Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Adriana Bostandzhieva

Design and Implementation of System for Managing Training Data for Artificial Intelligence Algorithms

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Adriana Bostandzhieva

Design and Implementation of System for Managing Training Data for Artificial Intelligence Algorithms

Bachelorthesisbased on the study regulations
for the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the Hamburg University of Aplied Sciences

Supervising examiner : Prof. Dr. -Ing. Lutz Leutelt
Second Examiner : Prof. Dr. Klaus Jünemann

Day of delivery 3. Juli 2019

Adriana Bostandzhieva

Title of the Bachelorthesis

Design and Implementation of System for Managing Training Data for Artificial Intelligence Algorithms

Keywords

AI, training data, database, labels, video

Abstract

This paper is part of a pilot project of the Hamburg University of Applied Sciences. The project aims to utilise object detection algorithms and visual data to analyse complex road scenes. The aim of this thesis is to determine the best tool to use to label data for training artificial intelligence algorithms, to specify what data should be saved and to determine what database is to be used to save the data. The validity of the findings is proved by building a small prototype to showcase integration between the labelling tool and the database.

Adriana Bostandzhieva

Titel der Arbeit

Entwicklung und Aufbau eines System zur Verwaltung von Trainingsdaten für Algorithmen der künstlichen Intelligenz

Stichworte

Trainingsdaten, Datenbanke, Video, KI

Kurzzusammenfassung

Diese Arbeit ist Teil eines Pilotprojekts der Hochschule für Angewandte Wissenschaften Hamburg. Das Projekt zielt darauf ab, Objekterkennungsalgorithmen und visuelle Daten zur Analyse komplexer Straßenszenen zu verwenden. Ziel dieser Dissertation ist es, das beste Tool für die Kennzeichnung von Daten für das Training von Algorithmen für künstliche Intelligenz zu ermitteln, anzugeben, welche Daten gespeichert werden sollen und welche Datenbank zum Speichern der Daten verwendet werden soll. Die Gültigkeit der Ergebnisse wird durch den Bau eines kleinen Prototyps bewiesen, der die Integration zwischen dem Kennzeichnungsystem und der Datenbank demonstriert.

Acknowledgement

I would like to thank my supervisor and mentor Prof. Dr.-Ing. Lutz Leutel from the Faculty of Engineering and Computer Science, Hamburg University of Applied Sciences. His expertise, guidance and encouragement played a vital role in making our thesis a success.

I would also like to thank my family and friends for the support and understanding they showed during my studies and during the writing of this paper.

Contents

List of Figures

Listings

1. Introduction

The aim of this section is to present the motivation behind this bachelor thesis as a part of a pilot project in the Hochschule für Angewandte Wissenschaften Hamburg.

The City of Hamburg is focused on making the road conditions more bicycle friendly [22]. The implementation of the policy targets an increase of the bicycle traffic from 12 percent to 25 percent of the total road users by 2020. Despite the vast expansion of the infrastructure in the recent years [23], more than 50 percent of the users still cite infrastructure deficit as one of the main obstacles for bike riding within the city [24].

The project, of which this paper is a part, aims to identify whether the increased number of cyclist is potentially dangerous because of congestions on some bicycle lanes. This requires the development of a system able to detect bicyclists using the bike lanes and determine their number over a given time period.

There are several ways to detect bicyclists. This can already be done through commercially sold equipment. However, this equipment often requires significant upfront investment and/or additional infrastructure such as inductive loops. As a result, a simple system for counting the bicyclist through video data is to be developed. Such a system would require small upfront investment for the equipment and flexibility as it can be positioned in different areas of the in the city.

The implementation of this type of system requires the use of video or image data and an object detection program in order to be able to count the cyclists. In general, there are two main types of object detection algorithms namely supervised and unsupervised. The former requires a set of already classified data in order to learn the common features of the object to be detected. The latter does not require such data but uses the given inputs to identify commonalities in the data. The project for now focuses on algorithms requiring supervised training. This raises the necessity of already classified data which can be used to train the algorithms.

This paper aims to summarise the requirements for a tool that can create classified data for training object detection algorithms, to provide a comprehensive comparison between the tools available on the market and to make a proof of concept tool for managing such data.

The most used algorithms and tools that can be used in order to create training data for this project are outlined in chapter 2; chapter 3 is dedicated on defining the requirements for the training data to be saved and the storage that would be used to save it; chapter 4 and chapter 5 explain the design decisions and provide implementation details. The results are summarised in chapter 6 and chapter 7 provides the conclusion suggestions for future work.

2. State of the Art

2.1. Object detectors

This section gives a short overview of the state of the art object detectors. There are currently two main types of object detecting algorithms namely those depending on feature extraction and those utilising Convolutional Neural Networks (CNN).

2.1.1. Feature based object detectors

The feature extraction algorithms gained popularity after Dalal and Triggs[5] published their paper in 2005 and later going on to win the PASCAL-VOC challenge for pedestrian recognition [7] in the same year. This original paper used a histogram of oriented gradients (HOG) to extract the features followed by a linear support vector machine (SVM).

During the decade after this publication, there were numerous object detection algorithms using HOG in configuration with different filters such as the deformable parts model (DPM)[8], another algorithm trains the system discriminatevely with one positive and a multitude of negative examples [15],

An interesting approach that particularly focuses on bicycle detection is presented by Tian and Lauer [21]. They use the HOG features calculated through DPM to train a cascade consisting of decision forests (DF) and ending with a liner SVM.

2.1.2. Convolutional neural networks

The current state of the art object detectors utilising Convolutional neural networks(CNNs) can be be divided in two groups namely multishot detectors and single shot detectors. The former have to resample the pixels or the feature maps in order to make bounding box hypothesises whilst the latter does not. An example of a multishot detecting system is R-CNN [11] and its derivatives [10], [19] while Single Shot MultiBox Detector (SSD)[14] and You only look once (YOLO)[18] algorithms are single shot and mostly focused on real time performance.

R-CNN

In 2014 Girshick et al. [11] proposed using CNNs to localise objects and a linear SVM. The method consists of region proposals (around 2000 per image) that are fed into a CNN whose job is to extract a fixed length feature vector. The feature vectors are classified through the linear SVM. The R-CNN (regions with CNN) method is improved in [10] and [19] where instead of region proposals the network is given the image as an input.

SSD

The Single Shot MultiBox Detector (SSD) [14] is based on a feed-forward forward convolutional network for the generation of bounding boxes with a fixed score related to the possible occurrence of an object class in these boxes, followed by a non-maximum suppression stage for the final decision. This algorithm is faster than R-CNN because it does not resample pixels or or features for bounding box hypothesises.

YOLO

You only look once (YOLO) [18] is different from the other object detection algorithms. Instead of applying classifiers or localisers multiple times on multiple scales as do other algorithms, YOLO applies a single neural network to the image. The network then divides the image into regions to predict the bounding boxes with the probability of object occurrence. This method is currently the fastest but it is not very good at predicting small objects.

2.1.3. Summary

The single stage CNN detectors seem to outperform the feature based detectors in the recent years. Thus, the following sections and chapters would consider the requirements for this types of algorithms.

