



**FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA**

**DEPARTMENT OF
CYBERNETICS**

Master's Thesis

Application for automatic tracking of animal movements in a confined space

Jakub Rada



FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
CYBERNETICS

Master's Thesis

Application for automatic tracking of animal movements in a confined space

Bc. Jakub Rada

Thesis advisor

Ing. Miroslav Hlaváč, Ph.D. et. Ph.D.

© 2024 Jakub Rada.

All rights reserved. No part of this document may be reproduced or transmitted in any form by any means, electronic or mechanical including photocopying, recording or by any information storage and retrieval system, without permission from the copyright holder(s) in writing.

Citation in the bibliography/reference list:

RADA, Jakub. *Application for automatic tracking of animal movements in a confined space*. Pilsen, Czech Republic, 2024. Master's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Cybernetics. Thesis advisor Ing. Miroslav Hlaváč, Ph.D. et. Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Bc. Jakub RADA**
Osobní číslo: **A22N0099P**
Studijní program: **N0714A150011 Kybernetika a řídicí technika**
Specializace: **Umělá inteligence a automatizace**
Téma práce: **Aplikace pro automatické sledování pohybu zvířat v uzavřeném prostoru**
Zadávající katedra: **Katedra kybernetiky**

Zásady pro vypracování

- Analýza stavu poznání v oboru automatického sledování pohybujících se objektů.
- Analýza záznamů pohybu zvířat.
- Výběr a otestování vhodné metody pro dostupná data.
- Vytvoření aplikace pro sledování a vyhodnocení pohybu zvířat v uzavřeném prostoru.


Rozsah diplomové práce: **40-50 stránek A4**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

- Ciaparrone, Gioele, et al. "Deep learning in video multi-object tracking: A survey." *Neurocomputing* 381 (2020): 61-88.
- Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

Vedoucí diplomové práce: **Ing. Miroslav Hlaváč, Ph.D.**
Výzkumný program 1

Datum zadání diplomové práce: **2. října 2023**
Termín odevzdání diplomové práce: **20. května 2024**



Doc. Ing. Miloš Železný, Ph.D.
děkan



Doc. Dr. Ing. Vlasta Radová
vedoucí katedry

Declaration

I hereby declare that this Master's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

Pilsen, on 20 May 2024

.....

Jakub Rada

The names of products, technologies, services, applications, companies, etc. used in the text may be trademarks or registered trademarks of their respective owners.

Abstract

V práci je popsán celý postup návrhu aplikace pro automatické sledování pohybu zvířat v uzavřeném prostoru, jak z teoretického, tak praktického hlediska. Jsou prozkoumány současné trendy v aplikacích pro sledování pohybujících se objektů a provedena analýza záznamů pohybu daných objektů, za využití různých metod, které budou společně se získanými výsledky porovnány a vyhodnoceny.

Abstrakt

The paper covers the whole process of designing an application for automatic tracking of animal movement in a confined space, both from a theoretical and practical point of view. The current trends in applications for tracking moving objects are investigated and the analysis of the motion records of the objects in question is performed, using different methods, which together with the obtained results are compared and evaluated.

Keywords

detection • tracking • application • neural networks • machine learning • biology research • liver transplantation

Contents

1	Introduction	5
2	Methods	7
2.1	Object representation	7
2.1.1	Definition of bounding boxes	7
2.2	Object detection	8
2.2.1	Faster R-CNN	9
2.2.2	DETR	11
2.2.3	Comparison of Faster-RCNN and DETR	12
2.3	Single object tracking	13
2.3.1	Discriminative methods	14
2.3.2	Generative methods	15
2.4	Multiple object tracking	17
2.4.1	Multiple object tracking methods	18
3	Dataset	23
3.1	Detailed information	23
3.2	Annotations	24
4	Experiments	27
4.1	Detection model	27
4.1.1	MMDetection	27
4.1.2	Detectron2	29

4.1.3	Comparison of MMDetection and Detectron2	31
4.2	Single object tracking	34
4.2.1	Implementation of methods	34
4.2.2	Results of single object tracking methods	35
4.3	Multiple object tracking	37
4.3.1	Implementation of methods	37
4.3.2	Results of multiple object tracking methods	39
5	Web Application	43
5.1	Application structure	43
5.1.1	Frontend	44
5.1.2	Backend	46
5.2	Application features	47
5.2.1	Data storage	47
5.2.2	Status of lights	48
5.2.3	Pig counter overview	49
5.2.4	Distance walked by each pig	49
5.2.5	Data export	50
5.2.6	Reporting	51
6	Conclusion	55
	Bibliography	57
	List of Figures	61
	List of Tables	63

Foreword

This thesis details the development of a web application designed to meet the specific needs of doctors at the Biomedical Center of Faculty of medicine of Charles University in Pilsen, aiming to facilitate their analysis of pig behavior. I am thankful to my advisor, Ing. Miroslav Hlaváč, Ph.D. et Ph.D., for his guidance and support throughout this research. I would also like to extend my gratitude to Ing. Miroslav Jiřík, Ph.D., for providing the innovative topic that inspired this work.

Bc. Jakub Rada,
thesis author, (may 2024)

Introduction

1

In the world of biomedical research and tech innovation, creating specialized tools to improve research methods and results has become very important. This thesis introduces an application designed for pig tracking to monitor their health state. Given the physiological similarities between pigs and humans, accurate and efficient health monitoring is important to ensure the validity of research. The thesis is structured to navigate through an examination of theoretical frameworks, data analysis, practical implementation, and the technological underpinnings of the developed application.

The initial part of the thesis establishes a theoretical background, presenting a detailed review of current methodologies in object tracking. This exploration into existing literature identifies gaps and opportunities for technological intervention, setting the stage for the development of an application created for the specific needs of medical research involving pigs. The theoretical discourse underscores the importance of integrating technological solutions with traditional research methodologies to enhance the efficiency and accuracy of health monitoring practices.

Moving from theoretical concepts to practical analysis, the thesis explores the data provided by the Biomedical Center of Faculty of medicine of Charles University in Pilsen. Dataset chapter then details the extent of the videos, which capture pigs behaviour inside of their pens, which is essential for assessing their health and overall condition. The thorough annotation of these videos is important for both the development and validation phases of the application, ensuring its precision. This process of video annotation directly contributes to tailoring the application to

effectively address the specific challenges encountered in the real-world monitoring of pigs in research settings.

The narrative then shifts to the practical application of methods, detailing the implementation of the earlier proposed methods. Chapter "Experiments" not only recounts the implementation process but also examines the challenges encountered. It also presents the results obtained from each experiment, highlighting its impact on improving the precision and efficiency of pig tracking, thereby contributing positively to final application functionality.

Concluding the thesis is an examination of the application's architecture, functionality, and features. This section offers a technical overview, elucidating the design principles, data management strategies, and user interface of the application. Special attention is given to the application's features, such as highlighting pigs movement inside their pens, analytical overview in form of graph and video information, which collectively empower applications ability to monitor pigs.

2.1 Object representation

Object representation is a crucial construct in computer vision [1], translating the physical world into digital data. These representations, including bounding boxes, key-points, and semantic masks, are important for machine understanding of visual information. The choice of object representation shapes the capabilities of computer vision systems and plays an important role in modern technology applications. This thesis will only employ bounding boxes as the object representation method, because it is optimal for mapping movements of pigs. Other representations such as semantic mask, would be unnecessarily complicated or inappropriate.

2.1.1 Definition of bounding boxes

Bounding boxes are mostly defined by two main parameters: the coordinates of the top-left corner (usually denoted as (x, y)) and the dimensions (width and height) of the rectangle that encloses the object of interest or the coordinates of the lower-right corner.

$$bbox = [x, y, width, height] \quad (2.1)$$

In single object tracking, these parameters are continuously adjusted and updated in each frame to maintain the object's accurate representation. Bounding boxes can vary in complexity and definition. Simple rectangular bounding boxes are commonly used for their easy implementation and efficiency. However, more advanced variants, such as rotated bounding boxes or contour-based representations, are employed in

scenarios where objects exhibit non-standard shapes or orientations. These variations are particularly relevant when tracking objects with irregular silhouettes or significant pose changes. Despite their simplicity and efficiency, bounding boxes have limitations. They may not accurately represent the object's shape, especially when objects undergo deformation or exhibit non-rigid motion. Furthermore, in cases of partial or full occlusion, bounding boxes can be less reliable, as they do not account for the interior object structure.

2.2 Object detection

Object detection is an important part of computer vision, where the goal is to find and identify objects in pictures or videos. Unlike classification of the image, object detection finds where multiple objects are, classify them, and shows their location with bounding boxes drawn around them. This is represented in figure 2.1.

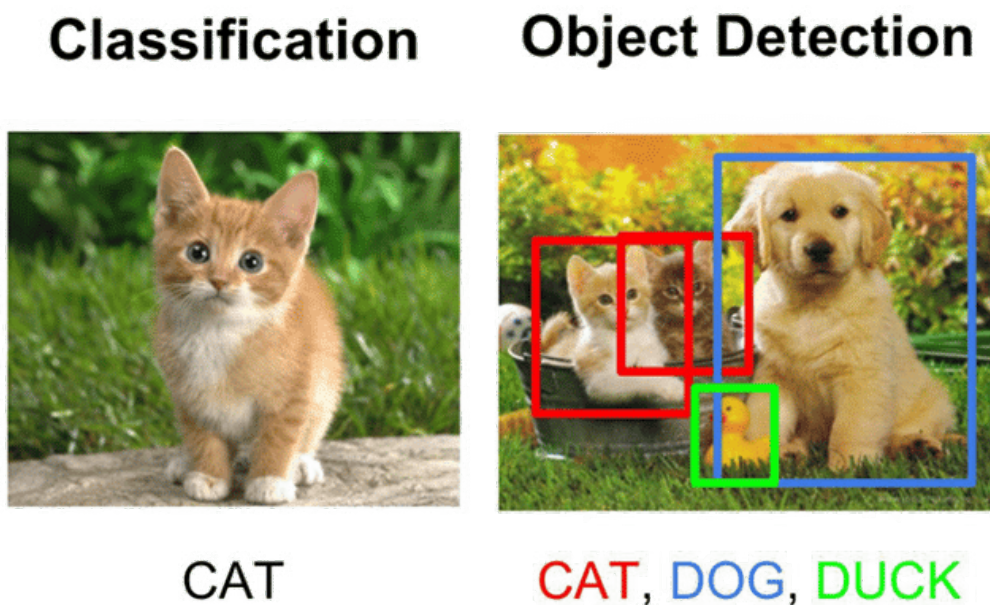


Figure 2.1: Difference between classification and detection [2]

To do this, it uses mostly methods and systems based on Neural Networks (NNs), which is type of machine learning based algorithms, but also classical computer vision methods. Neural network based systems are trained on huge collections of images that are marked to show different objects. This training helps them learn to spot various shapes, patterns, and features, so they can recognize objects even if they're in different places, under different lighting, or partly hidden.

Thanks to these advanced methods, object detection can be used in many areas, like helping self-driving cars see the road and other cars, making shopping easier with automatic checkouts, improving security by spotting unusual activities, and helping doctors find diseases in medical images.

Currently, the field of object detection algorithms is dominated by sophisticated models, including Faster R-CNN and DETR (Detection Transformer). These advanced frameworks represent the cutting edge in computer vision technology, showcasing significant improvements in both accuracy and efficiency over their predecessors.

