BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

DETECTION AND RECOGNITION OF LICENSE PLATES

DETEKCE A ROZPOZNÁNÍ REGISTRAČNÍCH ZNAČEK VOZIDEL

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JIŘÍ TYKVA**
AUTOR PRÁCE

SUPERVISOR **Ing. ROMAN JURÁNEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

Student: **Tykva Jiří**
Programme: Information Technology
Title: **Detection and Recognition of License Plates**
Category: Image Processing

Assignment:

1. Learn methods for object detection and recognition with neural networks, focus on methods for license plate detection and recognition
2. Select a method for automatic detection and recognition of license plates in images
3. Find suitable training data, if required generate synthetic data
4. Implement the method and develop software for traffic analysis in real time
5. Create demonstration materials

Recommended literature:

- Recommended by advisor

Requirements for the first semester:

- Items 1 to 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Juránek Roman, Ing., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: July 31, 2020
Approval date: December 3, 2019

Abstract

The goal of this Bachelor's thesis is to design, implement, and test a system that can detect and recognize license plates in real-time by using neural networks. The collected data will be saved into the database. The system's architecture is divided into three main parts. The first part handles the license plate detection in the image by making use of the TensorFlow Object Detection API. The detector reaches the accuracy of 98.15 % AP with a speed of roughly 14 fps. The second part deals with license plate tracking by using the algorithm SORT. The third part holistically recognizes the text of the license plate and can reach up to 0.6% character error rate and 2% word error rate. The system may be used by law enforcement for purposes such as for keeping track of stolen vehicles or for the automatic road tolling.

Abstrakt

Cílem této bakalářské práce je návrh, implementace a testování systému, který v reálném čase pomocí neuronových sítí bude detekovat a rozpoznávat registrační značky vozidel. Nasbíraná data budou ukládána do databáze. Architektura systému je rozdělena do tří hlavních částí. První část řeší detekci registrační značky v obraze pomocí TensorFlow Object Detection API. Detektor dosahuje přesnosti 98.15 % AP při rychlosti kolem 14 fps. Druhá část se zabývá sledováním značek ve videu pomocí algoritmu SORT. Třetí část systému se věnuje holistickému rozpoznávání textu registrační značky a dosahuje až 0.6% chybovosti při rozpoznávání jednotlivých znaků a 2% chybovosti při rozpoznávání celého textu. Výsledný systém lze použít například pro policejní oddělení za účelem sledování kradených vozidel či automatického vybírání dálničních poplatků.

Keywords

license plate, machine learning, dataset, neural networks, computer vision, image processing, character recognition, object detection, TensorFlow

Klíčová slova

registrační značka, strojové učení, dataset, neuronové sítě, počítačové vidění, zpracování obrazu, rozpoznávání znaků, detekce objektu, TensorFlow

Reference

TYKVA, Jiří. *Detection and Recognition of License Plates.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Roman Juránek, Ph.D.

Rozšířený abstrakt

Tato bakalářská práce se zabývá oblastí zpracování obrazu. Cílem této práce je návrh, implementace a testování systému, který v reálném čase pomocí neuronových sítí bude detekovat a rozpoznávat registrační značky vozidel. Všechna data budou ukládána do databáze či vyobrazena do videa pro tvorbu demonstračních materiálů.

Architektura systému je rozdělena do tří hlavních částí. První část řeší detekci registrační značky v obraze pomocí TensorFlow Object Detection API s předpřipraveným modelem SSD-MobileNet-v2-COCO, který byl následně přetrénován pro účely této práce. Detektor dosahuje přesnosti 98.15 % AP a rychlosti kolem 15 fps při zpracování videa o rozlišení 720×480, respektive 14 fps při rozlišení 1280×720.

Druhá část se zabývá sledováním značek ve videu pomocí algoritmu SORT. Díky sledování registračních značek je systém schopen vybrat pouze jednu potenciálně nejlepší detekci za celou dobu výskytu značky na scéně. SORT tedy drasticky redukuje počet detekcí určených k rozpoznání a předchází redundanci, čímž umožňuje data efektivně ukládat do databáze.

Třetí část systému se věnuje rozpoznávání textu. Vyhýbá se segmentaci znaků, namísto toho přistupuje k problému holisticky. K tomu je použita hluboká konvoluční neuronová síť a rekurentní neuronová síť s dlouhou krátkodobou pamětí. Experimentální výsledky dosahují 0.6% chybovosti při rozpoznávání jednotlivých znaků a 2% chybovosti při rozpoznávání celého textu. Tato část zvládne za jednu sekundu rozpoznat 39 obrázků o velikosti 9.5 kilopixel, respektive 46 obrázků o velikosti 5 kilopixel.

Za účelem trénování a testování detektoru a rozpoznávače byly použity tři ručně anotované datové sady. Datové sady celkově obsahují 184 000 vzorků evropských značek, které jsou pořízeny z několika různých pozic a mají rozdílné světelné podmínky, kvalitu, velikost, rozmazání, natočení, šum, atd.

Výsledkem této práce je program *alpr*, jenž je implementovaný v jazyce Python a je schopen pracovat ve dvou režimech. První režim zpracovává video v reálném čase, ukládá data o rozpoznaných značkách do databáze a vypisuje pro přehled informace na standardní výstup. Druhý režim slouží k vytvoření ukázkového videa, v nemž jsou zakresleny všechny detekce společně s rozpoznaným textem.

Nasbíraná data lze pak využít například pro policejní oddělení za účelem sledování kradených vozidel, jízdy na červenou, automatické vybírání dálničních poplatků, atd.

Detection and Recognition of License Plates

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Roman Juránek, Ph.D. from the Brno University of Technology, Faculty of Information Technology. The supplementary information was provided by Prof. Matthew Montebello from University of Malta, Faculty of Information & Communication Technology. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jiří Tykva
July 27, 2020

Acknowledgements

I would like to thank Ing. Roman Juránek, Ph.D. and prof. Matthew Montebello for supervising me during the whole semester and for their help and professional advices.

Contents

Chapter 1

Introduction

Detection and recognition of license plates consists of numerous methods which transfer image data such as camera recordings into a plain text containing the license plate numbers. This can be very practical for vehicle tracking especially when used by the police department for law enforcement purposes. Such data is frequently utilized for keeping track of stolen vehicles, red-light enforcement, speeding charges, and bus lane control. Moreover it can be applied for road tolling or parking management.

However it also has its drawbacks. There have been cases when the data was abused by the police to target and harass minorities and religious or gay people based on the places they were going to try to create opportunities to arrest them. Thus information of this nature need to be handled with caution.

In addition older systems happen to falsely accuse drivers often. Although improvements in technology have reduced the error rates, it is still a problem. The system has to deal with a combination of low image quality, motion blur, bad weather conditions, inferior lightning and contrast, poor camera angle, damaged or dirty license plate, etc.

The aim of this Bachelor's thesis is to design, implement, and test a system able to detect and recognize license plates from video records in real-time using neural networks. This project focuses only on European license plates.

I have chosen this topic because it gives me the opportunity to be a part of a solution that nowadays is considered a global issues. From this point forward, I would like to dedicate my studies to artificial intelligence, which I find extremely fascinating, and plays a huge role in this work. Furthermore, I believe artificial intelligence will eventually take over many other fields including but not limited to those in the computer science industries.

This paper is divided into six chapters. The introduction is followed by an overview of the current significant methods outlining the object detection, object tracking and license plate recognition approaches 2. The third chapter 3 describes the acquisition, the specifics, and the ground truth assignment for the used datasets. The next chapter 4 proposes the system architecture design among with the design of its individual parts. The fifth chapter 5 is dedicated to the implementation of the system, its usage and the experimental results. Finally, the last chapter 6 reviews the achieved goals of this work and potential improvements in the future.

Chapter 2

Related Work

Detection and recognition of license plates can be divided into three separate tasks. To begin with, it is required to localize the license plates within the image. The most common used technologies for this kind of a problem are presented in section 2.1. To keep track of each license plate throughout the scene, it is necessary to employ an object tracker. Different approaches on tracking multiple objects are described in section 2.2. After localization, the system must proceed to the recognition of the license plate. Such methods are briefly introduced in section 2.3.

2.1 Object Detection Methods

This section discusses object detection methods that could be split into two groups. The first group is classification based and it works in two stages. At first, the interesting regions are selected from the input image and then a Convolutional Neural Network (CNN) is used for classifying objects within those regions. In general, these solutions are slow and hard to optimize because of their complex pipeline. Therefore, some of them are not suitable for real-time situations. The most known examples of this type is the Region Convolutional Neural Network (R-CNN) 2.1.1 and its successor Fast R-CNN 2.1.2 and Faster R-CNN 2.1.3 which are described below.

The second group consists of algorithms based on regression. Instead of selecting interesting regions of the image, it scans the whole image and predicts classes and bounding boxes in one run. In general, this approach tends to be faster but lacks accuracy. The You Only Look Once (YOLO) 2.1.4 and Single Shot MultiBox Detector (SSD) 2.1.5 methods that fall into this category are outlined in the following sections.

2.1.1 Region Convolutional Neural Network

R-CNN is one of the object detection methods introduced by Ross Girshick et al. [10].

To fully understand R-CNN, it is important to know what a CNN is and how it is related with this method. CNN is composed of multiple layers. One input layer, one output layer and several hidden layers in between. The particles of the layers, called neurons, are connected to every single neuron in the next layer. Every connection has been given a certain weight which corresponds to how much the value of the first neuron will influence the value of the subsequent one. By adjusting the weights during the training phase we can achieve seemingly intelligent behavior of the network. Calculating the proper weights is done by loss function. It matches the predicted value obtained from the output layer

with the actual one to determine the loss and updates the weights based on the gradient descent method so that the overall loss is minimized.

CNNs are best used for recognizing patterns, shapes, colors, and textures of an input image. The hidden layers are convolutional layers in this context. They act as a filter that transforms the received input using a specific feature or pattern and sends it to the next layer. This continues up to the output layer which could be understood as a feature vector of the original image.

However, CNNs are too slow and computationally very expensive. R-CNN solves this problem in 3 stages as described in Figure 2.1. At first, only around 2,000 region proposals for the input image are generated by using a selective search. That greatly reduces the number of regions that it needs to work with. Every proposal is then fed to the CNN that extracts a 4096-dimensional feature vector. Afterwards, the feature vector is used to classify its respective region by category-specific Support Vector Machines (SVM).

In order to achieve a fixed-length feature vector, all pixels of the region are first warped to the required size, 224x224 in this case. This allows sharing features across all categories and appearance modes. Another key property is that the feature vector is rather small compared to other common approaches (for example 360k-dimensional used by UVA detection system [28] *vs.* 4k-dimensional).

R-CNN has relatively good mean average precision (mAP), achieving a final mAP of 43.5 % on PASCAL Visual Object Classes (VOC) Challenge 2010 [6] which serves as a benchmark for assessing object detector performance. However, the great lack of speed makes it virtually impossible to implement in real-time applications as one test image takes around 47 seconds to process.

