Escuela
Politécnica
Superior

Animal detection by drone

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Raquel Zarraoa Sardón
Tutor/es:
Miguel Cazorla Quevedo
Francisco Gomez-Donoso

Enero 2022

Universitat d'Alacant
Universidad de Alicante

Animal detection by drone

Autor
Raquel Zarraoa Sardón

Tutor/es
Miguel Cazorla Quevedo
Computer Science and Artificial Intelligence
Francisco Gomez-Donoso
Computer Science and Artificial Intelligence

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Enero 2022

Abstract

Despite animal husbandry being one of the most important industries in our society, the technology used is relatively rudimentary. This lack of technological advancements, make it a difficult and arduous job for humans and limit the quality of life given to the animals. The goal of this project is to create an animal detection application that could serve as a baseline for future projects to assist livestock farmers with tasks such as monitoring, care taking, herding and protection of the animals. To do this, different computer vision techniques have been analyzed, finally choosing the deep learning algorithm YOLO to develop the animal detection application. A detector is trained with a dataset of cows and sheep images, which is later improved by applying data augmentation to the dataset. To further improve the predictions of the detector, a pre-processing and post-processing step is applied to the input image. The detector obtained after applying data augmentation to the training dataset offers fast predictions, making it viable for real-time applications, although obtaining worse results for far away views of the animals. The added processing steps, on the other hand, offer good prediction results for far away views of the animals but cannot be used for real-time applications. Depending on the requirements of each application, one or the other approach can be taken.

Acknowledgments

This dissertation would not have been possible without the support of my family and friends over the years, and particularly during these difficult times with the Covid-19 pandemic.

Special thanks to Miguel Cazorla and Francisco Gómez for helping me and guiding me through this project.

And finally, thank you to all the teachers for the knowledge acquired in the different subjects studied during the last few years.

Contents

ListofFigures

1 Introduction

Agriculture and animal husbandry are some of the oldest occupations in human history. They form the base of our society as they provide us with many of the products needed to survive such as food or clothes. Despite their importance and being part of human history for so long, they have not evolved as much as other industries throughout the years. Especially in animal farming, the way of working is very similar to what it was hundreds of years ago, using animals such as dogs and horses to help with the herding and protection of the cattle.

However, in the last few years we are seeing a change. With the development of technology and specifically artificial intelligence, doors are being opened to bring new techniques and solutions to these fields. These new applications could greatly improve the quality of life of the workers as well as improving the productivity.

Agriculture has already seen a big technological boom with the appearance of applications capable of monitoring soil and crops health, optimizing crop production, caring for the crops and even harvesting them.

On the other hand, the cattle raising industry has not seen much development until very recently, when the first artificial intelligence applications are starting to emerge. In this project, I will use a deep learning algorithm to develop an application capable of detecting livestock for further use in the farming industry.

1.1 Motivation

The idea behind this project came up after I found three sheep that had fallen into a well. The shepherds didn't know they were missing and the sheep had no way to get out of it. Had they not be found, they would have most likely suffered a long and painful death. And certainly, this kind of situation is fairly usual.

But what if they were being monitored at all times? It would be an impossible task for a human, but not for a robot. The robot would be able to regularly check the number of animals and, in case there were some missing, inform the shepherds and even go looking for them.

The base of most artificial intelligence application for farming would be the detection of the different animals. Therefore, I decided to focus the project on developing this basic capability.

1.2 Goals

The main goal of this project is to develop a livestock detection application that can be used in a real life setting. In order to achieve this, it is necessary to:

- Learn about the different techniques available for object detection applications and their qualities
- Generate a dataset with images of different classes of animals
- Train a detector capable of predicting the location and class of different animals in an image
- Optimize the detector's results

2 State of the art

With the technological advancements during past years, more automation solutions have appeared to assist or relieve humans from highly demanding, monotonous or dangerous jobs. One of the main fields where these solutions have been applied is agriculture, where artificial intelligence combined with sensorized vehicles assist in tasks like monitoring, taking care of and even harvesting crops.

In recent years, similar applications are starting to appear in animal husbandry to assist with the monitoring and care taking of the cattle. However, since it is a relatively new research field, not much information can be found about the few existing solutions. In this chapter, I am going to talk about some of the most relevant ones.

2.1 Livestock detection in aerial images using a fully convolutional network

Han et al. (2019) presented a livestock detection algorithm using modified version of U-net and Google Inception-v4 net to monitor the number of livestock on grassland and avoid its overutilization. It follows a similar approach to R-CNN, which will be described later, using U-net to perform pixel-level segmentation of the image and generate regions of interest. An example of the process is shown in Figure 2.1. The precision obtained by this solution is better than that of YOLOv3 and comparable to Faster R-CNN.

2.2 Autonomous farm work

Rocos (2020), a robot operation software company partnered with Boston Dynamics, a robotics company to turn their mobile robot Spot into a sheep herding robot, as represented in Figure 2.2. Spot has been designed to be able to navigate difficult terrain and, combined with the Rocos platform, can be commanded from remote locations or even manually teleoperated. Thanks to the different sensors it is provided with, Spot can collect data from its environment in real time.

2.3 Beefree agro

BeeFreeAgro (2020), presents JOE, an autonomous herding system capable of analyzing terrain obstacles and cow locations; planning the best route to move the herd to a desired location and herding the cattle accordingly using a single drone.

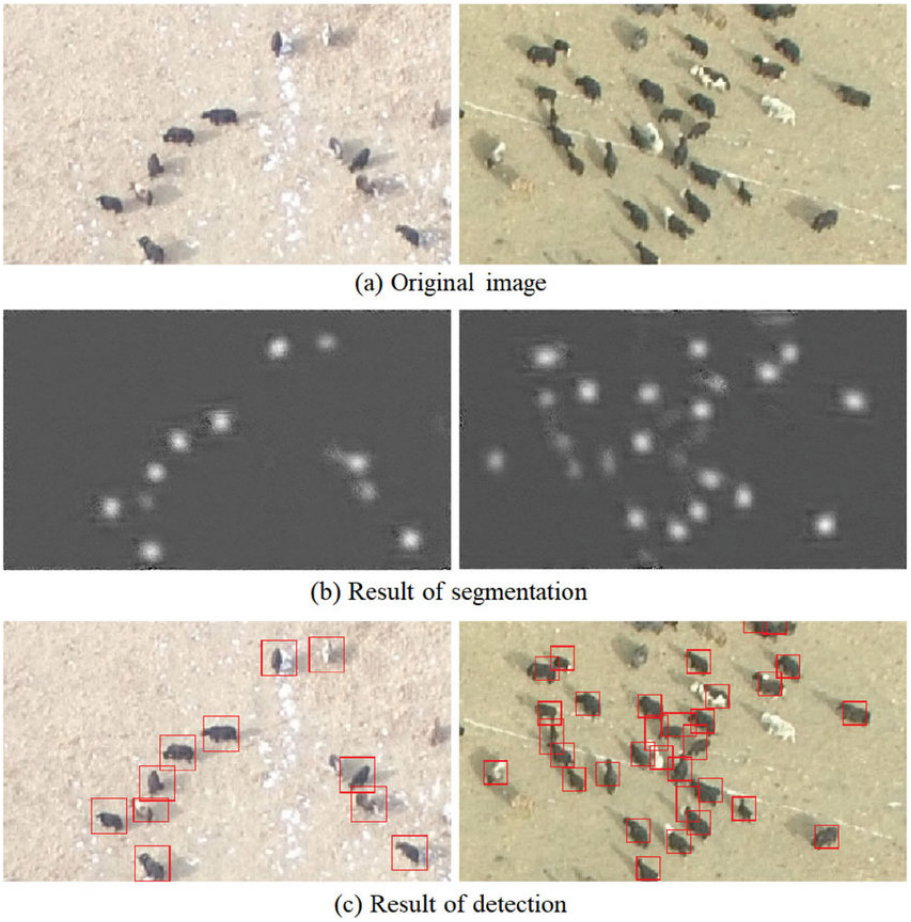


Figure 2.1: Processing example of the livestock detection algorithm. Picture by Han et al. (2019)

Figure 2.2: Mobile robot Spot being used to herd sheep. Picture by Rocos (2020)

3 Methodology

3.1 Object detection

One of the most important senses humans possess is sight. Our eyes are constantly obtaining images from our surroundings and sending them to the brain for processing. The brain is then able to obtain all the necessary information from these images at incredible speed to better understand, navigate and interact with the world.

Computer vision started in the 1960s with the development of artificial intelligence as a way to mimic the human visual system. One or more cameras (the eyes) acquire images and send them to the computer (the brain) for processing. There are different sub-domains of computer vision according to what information is being obtained; such as object detection, scene reconstruction, video tracking, etc. For this project, I will focus on object detection.

There are two main computer vision techniques for object detection: traditional and deep learning.

3.1.1 Traditional techniques

Traditional techniques are based on the extraction of certain features of the image. Features are descriptive or informative patches of an image that can be used as a set of rules an object has to satisfy to be a certain class. These techniques require the user to choose what features to look for, which can become an impossible task for complex object classification.

Some examples of traditional computer vision algorithms are:

- Scale Invariant Feature Transform (SIFT): invented by Lowe (1999), based on the extraction of features invariant to uniform scaling, illumination changes and orientation.
- Speed Up Robust Features (SURF): published by Bay et al. (2006), is similar to SIFT although several times faster and more robust. The extracted features are invariant to scale but not orientation.
- Features from Accelerated Segment Test (FAST): originally proposed by Rosten & Drummond (2006), is a very efficient corner detection algorithm for use in real time applications.

3.1.2 Deep learning

Deep learning, on the other hand, requires the machine to learn what features to look for when classifying objects. One of the main elements of deep learning are convolutional neural

networks, a type of artificial neural network capable of recognizing patterns very efficiently

3.1.2.1 Artificial neural networks

Artificial neural networks are computing systems inspired by the biological neural networks of our brain. They consist of simple units (artificial neurons) interconnected with each other. These neurons receive inputs and produce a single output that is sent to other neurons. As seen in Figure 3.1, each input has an associated weight that represents its relative importance. The neuron takes the weighted sum of its inputs, to which we can add a bias term, runs it through the activation function to compute the output and determines if the neuron is activated or not, also shown in Figure 3.1.

