. The red dotted line marks the boarder indicated by the markers.

markers. Therefore, we decided to perform manual extraction by cropping each of the images around the orange markers. We are aware that this is likely to introduce an additional risk of missing some of the flowers growing on the edges of the investigated sites, however, this was the most accurate available method which worked reasonably well in practice (Figure 6.16).

This procedure resulted in generating 12 *validation site* tiles (i.e. 3 per site, each from the different flight) which were fed into our network in order to perform the detection and generate the counts. Some of the example detections are presented in Figure 6.17 and indicate a very good performance of the network. We were not able to extract a bigger number of images due to the ground-counts being collected only on a single day.

In order to perform network assessment against humans, we asked three annotators (i.e. A, B, C) to count the flowers within each of the 12 tiles. Despite its limited size, influenced by the time constraints and limited resources (i.e. lack of funds to pay for the annotation task), this set of annotations was a relatively fair representation of human performance. The results for each of the images has been presented in Appendix D.3 and show that our network was the closest to the ground count value in the majority of the cases (i.e. 58%), yielding more accurate results than human annotators who were not able to notice all the flowers.

We have also presented the same results as a proportion of the true counts in Figure 6.18 which suggest a variable performance of both humans and the network. This might be the result of some of the flowers being hardly visible from the particular angle (i.e. hidden behind grass and other objects). Nonetheless, the results clearly indicate that the network did not detect more objects than actually present within the area which also proves that the ground-truth annotations that we had used before were incomplete and were missing some of the flowers.



(a) Example tile for *PS* 1 area (b) Example tile for *PS* 3 area

Figure 6.17: Example network detections (i.e. red bounding boxes) on the extracted *validation sites*. Both tiles indicate very good performance of our network.

Measure	A	B	C	Model
Average proportion of the ground count value	0.80	0.80	0.83	**0.90**
ME	9.75	9.42	7.67	**5.33**

Table 6.1: Table of the average proportion of the ground count and the mean error (i.e. ME) generated by both subjects (i.e. network, human annotators). Our network outperforms the annotators in both measures.

As previously mentioned, our network was the closest to the *true* amount of flowers present within each *validation site* in the majority of cases when compared with human annotators, predicting 0.90 of the *true* count value on average (Table 6.1). This is a noticeably better result than the most accurate annotator (i.e. C) who managed to detect only 0.83 of the ground count value in all the evaluated tiles.

It is important to note that since we only considered the absolute amount of flowers present within each of the sites, we weighted under- and overestimation equally, meaning that if a detector missed a flower in one tile and overestimated their number by one in the other, we still considered it as a *fully accurate* result. This might seem counterintuitive at first, however, we treat each subject (i.e. network, annotators) as a counter rather than a detector in this case.

We have also compared the network with humans in case of quantity disagreement expressed as the mean error of *raw* counts (ME in Equation 6.13), presented in Table 6.1. Considering the set of 12 tiles, our network detected 5.33 less flowers on average than indicated by the ground-counts (positive number indicates underestimation). This is a satisfactory result considering a size of the validation site (i.e. $2m \times 2m$) and a typical diameter of the flower (i.e. 3-5 cm). On the other hand, human annotators presented relatively varying performance, with the best-performing individual (C) detecting almost 2 flowers less on average in each of the tiles compared to our detector (i.e. $ME = 7.67$).

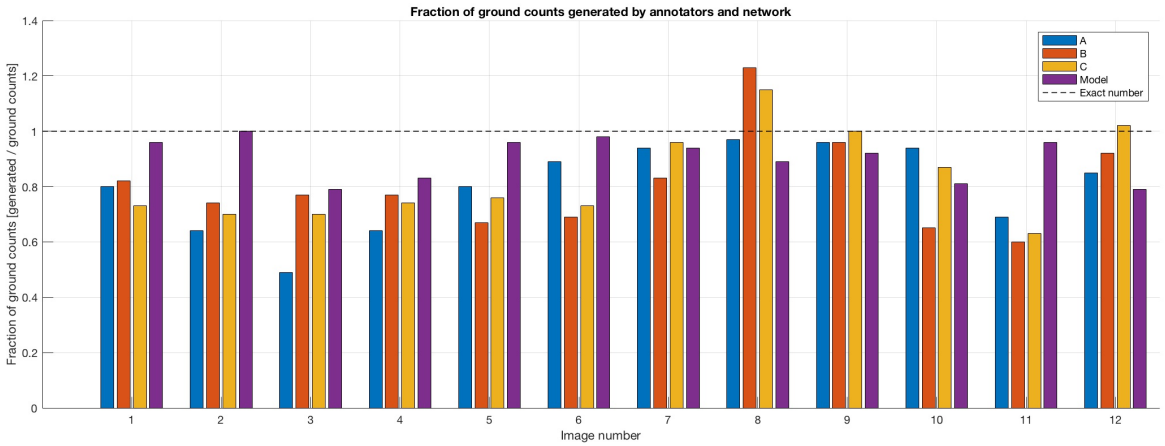


Figure 6.18: Bar graph indicating the proportion of the ground-count number generated by all subjects (i.e. network, human annotators). Our network tends to generate the number of counts being the closest to the real amount of flowers in the majority of cases.

$$ME = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i) \tag{6.13}$$

Moreover, we also assessed the network's quality of detections and did not notice any random white objects such as markers, grass or light reflections being classified as flowers. This demonstrates that the network was able to extract appropriate flower features to perform the detection. We are also fully aware of the need to perform a similar analysis on a bigger set of data (i.e. tiles), however, even this randomly selected sample suggests our network's superiority over human performance and its ability to extract appropriate flower features in any conditions.

6.4 Conclusions

Throughout our extensive evaluation process, we have investigated various assessment metrics, different factors influencing the network performance (i.e. image and annotation quality) as well as conducted both model-to-model and human-to-model assessment. These procedures allowed us to draw the following conclusions:

- **Ground-truth incompleteness**: Throughout the evaluation process (Figures 6.1 and 6.8), we have indicated that the ground-truth annotations, used as the test set, were missing some of the *true* flower objects, greatly influencing the initial assessment. Due to the FPs analysis (Section 6.1.5) we were able to update the scores resulting in more reliable assessment and proving that our network did not detect many random *non-flower* objects (i.e. network's lower precision).
- **Single performance metric**: Our extensive discussion indicates F_1 score being the most appropriate metric for the network assessment. By combining both

precision and recall with no need for confidence scores, F_1 allows for both reliable model-to-model as well as human-to-model evaluation using just a single metric.

- **Human detections inconsistency**: Varying performance among human annotators (Figures 6.3 and 6.9) indicates their inability to deliver a reliable quantitive data to investigate the patterns within the *E. vaginatum* population. This also proves how difficult and non-trivial our detection task actually is, making the evaluation process extremely complex due to the lack of any *real* ground-truth flower locations.
- **Image quality robustness**: The results presented in Section 6.2 suggest that both humans and the network are relatively robust to varying image quality factors (Figures 6.11 and 6.14). Such results indicate that our network is capable of satisfactory flower detection even in very extreme weather conditions as well as during different times of the day.
- **Outstanding network's performance**: Throughout our evaluation process we observed a satisfactory model's performance on the test set (Figures 6.5 and 6.6). However, most importantly, our network tends to outperform human annotators (Figure 6.9 and Table 6.1) yielding better detection performance, quality and consistency despite varying image quality. Such results are very likely to prove valuable for the future studies of changing patterns within different plant populations.

