Master's thesis

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

Ole Kildehaug Furseth and Anders Ottersland Granås

Real-time Sheep Detection

Improving Retrieval of Free-ranging Sheep Using Deep Learning-based Detection on Drone Imagery Running on Mobile Devices

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
June 2021

NTNU
Norwegian University of Science and Technology

Ole Kildehaug Furseth and Anders Ottersland Granås

Real-time Sheep Detection

Improving Retrieval of Free-ranging Sheep Using Deep Learning-based Detection on Drone Imagery Running on Mobile Devices

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
June 2021

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

NTNU

Kunnskap for en bedre verden

Abstract

In Norway, more than 2 million sheep are grazing freely during the summer. The pastures in which they graze are often large, covered in vegetation or rough terrain, and lay some distance from the sheep's home farm. All of these factors make retrieval of the herd a difficult and time-consuming task for the farmer. This thesis aims at developing and proposing a system to assist in this retrieval by automatically detecting sheep in images captured by drones.

To this end, several deep learning models based on the YOLOv5 architecture were developed and evaluated. Models were developed using differently sized pre-trained checkpoints and further trained on images of different image types and resolutions. The models are evaluated on their ability to detect sheep in real-time while running on mobile devices. Earlier work has shown that a fusion of models using images from the visual and thermal spectrums leads to improved results. An important aspect of this thesis was investigating whether the use of images captured with MSX-technology would yield similar results.

Performance of MSX-based models turned out to be lacking, mostly due to lack of quality in the images themselves. Models trained on regular high-resolution images performed well, with a top retrieval rate of 98% running on mobile hardware with an inference time of 851ms per image.

These results show that deep learning models are able to quickly and reliably detect sheep in drone images running on mobile hardware. This suggests that, with further development, such technology can be used to greatly effectivize sheep retrieval.

Sammendrag

På landsbasis slippes over 2 million sau på sommerbeite hvert år. Beiteområdene er ofte store, i ulendt terreng og et stykke unna selve gården. Disse faktorene gjør det vanskelig og tidkrevene for bonden å drive sauene hjem når høsten kommer. Denne masteroppgaven har som mål å utvikle og foreslå et system som kan bistå i dette arbeidet ved å automatisk detektere sau i dronebilder.

For å oppnå dette ble flere dyp læringsmodeller basert på YOLOv5-arkitekturen utviklet og evaluert. Modellene tok utgangspunkt i forhåndstrente modeller av forskjellig størrelse. De ble så trent videre på bilder av varierende type og oppløsning. Evaluering av modellene ble gjort basert på evnen deres til å detektere sau i sanntid mens de kjører på mobile enheter. Tidligere arbeid har vist at en sammenslåing av to separate modeller basert på bilder fra det synlige og termiske spektrumet fører til bedre resultater. Et viktig aspekt ved denne oppgaven var derfor å undersøke om bilder tatt med MSX-teknologi gir lignende resultater.

Ytelsen til de MSX-baserte modellene viste seg å være mindre gode – i hovedsak grunnet manglende kvalitet på selve bildene. Resultatene fra modellene trent på ordinære visuelle bilder i høy oppløsning var gode: Den beste modellen gjenfinner hele 98% av sauene med en inferenstid på 851ms per bilde på mobil maskinvare.

Disse resultatene viser at dype læringsmodeller er i stand til å detektere sau i dronebilder både hurtig og pålitelig. Med videre utvikling kan denne teknologien brukes til å effektivisere arbeidet med sauegjenfinning.

Preface

This is a master thesis written for the Department of Computer Science at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. The authors of this thesis are part of the study program Computer Science, with specialization in Databases and Search.

We would like to thank our supervisor, Svein-Olaf Hvasshovd for the weekly assistance and encouragement during our work on this thesis. We would also like to thank friends and family for valuable input and discussions throughout the project – especially Even, who has spent hours proofreading and providing tons of constructive feedback. A special thanks must also be given to Jesper, whose stressed nature has provided comic relief as well as frequent coffee breaks. We would also be remiss not to mention the *Kanelbolleonsdag* provided by Sit – a highlight of every working week!

Contents

List of Figures

List of Tables

1 Introduction

This chapter gives a brief introduction to the problem and the motivation for this thesis as well as the main goal and research questions. This is followed by a short explanation of the research method and the thesis structure.

1.1 Background and Motivation

The grazing season for sheep in Norway is an important aspect of sheep welfare and provides a use for the vast outlying fields which are hard to cultivate [1]. The sheep live on vegetation that is found naturally in these environments and help maintain the mountainous areas.

In 2019, more than 2 million sheep were released to outlying fields throughout Norway. Of these, approximately 100 000 sheep were lost during the summer season [2]. The losses are caused by a combination of natural predators, disease, accidents, etc. Additionally, rounding up all the sheep at the end of the grazing season is challenging for sheep farmers, who might spend hundreds of hours in the field – and still not find everyone. The retrieval process is made difficult by the sheer size of the pasture area, often in rugged terrain.

The Digital Revolution has been partially embraced by the agricultural sector and there are several tools to assist sheep farmers in their work. For instance the use of radio-bells and UAVs to supervise the herd. Ever improving UAVs can enable farmers to survey even the most challenging terrain to find sheep. With their ability to cover large areas they can be used to search for stragglers during the roundup.

While today drones are mainly for manual supervision [3], this project aims to research the possibilities for UAVs to scan large areas from above and automatically search for sheep using deep learning networks. With the recent rapid development in deep learning-based object detection and computer vision, there is reason to believe automatic detection of sheep in these UAV images could be a viable alternative to conventional roundup methods.

1.2 Goal and Research Questions

Goal *Develop a deep learning model that automatically detects sheep in UAV images in a real-time detection application.*

The main goal of this thesis is to develop deep learning models for finding sheep in visual and thermal drone images. The main focus is to develop and test real-time detection models to be used on a mobile device or directly on the drone itself. This kind of application would enable farmers to quickly and effectively search for missing sheep in the field. For the model to be suitable in a real-world scenario, the total processing time per image must be brief, and the number of false detections must not be prohibitively high.

RQ1 *How does running on mobile devices affect inference time and detection performance of a deep learning sheep detection model?*

A mobile device such as a smartphone or a drone has limited hardware capacity and normally sports reduced processing power compared, to a computer. Image detection and deep learning, in general, require a lot of processing power, as a great number of operations are required for a deep learning network to work properly. It is key to examine how the hardware constraints affect the time usage and quality of the detection operation.

Some deep learning architectures offer smaller versions that are less memory intensive, to make them better suited for a mobile environment. The smaller models generally perform slightly worse, but this might be a reasonable trade-off that should be studied.

RQ2 *How does a reduction in the degree of localization affect detection performance of a deep learning sheep detector?*

In practice, the detector does not need to detect every sheep in an image for every sheep to be found by the farmer – only detecting one in a herd of many should be sufficient; despite the potential poor scores this could give using traditional methods of evaluation. By evaluating the detector on what in practice is a binary classification task, i.e. «does this image contain at least one sheep?», we can make a more precise judgment of the detector's performance in a real scenario.

RQ3 *How does the use of combined visual and thermal images affect inference time and detection performance of a deep learning sheep detector?*

Some UAVs with both visual and thermal cameras provide images where both visual and thermal features are combined into a single image. These images provide features from both spectrums to the deep learning sheep detector without needing multiple input images and may contribute to a better basis for detections.

1.3 Research Method

The steps carried out to realize the thesis' goal are described below. Steps 3-5 are executed as more of an iterative process where new models are trained based on knowledge from previous results.

1. Collect and preprocess visual data of sheep. This includes expanding the already existing data set with visual and thermal images. Additionally, making sure the data is suitable and has the correct metadata.
2. Explore state-of-the-art deep learning models and choose the best suited for the project's data and detection requirements. Customize and adapt this model to use case if needed.
3. Train the deep learning model using images of sheep.
4. Develop an Android application with the ability to test different sheep detection models.
5. Test the model by exposing it to unseen images and evaluate it based on precision, recall, sheep retrieval, average precision and inference time.

1.4 Thesis Structure

Chapter 1: Introduction introduces the problem and motivation that form the basis of this thesis. It gives a short introduction to the research method and includes the research goal and questions.

Chapter 2: State of the Art covers the state-of-the-art technologies for both sheep roundup and object detection, as well as summarizing earlier thesis work related to the field – mainly focusing on the work of K. M. Johannessen [4] and our specialization project.

Chapter 3: Project Description covers the project requirements and hardware restrictions.

Chapter 4: Method describes the methods used to answer the research questions – showing how data was collected and pre-processed, the training of object detection models, and how they were compared.

Chapter 5: Results presents the results from our work.

Chapter 6: Discussion covers an analysis of our results and their implications.

Chapter 7: Conclusion attempts to summarize the thesis in one, relatively short section, as well as suggest possibilities for future related work.

2 State of the Art

This section covers the state of sheep grazing and retrieval, an overview of deep learning object detection models with an emphasis on the YOLOv5 architecture, followed by a summary of earlier work under supervisor Hvasshovd.

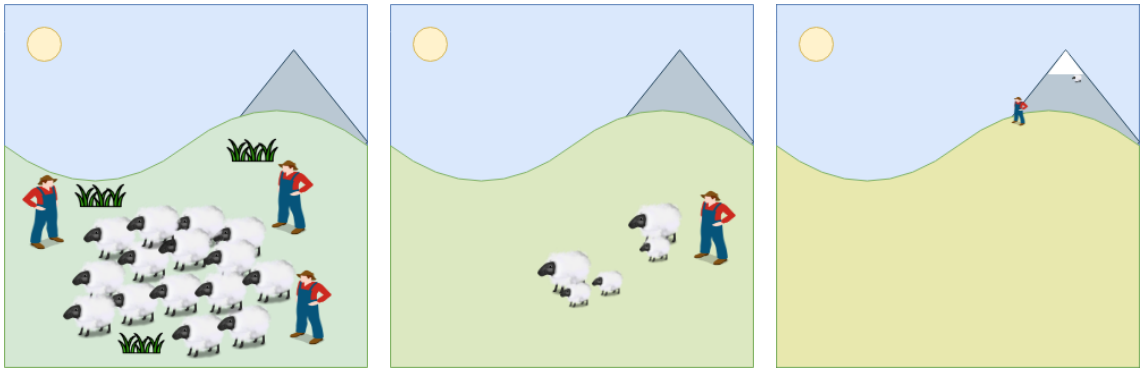
This thesis is a continuation of our specialization project [5] which in turn learned a lot from the specialization project[6] and master's thesis of K. M. Johannesen [4]. Johannesen explored the combination of RGB and IR-images in object detection as well as providing a detailed overview of several ResNet and ResNeXt architecture's performance.

2.1 Sheep Grazing and Roundup

2 million sheep and lambs were released to grazing pastures in 2019 [2]. Of these, 100 000 were never retrieved. Of the losses with a documented cause, most can be attributed to predators, accidents and illness. More than half of the losses, however, were never found and do not have a documented cause.

By law, farmers are required to supervise their grazing sheep every week. This is a task made difficult by the herd being spread over a large area in difficult terrain, often obscured by vegetation. Some measures are in place to mitigate this, such as strategically placed salt blocks and fences, in an attempt to encourage sheep to stay within a more limited area.

Bringing the sheep in from their pastures is a difficult task, usually carried out over three main phases [7] as illustrated in Figure 1.



(a) *Phase 1: Main roundup* **(b)** *Phase 2: Second roundup* **(c)** *Phase 3: Search for stragglers*

Figure 1: *The three phases of sheep roundup.*

Roundup phase 1: Carried out over several weekends, usually with the help of extra farmhands. Most sheep – about 90% – are found during this phase.

Roundup phase 2: Carried out over several weekends, usually by only the farmer or with limited manpower.

Roundup phase 3: Carrier out after the main roundup phases. The farmer now tries to retrieve the stragglers that have not been located. This is also done with limited manpower and is often extra physically demanding, since the sheep have left the main grazing area and moved into more difficult terrain.

2.2 Existing Technologies

Farmers already employ several solutions to assist in the supervision and roundup of their sheep. However, these are limiting in either their cost or efficiency.

Bells

Tried and tested, the metal bell has been in use worldwide for millennia. It is cheap, easy to use and low-maintenance. Its efficiency is limited by the farmer having to be in relative proximity to the bell in order to hear it – a problem often made worse by the rugged terrain in the Norwegian mountains. Despite this, they remain the most widely used solution to the sheep tracking problem, simply because its competitors are prohibitively expensive.

Recently there have also been questions raised regarding the welfare of the sheep having to carry around a loud bell all summer long [8] – suggesting a less noisy solution could be needed.

Radio Bells

A modern version of the traditional bell, the radio bells produce no sound and allows for remote tracking of the sheep. The main Norwegian radio bell providers are *Findmy*[9], *Telespor*[10] and *Smartbjella*[11]. In addition to being quiet, the radio bells include nifty features such as movement alerts and geofencing.

The bells, however, are pricey and dependent on signal coverage. *Telespor* and *Smartbjella* use *NarrowBand IoT* to communicate, which in turn is dependent on network coverage from either Telenor or Telia. *Findmy* is connected to low-orbit satellites and GPS which is more expensive, but also more reliable.

Figure 2: *Example of radio bells. Findmy [9], Telespor [10] and Smartbjella [11] respectively.*

Drones

Some farmers already employ unmanned aerial vehicles (UAV) in their work. UAVs can be used for manual monitoring as well as scouting for predators. Their start cost is relatively high, but they are cheap to use and can be used for several years with proper maintenance.

Summary

Despite the emergence of more modern solutions, the basic bell is still the most widespread, often mixed with the more technologically advanced solutions. Equipping large herds of sheep with radio bells is prohibitively expensive. Instead, a select few sheep are fitted with them to make general monitoring easier. The radio bells do relatively little to help with the retrieval of stragglers during

	Findmy (model 2)	*Telespor (4. gen)*	*Smartbjella 2*
Price per unit (NOK)	1890,-	1124,-	899,-
Yearly subscription cost	229,-	186,-	238,-
Seasonal subscription cost	*Not an option*	124,-	99,-
Communication strategy	low-orbit satellite, GPS	NB-IoT, GPS	NB-IoT, GPS
Battery life (years)	2-3	1	2-17
Geofencing	✓	✓	✓
Movement alert	✓	✓	✓
Stress warning	✓		
Approx. number of bells in use	40000	-	24000

Table 1: *Price and features of existing radio bells.*

phase 3 of the roundup, seeing as it is unlikely that the sheep that go missing are equipped with radio bells.

The same can be said about drones – they are a useful tool in maintaining a general overview of the herd, but without anything but the raw images from the drones, they are not very useful in retrieving stragglers either. Reviewing hours of UAV footage is time-consuming, prone to human error and several hours, if not days, out of date when the review is complete, rendering the process useless as the sheep has likely moved on.

2.3 Growth of the Drone Industry

Formerly being limited to military use, usage of drones in civilian industries has exploded over the past decade – a growth that is presumed to continue in the coming years. PwC predicts drone use in construction and mining could eventually become a $28.3 billion global market [12], and The European Commission predicts that by 2035 the European drone sector will directly employ more than 100,000 people and have an economic impact exceeding 10 billion euros per year, mainly in services [13].

Providers such as DJI offer their drone services to all fields of industry ranging from industrial surveying and inspection, to agriculture mapping and real estate media [14]. Their state-of-the-art UAVs allow for carrying heavy cinema-grade cameras. They are, however relatively limited by an airtime of 30–60 minutes per battery depending on the payload.

While enterprise is the fastest growing drone market, more inexpensive drones are becoming increasingly popular for recreational use as well. Flaring competition among drone providers is pushing down costs for these types of consumer drones – particularly among higher-end models that can shoot photos and live stream video. For example, in 2019 Parrot launched the Anafi Thermal in response to DJI's 2018 Mavic 2 Enterprise Dual – but $700 cheaper. Both devices are portable thermal imaging drones that incorporate FLIR's Lepton 3.5 miniature thermal imaging unit[15].

2.4 Object Detection

A deep learning network attempting to solve an object detection problem focuses on detecting and classifying objects in an image. Objects detected by the network are identified by a bounding box – a square surrounding the object. The bounding box is accompanied by the type of object the network believes it has detected as well as a numerical confidence value, which indicates how confident the network is that its prediction is correct. Figure 3 shows an example of the results after an object detector is applied to an image.

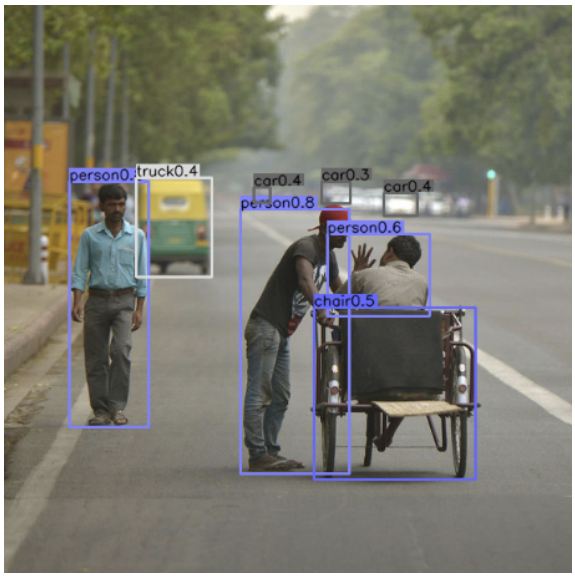


Figure 3: *An object detection model at work.*

2.4.1 The Cutting Edge

The field of computer vision is at the cutting edge of computer technology research. Established methods and models are constantly being improved upon while new ones are emerging monthly(!). An overview of the recent developments and their performance is illustrated in Figure 4.

