



# GIMA

Geographical Information Management and Applications

## Identifying anthropogenic pressure on beach vegetation by means of detecting and counting footsteps on UAV images

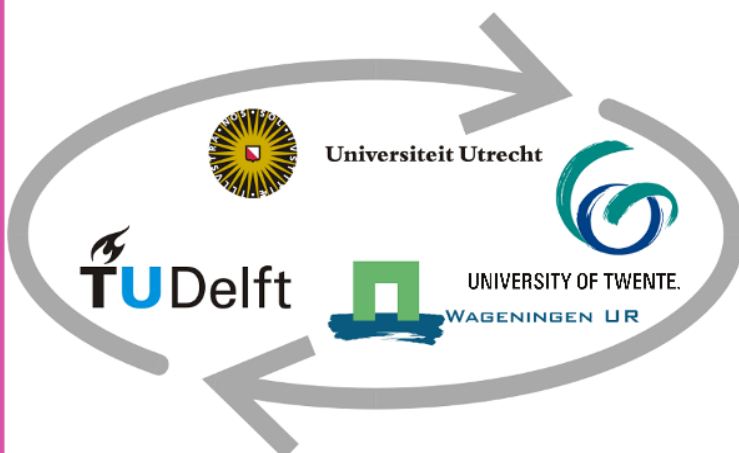
Author: **Lola Varga**

Student number: 0884537

Email: [l.varga@students.uu.nl](mailto:l.varga@students.uu.nl)

Supervisors: Sasja van Rosmalen, Prof. Dr. Lammert Kooistra

Responsible professor: Dr. Ron van Lammeren



## Contents

1.	Introduction.....	6
1.1	Overview of the topic.....	6
1.2	Purpose of this Thesis – Scope .....	6
1.3	Research Objectives .....	6
1.4	Structure of the Thesis .....	7
2.	Literature review.....	8
2.1	Artificial Intelligence.....	8
2.2	Machine Learning .....	8
2.2.1	Supervised Learning .....	9
2.2.2	Unsupervised Learning.....	10
2.3	Deep Learning.....	11
2.3.1	Major Components.....	12
2.4	Convolutional Neural Networks .....	14
2.5	Object Detection .....	16
2.5.1	One Stage – YOLO, SSD, RetinaNet .....	17
2.5.2	Two Stage – R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN .....	18
2.5.3	Comparison of algorithms .....	19
3.	Methodology .....	21
3.1	Used Software and Libraries .....	24
3.2	Data Acquisition .....	24
3.2.1	Research Area.....	24
3.2.2	Aerial Imagery .....	25
3.3	Image Pre-Processing .....	25
3.3.1	Footprint Definition.....	26
3.4	Dataset Construction.....	28
3.4.1	Annotation Process .....	29
3.5	Object Detection .....	30
3.6	Initial Run.....	32
3.6.1	Refinements .....	34
3.7	Implementation.....	36
3.7.1	Orthomosaic Creation .....	36
3.7.2	Image Tiling .....	37
3.7.3	Image Stitching.....	38
3.7.4	Georeferencing.....	39
4.	Results .....	40

4.1	Footprint Detection .....	40
4.1.1	Accuracy .....	40
4.2	Spatial Analysis .....	41
4.2.1	Temporal Changes .....	45
4.2.2	Spatial Changes .....	48
5.	Discussion .....	50
5.1	Findings and Obtained Results .....	50
5.2	Future Research and Possible Refinements .....	52
6.	Conclusion .....	54
7.	References.....	55
8.	Appendices .....	60
8.1	Appendix A .....	60
8.2	Appendix B .....	61
8.3	Appendix C.....	64

## Abstract

Unmanned Aerial Vehicles (UAV) have become popular in recent years, as these vehicles play an important role in scientific measurements and ecological researches. The thesis aimed to use UAV-based RGB images as an input for object detection, namely, to detect footprints in the sand and analyse the data in a GIS environment to spatially relate human visitation pressure to artificially planted vegetation (dune fields). Two beach sections were surveyed in 2021, Hargen aan Zee and Zandmotor, which are located in The Netherlands. A script on the Google Colaboratory platform was written to facilitate object detection using Detectron2 framework and other open-source components. The developed methodology can be reused for any relevant project involving UAV imagery and automatic object detection. The source UAV images were first merged into a georeferenced orthomosaic for later footprint detection and instance segmentation by using Mask-RCNN. An image tiling system was also developed using scripts written in Python applying the Pillow package. After annotating these tiled images to create a training dataset, a Mask-RCNN model was trained and validated. A basic training and performance optimization was performed by examining the effect of hyperparameters on the dataset used to minimize the training time and maximize the AP (average precision) value to make the model as accurate as possible. After obtaining visually and numerically satisfactory results, the masks of the detected footprints were extracted as tiles, keeping the filename structure consistent for later reassembly. It was then stitched and cropped to the resolution of the source orthomosaic. In this way, the georeferenced data of the results were preserved, allowing further spatial analysis in GIS software. The resampling was done to eliminate noise caused by JPEG compression and to convert the results into a binary raster containing boolean values. Subsequently the footprint polygons were created by vectorization, spatial analyses were performed to examine both temporal and spatial changes, including the generation of centroids and heatmaps, calculation of differences between the heatmaps to visualize temporal changes, analysis of the numbers and areas of the footprints near the dune fields and making histograms and statistics of the data. The median footprint area also confirmed the spatial integrity and accuracy of the data in the GIS system. Examining heatmaps from three time periods over one year of growing season, all showed that around the artificial dune fields there were indeed many footprints that could potentially put pressure on vegetation growth. Approximately 13-17% of the footprints in the study areas were located near and in dune fields, with no clear effect of season (summer vs. autumn). Similarly, the artificial beach entrance and beach pavilion did not appear to have an effect on footprint frequency and location. Some recommendations for future improvements in methodology and code use were also made.

**Keywords:** *object detection, detection of footprints, UAV imagery, anthropogenic pressure, image processing, image annotation, deep learning, Detectron2, Mask-RCNN, instance segmentation, spatial analysis*

## List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CUDA	Compute Undefined Device Architecture by NVIDIA
DL	Deep Learning
FPN	Feature Pyramid Networks
GDAL	Geospatial Data Abstraction Library
GIS	Geographical Information System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
LR	Learning Rate
ML	Machine Learning
MRCNN	Mask Regional Based Convolutional Neutral Network
R-CNN	Regional Based Convolutional Neural Network
ROI	Region of Interest
RTK	Real-Time Kinematic positioning
SSD	Single-Shot Multibox Detector
TIFF	Tag Image File Format
UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once

# 1. Introduction

## 1.1 Overview of the topic

Several studies have already discussed the formation of dunes, the development of vegetation and the influencing environmental factors (Van Puijenbroek et al., 2017). However, the coast plays an important role in the ecosystem for a number of reasons, namely biodiversity, recreation and coastal protection (Martínez et al., 2013). Exploring and understanding these factors is a complex task and it is especially true for the growth of vegetation on the upper part of the beach. Few studies have been done on the human use of the coast and the effects of this activity on coastal vegetation. In order to obtain accurate information about the response of vegetation to anthropogenic impact, it is essential to know exactly where the pressure is the greatest (Martínez et al., 2013). Based on images taken by Unmanned Aerial Vehicles (UAVs), correlations can be found between visitor pressure and vegetation growth. This can be accomplished by the classification of the beach in terms of visible footsteps and the presence of vegetation.

To recognise human footprints, various modern technical methods are available, and all of them fall under the concept of computer vision tools (Vassányi, 2022). In the world of information technology, computer vision is a field of science that enables computers to understand/interpret or "see" still or moving images in an automated manner at a high level that only humans could so far (Kollár & Nagy, 2021). Just like any pattern, footprints can be recognised by pattern recognition applications too.

In this thesis, an object recognition method is applied to the detection of footprints. The difficulty, however, lies in the nature of the medium itself. Since the beach is composed of sand, footprints in this highly varied material do not show a certain mathematically well-defined shape, their appearance is highly variable and thus cannot be considered uniform. It cannot be said that a footprint can only look one way in the sand, as there are many different types that are formed at different depths and in different moisture contents of the sand. The challenge lies in these conditions.

## 1.2 Purpose of this Thesis – Scope

The main objective of this study is to identify human visitation pressure on and around beach vegetation by detecting and counting footprints on UAV-based RGB images. This process requires the development of a specific pattern recognition program that is capable of detecting human footprints in sand. In addition, one of the goals is to investigate the possible relationship between the pressure of anthropogenic activity and vegetation cover in coastal environments. Since knowing the exact location of footprints is essential to achieve this goal in order to map them in a GIS software, the output of object recognition will be used for several spatial analyses.

## 1.3 Research Objectives

To achieve the main objective described above, a methodological approach needs to be carried out, in order to detect human impact in form of footsteps in sand on UAV images. What methods are needed to place the results in space in the final step? How great is the area covered by the footprints? Where are they exactly located on the different coastlines? What is the percentage distribution of the human presence in relation to the total area?

#### Objectives:

- Creating a pattern recognition program suitable for recognizing footprints in UAV images. Within this, object detection from source images as precisely as possible, using manually defined training samples.
- Knowing the location of the footprints and mapping them spatially on the beach sections. This involves some image processing steps such as creating a georeferenced orthomosaic, an image tiling and stitching system using Python scripts and vectorisation with GIS software.
- Investigating the potential correlation between human visitation pressure on different beaches through spatial analysis.

As this thesis is a supporting research of a PhD project, the knowledge of the location of human footprints is of primary importance. Further biological effects and background processes are not part of the thesis, as they will be discussed in the PhD research.

#### 1.4 Structure of the Thesis

In this thesis, the literature review (Chapter 2) is discussed in detail covering the basic concepts of artificial intelligence, machine learning and its supervised and unsupervised forms. Deep learning, convolutional networks and object detection methods are also examined. The methodology (Chapter 3) describes the general workflow with both software and programme packages used, the research area including the data acquisition itself. Data preparation, model building, object recognition and output data collection are presented. Running on the whole data set and visualizing results are covered in a separate subchapter. Model refinement is also discussed. This is followed by an overview of GIS analyses (Chapter 4). The thesis concludes with a discussion (Chapter 5), describing the experiences and observations made with a concise summary of the whole project.

## 2. Literature review

The purpose of this chapter is to present the theory behind the methodology in detail for later understanding. Firstly, artificial intelligence, machine learning and its subtypes are revealed. After that deep learning is discussed in terms of its main components, subsequently the types of convolutional neural networks. The key concepts, basic definitions and computational capabilities of neural network architecture are explored. In this way, the different components of object detection will be understood in the methodology chapter.

### 2.1 Artificial Intelligence

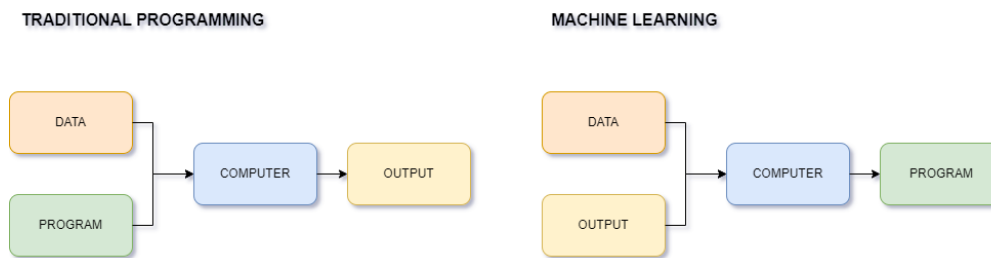
Artificial intelligence (AI) research focuses on how computers can use pre-programmed rules to solve questions that humans can answer intuitively (Rosebrock, 2017). Therefore, the goal is to replicate intelligent human behaviour in computers by exploring the processes that take place in our own intelligence and automating them. The functioning of AI depends entirely on the rules that humans feed into it, and the computer will react to situations based on these rules (Rosebrock, 2017). The mechanism aims to create a program that solves complex problems with similar efficiency as a human but in a way like a computer, this means if such a program could be perfected, the best features of human brains (intuition) and machine brains (speed, logic) could be combined, and countless tasks could be automated (Rosebrock, 2017).

Scientists have been working on the subject for more than 70 years, since the concept of AI was first developed in the 1950s, but only really took off in the 1990s with the use of personal computers. In 1997, a program called Deep Blue defeated the chess champion Garry Kasparov, after which it became the subject of public interest (Dusek, 2020). In recent years, it has also enjoyed enormous popularity in numerous scientific fields (biology, mathematics, statistics, technical sciences) (Rosebrock, 2017).

### 2.2 Machine Learning

AI can therefore perform complex tasks quickly and automatically, but only according to predefined rules (Ongsulee, 2017). A subset of this approach is the machine learning (ML), an approach that applies an algorithm to pre-processed data so that it can recognise patterns in the dataset, learn and override itself based on these patterns. It does not operate according to predefined rules, but according to data sets and patterns (Ongsulee, 2017). As shown in Figure 1, in the traditional method, the code is told exactly how to solve the problem, while in the other method a model is created that continuously "teaches" the program. The results of machine learning become more accurate as the amount of data and experience increases - just as humans become more skilled with practice (Dusek, 2020; Microsoft Azure, 2022)





*Figure 1: Traditional versus machine learning approach to a problem*

However, since around the turn of the millennium, and especially in the 2010s, machine learning solutions have become widespread. The uses range from image analysis to economic forecasting, and an industry has now emerged around it. Now software libraries are available to anyone to put machine learning into practice.

The literature divides machine learning methods into two broad subcategories: supervised and unsupervised learning (Ongsulee, 2017). There are more of them such as semi-supervised, reinforcement, but these categories are not relevant for this thesis.

### 2.2.1 Supervised Learning

In this training algorithm each piece of data is associated with one or more pieces of direct information. This means that only "labelled" data is used to tell the machine what output to expect from it in a given case. In other words, the algorithm receives both the input data and the correct outlined data. The algorithm finds errors, corrects, and learns by comparing the actual output values with the correct output data. It alters the model accordingly. Different methodologies are used such as classification, regression, prediction, and gradient boosting. This approach is usually used in situations where past data is used to predict future tasks that are likely to be performed (JavaTpoint, 2022d; Ongsulee, 2017). For example, email complaints received by customer service are automatically forwarded to the appropriate staff member who will respond to them. To do this, there are thousands of emails from the past (data) and to which employee the email should have been routed (expected target). From this training database, the model can be trained to automatically route incoming emails to the right employee with the highest possible accuracy (Farkas, 2022)

#### 2.2.1.1 Classification

The most common machine learning task is classification. In this case, there is a given set of categories, the task is to classify the individuals into one of the given classes and label them with the corresponding class label (Kollár & Nagy, 2021). An example of a classification task is the well-known cat vs. dog task, where pictures have to be classified into one of the two classes {cat, dog} (JavaTpoint, 2022d). Example of different types of machine learning classification algorithms are: Logistic Regression, K-Nearest Neighbours, Support Vector Machines, Kernel SVM, Naïve Bayes, Decision Tree Classification, Random Forest Classification (JavaTpoint, 2022d).

The model's performance and accuracy must be quantified. Since the model has to make an accurate decision on data that has not been seen before and it has to classify it into the appropriate class, the available labelled database has to be randomly split into two parts. These are training and evaluation

databases (Farkas, 2022). The reliability of the evaluation depends largely on the size of the data set being evaluated. If a relatively small, labelled dataset is available, the evaluation metrics will depend heavily on the individuals randomly selected in the training and evaluation datasets. In this case, the random training/evaluation set must be randomly partitioned several times. Finally, the evaluation metrics from different runs should be aggregated into a single number. The most known training - testing set cutting methodology is the k-fold cross validation (Farkas, 2022).

#### 2.2.1.2 Regression

However, while in the classification task the target variable is a class label, in the regression task the target variable is a continuous value, thus correlations between dependent and independent variables matter. Such continuous values are market trends or house price forecasts. The task of the algorithm is to map the input variable to the output variable via a function (Farkas, 2022; JavaTpoint, 2022c). The machine learning regression task is very similar to the regression analysis task in the field of statistics. However, while in statistics the only goal is to characterize a data set, in machine learning the goal is to predict for unknown individuals as accurately as possible, this requires generalization skills (Farkas, 2022). There are several types of machine learning regression algorithms: Simple Linear Regression, Multiple Linear Regression, Polynomial Regression, Support Vector Regression, Decision Tree Regression, Random Forest Regression (JavaTpoint, 2022c).

Since regression is also a supervised machine learning approach, the evaluation methodology is the same as for the classification approach, i.e., the labelled dataset can be decomposed into a training and an evaluation dataset. The most commonly used evaluation metric in regression exercises is the mean squared error (MSE) (Farkas, 2022).

#### 2.2.2 Unsupervised Learning

In the case of unsupervised learning, the system does not get the correct answer based on labels, the algorithm has to figure out what the data set shows. The goal then is to learn about the data by finding some structure, identifying patterns and correlations (JavaTpoint, 2022e; Ongsulee, 2017). For example, data on the company's customers' past transactions is available and the task is to find customer groups that behave similarly without pre-specifying the behaviour group (Farkas, 2022).

##### 2.2.2.1 Clustering

Clustering is the process of finding unlabelled individuals in a database within an unsupervised learning framework. Data in one cluster will be more similar to each other than to data in another cluster. Such an application could be, for example, identifying clusters (groups) of customers based on their past transactions (Farkas, 2022; JavaTpoint, 2022e).

The goal of the k-means clustering algorithm is to partition clustering of individuals in a database, i.e., the output will be k clusters, where each individual in the database belongs to exactly one cluster. The basic idea is that points in a cluster are closer to the centre of their own cluster than to the centre of any other cluster (Farkas, 2022; JavaTpoint, 2022b).

### 2.2.2.2 Dimensionality Reduction

The goal of dimensionality reduction is to provide a transformation of the feature space of the individuals in a database in which the individuals can be described in a much smaller dimensional space, but the properties of the original feature space are distorted as little as possible (Farkas, 2022; JavaTpoint, 2022a)

Since the unsupervised machine learning works with unlabelled data (i.e., no supervision), it is not possible to explicitly evaluate a possible solution (Farkas, 2022).

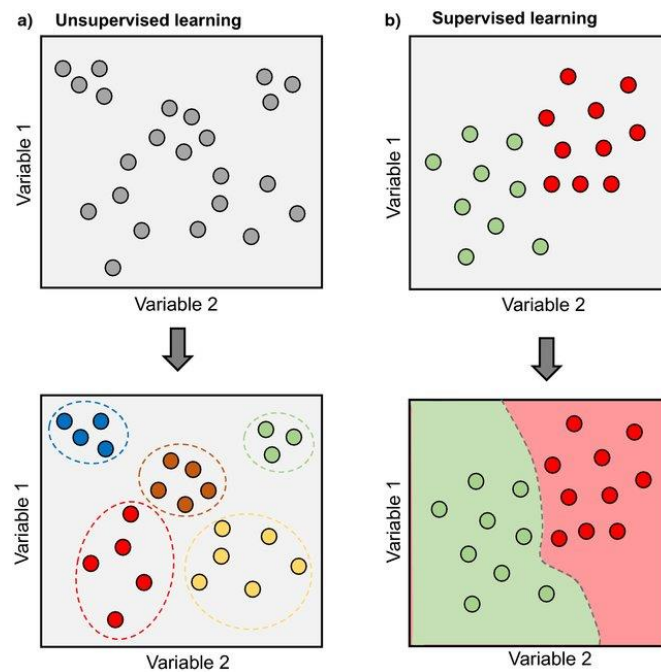


Figure 2: Segmentation approach of Supervised and Unsupervised machine learning methods (Source: Morimoto & Ponton, 2021)

Figure 2 provides a good summary of the previous chapters. A schematic representation of supervised and unsupervised learning makes it clear that unsupervised learning does not use labels for clustering and that the algorithm learns how to classify and predict the output from observations on its own, while in supervised learning, classification is done through the labelled data and the output is known to the algorithm and it is used to infer and predict the future (Morimoto & Ponton, 2021).

## 2.3 Deep Learning

Deep learning falls under artificial intelligence and machine learning (Shinde & Shah, 2018). While in machine learning it is important to give some guidance on what features or characteristics the machine should look for when distinguishing two objects, in deep learning there are no such constraints, as the machine itself has the ability to learn based on training objects. So, it does not need to define in advance what the object in the image means, but will construct it itself (Lecun et al., 2015).