Chapter 7

Conclusions

The main objective of our honours project was to investigate possible approaches for a fully-automated method of real-world phenology data collection in tundra ecosystems. Such tool would enable researchers to acquire more landscape-level information on plant responses to global temperature change.

In order to achieve the desired goal, we generated an extensive dataset of 2160 images containing 34855 annotated *E. vaginatum* flower objects, making it a valuable resource for future studies regarding the phenology of this particular species as well as tundra biome in general.

Our final model, based on the highly successful Faster R-CNN architecture, used a 2-convolutional-block feature extractor (i.e. base network) in combination with a parametric ReLU (PReLU) activation unit alleviating the issue of vanishing gradients. Our modified Faster R-CNN implementation achieved a relatively high F_1 score of 0.7178 on our test set as well as 0.7460 on the *validation subset* (i.e. 15 randomly selected tiles form the test set used for human-to-model evaluation) which has been later revised to 0.8613 due to the poor quality of the ground-truth annotations (see Section 6.1.5). Hence, despite the subset containing a limited number of images (i.e. 15 tiles) as well as a poor-quality of the test set (Section 6.1.5), our model indicates the potential to outperform human annotators in the task of *E. vaginatum* flower detection and counting (highest human F_1 score of 0.7980 on the *validation subset*).

Moreover, when considering only *raw* count values based on the counts gathered at the place of data collection (i.e. ground counts), our network managed to detect objects accounting for 0.90 of the ground-truth value, outperforming human annotators whose best score was only 0.83.

Presented results indicate the potential of deep-learning-based methods for accurate and fast detection and counting of small objects, despite using high spatial resolution drone imagery characterised by high amount of noise and a limited field of view. Therefore, our project provides an efficient and fully-automated method for species tracking with a potential to be used as an innovative tool for real-world ecological data collection in tundra as well as other flowering ecosystems such as grasslands or deserts.

Chapter 8

Future Work

In this chapter, we will discuss some potential approaches to extend the project in the next year (i.e. year 2).

8.1 Alternative approach

Due to the limited amount of annotated images and the annotation process being very time-consuming, we would like to investigate the most recent approaches for semi-supervised and unsupervised methods for object detection. Such approach would allow us to fully utilise an extensive database of drone imagery collected by the Team Shrub members [153] throughout the span of previous 2-3 years with no or a limited need for carefully annotated objects.

Despite showing a great potential to further improve automatic phenology data collection, most of the recent unsupervised approaches involving region proposals [28], object saliency maps [154] or deep descriptor transforming [172] are used in object discovery and localisation tasks on standard datasets containing objects of large sizes. Such tasks are substantially less challenging than dealing with a very a noisy drone imagery involving very small flower objects.

A possible alternative would be to utilise semi-supervised methods involving only image-level labelled data [176], visual and semantic knowledge transfer [163] or active learning [135], some of which already achieving close to state-of-the-art detectors performance levels (including Faster R-CNN) [176]. However, due to their novelty and a lack of testing on more challenging tasks involving very small objects, these approaches need to be studied further and evaluated towards adjustment to our task.

8.2 Dataset

In case of the fully and semi-supervised learning approaches, an extensive dataset including good-quality annotations is crucial for training a well-performing detector.

Therefore, we would like to revise currently available 2160 tiles and supplement them with additional flower annotations that had not been previously spotted by other people. We do not expect this task to be extremely time-consuming since the annotators will receive already marked flowers and will need to add extra annotations for any omitted objects with an option to exclude the existing bounding boxes which were annotated incorrectly.

Moreover, in order to make our model more suitable for *E. vaginatum* flower objects detection from higher altitudes, we are planning to extend our initial dataset with additional images collected at 25 meters above the ground level. This would allow us to investigate how deep-learning-based architectures are able to perform is even more challenging conditions, trying to detect even smaller objects. We expect that bigger dataset (i.e. more annotated tiles from each altitude) would allow our model to achieve better generalisation ability, hence increasing its detection performance.

Additionally we might also extend our dataset with flower annotations of other species which grow within the investigated tundra biome such as *Dryas integrifolia* [87], *Lupinus arcticus* [146] or *Salix pulchra* [156]. This way we could investigate if our model could be applied to a more broad task of flower object detection and species differentiation.

8.3 Model architecture and implementation

With an already available dataset, we plan to further explore possible approaches to improve the performance of our model regarding both detection quality as well as the time required to process a single image (i.e. test time).

We have made most of our architecture and hyper-parameter choices based on the test set, therefore, our main focus would be to introduce a validation set in order to track the level of data overfitting by the network. Moreover, we aim to utilise methods which had already been proven to be very successful in other detection tasks such as dropout [159] (Section 4.1.5) and batch normalisation [77] (Section 4.1.6) layers as well a deeper base network architecture (Section 5.2.2) for better feature extraction of more complex representations [134]. Such approach would involve further hyper-parameter adjustment with a high possibility of lengthening the training process. Nevertheless, these enhancements are very likely to potentially improve the quality of detections generated by our model.

Furthermore, if we decide to reuse our Faster R-CNN architecture next year, we plan to reimplement our entire model using TensorFlow [165] or PyTorch [128] which may give us more freedom in various parameter tweaking as well as allowing for faster code execution (i.e. applying multi-GPU training). We could utilise and adjust some of the already existing implementations such as [24]. However, the final decision has to be based on further research and benefits of using other frameworks or libraries such as TensorFlow and PyTorch over MATLAB [109].

8.4 Real-world application testing

Our final project enhancement would involve testing our flower detector in a real-world scenario involving almost real-time *E. vaginatum* flower detection from the actively sensing unmanned aerial vehicle steered by a human. To do so, we would require a relatively fast data transfer between the drone and the computing node with the trained model performing constant image splitting, object detection and tile merging to provide the researchers with almost instant flower count data. Although not necessary, a successful execution of such a task would be synonymous with reaching the ultimate goal of implementing a fully-automated tool capable of almost instant real-world ecological data collection.

Such a scenario might be feasible since our network is already able to process a single tile in less than $0.6s$ for the current set-up. This does not seem to be a fast enough testing speed due to the drone generating a single high-resolution image (5320×4200) every 4-5 seconds with each image requiring a further split into approximately 150 tiles (440×440). However, with possible future architecture enhancements focused on further performance improvement and approximation by using only a fraction of the tiles from each image, our model has a potential of very efficient phenological data generation.

Appendices

Appendix A

Dataset

A.1 Data collection manuscript

Platform: Phantom4Advanced [40]

Area: four $100m \times 100m$ or $50m \times 200m$ transects (depending on topography and site)

Data collection frequency: Every three days to a week during peak flowering and once a week to every two weeks during the lead up and tail of the flowering season.

Data collection times: Focus on solar maxima for time series flights, but also collect data during low light conditions and in variable cloud and cloudy days.

