Master's thesis

Master's Programme in Computer Science

Dairy cow re-identification and tracking using computer vision

Hilla Fred

November 8, 2022

Faculty of Science
University of Helsinki

Supervisor(s)

Assoc. Prof. M. Pastell, Assoc. Prof. L. Ruotsalainen

Examiner(s)

Assoc. Prof. M. Pastell, Assoc. Prof. L. Ruotsalainen

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Study programme Master's Programme in Computer Science

Tekijä — Författare — Author

Hilla Fred

Työn nimi — Arbetets titel — Title

Dairy cow re-identification and tracking using computer vision

Ohjaajat — Handledare — Supervisors

Assoc. Prof. M. Pastell, Assoc. Prof. L. Ruotsalainen

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year November 8, 2022	Sivumäärä — Sidoantal — Number of pages 66 pages

Tiivistelmä — Referat — Abstract

Improving the monitoring of health and well-being of dairy cows through the use of computer vision based systems is a topic of ongoing research. A reliable and low-cost method for identifying cow individuals would enable automatic detection of stress, sickness or injury, and the daily observation of the animals would be made easier. Neural networks have been used successfully in the identification of cow individuals, but methods are needed that do not require incessant annotation work to generate training datasets when there are changes within a group. Methods for person re-identification and tracking have been researched extensively, with the aim of generalizing beyond the training set. These methods have been found suitable also for re-identifying and tracking previously unseen dairy cows in video frames.

In this thesis, a metric-learning based re-identification model pre-trained on an existing cow dataset is compared to a similar model that has been trained on new video data recorded at Luke Maaninka research farm in Spring 2021, which contains 24 individually labelled cow individuals. The models are evaluated in tracking context as appearance descriptors in Kalman filter based tracking algorithm. The test data is video footage from a separate enclosure in Maaninka and a group of 24 previously unseen cow individuals. In addition, a simple procedure is proposed for the automatic labeling of cow identities in images based on RFID data collected from cow ear tags and feeding stations, and the known feeding station locations.

ACM Computing Classification System (CCS)
Computing methodologies → Artificial intelligence → Computer vision → Computer vision problems → Object identification
Computing methodologies → Artificial intelligence → Computer vision → Computer vision problems → Tracking

Avainsanat — Nyckelord — Keywords

computer vision, object tracking, cow re-identification, metric learning, object detection

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Contents

1 Introduction

A reliable and low-cost computer vision based method for the identification of dairy cows would be an important step in improving the monitoring of health and well-being of the animals. It would make the daily observation of the cows easier and enable developing algorithms for automatic detection of changes in behavior, indicating stress, sickness or injury. Supervised, closed-set cow identification has been researched with good results with deep neural networks [3][17], but as the herd inhabiting the barn can change frequently, a solution is needed that also works on previously unseen individuals. This way, the costly need of manually annotating a new set of images and re-training the network every time a new cow individual is introduced could be avoided, which is important for a real-world system in agricultural setting.

Instead of posing the identification as a supervised classification problem, one approach is to aim to re-identify the cow individuals. In general, re-identification has been widely studied in humans [46] and somewhat animals [36]. The problem of cow re-identification can be framed as an open-set re-identification problem: given an input image of an individual, a re-identification model then outputs k given number of images that are predicted to be taken from the same individual. Recent work by Andrew et al. [2] showed promising results in open-set cow re-identification that generalizes to previously unseen individuals, and in this thesis further experiments are conducted with their proposed model in chapter 4, by training the model with a new video dataset collected from Luke Maaninka research farm in 2021.

A popular approach to tracking is a tracking-by-detection scheme, where the bounding boxes of the individuals of interest are first extracted from the image, and only after this step the detected bounding boxes are connected to a track or an identity. Therefore a well-performing detection model is needed before tracking or re-identification models can be applied. The detection model choice is discussed in chapter 3, and the performance of different models and training sets are evaluated over a test set from Maaninka.

Another prerequisite for the presented methods is a reliable ground truth labeling, needed both for evaluating the models and being able to connect the track to real-world cow identities in predict time. Luckily, the cows in Maaninka research farm wear RFID ear tags, which are used to record their visits to the feeding stations located around the barn area. The individuals visiting the feeding stations can be trivially identified in the images simply by comparing the coordinates of the bounding box of the detected individual and known coordinates of the feeding station. As the RFID data collected from the Maaninka research farm is very reliable and accurate, it is used both as a basis in the manual annotation work, as described in chapter 2, and as a reference point for the tracking algorithm, mapping the found tracks to the actual cow identities, as shown in chapter 5.

The question this thesis aims to answer is: does dairy cow tracking performance of a simple Kalman-filter approach improve when using a re-identification model as a visual descriptor? And how do different parameters and choice of training data for the re-identification model affect the tracking model performance? In practice, a dataset of 4022 images with individually labelled 48 cow identities is presented, based on video material recorded at Luke Maaninka research farm in Spring 2021. In addition, a simple procedure is proposed for the automatic labeling of cow identities based on RFID data and known feeding station locations. Lastly, neural network models for both detection and re-identification are trained on 24 cow individuals from the Maaninka dataset. The models are evaluated against pre-trained models and an appearance-indifferent baseline in tracking context with previously unseen location and new group of 24 cow individuals.

2 Data

For the experiments, video footage and feeding station data were gathered from Luke Maaninka research farm in Spring 2021. There were two distinct departments, each containing 24 cow individuals of Nordic Red and Holstein-Fresian dairy breeds. Only data from the first department was used for training the models described in the experiments, and the second department was preserved as test data, in order to fairly evaluate the models' ability to generalize to a completely new cow population and a different, although similar-looking, barn environment. The time spent visiting the feeding station was recorded using the RFID tags on the ear tags of the cows [35]. As the measurements from the feeding stations were very accurate, the data could be successfully matched with the video frame with the corresponding timestamp, where the locations of the feeding stations in the image were known. As a result, the cows could be reliably identified and located in the image when visiting any feeding station.

2.1 Video data

A total of four cameras were installed on the ceiling of the two departments of the research farm in Maaninka, two in each department. The cameras were of similar model, all being 5 megapixel state-of-the-art surveillance cameras with a fixed 2.8mm lens, capable of recording video at around 20 frames per second with a resolution of 1944×2592. The recording started on 19.2.2021 and the cameras were kept on every day from 7 in the morning until midnight, as cows are typically more active during the day. Altogether hundreds of hours of video were recorded over a period spanning several weeks.

The cows spend a significant part of the day ruminating, when they can either lie down or stand up, but generally do not move around much. Other typical activities include sleeping, visiting the feeding stations or concentrate feed dispenser, socializing with other cows and carefully observing everything happening around the enclosed department. In addition to cows, every now and then caretakers and cleaning robots make an appearance in the video footage. The cameras cover most of the space where the cows have access, but in both departments there are areas not covered by the cameras, as can be seen in figures 2.1 and 2.2. Cows are also occasionally partly or fully occluded in the images, mostly by metal poles on the ceiling and partitions that separate the feeding area from the resting area. Sometimes cows are occluded by other cows, as they crowd around the feeding stations.

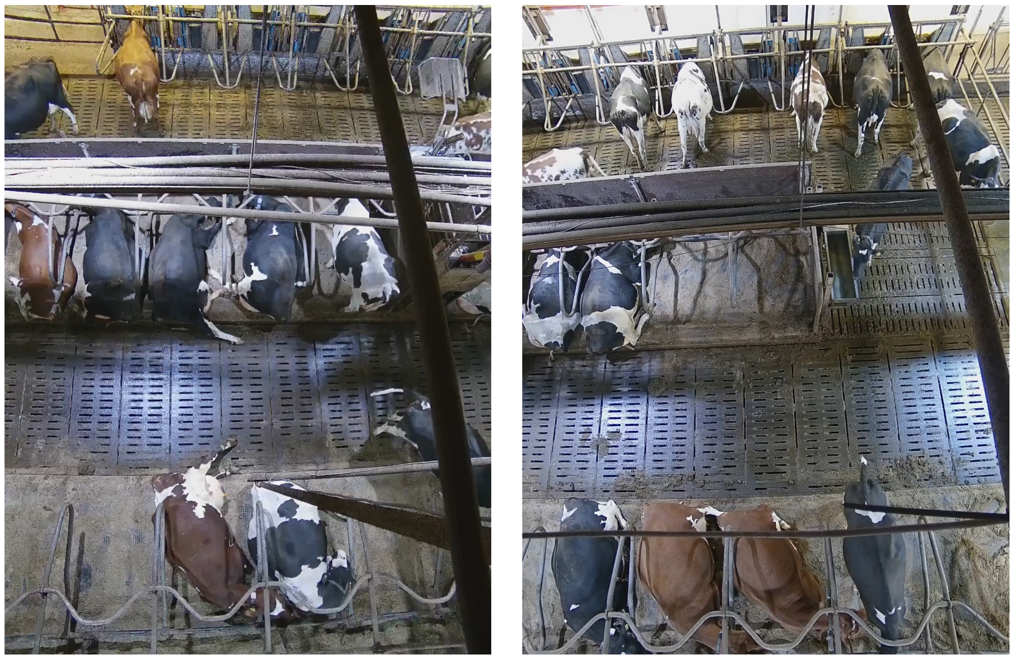

Figure 2.1: Department 1, camera A (left) and camera B (right). A blind spot is left on the left side of camera A, where there is room for the cows to pass through. Feeding stations are located in the top part of the image.

The Maaninka video data was used to create three datasets: one for cow detection, one for cow re-identification and one for the tracking task. For the detection task, a separate training dataset was created by manually annotating bounding boxes over the animals, using a single class for all the different cow identities. As the detection annotation does not require any information on the cow individuals, sampling images infrequently from a longer time span was feasible. The dataset and the evaluation of the detection model on different departments and cameras is described in more detail in section 3.2. For training the re-identification model, an hour of data, with each frame being two seconds apart, were manually annotated using the real cow identities as their labels. The images for each cow individual were then extracted from the images by cropping the bounding boxes, and forming a training set for each specific individual, as presented more thoroughly in section 4.2. Finally, for evaluating the trained detection and re-identification models, a 20 frames-per-second tracking dataset was annotated using footage from department 2, as described in section 5.2. For all of the datasets, a necessary pre-processing step was to apply distortion correction (Figure 2.3). Other pre-processing steps, namely resizing and

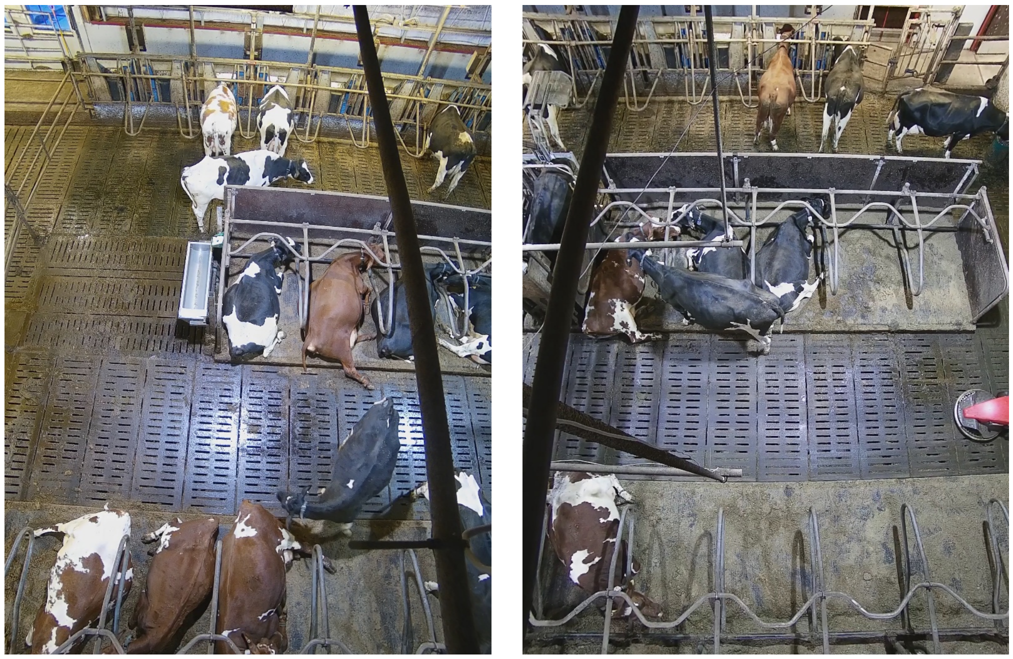

Figure 2.2: Department 2, camera C (left) and camera D (right). The general layout is similar to department 1, although the camera placement differs somewhat; the blind spot is to the right of camera D. An appearance of a cleaning robot can be seen at the right edge of the image from camera D.

augmenting the images, were specific to the model architectures.

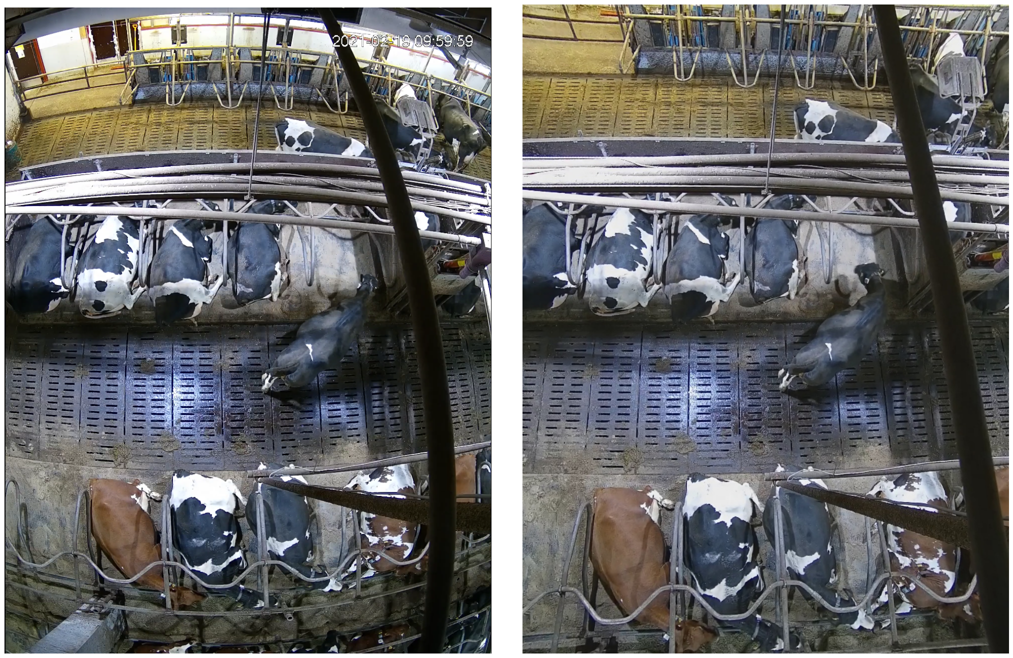

Figure 2.3: Distortion correction applied to video footage from camera A in department 1.

2.2 RFID data from feeding stations

It is very useful to be able to identify a cow when they are moving freely around a barn, for example in order to monitor the feeding time and access to different areas of the barn. Commonly the sensors used for this task are radio-frequency identification (RFID) tags, attached to the cows collar or ear [14]. RFID tags are cheap and scalable, and make it possible to localise the cows with high precision in the parts of the barn area where the RFID readers are installed. The challenge and limitation of the technique is getting the cows to reliably position themselves so that the tag can be read. In Maaninka research farm, the readers are located in the feeding station units, where the tag on the cow's ear is read when animals reach for the trough. This provides information on which cows have visited which feeding stations at a certain time.

With a fixed camera angle, the feeding station locations are known also in the video data and they remain static. By matching the RFID data from the feeding stations at time t with the video data at time t it is possible to assign an identity to a detected cow i visiting a feeding station j at time t, by comparing the bounding box of the detected cow to all the feeding stations, to see if there is sufficient overlap with any of them. The metric used to measure the overlap is the intersection over union score (IoU) of the two bounding boxes:

$$\text{IoU}(i, j) = \frac{|A_i \cap A_j|}{|A_i \cup A_j|} \tag{2.1}$$

The IoU threshold to determine sufficient overlap was chosen by manually assessing an hour's worth of automatic identity annotations generated using the feeding station data and hand-annotated detections from each camera of department 1. A threshold of $c_{iou} = 0.4$ was found to be suitable for the task, as it resulted in zero false negatives or false positives present during the examined time period. A cow i detected in the image was observed to be visiting feeding station j if the intersection over union score of the bounding box of i and the bounding box of j was greater than the threshold. The feeding station locations were annotated by hand (Figure 2.4), based on the layout drawing of the research barn departments (Figure 2.5).

The ground truth obtained using the feeding station data was used in annotation work for the dataset used in the re-identification task. The preliminary identity labels were created by using the feeding station data and the known location of the feeding stations in the image, as explained above. Subsequently, the rest of the labels were annotated by hand (Figure 2.6), resulting in one hour's worth of training data, so that all of the identities

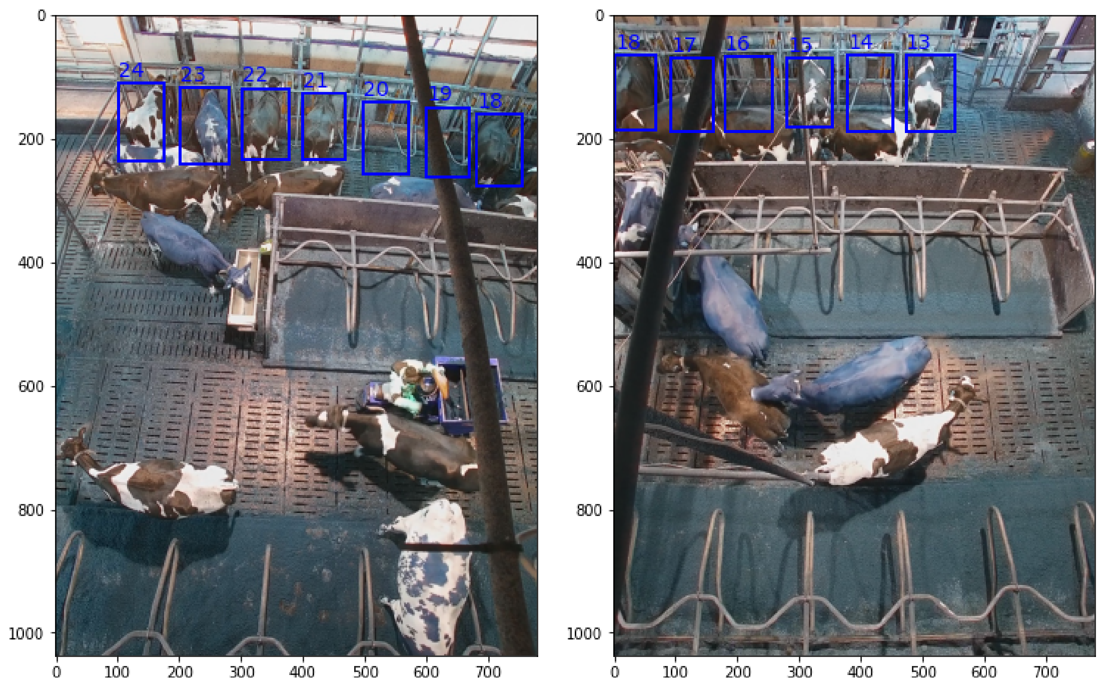

Figure 2.4: Feeding station annotations of department 2, cameras C (left) and D (right).

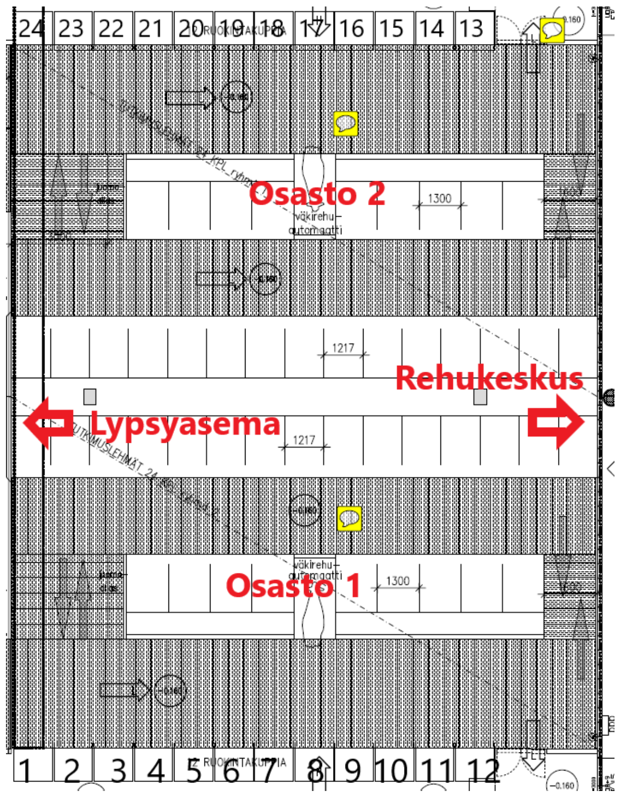

Figure 2.5: The layout drawing of the two departments used for feeding and video data collection on Maaninka research farm. The black numbers correspond to feeding station ids.

in all the images were known. This fully labelled re-identification dataset is described in more detail in section 4.2. In the tracking experiments in section 5.2, the feeding station data is used to match the tracks with true cow identities.

