FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	System for traffic surveys automation
Student:	Bc. Ondrej Pudiš
Supervisor:	Ing. Marek Sušický
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of winter semester 2022/2023

Instructions

The main goal of this thesis is to measure the traffic density from stationary cameras.
The application counts vehicles that pass the selected area and the time they spend there. Measured values are accessible as statistics about the traffic in time.
The student will study the state-of-the-art methods for multiobject-tracking, compare them using real data and design an architecture that fits the application's needs.
The final application will be evaluated against the human-annotated data.

Electronically approved by Ing. Karel Klouda, Ph.D. on 19 August 2021 in Prague.

Master's thesis

SYSTEM FOR TRAFFIC SURVEYS AUTOMATION

Bc. Ondrej Pudiš

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Marek Sušický
May 3, 2022

Contents

List of Figures

List of Tables

List of code listings

I thank my supervisor Ing. Marek Sušický and his colleagues Mgr. Tomáš Karella and Mgr. Adam Szabó for their help, thoughtful notes and suggestions about my work.
I express gratitude to Ing. Bc. Petr Kumpošt, Ph.D. and his team at the Mobile Laboratory of Traffic Surveys for sharing their knowledge, experience and video data to be used in my work.
I acknowledge the AI City Challenge organizers and contributors for creating a video recording dataset that I used for the proposed framework's evaluation in my work.
I acknowledge the support of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" which provided the resources to run all the experiments.
Finally, I thank my family, my colleagues and my friends for their never-ending support, belief and understanding throughout the years of my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 3, 2022 .....

Abstrakt

V posledných rokoch sme svedkami masívneho rozvoja cestnej dopravy, ktorá spôsobuje niekoľko problémov, najmä v mestských oblastiach. Miestne samosprávy a vlády preto investujú nemalé prostriedky do dopravných prieskumov, ktoré umožňujú získať predstavu o úzkych hrdlách súčasnej infraštruktúry. Súbežne dosahujú vedci významný posun v oblasti hlbokého strojového učenia, ktorý prináša automatizované, presné a výkonné riešenia v oblasti strojového videnia. V našej práci sa venujeme vývoju otvoreného softvéru, ktorý umožní automatizáciu dopravných prieskumov. Na začiatku diskutujeme a objavujeme rôzne prístupy k sledovaniu viacerých objektov vo videu súčasne. Následne navrhneme modul, ktorý analyzuje nahrávku zo stacionárnej kamery a spočíta dopravné prostriedky prechádzajúce cez oblasti záujmu definované používateľom. Kvalitu nášho modulu ohodnotíme na niekoľkých videách, popíšeme metriky a využitie hardvéru. Nakoniec zhrnieme presnosť, naše požiadavky na zdrojové videá a navrhneme niekoľko vylepšení. Zistíme, že modul funguje dobre a vrámci prípustnej chyby, pokiaľ vo výhľade neprekáža dopravné značenie alebo iné automobily.

Kľúčové slová rozpoznávanie objektov, sledovanie viacerých objektov, rozpoznávanie a sledovanie vozidiel, počítanie vozidiel, dopravný prieskum

Abstract

In recent years, we have witnessed an expansion of road traffic that causes several problems, especially in urban areas. Local authorities and governments invest a lot into traffic surveys to keep track of infrastructure bottlenecks. At the same time, researchers have made significant progress in deep learning, allowing for automated, accurate and performant solutions for computer vision tasks. In our work, we aim to create an open-source software tool that enables the automation of traffic surveys. We investigate approaches to multi-object tracking and suggest a framework that analyses a stationary camera recording of road traffic and counts the vehicles that move through the user-defined regions of interest. We evaluate the performance of our framework on a couple of datasets, comment on metrics and hardware usage, and report accuracies. We conclude with findings and requirements placed on the source videos; we suggest a few improvements to the framework. We identify that the framework works well and within the boundaries of allowed error if there are no or just a few traffic signs or automobiles blocking the view.

Keywords object detection, multi-object tracking, vehicle detection and tracking, vehicle counting, traffic survey

Glossary

CRUD Create, read, update and delete operations on a set of objects.

CUDA NVIDIA's platform for GPU-accelerated parallel general purpose computing.

Docker A tool that enables shipping software as containers and run them using operating system virtualisation.

GitHub An online code repository with GIT version control support.

OpenAPI A specification, originally called Swagger, of how to describe, produce, consume and visualise RESTful web services.

Acronyms

API application programming interface.

ASGI asynchronous server gateway interface.

BiFPN weighted bi-directional feature pyramid network.

CLI command-line interface.

CNN convolutional neural network.

CTU Czech Technical University in Prague.

DDL data definition language.

Deep-SORT Deep Associations Online and Realtime Tracking.

DL deep learning.

DSN data source name.

FPS frames per second.

FTS CTU Faculty of Transportation Sciences of the Czech Technical University in Prague.

GDPR General Data Protection Regulation.

GPU graphics processing unit.

GUI graphical user interface.

HD high-definition video (720p).

HPC high-performance computing.

HTML hyper-text markup language.

IoU intersection over union.

ML machine learning.

MOT multi-object tracking.

MOTA multi object tracking accuracy.

NMS non-max suppression.

OOM out-of-memory.

RAM random access memory.

ReLU rectified linear unit.

SORT Simple Online and Realtime Tracking.

UUID universal unique identifier.

Chapter 1
Introduction

With millions of new cars arriving on the roads every year and increasing mobility in urban areas, new challenges in traffic management start arising. Everyday traffic jams on arterial routes, serious collisions, and high volumes of greenhouse gasses emitted into the atmosphere are problems that need immediate attention from the responsible authorities.

Having quality statistics about the volume of vehicles that move around roads is essential for finding reasonable and sustainable solutions to various traffic-related problems.

Nowadays, manual and automated surveys are two main approaches to collecting traffic data. While in manual surveys, people are standing next to roads and counting vehicles passing by, either by writing on paper or using dedicated counting devices, automated surveys use machines like radars or event loggers to do the job.

Both private companies and universities help local authorities and governments with traffic research. Even at the Faculty of Transportation Sciences of the Czech Technical University in Prague (FTS CTU), there is *Mobile traffic analysis laboratory* which actively helps cities across the Czech Republic to collect and process traffic data.

Nevertheless, modern and innovative alternatives keep appearing. Advancements in machine learning allow researchers to apply deep learning methods to the problem of traffic surveys as well. The resulting models accept live or recorded video input, apply object recognition, classification, object tracking techniques, and output various statistics about traffic.

Utilising such approaches for traffic surveys can save much time for people who would have to spend it on producing statistics. Moreover, they can result in widely available real-time traffic analysis. Regular people could benefit from such an application because it would enable them to plan their travels and avoid heavily utilised roads.

In this thesis, we aim to discover object detection and tracking models and create a *deep-learning-driven* application that estimates the traffic volume by analysing camera recordings. To be more precise, it outputs counts of vehicles crossing any of the given lines.

We organise this thesis into eight chapters. This chapter introduces the topic of traffic surveys, analyses the evolution of road traffic, and describes a few institutions that provide traffic survey-related products, both physical and software.

We dedicate the second and the third chapter to the theoretical background and study of the related work. We introduce deep learning, object detection, object tracking and the algorithms that we use across our work.

Next, we analyse the requirements of the application based on a couple of interviews [1, 2] that we had at FTS CTU. It is followed by proposing the whole framework in chapter five. We describe the pipeline, how the video data flows in the system, what object detection and object tracking algorithms we use, how the framework stores partial results and creates the final results.

Chapter six presents the implementation: what machine learning packages we use, how the

technological stack of both backend and frontend looks like, how components communicate and how to start and use the software.

Finally, we test the proposed framework on real videos from the Czech Republic and the USA with the counts of vehicles gathered manually, conclude our findings, commenting on the performance of the presented models and suggesting further steps that could be taken to make the framework better in terms of accuracy and usability.

1.1 Goals

The main goals of this thesis are to *discover machine learning approaches in traffic analysis* and *to build a deep-learning-driven application* for performing an analysis of traffic recordings. We focus on building a pipeline from gathering and preprocessing the data through analysing it to presenting the data to users. Moreover, we provide our software as open-source, so even institutions without an extensive budget could perform a baseline traffic survey analysis using our tools.

The theoretical part introduces traffic surveys and commercial products available in this industry. Next, we describe the problem of *object detection*, *multi-object tracking*, and we comment on the existing approaches and methods that researchers propose and use in their work related to multi-object tracking, not necessarily limited to vehicles.

In the practical part, we suggest a framework based on the theoretical part's findings, implement it, and create a functional application that performs a simplified traffic survey. The job includes choosing the right technologies for working with the source video, building the prediction model, storing the results, computing the statistics, and presenting them to users.

1.2 Traffic surveys

In this introductory section, we take a brief look at the history of road traffic, introduce the concept of a traffic survey, explain its benefits, and enumerate a few traditional methods for carrying out such surveys.

1.2.1 Road traffic history

Humankind's need for moving goods or themselves between places has existed since the dawn of civilisation. Initially, people travel on foot, carrying goods on their backs or heads. The first non-trivial means of transport appear in the Upper Paleolithic era when they utilise animals like donkeys or craft the first rafts for moving on rivers.

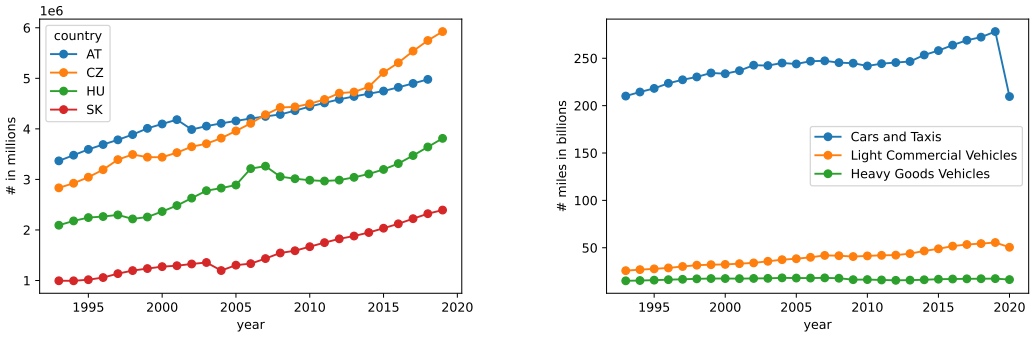
Mesopotamians make a significant leap forward by inventing a wheel 3000 years BCE, using it for pottery, and later building carts and chariots, which simplify the transport of goods.

During the Middle Ages, carriages and coaches with four wheels, drawn by horses, allow people to travel more comfortably and faster, even for long distances. These become popular mainly with wealthy aristocrats and businesspeople.

The golden era for transportation comes in the 19^{th} century. Many new means of transport are invented, notably steam-powered locomotives, steamboats, bicycles, trams, motorbikes and railways. These new machines, especially trains, allow for a safe and fast movement of commodities and contribute to massive public transport usage.

Personal transport develops only after the invention of internal combustion engines. Mass production of affordable and reliable cars allows people to move freely and independently. Travelling requires less time than ever before, and far-away destinations are reachable within hours. [3]

Now, let us fast-forward to the beginning of the 21^{th} century. There are a million new cars registered across the European Union each month, every second citizen owns a passenger



(a) Passenger cars in Central Europe [4]

(b) Driven vehicle miles in the UK [6]

Figure 1.1 European vehicle statistics since 1993

car [4], and transport accounts for almost 30% of CO_2 production [5], we suddenly face challenges that were unimaginable only a 100 years ago. Figure 1.1a shows how the number of registered passenger cars developed in the last twenty-eight years.

Since 1949, the traffic in the United Kingdom, represented in vehicle miles, was increasing fast, mainly in the private transport segment (Figure 1.1b), with almost 2000% increase over the last seventy years. The data from 2020 show that the global pandemic of Covid-19 had a significant impact on both private and public transportation. In fact, people drove 24% fewer miles than in 2019; lorry traffic experienced only a slight decrease of approximately 5%. On the other hand, the number of miles ridden on bikes increased by more than 45%.

We can expect that once governments withdraw restrictions, the traffic volume will return to the pre-pandemic state. It might be even worse because people will be afraid to travel by public transport and prefer to use their cars to feel safer. [6]

Despite the slight drop in traffic during the Covid-19 crisis, the high numbers of vehicles moving across streets, roads and highways cause a number of issues. Let us mention just a few of them:

- air pollution,
- significant emission of greenhouse gasses,
- congestions,
- collisions.

These phenomena harm the environment, economics, and human emotions. [7]

According to an article by Afrin and Yodo [8], congestions usually appear when a traffic flow is interrupted by unexpected incidents or by a high density of vehicles, usually at the morning and afternoon peaks. They are an extensive problem, especially in urban areas. Moreover, they can be the root cause of the other listed issues.

While vehicles stay still, their engines keep running, producing a significant amount of fumes that concentrate in the same place. Furthermore, congestions have a noticeable impact on the economy due to lost productivity, estimated to $87 billion. [8]

Governments, local authorities, researchers, and public and private institutions must continuously monitor and gather data about traffic conditions to avoid socio-economic-environmental issues. This activity allows for better planning of future infrastructure development and optimising the traffic flow. [8]

1.2.2 A traffic survey

McClintock's article [9] describes a traffic survey as an elementary step for achieving control, ensuring safety and convenience in traffic movement. It provides a rational basis for making decisions about traffic control systems, assessing the quality of infrastructure and data-supported prioritisation of new road construction. The author believes that such studies need to be realised regularly, hence keeping local authorities up to date. He also strongly emphasises the importance of the role of pedestrians in traffic surveys.

Others relate a traffic survey to a set of activities designed to gather data about a particular area's real traffic situation. Their primary purpose is to improve the quality of transport. The gathered data can be:

- numbers of vehicles on-road,
- journey information like speed or delay,
- directions of traffic,
- accidents statistics and
- origins and destinations identified by licence plates or by interviewing drives. [9] [10]

A traffic survey data is usually gathered either manually or automatically. Manual counting requires people to stand next to a road and record their observations on paper. Alternatively, they can create a video recording and count later in an office. The results are subjected to an observer's judgement, mainly when the survey includes vehicle classification. In their article, Pengjun and McDonad [11] claim that the total vehicle count error is usually around 1%, while the classification error is remarkably higher – between 4% to 5%. Another disadvantage of manual counting is considerable time consumption.

On the contrary, automatic surveys utilise a radar device placed next to a road so that it does not affect regular traffic flow and does not require much human interaction. Other automatic devices require a pair of cables placed on a road.

Paľo et al. [10] state that such devices record accurate, straightforward and unbiased data. They provide vehicle counts, but they also capture speed, vehicle length, and direction. However, some limitations and prerequisites need to be met to hit a certain level of accuracy and consistency. For example, a good monitoring location, sufficient mounting height, distance between multiple devices, and absence of metal constructions. The authors conclude their study with a 2% - 3% difference between manual and automatic counting. They believe there is a long-term traffic counting use case for automatic counters.

1.3 Commercial products

In this section, let us mention companies that run their businesses in the domain of traffic analysis. Some firms produce physical equipment that helps collect traffic data, provide end-to-end service according to customers' needs and targets, or create deep learning tools which help annotate raw video data.

1.3.1 Hardware producers

MetroCount (`https://metrocount.com`) is an Australian company that provides hardware capable of gathering traffic data and classifying vehicles. Their monitors collect information about speed, direction, vehicle volumes and gaps. The device is composed of a box placed on the side of a road and two to four pneumatic tubes taped to the surface of the road (Figure 1.2). [12]

Figure 1.2 RoadPod VT4 [12]

The company claims that its monitors are the most reliable ones currently available on the market, with an accuracy of counting exceeding 95%.

In 2018, MetroCount's bike monitoring technology was used to survey traffic on bike roads in Amsterdam. Based on the collected average speeds and percentage of bikers and scooter riders, the municipality decided to ban the second group from bike roads, which resulted in a remarkable decrease in accidents involving scooters. [13]

1.3.2 Traffic engineers

Companies worldwide focus on providing an end-to-end traffic-related service tailored to customers' needs. They plan all the experiments, gather required data, analyse, and deliver reports to the customer.

TCS (`https://tcsforsurveys.com.au`) offers a wide variety of services, not only traffic, intersection and pedestrian surveys, but also origin-destination or travel time surveys. [14]

Yarger Engineering (`http://www.yargerengineering.com`) focuses mostly on providing general traffic impact (Figure 1.3), parking and operation studies in the USA. Moreover, they can deliver forecasts, speed limit studies, and suggest safe routes to schools. [15]

A usual customer of traffic engineering companies is a city or a state authority, developers, and architects who need to understand how specific infrastructure changes impact the traffic flow and safety on-road.

Ordering a study from a well-established and experienced company that possesses all the necessary equipment and has people capable of correctly interpreting data and making conclusions is often much cheaper and more reliable than trying to perform the job on one's own.

1.3.3 Deep learning-based software

Exploration of new deep learning techniques is extremely popular these days. Countless research teams at universities and private companies invest a lot of time and money to develop deep learning models for solving real-world problems. Traffic video analysis is no exception.

In this subsection, we introduce three companies established in the Czech Republic which build software solutions for managing traffic in smart cities, namely, *GoodVision*, *Data from sky* and *Certicon*.

While the first two focus solely on traffic analysis, Certicon's tooling for image analysis (`https://www.certiconvis.cz`) can be customised to analyse a shopping centre's visitors' movement, count the number of free parking spots at a parking lot or increase security at an airport. A major downside of this solution is the required interaction with a customer to understand their needs and tailor the product accordingly. [16]

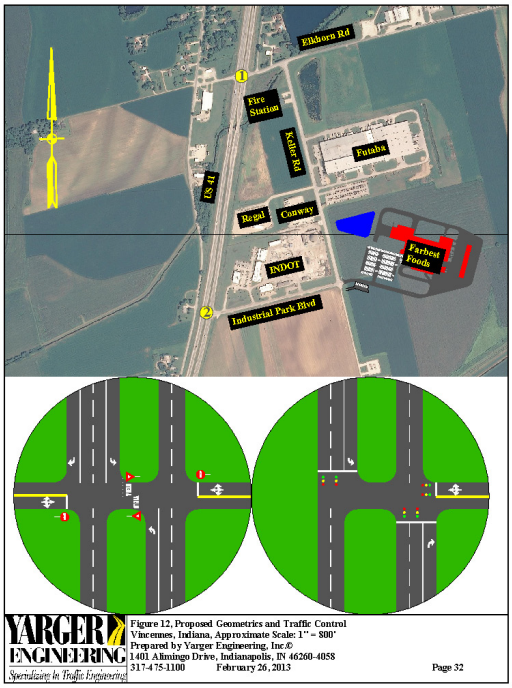


Figure 1.3 Traffic impact study in Indiana [15]

On the other hand, *GoodVision* offers a cloud-based solution entirely concentrated on traffic. This robust tool is called *Video Insight* and it is available at `https://my.goodvisionlive.com/en/login`. Figure 1.4 shows a preview of the service's user interface and functionality. This complex tool allows users to perform the following jobs:

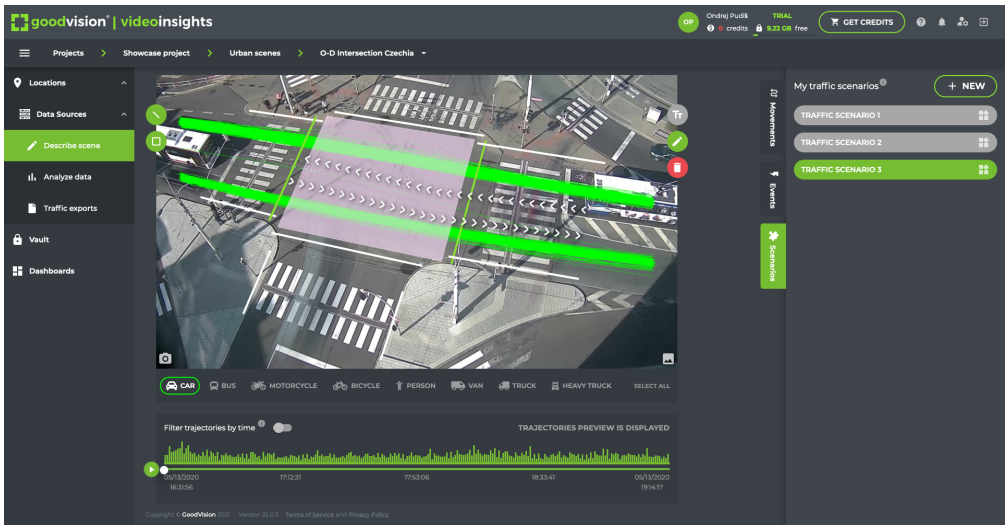
- to upload video recordings or connect live cameras,
- to describe a traffic scene by drawing lines and areas directly on the video,
- to analyse and perform counting,
- to classify objects into eight classes.

Video recordings can come from fixed cameras, time-lapse, and even drone views. They claim that data extraction usually takes up to an hour, and the results are 95% to 100% accurate.