Deep learning is a subcategory of machine learning, the creation of which was primarily inspired by the way the human brain works. It is a method for creating models that attempt to reconstruct the

biological activities of our neural network (Rosebrock, 2017), known as Artificial Neural Network (ANN). While machine learning algorithms rely on computer engineers to identify features, deep learning algorithms implement this on neural networks (Lecun et al., 2015).

Deep learning consists of a cascade of layers: the input layer, the output layer, and the hidden layer (Shinde & Shah, 2018). While the first two have names that speak for themselves, the last one, the hidden layer, is the layer that does the calculations. The processing and work itself is not done by the layers, but by the nodes or neurons that make them up. Each layer consists of a predefined number of nodes (Lemley et al., 2017). These are connected to the previous layer in a pattern. The whole theory was given by the biological neurons of the human body, in the same way axons get input from previous neurons. The dendrites are responsible for the information transfer, just like the connection between the input and output layers in a neural network (Figure 3) (Lemley et al., 2017). The architecture itself is hierarchical, this means the input layer learns the simpler features and the higher layer learns from the lower layers, i.e., each layer is responsible for teaching the model to recognize new features based on the output of the layer before it. The more hidden layers there are, the more subtle features and attributes the model can examine, regardless of the quality of the data being examined (Shinde & Shah, 2018).

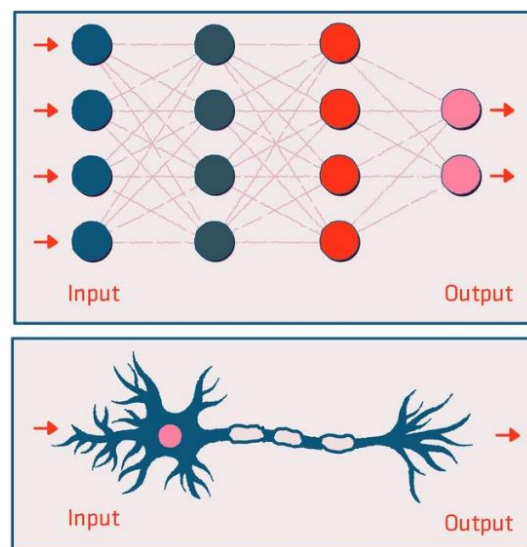


Figure 3: Comparison of artificial neural and biological networks (Source: Soffer et al., 2021)

### 2.3.1 Major Components

The main components of such a network are the input, the weights – bias, the summation functions, and the activation function.

The concept is that there are certain input values which are stored in the input layer, these are going through different weightings (multiplications, additions), then with the input weighted sums the activation function performs the calculation, and the output is obtained (Figure 4) (Rosebrock, 2017).

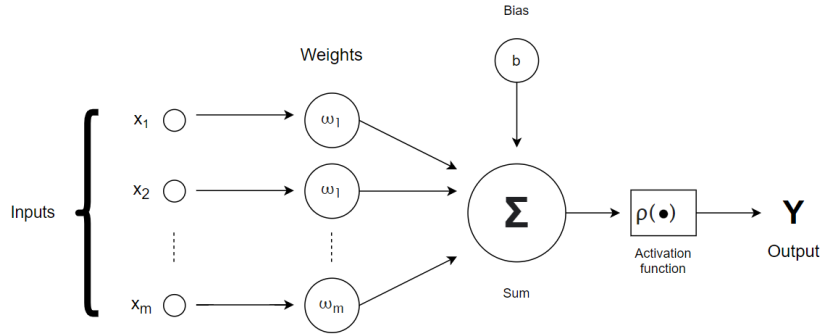


Figure 4: A simple artificial neural network that takes the sum ( $\Sigma$ ) of the weighted ( $w$ ) inputs ( $x$ ) and the bias ( $b$ ) and runs this sum through an activation function ( $\rho$ ). The function contains the threshold and thus decides whether the neuron should be activated or not (Rosebrock, 2017)

In order to see what makes neuron activation work well, the activation functions need to be introduced, which are shown in Figure 5. Traditionally, the sigmoid and tanh functions have been the most commonly used functions for training artificial networks, but these are increasingly being superseded by newer, more efficient functions. Currently, ReLU as well as its numerous variants, such as Leaky ReLU, ELU are considered the most widely used activation functions within deep learning systems (Rosebrock, 2017).

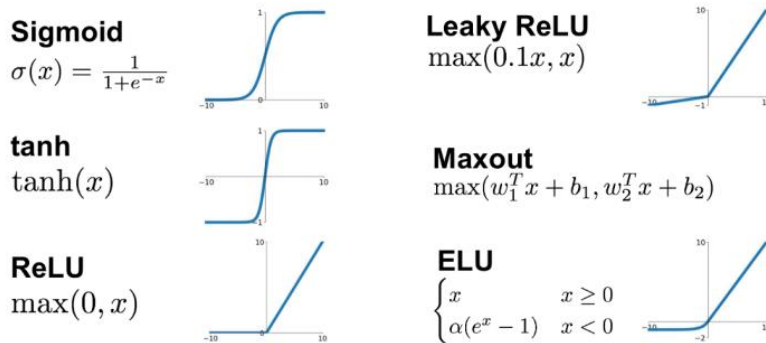


Figure 5: The most known activation functions (Source: Jayawardana & Sameera, 2021)

What happens if the model does not produce good or adequate results? And if they are not good enough, how will you know what to change, in what direction and how much to change to eliminate the error? The answer lies in optimizations and loss functions.

When teaching, randomly chosen initial weights can be used, which will not provide good results (Bushaev, 2017). The performance of the network can be improved by adjusting and fine-tuning the weights. For this, it is important to mention the concept of a loss function, which at the end of an iteration shows how accurately the network has produced the desired result. For this purpose, several functions can be used, depending on the type of the problem, such as mean-squared deviation, binary or multcategory cross-entropy (Verma, 2019). The loss function optimization is the most efficient, using the gradient of the function in terms of weights. The simplest gradient based optimization

procedure is gradient descent (Durán, 2019). The training rate determines the degree of adjustment of the weights, and therefore should be chosen carefully. If the value is too small, the learning will become slow, requiring more iterations. At the same time, if the learning rate is higher than necessary, it is not possible to get close enough to the minimum of the loss function (Figure 6) (Vassányi, 2022).

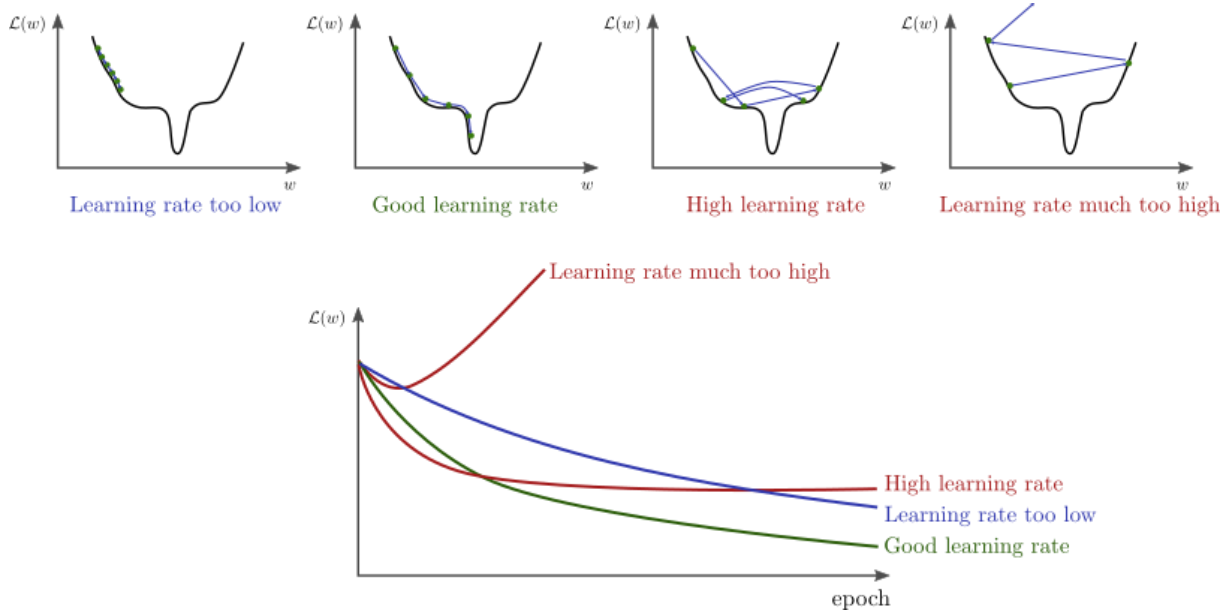


Figure 6: Optimization of the loss function for different learning rates. In the upper graphs, the black curve is the loss function; the vertical axis represents the recorded value of the function and the horizontal axis the weights. Blue lines and nodes show the approximation in iteration steps towards the minimum of the function, from left to right: for low (a), adequate (b), high (c) and very high (d) learning rates. The graph below them shows the value of the loss function as a function of iterations, from top to bottom very high, high, low and appropriate learning rates (Durán, 2019; Vassányi, 2022)

## 2.4 Convolutional Neural Networks

In the recent decades, neural networks have become widespread for solving a wide variety of problems and, as the technology has progressed, they have become more efficient at performing their task. New developments and variations have emerged over time, making convolutional neural networks one of the most popular networks of today (Durán, 2019).

These networks are very different from the others, as they have primarily image processing functions, i.e., they have pattern detection properties (edges, shapes, textures, etc.). They are also able to interpret other types of input such as videos, audios, etc. (Dusek, 2020).

A convolutional neural network consists of neurons, activation functions, loss functions and optimizers just as neural networks do. On the other hand, this cannot be said in terms of the layer scheme, because these networks use convolutional layers, pooling layers and fully connected layers (Karlsson & Rosin, 2020; Kumar, 2022; Lemley et al., 2017).

**Convolutional layer:** The network needs an input and an output layer in the same way as mentioned in the previous sections, and the operations-calculations are performed in the special hidden layers of the model performed by filters, also known as kernels (Kumar, 2022). These kernels are in fact small matrices with weight values, the size of which depends on the developer's preferences. It is advisable

to create odd side lengths to make it as easy as possible to define the centre of the matrix (3x3, 5x5, 7x7) (Dusek, 2020). The images themselves can be conceived as matrices with a defined height and width, in which case each pixel is given a value based on the colour gradient. The convolution filters move across the image pixels as a sliding window by superimposing the centre of the matrix on each pixel of the image. In each case, the image centre and the average values in its surroundings are considered, so each value is convolved separately, which means multiplying the elements of the matrix by the pixel value of that pixel gives the value of convolution (Figure 7) (Ganesh, 2019). The larger the matrix, the stronger and the more accurate the result, since the average is calculated from several results (Ganesh, 2019). By applying filters to the inputs of the outer layers, the convolution is able to detect objects on them. In the first layer, these filters are individually capable of detecting different simple features (horizontal, vertical, diagonal edges), but if the model has more convolution layers, these detectors will be able to detect more and more complex shapes (circles, corners, curves, etc.). And with a sufficient number of layers, they become capable of detecting phenomena as complex as cats' tails, car doors or stop signs - depending, of course, on the content of the input data (Deeplizard, 2017).

**Pooling layer:** The pooling layer is designed to reduce the spatial size of the input to facilitate processing and memory freeing. It also reduces the number of parameters and speeds up the training process. Two types of pooling layers are distinguished: max pooling and average pooling layers. Their name expresses their meaning, the first takes the maximum value of the feature maps, while the average pooling takes the average value. The input goes to the fully connected layer after the size reduction (Kumar, 2022; Lemley et al., 2017)

**Fully connected layer:** The final layer of any neural network is the fully connected layer, where every neuron is connected to everything else. It applies an activation function to the input (He et al., 2015; Kumar, 2022)

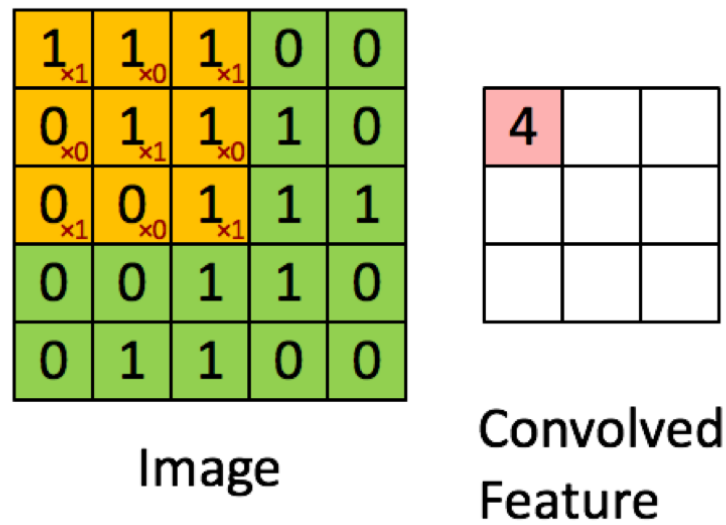


Figure 7: Arithmetic operation in the convolution layer. In this case it is the sum of the input channel and the kernel values. 9 elements will ultimately result in only one element after convolution (Ganesh, 2019)

## 2.5 Object Detection

Object recognition is a further development of image classification, since here the goal is not only to determine the object itself, but also to locate it precisely. Traditional computer vision methods have become obsolete with the advent of CNNs. Such an algorithm actually consists of two modules: the basic network (image classification methods: LeNet, AlexNet, VGG, ResNet and Feature Pyramid Networks (FPN)) and the detector network. The latter can be further categorized into single-stage and two-stage detectors and can be seen in Figure 8 (Darkhan, 2022).

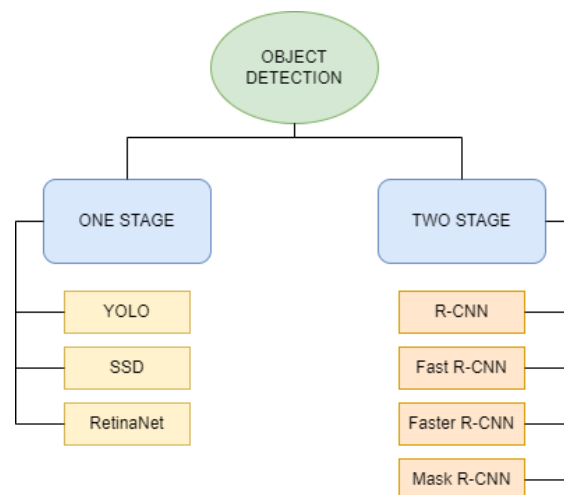


Figure 8: Object detection methods by stages



### 2.5.1 One Stage – YOLO, SSD, RetinaNet

Such single-step detectors do not include a Region of Interest (RoI) selection process but classify directly. This category includes the YOLO family (YOLOv2, YOLOv3, YOLOv4, and YOLOv5) the CenterNet family, RetinaNet and many others (Darkhan, 2022).

**YOLO (You Only Look Once):** Yolo is a neural network that, like human perception, looks at a given region of the image and decides whether there is an object there. If it finds multiple overlapping objects in the regions, it summarises them and selects/highlights the individually found object (Figure 9). According to the research of Rizzoli (2022), the GPU processed 45 frames per second, which is considered extremely fast, however, due to its speed, its accuracy rate is reduced compared to other approaches that will be discussed later on. On the other hand, in further developments (Yolo v2, Yolo v3), researchers have already made significant progress in accuracy.

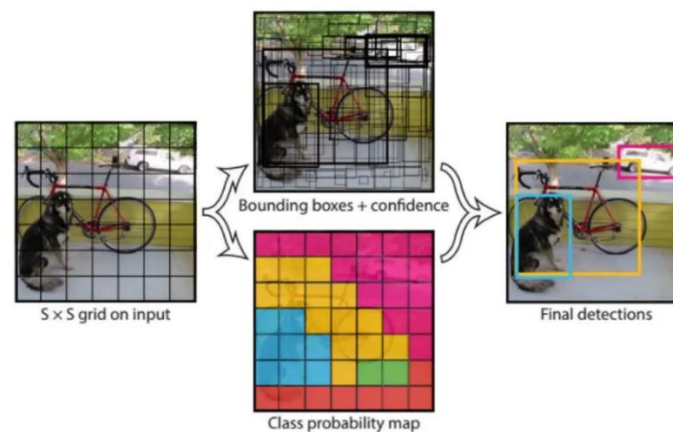


Figure 9: The workflow of YOLO models, consisting of input gridding, creation of bounding boxes, class probability maps and confidence values (Source: Chablani, 2017)

**SSD (Single-Shot MultiBox Detector):** SSD model works with fixed-size object frames (anchor boxes), and then for each predicated frame, it adds a value for the probability of the object class prefetching. Overlapping frames are intersected by the model. Its operation is based on a simple convolutional network, to which additional layers are added. It is important to know that the size of these layers will be much smaller than the size of the base network, which will result in a faster runtime (Figure 10). These attached layers perform the detection. According to the documentation, they are extremely accurate (Gróf, 2018) (Rizzoli, 2022).

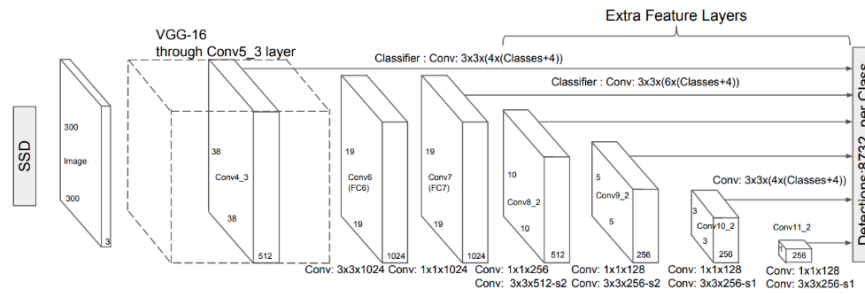


Figure 10: SSD architecture (Source: Khandelwal, 2019)

## 2.5.2 Two Stage – R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN

Two stage detection breaks object detection into two parts: Region of interest selection and then classification and regression.

**R-CNN:** There are several types of convolutional networks, but not all of them are effective for detecting objects (Vassányi, 2022). In recent years, there has been a growing emphasis on the region based approach, i.e., the R-CNN (Region Based Convolutional Neural Network) (Girshick et al., 2013). This is a network that tries to fit lots of rectangles to an image and then examines them to see where objects may appear. It generates the locations of the quadrilaterals based on the properties of the image (pixels) and setting up roughly 2000 recommended regions. Then it attempts to classify each region and sends them to a CNN (Figure 11) (Girshick et al., 2013). Using R-CNN, the CNN feature extraction has to be run for every two thousand regions in each image, thus the process can be considered slow, therefore, the method remains irrelevant in the field of object detection (Darkhan, 2022).

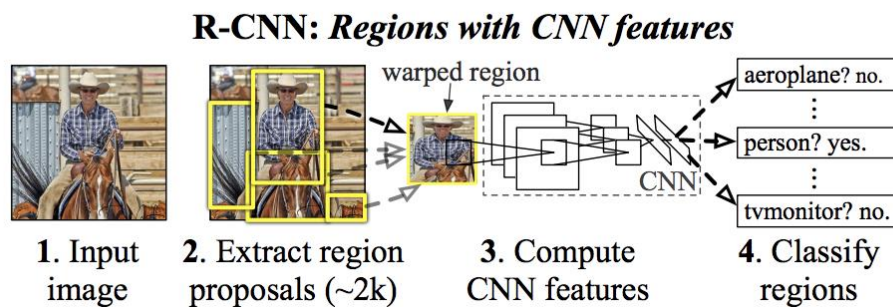


Figure 11: R-CNN object detection system overview. The system works with an input image (1), selects 2000 regions on the input (2), computes features for each region using a convolution network (3), classifies objects (4) (Source: Gandhi, 2018)

**Fast R-CNN:** The Fast R-CNN is an extension of the R-CNN discussed above. The concept of Region of Interest (RoI) is also introduced. RoI divides the image into a fixed size grid applying average pooling (Darkhan, 2022). While in R-CNN each step is performed by a separate model, Fast-R-CNN combines all the elements (RoI calculation, convolution steps, classification) into a single model (Darkhan, 2022). So, in this case, the input image is fed into the CNN to produce a convolutional feature map. From this, regions can now be identified, squares can be drawn and then converted to a fixed size using the RoI

pooling layer (Figure 12). This solution significantly shortens the learning and testing time compared to a plain R-CNN, but the process is still slower than desired (Gandhi, 2018).