2.2.1 Faster R-CNN

Faster R-CNN [3] (Region-based Convolutional Neural Network) brought a significant advancement in object detection algorithms, back in 2015, characterized by its enhanced efficiency and accuracy. This method builds upon principles used by its predecessors, R-CNN [4] and Fast R-CNN [5]. Through the innovative integration of a Region Proposal Network (RPN) with a deep convolutional network, Faster R-CNN optimizes the object detection workflow, enabling real-time application while maintaining high accuracy. It does not only accelerate the region proposal part but also enhances the detection accuracy. Faster-RCNN uses Loss function defined in equation 2.2.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i p_i^* L_{\text{reg}}(t_i, t_i^*) \quad (2.2)$$

Algorithm of Faster R-CNN begins with the input image being passed through a convolutional neural network to produce a feature map. The RPN then scans this

feature map with a sliding window approach, utilizing anchor boxes of various scales and ratios to predict object boundaries and scores indicating the presence of an object.

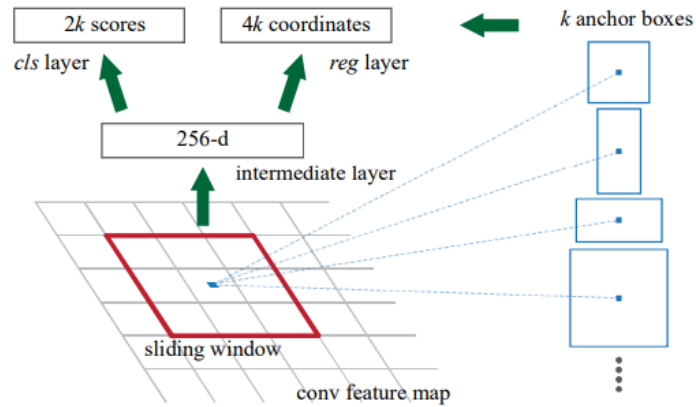


Figure 2.2: Region proposal network [3]

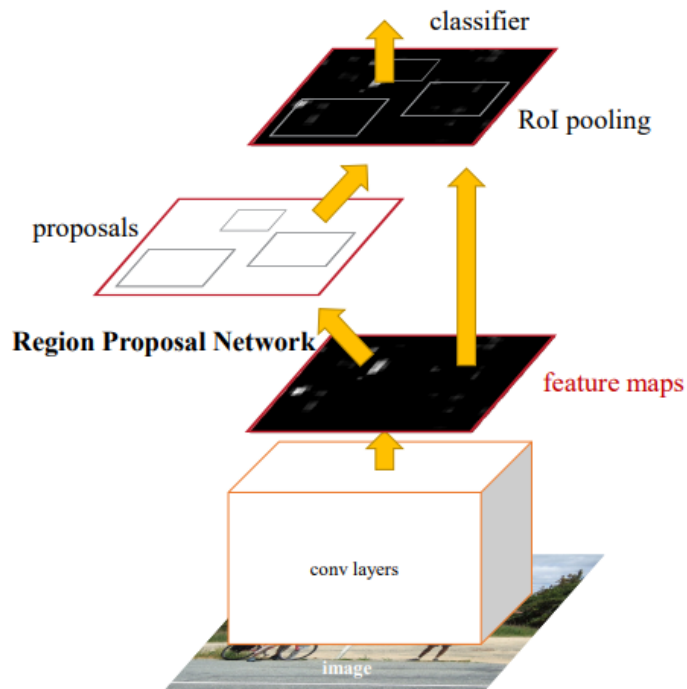


Figure 2.3: Graphical representation of Faster-RCNN [3]

These proposals are refined and classified through a process known as Region of Interest (RoI) pooling, which extracts and normalizes features for each proposal, followed by a series of fully connected layers that output the final object classifications and adjusted bounding box coordinates. By integrating proposal generation and object detection into a single, unified model, Faster R-CNN achieves remarkable detection speeds and accuracy, making it well-suited for real-time applications.

2.2.2 DETR

DETR [6] (Detection Transformer) differs in object detection by applying the transformer architecture, traditionally used in NLP (Natural Language Processing), to directly predict bounding boxes and class labels from an image. This method simplifies the object detection algorithm by eliminating the need for manually created parts such as creating anchors and filtering overlaps. As its backbone, DETR uses a CNN to extract feature maps from the input image, which are then processed by a transformer structure, using encoder-decoder architecture. The encoder captures global relationships within the image, while the decoder, using a set of learned queries, predicts objects classes and bounding boxes in a single step. This end-to-end model uses bipartite matching for loss calculation (defined in equation 2.3), ensuring a direct and efficient mapping between predicted and actual objects.

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^N \left[-\log \hat{p}_{\theta}(c_i) + I_{\{c_i \neq 0\}} L_{\text{box}}(b_i, \hat{b}(\hat{y}_i)) \right] \quad (2.3)$$

Despite its simplicity and the elimination of many traditional detection steps, DETR achieves competitive performance, marking a significant advancement in object detection with its ability to understand global image context, while reducing complexity.

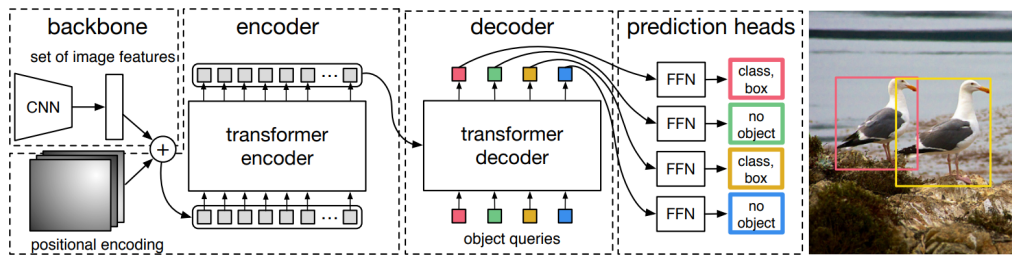


Figure 2.4: Graphical representation of DETR [6]

2.2.3 Comparison of Faster-RCNN and DETR

This subsection covers a comparison of the two NN architectures mentioned above. As it was said earlier, both DETR and Faster R-CNN are architectures for object detection in images, but their approach to the problem fundamentally differs in many ways. They exhibit distinct characteristics in terms of model complexity, training speed, parameter count, and model size. DETR presents an interesting contrast to Faster R-CNN in terms of learning time and model size. Contrary to what might be expected from its use of a Transformer architecture, DETR offers a more efficient learning process and a smaller parameter count compared to the conventional Faster R-CNN architecture. This efficiency is derived from DETR's end-to-end design, which eliminates the need for the manually designed components and heuristics, such as anchor generation and non-maximum suppression, that are typical in Faster R-CNN. As a result, DETR not only simplifies the training and inference pipelines but also achieves this with fewer parameters, resulting in a leaner model, characterized by its enhanced efficiency in computational resource use, memory consumption, all while maintaining robust performance. While Faster R-CNN has been a fundamental architecture in object detection for its effectiveness and real-time performance capabilities, DETR's approach reduces model complexity and speeds up the learning process, offering a compelling alternative that combines the benefits of transformer architecture with the efficiency required for practical applications in object detection. Comparison table 2.1 involves Faster R-CNN models

Model	FPS	#params	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Faster RCNN-DC5	16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	20	60M	44.0	63.9	47.8	27.2	48.1	56.0
DETR	28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	10	60M	44.9	64.7	47.7	23.7	49.5	62.3

Table 2.1: Comparison of object detection models

utilizing ResNet-50 and ResNet-101 backbones [7], evaluated on the COCO validation dataset. Initially, the results presented are from Faster R-CNN models based on the Detectron2 [8] framework. Subsequently, it details the outcomes for Faster R-CNN models enhanced with GIoU (Generalized Intersection over Union), random crop augmentations during training, and an extended 9x training regimen. DETR models showcase performances that are on par with extensively optimized Faster R-CNN counterparts, albeit with a decrease in AP for small objects (AP_S) but a significant improvement in AP for large objects (AP_L). The evaluation of FLOPS and FPS was performed using torchscript [9] versions of both Faster R-CNN and DETR models. It is noted that models not labeled with 'R101' are based on the ResNet-50 backbone.

2.3 Single object tracking

In the field of Single Object Tracking (SOT), the initial process involves the identification and outlining of the target object's boundaries within the initial frame of a video sequence. This preliminary identification serves as the foundation for the tracking algorithm, which is then tasked with the continuous localization and monitoring of the specified target throughout the successive frames of the video. Distinguished as a method within the detection-free tracking paradigm, SOT is characterized by its reliance on manual input for the initial bounding box, obviating the necessity for pre-trained detection algorithms to recognize the object of interest.

This manual specification facilitates the tracker’s ability to maintain focus on the designated object across varied and potentially dynamic visual contexts.

2.3.1 Discriminative methods

Discriminative models learn a discriminative model that can distinguish between the target object and the background. These models are typically faster to train and deploy than generative models, but they can be more sensitive to occlusions and changes in illumination.

Mean-shift tracking method

Mean-Shift Tracking [10] is a computer vision method for real-time object tracking in video sequences. It works by using color or density information to define and follow an object’s movement. This method is advantageous for its robustness to changes in object scale, rotation, and partial occlusion, adaptability to varying object appearance, simplicity, and computational efficiency. However, it may struggle with abrupt object motion, significant illumination changes, or when multiple similarly colored objects are present.

In the first frame, it creates a histogram representing the object’s color or density distribution. In subsequent frames, it calculates a back projection, which shows the likelihood of pixels belonging to the object. Mean-Shift iteratively updates the object’s position by shifting the region towards the peak of the back projection. It stops when the shift becomes minimal or after a set number of iterations, determining the final object location. Computing a mean-shift vector is defined in equation 2.4. A bounding box is drawn around the object for visualization.

$$M_t = \frac{\sum w_i \cdot x_i}{\sum w_i} \quad (2.4)$$

Where M_t is the calculated mean-shift vector. The w_i are weights that indicate how similar a pixel is to the target object, and x_i is the pixel location.

2.3.2 Generative methods

Generative models learn a generative model of the target object, which is then used to predict the object's appearance in the next frame. These models are typically more robust to occlusions and changes in illumination, but they can be computationally expensive to train and deploy.

MOSSE tracking method

MOSSE (Minimum Output Sum of Squared Errors) [11] is an effective object tracking algorithm widely applied in the field of computer vision. This method is renowned for its real-time tracking capabilities, particularly in challenging tracking conditions, such as low lighting and object occlusions.

It works by learning a representative object template from an initial frame, training filters to spot the object efficiently, applying these filters to each video frame to identify the object's likely location, continually updating filters for adaptability, and smoothly handling objects leaving or re-entering the frame. Given an input image sequence x_t and a desired output g_t , the MOSSE filter H is updated as follows:

$$H_{t+1} = \frac{\sum_t (X_t \cdot G_t^*)}{\sum_t (X_t \cdot X_t^*)} \quad (2.5)$$

where X_t is the Fourier transform of x_t , G_t is the Fourier transform of g_t , and $*$ denotes complex conjugation.

KCF tracking method

The Kernelized Correlation Filter (KCF) [12] is a widely used algorithm for real-time object tracking in computer vision. It creates an object template from the initial frame, employs the kernel trick for non-linear relationships, learns a correlation filter for tracking, and uses correlation scores to locate the object in subsequent frames. Given a set of training data, the KCF filter H in the frequency domain is updated by:

$$H = \frac{\sum_t y_t \cdot \phi(x_t)^*}{\sum_t \phi(x_t) \cdot \phi(x_t)^* + \lambda} \quad (2.6)$$

where y_t is the Fourier transform of the Gaussian shaped training label for frame t , $\phi(x_t)$ is the non-linear mapping of the training data x_t into a higher dimensional space, $*$ denotes complex conjugation, λ is a regularization parameter to prevent overfitting.