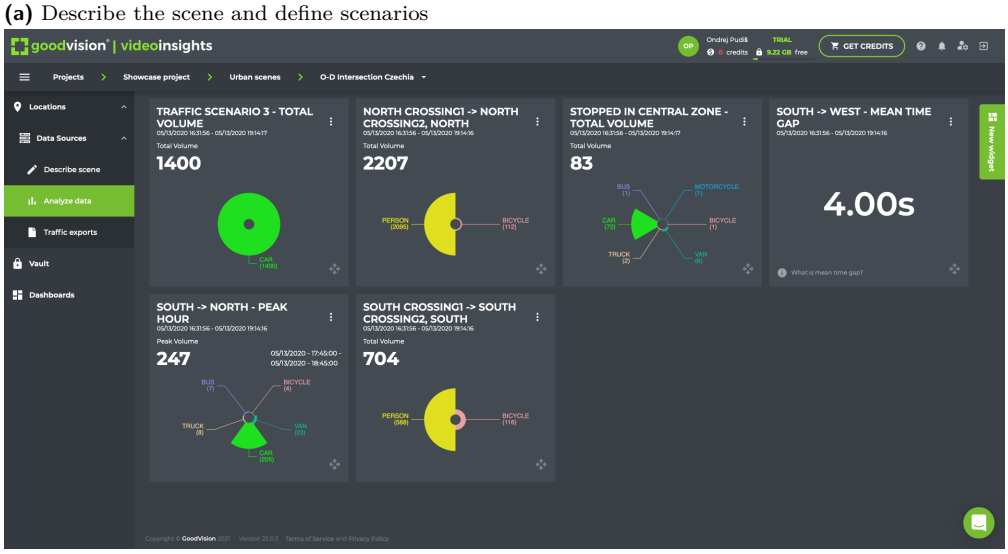
The company expects users to use their cloud-deployed version, but they also offer the possibility to run the platform on a client's hardware. Clients can pay monthly (or yearly) subscriptions or buy credits and use those for running an analysis if they rarely utilise the platform. Prices start at €15 per credit and €199 for a monthly *Traffic Surveyor* subscription plan. [17]

Finally, *Data from sky* provides a tool named *Flow* which they refer to as a traffic framework, the ultimate tool for traffic analysis (`https://datafromsky.com/flow`). They mention many different use cases for their technology such as:

- adaptive traffic control,
- dangerous situations monitoring,
- smart parking and
- pedestrian and cyclist safety.



(a) Describe the scene and define scenarios



(b) Analyse data and see statistics

Figure 1.4 GoodVision Video Insight Tool

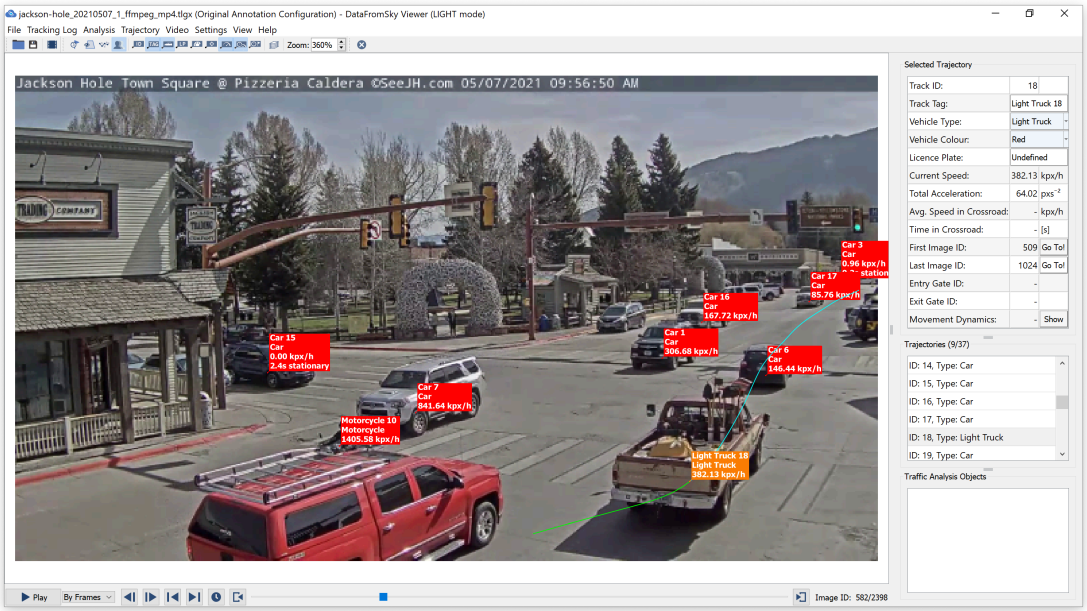


Figure 1.5 Data from Sky Viewer

Flow is also a visual programming language that allows setting up various scenarios and triggering actions in the streets. The company targets their products on municipalities, aspiring to turn them into smart cities. Moreover, the company produces their own camera hardware, which they equipped with all the processing power required for on-device deep-learning video analysis. This decreases data transfer as only extracted data is sent to a server node.

The company also offers a light-weight online tool *Data from Sky Light*, available at `https://ai.datafromsky.com/light`, which analyses an uploaded video and lets the user download the results and see them in a Windows-only tool *Data from Sky Viewer* (Figure 1.5).

While the pricing of the Flow framework solution is not public, the Light version has credit-based pricing where processing one hour of footage costs one credit. The credit's price differs based on the purchased quantity. The cheapest we can get is €2.90 per credit when buying 5000 of them. On the contrary, when purchasing just a single credit, its price is €4.20. [18]

According to researchers from FTS CTU, the significant problems of the platforms are a restricted possibility of vehicle categorisation and higher error rates for footage where vehicles hide behind obstacles or disappear from the scene for a while. [1]

In this thesis, it is not our intention to compete or compare with the companies mentioned above whatsoever. They invest a lot of resources into the research, have significant experience in the field, and own high-quality private datasets used for training and evaluation.

Chapter 2
Theory

In this chapter, first, we make a brief introduction to *deep learning* while focusing on the image processing applications.

Second, we introduce the problem of *object detection*, various approaches and architectures of models that address it.

Third, we move to the field of statistical theory and introduce *Kalman filter*, followed by *Hungarian algorithm*. Both of these algorithms are a vital part of the *object tracking* methods.

Finally, we comment on *multi-object tracking (MOT)* which is the problem that our thesis focuses on finding a solution for.

2.1 Deep learning

The term *deep learning* refers to a subset of machine learning models called *deep artificial neural networks*. They are networks composed of a single input layer, at least two hidden layers and an output layer, while each layer consists of multiple units. A neural network with just a single hidden layer is usually considered a *shallow* neural network.

A single unit, *perceptron*, performs a weighted sum of its inputs, applies an activation function and outputs this single number. Popular choices of activation are *sigmoid*, *rectified linear* or *softmax* functions.

Let us assume an output $y^{(i,j)}$ of a j-th perceptron on a i-th hidden layer that has n inputs:

$$y^{(i,j)} = \phi\Big(b^{(i,j)} + \sum_{k=1}^{n} w_k^{(i,j)} x_k\Big) \tag{2.1}$$

where ϕ is an activation function, x is a input vector, $b^{(i,j)}$ is a trainable bias and $w^{(i,j)}$ are trainable weights.

A neural network training process comprises two phases. The first one is a *forward* pass, which evaluates the outputs for a given input and compares the result to the ground truth. The error is then passed back to the network in the *backpropagation* phase, which updates the weights of the units to minimise the error. This procedure is called *supervised learning*.

Even though this thesis's goal is not to make an extensive introduction to deep learning, we dedicate the rest of this subchapter to a brief explanation of those types of neural networks extensively used in our work. For a more thorough understanding of the topic, we refer to other literature, for example, Introduction to Deep Learning by S. Skanosi [19], [20] or Deep Learning by I. Goodfellow [21].

Convolutional neural networks (CNNs), initially introduced by Lecun et al. [22], is usually used when a strong grid-like topology of input data is observed. They are a massive success, especially in the field of image processing.

Instead of connecting each input unit to an output unit as *fully-connected* layers do, convolutional layers repeatedly apply a *convolution* operation over small neighbourhoods, extracting local features such as corners, endpoints or edges. This approach has its foundations in biology because, as it was discovered in the 1960s, this is how animals' visual receptive field works [23].

2.1.1 Convolution

Convolution, in general, as described by Goodfellow [21], is a mathematical operation on two real-valued functions f and g, often noted as $(f * g)$ and defined as:

$$(f * g)(x) = \int f(a)g(x - a)\, da. \tag{2.2}$$

If we restrict the range of x to take on just integer numbers, we can define discrete convolution as:

$$(f * g)(n) = \sum_{a=-\infty}^{\infty} f(a)g(n - a). \tag{2.3}$$

In machine learning, f is referred to as *input*, g is a *kernel* and the output of the convolution is a *feature map*.

If we consider the input a 2D image, the kernel is also 2-dimensional. The infinite sum shrinks only to the image size because the value is 0 everywhere else. Moreover, because of convolution's commutative property, we can define 2D convolution as:

$$(K * I)(i, j) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n). \tag{2.4}$$

2.1.2 Convolutional layer

It is common for a convolutional layer to perform multiple convolutions at once and output a feature map that contains more channels than the original input image. Skansi [20] suggests that this increases the chance for the kernels to learn good representations of local features, which can result in better overall performance.

Let us give an example. We have a grey-scaled image of size 100×100, and we pass it through a single 2D convolutional layer with 9 filters and a 3×3 kernel. The output of such a layer is a feature map $98 \times 98 \times 9$ while the number of trainable parameters is 90 (9×9 multiplication weights, 9 biases).

On the other hand, if we wanted to pass the same image through a fully-connected layer, we would first need to flatten the 2D vector, effectively removing local connections. Secondly, we would define the output size of the fully-connected layer on which the number of parameters would depend. If there were only a single neuron on the output, there would be 10 001 parameters, 500 output neurons would mean more than 5 millions of parameters to train.

Convolutional layers, therefore, bring additional benefits and improvements to both the training process and resulting quality scores:

- input data size is not strictly given by the size of the input layer,
- sparse connections between units allow for a significant decrease in the number of trained weights,
- local correlations of input data are taken into account. [22]

Goodfellow et al. [21] further argue that both sparse connections and parameter sharing contribute to decreased computational complexity and low memory requirements.

Figure 2.1 An example of visualized general object detection [24]

2.1.3 Pooling

Two additional steps typically follow convolution. The first one is applying a non-linear activation function, and the second one is *pooling*. More complicated architectures might add additional steps like batch normalisation, residual connections or dropout for regularisation.

The most common choice of the activation function with convolutional layers is rectified linear unit (ReLU):

$$\rho(x) = \max(0, x). \tag{2.5}$$

A pooling function replaces all values from a small neighbourhood defined by the pool size with a single value that can be either a maximum, a minimum, an average, or other statistical operation [21].

According to Skansi [20], the motivation behind pooling is that it drops useless information, which adjacent pixels often include. A pooling layer of size 2×2 decreases the dimension of the feature map to a half but keeps the number of channels untouched. Skansi refers to this step as decreasing the resolution of the original image.

If we applied pooling to the example as mentioned earlier image, we would get a feature map with dimension $49 \times 49 \times 9$.

2.2 Object detection

Object detection is a fundamental computer vision task that aims at localising and classifying objects in an image, often providing confidence in the detection. Localisation outputs coordinates of boxes that bound the objects while classification assigns the objects a class they belong to. Figure 2.1 shows an example of how the detections look when drawn in the original picture.

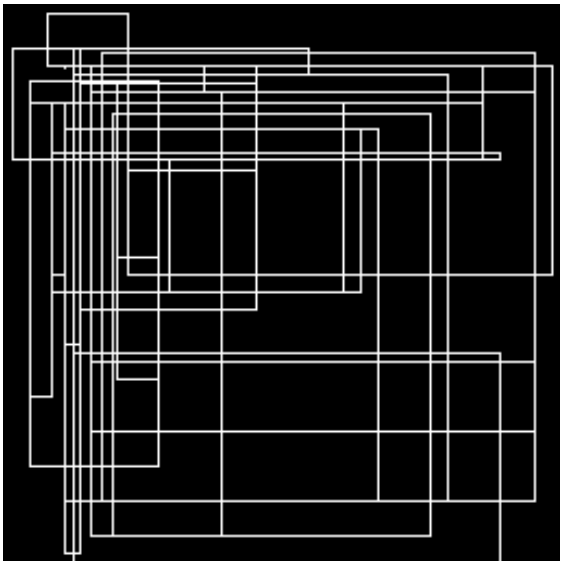


Figure 2.2 Generated anchors on an image of size 256×256

Object detection has become increasingly popular because it helps with world-class problems such as human pose estimation, autonomous driving, traffic control, security and medicine. Nonetheless, it is also a controversial topic as it can be easily misused for tracking people or analysing their behaviour. [25]

2.2.1 Anchor-based detectors

Models which belong to this category generate thousands of rectangular regions of interest called *anchors*. They are of various sizes and shapes to cover as many objects in the image as possible, often overlaying one another. Figure 2.2 shows an example of how anchors can be distributed across an image.

Convolutional architecture is crucial as it extracts a feature map of the image. Researches often utilise already existing and pre-trained models such as ResNet or EfficientNet. This measure saves a lot of effort and computational resources.

Next, the anchors are used to build feature vectors. Then, one sub-net assigns the feature vectors a probability of how likely the areas are background or belong to predicted classes. Another sub-net generates the final bounding boxes coordinates. [25]

Since anchor-based models have to process large numbers of anchors, they are computationally expensive. On the other hand, they yield better results than anchor-free detectors. [26]

2.2.1.1 Single-stage

Such models consist only of a single place where anchors are generated. They usually follow the same distribution. Popular models are *Single Shot MultiBox Detector* (SSD) [27], *RetinaNet* [28] or YOLOv2 [29].

This approach allows for faster inference. On the other hand, it introduces a considerable class imbalance into training which means that most training samples belong to the background and do not contribute any helpful input to the training process.

RetinaNet [28] tackles the class imbalance problem by introducing a new *focal loss* and combines it with *feature pyramid network* architecture [30]. Outputs are provided by a classification

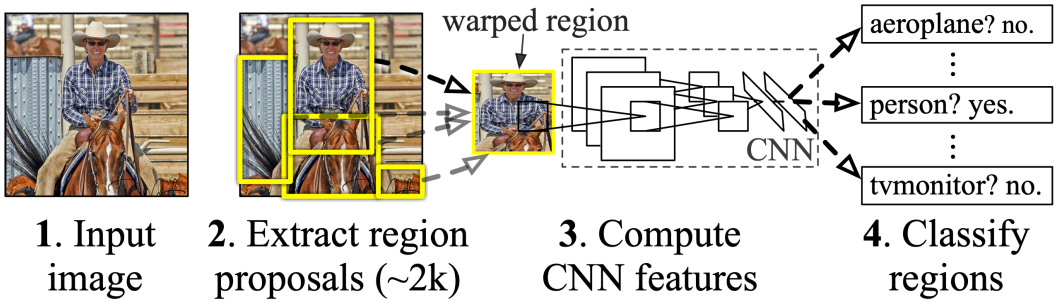


Figure 2.3 R-CNN object detection architecture [31]

and bounding box sub-nets. The former provides a classification vector for each of the anchors, while the latter computes the relative position of the bounding box to the original anchor.

2.2.1.2 Two-stage

These models add one additional step at the beginning of the whole pipeline. Instead of generating anchors from the whole picture, they first propose a set of candidate regions covering all the objects. The rest of the image is filtered out and labelled as a background.

Figure 2.3 shows an architecture of the R-CNN (Regions with CNN) model, which was a breakthrough in the field of object detection back in 2014 [31].

It was followed by other models like Fast R-CNN [32] and Faster R-CNN [33]. The second one further improved performance by introducing new ways of how region proposals are computed or taking advantage of the feature pyramid networks already mentioned in 2.2.1.1.

Zhao et al. [25] claim that this approach somewhat imitates the behaviour of the human brain, which scans the scene first and then focuses only on identified regions of interest.

2.2.2 Anchor-free detectors

In contrast with anchor-based detectors, anchor-free ones locate objects in an image without predetermined anchors. Consequently, it removes all the anchor-related hyperparameters. There are two main categories of anchor-free models: keypoint-based and center-based. [26]

Tian et al. [34] further point out that anchor-based models require careful, heuristic tuning of some of the hyperparameters, like intersection over union (IoU) to get good performance.

2.2.2.1 Center-based

You only look once (YOLOv1) [35] split an image into a $S \times S$ grid and generate bounding box proposals of various sizes and aspect ratios per each grid cell while keeping the centre of the box in the cell itself. As stated by the authors themselves, the model suffers from issues with small objects' detection and failures to generalise on unobserved aspect ratios of known objects.

In *Fully Convolutional One-stage Object Detection* (FCOS) [34] authors suggest an approach where each (x, y) location on a feature map is considered as an object center point. A 4D $(l*, t*, r*, b*)$ vector is regressed for the center points that contain objects. These vectors define the bounding boxes positions (Figure 2.4).

The authors further identify a flaw of numerous bounding boxes being far away from actual object centres. They decided to address it by introducing a metric they call *centreness* applied to the final detection score.

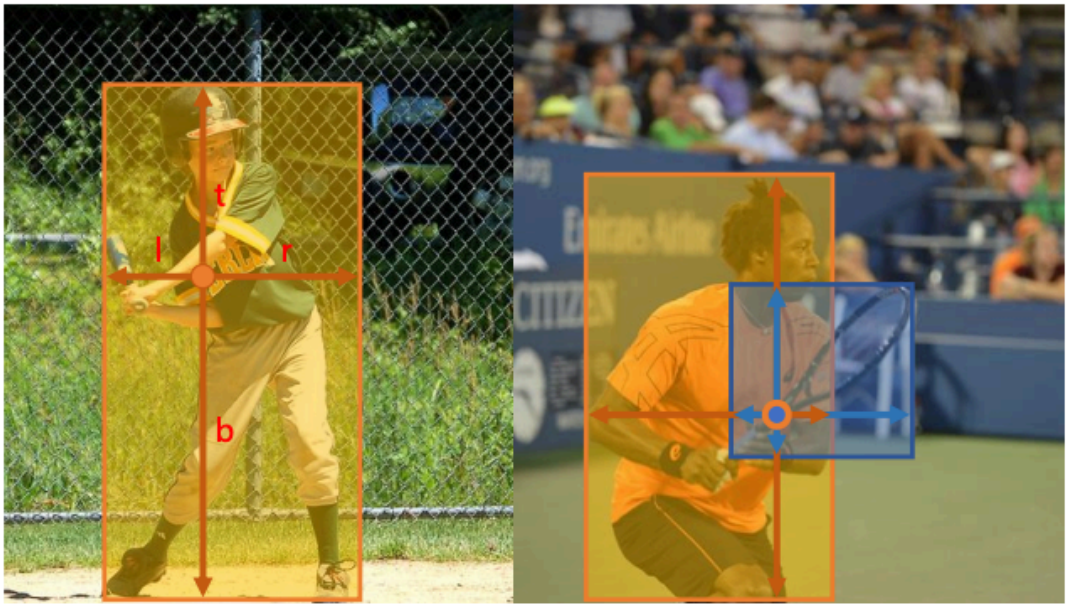


Figure 2.4 FCOS center-based detection [34]

2.2.2.2 Keypoint-based

Keypoint-based models utilise either predefined or self-learnt sets of keypoints for defining the bounding boxes.

CornerNet [36] detects an object as a pair of top-left and bottom-right corners coordinates. The authors argue that corners are easier and more efficient to learn than centres or anchors.

2.2.3 EfficientDet

EfficientDet [37] is a family of single-stage, anchor-based models which aims at scalability, efficiency and accuracy.

Increased accuracy while keeping complexity rational is achieved by combining the *EfficientNet* backbone, which is responsible for feature map extraction with the proposed weighted bi-directional feature pyramid network (BiFPN).

They achieve scalability by introducing compound scaling to the BiFPN. This method is originally presented in the article about EfficientNets [38] and reused in EfficientDets. It allows for efficient scale-up when more resources are available.

The authors present 8 different versions of the architecture denoted as **D0**, **D1**, ..., **D7** (similarly to EfficientNet's B0, B1, ..., B7). The higher the number, the better the accuracy, nonetheless, more parameters and higher computational complexity.

2.2.3.1 EfficientNet

The breakthrough that EfficientNet brings is the *compound model scaling* method. It uniformly scales the network in width, depth and resolution. The paper argues that all of these dimensions must be balanced in order to get improvement in both accuracy and efficiency. Usually, networks are scaled only in one of the parameters, trading performance for the accuracy or vice versa.

The authors define a baseline **B0** architecture composed of multiple convolutional and inverted residual blocks [39] of various kernel sizes and channel dimensions, outputting features at

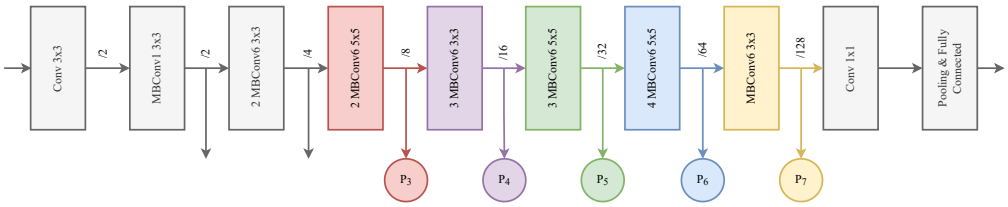


Figure 2.5 EfficientNet **B0** architecture, based on [38]. An image flows through convolutional and inverted residual blocks of 3 or 5 kernel sizes, always half in size, but multiplied in depth.

several levels P_i. Figure 2.5 shows the architecture, highlighting the feature map levels taken as an input to the feature pyramid.

