

Bjerken, Benjamin Letnes
Holter, Lars Blütecher
Huynh, Daniel Hao
Wangerud, Lillian Alice

Fish Detection in Underwater Video

Bachelor's thesis in Programming
Supervisor: Pedersen, Marius
May 2023

Bjerken, Benjamin Letnes
Holter, Lars Blütecher
Huynh, Daniel Hao
Wangerud, Lillian Alice

Fish Detection in Underwater Video

Bachelor's thesis in Programming
Supervisor: Pedersen, Marius
May 2023

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



Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lars Blütecher Holter,
Lillian Alice Wangerud

Fish Detection in Underwater Video

Bachelor's thesis in Programming
Supervisor: Marius Pedersen
May 2023

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



ABSTRACT

Studying wildlife entails the collection and analysis of a vast amount of data, which can be a time-consuming and laborious task. The Norwegian Institute of Nature Research (NINA) has conducted extensive research on aquatic animals using traditional methods, such as manual observation and data extraction from underwater footage. However, this process is both tedious and inefficient, prompting NINA to explore the possibility of using Artificial Intelligence (AI) to create an automatic detection and clipping tool for underwater videos. This tool would aid researchers by eliminating the need for manual sorting of irrelevant video segments, thereby streamlining the research process.

To this end, the present project undertakes an in-depth examination of various approaches to developing a fully-functional solution for NINA. The proposed solution leverages a deep learning-based object detection model known as You Only Look Once (YOLO) to detect aquatic life in underwater environments. The module is trained on a dataset of relevant species to ensure the accuracy and reliability of the results. The processed data is then stored in a SQLite database, and the user can select the location of the video clips via a User Interface (UI).

The UI provides the user with a range of options to adjust the settings of the model, clips, and report. The proposed solution offers a comprehensive tool for use in NINA's continued research on aquatic animals. The use of AI-based technology has the potential to significantly improve the efficiency and accuracy of data collection and analysis, paving the way for new insights into the behavior and ecology of aquatic species.

SAMMENDRAG

Studier av dyreliv krever innsamling og analyse av store mengder data, som kan være en tidkrevende og arbeidskrevende oppgave. Det norske instituttet for naturforskning (NINA) har gjennomført omfattende forskning på akvatiske dyr ved hjelp av tradisjonelle metoder, som manuell observasjon og datautvinning fra undervannsoptak. Denne prosessen er både kjedelig og ineffektiv, noe som har ført til at NINA har utforsket muligheten for å bruke kunstig intelligens (AI) til å lage et automatisk deteksjons- og klippeverktøy for undervannsvideoer. Dette verktøyet vil hjelpe forskerne ved å eliminere behovet for manuell sortering av irrelevante videosegmenter og dermed effektivisere forskningsprosessen.

For å oppnå dette gjennomfører prosjektet en grundig undersøkelse av ulike tilnærminger for å utvikle en fullt fungerende løsning for NINA. Den foreslåtte løsningen benytter en objekt-deteksjonsmodul basert på dyplæring kjent som "You Only Look Once" (YOLO) for å oppdage akvatisk liv i undervannsmiljøer. Modulen er trent på et datasett av relevante arter for å sikre nøyaktighet og pålitelighet av resultatene. De bearbeidede dataene lagres deretter i en SQLite-database, og brukeren kan velge plasseringen av videoopptakene via et brukergrensesnitt (UI).

UI gir brukeren en rekke alternativer for å justere innstillingene til modellen, opptakene og rapporten. Den foreslåtte løsningen tilbyr et omfattende verktøy for bruk i NINAs forskning på akvatiske dyr. Bruken av AI-basert teknologi har potensial til å betydelig forbedre effektiviteten og nøyaktigheten av datainnsamling og analyse, og åpne for nye innsikter i atferden og økologien til akvatiske arter.

PREFACE

This bachelor's thesis is authored by Benjamin Letnes Bjerken, Daniel Hao Huynh, Lars Blütecher Holter, and Lillian Alice Wangerud. The study was conducted at the Department of Computer Technology and Computer Science at the Norwegian University of Science and Technology (NTNU) in Gjøvik.

The research project was undertaken with the support and involvement of several individuals and institutions. Firstly, we express our gratitude to our supervisor, Marius Pedersen, for providing consistent guidance and feedback throughout the research process. We also extend our appreciation to our client, the Norwegian Institute for Nature Research (NINA), and specifically to Knut Marius Myrvold and Tobias Holter, for entrusting us with a compelling research task. Their regular bi-weekly meetings and contributions were instrumental in driving our project forward. Additionally, we would like to thank Francesco Frassinelli from NINA's IT department for his valuable assistance in performing integration tests and his contributions to the project. His expertise and support was greatly appreciated.

Furthermore, we wish to acknowledge the assistance provided by our peers, friends, and family, who assisted in reviewing and ensuring the quality of our work.

Lastly, we would like to acknowledge our own significant contributions and effective collaboration during the course of the research project.

CONTENTS

Abstract	i
Sammendrag	ii
Preface	iii
Contents	vii
List of Figures	viii
List of Tables	x
Code Listings	xi
Glossary	xii
Abbreviations	xxi
1 Introduction	1
1.1 Project Description	2
1.2 Group Background	3
1.2.1 Academic Background	3
1.2.2 Group Motivation	4
1.3 Delimitations	4
1.4 Organizing	4
1.4.1 Project Roles	5
1.4.2 Professional Areas of Responsibilities	6
1.5 Thesis Structure	6
2 Requirement Specifications	9
2.1 Project Goals	9
2.1.1 Result Goals	10
2.1.2 Impact Goals	10
2.1.3 Learning Objectives	11
2.2 Use Case	11
2.2.1 Functional Specification	12

3	Development Process	15
3.1	Development Model	15
3.1.1	Choice of Software Development Model	15
3.1.2	Our Software Development Model	16
3.2	Meetings	17
3.2.1	Client Meetings	18
3.2.2	Supervisor Meetings	18
3.2.3	Sprint Planning Meetings	18
3.2.4	Daily Scrum	19
4	Tools and Technologies	21
4.1	Tool Overview	21
4.1.1	Collaboration	22
4.1.2	Documentation	24
4.1.3	Programming Language	24
4.1.4	Prototyping	25
4.1.5	Poetry	25
4.1.6	Pre-commit	26
4.1.7	Annotation	27
4.2	Technologies	29
4.2.1	Quality Assurance	29
4.2.2	GUI	29
4.2.3	Local Database	30
4.2.4	File Generation	31
4.2.5	Video Processing	32
4.2.6	Artificial Intelligence	32
4.3	Technology Integration and Interaction	35
5	System Architecture	37
5.1	Disk	38
5.2	Front-end	38
5.3	Back-end	38
5.3.1	Data Manager	38
5.3.2	Report Manager	39
5.3.3	Video Processor	39
5.4	AI	39
6	Implementation	41
6.1	Development tools	41
6.1.1	Poetry	41
6.1.2	Pre-commit	42
6.2	Front-end	43
6.2.1	Initialization and Main Function	43
6.2.2	MainWindow and User Interface	43
6.2.3	Processing and Output	44
6.2.4	Settings	44
6.3	Back-end	45
6.3.1	Data Manager	46
6.3.2	Report Manager	47

6.3.3	Video Processor	49
6.3.4	The Video Processing Pipeline	51
6.3.5	Time Estimation in Video Processing	52
6.4	AI Model	53
6.4.1	Image Loader	54
6.4.2	BatchYolov8: A Custom Object Detection Model	55
7	Graphical User Interface	57
7.1	Prototyping	57
7.2	Ergonomics	58
7.3	GUI-Elements	59
7.3.1	File Manager	59
7.3.2	Tool Tips	60
7.3.3	Progress Bar and Feedback Window	61
7.4	GUI-Evolution	61
8	Object-detection	63
8.1	Dataset Creation and Utilization	63
8.1.1	Gathering the Dataset	64
8.1.2	Annotation Rules	66
8.1.3	Dataset Generation Script	67
8.1.4	Distribution of Annotations per Class Script	68
8.2	Model Training and Optimization	69
8.2.1	Training Setup	69
8.2.2	Data Splits and Class Definitions	70
8.2.3	Training Details and Rationale	70
8.2.4	Model Augmentations and Hyperparameters	71
8.2.5	Training Resource Considerations	73
8.2.6	Training and Evaluation Metrics	73
8.3	Performance Evaluation	77
8.3.1	Validation Metrics	77
8.3.2	Evaluation Videos	78
8.3.3	Fish Ranges of Videos	80
8.3.4	Average Recall over Confidence	82
8.4	Challenges and Limitations	84
8.4.1	Issues During Training	84
8.4.2	Performance Evaluation Limitations	85
9	Quality Assurance	87
9.1	Source Code Quality Assurance	87
9.2	Logging	88
9.3	Git Branching	88
9.4	Documentation	89
9.5	Testing	89
9.5.1	Continuous Integration: Build and Unit Testing	89
9.5.2	User Testing	90
9.5.3	Integration Testing	90
9.6	Group Work	91

10 Discussion	93
10.1 Preparatory Work	93
10.1.1 Project Plan	93
10.1.2 Goals	93
10.2 Cooperation	94
10.2.1 Meetings	94
10.2.2 Work Allocation	95
10.3 Flexibility of Development	96
10.3.1 Change in Task Description by Client	96
10.3.2 Report Structure Changes	96
10.3.3 CVAT	97
10.3.4 Windows Python	98
10.4 Process Critique	98
11 Conclusion	101
11.1 Project Achievements	101
11.1.1 Implementation and Interface	101
11.1.2 Performance	102
11.2 Future Work	103
11.3 Final Words	104
Bibliography	105
Appendices:	109
A Project Agreement	109
B Task Description	116
C Project Plan	118
D Prototype	150
E User Test Form	152
F Timekeeping	155
G Meeting Logs	178
H Annotation Guide	203
I Code	208
A - Github repository	247

LIST OF FIGURES

1.0.1 Example Pictures from Video.	1
1.1.1 Original Idea of Application	2
1.4.1 Group Organization with External Partners and Supervisor	5
2.1.1 Original draft of the application.	9
2.2.1 Use Case Diagram of Front-end GUI	11
2.2.2 Functional Specifications Layed out.	12
3.1.1 Our Kanban Board	17
4.1.1 Tools Used in the Project	21
4.1.2 Discord Threads	24
4.2.1 Example of the Annotation Overlay on an Output Video.	33
4.2.2 YOLOv8 Compared to Other YOLO Models	33
4.3.1 Specifications with Libraries	36
5.0.1 Full System Architecture	37
6.2.1 Front-end Layout	44
6.3.1 Back-end Flowchart	45
6.3.2 Local Database ER Diagram	46
6.3.3 CSV Report Structure	47
6.3.4 XLSX Report Structure of Summary	48
6.3.5 XLSX Report Structure of Detections	48
6.4.1 Image Loader System	55
7.1.1 Prototype of the GUI	58
7.3.1 Directory Window	60
7.3.2 Tool Tip Example.	60
7.3.3 Progress Bar and Feedback Window.	61
7.4.1 Final Version of the GUI.	62
8.1.1 Google Sheet for Tracking Annotated Videos	66
8.1.2 Table and Graph: Annotation Distribution per Class	68
8.2.1 Example Training Batch.	73
8.2.2 Performance Metrics.	74
8.2.3 Confusion Matrix.	75
8.2.4 Precision and Recall Curve.	76

8.3.1 Acceptable Conditions for Evaluation.	79
8.3.2 Invalid Conditions for Evaluation.	79
8.3.3 IoU for Valid Videos	81
8.3.4 IoU for Invalid Videos	82
8.3.5 Recall over Confidence	83
10.2.1 Work Distribution.	95

LIST OF TABLES

3.2.1 Visualization of Meetings	18
4.1.1 Self-hosted Server Specifications	28
4.2.1 Pros and Cons of PyQt and TKinter	30
4.2.2 YOLOv8 Model Sizes	34
8.1.1 Combined Annotation Distribution	64
8.1.2 Annotation Distribution from Previous Group	64
8.2.1 Hardware Specifications of Personal Computer	69
8.2.2 Hardware Specifications of the School-Allocated Computer	70
8.2.3 Final and Default Configurations for Hyperparameters	71
8.2.4 Model Augmentations Comparison	72

CODE LISTINGS

6.1	PyTorch CUDA Installation Configuration	42
6.2	Example Usage of the Ultralytics Annotation Implementation	49
6.3	Example Usage of our Annotator Implementation	50
6.4	Ultralytics Prediction Benchmark	53
6.5	Our Prediction Benchmark	53
	Code/video_processor.py	208
	Code/video_processor_first_iteration.py	214
	Code/fun_detected_frames_to_ranges.py	216
	Code/fun_add_buffer_to_ranges.py	217
	Code/fun_update_time_prediction.py	218
	Code/timer.py	219
	Code/vdi_test.py	219
	Code/settings.py	221
	Code/generate_dataset.py	224
	Code/find_annotation_distribution.py	233
	Code/find_overlapping_videos.py	234
	Code/logger.py	239
	Code/batchyolov8.py	240

GLOSSARY

Adobe XD A user experience design tool for creating interactive prototypes and wireframes. 25

Agile A project management methodology focused on flexibility and collaboration. 15–17

Back-end The back-end refers to the server side of a software application responsible for handling data storage, application logic, and processing. 13, 38, 39, 44, 45, 47, 95

background frame An image without any object of interest, used to help the model distinguish between objects and background noise.. 67

Balsamiq A software tool for creating quick wireframes and mockups. 25

Batch Size Batch size refers to the number of data samples or inputs processed together in a single iteration or forward pass through a machine learning model. It affects the trade-off between computational efficiency and model accuracy, with larger batch sizes generally offering faster training but potentially sacrificing some level of generalization performance.. 60, 62

Black Black is a Python code formatter that automatically reformats code to a consistent style and formatting using default rules. 29, 87, 89

ChatGPT An AI language model capable of generating natural language text. 24, 89

CNN Convolutional Neural Network (CNN) is a type of deep learning model commonly used for analyzing visual data. It is specifically designed to process grid-like structures, such as images, by employing convolutional layers that automatically learn and extract relevant features. CNNs have revolutionized computer vision tasks, including object detection and image classification.. 32

COCO Large-scale dataset for object detection, segmentation, and captioning in computer vision. 28

- commitlint** Commitlint enforces conventional commit messages in software development to ensure consistency and clarity, analysing messages to ensure compliance with predefined rules and conventions.. 90
- CRF** Constant Rate Factor: A quality setting used by the H.264 video codec that controls the trade-off between video quality and file size. Lower values result in higher quality but larger file sizes, while higher values produce smaller file sizes at the cost of video quality.. 50, 102
- CSV** Simple file format used to store tabular data in plain text. 31, 39, 47, 49, 62, 68, 101
- CVAT** Open-source tool for annotating images and videos for computer vision tasks. 27, 28, 66, 67, 97
- Daily scrum** Short daily meeting in Scrum / Scrumban software development for team synchronisation. 18, 19
- Dark mode** User interface setting that displays a dark colour scheme. 61
- Decoding** The process of converting a compressed video bitstream back into a usable format for playback. It involves interpreting the encoded data and reconstructing the original video frames. Decoding requires a compatible video codec that can interpret the compression format and produce the uncompressed video data for playback.. 51
- Directed Acyclic Graph (DAG)** A Directed Acyclic Graph (DAG) is a finite directed graph that contains no directed cycles. It consists of a set of vertices (or nodes) and edges, where each edge has a direction and connects two vertices. In the context of video processing, a DAG is used to represent the video processing pipeline, depicting the flow of operations and dependencies between them. Each operation or step in the pipeline is represented as a node, and the edges indicate the order of execution. The DAG ensures that the operations are performed in a specific sequence without circular dependencies, allowing efficient and parallel processing of video data.. 50
- Discord** Discord is a VoIP and instant messaging social platform. Users have the ability to communicate with voice calls, video calls, text messaging, media and files in private chats or as part of communities referred to as discord servers. A discord server is a collection of persistent chat rooms and voice channels which can be accessed via internalised invite links. Discord runs on Windows, macOS, Android, iOS, iPadOS, Linux, and in web browsers.. 22
- Docker** A containerization platform for applications.. 23
- Docstring** A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. It is used to explain the purpose of the code.. 89
- Drop-down** Graphical user interface element for selecting an item from a list. 59

- Encoding** The process of compressing raw video data to reduce file size while preserving visual quality. It involves converting the data into a codec-specific bitstream for storage, transmission, or decoding. Parameters like bitrate, resolution, and codec selection impact video quality, file size, and compatibility. Common video codecs include H.264, HEVC, and VP9.. 51
- FFmpeg** FFmpeg is a powerful multimedia framework used for processing audio and video files. It provides a command-line tool and a set of libraries that can handle various multimedia formats and perform tasks such as encoding, decoding, transcoding, and streaming. FFmpeg is widely used in the industry and has extensive capabilities for multimedia processing.. 32, 39, 50, 51
- Figma** Cloud-based design tool for creating and collaborating on user interfaces. 25, 57
- FK (foreign key)** A field in a database table that refers to the primary key of another table, used to establish a relationship between the two tables. 46
- FPDF** A Python library for creating PDF documents. 31, 39, 47
- Front-end** Front-end refers to the part of a website or application that users interact with, including the UI user interface, design, and UX user experience.. 13, 29, 30, 38, 39, 43, 90, 98
- Git** Git is a popular distributed version control system used to track changes in source code during software development, enabling collaboration and coordination between team members working on the same project. 22, 27, 43, 89–91, 96
- GitHub** A web-based platform for hosting and collaborating on code repositories. 22, 42, 43, 56, 89
- GitKraken** A premium Source Code Management Application for version control systems such as Git, for Windows, linux and Mac platforms. It provides an expansive and improved graphical interface for repositories, between users and Git, in which it often updates its application, this makes the workflow go more efficiently for those who haven't mastered Git in its terminal state, and the experts, who can be more productive focusing solely on the code. 22
- H.264** H.264 is a widely-used video codec standard, also known as AVC (Advanced Video Coding). It offers efficient compression of video data while maintaining good video quality. H.264 is commonly used for video streaming, video conferencing, and video storage applications.. 50
- high fidelity prototype** A high-fidelity prototype is a detailed and interactive prototype of a design that closely resembles the final product, created using advanced materials and tools, for thorough testing and evaluation before launch.. 25

isort isort is a Python utility for sorting and organizing import statements in Python code files. It ensures consistent and standardized import ordering, improving code readability and maintainability. isort automatically analyzes Python source code files, detects import statements, and sorts them according to defined rules and conventions. It can handle imports from both standard library modules and external third-party packages. By using isort, developers can enforce a consistent import style across their Python projects, reducing merge conflicts and making the codebase more manageable.. 29, 88

Jupyter An open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.. 23

Kanban A project management methodology that emphasises visualisation of work and limiting work in progress. 15–17

Letterboxing Letterboxing is a technique used in image or video processing to maintain the original aspect ratio of the content when displaying it on a different aspect ratio screen or frame. It involves adding black bars to the top and bottom (or sides) of the image or video to fill the unused space and maintain the correct proportions. Letterboxing is commonly used when adapting content from one aspect ratio to another, such as displaying widescreen content on a standard 4:3 screen without distorting or cropping the image.. 54, 56

Light mode A user interface display setting that uses a lighter colour scheme. 61

Lint A linter is a tool used to detect and correct errors in LaTeX code. It can be used to check for syntax errors, typos, and other issues that can affect the output of the code.. 89

Linting Linting is the process of analyzing source code for errors, bugs, and stylistic inconsistencies using a specialized tool called a linter. It helps improve code quality, maintainability, and readability by enforcing coding standards and identifying common programming mistakes. Popular linters include ESLint for JavaScript, Pylint for Python, and RuboCop for Ruby.. 41

low fidelity prototype A low-fidelity prototype is a basic and rough prototype of a design created using simple materials and tools to validate design concepts before investing significant resources in prototypes of higher fidelity or the final product. 25

Matplotlib A Python library for creating 2D plots and graphs.. 24

Minidom A Python module for working with XML documents. 31, 32, 39, 47

- MP4** MP4 (MPEG-4 Part 14) is a popular digital multimedia container format used for storing audio and video files. It supports efficient compression, making it suitable for streaming, distribution, and playback of multimedia content. MP4 files can contain video streams encoded with codecs like H.264 and audio streams encoded with codecs such as AAC. It also supports features like subtitles, metadata, and interactive menus.. 39
- Muxing** Muxing, short for multiplexing, is the process of combining multiple data streams into a single stream or file. In the context of video processing, muxing involves combining audio and video data, along with any other necessary streams, into a single container format, such as AVI, MP4, or MKV. The resulting muxed file contains all the synchronized data streams, allowing for simultaneous playback or storage. Muxing is commonly performed during video encoding or transcoding to create a final video file that can be easily played or shared.. 51
- mypy** Mypy is a static type checker that analyses the code for potential errors and inconsistencies, allowing early detection of bugs and improving code quality and maintainability. 29, 88, 89
- NextCloud** A cloud storage and collaboration platform. 23
- NMS** Non-Maximum Suppression (NMS) is a post-processing technique commonly used in object detection algorithms. It is used to remove duplicate or overlapping bounding box predictions generated by the detection model. NMS works by comparing the confidence scores of the bounding boxes and suppressing the boxes that have a high overlap with other boxes and lower confidence scores. This helps to eliminate redundant detections and keep only the most relevant and accurate bounding boxes. NMS is an important step in improving the precision and reducing duplicate detections in object detection systems.. 55, 56
- Outlook** Microsoft Outlook is a part of Office package that can be used as its own standalone application. It helps you access Microsoft Exchange Server email. Additionally, it provides contacts, calendaring, and task management functionality. This advanced email application is widely used for business purposes. 23
- Overleaf** An online LaTeX editor for collaborative writing and publishing. 24
- PASCAL VOC** A benchmark dataset for object recognition and detection in computer vision. 28
- PDF** Portable Document Format, a file format used for representing documents in a manner that is independent of the software, hardware, and operating system. It provides a consistent visual representation of the document, including text, fonts, images, and formatting, making it widely used for sharing and distributing documents across different platforms.. 39, 47
- PEP 8** A style guide for Python code that provides guidelines for code formatting and organisation. 29, 87

phash Short for perceptual hash, *phash* is a technique used for generating a compact representation of an image based on its visual features. It aims to capture the essential characteristics of an image in a fixed-size hash value. *phash* algorithms analyze the frequency and spatial information of the image to create a hash that remains relatively unchanged even if the image is subjected to minor alterations, such as scaling, rotation, or compression. By comparing the *phash* values of two images, it is possible to determine their similarity or dissimilarity. *phash* has applications in various areas, including image retrieval, duplicate image detection, and content-based image identification.. 65

PIL Python Imaging Library (PIL) is a library that adds image processing capabilities to the Python interpreter. PIL supports a wide variety of image file formats and provides powerful image processing and graphics capabilities, such as resizing, cropping, and drawing.. 49

Poetry A Python packaging and dependency management tool. 25, 26, 41, 42, 89

Prediction threshold Prediction threshold, in the context of machine learning, is a value that determines the boundary or cutoff for classifying a prediction as a positive or negative outcome. It is used in binary classification tasks where the model's output probability needs to be compared to the threshold to make a final prediction decision. Adjusting the prediction threshold can impact the trade-off between precision and recall in the classification results.. 60, 62

PyAV PyAV is a Python library that provides bindings for FFmpeg, allowing seamless integration of FFmpeg's multimedia processing capabilities into Python applications. PyAV enables developers to work with audio and video files, perform tasks like decoding, encoding, and transcoding, and access metadata and other information about multimedia content. By leveraging FFmpeg's functionality, PyAV provides a powerful and flexible solution for multimedia processing in Python.. 32, 39, 51

Pylint A Python static code analysis tool to identify potential errors and issues. 29, 87, 89

pyproject.toml A file used in Python projects to specify various project settings and configurations. It is written in TOML (Tom's Obvious, Minimal Language) format, which is a human-friendly configuration file format. The `pyproject.toml` file commonly includes information such as project metadata, build configurations, tool settings, and most importantly, project dependencies managed by tools like Poetry. It provides a centralized location for defining project-specific settings and simplifies the management of dependencies and build processes. The file is often located at the root directory of a Python project and is recognized by various tools and build systems in the Python ecosystem.. 41, 42

PyQt A Python binding for the Qt application framework to build desktop applications. 24, 29, 30, 43, 44, 59, 96

- R-CNN** Region-Based Convolutional Neural Network (R-CNN) is an object detection model that combines selective search with a convolutional neural network. R-CNN first generates a set of region proposals using selective search, which are then classified and refined using a CNN. R-CNN improved the accuracy of object detection by utilizing region-based processing and sharing convolutional features across different regions. However, R-CNN has a relatively slow inference speed due to its multi-stage pipeline.. 32
- Scrum** A project management framework that emphasises iterative development, adaptability, and team collaboration. 5, 6, 15–17
- Scrum master** A role in the Scrum framework responsible for facilitating and guiding the Scrum process and the team. 5
- Scrumban** A hybrid project management methodology that combines elements of Scrum and Kanban. 15–17, 96
- SourceTree** A free Source Code Management Application for version control systems such as Git, for Windows and Mac platforms. It provides a graphical interface for repositories, between users and Git, in which simplifies its use for beginners, who haven't mastered Git, and experts, who can be more productive focusing solely on the code.. 22
- Spin-box** A graphical user interface element for selecting a numerical value within a range. 59
- Sprint planning** A Sprint Planning Meeting is a meeting in the Scrum framework that sets the goals, defines the work, and plans the tasks for the next sprint, involving the entire Scrum team.. 17, 18, 95
- Sprint Retrospective** A meeting in the Scrum framework where the team reflects on the previous sprint and identifies opportunities for improvement. 18
- Sprint Review** A meeting in the Scrum framework where the team demonstrates the completed work from a sprint and gathers feedback. 18
- SQL** Structured Query Language, a standardized programming language used for managing and manipulating relational databases. It provides a set of commands and statements for querying data, modifying database structures, and controlling access and security.. 46
- SQLite** A lightweight, open-source, SQL-based relational database management system. 30, 31, 46
- Teams** Microsoft Teams is an application used for collaboration with Microsoft 365. The service enables instant messaging, audio and video calls, rich online meetings, mobile experiences, and extensive web conferencing capabilities. Additionally, Teams provides file and data collaboration and extensibility features and integrates with Microsoft 365 and other Microsoft and partner applications. 22

- TeamViewer** A remote access and support software for desktop sharing and on-line meetings. 23
- TKinter** A Python binding for the Tk GUI toolkit to build desktop applications. 29, 30
- Toggl** A cloud-based time tracking tool for individuals and teams that allows users to track the time spent on tasks and projects, generate reports, and analyse productivity. 22, 24
- VRAM** VRAM (Video Random Access Memory) is a dedicated type of memory used in graphics processing units (GPUs). It serves as a high-speed buffer for storing and accessing graphical data, including textures, shaders, and frame buffers. VRAM is optimized for parallel data processing, allowing for efficient rendering and display of graphics-intensive applications, such as video games and 3D modeling. By having its own memory separate from the system RAM, VRAM helps improve overall graphics performance and responsiveness.. 69, 70
- Waterfall** The Waterfall development model is a linear sequential software development approach that follows a strict set of phases in which each phase must be completed before moving on to the next.. 15, 16
- Widget** A graphical user interface element that displays information or allows user interaction. 25, 43, 44
- X11 forwarding** A feature of the X Window System (X11) that allows applications running on remote computers to be displayed on a local computer.. 96
- XCB** X Window System protocol client library. 96
- XLSX** XLSX is a file extension used for spreadsheets created with Microsoft Excel, which is a popular spreadsheet software used for data management and analysis.. 31, 39, 48, 62, 101
- xlsxwriter** XlsxWriter is a Python module used to create Excel XLSX files, which allows the creation of spreadsheets with formatting, charts and images using Python code.. 31, 39, 47
- XML** XML (Extensible Markup Language) is a markup language used to store and transport data, which is both human-readable and machine-readable, making it a common format for the exchange of information between different systems and applications.. 31, 32, 39, 47, 67
- YAML** YAML (YAML Ain't Markup Language) is a human-readable data serialization format. It stands for "YAML Ain't Markup Language" and is commonly used for configuration files and data exchange between languages with different data structures. YAML's simple and intuitive syntax makes it easy for humans to read and write, while its flexibility allows for complex data structures and hierarchical data representations. YAML files typically use the .yaml or .yml file extension.. 67, 89

YOLOv8 A real-time object detection system that uses deep neural networks to detect and locate objects in images and videos. It is the eight-version of the YOLO (You Only Look Once) system and features improved accuracy and speed compared to previous versions. . viii, 32–35, 39, 49, 51, 53–56, 63, 69, 70, 96, 103

ZIP ZIP is a file compression format commonly used for creating smaller-sized archive files. It employs lossless compression algorithms to reduce file size without losing data. ZIP archives are easily extractable and widely supported by operating systems and file compression tools.. 67

ABBREVIATIONS

- AI** Artificial Intelligence. 3, 4, 6, 11, 23, 27, 38, 39, 78, 96, 104
- CI** Continuous Integration. 42, 43, 89, 90
- CPU** Central Processing Unit. 55
- CV** Computer Vision. 11
- GPU** Graphical Processing Unit. 42, 53–56, 69, 102
- GUI** Graphical User Interface. 3, 7, 13, 22, 41, 43, 57–62, 96, 98, 101
- NINA** Norwegian Institute for Nature Research. 1–4, 6, 10, 11, 16, 18, 22, 23, 27–29, 33, 34, 47, 50, 52, 57, 65, 85, 88, 90, 94, 96, 97, 101–104
- NTNU** Norwegian University of Science and Technology. 3
- OS** Operating System. 23, 30, 59, 61, 96
- UI** User Interface. xiv, 6, 11, 27, 38, 44
- UX** User Experience. xiv, 11, 60
- VDI** Virtual Desktop Infrastructure. 23, 90, 91, 96, 102
- VoIP** Voice over Internet Protocol. xiii
- YOLO** You Only Look Once. 33

INTRODUCTION

This project was commissioned by the Norwegian Institute for Nature Research (NINA) and was undertaken with the guidance of Tobias Holter and Knut Marius Myrvold, our contact persons at NINA. NINA is an autonomous foundation dedicated to nature research and investigating the connection between human society, natural resources, and biodiversity [1].

Every year, NINA deploys underwater cameras in Mjøsa, Norway’s largest lake, to capture video footage of underwater wildlife. NINA informs that these cameras operate continuously for approximately 17 hours per day, starting from May 15th until August 1st, and then for 12 hours per day from August 1st to October 15th. Consequently, a substantial amount of video content is generated, totaling approximately up to 2175 hours or around 90.5 days of footage annually. The manual inspection of this extensive video collection for wildlife analysis is an arduous and resource-intensive task. Recognizing the need to overcome this challenge, NINA sought to find a solution that could significantly reduce the time required for manual video inspection. The creation of such a solution enables NINA to effectively monitor and study fish populations, aligning with the objectives of United Nations Sustainable Development Goal 14 [2].



(a) In the day.



(b) In the evening.

Figure 1.0.1: This figure presents two examples of the videos acquired by NINA. Image (a) depicts a daytime scene with ample lighting, while image (b) captures an evening setting after sunset. The camera placement encompasses various locations, leading to a diverse range of environmental conditions and the quality of the videos can significantly vary. Certain environments may exhibit challenges such as algae accumulation obscuring the camera lens, as depicted in image (a).

In 2022, NINA issued a bachelor’s project that entailed the development of a program that would allow the elimination of parts of videos where fish are not present, leaving only video footage where fish appear. A previous bachelor’s group had designed a system that employed machine learning and computer vision to detect fish species in a video, but it did not include the functionality to cut the video to include only pertinent data. Therefore, considerable manual work remained for the researchers at NINA.

The objective of this project is to develop a software that could be used to automate the process of processing recorded videos. They wanted an easy-to-use program linked to a object detection model which could scan through the videos for fish and remove unusable dead time. NINA has also requested that the system includes a wide range of settings to alter the output received and incorporate other modifications. Furthermore, we strive to enhance the accuracy of the fish detection and produce a relevant report that NINA can actively use in their research. The complete task description is presented in Appendix B.

1.1 Project Description

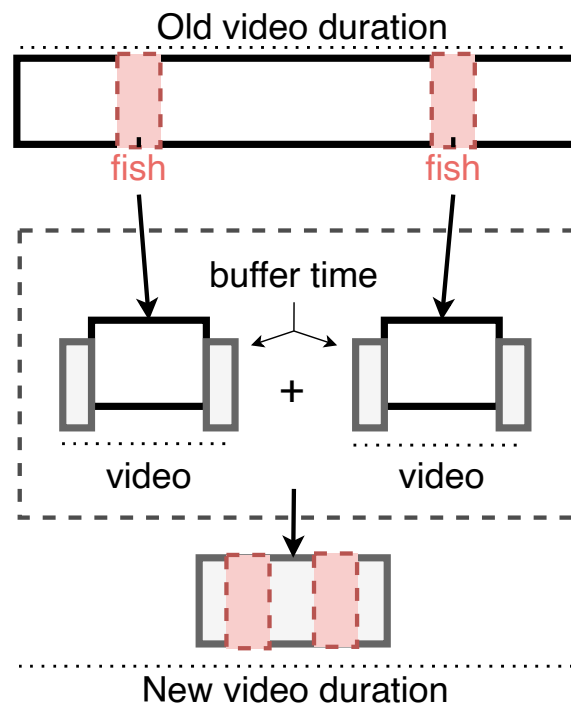


Figure 1.1.1: Original ideation sketch of application, this was drawn in the first meeting with our supervisor

The project consisted of developing a software that could be used to automate the process of processing recorded video. The software would specifically be used to scan through underwater video of fish to remove unusable dead time. In order to meet the end-users’ needs, the application was developed to be straightforward. This is such that researchers with little technological experience are able to utilize the program.

As per the project plan (see Appendix C) and description, these are some of the features that were planned to be part of the application:

- Gather data from an external hard disk
- Process large data sets
- Recognize and determine when fish or other underwater life is seen in a pre-recorded video.
 - Through the use of AI or other movement detection methods
- Process footage by cutting out and removing the clips where wildlife is not within the frame.
 - Save the processed data in a configurable format
- Allow configuration for the processing of the data input(video clips), this refers to; keep or delete the original data, add a customizable amount of dead time before and after the created clip, merge the clips, etc.
 - Everything that is not configurable is likely to be set by NINA prior, such as a filename standard, video length standard, etc.
- Have an intuitive and well formed GUI that allows for a good user experience with the program.
 - This should allow the user to easily choose the input folder, the output destination and customization to the clips.

1.2 Group Background

This section intends to provide an overview of our academic background and the rationale behind our selection of this bachelor's project.

1.2.1 Academic Background

The members of this bachelor project are all enrolled in the Bachelor in Programming program at the NTNU in Gjøvik. Collectively, we possess a similar academic background, with a few variations in elective courses. Specifically, three of us have completed the AI course, while the remaining member has taken courses in Reverse Engineering and Malware Analysis.

The AI course has equipped us with foundational knowledge on various types of models and research methods, which are highly relevant to the present project. Additionally, courses such as Advanced Programming, Software Development, and Cloud Technologies have provided us with a strong foundation in the relevant areas to undertake this bachelor's thesis.

1.2.2 Group Motivation

Upon review of all available bachelor theses proposals, the project description outlined in Appendix B stood out as particularly intriguing and promising. The project presents itself as both challenging and expansive, while still maintaining an element of excitement and novelty that drew our attention. Moreover, the project's multi-faceted approach aligns well with our academic pursuits, as it encompasses various fields of study such as AI, design principles, and software development throughout the program's entire life cycle.

While we possess some degree of familiarity with the technologies required for this project, this thesis offers a unique opportunity to acquire more comprehensive knowledge and practical experience. This endeavor enables us to employ our theoretical knowledge in a practical setting and push ourselves to acquire new skills and expertise.

In addition, our interactions with NINA, conveyed a strong sense of commitment to ensuring the success of the project. This was demonstrated through their prompt email responses and their enthusiasm for the project's potential. As a result, we are confident in our ability to develop a high-quality thesis with NINA's support.

1.3 Delimitations

In order to ensure that the present bachelor's thesis is completed within a feasible timeframe, certain delimitations have been established. Within the scope of this project, the following elements have been excluded:

Real-time processing of videos will not be implemented in the final product. This decision has been made due to limitations in the client's on-field equipment, which may not be adequate to perform image recognition and data processing, and has restricted network access, thereby preventing the transmission of data across the network for processing.

Improvements to onsite hardware will not be considered as part of this project, as it falls outside the scope of our relevant research area.

The counting, classification and tracking of fish will not be prioritized in the final application. This is due to these features not being part of the planned core software, and also because NINA already possesses software that can classify fish in recordings, which we may integrate with our project.

1.4 Organizing

Organizing a group can be a challenging task, especially when it involves different individuals with diverse skills, experiences, and perspectives. However, effective organization is essential for achieving the objectives of the group and ensuring successful outcomes. This section aims to display how we organize our group for the bachelor thesis project. The section covers the key aspects of organizing, including project roles, professional areas of responsibilities, conflict management, and decision-making.

1.4.1 Project Roles

At the inception of the project, a comprehensive delineation of roles was formulated to ensure that each group member had a clear sense of ownership and responsibility for specific aspects of the project. This strategy was implemented to optimize work efficiency and ensure that all critical elements of the project were adequately followed up. The roles assigned to each member are presented below for clarity and transparency.

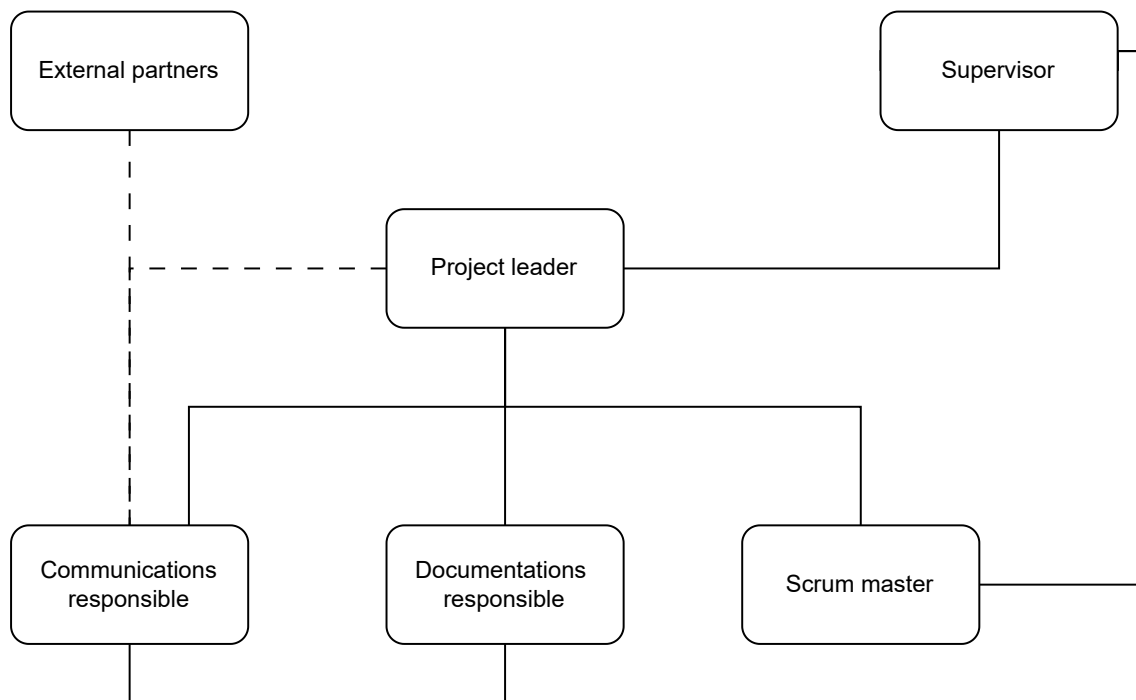


Figure 1.4.1: Organization of the group with external partners and supervisor

- **Project leader** - Lars Blütecher Holter
 - Ensures that every member contributes to the project and that the project follows the project plan
 - Responsible for taking decisions during group disputes
- **Communications responsible** - Benjamin Letnes Bjerken
 - Responsible for keeping communication out of and in to the group
 - Set up meetings between group and client
- **Documentations responsible** - Lillian Alice Wangerud
 - Writes reports for meetings
 - Responsible for ensuring that the project is adequately documented
- **Scrum master** - Daniel Hao Huynh
 - Responsible for summoning all members for Scrum meetings

- Leads Scrum meetings and ensures that every thing on the agenda is brought up during the meeting
- **External partners** - Tobias Holter, Knut Marius Myrvold, and Francesco Frassinelli
 - Provide information from NINA needed for the thesis
 - Provide feedback
- **Supervisor** - Marius Pedersen
 - Provide guidance and suggestions

In addition to the assigned project roles, each member of the group assumes the additional responsibility of serving as a developer.

1.4.2 Professional Areas of Responsibilities

Throughout the course of the bachelor’s thesis project, the group members opted to divide the various technology areas while maintaining a willingness to provide assistance where necessary. This strategy aimed to enhance each member’s immersion into specific areas of the project by dedicating sufficient time and resources to each area.

The decision to divide the different areas was based on each member’s personal interests and unique skill sets. Each member selected the area that most piqued their interest and where they could best leverage their expertise to contribute meaningfully to the project. This approach facilitated the development of a comprehensive understanding of the various technology areas, fostering a sense of ownership and accountability among the group members. By dedicating ample time and resources to specific areas, the group members could work more efficiently, leveraging their strengths to deliver optimal project outcomes.

The areas of responsibilities assigned to each member are presented below.

- **User Interface** - Lars Blütecher Holter and Lillian Alice Wangerud
- **Report output from program** - Lillian Alice Wangerud
- **Final report structure** - Lars Blütecher Holter
- **AI** - Benjamin Letnes Bjerken and Daniel Hao Huynh

1.5 Thesis Structure

The thesis report is intended to be read sequentially, with each chapter building upon the preceding one to provide a comprehensive overview of the project. However, for those seeking to isolate specific parts of the report, this section serves as a brief summary of each chapter’s content.

The report begins with a list of abbreviations and a glossary of key terms and concepts, providing readers with a comprehensive understanding of the terminology used throughout the thesis. This section helps ensure that readers are

equipped with the requisite background knowledge to understand the more complex concepts and ideas presented in subsequent chapters.

1. **Introduction**

This chapter introduces the thesis project and presents the group's organization and backgrounds, providing context for the research question.

2. **Requirement Specifications**

Any requirements related to the project are specified within this chapter. This includes the project goals, such as result goal, impact goal and learning objectives, but also functional requirements and use cases.

3. **Development Process**

The development process of the project is introduced and discussed within this chapter. It provides context to the chosen development model and explains choices made about how we performed meetings throughout the process.

4. **Tools and Technologies**

This chapter introduces and presents context to choices made regarding libraries, packages, models and other technologies that were utilized in order to develop the product.

5. **System Architecture**

This chapter provides an overview of the entire system and its architecture and explains the components of the system.

6. **Implementation**

The implementation of the technologies used and the development will be introduced and explained within this chapter. It provides a more detailed look at how components are built and interacts with each other.

7. **Graphical User Interface**

This chapter explores the thought process behind designing the GUI, focusing on ergonomics and the selection of GUI elements from prototyping to the final version.

8. **Object-detection**

The use of object detection is an integral part of the application and the usage of it will be discussed within this chapter. This chapter thus covers the utilization, performance and challenges of the object detection algorithm and creation of datasets to train, test and validate said algorithm.

9. **Quality Assurance**

The measures performed to ensure the quality of the application are detailed within this chapter. This includes explanation of how we utilized the quality assurance tools, the importance of documentation and the testing performed.

10. **Discussion**

This chapter gives a thorough discussion on choices that was made throughout the development process and of the following results. It will also provide

insight on the success of the planned development process and changes that was made or should have been made during the process.

11. **Conclusion**

This chapter concludes our bachelor's thesis by summarizing our goals, outlining their achievement, discussing potential future work, and providing our final thoughts on the project.

REQUIREMENT SPECIFICATIONS

To ensure our product aligns with the client’s requirements, it is imperative to establish clear goals and objectives. Additionally, anticipating and addressing potential challenges and uncertainties allows us to develop effective contingency plans. Several of these aspects were discussed and defined within the project plan, as outlined in Appendix C.

2.1 Project Goals

This section presents an overview of the objectives pursued throughout the course of the project. The objectives are classified into three categories: result goals, impact goals, and learning objectives. These classifications serve as a framework to outline the specific aims and intentions that guided our project endeavors.

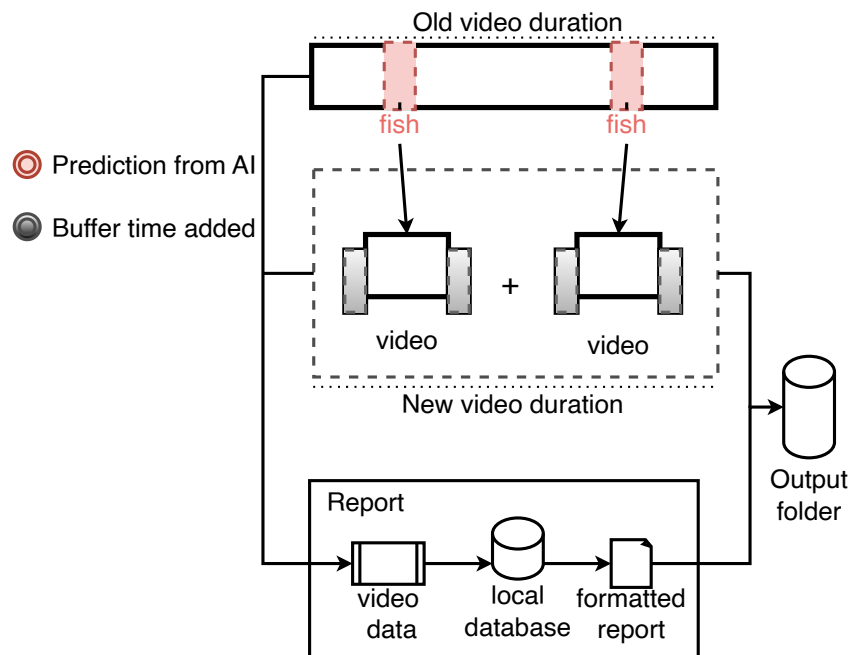


Figure 2.1.1: Based on the original draft of the Application, as seen in the Project plan.

2.1.1 Result Goals

Following further analysis at the onset of development and after insightful consultations with NINA, we revised our project goals from the initial plan presented in Appendix C to better cater to the specific requirements of fish detection.

In our project, our primary objective is now to accurately identify the ranges within the videos where fish are present, aiming to detect at least 95% of the ranges where fish are actually present. Unlike our initial goal, we are no longer focused on a balanced measure of precision and recall, as quantified by the F_1 -score. Instead, we are now more concerned with achieving a high recall rate, primarily focusing on detecting the presence of fish within specific ranges in the videos, regardless of whether every single frame with fish is identified or not.

Recall has become a more critical measure as it evaluates the model's capability to identify all ranges with fish present in the video. This change aligns better with NINA's requirements as it emphasizes the importance of minimizing false negatives - missing a fish is now considered a more significant issue than falsely detecting a fish. Consequently, we are more interested in ensuring that the model captures the existence of fish within a certain range, even if it mistakenly predicts fish in a few frames where there are none.

We complement the recall metric with the Intersection over Union (IoU) metric, which measures how accurately the model's predicted range of frames aligns with the actual range where fish are present.

To account for potential variations in frame quality and to allow for some level of inaccuracy in detection, we will incorporate a buffer of a few seconds before and after the detected fish instances in the output video. This approach ensures a more robust detection process, reduces the risk of missing any fish due to specific frames not being detected accurately, and aligns well with our focus on identifying ranges with fish, rather than individual frames.

Furthermore, we continue to aim for a reduction in the amount of human effort required for video processing tasks. We maintain our goal to decrease the processing time to only 25 minutes for a 30-minute video, allowing human operators to focus on critical tasks and enhance their overall productivity.

2.1.2 Impact Goals

Our impact goals for the project include improving the efficiency of NINA's operations by reducing the time it takes to process video footage. By achieving our result goals of accurate fish detection and faster video processing, we can help NINA to quickly analyze and make use of the video data without building up a backlog of unprocessed video. This will help NINA to more effectively monitor and study fish populations.

Another impact goal is to reduce the amount of harddrives required to store video data. By processing and removing unnecessary footage, we can help NINA to save storage space and reduce the cost of video data storage.

We also aim to focus on future development so that it is easy for others to work on it, maintain it, and add new features. This will make it easier for NINA to continue to improve and update the software as new technologies and methods become available.

The research conducted by NINA, in conjunction with our software, contributes to advancing the United Nations' Sustainable Development Goals, with a particular focus on Goal 14, "Life Below Water" [2]. The primary objective of this collaboration is to conserve and sustain marine wildlife, fostering sustainable development and facilitating a deeper understanding of marine ecosystems. By leveraging our software solution, NINA aims to enhance their research efforts and contribute to the overarching goal of achieving a sustainable and thriving marine environment.

2.1.3 Learning Objectives

Even though the bachelor thesis is about showing what we have learned thus far in our study, it is also about learning new subjects needed to complete our thesis. These learning objectives are far between each other but all are relevant to becoming a software engineer. The learning objectives we strive to become proficient in throughout the thesis consists of but are not limited to; the complete development life cycle of a software on a commercial project, implementing Artificial Intelligence (AI) and/ or Computer Vision (CV), User Experience (UX) and User Interface (UI), processing of video data and how to connect all aspects we create through coding in different languages.

2.2 Use Case

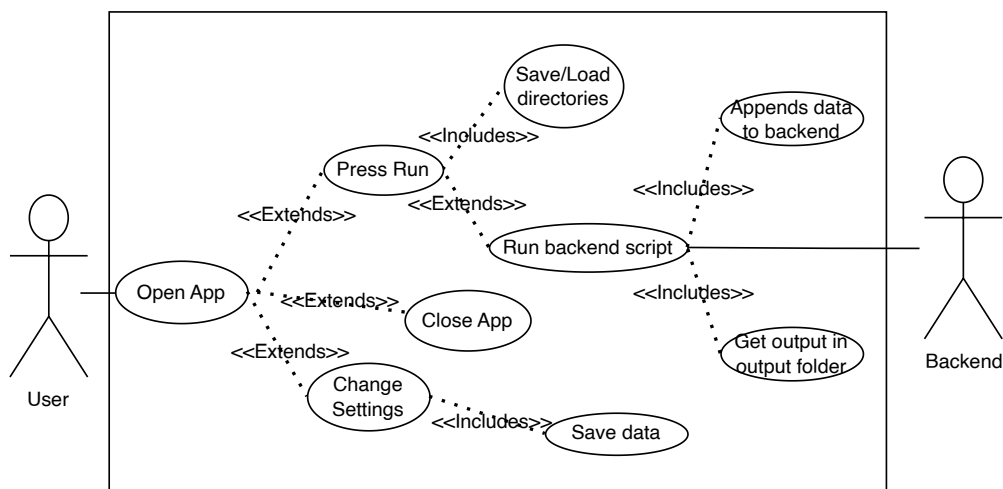


Figure 2.2.1: Use Case Diagram of Front-end GUI

To ensure that we capture the fundamental functionalities of our system and illustrate the interactions between various actors, we have chosen to utilize a use case diagram that includes both high-level and low-level use cases. This approach enables us to keep the system requirements at the forefront of our development process, making it easier to track progress and identify any necessary modifications.

The use case diagram illustrated in Figure 2.2.1 provides a broader perspective of the system, facilitating important decisions during the requirement phase. By visualising the bigger picture, we can gain a comprehensive understanding of the

system's functionalities and identify any potential issues or conflicts. Overall, the use case diagram serves as a valuable tool in enhancing the efficiency and effectiveness of our development process.

2.2.1 Functional Specification

The functional specification figure underneath, see Figure 2.2.2, includes a description of the intended capabilities of our product. The figure is based on the requirements detailed within this chapter, and includes how the user interfaces with the application and how that affects different aspects of the planned functionalities.

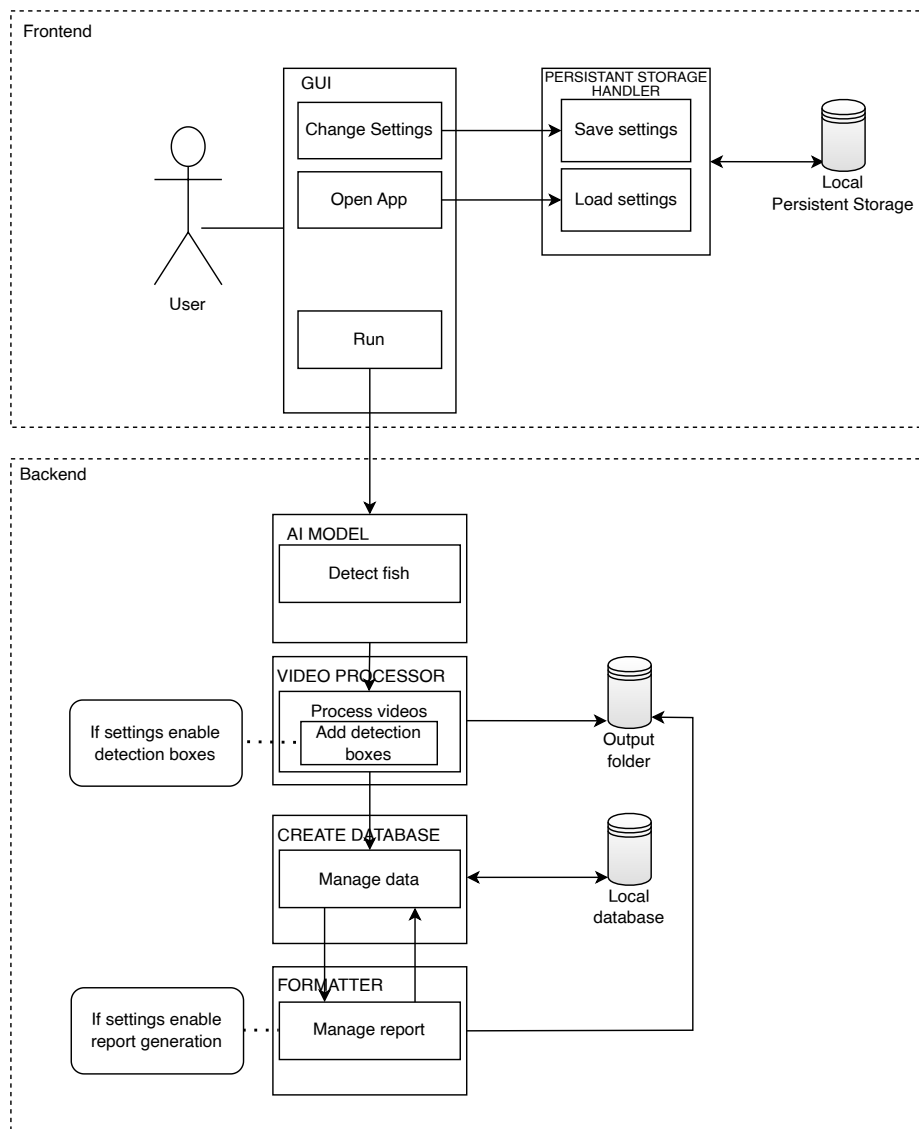


Figure 2.2.2: Functional Specifications Layed out.