KCF is known for its speed and efficiency, making it suitable for real-time applications. However, it may have limitations with objects subject to occlusions or abrupt motion changes, where more advanced tracking methods may be necessary. KCF has influenced the development of other tracking algorithms.

TLD tracking method

The Tracking-Learning-Detection (TLD) [13] method is a comprehensive algorithm for object tracking that combines tracking, learning, and detection. It tracks objects, learns from their appearance changes, performs real-time detection, and includes mechanisms for recovery when tracking is briefly lost. The learning module in TLD updates the object model based on the detector's output and tracker's prediction:

$$M_{t+1} = \text{UpdateModel}(M_t, D_t, T_t) \quad (2.7)$$

Here, M_{t+1} is the updated model at time $t + 1$, M_t is the current model, D_t is the detection result, and T_t is the tracker's prediction. The function `UpdateModel` represents the process of updating the model based on the incoming information from the detector and tracker.

TLD is versatile and adaptable, but it may face challenges in complex scenarios with heavy occlusions or multiple objects of similar appearance. It has been influential in addressing complex tracking challenges by integrating these critical components.

Neural network tracking

Neural Network tracking methods represent a modern and powerful approach to object tracking in computer vision. These methods leverage deep neural networks

for tracking, offering increased accuracy, robustness, and adaptability to various tracking scenarios. Those methods use deep learning architectures, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) [14], to perform object tracking. These networks are capable of learning complex object representations, making them highly effective in handling variations in object appearance. For example Siamese network is one of the more popular architectures used for tracking. They learn to calculate similarity scores between a target object template and candidate regions in the video frames. This similarity score helps in locating the object in subsequent frames. This functionality is represented in figure 2.5.

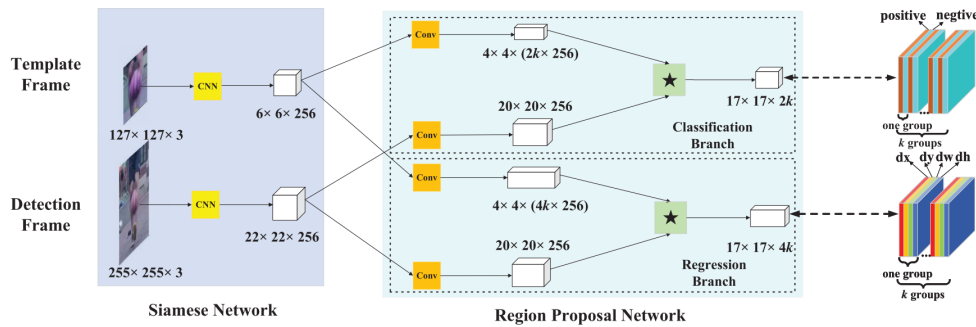


Figure 2.5: Graphical representation of SiameseRPN neural network [15]

2.4 Multiple object tracking

This part covers multiple object tracking (MOT) [16] which is important area in computer vision that focuses on detecting and tracking several objects in videos. This process involves identifying objects using bounding boxes and keeping track of each one over time with unique identification. Many MOT methods connect these boxes based on a minimum confidence level, often overlooking lower-confidence detections caused by obstacles, leading to inaccuracies and interrupted tracking paths. To tackle these challenges, there is a push for new tracking algorithms that can effectively track objects, even when they are partially hidden. MOT techniques

are generally divided into two groups: Detection-Based Tracking (DBT) [17] in figure 2.6, which starts tracking from object detections in video frames, and Detection-Free Tracking (DFT) in figure 2.7, which tracks objects without initial detections, using alternative methods like motion. This division highlights the variety of approaches within MOT to handle the complexities of tracking multiple moving objects.

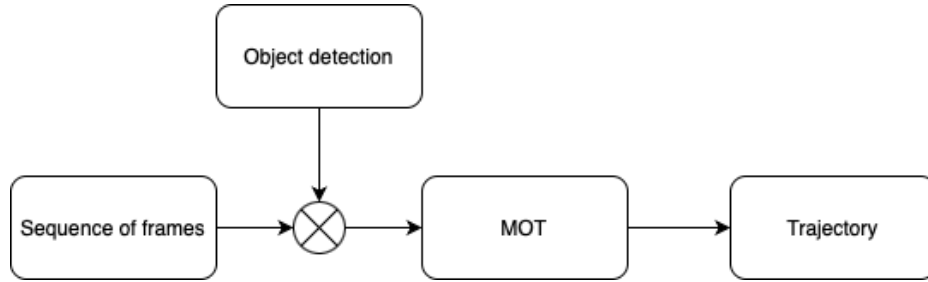


Figure 2.6: Graphical representation of DBT

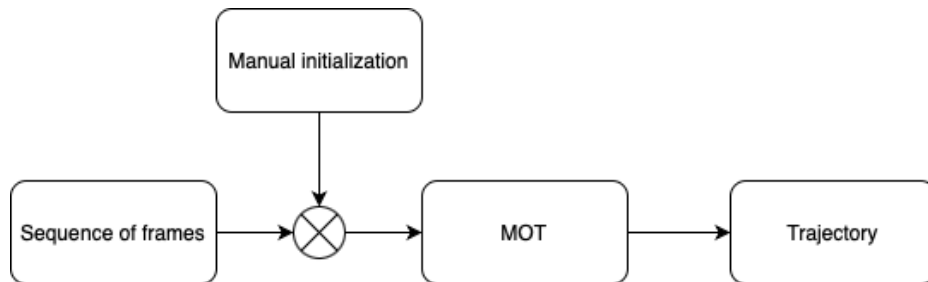


Figure 2.7: Graphical representation of DFT

2.4.1 Multiple object tracking methods

In line with the current trends in the field of computer vision, this thesis primarily explores MOT methods that leverage neural networks. Additionally, for comparative analysis, it incorporates one traditional approach. This approach allows for a comprehensive understanding of how modern neural network-based techniques measure up against conventional methods in the context of MOT.

Dense optical flow

Optical flow [18] refers to the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and the scene. It is one of the classical approaches in computer vision, used for tasks such as motion detection, object tracking, and video compression. Standard, also called spars, optical flow method focuses on computing the motion for a set of feature points between two image frames. These methods identify specific points or features within an image (such as corners, edges, or significant textures) and track their movements across frames. The essential equation that governs optical flow, derived from the assumption of brightness constancy and spatial and temporal smoothness, is given by the Optical Flow Constraint Equation 2.8.

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \quad (2.8)$$

Where I represents the image intensity, V_x and V_y represents in x and y axis, respectively $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ denote the gradients of the image intensity along x , y and t dimensions.

Dense optical flow, on the other hand, aims to compute the motion for every pixel in the image frame. This approach provides a more detailed motion estimation across the entire image, which is particularly useful for understanding complex scenes with varied motions. Dense optical flow can capture subtle movements and finer details of the scene, offering a comprehensive map of flow vectors. For dense optical flow, algorithms like the Lucas-Kanade method or the Horn-Schunck algorithm are often used. The Lucas-Kanade method for optical flow, which is implemented in OpenCV [19], estimation is based on the assumption that the flow is essentially constant in a local neighborhood of the pixel under consideration. It solves for the flow velocities V_x and V_y by minimizing the error in the brightness constancy equation over a window surrounding each pixel. The method can be formulated as solving a set of linear equations derived from the brightness constancy

assumption as seen in equation 2.9.

$$\sum_{(x,y) \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = - \sum_{(x,y) \in W} \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix} \quad (2.9)$$

Where I_x , I_y and I_t represents partial derivatives of the image with respect to spatial coordinates, V_x and V_y represent the components of the optical flow vector in x and y directions. The sum over W denotes aggregation over the pixels within a specified window, encapsulating the assumption that flow is constant or varies smoothly within this neighborhood.

DeepSort

DeepSORT [20] (Deep Simple Online and Realtime Tracking) is an advanced algorithm designed for real-time multi-object tracking, that is building upon the foundational SORT [21] (Simple Online and Realtime Tracking) algorithm by integrating deep learning features to enhance accuracy. It excels in scenarios where objects need to be tracked across frames in a video, combining motion information with appearance information extracted from a deep convolutional neural network. This dual approach significantly improves the tracking of objects, especially in challenging situations where objects may occlude each other or disappear from view temporarily. DeepSORT is particularly valued in applications requiring precise object tracking in real-time, such as in surveillance, sports analytics, and autonomous driving systems, due to its ability to maintain stable object identities over time, even in densely populated scenes.

It operates by integrating high-dimensional appearance features extracted through a deep neural network with the SORT algorithm, which improves the tracking accuracy and robustness in complex scenarios. At its core, DeepSORT utilizes a combination of Kalman filtering (2.10, 2.11) for predicting object motion.

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \quad (2.10)$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (y_k - H_k \hat{x}_{k|k-1}) \quad (2.11)$$

And a Hungarian algorithm (2.12) for assignment optimization.

$$\min \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} \quad (2.12)$$

subject to:

$$\sum_{i=1}^n x_{ij} = 1 \quad \text{for } j = 1, 2, \dots, m, \quad (2.13)$$

$$\sum_{j=1}^m x_{ij} = 1 \quad \text{for } i = 1, 2, \dots, n, \quad (2.14)$$

$$x_{ij} \in \{0, 1\}. \quad (2.15)$$

The deep learning component comes into play by generating appearance descriptors from detected objects, which are used to compute similarity scores. These scores help in associating detections across frames, even in situations of temporary occlusion or when objects closely interact. A complete DeepSORT architecture can be seen in figure 2.8. By maintaining a continuous identity for each tracked object and employing a more sophisticated matching mechanism that considers both spatial and appearance information, DeepSORT effectively addresses the limitations of purely motion-based tracking systems, making it highly effective for applications that requires a precise object tracking in real-time video feeds.

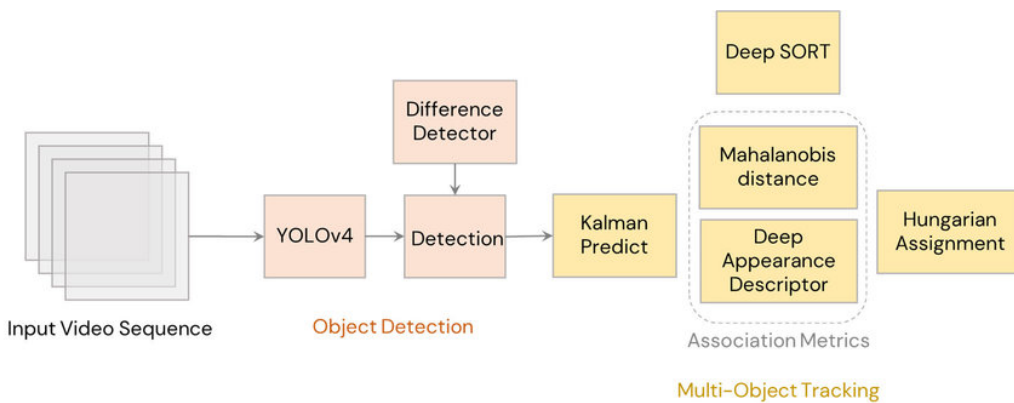


Figure 2.8: Architecture of DeepSORT [22]

Dataset

3

This chapter provides details on the dataset used in this project, including its features and how it was prepared. The quality of data is crucial in machine learning. Good data preparation is a key to building models that learn well and make accurate predictions. It involves cleaning and organizing the data to reduce errors and make sure the model gets the right information. This helps in faster training and better outcomes. Simply put, well-prepared data is the foundation of any successful machine learning project.

