Automatic animal detection using unmanned aerial vehicles in natural environments

Rik Smit
June 28, 2016

Master's thesis
Artificial Intelligence
Department of Artificial Intelligence
University of Groningen, The Netherlands

First supervisor:
Dr. M.A. Wiering (Artificial Intelligence, University of Groningen)

Second supervisor:
Dr. F.N. Martins (Federal Institute of Education, Science and Technology of Espirito Santo, Brazil)

university of groningen

faculty of mathematics and natural sciences

Abstract

In the past decade, small unmanned aerial vehicles have become increasingly popular for remote sensing applications because of their low costs, and easy and fast deployment. Together with the development of light-weight imaging sensors, these UAVs have become valuable tools for monitoring and analyzing large areas from above. In the Netherlands, the development of agricultural and livestock sectors plays an important role. The use of an unmanned aerial vehicle allows visualization of the crowns of cultures and monitoring livestock in a large area, which increases the ability of interpretation and diagnosis from the data collected, thus contributing to increase agricultural productivity. While quickly collecting large amounts of imagery data from the UAVs is becoming more straightforward, analyzing these data is still mostly a laborious demanding manual task. Major issues for object detection are annotating large amount of training data and finding correct feature descriptors and classifiers. A general framework for detecting objects in natural environments using UAVs is developed in this research. The object detection method can be bootstrapped with minimal expert annotation of data that are collected using an affordable commercial UAV. Different machine learning techniques are analyzed to find which maximize the object detection success. The resulting object detector can be trained using active learning techniques to reduce manual labeling effort, and allows for harvesting detected objects to increase the output performance.

Contents

Chapter 1

Introduction

In the past decade, the development of small low-cost unmanned aerial vehicles (UAVs) has increased the interest of their use for remote sensing applications. UAVs have proven to be a valuable alternative to satellite imagery and helicopter or air plane monitoring due to the low costs, and easy and fast deployment. They are now used for tasks that before were economically not profitable because of the high operating costs of traditional aerial vehicles. Together with the development of light-weight imaging sensors, these UAVs have become valuable tools for monitoring and analyzing large (rural) areas from above. Figure 1.1 shows some examples of (small) UAVs that could replace traditional aerial vehicles for different types of tasks.

(a) UAV for arctic used for research.

(b) Ascending Technologies Pelican research UAV.

(c) DJI Phantom 2, a popluar commercial UAV.

Figure 1.1: Several (small) UAVs that are used for various applications like research (left and middle) but also for commercial purposes (right).

There is an increasing number of fields of application where UAV setups are used from crowd control [8] to automatic detection of forest fires [13]. UAVs can obtain imagery for rangeland monitoring and create orthophotos by mosaicking recorded video at near real-time [23]. Together with remote sensing techniques this can either complement or even replace ground-based measurements [10]. The application of geospatial techniques and sensors to identify variations in the field and to deal with them using alternative strategies is called precision agriculture (PA). PA aims to increase

the efficiency and productivity of the agricultural sector. It is becoming increasingly important for farmers to reduce costs and increase yields [21]. The use of robots, and also UAVs has a key role in the development of PA. For example, new methods are found for classification of natural vegetation using UAV imagery which are used to discriminate between weeds of interest and background objects [7]. Tools like this can be used for selective weed treatment to reduce for example the use of herbicides [11].

Another application is livestock detection and counting [18]. Farmers with large herds covering fast amounts of grounds may use these methods to monitor their livestock at low expenses. A more idealistic application however is the use of these tools for wildlife monitoring, where different species of (endangered) animals can cover fast areas of rural territory. Poaching and other natural changes of environment still endanger wildlife at various locations is the world. UAVs are therefore becoming a helpful tool for acquiring valuable data in this field [9, 5]. See Figure 1.2 for a visualization of detecting animals in an image.

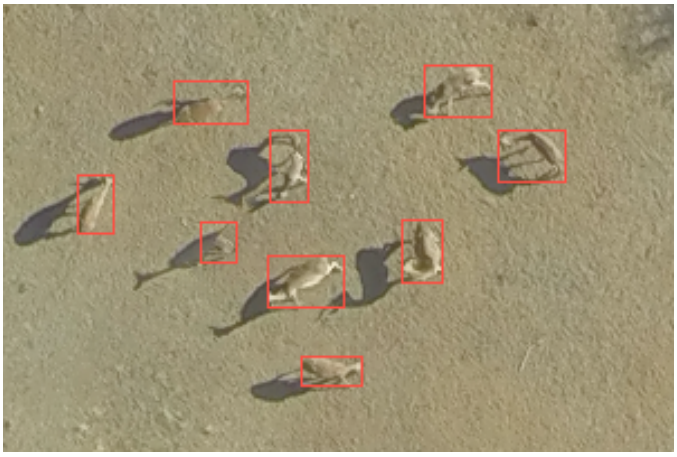


Figure 1.2: Detecting animals (surrounded by the red bounding box) can be useful for counting a population of a herd in remote environments.

In the Netherlands, the development of agricultural and livestock sectors plays an important role. Despite its small area, the Netherlands is an important producer of flowers, milk (and its derivatives), among other agricultural products. In fact, it is the second most important country in exports of agricultural needs in the world [1]. The use of an unmanned aerial vehicle allows visualization of the crowns of cultures and monitoring of a large area, which increases the ability of interpretation and diagnosis from the data collected, thus contributing to increase agricultural productivity.

While quickly collecting large amounts of imagery data from the UAVs

[1] https://www.cbs.nl/nl-nl/publicatie/2016/23/internationaliseringsmonitor-2016-tweede-kwartaal

is becoming more straightforward, analyzing these data is still mostly a laborious demanding manual task. Efforts have been made to combine human computing and machine learning to make sense of large (aerial) datasets for specific tasks like disaster response [14], but some of the major issues for object detection still are 1) annotating the large amounts of training data, and 2) finding the correct features and classifiers. The goal of this thesis is to propose a general framework for detecting and inspecting (natural) objects in rural environments using UAVs. The object detection method should be bootstrapped with minimal expert annotation and be able to generalize over various target objects. The focus will be on monitoring rural areas due to the significantly increasing role of robots in precision agriculture and wildlife monitoring [21, 9, 5]. Several studies show how UAVs can be used in combination with object detection methods to monitor rural areas and livestock [18, 7].

Most research on detecting objects like animals in a natural scene is based on a perspective similar to that of humans, i.e. a horizontal perspective. This means that the majority of the datasets that are available for research in this field contain images and videos that are made from this perspective. With the detection of animals on the ground from a UAV in the air, a top-down perspective is used. This means that datasets traditionally used for object detection are not adequate. Few datasets are available that use this top-down perspective, and each of these datasets have their own advantages and disadvantages. For this research we chose to build a new dataset that is specific to the needs of this project. In chapter 2 the focus is on the acquisition of this dataset. With the use of a recent model UAV there are recordings made from animals in a top-down perspective from a relatively low altitude. The animals that are used as subjects are typical livestock in the Netherlands. The UAV that is used is a popular commercially available and affordable quadcopter. This type of drone is easy to operate without much prior experience in a flying model aircraft. This means that making recordings for the dataset will be relatively easy. The dataset acquisition chapter describes how the dataset is built from the start of recording the videos to annotating the videos to train and test detectors. Other researchers should be able to use this as a reference to build their own dataset, or extend the dataset that is used for this research. The used hardware is described, as well as the flight method that is used to record the videos. Special focus is on the labeling method, which is usually a labor-intensive task when it comes to dataset acquisition. For this purpose the Vatic video annotating tool [20] is used. This software package allows for collaborate labeling of videos with minimum effort.

In chapter 3 the methods are described that are used for this research to detect animals with a UAV using computer vision (CV). The annotated dataset is used as a starting point to develop a generic framework that allows for the detection of objects in a natural environment. Although for

specific tasks there are usually specialized solutions that perform the task optimally, the aim is here to develop a solution that can be translated for multiple purposes in a generic matter. For example, when detecting animals in a natural environment one could think of an infra-red sensor that detects the body-heat of animals which makes them easy to distinguish from the background [15]. This solution would then not work for detecting natural objects with the same temperature as the ground, like vegetation. For this thesis, the example of detecting cows by distinguishing them from the ground is used with as input (color) video recordings. Several Computer Vision techniques are compared to explore the framework's performance with different detection methods. First several types of features descriptors are compared for extracting features from the annotated dataset cutouts. One of the more basic types of features descriptors is the color histogram. With this method the occurrence of pixel color values in an image cutout is used to build a histogram. The values of the constructed histogram are used as a descriptor for the samples that are compared. A more complex feature descriptor that has become popular for use in object detection problems is the histogram of oriented gradients (HOG) [3]. This method analyzes local regions of an image sample and builds a histogram based on the occurrence of gradient orientations in these regions. Figure 1.3 shows an example of these orientations in an image. A third feature descriptor is constructed by combining both the Color Histogram and HOG features. The hypothesis is that the combination of these exploits both the benefits of using color information from the Color Histogram, as well as the gradient information from the (gray-scale) HOG features.

The feature vectors are input for a classifier that is then trained to distinguish background samples from objects. Several popular algorithms for

[2]Image taken from `http://scikit-image.org/docs/dev/auto_examples/plot_hog.html`

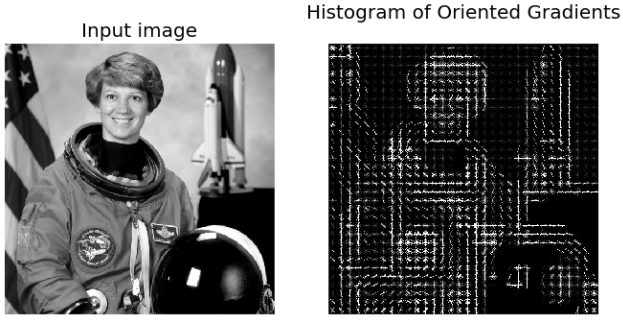


Figure 1.3: Input image (left) and a visualization of the Histogram of Oriented Gradients (right)[2].

classifiers are compared that have proven to be useful for other object detection tasks. First the widely used k-Nearest Neighbors (k-NN) algorithm is used to train a classifier. Later the performance of this classifier is compared to Support Vector Machines (SVM) with different kernel functions: the linear function kernel and the radial basis function kernel. When a classifier is trained to discriminate between different object samples (e.g. background and foreground samples), a detector can be constructed. The sliding window approach is used to focus the classifier on portions of a video frame one at the time, until the frame is completely analyzed. One of the challenges is to build a detector with decent performance while being trained on a limited dataset. A harvesting technique is used to retrain the detector on new input data as soon as more samples are available. This technique requires an extra step during the detection process where a (human) annotator reviews labels that are given by the detector.

To compare the different types of feature descriptors and classifiers, experiments are setup as described in Chapter 4. The first experiment is setup to test the performance on cutout object samples from the dataset. The goal of this experiment is to test a classifier that is capable of discriminating between animal samples and background samples. For a more practical application, a second experiment is setup that should give an indication of how an animal detection system may be developed. Here a trained detector is used to detect objects in a video stream as would be provided by a camera mounted under an UAV. An extra experiment is conducted to show how active learning can help in training a classifier with a minimal amount of training samples. A final experiment is an improvement where the harvesting technique is applied to the detector to increase the performance of a detector that is trained on minimal training data. The results of the experiments are provided in Chapter 5. These results give an indication of the differences in performance of the different types of methods that are used. Finally in the discussion of Chapter 6 we look back at the work that has been done in this research. A more critical view is given on the project results, and how these relate to comparable work. Also there is a focus on how the results can be used in a practical context like the counting or tracking of animals.

For this project the following research question is posed: can a (low-end) UAV automatically detect animals like cows in a natural environment? As part of this question we ask the following: which of the popular feature descriptors (color histogram, HOG) and classifiers (k-NN, SVM) that are used for object detection maximize the results? Also, how can active learning and harvesting improve the object detection process for this task?