However, all these algorithms have something in common. They all need initial data to be able to learn the features or the develop the networks to detect a particular object. This data is called training data and is crucial in developing every object recognition algorithm.

The creation of the training data is mainly a manual process of looking at pictures and marking the position of the object of interest and labelling it. The labelled images are then fed into to the training process. There are already labelled sets of data such as KITTI, PASCAL VOC and COCO. But they do not have "bicyclist", "cyclist" or any other category to describe

"person riding a bike". Thus there is the need for customised data set for the training of the algorithms.

Therefore, in order to train the algorithms used in this project a new set of training data has to be created. There are tools on the market (both open and closed source) which can be used to annotate such images. Some of the most popular are described in next section.

2.2. Labelling tools

There are many open source and proprietary tools available for image and video labelling. This section shortly outlines the most prominent of them.

Matlab[16] is proprietary software that has an integrated application for ground truth labelling. It is also runnable on all three main operating systems.

Microsoft VOTT [17] is an open source labelling tool designed to label images and video sequences. It is written in Electron (based on Node.js) and is runnable on Linux, Windows and macOS machines. The tool is capable of labelling both images and video sequences;it can create custom tags and export labels in several formats.

Labelbox [12] is a proprietary software for creating labelled data. It can provide extended functionality. It is a full commercial solution with data management and application programming interface (API). This solution targets fast development environments.

Labelme [25] is an open source labelling tool written in Python and can be run on all major operating systems. It has extensive functionalities such as semantic and instance segmentation, image annotations for polygon, rectangle, circle, line and point. It can export the labels in JSON format.

2.3. Databases

Another object of this paper is to choose the appropriate storage for the data created by a labelling tool. This paper does not look at a simple data storage such as file system or a spreadsheet because there are more efficient technologies for storing and retrieving of structured and semi structured data namely databases. The two main types of databases are described in the following paragraphs.

2.3.1. Relational database

The idea of a relational database was first proposed by E. Codd in 1970 [3]. Since then relational databases are the dominant database type [20]. They scale vertically by increasing the resources (the CPU, RAM, storage, etc.) available to the server. Relational databases can be open-source such as MySQL, PostgresSQL, SQLite, or closed-sourced such as Oracle RDBMS, IBM DB2 and Microsoft SQL Server.

Relational databases use the Structured Query Language (SQL). In a relational database the data is represented in a form of one or more tables with rows and columns and it has to adhere to previously defined schema. One of the main properties of the SQL databases is that they emphasise the ACID transactions (Atomicity, Consistency, Isolation and Durability) where the consistency of the data is of more importance than its availability.

The connections between the tables are called relations and are represented through foreign keys i.e. the primary key of a record in one table is referenced by an attribute (column) in another table.

2.3.2. Non relational databases

The non relational databases are distributed databases and are commonly known as NoSQL databases. This paper looks at the main properties of the document type of non-relational databases as opposed to graph, column and XML databases. Some of the open source NoSQL databases include MongoDB, Cassandra, CouchDB and more.

The non relational databases rely on key-value pairs to store the information and consequently they are more suited for hierarchical data. They do not rely on ACID transactions but instead are based on the Brewer's theorem (CAP conjecture) [2] where the availability is preferred over the consistency of the data.

In this type of databases, the data from the same type are stored in collections as opposed to tables. The data modelling is flexible and emphasises the logical entities thus allowing in the possibility of some data duplication.

2.4. Problem statement

The literature research implies that there are not many tools for creating and managing visual data for training artificial intelligence algorithms. The Purpose of this paper is to determine which labelling tool is suitable for use for the project outlined in chapter 1, which database is

appropriate for storing the label data for video and image material. To achieve this objective a comprehensive comparison of the available tools is to be performed and a basic prototype is to be developed and evaluated for future extension and use. The requirements for the video and image data, the label data and data storage are outlined in chapter 3

3. Requirements analysis

This chapter is divided into several separate sections in order to outline the requirements sets for the video and image data; the labelling tool and the data that needs to be saved for each image/video, as well as the requirements for the storage solution for this data so that a prototype application can be built according to these requirements.

3.1. Video data

This section outlines the requirements for the video material such as the position of the camera with respect to the objects of interest. There is already some video data produced as explained in section 5.4.

The requirements for the geometry of the video material are shown in Figure 3.1 and are as follows:

- The camera should be mounted at 3m high;
- The the field of view should start at 2,5m from the base of the camera mounting point;
- The field of view should end at 14m from the mounting point;
- The narrower field of view width should be 5m;
- The widest field of view width should be 16,2m;

Moreover, the video data for should be easily interpretable and manipulated to provide support for different frameworks and possible future usages. The requirements for the video material are as follows:

- The video should be saved in mp4 format;
- The frame rate is not fixed but it should be made available in the metadata of the video;
- The frame rate should be high enough to avoid missing relevant objects and to be able to determine their direction of travel;

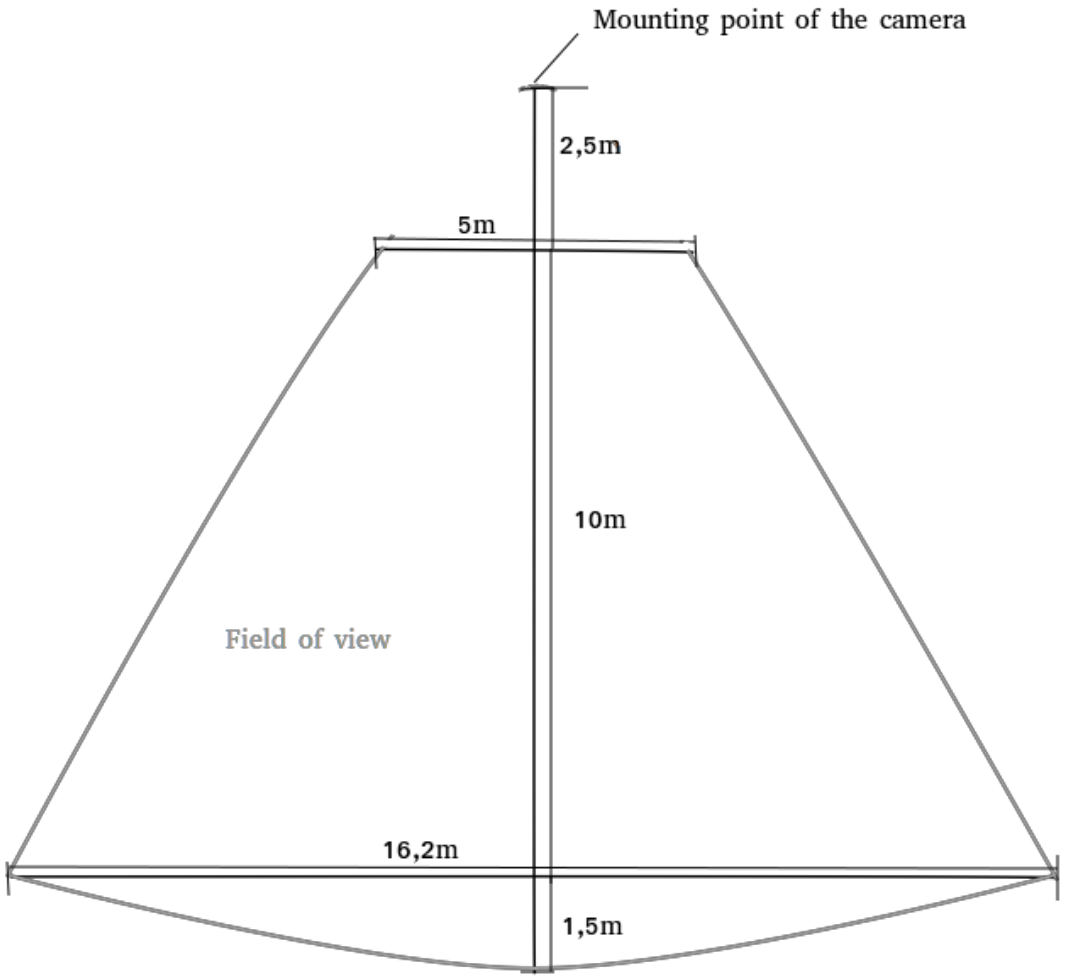


Figure 3.1.: Position and field of view of the camera.