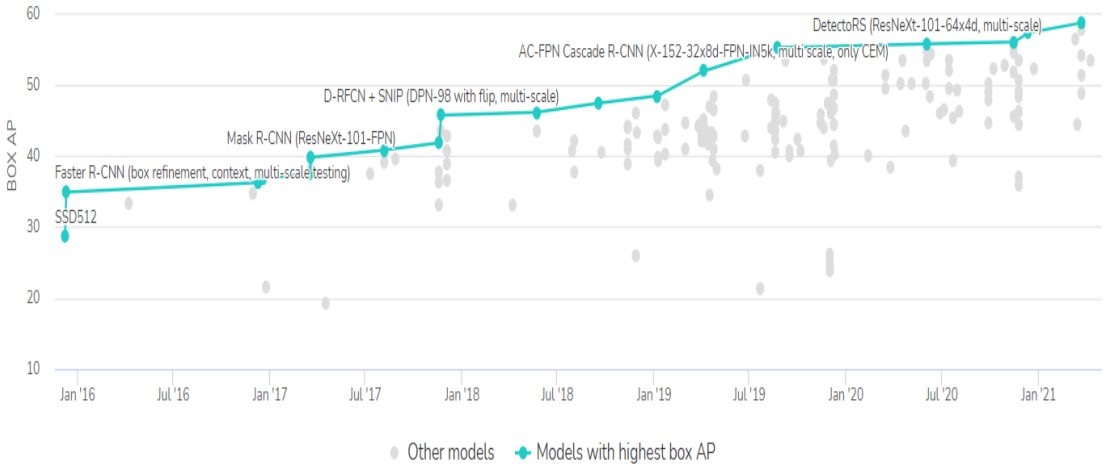


Figure 4: *Development of SOTA object detection models the last five years [16].*

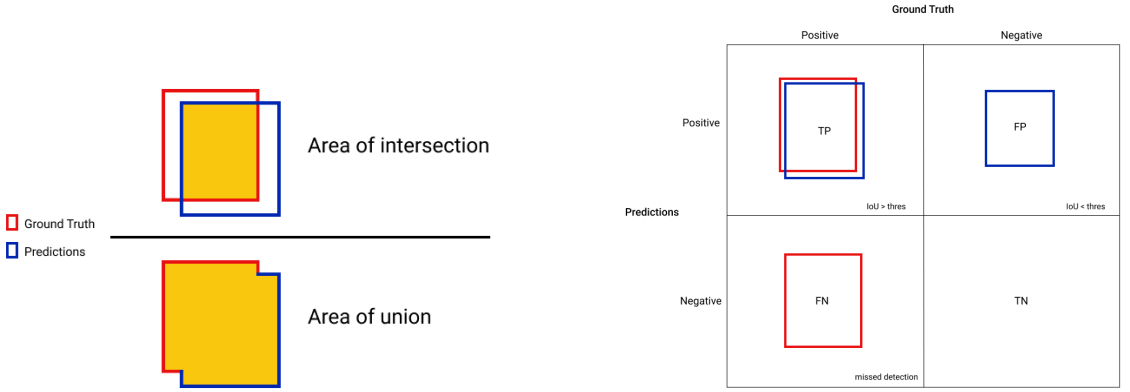
When tens of papers are published yearly, alongside several models that are made available without official research papers (such as YOLOv5), it is nigh on impossible to determine which model provides the best starting point for a thesis such as this. Intuitively one might expect simply

picking the model with the highest performance as displayed in Figure 4, but it should be noted that these performances are based on the COCO data set (Common Objects in Context)[17]. This data set provides a broad and general testing ground for object detection models, but great performance on COCO does not necessarily equate to great performance in a specialized use case such as sheep detection in the field – especially when one considers inference time and suitability for mobile hardware.

2.4.2 Object Detection Metrics

Several metrics are used to evaluate the performance of object detection models. *Precision*, *recall* and *average precision* are the most widely used. For these metrics to be used, one needs a way to determine whether the model's predictions are correct.

For this, *intersection over union* (IoU) is used. IoU compares the model's bounding box to the ground truth – a «true» bounding box as set by a human, as illustrated in Figure 5a. If a certain overlapping threshold is met between the predicted bounding box and the ground truth, the prediction is counted as correct, or a *true positive* (TP). If the threshold for a ground truth fails to be met, the model fails to identify an object, this is a *false negative* (FN). If the model predicts a bounding box where there is none, it is counted as a *false positive* (FP). Finally, if the model predicts nothing where there is no ground truth, we have a *true negative* (TN). The prediction classes are illustrated in Figure 5b.



(a) *Intersection over union (IoU)* **(b)** *The prediction classes TP, FP, FN and TN*

Figure 5: *All detections are evaluated and categorized in prediction classes using IoU.*

The precision metric is a measure of the accuracy of the predictions made by the model. It is calculated as shown in Equation 1. A high-precision model rarely gives false positives, but the metric does not consider missed detections. A model that correctly predicts one sheep in a herd of one hundred will have a precision of 100% despite its poor performance.

The recall metric measures how many of the ground truths the model is able to find. It is calculated as shown in Equation 2. A high recall model can find most objects in an image, but the metric does not consider false positives. A model that predicts 1000 sheep in a herd of 100 will have a recall of 100% despite its poor performance.

$$Precision = \frac{TP}{TP + FP} = \frac{correct\ predictions}{all\ predictions} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} = \frac{correct\ predictions}{all\ ground\ truths} \tag{2}$$

As such, there is a trade-off between precision and recall, often visualized by a *precision-recall graph*. Which of these metrics are to be maximized depends on the use case. If every single object must be correctly identified one might opt to maximize recall, despite the false positive noise that

must be handled. On the other side, if false positives are too expensive to handle, one might want to prioritize precision despite missing out on some of the harder-to-spot objects in the images.

Average Precision (AP) takes both precision and recall into account by averaging precision over a range of recall values – in practice calculating the area below the precision-recall curve. This makes AP a good metric to evaluate the overall performance as well as the potential of a detector.The AP is often used with an additional number like $AP.5$/AP@0.5, which means the IoU threshold of a detection is 50%.

A model with a high AP-value can still be lacking in precision. This suggests that the model has high confidence in its correct predictions and low confidence in its wrong ones – meaning that its confidence threshold can be increased to boost precision without a significant loss in recall.

Conversely, if precision is high and AP relatively low, the confidence threshold may be lowered to increase recall and AP. Hopefully without too much of a hit to precision.

2.4.3 YOLO – You Only Look Once

The first version of the *You Only Look Once* (YOLO) object detection algorithm was introduced by Josh Redmond in 2015 [18]. It demonstrated competitive results thanks to its new approach with its single convolutional neural network (CNN) used to predict both bounding boxes and class probabilities. Its name is derived from the fact that each image only has to pass through the network once for a prediction to be made. The single-stage approach to detection has several advantages, the greatest being that it is very fast, which allows for real-time analysis. Additionally, it learns generalized representations of objects very well. Compared to other multistage detectors, YOLO «sees» and considers the entire image which allows it to reason globally and use contextual information about objects. The architecture of YOLO has been gradually improved and released in new versions since the first in 2015.

YOLOv5

The latest version of the algorithm is *YOLOv5* [19], and is today considered state-of-the-art for real-time object detection – still based on the single stage-network principle of the first publicized version. An overview of the YOLOv5 architecture layers is shown in Figure 6.

YOLOv5's performance on the COCO dataset [17] can be seen in Figure 8. This data set contains 330,000 images with varying objects in a 640x480 resolution, and is widely used to evaluate the performance of object detection models.

The YOLOv5 architecture is primarily available in four different sized models (s/m/l/x) which contain increasingly more parameters. Because of this, the larger models tend to produce better results, but require more memory and are slower to run. The checkpoints' statistics and performance can be seen in Figures 7 and 8.

YOLO uses its own format for ground truth labeling. Every image file is paired with a text file where each of the image's ground truth bounding boxes are represented with five numerical values: Object class index, center x-value, center y-value, bounding box width and bounding box height. A labeled image with its corresponding label file can be seen in Figure 9.

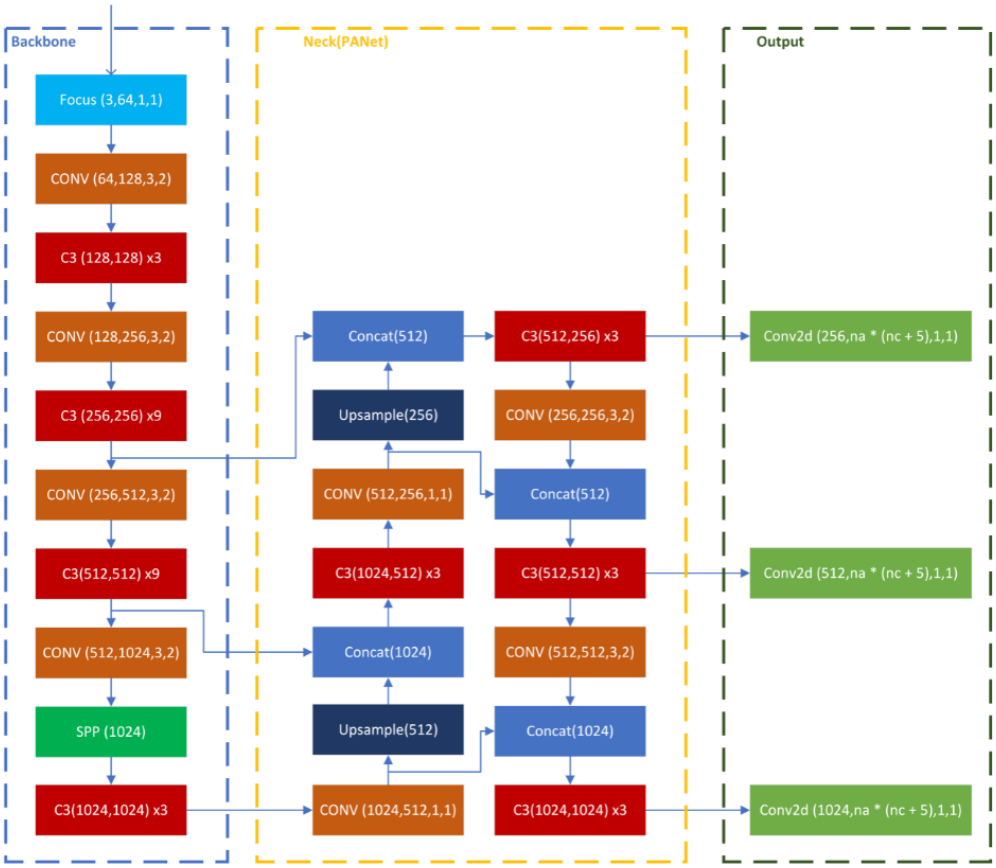


Figure 6: *YOLOv5 architecture illustrating the backbone layers, the network neck and the detection output.*

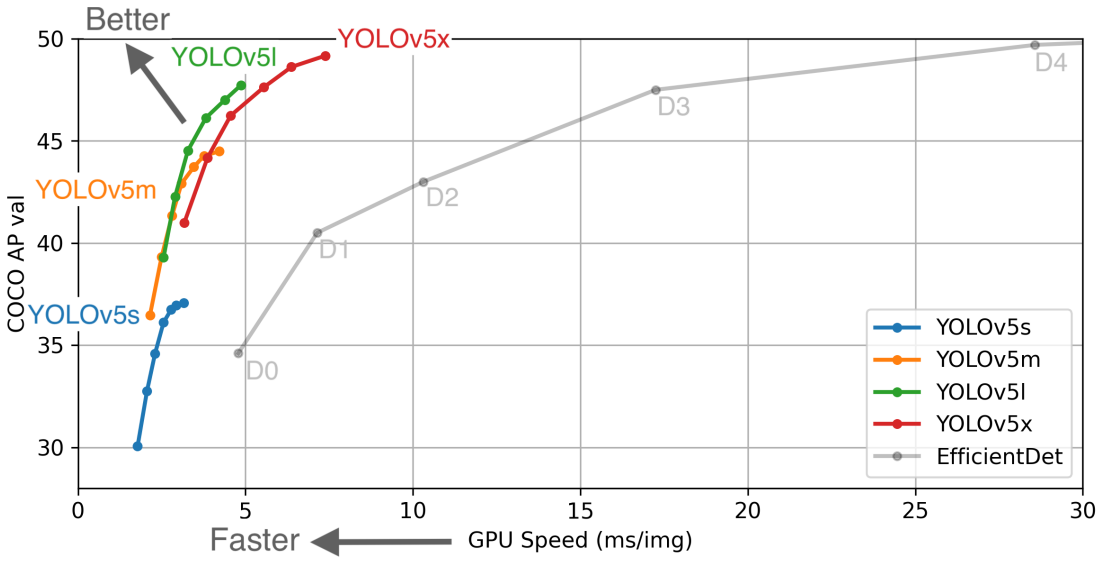


Figure 7: *Pretrained checkpoints' performance for different sizes of the YOLOv5 architecture on the COCO data set. Also includes the performance of a competing architecture EfficientDet. Image origin: [20]*

Model	AP^{val}	AP^{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}		params	FLOPS
YOLOv5s	37.0	37.0	56.2	**2.4ms**	**416**		7.5M	13.2B
YOLOv5m	44.3	44.3	63.2	3.4ms	294		21.8M	39.4B
YOLOv5l	47.7	47.7	66.5	4.4ms	227		47.8M	88.1B
YOLOv5x	**49.2**	**49.2**	**67.7**	6.9ms	145		89.0M	166.4B
YOLOv5x + TTA	**50.8**	**50.8**	**68.9**	25.5ms	39		89.0M	354.3B
YOLOv3-SPP	45.6	45.5	65.2	4.5ms	222		63.0M	118.0B

Figure 8: *Pretrained checkpoints' performance for different sizes of the YOLOv5 architecture on the COCO data set. Image origin: [20]*

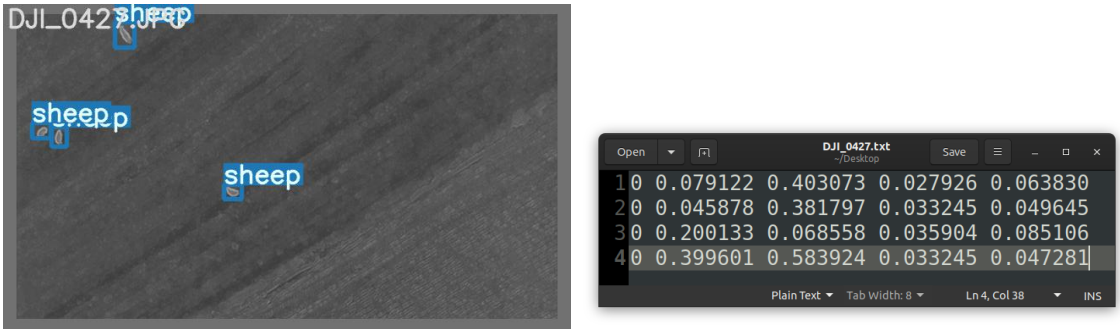


Figure 9: *An example image with its labels displayed, and the corresponding text label file on YOLO-format.*

2.4.4 Detection of Small Objects

COCO evaluation results for recent SOTA algorithms show models struggling when it comes to detecting small objects compared to the larger ones. In some instances, the AP for small objects is a fifth of that of the larger objects [21].

Sheep in drone images will in most cases be relatively small objects to detect, depending on the flight height. Some measures that can help to detect small objects are maximizing the capture resolution of images and increasing the detection model's input resolution. This will increase the richness of features the object detector may form for small objects.

Image Tiling

Tiling an image means dividing it into equal-sized tiles, with each tile keeping its original resolution and detail. Tiling can be done as a preprocessing step and will effectively zoom in on small objects while maintaining small input resolution for the detection model. The small resolution will help its ability to run fast inference.

In theory, tiling allows for the processing of a high-resolution image without having to handle it in its entirety all at once. This might make a smaller model with fewer parameters sufficient to handle all the information in a 4K-image. In addition, the lower resolution of each image speeds up training, even though there are more images in total.

Tiling might lead to increased inference times, depending on how the inference is performed. While the information that needs to be processed is the same, a single 4K-image only needs to be loaded

and unloaded to and from the model a single time. A tiled image might load 50, albeit smaller, images in 50 separate operations. This overhead might lead to a prohibitively expensive time loss. It is possible to remove this added time by performing inference on non-tiled images, but this will likely negatively impact prediction scores, as the model has trained on smaller, tiled images [22] [23].

2.5 Deep Learning on Mobile Devices

Deep learning strongly depends on and thrives with a lot of processing power, memory and storage capacity. The drive to maximize the accuracy of deep learning models has led to an increase in model size and a reduction in power efficiency [24]. This section covers some measures and tools to help make a deep learning model suitable and able to work on smaller devices.

The TensorFlow Lite Framework

Many deep learning frameworks come with tools for mobile deployment of machine learning models.

TensorFlow Lite [25] (TFLite) is a well-known and documented framework specialized for mobile devices or IoT gadgets. TFLite is part of the TensorFlow framework developed and maintained by Google. The key points of TFLite are converting and interpreting models to optimize for speed. The conversion focuses on reducing the model's size to enable storage while the interpretation optimizes the processing speed.

Model Quantization

Quantization of deep learning models is done by reducing the precision of the numbers used to represent the parameters – by default a 32-bit floating number. A reduction to 16-bit floating numbers, which only yields a small reduction in precision, will reduce the model size by 50% [26].

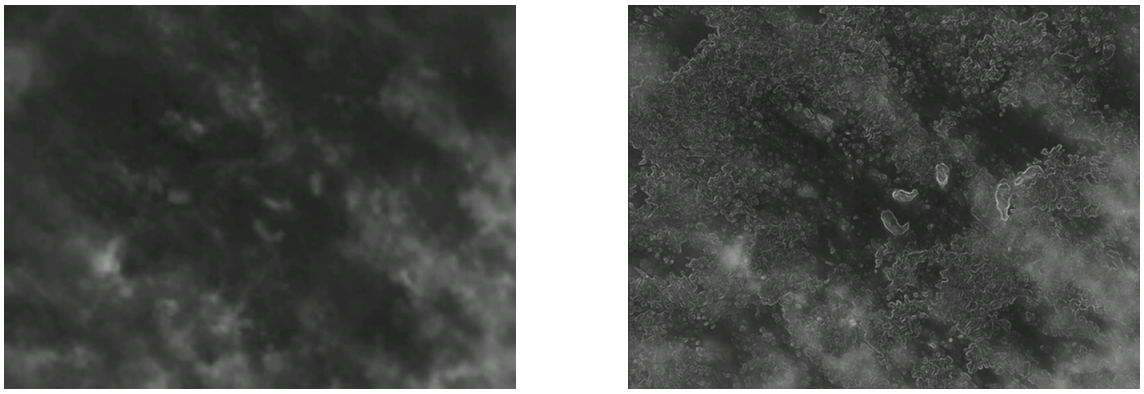
GPU Delegation

Graphical processing units (GPUs) are typically more efficient than CPUs on highly parallelizable tasks like deep learning. The reason is that deep learning models consist of a huge number of operators, each working on an input tensor that can easily be divided into smaller workloads and carried out in parallel [27]. Most new smartphones have a dedicated GPU that can be used to speed up detection tasks. In addition, the GPU carries out computations efficiently and consumes less power than if the same task were to be performed on a CPU.

2.6 MSX – Multi-Spectral Dynamic Imaging

MSX is a FLIR-patented technology that processes digital features in real-time [28]. In this use case, it enhances the UAV's infrared images by using features from the onboard digital camera, while retaining the same resolution as the original IR image. MSX allows for easier target acquisition without compromising the thermal data and makes outlined details easier to see by superimposing high contrast features from the RGB images onto the thermal images [29]. An example of a regular thermal image versus MSX is shown in Figure 10.

Performing detection on a single image with both visual and thermal features has the potential of saving time on mobile devices where each detection is slower. Although inference time could improve due to the low-resolution, the loss of information compared to a high-resolution visual image is significant.



(a) *Regular thermal image* **(b)** *Thermal image using MSX-technology*

Figure 10: *Example of normal thermal image versus the same image using MSX.*

2.7 Related Thesis Work

Svein-Olaf Hvasshovd [30] at NTNU has supervised several theses that have tackled problems related to sheep retrieval – often focused on object detection using deep learning – over the past few years. This work makes the foundation for this project.

2.7.1 Previously Collected Data

Since the autumn of 2018, students working on related theses have been collecting images for a dataset to be used by themselves as well as future projects to come. In this thesis we only make use of images taken from August 2019 until today, all captured with the same UAV as described in Section 3.2.

An overview of the dataset as of August 2020 is given in Table 2. Note that the UAV always captures images in pairs – one with its regular RGB-camera and one with its thermal camera, but not necessarily using MSX-technology on the thermal image. The column named *MSX-images* indicates the number of thermal images that were captured using the UAV's MSX-mode as described in 2.6.