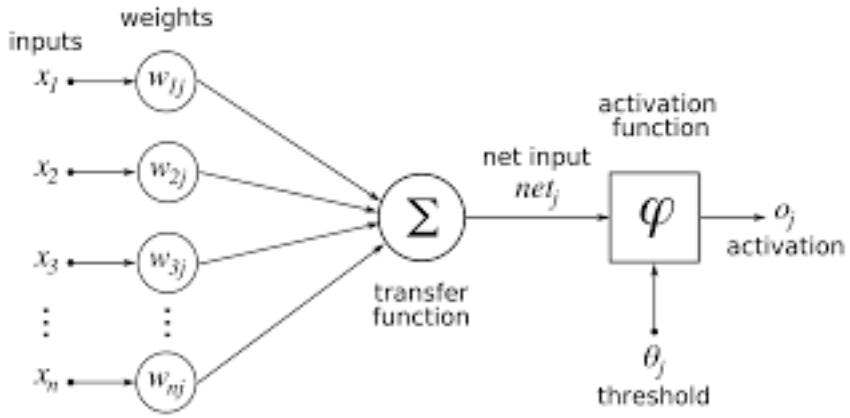


Figure 3.1: Artificial neuron representation

To form the artificial neural network, these neurons are organized in different layers, with each layer only communicating with the ones immediately before and after. All artificial networks consist of an input layer which receives the external data and an output layer which produces the result. In between these layers, there are zero or more hidden layers.

The layers can be connected in many different ways. Some of the most important ones are: fully connected layers, they have every neuron of one layer connected to every neuron of the next layer; pooling layers, they have a group of neurons on the first layer connecting to a single neuron on the next layer, reducing the number of neurons on each layer; and recurrent networks, which allow connections to neurons in the same layer or previous ones.

Neural networks learn by readjusting the weights within the network to improve the accuracy of the result. The time needed to train a network depends, in part, on the learning rate. The learning rate defines the level of adjustment taken after each observation. A higher learning rate has lower accuracy but makes training faster, while a lower learning rate takes longer but may improve accuracy. The training is complete when new observations no longer improve accuracy. To examine this, a cost function is defined and evaluated periodically; when its values stop declining, training is complete.

3.1.2.2 Convolutional neural networks

Convolutional neural networks (CNN) are a type of artificial neural network very efficient at capturing patterns in multidimensional spaces, which makes them good for object detection. The architecture of CNNs is based on the the connectivity patterns of neurons found in the visual cortex part of the human brain. Each neuron is only affected by their respective receptive field, a region of the visual field, and together they cover the entire visual field.

The network consists of one or more convolution layers, which extract certain features from the image; pooling layers, which reduce the number of features by only keeping the most important ones; and a fully connected layer to learn non-linear combinations of the represented features. The CNN architecture is represented in Figure 3.2

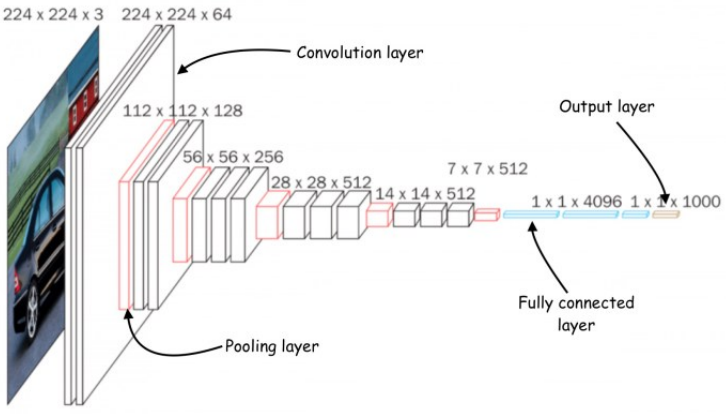


Figure 3.2: Representation of a CNN architecture

The convolutional layer performs a matrix multiplication operation between the kernel or filter and each portion of the image. The kernel has the same depth as the image. For example, an RGB image has three channels (red, green and blue); therefore, the kernel will also have three channels . The goal of this layer is to extract high level features from the image to create feature maps. The first convolutional layer of a CNN obtains low level features and, with added layers, higher level features are added.

The pooling layers reduce the size of the feature maps. These layers decrease the computational power needed and maintain the dominant features. The pooling layers can be: max pooling, which return the highest value obtained by the kernel operation, or, average pooling, which return the average of all the values. Max pooling can act as a noise suppressant and performs better than average pooling.

3.1.2.3 Common deep learning algorithms

Some of the most important deep learning architectures for object detection are:

- R-CNN model family: stands for Region-based Convolutional Neural Network and includes three techniques:
  - R-CNN: described by Girshick et al. (2013), is one of the first applications of CNN. The model proposes bounding boxes as candidates for potential objects.The proposals are then computed by the CNN feature extraction one by one, which makes this a slow process.
  - Fast R-CNN: an extension to improve the speed of R-CNN, Girshick (2015).
  - Faster R-CNN a further improved version of Fast R-CNN, Ren et al. (2015).
- You Only Look Once (YOLO): first described by Redmon et al. (2015). The model is a single-stage detector, meaning it needs only one pass through the neural network to predict the object's bounding box and their class. This approach has a lower accuracy than the R-CNN model but is significantly faster. The model has since been updated and improved multiple times:
  - YOLOv2 (YOLO9000): Redmon & Farhadi (2016), model capable of predicting 9000 object classes thanks to the use of WordTree.
  - YOLOv3: Redmon & Farhadi (2018), including a deeper feature detector.
  - YOLOv4: Bochkovskiy et al. (2020), proposed as a faster and more accurate model.
  - YOLOv5: Jocher (2020), open source version of the model that offers easier integration and high inference speed.

3.2 YOLOv5

Traditional techniques can sometimes solve simple problems more efficiently than deep learning, such as image stitching or 3D mesh reconstruction. Since the desired features are decided by the user, they are more general and can be used for any image, whereas deep learning features are specific to the training images used and need a well constructed dataset to obtain the same results.

However, deep learning offers greater accuracy for applications such as object detection and requires less expertise to correctly utilize. For these reasons, I decided to use the YOLOv5 algorithm mentioned earlier.

Despite its name, YOLOv5 cannot be considered a new YOLO algorithm, since it does not implement any new architectures or techniques to the previous versions of the algorithm. However, as a piece of software developed in Python, is more accessible and easily integrated to existing pipelines.

3.2.1 Model architecture

As mentioned earlier, YOLOv5 is a single-stage object detector and therefore the architecture consists of three main parts: backbone, neck and head. A representation of the model architecture is shown in Figure 3.3

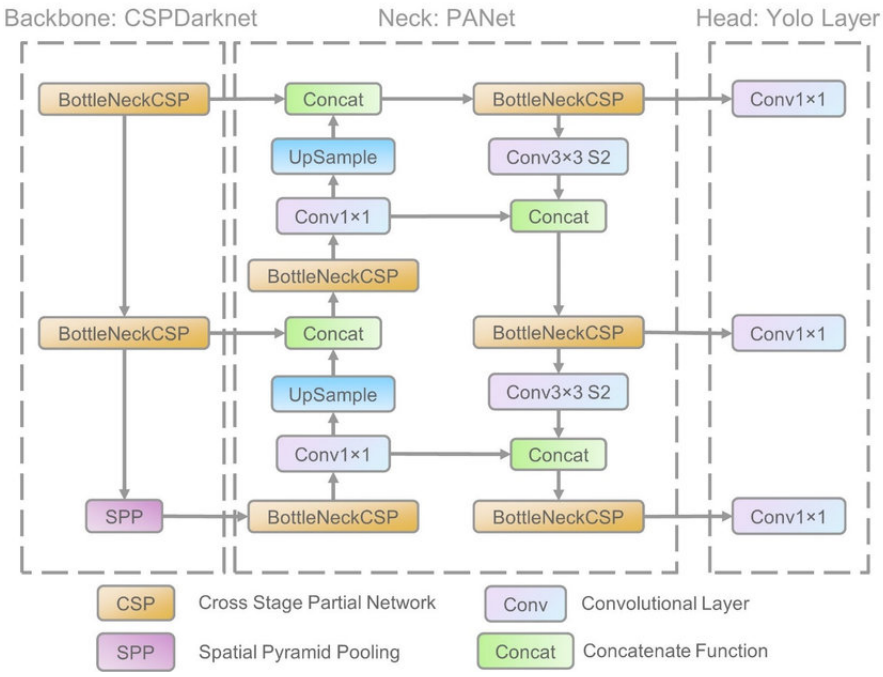


Figure 3.3: YOLOv5 model architecture representation

The model backbone is formed by Cross Stage Partial Networks (CSP), Wang et al. (2019), used to extract important features from the input image. The model neck's main function is to generate feature pyramids which help with object scaling generalization. To do this, YOLOv5 uses Path Aggregation Networks (PANet), Liu et al. (2018). Finally, the head performs the final detection part and generates output vectors with the class, score and bounding box.

The activation function used is a very important part of any neural network. YOLOv5 uses a combination of Leaky ReLU and Sigmoid functions. The Leaky ReLU function is used in the hidden layers while the sigmoid function is used for the final detection layer.

Leaky ReLU is a variation of the Rectified Linear Unit (ReLU) function shown in equation 3.1. The ReLU function can be computed inexpensively, making it very popular. However, it does present two main issues. Firstly, it is not continuously differentiable, which can impact the training performance. Secondly, it sets all negative values to zero. Since the gradient of zero is zero, if a neuron is initialized with large negative values, it is stuck at zero and it "dies".

This can potentially greatly impact the network making it stop learning and underperform.

$$f(x) = \begin{cases} 0, \ if \ x < 0 \\ x, \ otherwise \end{cases} \tag{3.1}$$

Leaky ReLU solves this problem by causing a slight information leak in the left part of ReLU as shown in the equation 3.2 This allows neurons initialized with large negative values to recover.

$$f(x) = \begin{cases} 0.01x, \ if \ x < 0 \\ x, \ otherwise \end{cases} \tag{3.2}$$

The Sigmoid function used for the final detection layer, shown in equation 3.3, outputs in a range of (0,1) which is ideal when estimating probabilities as they have a very similar range of [0,1].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

3.2.2 Training

Training the neural network in YOLOv5 is a highly customizable process, with many different parameters from the image size to the utilization of custom network architectures instead of the pre-defined models. However, most users will only modify a small number of them. Some of the most important ones are:

- Image size: size of the square input image in pixels. The input image is scaled to match the longer side with the image size given; the shorter side is padded with grey if needed to get a square image.
- Batch size: number of images passed through the network at a time. Recommended number is the biggest allowed by the graphics processing unit (GPU) as it decreases training time.
- Number of epochs: number of times all the training images are passed through the network.
- Data augmentation options: YOLOv5 offers a number of different data augmentation options such as image rotation, scaling, upside-down flipping, left-right flipping, translation, etc. The user can define the level of occurrence for each of them.
- Learning rate: level of adjustment taken after each epoch. A higher learning rate has lower accuracy but makes training faster, while a lower learning rate takes longer but may improve accuracy

Once the parameters have been established, the training can be done. The neural network takes as input a set of images with their corresponding annotation file. The annotation file includes all the annotations belonging to one image. Each annotation describes the class and location of an object withint the image. Annotations must follow a specific format to be

usable by YOLOv5.