Figure 2.2.2, shows how different aspects of the system interact. Thus the list underneath is an explanation of each actor shown within the figure. In the current model shown, we have not included which technologies are going to be utilized, however that will be expanded upon in Chapter 4 Figure 4.3.1

- **User:** Works in the field and will use the application to cut down the dead time in the videos.
- **Front-end:** The GUI of the application; this is where the interaction with the application happens.
- **Back-end:** The backbone of the application, this is where all the processing and outputting of the files happens.

DEVELOPMENT PROCESS

This chapter outlines the methodology and tools employed in the execution of the project from initiation to completion. The software development model chosen for the project is discussed in detail, alongside the meeting protocols established to facilitate the smooth running of the project. Additionally, the tools utilized in the project's execution are elaborated upon. This section serves as a sequel to the project plan and provides a comprehensive summary of the development process.

3.1 Development Model

The software development process is reliant on the choice of an appropriate development model. The industry offers various models, such as the Waterfall, Kanban, Scrum, Scrumban and Agile model. Each model presents its own advantages and disadvantages in the development process, emphasizing the importance of carefully evaluating these factors to determine the optimal model for the project's requirements. In this section, we discuss our thoughts in choosing a model as well as how we used our chosen model.

3.1.1 Choice of Software Development Model

The Waterfall model is a linear, sequential approach to software development that follows a predetermined set of steps, from conception to deployment. The model's benefit lies in its simplicity, enabling easy understanding of the project's requirements and process flow. This clarity of process makes it easier to plan and schedule the project, and it is also easier to manage. However, its drawback lies in its rigidity, which makes it challenging to incorporate changes in the project's lifecycle. In essence, the model follows a "one size fits all" approach, which may not suit all projects. Changes to requirements or scope may require starting the project from scratch, which can be costly and time-consuming.

The Kanban model emphasizes continuous delivery and focuses on creating a highly efficient system. Its strength lies in its focus on workflow management, allowing teams to visualize the entire development process, identify bottlenecks, and address them accordingly. This approach enables faster product releases and reduces waste, enhancing productivity and quality. However, the Kanban model lacks the necessary structure to handle large-scale projects, presenting a notable

disadvantage. Without clear guidelines, there may be confusion regarding what tasks should take priority, and how to manage dependencies between tasks.

The Scrum model is a flexible and adaptable approach that prioritizes collaboration and feedback. It enables teams to quickly adapt to changes in the project's requirements and deliver value incrementally. Its significant advantage lies in its adaptability, with its flexibility enabling it to accommodate evolving requirements in the project. However, its lack of clear requirements documentation can cause confusion during development. Additionally, the Scrum model requires a highly skilled team to be successful, which can be challenging for some projects.

Scrumban is a hybrid approach that combines elements of both Scrum and Kanban. Its main advantage lies in its flexibility, enabling teams to customize their development process to suit their specific project's needs. Scrumban allows teams to manage their workflow with a Kanban board while maintaining the structure of Scrum's time-boxed iterations. This approach allows teams to prioritize tasks effectively and make changes to the project's requirements when necessary. However, Scrumban's flexibility can also be a disadvantage, as it may lack clear guidelines, making it challenging to manage large-scale projects effectively. Additionally, implementing Scrumban requires teams to have a deep understanding of both Scrum and Kanban, which may require additional training and resources. Overall, Scrumban can be an effective development model for teams that require the flexibility of Kanban but also need the structure of Scrum.

The Agile model is an iterative approach to software development that focuses on delivering high-quality products while enhancing team collaboration. It is highly adaptable to changes in project requirements, ensuring that the product delivered meets the customer's needs. Its primary benefit is its ability to respond quickly to changing requirements and deliver a quality product within a shorter timeframe. However, implementing the Agile model requires a significant investment in resources and expertise, making it challenging to adopt for small projects. Additionally, its emphasis on team collaboration may not work for all organizations, particularly those with a hierarchical structure.

In conclusion, the choice of a software development model plays a critical role in the success of a project. Each model has its own set of advantages and disadvantages, and careful consideration of these factors is necessary when selecting the optimal model for a project. It is also essential to recognize that a model that works for one project may not work for another, and adaptability is crucial when making changes to the project's requirements or scope. Ultimately, the success of a project is dependent on the effective implementation of the chosen software development model and the team's ability to work collaboratively and adapt to changes throughout the development process.

3.1.2 Our Software Development Model

After analyzing the project description provided by NINA, it became apparent that the lack of flexibility in the Waterfall was unsuitable for the project. The description was open-ended and allowed for various approaches to the project, making the Waterfall too rigid to accommodate such a dynamic process. While Kanban can provide a good structure for smaller projects, we believed that the scope of our project was too large, encompassing various subject areas, which made

it unsuitable for the model. A pure Scrum model for a project of this size was also not ideal since it was imperative to be clear in the documentation and reduce possible confusion throughout the development process. Although the Agile model seems appropriate, we decided not to use it since it would take time to master and implement it correctly. Instead, we opted for a model that combined two models we were more familiar with from previous projects.

The Scrumban model was chosen for our software development project after careful consideration of the advantages and disadvantages of various development models. The flexibility of Scrumban, combining the best of both Scrum and Kanban, was the main motivation for our decision. The best of Scrumban for our group include the following. A visual board to track the progress of the project and identify tasks to be worked on next was critical for our team's efficiency. Additionally, we found that having regular Scrum meetings to keep everyone updated and plan for each sprint was essential for collaboration and achieving our project goals. By using the Scrumban model, we were able to adapt to changes in project requirements and deliver a quality product within a shorter timeframe.

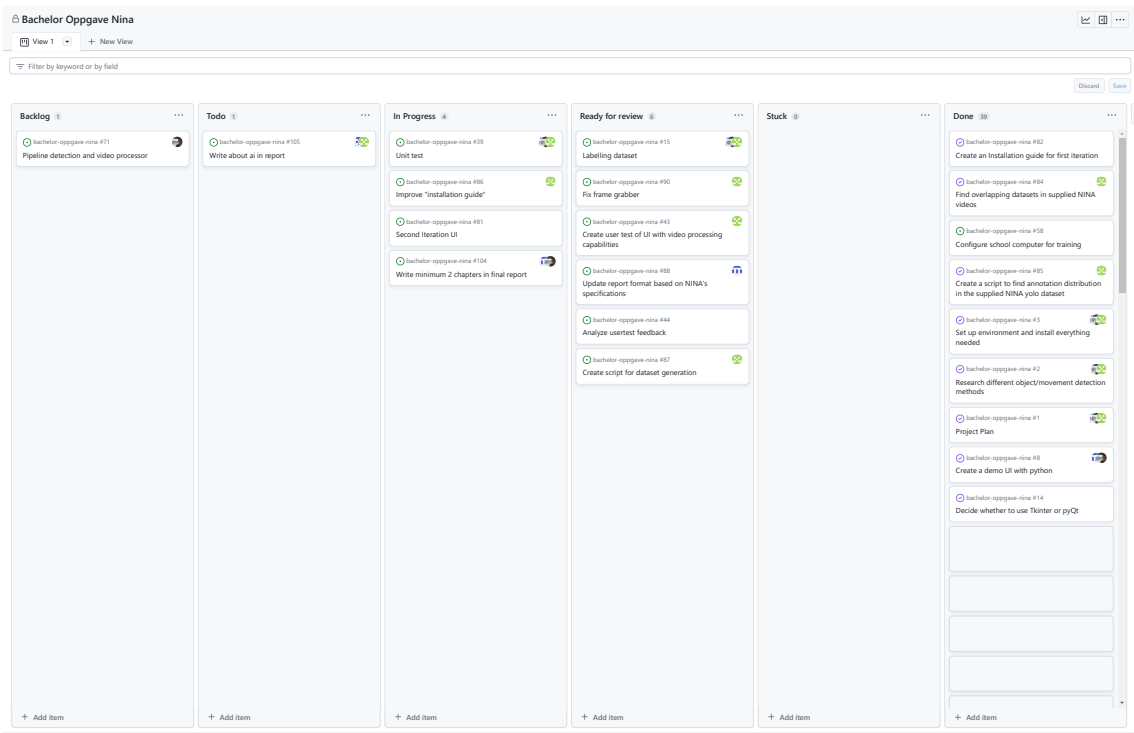


Figure 3.1.1: Our Kanban board towards the end for demonstration

3.2 Meetings

With the use of Scrumban we had a clear view of how we wanted to organize our meetings. We ended up having four different kind of meetings throughout the development of the project. These meetings were a weekly meeting with our supervisor Marius on Tuesdays at 14:00, which usually lasted an hour, a Sprint planning meeting right after our supervisor meeting was done with only the group members, a meeting with our client every other week on Mondays at 14:00 and

lastly we had a Daily scrum meeting at 12:00 on weekdays. The decision for this scheduling is discussed in following sections.

Table 3.2.1: This is a visualization of the meetings planned over each two week period throughout the development process.

Week	Monday	Tuesday	Wednesday	Thursday	Friday
1	Daily scrum Client meeting	Supervisor meeting Sprint planning	Daily scrum	Daily scrum	Daily scrum
2	Daily scrum	Supervisor meeting Sprint Retrospective Sprint Review	Daily scrum	Daily scrum	Daily scrum

3.2.1 Client Meetings

The client meetings were conducted every other week as previously noted. This was because we mostly used these meetings to showcase our progress to NINA and gather feedback. We also occasionally used some time to get some questions we were pondering answered. Besides these meetings, we used mail actively to get smaller questions answered by NINA. These meetings were scheduled on Mondays. This decision was made based on the understanding that any concerns, issues or new information gained during the meeting could be reflected upon by team members before the subsequent supervisor and Sprint planning meeting. By doing so, the team was able to engage in timely discussions and receive input from their supervisor regarding how to resolve issues or improve processes moving forward. Additionally, the meetings allowed for a more comprehensive and integrated approach to Sprint planning, ensuring that any new insights or developments were taken into account in the planning process. Ultimately, this approach supported the team's efforts to optimize productivity, enhance communication, and deliver high-quality software products.

3.2.2 Supervisor Meetings

By having weekly meetings with our supervisor, it allowed us to ask key questions in a orderly and efficient way at a regular basis. It also made it possible to bring up problems and questions no matter how large or small they were. These meetings were physical meetings at campus.

3.2.3 Sprint Planning Meetings

By conducting the Sprint planning meetings immediately after the supervisor meetings, we had just gained important informations which we used to plan the week ahead. The information and viewpoints received from our supervisor could be everything from what needed more refinement, to which specific parts needed to be developed next. This was done to optimize our advances and make sure we build the foundations before add-on features. The purpose was to ensure that we never had to go back and rework our software.

3.2.4 Daily Scrum

The last kind of meeting we had, the Daily scrum meeting, happened at 12:00 every weekday. These meetings were intended to make sure everyone made a habit to work throughout the day such that if we worked alone, we could still be sure that the other group members were available to help each other if ever needed. Additionally we used these meetings to showcase and talk about what we had worked on the day before and plan the current day. These meetings usually lasted about 30 minutes. It was the group leaders role to adjour the meeting. If any group members had any remaining questions, wanted to discuss or needed help, this was then done after the meeting was over. These meetings was an excellent way to keep every group member on the same page through the entire development cycle.

TOOLS AND TECHNOLOGIES

In this chapter, we will talk about what kind of tools and technologies are used in our project to achieve our goals.

4.1 Tool Overview

In this bachelor's project, we have carefully selected and planned the tools and environments that we intend to use in the development, documentation, quality assurance, core libraries and collaboration aspects of the project. An overview of the tools and environments selected is presented in Figure 4.1.1. By selecting and using these tools and environments effectively, our goal is to optimise our project development process, ensure high-quality results, and facilitate effective collaboration among team members.

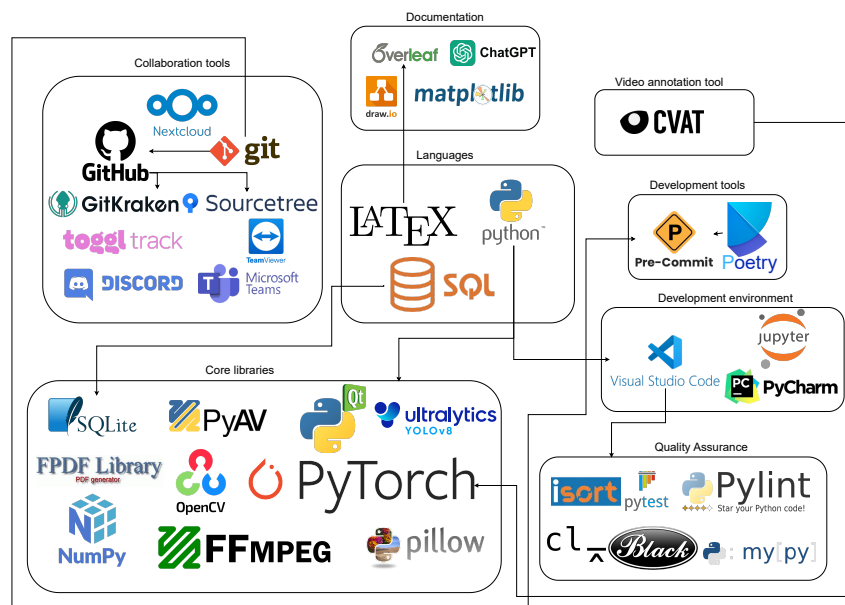


Figure 4.1.1: Tools used in the project and how they are linked.

4.1.1 Collaboration

- Git[3] - Git is a DevOps tool used for version control and source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. And it is also an industry standard. We use Git because of its relevancy throughout our bachelor degree, where it is the only tool of its kind we have been taught to use, as well as how its been internalized for everyone in our group.
- GitHub[4] - GitHub is an Internet hosting service for software development and version control with the aforementioned Git. It provides the distributed version control of Git[**Git**], plus access control, bug tracking, software feature requests, task management, continuous integration, and wikis for every project. We use it because we are very familiar with its use and it fits the scope of our open-source project, especially with the features mentioned earlier.
- SourceTree[5] - A free Source Code Management Application for version control systems such as Git. Some of us use this tool because its been introduced to us through previous courses.
- GitKraken[6] - A Source Code Management Application for version control systems such as Git. Some of us use this tool because of its more intuitive GUI and user support through regular updates.
- Discord[7] - Discord is a free communication tool used by people to communicate in chat channels in the form of text, images, video and audio. Discord is our main tool for communicating with group members, and is used to share resources, documents, and just collaborating in general. The reason for this choice of communication tool is because of our constant use of Discord to communicate during the bachelor degree. All of our group members use Discord regularly and have become familiarised and adept in its use, and have been in constant contact with each other prior to the bachelor project. This makes communication between group members operate smoothly while using this platform. Discord also features threads and channels for our project where we have differentiated threads and channels for working and resource sharing and management. See Figure 4.1.2.
- Toggl[8] - Toggl track is a great tool to track and manage time. Its intuitive and user-friendly design makes it easy to use, and its comprehensive reporting features provide valuable insight into how you spend your time. Toggl track allows its users to label and visualise their work and provides cooperation workspaces to register and report the users' work.
- Teams[9] - Microsoft Teams is a communication tool for us to communicate in real time. We only use it to communicate with NINA and their delegated tech representatives. We used Teams because of how standardised it is today. This choice was made because Microsoft Teams is recognized as a "default" tool for communicating between organisations/workgroups. Seeing as NINA

already had it configured, it can even be recognised as a standard. We did not question to use another tool, this was because everyone, including NINA, had it configured. This made it easier to plan the meetings for us due to Teams cross-compatibility with Outlook, thus setting up meetings was as easy as sending a mail, with a planned date and time.

- TeamViewer - TeamViewer would be used in tandem with NINA meetings when showcasing the different iterations in order to perform user testing and get valuable feedback.
- VDI - Virtual desktop infrastructure, or VDI is in essence virtualisation technology that hosts desktop environments on a central server or datacenter. Through this centralised server, the VDI will deploy the desktop environments to end clients or users over a network through a virtual desktop image to an endpoint device, defaulted to the researchers at NINA. This is what NINA will use to host our application.
- Docker[10] - Docker is an open source platform that enables developers to build, deploy, run, update and manage containers—standardised, executable components that combine application source code with the OS libraries and dependencies required to run that code in any environment. We would use it in tandem with one of NINA’s technical representatives in order to help them test our application for deployment using docker for their VDI environment. This would be done mostly in collaboration with one of the representatives, Francesco Frassinelli.
- Jupyter [11] - Jupyter is an open source web application that you can use to create and share documents that contain live code, equations, visualisations, and text. We used this to contain our work for AI.
- NextCloud [12] - Nextcloud is an open-source, self-hosted file share and communication platform. It allows users to store and share files, contacts, calendars, and more, ensuring privacy and data security. In our project, we leveraged Nextcloud to share files from the disks we received from NINA (detailed in Section 8.1.1) among our team members. This service was hosted on the home server of one of our members, providing us with a centralized and secure platform to access and manage these crucial data files.

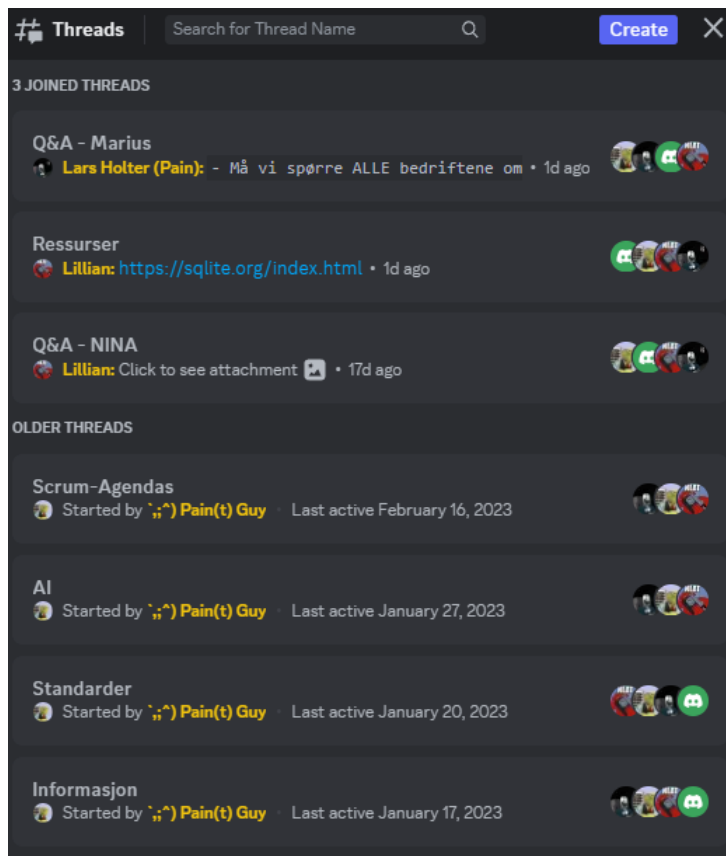


Figure 4.1.2: A list of all threads created, the recently used are the threads that are still relevant for use, and the others are relevant for documentation

4.1.2 Documentation

Throughout the development process it was important that the process was properly documented. In order to keep the quality of the work persistent over the period, it was therefore integral that we would decide upon which documentation tools we would use before starting the process. For writing the reports we decided on utilizing Overleaf, which is an online Latex editor, useful to collaborate on creating scientific documents. Other than Overleaf for creating reports, we used other tools such as ChatGPT as a language checker, Toggl as a time tracker and draw.io for figures. These were used due to their out-of-the-box functionality and our familiarity with these tools. For plotting figures based on data gained from the AI model, we used the library Matplotlib [13], because it is compatible with PyQt [14] as a GUI for the data presented in Section 8.3.

4.1.3 Programming Language

Python is an important language for AI programming, especially in machine learning. Its popularity surpasses other programming languages for machine learning such as Java, or C++ due to the numerous advantages it offers. Python has a comprehensive library ecosystem and provides excellent data visualization options, making it a powerful tool for AI development. It also has low entry barriers, making it accessible to beginners and benefits from strong community support. Python

is highly flexible, allowing developers to customize it to a variety of requirements, and its code-readability simplifies the development process. Furthermore, Python is platform independent, allowing for seamless deployment across all operating systems. It is widely used for constructing regression models and rendering visually appealing graphs, making it an essential choice for effective data visualisation in AI applications.

4.1.4 Prototyping

To reduce the amount of time needed to redesign the program, a good prototype is crucial. Therefore, under the project planning stage, we started looking for tools which would enable us to create a prototype with ease. We had a few demands when it came to choosing a prototype tool. Some of those demands were that it had to be easy to get into and learn, could create a wide set of different size programs and offer a great option of different Widgets, customisation and intuitive ways to create it. We also wanted the tool to be able to create both high- and low fidelity prototypes. Some of the tools we considered were Figma, Adobe XD, and Balsamiq.

After conducting extensive research and evaluating various prototyping tools, our project team ultimately chose Figma as our prototyping tool. We identified several key factors that influenced our decision. First, Figma is a versatile tool that can create both high fidelity prototype and low fidelity prototypes, whereas Balsamiq is primarily suitable for low fidelity prototypes. Second, we selected Figma over Adobe XD because of its cost-effectiveness, since Figma is a free tool, whereas Adobe XD is not. Third, our team had prior experience using Figma extensively throughout our bachelor's program, making it easier for us to quickly start creating the prototype without investing significant time in learning a new tool.

4.1.5 Poetry

In any software development project, managing dependencies and their versions is crucial to ensure consistent and reliable execution of the application. To address this requirement in our project, we chose to utilize Poetry as our dependency management tool.

Poetry is a powerful Python packaging and dependency management tool [15] that simplifies the process of managing project dependencies, including both runtime dependencies and development dependencies. It offers several advantages over other dependency management tools, such as pip and virtualenv:

- **Dependency Resolution:** Poetry provides robust dependency resolution algorithms that ensure compatibility between different packages and their versions. It resolves complex dependency graphs and helps avoid conflicts or compatibility issues that can arise when multiple dependencies are involved.
- **Virtual Environments:** Poetry integrates virtual environment management seamlessly. It automatically creates and manages project-specific virtual environments, isolating the project's dependencies from the global Python

environment. This ensures that the project's dependencies are consistent and independent of other Python projects running on the same machine.

- **Lock File:** Poetry generates a lock file, 'poetry.lock', which pins the exact versions of all dependencies used in the project. This guarantees reproducibility and stability of the application, as subsequent installations or deployments will use the same versions specified in the lock file.
- **Build and Packaging:** Poetry provides tools for building and packaging Python projects, making it easier to create distributable packages. It supports generating source distributions (sdist) and binary distributions (wheel), which are commonly used for sharing and deploying Python applications.
- **Ease of Use:** Poetry simplifies the process of managing dependencies and configuring project settings. Its intuitive command-line interface allows developers to add, remove, and update dependencies with ease. It also provides clear error messages and helpful documentation, making it user-friendly for both experienced and novice Python developers.
- **Integration with PyPI:** Poetry seamlessly integrates with the Python Package Index (PyPI), the official repository for Python packages. It allows easy installation of packages from PyPI, and it can also publish packages to PyPI for sharing with the broader Python community.

Given these advantages, Poetry was the ideal choice for managing dependencies in our project.

4.1.6 Pre-commit

Ensuring high-quality code throughout the project is a top priority. It minimizes bugs, improves readability, and facilitates collaboration among developers. To aid us in this effort, we incorporated pre-commit, a framework for managing and maintaining multi-language pre-commit hooks, into our development workflow.

Pre-commit provides a standardized way to automatically check and enforce code quality standards before changes are committed to the repository [16]. Some of the key features of pre-commit that made it a valuable addition to our project are:

- **Consistency:** Pre-commit helps ensure consistent coding standards across the project, making it easier for developers to understand and work with each other's code.
- **Automation:** Pre-commit checks are automatically run before every commit, catching potential issues before they are committed to the repository. This reduces the likelihood of bad code making it into the codebase and saves time that would otherwise be spent reviewing and fixing code manually.
- **Flexibility:** Pre-commit supports a wide range of pre-built hooks for various languages and tools, and also allows custom hooks to be defined. This makes it adaptable to the specific needs of the project.

- **Integration:** Pre-commit integrates seamlessly with Git, our chosen version control system. This ensures that the checks are run at the appropriate point in the version control workflow.

The tools we run through pre-commit is detailed later in subsection 4.2.1.

4.1.7 Annotation

In the context of training and deploying a suitable detection model for NINA, annotation played a crucial role. Annotation refers to the process of labeling or marking specific objects or regions of interest in a dataset. In this project, the dataset provided by NINA was unannotated, requiring us to annotate the data ourselves for training the AI model. To facilitate this process, we sought out an annotation tool that was user-friendly, functional, and capable of effectively annotating the data to train the AI model.

CVAT (Computer Vision Annotation Tool)[17] is a powerful tool for annotating images and videos to create custom datasets with bounding boxes, polygons, and labelling for said images and videos. The need for CVAT is there because it simplifies annotating data, and adds several quality of life features such as:

- Tracking across frames: CVAT allows users to activate assisted tracking where it adds bounding boxes around the tracked element and moves the box the next frames afterwards. This is not a perfect tracking method; however, it is very helpful.
- Backups and transfers: CVAT allows users to backup their jobs and transfer jobs from one CVAT service to another.
- Shared labels: CVAT allows users to share labels with others in the same workspace, as well as exporting those labels as files.

There is a lot more than just this; however, these features were particularly helpful for us, which makes annotating data a more pleasant experience and improves efficiency in itself. There are several reasons why CVAT is a considerably great tool for annotating the activity of fish in murky water:

- Flexibility: CVAT allows users to define custom annotation labels, which is important when working with a complex and modular underwater environment, which may contain a variety of underwater species.
- Collaborative: CVAT is designed to support collaborative work on annotation projects. This means that it currently allows multiple users to work together on the same data set. This can be particularly useful when working with large volumes of underwater video footage that needs to be annotated and labelled correctly.
- Ease of use: CVAT has a user-friendly interface that makes it easier to annotate images and videos. Users with limited computer vision experience will be able to quickly grasp the UI of CVAT. The tool also includes features such as auto-annotation, video playback, and frame-by-frame annotation, which makes the annotation process faster, easier, and more accurate.

- **Compatibility:** CVAT supports a wide range of annotation formats, including popular formats such as YOLO [18], PASCAL VOC and COCO, as well as custom formats. This makes it easier to integrate with other tools and set better workflows.
- **Accuracy:** CVAT allows users to adjust the automated tracking, which is important when working with complex and difficult-to-annotate underwater images. This ensures that the scope of the annotations always accurately encompasses objects of interest, reflecting the activity of fish and other species that might be of interest to NINA.

We did consider other tools such as: Encord Annotate, Scale, Labelbox, V7 Labs and Dataloop however most of them were not free to use, and did not offer the same out-of-the box experience such as CVAT. We were able to deploy CVAT on our own and able to start annotating almost immediately after deployment after all. The other tools were also too advanced for our use case as well seeing as they offered functionality not necessary for our scope, which would have added overhead to our workflow.

To better facilitate the annotation process and increase our control over the process, we decided to self-host CVAT on a powerful private server rather than using the provided CVAT service. Self-hosting offered several advantages, including:

- **Storage:** With self-hosting, we had virtually unlimited storage space, as the server included multiple high-capacity drives. This facilitated the storage and management of our large datasets without worrying about space constraints.
- **Processing Power:** Our server boasted a powerful CPU and GPU, giving us the possibility of deploying our own models for automatic annotation.
- **Cost Efficiency:** Self-hosting saved us from subscription costs for automated annotation services. We were able to utilize the robust features of CVAT without additional expenditure.

The hardware specifications of our self-hosted server are provided in Table 4.1.1.

Table 4.1.1: Hardware specifications of the self-hosted server

Component	Specification
CPU	Intel Core i7-7700K 8-Core Processor
GPU	NVIDIA GeForce GTX 1070
RAM	32.0 GB
SSD	2x 1TB NVMe SSD, 1x 250GB SSD
HDD	2x 10TB HDDs

In general, CVAT is a great tool for annotating fish in murky water due to its flexibility, collaborative features, ease of use, compatibility with other tools and workflows, and precision. It can help researchers and practitioners identify and study fish species and other objects of interest in underwater environments and develop more effective strategies for managing and protecting these ecosystems.

4.2 Technologies

The application utilizes several different libraries and modules. These are employed to implement functionality such as video processing, file manipulation, local database, and the movement detection. Some of the libraries and modules does not provide functionality for the application, but rather are used to ensure the quality of the program. The selection of these tools and reasoning behind it are detailed below.

4.2.1 Quality Assurance

In order to ensure the quality of the code we wrote, we utilized a set of different tools. These are shown in the list below. For more information on the usage of these tools see Chapter 9.

- PEP 8 is a widely accepted Python code formatting guide emphasizing readability and maintenance.
- Black is a strict Python code formatter that follows PEP 8 guidelines and reduces decision fatigue.
- Pylint is a comprehensive Python linter that enforces coding standards and performs in-depth code analysis.
- mypy is a seamless static type checker for Python with comprehensive capabilities.
- isort organizes and sorts imports in Python code, improving readability and ensuring correct placement and formatting.

4.2.2 GUI

In the development of an application, the Front-end is the component the users interact with. Accordingly, its design must prioritise the target user group. In our case, the target users are the researchers at NINA, who have been identified to have limited technological expertise. Therefore, it was crucial to create a Front-end interface that is visually and practically intuitive, while also providing clear and concise information. It was also important for our group (the development team) to utilise this process as an opportunity to facilitate learning.

After extensive research and consideration, our group identified two primary options for consideration in the development of the Front-end, namely PyQt and TKinter. Both options presented viable possibilities given our goal. However, it was necessary to evaluate the respective advantages and disadvantages of each package before choosing one, which can be seen in Table 4.2.1.

In conclusion, we have opted to make use of PyQt for the Front-end of our application. This decision was based on a few factors. For one PyQt seemed to be more prevalent within the industry as mentioned in pythonGUIs [19]. It also proved to be more configurable compared to TKinter and can facilitate the execution of external processes when needed. These advantages would allow us to use this project to learn and expand our abilities with Front-end development tools in Python.

Table 4.2.1: Table with pros and cons for the Front-end options PyQt and TKinter

	PyQt	TKinter (customTKinter)
Pros	<ul style="list-style-type: none"> • More widely used in the industry, thus it might be relevant later • It may execute external programs • Qt designer allows for more visual design of application • Includes many useful frameworks and templates 	<ul style="list-style-type: none"> • Simple to understand • Included in python (customTKinter is not) • Available for commercial use, however this is not relevant for our project
Cons	<ul style="list-style-type: none"> • Considered more complicated to learn • Need licence for commercial use, however this is not relevant for our project • Lacks some python-specific documentation 	<ul style="list-style-type: none"> • Struggles with scaling and customization • TKinter need third-party package to have a native and intuitive look and feel • Custom TKinter is not official

4.2.3 Local Database

When planning the application we figured that a local database would assist the functionality of the program. The implementation of this would allow for efficient and organised management of data across the program and enabling the program's components to have easy access to the necessary data. The database is also useful to grant the user the ability to view and analyse all the data that will be saved within the database.

The choice of tool that we made for our database was SQLite. SQLite is a lightweight and portable database management system that is used to develop local databases. It can therefore create self-contained and server-less databases, and it does not require a separate server process or installation software. This makes it easy to deploy and manage. Furthermore, SQLite is platform independent, meaning that it can be used on different OS. SQLite has also been designed to be efficient and have low memory usage.

There were however a few different other options that we did consider, such as MySQL, TinyDB, MongoDB and MariaDB. However even if they were viable alternatives they were not as optimal to our application as SQLite. MySQL

and MariaDB requires a separate database server, which SQLite doesn't require, thus complicating and increasing the resources consumed during the setup of the database. Considering the self-contained nature and scale of the application, MongoDB would neither be the most favorable for our system as its large scale and distribution is not required. TinyDB is a useful module for small-scale projects and databases which appeared promising. However on closer inspection it appeared too small-scale and it did not include the comprehensive SQL feature set needed for relational data modeling that we needed for our database.

In summary, SQLite is a popular and efficient database management system that is ideal for local databases. Furthermore, the library allowed us to efficiently create a server-less local database that would not take too much computing power from the rest of the program, which is what we needed. Thus, deeming the other options less suitable for the purposes of our application.

4.2.4 File Generation

For the purpose of our program it was known that it would need to generate a file with the application's findings. To generate a file containing the collected data, it was essential to employ specialised libraries.

No specified file format was requested during the project planning. It was thus decided to allow the user to choose from a set of preset choices. The preset choices were CSV, XLSX, FPDF and XML. All of these choices are flexible and has each their accommodations for what the user may need from the report. The PDF format were chosen due to it preserving its layout across devices and files in this format are easily shared and printed. XML is a format that is highly customizable and allows for the storing of metadata along side the data. Where the XLSX and CSV are both able to be opened and altered as spreadsheets, which makes them suitable for their readability and the possibility to do further in depth analysis of the data presented.

The libraries necessary to implement reading and writing of these file formats are as follows.

- **csv module:** This is a built-in Python module for working with comma separated values (CSV) files. Provides functionality for reading and writing CSV files. CSV files are a common file format to store tabular data, and the CSV module makes it easy to read and write these data using Python.
- **xlsxwriter:** This is a third-party Python library for working with Microsoft Excel files in XLSX format. This provides functionality for creating, reading, and modifying Excel files. XLSX files are a common file format to store spreadsheet data, and the xlsxwriter library makes it easy to work with these files using Python.
- **FPDF:** This is a third-party Python library for creating PDF files. Provides functionality for adding text, images, and other elements to a PDF document. PDF files are a common file format for sharing documents, and the FPDF library makes it easy to create PDFs using Python.
- **Minidom:** This is a built-in Python module for working with XML files. This also provides functionality for parsing and creating XML documents.

XML files are a common file format to store structured data, and the Minidom module makes it easy to work with these files using Python.

4.2.5 Video Processing

Video processing was an essential component of the project application. This is to implement the crucial features of eliminating non-interesting portions of video, and overlaying annotations on output videos. It was therefore important to find a tool which could perform these tasks. For this we found the tools detailed in the list below to be suitable for this purpose.

- FFmpeg [20] is a widely-used and powerful multimedia framework that can be employed to process video and audio files in various formats. It is also a powerful open-source tool for processing video files, as it provides a wide range of features for manipulating, converting, extracting, resizing, cropping, applying various elements (such as images, text and sound), and reformatting video files. It also supports a wide range of codecs.
- PyAV [21] is a powerful and versatile tool for processing videos. It provides a comprehensive set of functions which provides Python bindings for FFmpeg intended for manipulating video files, including encoding, decoding, transcoding, and streaming. It also supports a wide range of video formats, including AVI, MP4, and MKV. In addition to being highly efficient, allowing for fast and efficient video processing. PyAV is also open source and free to use.

We decided to utilize these two libraries due to the combination enhancing the integration of the video processing functionality. The PyAV library, which provides Python bindings for FFmpeg, would allow for seamless integration of the video processing capabilities of FFmpeg into our project. This integration would ensure the facilitation of video cutting and overlaying of annotations on the output video while maintaining performance and flexibility for our application.

4.2.6 Artificial Intelligence

To detect fish- and other underwater life activity we have to use a tool specifically made for object detection. In this project, we have chosen YOLOv8 [22] as our object detection model over other detection models, including other Convolutional Neural Network (CNN)s and Region-Based Convolutional Neural Network (R-CNN)s, and non-AI motion or object detection algorithms. The rationale behind this decision is based on the following key factors:

- **Unified Detection Framework:** Unlike R-CNNs that employ a two-stage detection process (region proposal and classification), YOLOv8 follows a single-stage detection approach. This allows YOLOv8 to perform both localization and classification simultaneously, making it faster and more efficient than two-stage detectors [23].

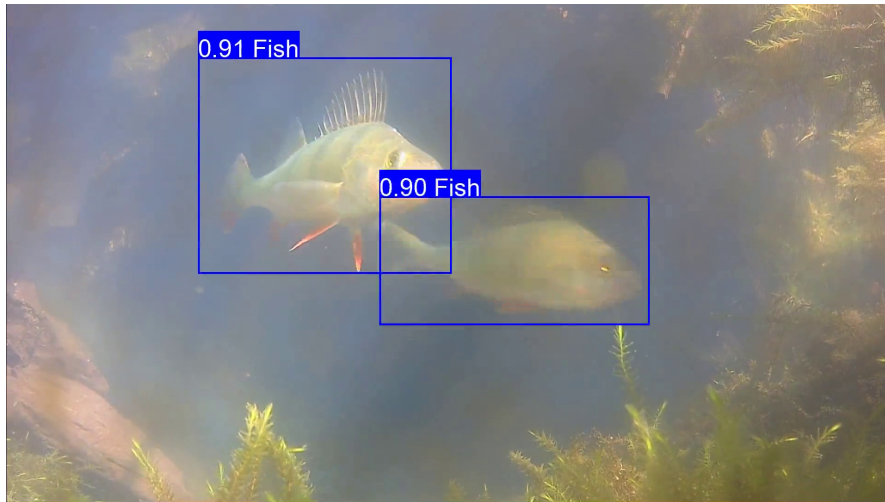


Figure 4.2.1: Example of the annotation overlay on an output video. The annotations, including the labels of the detected objects and its associated confidences, are overlaid on the video frame.

- **Speed:** The You Only Look Once (YOLO) approach is known for its exceptional speed [23], which allows for real-time object detection in video streams. This is crucial for our application, as NINA accumulates a large backlog of videos, and the detection process must be efficient. It also improves on speed over its predecessor yolov5 [24] as illustrated in Figure 4.2.2
- **Accuracy:** YOLOv8 maintains a good balance between speed and accuracy. While it may not be the most accurate object detection model, its performance is sufficient for our application, where the primary focus is on detecting the presence of fish rather than identifying species, and detecting ranges rather than individual frames. See Figure 4.2.2 to see a performance comparison.

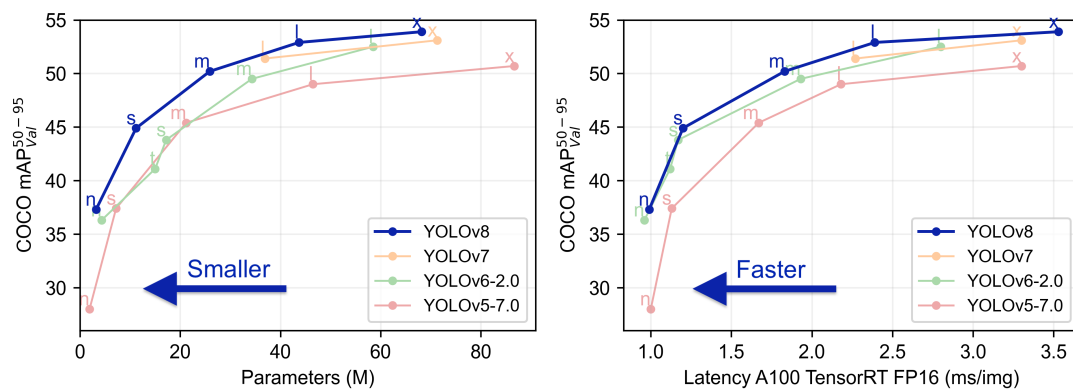


Figure 4.2.2: YOLOv8 compared to other YOLO models. Figure from Ultralytics repository [22].

- **Lower Epochs:** YOLOv8 requires fewer epochs for training compared to its predecessor, YOLOv5, which enables faster iteration when training, which allows for more tweaking and experimentation to improve performance.

- **Partial Object Detection:** The model is robust to occlusion, meaning that YOLOv8 can detect partially visible objects, which is important for our application, as fish may not always be entirely visible in the video frames [23]. In murky water environments, where fish can be partially hidden by plants or other debris, this is an important feature that can improve the accuracy of fish detection. This is also an important feature we had in mind when deciding on our annotation rules discussed later in Section 8.1.2.
- **Scalability:** YOLOv8 can be easily scaled to different resolutions and is available in various sizes (YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x), providing options for trade-offs between speed and accuracy based on the specific requirements of the project and what NINA requires at any time.

Table 4.2.2: YOLOv8 Model Sizes. Source: Ultralytics repository [22]

Model	Size (pixels)	mAP (val 50-95)	Speed (CPU ONNX, ms)	Speed (A100 TensorRT, ms)	Params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- **State of the Art and Ongoing Development:** YOLOv8 is a brand new state-of-the-art (SOTA) object detection model, and it is still under active development. This ensures that the model benefits from the latest research advancements and optimizations. As a result, YOLOv8 is likely to receive continuous improvements in terms of speed, accuracy, and robustness, making it a future-proof choice for the project [22].
- **Superiority over Non-AI Algorithms:** Traditional non-AI object detection methods, such as background subtraction, optical flow, feature-based detection, Gaussian mixture model (GMM), and template matching, encounter challenges in the complex underwater environment for various reasons:
 - *Background Subtraction:* Background subtraction methods rely on estimating a static background and detecting moving objects by comparing the current frame with the background model [25]. In underwater environments, background subtraction methods struggle due to rapid and drastic background changes caused by water currents, moving marine plants, or floating debris. Additionally, low visibility and varying illumination conditions can lead to inaccurate background modeling, resulting in false detections or missed objects.
 - *Optical Flow:* Optical flow methods estimate the motion of objects based on the apparent motion of their intensities in consecutive frames [26]. However, these methods can be sensitive to noise, which is often

present in underwater environments. They may also fail in areas with low texture, common in underwater scenes, since optical flow relies on the presence of distinguishable features to compute motion [27].

- *Feature-based Detection*: Feature-based detection methods identify objects by detecting and matching distinctive features, such as corners or edges, between frames [28]. Underwater environments, characterised by low visibility, varying illumination, and floating debris, can alter the appearance of these features, making them difficult to detect and match accurately.
- *Gaussian Mixture Model (GMM)*: GMM-based methods model the background using a mixture of Gaussian distributions to identify moving objects [29]. Although these methods can handle some level of dynamic backgrounds, they struggle with rapidly changing underwater backgrounds due to water currents, moving marine plants, or floating debris. Furthermore, low visibility and varying illumination can cause GMM-based methods to incorrectly model the background or foreground, resulting in false detections or missed objects [30]. GMM methods usually require a period of training on a stable background, which can be challenging to obtain in underwater environments.
- *Template Matching*: Template matching methods search for a predefined template in an image by comparing the template with local regions in the image [31]. These methods struggle in underwater environments due to several factors, including low visibility, varying illumination, and the presence of similar-looking objects or floating debris, which can cause false detections. Additionally, template matching methods are not robust to changes in object scale, rotation, or perspective, which are common in dynamic underwater environments [31].

Compared to these traditional non-AI object detection methods, CNN-based models such as YOLOv8 are better suited to handle the challenges posed by underwater environments. They can learn to recognise and generalise essential object features, despite low visibility, varying illumination, floating debris, dynamic backgrounds, or changes in object scale, rotation, or perspective. This robustness and adaptability make them more accurate and reliable in detecting objects in underwater environments [23, 32].

Given the aforementioned advantages, YOLOv8 was deemed the most suitable choice for our fish detection task in murky underwater environments. Its combination of speed, robustness, and accuracy makes it an ideal choice for the specific challenges presented by this project.

4.3 Technology Integration and Interaction

After considering and choosing the technologies for developing the application, it was important to map out how these libraries and modules would be integrated within the system and how they were going to interact. Figure 4.3.1 below is an

expanded version of Figure 2.2.2, which visualizes the how the technology discussed within this chapter was going to be integrated into the system.

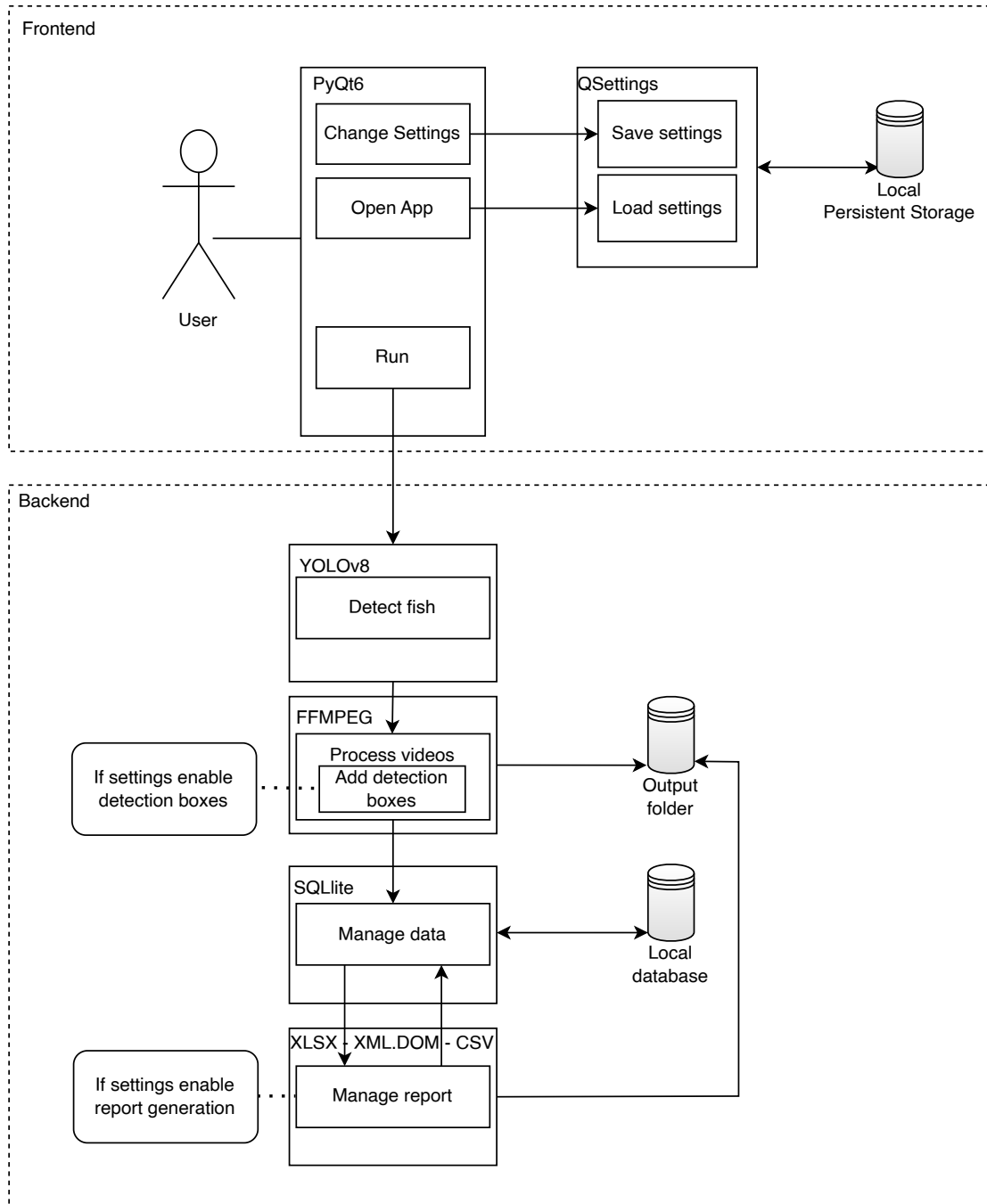


Figure 4.3.1: The functional specifications laid out, and integrated with the intended technologies.

SYSTEM ARCHITECTURE

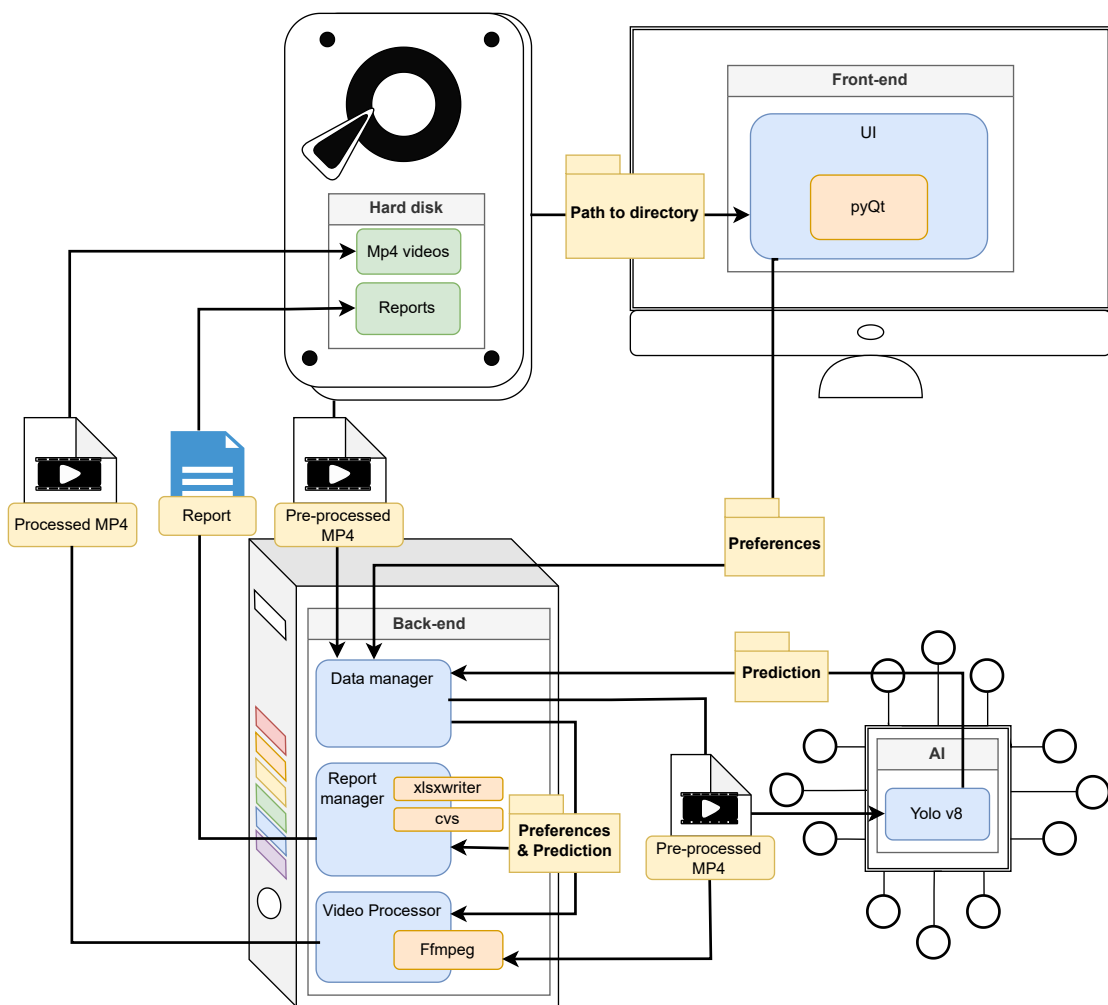


Figure 5.0.1: The full system architecture, for an in depth view of each part of the figure, see Figure 6.2.1 and 6.3.1.

Prior to program development it was important to establish the architecture of the full application. This was to provide clear understanding between all group members of each components' purpose and interdependencies. This facilitated effective communication about the system between group members and expedited

the development process. The system architecture, as visualized in Figure 5.0.1, comprises of the four primary components: Front-end, Back-end, AI, and disk, each which is integral to the overall system.

5.1 Disk

The disk component within Figure 5.0.1 represents the storage in which the user stores the preprocessed videos, the processed videos and the process report. This conceptual storage encompasses both the directory from which the preprocessed videos are fetched and the directory to which the processed videos and reports are saved. This may thus be the computer's local storage and or external storage devices.

To run the process the application provides, an input and output directory must be defined. The user interface interacts with the disk component through user input, where the user specifies the directory paths prior to their transfer to the Back-end component. The Back-end utilizes the input path to gather the preprocessed videos to send to the AI and video processor, while it uses the output path to save the process video from the video processor and the report from the report manager.

5.2 Front-end

The Front-end component of the system consists of a User Interface such that the user can interact with the application. The User Interface is utilized by the Back-end to collect data from the user, such as processing preferences, and input and output directories. Once the data is gathered, it is then passed on to the Back-end component for processing. The Front-end thus plays a crucial role in the system architecture by facilitating communication between the user and the Back-end component.

5.3 Back-end

The Back-end component is the connection point of all the other components within the system, which can be seen in Figure 5.0.1. This component is responsible for ensuring that data is stored and transferred to the correct components, that the AI prediction and video processing procedure runs smoothly and that the information about the process is recorded and written to a report. In order to perform these duties the Back-end component is further divided into a data manager, a report manager, and a video processor.

5.3.1 Data Manager

The data manager component is a script within the Back-end, that manages the data within the system. It ensures the preferences and the directories from the Front-end is transferred to the corresponding components of the system. Such as the preprocessed videos are sent to the AI model, and the prediction from the

model and preferences from the Front-end are sent to the report manager and video processor. The local database, referenced within Section 4.2.3, is created and managed by this part of the Back-end, to save the data about the videos and predictions.

