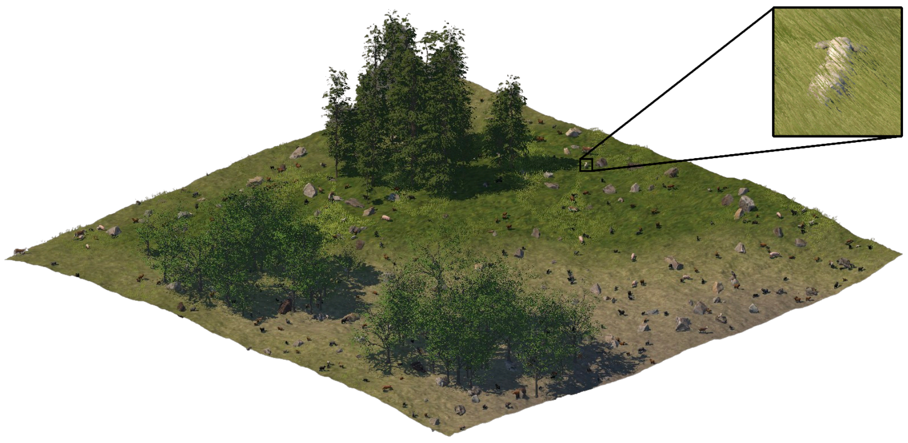
Bjørnar Østtveit

Using synthetic data to improve the detection of sheep in drone images

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
June 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Bjørnar Østtveit

Using synthetic data to improve the detection of sheep in drone images

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Retrieving grazing sheep at the end of the season is typically a very time-consuming process. After the majority of the animals have been returned, there can often be a few animals left that did not make it back with the rest. Locating and retrieving the last few animals usually entails tracking long distances and searching for a long period of time. By properly utilizing modern technology, this labor-intensive process can be alleviated. Drones, or unmanned aerial vehicles (UAVs), have become increasingly popular in recent decades for different commercial sectors such as agriculture. Previous research has shown that using machine learning methods to detect sheep in drone images can be a viable method of locating grazing sheep for retrieval.

Sheep come in different colors and are often partially occluded by vegetation such as leaves and grass when being imaged by a drone. This thesis aims to discover which categories of sheep are difficult to detect in drone images, as well as propose a method of generating additional synthetic training data to attempt to improve the detection of the categories that were found to be the most difficult to detect. To generate the synthetic data, the Perception package for the game engine Unity was used. The machine learning method used for detecting sheep was YOLOv5. Two models were trained; a baseline model using only real data, and a mixed model using both real and synthetic data in the training set.

The categories of sheep that were found to be the most difficult to detect were all colors of sheep as long as they are partially occluded by vegetation and in particular occluded sheep of darker colors. The mixed model achieved a **9.43%** higher recall over the baseline for occluded sheep in general, and a **37.5%** higher recall over the baseline for dark occluded sheep at a confidence threshold of 0.8.

Sammendrag

Gjenfinning av sau på slutten av beitesesongen er ofte en veldig tidkrevende prosess. Etter flesteparten av sauene er returnert kan det ofte være noen få dyr igjen som ikke kom seg tilbake med resten. Lokalisering og gjenfinning av de siste dyrene innebærer ofte sporing og leting over lange distanser over en lang tidsperiode. Denne arbeidskrevende prosessen kan bli lettere ved å ta i bruk moderne teknologi. Droner har blitt mer populært de siste tiårene i ulike kommersielle sektorer som for eksempel i jordbrukssektoren. Tidligere forskning har vist at å benytte seg av maskinlæringsmetoder for å detektere sauer i dronebilder kan være en praktisk måte å lokalisere sauer på beite for gjenfinning.

Sauer finnes i ulike farger og kan ofte være delvis tildekket av vegetasjon som blader og gress når man tar bilder med drone. Denne oppgaven har som mål å finne ut av hvilke kategorier av sau som er vanskelig å detektere i dronebilder, i tillegg til å foreslå en metode for å generere syntetisk data for å forsøke å forbedre deteksjonen av de kategoriene som ble funnet å være de vanskeligste å detektere. For å generere den syntetiske dataen, ble *Perception* pakken for spillmotoren Unity brukt. Maskinlæringsmetoden som ble brukt til å detektere sauer var YOLOv5. To modeller ble trent; den såkalte *baseline*-modellen som kun ble trent på ekte dronebilder, og den såkalte *mixed*-modellen som brukte både ekte og syntetisk genererte data til trening.

Kategoriene av sau som ble funnet å være de vanskeligste å detektere var sauer av alle farger så lenge de var delvis tildekket av vegetasjon, men spesielt sauer av mørke farger som var delvis tildekket. *Mixed*-modellen oppnådde en **9.43%** høyere fullstendighet (recall) enn *baseline*-modellen for tildekkede sauer generelt, og en **37.5%** høyere fullstendighet (recall) enn *baseline*-modellen for mørke tildekkede sauer ved en konfidensterskel på 0.8.

Preface

This is a master's thesis is written as part of the Computer Science program at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). I would like to thank my supervisor Svein-Olaf Hvasshovd for his support, motivation, and his weekly follow-ups. The dataset used in this thesis was hand-labeled collaboratively with the other master's students working on related theses, so I would like to thank Ingebright Nygård, Sebastian Vittersøe, and Hallvard Stemshaug for the time they put into this. Finally, I would also like to thank the other master's students located at the *Gamle Fysikk* building for the excellent work environment and moral support.

Table of contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network
AP	Average Precision
CNN	Convolutional Neural Network
COCO	Common Objects in Context
FN	False Negative
FOV	Field of View
FP	False Positive
FPS	Frames per Second
GAN	Generative Adversarial Network
GPS	Global Positioning System
IoU	Intersection over Union
mAP	Mean Average Precision
MLP	Multilayer Perceptron
MSX	Multi-Spectral Dynamic Imaging
NIBIO	The Norwegian Institute of Bioeconomy Research
NMS	Non-Maximum Suppression
NSG	The Norwegian Association of Sheep and Goat Farmers
R-CNN	Region-Based Convolutional Neural Network
RGB	Red-Green-Blue
TN	True Negative
TP	True Positive
UAV	Unmanned Aerial Vehicle
UTM	Universal Transverse Mercator
VAT	Value Added Tax
VOC	Visual Object Classes

WGS84	World Geodedic System 1984
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Problem description

Every fall at the end of the grazing period, sheep farmers need to retrieve all of their sheep back from the grazing areas. The areas where sheep graze can be quite extensive, and so finding and returning all of the sheep can be very labor-intensive and time-consuming. One major component of sheep retrieval is simply locating the sheep in the first place, which can be quite challenging. Typically, sheep wear bells around their necks so that the farmer can hear when they get close. This is helpful, but the farmer still needs to get within hearing range of the sheep first. After most of the sheep have been retrieved, there can be some left that have strayed further away or for other reasons did not make it back with the rest of the herd (Hvasshovd 2017). The farmer may spend a large amount of time finding and retrieving these last few animals, tracking very long distances in the process.

According to the Norwegian Agriculture Agency (Landbruksdirektoratet), out of 748,106 sheep and 1,154,064 lambs sent out to graze in 2021, 25,485 sheep, and 79,155 lambs never made it back at the end of the season (Landbruksdirektoratet 2022). There are many different reasons for sheep not returning, some are killed by predators; according to Rovbase, 2,527 sheep and 14,388 lambs were declared to be killed by predators in 2021. (Rovbase 2022). This still leaves many sheep that do not return for other reasons, some die of other causes than predators like disease or accidents, and some are lost for unknown reasons. Losing sheep during the grazing period can be very costly to the farmer. The Norwegian Association of Sheep and Goat Farmers (NSG), values a lost lamb at 1,850 NOK, and lost sheep at 3,585 NOK (NSG 2022). In Norway, farmers by law get compensated for sheep killed by predators, but not for sheep that are lost for other reasons (Lovdata 2014).

For the reasons stated above; the amount of time it takes to locate the last sheep, as well as the high cost of losing sheep, the farmer can save a lot of time and money by more easily locating sheep in the vast grazing areas. Being able to track the sheep, or at least getting an approximate location before going out to search can be a massive help to alleviate these problems.

1.2 Existing commercial solutions

There are several commercial solutions available to address these problems. The following products all have in common that they are tracking devices mounted around the neck of the sheep just like their regular bells. They vary in their connectivity, features, and pricing.

1.2.1 E-bjella

E-bjella is made by the company Findmy and is a tracking collar made for sheep, cattle, and reindeer (*Findmy* 2022). It uses Global Positioning System (GPS) to get the sheep's location and transmits this data via satellites. They have an advertised battery capacity of 2 to 3 seasons and batteries are replaceable. They claim to be built to the United States military standard MIL-STD-810, which is a standard for creating *rugged* products, that can withstand the environmental conditions that they will face in their use. There is no authority for certifying such compliance however (*MIL-STD-810* 2022). Some other features of E-bjella are; notifications when an animal has not moved more than 40 meters in two days, which can be a sign that the animal has died; notifications when an animal moves outside of a predetermined geofence; and notifications of distress in a flock based on "abnormal" behavior. The price of E-bjella is 1,749 NOK per unit, in addition to an annual fee of 239 NOK per unit (Value Added Tax (VAT) excluded). They claim to have sold 40,000 units in total.

1.2.2 Radiobjella

Radiobjella is sold by the company Telespor and is another tracking collar for grazing animals such as sheep (*Telespor* 2022). It also uses GPS to track the animal's location, however, it transmits over the cellular infrastructure as opposed to using satellites. Radiobjella uses replaceable batteries that they recommend changing every season. The unit is waterproof and equipped with a motion sensor. Some extra features of this product are notifications when the motion sensor has not detected movements in the last three hours; when the animal has stayed in the same location for a long period; and if it has not been able to report its position in the last two attempts. The price per unit is 989 NOK, and the annual cost of subscription and batteries is either 129 NOK or 209 NOK (VAT excluded) depending on whether a 5 or 12-month subscription is needed.

1.2.3 Smartbjella

Smartbjella is another such tracker, made for sheep, goats, cattle, and reindeer (*Smartbjella* 2022). It uses GPS to get the animal's location and transmits over a cellular connection. It is waterproof (IP67 certified), and the battery is advertised to last up to between 1,5 years (reporting its position every hour) and 17 years (reporting its position every 24 hours). This product also has a temperature sensor and notifies the farmer when it believes that the sheep is dead. The cost of Smartbjella is 999 NOK, and the annual subscription cost per unit is between 109 NOK and 149 NOK (VAT excluded) as they have three different subscription plans, lasting 5, 7, and 12 months. Smartbjella claims to have sold 30,000 units in Norway.

1.2.4 Discussion of existing solutions

The existing solutions do address the problems raised in section 1.1, but are by no means perfect solutions. Since they are all devices mounted around the necks of sheep, they can generally only be worn by adult sheep since lambs grow too much during the grazing season. The statistics from the Norwegian Agriculture Agency indicate, that over 50% more lambs than sheep were sent out to graze in 2021 (Landbruksdirektoratet 2022). However, sheep typically organize themselves into family groups of 8 to 10 sheep, usually a ewe along with its offspring (Johanssen and Sørheim 2018). By only tracking the ewe then, one is often able to track the entire family group. Because of these behavioral characteristics, FindMy recommends tracking a minimum of only 25% of the adult sheep (*Findmy* 2022). However, should the ewe die during the grazing period, one will potentially lose track of the entire family group.

Another issue with Radiobjella and Smartbjella is that they rely on cellular infrastructure to transmit data. Even though Norway has very good national cellular coverage, the telecommunications companies focus on covering the areas where their customers live and do not necessarily cover remote areas where there are more sheep than people. For this reason, not all farmers will be able to use these solutions. E-bjella, which uses satellites, should not have this problem but is also far more expensive than the other options.

Even if only a portion of the sheep needs to be equipped with any of the above products, it is both a significant upfront investment to purchase these products, as well as a significant ongoing cost related to using them. It is not known how long these products last before they need to be replaced, so only the initial purchase costs, as well as the ongoing subscription costs, can easily be examined.

According to The Norwegian Institute of Bioeconomy Research (NIBIO), the average farmer had 162 winter-fed sheep (NIBIO 2020a), and for context, they had an average operating profit on their sheep of 159,000 NOK in 2020 (NIBIO 2020b). Using these numbers the cost of using these different products for an average sheep farmer in Norway can be estimated, this can be seen in Table 1.1. For the products with multiple subscription plans, the cheapest available plan for the latest model they sell was used.

	E-bjella	**Radiobjella**	**Smartbjella**
25% tracked			
Up-front cost	69,960	39,560	39,960
Seasonal cost	9,560	5,160	4,360
50% tracked			
Up-front cost	139,920	79,120	79,920
Seasonal cost	19,120	10,120	8,720
75% tracked			
Up-front cost	211,629	119,669	120,879
Seasonal cost	28,919	15,609	13,189
100% tracked			
Up-front cost	283,338	160,218	161,838
Seasonal cost	38,718	20,898	17,658

Table 1.1: Up-front and seasonal cost (in NOK) of existing commercial solutions assuming 162 adult sheep

This is by no means a proper cost-benefit analysis of these products; however, they clearly represent a significant up-front and seasonal cost to the average sheep farmer in relation to their slim profits. As mentioned above, FindMy reports having sold approximately 40,000 units in total, and Smartbjella, approximately 30,000 units. Telespor has not reported their sales numbers for Radiobjella. Still, this is not close to covering the population of adult sheep being sent out to graze in Norway each year and it, therefore, appears to be ample room for more competitive solutions in this market.

1.3 Using drones for sheep retrieval

Drones, also known as Unmanned Aerial Vehicles (UAVs) have traditionally been developed for military applications (Blyenburgh 1999). In more recent years, drones have seen applications in commercial sectors, including in agriculture for crop monitoring, crop spraying, soil analysis, and more. Although the costs of drones have come down significantly over the last decades due to more companies offering drones for commercial and consumer use, they still represent a significant investment to an individual farmer (Huang et al. 2013).

There are primarily two different approaches to using drones for sheep retrieval. The first is using a low-power radio transmitter that can be small enough to be attached to the ear tag of the animal. This could be possible since the drone can be flown close enough to pick up these weak signals to get an estimated position. These radio-based ear tags could be made much cheaper than the existing solutions mentioned above as they do not require GPS or internet connectivity which would also require much less power and hence, a much smaller battery. The localization, in this case, comes from the drone's GPS as well as range estimation between the drone and sheep. This has been examined by previous master's students, like (Nyholm 2020), (Steinsvik 2021), and (Nerland 2021). The second approach is using the drone to image the areas of interest and using machine learning techniques to automatically detect the presence of sheep in the images. This has the advantage of not requiring any hardware to be attached to the sheep. Range estimation is not necessary in this case, as the sheep will be sufficiently close to the drone for each image taken, assuming that the camera is pointed mostly straight down.

This thesis focuses on using drones that image an area looking for sheep. In this case, the costs are relatively fixed and tied to the upfront cost of purchasing the drone itself. In practice, however, a potential commercial software solution for guiding the drone automatically and detecting sheep in the images taken may or may not have a recurring cost associated with it. In theory, at least, the cost would not need to increase with the number of sheep detected in contrast to the other solutions, so it could create more opportunities for economics of scale. However, given that the average sheep farmer in Norway has a relatively small number of sheep (NIBIO 2020a), it could still be prohibitively expensive to purchase a drone outright, so it may be more economical for multiple farmers to share such a system, or rent a system only when they need to locate their sheep.

Using drone imaging for sheep retrieval comes with a few key challenges.

1. Making sure the drone is able to image the entire area of interest so that no areas are left unchecked.

2. Actually detecting sheep everywhere a sheep is present.

The first problem is related to planning a route for the drone to follow and when images should be taken to completely cover an area of interest. The route planning problem for sheep localization has been examined by previous master students, e.g. (Rognlien and Tran 2018). The second problem relates to the automatic detection of sheep in the captured images and making sure that the method of choice is able to detect as many of the sheep in the area of interest as possible. Some of the previous master's theses related to this topic are e.g. (Muribø 2019), (Kaarud, Nordvik and Paulsen 2020), and (Furseth and Granås 2021). Detecting every single sheep in an image is not strictly necessary, as it is only necessary to determine whether any sheep are present in a specific area or not; therefore, it is sufficient as long as at least one sheep is detected for every image where sheep are present. However, one hard limitation of using drone imaging for sheep retrieval is that if a sheep is completely occluded in the camera's field of view when the image is taken, the sheep cannot possibly be detected. This is especially problematic in densely forested areas.

1.4 Aim of this thesis

This thesis focuses on the second challenge related to drone imaging for sheep retrieval mentioned above, that is, making sure the automatic detection performance of sheep in drone images is adequate. Specifically, this thesis aims to discover which categories of sheep are the most difficult to detect and whether the detection of these difficult categories can be improved by generating synthetic training data to supplement the real data that is available.

First, the sheep in the drone images are classified into subcategories based on the color of their wool and whether they are partially occluded by vegetation. Then, a machine learning model for sheep detection will be trained on this data and the performance of the different subcategories will be compared to discover which categories of sheep are the most difficult to detect based on the recall achieved on the different subcategories. This will give a good insight into where there is the most room for improvement with regards to detecting sheep in all images where sheep are present. Next, an attempt will be made to improve the detection of the subcategories of sheep that are the most difficult to detect. This will be done by generating additional synthetic training data which focuses on these difficult categories. Finally, a second machine learning model will be trained using both the real and the synthetic data, and the performance of both models will be compared to see if it is possible to improve the detection of these difficult cases without having to go out and get more real data for those categories.

The initial hypothesis is that the categories of sheep that have the lowest number of instances in the real dataset will also be the most difficult to detect, as the model will have fewer examples available to learn from. The synthetic data generation will therefore focus on creating more examples of these underrepresented categories so that the detection can be improved.

Real data is very expensive and time-consuming to create, as going out with a drone and capturing real images is a lot of work. To get a good amount of variation in the data one would need to go out at different times of day and year, and in different areas. After that, one would need to manually label thousands of images which is very time-consuming. Generating synthetic data is very appealing as it allows doing a fixed amount of work which in principle can yield an infinite amount of data. Moreover, it gives a high degree of control with regard to the variation of the data that is generated.

1.5 Technology choice

YOLOv5 was the chosen machine learning model to detect sheep. It is an object detection method, meaning that it attempts to find and localize every sheep in a given image. This model was chosen because of its ease of use, its wide selection of model sizes, and its popularity; it has over 25k stars on GitHub. It is also relatively new, being initially released in May of 2020 and receiving continual updates ever since to keep up with new developments in the field (Jocher et al. 2022). It was also the model that was used in the preparatory project for this thesis.

Unity perception was chosen as a method of generating synthetic data. This is a package for the popular game engine Unity, and the package is developed by the same company as the game engine itself. It is currently in an experimental stage; however, it is already quite feature-rich. The primary appeal of this method for generating synthetic data is that one gets to utilize the powerful game engine Unity, which is capable of rendering realistic scenes in real-time, along with all of the existing infrastructure and community support a mature game engine provides (Unity Technologies 2020).

1.6 Preparatory project

In the preparatory project for this thesis, the inference speed and performance of different variants of YOLOv5 were examined. There were 10 different variants at the time, all were pretrained on the Common Objects in Context (COCO) dataset, five were pretrained at a resolution of 640x640, and the other five were pretrained at a resolution of 1024x1024. The input data had the same resolution as the pretraining for each model. The different variants differ in size in terms of no. of parameters. The aim of the preparatory project was to examine which model was most suited to being run on lower performance hardware that could be mounted directly onto a drone. This was done by comparing the inference speed on a Raspberry Pi 3 Model B. The models that were trained on a resolution of 1024x1024 significantly outperformed the models trained on 640x640 in terms of recall. Models of greater no. of parameters only had a marginal increase in performance while having a significantly slower inference speed. Inference on higher resolution also had a significant inference speed reduction, however, they also had a significant performance increase. The conclusion was that the increase in performance was mostly due to training the custom dataset at a higher resolution since the no. of parameters had a much smaller impact on performance. Therefore, using the smallest model available with a suitable input resolution was deemed to be the best option for running inference on limited hardware, as the larger models used an unacceptable amount of time with only marginal performance increases.

1.7 Outline of this document

Chapter 2 - Previous work, presents previous academic work related to topics that are relevant to this thesis including: previous master's theses, object detection in drone images, wildlife monitoring, and synthetic data generation.

Chapter 3 - Theory, gives a theoretical basis for understanding the machine learning method used, as well as how its performance is evaluated.

Chapter 4 - Sensor data, relates to the dataset captured in the field by the actual drone. It presents information extracted from the image metadata about when and where the images were taken, how the images were labeled, how many sheep of different categories are in the dataset, how the data was split into sets for training, validation, and testing, and what kind of preprocessing was applied to the images. Throughout this thesis, the dataset captured by the drone is referred to interchangeably as either the *sensor dataset* or the *real dataset*.

Chapter 5 - Synthetic data, explains in detail how the synthetic dataset was generated.

Chapter 6 - Experiment structure, concretizes what will be examined into research questions, and explains how those questions will be answered, as well as specifying the training and testing regime.

Chapter 7 - Results and discussion, presents and discusses the results of the experiment as it relates to the research questions.

Chapter 8 - Conclusion and future work, using the results and discussion from the previous chapter attempts to answer the research questions posed in chapter 6, as well as suggest future work in part based on these answers.