In the YOLOv5 format each line of the annotation file belongs to a different object. Each annotation consists of five parameters. The first parameter is an integer starting at 0 and belongs to the object class. The next four parameters describe the object bounding box by the x coordinate of the center, y coordinate of the center, width and height respectively. These values are also normalized by the image dimensions.

4 Development

4.1 Dataset creation

One of the most important parts of any deep learning application is the training dataset; the set of images or other data given to the neural network to learn from. YOLOv5 is based in supervised learning, meaning it requires labelled information to learn. Therefore, the dataset needs to contain annotations with every object's class and location for each image. The main goal of this project is to create an animal detection application for drones. For that reason, the network must be trained with drone footage.

When trying to put together the training dataset, it was clear there were not many public datasets that fulfilled the requirements. In fact, only two were found: the Verschoor Aerial Cow Dataset and the Aerial Sheep Dataset.

The Verschoor Aerial Cow Dataset, created by van Gemert et al. (2014), consists of multiple videos taken with a GoPro HERO 3 at 60fps with 1080p quality. An annotation file for each video is also included. Each line of the annotation file belongs to a different annotation and it contains the following information:

1. Track ID: All rows with the same ID belong to the same path.
2. Xmin: The top left x-coordinate of the bounding box.
3. Ymin: The top left y-coordinate of the bounding box.
4. Xmax: The bottom right x-coordinate of the bounding box.
5. Ymax: The bottom right y-coordinate of the bounding box.
6. Frame: The frame that this annotation represents.
7. Lost If 1, the annotation is outside of the view screen.
8. Occluded If 1, the annotation is occluded.
9. Generated: If 1, the annotation was automatically interpolated.
10. Label: The label for this annotation, enclosed in quotation marks.

Since the YOLOv5 software requires an image dataset and a specific annotation format, this dataset had to be adjusted. The different videos were converted to frames and individual annotation files were created for each of them; the resulting number of images was over 15k.

The annotations also had to be modified to conform to the YOLOv5 format. First, all the annotations with the Lost flag were eliminated as they were not located within the image. Second, the class labels were converted from strings to integers; in this case there was only one class "Cow" which became class number 0. Lastly, the bounding box coordinates were transformed to the YOLOv5 format following equations 4.1, 4.2, 4.3 and 4.4.

$$width = \frac{Xmax - Xmin}{Image\,width} \tag{4.1}$$

$$height = \frac{Ymax - Ymin}{Image\,height} \tag{4.2}$$

$$x_{center} = \frac{Xmin + width/2}{Image\,width} \tag{4.3}$$

$$y_{center} = \frac{Ymin + height/2}{Image\,height} \tag{4.4}$$

The *Aerial Sheep Dataset* (2021), contains over 4.1k images of sheep with a resolution of 600x600 pixels. The annotations follow the YOLOv5 format. The only modification needed was changing the class number from 0 to 1.

Both datasets were combined to create the training dataset with over 21k images and later partitioned into a training, validation and test sets containing 80%, 10% and 10% of the images respectively.

Due to the small amount of datasets found for training, a secondary dataset was created to be able to analyze the quality of the detection in different conditions such as changes in lighting, weather, footage quality, background,etc. The testing dataset is not labelled and consists of four videos, three of them containing footage of cows and the last one containing footage of sheep:

1. *Checking Cattle with Drone* (2015)
2. *Drone Cattle Monitoring DJI Mavic Pro* (2018)
3. *Dutch cows | The Netherlands |Droneshots | DJI Mavic Mini 2,7k* (2020)
4. *Moving Sheep Aerial View* (2017)

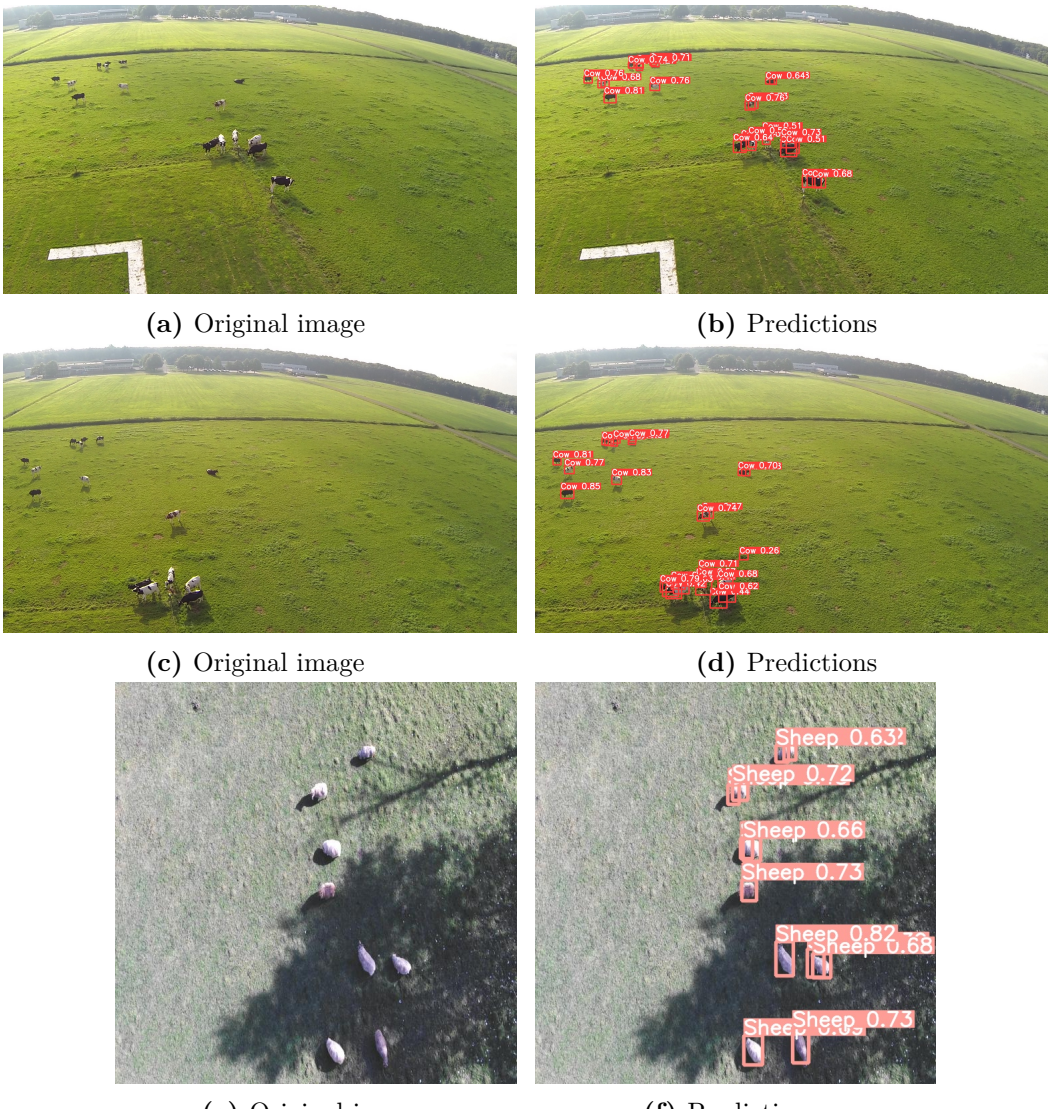
4.2 Training a preliminary neural network

To understand and learn how to work with the YOLOv5 detector, I first trained a preliminary neural network. Since the results weren't very important, the training was done with an image size of 200 pixels and for only 50 epochs, which allowed for faster training.

This detector was run on the training dataset's test set of images. Due to the lack of diverse datasets available, the images used to train and test each class depict the same individuals,

landscape and are taken with the same cameras. Therefore, they are very similar and the detector should be able to locate and identify the different animals accurately. The results are shown in Figure 4.1.

Each prediction is represented as a coloured bounding box of the location of the animal (in this case red for Cow and pink for Sheep), the class name and the confidence score. Predictions with a confidence score below a certain threshold, in this case 0.25, are not returned by the detector as it's considered too low.



(a) Original image **(b)** Predictions

(c) Original image **(d)** Predictions

(e) Original image **(f)** Predictions

Figure 4.1: Comparison sample of the original images and the preliminary detector predictions

As expected, the detector is able to successfully locate and identify the vast majority of

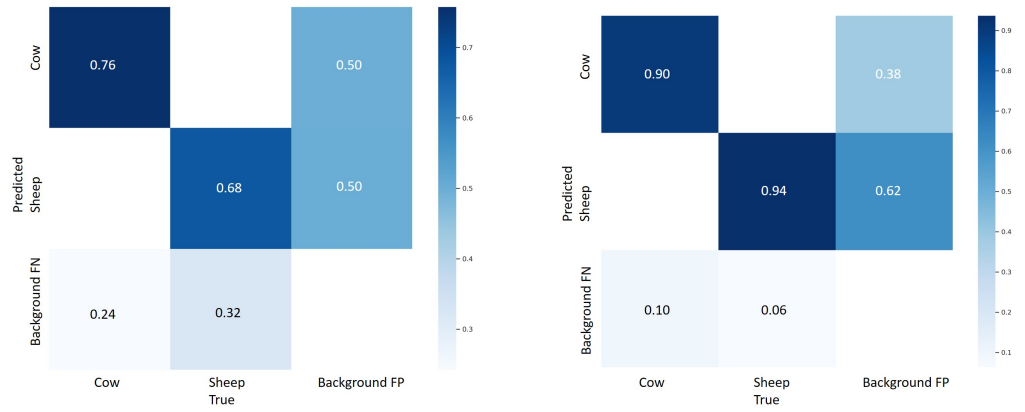
the animals. However, there's frequently multiple detections for a single animal. The reason behind it being the training images are taken from a high up drone view, meaning the objects are smaller; this, combined with the scale reduction from 1080x720 to 200x200 pixels during training means the information the neural network can extract from each object is highly reduced. Despite the limited resources, the results obtained are very good.