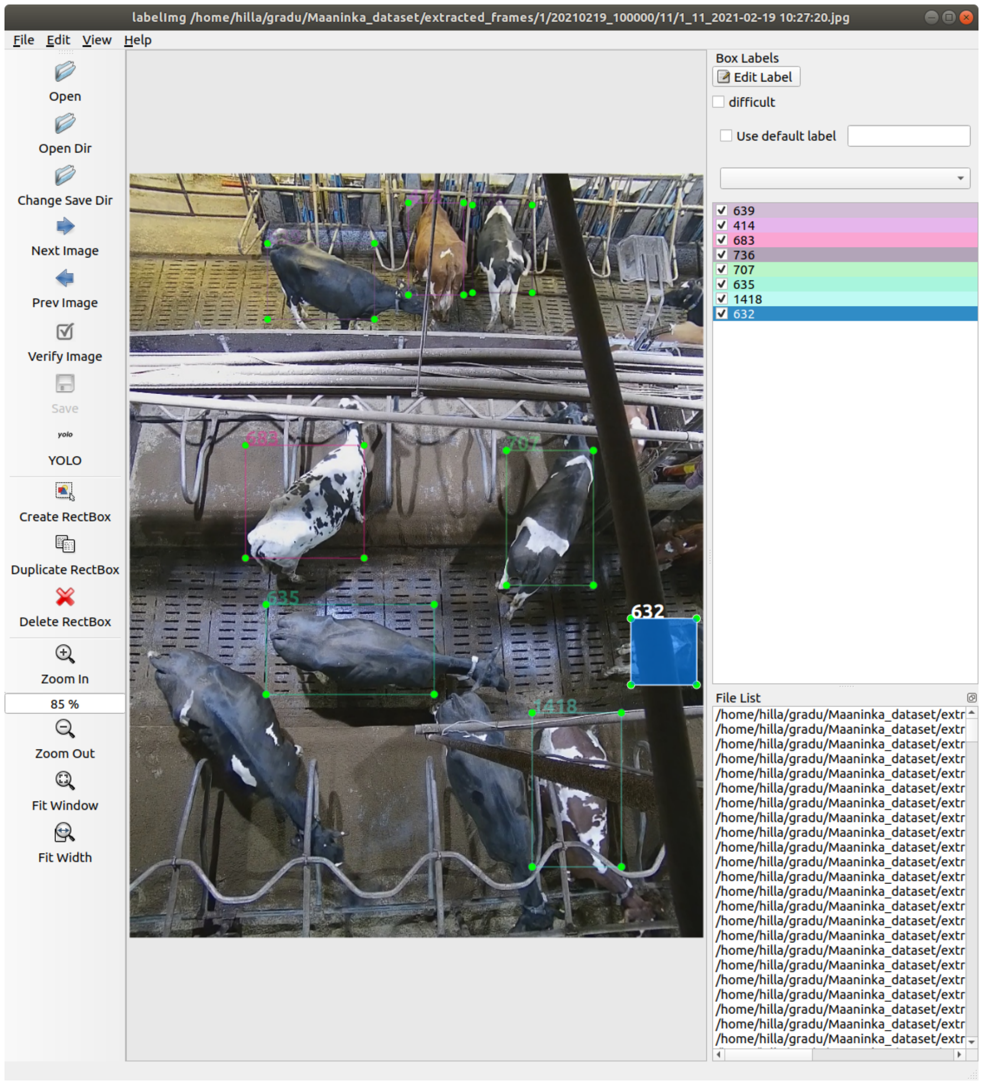

Figure 2.6: Annotating cow identities by hand, using a graphical image annotation tool [41]. The two topmost cows (414 and 736) located at the feeding stations were identified directly using the RFID data.

3 Object detection

Object detection is one of the many tasks that has gained benefit from the technical advances in deep convolutions neural network models [48]. These models have been successfully applied to detecting a wide variety of objects, including cows and other animals [3] [33]. In this chapter, a brief technical background on convolutional neural network models is presented and how they are used in the object detection task. In addition, an object detector model is trained on Maaninka cows, and its performance is evaluated.

3.1 Methods

The goal of this section is to present the building blocks and basic idea of the learning process in deep neural networks in a brief overview, and present the concepts and parameters used later when discussing the experiments of this thesis. From the variety of object detection neural network models, YOLO [31] is chosen as the architecture for the cow detection experiments due to its robustness, speed and competitive performance on similar tasks [48] [2].

3.1.1 Learning in deep neural networks

At the core of a deep learning model is optimization, that is, minimizing or maximizing a loss function $f(\theta)$. *Gradient descent* is a technique for finding some local minimum of the loss function using its derivative $f'(\theta)$, and taking small steps towards the opposite direction of the derivative, until it is no longer possible to decrease $f(\theta)$ in any direction. The size of the step is called the *learning rate*. With multiple inputs, partial derivatives are needed. The vector that contains all the partial derivatives $\frac{\partial}{\partial \theta_i} f(\boldsymbol{\theta})$ is called the *gradient*, denoted $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$. Depending on the loss function, it might very well be that it has multiple local minima, and that gradient descent stops at a local minimum that is not global minimum of $f(\boldsymbol{\theta})$. In deep learning, finding this global minimum is typically not attempted, but aimed for a sufficiently good low value instead [13].

Computing the gradient of the loss function f with respect to the learnable parameters $\boldsymbol{\theta}$ requires two passes through the network: in *forward propagation* the information flows through the network, producing the cost $f(\boldsymbol{\theta})$ from the input $\boldsymbol{x}$. In *back-propagation* [34], information flows backward in the network, resulting in the computed gradient $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$. The backpropagation algorithm allows for an efficient computation of the gradient, which has made the implementation of deep neural networks feasible [48].

As the datasets for training deep learning models are typically very large, it quickly becomes infeasible to compute the gradient for the whole dataset on every iteration of the algorithm. *Stochastic gradient descent* (SGD) estimates the gradient using a small, fixed number of random samples, called a *mini-batch* [13]. After all the samples in the dataset have been processed, one mini-batch at a time, we have trained the model for one *epoch*, denoting one pass through the whole dataset.

A deep learning model is typically trained for hundreds or thousands of epochs, while monitoring the developments of the *training loss*, which is computed on the training dataset used to fit the model, and the *validation loss*, computed on the validation dataset used for evaluating the model during training. This is important in order to avoid both *underfitting*, that is, model not reaching a small enough loss on the training data, and *overfitting*, a situation where the model has learned the training data well but is unable to generalize past that, resulting in a high validation loss compared to the training loss. The latter problem is typically tackled using different regularization techniques, one of the popular methods being *weight decay*, which adds a penalty for larger weights, giving preference for a less complex model.

3.1.2 Convolutional neural networks

Convolutional neural networks (CNN) [24] have been widely successful in various image-related tasks. They are based on the a linear operation called *convolution*, which is mathematically defined as follows:

$$(f * g)(t) = \int f(a)g(t-a)da \tag{3.1}$$

When processing 2D image inputs with CNNs, the operation used is a discrete convolution with a two-dimensional kernel [13]:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.2}$$

As seen in Figure 3.1, the kernel window can be thought of sliding across the image, computing the convolution between input I and kernel K; the elements of the input are added together, weighted by the elements of the kernel. During training, the weights in K are learned, and the convolutional layers can be seen as specializing extracting different features from the image.

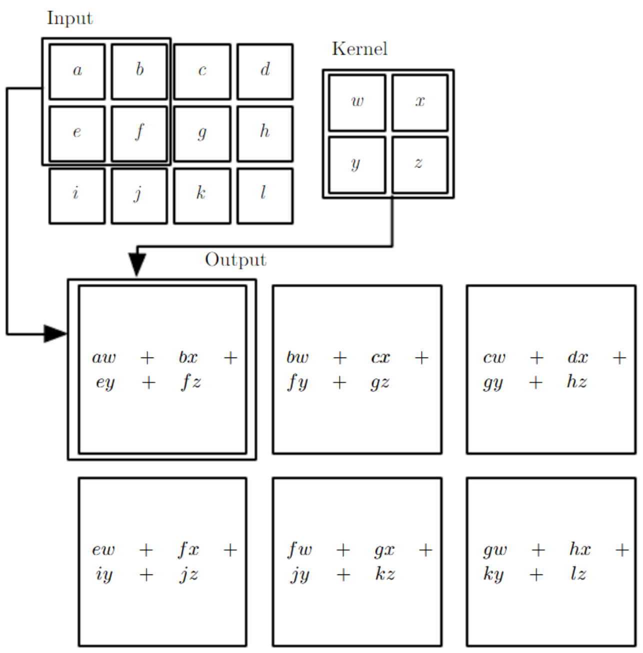

Figure 3.1: Convolution [13].

Convolutional neural network architectures typically employ *pooling* layers in between the convolutional layers (Figure 3.2). Pooling replaces the output of the layer with an aggregate of the nearby outputs, and helps the model learn features that are invariant to small translations in the image [13]. The final fully connected layers provide the final predictions and probabilities for the object classes and locations.

3.1.3 You Only Look Once (YOLO)

You Only Look Once (YOLO) is a fast, robust object detection model that was first proposed by Redmon et al. in 2016 [31]. The main idea of YOLO is to divide the image in $S \times S$ grid cells, and predict class probabilities for each cell (Figure 3.3). The class probabilities are combined with the predicted bounding boxes and and confidence scores:

$$p(\text{class}_i \mid \text{object}) * p(\text{object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = p(\text{class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \tag{3.3}$$

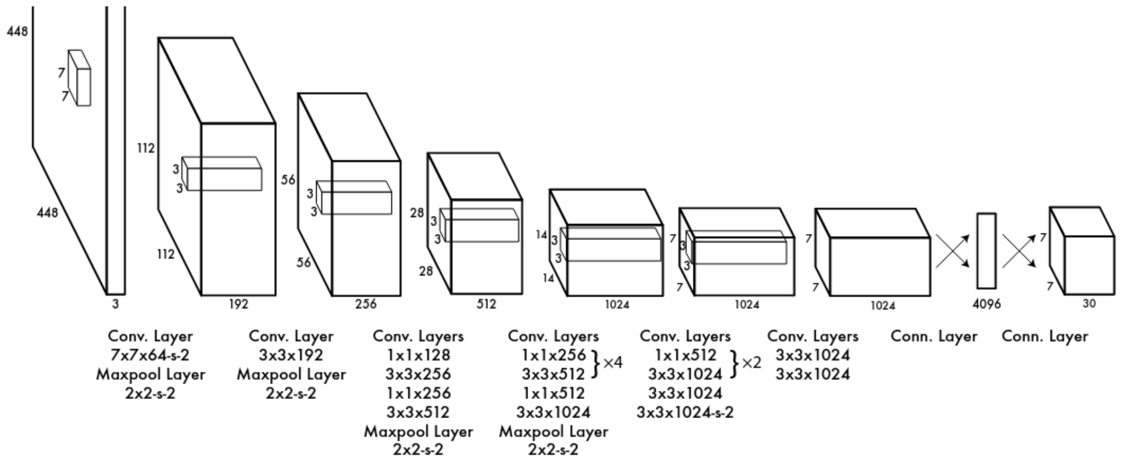

Figure 3.2: Example of a convolutional neural network architecture [31].

in order to get class-specific confidence scores for each bounding box. As YOLO handles the detection problem as a single regression problem, it is much faster than region proposal models [31]. The latter run classifiers on proposed bounding boxes, and the pipelines are slow and hard to optimize. Unlike region proposal methods, it is also able to encode contextual information on the classes in the image, as it sees the whole image during training, and not just the proposed regions. The limitations of YOLO include restricting the number of bounding boxes predicted for each grid cell, which can result in the model facing issues with crowds and small objects [31].

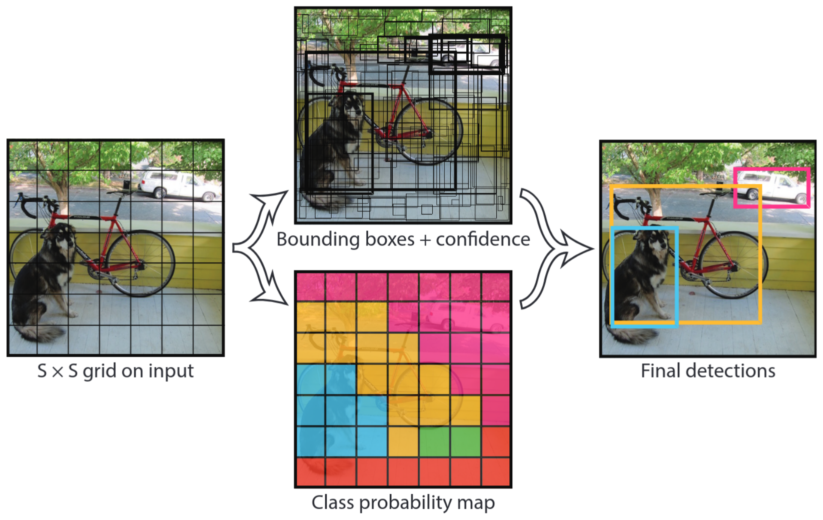

Figure 3.3: YOLO incorporates class probability map together with the bounding box confidence score [31].

The low detection accuracy of small objects has been addressed in later versions, by predicting across different scales in YOLOv3 [32] and increasing the model resolution by using a different backbone architecture in YOLOv4 [9]. Other examples of the many improvements introduced in YOLOv4 include more extensive augmentation techniques and replacing spatial attention mechanism with point-wise attention [9]. In the following experiments, YOLOV4 is used, due to it's speed, training stability, good benchmark performance and proven suitability for a similar task [3].

3.2 Experiments

A detector with YOLOv4 architecture was trained and evaluated on a Maaninka detection dataset, including a search for suitable parameters. The objective of training a detector is to obtain a highly accurate model that can be later used to generate detections for a tracking model. A poor detection model will hamper the performance of the tracking model, so achieving a near-perfect result is desirable.

3.2.1 Dataset

For training the detection model, one frame per minute was extracted, on the date 5.3.2021 between 8:00 and 18:00 from the two cameras of department 1, with a total of 1211 images. The one-minute interval was chosen to include more variety in cow positioning and avoid overfitting. The time range was chosen based on when the cows were most active, which can be seen from the increased number of visits at the feeding stations during that time (Figure 3.4).

For evaluating the detection model, one frame per minute was extracted but from a period of one hour between 17:00 and 18:00 on 19.2.2021, this time also including cameras from department 2 containing different cow individuals, totalling 240 images. Even though the departments have a similar floor plan, the environment is slightly different and the camera angles differ somewhat, as seen previously in figure 2.2. The 120 images from department 1 were used for searching suitable parameters later to be used in the tracking task, whereas the 120 images for department 2 was preserved solely for evaluating the detection model. For both training and evaluation datasets, the bounding boxes of the cows in the frames were annotated by hand (Figure 3.5).

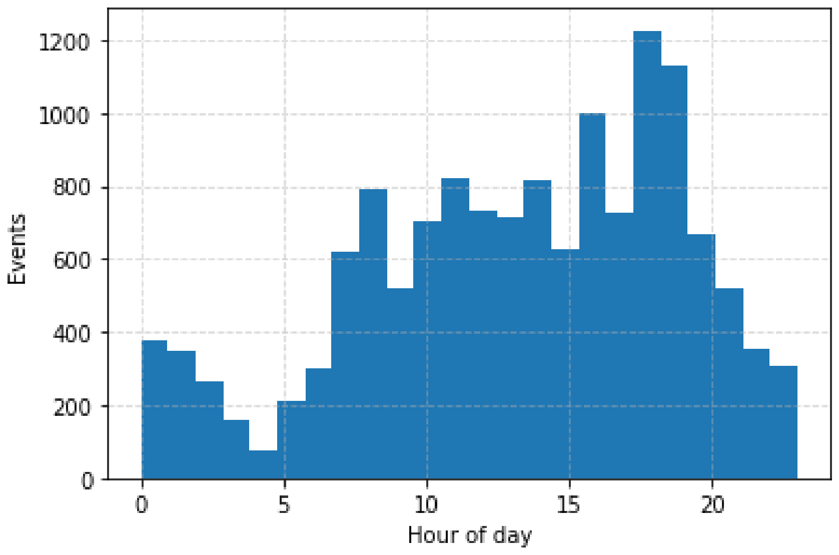

Figure 3.4: Histogram of eating start time events of cows at the feeding stations in Maaninka research farm over several days in Spring 2021.

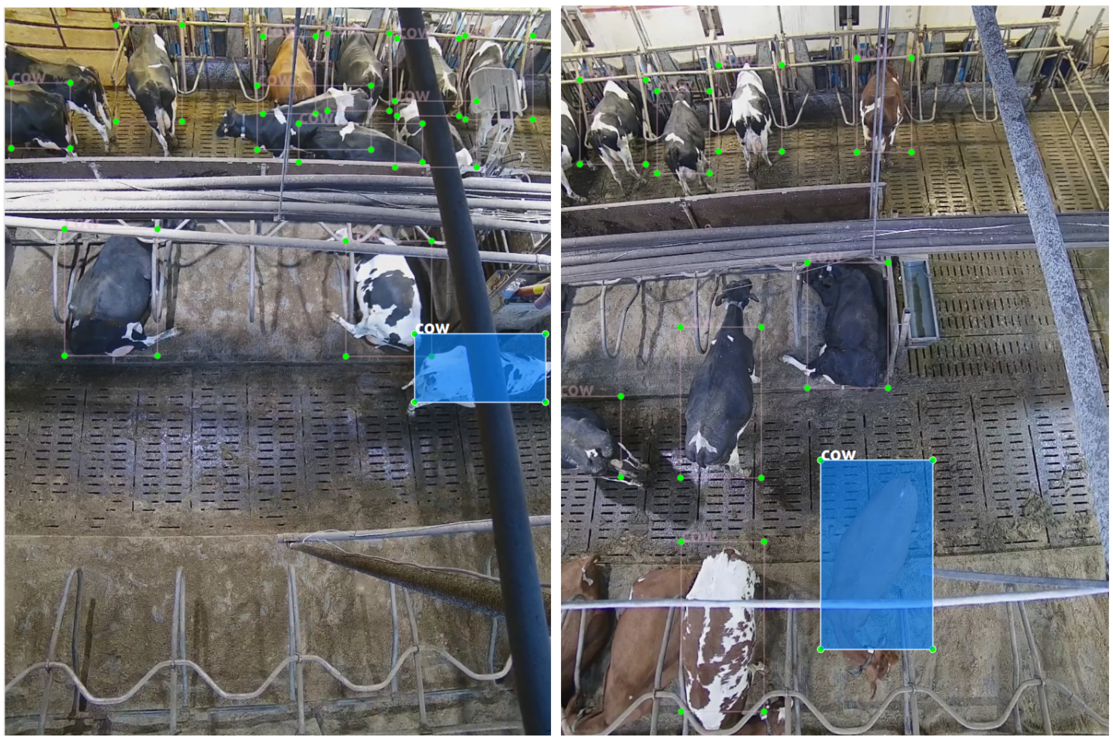

Figure 3.5: Examples of detection ground truth annotations from camera A (left) and camera B (right).

3.2.2 Model training

Before being fed to the network, all of the images were resized to 604 × 604. During training, the images were augmented with saturation, exposure and hue transformations randomly at each training iteration. No angle, mosaic or cutmix augmentations were used, as the locations in the image hold meaning: for example, due to the camera position and perspective, objects on the top part of the image are smaller than ones in the lower parts of the image.

The YOLOv4 model was trained using SGD as an optimizer with a learning rate of 0.001, momentum of 0.949 and weight decay of 0.0005. The model was trained for 40000 iterations using a batch size of 64, so the training over the 1211 images consisted of 2113 epochs. The training was decided to have finished at this point, as the validation metric did not improve, but rather seemed to have a downward facing trend, reaching the best mean average precision (mAP) of 99.40% quite early at iteration 1300 (Figure 3.6). The validation set during the training consisted of the temporally last 20% of the dataset, from both cameras A and B.

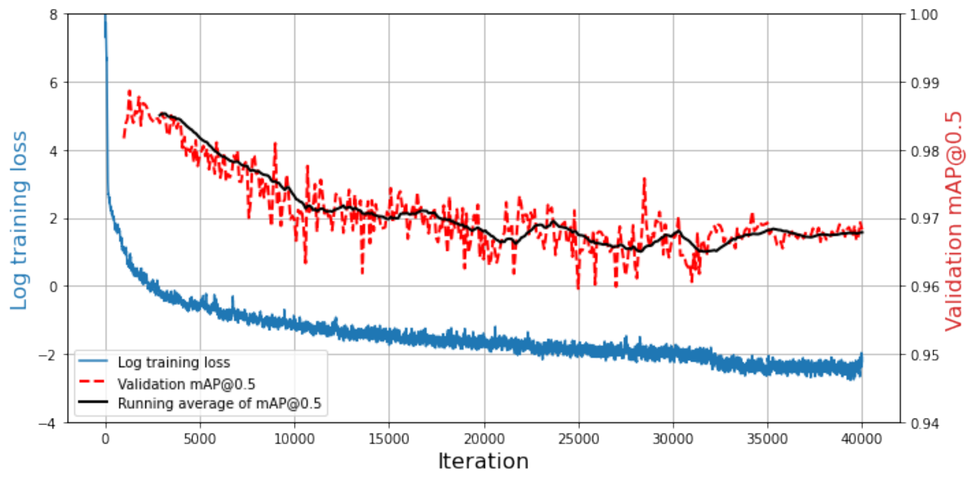

Figure 3.6: Loss and mean average precision during the training of YOLOv4 on Maaninka dataset. Mean average precision at IoU threshold 0.5 was computed on the validation set every 180 iterations after an initial burndown of 1000 iterations.

Mean average precision (mAP) is used to evaluate the performance of a detection model during training. It is computed by processing all detections over all of the images in the order of descending confidence. Each detection is checked against the ground truths for the image, and if the IoU overlap between bounding boxes is greater than a given IoU threshold, the detection is marked as a true positive (TP). If no matching ground truth bounding box can be found in the image, the detection is determined to be a false positive (FP). At each detection i, precision and recall are computed. Precision is calculated using the accumulated true positive and false positive counts:

$$p_i = \frac{TP_i}{(TP_i + FP_i)} \tag{3.4}$$

Recall is calculated by dividing the accumulated true positive counts with the total number of ground truth labels over the dataset N_{gt}:

$$r_i = \frac{TP_i}{(TP_i + FN_i)} = \frac{TP_i}{N_{gt}} \tag{3.5}$$

Average precision (AP) is the area under the precision-recall curve:

$$AP = \int_0^1 p(r), \tag{3.6}$$

where precision p is plotted as a function of recall r. Mean average precision is the mean AP over all of the classes, which in a single-class case is equal to average precision.