Chapter 2

Previous work

2.1 YOLO

You Only Look Once (YOLO) is a series of object detection models. What they have in common is that they are single-stage detectors, meaning that they perform the entire detection task from feature extraction to bounding box prediction in a single forward pass. When the first *version* of YOLO was released in 2015, the most popular object detection models were Region-Based Convolutional Neural Networks (R-CNNs), which were reasonably accurate, but due to their multi-stage detection process were quite slow. The aim of the original YOLO model was to enable real-time object detection by doing everything in a single stage (Redmon, Divvala et al. 2015). In their paper, they compare the performance of different object detection models with YOLO on the Pascal Visual Object Classes (VOC) 2007 dataset. One interesting comparison is YOLO versus Faster R-CNN ZF. YOLO achieved a very similar Mean Average Precision (mAP) of **0.634** versus **0.621** for Faster R-CNN ZF. The inference speed of YOLO was in this case approximately **2.5x** faster, achieving 45 Frames per Second (FPS) versus 18 FPS for Faster R-CNN ZF. There were other models in the comparison that performed significantly better than YOLO in terms of mAP, but they in turn had an abysmal inference speed compared to YOLO.

YOLOv2 was released one year later by two of the original authors of YOLO (Redmon and Farhadi 2016). This version of YOLO improved both speed and mAP over the original YOLO. The inference speed improvement was mostly due to using *Darknet-19* as the new backbone for feature extraction. This backbone requires significantly fewer floating-point operations than the backbone used in the original YOLO. The mAP was improved by a multitude of modifications, including but not limited to; adding batch normalization, using a higher resolution classifier, introducing a passthrough layer that concatenates high resolution and low-resolution feature maps to improve localization, and randomly scaling the input images when training. Instead of using a fixed input resolution, YOLOv2 can be run at several different resolutions, where lower resolutions have a faster inference speed. This enables a smooth trade-off to be made between speed and precision by running the model on different resolutions. In one of their tests, they ran YOLOv2 on a resolution of 288x288, which still outperformed the original YOLO in terms of mAP, but at double the speed. When running at a resolution of 544x544, the speed was roughly **11%** slower than the original YOLO but achieved a **24%** higher mAP.

They also introduced a method of training the network on both object detection datasets like COCO in addition to classification datasets like ImageNet. Classification datasets do not have any bounding box information, but datasets like ImageNet are very large with a wide variety of classes. They did this by only reporting a classification loss when applied to classification datasets, whereas for object detection datasets they would also be reporting an object loss to improve bounding box fit. Using this method they were able to detect over 9000 different objects in real-time.

In 2018, the same authors who published YOLOv2 published YOLOv3 (Redmon and Farhadi 2018). This version includes a number of incremental improvements to YOLOv2, including a new backbone, *Darknet-53*, in addition to utilizing feature pyramid networks in order to upsample the feature maps to detect objects at three different scales. It is a bit slower on inference than YOLOv2 but is in turn more accurate. This is the last version of YOLO created by the original authors, as they at the end of their paper announced their ethical concerns regarding computer vision research being used for military applications and to the detriment of personal privacy.

YOLOv4, released in 2020 by different authors (Bochkovskiy, C. Wang and Liao 2020), introduced several improvements over YOLOv3, starting with their so-called *bag of freebies*. These are improvements that strictly concern the training regime, and therefore have no impact on inference speeds. This includes data augmentation techniques that create more variation in the input data before being fed into the model for training, for example mixing together multiple images in random configurations. In addition to these, changes were also made to the architecture of the model which resulted in a model which is more accurate than YOLOv3, and much faster than other state-of-the-art models with comparable accuracy.

Shortly after YOLOv4 was released, YOLOv5 was released by the company *Ultralytics* (Jocher et al. 2022). It is first and foremost an implementation of YOLO in the PyTorch framework and is not very different from YOLOv4 in its architecture. The main contribution of YOLOv5 is its ease of use for machine learning practitioners. It is easier to configure as it uses the more readable YAML configuration files, and provides ready-to-run scripts for training, validation, and inference, in addition to offering several different model variations with different trade-offs between inference speed and accuracy. YOLOv5 has not released an official paper, which makes it difficult to precisely compare its performance with the other versions of YOLO, but they utilize many of the same improvements as YOLOv4, with even more extensive data augmentation techniques, and similar improvements in model architecture.

2.2 Previous masters theses

(Muribø 2019) examined how well sheep could be detected by the object detection model YOLOv3 in UAV images. He additionally examined whether treating all colors of sheep as a single class or treating them as different classes were preferable in terms of locating the most sheep. He also examined whether tweaking the input resolution post-training could improve performance. The model was trained on resolutions between 608x608 and 896x896 with random scaling between these resolutions. He found that representing sheep as one super-class comprising all colors outperformed representing differently colored sheep as different classes. Between the input resolutions tested (608x608, 832x832, and 1024x1024), he found that 832x832 gave the best results. The best performing model achieved a mAP@0.5 of **0.94** and a recall of **0.99** at a confidence threshold of 0.1. He goes on to theorize that the reason for performance dropping on an input resolution of 1024x1024 could be due to the fact that the model only trained on images significantly smaller than this.

(Kaarud, Nordvik and Paulsen 2020) proposed a complete system for sheep retrieval, from the flight planning of the drone, transferring the images to a server for performing object detection to locate sheep, and presenting the results to the user. They also performed tests of object detection models on the data they had available taken from a DJI Mavic 2 Enterprise Dual drone to evaluate its performance using different techniques. They used the regular and thermal images taken by the drone, and evaluated two different object detection models; YOLOv3, and YOLOv3-tiny. Additionally, they trained the models on the regular images both split into a grid of tiles and with no splitting, as well as at different resolutions. The performance achieved by the model when trained on images split into tiles was generally better than when trained on the full images, especially when the drone was flying at higher altitudes where the sheep will appear smaller. They also found a resolution of 832x832 to be optimal, which falls in line with Muribøs results. The performance of YOLOv3-tiny was surprisingly good in comparison to the regular YOLOv3, only showing a marginal decrease in mAP while the inference speed of YOLOv3-tiny was much faster.

(Furseth and Granås 2021) used different object detection models from the YOLOv5 family to evaluate their inference speed and detection performance running on mobile devices. They used a smartphone to test the different models' inference time for mobile devices. They used a similar dataset to this thesis, both Red-Green-Blue (RGB) and Multi-Spectral Dynamic Imaging (MSX) images of sheep taken with a DJI Mavic 2 Enterprise Dual drone. MSX is an image format where the thermal camera on the drone is combined with the regular camera. For the RGB images, they compare the YOLOv5s and YOLOv5m model, where the YOLOv5m model is larger as it has more trainable parameters. They compared these models at different resolutions and used both downscaling and tiling to adjust the resolution. They found that YOLOv5m only had a marginal improvement in performance over YOLOv5s overall, although the difference was much greater at lower resolutions. The inference time of YOLOv5m was, however, significantly higher when running on a smartphone. Two interesting results that are worth highlighting are:

1. Training on tiled images yielded significantly better performance than training on downscaled images. Additionally, training on higher resolutions also yielded significantly better performance up to a point.
2. Having trained the model on tiled images, running inference on high-resolution non-tiled images outperformed running inference on tiled images.

2.3 Object detection with UAVs

(Petso et al. 2021) applied object detection methods to wildlife monitoring using UAVs. For monitoring wildlife, the altitude of the UAV is very important because drones flying at low altitudes tend to disturb the animals, which is not desirable. However, as the altitude of the

UAV increases, the animals will appear smaller in the images and will therefore be more difficult to detect. For these reasons, they examined the performance differences in object detection for images taken at different altitudes to quantify what sort of detection performance can be expected at different altitudes for a use case such as this. They set out to detect four different wild African animals; giraffes, white rhinos, wildebeests, and zebras. The animals were labeled both individually, as well as additional classes for herds of the same animals. The images were taken at nine different altitudes between 15 m and 130 m. The dataset consisted of a total of 8,659 images. They used two different object detection models; YOLOv3 and YOLOv4. For low altitudes, both models performed very similarly; YOLOv3 achieving a mAP of **0.863**, and YOLOv4 achieving a mAP of **0.875** at an altitude of 15 m. At higher altitudes the difference was much greater; YOLOv3 achieving a mAP of **0.639**, and YOLOv4 achieving a mAP of **0.765** at an altitude of 130 m. In Figure 2.1, the mAP achieved at different altitudes for the two models is shown. For YOLOv4, the drop in mAP is close to linear; however, for YOLOv3, the drop in mAP appears to accelerate after about 90 m of altitude. Their results indicate a negative correlation between detection performance and altitude when trying to locate animals in images from UAVs.

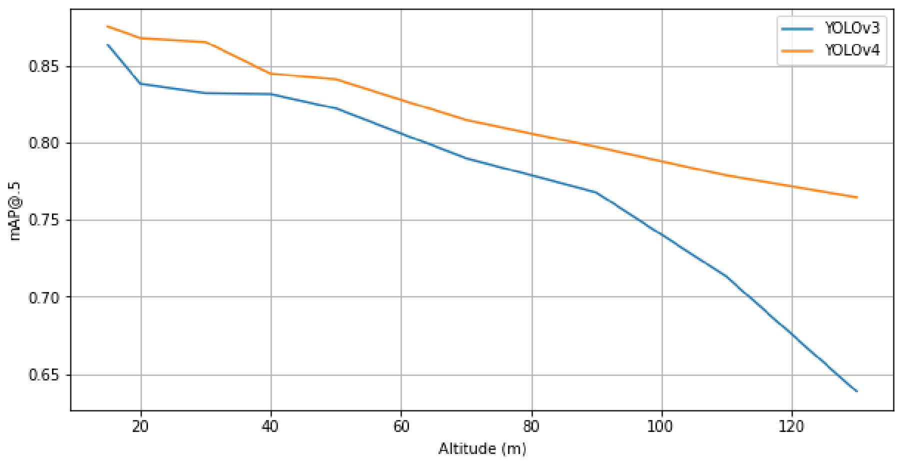


Figure 2.1: mAP achieved for images taken at different altitudes
Note. The data is from Table 1 in (Petso et al. 2021)

(S. Wang 2021) compared different versions of YOLO intending to determine which model is best suited for running in real-time onboard a UAV. They evaluated and compared the performance of YOLOv3, YOLOv3-tiny, YOLOv3-SPP3, YOLOv4, and YOLOv4-tiny on the Pascal VOC dataset. The *tiny*-models are specifically made to have a faster inference speed, as a consequence, they tend to have fewer trainable parameters which cause the performance to suffer as a result. The metrics used for comparison were mAP and FPS of inference. To test the performance running on a drone, they tested the models on the *XTDrone* simulation platform. Even though YOLOv4 had the best performance by far in terms of mAP, they concluded that for real-time detection on a drone, YOLOv3-tiny was the most optimal. YOLOv4-tiny had a significantly higher inference speed than YOLOv3-tiny; however, its mAP was also significantly lower, making YOLOv3-tiny the best compromise between speed and performance for this use case.

(Varga and Zell 2021) proposed a tiling method for input images for object detection models. They examined whether it is beneficial in terms of detection performance to divide UAV images (which are typically sparsely labeled) into tiles. Additionally, they examined the causes of improvement. They explore different tiling sizes and different levels of overlap between tiles. They test different configurations of *EfficientDet*, and *CenterNet* models, in addition to YOLOv4 on the *VizDrone*, *SeaDroneSee*, and *DOTA* datasets. When images are split into tiles, they also discard empty tiles containing no labels. They found that training on tiled data consistently outperformed training on non-tiled data for every model and dataset they examined. They found a reduction in the background bias to be a significant part of the performance improvement, as many empty tiles are discarded, meaning that the portion of the unlabeled background is reduced. Additionally, they found training on tiled data to be more efficient as training on full-sized high-resolution images requires much more memory during the backpropagation step in the training process. For sparsely labeled images such as UAV images, objects can appear quite small and therefore benefit from maintaining a high resolution as opposed to downscaling the images, which is an often-used method in other cases. When running inference on the trained models, they used the full images as opposed to tiling them as was done during training. Running inference requires much less memory than training, and tiling is therefore not necessary then. Out of the different resolutions they tested, they found a tile size of 512x512 to be optimal. For the degree of overlap, they found a minimum of 15% overlap to be optimal, with more overlap neither increasing nor decreasing performance.

2.4 Synthetic data

(Nowruzi et al. 2019) examined the performance achieved by the object detection model *SSD-MobileNet* on detecting cars and persons in autonomous driving datasets. They set out to discover whether adding cheaply generated synthetic data to a limited real dataset improved detection performance. They evaluated combinations of three real datasets and three synthetic datasets in different fractions by reporting their average precision and recall. They also compared combining the real and synthetic data into a mixed training set versus first training on only the synthetic data and then fine-tuning on the real data. Their two most interesting findings are that:

1. Fine-tuning on real data after training on purely synthetic data tended to outperform mixed training.
2. Fine-tuning on real data after training on purely synthetic data consistently outperformed training on only real data of the same amount.

Using more real data in combination with synthetic data also tended to increase performance overall. Additionally, the largest improvement was seen in the average recall, compared to a smaller improvement in the average precision. In fact, the average recall achieved by using only a fraction of the data to fine-tune a model trained on synthetic data was in many cases higher than using 100% of the real data with no synthetic data. They also mentioned that based on findings in previous literature realistic sensor distortion and environment distribution are more important than photo-realism, and their results further support this theory as the most photo-realistic synthetic dataset they used did not in fact produce the best performance, suggesting that other factors are more important.

(W. Liu, J. Liu and Luo 2020) proposed a method of generating realistic synthetic data using the JavaScript-based 3D rendering system Three.js, followed by a style transfer using a Generative Adversarial Network (GAN) called CycleGAN to make the synthetically generated images more realistic looking. In this case, they were trying to detect aircraft from aerial images, which has the challenge of detecting objects at different scales depending on the altitude of the camera. To generate the synthetic images, they used 3D models of aircraft over different background images, they also added a directional light representing the sun and also added some fog. The camera in the scene was looking down from above to simulate aerial photographs. They used three real datasets; *NWPU VHR-10*, *UCAS-AOD*, and *DIOR*, and trained two object detection models; Faster R-CNN, and R-FCN. They evaluated and compared the performance between using purely synthetic data and purely synthetic data with style transfer from CycleGAN. They also evaluated and compared using different amounts of real data in addition to the synthetic data. When real data was used, the model was first trained on just the synthetic data, and then it was fine-tuned on the real data. For the Faster R-CNN model trained on the *NWPU VHR-10* dataset, they found that using only synthetic data without style transfer yielded a mAP of **0.450**, and using only synthetic data with style transfer yielded a mAP of **0.605**. For context, using only real data, the model achieved a mAP of **0.641**. This indicates that using style transfer on synthetic data can be beneficial. Using the synthetic data with style transfer and the real data for fine-tuning gave a mAP of **0.685**, meaning that adding synthetic data can increase performance over just using the real data which is available. It was also clear from their results that using more data for fine-tuning increases performance. There was also a clear trend that the less real data was used, the bigger the impact of adding synthetic data was.

(Borkman et al. 2021) introduced the Unity Perception package for the popular game engine, Unity. This package provides a toolset for generating random scenes in 3D as well as the tools to automatically capture and label images taken by cameras positioned within the scene. In their paper, they give an overview of how these tools work as well as perform an experiment comparing the performance of an object detection model with and without synthetic data generated by Unity Perception. For the experiment, they use the object detection model Faster R-CNN with a *ResNet50* backbone, pretrained on the *ImageNet* dataset. They created the UnityGroceries-Real dataset, consisting of 1267 images of 63 classes of grocery items for the real dataset, and then generated 400,000 synthetic images of groceries for the synthetic dataset. They evaluated the performance of a model trained only on the real training set versus a model trained on the synthetic dataset followed by fine-tuning on the real dataset. They trained several models using different amounts of real data for fine-tuning. The baseline model trained only on the real dataset achieved a mAP@0.5 of **0.719**, and the model trained only on synthetic data achieved a mAP@0.5 of **0.538**. When they used half of the real dataset for fine-tuning the model's performance surpassed the baseline achieving a mAP@0.5 of **0.815**, and when using the full real dataset for fine-tuning they achieved a mAP@0.5 of **0.854**. This shows that cheap synthetic data generated by Unity Perception can be beneficial to object detection performance as enough synthetic data can compensate to some degree for the lack of real data. It is still clear that more real data is always better, but real data is much more expensive and time-consuming to create.

Chapter 3

Theory

In the following sections, some of the theoretical background behind the machine learning method used, and the methods of evaluation will be explained.

3.1 Artificial neural networks

The brain uses biological neurons connected by axons and dendrites in order to perform computations. Artificial Neural Networks (ANNs) are inspired by this structure, using artificial neurons connected together to perform computations (McCulloch and Pitts 1943). ANNs *learn* a mapping between inputs and outputs by being presented with input data, and what its corresponding output should be. This is called supervised learning, and adjusting the weights between neural connections is typically done through backpropagation. This works by computing a *loss*; the error between what the network predicted and what the correct output is, and propagating this loss backward through the layers by differentiating the layers to compute the gradient with respect to the loss, and adjusting the weights in the direction of lowering the loss (Goodfellow, Bengio and Courville 2016).

3.1.1 Multilayer perceptrons

Multilayer Perceptrons (MLPs) are the simplest form of neural network. What characterizes MLPs is that they have multiple layers of neurons, where every neuron is connected with every other neuron in the next layer (fully connected layers). MLPs consist of an input layer, a number of *hidden* layers, and finally, an output layer (Haykin 1999). The structure of this kind of neural network can be seen in Figure 3.1, where the thickness of the lines between the neurons represents the weight of the connections.

The value v_i of a neuron i depends on the outputs from the previous layer $(x_1, x_2, \ldots, x_n)$, the weight of the connections into the neuron i $(w_{i1}, w_{i2}, \ldots, w_{in})$ and the bias b_i of the neuron. The weights between the current neuron and the neurons in the previous layer, as well as the bias, are all trainable parameters. The value of a neuron is computed according to Equation 3.1, where n is the number of neurons in the previous layer (Haykin 1999).

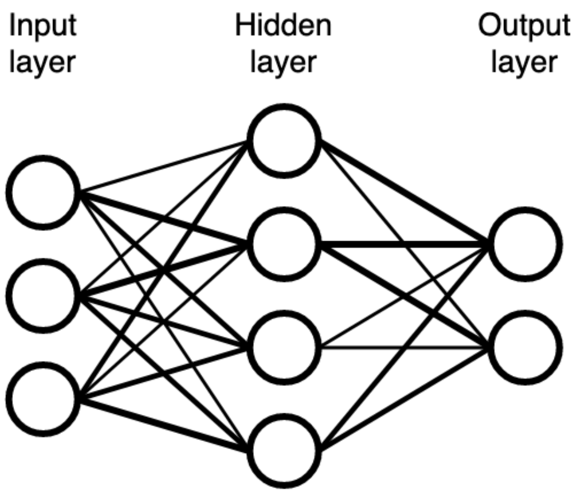


Figure 3.1: Multilayer perceptron

$$v_i = \sum_{j=1}^{n} (x_j w_{ij}) + b_i \tag{3.1}$$

This can be calculated very efficiently since it can be trivially expressed as matrix-vector multiplications that can be computed in parallel on modern hardware. For the network in Figure 3.1, the input layer can be represented by a vector of length 3, and the hidden layer and biases as vectors of length 4. The weights between these two layers can then be represented as a 4x3 matrix. To compute the values of the neurons in the hidden layer, it is simply a matter of performing a matrix multiplication of the weight matrix and the input vector and adding the bias vector at the end.

However, before the values are passed on as the input to the next layer, a so-called activation function is applied to each neuron value. The activation function has to be differentiable in order to be able to perform backpropagation, and it is usually a non-linear function. If a linear activation function is used, the network itself is constrained to being linear, and will not be able to approximate non-linear functions. When using non-linear activation functions between the hidden layers, the neural network can be seen as a universal function approximator (Cybenko 1989).

3.1.2 Convolutional neural networks

Convolutional Neural Networks (CNNs) are traditionally used for image classification, that is, determining to which class a certain image belongs. CNNs typically consist of convolution layers, pooling layers, and fully connected layers. Convolution layers consist of matrices of numbers called kernels, which passes over the input image and outputs feature maps. For each image patch the kernel passes over, it multiplies pixel values with the kernel weights at the corresponding locations, and sums the result; this is illustrated in Figure 3.2. The kernel weights are learned through training, and learn to map specific patterns into features (Goodfellow, Bengio and Courville 2016). The *output* in this figure is an example of a feature map. Each convolution layer is usually followed by an activation function and then a pooling layer.

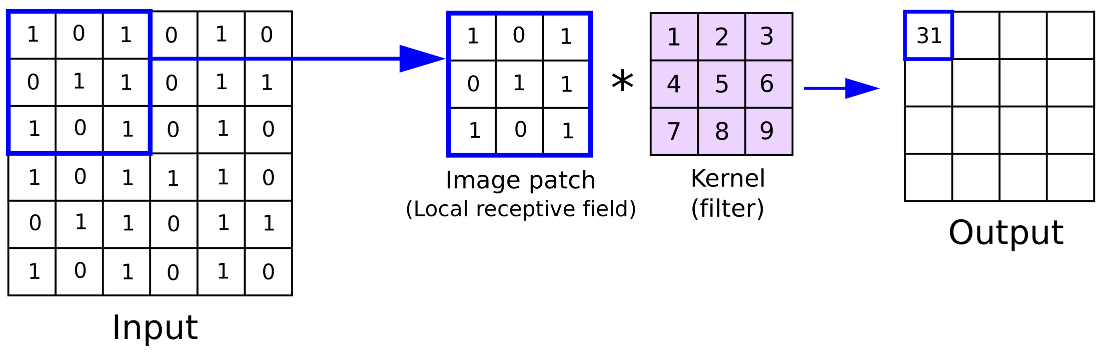


Figure 3.2: Convolution layer
Note. From "Convolutional Neural Networks (CNNs)", by Anh H. Reynolds (https://anhreynolds.com/blogs/cnn.html).