This architecture scales according to a parameter ϕ and forms the whole EfficientNet family. The scaling methodology enables for usage of EfficientNet models on both mobile and GPU-heavy devices.

2.2.3.2 BiFPN

Weighted bi-directional feature pyramid network (BiFPN) is a representative of *multi-scale feature fusion* methods which aggregates feature maps at different resolution levels.

Such methods usually use the output at level P_3 to P_7 and aggregates in different manners, including top-down (feature pyramid networks [30]), top-down and bottom-up (PANet [40]) or cross-scale which connect different feature levels.

BiFPN is a cross-scale method with several new ideas, such as the removal of nodes with just a single input or connecting input with output nodes on the same layer. The cross-scale connections require resizing of features coming from the other layers before summing them up since their dimension is different. The authors propose a *fast normalized fusion*. It features a trainable parameter on each level that allows the network to learn which features are important during the sum-up:

$$P_5^{\text{out}} = Conv\left(\frac{\hat{w}_1 P_5^{\text{in}} + \hat{w}_2 P_5^{\text{middle}} + \hat{w}_3 Resize(P_4^{\text{out}})}{\hat{w}_1 + \hat{w}_2 + \hat{w}_3 + \epsilon}\right) \tag{2.6}$$

Figure 2.6 shows the architecture of a single BiFPN block, which can be repeated multiple times in the actual network. The outputs of the last BiFPN block are connected to two networks that predict object classes and the bounding boxes.

Further details, including the scaling configurations for the EfficientDet family, are available in the original paper by Tan et al. [37].

2.2.4 Non-max suppression

Anchor-based object detection models generate thousands of proposed bounding boxes. Hence, it easily happens that the same object is detected multiple times, as shown in Figure 2.7a.

Non-max suppression (NMS) is a greedy algorithm that removes duplicate detections based on intersection over union and confidence rule. If two or more bounding boxes significantly overlap (IoU is greater or equal to a given threshold), only the bounding box with the highest confidence score is kept; the rest is suppressed.

Bounding boxes that do not score over a given confidence score get suppressed as well in order to decrease the number of falsely detected objects.

In our implementation, we use an improved version of the algorithm called *Soft-NMS* which improves the original technique by introducing a confidence score decaying function based on IoU, instead of directly comparing the IoUs. [41]

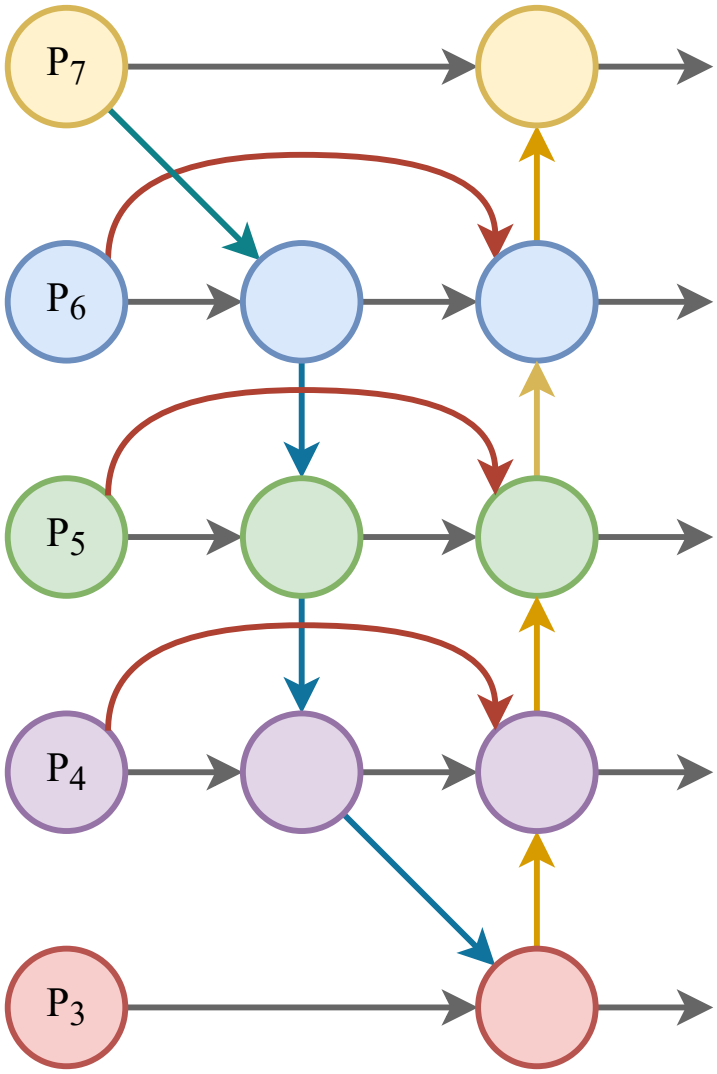
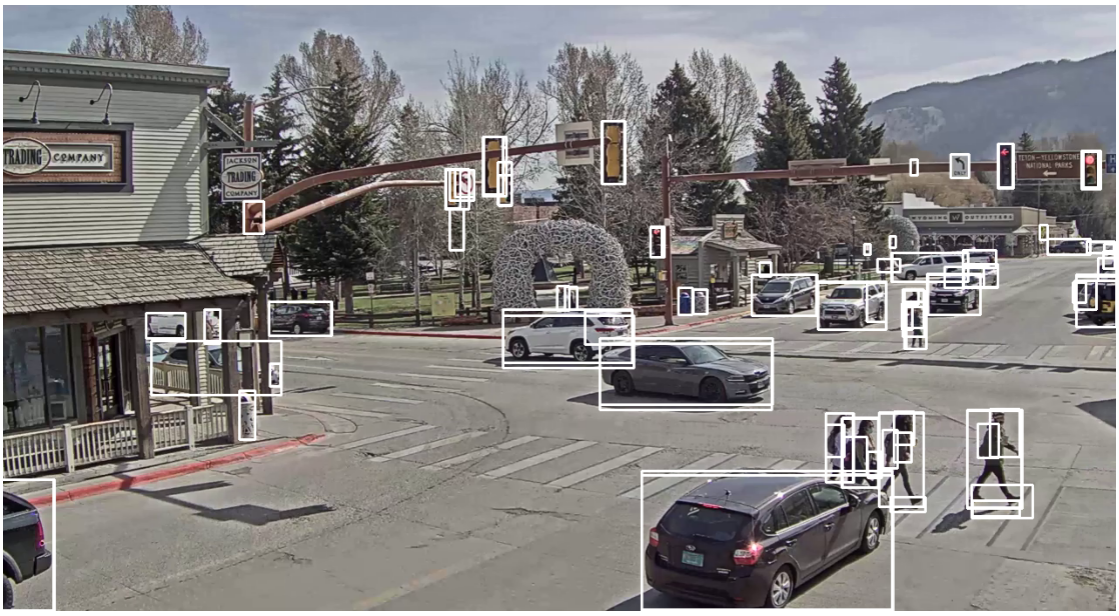


Figure 2.6 A single BiFPN block architecture, based on [37]. The arrows indicate the directions how information gets to the nodes. The cross-level connections apply a resizing function to get the values from different levels to the same dimension.



(a) No non-max suppression applied

(b) Non-max suppression with minimum 0.1 confidence score and maximum 0.75 IoU

Figure 2.7 NMS effect on object detection

2.3 Kalman filter

Let us now diverge from the topic of deep learning to statistics and talk over the Kalman filter, which is of the essence for algorithms introduced in section 2.5.

The Kalman filter, originally introduced and proved by R. E. Kalman in 1960 [42], is a statistical method that solves the problem of estimation.

Say, we receive values $y(t)$ that come from a measurement device that tracks a signal $x(t)$. Because the device does not measure perfectly we need to account for a noise: $y(t) = x(t) + n(t)$ [42].

In other words, the Kalman filter approximates the unobservable state of a system by observed, often noisy, data. It assumes that the measurements, the hidden state, and the noise are coming from a Gaussian distribution, and the system is dynamic and continuous.

Representation of the system's state depends on the given use case, but in general, it is a vector of numbers. For example, a single unit like height, weight or price, (x, y, z) coordinates or a combination of position and velocity of an object.

The Kalman filter is just one of the methods belonging to the Bayesian filters family. It can be applied in robotics, radar tracking or signal processing for data-smoothing, filtering or predicting the next state. [43]

It consists of two steps, namely, a prediction and a correction steps. While the first one estimates the next state s_{t+1} using the state transition equation, the second one amends the estimate based on the provided measurement. The correction step is skipped if no measurement is available at $t + 1$.

2.3.1 Prediction step

The prediction step applies a *linear state transition equation* to extrapolate the next state of the system based on the current one:

$$s_{t+1} = Fs_t + Gu_t + w_t, \tag{2.7}$$

where F is a state transition matrix, G is a control matrix, u_t is a control input and w_t is Gaussian noise [44].

The state is a Gaussian distribution parameterised by mean s_t and covariance matrix P_t where P_t represents the uncertainty of the current state. It has to be updated alongside the state:

$$P_{t+1} = FP_tF^T + Q, \tag{2.8}$$

where Q is a process noise matrix.

2.3.2 Correction step

After the prediction step concludes and we receive a new measurement y_t, we can use it to correct the current state s_t through the *state update equation*:

$$s_{t+1} = s_t + K_{t+1}(y_{t+1} - Hs_t), \tag{2.9}$$

where H is a measurement matrix that transforms the state s_t to the same dimension as the measurement; K_t is the Kalman gain.

Introducing the gain is, according to [43], the most valuable contribution of Kalman's original work:

$$K_{t+1} = P_tH^T(HP_tH^T + R)^{-1}, \tag{2.10}$$

where R is a measurement uncertainty. The intuition behind K_t is that during the update it gives more importance to the measurement or the prediction based on current uncertainty covariance matrix. For the definition of *uncertainty update equation* we refer to [44].

2.3.3 Summary

When formulating a task as the Kalman filter problem, we need to define the spaces, the constants and the initial values required by the algorithm. Below we list all of them, including their dimensions in brackets. First, the space sizes:

- the measurement space y $(d_y \times 1)$,
- the state space s $(d_s \times 1)$,
- the control space u $(d_u \times 1)$.

Then, the constant matrices:

- the measurement matrix H $(d_y \times d_s)$,
- the state transition matrix F $(d_s \times d_s)$,
- the control matrix G $(d_s \times d_u)$,
- the process noise matrix Q $(d_s \times d_s)$,
- the measurement uncertainty matrix R $(d_y \times d_y)$,
- the gain matrix K_t $(d_s \times d_y)$.

Finally, the initial values for the mean s_0 $(d_s \times 1)$ and covariance P_0 $(d_s \times d_s)$. [44]

2.4 2D Rectangular Assignment

The 2D assignment problem is a combinatorial optimization problem, also known as a *linear sum assignment*, defined on a matrix of **costs C** of dimension $(N_R \times N_C)$, $N_C \geq N_R$.

The result is a matrix **X** of assignments which picks exactly one column for each row in a way that the sum of costs is minimized or maximized. Written mathematically [45]:

$$\begin{aligned} \mathbf{X} &= \arg\min_{\mathbf{x}} \sum_{i=1}^{N_R} \sum_{j=1}^{N_C} C_{i,j} x_{i,j} \\ \text{subject to} \sum_{j=1}^{N_C} x_{i,j} &= 1 \, \forall i, \\ \sum_{i=1}^{N_R} x_{i,j} &\leq 1 \, \forall j, \\ x_{i,j} &\in \{0, 1\} \, \forall i, j. \end{aligned}$$

Many algorithms solve the problem, including a popular, so called *Hungarian method*. These algorithms are based on linear programming, maximum flow or shortest path methods. While the older methods offered a $\mathcal{O}(n^4)$ complexity, the newer approaches reached $\mathcal{O}(n^3)$.

Jonker-Volgenant algorithm [45] is one of the popular choices. They take the existing shortest augmenting paths-based algorithms and suggest changes to the initialisation and augmentation steps of the algorithm.

The disadvantage of the original paper and implementation is that it works just with squared cost matrices. Castanón et al. [46] introduced a modified version of the algorithm that generalised for rectangular matrices. Hence, the currently used implementation is referred to as Jonker-Volgenant-Castanón algorithm.

2.5 Multi-object tracking

Multi-object tracking (MOT) or *multi-target tracking* is another task of computer vision. In a sense, it is an extension of object detection, explained in section 2.2, that maintains identity of the objects across multiple frames.

Given a series of video frames, MOT algorithm extracts a set of trajectories of objects, usually pedestrians, vehicles, animals or players on a pitch.

In their literature review, Luo et al. [47] define the task as a *maximum a posteriori* estimation from a distribution of sequential states $S_{1:t}$ of objects in scenes given observations $O_{1:t}$.

There are several challenges and obstacles related to MOT, notably:

- occlusion (one object blocks a view on other ones),
- initialisation and termination of a single trajectory,
- distinction of similarly looking objects and
- the choice of similarity measurement between objects.

The most commonly used approach to the problem is detection-based tracking or a so-called *"tracking-by-detection"*. It consists of two steps. First, a detection backbone generates proposals of objects it can see in each frame. Second, the tracking algorithm assigns identifiers to the objects and merges them into trajectories.

The two-staged approach benefits from the automatic discovery of new objects to track and destroying the ones that have left the scene. The main drawback is decreased performance caused by the usage of the detection backbone. Moreover, pre-training of the backbone restricts its usage across various scenes.

An alternative to the above is a *detection-free* method that removes the requirement for the detection model. Nonetheless, it takes a manual input of initial objects to track in the subsequent frames. The number of trajectories remains constant for the whole video.

The MOT models work either in *online* or *offline* mode. The first ones use only the information from the past and the current frames to determine the next states. They process the video sequentially. On the contrary, offline models utilise information from a batch of frames both in the past and future to estimate the final states.

Next, the choice of an optimisation method used to solve the MAP formulation mentioned above divides the models into *probabilistic* and *deterministic*. While probabilistic models apply a portion of randomness into the process, deterministic ones return constant output to the same input data. [47]

A MOT model is composed of multiple optional components. They contribute to the ability of computing similarity between objects, address occlusion and re-identify existing objects.

Luo et al. [47] identify six main components: *appearance*, *motion*, *interaction*, *exclusion*, *occlusion* and *inference*. Usually, and also in our work, appearance, motion and inference components are compulsory. The rest of them is rather scenario-specific. For instance, crowded scenes would need an occlusion component incorporated in order to handle them with a high success rate.

The *appearance* component addresses a visual representation of the object and its region. It can be a pixel representation, histogram or CNN-based feature representation.

The *motion* component aims at capturing the movement dynamics of an object. This is important for reducing the space where the tracked object can appear in future frames.

The *inference* component takes care of determining the final trajectories. The probabilistic approach, usually an online method, estimates a distribution of the next states. Kalman filter (section 2.3) is a good example of such approach with *predict–update* cycle in place. The deterministic approach takes advantage of optimisation algorithms to find the best-suited trajectories in a batch of frames.

More information on the topic can be found in Luo et al.'s comprehensive review [47].

Figure 2.8 An example of multiple vehicles movement tracking. The same vehicle is bordered with the same color.

Chapter 3

Related work

In the following sections, we aim to describe how other teams of researchers approach the problem of multi-object tracking and vehicle detection, tracking and counting. We commence the chapter with early vehicle tracking methods and general MOT techniques. Next, we follow with an example of an end-to-end deep-learning-based method and end with techniques tailored for vehicle counting.

In fact, we have already commented on existing commercial products, both physical devices and software products in section 1.3. Instead, this chapter focuses on academic papers suggesting various methods and procedures that tackle the problem.

On-roadway vehicle tracking problem was prevalent even in the late '90s and the early '00s. Back then, neural network approaches were not popular due to insufficient computational performance. Therefore, researchers widely used algorithms of *computer vision*, probabilistic methods and direct examination of images' properties.

Betke et al. [48] come up with a real-time vehicle detector that acts as a control system capable of monitoring the environment and taking actions in dangerous situations. Their camera is placed inside the vehicle, just behind the windshield, covering a wide range in front and on the sides.

When a vehicle is close to ours, it covers a considerable part of the image, which causes a noteworthy difference in brightness. Thus, looking for changes in brightness over a couple of frames allows the authors to detect vehicles approaching from behind or getting close in the opposite direction.

The team notices that this is not good enough for distant vehicles' recognition; hence, they introduce another method that evaluates vertical and horizontal edges in the frames.

Other works, similarly to us, focus on tracking cars from a stationary camera placed high above a road to cover as much of the driving area as possible. Koller and Malik [49] introduce a method based on motion segmentation, contour representation and affine motion model of the tracked objects. They split the estimation process into two parts - two Kalman filters. First, to estimate the affine motion parameters, and second, to approximate the contour's control points, represented as a closed cubic spline.

Finally, open-source projects are available that implement the well-known and widely-used models and methods in a single repository. This can save a substantial amount of time needed to develop a new MOT model. To give an example, `MMTracking` [50] implements a few video-object and multi-object tracking methods.

3.1 Simple Online and Realtime Tracking

Bewley et al. [51] suggest an algorithm called "Simple Online and Realtime Tracking (SORT)", a simple yet effective and fast framework for the tracking-by-detection problem, building on top of CNN-based detection. According to their findings, the detection accuracy significantly impacts the tracking quality. This is a noteworthy discovery for our work, suggesting to use a detection model that gives precise and confident information.

In their work, Bewley et al. focus on applying the suggested procedure to pedestrian tracking. Nonetheless, the detection model is easily replaceable, allowing for generalisation to different objects.

The contribution of their work is experimenting with CNNs, founding the tracker on Kalman filter (motion component) and Hungarian algorithm (inference component) and open sourcing their code to the public.

The pipeline of the framework is straightforward. First, a frame at time t passes through the detection model, which outputs a list of detected objects' bounding boxes D_t. Detection models usually output bounding boxes as a list of top-left and bottom-right coordinates.

Second, Kalman filter prediction step (section 2.3) is utilized to estimate the positions E_t of objects at t based on the state at $t-1$.

Bewley et al. design a linear constant velocity model:

$$x(t) = x_0 + v_0 t. \tag{3.1}$$

A single state is represented as:

$$\mathbf{s} = [c_x, c_y, a, r, \hat{c_x}, \hat{c_y}, \hat{a}], \tag{3.2}$$

where c_x, c_y are coordinates of the centre of a bounding box, a is the area of the bounding box, and r is an aspect ratio. $\hat{c_x}$, $\hat{c_y}$ and $\hat{a}$ are the velocity components solved by the Kalman filter.

Third, they form a matrix of intersection over unions between E_t and D_t, called *association matrix*, which describes how much the estimated and actual bounding boxes overlap. Hungarian algorithm (section 2.4) provides a solution to this described assignment problem.

Finally, there are three possible outcomes and actions taken:

1. d_{ti} gets linked to e_{ti}: estimate is corrected according to d_{ti},
2. d_{ti} stays separate: it is a new detection; a new tracker is initialized and marked as *active* if it is detected for T_{new} consecutive frames,
3. e_{ti} stays detached: the tracker is followed for T_{lost} frames and destroyed afterwards if no correction appears.

The actions described above prevent the algorithm from growing the number of tracks to infinity or starting to track false detections.

3.2 Deep Associations Online and Realtime Tracking

The work by Wojke et al. [52] builds on top of SORT introduced in the section above and presents Deep Associations Online and Realtime Tracking (Deep-SORT). They propose a couple of changes with the intention of decreasing the number of identity switches caused mainly by occlusion. They claim to have achieved this goal by cutting the undesired switches down to a half of what Simple Online and Realtime Tracking (SORT) would produce.

This enhancement is achieved through replacing the original IoU association matrix with a cost matrix $\mathbf{C}$ that combines both *motion* and *appearance* information:

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j), \tag{3.3}$$

where $d^{(1)}$ is the motion measurement and $d^{(2)}$ is the appearance measurement. λ is a hyper-parameter that specifies which of the two measurements is given more preference.

Moreover, they slightly alter the state space representation while still sticking to the linear constant velocity model. The area of a bounding box is replaced with its height:

$$\mathbf{s} = [c_x, c_y, r, h, \hat{c_x}, \hat{c_y}, \hat{r}, \hat{h}]. \tag{3.4}$$

The motion information is accounted for though a *Mahalanobis distance* between each state's normal distribution parameters projected into measurement space, denoted as (μ_i, Σ_i), and a new detection d_j:

$$d^{(1)}(i,j) = (d_j - \mu_i)\Sigma_i^{-1}(d_j - \mu_i). \tag{3.5}$$

The distance indicates how far d_j is from the tracker's position. Additionally, it excludes highly unlikely associations by thresholding the distance value on a 95% confidence interval.

The appearance information is generated from pixel representation of the bounding boxes. Each d_i is passed through a standalone CNN which generates a unit-length embedding – *appearance descriptor* r_i. Architecture of the network is described in [52].

The framework keeps a gallery of seen descriptors $\mathcal{G}_i$ and computes a *cosine distance* between each descriptor in the gallery and new detection's descriptor r_j.

$$d^{(2)}(i,j) = \min\left(1 - r_k^T r_j \mid r_k \in \mathcal{G}_i\right). \tag{3.6}$$

It achieves better overall tracking accuracy while keeping the inference fast to operate in real-time.