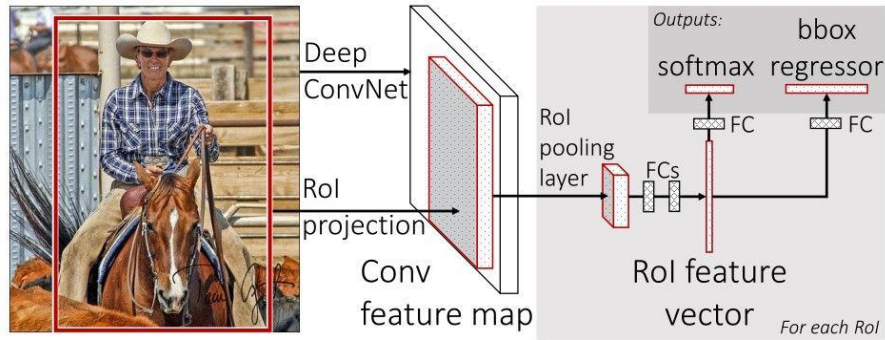


Figure 12: Fast R-CNN architecture (Source: Gandhi, 2018)

**Faster R-CNN:** Fast R-CNN also has a more advanced version called Faster R-CNN, where the recommendation of regions is also done by a neural network (Region Proposal Network - RPN). So, it consists of two modules: the first one is the deep neural network for region recommendation and the second one is Fast R-CNN, which uses the recommended regions. The Faster R-CNN allows detection in near real time and is currently one of the most efficient methods (Ren et al., 2015).

**Mask-R-CNN:** There are several extensions of the Faster R-CNN, of which Mask-RCNN (MRCNN) is one of the most popular, which can perform instance segmentation (detecting and localizing the object in an image). This involves, in addition to object detection, the specification of a particular pixel-level extent of object instances, which is generated by the procedure in the form of a mask. The creation of the mask is performed by a fully connected neural network, which generates the mask itself by binary classification on the pixels of the RoI and then assigns the class proposed by the classification algorithm (Figure 13). Pixel-level segmentation requires higher accuracy for RoIs, so the model also includes a layer called RoIAlign to eliminate the slippage of the position of the inclusion rectangles (He et al., 2018; Vassányi, 2022).

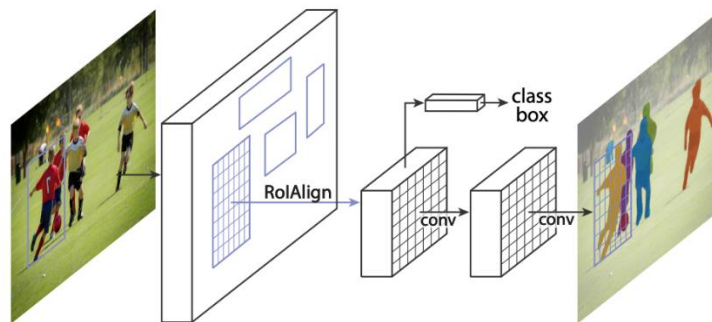


Figure 13: Mask R-CNN architectures (Source: He et al., 2018)

### 2.5.3 Comparison of algorithms

A number of CNN based detection algorithms from both One Stage and Two Stage methods have been presented in the previous sections. It cannot be clearly stated that one is better than the other as

different initiatives may be correct for different applications (Srivastava et al., 2021). In addition to the fact that one type has a different running time and accuracy from the other, care must be taken when making a decision since not only the type of detector is important, but also other factors that influence the choice:

“Feature extractors (VGG16, ResNet, Inception, MobileNet); Output strides for the extractor; Input image resolutions; Matching strategy and IoU threshold (how predictions are excluded in calculating loss); Non-max suppression IoU threshold; Hard example mining ratio (positive v.s. negative anchor ratio); The number of proposals or predictions; Boundary box encoding; Data augmentation; Training dataset; Use of multi-scale images in training or testing (with cropping); Which feature map layer(s) for object detection; Localization loss function; Deep learning software platform used; Training configurations including batch size, input image resize, learning rate, and learning rate decay” (Hui, 2018).

Many articles discuss these methods by comparing algorithms on the same datasets. It can be said that in general, if accuracy is the issue, Faster R-CNN is a better choice, on the other hand SSD and YOLO beat it in speed (Srivastava et al., 2021). In terms of the feature extractor, Faster R-CNN is the winner again. For large objects, the SSD is the best model on the other hand, for small objects it is the least suitable. Of course, as the resolution deteriorates, they all produce worse results (Hui, 2018).

The reason why they are listed is that in this thesis all of what has been described so far is used in a summary framework called Detectron2. This is discussed in more detail in the Methodology section, under the chapter on software and library packages used.

### 3. Methodology

After reviewing the basic concepts necessary for a complete understanding of the detection process, the practical part of the thesis is explained, namely the description of the construction process of the footprint detection system. To construct a footprint detection system and fulfil research objective 1, it was necessary to build from the basics, so after the fundamental concepts detailed above, the knowledge acquired had to be put into practice step by step.

In this thesis, the artificial neural networks, convolutional networks and their associated methodologies described in the literature chapter were applied. The following strategy, which can be seen in the flowchart too (Figure 14), was used. It contains four main parts: image pre-processing, dataset construction, object detection and spatial analysis. The image processing (tiling and stitching) required for the recognition process to produce usable results serves research objective 2. As for objective 3, different spatial analyses will be carried out to analyse the data and to investigate the human visitation pressure on the research area.

The acquired data, i.e., the images taken with the drone, were cropped in the Global Mapper software into smaller parts for easier processing, and then the set of images best suited for training and validation were selected. A significant aspect is that such a deep learning process can be separated into learning and validation processes (Kumar, 2013). The training of a neural network requires a large amount of annotated training data, which is a serious limitation for the use of deep learning methods (Vassányi, 2022). The more objects, in this case annotated polygons, are fed to it, the more likely it is to learn about that particular object with greater accuracy (Shah, 2017). However, it is more important that each annotated polygon expresses that the object outlined is a footprint. Footprints were annotated in the open source VGG Annotator, during which the marked objects were saved in a JSON format. The annotated images were divided into training and validation images with a ratio of 70:30 or 80:20 (Kumar, 2013). The finished JSON files had to be converted into a format suitable for detection, by the Detectron2 package. Detectron2 includes all the theories presented so far, such as the image segmentation (semantic, instance and panoptic segmentation) and object detection methods (Fast R-CNN, Faster R-CNN, Mask R-CNN, RetinaNet, RPN, R-FC, DensePose, Cascade R-CNN) presented in previous chapters (Chapter 2) (Girshick et al., 2019). The process uses an open-source object detection procedure, which is available to everyone on GitHub. Detectron2 provides pre-built models for object detection, instance segmentation, person key point detection (Adamczyk, 2020). It requires a CUDA-enabled GPU to run – it currently does not support CPU computation. The library in GitHub contains the pre-learning of the network on the MS COCO dataset. The detection program was run in a virtual environment using the services of the Google Colaboratory (Colab). The program code is available in a Notebook format, allowing interactivity and the visualisation of the results. Google Colab was chosen because it offers a suitable GPU on which Detectron2 can be run. Another aspect is speed – due to Colab's powerful GPU, it runs much faster than on a home computer (Radečić, 2020). The Detectron2 framework was trained with selected objects in the input images. Training on a custom dataset involves fine-tuning the training parameters (learning rate, number of iterations, batch size, etc.) to achieve the desired accuracy for reliable and accurate footprint detection. Once the deep

learning model was applied to all input images, refined, and run, and the observations were accurate and reliable, the next step was to export the results, which could be used for further analysis with a GIS software. Detectron2 offers the detection results (in other words, the model output) in a JSON-like dictionary format, in which there are various fields corresponding to the detection. This information can be read and extracted with simple scripts. In terms of the thesis, the most important piece of data in this output was the detected instances' masks, in form of tensors (multidimensional arrays).

Another thread of the workflow gives the final results detected by the trained model during the already mentioned progress. The test dataset itself contains the images of the flights that took place at the given time. These images first had to be merged into one image containing the whole or part of the research area at that particular time of flight. This was implemented in the form of an orthomosaic in the Agisoft Metashape software. The idea was to get a complete overview of the area that could be divided into structured pieces. A short code was used to automatically slice up the orthomosaic in PyCharm, so that these small, tiled images formed the test dataset itself. After running the program on the Colab Notebook interface and downloading the results from the test dataset, these image fragments (with structured image labels) could be stitched together with another script based on their names. As the source image was automatically georeferenced when the orthomosaic was created, the stitched image containing the detected footprints has the same final resolution and extent, thus the resulting image can be considered georeferenced. The pre-processed and detected image can be used in any GIS software for various spatial analyses, providing visual and unambiguous results on area, distribution and vegetation data in such a way that the data can be loaded as features, and further analyses can produce a heat map showing the distribution of the footprint manifold.

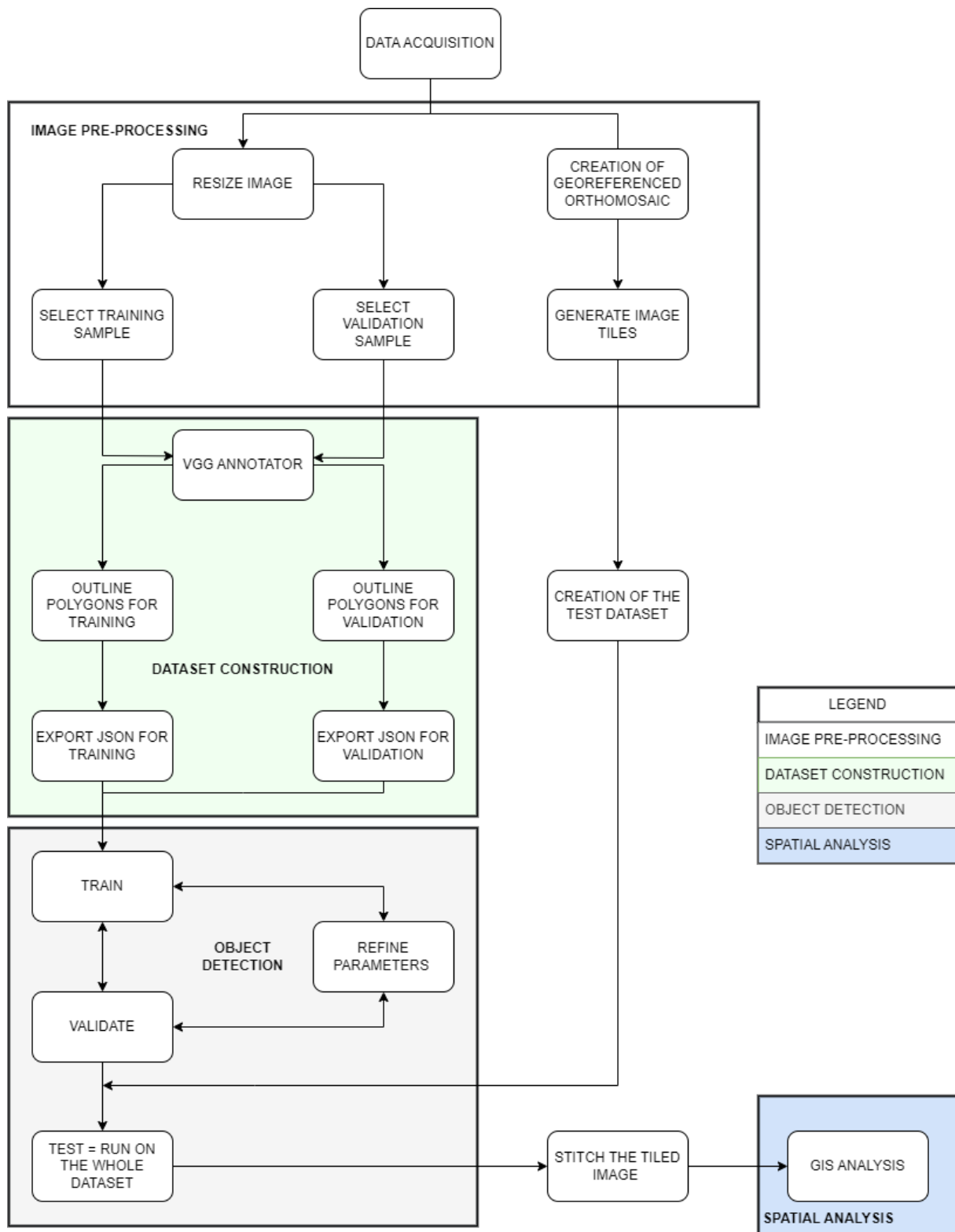


Figure 14: Flowchart of the process, divided into 4 major parts: Image Pre-Processing (white), Dataset Construction (green), Object Detection (grey), Spatial Analysis (blue)

### 3.1 Used Software and Libraries

One of the great benefits of the current state of information technology is that it is easy and simple for everyone to access tools that carry a huge number of resources. In just two decades, almost all computer hardware components have seen an order of magnitude jump in computing power. There are already such powerful CPUs, GPUs, RAM, and storage space that one can comfortably do such a project even on a laptop. I used my own Acer Aspire laptop to complete the task. The specifications of the machine can be seen in the table below (Table 1). Every step of the workflow was performed on this computer except for the main object detection that runs on Google Colab's computers.

*Table 1: Laptop's Hardware Specifications*

#### **Laptop's Hardware Specifications**

Processor	Intel Core i5 - 10210 CPU
Memory	8 GB
SSD	512 GB
Graphics Card	NVIDIA GeForce MX250

To perform all the relevant steps outlined in Figure 14, a number of software and libraries were used, including PyCharm, Agisoft Metashape, Google Colaboratory platform, VGG annotator and Detectron2. Other libraries, dependencies and components associated with Detectron2 include PyYAML, CUDA, Torch, Torchvision, NumPy, OpenCV. The core programming language was Python (both locally and on the Colab platform). Further description of each component can be found in Appendix B.

### 3.2 Data Acquisition

Aerial images were taken to obtain the data that allowed the process to be carried out. The images captured were used as part of the data collection for an experiment on the effects of recreation on embryonic dune development.

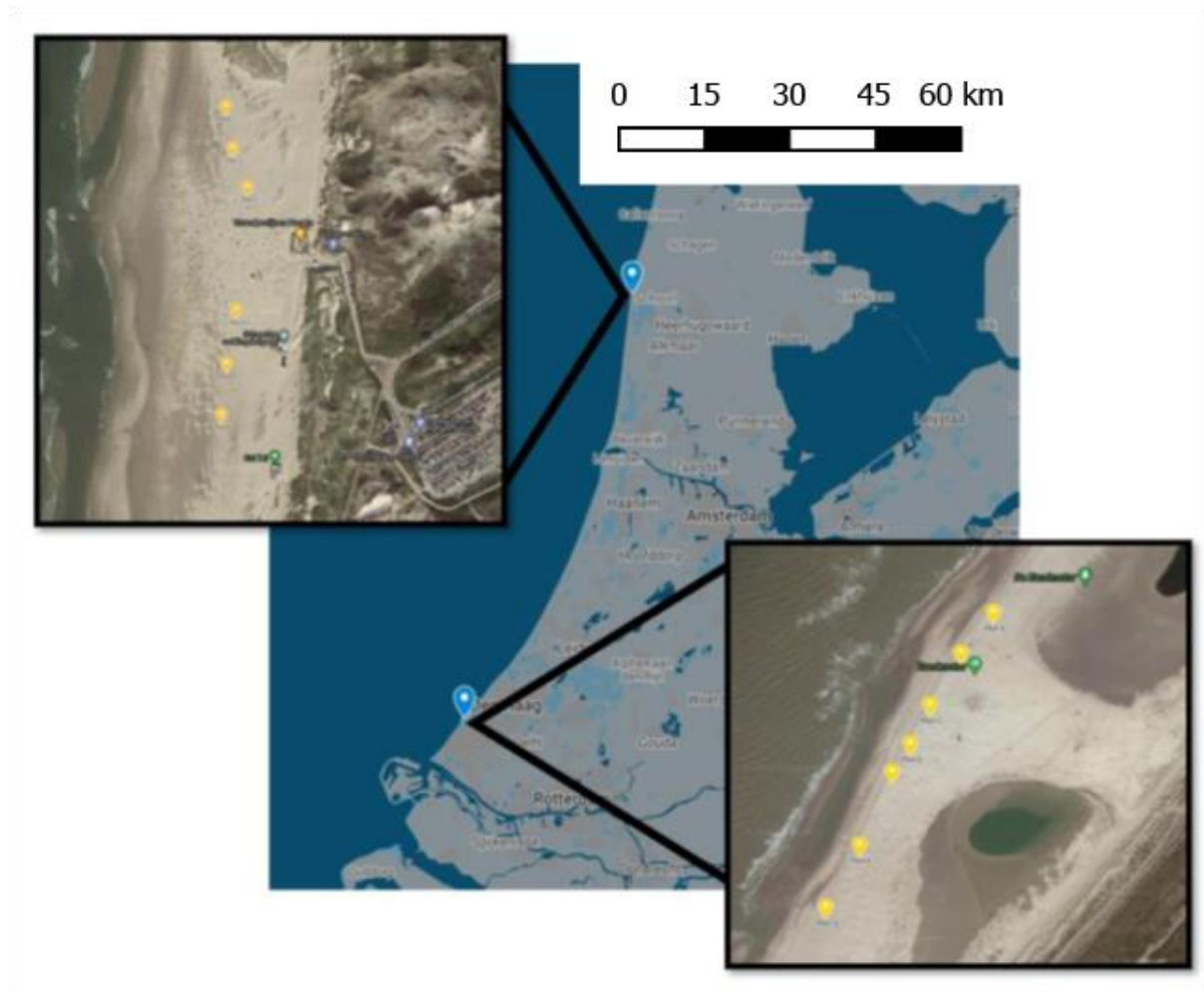
#### 3.2.1 Research Area

For this thesis, the data was collected from two locations in the Netherlands, Hargen aan Zee and the Zandmotor (Figure 15). The former is a relatively well-visited beach with a number of facilities (restaurants, storage booths, lifeguard station). This beach section is located at the southern point of the Hondsbosche Duinen nature coastal project. Six artificial dune fields, evenly spaced at 60-m intervals, were constructed in 2021.

The Zandmotor (Sand Engine) was created in 2011 to improve flood safety on the surrounding coastline and provide a natural sediment recharge for the next 25 years, while the artificial dune fields were created in 2020. At this location, 7 artificial dune fields were constructed in 2020.



The data from both locations was processed as part of the training data to increase the diversity of training patterns from the two datasets, potentially making the upcoming footprint detection more accurate. Due to time constraints of the thesis, the decision was made for the spatial analyses to focus only on the location Hargen aan Zee. The reason for this is the higher visitor frequency and a clearer set-up in the field experiment at this location.



*Figure 15: Dune field locations in the Netherlands, top insert Hargen aan Zee, bottom insert Zandmotor. Data captured at the Zandmotor was only used additionally to the data from Hargen aan Zee for model training purposes to increase footprint sample variety and overall precision. The Hargen aan Zee location was used of the spital analysis. The yellow markers show the artificial dune field locations*

### 3.2.2 Aerial Imagery

The aerial photos were taken with a DJI Mavic pro 2 drone. During automatic flight, the images were taken at 30 m from the ground surface at a 90° angle with 80% overlap. To facilitate georeferencing, 5 ground control points were placed (their exact location was also verified by RTK-GNSS). The images were taken monthly during 7 flights at both locations between April and October 2021.