Pooling layers reduce the size of the feature maps by reducing groups of adjacent pixels into a single value. The most common pooling function is *max-pooling*, which takes the maximum value of a group of adjacent pixels (Goodfellow, Bengio and Courville 2016). An example of this process can be seen in Figure 3.3, where a pooling size of two is used, taking the maximum value of 2x2 sections of the image, and thereby reducing the size of each image dimension by a factor of two.

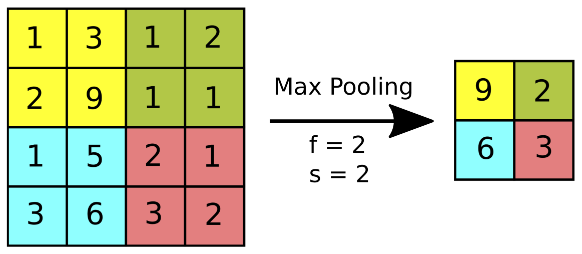


Figure 3.3: Max-pooling
Note. From "Convolutional Neural Networks (CNNs)", by Anh H. Reynolds (https://anhreynolds.com/blogs/cnn.html).

After a number of convolution and pooling layers, the final feature maps are flattened into a vector, and then fully connected layers are used at the end to produce the final class predictions.

CNNs essentially work by extracting features from the image. The first convolution layers extract simple features like edges and corners; deeper into the network these features become more abstract and semantically meaningful. Since the learned parameters are the kernel weights themselves, and the kernels pass over the entire image, CNNs are invariant to translation, meaning that the location of the object in the image is irrelevant. This is a very useful feature for image classification and a key reason why they outperform simple fully connected neural networks for image classification. A fully connected neural network would necessarily map specific pixels with specific weights, making it difficult for the network to generalize when the objects to be classified in an image are translated.

3.2 Object detection

Object detection is the task of identifying and localizing multiple objects within an image. Object detection methods should ideally be able to detect and localize an arbitrary number of objects of an arbitrary number of classes within the same image. Object detection methods typically produce 2-dimensional bounding boxes around each object instance as well as a class prediction and a confidence score for each object prediction. An example of such predictions can be seen in Figure 3.4. There are other approaches to object detection like producing 3D bounding boxes around objects. Additionally, there are segmentation techniques that produce masks around objects with a class prediction instead of simple bounding boxes (Zhao et al. 2018). However, in this thesis, 2-dimensional bounding boxes are used.

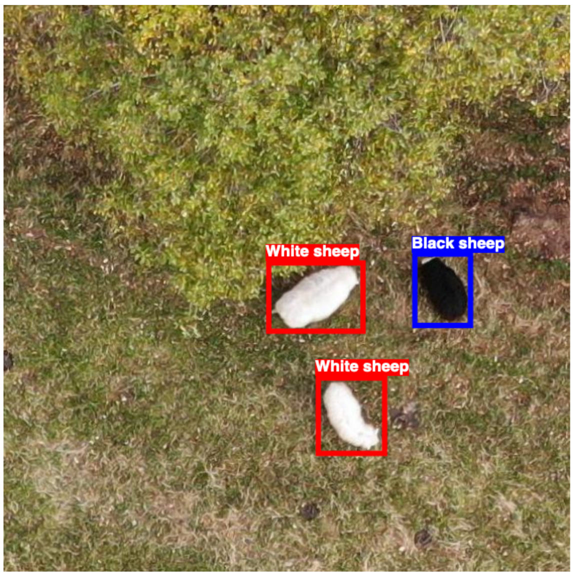


Figure 3.4: Example predictions from an object detection model

3.2.1 Intersection over union

Intersection over Union (IoU) in the context of bounding boxes is a measure of overlap between two different bounding boxes. It is calculated by dividing the area of intersection (overlap) between the two boxes by the union area of the two boxes. The intersection area can be calculated directly by knowing the bounding box coordinates. To calculate the union area, sum the individual areas of the bounding boxes, and then subtract the intersection area (Everingham, Van Gool et al. 2010). This can be seen intuitively in Figure 3.5, where the shaded blue regions represent the areas in each part of the fraction.

When two boxes overlap perfectly this will yield a value of 1, and if two boxes do not intersect at all it will yield 0. This is a very important measure in the context of object detection as it is used to determine whether predictions are correct depending on whether the overlap between the predicted bounding box and the ground truth bounding box is large enough; the IoU must be greater than a predetermined threshold, e.g. 0.5 (Everingham, Van Gool et al. 2010). It is also used for non-maximum suppression.

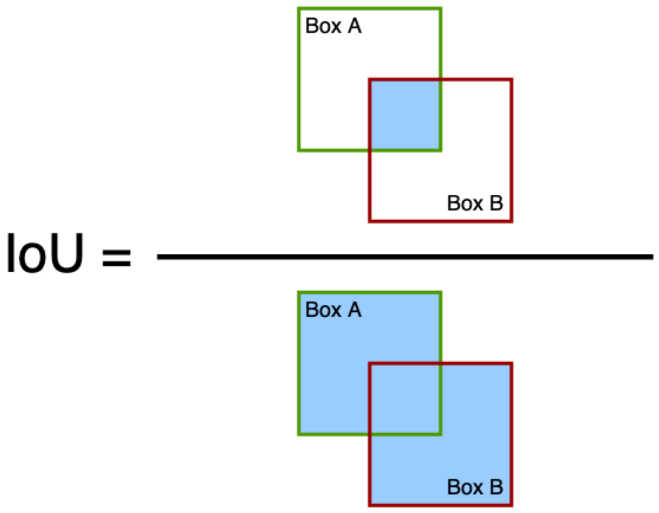
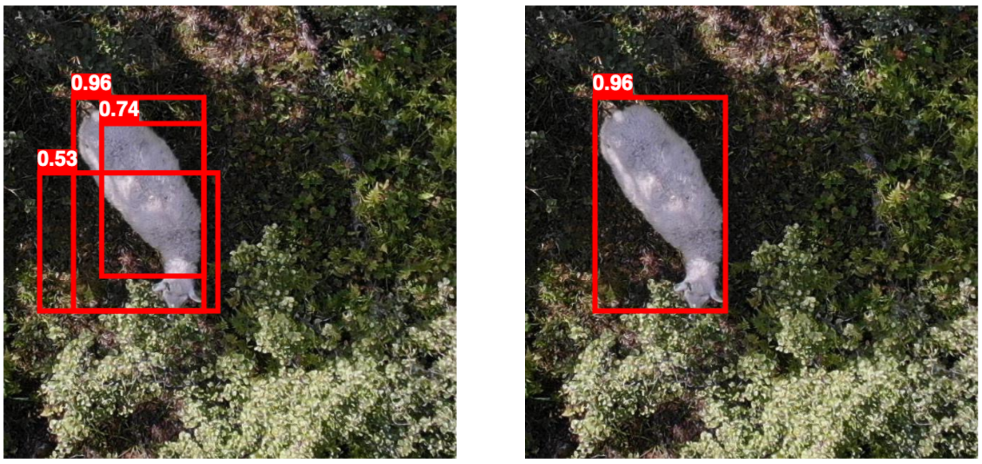


Figure 3.5: Intersection over union of two bounding boxes

3.2.2 Non-maximum Suppression

When an object detection model makes predictions, it may often produce a large number of partially overlapping boxes related to the same object. However, one true object should ideally only have one prediction associated with it; this is where Non-Maximum Suppression (NMS) comes in. In addition to a class prediction, object detection methods typically produce a confidence value for each prediction. For each overlapping prediction, only the prediction with the highest confidence is desired, this is illustrated in Figure 3.6.



(a) Candidate boxes before NMS (b) Final prediction after NMS

Figure 3.6: Bounding box prediction before and after NMS

First, an IoU threshold must be set, which will determine whether two boxes overlap enough to be considered to predict the same object. The raw predictions from the object detector are considered candidate boxes at this stage and are sorted descending by their confidence scores. Until the list of candidate boxes is empty, take the first element from the list and add it to the list of output predictions. Next, look through the list of candidate boxes and remove those which have an IoU with the added box over the threshold set earlier. This is repeated until the list of candidate boxes is empty (Buil 2011). Since the list of candidate boxes is sorted by confidence, only the highest confidence prediction for each object will be added as the overlapping boxes of lower confidence are removed before they can be added to the list of output predictions.

3.3 Evaluation metrics

In standard image classification, an image is classified either correctly or it is not, meaning that evaluating performance can be as simple as reporting the accuracy as a fraction of correct predictions over the total number of predictions. For object detection, it is slightly more complicated because there is more than one way to be incorrect. A False Positive (FP) is when the model predicts an object that is not actually present, and a False Negative (FN) is when the model does not predict an object that is present. Correct predictions are either True Positives (TPs) where an object is present and correctly predicted, or True Negatives (TNs) where objects are absent and correctly omitted from prediction. However, true negatives represent correctly predicting the background, which naturally does not contain any boxes, and therefore cannot be quantified in the same way; it is therefore not a very useful measure in this context.

Because these three different measures (FP, FN, TP) are obtained when comparing predictions from object detectors to the ground truth, the metrics that are commonly used to evaluate object detection models utilize these measures.

It is also relevant to consider the confidence score when looking at these metrics. When using an object detection model in practice, one would often want to exclude predictions below a certain confidence threshold. The metrics presented in the following subsections can be used to find a suitable confidence threshold for the problem at hand.

3.3.1 Precision

Precision measures how many of the total predictions made, are correct predictions. In the context of object detection, it is a measure of how many of the predicted bounding boxes actually correspond to an object in the image. Precision ranges between 0 and 1, where 0 means that all predicted boxes are false positives, and 1 means that all predicted boxes are true positives. This means that the model can get a precision of 1 if it only makes one prediction as long as that one prediction is correct, even if there are hundreds of other objects in the image that are not detected. Precision is calculated according to Equation 3.2 (Taha and Hanbury 2015).

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

How does the confidence threshold relate to precision? Typically, a lower confidence threshold will include more false positives. This is because low-confidence predictions that our model is very uncertain about are not discarded. When using a high confidence threshold, all but the most confident predictions are discarded, which only leaves the predictions that are most likely to be true positives, and hence, precision should increase when the confidence threshold increases. Figure 3.7 shows the general relationship between precision and confidence, the exact shape of the curve varies in practice.

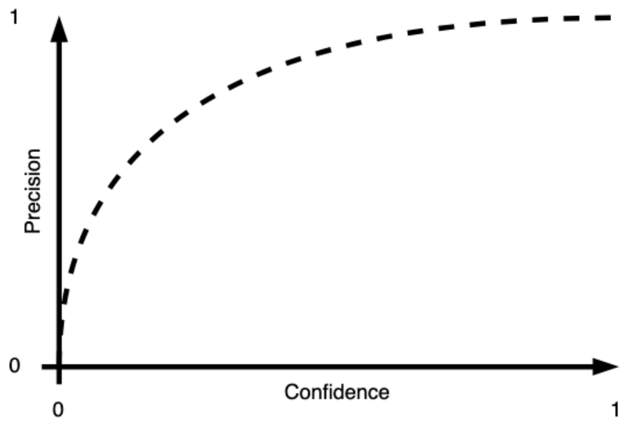


Figure 3.7: Relationship between precision and confidence

3.3.2 Recall

Recall measures how many of the true examples were actually predicted by the model. In the context of object detection, it measures how many of the true objects had a bounding box prediction from the model. Recall ranges between 0 and 1, where 0 recall means that none of the true objects were detected, and 1 recall means that all of the true objects were detected, irrespective of the number of false positives. Equation 3.3 shows how recall is calculated (Taha and Hanbury 2015).

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

Recall has the opposite relationship with the confidence threshold as precision. A lower confidence threshold will include more predictions, making it more likely to correctly predict more true boxes. When the confidence threshold increases, the number of false negatives typically increases since some of the predictions of low confidence will still be correct, but they are no longer included. That is why recall typically has a negative correlation with the confidence threshold. Figure 3.8 shows the general relationship between recall and confidence.

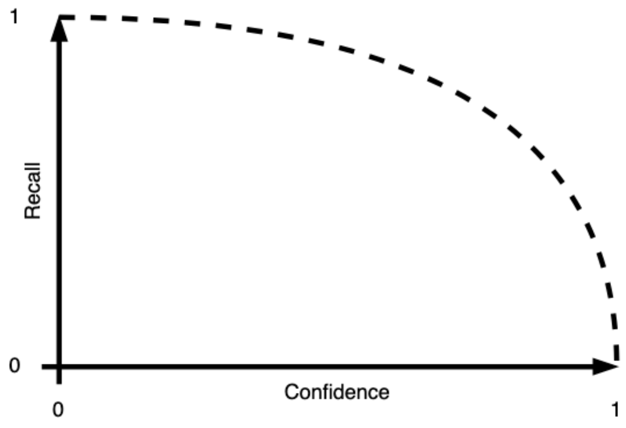


Figure 3.8: Relationship between recall and confidence

3.3.3 F-score

It is clear that precision and recall by themselves are not sufficient; however, they paint a clearer picture when considered together. The F-score combines precision and recall into a single metric. The standard F-score, called the F_1-score is the harmonic mean between precision and recall, and values each measure equally. The F_1-score is calculated according to Equation 3.4 (Taha and Hanbury 2015).

$$F_1 = \frac{TP}{TP + \frac{1}{2} \cdot (FN + FP)} \tag{3.4}$$

The F-Score also produces an output between 0 and 1, where 0 indicates that either measure is 0, and 1 indicates that both measures are 1.

There is also a more general F-score; the F_β, that uses a positive factor β to put more weight on either precision or recall. A β of 1 yield the standard F_1-score, a β of 2 means that recall is twice as important as precision, and a β of $\frac{1}{2}$ means that precision is twice as important as recall. The more general F_β with the additional factor β is calculated according to Equation 3.5 (Taha and Hanbury 2015).

$$F_\beta = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP} \tag{3.5}$$

The F-scores relationship with the confidence threshold is a little more complicated. Since it depends on both precision and recall, which have an opposite relationship to each other with respect to the confidence threshold, the F-score does not simply increase or decrease with the confidence threshold. However, since precision is usually close to zero at low confidence thresholds, and recall tends to be close to zero at high confidence thresholds, the F-score is usually close to zero at both ends. It will then be at its highest somewhere in between, depending on the weighting β, and the shape of the precision and recall curves. There are many possible shapes of the F-score curve, some simplified examples are illustrated in Figure 3.9.

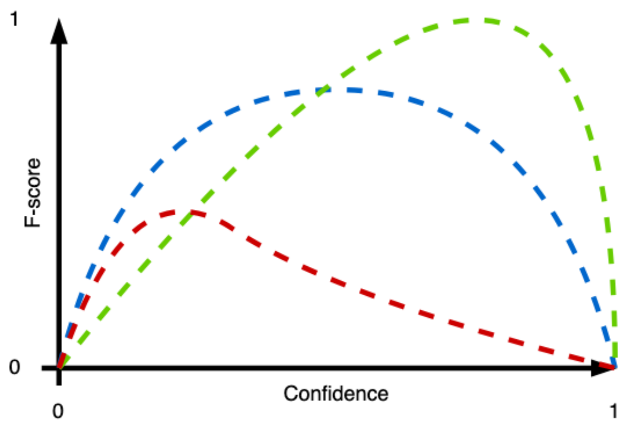


Figure 3.9: Possible relationships between F-score and confidence

3.3.4 Precision-recall curve

Another useful metric is the precision-recall curve. This shows the relationship between precision and recall directly, that is, for a given recall level, what kind of precision does the model get. This can be calculated by generating a set of recall levels between 0 and 1, and checking the maximum precision at or above each level (Everingham, Van Gool et al. 2010). Remember that precision and recall have an inverse relationship, so when the recall increases, the precision tends to decrease. If a higher precision is found at a recall level above the currently examined one, it is always preferable to go to that higher recall level since it is essentially a free lunch. That is why precision is interpolated this way. This means that the interpolated precision-recall curve is monotonically decreasing, but not strictly decreasing, as it can be flat in certain sections (Encyclopedia of Mathematics 2020). The general shape of a precision-recall curve can be seen in Figure 3.10.

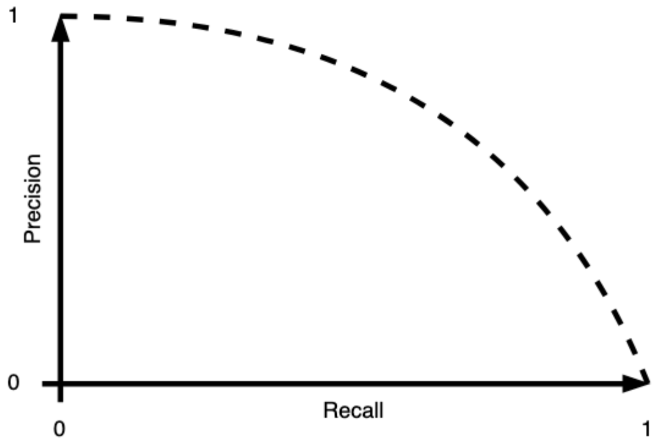


Figure 3.10: Relationship between precision and recall

This interpolated precision-recall curve is also used to calculate other metrics like Average Precision (AP). AP is calculated as the weighted average precision at each recall level, where the increase in the recall level from the previous value is used as the weight. For a precision-recall curve with a constant spacing between recall levels, this is equivalent to taking the mean of the precision-recall curve (Everingham, Van Gool et al. 2010).

3.3.5 mAP@.5

mAP@.5 has a rather cryptic-sounding name but is a very widely used metric in the field of object detection. Breaking down the name; mAP stands for *mean average precision*, @.5 means *at 0.5 intersection over union*. It is calculated by creating a precision-recall curve using 0.5 as the IoU threshold for determining whether predictions are correct or not. Next, the average precision is calculated for each class in the way explained above, and finally, take the mean of the average precision for each class. For a single class object detector, AP and mAP are equivalent (Everingham and Winn 2010).

Since the precision/recall curve is used, which reports the precision at different recall levels, by taking the mean of this curve both precision and recall are taken into account in a single number, which turns out to be a very effective way to compare the performance of different object detection models on the same dataset.

3.3.6 mAP@.5:.95

mAP@.5:.95 is very similar to mAP@.5. To explain the naming; @.5:.95 means at IoU from 0.5 to 0.95, at steps of 0.05. This means that the mAP is evaluated at 10 different IoU thresholds between 0.5 and 0.95, the average of these values is reported as the metric (Everingham and Winn 2010). This is a good metric for the same reasons as the mAP@.5 metric. However, in addition to taking the precision and recall into account, this metric also measures how well the bounding boxes fit the ground truth boxes since the higher IoU thresholds will result in more errors if the predicted bounding boxes do not have sufficient overlap with the ground truth. For this reason, they do not measure precisely the same thing, and mAP@.5:.95 is therefore not a substitute for mAP@.5, and one would typically report and compare both metrics.

3.4 YOLOv5

YOLOv5 is a family of object detection models that produce 2D bounding boxes as predictions. YOLOv5 offers multiple model variations of different sizes in terms of no. of parameters that affect its speed and accuracy. In general, YOLOv5 consists of four main components: input augmentation, the backbone, the neck, and the head. The latter three components are what make up the actual neural network, and its architecture can be seen in Figure 3.11. YOLOv5 uses three different loss functions:

1. **Box loss** measures the error in box boundaries between the prediction and ground truth.
2. **Object loss** measures the error in confidence of object presence.
3. **Class loss** measures the error in class prediction.

Input augmentation is used to improve the model's generalizability; many augmentation techniques are applied in this stage. Mosaic augmentation takes several input images and puts them together into a mosaic of a standard size. The individual images are also transformed randomly using affine transformations (rotation, scale, shear, and translation), images can also with a certain probability be flipped. Finally, the hue, saturation, and value of the image colors are augmented as well (Li et al. 2022).

The backbone is responsible for feature extraction, and YOLOv5 uses CSP-Darknet as its backbone. Although it is more complicated and has some other components in addition to convolutional layers, it is essentially a CNN, which as explained in subsection 3.1.2 is able to extract feature maps from the input. However, it lacks the flattening and fully connected layers at the end for classification, as its job is only to produce the feature maps and hand this over to the neck (Li et al. 2022).