In addition to mAP, another metric used to summarize the precision and recall of a model in a single metric is the F1 score, which is the harmonic mean of precision and recall:

$$\mathrm{F1} = \frac{2 \cdot P \cdot R}{(P + R)} \tag{3.7}$$

The F1 score is more typically used in information retrieval, whereas mAP score is the standard evaluation metric for object detection tasks. The mAP score describes the model performance over all of the confidence thresholds and all of the classes, so it gives a reasonable idea of the model stability and whether the training has converged or not. As we want to rule out detections with low confidence, we also want to evaluate the model with different confidence thresholds, for which the F1 score gives us a better idea on the model's precision and recall in practice.

3.2.3 Parameter search

Some parameter search was conducted on a separate parameter validation set of department 1. This validation set was not included in the training data. One of the important tunable parameters in YOLOv4 is the the non-maximum suppression (NMS) threshold. Non-maximum suppression is a post-processing algorithm used to filter overlapping detections: if the overlapping detections have a higher IoU score than the given NMS threshold, the detection with a lower confidence score is discarded. With a suitable NMS threshold, the algorithm improves the precision of the model, as the number of false positive detections of the same object decrease. On the other hand, if the NMS threshold is too low, it can negatively affect the recall of the model if the data does indeed contain crowded scenes and heavily overlapping bounding boxes.

The best AP score of 99.40% was achieved by using NMS threshold of 0.64 (Table 3.1). In their work, Andrew et al. decide to use a lower NMS of 0.28 to avoid filtering out true positives in crowded scenes [2]. Within the range of 0.22–0.64 the difference in performance between the NMS thresholds is quite small on the Maaninka dataset (Figure 3.7). In addition, the detection results on the later iterations and the F1 scores on iteration 1300 all point towards choosing a lower parameter (Table 3.1), so for the evaluation of the model NMS = 0.40 is selected instead of 0.64. For clarity, results for NMS = 0.4 on iteration 1300 are also included in Table 3.1. The differences between some of the confidence thresholds were small, and $t_{nms} = 0.4$ and $t_{conf} = 0.4$ resulted in a F1 score of 98.02, so $t_{conf} = 0.4$ was chosen for further experiments.

Table 3.1: Results of parameter search for NMS (range 0.04–0.96) and confidence thresholds (range 0.05–0.95) computed on the validation set from department 1. The model weights were saved at every 10000 iterations, in addition to the best performing weights (at iteration 1300). For each of these iterations, the NMS and confidence threshold pairs with the highest mAP@0.5 score and the highest F1 score are presented on the table. In the case of a tie, the median value is chosen.

Iteration	t_{nms}	t_{conf}	mAP@0.5 %	F1 score %
1300	0.28	0.40	99.22	98.05
	0.40	0.45	99.28	98.05
	0.64	0.55	**99.40**	97.57
10000	0.28	0.35	97.99	98.00
20000	0.40	0.40	98.27	97.83
30000	0.28	0.45	98.26	**98.15**
	0.40	0.45	98.26	**98.15**
40000	0.28	0.55	98.24	**98.15**
	0.40	0.55	98.24	**98.15**

3.2.4 Results

After choosing the parameters $t_{conf} = 0.4$ and $t_{nms} = 0.4$ based on the validation set of department 1, evaluation metrics were computed also for a detection test set from the cameras of department 2. The complete metrics can be seen on table 3.2, where the performance on each camera is laid out separately. To summarize, the performance of

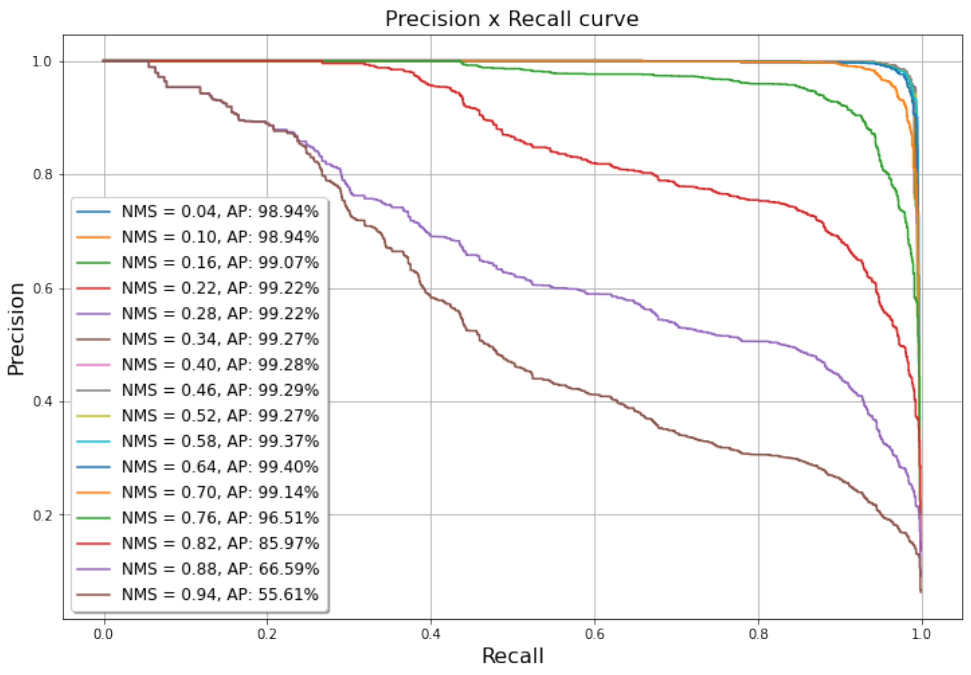

Figure 3.7: Precision-recall curve using different NMS thresholds on the validation set of department 1, with weights from iteration 1300.

the model on cameras A, B and D is excellent, but in camera C there is some room for improvement, namely in terms of recall.

Table 3.2: Detection results with YOLOv4 on datasets of size $n = 60$ from different cameras using a model trained and parameter-tuned on data from cameras A and B. The best performing weights weere used from iteration 1300. Calculated using $t_{nms} = 0.4$ and $t_{conf} = 0.4$ for the F1 score.

Dataset	Camera	mAP@0.5 %	Precision %	Recall %	F1 score %
Evaluation	A	99.10	97.63	98.74	98.18
(dept 1)	B	99.53	98.31	97.32	97.81
Test	C	95.24	97.15	91.70	94.35
(dept 2)	D	99.75	98.94	97.74	98.33

The most common natural reasons for the model to fail were occlusion, crowded scenes and confusing cleaning robots with cows (Figure 3.8). The model also had difficulties to correctly detect partly visible cows near the borders of the images, as these cases were aggravated by inconsistent ground truth labelling caused by the human annotator. In the case of the crowded scenes, some errors can be seen to be caused by the rectangular shape of the bounding boxes, causing a third detection to appear between two cows standing close to each other.

Some of the difference in performances of cameras C and D can explained by the data in camera C containing more labelling mistakes. However, it is not otherwise immediately clear why camera C suffers from a lower recall than the others. Footage from camera D has not been included in the training data either, so the previously unseen environment does not explain the difference. At least on a superficial level, C seems to bear greater resemblance to cameras A and B than D does, in terms of the camera angle and the placement of the occluding structure in the image (Figure 3.9). Differences include camera C having some open space and occluded resting spots. Despite the lower recall on camera C, the resulting mAP@0.5 scores are in the same ballpark as the detectors trained by Andrew et al. [2]. It can be considered a decent result for a detector to be used as part of a tracking model, so the performance is satisfactory for us to use the model in further experiments [2]. It is good to note that detection accuracies both in detection and multi-object tracking benchmarks are typically much lower due to the benchmarks containing significantly more challenging and varying scenes [48].

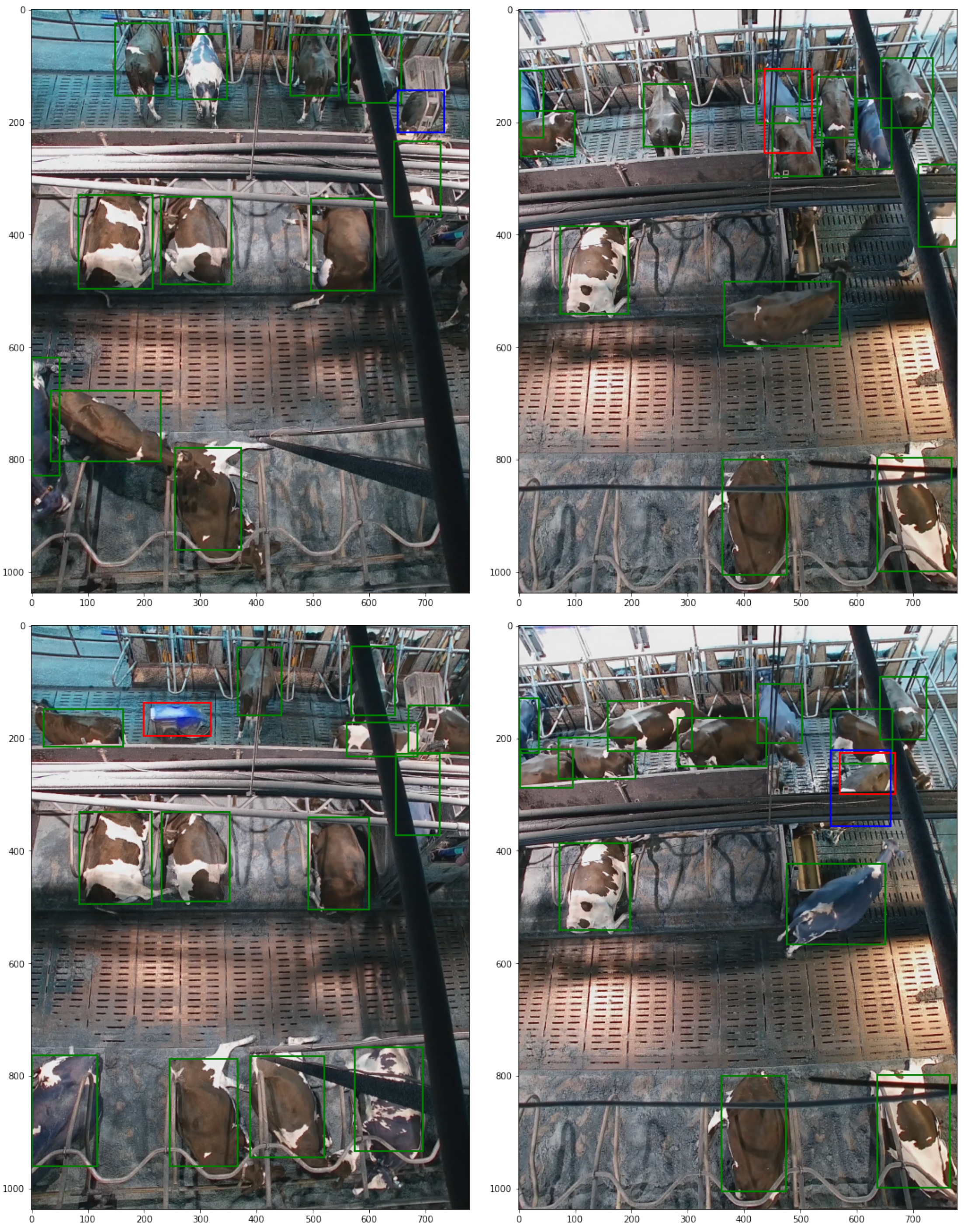

Figure 3.8: Examples of errors in detector predictions in cameras A and B, caused by occlusion (top left), crowded scenes and bounding box shape (top right), cleaning robot (lower left) and inconsistent annotations (lower right). Green denotes true positives, red false positives and blue false negatives.

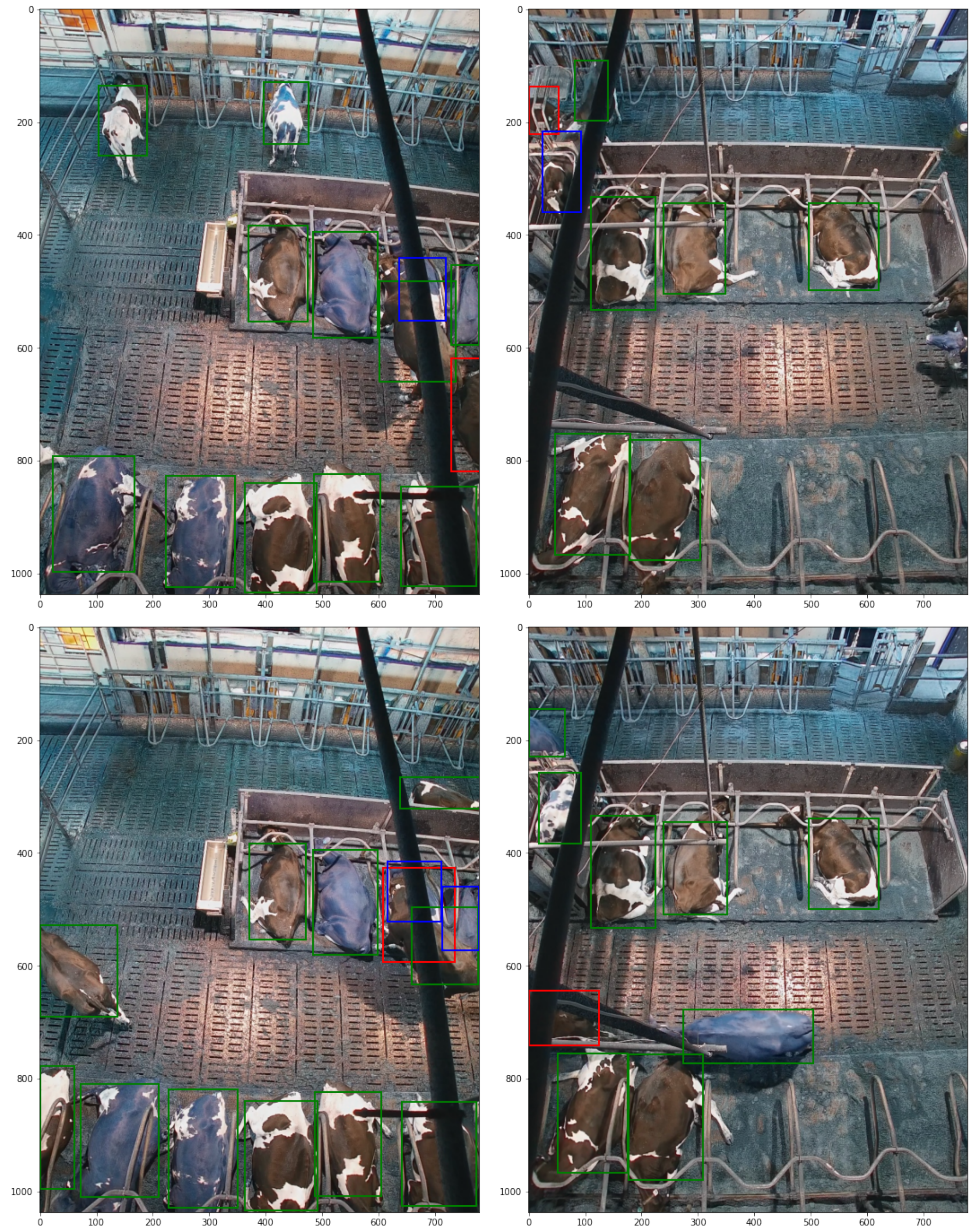

Figure 3.9: Examples of errors in detector predictions in cameras C and D, caused by annotation inconsistencies and occlusion. Green denotes true positives, red false positives and blue false negatives.

4 Re-identification

Even though deep learning based re-identification methods have mostly been developed with human re-identification in mind, they have been found to work also on individuals of other species [36]. In this chapter a type of metric learning method for re-identification is described, as explored in the context of cow re-identification by Andrew et al. on their work on the OpenCows2020 dataset [2]. The presented model is further trained on the Maaninka dataset, and the performance of the model on Maaninka test set is compared to two baselines: a similar model trained on OpenCows2020 data, and one trained on ImageNet only, to assess the model's ability to adapt to a different domain without additional adjustments.

4.1 Methods

Animal re-identification has been somewhat studied on wild animals, where the objective is often estimating the size of the population without capturing individuals. The traditional methods, such as Global Positioning System (GPS) tagging or bird ringing, are often expensive, invasive to the animal and require special expertise to implement [36]. For domestic animals, there are existing supervised methods that can successfully identify individuals [3]. However, supervised training requires a lot of manual annotation work, which does not scale in an industrial setting. Redefining the objective and aiming to re-identify instead, that is, to tell whether the images are of the same individual or not, in contrast to classifying the images to known identity classes, makes it possible to also correctly re-identify individuals that are not present in the training data. This capability to do open-set identification – as opposed to closed-set identification where all of the individuals are known during the training phase – is very useful, as re-training a closed-set identifying network every time a new individual is introduced to the herd would be completely unfeasible.

Modern state-of-the-art methods for object re-identification are based on deep convolutional neural networks. A popular approach is to learn a latent space embedding using a combination of identity loss and triplet loss [46]. If robust enough, this embedding can then also be used to re-identify previously unseen and unlabelled individuals [23]. In this section, a method for re-identification of cows is explained as proposed by Andrew et al. [2] on their work on the OpenCows2020 dataset, for which they use a ResNet [16] based metric learning approach with the triplet loss function. ResNet is a convolutional neural network architecture, where residual mappings are learned instead of trying to fit the the desired underlying mapping directly, allowing for deeper networks and better accuracy.

4.1.1 Metric learning

In contrast to directly learning a classifier model, in metric learning distances between samples in the training data are learned. In practice, this means using a neural network to learn a distance metric in a latent feature space that gives a small distance for two sample images that are of the same individual, and a large distance for two images of different individuals. After learning this feature space, we can project images to it, creating embeddings that can be used e.g. in a k-NN classification. This is necessary, as the embedding space only describes the distances between the samples, and does not produce classification results directly. Ideally, the learned feature space is robust enough to generalize to previously unseen individuals that are not present in the training data.

A metric learning model is typically equipped with either contrastive loss or triplet loss function. When using contrastive loss, distances are updated between pairs of samples, either growing the distance if the sample is of a different class, or reducing it if they are of the same class, and including some weight decay in the loss in order to make the training converge [15]. With triplet loss, the distances are updated using tree data points: the anchor, a positive sample and a negative sample. An early formulation of triplet loss was proposed by Schultz and Joachims [38], but was later made known in the context of deep learning by Schroff et al. as part of their work on a human face recognition system in 2015 [37].

In triplet loss, the key idea is to consider three samples at a time, and minimize the distance of the anchor sample x_i^a and the positive sample x_i^p, while simultaneously enforcing a margin of length α between x_i^p and the negative sample x_i^n, so that the following inequality holds:

$$||f(x_i^a) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^a) - f(x_i^n)||_2^2. \tag{4.1}$$

The positive sample belongs to the same class, i.e. has the same identity as the anchor sample, while the negative sample is drawn from another class, and thus has a different identity. By iterating over the samples, we aim to achieve a situation where all samples belonging to class c reside closer together than any sample outside that class. From this perspective, it is clear how well-suited the k-NN algorithm is for classifying the samples projected in the resulting embedding space, provided that the model has learned relevant features from the training data.

Based on above, the loss function that we aim to minimize is formulated as follows:

$$L_{TL} = d(x_a, x_p) - d(x_a, x_n) + \alpha = ||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha, \tag{4.2}$$

where d denotes the distance between the samples. However, when taking into account the whole training dataset, there are a lot of samples that, after a short period of training, easily satisfy our inequality requirement. In other words, comparing these samples provides little new information that the model needs in order to improve. To make the learning more effective and the model converge faster, it has been discovered to be a good practice to focus on the samples that are more difficult for the model to get right [37]. This is called hard negative mining; when choosing the samples for training, only negative samples that violate the margin are chosen to the mini-batch.

A problem with the triplet loss is that the margin between the positive and the negative sample can be satisfied at any distance from the anchor sample (Figure 4.1). To combat this, Andrew et al. use a variation called reciprocal triplet loss (RTL), introduced by Masullo et al. in 2019 [27]:

$$L_{RTL} = d(x_a, x_p) + \frac{1}{d(x_a, x_n)} \tag{4.3}$$

Using RTL, we avoid the margin problem of triplet loss, as the distances between the anchor sample and the positive and negative samples are continuously optimized [27].

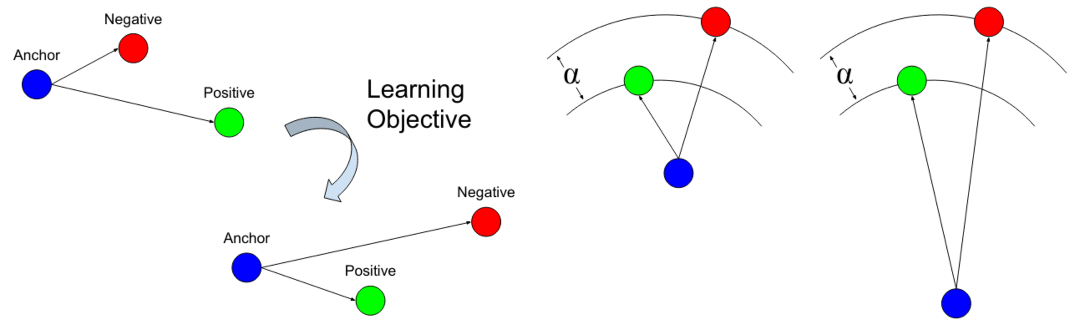

Figure 4.1: Learning with triplet loss. The margin problem is depicted on the right: the margin α between positive and negative samples can be satisfied at any distance from the anchor sample [2].

Ultimately, the loss Andrew et al. employ is a combination of RTL and fully supervised softmax cross entropy loss [23]. Softmax loss provides the supervised loss based on the classes of the samples, allowing the model to efficiently utilize all the images in the mini-batch during learning:

$$L_{softmax} = -log\left(\frac{e^{x_{class}}}{\sum_i e^{x_i}}\right), \tag{4.4}$$

whereas the triplet loss generates a more discriminative embedding space, enabling the model to correctly predict also previously unseen classes [23].