3.3 FairMOT

Traditionally, MOT frameworks have been using two independent models for detection and subsequent re-identification. The authors of *FairMOT* [53] introduce a single model with a common encoder-decoder backbone followed by two separate detection and re-identification branches.

The two-model approaches, such as SORT or Deep-SORT (section 3.1 and section 3.2, respectively), get slower when many objects appear in scenes as each bounding box is re-identified separately.

The authors identify three reasons why early single-network trackers suffer from a significant drop in tracking accuracy:

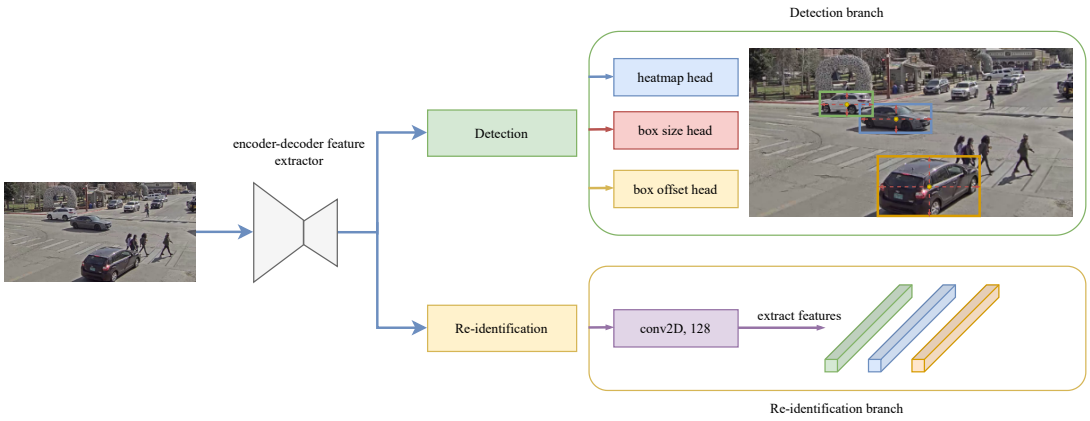


Figure 3.1 FairMOT model architecture, based on [53]

- extracting re-identification features from anchors which they find unsuitable,
- sharing features between detection and re-identification, which are opposite tasks - the first one finds common characteristics while the other distinguishes between instances of the same class,
- insufficient feature dimension for re-identification.

Zhang et al. [53] solve these issues by proposing two heads that treat the tasks fairly. First, an image is processed by a common *ResNet*-like encoder-decoder architecture, which extracts high-level features. Next, these features are passed to both detection and re-identification branches. We visualise the architecture in Figure 3.1.

The detection branch is an anchor-free (subsection 2.2.2) *CenterNet* architecture composed of three heads. One for estimating centres of objects (heatmap head), another for proposing the bounding box sizes (size head) and the last for more precise localisation (box offset head).

The re-identification branch uses just a single convolutional layer with 128 channels to generate features for distinguishing objects. Only features at the centres of actual objects are cherry-picked in a later stage of the algorithm.

The final association of tracklets and detections follows the very same process as Deep-SORT. Both Mahalanobis distance and the cosine distance of the appearance descriptors are taken into account. The descriptors are extracted from the re-identification branch.

According to the authors, the framework ranked first place in the MOT challenge benchmarks in 2021 while still running fast on a single GPU. The algorithm's speed owes to the fact that both detections and appearance descriptors are generated together.

3.4 Vehicle counting framework

Dai et al. [54] propose a framework that counts vehicles based on a video recording. They split the problem into three stages: object detection, object tracking and trajectory processing.

The first two stages follow the algorithms that we presented in the sections above. The last one merges the tracklets into trajectories, infers the directions and classifies the vehicles into three categories: car, bus and truck.

The framework's output is a matrix with counts of vehicles that pass in specific directions grouped by the class they belong to, as shown in Figure 3.2. It is achieved by clustering the trajectories' start and end points together with their directions. The sizes of the clusters are considered to be the results. However, the team fails to provide detailed information on how they identify the number of clusters or find out which cluster represents the final categories.

They report achieving 87.6% accuracy on their custom dataset using YOLOv3 and a simple Kalman filter-like tracking method.

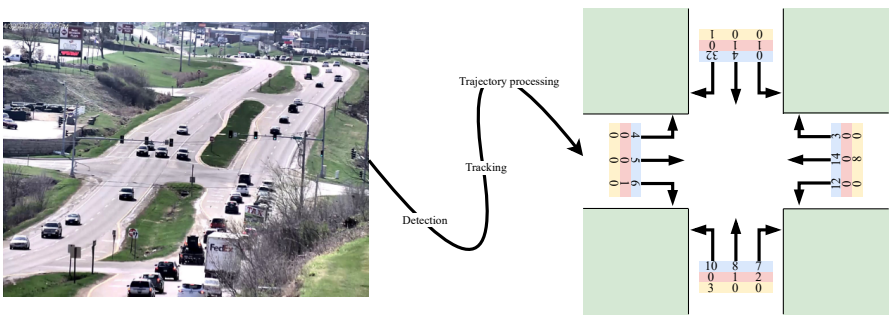


Figure 3.2 Vehicle counting framework visualisation, based on [54]

Chapter 4
Analysis

One of the main deliverables of our work is proof-of-concept software. In this chapter, we clarify the problem we are trying to solve. Next, we list the functional and non-functional requirements the application needs to meet. Finally, we propose our software solution and give motivations for our decisions during the design. The details of the pipeline that transforms a video into numbers and the implementation are presented in chapter 5 and chapter 6 respectively.

4.1 Problem

As already mentioned in chapter 1, this thesis automates the evaluation phase of traffic surveys.

It is the phase when a person needs to go through a pre-recorded traffic video and manually or semi-manually, taking advantage of specialised software, count the vehicles while categorising them. The expected output reports the number of vehicles that pass the scene, aggregated per hour. This process usually takes three to seven days to complete [2], depending on the length of the video.

Typically, it is 24 or 48 hours long, 20-25 FPS, recorded in HD. This configuration is a good trade-off between saving storage and battery life of cameras while keeping the quality high enough for people and machines to recognise objects in the scenes.

The human-counting approach of a pre-recorded video has the advantage of a low error rate, usually around $3 - 5\%$, whereas performing the counts directly on the roads increases the error to 10%. The automatic tools such as those mentioned in section 1.3 can be up to 30% off the correct number. [2]

Classification of the vehicles is a delicate problem on its own. It seems to be quite a simple task at first glance, but traffic researchers know there are multiple classification standards and guidelines to follow. Let us list just a couple of them:

5-category classification is the most common classification of wheeled vehicles: bicycle, motorbike, passenger car, truck and bus [2].

United Nations ECE standard introduces categories L, M, N, O, T, R, S, G that represent vehicles from motor-bikes to off-road vehicles. These are split into several subcategories based on their maximum weight, engine capacity, number of wheels, speed, etc. [55]

EU vehicle categorisation defines four primary categories: a light-duty vehicle, a heavy-duty vehicle, a agricultural or forestry tractor and 2-3-wheel vehicles. These are divided into subcategories based on how many axles they have. Trucks are additionally branched to non-trailer and trailer. [56]

Czech Technical Norm (ČSN) introduces the types of road vehicles and their classification valid in the Czech Republic [57].

The differences between vehicle types are sometimes so difficult to tell apart in a video recording that both people and machines make many mistakes, especially with trucks. Therefore, a two-category classification to *passenger* and *slow* vehicles is popular in traffic modelling. [2]

4.2 Requirements

Now that we have defined the problem, we look at various functional and non-functional requirements which we need to meet to make our application usable.

4.2.1 Functional requirements

FR1 Analyse a traffic video
The main goal of traffic survey automation is an automated analysis of a video recording. Such an analysis is a complex process and requires a substantial amount of hardware resources, mainly RAM and GPU.

In this context, running an analysis means applying a selected pre-trained deep learning model to the recording's frames to extract information about vehicles in the scenes.

However, user machines are usually not capable of delivering enough performance to finish this task in a reasonable time.

- FR1a: Run the analysis on a dedicated remote server.
- FR1b: Schedule the tasks using a queue.
- FR1c: Offer multiple deep learning models.

FR2 Choose a video from a library
Because a remote server performs the computations, a user needs to get the video to the server first. The upload itself can take a significant amount of time, depending on the Internet connection quality.

In order to keep our initial implementation of the application simple, we decided to leave this requirement up to the user and let them upload videos to the server on their own, using commands like `scp` or `kubectl cp`.

Once a video is uploaded to the server, the application needs to discover it and allow the user to pick it up for processing.

FR3 Store the analysis persistently
The results of the analysis need to be stored for later repeated usage. The data is stored as a `JSON` file to allow the user easily download and work with the raw data. The file includes an identification of the objects and their tracked paths.

FR4 Identify the regions of interest
The user can define regions of interest. These are lines that determine places where the vehicle counting happens. Each vehicle crossing that line adds one to the total sum.

To allow the user to provide these regions, we show them the paths that the vehicles usually take. This is achieved by drawing smooth and aggregated lines formed from the centres of the bounding boxes.

- FR4a: The application is capable of working with multiple lines.
- FR4b: To avoid confusion, each line can be assigned a name.

- FR4c: More sets of lines can be specified for a single analysis.

FR5 **Present the final data**
Results of the analysis are offered as either *video*, *raw* or *aggregated* data. The user can pivot between these output formats without re-analysing the original video.

Video shows a recording similar to the original one but with lower FPS and coloured bounding boxes of tracked vehicles. The original video is required.

Raw `JSON` data can be downloaded for future custom use.

Aggregated `CSV`-formatted table which contains sums of vehicles that pass the regions of interest. If there are multiple, the table is a matrix that shows data for each two of the lines (similarly to Figure 3.2).

This thesis assignment requires the aggregated traffic statistics to be presented "in time". We find this statement ambiguous so let us clarify how we understand it. The original idea was to present the aggregated counts on a timeline – per a time unit (a couple of minutes, an hour, etc.). On top of that, we would be able to provide the average time the vehicles spend in a camera's view.

After discussing the topic with traffic researchers, we identified these two functionalities as irrelevant for the initial implementation and evaluation. The primary reason for the decision is a lack of long enough videos where the feature would bring any benefit. Moreover, we do not have any reference data that we could compare the output of these two features with.

We refer to section 5.4 and section 7.5 where we provide more information about storing the analytic data and propose the way of implementation of the "in time" feature in future.

4.2.2 Non-functional requirements

NFR1 **Web user-interface**
We intend the app to be useful for a broader audience. Therefore, a friendly and usable user interface is necessary. Alongside the web UI, the app offers a palette of CLI tools that support an additional complex task of exporting an analysis into a video output.

NFR2 **Asynchronous interactions**
All the performed operations are time-consuming, so they are executed in a way that does not block the user. Moreover, the application provides feedback about the status of the performed jobs.

NFR3 **Reasonable processing time**
In section 4.1, we stated that it could take more than three days for a human being to process a video. So, time is not of high priority, yet we believe the analysis of a recording does not exceed two times the length.

The processing time is affected by the available hardware. An expectation of processing a video in almost real-time without proper hardware is unachievable.

NFR4 **Reasonable accuracy**
For the model to be applied in practice, the error rate cannot diverge much from the average values that human-performed counting yields. According to our discussion, an error rate of around 10% is acceptable [2].

Still, the position of a camera and the quality of the recording influence the resulting accuracy prominently.

NFR5 **Hardware requirements**
The model is built with a GPU support to run faster. Single CUDA-supported device is sufficient. The amount of available graphical unit's memory affects the maximum possible batch size. We have tested that 32GB of GPU memory is enough for a batch of 32 frames.

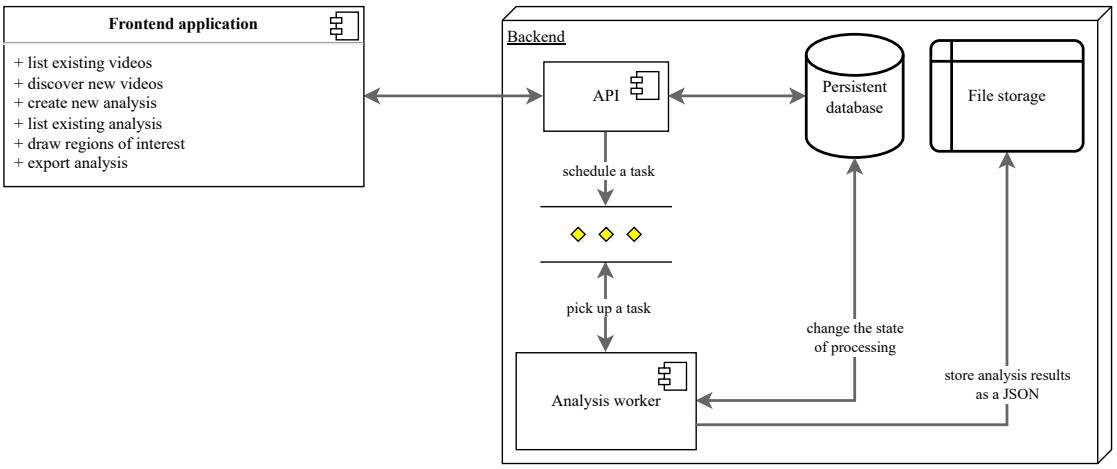


Figure 4.1 Application architecture diagram

4.3 Solution

Based on the problem statement and the requirements defined in the sections above, we propose a software solution in this section.

We split our application into a *backend* and *frontend* part. While the frontend is responsible only for serving information to a client and interacting with them, the backend does all the heavy lifting, including communication with clients, scheduling the analysis jobs, performing counting operations, and storing information in a database. Figure 4.1 shows a simplified architecture and communication of the app's components.

4.3.1 Backend

The backend runs as a standalone application that provides an API for communication with clients. It allows the following operations:

- list existing video source files,
- discover new source files by indexing the source videos folder,
- schedule an analysis with the possibility to choose the combination of models and their initial parameters,
- browse through existing analyses,
- download raw or video version of an analysis,
- see a visualisation of an analysis,
- create a new set of regions of interest,
- to count the vehicles crossing the regions of interest.

In our work we follow the tracking-by-detection paradigm (section 2.5). Detection is performed by the EfficientDet model family, precisely **D5** or **D6** architecture which we identify as a good trade-off for performance and accuracy given the assumption of HD videos.

For tracking, we utilize the Kalman filter-based SORT and Deep-SORT algorithms defined in chapter 3. To improve their abilities, we decided to replace the constant velocity model with a constant acceleration model; refer to subsection 5.3.1 for more details.

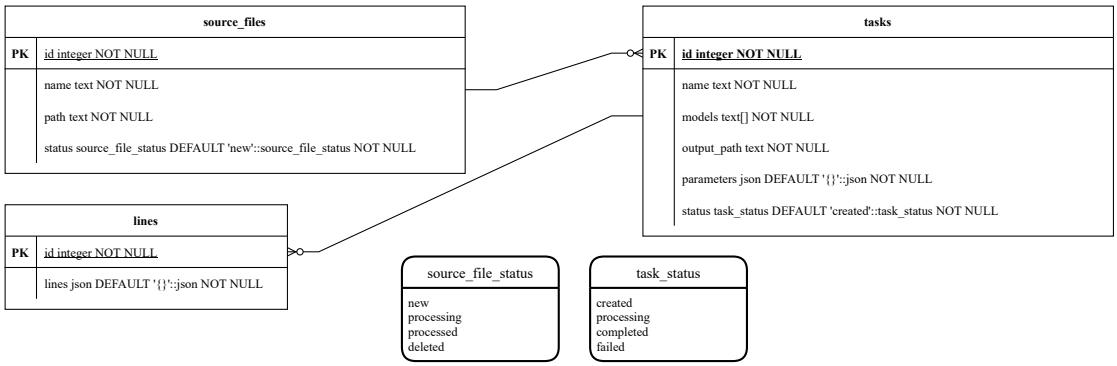


Figure 4.2 Database schema

In the initial version of the application, we pay no attention to the classification of the detected objects.

4.3.1.1 Worker

In order not to overload the app that handles the API, we keep a separate `worker` that accepts scheduled analysis tasks from a queue and runs them one by one.

This allows for more manageable scaling in future. If we ever decide to process more tasks at once, we only need to spin up more workers, but add the same number of new GPUs and map them correctly so that each worker uses its own GPU.

When a task starts, the worker, changes the state of the source video and the task to `processing`. Once finished, it translates the states accordingly to indicate a success or a failure. Yet, the transition does not happen in the case when the operating system suddenly kills the process because of a fatal event.

Having an independent set of workers processing the analyses queue allows the user to override the default configuration variables of an analysis. Namely:

- video properties like output frame rate or maximum number of frames to process,
- detection model parameters like IoU and score threshold,
- tracking model parameters, for example, the minimum number of consequent detections to start tracking,
- visualisation parameters such as a smoothing polynomial function's degree.

4.3.1.2 Database

To comply with the requirements, we have to store data about the source videos, the performed analyses and the drawn regions of interest, together with their states and references to video and `JSON` files.

A *relational* database is a perfect candidate for the job. The decision has several reasons. There are evident connections between the entities; we do not require a high-performance distributed solution, nor do we store a huge amount of unstructured data.

Figure 4.2 shows the schema of our database model, including their required presence and the data type. We point out that `source_file_status` and `task_status` are enumeration data types used in `source_files` and `tasks` tables to ensure consistency of the `status` values.

Backend satisfies the following requirements: FR 1, FR 3, NFR 3, NFR 4 and NFR 5.

4.3.2 Frontend

The frontend is a single page application that runs in a web browser in `JavaScript`. It provides a graphical interface for the user to interact with the application.

It is a small piece of software that lacks many fundamental features expected of a production version, including authentication and permissions management to ensure that others cannot access one user's video and analysis data.

One of the main reasons to provide the GUI is to allow the user to quickly draw lines into a visualisation of an analysis and provide the regions of interest for the final task of counting the vehicles.

The other important action the user does is creating a new analysis task. It is done via a form where the user chooses the detection and tracking models to use, together with a list of parameters and hyper-parameters that impact the models' behaviour. These parameters are task-specific and cannot be quickly transferred to another task.

Frontend satisfies the requirements FR 2, FR 4, FR 5, NFR 1 and NFR 2. Nonetheless, not all of the export formats defined in FR 5 are available in the initial implementation of the frontend application. Some require usage of a CLI command.

Chapter 5

Proposed framework

Now that we have defined all the necessary terminology, discovered how others approach the problem of MOT and highlighted expectations of our solution, we can explain our framework for automated traffic survey analysis.

The pipeline consists of six theoretical steps which map to four implemented functions. This chapter focuses on enumerating and providing details about the theoretical steps. For implementation details, please see chapter 6.

1. read video frames,
2. detect vehicles,
3. track vehicles,
4. store tracklets,
5. visualise tracklets as trajectories,
6. count vehicles based on regions of interest.

In the text below, we refer to a list of constants, parameters and hyper-parameters that are listed and explained in Appendix B. As we previously mentioned, all the parameters are user-customizable.

5.1 Read video frames

We open the requested source video and find its original frame rate $\mathbf{F_i}$. The desired output frame rate $\mathbf{F_o}$ is represented by `VIDEO_FRAME_RATE`. Value f specifies that only each f-th frame is accepted and the rest of them is truncated.

$$f = \begin{cases} \frac{\mathbf{F_i}}{\mathbf{F_o}}, & \text{if } \mathbf{F_o} < \mathbf{F_i} \\ 1, & \text{otherwise} \end{cases}$$

In order to keep the pipeline efficient, we form batches of frames with size `ED_BATCH_SIZE` denoted as b in the following text. The next stages of the pipeline process such batches.

If `VIDEO_MAX_FRAMES` is specified, then only this number of frames are read from the video and passed to the following stages of processing. This parameter effectively cuts the video and is helpful, especially for testing purposes, as it dramatically decreases the processing time. It should not be used in production analyses.

This step produces a structure of shape $b \times \mathrm{W} \times \mathrm{H} \times \mathrm{C}$ where the capital letters represent width, height and channels of the frames.