3.1 Detailed information

The input data are video recordings from the Biomedical Centre of the Faculty of Medicine of Charles University in Pilsen. This centre mainly focuses on a research and a development in the field of organ replacement and regeneration. Specifically, it involves video recordings of pigs from several pens. Total number of videos was 21, of which 18 were used. These videos were selected primarily on the basis of the pigs' activity, so that recording holds as many different conditions as possible. Also nighttime video footage and footage where multiple pigs were in the same pen were included. All of these video recordings were in .mp4 format. Table 3.1 shows those summed up information about dataset.

Number of videos	18
Number of frames	44 821
Total length of videos	31.12 min
Dataset size	48.97 GB

Table 3.1: Dataset information

3.2 Annotations

Data was processed using the CVAT [23] (Computer vision annotation tool), which is a free and web-based image and video annotation tool used to annotate data for computer vision algorithms. The tool currently also supports supervised learning, image classification and segmentation, and 3D annotation.

The first step in the processing was to upload the data to the project in the CVAT tool. This tool then takes each video and divides it into the appropriate number of frames so that it can be labeled. The annotation itself was then done by simply marking the position of the pig with a bounding box, which then allowed the tool to record the position of the pig for that frame. Bounding boxes were used to represent detected objects. The following table displays the class names alongside the respective counts for each class.

Class name	Number of occurrences
pig_shape	79708
feeder_empty	52712
feeder_full	9893
drinker	96707

Table 3.2: Information about classes

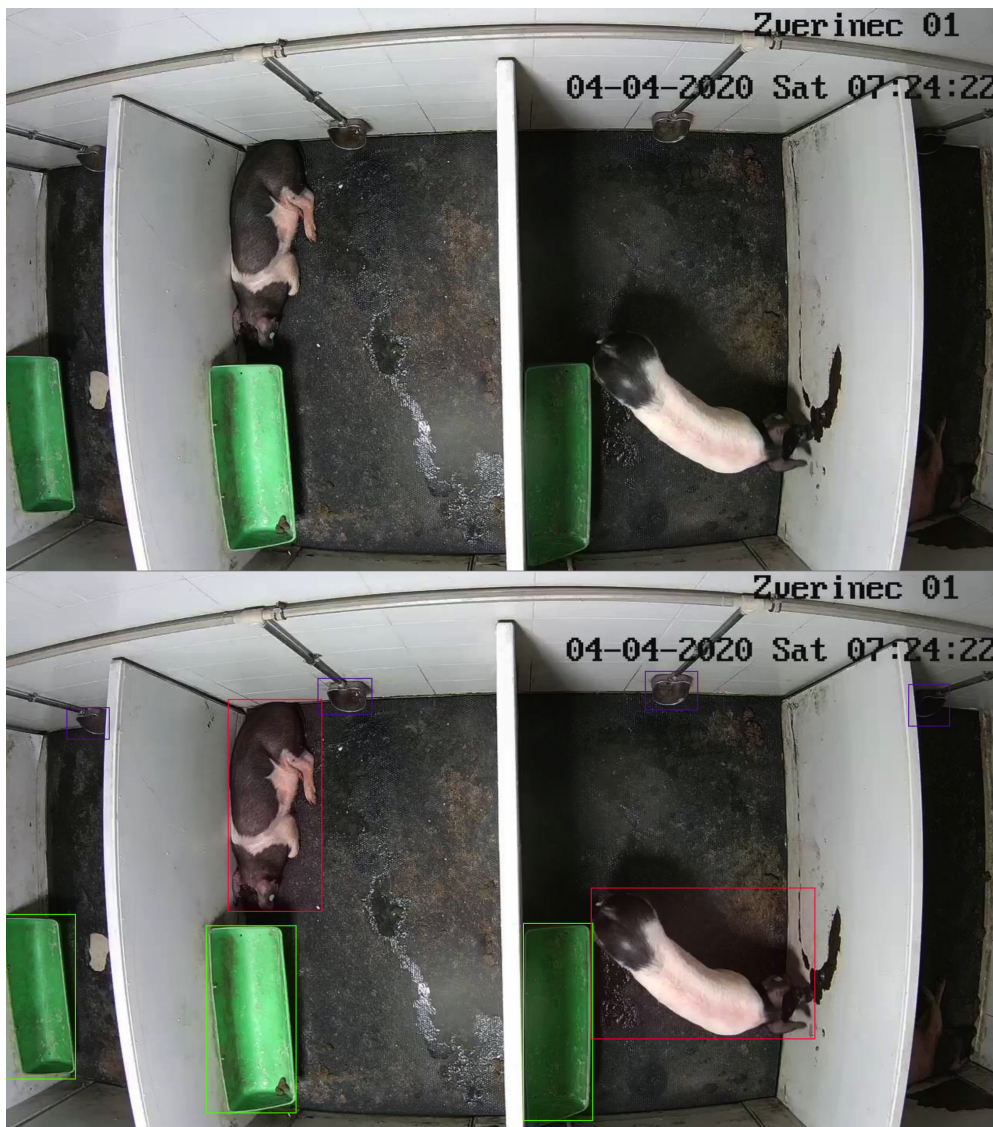


Figure 3.1: Example of input frame and annotated frame

Annotations were exported in COCO [24] format to json file. An example of annotations can be seen below. Each image within the dataset is distinctly identified by a specific ID and is accompanied by a set of parameters underlining its characteristics, such as width, height, and file name. In parallel, each annotated object within these images is linked to its corresponding image by image ID and carries a comprehensive array of attributes, including bounding box coordinates, object class, and the area encompassed by its bounding box.

3. Dataset

```
1 "images": [{"id": 1,
2           "width": 1280,
3           "height": 720,
4           "file_name": "frame_000000.PNG",
5           "license": 0,
6           "flickr_url": "",
7           "coco_url": "",
8           "date_captured": 0}]
9
10 "annotations": [{"id": 2,
11                  "image_id": 1,
12                  "category_id": 2,
13                  "segmentation": [],
14                  "area": 19137.7,
15                  "bbox": [0.0, 440.5, 90.7, 211.0],
16                  "iscrowd": 0,
17                  "attributes": {"occluded": false,
18                                 "rotation": 0.0}},
19                  {"id": 3,
20                  "image_id": 1,
21                  "category_id": 3,
22                  "segmentation": [],
23                  "area": 27577.000000000007,
24                  "bbox": [258.29, 455.14, 115.0, 239.8],
25                  "iscrowd": 0,
26                  "attributes": {"occluded": false,
27                                 "rotation": 0.0}}
```

After the preprocessing phase, which included manual annotations, the dataset was partitioned into three distinct subsets: one designated for training, another for testing, and a third for validation. This division into three sets serves the purpose of enabling effective model training, assessing its generalization capabilities, and refining it for optimal performance on unseen data, ultimately enhancing the model's robustness and reliability.

Experiments

4

This section focuses on the experiments conducted in this thesis. Detailed explanations of the experiments will be provided, including how they were carried out, the tools and methods used, and the results obtained. The goal is to give a clear and comprehensive view of the experiments, their importance, and the insights they provide, contributing to the overall thesis.

4.1 Detection model

To address the specific requirements of this thesis, a new detection model was developed, primarily to accommodate additional classes. The training dataset was described in the chapter 3. Two distinct fine-tuning approaches were undertaken. The first involved fine-tuning with MMDetection, using Faster-RCNN and DETR as a backbone, while the second employed Detectron2, using just Faster-RCNN.

4.1.1 MMDetection

MMDetection [25] is an open-source object detection toolbox based on PyTorch. It is part of the OpenMMLab project, which aims to develop an unified codebase for object detection, segmentation, and instance recognition. It is known for its modularity, flexibility, and scalability. It supports a wide range of object detection algorithms, including one-stage detectors, two-stage detectors, and cascade detectors. It also supports a variety of common datasets, such as COCO, Pascal VOC, and ImageNet.

DETR

Initially, my attempt to train a DETR model faced challenges because the used dataset was too small and lacked variety, despite the images being correctly labeled. This issue made it difficult for the model to effectively learn and identify the intended objects. The training process took over 26 hours to complete, with no applicable results.

Given that the FasterRCNN model had shown good results with the same dataset, I decided to halt my work with the DETR model and concentrate solely on developing with FasterRCNN for my thesis.

FasterRCNN

Following the decision to shift my focus, I continued thesis by working with the Faster R-CNN model. Training was performed on MetaCentrum [26], which is virtual organization, that operates and manages distributed computing infrastructure consisting of computing and storage resources owned by CESNET as well as those of cooperative academic centers within the Czech Republic. Training information is summed up in table the 4.1. Number of epochs was set to 10 with regard to model complexity, to address optimal computational time and to prevent overfitting. Number of iterations per epoch was calculated as shown in equation below:

$$\text{number of iterations} = \frac{\text{number of images}}{\text{number of images per iteration}} \quad (4.1)$$

Initial learning rate, optimizer and augmentations were default to MMDetection framework.

Epochs	10
Iteration per epoch	21 163
Initial learning rate	0.02
Optimizer	Momentum SGD
Augmentations	random flip, resizing

Table 4.1: Training parameters

The training process finished after 21 hours and 41 minutes. Training outcomes are quantified using the AP (Average Precision) metric for IoU (Intersection over Union), a pivotal metric in object detection. AP evaluates the overlap between predicted and actual bounding boxes and aggregates precision, recall, and model confidence into a singular measure of performance. It details accuracy per object class, condensing the Precision-Recall curve into a numerical summary. Specifically, metrics AP50 and AP75, highlighted in table 4.2, offer insights into model accuracy at 50% and 75% IoU thresholds. AP50 focuses on the model’s object detection capability, requiring a minimum 50% overlap between bounding boxes for a positive identification. AP75 sets a stricter criterion of 75% overlap, underscoring the model’s ability for precise object localization.

Epoch	test-0.50	test-0.75	val-0.50	val-0.75
1	0.8423	0.7867	0.8420	0.7870
2	0.8434	0.7680	0.8430	0.7680
3	0.8251	0.7603	0.8250	0.7600
4	0.8439	0.7508	0.8440	0.7510
5	0.8118	0.7632	0.8120	0.7630
6	0.7976	0.6982	0.7980	0.6980
7	0.8057	0.6931	0.8060	0.6930
8	0.8365	0.7605	0.8370	0.7610
9	0.7980	0.7261	0.7980	0.7260
10	0.8536	0.7498	0.8540	0.7500

Table 4.2: IoU statistic over training - MMDetection

4.1.2 Detectron2

Detectron2 [8] is a state-of-the-art open-source computer vision library developed by Facebook AI Research that serves as a powerful and flexible framework for building and training object detection, instance segmentation, and other computer vision models.

FasterRCNN

Upon realizing that the dataset was not suited for training the DETR model, the decision was made to train just the Faster R-CNN model. As with previous experiment, the training process was carried out on the MetaCentrum infrastructure, leveraging its computational resources. Detailed information regarding the training parameters, settings, and configurations used during this phase is summarized in the table 4.3. As previously number of epochs was set to 10 with regard to model complexity, to address optimal computational time and to prevent overfitting and to match MMDetection settings. Number of iterations per epoch was also calculated as $number_of_images/number_of_images_per_iteration$. Initial learning rate, optimizer and augmentations had to be changed to match MMDetection settings.