4.1.2 K-Nearest Neighbors

After the latent feature space has been learned, the k-Nearest Neighbors algorithm is applied, as our end goal is to classify images based on their appearance. Predicting with k-NN is straightforward: the predicted class of the new sample is simply the majority vote of the classes of its k closest neighbors. The distance is computed within the learned embedding space, so a closer proximity between points in the embedding space correspond to greater visual similarity between the individuals. Predicting a class for a new sample is demonstrated in figure 4.2, where class A represents the majority in 3 of the closest neighbors, so the predicted class for the sample is A. Precision for the model is determined by the fraction of neighbors that belong to the correct class. In this case, if A was the real class of the sample, the precision of the prediction would be $\frac{2}{3}$.

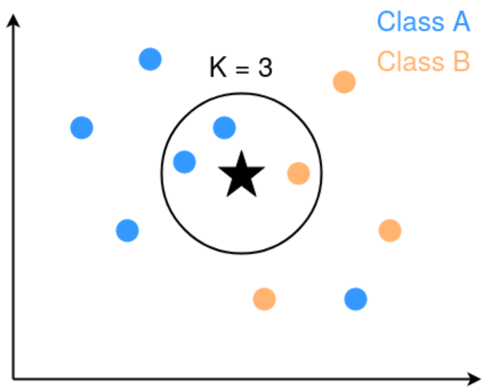

Figure 4.2: Predicting the class of a new sample using a k-NN classifier with $k = 3$.

As mentioned earlier in section 2.1, to fairly evaluate a model's classification performance on video data with high temporal correlation, some attention must be shown to the choice of the images that make up the source embedding (the neighbors with known labels), and the target embedding (unseen test samples without labels) of the classifier. If the source and target images are visually similar due to having strong temporal correlation, we do not gain real insight on how well the model generalizes and whether it has truly learned relevant, robust features that are needed in a real-life setting; the visually similar images taken seconds apart will reside close to each other in the embedding space, resulting in high accuracy. This problem is discussed in more detail in section 4.2.

4.2 Experiments

An architecture proposed by Andrew et al. [2] was re-trained using the dataset from Maaninka research farm. The results were compared against two pre-trained models using the same architecture but different training datasets: ImageNet and OpenCows2020.

4.2.1 Dataset

For training the re-identification model, frames were extracted from the video files on 19.02.2021 between 10:00 and 11:00 AM in department 1 every two seconds, taking into account the video's frame rate. The frequent rate was chosen in order to make the annotation work easier, as it is otherwise difficult to tell apart similar-looking cows that switch places in a few seconds' time. Altogether 1 hour of video was annotated so that all the identities of the cows were manually labelled, from which a training dataset of 3602 images of the 24 cow individuals was constructed, resulting in over 40K bounding boxes with annotated ground truth identities. An example of the resulting gallery of bounding boxes from one frame can be seen in figure 4.3.

An important thing to note when using video as a data source is that the changes between consecutive frames are very small. Due to this temporal correlation, one can easily end up obtaining a large number of images of an individual that are highly similar, even if the images are technically from a different timestamp. As cows tend to walk around slowly and have a habit of standing still for long periods while ruminating, this effect is amplified. Therefore, in addition to taking this into account when forming the training set, it would be important to source the validation data, and especially the test data, from a different day or different setting to make a fairer evaluation of the model, and not rely on separating the test set from the same pool of images that is used for training. This is necessary in order to show that the model has actually learned relevant features, and not simply been overfit, and that it can generalize past the easier cases where the images are highly similar to each other.

In the context of tracking, the images that constitute the source embedding and the

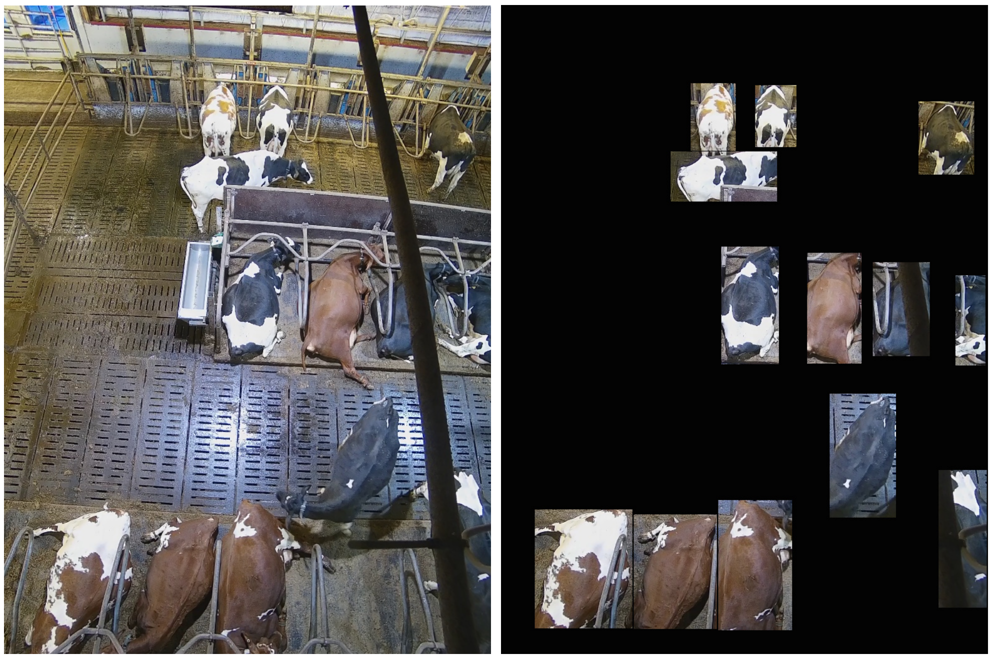

Figure 4.3: For the re-identification task, cow identities were individually annotated and the cropped images of the individuals were extracted from the frames.

target embedding really are temporally connected. One can argue that specifically in this context it is not as important to evaluate re-identification models with test data that has low temporal correlation to the source embedding. In contrast, it is crucial when using the re-identification model as a classifier over a longer time span, as it requires a truly robust model that can generalize past the easier cases. The concept is good to keep in mind as the results of the experiments using different source and target embeddings are discussed.

To reduce the effect of this temporal correlation when evaluating the re-identification model, only a portion of the images were used in order to create a less biased dataset for the task: the cropped images were manually processed in a sequential order, and only retained if the bounding box size changed or the image were deemed to be dissimilar enough. Mean Squared Error and Structural Similarity Index were used as supporting metrics describing visual similarity, with higher dissimilarity indicating movement withing the bounding box, leaving a total of 5043 images after the processing (Figure 4.4). After filtering the training set, 10 of the remaining cropped images were randomly chosen from each individual to form the test set. This set was kept hidden from the model during training. Additionally, 20 images were randomly chosen to form the validation set that was used to assess model convergence during training.

As an example of temporal correlation, in figure 4.5 some samples are shown from the

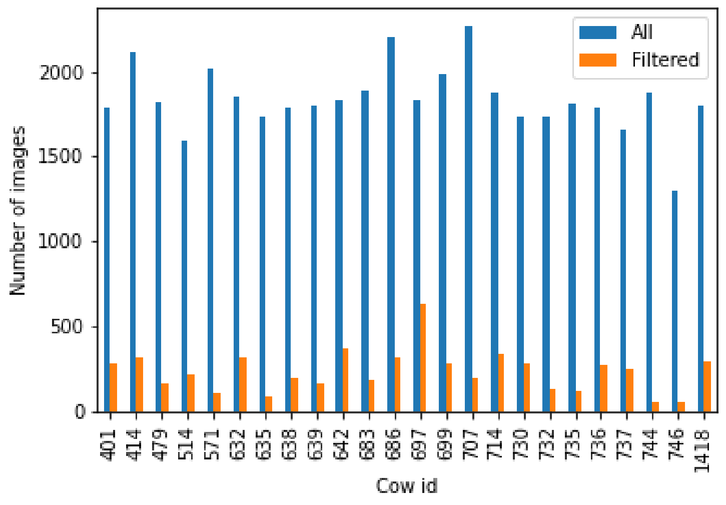

Figure 4.4: Number of images for each cow individual before and after filtering highly similar images from the dataset.

OpenCows2020 dataset [1], where an individual has very similar counterparts in the training set and the test set. OpenCows2020 is a cow dataset collected by Andrew et al., which contains 4736 images of 46 cow individuals in total, including both indoor and outdoor imagery [2]. The similarities in angle, lighting and background between the training and test images suggest high temporal correlation. This is to be expected, as the OpenCows2020 images were randomly split intro train and test data [2], even though the temporal correlation between the example images could not be verified as the dataset did not contain timestamps for the images. Nevertheless, having highly similar images used as the source and target embeddings while evaluating the model is likely to produce optimistic results on model performance. It is good to note that even though hold-out individuals were used in order to evaluate the model on previously unseen individuals, the source embeddings that were used in the evaluation consist of the training data of these hold-out individuals. If there is high temporal correlation between the source and the target data, it will introduce a bias in the evaluation of the model, regardless of the hold-out status of the individual.

4.2.2 Model training

For training the Maaninka re-identification model, the cropped images of the individual cows were resized to 224 × 224 before being fed to the network. Black padding was used in order to preserve the aspect ratio of the image. For the training, SGD was used as an

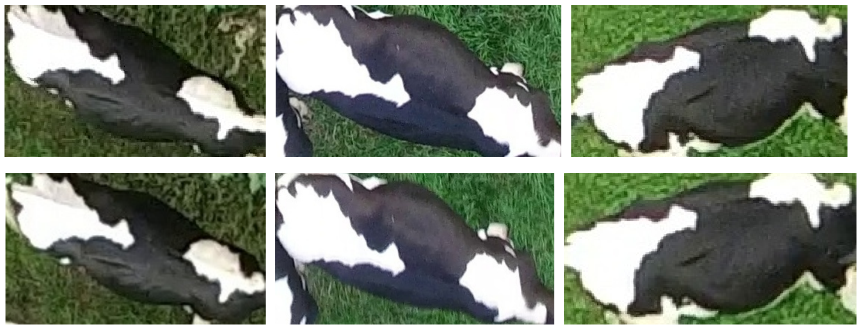

Figure 4.5: Pairs of example images from OpenCows2020 dataset of individual with id 3, where the upper images are from the training set and the lower images are from the test set.

optimizer with a learning rate of 0.001 and weight decay of 0.0001. The embedding size used was 128, as in previous research it has been found to suffice when training a model for a similar task [37] [2]. The model was trained for 150500 iterations (500 epochs), achieving the best validation accuracy of 93.54 at iteration 87310 (Figure 4.6).

The first model to be compared against Maaninka model is based on ImageNet [12], a large-scale dataset with millions of images and thousands of different classes, typically used for benchmarking and pre-training neural networks. This way, useful general-purpose features are learned by training the model on a huge dataset with varying objects, so that only fine-tuning is needed when training a custom detector, reducing training time needed significantly [48]. It can also be used as a crude baseline in re-identification model evaluation, by removing the last classification layer of the network trained on ImageNet and using the resulting output as embeddings.

The second model to be used in the experiments is based on the previously introduced OpenCows dataset. As the model is trained on a similar domain, we expect the results to improve compared to ImageNet weights. To get an idea of the weights and their suitability for the re-identification task in Maaninka data, a t-distributed Stochastic Neighbor Embedding (t-SNE) [26] visualization on the Maaninka training data produced by using an OpenCows-trained encoder is computed (Figure 4.7). T-SNE is a popular technique for visualizing structures of high-dimensional data. It uses the Student-t distribution to compute the distance between two points in the lower-dimensional space it creates, and for this reason the distances produced by the algorithm are not guaranteed to preserve the local structure of the original data [26]. Thus is cannot be used to produce reliable distance metrics in itself. However, it is a great exploration tool, and as the visualization on OpenCows does show some of the cow individuals clustered close together, it suggests at least some ability to tell Maaninka cow individuals apart.

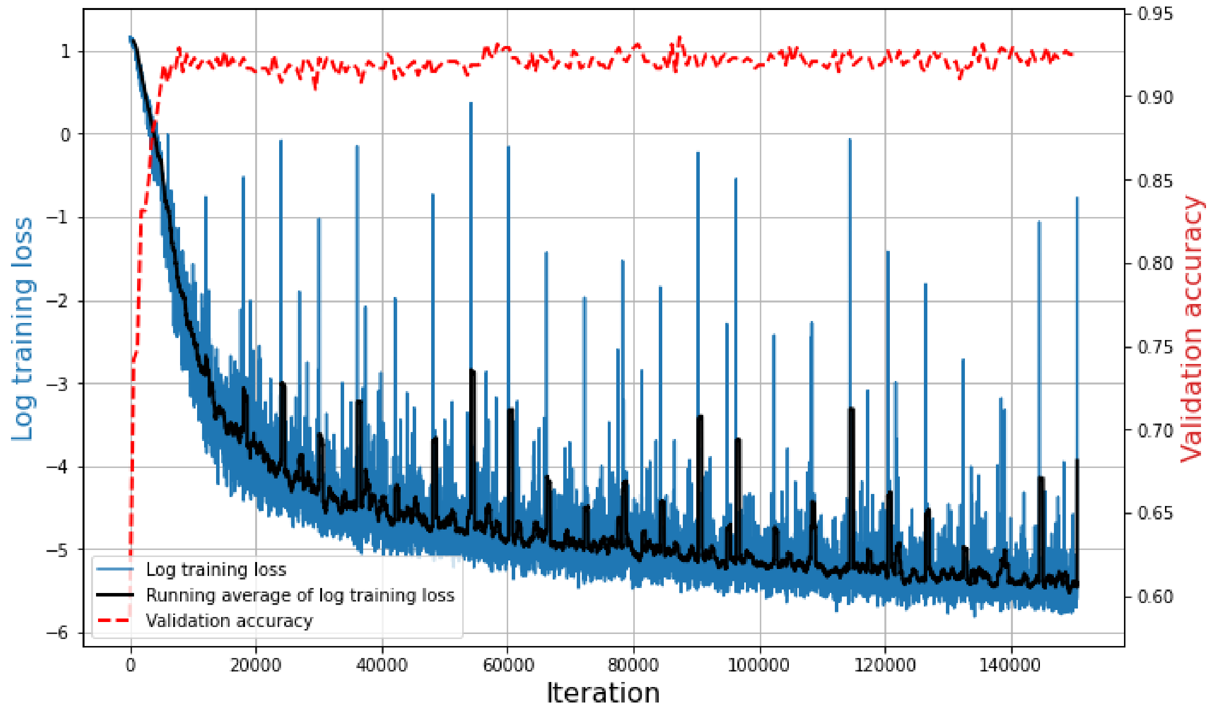

Figure 4.6: Training loss and validation accuracy of the metric learning model trained on cow identities of Maaninka.

4.2.3 Results

Accuracy and precision were computed using the same test set for all three models. The results of the Maaninka model were computed using the weights that achieved the best validation accuracy during training. Prediction of the class was based on taking the vote from $k = 5$ nearest neighbors and using the majority as as the predicted class. Accuracy was calculated by comparing these predictions to the ground truth labels. Precision is the percentage of of the k nearest neighbors in the prediction matching the ground truth label.

The results in table 4.1 show that model performance with ImageNet weights is exactly would be expected if we predicted a label from the 24 individuals randomly: $p(x) = \frac{1}{24} \approx 0.0417$. With a closer look we discover the reason for this is that with ImageNet, the label 635 is predicted for all of the samples and all cows.

While the model trained with OpenCows2020 dataset shows substantial improvement to the ImageNet model, the accuracy is not high enough as-is for standalone re-identification. As could be expected, the supervised model trained on the same cows in Maaninka attained the the best performance.

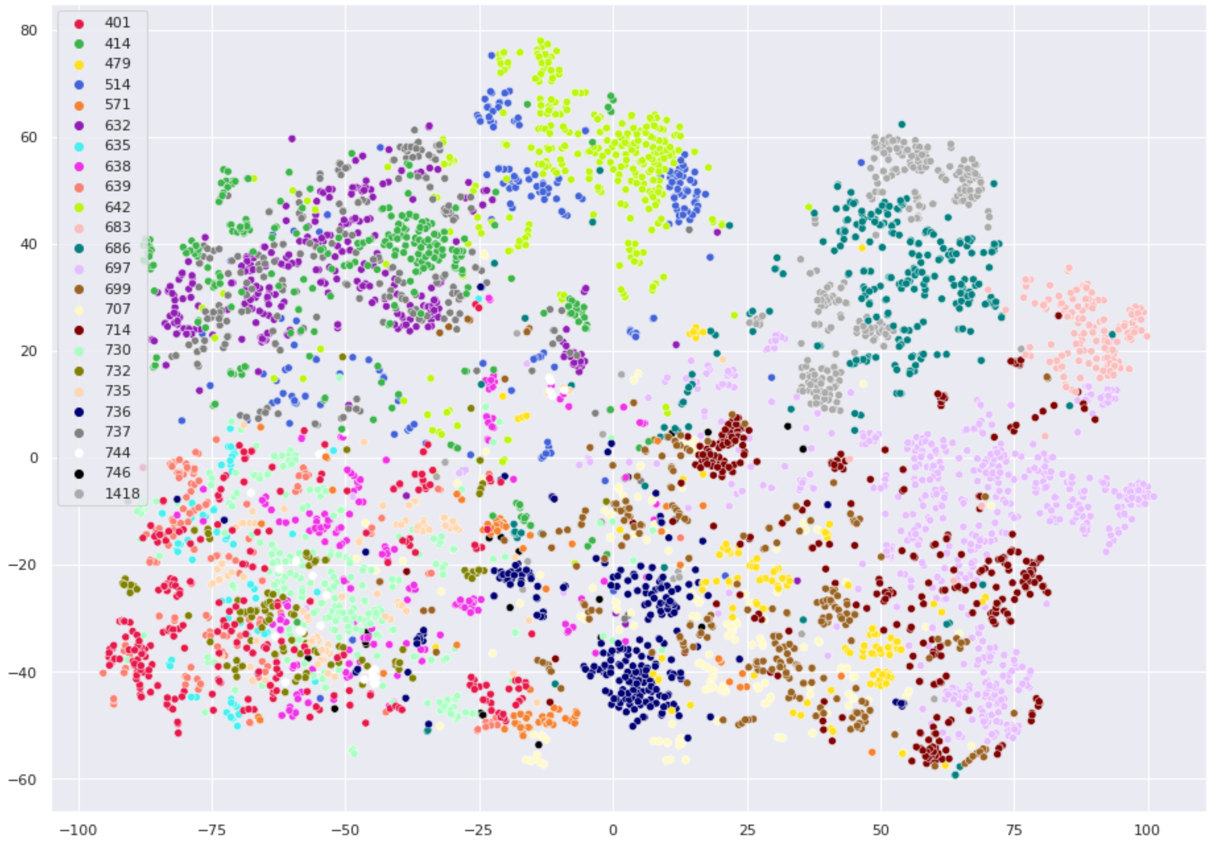

Figure 4.7: A t-SNE visualization of the Maaninka source image embeddings used for the KNN classfication, with the encoder trained on OpenCows2020. The colours represent different cow identities.

Table 4.1: KNN classification results with Maaninka test set using the learned embeddings.

Encoder training dataset	Accuracy$_{K=5}$	Precision$_{K=5}$
ImageNet	4.17%	4.17%
OpenCows2020	42.08%	36.67%
Maaninka	92.92%	89.42%

From all the individual cows, 639 had the lowest test accuracy of 70% (Figure 4.8) on the Maaninka-trained model. The individual was confused with other Holstein-Fresians 635, 401 and 744, all of which have a black coat with either very small or no markings (Figure 4.9). Here we see that identifying cows based exclusively on visual appearance is inherently quite a difficult task: distinguishing 639 and 635 based on a single image is hard even for a human observer. In a real-life setting we face this challenge in multitudes, as the group size in industrial barn environments can be much larger than the 24 individuals used in these experiments.

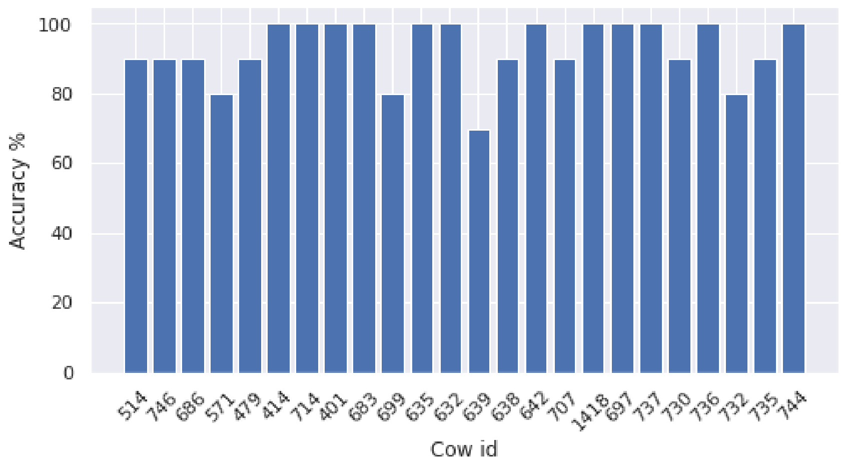

Figure 4.8: Test accuracy of the different cow individuals in Maaninka model.

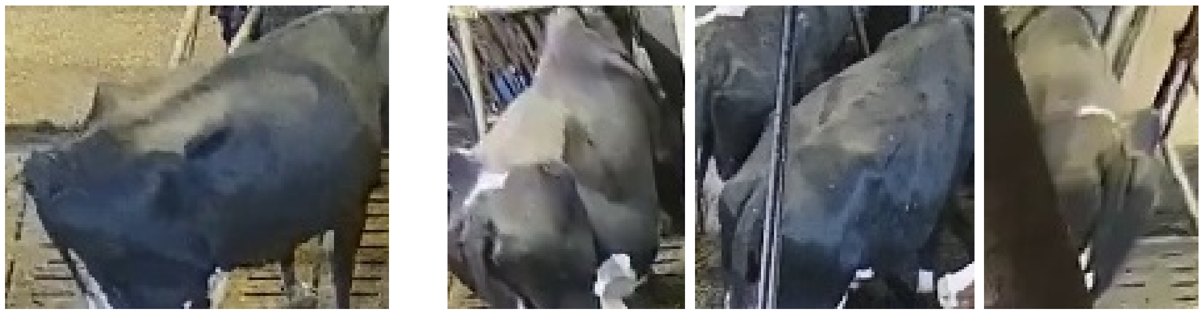

Figure 4.9: Confusing individuals causing false predictions for cow 639 (leftmost), example images from Maaninka test set: 401, 635 and 744 (left to right).

