RIGA TECHNICAL UNIVERSITY
Faculty of Electronics and Telecommunications
Institute of Radio Electronics

Roberts Kadiķis
Doctoral Student of the Study Programme "Electronics"

EFFICIENT METHODS FOR DETECTION AND CHARACTERIZATION OF MOVING OBJECTS IN VIDEO

Doctoral Thesis

Scientific supervisor
Senior researcher Dr. sc. comp.
MODRIS GREITĀNS
Institute of Electronics and Computer Science

RTU Press
Riga 2018

A DOCTORAL THESIS SUBMITTED TO RIGA TECHNICAL UNIVERSITY IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF SCIENCE IN ENGINEERING

To be granted the scientific degree of Doctor of Engineering Sciences, the present Doctoral Thesis has been submitted for the defence at the open meeting of RTU Promotion Council on 19 April 2018 at the Faculty of Electronics and Telecommunications of Riga Technical University, 12 Azenes Street, Room 212.

OFFICIAL REVIEWERS

Professor Dr. habil. phys. Andris Ozols
Riga Technical University, Latvia

Professor Dr. sc. ing. Aleksandrs Grakovskis
Transport and Telecommunication Institute, Latvia

Professor Dr. sc. ing. Pēteris Grabusts
Rezekne Academy of Technologies, Latvia

CONFIRMATION

I confirm that this doctoral thesis, submitted for a degree in engineering at the Riga Technical University, is my own work. The doctoral thesis has not been submitted for a degree in any other university.

Roberts Kadiķis . . . . . (Signature)

Date: . . . . . .

The Doctoral Thesis has been written in English. It consists of an introduction, 5 chapters, Bibliography, 6 appendices, 37 figures, 4 tables; the total number of pages is 132. The Bibliography contains 111 titles.

ABSTRACT

In recent years, the computer vision field has had significant success on the object detection task in images. However, current state-of-the-art approaches are computationally demanding and not well suited for efficient processing of videos. This Thesis focuses on methods that efficiently detect moving objects in a video and can be scaled for such applications as highway or people monitoring using inexpensive devices with limited computational power.

The literature review discusses the different object detection methods and identifies which of the approaches are more accurate, which are robust in the changing environment, and which are computationally efficient. Based on the literature review, the Thesis develops novel video-based moving object detectors that process only a line of pixels in a frame.

The first developed method – Intervals on a Virtual Detection Line (IoVDL) – exceeds other existing efficient methods with an interval approach, which allows use of the detection line in more varied scenes than conventional region-based detectors. The versatility of the interval approach is further demonstrated by developing an extended IoVDL. Using several detection lines in a frame, the extended IoVDL is capable of tracking and characterizing objects, while being computationally efficient.

Another proposed method – Recurrent Neural Network-based Virtual Detection Line (RNN-VDL) – combines the efficiency of the detection line approach and the versatility of machine learning. This method requires specific training data, so this Thesis explores and develops novel data labeling methods.

The tests of developed and implemented methods include the comparison of vehicle counting by IoVDL and RNN-VDL, the measurement of classification accuracy of the extended IoVDL, and the measurement of the computational efficiency of the proposed methods. The versatility of RNN-VDL is tested by retraining the neural network-based method and using it for people counting in a video.

The Thesis proves four statements and, as a result, efficient and adaptable methods for detection of moving objects in a video are developed. It consists of 132 pages, 37 figures, 4 tables, 2 algorithms, 111 sources of literature and 6 appendices.

The Doctoral Thesis has been developed in the Institute of Electronics and Computer Science (EDI).

ANOTĀCIJA

Pēdējos gados datorredze ir guvusi būtiskus panākumus objektu atklāšanas uzdevumā attēlos. Tomēr, šī brīža labākie paņēmieni pieprasa būtiskus skaitļošanas resursus, tādēļ to pielietojums reāla laika video apstrādē ir ierobežots. Šis promocijas darbs pievēršas metodēm, kuras efektīvi atklāj kustīgus objektus video, un kuras var īstenot ar lētām un pēc skaitļošanas jaudas ierobežotām iekārtām tādos liela mēroga uzdevumos kā lielceļu un cilvēku monitorings.

Darbā veiktā literatūras analīze apskata dažādas objektu atklāšanas metodes, nosakot, kuras metodes dod precīzākus rezultātus, kuras ir robustas pret mainīgiem apstākļiem, un kuras ir efektīvas patērēto skaitļošanas resursu ziņā. Balstoties uz literatūras analīzi, tiek izveidotas jaunas video apstrādes metodes kustīgu objektu atklāšanai.

Pirmā izstrādātā efektīvā metode – intervāli uz virtuālas atklāšanas līnijas IoVDL (Intervals on a Virtual Detection Line) – apstrādā tikai to daļu no kadra pikseļiem, kas atrodas uz virtuālas atklāšanas līnijas. Salīdzinot ar eksistējošām efektīvām metodēm, kas apstrādā tikai kadra reģionu, piedāvātā intervālu pieeja ļauj IoVDL pielietot plašākam uzdevumu lokam. IoVDL daudzpusība tiek vēl vairāk parādīta ar paplašinātās IoVDL izveidi. Izvietojot kadrā vairākas atklāšanas līnijas, paplašinātā IoVDL metode spēj izsekot un raksturot kustīgus objektus, joprojām paliekot efektīva skaitļošanas resursu ziņā.

Cita izveidotā metode – uz rekurentā neironu tīkla balstīta virtuāla atklāšanas līnija RNN-VDL (Recurrent Neural Network-based Virtual Detection Line) – apvieno atklāšanas līnijas efektivitāti ar mašīnmācīšanās pieeju, kas padara metodi apmācāmu dažādu objektu atklāšanai. Piedāvātā metode pieprasa specifiskus apmācības datus, tādēļ daļa darba ir veltīta datu marķēšanas pieeju izpētei un jaunu pieeju izveidei.

Piedāvātās atklāšanas metodes tiek īstenotas un eksperimentāli testētas, veicot gan precizitātes, gan skaitļošanas efektivitātes mērījumus. IoVDL un RNN-VDL tiek salīdzinātas uz automašīnu skaitīšanas uzdevuma. Paplašinātā IoVDL tiek testēta uz automašīnu klasificēšanas uzdevuma. Plašā RNN-VDL pielietojamība tiek testēta to pārtrenējot un pielietojot cilvēku skaitīšanas uzdevumam.

Disertācijā tiek pierādītas četras tēzes. Darbā ir 132. lpp., 37 attēli, 4 tabulas, 2 algoritmi, 111 izmantotie literatūras avoti un 6 pielikumi.

Darbs ir izstrādāts Elektronikas un datorzinātņu institūtā (EDI).

ACKNOWLEDGEMENTS

I thank my supervisor Modris Greitāns for all the good advice and the opportunity to develop this Doctoral Thesis at EDI.

I am grateful to Kārlis Freivalds, who consulted and taught me on different aspects of the research – the development of new ideas, implementation of algorithms in code, and writing of scientific papers.

There are many more people to thank, who helped the development of this thesis in indirect ways.

My parents Edgars and Natālija, for nurturing my curiosity and ambition.

My cousin Elena and the family of my sister Ērika for coloring the days of the writing.

Finally, all my friends and colleagues (including but not limited to Kaspars, Atis, Krišjānis, Rolands, Ričards, Armands, and many others) for making the completion of Thesis the only viable option.

Thank you!

Roberts Kadiķis,
Riga 2017

This Thesis is the result of the research carried out at the Institute of Electronics and Computer Science within the framework of the National Research Program "Cyber-physical systems, ontologies and biophotonics for safe&smart city and society" project No. 4: "Development of technologies for secure and reliable smart-city".

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

2D – 2 Dimensional
3D – 3 Dimensional
ANN – Artificial Neural Network
BPTT – Backpropagation Trough Time
BRIEF – Binary Robust Independent Elementary Features
BRNN – Bidirectional Recurrent Neural Networks
CNN – Convolutional Neural Network
CPU – Central Processing Unit
CV – Computer Vision
DL – Deep Learning
DoG – Difference of Gaussians
EDI – Institute of Electronics and Computer Science
EGFP – Enhanced Green Fluorescent Protein
FAST – Features from Accelerated Segment Test
FCN – Fully Convolutional Network
FN – False Negative
FP – False Positive
FPS – Frames Per Second
GPU – Graphich Processing Unit
GUI – Graphical User Interface
IDL – Inductive Detector Loop
ILSVRC – ImageNet Large Scale Visual Recognition Challenge
IoVDL – Intervals on Virtual Detection Line
ITS – Intelligent Transportation Systems
LBP – Locally Binary Patterns
LSTM – Long Short-Term Memory
MSER – Maximally Stable Extremal Regions
MTBF – Mean Time Between Failure
ORB – Oriented FAST and rotated BRIEF
PASCAL – Pattern Analysis, statistical modelling and computational learning
PCA – Principal Component Analysis
PTD – Pneumatic Tube Detector
RAM – Random-Access Memory
rBRIEF – rotated BRIEF
R-CNN – Regions with CNN features
RGB – Red, Green, Blue

RGB-D – Red, Green, Blue, Depth
ReLU – Rectified Linear Unit
RNN – Recurrent Neural Network
RNN-VDL – Recurrent Neural Network-based Virtual Detection Line
ROI – Region of Interest
RPN – Region Proposal Network
SIFT – Scale Invariant Feature Transform
SURF – Speeded Up Robust Features
SVM – Support Vector Machine
TBPTT – Truncated Backpropagation Through Time
TIRTL – The Infra-Ref Traffic Logger
TP – True Positive
VOC – Visual Object Classes
WIM – Weight in Motion
YCrCb – Luminance, Chrominance (red-yellow), Chrominance (blue-yellow)
YOLO – You Only Look Once
YOLOv2 – You Only Look Once version 2

NOMENCLATURE

A – activation value (output) of a neuron
B – bias value
$\mathbb{B}$ – storage buffer
b – bias vector
C – width in the image plane (number of columns)
D – distance
E – adaptation (update) rate
F_1 – F-score
G – magnitude of a gradient
G – matrix of magnitudes of pixel gradients
H – height in the world coordinates
H – 2D response to the Haar wavelet
h – vector of hidden node values
I – intensity value of the pixel
I – grayscale image (2D matrix)
i – index
J – intensity centroid
K – kernel (mask)
k – a constant
L – ownership parameter
l – detection line
M – a 2D matrix
M – mean value
N – a number of items
n – output of a node in an LSTM cell
O – orientation
$\mathbb{O}$ – virtual object described by a rectangle in a frame
$\mathbf{O}_g$ – matrix of gradient orientations
P – scale of a Gaussian
P^2 – variancei
P – covariance matrix
Q – cornerness measure
R – height in the image plane (number of rows)
S – length of a sequence (number of steps) considered by RNN
s – pixel's spectrum in a multispectral or hyperspectral image
T – threshold value

t – current time step (number of the current frame)
V - width in the world coordinates
W – weight value
W – matrix of weights
X – x coordinate (number of the row)
x – input vector
Y – y coordinate (number of the column)
y – target vector
$\hat{\mathbf{y}}$ – predicted vector (output of a model)
Z – confidence parameter
$\mathbb{Z}$ – set of integers
$\alpha(x)$ – activation function
$\beta(x, y, P)$ – difference of Gaussian function
$\gamma(x, y, P)$ – Gaussian function (Gaussian)
$\delta(x, y)$ – distance function
$\eta(x)$ – number of pixels with value x
$\rho(x)$ – linear rectifier
$\rho_{\mathrm{l}}(x)$ – leaky linear rectifier
$\sigma(x)$ – sigmoid function
$\tau(x, y)$ – test of pixel pair in ORB method
$\lambda(x)$ – probability of the pixel value
$\phi(x)$ – hyperbolic tangent (*tanh*)

INTRODUCTION

Our brain consumes more energy than any other human organ. One of the largest systems in the brain is the visual cortex. In non-human primates, half of the neocortex is devoted to visual processing [1]. The demand for such resources indicates that processing and recognition of visual information is a computationally complex and important task for humans and other animals. Successful visual processing allows us to understand the surrounding environment and successfully interact with it. These abilities are also crucial for making useful cyber-physical systems.

In this Thesis, a video is defined as a sequence of digital images (frames). The methods of interest acquire information about the real world by processing the input video. It is a task of a computer vision (CV) field, which is used in many practical applications, including:

- automation (control of industrial robots [2], assembly line supervision [3]);
- health (analysis of biomedical images for diagnosis [4], drug discovery [5]);
- security (face recognition [6], smart surveillance systems [7]);
- transportation systems (detection of vehicles [8], license plate recognition [9]).

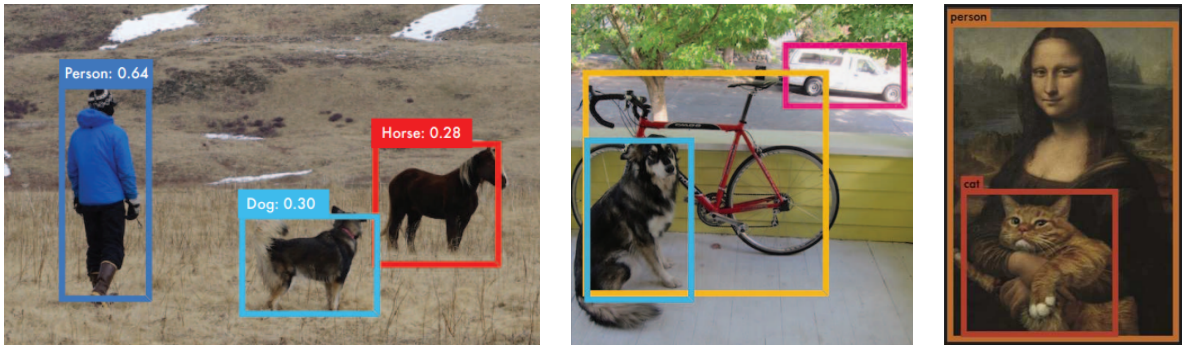
The capability of computer vision continues to grow and so do the applications of CV. It seems that artificial vision approaches the needed reliability for self-driving cars [10] and human level image classification [11].

In this Thesis, the efficiency of the method relates to its computational demands. One method is more efficient than another if it consumes less computational resources to process the same amount of input data. At the time of writing this Thesis, Moore's law is still relevant. It means that ever more complicated vision algorithms can be implemented on ever smaller devices. However, the field of CV cannot rely only on the growing computational power. New and more efficient algorithms can speed up the spread of computer vision by enabling the existing size-limited devices to use modern CV. For example, today's smartphones are capable of automatically detecting faces while taking photographs. Also, for gaming purposes, these phones can augment the reality seen by the camera in real time. Future trends indicate the coming of even smaller devices, which nevertheless require CV capability. The efficiency of video processing algorithms becomes crucial in such devices as smart glasses or even smart lenses.

The efficient solutions are also needed for scalability. One of the main practical beneficiaries of the work carried out within the scope of this Thesis might be the field of Intelligent Transportation Systems (ITS). The aim of such systems is to efficiently manage transportation, which is facing a growing number of vehicles, congestions, energy consumption, pollution, health risks, costs and increasing human demand for mobility. The target infrastructure for ITS is large, thus the need for scalable solutions. The speed of methods is also crucial when dealing with fast moving objects.