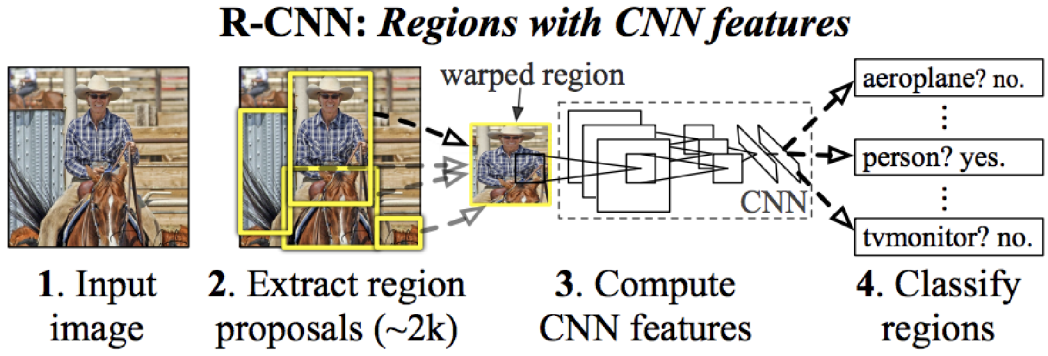

Figure 2.1: **(1)** The system takes an input image. **(2)** The input image is extracted to ~2,000 region proposals. **(3)** The CNN computes a feature vector for every proposal. **(4)** Each region is being classified by class-specific SVM. Source [7].

2.1.2 Fast R-CNN

One of the authors of R-CNN, Ross Girshick, proposed Fast R-CNN [9] which builds directly on top of R-CNN and employs several mechanisms to improve speed and accuracy. Moreover, unlike previous work, now the architecture is end-to-end and can be trained with back-propagation.

Fast R-CNN architecture is visualized in Figure 2.2 and is described as follows. The key here is that the CNN is not fed with 2,000 region proposals every time anymore. Instead, the network processes the whole image and produces a feature map. Then for each region proposal, or region of interest (RoI), the pooling layer extracts a fixed-length feature vector from the feature map. Every feature vector then results in two outputs. The first indicates the softmax probabilities representing which object class was found. The second is the bounding box regression predicting offset values for the bounding box.

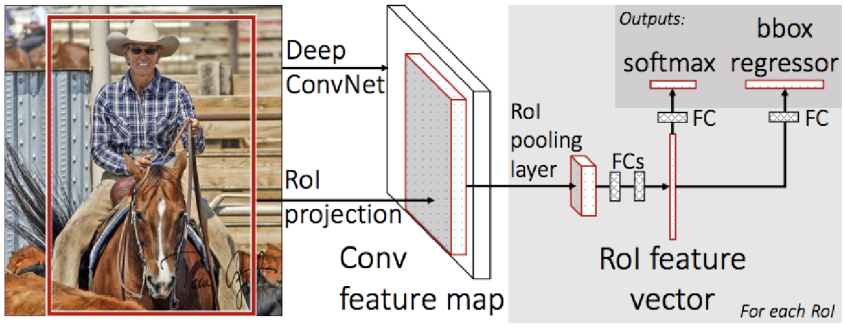

Figure 2.2: An illustration of the Fast R-CNN workflow. Source [7].

2.1.3 Faster R-CNN

Region proposals are still the biggest computational bottleneck of the Fast R-CNN. Therefore, Ross Girshick et al. [25] introduces the Regional Proposal Network (RPN) to replace the selective search process.

The input image is passed first through a pre-trained CNN creating a convolutional feature map. The RPN then uses the feature map to find bounding boxes, which may contain objects. The bounding boxes are represented by anchors of different sizes and ratios spread uniformly throughout the image. In [25], $k = 9$ anchors is used for each location with box areas of 128^2, 256^2, and 512^2 pixels and 3 aspect ratios of 1:1, 1:2, and 2:1 as illustrated in Figures 2.3 and 2.4. For a convolutional feature map of a size $W \times H$ there are WHk anchors in total.

In the next step, RPN feeds each anchor to a box-classification layer (cls) and a box-regression layer (reg). The cls layer outputs $2k$ scores that estimate the probability of the object both being inside and not being inside the anchor. The reg layer produces $4k$ offsets for each anchor to better fit the object $(\Delta_{x_{center}}, \Delta_{y_{center}}, \Delta_{width}, \Delta_{height})$. Since many anchors usually overlap, non-maximum suppression (NMS) is applied to reduce redundancy. If two anchors have Intersection-over-Union (IoU) bigger than some predefined threshold, it discards the one with smaller probability to contain an object. This decreases the overall number of proposals without any harm to the ultimate detection accuracy.

After the region proposals are generated by the RPN, the Fast R-CNN 2.1.2 is adopted to work as the detection network. Both the RPN and Fast R-CNN share a common set of convolutional layers in order to share computation. Thus making the region proposals nearly cost-free. Faster R-CNN enables the system to run at 5–15 fps with a high-quality region proposal and therefore a better overall mAP.

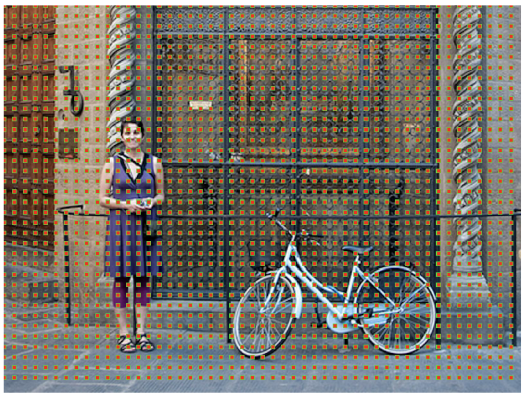

Figure 2.3: Anchor centers spread throughout the input image. Source [26].

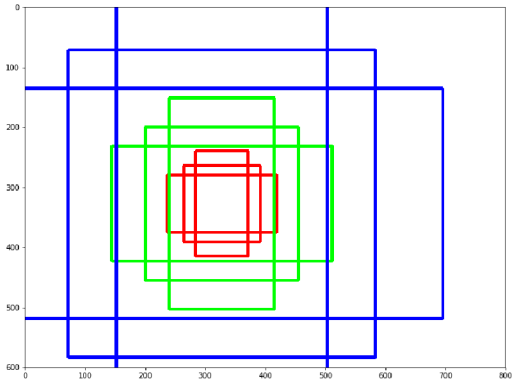

Figure 2.4: Nine anchors with box areas of 128^2, 256^2, and 512^2 pixels and 3 aspect ratios of 1:1, 1:2, and 2:1. Source [8].

2.1.4 You Only Look Once

You Only Look Once (YOLO) is a real-time object detection method published by Joseph Redmon et al. [23]. It uses one CNN to simultaneously predict bounding boxes and class probabilities for those boxes in one run of the algorithm processing the whole image.

It divides the input image into a grid of S x S cells. If the center of the object is located inside the grid cell, that cell is responsible for detecting that object. Each of these grid cells predicts B bounding boxes and whether the bounding box contains the object, or part of it. Then it uses this information to try and predict the class of the object with certain confidence. Hence, every bounding box, the output of the system, is specified as: The (x, y) coordinates representing the center of the bounding box, *width* and *height* of the bounding box, the *class* of the object and C class probability which indicates the class-specific confidence. Typically, a majority of the bounding boxes will not have an object inside. Therefore, a process called non-max suppression is used to remove unnecessary boxes with low probability to contain objects and those that share big areas with other boxes. The YOLO workflow is shown in Figure 2.5.

The fact that it sees the complete image at once gives it contextual information that helps avoid false positives. Furthermore, YOLO understands generalized representations of objects. This means it can be trained on real world images and is still quite good at predicting bounding boxes and detection, compared to other methods, if applied to artwork. The YOLO architecture is extremely fast, enabling it to process 45 frames per second or 155 frames per second while using a smaller version of the network called Fast YOLO. Nonetheless, YOLO falls behind in accuracy, especially when dealing with small objects that appear in groups, since the strong spacial constraints limit the number of nearby objects that can be predicted.

2.1.5 Single Shot MultiBox Detector

Liu et al. introduced a method called Single Shot MultiBox Detector (SSD) [19] which is fast and designed to detect multiple categories with a single shot evaluation of an input image.

SSD eliminates bounding box proposals which is typically expensive. Instead, it encapsulates all of the computation in a single network. The architecture includes progressively

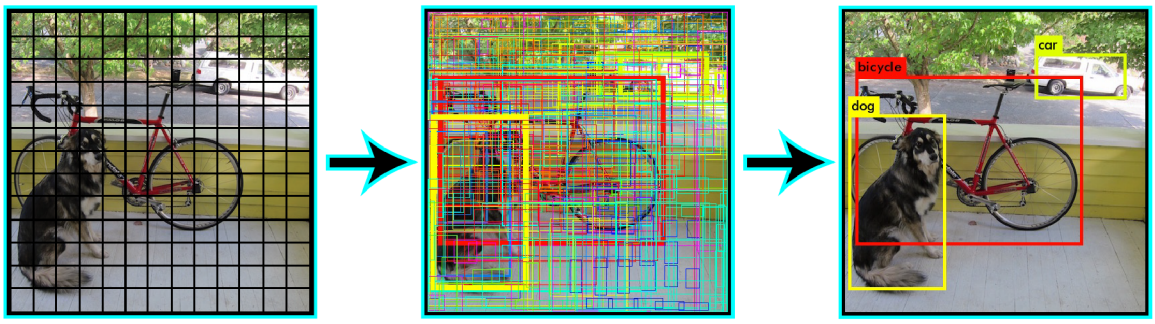

Figure 2.5: **Left:** The image is divided into grid of S x S cells. **Center:** The network predicts bounding boxes and probabilities for each grid cell. **Right:** The result after bounding boxes being removed based on the predicted probabilities. Source [24].

decreasing convolutional layers thus enabling it to extract multiple feature maps at different resolutions. This naturally handles detections of different scales and improves the detection quality. The topmost feature maps have large receptive fields and can deal with large objects; while the feature maps from the lower layers capture more fine details and are able to detect very small objects.

To create predictions, every feature map location is associated with a set of bounding boxes at different scales and aspect ratios. These are called priors and could be referred to as anchors in Faster R-CNN 2.1.3. The priors are illustrated in Figure 2.6. Then both confidences for all object categories and four offsets are computed for each prior. However, a large number of boxes will not contain any object therefore it is essential to perform a non-maximum suppression to produce the final detections. By combining predictions for all priors from all locations of many feature maps, the predictions become diverse and cover objects of many various shapes and sizes.

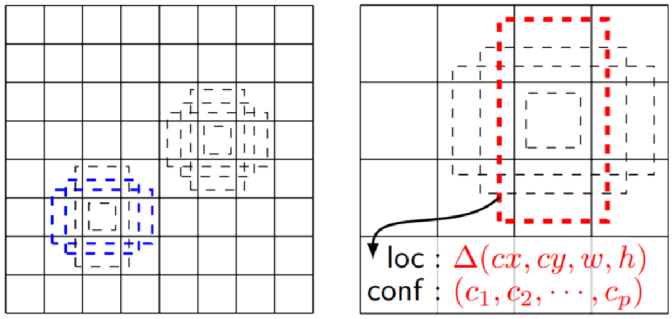

Figure 2.6: Exemplary feature maps of different sizes (e.g. 8×8 (left) and 4×4 (right)). Each location is assigned a set (e.g. 4) of priors of different aspect ratios and scales. Each box is then predicted with both the shape offsets and the confidence for all object categories. Source [19].