- The video should contain relevant traffic scene i.e. different road users such as cyclists, pedestrians, scooter users etc.

3.2. Data needed for training the object detection algorithm

In general, object detection algorithms need training data to be able to "learn" to detect a particular object class. The training data consists of images that have labels corresponding

on the object classes that are found in the image. The label data also include the location of the particular object in the image. There are two types of data related to the training data namely the metadata and the label data.

3.2.1. Metadata

The label data for each image/video is to be saved in a database. However, there are additional attributes that can define an image or a video such as time of recording and dimension, etc. This requires additional data to be saved along with the label specific data. This type of data is referred to as metadata.

The metadata should include:

- The image / video name;
- The time and date of recording;
- The frame rate if applicable;
- The duration / total number of frames if applicable;
- The location where the image / video was taken;
- The object classes appearing in the image / video.

Some additional information that can be stored for the meta data are defined below:

- The camera type;
- The settings of the camera;
- The light conditions during;
- Position and/or angle of the camera if different from the ones defined in Figure 3.1.

The metadata would allow for easier manipulation of the video data in particular and would provide additional information for further refinement of which training data is best suited for which purpose.

3.2.2. Label data

In general, the data needed to train an object detection algorithm consists of labelled images. The majority of the state of the art algorithms require a bounding box around the object of interest. There are, however, algorithms that use mask instead of a bounding box and these algorithms require annotations.

A bounding box can be represented either as x,y coordinate pairs of the upper left and lower right edges of the box or as the starting x,y coordinate of the upper left corner of the box together with the box width and length.

Annotations represent a polygon outline around the object of interest and thus consist of multiple point coordinates.

The requirements for the label data are as follows:

- The object class of the label;
- The direction of movement (left, right) or a vector representing the movement of the object of interest;
- The bounding box coordinates if applicable;
- The annotations coordinates if applicable;

The labelling data is required purely for the training of the algorithms and its requirements are limited to that use.

3.3. Labelling tool requirements

3.3.1. Data related requirements

The labelling tool should provide all the data defined in 3.2. It has to be noted that some of the definitions in 3.2 are not strict and the integrity of the data is more important than the exact data structure e.g. a bounding box can be defined in different ways as long as the conversion from one format to another is possible. This is to say that all tools will be taken into account as long as such conversions are trivial.

3.3.2. Functional requirements

In order to achieve faster labelling process the labelling tool should preferably be semi-automatic or automatic. A semi-automatic tool would have object tracking between the images / video frames. An automatic tool would be able to detect objects, track them and provide an option to label them.

The labelling tool should have the following functionalities:

- Ability to label images and video sequences manually;
- Ability to export the labels in easy to interpret format plain text format such as JSON, XML or CSV;
- Preferably the ability to track objects between frames.
- Ideally a non restricted number of labels per image;
- Ideally the ability to create both bounding boxes and annotations (polygons);

3.3.3. Non-functional requirements

In the case of creating a custom solution for the labelling tool, the time limit for a Bachelor thesis should be taken into account.

3.3.4. Usability requirements

The role of the labelling tool is to facilitate the people tasked with labelling the training data and to make the process as fast as and as accurate as possible.

This leads to the following usability requirements:

- Should be able to run on Ubuntu 18.04;
- Should have a graphical user interface (GUI);
- Keyboard shortcuts for the most used actions such as changing the image / frame, choosing a label from a list, etc.;
- Ability to manually (using the mouse) adjust the bounding box position and size between frames in case of tracked objects in videos;
- Ability to stop / pause the labelling process and continue in another point of time;
- Ability to perform CRUD (create, read, update, delete) operations on the labels.

3.4. Database requirements

The created labels should be easily stored and accessed. This requires the use of a database. The database should be have the following characteristics:

- Free to use;
- Easily accessible;
- Fast CRUD operations;
- Fast searching;
- Available on Ubuntu 18.04;
- Scalable;

The search operations should include the label and meta data and should result in matching matching the search criteria.

4. Concept

This chapter describes in detail each labelling tool from section 2.2 and discusses their advantages and disadvantages. It also looks at the attributes of the databases and explains the main implementation decisions for the prototype.

4.1. Labelling tools

The labelling solutions outlined in section 2.2 all have their positives and negatives. They all fulfil the requirements set in subsection 3.2.2 apart from the requirement for the direction of movement of the object of interest. This requirement, however, can be easily satisfied by including the direction of movement in the label name. Nevertheless, in order to be able to make an informed decision, it should be mentioned that the development of an own solution is also a valid option. The main advantages and disadvantages of each labelling tool are presented below.

4.1.1. Labelling tools candidates

Microsoft VoTT

This tool was tested in the process of writing this paper. The installation of the program was not difficult and a sample video from section 5.4 was used for evaluation.

Microsoft VoTT [17] satisfies many of the requirements. It can label both image and video sequences; it can create custom tags and export label data in several different formats i.e. CNTK[1], TensorFlow (Pascal VOC) and YOLO. It also allows the use an already trained CNTK model to automatically create new labels. It does not require that the video sequence is broken into frames in order to label it.

On the other hand, it has some major drawbacks. The tools does not support the creation of polygons and it does not automatically detect the frame rate of the video but relies on user

[1] CNTK is an open source Cognitive Tool developed by Microsoft

input for it. Using the correct frame rate is important because mismatch can result in skipped frames. Moreover, despite the fact that it has a tracking algorithm for the movement of the object, this algorithm seems to be slower than the time needed to draw a bounding box[2] around the object.

In general, Microsoft VoTT is a useful tool but it cannot create polygon annotations and the tracking algorithm is not efficient.

Matlab

Matlab [16] has a specialised app in the Computer Vision Toolbox for labelling data. The app can be used to create custom bounding boxes or semantic segmentations. It works with both video data and images.

On the other hand, it can only export labels in matrix form which is not so easy to export to other software. As Matlab is a proprietary project, its usage is tied to specific licence which may not be available in the future or the current university licence may not cover some use cases.

Labelbox

Labelbox [12] is an all round solution for creating and managing label data. As it is a commercial solution it comes with several licences. There is an educational licence [3] which is free but is limited to maximum 10 projects and maximum 30 users. The tool does not provide a desktop applications but a web application.