4.3 Training the neural network

After the results obtained with the previous training parameters, a second neural network was trained to solve the problems found and improve the accuracy.

As mentioned earlier, the preliminary detector obtained multiple predictions for a single animal due to the reduced image size. To solve this, the training image size was augmented from 200 to 416 pixels.

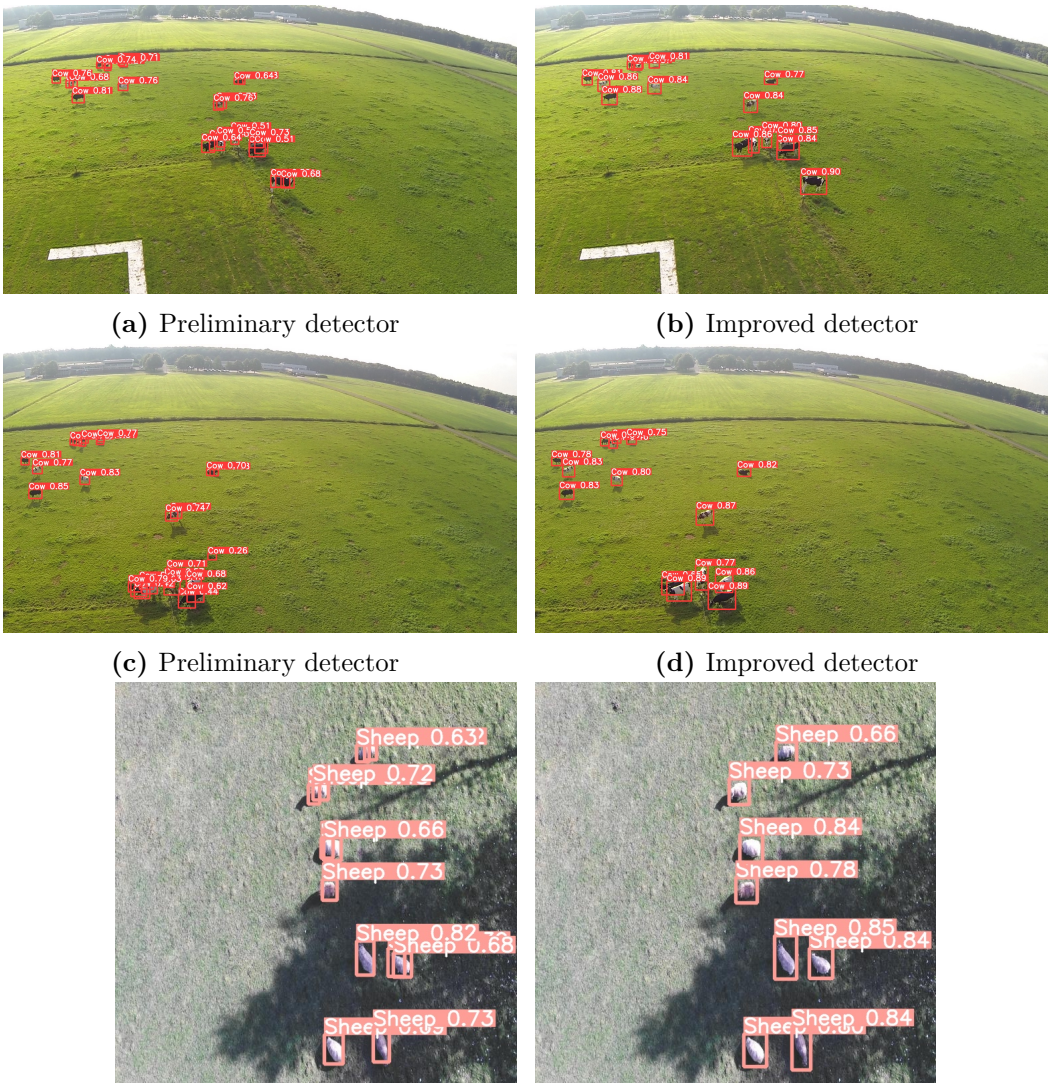
To improve the detector performance, the number of epochs was increased from 50 to 100. As shown in Figure 4.2, the prediction accuracy for cow improves from 0.76 to 0.9 and the accuracy for sheep improves from 0.68 to 0.94. For the preliminary detector the false positives are equally caused by both classes while for the improved detector, the false positives are mostly cause by the sheep class.



(a) Preliminary detector confusion matrix **(b)** Improved detector confusion matrix

Figure 4.2: Confusion matrix comparison between the preliminary detector and the improved detector

The detector was firstly run on the training dataset's test set of images. As seen in Figure 4.3, this detector no longer produces multiple predictions for a single animal and has a high prediction accuracy. Once this problem was addressed, the detector was tested on the testing dataset to check the accuracy of the predictions with different animals and background from those of the training dataset. The results for each of the testing videos are shown in Figures 4.4, 4.5, 4.6 and 4.7 .



(a) Preliminary detector **(b)** Improved detector

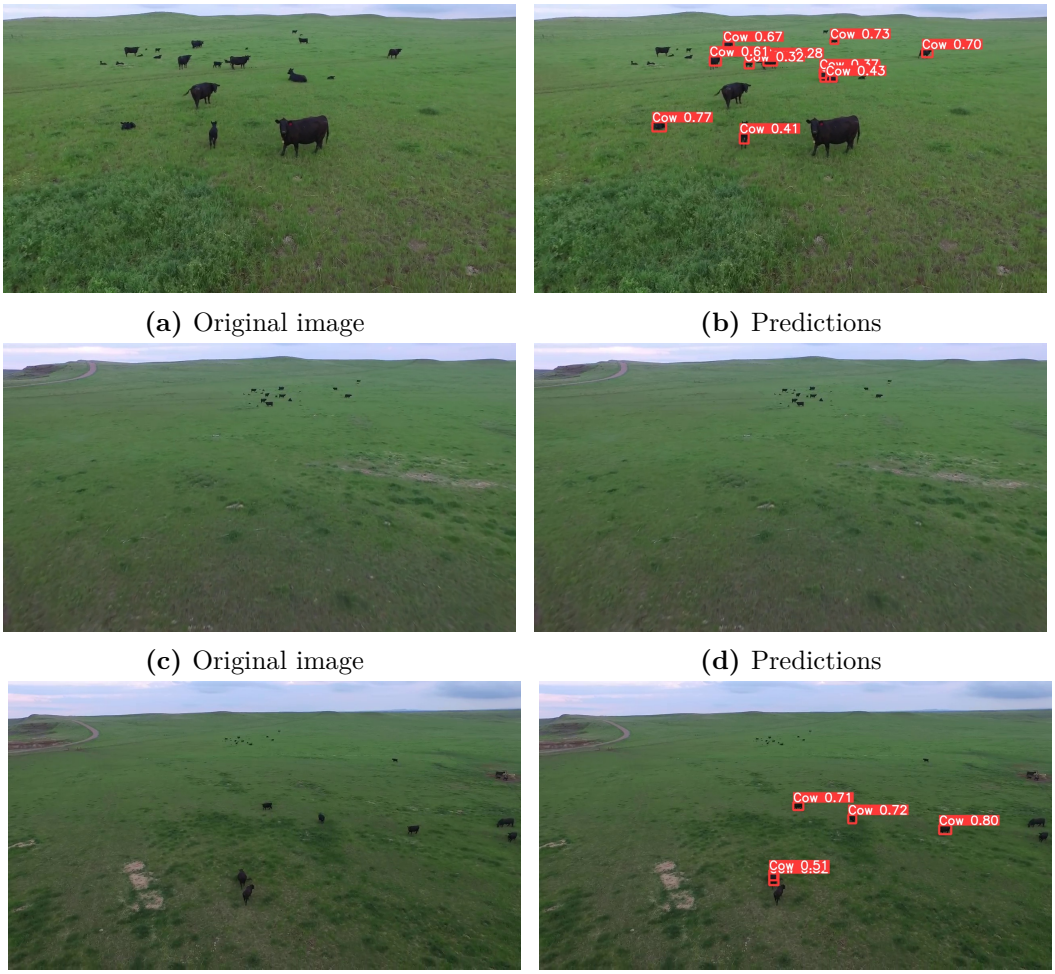
(c) Preliminary detector **(d)** Improved detector

(e) Preliminary detector **(f)** Improved detector

Figure 4.3: Comparison sample of the predictions obtained for the training dataset

For the first video, Figure 4.4, the detector is capable of successfully identifying the animals but with a much lower accuracy. It has trouble detecting animals up close, as shown in Figure 4.4b because the training images are taken from afar, but it does recognize the calves because they are smaller and thus seem further away.

The detector is also not able to detect the animals very far away as shown in Figure 4.4d due to the scale down of the image when running it through the detector, which makes those animals practically disappear.



(a) Original image **(b)** Predictions

(c) Original image **(d)** Predictions

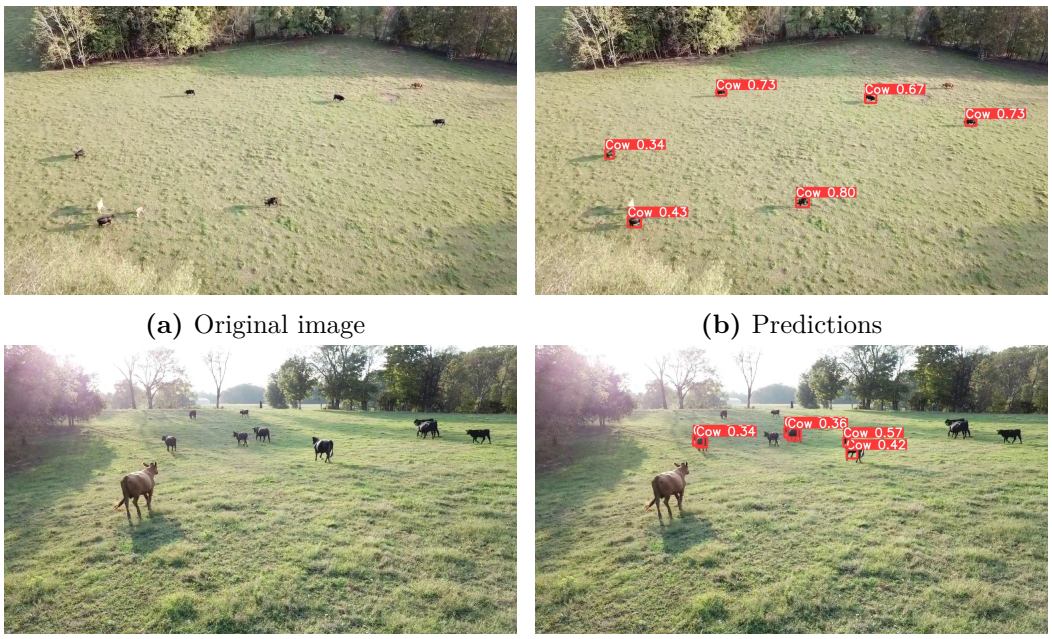
(e) Original image **(f)** Predictions

Figure 4.4: Comparison sample of the original images and the predictions obtained for the first testing video

For the second video, Figure 4.5, the detector has some problems recognizing the brown and fully white cows since the ones from the dataset are black or Friesian (black and white) as shown in Figure 4.5b.