Vision systems for ITS and many other applications have to operate in an outdoor environment. Different and changing weather conditions, seasons and times of the day are unavoidable challenges in computer vision. The algorithms must be able to adapt to the changing environment.

Many of the CV applications include the task of object detection which is a crucial step in understanding the scene in an image or video. A paper [12] that complements the object detection competition PASCAL Visual Object Classes Challange states that object detection must be able to show, where "the instances of a particular object class in the image (if any)" are. Similarly, ImageNet Large Scale Visual Recognition Competition [13] defines the goal of the detection as follows: "Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of one instance of each object category." The result of such detection is shown in Fig. 0.1, where bounding boxes are drawn over the detected objects.



(a) (b) (c)

Fig. 0.1. Object detection in images [14].

Both challenges are widely used as important benchmarks in computer vision, and at the time of writing the corresponding papers have been cited 5204 and 3918 times. Therefore, it is sensible to use the same definitions for object detection in images. However, the central focus of the current Thesis is the processing of videos. While a video consists of frames and the image orientated object detectors can be used to also detect objects in videos, it is not necessary to detect every object in every frame in order to verify the presence of all these objects in a video. In this paper, all objects in videos will be considered as detected, if their presence is recognized at least once during their existence in the scene. This is a useful definition of object detection in videos for such practical task as object counting.

This Thesis aims to improve efficient detection of moving objects in a video, by developing new video processing methods. The developed methods need to be adaptable to the changing environment and usable on devices with limited computational power. These properties would allow the methods to be used in such outdoor applications as ITS, where the large-scale road network would benefit from a low-cost solution.

As described in the section on existing detection methods, the approaches that process whole frames of a video and return bounding boxes of the detected objects can be used to not only detect the presence of an object but also measure its parameters, for example, its dimensions. The existing efficient methods that process only some part of a frame usually lack such capability. **The novel efficient methods that are presented in this paper process only a small part of a frame. However, the advantage of proposed methods is that they are more flexible in their application than the conventional efficient methods. In addition, the proposed methods can be used to acquire an object's parameters.** For this reason, the title of this paper not only refers to object detection but also includes characterization thereof.

Several tasks have been defined in order to reach the aim of this paper:

- to perform a review of literature on object detection in images and videos;
- to identify efficient approaches for detection of moving objects in videos;
- to develop efficient object detection methods with improved capabilities compared to the existing efficient methods;
- to develop data acquisition methods and acquire the data needed for the development and testing of proposed methods;
- to implement and conduct experimental research on the proposed methods;
- to draw conclusions about the results of this Thesis.

The first object detection method proposed in this Thesis is based on extraction and use of several features on a virtual detection line. These features are hand-engineered and chosen after experiments with different combinations of features. However, during the development of this Thesis, the computer vision field underwent a significant change in focus. In 2012, a machine learning approach [15] that used a deep convolutional neural network (CNN) won the object classification competition Large Scale Visual Recognition Challenge (ILSVRC2012) [13]. It noticeably outperformed all other methods. In all ILSVRC classification challenges since 2012, approaches with learned features have won over the approaches with hand-engineered features. Therefore, currently, deep learning (DL) methods experience a large interest from the computer vision community. The best modern object detection methods are based on deep learning approaches. Such methods offer the most accurate results and are potentially the most robust in a changing environment.

This Thesis follows the current trend in the computer vision field by proposing a second object detection method which incorporates the deep learning approach. The usual DL methods are computationally expensive, which currently prevents their widespread application in such areas as highway monitoring and smart cities. Thus, another task of this Thesis is to combine the accuracy of deep networks and the efficiency of detector region-based object detectors.

The success of a fully trainable system on the task at hand and on other tasks largely depends on the availability of labeled training data. The deep learning methods are state-of-the-art approaches for those computer vision tasks that have open datasets. For example, the classic object

detection and localization tasks have the ImageNet data set [13], whereas the image captioning tasks have the MS COCO data set [16]. The proposed machine learning approach is based on the processing of a small portion of input frames, while also incorporating temporal information of the video. Therefore, the existing datasets are not usable for training this method, so particular attention is paid to the creation of data labeling methods.

On the basis of the set tasks, four statements are put forward:

1. The developed IoVDL method detects moving objects in a video with a similar accuracy and computational efficiency as alternative virtual detection region-based methods while being less sensitive to the differing number and trajectories of objects that cross the detection region.
2. The combination of several IoVDL detectors (extended IoVDL) tracks and measures the speed and size parameters of objects with less computational resources than conventional tracking methods that process the whole frame.
3. The combination of the virtual line detector approach with a recurrent neural network results in an adaptive virtual detector RNN-VDL that can be retrained for the detection of different kinds of objects without changes in its architecture and the handcrafted feature engineering.
4. The speed of labeling the several hour long video data for the training of recurrent neural network-based object detector RNN-VDL is increased at least ten times by the developed semi-automatic labeling method.

The Thesis is divided into five main sections. Section 1 is an overview of object detection methods. It begins with an example of a practical task – vehicle detection on the roads. Non-camera based detection methods are compared with the video processing approach, whose different methods are then described in more detail, beginning with a simple analysis of pixel intensities and ending with deep learning approaches. Section 2 highlights the existing efficient video processing methods that process only limited regions of video frames. This virtual detector-based approach is then used to develop a novel virtual detector. First, the most basic form of the proposed method is described, then additional improvements are introduced. The proposed method is expanded so that it not only detects moving objects but also efficiently tracks and classifies them. The section ends with the development of another method – a trainable recurrent neural network (RNN) based approach for detection of moving objects. This machine learning approach needs sufficient amount of labeled data, so Section 3 is dedicated to approaches for data labeling. In Section 4 the proposed methods are tested on vehicle and people counting tasks. The conclusions about developed methods and test results are presented in Section 5.

The results of this Thesis are protected by a European patent [17]. They have been described in several papers [18]–[21] and in image processing related sections of papers [22]–[24].

1. OVERVIEW OF OBJECT DETECTION METHODS

Computer vision offers various techniques for detection of moving objects. However, the video camera is not the only kind of sensor used for such tasks. Since the methods proposed in this paper are mostly tested on detection of vehicles, this section begins by comparing the camera-based approach with alternative detectors used on the roads.

Next, the different kinds of camera-based object detection methods are discussed in more detail. For the current Thesis, the existing methods are divided into three groups. The first group detects objects based on the parameters of individual pixels. The method that is developed in Subsection 2.2 is an example of such approach. The algorithms of the second kind use the features of the whole object or a part thereof, so the patch of the image becomes more important than the individual pixels. The methods in the third group are based on the machine learning approach in which the features of objects are automatically learned from the training examples rather than hand-crafted by the developers. Particular attention is paid to the deep learning methods, which currently are the backbone of the most accurate object detectors.

1.1. Sensors for Vehicle Detection

A summary paper [25] lists and describes different sensors used on the roads. Some of the sensors are intrusive and have to be embedded in the surface of the road or attached to it:

- Pneumatic Tube Detector (PTD);
- Inductive Detector Loop (IDL);
- Magnetometer;
- Weight In Motion (WIM) system;
- WIM based on quartz sensors.

Non-intrusive sensors are usually placed above or beside the road:

- Microwave radar;
- Active infrared laser radar;
- Passive infrared sensor;
- Ultrasonic sensor;
- Passive acoustic sensor;
- Video camera.

1.1.1. Alternatives to Video Processing

The intrusive sensors are more complicated to install than non-intrusive. This difference is even more noticeable if one wants to change the configuration of the sensors after installation.

However, the sensors that are embedded into the surface of the road are hard to vandalize or steal.

A conventional intrusive method uses inductive loops embedded in the road's surface for automatic vehicle detection. Such loops detect the metal components of the close-by vehicles. Passing vehicles reduce the inductance of the loop, which can be measured. By analyzing the signal's signature, the inductive-loop detectors can also be used to determine the type of the vehicle as well as its speed [26]. The magnetometers also detect the vehicle's metal components, which cause the perturbations in the Earth's magnetic field. The magnetometers are smaller than loops and easier to install.

Pneumatic tubes are rubber pipes filled with air. When a vehicle passes over such tube, the burst of air pressure inside the tube closes a switch, which produces an electric signal.

The Weight In Motion systems measure the weight of the vehicle that drives over the sensor. In a piezoelectric WIM system, the sensor generates electric potential proportional to the force applied by the vehicle. It can measure vehicle count, speed, class, weight, length, headway and axis gap.

Radars are used to detect vehicles in motion within a limited detection area. Microwave radars transmit energy towards the road. The cars that pass through the beam reflect a portion of the transmitted energy back to the antenna. The received signal can be used to detect the volume, speed, and length of the vehicle.

Infrared sensors can detect moving and stopped vehicles. They are also used for pedestrian detection tasks. The active infrared sensors use laser diodes to illuminate regions of the road. The cars reflect part of the energy to the infrared-sensitive material. The passive laser approach omits the illumination part, but the infrared sensitive material still gathers energy emitted from the road or the surface of vehicles. The metallic surface of the cars emits more heat than the surface of the road. Similarly to the video processing-based approaches, the infrared sensors are working better at conditions when a human can see the vehicle.

The operation principles of ultrasonic sensors are similar to the radar approaches, but they generate and receive acoustic waves instead of radio waves. The measured delay between emitted and reflected signals indicate the presence of objects in the detection region.

1.1.2. Comparison of Different Sensors

To fairly compare the detection accuracy of different methods, they must be tested on the same road at the same time. In 2010, seven different sensors, including video cameras, were compared within the scope of a project [27]. These methods were tested on the same road for a year in different traffic, lighting and weather conditions. Another study [28] compared three non-intrusive sensors in 2011. These sensors were the video processing system *Autoscope*, microwave radar *SmartSensor HD*, and active infrared system *The Infra-Red Traffic Logger*

(TIRTL). The TIRTL sensor is different from the previously discussed active infra-red sensors since it consists of separate transmission and detection units placed at the opposite sides of the road. The vehicles are detected when they interrupt the beams between the transmitter and the receiver.

The results of these studies are compiled in Table 1.1.

Table 1.1

Results of Different Sensors for Vehicle Counting (2010–2011)

Sensor	MTBF (h)	Environmental sensitivity				Detection error (%)	
		Climate	Light	Traffic	Rain	Study [27]	Study [28]
Video Camera	1026	1	5		1	34.1 ± 4.9	5–10
Microwave radar							> 10
TIRTL							3–5
Laser scanner	1094	1	1.5	1	1	19.8 ± 3.6	
Inductive loops	5064	1	0.5	0.5	1	9.4 ± 1.3	
WIM Piezoelectric	531	2.5	0.5	0.5	1.5	5.7 ± 1.8	
WIM Quartz	85	5	1.5	1	2	12.3 ± 4.1	
Double technology	946	0.5	0.5	0.5	1	2.6 ± 0.6	
Triple technology	549	0.5	0.5	0.5	1	3.5 ± 0.7	

The *double technology* in the above table is a combination of a radar and a laser scanner. It uses the laser for vehicle detection, while the radar measures its speed. The *triple technology* combines a radar, an ultrasonic sensor, and a passive infrared laser. The last two are used for the detection of a vehicle.

The tests also compared the reliability of the systems using the Mean Time Between Failure (MTBF) parameter. The inductive loops proved to be the most reliable technology with MTBF of 5064 hours, while the quartz-based WIM was the most unreliable method with MTBF of only 85 hours. MTBF for the video processing system was similar to the one for the laser scanner and the double technology with 1026, 1094, and 946 hours, respectively. The triple technology and the piezoelectric sensor WIM had MTBF of 549 and 531 hours.

Table 1.1 also shows the environmental sensitivity of the different methods, which was measured in [27]. The measure ranges from very low (0.5) to very high (5) sensitivity.

The comparison of different approaches shows that inductive loops are robust in different environments, including varying temperature, lighting, traffic intensity, and rain. However, not all intrusive methods are as robust as the loops since the WIM methods are sensitive to the changes in temperature and rainfall.

Radars are the most accurate non-intrusive sensors for measuring speeds of vehicles, while some infrared laser-based approaches (triple technology, TIRTL) return the most accurate counts of cars. These methods are also quite robust in different environments, deteriorating only in the case of dense fog or snow. Passive laser sensors are more susceptible to temperature and air turbulence but are robust in changing lighting and traffic conditions.

Unsurprisingly, the most accurate approaches are the combinations of different sensors. The laser scanner used in the double technology was installed on the gantry and acquired the vertical profiles of the cars that drove under it.

The separately tested laser system used several laser diodes and was able to acquire the three-dimensional profile, the headway, and the speed of a vehicle. This system was sensitive to temperature variations. It worked better in winter than in summer, because of lower solar radiation.

The primary interest of the current Thesis is the performance of video-based detection. The described studies conclude that failure of video camera-based methods increases at the time of pronounced shadows, intense precipitation, and, most of all, at night. Small rain does not influence the performance.

The result of video processing significantly differs in the two studies. Both papers observed one possible explanation – the accuracy of video processing methods strongly depends on the lighting conditions. Another plausible reason is that both papers used different video processing approaches. The following subsection demonstrates the diversity of object detection methods in images and videos. Nevertheless, in the described studies video processing was not found to be the most accurate or the most robust approach. So it may raise a question, why even consider computer vision as a solution for vehicle detection and similar tasks?

A timely reason is that the computer vision field has undergone the most rapid progress in the years since these tests were carried out. In 2010, the ImageNet classification competition was held for the first time. In this contest, the prediction of a model is considered to be successful if one of the top five most confident guesses corresponds to the correct class. In the first competition, the best image classification approaches failed at 28.2 % of the test images. Each consecutive year, the classification results were improved, and in 2017 the winning method achieved 2.25 % error. The human performance on this dataset is approximately 5 % [29], and in 2010 it was hard to imagine computer vision methods that would show similar results. However, today the dataset is considered too easy for modern techniques, and the 2017 challenge is planned to be the last competition on the ImageNet dataset.

Furthermore, a single camera sensor can cover a wide road with many lanes. This ability is dependent on the height of the camera and its angle of view. Appropriate choice of these parameters could make the use of cameras a cost-effective approach since a single camera can replace several alternative sensors. However, the processing of camera data requires more computational power than that of a radar. The current Thesis aims to solve this problem and to make camera-based solutions even more appealing than the alternatives.

Finally, the camera is a versatile sensor. For example, similarly to vehicle detection, the video processing algorithm in [30] counts the pedestrians going through a gate. This Thesis goes even further, showing that the same algorithm can be used not only for detection of vehicles but also for the detection of people. The same camera that is used for efficient detection can also

supply the necessary input for other computer vision tasks. Additional algorithms may acquire different parameters of vehicles, classify them, detect emergency vehicles, detect accidents, detect animals or other obstacles on the roadway, estimate the condition of the road, analyze the behavior of drivers, and record the unusual events.