When	Where	Total images	MSX images
May 2019	unknown	170	47
Aug 2019	Storlidalen	1476	432
Sep 2019	Storlidalen	800	5
Oct 2019	Storlidalen	309	0
May 2020	Klæbu and Orkanger	222	0
Total		**2977**	**484**

Table 2: *Summary of the existing data set of images*

2.7.2 Combining RGB and IR-models to Improve Performance

K. Johannessen's thesis *Towards Improved Sheep Roundup* [4] provides the theoretical backbone of our work. Where earlier theses only utilized RGB-images and by now outdated methods of object detection, Johannessen uses both RGB and IR-images in deep neural network models. It is shown that a fusion of two separate models – each fitted to one of the image types – yields better results than using the two independently. This strategy exploits both the information of body heat in the IR-images as well as the texture and color in the RGB-images to great effect.

Johannessen provides a detailed study of the impact of fusion depth, model complexity and image resolution on the precision and inference time metrics.

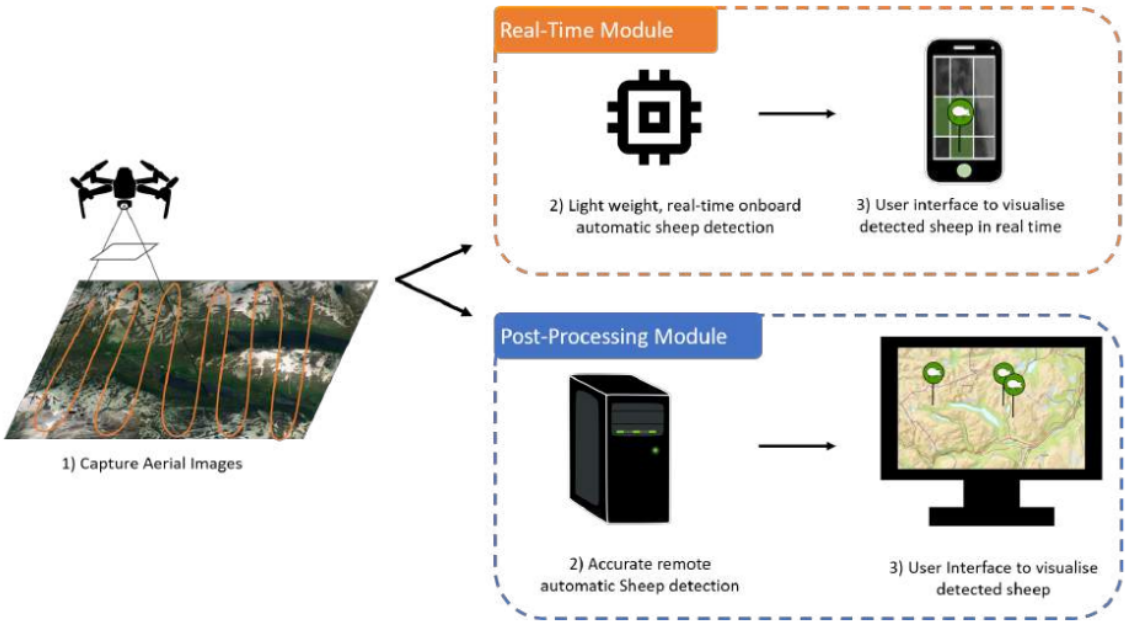


Figure 11: *Envisioned system from previous work by K. Johannessen [4]*

As part of the study, an effort was made to expand the size of the dataset of relevant drone images, increasing the amount of data tenfold. The majority of images used in training the models presented in this thesis come from this dataset.

Based on their findings, Johannessen suggests several approaches for future work.

Alternative network architectures should be explored. Deep learning evolves at a rapid pace, and the *ResNeXt*-architecture – the newest architecture presented in Johannessen's thesis – had already been around for four years at the time.

Alternative approaches to object detection should also be explored. Johannessen's solution outputs grid probabilities – an approach that reduces processing time at the expense of precise position location, when compared to bounding boxes. Further studies should investigate whether this trade-off is justified.

Finally, the more external factors that might impact the system's performance should be studied. The altitude at which images are captured and the use of MSX-images are seen as the main ones.

2.8 Experimental Results from Specialization Project

The training results from our specialization project form the basis for the method and final mobile models presented later in this thesis.

The training resulted in several models which were deemed suitable for further experimentation and to improve upon. Following is a summary of the results and a short discussion around them. A complete presentation of training results was presented in our specialization project [5].

MSX-model

This MSX-model was trained using separate training (334 images) and validation (43 images) sets – the same ones used for training the finalized models presented later. The image resolution was set to 640p, which is the native resolution for the MSX-images. The model was trained for 400 epochs.

The model was then tested on the same independent test set as the RGB-models, with the corresponding MSX-images. Some example predictions are shown in Figure 12a, accompanied by the model's prediction-recall curve from the test set in Figure 13a.

Downscaled 1024p RGB-model

This RGB-model was trained using separate training (334 images) and validation (43 images) set. The image resolution was set to 1024 (downscaled from the original 4056). The model was trained for 200 epochs.

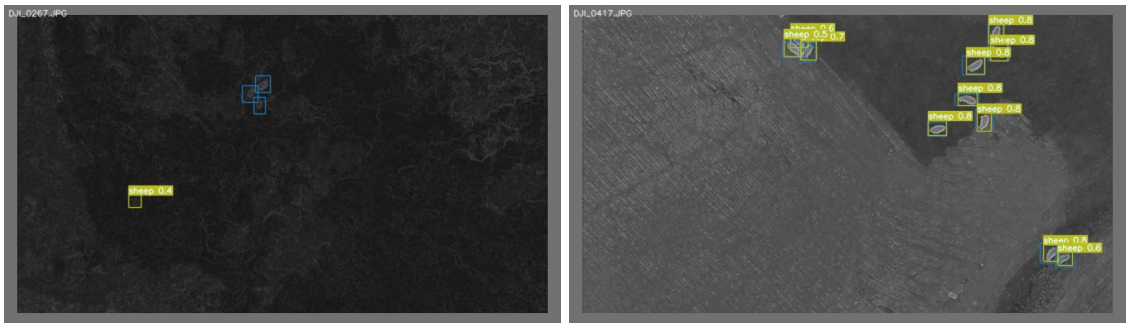
Samples of its test predictions are shown in Figure 12b.

We can see that the 1024p-model provides results comparable to that of the full-resolution model. The model is able to pick up most sheep in the open while it struggles more with partially covered sheep. Notably, this model seems more prone to marking small, white details as sheep than the 4064p-model. The false positives tend to have lower confidence than that of the correct ones. Figure 13b shows the precision-recall graph of the model.

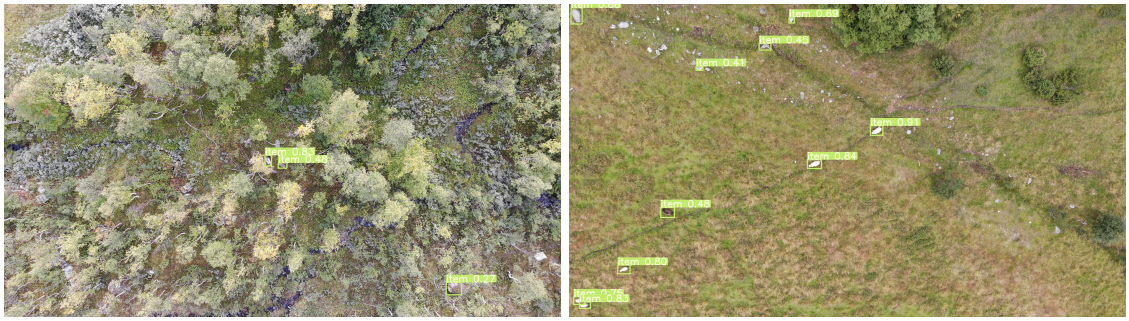
Full Resolution 4064p RGB-model

The model was trained for 200 epochs using identical training and validation sets as the previous downscaled model. Due to the high native resolution of the visual images and limited memory of the hardware used for training, this model uses a batch size of 8 instead of the default 16. Other control variables are the same as the RGB-model trained with downscaled images.

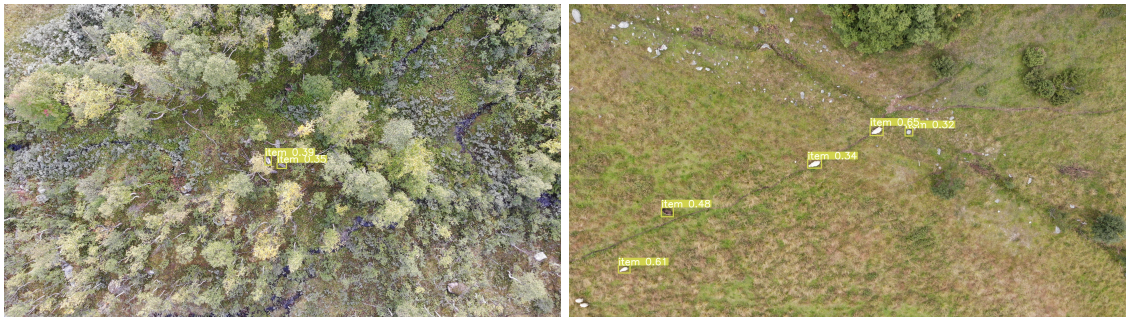
When exposed to the test set, the model is able to detect sheep as demonstrated in Figure 12c. It is not as prone to false positives as the model using a lower resolution.



(a) *Full resolution 640p MSX-model*

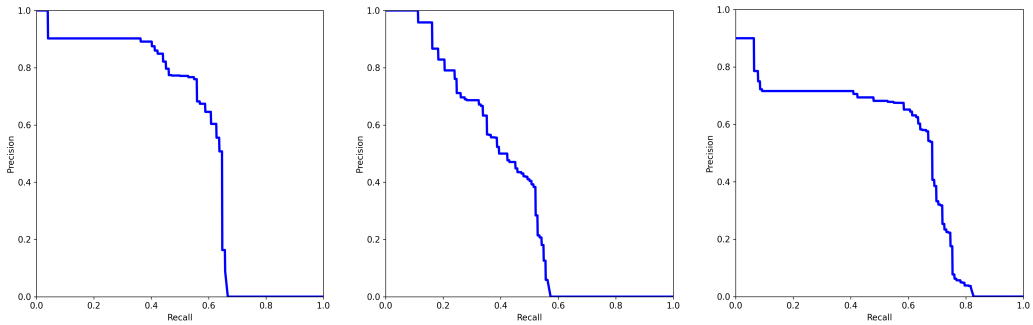


(b) *Downscaled 1024p RGB-model*



(c) *Full resolution 4064p RGB-model*

Figure 12: *Example of predictions on the test data set by MSX, downscaled 1024p RGB and full resolution 4064p RGB-models.*



(a) *640p MSX-model* **(b)** *1024p RGB-model* **(c)** *4064p RGB-model*

Figure 13: *Precision-recall graphs for a MSX and the two RGB models. The resulting numerical AP can be seen in Table 3*

2.8.1 Summary of Results

Table 3 contains the models' detection performance metrics on the test set – all tests were run on computer hardware.

Model name	Image resolution (px)	Precision	Recall	Average Precision	Inference time (s)
MSX	640	**0.774**	0.627	**0.546**	**0.032**
RGB_1024	1024	0.367	0.521	0.389	0.079
RGB_4064	4064	0.531	**0.683**	0.510	1.112

Table 3: *Summarized performance of experimental models on the test set. Best values for each column marked in bold.*

We expected the full resolution 4064p-model to outperform the 1024p-model, and this did end up being the case for all detection metrics. There is a notable difference in inference time between the two RGB-models, as to be expected when the image resolution is different. This tells us that the input resolution of images is an important factor when it comes to inference time, and it might also impact the model's performance.

The MSX-model had the highest precision and average precision, but did not beat the 4064p RGB-model's recall. This seems to be caused by the MSX-model relying on contours around objects in contrast to their surrounding. As such, its predictions are mostly correct, but it struggles in detecting darker sheep with a less distinct outline. This was somewhat contrary to our expectations regarding the MSX-images, as we believed they should rely more on thermal signatures rather than visual features transposed onto the thermal image by the MSX-software.

2.9 SOTA Summary

The traditional bell is still the most widely used tool in tracking grazing sheep, but farmers have started experimenting with more technological solutions such as radio bells and UAVs.

Earlier work under supervisor Hvasshovd has shown that deep learning models can be trained to spot sheep in images captured from relatively cheap UAVs, utilizing both regular and thermal images. This work has also produced a data set of images for use in further work.

The drone industry itself is growing – service providers make use of drones in most industries by now, and competition between drone manufacturers lowers the prices to the point where industrial purpose drones are affordable to small business owners.

Object detection using deep neural networks has seen a significant boost in power over the past years, with new and improved models emerging constantly. This, combined with the growing power of mobile devices makes it possible to run relatively high-performing object detection on mobile devices.

3 Project Description

A finalized product would be a mobile application able to connect to a UAV, perform sheep detection analysis on its image feed, and provide instant feedback to the user, as illustrated in Figure 14.

The first step towards this goal, and the practical purpose of this thesis, is to develop and evaluate detection models able to run on a mobile device. We will develop a testing environment for such models in an Android application able to connect to a UAV, and evaluate the models based on their performance running in the application.

Earlier work has shown that models from deep neural networks can be used to detect sheep in images captured by UAVs, but real-time feedback and hardware restrictions of mobile devices have not been taken into consideration. These factors will be key points in this thesis.

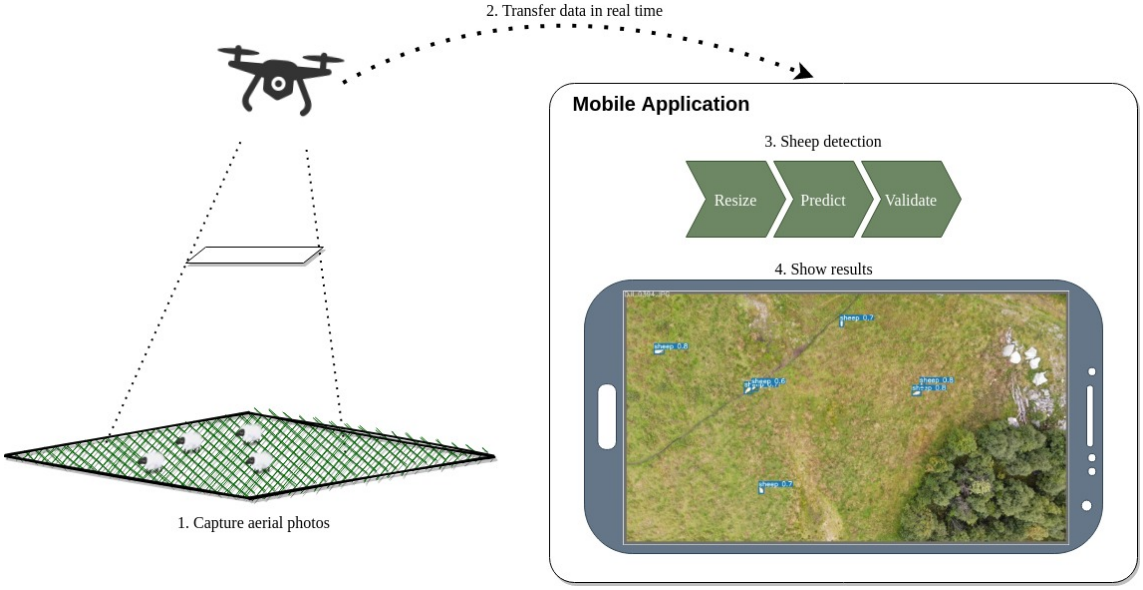


Figure 14: *An overview of the envisioned solution using a mobile application for processing drone video and perform sheep detection.*

3.1 Requirements

The following requirements must be met for the MVP to be successful.

R1: Real-time Feedback

The user should be given real-time feedback while the UAV is scanning the pasture area. This requires the images to be processed by the object detection model at least at the same speed at which they are captured by the camera. It is difficult to pinpoint the exact rate at which a drone in a finalized product would be capturing images. The rate is mostly dependent on the speed at which the drone travels. Naturally, a slow-moving drone allows for a longer processing time. Increasing the altitude of the drone also allows for longer processing time, but this requires the model to be able to detect sheep from higher up.

As such, we believe a processing time of below 1 second will be fast enough for a real-time mobile object detection MVP.

Instant feedback is a necessity, as the alternative would be a post-flight analysis which – depending on the pasture size – might be several hours after the image was captured, at which point the sheep

might have moved.

R2: Precision and Number of False Positives

The sheep detection model's precision and recall performances should be good enough to show potential to be used in a finalized product. These metrics are described in section 2.4. There is always a trade-off to be made, and we believe precision is the most important of the two – or rather, avoiding false positives is of utmost importance. If we imagine an image processing rate of 0.5 seconds, one false positive per 20 images would mean one false notification to the user every 10 seconds. This might be acceptable in short intervals, but during hour-long searches this would quickly become too tiresome to handle, and should the false positive rate be any higher, the system would lose a lot of effectiveness. Ideally, false positives are avoided altogether, but realistically the goal is to keep the rate at which they appear low enough for the user to comfortably be able to filter through them.

R3: Recall vs Actual Sheep Retrieval

Keeping the FP-rate low means sacrificing recall. This might seem counter-intuitive as the system's main purpose is to help farmers find all their sheep. There are, however, a couple of key points to consider: Firstly, sheep, even the stragglers, rarely venture alone. As such, when flying over a group of wayward sheep, it is sufficient to detect only one of the sheep for the whole group to be found by the user. Furthermore, a single sheep is likely to be present in several images, as there will be a certain overlap between images. It is sufficient for the model to detect the sheep in only one of them for the sheep to be found. Depending on the altitude and speed of the UAV, the angles from which the sheep is seen might change, or the sheep might be triggered to move – both of which might improve the chances of the sheep being detected at least once.

As such, determining a target recall value is difficult. Imagine the drone flying over a small herd of five sheep, capturing three images. If the model only detects two separate sheep in total, the recall value is only a measly 13%, yet all of the «fifteen» sheep end up being found in the end.

For the MVP to be successful, it should be able to realistically retrieve a large portion of the sheep it is presented. A metric for measuring the fulfillment of this is proposed in Section 4.6.1.

R4: The Model Should Run on Mobile Devices

For the MVP to be usable in the field, it must be able to run on mobile devices. The most taxing part of the system is running the object detection model. As such, the model to be used must be light enough to not require specialized GPUs to run effectively, but rather should run on a smartphone or a portable computer. For the MVP to be successful it should run on a smartphone without hampering the performance too much.

3.2 Hardware Constraints

This project is dependent on a lot of hardware used for different purposes. The image quality, computing processing power for training and testing are two examples limited by hardware constraints. This section includes the hardware and its specifications used throughout this project. An overview of this is shown in Table 4.