The detector also presents the same issue the preliminary detector had, where it makes multiple predictions for a single animal, when presented with closer views as shown in Figures 4.5d.

The third video, shown in Figure 4.6, is the most similar to the training images for cows in terms of lighting, background and animals and therefore, has a higher accuracy than the others. However, the detector still produces multiple predictions for a single animal and is not able to correctly identify the animals from a closer view.



(a) Original image **(b)** Predictions

(c) Original image **(d)** Predictions

Figure 4.5: Comparison sample of the original images and the predictions obtained for the second testing video

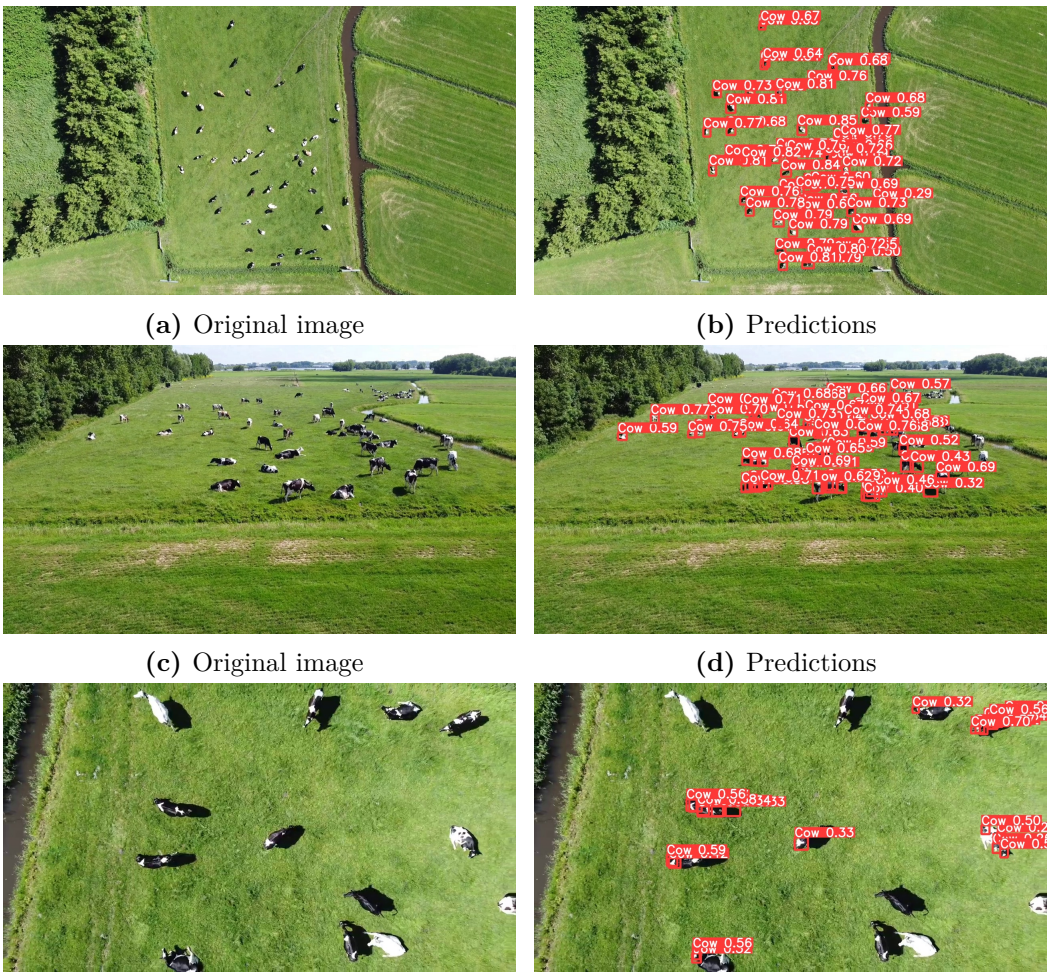
The fourth video, shown in Figure 4.7 presents good predictions for top down and far views of the sheep since they resemble the training dataset more. However the predictions are much worse when presented with angled and closer up views of the animals.

4.4 Data augmentation

The lack of diversity in terms of lighting, background, points of view and animals in the training dataset is one of the main issues for the detector. Since there are no other serviceable datasets available, to mitigate this problem, I used data augmentation to expand the number of images in the existing training dataset.

Each of the dataset images were divided in smaller sub-images of 416x416 pixels in size. The amount of sub-images taken was calculated as the minimum necessary to fully contain the pixels from the original image with a certain overlap between them. In this case, this was calculated as the integer division of the larger original image size and the new image size. The larger original image size is 1920x1080 and the desired image size is 416x416; therefore, each image in the dataset was partitioned into three rows of five images; a total of 15 images.

The first sub-image of each row or column must start on the furthest left or up pixel of the original image respectively; the last sub-image of each row or column must end on the furthest right or down pixel of the original image respectively. Therefore, the overlap between the different sub-images in pixels with the ones surrounding it is calculated as shown in equations



(a) Original image **(b)** Predictions

(c) Original image **(d)** Predictions

(e) Original image **(f)** Predictions

Figure 4.6: Comparison sample of the original images and the predictions obtained for the third testing video

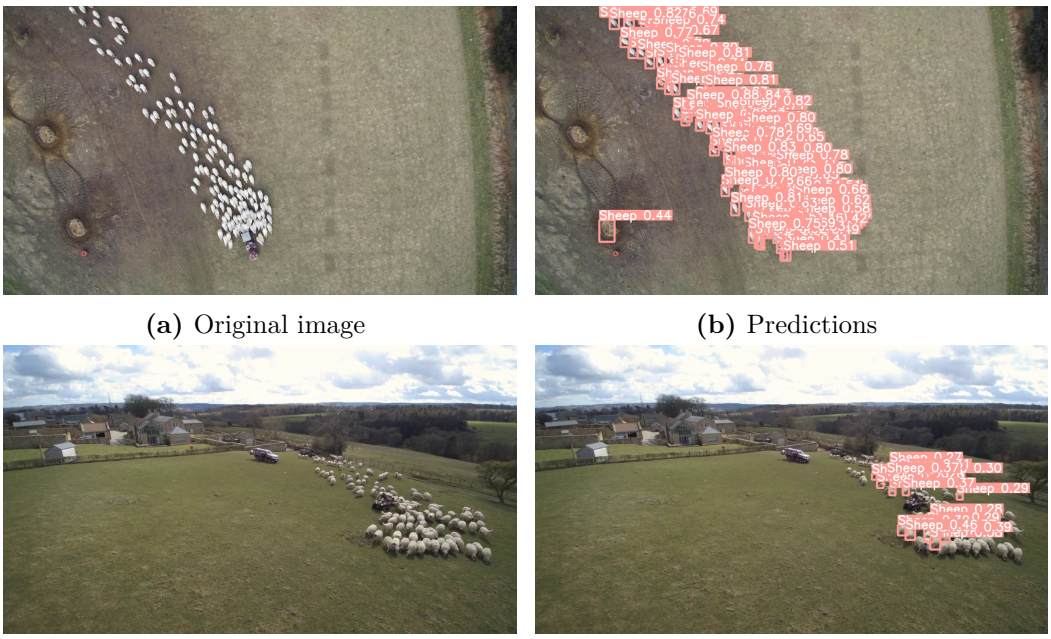
4.5 and 4.6:

$$width\ overlap = \frac{number\ of\ columns \times desired\ width - original\ width}{number\ of\ columns - 1} \tag{4.5}$$

$$height\ overlap = \frac{number\ of\ rows \times desired\ height - original\ height}{number\ of\ rows - 1} \tag{4.6}$$

Knowing the desired size and the necessary overlap, the location of each sub-image within the original is also known. This information allows us to identify which annotations from the original image are located within each sub-image.

These annotations must also be adapted to the new sub-image since they depend on the image size and their relative position within it. The size of the new annotation can also



(a) Original image **(b)** Predictions

(c) Original image **(d)** Predictions

Figure 4.7: Comparison sample of the original images and the predictions obtained for the fourth testing video

change if it is not fully included in the sub-image.

Firstly, I transformed the annotations from the YOLO format to bounding boxes. This allowed me to compare the position of the left, up, right and down sides of each original bounding box (BB) with the position of the corresponding sub-image side. Only the values contained within the new sub-image were mantained; the ones outside of the sub-image were changed to the value for the corresponding sub-image side. Therefore, the values for the left and right side of an image outside of the sub-image to the right would both be the position of the right side of the sub-image. Meanwhile, an annotation contained within the sub-image would be unchanged and one partly inside of it would be reduced to fit inside the sub-image. The annotations with a null width or height are the ones outside the sub-image and are therefore eliminated.

Secondly, I had to adapt the annotations to their new location and the new image size. The annotations are normalized, therefore, knowing the position of the bounding boxes, the new annotations can be calculated as shown in equations 4.7, 4.8, 4.9 and 4.10.