1.2. Detection Using Parameters of Individual Pixels

2D digital images consist of $R_\mathrm{I} \times C_\mathrm{I}$ pixels, where each pixel is described by some value. In the case of grayscale images, the value of each pixel is the intensity I. The color RGB images consist of three channels (red, green, and blue), so each pixel is described by a vector with three values $[I_\mathrm{r}, I_\mathrm{g}, I_\mathrm{b}]$. In simple cases of the object detection task, the objects can be distinguished from the background by analyzing the values of each pixel.

1.2.1. Thresholding

If objects in a grayscale image are lighter (or darker) than the background, they can be detected by simply finding the bright (or dark) pixels in the image. Thresholding operation compares each pixel with a threshold value T. The result of thresholding is a binary image, where pixels with $I > T$ are white (have an intensity value of 1), while pixels with $I \leq T$ are black (value of 0). Fig. 1.1 shows the original image with light objects and a thresholded image. In this example, the finding of white blobs in the thresholded image equals the detection of the objects.

(a) (b) (c)

Fig. 1.1. Otsu thresholding.

(a) input grayscale image; (b) image after thresholding; (c) histogram of the image.

The important parameter of such approach is the value T. Approaches exist, such as Otsu's method [31], which analyzes the input image and automatically finds the value of T. In this method, all possible T values are tried. At each T, some number N_bg of all pixels ($R_\mathrm{I} \times C_\mathrm{I}$) have values less than T, while $N_\mathrm{fg} = R_\mathrm{I} \times C_\mathrm{I} - N_\mathrm{bg}$ pixels belong to the foreground class. This is also observable in the histogram of the image (Fig. 1.1 c) where the x-axis shows all possible

intensity values I, while the y-axis indicates the number of pixels with the corresponding value $\eta(I)$. The histogram is used to find the weights W_{bg}, W_{fg} (1.1), the means M_{bg}, M_{fg} (1.2), and the variances P^2_{bg}, P^2_{fg} (1.3, 1.4) of the respective foreground and background regions.

$$W_{\text{bg}}(T) = \sum_{I=0}^{T} \eta(I), \qquad W_{\text{fg}}(T) = \sum_{I=T+1}^{max(I)} \eta(I), \tag{1.1}$$

$$M_{\text{bg}}(T) = \sum_{I=0}^{T} I \frac{\eta(I)}{W_{\text{bg}}(T)}, \qquad M_{\text{fg}}(T) = \sum_{I=T+1}^{max(I)} I \frac{\eta(I)}{W_{\text{fg}}(T)}, \tag{1.2}$$

$$P^2_{\text{bg}}(T) = \sum_{I=1}^{T} (I - M_{\text{bg}}(T))^2 \frac{\eta(I)}{W_{\text{bg}}(T)}, \tag{1.3}$$

$$P^2_{\text{fg}}(T) = \sum_{I=T+1}^{max(I)} (I - M_{\text{fg}}(T))^2 \frac{\eta(I)}{W_{\text{fg}}(T)}. \tag{1.4}$$

The optimal threshold in Otsu's method is the T that minimizes the within-class variance P^2_{bf} in equation (1.5).

$$P^2_{\text{bf}} = W_{\text{bg}}(T) \cdot P^2_{\text{bg}}(T) + W_{\text{fg}}(T) \cdot P^2_{\text{fg}}(T). \tag{1.5}$$

1.2.2. Color

Color can be used as another distinguishable parameter. In an ordinary RGB image, the color of each pixel is determined by three values. The distance between the color values of different pixels can separate differently colored objects in the image. This idea can be generalized to other color models besides RGB. It also can be used for multispectral and hyperspectral images, where each pixel consists of more than three values ($N_{\text{k}} > 3$) so that the color of a pixel becomes its spectrum. For example, in the papers [22] and [24], the Author of this Thesis uses this spectrum parameter in multispectral images for an image segmentation task. In the concrete example, the pixels of the multispectral image had to be segmented into two classes.

In the proposed method users have to manually select two points in the image. Each point must belong to a different class. The region of 3×3 pixels around each selected point is used to find a mean reference spectrum for both classes $\mathbf{s}_{\text{bg}}$ and $\mathbf{s}_{\text{fg}}$. Figure 1.2 a shows two corresponding spectra. The spectra of all other pixels $\mathbf{s}$ are compared to these two mean spectra, and the corresponding distances are acquired with equations (1.6) and (1.7).

$$\delta(\mathbf{s}, \mathbf{s}_{\text{bg}}) = \sum_{i=1}^{N_{\text{k}}} |\mathbf{s}(i) - \mathbf{s}_{\text{bg}}(i)|, \tag{1.6}$$

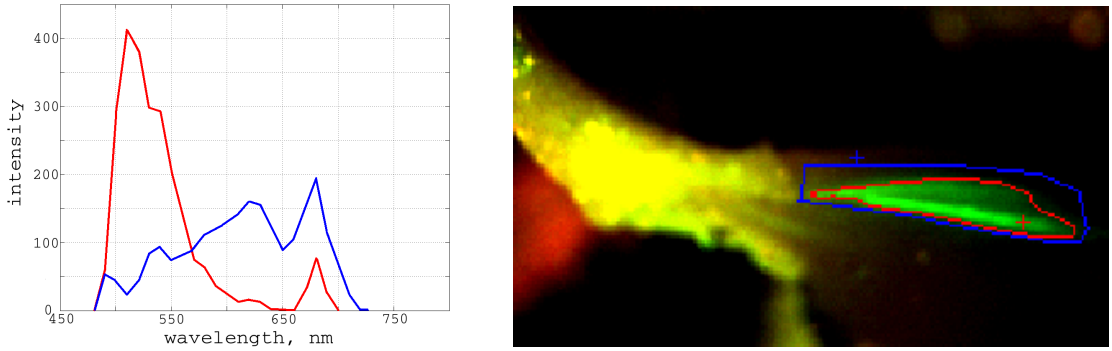
$$\delta(\mathbf{s}, \mathbf{s}_{\text{fg}}) = \sum_{i=1}^{N_{\text{k}}} |\mathbf{s}(i) - \mathbf{s}_{\text{fg}}(i)|, \tag{1.7}$$

where N_{k} – the number of spectral bands (channels).

Then the distances are normalized by equation (1.8).

$$D_{\text{s}} = \frac{\delta(\mathbf{s}, \mathbf{s}_{\text{fg}}) - \delta(\mathbf{s}, \mathbf{s}_{\text{bg}})}{\delta(\mathbf{s}, \mathbf{s}_{\text{fg}}) + \delta(\mathbf{s}, \mathbf{s}_{\text{bg}})}. \tag{1.8}$$

The parameter D_{s} shows how likely is the pixel to belong to the foreground. D_{s} approaches the value -1 if the spectrum of the current pixel is more similar to the first class. In the opposite case, the value of D_{s} approaches $+1$.



(a) (b)

Fig. 1.2. Segmentation of multispectral pixels using the spectrum parameter.

(a) spectrum of pixels of two different classes; (b) segmented image of a mouse leg with fluorescent protein.

Thus far the method is equally applicable to any segmentation task that uses the color or spectrum of a pixel. In the concrete example, the method was further customized for use with multispectral skin images (Fig. 1.2 b). The segmentation task was to identify fluorescent muscle regions with active enhanced green fluorescent protein (EGFP). The green protein has pronounced intensity in the spectral bands of the corresponding green wavelengths. Therefore, the band of 510 nm wavelength is used to improve the segmentation. Thresholding this band with the automatic Otsu's method yields a binary image, where fluorescent pixels are more likely to have a value of 1 than a value of 0.

The final segmentation decision combines the information from the comparison of pixel spectra and automatic thresholding. I_{t} is a value of the pixel after Otsu's segmentation. $I_{\text{t}} = 1$ if the pixel was segmented as fluorescent, and $I_{\text{t}} = 0$ otherwise. Z_{s} is the confidence parameter of the pixel acquired by spectral analysis. Greater value of Z_{s} indicates a greater confidence that the pixel is fluorescent. $Z_{\text{s}} = 0.5$ if $-0.75 \leq D_{\text{s}} < 0$. $Z_{\text{s}} = 1.5$ if $D_{\text{s}} < -0.75$. Otherwise $Z_{\text{s}} = 0$.

The final segmentation decision is computed by equation (1.9), where class $= 1$ means that the pixel is segmented as a fluorescent pixel.

$$\text{class} = \begin{cases} 1 & \text{if } I_{\text{t}} + Z_{\text{s}} \geq 1.5 \\ 0 & \text{otherwise.} \end{cases} \tag{1.9}$$

1.2.3. Background Subtraction

In more complex cases, the values of object pixels may significantly overlap with the values of background pixels. Still, it is possible to detect objects by processing individual pixels, if the pixel values of the empty background image are known in advance. Such approach is widely used in video processing since objects of interest are usually moving, so most areas of the scene display background pixels at some time.

In video processing, background subtraction means the comparison of the current frame with a reference frame of an empty scene. As seen in previous subsections, the objects of interest can be distinguished from the background in different ways. A frequently used parameter is the grayscale intensity of the pixels or values of the color components in some color space [32]–[34]. Edges and corners are robust parameters in changing lighting or weather conditions [35]. In such methods, input images are converted to edge images by using some edge detection methods, for example, convolving the image with Sobel filter or using Canny edge detector [36]. Then, the current edges are compared with the background edges to detect if there are some new objects in the scene.

Since many object detection systems have to deal with the changing outdoor environment, they use an adaptive background image, which can be estimated in different ways. There are non-recursive methods where a specified number of frames are stored in a buffer. The value of the background pixels may be then selected as some statistical value of stored frames, such as mean or median. In the recursive methods, the background is updated using Kalman filter [35], Mixture of Gaussians [37], a weighted sum of the background image and the current frame [34] or a similar technique. The algorithms do not need a buffer in the case of recursive methods; therefore such algorithms are typically more computationally efficient.

1.3. Detection Using the Object's Features

1.3.1. Template Matching

If there is an image (template) of the object $\mathbf{I}_{\text{t}}$ with size $R_{\text{t}} \times C_{\text{t}}$, then this image can be compared to all regions of the input image $\mathbf{I}$ to find the areas of $\mathbf{I}$ that are similar to the object. Usually the comparison is done in the sliding window manner so the displacement of the template is $u = 1, 2, 3...C_{\text{I}} - C_{\text{t}}$ and $v = 1, 2, 3...R_{\text{I}} - R_{\text{t}}$.

The different distance measures are used to assess the similarity of the region and the template. For example, one can use the least squares solution by finding the position with the minimum sum of squared differences:

$$\delta_{\text{ls}}(u,v) = \sum_{i=1}^{R_{\text{t}}} \sum_{j=1}^{C_{\text{t}}} (\mathbf{I}(x_i+u, y_j+v) - \mathbf{I}_{\text{t}}(x_i, y_j))^2. \tag{1.10}$$

Another example is a sum of absolute differences in equation (1.11). This function is more robust to outliers since its penalty grows slower than that of a quadratic function [38].

$$\delta_{\text{abs}}(u,v) = \sum_{i=1}^{R_{\text{t}}} \sum_{j=1}^{C_{\text{t}}} |\mathbf{I}(x_i+u, y_j+v) - \mathbf{I}_{\text{t}}(x_i, y_j)|. \tag{1.11}$$

In [39], a sign change criterion is proposed. After the patch of the input image is subtracted from the template, the algorithm counts the consecutive sign changes in the resulting matrix. The maximum number of sign changes usually occurs when the patch is most similar to the template since then the pixel differences fluctuate around 0. This approach is successfully used in [40]. The author also uses a cross-correlation of the log-polar Fourier spectrum of images [41] to determine the changes in the rotation and scale. This robustness to rotation and scale is challenging for template-based detection. A straightforward way to address these transformations is to try out different sizes (image pyramid) and rotations of the template. This approach, however, significantly increases the number of computations. It is one of the reasons that the feature point-based methods are often more appealing than the template matching.

1.3.2. Keypoint Detectors

An object can be detected by finding distinctive points or regions that are peculiar to this object. In this way, even partially occluded objects can be detected. The points can be the corners, blobs, and T-junctions of an image. Such points are usually described by their surrounding area and are called keypoint features or interest points [38]. The feature-based object detection can be divided into three stages: detection of feature points (keypoint extraction); description of these points; finding of the point pairs between the object and the image (keypoint matching). Video applications often include an additional step of tracking of the feature points. This subsection further describes some widely used detectors.

Harris Corner Detector [42] is based on the second moment matrix $\mathbf{M}_2$:

$$\mathbf{M}_2 = \begin{bmatrix} \mathbf{I}_{\text{x}}^2(x,y) & \mathbf{I}_{\text{x}}\mathbf{I}_{\text{y}}(x,y) \\ \mathbf{I}_{\text{x}}\mathbf{I}_{\text{y}}(x,y) & \mathbf{I}_{\text{y}}^2(x,y) \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \tag{1.12}$$

where $\mathbf{I}_{\mathrm{x}}$ and $\mathbf{I}_{\mathrm{y}}$ denote the first derivatives of the image $\mathbf{I}$ in the directions x and y. The existence of a significant corner at the position (x, y) is determined by a *cornerness measure* Q:

$$Q = (a \cdot c - b^2) - k \cdot (a + c)^2. \tag{1.13}$$

The equation uses an empirically determined constant k with a value of 0.04–0.06. Then a non-maximum suppression is used so that only points with a local maximum of the cornerness measure are considered to be corner points.

Hessian Matrix Detector [43] is similar to the Harris corner detector, but instead of the first order derivatives the second are used:

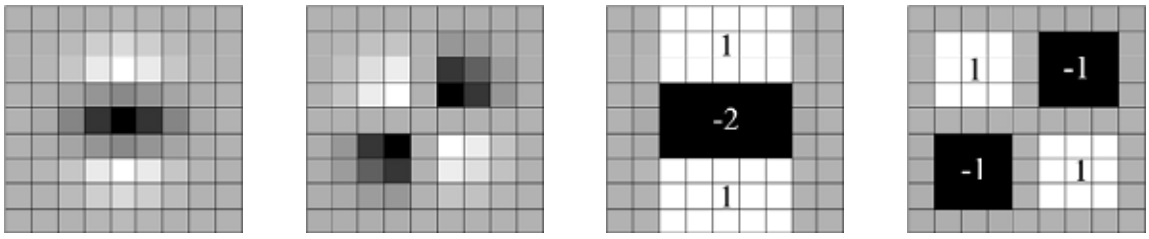
$$\mathbf{M}_{\mathrm{H}} = \begin{bmatrix} \mathbf{I}_{\mathrm{xx}}(x, y) & \mathbf{I}_{\mathrm{xy}}(x, y) \\ \mathbf{I}_{\mathrm{xy}}(x, y) & \mathbf{I}_{\mathrm{yy}}(x, y) \end{bmatrix}, \tag{1.14}$$

where $\mathbf{I}_{\mathrm{xx}}(x, y)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2}\gamma(x, y, P)$ with the image $\mathbf{I}$ at point (x, y) and at scale P. $\mathbf{I}_{\mathrm{xy}}$ and $\mathbf{I}_{\mathrm{yy}}$ are mixed derivatives in directions x and y of the image. The determinant of the Hessian matrix is found for every point in the image:

$$det(\mathbf{M}_{\mathrm{H}}) = \mathbf{I}_{\mathrm{xx}}\mathbf{I}_{\mathrm{yy}} - \mathbf{I}_{\mathrm{xy}}^2. \tag{1.15}$$

Non-maximum suppression is applied to the resulting matrix of determinants. A 3×3 window slides over this matrix and discards pixels that are not larger than all their neighbors. The remaining pixels that are also larger than some threshold are the detected keypoints.