2.2 Object Tracking Methods

Since the object detection is applied on the whole sequence of frames from the video, it would produce many redundant results since the subsequent frames are very similar. To avoid this issue, the object tracker can be employed to assign IDs to individual objects and

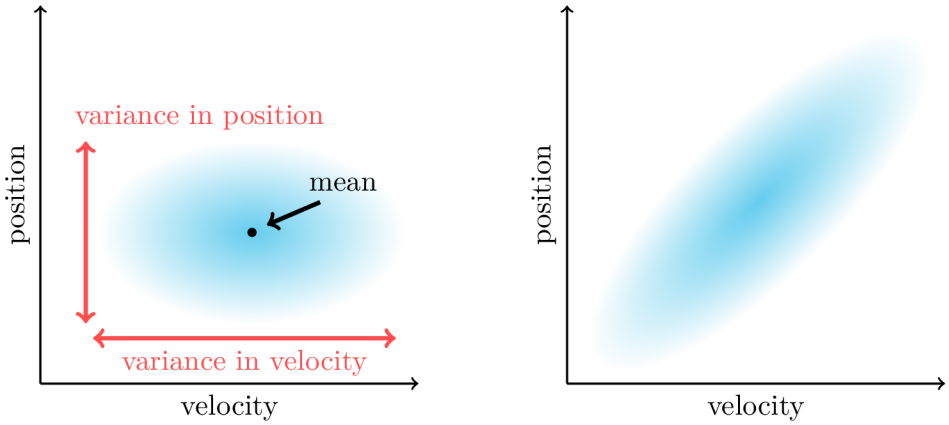

Figure 2.7: **Left:** An example of inaccurate position and velocity sensor readings. **Right:** The position and velocity are correlated. The faster the object is, the bigger the change that it will travel further and vice versa.

then the tracker carries them forward across the following frames. This way the system is able to capture only one representative sample for each license plate that appears in the video.

An auxiliary algorithm called the Kalman filter is used to predict possible object locations in the future and it is presented in 2.2.1. SORT method focuses on the simplicity and real-time application which is described in 2.2.2 and its successor Deep SORT which on top of that applies deep association metric is summarized in 2.2.3.

2.2.1 Kalman Filter

In 1960, R.E. Kalman published his paper [15] describing a recursive solution to the discrete-data linear filtering problem. Named after him, the Kalman filter can be exploited to track the license plates in the image sequence. Based on information that is uncertain, it produces an educated guess of what the dynamic system is going to look like in the next step. The main idea is that the sensor readings (the outputs of the object detection method in our case) are not always totally accurate. The Kalman filter improves the overall accuracy by combining the sensor readings with its own prediction of the next step in the system.

To formulate a prediction, the Kalman filter needs information about the current state read from the sensors. It then gathers more information about the measured data by applying the correlation captured in the covariance matrix. In another words, one measurement can indicate something about what the others could be as illustrated in the example Figure 2.7.

The next step is to predict the future state. It is usually done by using formulas represented by the state transition matrix that logically corresponds to the measured values. In the case of the position and velocity, it would be the basic kinetic formula. This matrix is used on the whole distribution of the sensor readings and its noise. The covariance matrix then needs to be updated.

However the prediction is not over yet. There might be some external influences that are not related to the state itself but still affect the system. If these additional influences

are known they can be modeled by the control matrix and control vector and be added to the prediction as a correction. Moreover, there could be some external uncertainty that is not being tracked. To include this into the estimation, the Kalman filter adds the process noise that can be constant after every prediction step as shown in Figure 2.8.

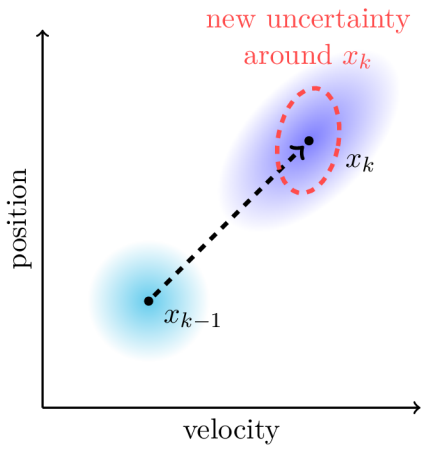

Figure 2.8: The transition of the state x_{k-1} into x_k with external uncertainty taken into consideration. This is done for every state in the original estimation which produces a bigger Gaussian blob.

The final stage is to incorporate a new noisy measurement into its prediction. The noisy measurement is a single Gaussian blob, where the noise is modeled as covariance matrix. To put this information together, the Kalman filter multiplies both Gaussian blobs producing only their overlap. If there is a strong confidence about the prediction, this overlap can be adjusted before the multiplication by setting the measurement noise level high so that it leans more towards the prediction blob. In the same manner, the measurement noise can be set to low emphasizing the sensor readings. This is called the Kalman gain and it specifies how much the estimate is going to change by a given measurement. Finally, the mean of this newly created overlap is the final product of the Kalman filter describing the current state of a dynamic system. It is more precise than if the measurements or predictions were treated individually and it represents the optimal solution for the tracking problem.

2.2.2 Simple Online and Real-Time Tracking

Simple Online and Real-Time Tracking (SORT) is an algorithm used for multiple object tracking published by Bewley et al. [2]. It ignores a lot complex issues such as an object re-identification or short-term, and long-term occlusion. Instead, it focuses on efficient and reliable handling of the common frame-to-frame associations. Due to its simplicity it needs only the bounding box positions and sizes to estimate both motion and data association.

The state of each tracked target is modeled as:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T$$

Where u and v represent the vertical and horizontal pixel of the center of the target, s and r represent the scale (area) and the aspect ratio of the target's bounding box. $\dot{u}$, $\dot{v}$, and $\dot{s}$

denote the target's predicted state via the Kalman filter 2.2.1. Note that r is considered a constant.

When the object detector outputs the bounding boxes, they are assigned to the existing targets by maximizing the IoU distance between each detection and all predictions. The assignment is rejected if the overlap is less than IoU_{min}. In this case the state is simply predicted without any corrections. However, if the target is not assigned for T_{Lost} frames, the track will get terminated. To create a new track the SORT considers any detection with overlap less than IoU_{min} as a new untracked object.

2.2.3 Deep SORT

Simple Online and Real-Time Tracking with a Deep Association Metric (Deep SORT) was published by Bewley et al. [30] as an extension to SORT 2.2.2. An exemplary output of Deep SORT is shown in Figure 2.9. It minimizes the issue with identity switches and it performs better at tracking through occlusions. This is accomplished by incorporating more informed metric that combines motion and appearance information into the association stage.

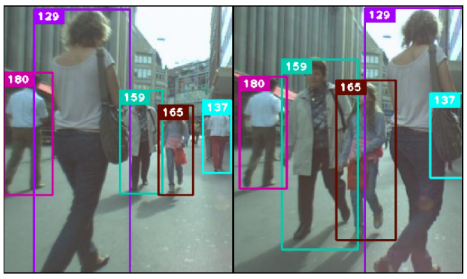

Figure 2.9: An example of Deep SORT output on the MOT challenge dataset [20] in a common tracking situation with frequent occlusion. Source [30].

The motion metric is obtained by using the Mahalanobis distance. It provides possible object locations based on motion and it is useful for short-term predictions. It takes the uncertainties from the Kalman filter into account by effectively measuring the distance between two distributions. Further, thresholding this distance can exclude unlikely associations.

The appearance information is helpful to recover identities after long-term occlusions. A pre-trained CNN is applied for each bounding box detection and it computes an appearance descriptor. It is able to extract features in a way that features from the same identity are close together while maintaining the features from the different identities far away in the feature space. In combination, both metrics complement each other by serving different aspects to the assignment problem. Influence of each metric can be controlled through hyperparameter as needed.

In comparison to SORT, the experimental results show that the number of times when the ID switches to a different previously tracked object was reduced by approximately 45 %. There is also a significant increase in maintaining identities throughout the longer occlusions. Although the runtime drops around 33 % due to the algorithm's complexity, it remains able to work in real-time.

2.3 License Plate Recognition Methods

Once we obtain a cropped image of a license plate, Optical Character Recognition (OCR) methods can be applied to produce the desired plain text. One of the advantages over the hand written text is that license plate characters are fairly uniformed, which makes this task easier. Nonetheless, there are still many challenges to face. This section describes some of the methods which focus on character segmentation and character recognition 2.3.1, 2.3.2 as

well as on holistic methods that perceive a license plate as a whole and avoid the character segmentation 2.3.3, 2.3.4.

2.3.1 Template Matching

The template matching approach was summarized in [22] by Poorvi Hedau et al. It is decomposed into character segmentation and character recognition.

At first, the input image with an extracted license plate is being converted into a grayscale. Contrast and brightness adjustments as well as noise removal by using median filtering follow up. The next step is to identify and segment all characters. The bounding box method is used to map each character into a single image.

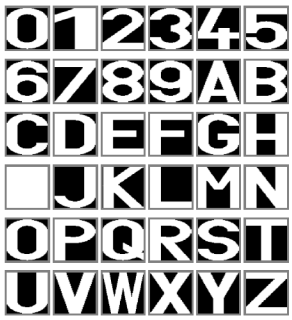

Figure 2.10: Example of character templates. Source [21].

This method then compares portions of individual characters against the templates 2.10. Matching is pixel to pixel based and it moves the template over all possible positions in the input image. As a result, it produces a coefficient indicating the deviation between the template and the input image in a certain position. If the deviation is small enough, the template is considered as the resultant character.

Experimental results of this method conducted by Poorvi Hedau et al. are shown in Table 2.1.

Total License Plates	**Total Number of Alphanumeric**	**Total Correct Alphanumeric Recognition**	**Total Incorrect Alphanumeric Recognition**	**Percentage of correctly recognized Alphanumeric**
118	1108	972	135	87.72 %

Table 2.1: The experimental results of Template Matching. Misrecognized alphanumeric are common characters similar in shape such as O *vs.* D, 5 *vs.* S or 8 *vs.* B. Source [22].

2.3.2 Support Vector Machine Based Recognition System

A Support Vecotr Machine (SVM) is a supervised machine learning model. It was used by Wen et al. [29] or by Parasuraman and Subin P.S [17] for license plate recognition.

This method requires character segmentation. It takes a cropped license plate image as an input and resizes it to a given width and height. Then it executes horizontal and vertical correction because license plates are prone to slant and distort due to different angles of orientation. Consequently, image gray equilibrium is performed and afterwards the whole image is binarized. Next, the projective technique is used to segment each character which is later resized to a uniform size.

Several different features are then extracted from each character to be classified by a pre-trained SVM. The SVM is able to find a hyperplane in an N-dimensional space (N – the number of features). It separates two classes of data points, thus creating a boundary.

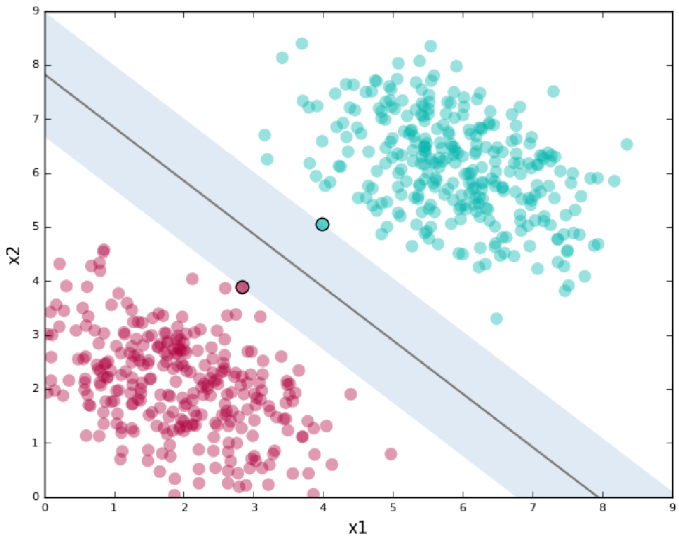

Figure 2.11: An example of the SVM. The two classes of data points (red and blue dots) are separated by a hyperplane. In 2-dimensional space, the hyperplane is a line. The line's blue background represents the maximum possible margin between data points from both classes. Source [16].