Hardware Unit	Primarily used for
DJI M2ED-drone	Capture IR and RGB images of sheep
The Idun cluster	Training deep learning sheep detection models
Dell XPS 9560	Testing sheep detection models using computer hardware
Huawei P30 Pro	Testing sheep detection models using smartphone hardware

Table 4: *An overview of the hardware units used in this project and what they are used for.*

The Drone

The Mavic 2 Enterprise Dual (M2ED)[31], seen in Figure 15, is the UAV used to expand the data set. The drone's key specifications are outlined in Table 5.

Takeoff weight:	899 g
Dimensions (l x w x h):	Folded: 214x91x81 mm Unfolded: 322x242x84 mm
Max flight time:	31 min
Max speed:	71 km/h
Visual camera	
Image size:	4056 x 3040 px
Thermal camera	
Model:	FLIR longwave
Image size:	640 x 480 px

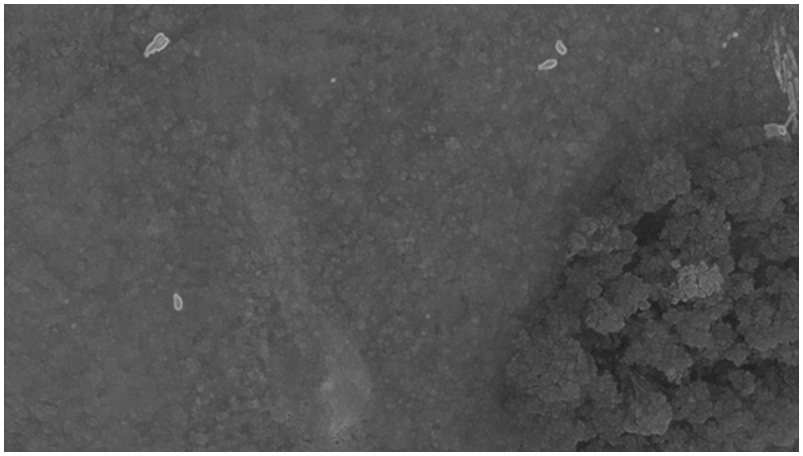
Table 5: *DJI M2ED specifications*

Figure 15: *DJI Mavic 2 Enterprise Dual*

The drone is suitable for use in an MVP due to its relatively low price while still providing a high-resolution camera as well as access to FLIR thermal imaging. When capturing images both

cameras capture images immediately after each other. This allows for the creation of two separate models using virtually identical images. It must be noted that since there is a physical offset and resolution difference between the two cameras, the resulting images are not exact RGB/IR-copies of each other. This can be seen in Figure 16.

(a) *RGB-image*



(b) *MSX-image*

Figure 16: *Corresponding RGB and MSX-images captured with the M2ED drone. Note the smaller frame and lower resolution of the MSX-image.*

While the drone is good for the relatively small-scale experimentation done in this thesis, it is lacking on several points which makes large-scale solutions difficult. A single battery is only enough for roughly 30 minutes of airtime – even less while recording or capturing images. If one factors in take-off time, camera adjustments and flight distance to the area to be surveyed, the effective flight time can be significantly less than 30 minutes.

The M2ED-drone is also limited by a 125-meter altitude restriction in relation to its launch point. In the highly undulating mountainous terrain, such an elevation limit makes covering large areas challenging, and creating a realistic testing scenario for a final industrial product is difficult. Ideally, such a product would use a larger UAV flying at a higher consistent altitude over varied terrain to cover the large areas.

The Idun Cluster

The Idun cluster [32] is used to train every sheep detection model. Idun is an NTNU project that aims at providing a high-availability and professionally administrated computing platform for NTNU, allowing for rapid testing and prototyping of HPC-software.

The cluster provides both CPU and GPU power, but only GPUs were used in the work presented here. The GPU power required for a training session is dependent on several factors – batch size and model size being the most impactful ones. Depending on the memory requirements, training was performed on 1–4 NVIDIA Tesla V100 16GB/32GB GPUs [33].

Idun uses the Slurm Workload Manager [34] to manage the provided resources and to schedule jobs on these resources. This means that if one wants a lot of resources for a relatively long training job, one must wait longer for those resources to be reserved than if the job was shorter or less demanding of resources. As such, it is sensible to start with shorter, experimental model training runs to get a feel for how the model will perform, rather than committing to a several days-long wait before a week-long job from the get-go. This strategy for training is reflected in the results presented later.

Personal Computer

All computer inference tests of sheep detection models were run on a personal computer: A Dell XPS model 9560 laptop, with an Intel® Core™ i7-7700HQ CPU @ 2.80GHz and a dedicated NVIDIA GP107M (GeForce GTX 1050 Mobile) GPU.

Smartphone

The device used for smartphone inference tests is the Huawei P30 Pro. This smartphone, though a couple of years old, can be characterized as a well-performing phone from the normal consumer market. It sports 6 GB of RAM and a Kirin 980 chipset with an eight-core CPU and the ARM Mali-G76 MP10 GPU [35].

4 Method

This section describes our approach to answer the research questions as presented in Section 1.2. We describe the collection and preprocessing of data, the development of the Android application, the training of the detection models and the metrics they are evaluated by.

4.1 Data Collection

In general, deep learning models are dependent and based on large amounts of data, and expanding the data set will expose the model to new environments and situations. A larger data set will, in almost all cases, contribute to a better performing deep learning model. We expanded on the existing data set by capturing more images, using the UAV provided by supervisor Hvasshovd, described in Section 3.2.

Data Collection Field Trip

Data collection was carried out in Storlidalen, Oppdal over two days in early September 2020. During this period, some sheep were still grazing freely and some had been retrieved and returned to fenced areas around the local farms.

The timing was ideal to guarantee images of the fenced sheep, while also giving a possibility to search for and capture images of free-ranging sheep. Images of free-ranging sheep are preferable because they better represent the challenging conditions, vegetation and background terrain that the sheep detector model would face in real applications. Weather conditions were partly cloudy at about 5–10 degrees centigrade.

Approximate areas in which the drone captured images can be seen in Figure 17.

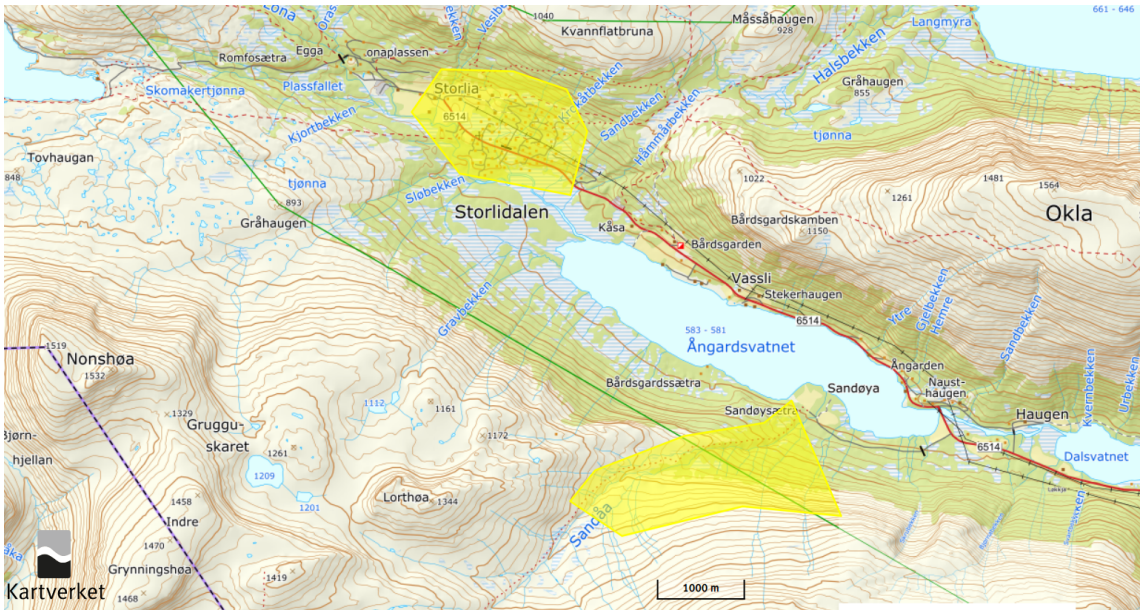


Figure 17: *Marked in yellow are areas covered by UAV during data collection in Storlidalen.*

The focus of the first day was sheep grazing in fenced areas close to the farms and cabins in and around Storlia. The fenced areas mostly consist of relatively flat terrain or fields. This would guarantee some images of sheep, as well as allowing for adjustment of camera settings, before heading into the mountains the following day. The second day we followed the river Sandåa from its mouth at Ångardsvatnet some kilometers up the mountainside. This area featured varying terrain with different degrees of vegetation.

In total, images were captured in 7 separate batches. Five containing sheep in fenced areas and two containing free-ranging sheep in a more realistic testing environment further away from the farm.

The M2ED drone features an MSX-mode where visual and thermal images are combined, as described in section 2.6. As suggested by Johannessen, further experimentation with the use of MSX-images would likely be a worthwhile endeavor [4]. Expanding the set of MSX-images was therefore prioritized during the field trip.

The MSX-images were captured using MSX-mode, grey thermal palette and minimum and maximum temperature limits were set to 0 and 45 degrees centigrade respectively.

4.2 Data Preprocessing

Both new and existing data had to be preprocessed to best make use of the YOLOv5 architecture.

Data Selection

In order to make use of and properly evaluate the models using MSX-images, we deemed it suitable to compare them to «identical» models using RGB-images. Meaning a model trained with the corresponding RGB-images (as described in Section 3.2 the drone captures images in pairs). As such, RGB-images that did not have corresponding MSX-images (recall Table 2) were not considered for use in the training of the model.

Earlier work has also shown that object detection is most effective and precise when images are taken from roughly the same altitude [4]. This led to the exclusion of images captured from a significantly higher or lower altitude than the average. No formal height thresholds were decided upon, and pruning was done manually by simply removing images that obviously differed from the average.

The pruned data set was then divided into training, validation and test data sets. There is no definitive best ratio of images, but a commonly used distribution is 80%, 10% and 10% for training, validation and test respectively was decided to be used [36].

Images were chosen so that similarity between the sets was minimized. This meant making sure images of the same herd were not present in more than one set, as well as making the sets as geographically independent from each other as possible. Low similarity between the sets should help in creating a model which is able to generalize well and recognize objects that differ from the ones it has seen during training. This independence is particularly important in the test set, as it should expose whether the detector can fulfill its purpose on a real data set.

Data Annotation

Data annotation, or labeling, is the process of marking the images with tags for the object detector to train and evaluate against. Different object detection algorithms use different annotation formats. The old pictures utilized the *Pascal visual object classes*-format(VOC), which represent bounding boxes by two pixel values indicating opposite corners of the box. As described in Section 2.4.3, YOLO uses a different format.

Thus, the old images needed to be converted from VOC to YOLO. This was done by calculating the width, height and center of the bounding box. Then, each value is normalized from pixel coordinates to an absolute value between 0 and 1.

The new images were labeled using the open-source tool *MakeSense* [37]. MakeSense generates labels in both YOLO and Pascal VOC formats, allowing for both to be used in future work.

Image Tiling

As described in Section 2.4.4, tiling images as a preprocessing step might increase the model's ability to help detect small objects. The 4056x2280p original images were divided into 28 smaller images, most of them 640x640p in size. The edge-tiles on the right and lower sides of the original image are slightly lower and thinner. An example of tiling an image can be seen in Figure 18.

The process of tiling images will result in many tiles not containing any sheep. The training set was first pruned to only include images containing sheep. Then, a set of empty tiles (roughly 10% of the set size) was added to give a basis for empty images. For the validation and test sets, all images are used and the sets therefore contain many empty tiles.

Even though the sets consist of the same images, the non-tiled sets are in practice much smaller than the tiled ones, because the models will be exposed to significantly fewer unique training and validation image instances. By including the empty tiles in the validation set, we hope the model will learn to avoid terrain, thus lowering the number of false positives. The script used to tile images and transform the labels to correct tiles is described in Appendix B.2

(a) *Original image*

(b) *Overview of tiled image files*

Figure 18: *Sample of an image in the data set, the original size and the tiled image files of this image. The generated tiled image size is 640x640p.*

4.3 Deep Learning Model Training

This section covers the methods used to train several object detection models, which hardware and software were used, which data was used, and which training parameters were used.

4.3.1 Training with YOLOv5

The YOLOv5 repo [20] offers a wide variety of object detection accessories based on the YOLOv5 architecture.

To train a YOLOv5 model one has to run the *train.py* script, which can be configured with customizable training parameters depending on the desired model. The most impactful parameters include number of training *epochs*, *batch size* and *image size*. Configuration of these is discussed further in Section 4.6.

Other parameters the script requires to be correctly defined are *weights*, *data* and *device*. The weights parameter defines the path to a pre-trained set of weights to use as a starting point for training the model – if left empty, the weights are randomly initialized. Every trained model presented later will have used one of these pre-trained checkpoints provided by YOLOv5. Their details can be seen in Figure 7.

Using pre-trained checkpoints is considered sensible for several reasons. Firstly, the checkpoints have already demonstrated state-of-the-art performance. While they have not been directly trained in detecting sheep in UAV-images, their results indicate an inherent ability to generalize and learn to recognize objects well. Secondly, the size of the available data set, as well as time limitations, make it unrealistic to train a well-performing model from scratch. The pre-trained checkpoints have had hundreds of thousands of images available to them, while we have only a fraction of that.

Furthermore, the YOLOv5 framework allows for the configuration and evolution of hyperparameters. All hyperparameters were kept at their default values for the models presented here.

After a training run has been completed YOLOv5 provides a breakdown of the models' performance during training. This breakdown includes graphs showing the development of recall and precision for each epoch as well as showing the model's predictions on a subset of the validation images. An example of a breakdown such as this is seen in Figure 19. Alongside statistics, two sets of model weights: *best* and *last*. The *last* is intuitively the model weights as a result of the final epoch. The *best* weights are determined by a customizable function set to reward the epochs with the highest reported recall, precision and mean average precision.

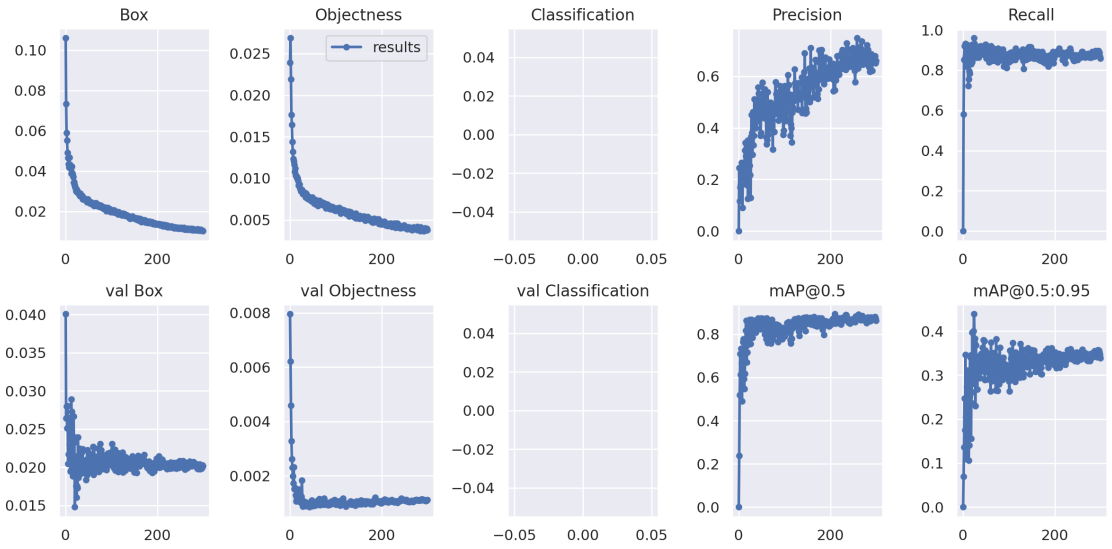


Figure 19: *An example of the breakdown of a model's performance after training is completed.*

The model weights from the training run are then tested on a set of images it has not been exposed to before. It is possible that the model simply has learned the images in the training set – showing very good results during training – but will fail when exposed to an independent test set. This testing does not require a high-performance GPU to carry out and can be run on personal computers. The tests using computer hardware were run on the computer described in Section 3.2.

Models that show workable results are then converted to a format better suited for mobile devices for further testing on mobile hardware, as described in Subsection 4.4.

More Data Visualization with Weights & Biases

To easier evaluate how different models performed against each other, the online tool *Weights & Biases* (wandb) was used [38].

Weights & Biases helps in keeping track of machine learning projects – with tools to log hyper-parameters and output metrics from runs, then visualize and compare results and quickly share findings. Training output is logged to wandb (in addition to the default logging by YOLOv5 – which is stored locally) where it is cleanly visualized – allowing for performance evaluation of several models at a glance. An example of this can be seen in Figure 20.

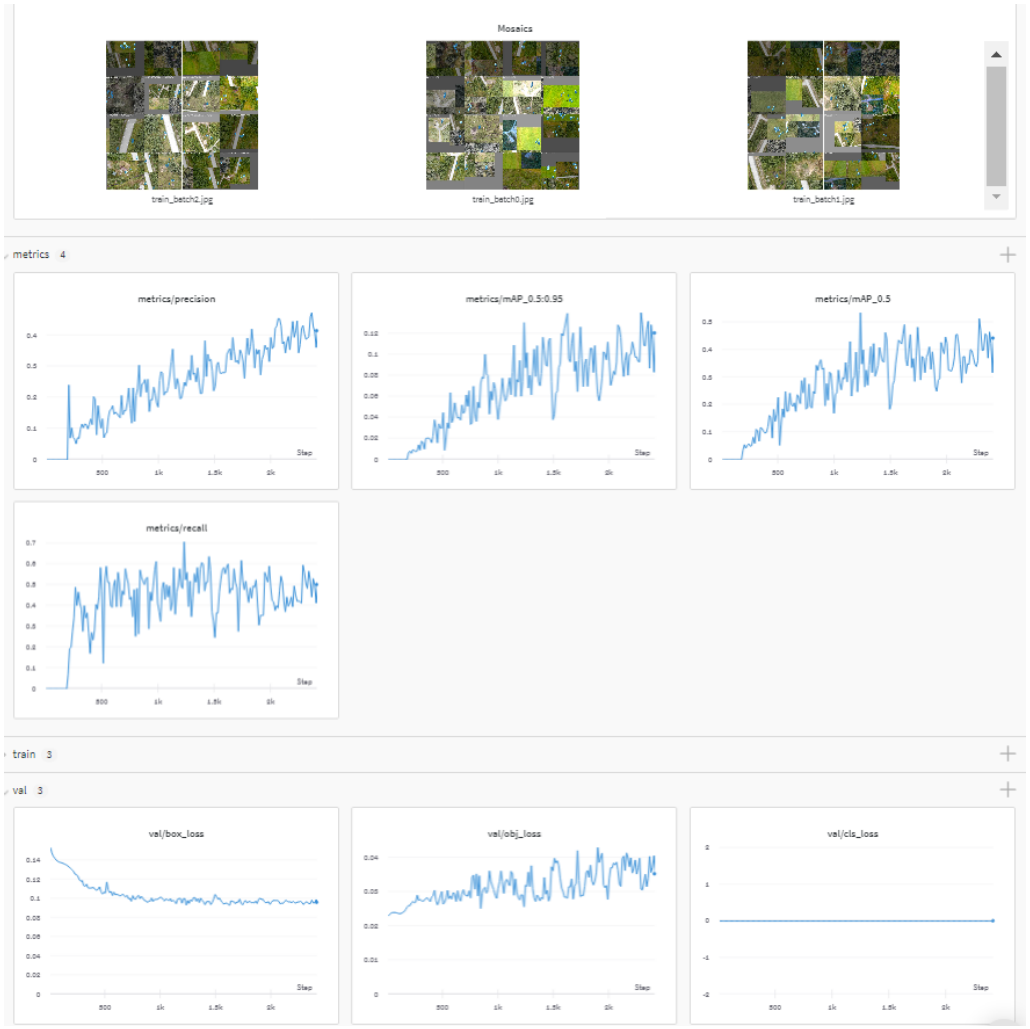


Figure 20: *Screenshot of a wandb-report. The report highlights the model's performance and losses during training.*

4.4 Making the Model Mobile

The constraints of a mobile device compared to a computer requires some adaptations of the deep learning model and how the detection is performed. Handling the reduced availability of memory and processing power is a key factor when dealing with mobile detection models. Due to hardware constraints as described in Section 3.2, the focus is solely on an Android implementation. An illustration of the process can be seen in Figure 21.