It can create both semantic segmentations and bounding box labels and it supports various options for exporting the data. Nevertheless, the software claims to be able to label video sequences but in fact it only accepts images (in PNG, JPEG, APNG, SVG, BMP) and labels the said images.

The possible use of this software is also tied to a licence as in the Matlab case and would create an external dependency for the project.

[2] The computer used for testing had 16GB ram and was running Windows Vista x64

[3] The limitation of the Educational licence are not readily available on the Labelbox site. After registering and requesting an educational licence the terms were obtained.

Labelme

Labelme[25] is an open source labelling tool written in Python 3.6 and using Qt as GUI framework which allows it to be run on all major operating systems. Its functionalities include semantic and instance segmentation, image annotations for polygon, rectangle, circle, line and point. It can export data in JSON format.

However, it works only with image data and in order to label videos the individual frames have to be passed to the tool in .jpg format.

Developing own solution

On one hand this would provide full customisation and scalability of the solution. On the other hand, the time and effort required for the development would be excessive for the time dedicated to this paper. It also has to be noted that the project to which this paper belongs, is rather focused on the performance of the object detection algorithms than on the data needed for training them thus a entirely own solution would not be time effective at the current situation.

4.1.2. Discussion

As far as usability is concerned, the most of the applications above are build in such a way that they are easy to use and provide GUI with extended functionalities such as shortcuts for example.

The critical distinguishing points between the tools are their availability, their price and their convenience to extend.

Labelbox's restrictions for number of users and projects make it unsuitable for an educational institution where diverse teams of students and academics are likely to use the tool.

Matlab Ground Truth Labeller is tightly coupled with the other Matlab products. This would make the integration of the Ground Truth Labeller with other machine learning frameworks such as TensorFlow difficult. Moreover, the project would be dependent on a proprietary software that requires a subscription to function.

The other two labelling tools left, Microsoft VoTT and Labelme are both open source and currently well maintained.

Microsoft VoTT seems to be advantageous because of its ability to track objects between frames and to work directly on video sequences without the need of breaking up the video

into separate picture representing a frame. Nevertheless, the above mentioned concerns regarding the speed of the object tracking between the frames are significant as this would have been one of the main advantages of this tool.

On the other hand, the Labelme tool does not directly work on video sequences and some pre-processing of the video to transform it to images would be needed. Nonetheless, Labelme is written in Python which also make it easier to pre-process as Python has good image and video processing support. Python together with C++ are standard languages for machine learning frameworks (e.g. TensorFlow, OpenCV, PyTorch etc.) whilst Microsoft VoTT is written in Node.js. Having this in mind, the use of Labelme would make it easier to integrate with a future framework if the need be and it would reduce the stack of the project. As it is an open source project Labelme can be forked and modified.

To conclude, the use of Labelme as a base and its future customisation would save time and costs, and would not contribute significantly to the complexity of the project.

4.2. Database

One of the main tasks of this paper is to take a decision on the used storage for the future label data. A database approach is chosen as specified in section 3.4 to full-fill of the need for efficient searching. The paper looks in two different types of databases namely the relational database and the document type of database.

A summary of the main attributes of SQL and NoSQL is provided in section 2.3, this section would only compare them.

4.2.1. Discussion

Considering that one of the requirements for the database is to be free, the paper takes into account MySQL and PostgresSQL for the relational databases, and MongoDB for the document databases. These three databases are among the most popular on the market in 2019 [20].

All three databases are open source and free, and are supported on Linux. They also have extended support for popular programming languages such as JavaScript and Python.

They all support fast CRUD operations, although, a study in 2018 [6] found that MongoDB performs slightly better than MySQL. Another report [9] found that MongoDB performed better in complex query situations.

With regards to the requirements in section 3.4 both relational and document based databases satisfy them. The main difference between the SQL databases and NoSQL is the rigidity of the schema. The SQL databases require fixed schema and the data cannot deviate from it. The NoSQL databases, on the other hand, have flexible schemas and are more robust with regards to changing data. This flexibility would be beneficial for the project as future algorithms might require more detailed data and in such case a NoSQL database would be easier to extend.

Another point to be made is related to the organisation of the data within the database.The use of a relational database would produce many more tables if it is fully normalised. An attempt to design a SQL database without normalisation resulted in three tables. An attempt to reduce redundancy on these tables resulted in six tables. If such a database was to be extended at some point in the future, the effort related to changing the its structure would not be trivial.

On the other hand, a NoSQL database can put all the information into one or two documents depending on the expected size of those records. Extending these records would not be a problem due to the non-rigidity of the database model.

To conclude, the NoSQL provides flexibility, fast CRUD and search operations, and scalability. These satisfy the requirements for a database and provide easy extension options for future use cases thus MongoDB would be used as label information storage.

4.3. Combining Labelling tool and Database

So far this chapter discussed the advantages and disadvantages of the main labelling tools and the main databases on the market. However, in order to create a working system the two should be linked together so that the label data for each image/video file could be saved into the database.

None of the labelling tools, except Labelbox, have this saving functionality and as this is not the tool of choice, the link between the two functionalities has to be build. As Labelme is an open source project it can be extended.

The Labelme Graphical User Interface (GUI) is already crammed and additional functionality would make it difficult to work with. For this reason a separate window with the added database related functionality would be a better option. This approach would also allow the future extension of the database related functionality.

4.4. Implementation implications

The above decisions have direct ramifications for the implementation of the prototype application. These are summarised below.

4.4.1. GUI implications

The choice of Labelme implies that the same GUI framework has to be used to extend its functionality and to avoid rewriting of the program. The Labelme GUI is based on Qt and as such using qtpy framework would allow for easier integration of Labelme. This framework allows to switch between the Qt python bindings between PyQt5 [13] and Qt for Python(also known as PySide2) [4]. The most notable difference in the two bindings is the licensing. PyQt5 uses GNU GPL v3 and the Riverbank Commercial License; the Qt for Python uses LGPLv3 and a commercial license. This means that the application should be compatible with these licenses[4].

4.4.2. Database model implications

MongoDB is, as stated above, a document based database that saves records in BSON format which is a binary serialisation of a JSON file. It supports embedded documents and arrays within other documents. This provide the opportunity to put all related data in the same place.

Theoretically, all the data related to a video file can be put into the same document. However, most videos can be hours long and with at least one label in each frame, the record for the video would grow. Although, the database is capable of handling big documents there is a 16MB limit to each document as well. This implies that the data has to be separated and this can be easily done as there are two logical types of records: metadata and label data.

The metadata would represent the generalised data related to the video or the image as outlined in subsection 3.2.1. The main information saved in the metadata document is shown in Listing 4.1.

```
1 {
2     name : "Video 5_short.mp4",
3     media_type : "video",
4     location : "Hamburg",
5     recorded_at : ISODate("2019-05-04T15:15:00.000Z"),
```

[4]More about these two licences can be found on https://www.gnu.org/licenses/licenses.html

```
    label_types : ['rectangle'],
    labels: ['bicyclist']
    url : "/home/adi/thesis/videos/Video 5_short.mp4",
    video_data : {
        frame_rate : 29.807,
        frames : 4,
        codec : "mp4v"
    },
    dimensions : {
        height : 1080,
        width : 1920
    }
}
```

Listing 4.1: Meta data

The label data is related to each image or frame. It is an array of objects with the labels represented in the image as shown in Listing 4.2.

```
{
    labels : [
        {
            object_class : "bicyclist",
            label_type : "rectangle",
            points : [
                [
                    1260,
                    96
                ],
                [
                    1602,
                    561
                ]
            ]
        }
    ]
}
```

Listing 4.2: Label data

5. Implementation

This chapter describes the concrete implementation details for the decisions made in chapter 4.