Chapter 2

Animal Dataset Acquisition

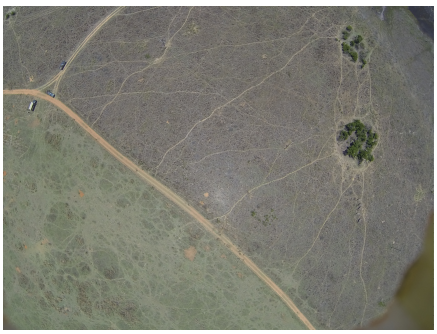
Few labeled datasets are available with aerial images or videos of natural objects like animals in natural environments. Some available datasets are the Dutch UAS Dataset 001 [19] and the Verschoor Aerial Cow Dataset [18] with recordings of cows in a meadow made by a UAV.

The Dutch UAS dataset contains video frames with rhinos, zebras, rangers and cars in a wildlife reservation. The type of animals and the recording location of the videos makes this a unique dataset. The recordings are shot from an airplane-type UAV with a camera that allowed for high-resolution and high-quality videos. The dataset contains annotations where the object locations in the video frames are marked by a bounding box. The boxes are however not (yet) labeled with the type of object that is within the bounding box. Due to the high flight altitude not all animals are easy to recognize. See Figure 2.1a for an example frame from this dataset.

The Verschoor Aerial Cow dataset contains recordings of cows in a meadow, made by a quadrotor UAV (an Ascending Technologies Pelican[1]). Videos are shot with a GoPro HERO 3 camera attached to the UAV. The videos are recorded with a bird's-eye view with an angle to the ground as shown in Figure 2.1b. This dataset includes labels with the location of the cows in each video frame. The location of each cow is denoted with the coordinates of the bounding box surrounding that cow.

The datasets described above each have their own advantages and disadvantages for use within this project. The recordings from the Dutch UAS dataset are interesting because multiple objects and different backgrounds are present in a frame. The downside is however that the recordings are shot from such a high altitude that the objects in each frame are pictured with only a small amount of pixels. The flight altitude in the Verschoor Aerial Cow dataset is much lower and more representative for the goal of this project, but the videos are shot with different angles respective to the ground (bird's-eye perspective). Therefore, for this research a new dataset

[1] See http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-pelican/



(a) Dutch UAS dataset

(b) Verschoor Aerial Cow dataset

Figure 2.1: Sample frames from two datasets that are comparable to the dataset created for this project. The image on the left is from the Dutch UAS dataset, and the right image is from the Verschoor Aerial Cow dataset.

is recorded with video recordings at a top-down perspective at a relatively low altitude. This results in higher resolution objects than recorded in most other UAV datasets. The recordings are manually labeled using a labeling tool specialized for the task of labeling video recordings. The downside of building a new dataset is that it takes a lot of time from recording videos with a UAV to labeling the objects in the recordings. Doing so however allows us to build a large enough dataset that is specified to our specific needs for this project. Recordings are made from a top-down perspective at an altitude that results in objects (animals) with a decent resolution.

2.1 Recording videos with an unmanned aerial vehicle

Building a new dataset requires video recordings made with an (Unmanned) Aerial Vehicle. The relatively recent popularity of small UAVs with video capabilities in the last years, and the low amount of research projects done that required videos similar to those used for this project, required us to shoot our own recordings. The small UAVs with video capabilities that became more popular the last couple of years are ideal for shooting the recordings for this project. Many of these UAVs are relatively cheap compared to using manned helicopters or airplanes, and lightweight digital cameras that are attached under the UAVs allow for high-resolution video recordings.

With the increasing popularity of small UAVs in the Netherlands, flight restrictions have become an increasing issue. As of October 1st 2015 however, new rules for remotely piloted aircrafts up to 4 kilograms are implemented. With the exception of specific areas such as in controlled airspace and crowded areas, both commercial as private pilots are allowed to fly their UAVs without the need of a special certificate.

2.1.1 Hardware

A DJI Phantom 3 Advanced UAV is used for recording the videos. This quadrotor UAV can be commercially purchased and is easy to operate for most people, even those with no experience in flying (model) airplanes. The drone is controlled manually from the ground using a controller with an attached mobile device like a tablet or mobile phone (see Figure 2.2). A life stream of the Phantom's on-board camera is shown on the mobile device in real-time. The life stream is established using the built-in Wi-Fi capabilities of the controller and the UAV and thus does not require an external router within range. The location of the UAV is also shown on a map on the mobile device, which therefore requires GPS capabilities and an Internet service to synchronize the map details. The GPS data together with the Internet service prevent the UAV to be flown in restricted areas around for example airports and military locations.

The Phantom is out of the box equipped with an HD-camera that is capable of recording videos at 60fps with a full-hd resolution (1920 by 1080 pixels). The resolution is important because it allows for flying at a greater altitude while still having a decent amount of pixels-per-object ratio. Flying on higher altitudes may be necessary when animals are easily startled from objects flying over them. The camera is attached to a gimbal with 3-axial stabilization that keeps the camera steady in most flight conditions, resulting in stable footage during the flight. The gimbal pitch can be controlled remotely, allowing the camera to be facing forward, down or any position in between. For this project the pitch is set to 90°, meaning that it is facing down to the ground for a top-down perspective when recording the video.

Figure 2.2: DJI Phantom 3 setup. Left the Phantom including the camera. Right the controller with the mobile device attached.

The Phantom 3 has a limited flight time of approximately 20 minutes on a fully charged battery. This is enough for our purpose to cover at least an entire meadow with animals. At an open field the range of the UAV, i.e. the distance between the controller and the quadcopter, is up to 2 kilometers. In the Netherlands this is usually enough, as most meadows are only a couple of hundred meters wide. See Table 2.1 for the Phantom 3 specifications.

Table 2.1: DJI Phantom 3 Advanced specifications. See Appendix B for a more detailed specification list as provided by the manufacturer.

Weight	1280g
Size including propellers	689 mm
Maximum speed	up: 5m/s, down: 3m/s, horizontal: 16m/s
Hover accuracy	vertical: ±10cm, horizontal: ±1m
Video resolution	1920 x 1080
Video fps	60
Flight time	approx. 20 minutes (single charge)
Maximum range	2km (open range)

2.1.2 Flight method

The flight time of the Phantom is long enough to easily record entire meadows and all the animals inside on a single battery charge. During tests it became clear that different animals behave differently when a UAV flies over. Young cows for example are more likely to start moving when a UAV flies over than older cows. Also different types of sheep (Drenthe Heath sheep versus the Schoonebeker Heath sheep) behave differently. As a result, for most recordings in the dataset a flight altitude of 30 meters is taken as a compromise between having a large enough distance from the animals to not startle them, and having a small enough altitude for video recordings with a high enough pixel resolution per animal.

During the entire recording time the UAV is operated manually. For each recording session, the same process of operation is used to reduce the risk of making mistakes and to ensure a consistent result. The UAV takes of at a safe distance from any animals that might be around. During the entire time of flight the operator makes sure that the UAV is within sight. Before flying close to animals, the operator makes sure that the UAV flies at a high enough altitude to not disturb the animals. Built-in safety mechanisms make sure that the UAV will not drop down due to low power: when battery charge is running low the operator will receive a warning. If the battery runs critically low, the UAV will fly back home to its home point automatically

(this is usually the point where it took off at the beginning). In some circumstances the connection between the UAV and the controller might be lost. In this case the UAV will also return to its home point automatically. When returning to its home point, the UAV will first rise to a preset altitude (e.g. 50 meter) to make sure it will not crash into trees or buildings along its path.

When making a recording, the operator will try to record most of the animals in the meadow at least once. Flying in straight lines back and forth will ensure most of the ground space is covered by the camera with minimal flight time. Figure 2.3 shows the steps that are taken during this process of operation.

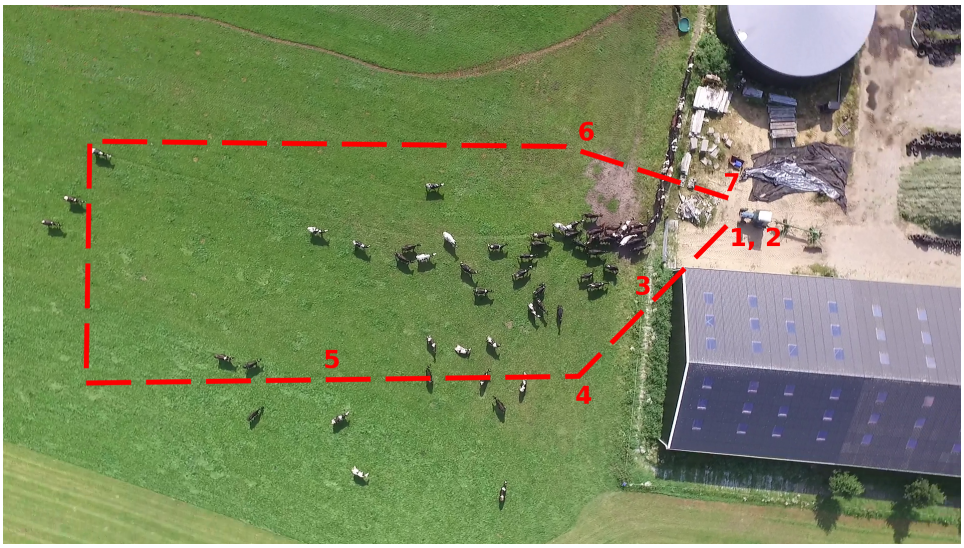


Figure 2.3: Example flight path for making a dataset recording. 1) Start the drone and takeoff, 2) Rise to required altitude (30 meters), 3) Fly to start of recording area, 4) Start camera recording, 5) Fly over recording area until most present animals are recorded at least once, 6) Stop camera recording, 7) Return to home.

2.1.3 Recorded videos

During recording the video stream is saved in the SD-card that is inserted in the Phantom. After recording, the SD-card is taken out to transfer the videos to a hard-drive so that they can be used for off-line processing. Several videos are recorded, all with cows in meadows at different locations. Not all videos are recorded on the same day so that light illumination will vary among the videos. See section 2.3 for more details.

2.2 Video labeling

The process of labeling involves marking in the recordings where interesting objects (animals) are located. The result of labeling a recording is that for each frame, all the animals in that frame are marked with a surrounding bounding box. The bounding box can later be used to cutout the object from the frame it is in. Before labeling the videos undergo a preprocessing step. During this step the videos are cropped in time: only the parts of the video are taken where the UAV is flying on the same correct altitude. This will make sure that the same objects are roughly of the same size. Also uninteresting parts (without animals) at the beginning and end of the video are removed as these parts will add little to no valuable information to the final dataset. A simple script using FFmpeg[2] is built for this purpose (see appendix A).

2.2.1 Labeling with Vatic

Labeling videos for the first time is a manual and time-consuming process so a good labeling tool is required for this work. Several labeling tools are available to label video recordings with meta-data and to annotate objects within the videos. For this project the choice fell on the labeling tool Vatic [20]. This piece of software is a labeling tool produced at the University of California, Irvine which provides an interface to manually label video datasets.

The labeling process using Vatic begins with importing the video recording that the user wishes to label. This video is then preprocessed by Vatic: if needed the video is rescaled to a different resolution for faster processing, after which individual frames are extracted. The extracted frames are uploaded to a web server. Labelers can then label these frames using a web interface. This web interface has several advantages:

- Multiple people can access the labeling tool at the same time
- Videos can be labeled by different people
- Labelers can access the tool from anywhere they want

During the labeling process the labeler will draw a bounding box around each object he (she) finds in the frame. Objects that are partly obstructed by other objects or that are partly outside the frame can be marked as such. During the labeling process the labeler scrolls through the video frame by frame. At the end of the labeling process all the frames in the video are labeled. Figure 2.4 shows the Vatic interface with a single labeled frame. One of the most time-saving features of Vatic however is providing functionality

[2] http://www.ffmpeg.org

that requires the labeler to only label certain frames. Frames in between those labeled frames are automatically labeled by Vatic. This process works on the assumption that objects in frames move more or less in a predictable (linear) motion with respect to the frame. For example, for this project the UAV mostly flies over a field in a straight path. The location of cows in the video frame will then change in a straight line as well. If a cow is labeled only in frame x_i and frame x_{i+10}, then the location of the cow in frames x_{i+1} to x_{i+9} can be interpreted with reasonable precision. A large amount of unlabeled frames in between two labeled frames may however result in an increasingly larger error due to irregular motion of both the UAV with respect to the ground, as well as (irregular) motion of objects on the ground. In Vatic the automatically added labels can directly be reviewed by the labeler, and adjusted when needed.