### 3.3 Image Pre-Processing

The first step was the data pre-processing. As the input data was drone images, the physical size of these images is always enormous due to their high resolution. As a result of this fact, they must be sliced into smaller pieces, in this way specific parts of the image could be worked on. This improves

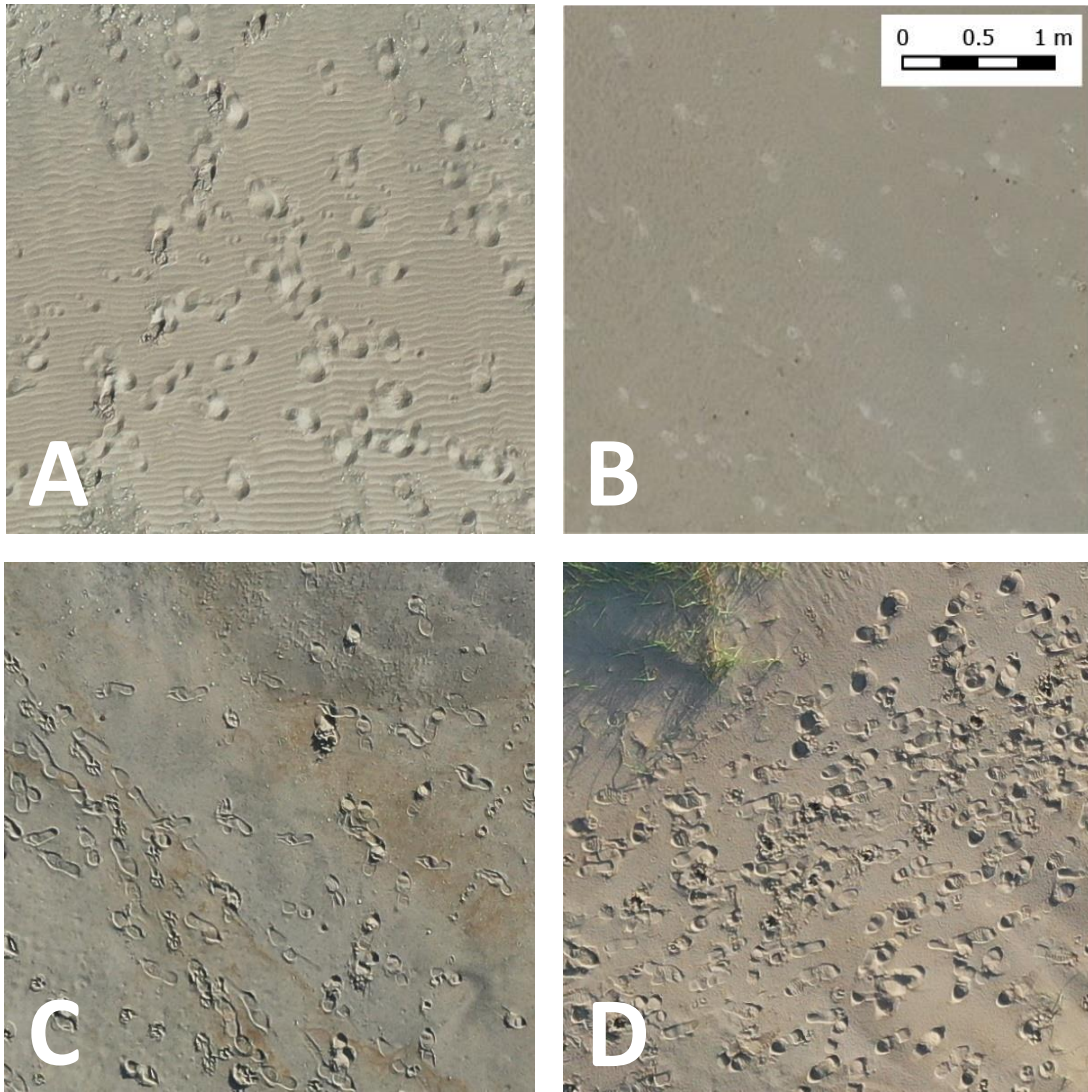
annotation, and neural networks work better and give more accurate results on images that focus on a specific part of the image to find specific objects, thus cropping the images was essential. Global Mapper was used as GIS software for this purpose because it supports raster tiling with pre-defined sizes. The output size for the individual tiles was chosen as 512 x 512 pixels. This size was chosen because it appeared to be optimal from all aspects; if the image was divided into too large parts, the meaning of the cropping would be lost; if it was divided into too small parts, it would result in numerous small images in which the surrounding area of the prints is not visible and hence irrelevant for detection. The images were cropped from the original resolution of 5472 x 3648 to roughly 10 columns and 7 rows with a pixel size of 512 x 512. Original size of the JPG images: 6 – 11 MB; Cropped 512 x 512 pixel size images: 150 – 170 kB.

Following the image cropping, the next step was to select which images could be annotated. The criterion was, of course, the quality of the images and the clear outlining of the footprints. The better the quality and the more accurate and clearer the footprints, the easier to annotate the images.

### 3.3.1 Footprint Definition

The aim of this thesis is to detect footprints in the two specific study areas. The fact that this must be done in sand emphasizes how difficult the task is, as the sand is constantly changing. As stated in the literature review, neural networks perform exceptionally well when fed precise, specific, and well-defined data. However, in this case, this cannot be said to be feasible. Sand as a fine-grained material is non-stop malleable and changeable. A footprint looks different every time in it, which is clearly visible in Figure 16. Needless to say, it depends on the moisture content, grain size, depth, but it can be said that there is no singular pattern. It cannot be stated a footprint in sand is X wide and Y long or elliptical because it will look different in every weather condition.

In order to train the machine to recognise what it sees in the input data, i.e., the images taken by the drone, it has to be trained what to see. This requires a well-defined footprint definition. What shape do we expect the machine to take as a footprint that it can recognize? For this reason, the footprint definition was divided into parts.



*Figure 16: Sample data of different types of footprints: A - soft, dry sand, unclear footprint contours, B - hard, wet sand, superficial faint footprints, C - wet and deeper area, clear footprints, D - drier sand with distinct contours, overlapping footprints*

Shape: Since the disciplines have not yet addressed the detection of footprints in sand or similar variable environment for object recognition, an exact formula describing this specific pattern is not available. To make the program uniformly tractable, a specific framework was needed. An initiative to define a shape might be, for example, an ellipse. However, this shape only remotely approximates the dimensions of a footprint. The shape of a peanut, on the other hand, is more descriptive of what a footprint actually is. In this thesis, however, the number of footprints with the ideal shape (similar to the shape of a peanut) is very small due to the aforementioned sand as a medium, and therefore the circle was chosen as the shape of a footprint. Since the sand around the vegetation is much deeper and drier, a footprint is drawn as a circle in these places. However, it is not only the perfectly regular circular shape that can be captured as a footprint, but any shape that approximates a circle, which is visible in Figure 17.

It should be noted that the image does not only show footprints with shoes, but also barefoot prints, dog footprints, and occasionally horse and car patterns. These were excluded in this thesis.



*Figure 17: Samples of circular footprints*

**Contour:** Determining the contour of a footprint is both of paramount importance and difficulty. Undoubtedly, the difficulty is the consistency of the sand. In wetter sand, the whole outline of a footprint can be clearly seen, whereas in drier sand these patterns are often lost or only half-drawn. The other problem to solve is the depth of the sand, in drier and deeper sand the outlines are wider and the prints appear as small mounds on a wider base because it does not retain the detail. In addition, shadows can also alter the recognisability of the outline, because if the contour of the traces is higher, for example, because of foot-pushed sand, the sunlight can cast shadows on parts of the trace, making it harder to tell which part the trace is.

**Size:** Since footprints of different sizes can be detected in the images (partly because of the different sizes of people's feet and the types of footprints), it is also important to look at the size - how large an object actually being searched for in the sand is. It is clear that since the primary focus on beaches is to find human footprints, only footprints that are approximately 25-30 cm long matter (or less, if the shape is regarded as a circle). In addition, there are also patterns of different sizes (vehicle tracks) in the images, but their pattern and extent are completely different from the footprints that are the focus of the task, so they are not a priority in this research.

**Colour:** The particular colour of the sand means that no particular delimitation is needed here, as the sand ranges from a relatively dry, lighter whitish colour to a brownish reddish colour. However, it is worth noting that there are patterns of different shades of colour due to the different moisture content of the sand. Naturally, in the wetter harder areas the sand colour is also darker than in the drier deeper areas. The colour shade of the sand under the traces is similar to the colour shade of the intact sand, except that the saturation and the grain size are different (this may be due to the fact that the moisture content of the compacted sand under the traces is slightly different from the intact sand).

Based on these criteria, the footprints are considered to be slightly elongated, circular-elliptical shape, in this study about 25-30 cm long. Colour was not strictly defined as it is highly variable. In fact, the very first approach was to plot perfect footprints, but this type of pattern is so rarely encountered due to the variability of the sand that it made no sense to teach the machine to do something that is barely detectable in the area under research.

### 3.4 Dataset Construction

As already mentioned in the principles of machine learning, the data must be split into three parts. In this case, this ratio was 70:30, which was randomly selected as is common practice. It is important that there is no overlap between the two sets, i.e. that they do not share common data. In the same way,

the third set, the test set, should only include images whose subsequent values are not included in either the training or the validation data set (Kumar, 2013). 20 images were collected to the training dataset, 6 images for the validation and the test dataset is the result itself.

#### 3.4.1 Annotation Process

It can be said that a computer will not be able to think by itself, so it does everything that is expected to be done by following a series of instructions. That is why when an untrained device looks at a picture of a dog, it does not know what it is, so the machine has to learn what a dog is and all the different breeds in the world.

“Image annotation is a subset of data tagging, also known as image tagging, transcription, or labelling of a name, that image annotation involves humans in the background, tirelessly tagging images with metadata and attributes that help machines better identify objects” - (Shaip, 2022).

Annotation techniques include rectangle, circle, ellipse, polygon, point and polyline (Dutta & Zisserman, 2019). The rectangle is one of the most commonly used techniques as it is actually a bounding box around the object. These boxes provide information about the length and width of the object and occasionally its depth (Shaip, 2022). A polygon can also be a complex shape, with irregular or random shapes, this means that the goal is to manually draw lines around the perimeter of the object. Points are used to define feature points, they are mostly used in MRI images (Shaip, 2022), while circle, ellipse and polyline are the least used methods (Dutta & Zisserman, 2019).

Annotatable images can be 2D, i.e., data from cameras or optical microscopes, or 3D, i.e., data from cameras, electron, ion or scanning probe microscopes (Shaip, 2022).

Types of image annotation (CloudFactory, 2022):

- Image Classification: An unknown image should be classified into a formula of predefined categories, but only at a rudimentary level, e.g., a dog or a cat is in the image.
- Object Detection: The task is to identify certain types and number of objects in images.
- Image Segmentation: To bring additional information to the object, so that the objects are more precisely defined. There are 3 types of segmentation: Semantic segmentation; Instance segmentation; Panoptic segmentation. The common essence is that the object is localized, shape and dimension are examined, and pixels of the searched elements are filtered out by colour value subtraction.

Since the aim is to identify the presence, location, number, size and shape of all footprints in the images, annotations were done to ensure instance segmentation.

The workflow of the annotation is the following (Dutta & Zisserman, 2019):

1. Load images: All images to be annotated must be loaded. To do this, selection of files from the local machine is needed. It also allows to load files with URL or full path.
2. Draw regions: To draw a circle around the regions with the mouse, the appropriate shape needs to be selected (rectangle, circle, ellipse, polygon, point, polyline).



3. Create Annotations: Here attributes can be added (name, type, image\_quality).
4. Export Annotations: Exported possible file formats are: json or csv.
5. Save Projects: The project can be saved with the path to the images.

The footprint annotation followed the Dutta & Zisserman (2019) workflow, with the identification of individual polygons in the images. Annotating the images is a relatively lengthy process, with about 7 seconds per footprint. This is compounded by the decision-making whether that particular pattern is a footprint or not, so it adds up to 10 seconds per footprint. This is still a small amount of time compared to manually determining the number of footprints for each image one by one, especially when thousands of images are involved.

When exporting as JSON, VGG Annotator outputs a structured JSON object with all the annotations and their corresponding attributes. The basic structure of an exported JSON file is the following:

The first layer is an array of the annotated source images. Each of these objects have a “filename” string, a “size” integer attribute and an array object called “regions”. Filename references the annotated source images’ filename in which the polygons were marked (for example, DJI\_0304\_B\_07.jpg), size contains the images’ file size in bytes (47948). The „regions” array contains all the information about the marked polygons, the annotations themselves (if 15 polygons were marked, 15 objects are included in this “regions” array). Inside a region, there are shape attributes, which have a name that indicates their type (polygon), and two arrays of matching object count, “all\_points\_x” and „all\_points\_y”. These contain the image-relative coordinates in pixels of all the nodes that build up a polygon. For example, if a polygon was drawn using 15 nodes on its outline, these arrays will both have 15 objects inside them (since each node has an X and a Y coordinate value). The JSON structure is attached in Appendix A.

The annotation was done for both datasets, i.e., both for the training and validation datasets, after which the annotations were stored in a JSON file for each dataset. This way, both the training and validation data (images and JSON file) were separated into two folders. Since Detectron2 natively supports a COCO dataset and VGG Annotator outputs a different JSON structure, a custom function was needed to parse this data and prepare it in Detectron2's standard format.

### 3.5 Object Detection

The application is accessible via the Google Colaboratory interface, and the calculations were thus performed on that computer. First, fixed versions of Detectron and Pyyaml packages were installed. Specifying the version of any package or library is critical to ensure compatibility and functionality. Any new version of a library that is unknowingly loaded and installed can alter its functionality in a way that may cause incompatibility. The version numbers used are given in Appendix B.

The whole program requires the data to be loaded in the correct format, a training loop that goes through over the dataset by batches, a model that can be trained, an evaluator that tells how to correct the model and test its performance after each run and finally, a logger (Methodology section imported dependencies) that always saves the current results.

Data can be loaded from own machine or downloaded from a server. The main thing is that it has to be in .zip format and the program itself will unpack the data. The structure of the dataset should look as already described.

Detectron2 has two large global dictionaries, DatasetCatalog and MetadataCatalog. The former is used to manage, load and store data in specific formats (masks, bitmaps, polygons). The latter, as its name suggests, deals with metadata, i.e., it can store the names, labels and identifiers of data. These were used in a for loop by registering a dataset with the DatasetCatalog function, which has several parameters such as the name of the dataset, the data to be loaded (Cankaya, 2022).

Since the format of the JSON exported from VGG Annotator was not ready to be registered as a DatasetCatalog yet, it was needed to read this file and save it in a format which was compatible with Detectron2's datasets. In order to convert the annotations, a function was used, which, through many loops, reads the structured data and builds an array with the annotation data, which is compatible with Detectron2. This function first loads the JSON annotation itself and creates an empty array to collect the data of each polygon object per image. This array contains one record for each image, and at a deeper level, each of these records contains one object describing the polygons themselves. This requires two main loops to read and collect the data, one for the image level and one for the polygons of that specific image. The polygons were defined by their X and Y coordinates, as described in Section Annotation Process. After all the images and their polygons were processed, the function returned an array with all the annotation data, now in a proper format that Detectron2 could actually understand and register as a DatasetCatalog.

Before proceeding with the training of the model, it is worth checking that the dataset was loaded correctly. A short code is run to visualize the annotations on a random sample of images to make sure everything is placed correctly. Once registered, the dataset is relatively easy to access.

Then comes the learning phase itself.

To start training on the dataset, a "trainer" has to be initialized first, with an appropriate configuration. Detectron2 offers two trainers: SimpleTrainer and DefaultTrainer. According to the documentation, SimpleTrainer is a minimal training loop, with no logging (Detectron2 documentation, 2023). The DefaultTrainer was chosen, since it offers more standard behaviours including default set-up for logging and other functions.

This trainer element requires hyperparameters, based on which it actually runs and trains. A basic configuration for Mask-R-CNN is loaded (mask\_rcnn\_R\_50\_FPN\_3x.yaml), while other parameters are set later. The most important parameters in a configuration are (as a domain of "cfg") (Detectron2 Documentation, 2023):

- `cfg.DATASETS.TRAIN` and `cfg.DATASETS.TEST`: Sets the previously registered datasets by their name
- `cfg.SOLVER.IMS_PER_BATCH`: The number of training images the machine (GPU) sees in each step (iteration), commonly known as "batch size"

- `cfg.SOLVER.BASE_LR`: Learning rate
- `cfg.SOLVER.MAX_ITER`: The number of iterations when the training is instructed to stop
- `cfg.MODEL.ROI_HEAD.BATCH_SIZE_PER_IMAGE`: The number of RoIs per image
- `cfg.MODEL.ROI_HEADS.NUM_CLASSES`: The number of distinct, labelled classes in the dataset.

After applying the configuration to the trainer (`trainer = DefaultTrainer(cfg)`), the `train()` function is called, which starts the training itself.

When the iteration threshold is reached, training is complete. The result of the training is a model that can be used for inference, practically any perception.

In order to actually use the model to infer any image, a `DefaultPredictor` object must be initialized in the same way as `DefaultTrainer`. This object requires the same configuration (`cfg`) as the `DefaultTrainer` needed, including "`cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST`", which specifies the score threshold in percentage above which the detected results are considered valid and saved as results. Once the configuration has been applied to the predictor (`predictor = DefaultPredictor(cfg)`), the `predictor(img)` function can be called, whose only parameter is the path to the image. This is convenient because it allows the predictor to be used on a batch of images in a single loop.

The detection output that Detectron2 provides is in a JSON-like format, so the elements are structured logically. The most important field in this structure is the `pred_masks` field, which contains the actual results (the detected polygons' geometry on a specific image). This can be extracted as a multidimensional array: an 512x512 sized, boolean-type array for each detected polygon (for example, an array of shape (18,512,512) means 18 detected results, packed in one single array). The 512x512 size corresponds to the image resolution, and since this has boolean values (True or False), it has True values on all cells in the area which the footprint covers. This multidimensional array is saved as numpy array for every image the detection ran on. The saved files can be transferred from the Google Colab computer to a local machine, so that further processing can be done on the results.

### 3.6 Initial Run

The first set of results are from the test dataset. The experimental set-up results are composed of this data structure:

- Training: Result = Trained Model: 20 images were annotated from both areas with 687 footprints, which is an average of 34 footprints per image.
- Validating: Result = Average Precision Value (result of the measurement between the Trained Model and the Annotated JSON): 6 images were annotated from both areas with 306 footprints, which represents an average of 51 footprints per image.
- Testing: Result = Detected footprints: 6 images were tested from both areas.

The images were chosen to show areas with footprints, but not only footprints in perfect conditions, which are clearly visible, but also footprints with all kinds of sand depths and moisture contents.



Certainly, in order for the machine to learn to "see" as much as possible, it has to adapt to the fact that a footprint can appear in several different ways in a given image.

The first initial results, the detection itself, were run on the testing dataset. The model has been created with the training database, on which the validation database can be run to get an average precision value (AP), which gives the difference between the trained model and the actually annotated images, and the final detection is shown on the testing dataset.

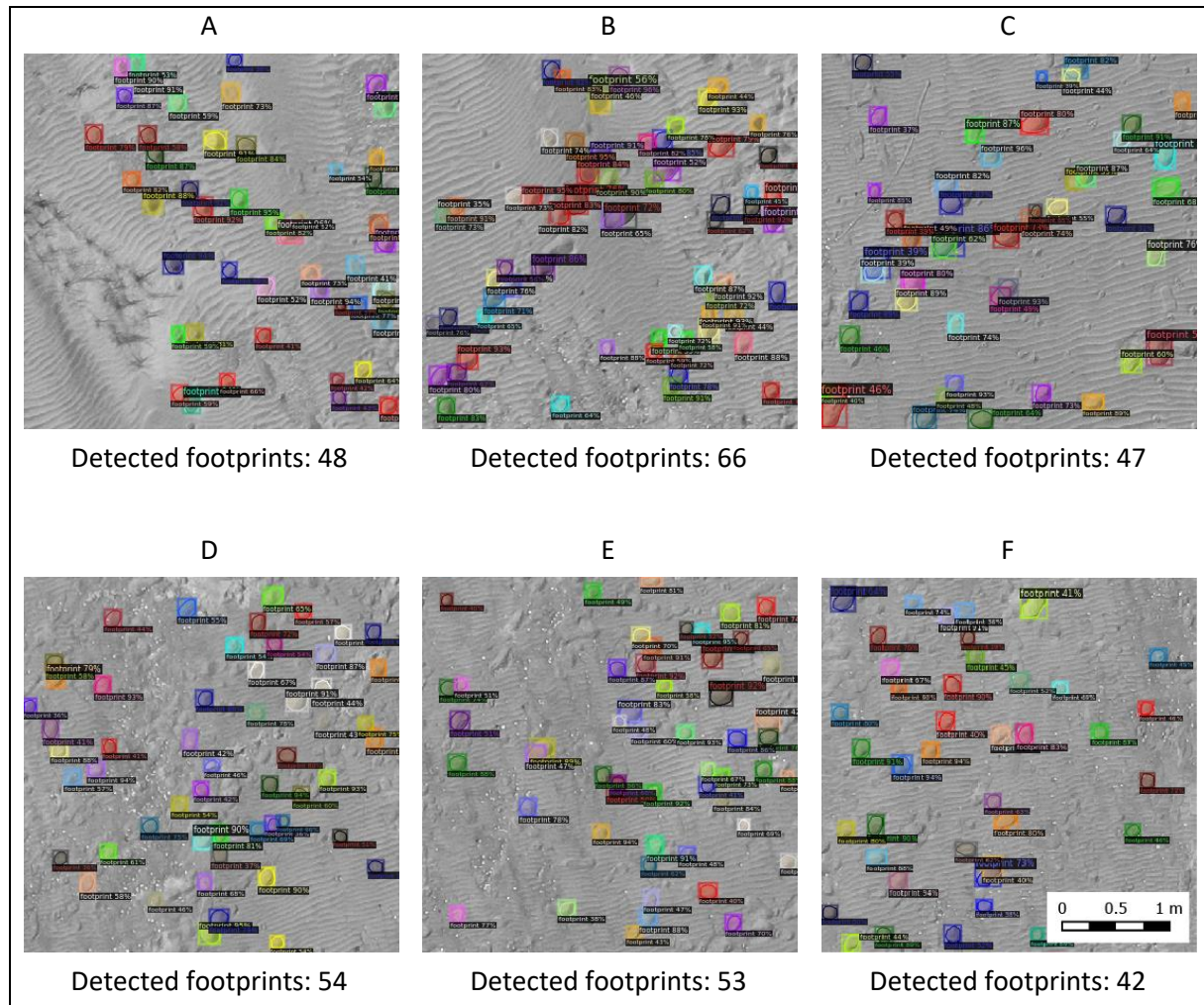


Figure 18: Visualization of detected footprints in the six image tiles used for testing the experimental run. Detected masks of footprints are coloured randomly and labelled with a confidence value (%). Image tiles are labelled A to F, with an overall detection count below each tile (ranging from 47 to 66 with an average of 51 over six tiles)

Figure 18 shows the detected footprints. The average number of footprints found is 51, naturally, this is only an approximate number, as it depends on the image. Some have only 42 (Figure 18 F) and others have 66 (Figure 18 B) footprints. This is not representative, as not all images have the same number of footprints. What is clearly important and visible is that there are footprints around the vegetation (clearly visible in Figure 18 image A), which is important because the aim is to examine the anthropogenic impact on the vegetation. Another fact worth noting is that in several places the paths are marked out, showing where people have walked and which route they have followed (especially in Figure 18 B and C images). This was promising.