Locations: Phenology ridge (covering the area of the phenology transects and phenocams), $PS1$ or $PS2$ and $PS3$ or $PS4$ (perhaps install a couple of the additional 6 phenocams at these sites). Try to cover both the Herschel vegetation type and Komakuk vegetation type in each flight or conduct multiple flights in each vegetation type.

Flight altitude: The priority are 12m and 25m surveys, if flight time permits another survey of the same extent at 50 m with a few photos at 100m, perhaps include a ladder climb with one photo every 10m or something similar if possible up to 100m at the beginning or end of the flight. Exact flight extents and data collection can be determined by flight time so that ideally all data collection can be conducted using one flight of the Phantom4Advanced.

Photo overlap: Following protocols for SfM builds, $\sim 75\%$ front and $\sim 50\%$ side overlaps.

Imagery format: Collect both RAW and JPEG formats or only RAW if it is too time consuming to save both.

Markers: at each corner of the area plus additional four to mark the *validation site* for the ground counts.

Additional data: Collect metadata following general protocols on flight and weather conditions. GNSS data for marker positions and phenology transect. Meta data for

flights including georeferencing of all imagery. On site assessment of flower development for key species: *Eriophorum vaginatum*, *Dryas integrifolia*, etc.

A.2 Sky conditions scale

Scale	Description
0	Clear sky
1	Haze
2	Thin cirrus- sun not obscured
3	Thin cirrus- sun obscured
4	Scattered cumulus- sun not obscured
5	Cumulus over most of sky- sun not obscured
6	Cumulus- sun obscured
7	Complete cumulus cover
8	Stratus- sun obscured
9	Drizzle

Table A.1: Scale used to describe sky conditions.

Appendix B

Methodology

B.1 Complete architecture

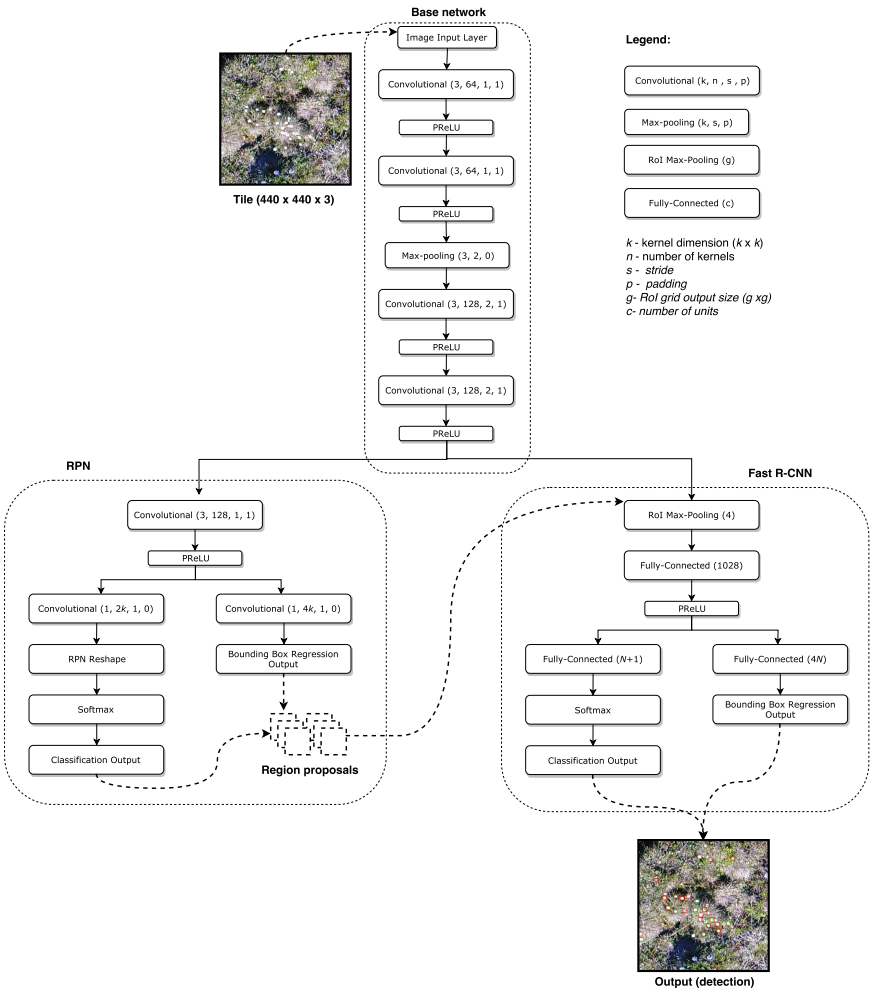


Figure B.1: Complete Faster R-CNN architecture.

Appendix C

Training and experiments

C.1 Training progress graphs

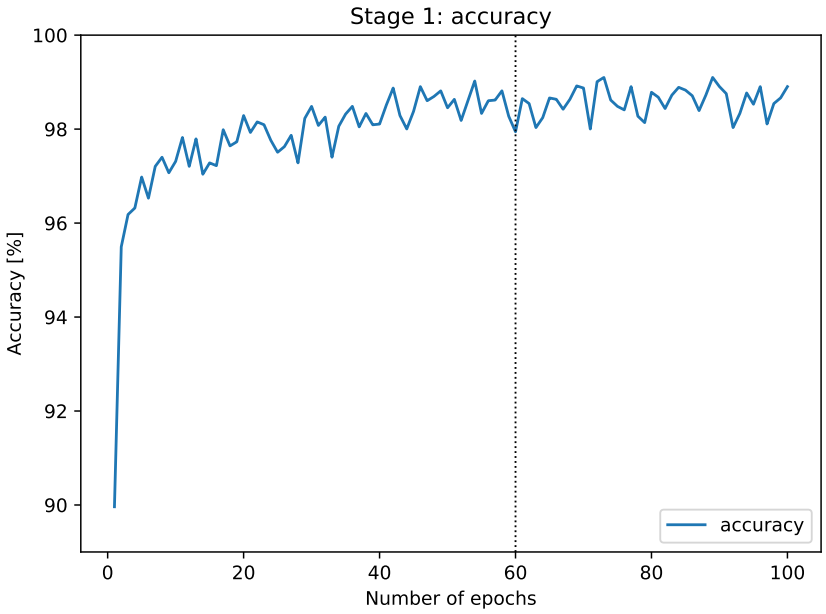


Figure C.1: Training process expressed as an accuracy plot (i.e. averaged mini-batch accuracies for each epoch) of the RPN in the first training stage when trained for 100 epochs. The black dotted line denotes the training length of our final model implementation (i.e. 60 epochs).