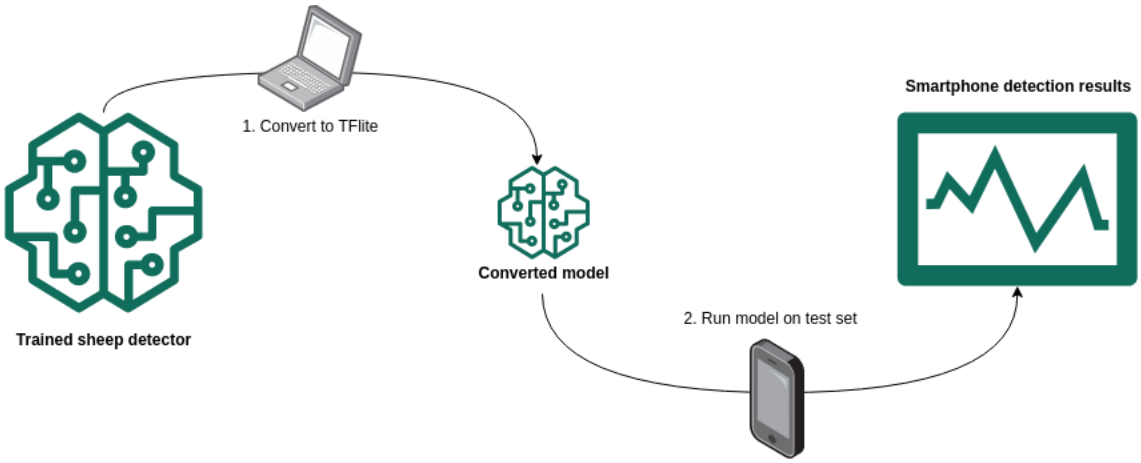


Figure 21: *The process of making a sheep detection model specialized for smartphones.*

Converting Models to TensorFlow Lite

As mentioned, YOLOv5 is natively using the PyTorch framework. To enable faster detection on mobile devices the trained models are converted to TFLite. Conversion to a TFLite model is recommended by the developers of YOLOv5 for detection on Android. In simple terms, a new YOLOv5 TF model is created, then the PyTorch weights of the trained model are transferred. The final TFLite model is exported using built-in methods in the framework. The code for converting the model is currently an addition to the official repository and includes guides and descriptions on how it is done [39]. In addition to conversion, it includes implementation of the model in an Android application for object detection. This implementation is based on examples in the official guides of TFLite [40].

Developing the Smartphone Application

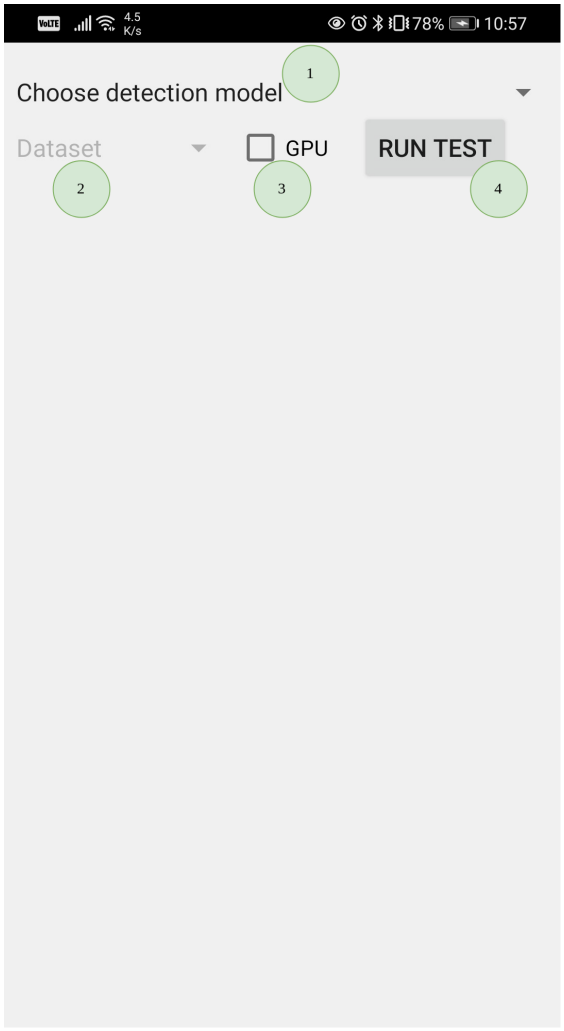
The MVP Android application should primarily enable the testing of different sheep detection models with pre-captured images of sheep. The tests are performed by loading a trained model into a YOLOv5 detector. This detector is then used on the given images and information about the performance is shown to the user. The detection is performed on the same test set of images as described in Section 4.2, to make sure images are independent and equal to the test set used on computer hardware. The key features of the application are highlighted in Figure 22.

To run a test a user has to complete four steps:

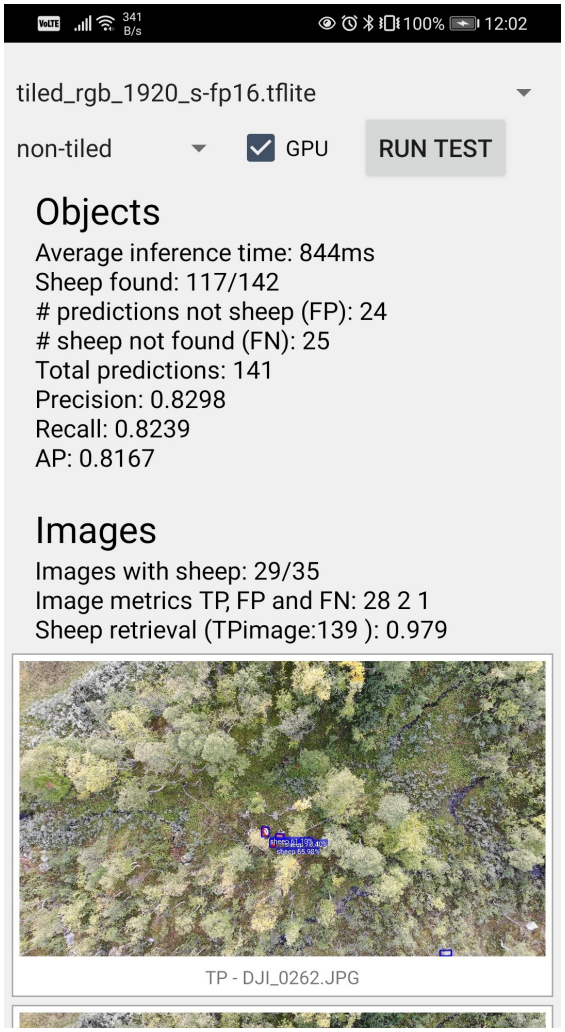
1. Select a sheep detector from all converted models in the dropdown menu
2. **Only available for *tiled* models:** choose inference on tiled or non-tiled images
3. Check box to run inference using the smartphone's GPU (requires the device to have a GPU). The device's CPU will be used if not checked.
4. Run the test

These steps are shown in Figure 22a.

Figure 22b shows key statistics after a complete test run. The most important metrics are inference time (per image), precision, recall and sheep retrieval. In addition to the detection performance metrics, images with their respective ground truths and sheep predictions are included. All model performance results for smartphone presented later in this section have all been tested using this application.



(a) *The steps to select a detection model and other options prior to running an inference test.*



(b) *Detection performance is provided on completion. The test images, including labels and predictions, are shown below the statistics.*

Figure 22: *Screenshots from the application used to test detection models on smartphone.*

4.5 Training Models for Smartphone Performance Evaluation

Initial trials on smartphones showed, not unexpectedly, significantly slower inference times than those of computer hardware. To better evaluate what factors into this inference time, and whether it can be shortened while still maintaining performance, it was decided to train several models covering large parts of the spectrums of *model size* and *image resolution* for both MSX and RGB-images.

MSX-models were trained with downscaled image resolutions of 160p, 320p and 640p. Additionally, upscaled image resolutions from the original 640p to 960p and 1280p were tested. Each resolution was trained with three of YOLOv5's model sizes: s, m and x.

RGB-models were trained with resolutions 320p, 640p, 1024p, 1280p and 4064p with model sizes s and m. We do not believe the smartphone hardware will be able to achieve close to real-time inference time on the 1920p images, let alone the 4064 ones. However, it will still be useful to see whether the predictive performance will be better at these high resolutions, as it will indicate to what degree the smartphone hardware is a bottleneck in realizing mobile real-time detection as a product.

4.6 Experimental Variables

The performance of the models is dependent on several variables. To investigate how the variables affect performance, they are divided into two sets: control variables and independent variables. The control variables are kept constant for all models. The independent variables, on the other hand, are varied between each model. Finally, the models were evaluated based on a set of performance metrics.

4.6.1 Performance Metrics

Precision

Precision, as described in Section 2.4.2, measures the share of correct predictions.

Recall

Recall, also described in Section 2.4.2, measures the portion of sheep detected by the model.

Average Precision

A model's average precision (AP) is found by calculating the area under the curve of the interpolated precision-recall curve, as described in Section 2.4.2.

Sheep Retrieval Measure

As described in Section 3.1, a 100% recall is not needed to actually retrieve every sheep. To better put a numerical value on the portion of retrieved sheep, we propose a *sheep retrieval* measure.

Instead of a sheep having to be detected in order to be retrieved, it is sufficient for any sheep in a given image to indirectly detect them all. The sheep retrieval metric includes these indirectly detected sheep.

Sheep retrieval for a set of images is determined by summarizing all retrieved, divided by the total amount of sheep. Figure 23 shows an example of an image with some detections. In this example the recall and sheep retrieval score would be 0.32 and 1 respectively.

Figure 23: *Image where 2 out of 8 sheep are detected.*

Inference Time

We define inference time as the time in seconds a model needs to return a list of predictions given an image. The inference time only includes computing the prediction(s) for each image.

A low inference time is necessary for the application to be used in real-time as stated in Section (3.1).

Confidence Threshold

While not strictly a performance metric, nor an independent variable, the confidence threshold can be changed to better evaluate a model's performance. Predictions made by the model are accompanied by a confidence value between 0 and 1, which indicates how certain the model is that the prediction is correct.

In general, true positives tend to have a higher confidence than that of false positives. Ideally, this means that by ignoring predictions below a certain confidence threshold, one will raise the model's precision metric. Of course, not every true positive is likely to have a high confidence value, so by increasing the threshold, the recall value may drop.

By default we will test the models with a confidence threshold of 0.01, meaning virtually every prediction will be counted. This may lead to a prohibitively large number of false positives, with regards to deployment in a real scenario. If this is the case, the threshold will be increased to a value that gives a better indication of the models' potential precision and recall.

4.6.2 Independent Variables

Image Type

The primary input types are RGB and MSX-images. Both have different strengths and weaknesses which might cause different results when trying to detect sheep. The standard RGB-images contain the most information due to the highest possible image resolution. In addition, the visual color and texture of sheep will hopefully provide the network with information to make good predictions.

MSX-images contain certain visual features combined with temperature information. The low resolution of MSX images limits the amount of information which might make it challenging to distinguish sheep from other warm objects or noise. Additionally, the inclusion of high contrast visual features may cause too much noise for the model to only detect sheep.

Image Resolution

Intuitively a lower image resolution has fewer details and downscaling of images should cause models to perform worse due to this information loss. The main motivation for reduced resolution is the reduction in inference time. The most interesting factor to investigate is how much the images can be downscaled without causing too large of a reduction in performance metrics.

A series of different resolutions will be tried out to better measure the trade-off between inference time and the other performance metrics. As image types have different native resolutions, the scaling of models is different between the two. MSX-images are resized to the resolutions: 1280, 960, 640, 320 and 160p. This is a range of 25 to 200 % of the original resolution. RGB-models are tested with the following resizes: 4064, 3200, 2560, 1920, 1280, 1024 and 640p. This is a range of 16 to 100 % original resolution.

Tiled vs. Non-tiled Images

In addition to different resolution scales of 4K-images, RGB-models will be trained using tiles of 4K-images as their training and validation sets as described in Section 4.2. This decreases training time and zooms in on objects to hopefully make the sheep detector learn features better.

Model Size

All models are based on one of YOLOv5's pre-trained checkpoints, instead of randomly initialized weights. The size of our available data set makes it unrealistic to train a well-performing model from scratch. The small (s) and medium (m) sized models are the main focus because they have the best potential to perform at a reasonable speed on a mobile device, however, larger models (x) will also be tested. An overview of some pre-trained checkpoints can be seen in Figure 7.

4.6.3 Control Variables

Batch Size

The default batch size of 16 was used for training all models except *rgb_4064_s*. Due to memory and time constraints, this model was trained with a batch size of 4.

Number of Training Epochs

In the early stages, epochs ranging from 50 to 500 were tried out. The first 100 epochs sported the most impactful change in the models' performance, although in most cases performance slowly increased for the entirety of the 500 epochs without seeming to overfit. 300 epochs was deemed to be a suitable middle ground between training time and representative results, and all models presented in Section 5 were trained with 300 epochs.

Other Hyperparameters

Models use the default hyperparameters as provided by the YOLOv5 developers. It is possible to perform a evolution of these parameters but this was not seen as impactful enough to be worth spending effort on.

4.7 Summary

Data collection will be carried out over two days in Storlidalen, with an emphasis on creating a distinct test set and capturing MSX images. A DJI Mavic 2 Enterprise Dual drone will be used for this purpose (3.2).

Collected data will be filtered and processed in preparation for training the deep learning models.

The models will be trained using the YOLOv5 architecture. Training hardware is provided by the IDUN HPC cluster at NTNU. Several models will be trained using different pre-trained YOLOv5 checkpoints and image resolutions.

Models will then be converted to a less memory-intensive format – TFLite – using the PyTorch framework and tested using a simple Android application.

5 Results

This section presents the results of the work done; the data that was collected and preprocessed, the models that were trained and their performance running detection on computer hardware as well as the converted models tested on smartphone hardware.

5.1 Data Collection

After filtering the existing images, the data set was reduced in size. Selecting images with RGB-MSX-pairs and of suitable elevation led to a total of 321 images, primarily from the August 2019 sessions in Storlidalen.

The field trip to Storlidalen in September 2020 resulted in images of both fenced and free-ranging sheep. 166 image pairs were captured using the M2ED drone – 95 of which were later deemed ideal for further use. Most of the images were captured in fenced areas. Images in these environments often contain larger herds of sheep. Additionally, sheep tend to be more visible and easier to spot, as the fields are often clear of other background elements and distractions like vegetation and rocks. An example of sheep in a fenced environment can be seen in Figure 24a.

The free-ranging sessions focused on capturing images for a more realistic test set than existed earlier. A set of images were captured along the river Sandåa in an attempt to create a consecutive image series of challenging terrain and sheep. These images make up the majority of the test set and an example of sheep in this environment can be seen in Figure 24b. Images of free-ranging environments are often more diverse and have more distractions than the fenced images. This results in images where sheep are harder to spot, even for a human.

(a) *Fenced environment* **(b)** *Free ranging environment*

Figure 24: *Sample image-cuts showing sheep in different grazing environments. Images were captured in Storlidalen September 2020.*

5.2 Data Preprocessing

Existing data annotations were converted to the YOLO format. To do this, a script to transform Pascal VOC labels to YOLO labels was created. This script can be seen in Appendix B.1.

Data Sampling

The final distribution of training, validation and test images is shown in Table 6. The columns show where and when images were captured. Note that the numbers indicate RGB-IR pairs of images. Sample images from the different data sets can be found in appendixes A.1, A.2 and A.3.

Data set	Storlidalen - Aug. 2019	Storlidalen - Sep. 2020	Data set total
Training	307	27	334
Validation	14	29	43
Test	0	35	35

Table 6: *Distribution of images used across data sets.*

In order to test the models' ability to generalize over a wide range of inputs, we ensured that the test set was significantly different from the other images. This was done by making sure that images in the test set were geographically distinct from the other sets.

Tiling Images

The tiling of images as described in Section 4.2 resulted in the distribution of images shown in Table 7.

Data set	Total sheep	Images containing at least one sheep	Total images	Percentage containing sheep
Training	1585	877	1002	88 %
Validation	176	74	1302	6 %
Test	159	77	980	8 %

Table 7: *Distribution of tiled images across data sets.*

5.3 Model Performances

MSX-models

This section covers the performance of the MSX sheep detection models. Firstly, performance on computer hardware, secondly, converted models on smartphone hardware.

The average precision of the models can be observed in Figure 25. As described in Section 4.6.2, 640p is the native resolution of MSX-images. At this resolution, there is little to separate the three model sizes with the s-model scoring 0.654, the m-model 0.624 and the x-model 0.682. The 960p resolution models notably sport a wider spread between the model sizes. A spread that continues at the 1280p resolution, where the smaller models drop, while the largest model retains its performance. A complete overview of the test results can be reviewed in Appendix C.