After an initial run, the results were promising. After 1000 iterations, the total loss value drops to around 1.14 with an AP value of ~30. The trainer configuration settings remain the same as the default ones:

- SOLVER.IMS\_PER\_BATCH = 2
- SOLVER.BASE\_LR = 0.00025
- SOLVER.MAX\_ITER = 1000
- MODEL.ROI\_HEAD.BATCH\_SIZE\_PER\_IMAGE = 128
- MODEL.ROI\_HEADS.NUM\_CLASSES = 1

### 3.6.1 Refinements

Further refinement of the recognition process involves fine-tuning the training parameters for optimal accuracy and better results. The most common and easily accessible training parameters are the batch size, the learning rate, the RoI batch size and the number of iterations (epochs). The performance analysis and the refinement processes described in this section were carried out by modifying these parameters and examining their effects on overall training loss, AP and training time. Of course, these are different for each model, customized and depend on the nature and quality of the datasets. In general, the aim was to minimise loss (mean squared error), increase the AP and achieve acceptable training times.

In this performance analysis, a set of training parameters was defined as a baseline. While this set of training parameters gave visually accurate and pleasing results in one of the first runs (as shown in Figure 18), further refinements had to be made. The analysis was repeated by changing one training parameter at a time (in both directions: decreasing and increasing) and comparing it to the values of the baseline run. By observing this change in the loss, the AP, and the training time, some findings could be made. In addition to the change caused by the modified parameters, correlations between a parameter and the loss, the AP and the training time were also investigated to find the parameters that most influence these values.

The AP is the most popular metric for testing performance (Liu et al., 2019). This is a % value that compares the samples with the predicted samples. Hence, the higher the value, the higher the accuracy of the model, therefore the aim was to increase this percentage. Several variants of the AP are known, such as mAP (mean average precision), which is relevant if the prediction distinguishes several classes. For this reason, in this thesis, they were negligible.

The total loss parameter was presented as a loss function in the literature review. The essence of this is to minimize errors. Here, convergence to 0 was of primary importance (Durán, 2019; Vassányi, 2022).

The training time was also relevant, since the free version of Google Colab notebook automatically disconnects after 30 minutes, thus the running time could not exceed this. Given the limitations of the free Colab environment, training time should always be reasonable and tolerable (although some complex datasets and models may require much longer training times).

**Baseline:** As can be seen from the performance overview tables (see Appendix C), the baseline run was set to a batch size (IMS\_PER\_BATCH) of 2, a learning rate of 0.00025, a number of target iterations of 1000 and a RoI batch Size (ROI\_HEADS.BATCH\_SIZE\_PER\_IMAGE) of 128. For each parameter varied, the other three remained unchanged.

**Changing batch size:** The batch size (IMS\_PER\_BATCH) had the largest effect on the training time, with a very strong positive correlation ( $R=0.9997$ ), while it had a moderate positive correlation with the AP ( $R=0.5267$ ). The total loss was not dependent on the batch size, although in the extreme case (1), the value of the total loss (1.231) was worse than the baseline value (1.129).

**Changing learning rate:** The learning rate (BASE\_LR) clearly correlated with the loss, in a negative way. They showed a very strong correlation ( $R=-0.8981$ ), which means that a higher learning rate is associated with further loss minimization. However, the choice of the correct learning rate value is crucial, as in some cases choosing a value that is too high may result in the model moving towards a suboptimal solution, while a too low value may result in the training process stalling (Brownlee, 2019). The LR showed only a moderate correlation with the resulting AP value ( $R=0.4016$ ) and did not correlate with the training time ( $R=-0.0683$ ).

**Changing number of iterations:** The number of iterations (MAX\_ITER), which defines an epoch for the time when the training is completed, is clearly the most decisive parameter in terms of training time. With a positive correlation of  $R=0.9997$ , it can be said that choosing a higher target number of iterations results in an almost perfectly linear increase in training time. Compared to the 1000 iterations of the baseline, choosing 2000 iterations as the target number resulted in a much better total loss value (0.8537, -24.38% compared to the baseline). Therefore, there is a very strong negative correlation between longer iteration processes and lower total loss values ( $R=-0.9244$ ). However, at the same time, the AP value also decreased to 28.837, -7.64% lower than the baseline (iterations versus AP correlation  $R=0.6114$ ). This is still a strong correlation, and iteration numbers lower and higher than the baseline value show the outline of a bell curve (MAX\_ITER = 1020 was the highest, while lower and higher values resulted in lower AP values). However, this apparently "optimal" iteration number of 1020 also worsened the overall loss compared to the baseline (+4.96%).

**Changing RoI batch size:** The effect of the RoI Batch Size (ROI\_HEADS.BATCH\_SIZE\_PER\_IMAGE) parameter appeared to be negligible for both loss ( $R=-0.4087$ ) and AP ( $R=-0.1140$ ). However, this parameter showed a very strong correlation with the training time, with an R-value of 0.9983. This parameter linearly correlated with the training time (choosing a larger RoI Batch Size implies a longer training time).

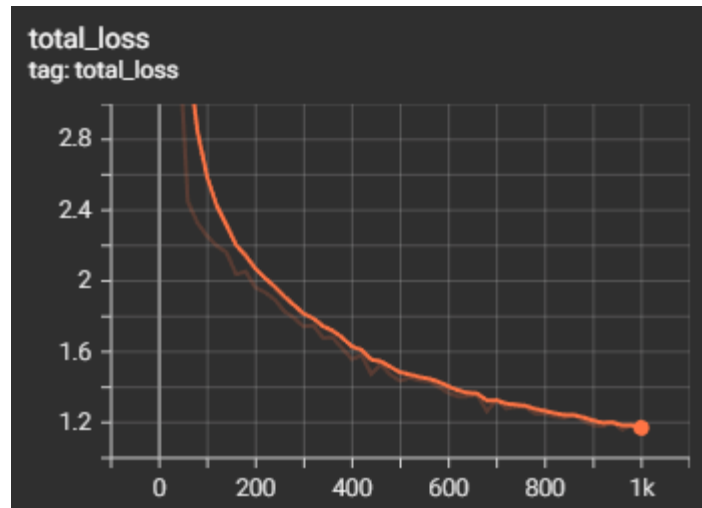


Figure 19: Changing of total loss over time (x-as: number of epochs; y-as: total loss) of the baseline run

Figure 19, which plots the change in total loss over time (1000 iterations of the baseline run), shows that the chosen LR appears to be optimal. Although it could be further optimized to achieve a lower total loss value, it did not show any anomalies that would indicate that the LR was too high (Brownlee, 2019).

### 3.7 Implementation

In the previous subsections, the entire workflow was presented in detail, starting from the use of an annotated sample image for the detection of objects to the visualization of detections on random samples. At this point, the workflow was considered feasible and ready for real-life use. While in the previous section small test dataset was used with randomly cropped images to make sure the recognition works well, it was time to present a real use case where a full survey dataset for feature recognition was used.

#### 3.7.1 Orthomosaic Creation

To do this, the raw images captured by the drone during a single grid flight must be merged as georeferenced orthomosaics. In this way, image fitting algorithms using spatial data embedded in the images (drone latitude, longitude, altitude, camera pitch, rotation, roll) and manually measured and defined (with RTK GPS device), high-precision control points can be used to spatially position the captured data as a single entity, an image.

This was done with the Agisoft Metashape software, first by aligning the photos, then by creating the meshes and orthomosaics. After the orthomosaic was done, which is a time consuming and resource intensive task, the next step was the export. It is important to note that the final image could be exported in several file formats (e.g., TIFF), however, the selection was made on JPEG, since the original material is already a compressed JPEG. Another important aspect was that the uncompressed TIFF file format produces enormous file sizes, which makes the processing difficult. What is crucial to mention is that when the JPEG is generated, a world file is also made, and when they are used together, the image itself is georeferenced.

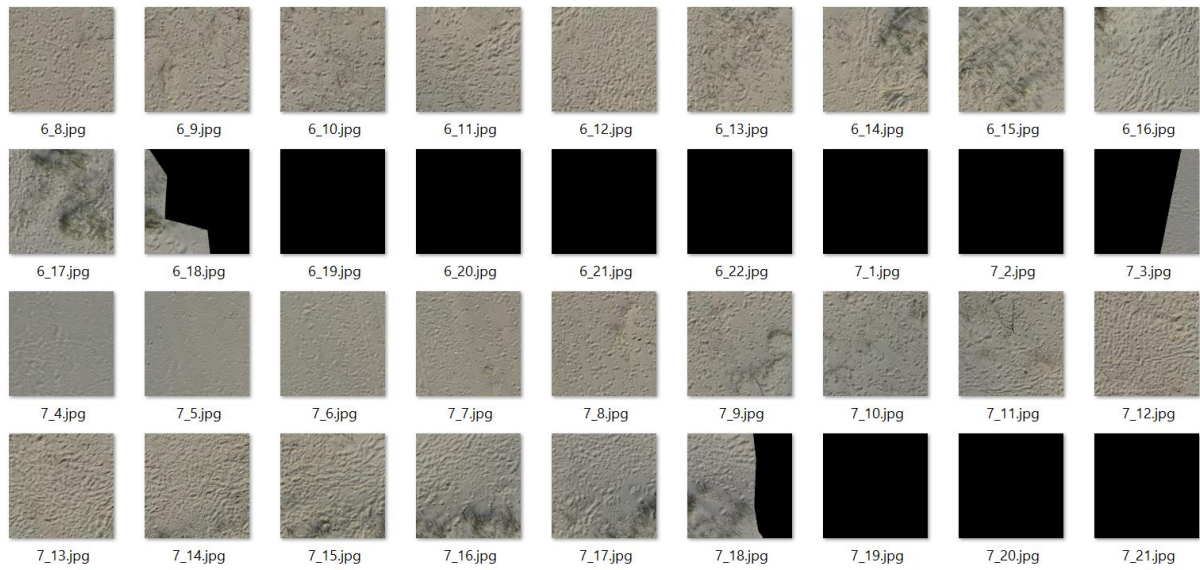
### 3.7.2 Image Tiling

Theoretically, it would be ideal to provide the full orthomosaic image to the object detection software, as in this way the georeferencing can be preserved and reused when the results are imported into the GIS software for spatial analysis. However, since the model was trained with tiles of 512x512 resolution, as R-CNNs work well on small images (Girshick et al., 2013), the predictor engine should receive a test dataset of tiles of the same resolution. In the case of using a much larger image than the one for which the model was trained, it is more than likely that the detection would fail with unexpected results (Thukur, 2022). For this reason, the merged orthomosaics should be tiled. To assist this tiling process, a library called SAHI (Slicing Aided Hyper Inference) exists, which is a lightweight vision library for performing large-scale object detection and instance segmentation and is intended to assist in automatic tiling of the detection and management of the results as a whole (Akyon et al., 2022). However, it was not functional with detectron2 for the current use case of footprint detection.

To tile the georeferenced orthomosaic, custom Python scripts were run to generate and join the tiles using row and column indexes. Since the exact resolution of the orthomosaic (which does not change) and its georeferencing data were known, if the entire image is tiled according to a defined structure and then reassembled in a later phase, the resulting image will have the same resolution as the source image. In between the generation and stitching of the tiles, there was, naturally, the object recognition process itself, based on which the footprints were exported as matching tiles.

Both Python scripts use the Pillow (PIL, Python Imaging Library) package, which provides image processing capabilities (Zenodo, 2023). The script to create the tiles was quite simple. After reading the input image (orthomosaic JPG or TIFF), it determined the input resolution and calculated the number of rows and columns to make 512x512 tiles. The script used two nested loops to handle rows and columns separately and iterated through the expected number of rows and columns to tile the image. Meanwhile, for each expected tile raster position, the resulting RGB tile images were saved locally in a subfolder, with the row and column indexes in their filenames. The file names were constructed in the form of "{row}\_{column}.jpg". This way, the tile structure was simple and persistent, and the file names helped to stitch them back together later.





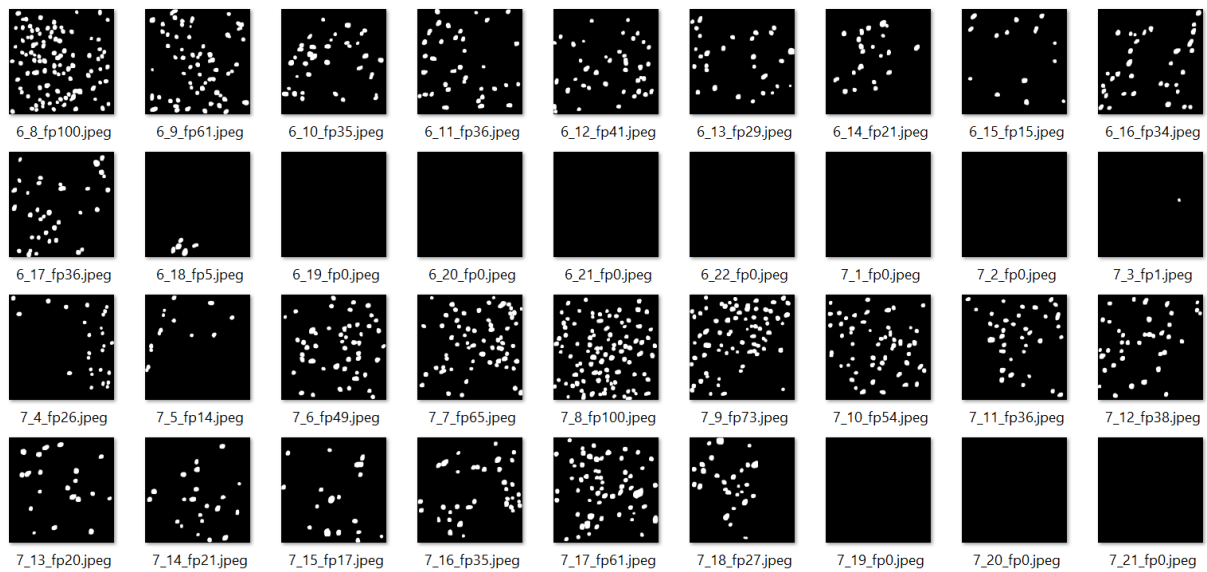
*Figure 20: Sample of tiles generated from the orthomosaic. The file name of each tile represents the row and column of the tiling structure for later reassembly. Partially and fully black tiles represent areas that are on the edge or outside of the orthomosaic*

The result of tile generation was a subfolder of tiles (Figure 20), which was used as an input "test" dataset for footprint detection. This dataset was incorporated into the data set as in the Methodology section, Dataset Construction subsection, and uploaded to the Colab virtual machine. As described earlier, the predictor was then used on this test dataset. To process and save the results, the numpy arrays of the resulting detection results (with Boolean data, True in place of footprint masks, False otherwise) were automatically converted to JPEG by the Pillow package and saved with the same file name as the input tiles. This step was combined and ran on the Colab machine to make the usage easier. At the end of the script that handled the detection results, the tiles containing the detection data were compressed into a .zip file, making it easy to download the detection data from the Colab machine to a local machine.

### 3.7.3 Image Stitching

This .zip file contained all the tiles with the detection data, with the same number of tiles and the same exact file name structure as they had when they were generated (Figure 21). To stitch these tiles together, another Python script was developed. This script scanned a given folder and used Regular Expressions to search for JPG images and the row column indexes in their file names. It detected the maximum number of rows and columns (based on which the image was reassembled). It then created an empty image in memory with a resolution corresponding to all tiles (rows\*512 and columns\*512). As with the tiling script, there were nested loops to handle rows and columns. When a matching tile was found, its contents were inserted into the empty image in memory. After inserting all tiles, the image in memory supposed to match the original resolution of the source orthomosaic. However, since integer numbers (whole numbers) were used to define the rows and columns for tiling, the bottom and rightmost rows and columns contained some blank spaces. This did not cause any problems or anomalies in object recognition but had to be managed to match the resolution of the source orthomosaics to maintain georeferencing. At this point, the script expected input from the user: the

user had to enter the width and height (in pixels) of the source image. Now the target sizes were known, and the image in memory could be cropped to this resolution and eventually saved to disk.



*Figure 21: Sample of tiles which underwent the detection process and were exported as JPEG images. In addition to the row and column number for the tiling structure, file names also contain the number of detections on a specific tile as complementary information. The tiles now contain white pixels for the footprint masks, and black for the surrounding area*

### 3.7.4 Georeferencing

When initially creating orthomosaics and exporting to JPG in Agisoft Metashape, a world file was also required and generated to maintain georeferencing. This file contained the georeferencing data and, as the resulting stitched image had the same resolution as the original orthomosaic image, it could be reused.

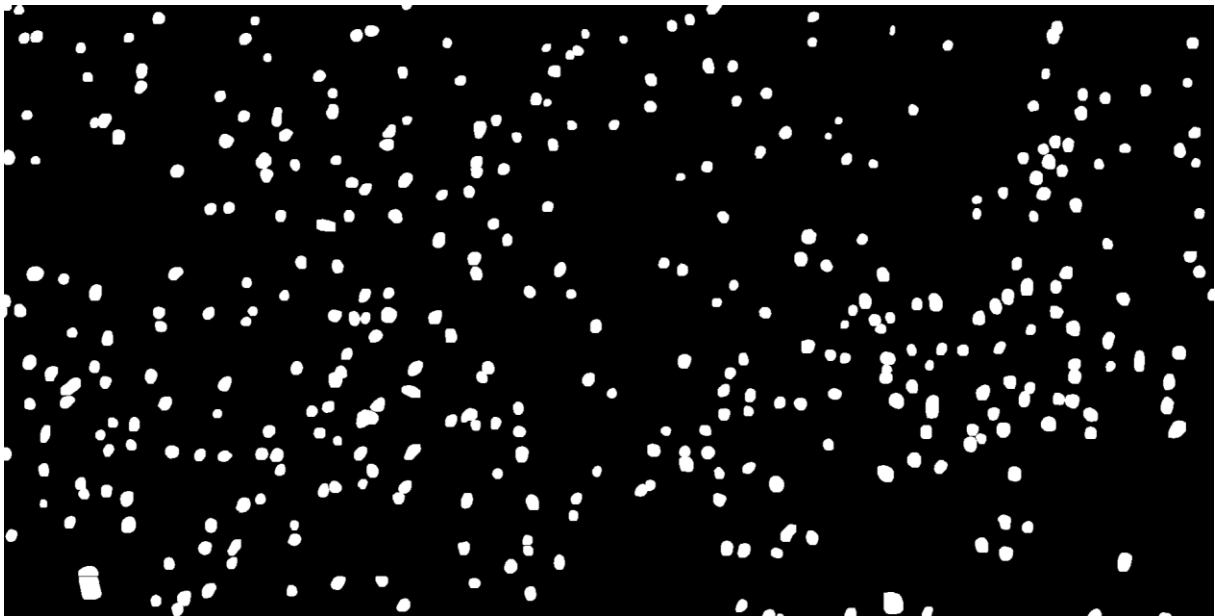
When imported into GIS software, the world file needed to have the same file name as the resulting image and had to be placed at the same location. This way, the GIS software automatically recognized the world file as georeferenced and placed the image correctly in space. At this stage, the raster image could be processed, and spatial analysis could be performed on it.