5.3.2 Report Manager

The report manager is another script within the Back-end. This script writes a file into the output directory based on the findings of the AI model through a connection with the local database managed by the data manager. The format of the report is based on the user preferences, as the user decides on a format among a set of preset choices. Originally it was planned for there to be four preset choices, XML, PDF, CSV and XLSX, thus utilizing the Minidom, FPDF, CSV module and xlsxwriter libraries. However due to changes per request these choices was cut into only CSV and XLSX, thus only utilizing the CSV and xlsxwriter libraries within the final product.

5.3.3 Video Processor

The video processor is a component that processes the videos using FFmpeg through the PyAV library. The component edits a video by cutting out frame-ranges based on the AI's predictions and assemble the cuts into a new processed video. Thus creating a processed video that does not have all the dead time that the original video had in between wildlife. The video processor uses FFmpeg to process the video and apply any additional options specified by the user in the Front-end.

5.4 AI

The AI component is an AI model that is used to detect when there is fish present in a video. This component uses the YOLOv8 model with our weights. The way in which the model detects the fish within an MP4 video file is by sifting through all frames of the video. For each frame the model processes the frame and predicts whether it contains aquatic life or not. When it has performed prediction on all the frames, it will create frame-ranges based on when it detects activity over multiple frames close to each other thus building up a prediction dataset consisting of several frame-ranges that will later be used in the video processor. The model then returns the prediction data to the Back-end component to be stored by the data manager.

IMPLEMENTATION

This chapter provides a comprehensive overview of the implementation process, focusing on essential components of our software. It encompasses the selection of libraries and functions, the tools employed to facilitate project advancement, and an explanation of the overall solution. For a more detailed exploration of the GUI, please refer to Chapter 7. Similarly, Chapter 8 delves into the intricacies of object detection, providing an in-depth analysis of the underlying mechanisms.

6.1 Development tools

In this section we will delve into the implementation of various development tools. These tools were used throughout the development process to enhance the project's integrity and quality.

6.1.1 Poetry

To leverage the benefits of Poetry [15], we followed the following workflow for dependency management:

1. **Dependency Specification:** We defined our project dependencies by adding them to our 'pyproject.toml' file, which Poetry uses to manage project settings and dependencies. We specified both direct dependencies required for our application and any development dependencies needed for testing, Linting, or other development-related tasks.
2. **Environment Setup:** Poetry automatically created a virtual environment for our project and isolated it from the system's global Python environment. This ensured that our project's dependencies were contained and did not interfere with other Python projects or system-wide packages.
3. **Dependency Installation:** We used the 'poetry install' command to fetch and install all the project dependencies specified in the 'pyproject.toml' file. Poetry resolved the dependencies, downloaded the required packages from PyPI, and installed them into our project's virtual environment.

4. **Dependency Updates:** As we worked on the project, we occasionally needed to update or add new dependencies. Poetry made this process straightforward. We used the ‘poetry add’ command to add new dependencies and the poetry update command to update existing dependencies. Poetry automatically resolved any conflicts and ensured the compatibility of the dependencies.
5. **Lock File Management:** Poetry generated and maintained the poetry.lock file, which locked the exact versions of all dependencies. This file was committed to version control, ensuring that all developers and deployment environments used the same dependency versions. When deploying the application, we used the poetry install –no-dev command to install only the runtime dependencies, excluding any development dependencies.

In addition to these standard use cases, we also utilized Poetry in our GitHub Continuous Integration (CI) workflow to install dependencies for running tests and pre-commit checks, which we will discuss more in Section 9.5.1. By incorporating Poetry into our CI pipeline, we ensured that the correct versions of dependencies were installed and used during the automated testing and quality assurance processes.

Furthermore, Poetry played a crucial role in managing the installation of PyTorch [33] with CUDA [34] on Windows. While PyTorch does not come bundled with CUDA support on Windows, its Linux counterpart does. To overcome this limitation, we leveraged a tool called poethepoet (poe) [35] within the Poetry ecosystem. We added the following configuration to our ‘pyproject.toml’ file:

```

1 [tool.poe.tasks]
2 torch-cuda = "pip install torch torchvision torchaudio --force-reinstall --no-deps --no-cache-dir --extra-index-url https://download.pytorch.org/whl/cu117"

```

Listing 6.1: PyTorch CUDA Installation Configuration

This configuration instructed Poetry to execute the pip install command with the specified parameters, ensuring the installation of PyTorch with CUDA support on Windows. We then used the command poetry run poe torch-cuda to install PyTorch with CUDA via Poetry.

By incorporating these additional aspects into our usage of Poetry, we enhanced our development workflow, ensured compatibility with GPU acceleration on Windows, and streamlined the installation of project dependencies within our CI environment. Poetry’s flexibility and extensibility made it a valuable tool for managing our project’s dependencies and ensuring smooth development and deployment processes.

6.1.2 Pre-commit

We integrated pre-commit [16] into our development process as follows:

1. **Configuration:** We defined a configuration file, ‘.pre-commit-config.yaml’, specifying the hooks we wanted to use. This file was committed to the repository, ensuring that all developers were using the same set of checks.

2. **Installation:** Developers installed pre-commit locally by running ‘pre-commit install’. This set up the necessary Git hooks to automatically run the checks before each commit.
3. **Automatic Checks:** Whenever a developer attempted to commit changes, pre-commit automatically ran the specified checks on the changed files. If any checks failed, the commit was blocked and the developer was shown an error message detailing the issues.
4. **Manual Checks:** Developers could also run the checks manually at any time using the ‘pre-commit run’ command. This was useful for checking code quality during development, before attempting to commit changes.
5. **Updates:** We occasionally updated the pre-commit configuration to add, remove, or adjust checks as the project evolved. Developers could pull the latest configuration from the repository and update their local setup using the ‘pre-commit autoupdate’ command.

Furthermore, we integrated pre-commit into our GitHub CI workflow. This ensured that the checks were run on every pull request before it could be merged, providing an additional layer of quality assurance. You can read more about that in section 9.5.1.

By incorporating pre-commit into our development process and CI pipeline, we were able to automatically enforce consistent coding standards and catch potential issues early, improving the overall quality of our codebase and facilitating effective collaboration among developers.

6.2 Front-end

The Front-end of the application was implemented utilizing the PyQt6 [14] library as discussed in the Technologies chapter in Section 4.2.2. This library was useful in order to create a GUI that satisfied the requirements in which we had for the user interface.

6.2.1 Initialization and Main Function

The main function is the entry point of the application, it creates a custom logger (Section 9.2) and sets up application settings using our custom settings module, detailed in subsection 6.2.4.

After this, the main function creates an instance of the `QApplication` class and sets up the theme of the application, which is automatic based on the default theme on the computer. Finally, it creates an instance of the `MainWindow` class and shows it.

6.2.2 MainWindow and User Interface

The `MainWindow` class is responsible for creating the main window of the application. The constructor of the class sets the default settings for the window, such as its title, size, and icon. It also creates the layout for the central `Widget`

and adds several sub-Widgets, such as file browsers, options panels, and buttons. These sub-Widgets are grouped collections of Widgets from the PyQt6 library that are created within their own classes. The user interface elements also reflect user-defined settings, ensuring a personalized user experience.

Figure 6.2.1, visualizes the setup of the widgets for the user interface. See Chapter 7, for a more in depth explanation of widget placement.

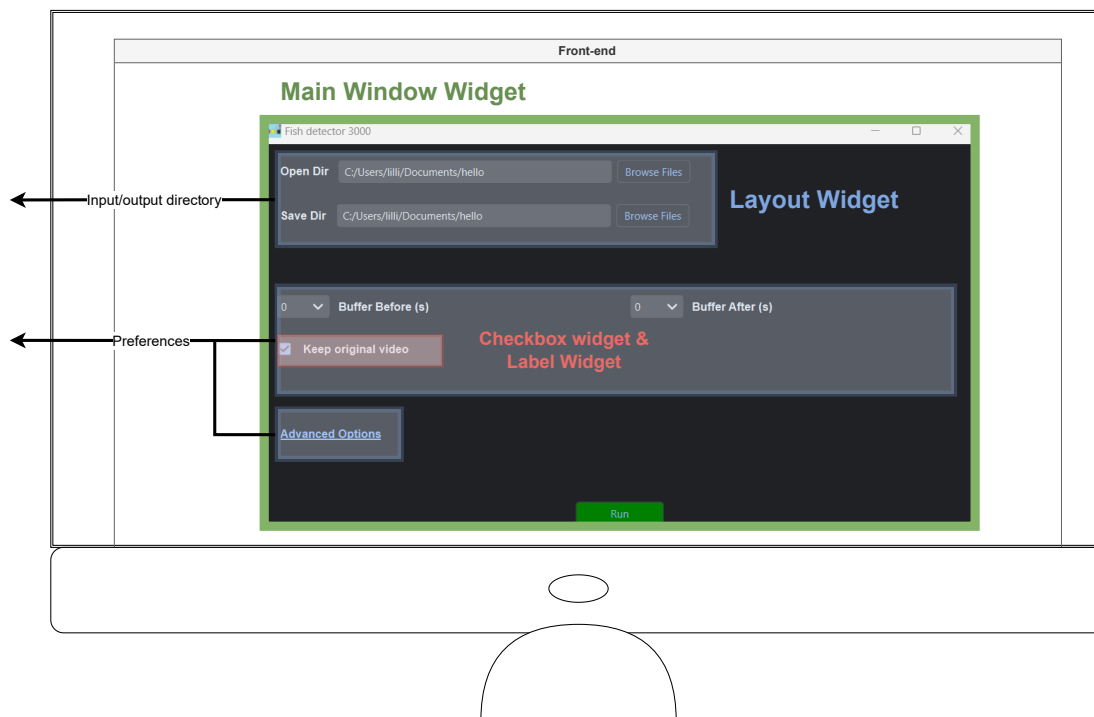


Figure 6.2.1: A visualization of the front-end widget layout.

6.2.3 Processing and Output

A run method of the MainWindow class is called when the "Run" button is clicked. This method checks whether the input and output folders exist and shows an error dialog if they don't. If the folders exist, it creates a DetectionWindow object and executes it. The DetectionWindow class calls upon the Back-end processes of the DataManager, ReportManager and VideoProcessor. These processes detect fish within the video and process the detections into new clipped videos, while the UI of the DetectionWindow shows the progress of the detection and video processing.

6.2.4 Settings

In order to provide customizable experience to the users and save their preferences, we designed and implemented a custom settings module in our application. This module holds variables for settings related to window dimensions, file handling, advanced parameters, and more. It utilizes QSettings from PyQt6, a built-in method for storing settings in PyQt applications, to persist these settings to disk, ensuring that user preferences are remembered across sessions.

To avoid manually writing code for each setting, we took a dynamic approach by automatically generating settings based on module-level variables. The module's default values for the settings are provided as module-level variables which are dynamically added to the settings entries. When a value is set to an entry, it's stored using QSettings, allowing it to be recalled even after the application is restarted. Moreover, the module ensures the integrity of the settings by checking the data types of the values being set, hence avoiding potential application errors.

The detailed script for our custom settings module is included in Appendix I.

6.3 Back-end

This section details the implementation of the Back-end components. This includes the three components; data manager, report manager and video processor. Figure 6.3.1 shows a close up of the back-end from the System Architecture seen in Figure 5.0.1. It visualizes how functions within and in between the components interact with each other. The following sections contains further details on each component's implementation within the system.

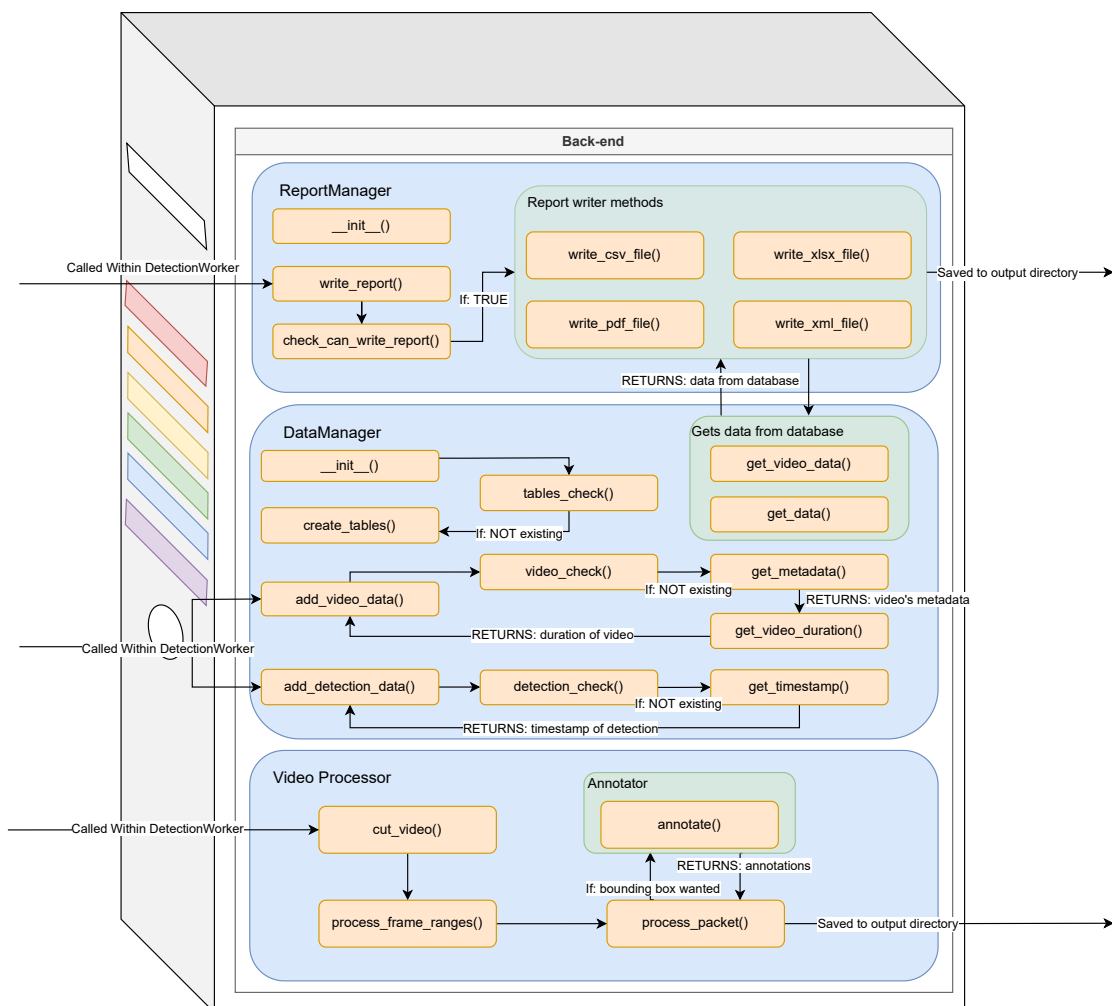


Figure 6.3.1: A visualization of the flow between components in the Back-end

6.3.1 Data Manager

One of the data manager responsibilities is to save information about videos and their corresponding detections in a local database. The database consists of two tables: the video table and the detection table. The video table contains information about the video, such as the title, date saved, duration, and number of detections. The detection table contains a FK (foreign key) to the video it originated from, as well as the start and end times of the detection. See Figure 6.3.2 for the entity relation diagram.

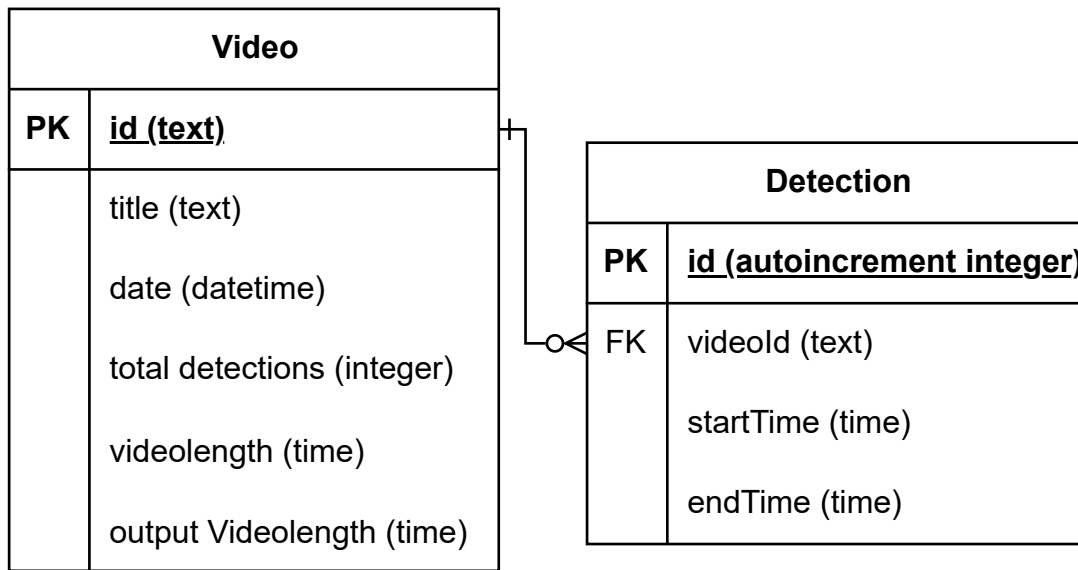


Figure 6.3.2: The entity relation diagram for the local database consists of mainly two tables which are connected through the video id.

The data manager script mainly consists of a class called DataManager. The class utilizes the SQLite library that was introduced within Section 4.2.3, in order to interact with the local SQL database. As it is initialized it connects to the local .db file, or creates one if it is not present. The `__init__()` method ensures to check if the necessary tables are within the file, if not the are generated through the use of the .sql file. The class also has the automatic `__exit__()` method that ensures that the SQL connection is closed after usage of the class. This makes sure that the program doesn't timeout at any point during run time due to too many connections happening at the same time.

The DataManager class consists of several methods that interacts with the local SQL database. The methods `add_video_data()` and `add_detection_data()` appends new data about the videos that are being processed within the tables of the database. It makes sure to check if the data already exists within the database first before fully appending the data. In the case of the data already existing within the `add_video_data()` method stops the inputting process, while

the `add_detection_data()` method will override the existing data with the new data to ensure that the data is updated. These 'add' methods utilizes a few helping methods that gathers or formats data before inputting it into the database. These are methods such as `get_metadata()`, which gets a video's metadata, and `get_timestamp`, which finds the timestamps for when the detected frame ranges starts and ends in the original video.

The data from the local database is retrieved through the two method methods `get_data()` and `get_video_data()`. They are mainly utilized by the report manager to write reports. These methods receive a list of videos, which is the list of video's that has been processed during run time, and they recover the data connected to each of the videos. Each of these methods thus returns a list of data from each of the tables within the database. Where `get_data()` returns a list of data from the detection table, while the `get_video_data()` returns a list of data from the video table.

6.3.2 Report Manager

One of the components planned and implemented for the project was the report manager Back-end component. The report manager generates and saves a report in a format that is determined by the user based on certain preset options. The report includes comprehensive data pertaining the processed videos and any associated detections of fish within. The code used four different libraries; CSV, `xlsxwriter`, `FPDF`, and `Minidom`. All of these libraries was explained within the subsection 4.2.4. However, at a certain stage of the development process, two out of the four methods was discarded. This was grounded in feedback from NINA, who deemed it unlikely that they would utilize the file formats PDF and XML.

The report manager script is primarily made out of a singular class that handles the writing of the file. This class is the `ReportManager` class. The class writes a new file or override the previous file with the same name within the output directory. When initiating the class it require an instance of the `DataManager` class, the path to the output directory and the file format that has been saved within the preferences. The `DataManager` instance is needed in order to utilize the class' `get_data()` and `get_video_data()` methods, as the data within the local database is used to write the reports.

	A	B	C	D
1	Video	Date	Input Videolength	Output Videolength
2	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	27/04/2023	00:07.4	00:07.3
3				
4	Video	detectionID	Start	End
5	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	1	00:00:00	00:07.3
6	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	2	00:00:00	00:07.3
7	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	4	00:00:00	00:07.3
8	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	5	00:00:00	00:07.3
9	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	7	00:00.1	00:05.2
10	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	8	00:00.1	00:05.2
11	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	9	00:00.1	00:05.2
12	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	10	00:00:00	00:07.3
13	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	11	00:00:00	00:07.3
14	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	72	00:00:00	00:07.3
15	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	73	00:00:00	00:07.3
16	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	74	00:00:00	00:07.3
17				

Figure 6.3.3: The structure of the CSV report

The report consists of two main parts; the summary and the list of detections, see Figure 6.3.3. The summary within the report is a list for every video that was processed during run time, and shows the video's filename, video's date, video length and the length of the new video created after the process. This can be seen in Figure 6.3.4. While the list of detections includes every 'frame range' with fish found by the model throughout the run time and consists of the data the detection Id, the filename of the video the detection was found within, the start timestamp within the video and end timestamp within the video. This is better visualized in Figure 6.3.5.

	A	B	C	D
1	Video	Date	Input Videolength	Output Videolength
2	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	2023-04-27	0:00:07.360000	0:00:07.320000
3				
4				

Figure 6.3.4: The structure of the first sheet of the XLSX report, which contains a summary of the videos processed

	A	B	C	D
1	Video	detectionID	Start	End
2	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	1	0:00:00	0:00:07.320000
3	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	2	0:00:00	0:00:07.320000
4	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	4	0:00:00	0:00:07.320000
5	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	5	0:00:00	0:00:07.320000
6	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	7	0:00:00.080000	0:00:05.240000
7	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	8	0:00:00.080000	0:00:05.240000
8	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	9	0:00:00.080000	0:00:05.240000
9	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	10	0:00:00	0:00:07.320000
10	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	11	0:00:00	0:00:07.320000
11	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	72	0:00:00	0:00:07.320000
12	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	73	0:00:00	0:00:07.320000
13	MYGGBUKTA2022-2022-08-13_05-29-48-025 (1).mp4	74	0:00:00	0:00:07.320000
14				
15				

Figure 6.3.5: The structure of the second sheet of the XLSX report, which contains the list of detections that were made within the videos processed

In order to utilize the ReportManager's ability to write report the `write_report()` method must be called. This method first checks if it is possible to write a file within the output directory. This is done by using the helper method `check_can_write_report()`, which attempts to open the file to write. In the scenario in which the opening of the file fails, which usually is due to the file being open, it returns an exception and logs a warning. Passing the check allows the `write_report()` to continue by calling one of the methods corresponding with the file formats that may be chosen by the user.

There are two methods that may be used to write files. These are the `write_csv_file()` and `write_xlsx_file()`. Both functions utilize the DataManager class' `get_data()` and `get_video_data()` methods in such that the data can be written into the report. Depending on the format chosen the file written appears slightly different. The XLSX format allows for multiple sheets within the workbook thus the report written out consists of two work sheets. One with the

summary, see Figure 6.3.4, while the second with the detections, see Figure 6.3.5. On the other hand the CSV format does not allow for multiple sheets and does therefore contain the full report within a single worksheet, see Figure 6.3.3. The two methods `write_pdf_file()` and `write_xml_file()` still exists within the script, but are not possible to access by the user.

6.3.3 Video Processor

The video processor module is responsible for cutting and annotating videos. It contains several functions and classes that work together to achieve this task. Here is a more detailed overview of the main components of the video processor module:

6.3.3.1 Annotator Class

The Annotator class is designed to annotate video frames with bounding boxes and labels for detected objects. It takes a frame, a list of detections, and some optional parameters, such as color, line width, and font size, and returns an annotated frame. The Annotator class has been adapted from the Ultralytics [22] YOLOv8 implementation with some modifications to better suit our application.

In the Ultralytics version (Listing 6.2), the Annotator is initialized once for every single frame, which is computationally expensive due to the loading of the font (even with caching) and other resources. In our implementation (Listing 6.3), we improved the performance by initializing the Annotator only once for the entire video, resulting in significant performance improvements.

To quantify these improvements, we benchmarked the two implementations on a test video that resulted in an output video with 6384 frames, or approximately 3 minutes and 12 seconds. The Ultralytics implementation took an average of 468.65 seconds over three runs, whereas our implementation required an average of 396.57 seconds over the same number of runs. This corresponds to a performance increase of over 18% with our modified approach. The tests benchmarked the time it took for the `cut_video()` function (detailed in subsection 6.3.3.4) to run using the two different implementations.

Furthermore, we made some modifications to the original implementation to enhance its functionality and flexibility. We removed unused features from the Ultralytics version, streamlining the code for better efficiency. Additionally, we opted to use the Python Imaging Library (PIL) instead of OpenCV for image manipulation and drawing. PIL provides more flexibility and produces visually appealing results.

Listing 6.2 demonstrates the annotation implementation in the Ultralytics version, where the Annotator is initialized for each frame. In contrast, Listing 6.3 shows our implementation, where the Annotator is initialized once for the entire video. *Note: These examples are not representative of real code found in the project.*

```
1 def example_annotation_ultralytics(frames, detections):
2
3     for frame, detections in zip(frames, detections):
4
5         # expensive operation
6         annotator = UltralyticsAnnotator(frame)
```

```

7
8     for detection in detections:
9         annotator.box_label(detection.box ...)
10        frame = annotator.result()
11
12    ...

```

Listing 6.2: Example Usage of the Ultralytics Annotation Implementation

```

1 def example_annotation_ours(frames, detections):
2
3     # expensive operation
4     annotator = OurAnnotator(video_width, video_height)
5
6     for frame, detections in zip(frames, detections):
7         for detection in detections:
8             annotator.annotate(detection.box ...)
9             frame = annotator.result()
10
11    ...

```

Listing 6.3: Example Usage of our Annotator Implementation

6.3.3.2 Codec and CRF

The video processor module uses the H.264 codec for video encoding. The H.264 codec is widely supported and offers a good balance between compression efficiency and video quality. The Constant Rate Factor (CRF) parameter is a quality setting used by the H.264 codec. Lower CRF values result in higher quality but larger file sizes, while higher CRF values produce smaller file sizes at the cost of video quality.

The video processor module allows the user to adjust the CRF value to control the quality and file size of the output video. By exposing this parameter, users can fine-tune the trade-off between video quality and file size based on their specific requirements. In the context of NINA, this is particularly helpful when dealing with large-scale video analysis, where optimizing storage space is crucial without sacrificing too much video quality. It can also depend on the type of video, clear videos without much movement will naturally produce less compression artifacts, so a higher CRF might be appropriate to save on storage space.

6.3.3.3 First Implementation: `ffmpeg-python` Library

The first implementation of the video processor (see Appendix I) used the `FFmpeg-python` [20] library, a Python wrapper around the popular `FFmpeg` CLI tool. The library simplifies the process of constructing `FFmpeg` command lines by providing a high-level API for working with video and audio streams. The initial implementation involved generating a command for the `FFmpeg` CLI using the `FFmpeg-Python` [20] library.

However, we encountered issues with this approach due to the slow compilation of the command when processing videos with a large number of detections. The Directed Acyclic Graph (DAG), which the library uses to represent the video processing pipeline, could not finish sorting in a reasonable amount of time, or

would at times just crash without raising any exceptions. As a result, we decided to switch to the PyAV library, which provides a more efficient, low-level interface to the FFmpeg libraries, resulting in faster processing times and better performance for videos with many detections.

6.3.3.4 Implementation Algorithm

The main function of the module, `cut_video()`, ties everything together. Here are the steps performed by the function:

1. Open the input video file using the PyAV library.
2. Set up the output video file with the same dimensions and format as the input.
3. Create an instance of the Annotator class for annotating frames.
4. For each frame range:
 - (a) Seek the input video to the starting frame of the range.
 - (b) Process packets in the range using the `process_packet()` function.
 - i. Decode (Decoding) the packet into a frame.
 - ii. Annotate the frame if necessary using the Annotator class.
 - iii. Encode (Encoding) and mux (Muxing) the frame into the output video stream.
 - iv. Update the progress bar and notify the progress of the video processing.
5. Close both the input and output video files.

The implementation details for the video processor can be found in Appendix I.

6.3.4 The Video Processing Pipeline

Our application's pipeline is a carefully orchestrated process involving a series of components that work collaboratively. The pipeline starts with an input video, which is analyzed frame by frame using the YOLOv8 model. The model is used to predict the presence of fish in each frame, outputting a list of frames where fish are detected. We will discuss this more in-depth in Chapter 8.

6.3.4.1 Turning Frames Into Ranges

One of the crucial steps in our pipeline is transforming the list of detected frames into frame ranges. This process is implemented in the `detected_frames_to_ranges()` function. It helps to compensate for potential inaccuracies in the YOLOv8 model's predictions. There could be instances where fish are present in consecutive frames, but the model fails to detect them in a few. To handle such cases, we introduce a concept of a *frame_buffer*. This *frame_buffer* allows for a certain number of consecutive undetected frames to still be considered part of the ongoing range, as

long as the number of these 'dead' frames does not exceed the defined threshold. The implementation details of this function can be found in the code provided in Appendix I.

6.3.4.2 Incorporating Buffer Time and Managing Overlapping Clips

To enhance the viewing experience and provide additional context in our output videos, we incorporated a feature to add buffer time before and after each detected range. This feature, requested by NINA, allows the viewers to see a bit of the scene before a fish enters and after it exits the frame.

However, adding buffer time resulted in overlapping clips. To address this, we developed the `add_buffer_to_ranges()` function. This function works in two steps: first, it extends each range by adding the defined buffer time to the start and end. Then, it merges any ranges that overlap as a result of this extension. The detailed implementation of this function is available in Appendix I.

6.3.5 Time Estimation in Video Processing

An essential feature in the video processing pipeline is the estimation of the remaining time until the completion of the task. This feature, requested by NINA, provides the user with a clear expectation of how long the processing task will take, which is particularly useful when dealing with a large number of videos.

The function responsible for this calculation is the `update_time_prediction()` function (see Appendix I). It is designed to provide a dynamic estimation of the remaining processing time based on several factors.

The function is invoked during the processing of each video, where it computes the remaining time based on three primary parameters:

- The progress of the current video: Calculated as the percentage of completed frames out of the total frames.
- The elapsed time since the start of the video processing task.
- The total number of videos left to be processed.

The function operates by first determining the total elapsed time since the start of the video processing task, and the elapsed time for the current video being processed. The progress of the current video processing is computed as a ratio and used to estimate the time remaining for the completion of the current video.

If more than one video is being processed, the function calculates the average time spent per video based on the videos processed so far. This average is then used to estimate the time left for the remaining videos. If the current video is the first one being processed, the total time estimated for the current video is used as a benchmark for the remaining videos.

The total time left is the sum of the time left for the current video and the estimated time for the remaining videos. The result of this calculation is then converted into a human-readable format and displayed to the user in the upper left corner of the worker window as visualized in figure 7.3.3.

This feature improves the overall usability of the application by giving users a clearer expectation of the processing duration, allowing for better time management and planning.

6.4 AI Model

To implement the YOLOv8 model, we use the ultralytics library [22], which allowed us to import the model, initialise it with appropriate weights and use the `predict()` function to obtain prediction results for a given image. However, the initial performance of this implementation was not satisfactory, as it did not fully utilise the GPU.

```

1 from ultralytics import YOLO
2
3 from tools.timer import Timer
4
5 model = YOLO("yolov8s.pt")
6 with Timer("Ultralytics predict"):
7     results = model.predict(
8         source=r"myggbukta.mp4", verbose=False
9     )

```

Listing 6.4: Ultralytics Prediction Benchmark

Ultralytics predict took 46.28121638298035 seconds

The implementation details for the Timer class can be found in Appendix I.

The test video used for this benchmark was a 2-minute video, and the benchmark was run on the hardware detailed in Table 8.2.1.

To improve performance and better utilise the GPU, we developed a custom implementation using a batch processing approach:

```

1 import threading
2 from pathlib import Path
3
4 from app.detection import detection
5 from app.detection.batch_yolov8 import BatchYolov8
6 from tools.timer import Timer
7
8 model = BatchYolov8(Path(r"yolov8s.pt"))
9
10 with Timer("Our predict"):
11     frames_with_fish, results = detection.process_video(
12         model=model,
13         video_path=Path(r"myggbuktav2.mp4"),
14         batch_size=32,
15         max_batches_to_queue=4,
16         output_path=None,
17         stop_event=threading.Event(),
18     )

```

Listing 6.5: Our Prediction Benchmark

Our predict took 10.909997463226318 seconds

The benchmark results clearly demonstrate the significant performance improvements achieved with our custom implementation. The initial implementation using the ultralytics library took 46.28 seconds to process a 2-minute video, which was not optimal considering our hardware specifications detailed in Table 8.2.1. However, through our custom implementation, leveraging a batch processing approach, we were able to reduce the processing time to just 10.91 seconds. This substantial improvement showcases the efficiency gains obtained by optimizing the utilization of the GPU. It highlights the importance of tailoring the implementation to our specific needs and demonstrates our ability to enhance the performance of the AI model for real-time processing videos. *Note: These examples are not representative of real code found in the project.*

6.4.1 Image Loader

The Image Loader is designed to efficiently load and preprocess video frames for object detection. One of the primary issues with the Ultralytics implementation is the inefficiency of their data loader, which processes single images in a sequential manner, leading to waiting on the GPU between frame transfers. To overcome these limitations, our solution uses a multithreaded approach with one frame loading thread and multiple worker threads to process the loaded images.

- **Frame Loading Thread:** This thread continuously reads the video frames and stores them in an array with a size of *batch_size*. Once the array is filled, it adds the batch to an *unprocessed_batch_queue*.
- **Worker Threads:** Worker threads retrieve unprocessed batches from the *unprocessed_batch_queue* and perform Letterboxing on each image in the batch. After processing all images in the batch, the worker thread places the processed batch into a *processed_batch_queue*.
- **Batch Retrieval:** A `get_batch()` function is available to request a processed batch from the *processed_batch_queue*, returning either the batch data or `None` if no batch is available.

This multithreaded solution allows a detection worker thread to fetch a ready batch, transfer it to the GPU, and wait for data from the GPU while concurrently loading and processing new images from the video in the background. It also reduces the number of transfers to and from the GPU due to batching the images. Figure 6.4.1 illustrates the Image Loader system.

The Image Loader employs the `prepare_images()` method from the `BatchYolov8` class (detailed in subsection 6.4.2) to preprocess the images by Letterboxing them before they are sent to the YOLOv8 model for inference. By integrating the Image Loader with the `BatchYolov8` class, the system achieves a streamlined and efficient workflow for video frame processing and object detection.

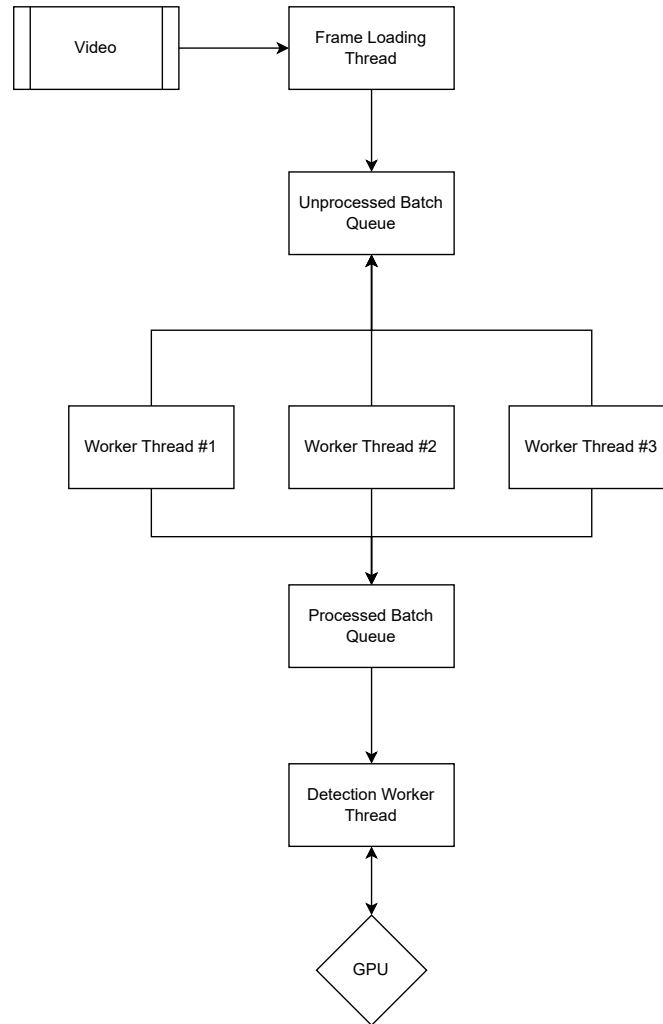


Figure 6.4.1: Illustration of the image loader system with frame loading thread, worker threads, and batch queues.

6.4.2 BatchYolov8: A Custom Object Detection Model

BatchYolov8 is a custom class designed for running inference on video streams using the YOLOv8 object detection model. It is tailored to suit the specific requirements of the project and offers several improvements over the default implementation provided by Ultralytics. In this subsection, we will discuss the structure, features, and benefits of the custom BatchYolov8 class. You can find the implementation details in Appendix I.

- Initialization:** The constructor of the BatchYolov8 class accepts several parameters, including the path to the model weights, device to run on (CPU or GPU), image size, confidence and IoU thresholds, and optional parameters for data augmentation, agnostic NMS, classes, and colors. During initialization, the model is loaded, device selected, and image size checked for compatibility. The class also supports half-precision inference on compatible hardware for improved performance.
- Preparing Images:** The `prepare_images()` method accepts a list of images or a single NumPy[36] array as input. It reshapes and pads the images

to the required size, normalizes them, and converts them into a torch tensor. The prepared images are then sent to the YOLOv8 model for inference. This method is utilized by the image loader to offload the work from the detection worker thread, to the image loader worker threads mentioned in 6.4.1.

- **Prediction:** The `predict_batch()` method runs inference on a batch of images and applies non-maximum suppression (NMS) to the model's output. It then processes the results and returns a list of predictions, including bounding box coordinates, class names, confidence scores, and colors for visualization. Users can also specify a maximum number of objects to return per image for each class. This feature helps improve performance both during inference and when outputting a video with annotations, as too many annotations may slow down the process and obscure the fish in the output video.
- **Efficient Image Reshaping:** The `reshape_copy_img()` method uses the Letterboxing technique to efficiently reshape and copy input images while preserving their aspect ratio. This ensures minimal distortion during inference and improves the model's detection accuracy.
- **Batch Image Padding:** The `pad_batch_of_images()` method pads a batch of images to the same size, which is necessary for running inference on batches. It fills the padded area with a constant rgb value (114, 114, 114) as a neutral grey color to minimize the impact on the model's predictions.
- **Post-Processing:** The `min_max_list()` and `max_objects_filter()` methods process the raw detection output to create a list of bounding boxes with class names, confidence scores, and colors. They also enable filtering the results based on the maximum number of objects per class.

The main benefit of the `BatchYolov8` class is that it enables batch processing of images, which utilizes the GPU more and provides more control over the inference process. The implementation is based on a pull request from "Ownmarc" on GitHub [37].

Compared to the default Ultralytics implementation, the custom `BatchYolov8` class offers the following benefits:

- Batching of images.
- Streamlined initialization, prediction, and post-processing methods tailored for this specific project by exposing functionality to the image loader.
- Graceful stop/shutdown between to accommodate the stop controls referenced in Section 7.1

GRAPHICAL USER INTERFACE

This chapter provides an in-depth discussion of the decisions made with regards to GUI elements, usability, visual feedback, and aesthetics for the computer application developed as part of this research project.

7.1 Prototyping

During the initial stages of the Bachelor's thesis development process, the team commenced work on the design of the GUI prototype. The GUI prototype was developed through the utilization of Figma [38], which is a digital tool used for interface design. See section 4.1.4 in Technologies for the reasoning in choosing Figma. The prototype was refined to meet the desired specifications and expectations of the team. Thereafter, it was presented to the client, NINA, for feedback.

In the majority of responses received, participants praised the GUI for its intuitiveness and aesthetically pleasing design. Nevertheless, there were several aspects the respondents recommended for improvement. These included the incorporation of more distinct pause and stop controls during the operation of the software, the addition of advanced customization options for the video output and report generation, and minor enhancements to the visual appearance. Subsequent to the presentation of the initial prototype, the team diligently integrated the client's feedback, embarking on a series of iterative refinements to optimize the GUI and ensure it adhered to the requisite standards. A visual representation of the GUI is exhibited in Figure 7.1.1.

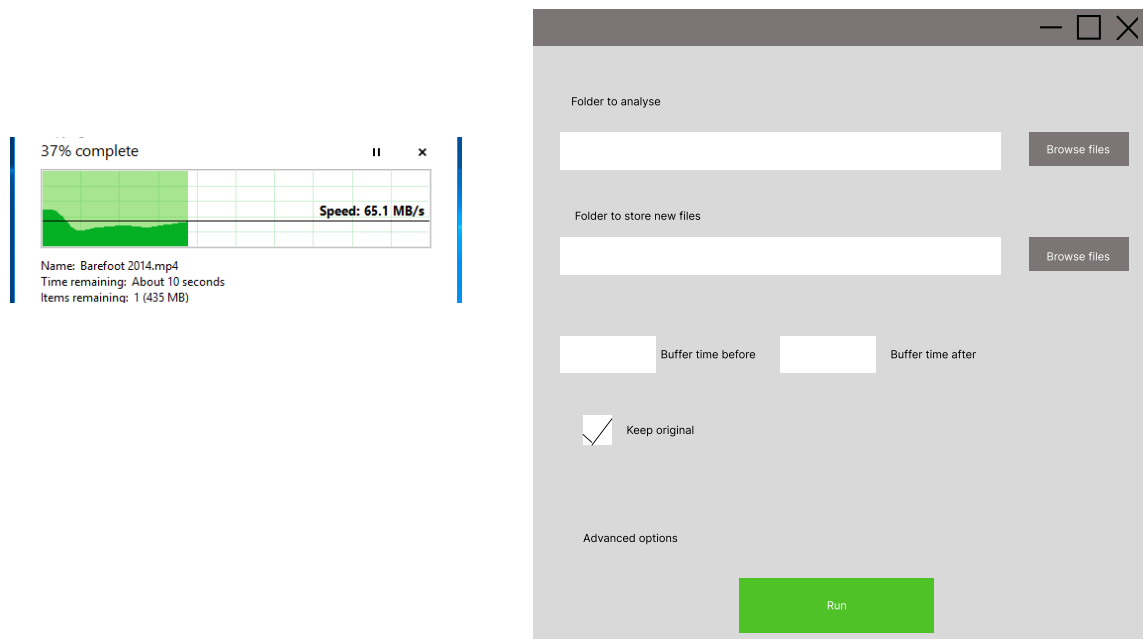


Figure 7.1.1: Prototype of the GUI.

7.2 Ergonomics

In developing our software, the primary objective was to create an interface that was both easily accessible and required minimal time for users to become proficient in its operation. To accomplish this, we employed a simplistic aesthetic while adhering to a set of design principles articulated by Don Norman in his influential book, *"The Design of Everyday Things"* [39]. Key principles that guided the GUI design included visibility, affordance, mapping, feedback, consistency, constraints, and flexibility.

Initially, our focus was on ensuring optimal visibility, which aligns with the goal of promoting ease of use and rapid user comprehension. We adopted the F-pattern layout, a visual scanning pattern identified by the Nielsen Norman Group in 2006 [40], wherein users typically scan the screen starting from the top-left corner and move across the top of the page, then down the left-hand side. This layout fosters a natural and intuitive navigation experience.

Next, we emphasized affordance in the design of the GUI elements, ensuring that each component clearly conveyed its intended function. For example, buttons were designed to be prominent, with the "run" button featuring a green color to signify the start of a process, while drop-down menus included arrows and adjacent text.

In accordance with the F-pattern layout, we arranged the GUI elements logically to reflect the order of task execution and the relationship between actions. Advanced options were grouped under a designated "Advanced Options" category, indicating that these settings were not essential and were intended for more sophisticated customization.

Feedback was another crucial consideration in our GUI design. We implemented features such as Drop-down menus that displayed the selected option when clicked, check-boxes that visually changed when activated, and a clear progress bar with supplementary information during process execution. Customizing the progress bar, as opposed to utilizing a standard operating system progress bar, facilitated a better understanding of the process's progression and the number of detections made in the videos. Tool-tips were also incorporated to provide additional information when users hovered over individual elements.

To further streamline the learning process, we ensured consistency across all GUI elements, both within the software itself and in relation to commonly used software programs. This consistency allowed users to transfer knowledge and skills from one context to another, reducing the learning curve.

We introduced constraints by placing several options under the "Advanced Options" category, simplifying the overall interface by removing non-essential options. Additional constraints included fixed Drop-down menu options and a Spin-box prevention mechanism that allowed users to enter a numeric value between 0-100% for the prediction threshold.

Lastly, we addressed flexibility by providing two primary methods for interacting with the software: mouse and keyboard navigation or keyboard-only navigation. Users could click buttons and select check-boxes with a mouse, input file paths directly, or browse files by clicking through folders on their computer. Alternatively, they could navigate using the keyboard's tab, space, and arrow keys.

By incorporating these design principles from Don Norman's work, we successfully developed a software program characterized by a simple, intuitive, and user-friendly interface.

7.3 GUI-Elements

In order to efficiently allocate resources and streamline the design process for the GUI elements, our team opted to utilize the PyQt6 [14] framework. Employing PyQt6 facilitated the creation of a contemporary aesthetic with minimal exertion, while simultaneously enabling automatic adaptation of the GUI to the OS of the target device. This strategic decision allowed us to focus on other aspects of the software development process, ultimately contributing to a more effective and user-friendly application.

7.3.1 File Manager

During the initial phase of development, the file explorer employed was the inherent file explorer window associated with the OS. However, following user tests conducted with our clients, feedback revealed an inability to visualize the files contained within directories, rendering the directories ostensibly vacant. This deficiency generated uncertainty regarding the accuracy of the selected directory.

In response to this feedback, modifications were implemented in subsequent iterations. The final version incorporated a platform-independent file explorer, ensuring uniform appearance across various operating systems and displaying both files and directories within each folder. A depiction of the finalized file explorer is provided in Figure 7.3.1.

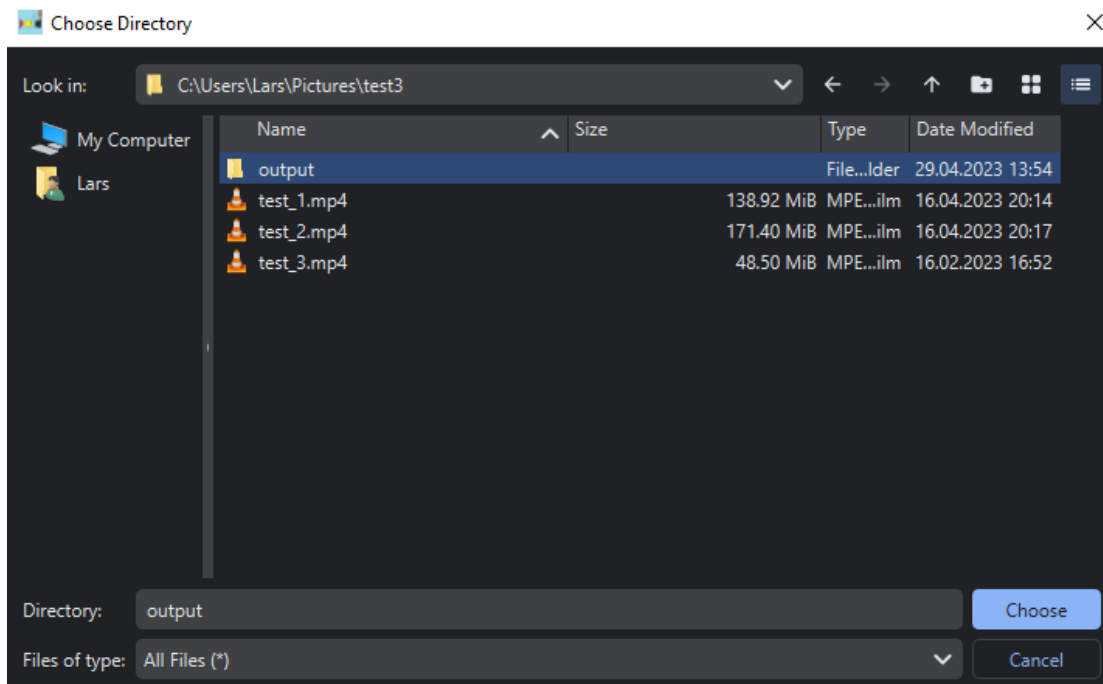


Figure 7.3.1: How the window where you chooses a directory looks when selecting either input or output directory.

7.3.2 Tool Tips

In order to improve the User Experience in our GUI, we implemented a tool tip feature to provide users with more detailed information on the available options. This feature was added due to the wide set of different options available for the user, some of which were not as self-explanatory as others. By providing tool tips, we aimed to reduce the likelihood of user errors and to inform the user on what changing the values will result in. As a result, we were able to include more advanced options such as Batch Size and Prediction threshold. An example of a tool tip added in our program can be seen in Figure 7.3.2.

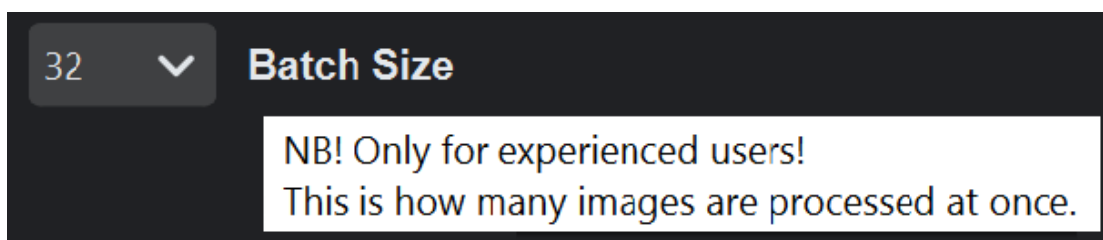
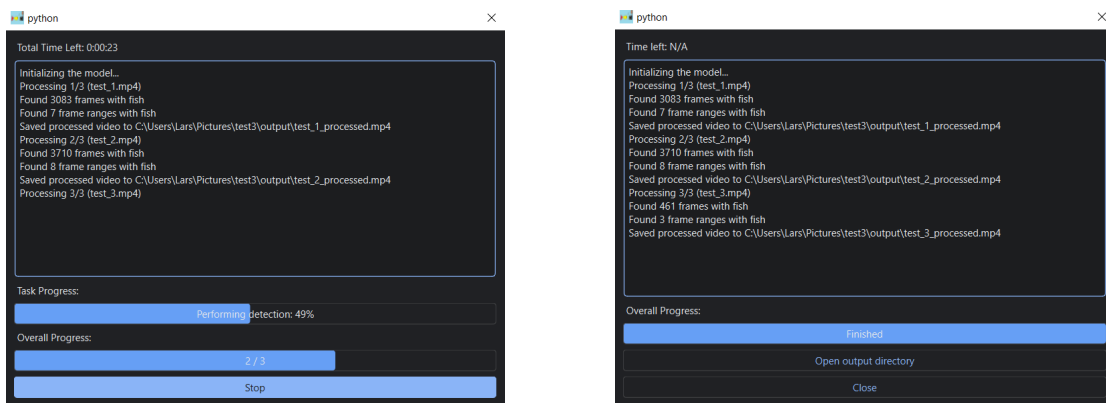


Figure 7.3.2: Tool Tip Example.

7.3.3 Progress Bar and Feedback Window

Given the potential for the application to process extensive data sets, encompassing hundreds of videos or more, it is of paramount importance to provide users with sufficient information throughout the procedure. Consequently, a custom progress and feedback window was developed to convey pertinent information in a manner tailored to user requirements. This approach diverged from reliance on preexisting OS windows, enabling the delivery of all necessary information to users as the program operates.

The customized window offers an array of crucial details, including the number of detected frames and frame-ranges of fish, the video currently undergoing processing, the number of remaining videos, an estimated time to completion, and additional pertinent data. Furnishing this information equips end users with valuable insights into the ongoing process, enhancing the ease of monitoring progress and obviating the need for excessive waiting periods. A visual representation of the progress bar and feedback window is exhibited in Figure 7.3.3.



(a) While performing detections.

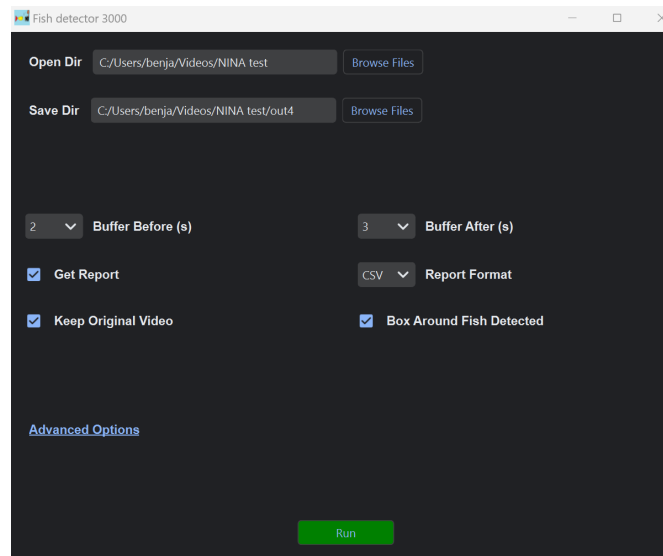
(b) When all detections and cutting of video is finished.

Figure 7.3.3: This Figure showcases the appearance of the progress bar and feedback window in the final iteration of our GUI. The design effectively presents pertinent information to the user during the processing phase, offering essential details to enhance their waiting experience. This includes an estimated time remaining, an indication of the currently processed video, the number of videos remaining, and comprehensive information regarding the detected objects within each video. Notably, once the processing is complete, the loading bar seamlessly transitions into an "Open Output Directory" button, providing convenient access to the newly generated files.