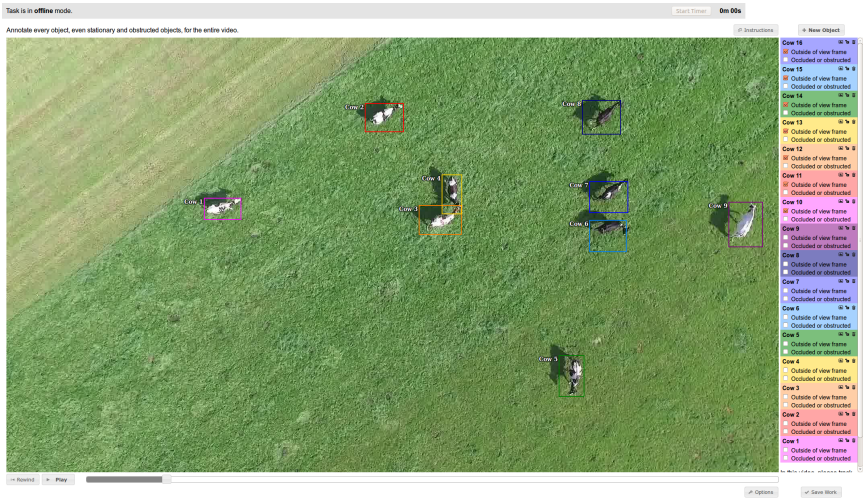


Figure 2.4: Labeling with Vatic. A bounding box is drawn around each object (cow). Unique objects receive unique identifications, as denoted by the different colors in the image.

Unique animals will all receive their own unique identifier such that later each bounding box can be tracked back to a unique animal. Knowing the individual animals is important when splitting a dataset into train and test sets where we want unique animals to appear only in one of these sets (either test or train). During or after labeling, the labeler can save this process. All labels are stored to the database on the server where Vatic runs, and the labeler can continue working on annotating the video later by loading the saved process.

2.2.2 Labeling output

After all the frames in a video recording are labeled, the annotated data are exported to a simple *.txt* file that can be easily interpreted by scripts. This output file describes all the labeled objects in the video, with the correct frame and the coordinates of the bounding box in that frame. Each entry in the output file contains the following relevant data:

- Object ID (e.g. each unique cow gets its own identifier)
- Coordinates in the frame (Xmin, Ymin, Xmax, Ymax)
- Frame number
- Label (e.g. 'Cow')

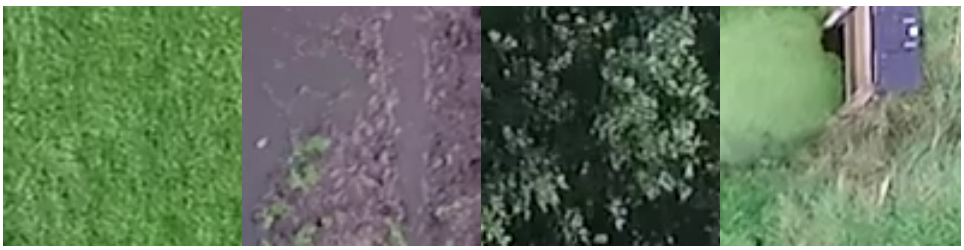
2.3 Recording statistics

Several video recordings are made of cows in different meadows. From most of the animals that are labeled there are multiple positive samples available since they usually appear in more than one frame within a video. From these frames also the negative samples are extracted. Negative examples can be any part of a frame that contains no (positive) object. All samples are cutout from the video frames with a fixed cutout size (usually 100 by 100 pixels). Section 3.1 describes in more detail how the cutouts are extracted from the frames. Figure 2.5 shows some examples of cutouts that are made based on the manually labeled samples (in the case of positive samples) and automatically extracted samples (in the case of negative samples). The statistics are shown in table 2.2.

Table 2.2: Statistics of the recorded and annotated datasets.

	Video ID	Length (s)	Unique objects	Pos samples	Neg samples
1	DJI_0005_cut_233-244	11	10	37	225
2	DJI_0007_cut_22-65	43	82	475	2094
3	DJI_0081	22	10	50	1100

(a) cow **(b)** cow **(c)** cow **(d)** cow



(e) grass **(f)** mud **(g)** trees **(h)** culvert

Figure 2.5: Some examples of positive cutout samples (a-d) and negative cutout samples (Figure e-h).

Chapter 3

Detecting Animals with a UAV Using Computer Vision

Objects on the ground are detected by a UAV using the on-board camera and Computer Vision (CV) algorithms. A framework is built that uses a video recorded with a UAV as input for a detector that is given the task to locate the different objects (animals) in the recording. The detector will run off-line, i.e. after the UAV recorded the videos and landed safely on the ground. Depending on the practical application however, it might be required to locate objects in real-time while the UAV is still in the air. For this project the focus is on building the foundations of a framework that later might be implemented for on-board processing when that is needed. It is expected that in the near future the processing capabilities of affordable UAVs will increase, while it is now still difficult to find UAVs that are capable of processing CV tasks in real-time on-board. After the recorded videos are downloaded from the UAV, the analysis process starts. Detecting objects automatically in the recordings can be done on a commercial grade laptop on-site, or afterwards on a different location. The detection of animals in the environment should give an insight on the location and distribution of animals at a particular location. Further practical uses (not part of the method in this research) are animal counting and tracking. A requirement for the detector is that it is trained on a limited labeled dataset.

The limitations on the used hardware, like commercial grade cameras and laptops, pose limitations to what CV tools can be used in the detection framework. These limitations are taken into consideration during the design and implementation of the framework. For example, some machine learning algorithms require fast amounts of processing power that can not be provided by a standard laptop. Also the limitations of the camera module of the UAV (medium resolution images with noise for example) require robust feature descriptors.

3.1 Animal and background cutouts

Labeling the video recordings with Vatic results in detailed information about the location of interesting objects (animals) in the frames of these videos. During the labeling process, all objects in each frame are annotated with a bounding box around that object. We could think of a separation between the foreground (the objects) and the background (everything that is not an object). The bounding box is then a rough estimation of what is the foreground and what is the background. Naturally, this estimation comes with an error as the bounding box is always a rectangle, opposed to the arbitrary shaped animals. The coordinates of the bounding boxes in each frame is used to cutout the foreground objects. In Vatic, each bounding box is associated with a unique identifier that is used to distinguish unique animals from each other. This is important when the classifier is later trained on different animals. During the manual labeling process the labelers place the bounding box tightly around the object. For the feature extracting and classification process the sample cutouts are presented with all the same dimensions. Instead of making a cutout directly based on the sample bounding box, the bounding box is extended to a predefined size and aspect ratio which depends on the size of the objects. A fixed aspect ratio of 1:1 is chosen, while size depends on the video resolution and flight altitude. For example, for the cow dataset recorded at an altitude of 30 meters, the dimensions of the cutouts are 100x100 pixels.

A classifier is trained on both positive (foreground) and negative (background) samples. Each sample is a squared image with the size of a typical object in a video frame. All sample images will be of the same size. Some noise is present in the form of background pixels in the positive sample images. In order for the classifier to be trained properly, enough of these samples need to be extracted. For the positive samples this is a fixed amount determined by the size of the labeled dataset. There are two factors that determine the amount of these samples in the dataset:

1. The amount of unique animals that were present in each video recording in the dataset.

2. The amount of samples that are taken from each unique animal. From each available frame, at most one sample can be taken per animal.

The amount of negative samples will generally be much higher than the amount of positive samples in order to provide a diverse variety of background types. Because of the nature of the recorded areas, much of the background will simply be green land. It is expected that the detector will have little trouble distinguishing for example cows from green land because of the large difference in color (usually white and black versus green). The challenge is to also cope with other types of background like mud, farm

equipment and other objects that happen to be present. The background in frames from the dataset are not explicitly labeled in Vatic. This can however be derived automatically by subtracting the labeled foreground objects. What results are all the pixels that are not associated with a foreground object. Depending on the saturation of foreground objects in the frames, many more negative samples can be extract than there are positive samples. Algorithm 1 shows the process of extracting the negative samples automatically given a list of positive samples in each frame. Figure 3.1 visualizes an example of what parts of a video frame are cutout for negative and positive samples.

Algorithm 1 Finding negative samples in a video recording given the location of the positive object in the frames.

```
neg ← []                                        ▷ List of negative samples
for every n-th frame do
    pos ← positiveSamplesInFrame(n)
    repeat
        bb ← randomBoundingBoxInFrame(n)
        if noOverlapWith(bb, pos) then
            neg+ = bb
        end if
    until enough neg samples
end for
```

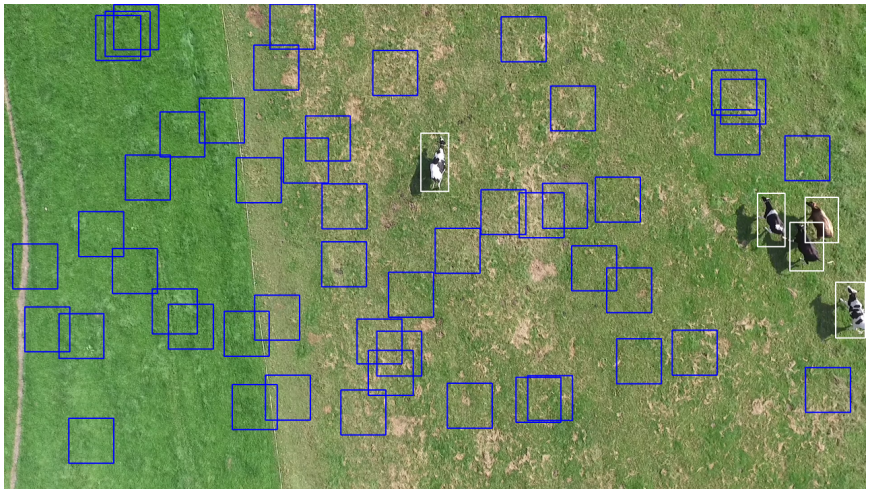


Figure 3.1: Visualization of which portions of a video frame are cutout. The white bounding boxes show the positive cutouts. The blue bounding boxes are the negative cutouts that should never overlap with the positive cutouts.

3.2 Feature extraction

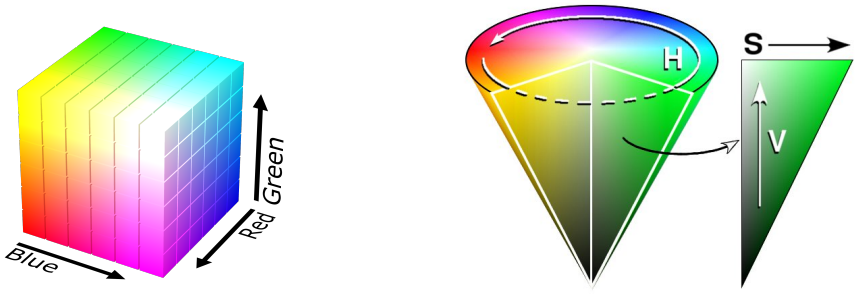
The detector classifier will be trained on features that are extracted from the positive and negative cutout samples. The quality of these features determines for a large part the performance of the detector. There are many different types of feature extractors that can be used for object detection in images. For this thesis two popular feature extractors are compared: the Color Histogram (ColHist) and the histogram of oriented gradients (HOG) [3]. Also a third feature extractor is formed that combines these two (ColHist-HOG). All of these three extractors transform an input sample (an image cutout) into a descriptive feature vector that can be used as input for the classifier. The architecture of the Animal Detection Framework is built to easily adapt to different feature extractors. If one would like to experiment with different image feature extractors, this can be easily done in the programming code due to the modular design of the framework.

3.2.1 Color histogram

The color histogram feature extraction method analyzes the pixel color values from an image. The underlying concept assumes that these color values provide valuable information about the subject in the image. One clear example is the difference between cows as foreground objects, and the grass as background objects. There will be a clear distinction between the colors of the cow (usually white and black in this dataset) and the green grass. These pixel color values can be transformed to a feature vector that can be used as input for the classifier. For this transformation a histogram is generated. Each bin in the histogram represents a range of values within a color space channel.