Training outcomes are quantified in table 4.4 and the evaluation of the model's performance adhered to the same metrics as previously established to ensure consistency.

Epochs	10
Iteration per epoch	21163
Initial learning rate	0.02
Optimizer	Momentum SGD
Augmentations	random flip, resizing

Table 4.3: Training parameters

Epoch	test-0.50	test-0.75	val-0.50	val-0.75
1	0.8358	0.7513	0.8199	0.7540
2	0.8344	0.7441	0.8183	0.7438
3	0.8120	0.7342	0.8118	0.7339
4	0.8119	0.7338	0.8112	0.7331
5	0.8099	0.7269	0.8069	0.7250
6	0.8071	0.7232	0.8001	0.7123
7	0.8189	0.7401	0.8091	0.7322
8	0.8282	0.7425	0.8180	0.7419
9	0.8375	0.7487	0.8123	0.7489
10	0.8298	0.7312	0.8259	0.7398

Table 4.4: IoU statistic over training - Detectron2

4.1.3 Comparison of MMDetection and Detectron2

I will start by comparing MMDetection and Detectron2 based on their documentations, looking at their features and how they stack up against each other. After that, I will compare the outcomes I got from training models with each framework, focusing on how well they performed. Finally, I will provide subjective comparison based on personal experience using both frameworks, touching on how user-friendly and flexible I found them to be. This way, a clear picture of both frameworks from a technical standpoint and from my own hands-on perspective will be provided.

First I will provide tables of how well each framework works based on performance, training speed, inference speed and training memory. Those results were obtained on same machine with this hardware specifications - processor: Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, GPU: 8x NVIDIA Tesla V100 (32G) and those software specifications: Python 3.7, PyTorch 1.4, CUDA 10.1, CUDNN 7.6.03, NCCL 2.4.08.

Performance

In table 4.5 below the learning rate schedule defines how the learning rate changes over time (or over epochs/batches) and the numbers in Detectron2 and MMDetection columns refers to performance metrics, specifically mean Average Precision (mAP) percentages, which are standard for evaluating object detection models.

Model	LR schedule	Detectron2	MMDetection
Faster R-CNN	1x	37.9	38.0
Mask R-CNN	1x	38.6 & 35.2	38.8 & 35.4
RetinaNet	1x	36.5	37.0

Table 4.5: Performance details of each framework

Training speed

The speed of model training is gauged by the metric "seconds per iteration" (s/iter). A lower value for this metric is preferable, as it indicates faster training per iteration. Table 4.6 illustrates the results.

Model	Detectron2	MMDetection
Faster R-CNN	0.210	0.216
Mask R-CNN	0.261	0.265
RetinaNet	0.200	0.205

Table 4.6: Training speed details of each framework

Inference speed

The metric used to assess the inference speed of a model is frames per second (fps), also referred to as images per second (img/s), when running on a single GPU. Higher fps values indicate better performance, as more images can be processed in less time. In alignment with Detectron2's reporting standards, raw inference speeds is presented, which do not account for the time taken to load data. For the Mask R-

CNN model, the time taken for Run-Length Encoding (RLE) in post-processing is also not included. Results are shown in the table 4.7

Model	Detectron2	MMDetection
Faster R-CNN	25.6	22.2
Mask R-CNN	22.5	19.6
RetinaNet	17.8	20.6

Table 4.7: Inference speed details of each framework

Training memory

Values in columns Detectron2 and MMDetection refer to GB (gigabytes). Smaller number indicates less memory allocated during training process. Obtained results are show in table 4.8

Model	Detectron2	MMDetection
Faster R-CNN	3.0	3.8
Mask R-CNN	3.4	3.9
RetinaNet	3.9	3.4

Table 4.8: Training speed details of each framework

Thesis result comparison

Table below 4.9 shows a comparison of training results performed on test set with the usage of AP50 metrics. Table below shows, that better results got MMDetection framework, that was later used in application development, even though the inference speed was a little worse.

Epoch	Detectron2	MMDetection
1	0.8358	0.8423
2	0.8344	0.8434
3	0.8120	0.8251
4	0.8119	0.8439
5	0.8099	0.8118
6	0.8071	0.7976
7	0.8189	0.8057
8	0.8282	0.8365
9	0.8375	0.7980
10	0.8298	0.8536

Table 4.9: IoU statistic over training - Detectron2

4.2 Single object tracking

This section will cover comparative analysis of different SOT methods. At the end of this section a table will provide an overview, where precision, calculated using average Intersection over Union (IoU), and time, representing method performance, are key metrics for evaluation. This comparison sheds light on the effectiveness and efficiency of these SOT techniques. This section will also present a brief description of implementation of methods presented in section that describes single object tracking methods.

4.2.1 Implementation of methods

Each of the classical tracking methods were implemented using OpenCV framework. All methods were given the same initial bounding box, that was provided by detection neural network and same input video, from dataset provided in chapter 3. Every new bounding box provided by OpenCV trackers was stored in python list, that was exported to file, so it can be evaluated later. The only neural network method was implemented using MMTracking framework, where the initial bounding box and input video was same as in OpenCV implementation and the output, as

file containing bounding boxes was also the same. To determine a code runtime a python time library was used. The input video was 69 seconds long.

4.2.2 Results of single object tracking methods

All obtained results are stored in table below. Last row of table contains results for single object tracking that was achieved by using neural network model Faster-RCNN trained on specific data.

Method	Elapsed time [s]	Average IoU [%]
Mean shift	4.05	2.19
MOSSE	2.23	55.01
KCF	5.49	59.95
TLD	31.60	30.31
NN (SiameseRPN)	20.74	70.98
Detection per frame	56.38	94.62

Table 4.10: Comparison of used methods

Based on the obtained results, it becomes evident that classical tracking techniques are generally faster but not as precise. An exception to this trend is the TLD tracking method, which is a classical approach that not only adjusts the bounding box position but also accounts for changes in its appearance. However, this method struggles with larger and more erratic movements, making it less effective than the others. As anticipated, the discriminative tracking method proved to be the least accurate, mainly due to its simplicity. On the other hand MOSSE and KCF tracking methods were quite fast and accurate. After watching the output video, it could be assumed that those methods were tracking the object really well, but due to the change of bbox appearance the IoU metric was not as high. The best results were obtained by NN method - specifically by used SiameseRPN model. Although the code runtime for this method was longer than for classical methods, considering the video's length, these results are entirely acceptable.

4. Experiments



Figure 4.1: Example of Mean-Shift tracking output



Figure 4.2: Example of TLD tracking output



Figure 4.3: Example of KCF tracking output

4.3 Multiple object tracking

This section covers a detailed comparative analysis of various MOT techniques. It concludes with a summary table that highlights two critical metrics for evaluation: precision, measured by average Intersection over Union (IoU), and performance time. This table allows for a clear comparison of the effectiveness and efficiency of these MOT strategies. Additionally, this section will include a concise explanation on the implementation of the methods introduced in the section on MOT methods.

4.3.1 Implementation of methods

In the previous part an OpenCV framework was used for classical methods, this goes same for the dense optical flow used here, which, while is not a tracking method itself, serves as a tool to analyze the motion between consecutive frames. Since optical flow alone does not track objects, a basic tracking mechanism was integrated into a code. This simple tracking logic was essential for maintaining continuity in object identification across frames, building on the motion information provided by optical flow to assign and update object positions. Tracking mechanism implemented assigns unique IDs to detected objects in video frames and tracks their movement across frames based on the Euclidean distance between their centroids. Initially, each detected object is identified by a bounding box. For each new frame, this mechanism calculates the centroids of new detections and compares them to the centroids of previously tracked objects. If the distance between a new centroid and an existing tracked object's centroid is below a specified threshold, the new detection is considered to be the same object, and thus retains the original ID.

For the implementation of other method, neural network-based in this case, I used the MMTracking framework [27], a complex library designed for both single and multi-object tracking tasks. To prepare these methods to my specific requirements, adjustments to the configurations files and certain sections of the code were necessary to achieve the desired output. Those adjustments were crucial for adapt-

ing the tracking algorithms to work efficiently with the unique characteristics of my dataset and objectives.

Second used method was DeepSORT. Modifications were made to integrate it with a custom detection model that I had previously trained on videos of pigs in a confined space. This model was crucial for accurately detecting objects, pigs in my case, in the challenging conditions of a confined space, where lighting, occlusion, and the animals' proximity to each other could significantly impact detection performance. Additionally, for the re-identification (Re-ID) process within DeepSORT, I opted for a TRACTOR [28] approach, that was chosen for its effectiveness in distinguishing individual subjects based on unique features, an essential capability for maintaining consistent tracking of each pig over time, despite the potential for occlusion and movement within the confined space.

These modifications and the choice of specific models for detection and Re-ID processes were instrumental in optimizing the tracking performance for the specific scenario of monitoring pigs in a confined space. This used approach facilitated the extraction of accurate and meaningful data from the video footage, enabling detailed analysis and insights into the behavior of pigs in confined environments.

Upon realizing that the Dense Optical Flow technique fell short in terms of performance, subsequent development efforts were concentrated on the DeepSort method. In earlier stages, as modifications to this method were implemented, particular attention was paid to ensuring that the format of its output is standardized. This meant aligning the data structures and types produced by this method to be always the same, a strategic decision that aimed at facilitating a more efficient development process. This standardization of outputs was critical for several reasons. Firstly, it allowed easy integration of data into subsequent stages of the code, ensuring that any tools or analyses developed could be universally applied without needing adjustments specific to each method. Secondly, this uniformity in output formats simplified the comparison and evaluation, making it easier to assess its strength and weakness.

4.3.2 Results of multiple object tracking methods

All obtained results are stored in table below. Last row of table contains results for multiple object tracking that was achieved by using neural network model Faster-RCNN trained on specific data.

Method	Elapsed time [s]	Average IoU [%]
Dense optical flow	7.03	4.67
DeepSort	232.23	89.79
Detection per frame	56.38	94.62

Table 4.11: Comparison of used MOT methods

From those results it is clear that classical multiple object tracking technique offers speed but lack the precision of neural network methods. The primary drawback of classical approach lies in accurately detecting the correct object. For instance, dense optical flow methods, while fast, are not ideal for this task. It only detects moving objects, which can be seen in figure 4.4 at position (0,1), and even then, it struggles to identify individual objects like a single pig as a complete entity, which is evident when the algorithm only detects head, or part of pig's body. This can be seen in the figure 4.4 at position (0,0) where the flow map is visualized and showing only part of one pig's body, while the second pig remains still. Flow map, sometimes referred to as a "heatmap" or color wheel, is a visualization where color represents the direction of motion and intensity (brightness or saturation) indicates the speed of motion. This color-coding is done using the Hue, Saturation, Value (HSV - "H" meaning Hue and representing direction of motion, "S" meaning saturation and "V" meaning value and representing magnitude or speed of motion) color space.

In the figure 4.4 at position (0,1) is a final output of Dense optical flow method. It is worth mentioning that tracking output represented by bounding box is only detected on non-static part of a pig's body, while the rest of its body remains undetected just as with second pig, which is not moving at all. In case, where one pig is moving, can be seen, that this method is able to detect and track properly with

only a little noise, in this case, represented by small bounding box inside of bigger bounding box. If I were to proceed with this method, then a filtering method would be implemented, which would likely help in removing noise from final results, as smaller, less significant objects, that can often create false positives or irrelevant information, would be filtered.