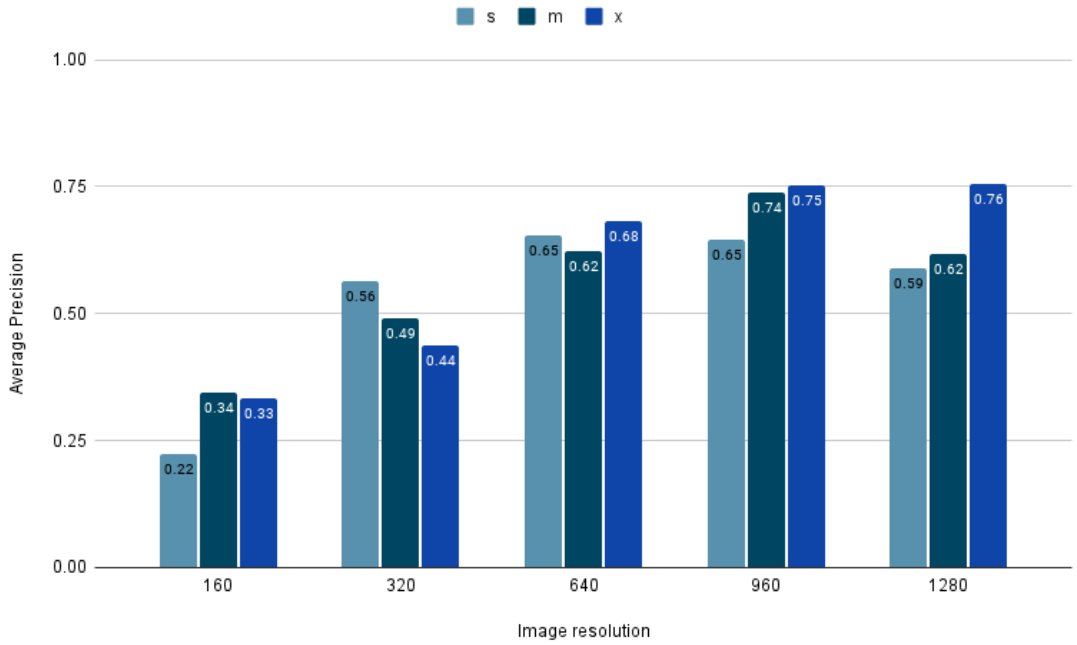
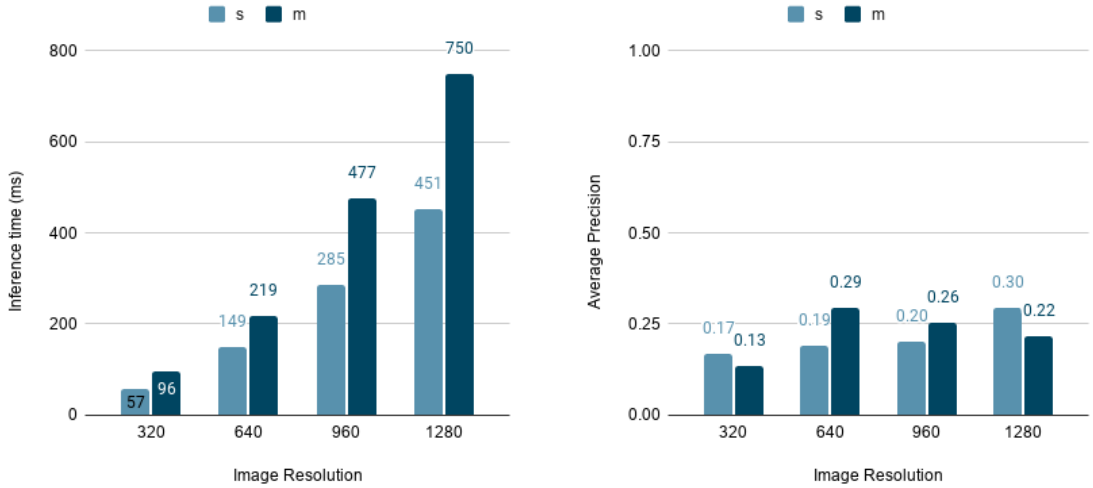


Figure 25: *Average precision for MSX-models grouped by image resolution and model size. The results are from the test data set using computer hardware.*

The x-sized models are the largest and consequently have the longest inference time. Further converting and testing on mobile devices will not be conducted with the x-sized models. They did not show significant improvement to the detection performance to warrant the longer inference time compared to smaller model sizes.

Figure 26 shows inference time and AP when running tests using the converted models on a smartphone. The inference time shows, as expected, a time increase in tandem with both image resolution and larger model size. Figure 26b shows a rather large drop in AP compared to what the corresponding, non-converted, models' performance on computer hardware in Figure 25. All models have relatively high precision, but many sheep remain undetected.

The best model judging by these results is the *msx_m_640*, i.e the m-sized model trained on image resolution 640p. It sports the highest AP at 0.29 with a low inference time of just 219ms.



(a) *Inference time* **(b)** *Average Precision*

Figure 26: *Inference time and average precision of MSX models grouped by image resolution and model size. The results are from the test data set using smartphone hardware.*

RGB-models

Figure 27 shows an overview of the models using RGB images and their average precision.

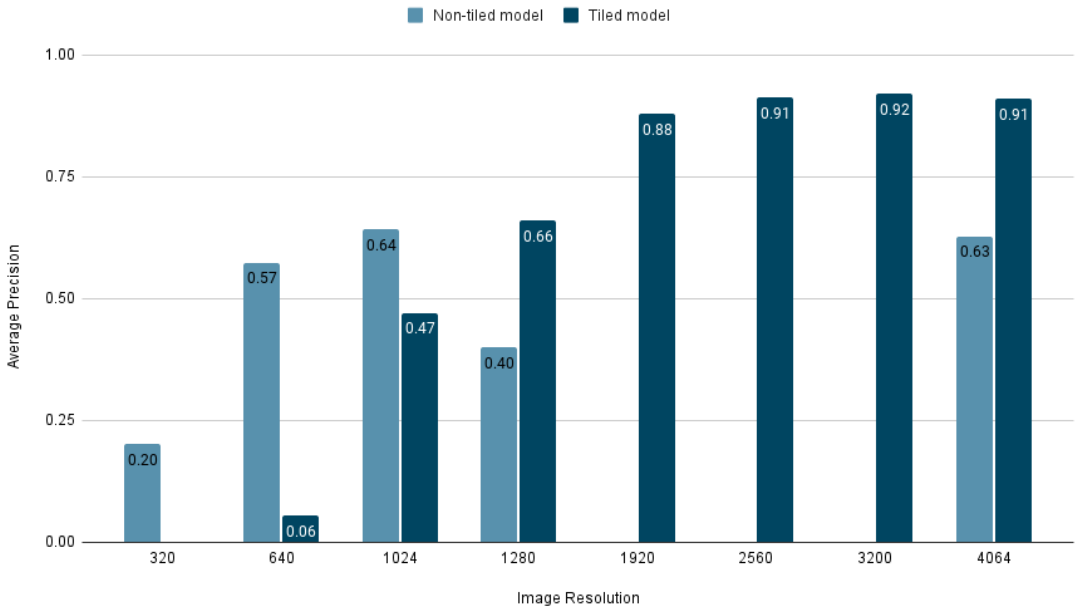


Figure 27: *Average precision for all s-sized RGB-models grouped by image resolution and whether they are trained using tiled or downscaled images.*

It quickly became apparent that training models with tiled 4K-images is the way to go, as opposed to training using downscaled images. As can be seen in Figure 27, the tiled model outperforms the non-tiled models significantly on resolutions above 1024p.

The tiled model is trained on tiled 4K-images, but as shown in Figure 27 there is only a minor AP-loss by downscaling before going below 1920p. Below this point, the differences between the tiled model and the specialized lower resolution models begin to even out. From 1024p and below

the tiled model is incapable of making good predictions.

Despite the relatively good results on lower resolutions, the non-tiled models do not compete with the tiled model. As we shall see shortly, this holds true even when performance on mobile devices is considered. Because of this, the following results will mainly focus on the tiled models. The complete test results for both tiled and non-tiled models can be seen in Appendix C.

Tiled models were trained in both model size *s* and *m*. Running inference on resolutions 1920p and up, the two model sizes are nigh on inseparable, with the larger model *m* winning out on lower resolutions, as seen in Figure 28.

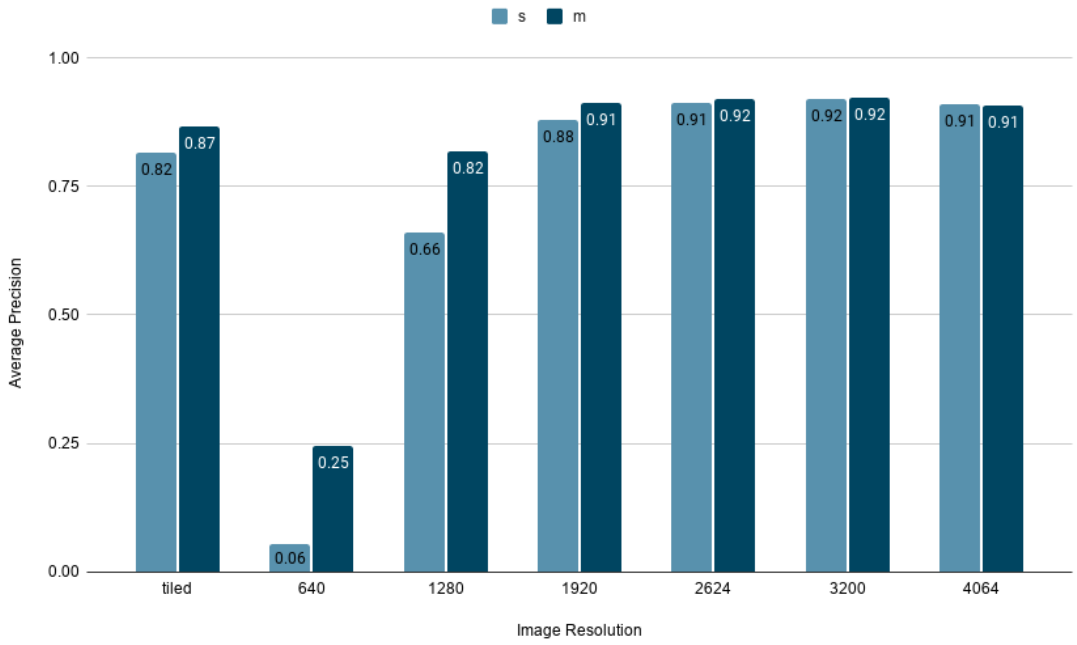
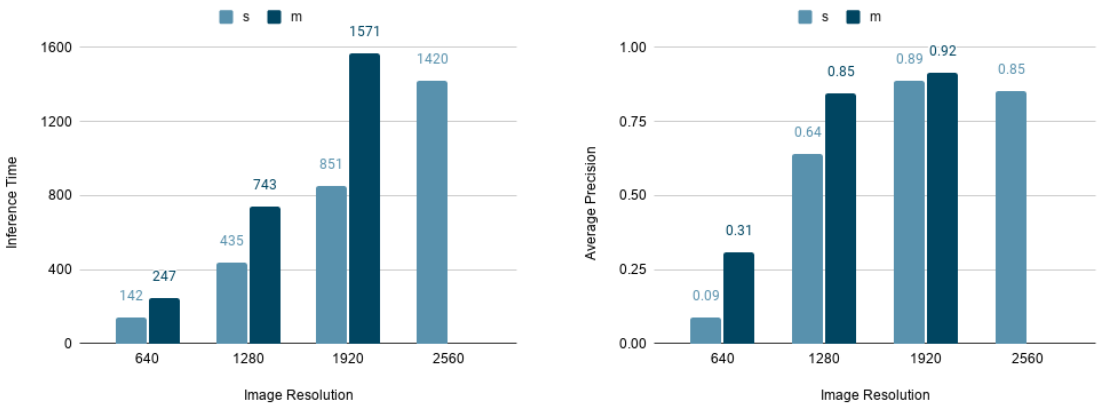


Figure 28: *Average precision grouped by image resolution and model size. All models are trained using tiled images and AP is based on test set results. The* tiled *bar represents AP for models on the tiled test set while the rest are complete images resized to the given resolution.*

While there is some increase in predictive performance, the inference time increases dramatically with the increase in model size. A comparison of the models' performance on mobile devices can be seen in Figure 29. Note that due to memory constraints on the device used, the highest possible m-sized resolution is 1920p. Similarly, the highest s-sized resolution is 2560p.

Recall that the project requirements state that inference time should be below 1 second (Section 3.1). Given this requirement, we evaluate the best performing model to be the s-model running inference on images downscaled to 1920p. The s-model scores higher AP than its m-model competitor at 1280p, with an increase of 100ms in inference time, for a total of 851ms.



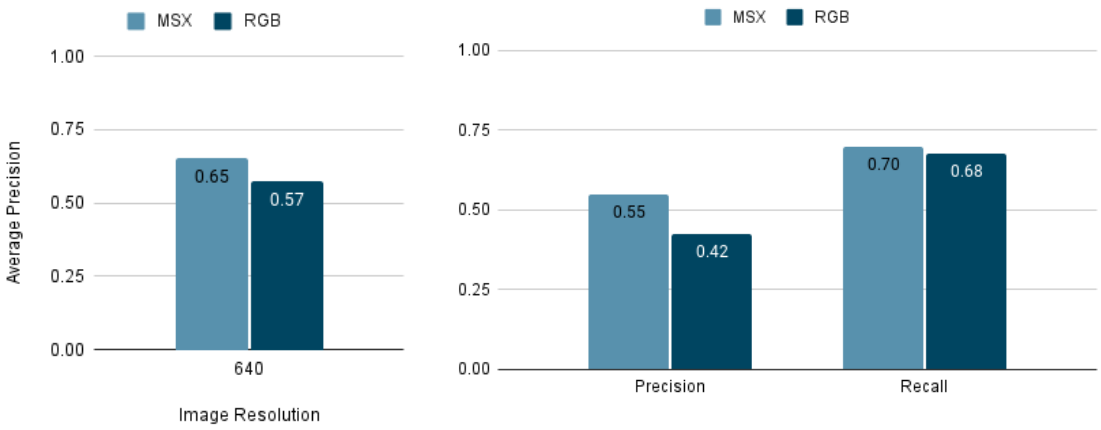
(a) *Inference time*

(b) *Average Precision*

Figure 29: *Inference time and average precision of RGB-models grouped by image resolution and model size.*

Comparison of Corresponding MSX and RGB-Models

The difference in detection performance with corresponding models trained on MSX and RGB-images can be seen in Figure 30. At its native resolution of 640p, the MSX-model outperforms the RGB-model in both AP, precision and recall.



(a) *Average Precision*

(b) *Precision and recall obtained with a confidence threshold of* 0.5.

Figure 30: *Detection performance of corresponding models trained and tested using MSX and RGB images. Both are s-sized models and tested on computer hardware.*

5.4 Impact of Converting to Mobile-friendly Format

Figure 31 shows the AP results after training (validation), tests on computer hardware and tests on smartphone hardware for the best MSX and RGB-models.

During training, the models' AP is relatively similar, but when tested on unseen images, the MSX-model struggles, especially when converted to a mobile-friendly format. The tiled RGB-model actually shows the opposite, with a slight gain in AP when on a smartphone.

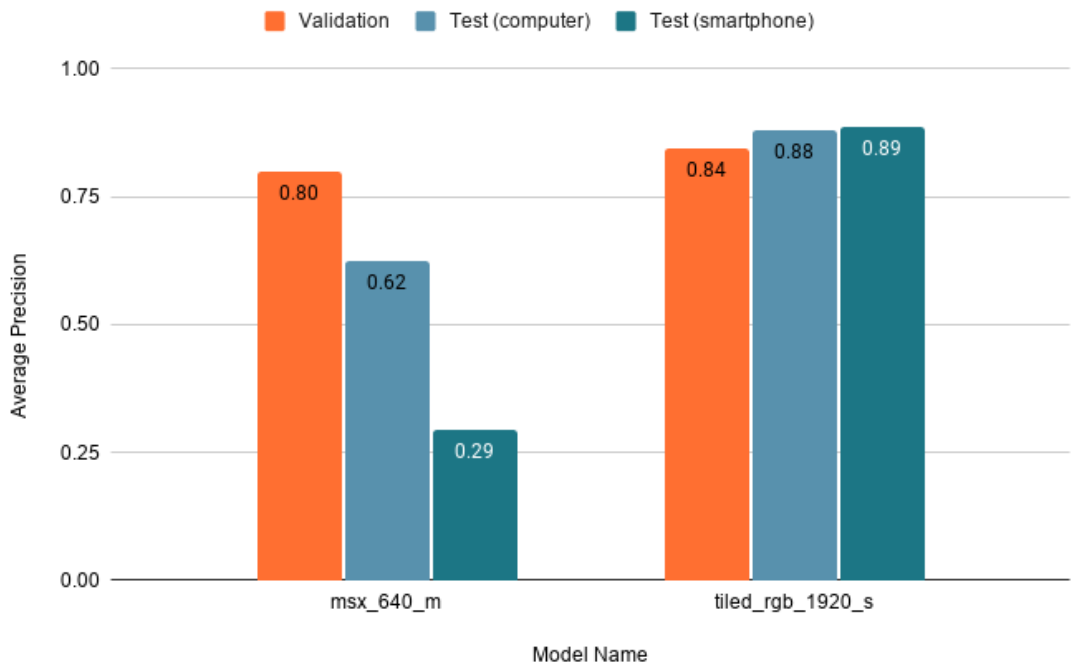
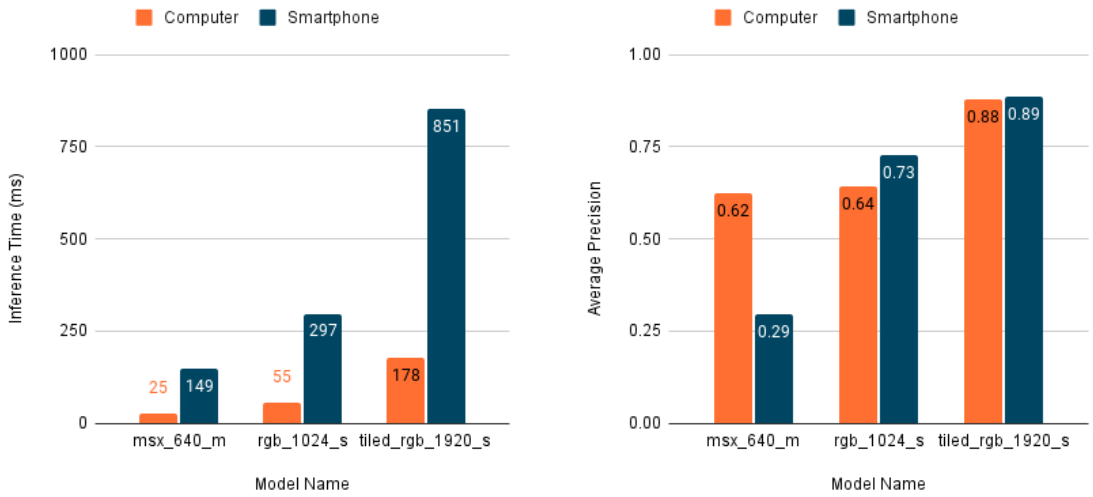


Figure 31: *Average precision performance for MSX and RGB models on the validation and test data sets.*

Performance on Smartphone Versus Computer

Figure 32 shows selected models' inference time and AP grouped by image resolution and type of hardware. There is a distinct increase in inference time of the converted models on a smartphone. The AP measurements as shown in Figure 32b has no obvious trend when comparing computer to smartphone hardware. The MSX-model shows a significant drop in AP using the converted model while the RGB-models perform at the same level – even slightly better.



(a) *Inference time* **(b)** *Average Precision*

Figure 32: *The difference in inference time and average precision grouped by detection model and hardware devices.*

Sheep Retrieval

Sheep retrieval, as described in Section 4.6.1, is an alternative metric to evaluate the detection of sheep. By this measure, a sheep is detected if it is present in an image where at least one other sheep is detected.

Figure 33 shows the precision, recall and sheep retrieval scores for some models, including the best performer *tiled_rgb_1920_s*. Note that the model *rgb_1024_s* is actually able to «retrieve» slightly more sheep than the tiled model, despite its lower precision and recall.