As mentioned in subsection 3.1.2, the early feature maps of the CNN have a high degree of localization and low semantic information, and deeper into the CNN, the features have more semantic information, and because of pooling, the spacial information gets compressed. The function of the neck of YOLOv5 is to preserve the spatial information of features, otherwise, precise localization of objects would be difficult. For this, YOLOv5 uses a variation of PANet. This works by propagating strong semantic features from the high feature maps into the low feature maps and propagating strong localization features from the low feature maps into the

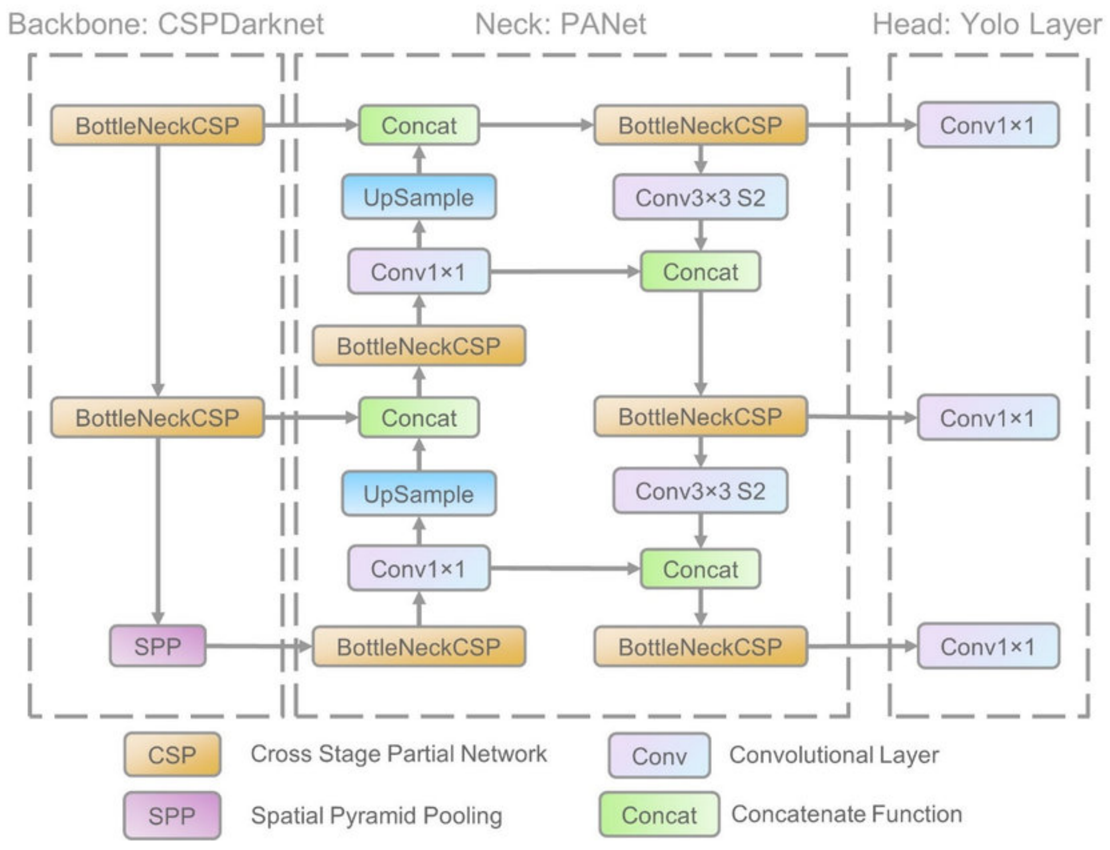


Figure 3.11: The network architecture of YOLOv5
Note. From "A Forest Fire Detection System Based on Ensemble Learning", (p. 5), by Xu et al., 2021 (https://www.researchgate.net/publication/349299852_A_Forest_Fire_Detection_System_Based_on_Ensemble_Learning).

high feature maps (Xu et al. 2021).

Finally, the head (called the YOLO layer) is responsible for predicting the labels. It generates three different feature maps of different sizes to be able to predict small, medium, and large objects (Xu et al. 2021). The bounding box prediction utilizes anchor boxes. Anchor boxes are a set of predefined boxes of different aspect ratios. The model will try different anchor boxes and select the one with the best fit, and the actual numerical prediction is an offset from the anchor box as opposed to predicting the coordinates directly. This is done because it simplifies the problem, and models utilizing this technique typically perform better on standard benchmarks (Zhong et al. 2020). Before the actual training begins, YOLOv5 will analyze the training set and use K-Means clustering to automatically find appropriate anchor boxes for the dataset.

Chapter 4

Sensor data

The dataset of drone images consists of 2,125 aerial images containing 20,431 total instances of sheep. The images have a resolution of 4,056×3,040. The images were captured with a DJI Mavic 2 Enterprise Dual, which has a regular camera as well as a thermal camera. Only the visual camera images were used. The visual camera has a horizontal Field of View (FOV) of 85°. (DJI 2022)

4.1 Data capture

The data was captured in a few different locations over different sessions. When an image is taken, a lot of metadata is stored along with the image, including capture date and time, GPS location, and elevation. Insight into these parameters will be useful when generating synthetic data. In Figure 4.1, the locations where images were taken can be seen in the form of a heat map.

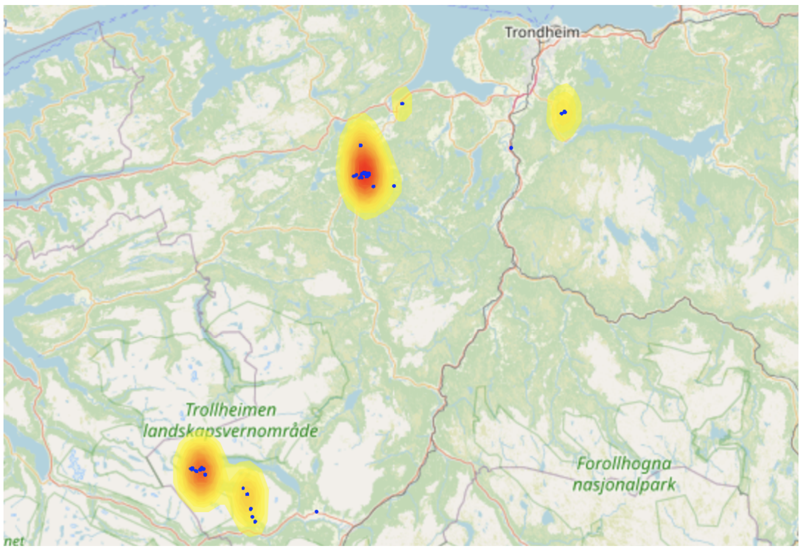


Figure 4.1: Heatmap showing capture locations

The majority of images are clustered around two different locations, Storlidalen in Oppdal, and Orkanger, over relatively small areas. This likely limits the variance of background terrain which can be expected in the dataset; capturing data from more locations could potentially improve the generalizability of detection. Next, the day of the year when images are captured affects vegetation, climate, and how much sun is present at different times. This can be seen in figure Figure 4.2.

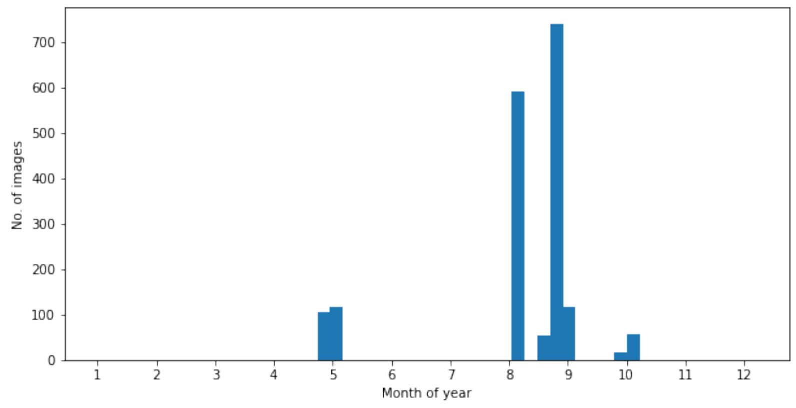


Figure 4.2: Day of year when images were captured

The vast majority of images were captured between August and October, with only a few images captured around April/May. Since sheep are usually retrieved around fall, this is okay. The time of day when images are captured has a significant impact on the resulting images as it determines how much sunlight is present as well as the angle of the sun; which in turn determines how shadows are cast over the terrain and sheep. In Figure 4.3, the time of day when images are captured can be seen.

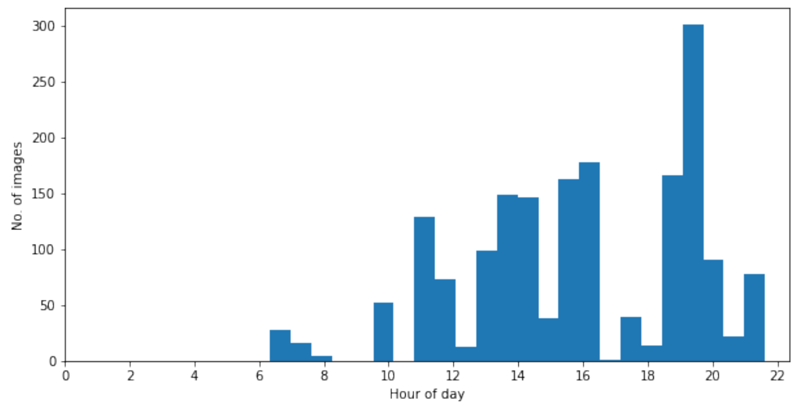


Figure 4.3: Time of day when images were captured

The majority of images were captured from mid-day to late in the evening, with no images captured after midnight before 06:00, and only a few images were captured early in the morning. For a practical case, having no images during the night is probably okay. However, since a farmer looking to retrieve sheep would want to go out as soon as possible after images are taken before the sheep move too far, it would probably be useful to capture images early in the morning, so that there is time to go out the same day. Having more data for adequate detection between 06:00 and 10:00 would likely be useful for a practical case.

4.1.1 Elevation

The GPS data also includes elevation, which is based on signals from at least four different satellites. This gives a height in meters above mean sea level. However, this in and of itself is not very useful, as the height above ground level is much more interesting. To calculate this, the ground height of each GPS coordinate is needed. In Norway, this data is available from Kartverket at hoydedata.no. This service uses a Universal Transverse Mercator (UTM) projection at zone 33; however, the GPS reports latitude and longitude based on the World Geodedic System 1984 (WGS84). Therefore, the GPS coordinates are first converted into UTM33 using the coordinate transformation software *PROJ* (PROJ contributors 2022). Then, each position is queried at hoydedata.no to get the ground elevation above mean sea level at that position. To get the altitude above the ground then, simply subtract the ground elevation above sea level from the GPS elevation above sea level. The distribution of capture elevations above ground level can be seen in Figure 4.4.

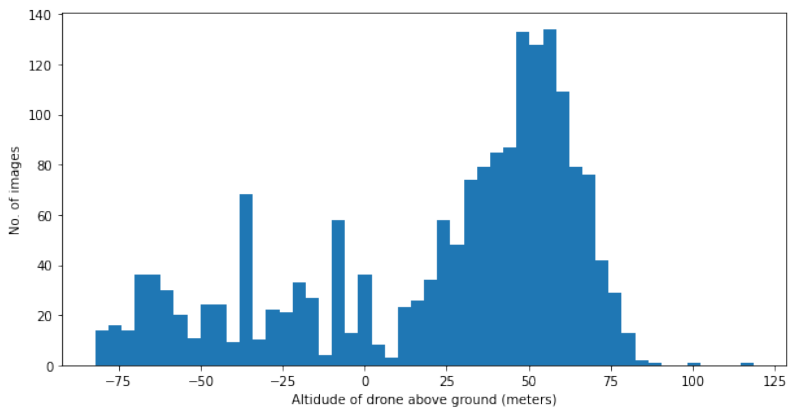


Figure 4.4: Elevation of the drone above ground

The results of these calculations are clearly not perfect, as a large number of images appear to be taken below ground level, which is obviously incorrect. However, the majority of images appear to have a reasonable elevation of between 30 to 60 meters. Several factors could be contributing to these errors. First and foremost, there is a margin of error in the latitude/longitude output from the GPS (Ünsalan 2020), meaning that the GPS could report a mostly correct elevation measurement with an incorrect latitude/longitude location. This would result in sampling the ground elevation at the incorrect location which is then subtracted from the GPS elevation. This would then contribute to the error in altitude above ground. Also, the elevation measurement from the GPS has a margin of error, which is typically larger than the margin of error for the latitude/longitude (Al-Bukhaiti 2018), meaning that the GPS could report a mostly correct latitude/longitude position with an incorrect elevation. Additionally, the ground height data from Kartverket is sometimes calculated using image matching, particularly in larger areas above the tree line. Image matching is less accurate than laser scanning which they would otherwise use (Kartverket 2022). This could be a contributing factor to the error in ground elevation, which would propagate to the error in altitude above ground.

4.2 Labeling

The images were hand-labeled by the master's students working with this dataset. The sheep were labeled into four different classes based on their color (white, gray, black, and brown). A small number of sheep were labeled as a fifth class; occluded, without any color information. Since it was necessary to know both the color and whether or not sheep were occluded, the number of classes had to be increased to eight; one for every color when the sheep was not occluded, and one for every color when the sheep was occluded. The labels were then modified where necessary. To clarify, a sheep is defined as *occluded* when there is something semi-transparent partially covering it up, like leaves or grass which can be seen through; not if a sheep is standing behind a rock and only half of the sheep is visible; this is not considered occlusion in this context.

Some cases were difficult to label; in some cases, the color was difficult to determine, e.g. whether a sheep is gray or black is sometimes ambiguous. In addition to this, how much occlusion is necessary to define it as such is somewhat ambiguous, and the author's best judgment was used in determining this instead of trying to define occlusion rigorously. In a few rare cases, like when the drone was flying at a high altitude, the picture was somewhat blurry, or when partially occluded, it could be difficult to tell whether a sheep was present at all. These cases did not constitute a significant portion of the data though, and likely had a small impact on the results. However, due to the factors mentioned above, it is important to keep in mind when interpreting the results that the distinction between different classes is not perfect.

4.3 Class distribution

The sheep were classified in four different colors (white, gray, black, and brown) and whether or not they were partially occluded. The class distribution of the sensor dataset is listed in Table 4.1.

	White	**Gray**	**Black**	**Brown**	**Total**
Non-occluded	12,380	3,515	2,362	1,071	19,328
Occluded	904	149	40	10	1,103
Total	13,284	3,664	2,402	1,081	20,431

Table 4.1: No. of occluded vs. non-occluded sheep for each color

There is a very significant class imbalance in this dataset, where white is by far the most prevalent color, and occluded sheep are much rarer than non-occluded sheep. The expectation based on this information is that the classes with the fewest instances will be the most difficult to detect. This information was also used to inform which classes should be prioritized when generating synthetic data.

4.4 Training, validation, and test split

The dataset was split into a set for training the model, a validation set used to track the performance during training and select an appropriate model without overfitting to the training data, and finally, a test set used to evaluate the performance of the model on data that the model has never seen before. A split of 70%, 15%, and 15% was chosen for the training, validation, and test sets respectively. The naïve approach to splitting the data would be to simply split the images randomly in the above proportions. However, due to the large variety in the number of instances per image, this will not guarantee a proportional split in the number of instances for each set, which is more important. Additionally, the naïve method does not take into account the number of instances for each specific class per image, and given the large numerical disparity between different classes, care should be taken in making sure the data is split proportionally between each class.

A more suitable approach would be a proportionate stratified sampling. However, since only images can be selected and not individual instances, optimal samples cannot be guaranteed. Therefore, the number of instances for each class per image is needed to inform the selection such that an approximately proportional sample for each set can be achieved. Since some of the classes such as *black occluded sheep* and *brown occluded sheep* have an extremely low number of instances, it is necessary to ensure that the validation and test sets get a number of instances at least proportional to the number they optimally should have. Unfortunately, this is likely still too small to make statistically significant observations. In order to accomplish this, a custom split method was used.

Listing 4.1 shows an excerpt of the Python script used for splitting the data. Some global variables were declared earlier in the script; including the list of image label names in the **labels** variable, the number of instances of each class for each label name in the **img_class_count** variable, and the number of total instances of each class in the **class_count** variable. First, a target is assigned based on the split proportions; this is the number of instances per class each set should ideally contain. Next, the list of label names is shuffled to provide randomness. The **calc_diff** function, is a method of calculating the distance between two set assignments based on how many instances of each class are contained in both. An important feature of this function is that the distance for a given class is calculated by division and that this is again divided by the total number of instances for the class in order to normalize the scale of the differences. This is important because the classes with a low target will see a big reduction in the distance by adding only a few instances; that way, underrepresented classes are prioritized when assigning labels. If a class has assigned an equal or greater number of instances than the target, the distance for this class is zero. The function **calc_modified** simply adds class instances to a set. The script iterates over all of the labels, and stores its class instances. For each iteration, it calculates the distance between what is currently assigned and the target for each set. It then assigns the class instances of the current label to each set and again calculates the distance between the modified set and the target. The split with the greatest reduction in distance will be assigned the label. The resulting split is listed in Table 4.2.

```python
split = {"train": 0.7, "val": 0.15, "test": 0.15}

target = {}
for (a_set, proportion) in split.items():
  target[a_set] = {}
  for (clas, count) in class_count.items():
    target[a_set][clas] = count*proportion

labels = files.copy()
random.shuffle(labels)

sets = {"train": [], "val": [], "test": []}

def calc_diff(assigned, target):
  diff = 0
  for clas in target.keys():
    if assigned[clas] >= target[clas]: continue
    diff += (target[clas] / max(assigned[clas], 0.1)) / class_count[clas]
  return diff

def calc_modified(assigned, c_count):
  modified = assigned.copy()
  for clas, count in c_count.items():
    modified[clas] = modified.get(clas, 0) + count
  return modified

assigned = {}
for (a_set, proportion) in split.items():
  assigned[a_set] = {}
  for (clas, count) in class_count.items():
    assigned[a_set][clas] = 0

for label in labels:
  c_count = img_class_count.get(label[:-4])
  if c_count is None: continue

  weights = {"train": 0, "val": 0, "test": 0}
  for a_set in weights.keys():
    start = calc_diff(assigned[a_set], target[a_set])
    modified = calc_modified(assigned[a_set], c_count)
    end = calc_diff(modified, target[a_set])
    weights[a_set] = (start - end)

  winner = max(weights, key=weights.get)
  assigned[winner] = calc_modified(assigned[winner], c_count)
  sets[winner].append(label)
```

Listing 4.1: Python code for dataset splitting

Training set					
	White	**Gray**	**Black**	**Brown**	**Total**
Non-occluded	8,585	2,427	1,638	744	13,394
Occluded	627	87	26	6	746
Total	9,212	2,514	1,664	750	14,140

Validation set					
	White	**Gray**	**Black**	**Brown**	**Total**
Non-occluded	1,910	546	360	165	2,981
Occluded	139	31	6	2	178
Total	2,049	577	366	167	3,159

Test set					
	White	**Gray**	**Black**	**Brown**	**Total**
Non-occluded	1,885	542	364	162	2,953
Occluded	138	31	8	2	179
Total	2,023	573	372	164	3,132

Table 4.2: No. of class instances for training, validation, and test split

4.5 Data preparation

The high resolution of the images posed a challenge during training as the backpropagation algorithm requires a lot of information to be kept in memory. Since the error is propagated backward through the network, the activations of the previous layers need to be kept in memory the whole time. This is especially problematic for CNNs with a large input image, as the kernel activations over the entire image take a lot of space; this also increases with the depth of the network. It is however significantly better than fully connected layers of a similar size (Varga and Zell 2021).

This significantly limits the batch size and therefore the training speed. In this case, training on the full resolution required too much memory for the hardware used and was too slow to be feasible. A common solution to this problem is downscaling the images to a more manageable resolution. However, since the drone flies at high altitudes, the sheep will appear quite small and downscaling will have a significantly negative impact on the performance as the number of pixels for each sheep will be too low (Petso et al. 2021). In the preparatory project for this thesis, the image resolution had a very big impact on the performance achieved on similar images. Additionally, (Furseth and Granås 2021) and (Varga and Zell 2021) both found that training on tiled images yielded a significantly better overall performance than training on downscaled images. For these reasons, the decision was made to divide the images into tiles instead of downscaling. Tiles including no sheep were discarded in this process, in addition to labels with a width or height of fewer than 12 pixels to eliminate noisy labels that are too hard to detect. This has the benefit of keeping the original resolution of the sheep while significantly improving the training speed.

Each image was split into 48 tiles (width was divided by 8 and height was divided by 6). The tiles include a padding of 64 pixels in every direction in order to not clip sheep located on the border between tiles. This ensures that most sheep will be fully visible in at least one tile, as long as they are not too big. However, it also means that the same sheep can appear in multiple tiles. For this reason, the training, validation, and test splitting was done before tiling the images. Otherwise, the same sheep could appear in both the training and test set, which would threaten the validity of the results. An example of the image tiling is visualized in Figure 4.5, where red tiles indicate an excluded section (containing no sheep), green tiles indicate an included section (containing sheep), and the padding width is shown in yellow around the grid lines.

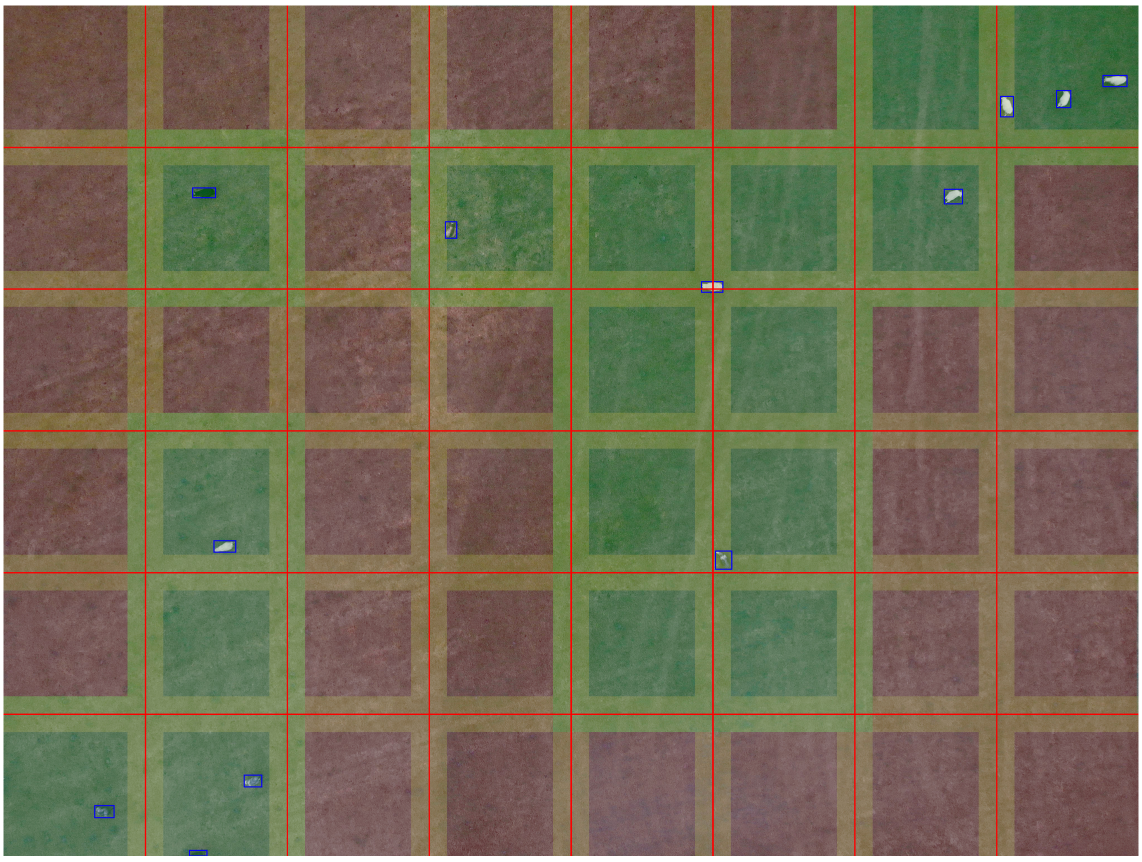


Figure 4.5: Example of image tiling

Chapter 5

Synthetic data

The game engine Unity along with the Perception package was used to generate the synthetic images. The Perception package includes a range of tools for generating randomized scenes and also performs the automatic labeling. First, a so-called scenario is created, where the number of iterations is specified along with a set of so-called randomizers. The randomizers are used to generate a new scene for each iteration in the scenario, drawing random samples from specified distributions and using these values to modify the scene in different ways. The Perception package comes with a few randomizers out-of-the-box, along with tools for creating your own randomizers such as methods for drawing random samples from different distributions and of different data types (Unity Technologies 2020). Some of the built-in randomizers were used; however, most of the randomizers were created specifically for this application.