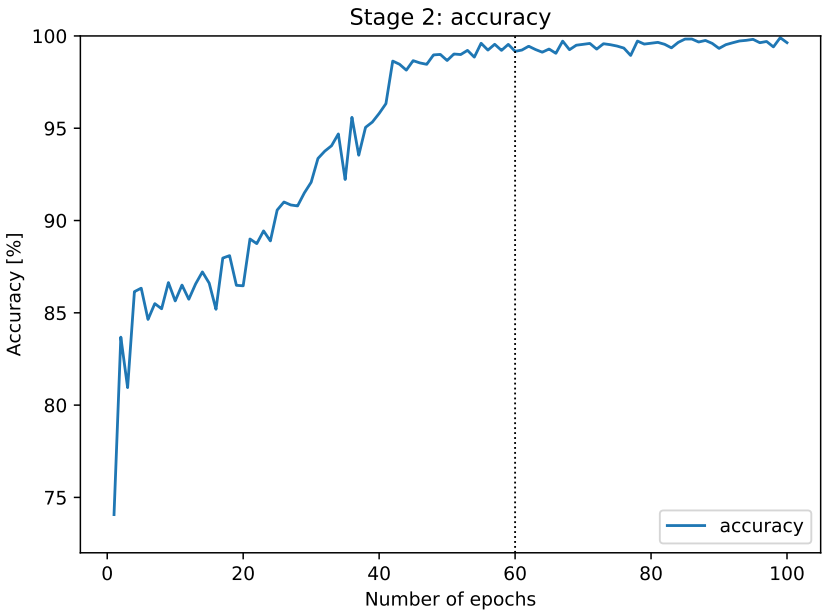


Figure C.2: Training process expressed as an accuracy plot of the Fast R-CNN detector component in the second stage when trained for 100 epochs. The black dotted line denotes the training length of our final model implementation (i.e. 60 epochs).

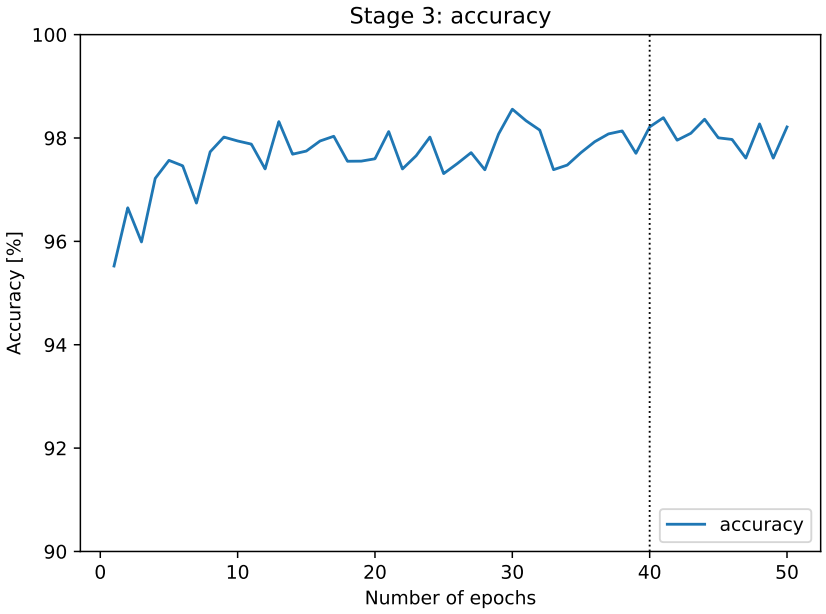


Figure C.3: Accuracy plot illustrating the training progress of the fine-tuned RPN during the third stage of training when trained for 50 epochs. The black dotted line denotes the training length of our final model implementation (i.e. 40 epochs).

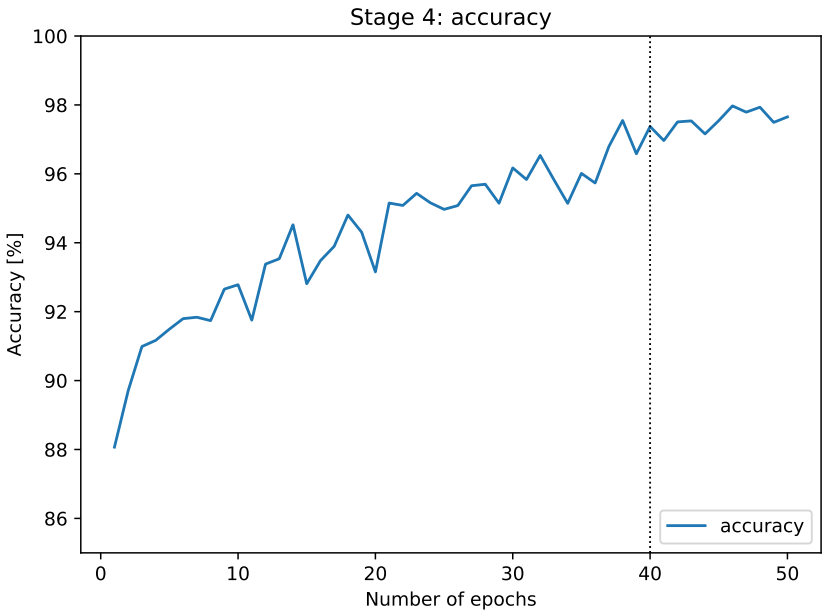


Figure C.4: Accuracy plot illustrating the training progress of the fine-tuned Fast R-CNN detector component during the third stage of training when trained for 50 epochs. The black dotted line denotes the training length of our final model implementation (i.e. 40 epochs).

Appendix D

Evaluation

D.1 Average precision

		Result							
		Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Ground-truth	Model
Target	Annotator A	1.0000	0.7586	0.5231	0.6537	0.5991	0.6160	0.5226	0.8221
	Annotator B	0.7568	1.0000	0.5240	0.6337	0.5410	0.6069	0.4902	0.8289
	Annotator C	0.5482	0.5412	1.0000	0.5336	0.3939	0.4850	0.3906	0.6069
	Annotator D	0.6953	0.6680	0.5368	1.0000	0.5382	0.6886	0.4607	0.7762
	Annotator E	0.6569	0.5940	0.4144	0.5654	1.0000	0.4959	0.4528	0.6608
	Annotator F	0.6562	0.6518	0.5082	0.6867	0.4706	1.0000	0.4546	0.7056
	Ground-truth	0.6182	0.5805	0.4387	0.5148	0.4894	0.5029	1.0000	0.7321
	Model	0.5495	0.5244	0.3539	0.4490	0.4390	0.4028	0.5250	1.0000

Figure D.1: Average precision scores for each individual annotator and the model evaluated against each other on the set of 15 images (target and result annotations represented in rows and columns respectively).

D.2 Log-average miss-rate

		Result							
		Annotator A	Annotator B	Annotator C	Annotator D	Annotator E	Annotator F	Ground-truth	Model
Target	Annotator A	0.0000	0.6862	0.9457	0.7438	0.9583	0.9102	0.9833	0.6488
	Annotator B	0.9156	0.0000	0.9488	0.9022	0.9524	0.9351	0.9805	0.7260
	Annotator C	0.9380	0.9386	0.0000	0.9436	0.9693	0.9681	0.9781	0.9071
	Annotator D	0.9400	0.9380	0.9768	0.0000	0.9767	0.9325	0.9839	0.8363
	Annotator E	0.8719	0.8267	0.9613	0.7776	0.0000	0.9462	0.9909	0.8137
	Annotator F	0.9225	0.8881	0.9580	0.8205	0.9717	0.0000	0.9846	0.7901
	Ground-truth	0.8368	0.7355	0.9381	0.7951	0.9661	0.9307	0.0000	0.6267
	Model	0.8923	0.8055	0.9736	0.8716	0.9623	0.9598	0.9738	0.0000

Figure D.2: LAMR scores for each individual annotator and the model evaluated against each other on the set of 15 images (target and result annotations represented in rows and columns respectively).