HSV color space

The choice of color space is important when using the color histogram as a feature extractor. The color space is a description of a color how humans can see it to a representation that is useful for computers that handle digital values. For many applications (outside computer vision) the RGB color space is used. This color space describes a color based on their red (R), green (G) and blue (B) color values (see figure 3.2a). The values of each of these channels taken together specify the final color as humans will see it. While these values are enough to describe all possible colors, the values by itself say nothing about the intensity of a color, or its saturation. If for example the intensity of a color changes, then all the three channels are updated. This makes it hard to specify specific ranges of values that describe a color independent from its perceived intensity. As an alternative, the HSV color space is used (see figure 3.2b). Here colors are described based on their hue (H), saturation (S) and value (V).



(a) RBG color space represented by a cube

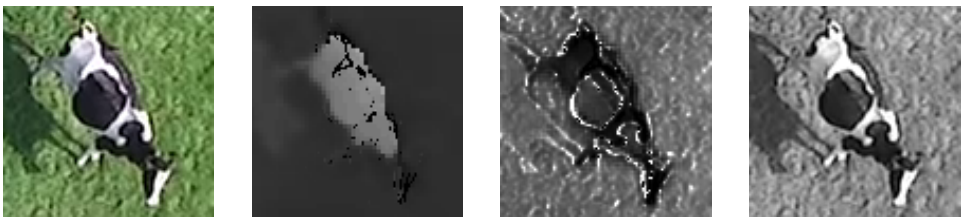
(b) HSV color space representad by a cone

Figure 3.2: Two different color representations: RGB versus HSV[1].

For computer vision applications this color space can be more useful as the value parameter can be observed independently from the other parameters. When comparing images with different brightness values, the value parameter can be isolated while focusing on the hue or saturation. Figure 3.3 gives a visualization of the different channels for a cutout samples as used in our dataset.

A common method for creating a color histogram is to build a 3D cube with on each of the axes one of the channels. Color values within a specific range (around a point inside the cube) are then stacked to build the histogram. An alternative method is to build the histogram for each channel separately. The reduces the size of the resulting feature vector (from b^c to bc, where b is the number of bins and c the number of channels that are used), at the cost of information that is lost in the process. The latter method is used here to reduce the processing time for the detector in the experiments.

[1]Images taken from Wikipedia.org.



(a) original **(b)** hue **(c)** saturation **(d)** value

Figure 3.3: The channels of a colored image (a) in HSV space can be visualized individually as shown in images b-c. These images show how each of the channels is affected by the different colors.

The process of creating a ColHist feature vector from a color image is as follows:

1. Convert color space of the input image from RGB to HSV if needed.
2. For each channel take the values of each pixel for that channel.
3. Build a histogram for each channel based on these values.
4. Concatenate the histograms for the final feature vector.

Parameters

Two parameters are tuned when experimenting with the color histogram feature extractor. The first parameter is the size of the bins. The size of the bins will represent the resolution of the histogram. The smaller the bin size the higher the obtained resolution of the representation: each bin will only represent a small range of values. A larger bin size is expected to result in more generalizing features, while using smaller bins might result in overfitting on specific color values. When decreasing the bin size, (accidental) peaks in the value range will have a larger influence in the model than when a larger bin size is chosen. When the peaks are flattened out by using a larger bin size, the influence off these peaks will decrease, and thus generalize the model.

The second parameter specifies which channels (Hue, Saturation, Value) are used when generating the feature vector. The choice of the used channels may affect the performance of the detector. We want the detector to be robust against different brightness values. Excluding the Value parameter of the HSV color space ensures features that can be used for a detector will not look at brightness values.

3.2.2 Histogram of oriented gradients

A feature extractor that has become popular in object detection tasks in images is the histogram of oriented gradients (HOG) [3]. The HOG feature descriptor analyzes local regions of an image and builds a histogram based on the occurrence of gradient orientations in these regions. The method has been used widely in object detection tasks with good overall performance [3, 24, 4, 22]. Opposed to the ColHist feature extractor, HOG uses gray-scale images as input.

The first step to create features using HOG is thus to transform the color sample cutouts to a grayscale color space. From the grayscale image the gradient values are calculated. First the intensity data of the image I are filtered with two kernels for gradient computation:

$$I_x = I * K_x \tag{3.1}$$

$$I_y = I * K_y \tag{3.2}$$

Where:

$$K_x = [-1, 0, 1] \tag{3.3}$$

$$K_y = [-1, 0, 1]^T \tag{3.4}$$

Now the magnitude $|G|$ of the gradient can be calculated:

$$|G| = \sqrt{I_x^2 + I_y^2} \tag{3.5}$$

And the orientation θ of the gradient:

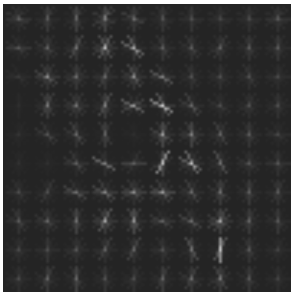
$$\theta = arctan\frac{I_y}{I_x} \tag{3.6}$$

The found gradients for each pixel within a cell are gathered in bins, where each bin accounts for a specific orientation range. The amount of orientations for the bins to use is one of the parameters that can be tuned. To increase the robustness of the feature descriptor, it should handle variations in illumination and contrast within an image sample. Therefore, they are divided into blocks where each block is described by several cells. The cells that are used for each block may overlap. The final feature vector is then generated by concatenating the cell histograms of all the block regions. The process of transforming an input sample to a final feature vector is finally as follows:

1. Convert color image to grayscale.
2. Calculate gradients over the pixels values.
3. Bin the oriented gradients within each cell.
4. Group cells together in blocks.
5. Concatenate cell histograms from all the block regions.

Figure 3.4 visualizes the oriented gradients from a grayscale input image sample of a cow.

(a) Grayscale input image



(b) Visualization of HOG orientations

Figure 3.4: From the grayscale input image (a), the orientations are calculated. These orientations can be visualized as shown in image (b).

Parameters

The HOG feature descriptor requires several parameters to be tuned in order to obtain the best performance:

window_size
The size of the window that is used as input. E.g. 100 by 100 pixels. When detecting objects the size of this window should be roughly the size of the object that needs to be detected.

orientations
The number of orientations that are analyzed for each cell. E.g. 8 orientations. More orientations mean that each bin will represent a smaller range in degrees of the gradient.

pixels_per_cell
The amount of pixels that are in each cell. E.g. 32 by 32 pixels. Choosing more pixels per cell will reduce the resolution of the feature descriptor.

cells_per_block
The number of cells per block. E.g. 2 by 2 cells per block.

3.2.3 Combining features

A combination of the color histogram and histogram of oriented gradients is used to benefit from both the color features of ColHist and the spatial features of HOG. First the feature vectors are calculated from the ColHist and HOG separately. After normalization these feature vectors are concatenated into a final feature vector. The parameters that need to be tuned for this combined feature descriptor are the same as for the individual feature descriptors.

3.3 Classifiers

Three classifiers are compared: k-Nearest Neighbors (k-NN), a Support Vector Machine (SVM) with a linear function kernel (SVM-Linear) and a Support Vector Machine with a radial basis function kernel (SVM-RBF).

3.3.1 k-nearest neighbors

A basic k-NN classifier is evaluated as an initial detector that should indicate which kind of performance can be obtained for detecting objects. The k-NN algorithm is a non-parametric method that can be used for (supervised) classification. K-NN as a classifier is known to be a simple but powerful method for classification problems in a wide range of applications. It is trained on an initial set of samples N, where each sample has its calculated features x and associated class label c (object or non-object). The output y of the classifier given a new (unlabeled) sample is the expected class of that sample. The class y of a new sample, represented by its feature vector $\vec{x}$, is determined based on a majority vote among the k closest training samples $N_k(\vec{x})$ to that sample:

$$y = f(\vec{x}) = \arg\max_{c} \sum_{\vec{x}_i \in N_k(\vec{x})} \mathbb{I}(y_i = c) \tag{3.7}$$

The distance between two sample feature vectors a and b is determined using a distance function $d(a, b)$. A popular distance metric which is used here is the Euclidean distance:

$$d_E(a, b) = \sqrt{\sum_{i=1}^{N} (a_i - b_i)^2} \tag{3.8}$$

Parameters

Although k-NN is a non-parametric method, the number of samples k that every new input sample is compared with, can be tuned for the best performance. A larger k will usually result in a more generalizing classifier (at the potential cost of precision), while a smaller k can lead to higher precision at the potential cost of overfitting. In the experiments one of the goals will be to find an optimal setting (k value) for the best performance.

3.3.2 Support vector machine

Support vector machines (SVMs) are supervised learning models that can be used as non-probabilistic (binary) classifiers. SVMs are used in a wide variety of classification tasks and are known for having a good performance compared to traditional methods like k-NN. Training an SVM results in a model

(a) Large C **(b)** Small C

Figure 3.5: Data points in the feature space with the decision boundary (solid line) and the margin (dashed lines)[2]. New data points are classified based on which side of the decision boundary they appear.

that is built using a training set with samples of two classes (object/non-object). The samples are represented as points in space (using their respective features), where the model tries to map the points in such a way that a decision boundary surrounded by a margin can separate the points of different classes from each other. Samples on the margin are called the support vectors, hence the name support vector machine. For optimization, the largest suitable margin is found using the following optimization problem:

$$\min_{\vec{w},b} \quad C\sum_{i=1}^{m}\xi_i + \|\vec{w}\|^2 \tag{3.9}$$

$$s.t. \quad y_i(\vec{w}\vec{x}_i + b) \geqslant 1 - \xi_i \tag{3.10}$$

$$\xi_i \geqslant 0, \quad i = 1, 2, \ldots, m \tag{3.11}$$

Where $\vec{w}$ is the weight vector of the decision boundary, ξ the slack variable for a sample and b the bias value. If $0 < \xi \leq 1$, then the sample is between the margin and the correct side of the decision boundary. If $\xi > 1$, then the sample is at the wrong side of the decision boundary, and thus incorrectly classified. The penalty parameter C of the error term is tuned in the experiments to maximize the performance of the classifier. A small C results in a large margin while a large C narrows the margin (or makes it a hard margin when $C = \infty$). This effect is visualized in Figure 3.5.

[2]Images taken from `http://scikit-learn.org/stable/auto_examples/svm/plot_svm_margin.html`

New samples are placed in the model as points, where the location (compared to the decision boundary) of the points determines the assigned class (object/non-object). The decision rule is:

$$sign\left(\sum_{i=1}^{m} \alpha_i \kappa(\vec{x}, \vec{x_i}) y_i + b\right) \tag{3.12}$$

Here $\kappa(\vec{x}, \vec{x'})$ is the kernel function as described below. One of the benefits of an SVM model (compared to k-NN) is that a large training set can be described with a relatively simple function (representing the decision boundary). New samples only need to be compared with this function, instead of the entire initial set of samples.

Linear function kernel

An SVM with a basic linear kernel is used, which is represented as

$$\kappa(\vec{x}, \vec{x'}) = \vec{x}^T \vec{x'} \tag{3.13}$$

Based on the input samples, a hyperplane is computed that separates the two classes (object/non-object) with the largest possible margin. Typically, with more complex problems like ours, the input data are not linearly separable. A soft-margin is therefore used with a loss function that is minimized.

Radial basis function kernel

In addition to the linear function kernel SVM, an SVM with a non-linear kernel function is tested. The popular radial basis function (RBF) kernel function is used:

$$\kappa(\vec{x}, \vec{x'}) = \exp\left(-\frac{\|\vec{x} - \vec{x'}\|^2}{2\sigma^2}\right) \tag{3.14}$$

Like with the linear kernel function, a maximum-margin hyperplane is fitted, but now in a transformed feature space. In addition to the penalty parameter C, also the kernel coefficient γ is tuned in the experiments. In the formula above this coefficient is represented using

$$\gamma = \frac{1}{2\sigma^2} \tag{3.15}$$

3.4 Sliding window approach

The detectors are trained on features from images with a fixed size (e.g. 100x100 pixels). Each of these images contains either an object (filling that image) or not. Since objects can be located anywhere in the (much larger) video frame, the entire frame needs to be inspected by the detector. A

sliding window is used that moves over the frame step by step. At each step the cutout from the window is fed to the detector. To cover the entire frame, the window should move pixel by pixel until all possible locations are analyzed. Since the detection of each window is computationally expensive, the window moves with multiple pixels at each step. This reduces the total time of processing each frame at the cost of precision. An optimal step size reduces the total time of processing with little loss of the final performance accuracy.