Then a class can be assigned to anything that falls onto either side of the hyperplane. The objective is to find the hyperplane that has the maximum margin from data points of both classes so that it can predict future data points more confidently. An example of the SVM is shown in Figure 2.11. Since SVM usually classifies in only two pattern classes, license plate recognition approaches this issue as a series of binary classification problems. After all possible characters are tested one by one using individual SVMs, the result with the highest confidence is chosen.

Although this method achieves satisfactory overall performance, it has the drawback that it heavily relies on correct character segmentation. Moreover, there must be pre-trained classifiers for each corresponding character.

2.3.3 License Plate Recognition by using Deep CNN and Long Short-Term Memories

Li and Shen have developed a novel approach for license plate recognition by using deep CNN and long short-term memories (LSTMs) [18]. This method omits segmentation of license plate characters, which should be highlighted. The reason is, proper character segmentation has a major influence on the whole recognition process. If the segmentation is executed poorly due to challenging conditions such as image blur, shadows, noise, etc., it will lead to incorrect recognition no matter how robust the recognizer is.

This method could be broken down to the three stages, as shown in the in Figure 2.12. To begin, the input image of the license plate needs some preprocessing. It is converted into a grayscale, padded with additional pixels on both left and right side and resized to the height that matches the input height of the CNN model. After, the CNN extracts a sequence of feature vectors from the image.

In order to complete the task of character string recognition, it is very helpful to have access to the past and the future context information. These contextual clues will make

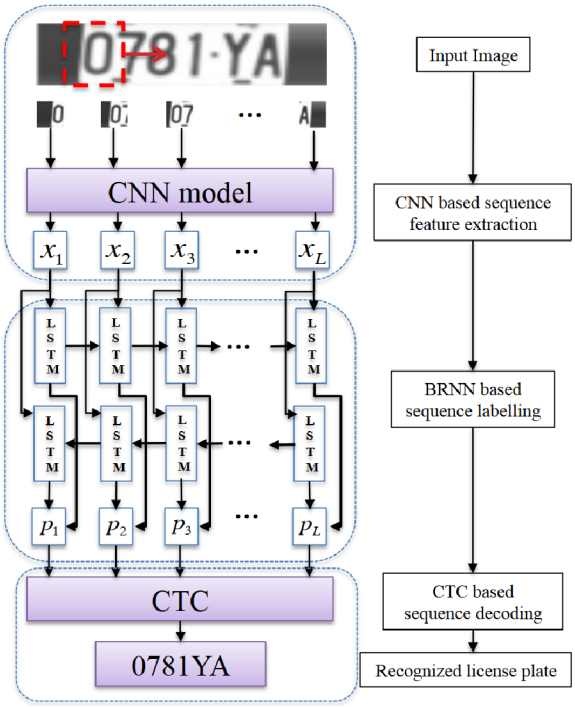

Figure 2.12: The workflow of the method proposed by Li and Shen. In the first section, the feature sequence is extracted from the input image by using the CNN. Then in the second section, BRNNs with LSTMs process the sequence labelling. Finally, CTC decodes the produced sequence into a character string. Source [18].

the process more stable than treating each feature independently. Therefore Bidirectional Recurrent Neural Networks (BRNNs) are applied to help capture the interdependencies between the features. There are two separated hidden layers in BRNNs. One of them processes the feature sequence forwards for the future context information, while the other one processes it backwards for the past context information. The LSTMs are also employed to store contexts for a long period of time and capturing long-range dependencies between the features.

Sequence labelling is then done by recurrently implementing BLSTMs for each feature in the feature sequence. The state of the current feature is updated with both its own state and the state of the past or the future neighbour feature. After, the soft-max layer transforms LSTMs' states into a sequence of probability distribution over 37 classes (26 upper-case letters, 10 digits, and 1 non-character class). Subsequently, the whole feature sequence is transformed into a sequence of probability estimations. Finally, a Connectionist Temporal Classification (CTC) [12] is applied to the output layer of RNN. It decodes the prediction sequence into a character string.

This segmentation-free method was the first of its kind. The experimental results have proven higher accuracy than methods that use character segmentation. It is robust to various illumination, rotations, and distortions in the image. However, it is only focused on standard high-quality images of the license plates.

2.3.4 Holistic Recognition of Low Quality License Plates by CNN

Holistic Recognition of Low Quality License Plates by CNN using Track Annotated Data was published by Adam Herout et al. [31].

This method focuses on a holistic approach, meaning it perceives the license plate as a whole. The architecture of this solution is designed for maximum of 8 characters per license plate. It uses one CNN to process the whole RGB image. The last convolutional layer is connected to 8 branches of fully connected layers, each with 36 outputs. This way every branch predicts one character on the same position in the license plate text every time. Figure 2.13 demonstrates where the fully connected layers search for their respective characters. It represents the mean of weights in the first fully connected layer which corresponds to different spatial locations. The 4^{th} character does not have a clearly given blob since in majority of training license plate texts was the 4^{th} spot designated as the blank fill character.

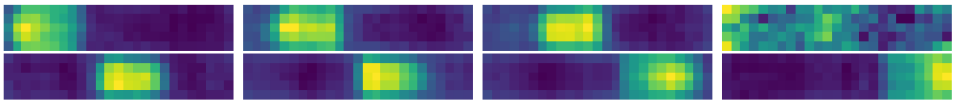

Figure 2.13: The demonstration of where the fully connected layers search for their respective characters. Left to right, top to bottom: 1^{st} to 8^{th} character. In most cases in the training data, the 4^{th} character is a blank fill character. Source [31].

The solution proposed in this work can also deal with license plates having less than 8 characters. It simply fills the missing characters with the blank fill character („#“) during the training phase. However, it is limited by the highest number of characters that the net is trained for. Moreover, it is unable to recognize multi-line license plates as it expects all characters to be in one line. Nevertheless evaluation results on multiple datasets prove that this method greatly surpasses other commercial solutions for low quality license plate recognition.

Chapter 3

License Plate Datasets

Multiple annotated datasets were acquired for the purposes of this work. Ideally, they should all simulate real world conditions. This means that the license plates should be blurred, partially obscured, distorted, vary in lighting conditions, etc.

On top of that, both the license plate detector and license plate recognizer require different kinds of images and annotations. The detector not only needs the license plate image but also a significant amount of background around it. The dataset should also contain the annotations of coordinates of the bounding box corners. On the other hand, the recognizer works with only already cropped license plate images and their annotations in form of ground truth labels.

Details on both kinds of datasets for license plate detection 3.1 and license plate recognition 3.2 are provided below.

3.1 Dataset for License Plate Detection

One dataset for license plate detection with manually verified bounding box annotations was provided by my supervisor Ing. Juránek, Ph.D. The dataset was recorded from seven different locations by surveillance cameras placed above the roads under different light conditions. It contains 1013 images each with 1920×1080 resolution. All images were captured from the frontal view of the vehicle thus every license plate is more or less parallel with the bottom border of the image. The majority of license plates come from the Czech Republic with a standard European design. There are always one or more license plates in an image. Random samples of the datasets are shown in Figure 3.1.

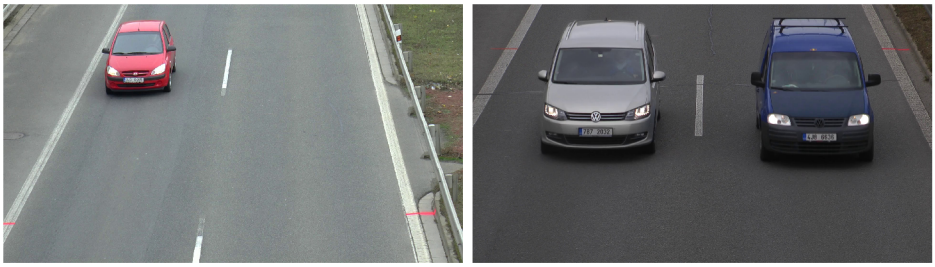

Figure 3.1: Random samples of the license plate detection dataset. Note; the license plates are not distorted and there are different light conditions.

Following a good splitting practice, I have decided to divide the dataset into the training and testing parts in a ratio of 80/20. This proportion is based on the Pareto Principle which is also known as 80/20 rule. It states that roughly 80 % of the effects come from 20 % of the causes. This results in 809 images for the training and 204 for the testing. Note that the seven sets of images from different areas were split independently to prevent the data being imbalanced.

3.2 Datasets for License Plate Recognition

Two novel real world (non-synthetic) user-annotated datasets of low quality images ReId and HDR, previously collected and used by Adam Herout et al. for their research [31], were acquired for the license plate recognition. The following sections describe each dataset acquisition 3.2.1 as well as datasets specifics, ground truth assignment, and division into training and testing parts 3.2.2.

3.2.1 Datasets Acquisition

As stated in the paper [31] „Multiple videos from various locations and under different conditions were recorded for ReId dataset. The data was captured by Full-HD video cameras placed on bridges above the highway, simulating surveillance cameras on toll gates. 9.5 hours of license plate recordings were collected in total from eight different locations during various times of the day. Samples of recorded data are shown in Figure 3.2. License plates were then automatically detected using a pretrained detector.

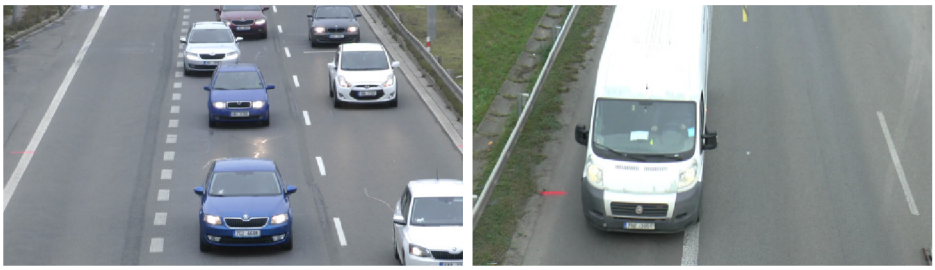

Figure 3.2: Samples of recordings from different camera locations. Source [31].

HDR dataset was captured by Digital Single Lens Reflex (DSLR) camera with three different exposures. License plates were hand-cropped from the images. Samples from HDR dataset are shown in Figure 3.3“.

Figure 3.3: Randomly selected samples from the HDR dataset. Source [31].

3.2.2 Datasets Specifics and Ground Truth Assignment

The ReId dataset consists of user-annotated tracks (a sequence of the same license plate in the video) that can be seen in Figure 3.4. This enables a person to annotate the tracks with at least one human-readable license plate therefore, it provides ground truth labels for the rest of the images on the track which could be naturally blurred, small, partially covered, or hardly readable in general. This is very convenient for building a very robust license plate recognizer. The number of characters on the license plates ranges from 5 to 8 in both datasets. Note that the license plates are only European.

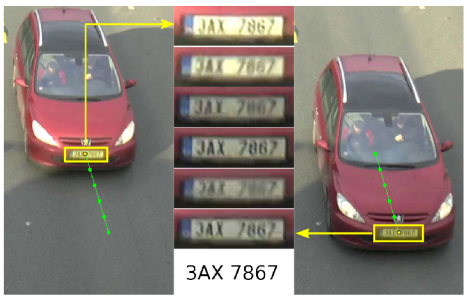

Figure 3.4: An example of license plate track. The whole track can be annotated as long as at least one of the license plates is human-readable. Source [31].