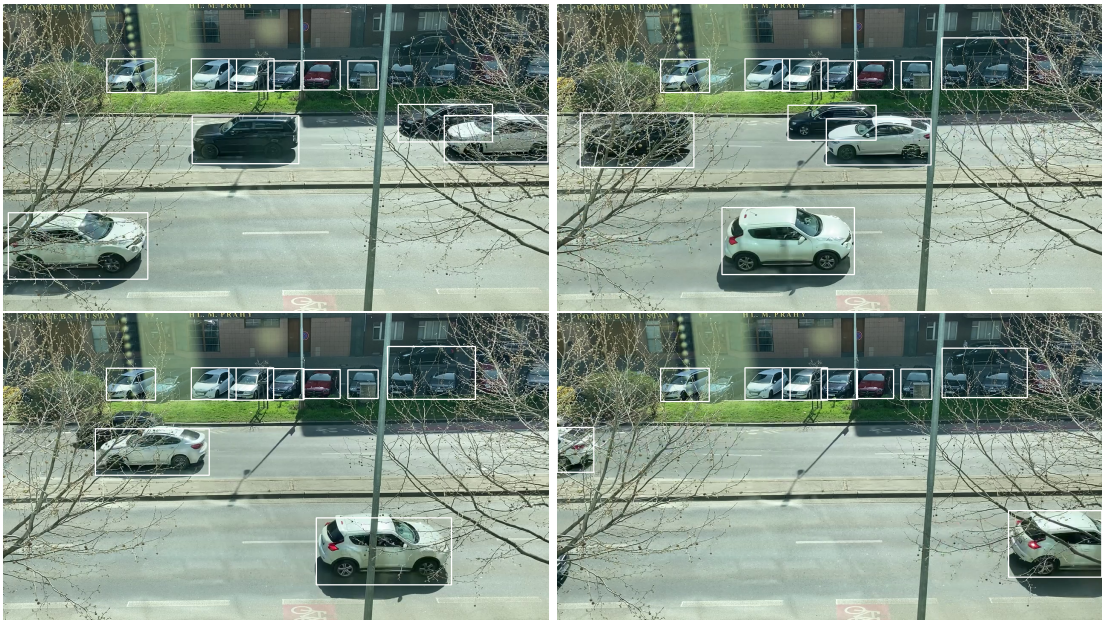


Figure 5.1 Detection on 4 sequential frames

5.2 Detect vehicles

The opening step is to pass the input batch of frames through the model's prediction step. The model is either the **D5** or the **D6** architecture of `EfficientDet` (subsection 2.2.3). Both the models are pre-trained on the `COCO` dataset (section 6.2).

The output of the prediction step is a tuple of bounding boxes, class predictions and scores indicating confidence in the prediction. Let us assume that there are $\mathbf{N} = [N_0, N_1, \ldots, N_{b-1}]$ objects in the frames. Then, the shapes of the outputs are $\mathbf{N} \times 4$, $\mathbf{N} \times 1$ and $\mathbf{N} \times 1$ respectively.

Because the models are trained on the `COCO` dataset, which contains more objects than just vehicles, they also discover people, animals, food, sports equipment, etc. We are not interested in these additional categories. For this reason, the second step of our detection pipeline is to drop the non-vehicle-like objects. We achieve it by assigning a score of 0 to all the detected objects that do not fit into our predefined vehicle classes.

The vehicles recognised by both models are bicycles, cars, motorcycles, aeroplanes, buses, trains, trucks and boats. The numbers of the classes are **2** to **9**. Since we analyse the on-road traffic, only the set of 2, 3, 4, 6, 8 is allowed.

Object detection models often capture the same object in multiple bounding boxes. To get rid of the duplicates we use non-max suppression (NMS) (subsection 2.2.4). The effect of the algorithm is parameterised using several variables like `ED_SCORE_THRESHOLD` and `ED_IOU_THRESHOLD`. It outputs the same triplet of bounding boxes, classes, and scores, yet the number of detections in the frame dramatically decreases.

Finally, to comply with the rest of our framework, we switch the order of bounding boxes' top-left and bottom-right corner coordinates from $[y_1, x_1, y_2, x_2]$ to $[x_1, y_1, x_2, y_2]$.

Figure 5.1 show an example visualisation of all the detections in four sequential frames.

5.3 Track vehicles

The tracking algorithm takes only the batch of bounding boxes as an input parameter. Let us now consider only bounding boxes N_i of i-th frame in the batch.

Figure 5.2 Tracking on 4 sequential frames. Detected vehicles are bordered with white while tracking predictions are black. We can notice a couple of false detections at the top (a bush). Since they are not located on the road they do not cause problems to the vehicle counting.

Each bounding box corresponds to an object, and every object is assigned its own `tracker` that predicts the following states and corrects it later based on associated detection.

First, we get a prediction for each of the existing objects. It returns us an array of m locations that are assigned to at most one of the N_i detections using an association mechanism.

Some of the locations and detections stay without its counterpart. Those are either new detections or locations for objects that temporarily or permanently disappeared from the scene. We set up a new `tracker` for new detections and delete it for old locations – no correction happened in the last `MAX_AGE` iterations.

In the end, the algorithm returns a concatenated list of detections and predictions, and their identifiers. The upper limit for size of both the lists is $N_i + m$, typically $N_i + o$ where $o < m$.

5.3.1 Tracker

A tracker maintains its own Kalman filter (section 2.3) to represent a vehicle motion, represented by movement of a bounding box. In our implementation we assume constant acceleration model:

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2. \tag{5.1}$$

We define a four-dimensional measurement space, ten-dimensional state space and no control space. This follows the suggestion in section 3.2, extends it by the acceleration variables:

$$\mathbf{d} = [c_x, c_y, r, h],$$
$$\mathbf{s} = [c_x, c_y, r, h, \hat{c_x}, \hat{c_y}, \hat{r}, \hat{h}, \tilde{c_x}, \tilde{c_y}],$$

where (c_x, c_y) is a center coordinate, r is a ratio, h is a height of a bounding box, $\hat{c_x}$, $\hat{c_y}$, $\hat{r}$ and $\hat{h}$ are the velocity variables, and $\tilde{c_x}$, $\tilde{c_y}$ are the acceleration variables. This extension improves the model's tracking quality because it accounts for vehicles' speed changes while in view, especially for traffic lights-controlled recordings.

Based on Equation 5.1 and the spaces, we define measurement matrix H, measurement noise covariance R, transition matrix F and process noise covariance Q. As we do not use the control space, control matrix is not required. We follow the equations listed in [44]:

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{pmatrix} \quad R = \begin{pmatrix} \sigma_m^2 & 0 & 0 & 0 \\ 0 & \sigma_m^2 & 0 & 0 \\ 0 & 0 & \sigma_n^2 & 0 \\ 0 & 0 & 0 & \sigma_n^2 \end{pmatrix}$$

$$F = \begin{pmatrix} 1 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Q = \begin{pmatrix} \frac{\Delta t^4}{4} & 0 & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & \frac{\Delta t^4}{4} & 0 & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^4}{4} & 0 & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 \\ \frac{\Delta t^3}{2} & 0 & 0 & 0 & \Delta t^2 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & 0 & 0 & \Delta t^2 & 0 & 0 & 0 & \Delta t \\ 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & 0 & \Delta t^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & 0 & \Delta t^2 & 0 & 0 \\ \frac{\Delta t^2}{2} & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 1 & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 1 \end{pmatrix} \sigma_a^2,$$

where Δt is a difference between two successive measurements and σ_a^2 is variance in acceleration.

5.3.2 Association methods

When SORT is picked as the tracking algorithm, just a simple association using the Hungarian algorithm (section 2.4) on a matrix of IoUs is employed.

Deep-SORT adds a few more layers. First, it generates feature embeddings for the bounding boxes pixels, then uses Mahalanobis distance and cosine distance to form the matrix used during association.

Figure 5.2 shows an example of how the tracking algorithm estimates the movement of the bounding boxes. The tracker's estimated position is drawn in black while the correct detected position is white.

5.4 Store tracklets

Once tracking is complete, we store the tracklets in a `JSON`-formatted file. To support visualisations and vehicle counting, we have to store the list of tracks. Video export is available only when the source video is still present in the source files folder.

```
[
  {
    "identifier": "ef490615-e9d6-4a41-9715-02647addb38a",
    "class": 3,
    "score": 0.4988304376602173,
    "frames": [4, 5, 6, 7, 8, 9, 10, 11],
    "path": [
      [
        996.8204345703125, 396.7250671386719,
        0.9125178456306458, 110.7529296875
      ],
      [
        1000.4480590820312, 394.2456359863281,
        0.920068621635437, 106.69781494140625
      ],
      "..."
    ]
  },
  {
    "identifier": "40161aa5-0efc-46f0-a8ac-698c6690dc77",
    "class": 8,
    "score": 0.06900424510240555,
    "frames": [70, 71, 72, 73, 74, 75, 76, 77],
    "path": [
      [
        497.4085998535156, 564.85986328125,
        2.6691994667053223, 310.2802429199219
      ],
      "..."
    ]
  }
]
```


Code listing 1 A shortened analysis output

The list mentioned above contains dictionaries with an identifying UUID, class, average score, frames the vehicle appears in, the path and bounding box size stored as a ratio and a height. Listing 1 shows a shortened example of the stored data.

We store the numbers of frames the vehicle appears in to support the following use cases our application already supports or is supposed to support in future:

- video-file export,
- “statistics in time”,
- average time vehicles spend in the view or between regions of interest.

5.5 Visualise

Visualisation is vital for the user since they base the selection of regions of interest on it. It provides a perception of how vehicles move in the recording allowing for a feasible choice of the

	Line 1	Line 2	Line 3	Line 4	Unknown
Line 1	-	7	4	2	0
Line 2	7	-	1	0	0
Line 3	5	0	-	3	2
Line 4	0	0	1	-	2

Table 5.1 An example result of counting. The numbers do not match the visualisation in Figure 5.4.

counting lines. Without visualisation, the user could easily draw the lines in places where no vehicles are detected. Alternatively, the curves might be moved to unexpected positions, for instance, on pavements or edges of the roads.

The visualisation process starts with a transformation of the unconnected path points of a tracklet into a single smooth polynomial curve. Actually, we form a *least-squares spline* of `INTERPOLATION_POLYNOMIAL_DEGREE` degree that approximates the path points, at the same time squashes noises.

The difference that smoothing makes is visible in Figure 5.3. We notice that some of the curves are not straight but move up and down, caused by sudden changes in the vehicles' bounding box. This usually happens when another vehicle partially fills the view in front of the original vehicle. The effect of smoothing is more profound at crossroads with a closer camera view.

Next, we filter out insignificant curves that would only produce confusion. It is the ones that are shorter than `VISUALIZATION_MIN_PATH_LENGTH` parameter. This is followed by separating them into `VISUALIZATION_N_CLUSTERS` clusters according to the starting and ending points to provide a visual hint of the general movements.

Lastly, all of the curves are drawn as polynomial lines into the first frame of the video using a colour determined by the cluster it belongs to. The restricted colour range allows for better orientation in the image and understanding of which vehicles share similar behaviour.

5.6 Count

The final step of our framework is to perform counting and form a matrix of movement that specifies how many vehicles passed from one region of interest to another.

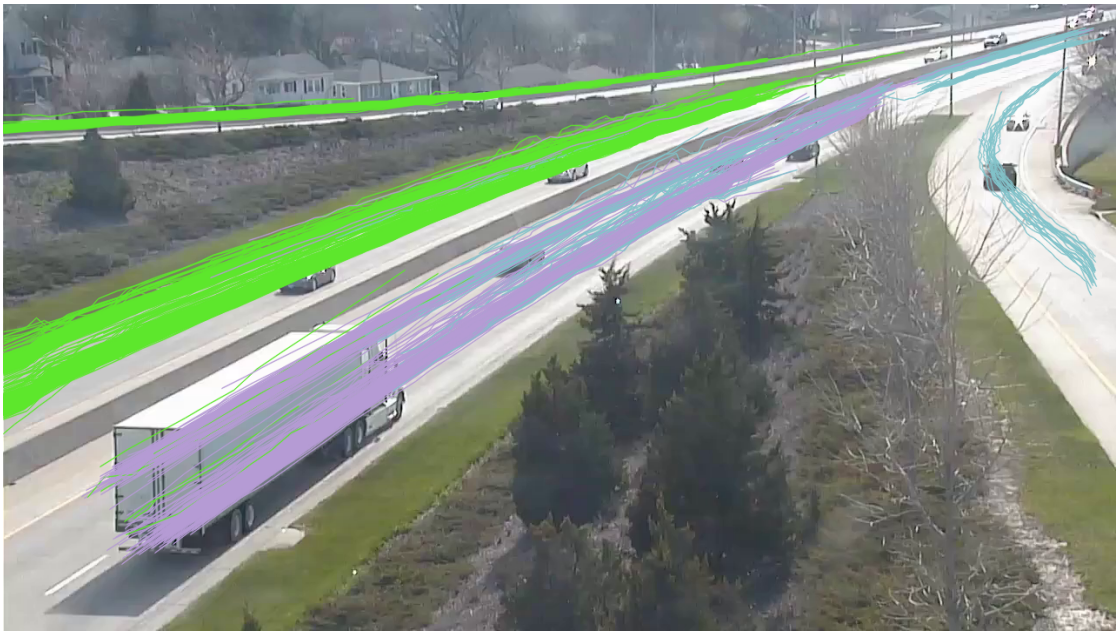
Let us imagine a cross-road, such as the one in Figure 5.4, where we draw four lines on its entries. The result is a matrix of shape 4×5 (Table 5.1). Four lines and four columns for the combinations of crossings and one additional column for those vehicles which source or destination line is unknown.

The prerequisite of the counting step is a set of regions of interest represented by a set of lines that the user draws in the visualisation from the previous step (section 5.5).

We iterate over all the tracks, the result of the analysis, and use their curve representation to find intersections with the regions of interest. If no such crossing exists, the track is skipped. Provided that there is an intersection with precisely one of the lines i, 1 is added in the last column of i-th line's index. It indicates that a vehicle crossed the line i; however, we do not know where it has come from or where it is heading.

In the case of multiple intersections, we add 1 at the proper indices of the matrix, marking the vehicle's way through the lines. To give an example, let us assume that a track has n intersection points $i_1, i_2, \ldots, i_n$ in this order. This means that the vehicle moved from place i_1 to i_2, from i_2 to i_3 and so on until it crossed i_n. We count the vehicle at all the positions $[i_1, i_2], [i_2, i_3], \ldots, [i_{n-1}, i_n]$.

Nevertheless, we do not know the exact order of the intersection points i_j a priori as the order of the regions of interest is not informative at all. On top of that, the *perpendicularly* closest point is not necessarily the first one because of turn-like curves, road obstacles, positions of the



(a) No smoothing applied. We can notice that the curves are “bumpy”.

(b) Smoothed using least squares spline interpolation with cubic polynomials.

Figure 5.3 An analysis visualisation

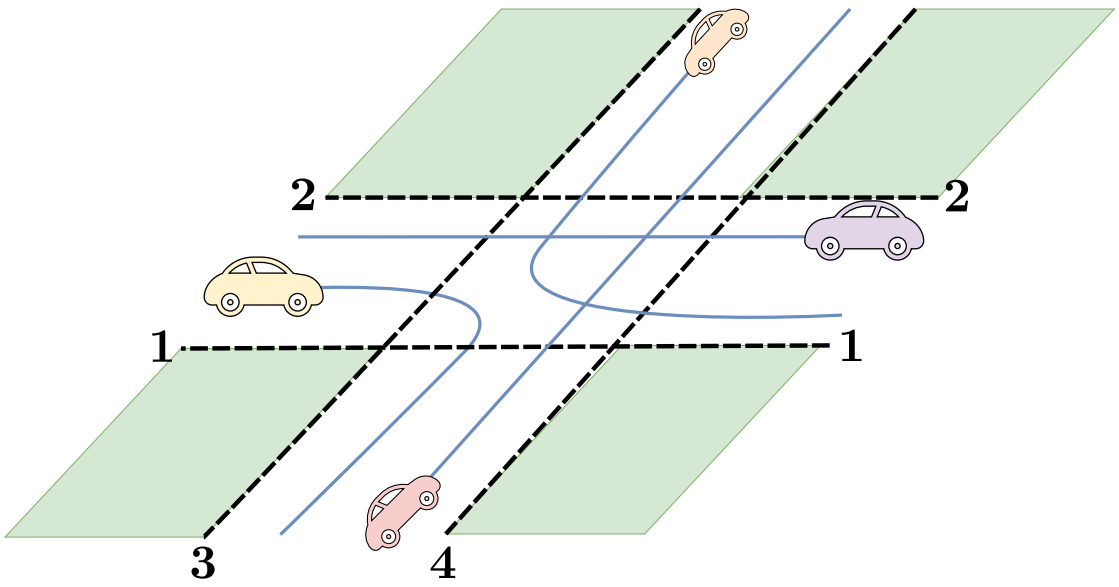


Figure 5.4 Crossroad visualisation

regions of interest or the camera location and its properties. Given a vehicle's curve and all the intersections for the curve, we address the issue the following way:

1. computing the distance along the curve between its start and each of the intersection points,
2. sorting the distances in the ascending order,
3. returning the pairs of indices of the subsequent region of interest.

Chapter 6
Implementation

The following chapter gives details of the technological stack and libraries that we use in our work; it explains how we exported the deep learning models and how data flows inside the framework. Finally, it shows how to build and start the application locally or from Docker images.

As we already discussed in chapter 4 we implement backend and frontend parts of the application to provide a complete user experience and publish it under the `GNU GPL-v3` licence. The source code of our application is publicly available on GitHub, at `https://github.com/opendatalabcz/traffic-surveys-automation/tree/v1.0`

6.1 Packages

Our backend solution is written in `Python`, version **3.9**. One of the advantages of this programming language is the countless number of packages and libraries one can use for their profit. On the other hand, it is slow on its own; therefore, many libraries provide an underlying `C` implementation, intending to make the code run faster.

We also use quite a few such external dependencies throughout the work. This section introduces only a subset of them that we believe are somewhat interesting to write about.

All of the packages are available under a licence that allows commercial use, distribution, modification and private use. The most common licences are *BSD*, *Apache* and *MIT*. Additionally, there are packages with less typical licences as `HPND`, `ISCL` and `MLP 2.0`. These three are requirements-wise the same as those licences mentioned sooner.

Except for the packages installed using a dependency manager, we follow the implementation of SORT [51] and Deep-SORT [52]. We copy and adjust their code to meet our project's needs, standards and structure. Those parts of our code are explicitly marked. The two packages are published under the `GNU GPL-v3`. This licence requires us to publish our work under the same licence and explicitly include the licences of the packages in our code.

`konfetti` is a configuration management library. It helps us with setting up the parameters of models and algorithms. It supports loading configuration variables from the environment, Python objects or `JSON` files. It gives a possibility to override the default variables during runtime easily.

`numpy` offers powerful tools for fast mathematical operations, especially with matrices. We use the package for working with matrices of video frames and bounding boxes, randomly generating colours or computing the IoU.

`OpenCV` provides optimized APIs for computer vision tasks. The package is capable of a lot

more than what we use it for - reading frames from video files as `numpy` arrays, writing arrays `MP4`-formatted videos and processing Kalman filter steps.

`TensorFlow` is a library for machine learning (ML). It features comprehensive tools for building, testing and deploying ML models. It comes with CUDA optimised computations, which means that they are capable of running fast on CUDA GPUs. We chose TensorFlow over `PyTorch`, the other profound ML library, because of the experience we have gathered throughout the years, which would be hard to obtain with `PyTorch`.

`Shapely` is a package of functions and classes that help with representing and making operations with planar objects like lines and polygons. We use it to represent curves and lines, find intersections between them and sort the intersection points.

`simplejson` is an extension of the standard library `json` package. It is capable of dumping and loading more data types than the standard package, for example, `datetime` and `Decimal` objects.

`Scipy` supplies optimized scientific tooling and implementations of various optimization, interpolation or linear algebra algorithms. We use its implementation of linear assignment problem (section 2.4) solver in `scipy.optimize.linear_sum_assignment`.

6.2 Models

Our work requires two types of deep learning models:

- `EfficientDet` **D5** and **D6** for object detection,
- any object classification model like `EfficientNet`, `RestNet` or `VGG16`.

We take advantage of pre-trained models to avoid complicated, resource-demanding and time-consuming training. One of the sources of pre-trained models is `TensorFlowHub` which provides an extensive library not only of object detection models but also language processing, video classification and many others.

Nonetheless, it has a considerable disadvantage – the models do not support batching. Processing the frames one by one is inefficient and slow, taking no advantage of the available hardware optimisations.

Instead, we create our own versions based on Google's `AutoML` [58]. They provide checkpoints of all the `EfficientDet` architectures, trained on `COCO` dataset. The **D6** architecture is a vanilla version. The **D5** is fine-tuned using a data augmentation technique [59] to address corruptions, blur and adversarial changes of images.

The authors give us the possibility to alter the original checkpoints using a command-line utility; see Listing 2 for details. We do this for the following reasons:

- to allow batching,
- to suppress the effect of their NMS which is also a part of the checkpoint and
- to set proper model parameters like expected image size (see Listing 3).

Moreover, we adjust the code in `efficientdet/tf2/postprocess.py`, line 186, to allow customizing the value of IoU threshold which controls dropping of overlapping bounding boxes:

```
iou_thresh = nms_configs['iou_thresh'] or 1.0
```

Our checkpoints of the models are available on CESNET's OwnCloud and on the enclosed medium:

```
python efficientdet/model_inspect.py
    --runmode=saved_model
    --model_name=MODEL_NAME
    --ckpt_path=PATH_TO_ORIGINAL_CHECKPOINT
    --saved_model_dir=PATH_TO_NEW_CHECKPOINT
    --batch_size=0
    --hparams=params.yaml
    --max_boxes_to_draw=512
    --min_score_thresh=0.05
    --nms_method=gaussian
```

Code listing 2 Export an EfficientDet checkpoint

```
image_size: "1280x720"
nms_configs:
  sigma: 0.6
  iou_thresh: 0.97
```

Code listing 3 Contents of `params.yaml`

EfficientDet D5 `https://owncloud.cesnet.cz/index.php/s/IUnmEK9iFol9NaF`,

EfficientDet D6 `https://owncloud.cesnet.cz/index.php/s/KPeGv4rl3cJKLDj`.