Using a game engine for generating synthetic data has several benefits.

- The resulting images can be close to photo-realistic depending on the quality of the assets used.
- Realistic lighting and shadows.
- Camera specifications can be emulated.
- Automatic labeling is straightforward and precise.
- Highly controllable and predictable.
- Easy to produce a high degree of variance.

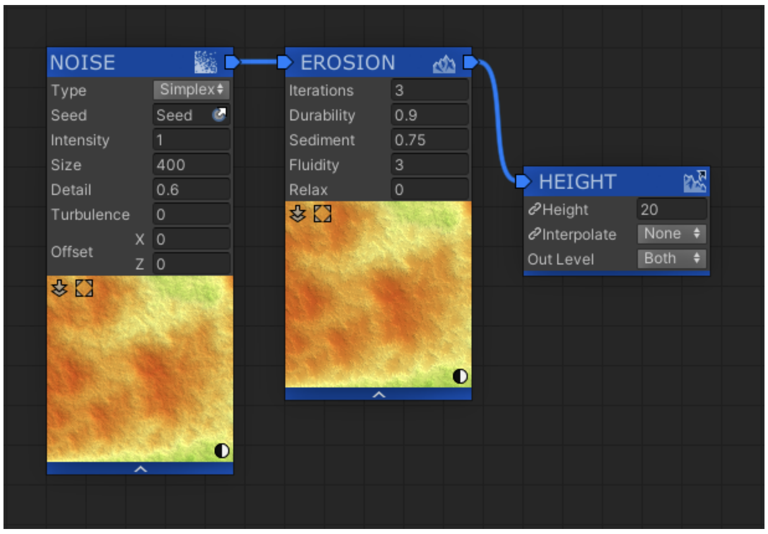
5.1 Randomization

The following subsections describe the different components that make up the scene and how these components were randomized between iterations.

5.1.1 Terrain

To generate random terrain, a package called MapMagic 2 was used. This package can procedurally generate terrain geometry, as well as apply textures and grass. MapMagic 2 generates terrain asynchronously, which results in Unity Perception finishing the iteration before the terrain is generated, as it has no way of knowing when the asynchronous task is finished. A small modification to MapMagic 2's code was therefore necessary in order to generate the terrain synchronously so that it would work within Unity Perception's framework.

MapMagic 2 works by creating a graph that dictates the geometry, textures, and grass. To generate the terrain geometry, the first node in the graph generates simplex noise. This is a type of gradient noise that is suitable for generating terrain. Next, erosion simulates water flow moving material from one location and settling elsewhere as sediment. Finally, the resulting map is applied to the height of the terrain. This can be seen in Figure 5.1. The size of the generated terrain is 60x60 meters.


NOISE
Type Simplex
Seed Seed
Intensity 1
Size 400
Detail 0.6
Turbulence 0
Offset X 0
Offset Z 0

EROSION
Iterations 3
Durability 0.9
Sediment 0.75
Fluidity 3
Relax 0

HEIGHT
Height 20
Interpolate None
Out Level Both

Figure 5.1: MapMagic graph of terrain geometry

An example of the resulting terrain can be seen in Figure 5.2. The only parameter that is changed between iterations is the random seed used to generate the simplex noise.

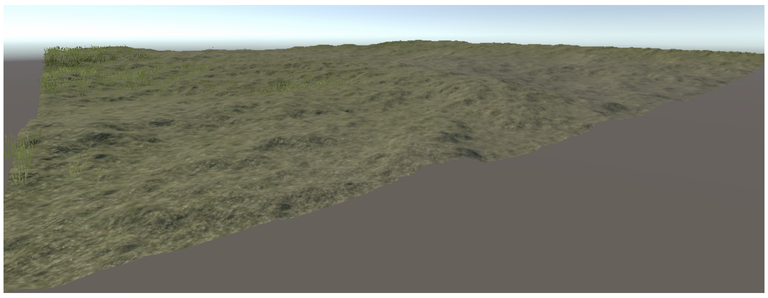
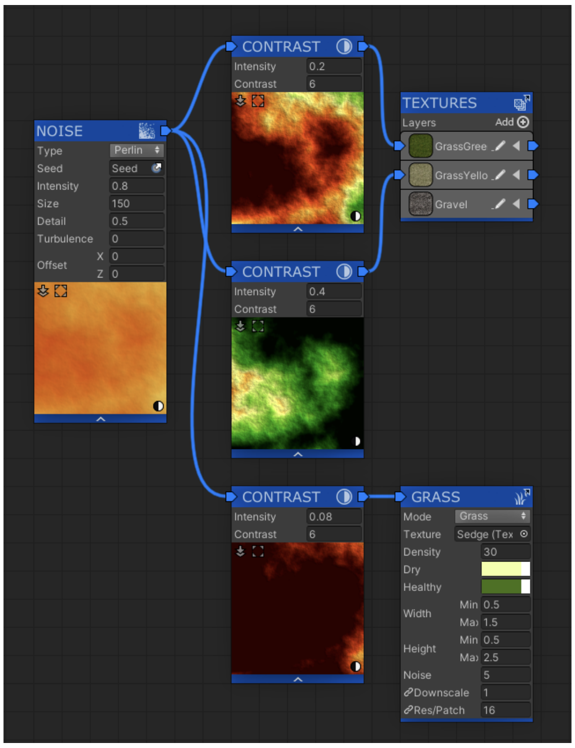


Figure 5.2: Procedurally generated terrain

The texture applied to the terrain geometry consists of three different textures that were blended together; gravel, yellow grass, and green grass. First, a separate noise map was generated using Perlin noise. The output of the Perlin noise was fed into three separate intensity/contrast modifiers, two of which are for the yellow and green grass textures. The only difference between the contrast modifiers is the intensity, otherwise, they use the exact same noise map. The textures are applied in layers and are visible depending on if their input value (the noise map) is above a certain threshold. Gravel does not have an input value as it is the base layer. Since green grass has a lower intensity input than yellow grass, green grass will only be applied if the noise value is sufficiently high, and yellow grass is applied at a slightly lower value since its intensity modifier is higher. This will effectively blend from gravel to yellow grass and finally to green grass texture with an increasing value in the noise map. The graph controlling the texture of the terrain can be seen in Figure 5.3.


NOISE
Type: Perlin
Seed: Seed
Intensity: 0.8
Size: 150
Detail: 0.5
Turbulence: 0
Offset X: 0
Offset Z: 0

CONTRAST
Intensity: 0.2
Contrast: 6

CONTRAST
Intensity: 0.4
Contrast: 6

CONTRAST
Intensity: 0.08
Contrast: 6

TEXTURES
Layers Add
GrassGree
GrassYello
Gravel

GRASS
Mode: Grass
Texture: Sedge (Tex
Density: 30
Dry
Healthy
Width Min: 0.5
Width Max: 1.5
Height Min: 0.5
Height Max: 2.5
Noise: 5
Downscale: 1
Res/Patch: 16

Figure 5.3: MapMagic graph of terrain texture and grass

In addition to the green grass texture, tall grass was also generated using the same Perlin noise map. This has an even higher threshold than the green grass texture so that tall grass was only generated over the green grass texture. The height of the tall grass varies between 0.5 and 2.5 meters, this is to cause some sheep to be partially occluded by the grass so that the detection of occluded sheep could be improved. Wind is also simulated for the grass, so the grass would be bent over in a certain direction instead of simply standing straight up. This increases the grass's ability to occlude sheep since they are seen from above. An example of tall grass can be seen in Figure 5.4.

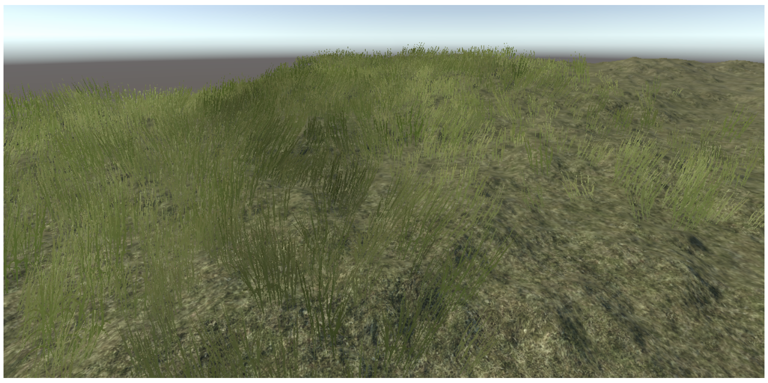


Figure 5.4: Terrain grass

5.1.2 Light

The scene had a single directional light representing the sun. The angle of the directional light was randomized using the built-in *Sun angle randomizer*. This randomizer works by taking the hour of the day, the day of the year, and latitude as input parameters. These parameters are specified as distributions, and in this case, uniform distributions were chosen. The hour of the day is sampled uniformly between 09:00 and 18:00. These times were chosen as it covers the majority of the time when the real images were captured. The day of the year is sampled uniformly from 0 to 365. Latitude is sampled between 50°and 70°.

In addition to randomizing the angle, the light intensity, hue and saturation were also randomized to provide additional variation. The intensity varied from 0.5 to 2.0. For reference, the default intensity is 1.0. Hue and saturation were randomized within a much narrower range of values, as a blue or green sun is unlikely to occur in the real world.

The combination of randomizing the sun angle and terrain geometry provides a good amount of variance in the background, as the terrain casts shadows onto itself (see Figure 5.5), in addition to the wide variety of shadows cast by the other objects placed in the scene. This makes the images overall appear much more photo-realistic than e.g. using a flat background image.

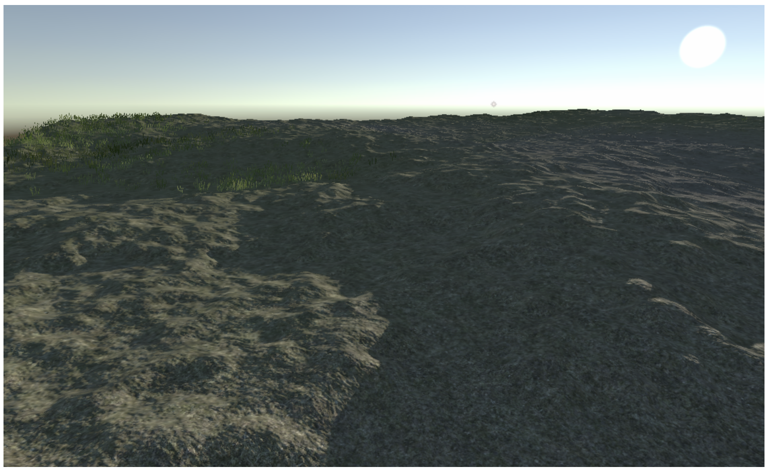


Figure 5.5: Shallow sun angle casting shadows on the terrain

5.1.3 Cameras

It is desirable to match the camera specifications from the sensor dataset, as mentioned by (Nowruzi et al. 2019). Since the real images are tiled, the FOV of the cameras in Unity were reduced correspondingly. That way, tiling the synthetic images was not necessary. The camera on the DJI Mavic 2 Enterprise has a horizontal FOV of 85°(DJI 2022). Since the images are divided by 8 horizontally, the cameras in Unity were given a horizontal FOV of 10.625°.

For the sensor data, a single image is tiled into many images, therefore having multiple cameras per iteration when generating the synthetic data is analogous. It also takes a lot of computational power to generate each iteration, so using multiple cameras in different locations speeds up the process of generating data significantly. Therefore, an array of 28 cameras per iteration that are placed in a 7x4 grid were used.

The cameras were randomized by adjusting the height and angle in the X-direction. The height was sampled uniformly between 30 and 60 meters since the majority of the sensor data was in that range. The angle was sampled uniformly between 75°and 90°, where 90°corresponds to looking straight down, and 0°corresponds to looking straight ahead. Because of the rotation randomization, the cameras are placed further apart along the X-axis to avoid multiple images of the same area. In the sensor dataset, the tiles close to the edges of the image have a significant parallax effect which results in the objects in these tiles not being seen from directly above. The random rotation of the cameras in Unity provides a similar effect; however, it may not be of the same magnitude. The camera configuration can be seen in Figure 5.6.

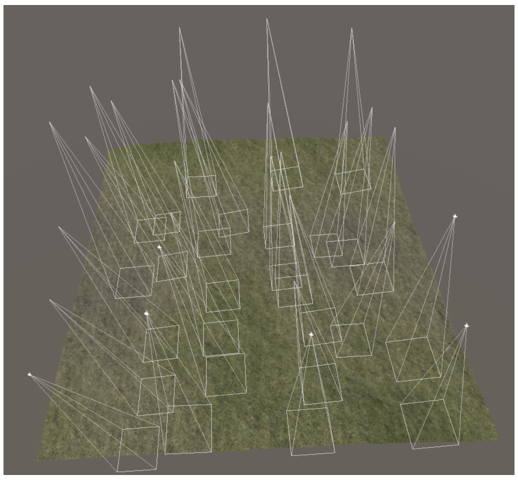


Figure 5.6: Camera array with randomized height and angle

5.1.4 Rocks

In the sensor dataset, there are often rocks on the terrain, these can sometimes be of a similar size to sheep and could therefore be mistaken for sheep. It is important to include this kind of variety in the synthetic dataset as well, especially since it is a very low-hanging fruit that is easy to implement. In each iteration, a random number of rocks were sampled uniformly between 50 and 200. There were six different rock models and one was selected at random for each rock to be placed. Additionally, the scale of each rock was sampled randomly between a very small size, and a size larger than a sheep. The rotation of each rock was also selected randomly. Finally, the rock was placed at a random point on the terrain surface. An example of rocks randomly scaled, rotated, and placed on the terrain surface can be seen in Figure 5.7.

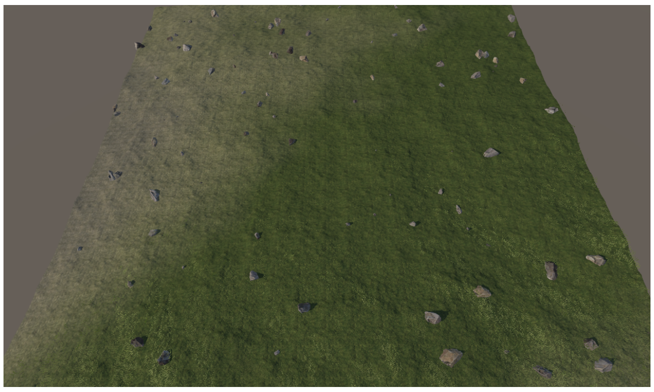


Figure 5.7: Example of rock placement and ground texture transition

5.1.5 Trees

There are also trees in the sensor dataset, and often, sheep could be located under or near the edges of trees. This is often the source of partially occluded sheep and so placing enough trees is critically important for improving the detection of occluded sheep. In the sensor dataset, there are generally a lot of fir and birch trees. The trees are positioned slightly differently from rocks, as in the real world some areas are completely free of trees (like grassy fields) and other areas are densely forested. For this reason, trees were positioned in clusters of the same species. Two different tree species were used; pine and oak (high-quality free birch models were difficult to find). Each species had several different 3D models. For each iteration, a random number of tree clusters were sampled uniformly between 3 and 7. For each cluster, a species was selected at random. Next, a random number of trees per cluster was sampled uniformly between 15 and 30. The area of the cluster was determined by the number of trees together with a density that was also sampled uniformly. For each cluster, the trees were placed randomly within a circle around a randomly selected point on the terrain surface. Each tree placed got a randomly selected model from its species. The height and width, as well as the rotation along the Y-axis, were randomized in order to increase the variance. An example of tree-randomization and clustering can be seen in Figure 5.8.

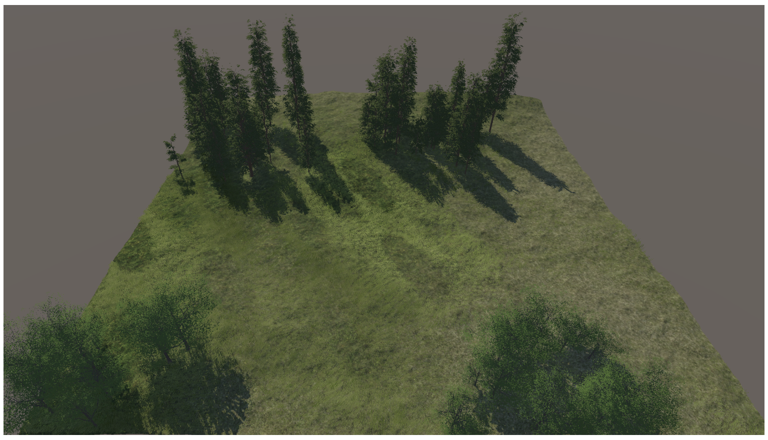


Figure 5.8: Examples of tree clusters

5.1.6 Sheep

Randomizing the sheep is the most important aspect of generating the synthetic data. It is very important to get enough variance so that the model can generalize to real images of sheep. Unfortunately, there was only one high-quality 3D model of sheep available for use in this project; however, the randomizations performed should ensure an adequate level of variance regardless. For each iteration, 300 sheep were placed randomly on the terrain surface with a random Y-rotation. Each sheep got a random scale ranging from the size of a small lamb to a big fully grown sheep. In addition to this, the width was multiplied randomly to try and emulate the fact that many sheep in the sensor dataset have very thick wool. Additionally, the sheep model was animated, so a random animation pose was selected for each sheep. The animations range from running, lying down, standing, eating, and much more. This is critical for ensuring enough variance as the sheep are rarely standing completely straight in the sensor dataset. Next, since the sheep were classified into four different colors (white, gray, brown, and black) the wool texture also needed randomization. The texture that came with the 3D model was a relatively plain white color. However, in the sensor dataset, the wool had more variety in both colors and spots of lighter or darker wool. To emulate this, a set of textures for each color was generated. First, a general base color was selected for each of the four colors. Next, the wool texture is shifted in hue, saturation, and value so that the average color of the wool texture matches this base color. Finally, a fractal noise map for saturation and value was generated and applied to the texture in order to increase the variance of the wool, and provide spots of lighter/darker and more/less saturation, while still keeping the general color of the wool.

The goal was to improve the detection of the most difficult classes, which were assumed to be the classes with the lowest number of instances in the real dataset. The probability of selecting a specific base color was therefore informed by Table 4.1 from the previous chapter. Colors that were underrepresented in the real dataset were given a higher probability of being selected. The probability of selecting a specific base color was the following: brown (40%), black (30%), gray (20%), and white (10%). After the base color was selected, one of the randomly generated textures of that color was applied to the sheep. An example of the sheep-randomization can be seen in Figure 5.9.

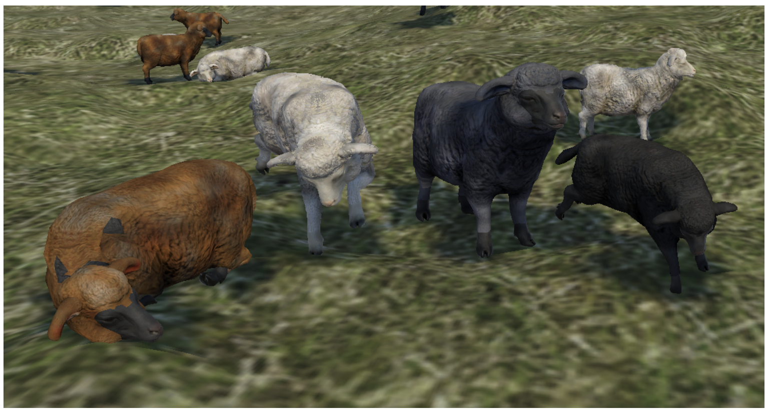


Figure 5.9: Sheep with randomized features

5.1.7 Other animals

In the sensor dataset, there are sometimes other animals present, particularly cows. The model may mistake other animals for sheep, so it is important to include some other animals in the synthetic dataset so that the model will learn to distinguish sheep from other animals. Another use case for this could be to include other animals explicitly, this is especially relevant for detecting predators like wolves, which are notorious for killing sheep. Since wolves are quite rare in Norway, it would be difficult to get enough real data for detection, so using synthetic data could be very useful. This is however outside the scope of this thesis and so predators were not included. The other animals that were included are 3 chicken models, 2 cow models, 1 goat model, and 1 pig model. For each iteration, a random number of animals were sampled uniformly between 20 and 100. For each animal, one of the above models was selected at random. Each model got a random Y-rotation and a random scale. In addition to this, all the models were animated, so a random animation pose was also selected at random before being positioned at a random point on the terrain surface. Randomizing the rotation, scale, and animation should provide a sufficiently high degree of variance for the animals, even if there were a limited number of models available. Some examples of the other animal models with a random animation pose applied can be seen in Figure 5.10.