Following an already existing distribution of subsets proposed in [31], the ReId dataset is split into training and test parts per video. This guarantees an appropriate evaluation because the training and testing samples come from mutually exclusive sets of videos. The training part contains 7,393 tracks (105,924 images) and the testing part contains 6,967 tracks (76,412 images). The HDR dataset is used exclusively for evaluation. It does not contain tracks but only a total of 652 user-annotated images. It is worth noting that some of these images contain rotated license plates which differ from the ones used for training.

Chapter 4

Design Proposal for the Real-Time Traffic Surveillance Tool

The goal of this work is to design, implement and test a system that would be able to successfully detect and recognize license plates from video recordings from a stationary camera in real-time by using neural networks. The following sections provide a system's architecture design 4.1 along with an assessment of the current state of the art and propose a draft for the license plate detection 4.2, tracking 4.3, and recognition 4.4.

4.1 System's Architecture

The system will be designed to make use of the TensorFlow 1.15.2 GPU version. The TensorFlow is an end-to-end open source platform for machine learning written in Python, C++ and CUDA and it can run on various platforms. The final application should work on Ubuntu 16.04 or later (64-bit) and Windows 7 or later (64-bit). The code will be written in Python 3.6. The architecture of the system is illustrated in Figure 4.3. The system will be composed out of three main parts:

- License Plate Detector
- License Plate Tracker
- License Plate Recognizer

Initially, the input video stream will be loaded from the provided program argument in form of a file path or a link to an IP webcam. Next, every n^{th} frame from the input feed will be extracted, resized to 400×225 pixels and passed to the detector. The detector will then find possible license plate locations with a certain confidence and output an array where each element will compose a $[\mathbf{x_{min}}, \mathbf{y_{min}}, \mathbf{x_{max}}, \mathbf{y_{max}}, \textbf{ confidence}]$; x_{min} and y_{min} are coordinates of the upper left bounding box corner and x_{max} with y_{max} denote the lower right corner. The coordinates are relative to the image size meaning they will always have values ranging from 0 to 1. The *confidence* indicates how likely the object is classified as a license plate. The boxes with a confidence below a predefined threshold will be eliminated.

Then, such arrays will be forwarded to the tracker in order to monitor each plate throughout the scene. If the tracker assigns an ID to a license plate in the image, it will return a similar array with the ID in the last column. These outputs from the tracker will be used to form tracks **[ID, frame,** $[\mathbf{x_{min}}, \mathbf{y_{min}}, \mathbf{x_{max}}, \mathbf{y_{max}}]$**, confidence, timestamp, age]**.

The *timestamp* will use the hh:mm:ss dd/mm/yyyy format e.g. 09:45:35 20/04/2020. If there is another detection with a higher confidence of the same license plate, the track will be replaced with the more confident one. This ensures that the recognizer will potentially receive the best detections and it will not get overloaded with multiple images of the same object. The system will wait until the license plate leaves the scene, in the sense that the plate will not be spotted for several consecutive frames and the track will reach the maximum age. Then, the track will be stored in a queue for the recognizer. Note that the coordinates will be multiplied by the frame width and height to be converted in the absolute form and padding of 10 pixels will be added for better recognition results.

With each frame, the recognizer will check for the data in the queue. If successful, it will crop the frame by the given bounding box coordinates and run the inference. As a result, it will enrich the data by the recognized text and its probability.

Finally, the data will be saved into the database. The SQLite library will be used because it provides a lightweight disk-based database and does not require a separate server process. The database will consist of the following elements: **ID, text, confidence,** $\mathbf{x_{min}}, \mathbf{y_{min}}, \mathbf{x_{max}}, \mathbf{y_{max}}$**, timestamp, image**. The *ID* is a primary key and it is automatically generated by the database. Since it is not a good practise to save images to the database, the images are given an unique name and are saved into a separate folder. Therefore the database column *image* only refers to the images by their names.

The data from the database may be used to check whether a vehicle a with given license plate has passed the area that is covered by the surveillance camera and at what time it happend. It may also serve for statistical purposes such as calculating the average traffic flow and many others.

Figure 4.1: Most commonly used formats of license plates in Europe. Black on white (left). Black on yellow (right). Source [3].

Since all of the datasets only contain European license plates, the system is intended just for that selection. Although European license plates slightly differ in every country, it is safe to say that they have rectangular shape and most of them use the standard black on white or black on yellow format. Examples of such formats are shown in Figure 4.1. Typically, countries ban certain characters for their interchangeability. For Example in the Czech Republic letters G, CH, O, Q, and W are not allowed [1].

Figure 4.2: Examples of the blue zones on the European license plates. Source [3].

Every license plate issued in the member state of the European Union (EU) must have a reflecting blue zone on the extreme left with twelve stars representing the European flag and up to three characters distinguishing the state of registration [4]. Several non-EU states have developed a similar design with national flags or other symbols instead of the European flag. Examples of blue zones are illustrated in Figure 4.2.

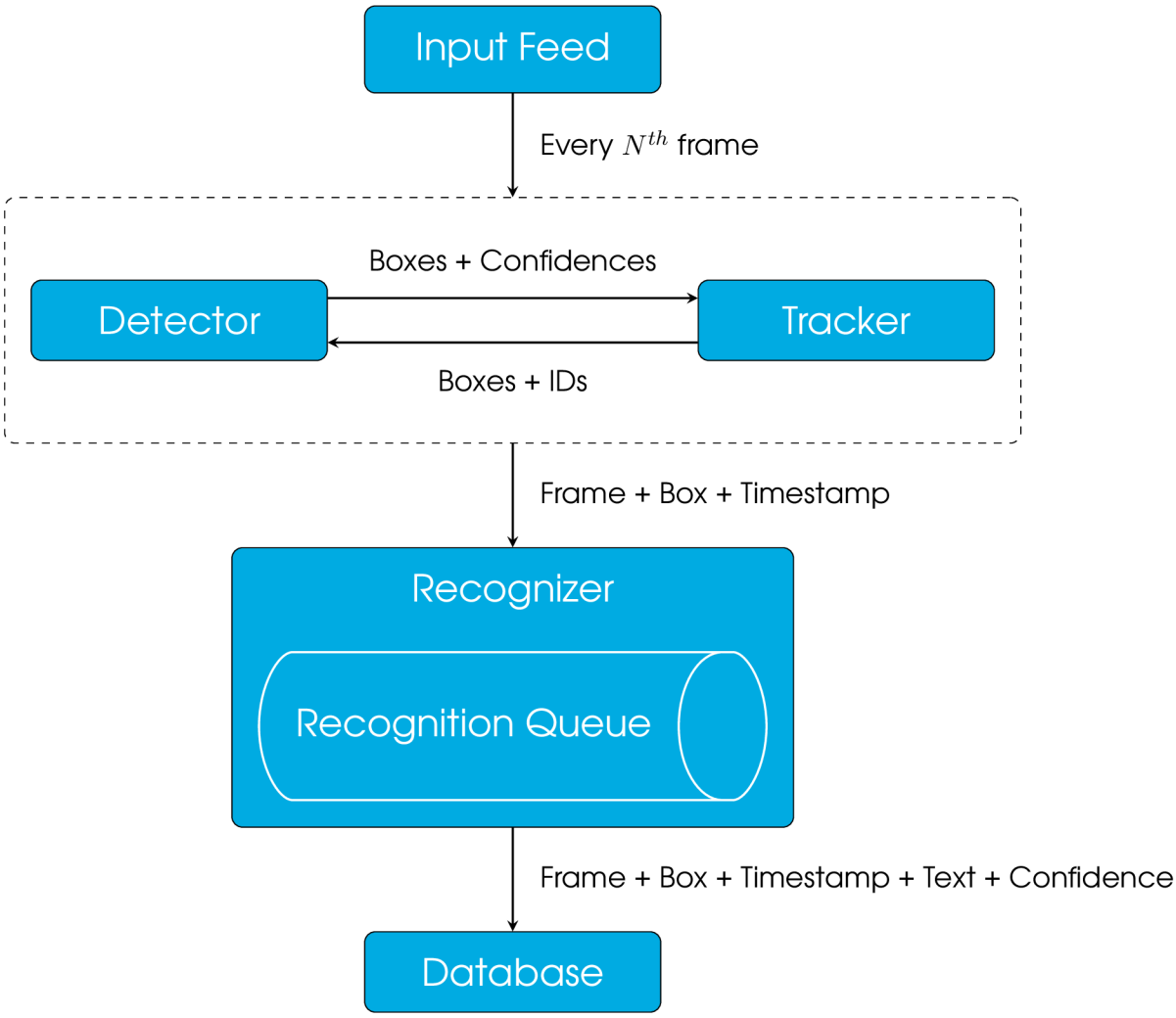

Figure 4.3: The proposed architecture of the system for a real-time traffic surveillance.

4.2 License Plate Detection

Over the years, object detection has been widely used and many diverse methods have been developed. These methods can be trained for detecting various desired object classes if they are supplied with a sufficient dataset. Every method has its pros and cons, usually it sacrifices speed over accuracy or vice versa.

Even though this work's detector is supposed to run in real-time, it is not necessary to capture every single frame of the video recording. To provide reasonably good input for the tracker and the recognizer, it is enough to extract just a few images per second. The goal is to aim for around 10 fps. Thanks to this the architecture can afford to use slower but more precise approaches while not suffering from the lack of the input data.

For the purposes of the license plate detection, I have decided to make use of the TensorFlow Object Detection API [13] and its source codes on GitHub[1]. The TensorFlow provides several object detection models (pre-trained classifiers with specific neural network architectures). For the sake of accuracy, I have chosen the Faster-RCNN-Resnet50 model.

The detector will be trained and tested on the dataset described in 3.1. The training will be done on the GPU for extra processing power and it will proceed until the loss will consistently move around 1.

Metrics from the latest Pascal VOC Challenge 2012 [5] will be used for the object detector evaluation. Individual detections are categorized based on the overlap with the ground truth bounding box into three groups:

- True Positive (TP): A correct detection. $IoU \geq 0.5$
- False Positive (FP): An incorrect detection. $IoU < 0.5$
- False Negative (FN): A ground truth bounding box is not detected.

If multiple detections of the same object are detected, it only counts one of them as a positive while the rest are negatives. The precision which identifies the system's ability to identify only the relevant objects is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

Note that the precision applies only for the objects that were found and does not take false negatives into consideration. The recall measures the success rate of the model to find all ground truth bounding boxes. It is given by:

$$Recall = \frac{TP}{TP + FN}$$

A good way to evaluate the object detector is to order the detections by their confidences, calculate the precision and recall and create a precision-recall curve 4.4. The area below the curve is defined as Average Precision (AP). Nevertheless, in practice the precision-recall curve is being interpolated across all data points. The intention is to reduce the impact of the small deviations within the curve. The interpolation process is shown in Figure 4.5. The resultant AP ranges from 0 to 1 and it will serve as a metric for evaluation.