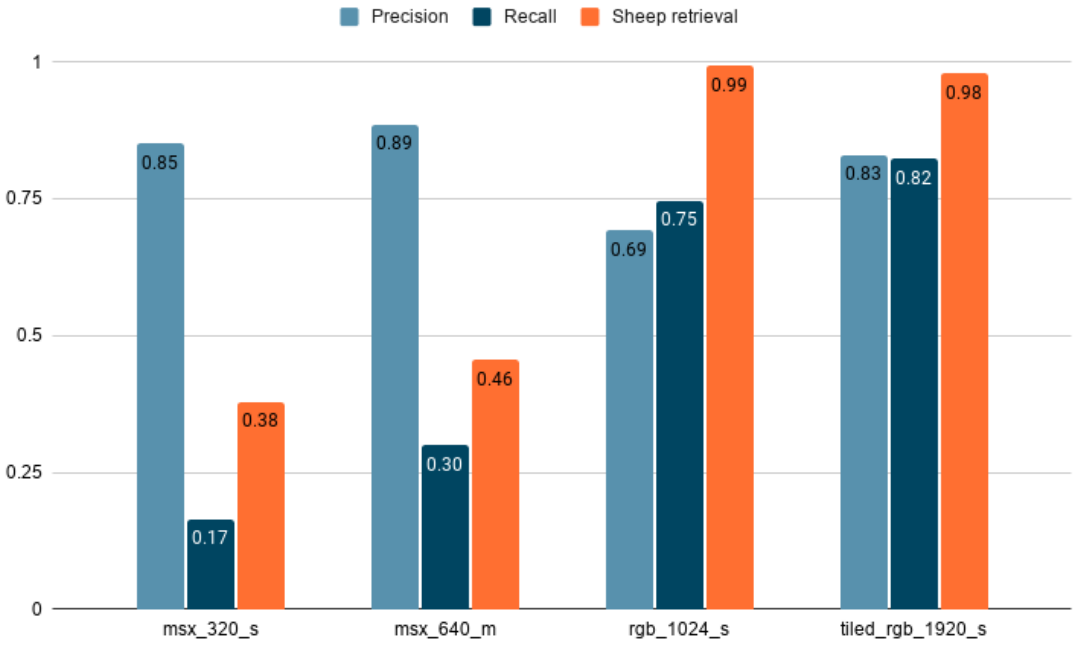


Figure 33: *Precision, recall and sheep retrieval for four different models.*

6 Discussion

This section discusses the results; why we ended up with these them, what do they imply, and what could have been done better.

6.1 The Data Set

Most state-of-the-art models present their performance based on having trained on thousands, if not tens of thousands of images – an unrealistic amount for a thesis such as this. YOLOv5's built-in image augmentation alleviates some of this problem by slightly changing and tweaking the images during training. Still, it is difficult to judge the potential of models trained on data sets, say, a hundred times bigger. Surely, there would be at least *some* increase in performance, and an expansion of the data set should certainly be a focus for potential future work.

Despite its relatively small size, we believe the test set serves its purpose well and the images therein to be of high quality. The areas in which the images are captured do not overlap with either the training or validation set images, and it properly tests the models' ability to detect sheep in new environments. Many of the images in the test set contain heavy vegetation and some darker sheep. When annotating these images we ourselves had problems identifying every sheep and separating them from small white details in the images. The fact that the models are able to retrieve almost every sheep from this difficult set of images speaks loudly of the potential for use in the field.

The biggest downside with the test set is the fact that almost every image contains a sheep. As such, it does not properly measure the effect false positives would have in a real-life scenario. To this end, a large, realistic test set should be created, possibly simulating a video stream in which only a small portion of the images actually contain sheep. To some extent, this happens when testing the tiled models on tiled images, where we see the precision score drops significantly.

6.2 The MSX-models

The MSX-models sport high precision values across the board, beating its RGB equivalents significantly and competing with the tiled models. Their recall and sheep retrieval measures are lacking, with values between 20 and 40% of those of the RGB-models – showing an inability to detect sheep without a distinct outline. This is contrary to the motivation behind using MSX-imaging, as they should, in theory, be more able to detect darker or partially obscured sheep much better, thanks to their thermal signature. In practice, though, this did not end up being the case – mostly as a result of the quality (or, rather, lack thereof) of the MSX-images.

The Difficulties of Thermal Imaging

When the images were to be captured we had relatively little experience using the thermal imaging hardware and software, which likely prevented us from extracting all its potential. When practicing tweaking the camera, we used people on a soccer field on a warm and sunny day as «thermal hotspots». The settings for detecting human heat signatures under those conditions turned out to be a lot easier to find than those for detecting small, woolly, partially obscured sheep in a hillside where elevation and temperatures can change quickly.

Ideally, we would have spotted sheep in the field, made on-the-fly adjustments to the camera settings for the given weather conditions to find the optimal thresholds to strengthen the sheep's heat signature.

With limited time in the field, we decided that many not-great images would have to suffice, lest we spend the entire day tweaking the camera. This resulted in the MSX-images as shown in this thesis. While they are able to show some of the strength of this technology – namely the outlining

of distinct objects against their background – thermal signatures are virtually non-existing.

When comparing to models with similar inference time, things are less bleak. No model of the same speed comes close to the same precision level, meaning the model is relatively noise-resistant for its speed. This makes us inclined to believe that given images with more distinct thermal signatures (red sheep against a blue environment), could be able to outperform even the tiled RGB-models.

Its focus on outline against background seems to lead to fewer false positives in areas of heavy foliage. Johannessen showed that a synergy effect between RGB and thermal images could be achieved [4]. Our results support this, and we consider it a promising lead for further research.

Aside from combining the models with an RGB-model, we believe the most fruitful work to improve results from thermal images would be to perform an in-depth study of thermal image settings in the field to discover which thresholds to be used to best pick out sheep under different weather conditions.

6.3 RGB-models

6.3.1 Regular Models

The regular RGB-models are the middle of the bunch. Their AP, precision and recall performance peaks around the 1024p resolution before dropping rapidly. It is somewhat surprising that down-scaling the images from 4056p down to 1024p would yield the best result, especially considering the tiled model results indicate that higher resolution is better.

Impact of Image Resolution and Model Size

Interestingly, simply increasing image resolution does not necessarily yield better performance, especially when one considers the additional inference time. The *rgb_4064_s*-model performs very similarly to the *rgb_1024_s*-model despite a 14 times increase in inference time. The convolutional scaling method presented by the EfficientNet paper [41], suggests that when increasing image resolution, one should also upscale the model width and depth accordingly to better exploit the increased details of higher resolution images. This was tried out during experimental phases as well as our specialization project, but the larger models did not demonstrate better performance on full resolution images – often performing worse! This can also be seen by comparing the results of the *rgb_1024_s*, *rgb_1280_s*, and the *rgb_1280_s6* models, the latter being based on a larger checkpoint, pre-trained on 1280p images.

Model	AP@0.5	Precision	Recall	Sheep Retrieval	Inference Time (ms)
rgb_1024_s	**0.695**	**0.693**	**0.747**	**0.97**	**297**
rgb_1280_s	0.497	0.624	0.620	0.83	447
rgb_1280_s6	0.510	0.525	0.585	0.86	435

Table 8: *Mobile performance of* rgb_1280_s *and* rgb_1280_s6

As we can see in Table 8, the model trained on the lowest resolution images clearly wins out. Increasing the model size to accommodate the higher resolution images only has negligible effects.

It is difficult to say why performance seems to peak at 1024p for the non-tiled models, while it increases with resolution in the tiled models. This is discussed further below.

6.3.2 Tiled Models

It is clear that tiling the 4K-images before using them for training yields improved results, but why? The images are not altered in any way except being split into several parts, and yet, the

models trained on tiled 4k-images perform better than their counterparts even on non-tiled test images.

The Effectiveness of Tiling

The immediate practical effects of tiling images are described in Section 4.2.

As expected, a trade-off when using tiled models was the increased time spent during inference, when several smaller tiles have to be processed individually, rather than a single large frame. Observing both the *s* and *m*-model, the total inference time when processing 28 smaller individual tiles is roughly 30% higher than that of processing a single complete image.

What we did not expect, however, was the tiled model's performance on non-tiled images. Related literature [22][23] suggests images also need to be tiled during inference, but our results demonstrate otherwise. In fact, the model's performance running inference on tiled images seems to be *worse* than running on complete images – let's take a closer look at the *tiled_640_s*-model's performance on tiled images compared to complete images downscaled to 1920p.

As we can see in Figure 34, in both cases the model is able to spot every sheep in the area, and in both cases, the model mistakenly predicts a stone in the lower left area to be a sheep. The correct predictions are mostly within the 0.7–0.8 confidence range.

The major difference lies in the false positives in the tiled images. In the 1920p-image, the model only predicts one sheep among the large white rocks, with a relatively low confidence of 0.3, meaning the prediction could easily be removed by increasing the confidence threshold to a more suitable level. On the tiled image, however, more wrong predictions are made with higher confidence levels.

By looking at Table 9, it is clear that this is not restricted to just one example.

Image Type	AP@0.5	Precision	Recall	Inference Time (ms)
1920p	**0.817**	**0.830**	**0.824**	**851**
Tiled 4K	0.423	0.157	0.757	225(x28)

Table 9: *Mobile performance of* tiled_640_s *on downscaled 1920p and tiled 4K-images.*

When running inference on tiled images, the model is clearly more prone to predicting false positives.

Or – the Weakness of the Data Set

It is possible that the comparatively good scores of the tiled models are not just a result of the effectiveness of tiling, but also the weakness of the training and validation sets the non-tiled models use. As described in Section 4.2, the original sets does not provide the model with as many instances to learn from.

During training, the tiled models are validated on 1300 images, only 6% of which containing sheep – it is likely this gives the tiled model an edge in separating sheep-ish terrain features compared to their non-tiled competitors. Perhaps, if the non-tiled data sets contained a similar quantity of unique instances, their models' performance would also improve.

6.4 The Impact of Running on Mobile Hardware

As expected, the use of smartphone hardware leads to higher inference time compared to computer hardware. Ideally, inference time should be the only performance difference when running on a mobile device while the number of detected sheep stays the same.

(a) *Complete 1920p-image.*

(b) *Tiled 4K-image. Note that the model made predictions on each tile individually. This image was stitched together after the test was run.*

Figure 34: *Predictions made by the* tiled_640_s-*model on a complete image downscaled to 1920p vs a tiled 4K-image*

As we have demonstrated, the performance of converted models does differ from the original YOLOv5 models – the MSX-models in particular.

While the MSX-models suffer from conversion to the mobile format, the best RGB-models see a minor increase in AP performance. This is surprising, considering the model quantization of the converted models leads to less accurate parameters using 16-bit floating-point numbers instead of 32-bit. Judging by this, low-resolution images with relatively little information suffers the most when the model is reduced in this manner, while the models using high-resolution images are relatively unaffected.

7 Conclusion and Future Work

7.1 Conclusion

In this thesis, we have proposed several deep learning models able to detect sheep in UAV images, real-time, while running on a mobile application. In an effort to find the best performing models, model development was done using different image types, image resolution, architecture sizes and pre-processing methods with the YOLOv5 architecture. The models were evaluated by average precision, sheep retrieval, precision, recall and inference time on a test set of independent images. This gives a solid measure of the models' ability to generalize and adapt to unseen environments. An Android application has been developed as an environment for the models to run in – demonstrating the models are able to perform sheep detection in real-time on smartphone hardware.

The proposed sheep retrieval metric provides a solid measurement of a model's performance at a glance by counting sheep as retrieved if another sheep in their image is correctly identified.

Our findings show that tiling the high-resolution RGB-images during training greatly boosts the predictive performance of models, even if the images used during inference are not tiled themselves.

Earlier work has shown that a fusion of models using both visual and infrared images as input yields improved results compared to using them separately. The use of MSX-images did not end up showing this same improvement – mostly due to the lack of clear thermal signatures in the captured images. Nonetheless, the high precision of the MSX-models shows that this type of imagery is resistant to false positives. If the images were of higher quality, we believe MSX-based models would compete with RGB-models.

The best performing model – *tiled_s* – has proven to be a robust model. The model's predictive power is retained even after converting to a mobile format. The inference time is higher at 851ms, however, this is still within the realms of what we consider to be real-time.

All things considered, the results presented in this thesis show that deep learning object detection models are very capable of detecting sheep in drone images both quickly and accurately – even when running on budget smartphone hardware. Such a model has the potential to be integrated with a mobile application as part of a fully-fledged commercial solution, and could be of great help to farmers who would rather let technology handle the tiresome sheep retrieval process.

7.2 Future Work

An expansion of the data set would be worthwhile. Both to provide more training and validation data, but also to create a large, realistic test set, to even better evaluate the FP-rate in a real scenario.

Should work on this field continue, we believe properly utilizing and optimizing the sheep retrieval metric alongside precision should be the main focus. In this thesis, the metric counts a sheep as retrieved if any other sheep in its image is correctly detected. This was done due to the relatively small size of the test set. Realistically a sheep should be counted as retrieved if any sheep in its herd is correctly detected in any image.

It is possible utilizing a dedicated mobile CNN would yield better results than converting a heavier model to a lighter format. Architectures such as PeeleNet sport good results, with a significantly reduced size [42].

Last, but not least: To fully realize the potential of MSX-images, an in-depth study should be carried out, examining the optimal thresholds for maximum and minimum temperature settings of the thermal camera under different weather conditions. Properly defined thresholds are necessary for the thermal signatures of sheep to be properly visible in the MSX-images. We believe that if this potential is realized, models trained on MSX-images will be able to outperform RGB-models with regards to both predictive power and inference time.

References

[1] Dyrebeskyttelsen Norge. *Tap av sau på beite*. URL: https://www.dyrebeskyttelsen.no/tap-sau-pa-beite/.

[2] Mattilsynet. *Årsrapport 2019*. 2020. URL: https://www.mattilsynet.no/om_mattilsynet/aarsrapport_2019__mattilsynet.38708/binary/%C3%85rsrapport%202019%20-%20Mattilsynet.

[3] Anne-Cath. Grimstad. *Tilsyn med drone: Rimelig og effektivt*. 2017. URL: https://www.fag.nsg.no/artikkel_vedlegg_serve.cfm?artikkel_id=327.

[4] Kari Meling Johannessen. 'Towards Improved Sheep Roundup'. Using Deep Learning-Based Detection on Multi-Channel RGB and Infrared UAV Imagery. MA thesis. NTNU, July 2020.

[5] Anders Ottersland Granås and Ole Kildehaug Furseth. 'Real Time Detection of Sheep in Drone Images'. Specialization project. Dec. 2020.

[6] Kari Meling Johannessen. 'Detecting Sheep in Drone Images by Deep Learning'. Specialization project. Dec. 2019.

[7] Svein-Olaf Hvasshovd. *HerdIT Elektronisk oppfølging av sau på beite*. 2017. URL: https://www.fylkesmannen.no/contentassets/cbf122460efa4e37a051c17c07fade0d/droner-buskerud-2017.pdf.

[8] Nina Kristiansen. *Hvor plagsomt er det for sauen å gå med ei bjelle rundt halsen hele sommeren*. July 2019. URL: https://forskning.no/dyreverden-husdyr-landbruk/hvor-plagsomt-er-det-for-sauen-a-ga-med-ei-bjelle-rundt-halsen-hele-sommeren/1360767.

[9] Findmy. *Findmy | GPS sporing av husdyr på utmarksbeite - uten mobildekning*. URL: https://www.findmy.no/.

[10] Telespor. *Telespor AS – Produkter og tjenester for elektronisk overvåkning av husdyr på beite*. URL: http://telespor.no/.

[11] Telespor. *Hjem - Smartbjella Sporing*. URL: http://smartbjella.no/.

[12] A. Wiśniewski M. Mazur and J. McMillan et.al. 'Clarity from above'. In: (2016).

[13] EU council. *Drones: reform of EU aviation safety*. URL: https://www.consilium.europa.eu/en/policies/drones/.

[14] DJM Aerial Solutions. *Drone Services*. URL: https://djm-aerial.com/drone-services/.

[15] Insider Intelligence. *Drone market outlook in 2021: industry growth trends, market stats and forecast*. URL: https://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts?r=US&IR=T.

[16] Paperswithcode. *Object detection on COCO test-dev dataset*. Jan. 2021. URL: https://paperswithcode.com/sota/object-detection-on-coco.

[17] T. Y Lin et al. 'Microsoft COCO: Common Objects in Context'. In: (2015).

[18] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[19] Glenn Jocher et al. *ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration*. Version v4.0. Jan. 2021. DOI: 10.5281/zenodo.4418161. URL: https://doi.org/10.5281/zenodo.4418161.

[20] Ultralytics. *ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite*. URL: https://github.com/ultralytics/yolov5.

[21] Jakob Solawetz | Roboflow. *Tackling the Small Object Problem in Object Detection*. Aug. 2020. URL: https://towardsdatascience.com/tackling-the-small-object-problem-in-object-detection-6e1c9976ee69.

[22] B Özkalaycı F. Özge Ünel and C. Çığla. 'The Power of Tiling for Small Object Detection'. In: (2019).

[23] Jacob Solawetz. *Tackling the Small Object Problem in Object Detection*. Aug. 2020. URL: https://blog.roboflow.com/detect-small-objects/.

[24] Heartbeat - Fritz AI Jameson Toole. *Deep learning has a size problem*. URL: https://heartbeat.fritz.ai/deep-learning-has-a-size-problem-ea601304cd8.

[25] TensorFlow. *TensorFlow Lite | ML for Mobile and Edge Devices*. URL: https://www.tensorflow.org/lite.

[26] TensorFlow. *Model Optimization | TensorFlow Lite*. Jan. 2021. URL: https://www.tensorflow.org/lite/performance/model_optimization#quantization.

[27] TensorFlow. *TensorFlow Lite GPU delegation*. Jan. 2021. URL: https://www.tensorflow.org/lite/performance/gpu.

[28] FLIR. *Multi-Spectral Dynamic Imaging | FLIR Systems*. 2020. URL: https://www.flir.eu/instruments/multi-spectral-dynamic-imaging/.

[29] FLIR. *flir-msx-tech-note.pdf*. 2019. URL: https://www.flir.eu/globalassets/instruments/flir-msx-tech-note.pdf.

[30] NTNU. *Svein-Olaf Hvasshovd - NTNU*. URL: https://www.ntnu.no/ansatte/sophus.

[31] DJI. *Mavic 2 Enterprise Series*. Apr. 2021. URL: https://www.dji.com/no/mavic-2-enterprise.

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

[33] Nvidia. *NVIDIA V100 | NVIDIA*. URL: https://www.nvidia.com/en-us/data-center/v100/.

[34] SchedMD. *Slurm Workload Manager - Documentation*. URL: https://slurm.schedmd.com/.

[35] Huawai. *HUAWAI P30 Pro Specifications*. 2021. URL: https://consumer.huawei.com/en/phones/p30-pro/specs/.

[36] *Preparing your training data | Cloud AutoML Vision Documentation*. URL: https://cloud.google.com/vision/automl/object-detection/docs/prepare#manual_and_automatic_dataset_splits.