3.4.1 Suppression of detected objects

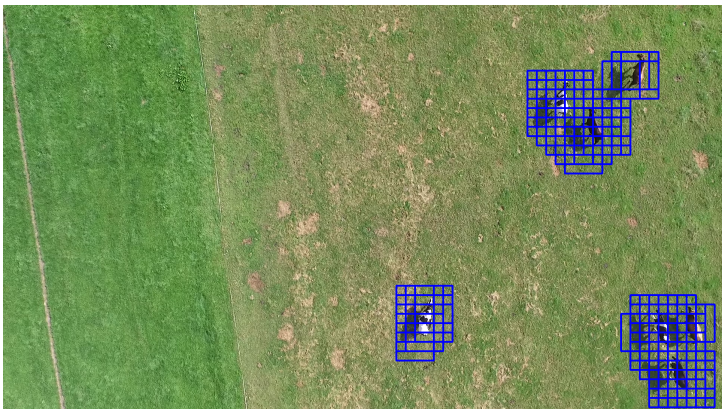
Because of the sliding window approach, multiple positive detections may be found for a single object. As the window moves step by step over the video frame, at some point an object might be detected, while at the next step the same object is detected. When the detections are visually analyzed, it is clear that there are margins around each of the objects when there are multiple detections for that object. Non-maximum suppression (NMS) is used to reduce these margins by choosing the detections that are expected to cover the object the best (i.e. the detection located in the center). This method has already provided good results in for example human detection using histograms of oriented gradients [3] and other object detection tasks [4, 6]. Figure 3.6 shows how the suppression algorithm reduces the amount of detections that are found by a detector in a single frame.

3.5 Learning while recognizing

When only a limited training set is available, a classifier will be trained with limited annotated samples. In general, the performance of the classifier will increase when more training data is available. One method to improve the results is by adding more samples while recognizing objects at the same time. This method is called active learning and is widely researched for use in different applications [2, 12]. Learning while recognizing requires feedback from an expert (human) annotator. There are several scenarios for performing active learning: membership query synthesis, stream-based selective sampling and pool-based sampling [17]. The latter, shown in Figure 3.7, is used in this case: first, a classifier is trained on an initial set of training samples. This initial set is typically very small. After this, the classifier will recognize objects in an unlabeled set of samples. From the results of this step, some of the new samples are sent to a human labeler, that annotates these samples. The samples that are sent to the labeler are those that are expected to provide the most valuable information for the classifier. The classifier is then retrained with the initial training set together with the new samples. This process of recognizing and retraining on labeled samples

is repeated until the performance of the classifier is assumed to be good enough. The complete steps are as follows:

1. Train classifier on an initial dataset
2. Start active learning iteration:
   (a) feed unlabeled samples to the classifier
   (b) take most valuable samples for the classifier
   (c) annotate those samples
   (d) retrain the classifier including the new labeled samples
3. repeat the iteration until stop criteria is met.



(a) Detections

(b) Suppressed detections

Figure 3.6: The top image shows the detections as initially provided by the detector. In the bottom image these detections are shown after applying the suppression algorithm. Note that the smaller rectangles/squares are the result of overlapping detections.

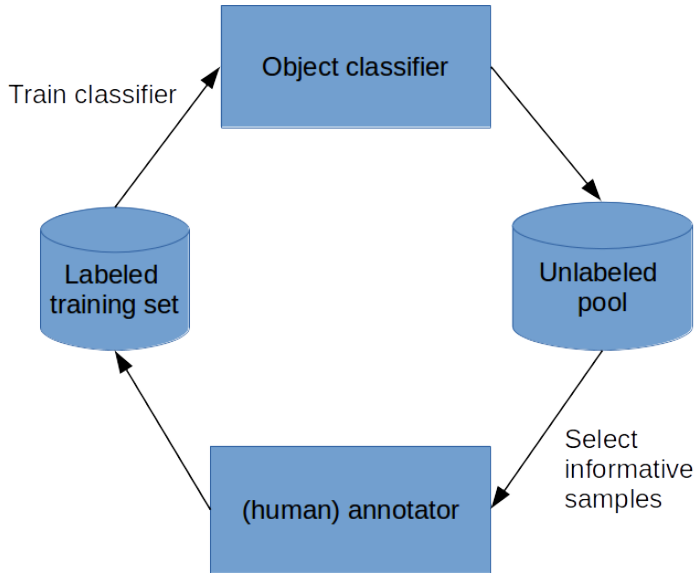


Figure 3.7: The active learning cycle. An object classifier is trained on the available labeled training set. Each cycle new training samples are added to this set by means of human annotating.

It is important how the samples are chosen that are likely to be most valuable for the classifier. The confidence of the classifier for a sample to be of a certain class is used as an indication. The classifier will return this confidence based on the probability that the sample belongs to a certain class. Those samples where the probabilities for the two classes are not far from each other are expected to be most valuable for the classifier. Since it is not too uncertain which class the sample belongs to, it can learn from the human labelers feedback.

3.6 Harvesting detection results

While the object detector tries to find in the video stream the objects, and the objects only, it is possible that parts of the video frames are recognized as objects while they are actually part of the background (false positives). The goal of harvesting is to have a human annotator that verifies detected objects from the object detector while it is running. After each frame (or set of frames), the found objects are presented to the human annotator. This annotator will then verify for each of the found objects whether it is indeed an object. The feedback can then be used to retrain the detector. The process is visualized in Figure 3.8.

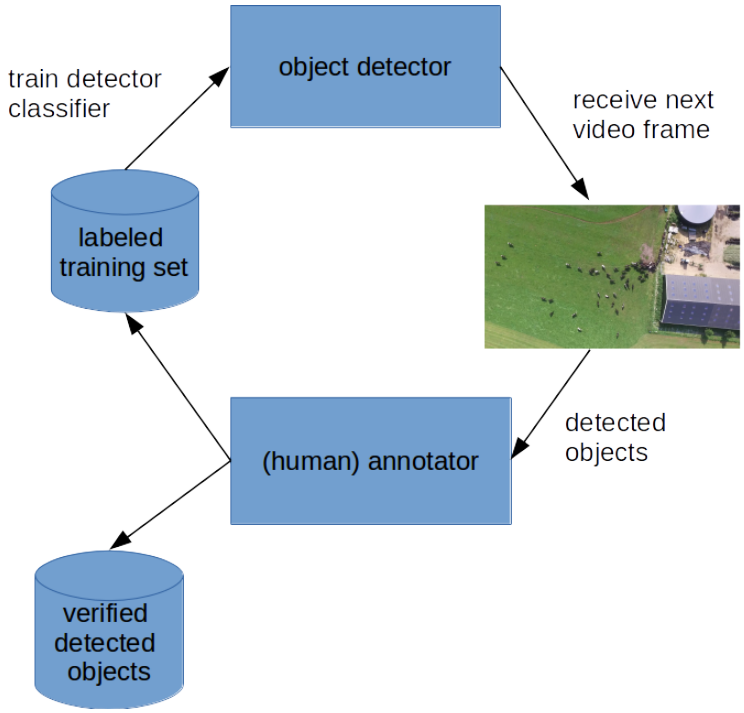


Figure 3.8: The harvesting method. The classifier of the object detector is trained on the available labeled training set. Each detection process in a frame results in detections that are verified by an annotator. The verified detection are passed to the detector for retraining.

3.7 Framework implementation

For most of the experiments it would have been sufficient to build some scripts specific for that task. For this research however we have chosen to build a framework that can be easily used for a wide range of similar tasks with only minor changes in the code.

Python is chosen as the main programming language. The language is easy to learn and supports a wide range of modules, some of which are dedicated to machine learning or computer vision. For most of the image processing and machine learning the scikit-learn [16] and the OpenCV [1] modules are used. The major advantages of using these packages is that they provide optimizations for a wide variety of machine learning and image processing processes, are widely tested for validity by a large community, and are easy to use. This will save both computational time (many functions are implemented in a lower-level language than Python) and implementation time (no need to reinvent the wheel). The format of data that is parsed to and returned by components like classifiers and feature descriptors is standardized when possible. This allows for easy extension of the framework with other (types of) classifiers and feature descriptors.

The main functions of the framework are:

- Reading and interpreting labeled data
- Processing videos
- Calculating image features based on labeled data
- Training and testing of classifiers
- Run experiments on existing datasets
- Detecting objects in new datasets

Chapter 4

Animal Detection Experiments

Several experiments are conducted to explore the capabilities of the constructed framework, and to find out how the methods that are used in the framework perform. Each time the labeled dataset as described in Chapter 2 is used. The first experiment is an exploration on what features and classifiers work the best for distinguishing objects from the background. The results are later applied to a streaming detector where objects are found in a video recording. Next to these experiments, the application of active learning and harvesting are tested.

4.1 Animal recognition in segmented images

A classifier is trained on the cutouts from the labeled dataset to give an initial performance indication. The experiment will show how well a trained classifier can distinguish objects from non-objects. The goal of this experiment is to find the (optimal) performance for different feature descriptors and classifiers.

- Features descriptors
    1. Color histogram
    2. Histogram of oriented gradients
    3. Combined
- Classifiers
    1. k-Nearest Neighbors
    2. Support vector machine (linear function kernel)
    3. Support vector machine (radial basis function kernel)

4.1.1 Dataset splits

The input for the classifier are the positive and negative sample cutouts from the dataset. The positive samples are the manually labeled objects from the dataset, while the negative samples are the automatically generated cutouts that contain no objects. The dataset is split into a training set and a test set for performance testing. K-fold cross-validation is used with two test methods:

1. inter-set splits
Each individual subset is split into folds. There are several subsets with samples available, each based on a single recording and all with different individual objects. Every subset is split into folds. The detector is trained on the folds of a single subset, leaving one fold for testing. The average results over all the subsets is taken as an overall performance indicator. The goal of this test is to find a base performance level of the detector when using training and test data from a single recording.

2. cross-set splits
Each subset is regarded as a fold. Using this method the detector is trained on several complete subsets (the folds), while one subset is used for testing. The goal of this test is to explore the performance when detecting objects on a completely novel dataset which should give an indication of how robust the detector is.

4.1.2 Feature descriptor parameters

The first step is to find what parameters work the best for the different feature descriptors. A default k-NN classifier is used (with $k = 5$). The performance of this classifier is measured multiple times while using different feature descriptors with different parameters each time. Different parameters are tested depending on the used feature descriptor.

Color histogram
The following parameters are tested for the color histogram feature descriptor:

- Which channels are used (Hue, Saturation and/or Value)
- The bin size used for generating the histogram

Histogram of oriented gradients
For the HOG feature descriptor, the following parameters are tested:

- The amount of orientations used for each cell
- The amount of pixels per cell
- The number of cells per block

Combined feature descriptor
Finally, for the combined feature descriptor the tested parameters are chosen from the best parameters that were found from the results of the individual tested feature descriptors. For each of these feature descriptors there are two optimal sets of parameters found: one for the test on individual subsets, and one for the test where all subsets are taken together (see Section 4.1.1). This result in 4 sets of parameters that are combined to form the tested parameters for the combined descriptor. This method is used to greatly reduce the amount of to be tested parameters.

4.1.3 Classifier parameters

Three classifiers are analyzed to find what parameters work best for those classifiers. Each of them is trained and tested on features that are built using the different feature descriptors and their optimal parameters that were found before. Each classifier requires different parameters to be tuned.

k-NN
The number of nearest neighbors k is tuned

SVM (linear function kernel)
The penalty parameter C is tuned

SVM (radial basis function kernel)
Both the penalty parameter C and the kernel coefficient γ are tuned

4.2 Animal detection in video streams

After recognizing objects in segmented image samples, the task of detecting objects in videos streams is explored. The goal of this experiment is to demonstrate the usage and performance of the detector in a situation where objects need to be found from a video stream taken with a UAV.