As a result the license plate detector model should be capable of processing several images per second and extracting objects classified as license plates together with their

[1] https://github.com/tensorflow/models/tree/master/research/object_detection

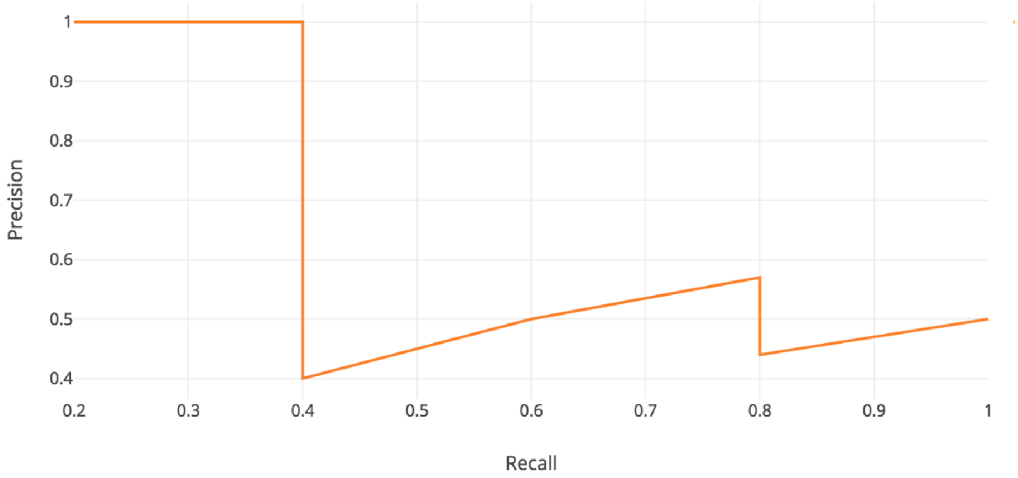

Figure 4.4: An exemplary precision-recall curve. The precision usually starts with high values and decreases while the recall increases. Source [14].

bounding box coordinates with certain confidence. By changing the confidence threshold the amount of detected objects considered as correct can be effected. There is room for experimenting but in general the aim is to try to achieve the highest AP possible.

4.3 License Plate Tracking

Due to the recent progress in object detection, tracking-by-detection has become the leading paradigm in multiple object tracking. With powerful state of the art object detectors, it is possible to employ object trackers where the detection quality strongly influences the tracking performance. The authors of SORT 2.2.2 have made an assertion that states modern object detection algorithms can do most of the work and object tracking can be reduced to simple heuristics.

I have chosen to use SORT for several different reasons. It is designed for real-time applications and is therefore capable of achieving a high processing speed. Moreover, it omits handling issues such as identity switches or object re-identifications which should rarely happen when surveilling traffic with a stationary camera.

After the detection is done, the object tracker will receive an array of bounding boxes with their respective confidences. As a result, it will output a similar array with assigned IDs. However, the aim is to select only one instance per plate throughout the scene with the highest confidence. In order to do this, the tracker will keep a list of currently tracked objects which will contain [**ID, frame,** $\mathbf{x_{min}}, \mathbf{y_{min}}, \mathbf{x_{max}}, \mathbf{y_{max}}$**, confidence, timestamp, age**]. The age symbolizes the number of frames the object was lastly assigned. If the same object is spotted multiple times and if the new occurrence has a higher confidence, the list will be updated; otherwise it will be ignored. After the object is no longer spotted for T_{Lost} frames, it will be removed from the list and it will get added to the queue waiting for the recognition. Note that the *ID*, *confidence*, and *age* is no longer needed so it will be excluded.

T_{Lost} will be set to 1, which is the same as in the other conducted experiments in [2]. IoU_{min} parameter is subject to change. The authors of the SORT used 0.3. Nevertheless it is apparent that it can be set to a lower value, approximately around 0.1, since license

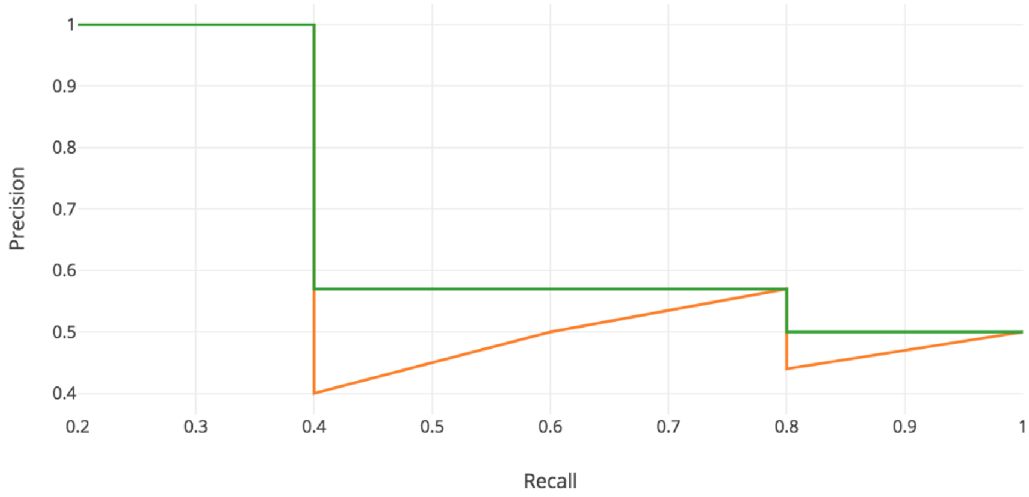

Figure 4.5: The green line represents an interpolated precision-recall curve while the orange line is the original one. The interpolated precision values are obtained by taking the maximum precision of those whose recall value is greater than its current recall value. Source [14].

plates do not tend to cluster together as much as other objects. This decision should help robustly track individual objects and thus prevent perceiving the same object as a new one after losing its track.

4.4 License Plate Recognition

The first license plate recognition method was employed in 1976. Since then, many research groups have proposed their own methods to improve the state of the art at that time. Nowadays many solutions typically depend on character segmentation. However, this may be the accuracy bottleneck as improper segmentation caused by many different challenging factors makes the rest of the recognition process impossible.

It has been proven by Goodfellow et al. [11] that CNNs with segmentation-free approaches are capable of multi-character text recognition. Motivated by this fact and inspired by [18], I have chosen to determine this stage of the system using a holistic approach.

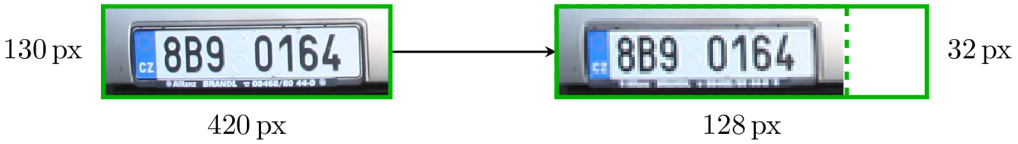

Figure 4.6: The cropped license plates will be resized to 128×32 pixels without any distortion. In another words, the image will keep its aspect ratio and any extra space will be filled with white pixels.

The system will consist of 5 CNN layers, 2 RNN (LSTM) layers, and the CTC loss and decoding layer. First, the license plate will be cropped from the input image and will be resized to 128×32 pixels without any distortion. Any extra space will be filled with white pixels as shown in Figure 4.6. The image will then be turned into a grayscale and fed to the

CNN layers to extract its features. Consequently, the LSTM implementation of RNN will be used to propagate relevant information through the sequence. Finally, the CTC layer will compute the loss value during the training or it will decode the output of the RNN layers into the final text.

The system will be implemented by using TensorFlow 1.15.2 and it will run on the GPU. The training and testing is going to be performed on ReId and HDR datasets as described in 3.2.2. Additionally, 5 % of the training part will be excluded for the validation purposes. After each epoch of training, the validation will be done and if there are no improvements in the character error rate in 5 consecutive epochs, the training will be terminated. The model will be evaluated based on the Character Error Rate (CER) and the Word Error Rate (WER).

$$CER = \frac{\text{character errors}}{\text{all characters}}$$

$$WER = \frac{\text{word errors}}{\text{all words}}$$

As a result, the recognizer should output several characters per image with a certain confidence representing the prediction of the license plate text.

Chapter 5

Implementation and Testing of the Real-Time Traffic Surveillance Tool

The training and the employment of the models require a powerful hardware and is very time intensive. In order to speed up the process, a cloud service for machine learning and research called Google Colaboratory[1] was utilized. It is a Jupyter notebook environment with the TensorFlow already pre-installed and optimized for the hardware being used. It provides two Intel Xeon CPUs at 2.30 GHz, NVIDIA Tesla P100-PCIE-16GB GPU, and 13 GB RAM.

The following sections describe the training and experimental results of the license plate detector 5.1 and recognizer 5.3. The license plate tracker implementation and its demonstration is summarized in section 5.2. Finally, the last chapter 5.4 covers topics including how to work with the tool and the necessary dependencies.

5.1 License Plate Detector Training and Experimental Results

Originally, the object detection was designed to make use of one of the TensorFlow pre-trained models Faster-RCNN-Resnet50. However, with the computational power available, it was not capable of achieving the desired speed. On average, it processed only 4 fps when working with videos of 720×480 or 1280×720 resolution. For this reason, another model from TensorFlow detection model zoo[2] called SSD-MobileNet-v2-COCO was selected. It is supposed to be three times faster yet it suffers from less accuracy especially when detecting small objects like the license plates.

The first step was to convert the dataset into the TFRecord format that is supported and recommended by the TensorFlow. TFRecord is a simple format for storing a sequence of binary records and it positively influences the training time.

Then the model was trained for ∼15,000 steps or 5.5 hours. It was terminated when the loss kept stagnating around the value 1 for the last 5,000 steps. Nonetheless, the results of the experiments executed on the testing part of the dataset came out unsatisfactory with the highest AP reaching only 88.63 %. Therefore, the model was retrained for a longer period of time on a newly created and augmented dataset.

[1] https://colab.research.google.com/notebooks/intro.ipynb

[2] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

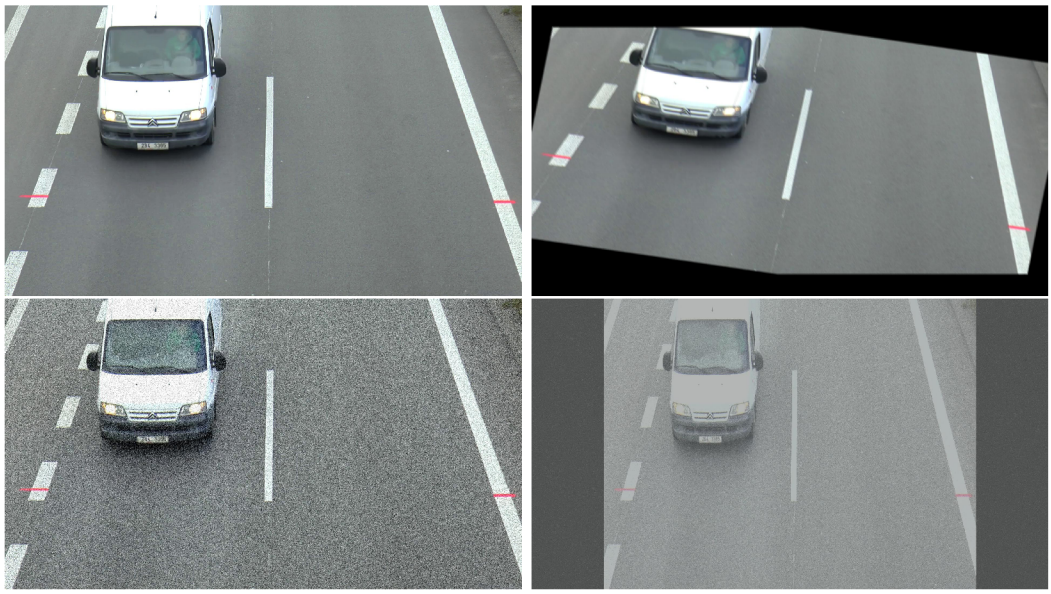

Figure 5.1: Random examples of the images after the dataset augmentation. The top left image is the original one.