D.3 Ground counts

Image	A	B	C	Model	Ground count
1	36	37	33	43	45
2	30	35	33	47	47
3	23	36	33	37	47
4	30	36	35	39	47
5	36	30	34	43	45
6	40	31	33	44	45
7	67	59	68	67	71
8	69	87	82	63	71
9	68	68	71	65	71
10	49	34	45	42	52
11	36	31	33	50	52
12	44	48	53	41	52

Table D.3: *Raw* counts per image for all subjects (i.e. annotators and the network) along with the *true* ground-count values.

Bibliography

[1] Pulkit Agrawal, Ross B. Girshick, and Jitendra Malik. Analyzing the Performance of Multilayer Neural Networks for Object Recognition. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII*, pages 329–344, 2014.

[2] Haider Allamy. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). 12 2014.

[3] Paul E. Allen and Chuck Thorpe. Some Approaches to Finding Birds in Video Imagery. Technical Report CMU-RI-TR-91-34, Carnegie Mellon University, Pittsburgh, PA, December 1991.

[4] Javed A. Aslam, Emine Yilmaz, and Virgiliu Pavlu. A geometric interpretation of r-precision and its correlation with average precision. In *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, August 15-19, 2005*, pages 573–574, 2005.

[5] D Bajzak and John Piatt. Computer-aided procedure for counting waterfowl on aerial photographs. 18:125–129, 01 1990.

[6] Aurele Balavoine, Justin K. Romberg, and Christopher J. Rozell. Convergence and Rate Analysis of Neural Networks for Sparse Approximation. *IEEE Trans. Neural Netw. Learning Syst.*, 23(9):1377–1389, 2012.

[7] Shannon M. Barber-Meyer, Gerald L. Kooyman, and Paul J. Ponganis. Estimating the relative abundance of emperor penguins at inaccessible colonies using satellite imagery. *Polar Biology*, 30(12):1565–1570, 2007.

[8] Jonathan Barichivich, Keith R. Briffa, Ranga B. Myneni, Timothy J. Osborn, Thomas M. Melvin, Philippe Ciais, Shilong Piao, and Compton Tucker. Largescale variations in the vegetation growing season and annual cycle of atmospheric CO2 at high northern latitudes from 1950 to 2011. *Global Change Biology*, 19(10):3167–3183, 2013.

[9] C. J. Bartleson. The Combined Influence of Sharpness and Graininess on the Quality of Colour Prints. *The Journal of Photographic Science*, 30(2):33–38, 1982.

[10] Rodrigo Benenson, Mohamed Omran, Jan Hendrik Hosang, and Bernt Schiele. Ten Years of Pedestrian Detection, What Have We Learned? In *Computer Vision - ECCV 2014 Workshops - Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part II*, pages 613–627, 2014.

[11] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[12] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Improving Object Detection with One Line of Code. *CoRR*, abs/1704.04503, 2017.

[13] Leon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization Methods for Large-Scale Machine Learning. *CoRR*, abs/1606.04838, 2016.

[14] Y-Lan Boureau, Francis R. Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pages 2559–2566, 2010.

[15] Gavin J. Bowden, Holger R. Maier, and Graeme C. Dandy. Optimal division of data for neural network models in water resources applications. *Water Resources Research*, 38(2), 2002.

[16] Jason Brownlee. What is the difference between test and validation datasets? `https://machinelearningmastery.com/difference-test-validation-datasets/`. Accessed: 20-03-2018.

[17] Gang Cao, Lihui Huang, Huawei Tian, Xianglin Huang, Yongbin Wang, and Ruicong Zhi. Contrast Enhancement of Brightness-Distorted Images by Improved Adaptive Gamma Correction. *CoRR*, abs/1709.04427, 2017.

[18] Christin Carl, Dirk Landgraf, Marieke van der Maaten-Theunissen, Peter Biber, and Hans Pretzsch. Robinia pseudoacacia L. flower analyzed by using an Unmanned Aerial Vehicle (UAV). *Remote Sensing*, 9(11), 2017.

[19] Dominique Chabot and Charles M. Francis. Computerautomated bird detection and counts in highresolution aerial images: a review. *Journal of Field Ornithology*, 87(4):343–359, 2016.

[20] Liang-Chieh Chen, Alexander Hermans, George Papandreou, Florian Schroff, Peng Wang, and Hartwig Adam. MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features. *CoRR*, abs/1712.04837, 2017.

[21] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. *CoRR*, abs/1412.7062, 2014.

[22] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *CoRR*, abs/1606.00915, 2016.

[23] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *CoRR*, abs/1706.05587, 2017.

[24] Xinlei Chen and Abhinav Gupta. An Implementation of Faster R-CNN with Study for Region Sampling. *CoRR*, abs/1702.02138, 2017.

[25] Zhong Chen, Ting Zhang, and Chao Ouyang. End-to-End Airplane Detection Using Transfer Learning in Remote Sensing Images. *Remote Sensing*, 10(1), 2018.

[26] Zhong Chen, Ting Zhang, and Chao Ouyang. End-to-End Airplane Detection Using Transfer Learning in Remote Sensing Images. *Remote Sensing*, 10:139, 01 2018.

[27] N. Ng Kuang Chern, Poo Aun Neow, and M. H. Ang. Practical issues in pixel-based autofocusing for machine vision. volume 3, pages 2791–2796 vol.3, 2001.

[28] Minsu Cho, Suha Kwak, Cordelia Schmid, and Jean Ponce. Unsupervised object discovery and localization in the wild: Part-based matching with bottom-up region proposals. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1201–1210, 2015.

[29] Suman Kumar Choudhury, Ram Prasad Padhy, and Pankaj Kumar Sa. Faster R-CNN with densenet for scale aware pedestrian detection vis-a-vis hard negative suppression. In *27th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2017*, pages 1–6, 2017.

[30] Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jurgen Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2852–2860, 2012.

[31] Elsa E. Cleland, Isabelle Chuine, Annette Menzel, Harold A. Mooney, and Mark D. Schwartz. Shifting plant phenology in response to global change. *Trends in Ecology & Evolution*, 22(7):357 – 365, 2007.

[32] Mitchell B. Cruzan, Ben G. Weinstein, Monica R. Grasty, Brendan F. Kohrn, Elizabeth C. Hendrickson, Tina M. Arredondo, and Pamela G. Thompson. Small unmanned aerial vehicles (microUAVs, drones) in plant ecology. *Applications in Plant Sciences*, 4(9), 2016.

[33] Yann Le Cun, Ido Kanter, and Sara A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Phys. Rev. Lett.*, 66:2396–2399, 1991.

[34] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 379–387, 2016.

[35] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 886–893, 2005.

[36] Yann N. Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. RMSProp and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015.

[37] J.P. Davey. Autofocus algorithms 1: Effect of astigmatism. In *LEO Instruments - Technical Memo*, 1992.

[38] Jesse Davis and Mark Goadrich. The relationship between Precision-Recall and ROC curves. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 233–240, 2006.

[39] Ludovic Denoyer and Patrick Gallinari. Deep Sequential Neural Network. *CoRR*, abs/1410.0510, 2014.