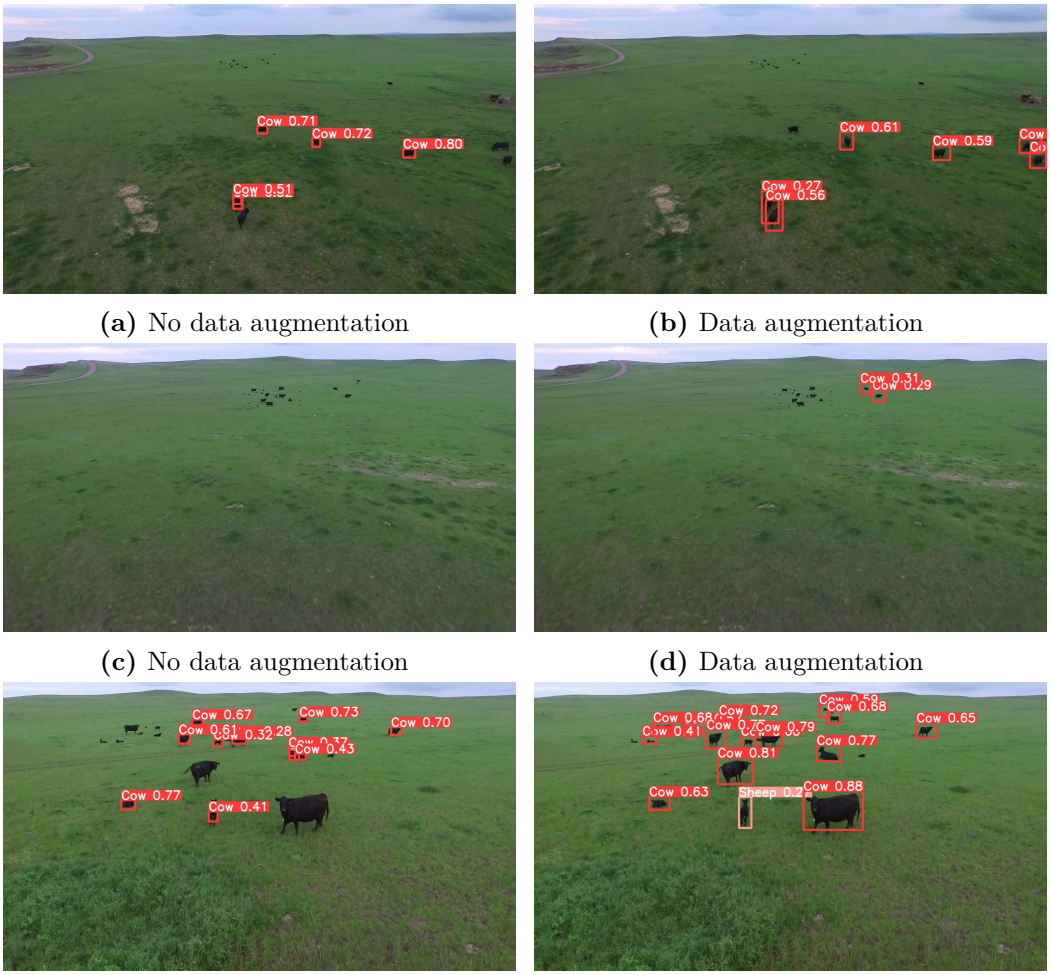
$$width = x_{max} - x_{min} \tag{4.7}$$

$$height = y_{max} - y_{min} \tag{4.8}$$

$$x_{center} = \frac{original\ image\ width \times (sub\,image\ x_{min} + width/2)}{sub\,image\ width} \tag{4.9}$$

$$y_{center} = \frac{original\ image\ height \times (sub\,image\ y_{min} + height/2)}{sub\,image\ height} \tag{4.10}$$

After completing the data augmentation process, the network was trained again with the new images and the original ones. The detector was then run on each of the testing videos. The results for each video are shown in Figures 4.8, 4.9, 4.10 and 4.11 .



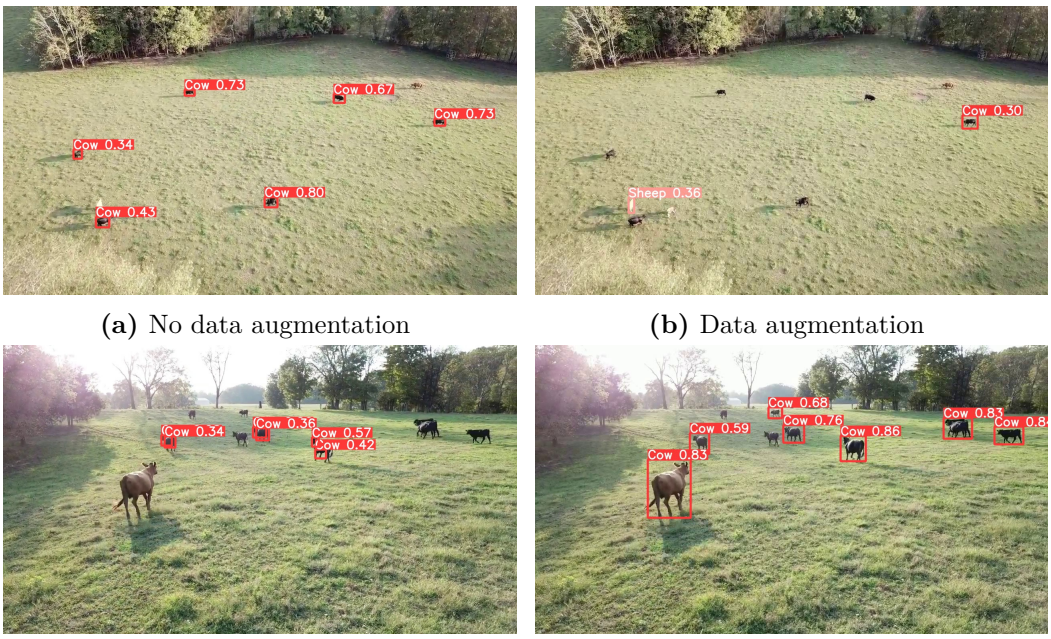
(a) No data augmentation **(b)** Data augmentation

(c) No data augmentation **(d)** Data augmentation

(e) No data augmentation **(f)** Data augmentation

Figure 4.8: Comparison sample of the results obtained with and without data augmentation for the first testing video

The results for the first video (Figure 4.8) show much better predictions with data augmentation. The detector is able to identify and locate the animals from both up close and far away where it couldn't before. However, it still presents some cases of wrong classification.



(a) No data augmentation **(b)** Data augmentation

(c) No data augmentation **(d)** Data augmentation

Figure 4.9: Comparison sample of the results obtained with and without data augmentation for the second testing video

The results for the second video (Figure 4.9) show the same improvement in close up view detection but it fails to identify the animals from a further away point of view.

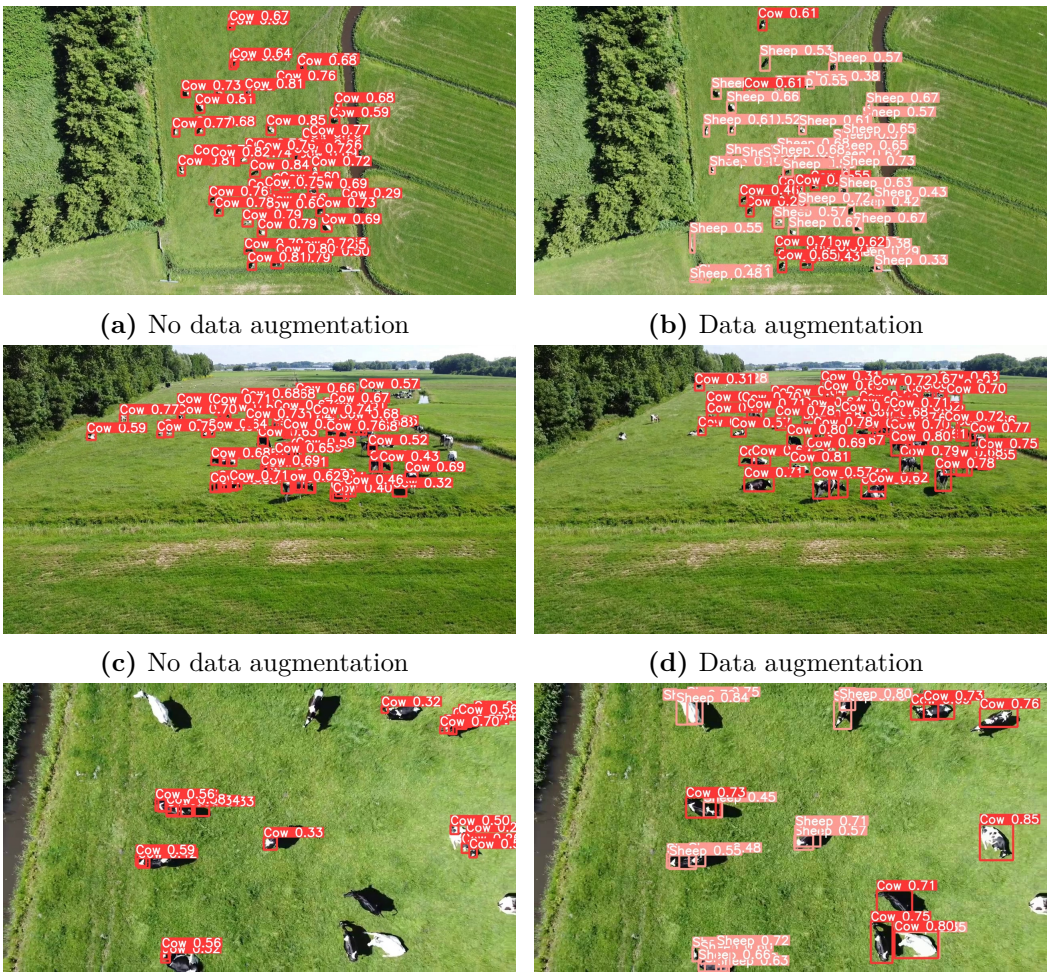
The results for the third video (Figure 4.10) show worse classification in top down views with data augmentation, while maintaining good predictions for angled views. This could be caused by the training images for each class. The ones for the cow class are mostly taken at an angle while the ones for the sheep class are mostly taken from a top down view.

The results for the fourth (Figure 4.11) video show improvement in the predictions of closer animals after data augmentation but also a decline in the prediction for animals further away.

4.5 Added processing steps for detection

After applying data augmentation to the training dataset, the close up view predictions have improved greatly. However, far away view predictions have not improved and in some cases have even worsened.