They have to be downloaded, unzipped and stored in a specified path where the application can find and read them.

We choose `VGG16` as the object classification model, mainly because of the smaller embedding layer size – the one that precedes the classification layer. While the other models are 1024 values large on the embedding layer, `VGG16` has 512 which saves a significant amount of memory. We use the values on this layer as feature vectors required for Deep-SORT (section 3.2).

The low memory requirement is essential. We store a short history of feature vectors for each detected vehicle. Hence, having a large feature vector can consume a significant amount of memory, even causing the whole pipeline to fail on out-of-memory error.

Yet again, we use the pre-trained version of `VGG16`, this time it is a checkpoint implemented in `tensorflow` and initialized as follows:

```
tf.keras.applications.vgg16.VGG16(
    input_shape=(160, 160, 3),
    include_top=False,
    pooling="max",
    weights="imagenet",
)
```

6.3 Backend

Our backend application consists of two parts. The first one is an API that communicates with clients and schedules video analysis tasks. The other part is a set of asynchronous workers that process video analyses – steps one through four of our proposed framework (chapter 5). The structure of the backend repository folder is described in Appendix A.

6.3.1 API

The API runs as an ASGI server with underlying `FastAPI` framework. In our implementation, we take advantage of `async` support, even though we do not expect high volumes of requests and do not perform many blocking operations.

`FastAPI` has many benefits. It easily integrates with packages that define the API's endpoints, schemas, and various database frameworks. It generates an OpenAPI schema and a Swagger UI automatically from the code.

We choose `PostgreSQL` as the relational database management system for our application. We choose it because of our deep knowledge of the system, the possibility to store `JSON` objects in a column, and define custom enumeration data types.

Our application uses `asyncpg` driver to connect to the database. The driver itself is quite inconvenient to work with, so we employ `SQLAlchemy` and `databases` frameworks to simplify the database layer management and code. We define the tables structure using `SQLAlchemy`'s core DDL tools. The database migrations are generated and applied using `alembic`.

We use `pydantic` package to represent the database objects on the business logic level. The `pydantic` models' definitions come in handy on the API level as well, since they define the API schemas for `FastAPI`.

The frameworks mentioned above form a simple yet powerful stack that handles all the user requests. On top of providing basic CRUD operations, the API handles steps 5 and 6 of the framework.

6.3.2 Worker

Upon a new task creation, it is queued by the API to be processed later by a worker. We use `celery` package, distributed task queue, to cover this functionality. `Celery` workers are standalone processes that pick up tasks from the queue, run the analysis and store results.

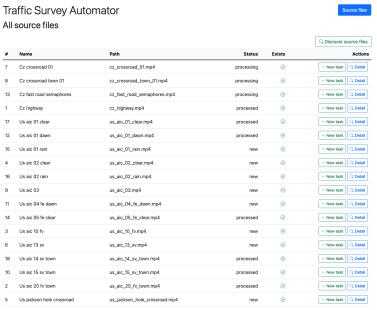
We work with `redis` in-memory key-value database as a Celery *backend broker* that orchestrates the communication between workers. Alternatives to `redis` such as `RabbitMQ` or a relational database exist, yet, we find those sledgehammers to crack a nut. Our choice fits the use case sufficiently.

Videos are large files; loading them into memory at once is infeasible. Therefore, we load the batches on-demand, process and delete them. We achieve this behaviour by implementing each of the steps of the framework as `Python` *generators*. The batches are formed using `tensorflow`'s `Dataset` object as shown in Listing 4.

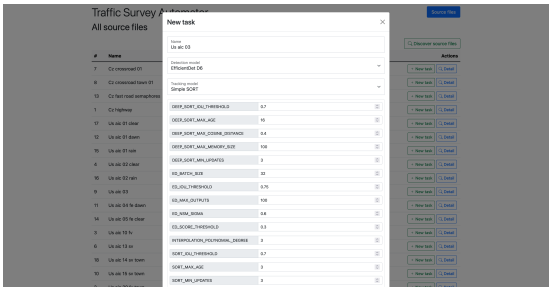
The object detection model iterates over the dataset, yielding each batch of the frames and bounding boxes to be tracked and stored. Object detection proceeds to a new batch only after the

```
def as_tf_dataset(self, batch_size: int) -> tf.data.Dataset:
    dataset = tf.data.Dataset.from_generator(
        lambda: self.frames,
        output_signature=tf.TensorSpec(
            (None, None, 3), dtype=tf.uint8
        ),
    )
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    dataset = dataset.batch(batch_size)
    return dataset
```

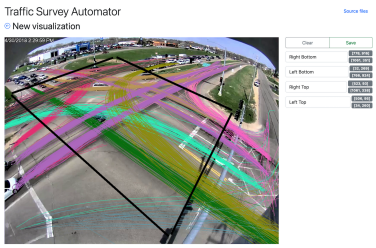
Code listing 4 Video dataset generator



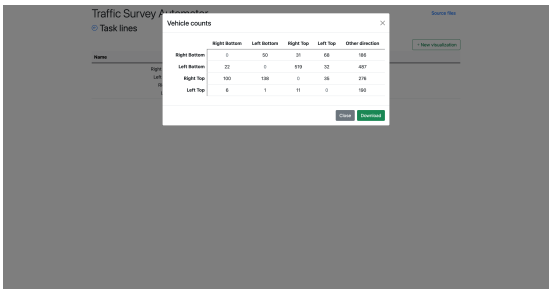
(a) List of the source files



(b) Create a new analysis task



(c) Visualisation and line drawer



(d) Vehicle counts

Figure 6.1 Screenshots of the frontend

current batch is saved. This strategy keeps memory consumption at reasonable levels. Moreover, usage of `tf.data.Dataset`'s prefetching functionality ensures that new data is loaded into the memory sooner, decreasing the time wasted for blocking input-output operations.

6.4 Frontend

The frontend is a simple web application written in `typescript`, on top of the `ReactJS` framework and the `bootstrap` toolkit. It's repository structure is described in Appendix A.

The frontend reads data about source files, analyses and results from the API and displays it to the user in table views.

Except for that, it allows the user to draw regions of interest inside a visualisation, give them names and download a report of counted vehicles. Figure 6.1 shows a few screenshots of the user interface.

The most difficult part of the frontend implementation is a custom component that handles drawing straight lines in the visualisations. It places an HTML `canvas` element on top of the visualization image. Both the elements are scaled properly to fit the window. An `onMouseDown` event is registered, and a black line is drawn according to the coordinates of the mouse.

6.5 Build and run

This section describes how to run the application directly from code or to use the Docker technology. For the purposes of this section, we expect that the reader is familiar with the technology and knows how to pull and run Docker images.

A set of environment variables has to be defined before starting the application. They provide paths and URLs of resources which are inevitable for the application to work.

`CELERY_BROKER` is a data source name (DSN) to the `redis` cache which stores information about scheduled tasks.

`DATABASE_NAME` is the name of the database.

`DATABASE_URL` is a part of the PostgreSQL database DSN. It has the following form:

```
username:password@address:port
```

The connection type prefix `postgresql+asyncpg://` is added automatically on the code level.

`MODELS_PATH` is a path to the object detection models.

`NEPTUNE_API_KEY` is an optional key that connects to `neptune.ai`'s service and enables monitoring of the analysis.

`NEPTUNE_PROJECT` is an optional name of the project where to send the monitoring statistics.

`OUTPUT_FILES_PATH` is a path where to store the output analyses files.

`PYTHON_PATH` variable points at the `backend` dictionary, otherwise some commands might not be properly recognized.

`SOURCE_FILES_PATH` is a path to the source videos. The application discovers the content of this folder.

Once the environment variables have been set, it is possible to run the application locally. The step-by-step user guide is available in Appendix C.

As an alternative to running the application locally described above, we offer pre-built Docker images to be downloaded and used right away, either locally or as server deployments. They are available as a part of our GitHub repository:

backend ghcr.io/opendatalabcz/traffic-surveys-automation-be:latest

frontend ghcr.io/opendatalabcz/traffic-surveys-automation-fe:latest

The backend image is based on `tensorflow/tensorflow:2.7.0-gpu` that contains all the essential packages and libraries to run on CUDA-supported GPUs. The frontend image is based on `node:alpine` that builds the project and `nginx:stable-alpine` that servers the static content to the user.

On top of the images, we provide a `docker-compose` orchestration of both the application images together with a database and `redis` instances. The composed version takes care of all the setup; the user needs:

- to download the object detection models and store them at the `MODELS_PATH` location,
- to create the folders that mount to the containers, namely, source files, output analyses, models and PostgreSQL data,
- to place source videos into the source files folder.

If orchestration is not used, the user has to additionally provide a connection to a `PostgreSQL` database, a `redis` instance, the backend API URL, and also manually run the database migrations `alembic upgrade head` from inside the `backend` folder.

Chapter 7
Experiments

In the following chapter, we present the results of how our proposed framework performs on real video data. To do that, we first need to introduce the video data itself, including its origin and quality. Second, we describe our computational environment where we run the analysis tasks and how we monitor them in the sense of time and resources.

Only then do we compare the results of our framework with human-counted numbers of vehicles and explain issues. Finally, we propose improvements that can be implemented in future to make the framework better and more accurate.

7.1 Data

We have two sources of videos at hand. Because we do not need any of them for training, they are both available during evaluation.

It was unachievable for us to get such data from local authorities or find them online. We believe this has several reasons. First, creating and annotating the datasets is complicated and expensive, so institutions keep them closed. Second, publishing video recordings is problematic due to privacy protection acts like GDPR. Therefore, we used sources available to researchers and asked for help at our home university, CTU, and we got a set of video recordings from roads in the Czech republic. On top of that, we created one video on our own, capturing the traffic at the Rohan Embankment in Prague, making sure that people and licence plates are hardly visible.

The first dataset is a collection of urban intersection and highway recordings provided by the *AI City Challenge* [60]. It consists of approximately 30 short videos filmed in various weather conditions and from different angles, including front-view (Figure 7.1c), side-view (Figure 7.1d) and fish-eye (Figure 7.1a). All the videos are filmed in 960 or 720 height pixels at 10 FPS.

The dataset has the downside of missing the actual counts of vehicles moving in specific directions. To compare, we have to count the number of vehicles ourselves. As these values are not re-validated, the reader must understand that there might be a slight error in those counts. In addition, to decrease the amount of work, we select only a subset of 6 of these videos for our experiments, keeping them as diverse as possible.

The second dataset is a selection of 3 longer videos provided by the Mobile Laboratory of Traffic Analysis at FTS CTU [61]. These videos have a quality of 720p and a higher frame rate than the first dataset.

However, the positioning of the cameras is more challenging (see Figure 7.2) for the framework because it often happens that the vehicles are occluded by each other or by traffic signs leaving the tracking algorithm with less information.

(a) Fish-eye view

(b) Highway

(c) Front camera view

(d) Side camera view

Figure 7.1 AIC dataset

Figure 7.2 Czech roads dataset

This dataset also comes with expected counts of vehicles moving in different directions. We have high confidence in these numbers as they were re-evaluated and validated a couple of times with high precision.

The AI City Challenge dataset is provided to us under the conditions of the licence agreement listed in Appendix D. According to it, we can neither distribute the dataset on the enclosed medium nor publish it on GitHub. However, we can use a minimum portion of the data as a part of our thesis.

It is the same case with the recordings by [61]. The spoken agreement allows only using the videos from their private dataset for evaluation purposes and reporting the results in our work, not distributing them as a part of it.

The video recorded by us in Prague in the Czech Republic is also not disclosed because of the privacy protection rights. The only video included on the enclosed medium is an eighty seconds long recording from Jackson Hole in the USA, available as a live stream on YouTube (`https://www.youtube.com/watch?v=1EiC9bvVGnk`). We use frames from both the videos in the thesis.

7.2 Environment

We run our experiments on a cluster dedicated for high-performance computing (HPC) called `RCI` [62], located at CTU. It provides resources for processing complex algorithms, AI and ML tasks. It features Intel and AMD processor nodes with hundreds of gigabytes of RAM. The cluster is also equipped with NVIDIA Tesla V100 and A100 GPUs that come with up to 40GB of graphical memory.

It is usually cumbersome to get a project with lots of dependencies up and running on such a machine. Conveniently, `RCI` supports running the computations inside containers. This functionality is provided by `singularity` – a tool that brings virtualisation to scientific computing. Moreover, it works well with Docker images. Not only does it transform Docker images into its own `sif` format by running a

```
singularity pull -F <docker-image>
```

command, it also feature a similar CLI for intuitive usage.

`RCI` comes with a workload management system. It means that we are not allowed to run long tasks interactively. We rather schedule them for execution using `sbatch` command and wait for the system to execute it, depending on various prioritisation factors.

To run our analysis tasks in this non-interactive way, the `celery` worker that is running on `RCI` is not executing the analysis directly. Instead, it generates and stores a configuration file on disk and executes the `sbatch` command, passing it a reference to the configuration file. An example of such a file is provided in Listing 5.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --partition=gpu
#SBATCH --gpus=1
#SBATCH --job-name=tsa
#SBATCH --error=logs/tsa.err
#SBATCH --output=logs/tsa.out