The second derivatives make this detector more responsive to the blobs and ridges in the image. Fig. 1.3 a shows a discretized and cropped Gaussian second order partial derivative in direction y, while Fig. 1.3 b shows the derivative in direction xy. Gray pixels in those images have a value of zero. In practical applications, those Gaussian derivatives may be approximated with box filters 1.3 c and 1.3 d.



(a) (b) (c) (d)

Fig. 1.3. Discretized Gaussian second order partial derivatives [44].

An object may have a different size in different images. The detectors that could find the same parts of the object in different scales are called scale invariant. The Hessian Matrix De-

tector can be modified to be scale invariant by smoothing the image with Gaussians of different scales. Fig. 1.3 shows 9×9 box filters that are approximations of Gaussian second order derivatives when the scale $P = 1.2$. This is the lowest scale which provides the highest spatial resolution. The upscaled filters have sizes of 15×15, 21×21, 27×27 etc. With bigger filters, the step between consecutive scales also increases.

DoG – Difference of Gaussian detector [45] is also scale-invariant. It uses the DoG function (1.16) to blur the image.

$$\beta(x, y, P_i) = (\gamma(x, y, P_i) - \gamma(x, y, P_{i+1})) * \mathbf{I}(x, y), \tag{1.16}$$

where * is a convolution operation and $\gamma(x, y, P)$ is a Gaussian of scale P:

$$\gamma(x, y, P) = \frac{1}{2\pi P^2} e^{-\frac{x^2+y^2}{2P^2}}. \tag{1.17}$$

The DoG results in an image that is the difference between images **I** blurred by the Gaussian filters at different adjacent scales. This result approximates the second derivative of Gaussian. The keypoints are the extrema of the function $\beta(x, y, P_i)$ in the image plane and the scale dimension. This means that the maxima point needs to be greater than eight of its immediate neighbors as well as the nine neighbor pixels from images of the adjacent scales.

MSER – Maximally Stable Extremal Regions [46] are such regions in the image which appear stable when the image is thresholded with different values T. If one thresholds the input image with $T = 0$, then all pixels overcome the threshold and the result is an entirely white image. If T is incrementally increased, some pixel intensities will be lower than T and some black areas will appear in the threshold image. Continually increasing the T, some of the black regions will grow, some will connect, and some will be stable for some interval of T values. These stable blobs are the MSER. The actual thresholdings of the input image are not necessary, and the paper [46] proposes an efficient algorithm for finding such regions. If needed, the found blobs can be converted to points, by computing the centroid (center of gravity) of the stable region.

FAST – Features from Accelerated Segment Test [47], [48] approach determines if a pixel is a corner point by analyzing an unbroken circle of 16 pixels that surround this point. If the intensities of at least 12 contiguous pixels in this ring are all greater or lesser than the intensity of the central pixel by some margin T, then the central pixel is considered to be a corner. Often, it is not necessary to compare the central pixel to all 16 pixels of the circle to determine if the condition is false. The FAST method starts by comparing the central pixel to the circle's pixels 1, 9, 5, and 13. At least three of these pixels must meet the stated condition; otherwise, the central point is quickly rejected as not a corner point.

1.3.3. Hand-crafted Region Descriptors

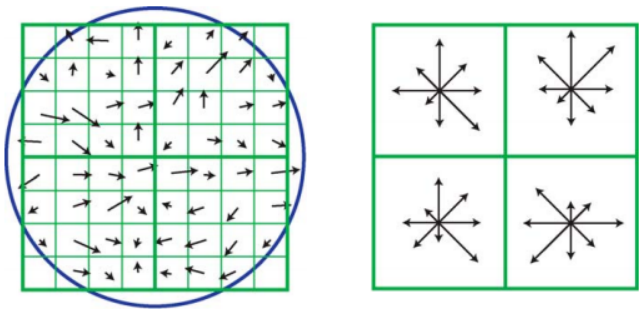
After the discovery of the keypoints, the neighborhood of these points is described by the feature vector. A good descriptor is distinctive and robust to noise and geometric and photometric deformations. Descriptor vectors are matched between different images and the distance between the vectors is used to determine if they represent the same point of the same object.

SIFT – Scale Invariant Feature Transform [49] is a combination of a DoG detector and a SIFT-key descriptor. This descriptor can also be used with other detectors. The SIFT-key computes the gradient magnitudes G (1.18) and orientations O_{g} (1.19) at each pixel surrounding the keypoint.

$$\mathbf{G}(x,y) = \sqrt{(\mathbf{I}(x+1,y) - \mathbf{I}(x-1,y))^2 + (\mathbf{I}(x,y+1) - \mathbf{I}(x,y-1))^2}, \tag{1.18}$$

$$\mathbf{O}_{\mathrm{g}}(x,y) = tan^{-1}\left(\frac{\mathbf{I}(x,y+1) - \mathbf{I}(x,y-1)}{\mathbf{I}(x+1,y) - \mathbf{I}(x-1,y)}\right). \tag{1.19}$$

Fig. 1.4 a depicts the resulting gradients of a 20×20 pixel region. The local gradients are weighted by a 2D Gaussian which is shown as a circle in Fig. 1.4 a. The algorithm combines near pixels into larger regions and computes the histogram of the local oriented gradients inside these regions (Fig. 1.4 b). The example in the Figure shows 2×2 regions around the keypoint. The actual SIFT descriptor finds 8-bin histograms in 4×4 regions. These histograms are combined to form a single descriptor, which is a 128-dimensional vector. The highest value in the vector corresponds to the direction of the most pronounced gradient in the analyzed region. This direction is assigned as the main orientation of the keypoint (some keypoints may have several possible directions). In this way, the keypoints of the object and the keypoints found in the image can be matched even when the object is differently rotated. Therefore, the SIFT descriptor is not only scale-invariant but also rotation-invariant.



(a) (b)

Fig. 1.4. SIFT descriptor [49].

The high dimensionality of the SIFT descriptor is a drawback at the matching step. For real-time applications, all three stages (detection, description, and matching) have to be fast. A variation PCA-SIFT [50] uses principal component analysis to reduce the size of the descriptor.

SURF – Speeded Up Robust Features [44] approach uses the Hessian matrix as a detector. The formation of a descriptor consists of two steps.

The first step detects the main orientation of the keypoint. The region around the keypoint is convolved with horizontal and vertical Haar wavelets, acquiring the responses $\mathbf{H}_\mathrm{x}$ and $\mathbf{H}_\mathrm{y}$. The Haar wavelet response is invariant to bias intensity, which means that the responses of the same object are similar even when the illumination intensity of that object changes. The size of each wavelet used in SURF depends on the scale at which the particular feature point was detected. The wavelet responses are weighted with a Gaussian centered at the keypoint. The weighted responses are converted to vectors in a new 2D space, where the x-axis is the response of the point to the horizontal wavelet, and the y-axis represents the response to the vertical wavelet. A sliding window sums the horizontal and vertical responses in this space. This sum results in a new vector for each position of the sliding window. The orientation of the longest summary vector becomes the orientation of the keypoint.

The second step processes a square region around the keypoint. The orientation of this square matches the orientation of the keypoint found in the previous step. The region is split into 4×4 subregions. Similarly to the preceding step, the Haar responses $\mathbf{H}_\mathrm{x}$ and $\mathbf{H}_\mathrm{y}$ are acquired, only this time the vertical and horizontal directions correspond to the orientation of the keypoint. The responses are weighted with a Gaussian and summed inside each sub-region, resulting in $\sum \mathbf{H}_\mathrm{x}$ and $\sum \mathbf{H}_\mathrm{y}$. Additionally, the algorithm computes the sums of absolute values $\sum |\mathbf{H}_\mathrm{x}|$, $\sum |\mathbf{H}_\mathrm{y}|$. Each subregion forms a four-element feature vector $(\sum \mathbf{H}_\mathrm{x}, \sum \mathbf{H}_\mathrm{y}, \sum |\mathbf{H}_\mathrm{x}|, \sum |\mathbf{H}_\mathrm{y}|)$. Combining the vectors of all subregions results in a 64-dimensional SURF feature vector.

LBP – Locally Binary Patterns [51] are computed in the region of interest by analyzing a 3×3 or larger subregions. In the simplest example of the nine-pixel subregion, the intensity of the central pixel $\mathbf{I}(x_0, y_0)$ is used to threshold its surrounding pixels. The now binary values of surrounding pixels $(x_1, y_1), (x_2, y_2)...(x_8, y_8)$ are weighted with values $1, 2, 4, 8, 16, 32, 64, 128$. The sum of the weighted neighbors (LBP value) becomes the new value of the central pixel. The LBP values of the region are combined into a LBP-histogram, which is a distinctive descriptor of this region.

ORB – Oriented FAST and Rotated BRIEF [52] is presented as an efficient alternative to SIFT and SURF. It uses the FAST keypoint detector to find feature point candidates. In addition, the Harris corner approach is used to select only the most corner-like of the found keypoints. The ORB method adds a corner orientation measurement to the FAST algorithm. In the region surrounding the corner, the method computes the following moments:

$$m_{pq} = \sum_{x,y} x^p \cdot y^q \cdot \mathbf{I}(x, y). \tag{1.20}$$

These moments are used to find an *intensity centroid* [53] of the region:

$$J = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right). \tag{1.21}$$

The vector from the keypoint to the intensity centroid shows the orientation of the keypoint:

$$O_{\mathrm{k}} = tan^{-1} \left(\frac{m_{01}}{m_{10}} \right). \tag{1.22}$$

The ORB uses a modified Binary Robust Independent Elementary Features (BRIEF) [54] descriptor. This descriptor is formed by comparing pairs of the smoothed region's pixels $\mathbf{I}(x_1, y_1)$, $\mathbf{I}(x_2, y_2)$ using test $\tau(x, y)$:

$$\tau\left(\mathbf{I}(x_1, y_1), \mathbf{I}(x_2, y_2)\right) = \begin{cases} 1 & \text{if } \mathbf{I}(x_1, y_1) < \mathbf{I}(x_2, y_2) \\ 0 & \text{otherwise.} \end{cases} \tag{1.23}$$

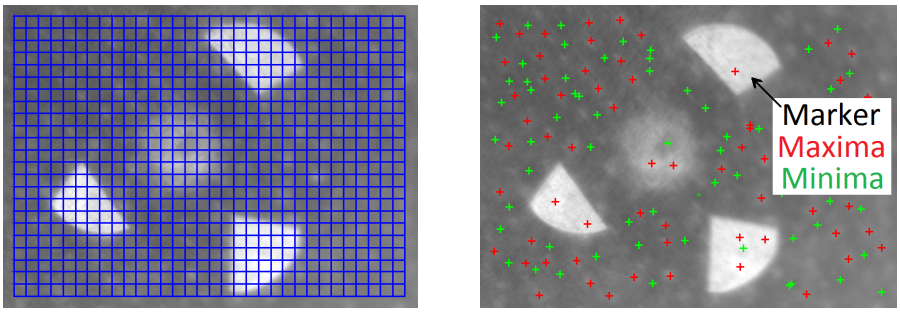
The full ORB descriptor has 256 elements, so this is how many tests have to be performed. There are different ways to choose the point pairs $\mathbf{I}(x_1, y_1)$, $\mathbf{I}(x_2, y_2)$. For example, they might be sampled from the Gaussian distribution around the keypoint. The authors of ORB propose a rotated BRIEF (rBRIEF) approach, where the good test pairs are learned from the data. In the article [54], the authors use 300,000 keypoint examples from the PASCAL 2006 dataset. In the training process, a greedy learning algorithm finds 256 test pairs so that the tests are uncorrelated and the resulting binary feature vector has a high variance and a mean close to 0.5. The high variance makes the features more discriminative, while the uncorrelated tests mean that each test is contributing to the final decision.

1.3.4. Matching the Feature Points

The type of features that are frequent and persistent in the object of interest may differ for different objects. Therefore, a keypoint matching algorithm that can process various kinds of features may be usable in more applications than an algorithm that only uses a specific type of features.

This Thesis proposes a novel feature-matching method that allows the use of different kinds of feature points, for example, centroids of objects, local extrema, and corners. The specific type of the feature points can be changed depending on the actual application, and different kinds of features can also be combined to get more reliable results. For example, in the article [23], the proposed method was used for the registration of multispectral skin images. The local extrema points turned out to be good feature points for such application.

The proposed method to detect those local extrema is to divide the area of an input image



(a) (b)

Fig. 1.5. Detection of local extrema points.

(a) splitting the image into patches; (b) found extrema points.

into uniform patches (Fig. 1.5 a). The minimum and maximum values of each patch become the local extrema (Fig. 1.5 b). If the found local feature point is located on the border of the patch, then the algorithm decides that no keypoint of the specific kind is inside this patch. This approach misses some local extrema points. However, it is fast and can be implemented using efficient matrix operations.

The most straightforward version of the algorithm is not scale or rotation invariant. Initially, the method was proposed as a registration method that finds a translation between two images. In the object detection task, one of the images is replaced by the object. The second image is the input in which the algorithm tries to detect the object. The algorithm assumes an initial position of the object in the input image, then uses the identified keypoints to find the most probable translation of this object. If several different features are used, then the keypoints of a different type are compared separately. For example, it is unlikely that a local maximum point corresponds to a local minimum in the image.

After the detection of the feature points, all possible shifts among the object's keypoints and the image's keypoints are measured. An example is illustrated in Fig. 1.6, where the image of the object is overlaid onto the input image at the initially assumed location. The keypoints of the object are depicted by green crosses, while the feature points of the input image are depicted by purple stars. The algorithm finds the distances x and y between every cross and every star. Each of the distances may represent the actual translation between the assumed position of the object and its actual position in the image.

All measured distances are combined into a single matrix $\mathbf{M}_\mathrm{t}$ with a size of $2 \times N_\mathrm{d}$, where N_d is the number of all calculated distances. Even if different kinds of feature points are used, at this point in the algorithm, the found distances x and y are all combined into a single matrix.

Matrix $\mathbf{M}_\mathrm{t}$ is used to construct a new image $\mathbf{I}_\mathrm{d}$. Fig. 1.7 a shows a part of the formed image. Initially, the values of all the pixels in $\mathbf{I}_\mathrm{d}$ are zeros. Each distance in matrix $\mathbf{M}_\mathrm{t}$ increases the value of the corresponding pixel (x, y) in the new image. If some of the distances stored in matrix $\mathbf{M}_\mathrm{t}$ are equal, then the corresponding pixel acquires even a greater value.