First the detector is trained using the same method as in the previous experiment of recognizing animals in segmented images. The best classifier from that experiment (the SVM with RBF kernel) is used, in conjunction with the fast but good performing color histogram feature detector. In each run two subsets are chosen for training, while a third is used for testing.

This is done to see how the detector performs on unknown video stream input.

Because of the altitude and flight speed of the UAV, individual animals are in view for a longer period of time, usually several seconds. It is therefore not needed to analyze every frame of the video (which is recorded at usually 25 fps) to find every object at least once. Every n-th frame is used, and the rest of the frames is discarded. n is chosen to be large enough while still making sure that objects are at least once visible in the stream of chosen frames.

A sliding window is moved over each chosen frame. The size of the window is the same as the size that is used to train the detector. The window moves with a step size at least half of the window size such that windows overlap each other. Using this technique objects in the frame are more likely to at least once be completely surrounded by the borders of one of the sliding windows.

From every window the features are calculated which are then passed to the trained object detector. The detector outputs whether there is an object in the frame window or not.

The result is a series of object detections for each frame. A small step size of the moving window may likely result in multiple detections for each object. Non-maximum suppression is therefore applied on the detection, with the goal to eliminate as many unnecessary detections as possible.

The complete steps are as follows:

1. Train detector on an initial dataset
2. Take every n-th frame of the video stream
3. Apply sliding window on each chosen frame
4. Analyze every sliding window image:
   (a) extract features from image
   (b) feed features to the trained classifier
   (c) analyze classifier result: object yes/no
5. Suppression is applied to windows where objects are detected.

4.2.1 Performance measurement

The performance of the object detector is measured by how close the detected objects match with the ground truth. This is done by analyzing how the pixels from the detections overlap with the pixels of the ground truth. The first measurement is the *overlapping window ratio*:

$$ratio = \frac{O}{\sqrt{DT}} \tag{4.1}$$

Here the number of overlapping pixels O are compared with the number of pixels from detections D and the number of pixels from the ground truth T. With the same variables, the precision and recall as used by [18] for example can also be computed as follows:

$$precision = \frac{O}{D} \tag{4.2}$$

$$recall = \frac{O}{T} \tag{4.3}$$

4.3 Learning while recognizing

The goal of this experiment is to show how the concept of active learning can be applied to an object classifier to reduce the amount of samples that need to be manually labeled. The subset with the most unique objects (based on video DJI_0007_cut_22-65) is used for this experiment. This set contains enough positive samples to split it into a reasonable amount of batches for training, opposed to the other subsets that have relatively few positive objects. In this experiment the labeling of samples in each iteration of the active learning process is simulated, so there is no real human annotator needed. The labels from the dataset as described in Chapter 2 are applied automatically.

The dataset is split such that 10% of the samples are used for testing the classifier to obtain its score. From the remaining 90%, 10% is used as initial training set for the classifier, the rest will be the unlabeled pool. First a classifier is trained on the samples from the initial training set. Then, the unlabeled pool is fed to to the classifier. For each sample in the unlabeled pool, the classifier will return a confidence on how certain it is that the sample is an object or not. The samples that the classifier is the least certain about are fed to a human annotator and removed from the unlabeled pool. The amount of samples that are annotated in each iteration is about the same as the initial training set, which means there are 9 iterations until the entire unlabeled pool is empty. The annotated samples are then added to the training set, and the classifier is retrained. The process stops when there are no more samples left in the unlabeled pool. After each iteration the classifier is tested against the test set to obtain the classification score. The results are compared to a classifier that is retrained after each step on randomly picked samples.

4.4 Harvesting detection results

For harvesting detection results, an extra step is introduced after detecting objects in video frames. A human annotator is provided with the results of the detector and verifies samples that are detected by the labeler. In this

experiment the human annotation process is simulated just like in the active learning experiment. The goal is to improve the quality of the results from the detector. False positives (non-objects that are recognized as objects) pollute the results and should be filtered using this manual step.

The method of object detection is used as in the experiment of animal detection in video streams. Instead of keeping the object detector running until all frames are processed, now after each processed frame the results are presented to the annotator. This annotator verifies the results of the detector. The classifier of the detector is then retrained on the labeled samples in addition to the training set that was initially used.

Chapter 5

Results

5.1 Animal recognition in segmented images

In this experiment the feature descriptors and classifiers are analyzed, where the dataset consists of cutout samples from objects and non-objects. In each case the score is defined as the mean accuracy given by the classifier unless otherwise stated. This accuracy is calculated by the percentage of samples that are correctly labeled by the classifier.

5.1.1 Feature descriptor parameters

First the optimal scores are found for each of the feature descriptors when used on a default classifier as described in the previous chapter. These are shown in Figure 5.1. The best results are found in the inter-set split test. This should not be a surprise as the classifier is here trained and tested on samples that were extracted from the same video recording. This means that there is less variation in light conditions for example. For the cross-set tests the classifier is trained and tested on samples from different video recordings. The results show that the color histogram as feature descriptor gives the best results together with the combined feature descriptor.

Table 5.1 shows what are the optimal parameters for each feature descriptor depending on what type of dataset split is used. Although the results for the histogram of oriented gradients seem quite consistent between the types of dataset splits, there is some difference in what parameters are the best for the color histogram.

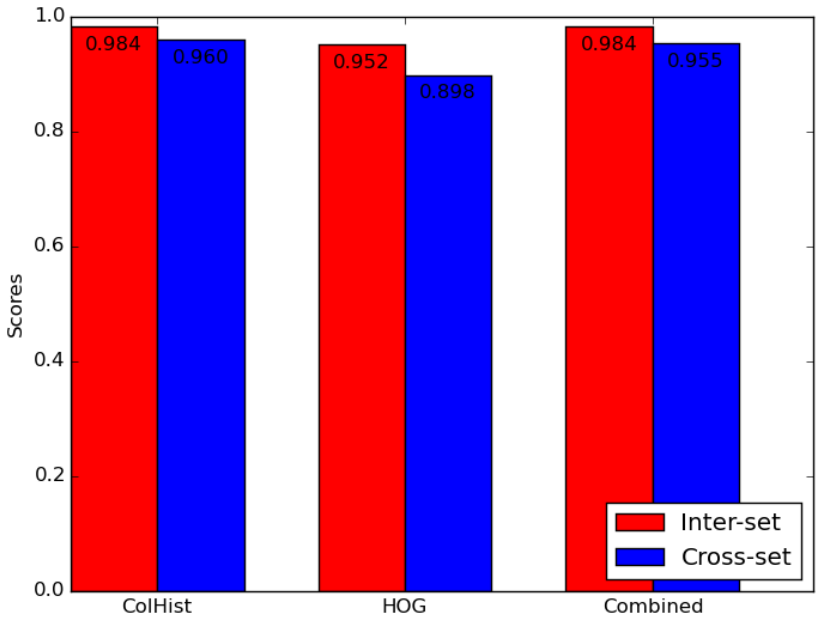


Figure 5.1: Scores by feature descriptor and test type. The inter-set test results are obtained by doing k-fold testing on each subset individually, and then taking the average. For the cross-set test results each subset is regarded as a fold. The k-nearest neighbors classifier with $k = 5$ is used in each test.

Table 5.1: The scores and the found optimal parameters by feature descriptor and test type. The inter-set test results are obtained by doing k-fold testing on each subset individually, and then taking the average. For the cross-set test results each subset is regarded as a fold. A complete overview of the scores for the various parameters can be found in Appendix C, tables C.1, C.2 and C.3.

Feature descriptor	Test type	Score	Parameters
Color histogram	inter-set	0.984	channels: [hue, saturation, value] bin size: 8
	cross-set	0.960	channels: [saturation] bin size: 32
HOG	inter-set	0.952	orientations: 4 pixels-per-cell: 32 cells-per-block: 3
	cross-set	0.898	orientations: 4 pixels-per-cell: 32 cells-per-block: 3
Combined	inter-set	0.984	orientations: 4 pixels-per-cell: 32 cells-per-block: 2 channels: [hue, saturation, value] bin size: 8
	cross-set	0.955	orientations: 4 pixels-per-cell: 32 cells-per-block: 3 channels: [hue] bin size: 32

5.1.2 Classifier parameters

With the optimal parameters for the feature descriptors found, now the three different types of classifiers are tested. The cross-set test is used for training and testing the classifiers each time. The score of the best classifiers are shown in Figure 5.2. In each case the support vector machine with the radial basis kernel function gives the best scores. Details on the scores and the used parameters are shown in Table 5.2.

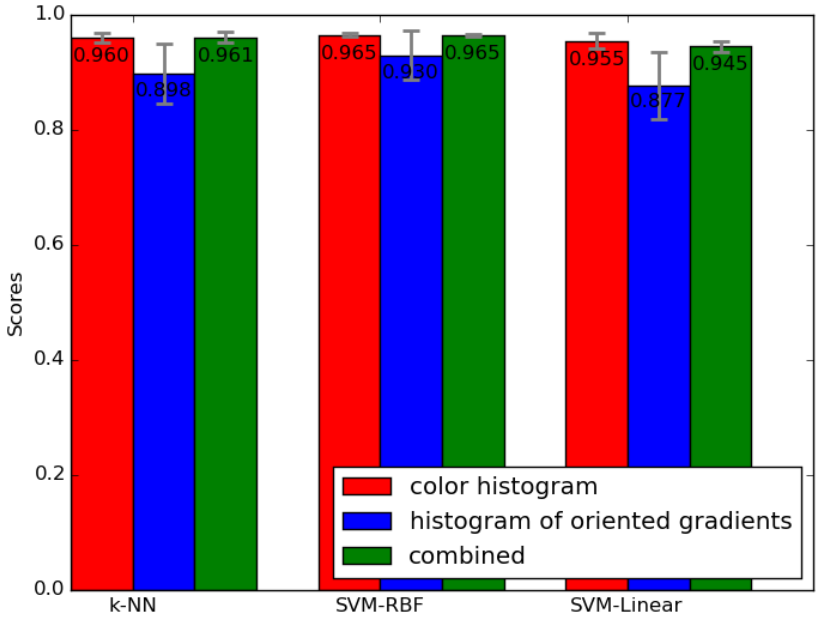


Figure 5.2: Classifier scores when run with the optimal parameters. Each classifier type is run with the three different types of feature descriptors that are tested using the cross-set split method.

Table 5.2: The scores and the found optimal parameters by used classifier and feature descriptor. Each time the cross-set split method is used. A complete overview of the scores for the various parameters can be found in Appendix D, tables D.1, D.2 and C.3.

Classifier	Feature descriptor	Score	Parameters
k-NN	ColHist	0.960	k: 3
	HOG	0.898	k: 1
	Combined	0.961	k: 3
SVM-RBF	ColHist	0.965	C: 1
			γ: 100
	HOG	0.930	C: 3
			γ: 1000
	Combined	0.965	C: 1
			γ: 100
SVM-Lin	ColHist	0.955	C: 8192
	HOG	0.877	C: 8
	Combined	0.945	C: 256

5.2 Animal detection in video streams

The experiment shows that the object detector is able to detect objects in a video stream. Using the SVM classifier with RBF kernel function and the color histogram for the feature descriptor, the results are obtained as shown in Table 5.3.

Table 5.3: Object detection in video streams results. Each time the detector is trained on two training sets, while a third set is used for testing the performance. The overlapping window ratio, precision and recall are measured for each run.

Train sets	Test set	ratio	precision	recall
1, 2	3	0.080	0.034	0.245
1, 3	2	0.567	0.446	0.742
2, 3	1	0.380	0.248	0.631

The performance varies widely over the different train-test runs. In general the recall is higher than the precision, which suggests that although many objects are indeed detected, there are also many false positives. The false positives can partly be explained by the fact that windows that contain only parts of an object are also classified as a detection by the detector. These errors are shown in Figure 5.3.

Figure 5.3: (Poor) detection in a video frame. The image shows mistakes where the detector does not recognize a cow (left) or recognizes a patch of grass as a cow (right). Also it is clear that due to the sliding window approach patches of ground around the cows are detected.

5.3 Learning while recognizing

Figure 5.4 shows the results of performing active learning in combination with object recognition. Two methods for training a classifier with a human labeling step after each iteration are compared. For the first 'active' method, the to be labeled samples are chosen by expected value for the classifier. With the 'random' method the to be labeled samples are chosen randomly. The results clearly show that in the active case the classifier obtains a higher score much faster than in the random case. When using active learning there are less labeled training samples needed to obtain the same scores.