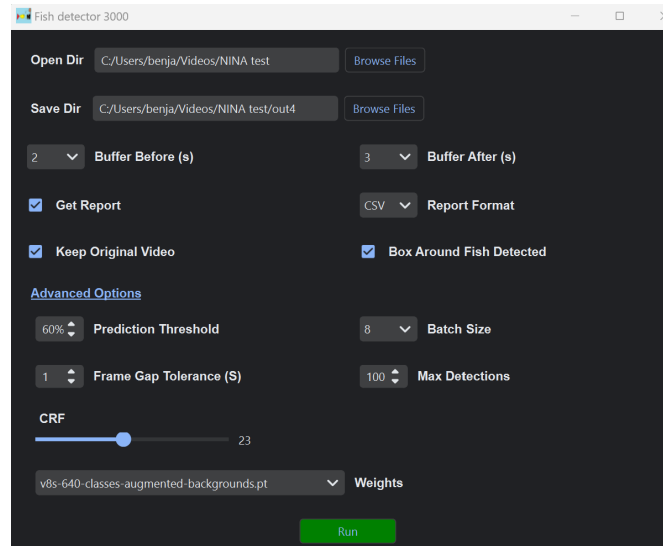
7.4 GUI-Evolution

Throughout the course of the bachelor project, the GUI has undergone significant evolution and improvement. Notable changes include the incorporation of intuitive and modern widgets, automatic switching between Light mode and Dark mode based on the OS settings, and a refinement of the overall appearance. Additionally, we responded to our client's request for more advanced options, including the

provision of report formats such as XLSX and CSV, the inclusion of detection boxes in the video, the option to modify Batch Size, and a Prediction threshold toggle. Moreover, we extensively revamped the loading window, enhancing it from a rudimentary Windows loading bar to a more comprehensive window that furnishes users with more information and program-specific options. Detailed comparative visualizations of these transformations are presented in Figures 7.1.1 and 7.4.1.



(a) Without Advanced Options Toggled.



(b) With Advanced Options Toggled.

Figure 7.4.1: This is the final version of how the GUI turned out. Here it shows the program when you open it up and after clicking the text to extend it to include the Advanced Options.

OBJECT-DETECTION

This chapter provides a comprehensive exploration of the implementation and optimization of the YOLOv8 object detection model in our video processing pipeline. We delve into the practicalities of assembling and processing a robust dataset, detailing the techniques used to gather and curate data suitable for training our model. The annotation process, which forms a crucial part of generating our training dataset, is discussed to provide insights into the intricacies involved in preparing high-quality ground truth data.

Furthermore, we unpack the training process, discussing the decisions made regarding model configuration training. A particular focus is given to the trade-offs we navigated regarding computational resources, training time, and the desired model performance.

In addition to outlining our methodology, we share our approach to evaluating the performance of our trained model, illustrating the metrics used for this purpose. This section includes a discussion on the limitations we encountered with respect to the amount of available ground truth data for validation.

Throughout this chapter, we also address the challenges and limitations we encountered during the implementation of our video processing pipeline. These include issues related to hardware and software constraints, data transfer challenges, as well as challenges posed by external interruptions during the training process.

By examining the various stages of our work with the YOLOv8 object detection algorithm, this chapter provides a holistic view of its practical application in a real-world scenario.

8.1 Dataset Creation and Utilization

In order to train our YOLOv8 model effectively, we required a substantial and diverse dataset. The process of building such a dataset involved both utilizing pre-existing data and generating new annotated data. A broad dataset ensures a more generalized model that can better handle real-world scenarios. It also allowed us to consider multiple classes of aquatic organisms, including fish and amphibians.

The final dataset used for training the model is a combination of data obtained from a previous group's work [41], illustrated in Figure 8.1.2, and data that we

personally collected and annotated. $\sim 66\%$ of the dataset was sourced from the prior group, who focused on identifying fish species. We managed to procure their dataset through email. The remainder of the dataset was created using videos that we annotated using CVAT. The distribution of annotations in the final dataset, which consists of 128,954 images, is illustrated in Figure 8.1.1.

Table 8.1.1: Combined Annotation Distribution

Name	# Annotations	Distribution
Gjedde	47,353	0.23
Gullbust	1,884	0.01
Rumpetroll	14,926	0.07
Stingsild	360	0.00
Ørekyt	45,809	0.22
Abbor	33,781	0.16
Brasme	285	0.00
Mort	41,282	0.20
Vederbuk	18,980	0.09
Frosk	120	0.00
Annen fisk	1,500	0.01
Total	206,280	-

Table 8.1.2: Annotation Distribution from Previous Group

Name	# Annotations	Distribution
Gjedde	46,173	0.34
Gullbust	1,884	0.01
Rumpetroll	10,335	0.08
Stingsild	360	0.00
Ørekyt	34,873	0.26
Abbor	15,713	0.12
Brasme	285	0.00
Mort	7,827	0.06
Vederbuk	18,980	0.14
Frosk	0	0.00
Annen fisk	0	0.00
Total	136,430	-

Note: The label "Annen fisk" represents fish that could not be identified during the annotation process. This label was used to group such instances in the dataset.

8.1.1 Gathering the Dataset

The dataset compilation process relied on two primary sources: the previously obtained group's dataset and our personally curated dataset. The dataset from the previous group was a YOLO-formatted dataset containing train.txt, val.txt,

images, and annotations. Additionally, we obtained two disks from NINA, which contained video data. The first disk held 41.2GB of video data from 56 videos, while the second disk comprised 3.47TB of video data from 1331 videos.

8.1.1.1 Identifying Annotated Videos Script

The challenge with incorporating the previous group’s dataset was that they had already annotated a substantial portion of the same video data. To avoid duplicating annotation work and wasting time, it was crucial for us to determine which videos had already been annotated by the previous group. To resolve this issue, we developed a script to identify what videos had already been annotated.

The script performed the following tasks:

1. Extracts images from each video in the dataset using the `extract_images_from_video()` function. This function utilizes OpenCV to read frames from the video and saves them as images on the disk.
2. Computes a perceptual hash (phash) using the `phash()` function for each extracted image. This hash function provides a unique representation of the image content, allowing us to compare images for similarity.
3. Loads the hashes of the images from the previous group’s dataset using the `load_hashes()` function. This function searches for images in the previous group’s dataset and computes their perceptual hashes. The hashes are then saved in a cache file for faster loading in future runs.
4. Compares the perceptual hashes of images from the videos to those from the previous group’s dataset. If a match is found, the script determines that the video had already been annotated by the previous group.
5. Writes the names and paths of the annotated videos in a text file for easy reference.
6. Maintains a cache of processed videos to avoid reprocessing the same videos in future runs.

We used this script to find a substantial amount of the videos that had been annotated by the previous group. While we don’t have precise data to confirm how many of the annotated videos we found, it significantly reduced the time and effort required for this task, allowing us to focus on annotating the remaining videos. It’s worth noting that the script’s results were manually reviewed afterward to ensure accuracy in regard to false positives. We kept track of all the annotated videos in a Google Sheet visualised in Figure 8.1.1.

The detailed implementation for this script can be found in the Appendix at I.

Annotation statistics		<input checked="" type="checkbox"/>			
Time	Videos tagged	Annotated videos	Target amount	Progress	
Day	14	11	20	55%	
Evening	3	3	20	15%	
Visuals	Videos tagged	Annotated Videos	Target amount	Progress	
Murky	6	4	20	20%	
Clear	10	9	20	45%	
Locations	Videos tagged	Annotated Videos	Target amount	Progress	
Myggbukta	10	7	10	70%	
Høyregga	6	6	10	60%	
Fish ladder	0	0	10	0%	
Video Specifics					
Video title	Folder-name	Location	Visuals	Time	Annotated
File1-[2020-05-26_09-26-43]-047	myggbukta 2020 mai/ 27.05.2020 - 28.05.2020	Myggbukta	Murky	Day	<input checked="" type="checkbox"/>
File1-[2020-05-26_09-26-43]-050	myggbukta 2020 mai/ 27.05.2020 - 28.05.2020	Myggbukta	Murky	Day	<input checked="" type="checkbox"/>
Myggbukta2022-[2022-10-11_06-28-47]-017	Myggbukta 2022	Myggbukta	Murky	Day	<input type="checkbox"/>
Myggbukta2022-[2022-05-20_10-36-34]-259	Myggbukta 2022	Myggbukta	Murky	Day	<input checked="" type="checkbox"/>
Myggbukta2022-[2022-05-20_10-36-34]-260	Myggbukta 2022	Myggbukta	Murky	Day	<input type="checkbox"/>
Myggbukta2022-[2022-10-11_06-28-47]-017	Myggbukta 2022	Myggbukta	Murky	Day	<input checked="" type="checkbox"/>
File1-[2017-08-18_11-46-59]-000		Høyregga	Clear	Day	<input checked="" type="checkbox"/>
File1-[2017-08-18_11-46-59]-001		Høyregga	Clear	Day	<input checked="" type="checkbox"/>
File1-[2017-08-18_11-46-59]-002		Høyregga	Clear	Day	<input checked="" type="checkbox"/>
File1-[2018-06-21_11-49-09]-069		Høyregga	Clear	Evening	<input checked="" type="checkbox"/>
File1-[2018-06-21_11-49-09]-070		Høyregga	Clear	Evening	<input checked="" type="checkbox"/>
File1-[2018-06-21_11-49-09]-099		Høyregga	Clear	Day	<input checked="" type="checkbox"/>
MYGGBUKTA2022-[2022-05-20_10-36-34]-110	Myggbukta 2022	Myggbukta	Clear	Day	<input checked="" type="checkbox"/>
MYGGBUKTA2022-[2022-05-20_10-36-34]-111	Myggbukta 2022	Myggbukta	Clear	Day	<input checked="" type="checkbox"/>
MYGGBUKTA2022-[2022-05-20_10-36-34]-112	Myggbukta 2022	Myggbukta	Clear	Day	<input checked="" type="checkbox"/>
MYGGBUKTA2022-[2022-08-13_05-29-48]-025	Myggbukta 2022	Myggbukta	Clear	Day	<input type="checkbox"/>
MYGGBUKTA2022-[2022-06-16_15-41-38]-300	Myggbukta 2022	Myggbukta	Clear	Evening	<input checked="" type="checkbox"/>

Figure 8.1.1: Google Sheet for Tracking Annotated Videos

8.1.2 Annotation Rules

To maintain consistency and accuracy in our dataset, we established a set of annotation rules to be followed during the annotation process. These rules were similar to the ones followed by the previous group, as described in their thesis [41]. The primary rules we adhered to were as follows:

- A fish should be annotated if at least 50% of its body was visible within the frame. This rule helped to ensure that we only annotated fish with enough visible features for the model to learn from.
- We would only annotate fish that were clearly identifiable within a single frame. Since the algorithm did not have access to temporal data or context, we refrained from annotating fish that could only be recognized by examining preceding or following frames where they were more visible. By annotating only the fish that are clearly identifiable within a single frame, we ensure that the model learns to detect fish based on their distinct features in isolation, promoting better generalization and reducing the likelihood of false positives.
- If we started annotating a video, we were required to complete the entire video. If it was not possible to finish annotating the entire video, we needed to cut the video up to the last annotated frame and re-upload it to CVAT with the existing annotations. This was necessary because the dataset generation script (subsection 8.1.3) would otherwise include un-annotated frames as background images when that feature was still in use.

These rules were established to ensure consistency with the annotations provided by the previous group, as $\sim 66\%$ of the dataset was provided by them. By following similar annotation rules, we aimed to maintain consistency and compatibility between the two halves of the dataset.

8.1.3 Dataset Generation Script

To facilitate the creation and management of our dataset, we developed a custom Python script that parsed the "CVAT for video 1.1" XML format annotations and generated a YOLO-formatted dataset. This script was designed to address issues we encountered when using Datamaro [42] for merging datasets, such as duplicate entries and image name conflicts.

To create the dataset, we first manually exported each annotated video from CVAT using the "CVAT for video 1.1" format. This process produced a ZIP file for each video, which we then placed together in a folder. Our custom script subsequently performed the following tasks on the exported files:

1. Extracted and parsed the CVAT XML annotation file from each ZIP file, corresponding to an annotated video.
2. Fetched the source video name from the annotation file and located the video on the two disks obtained from NINA.
3. Extracted the annotated frames from the source video and saved them to disk, organizing them into separate folders for each video to ensure unique paths and easy navigation.
4. Generated the annotation .txt files for each image, following the YOLO dataset format specifications [43].
5. Automatically generated background frames using the `determine_background_frames()` function for initial testing, aiming for a 10% background frame ratio to avoid false positives. Later, we manually gathered background frames from various videos.
6. Generated `train.txt` and `val.txt` files based on the training split, containing a list of paths to images in the dataset.
7. Created a YAML file with paths to `train.txt`, `val.txt`, the number of classes (`nc`), and the names of the objects/labels.

The script also facilitated the merging of additional datasets. When a new dataset containing images and annotations was added to the output dataset folder, running the `split_train_val()` function again would include those images in the dataset.

If CVAT had been able to export frames with annotation data directly, we would not need to use the script to search for and manually extract the matching frames. We could have simply executed the `split_train_val()` function after placing the exported dataset into the output dataset folder.

The detailed implementation of the dataset generation script can be seen in Appendix I.

8.1.4 Distribution of Annotations per Class Script

To maintain a balanced dataset and prevent model bias towards overrepresented fish species, we created a Python script to calculate the distribution of annotations per class in our dataset. This script provided us with insights into the annotation count for each fish species, helping us identify underrepresented classes that required additional images. The script performed the following tasks:

1. Loaded the class names from the `obj.names` file using the `load_names()` function. This function read the file and returned a list of class names.
2. Counted the number of annotations for each class using the `get_annotation_distribution()` function. This function iterated through the YOLO-formatted dataset's subdirectories, locating and opening each annotation `.txt` file. It then read the class labels from the files and counted the occurrences of each class label in the dataset.
3. Generated a list of tuples containing class names and their respective annotation counts, obtained from the `get_annotation_distribution()` function.
4. Exported the annotation distribution data to a CSV file using the `write_annotation_distribution_to_csv()` function. This function wrote the class names and their corresponding annotation counts to the CSV file, which we then added to our Google Sheet for easy tracking and updating. This can be visualized in Figure 8.1.2.

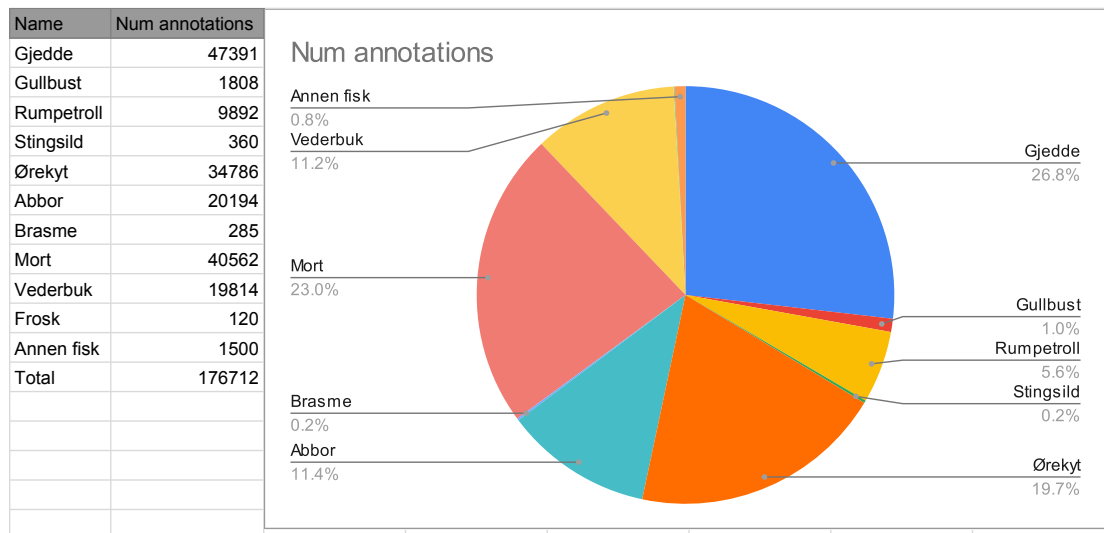


Figure 8.1.2: Table with class names and number of annotations and a corresponding graph showing the distribution of annotations per class

By using this script, we could quickly identify species with fewer annotations and seek to improve the balance of the dataset. Although this didn't result in a perfectly balanced dataset, the steps we took included searching for more images of underrepresented species from the dataset and prioritizing the annotation of videos containing those species. This process helped us to incrementally enhance the representation of various fish species in our dataset. The script also enabled

us to monitor the progress of our annotation work and promote a better, though not perfectly balanced, dataset for training the YOLOv8 model.

The detailed implementation for the script can be found at Appendix I.

8.2 Model Training and Optimization

This section delves into the intricacies of training our object detection model and optimizing its performance. We discuss the training setup, dataset splits, class definitions, and the rationale behind our training choices. Additionally, we explore the adjustments made to hyperparameters and data augmentations to fine-tune the model for underwater fish detection. The section also provides insights into the resource considerations involved in deep learning model training and highlights the evaluation metrics employed to assess the model’s performance. With explanations and figures illustrating the training results, this section provides a comprehensive overview of our approach to training and optimizing the object detection model.

8.2.1 Training Setup

To facilitate an efficient workflow, the YOLOv8 model was trained using two distinct computer systems. The personal computer, boasting superior GPU capabilities and more VRAM, was employed for part of the training and experimentation tasks. A comprehensive outline of this computer’s hardware specifications can be found in Table 8.2.1. Although the school-allocated computer was equipped with dual GPUs, PyTorch’s limitations with distributed training on Windows confined us to utilizing only one of them. A detailed overview of the school-allocated machine’s specifications is presented in Table 8.2.2.

The two systems’ combined computational resources, along with their parallel experimentation and hyperparameter evaluation capabilities, facilitated a rapid optimization of the model. This approach effectively curtailed the overall model training time, despite the hardware and software limitations inherent to the school’s computer.

A consistent Python environment was maintained on both computers during training. This environment, defined in the ‘pyproject.toml’ file at the root of the project, was based on Python 3.10 and PyTorch version 1.13.1. To ensure compatibility, all the necessary libraries and tools for training were installed on each machine. This setup, despite its advantages, was not without challenges and limitations, which are elaborated on in Section 8.4.

Table 8.2.1: Hardware Specifications of Personal Computer

Component	Specification
CPU	AMD Ryzen 9 5950x 16-Core Processor
GPU	NVIDIA Geforce RTX 3090
RAM	2x16GB G.Skill F4-3600C16
Disk	WD Black SN750 NVMe SSD

Table 8.2.2: Hardware Specifications of the School-Allocated Computer

Component	Specification
CPU	AMD Ryzen Threadripper 2950X 16-Core Processor
GPU	2x NVIDIA GeForce RTX 2080 Ti
RAM	32.0 GB
Disk	2TB NVMe TOSHIBA 2048GB

8.2.2 Data Splits and Class Definitions

We adopted a 70/30 random split for the training and validation data. The training utilized eleven different species and categories of fish, which are visualized in Figure 8.1.1.

8.2.3 Training Details and Rationale

Several models were trained using varied hyperparameters and training strategies to optimize the detection system’s performance and efficiency. While more models were trained, two models named "v8s-640-classes-augmented-backgrounds" and "v8m-640-classes-augmented" were chosen as the final models based on their balance between performance, efficiency, and computational demands.

8.2.3.1 v8s-640-classes-augmented-backgrounds

This model utilized manually gathered background images, which provided several advantages over the automatically generated ones discussed earlier (see Section 8.1.3). The manually gathered background images offered greater diversity and higher quality compared to the automatically generated backgrounds. By manually selecting and curating background images from various sources, we were able to include a wider range of backgrounds with different textures, lighting conditions, and compositions. This diversity in backgrounds helps improve the model’s ability to generalize to different environments, including those dense with seagrass. Additionally, the manual gathering process allowed us to cherry-pick high-quality background images, ensuring that the model was exposed to more representative backgrounds during training.

8.2.3.2 v8m-640-classes-augmented

The second model was trained using the medium-sized variant of the YOLOv8 model (YOLOv8m). This model explored the trade-off between model complexity, performance, and computational demands. Owing to the graphics card’s VRAM limitations, the batch size was restricted to 8, a constraint that can potentially affect the model’s generalization performance [44]. It used the same dataset as "v8s-640-classes-augmented-backgrounds" to train.

Both models were trained using a resolution of 640x640 pixels, a strategic choice to balance computational efficiency and model performance. To assess the impact of resolution on inference speed, we conducted experiments at 1280x1280, twice the initial resolution. The results, which were generated on the personal

computer and are illustrated in Table 8.2.1, revealed that the original configuration was approximately $2.4 \times$ faster.

8.2.4 Model Augmentations and Hyperparameters

In order to fine-tune our model for fish detection in underwater environments, we adjusted several hyperparameters and augmentations. This section provides an overview of these adjustments and their impacts.

8.2.4.1 Hyperparameters

Hyperparameters are vital knobs that govern the learning process of a neural network model. They are typically set before training begins and remain constant throughout the learning procedure. Fine-tuning these hyperparameters can significantly enhance the model’s performance.

Here, we made adjustments to several key parameters to manage the speed and quality of learning. The initial learning rate ($lr0$), final learning rate (lrf), and weight decay were tuned to control the learning pace and regularize the model, respectively. The default and final configurations for these parameters are provided in Table 8.2.3.

Table 8.2.3: Final and Default Configurations for Hyperparameters

Hyperparameter	Default Value	Final Value
lr0 (initial learning rate)	0.01	0.001
lrf (final learning rate)	0.01	0.01
weight decay	0.0005	0.001

Note that while other hyperparameters were available for adjustment, they were left at their default settings for this thesis. The default configurations for all parameters can be found in the official documentation [45].

8.2.4.2 Data Augmentations

Data augmentations, which modify training images, are a powerful technique to boost the performance of deep learning models. In the context of underwater fish detection, they can assist the model in learning to recognize fish under a variety of different conditions and perspectives.

Our final models were trained with various augmentations, including alterations to hue (hsv_h), saturation (hsv_s), and brightness (hsv_v), which helps the model account for variations in water clarity and lighting conditions, as well as the diverse colorations of different fish species.

Other implemented augmentations include rotation ($degrees$), translation ($translate$), scaling ($scale$), and vertical flipping ($flipud$). The decision to include these augmentations was significantly driven by the nature of our dataset, which consisted of images captured from a limited number of static camera angles. These augmentations introduce variations that mimic different perspectives and orientations, effectively expanding the diversity of our dataset. By doing so, they enable the model to learn robust features that remain consistent regardless of the fish’s

orientation, position, size, or reflection in the water. The final and default values for these augmentations are provided in Table 8.2.4.

While the default settings for the above augmentations provided a solid starting point, we adjusted several parameters to better suit our task. For instance, we increased the degrees of rotation and scale range to account for the diverse poses and sizes of fish. Similarly, the augmentation range for hue, saturation, and brightness was boosted to cope with the wide spectrum of colors and lighting conditions prevalent in underwater environments. The probability of a vertical flip was also slightly increased, considering the possibility of upside-down fish appearances in the training data.

Table 8.2.4 summarizes the final and default configurations for the augmentations.

Table 8.2.4: Final and Default Configurations for Model Augmentations

Parameter	Default Value	Final Value
hsv_h	0.015	0.2
hsv_s	0.7	0.7
hsv_v	0.4	0.7
degrees	0.0	10
translate	0.1	0.1
scale	0.5	0.6
flipud	0.0	0.01

In addition to these augmentations, the training process also incorporated the mosaic data augmentation technique. Mosaic augmentation combines four training images into one, increasing intra-image variance to mimic a broader data distribution and promote the robustness of the trained model. However, to avoid potential overfitting risks associated with this technique, mosaic augmentation was disabled for the last 10 epochs of training. This practice is aligned with advice provided by the developers of the YOLOv8 model [46]. An example batch of training data is illustrated in Figure 8.2.1.

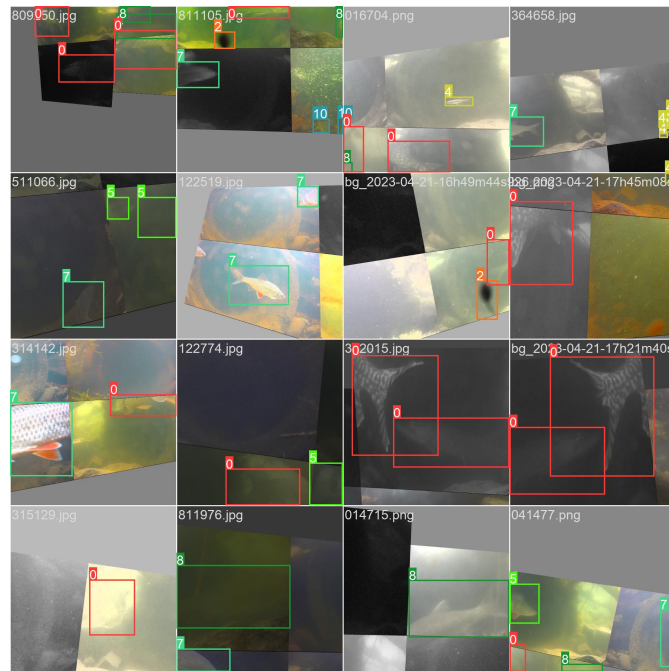


Figure 8.2.1: Example training batch with augmentations and mosaic.

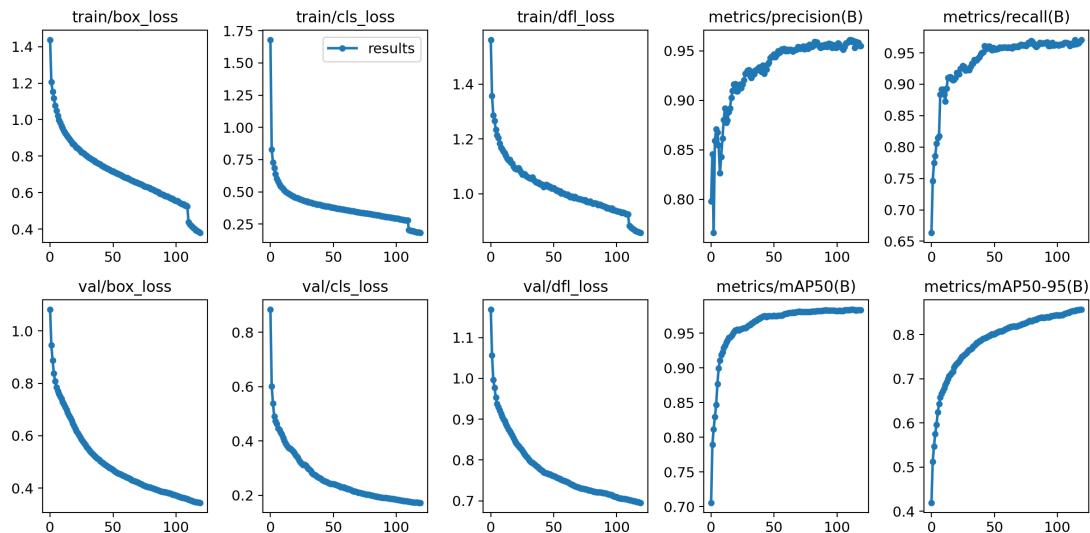
8.2.5 Training Resource Considerations

The training process of deep learning models in this project underscored the resource-intensive nature of the task. Notably, the small model necessitated approximately 49 hours to complete 100 epochs on the school's computer, while the medium model, being larger with, required approximately 74 hours for the same number of epochs. These extensive time requirements highlight the substantial computational costs associated with training deep learning models.

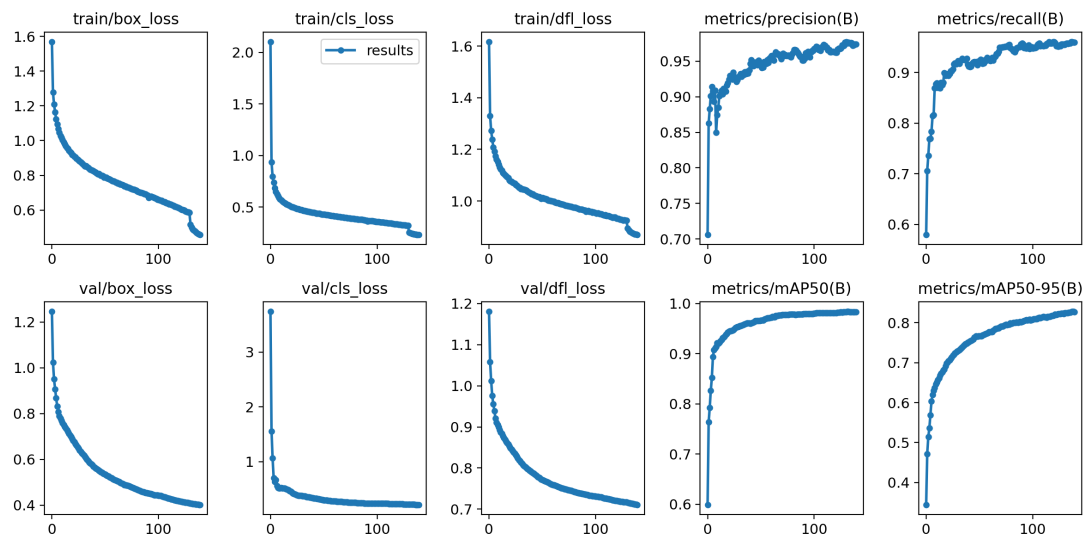
8.2.6 Training and Evaluation Metrics

The models were trained using Stochastic Gradient Descent (SGD), a popular optimization algorithm widely utilized in machine learning and deep learning for its efficiency and effectiveness. SGD is advantageous as it retains the comprehensive approach of regular gradient descent while significantly reducing computational demand by performing parameter updates using a single training sample at each iteration.

The training process relied on a variety of metrics for model evaluation, including F1 score, precision, recall, and mAP (mean Average Precision). These metrics, along with the associated losses: Box Loss, Classification Loss (Cls Loss), and Distribution Focal Loss (DFL Loss), during the training phase, are shown for each model in Figure 8.2.2.



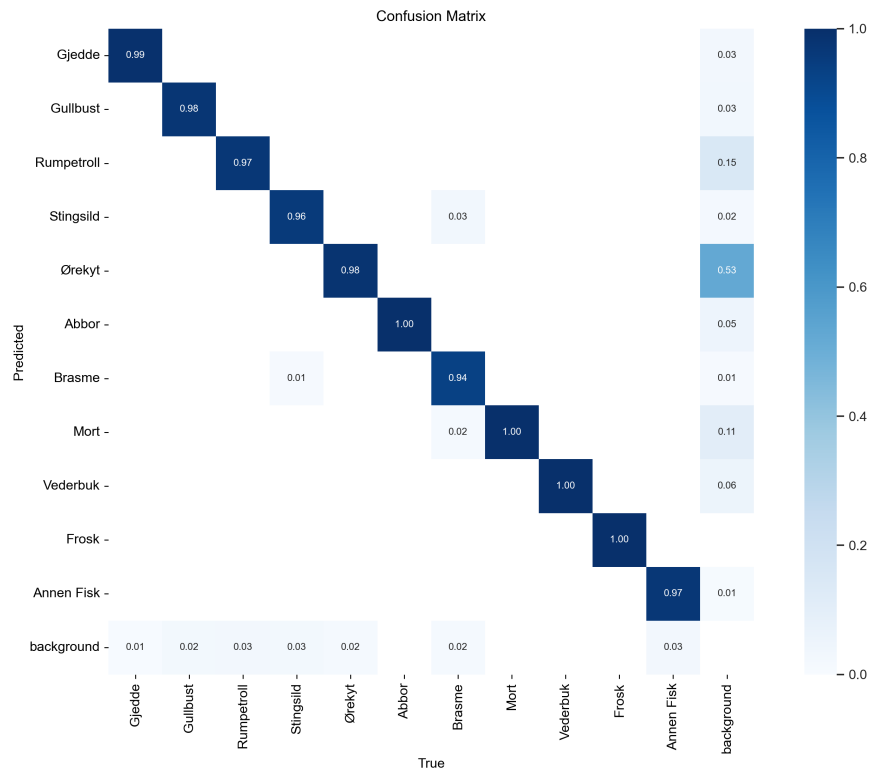
(a) v8m-640-classes-augmented.



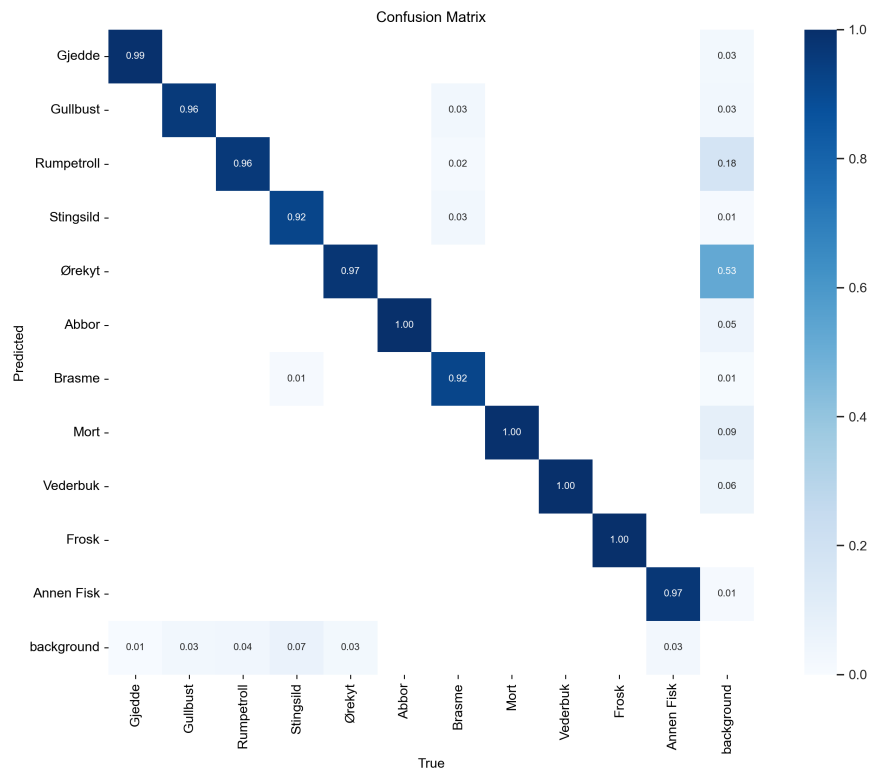
(b) v8s-640-classes-augmented-backgrounds.

Figure 8.2.2: Performance metrics over epochs for both models. The graphs provide an overview of the training progression, highlighting the evolution of critical metrics such as precision, recall, and mAP.

The confusion matrices for the "v8m-640-classes-augmented" and "v8s-640-classes-augmented-backgrounds" models offer an additional perspective on the models' performance. Figure 8.2.3 provides a visual summary of how accurately the models are classifying each category.



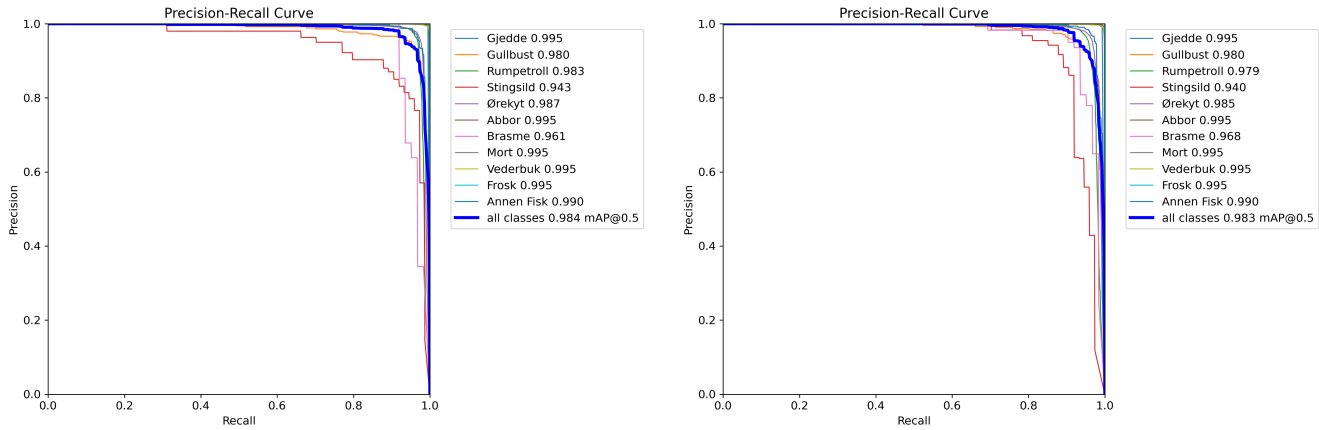
(a) v8m-640-classes-augmented.



(b) v8s-640-classes-augmented-backgrounds.

Figure 8.2.3: Confusion Matrix for Models. Each matrix demonstrates how well the respective model is able to accurately classify each category, by showing the proportion of true and false positives for each category. It is worth noting that a significant number of incorrect predictions are classified as backgrounds and vice versa, indicating a potential challenge in distinguishing between these categories.

To further delve into the performance of the models, Figure 8.2.4 displays the precision-recall curve for both models. This graph is a valuable tool for understanding the trade-off between precision and recall for different threshold settings and overall model performance.



(a) v8m-640-classes-augmented.

(b) v8s-640-classes-augmented-backgrounds.

Figure 8.2.4: Precision-recall curve for both models. The curves provide a graphical overview of the trade-off between precision and recall at various threshold settings, offering additional insights into the models' performance characteristics.

In conclusion, these figures provide a comprehensive and detailed overview of the model training process and subsequent performance. By examining these visual representations, we can gain valuable insights into model behavior, track learning progress, and identify potential areas for improvement.

8.3 Performance Evaluation

This section provides an overview of the evaluation of the weights using different models. It begins by discussing the importance of model evaluation and the various metrics used to measure model performance. Then we examine the different types of models used for evaluation and their respective techniques. Finally, it provides an overview of the challenges associated with model evaluation and potential solutions to these challenges.

8.3.1 Validation Metrics

We measured using the Intersection over Union (IoU) metric because of its use for model validation. We use it in order to get a generalised idea of how accurately the model predicts the ranges with fish.

We use IoU as a metric used for evaluating how accurately the model creates predictions as ranges with "fish". The metric quantifies the accuracy of the predictions by measuring the overlap between the predicted frame ranges and the ground truth frame ranges. Calculating the IoU is done by dividing the area of overlap between the predicted and ground truth bounding ranges by the area of their union.

$$IoU = \frac{AreaofOverlap}{AreaofUnion}$$

This ratio provides a measure of how well the prediction's bounding boxes aligns with the ground truth's box. The value of IoU ranges from 0 to 1, where a higher value indicates a better alignment. This is important because it allows us to compare different models and determine which one is more accurate in its predictions.

One advantage of IoU is that it is not affected by the absolute size of the ranges, however it focuses on the relative overlap of the intersected ranges, making it a useful metric for evaluating the different weights of our model on different environments based on the metric over the predicted confidence.

The videos used for evaluation were selected for evaluation based on: Murkiness, time of day, resolution, filters, and if they fit our predefined conditions for being valid videos for evaluating our model with.

8.3.2 Evaluation Videos

The videos used for evaluating the predictions of our AI model have to fit certain predefined conditions that we have set out, these conditions consist of:

- Conforming to our annotation rules. See Subsection 8.1.2. In order for us to accurately evaluate if the model has predicted the ranges with fish, we have to have footage that our annotation rules would fit, if the footage was to be annotated.
- Be within reasonable state of clarity to murkiness/noise. See Figure 8.3.1 which represents reasonable and valid camera conditions. This is to ensure that the footage we use is reasonable enough to use in order to shorten and get the ranges with fish.
- The video has to contain fish with reasonable visibility. This is important because using footage that contains no visible fish without data manipulation, where it is hard for even us humans to see fish, or if the footage has a lot of noise in addition to being low resolution, where bubbles become indistinguishable to small fish, is very unreasonable.

We will later provide further references to these videos, which conform to the aforementioned conditions as valid videos, and vice versa with the invalid non-conforming videos.

When evaluating our model using binary evaluation, we need to first establish a range of frames for analysis, this being the aforementioned ground truth ranges, typically within a video sequence. Which will be used to compare to the prediction ranges our model makes.

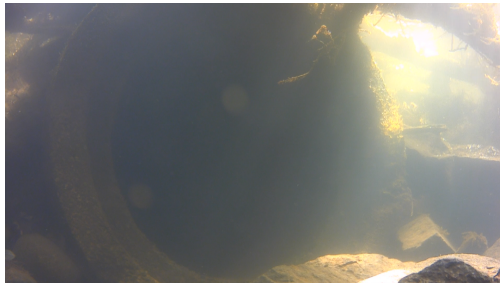
To perform the binary evaluation, we calculate the IoU between each frame predicted range and the corresponding ground truth frame range. If the IoU value exceeds the predefined threshold, typically set at 0.5, the detection is considered correct; otherwise, it is considered incorrect.

Once we have evaluated all the predictions in the frame range, we can compute various metrics such as precision and recall.

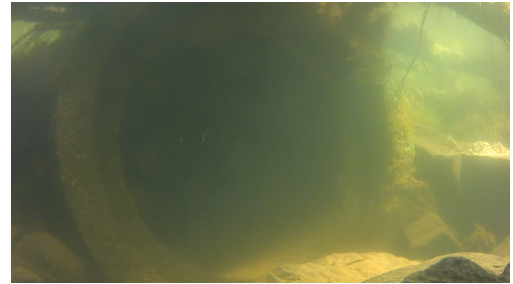
$$\begin{aligned}
 Precision &= \frac{TruePositives}{TruePositives + FalsePositives} \\
 Recall &= \frac{TruePositives}{TruePositives + FalseNegatives}
 \end{aligned}
 \tag{8.1}$$

Precision measures the proportion of correct detection among all the predicted ranges, while recall measures the proportion of correct ranges among all the annotated ground truth ranges.

Videos within our predefined conditions, valid videos



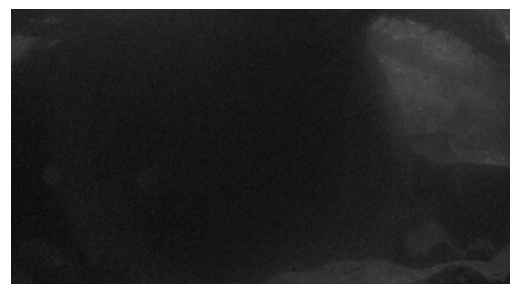
(a) Clear camera, clear water, daytime, colour



(b) Clear camera, cloudy water, daytime, colour



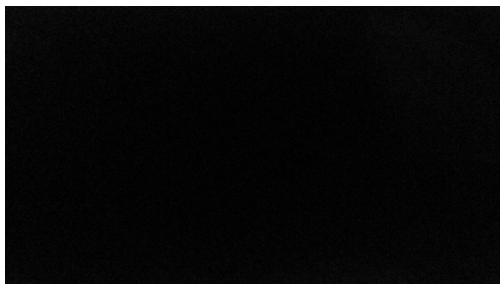
(c) Murky camera, cloudy water, daytime, colour



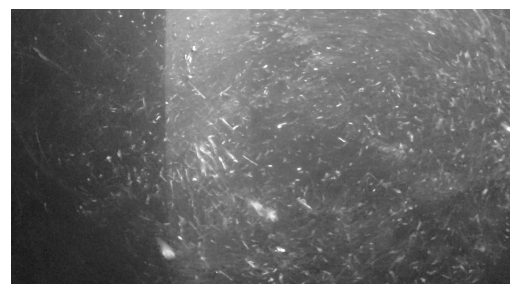
(d) Noisy camera, cloudy water, nighttime, black and white

Figure 8.3.1: This figure represents all of the right camera conditions used for validation for the model. The conditions are within the scope of our set annotation rules as seen in Section 8.1.2.

Videos outside of our predefined conditions, invalid videos



(a) Unclear camera, unclear water, nighttime, black and white



(b) Murky camera, noisy water, daytime, black and white

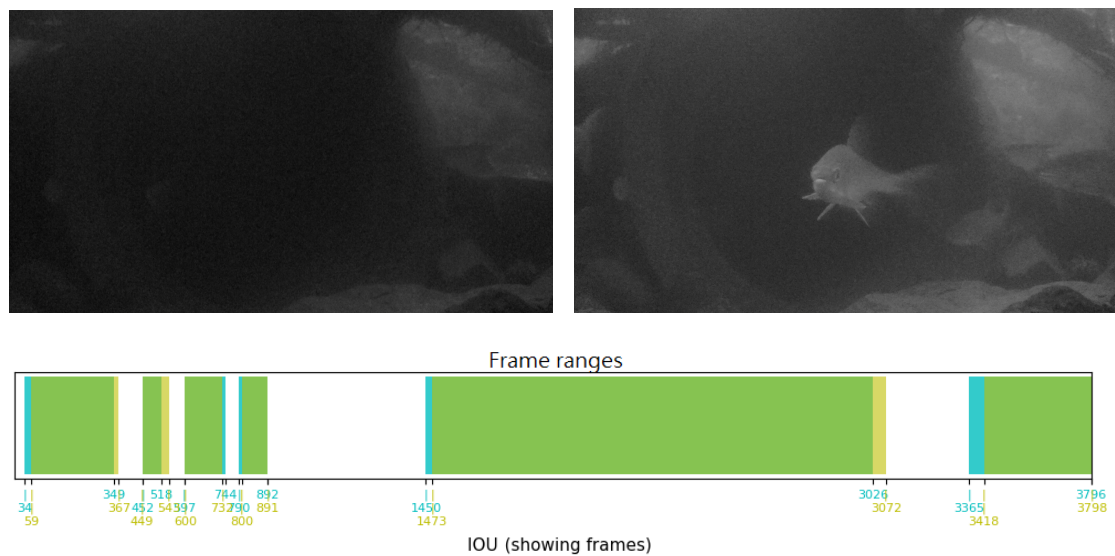
Figure 8.3.2: This figure represents all of the camera conditions not suitable for the validation of the model. The conditions are not within the scope of our set annotation rules as seen in Section 8.1.2.

We will eventually evaluate both of the conditions in order to better the model itself. This includes how the weights we have trained, used and tested will accurately detect frame ranges within reasonable conditions.

8.3.3 Fish Ranges of Videos

The Intersection over Union (IoU) Here we have visualised the IoU of each video using the *v8s-640-classes-augmented-backgrounds* weights for our model. This is where cyan represents the ground truth frame ranges, and yellow represents the predicted frame ranges from our model. Green is the overlap between the two.

IoU of valid videos



(c) This is the best example, with a confidence of 0.7 the average recall is at 0.964

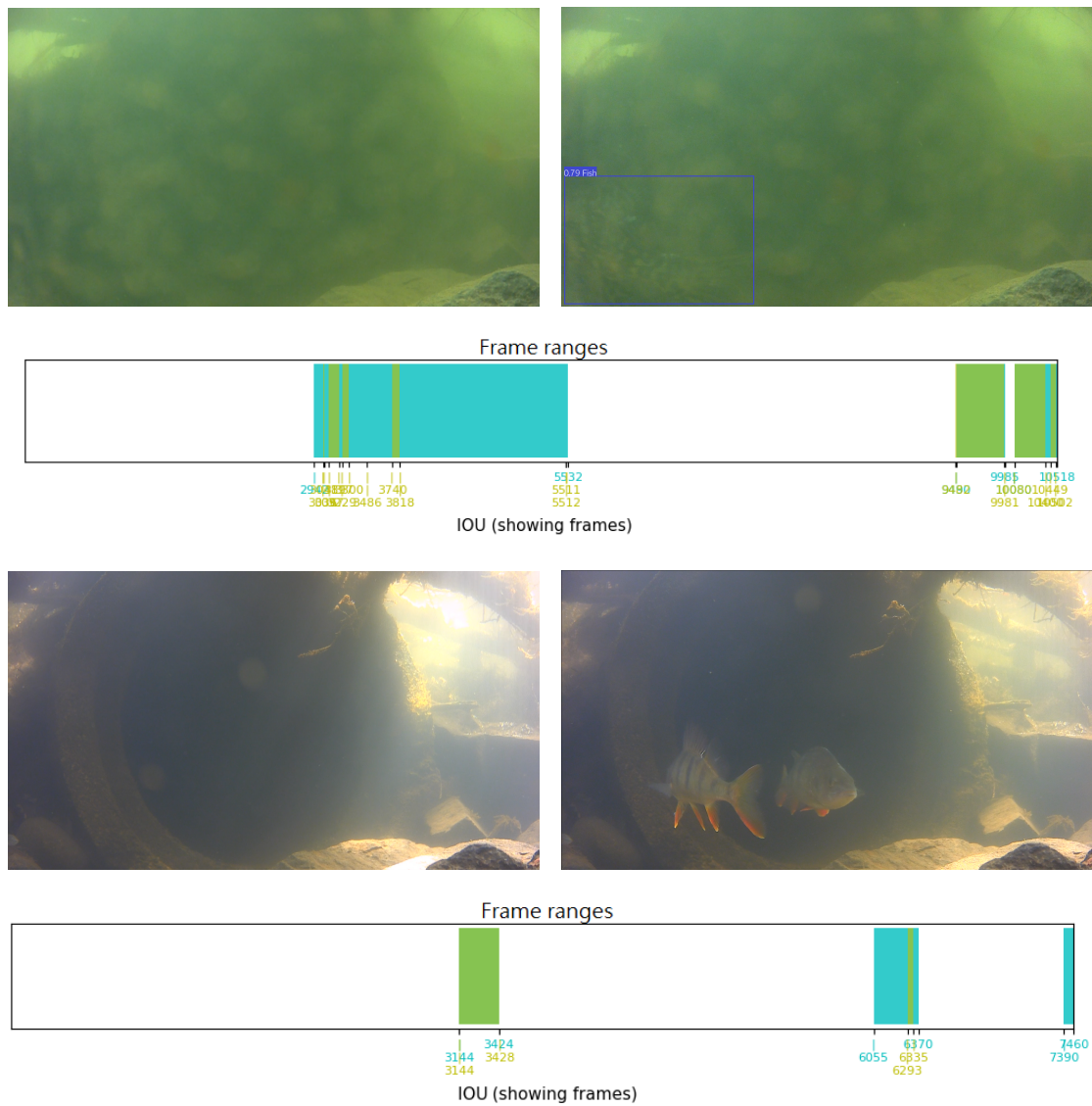


Figure 8.3.3: This figure shows the IoU for every valid video used during evaluation.

As Figure 8.3.3 above illustrates the different conditions we have outlined, you can see that the model is able to predict the fish in the ground truth. The green as aforementioned represents the IoU, where you can tell that the average IoU here is a high value.

IoU of invalid videos

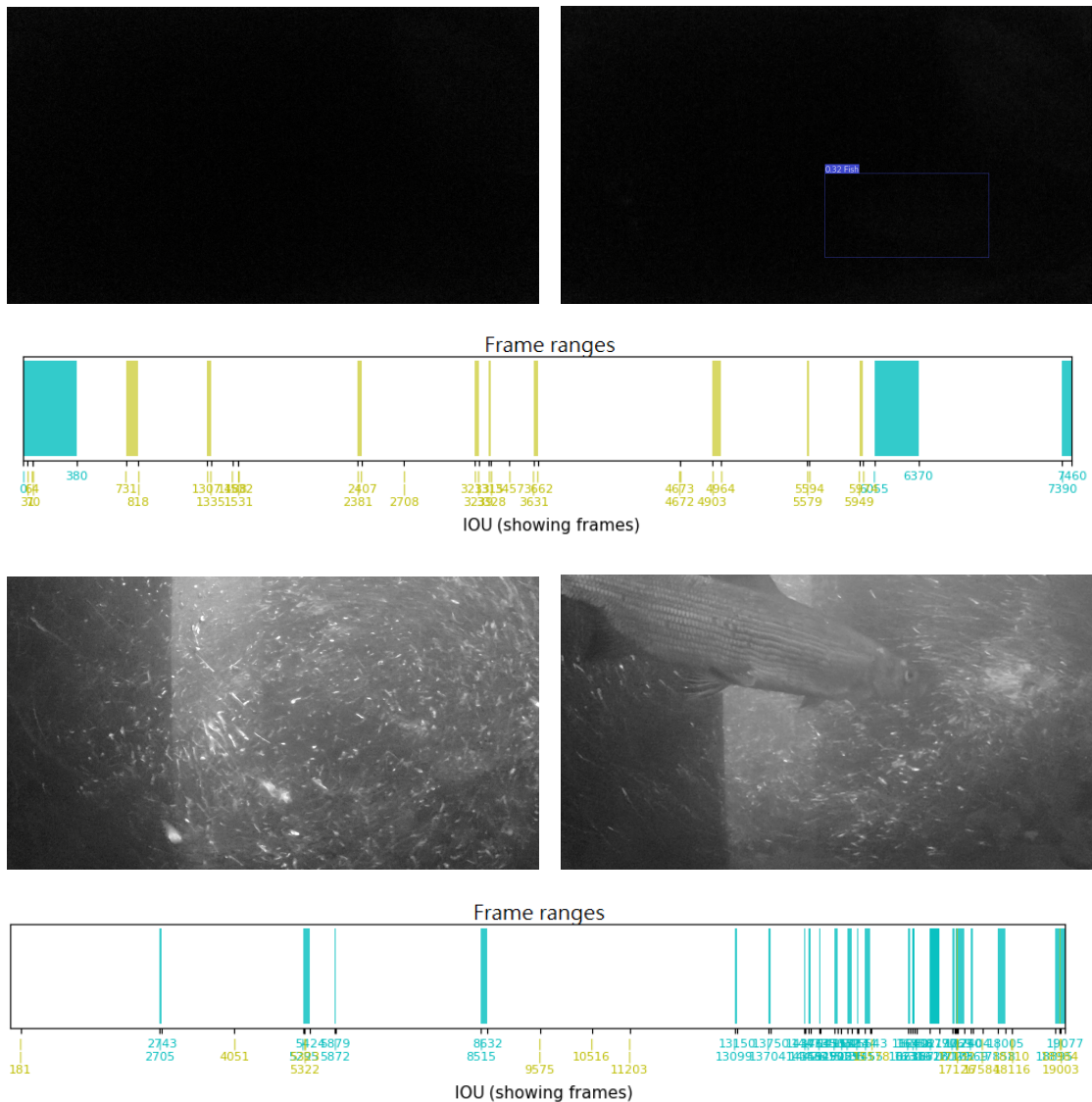
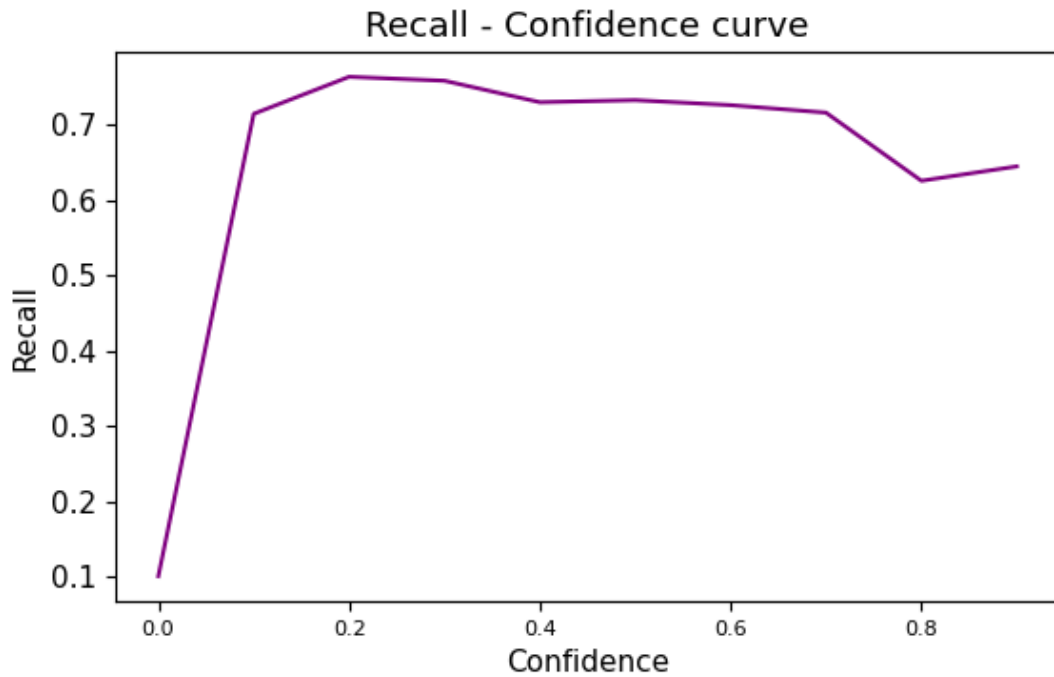


Figure 8.3.4: This figure shows the IoU for every invalid video.