singularity exec --bind ~/models:/app/models --nv
--env MODELS_PATH=/app/models,
      DATABASE_NAME='tsa',
      DATABASE_URL='localhost'
traffic-surveys-automation_master.sif python /app/cli/analyse.py
    -f input_video/video.mp4
    -o output_analysis/analysis.json
    -d efficientdet_d5_adv_prop
    -t deep_sort
    -i 8 24
    -c task_configs/config.json
```

Code listing 5 `sbatch` configuration file

7.3 Monitoring

We use an online platform *neptune.ai* to monitor the performance of our framework. The service offers a `Python` client called `neptune-client` that is effortlessly usable in storing ML-related metadata, particularly when training models. As our framework does not realise the training activity, we utilise `neptune` to observe the following metrics:

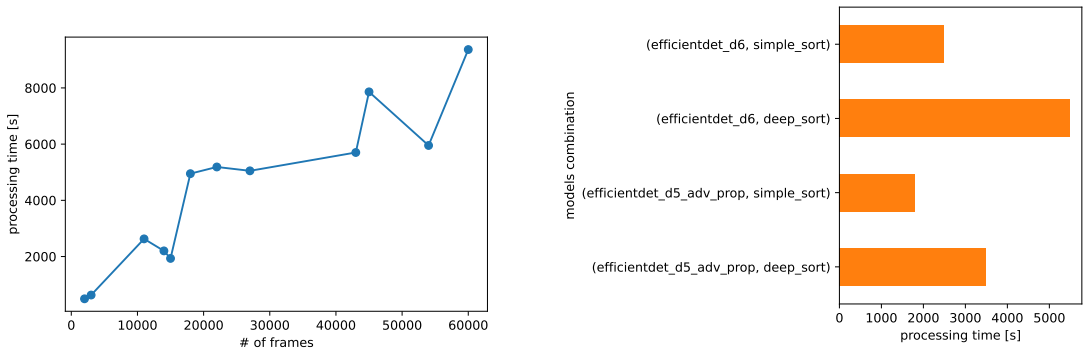
- total runtime,
- hardware resources usage,
- count of detections,
- number of processed frames and
- time spent on detection and tracking part of the framework.

All of our experiments and their metrics are available online at `https://app.neptune.ai/ondrejpudis/traffic-survey-automation/experiments` and as a `CSV` file on the enclosed medium.

Figure 7.3a shows that longer videos, quite intuitively, take longer time to process. We can further identify some exceptions in the graph. Those are related to videos with sparse detections, especially for videos recorded during non-peak hours.

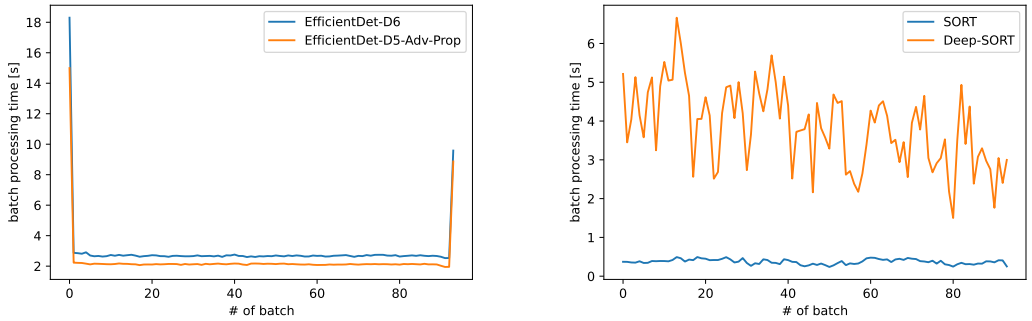
Figure 7.3b confirms the expectations of time complexity of our implemented object detection models. `EfficientDet` **D6** is more expensive than **D5** as it is a bigger architecture, similarly, Deep-SORT requires more time than SORT as it computes the feature embeddings of detected objects.

Figure 7.4 compares the average time spent on evaluating detection and tracking models on multiple 5-minute long videos. Batch size is configurable by `ED_BATCH_SIZE` parameter. It is set



(a) Video length **(b)** Models combinations

Figure 7.3 Average framework runtime



(a) Detection models **(b)** Tracking models

Figure 7.4 Average batch processing time

to 32 for all of our experiments since bigger batches (64 and 48) tended to cause OOM issues on the GPU.

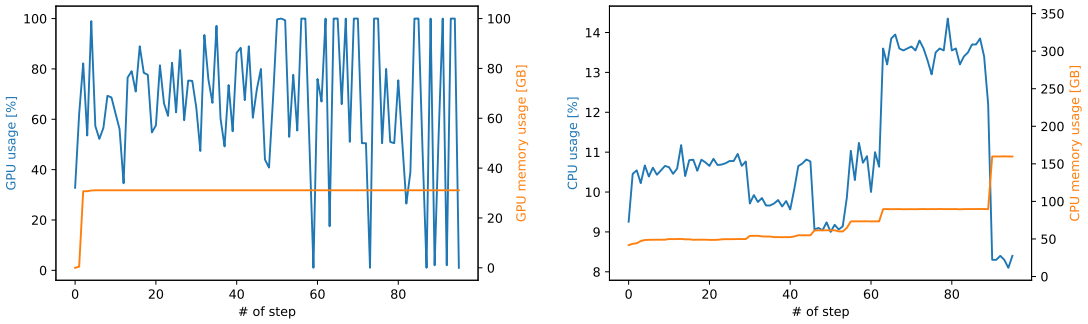
We notice that both the detection models take a significant amount of time during the first iteration and decrease afterwards. That happens because `tensorflow` loads the model from a stored checkpoint. After the initialisation, it keeps a stable 2-3 seconds per batch performance. If we keep video frame rate below 16 FPS then the framework is processing nearly real-time as a batch of 32 frames which equals to 2 seconds of video are processed in 2 seconds.

SORT needs only a few milliseconds to process a batch of frames. On the other hand, Deep-SORT takes 3-5 seconds to compute the feature embeddings, Mahalanobis and cosine distances. The time varies based on the number of detections delivered from the detection model.

We compare the hardware resources usage (Figure 7.5) on a subset of our experiments that are 3000 frames long.

It is evident that the framework takes considerable advantage of GPU. It is utilised heavily through the runtime. Real GPU memory usage is not available because `tensorflow` allocates all of it and manages its contents on its own.

On the contrary, CPU utilisation stays low. The memory consumption grows across time, especially Deep-SORT tasks that require more memory to store the feature vectors' history. We keep it in between 20 to 30 vectors per detection in our experiments to avoid frequent task failures because of OOM.



(a) GPU resources **(b)** CPU resources

Figure 7.5 Average hardware resources usage. `Neptune` snapshots the state of processing units utilization and memory consumption every 10 seconds.

7.4 Evaluation

We evaluate the accuracy of our framework on eight datasets. Three of them come with calibrated and validated numbers; we manually screen another five videos. Both have the counts split by direction.

7.4.1 Accuracy metric

Our goal is to understand how accurate the framework is in estimating the total number of vehicles passing through the regions of interest. We base our metric on multi object tracking accuracy (MOTA) [63]. Since, we have only aggregated data for the whole video, we simplify the accuracy equation as follows:

$$\mathrm{ACC} = 1 - \frac{|y - \hat{y}|}{y}, \tag{7.1}$$

where y is the ground truth value and $\hat{y}$ is the output of our framework. This equation accounts for both missed and false tracked values, always keeping the accuracy ≤ 1. Nonetheless, there is a catch. The value can be smaller than 0 when the model produces a lot of false positives, mathematically speaking, when $\hat{y} > 2y$.

We calculate the accuracy score for each direction separately and then average it to get the total score of the whole video. The goal is to keep the accuracy $\geq 90\%$ (NFR 4).

7.4.2 Visualisations

First, we present visualisations of some of our datasets in Figure 7.6. Only vehicle curves longer than `VISUALISATION_MIN_PATH_LENGTH` appear in the frame.

We identify that the quality of the visualisation and the resulting counting is highly dependent on the model's configuration. The correct calibration of the parameters requires a deep understanding of the parameters, the input video and the conditions it was captured in.

For instance, a recording of a highway, rated at 12 FPS, where vehicles move fast, requires lower values of tracking algorithm's `IOU_THRESHOLD` to accommodate the fast-moving bounding boxes. Contrariwise, vehicles move slowly in intersections and often overlap one another. A low `IOU_THRESHOLD` value would cause the vehicles bounding boxes to absorb each other.

The NMS score threshold is debatable as well. Decreased value of around 0.1 might be necessary for detecting faraway vehicles or during bad weather conditions. Nevertheless, this can

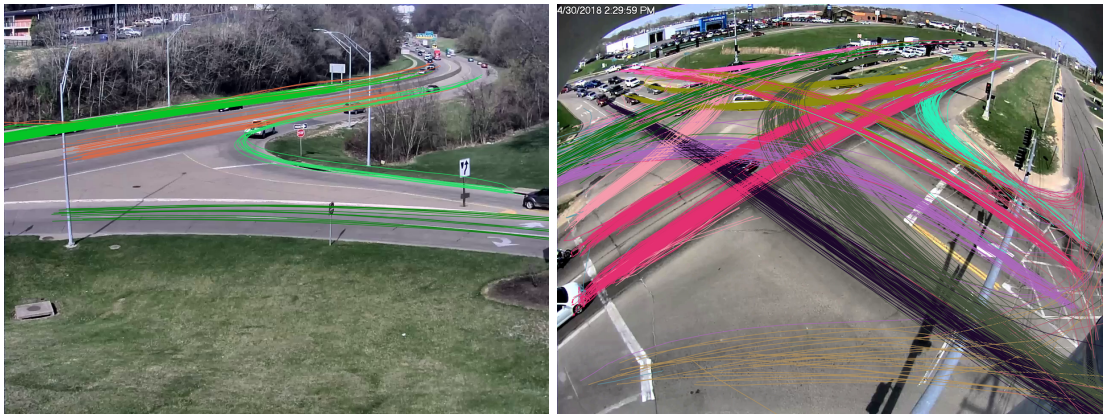


Figure 7.6 Vehicle curves visualizations

Figure 7.7 Vehicle segmentation in videos, each of the identified vehicles is bounded in a coloured box.

potentially lead to many false positives. Therefore, we run the framework over our evaluation dataset multiple times, combining the models with different sets of parameters.

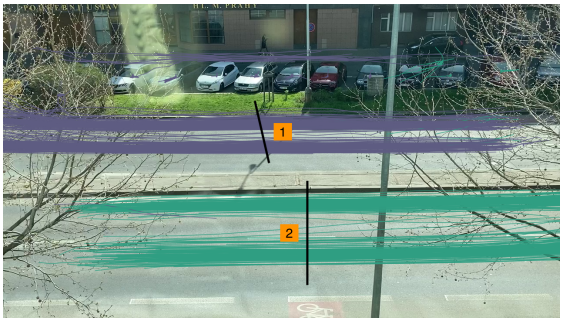
Once the analysis is completed, we inspect the visualisations to determine where the vehicles were mainly recognised. Seeing the curves of vehicles' movements prevents the cases when the user draws the region-of-interest lines outside of our framework's recognition area.

According to our experience, it is better to draw the lines approximately in the middle of the road tracks. On the other hand, it is better to draw the regions of interest closer to the camera on parallel, low-above-the-surface views because the model could have trouble recognising distant objects, especially if a high score threshold is set.

In our experiments, if we have multiple analyses on the same video, we share the regions of interest to achieve comparability across the experiments. Moving the lines and drawing them at different angles can result in different final counts.

7.4.3 Results

Tables 7.1 through 7.8 present our measured vehicle counts (*estimated*), compare them to the *real* values and report *accuracy* as defined in Equation 7.1. Moreover, we provide a labeled visualization of the regions of interest in each of the visualizations so that the reader is aware of the locations. Each table contains a reference to the `ID` of the experiment which yielded the results.

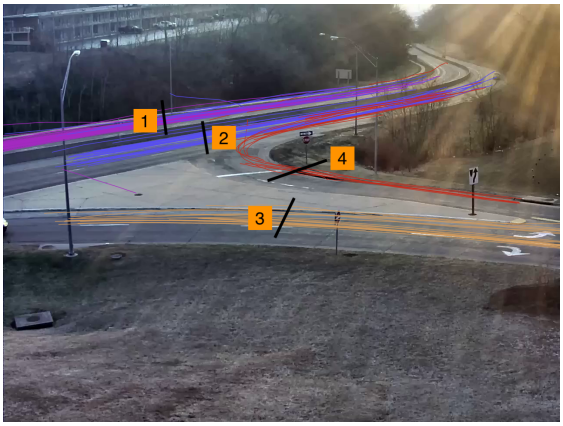


	1	2
real	244	257
estimated	241	258
accuracy	0.988	0.996

Table 7.1 The road at Rohan Embankement features two tracks in both directions. The section is semaphored so the traffic is sudden and traffic jams are expected. Average accuracy is 0.992 (experiment #26).

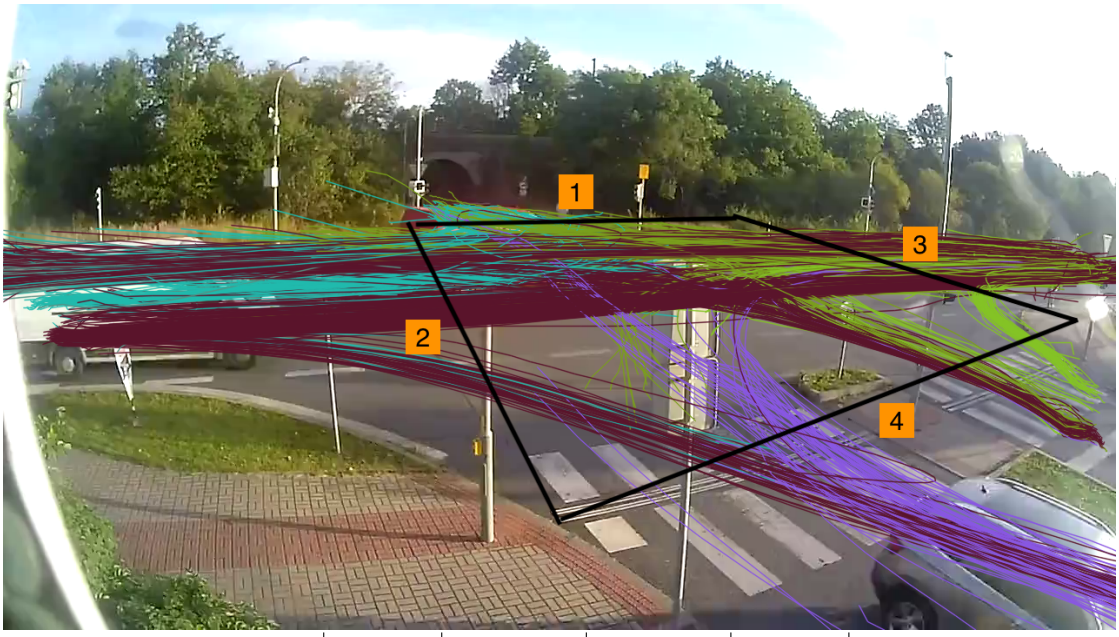
	1	2
real	476	630
estimated	417	635
accuracy	0.876	0.992

Table 7.2 The Czech highway with two narrowed tracks in both directions. The video quality is decreased due to low bit-rate. Moreover, because of the camera position, objects are often occluded. Despite the issues, average accuracy is 0.934 (experiment #37). Yet, it requires a closer inspection of the visualization, since the tracks often end sooner because of loosing sight of the vehicles.



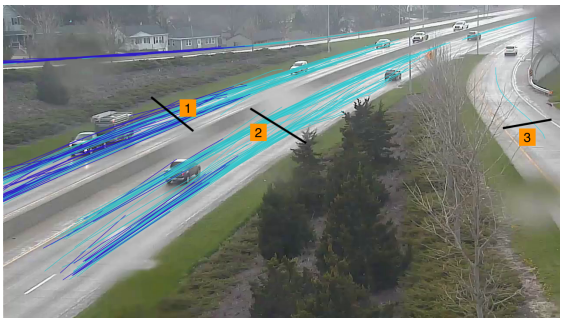
	1	2	3	4
real	74	63	20	7
estimated	68	44	18	7
accuracy	0.92	0.7	0.9	1.0

Table 7.3 A US highway features two main tracks in both directions, the exit and the entry ramp. Recorded at dawn with altered brightness conditions and decreased amount of vehicles on-road. Average accuracy is 0.879 (experiment #11).



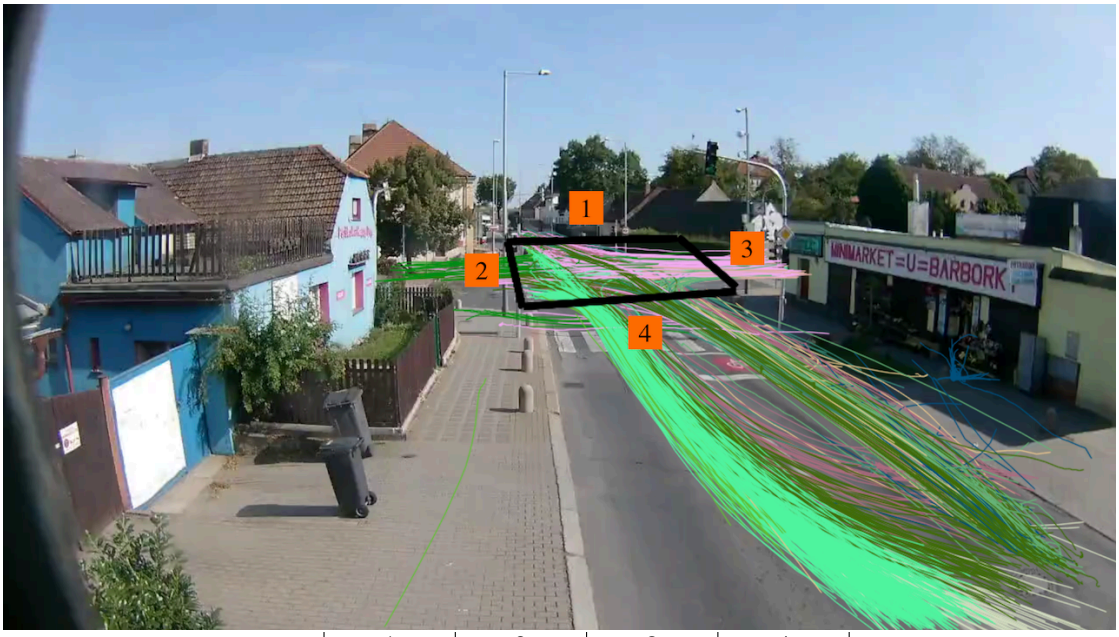
		1		2		3		4		
		y	$\hat{y}$	y	$\hat{y}$	y	$\hat{y}$	y	$\hat{y}$	unknown
1	value	-	-	114	89	35	21	19	9	128
	accuracy	-		0.78		0.6		0.474		
2	value	75	47	-	-	298	294	43	44	270
	accuracy	0.627		-		0.987		0.977		
3	value	27	36	452	344	-	-	43	17	400
	accuracy	0.667		0.761		-		0.4		
4	value	21	18	84	40	51	45	-	-	116
	accuracy	0.857		0.476		0.882		-		

Table 7.4 A complicated, four-directional, semaphored crossroad. The table shows movement from one of the regions to the other, real values as y, estimated as $\hat{y}$. Vehicles which source or destination cannot be determined is denoted as `unknown`. Overall accuracy, while ignoring the unknowns, is roughly 0.7 (experiment #41). We identify a higher amount of occlusion and traffic signs blocking the view as the main issues. We can notice that the lines closer to the camera achieve better accuracies.



	1	2	3
real	71	79	14
estimated	63	64	7
accuracy	0.887	0.81	0.5

Table 7.5 A US highway with two main tracks in both directions and the entry ramp. The side-road at the top is ignored. The average accuracy is 0.732 (experiment #36). The recognition ability is considerably affected by the rain which distorts the video, especially on the entry ramp, causing less accurate results.

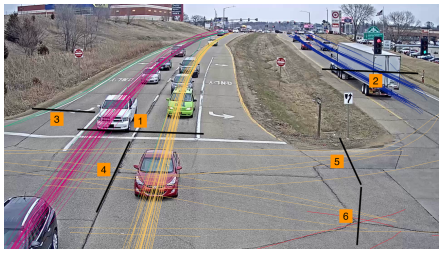


		1		2		3		4		
		y	$\hat{y}$	y	$\hat{y}$	y	$\hat{y}$	y	$\hat{y}$	unknown
1	value	-	-	109	5	142	20	394	76	396
	accuracy	-		0.046		0.141		0.192		
2	value	45	5	-	-	153	9	70	4	184
	accuracy	0.111		-		0.059		0.057		
3	value	54	11	181	30	-	-	61	21	355
	accuracy	0.204		0.166		-		0.344		
4	value	278	85	124	32	79	56	-	-	728
	accuracy	0.306		0.258		0.709		-		

Table 7.6 A four-directional crossroad at a distance from the camera. The table notation is the same as for Table 7.4. Average accuracy with unknowns ignored scores 0.2 (experiment #40) which we believe is mostly caused by the location of the crossroad too far from the camera, making it hard for the model to recognise and follow the objects without track interruptions.

	1	2
real	30	12
estimated	30	12
accuracy	1.0	1.0

Table 7.7 A side view of an in-town road. The position of the camera is fine as it provides a clear with of both tracks without notable overlaps of vehicles. This results in an average 1.0 accuracy (experiment #3).



	1	2	3	4	5	6
real	64	49	3	8	3	5
estimated	64	48	3	5	3	2
accuracy	1.0	0.98	1.0	0.625	1.0	0.4

Table 7.8 A large semaphored crossroad in the USA contains two tracks down and up, vehicles are also turning left, right or going straight. The position of the camera hides a part of the crossing. Average accuracy is 0.834 (experiment #31) which is mostly affected by inability to track the vehicles turning left (crossing the position **6**).

Based on the results of our evaluation, we conclude that our framework works well as far as the source video meets certain conditions:

- the view is not distorted by weather conditions like rain, snow or fog,
- no static objects are blocking the vehicle view for a longer time,
- the position of the camera does not cause vehicles to hide behind one another and
- the quality of the recording is satisfying.

Yet, we did not succeed in providing a tool general enough that would work fine under circumstances such as:

- the regions of interest are far away from the camera (Table 7.6),
- vehicles occlude and hide behind other objects (Table 7.4),
- poor recording quality due to weather conditions (Table 7.5).

Our solution meets or even exceeds the expectations of NFR 4 when the positioning of the camera gives a good view of the road and the weather does not affect the camera's properties. However, there are cases where it performs worse. As we found out during the evaluation, achieving satisfying results in general conditions is a difficult task, therefore, the positioning of the camera should be chosen carefully as it can make significant difference.

7.5 Future improvements

The above-presented results show that the framework's accuracy can improve, particularly when cameras are placed low above the roads or with multiple occlusions and identity cuts. Additionally, we suggest a couple of user-interface changes that we identified to make the application more usable.

7.5.1 Framework

In section 3.3 or the related work chapter, we presented a model called FairMOT. It covers both detection and tracking end-to-end by generating not only the proposed bounding boxes, classes and scores but also feature vectors of the objects, allowing for more precise re-identification.

Implementing this, or a similar, more sophisticated model, and comparing its performance to the baseline that we have set in our work would be one of the possible future steps and a possible improvement of tracking.

Transformers were recently found to be suitable for computer vision use-cases. Liu et al. [64] propose the `Swin` transformer architecture to solve the object detection task without utilizing traditional CNN-based approaches. While writing this thesis, the Swin transformer ranks high in object detection challenges. Nonetheless, the transformer only addresses the detection part of the framework.

From the work of Bewley et al. [51] we know that the accuracy of the detection model is vital for the whole framework to work correctly. Even though the object detection checkpoints and the embedding model are well pre-trained on the `COCO` and the `ImageNet` datasets, we can still *fine-tune* all of them on a vehicles-only dataset. It can bring more attention to these objects in the models.

When exporting some of the evaluation recordings to videos with tracking in them, we noticed that even Deep-SORT was not able to recover the identities of vehicles after hiding behind static objects or other vehicles. This resulted in tracklets ending and re-starting again a few meters away with a different identity.

Consequently, we have seen many of the vehicles counted in the `unknown` column and possibly duplicated a couple of times. Results presented in both Table 7.4 and Table 7.6 suffer from this issue. To mitigate the problem, we have two options:

1. to further improve the existing tracking algorithms so they are not seriously affected when the bounding boxes disappear behind an object through a few sequential frames,
2. to introduce an additional step in the tracking algorithm that joins originally separated curves that end and restart somewhere "not close" to the corners of the frame, and they share the common direction.

7.5.2 Statistics in time

The assignment of the thesis required us to provide the statistics about traffic "in time". As we discussed in FR 5, it was identified as unnecessary for the initial version due to a lack of long enough videos and rather a desire for aggregated values.

Enabling the feature takes altering the final counting steps of the framework. It just needs to consider the already stored information about the frames in which a vehicle appears.

1. Turn the frame numbers into timestamps. Knowledge of the processing frame rate is needed.
2. Group the timestamps to buckets. Each bucket can be as small as one minute, large as an hour or a day, depending on the length of the video itself.
3. Count only vehicles which appear in the scene in a timestamp that belong to a bucket.

7.5.3 Usability

Analysing a more extended sequence takes an extensive amount of time which can become impractical in the long term. Slicing the recording into several pieces and processing them in parallel would speed up processing. Nevertheless, it would require additional hardware resources.

The first version of the user interface is simple but, in certain situations, hard to work with. The actions that we identified as problematic are the ones that required the user to repeat the same actions multiple times:

- creating new analyses tasks with the same parameters for different videos,
- using the same regions of interest in multiple visualisations.

These actions are currently not available; the user has to draw new regions and change the parameters every time manually. Applying the changes would save a lot of time and frustration.

Chapter 8
Conclusion

The goal of our work was to create an open-source application that automates traffic survey evaluation. The software achieves the target by analysing videos from stationary cameras and counting the number of vehicles crossing handpicked regions of interest.

In the theoretical part of our work, we provided an insight into the problem of object detection as a computer vision task and how deep learning models can fulfil it, presenting common architectures in this area. Next, we discussed the multi-object tracking problem and introduced algorithms that tackle it.

Based on the related work, we proposed a framework that is a set of steps that transforms a video recording into vehicle counts that pass the proposed regions, represented as lines that the vehicles traverse.

Our decisions regarding the implementation and use-cases were driven by discussions with researchers at the Mobile Laboratory of Traffic Analyses at the FTS CTU. We identified that the most valuable output is an automatically generated matrix of movements between the regions of interest.

The analysis step captures two pieces of information for each vehicle: its trajectory across the scene and the number of frames it stays in the video. This enables computation of the period vehicles spend in the view and the "in time" statistics. Nonetheless, this information was not provided to the user because it had been identified as unnecessary for the specialists in the area of traffic analyses. Furthermore, we did not have any ground truth data to compare the accuracy of these two outputs with. Therefore, we rather provided the output as absolute counts per the whole video sequence.

In the practical part, we implemented our framework as a `Python` backend service that exposes a set of API endpoints to communicate with the framework. Moreover, we provided a frontend in `ReactJS` to create tasks, draw the regions of interest and see the results of counting.

Our solution also provided a set of tools for customising the analysis parameters, choosing the deep learning models, and further working with the final analysis. The application was designed in a way that it is easily extensible by other detection and tracking models. It took advantage of `tensorflow`'s APIs to run detection and embedding-related operations on GPU. The application's performance was observed through `neptune.ai`, a machine learning meta-data registry. We provided the application's backend and frontend as separate Docker containers that can run independently of each other and are built automatically during the run of a continuous integration pipeline. On top of that, we created a `docker-compose.yaml` orchestration file which starts the application just with a single command.

Finally, we ran a couple of experiments on our evaluation data and compared the estimated vehicle counts to human-annotated data that come from two sources – one is a precisely counted dataset provided by the Mobile Laboratory of Traffic Analyses, second is the AI City Challenge

videos screened by us. The results show that our framework works well when a camera has a clear and close view of the road, without strong occlusion, distortion or objects blocking the view. In such cases, the framework achieved a very good accuracy of 80% and more. It does not perform that well on the challenging datasets, leaving the numbers just a rough estimate of the actual traffic. From our experience, the difficult recordings are complicated for humans to count correctly. The sight is unclear, and it is easy to miss the vehicles, especially when they are small.

Despite the issues with the more challenging datasets, we believe that we managed to build a helpful tool and fulfilled the requirements and the assignment.

Appendix A

Repository structure

Backend

```
backend/
├── cli/ ..... command-line utilities
├── migrations/ ..... database migrations
├── test/ ..... a few unit and integration tests
├── tsa/ ..... the backend application code
│   ├── app/ ..... the API implementation
│   ├── config/ ..... the default configuration parameters and konfetti setup
│   ├── cv2/ ..... wrappers of OpenCV tools
│   ├── dataclasses/ ..... class representation of objects used in the framework's processes
│   ├── models/ ..... detection and tracking models
│   ├── processes/ ..... the framework's steps implementation
│   └── storage/ ..... methods for saving the analysis data
├── alembic.ini ..... database migration management tool's configuration
├── poetry.lock ..... locked versions of installed dependencies
└── pyproject.toml ..... project definition, dependencies and CLI utilities
```

Frontend