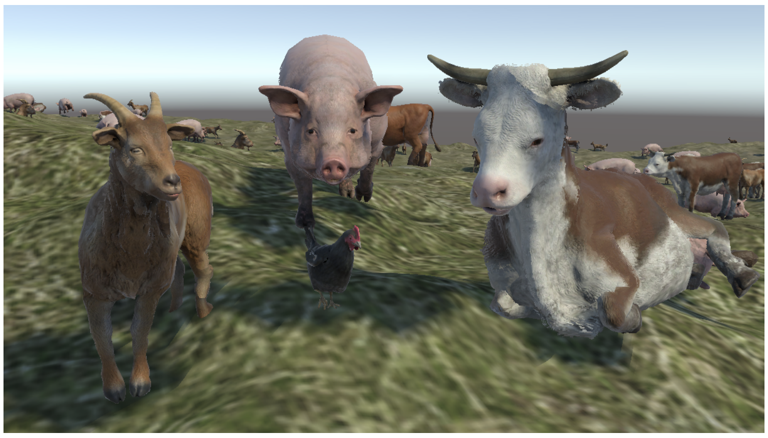


Figure 5.10: Other animals with randomized pose

5.2 Limitations

One limitation of Unity Perception is that it currently does not handle transparent objects very well. Bounding boxes are drawn correctly when occluded by an opaque object, however, if the occluding object is semi-transparent it does not properly deal with it. For example, grass and leaves are typically rendered using a simple mesh and a semi-transparent material to reduce the number of polygons used and thereby speeding up the rendering significantly. This is also the case for the vegetation models used in this project. In these cases, the shader used can be modified so that Unity Perception either sees the mesh as entirely opaque or entirely transparent; it cannot properly label sheep that are easily seen through a transparent object while at the same time excluding those that are completely occluded by transparent objects. This is something that the Perception team is planning to address, but at the time of writing, this limitation exists. Since one of the most important aspects of generating the synthetic data was increasing the presence of sheep partially occluded by leaves and grass, this is rather unfortunate. For this reason, all grass and leaves were seen as transparent when generating labels for the synthetic data, as no better option was possible with the current state of the Perception package. This means that the bounding boxes of sheep behind leaves were drawn as if the leaves were not there. For most cases, especially with respect to grass cover, this is fine. However, if a sheep was located behind a tree in such a way that the leaves completely occluded it, it would still draw the bounding box around the sheep even though it was not visible. The tree trunks and branches are still considered as opaque and were handled properly. To address this limitation, these labels had to be manually removed from the synthetic dataset.

Another problem with the current state of Unity Perception is the stability of the package. While creating the scene in Unity, several bugs and problems occurred where the cause was unknown and that were not straightforward to fix. Because of these stability issues, the Unity project had to be recreated several times over the course of this project. This is not completely unexpected as the Perception package is still in an experimental stage, and bugs and stability issues are to be expected.

For the above reasons, it is the author's view that Unity Perception still has some work ahead of it before it can be widely used in practical applications.

5.3 Resulting data

With everything put together, the final scenes can look something like in Figure 5.11. This represents only one example iteration, and everything about the scene changes between iterations as specified in section 5.1.

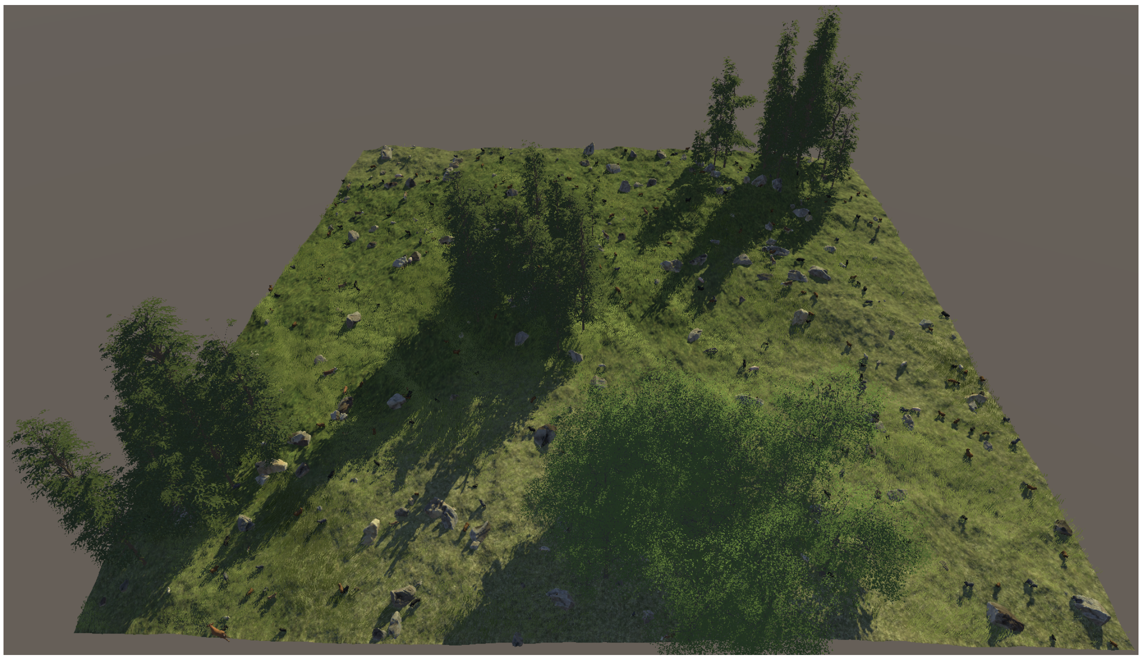


Figure 5.11: Result of a complete scene generation

For the generation of the synthetic dataset, the scenario was run for 200 iterations; creating 200 random scenes. For each scene, the 28 cameras generated one image each resulting in 5,600 images. The images were rendered at a resolution of 640x640 pixels. Some examples of the generated images along with their labels can be seen in Figure 5.12. Generating these images only took around 14 minutes on a Macbook Air (M1), running the Intel version of Unity using Rosetta 2 (Apple's software for converting x86 instructions to be compatible with the ARM instruction set used by Apple's chips). Any later version of Unity with native support for Apple silicon is not officially supported by the Unity Perception package yet. This is very fast, and one could easily generate very large synthetic datasets this way. One limiting factor, in this case, was that certain labels needed to be removed manually because of the transparency limitation mentioned in section 5.2.

After generating the images, images containing no sheep were removed from the dataset since that was also done for the sensor dataset. That left 4,570 images containing 10,410 sheep. Next, the labels where sheep are completely occluded by leaves or grass were removed manually. Labels were removed if there was only vegetation visible within the bounding box or if it was very difficult to tell that a sheep was present. This turned out to be quite labor-intensive, so, unfortunately, there was not enough time to create as much synthetic data as initially planned. However, it adds the most data to the underrepresented classes with the fewest instances, so it should hopefully still be helpful. After removing those labels there were 3,867 images left with 8,079 total instances of sheep. Finally, the synthetic dataset could be added to the sensor training set to create the mixed training set. The number of instances for each color for the sensor, synthetic, and mixed training sets can be seen in Table 5.1.

In the synthetic dataset, the sheep were only labeled by their color and not whether they were partially occluded. This is because automatically determining whether sheep are occluded is not a trivial task, since the question of how much occlusion is enough to define it as such is not straightforward, and not necessarily comparable to the result of doing it manually. In addition to this, integrating such an automatic checking and label modification into Unity Perception would not be straightforward; it would likely be necessary to extend the *Bounding Box Labeler*. For these reasons, and because manually checking each instance for occlusion in the synthetic dataset would be very labor-intensive, the sheep in the synthetic dataset were not classified by occlusion. Instead, the scenario was made to generate plenty of grass and trees, providing ample opportunity for partial occlusion by vegetation. For the sensor dataset, it was necessary to manually label occlusion to determine whether the performance of the test set on occluded sheep had improved or not, in addition to making sure that the split between the training, validation, and test sets had a representative sample of each class. This information was not strictly necessary for the synthetic dataset, even though it would be nice to know precisely how many occluded examples were added.

	White	**Gray**	**Black**	**Brown**	**Total**
Sensor	9,212	2,514	1,664	750	14,140
Synthetic	851	1,644	2,401	3,183	8,079
Mixed	10,063	4,158	4,065	3,933	22,219

Table 5.1: No. of instances of each color for the sensor, synthetic, and mixed training sets

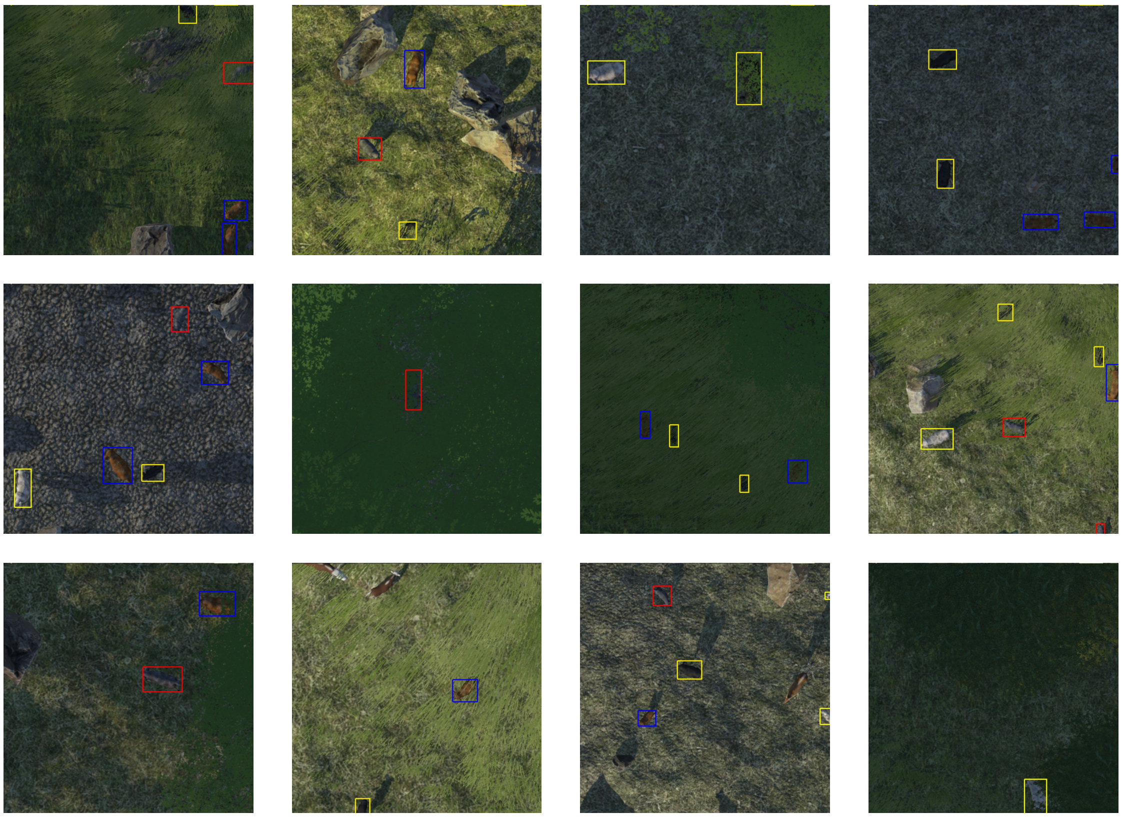


Figure 5.12: Synthetically generated training images

Chapter 6

Experiment structure

6.1 Research questions

This thesis will attempt to answer the following research questions:

- **RQ1**: Which classes of sheep are difficult to detect?
- **RQ2**: Can the detection of the difficult classes be improved by adding synthetic training data?

To clarify, the different classes of sheep that were explored are as previously described; white sheep, gray sheep, brown sheep, black sheep, white occluded sheep, gray occluded sheep, brown occluded sheep, and black occluded sheep. Additionally, some of the classes were grouped together for analysis. The groups of classes that were examined are the following:

- Non-occluded sheep (white sheep, gray sheep, brown sheep, and black sheep)
- Occluded sheep (white occluded sheep, gray occluded sheep, brown occluded sheep, and black occluded sheep)
- Dark non-occluded sheep (gray sheep, brown sheep, and black sheep)
- Dark occluded sheep (gray occluded sheep, brown occluded sheep, and black occluded sheep)

To determine whether sheep were difficult to detect or not, the recall of each class was used. To see whether there is an improvement in using synthetic data, two models were trained; the *baseline model* using only sensor data, and the *mixed model* using a combination of sensor data and synthetic data. Results of these models on the test set were then compared to determine whether there was an improvement in detection or not. Even though precision is not considered to be the most relevant metric, in this case, the precision before and after adding synthetic data was also reported to see if the model suffers from any unintended side effects.

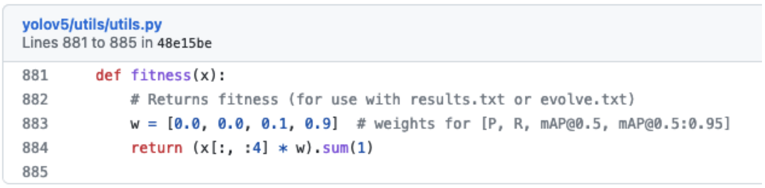
The hypothesis related to RQ1 is that classes that are underrepresented in the dataset will be more difficult to detect. It is expected that the less data, the lower the recall will be. The hypothesis for RQ2 is that the detection of these difficult-to-detect classes can be measurably improved by adding additional synthetic examples of these classes, without any measurable negative side effects.

6.2 Training

The object detection model YOLOv5 was used for these experiments. There are several variants of YOLOv5 of different sizes in terms of no. of parameters. In these experiments, YOLOv5m was used. YOLOv5m represents a trade-off between speed and complexity as it is in the middle of the pack compared to the rest of the YOLOv5 family of models. The model has been pretrained on the COCO dataset at a resolution of 640x640 (Jocher et al. 2022), which is very similar to the resolution of the images used in this experiment after tiling is performed. Both models were trained for 1,000 epochs; some preliminary testing showed that the model's performance on the validation set stabilized well before this. The hardware used to train the models is part of the IDUN cluster at NTNU (Själander et al. 2019). The machine used had two Intel Xeon Gold 6148 CPUs, where both jobs were allocated 32 CPUs and 64GB of RAM. Both jobs were also allocated four NVIDIA Tesla V100 GPUs with 32GB memory each. Since YOLOv5 is implemented using PyTorch, training on multiple GPUs was trivial using the *torch.distributed* module. A batch size of 300 was used, as this was the maximum batch size that was possible to achieve for the hardware used. Aside from this, all other hyperparameters related to training used the default values specified by YOLOv5.

The baseline model used the training, validation, and test split specified in section 4.4. The mixed model was trained on the exact same split, except that the synthetic dataset was added to the training set. The validation and test sets were not altered when training the mixed model. This means that mixed training was used as opposed to first training on purely synthetic data before fine-tuning on the real data. For fine-tuning to be practical, it would be necessary to have a much larger synthetic dataset in comparison to the real dataset, which because of the limitations of Unity Perception discussed in section 5.2 was not practical to achieve. Instead, an attempt was made to rectify the class imbalance in the real dataset by adding a relatively small amount of synthetic data where the classes which are underrepresented in the real dataset are more numerous.

After the 1,000 epochs were completed, the best model achieved throughout the training was used instead of the latest model. This was to avoid *overfitting* the model. YOLOv5 determines the best model by using a fitness function, which can be seen in Figure 6.1. This function computes a score from the weighted average of precision, recall, mAP@.5, and mAP@.5:.95 achieved for the validation set during training. The standard weights used for this calculation are 0, 0, 0.1, and 0.9 respectively.


yolov5/utils/utils.py
Lines 881 to 885 in 48e15be

```python
def fitness(x):
    # Returns fitness (for use with results.txt or evolve.txt)
    w = [0.0, 0.0, 0.1, 0.9]  # weights for [P, R, mAP@0.5, mAP@0.5:0.95]
    return (x[:, :4] * w).sum(1)

```

Figure 6.1: The fitness function used to determine the best model

Although the sheep were classified into different categories depending on color and whether they were partially occluded, this information was not used during training. The model only used a single super-class representing all sheep. The different classes were only used for evaluating the performance between the different classes afterward. This means that an individual precision for each sub-class cannot be computed, however, an individual recall is still possible to calculate. The reason for training the model on a single super-class is that it is not necessary to detect what color the sheep is, only whether a sheep of any color is present or not. Additionally, results by (Muribø 2019) suggest that modeling all sheep into a single super-class gives better performance overall than using separate classes for each color.

6.3 Testing

Even though the drone images were divided into tiles, and the synthetic images generated are of a similar resolution; when testing, no tiling was done to the test set, and the full images were used instead. As explained in section 4.5, tiling the images was only necessary for training, as the backpropagation step in the training process requires too much memory to use the full images without any downscaling or tiling. This is not necessary when running inference, as there is no backpropagation step involved then. Additionally, (Furseth and Granås 2021) found that running inference on the full images, after having trained the model on tiled images actually yielded a higher average precision than running inference on tiled images.

To test the models, both models ran inference on the test set, and the output predictions were stored. Next, NMS was applied to the raw predictions with an IoU threshold of 0.6, before being checked against the true labels of the test set to determine the number of true positives, false positives, and false negatives at 1,000 evenly spaced confidence thresholds between 0 and 1, as well as at the 10 different IoU thresholds necessary to compute the mAP@.5:.95. Using this information, the relevant metrics were computed in accordance with section 3.3. Metrics like precision, recall, and F-score were computed at an IoU threshold of 0.5, and for all confidence thresholds.

Chapter 7

Results and discussion

The links to the code repository, training logs, and datasets are available in Appendix A.

7.1 General performance comparison

Figure 7.1 shows the mAP@0.5:0.95 achieved for each epoch on the validation set during training. It shows a running average, as the actual data is quite noisy. It is clear that training for 1,000 epochs is more than sufficient to reach maximum performance from the models. Even though it may appear as if performance increases towards the end, it is actually just in the very last epoch where it jumped higher for an unknown reason. Figure 7.2 shows the box loss and object loss on the validation set during training. The validation loss starts to rise at around 200 epochs suggesting that the model is overfitting beyond this point. The class loss is not relevant since the model is trained on a single class and will therefore always have a class loss of zero.

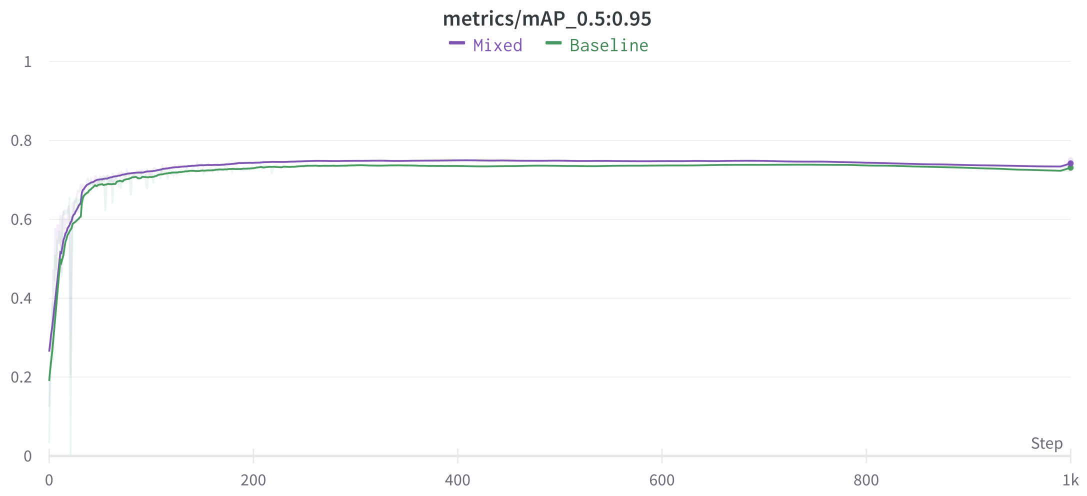
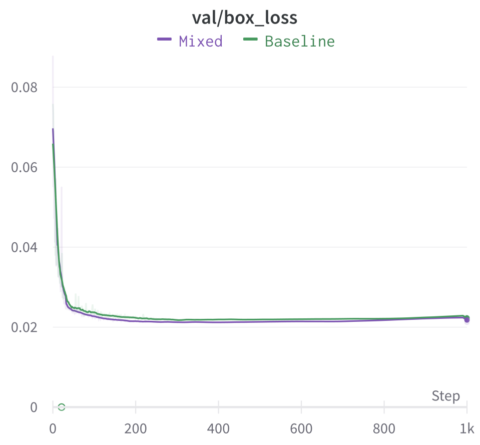
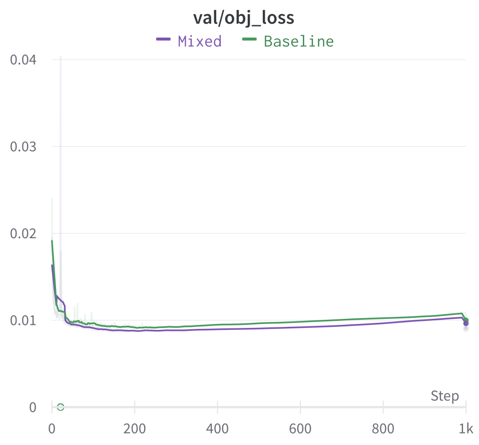


Figure 7.1: Baseline vs. mixed: mAP@0.5:0.95 on the validation set during training

Training only took approximately 9 hours for the baseline model and 11 hours for the mixed model. Even though powerful hardware was used to train the models, this is quite good. Since the model reaches maximum performance well before the 1,000 epochs are completed, similar results would likely be achievable in a fraction of the number of epochs used in this case. One



(a) Box loss



(b) Object loss

Figure 7.2: Baseline vs. mixed: Box and object loss on validation set during training

would therefore be able to quite rapidly improve upon these results by experimenting with YOLOv5's numerous hyperparameters, e.g. by using their hyperparameter evolution method.