As Figure 8.3.4 represents the different conditions we have mentioned as invalid, you can see that the model is unpredictable where it is erratic and gives seemingly random predictions. The green as aforementioned represents the IoU, here you can already tell that the average IoU is low.

8.3.4 Average Recall over Confidence

Here we calculated the average recall of all of the four valid control videos, and the same with three invalid videos, over a confidence range from 0 to 1. This is done utilising the aforementioned formula for recall, which is averaged and taken off every range of every video evaluated and plotted into the following figures:

Evaluation of videos within our predefined conditions, valid videos

(a) Average recall to the confidence of all the valid videos is 0.65

Evaluation of videos outside of our predefined conditions, invalid videos

(b) Average recall to the confidence of all the invalid videos is 0.26

Figure 8.3.5: This figure shows the average recall over confidence of both the valid and invalid videos used during evaluation.

To summarize, we calculated the average recall for both the valid and invalid videos within a confidence range of 0 to 1. This was done using the formula for recall and averaged across all videos and confidence ranges, where the results were plotted in Figure, 8.3.5.

In order to validate our models recall over confidence we got the number for the average recall of all the valid videos used for evaluation. The average recall reached 0.65 which indicates that our system successfully identified a significant portion of the ground truth ranges in these videos.

In contrast, the average recall for the invalid videos was considerably lower, measuring at 0.26. This suggests that our system struggled to accurately detect the ground truth ranges in videos that fell outside our predefined conditions.

These findings provide insights into the performance of our system in different scenarios. By analyzing the recall over confidence, we can assess the system's ability to detect objects of interest accurately. These results may guide further improvements to enhance the system's performance and make it more robust across various video conditions.

8.4 Challenges and Limitations

Throughout the development process, we encountered various challenges and limitations that impacted our object detection development process. In this section, we outline the issues we faced during training and the limitations we encountered in the performance evaluation.

8.4.1 Issues During Training

During the model training phase, we encountered several challenges and limitations that significantly impacted our process. These issues, and the ways we navigated them, are elaborated below.

8.4.1.1 Hardware Limitations

Although the personal computer boasted superior hardware, it was primarily used for short experiments due to substantial heat and noise generation during extended training sessions. Consequently, the school's computer, despite being less powerful, became our primary training device as it was more suitable for long-duration tasks.

8.4.1.2 Data Transfer Issues

Transferring our large training dataset between the personal computer and the school's machine presented a substantial challenge. The dataset comprised 263,767 files, totaling 113GB, making the transfer process time-consuming. To address this, we adopted a distinct folder structure for the dataset, as explained in Section 8.1.3. This structure facilitated more targeted data transfers. We could segregate each video's corresponding images and annotations into separate folders. Therefore, we only needed to transfer new or altered data, usually consisting of a specific video's images and annotations, between the machines. After the transfer, we could locally re-run the dataset generation script on the personal computer

to create updated 'train.txt' and 'val.txt' files, which were then transferred to the school's machine, allowing the training process to resume with the updated dataset.

8.4.1.3 Software Limitations

We encountered software limitations on the school's computer, which operates on Windows OS. Notably, we faced issues with the YOLOv8's v8 dataloader, as reported in a couple of GitHub issues [47, 48]. As a workaround, we used the older v5 dataloader designed for YOLOv5, which proved more reliable.

8.4.1.4 External Interruptions

External interruptions posed a further challenge in our training process. The school-allocated computer would sometimes restart for updates in the middle of a training session, resulting in progress loss. The shared nature of the device meant that other students could inadvertently disrupt our ongoing training. To mitigate these issues we tried leaving notes on screen to notify others that training was in progress.

8.4.1.5 Model and Resolution Limitations

Initially, we contemplated training a medium-sized model on a higher resolution, hypothesizing that it might yield better results, particularly in detecting smaller fish. As highlighted in Section 8.2.5, training the medium model took 74 hours, compared to the 49 hours required for the smaller model. Additionally, as stated in Section 8.2.3.2, inference using 1280 vs 640 took approximately 2.4 times longer. From these observations, we estimated that it would take roughly 7.4 days to complete a single training run to 100 epochs with the medium model at a higher resolution, assuming no interruptions occurred on the school-allocated computer. Given our time constraints, this was not a feasible option. As such, our current medium-sized model did not demonstrate significantly better results compared to models trained with the current resolution. In future work, exploring the impact of different model sizes and resolutions on model performance could provide further opportunities for improvement. This is also something NINA can do with our provided training guide found in Appendix H.

8.4.2 Performance Evaluation Limitations

Despite our best efforts in carrying out the performance evaluation, we faced certain limitations in terms of available data. The amount of ground truth data that we had available for evaluation was a significant limitation.

Due to the complexities involved in the manual annotation process and the need to balance this task with other project priorities, we were only able to generate ground truth data for seven videos, four of which adhered to our annotation rules and were deemed valid for our purposes, the remaining three were deemed invalid. The ground truth annotation process is a meticulous one, requiring substantial time and human effort to ensure accuracy, which limits the volume of data that can be produced within a given time frame.

While this may appear to restrict the comprehensiveness of our model evaluation, we believe that the selected videos still provide a meaningful representation of diverse conditions under which the model is expected to perform. These videos encompass a range of scenarios, including differences in water clarity, lighting conditions, and fish visibility, that our model should handle. Therefore, our performance evaluation, conducted with a limited set of videos, provides valuable insights into our model's capabilities and areas for improvement. This experience has reinforced the importance of careful resource management, particularly when manual data preparation is a critical component.

QUALITY ASSURANCE

Ensuring the adherence to high standards throughout all aspects of our project necessitated the utilization of various tools. These tools encompassed code development, documentation, testing, and facilitating effective group collaboration. This chapter elucidates the utilization and application of these different tools and practices, providing insights into how they contributed to achieving optimal quality and supporting our project's progress.

9.1 Source Code Quality Assurance

Quality assurance of our source code is paramount in this project. We maintain a high standard of code quality by adopting stringent guidelines and a set of tools, each serving a specific purpose.

- PEP 8[49]: The Python Enhancement Proposal 8, commonly known as PEP 8, is a set of recommendations for how to format Python code. We chose PEP 8 over other style guides such as Google's Python Style Guide due to its wide acceptance in the Python community and its emphasis on readability and ease of maintenance. PEP 8 is recognized as the de-facto standard for Python code, making it a clear choice for our project.
- Black[50]: This is a strict code formatter for Python that adheres to the PEP 8 guidelines. We chose Black over other formatters like autopep8 or yapf due to its uncompromising nature. Unlike autopep8, which only corrects parts of the code that violate PEP 8, Black takes a more proactive approach, formatting all code to its own standard. And compared to yapf, which allows some customization, Black's philosophy of reducing decision fatigue by having a single style setting stood out to us.
- Pylint[51]: This is a robust source code, bug and quality checker for Python. We opted for Pylint over other linters like flake8 or pyflakes due to its comprehensive checks and ability to enforce coding standards. While flake8 and pyflakes can detect some issues in the code, Pylint offers a more in-depth analysis. This includes checking if modules are imported but unused, if variable names follow the naming convention, and even if the code's docstrings are up to standard.

- mypy[52]: This is a static type checker for Python. We chose mypy over other static type checkers like PyType and Pyright due to its seamless integration with existing Python syntax and its comprehensive type checking capabilities. While PyType can infer types and catch common errors, and Pyright offers performance benefits, Mypy's balance of robust type checking and ease of use made it our top choice.
- isort[53]: This tool is used to sort and organize imports in your Python code. We chose isort over other tools like import-linter or flake8-import-order due to its ability to automatically separate imports into sections and by type, which not all tools provide. This makes the code cleaner and easier to navigate. It also checks the code against its sorting and formatting standards, ensuring that all imports are correctly placed and formatted.

9.2 Logging

We incorporated a custom logging system into our codebase. This tool was instrumental in enabling effective debugging. It also facilitated communication within the team by providing clear, timestamped logs of system behavior.

The logger was implemented as a Python script, which created a logger with both a file handler and a console handler. It uses a timed rotating file handler from Python's logging module, which rotates the log files at midnight and retains ten old log files as a backup. The logs are saved in a designated folder, created automatically if it doesn't exist. The system also uses a console handler to output the logs in the console, aiding in real-time debugging.

Logging played a significant role in our testing practices. The detailed logs provided by our custom logger were essential in tracking and identifying bugs, system behavior, and performance issues. It was particularly useful during the integration testing with NINA, (described later in subsection 9.5.3), as the logs provided insights into system performance and bottlenecks.

The logger script can be referred to in Appendix I for detailed reference.

9.3 Git Branching

We follow the standard Git branching model in our project. This involves using feature-based branches for isolated development of new features or tasks. Once a feature is complete and has been thoroughly tested, it is merged into a development branch. This workflow allows us to work on multiple features simultaneously without causing conflicts or disruptions. It also helps keep the codebase clean and well-structured.

Upon reaching a stable state, the development branch is merged into the main branch, which is often considered the "production-ready" branch. This strategy ensures that the main branch always hosts stable, tested, and reliable code.

9.4 Documentation

Documentation forms an integral part of our project. It serves as a roadmap for both our code and processes, making our work accessible and understandable to others, including future contributors.

Our approach to code documentation involves using Docstring - a type of comment used to explain the purpose of a function, method, class, or module in Python. Docstring are placed immediately after the definition of a function, method, or class and are enforced by Pylint, our chosen Linter.

We also use Git commits as an integral part of our documentation strategy. Every change committed to the repository should be accompanied by a clear and concise message that justifies the change and explains what it does. This provides a comprehensive version history and allows us to use the ‘git blame’ command to easily track who made a certain change and why it was introduced. This form of documentation ensures a high level of transparency and accountability within our team, making it easier to understand the evolution of the codebase over time.

To document our processes, we maintain summaries of our meetings, detailing significant discussions and decisions. We also provide a comprehensive installation guide for setting up the application, ensuring a repeatable and straightforward process for all users.

Toward the end of our project, we use ChatGPT as a language correction tool for our final report. This tool enhances language usage, rectifies grammatical errors, and ensures a consistent writing style, thereby contributing to a high-quality final report.

9.5 Testing

Testing is an essential part of the software development process. It helps us to identify any missing requirements, bugs, or errors and evaluate the software’s security and reliability.

9.5.1 Continuous Integration: Build and Unit Testing

Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently. This leads to multiple integrations per day. Each integration is verified by automated builds and tests to detect integration errors as quickly as possible.

Our project uses GitHub for Continuous Integration. The CI pipeline is defined in a ‘.github/workflows/code-quality.yaml’ file located in the root directory of our project. This file specifies the steps that GitHub performs to build and test our project.

The pipeline setup involves several steps. First, we set up the Python environment with the required version. Then we install the necessary dependencies using pip and Poetry, a tool we utilize for dependency management. After setting up the environment, the pipeline runs the pre-commit hooks. These hooks include trailing whitespace removal, end-of-file fixer, YAML checks, and formatting checks using Black. mypy is run to check for type consistency in the code. Pylint is used to identify bugs and quality issues.

The pre-commit [16] hooks also include checks using commitlint, a tool that helps enforce a consistent commit message style across the project. This is particularly important for maintaining a readable and navigable commit history. We use the '@commitlint/config-conventional' configuration, which adheres to the Conventional Commits[54] specification. Conventional Commits is a lightweight convention that encourages readable commit messages that are easy to automate. It prescribes a simple set of rules for creating an explicit commit history, which makes it easier to write automated tools on top of. This can be particularly useful for generating release notes, for example. If a commit message doesn't follow the specified format, the commit is rejected, encouraging the developer to properly document their changes. This way, we can use Git blame effectively to understand why changes were made.

The main tests we run on the CI are unit tests. Unit tests are written for functions and methods to validate that each piece of the program performs as expected. These tests are crucial for catching and fixing bugs early in the development process, before they make their way into the final product.

These checks ensure that the code that is being integrated passes all quality and format checks, including unit tests. If any of these checks fail, the integration is stopped, and the developer is notified to fix the issues. This way, no code is merged into the main branch without passing all the checks and tests.

9.5.2 User Testing

We also carry out user testing with NINA in a controlled environment. This allows us to receive valuable feedback and iterate on our product. Self-testing is also utilized to validate the steps for software installation and to ensure its core functionalities.

In appendix E is the form that we utilized to perform user tests with NINA's representatives during our client meetings. Through the utilization of this form we discovered issues the representatives may have encountered while using the application. These had a main focus on the Front-end's user interface. Further details on the contents of the user test can be found in the meeting logs with NINA in appendix G on the date 17th of April.

9.5.3 Integration Testing

Integration testing forms a vital part of our project's quality assurance. In particular, we carried out integration testing with Francesco Frassinelli from NINA. This involved the development of a custom script designed to test various model sizes and batch sizes on NINA's VDI setup.

The script, as detailed in Appendix I, initializes different model sizes in a sequential manner, ranging from small to large. For each initialized model, it processes the video using different batch sizes. If the model processing hits an out-of-memory error, it skips the current batch size and moves on to the next. This script was designed to be robust and flexible, allowing us to effectively gauge the performance of our solution across different configurations.

This script was incorporated into a dedicated test branch named "vdi-perf-test". We shared this branch with Francesco, enabling him to test the script on

their VDI. He created a Docker setup for our application and provided us with the performance results. This valuable collaboration helped us obtain real-world data on our solution's performance on the actual deployment hardware.

This real-world testing also enabled us to identify and address specific issues. Notably, it led us to revise the operation of our image loader to optimize its performance on their hardware. This testing phase was instrumental in ensuring that our solution met our performance expectations in the actual deployment environment.

9.6 Group Work

To ensure effective collaboration, we held daily meetings to discuss our work and progress. Our work is documented and managed through issues on our version control system. This ensures that every aspect of the project is discussed, and progress is made at each step.

In essence, our approach to code quality assurance, Git branching, documentation, testing, and group work, is designed to ensure a high standard of work, while fostering a productive and harmonious working environment for the group. These principles and practices would not only lead to a successful project but also provide a solid foundation for any future work we undertake.

DISCUSSION

This chapter gives a thorough discussion of choices that were made throughout the development process and of the following results. It will also provide insight into the success of the planned development process and changes that were made or should have been made during the process.

10.1 Preparatory Work

This section critically evaluates the adequacy of the preparatory work undertaken and its effectiveness in providing a solid foundation for conducting the bachelor's thesis. Additionally, an assessment will be made regarding the feasibility and attainability of the set goals, followed by an analysis of their ultimate outcomes and achievements.

10.1.1 Project Plan

The project plan, outlined in Appendix C, provided a comprehensive roadmap for the entire project prior to its initiation. Throughout the course of the project, we adhered closely to this plan, deviating minimally from the agreed-upon framework. The presence of a well-defined plan and established rules facilitated a structured and efficient workflow, allowing for timely resolution of any issues that arose. An exemplification of this can be observed in section 10.3.1. By meticulously outlining the project requirements, we were able to adhere closely to the Gantt chart we devised, successfully following it almost entirely. Although some minor deviations occurred around two-thirds into the planned Gantt timeline, overall, we believe that the presence of a well-crafted plan and its active utilization throughout the project enhanced our efficiency and organization.

10.1.2 Goals

As we started on this project, our original result goals (as stated in Appendix C) were centered on developing an object detection model that would offer a balanced measure of precision and recall in identifying individual frames with fish. We aimed to achieve an overall detection rate of 95%, striving for the F_1 -score to capture a high degree of accuracy and comprehensiveness in fish detection.

However, following further analysis at the onset of development and insightful consultations with NINA, we revised our project goals. Our shift was largely influenced by two factors: a clearer understanding of NINA’s operational needs and the practical challenges associated with detecting fish in individual frames.

Firstly, our discussions with NINA revealed a distinct operational requirement: to know the ranges within which fish were present in the videos, rather than knowing the presence of fish in each specific frame. This requirement was rooted in the nature of their research and operational workflows, where continuous detection was more critical than isolated frame-wise identification.

Secondly, the challenge of achieving high precision in detecting fish in individual frames became apparent as we started the development process. Given the variability in frame quality and the potential for many frames to lack distinctive features, it was more practical and efficient to focus on ranges of frames where fish were present.

Hence, our revised goals emphasized achieving a high recall rate for detecting fish within ranges in the videos, targeting at least 95% of the ranges where fish are actually present. This goal allowed us to concentrate on minimizing false negatives, aligning well with NINA’s need to ensure that the presence of fish within a range is detected, even at the expense of occasional false positives.

This shift in focus had a profound impact on our development strategy. We designed an algorithm to convert a list of detected frames into a list of ranges, accommodating some frames without detections within a valid range. This algorithm, outlined in Section 6.3.4.1, became a key part of our model and allowed for a more robust detection process.

Furthermore, we maintained our goal to reduce human effort required for video processing tasks, which remained a critical aspect of the project. Our revised goal aimed to ensure the model was both effective in achieving its primary purpose and efficient, enhancing productivity in NINA’s research endeavors.

In conclusion, our goal revision led to a more tailored solution that better addressed NINA’s needs and allowed us to navigate the practical challenges associated with video-based fish detection. Despite the shift in focus, we stayed true to our overarching aim of optimizing fish detection and processing workflows, thereby benefiting NINA and improving the efficiency of their video analysis pipeline.

10.2 Cooperation

This section provides an assessment of the cooperation and collaboration within the project team throughout the duration of the project. We critically evaluate whether there were any areas that could have been improved upon and reflect on the allocation of work in relation to each group member’s skills and preferred areas of expertise. Additionally, we analyze whether the allocation of tasks was carried out in a time-efficient manner.

10.2.1 Meetings

The scheduling and sequence of our meetings proved to be highly effective. As depicted in Table 3.2.1, our sequential approach enabled us to address and discuss pertinent matters from previous meetings in a timely and organized manner. For

instance, when our client expressed the need for adjustments to the report format (Section 10.3.1), we promptly tackled this issue in subsequent meetings with our supervisor and incorporated it into our Sprint planning sessions. However, we acknowledge that there were opportunities for improvement in terms of the structure and focus of our meetings. The group meetings often took on a more casual nature, fostering an open atmosphere conducive to group discussions. Nevertheless, this informality occasionally resulted in extended meeting durations and topics that were tangential to the project work.

10.2.2 Work Allocation

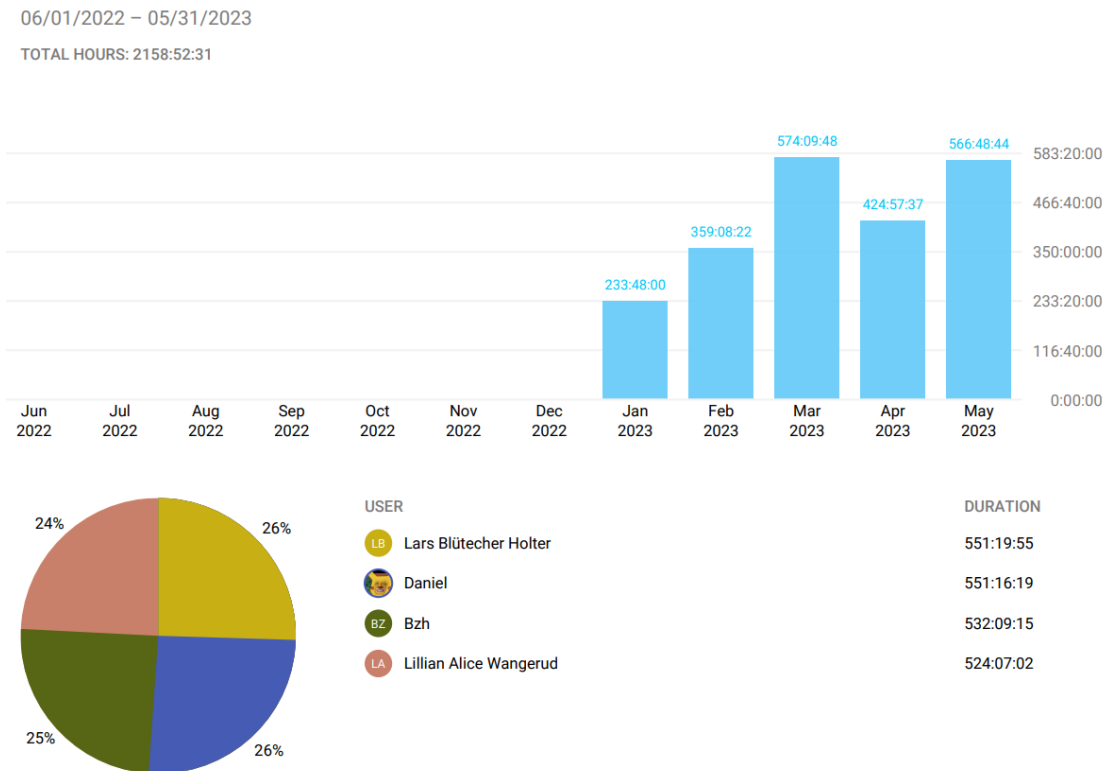


Figure 10.2.1: Work Distribution.

The allocation of different areas of responsibility, as outlined in section 1.4.2, enabled us to concentrate on our respective areas and acquire in-depth knowledge in those domains. However, this focused approach inadvertently led to a certain level of unawareness regarding specific key aspects of each other’s work. An illustration of this challenge is evident in our limited understanding of the inner workings of certain scripts used for dataset generation. Consequently, when the need arose to collaborate and support each other in the later stages of the project, a substantial amount of time had to be dedicated to teaching and familiarizing ourselves with these intricate details. In hindsight, it would have been advantageous to ensure that each group member contributed more significantly to the development of Back-end scripts. Despite this limitation, we believe that the overall allocation of work was successful and effectively assigned responsibilities for each aspect of the project, giving every team member a sense of ownership and accountability. See Figure 10.2.1 for how the hours were distributed.

10.3 Flexibility of Development

In this section, we will revisit our choice of development method and how it has effected our development. We discuss how it allowed us to make changes to key aspects of our project, without having to redo substantial amounts of work.

10.3.1 Change in Task Description by Client

From the project's inception, we sought a software development model that offered flexibility and facilitated accommodating changes as they arose. This decision proved wise as, approximately one month into development, NINA approached us with a request to modify certain task descriptions. Specifically, the change involved transitioning from a Windows-based program to a Linux-based program accessible through a VDI. Following discussions during subsequent meetings, we determined that this alteration was feasible without significant obstacles. The success in addressing this request was attributable to our software development approach, which embraced the Scrumban model and prioritized modularity and flexibility. By adopting this strategy, we could isolate the specific sections of our program requiring modification, thereby minimizing the required rework. Such changes encompassed code revisions to detect the OS on which the program was running, as well as the incorporation of more substantial elements. These additional elements encompassed configuring PyQt to function with XCB, ensuring compatibility of our GUI within a Docker container and implementing X11 forwarding, developing performance tests for the image loader and batch YOLOv8 model processor (Section 9.5.3), and creating a suitable Dockerfile for the VDI environment utilized by NINA. Despite these changes being requested approximately one-third into the development phase, we effectively considered and integrated them into our project. Furthermore, we commenced collaboration with NINA's IT department to align our solution with their internal system configurations. This collaboration involved inviting IT personnel to client meetings, engaging in email correspondence, and sharing our code repository on Git for internal testing, suggestions, and review of pull requests. By collaborating closely with NINA's IT department to integrate our solution, we ensured the usability of our project and provided the foundation for potential future iterations by the IT department beyond the scope of our thesis.

10.3.2 Report Structure Changes

During the later stages of our product development, a user test was conducted in collaboration with NINA. In this test, valuable feedback was obtained regarding the format of the report generated by our program. Specifically, NINA expressed a preference for an alternative data format in the output file. In response, we requested that they provide a template illustrating their desired format. After a few days, we received the template; however, it became apparent that their request included the tracking of individual fish in the report, along with several other data sets that were not feasible to extract from our AI model. Subsequent discussions took place during group meetings and client interactions, aiming to explore the reasons behind this technical limitation. It was concluded that NINA had overlooked the project delimitations outlined in Section 1.3. Despite the

inability to fulfill certain requested data requirements, a mutual agreement was reached, resulting in the inclusion of additional data that had not been previously incorporated. An example of this additional data is a list of the summary of every file processed with date, original- and new video length. Moreover, the data was organized in an intuitive manner, allowing NINA to access both a summary of all video files and their corresponding data, as well as individual detections for each video. Consequently, the final version of the report provides detailed information regarding each detection, including their respective start and stop times.

10.3.3 CVAT

CVAT, or Computer Vision Annotation Tool, is a powerful tool for annotating and labeling datasets. However, it also has some limitations and problems that must be addressed.

One issue that we have encountered with CVAT is that it can become overloaded when working with multiple people simultaneously. This can slow down the process and make it difficult to collaborate effectively. Additionally, using too many instances of tracking on a large sample size of fish can also lead to performance issues.

Other problems we have encountered occurred when we tried to upload files that were partially corrupt, CVAT had difficulty loading corrupt video files, this was eventually fixed when the problem was identified, however, it was still a problem that was unexpected. Another very frustrating problem that occurred, occurred when we tried to track a lot of fish at the same time with large datasets, which would eventually lead to extreme inaccuracies and inconsistencies in retaining information, which led to wasted time and effort. Finally, there have been cases where we have inadvertently labelled datasets that were already included in our collection, leading to even more unnecessary redundancy and wasted resources.

Despite these issues, there are many benefits to using CVAT in a collaborative environment. One of the most significant advantages is the ability to share labels between multiple users, making it easy to collaborate on large datasets. Additionally, CVAT allows for the superclassing of labels, which can help streamline the annotation process and improve accuracy. Finally, CVAT ensures that there is no overlap of annotated data (except in cases where we have provided the COCO - YOLO dataset), which helps to ensure the accuracy and consistency of our dataset.

If we continued to work with CVAT, there would be several future considerations that we would have to keep in mind. One possibility would be to outsource more annotated data to reduce the workload on our internal team. Additionally, we could have explored the use of a 3D game renderer to generate training data, which could help mitigate some of the challenges associated with annotating real-world footage. Overall, while CVAT has its limitations and challenges, it has been a valuable tool for our work on fish detection and monitoring in murky water environments.

10.3.4 Windows Python

And an important thing to note is that when downloading Python from the windows store and using that instance of Python for the environment will lead to an insurmountable amount of issues related to Paths, permissions and directories. This is because of how Windows store handles its installed version of Python which differs from how the default installation of Python is handled. Windows Store edition handles set up of your PATH settings for the current user (avoiding the need for admin access), in addition to providing automatic updates, which might be problematic to compatibility with libraries that have not updated. Multiple group members experienced problems regarding Poetry and other installation packages due to the Windows Store Python which were inevitably fixed by switching Python installs.

10.4 Process Critique

In addition to the aforementioned points, there are several areas of improvement that we would address if given the opportunity to continue or replicate the project. Firstly, we recognize the need for a stronger emphasis on testing throughout the development process. This entails the creation of more comprehensive unit tests and the automation of code testing procedures. Such measures would result in improved code quality, fewer bugs, and a more robust and reliable codebase. Nevertheless, we did implement logging mechanisms in most aspects of our program. This allowed us to identify instances where code did not behave as expected or, in worst-case scenarios, resulted in program crashes. Different types of log messages, such as CRITICAL, WARNING, ERROR, DEBUG, and INFO, were utilized to provide concise descriptions of when and why such events occurred.

Another area requiring more attention is the inclusion of additional and enhanced user testing, as well as increased iterations of our program. These measures would have contributed to refining the appearance and functionality of our GUI. However, considering the intended purpose of our program and its target end-users, we believe the current state of the GUI to be adequate. It is user-friendly and avoids unnecessary distractions that could divert attention from its intended functionality. The feedback received from the conducted user tests has generally been positive, with minimal suggestions for major revisions, indicating a satisfactory initial impression.

An aspect we would prioritize for improvement and alteration pertains to the organization and structure of our Front-end code. Although not directly discernible to end-users, the current code structure may present challenges for future developers who aim to continue working on the Front-end. Specifically, the code is divided into a combination of appropriately nested files within folders, as well as some loosely placed files within the root directory of our application. This inconsistency may potentially impede efficient comprehension and navigation within the codebase. Enhancing this aspect would result in improved code management and maintenance. Although we initially planned to address this issue, time constraints necessitated a shift in priorities toward other aspects of our project.

Apart from these areas of critique, we are generally content with the project's overall execution. If given the opportunity to undertake the project again, we

would largely adhere to the planned approach outlined in the project plan (Appendix C), with only minor modifications.

CONCLUSION

This chapter serves as the concluding section of our thesis, providing a comprehensive summary of the achievements and outcomes of our project. It highlights the key findings, contributions, and accomplishments that have emerged throughout the research and development process. Furthermore, we identify potential areas that can be further explored and improved upon in future iterations or related studies. This concluding chapter concludes with our final remarks on the thesis, reflecting on the overall significance of the work conducted and its potential implications for the field. By encapsulating the essence of our project and offering insights for future work, this chapter brings closure to our thesis and solidifies its contribution to the existing body of knowledge.

11.1 Project Achievements

Throughout this thesis, we have successfully attained a multitude of predetermined goals. These objectives encompassed our overarching vision for the program's implementation, as well as the aesthetic and functional aspects of the GUI. Furthermore, we aimed to accomplish various performance-related and impact-oriented goals. By achieving all these goals, NINA's problem would hopefully be solved. The conclusive outcomes of these endeavors are comprehensively discussed in the subsequent sections.

11.1.1 Implementation and Interface

As a culmination of our efforts, the implementation of our project has resulted in the development of an intuitive and user-friendly program. This program effectively processes input videos, generating new videos that exclusively contain fish detections. Additionally, the program generates a comprehensive report available in both XLSX and CSV file formats. This report serves as a valuable resource for the researchers at NINA, enabling them to not only access visual data in the form of videos for analysis but also leverage written data for rapid examination and statistical analysis.

The availability of both visual and written data provides researchers with a multifaceted approach to study and understand the aquatic environment. By utilizing the generated videos, researchers can visually analyze fish behaviors, move-

ments, and interactions. Simultaneously, the written data in the form of reports offers a structured and organized representation of the detected fish, facilitating efficient data exploration, extraction, and statistical analysis.

By placing a strong emphasis on modularity and iterative development, we have purposefully designed our project to facilitate ongoing improvement and enhancement by NINA's IT department. Furthermore, we would like to promote knowledge sharing and collaboration, which is why we plan to publish the dataset and accompanying scripts on Kaggle.com. A comprehensive guide has also been provided to assist researchers and practitioners in leveraging our work for advancing video detection research in diverse environments. Through these initiatives, we aspire to contribute to the broader research community and support the pursuit of United Nations Sustainable Development Goals.

11.1.2 Performance

We had several major goals for this project, which is detailed in Section 2.1.1. These goals focus on the quality of the application and its functionality. In this section, we discuss how we conducted the performance evaluation of our solution and the results we obtained. Thus, also assessing how well our solution satisfies our result goals.

The performance evaluation was conducted using a high-performance computer system (as detailed in Table 8.2.1), equipped with an RTX 3090 GPU. This setup closely resembled the VDI setup used by NINA, which is powered by an RTX A4000 GPU, hence providing a valid and relevant testing environment for our solution.

We chose three 30-minute videos for the testing process. These videos, provided by NINA, were each 1920x1080 in resolution, with a frame rate of 25 frames per second, and a file size of 960MB. We processed these videos through the entire pipeline of our solution, which included detection, video cutting, and report generation, using the YOLOv8s model (*v8s-640-classes-augmented-backgrounds*).

The performance results, obtained from our solution, demonstrate a significant improvement, exceeding the initial goals set for this project as outlined in Section 2.1.1.

In our initial goals, we aimed to reduce the processing time to 25 minutes for a 30-minute video. However, the testing showed that our solution could process a 30-minute video in an average of 130.39 seconds, approximately 2.2 minutes. This achievement denotes a substantial reduction in the resources required for video analysis operations, considering our solution could process videos around 14 times faster than real time. Consequently, we were also able to reduce the storage needed for a single file, from 960MB to only 3.7MB, which is around 259 times smaller when using CRF on 21.

Such success was made possible due to our dedicated work on the optimization of different components of our solution. The Image Loader's efficiency was significantly improved (as discussed in Section 6.4.1). The implementation of a custom Model that utilizes batching (as described in Section 6.4.2) and the Video Processor that efficiently cuts and annotates video (explained in Section 6.3.3) further boosted the performance of our system.

However, these results were obtained under optimal testing conditions with a

high-end hardware setup. Therefore, it's essential to note that actual performance may vary depending on the specific system configuration and the complexity of the video content being processed.

As delineated in Section 2.1.1, our primary objective was to ensure a 95% accuracy rate for detecting fish ranges in NINA's provided videos. This goal guided our methodology in testing and applying various weights from the YOLOv8 model, such as *v8s-640-classes-augmented-backgrounds*, *v8s-640-classes-augmented-oreskyt*, *v8s-640-classes-augmented*, and *v8s-640-classes*.

Our systematic and comparative testing strategy enabled us to isolate the most proficient configuration. In this case, the *v8s-640-classes-augmented-backgrounds* weight consistently outperformed the others and was thus employed in our final performance evaluation. To assess this efficacy, we computed the average recall accuracy across all valid videos. More specifics about this process are documented in Subsection 8.3.3.

Following a thorough and meticulous evaluation, we were pleased to discover that our solution's average recall accuracy stands at an impressive 96.4%, exceeding our initial target by maintaining a confidence threshold of 70%. These outcomes underscore the precision of our solution when it comes to detecting ranges of fish. Nevertheless, it's worth noting that these assessments were conducted solely on valid videos; performance may fluctuate with different conditions or datasets.

In conclusion, we are delighted to report that we have not only achieved, but surpassed our initial result goals. These results testify to the effectiveness of our solution, and they promise significant improvements in speed, storage, and accuracy in fish detection.

11.2 Future Work

While we are satisfied with the overall outcome of our project, there are several potential areas that warrant further exploration and improvement. The following list outlines potential areas for enhancement:

- To enhance the accuracy of our model, it is recommended to annotate a larger set of relevant videos for training the weights. This entails annotating videos containing a lower representation of specific fish species and expanding the dataset to include additional species that are not currently being tracked. By encompassing a comprehensive range of species likely to appear in the video frames, we aim to ensure coverage of all possible species. To facilitate future work in this area, a detailed guide has been developed, providing instructions on the annotation process and methodology. For further information, please refer to Appendix H
- To broaden the scope of our application and accommodate diverse environments, you can generate additional datasets and train new models. These datasets should encompass various settings and include different species and animals beyond the current scope. For instance, the creation of a new model capable of detecting woodland animals captured by trail cameras would expand the applicability of our program. Similarly, developing a distinct model

capable of detecting objects and organisms in other bodies of water, such as the ocean, would further extend the program's versatility. By undertaking these endeavors, we can effectively cater to a wider range of scenarios and provide valuable solutions for different ecological contexts.

- Implement a feature to run the program on real-time recordings.
- Include option for tracking fish, not just detection on a per image level. To do this you would however need to use a different AI model and train it up. This feature could enable the possibility to recognize already tracked fish and detect specific fish behavior.
- Create an API for the software such that it can be used over a server, enabling a whole network to use the software without downloading it separately on all computers.
- Implement more testing on the software. See Section 10.4 for more in depth on what kind of tests and how this could be done.

11.3 Final Words

Throughout the course of this bachelor thesis, we have leveraged the knowledge and expertise acquired during our undergraduate studies, while also acquiring new profound knowledge in various areas of specialization, including AI and image recognition, full-stack development, and more. This extensive experience has equipped us with the necessary skills to effectively tackle new challenges and successfully overcome them.

By harnessing our knowledge and skills, we have developed a comprehensive system that significantly improves workflow efficiency, reduces resource requirements, and empowers the researchers at NINA to gather a wealth of valuable data for their ongoing research endeavors.

In conclusion, through effective teamwork and meticulous planning, we have accomplished the creation of a fully functional system that will make substantial contributions to scientific advancements pursued by NINA in the years to come. Overall, we take great pride and satisfaction in our work and are delighted with the successful outcome of the project.

BIBLIOGRAPHY

- [1] NINA - Norwegian Institute for Nature Research. <https://www.nina.no/english/Home>. Accessed January 11, 2023.
- [2] United Nations. *Sustainable Development Goals: Goal 14 - Life Below Water*. Accessed: January 18, 2023. n.d. URL: <https://sdgs.un.org/goals/goal14>.
- [3] Scott Chacon and Ben Straub. *Git*. <https://git-scm.com/>. 2023.
- [4] *GitHub*. Web Page. 2020. URL: <https://github.com/>.
- [5] Atlassian. *SourceTree*. <https://www.sourcetreeapp.com/>. 2023.
- [6] Axosoft. *GitKraken*. <https://www.gitkraken.com/>. 2023.
- [7] Discord Inc. *Discord*. Computer Program. 2015. URL: <https://discord.com/developers/docs/intro>.
- [8] Alari Aho and Krister Haav. *Toggl Track*. Web Page. 2006. URL: <https://developers.track.toggl.com/docs/>.
- [9] Microsoft. *Microsoft Teams*. <https://www.microsoft.com/en-us/microsoft-teams/>. 2023.
- [10] Docker Inc. *Docker*. <https://www.docker.com/>. 2023.
- [11] Project Jupyter. *Jupyter*. Documentation. 2014. URL: <https://jupyter.org/>.
- [12] Nextcloud GmbH. *Nextcloud*. <https://nextcloud.com/>. 2023.
- [13] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.
- [14] Riverbank_Computing. *PyQt*. Documentation. 1998. URL: <https://wiki.python.org/moin/PyQt>.
- [15] Sébastien Eustace. *Poetry*. 2018. URL: <https://python-poetry.org/>.
- [16] Sottile, Anthony and contributors. *pre-commit*. <https://pre-commit.com>.
- [17] CVAT Contributors. *CVAT: Computer Vision Annotation Tool*. <https://github.com/openvinotoolkit/cvat>. 2023.
- [18] OpenCV. *CVAT Documentation: YOLO Format*. 2021. URL: <https://opencv.github.io/cvat/docs/manual/advanced/formats/format-yolo/>.

- [19] Martin Fitzpatrick. *PyQt vs Tkinter – a Guide on Choosing the Right GUI Framework for Your Python App*. 2021. URL: <https://www.pythonguis.com/faq/pyqt-vs-tkinter/>.
- [20] Karl Kroening. *ffmpeg-python: Python bindings for FFmpeg*. Documentation. 2017. URL: <https://kkroening.github.io/ffmpeg-python/>.
- [21] K. S. Bealer. *PyAV: A Pythonic binding for FFmpeg/Libav*. 2020. URL: <https://github.com/mikeboers/PyAV>.
- [22] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: <https://github.com/ultralytics/ultralytics>.
- [23] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].
- [24] Ultralytics. *YOLOv5: YOLOv5 in PyTorch*. <https://github.com/ultralytics/yolov5>. 2021.
- [25] Thierry Bouwmans. “Recent Advanced Statistical Background Modeling for Foreground Detection: A Systematic Survey”. In: *Recent Patents on Computer Science* 4 (Sept. 2011), pp. 147–176. DOI: 10.2174/1874479611104030147.
- [26] John Barron, David Fleet, and S. Beauchemin. “Performance Of Optical Flow Techniques”. In: *International Journal of Computer Vision* 12 (Feb. 1994), pp. 43–77. DOI: 10.1007/BF01420984.
- [27] Alper Yilmaz, Omar Javed, and Mubarak Shah. “Object tracking: a survey. ACM Comput Surv”. In: *ACM Comput. Surv.* 38 (Dec. 2006). DOI: 10.1145/1177352.1177355.
- [28] David Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94.
- [29] Chris Stauffer and W. Grimson. “Adaptive background mixture models for real-time tracking”. In: *Proceedings of IEEE Conf. Computer Vision Patt. Recog, vol. 2* 2 (Jan. 2007).
- [30] Donovan H. Parks and Sidney S. Fels. “Evaluation of Background Subtraction Algorithms with Post-Processing”. In: *2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance* (2008), pp. 192–199.
- [31] Roberto Brunelli. *Template matching techniques in computer vision*. en. Hoboken, NJ: Wiley-Blackwell, Mar. 2009.
- [32] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-yuan Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: (Apr. 2020).
- [33] Adam Paszke et al. *PyTorch*. 2017.
- [34] NVIDIA. *CUDA Toolkit Documentation v12.0*. Documentation. URL: <https://docs.nvidia.com/cuda/>.
- [35] Nat Noordanus. *Poet the Poet*. Online. Accessed March 23, 2023. 2020. URL: <https://poethepoet.natn.io/>.

- [36] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [37] *Pull Request 668*.
url<https://github.com/ultralytics/yolov5/pull/668>. Accessed May 16, 2023.
- [38] *Figma: the collaborative interface design tool*.
url<https://www.figma.com/>. Accessed May 21, 2023.
- [39] D. A. Norman. *The Design of Everyday Things: Revised and Expanded Edition*. New York City: Basic Books, 2013.
- [40] J. Nielsen. *F-shaped pattern for reading web content*. Retrieved May 3, 2023. 2006. URL: <https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/>.
- [41] Birger Johan Nordølum et al. “Artsgjenkjenning av fisk”. Norwegian (Bokmål). In: (2021). Bachelor thesis. URL: <https://hdl.handle.net/11250/2777966>.
- [42] *Dataset Management Framework (Datumaro)*.
url<https://github.com/openvinotoolkit/datumaro>. Accessed May 21, 2023.
- [43] *How to train to detect your custom objects in darknet*. GitHub README. <https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects>. May 2023.
- [44] Kevin Shen. “Effect of Batch Size on Training Dynamics”. In: *Medium* (June 2018). URL: <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>.
- [45] *Ultralytics Documentation: Train*.
url<https://docs.ultralytics.com/usage/cfg/#train>. Accessed May 18, 2023.
- [46] *Issue 1839 Comment*.
url<https://github.com/ultralytics/ultralytics/issues/1839#issuecomment-1497317519>. Accessed May 16, 2023.
- [47] *Issue 509*.
url<https://github.com/ultralytics/ultralytics/issues/509>. Accessed February 02, 2023.
- [48] *Issue 577*.
url<https://github.com/ultralytics/ultralytics/issues/577>. Accessed February 02, 2023.
- [49] Guido van Rossum, Barry Warsaw, and Nick Coghlan. *PEP 8 – Style Guide for Python Code*. Documentation. 2001. URL: <https://peps.python.org/pep-0008/>.
- [50] Łukasz Langa et al. *Black*. Documentation. 2011. URL: <https://black.readthedocs.io/en/stable/>.
- [51] *Pylint*. Documentation. URL: <https://pylint.readthedocs.io/en/latest/>.
- [52] Python Software Foundation. *MyPy*. <http://mypy-lang.org/>. Accessed 2023.

- [53] Timothée Mazzucotelli. *isort*. <https://pycqa.github.io/isort/>. Accessed 2023.
- [54] *Conventional Commits*. <https://www.conventionalcommits.org/>. Accessed: May 22, 2023. May 2023.

APPENDICES

A Project Agreement

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett

Er den rett som den som skaper et åndsverk har til å fremstille eksemplarer av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:
Veileder ved NTNU: e-post og tlf.
Ekstern virksomhet: Ekstern virksomhet sin kontaktperson, e-post og tlf.:
Student: Fødselsdato:
Ev. flere studenter ¹ Daniel Hao Huynh, 13.03.2001 Lars Blütecher Holter, 17.09.1999 Lillian Alice Wangerud, 16.06.2001 Benjamin Letnes Bjerken, 09.05.1999

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	x
Prosjektoppgave	
Annen oppgave	

Startdato: 11.02.2023
Sluttdato: 22.05.2023

Oppgavens arbeidstittel er: Deteksjon av bevegelse i undervannsvideo

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

¹ Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:
--

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven². Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

² Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

Alternativ a) (sett kryss) Hovedregel

<input checked="" type="checkbox"/>	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven
-------------------------------------	--

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

<input type="checkbox"/>	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt
--------------------------	---

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

<input checked="" type="checkbox"/>	Opgaven skal være offentlig
-------------------------------------	-----------------------------

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i

denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss	Sett dato
<input type="checkbox"/>	ett år
<input type="checkbox"/>	to år
<input type="checkbox"/>	tre år

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt





Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato:
Veileder ved NTNU: Dato:
Ekstern virksomhet: Dato:
Student: Dato:
Ev. flere studenter  Daniel Hao Huynh: 19.01.2023  Lillian Alice Wangerud: 19.01.2023  Lars Blütecher Holter: 19.01.2023  Benjamin Letnes Bjerken: 19.01.2023

B Task Description

Oppdragsgiver: Norsk Institutt for Naturforskning (NINA)

Kontaktperson NINA: Knut Marius Myrvold / knut.myrvold@nina.no / 920 64 963

Tittel: Deteksjon av bevegelse i undervannsvideo

Videooptak brukes i økende grad i overvåking av natur og dyreliv, både på land og i vann. En viktig grunn er at video er en passiv og skånsom metode som ikke er avhengig av fangst og håndtering av dyr. Norsk Institutt for Naturforskning (NINA) bruker i dag undervannskamera til å filme fisk i ulike miljøer og for ulike formål. Fram til nå har vi i samarbeid med NTNU undersøkt vandring i fisketrapper og hvordan oversvømte arealer brukes av fisk på våren og sommeren.

Filming under vann byr på utfordringer for både biologer og dataingeniører: Grumsete vann, algevekst, andre objekter som oppleves som støy (eks. gress og alger som beveger seg i bildet), og ikke minst mye dødtid mellom de relevante klippene. Utfordringen er å skille relevant informasjon i et mylder av bevegelige objekter, og å unngå å bruke opp lagringsplass på «unødvendige» klipp.

Vi ser et stort behov for å effektivisere datahåndteringen. Til nå har vi lagret alt videomateriale på harddisker ute i felt (dvs. vekk fra pålitelig bredbånd og strømtilgang), noe som er både dyrt og sårbart. Vi ser at et naturlig neste steg er å unngå å lagre materiale som ikke er relevant. Vi ønsker en løsning som kan bruke en kontinuerlig videostrøm til å lagre *viktige klipp*, og dermed unngå å fylle opp harddisker med video som ikke gir noe relevant informasjon. Det er også ønskelig med en rapportering over nett som ikke trekker for mye data (over 4G nett). Programvaren må kunne kjøres på Windows fra en enkel GUI. Programmet må kunne snakke med ekstern harddisk for lagring/sletting av klipp.

Interesse for eller bakgrunn innen maskinsyn/bildebehandling er ønskelig, samt en pragmatisk tilnærming til praktiske løsninger innen datahåndtering. Utviklingsmiljø og konkrete oppgaver bestemmes i dialog med oppdragsgiver. Gruppen vil få tilgang til videomateriale, men står også fritt til å velge eksperimentelle oppsett for å teste ut konsept eller undersøke hva som fungerer under varierende forhold.

Oppgaven må løses helt frikoblet fra et påbegynt arbeid med samme case for et år siden. Oppgaven passer en gruppe på 3-4 personer.



C Project Plan

Motion detection in underwater video
Project plan

Benjamin Letnes Bjerken Daniel Hao Huynh
Lars Blütecher Holter Lillian Alice Wangerud

January 25, 2023

CONTENTS

Contents	2
List of Figures	2
List of Tables	3
Abbreviations	5
1 Project Goal	1
1.1 Background	1
1.2 Project goals	2
1.2.1 Result goals	2
1.2.2 Impact goals	3
1.2.3 Learning objectives	3
2 Scope	5
2.1 Subject Area	5
2.2 Delimitation	5
2.3 Task Description	6
3 Planning	7
4 Project Organization	9
4.1 Responsibilities and roles	9
4.2 Routines and rules for the group	10
5 Organization of Quality Assurance	11
5.1 Plan for documentation, configuration management, tool use and source code	11
5.1.1 Documentation	11
5.1.2 Standards and Conventions	11
5.1.3 Configuration Management	12
5.1.4 Testing	13
5.2 Early Risk Analysis (identify, analyze, measures, follow up)	13

<i>CONTENTS</i>	2
6 Progress Plan for Implementation	17
6.1 Gantt-schema for the project period	17
6.2 First Draft of product backlog	18
7 Preliminary Sketch of Development Environment and Technologies	19
8 Relevant Bachelor Theses and Future Development	21
References	22

LIST OF FIGURES

1.1.1 A lot of activity. 1
1.1.2 No activity. 1
1.2.1 Video processing 2
6.1.1 Gantt-schema 17

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations:

- **NTNU** Norwegian University of Science and Technology
- **NINA** Norwegian Institute for Natural Research (Norsk institutt for naturforskning)
- **UI/GUI** User Interface/ Graphical User Interface
- **UX** User Experience
- **CV** Computer Vision
- **GPU** Graphics Processing Unit
- **CPU** Computer Processing Unit
- **IDE** Integrated Development Environment
- **CI** Continuous Integration
- **MVP** Minimum Viable Product
- **AI** Artificial Intelligence
- **CNN** Convolutional Neural Network
- **RNN** Recurrent Neural Network
- **GAN** Generative adversarial network

PROJECT GOAL

1.1 Background

Video recording is used increasingly in research to monitor nature and wildlife. This is due to the method being a non-intrusive and passive way to observe behavior, that does not rely on catching and handling animals. The Norwegian Institute for Nature Research (NINA¹) is one of many that utilizes this method in their research. One of these studies performed by NINA is on the behavior of fish in different environments underwater. Filming underwater however brings on a few challenges, both for the biologists and computer engineers. Some of these challenges are; a lot of dead time between life observed, murky waters, algae and objects that create noise in recordings.

As of now NINA are recording a lot of video out in the field that is saved on hard disks without processing the video recorded. This causes a lot of the storage to hold unnecessary video that includes a lot of dead time between when life is observed, which then results in the researchers manually having to look through and process the data for samples. Therefore they need to streamline their method of data processing and optimize their use of storage space. The aim is thus to be able to automate the process of removing excessive data that is not relevant to the institutes' research.



Figure 1.1.1: A lot of activity.

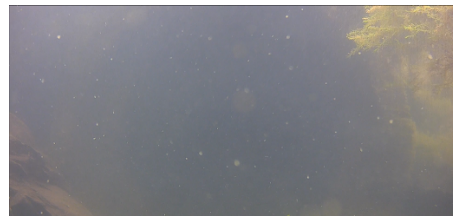


Figure 1.1.2: No activity.

¹NINA - See Abbreviations

1.2 Project goals

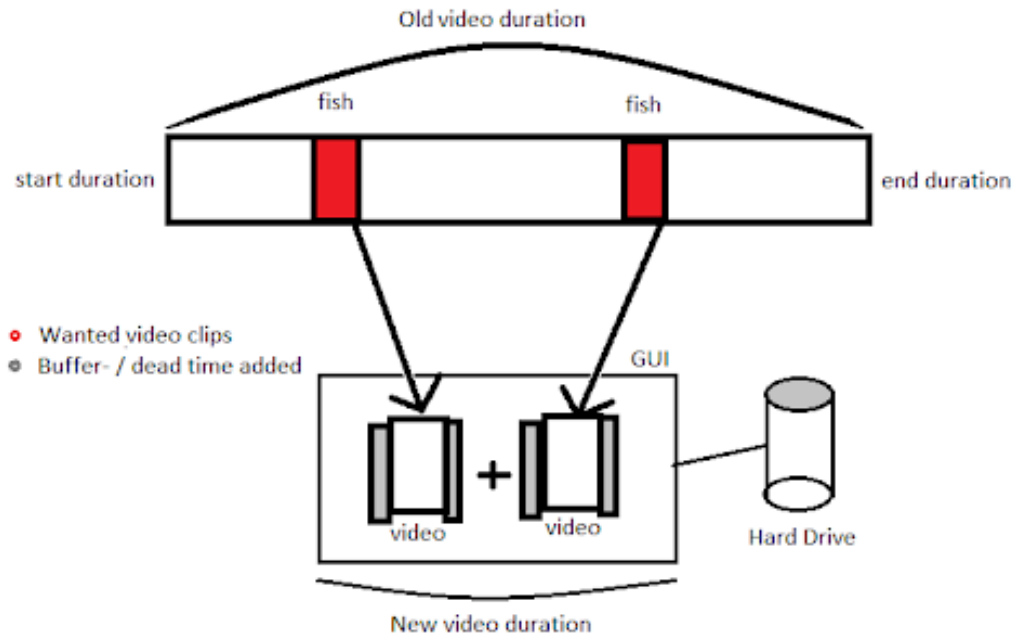


Figure 1.2.1: Diagram illustrating the process of identifying and extracting fish footage from a video and storing it to a hard drive.