Continuing with the evaluation of the Deep SORT, it becomes apparent that this method, although slower, offers significantly enhanced precision in terms of IoU metrics. Despite its reduced speed, the computing performance is still meeting the requirements for the purposes outlined in this thesis. The algorithm demonstrates exceptional accuracy, particularly in scenarios where only a single pig is present within each pen, that aligns with the video datasets to be analyzed in the application. These outcomes are illustrated in figure 4.4 at positions (1,0) and (2,1).

An instance of tracking inaccuracies is observed when multiple pigs are present within the same enclosure. Under such circumstances, although the pigs are initially detected and tracked accurately, their proximity causes the tracking bounding boxes to merge into one temporarily. Consequently, upon separation, there is occasionally a swap of identity tags among the individuals. This scenarios are visualized in figure 4.4 at positions (1,0), (1,1) and (2,1). However, as previously noted, the scenario involving more than one pig per pen will not be happening in the operational setting of the application. Thus, the Deep SORT algorithm is well-optimized and nearly ideal for its intended deployment in the final application.

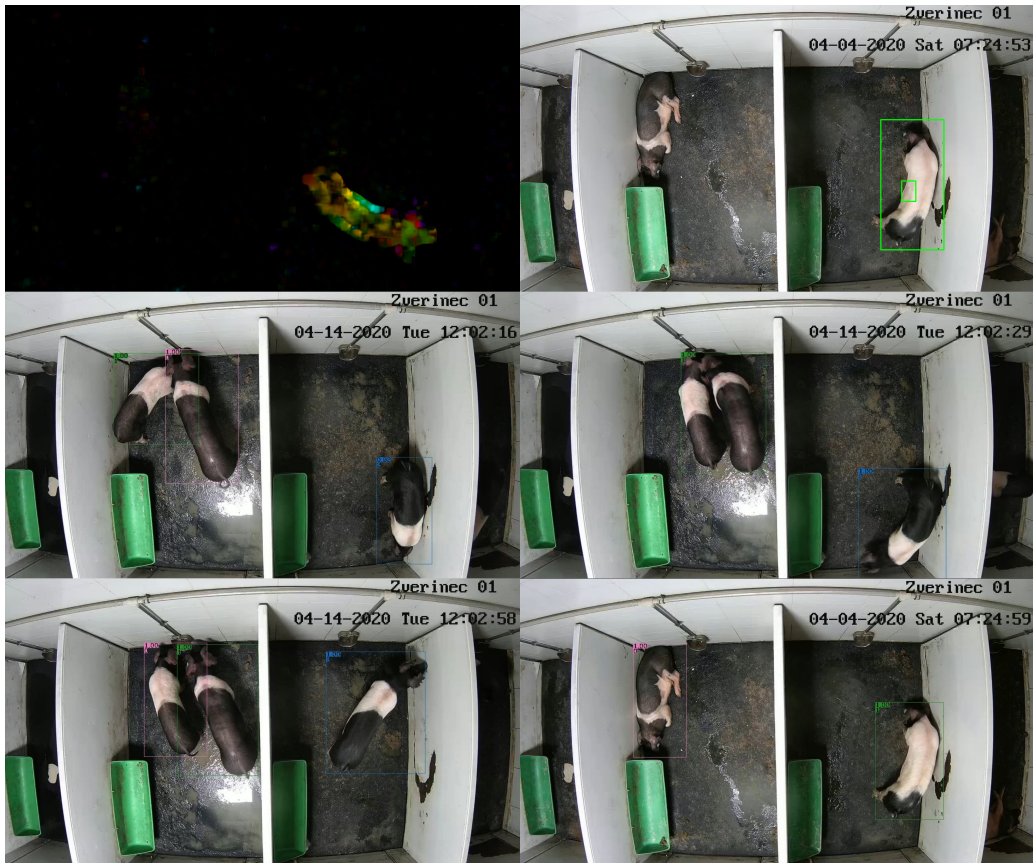


Figure 4.4: Example of MOT outputs

Web Application

5

In this chapter a final result of this thesis will be presented as a user-friendly web application designed to monitor the movements of pigs, facilitating tracking of their medical condition. This application provides a streamlined platform for doctors to observe and analyze pig behavior, aiding in the early detection of potential health issues.

5.1 Application structure

The web application was developed using the Flask framework, effectively integrating backend frameworks from previous experiments. The frontend is crafted with HTML and CSS, providing an intuitive user interface. To enhance scalability and deployment efficiency, the entire application is containerized using Docker, distributed across three distinct containers. The first container hosts the Flask web application, serving as the main interface for users. The second container is dedicated to video processing, handling computational tasks related to video manipulation and analysis. Lastly, the third container is utilized for MongoDB [29], offering a robust and scalable database solution. This architecture not only simplifies deployment across different environments but also facilitates independent scaling and maintenance of each component, ensuring high availability and performance of the application. Whole solution represented by diagram is shown in figure below.

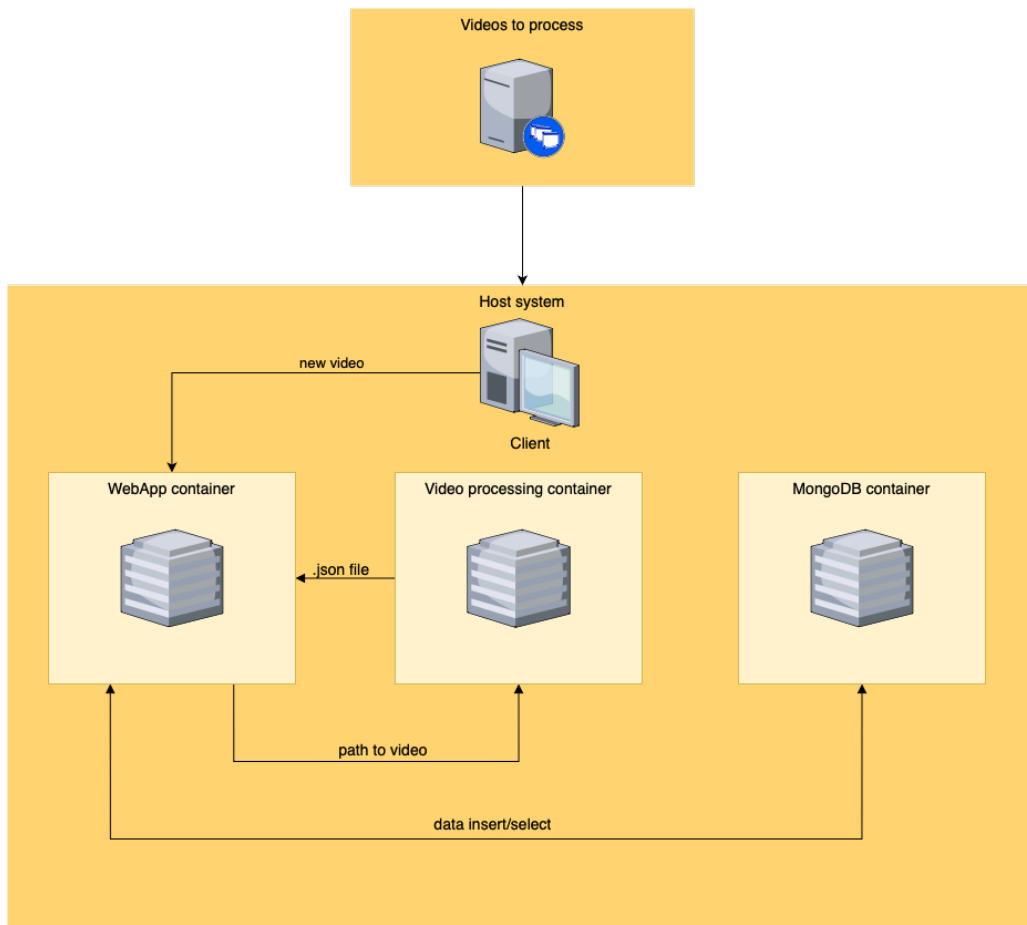


Figure 5.1: Simple app structure diagram

5.1.1 Frontend

As mentioned earlier, the web application’s frontend is meticulously designed using HTML and CSS, focusing on delivering content with precision and aesthetic appeal. The design philosophy centers around simplicity and clarity, aiming to offer an effortless and engaging browsing experience. Instead of streaming live video, the application provides access to a collection of pre-processed videos. This decision was made to ensure that the application will run smoothly, with possible playback. These videos come enhanced with bounding boxes around each pig, allowing for easy identification and analysis. Users have the flexibility to play these videos, observing the highlighted pigs within their environments.

In addition to video playback, the application introduces a feature for in-depth analysis. Users can explore various graphical representations, such as trajectories illustrating the paths walked by each pig or bar graphs detailing the distance covered by individual pigs. This analytical component enriches the user experience by providing meaningful insights into the behavior and movement patterns of pigs, making the data not only accessible but also comprehensible. The application comprises three main pages: the homepage, where users can select and view videos alongside their analyses; an informational page detailing the thesis and the research it encompasses and a video detail page. This structured design ensures navigational ease, allowing users to seamlessly interact with the application's diverse features and content.