To solve this problem, instead of modifying the training dataset, I am going to include some pre-processing and post-processing to the testing dataset. The pre-processing step will divide each input image into smaller sub-images to create a ”zoom in” effect. All images will be run by the detector and later, the post-processing step will analyze the predictions for



(a) No data augmentation **(b)** Data augmentation

(c) No data augmentation **(d)** Data augmentation

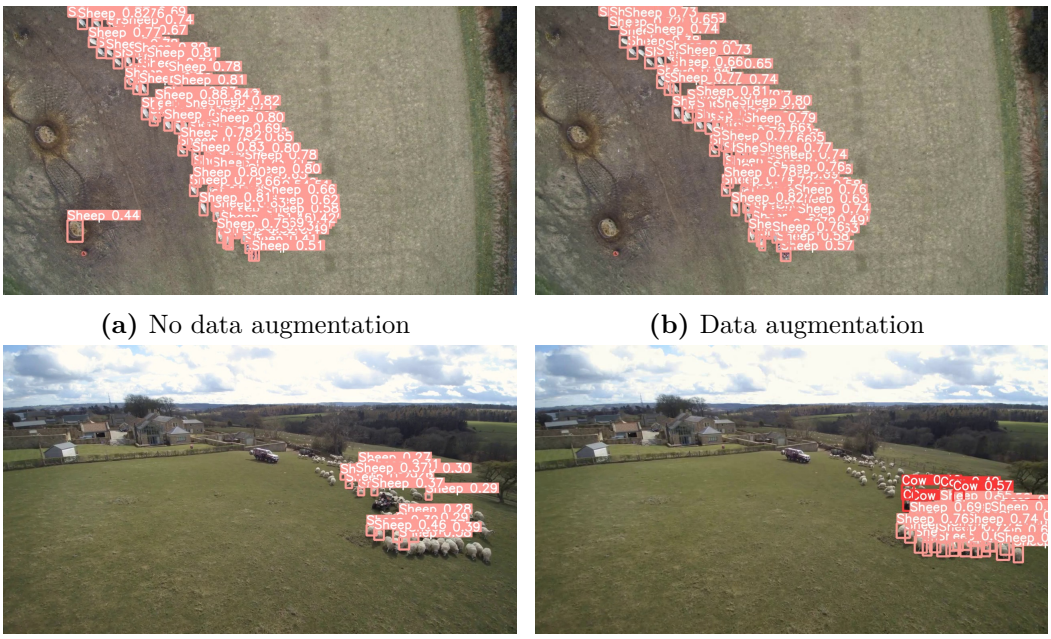
(e) No data augmentation **(f)** Data augmentation

Figure 4.10: Comparison sample of the results obtained with and without data augmentation for the third testing video

each sub-image and combine them to generate the original image's predictions.

The idea is similar to the data augmentation technique used earlier. The input image is going to be partitioned into smaller sub-images as it was done for the data augmentation. The image together with the new set of sub-images will be passed through the detector. Since the input images are scaled down when passed through the detector, using a set of smaller images allows for a lower loss of information.

After the inference is complete, all the bounding boxes resulted from the predictions for each sub-image together with the initial image are combined. To do this, the annotations for the sub-images must be adapted to the initial image dimensions, as a reversion of the process used for the data augmentation. Since the sub-images have some overlap with each other, combining all the predictions results in multiple predictions for a single image.



(a) No data augmentation **(b)** Data augmentation

(c) No data augmentation **(d)** Data augmentation

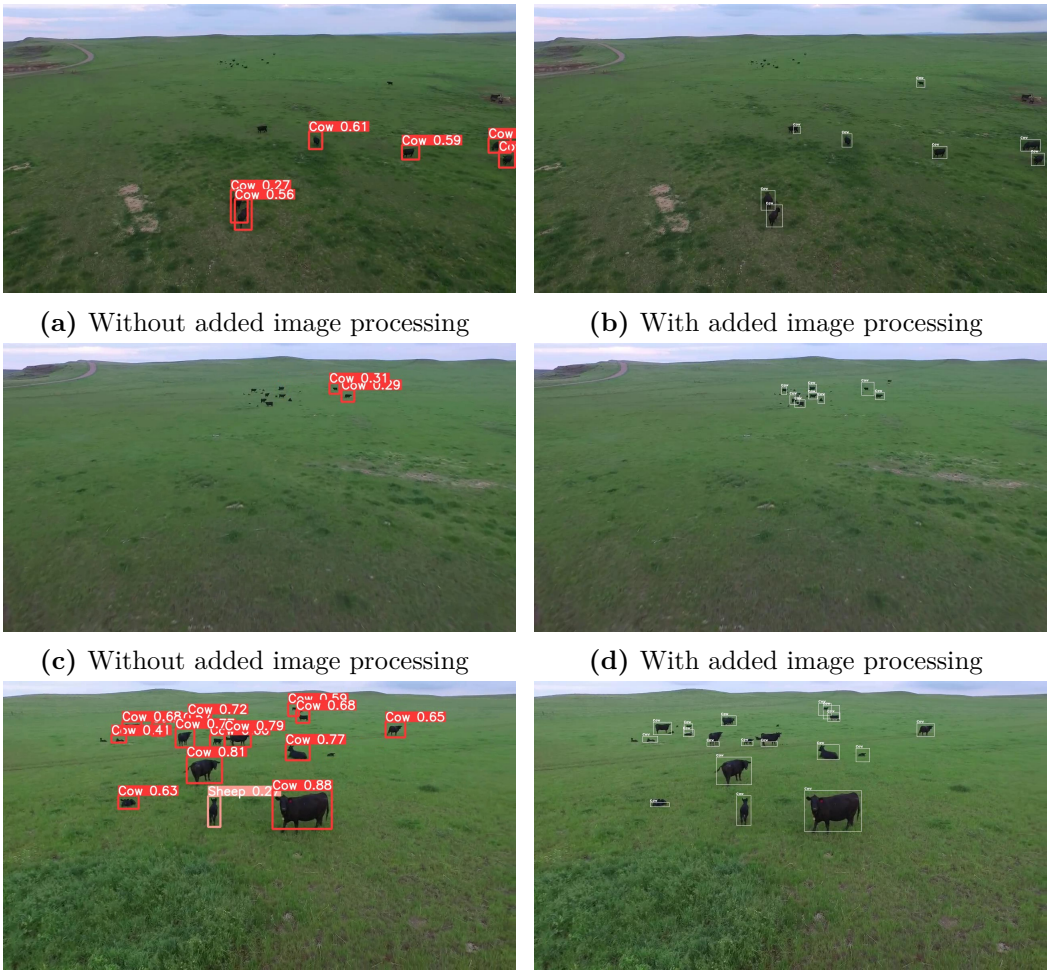
Figure 4.11: Comparison sample of the results obtained with and without data augmentation for the fourth testing video

The redundant predictions can be eliminated using Non-Maximum Suppression (NMS). NMS, first, calculates the area of every bounding box. Then, it calculates the intersection of a bounding box with every other bounding box. The different overlaps are then calculated as the quotient of the intersection with each bounding box and their area. Lastly, if any of the overlaps calculated is greater than a threshold given, the current bounding box is eliminated. This processed is repeated for every bounding box.

The threshold given for the NMS is 0.8; low enough to account for prediction error but high enough to prevent predictions for overlapping animals to be falsely eliminated. The results for each testing video are shown in Figures 4.12, 4.13, 4.14 and 4.15. Due to the post-processing step, the predictions are now represented slightly differently. The bounding box is white for all classes, thinner (which may make it harder to notice in certain images) and no longer shows the confidence score.

The use of extra processing allows the detector to obtain predictions from closer points of view as well as the original one. This results in higher animal detection as shown in Figure 4.15, although increasing the number of predictions per animal. The use of NMS greatly reduces the number of redundant predictions but it is not perfect and can sometimes eliminate the most accurate prediction as shown in Figure 4.16.

All things considered, the extra processing benefits are situational, they improve the num-



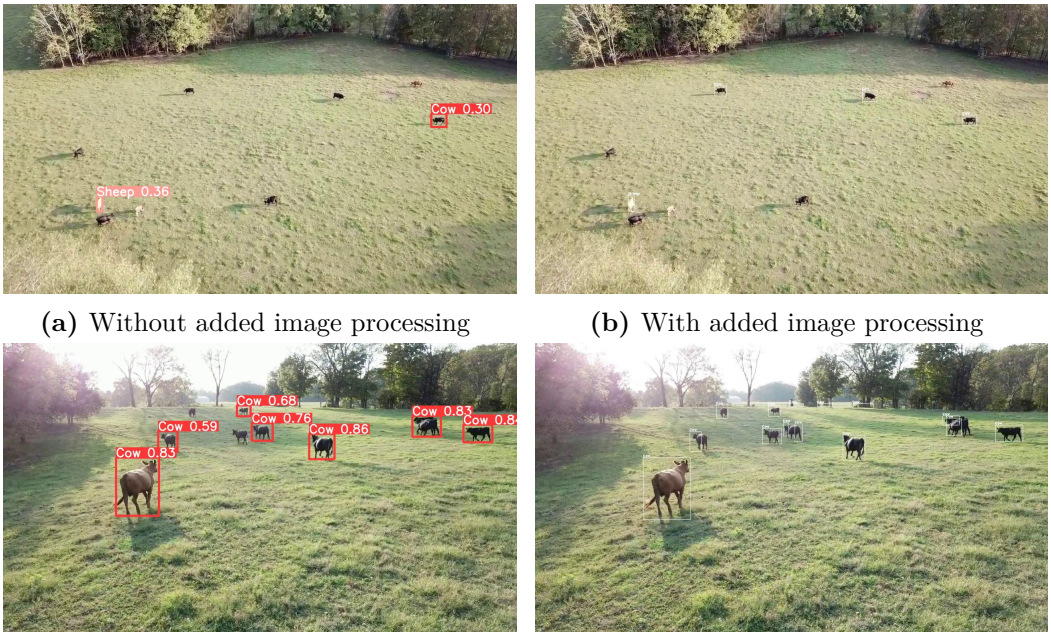
(a) Without added image processing **(b)** With added image processing

(c) Without added image processing **(d)** With added image processing

(e) Without added image processing **(f)** With added image processing

Figure 4.12: Comparison sample of the predictions obtained before and after applying added image processing steps for the first video

ber of animals detected but also introducing more error. Moreover, the detector's average inference time is only 23ms per image, which averages 43 images per second, allowing for real time usage. However, with this technique, the detector runs sixteen times more images (the original and the fifteen sub-images). Adding in the extra processing times for each image make it impossible to be used in real time applications. Therefore, the cost greatly outweigh the benefits.