## 4. Results

In this chapter, first the raster file created from the processed data and the developed workflow are presented as the result of the footprint detection. Then the accuracy of the model is discussed, in addition to what has already been mentioned in the refinement section. The next subsection (Chapter 4.2) contains the spatial analysis, which is broken down into further subsections to present the temporal and spatial differences.

### 4.1 Footprint Detection

The previous object detection in the Methodology phase yielded the expected results, from which a merged file was obtained containing the footprints detected on and around the dunes of Hargen aan Zee. An extract of the resulting data is shown in Figure 22. This sample contains white and black areas (values 1 and 0 in the raster), the first of which represents the footprints observed around the dunes during the April 2021 survey.



*Figure 22: Sample binary map of the detected footprints (excerpt). Footprints are marked white, non-detected areas remained black. (Hargen Aan Zee, April 2021)*

#### 4.1.1 Accuracy

In the previous chapter, the refinements of the hyperparameters were laid out in the model refinements after the experimental setup. This was necessary in order to run on the full data set with well calibrated parameters to get the best possible results. Once all options were tested, the model showed the best performance with the same parameters as in the default settings. The best parameter settings are not only shown by the Average Precision value, but as explained in the refinement, by the total loss and the training time together. The parameters thus remained the same as for the experimental setup, which were: batch size = 2, learning rate = 0.00025, number of target iterations = 1000 and RoI batch size = 128, with total loss = 1.129, AP = 31.224 and training time = 351 seconds.

Although this result can be considered a significant milestone, as the object detection was successful, not many conclusions can be drawn from this detail alone, as the data is not yet linked to other data

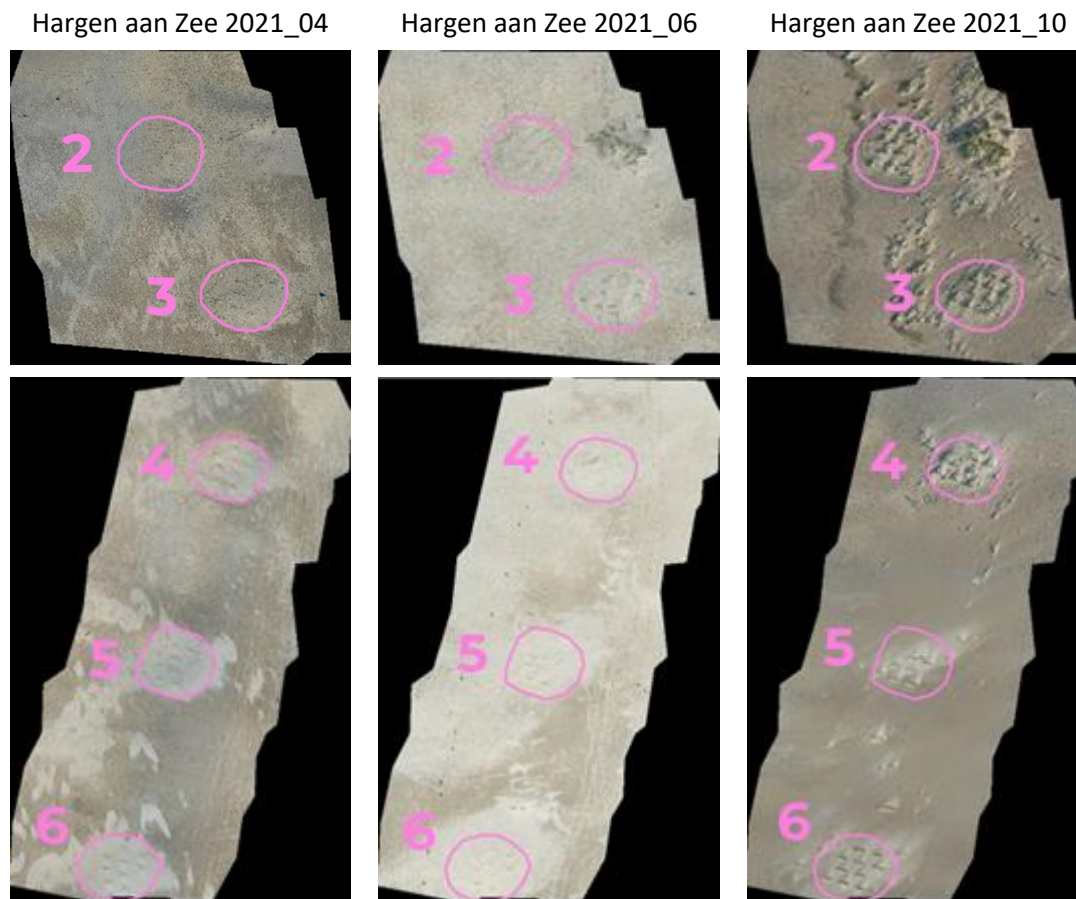


(either spatial or temporal). At this point, the image pre-processing, the detection and the post-processing were complete, and the workflow can be used in any relevant project involving UAV imaging and automated object detection – it all depends on the project in which the approach is applied. However, in addition to the development and validation of the object detection method, this thesis also aims to use the detection data to correlate and draw basic conclusions about the vegetation of the dunes.

## 4.2 Spatial Analysis

As the final result (raster file) matches the resolution and extent of the original orthomosaics, the raster can be imported into the GIS software with georeferencing to allow further analysis, linking the data to the vegetation. QGIS was used to display the results and examine the spatial locations. The location of the footprints can be visualised in this geographic information system, so the focus in this chapter will be on their analysis.

Three dates were analysed, which are the UAV images taken from flights in April, June and October, 2021, since it is beneficial to set up a study from data collected from different periods. Another aspect of choosing particular flight times was that the other surveys were irrelevant due to the presence of wet sand caused by heavy raining. No footprints were visible on these, so detecting and analysing them were undoubtedly pointless. Nonetheless, the images from the three selected dates were sufficient, as these images were taken with different sand moisture contents, contrasting visitation frequencies and different growths of vegetation (Figure 23). They will be referred to as 2021\_04, 2021\_06, 2021\_10. It is also important to note that 6 dunes were built in the Hargen aan Zee area, numbered as 1-2-3 and 4-5-6. However, shortly after the first flight in April, dune number 1 disappeared due to weather conditions, thus only the rest (dunes 2 to 6) will be presented in detail.



*Figure 23: Orthomosaics of images taken in Hargen aan Zee, on three different flights (April, June and October). The individual dune fields are highlighted in pink and numbered 2 to 6 (southwards) to ease identification. The closest beach entrance is between fields 3 and 4*

The flights did not take place on a fixed route every time, therefore the orthomosaics did not cover the exact same area. On the other hand, to keep analysis and comparison as objective and comparable as possible, it was necessary to define a common area that was covered by the orthomosaics of all three flights. Within this, the dunes in the area shown in Figure 23 are outlined in such a way as to cover the dune groups in all the images taken at the different flight times. This means that while in the April survey one dune group was X and Y in extent, this changed in all subsequent surveys. These are due to natural processes, weathering, vegetation growth and erosion. These were summed up to create a polygon around them so that they could be investigated, overlaid, and analysed in future. The polygons are approximately 290 – 345 m<sup>2</sup>.

The first thing to note is that the final result was created in a JPEG format, as mentioned in the previous chapter, in order to avoid having to work with large files in uncompressed TIFF format. Since JPEG works with RGB bands (as opposed to TIFF, which supports a single-band data format), when imported into QGIS, the image appears as a three-band image, as expected. The JPEG images containing the footprint masks also suffered some compression, as the tiles were saved in JPEG format and there was some noise around and within the footprint masks (~254/253/252 and ~1/2/3 values). To address both of these issues in a single step, a reclassification had to be performed as the first step of the spatial analysis. The reclassification was performed using a simple remapping table, which assigned all values

above 250 to 1 and all values below 5 to 0 in all three bands. The raster now had boolean type values, so where there is a footprint, the value is 1, otherwise it is 0. This can be seen and checked by using the single band pseudocolour symbology, which shows 1 as white and 0 as black. The processing continued with the vectorisation, during which the area of the detected footprints was also calculated. The result is shown in Figure 24. While Figure 22 shows only the detected footprints, here the vectorised footprints and the orthomosaic underneath demonstrate the visual success of the detection model.

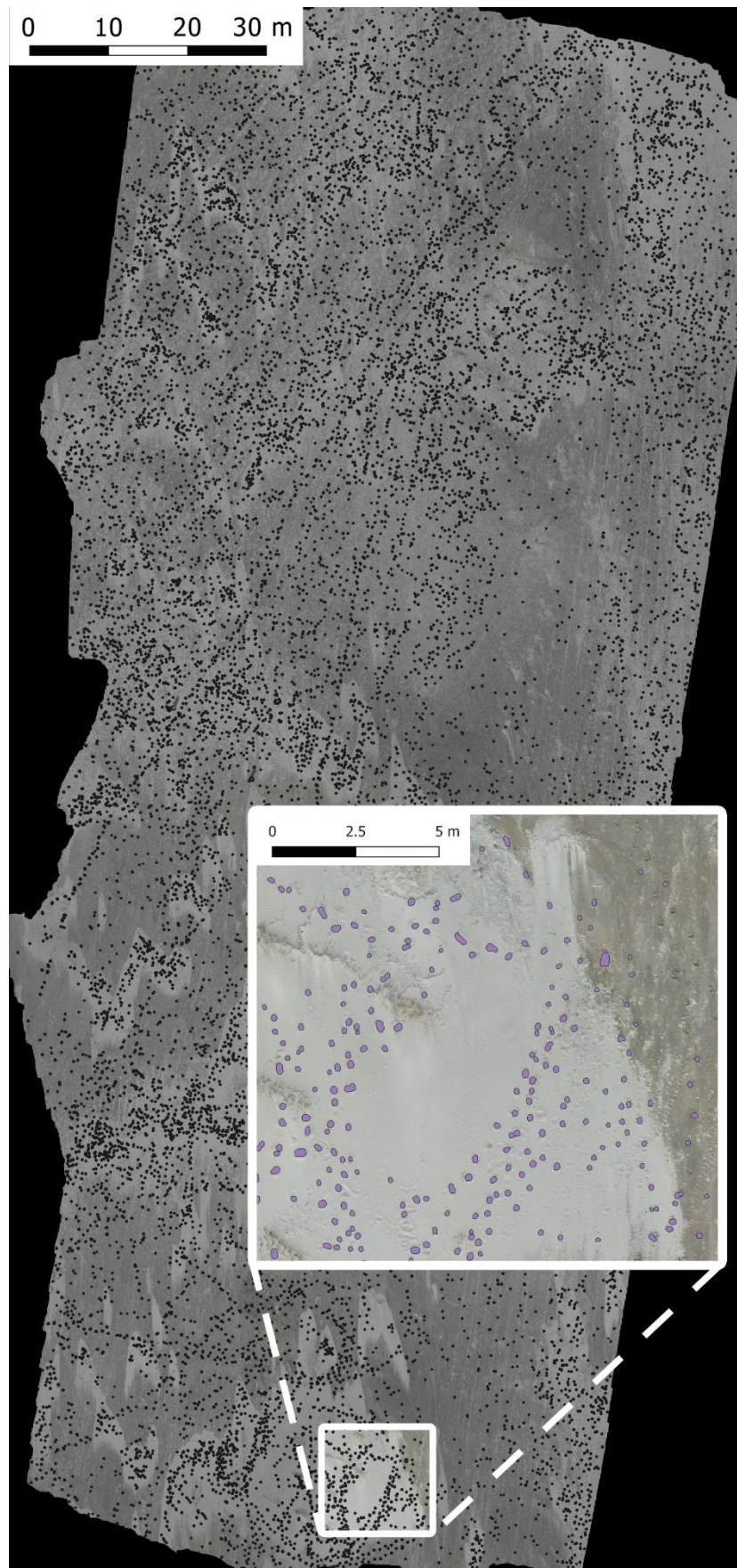
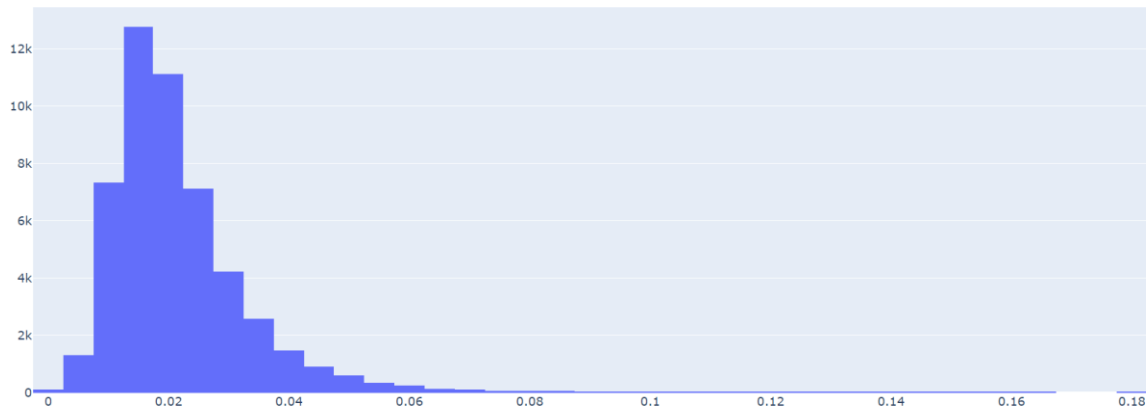


Figure 24: Result of vectorization in the orthomosaic in the survey of April, with vicinity of dune field 6 highlighted



*Figure 25: Histogram of the footprints' areas from all the surveys*

By examining the merged shapefile created by the vectorized footprints from all surveys, further results were revealed. Examining the footprint area histogram (Figure 25) generated and combined including all three datasets (04, 06 and 10) reveals that the average footprint area is 0.0189 m<sup>2</sup> (median). The dimensions vary slightly between the three surveys, but no significant conclusions can be drawn. This median value seems realistic for an average human footprint with one shoe. The median value also confirms the spatial integrity and accuracy of the data in the GIS system.

#### 4.2.1 Temporal Changes

Figure 24 clearly reveals where the actual footprints are, what the trained model found, the quality of the sand at this particular flight and the physical presence of vegetation in April. Since the primary goal in the spatial analysis was to produce a heatmap, it is known that this can only be generated from point type objects. Thus, the processing continued by determining the centroids of the polygons (generating points) and creating a heatmap from them. The heat map is the first analysis that actually expresses the spatial distribution of the footprints. However, care must be taken when creating it because calibrating the parameters, i.e., finding the right values, can be time-consuming. Since the area of each polygon feature was included in the attribute table, these area values were used as weights to create the heat map (Figure 26). To generate heatmaps a search radius of 5 m was used consistently.



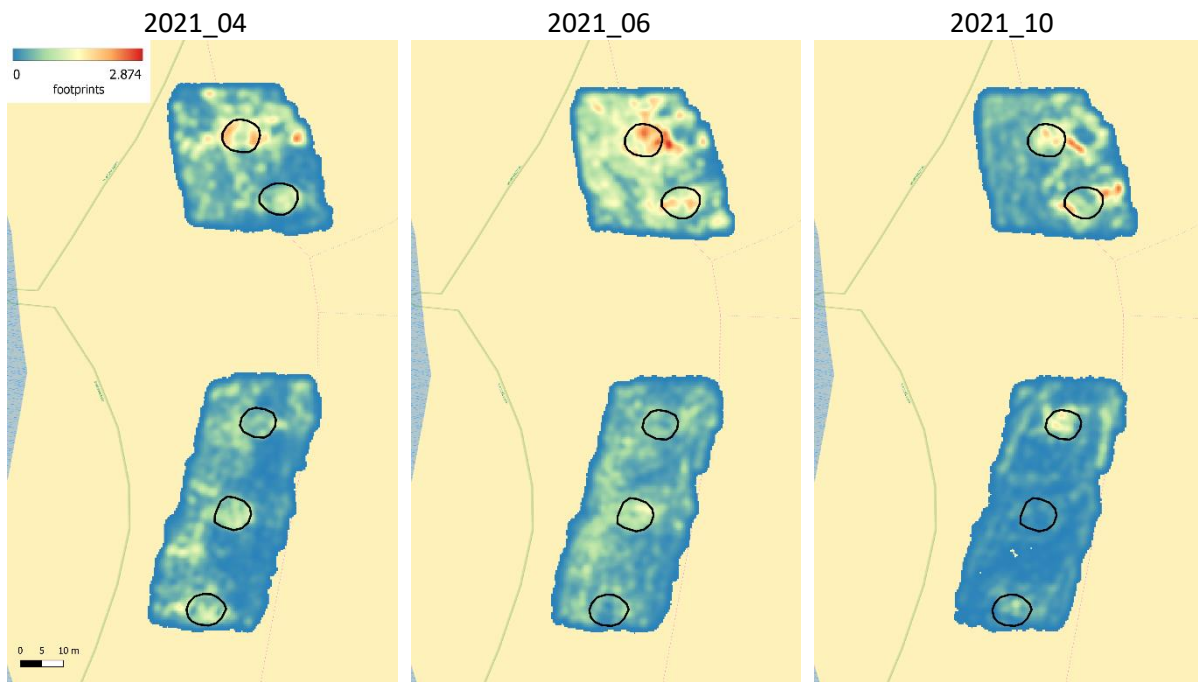
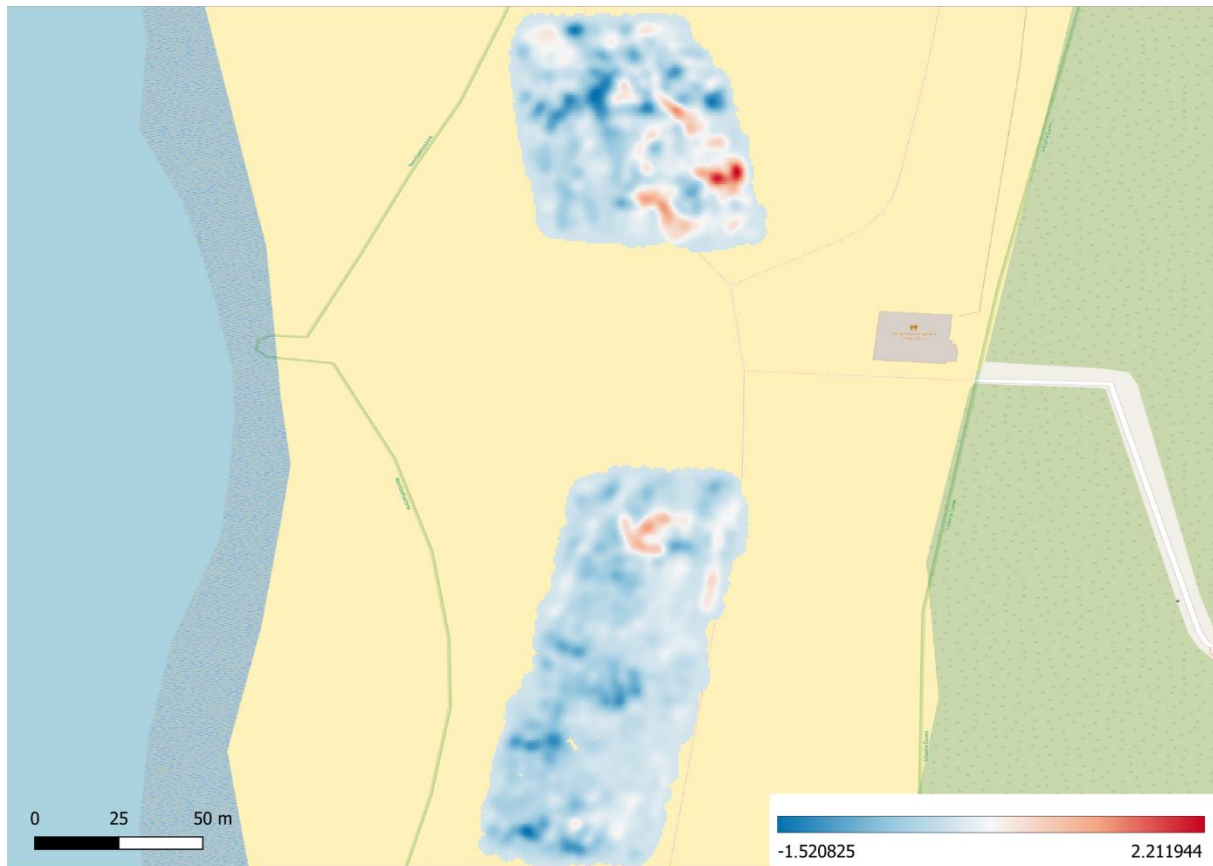


Figure 26: Heatmaps generated from the number of footprints falling in a defined search radius of 5 m, weighted by footprint area (2021\_04; 2021\_06; 2021\_10)

In Figure 26, the heatmaps show the three analysed periods. The values, i.e., the colour scale from blue to red, show the amount of footprints within the given search radius as pixel values. This is a pleasant presentation of the density and location of footprints, but the result is not unambiguous. It can be seen that the visitation was the highest in June in both areas, on the dunes north (2-3) and on the south (4-5-6) of the entrance. In October, hardly any footprints appear on the more distant southern dunes, and much less often in the northern parts compared to the survey made in April and June. However, this heatmap is not objective as the different sand texture may alter the results. The measurement in October was taken after a rainy day where no footprints are visible in the wet sand. The sand around the vegetation, on the other hand, looks drier, thus the detection model predicted more in those areas.