Figure 4.10 shows Maaninka training data projected on the Maaninka-trained embedding space. These source images are the same images from Maaninka that were used in training the encoder, and so the visualization also demonstrates the objective of the training with triplet loss: to make all the instances belonging to the same class closer to each other than to instances from other classes. A visualization of the test data, projected on the same embedding space, can be seen in Figure 4.11.

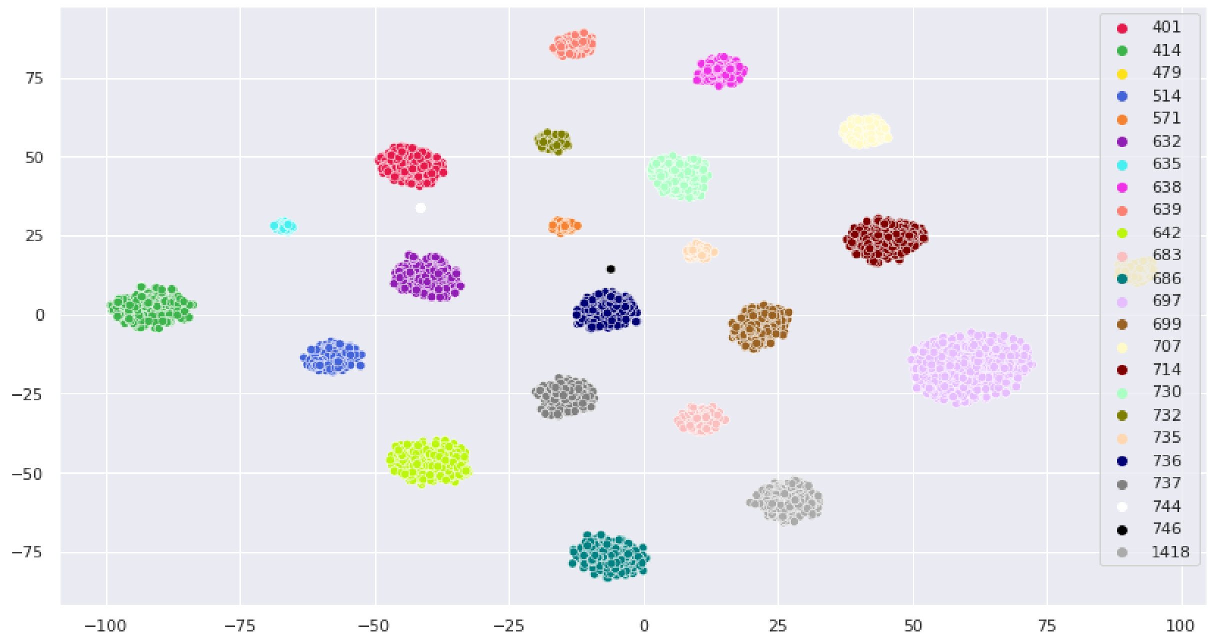

Figure 4.10: A t-SNE visualization of the Maaninka source image embeddings used for the KNN classification.

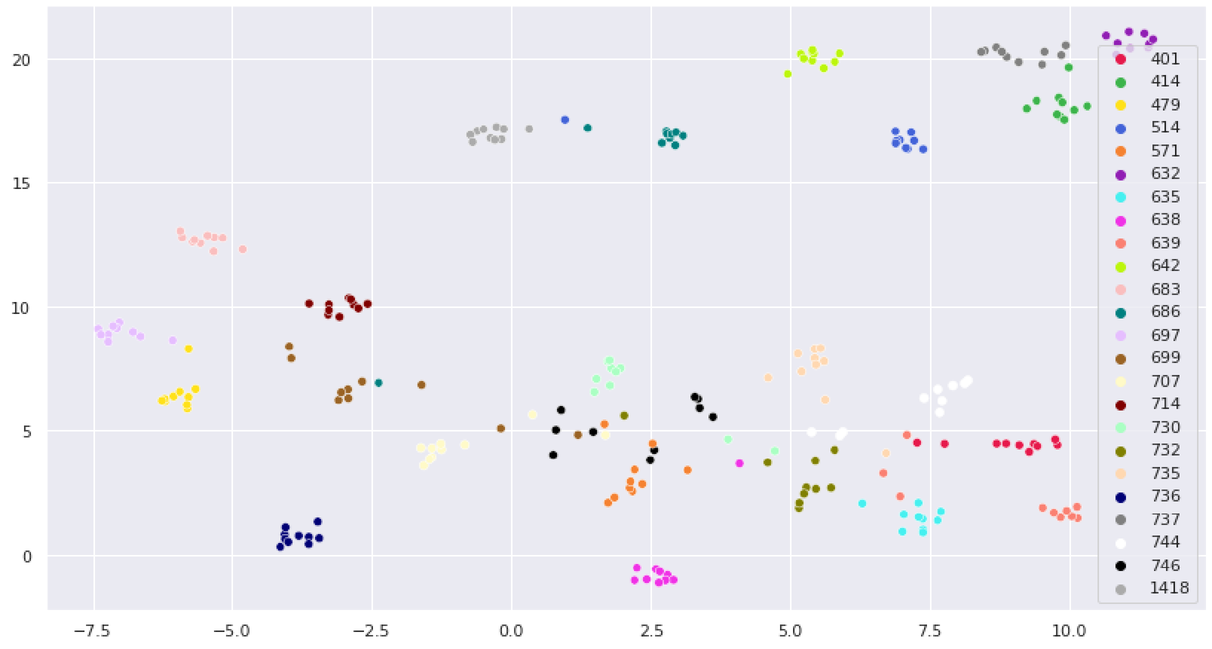

Figure 4.11: A t-SNE visualization of the image embeddings in the test set of Maaninka, with the encoder trained on Maaninka training set.

5 Multi-object tracking

Multi-object tracking has been widely studied on different objects of interest: sports players, animals, vehicles and most notably pedestrians, and the interest in this field of research has been further increasing as there is great commercial potential in practical applications [25]. In this chapter, different potential algorithms for cow tracking are discussed, mostly focusing on simple and fast Kalman filter based object trackers. In the final experiments, the differently trained re-identification models presented in chapter 4 are evaluated as a part of a tracking model on a new video dataset.

5.1 Methods

To choose a suitable model for cow tracking, a set of models that perform well on relevant available benchmarks are to be considered. In multi-object tracking, the object of interest is typically a pedestrian walking on the street, and the contents of the benchmarking datasets reflect that [25]. One popular and widely referenced dataset is MOT16 [28], which consists of 14 sequences of video, where pedestrians have been annotated frame-by-frame with a given tracking identity, so that evaluating tracking performance on them is possible. An example of MOT16 data along with the ground truth annotations of the pedestrians can be seen in figure 5.1. To contextualise the discussion in this section regarding the model choices in this thesis, some popular state-of-the-art models along with their performance in MOT16 dataset are shown in table 5.1, along with explanations for the common metrics used for benchmarking. Typically this collection of metrics is used for evaluating the performance of multi-object tracking models, as capturing all relevant aspects of the performance in just one or two metrics is difficult due to the complexity and temporal nature of the tracking problem.

A common approach for multi-object tracking is tracking-by-detection, in which the bounding boxes for the objects of interest are first obtained by using a detection model, typically a neural network. This leaves us with a problem of tracking the detections through the video frames, which means assigning the detections to the correct tracks. In this section, a popular baseline model SORT [8] is presented, and shown how it uses Kalman filtering and the Hungarian algorithm to assign the detections to the tracks using their estimated

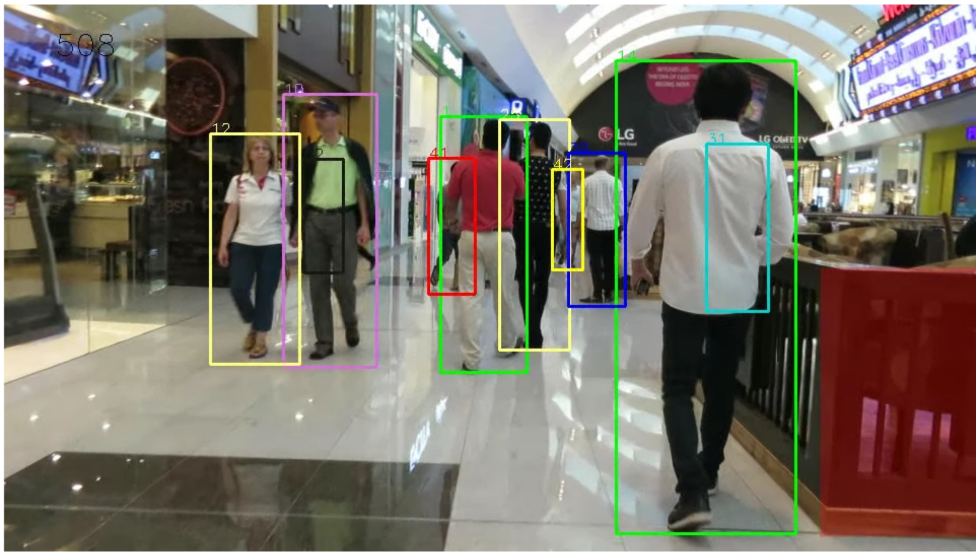

Figure 5.1: Sample image from one of the MOT16 training sequences with ground truth annotations marked [28].

locations and velocity. While SORT is mainly a decent baseline method in terms of accuracy, its performance in terms of prediction speed is of a different order compared to more complicated models; through a selection of optimizations, SORT can be implemented to run on the MOT16 benchmark at a average speed of 687 frames per second [30]. While recurrent neural network models like GRTU [43] achieve good accuracy, they are very slow for practical purposes, as seen in the comparison of table 5.1.

Table 5.1: MOT16 benchmarks on selected state-of-the-art multi-object tracking models. 1) Multi-object tracking accuracy (MOTA): Summary of overall tracking accuracy in terms of false positives, false negatives and identity switches. 2) Mostly tracked (MT): Percentage of ground-truth tracks that have the same predicted tracking label for at least 80% of their life span 3) Mostly lost (ML): Percentage of ground-truth tracks that are tracked for at most 20% of their life span. 4) Identity switches (ID): Number of times the reported identity of a ground-truth tracletk changes. 5) Fragmentation (FM): Number of times a track is interrupted by a missing detection. 6) Method speed (Hz): Predicted frames per second

Model	MOTA$^{1)}$ ↑	MT$^{2)}$ ↑	ML$^{3)}$ ↓	ID$^{4)}$ ↓	FM$^{5)}$ ↓	Hz$^{6)}$ ↑
SORT [8]	59.8	25.4%	22.7%	1423	1835	**59.5**
DeepSORT[45]	61.4	32.8%	18.2%	781	2008	17.4
Fair [47]	69.3	40.3%	16.7%	815	2399	26.0
FairMOT_V2 [47]	75.7	48.1%	14.4%	621	934	25.4
GRTU [43]	76.5	**51.5%**	17%	**584**	**597**	0.3
TLR [42]	**76.6**	47.8%	**13.3%**	979	1709	15.9

Many of the state of the art tracking models use some kind of re-identification method for comparing the appearances of the detections and the appearances of the tracks. This appearance information is also what DeepSORT [45], an improved successor of SORT, uses in solving the assignment problem. Even though DeepSORT is not the highest-performing tracking model on the benchmark datasets, it a relatively simple method, which lends itself to be an useful base for the experiments presented in section 5.2, where different re-identification models are plugged in to DeepSORT. Compared to joint detection and re-identification schemes with superior accuracy, such as FairMOT [47] and TLR [42], this type of experimentation is very lightweight as in DeepSORT the detection and re-identification models can be trained separately, while not critically falling behind in prediction speed either.

5.1.1 Simple Online and Realtime Tracking (SORT)

A simple yet effective tracking algorithm SORT was proposed by Bewley et al. [8] in 2016. It is a detection-based method that uses Kalman filtering to estimate the movement of the objects in the detected bounding boxes frame-by-frame, and Hungarian algorithm to assign the bounding boxes to the most probable tracks.

The algorithm takes detections as an input, so any detection model can be plugged in to be used with SORT. The cost matrix used by the Hungarian algorithm for associating the detections and tracks together is computed by simply calculating the intersection over union of the bounding boxes of the detections and the tracks estimated by the Kalman filter. After the assignment of detections and tracks has been computed, an additional filtering is performed on the matchings, where the matchings that have an IoU score lower than a pre-defined threshold are removed. In addition, if there are tracks that we have not been able to assign to a detection for a given number of frames, they are removed. Similarly, if there are new detections that cannot be matched to any track, a new track is created.

Kalman filtering

Developed by R. E. Kalman in 1960, the Kalman filter [20] is widely used in signal processing, navigation and object tracking. As it can be used to predict the object's position, velocity and acceleration with a relatively lightweight computation, it is a powerful tool for estimating the movement of an object frame-by-frame in real time.

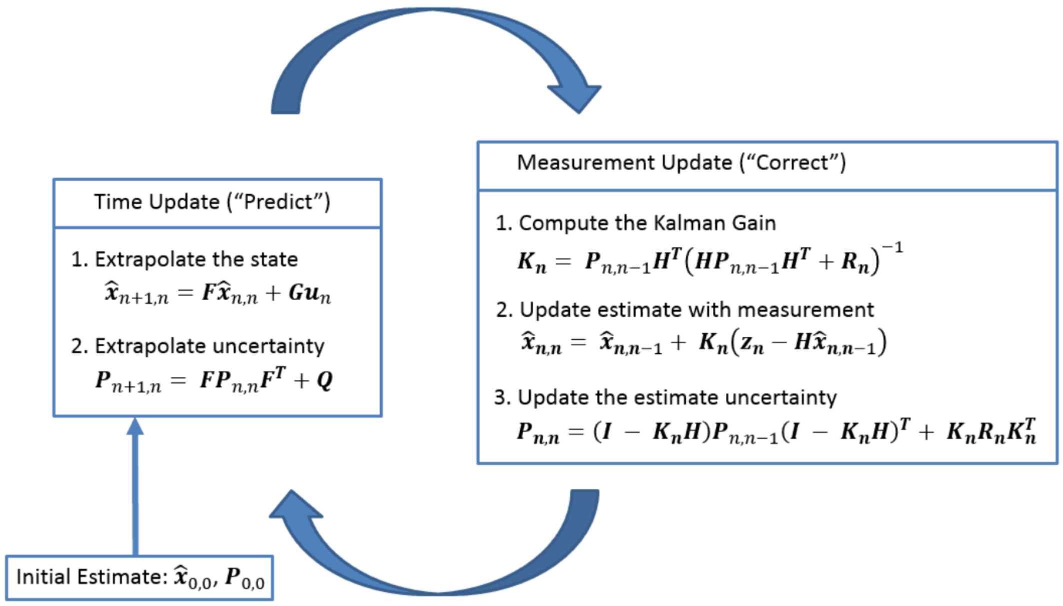
Time Update ("Predict")
1. Extrapolate the state
$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_n$
2. Extrapolate uncertainty
$P_{n+1,n} = FP_{n,n}F^T + Q$
Measurement Update ("Correct")
1. Compute the Kalman Gain
$K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1}$
2. Update estimate with measurement
$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - H\hat{x}_{n,n-1})$
3. Update the estimate uncertainty
$P_{n,n} = (I - K_nH)P_{n,n-1}(I - K_nH)^T + K_nR_nK_n^T$
Initial Estimate: $\hat{x}_{0,0}$, $P_{0,0}$

Figure 5.2: Kalman states [5].

The Kalman filter takes in a series of noisy measurements over time and aims to estimate the hidden state of the system recursively, using the new measurement and a previously calculated prediction to update the estimated state. As can be seen in figure 5.2, the process of Kalman filter proceeds in two phases for every time step k: the prediction step and the update step. In the prediction step the chosen model dynamics are applied to the last state estimate at time step $k-1$, resulting in predictions for the state at time step k. In the update step, the new measurement collected at time step k is used to update the state estimate by comparing it to the previously computed prediction.

The estimated hidden state $\hat{\boldsymbol{x}}_k^+$ at time step k is updated as follows:

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\hat{\boldsymbol{x}}_k^-), \tag{5.1}$$

where $\hat{\boldsymbol{x}}_k^-$ is the predicted state at time step k before new the measurement $\boldsymbol{z}_k$ is taken into account, and $\boldsymbol{K}_k$ is the Kalman gain, which determines how much weight is given to the measurement, as opposed to the predicted state. The observation matrix $\boldsymbol{H}$ transforms the state vector to the measurement space. In typical object tracking task this simply means dropping the velocities and keeping the estimated coordinates of the bounding box, as we can only observe the position of the bounding box directly in $\boldsymbol{z_k}$, and not the velocity.

The predicted state is based on all the previous measurements before time step k and the chosen model dynamics. In other words, $\hat{\boldsymbol{x}}_k^+$ is *a posteriori* estimate of the true hidden state

$\boldsymbol{x}_k$, where the measurement at time step k is known and taken into account. Accordingly, the predicted estimate, $\hat{\boldsymbol{x}}_k^-$ is *a priori* estimate of the hidden state $\boldsymbol{x}_k$. The predicted estimate depends both on the previous a posteriori estimate and the model chosen to describe the dynamics in the system, represented by the state transition matrix $\boldsymbol{F}$:

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{F}\hat{\boldsymbol{x}}_{k-1}^+ \tag{5.2}$$

For object tracking in 2D images, a popular choice for $\boldsymbol{F}$ is the constant velocity model [4].

In observation $\boldsymbol{z}_k$, two kinds of noise are present. The uncertainty caused by measurement noise is modelled in measurement covariance matrix $\boldsymbol{R}$. The inherent noise originating from the observed process, also present in true hidden state $\boldsymbol{x}_k$, is taken into account in the model through the process noise covariance matrix $\boldsymbol{Q}$. Both process noise and measurement noise are assumed to be uncorrelated, Gaussian distributed with zero mean. Setting a high value for $\boldsymbol{Q}$ corresponds to assuming higher uncertainty regarding the chosen model dynamics, and placing more weight on the observations. Also the opposite is true, and setting $\boldsymbol{Q}$ closer to zero will make the filter react more slowly to the new observations.

The covariance of the estimation error is denoted as $\boldsymbol{P}_k$, with $\boldsymbol{P}_k^-$ being the covariance of the estimation error of $\hat{\boldsymbol{x}}_k^-$, and $\boldsymbol{P}_k^+$ being the covariance of the estimation error of $\hat{\boldsymbol{x}}_k^+$. The time-update equation for $\boldsymbol{P}_k^-$ is computed using a posteriori covariance matrix of the previous time step and adding the process noise $\boldsymbol{Q}$:

$$\boldsymbol{P}_k^- = \boldsymbol{F}\boldsymbol{P}_{k-1}^+\boldsymbol{F}^T + \boldsymbol{Q} \tag{5.3}$$

The posteriori estimate covariance matrix is updated using the priori estimate $\boldsymbol{P}_k^-$, Kalman gain $\boldsymbol{K}_k$, and adding a term incorporating the measurement uncertainty $\boldsymbol{R}$:

$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_k^-(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})^T + \boldsymbol{K}_k\boldsymbol{R}\boldsymbol{K}_k^T \tag{5.4}$$

It is notable that computing $\boldsymbol{P}_k^-$, $\boldsymbol{P}_k^+$ and $\boldsymbol{K}_k$ does not depend on the observations, but only the parameters chosen [39]. This highlights how important it is to define the model dynamics and assumptions in the form of $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ so that they accurately depict the underlying phenomenon. In addition to defining the model dynamics to start the filtering process, we need to set the model's initial conditions $\hat{\boldsymbol{x}}_0^-$ and $\boldsymbol{P}_0^-$.

On each update step, the Kalman gain is computed using the the measurement uncertainty $\boldsymbol{R}$ and the uncertainty of the previous estimation, represented by the covariance matrix $\boldsymbol{P}_k^-$:

$$\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_k^-\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{5.5}$$

As equation 5.5 is one that minimizes the trace of $\boldsymbol{P}_k$, which results in minimizing the estimation error variance, the uncertainty represented by $\boldsymbol{P}_k$ decreases on its update step [39]. If the model dynamics and assumptions are correct, the filter should converge to such $\boldsymbol{K_k}$ and $\boldsymbol{P_k}$ that the estimate of $\boldsymbol{x_k}$ stays close to the real signal.

The underlying assumption of the Kalman filter is that the underlying motion in the estimated system is linear. Clearly dairy cow movement, as well as pedestrian movement, in 2D video footage can be non-linear. Approaches to more accurately model the true underlying motion have been researched in the context of multi-object tracking, but a simple Kalman filter often outweighs approaches like Extended Kalman Filters and particle filters in computation speed, straightforward implementation and more forgiving parameter choices [25] [19]. In practice, modelling process noise $\boldsymbol{Q}$ helps the Kalman filter to put more emphasis on the observations compared to the defined model dynamics when needed, causing the model to perform passably, even if the assumptions on the underlying motion are not strictly accurate [39]. Due to their simplicity and robustness, Kalman filter based methods can achieve high scores on benchmarks [47].

Hungarian algorithm

After the objects have been detected in the most recent frame and the estimated new positions of the tracks in the previous frame have been computed, we have a list of detections and a list of tracks along with their estimated new positions. These detections and tracks should be matched so that the most probable ones are matched together. A way to approach the problem is to match together the pairs whose bounding boxes overlap, considering that if there are several overlapping pairs, a solution that maximises the total overlap over the matchings should be chosen.

The problem corresponds to an assignment problem, which is defined as finding a perfect matching in a balanced bipartite graph $G = (V, E)$ of a maximum total weight w. A *bipartite graph* is a graph whose vertices V can be divided into two distinct parts, $S \subseteq V$ and $T \subseteq V$, so that every edge $e \in E$ connects a vertex $v \in V$ from S to T. A *matching* M in G is defined as a set of edges without common vertices. A *maximum* matching of a graph is a matching that contains the largest possible number of vertices. A *perfect* matching of a graph is a matching where every vertex of the graph is adjacent to exactly one edge in the matching. If S and T have the same number of vertices, i.e. they have equal cardinality $|S| = |T|$, the graph is considered a *balanced* bipartite graph. A *complete* bipartite graph has edges going out from each vertex in S to all the vertices in T. Example of a perfect matching on a balanced bipartite graph is shown in figure 5.3.