(a) Without added image processing **(b)** With added image processing

(c) Without added image processing **(d)** With added image processing

Figure 4.13: Comparison sample of the predictions obtained before and after applying added image processing steps for the second video

(a) Without added image processing

(b) With added image processing

(c) Without added image processing

(d) With added image processing

(e) Without added image processing

(f) With added image processing

Figure 4.14: Comparison sample of the predictions obtained before and after applying added image processing steps for the third video

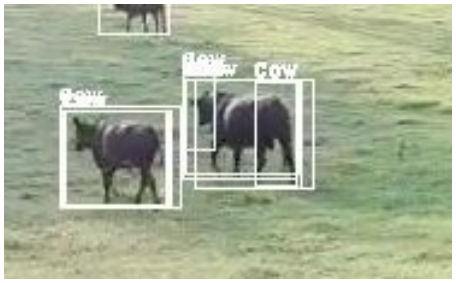
(a) Without added image processing

(b) With added image processing

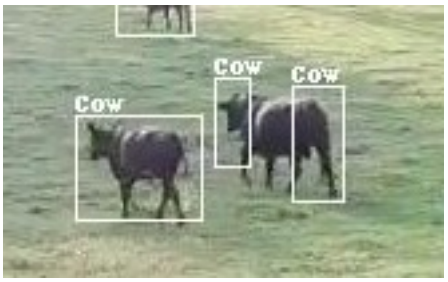
(c) Without added image processing

(d) With added image processing

Figure 4.15: Comparison sample of the predictions obtained before and after applying added image processing steps for the fourth video



(a) Before applying NMS



(b) After applying NMS

Figure 4.16: NMS wrongful bounding box elimination

5 Conclusions

The main goal of this project was to develop an application of animal detection for drones which could serve as the base for a number of different applications in the future such as livestock monitoring, protection or even autonomous herding.

An animal detector was successfully trained for the YOLOv5 architecture. This detector presented an average precision of 0.92 with the training dataset's test set. However, when tested for images with different properties, such as change in lightning, backgrounds, points of view, etc, the precision was much lower, especially for animals closer to the camera.

To improve the training dataset I used data augmentation to generate a set of "zoomed in" images of each original image from the dataset. When the new images were added to the training dataset, the detector improved the precision for animals close to the camera. However, it also cause a slight decrease on the detector's precision for animals far away from the camera.

Finally, to improve the detection of animals far away from the camera, instead of continuing to modify the training dataset, I implemented a pre-processing step that divides the detector's input image into a set of smaller images to create the same "zoom in" effect as before. The set of smaller images together with the original one are then passed through the detector to obtain predictions for all of them. Finally a post-processing step is needed to analyze all the predictions and introduce them into the original input image. This process, improves the number of animals detected although introducing redundance in the predictions and greatly increasing the inference time, making it impossible to use in real time applications.

Depending on the application's needs the detector can be applied with or without the added processing steps. For applications working in real time and requiring the drone to be closer to the cattle such as herding, the best approach is to use the detector without the added processing steps. However, for applications that do not need fast inference times such as obtaining the general location of the cattle, which can be done with the drone further away from the animals, applying the added processing steps would significantly improve the performance.

5.1 Future projects

Some of the things that could be done in the future to improve this project are:

- Expanding the training dataset. One of the main issues of this project is the lack of image variety in the training dataset. Generating and labelling a better training dataset could greatly improved the performance of the detector.

- Optimizing the added processing steps for input images. At the moment, the image processing takes too long and makes it impossible to use for real time applications. However, it could be optimized by implementing a different processing approach or using batch inference in the detector, generating predictions for multiple images at once and saving time.
- Trying different deep learning algotithms. YOLOv5 is one of many deep learning algorithms, the same application could be implemented for different algorithms to analyze which one presents better results.
- Testing the applications in a real setting. Collaborating with the intended users to gather their specific needs, adapt the application accordingly and test it in real situations.

Bibliography

Aerial sheep dataset. (2021). RoboFlow. Retrieved from https://universe.roboflow.com/project/aerial-sheep/1

Ampadu, H. (2021). Yolov3 and yolov4 in object detection. *AI Pool*. https://ai-pool.com/a/s/yolov3-and-yolov4-in-object-detection.

Bay, H., Tuytelaars, T., & Gool, L. V. (2006). SURF: speeded up robust features. In A. Leonardis, H. Bischof, & A. Pinz (Eds.), *Computer vision - ECCV 2006, 9th european conference on computer vision, graz, austria, may 7-13, 2006, proceedings, part I* (Vol. 3951, pp. 404–417). Springer. Retrieved from https://doi.org/10.1007/11744023_32 doi: 10.1007/11744023_32

BeeFreeAgro. (2020). https://www.beefreeagro.com.

Bochkovskiy, A., Wang, C., & Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *CoRR, abs/2004.10934*. Retrieved from https://arxiv.org/abs/2004.10934

Brownlee, J. (2019). A gentle introduction to object recognition with deep learning. *Machine Learning Mastery*. https://machinelearningmastery.com/object-recognition-with-deep-learning/.

Checking cattle with drone. (2015). Michael Delaney. Retrieved from https://www.youtube.com/watch?v=hqVJPW7G5Rw

Chris. (2019). Leaky relu: improving traditional relu. *MachineCurve*. https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/.

Debusmann, B. (2021). Farms are going to need different kinds of robots. *BBC*. https://www.bbc.com/news/business-56195288.

Dickson, B. (2021). An introduction to object detection with deep learning. *TechTalks*. https://bdtechtalks.com/2021/06/21/object-detection-deep-learning/.

Drone cattle monitoring dji mavic pro. (2018). Sky Angel Drones. Retrieved from https://www.youtube.com/watch?v=v82x3L_aqb8

Dutch cows | the netherlands |droneshots | dji mavic mini 2,7k. (2020). Drones & Life. Retrieved from https://www.youtube.com/watch?v=MWoe0ryj82o

Gandhi, R. (2018). R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. *Towards Data Science*. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

Girshick, R. B. (2015). Fast R-CNN. *CoRR*, *abs/1504.08083*. Retrieved from http://arxiv.org/abs/1504.08083

Girshick, R. B., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, *abs/1311.2524*. Retrieved from http://arxiv.org/abs/1311.2524

Han, L., Tao, P., & Martin, R. (2019, 03). Livestock detection in aerial images using a fully convolutional network. *Computational Visual Media*, *5*. doi: 10.1007/s41095-019-0132-5

Jocher, G. (2020). *Yolov5 in pytorch.* https://github.com/ultralytics/yolov5. GitHub.

Kathuria, A. (2021). How to train yolo v5 on a custom dataset. *PaperspaceBlog*. https://blog.paperspace.com/train-yolov5-custom-data/.

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path aggregation network for instance segmentation. *CoRR*, *abs/1803.01534*. Retrieved from http://arxiv.org/abs/1803.01534

Lowe, D. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh ieee international conference on computer vision* (Vol. 2, p. 1150-1157 vol.2). doi: 10.1109/ICCV.1999.790410

Mahony, N. O., Campbell, S., Carvalho, A., Harapanahalli, S., Velasco-Hernández, G. A., Krpalkova, L., … Walsh, J. (2019). Deep learning vs. traditional computer vision. *CoRR*, *abs/1910.13796*. Retrieved from http://arxiv.org/abs/1910.13796

Moving sheep aerial view. (2017). Images of Farming. Retrieved from https://www.youtube.com/watch?v=WW_NsPKvDQY

Rajput, M. (2020). Yolov5 explained and demystified. *Towards AI*. https://towardsai.net/p/computer-vision/yolo-v5%E2%80%8A-%E2%80%8Aexplained-and-demystified.

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, *abs/1506.02640*. Retrieved from http://arxiv.org/abs/1506.02640

Redmon, J., & Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, *abs/1612.08242*. Retrieved from http://arxiv.org/abs/1612.08242

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, *abs/1804.02767*. Retrieved from http://arxiv.org/abs/1804.02767

Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, *abs/1506.01497*. Retrieved from http://arxiv.org/abs/1506.01497

Rocos. (2020). Rocos partners with boston dynamics to provide remote operation of spot robots. https://blog.rocos.io/rocos-partners-with-boston-dynamics.

Rosten, E., & Drummond, T. (2006). Machine learning for high-speed corner detection. In A. Leonardis, H. Bischof, & A. Pinz (Eds.), *Computer vision - ECCV 2006, 9th european conference on computer vision, graz, austria, may 7-13, 2006, proceedings, part I* (Vol. 3951, pp. 430–443). Springer. Retrieved from https://doi.org/10.1007/11744023_34 doi: 10.1007/11744023_34

Saha, S. (2018). A comprehensive guide to convolutional neural networks. *Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

van Gemert, J., Verschoor, C., Mettes, P., Epema, H., Koh, L., & Nature, S. W. (2014). *Verschoor aerial cow dataset.* Retrieved from https://isis-data.science.uva.nl/jvgemert/conservationDronesECCV14w/

Wang, C., Liao, H. M., Yeh, I., Wu, Y., Chen, P., & Hsieh, J. (2019). Cspnet: A new backbone that can enhance learning capability of CNN. *CoRR*, *abs/1911.11929*. Retrieved from http://arxiv.org/abs/1911.11929

Weng, L. (2018). Object detection part 4: Fast detection models. *lilianweng.github.io/lil-log*. Retrieved from http://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html

Yolov1, v2, v3 del algoritmo de detección de objetivos. (n.d.). *Programador Clic*. https://programmerclick.com/article/6782885257/.

Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). Object detection in 20 years: A survey. *CoRR*, *abs/1905.05055*. Retrieved from http://arxiv.org/abs/1905.05055

List of Acronyms and Abbreviations

ANN	Artificial Neural Network.
BB	Bounding Box.
CNN	Convolutional Neural Network.
CSP	Cross Stage Partial Networks.
FAST	Features from Accelerated Segment Test.
GPU	Graphics Processing Unit.
NMS	Non-Maximum Suppresion.
PANet	Path Aggregation Networks.
R-CNN	Region-based Convolutional Neural Network.
ReLU	Rectified Linear Unit.
SIFT	Scale Invariant Feature Transform.
SURF	Speed Up Robust Features.
YOLO	You Only Look Once.