5.1. System requirements

As per the requirements in chapter 3 the program should be able to run on Ubuntu 18.04. The program is written in Python 3 and since Labelme supports Python 3.6 this is the minimum required version to run the application. In order to run the application it is advisable to create a separate virtual environment with either an environment manager such as Anaconda or Miniconda [1]. This allows the separation of the dependencies from the system dependencies.

5.2. Installation

This section outlines the installation procedure for the application and assumes that there is a virtual environment for python 3+ already set up and activated. This application can be installed with the python package manager *pip*.

The project has a *setup.py* and a wheel file (with extension *.whl*) which can be used to install it. Since the source code is provided with this paper, it is assumed that these files are accessible to the reader.

If only the source code is available, the .whl file can be created from the top most directory of the project by running the command:

```
1   $ python setup.py bdist_wheel
```

After the .whl file is available, the following commands would install the project and start the application.

```
1  $ python -m pip install dist/hawtool-0.0.0-py3-none-any.whl
2  $ cd ../ && pip install -e hawtool
3  $ hawtool
```

The *-e* option on line 2 allows for interactive development as if the source code is changed, there is no need for reinstalling the project in order for the changes to be visible. This option can be omitted if the user does not intend to change the source code.

5.3. Workflows

There are several different workflows depending on the existence of database connectivity and the availability of the files.

Separately from the labelling use cases, the application can generate the video frames from the video file. This is needed when only the video file is available or at the first labelling the data. The generated frames are saved in a sub-folder in the same directory as the video; the name of the folder is in the format <video file name>_frames.

Additionally, there are two modes of generating the files. In the first, all f the frames are generated regardless of the content. In the second a simple average algorithm is employed a with threshold that skips the frames that are not much different from the previous frames. This functionality keeps the original frame number so that the scenes can be properly referenced afterwards. The idea is that the person labelling the images would not have to go through all the frames (eg. the longer video from section 5.4 contains approximately 19,000 frames). This approach is not ideal but motion detection is a big topic on its own and it would not be discussed in this paper.

Use case 1

The first use case is when the user labels in "offline" mode meaning that there is no database connectivity. In this mode, the frames can be generated as described above and consequently labelled. The labelling information is stored in a .json file.

Use case 2

It is also possible to import the media information in the database without labelling. The labelling process in this case can be resumed in another point of time. This is useful when

the data has already been labelled as per use case 1 and it needs to just be imported to the database.

Use case 3

Another use case is to import the media information into the database and consequently label the generated frames. This process extracts the metadata information from the video header. The information of interest is the recording date, the codec, the frame rate, the number of frames, and the dimensions of the video. Since the recording date is not always available in the file header, an error message is displayed to let the user know that the file creation date would be used as the recording date. This date can be changed in the next screen where the user has the opportunity to change the recording date, the location and the url of the file. This information is afterwards saved in the database.

In this case the frames are always generated in a subfolder as the presumption is that this is the first time they have to be generated. The frame generation process overwrites the image files if the subfolder already exists. In case there are already label files in json format in the subfolder they are not overwritten and the information is preserved.

Use case 4

The final use case is when the user wants to resume the labelling process. In this case, the user has to search for the correct media in the database. This is done through the section of the GUI. The process syncs the label information that is already in the database with the one that is in the label files (if any). In this case there is no loss of information if the user was labelling as per the first use case described above.

The saving process for all three use cases is automatic and it is triggered once the user is back from the Labelme Interface. The saving of the labelling files in json format is handled by the Labelme part of the program and the files are saved on when the labels are edited in the Labelme interface. On the other hand, the label information is saved to the database after the labelling process finishes and the user is back to the main application.

The following section explains the video material handling.

5.4. Video material

5.4.1. Video creation

In the course of writing this paper, two video materials were produced. The first is a testing video to refine the position of the camera outlined in section 3.1 and shown in Figure 3.1. This video is 1:29 minutes long. The second video was shot to test the position of the camera in real life conditions and is 21:31 minutes long. Both videos were shot on 13.09.2018 in the vicinity of Hochschule für Angewandte Wissenschaften Hamburg. The equipment used consisted of Logitech C920 HD Pro web camera and the accompanying it software on a Windows Vista machine. It has to be noted that using this software is not optimal as the video metadata lacks the recording time in both instances.

5.4.2. Video material handling

As specified above the application takes a video file, reads the data in the file header and generates images representing the individual frames of this file. There are two external dependencies involved in this process.

Ffmpeg

The file metadata (from the header) is read by the *ffmpeg-python* library. This library extracts the recording. In case this date is not available (if the date is 01.01.1970 is also considered missing because in unix format 0 is this date) the program uses the file creation date.

OpenCv

The *opencv-python (cv2)* library was chosen for more complex handling of the video material because it provides methods for reading the video file data and also has modules that can be used for calculating the average algorithm when generating selected frames (see section 5.3).

The library is essential to the video material handling. It is used for reading the frame rate, the frame number, the codec information and the dimensions of the video. This library is also used to generate the individual frames from the video file, which are saved in *.jpg* format.

5.5. Graphical User Interface

The Graphical User Interface (GUI) has two windows. On the start of the program the user sees the main application window. This is changed for the Labelme interface when the users labels the images. The user can switch back to the main application window by using the Ctrl+Q shortcut or choosing Back from the file menu in the Labelme window (more shortcut options can be found in Appendix A).

The main application window allows the user to connect to a database, start the labelling process according to one of the user cases in section 5.3 and search for video and image materials.

The GUI can import new media and continue the labelling process for already imported media. It is to be noted though that the image files for either videos or images have to be available on the local machine. The application is not capable of downloading files from other machines be it servers or other web locations.

The search widget has several fields, including filtering by the MongoDB identifier, the date of recording, the existing labels, the label shape, the location and the media type. Currently, these are used for easier choice of media to be continued labelling as this would assure the referencing of the exact media record. The filtered media is shown in a tabular form.

5.5.1. Label file creation

Labelme creates a .json file containing image information such as the dimensions and the already labelled objects. These files have the same name (with .json extension) as the image file and are in the same directory.

The main application complies with this approach and creates empty label files in the same format as the ones from Labelme. These are created before the actual labelling process to ensure that all label files have the media reference identifier when available. This is needed when saving the labels in the database and also if the same images are used for labelling different media records. The empty label file structure is shown in 5.1.

```
1 {
2   "flags": {},
3   "shapes": [],
4   "lineColor": null,
5   "fillColor": null,
6   "imagePath": "/home/adi/thesis/videos/Video 5_short/Video 5_short_0.
        jpg",
7   "imageHeight": 1080,
8   "imageWidth": 1920,
```

```
9   "media_id": "5d17ed28fbfd7d037ba4a95f"
10 }
```

Listing 5.1: Empty label file

The .json files with the labelling information are synced with the labels in the database once the labelling processed is resumed as stated above. The synchronisation process is triggered if the existing label file has the same *media_id* as the database entry or it has no *media_id* at all.