It can see from Figure 7.3, Figure 7.4, and Figure 7.5 that the two models perform very similarly to each other in general, and it looks like the mixed model performs marginally better than the baseline. From the F-score curve in Figure 7.5, it is apparent that the maximum F-score is obtained at a confidence threshold of 0.78 for the baseline model, and 0.80 for the mixed model. This represents the *best* confidence threshold to use assuming that precision and recall are equally important. Since a confidence threshold of around 0.8 is a likely value to use in practice for both models, this is used as a point of comparison in addition to the entire curves for context. In Table 7.1, the standard metrics for object detection are listed for both models. Note that precision and recall values are evaluated at a confidence threshold of 0.8.

Based on the results in Table 7.1 and in the previously mentioned figures it is apparent that the models, on the whole, perform very similarly and that there are not any major disadvantages to adding the synthetic data. If anything, the results as a whole appear to suggest a marginal improvement after adding synthetic data to the training set. The improvement is however small and could be partially attributable to random differences during training.

	Precision	**Recall**	**mAP@.5**	**mAP@.5.95**
Baseline	0.9469	0.9457	0.9619	0.7472
Mixed	0.9512	0.9518	0.9620	0.7424

Table 7.1: Baseline vs. Mixed: main performance metrics on the test set

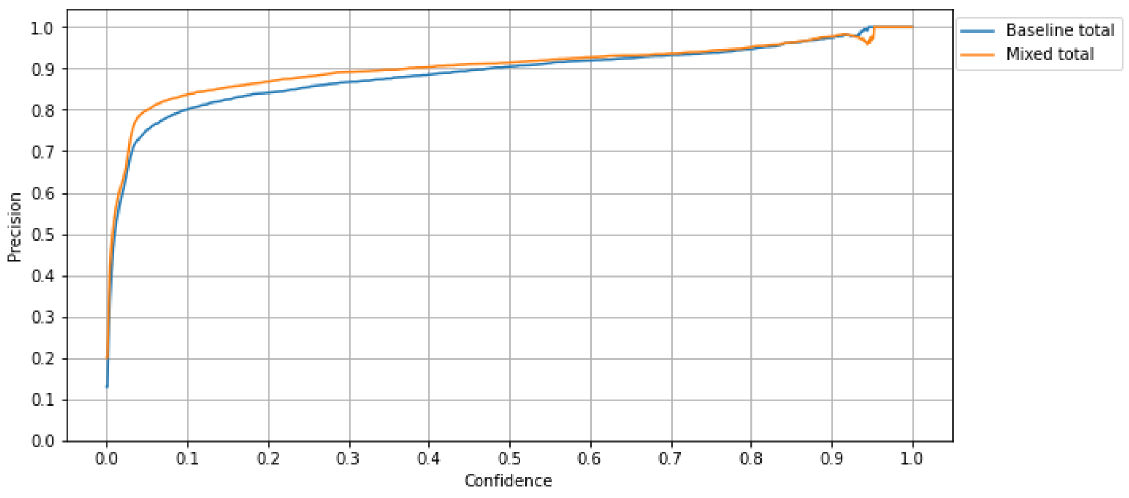


Figure 7.3: Baseline vs. mixed: total precision on the test set

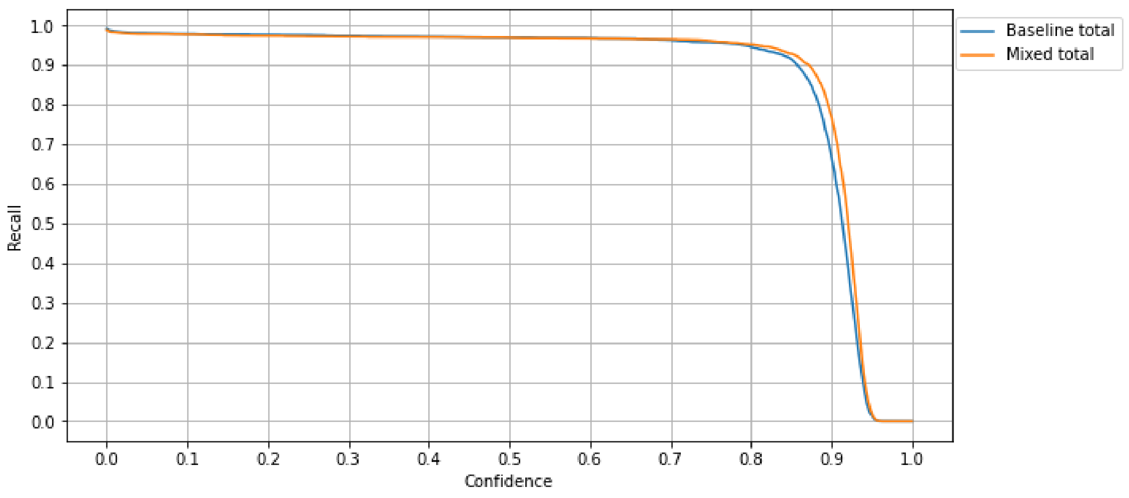


Figure 7.4: Baseline vs. mixed: total recall on the test set

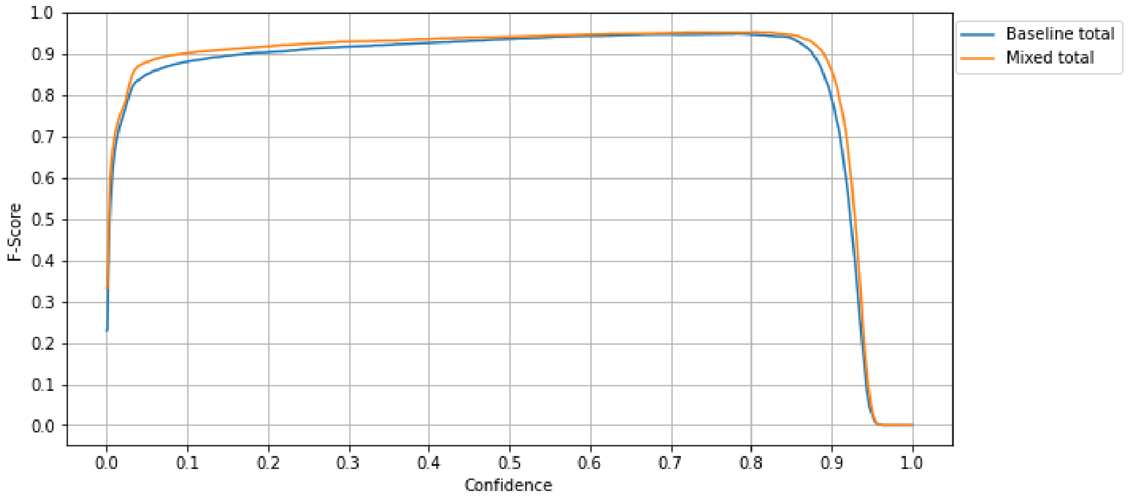


Figure 7.5: Baseline vs. mixed: F_1-score on the test set

7.2 Research question 1

Which classes of sheep are difficult to detect? To answer this question, the recall of each class obtained by the baseline model trained only on the sensor dataset must be examined. Figure 7.6 shows the recall individually for sheep of each color and whether or not they are occluded.

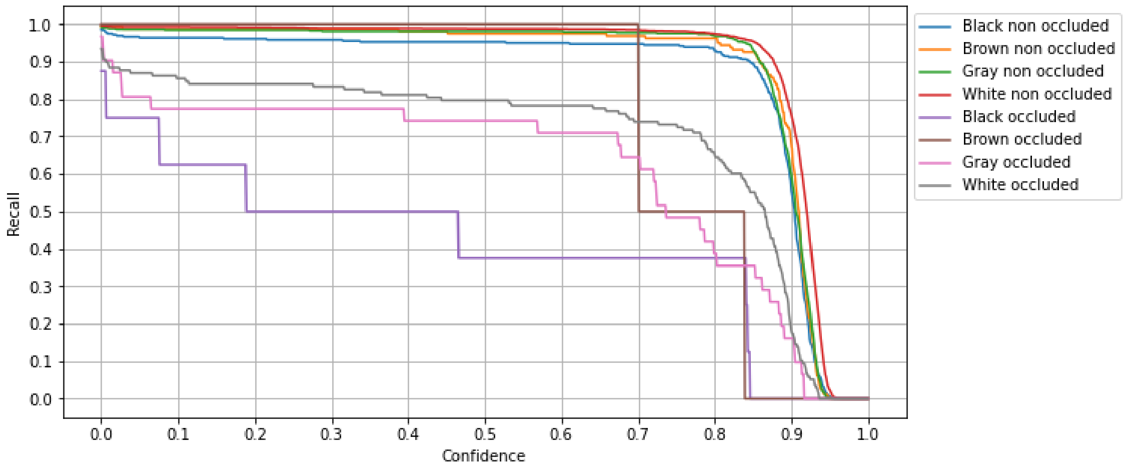


Figure 7.6: Baseline: recall for all classes on the test set

It is very clear from Figure 7.6 that sheep of every color are difficult to detect as long as they are occluded, and that non-occluded sheep in general have a similar performance between the different colors that is significantly better than their occluded counterparts. It is also clear that of the four colors, white is the least difficult. Unfortunately, there is very little data available in the test set for black and brown occluded sheep, which makes it difficult to make accurate reports about the degree of improvement. For this reason, a clearer picture may be seen by looking at the *dark* groups, which aggregate the gray, black, and brown classes. This can be seen in Figure 7.7.

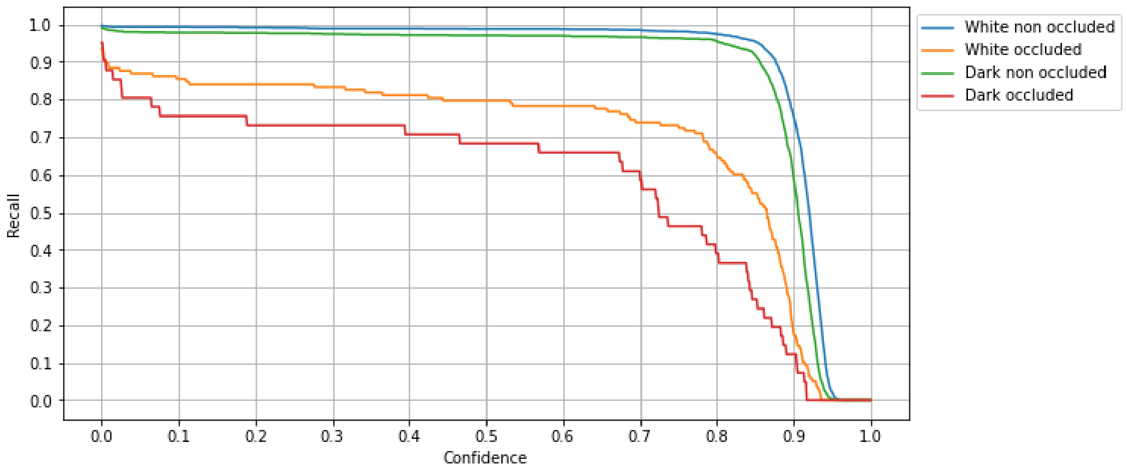


Figure 7.7: Baseline: recall for white/dark and occluded/non-occluded on the test set

As mentioned, there is also a large difference in performance between occluded and non-occluded sheep in general which is also worth examining. The comparison of occluded vs. non-occluded groups can be seen in Figure 7.8

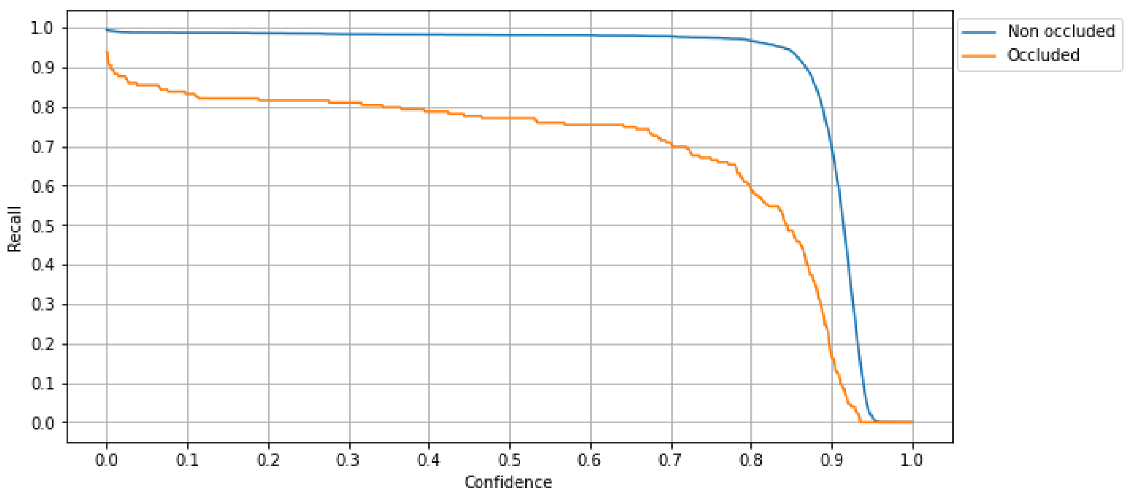


Figure 7.8: Baseline: recall for occluded vs. non-occluded sheep on the test set

In the introduction and the previous chapter, it was hypothesized that the most underrepresented classes of sheep would also be the most difficult to detect, and this hypothesis was applied when generating the synthetic data. From these results, it appears that in reality, it was not quite that simple. As expected, white sheep performed the best by far compared to any other color, and non-occluded sheep performed better than occluded sheep. However, even though brown sheep were by far the rarest, black sheep performed worse both when occluded and not occluded. Brown actually performed quite similarly to gray in both cases. However, it is difficult to say anything definitive about brown occluded sheep, as there were only two such instances in the test set. When looking exclusively at non-occluded sheep, which has a more comfortable number of instances in the test set, the brown class clearly outperforms black, suggesting that there is something inherently difficult about detecting black sheep and that its underperformance cannot be entirely attributed to its under-representation in the dataset. Additionally, even though it falls in line with the hypothesis, there is likely also something inherently difficult about detecting sheep that are partially occluded, and that its underperformance is also not completely attributable to its underrepresentation. This is because many of the instances had quite a lot of vegetation covering them, and it would be difficult even for a human to quickly tell whether there was a sheep present or not. Additionally, the number of occluded white sheep is only marginally smaller than the number of non-occluded brown sheep, yet the recall achieved for non-occluded brown sheep is significantly better than for occluded white sheep.

7.3 Research question 2

Can the detection of the difficult-to-detect classes be improved by adding synthetic training data? To answer this question, the recall of the different classes obtained by the baseline and mixed model must be compared. Based on the results obtained for **RQ1**, the classes/groups that were determined to be difficult to detect were: occluded sheep, dark occluded sheep, and black occluded sheep. The recall achieved by both models on these classes/groups at a confidence threshold of 0.8 is listed in Table 7.2.

Figure 7.9 shows the difference in recall for occluded sheep between the baseline model and the mixed model. They are mostly the same for low confidence thresholds, however, at higher confidence thresholds the mixed model clearly outperforms the baseline model. In fact, at a confidence threshold of 0.8, the mixed model has a **9.43%** higher recall than the baseline model.

Figure 7.10 shows the difference in recall between the baseline model and the mixed model on the dark occluded group. The results for this group show a similar pattern, where the recall is largely similar at low confidence thresholds, but the mixed model outperforms at higher confidence thresholds. At a confidence level of 0.8, the mixed model has a **37.5%** higher recall than the baseline model.

Finally, Figure 7.11 compares the difference in performance achieved on black occluded sheep between the two models. In this case, the mixed model outperforms over most confidence thresholds with a **66.7%** improvement at a confidence threshold of 0.8. However, for this class there are only eight samples in the test set, so we cannot put too much stock in the exact differences in performance. Adding synthetic data likely made a positive contribution to the recall of black occluded sheep, but nothing definitive can be said about the degree of improvement without significantly more instances of this class in the test set.

	Occluded	**Dark occluded**	**Black occluded**
Baseline	0.5922	0.3902	0.3750*
Mixed	0.6480	0.5366	0.6250*

*Black occluded only has eight samples in the test set

Table 7.2: Baseline vs. Mixed: recall achieved for the difficult classes at a confidence threshold of 0.8 on the test set

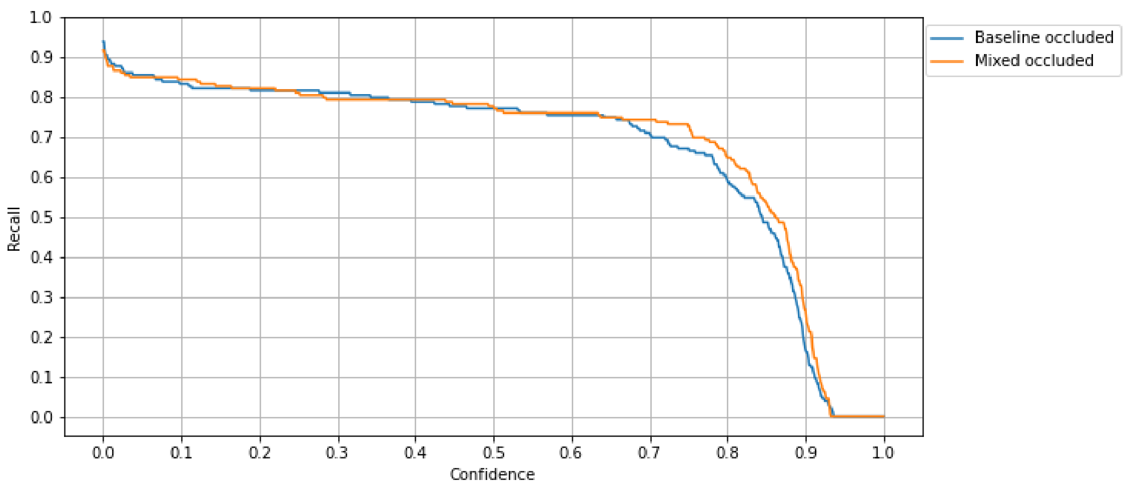


Figure 7.9: Baseline vs. mixed: recall for occluded sheep on the test set

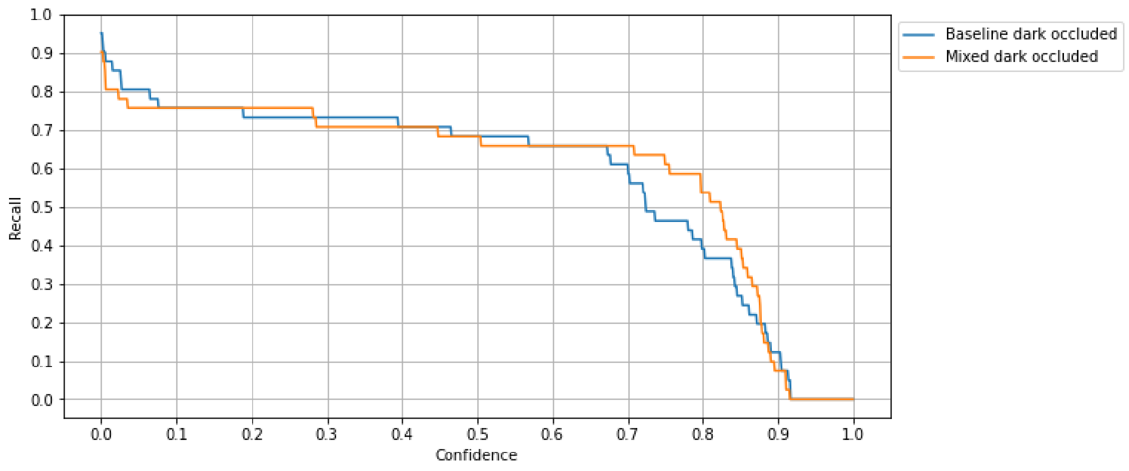


Figure 7.10: Baseline vs. mixed: recall for dark occluded sheep on the test set

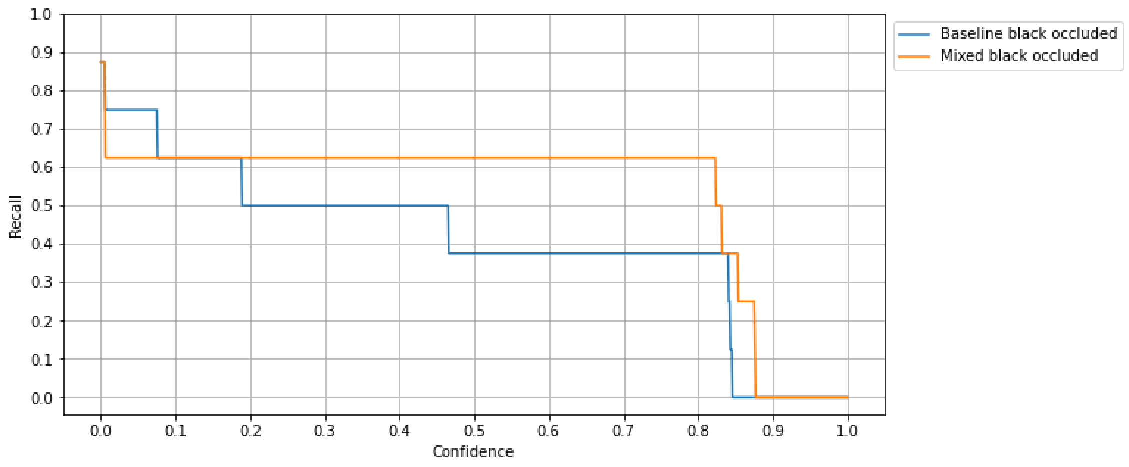


Figure 7.11: Baseline vs. mixed: recall for black occluded sheep on the test set

Chapter 8

Conclusion and future work

8.1 Conclusion

Based on the results and discussion in the previous section, the two research questions posed in chapter 6 will be answered, as well as determining whether the hypotheses related to the research questions were accurate or not.