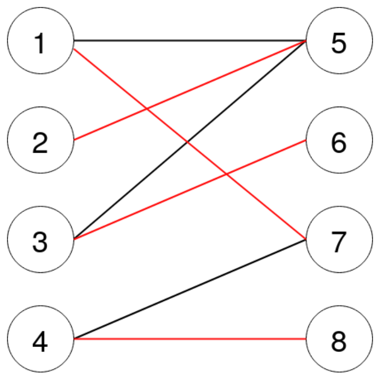

Figure 5.3: A balanced bipartite graph where S and T both have a cardinality of 4, with its perfect (and maximum) matching marked in red. This graph is not complete, as it does not have edges connecting all vertices in S to all vertices in T.

Solving the assignment problem can be done using the Hungarian algorithm, formulated by Harold W. Kuhn in 1955 [22], and later reviewed by James Munkres, who proved its polynomial time complexity [29]. It is substantially more efficient than the naive solution, in which $\mathcal{O}(n!)$ possible assignments have to be iterated through. The Hungarian algorithm, as its name suggests, is based on the work of two Hungarian mathematicians, Dénes Kőnig and Jenő Egerváry, and their essential theorem on bipartite graphs:

Theorem 1 (Kőnig-Egerváry theorem) *In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover. [10]*

Based on theorem 1, by finding a minimum vertex cover in our bipartite graph, we also find a maximum matching. As it is clear that a maximum matching in a complete bipartite graph is also a perfect matching, consequently by finding a minimum vertex cover we also find the perfect matching – a solution to the assignment problem, given that the total weight of the solution is also maximized.

In tracking context, S and T are the detections and tracks, and the edge weights are the intersection over union scores of the bounding boxes of the detections and the estimated bounding boxes of the tracks, forming a cost matrix $\boldsymbol{C}$ of size $|S| \times |T|$. In this section a weight minimizing solution is presented, which can also be used to solve maximum weight problem by simply negating and re-scaling the cost matrix to have a zero minimum. In the classic form of the Hungarian algorithm, the bipartite graph is considered to be balanced and complete. If the number of detections and tracks are not equal, dummy variables are used in order to get a square cost matrix. In the SORT implementation used in the experiments of section 5.2, Jonker–Volgenant [18] is used for the assignment problem. It is an optimized variation of the Hungarian algorithm that can also handle rectangular cost matrices more efficiently [18]. However, in order to focus on the mathematical background of the Hungarian algorithm, the classic Kuhn-Munkres [29] version is presented in this section.

In practice, the Hungarian algorithm in matrix formulation for a matrix of size $n \times n$ works in four steps, demonstrated on a small 4×4 example for clarity:

1. The smallest entry in each row is subtracted from all the other entries in that row.

$$\begin{bmatrix} 2 & 6 & \mathbf{1} & 6 \\ \mathbf{3} & 5 & 5 & 6 \\ \mathbf{1} & 7 & 2 & 4 \\ 5 & 2 & \mathbf{1} & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 5 & 0 & 5 \\ 0 & 2 & 2 & 3 \\ 0 & 6 & 1 & 3 \\ 4 & 1 & 0 & 1 \end{bmatrix}$$

2. The smallest entry in each column is subtracted from all the other entries in that column.

$$\begin{bmatrix} 1 & 5 & \mathbf{0} & 5 \\ \mathbf{0} & 2 & 2 & 3 \\ 0 & 6 & 1 & 3 \\ 4 & \mathbf{1} & 0 & \mathbf{1} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 0 & 4 \\ 0 & 1 & 2 & 2 \\ 0 & 5 & 1 & 2 \\ 4 & 0 & 0 & 0 \end{bmatrix}$$

The key to steps 1 and 2 is that when a constant has been subtracted from all the entries in a row or column, it results in every possible perfect matching having had each of these constants subtracted from one of its vertices exactly once. Thus, the inequalities present in the original matrix between the perfect matchings are also present after the operations. Now, if there is an optimal solution given by the matrix after these operations, it is also the optimal solution to the original problem.

3. The rows and columns are marked that contain 0 as an entry, *however done so that as few as possible are marked.*

$$\left[\begin{array}{c|c|c|c} 1 & 4 & 0 & 4 \\ 0 & 1 & 2 & 2 \\ 0 & 5 & 1 & 2 \\ \hline 4 & 0 & 0 & 0 \end{array}\right]$$

Using the minimum number of lines corresponds to a minimum vertex cover. As Kőnig's theorem states that the number of vertices in a minimum vertex cover equals the number of edges in a maximum matching in any bipartite graph, we can deduce that if the number of rows and columns marked is n, we have found the minimum vertex cover, and thus a maximum matching. Additionally, a zero entry in the marked row or column ensures that there is a minimum weight assignment between two vertices on different sides of the bipartite graph. If the number of marked rows and columns is less than n, a maximum matching with minimum weight has not been found yet, and we continue to step 4.

4. The smallest entry is found that is not covered by a mark. This entry is subtracted from each row that is not marked and added to each column that is marked.

$$\begin{bmatrix} 1 & 4 & 0 & 4 \\ 0 & \mathbf{1} & 2 & 2 \\ 0 & 5 & 1 & 2 \\ 4 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 3 & -1 & 3 \\ -1 & 0 & 1 & 1 \\ -1 & 4 & 0 & 1 \\ 4 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 0 & 3 \\ 0 & 0 & 2 & 1 \\ 0 & 4 & 1 & 1 \\ 5 & 0 & 1 & 0 \end{bmatrix}$$

This corresponds to adding an *augmenting path* in a graph. An augmenting path is an alternating path between two vertices that do not belong to the matching M. It can be used to increase the size of the matching by taking the symmetric difference of M and the edges of the augmenting path, changing the unmatched edges to matched ones, and vice versa. After this, we go back to step 3 to see if the maximum matching can now be found.

Continuing our example, we mark the rows and columns again, marking as few as possible:

$$\begin{bmatrix} 1 & 3 & 0 & 3 \\ 0 & 0 & 2 & 1 \\ 0 & 4 & 1 & 1 \\ 5 & 0 & 1 & 0 \end{bmatrix}$$

Our matrix now has a total of $n = 4$ marked rows and columns, so a maximum matching with minimum total weight has been found. As the graph is a complete bipartite graph, the maximum matching is also a perfect matching. It can be read by finding the combination of zeroes so that every zero belongs to only one row and one column, defining an perfect matching between the rows and the columns with minimum total weight:

$$\begin{bmatrix} 1 & 3 & \mathbf{0} & 3 \\ 0 & \mathbf{0} & 2 & 1 \\ \mathbf{0} & 4 & 1 & 1 \\ 5 & 0 & 1 & \mathbf{0} \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 6 & \mathbf{1} & 6 \\ 3 & \mathbf{5} & 5 & 6 \\ \mathbf{1} & 7 & 2 & 4 \\ 5 & 2 & 1 & \mathbf{2} \end{bmatrix}$$

If the number of rows and columns would have still been less than n after adding the augmenting path, the algorithm would have continued iterating between steps 3 and 4, until the maximum matching was found. Berge's theorem, which was already observed by Kőnig in his work but proven by Claude Berge in 1957, states that a matching M is maximum if and only if there does not exist an augmenting path [6]. Thus, when there are no more augmenting paths to be added, we know that a maximum matching has been found.

5.1.2 DeepSORT

DeepSORT [45] is an extension of SORT that aims to increase the accuracy of the model by using an association metric computed from the appearance of the objects in the bounding boxes. A bare-bones SORT implementation does not take into account the object appearance at all, only its position and velocity, so DeepSORT aims to improve the tracking by computing similarity metrics for the contents of the detected bounding boxes. On a pedestrian benchmark MOT16 DeepSORT gained a 45% decrease on identity switches compared to SORT, with additional improvements on the number of mostly tracked objects and mostly lost objects [45]. In order to achieve this, in addition to utilizing the appearance metric, DeepSORT uses the squared Mahalanobis distance to measure the distance between the estimated Kalman state and the measured position of the detected bounding box, to filter out unlikely assignments. It also uses a matching cascade algorithm instead of solving the assignment problem for all of the pairs at once, which gives priority to the tracks that have been seen more frequently.

The Mahalanobis distance is a metric that describes the distance between a point and a distribution, which means that it takes into account the state estimation uncertainty encoded in covariance matrix $\boldsymbol{P}$ of the Kalman filter. The distance between detection $j \in \mathcal{D}$ and track $i \in \mathcal{T}$ is calculated as follows:

$$d^{(1)}(i,j) = (\boldsymbol{z}_j - \hat{\boldsymbol{x}}_i)^T \boldsymbol{P}_i^{-1} (\boldsymbol{z}_j - \hat{\boldsymbol{x}}_i), \tag{5.6}$$

where the variables used for the computation are all familiar to us from SORT: $\boldsymbol{z}_j$ is the measured position of detection j, and $\hat{\boldsymbol{x}}_i$ is the state estimate of track i at some time step. The thresholding to filter out the unlikely positioned detections is done by calculating whether the Mahalanobis distance of the detection lies within the 95% confidence interval of the inverse χ^2 distribution of the state estimate, and if not, ruling out the detection as a possible match for the track. The underlying assumption is that the data is normally distributed in our multivariate 4-dimensional measurement space, from which results that the sum of these normal random variables is χ^2 distributed. Inverse χ^2 distribution is the probability distribution of a reciprocal of a variable distributed according to the χ^2

distribution:

$$X \sim \chi^2(\nu), Y = \frac{1}{X} \Rightarrow Y \sim \text{Inv-}\chi^2(\nu) \tag{5.7}$$

The appearance descriptors used by Wojke et al. are obtained by using a CNN that has been pre-trained on a large pedestrian re-identification dataset. The appearance metric is the smallest cosine distance found between the appearance descriptor of the detection j and the appearance descriptors of the track $\mathcal{R}_\rangle$, which are the last $L_k = 100$ detections that have been associated with the track i:

$$d^{(2)}(i,j) = min\{1 - \boldsymbol{r}_j^T \cdot \boldsymbol{r}_k^{(i)} \mid \boldsymbol{r}_k^{(i)} \in \mathcal{R}_i\} \tag{5.8}$$

The appearance information is incorporated to the cost matrix C by computing a weighted sum of the Mahalanobis distance and the appearance metric for these pairs:

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda)d^{(2)}(i,j) \tag{5.9}$$

Hyperparameter λ is used to control the weight between the two metrics. Wojke et al. decide to use $\lambda = 0$ on their experiments, as they found it a good choice when there is substantial camera motion present in the data [45]. This leaves only the appearance distance as the cost of the assignments, and the Mahalanobis distance is used purely to filter out unlikely assignments.

After the distances have been calculated and the very unlikely matches have been filtered out, the matching cascade algorithm is run on the cost matrix $\boldsymbol{C}$. It begins by running the Hungarian algorithm on the unassigned detections and tracks of age $t = 1$, which means the tracks have been associated with a detection in the previous frame. The next iteration takes all the unmatched detections that are left and tracks of age $t = 2$. The algorithm is iterated until all the detections have been matched or all tracks younger than maximum age a_{max} have been processed. This way, it gives priority to the more recently seen tracks, as solving the assignment problem without regard to track age can lead to unwanted prioritization of old tracks due to the Kalman filter assigning higher uncertainties to track locations that have not been detected for a longer time, together with the Mahalanobis distance assigning a lower cost to a matching when the uncertainty is high. Giving accidental priority to older tracks can lead to track fragmentation and decrease track stability, which the matching cascade aims to prevent [45].

In an experiment presented in section 5.2, the performance of SORT and DeepSORT are compared, plugging appearance descriptors trained on different datasets in DeepSORT: namely a pre-trained re-identification model trained on OpenCows2020 and a freshly trained re-identification model using the Maaninka dataset. As the model published by Wojke et al. has been pre-trained with pedestrians, replacing the re-identification model is an obvious choice in improving the performance of the tracking model when identifying cows instead of humans.

5.2 Experiments

For the experiments, a SORT model using a YOLOv4 detector with confidence threshold of 0.4 and NMS threshold of 0.4 was used, as deemed suitable based on the previous parameter tuning experiments in chapter 3. On the tracking test dataset described in subsection 5.2.1, the detector achieved mean F1 score of 95.47% and mAP@0.5 score of 96.94%. Predicted bounding boxes of the cows are pre-computed by the YOLOv4 detector, and only the detections are given as an input to the SORT model. DeepSORT also uses these detections, but in addition uses the bounding box contents to form a gallery for computing visual similarities. Feeding station data can be used during tracking in order to connect the tracks to real cow ids. The overview of the whole system can be seen in Figure 5.4. For DeepSORT, two differently trained re-identification models were used as the descriptor model: one trained on OpenCows data and one trained on Maaninka, as described in section 4.2.

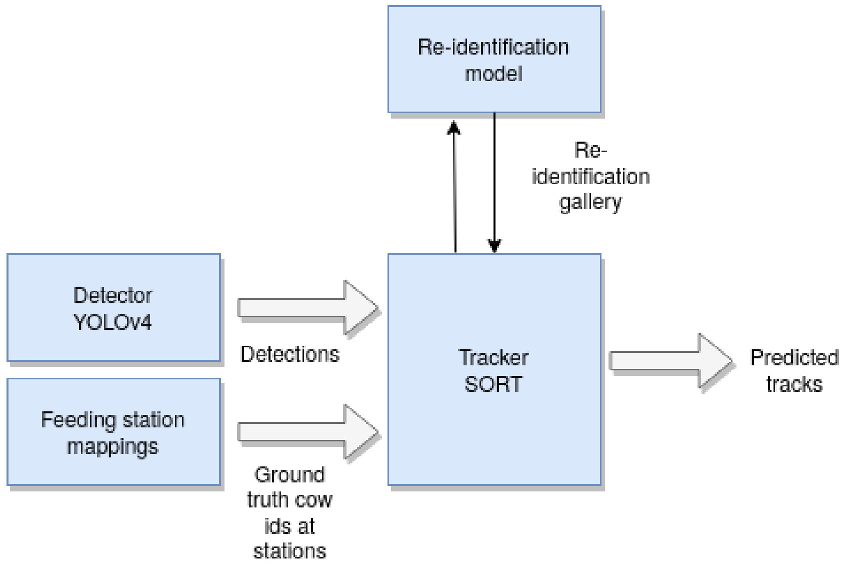

Figure 5.4: Overview of the DeepSORT based tracking system, using feeding station mappings and a re-identification model as visual descriptor.

5.2.1 Dataset

The test dataset for the multi-object tracking evaluation consisted of two 7-minute video recordings with 20 frames-per-second frame rate, one from camera C and one from camera D, 16800 frames in total. For practical purposes, the ground truths were annotated every 2 seconds, and the resulting 420 ground truth frames were used to evaluate the model performance. As the model is not evaluated on every frame, the true number of identity switches and track fragmentation is likely greater than presented, but under the pretext of knowing this should give us a good idea of the model performance.

In camera C, 21 of the 24 individuals were seen during the 7-minute period, totaling 25 tracks with a median length of 310 seconds. In camera D, 14 individuals appeared, forming 25 tracks in total, with a median track length of 118 seconds. Even in a short observation time, there were differences in cow activity and behavior: some of the cow individuals stayed still for the whole seven-minute recording time, while some individuals moved around more and frequently exiting and re-appearing in the frames, resulting in several tracks (Figure 5.5). Of the 25 tracks in camera C, 14 included visits to feeding stations, and 9 out of 25 tracks in camera D. Examples of the data can be seen in Figure 5.6.

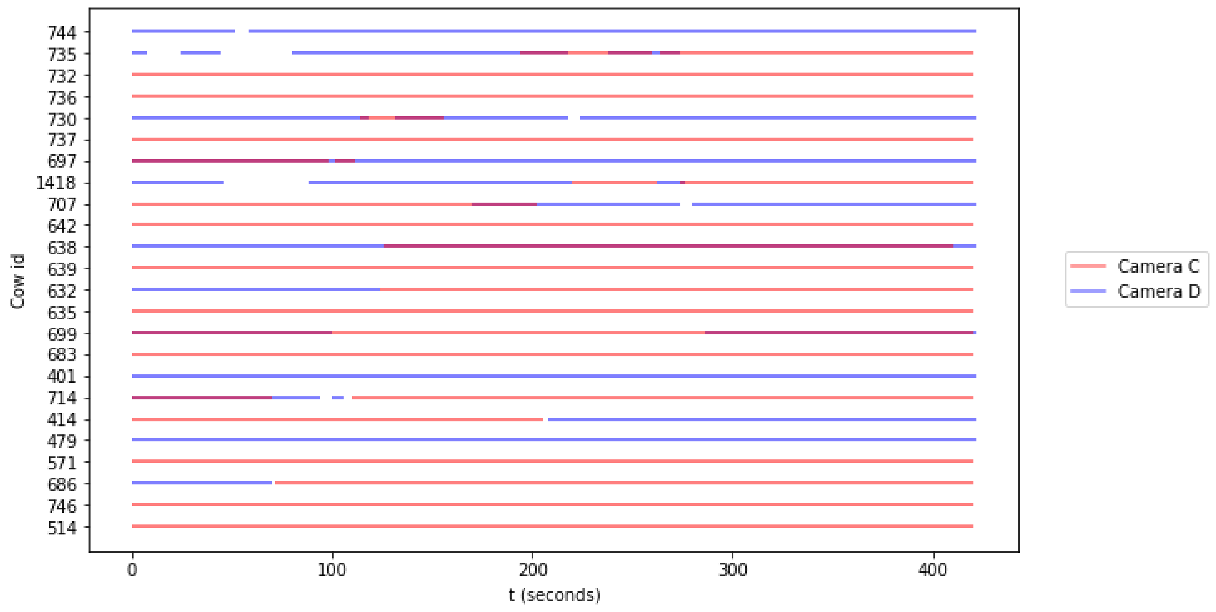

Figure 5.5: Cow ground truth appearances at points in time in cameras C and D during the 7-minute recording. A line on the y-axis means that the corresponding cow is present in the image at the point in time indicated by the x-axis. Disconnections on lines show cows that are outside the frame, in a blind spot for the camera. Overlap of blue and red lines shows occurrences of cows visible in both cameras.

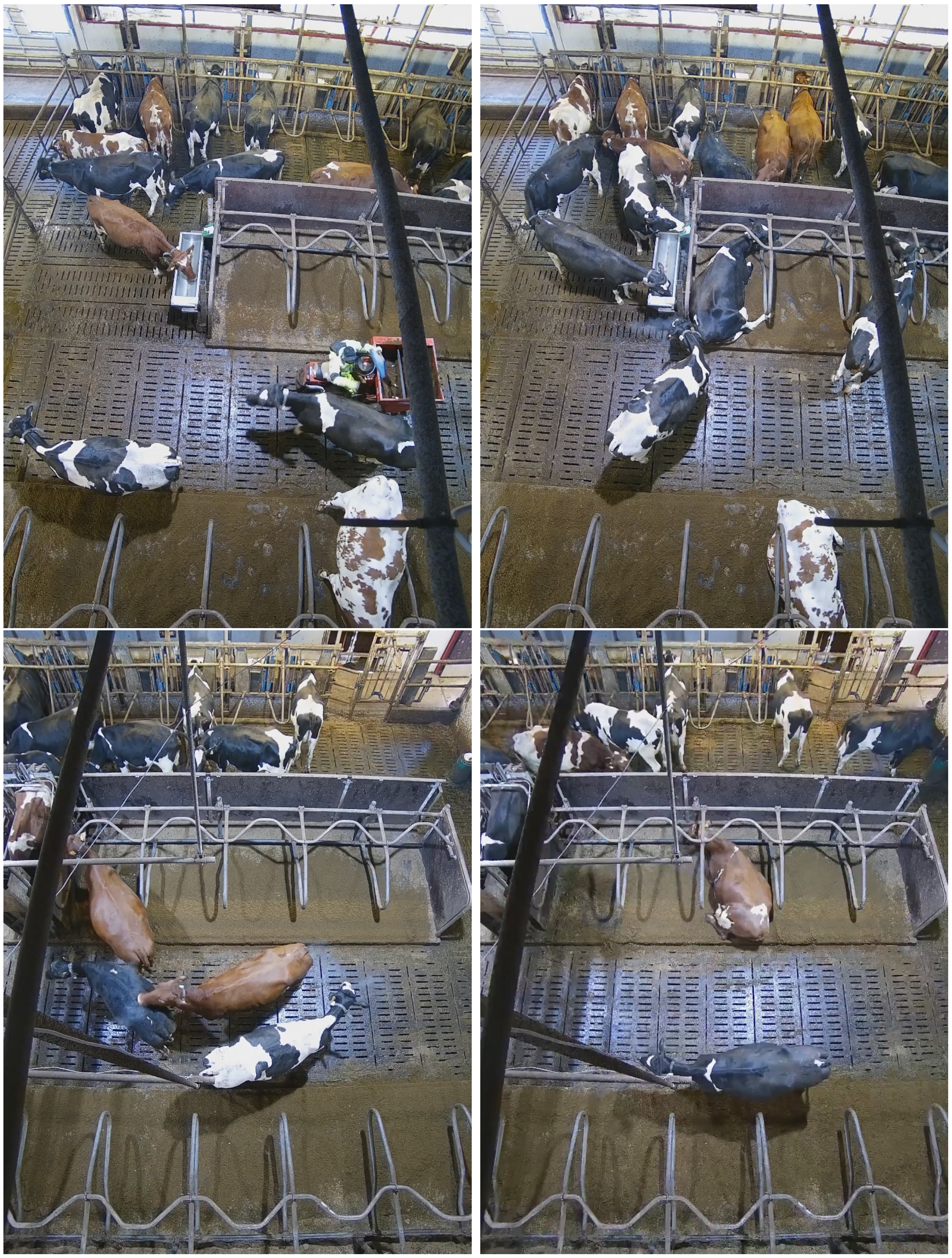

Figure 5.6: Examples of data from cameras C (upper) and D (lower), taken at the start and end of the 7-minute period. In camera D there was more movement around the department.