The saving of the labels in the database is triggered once the user is back to the main window. The Labelme part sends the changed label file names to the main application which triggers the database saving process. Only files which have been changed are saved aka even if empty .json files are created at the start of the labelling process the database entries would not be created if the image file was not open in Labelme.

It has to be noted that once the current labelling session is finished, the database label entry is overwritten. This can creates a racing condition in the case that two users are labelling the same media on different machines.

5.6. Database

In order to establish connection to the database, the database server has to be running. The correct version of MongoDB has to be installed[1].

It has several input fields for the port, host, the database name, user and password. The port and the host are defaulted to the local instance of the database at *localhost:27017* and the database name defaults to *"hawtool"*. In case the database does not exist in the MongoDB server, any writing operation would result in its creation. The GUI can connect to only one database in a given instance of the application i.e. if the need be to connect to a different database, the application has to be restarted.

The application uses *MongoEngine*[2] which is an open-source Document-Object Wrapper for MongoDB and Python. The framework can be used for establishing the connection to the database. The connection is initialised by the *DbConnectWidget* class and uses a worker to establish the connection without blocking the main tread. This is needed in case the MongoDB server is not running. In this case the error message requires considerable time

[1]The official installation documentation can be found on https://docs.mongodb.com/manual/administration/install-community

[2]More can be found on http://mongoengine.org

thus the application seem non responsive and the operating system might consider it non responsive. If that happens an error message with the error is displayed.

5.6.1. Schemas

The use of *MongoEngine* also allows the definition of schemas and easier introduction of indices. The schema definition shown in Listing 5.2 is the definition for the media data. It can be seen that the fields have pre-defined type such as String, Int, List, etc.. The list fields could contain simple Strings or EmbeddedDocuments.

The data is saved in the database and a unique id is given to each of the records. These, however, cannot guarantee the one record per video file and can lead to multiple insertion of the same data. Thus, an index consisting of the name of the file and the time of recording (or if that is not available in the metadata of the video file, the file creation date) is introduced through the use of *unique_with* option. This means that every insertion of files with the same name and recording date would result in an error.

The fields can be made obligatory through the *required* option or can be limited to particular pre-defined values as it is the case with the *name* field for example.

The framework allows for skipping the non required fields but it does not allow for the insertion of non defined fields. If this is attempted, an error would be triggered and the record would not be created.

Metadata schema

The metadata schema in Listing 5.2 have some additional fields. The *created_at* and *updated_at* fields can be used for filtering the in the future when there would be more data. For example, if the user would be able to see when they last worked on the specific file or when it was imported.

The *tags* list can be used for scene specific information descriptors such as sunny, cloudy, rainy etc.

The custom index for this schema is the combination of *name* and *recorded_at* fields.

```
1 class VideoData(EmbeddedDocument):
2     frame_rate = FloatField()
3     frames = IntField(min=1)
4     codec = StringField()
5 class Dimensions(EmbeddedDocument):
6     height = IntField(required=True)
7     width = IntField(required=True)
```

```
8
9  class MediaData(Document):
10     name = StringField(required=True)
11     created_at = DateTimeField(default=datetime.datetime.utcnow)
12     updated_at = DateTimeField(default=datetime.datetime.utcnow)
13     media_type = StringField(choices=['video', 'image'])
14     location = StringField()
15     recorded_at = DateTimeField(required=True, unique_with='name')
16     label_types= ListField(StringField())
17     labels = ListField(StringField())
18     tags = ListField()
19     url = StringField()
20     video_data = EmbeddedDocumentField(VideoData)
21     dimensions = EmbeddedDocumentField(Dimensions, required=True)
```

Listing 5.2: Meta data schema

Label schema

The Label collection schema is shown on Listing 5.3. As it can be seen the fields *media_id* and *frame* create an index to ensure the uniqueness of the record. Consequently, for every frame of a video there can only be one label data record in the database.

The label record has currently unused field *direction*. The current GUI does not support the input of direction information but this can be interesting information to look for in future use of the database. The use of the *tags* field is the same as in the media schema.

```
1  class LabelData(EmbeddedDocument):
2      label = StringField(required=True)
3      direction = StringField(choices=LABEL_OPTIONS['directions'])
4      shape_type = StringField(required=True)
5      points = ListField()
6
7  class Label(Document):
8      media_id = ObjectIdField(required=True)
9      frame = IntField(min=0, unique_with='media_id')
10     labels = EmbeddedDocumentListField(LabelData)
11     image_height = IntField(min=0)
12     image_width = IntField(min=0)
13     tags = ListField()
```

Listing 5.3: Label data schema

Linking the media data and the label data

The media collection is the one used for the GUI searching since the number of records is going to be always be less than the number of records in the label collection. This, however, requires synchronisation between the two collections. Currently, this involves only the *labels* list in the media data which is updated with the existing labels related to record after saving the new labels in the database. The list is essentially overwritten every time new labels are saved to ensure the removal of deleted labels. The same should be done in the future for the tags list.

6. Results and Evaluation

This chapter looks at the results of the implementation after testing the application with the video from section 5.4.

One of the objectives of this paper is to define a database to be used for storing the training data and to define which data exactly is to be stored.

After considering both relational databases and non-relational databases, a non-relational database was chosen. This type of database would has more flexible data model and in the event that there are unforeseen data to be stores, this can be added without much additional effort.

The data defined to be stored in the database is shown in Listing 4.1 and Listing 4.2.

Another objective of this paper is to choose a labelling tool that is capable of producing the data needed for training the object detection algorithms. The tool of choice is Labelme [25].

This tool satisfies the requirements from subsection 3.2.2, is easy to extend and is open source.

6.1. Prototype

A prototype was build in the course of this paper in order to showcase the integration between the database and the labelling tool. This section looks at it and some of its functionalities. The main application window can be seen on Figure 6.1.

6.1.1. Handling video material

The labelling part of the prototype accepts only image files however video material is easier to produce in high quantities than image material. In order to be able to handle video material, the videos are separated into individual frames.

The simpler mode of creation of the images is to save all frames from a video material. This approach is suitable for videos with constant movement of the objects or for shorter videos.