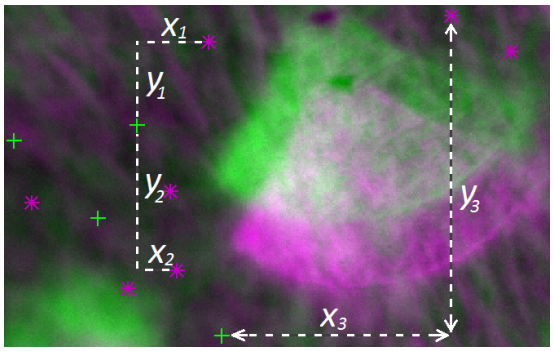
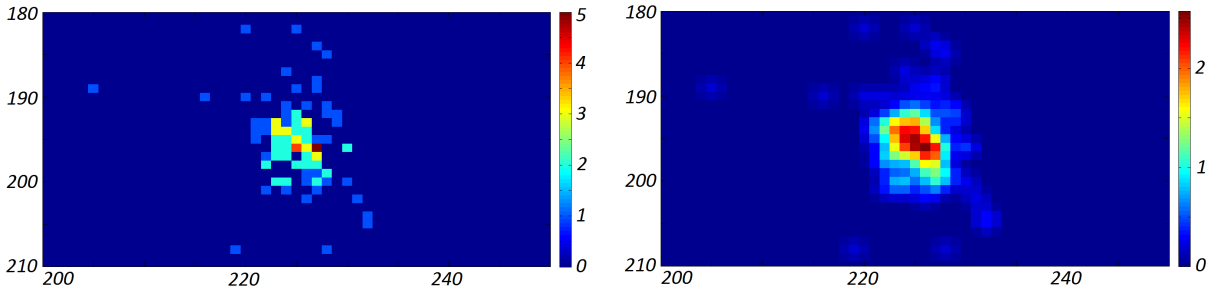


Fig. 1.6. Measurement of possible shifts among feature points.

In the example image of possible translations, the pixels with non-zero values form a visible cluster. This means that many keypoints of the object are similarly shifted from the initially assumed position of the object. If such cluster does not occur, and the non-zero pixels are randomly scattered, then the algorithm decides that the object is not present in the input image.



(a) (b)

Fig. 1.7. Possible translations between the assumed and actual positions of the object.

In the concrete example of Fig. 1.7 a, the cluster indicates the translation of approximately 220–230 pixels in direction x and 190–200 pixels in direction y. The precise translation between the initial position and the actual position of the object is assumed to be the coordinates of the pixel with the greatest value – $max(\mathbf{I}_\mathrm{d})$. This assumption is true if the only transformation of the object is the translation and a significant portion of the keypoints of the object have corresponding points in the input image.

Two unconnected pixels in Fig. 1.7 a have distinctly greater values than the rest. Each of these pixels might point to a correct translation, so the result indicated by the pixel with a value $max(\mathbf{I}_\mathrm{d})$ is not confident. The proposed solution to reduce this ambiguity and the impact of the noise is to use the values of matrix $\mathbf{M}_\mathrm{t}$ to add 2D Gaussian kernels instead of single points when forming the translation image $\mathbf{I}_\mathrm{d}$. Fig. 1.7 b. shows the image of possible translations formed in the proposed fashion.

The proposed keypoint matching method can be made rotation-invariant. To determine the rotation, the feature points of the object are iteratively rotated by some angle. At each angle, a

new matrix $\mathbf{M}_t$ and a new image $\mathbf{I}_d$ are acquired and analyzed as before. The angle that results in the greatest value of $max(\mathbf{I}_d)$ is the most confident result. Similarly, the scale transformation between the initial and the actual scale of the object can be detected by iteratively changing the distances between the object's keypoints and finding the scale with the largest $max(\mathbf{I}_d)$. This extensive search approach significantly increases the number of computations. However, the many rotation and scaling transformations have to be applied only to the keypoints of the object, leaving the usually more numerous keypoints of the input image unchanged.

The capability of the method to use and even combine different types of feature points makes it appealing for various applications. If this method is used for object tracking, then some limits can be set on the largest possible values of measured distances in directions x and y. Tracked objects usually do not move by great distances in consecutive frames, so it is not necessary to analyze the whole frame to find the matching features (this is also shown in the paper [55]). The dropping of the unlikely large translations decreases the number of computations of the proposed method.

More detailed description of the proposed keypoint matching method can be found in the paper [20].

1.4. Deep Learning Approaches

Since a deep learning method [15] won the ILSVRC2012 challenge, the deep artificial neural networks have become the state-of-the-art approach for several computer vision tasks. This machine learning approach not only learns to classify images but also learns to retrieve meaningful features from the input. In the case of the images, the most commonly used type of neural network is the Convolutional Neural Network (CNN). The winning architecture of 2012 (known as AleXnet) is depicted in Fig. 1.8.

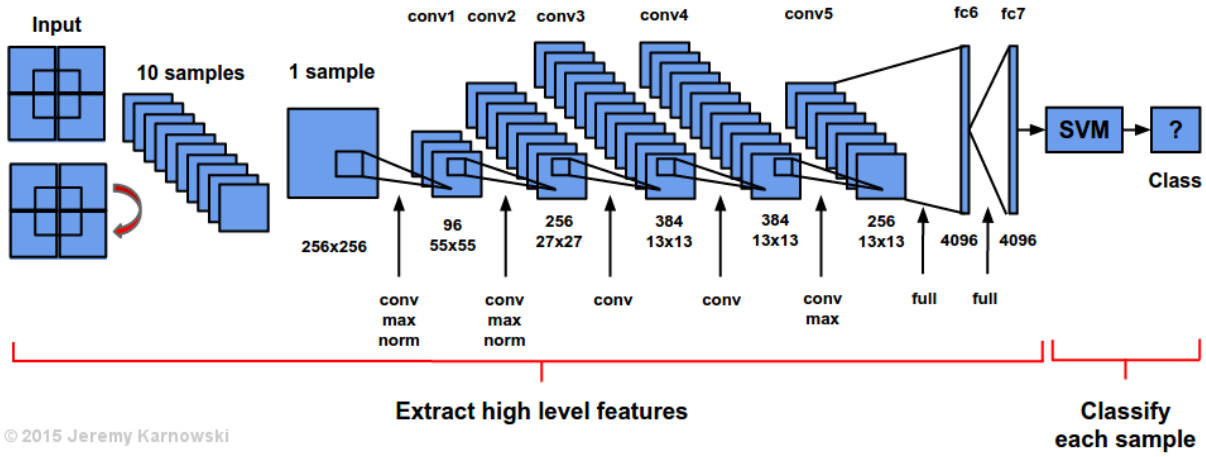


Fig. 1.8. AlexNet architecture [56].

1.4.1. Convolutional Neural Networks

The main layers of CNNs are the convolutional layers. The input to such layer is an image with N_{k1} number of channels. This image is convolved with N_{k2} number of filters. Each convolution returns a feature map. The feature maps are combined into a new image with N_{k2} channels. The new image is similarly processed by the following convolution layers. The aim of training convolutional layers is to learn the coefficients of the convolution filters.

Fig. 1.9 depicts examples of learned layers in CNN. These images are patterns that cause high activations in randomly chosen feature maps of an already trained network. The first layers usually learn to act as edge detectors. Deeper layers learn to activate on more complicated structures and textures in the image.

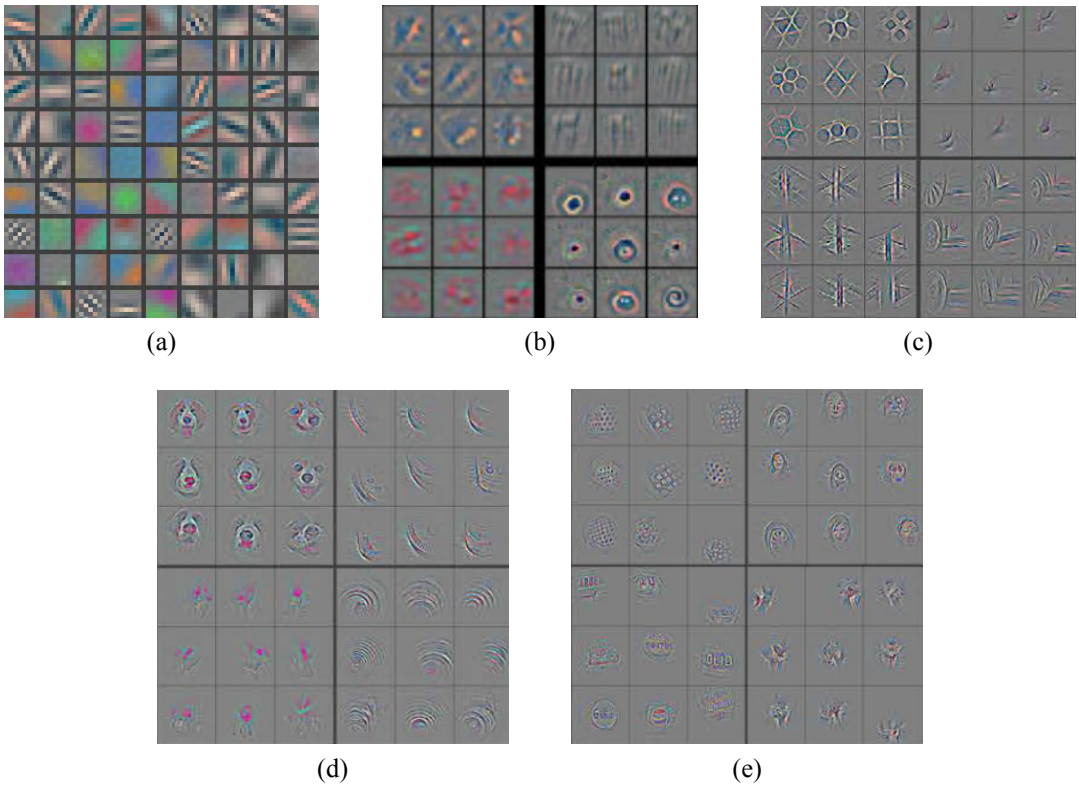


Fig. 1.9. Pattern examples that cause high activations in CNN layers 1–5 [57].

The supervised learning is the most often used approach to learn the weights of the filters. It uses labeled data, where the training images have annotations with the correct answers, which should be returned by a correctly trained model. In the case of a classification task, the labels of images might be {cat}, {dog}, {chainsaw}; in the case of a detection task, the labels may contain the parameters of a bounding box, for example: {14, 1, 88, 88}, {237, 555, 42, 42}. In the training process, the prediction of the network is compared with the correct label using some cost function. An optimization algorithm tries to minimize the cost function by finding

a gradient in the direction of a smaller cost. This gradient is backpropagated [58] through the network, where it changes the coefficients of the model according to their contribution to the error.

Often, a CNN includes pooling layers between the convolutional layers. Pooling reduces the size of feature maps by dividing the maps into regions and choosing a single pixel out of each region. A downsampled feature map is created out of the chosen pixels. Max pooling is a choosing criterion commonly used in CNNs. This approach chooses the pixels with the largest value in the region.

The last layers of CNN usually are fully connected layers. The feature maps are rolled out into a vector prior to entering the first fully connected layer. The output of such layer is a feature vector. In the classic artificial neural networks (ANN), all the layers are fully connected, so all of the neurons in the layer l are connected to all of the neurons in the previous layer $l-1$. Each connection is characterized by a weight W and a bias B. Equation (1.24) shows the output (activation) value A of the j^{th} neuron in the l^{th} layer. The previous layer $l-1$ has $N_{\mathrm{n}}^{(l-1)}$ neurons. The W depicts a connection weight between the neurons of the adjacent layers. B is a bias weight of each connection. The $\alpha(x)$ is a nonlinear activation function. Frequently used activation functions are sigmoid $\sigma(x) = e^x/(1+e^x)$, hyperbolic tangent $\phi(x) = 2/(1+e^{-2x})-1$ and linear rectifier $\rho(x) = max(0, x)$.

$$A_j^l = \alpha\left(\sum_{i=1}^{N_{\mathrm{n}}^{(l-1)}} W_{ji}^{(l)} \cdot A_i^{(l-1)} + B_j^{(l)}\right). \tag{1.24}$$

The whole layer converts the input vector $\mathbf{x}$ into an output vector $\hat{\mathbf{y}}$. This is more conveniently described using vector notation (1.25). The goal of training such layer is to learn the weights $\mathbf{W}$ and the bias vector $\mathbf{b}$.

$$\hat{\mathbf{y}} = \alpha(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{1.25}$$

The AlexNet architecture in Fig. 1.8 ends with a Support Vector Machine that uses the output feature vector from the neural network to classify the input image. The use of CNNs for object detection has also been researched even before the success of the AlexNet. In [59] and [60], a CNN is used for scene parsing. It is also called semantic segmentation or full scene labeling. The goal of semantic segmentation is to detect, segment and recognize all objects in the image, i.e. to label each pixel of the image as belonging to a predefined class. Authors of [60] use a convolutional network to extract features from a multi-scale pyramid of images, instead of using the previously discussed hand-engineered features detectors (such as SURF, ORB, and others).

Detection as a classification task. Since for the past several years the convolutional networks have been state-of-the-art image classifiers, it is appealing to convert the object detection task into a classification task. A simple way to do this is to use a sliding window approach. A re-

gion of the image is fed into CNN, which outputs a class of the region. The predicted class is prescribed to the central pixel of the region, so the pixel can either belong to some object or be a background pixel. The sliding window provides that all pixels of the image are sequentially classified. Adjacent pixels of the same class can then be connected to form objects. This approach is slow since many overlapping regions of the image have to be forwarded through the CNN. Also, the processing of each patch does not take into account the information from the rest of the image.

Detection as a linear regression task. The CNN can also be used for linear regression tasks, where the output is not a class of the input but an input-dependent value. The example of using such approach for object detection is a CNN with four outputs. The first two outputs return positions x and y of a corner of an object's bounding box ($X_{\text{bb}}, Y_{\text{bb}}$), while the remaining outputs return the width C_{bb} and height R_{bb} of this bounding box.

Fully Convolutional Network is a modification of CNN proposed in [61], where it is used for a segmentation task. In image segmentation, each pixel of the image is attributed to one of the possible classes.

The FCN consists only of convolutional layers. Such network can take as its input an image of any size because only the fully connected layers need an input of a specific size. The FCN outputs an image with the same size as input or, more often, an image with resampled dimensions.

1.4.2. R-CNN

Several sophisticated types of convolutional networks have been proposed specifically for detection tasks. In 2014 an R-CNN (regions with CNN features) [62] achieved state-of-the-art results in object detection.

In the R-CNN approach, many regions are extracted from the input image. These extracted regions are classified with the CNN in order to determine if they correspond to one of the possible object classes. Fig. 1.10 shows the main steps of R-CNN.