5.2.2 Metrics

Due to the temporal nature of the tracking problem and the fact that in a multi-object problem the frames usually contain a varying number of objects, a simple frame-averaged accuracy metric does not describe the performance of a model precisely enough. Instead, a collection of metrics is used that aim to capture the different aspects of the problem – a subset of the collection of metrics that is a common standard when comparing different MOT models, as discussed previously when comparing MOT16 benchmarks in section 5.1:

- Mostly tracked (MT): Percentage of ground-truth tracks that have the same predicted tracking label for at least 80% of their life span.
- Mostly lost (ML): Percentage of ground-truth tracks that are tracked for at most 20% of their life span.
- Identity switches (ID): Number of times the reported identity of a ground-truth track changes.
- Multi-object tracking accuracy (MOTA): Summary of over-all tracking accuracy in terms of false positives (FP), false negatives (FN) and identity switches (ID):

$$\text{MOTA} = 1 - \frac{\sum_t (FN_t + FP_t + ID_t)}{\sum_t GT_t} \tag{5.10}$$

Of all of these metrics, MOTA tends to be the most commonly used due to summarizing the false positives, false negatives and identity switches in one metric [28]. As we use the same detector to evaluate all the tracking models, two of the MOT16 metrics, multi-object tracking precision (MOTP) and fragmentation (FM), are omitted in the following experiments, as they assess mostly the performance of the detection model. It is good to note that for all of the metrics, due to the varying number of objects in each frame, it is important to sum up the errors across frames instead of calculating the metrics for each frame and then computing the average, to get a more reasonable assessment of the model [7].

An important part of a multi-object tracking algorithm is the accuracy of the associations: the algorithm should not get individuals mixed up, even if they move close to each other, and ideally there would be a one-to-one mapping between the detected tracks and ground truth tracks. In reality, this is often not the case. The number of identity switches gives us some idea about the number of these mismatches: it tells us how many times the ground truth track is either mixed up with another individual or a new track is initialized due to losing the previous track. However, the metric does not differentiate between these two, and as such tells us more about the general tracking stability than the model's ability to not mismatch. In addition, even a single identity switch can be very costly in animal behaviour applications, if the behaviour of an individual is attributed to another due to two long tracks getting connected by a single identity switch. Therefore, in order to get an estimate of the prevalence of mismatches, association precision ($\mathrm{P_A}$) is calculated: it is the number of true positive associations divided by the total number of associations for the ground truth track. Association to the detected track is considered true positive for a ground truth track if it has the highest number of associations to the detected track over the frames, compared to the other ground truth tracks.

In addition, a metric aiming to connect the proposed model's performance to its suitability in the real-world behavior observation task is presented: the feeding station coverage (FS) denotes the percentage of detections that can be connected to a ground truth cow identity through contiguous tracks that contain a visit to a feeding station. Ground truth tracks in the Maaninka test set have an FS of 64.38%, and ideally the model should attain a score that is close to this.

5.2.3 Parameter search

In SORT, there are three essential parameters present: firstly, maximum age a_{max} defines the maximum number of frames that the track is kept alive if no matching detections are found. The second parameter, the IoU threshold t_{IoU}, is used during predictive tracking to determine if the detection and track are a possible match or not. Third parameter, minimum hits a_{min}, is the number of consequent detections required before a new track is initialized.

A search over the max age parameter a_{max} suggests that while the track should be kept alive long enough to avoid unnecessary identity switches, after the value exceeds roughly 80 frames (4 seconds), the performance of the tracking model does not improve anymore in terms of MOTA (Figure 5.7). Increasing a_{max} has a small negative effect on the association precision (Figure 5.8).

A low IoU threshold causes the model to more readily connect a track to a detection. When this threshold is increased, the number of false positives decreases (Figure 5.9). On the other hand, higher IoU threshold causes more false negatives, as some of the correct

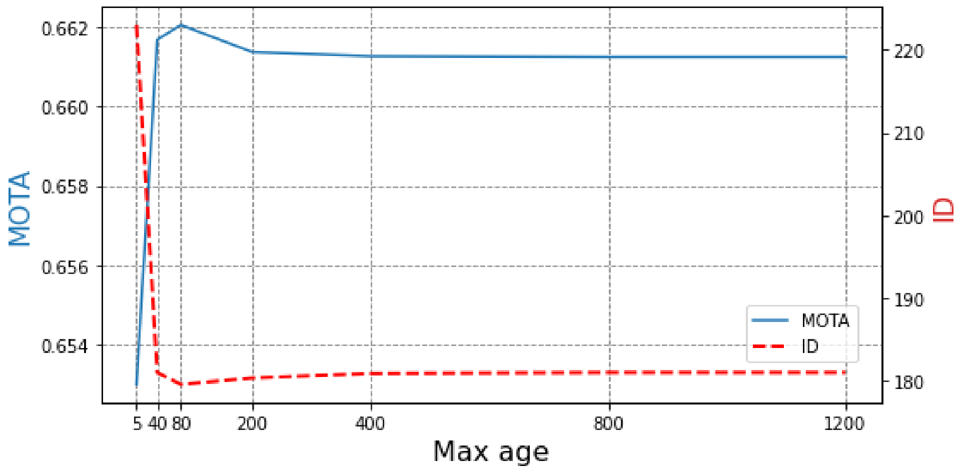

Figure 5.7: The performance of SORT in terms of MOTA and identity switches (ID) over different max age parameters, using $a_{min} = 1$ and averaged over $\{t_{IoU} \mid t_{IoU} \in \{0.1, 0.2, ..., 0.9\}\}$.

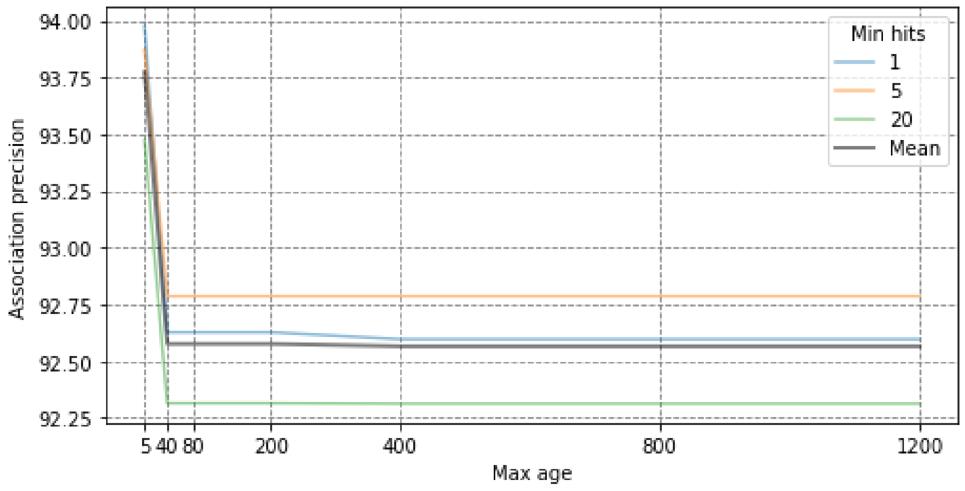

Figure 5.8: Association precision of SORT over max age, using different min hits values and $t_{IoU} = 0.5$.

matches get filtered out (Figure 5.10). In these occurrences the number of identity switches also increases, as a new track is initialized for the detection now missing a matching track (Figure 5.11). The suitable IoU threshold is a compromise between these metrics, summarized by MOTA: an IoU threshold of 0.5 results in the highest MOTA score of 68.90 when using $a_{max} = 80$ and $a_{min} = 1$. Using these parameters we end up with fairly reasonable $\mathrm{P_A} = 92.63$, ID $= 52$ and 82 tracks in total.

In addition to the parameters already present in SORT, DeepSORT has two of its own: gallery size h and maximum cosine distance d_{cos}. Gallery size is the maximum number of images that are kept in memory for the appearance-based matching. Maximum cosine distance is the maximum allowed visual difference between the detection and the track.

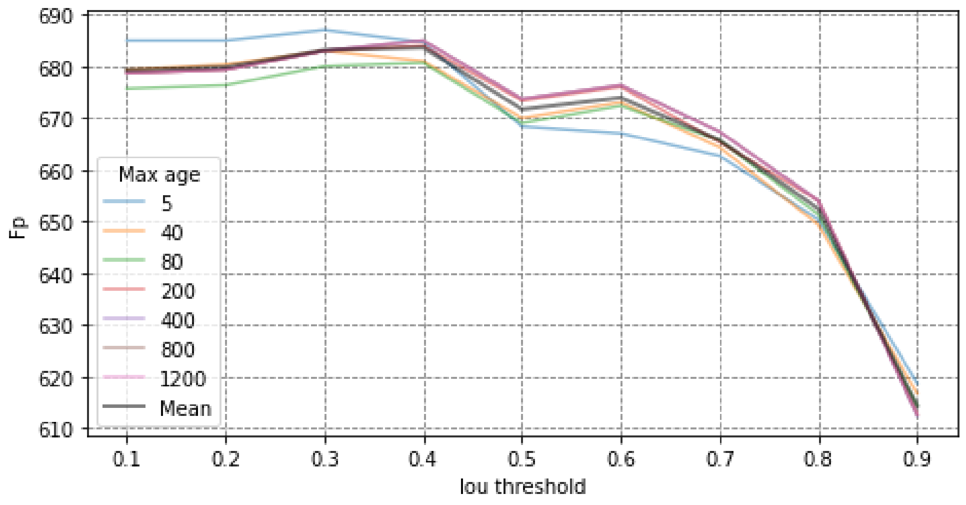

Figure 5.9: SORT: Number of false positives over IoU thresholds using different max age parameters.

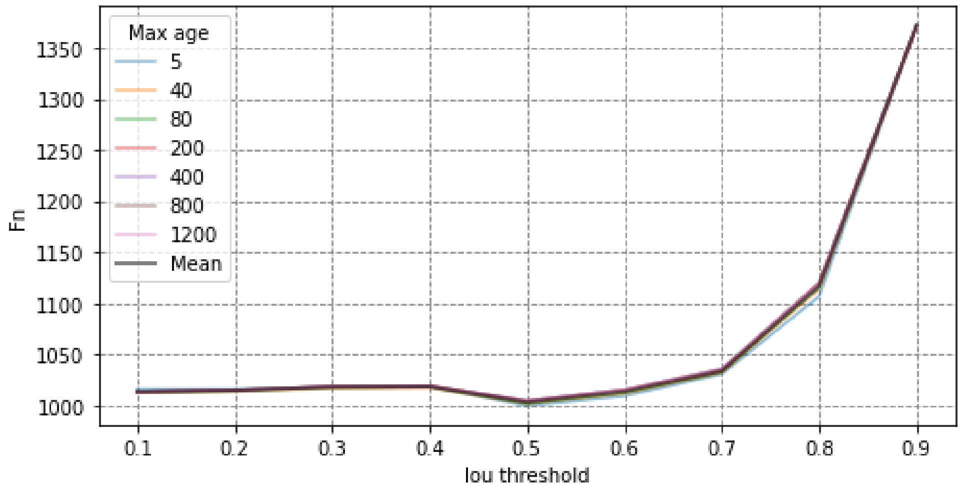

Figure 5.10: SORT: Number of false negatives over IoU thresholds using different max age parameters.

An interesting factor in addition to the parameters is the Mahalanobis gating that is used to filter out improbable detection matches for a track based on its location. With closer inspection, we can see that when the Mahalanobis gating is in use, it effectively rules out most of the detections as possible matches in our dataset. The result is that after the gating, we are left with a matching that is close to perfect, and only a few tracks are matched based on appearance (Figure 5.12). We can demonstrate the difference by disabling the Mahalanobis gating, and we see that only now most of the tracks actually use the visual descriptors computed from the bounding box contents (Figure 5.12). For this reason, the parameter search on DeepSORT-specific parameters is performed on a model without the Mahalanobis gating. For the IoU thresholding, a value of 0.5 that resulted in

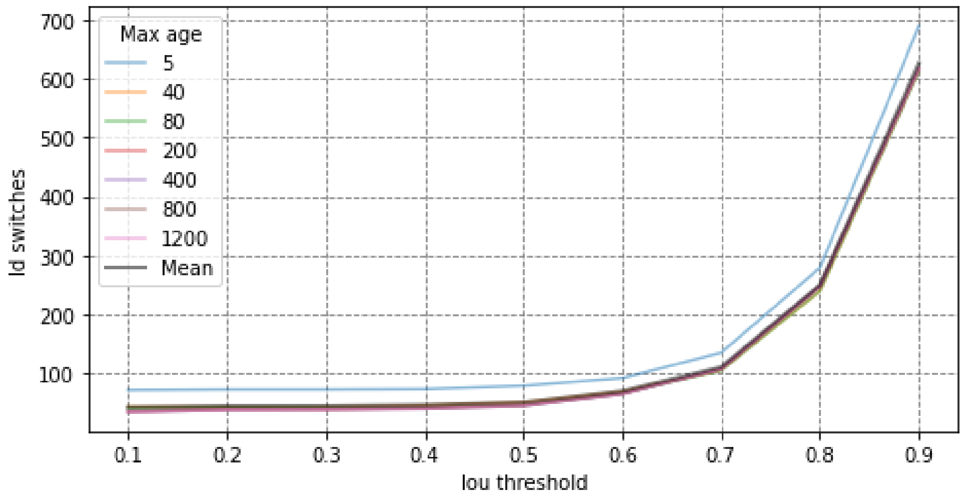

Figure 5.11: SORT: Number of identity switches over IoU thresholds using different max age parameters.

the best performance in SORT is used throughout these experiments. Similarly, $a_{min} = 1$ is used.

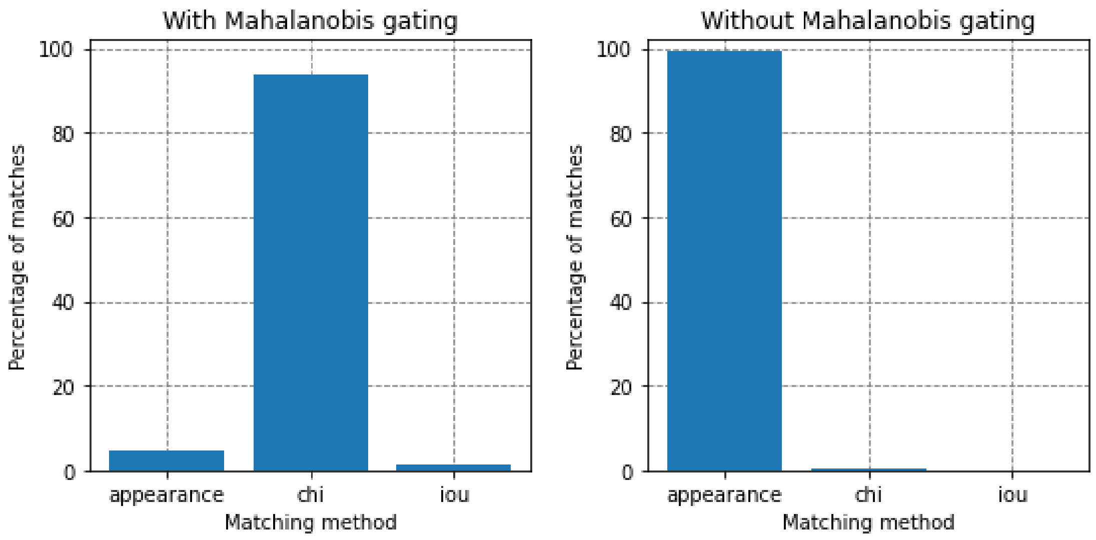

Figure 5.12: Differences in DeepSORT matching method prevalence with and without Mahalanobis gating, using $a_{max} = 80$, $h = 100$ and $d_{cos} = 0.3$. Matching method 'appearance' indicates that the track and detection are matched using matching cascade with visual distances as cost matrix, 'chi' a perfect matching produced by the location-based Mahalanobis gating, and 'iou' the fallback Hungarian algorithm using IoU scores as cost matrix.

Tuning gallery size h in itself does not affect whether the track is matched based on appearance or not, but affects the quality of the appearance matching. Thus, the assumption is that having a reasonably large gallery is recommendable, with the cost of memory use and computation speed. We see that growing gallery size from 1 to 100 reduces identity switches and consequently increases MOTA (Figure 5.13). Further increasing the gallery size to 1000 causes the number of identity switches to increase again, suggesting that the gallery size should not be too large in order to preserve the meaningful temporal connection between the images from previous frames kept in gallery, and the images present in the most recent frame. In addition, the effect the gallery size has on P_A is dependent on the choice of d_{cos} (Figure 5.14).

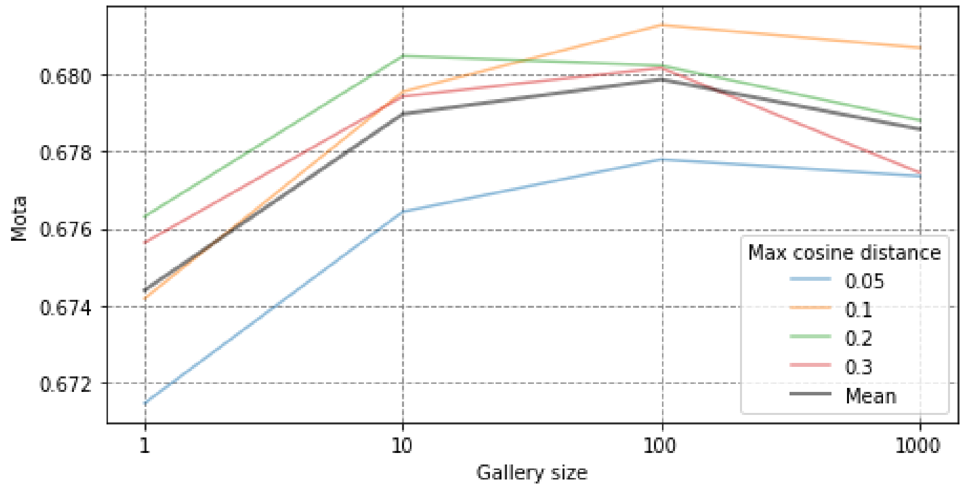

Figure 5.13: DeepSORT: MOTA with gallery size and different max cosine distances.

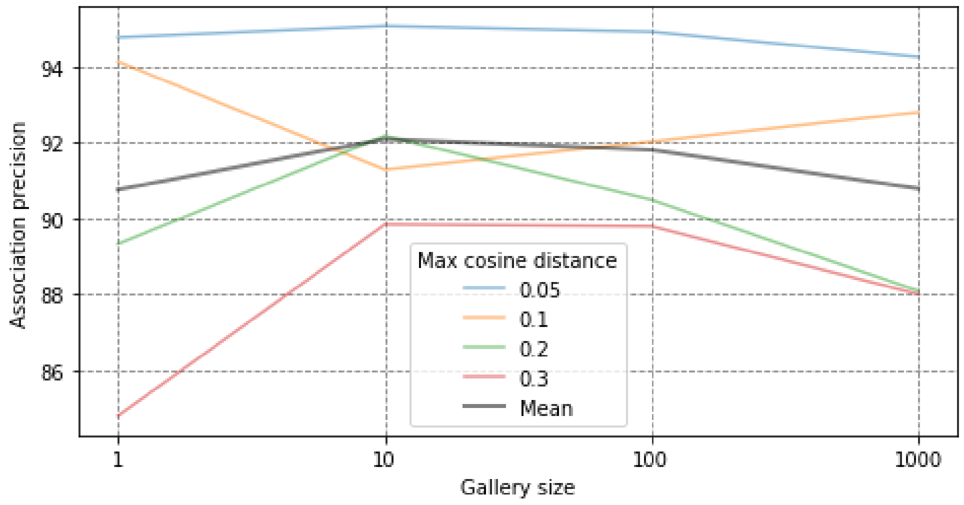

Figure 5.14: DeepSORT: Association precision with gallery size and different max cosine distances.

We can clearly see that the appearance-based matching does in fact decrease identity switches and result in more contiguous tracks (Figure 5.15). However, when we evaluate the models by using the association precision metric, we notice that these changes now come with a cost of adding more mixups between the cow identities; the decrease in identity switches is due to the model not assigning a new track to a detection, even when it should be doing so. An extreme example can be seen using large cosine distance and maximum age parameters $d_{cos} = 0.3$ and $a_{max} = 400$, where DeepSORT produces only 35 tracks, when the ground truth dataset has 50 tracks in total, causing $\mathrm{P_A}$ to drop down to 79.46 (Figure 5.16). The example shows why $\mathrm{P_A}$ and the number of tracks are important indicators to take into consideration when evaluating the tracking model – the other metrics can still give an impression of a reasonable performance, also in terms of identity switches, even though in reality the individuals are getting eagerly mixed up.

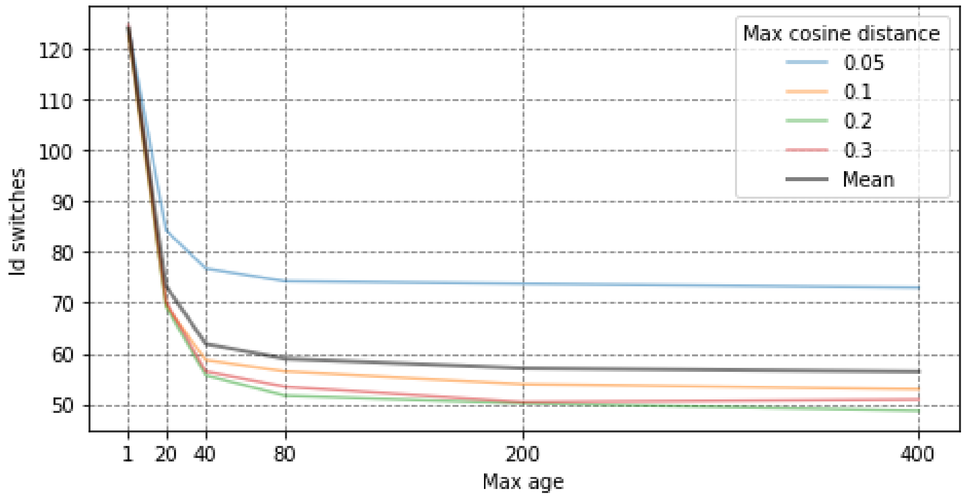

Figure 5.15: DeepSORT: identity switches with max age and max cosine distance.