The augmentation was performed on every image from the training part of the dataset via the python library imgaug[3]. Each image was modified by 1 to 9 transformations in a random order.

- Rotation from $-10\,^\circ$ to $10\,^\circ$
- Shear of the x axis from $-20\,^\circ$ to $20\,^\circ$
- Shear of the y axis from $-20\,^\circ$ to $20\,^\circ$
- Scale of the x axis from 0.5 to 1
- Scale of the y axis from 0.5 to 1
- Adding the Gaussian noise with the standard deviation of the normal distribution that generates the noise ranging from 10 to 60
- Brightness multiplication from 0.6 to 1.5
- Contrast with low severity i.e. low strength of the corruption
- Motion blur using a kernel of size 10

Each image resulted in 19 slightly modified new images as well as their corresponding bounding boxes. These images have kept their original resolution therefore if scaling, shearing, or rotation were applied, the empty spaces would be filled with black pixels. Additionally, the overlaying parts along with the bounding boxes would be clipped. Random examples are shown in Figure 5.1.

The second training on the augmented dataset lasted for ∼44,000 steps or 17.5 hours. The course of the training was observed via a Tensorboard. It is a TensorFlow's toolkit

[3] https://imgaug.readthedocs.io/en/latest/

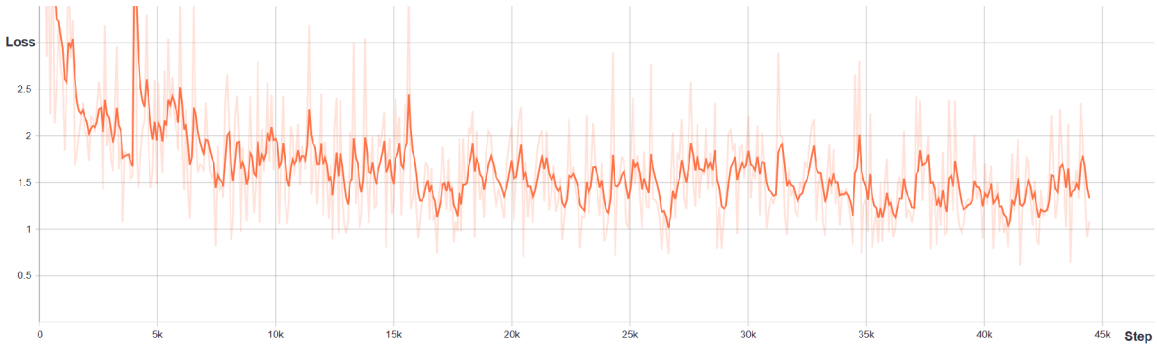

Figure 5.2: A graph showing the loss during the training on the augmented dataset. The light orange shows the original values. The dark orange represents the values after the smoothing (exponentially weighted moving average) with weight = 0.7 was applied.

for visualization of machine learning experimentation. A graph of the loss provided by Tensorboard is shown in Figure 5.2.

Different confidence thresholds were evaluated on the testing part of the dataset as shown in Table 5.1. The highest AP achieved was 98.15 % while having the confidence threshold set to 0.1, 0.2, or 0.3. The threshold 0.1 was chosen for this work as it seems like the AP increases when the confidence threshold decreases. Moreover, even the less accurate detections can be helpful for the license plate tracker. The precision-recall curve for the threshold 0.1 is illustrated in Figure 5.3.

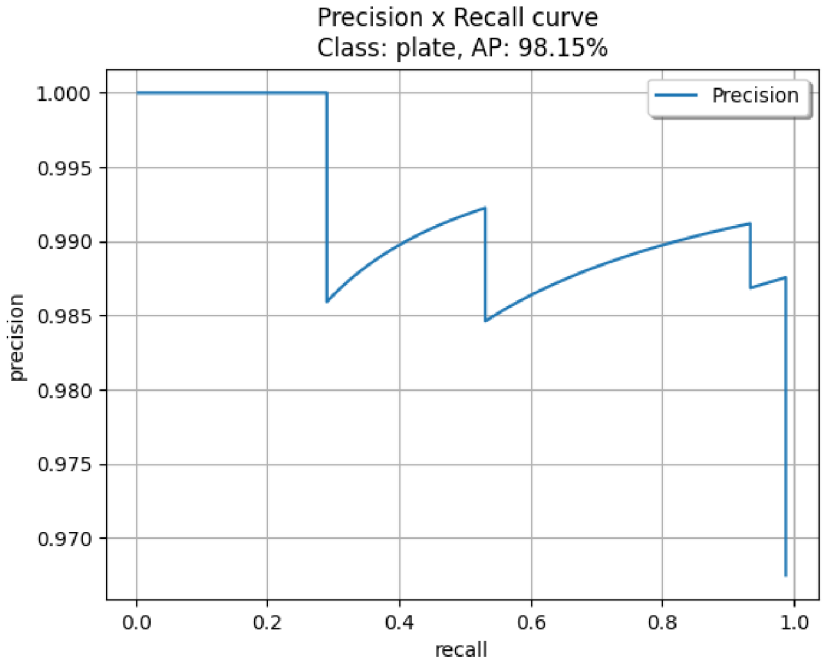

Figure 5.3: The precision-recall curve for the evaluation of the license plate detector. The experiments were conducted on the detections with the confidence value higher than 0.1. The recall, which is an important factor for the license plate tracker, reached 0.99.

confidence threshold	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1^{st} training AP (%)	88.63	88.63	87.87	87.10	85.18	83.26	80.96	79.80
2^{nd} training AP (%)	98.15	98.15	98.15	97.74	96.51	94.05	91.99	89.10

Table 5.1: The results of the experiments executed on the training part of the object detection dataset. The first training was performed on the original dataset while the other training was performed on the augmented one. The AP grows as the confidence threshold decreases until it reaches the point where there are no more detections with less confidence than the previous higher threshold.

Given the Google Colaboratory computational power, the detector is able to process a video with resolution 1280×720 with a speed of roughly 14 fps. When working with smaller videos with resolution 720×480 the detector reaches around 15 fps. The output of the detector is illustrated in Figure 5.4.

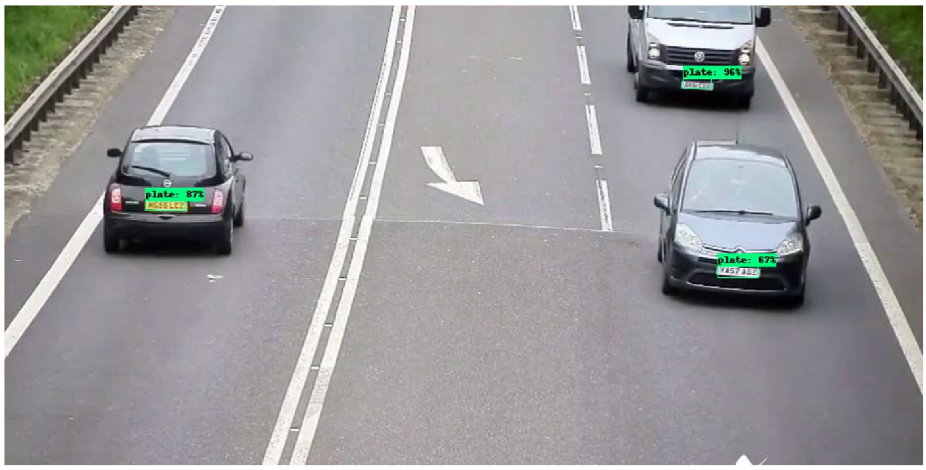

Figure 5.4: The visualized output of the detector — predicted bounding boxes with their respective confidences.

5.2 License Plate Tracker Implementation and Demonstration

The license plater tracker used the original implementation of SORT that was written in Python by its creators[4]. Nevertheless, there were a couple of adaptations that needed to be done.

Due to its nature, the SORT performs poorly if it works with small bounding boxes, as opposed to the larger boxes. Only a small deviation in the prediction can result in a denial of the assignment. Therefore, the coordinates of the bounding boxes get expanded before they are fed to the tracker and then the reverse process is used to retrieve the original coordinates back. Figure 5.5 shows the difference between the normal boxes and the expanded ones. The width of the detection gets stretched by three times and is distributed equally on both sides. The height gets enlarged by nine times and is distributed upwards and downwards

[4] https://github.com/abewley/sort

in a ratio of 5:1. It covers more of the upper section above the license plate because it will very likely only overlap with the rest of the vehicle and it will not collide with the other detections.

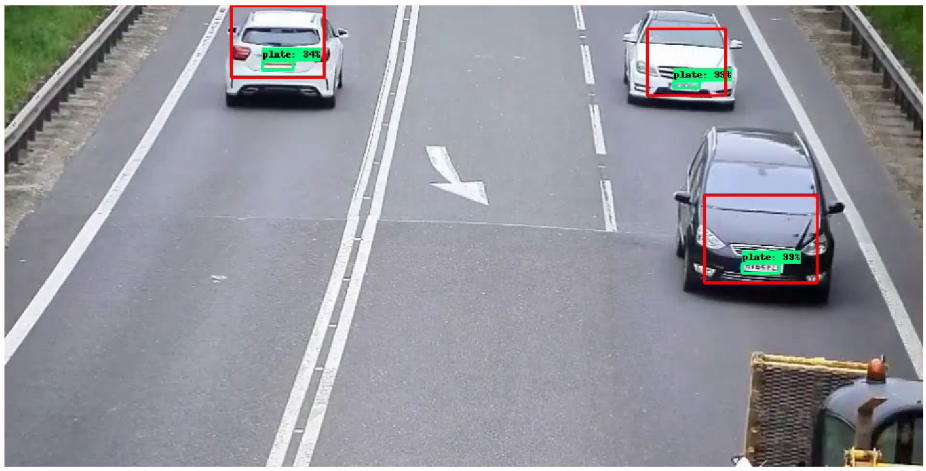

Figure 5.5: The green bounding boxes are the original detections. The red bounding boxes are expanded for better tracking performance. The expansion grows mainly upwards since it will mostly likely cover only the rest of the vehicle and it will not collide with the other bounding boxes.

Another issue is the detector may not always be correct. The position of the detection can be either a little off or the license plate may not get spotted even up to several times in a row. This can lead to loosing track of the license plate and possible re-identification later but with a different ID. To prevent this, the T_{Lost} was raised to 5, meaning the ID has 5 frames to get assigned again before the track gets terminated. However, the predictions from the Kalman Filter get less accurate with each frame that has no detections and they can become difficult to assign. For this reason, the IoU_{min} was set to 0.1 to tolerate greater mistakes accumulated over time. The impact of the $IoU_{min} = 0.1$ is illustrated in Figure 5.6.