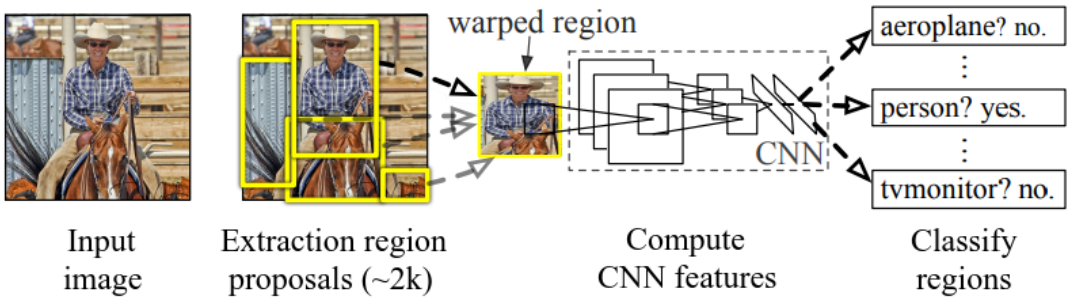


Fig. 1.10. The R-CNN method [62].

The candidate regions are found using *Selective search* method [63], which consists of segmentation of the image and hierarchical grouping of the segmented regions. Selective search begins with the discovery of small areas that consist of internally similar pixels. The areas are compared to the neighboring regions, and similar neighbors are connected. Such grouping continues until the whole image consists of a single region. All regions that existed in the previous steps of this grouping are considered as possible objects. In this way, a single image may yield several thousand candidate regions.

Next, each candidate region is forwarder trough trained CNN. The acquired feature vectors are classified by the Support Vector Machine. The SVM returns the class and also the confidence of its result. If some candidate areas of the same class are overlapping, then the region with smaller confidence is ignored.

The original R-CNN is a slow object detector. The code provided on [64] needs 20 seconds to process a single image on NVIDIA Tesla K40 GPU. Several papers address this drawback of R-CNN and propose *Fast* R-CNN and *Faster* R-CNN methods.

***Fast* R-CNN** [65] has less intermediate stages than the original R-CNN method. In the R-CNN, each candidate region is processed by CNN, even if some regions are overlapping. In this manner, the features from the same areas of the input image are extracted repeatedly. In *Fast* R-CNN, the convolutional network is used once on the whole input image and returns a feature map. When the Selective Search algorithm proposes a candidate region, the appropriate feature vector is acquired from this feature map. The feature vector is then forwarded to the fully connected layers of the CNN, where the network splits into two output layers. One of the layers determines the class of the detected object, while the other layer outputs the parameters of the found object ($X_{\text{bb}}, Y_{\text{bb}}, C_{\text{bb}}, R_{\text{bb}}$). Tests by the authors show that *Fast* R-CNN can be trained faster. It is also more accurate on a PASCAL VOC 2012 [66] dataset and it detects objects 200 times faster than R-CNN.

***Faster* R-CNN** [67] proposes to replace the Selective Search method with a specific convolutional network - Region Proposal Network (RPN). The input of RPN is an image, and the output consists of bounding box coordinates and a classification confidence score for each possible class of the object. In the *Faster* R-CNN approach, the input of RPN is the feature map acquired by the CNN on the input image. In a sliding window manner, an area of the image is fed into a small neural network, which acquires feature vector for each position of the window. Similarly to the *Fast* R-CNN, each vector is fed into two fully connected layers - the regression layer and classification layer. For each position of the sliding window, the network finds several bounding boxes of possible objects with the corresponding confidence scores. Tests on PASCAL VOC dataset show that *Faster* R-CNN is even faster and more accurate than *Fast* R-CNN.

1.4.3. YOLO

The YOLO (You Only Look Once) method is another state-of-the-art approach for object detection. Its precursor is presented in [68], where CNN is used to detect a position in an RGB-D image (D for depth) at which a robot arm can grasp an object by its handle.

The artificial neural network used in [68] has five convolutional layers followed by three fully connected layers. It also includes normalization and maxpooling layers between some of the convolutional layers. Using this CNN as a base, authors propose several models for localization of grasp positions. The simplest model carries out a regression task. The output of the network consists of six neurons - four neurons for the location and size of a bounding box ($X_{\text{bb}}, Y_{\text{bb}}, C_{\text{bb}}, R_{\text{bb}}$) and two neurons for its rotation angle.

In order to detect several graspable objects in an image, authors propose to divide the image into $N_{\text{p}} \times N_{\text{p}}$ cells. The method assumes that each cell of a grid contains no more than one grasp. The model predicts one grasp per cell and also gives the likelihood that the found grasp is valid. The center of the grasp must be within the cell for it to be valid. The size of the output of such model is $N_{\text{p}} \times N_{\text{p}} \times 7$. The first channel is a heatmap showing a likelihood of a region to contain a correct grasp. Other six channels show the predicted grasp coordinates. For example, if the output size is $7 \times 7 \times 7$, it has 343 neurons.

As is shown in [69], a computer vision CNN model can be pre-trained on large existing generic datasets. Then the model can be quickly fine-tuned for a particular task by training it on a smaller dataset of the specific examples. The approach in [68] is based on the AlexNet architecture, which takes an RGB image as an input. Authors pre-trained this model on the ImageNet dataset. Then, to use this model on the RGB-D images, authors replaced the B channel of their images with the depth acquired by the Kinect depth sensor. The depth channel is normalized to fall between 0 and 255. From the experimental results of the paper, it seems that the filters learned on the blue channel (or any other visual channel) are suitable also for the depth of the image.

In [14], the grasp detector is modified to be a generic object detector. The localization task in this paper is addressed as a regression problem. After the processing of the image, the proposed YOLO model returns bounding box coordinates and class probabilities. The model consists of a single CNN, which is similar to the GoogLeNet [70] architecture. It has 24 convolutional layers and two fully connected layers. Instead of inception modules used in GoogLeNet, YOLO network uses 1×1 reduction layers followed by 3×3 convolutional layers. The 1×1 layers reduce the feature space between the convolutional layers.

The paper also proposes a *Fast* YOLO network, which has nine convolution layers and fewer filters in those layers. Everything else is the same between YOLO and *Fast* YOLO.

Fig. 1.11 depicts the main steps of the YOLO method. As in the case of grasp detector, the input image is divided into a grid of cells, and each cell predicts bounding boxes and their confidence scores. Each cell also returns a prediction of the object's class. The first 20 convolu-

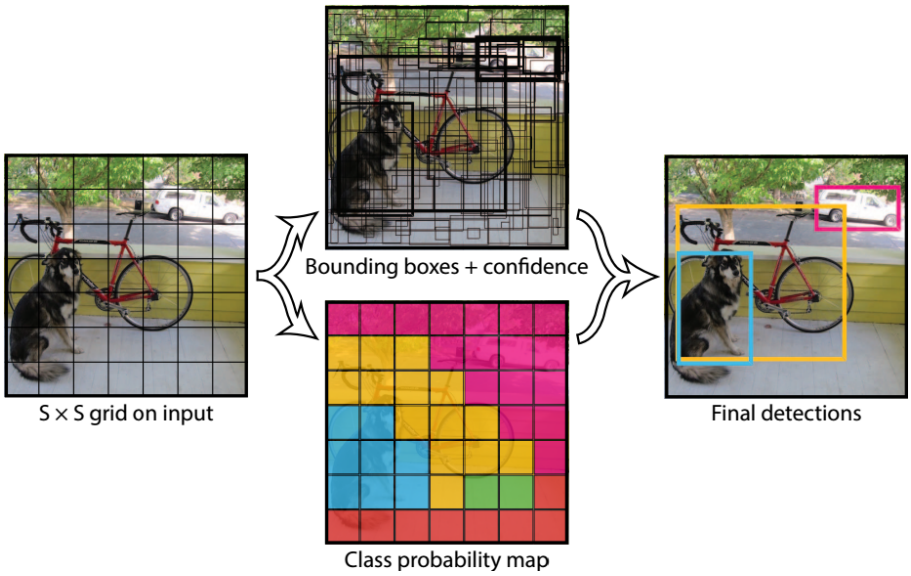


Fig. 1.11. The YOLO method [14].

tional layers of the YOLO model are pre-trained on the ImageNet 1000-class dataset. Then four convolutional and two fully connected layers with randomly initialized weights are added to the pre-trained network. The final layer uses the linear activation function while all other layers use leaky rectified linear activation:

$$\rho_{\mathrm{l}}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.1x & \text{if } x \leq 0. \end{cases} \tag{1.26}$$

The YOLO model is further improved in the paper [71]. In the new model, a batch normalization is added for each convolutional layer. It helps to regularize the model, which means that the model generalizes better on the new data, rather than overfitting on the training data.

In the new model, the fully connected layers are removed. The prediction of bounding boxes incorporates the RPN (region proposal network), which was proposed in a paper about *Faster* R-CNN [67].

The YOLOv2 predicts bounding boxes on a 13×13 feature map, which is a small resolution compared to the input images. To find smaller objects authors add a pass-through layer that keeps the features from an earlier layer at 26×26 resolution. This pass-through layer concatenates the higher and lower resolution features by stacking adjacent features into different channels instead of spatial locations. This converts the size of the feature map from $26 \times 26 \times 512$ to $13 \times 13 \times 2048$. The transformed map is then concatenated with the original 13×13 feature map.

1.4.4. Recurrent Neural Networks

The conventional neural networks, as well as CNNs, are not explicitly suitable for learning events that change in time. When the different input vectors or images of these feed-forward networks are related in time, this information is lost to the network. Thus, another type of neural network is successful in the tasks where sequences of images, audio snippets or words are involved. The recurrent neural networks (RNN) keep the temporal information of the input and, as summarized in the survey [72], are state-of-the-art methods for several sequence-related tasks.

The input to the RNN may be a sequence $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N_{t1})})$. The output also may be a sequence $(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, ..., \mathbf{y}^{(N_{t2})})$. RNNs have cycles in the network of nodes. At time step t, neurons with recurrent edges are receiving not only the input values $\mathbf{x}$, but also the hidden node values $\mathbf{h}^{(t-1)}$ from the previous time step. On the forward step, the simple recurrent network makes calculations shown in equations (1.27) and (1.28).

$$\mathbf{h}^{(t)} = \alpha_{\mathrm{h}}(x)(\mathbf{W}_{\mathrm{hx}}\mathbf{x}^{(t)} + \mathbf{W}_{\mathrm{hh}}\mathbf{x}^{(t-1)} + \mathbf{b}_{\mathrm{h}}) \tag{1.27}$$

$$\hat{\mathbf{y}}^{(t)} = \alpha_{\mathrm{y}}(\mathbf{W}_{\mathrm{yh}}\mathbf{h}^{(t)} + \mathbf{b}_{\mathrm{y}}), \tag{1.28}$$

where $\mathbf{W}_{\mathrm{hx}}$ is a matrix of weights between the input and hidden layers; $\mathbf{W}_{\mathrm{hh}}$ – recursive weights between the hidden layer and itself; $\mathbf{W}_{\mathrm{yh}}$ – weights between the hidden and the output layers. Different activation functions α_{h} and α_{y} can be used.

RNN can be trained using Backpropagation Trough Time (BPTT) method described in [73]. However, because the weights of the recurrent edge $\mathbf{W}_{\mathrm{hh}}$ are equal at each time step, the propagated gradients may quickly vanish or explode. One solution to this problem is the Truncated Backpropagation Through Time (TBPTT) method [74]. This method allows the error gradient to propagate for only a limited number of time steps. Therefore, TBPTT is not suitable for learning long-range dependencies. A more frequently used solution is the Long Short-Term Memory (LSTM) architecture [75].

LSTM. A hidden layer of RNN is replaced with an LSTM memory cell depicted in Fig. 1.12. The literature proposes a variety of LSTM architectures, and an extensive comparison of the different variants is compiled in paper [76]. The typical structure of a memory cell consists of internal state node n_{s} and several additional nodes $n_{\mathrm{g}}, n_{\mathrm{i}}, n_{\mathrm{f}}, n_{\mathrm{o}}$. All the latter nodes receive the signals $\mathbf{x}^{(t)}$ and $\mathbf{h}^{(t-1)}$ as inputs. $\mathbf{h}^{(t-1)}$ is the output signal formed by the hidden layer (the LSTM cell) in the previous time step. The input node n_{g} sums all its inputs and uses a nonlinear function to form an activation value. This value is multiplied with the activation from the *input gate* n_{i}, which may be 1 or 0, so the n_{i} decides if the input nodes output is propagated to the n_{s}. If the output activation of n_{i} is 1, then the output of n_{g} is propagated to the *internal state* node of the memory cell n_{s}.

The internal state node n_s has a linear activation function and a recurrent edge to itself with weight value 1. Because of this value, the error can propagate across time steps in this recursive edge without vanishing or exploding. Another gate driven by $\mathbf{x}^{(t)}$ and $\mathbf{h}^{(t-1)}$ is the *forget gate* n_f. It can trigger the deletion of internal state n_s, or it can leave the n_s unchanged.

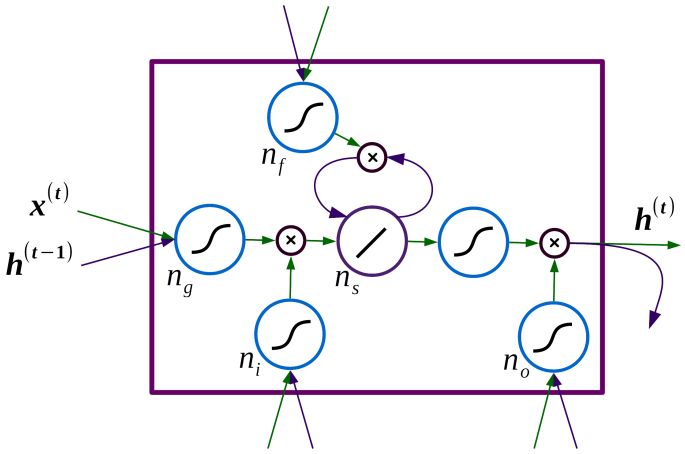


Fig. 1.12. Memory cell of LSTM network.

In order to get the output signal of the cell, the internal state value n_s is run through a non-linear function and multiplied with the value from the *output gate* n_o. The full forward pass computations inside the described memory cell are shown in equations (1.29–1.34), where $\odot$ stands for pointwise multiplication.

$$\mathbf{n}_\mathrm{g}^{(t)} = \alpha(\mathbf{W}_\mathrm{gx}\mathbf{x}^{(t)} + \mathbf{W}_\mathrm{gh}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{g}) \tag{1.29}$$

$$\mathbf{n}_\mathrm{i}^{(t)} = \alpha(\mathbf{W}_\mathrm{ix}\mathbf{x}^{(t)} + \mathbf{W}_\mathrm{ih}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{i}) \tag{1.30}$$

$$\mathbf{n}_\mathrm{f}^{(t)} = \alpha(\mathbf{W}_\mathrm{fx}\mathbf{x}^{(t)} + \mathbf{W}_\mathrm{fh}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{f}) \tag{1.31}$$

$$\mathbf{n}_\mathrm{o}^{(t)} = \alpha(\mathbf{W}_\mathrm{ox}\mathbf{x}^{(t)} + \mathbf{W}_\mathrm{oh}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{o}) \tag{1.32}$$

$$\mathbf{n}_\mathrm{s}^{(t)} = \mathbf{n}_\mathrm{g}^{(t)} \odot \mathbf{n}_\mathrm{i}^{(t)} + \mathbf{n}_\mathrm{s}^{(t-1)} \odot \mathbf{n}_\mathrm{f}^{(t)} \tag{1.33}$$

$$\mathbf{h}^{(t)} = \alpha(\mathbf{n}_\mathrm{s}^{(t)}) \odot \mathbf{n}_\mathrm{o}^{(t)}. \tag{1.34}$$

1.5. Summary

Video camera is not the only sensor that can be used for object detection. This section began by describing alternative approaches used on the roads. The comparison of different vehicle detection methods showed that camera was not the best choice for this task in 2010.