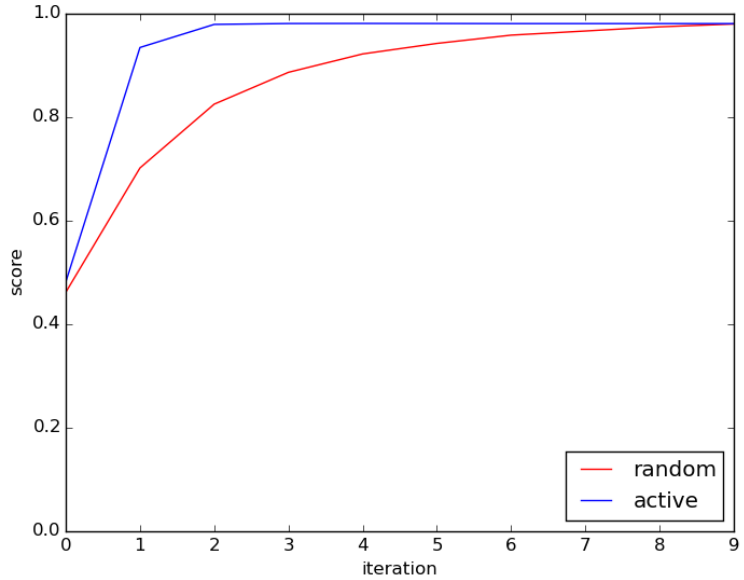


Figure 5.4: Scores when training a classifier which is retrained after each iteration with a larger training set than the previous iteration. Active learning is compared with random learning. The color histogram feature descriptor is used together with a support vector machine classifier with an RBF kernel function. The results for different feature descriptors are available in Appendix E.

5.4 Harvesting detection results

Objects that are found by the object detector are labeled by the annotator. These are then used to retrain the classifier of the object detector. Since only the objects that the detector considers 'positive' are fed to the annotator, these are the only objects that are harvested. The downside of this method is that false negatives (objects that are recognized as background) will not be added to the harvested results. Also for this reason these objects are then not used for retraining the classifier. Table 5.4 shows the results of retraining the object detector in harvesting mode compared to the normal object detecting mode.

Table 5.4: Performance of the object detector when annotated detections after each harvest are used for retraining the classifier.

Train sets	Test set	method	ratio	precision	recall
1, 2	3	normal	0.080	0.034	0.245
1, 2	3	harvest	0.126	0.105	0.175
1, 3	2	normal	0.567	0.446	0.742
1, 3	2	harvest	0.568	0.347	0.938
2, 3	1	normal	0.380	0.248	0.631
2, 3	1	harvest	0.455	0.325	0.649

It is clear the retraining after harvesting can benefit the object detector performance, depending on what measurement is used. The true value of the harvesting method is however that detected objects can be verified easily by a human annotator, which reduces the amount of errors made by the detector.

Chapter 6

Discussion

This research took the idea that the advance of technology in the last decade allows us to use unmanned aerial vehicles to detect animals automatically in natural environments. A framework has been developed that incorporates the aspect of collecting and annotating data, and using machine learning and computer vision for detecting animals in natural environments. Instead of the use of airplanes, helicopters or satellites to obtain ground images, also UAVs have been used. The UAV used in this thesis is a commercially affordable vehicle that can be easily operated remotely by most people. The specifications of the UAV and the used sensors have proven to be sufficient for the task of retrieving quality imagery for detecting animals (cows) on the ground.

Several feature descriptors and classifiers are compared and analyzed for their performance in the task of recognizing animals. The results show that color features (color histogram) provide valuable information for distinguishing cows from their background counterpart samples. It seems natural that this feature descriptor performs well in the environment that is tested: black and white cows on green land. In general this color histogram performs better than the HOG. The latter feature descriptor uses the gradients in (grayscale) images but lacks color information that is available in the original images made by the UAV. A combination of the two feature descriptors is analyzed and showed a minor improvement, at the cost of execution time (for calculating the features of both feature descriptors). The expectation was that the use of HOG in combination with the color histogram would provide the required information to distinguish those samples that are difficult for the color histogram alone (e.g. mud as background instead of green grass). From the compared classifiers the support vector machine (SVM) with the radial basis function kernel shows the best overall results. The results show however that the k-nearest neighbor classifier (k-NN) achieves almost comparable results. Comparing the kernel functions for a SVM it shows that using a radial basis function (RBF) kernel over a linear function

kernel provides better results in performance. The developed framework provides the tools that are required to detecting animals in a video recording. The constructed object detector can be used to find for example cows in a meadow. With some exception most of the cows are detected, but false positives introduce a significant error. The main cause of this error is due to the sliding window approach and can be reduced by suppressing detections. Two concepts that are useful for bringing object detection into practice are tested: learning while recognizing and harvesting detection results. Learning while recognizing allows us to bootstrap an animal recognition (and detecting) process with a minimal initial annotated dataset. The results show that using this active learning technique benefits the training of classifiers such that they achieve their required performance much faster (with less training samples) than when randomly selected data are labeled. Harvesting the detection results benefits the object detection process in that it validates data returned by the object detector, which in its turn can be used to further train a classifier.

The main research question for this project is: can a (low-end) UAV automatically detect animals like cows in a natural environment? The results have shown that we succeeded in doing so. A commercially affordable (low-end) UAV is used to detect cows in meadows using the video recordings that were made with the camera on the UAV. Although with some error, cows could be distinguished from their background using the developed framework. As a subquestion we asked: which of the popular feature descriptors and classifiers that are used for object detection maximize the results? From the tested feature descriptors (color histogram, HOG and the combined feature descriptor), the color histogram together with the combined feature descriptor performed the best. From the tested classifiers, the SVM with RBF kernel outperformed both the SVM with a linear kernel and the k-NN classifier. A second subquestion was asked: how can active learning and harvesting improve the object detection process for this task? The results show that using active learning a classifier can be trained faster when the correct samples are labelled by an annotator. This makes this tool useful for object detection as less training data is needed for the same performance. Also harvesting can be used to improve the process of detection, but is mainly useful for verification of the detected objects from the object detector.

This research has shown several but not a complete set of computer vision techniques and machine learning algorithms. A single UAV is used and a limited set of classifiers and feature descriptors are tested. As future work different hardware (other UAVs and sensors), feature descriptors and classifiers could be tested to explore which works best for the task of detecting objects in natural environments. The framework can be tested on different objects (other animals e.g.) and environments. The framework that is developed should invite other researchers to build further upon this

project. One of the major functionalities that would improve the practical use of detecting ground-based objects from the air is on-board processing. Software that is now run on a computer after the UAV has landed, may run on the UAV itself. This makes real-time interaction with the environment possible. An unmanned aerial system (UAS) can be developed to automatically detect and inspect targets located on the ground. This method can reduce the time required to act on situations where now the UAV first has te return to a central point for inspection.

Bibliography

[1] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[2] Chen, Y., Shioi, H., Montesinos, C. F., Koh, L. P., Wich, S., and Krause, A. (2014). Active detection via adaptive submodularity. In *Proceedings of The 31st International Conference on Machine Learning*, pages 55–63.

[3] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893.

[4] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645.

[5] Gates, J. E. and Trauger, D. L. (2014). Future trends in wildlife conservation and management programs. In *Peak Oil, Economic Growth, and Wildlife Conservation*, pages 281–298. Springer.

[6] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

[7] Hung, C., Xu, Z., and Sukkarieh, S. (2014). Feature learning based approach for weed classification using high resolution aerial images from a digital camera mounted on a UAV. *Remote Sensing*, 6(12):12037–12054.

[8] Khaleghi, A. M., Xu, D., Wang, Z., Li, M., Lobos, A., Liu, J., and Son, Y.-J. (2013). A DDDAMS-based planning and control framework for surveillance and crowd control via UAVs and UGVs. *Expert Systems with Applications*, 40(18):7168–7183.

[9] Koh, L. P. and Wich, S. A. (2012). Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation. *Tropical Conservation Science*, 5(2):121–132.

[10] Laliberte, A. S., Herrick, J. E., Rango, A., and Winters, C. (2010). Acquisition, orthorectification, and object-based classification of unmanned aerial vehicle (UAV) imagery for rangeland monitoring. *Photogrammetric Engineering & Remote Sensing*, 76(6):661–672.

[11] López-Granados, F., Torres-Sánchez, J., Serrano-Pérez, A., de Castro, A. I., Mesas-Carrascosa, F.-J., and Peña, J.-M. (2016). Early season weed mapping in sunflower using uav technology: variability of herbicide treatment maps against weed thresholds. *Precision Agriculture*, 17(2):183–199.

[12] Luo, T., Kramer, K., Samson, S., Remsen, A., Goldgof, D., Hall, L., and Hopkins, T. (2004). Active learning to recognize multiple types of plankton. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 478–481. IEEE.

[13] Merino, L., Caballero, F., Martínez-de Dios, J. R., Ferruz, J., and Ollero, A. (2006). A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23(3-4):165–184.

[14] Ofli, F., Meier, P., Imran, M., Castillo, C., Tuia, D., Rey, N., Briant, J., Millet, P., Reinhard, F., Parkan, M., et al. (2016). Combining human computing and machine learning to make sense of big (aerial) data for disaster response. *Big Data*, 4(1):47–59.

[15] Patterson, M. C. and Brescia, A. (2008). Integrated sensor systems for uas. Technical report, DTIC Document.

[16] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[17] Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11.

[18] van Gemert, J. C., Verschoor, C. R., Mettes, P., Epema, K., Koh, L. P., and Wich, S. (2014). Nature conservation drones for automatic localization and counting of animals. In *Computer Vision-ECCV 2014 Workshops*, pages 255–270. Springer.

[19] Visser, A. E., Noordam, N. M., Epema, H. K., Verschoor, C. R., Kop, R., and Dokoupil, A. (2014). DutchUASDataset001: 577 annotated aerial images of rhinos, zebras, rangers and cars. `http://www.dutchuas.nl/dataset/`.

[20] Vondrick, C., Patterson, D., and Ramanan, D. (2013). Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 101(1):184–204.

[21] Zhang, C. and Kovacs, J. M. (2012). The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6):693–712.

[22] Zhang, W., Zelinsky, G., and Samara, D. (2007). Real-time accurate object detection using multiple resolutions. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8.

[23] Zhou, G. (2009). Near real-time orthorectification and mosaic of small UAV video flow for time-critical event response. *Geoscience and Remote Sensing, IEEE Transactions on*, 47(3):739–747.

[24] Zhu, Q., Yeh, M.-C., Cheng, K.-T., and Avidan, S. (2006). Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498.