Since heatmap rasters are available for all three dates, further results can be inferred from them. These heatmaps contain pixel values to determine the number of footprints within a search radius. To compare and calculate the differences between two contrasting dates (April and October, the first and last datasets in time), the two heat maps were subtracted from each other (by subtracting the oldest from the most recent: October - April). By using the Raster Calculator, a new heatmap (Figure 27) can be used to produce a practical result with both positive (more footprints) and negative (fewer footprints) values, indicating the change from April to October. The difference in the heatmap shows that the area around the northern dune group was exposed to higher visitation pressure in October than in April, while the southern part had lower visitation pressure.



*Figure 27: Calculated difference heatmap to visualize visitation change between April and October (blue - decrease, red - increase)*

Another interesting quantification of the results is the ratio of the number of footprints in dune groups to the number of footprints in the total surveyed area. The number of footprints in all dune groups (2 to 6) and in the surrounding areas was examined and the percentages were calculated. For this purpose, the contour lines of the pre-drawn dunes were used. The values are 17%, 13% and 16% during the April, June and October flights, respectively (Table 2). This is the percentage of the total area that had footprints on the dunes and not in the surrounding areas. This clearly demonstrates the findings in the heatmap section namely, that there are not necessarily more footprints during the June survey despite what the numbers show. There are clearly far fewer footprints during the October survey, but the proportion of footprints in relation to the total area is the same, 16%. The same relationship was made for the areas. The ratio between the area covered by total footprints and the area covered by footprints visible on the dunes correlates to the previous calculation of the number of footprint pieces. There are many factors involved, and due to the consistency of the sand, it is not possible to state univocally that there were more footprints in one survey than in another one.

Table 2: Footprint statistics (number and area of footprints) for three months (April, June, October)

	2021_04	2021_06	2021_10
<b>Number of footprints on and around the dune fields</b>	2820	2476	2281
<b>Number of footprints in the whole study area</b>	16770	19776	14154
<b>Ratio</b>	17%	13%	16%
<b>Area covered by footprints on and around the dune fields (m<sup>2</sup>)</b>	69.328	78.279	53.187
<b>Area covered by footprints in the whole study area (m<sup>2</sup>)</b>	344.184	493.408	261.699
<b>Ratio</b>	20%	16%	20%

#### 4.2.2 Spatial Changes

Since it is difficult to compare images taken at different times, this section will analyse the spatial extent and the relationships between them, thus, the comparison of dune fields.

Figure 28 shows the total footprints found by the model for these specific flight periods. The blue, green and grey colours indicate the April, June and October survey, respectively. The footprints are also divided into dunes representing which dunes are most exposed to anthropogenic pressure. Figure 29 shows the same data, normalized by the total number of footprints found in the study area in a specific month. Dune 2 has the highest numbers and as we move away from it, there are fewer and fewer footprints on the dunes. The assumption that dune fields 3 and 4, closest to the entrance, have the most footprints is not correct. However, it is noticeable that people mainly walk northwards. This can hypothetically be explained by the nearby presence of the Uitzichtpunt De lagune located North of Hargen aan Zee, which attracts visitors. The dunes more distant from the lagoon (the distance from the dune field 6 to the nearest point of the lagoon is 760 m) have fewer footprints. Besides, it is worth mentioning that there is also a beach entrance in the norther part, which can only be reached by a relatively long walk. Although the dunes from the October flight are the clearest and the most distinguishable, the footprints are less visible due to the aforementioned wet sand. What is not insignificant is that half of dune 5 disappeared over the months, probably due to vehicles used by the water rescue teams.



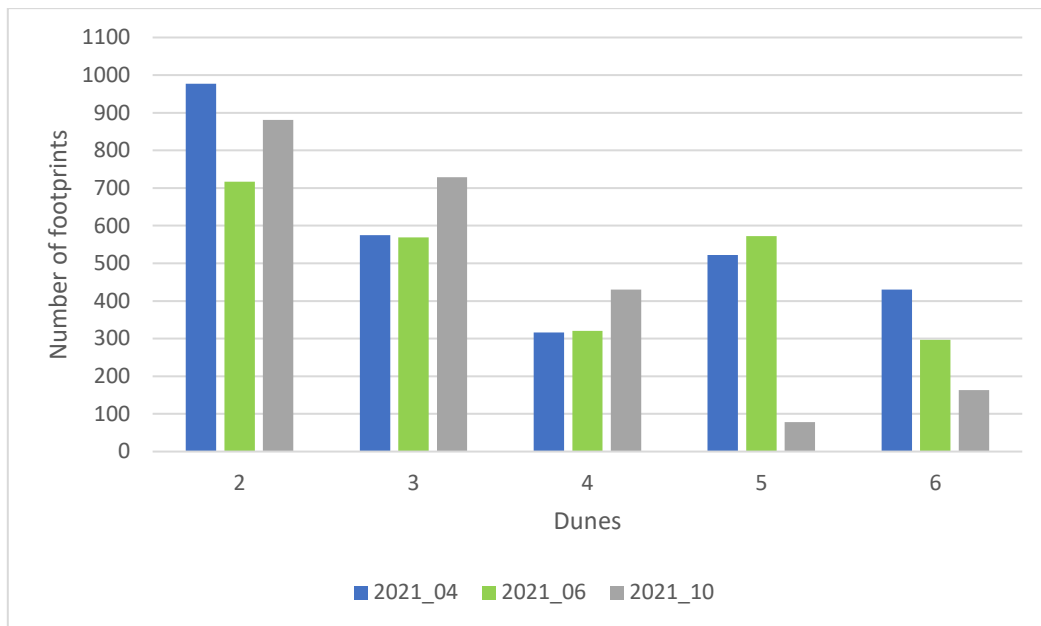


Figure 28: Number of footprints for different times and dunes

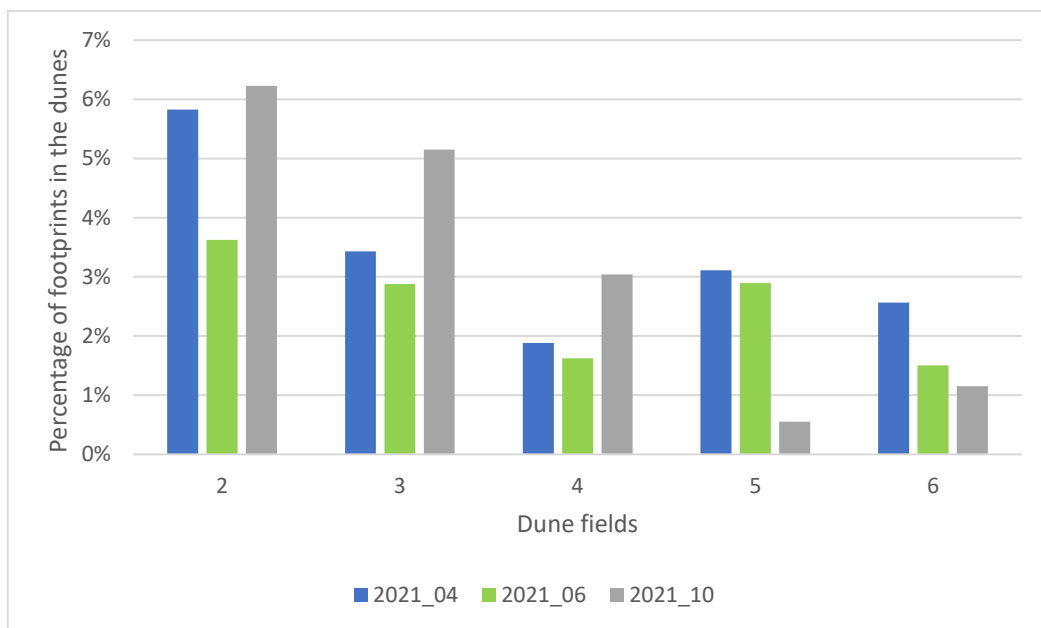


Figure 29: Number of footprints for different times and dunes, normalized by the total number of footprints found in the whole study area, expressed as a proportion

## 5. Discussion

This research project was successful with several new conclusions to be drawn. It must be stated that many decisions were made during the process, each of which influenced the other and thus the final result. This section both discusses the methods used and evaluates the results obtained. Limitations, future work and possible improvements will also be elaborated. The objective evaluation and reflection of the topic are realized here.

Based on Martínez et al. (2013) in order to obtain accurate data on the response of vegetation to anthropogenic impact, it is essential to locate the places where the pressure is the greatest. With the presented project and methodology, a step towards identifying these locations was taken. The methodology can potentially help to address this issue in a semi-automated way, so that further findings can be derived and the relationship between vegetation and human pressure can be investigated.

### 5.1 Findings and Obtained Results

The method presented in this thesis achieved a detection process that even though requires some knowledge in programming and file handling, is relatively easy to use. In the light of the results, it can be stated that the trained network is able to detect footprints in the test images, although the efficiency of the method can be significantly increased mainly in terms of accuracy.

Going sequentially through the whole report, first of all, the footprint definition must be mentioned. The initial idea of finding “peanut” shape footprints and then replacing them with an object approximating a circle proved to be a good approach. As the varying consistency of the sand means that circular shapes of footprints appear the most often, it made sense to work with them. However, it should be mentioned that this limited the diversity of footprints to one type of shape, while there are more types as shown in Figure 16. Since this kind of training of an object recognition model relies solely on the training patterns obtained, the omission of other shape types from this training dataset resulted in a slight decrease in accuracy. The training dataset also did not include other types of footprints found in specific areas (e.g., vehicle track lines or patterns created by dogs and horses). Although the majority of footprints in sand were recognized (even those shapes that were not a 100% match and therefore slightly different from the training samples), the results showed some recognition errors (including missed footprints). Although this reduced the overall number of footprints found in the study area, the main paths were still clearly visible, and the relative density was still considered correct and useful for further spatial analysis. The trained model produced only a few false positives (none in some orthomosaics), which in the magnitude of tens of thousands of found footprints is clearly a positive outcome and an indication of uncontaminated, pure results whose accuracy can be further improved.

The physical volume of the training database proved to be sufficient for the operation of the model. This means that around 700 footprints have been annotated, but raising this number is likely to produce better results, as well as increasing the diversity of the footprint appearance. This is an extremely time-consuming task and bringing it to a near-complete level with thousands of annotation

samples can take a very long time. During the thesis, the training database was created from several types of sand and sand depths. This variety is the key when training such a model.

Examining the methodology, it can be stated that although the workflow can be implemented at any time, it is relatively divided and consists of several parts. It was designed offline so that the tiling of the images and then the post image processing is also done manually on a local computer. The reason for this is simplicity and faster workflow editing, therefore the development of scripts and the experimentation in the meantime was easier in this way.

The examination of the results reveals that they are adequate, the detected footprints are clearly visible and geographic information is provided for them. This facilitated the spatial analyses during which the locations of the footprints and their relation to each other were evaluated. However, a related piece of information is that the final result is in JPEG format, thereby facilitating the speed of processing. Of course, the lossless TIFF format is considered the best when it comes to image quality, but one has to consider whether it is worth working with these TIFF files when their physical size is huge, slowing down all the steps in the workflow. With this, the image quality of JPEG slightly degraded, which was accepted, as well as the well-known information associated with this decision that JPEG images store values in 3 bands (R, G, B), while TIFF would have stored in 1 band. Therefore, reclassification had to be done as the first step of the spatial analysis in order to eliminate unnecessary noise. By doing reclassification, a binary raster was created with a single band, which also facilitated vectorization.

During the different analyses, some insight into the anthropogenic effect on vegetation was provided, but as already mentioned, it is not completely black and white as several factors influenced the model during the detection. Such a model is very sensitive to the environment in which the predicted object resides, thus the results are different for each flight period. This environment is outside, the moisture content of the sand and the presence of wind cannot be influenced. This situation was demonstrated in the heatmap analysis, where it was not possible to objectively compare one time with another due to the fact that the environmental conditions were different when the images were taken. Therefore, it cannot be said unequivocally that June was the most visited period, despite the fact that the absolute footsteps numbers suggest this to be the case. Analysis of the images from June indicated that more footprints were detected in the study area on this flight than in April and October, but this is in inverse correlation with the percentage of the footprints that are located around the dune fields. In summer, the sand is generally drier, which, combined with the constant wind from the coast, may have made accurate detection more difficult. Although the dune fields provide some wind protection to prevent footprints between vegetation from fading, there were fewer footprints overall in these areas. Despite the fact that more people are likely to visit the entire beach area for recreational purposes in June than in October and April, people are less rather to wander around the artificial dune fields in summer. The October survey showed the fewest footprints, but the sand surface was also wetter, and the conditions were less tolerable due to the typically higher wind speed. The low moisture content of the sand is more favourable for optimal detection (due to contrasting edges and well-defined shapes), however,

the October survey still showed fewer footprints - these factors together confirm the lower visitation of the area in autumn. In terms of the spatial distribution, the upper part (dune fields 2 and 3) was more popular than the southern area (dune fields 4, 5 and 6) in all three months. This may indicate that people are moving northwards from the beach entrance towards the Uitzichtpunt De lagoon.

## 5.2 Future Research and Possible Refinements

Future work and refinements include upgrading the footprint definition. Not only one type of footprint can be trained in the model, it can also be trained to many object types. With this option, a "peanut" shaped footprint can be added as another type of footprint, or even footprints that are only half drawn. This increases the amount of data, potentially increasing the accuracy of the model on images captured under different environmental conditions.

It is not only the quality of the dataset that could be improved, but also the quantity. Making the dataset in the desired size could provide a solution to higher accuracy, and then this could be used to retrain the neural networks. There is no limit to the amount of data that can be trained, and in fact, by plotting thousands of footprints, a higher Average Precision can be achieved. Another important aspect that needs to be addressed when increasing the quality and quantity of the dataset is that the footprints seen in Figure 16 often appear to be squishy, this problem can also be eliminated with more training. If the time is available, this dataset can even be made global in the same way as the COCO one, which contains thousands of images, but the current dataset could be a basis for different footprints in the sand. However, creating a global footprint dataset that could be used as a benchmark would probably require a huge community effort to combine the most diverse footprint types and patterns.

Theoretically, the methodology can also be applied in the Google Colaboratory interface. This would mean that the dataset uploaded to the Colab interface would contain the orthomosaic created in Agisoft Metashape, and the image tiling, footprint detection, and all stitching can be applied in the program code on the Colab interface. The result could be downloaded to the local computer as a single image containing the detected footprints. This workflow would eliminate the need for a local computer, which would make the methodology more streamlined. In this case, only the orthomosaics would have to be done manually. The result would be the same as the methodology that this thesis used, on the other hand, a single platform would make the process more straightforward and easier to use. However, it is also important to note that if the whole process were run on Google Colab Notebook, there is a chance that the 30 minutes of training time would be exceeded and the connection to the remote computer would be lost. In this case, it is worth using an upgraded package for which a fee is charged.

For easier orthomosaic creation, extraction of results, and easier implementation of spatial analysis, it is worth running the same flight path every time, making it easier to process and compare the data.

Overall, since training time was not excessive, improving the quality of the training dataset seems to be the most optimal approach to achieve better recognition results. This could be accomplished by increasing the number of annotated samples and by including all types of footprints in the images.

## 6. Conclusion

Coastal vegetation and the formation of new dunes can be linked to the anthropogenic pressure on the coast. The aim of this thesis was to map the human use of the coast by processing UAV images and to establish an association with the vegetation present. For this purpose, an object detection method was developed to detect human footprints in orthophotos of the sand. First, the dataset was processed, the footprint definition was established, and the images were annotated. Detection was then performed by the Detectron2 framework, which included training the model with manually outlined footprints, validation and testing. The detection carried out was considered an initial run, as it was a run that used sample images to verify the correct recognition of the dataset, the training parameters and the integrity of the detected footprints themselves. After this initial run produced optimal results, both quantitatively and visually, the dataset for the real-world use case was created. In this stage, the original source images (captured by a drone at Hargen aan Zee at three different times) were processed to generate georeferenced orthomosaics of the study area. In order to use this data as a test dataset for object recognition, the image had to be tiled using a predefined structure to allow further assembly (after object recognition). Specific scripts were developed to assist and automate tiling in the pre- and post-detection phases. The results were displayed in a GIS software for spatial visualisation and further examination. Various spatial analyses were performed on the Hargen aan Zee dataset: raster reclassification to eliminate noise, heatmaps, temporal and spatial data analysis, vectorisation, centroid generation, footprint count and area analysis (including histograms). These analyses showed that there are numerous footprints around the dunes which may influence the vegetation growth, but also that there was no significant difference in the number of visitors between the different seasons. Around the dune fields 13-17% of the total footprints were found. The average size of the footprints (median) was 0.0139 m<sup>2</sup>. It seems that looking at the data spatially, it cannot be said that there are more footprints at the beach entrance. This is certainly due to the fact that images taken in different environmental conditions were examined. The model was evaluated under parameter refinements. A few findings were collected, and recommendations were made for future improvements.

The presented method is based on open-source components, which enables wide applicability. Training and detection can be performed on any UAV image, as long as enough training data can be assigned. Although the objective defined at the beginning of this thesis was fulfilled, it can still be stated that deep learning neural networks offer much more possibilities and this topic only exploits a fraction of these state-of-the-art areas. There are many applications of this field in spatial computing, but also this could be an excellent direction for building new models in the future. By using the procedure, the amount of monotonous, time-consuming human work can be greatly reduced, which can result in a more useful allocation of energy devoted to further related research.