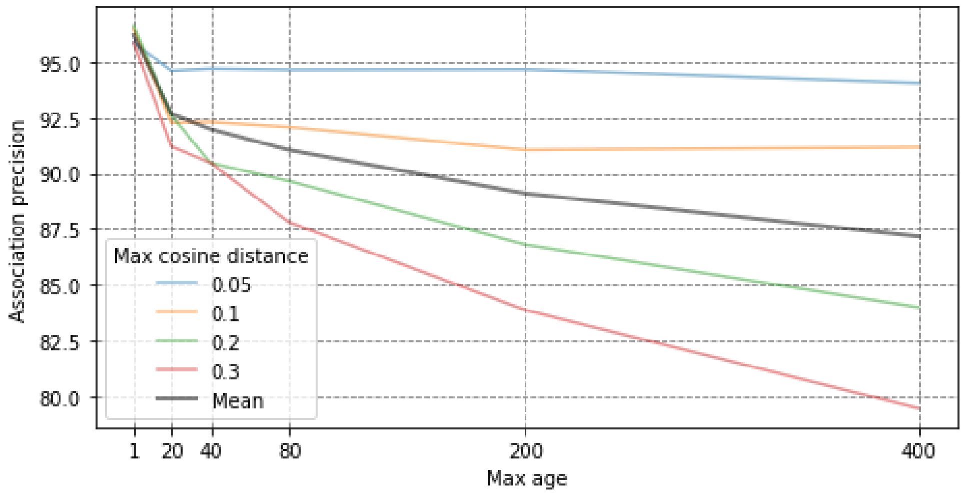

Figure 5.16: DeepSORT: association precision with max age and max cosine distance.

The drastic effect that the maximum age parameter has on the association precision in DeepSORT can be explained by the parameterization of the matching cascade algorithm:

it does not match tracks that are older than a_{max}. Older tracks are matched using the Hungarian algorithm with an IoU cost matrix. Consequently, high a_{max} parameter results in the model using the appearance-based matching cascade, while a low parameter value makes the model act closer to SORT, and only rely on location. A similar effect can be achieved by decreasing the maximum cosine distance parameter, requiring the visual similarity to be very high in order to match based on visual appearance. As we saw from previous experiments with SORT, increasing a_{max} by itself does not worsen the results to the extent seen in DeepSORT. Therefore, the culprit must be the appearance matching and its use of a_{max}. A natural step would be to separate the matching cascade depth from a_{max} to its own searchable parameter, but this is left for future research. Working with the existing architecture, choosing a small d_{cos} restricts the most improbable visual matches enough for us to increase a_{max} to a more optimal value. Without Mahalanobis gating, the highest MOTA score of 68.59 was achieved using $d_{cos} = 0.1$, $h = 100$ and $a_{max} = 200$, resulting in $P_A = 90.41$, ID = 38 and 50 predicted tracks in total.

For the model using OpenCows weights, similar parameter search was conductedl. It was found that the more optimal d_{cos} differed somewhat between the two models. This was to be expected due to them forming a completely different embedding space. For OpenCows, $d_{cos} = 0.05$ was found to result in the best MOTA scores over a range including smaller values (Figure 5.17). Models using these parameters, along with $a_{max} = 200$ and $h = 100$ based on experiments with the Maaninka model, are presented in the final results.

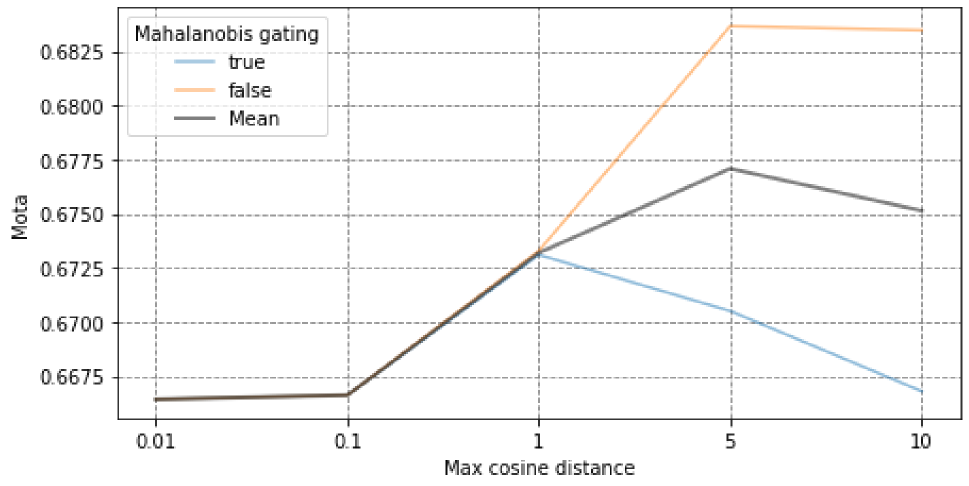

Figure 5.17: DeepSORT with OpenCows weights: MOTA over different d_{cos} values.

5.2.4 Results

The results for DeepSORT with different re-identification modules can be seen on Table 5.2. Both Maaninka and OpenCows models are able to decrease identity switches compared to SORT, but only when Mahalanobis gating is not in use.

Overall, SORT attains highest MOTA and a model trained on Maaninka with no Mahalanobis gating results in most improvement in terms of ID, while suffering some loss in P_A. The Maaninka model gains a 35.7% improvement in the number of mostly tracked tracks and 26.9% reduction in the number of identity switches compared to SORT. It is good to note that even though the P_A of OpenCows gated model is high, the overall performance of the model is low. Similar improvements in P_A can be achieved with SORT by decreasing the a_{max} parameter, which similarly results in higher number of tracks and identity switches.

Table 5.2: Tracking results on the different models with and without Mahalanobis gating. 1) Number of predicted tracks (N) 2) Feeding station coverage (FS): Percentage of detections connected to a ground truth cow identity through a station visit during a track (ground truth 64.38%).

Model	Gating	$N^{1)}$	MOTA ↑	P_A ↑	ID ↓	MT ↑	ML ↓	$FS^{2)}$ ↑
SORT [8]		82	**68.90**	92.63	52	32%	28%	46.41%
OpenCows [45] [1]	✓	115	67.31	**95.57**	97	26%	28%	28.39%
OpenCows [45] [1]		54	68.37	90.62	47	38%	28%	37.36%
Maaninka [45]	✓	72	67.40	93.98	111	36%	28%	38.57%
Maaninka [45]		**50**	68.59	90.41	**38**	**42**%	28%	**52.38**%

For a pedestrian multi-object tracking model, the resulting best MOTA score of SORT 68.90 would be a fairly good result. As we saw earlier in section 5.1, SORT only achieves a MOTA of 59.8 on the MOT16 benchmark, which would indicate that the Maaninka tracking setup is likely an easier problem than a typical benchmark scene, especially in terms of detection accuracy. In Maaninka, the camera angles and department layouts of the training data and the test data bear great resemblance to each other. In addition, cows move quite slowly and the view over the animal is mostly unoccluded during the track, which might often not be the case in a crowded scene of people filmed from ground level.

Neither of the Mahalanobis-gated models beat SORT in Maaninka test data, on the contrary resulting in worse performance than a non-gated model using the same parameters.

As previously shown, using the Mahalanobis gating results in the model relying almost solely on the estimate produced by the Kalman filter.

From the viewpoint of minimizing association mismatches, it was found that the Maaninka-trained model that got the highest $\mathrm{P_A}$ of 93.66 while also attaining a comparable MOTA (> 68.1) was achieved by using $d_{cos} = 0.1$, $h = 1000$ and $a_{max} = 80$ without Mahalanobis gating, resulting in MOTA = 68.33, ID = 46, FS = 46.10% and 53 predicted tracks in total. The model improved SORT results both in terms of ID and $\mathrm{P_A}$, but not FS.

Based on the experiments presented, when choosing the suitable parameters for a Deep-SORT tracking model, it would be wise to balance between MOTA and an association metric such as $\mathrm{P_A}$, depending on the application at hand. The more important it is to be able to reliably connect the behavior to an individual, the more weight should be placed on the association metric. When aiming to extract longer behavioral patterns, the MOTA, ID and MT are emphasized to achieve longer, coherent tracks.

6 Conclusions

We find that dairy cow tracking performance of a simple Kalman-filter approach can be improved by using a re-identification model as a visual descriptor using similar training data to the test footage, resulting in reduced number of identity switches. We also see that different parameters and choice of training data for the re-identification model affect the tracking model performance greatly.

A pre-trained re-identification model trained on OpenCows did reduce the number of identity switches compared to SORT, even if its performance as a standalone re-identification model was poor. However, it was not able to improve the percentage of detections connected to a ground truth cow identity through feeding station visits. The best-performing model trained on Maaninka data connected over half of the test detections to a ground truth identity, and resulted in 35.7% improvement in the number of mostly tracked tracks and 26.9% reduction in the number of identity switches compared to SORT. On the other hand, MOTA decreased 0.4% and association precision decreased 2.4%, due to identity mismatches caused by the favoring of more continuous tracks.

During the experiments with DeepSORT, it was noticed that the Mahalanobis distance gating causes the model to restrict the matching of the tracks and detections heavily, effectively making the model to rely on the Mahalanobis gating to produce the matchings and ignore appearance information.

The limitations of this thesis include not utilizing the combined information from the two-camera view of the department to track individuals from one camera view to the next or rule out already recognized individuals. In addition, the annotation inconsistencies seen in the test results of the detection experiments suggest that more precise annotation would have improved the detection accuracy and thus the resulting tracking accuracy.

As there are commonly crowded scenes and blind spots in a cow tracking scenario, better occlusion handling through explicit modelling [40] than presented in this thesis would important. Furthermore, even though the results in this thesis did not show satisfactory improvements to tracking through the use of re-identification model trained on different dataset as-is, domain adaptation methods could help better utilize the features learned on a different dataset [11]. For tracking, nonlinear estimation methods, such as the extended Kalman filter, unscented filtering and particle filtering could prove to be more suitable for cow movement.

Being able to reliably identify cow individuals even at some locations automatically is one way of reducing the time and costs of the labelling work required by deep learning models, and it opens up interesting paths of research in the direction of self-supervised learning and automatic data collection, especially when combined with a tracking algorithm [21] [44]. Using a self-supervised method to extract images along the track together with an RFID setup or other reliable identification method, the architecture proposed in this thesis would enable gathering a fully annotated re-identification dataset, making it possible to train a fully supervised identification model without any manual labelling.

Bibliography

[1] W. Andrew, J. Gao, N. W. Campbell, A. W. Dowsey, and T. Burghardt. *OpenCows2020 dataset*. https://doi.org/10.5523/bris.10m32xl88x2b61zlkkgz3fml17. [Online; accessed 21-January-2021]. 2020.

[2] W. Andrew, J. Gao, S. Mullan, N. W. Campbell, A. W. Dowsey, and T. Burghardt. "Visual identification of individual Holstein-Friesian cattle via deep metric learning". In: *Comput. Electron. Agric.* 185 (2021), p. 106133. DOI: 10.1016/j.compag.2021.106133. URL: https://doi.org/10.1016/j.compag.2021.106133.

[3] W. Andrew, C. Greatwood, and T. Burghardt. "Visual Localisation and Individual Identification of Holstein Friesian Cattle via Deep Learning". In: *2017 IEEE International Conference on Computer Vision Workshops, ICCV Workshops 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 2850–2859. DOI: 10.1109/ICCVW.2017.336. URL: https://doi.org/10.1109/ICCVW.2017.336.

[4] N. L. Baisa. "Derivation of a Constant Velocity Motion Model for Visual Tracking". In: *CoRR* abs/2005.00844 (2020). arXiv: 2005.00844. URL: https://arxiv.org/abs/2005.00844.

[5] A. Becker. *KalmanFilter.NET*. https://www.kalmanfilter.net/. [Online; accessed 20-January-2022]. 2022.

[6] C. Berge. "Two Theorems in Graph Theory". In: *Proceedings of the National Academy of Sciences* 43.9 (1957), pp. 842–844. DOI: 10.1073/pnas.43.9.842. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.43.9.842. URL: https://www.pnas.org/doi/abs/10.1073/pnas.43.9.842.

[7] K. Bernardin, A. Elbs, and R. Stiefelhagen. "Multiple object tracking performance metrics and evaluation in a smart Room environment". In: *Proceedings of IEEE International Workshop on Visual Surveillance* (Jan. 2006).

[8] A. Bewley, Z. Ge, L. Ott, F. T. Ramos, and B. Upcroft. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing, ICIP 2016, Phoenix, AZ, USA, September 25-28, 2016*. IEEE, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003. URL: https://doi.org/10.1109/ICIP.2016.7533003.

[9] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[10] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan Education UK, 1976. ISBN: 978-1-349-03521-2. DOI: 10.1007/978-1-349-03521-2. URL: https://doi.org/10.1007/978-1-349-03521-2.

[11] G. Csurka. “A Comprehensive Survey on Domain Adaptation for Visual Applications”. In: *Domain Adaptation in Computer Vision Applications*. Ed. by G. Csurka. Advances in Computer Vision and Pattern Recognition. Springer, 2017, pp. 1–35. DOI: 10.1007/978-3-319-58347-1_1. URL: https://doi.org/10.1007/978-3-319-58347-1%5C_1.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848. URL: https://doi.org/10.1109/CVPR.2009.5206848.

[13] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] O. Guzhva, H. Ardö, M. G. Nilsson, A. H. Herlin, and L. Tufvesson. “Now You See Me: Convolutional Neural Network Based Tracker for Dairy Cows”. In: *Frontiers Robotics AI* 5 (2018), p. 107. DOI: 10.3389/frobt.2018.00107. URL: https://doi.org/10.3389/frobt.2018.00107.

[15] R. Hadsell, S. Chopra, and Y. LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*. IEEE Computer Society, 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100. URL: https://doi.org/10.1109/CVPR.2006.100.

[16] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://doi.org/10.1109/CVPR.2016.90.

[17] H. Hu, B. Dai, W. Shen, X. Wei, J. Sun, R. Li, and Y. Zhang. “Cow identification based on fusion of deep parts features”. In: *Biosystems Engineering* 192 (Apr. 2020), pp. 245–256. DOI: 10.1016/j.biosystemseng.2020.02.001.

[18] R. Jonker and A. Volgenant. “A shortest augmenting path algorithm for dense and sparse linear assignment problems”. In: *Computing* 38.4 (1987), pp. 325–340. DOI: 10.1007/BF02278710. URL: https://doi.org/10.1007/BF02278710.

[19] S. J. Julier and J. K. Uhlmann. “New extension of the Kalman filter to nonlinear systems”. In: *Defense, Security, and Sensing*. 1997.

[20] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[21] S. Karthik, A. Prabhu, and V. Gandhi. “Simple Unsupervised Multi-Object Tracking”. In: *CoRR* abs/2006.02609 (2020). arXiv: 2006.02609. URL: https://arxiv.org/abs/2006.02609.

[22] H. W. Kuhn. “The Hungarian Method for the Assignment Problem”. In: *Naval Research Logistics Quarterly* 2.1–2 (Mar. 1955), pp. 83–97. DOI: 10.1002/nav.3800020109.

[23] M. Lagunes-Fortiz, D. Damen, and W. W. Mayol-Cuevas. “Learning Discriminative Embeddings for Object Recognition on-the-fly”. In: *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*. IEEE, 2019, pp. 2932–2938. DOI: 10.1109/ICRA.2019.8793715. URL: https://doi.org/10.1109/ICRA.2019.8793715.

[24] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Comput.* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541.

[25] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim. “Multiple object tracking: A literature review”. In: *Artif. Intell.* 293 (2021), p. 103448. DOI: 10.1016/j.artint.2020.103448. URL: https://doi.org/10.1016/j.artint.2020.103448.

[26] L. van der Maaten and G. Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[27] A. Masullo, T. Burghardt, D. Damen, T. Perrett, and M. Mirmehdi. “Who Goes There? Exploiting Silhouettes and Wearable Signals for Subject Identification in Multi-Person Environments”. In: *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*. IEEE, 2019, pp. 1599–1607. DOI: 10.1109/ICCVW.2019.00199. URL: https://doi.org/10.1109/ICCVW.2019.00199.

[28] A. Milan, L. Leal-Taixé, I. D. Reid, S. Roth, and K. Schindler. “MOT16: A Benchmark for Multi-Object Tracking”. In: *CoRR* abs/1603.00831 (2016). arXiv: 1603.00831. URL: http://arxiv.org/abs/1603.00831.

[29] J. Munkres. “Algorithms for the Assignment and Transportation Problems”. In: *Journal of the Society for Industrial and Applied Mathematics* 5.1 (1957), pp. 32–38. DOI: 10.1137/0105003.

[30] S. Murray. “Real-Time Multiple Object Tracking - A Study on the Importance of Speed”. In: *CoRR* abs/1709.03572 (2017). arXiv: 1709.03572. URL: http://arxiv.org/abs/1709.03572.

[31] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91. URL: https://doi.org/10.1109/CVPR.2016.91.

[32] J. Redmon and A. Farhadi. “YOLOv3: An Incremental Improvement”. In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[33] A. Rivas, P. Chamoso, A. González-Briones, and J. M. Corchado. “Detection of Cattle Using Drones and Convolutional Neural Networks”. In: *Sensors* 18.7 (2018). ISSN: 1424-8220. DOI: 10.3390/s18072048. URL: https://www.mdpi.com/1424-8220/18/7/2048.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[35] S. Ruuska, W. Hämäläinen, A. Sairanen, E. Juutinen, L. Tuomisto, M. Järvinen, and J. Mononen. “Can stealing cows distort the results of feeding trials? An experiment for quantification and prevention of stealing feed by dairy cows from roughage intake control feeders”. In: *Applied Animal Behaviour Science* 159 (2014), pp. 1–8. ISSN:

0168-1591. DOI: https://doi.org/10.1016/j.applanim.2014.08.001. URL: https://www.sciencedirect.com/science/article/pii/S016815911400207X.

[36] S. Schneider, G. W. Taylor, and S. C. Kremer. "Similarity Learning Networks for Animal Individual Re-Identification - Beyond the Capabilities of a Human Observer". In: *IEEE Winter Applications of Computer Vision Workshops, WACV Workshops 2020, Snowmass Village, CO, USA, March 1-5, 2020*. IEEE, 2020, pp. 44–52. DOI: 10.1109/WACVW50321.2020.9096925. URL: https://doi.org/10.1109/WACVW50321.2020.9096925.

[37] F. Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 815–823. DOI: 10.1109/CVPR.2015.7298682. URL: https://doi.org/10.1109/CVPR.2015.7298682.

[38] M. Schultz and T. Joachims. "Learning a Distance Metric from Relative Comparisons". In: *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*. Ed. by S. Thrun, L. K. Saul, and B. Schölkopf. MIT Press, 2003, pp. 41–48. URL: https://proceedings.neurips.cc/paper/2003/hash/d3b1fb02964aa64e257f9f26a31f72cf-Abstract.html.

[39] D. Simon. *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches*. Wiley, 2006. ISBN: 978-0-47170858-2. DOI: 10.1002/0470045345. URL: https://doi.org/10.1002/0470045345.

[40] D. Stadler and J. Beyerer. "Improving Multiple Pedestrian Tracking by Track Management and Occlusion Handling". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 10958–10967. DOI: 10.1109/CVPR46437.2021.01081. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Stadler%5C_Improving%5C_Multiple%5C_Pedestrian%5C_Tracking%5C_by%5C_Track%5C_Management%5C_and%5C_Occlusion%5C_Handling%5C_CVPR%5C_2021%5C_paper.html.

[41] Tzutalin. *LabelImg, a graphical image annotation tool*. https://github.com/tzutalin/labelImg. [Online; accessed 18-January-2022]. 2015.

[42] Q. Wang, Y. Zheng, P. Pan, and Y. Xu. "Multiple Object Tracking With Correlation Learning". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 3876–3886. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Wang%5C_Multiple%5C_Object%5C_Tracking%5C_With%5C_Correlation%5C_Learning%5C_CVPR%5C_2021%5C_paper.html.

[43] S. Wang, H. Sheng, Y. Zhang, Y. Wu, and Z. Xiong. "A General Recurrent Tracking Framework without Real Data". In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 13199–13208. DOI: 10.1109/ICCV48922.2021.01297. URL: https://doi.org/10.1109/ICCV48922.2021.01297.

[44] X. Wang and A. Gupta. "Unsupervised Learning of Visual Representations Using Videos". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 2794–2802. DOI: 10.1109/ICCV.2015.320. URL: https://doi.org/10.1109/ICCV.2015.320.

[45] N. Wojke, A. Bewley, and D. Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE International Conference on Image Processing, ICIP 2017, Beijing, China, September 17-20, 2017*. IEEE, 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962. URL: https://doi.org/10.1109/ICIP.2017.8296962.

[46] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. H. Hoi. "Deep Learning for Person Re-Identification: A Survey and Outlook". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 44.6 (2022), pp. 2872–2893. DOI: 10.1109/TPAMI.2021.3054775. URL: https://doi.org/10.1109/TPAMI.2021.3054775.

[47] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu. "FairMOT: On the Fairness of Detection and Re-identification in Multiple Object Tracking". In: *Int. J. Comput. Vis.* 129.11 (2021), pp. 3069–3087. DOI: 10.1007/s11263-021-01513-4. URL: https://doi.org/10.1007/s11263-021-01513-4.

[48] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu. "Object Detection With Deep Learning: A Review". In: *IEEE Trans. Neural Networks Learn. Syst.* 30.11 (2019), pp. 3212–3232. DOI: 10.1109/TNNLS.2018.2876865. URL: https://doi.org/10.1109/TNNLS.2018.2876865.