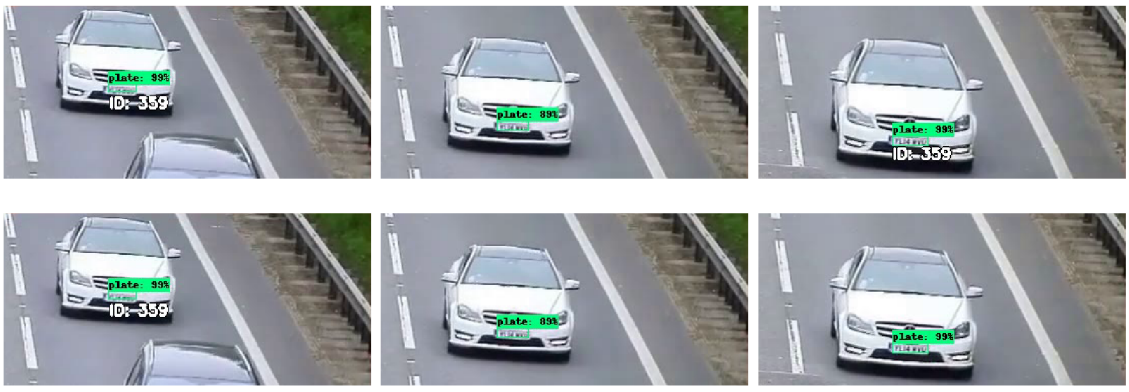

Figure 5.6: The upper line shows the results for $IoU_{min} = 0.1$; the bottom line for $IoU_{min} = 0.3$. The video was processed by only 3 fps to induce the inaccurate predictions. **Left:** The license plate gets assigned an ID (359). **Center:** The ID is not assigned due to the poor prediction. **Right:** The same ID is again assigned to the license plate but only with the lower IoU_{min}.

5.3 License Plate Recognizer Training and Experimental Results

The license plate recognizer training and implementation was adapted from the Handwritten Text Recognition (HTR) system created by Harald Scheidl [27] and its publicly shared source codes[5]. It is expected that the dataset is compatible with the IAM dataset[6], therefore the first step was to convert the ReId and HDR datasets 3.2.2 into the required format.

Then the model was trained in the Google Colaboratory environment for 6 epochs or 21 hours. However, the training pace was unsatisfactory due to the large amount of files being slowly read from the Google Drive. For that reason, the training was resumed on the local machine with Intel Core i5-6198DU CPU at 2.30 GHz, NVIDIA GeForce 930MX GPU, and 8 GB RAM. Despite the worse hardware specifications, the overall training speed improved by more than 4.5 times. The training continued from the last checkpoint for another 28 epochs or 21 hours. It was executed after 5 consecutive epochs with no improvement while the loss was consistently moving around 0.1.

The experimental results of the recognizer are shown in Table 5.2. The recognizer is able to correctly infer alphanumerical characters 0123456789ABCDEFGHIJKLMNPQRSTUVWXYZ. With the previously mentioned hardware setup, it can recognize 39 frames per second with the average size of 9.5 kilopixel or 46 frames per second with the average size of 5 kilopixel respectively.

	Character Error Rate (CER)	Word Error Rate (WER)
ReId validation subset	0.846581 %	4.4 %
ReId testing subset	0.584667 %	2.019343 %
HDR dataset	8.146684 %	21.47239 %

Table 5.2: The experimental results of the trained license plate recognizer.

5.4 The Application Usage

There are several dependencies that must be installed before running the application. All of the required modules can be found in the *requirements.txt* file. The application is called *alpr.py* and has the following synopsis:

```
alpr.py [-h] -s <source> -d <destination> (-i <img_dir_destination> | -v)
```

Table 5.3 describes the program arguments. It can run in two different modes. When the `-v` switch is applied, outputs of the detector and recognizer are rendered into the frames which are used to create a new exemplary video. The result of this mode can be seen in Figure 5.7. Although the tracking is turned off because it is no longer necessary, this mode does not fulfil the real-time characteristics due to the need to process every single frame. The second mode is the default one that has been reviewed before; it runs in real-time and saves the data into the database.

Standard output information such as the recognized license plate text, its probability and the timestamp are written to the stdout. If an error occurs, the exception is raised and the error is emitted as a command-line message to the stderr.

[5] https://github.com/githubharald/SimpleHTR

[6] http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

Argument	Description
-h, --help	Shows the help message and exits
-s, --src	The source video stream
-d, --dest	The database or the output video destination
-i, --img_dir_dest	The image directory destination
-v, --video_output	The program will produce a video output

Table 5.3: The description of the *alpr.py* arguments.

Figure 5.7: An example of the video output mode showing the bounding boxes and license plate texts.

Chapter 6

Conclusion

The goal of this Bachelor's thesis was to design, implement, and test a system that would be able to detect and recognize license plates in real-time by using neural networks. The collected data should be saved to a database or used to produce a new video for the demonstration purposes. Such a system was developed in Python and can be employed to process the offline videos as well as the live webcams. A video showing the detections and recognized license plate texts was created to illustrate the program's performance.

Several methods were examined while considering the most suitable ones for this topic. Ultimately, The Single Shot MultiBox Detector (SSD) method was chosen for the license plate detection. The model was trained by utilizing the TensorFlow Object Detection API. It reaches the accuracy of 98.15 % AP with a speed of roughly 14 fps when working with a video with a resolution of 1280×720 or 15 fps when the resolution is 720×480.

Simple Online and Real-Time Tracking (SORT) was selected for tracking the license plates in the video due to its capability of achieving high processing speeds. SORT allows it to effectively filter the data to prevent redundancy, thus drastically reducing the number of detections meant to be recognized later in the system.

The license plate recognition was implemented by following the segmentation-free approach by using deep Convolutional Neural Networks (CNN) and Long Short-Term Memories (LSTMs). The model was trained by using TensorFlow and it achieves up to 0.6% character error rate (CER) and 2% word error rate (WER). It manages to recognize 39 fps when given images with the average size of 9.5 kilopixel or 46 fps with the average size of 5 kilopixel respectively.

Three real world user-annotated datasets were acquired for the purposes of the license plate detector and recognizer training and testing. In total, the datasets contain 184,000 samples of the European license plates taken from multiple different locations and have various lightning conditions, quality, size, blur, distortion, noise, etc.

There are several possible improvements that could be done for the further advancement of the project. The detector was dedicated to the SSD approach due to its great speed. However, this method is weak when working with small objects such as license plates. By designing the individual parts of the system in parallel, it would be doable to employ more accurate but slower models and yet run the program in real-time. Another suggestion would be to adapt the application to collect the data from more than just one camera at a time. Each detection node would run in its own process or even on a different computer and the communication between the rest of the system would be solved by a message broker.

Bibliography

[1] *Updated questions and answers on the issuance of license plates on demand* [online]. 2016 [cit. 2020-3-4]. Available at: https://www.mdcr.cz/Media/Media-a-tiskove-zpravy/Aktualizovane-otazky-a-odpovedi-k-problematice-zna?lang=en-GB.

[2] Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, p. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[3] contributors, W. *Vehicle registration plates of Europe* [online]. 2020 [cit. 2020-2-4]. Available at: https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Europe.

[4] European Union, C. of the. *Council Regulation No 2411/98* [online]. 1998 [cit. 2020-2-4]. Available at: http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:31998R2411.

[5] Everingham, M., Eslami, S., Van Gool, L., Williams, C., Winn, J. et al. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*. Springer Nature. Jan 2014, vol. 111. DOI: 10.1007/s11263-014-0733-5.

[6] Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J. and Zisserman, A. The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*. 2010.

[7] Gandhi, R. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms* [online]. 2018 [cit. 2020-17-1]. Available at: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[8] Gao, H. *Faster R-CNN Explained* [online]. 2017 [cit. 2020-31-03]. Available at: https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8.

[9] Girshick, R. B. Fast R-CNN. *CoRR*. Apr 2015, abs/1504.08083. Available at: http://arxiv.org/abs/1504.08083.

[10] Girshick, R. B., Donahue, J., Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, abs/1311.2524. Available at: http://arxiv.org/abs/1311.2524.

[11] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S. and Shet, V. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *ArXiv e-prints*. Dec 2013, p. arXiv:1312.6082. Available at: https://arxiv.org/abs/1312.6082v1.

[12] Graves, A., Fernández, S., Gomez, F. and Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In: *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2006, p. 369–376. ICML '06. DOI: 10.1145/1143844.1143891. ISBN 1595933832. Available at: https://doi.org/10.1145/1143844.1143891.

[13] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A. et al. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*. 2016, abs/1611.10012. Available at: http://arxiv.org/abs/1611.10012.

[14] Hui, J. *MAP (mean Average Precision) for Object Detection* [online]. 2018 [cit. 2020-28-4]. Available at: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

[15] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*. 1960, vol. 82, Series D, p. 35–45.

[16] Kumar, D. *Demystifying Support Vector Machines* [online]. 2019 [cit. 2020-6-4]. Available at: https://towardsdatascience.com/demystifying-support-vector-machines-8453b39f7368.

[17] Kumar Parasuraman, S. P. SVM Based License Plate Recognition System. *IEEE International Conference on Computational Intelligence and Computing Research*. 2010.

[18] Li, H. and Shen, C. Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs. *CoRR*. 2016, abs/1601.05610. Available at: http://arxiv.org/abs/1601.05610.

[19] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E. et al. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, abs/1512.02325. Available at: http://arxiv.org/abs/1512.02325.

[20] Milan, A., Leal Taixé, L., Reid, I. D., Roth, S. and Schindler, K. MOT16: A Benchmark for Multi-Object Tracking. *ArXiv*. 2016, abs/1603.00831.

[21] Petr Cika, M. S. Detection and Recognition of License Plates of Czech Vehicles. *Elektrorevue*. Dec 2011, vol. 2, no. 4. ISSN 1213-1539.

[22] Poorvi Hedau, S. D. and Datar, A. Recognition of License Number Plate using a Template Matching Technique. *IJCET*. Sept/Oct 2017, vol. 7, no. 5, p. 1901–1905. ISSN 2347-5161. Available at: http://inpressco.com/recognition-of-license-number-plate-using-a-template-matching-technique/.

[23] Redmon, J., Divvala, S. K., Girshick, R. B. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, abs/1506.02640. Available at: http://arxiv.org/abs/1506.02640.

[24] Redmon, J. and Farhadi, A. *YOLO: Real-Time Object Detection* [online]. 2016 [cit. 2020-16-01]. Available at: https://pjreddie.com/darknet/yolov2/.

[25] Ren, S., He, K., Girshick, R. B. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, abs/1506.01497. Available at: http://arxiv.org/abs/1506.01497.

[26] Rey, J. *Faster R-CNN: Down the rabbit hole of modern object detection* [online]. 2018 [cit. 2020-31-03]. Available at: https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/.

[27] Scheidl, H. *Build a Handwritten Text Recognition System using TensorFlow* [online]. 2018 [cit. 2020-22-7]. Available at: https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5.

[28] Uijlings, J., Sande, K., Gevers, T. and Smeulders, A. Selective Search for Object Recognition. *International Journal of Computer Vision*. september 2013, vol. 104, p. 154–171. DOI: 10.1007/s11263-013-0620-5.

[29] Wen, Y., Lu, Y., Yan, J., Zhou, Z., von Deneen, K. M. et al. An Algorithm for License Plate Recognition Applied to Intelligent Transportation System. *IEEE Transactions on Intelligent Transportation Systems*. 2011, vol. 12, no. 3, p. 830–845.

[30] Wojke, N., Bewley, A. and Paulus, D. Simple Online and Realtime Tracking with a Deep Association Metric. *ArXiv preprint arXiv:1703.07402*. 2017.

[31] Špaňhel, J., Sochor, J., Juránek, R., Herout, A., Maršík, L. et al. Holistic recognition of low quality license plates by CNN using track annotated data. In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. Aug 2017, p. 1–6. DOI: 10.1109/AVSS.2017.8078501. ISSN null.