```
frontend/
├── public/ ..... static content served to the user
├── src/ ..... source code of the application
│   ├── api/ ..... HTTP clients that connect to the backend endpoints
│   └── components/ ..... a set of functions that compose the application's content
├── .env.example ..... an example version of runtime environment variables
├── package.json ..... project definition and dependencies
├── package-lock.json ..... locked versions of installed dependencies
└── tsconfig.json ..... typescript compiler configuration
```

Appendix B

Parameters

This appendix section contains a list of parameters and hyper-parameters used in our code. Each of them contains a prefix which specifies to which module they belong. First, let us list the prefixes, then the parameters themselves.

ED `EfficientDet`,

DEEP_SORT Deep-SORT,

SORT SORT,

VIDEO parameters related to the video input or output properties,

VISUALIZATION parameters that specify how an analysis visualization looks like.

`ED_BATCH_SIZE` size of a single batch of frames to process (32).

`ED_IOU_THRESHOLD` minimum IoU to consider two bounding boxes the same in NMS (0.75).

`ED_MAX_OUTPUTS` maximum number of detections that NMS outputs (100).

`ED_NSM_SIGMA` softening parameter that decays score (0.6).

`ED_SCORE_THRESHOLD` minimum score of a detection to accept it (0.3).

`DEEP_SORT_IOU_THRESHOLD` minimum IoU to consider two bounding boxes to belong to the same object (0.7).

`DEEP_SORT_MAX_AGE` maximum number of predictions without update, after exceeding this number, the tracker is deleted (16).

`DEEP_SORT_MAX_COSINE_DISTANCE` maximum cosine distance to still consider two feature vectors to belong to the same object (0.4).

`DEEP_SORT_MAX_MEMORY_SIZE` maximum number of feature vectors to remember for a single tracker (100).

`DEEP_SORT_MIN_UPDATES` minimum consecutive updates to consider a tracker active (3).

`INTERPOLATION_POLYNOMIAL_DEGREE` degree of an interpolating polynomial when smoothing the tracklets for visualization (3).

`SORT_IOU_THRESHOLD` see `DEEP_SORT_IOU_THRESHOLD` (0.7).

`SORT_MAX_AGE` see `DEEP_SORT_MAX_AGE` (3).

`SORT_MIN_UPDATES` see `DEEP_SORT_MIN_UPDATES` (3).

`VIDEO_FRAME_RATE` FPS to use during analysis and in output video (15).

`VIDEO_MAX_FRAMES` maximum number of frames to process, this effectively cuts the video, default is None, processing the whole video.

`VIDEO_SHOW_CLASS` show class and score in video output.

`VISUALIZATION_MIN_PATH_LENGTH` minimum path length of a tracklet to consider it a valid one (150.0).

`VISUALIZATION_N_CLUSTERS` number of clusters to use when visualizing (12).

Appendix C

User guide

This appendix chapter lists the user's steps to run our application on their local or server machine.

Running the backend application locally requires either `Python 3.8` or `Python 3.9` installed on the machine alongside the `poetry` dependency manager.

To run the frontend locally, `node` and `npm` have to be installed. We tested that all works fine on `node`'s version **17.5.0** and `npm`'s version **8.4.1**.

Backend

1. Run

   ```
   poetry install
   ```

   inside of the `backend` folder, let the dependency manager create a new virtual environment and install all the necessary dependencies.

2. Download, unzip and place the object detection models in a dedicated directory, section 6.2 lists the location of these files.

3. Set the necessary environment variables mentioned in section 6.5.

4. Optionally, set up the `neptune.ai` monitoring configuration by providing a project's name and API key.

5. Activate the virtual environment through

   ```
   poetry shell
   ```

6. Apply migrations if it has not been done already.

   ```
   alembic upgrade head
   ```

Now, it is possible to use the CLI tools from the `backend/cli` package, run the API through

```
uvicorn --port 8000 tsa.app.app:fast_app
```

or run the `celery` worker

```
celery -A tsa.app.celery.celery_app worker
```

Alternatively, we provide a set of `poe-the-poet` commands:

poe black apply formatting to the code,

poe sort sort the imports,

poe test run the few unit and integration test that we have created,

poe run starts serving the application at the default port,

poe worker starts a single `celery` worker.

We believe three additional commands for performing formatting and code quality checks are not essential to list here.

Frontend

1. Create a copy of `.env.example` and name it `.env`. Set the `API_URL` to the address where the backend is located.
2. Run

   ```
   npm install
   ```

3. Run

   ```
   npm start
   ```

   to run the frontend app on port 4000.

Appendix D

AI City Challenge Data Licence

NVIDIA is making a data set ("Data") available to participants of the Challenge. The Data include:

1. Urban Intersection and Highway Data – Close to 12 hours of traffic videos synchronously captured from multiple vantage points at various urban intersections and along highways. Videos are 960p or better and most have been captured at 10 frames per second.
2. Iowa State University Data – More than 62 hours of video data captured on highways in Iowa.
3. Metadata about the collected videos including GPS locations of cameras, natural language descriptions for vehicle tracks, camera calibration information and other derived data from videos.
4. VehicleX: A synthetic vehicle dataset (third party data made available at `https://github.com/yorkeyao/VehicleX` and may be subject to additional license)

Access to and use of the Data are limited to participants and conditioned upon acceptance of the following terms:

- During the duration of the Challenge, the Data may be used solely for Challenge-related purposes, including but not limited to reading and learning from the Data, analyzing the Data, modifying the Data and generally preparing your Submission and any underlying models.
- Participants agree to use suitable measures to prevent persons who have not formally agreed to these rules from gaining access to the Data and agree not to transmit, duplicate, publish, redistribute or otherwise provide or make available the Data to any party not participating in the Challenge.
- Participants agree that participation in the Challenge shall not be construed as having or being granted a license (expressly, by implication, estoppel, or otherwise) under, or any right of ownership in, any of the Data.
- During the course of the challenge the participants will generate metadata. Any metadata submitted to the Challenge may be used by NVIDIA for any purpose, including distribution to other participants in the Challenge as part of the Data, or for any other commercial or non- commercial purpose.

Upon the conclusion of the Challenge, participants may use the Data, including models trained on the Data, for non-commercial, academic purposes only. Participants may not distribute the Data, in whole or in part; a minimal portion of the Data may be reproduced as part

of research papers, as appropriate, provided that any such use of the Data is accompanied by the appropriate acknowledgement(s) and no unredacted faces or license plates are reproduced or otherwise displayed. Phrasing for acknowledgements will be provided prior to the submission deadline for publications to the Challenge workshop.

Bibliography

1. KUMPOŠT, Petr; PUDIŠ, Ondrej; SUŠICKÝ, Marek; KARELLA, Tomáš. *Introduction to the traffic surveys*. 2021.
2. KUMPOŠT, Petr; PUDIŠ, Ondrej. *Understanding the problems of traffic surveys*. 2021.
3. KOBASA, Paul A.; DAVIES, Brad. *Transportation*. World Book Inc., 2009. Inventions and Discoveries. ISBN 978-0-7166-0381-8. Available also from: `https://www.proquest.com/legacydocview/EBC/5994249`.
4. EUROSTAT. Passenger cars in the EU [online]. 2020 [visited on 2021-09-22]. ISSN 2443-8219. Available from: `https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Passenger_cars_in_the_EU`.
5. KEIM, Martin; CERNY, Philipp. *European Mobility Atlas*. 2021. ISBN 978-9-46400743-5. Available also from: `https://eu.boell.org/sites/default/files/2021-07/EUMobilityatlas2021_2ndedition_FINAL_WEB.pdf`.
6. HEYWORTH, Anna; DEPARTMENT FOR TRANSPORT. *Road traffic estimates in Great Britain: 2020* [online]. 2021 [visited on 2021-09-25]. Available from: `https://www.gov.uk/government/statistics/road-traffic-estimates-in-great-britain-2020`.
7. CABRERA-ARNAU, Carmen; BISHOP, Steven R. Urban population size and road traffic collisions in Europe. *Public Library of Science*. 2021, vol. 16, no. 8, pp. 1–13. Available from DOI: `10.1371/journal.pone.0256485`.
8. AFRIN, Tanzina; YODO, Nita. A Survey of Road Traffic Congestion Measures towards a Sustainable and Resilient Transportation System. *Sustainability*. 2020, vol. 12, no. 4660. ISSN 2071-1050. Available from DOI: `10.3390/su12114660`.
9. MCCLINTOCK, Miller. The Traffic Survey. *The ANNALS of the American Academy of Political and Social Science*. 1927, vol. 133, no. 1, pp. 8–18. Available from DOI: `10.1177/000271622713300103`.
10. PAĽO, J.; CABAN, J.; KIKTOVÁ, M.; ČERNICKÝ, Ľ. The comparison of automatic traffic counting and manual traffic counting. *IOP Conference Series: Materials Science and Engineering*. 2019, vol. 710. ISSN 1757-899X. Available from DOI: `10.1088/1757-899X/710/1/012041`.
11. PENGJUN, Zheng; MCDONAD, Mike. An Investigation on the Manual Traffic Count Accuracy. *Procedia - Social and Behavioral Sciences*. 2012, vol. 43. ISSN 18770428. Available from DOI: `10.1016/j.sbspro.2012.04.095`.
12. METROCOUNT LTD. *Traffic Counters and Classifier* [online]. 2021 [visited on 2021-10-08]. Available from: `https://metrocount.com/products`.

13. METROCOUNT LTD. *The Traffic Survey That Led Amsterdam to Ban Mopeds from Bike Paths* [online]. 2020 [visited on 2021-10-08]. Available from: https://metrocount.com/ban_mopeds_from_bike_paths_amsterdam.

14. TCS INC. *TCS For Surveys* [online]. 2021 [visited on 2021-10-08]. Available from: https://tcsforsurveys.com.au.

15. YARGER ENGINEERING INC. *Traffic Study Project Descriptions* [online]. 2021 [visited on 2021-10-08]. Available from: http://www.yargerengineering.com/project_descriptions_studies.html.

16. CERTICON A.S. *CertiConVis* [online]. 2021 [visited on 2021-10-15]. Available from: https://www.certiconvis.cz.

17. GOODVISION LTD. *GoodVision Solutions* [online]. 2021 [visited on 2021-10-15]. Available from: https://goodvisionlive.com/solutions.

18. RCE SYSTEMS S.R.O. *Data from sky - traffic framework* [online]. 2021 [visited on 2021-10-15]. Available from: https://datafromsky.com.

19. SKANSI, Sandro. Feedforward Neural Networks. In: *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer International Publishing, 2018, pp. 79–105. Undergraduate Topics in Computer Science. ISBN 978-3-319-73004-2. Available also from: https://doi.org/10.1007/978-3-319-73004-2_4.

20. SKANSI, Sandro. Convolutional Neural Networks. In: *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer International Publishing, 2018, pp. 121–133. Undergraduate Topics in Computer Science. ISBN 978-3-319-73004-2. Available also from: https://doi.org/10.1007/978-3-319-73004-2_6.

21. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. ISBN 978-0-262-33737-3.

22. LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, vol. 86, no. 11, pp. 2278–2324. ISSN 1558-2256. Available from DOI: 10.1109/5.726791.

23. HUBEL, D. H.; WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*. 1968, vol. 195, no. 1, pp. 215–243. ISSN 0022-3751. Available from DOI: 10.1113/jphysiol.1968.sp008455.

24. TENSORFLOW. *Object detection example* [online]. 2022 [visited on 2022-03-05]. Available from: https://tfhub.dev/tensorflow/efficientdet/d5/1.

25. ZHAO, Zhong-Qiu; ZHENG, Peng; XU, Shou-tao; WU, Xindong. Object Detection with Deep Learning: A Review. 2019. Available from DOI: 10.48550/arXiv.1807.05511.

26. ZHANG, Shifeng; CHI, Cheng; YAO, Yongqiang; LEI, Zhen, et al. Bridging the Gap Between Anchor-Based and Anchor-Free Detection via Adaptive Training Sample Selection. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, 2020, pp. 9756–9765. ISBN 978-1-72817-168-5. Available from DOI: 10.1109/CVPR42600.2020.00978.

27. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian, et al. SSD: Single Shot MultiBox Detector. In: LEIBE, Bastian; MATAS, Jiri; SEBE, Nicu; WELLING, Max (eds.). *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. Lecture Notes in Computer Science. ISBN 978-3-319-46448-0. Available from DOI: 10.1007/978-3-319-46448-0_2.

28. LIN, Tsung-Yi; GOYAL, Priya; GIRSHICK, Ross; HE, Kaiming, et al. Focal Loss for Dense Object Detection. 2018. Available from DOI: 10.48550/arXiv.1708.02002.

29. REDMON, Joseph; FARHADI, Ali. YOLO9000: Better, Faster, Stronger. 2016. Available from DOI: 10.48550/arXiv.1612.08242.

30. LIN, Tsung-Yi; DOLLAR, Piotr; GIRSHICK, Ross; HE, Kaiming, et al. Feature Pyramid Networks for Object Detection. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, 2017, pp. 936–944. ISBN 978-1-5386-0457-1. Available from DOI: 10.1109/CVPR.2017.106.

31. GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *Conference on Computer Vision and Pattern Recognition*. 2014. Available from DOI: 10.48550/arXiv.1311.2524.

32. GIRSHICK, Ross. Fast R-CNN. 2015. Available from DOI: 10.48550/arXiv.1504.08083.

33. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016. Available from DOI: 10.48550/arXiv.1506.01497.

34. TIAN, Zhi; SHEN, Chunhua; CHEN, Hao; HE, Tong. FCOS: Fully Convolutional One-Stage Object Detection. *International Conference on Computer Vision*. 2019, p. 13. Available from DOI: 10.48550/arXiv.1904.01355.

35. REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 779–788. ISBN 978-1-4673-8851-1. Available from DOI: 10.1109/CVPR.2016.91.

36. LAW, Hei; DENG, Jia. CornerNet: Detecting Objects as Paired Keypoints. [N.d.], no. 2, p. 17. Available from DOI: 10.48550/arXiv.1808.01244.

37. TAN, Mingxing; PANG, Ruoming; LE, Quoc V. EfficientDet: Scalable and Efficient Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020, no. 7. Available from DOI: 10.48550/arXiv.1911.09070.

38. TAN, Mingxing; LE, Quoc V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *International Conference on Machine Learning*. 2020, no. 5. Available from DOI: 10.48550/arXiv.1905.11946. arXiv: 1905.11946.

39. SANDLER, Mark; HOWARD, Andrew; ZHU, Menglong; ZHMOGINOV, Andrey, et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520. Available from DOI: 10.48550/arXiv.1801.04381.

40. LIU, Shu; QI, Lu; QIN, Haifang; SHI, Jianping, et al. Path Aggregation Network for Instance Segmentation. *Conference on Computer Vision and Pattern Recognition*. 2018. Available from DOI: 10.48550/arXiv.1803.01534.

41. BODLA, Navaneeth; SINGH, Bharat; CHELLAPPA, Rama; DAVIS, Larry S. Soft-NMS – Improving Object Detection With One Line of Code. 2017. Available from DOI: 10.48550/arXiv.1704.04503.

42. KALMAN, R. E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*. 1960, vol. 82, no. 1, pp. 35–45. ISSN 0021-9223. Available from DOI: 10.1115/1.3662552.

43. MONTELLA, Corey. The Kalman Filter and Related Algorithms: A Literature Review. 2011. Available also from: https://www.researchgate.net/publication/236897001_The_Kalman_Filter_and_Related_Algorithms_A_Literature_Review.

44. BECKER, Alex. *Introduction to Kalman Filter* [online]. 2022 [visited on 2022-03-13]. Available from: https://www.kalmanfilter.net/multiSummary.html.

45. JONKER, R; VOLGENANT, A. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*. 1987, vol. 38, pp. 325–340. Available from DOI: 10.1007/BF02278710.

46. CASTANÓN, David A.; DRUMMOND, O. E.; BELLOVIN, M. S. Comparison of 2-D assignment algorithms for sparse, rectangular, floating point, cost matrices. *Proceedings of the SDI Panels on Tracking*. 1990, vol. 4, pp. 4–81.

47. LUO, Wenhan; XING, Junliang; MILAN, Anton; ZHANG, Xiaoqin, et al. Multiple Object Tracking: A Literature Review. *Artificial Intelligence*. 2021, vol. 293, p. 103448. ISSN 00043702. Available from DOI: 10.1016/j.artint.2020.103448.

48. BETKE, Margrit; HARITAOGLU, Esin; DAVIS, Larry S. Real-time multiple vehicle detection and tracking from a moving vehicle. *Machine Vision and Applications*. 2000, vol. 12, no. 2, pp. 69–83. ISSN 0932-8092, ISSN 1432-1769. Available from DOI: 10.1007/s001380050126.

49. KOLLER, Dieter; WEBER, Joseph; MALIK, Jitendra. Robust Multiple Car Tracking with Occlusion Reasoning. 1996.

50. CONTRIBUTORS, MMTracking. *MMTracking: OpenMMLab video perception toolbox and benchmark*. 2020. Available also from: https://github.com/open-mmlab/mmtracking.

51. BEWLEY, Alex; GE, Zongyuan; OTT, Lionel; RAMOS, Fabio, et al. Simple Online and Realtime Tracking. *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. Available from DOI: 10.1109/ICIP.2016.7533003.

52. WOJKE, Nicolai; BEWLEY, Alex; PAULUS, Dietrich. Simple Online and Realtime Tracking with a Deep Association Metric. 2017, p. 5. Available from DOI: 10.48550/arXiv.1703.07402.

53. ZHANG, Yifu; WANG, Chunyu; WANG, Xinggang; ZENG, Wenjun, et al. FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking. *International Journal of Computer Vision*. 2021, vol. 129, no. 11, pp. 3069–3087. ISSN 0920-5691, ISSN 1573-1405. Available from DOI: 10.48550/arXiv.2004.01888.

54. DAI, Zhe; SONG, Huansheng; WANG, Xuan; FANG, Yong, et al. Video-Based Vehicle Counting Framework. *IEEE Access*. 2019, vol. 7, pp. 64460–64470. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2019.2914254.

55. ECONOMIC AND SOCIAL COUNCIL OF THE UNITED NATIONS. Consolidated Resolution on the Construction of Vehicles. *World Forum for Harmonization of Vehicle Regulations* [online]. 2017, no. 6 [visited on 2022-05-03]. Available from: https://unece.org/fileadmin/DAM/trans/main/wp29/wp29resolutions/ECE-TRANS-WP.29-78r6e.pdf.

56. EUROPEAN COMMISSION. Vehicle categories [online]. 2022 [visited on 2022-05-03]. Available from: https://ec.europa.eu/growth/sectors/automotive-industry/vehicle-categories_en.

57. ČESKÝ NORMALIZAČNÍ INSTITUT. Základní automobilové názvosloví. Druhy silničních vozidel. Definice základních pojmů. In: *Česká technická norma*. 1983. No. ČSN 30 0024. Available also from: http://www.normy.cz/Detailnormy.aspx?k=23165.

58. MINGXING, Tan. *Brain AutoML EfficientDet*. Google Research, 2021. Available also from: https://github.com/google/automl/tree/1.2/efficientdet.

59. CHEN, Xiangning; XIE, Cihang; TAN, Mingxing; ZHANG, Li, et al. Robust and Accurate Object Detection via Adversarial Learning. *Conference on Computer Vision and Pattern Recognition*. 2021. Available from DOI: 10.48550/arXiv.2103.13886.

60. NAPHADE, Milind; WANG, Shuo; ANASTASIU, David C.; TANG, Zheng, et al. The 5th AI City Challenge. 2021. Available from DOI: 10.48550/arXiv.2104.12233.

61. KUMPOŠT, Petr; MOBILE LABORATORY OF TRAFFIC ANALYSIS. *Czech road video recordings*. Faculty of Transporation, CTU in Prague, 2022.

62. CZECH TECHNICAL UNIVERSITY. *Research Center for Informatics*. 2022. Available also from: http://rci.cvut.cz/.

63. BERNARDIN, Keni; ELBS, Alexander; STIEFELHAGEN, Rainer. Multiple Object Tracking Performance Metrics and Evaluation in a Smart Room Environment. *Proceedings of IEEE International Workshop on Visual Surveillance*. 2006, p. 8.

64. LIU, Ze; LIN, Yutong; CAO, Yue; HU, Han, et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. 2021. Available from DOI: 10.48550/arXiv.2103.14030.

Contents of enclosed CD

```
/
├── code/ ..... the source code of our application
│   ├── .github/ ..... GitHub actions configuration
│   ├── backend/ ..... the code of the backend service
│   ├── frontend/ ..... the code of the frontend service
│   ├── docker-compose.yaml ..... all the services orchestrating configuration
│   └── README.md
├── evaluation_exports/ ..... CSV exports of our vehicle counting experiments
├── input_video/ ..... source recordings folder
├── jupyter/ ..... jupyter notebooks with tries and exports of graphics
├── models/ ..... object detection models checkpoints
│   ├── efficientdet-d5-advprop-aa/
│   └── efficientdet-d6/
├── output_analysis/ ..... folder to store results of our analyses
├── text/ ..... the source code of the thesis text
├── video_export/ ..... video exports of some of our experiments
├── database_state.sql ..... a PostgreSQL database dump
└── neptune_experiments.csv ..... experiments monitoring data
```