Appendix A

Cut video script

The following script is used to crop a video to the desired length. This is useful to remove unwanted parts of the video that are not useful for the video annotation tool or the object detector.

```
#!/bin/sh

# $0 is the script name
# $1 is the input filename
# $2 is the output filename
# $3 is the start time ARG
# $4 is the end time

INPUTFILENAME="$1"
TMPFILENAME="tmp_file.MOV"
OUTPUTFILENAME="$2"
DURATION=`expr $4 - $3`

ffmpeg -i $INPUTFILENAME -ss $3 -c copy $TMPFILENAME
ffmpeg -i $TMPFILENAME -t $DURATION -c copy $OUTPUTFILENAME
rm $TMPFILENAME
```

Appendix B

DJI Phantom 3 Advanced specifications

Table B.1: DJI Phantom 3 Advanced specifications

Phantom 3 Advanced	
Weight	1280g
Size including propellers	689 mm
Maximum ascent speed	5m/s
Maximum descent speed	3m/s
Maximum horizontal speed	16 m/s
Hover accuracy	vertical ±10cm; horizontal ±1m
Operating temperature	$0 - 40°C$
Camera	
Sensor	Sony EXMOR 1/2.3”
Effective pixels	12.4M
Lens	FOV 94° 20mm f/2.8
ISO Range	100-3200 (video) 100-1600 (foto)
Shutter time	8s - 1/8000s
Max photo size	4000 x 3000
Photograph modes	Single shot, Burst, time-lapse, Auto Exposure Bracketing:3/5, Bracketed Frames at 0.7EV Bias.
Video recording modes	
UHD	2704x1520p 24/25/30 (29.97)
FHD	1920x1080p 24/25/30/48/50/60
HD	1280x720p 24/25/30/48/50/60
SD card	Micro SD with maximum 64 GB (preferable class 10 or UHS-1)
File format	FAT32/exFAT; JPEG, DNG, MP4, MOV
Gimbal	
Pitch	$-90° - 0°$

Stabilization	3-axial
Remote controller	
Frequency	2.400 GHz - 2.483 GHz
Maximum range	2 kilometer
Video output	USB
Battery	6000mAh LiPo 2S
Mobile device holder	For smart-phone or tablet
Voltage	1.2 A @7.4V
Battery charger	
Voltage	17.4V
Maximum power	57W
Battery	
Capacity	4480 mAh
Voltage	15,2 V
Type	LiPo 4S
Energy	68Wh
Maximum flight time	23 minutes
Maximum charging power	100W
App/Live View	
Mobile app	DJI Go
EIRP	100mW
Frequency live view	2.4GHz ISM
Quality live view	720P @ 30fps
Operating system	iOS 8.0 or higher; Android 4.1.2 or higher

Appendix C

Feature descriptors parameter sweep results

Table C.1: Classification results with different parameters for the color histogram feature descriptor. The tested parameters are the amount of bins in the histogram per channel and which channels are used: hue (H), saturation (S) and value (V). The k-nearest neighbors classifier with $k = 5$ is used for each test. The inter-set split test results (Set 1, Set 2, Set 3 and their average) are shown with the cross-set test results.

Channels	Binsize	Set 1	Set 2	Set 3	Average (inter-set)	Cross-set
H, S, V	32	0.962	0.981	0.987	0.977	0.807
H, S, V	16	0.969	0.970	0.985	0.974	0.836
H, S, V	8	0.982	0.985	0.986	0.984	0.885
H, S	32	0.983	0.970	0.981	0.978	0.906
H, S	16	0.982	0.977	0.982	0.980	0.902
H, S	8	0.994	0.959	0.972	0.975	0.897
H	32	0.992	0.954	0.982	0.976	0.916
H	16	0.997	0.958	0.979	0.978	0.937
H	8	0.992	0.927	0.980	0.966	0.931
S	32	0.985	0.966	0.974	0.975	0.960
S	16	0.988	0.958	0.977	0.974	0.959
S	8	0.990	0.951	0.963	0.968	0.950

Table C.2: Classification results with different parameters for the histogram of orientation gradients feature descriptor. The tested parameters are the amount of orientations (Orient.), the amount of pixels per cell (Ppc) and the amount of cells per block (Cpb). The k-nearest neighbors classifier with $k = 5$ is used for each test. The inter-set split test results (Set 1, Set 2, Set 3 and their average) are shown with the cross-set test results.

Orient.	Ppc	Cpb	Set 1	Set 2	Set 3	Average (inter-set)	Cross-set
4	8	1	0.848	0.862	0.957	0.889	0.878
4	8	2	0.843	0.862	0.957	0.888	0.877
4	8	3	0.848	0.862	0.957	0.889	0.877
4	16	1	0.903	0.899	0.961	0.921	0.880
4	16	2	0.913	0.899	0.958	0.923	0.881
4	16	3	0.928	0.899	0.958	0.928	0.882
4	32	1	0.929	0.933	0.960	0.940	0.873
4	32	2	0.963	0.919	0.968	0.950	0.898
4	32	3	0.966	0.922	0.966	0.952	0.886
8	8	1	0.843	0.862	0.957	0.887	0.877
8	8	2	0.836	0.862	0.957	0.885	0.877
8	8	3	0.839	0.862	0.957	0.886	0.877
8	16	1	0.892	0.888	0.957	0.912	0.879
8	16	2	0.904	0.877	0.957	0.913	0.881
8	16	3	0.913	0.888	0.957	0.919	0.882
8	32	1	0.924	0.932	0.957	0.938	0.884
8	32	2	0.961	0.912	0.966	0.947	0.896
8	32	3	0.959	0.911	0.964	0.945	0.884
16	8	1	0.832	0.862	0.957	0.884	0.877
16	8	2	0.832	0.862	0.957	0.884	0.877
16	8	3	0.834	0.862	0.957	0.884	0.877
16	16	1	0.875	0.877	0.957	0.903	0.879
16	16	2	0.887	0.866	0.957	0.903	0.879
16	16	3	0.899	0.870	0.957	0.909	0.881
16	32	1	0.921	0.920	0.959	0.933	0.888
16	32	2	0.956	0.905	0.967	0.943	0.891
16	32	3	0.956	0.908	0.963	0.943	0.884

Table C.3: Classification results with different parameters for the histogram of orientation gradients (HOG) combined with color histogram feature descriptor. The tested HOG parameters are the amount of orientations (Orient.), the amount of pixels per cell (Ppc) and the amount of cells per block (Cpb). The tested color histogram parameters are the amount of bins in the histogram per channel and which channels are used: hue (H), saturation (S) and value (V). The k-nearest neighbors classifier with $k = 5$ is used for each test. The inter-set split test results (Set 1, Set 2, Set 3 and their average) are shown with the cross-set test results.

Orient.	Ppc	Cpb	Channels	Binsize	Set 1	Set 2	Set 3	Average (inter-set)	Cross-set
4	32	2	H, S, V	32	0.971	0.974	0.976	0.974	0.774
4	32	3	H, S, V	32	0.963	0.977	0.980	0.973	0.795
4	32	2	H, S, V	8	0.981	0.989	0.983	0.984	0.879
4	32	3	H, S, V	8	0.981	0.985	0.984	0.983	0.883
4	32	2	S	32	0.988	0.954	0.975	0.972	0.947
4	32	3	S	32	0.986	0.962	0.974	0.974	0.955
4	32	2	S	8	0.991	0.943	0.967	0.967	0.951
4	32	3	S	8	0.990	0.951	0.966	0.969	0.950

Appendix D

Classifier parameter sweep results

Table D.1: Classification results for k-NN with different k values

	Color Hist		HOG		HOG Color Hist	
K	mean	std	mean	std	mean	std
1	0.960	0.010	0.898	0.052	0.957	0.008
2	0.958	0.008	0.891	0.057	0.958	0.009
3	0.960	0.008	0.897	0.056	0.961	0.009
4	0.956	0.007	0.892	0.059	0.957	0.009
5	0.957	0.007	0.893	0.058	0.956	0.007
10	0.953	0.009	0.886	0.061	0.953	0.009

Table D.2: Classification results for Support Vector Classifier with RBF kernel with different parameters

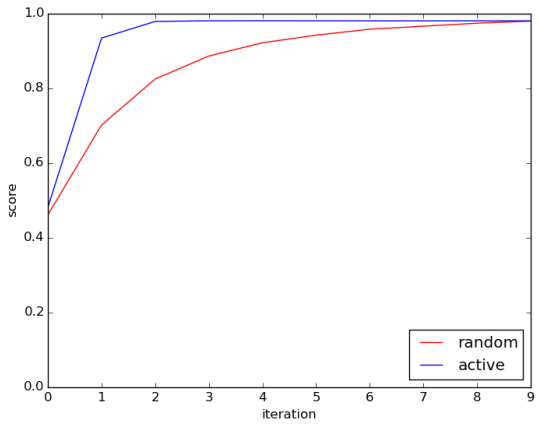
		Color Hist		HOG		HOG Color Hist	
C	gamma	mean	std	mean	std	mean	std
1	0.1	0.863	0.041	0.877	0.059	0.863	0.041
1	1	0.897	0.059	0.877	0.059	0.897	0.059
1	10	0.955	0.006	0.874	0.055	0.955	0.006
1	100	0.965	0.003	0.902	0.058	0.965	0.003
1	1000	0.897	0.056	0.929	0.044	0.885	0.057
2	0.1	0.858	0.036	0.877	0.059	0.858	0.036
2	1	0.898	0.055	0.877	0.059	0.898	0.055
2	10	0.957	0.004	0.870	0.050	0.957	0.004
2	100	0.964	0.005	0.904	0.052	0.964	0.004
2	1000	0.900	0.051	0.929	0.042	0.889	0.056
3	0.1	0.872	0.043	0.877	0.059	0.872	0.043
3	1	0.910	0.043	0.877	0.059	0.910	0.043
3	10	0.957	0.004	0.865	0.044	0.958	0.004
3	100	0.960	0.010	0.907	0.051	0.961	0.007
3	1000	0.900	0.051	0.930	0.042	0.889	0.056
5	0.1	0.881	0.046	0.877	0.059	0.881	0.047
5	1	0.920	0.034	0.877	0.059	0.920	0.034
5	10	0.959	0.005	0.865	0.041	0.958	0.006
5	100	0.962	0.010	0.909	0.051	0.964	0.006
5	1000	0.900	0.051	0.930	0.042	0.889	0.056

Table D.3: Classification results for Support Vector Classifier with linear kernel with different parameters

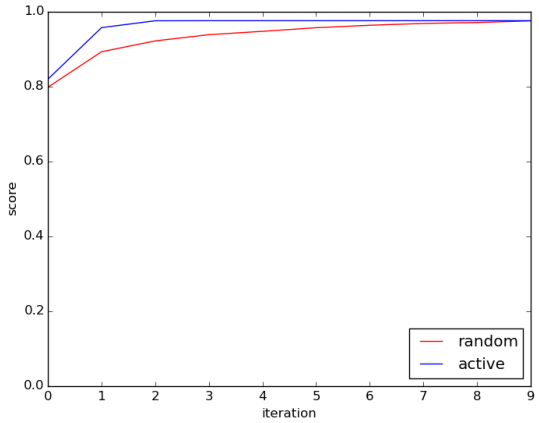
	Color Hist		HOG		HOG Color Hist	
C	mean	std	mean	std	mean	std
1	0.880	0.046	0.877	0.059	0.881	0.047
2	0.887	0.051	0.877	0.059	0.888	0.052
4	0.895	0.055	0.877	0.059	0.896	0.055
8	0.905	0.043	0.877	0.059	0.905	0.043
16	0.915	0.036	0.876	0.058	0.917	0.037
32	0.926	0.030	0.870	0.050	0.926	0.029
64	0.935	0.022	0.862	0.042	0.933	0.018
128	0.940	0.013	0.859	0.035	0.943	0.016
256	0.946	0.012	0.858	0.031	0.945	0.009
512	0.945	0.012	0.856	0.030	0.944	0.022
1024	0.948	0.017	0.855	0.028	0.939	0.029
2048	0.950	0.016	0.853	0.028	0.933	0.043
4096	0.953	0.016	0.851	0.030	0.925	0.040
8192	0.955	0.013	0.850	0.036	0.912	0.051
16384	0.953	0.017	0.855	0.034	0.920	0.047
32768	0.947	0.025	0.865	0.032	0.922	0.048
65536	0.942	0.034	0.872	0.025	0.920	0.047
131072	0.948	0.031	0.867	0.032	0.907	0.047
262144	0.951	0.024	0.869	0.041	0.910	0.045

Appendix E

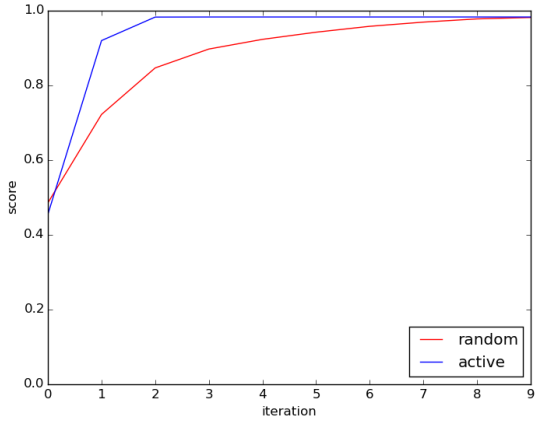
Active learning results



(a) color histogram



(b) histogram of oriented gradients



(c) combined

Figure E.1: Scores when training a classifier which is retrained after each iteration with a larger training set than the previous iteration. Active learning is compared with random learning. The different graphs show the results when different feature descriptors are used. In each case the support vector machine with an RBF kernel function is used.