[40] DJI. Phantom 4 Pro Specification. `https://www.dji.com/phantom-4-pro/info#specs`. Accessed: 29-03-2018.

[41] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, 2012.

[42] Leonardo Araujo dos Santos. Artificial Inteligence. `https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolutional_neural_networks.html`. Accessed: 14-03-2018.

[43] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[44] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *CoRR*, abs/1603.07285, 2016.

[45] Christian Eggert, Stephan Brehm, Anton Winschel, Dan Zecha, and Rainer Lienhart. A closer look: Small object detection in faster R-CNN. In *2017 IEEE International Conference on Multimedia and Expo, ICME 2017, Hong Kong, China, July 10-14, 2017*, pages 421–426, 2017.

[46] Lars Eidnes and Arild Nøkland. Shifting Mean Activation Towards Zero with Bipolar Activation Functions. *CoRR*, abs/1709.04054, 2017.

[47] Dan Ellis and Nelson Morgan. Size matters: an empirical study of neural network training for large vocabulary continuous speech recognition. In *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '99, Phoenix, Arizona, USA, March 15-19, 1999*, pages 1013–1016, 1999.

[48] S. J. Erasmus and K. C. A. Smith. An automatic focusing and astigmatism correction system for the SEM and CTEM. *Journal of Microscopy*, 127(2):185–199, 1982.

[49] Andreas Ess, Bastian Leibe, and Luc J. Van Gool. Depth and Appearance for Mobile Scene Analysis. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8, 2007.

[50] Mark Everingham, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[51] Serena Yeung Fei-Fei Li, Justin Johnson. CS231 lecture slides. `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`. Accessed: 04-04-2018.

[52] Pedro F. Felzenszwalb, Ross B. Girshick, David A. McAllester, and Deva Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.

[53] Peter A. Flach and Meelis Kull. Precision-Recall-Gain Curves: PR Analysis Done Right. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 838–846, 2015.

[54] Gianni Franchi and Jesús Angulo. Bagging Stochastic Watershed on Natural Color Image Segmentation. In *Mathematical Morphology and Its Applications to Signal and Image Processing - 12th International Symposium, ISMM 2015, Reykjavik, Iceland, May 27-29, 2015. Proceedings*, pages 422–433, 2015.

[55] Robert A. Frazor and Wilson S. Geisler. Local luminance and contrast in natural images. *Vision Research*, 46(10):1585 – 1598, 2006.

[56] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and José García Rodríguez. A Review on Deep Learning Techniques Applied to Semantic Segmentation. *CoRR*, abs/1704.06857, 2017.

[57] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[58] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 580–587, 2014.

[59] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256, 2010.

[60] Geoff Groom, Michael Stjernholm, Rasmus Due Nielsen, Andrew Fleetwood, and Ib Krag Petersen. Remote sensing image data and automated analysis to

describe marine bird distributions and abundances. *Ecological Informatics*, 14:2 – 8, 2013. The analysis and application of spatial ecological data to support the conservation of biodiversity.

[61] Isabelle Guyon. A Scaling Law for the Validation-Set Training-Set Size Ratio. In *AT & T Bell Laboratories*, 1997.

[62] Guangxing Han, Xuan Zhang, and Chongrong Li. Revisiting Faster R-CNN: A Deeper Look at Region Proposal Network. In *Neural Information Processing - 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part III*, pages 14–24, 2017.

[63] Phil Harvey. ExifTool by Phil Harvey. `https://www.sno.phy.queensu.ca/~phil/exiftool/`. Accessed: 17-03-2018.

[64] R. Hassen, Z. Wang, and M. M. A. Salama. Image Sharpness Assessment Based on Local Phase Coherence. *IEEE Transactions on Image Processing*, 22(7):2798–2810, 2013.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. *CoRR*, abs/1502.01852, 2015.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(9):1904–1916, 2015.

[68] G.H.R. Henry and U. Molau. Tundra plants and climate change: the International Tundra Experiment (ITEX). *Global Change Biology*, 3(S1):1–9, 2003.

[69] Alex Hernández-García and Peter König. Do deep nets really need weight decay and dropout? *CoRR*, abs/1802.07042, 2018.

[70] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[71] D. M. Holburn and K. C. A. Smith. Topographical analysis in the SEM using an automatic focusing technique. *Journal of Microscopy*, 127(1):93103, 1982.

[72] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. How good are detection proposals, really? *CoRR*, abs/1406.6962, 2014.

[73] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. Learning Non-maximum Suppression. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6469–6477, 2017.

[74] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama,

and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.

[75] Mohamed E. Hussein, Fatih Porikli, and Larry S. Davis. A Comprehensive Evaluation Framework and a Comparative Study for Human Detectors. *IEEE Trans. Intelligent Transportation Systems*, 10(3):417–427, 2009.

[76] Intel. Intel Xeon E5-2640 specification. `https://ark.intel.com/products/92984/Intel-Xeon-Processor-E5-2640-v4-25M-Cache-2_40-GHz`. Accessed: 04-04-2018.

[77] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167, 2015.

[78] J-Painter. J-Painter online image editor. `http://izhuk.com/painter/photo/`. Accessed: 29-03-2018.

[79] Eric Kauderer-Abrams. Quantifying Translation-Invariance in Convolutional Neural Networks. *CoRR*, abs/1801.01450, 2018.

[80] Gregoire H. G. Kerr, Christian Fischer, and Ralf Reulke. Reliability assessment for remote sensing data: Beyond Cohen's kappa. In *2015 IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2015, Milan, Italy, July 26-31, 2015*, pages 4995–4998, 2015.

[81] Roxaneh Khorsand, Steven Oberbauer, Gregory Starr, Inga La Puma, Eric Pop, Lorraine Ahlquist, and Tracey Baldwin. Plant phenological responses to a long-term experimental extension of growing season and soil warming in the tussock tundra of alaska. 21, 07 2015.

[82] Ron Kimmel, Cuiping Zhang, Alexander M. Bronstein, and Michael M. Bronstein. Are MSER Features Really Interesting? *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(11):2316–2320, 2011.

[83] Frederick A.A. Kingdom. Lightness, brightness and transparency: A quarter century of new ideas, captivating demonstrations and unrelenting controversy. *Vision Research*, 51(7):652 – 673, 2011. Vision Research 50th Anniversary Issue: Part 1.

[84] Diederik P. Kingma and Jimmy Ba. Adam: A method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[85] Patrick T. Komiske, Eric M. Metodiev, and Matthew D. Schwartz. Deep learning in color: towards automated quark/gluon jet discrimination. *Journal of High Energy Physics*, 2017(1), 2017.

[86] Saiprasad Koturwar and Shabbir Merchant. Weight Initialization of Deep Neural networks(DNNs) using Data Statistics. *CoRR*, abs/1710.10570, 2017.

[87] P. G. Krannitz. Reproductive ecology of Dryas integrifolia in the high Arctic semi-desert. *Canadian Journal of Botany*, 74(9):1451–1460, 1996.

[88] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[89] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *CoRR*, abs/1704.08863, 2017.

[90] Ilmari Kurki, Tarja Peromaa, and Jussi Saarinen. Visual Features Underlying Perceived Brightness as Revealed by Classification Images. *PLOS ONE*, 4(10):1–8, 2009.