In recent years, however, the computer vision field has made rapid progress in several vision tasks, including object detection. New methods, including the novel ideas in this Thesis, have led to either more accurate, more efficient, or more widely applicable object detectors. Therefore, the appeal of the camera as a versatile sensor has grown.

The overview of camera detection methods shows a great variety of existing algorithms. The simplest approaches, such as thresholding and background subtraction are not the most adaptive and accurate object detectors. However, these approaches are a good starting point for computationally efficient methods. This paper explores and proposes approaches to make such methods even more efficient by reducing the number of processed pixels in the frame.

More robust approaches, such as SURF and ORB, are based on feature detection and description. Since the descriptors need a patch of the image to describe a feature point, the number of processed pixels in the frame cannot be reduced as much as in the more straightforward methods.

The current state-of-the-art object detectors, such as R-CNN and YOLO, are based on artificial neural networks, especially convolutional neural networks. Similarly to hand-crafted feature descriptors, these methods need at least a patch of the image in order to detect the object. Also, both kinds of approaches do not use information from the past frames, while processing the current frame. So another kind of neural network – the recurrent neural network – might be a better suited deep learning approach for video processing since it bases its current output on the sequence of the past frames. However, using RNN in video processing is even more computationally demanding. This paper researches and proposes a method of using the RNN for video processing in a computationally efficient manner.

2. EFFICIENT METHODS FOR OBJECT DETECTION

2.1. Existing Efficient Methods

The processing of images and videos includes manipulation of many pixels. Therefore, one of the methods to create faster object detection algorithms is the reduction of the number of processed pixels. A simple method is to lower the resolution of the image or video; however, this approach may lead to a loss of important details. Another approach is to process only a part of the image or frame. In the case of videos acquired by a static camera, a region of interest (ROI) may be defined in the frame (Fig. 2.1). Only the pixels of ROI are processed, so the objects have to be either inside or eventually enter the ROI in order to be detected. This limitation still allows use of such methods in several practical computer vision applications, for example, vehicle counting on highways [77], detection of people entering buildings [30], [78], inspection of products on the conveyor belt [79], [80].

(a) (b) (c)

Fig. 2.1. Examples of ROI based object detectors [80]–[82].

In papers such as [82]–[84] the ROI is composed of one or more lines within a frame. The lines are perpendicular to the usual movement of the objects of interest. Such lines are also called virtual-lines, detection lines or virtual loop detectors (as they are conceptually similar to the inductive loops that are installed in the surface of a road to detect vehicles).

These methods use the background subtraction approach. The background intensity or the color of the pixels of the detection line is determined and subtracted from the values of the same line in the current frame, revealing objects that do not belong to the background. The background line is adapting to the changing lighting and weather conditions, but this adaptation is not instantaneous. Thus any rapid changes in the background may affect these algorithms.

Some uses of ROIs as virtual detectors still process the whole frame of a video. For example, in the [81] authors propose to use an inter-frame difference in order to remove moving objects and acquire the background image. This background is subtracted from the current frame segmenting the pixels into background and object pixels. The objects are actually counted only when they enter previously specified detection regions in the frame. The regions are used in

order to count, how many objects have crossed each specific ROI in the frame and not simply to count all objects in the scene. This example demonstrates that not all approaches with detection regions have the efficiency advantage that is achieved by reducing the count of processed pixels.

Another manner in which virtual detector-based approaches come to the processing of larger 2D images is shown in the paper on vehicle detection [85]. A virtual detection line is placed perpendicularly to the road and covers all the width of the road. The line is used to create a spatio-temporal image. One axis of such image corresponds to the frame number while the other axis displays the pixels on the detection line at the corresponding frame. Object detection then is performed on the spatio-temporal image, which is updated in time. Although in a single frame only one dimensional slice of the vehicle is located on the virtual detector, the vehicles become 2D blobs in the spatio-temporal image.

Similarly, in [86] several virtual detection lines are placed along the road and are used to create several spatio-temporal images. Close vehicles that correspond to a single blob in one spatio-temporal image because of occlusion may not be occluded on other virtual detectors. So, by matching the corresponding vehicles in different spatio-temporal images, the method deals with the occlusions and furthermore allows the classification of vehicles by their shape and texture in the spatio-temporal image.

In the specific application of highway monitoring, some ROI based approaches (for example [81], [84], [87]) have to precisely define the detection regions for each lane of the road as in Fig. 2.1 b. This approach is susceptible to errors caused by long shadows of vehicles that may be detected as independent vehicles by virtual detectors at adjacent road lanes. So additional shadow elimination techniques may be implemented [81], [84], [88]. Similar errors may occur if a large truck is covering adjacent lanes in a frame or a vehicle is changing lanes and hits two detectors simultaneously. Furthermore, these existing methods are not usable in cases when the number and position of lanes may vary with time. The method proposed in this Thesis solves this issue using a novel interval approach that is discussed in the next section.

The existing detection region-based approaches have an additional drawback compared to detection methods that process the whole image. The latter methods may not only determine if the object exists in the image, but can also return its position or bounding box. This allows the characterization of the objects, for example, measuring their size. If the method finds the location of an object in consecutive frames, it can also be used for the tracking task. Most of the region-based detectors can only detect if the object is present at a single location, and they lack the capability to find a bounding box of an object. However, the specific detector that is the contribution of this Thesis and is described in the following subsection is expandable for such applications as object parameter acquisition and tracking. This is another aspect in which the proposed detection method is more flexible than the existing efficient methods.

2.2. Novel Detection-line Based Method

The proposed ROI based method efficiently detects objects in videos. The method detects objects that arrive on a defined line in a video frame. This line (a row of pixels) is typically placed perpendicularly to the expected movement of the objects. All further processing is carried out on this line only, thus making the detection algorithm computationally efficient. The inter-frame difference is used to detect the presence of the movement on the line, while the background subtraction is used to find the actual objects that do not belong to an empty line.

2.2.1. Intervals on a Virtual Detection Line (IoVDL)

In the detection process, intervals are created on the detection line, denoting the segments covered by the objects. Those intervals are stored in a permanent buffer $\mathbb{B}_\text{i}$. Each interval in $\mathbb{B}_\text{i}$ is updated with the information acquired from the following frames to reflect the changing overlap of the corresponding object with the detection line. When the object leaves the detection line, its interval is closed, and the object's parameters are acquired. In the following text, the developed method is referred to as IoVDL – Intervals on Virtual Detection Line.

As a detection region-based approach, the proposed solution is limited in its applications. The objects of interest have to move in the frame and they have to cross the detection region in order to be detected. However, the proposed interval approach expands over the applicability of existing region-based detectors described in Subsection 2.1. The interval approach allows the detection of objects anywhere on the line since the location of the intervals on the detection line is not restricted. The IoVDL method also allows detection of several objects simultaneously as long as they form separate intervals.

In the simplest cases, the detection line $\mathbf{l}_\text{rgb}$ is a section of a frame's row. However, some positions of the camera may require an inclined detection line. A straightforward way to select pixels of a slanted line is to use linear equation. In order to reduce the computationally costly multiplications, the IoVDL method uses Bresenham's line algorithm [89]. Therefore, the pixels of an inclined detector are efficiently selected using only addition, subtraction and bit shifting operations. The pixels of a slanted line are converted to a row of pixels $\mathbf{l}_\text{rgb}$.

The main steps of the IoVDL method are depicted in Algorithm 1. The input of the proposed algorithm is a line of pixels $\mathbf{l}_\text{rgb}$, which is extracted from a stream of color video frames. In order to detect movement on this line (steps 2–4 of the algorithm), it is converted to a grayscale line $\mathbf{l}_\text{g}$. The grayscale line from a previous frame $\mathbf{l}_\text{g}^{(t-1)}$ is subtracted from the current one $\mathbf{l}_\text{g}^{(t)}$. The pixels with a significant change of intensity in adjacent lines possibly indicate movement, so the absolute value of the inter-frame difference is obtained as a line $\mathbf{l}_\text{d}$. A median filter is applied to $\mathbf{l}_\text{d}$ in order to reduce the noise. Then a threshold T segments the $\mathbf{l}_\text{d}$ into moving objects (white pixels) and static background (black pixels). The automatic acquisition of the T is described later in this subsection.

Algorithm 1. Detecting Objects on the Detection Line

1: **procedure** DETECT-OBJECTS(line from color frame $\mathbf{l}_{\text{rgb}}$, frame number t)
2: line $\mathbf{l}_{\text{g}}^{(t)} \leftarrow$ convert $\mathbf{l}_{\text{rgb}}$ to grayscale
3: line $\mathbf{l}_{\text{d}} \leftarrow$ medianFilter($|\mathbf{l}_{\text{g}}^{(t)} - \mathbf{l}_{\text{g}}^{(t-1)}|$)
4: line $\mathbf{l}_{\text{t}} \leftarrow$ threshold $\mathbf{l}_{\text{d}}$ with threshold value T
5: temporary buffer of intervals $\mathbb{B}_{\text{i\_temp}} \leftarrow$ FIND-INTERVALS($\mathbf{l}_{\text{t}}$)
6: add contents of $\mathbb{B}_{\text{i\_temp}}$ to $\mathbb{B}_{\text{i}}$
7: **for each** (interval i and j in $\mathbb{B}_{\text{i}}$) **do**
8: **if** i and j overlap **then**
9: merge i and j
10: **end if**
11: **end for**
12: line $\mathbf{l}_{\text{i\_t}} \leftarrow$ INTENSITY-BACKGROUND($\mathbb{B}_{\text{i}}, \mathbf{l}_{\text{g}}^{(t)}$)
13: line $\mathbf{l}_{\text{e\_t}} \leftarrow$ EDGE-BACKGROUND($\mathbb{B}_{\text{i}}, \mathbf{l}_{\text{g}}^{(t)}$)
14: line $\mathbf{l}_{\text{rgb\_t}} \leftarrow$ COLOR-BACKGROUND($\mathbb{B}_{\text{i}}, \mathbf{l}_{\text{rgb}}$)
15: **for each** (interval i in $\mathbb{B}_{\text{i}}$) **do**
16: int $Z_{\text{i}} \leftarrow$ the number of white pixels of $\mathbf{l}_{\text{i\_t}}$ covered by i
17: int $Z_{\text{e}} \leftarrow$ the number of white pixels of $\mathbf{l}_{\text{e\_t}}$ covered by i
18: int $Z_{\text{rgb}} \leftarrow$ the number of white pixels of $\mathbf{l}_{\text{rgb\_t}}$ covered by i
19: **if** $Z_{\text{i}} + Z_{\text{e}} + Z_{\text{rgb}} < closingConstant$ **and** i was not updated for *delay* frames **then**
20: **if** $minWidth < width(i) <$ *maxWidth* **and** $duration(i) >$ *minDuration* **then**
21: report a detected object
22: **end if**
23: remove interval i from $\mathbb{B}_{\text{i}}$
24: **end if**
25: **end for**
26: **end procedure**

The detection of moving objects is performed by the method FIND-INTERVALS() at Step 5, where corresponding intervals are created and added to the buffer $\mathbb{B}_{\text{i}}$. The detected interval consists of adjacent pixels that exceed the threshold T. The method consecutively analyses all C_{l} pixels of the line $\mathbf{l}_{\text{t}}$. The position where the current pixel $\mathbf{l}_{\text{t}}(x)$ is white and previous pixel $\mathbf{l}_{\text{t}}(x-1)$ is black is a possible beginning of the interval. The next intensity change from white to black at the position $\mathbf{l}_{\text{t}}(x+C_{\text{i}})$ indicates the end of the interval. Only intervals with the length C_{i}, which meets the condition $C_{\text{i\_min}} < C_{\text{i}} < C_{\text{i\_max}}$, are actually saved. These parameters are set manually and depend on the size of the objects one wants to detect. In the example of highway monitoring, the parameter $C_{\text{i\_min}}$ will determine if motorcycles and bikes will be detected along with vehicles.

The intervals found in the current frame are compared to all already existing intervals in the buffer $\mathbb{B}_{\text{i}}$. If the new interval overlaps with an existing one, they are merged to correctly identify the full extent of the object. Otherwise, new intervals are simply stored in the buffer $\mathbb{B}_{\text{i}}$ for comparison with future intervals (Steps 6–11).

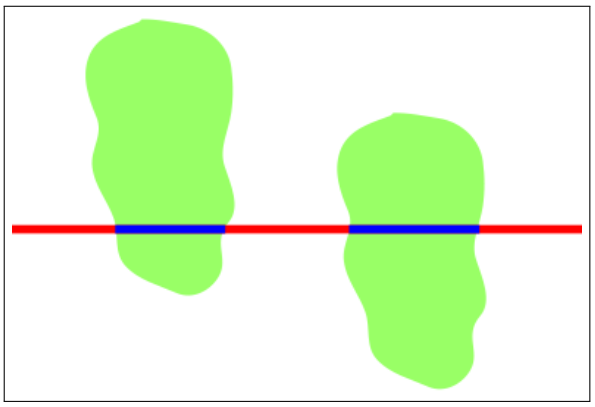


Fig. 2.2. IoVDL in a frame.

The detection line is perpendicular to the movement of the object. Two objects have entered the detection line, so appropriate intervals are created and maintained until the objects leave.

Fig. 2.2 depicts the detection line and two intervals that correspond to the objects crossing the detection line. The intervals are created after the detection of the movement. These intervals must exist while the object is still on the detection line and must be closed when the object has left. The inter-frame difference approach is good for detecting the arrival of the object, but it is not sufficient for keeping the corresponding interval existent until the object leaves. After crossing the line, the movement of uniformly textured objects is not detectable by the inter-frame difference. Furthermore, the objects may stop on the detector line. For this purpose, the method incorporates the background subtraction approach. The method uses the background lines of several parameters: intensity; edges; and color. Based on these parameters, Steps 12–14 of Algorithm 1 determine the objects that do not belong to the background. The output of these functions (described in more detail in the next section) are thresholded lines $\mathbf{l}_{i\_t}$, $\mathbf{l}_{e\_t}$, $\mathbf{l}_{rgb\_t}$ where white pixels correspond to the foreground objects.

Several interval closing conditions are examined in Steps 16–19 in order to suspend the closure of the intervals. In the area covered by every existing interval, the white pixels of different thresholded lines are counted. If the sum of pixels counted for each parameter is larger than the value of *closingConstant*, then it is assumed that the object is still on the detection line and the interval is not closed.