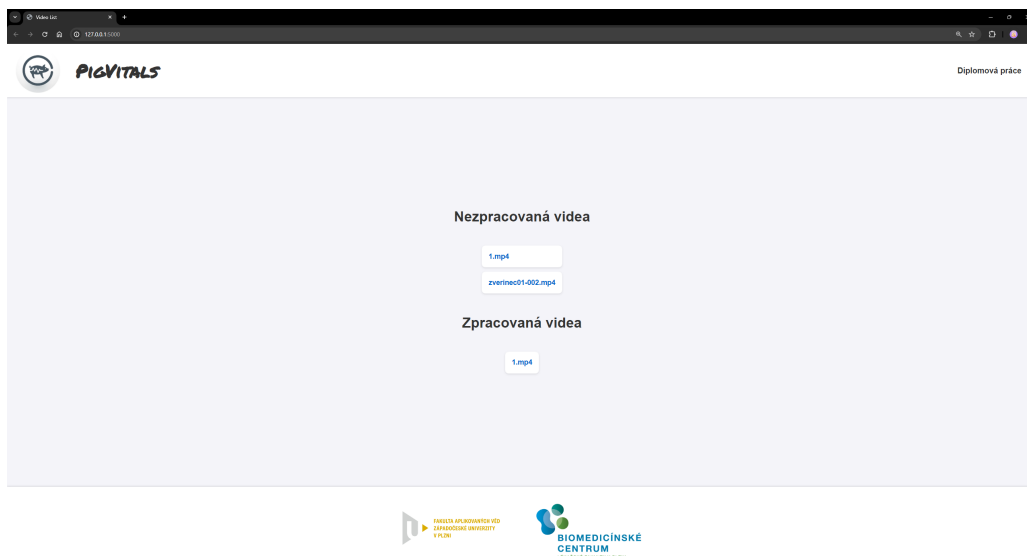


Figure 5.2: Web application - homepage

5. Web Application

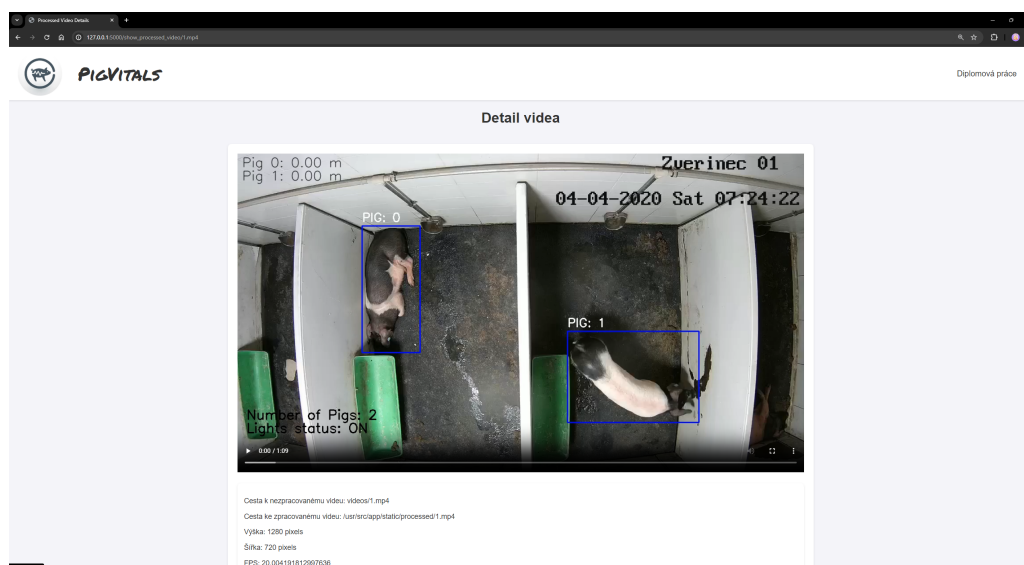


Figure 5.3: Web application - video detail

5.1.2 Backend

In the backend architecture of the application, a combination of technologies and frameworks is employed across three dockerized containers to optimize the processing, analysis, and management of data.

The first and main container is hosting the Flask web application. This container is responsible for several critical functions: it handles requests for videos, performs data analysis to generate graphical reports for the frontend, manages communication with the MongoDB database for data storage and retrieval. This multifaceted container ensures the seamless integration of frontend requests with backend processing and data management.

The second container operates as a processing unit. It runs scripts that are using the MMDetection and MMTracking frameworks, taking the path to a video as input. These frameworks are instrumental in analyzing the movement of pigs within the videos. After processing, the container outputs JSON-formatted data, which encapsulates the analyzed information, ready for further use by the web application in generating insights and visual representations.

The third container is dedicated for hosting a local MongoDB database. This database acts as the central repository for storing and managing the vast amounts of data generated and used by the application. By containerizing the MongoDB instance, the application ensures data persistence, scalability, and efficient data access, facilitating robust data management practices that support the application's complex data processing and analysis needs.

Together, these containers form a cohesive backend system that is not only modular and scalable but also capable of handling complex data analysis, storage, and communication tasks with efficiency and reliability. This backend structure supports the application's objective of providing detailed insights into the behavior and health of pigs through advanced data processing and analysis techniques.

5.2 Application features

This section provides a detailed overview of the many features integrated into the application, highlighting their implementation. The application offers a wide range of functionalities, each developed to enhance user experience and effectiveness.

Main feature of this application is tracking of pigs, more specifically drawing of bounding box for each pig in video. Since this implementation is already described in previous sections 4.1 and 4.2.1, this section will cover other application features, leveraging upon data obtained by the main feature.

5.2.1 Data storage

The application employs local MongoDB for the storage of data, capitalizing on its flexible schema and robust performance to manage diverse data formats efficiently. MongoDB's document-oriented nature allows the application to store data in a JSON-like format, which is both flexible and scalable. This ensures that data captured from various inputs is consistently formatted, facilitating easier data manipulation and retrieval. This approach enhances the application's capability to handle large

volumes of data seamlessly, while maintaining high performance and reliability. An example of stored data is shown below.

```
1 "object_ID": "123456",
2 "processed_path": "usr/src/processed_videos/zverinec1-001.mp4",
3 "original_path": "usr/src/videos/zverinec1-001.mp4",
4 "width": "1280",
5 "height": "720",
6 "data": {
7     "0": {
8         "track_bboxes": [
9             [
10                0,
11                286.43,
12                165.18,
13                119.9,
14                271.61
15            ]
16        ]
17    }
18 },
19 "scale_factor": "0.78",
20 "fps": "25",
21 "frame_count": "1350",
22 "camera": "zverinec01",
23 "date_processed": "2024-04-12 12:00:00"
```

5.2.2 Status of lights

To gather and display information about lighting conditions, a straightforward Python method that returns boolean values indicative of the lighting situation was developed. Essentially, this method returns *True* if the lights are on and *False* when they are off. This approach provides effective way to understand whether the environment was illuminated or not, enabling a clear and immediate insight into the

lighting status, without the need of seeing an actual video output. Another simple, yet useful feature providing practical information shown in video description.

The input for this method is a frame that is initially converted to gray-scale, followed by calculating the average brightness of all of its pixels. Utilizing this average brightness and a experimentally predetermined threshold the method then returns a boolean value. Equation representing this functionality is shown below.

$$\text{output} = \begin{cases} \text{True,} & \text{if } \frac{1}{N} \sum f(i, j)_{\text{grayscale}} > T, \\ \text{False,} & \text{otherwise.} \end{cases} \quad (5.1)$$



Figure 5.4: Lights on video on left and lights off video on right

5.2.3 Pig counter overview

Another simple feature of this application is the "pig counter" overview. This functionality is achieved by summing IDs in tracking algorithm output which is returned as .json file returned by video processing container. While a simple feature, it still provides a fine and useful insight for user, that can be shown in video description.

5.2.4 Distance walked by each pig

Next feature aimed to accurately visualize the distance each pig has traveled within a given environment. To do that pretrained object detection model was deployed to identify and locate the "feeder" object within the observational field, then coordinates of the "feeder" object's bounding box were extracted to determine the

pixel dimensions of the "feeder" object by calculating the differences between the bounding box coordinates.

After that a comparison of the pixel dimensions of the "feeder" object to its known real-world dimensions is made, to establish a pixel-to-physical dimension conversion ratio - scale factor. This conversion is represented in equations 5.2 and 5.3, where l represents the length feeder bounding box in the image, calculated as the Euclidean distance between two points within the image. This length is measured in pixels. L denotes the real-world length of the line. This length is known from measurements and is expressed in real-world units, specifically in meters. S is the scale factor, which quantifies the number of real-world units represented by each pixel in the image.

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (5.2)$$

$$S = \frac{L}{l} \quad (5.3)$$

This conversion ratio is used to translate the distances traveled by the pigs, measured in pixels between bounding box centroids, into real-world distance units. Conversion ratio is returned in .json file as one of the parameters, because it will be used in reporting part.

5.2.5 Data export

Another notable feature of the application is its ability to export data pertinent to a specific video upon the press of a button within the app. This function converts data from JSON format stored in MongoDB into a CSV file with usage of Python framework Pandas [30]. This transformation is crucial for users, as the CSV format is notably easier to read and manage. The exported CSV file is structured with columns that include *camera*, *pig_id*, *pig_position*, *light_status* and *time_stamp*. This structured data presentation allows for a straightforward analysis and review, enabling veterinarians to quickly assess the environmental and behavioral contexts

captured in the video footage. An example of how the exported data are structured is shown in table 5.1.

Camera	Pig_ID	Pig_Position	Light_Status	Time_Stamp
Zverinec1	001	(x1,y1,x2,y2)	1	2023-04-01 12:00:00
Zverinec2	002	(x3,y3,x4,y4)	0	2023-04-01 12:05:00

Table 5.1: Example of CSV file structure exported from the app

5.2.6 Reporting

The final and perhaps most significant feature introduced is reporting. This feature utilizes data provided by the video processing container, which are stored in the MongoDB database. Web application container retrieves this data through queries to MongoDB, leveraging it to create detailed reports. To present this data in an engaging and informative manner, the Python framework Plotly [31] is employed for its ability to generate interactive graphs. This approach not only enhances the presentation of data but also allows users to interact with the information for deeper insights.

For the purposes of this thesis, two specific types of graphs were implemented. The first is a bar graph that quantifies and displays the distance each pig has walked, providing a clear visual representation of their activity levels in meters. Data for this graph are obtained by summing the distances between two consecutive bounding boxes.

$$D_{pigID} = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (5.4)$$

The second graph is designed to visualize the trajectories of each pig within their pen, mapping out the paths they have taken. It is created by adding the centroids of bounding boxes that represent the location of each pig over time, to a list representing the trajectory. The centroid of a bounding box is determined by the equation 5.5, where (x_1, y_1) and (x_2, y_2) are the coordinates of the top-left and bottom-right

5. Web Application

corners of the bounding box.

$$(c_x, c_y) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right) \quad (5.5)$$

Creation of the final list is represented by the equation 5.6. This list serves as a set of simplified positional data points that is used to create the visualization of trajectory.

$$C_i = \left(\frac{x_{1i} + x_{2i}}{2}, \frac{y_{1i} + y_{2i}}{2} \right) \quad \text{for } i = 1 \text{ to } n \quad (5.6)$$

The data, having been standardized by the video processing phase and subsequently stored in MongoDB, facilitates the creation of enhanced reporting in the future, thereby enabling doctors to derive deeper insights. These visualizations provide observations into the behavioral and movement dynamics of the pigs, augmenting the comprehensive understanding of their physical activities. Notably, this reporting functionality emerges as an integral element of the application, presenting users with a mechanism for the analysis and interpretation of the data obtained during the video processing stage.