[91] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[92] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Muller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, 1998.

[93] Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, gated, and tree. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 464–472, 2016.

[94] Jianan Li, Xiaodan Liang, Shengmei Shen, Tingfa Xu, and Shuicheng Yan. Scale-aware Fast R-CNN for Pedestrian Detection. *CoRR*, abs/1510.08160, 2015.

[95] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014*, pages 661–670, 2014.

[96] Jie Liu, Xingkun Gao, Nianyuan Bao, Jie Tang, and Gangshan Wu. Deep convolutional neural networks for pedestrian detection with skip pooling. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 2056–2063, 2017.

[97] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, pages 21–37, 2016.

[98] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[99] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[100] S. Mader and G. Grenzdorffer. Automatic sea bird detection from high resolution aerial imagery. XLI-B7, 2016.

[101] Raman Maini and Himanshu Aggarwal. A Comprehensive Review of Image Enhancement Techniques. *CoRR*, abs/1003.4053, 2010.

[102] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[103] X. Marichal, Wei-Ying Ma, and HongJiang Zhang. Blur determination in the compressed domain using DCT information. volume 2, pages 386–390 vol.2, 1999.

[104] MathWorks. evaluateDetectionMissRate. https://www.mathworks.com/help/vision/ref/evaluatedetectionmissrate.html. Accessed: 16-03-2018.

[105] MathWorks. evaluateDetectionPrecision. https://www.mathworks.com/help/vision/ref/evaluatedetectionprecision.html. Accessed: 16-03-2018.

[106] MathWorks. fasterRCNNObjectDetector class. https://www.mathworks.com/help/vision/ref/fasterrcnnobjectdetector-class.html. Accessed: 14-03-2018.

[107] MathWorks. fastRCNNObjectDetector class. https://www.mathworks.com/help/vision/ref/fastrcnnobjectdetector-class.html?s_tid=doc_ta. Accessed: 14-03-2018.

[108] MathWorks. ImageInputLayer. https://www.mathworks.com/help/nnet/ref/nnet.cnn.layer.imageinputlayer.html. Accessed: 13-03-2018.

[109] MathWorks. Matlab. https://uk.mathworks.com/products/matlab/whatsnew.html. Accessed: 31-03-2018.

[110] MathWorks. trainFasterRCNNObjectDetector method. https://www.mathworks.com/help/vision/ref/trainfasterrcnnobjectdetector.html#bvkk009-1-trainingData. Accessed: 20-03-2018.

[111] Anderson Matthew, Motta Ricardo, Chandrasekar Srinivasan, and Michael Stokes. Proposal for a Standard Default Color Space for the InternetsRGB. In *4th Color and Imaging Conference Final Program and Proceedings*, pages 238–245, 1996.

[112] Sancho McCann. It's a bird it's a plane it depends on your classifier's threshold. https://sanchom.wordpress.com/tag/average-precision/. Accessed: 10-03-2018.

[113] Microsoft. Microsoft Windows 10 Pro specification. https://www.microsoft.com/en-gb/store/b/windows. Accessed: 04-04-2018.

[114] John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors. *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*. Morgan Kaufmann, 1992.

[115] J. Nagi, F. Ducatelle, G. A. Di Caro, D. CireÅan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 342–347, 2011.

[116] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[117] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning Deconvolution Network for Semantic Segmentation. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1520–1528, 2015.

[118] Nvidia. Nvidia Geforce CTX 1080 Ti specs. `https://www.geforce.co.uk/hardware/10series/geforce-gtx-1080-ti/#specs`. Accessed: 31-03-2018.

[119] Intergovernmental Panel on Climate Change. *Climate Change 2013 - The Physical Science Basis: Working Group I Contribution to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 2014.

[120] Sungheon Park and Nojun Kwak. Analysis on the Dropout Effect in Convolutional Neural Networks. In *Computer Vision - ACCV 2016 - 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II*, pages 189–204, 2016.

[121] Taejin Park, Sangram Ganguly, Hans Tommervik, Eugenie S Euskirchen, Kjell-Arild Hogda, Stein Rune Karlsen, Victor Brovkin, Ramakrishna R Nemani, and Ranga B Myneni. Changes in growing season duration and productivity of northern vegetation inferred from long-term remote sensing data. *Environmental Research Letters*, 11(8), 2016.

[122] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large Kernel Matters - Improve Semantic Segmentation by Global Convolutional Network. *CoRR*, abs/1703.02719, 2017.

[123] Viet-Quoc Pham, Satoshi Ito, and Tatsuo Kozakaya. BiSeg: Simultaneous Instance Segmentation and Semantic Segmentation with Fully Convolutional Networks. *CoRR*, abs/1706.02135, 2017.

[124] Shilong Piao, Pierre Friedlingstein, Philippe Ciais, Nicolas Viovy, and Jerome Demarty. Growing season extension and its impact on terrestrial carbon cycle in the Northern Hemisphere over the past 2 decades. *Global Biogeochemical Cycles*, 21(3), 2007.

[125] Pedro H. O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to Segment Object Candidates. *CoRR*, abs/1506.06204, 2015.

[126] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. In *Mach. Learn. Technol.*, volume 2, 2008.

[127] Janet Prevey, Mark Vellend, Nadja Rger, Robert D. Hollister, Anne D. Bjorkman, Isla H. MyersSmith, Sarah C. Elmendorf, Karin Clark, Elisabeth J. Cooper, Bo Elberling, Anna M. Fosaa, Gregory H. R. Henry, Toke T. Hye, Ingibjrg S. Jnsdttir, Kari Klanderud, Esther Lvesque, Marguerite Mauritz, Ulf Molau, Susan M. Natali, Steven F. Oberbauer, Zoe A. Panchen, Eric Post, Sabine B. Rumpf, Niels M. Schmidt, Edward A. G. Schuur, Phillip R. Semenchuk, Tiffany Troxler, Jeffrey M. Welker, and Christian Rixen. Greater temperature sensitivity of plant phenology at colder sites: implications for convergence across northern latitudes. *Global Change Biology*, 23(7):2660–2671, 2017.

[128] PyTorch. PyTorch specification. `http://pytorch.org/about/`. Accessed: 06-04-2018.

[129] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[130] Luc De Raedt. Logic, Probability and Learning, or an Introduction to Statistical Relational Learning. In *Advances in Artificial Intelligence - SBIA 2008, 19th Brazilian Symposium on Artificial Intelligence, Savador, Brazil, October 26-30, 2008. Proceedings*, page 5, 2008.

[131] Atiqur Rahman and Yang Wang. Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation. In *Advances in Visual Computing - 12th International Symposium, ISVC 2016, Las Vegas, NV, USA, December 12-14, 2016, Proceedings, Part I*, pages 234–244, 2016.

[132] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. *CoRR*, abs/1710.05941, 2017.

[133] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016.

[134] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[135] Phill-Kyu Rhee, Enkhbayar Erdenee, Shin Dong Kyun, Minhaz Uddin Ahmed, and SongGuo Jin. Active and semi-supervised learning for object detection with imperfect data. *Cognitive Systems Research*, 45:109–123, 2017.

[136] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[137] W.A.H. Rushton and H.D. Baker. Red/green sensitivity in normal vision. *Vision Research*, 4(1):75 – 85, 1964.