Throughout our bachelor thesis we have set a few different goals for what we want to achieve. These goals consist of different performance measures, learning objectives, results as well as personal growth.

1.2.1 Result goals

In our project, we aim to accurately detect 95% of fish in an input video spanning up to 12 hours. We will achieve this by using a binary classification model that uses a combination of precision and recall as the evaluation metrics.

Precision is a measure of how many of the positive predictions made by the model are actually correct. In this case, precision would measure the proportion of correctly identified fish out of all the fish predicted by the model.

Recall is a measure of how many of the actual fish present in the video were correctly identified by the model. In this case, recall would measure the proportion of correctly identified fish out of all the fish present in the video.

To minimize false positives and to ensure that we detect 95% of fish in the video, we will optimize the model to have a high precision and recall score. We will use a formula that calculates the F_1 -score as the harmonic mean of precision and recall. This formula is represented as:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (1.1)$$

We chose to use F_1 -score as the evaluation metric because it takes into account both precision and recall, providing a balance between the two. By maximizing the F_1 -score, we can ensure that the model has a high precision and recall, which will minimize false positives and increase the overall accuracy of fish detection.

It's important to note that we don't care about individual frames, just that the fish is detected at some point because we will include a few seconds buffer time before and after fish detection in the output video. This will allow us to have a more robust detection and not to miss any fish that might be missed due to a specific frame that might not be captured properly.

Another result goal of our project is the processing time for an input video. Specifically, we aim to process a 30-minute video in 25 minutes or less. This is an important aspect of our project as it allows us to quickly analyze and make use of the video data without building up a backlog of unprocessed video as NINA continues to collect new video data.

Another important result goal of our project is to reduce the amount of human time involved in processing a video. Currently, the process of reviewing and analyzing video footage is a time-consuming task that takes 1 minute per minute of video. With our solution we aim to reduce this time to only 5 minutes for a 30-minute video, allowing human operators to focus on more important tasks.

1.2.2 Impact goals

Our impact goals for the project include improving the efficiency of NINA's operations by reducing the time it takes to process video footage. By achieving our result goals of accurate fish detection and faster video processing, we can help NINA to quickly analyze and make use of the video data without building up a backlog of unprocessed video. This will help NINA to more effectively monitor and study fish populations.

Another impact goal is to reduce the amount of hard drives required to store video data. By processing and removing unnecessary footage, we can help NINA to save storage space and reduce the cost of video data storage.

We also aim to focus on future development so that it is easy for others to work on it, maintain it, and add new features. This will make it easier for NINA to continue to improve and update the software as new technologies and methods become available.

1.2.3 Learning objectives

Even though the bachelor thesis is about showing what we have learned thus far in our study, it is also about learning new subjects needed to complete our thesis. These learning objectives are far between each other but all are relevant to becoming a software engineer. The learning objectives we strive to become proficient in throughout the thesis consists of but are not limited to; the complete development life cycle of a software on a commercial project, implementing Artificial Intelligence (AI²) and/ or Computer Vision (CV³), User Experience (UX⁴) and

²AI - See Abbreviations

³CV - See Abbreviations

⁴UX - See Abbreviations

User Interface (UI⁵) , processing of video data and how to connect all aspects we create through coding in different languages.

⁵UI - See Abbreviations

2.1 Subject Area

Our client NINA, has a department on researching fish and other underwater wildlife. They do this through monitoring and recording video with an underwater camera, so as not to disturb the wildlife. This is to ensure that the behavior of the fish does not change.

NINA's current in-field setup consists of several components that save video recorded by an underwater camera. The use of a Matrox box, with a web interface, gathers the video recorded from the camera to save on a hard disk, and allows for a connected computer to stream from the underwater camera. The computer may also connect to 4G through an on-site laptop which allows researchers to remotely turn the camera on and off. As of now the only way for NINA to gain data that has been recorded is to gather that data through fetching the hard disks physically from the field and sort through and analyze the recordings.

The client is therefore requesting an application that uses image recognition and movement detection to automate the process. The information stored on the on-field hard disk would be processed after physically fetching it from the field, using AI or computer vision. This would increase the effectiveness and the quality of the NINA's research.

2.2 Delimitation

In order to keep the bachelor within a doable time frame, we have a few points to delimitation in order to narrow it down. Within our scope we have determined that these elements will not be implemented:

- The finished product will not do real time processing of videos. This is due to the client's limited on field equipment as it seems unlikely that the current setup is suitable to perform the image recognition and data processing and it has limited network access and can therefore not send data across the network to be processed.
- We will not be aiding in the improvement of onsite hardware as this is outside of what is relevant to our bachelor thesis.

- For the resulting application we will not prioritize counting and/or classifying fish. This is due to these falling outside of the planned core software and NINA already has software that can classify fish in recordings which we may integrate with our project.

2.3 Task Description

This project will consist of developing a software that can be used to automate the process of processing recorded video and removing dead time between life is observed. The application needs to be able to run on a Cuda[1] and CudNN[2] supported NVIDIA graphics card. It should be straightforward and researchers with little experience with computers should be able to utilize the program. Features that the application is expected to have is as follows:

- Gather data from an external hard disk
- Process large data sets
- Recognize and determine when fish or other underwater life is seen in a pre-recorded video.
 - Through the use of AI or other movement detection methods
- Process footage by cutting out and removing the clips where wildlife is not within the frame.
 - Save the processed data in a configurable format
- Allow configuration for the processing of the data input(video clips), this refers to; keep or delete the original data, add a customizable amount of dead time before and after the created clip, merge the clips, etc.
 - Everything that is not configurable is likely to be set by NINA prior, such as a filename standard, video length standard, etc.
- Have an intuitive and well formed Graphical User interface that allows for a good user experience with the program.
 - This should allow the user to easily choose the input folder, the output destination and customization to the clips.

PLANNING

For our bachelor thesis we have agreed to use scrum as our development model, this is because scrum fits very well with how we intend to work with our project. When working with modules or interconnected components, we want the workflow of the development model to be iterative instead of incremental in order to work on different modules in parallel. This improves the workflow, especially when we know we can work on different parts of the project asynchronously.

Each Tuesday at around 3 PM we plan to have the weekly scrum meeting where we go through what we were able to do since last week, as well as make plans for the week to come. Here Daniel Hao Huynh is going to act as the scrum master. In the meetings we will go over issues and considering we plan to have our meeting straight after our weekly supervisor meeting, we will discuss what the supervisor advised us on. In these meetings we will also do the sprint reviews.

Each scrum sprint is intended to last for two weeks. We plan to have a daily scrum meeting every weekday to just get everyone up and working on the project, as well as to be able to notify anyone if there are any concerns for what you are currently working on. In these daily meetings we will also make sure everyone is following their tasks and do the work that their roles intend them to do.

To keep track of how far we have come and how the development goes, we will be utilizing git issues. We plan to have medium sized issues with sub issues under them where we can mark what is done and what remains in the issue. Inside our github [3] we have made five different boards where we can change the status of our issues. These statuses include; Todo, In Progress, Ready for review, Stuck and Done. By having these statuses, we can easily see where we are in the development with all aspects of our project and give us a clear indication on what we need to work on next.

Throughout the development process we have several key milestones to work towards. These milestones consist of, but are not limited to; MVP¹, project plan delivery, final plan delivery, user tests and final program iteration. These milestones will be closely worked towards in our github by assigning every issue we

¹MVP - See Abbreviations

create to their respected milestone, allowing us to see how far along we are on each milestone. These milestones are also shown in our Gantt-schema[4].

PROJECT ORGANIZATION

4.1 Responsibilities and roles

- **Group leader:** Lars Blütecher Holter
 - Ensures that every member contributes to the project and that the project follows the project plan
 - Responsible for taking decisions during group disputes
- **Secretary/Document responsible:** Lillian Alice Wangerud
 - Writes reports for meetings
 - Responsible for ensuring that the project is adequately documented
- **Communications:** Benjamin Letnes Bjerken
 - Responsible for keeping communication out of and in to the group
 - Set up meetings between group and client
- **Scrum master:** Daniel Hao Huynh
 - Responsible for summoning all members for scrum meetings
 - Leads scrum meetings and ensures that every thing on the agenda is brought up during the meeting
- **Supervisor:** Marius Pedersen
- **Client representatives:** Knut Marius Myrvold and Tobias Holter

4.2 Routines and rules for the group

The routines and rules for the bachelor group that we have established are as follows:

- We will have a physical meeting with our supervisor every Tuesday at 14 o'clock thereafter we will continue the meeting as a sprint meeting preferably at 15 o'clock.
- Every day, except the weekends we will have normal digital meetings where we will continue our formal work on the project.
- Every other week we will have digital meetings with our client, NINA every monday at 14 until 16 o'clock.
- Every time we work we will log the amount of time used in Toggl [5].
- At any necessity for a Leave of Absence a notification to the group has to be established at least 1 day prior to the date of the absence.
- If the group encounters any type of internal dispute, it will be upon the group leader to take the responsibility for choosing the outcome of the dispute.

ORGANIZATION OF QUALITY ASSURANCE

5.1 Plan for documentation, configuration management, tool use and source code

In order to ensure the quality of the code in the project, it is important to have a well-organized system for quality assurance in place. This includes documentation, standards, configuration management, tools, and source code management.

5.1.1 Documentation

Documentation is a critical aspect of quality assurance, as it helps to clearly communicate the design, implementation, and testing of the code. This documentation will include the requirements, design, test plans, and results. Additionally, any decisions or trade-offs made during the development process will also be documented. To effectively document the python code, we will be using Pydoc [6]. Pydoc is a built-in documentation tool that can automatically generate documentation for the code based on the comments and docstrings. In addition to pydoc, we will be using Jupyter [7] notebooks to document the process of training and testing AI models. Jupyter notebooks allow for interactive computing, data visualization, and narrative text at each step. This will make it easy for others to understand and reproduce the process, and help to ensure that the research is transparent and reproducible. This makes Jupyter an ideal tool for documenting the process of training and testing AI models.

5.1.2 Standards and Conventions

In order to ensure that the python code is readable, maintainable, and consistent, we will be following the PEP8 [8] and PEP257 [9] guidelines. PEP8 is a set of coding conventions for the python programming language which recommends a specific style for formatting and structuring code. It covers various aspects of the code, including indentation, line length, naming conventions, and commenting conventions. PEP257 is similar to PEP8, but it describes a set of docstring conventions. We chose to go with PEP8 instead of something like the Google Python

Style Guide [10] because PEP8 is the python standard and we are already familiar with a lot of aspects from it.

Some key points we will follow from PEP8 include:

- Using 4 spaces for indentation (instead of tabs)
- Use lowercase letters and underscores for variable and function names (snake case)
- Use capitalized words for class names (camel case)
- Use double quotes instead of single quotes for string literals
- Include a space after the # of a comment
- Include docstrings in all functions, classes, and modules

By following the PEP8 guidelines, we can ensure that the code is easy to read and understand for others, and that it is consistent with the established conventions for python code. In addition to following PEP8, we will be using Pylint [11] which is a linting tool that checks the code for PEP8 compliance. This will help us catch any style errors before the code is committed by using pre-commit hooks. We will also be running Pylint as part of our GitHub Continuous Integration pipeline in the form of a GitHub action to catch commits that did not go through the pre-commit hook. The action will run on every push to the master branch and will fail if the code is not PEP8 compliant. Pylint is stricter than something like flake8 and has more checks, stricter coding standards, and more code smells, this gives it an edge over other linters. In addition to Pylint, we will be using the tool Black [12] to format our python code. Black is a python code formatter that automatically reformats code to conform to a consistent style based on the PEP8 guidelines. Black is also very opinionated compared to something like autopep8 which only fixes PEP8 violations which can lead to other inconsistencies in the code.

We will also use commitlint [13] which is a git commit linter that checks the formatting and content of git commit messages before they are committed. This will help ensure that commit messages are clear, concise, and written in the correct format. This will make it easier to understand the history of the code, and to quickly identify which commits made specific changes. We have not used a commit linter before, but we found commitlint to be easy to set up and is well supported with 2 million weekly downloads on NPM and 13,000 stars on GitHub.

5.1.3 Configuration Management

Configuration management will be implemented using a set of best practices and using tools such as Github and Toggl.

Version control will be handled by Github where we will store all versions of the code and track changes made over time. This will allow us to easily roll back to previous versions if necessary and also to collaborate with other team members. We will be using a branching model like git-flow, which allows for the creation of separate branches for development, testing, and production. CI is also done via Github action as mentioned above, this will help us automate the process of

building, testing, and deploying the code. Our branching model and CI pipeline will help ensure that the code is always in a stable state and that changes are thoroughly tested before being deployed to production. Task management is also handled via Github, using their "Project" feature. Github projects allow us to organize and prioritize tasks, assign them to team members and track the progress of the work. How we use boards in the project is described in 4.1.

We will be using Toggle to track the time spent on different tasks, this will help us have a better understanding of how much time is spent on different aspects of the project and make data-driven decisions to improve the project management.

5.1.4 Testing

Testing is a crucial step in ensuring the quality of the code. It helps us to identify and fix bugs, ensure that the code meets the requirements, and improves the overall performance and usability of the code. To implement testing into our project, we will be using a combination of automated unit tests, integration testing, manual testing, and user testing.

We will be writing automated unit tests to test individual units of the code. Integration testing will be performed once we have adequate coverage using unit tests, in order to test how the individual units of code work together. Both unit testing and integration testing will be performed using the Pytest testing framework, and these tests will run as Github Actions on all commits pushed to the repo. This will help to ensure that the code is working as expected before anything is merged into production.

Manual testing will be performed to test the code manually by running it and evaluating its output. This will be performed by us during development.

To perform user testing, we will be recruiting users from NINA and inviting them to participate in in-person moderated usability testing sessions. During these sessions, a moderator will guide the users through the testing process and ask them questions about their experience with the interface. This method is useful for gathering qualitative data about the user experience and providing insights into the user's thoughts and feelings. User testing is essential to ensure that the interface is user-friendly and that the product meets the needs of the users.

By implementing these testing methods, we can ensure that the code is functioning as expected, that it is able to detect fish in the videos, and that it meets the requirements. Additionally, by performing user testing, we can gather feedback on the user interface and the product and use it to improve the user experience. Overall, testing helps to guarantee the reliability, stability, and usability of the code and to avoid issues that can arise in the future.

5.2 Early Risk Analysis (identify, analyze, measures, follow up)

No.	Area of Assignment	Description	Possible consequences	Risk			Mitigation
				Pro	Imp	Risk index	
1	Product	Lack of training data	The model may not be able to detect fish accurately	2	4	6	Gather datasets from external sources. Spend adequate time labeling. Use data augmentation. Create synthetic data.
2	Product	Loss of source code on local machine	Non committed code is lost and has to be re-written	3	1	4	Use an external host for the git repository. Commit often.
3	Product	Loss of training data	We have to spend time re-collecting the data	1	4	5	Use an external host to store training data as well as local copies.
4	Product	Poor tracking of fish(?)	We are unable to accurately count fish and produce bad reports	4	1	5	Allocate time for researching and testing accurate tracking algorithms.
5	Product	Inadequate graphics processing unit for deployment	Unsupported GPU leads to no deployment.	1	5	6	Change to a supported GPU.
6	Product	Inadequate graphics processing unit during training	We are unable to train the model in a reasonable amount of time	1	4	5	Choose a smaller model.
7	Product	Complex/non-intuitive user interface	Users are unable to use the application	2	4	6	Allocate time for user testing and focus on simplicity.
8	Product	Product not finished by deadline	We are unable to deliver a product that satisfies client requirements	2	3	5	Focus on creating a minimum viable product before any extra features. Prioritize more important features over others. (write more about following gantt chart and good use of issues on github)

No.	Area of Assignment	Description	Possible consequences	Risk			Mitigation
				Pro	Imp	Risk index	
9	Product	Model size too small or too large/expensive	The model is unable to detect fish accurately, or we might be unable to process the videos in real time or less time than the length of the input video	1	5	6	Allocate time for research and testing of models that satisfy our accuracy requirements while being performant enough.
10	Project	Lack of documentation	Hard to maintain and efficiently perform collaborative work on the project	2	3	5	Dedicated team member to ensure adequate and proper documentation. (write about charts, schemas, architecture, source code comments, atomic commits and commit messages below)
11	Business	Wrong choice of detection method	Waste of time and resources	2	4	6	Allocate time for research into both traditional movement detection algorithms and different object detection methods using neural networks.
12	Business	Report not finished by deadline	We get a worse grade	2	5	7	Prioritize the report over the product. Work on the report in parallel with the product. Document all decisions as we make them. Set intermediate deadlines and milestones.
13	Business	Monolithic architecture	Hard to maintain and extend with new features and future models	1	3	4	Focus on modularity

ID	Mitigation
1	<p>Considering what our project is regarding, there is most likely not that many datasets we can use that are ready to be used. Thus to prevent this from being an issue, we will gather datasets from external sources. In addition to this, we will spend adequate time labeling the data to make sure we get the most accurate results as possible. To get more data to run our model on, we will use data augmentation and if needed and not too big of a challenge, we will try to add some synthetic data as well.</p>
5	<p>As the final part of our delivery for NINA is to deploy the product on the commissioned hardware, we should ensure that the hardware fits all the required specifications for our product. Therefore the Graphics Processing Unit (GPU) has to be up to date with our product. It should support Cuda and CuDNN in order to work, and if it does not, then we have to get them to switch the unsupported component to a supported one.</p>
7	<p>To ensure that the model is neither too small to perform the required tasks nor too sizable and costly we plan on allocating time to research and test models. This will make sure that the final model utilized for our projects will satisfy the accuracy requirements we have put on the product, such that it will accumulate as many clips with necessary data for NINA's research. However during this process we also need to ensure that the model is not too costly to process data on our client's equipment and can process it in less than real time, as the goal of this project is to streamline the processing of NINA's data.</p>
9	<p>To mitigate the risk of choosing the wrong detection method, we will conduct thorough research, create prototypes, review prior research, and continuously evaluate the performance of the chosen method. We will compare traditional movement detection algorithms and different object detection methods using neural networks, test them on a small scale and make an informed decision on which method to use. This will help to ensure that we are using the best method for our project and that we are using our time and resources effectively.</p>
10	<p>For our project to be used by the scientists at NINA, we will need to have a working UI which is intuitive to use. Even though our program won't have too many different elements embedded into it, it's still important to test how it is for the end user and if there are anything missing. Hence we will make sure to allocate time for user testing and focus on simplicity.</p>
12	<p>To mitigate the risk of not finishing the final report, we will prioritize it over the end product, work on the report in parallel with the product, document all the decisions as we make them, and set intermediate deadlines and milestones. We will create a detailed project plan and communicate regularly with the supervisor to ensure that we are on schedule to meet the deadline. We will also have a backup plan in case we face unforeseen obstacles that might delay the report, such as seeking extra help or adjusting the scope of the report if necessary. Additionally, we will review the report on a regular basis to ensure that it is of high quality and includes all the necessary information. By taking these steps, we ensure that the report is completed on time and to a high standard.</p>

PROGRESS PLAN FOR IMPLEMENTATION

6.1 Gantt-schema for the project period

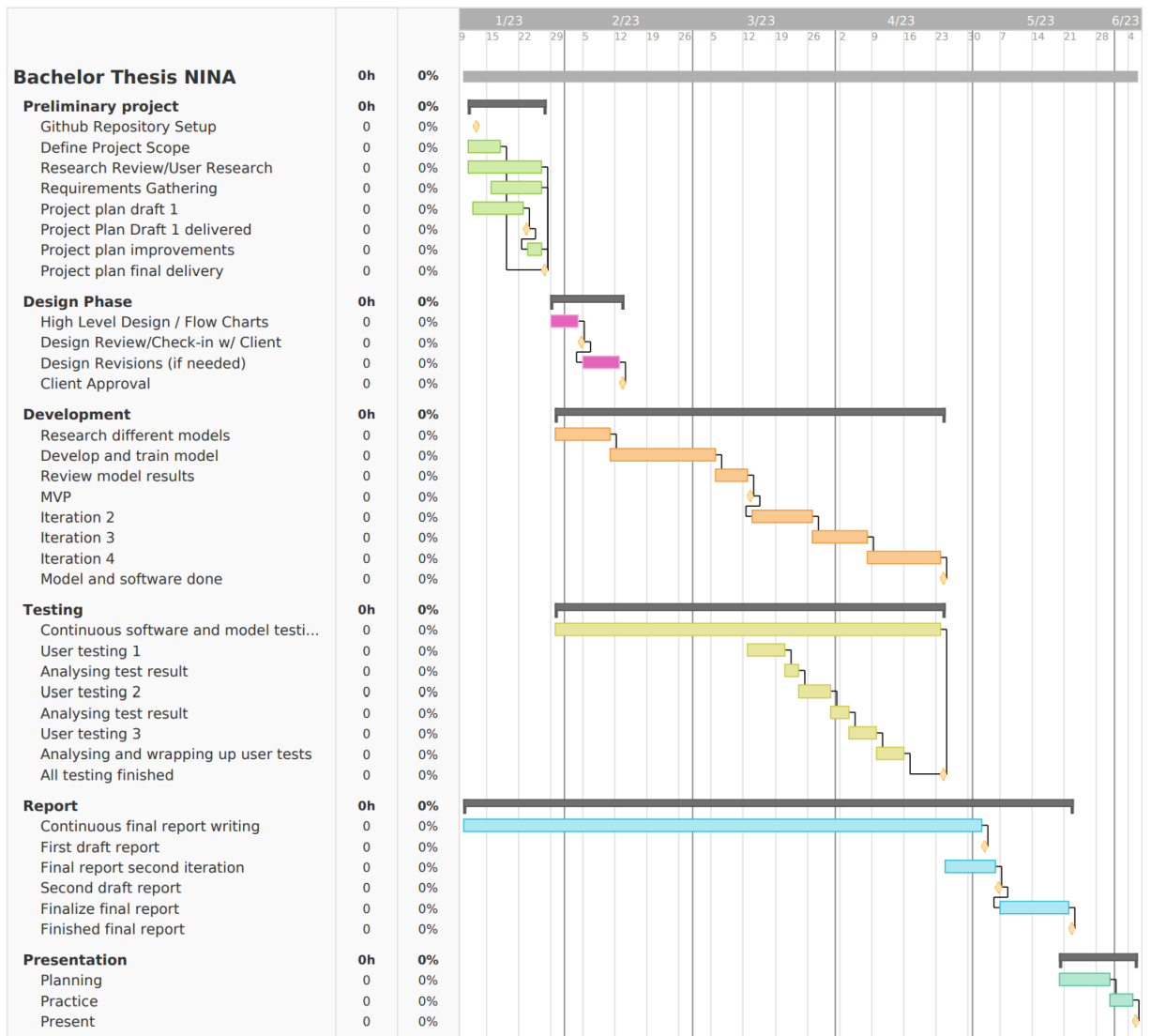


Figure 6.1.1: A Gantt-schema for the approximation of the project period.

6.2 First Draft of product backlog

The estimated time references to estimated hours spent on developing the feature

- Long - 20h or more
- Medium - 10h -20h
- Short - 10h or less

ID	Feature	Priority	Estimated Time
1	Detection of wildlife activity	High	Long
2	Process Videos/data	High	Medium
3	Setup dataset	High	Long
4	System Architecture	High	Medium
5	Save Videos/data	Medium	Short
6	Draft UI	Medium	Short
7	Fail safe mechanics	Medium	Long
8	Generate report after processing	Low	Short
9	Sorting of processed clips	Low	Short
10	Ability to pause and stop processing	Low	Medium

PRELIMINARY SKETCH OF DEVELOPMENT ENVIRONMENT AND TECHNOLOGIES

As we drafted a preliminary sketch for the development environment and discussed which technologies we would use for the project, we concluded that python would be a key programming language for the entirety of the project, this is because of how highly supported the language is by libraries, and how well documented it is. For drafting the AI model we used ChatGPT [14] for advice, this is because of its high versatility and speed regarding giving reasonable responses and documentation of said models, this information was used in tandem with physical and digital meetings in order to figure out which model fits the project's scope best.

We decided that we would use a standard IDE which supported python for our project such as Visual Studio Code[15] or PyCharm[16]. We would also include the use of black & flake8[17] as a code formatter and linter with said IDE, this is because of their standard of linting being very adamant and giving us a good foundation in keeping all of the code format similar to each other.

For developing the AI, we would definitely use the Tensorflow[18] which supports the usage of GPU instead of just CPU processing, this is because using just the regular tensorflow would bottleneck our progress severely. The performance of said tensorflow is far behind the tensorflow which uses the GPU in order to train the AI. Tensorflow GPU is therefore a must have for us especially when working with training the provided and other datasets in general. Tensorflow GPU requires that the physical GPU is of the NVIDIA brand and that the prerequisite cuDNN and Cuda software be installed.

The UI has been drafted to be made using python, for the same reasons stated previously, the libraries PyQt[19] or Tkinter[20] would be viable for use in development of the UI. PyQt is built on industry standard Qt and would be viable because of the different functionalities they provide. Qt uses a broad variety of native platform APIs for networking, database development, and more. It provides primary access to them through a special API. As PyQt is one of the most commonly used UI systems for Python, you can conveniently access a broad variety of documentation. Qt provides multiple widgets, such as buttons or menus, all designed with a basic interface for all compatible platforms. GUI programming

with Qt is built around the idea of signals and slots for creating contact between objects. This allows versatility in dealing with GUI incidents which results in a smoother code base.

Tkinter would be used in tandem with its auxiliary library CustomTkinter[21] for added functionality, such as a unified style, and easier implementation of a stylized GUI. This decision was based on the documentation they provided, and seeing examples of what kind of applications which could be achieved using said libraries. The libraries are also well documented, Tkinter has been documented thoroughly, seeing as it is the standard Python interface to the Tcl/Tk GUI toolkit. CustomTkinter is a python UI-library based on Tkinter, which provides new, modern and fully customizable widgets. They are created and used like normal Tkinter widgets and can also be used in combination with normal Tkinter elements. Surprisingly even though CustomTkinter is a newer library it is well supported and documented.

We would use a Library supporting ffmpeg[22] in order to process the videos so that they only contain data of value to our Client(NINA), this usually means fish of any kind however it can also be just wild life activity in general. We would have the program depend on our Model in order to decide which parts are necessary and cut out the unnecessary parts, this would also be done presumably using python with a ffmpeg library supporting the functionality which we need.

During the development there will be a need for testing different functionalities and mocking certain values, for testing pure python code, the inbuilt unit testing library or the much preferred Pytest library would be used during the development for that exact purpose, whereas UI would probably use something like PyAutoGUI[23].

We use Discord [24] as a primary technology for the use of communication, notes, resources and in general, it is likely going to reflect the development process later on as it will contain all resources related to physical and digital meetings and communication regarding the project itself, However we communicate using email/teams in order to communicate with our supervisor and client(NINA).

The development process itself and technologies used during that process would all be tracked using Toggl, a standalone website used for tracking the time usage for any kind of work supporting group tracking and labeling, and Git issues for the implementation of the product itself. Here we use GitHub for that with linted commit messages acting concurrently with the issues regarding the commits.

RELEVANT BACHELOR THESES AND FUTURE DEVELOPMENT

We are currently working on a bachelor thesis that aims to make the researchers' lives easier and acquire more data in a more efficient way. In order to achieve this goal, we are considering incorporating previous NINA bachelor theses that have similar aims and objectives. By doing so, we hope to create a more comprehensive and end-to-end system that would satisfy NINA's needs for a complete software solution.

One of the relevant bachelor theses that we will be studying is the "Artsgjenkjenning av fisk" [25] which focuses on species recognition of fish. This thesis worked on how to report which species of fish was recorded at a given time. Although it is not the same thesis as ours, it is closely related in terms of the goals and objectives. Our thesis, on the other hand, aims to cut out irrelevant video footage. By integrating parts of this previous project into our own, we hope to create a more end-to-end system that would make the researchers' lives easier and acquire more data in a more efficient way.

To ensure that our project is easily readable, maintainable and testable, we plan to work in a modular fashion. This means that we will break down the code into smaller, manageable components that are easier to understand and work on. Additionally, we will document our work throughout the whole process, so that anyone who picks up the project later on will be able to understand the reasoning behind every decision we made along the way.

Another relevant thesis we will study is the "Salamander Identification Application." [26] which is about identifying salamander species based on the patterns on their stomachs through the use of picture recognition and AI. Even though this thesis regards salamanders and creating an app for this, it still has many similarities with our bachelor thesis and we will use the results from this previous thesis to enhance our own project. By studying and incorporating the findings from these previous theses, we hope to create a more comprehensive and effective solution for NINA.

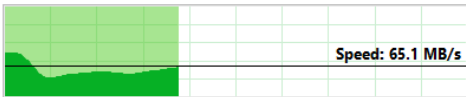
REFERENCES

- [1] NVIDIA. *CUDA Toolkit Documentation v12.0*. Documentation. URL: <https://docs.nvidia.com/cuda/>.
- [2] NVIDIA. *NVIDIA CUDNN DOCUMENTATION*. Documentation. URL: <https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html>.
- [3] *GitHub*. Web Page. 2020. URL: <https://github.com/>.
- [4] John Correlli and Nathan Gilmore. *teamgantt*. Web Page. 2009. URL: <https://support.teamgantt.com/article/77-welcome-to-teamgantt/>.
- [5] Alari Aho and Krister Haav. *Toggl Track*. Web Page. 2006. URL: <https://developers.track.toggl.com/docs/>.
- [6] Python Software Foundation. *pydoc – Documentation generator and online help system*. Documentation. 2023. URL: <https://docs.python.org/3/library/pydoc.html>.
- [7] Project Jupyter. *Jupyter*. Documentation. 2014. URL: <https://jupyter.org/>.
- [8] Guido van Rossum, Barry Warsaw, and Nick Coghlan. *PEP 8 – Style Guide for Python Code*. Documentation. 2001. URL: <https://peps.python.org/pep-0008/>.
- [9] David Goodger and Guido van Rossum. *PEP 257 – Docstring Conventions*. Documentation. 2001. URL: <https://peps.python.org/pep-0257/>.
- [10] *Google Python Style Guide*. Documentation. URL: <https://google.github.io/styleguide/pyguide.html>.
- [11] *Pylint*. Documentation. URL: <https://pylint.readthedocs.io/en/latest/>.
- [12] Łukasz Langa et al. *Black*. Documentation. 2011. URL: <https://black.readthedocs.io/en/stable/>.
- [13] marionebl. *commitlint*. Documentation. 2020. URL: <https://commitlint.js.org/%5C#/>.
- [14] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2019).
- [15] Microsoft. *Visual Studio Code*. Computer Program. 2015. URL: <https://code.visualstudio.com/docs>.

- [16] JetBrains. *PyCharm*. Computer Program. 2010. URL: <https://www.jetbrains.com/pycharm/guide/tips/quick-docs/>.
- [17] Ian Stapleton Cordasco. *Flake8: Your Tool For Style Guide Enforcement*. Documentation. 2016. URL: <https://flake8.pycqa.org/en/latest/>.
- [18] Martián Abadi et al. *Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Web Page. 2015. URL: <https://www.tensorflow.org/>.
- [19] Riverbank_Computing. *PyQt*. Documentation. 1998. URL: <https://wiki.python.org/moin/PyQt>.
- [20] Python Software Foundation. *tkinter – Python interface to Tcl/Tk*. Documentation. 2023. URL: <https://docs.python.org/3/library/tkinter.html>.
- [21] Tom Schimansky. *CustomTkinter*. Documentation. 2021. URL: <https://github.com/TomSchimansky/CustomTkinter>.
- [22] Karl Kroening. *ffmpeg-python: Python bindings for FFmpeg*. Documentation. 2017. URL: <https://kkroening.github.io/ffmpeg-python/>.
- [23] Al Sweigart. *PyAutoGUI*. Documentation. 2014. URL: <https://pyautogui.readthedocs.io/en/latest/>.
- [24] Discord Inc. *Discord*. Computer Program. 2015. URL: <https://discord.com/developers/docs/intro>.
- [25] Birger Johan Nordølum et al. “Artsgjennkjenning av fisk”. Thesis. 2021.
- [26] Eirik Martin Danielsen et al. “Salamander Identification Application”. Thesis. 2021. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2777982>.

D Prototype

37% complete



Name: Barefoot 2014.mp4
Time remaining: About 10 seconds
Items remaining: 1 (435 MB)

— □ ×

Folder to analyse

Browse files

Folder to store new files

Browse files

Buffer time before Buffer time after

Keep original

Advanced options

Run

E User Test Form

1. Innledning

- Forklar formålet med brukertesting.
- Beskriv kort applikasjonen og dens funksjoner.

2. Spørsmål før testen

- Spør deltakerne om deres bakgrunn og erfaring.
- Spør dem hva de forventer at applikasjonen skal gjøre.
- Spør om de har noen spesifikke bekymringer eller spørsmål om applikasjonen.

3. Testoppgaver

- Be deltakeren om å fullføre en rekke oppgaver ved hjelp av applikasjonen. Se nedenfor.
- Observer deltakeren mens de fullfører oppgavene.
- Be dem om å tenke høyt og forklare tankeprosessen mens de arbeider med oppgavene.
- Oppfordre dem til å stille spørsmål eller gi tilbakemeldinger underveis.

4. Spørsmål etter testen

- Spør deltakeren om deres generelle erfaring med å bruke applikasjonen.
- Spør om de har støtt på problemer eller hatt problemer med å fullføre noen av oppgavene.
- Spør om de syntes noen av funksjonene var forvirrende eller vanskelige å bruke.
- Spør om de har noen forslag til forbedringer av applikasjonen eller dens funksjoner.
- Takk deltakeren for deres tid og tilbakemeldinger.

5. Avslutning

- Minn deltakeren på at tilbakemeldingene deres er verdifulle og vil bli brukt til å forbedre applikasjonen.
- Gi dem eventuell tilleggsinformasjon eller ressurser de måtte trenge.
- Takk dem igjen for deres tid og deltakelse.

Oppgave 1:

Du har en mappe som inneholder flere videoer. Bruk programmet til å behandle videoene og lagre de behandlede videoene i en ny mappe. Forsikre deg om at de behandlede videoene bare inneholder deler av videoen der ferskvannsfisk oppdages. Aktiver også rapportfunksjonen og lagre rapporten i CSV-format.

Oppgave 2:

Du har en mappe som inneholder en video. Bruk programmet til å behandle videoen og lagre den behandlede videoen i en ny mappe. Forsikre deg om at den behandlede videoen bare inneholder deler av videoen der ferskvannsfisk er oppdaget. Deaktiver også rapportfunksjonen.

Oppgave 3:

Du har en mappe som inneholder flere videoer. Bruk programmet til å behandle videoene og lagre de behandlede videoene i en ny mappe. Forsikre deg om at de behandlede videoene inneholder 10 sekunder med video før og etter en snutten som inneholder fisk.

Oppgave 4:

Du har en mappe som inneholder flere videoer. Bruk programmet til å behandle videoene og lagre de behandlede videoene i en ny mappe. Forsikre deg om at de behandlede videoene inneholder 0 sekunder med video før og etter en rekke bilder som inneholder ferskvannsfisk.

Oppgave 5:

Du har en mappe som inneholder flere videoer. Bruk programmet til å behandle videoene og lagre de behandlede videoene i en ny mappe. Forsikre deg om at de behandlede videoene inneholder den opprinnelige input videoen og 5 sekunder med video før og etter en rekke bilder som inneholder ferskvannsfisk.

Oppgave 6:

Utforsk programmets "advanced options" og aktiver PDF-rapportformatet. Bearbeid deretter en video, og lagre den bearbeidede videoen i en ny mappe. Forsikre deg om at den behandlede videoen bare inneholder deler av videoen der fisk oppdages.

Oppgave 7:

Utforsk programmets avanserte alternativer og deaktivert rapportfunksjonen. Behandle deretter en video, og lagre den behandlede videoen i en ny mappe. Forsikre deg om at den behandlede videoen bare inneholder deler av videoen der ferskvannsfisk er oppdaget. Se om en rapport ble generert.

Oppgave 8:

Utforsk programmets avanserte alternativer og aktiver XML-rapportformatet. Behandle deretter en video av ferskvannsfisk som svømmer i en elv eller bekk, og lagre den behandlede videoen i en ny mappe. Forsikre deg om at den behandlede videoen bare inneholder deler av videoen der ferskvannsfisk oppdages.

Oppgave 9:

Utforsk programmets avanserte alternativer og deaktivert rapportfunksjonen. Behandle deretter en video av ferskvannsfisk som svømmer i en elv eller bekk, og lagre den behandlede videoen i en ny mappe. Forsikre deg om at den behandlede videoen bare inneholder deler av videoen der ferskvannsfisk er oppdaget.

Oppgave 10:

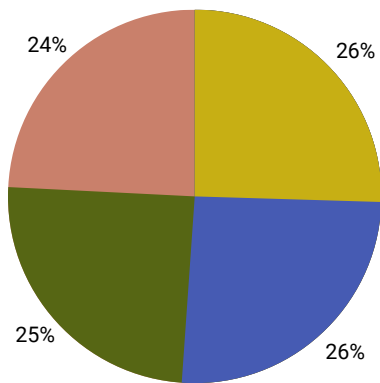
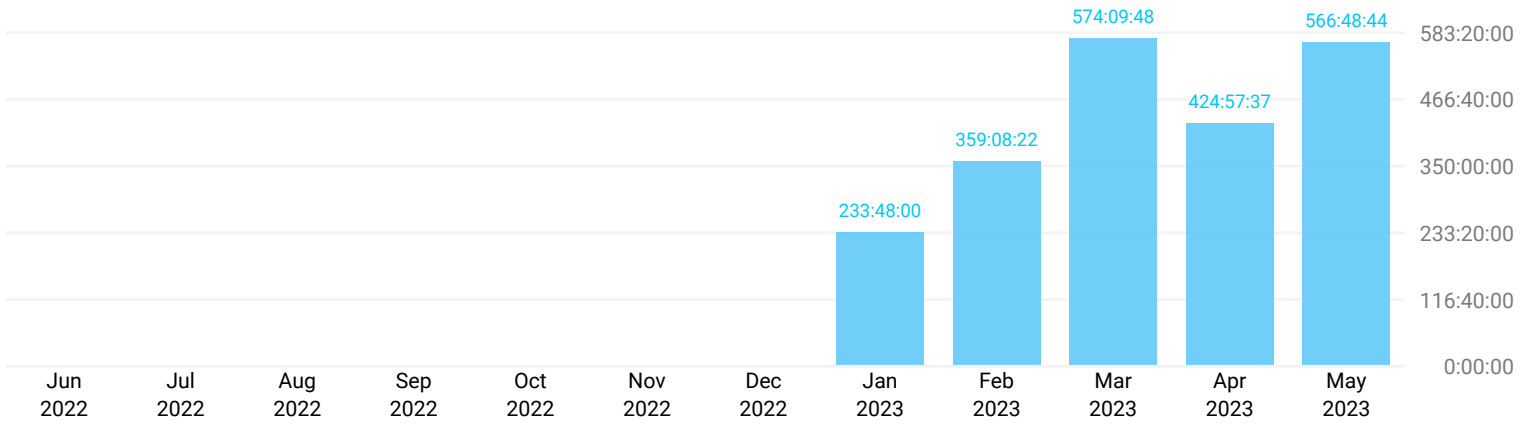
Bruk programmet til å behandle en video av ferskvannsfisk som svømmer i en elv eller bekk, og lagre den behandlede videoen i en ny mappe. Se på fremdriftslinjen og legg merke til hvor lang tid det tar å behandle videoen. Gi deretter tilbakemelding på programmets hastighet.

F Timekeeping

Summary Report

06/01/2022 – 05/31/2023

TOTAL HOURS: 2158:52:31

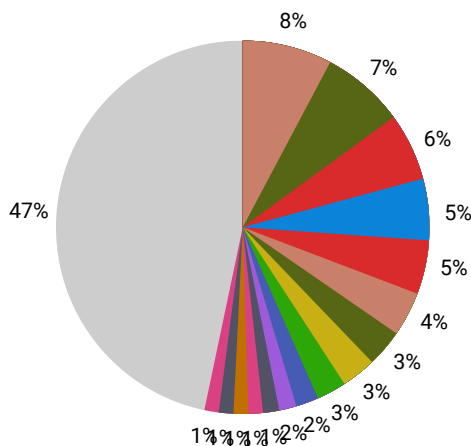


USER

- LB Lars Blütecher Holter
- Daniel
- Bzh Bzh
- LA Lillian Alice Wangerud

DURATION

- 551:19:55
- 551:16:19
- 532:09:15
- 524:07:02



TIME ENTRY

- Final report writing
- documentation
- Daily meeting
- Annotation
- Documentation
- Report writing
- annotation work
- Annotation work
- Supervisor meeting
- First iteration UI
- Work on project plan
- Research Previous bachelor assignment for RND
- Scrum meeting
- Daily scrum meeting
- R&D for output report from the program
- Project plan writing

DURATION

- 167:23:39
- 156:00:04
- 125:15:04
- 116:13:19
- 99:31:08
- 82:18:00
- 69:34:53
- 64:34:01
- 55:21:02
- 41:37:10
- 34:20:46
- 30:33:47
- 27:30:45
- 27:10:03
- 26:28:58
- 25:56:13

USER - TIME ENTRY	DURATION	PERCENTAGE
Bzh	532:09:15	24.65%
Add CI badge to repo	0:06:00	
Add functionality to new UI widgets	2:11:03	0.1%
Add more videos to evaluation script	0:47:37	0.04%
Allocate disk space for CVAT	3:04:50	0.14%
Annotation	19:55:34	0.92%
Automate brackish dataset generation	7:10:53	0.33%
Clean up video output annotation code	0:34:00	0.03%
Configure and start training Yolov8s model	3:16:30	0.15%
Configure CVAT server	4:41:39	0.22%
Configure self-annotation with yolov5x and create script to turn --save-txt output to be cvat compliant	4:14:00	0.2%
Convert dataset to binary classification	5:28:00	0.25%
Create evaluation script and create evaluation data	5:04:00	0.23%
Create new training split with new background images and start training	4:09:00	0.19%
Create new vdi test with new frame grabber	1:00:00	0.05%
Create PR and validate that everything works on linux	1:15:00	0.06%

USER - TIME ENTRY	DURATION	PERCENTAGE
Create PRs for settings rewrite and pytest CI	0:29:17	0.02%
Create script for generating dataset	10:00:14	0.46%
Create script to find fish distribution in un-annotated videos	5:01:00	0.23%
Create UI test template	1:30:00	0.07%
Daily meeting	42:01:26	1.95%
Daily meeting and discuss eval script	2:20:00	0.11%
Deploy model in cvat for automatic annotation	2:46:00	0.13%
Determine project name	0:25:00	0.02%
Diagram	0:37:00	0.03%
Discuss settings module	0:41:00	0.03%
Download and process Open Images Dataset V7	2:35:00	0.12%
Experiment with CVAT exports	4:00:00	0.19%
Export nina yolo dataset with images	0:39:24	0.03%
Finalize new output annotator	3:30:00	0.16%
Finalize ui-backend link	6:46:06	0.31%
Find already annotated videos	7:09:53	0.33%
Find annotation distribution in yolo dataset	0:30:00	0.02%

USER - TIME ENTRY	DURATION	PERCENTAGE
Find duplicate images in coco and yolo datasets	4:14:53	0.2%
Finish VDI test setup	5:00:17	0.23%
Fix and clean frame grabber among other small bugs and improvements	2:15:00	0.1%
Fix and convert brackish training data	6:25:36	0.3%
Fix and improve docker container for production env	3:14:29	0.15%
Fix and merge Francesco PR	1:15:00	0.06%
Fix annotation guide and scripts	2:15:00	0.1%
Fix CVAT and finish annotation	4:36:29	0.21%
Fix CVAT issues	2:30:00	0.12%
Fix LFS	0:54:20	0.04%
Fix linting and normalization issues in dataset generator	0:24:00	0.02%
Fix merge conflicts, do fixes, cleanup and features from user testing meeting	6:15:00	0.29%
Fix normalization issues and generate full dataset	3:03:36	0.14%
Fix output video annotation	3:35:00	0.17%
Fix poetry installation issues with torch and experiment with pre-commit.ci	5:11:40	0.24%
Fix pylint GitHub action	1:10:00	0.05%
Fix the annotation guide	0:44:00	0.03%

USER - TIME ENTRY	DURATION	PERCENTAGE
Fix Ubuntu CI	2:06:05	0.1%
Fix yolov8 segfaults	2:00:40	0.09%
Frame grabber fixes and UI stuff	4:18:00	0.2%
Gather more background images	3:44:11	0.17%
Gather relevant background images and re-generate dataset	1:15:00	0.06%
Help others	7:22:01	0.34%
Help others with setup	1:44:07	0.08%
Implement video annotation for processed videos	2:15:33	0.1%
Implement video cutting and processing	5:20:04	0.25%
Improve evaluation script	6:40:48	0.31%
Improve image fetching and batch processing	3:23:13	0.16%
Improve video loading performance	4:30:00	0.21%
Improve yolo dataset annotations	5:29:41	0.25%
Link UI and backend	3:38:15	0.17%
Make CI actions run on PR edits	0:07:14	0.01%
Meeting about project plan draft	3:01:00	0.14%
Merge before user testing	0:59:07	0.05%


USER - TIME ENTRY	DURATION	PERCENTAGE
Merge francesco PR and research better ways of batching images	3:30:00	0.16%
NINA meeting	4:09:53	0.19%
NINA meeting with IT	0:34:00	0.03%
Perform full export of new dataset(s) and find annotation distribution	1:22:00	0.06%
Planning	1:30:00	0.07%
Plotting	0:48:49	0.04%
Prepare for user testing	5:54:48	0.27%
Prepare usertesting	1:00:00	0.05%
Process "artsgjenkjenning" dataset	6:47:54	0.31%
Process annotations in unannotated videos	2:45:00	0.13%
Process detection data	2:30:00	0.12%
Process NINA yolo dataset	0:51:00	0.04%
PROG2009 Seminar - Crash course on Project management	1:45:00	0.08%
Project plan conversion to LaTeX	8:15:36	0.38%
Project plan draft 2	2:26:10	0.11%
Project plan writing	3:46:00	0.17%
Push detection changes for other work	0:56:00	0.04%

USER - TIME ENTRY	DURATION	PERCENTAGE
Read report and note feedback and TODOs and read other reports	3:45:00	0.17%
Reconfigure local disks for datasets	3:37:54	0.17%
Refactor checkbox and other misc stuff	2:06:00	0.1%
Refactoring	2:59:00	0.14%
Report writing	82:18:00	3.81%
Research AI models	1:57:00	0.09%
Research on pipreqs and pigar	1:05:00	0.05%
Rewrite settings	4:29:24	0.21%
Scrum meeting	4:30:18	0.21%
Scrum Meeting	2:14:35	0.1%
Set up commitlint locally and in CI	1:02:00	0.05%
Set up CVAT	5:33:00	0.26%
Set up isort	0:33:07	0.03%
Set up poetry	2:09:42	0.1%
Set up pre-commit for local linting of code and commit messages	1:48:00	0.08%
Set up sharing of external drive	1:03:12	0.05%
Set up test CI	1:28:00	0.07%

USER - TIME ENTRY	DURATION	PERCENTAGE
Set up training on school computer	2:10:00	0.1%
Set up training on school computer and fix issues	1:52:56	0.09%
Set up VDI test	0:29:21	0.02%
Set up Yolo v5 for training with nina dataset	0:42:50	0.03%
Show progress status of output video in UI	1:32:02	0.07%
Supervisor meeting	14:24:36	0.67%
Supervisor Meeting	1:00:00	0.05%
Tensorflow YOLOv5 integration	9:46:07	0.45%
Training configuration	5:00:00	0.23%
Tweak dataset generation script	3:00:00	0.14%
UI implementations	1:47:23	0.08%
Update and fix evaluation script	0:46:00	0.04%
Update and fix WSL/Docker install and start training	3:30:00	0.16%
Update yolo annotation distribution	0:15:00	0.01%
Upload training data and schedule weekly meetings	0:25:00	0.02%
User testing	2:24:00	0.11%
Various fixes and improvements	16:21:00	0.76%

USER - TIME ENTRY
DURATION
PERCENTAGE

Verify Dataset Integrity	5:45:00	0.27%
Work on project plan	5:24:00	0.25%
Work on Yolov8 integration	1:20:37	0.06%
Without description	17:43:16	0.82%

 Daniel	551:16:19	25.54%
Add formatting to technologies	0:45:00	0.03%
Add models, fix small issue with recall	4:15:00	0.2%
After meeting summary and information exchange for reference group	1:35:14	0.07%
Annotation work	64:34:01	2.99%
Approve pull requests	0:30:00	0.02%
Begun second draft on Project Plan	2:15:00	0.1%
Crash course with FRODE	2:00:00	0.09%
Create data for weights	0:30:00	0.02%
Daily meeting	49:06:46	2.27%
Daily meeting + Communication with Jana	1:31:04	0.07%
Data annotation research	3:15:00	0.15%
Documentation	81:58:20	3.8%

USER - TIME ENTRY

DURATION

PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Documentation organization	3:00:00	0.14%
Documentation organization for old models	1:30:00	0.07%
Documentation related to first draft of final report	2:04:48	0.1%
Documented for Annotations	2:23:58	0.11%
Documented merged and pushed base implementation of CustomTkinter variant	2:15:00	0.1%
Documented models	4:30:00	0.21%
Export models as PDF and organize in overleaf	1:45:00	0.08%
Export to Overleaf	1:15:00	0.06%
Extracted and changed data	2:00:00	0.09%
Finalized Overleaf Project Plan	3:00:00	0.14%
Finalizing Project Plan for first draft	10:39:17	0.49%
Fish range ground truth	4:49:01	0.22%
Fix COCO dataset	14:06:34	0.65%
Fix latex PDF figure issues, import all text into overleaf and fix problems that occurred when going from PNG to PDF	2:15:00	0.1%
Fix local poetry issue	1:00:00	0.05%
Fix local python issue	1:33:00	0.07%
Fix models for Quality assurance	1:30:00	0.07%

USER - TIME ENTRY	DURATION	PERCENTAGE
fix small bugs	1:00:00	0.05%
Fix some old models for documentation	0:45:00	0.03%
Fix UI bug	8:20:27	0.39%
Fixed a script's format to consider pylint and black	1:58:17	0.09%
Fixed abbreviations and glossaries	2:00:00	0.09%
Fixed Operating system instability	1:30:00	0.07%
Fixing the COCO formatted datasets	7:38:44	0.35%
Get up-to-date to current project progress	1:12:03	0.06%
Got data from videos	2:15:00	0.1%
Got numbers from evaluation	2:02:38	0.09%
Hardware Research for Deployment	1:37:57	0.08%
Help finalizing ui backend link	1:45:00	0.08%
Help other group members	4:30:00	0.21%
Help teach usage of source tree and GIT to group member	1:15:00	0.06%
Help with tool diagram	1:30:00	0.07%
Helped with ffmpeg report manager	1:34:26	0.07%
Helping group member with CUDA/CUDNN	1:32:56	0.07%

USER - TIME ENTRY

DURATION

PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Installed various requirements for AI on multiple devices	3:33:32	0.16%
Integrated UI fix into unified branch	0:51:01	0.04%
IT meeting	1:00:00	0.05%
IT meeting kick-off	1:14:01	0.06%
Kick-off meeting with supervisor	1:15:00	0.06%
Latex library issue fix	1:15:00	0.06%
Latex RND	4:25:01	0.2%
Making models for Quality Assurance	2:53:14	0.13%
Making more models for report	1:00:00	0.05%
Meeting	0:30:00	0.02%
Model making	3:15:00	0.15%
Model validation	1:01:00	0.05%
Model work	3:30:00	0.16%
NINA meeting	5:12:25	0.24%
NINA meeting preparation + small scrum	1:02:39	0.05%
NINA usertesting	1:41:00	0.08%
Planned Meeting with NINA	1:46:00	0.08%

USER - TIME ENTRY	DURATION	PERCENTAGE
Plot IOU threshold - Confidence	2:00:00	0.09%
Plotting horizontal diagram with dual axes	8:13:01	0.38%
Post-merge fixes and reformat	1:30:00	0.07%
Pre-merge UI bug fixes	1:00:00	0.05%
Prepare for testing	1:45:00	0.08%
Preparing for NINA meeting + help YOLOv8 batch implementation	0:39:00	0.03%
Present changes on CustomTkinter, proof of concept on saving directories in script	0:58:58	0.05%
Pretest Meeting	1:49:00	0.08%
PROG2900 Seminar - Crash course on Project management	1:45:00	0.08%
Python errors	0:45:00	0.03%
Python Script	9:49:31	0.46%
R&D best model for showcasing weight difference over iterations	1:45:00	0.08%
R&D for Overleaf formats	1:30:00	0.07%
R&D Matplotlib	2:00:00	0.09%
Read through notes from last week	1:44:59	0.08%
Read through pull requests and approve	0:45:00	0.03%
Reaffirm performance of past scripts	1:23:47	0.06%

USER - TIME ENTRY	DURATION	PERCENTAGE
Refactor UI + logging	2:45:00	0.13%
Reference meeting for bachelor course	1:40:00	0.08%
Refined implementation of CustomTkinter	2:13:41	0.1%
Research AI models	1:51:03	0.09%
Research Annotation tooling	4:44:59	0.22%
Research CNNs	3:04:15	0.14%
Research Datasets	1:17:38	0.06%
Research FFMPEG	2:42:27	0.13%
Research models for report	1:15:00	0.06%
Research on setup and reconfiguration of space management on device for later implementation of anotations for rnd and refinement of datasets + ordered more space for datasets	2:11:13	0.1%
Research Previous bachelor assignment for RND	30:33:47	1.42%
Research Tkinter	2:14:30	0.1%
Research YOLO models for use	1:16:00	0.06%
Researched annotation standard for AI	1:55:54	0.09%
Review and go through all of integration implementation for documentation	1:09:08	0.05%
Review and go through all of the quality assurance goals and document what we will need to implement later in the final report	5:00:00	0.23%
Review and setup pull requests related to development	1:00:00	0.05%

USER - TIME ENTRY	DURATION	PERCENTAGE
review profiling	0:12:09	0.01%
Review Report manager and compare to models	1:00:00	0.05%
Review Settings rework and review lint refactor/fix pull req	0:30:00	0.02%
RND on annotation software	1:15:00	0.06%
Scrum meeting	9:40:05	0.45%
Setting up environment for development	1:55:29	0.09%
Setting up Python for Assignment	2:24:45	0.11%
Settings RND	1:00:00	0.05%
Setup and research FFMPEG-Python for later use	3:00:00	0.14%
Setup Docker and Cvat for me and group member + Process dataset + Review annotations	3:00:00	0.14%
Setup for usertest	0:15:00	0.01%
Setup local CVAT on new disk	2:00:00	0.09%
Setup local dependencies for integrated project	0:53:02	0.04%
Setup NTNU-Machine + Jana request communication	3:24:08	0.16%
Setup Preliminary tools on laptop	1:45:00	0.08%
Skimmed through overleaf report	0:45:00	0.03%
Started finalizing Project Plan for first draft	3:02:02	0.14%

USER - TIME ENTRY

DURATION


PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Started work on Project plan	1:49:00	0.08%
Supervisor evaluation orientation	1:12:11	0.06%
Supervisor meeting	15:51:26	0.73%
Supervisor Meeting + Scrum	3:00:00	0.14%
Supervisor Meeting regarding first draft	1:01:37	0.05%
Tested second iteration	2:00:00	0.09%
Testing new branch for AI @benjamin	1:59:30	0.09%
Transfer COCO dataset to CVAT org.	1:45:15	0.08%
transferred script to new project branch	1:15:48	0.06%
Understanding the model performance evaluation branch	2:15:00	0.1%
Upgraded components of workstation in order to increase workflow and give QoL changes	2:00:00	0.09%
Uploading the COCO formatted datasets	3:39:07	0.17%
usertest	1:13:00	0.06%
Walk to meeting	7:30:00	0.35%
Work on Project Plan	3:02:00	0.14%
Work on second draft Project Plan	5:38:17	0.26%
Work on transferring second draft from docs to Overleaf	4:00:13	0.19%

USER - TIME ENTRY

DURATION

PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
 Lars Blütecher Holter	551:19:55	25.54%
Add abbreviations and glossarys through the whole report	3:27:26	0.16%
Add widgets to UI	2:40:14	0.12%
Annotation	96:17:45	4.46%
Collect EHDD from NINA	1:15:00	0.06%
Connecting UI and AI	2:10:28	0.1%
Daily scrum meeting	27:10:03	1.26%
Final report writing	167:23:39	7.75%
First iteration UI	20:58:32	0.97%
Getting everything ready for usertesting etc	7:36:52	0.35%
Helping with merge	3:38:07	0.17%
Implement logging	7:14:30	0.34%
Implement QSettings	11:18:21	0.52%
Install and setup everything needed	2:05:03	0.1%
Leaning TKinter	4:00:04	0.19%
Learning PyQt	10:55:58	0.51%
Looking up how to change from YOLOV5 to V8	1:49:36	0.08%

USER - TIME ENTRY

DURATION


PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Merge	0:25:00	0.02%
Merging	2:21:59	0.11%
NINA bachelor meeting	2:00:00	0.09%
NINA IT Meeting	1:05:00	0.05%
NINA Meeting	3:18:55	0.15%
NINA meeting	2:20:00	0.11%
PROG2900 Seminar	1:45:00	0.08%
PROG2900 Seminar Lynkurs prosjektstyring	1:45:00	0.08%
Read old bachelor thesis	14:35:35	0.68%
Reading through old bachelor theses	16:29:43	0.76%
Replacing YOLOV5 with YOLOV8	4:47:40	0.22%
Reviewing PRs	1:07:00	0.05%
Scrum meeting	4:51:46	0.23%
Scrum meeting Bacheloroppgave	14:13:05	0.66%
Set up school PC to train AI Model	1:00:00	0.05%
Setting up annotation	2:49:00	0.13%
Setting up dualboot with ubuntu	1:15:00	0.06%

USER - TIME ENTRY

DURATION

PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Setup all tools on laptop	1:15:00	0.06%
Setup anotation tools	2:21:37	0.11%
Setup environment	2:16:37	0.11%
Setup environment for running ai	1:31:52	0.07%
Started planing the bachlor theses	3:03:36	0.14%
Supervisor meeting	19:20:00	0.9%
System architecture	3:16:11	0.15%
Testing application looking for bugs	1:08:02	0.05%
Testing on UI	7:24:35	0.34%
Trying to fix gloassary and abbreviations	3:25:04	0.16%
UI changes based on usertesting	10:29:37	0.49%
User testing on NINA	1:37:05	0.07%
Work on project plan	28:56:46	1.34%
Working with pipeline	10:55:56	0.51%
Writing guide on how to annotate for NINA	8:06:36	0.38%
 Lillian Alice Wangerud	524:07:02	24.28%
annotation work	69:34:53	3.22%

USER - TIME ENTRY	DURATION	PERCENTAGE
Commenting code	1:27:29	0.07%
commenting on report and data managers	3:16:32	0.15%
Daily meeting	34:06:52	1.58%
daily meeting	14:23:04	0.67%
data manager and report component work	15:00:34	0.7%
data manager component	11:29:38	0.53%
documentation	156:00:04	7.23%
Documentation	17:32:48	0.81%
Documenting Choice for Front-end	0:47:57	0.04%
First iteration UI	20:38:38	0.96%
fixing database bugs	3:43:53	0.17%
Guidance meeting	2:15:00	0.1%
integrating report manager and data manager into process	15:14:51	0.71%
Learning PyQt	5:17:41	0.25%
Learning Tkinter	4:43:21	0.22%
Meeting with NINA	3:00:00	0.14%
Merging and setup together	2:16:00	0.1%

USER - TIME ENTRY

DURATION

PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Merging work together	2:33:59	0.12%
NINA meeting	7:01:55	0.33%
Planning Meeting	4:59:23	0.23%
Planning work ahead	2:57:02	0.14%
Project plan writing	22:10:13	1.03%
R&D for output report from the program	26:28:58	1.23%
Read previous bachelors	4:52:00	0.23%
Report component	14:06:56	0.65%
Report writing Seminar	1:00:00	0.05%
Scrum meeting	8:28:36	0.39%
scrums meeting	2:53:27	0.13%
scrums meeting and start of second draft	2:15:00	0.1%
Seminar	1:45:00	0.08%
Setting up work environment	3:30:00	0.16%
supervisor meeting	9:46:49	0.45%
Supervisor meeting	5:45:00	0.27%
System Architecture development	6:45:29	0.31%

USER - TIME ENTRY

DURATION

PERCENTAGE

USER - TIME ENTRY	DURATION	PERCENTAGE
Testing UI	2:20:00	0.11%
Travel	6:53:00	0.32%
UI changes	6:45:00	0.31%

G Meeting Logs

Møtereferat Veiledningsmøte

Dato: 24.01.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Feedback på prosjekt planen

Marius gir feedback på den nåværende prosjekt planen.