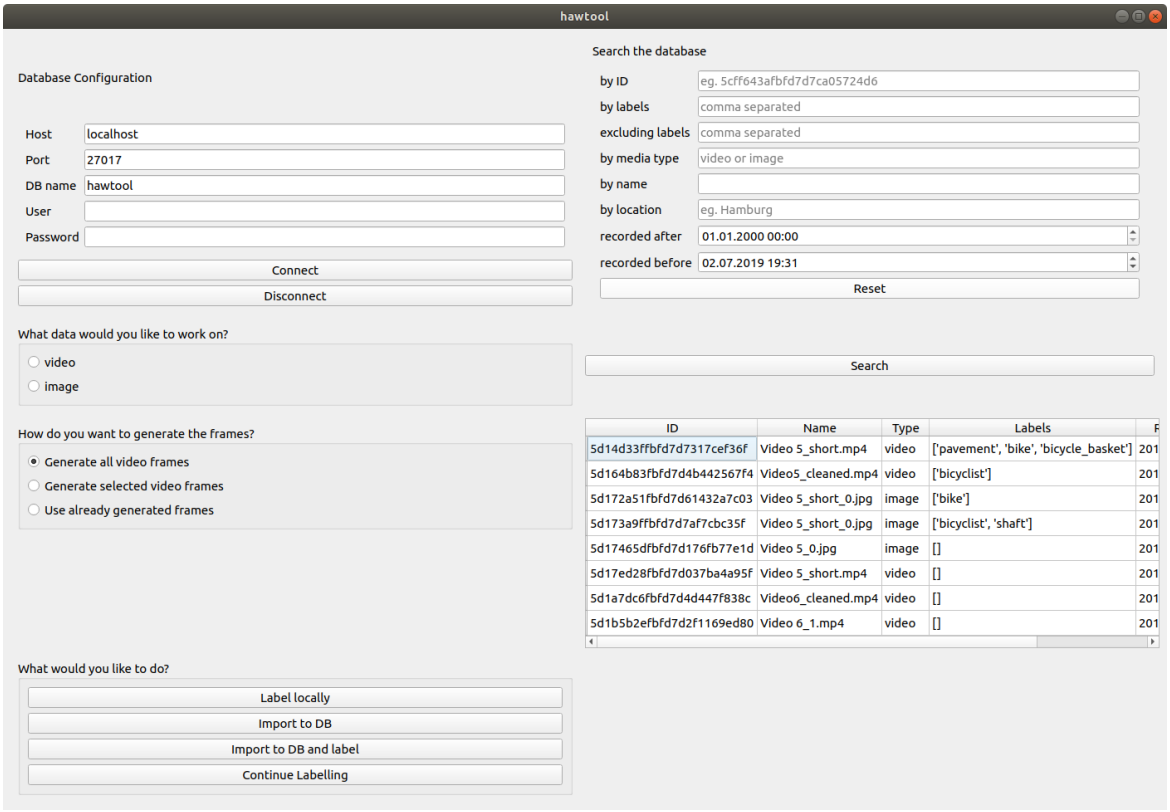


Figure 6.1.: The GUI with interactive elements.

The second option is to skip some of the frames through an average algorithm. This approach is convenient when the video data is repetitive and with long periods of no moving objects. Albeit simple, labelling images from videos created in this mode allows the user to save time. For an example, the longer video produced in section 5.4 has 19,200 video frames, whilst the program creates around 8,000 images for this video.

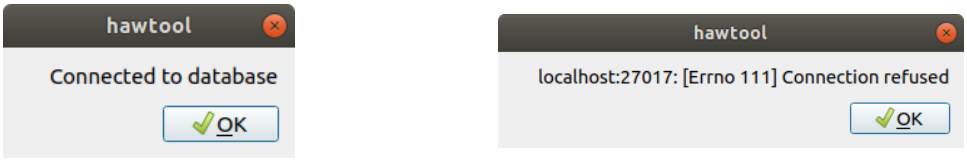
It has to be noted that this process can be time consuming and currently the frame generation is in the main thread. This can cause the GUI to look unresponsive. All message stating that the GUI is unresponsive should be ignored if the user is doing computationally intensive tasks.

6.1.2. Database connectivity

The connectivity functionality was tested while the server was running and while it was not.

As expected the connection to the server was established almost instantly after initialisation

of the process. In the other case when the server is not running, the detection of error takes considerable time and results in an error message as shown in Figure 6.3d



(a) Connected message.

(b) Error message.

Figure 6.2.: Messages on database connection

The application has a disconnect button to disconnect from the database and allow for connection to another MongoDB server.

6.1.3. Labelling

Some small changes were made to the Lableme interface so that the user cannot load new images or change the output directory. Another change that was made is that the labelling interface returns a list with changed files that have to be saved in the database. This allows for the update of the label data in the database.

The default saving action for the labels now is automatic, the original version defaulted to manual saving, but the automatic process makes the labelling faster.

6.1.4. Importing to the database

The functionality of the prototype is saving metadata information and label information to the database.

The importing the media data in the database can be done either separately of the labelling process or the process can trigger both saving the data and the labelling interface. This allows for more flexibility for the user.

Additionally to the two videos produced with this paper, the import functionality was tested with another unrelated mp4 video file [1]. In this case the metadata of the file header was read correctly and the recording date of the video file was used directly i.e. there was no need for substituting the recording date with the file creation date or user input.

[1] A file was downloaded from youtube.com with the aid of an online converter

The labels are imported to the database after the images have been opened by the labelling tool. This means that although empty labelling files have been created the information is not saved in the database until a user proves that there isn't an object of interest on the image. For example, media object with four frames have only 3 in the database, because the other image has not been reviewed by a user and it cannot be discarded as lacking objects of interest.

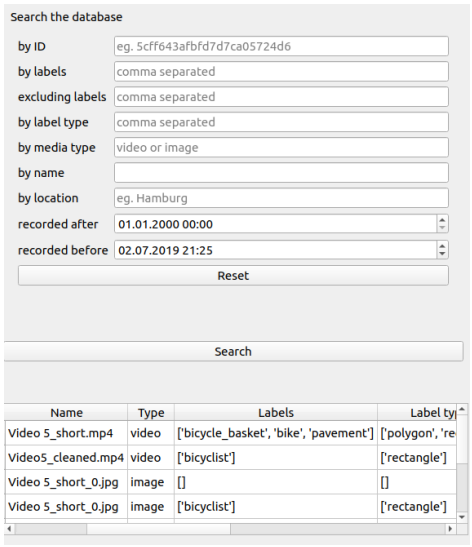
6.1.5. Searching

The prototype allows for searching the database. The user has several options to search the database as outlined below:

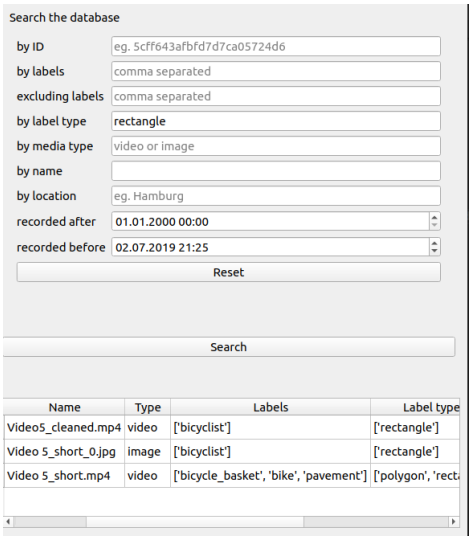
- By media_id - if there is a label file that contains the id, this is the fastest and the most secure option than the images would be saved to the appropriate media reference;
- By label - one label or a comma separated list of labels;
- By excluding labels - one label or a comma separated list of labels;
- By label type- one label type or a comma separated list of label types eg. rectangle;
- By media type - video or image;
- By name;
- By location - at the moment this has to be an exact match;
- By recording date after - this is currently always applied to filter;
- By recording date before - this is applied only if changed.