## 7. References

- Adamczyk, J. (2020, July 18). *Understanding Detectron2 demo and examples | Towards Data Science*. <https://towardsdatascience.com/understanding-detectron2-demo-bc648ea569e5>
- Agisoft Metashape. (2022). *Agisoft Metashape*. Retrieved January 29, 2023, from <https://www.agisoft.com/>
- Akyon, F. C., Altinuc, S. O., & Temizel, A. (2022). Slicing aided hyper inference and fine-tuning for small object detection. *Proceedings - International Conference on Image Processing, ICIP*, 966–970. <https://doi.org/10.1109/ICIP46576.2022.9897990>
- Blue Marble Geographics. (2022). *Global Mapper*. <https://www.bluemarblegeo.com/global-mapper/>
- Brownlee, J. (2019, January 25). *Understand the Impact of Learning Rate on Neural Network Performance*. Deep Learning Performance. <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/?fbclid=IwAR0JxJfAfOp5yrYP2B-99mNQrp-PDcCvhruEAYIdnvT0AlzKdf5mhNghkLM>
- Bushaev, V. (2017, November 27). *How do we 'train' neural networks?*. Towards Data Science. <https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73>
- Cankaya, F. (2022, October 5). *Detectron2 Starter Guide for Researchers*. Towards Data Science. <https://towardsdatascience.com/detectron2-starter-guide-for-researchers-fbafc82428dd>
- Chablani, M. (2017, August 21). *YOLO — You only look once, real time object detection explained*. Towards Data Science. <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>
- Cloud Factory. (2022). *Image Annotation for Computer Vision*. 2010 - 2022. <https://www.cloudfactory.com/image-annotation-guide>
- Darkhan, S. (2022). *Equipment identification through image recognition*. Aalto University School of Electrical Engineering.
- Deeplizard. (2017). *Deep Learning Fundamentals*. [https://deeplizard.com/learn/playlist/PLZbbT5o\\_s2xq7LwI2y8\\_QtvuXZedL6tQU](https://deeplizard.com/learn/playlist/PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU)
- Detectron2 documentation*. (2023). <https://detectron2.readthedocs.io/en/latest/tutorials/index.html>
- Detectron2 documentation*. (2023). *Training — detectron2 0.6 documentation*. <https://detectron2.readthedocs.io/en/latest/tutorials/training.html>
- Durán, J. (2019, September 19). *Everything You Need to Know about Gradient Descent Applied to Neural Networks*. Medium. <https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14>
- Dusek, B. (2020). *Közlekedési táblák automatikus térképezése*. Eötvös Loránd University.
- Dutta, A., & Zisserman, A. (2019). The VIA annotation software for images, audio and video. *MM 2019 - Proceedings of the 27th ACM International Conference on Multimedia*, 2276–2279. <https://doi.org/10.1145/3343031.3350535>

- Farkas, R. (2022). *Machine Learning in practise*. <https://www.inf.u-szeged.hu/~rfarkas/ML20/alapfogalmak.html>
- Gandhi, R. (2018, July 9). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. Towards Data Science. <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>
- Ganesh, P. (2019, October 18). *Types of Convolution Kernels*. Towards Data Science. <https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37>
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 580–587). <https://doi.org/10.1109/CVPR.2014.81>
- Girshick, R., Massa, F., Lo, W.-Y., Wu, Y., & Kirillov, A. (2019). *Detectron2: A PyTorch-based modular object detection library*. Meta AI. <https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>
- Google Colaboratory. (2022). *Google Colab*. <https://research.google.com/colaboratory/faq.html?fbclid=IwAR3c8UOmyMfecniNYIchi8rZau0LFaWDLx1hDguHFO3owFpeoTxZbBMghgc>
- Gróf, A. (2018). *Vezetést segítő funkciók fejlesztése okostelefonra mély tanulás alapon*. Budapest University of Technology and Economics.
- He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2), 386–397. <https://doi.org/10.1109/TPAMI.2018.2844175>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2015*, 770–778. <https://doi.org/10.1109/CVPR.2016.90>
- Heller, M. (2022, September 16). *What is CUDA? Parallel programming for GPUs*. Info World. <https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html>
- Hui, J. (2018, March 28). *Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)*. Medium. <https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359>
- Introducing JSON*. (2022). <https://www.json.org/json-en.html>
- JavaTpoint. (2022). *Introduction to Dimensionality Reduction Technique*. <https://www.javatpoint.com/dimensionality-reduction-technique>
- JavaTpoint. (2022). *K-Means Clustering Algorithm*. <https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning>



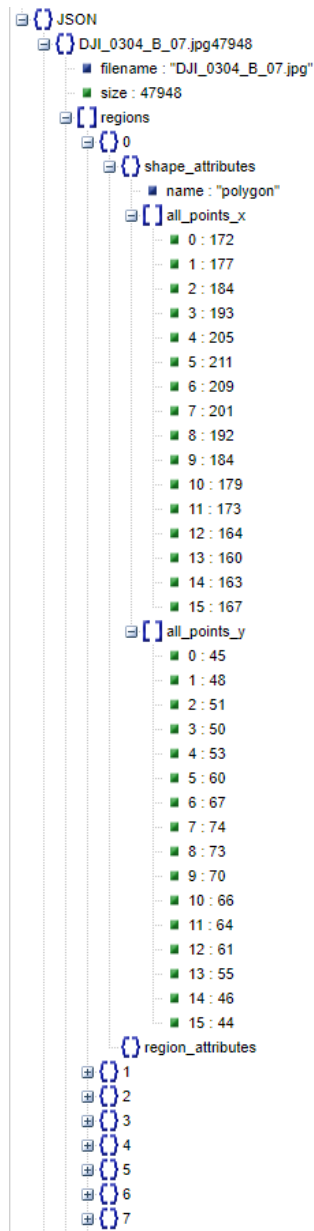
- JavaTpoint. (2022). *Regression Analysis in Machine learning*. <https://www.javatpoint.com/regression-analysis-in-machine-learning>
- JavaTpoint. (2022). *Supervised Machine learning*. <https://www.javatpoint.com/supervised-machine-learning>
- JavaTpoint. (2022). *Unsupervised Machine learning*. <https://www.javatpoint.com/unsupervised-machine-learning>
- Jayawardana, R., & Sameera, B. T. (2021). ANALYSIS OF OPTIMIZING NEURAL NETWORKS AND ARTIFICIAL INTELLIGENT MODELS FOR GUIDANCE, CONTROL, AND NAVIGATION SYSTEMS. *Research Gate, April*.
- JetBrains s.r.o. (2022). *PyCharm*. <https://www.jetbrains.com/pycharm/features/>
- Karlsson, A., & Rosin, G. (2020). *Deep Neural Networks and Semi-Supervised Learning*.
- Khandelwal, R. (2019, November 30). *SSD: Single Shot Detector for object detection using MultiBox*. Towards Data Science. <https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca>
- Kollár, C., & Nagy, B. (2021). A mesterséges intelligencia felhasználási lehetőségei az objektumfelismerésben. *Biztonságtudományi Szemle*, 3(1), 0–2.
- Kumar, A. (2013, June 13). *Machine Learning - Training, Validation & Test Data Set*. Data Analytics. <https://vitalflux.com/machine-learning-training-validation-test-data-set/>
- Kumar, A. (2022). Different Types of CNN Architectures Explained: Examples. *Data Analytics*. <https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/>
- Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- Lemley, J., Bazrafkan, S., & Corcoran, P. (2017). Deep Learning for Consumer Devices and Services. *IEEE Consumer Electronics Magazine*, 6(2), 48–56.
- Li, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollar, P., & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *Eccv, June*, 740–755.
- Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2019). Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, 128(2), 261–318. <https://doi.org/10.1007/s11263-019-01247-4>
- Martínez, M. L., Gallego-Fernández, J. B., & Hesp, P. A. (2013). Restoration of Coastal Dunes. *Springer Series on Environmental Management, January*. <https://doi.org/10.1007/978-3-642-33445-0>
- Meel, V. (2022). *What is the COCO Dataset?* <https://viso.ai/computer-vision/coco-dataset/>
- Microsoft Azure. (2022). *What is machine learning?* <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-machine-learning-platform>

- Morimoto, J., & Ponton, F. (2021). Virtual reality in biology: could we become virtual naturalists? *Evolution: Education and Outreach*, 14(7), 1–14. <https://doi.org/10.1186/s12052-021-00147-x>
- NumPy Developers. (2022). *Numpy documentation*. <https://numpy.org/doc/stable/user/whatisnumpy.html>
- Ongsulee, P. (2017). Artificial intelligence, machine learning and deep learning. *International Conference on ICT and Knowledge Engineering*, 1–6. <https://doi.org/10.1109/ICTKE.2017.8259629>
- OpenCV. (2022). *OpenCV*. <https://opencv.org/about/>
- Python Software Foundation. (2022). *Random — Generate pseudo-random numbers - Python documentation*. <https://docs.python.org/3/library/random.html>
- Python Software Foundation. (2023). *PyYAML · PyPI*. <https://pypi.org/project/PyYAML/>
- QGIS Documentation. (2022). *Documentation for QGIS*. <https://docs.qgis.org/3.22/en/docs/index.html>
- Radečić, D. (2020, April 30). *Google Colab: How does it compare to a GPU-enabled laptop?* . Towards Data Science. <https://towardsdatascience.com/google-colab-how-does-it-compare-to-a-gpu-enabled-laptop-851c1e0a2ca9>
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 1137–1149. <https://doi.org/10.1109/TPAMI.2016.2577031>
- Rizzoli, A. (2022, October 3). *Object Detection: Models, Architectures & Tutorial*. <https://www.v7labs.com/blog/object-detection-guide#h1>
- Rosebrock, A. (2017). Deep Learning for Computer Vision with Python. In *PYIMAGESEARCH*.
- Shah, T. (2017). *About Train, Validation and Test Sets in Machine Learning | by Tarang Shah | Towards Data Science*. <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>
- Shaip. (2022). *Image Annotation & Labeling for Computer Vision*. <https://www.shaip.com/blog/image-annotation-for-computer-vision/>
- Shinde, P. P., & Shah, D. S. (2018). A Review of Machine Learning and Deep Learning Applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1–6. <http://arxiv.org/abs/2104.12722>
- Soffer, S., Klang, E., Shimon, O., Barash, Y., Cahan, N., Greenspan, H., & Konen, E. (2021). Deep learning for pulmonary embolism detection on computed tomography pulmonary angiogram: a systematic review and meta-analysis. *Nature*, 11(1), 1–9. <https://doi.org/10.1038/s41598-021-95249-3>
- Srivastava, S., Divekar, A. V., Anilkumar, C., Naik, I., Kulkarni, V., & Pattabiraman, V. (2021). Comparative analysis of deep learning image detection algorithms. *Journal of Big Data*, 8(1). <https://doi.org/10.1186/s40537-021-00434-w>

- The Linux Foundation. (2022). *Torch — PyTorch documentation*.  
<https://pytorch.org/docs/stable/torch.html>
- The Linux Foundation. (2022). *Torchvision — Torchvision documentation*.  
<https://pytorch.org/vision/stable/index.html>
- Thukur, A. (2022, March 24). *How to Handle Images of Different Sizes in a Convolutional Neural Network – Weights & Biases*. <https://wandb.ai/ayush-thakur/dl-question-bank/reports/How-to-Handle-Images-of-Different-Sizes-in-a-Convolutional-Neural-Network--VmlldzoyMDk3NzQ>
- Ujjwal, K. (2016, August 11). *An Intuitive Explanation of Convolutional Neural Networks*.  
<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
- Van Puijenbroek, M. E. B., Nolet, C., De Groot, A. V., Suomalainen, J. M., Riksen, M. J. P. M., Berendse, F., & Limpens, J. (2017). Exploring the contributions of vegetation and dune size to early dune development using unmanned aerial vehicle (UAV) imaging. *Biogeosciences*, 14(23), 5533–5549. <https://doi.org/10.5194/bg-14-5533-2017>
- Vassányi, G. (2022). *Pontszerű jelek automatikus felismerése archív térképeken konvolúciós neurális hálózat használatával*.
- Verma, S. (2019, June 20). *Understanding different Loss Functions for Neural Networks*. Medium.  
<https://shiva-verma.medium.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718>
- Zenodo. (2023, January 2). *python-pillow/Pillow: 9.4.0* /. <https://zenodo.org/record/7498081#.Y-UyKnBMI2w>

## 8. Appendices

### 8.1 Appendix A



Appendix A: Structure of the exported annotation data, in JSON format

## 8.2 Appendix B

**Python (v3.10):** Python is a freely installable multi-platform general-purpose, very high-level, object-oriented dynamic programming language. It can be run on multiple operating systems and has a large number of free add-on libraries, making it very popular in the field of artificial intelligence and machine learning. Python was chosen because it has recently become one of the most popular languages for neural networks and deep learning development, thanks to the fact that it is one of the easiest programming languages to read and learn (Rosebrock, 2017).

**PyCharm (v2022.3):** PyCharm is an integrated development environment for programmers, designed for Python programming. It is developed by the Czech company JetBrains, and it is written in Java and Python. It runs on Windows, macOS and Linux. PyCharm IDE offers a complete set of tools for productive development for programmers working with Python. In addition, the development environment also provides high-level web tools in the Django framework (JetBrains s.r.o., 2022).

**Agisoft Metashape (v1.7.1):** Agisoft Metashape is a stand-alone software product that processes digital photogrammetry and generates 3D spatial data, which can be used in GIS applications. It runs on Windows, MacOS X and Linux, and there is no project size limit. “Its main functions are point cloud generation and classification, georeferencing from flight data or ground control points and it is also very well suited for orthophoto calculation. The software allows to process images from RGB or multispectral cameras, including multi-camera systems” (Agisoft Metashape, 2022)

**Google Colaboratory:** The detection program was run in a virtual environment using Google Colaboratory (Colab) which is a product of Google Research. It is mostly used for machine learning, data analysis and education in Python. The program code is in Jupyter Notebook format, which allows interactivity and visualization of results while the program is running. Although Colab is free to use, it has various usage restrictions (idle timeout, time limit) to ensure fair use of computer resources and to make the computers available to others as well. There are paid packages as well with less strict limitations and restrictions and/or access to more powerful computers. Through a Google service, Colab can be connected to Google Drive, which provides options for importing the files necessary for the operation of the program, as well as exports the trained models and the obtained results (Google Colaboratory, 2022). At the time of writing this thesis, the computer I was assigned had an NVIDIA Tesla T4 graphics processing unit, which meant that I could use the power of the GPU for object detection.

**VGG annotator (v2.0.12):** In this thesis, footprints are annotated using the VGG Image Annotator (VIA). This application is developed by Visual Group Geometry at the University of Oxford from 2016. It supports spatial and temporal annotation of images. It works as an offline application on the web without any installation, so it can be launched by lay users. It is written using HTML, Javascript and CSS and the whole software is only 400 kilobytes, which makes it easy to distribute, for example, via email. The interface is minimalistic and therefore simple and easy to use. It was released in 2017 under the BSD-2 license and has continuously been updated since then. Newer versions are already available,

currently Version 3. It is used by many industries for facial recognition (rectangle shape), arbitrarily shaped objects in electron microscope images (circle and polygon shapes), identification of century-old images (rectangle shape), etc. (Dutta & Zisserman, 2019). As it contains standard HTML components with CSS settings, many users will find the application familiar. It consists of 9000 lines of JavaScript code, none of which depend on any external library packages. It is completely open source and hosted by the Gitlab platform. It operates through a portal where suggestions for improvements are possible. It operates completely off-line (Dutta & Zisserman, 2019).

**COCO Common Objects in Context:** COCO is a dataset that is commonly used in machine learning computer vision tasks. It was developed by Microsoft and is filled with large-scale object recognition, segmentation, and labelling data. The purpose of computer vision is to recognize and localize 2D and 3D objects. For this purpose, this specific data set was created to classify different objects (Meel, 2022). The images collected are objects of everyday life in their naturalistic medium (T.-Y. Li et al., 2014). Features of COCO: Object segmentation with detailed instance annotations; 91 objects types; 2.5 million labelled instances in 328 000 images; Category detection; Instance plotting; Instance segmentation (Li et al., 2014). It contains objects like: 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light', etc. The COCO dataset contains certain key points, exactly 17. They are assigned three values, which are x, y (coordinates) and v (visibility). The dataset is free to use and downloadable since it runs under the Creative Commons Attribution 4.0 License.

**Detectron2 (v0.6):** Detectron as an open-source object recognition project was released in 2018, and Detectron2 is its rewritten version. Detectron2 is FAIR's next generation platform for object detection and segmentation. It was created by FAIR (Facebook AI Research) to provide state-of-the-art neural architectures with CUDA and PyTorch (PyTorch is a machine learning system developed by Facebook in Python) (Girshick et al., 2019). It uses the image segmentation (Semantic, Instance and Panoptic Segmentation) and object detection methods introduced in the previous chapters (Fast R-CNN, Faster R-CNN, Mask R-CNN, RetinaNet, RPN, R-FC, DensePose, Cascade R-CNN) (Girshick et al., 2019).

Other libraries and dependencies relevant to the use of Detectron2:

- PyYAML (v5.1): “YAML is a data serialization format designed for human readability and interaction with scripting languages. PyYAML is a YAML parser and emitter for Python.” (Python Software Foundation, 2023)
- CUDA (Compute Unified Device Architecture) (v11.6(cu116)): “CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on its own GPUs (graphics processing units). CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation.” (Heller, 2022)
- Torch (v1.13): “The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serializing of Tensors and arbitrary types, and other useful utilities”. (The Linux Foundation, 2022a)

- Torchvision: “The package consists of popular datasets, model architectures, and common image transformations for computer vision.” (The Linux Foundation, 2022b)
- NumPy (v1.22.4): “NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.” (NumPy Developers, 2022)
- JSON: “JavaScript Object Notation (JSON) is a standard text based format for representing structured data based on JavaScript object syntax.” (*Introducing JSON*, 2022)
- OpenCV (Open-Source Computer Vision Library): “An open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.” (OpenCV, 2022)
- Random: “This module implements pseudo-random number generators for various distributions.” (Python Software Foundation, 2022)
- Detectron2 utilities: A coherent package to aid visualisation. Includes modules that are useful for building models or training code, such as the logger (to retrieve verbose information while the code is running) or the visualizer (to display detection results). (*Detectron2 Documentation*, 2023)

**QGIS (v3.22.4):** QGIS (Quantum Geographical Information System) is an open-source geographic information system launched in 2002. Its great advantage is that it has a fast and easy-to-use graphical user interface (GUI). It supports both raster and vector formats and is suitable for analysing and editing spatial information. A lot of additional libraries and plugins can be added which are written in C++ and Python. GDAL (Geospatial Data Abstraction Library) as a raster vector converter and GRASS (Geographic Resources Analysis Support System) which includes a number of GIS functions are part of the QGIS repertoire (QGIS Documentation, 2022).

### 8.3 Appendix C

Batch Size	Learning Rate	Iterations	RoI Batch Size	Loss	Diff %	AP	Diff %	Training time (s)	Diff %
2	0.00025	1000	128	1.129		31.22		351	
1	0.00025	1000	128	1.231	9.03%	29.56	-5.33%	179	-49.00%
4	0.00025	1000	128	1.151	1.95%	30.67	-1.79%	740	110.83%
8	0.00025	1000	128	1.153	2.13%	31.03	-0.63%	1591	353.28%
16	0.00025	1000	128	1.156	2.39%	31.15	-0.23%	3312	843.59%

Correlation of BS with:		
Loss	AP	Time
-0.2852	0.5270	0.9998
	moderate	very strong

Batch Size	Learning Rate	Iterations	RoI Batch Size	Loss	Diff %	AP	Diff %	Training time (s)	Diff %
2	0.00025	1000	128	1.129		31.22		351	
2	0.0001	1000	128	1.33	17.80%	28.18	-9.74%	366	4.27%
2	0.00005	1000	128	1.518	34.46%	22.34	-28.46%	374	6.55%
2	0.001	1000	128	0.97	-14.13%	28.24	-9.56%	355	1.14%
2	0.002	1000	128	0.795	-29.58%	29.79	-4.59%	366	4.27%

Correlation of LR with:		
Loss	AP	Time
-0.8982	0.4016	-0.0683
very strong	moderate	

Batch Size	Learning Rate	Iterations	RoI Batch Size	Loss	Diff %	AP	Diff %	Training time (s)	Diff %
2	0.00025	1000	128	1.129		31.22		351	
2	0.00025	500	128	1.284	13.73%	28.56	-8.54%	181	-48.43%
2	0.00025	100	128	1.821	61.29%	15.78	-49.46%	36	-89.74%
2	0.00025	1500	128	1.009	-10.63%	29.62	-5.15%	546	55.56%
2	0.00025	2000	128	0.854	-24.38%	28.84	-7.64%	731	108.26%
2	0.00025	1020	128	1.185	4.96%	32.01	2.52%	366	4.27%

Correlation of Iterations with:		
Loss	AP	Time
-0.9245	0.6146	0.9998
very strong	strong	very strong

Batch Size	Learning Rate	Iterations	RoI Batch Size	Loss	Diff %	AP	Diff %	Training time (s)	Diff %
2	0.00025	1000	128	1.129		31.22		351	
2	0.00025	1000	64	1.188	5.23%	29.9	-4.23%	339	-3.42%
2	0.00025	1000	32	1.182	4.69%	29.82	-4.48%	325	-7.41%
2	0.00025	1000	256	1.147	1.59%	29.56	-5.32%	403	14.81%
2	0.00025	1000	512	1.154	2.21%	30.04	-3.80%	493	40.46%

Correlation of ROI Batch Size with:		
Loss	AP	Time
-0.40876	-0.11409	0.99840
		very strong