[138] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *CoRR*, abs/1409.0575, 2014.

[139] Jessica L. Schedlbauer, Ned Fetcher, Katherine Hood, Michael L. Moody, and Jianwu Tang. Effect of growth temperature on photosynthetic capacity and respiration in three ecotypes of Eriophorum vaginatum. *Ecology and Evolution*, 2018.

[140] Nicol N. Schraudolph. Centering neural network gradient factors. In *Neural Networks: Tricks of the Trade - Second Edition*, pages 205–223. Springer, 2012.

[141] ScreenShot. Screenshot. `https://screenshot.net/online-image-editor.html`. Accessed: 29-03-2018.

[142] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *CoRR*, abs/1312.6229, 2013.

[143] Khaled Shaalan. A Survey of Arabic Named Entity Recognition and Classification. *Computational Linguistics*, 40(2):469–510, 2014.

[144] Muhammad Shahid, Andreas Rossholm, Benny Lövström, and Hans-Jürgen Zepernick. No-reference image and video quality assessment: a classification and review of recent approaches. *EURASIP Journal on Image and Video Processing*, 2014(1), 2014.

[145] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2217–2225, 2016.

[146] Gregory J. Sharam and Roy Turkington. Diurnal cycle of sparteine production in Lupinus arcticus. *Canadian Journal of Botany*, 83(10):1345–1348, 2005.

[147] A. Sharma. Understanding activation functions in neural networks, 2017.

[148] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *CoRR*, abs/1605.06211, 2016.

[149] Jonathan Shen, Noranart Vesdapunt, Vishnu Naresh Boddeti, and Kris M. Kitani. In Teacher We Trust: Learning Compressed Models for Pedestrian Detection. *CoRR*, abs/1612.00478, 2016.

[150] Jamie Shotton, Matthew Johnson, and Roberto Cipolla. Semantic texton forests for image categorization and segmentation. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, 2008.

[151] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew W. Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose

recognition in parts from single depth images. *Commun. ACM*, 56(1):116–124, 2013.

[152] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training Region-Based Object Detectors with Online Hard Example Mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 761–769, 2016.

[153] Team Shrub. Team Shrub research group official website. `https://teamshrub.com/`. Accessed: 29-03-2018.

[154] Oriane Siméoni, Ahmet Iscen, Giorgos Tolias, Yannis S. Avrithis, and Ondrej Chum. Unsupervised deep object discovery for instance recognition. *CoRR*, abs/1709.04725, 2017.

[155] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[156] G. Warren Smith. Arctic Pharmacognosia. *Arctic*, 26(4):324–333, 1973.

[157] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pages 464–472, 2017.

[158] S.W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub., 1997.

[159] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[160] Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. DeepID3: Face Recognition with Very Deep Neural Networks. *CoRR*, abs/1502.00873, 2015.

[161] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, pages 1139–1147, 2013.

[162] Christian Szegedy, Scott E. Reed, Dumitru Erhan, and Dragomir Anguelov. Scalable, High-Quality Object Detection. *CoRR*, abs/1412.1441, 2014.

[163] Yuxing Tang, Josiah Wang, Xiaofang Wang, Boyang Gao, Emmanuel Dellandréa, Robert J. Gaizauskas, and Liming Chen. Visual and Semantic Knowledge Transfer for Large Scale Semi-supervised Object Detection. *CoRR*, abs/1801.03145, 2018.

[164] Michael W. Tao and Jitendra Malik. Sharpening Out of Focus Images using High-Frequency Transfer. *Comput. Graph. Forum*, 32(2):489–498, 2013.

[165] TensorFlow. TensorFlow. `https://www.tensorflow.org/`. Accessed: 04-04-2018.

[166] K. H. Thung and P. Raveendran. A survey of image quality measures. In *2009 International Conference for Technical Postgraduates (TECHPOS)*, pages 1–4, 2009.

[167] Emmanouil Tranos, Aura Reggiani, and Peter Nijkamp. Accessibility of cities in the digital economy. *Cities*, 30:59 – 67, 2013.

[168] M. Trentacoste, R. Mantiuk, W. Heidrich, and F. Dufrot. Unsharp Masking, Countershading and Halos: Enhancements or Artifacts? In *Proc. Eurographics*, 2012.

[169] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[170] Nanne van Noord and Eric O. Postma. Learning scale-variant and scale-invariant features for deep image classification. *Pattern Recognition*, 61:583–592, 2017.

[171] Chi-Chen Raxle Wang and Jenn-Jier James Lien. AdaBoost learning for human detection based on Histograms of Oriented Gradients. In *Computer Vision - ACCV 2007, 8th Asian Conference on Computer Vision, Tokyo, Japan, November 18-22, 2007, Proceedings, Part I*, pages 885–895, 2007.

[172] Xiu-Shen Wei, Chen-Lin Zhang, Jianxin Wu, Chunhua Shen, and Zhi-Hua Zhou. Unsupervised Object Discovery and Co-Localization by Deep Descriptor Transforming. *CoRR*, abs/1707.06397, 2017.

[173] Haibing Wu and Xiaodong Gu. Towards Dropout Training for Convolutional Neural Networks. *CoRR*, abs/1512.00242, 2015.

[174] Di Xie, Jiang Xiong, and Shiliang Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. *CoRR*, abs/1703.01827, 2017.

[175] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR*, abs/1505.00853, 2015.

[176] Ziang Yan, Jian Liang, Weishen Pan, Jin Li, and Changshui Zhang. Weakly- and Semi-Supervised Object Detection with Expectation-Maximization Algorithm. *CoRR*, abs/1702.08740, 2017.

[177] Yurong Yang, Huajun Gong, Xinhua Wang, and Peng Sun. Aerial Target Tracking Algorithm Based on Faster R-CNN Combined with Frame Differencing. *Aerospace*, 4(2), 2017.

[178] Fisher Yu and Vladlen Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *CoRR*, abs/1511.07122, 2015.

[179] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[180] Liliang Zhang, Liang Lin, Xiaodan Liang, and Kaiming He. Is Faster R-CNN Doing Well for Pedestrian Detection? In *Computer Vision - ECCV 2016 -*

14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II, pages 443–457, 2016.

[181] Shanshan Zhang, Rodrigo Benenson, Mohamed Omran, Jan Hendrik Hosang, and Bernt Schiele. How Far are We from Solving Pedestrian Detection? In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1259–1267, 2016.

[182] Liming Zhou, Compton J. Tucker, Robert K. Kaufmann, Daniel Slayback, Nikolay V. Shabanov, and Ranga B. Myneni. Variations in northern vegetation activity inferred from satellite data of vegetation index during 1981 to 1999. *Journal of Geophysical Research: Atmospheres*, 106(D17):20069–20083, 2001.

[183] Zongwei Zhou, Jae Y. Shin, Lei Zhang, Suryakanth R. Gurudu, Michael B. Gotway, and Jianming Liang. Fine-Tuning Convolutional Neural Networks for Biomedical Image Analysis: Actively and Incrementally. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4761–4772, 2017.

[184] Mu Zhu. Recall, Precision and Average Precision. `https://uwaterloo.ca/statistics-and-actuarial-science/`. Accessed: 18-03-2018.