This search options are designed to support future extraction of data related to particular object types in order to train algorithms. The search by label type would be particularly in this case as some algorithms might require polygons[2].

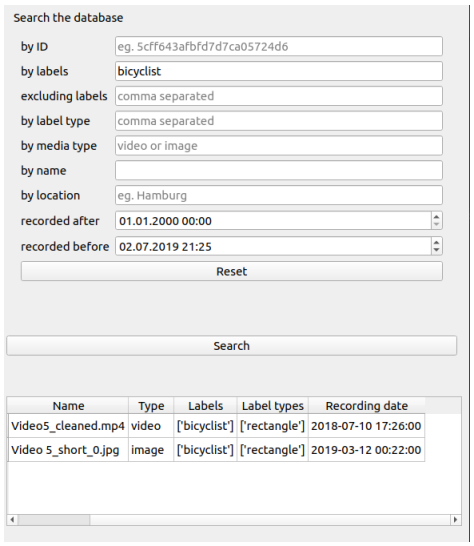
[2]The creation of bounding boxes should not be complicated as the furthest points can be taken as reference to draw a rectangle.



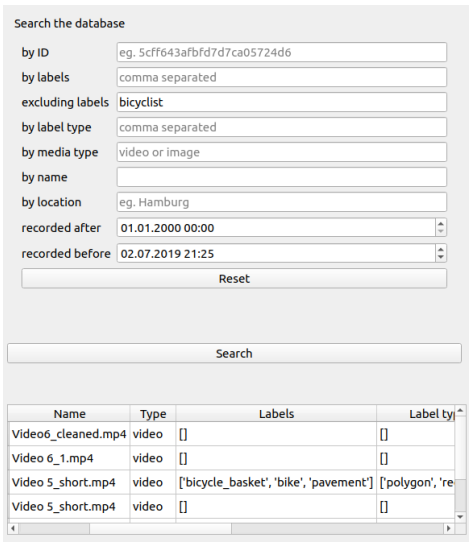
(a) Display all



(b) Display rectangular labels



(c) Display media having bicycles



(d) Display media excluding bicycles

Figure 6.3.: Search results for some filter options.

6.1.6. Missing functionality

Due to time constraints, some functionalities were not implemented. The prototype can be further extended to allow for updating the media record in the database. This is needed for updating the url of the media file. As the current program cannot download the media files, this field is not used.

Another functionality that could not be implemented due to time constraints was the downloading of the files. It is assumed that the files are stored on the local machine. This is not optimal and a better solution would be for them to be stored on a remote location. This approach would provide two advantages namely control on who accessed the video material and space optimisation as there would be only one instance of video.

The last functionality that was not implemented was the creation of a training dataset. This would have taken image data and converted it into a training set for framework of choice such as TensorFlow, Caffee2,etc. Such a functionality would employ the already provided searching part of the prototype which would be used to compile a list of media containing the desired object class. Afterwards, the appropriate images (frame numbers in case of video) would be filtered, the image data downloaded, resized if needed and finally converted to the required format.

7. Conclusion

This paper is part of a project at the Hamburg University of Applied Sciences. The project requires custom data for training artificial intelligence algorithms and this thesis's objective is to find a solution for creating and storing such data.

This paper looked at the state of the art algorithms for object detection and the data required for their implementation; researched the available tools used for creating training data for these algorithms and summarized the available storage solutions for the training data such that would allow convenient extraction of the data. Finally, it delivered a basic prototype displaying the possible used of the combination of the label creation tool and the database.

The current prototype can be further extended to be easier to get data for training the algorithms. The searching system already allows the filtering for certain object types. An additional module for downloading the image and video files from a remote location can be implemented. This would further allow for the creation of training datasets.

Bibliography

[1] Continuum Analytics. Conda - documentation. https://docs.conda.io/projects/conda/en/latest/index.html. Accessed: 2019-06-28.

[2] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[3] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[4] The Qt Company. Qt for python | the official bindings for qt. https://www.qt.io/qt-for-python. Accessed: 2019-06-28.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[6] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili, and B. Raufi. Analysis and comparision of document-based databases with relational databases: Mongodb vs mysql. In *2018 International Conference on Information Technologies (InfoTech)*, pages 1–4, Sep. 2018.

[7] Mark Everingham, Andrew Zisserman, Christopher KI Williams, Luc Van Gool, Moray Allan, Christopher M Bishop, Olivier Chapelle, Navneet Dalal, Thomas Deselaers, Gyuri Dork, et al. The 2005 pascal visual object classes challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 117–176. Springer, 2006.

[8] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[9] Alexander J Fiannaca and Justin Huang. Benchmarking of relational and nosql databases to determine constraints for querying robot execution logs. *Computer Science & Engineering, University of Washington, USA*, pages 1–8, 2015.

[10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[12] Labelbox Inc. https://labelbox.com/.

[13] Riverbank Computing Limited. Riverbank | software | pyqt | what is pyqt? https://riverbankcomputing.com/software/pyqt/intro. Accessed: 2019-06-28.

[14] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[15] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-svms for object detection and beyond. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 89–96. IEEE, 2011.

[16] Matlab ground truth labeler, 2018B. The MathWorks, Natick, MA, USA.

[17] Microsoft. Microsoft visual object tagging tool.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[20] solid IT gmbh. Engines ranking. https://db-engines.com/en/ranking, Jun 2019. Accessed: 2019-06-23.

[21] Wei Tian and Martin Lauer. Fast cyclist detection by cascaded detector and geometric constraint. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 1286–1291. IEEE, 2015.

[22] Freie und Hansestadt Hamburg. Mit dem fahrrad in die zukunft.

[23] Freie und Hansestadt Hamburg. Fortschrittsbericht 2018 radverkehrsstrategie für hamburg, 2018.

[24] Sinus Markt und Sozialforschung GmbH. Radverkehr und lebensqualität in hamburg, October 2018.

[25] Ketaro Wada. labelme: Image Polygonal Annotation with Python. https://github.com/wkentaro/labelme, 2016.

A. Lableme shortcuts

This appendix summarizes some shortcuts in Lableme.

Shortcut	Command
Ctrl+Q	Back to main window
D	Open next image
Ctrl+Shift+D	Copy label and open next image
A	Open previous image
Ctrl+Shift+A	Copy label and open previous image
Ctrl++, Ctrl+=	Zoom in
Ctrl+-	Zoom out
Ctrl+0	Zoom to original
Ctrl+F	Fit to window
Ctrl+Shift+F	Fit to width
Ctrl+Shift+P	Add point
Ctrl+N	Create polygon
Ctrl+R	Create rectangle
Ctrl+J	Edit polygon
Delete	Delete polygon
Ctrl+D	Duplicate polygon
Ctrl+Z	Undo
Ctrl+Z, Backspace	Undo last point
Ctrl+E	Edit Label
Ctrl+L	Edit line color
Ctrl+Shift+L	Edit fill color
Ctrl+P	Toggle copy label from previous image

Declaration

I declare within the meaning of section 25(4) of the Ex-amination and Study Regulations of the International De-gree Course Information Engineering that: this Bachelor report has been completed by myself inde-pendently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, July 3, 2019

City, Date

sign