8.1.1 Research question 1

Which classes of sheep are difficult to detect?

Based on Figure 7.6, Figure 7.7 and Figure 7.8, the classes/groups of classes that were deemed difficult to detect were in decreasing order of difficulty:

1. Black occluded sheep
2. Dark occluded sheep
3. Occluded sheep

It was initially hypothesized that the classes which are the most underrepresented in the dataset would also be the most difficult to detect, and the results mostly fall in line with this hypothesis. The only exception is that black sheep seem to be more difficult to detect than brown sheep even though black sheep were more numerous, suggesting that there is something inherently difficult about detecting black sheep. Comparing the performance between brown occluded sheep and black occluded sheep is more problematic, as there are not enough samples to say anything conclusive. Another case that is not strictly in conflict with the hypothesis, is that white occluded sheep had a significantly worse recall than brown non-occluded sheep, even though the number of instances of white occluded sheep was only marginally smaller than the number of brown non-occluded sheep, suggesting that there is also something inherently difficult about detecting occluded sheep. This was anticipated even though it was not explicitly stated in the hypothesis. It was omitted as the number of occluded sheep of every color was lower than the number of non-occluded sheep of every color, so it would not conflict with the stated hypothesis in this specific case.

8.1.2 Research question 2

Can the detection of the difficult classes be improved by adding synthetic training data?

Even though the amount of synthetic data used was very low compared to previous studies, by focusing the synthetic data on the underrepresented classes and thereby improving the class imbalance found in the real dataset, the addition of synthetic data did measurably improve the recall of the three classes/groups of classes that were determined to be difficult to detect. This can be seen in Table 7.2, Figure 7.9, Figure 7.10, and Figure 7.11. The mixed model achieved a **9.43%** higher recall over the baseline for occluded sheep in general, and a **37.5%** higher recall over the baseline for dark occluded sheep at a confidence threshold of 0.8. In conclusion, the answer to **RQ2** is that: yes, detection of difficult-to-detect classes can be measurably improved by adding synthetic training data. Additionally, there were not any measurable negative side-effects to the addition of the synthetic data judging by the other metrics in section 7.1, meaning that the initial hypothesis regarding **RQ2** was accurate.

8.2 Suggestions for future work

The use of synthetic data to improve the detection of grazing sheep appears to be quite promising, but there are many more avenues to explore concerning this specific problem. Some of these suggestions are methods applied successfully in previous research that was not examined here; other suggestions are ways of building upon and improving the synthetic data generation framework proposed in this thesis.

8.2.1 Fine-tuning as opposed to mixed training

(Nowruzi et al. 2019) found that first training on purely synthetic data before fine-tuning on the real data gave better results than training on a mixed set containing both real and synthetic data. In their experiments, they generally used much larger synthetic datasets than in this experiment. This synthetic dataset was limited in size due to the limitations of Unity Perception discussed in section 5.2; it was decided that to use the same technique a much larger synthetic dataset would be needed, or it would not really be fine-tuning at all. Given that the largest difference between mixed training and fine-tuning was the average recall, it would be very interesting to examine whether this method could be successfully applied to the problem of sheep retrieval. If Unity Perception fixes the limitation with regard to transparent objects, it would be trivial to produce very large synthetic datasets using the method proposed in this thesis.

8.2.2 Alternative camera setup

It is not apparent from the sensor dataset what angle the camera is pointing at when images are captured, it is assumed that the camera is pointed more or less straight down. However, due to the wide field of view, the image tiles close to the edges are not seen from straight above, but rather at a substantial angle. In the synthetic dataset generated here, the variation in angle is relatively small, and may not fully represent the diversity found in the real dataset when tiled. Therefore, a camera setup more akin to what is seen in Figure 8.1 could be considered, where the position of all of the cameras is centered in the scene, and only the angle is changed to cover the terrain as opposed to translating the cameras, which was done in generating the synthetic data here.

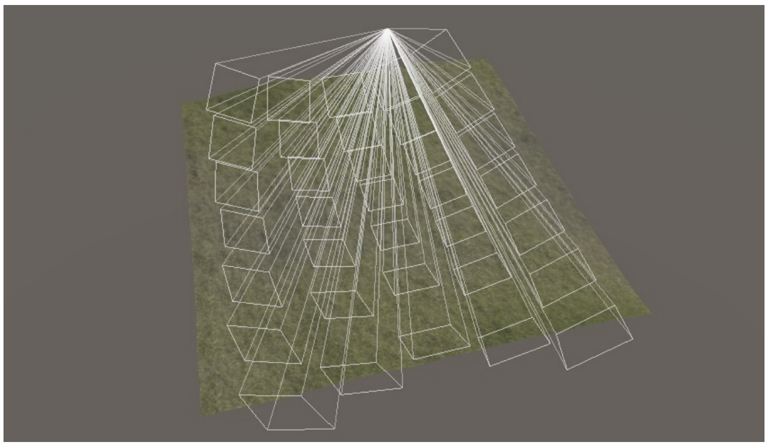


Figure 8.1: Suggestion for an alternative camera setup

8.2.3 Increased variation in the synthetic data

Even though the synthetic dataset used in this thesis had a lot of important variations, it is quite trivial to extend with even more variety. One area in particular that could benefit from more variation, is the ground textures and vegetation. The ground textures are limited to blending between gravel, yellow grass, and green grass. This could be extended to include more types of terrain where sheep or other livestock inhabit around the world. In this case, the vegetation was limited to one type of grass that grows when the underlying grass texture is green, as well as two different tree species. More vegetation in the forms of different grass species, as well as different types of bushes and flowers, could be added to further increase the variation. Additionally, a more sophisticated method of tree clustering could be considered to avoid trees overlapping with each other too much, and to avoid clusters of different species overlapping. Adding fog and simulating cloud cover is also a way to add more variation. Another possible avenue for increased variation is adding more animals that can be found in the areas where sheep graze. Additionally, increasing the variation of the sheep themselves. Even though they are scaled, and their fur is somewhat randomized using noise, more sheep models could be added, with more variety in their wool, as there is quite a bit of variation in the sheep's wool in the real dataset.

8.2.4 Hyperparameter optimization

In this experiment, the default hyperparameters for YOLOv5 were used. However, YOLOv5 has a large number of tunable hyperparameters that can be adjusted to try to get even better performance out of the available data. YOLOv5 also provides an option in its training script for performing hyperparameter evolution which uses a genetic algorithm. That way many different model configurations can easily be tested to see if there is anything to gain by tweaking hyperparameters.

8.2.5 Style transfer using GANs

As was mentioned in section 2.4, (W. Liu, J. Liu and Luo 2020) found that performance was significantly better when performing a style transfer on the synthetic data using CycleGAN to more closely resemble the real dataset. They did not however compare the performance with fine-tuning on the real data between the synthetic dataset before and after style transfer, as they only tested fine-tuning using the real data on the synthetic dataset after style transfer was performed with CycleGAN. It is possible that fine-tuning on the real data will to some degree serve the same purpose as applying the style transfer to the synthetic dataset, but this was unfortunately not tested in their experiments. They only used flat background images instead of complete 3D terrains which were used in this experiment, so it would be interesting to learn whether using GANs for style transfer on this type of synthetic dataset which is already quite photo-realistic could improve the performance over using the synthetic data as is. Comparing these methods in conjunction with fine-tuning on real data would also be very enlightening.

8.2.6 Determining the optimal flight altitude

Previous master's theses have already explored the problem of planning an optimal flight path for the drone to follow to cover an area of interest (Rognlien and Tran 2018). However, it would be interesting to explore the optimal flight altitude of the drone. When the drone is flying at a higher altitude, it will be able to image a larger area and as a result, fewer images need to be taken, speeding up the process of localizing sheep. However, a higher altitude will make the sheep appear smaller in the images, and therefore make detection more difficult (Petso et al. 2021). Determining the optimal flight altitude, therefore, is an interesting problem to examine. It requires examining the performance of object detection models on images taken at different altitudes, as well as determining the amount of time required to cover an area when flying at the same altitudes. That way, a suitable trade-off may be found.

References

Blyenburgh, Peter van (1999). “UAVs: an overview”. In: *Air & Space Europe* 1.5, pp. 43–47. ISSN: 1290-0958. DOI: https://doi.org/10.1016/S1290-0958(00)88869-3. URL: https://www.sciencedirect.com/science/article/pii/S1290095800888693.

Bochkovskiy, Alexey, Wang, Chien-Yao and Liao, Hong-Yuan Mark (2020). “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *CoRR* abs/2004.10934. arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

Borkman, Steve et al. (2021). “Unity Perception: Generate Synthetic Data for Computer Vision”. In: *CoRR* abs/2107.04259. arXiv: 2107.04259. URL: https://arxiv.org/abs/2107.04259.

Buil, Mikel Diez (Aug. 2011). “NON-MAXIMA SUPRESSION”. en. In: p. 34. URL: https://academica-e.unavarra.es/xmlui/bitstream/handle/2454/4626/577667.pdf (visited on 20/05/2022).

Al-Bukhaiti, Khalil (Apr. 2018). “STUDYING THE ACCURACY OF ELEVATION MEASUREMENTS IN THE POSITIONING SYSTEM GNSS IN COMPARISON OF CONVENTIONAL LEVELING METHODS”. In: DOI: 10.5281/zenodo.1218703. URL: https://www.researchgate.net/publication/324532576_STUDYING_THE_ACCURACY_OF_ELEVATION_MEASUREMENTS_IN_THE_POSITIONING_SYSTEM_GNSS_IN_COMPARISON_OF_CONVENTIONAL_LEVELING_METHODS.

Cybenko, G. (Dec. 1989). “Approximation by superpositions of a sigmoidal function”. en. In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274 (visited on 19/05/2022).

DJI (2022). *Mavic 2 Enterprise Series - Specifications*. en. URL: https://www.dji.com/no/mavic-2-enterprise/specs (visited on 08/04/2022).

Encyclopedia of Mathematics (June 2020). *Monotone function*. URL: https://encyclopediaofmath.org/wiki/Monotone_function (visited on 06/05/2022).

Everingham, Mark, Van Gool, Luc et al. (June 2010). “The Pascal Visual Object Classes (VOC) Challenge”. en. In: *International Journal of Computer Vision* 88.2, pp. 303–338. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-009-0275-4. URL: http://link.springer.com/10.1007/s11263-009-0275-4 (visited on 06/05/2022).

Everingham, Mark and Winn, John (May 2010). "The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Development Kit". en. In: p. 29.

Findmy (2022). URL: https://www.findmy.no/ (visited on 30/04/2022).

Furseth, Ole Kildehaug and Granås, Anders Ottersland (June 2021). "Real-time Sheep Detection". In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2834578/no.ntnu%3ainspera%3a74730513%3a33262087.pdf?sequence=1&isAllowed=y (visited on 09/05/2022).

Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Haykin, Simon S. (1999). *Neural networks: a comprehensive foundation*. 2nd ed. Upper Saddle River, N.J: Prentice Hall. ISBN: 978-0-13-273350-2.

Huang, Yanbo et al. (Sept. 2013). "Development and prospect of unmanned aerial vehicle technologies for agricultural production management". In: *International Journal of Agricultural and Biological Engineering* 6, pp. 1–10. DOI: 10.3965/j.ijabe.20130603.001.

Hvasshovd, Svein-Olaf (2017). *Droner og Sau og litt til !!* URL: https://www.statsforvalteren.no/contentassets/cbf122460efa4e37a051c17c07fade0d/droner-buskerud-2017.pdf (visited on 05/06/2022).

Jocher, Glenn et al. (Feb. 2022). *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Version v6.1. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936.

Johanssen, J. R. E. and Sørheim, K. M. (2018). "Atferd og velferd hos sau". no. In: p. 6. URL: https://orgprints.org/id/eprint/33947/1/NORS%5C%C3%5C%98K%5C%20FAGINFO%5C%20Nr.%5C%205%5C%202018%5C%20Atferd%5C%20og%5C%20velferd%5C%20hos%5C%20sau.pdf.

Kaarud, Jens Tobias, Nordvik, Magnus Falkenberg and Paulsen, Håkon Rosseland (June 2020). "Drone-based Detection of Sheep using Thermal and Visual Cameras: A Complete Approach". In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2777509/no.ntnu%5C%3ainspera%5C%3a57384149%5C%3a16365942.pdf?sequence=1&isAllowed=y (visited on 09/05/2022).

Kartverket (Mar. 2022). *Status for nasjonal detaljert høydemodell*. nb-NO. URL: https://kartverket.no/geodataarbeid/nasjonal-detaljert-hoydemodell/status-hoydemodell (visited on 04/05/2022).

Landbruksdirektoratet (2022). *Produksjonstilskudd PT-900*. URL: https://ldir.statistikkdata.no/pt-900_del2_2021_land.html (visited on 30/04/2022).

Li, Zhuang et al. (Jan. 2022). “A Two-Stage Industrial Defect Detection Framework Based on Improved-YOLOv5 and Optimized-Inception-ResnetV2 Models”. en. In: *Applied Sciences* 12.2, p. 834. ISSN: 2076-3417. DOI: 10.3390/app12020834. URL: https://www.mdpi.com/2076-3417/12/2/834 (visited on 20/05/2022).

Liu, Weixing, Liu, Jun and Luo, Bin (2020). “Can Synthetic Data Improve Object Detection Results for Remote Sensing Images?” In: *CoRR* abs/2006.05015. arXiv: 2006.05015. URL: https://arxiv.org/abs/2006.05015.

Lovdata (2014). *Forskrift om erstatning når husdyr blir drept eller skadet av rovvilt*. URL: https://lovdata.no/dokument/SF/forskrift/2014-05-30-677 (visited on 30/04/2022).

McCulloch, Warren S. and Pitts, Walter (Dec. 1943). “A logical calculus of the ideas immanent in nervous activity”. en. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259 (visited on 19/05/2022).

MIL-STD-810 (Feb. 2022). en. Page Version ID: 1069836851. URL: https://en.wikipedia.org/w/index.php?title=MIL-STD-810&oldid=1069836851 (visited on 30/04/2022).

Muribø, Jonas Hermansen (2019). “Locating Sheep with YOLOv3”. In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2619041.

Nerland, Trygve (June 2021). “Radio tracking of sheep - Developing MAVLink enabled devices, MAVLink control and the basis for MAVLink enabled autonomous UAVs”. In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2778080/no.ntnu%5C%3ainspera%5C%3a80718798%5C%3a14595913.pdf?sequence=1&isAllowed=y (visited on 03/05/2022).

NIBIO (2020a). *Hovedtabell 17f, del 3. Drift og driftsresultat på bruk med sauehold*. URL: https://driftsgranskingane.nibio.no/drgr/hovudtabellar/?vis=htab&tabell_id=52&aar=2020&lang=BM (visited on 03/05/2022).

– (2020b). *Hovedtabell 17f, del 4. Drift og driftsresultat på bruk med sauehold*. URL: https://driftsgranskingane.nibio.no/drgr/hovudtabellar/?vis=htab&tabell_id=53&aar=2020&lang=BM (visited on 03/05/2022).

Nowruzi, Farzan Erlik et al. (2019). “How much real data do we actually need: Analyzing object detection performance using synthetic and real data”. In: *CoRR* abs/1907.07061. arXiv: 1907.07061. URL: http://arxiv.org/abs/1907.07061.

NSG (2022). *Verdisatser - Norsk Sau og Geit*. no. URL: https://www.nsg.no/om-nsg/okonomi/verdisatser/ (visited on 30/04/2022).

Nyholm, Henrik (July 2020). “Localizing Sheep using a Bluetooth Low Energy enabled Unmanned Aerial Vehicle for Round-trip Time of Arrival-based Multilateriation”. In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2778147/no.

ntnu%5C%3ainspera%5C%3a53184405%5C%3a31392769.pdf?sequence=1&isAllowed=y (visited on 03/05/2022).

Petso, Tinao et al. (Oct. 2021). "Individual Animal and Herd Identification Using Custom YOLO v3 and v4 with Images Taken from a UAV Camera at Different Altitudes". In: *2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP)*, pp. 33–39. DOI: 10.1109/ICSIP52628.2021.9688827.

PROJ contributors (2022). *PROJ coordinate transformation software library*. Open Source Geospatial Foundation. DOI: 10.5281/zenodo.5884394. URL: https://proj.org/.

Redmon, Joseph, Divvala, Santosh Kumar et al. (2015). "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640. arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

Redmon, Joseph and Farhadi, Ali (2016). "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242. arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

– (2018). "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767. arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

Reynolds, Anh H. (2019). *Convolutional Neural Networks (CNNs)*. en-us. URL: https://anhreynolds.com/blogs/cnn.html (visited on 05/06/2022).

Rognlien, Even Arneberg and Tran, Tien Quoc (June 2018). "Detecting Location of Free Range Sheep - Using Unmanned Aerial Vehicles and Forward Looking Infrared Images". In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2558594/15911_FULLTEXT.pdf (visited on 24/05/2022).

Rovbase (2022). *Rovbase*. URL: https://rovbase.no/erstatning/sau (visited on 30/04/2022).

Själander, Magnus et al. (2019). *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure*. arXiv: 1912.05848 [cs.DC].

Smartbjella (2022). nb-NO. URL: https://smartbjella.no/ (visited on 30/04/2022).

Steinsvik, Gard (June 2021). "Radio-Tracking of Sheep: Developing a MAVLink-Enabled Ground Control Station with Location Estimation Based on Range-Only Measurements". In: URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2778091/no.ntnu%5C%3ainspera%5C%3a74730513%5C%3a17655346.pdf?sequence=1&isAllowed=y (visited on 03/05/2022).

Taha, Abdel Aziz and Hanbury, Allan (Aug. 2015). "Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool". In: *BMC Medical Imaging* 15.1, p. 29. ISSN: 1471-2342. DOI: 10.1186/s12880-015-0068-x. URL: https://doi.org/10.1186/s12880-015-0068-x (visited on 06/05/2022).

Telespor (2022). nb-NO. URL: http://www.telespor.no/ (visited on 30/04/2022).

Unity Technologies (2020). *Unity Perception Package*. https://github.com/Unity-Technologies/com.unity.perception.

Ünsalan, K. Izet (July 2020). "Classroom study of GNSS position accuracy using smartphones". en. In: *Scientific Bulletin of Naval Academy* XXIII.1, pp. 83–89. ISSN: 23928956, 1454864X. DOI: 10.21279/1454-864X-20-I1-011. URL: https://www.anmb.ro/buletinstiintific/buletine/2020_Issue1/02_EEA/23.pdf (visited on 05/05/2022).

Varga, Leon Amadeus and Zell, Andreas (2021). "Tackling the Background Bias in Sparse Object Detection via Cropped Windows". In: *CoRR* abs/2106.02288. arXiv: 2106.02288. URL: https://arxiv.org/abs/2106.02288.

Wang, Shuo (2021). "Research Towards Yolo-Series Algorithms: Comparison and Analysis of Object Detection Models for Real-Time UAV Applications". In: *Journal of Physics: Conference Series*. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1948/1/012021/pdf.

Xu, Renjie et al. (Feb. 2021). "A Forest Fire Detection System Based on Ensemble Learning". en. In: *Forests* 12.2, p. 217. ISSN: 1999-4907. DOI: 10.3390/f12020217. URL: https://www.mdpi.com/1999-4907/12/2/217 (visited on 07/05/2022).

Zhao, Zhong-Qiu et al. (2018). "Object Detection with Deep Learning: A Review". In: *CoRR* abs/1807.05511. arXiv: 1807.05511. URL: http://arxiv.org/abs/1807.05511.

Zhong, Yuanyi et al. (Mar. 2020). "Anchor Box Optimization for Object Detection". en. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Snowmass Village, CO, USA: IEEE, pp. 1275–1283. ISBN: 978-1-72816-553-0. DOI: 10.1109/WACV45572.2020.9093498. URL: https://ieeexplore.ieee.org/document/9093498/ (visited on 20/05/2022).

Appendix A

Code repository, training logs, and datasets

All of the code used for image tiling, dataset splitting, training, evaluation, and all of the custom randomizers used in Unity Perception are available in the GitHub repository for this project at:

- `https://github.com/bjosttveit/masters-thesis`

The training runs were logged using *Weights & Biases* and are available at:

- `https://wandb.ai/bjosttveit/Masters`

The two datasets used are available on *Kaggle* at:

- **Real, not tiled**:
  `https://www.kaggle.com/datasets/bjosttveit/yolo-sheep-colored-and-occluded`
- **Synthetic**:
  `https://www.kaggle.com/datasets/bjosttveit/yolo-sheep-synthetic`