Feedbacken bestod av disse kommentarene:

- Effektmål og resultat mål burde skilles
- Spesifisere inni resultat mål accuracyen som skal oppnås, som for eksempel ønsker å oppnå 95% accuracy
- Legge til en enkel og forklarende illustrasjon kan hjelpe med leserens forståelse
- Utvide hvorfor vi har valgt SCRUM som development model
- Begrunne valg av utstyr under standarder
- Kan diskutere rundt tidligere oppgave utført for NINA

Sak 2. Kildehenvisning og referering til kilder

Det ble diskutert hvordan kilder skulle henvises til i prosjekt planen.

Kildehenvisning skal inneholde:

- Forfatter eller organisasjon
- Software/programvare + versjon nummer
- Adresse til nettsiden
- Dato sist oppdatert/publisert

Sak 3. Stilling til teknologi bruk

Diskusjon rundt bruk av teknologi som chatGPT. Veileder har ingen spesifikk stilling til bruken av chatGPT, men oppfordrer til å kvalitet sikre og reflektere over bruken i sluttrapporten.

Sak 4. Videre utvikling av programmet som er utenfor oppgave beskrivelse

Spørsmål om hvordan det skulle skrives om forbedringer eller videreutvikling som er utenfor oppgave beskrivelsen blir tatt opp og diskutert. Dette omhandlet ideer som å forbedre på NINA sitt fysiske oppsett ute i feltet ved å sette opp RAID for å redusere muligheten deres til å miste data.

Via møtet kom veileder og gruppen fram til at dette kan bli tatt opp i diskusjonen på sluttrapporten. Det å utvikle applikasjonen for å kunne gjøre videre utvikling kan også bli satt opp som et mål ved prosjektet.

Eventuelt

Det ble også spurt spørsmål rundt presentasjonene ved bachelor slutt. Det ble dermed informert at presentasjonen er obligatorisk men vektet ikke på karakteren.

Møtereferat Veiledningsmøte

Dato: 31.01.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Innleveringer

Møtet startet med innlevering av dokumenter nødvendig for bacheloren.

Kontrakten mellom Bachelor gruppen og Oppdragsgiver ble levert på Blackboard. Mens Prosjekt planen ble levert til veileder.

Sak 2. Bruk av teknologi som chatGPT og GitHub Copilot

Det ble videre diskusjon rundt bruken av AI teknologi innen for bachelor oppgaven med veileder.

Etter grundig diskusjon så kom veileder og gruppen fram til at det skal være lovlig og muligheter for å bruke disse verktøyene. Men så lenge bruk av det er referert og koblet til kodesnuttene eller teksten skrevet ved hjelp av AI. Det er viktig å forklare og reflektere over bruken av disse verktøyene igjennom utviklings prosessen i slutt rapporten. Marius påpeker også at i forhold til å bruke direkte avsnitt fra slike verktøy så er det viktig å vite rettigheter og licences rundt commercial bruk av innholdet gitt fra verktøyene. Dette var spesielt med tanke på GitHub Copilot og koden som Copilot finner.

Møtereferat Veiledningsmøte

Dato: 17.02.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. AI modell valg og bruk av modellen

Det ble forklart valget av AI modell, Yolo v5, for veileder og diskuterte rundt hvordan denne modellen skulle bli brukt.

Modellen skal kunne oppdage fisk inne i en frame uavhengig av kvalitet og bakgrunn på videoen. Marius dermed oppfordret oss til å passe på at modellen blir trent på variert data, og metoden og fargene den blir trent opp til å se. De ble diskutert om modellen skulle være oversensitiv eller ikke, og vi kom fram til at vi ønsket en mer oversensitiv modell for å unngå å miste data.

Under møtet ble det også brakt opp at modellens hoved bottleneck er iterasjonen av still bildene og overføringen av data fra harddisk til modellen.

Sak 2. User interface

Vi gikk gjennom og viste et første utkast av UI til veileder. Planlegger å koble sammen UI sammen med modellen i løpet av neste sprint.

Sak 3. System arkitektur

Viser en start på system arkitekturen til programmet for å få tilbakemeldiger fra veileder.

Feedback er som følger:

- Burde være kobling mellom harddisk og data manager
- Raw data fra model til manager burde bli navn gitt til noe bedre
- Burde være kobling mellom data manager og video komponenten
- Veldig brukbar på final report/ presentasjon til å forklare
- Vær forsiktig med bruk av farger

Sak 4. Dataset for trening av modellen

Siste punktet tatt opp var rundt datasettet som vi trenger for å trene modellen, som hvordan annotering av data skulle foregå og hvor vi får dataene fra.

Marius nevner at han har opp til 50,000 allerede annoterte bilder som kan bli gitt til gruppen. Ellers er datasettene fra Oppgavegiveren.

Veileder påpeker at da vi annoterer må vi passe på at vi har bilder av fisk fra forskjellige vinkler, men vi burde være forsiktige med å augmente data ettersom det kan påvirke treningen av modellen negativt. Via diskusjon kommer vi fram til at vi skal bruke en en fisk superklasse, som er foreldreklasse til arten funnet, da vi annoterer datasettet.

Møtereferat Veiledningsmøte

Dato: 02.03.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Samarbeid med NINA sin IT avdeling

Det blir informert til veileder at NINA legger til rette for dockerfil for prosjektet vårt.

Veileder så informerer til gruppen at vi ikke skal si at det er vårt arbeid og burde nevne at dette har NINA tatt ansvar for i final report.

Sak 2. Annotering av datasett

Det blir snakket om prosessen av å annotere datasettene som gruppen har fått. Vi i gruppen har oppdaget noen problemer ved at deler av videofilene er blitt korrupt som har gjort at noen stillbilder forsvant og annoteringene er forskyvd.

Marius som veileder foreslår at vi skriver om prosessen av a kvalitetssikre testing/trenings datasettet på rapporten og om opplevelsene rundt disse korrupte filene vi møtte på. Han nevner også at det er viktig å ha ett bredt datasett som dekker vanskelige situasjoner (aka, forskjellige tider, lys, vinkler, andre objekter i frame etc), men vi burde sette en grense på hvor mye data som er nødvendig og hva vi trenger for å ha et representativt datasett. Det blir også foreslått av veileder at vi kan spørre NINA om hvordan de annoterer datasettene sine.

Sak 3. Grafisk user interface prosess

Front-ended og dens grafiske UI blir vist til veileder.

Det blir også fortalt om hva som vurderes å jobbes videre med i forhold til GUI:

- logging og testing
- Arbeider på å sette opp en typ cache som lagrer instiller fra tidligere sessions
- legge til en standard CSS styling

Tilbakemeldinger og vurderinger vi som gruppe fikk fra veileder basert på front-ended vår:

- Burde ta en vurdering av hvor viktig styling er for oss
- Forklare design valg vi har tatt i rapporten vår
- Burde ta vurdering for design i forhold til fargeblindhet, nedsatt syn osv

Sak 4. Dokumentering av prosjektet

Det ble diskutert hvor viktig det er å dokumentere gjennom utviklingsprosessen.

Gjennom denne diskusjonen kom vi fram til at dette var viktige punkter som burde bli med i slutt rapporten:

- Vi har har diskusjon med IT avdelingen til NINA og at de har sett på appikasjonen vår så langt, hvor de nevnte at det var forståelig hvordan man skulle sette det opp. Dette kan diskuteres som å være en vertifisering av utviklings prosessen vår så langt.
- Diskutere rundt strukturen vi har hatt på å samle spørsmål før møter i Q&A threads
- Diskutere rundt utviklings prosessen i henhold til tidligere erfaring og hva vi har gjort annerledes.

Derimot kom vi også fram til at logging av timer er ikke det mest nødvendige på rapporten.

Møtereferat Veiledningsmøte

Dato: 17.03.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Sluttrapporten

Det var hovedsaklig diskusjon på de første delene av rapporten og dokumentasjonen skrevet så langt.

Kommentarene tilbake fra Marius på abstrakten så langt var:

- Ta opp problemstillingen i abstrakten
- Hvert avsnitt i introen burde referere til det samme stikkordet
- Lurt å gå over abstrakten igjen da man er ferdig med rapporten siden abstrakten skal reflektere rapporten

Videre så ble det diskutert hvor detaljert vi skulle dokumentere AI modellen brukt i programmet, etter som vi bruker en allerede eksisterende modell. Diskusjonen kommer fram til at dette er viktig da vi dokumenterer AI modellen:

- Modellen er en blackbox -> trengs det ikke mye forklaring på modellen
- Modellen har blitt endret på -> trengs det å rapportere på struktur og endringer
- Tall på resultater og avvik er viktig å diskutere

Sak 2. Oppdragsgiver ønsker rundt prosess 'rapporten'

Det ble så tatt opp under møtet at NINA, vår oppdragsgiver, har sent et forslag til hvordan de ønsker rapporten som programmet skriver ut skal være satt opp. Dette forslaget viste seg å inkludere deler som allerede har blitt diskutert med NINA at er utenfor omfanget til oppgaven.

Ved hjelp av veileder kom vi fram til en liste med ting inne i rapport forslaget som ikke ville være gjennomførbart å legge til basert på oppgavens omfang slik at vi kunne komme til en annen løsning som er mulig å utføre. Dette var ønsker som å identifisere fisk og basere rapporten på FishID, eller å gjenkjenne fisk som har blitt identifisert.

Møtereferat Veiledningsmøte

Dato: 21.03.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Annotering og datasett

Det første saken i dette møtet handlet om prosessen av å annotere og noen hindringer møtt på i løpet av denne prosessen.

Etter diskusjon om emnet så kom vi fram til at en udefinert klasse for fisk en ikke kjenner igjen og at siden datasettet skal brukes til å trene, teste og validere modellen burde datasette være så balansert mellom klassene som mulig.

Det var også en del mangel på variert data og fikk dermed data fra tidligere oppgave for å detekte fisk i video.

Sak 2. Presisjon av AI modellen

Det var diskusjon av hvordan man skal utnytte AI modellen best mulig og hvordan vi skal måle modellens presisjon.

Vi kom fram til at dette var de forskjellige måtene vi burde vurdere å måle presisjonen av modellen:

- Presisjon av klassifisering
- Presisjon av deteksjon på riktig frames over tid
- Presisjon av boundary box
- Viktigste er deteksjon over tid

Det ble også diskutert dersom det var mer effektivt å generalisere klassifikasjon til; Fisk/ikke Fisk og heller sende en video med generalisert klassifikasjon gjennom en komponent som videre klassifiserer inn i arter. Marius nevner også at å bruke en confusion matix for å illustrere rutene vi ønsker å dekke med modellen kan være lurt på rapporten.

Møtereferat Veiledningsmøte

Dato: 17.04.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Sluttrapporten

I denne saken diskuterte vi hva sluttrapporten burde inneholde og feedback fra veileder.

Feedbacken fra veileder på introduksjonen:

- Burde inneholde hva NINA gjør og deres motivasjon for å lage prosjektet
- En figur i introen kan enklere forklare hva oppgaven går ut på
- Tall på NINA's data kan hjelpe å putte ting i perspektiv, og gjøre at det virker som en nødvendig ting å utvikle
- Kan sette decision making og conflict management inn i det samme seksjonen
- 1.2 Prosjekt beskrivelse skal være mer overordnet (mer lik oppgavebeskrivelsen gitt av NINA)

Videre så ble det diskutert rapportens struktur og innhold og dette var veileders tilbakemeldiger.

- Lurt å se over kapitlene og vurdere hva som skal være egne kapitler og om noe kan slå sammen
 - Quality Assurance og testing kan gå i samme kapittel
 - Graphical userinterface kan muligens gå under implementasjon
 - Object detection kan muligens gå under implementasjon
- Lurt å legge til en figur tekst under tabellen på 4.1 Front-end technology
- Kan snakke om etiske, sosiale og økonomiske konsekvenser av applikasjonen

Eventuelt

Annet tatt opp undermøtet var muligheten for å legge til en knapp som åpner output mappen/rapporten på prosesseringen og at vi kunne se på å sette opp programmet slik at det kan videreutvikles senere.

Møtereferat Veiledningsmøte

Dato: 25.04.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Feedback på sluttrapporten

Vi gikk gjennom sluttrapporten og fikk feedback på den fra veileder.

Feedback på 3.1 under Development Process:

- Gi begrunnelse for hvorfor vi ikke bruker de andre modellene
- Koble begrunnelsen til requirements og prosjekt beskrivelsen

Feedback på 3.2 under Development Process:

- Argumenter for hvorfor vi har møter når vi har det, som hvorfor har vi møte med NINA før Veiledning og Scrum
- Forklar hvorfor tidspunktet til dagelig møtet ble valgt

Feedback på kapittel 6 Implementasjon:

- Et ER diagram for databasen kan enklere visualisere databasen
- Forklar at valgene av rapport formater til brukeren er blandt forvalgte fomater
- Burde være klarere på om AI er del av backend eller ikke, vi har valgt at den er uavhengig fra backend fra rapportens perspektiv
- Lurt å forklare hvordan implementasjonen samhandler med harddisken til NINA, dette burde bli basert på antagelser vi har gjort om NINA's bruk at programmet
- Skrive om hva som har blitt implementert for å tilrette legge videre utvikling, kan bli videre diskutert under diskusjon

Feedback på kapittel 7 Graphical Userinterface:

- Forklar hvorfor Figma (brukt før, gratis, etc)
- Utvid på hva slags feedback vi har fått fra NINA
- Utvid på design rules some vi har brukt for Ulen

Eventuelt

Andre ting brakt opp under møtet lurt å bruke vektor format (pdf/svg) for bilder og diagrammer og å sjekke om overskrifter er forklarende og passer sammen.

Møtereferat Veiledningsmøte

Dato: 02.05.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Feedback på sluttrapportens framgang

De var diskusjon med veileder on framgangen til sluttrapporten.

Tilbakemeldingene om figurer og tabeller:

- Oppfordrer til å skrive en del inni caption
- Viktig at man skal forstå figuren/tabellen via caption uten å lese selve teksten
- `\caption [kort caption for figur tabellen]{long caption}`

Feedback på kapittel 4 om Technologies:

- Det er litt overlapp med prototype avsnittene
- Referere til kilder for statements, som at PyQt er en industri standard/vidt brukt i indeustrien
- Ikke går for mye inn i datamanager/ report manager inne i technologies ettersom det har ikke blir introdusert enda som et konsept

Feedback på kapittel 5 System arkitekturen var hovedsakelig at den var veldig kort og dersom den ikke ville bli lengere så kunne den slås sammen med implementasjon.

Det var noen spørsmål om referering til andre kapittler og det kom fram at det burde legges til labels for kapittler som var forståelige. Disse labelene kan brukes til å referere til seksjoner i teksten med `\ref {label}`.

Inne på Implementasjon så blir det påpekt av veileder at det kan være lurt å visualisere front-end implementasjonen på en eller annen måte.

Kapittel 7 Graphical user interface fikk denne feedbacken:

- teksten på bildene er vanskelig å lese og burde være samme størrelse som bildeteksten
- kanskje legges sammen med eller før implementasjonen av frontended for å gi kontekst til implementasjonen

Møtereferat Veiledningsmøte

Dato: 09.05.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Feedback på sluttrapportens framgang

Gruppen og veileder gikk gjennom sluttrapporten og fikk enda mer feedback på rapportens framgang.

4.5 Tool overview Feedback

- CVAT er veldig isolert og burde bli koblet til resten
- Om rokere på hvor piler pekerslik at det er mer forståelig
- Vær forsiktig med bruk og nevning av chatGPT, pass på å bruke det slik som allerede eksisterende verktøy vi har lov til å bruke
- mulig å referere til bruk av AI

4.3 Lokal database og Libraries for rapporten Feedback

- Utfylle hvorfor valg av SQLite, som er at SQLite var den mest fremtredende og vel dokumentert modulen for å lage en serverløs database
- Gi begrunnelse for fil formatene og hvorfor det ble endring, aka spurt etter csv og xlsx

Det var litt forvirring om noe skulle være i Glossary eller Technologies. Vi kom fram til at vi burde først og fremst korte ned beskrivelser inne i tech, men at vi kunne gjøre det dobbelt opp. Hvor terminologi er i måde glossary og technologies slik some dette:

- Glossary: kort beskrivelse (brukes som oppslagsverk/påminnelse)
- Technologies: kort beskrivelse og en begrunnelse for bruk

Feedbacken på kapittel 7 GUI var hovedsakleig at det var greit å diskutere rundt at vi har lært dette tidligere, som å nevne at dette er ting vi har lært fra user-centered design.

Feedback på framgangen til kapittel 8 AI og object detection:

- Å fylle ut dette er hovedfokuset videre
- Viktig å vise at programmet vårt er mer enn kun GUI

- Inkludere info om hjelpe script
 - hvor viktige de er
 - hva de gjør
 - hvordan de er brukt

Møtereferat Veiledningsmøte

Dato: 16.05.2023 **Tid:** 14:00

Lokasjon: Gjøvik Ametyst 2.etasje Rom A232, NTNU Gjøvik

Tilstede:

Marius Pedersen (veileder),
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1. Feedback på sluttrapportens framgang

Vi gikk gjennom det som har blitt gjort siden sist og fikk feedback fra veileder.

Kapittel 8 Object detection Feedback:

- Burde forklare hvordan vi balanserer datasettet
- Sier vi splitter opp datasettet inn til trening, test og validerings sett men ikke hvordan
- Yaml blir nevnt men ikke forklart

Kapittel 11 Discussion Feedback:

- Oppfordrer til å strukturere kapittelet i på en mer forståelig måte
- Deler diskuterer teknologien, og burde bli flyttet til kapittel 4 Teknologi

Kapittel 12 Conclusion Feedback:

- Utvid på punktet 'legg til flere tester' under videre utvikling
- Burde forklare bedre hva som menes med 'mer annotasjon' og hvilken type annotering som vi trenger mer av
- 'Improve structure of code' virker som å være på feil sted, og burde bli flyttet til diskusjon, hvor vi kan bedre forklare dette punktet
- Videre utvikling burde være større funksjoner som er relevant for sluttbrukeren

Det ble også diskutert at det var mangel på referanser igjennom teksten og vi burde dermed passe på at vi legger til navn i teksten. I referanse listen vår burde vi også huske å legge til dato hentet og versjon nummer.

Veileder på peker også at dersom det er kodesnutter i teksten så burde det bli forklart i teksten dersom det ikke er kommentarer i koden. Det ble også nevnt at Implementasjon burde ha et flyt diagram mellom delene diskutert.

Møtereferat Møte med NINA

Dato: 16.01.2023 **Tid:** 14:00

Lokasjon: Vormstuguvegen 40, 2624 Lillehammer

Tilstede:

Tobias Holter,
Knut Marius Myrvold,
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1: Planlegging av fremtidige møter

Det ble bestemt at videre møter vil finne sted hver andre mandag kl. 14:00 via Teams.

Sak 2: Diskusjon om filtrering av videoer

Det ble diskutert muligheten for å skille videoer med bevegelser eller fisker fra videoer uten. Målet er å filtrere ut støyelementer som bobler, sjøgress, plankton, osv. Det ble uttrykt behov for en filtreringsmekanisme som kan skille ønsket innhold fra uønsket støy.

Etter diskusjon kom vi fram til at harddisken med bideoer må bli hentet fra felten for å kunne prosessere videoene ettersom oppsettet ute i felten er ikke tilstrekkelig sterkt nok til å prosessere videoene i sanntid. Det ble enighet om at videoene må prosesseres etter at de er hentet inn fra felten.

Det ble diskutert viktigheten av å jobbe mot høy presisjon for å sikre pålitelige resultater og det ble satt et mål om å oppnå 95% presisjon i prosesseringen av videoene.

Sak 3: Ønsker til brukergrensesnittet (UI)

Det ble så diskutert ønskede funksjoner og egenskaper for brukergrensesnittet. Og vi kom fram til at disse var de mest viktige elementene:

- Valg av input folder og output destinasjon
- Valg om å beholde orginal klippet

- Output videoen har kun klippene fra input videoen (med en navn indikasjon at den er prosessert)
- Pause og/eller stoppe prosessen
- Varsel dersom det ikke er plass
- Generere en rapport på den prosesserte dataen (hvor i original klippet ble klippene funnet, antall klipp funnet etc)
- Sortering av data basert på metadata (som dato)

Eventuelt

Det ble påpekt at språket som skal brukes i løsningen kan velges av gruppen.

Møtereferat Møte med NINA

Dato: 30.01.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Tobias Holter,
Knut Marius Myrvold,
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1: Gjennomgang av prosjektplanen og starten av applikasjonen

Vi startet med å diskutere prosjektet planens framgang og starten av utviklingen av applikasjonen. Det ble besluttet å sende prosjektplanen til Knut og Tobias for gjennomgang og tilbakemelding.

Til dette hadde det blitt lagd en skisse av user interfacen, dette var feedbacken:

- UI-skissen virker intuitiv og brukervennlig.
- De fremmet ønske om tydelige pause og stop knapper
- De var åpne til at det var mulig å legge til eller fjerne alternativer
- De ønsket også avanserte alternativer for mer spesifikke valg, samt muligheten for å legge til ekstra alternativer

Sak 2: Valg av hardware for applikasjonen

Etter diskusjon om hardware nødvendig for applikasjonen kom vi fram til:

- Budsjettet for maskinen er satt til 25 000 kr.
- En lokal maskin blir målet for prosjektet.
- Det ble påpekt behovet for assistanse for å finne riktig hardware til maskinen.

Sak 3: Diskusjon om datasett og harddisk

Neste saken handler om datasett tilgjengelig og muligheten for å skaffe mer data fra NINA eller andre kilder.

- Det ble nevnt at det finnes flere hundrevis av timer med uannoterte data.
- Det ble foreslått å kontakte den tidligere gruppen som jobbet med fisk for å sjekke om de har noen annoterte datasett som de kan dele.
- Det ble enighet om å utforske mulighetene for å skaffe annoterte datasett fra andre kilder.

Videre ble det diskutert hvordan harddiskene til NINA var organisert og kom fram til dette:

- Harddiskfilene er organisert i en mappe med videoene.
- Filene er navngitt med disse elementene i navnet:
 - pre-fix
 - dato
 - tidsstempel
 - nummer

Det ble uttrykt ønske om å kunne gruppere filene før prosessering og ble diskutert muligheten for å legge til en funksjon i UI-en for å enkelt gruppe filene.

Eventuelt

Det ble nevnt muligheten for å legge til en funksjon for å trene modellen selv senere med ny data. Dette vil nok bli utført av NINA sin IT avdeling dersom det blir lagt til.

Diskutert usikkerhet rundt hvordan rapportene utskrevet fra programmet med info om prosessen ønskes presentert. Det ble foreslått å lage rapporter for hver "batch" som blir prosessert og å opprette en database for lagret data og metadata fra prosesseringen. Det ble snakket om muligheten for å dele opp rapportene basert på dager og inkludere en linje for hver fil.

Videre så ble det tatt opp mangelen på backup-løsning av dataene ute i feltet. Det ble påpekt at det ikke er økonomisk mulig å anskaffe en server eller RAID-løsning.

Møtereferat Møte med NINA

Dato: 27.02.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Tobias Holter,
Knut Marius Myrvold,
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1: Gjennomgang av prosesseringen og system arkitekturen

Det ble presentert en demonstrasjon av prosesseringen og informert om muligheten til å angi input- og output-kataloger. Basert på den nåværende AI-en skal det ta en estimert tid på omtrent 5-6 minutter å prosessere 30 minutters videoer.

Gruppen viser den planlagte system arkitekturen av programmet og det ble påpekt at systemarkitekturen ikke trenger å endres fra Windows til Linux.

Sak 4: Opprettelse av en lokal database

Gruppen og NINA's representanter diskuterer behovet for å opprette en liten lokal database. Databasen vil inneholde informasjon om hver video og gjenkjenningsprosessen. Informasjonen lagret inne på databasen er ment til å generere en lesbar rapport som vil kunne brukes til å krysse-referere med den opprinnelige videoen og sjekke tidsstempler.

Sak 5: AI modellens presisjon

Via diskusjon om AI modellens presisjon og det ble understreket viktigheten av at AI-en er oversensitiv. Det blir brakt opp at det er bedre å ha falske positive enn å gå glipp av gjenkjenninger av fisk og at veldig små fisker kan bli oversett hvis AI-en ikke er tilstrekkelig sensitiv.

Eventuelt

Det blir brakt opp at NINA's tekniske personell vil hjelpe med å sette opp deployment av applikasjonen via WebGL.

Møtereferat Møte med NINA

Dato: 29.03.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Tobias Holter,
Knut Marius Myrvold,
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1: NINA's servere og WebGL

Det ble informert om at NINA's servere eller WebGL ikke er helt klare for ytelsestesting ennå. Videre ble det diskutert muligheten for å sende programmet til Fransesco per e-post for testing.

Sak 2: Rapport

En diskusjon rundt rapporten programmet skriver ut om prosessen blir gjort. Det ble påpekt at systemet ikke kan oppdage individuelle fisker og kan dermed ikke utføre foreslått rapportstruktur.

Dette er noen av forslagene NINA's representanter gjorde for rapporten:

- Legge til sannsynlighet for deteksjon for hver fisk for å kunne filtrere ut deteksjoner med lav sannsynlighet.
- Legge til notater med mulig antall eller art.
- Et ark som gir et sammendrag av tiden der fisk er blitt detektert.
- Det ble diskutert muligheten for å endre filformatet fra CSV til XLSX.

Det ble informert om at NINA vil sende forslag til nytt filstruktur.

Sak 3: Brukertest

Det ble avtalt å utføre brukertest med Tobias, Knut, Fransesco og Benjamin etter forespørsel.

Eventuelt

Det ble besluttet å droppe møtet som var planlagt for 10.04 og flytte det til 17.04.

Møtereferat Møte med NINA

Dato: 17.04.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Tobias Holter,
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1: Usertesting

Under dette møtet ble det utført en bruker test med en av NINA's representanter, Tobias, slik at vi kunne få feedback på om applikasjonen var brukervennelig.

Feedback på user interface:

- Ønsker å kunne se innholdet i mappen de velger ved å trykke på "Browse files".
- Tydeliggjør at buffertime er angitt i sekunder.
- Boksen for rapportformat bør være større for bedre lesbarhet.
- Informasjon om hvor rapporten blir lagret.
- Ønsket forklaringer på "Batch size" og "Prediction Threshold".

Feedback gitt på rapporten:

- Endre navnet på rapporten.
- Kun nødvendig med XLSX eller CSV-formatene.
- Legge til en dato-kolonne i rapporten.
- Antall deteksjoner ikke trengs å inkluderes i rapporten.
- Burde kun skrive ut data fra databasen som er blitt prosessert.

Feedback på prosesseringsfunksjonalitet:

- Ønske om å kunne pause prosessen underveis.
- Implementere en sikkerhetsmekanisme i tilfelle lagringsplassen blir full.
- Legge til den opprinnelige videoens dato i den prosesserte videoen.
- Behov for å estimere prosesseringstiden.
- En knapp som fører til output-mappen eller åpner rapporten (en "nice to have" funksjon, ikke essensiell).
- Ønsker muligheten til å prosessere de fire mest vanlige videoformatene.
- Inkluder en boks for å velge mellom binær gjenkjenning eller artsgjenkjenning.

Møtereferat Møte med NINA

Dato: 08.05.2023 **Tid:** 14:00

Lokasjon: Digitalt Teams Møte

Tilstede:

Tobias Holter,
Knut Marius Myrvold,
Lars Blütecher Holter (gruppe leder),
Benjamin Letnes Bjerken,
Daniel Hao Huynh,
Lillian Alice Wangerud

Referat skrevet av: Lillian Alice Wangerud

Sak 1: Visning av nye endringer i programmet

De nyeste endringene i programmet ble vist og diskutert med NINA's representanter. Endringene gjort var som følger:

- En "Max detections" funksjon ble lagt til for å bestemme antall fisk som blir detektert per frame.
- En "Frame buffer" ble lagt til for å tillate buffer mellom frame ranges.
- En "CRF" (Constant Rate Factor) funksjon ble lagt til for å bestemme kvaliteten på videoen. Redusert kvalitet kan spare plass på harddisken.

Sak 2: Videre arbeid med applikasjonen

Det var diskusjon anngådede videre arbeid med applikasjonen fra dette punktet. Dette var ikke mye men inkluderte ting som:

- Behovet for å endre navnet på applikasjonen for å gjøre det mer forståelig.
- Endring på plasseringen av knapper for å gi en mer logisk brukeropplevelse.
- Legge til en dropdown for å kunne velge vektorer for modellen.

Eventuelt

Vi diskuterte at det hovedsaklig vil være arbeid på sluttrapporten videre og NINA's representanter oppfordret å ta kontakt hvis det trengs ytterligere informasjon til rapporten.

Det ble også snakket om muligheten for å levere tilbake harddisker etter presentasjonen som skal holdes 6. juni.

H Annotation Guide

Annotation guide

This guide requires some knowledge of python programming.

CVAT (Computer Vision Annotation Tool) is an open-source tool developed by Intel for annotating images and videos with various types of objects, such as bounding boxes, polygons, and keypoints. It is especially useful for creating labeled datasets for machine learning and computer vision applications.

You can install CVAT locally or on a server through this guide:

<https://opencv.github.io/cvat/docs/administration/basics/installation/>

Alternatively you can use their own service: <https://www.cvat.ai/pricing/cloud>

Step 1: Create a new project in CVAT

Click "Projects" in the top navigation bar, then click the blue "plus" symbol in the upper right corner -> Create a new project. Name it what you want and then add labels to it by clicking "Add label" under the "Constructor" tab. Make sure you add these labels to the dataset, you can also add more:

1. Gjedde
2. Gullbust
3. Rumpetroll
4. Stingsild
5. Ørekyt
6. Abbor
7. Brasme
8. Mort
9. Vederbuk
10. Frosk
11. Annen fisk

Step 2: Create a new annotation task

On the project page, click the blue "plus" button -> Create a new task.

Enter a name for the task.

In the "Files" section, click "Select files" to upload the videos you want to annotate.

The default settings are fine, but you can change them as you want.

Click "Submit" to create the task.

NOTE: You can also annotate images instead of videos, but then you have to export using the "YOLO 1.1" format, more on that below.

Step 3: Annotate the images or videos

For an in depth view on the annotation rules we used in our project, see section 8.4.3 in our bachelor

thesis.

In the "Tasks" list, click on the task you just created.

Click the "Open" button to start annotating.

Use the annotation tools in the sidebar to draw and label the objects in the images or videos.

Please refer to their documentation for more of a user guide: <https://opencv.github.io/cvat/docs/>

Save your progress frequently by clicking the "Save" button in the bottom-right corner.

When you're finished annotating, click the "Done" button.

NB! Only annotate using bounding boxes! (this is the only annotation method supported by our implementation of YOLO)

Step 4: Export annotations

In the task view, click the "Actions" dropdown and select "Export as a dataset."

If you have the CVAT installation installed on a sufficient server you might be able to export with images. If that is the case you can use the "YOLO 1.1" format and toggle on "Save images".

If not, you will have to export as "CVAT for video 1.1" without images. We will go through how to get the images through a generation script later.

Click the "Export" button to download the annotations as a zip file.

Now you have successfully installed CVAT, created an annotation task, annotated videos, and exported the annotations. You can use these annotations as input for your machine learning or computer vision algorithms.

Move all the exported zip files you exported using the "CVAT for video 1.1" format into a single folder if you have any. (do not extract the data from the zip files)

Also create a folder for where you want the final dataset to be (or use ours).

Now you will have to extract all the zip files you exported using "YOLO 1.1" (if you have any) into the final dataset folder, but only extract the "obj_train_data" folder (this is the folder containing all the images and annotations). Make sure to rename it to something else after extracting it, like the original video name to make it distinct from the others you extract.

The final dataset folder should now consist of folders containing images and .txt files for all the exports did using "YOLO 1.1", and you should have another folder containing all the zip files you exported using "CVAT for video 1.1".

You should also add more background images for new environments if you have new environments to minimize false positive detections. A good rule is to have about 10% of the total amount of images be background images (images with no fish), this is adjustable in the generation script, provided you have enough background images.

You can see that we have background images in the "backgrounds" and "backgrounds_2" folder in our dataset.

Dataset generation

Step 1: Clone the bachelor thesis repository

Open a terminal window and run the following command to clone the bachelor thesis repository:

```
git clone https://github.com/beuss-git/bachelor-oppgave-nina
```

Follow the README file in the repository to install the required development tools.

Step 2: Edit the generation script for your setup

We will be editing `tools/generate_dataset/generate_dataset.py`.

`CVAT_EXPORTS_FOLDER` should contain the path to the folder containing the zip files you exported using "CVAT for video 1.1"

`DATASET_FOLDER` should contain the path of the final dataset folder.

`SOURCE_VIDEO_FOLDERS` should contain the paths of all root folders where you sourced the videos from. This is used to extract images from videos you exported using "CVAT for video 1.1". The script will search recursively for the videos, so depth doesn't matter as long as the root folders encompass all the video folders.

The rest of the variables are fine as they are, just make sure to add to the `CLASSES` array if you added more classes in the CVAT project.

Now you just need to run `python generate_dataset.py` from the `tools/generate_dataset` folder and it should generate the final dataset to where you pointed `DATASET_FOLDER` to. The script should have generated `train.txt` and `val.txt`, pointing to all the images in the dataset.

To start training you can use our training script found at `model_research/train.py`

Just make sure to update `nina.yaml`.

To use our trained weights as a starting point, you have to change "yolov8s.pt" on the following line, to the path of our weights (for example "v8s-640-classes-augmented-backgrounds.pt" (you need to specify the full path)):

```
model = YOLO("yolov8s.pt")
```

To gain a bit more insight, you can follow this guide instead: <https://medium.com/augmented-startups/train-yolov8-on-custom-data-6d28cd348262>

You can skip to step-4 and instead of "Complete path to train folder" and "Complete path to val folder", set it to `train.txt` and `val.txt`, path is the same as `DATASET_FOLDER`.

You can look at `nina.yaml` in our dataset for reference.

Deploying the weights

When you have finished training you should have a `best.pt` in the results folder.

Step 1: Move trained weights file (`best.pt` found in `runs/detect/train[n]`) into `app/data/models`.

Step 2: Change the name of the weights file to something else

You should now be able to pick the new weights through our application.

I Code

```
1 """Video processor module. Contains functions for cutting and
2   annotating videos."""
3 from pathlib import Path
4 from typing import Any, Callable, Dict, List, Tuple
5
6 import av
7 import av.datasets
8 import av.packet
9 import av.video
10 import numpy as np
11 from PIL import Image, ImageDraw, ImageFont
12 from PIL import __version__ as pil_version
13 from tqdm import tqdm
14 from ultralytics.yolo.utils.checks import check_font,
15   check_version
16
17 from app import settings
18 from app.logger import get_logger
19 from app.video_processor import Detection
20
21 logger = get_logger()
22
23 def color_to_hex(color: Tuple[int, int, int]) -> str:
24     """Converts a color tuple to a hex string."""
25     return f"0x{color[0]:02x}{color[1]:02x}{color[2]:02x}"
26
27 class Annotator: # pylint: disable=too-few-public-methods
28     """
29     A more performant and specialized version of the
30     Annotator class from ultralytics.yolo.utils.plotting
31     """
32
33     def __init__(
34         self,
35         frame_size: Tuple[int, int],
36         color: Tuple[int, int, int] = (255, 0, 0),
37         line_width: int | None = None,
38         font_size: int | None = None,
39     ) -> None:
40         try:
41             font = check_font("Arial.Unicode.ttf")
42             size = font_size or max(round(sum(frame_size) / 2 *
43 0.035), 12)
44             self.font = ImageFont.truetype(str(font), size)
45         except Exception: # pylint: disable=broad-exception
46             self.font = ImageFont.load_default()
47             self.line_width = line_width or max(round(sum(frame_size)
48 / 2 * 0.003), 2)
49             self.color = color
50             self.pil_9_2_0_check = check_version(pil_version, "9.2.0")
51             # deprecation check
52             self.text_color = (255, 255, 255)
53
54     def annotate(
55         self, frame: np.ndarray[Any, Any], detections: List[
```

```

Detection]
52 ) -> np.ndarray[Any, Any]:
53     """
54     Draws bounding boxes and labels on a frame for the
    specified detections.
55
56     Args:
57         frame: The frame to draw on.
58         detections: A list of detections to draw.
59
60     Returns:
61         The frame with bounding boxes and labels drawn on it.
62     """
63     image = frame if isinstance(frame, Image.Image) else Image
    .fromarray(frame)
64     draw = ImageDraw.Draw(image)
65     for detection in detections:
66         label = f"{detection.confidence:.2f} {detection.label}"
67
68         box = (
69             int(detection.xmin),
70             int(detection.ymin),
71             int(detection.xmax),
72             int(detection.ymax),
73         )
74
75         draw.rectangle(box, width=self.line_width, outline=
    self.color) # box
76         if self.pil_9_2_0_check:
77             _, _, width, height = self.font.getbbox(
78                 label
79             ) # text width, height (New)
80         else:
81             width, height = self.font.getsize(
82                 label
83             ) # text width, height (Old, deprecated in 9.2.0)
84         outside = box[1] - height >= 0 # label fits outside
    box
85         draw.rectangle(
86             (
87                 box[0],
88                 box[1] - height if outside else box[1],
89                 box[0] + width + 1,
90                 box[1] + 1 if outside else box[1] + height +
    1,
91             ),
92             fill=self.color,
93         )
94         draw.text(
95             (box[0], box[1] - height if outside else box[1]),
96             label,
97             fill=self.text_color,
98             font=self.font,
99         )
100     return np.asarray(image)
101
102

```

```

103 # Thanks to https://github.com/PyAV-Org/PyAV/blob/main/tests/
104     test_seek.py
105 def frame_to_timestamp(frame: int, video_stream: av.video.stream)
106     -> int:
107     """
108     Convert a frame number to a timestamp using the time base and
109     frame rate of a video stream.
110
111     Args:
112     frame (int): The frame number.
113     video_stream (av.video.stream): The input video stream.
114
115     Returns:
116     int: The timestamp in microseconds.
117     """
118
119     time_base = float(video_stream.time_base)
120     rate = float(video_stream.average_rate)
121
122     target_sec = float(frame) * 1.0 / rate
123     timestamp = target_sec / time_base + video_stream.start_time
124
125     return int(round(timestamp))
126
127 def timestamp_to_frame(timestamp: float, video_stream: av.video.
128     stream) -> int:
129     """
130     Convert a timestamp to a frame number using the time base and
131     frame rate of a video stream.
132
133     Args:
134     timestamp (float): The timestamp in microseconds.
135     video_stream (av.video.stream): The input video stream.
136
137     Returns:
138     int: The frame number.
139     """
140
141     return int(
142         round(
143             (timestamp - video_stream.start_time)
144             * float(video_stream.time_base)
145             * float(video_stream.average_rate)
146         )
147     )
148
149 def process_packet( # pylint: disable=too-many-arguments
150     packet: av.packet,
151     current_frame: int | None,
152     start: int,
153     end: int,
154     video_stream: av.video.stream,
155     output_container: av.container.output,
156     output_stream: av.video.stream,
157     predictions: Dict[int, List[Detection]] | None,
158     annotator: Annotator,
159     pbar: tqdm,

```

```

156     notify_progress: Callable[[int], None] | None = None,
157 ) -> Tuple[int | None, bool]:
158     """
159     Process a packet of video frames, encode the frames,
160     and mux the resulting packets into an output container.
161
162     Args:
163         packet (av.packet): The input packet of video frames.
164         current_frame (int | None): The current frame number.
165         start (int): The starting frame number for the segment.
166         end (int): The ending frame number for the segment.
167         video_stream (av.video.stream): The input video stream.
168         output_container (av.container.output): The output
169         container.
170         output_stream (av.video.stream): The output video stream.
171         predictions (Dict[int, List[Detection]] | None): Optional
172         detections for each frame.
173         pbar (tqdm): A progress bar to update.
174
175     Returns:
176         Tuple[int, bool]: The current frame number and a flag
177         indicating
178         whether processing should continue.
179     """
180     for frame in packet.decode():
181         if current_frame is None:
182             current_frame = timestamp_to_frame(frame.pts,
183             video_stream)
184             assert (
185                 current_frame - start <= 0
186             ), f"Delta: {current_frame - start}, probably seeked
187             past start frame"
188         else:
189             current_frame += 1
190
191         if current_frame > end:
192             return current_frame, False
193
194         if current_frame >= start:
195             frame_image = frame.to_ndarray(format="bgr24")
196
197             if predictions is not None:
198                 frame_detections = predictions.get(current_frame,
199                 [])
200                 frame_image = annotator.annotate(frame_image,
201                 frame_detections)
202
203             output_frame = av.VideoFrame.from_ndarray(frame_image,
204             format="bgr24")
205
206             packet = output_stream.encode(output_frame)
207             if packet is not None:
208                 output_container.mux(packet)
209                 pbar.update(1)
210                 if notify_progress is not None:
211                     notify_progress(int((pbar.n / float(pbar.total
212                     )) * 100))

```

```

205     if current_frame is None:
206         return None, True
207
208     return int(current_frame), True
209
210
211 def process_frame_ranges( # pylint: disable=too-many-arguments
212     frame_ranges: List[Tuple[int, int]],
213     input_container: av.container.input,
214     video_stream: av.video.stream,
215     output_container: av.container.output,
216     output_stream: av.video.stream,
217     predictions: Dict[int, List[Detection]] | None,
218     annotator: Annotator,
219     notify_progress: Callable[[int], None] | None = None,
220 ) -> None:
221     """
222     Process a list of frame ranges, seek to the appropriate
223     timestamps,
224     and process each packet of video frames.
225
226     Args:
227         frame_ranges (List[Tuple[int, int]]): A list of (start,
228         end) frame ranges.
229         input_container (av.container.input): The input container.
230         video_stream (av.video.stream): The input video stream.
231         output_container (av.container.output): The output
232         container.
233         output_stream (av.video.stream): The output video stream.
234         predictions (Dict[int, List[Detection]] | None): Optional
235         detections for each frame.
236     """
237     with tqdm(
238         total=sum(end - start + 1 for start, end in frame_ranges),
239         desc="Processing frames",
240     ) as pbar:
241         for start, end in frame_ranges:
242             timestamp = frame_to_timestamp(start, video_stream)
243             # Verify that the timestamp is correct
244             assert timestamp_to_frame(timestamp, video_stream) ==
245             start
246             input_container.seek(
247                 int(timestamp), any_frame=False, backward=True,
248                 stream=video_stream
249             )
250
251             current_frame = None
252             for packet in input_container.demux(video_stream):
253                 current_frame, continue_processing =
254                 process_packet(
255                     packet,
256                     current_frame,
257                     start,
258                     end,
259                     video_stream,
260                     output_container,
261                     output_stream,
262                     predictions,

```

```

256         annotator,
257         pbar,
258         notify_progress,
259     )
260     if not continue_processing:
261         break
262
263
264 def cut_video(
265     input_path: Path,
266     output_path: Path,
267     frame_ranges: List[Tuple[int, int]],
268     predictions: Dict[int, List[Detection]] | None = None,
269     notify_progress: Callable[[int], None] | None = None,
270 ) -> None:
271     """
272     Cut a video into segments specified by a list of frame ranges,
273     and optionally annotate the frames with detections.
274
275     Args:
276         input_path (Path): The path to the input video file.
277         output_path (Path): The path to the output video file.
278         frame_ranges (List[Tuple[int, int]]): A list of (start,
279     end) frame ranges.
280         predictions (Dict[int, List[Detection]] | None, optional):
281             A dictionary mapping frame numbers to lists of
282             Detection objects. Defaults to None.
283
284     Raises:
285         FileNotFoundError: If the input file does not exist.
286         av.AVError: If there is an error opening or processing the
287             input file,
288             or encoding/muxing the output file.
289
290     Returns:
291         None
292     """
293     input_container = av.open(str(input_path))
294     video_stream = input_container.streams.video[0]
295     video_stream.thread_type = "AUTO"
296
297     output_container = av.open(str(output_path), mode="w")
298     fps = video_stream.average_rate.numerator / video_stream.
299     average_rate.denominator
300     output_stream = output_container.add_stream(
301         "libx264", str(fps), options={"crf": str(settings.
302     video_crf)})
303     )
304     output_stream.width = video_stream.codec_context.width
305     output_stream.height = video_stream.codec_context.height
306     output_stream.pix_fmt = video_stream.codec_context.pix_fmt
307
308     annotator = Annotator((output_stream.width, output_stream.
309     height))
310
311     process_frame_ranges(
312         frame_ranges,
313         input_container,

```



```

308     video_stream,
309     output_container,
310     output_stream,
311     predictions,
312     annotator,
313     notify_progress,
314 )
315
316 packet = output_stream.encode(None)
317 if packet is not None:
318     output_container.mux(packet)
319
320 output_container.close()
321 input_container.close()

```

```

1  """Video processor module. Contains functions for processing
2  videos."""
3  import os
4  import subprocess
5  from pathlib import Path
6  from typing import Dict, List, Tuple
7
8  import ffmpeg
9  from app.video_processor import Detection
10
11
12 def __run_ffmpeg(
13     args: List[str],
14     pipe_stdin: bool = False,
15     pipe_stdout: bool = False,
16     pipe_stderr: bool = False,
17     quiet: bool = False,
18 ) -> Tuple[bytes, bytes]:
19     stdin_stream = subprocess.PIPE if pipe_stdin else None
20     stdout_stream = subprocess.PIPE if pipe_stdout or quiet else
21     None
22     stderr_stream = subprocess.PIPE if pipe_stderr or quiet else
23     None
24     with subprocess.Popen(
25         args, stdin=stdin_stream, stdout=stdout_stream, stderr=
26         stderr_stream
27     ) as process:
28         out, err = process.communicate()
29         retcode = process.poll()
30         if retcode:
31             raise ffmpeg.Error("ffmpeg", out, err)
32     return out, err
33
34 def color_to_hex(color: Tuple[int, int, int]) -> str:
35     """Converts a color tuple to a hex string."""
36     return f"0x{color[0]:02x}{color[1]:02x}{color[2]:02x}"
37
38 def draw_detections(
39     video_stream: ffmpeg.nodes.Stream,
40     detections: Dict[int, List[Detection]],

```

```

40     start_frame: int,
41     end_frame: int,
42 ) -> ffmpeg.nodes.Stream:
43     """Draw the detections on the video stream."""
44     frame_offset = start_frame
45     for frame_number, frame_detections in detections.items():
46         # Simplify the above expression
47         if start_frame <= frame_number <= end_frame:
48             adjusted_frame_number = frame_number - frame_offset
49             for detection in frame_detections:
50                 # NOTE: We round the coordinates and dimensions to
51                 #       the nearest even number to
52                 #       avoid "flickering" when they are not
53                 #       perfectly aligned with the pixel grid
54                 x_pos = round(detection.xmin / 2) * 2
55                 y_pos = round(detection.ymin / 2) * 2
56                 width = round((detection.xmax - detection.xmin) /
57                             2) * 2
58                 height = round((detection.ymax - detection.ymin) /
59                             2) * 2
60
61                 video_stream = video_stream.drawbox(
62                     x=x_pos,
63                     y=y_pos,
64                     width=width,
65                     height=height,
66                     color=color_to_hex((255, 0, 0)),
67                     thickness=2,
68                     enable=f"eq(n,{adjusted_frame_number})",
69                 ).drawtext(
70                     # TODO: package the font ourselves to make it
71                     #       work on all platforms
72                     fontfile=r"C:\Windows\Fonts\consola.ttf",
73                     text=f"{detection.label}: {detection.
74                     confidence:.2f}",
75                     x=detection.xmin + 4,
76                     y=detection.ymin + 10,
77                     fontsize=16,
78                     fontcolor="white",
79                     enable=f"eq(n,{adjusted_frame_number})",
80                     box=1,
81                     boxcolor=color_to_hex((0, 0, 0)) + "80",
82                 )
83             return video_stream
84
85 def cut_video(
86     input_path: Path,
87     output_path: Path,
88     frame_ranges: List[Tuple[int, int]],
89     predictions: Dict[int, List[Detection]] | None = None,
90 ) -> None:
91     """Cuts the video to the given frame ranges.
92
93     Args:
94         input_path: The path to the input video.
95         output_path: The path to the output video.
96         frame_ranges: The frame ranges to keep.