[37] Piotr Skalski. *Make Sense*. 2019. URL: https://github.com/SkalskiP/make-sense/.

[38] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[39] zldrobit. *Add TensorFlow and TFLite export*. URL: https://github.com/ultralytics/yolov5/pull/1127.

[40] TensorFlow. *Object detection | TensorFlow Lite*. URL: https://www.tensorflow.org/lite/examples/object_detection/overview.

[41] M. Tan and Q. V. Le. 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks'. In: (2020).

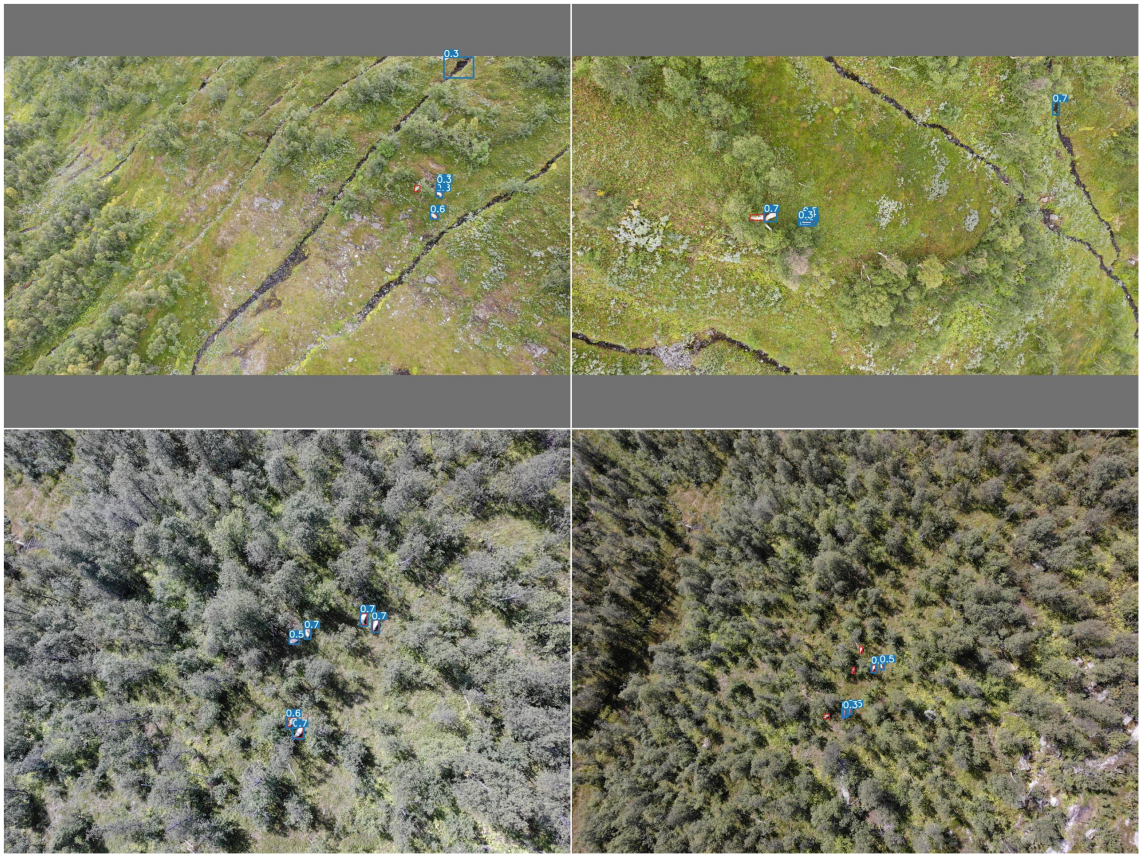
[42] C. X Ling R. J. Wang X. Li. 'Peele: A Real-Time Object Detection System on Mobile Devices'. In: (2019).

Appendix

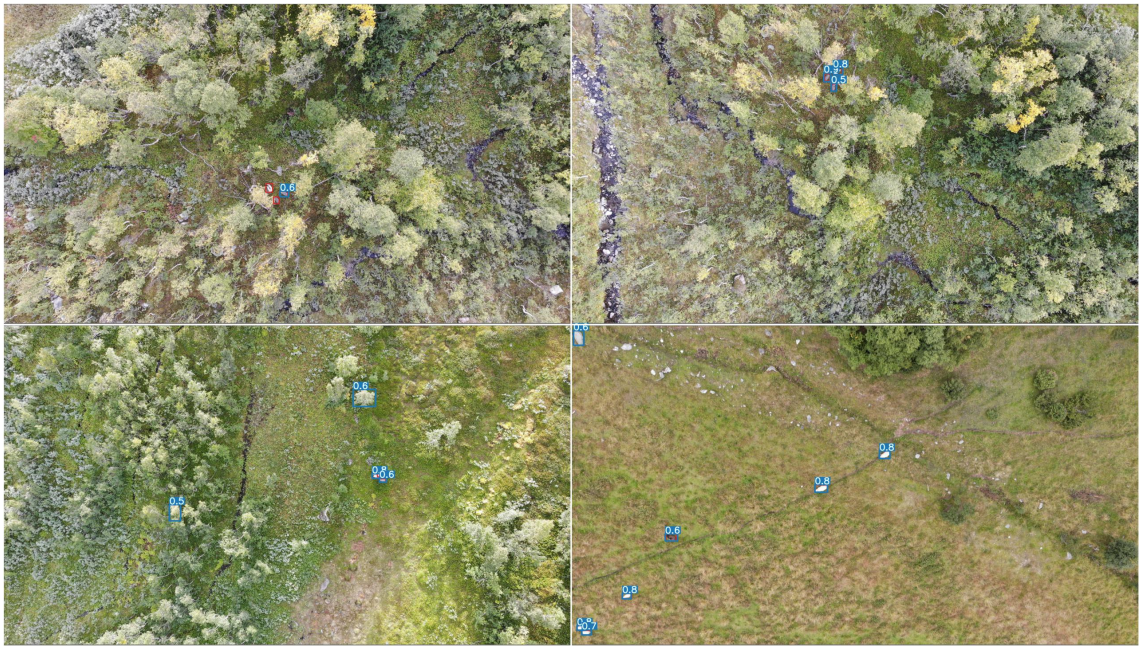
A Data Set Sample Images

A.1 Sample of Labelled Images in Training Data Set

A.2 Sample of Images and Prediction on the Validation Data Set



A.3 Sample of Images and Prediction on the Test Data Set



B Python code scripts

B.1 Converting Pascal VOC to YOLO-format

```
def format_geometry_to_yolo(geometry):
    IMG_WIDTH = 640  # Change according to image sizes
    IMG_HEIGHT = 480
    min_x = geometry[0][0]
    min_y = geometry[0][1]
    max_x = geometry[1][0]
    max_y = geometry[1][1]

    obj_class = 0 # Only used single class (sheep)
    x_center = (min_x + max_x) / (2 * IMG_WIDTH)
    y_center = (min_y + max_y) / (2 * IMG_HEIGHT)
    width = (max_x - min_x) / IMG_WIDTH
    height = (max_y - min_y) / IMG_HEIGHT
    return f'{obj_class} {round(x_center, 6)} {round(y_center, 6)}' + \
        f'{round(width, 6)} {round(height, 6)}\n'
```

B.2 Tiling images and transform labels

```
class Label():
    ...

    def transform(self, new_height, new_width, y_cord, x_cord, tile_size):
        self.width = float(self.width) * (self.img_width / new_width)
        self.height = float(self.height) * (self.img_height / new_height)
        self.x_center = (self.get_x_pixels() -
                         (tile_size * x_cord)) / new_width
        self.y_center = (self.get_y_pixels() -
                         (tile_size * y_cord)) / new_height
        self.img_height = new_height
        self.img_width = new_width

    def check_and_correct_horizontal(self):
        right_edge = self.x_center + (self.width / 2)
        left_edge = self.x_center - (self.width / 2)
        if left_edge < 0:
            self.width = right_edge
            self.x_center = self.width / 2
            left_edge = 0
            print("Cutting box left edge")
        if right_edge > 1:
            self.width = 1 - left_edge
            self.x_center = 1 - (self.width / 2)
            print("Cutting box right edge")

    def check_and_correct_vertical(self):
        top_edge = self.y_center - (self.height / 2)
        bottom_edge = self.y_center + (self.height / 2)
        if top_edge < 0:
            self.height = bottom_edge
            self.y_center = self.height / 2
            top_edge = 0
            print("Cutting box top edge")
```

```
        if bottom_edge > 1:
            self.height = 1 - top_edge
            self.y_center = 1 - (self.height / 2)
            print("Cutting box bottom edge")


# Crop/Splitting single image into the desired sizes
def split_and_save(image, new_size, labels, filename):
    im = np.array(image)
    tiles = [im[x:x + new_size, y:y + new_size]
             for x in range(0, im.shape[0], new_size)
             for y in range(0, im.shape[1], new_size)]

    width, height = image.size
    number_horizontal = (width // new_size) + 1
    number_vertical = (height // new_size) + 1

    # image and labels out folders
    IMG_OUT = "./assets/rgb-tiled/training/images/"
    LABEL_OUT = "./assets/rgb-tiled/training/labels/"

    for i, tile in enumerate(tiles):
        tile_contain_sheep = False
        tiled_img = Image.fromarray(tile)
        new_width, new_height = tiled_img.size
        y_pos = i // number_horizontal % number_vertical
        x_pos = i % number_horizontal
        for label in labels:
            label_in_correct_tile = (
                label.get_y_pixels() // new_size) == y_pos and (
                label.get_x_pixels() // new_size) == x_pos
            if label_in_correct_tile:
                label.transform(new_height, new_width, y_pos, x_pos, new_size)
                label.check_and_correct_horizontal()
                label.check_and_correct_vertical()
                with open(f'{LABEL_OUT}{filename}_{i}.txt', 'a') as f:
                    f.write(f'{label}\n')
                tiled_img.save(f'{IMG_OUT}{filename}_{i}.JPG', 'JPEG')
                tile_contain_sheep = True
        if not tile_contain_sheep:
            tiled_img.save(f'{IMG_OUT}{filename}_{i}.JPG', 'JPEG')
...
```

C Results of All Models

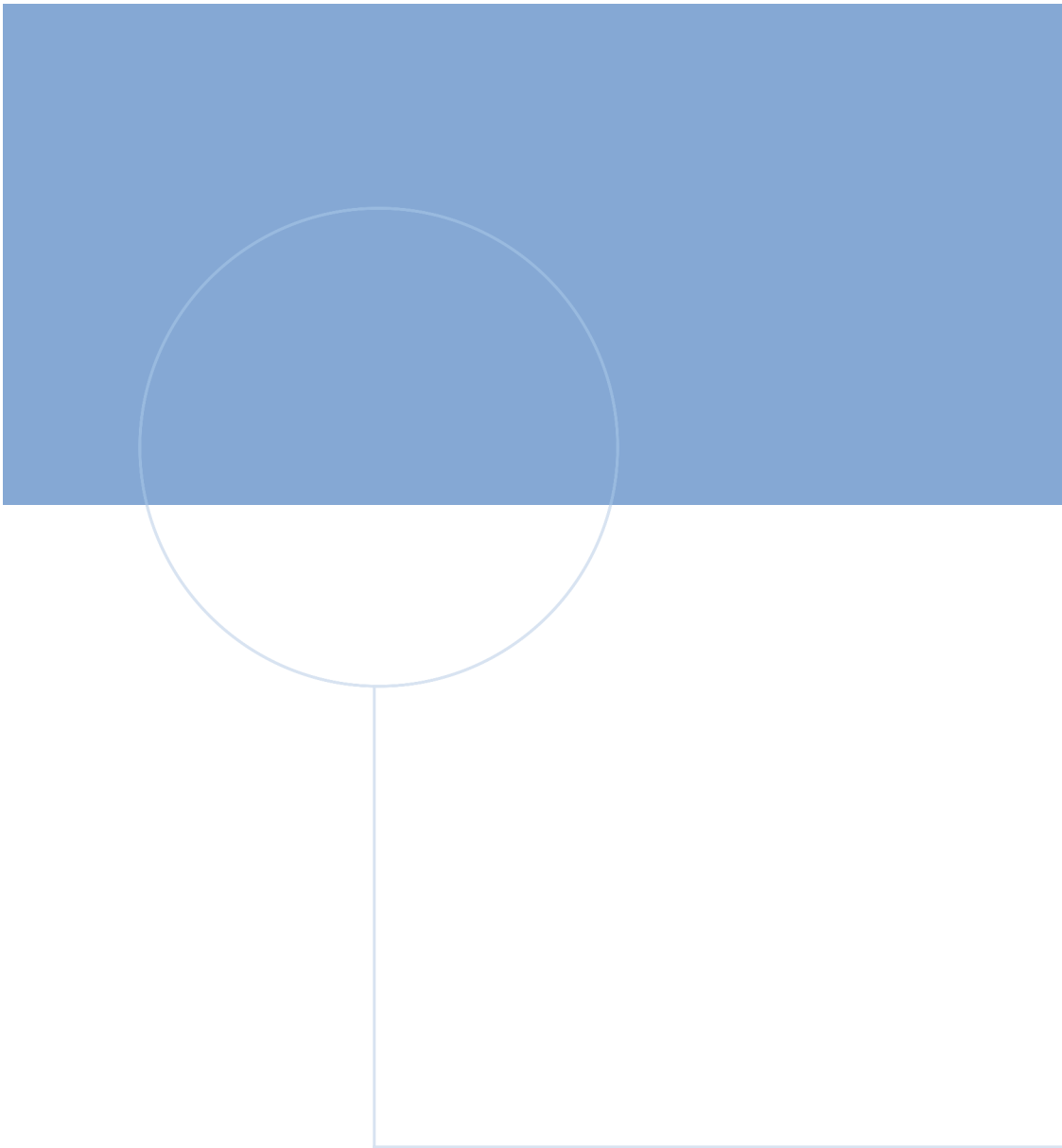
Table 10 and 11 contain results from all trained MSX and RGB models respectively. All results are from the test data set.

	Computer					Smartphone							
	Conf. threshold of 0.01					Conf. threshold of 0.01				Conf. threshold of 0.5			
Model Name	Inf. (ms)	AP	AP@.5:.95	Precision	Recall	Inf.	AP	Precision	Recall	AP	Precision	Recall	SR
msx_96_s	2.8	0.022	0.003	0.101	0.078	-	-	-	-	-	-	-	-
msx_160_s	8.4	0.222	0.056	0.190	0.398	-	-	-	-	-	-	-	-
msx_160_m	12.5	0.343	0.106	0.281	0.456	-	-	-	-	-	-	-	-
msx_160_x	35.4	0.334	0.081	0.266	0.427	-	-	-	-	-	-	-	-
msx_320_s	12.3	0.563	0.136	0.324	0.563	57	0.169	0.487	0.185	0.154	0.850	0.165	0.379
msx_320_m	22.4	0.490	0.137	0.245	0.631	96	0.134	0.500	0.146	0.134	0.790	0.146	0.379
msx_320_x	57.2	0.436	0.120	0.313	0.573	-	-	-	-	-	-	-	-
msx_640_s	24.9	0.654	0.184	0.549	0.699	149	0.192	0.700	0.204	0.176	0.864	0.185	0.417
msx_640_m	52.4	0.624	0.220	0.609	0.641	219	0.294	0.738	0.301	0.294	0.886	0.301	0.456
msx_640_x	144.2	0.682	0.256	0.598	0.689	-	-	-	-	-	-	-	-
msx_960_s	49.4	0.647	0.238	0.575	0.658	285	0.202	0.786	0.214	0.194	0.913	0.204	0.417
msx_960_m	104.4	0.739	0.270	0.620	0.767	477	0.255	0.477	0.272	0.248	0.794	0.262	0.437
msx_960_x	301.3	0.753	0.253	0.595	0.748	-	-	-	-	-	-	-	-
msx_1280_s	83.3	0.590	0.247	0.465	0.641	451	0.296	0.838	0.301	0.287	0.909	0.291	0.456
msx_1280_m	173.5	0.618	0.223	0.573	0.728	750	0.218	0.828	0.233	0.209	0.920	0.223	0.437
msx_1280_x	530.2	0.755	0.548	0.689	0.759	-	-	-	-	-	-	-	-

Table 10: *All results are from the test data set of MSX images. The model name describes the model's image input type, image resolution and size of the model. The confidence threshold is either 0.01 or 0.5 to maximise AP or balance precision and recall respectively. SR is the model's sheep retrieval.*

	Computer						Smartphone					
	Confidence threshold 0.01						Conf. 0.5					Conf. 0.01
Model Name	Inf. (ms)	Test res.	AP	AP@.5:.95	Precision	Recall	Inf.	AP	Precision	Recall	SR	AP
rgb_320_s	8	320	0.202	0.033	0.114	0.352	55	0.271	0.568	0.324	0.627	0.350
rgb_640_s	24.1	640	0.574	0.147	0.424	0.675	145	0.354	0.518	0.521	0.958	0.380
rgb_1024_s	55	1024	0.643	0.144	0.513	0.704	297	0.695	0.693	0.747	0.993	0.727
rgb_1280_s	87	1280	0.400	0.073	0.365	0.585	447	0.497	0.624	0.620	0.852	0.529
rgb_1280_s6	86	1280	0.442	0.083	0.355	0.570	435	0.510	0.525	0.585	0.958	0.538
rgb_4064_s	767	4064	0.628	0.115	0.584	0.683	-	-	-	-	-	-
	483	3200	0.559	0.102	0.559	0.669	-	-	-	-	-	-
tiled_rgb_s	34	tiled	0.816	0.495	0.257	0.931	144	0.430	0.157	0.755	0.881	0.442
	29	640	0.055	0.008	0.280	0.113	142	0.047	0.539	0.049	0.190	0.087
	82	1280	0.661	0.254	0.472	0.705	434	0.576	0.756	0.613	0.782	0.641
	178	**1920**	0.879	0.421	0.454	0.923	851	0.817	0.830	0.824	0.979	0.887
	326	2624	0.913	0.497	0.402	0.944	1420	0.851	0.826	0.866	0.979	-
	482	3200	0.920	0.534	0.392	0.965	-	-	-	-	-	-
	777	4064	0.910	0.535	0.359	0.972	-	-	-	-	-	-
tiled_rgb_m	77	tiled	0.866	0.521	0.256	0.975	253	0.435	0.140	0.868	0.925	0.438
	58	640	0.245	0.069	0.399	0.268	247	0.136	0.750	0.169	0.465	0.310
	178	1280	0.818	0.343	0.540	0.880	743	0.750	0.790	0.796	1.000	0.845
	387	1920	0.913	0.458	0.455	0.965	1571	0.897	0.763	0.930	1.000	0.917
	709	2624	0.921	0.554	0.397	0.972	-	-	-	-	-	-
	1043	3200	0.922	0.567	0.363	0.979	-	-	-	-	-	-
	1680	4064	0.908	0.555	0.319	0.979	-	-	-	-	-	-

Table 11: *All results are from the test data set of RGB images. The model name describes the model's image input type, image resolution and size of the model. The confidence threshold is either 0.01 or 0.5 to maximise AP or balance precision and recall respectively. SR is the model's sheep retrieval.*



NTNU

Norwegian University of
Science and Technology