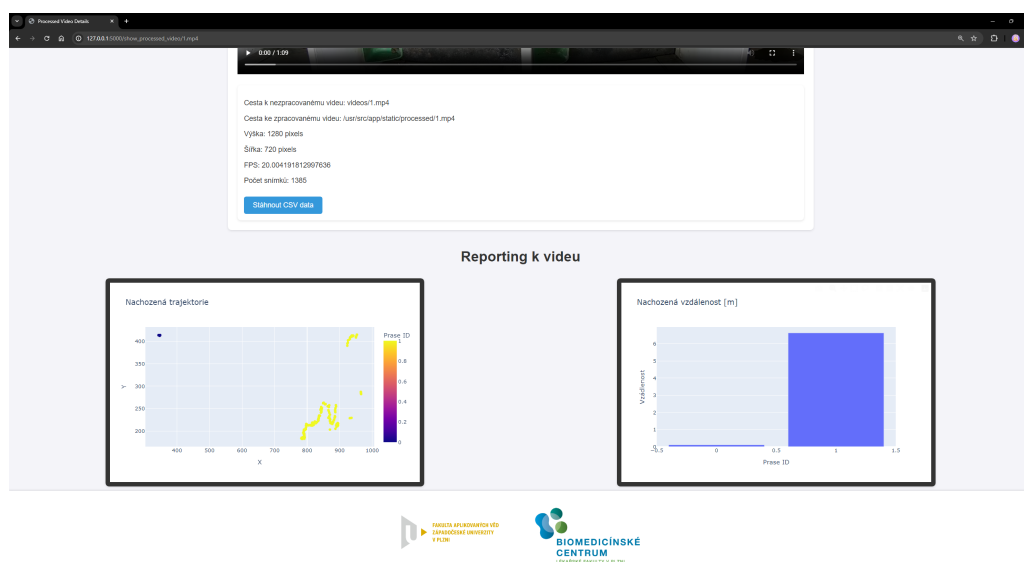


Figure 5.5: Web application - reporting

Walked distance

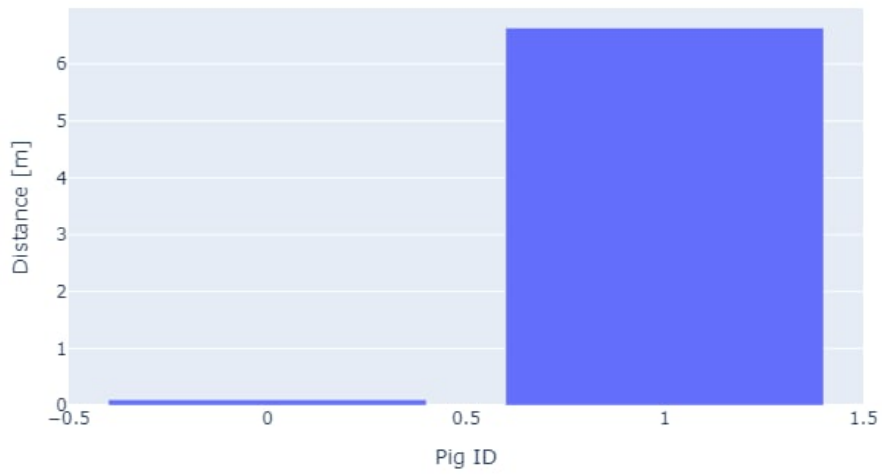


Figure 5.6: Reporting - walked distance

Walked trajectory

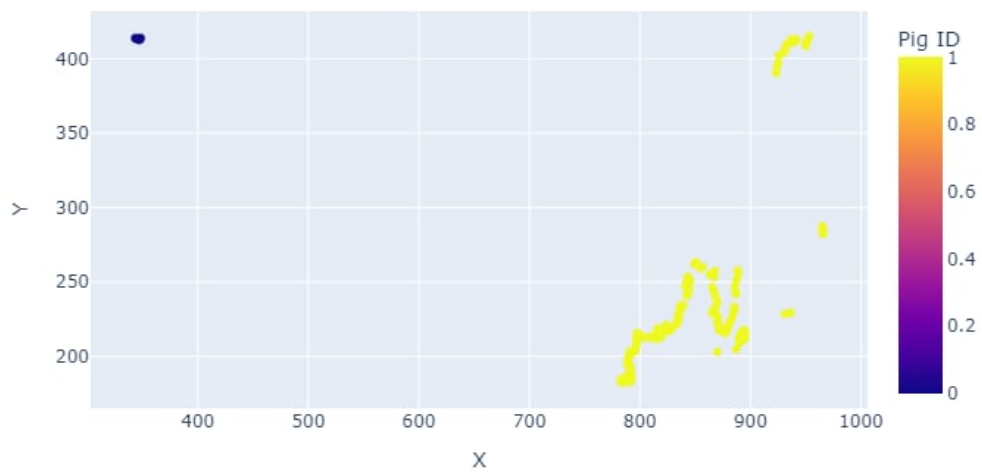


Figure 5.7: Reporting - walked trajectory

Conclusion

6

In conclusion, this thesis dealt with a problem of development and implementation of application for pig tracking that will help doctors and researchers with monitoring of pigs health state. By researching classical and current methods in the field of computer vision in first part, analyzing real-world data in second part, and overcoming practical challenges in last part, this thesis underlines the role of technology in enhancing the precision and efficiency in biomedical research.

The collaboration with the Biomedical Center of Faculty of medicine of Charles University in Pilsen provided a foundation for this thesis, offering dataset that facilitated the development and validation of the application. The process of video annotation and the subsequent implementation of object tracking and data analysis technologies underscore the potential for specific, technology-driven solutions to biomedical research, which is conducted at the Biomedical Center of Faculty of medicine of Charles University in Pilsen.

In the practical segment of this thesis, an extensive exploration and evaluation of various methods for single object tracking and multiple object tracking were conducted. To augment the functionality of some of these methods, it became necessary to develop and integrate a detection algorithm. A notable instance of this approach involved the DeepSORT method, which combines the Faster R-CNN pretrained detection model with the SORT algorithm. This methodology demonstrated good performance in tracking accuracy and efficiency, making it the preferred choice for incorporation into the application's development.

The last section of this thesis is dedicated to the web application that repre-

6. Conclusion

sents the culmination of this research. It details the operational functionality, design principles, and key features of the application, highlighting its utility and efficiency in the context of biomedical research. Significantly, the application was developed using a containerized approach, which underscores its scalability and ease of deployment. This design choice ensures that any future enhancements or modifications can be seamlessly integrated, facilitating continuous improvement and adaptation to emerging research needs.

Ultimately, this thesis contributes to the advancement of biomedical research by providing a tool for the health monitoring of pigs, an important animal model in medical research. It showcases the advantages of adopting technological solutions to enhance research methodologies. In future many new features may be needed for this application. One of the biggest improvement could be a new algorithm, that not only tracks pigs movements, but also tracks whether the pig is breathing or how much time does it spend eating or drinking.

Bibliography

1. SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.
2. HACHUEL, David; ESTRIN, Deborah; MARTINEZ, Alfonso; STALLER, Kyle; VELEZ, Christopher. *Augmenting Gastrointestinal Health: A Deep Learning Approach to Human Stool Recognition and Characterization in Macroscopic Images*. 2019.
3. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. Available from arXiv: 1506.01497 [cs.CV].
4. GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. Available from arXiv: 1311.2524 [cs.CV].
5. GIRSHICK, Ross. *Fast R-CNN*. 2015. Available from arXiv: 1504.08083 [cs.CV].
6. CARION, Nicolas et al. *End-to-End Object Detection with Transformers*. 2020. Available from arXiv: 2005.12872 [cs.CV].
7. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep Residual Learning for Image Recognition*. 2015. Available from arXiv: 1512.03385 [cs.CV].
8. WU, Yuxin; KIRILLOV, Alexander; MASSA, Francisco; LO, Wan-Yen; GIRSHICK, Ross. *Detectron2* [<https://github.com/facebookresearch/detectron2>]. 2019.

9. PASZKE, Adam et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. Available also from: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
10. ISWANTO, Irene Anindaputri; LI, Bin. Visual object tracking based on mean-shift and particle-Kalman filter. *Procedia computer science*. 2017, vol. 116, pp. 587–595.
11. BOLME, David S; BEVERIDGE, J Ross; DRAPER, Bruce A; LUI, Yui Man. Visual object tracking using adaptive correlation filters. In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2010, pp. 2544–2550.
12. HENRIQUES, João F; CASEIRO, Rui; MARTINS, Pedro; BATISTA, Jorge. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence*. 2014, vol. 37, no. 3, pp. 583–596.
13. KALAL, Zdenek; MIKOLAJCZYK, Krystian; MATAS, Jiri. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*. 2011, vol. 34, no. 7, pp. 1409–1422.
14. MEDSKER, Larry; JAIN, Lakhmi C. *Recurrent neural networks: design and applications*. CRC press, 1999.
15. LI, Bo; YAN, Junjie; WU, Wei; ZHU, Zheng; HU, Xiaolin. High performance visual tracking with siamese region proposal network. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8971–8980.
16. CIAPARRONE, Gioele et al. Deep learning in video multi-object tracking: A survey. *Neurocomputing*. 2020, vol. 381, pp. 61–88. ISSN 0925-2312. Available from DOI: 10.1016/j.neucom.2019.11.023.

17. HAN, Mei; SETHI, Amit; HUA, Wei; GONG, Yihong. A detection-based multiple object tracking method. In: *2004 International Conference on Image Processing, 2004. ICIP'04*. IEEE, 2004, vol. 5, pp. 3065–3068.
18. O'DONOVAN, Peter. Optical flow: Techniques and applications. *International Journal of Computer Vision*. 2005, vol. 1, p. 26.
19. BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000.
20. WOJKE, Nicolai; BEWLEY, Alex; PAULUS, Dietrich. *Simple Online and Realtime Tracking with a Deep Association Metric*. 2017. Available from arXiv: 1703.07402 [cs.CV].
21. BEWLEY, Alex; GE, Zongyuan; OTT, Lionel; RAMOS, Fabio; UPCROFT, Ben. Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016. Available from DOI: 10.1109/icip.2016.7533003.
22. PARICO, Addie Ira; AHAMED, Tofael. Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT. *Sensors*. 2021, vol. 21, p. 4803. Available from DOI: 10.3390/s21144803.
23. SEKACHEV, Boris et al. *opencv/cvat: v1.1.0*. Zenodo, 2020. Version v1.1.0. Available from DOI: 10.5281/zenodo.4009388.
24. VEIT, Andreas; MATERA, Tomas; NEUMANN, Lukas; MATAS, Jiri; BELONGIE, Serge. Coco-text: Dataset and benchmark for text detection and recognition in natural images. *arXiv preprint arXiv:1601.07140*. 2016.
25. CHEN, Kai et al. MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155*. 2019.
26. ŠUSTR, Z et al. Metacentrum, the czech virtualized ngi. In: *EGEE Technical Forum*. 2009.

27. CONTRIBUTORS, MMTracking. *MMTracking: OpenMMLab video perception toolbox and benchmark* [<https://github.com/open-mmlab/mtracking>]. 2020.
28. BERGMANN, Philipp; MEINHARDT, Tim; LEAL-TAIXE, Laura. Tracking without bells and whistles. In: *Proceedings of the IEEE international conference on computer vision*. 2019, pp. 941–951.
29. CHODOROW, Kristina; DIROLF, Michael. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly, 2010. ISBN 978-1-449-38156-1.
30. TEAM, The pandas development. *pandas-dev/pandas: Pandas*. Zenodo, 2020. Latest. Available from DOI: 10.5281/zenodo.3509134.
31. INC., Plotly Technologies. *Collaborative data science*. Montreal, QC: Plotly Technologies Inc., 2015. Available also from: <https://plot.ly>.

List of Figures

2.1	Difference between classification and detection [2]	8
2.2	Region proposal network [3]	10
2.3	Graphical representation of Faster-RCNN [3]	10
2.4	Graphical representation of DETR [6]	12
2.5	Graphical representation of SiameseRPN neural network [15]	17
2.6	Graphical representation of DBT	18
2.7	Graphical representation of DFT	18
2.8	Architecture of DeepSORT [22]	21
3.1	Example of input frame and annotated frame	25
4.1	Example of Mean-Shift tracking output	36
4.2	Example of TLD tracking output	36
4.3	Example of KCF tracking output	36
4.4	Example of MOT outputs	41
5.1	Simple app structure diagram	44
5.2	Web application - homepage	45
5.3	Web application - video detail	46
5.4	Lights on video on left and lights off video on right	49
5.5	Web application - reporting	52
5.6	Reporting - walked distance	53
5.7	Reporting - walked trajectory	53

List of Tables

2.1	Comparison of object detection models	13
3.1	Dataset information	24
3.2	Information about classes	24
4.1	Training parameters	28
4.2	IoU statistic over training - MMDetection	29
4.3	Training parameters	30
4.4	IoU statistic over training - Detectron2	31
4.5	Performance details of each framework	32
4.6	Training speed details of each framework	32
4.7	Inference speed details of each framework	33
4.8	Training speed details of each framework	33
4.9	IoU statistic over training - Detectron2	34
4.10	Comparison of used methods	35
4.11	Comparison of used MOT methods	39
5.1	Example of CSV file structure exported from the app	51