```

```

92     predictions: The detections to draw on the video. (
optional)
93     """
94
95     input_file: ffmpeg = ffmpeg.input(str(input_path))
96
97     streams = []
98
99     for start_frame, end_frame in frame_ranges:
100         # Trim the video to the frame range
101         trimmed_stream = input_file.trim(
102             start_frame=start_frame, end_frame=end_frame
103         ).filter("setpts", "PTS-STARTPTS")
104
105         if predictions is not None:
106             # Draw the detections on the video
107             trimmed_stream = draw_detections(
108                 trimmed_stream, predictions, start_frame,
end_frame
109             )
110
111         streams.append(trimmed_stream)
112
113     command = ffmpeg.concat(*streams).output(str(output_path)).
overwrite_output()
114
115     args = command.compile()
116
117     # NOTE: The filter_complex argument is too long to be passed
as a command line argument.
118     # ffmpeg-python doesn't support filter_complex_script so we
need to manually invoke it.
119
120     # find the index of the -filter_complex argument
121     filter_complex_index = args.index("-filter_complex")
122
123     # replace the -filter_complex argument with -
filter_complex_script
124     args[filter_complex_index] = "-filter_complex_script"
125
126     # write the filter_complex string to a file
127     with open("filter_complex.txt", "w", encoding="utf-8") as file
:
128         file.write(args[filter_complex_index + 1])
129
130     # replace the filter_complex string with the path to the file
131     args[filter_complex_index + 1] = "filter_complex.txt"
132
133     # run ffmpeg with the arguments
134     __run_ffmpeg(args)
135
136     # Delete the filter_complex file
137     os.remove("filter_complex.txt")

```

```

1 def detected_frames_to_ranges(
2     frames: List[int], frame_buffer: int
3 ) -> List[Tuple[int, int]]:
4     """Convert a list of detected frames to a list of ranges.

```

```

5     Due to detection inaccuracies we need to allow for some
6     dead frames
7     without detections within a valid range.
8
9     Args:
10    frames: A list of detected frames.
11    frame_buffer: The number of frames we allow to be without
12    detection
13
14    """
15
16    before we consider it a new range.
17
18    """
19
20    if len(frames) == 0:
21        return []
22
23    frame_ranges: List[Tuple[int, int]] = []
24    start_frame = frames[0]
25    end_frame = frames[0]
26
27    for frame in frames[1:]:
28        if frame <= end_frame + frame_buffer:
29            # Extend the range
30            end_frame = frame
31        else:
32            # Start a new range
33            frame_ranges.append((start_frame, end_frame))
34            start_frame = frame
35            end_frame = frame
36
37    # Add the last range
38    frame_ranges.append((start_frame, end_frame))
39
40    return frame_ranges

```

```

1 def add_buffer_to_ranges(
2     frame_ranges: List[Tuple[int, int]], video_path: Path
3 ) -> List[Tuple[int, int]]:
4     """Add buffer time before and after each frame range and merge
5     overlapping ranges"""
6
7     cap = cv2.VideoCapture(str(video_path))
8     fps = cap.get(cv2.CAP_PROP_FPS)
9     video_length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
10
11    # Add buffer time to each frame range
12    frame_ranges_with_buffer = [
13        (
14            max(0, start_frame - int(fps * settings.buffer_before))
15        ),
16        min(video_length, end_frame + int(fps * settings.
17        buffer_after)),
18    ]
19
20    # Merge overlapping frame ranges
21    merged_ranges: List[Tuple[int, int]] = []
22    for start_frame, end_frame in frame_ranges_with_buffer:
23        if not merged_ranges or start_frame > merged_ranges

```

```

23 [-1][1]:
24     merged_ranges.append((start_frame, end_frame))
25     else:
26         merged_ranges[-1] = (
27             merged_ranges[-1][0],
28             max(merged_ranges[-1][1], end_frame),
29         )
30     return merged_ranges

```

```

1 def update_time_prediction(
2     self, progress: int, video_num: int, num_videos: int
3 ) -> None:
4     """Update the time prediction label.
5
6     Args:
7         progress: The progress of the current task.
8         video_num: The number of the current video.
9         num_videos: The total number of videos.
10    """
11    current_time = time.time()
12    # Don't update more than once per second
13    if current_time - self.last_time_update < 1.0:
14        return
15    self.last_time_update = current_time
16
17    total_elapsed_time = current_time - self.start_time
18    video_elapsed_time = current_time - self.video_start_time
19
20    if progress > 0:
21        # Calculate the remaining time for the current video
22        time_left_current_video = video_elapsed_time * (100 -
23        progress) / progress
24
25        if video_num > 0:
26            # Calculate the average time spent per video for the
27            # videos processed so far
28            avg_time_per_video = (
29                total_elapsed_time - video_elapsed_time
30            ) / video_num
31
32            # Estimate the time left for the remaining videos
33            time_left_remaining_videos = avg_time_per_video * (
34                num_videos - (video_num + 1)
35            )
36        else:
37            # If this is the first video, estimate the total time
38            # for the current video
39            total_time_current_video = video_elapsed_time * 100 /
40            progress
41
42            # Use the total time for the current video as an
43            # estimate for the remaining videos
44            time_left_remaining_videos = total_time_current_video
45            * (num_videos - 1)
46
47            # Calculate the total time left
48            time_left = time_left_current_video +

```

```

time_left_remaining_videos
43
44     time_left_str = str(timedelta(seconds=int(time_left)))
45     self.update_time_prediction_sig.emit(f"Total Time Left: {
time_left_str}")

```

```

1 """A timer class to time code blocks."""
2 import time
3 from typing import Any
4
5 from app.logger import get_logger
6
7 logger = get_logger()
8
9
10 class Timer:
11     """A timer class to time code blocks."""
12
13     def __init__(self, name: str) -> None:
14         self.name: str = name
15         self.start: float = 0
16
17     def __enter__(self) -> None:
18         self.start = time.time()
19
20     def __exit__(self, exc_type: Any, exc_value: Any,
exc_traceback: Any) -> None:
21         if exc_type is not None:
22             # logger.error("Exception raised in timer block",
exc_info=True)
23             return
24
25         delta_time = time.time() - self.start
26
27         logger.info("%s took %s seconds", self.name, delta_time)

```

```

1 # pylint: skip-file
2 # mypy: ignore-errors
3
4 import logging
5 import os
6 import sys
7 from pathlib import Path
8
9 import torch
10
11 from app import settings
12 from app.detection import detection
13 from app.detection.batch_yolov8 import BatchYolov8
14 from app.logger import create_logger, get_logger
15 from app.timer import Timer
16 from app.video_processor import video_processor
17
18 MODELS_DIR = "data/models"
19
20 MODEL_NAMES = [
21     "yolov8n.pt",
22     "yolov8s.pt",

```

```

23     "yolov8m.pt",
24     "yolov8l.pt",
25     "yolov8x.pt",
26 ]
27 BATCH_SIZES = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
28 # BATCH_SIZES = [1024]
29
30 DEVICE = "cuda:0"
31
32 VIDEO_INPUT = Path("data/input.mp4")
33 VIDEO_OUTPUT = Path("data/output.mp4")
34
35 log = get_logger()
36
37
38 def perform_test(
39     model: BatchYolov8, batch_size: int = 64, max_batches_to_queue
40     : int = 4
41 ) -> None:
42     with Timer(f"Processing video (batch_size={batch_size}, device
43     ={DEVICE})"):
44         frames_with_fish = detection.process_video(
45             model=model,
46             video_path=VIDEO_INPUT,
47             batch_size=batch_size,
48             output_path=None,
49         )
50
51         # frame_ranges = detection.detected_frames_to_range(
52         frames_with_fish, frame_buffer=3)
53
54         # with Timer("Cutting video"):
55         # Cut the video to the detected frames
56         # video_processor.cut_video(VIDEO_INPUT, VIDEO_OUTPUT,
57         frame_ranges)
58
59 def main() -> int:
60     create_logger(level=logging.INFO)
61     settings.setup()
62
63     # MODEL_NAMES.reverse()
64     for model_name in MODEL_NAMES:
65         try:
66             # Clear CUDA cache to free up memory from the last
67             model/iteration
68             log.info("Initializing %s", model_name)
69             model = BatchYolov8(
70                 Path(os.path.join(MODELS_DIR, model_name)),
71                 DEVICE,
72             )
73
74             except Exception as err:
75                 log.error("Failed to load model %s: %s", model_name,
76                 err)
77
78                 return 1
79
80     for batch_size in BATCH_SIZES:

```

```

75         try:
76             perform_test(model, batch_size=batch_size,
max_batches_to_queue=4)
77         except RuntimeError as err:
78             if "CUDA out of memory" in str(err):
79                 log.warning("Ran out of memory, skipping batch
size %s", batch_size)
80             else:
81                 log.error("Failed to process video: %s", err)
82                 break
83
84         except Exception as err:
85             log.error("Failed to process video: %s", err)
86             break # might be due to OOM
87     return 0
88
89
90 if __name__ == "__main__":
91     sys.exit(main())

```

```

1  """Settings module for the application."""
2
3  import sys
4  from types import ModuleType
5  from typing import Any, Dict, Tuple
6
7  from PyQt6.QtCore import QSettings
8
9  from app.logger import get_logger
10
11 # region Settings
12
13 # Window settings
14 window_width: int = 700
15 window_height: int = 400
16
17 # File settings
18 open_path: str = "" # Path to open file
19 save_path: str = "" # Path to save file
20 buffer_before: int = 0
21 buffer_after: int = 0
22 keep_original: bool = True
23
24 # Advanced settings
25 get_report: bool = False
26
27 report_format: str = "CSV"
28
29 batch_size: int = 8
30
31 prediction_threshold: int = 50
32
33 box_around_fish: bool = False
34
35 video_crf: int = 23
36
37 max_detections: int = 100
38

```



```

39 frame_buffer_seconds: int = 1
40
41 # endregion
42
43 #
44     #
45 # key: name, value: ((default) value, type)
46 __entries: Dict[str, Tuple[Any, type]] = {}
47
48 # Create __settings as QSettings and store it as ini in data
49     folder
50 __settings = QSettings("settings.ini", QSettings.Format.IniFormat)
51
52 __logger = get_logger()
53
54 # This is a hack to get the above module variables
55 module_vars = locals()
56
57
58 def __setup_entries() -> None:
59     """Adds all the entries to the settings entries."""
60     # use the above variables and access them through the module
61
62     to_delete = []
63     for name, value in module_vars.items():
64         if name.startswith("__"):
65             continue
66
67         # Check if the value is a variable of the supported types
68         if not isinstance(value, (int, float, str, bool)):
69             continue
70
71         __logger.debug("Adding entry %s with default value %s",
72 name, value)
73         __add_entry(name, value, type(value))
74         to_delete += [name]
75
76     # Delete the attributes so it doesn't get exported, and we
77     force everything through __getattr__
78     for name in to_delete:
79         del module_vars[name]
80
81 def __add_entry(name: str, default_value: Any, value_type: type)
82 -> None:
83     """Adds a new entry to the settings entries.
84
85     Args:
86         name: The name of the entry as it will be exported by the
87         module and in the ini file.
88         default_value: The default value
89         value_type: The type of the value
90
91     Raises:
92         ValueError: If the entry already exists

```

```

90     ValueError: If the default value is not of specified type
91     """
92     if name in __entries:
93         raise ValueError(f"Entry {name} already exists")
94
95     # We could infer the type, but better to be explicit and raise
96     # an error if it's wrong
97     if not isinstance(default_value, value_type):
98         raise ValueError(f"Default value {default_value} is not of
99         type {value_type}")
100
101     __entries[name] = (default_value, value_type)
102
103 # TODO: rename
104 def __populate_entries_from_ini_config() -> None:
105     """Reads the values from the ini file and sets the values in
106     the module."""
107
108     for name, (default_value, value_type) in __entries.items():
109         value = __settings.value(name, default_value, value_type)
110
111         # If the value is not of the correct type, use the default
112         # value
113         if not isinstance(value, value_type):
114             value = default_value
115             __logger.warning(
116                 "Value for %s is not of type %s, using default
117                 value", name, value_type
118             )
119
120             __logger.debug("Setting %s to %s", name, value)
121             __entries[name] = (value, value_type)
122
123 class SettingsModule(ModuleType): # pylint: disable=too-few-
124     public-methods
125     """A subclass of the settings module that overrides
126     __setattr__"""
127
128     def __setattr__(self, name: str, value: Any) -> None:
129         # Access the entries through the module dict
130         entries = sys.modules[__name__].__dict__["__entries"]
131         settings = sys.modules[__name__].__dict__["__settings"]
132         logger = sys.modules[__name__].__dict__["__logger"]
133
134         if name in entries:
135             entry_type = entries[name][1]
136             entries[name] = (value, entry_type)
137
138             if not isinstance(value, entry_type):
139                 raise ValueError(
140                     f"[{name}] Value {value} is not of type {
141                     entries[name][1]}"
142                 )
143
144             settings.setValue(name, value)
145             logger.debug("Storing %s as %s", name, value)

```

```

140         settings.sync()
141     else:
142         super().__setattr__(name, value)
143
144
145 # Override the __getattr__ method of the settings module
146 def __getattr__(name: str) -> Any:
147     if name in __entries:
148         return __entries[name][0]
149     raise AttributeError(f"Module {__name__} has no attribute {
name}")
150
151
152 def setup() -> None:
153     """Sets up the settings module."""
154
155     __logger.debug("Settings stored at %s", __settings.fileName())
156
157     __setup_entries()
158     __populate_entries_from_ini_config()
159
160     # Replace the settings module with a subclass that overrides
__setattr__
161     sys.modules[__name__].__class__ = SettingsModule

```

```

1 # pylint: skip-file
2 # mypy: ignore-errors
3
4 """Module to generate a dataset for training and testing."""
5
6 import multiprocessing as mp
7 import os
8 import random
9 import re
10 import shutil
11 import xml.etree.ElementTree as ET
12 import zipfile
13 from concurrent.futures import ThreadPoolExecutor
14 from pathlib import Path
15 from typing import Dict, List, Tuple
16
17 import cv2
18 import yaml
19 from tqdm import tqdm
20
21 CVAT_EXPORTS_FOLDER = Path(r"D:\dataset_temp\cvat_exports")
22 DATASET_FOLDER = Path(r"D:\dataset_temp\generated_test")
23 SOURCE_VIDEO_FOLDERS = [
24     Path(r"X:\Myggbukta 2022"),
25     Path(r"X:\DISK1 - Hoyregga 17+18 og myggbukta 2020 mai NTNU"),
26 ]
27 BACKGROUND_IMAGE_PERCENTAGE = 0.1
28 TRAIN_SPLIT = 0.8
29 PNG_QUALITY = 3 # 0-9 where 0 is the best quality
30 # VAL_SPLIT = 0.3 # Not used, just the rest of train split
31 MAX_WORKERS = 8
32
33 CLASSES = [

```

```

34     "Gjedde",
35     "Gullbust",
36     "Rumpetroll",
37     "Stingsild",
38     "Orekyt",
39     "Abbor",
40     "Brasme",
41     "Mort",
42     "Vederbuk",
43     "Frosk",
44     "Annen fisk",
45 ]
46
47
48 def get_video_filename(annotation_xml: Path) -> str:
49     tree = ET.parse(annotation_xml)
50     root: ET.Element = tree.getroot()
51     assert root is not None, "Could not parse xml file"
52
53     task = root.find("meta").find("task")
54     return task.find("source").text
55
56
57 def extract_annotations(
58     cvat_exports_folder: Path, output_folder: Path,
59     processed_counter: mp.Value
60 ) -> None:
61
62     # NOTE: We do this because background images are randomly
63     #       selected and we
64     #       don't want to add more and more background images to
65     #       the dataset if run multiple times
66     #       also other parameters and the dataset itself might
67     #       have changed.
68     if os.path.exists(output_folder):
69         shutil.rmtree(output_folder, ignore_errors=True)
70
71     annotation_folder = output_folder / "cvat_annotations"
72
73     # Creates output folder and annotation folder
74     os.makedirs(annotation_folder)
75
76     def process_zip_file(
77         zip_filepath: Path, id: int, total: int, processed_counter
78         : mp.Value
79     ) -> None:
80
81         try:
82             with zipfile.ZipFile(zip_filepath, "r") as zip_file:
83                 zip_file.extract(
84                     "annotations.xml",
85                     path=annotation_folder / f"{id:04d}",
86                 )
87                 annotation_path = annotation_folder / f"{id:04d}"
88                 / "annotations.xml"
89                 video_filename = get_video_filename(
90                     annotation_path)
91                 video_filename = os.path.splitext(video_filename)

```

```

[0]
85
86         with processed_counter.get_lock():
87             tqdm.write(f"Started processing {zip_filepath.
name}")
88
89             try:
90                 generate_yolo_dataset(
91                     output_folder / video_filename,
92                     annotation_path,
93                     BACKGROUND_IMAGE_PERCENTAGE,
94                 )
95             except FileNotFoundError as err:
96                 print(err)
97                 raise
98
99         with processed_counter.get_lock():
100             processed_counter.value += 1
101             tqdm.write(
102                 f"Finished processing {zip_filepath.name} ({
processed_counter.value}/{total})"
103             )
104         except zipfile.BadZipFile as err:
105             print(err)
106             raise
107         except Exception as err:
108             print(err)
109             raise
110
111     # Create a ThreadPoolExecutor to parallelize the processing
112     with ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
113         zip_filepaths = [
114             cvat_exports_folder / zip_filename
115             for zip_filename in os.listdir(cvat_exports_folder)
116             if zip_filename.endswith(".zip")
117         ]
118         total_zip_files = len(zip_filepaths)
119
120         for position, zip_filepath in enumerate(zip_filepaths):
121             executor.submit(
122                 process_zip_file,
123                 zip_filepath,
124                 position + 1,
125                 total_zip_files,
126                 processed_counter,
127             )
128
129 def parse_xml_annotation(
130     xml_file: str,
131 ) -> Tuple[
132     str,
133     Tuple[int, int],
134     int,
135     Dict[int, Dict[str, List[Tuple[float, float, float, float]]]],
136 ]:
137     tree = ET.parse(xml_file)
138     root: ET.Element = tree.getroot()
139     assert root is not None, "Could not parse xml file"

```

```

140
141 task = root.find("meta").find("task")
142 video_filename = task.find("source").text
143
144 video_original_size = (
145     float(task.find("original_size").find("width").text),
146     float(task.find("original_size").find("height").text),
147 )
148
149 frame_count = int(task.find("size").text)
150
151 # dictionary to store annotations, with frame number as key
152 annotations: Dict[int, Dict[str, List[Tuple[float, float,
float, float]]]] = {}
153
154 # Add image annotations
155 for image in root.findall("image"):
156     frame_number = int(image.attrib["id"])
157     annotations[frame_number] = {}
158     for box in image.findall("box"):
159         label = box.attrib["label"]
160         occluded = box.attrib["occluded"]
161         outside = box.attrib["outside"]
162         # Skip occluded and outside boxes
163         if occluded == "1" or outside == "1":
164             continue
165
166         xtl = float(box.attrib["xtl"])
167         ytl = float(box.attrib["ytl"])
168         xbr = float(box.attrib["xbr"])
169         ybr = float(box.attrib["ybr"])
170         if label not in annotations[frame_number]:
171             annotations[frame_number][label] = []
172         annotations[frame_number][label].append((xtl, ytl, xbr
, ybr))
173
174 # Add track annotations
175 for track in root.findall("track"):
176     label = track.attrib["label"]
177     for box in track.findall("box"):
178         frame_number = int(box.attrib["frame"])
179         occluded = box.attrib["occluded"]
180         outside = box.attrib["outside"]
181         # Skip occluded and outside boxes
182         if occluded == "1" or outside == "1":
183             continue
184
185         xtl = float(box.attrib["xtl"])
186         ytl = float(box.attrib["ytl"])
187         xbr = float(box.attrib["xbr"])
188         ybr = float(box.attrib["ybr"])
189         if frame_number not in annotations:
190             annotations[frame_number] = {}
191         if label not in annotations[frame_number]:
192             annotations[frame_number][label] = []
193         annotations[frame_number][label].append((xtl, ytl, xbr
, ybr))
194
195 return video_filename, video_original_size, frame_count,

```

```

195 annotations
196
197 def generate_yolo_dataset(
198     yolo_dataset_folder: Path,
199     annotation_path: Path,
200     background_image_percentage: float,
201     background_image_no_annotation_buffer: int = 50,
202 ) -> None:
203     os.makedirs(yolo_dataset_folder)
204
205     (
206         video_filename,
207         video_resolution,
208         frame_count,
209         annotations,
210     ) = parse_xml_annotation(annotation_path)
211
212     def get_video_path(video_filename: str) -> Path:
213         """
214         This is kinda hacky, but it works.
215         Should have globbed it directly, but escaping the brackets
216         was a pain.
217         """
218         for source_video_folder in SOURCE_VIDEO_FOLDERS:
219             search_name = os.path.splitext(video_filename)[0]
220             # NOTE: we replace underscores with spaces due to
221             handbrake naming
222             search_name = re.sub(r"_", " ", search_name).upper()
223             files = source_video_folder.glob("**/*.mp4")
224             for file in files:
225                 if search_name in re.sub(r"_", " ", file.name).
226                 upper():
227                     return file
228
229             raise FileNotFoundError(
230                 f"Could not find {video_filename} in any of the source
231                 video folders"
232             )
233
234     video_path = get_video_path(video_filename)
235
236     def extract_frames(video_path: Path, background_frames: List[
237     int]) -> None:
238         video = cv2.VideoCapture(str(video_path))
239
240         frames_to_extract = list(
241             filter(
242                 lambda n: n in annotations or n in
243                 background_frames, range(frame_count)
244             )
245         )
246
247         for frame_number in frames_to_extract:
248             video.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
249             success, frame = video.read()
250             if success:
251                 cv2.imwrite(

```

```

246         str(yolo_dataset_folder / f"{frame_number:06}."
247         png"),
248         frame,
249         [int(cv2.IMWRITE_PNG_COMPRESSION), PNG_QUALITY
250         ],
251     )
252     video.release()
253
254     def determine_background_frames(max_frame_number: int) -> List
255     [int]:
256         """
257         Determines background frames based on the
258         background_image_percentage.
259         It needs to have at least
260         background_image_no_annotation_buffer amount of frames before
261         and after it to be considered a background frame.
262
263         This is to prevent the background frames from being too
264         close to the
265         annotated frames and containing half a fish.
266
267         Needs to be verified by a human.
268         """
269         # Store annotated frame numbers in a set
270         annotated_frames = set(annotations.keys())
271
272         # Create a set of frame numbers that are too close to
273         annotated frames
274         exclude_frames = set(
275             frame_number
276             for annotated_frame_number in annotated_frames
277             for frame_number in range(
278                 max(0, annotated_frame_number -
279                 background_image_no_annotation_buffer),
280                 min(
281                     max_frame_number,
282                     annotated_frame_number +
283                     background_image_no_annotation_buffer + 1,
284                 ),
285             )
286         )
287
288         # Create a set of background frames that are not too close
289         to annotated frames
290         background_frames = (
291             set(range(max_frame_number + 1)) - annotated_frames -
292             exclude_frames
293         )
294
295         # Convert the background frames set to a list and shuffle
296         it
297         background_frames = list(background_frames)
298         random.shuffle(background_frames)
299
300         # Determine the amount of background frames to use
301         n_background_frames = int(background_image_percentage *
302         len(annotations))

```



```

291     return background_frames[:n_background_frames]
292
293     # Iterate all the annotations and generate the yolo dataset
294     # Each entry of annotations contain the frame number as key
    and a dictionary of labels as value, with the bounding boxes as
    value
295     video_width, video_height = video_resolution
296
297     for frame_number, frame_annotations in annotations.items():
298         # Create annotation file
299         annotation_file = yolo_dataset_folder / f"{frame_number
:06}.txt"
300         with open(annotation_file, "w", encoding="utf-8") as file:
301             for label, boxes in frame_annotations.items():
302                 for box in boxes:
303                     xtl, ytl, xbr, ybr = box
304                     x_center = (xtl + xbr) / 2
305                     y_center = (ytl + ybr) / 2
306                     width = xbr - xtl
307                     height = ybr - ytl
308
309                     # Normalize the coordinates and size to be
between 0 and 1
310                     x_center /= video_width
311                     y_center /= video_height
312                     width /= video_width
313                     height /= video_height
314
315                     # Clamp the values to be between 0 and 1 (cvat
annotations are sometimes OOB when starting new tracks)
316                     x_center = max(0, min(1, x_center))
317                     y_center = max(0, min(1, y_center))
318                     width = max(0, min(1, width))
319                     height = max(0, min(1, height))
320
321                     file.write(
322                         f"{CLASSES.index(label)} {x_center} {
y_center} {width} {height}\n"
323                     )
324
325     background_frames = determine_background_frames(frame_count)
326
327     # Write empty annotation files for the background frames
328     for frame_number in background_frames:
329         annotation_file = yolo_dataset_folder / f"{frame_number
:06}.txt"
330         with open(annotation_file, "w", encoding="utf-8") as file:
331             pass
332
333     # Extract annotation frames and background frames
334     extract_frames(video_path, background_frames)
335
336
337 def split_train_val(dataset_path: Path, train_split: float) ->
None:
338     all_images: List[Path] = []
339
340     # Collect all image paths from subfolders

```

```

341     for folder in dataset_path.iterdir():
342         if folder.is_dir():
343             # Glob for .png or .jpg
344             images = list(folder.glob("*.png"))
345             images.extend(folder.glob("*.jpg"))
346
347             # Make it relative to the dataset folder
348             # images = [img.relative_to(dataset_path) for img in
images]
349             all_images.extend(images)
350
351     # Shuffle the images and split them into train and val sets
352     random.shuffle(all_images)
353     train_size = int(len(all_images) * train_split)
354
355     print(f"Total images: {len(all_images)}")
356     print(f"Train size: {train_size}, Val size: {len(all_images) -
train_size}")
357     train_images = all_images[:train_size]
358     val_images = all_images[train_size:]
359
360     # Write the train.txt and val.txt files
361     with open(dataset_path / "train.txt", "w", encoding="utf-8")
as train_file:
362         for img in train_images:
363             train_file.write(f"{img}\n")
364
365     with open(dataset_path / "val.txt", "w", encoding="utf-8") as
val_file:
366         for img in val_images:
367             val_file.write(f"{img}\n")
368
369
370 def merge_datasets(dataset_a: Path, dataset_b: Path, output_folder
: Path):
371     """
372     This will merge the train.txt and val.txt files from two
373     datasets into one dataset.
374     It will also copy the obj.names and obj.data files from the
375     first dataset.
376     And it will output the new train.txt and val.txt files in the
377     output folder.
378     """
379     train_a = dataset_a / "train.txt"
380     val_a = dataset_a / "val.txt"
381     train_b = dataset_b / "train.txt"
382     val_b = dataset_b / "val.txt"
383
384     with open(train_a, "r", encoding="utf-8") as file:
385         train_a_lines = file.readlines()
386     with open(val_a, "r", encoding="utf-8") as file:
387         val_a_lines = file.readlines()
388     with open(train_b, "r", encoding="utf-8") as file:
389         train_b_lines = file.readlines()
390     with open(val_b, "r", encoding="utf-8") as file:
391         val_b_lines = file.readlines()
392
393     # FIXME: This is a hack to fix the paths in the train.txt and

```

```

val.txt files
391 # I only need it for the yolo dataset (b) atm because prepends
    # the data/ folder to the paths
392 """
393 def remove_data_path(lines: List[str]) -> List[str]:
394     # Remove data/ from each line by removing first 5
characters
395     lines = list(map(lambda line: line[5:], lines))
396     return [f"{ line}" for line in lines]
397
398 train_b_lines = remove_data_path(train_b_lines)
399 val_b_lines = remove_data_path(val_b_lines)
400 """
401
402 # Merge the train and val files
403 train_lines = train_a_lines + train_b_lines
404 val_lines = val_a_lines + val_b_lines
405
406 # Shuffle the lines
407 random.shuffle(train_lines)
408 random.shuffle(val_lines)
409
410 # Write the new train and val files
411 with open(output_folder / "train.txt", "w", encoding="utf-8")
as file:
412     file.writelines(train_lines)
413
414 with open(output_folder / "val.txt", "w", encoding="utf-8") as
file:
415     file.writelines(val_lines)
416
417 # shutil.copy(dataset_a / "obj.names", output_folder / "obj.
names")
418 # shutil.copy(dataset_a / "obj.data", output_folder / "obj.
data")
419
420 # Create the dataset.yaml config file
421 dataset_yaml = {
422     "path": str(output_folder),
423     "train": "train.txt",
424     "val": "val.txt",
425     "names": CLASSES,
426 }
427 with open(output_folder / "dataset.yaml", "w", encoding="utf-8
") as yaml_file:
428     yaml.dump(dataset_yaml, yaml_file, default_flow_style=
False)
429
430
431 if __name__ == "__main__":
432     """
433     processed_counter = mp.Value("i", 0)
434     try:
435         extract_annotations(CVAT_EXPORTS_FOLDER, DATASET_FOLDER,
processed_counter)
436     except FileNotFoundError as e:
437         print(e)
438     """

```

```

439 split_train_val(DATASET_FOLDER, TRAIN_SPLIT)
440 # generate_obj_files(DATASET_FOLDER)
441
442 # Split the downloaded yolo dataset into train and val
443 split_train_val(Path(r"D:\dataset_temp\
yolo_updated_with_images"), TRAIN_SPLIT)
444 merge_datasets(
445     DATASET_FOLDER, # Path(r"C:\Users\benja\Documents\
datasets\nina_yolo_new"),
446     Path(r"D:\dataset_temp\yolo_updated_with_images"),
447     output_folder=Path(r"D:\dataset_temp\spliced"),
448 )

```

```

1 """Script to find the distribution of annotations in a yolo
dataset."""
2 # pylint: disable=missing-function-docstring
3 import csv
4 from pathlib import Path
5
6 from tqdm import tqdm
7
8 NAMES_FILE_NAME = "obj.names"
9 # DATA_FOLDER_NAME = "obj_train_data"
10
11
12 DATASET_PATH = Path(r"D:\dataset_temp\spliced")
13
14
15 def load_names() -> list[str]:
16     with (DATASET_PATH / NAMES_FILE_NAME).open("r", encoding="utf
-8") as file:
17         dataset_names = file.read().splitlines()
18     return dataset_names
19
20
21 def get_annotation_distribution() -> list[tuple[str, int]]:
22     annotations_dist: list[int] = [0 for _ in names]
23     # Get subdirectories of DATASET_PATH
24     subdirectories = [
25         subdirectory for subdirectory in DATASET_PATH.iterdir() if
subdirectory.is_dir()
26     ]
27     # Get all .txt files in the subdirectories
28     filenames = [
29         filename
30         for subdirectory in subdirectories
31         for filename in subdirectory.glob("*.txt")
32     ]
33     with tqdm(total=len(filenames), desc="Processing annotations")
as progress_bar:
34         for filename in filenames:
35             with filename.open("r") as file:
36                 for line in file:
37                     try:
38                         class_id, _, _, _, _ = line.split()
39                         annotations_dist[int(class_id)] += 1
40                     except ValueError as err:
41                         print(f"Error processing {filename}: {err}")

```

```

42     ")
43         progress_bar.update(1)
44     return list(zip(names, annotations_dist))
45
46
47 def write_annotation_distribution_to_csv(
48     annotations_dist: list[tuple[str, int]]
49 ) -> None:
50     with open("annotations_dist.csv", "w", newline="", encoding="
51         utf-8") as file:
52         writer = csv.writer(file)
53         writer.writerow(["Name", "Num annotations"])
54         writer.writerows(annotations_dist)
55
56 names = load_names()
57
58 annotations = get_annotation_distribution()
59
60 print(annotations)
61
62 write_annotation_distribution_to_csv(annotations)

```

```

1 # pylint: skip-file
2 # mypy: ignore-errors
3 import concurrent.futures
4 import hashlib
5 import os
6 import pickle
7 import queue
8 from pathlib import Path
9
10 import cv2
11 import ffmpeg
12 import numpy as np
13 import tqdm
14 from imagehash import dhash, phash
15 from PIL import Image
16
17
18 def extract_images_from_video(video_path, output_folder):
19     video_name = video_path.stem
20     cap = cv2.VideoCapture(str(video_path))
21
22     frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
23
24     # Create the output folder if it doesn't exist
25     os.makedirs(output_folder, exist_ok=True)
26
27     for i in range(frame_count):
28         ret, frame = cap.read()
29         if ret:
30             output_file_path = os.path.join(
31                 output_folder, f"{video_name}_frame_{i:04d}.jpg"
32             )
33             cv2.imwrite(output_file_path, frame)
34         else:

```

```

35         print(f"Failed to read frame {i} from {video_path}")
36
37     cap.release()
38
39
40 def extract_images_from_video_ffmpeg(video_path, output_folder):
41     video_name = video_path.stem
42
43     # Create the output folder if it doesn't exist
44     os.makedirs(output_folder, exist_ok=True)
45
46     output_file_pattern = os.path.join(output_folder, f"{
47 video_name}_frame_%04d.jpg")
48
49     try:
50         (
51             ffmpeg.input(str(video_path))
52             .output(output_file_pattern, format="image2", vcodec="
53 mjpeg")
54             .run(capture_stdout=True, capture_stderr=True)
55         )
56     except ffmpeg.Error as e:
57         print(
58             f"Error occurred while extracting frames from {
59 video_path}: {e.stderr.decode()}"
60         )
61
62 def md5_hash(image_path):
63     with open(image_path, "rb") as f:
64         file_hash = hashlib.md5()
65         while chunk := f.read(8192):
66             file_hash.update(chunk)
67         return file_hash.hexdigest()
68
69 hash_func = phash
70
71 def load_hashes(yolo_dataset_path, cache_file_path, batch_size=10)
72 :
73     yolo_hashes = set()
74
75     cache_file = Path(cache_file_path)
76     if cache_file.exists():
77         with open(cache_file, "rb") as f:
78             yolo_hashes = pickle.load(f)
79         print("Loaded", len(yolo_hashes), "YOLO hashes")
80     else:
81         # Find both .jpg or .PNG files
82         image_files = list(Path(yolo_dataset_path).glob("*.jpg"))
83         + list(
84             Path(yolo_dataset_path).glob("*.PNG")
85         )
86         num_images = len(image_files)
87         print("Found", num_images, "images in YOLO dataset")
88         progress_bar = tqdm.tqdm(total=num_images, desc="
89 Processing YOLO images")

```

```

87
88     for i in range(0, num_images, batch_size):
89         batch = image_files[i : i + batch_size]
90         for image_path in batch:
91             image = Image.open(image_path)
92             yolo_hashes.add(
93                 hash_func(image, hash_size=16)
94                 ) # You can use dhash() if preferred
95
96         progress_bar.update(len(batch))
97
98     progress_bar.close()
99     print("Found", len(yolo_hashes), "unique images in YOLO
dataset")
100     with open(cache_file_path, "wb") as f:
101         pickle.dump(yolo_hashes, f)
102     print("Saved hashes to", cache_file_path)
103
104     return yolo_hashes
105
106
107 def load_processed_videos(file_path):
108     processed_videos = set()
109     cache_file = Path(file_path)
110     if cache_file.exists():
111         with open(cache_file, "rb") as f:
112             processed_videos = pickle.load(f)
113         print("Loaded", len(processed_videos), "processed videos")
114     else:
115         print("Processed videos file not found")
116     return processed_videos
117
118
119 def save_processed_videos(processed_videos, file_path):
120     with open(file_path, "wb") as f:
121         pickle.dump(processed_videos, f)
122
123
124 def process_video(video_path, yolo_hashes, progressBars_queue):
125     try:
126         progress_bar = progressBars_queue.get()
127
128         video_name = video_path.stem
129         cap = cv2.VideoCapture(str(video_path))
130
131         fps = 1
132         frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
133         fps_original = int(cap.get(cv2.CAP_PROP_FPS))
134         interval = fps_original // fps
135
136         progress_bar.total = frame_count // interval
137         progress_bar.n = 0
138         progress_bar.set_description_str(f"Processing {video_name}
")
139
140         progress_bar.refresh()
141         progress_bar.reset()
142
143         for i in range(0, frame_count, interval):

```

```

143         cap.set(cv2.CAP_PROP_POS_FRAMES, i)
144         ret, frame = cap.read()
145
146         if ret:
147             # _, frame_buffer = cv2.imencode(".jpg", frame)
148             # frame_hash = hashlib.md5(frame_buffer.tobytes())
149             .hexdigest()
150             frame_pil = Image.fromarray(cv2.cvtColor(frame,
151             cv2.COLOR_BGR2RGB))
152             frame_hash = hash_func(frame_pil, hash_size=16)
153
154             if frame_hash in yolo_hashes:
155                 progress_bar_queue.put(progress_bar)
156                 return True
157
158         progress_bar.update()
159
160         cap.release()
161         progress_bar_queue.put(progress_bar)
162         return False
163     except Exception as e:
164         print(f"Error occurred while processing {video_path}: {e}")
165     )
166     return False
167
168 def find_annotated_videos(video_folder, processed_videos,
169 output_file, max_workers=4):
170     annotated_videos = set()
171     # Keep track of processed videos to avoid processing the same
172     video multiple times
173     video_files = list(
174         Path(video_folder).rglob("*.mp4")
175     ) # Adjust the extension if necessary
176     # Remove files in "RECYCLE.BIN" folders
177     video_files = [
178         video_path for video_path in video_files if "RECYCLE.BIN"
179         not in str(video_path)
180     ]
181     # Remove if in processed_videos
182     video_files = [
183         video_path for video_path in video_files if video_path not
184         in processed_videos
185     ]
186     # Randomize the order of videos
187     # np.random.shuffle(video_files)
188
189     print("Found", len(video_files), "videos in folder")
190
191     progress_bar_queue = queue.Queue()
192     progress_bar_queue.put(progress_bar)
193     for progress_bar in progress_bar_queue:
194         progress_bar_queue.put(progress_bar)

```



```

193     with concurrent.futures.ThreadPoolExecutor(max_workers=
max_workers) as executor:
194         futures = {
195             executor.submit(
196                 process_video, video_path, yolo_hashes,
progress_bars_queue
197             ): video_path
198             for video_path in video_files
199         }
200
201         overall_progress = tqdm.tqdm(
202             total=len(video_files), desc="Overall progress",
position=max_workers
203         )
204
205         for future in concurrent.futures.as_completed(futures):
206             video_path = futures[future]
207             if future.result():
208                 annotated_videos.add(video_path.stem)
209                 # Write the video name to a file
210                 with open(output_file, "a") as f:
211                     f.write(video_path.stem + " " + str(video_path
) + "\n")
212
213                 processed_videos.add(video_path)
214
215                 save_processed_videos(processed_videos,
processed_videos)
216                 overall_progress.update(1)
217                 overall_progress.set_description_str(
218                     "Overall progress (" + str(len(annotated_videos))
+ " annotated videos)"
219                 )
220
221             for progress_bar in progress_bars:
222                 progress_bar.close()
223
224         overall_progress.close()
225         return annotated_videos
226
227
228 video_folder = r"path/to/videos" # Videos to check
229 reference_images = r"path/to/reference/images" # Reference images
to check against
230 reference_images_hashes = "hashes.pkl" # Cache file for reference
images
231 processed_videos = "processed_videos.pkl" # Cache file for
processed videos (to avoid processing the same video multiple
times)
232 output_file = "annotated_videos.txt" # Output file with annotated
videos
233
234 annotated_videos = set()
235
236 yolo_hashes = load_hashes(reference_images,
reference_images_hashes)
237
238 processed_videos = load_processed_videos(processed_videos)

```

```

239 annotated_videos = find_annotated_videos(
240     video_folder, processed_videos, output_file, max_workers=4
241 )
242 )
243
244 print("Annotated videos:", annotated_videos)

```

```

1  """Main file for our application"""
2  import logging
3  import os
4  from logging.handlers import TimedRotatingFileHandler
5
6  LOGGER_NAME = "log"
7  LOG_PATH = "app/log"
8
9
10 def __create_file_handler(
11     formatter: logging.Formatter, filename: str
12 ) -> TimedRotatingFileHandler:
13     """Set up logging to file to rotate every midnight and set
14     formatter
15
16     Returns:
17         TimedRotatingFileHandler: The file handler
18     """
19     handler = TimedRotatingFileHandler(
20         f"{LOG_PATH}/{filename}",
21         when="midnight",
22         backupCount=10,
23     )
24
25     # Set up custom naming for log files
26     def namer(default_name: str) -> str:
27         base_filename, ext, filedate = default_name.split(".")
28         return f"{base_filename}.{filedate}.{ext}"
29
30     handler.suffix = "%d-%m-%Y"
31     handler.namer = namer
32     handler.setLevel(logging.DEBUG)
33     handler.setFormatter(formatter)
34     return handler
35
36 def __create_console_handler(formatter: logging.Formatter) ->
37 logging.Handler:
38     """Set up logging to console"""
39     console_handler = logging.StreamHandler()
40     console_handler.setFormatter(formatter)
41     return console_handler
42
43 # log_location = "main"
44 def create_logger(level: int = logging.DEBUG, filename: str = "
45 logfile.log") -> None:
46     """Creates a logger with a file handler and a console handler
47
48     Args:
49         level: The log level. Defaults to logging.DEBUG.

```

```

49     filename: The log filename. Defaults to "logfile.log".
50     """
51
52     formatter = logging.Formatter(
53         "%(asctime)s %(name)s %(levelname)s %(message)s",
54         datefmt="%H:%M:%S",
55     )
56     # Initialize the logger
57     logger = logging.getLogger(LOGGER_NAME)
58     logger.setLevel(level)
59
60     # create directory for logfiles
61     if not os.path.exists(LOG_PATH):
62         os.makedirs(LOG_PATH)
63
64     # Sets midnight rotation for logger
65     logger.addHandler(__create_file_handler(formatter, filename))
66
67     # Sets console handler for logger
68     logger.addHandler(__create_console_handler(formatter))
69
70     # document that logger is initialized
71     logger.info("Logger initialized")
72
73
74 def get_logger() -> logging.Logger:
75     """Get the logger"""
76     return logging.getLogger(LOGGER_NAME)

```

```

1  """Yolov8 class for running inference on video. """
2  import copy
3  import os
4  from pathlib import Path
5  from typing import Any, Dict, List, Optional, Tuple
6
7  import numpy as np
8  import torch
9  from torch import Tensor
10 from ultralytics.nn.tasks import attempt_load_one_weight
11 from ultralytics.yolo.data.augment import LetterBox
12 from ultralytics.yolo.utils.checks import check_imgsz
13 from ultralytics.yolo.utils.ops import non_max_suppression,
14     scale_boxes
15 from ultralytics.yolo.utils.torch_utils import select_device
16 from app.logger import get_logger
17
18 logger = get_logger()
19
20
21 class BatchYolov8: # pylint: disable=too-many-instance-attributes
22     """Yolov8 class for running inference on video."""
23
24     def __init__( # pylint: disable=too-many-arguments
25         self,
26         weights_path: Path,
27         device: str = "",
28         img_size: int = 640,

```

```

29     conf_thres: float = 0.4,
30     iou_thres: float = 0.5,
31     augment: bool = False,
32     agnostic_nms: bool = False,
33     classes: Optional[List[str]] = None,
34     colors: Optional[List[Tuple[int, int, int]]] = None,
35 ) -> None:
36     try:
37         self.device = select_device(device)
38     except Exception as err:
39         logger.error("Failed to select device", exc_info=err)
40         raise RuntimeError("Failed to select device", err)
41
42     self.weights_name = os.path.split(weights_path)[-1]
43
44     try:
45         (self.model, _) = attempt_load_one_weight(
46             str(weights_path), device=self.device
47         )
48         # self.model = attempt_load(weights_path, device=self.
49 device) V5
50     except Exception as err:
51         logger.error("Failed to load model", exc_info=err)
52         raise RuntimeError("Failed to load model", err) from
53 err
54
55     self.names = (
56         self.model.module.names
57         if hasattr(self.model, "module")
58         else self.model.names
59     )
60     if colors is None:
61         self.colors: List[Tuple[int, int, int]] = [
62             (
63                 np.random.randint(0, 255),
64                 np.random.randint(0, 255),
65                 np.random.randint(0, 255),
66             )
67             for _ in range(len(self.names))
68         ]
69         logger.debug("Color is none, setting random colors.")
70     else:
71         self.colors = colors
72     self.imgsz = check_imgsz(img_size, stride=self.model.
73 stride.max())
74     # self.imgsz = check_img_size(img_size, s=self.model.
75 stride.max()) V5
76     self.conf_thres = conf_thres
77     self.iou_thres = iou_thres
78     self.augment = augment
79     self.agnostic_nms = agnostic_nms
80     self.classes = classes
81     self.half = self.device.type != "cpu"
82     if self.half:
83         self.model.half()
84     if self.device.type != "cpu":
85         self.burn()

```

```

82
83     def prepare_images(
84         self, img_s: List[np.ndarray[Any, Any]] | np.ndarray[Any,
85         Any]
86     ) -> Tensor:
87         """Prepare a batch of images for inference by normalizing
88         and reshaping them.
89
90         Args:
91             img_s: The images to prepare.
92
93         Raises:
94             RuntimeError: If the type of the images is not
95             supported.
96
97         Returns:
98             The prepared images as a torch tensor.
99         """
100        if isinstance(img_s, list):
101            img_list = []
102
103            for img in img_s:
104                img_list += [self.reshape_copy_img(img)]
105
106            img_to_send = self.pad_batch_of_images(img_list)
107        elif isinstance(img_s, np.ndarray):
108            img_to_send = self.reshape_copy_img((np.ndarray(img_s)
109        ))
110        else:
111            print(type(img_s), " is not supported")
112            raise RuntimeError("Not supported type")
113
114        return self.prepare_image(img_to_send)
115
116    def __str__(self) -> str:
117        out = [
118            f"Model: {self.weights_name}",
119            f"Image size: {self.imgsz}",
120            f"Confidence threshold: {self.conf_thres}",
121            f"IoU threshold: {self.iou_thres}",
122            f"Augment: {self.augment}",
123            f"Agnostic nms: {self.agnostic_nms}",
124        ]
125        if self.classes is not None:
126            filter_classes = [self.names[each_class] for
127        each_class in self.classes]
128            out.append(f"Classes filter: {filter_classes}")
129            out.append(f"Classes: {self.names}")
130
131        return "\n".join(out)
132
133    def burn(self) -> None:
134        """Burn in the model for better performance when starting
135        inference."""
136        img = torch.zeros(
137            (1, 3, self.imgsz, self.imgsz), device=self.device
138        ) # init img
139        _ = self.model(img.half() if self.half else img) # run

```

```

once
134
135 def predict_batch(
136     self,
137     img0s: List[Any],
138     imgs: torch.Tensor,
139     max_objects: Optional[Dict[Any, Any]] = None,
140     max_detections: int = 300,
141 ) -> List[Any]:
142     """Predict on a batch of images.
143
144     Args:
145         img0s: The list of images to predict on.
146         max_objects: Max number of objects to return per image
147         for each class.
148
149     Returns:
150         A list of predictions.
151     """
152     # imgs = self.prepare_images(img0s)
153
154     with torch.no_grad():
155         # Run model
156         inf_out, _ = self.model(
157             imgs, augment=self.augment
158         ) # inference and training outputs
159
160         # Run NMS
161         preds = non_max_suppression(
162             inf_out,
163             conf_thres=self.conf_thres,
164             iou_thres=self.iou_thres,
165             max_det=max_detections,
166         )
167
168         batch_output = []
169         for det, img0, img in zip(preds, img0s, imgs):
170             if det is not None and len(det):
171                 det[:, :4] = scale_boxes(img.shape[1:], det[:,
172 :4], img0.shape).round()
173                 min_max_list = self.min_max_list(det)
174                 if min_max_list is not None and max_objects is not
175                 None:
176                     min_max_list = self.max_objects_filter(
177                         min_max_list, max_objects, name_key="name"
178                     )
179                 batch_output.append(min_max_list)
180
181         return batch_output
182
183 def prepare_image(self, original_img: np.ndarray[Any, Any] |
184 List[Any]) -> Tensor:
185     """Prepare image for inference by normalizing and
186     reshaping.
187
188     Args:

```

```

186         original_img: The image to prepare.
187
188     Returns:
189         The prepared image as a torch tensor.
190     """
191     new_img = torch.from_numpy(original_img).to(self.device)
192     new_img = new_img.half() if self.half else new_img.float()
193     new_img /= 255.0 # 0 - 255 to 0.0 - 1.0
194     if new_img.ndimension() == 3:
195         new_img = new_img.unsqueeze(0)
196
197     return new_img
198
199     def reshape_copy_img(self, img: np.ndarray[Any, Any]) -> np.
200     ndarray[Any, Any]:
201         """Reshape and copy image.
202
203     Args:
204         img: The image to reshape and copy.
205
206     Returns:
207         The reshaped and copied image.
208     """
209     _img = LetterBox(stride=32, new_shape=self.imgsz)(image=
210     img)
211     # _img = letterbox(img, new_shape=self.imgsz)[0] V5
212     _img = _img[:, :, ::-1].transpose(2, 0, 1) # BGR to RGB
213     new_img: np.ndarray[Any, Any] = np.ascontiguousarray(_img)
214     # uint8 to float32
215     return new_img
216
217     @staticmethod
218     def pad_batch_of_images(
219         img_list: List[Any], return_np: bool = True
220     ) -> np.ndarray[Any, Any] | List[Any]:
221         """Pad a batch of images to the same size.
222
223     Args:
224         img_list: The list of images to pad.
225         return_np: Whether to return a numpy array or a list
226     of images.
227
228     Returns:
229         The padded images as a numpy array or a list of images
230     .
231     """
232     max_height = 0
233     max_width = 0
234     padded_img_list = []
235     for img in img_list:
236         _, height, width = img.shape
237         max_height = max(max_height, height)
238         max_width = max(max_width, width)
239
240     for img in img_list:
241         padded_img = np.full(
242             (max_height, max_width, 3), (114, 114, 114), dtype
243     =np.uint8

```

```

238     )
239     padded_img = padded_img.transpose(2, 0, 1)
240
241     _, height, width = img.shape
242     offset_width = (max_width - width) // 2
243     offset_height = (max_height - height) // 2
244
245     padded_img[
246         :,
247         offset_height : offset_height + height,
248         offset_width : offset_width + width,
249     ] = img
250     padded_img_list.append(padded_img)
251
252     if return_np:
253         return np.array(padded_img_list)
254     return padded_img_list
255
256     def min_max_list(self, det: Any) -> Optional[List[Any]]:
257         """Create a list of bounding boxes from the detection.
258
259         Args:
260             det: The detection.
261
262         Returns:
263             The list of bounding boxes.
264         """
265         min_max_list = []
266         if det is not None:
267             for i, class_id in enumerate(det[:, -1]):
268                 obj = {
269                     "bndbox": {
270                         "xmin": min(int(det[i][0]), int(det[i][2])
271 ),
272                         "xmax": max(int(det[i][0]), int(det[i][2])
273 ),
274                         "ymin": min(int(det[i][1]), int(det[i][3])
275 ),
276                         "ymax": max(int(det[i][1]), int(det[i][3])
277 ),
278                         "width": max(int(det[i][0]), int(det[i]
279 ] [2]))
280                         - min(int(det[i][0]), int(det[i][2])),
281                         "height": max(int(det[i][1]), int(det[i]
282 ] [3]))
283                         - min(int(det[i][1]), int(det[i][3])),
284                     },
285                     "name": self.names[int(class_id)],
286                     "class_id": int(class_id),
287                     "conf": float(det[i][4]),
288                     "color": self.colors[int(det[i][5])],
289                 }
290                 min_max_list.append(obj)
291
292         return min_max_list
293
294     return None

```



```

290     @staticmethod
291     def max_objects_filter(
292         min_max_list: List[Any], max_dict: Dict[Any, Any],
293         name_key: str = "name"
294     ) -> List[Any]:
295         """Filter a list of bounding boxes based on the maximum
296         number of objects.
297
298         Args:
299             min_max_list: The list of bounding boxes.
300             max_dict: The maximum number of objects per class.
301             name_key: The name key for class names. Defaults to "
302             name".
303
304         Returns:
305             The filtered list of bounding boxes.
306         """
307         filtered_list = []
308         max_dict_copy = copy.deepcopy(max_dict)
309         for obj in min_max_list:
310             if max_dict_copy[obj[name_key]] > 0:
311                 max_dict_copy[obj[name_key]] -= 1
312                 filtered_list.append(obj)
313             else:
314                 pass
315                 # print(f"rejected {obj[by]} conf {obj['conf']}")
316
317         return filtered_list

```

A - GITHUB REPOSITORY

All code and latex-files used in this document are included in the Github repository linked below. Further explanations are given in the readme-file.

Github repository link

- <https://github.com/beuss-git/bachelor-oppgave-nina>



 **NTNU**

Norwegian University of
Science and Technology