Another condition that prevents removal of the interval is an appearance of a new interval that is merged with the interval of interest. If no new overlapping intervals are found in the area of the already existing interval for *delay* frames and the background parameters do not indicate the presence of an object in the current frame, the interval is removed from the buffer $\mathbb{B}_i$. Afterwards, the parameters of the interval, such as its width and duration, are examined in order to discard too small or too short-lived intervals that cannot correspond to objects of interest. If the closed interval corresponds to the user defined limitations (*minWidth*, *maxWidth*, *minDuration*), an object is detected. The interval's parameters also may be used to distinguish

different types of objects if such exist.

Value of the constant *delay* depends on the frame rate of the input video. Faster frame rates require higher values of *delay* since objects move by less amount of pixels in the consecutive frames. At the same time, the value of *delay* should not be too large, because the interval has to be closed before the next object arrives into the region of interest. In the tests with 15–25 FPS videos, $delay = 3$ turned out to be a suitable choice.

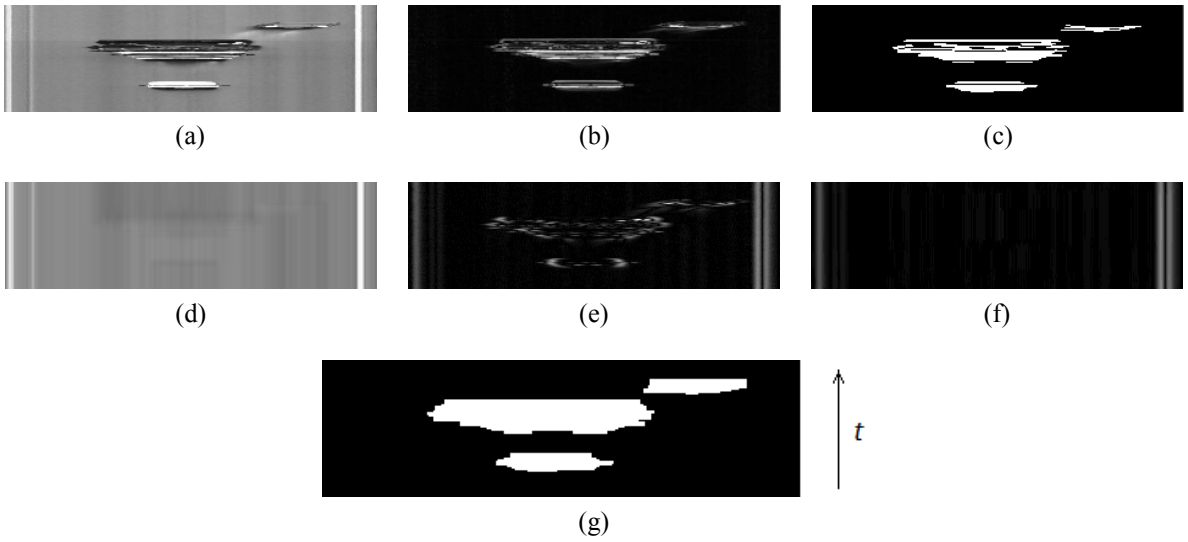


Fig. 2.3. Spatio-temporal images depicting internal steps of the IoVDL algorithm.

(a) intensity line; (b) inter-frame difference line; (c) thresholded inter-frame difference line; (d) intensity background line; (e) edge line; (f) edge background line; (g) intervals.

Several intermediate results of Algorithm 1 are shown in Fig. 2.3. These are spatio-temporal images where the horizontal lines show the pixels of the detection line and the vertical axis correspond to the time. The oldest lines are at the bottom of the images.

2.2.2. Adaptive Background Subtraction

Algorithm 2 shows the background subtraction process of the proposed method on the example of function EDGE-BACKGROUND(). The algorithm maintains and updates a background edge line $\mathbf{l}_{e\_bg}$. There is also a buffer $\mathbb{B}_e$ which contains a set of lines collected for the threshold calculation. Initially $\mathbb{B}_e$ is empty.

First, the intensity line $\mathbf{l}_g$ is blurred using Gaussian low pass filter (Step 2). The edges on the intensity line are found by computing the differences between adjacent pixels. The absolute values of this difference make up the edge line (Step 3). The resulting line of edges is depicted in Fig. 2.3 e.

In order to find objects that do not belong to an empty background, the edge background (Fig. 2.3 f) must be acquired. Every new frame contributes to a recursive formation of the

Algorithm 2. Background Subtraction Using Edge Lines

```
1:  function EDGE-BACKGROUND(intervals buffer B_i, grayscale line l_g)
2:      line l_g ← blur line l_g with a Gaussian low-pass filter
3:      l_e ← find edges on l_g
4:      if B_i is empty then
5:          update the whole background edge line l_e_bg according to (2.1)
6:      else
7:          update the regions of l_e_bg outside intervals according to (2.1)
8:          append l_e to B_e
9:      end if
10:     l_e ← |l_e − l_e_bg|
11:     int T_e ← FIND-THRESHOLD(B_e)
12:     line l_e_t ← threshold l_e with threshold T_e
13:     return (thresholded edge line l_e_t)
14: end function
```

background line $\mathbf{l}_{\text{e\_bg}}$ (Steps 4–7). Value I of each pixel that does not belong to any existing interval is updated as follows:

$$I_{\text{bg}}^{(t)} = E \cdot I_{\text{bg}}^{(t-1)} + (1 - E) \cdot I^{(t)}, \tag{2.1}$$

where $I_{\text{bg}}^{(t)}$ is intensity of the background pixel at the t-th frame; $I_{\text{bg}}^{(t-1)}$ is intensity of the background pixel in the previous frame; $I^{(t)}$ is intensity value of the pixel in the current frame; and E is background update rate. Smaller value of E gives the current frame a bigger impact on the formation of the background. A good starting value used in this work is $E = 0.98$.

If currently, some intervals exist on the detector line, then at Step 8 of Algorithm 2 the edge line is inserted into a buffer $\mathbb{B}_{\text{e}}$. Only a fixed number of lines are kept in this buffer (implementation used in this work kept 200). When the buffer is full and a new line is added, the oldest line is discarded.

Step 10 performs the actual background subtraction. The absolute value line of this difference is then thresholded with the automatically determined threshold T_{e}. This threshold is found by function FIND-THRESHOLD(), using the collected lines in $\mathbb{B}_{\text{e}}$. These lines are combined into a single image $\mathbf{I}_{\text{e}}$, where Otsu method automatically determines the T_{e}. Since only the lines with existing intervals are used to form $\mathbf{I}_{\text{e}}$, the image is likely to contain a balanced amount of foreground and background areas. This is good from the perspective of Otsu method. $\mathbf{I}_{\text{e}}$ is updated in time, so the T_{e} also adapts to the changing environment. For practical applications, one can adjust the frequency of changing the T_{e}. If new T_{e} is determined at every frame, then the system quickly adapts to the changes in the background; however, it also becomes computationally less efficient. At the system's initiation, not enough lines are accumulated to form $\mathbf{I}_{\text{e}}$, so an initial T_{e} has to be manually chosen.

The functions INTENSITY-BACKGROUND() and COLOR-BACKGROUND() are similar

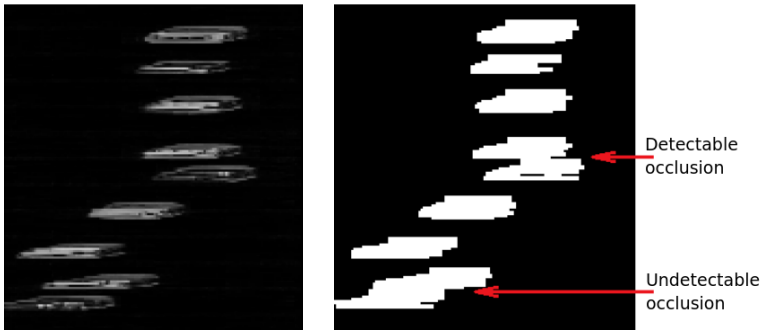
to the described function EDGE-BACKGROUND(), except that object detection from the intensity and color lines do not need the step of edge finding (Step 3). The color lines are formed by representing the RGB input line in the YCrCb color space. In the beginning of COLOR-BACKGROUND(), both color-related components (chrominances) Cr and Cb are processed independently. They are summed before thresholding step.

2.2.3. Improvements of the IoVDL Method

Until now, the proposed IoVDL method was described in its most efficient form. This subsection proposes optional, application-specific improvements, which expand the capabilities of IoVDL while reducing its computational efficiency.

Occlusion detection is an important intermediate task if one wants to improve the accuracy of video-based object detection. In the proposed IoVDL method, the objects that occlude each other in the frame create a single joint interval on a detection line. This type of detection error can be reduced by analyzing the changes in the interval's width during its existence. The proposed improvement is only valid when the objects of interest have a convex shape in the frame.

First, the interval has to be able to reduce its width if the object has become narrower than its corresponding interval. This is accomplished by further analyzing the edge line. In many practical applications, it is reasonable to assume that there is an edge between the object and background. So if the object has the same width as its interval, the edge line will have high-intensity pixels near the ends of the interval. This is true in the example presented in Fig. 2.3 e. If these expressive edges begin to move towards the center of the interval, leaving the ends of the interval empty of edges, then the object of interest most probably becomes narrower. This is used to narrow the interval accordingly.



(a) (b)

Fig. 2.4. Occlusion detection.

(a) spatio-temporal image of intensity line after background subtraction; (b) spatio-temporal image of intervals.

When some interval has existed long enough to possibly match two objects, its historical width is analyzed. If a few consecutive changes of the interval's width are narrowing and then the

interval becomes significantly wider, an occlusion is detected. This approach can be extended to detect an occlusion of more than two objects.

An example of occlusion detection is shown in Fig. 2.4. The spatio-temporal image shows two examples when separate objects have created a single interval. Although, in the background subtracted intensity image Fig. 2.4 a the objects are close to each other, but not exactly occluded, they fuse together in the spatio-temporal image of intervals Fig. 2.4 b. This happens because of the interval closing parameter *delay*, which sustains the existence of the interval for a few frames even when the other parameters are voting to close the interval. The use of a *delay* parameter proves to be important for a significant reduction of the opposite type of error when a single object creates more than one interval. In the concrete example of Fig. 2.4, one of the cases of occlusion is detected by the proposed improvement, while the other case remains an error.

Detection of shadows is important when a pronounced shadow of the object may incorrectly widen the corresponding interval. Fig. 2.5 a shows an example of spatio-temporal intensity image with pronounced shadows.

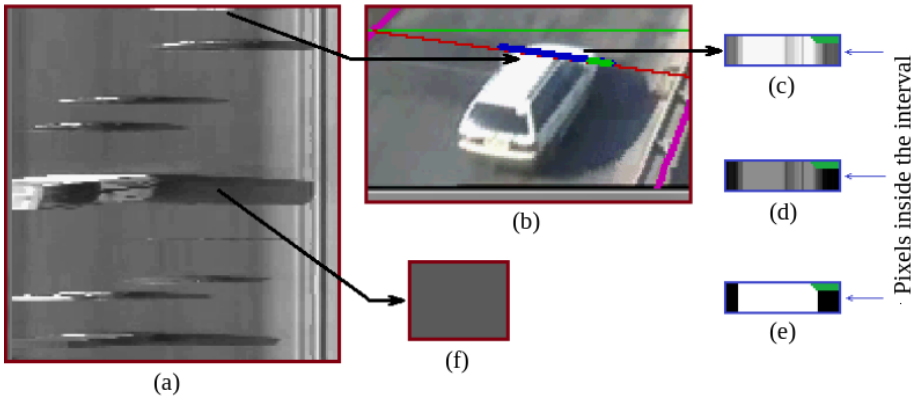


Fig. 2.5. Shadow detection.

(a) spatio-temporal intensity image; (b) inclined detection line with an object's interval (blue segment) and a detected shadow (green segment); (c) intensity line; (d) absolute value of the difference between the intensity line and shadow's mean intensity; (e) thresholding of line (d); (f) mean intensity of the shadow.

The proposed shadow detection method has two operating modes. The first mode is active while the mean intensity and the direction of the shadows are not known. The search of the shadow is carried out in the area of an existing interval by analyzing the intensity values of the interval's pixels (Fig. 2.5 c). Otsu threshold is applied to this line. If the resulting binary image consists of continuous white and black regions, and both of them are bigger than the manually set minimal width of the object ($minWidth$), then the dark region is considered to be a shadow (the region marked with a green line in Figures 2.5 c–e). The intensity of the pixels in this region is used to determine a mean value of the shadow's intensity.

In this mode, only shadows of bright objects are found. When the mean intensity value and the direction of shadows are confirmed by analyzing several objects, the shadow detection

switches to the second mode of operation.

The main addition of the second mode is the computation of the absolute value of the difference between the intensity line (Fig. 2.5 c) and the mean shadow value (Fig. 2.5 f). The resulting line is thresholded (Fig. 2.5 d). This time, the object is segmented as a foreground object when it is brighter, and also when it is darker than the mean shadow. Since it is known whether the shadow is expected on the left or on the right side of the object, a black region in this direction is considered to be a shadow. It no longer has to be wider than $minWidth$. If this black region is followed by another white region, it is likely that the interval is too wide and includes a region of the background or another object.

The mean intensity of the shadow is updated at every frame in which a shadow is detected. If several consecutive objects are detected without any shadows, then the information about the shadow's intensity and direction is discarded. Shadow detector returns to the operating mode 1.

Fig. 2.5 b shows an inclined detector (red line), the interval of found vehicle (blue line) and the detected shadow region (thick green line). In the following frames, the region of the detected shadow will increase, following the edge of the actual shadow in the frame.

Vehicle detection at night on poorly lit roads is challenging for any video-based detector. The proposed interval approach is usable at night, but it has an additional hardware requirement. Fig. 2.6 a shows a poorly lit road. In this case, the vehicle is faintly visible while the headlights and their reflections are the brightest objects in the scene. In order to remove the reflections, one can use a camera with a fast shutter. Figures 2.6 a–c show the resulting frames using shutter speeds of 1/60, 1/6000, and 1/24500 seconds. In the last example, the reflections are removed, while the headlights are still visible.

(a) (b) (c)

Fig. 2.6. Traffic monitoring at night using different camera shutter speeds.

The separate headlights are detected by the IoVDL method in the same manner as any other object. Then the pairs of headlights that belong to a single vehicle are identified. Two intervals are paired if they have existed at close times, their widths are similar, and they are close to each other on the detection line. These comparisons of different intervals are computationally efficient since for every frame an interval on the detection line is described by only two integers.

Detection of rapid lighting changes is important for the IoVDL method because it uses a slowly adaptive background subtraction approach. In an event of rapid change, the movement may be detected on the entire detector line. If the lighting does not return to the previous state, the current intensity values on the empty line are different than the saved backgrounds. However, the backgrounds are not updated to reflect the new situation because the method assumes there is an object on the detection line.

In order to resolve such situation, continuously, an inter-frame difference is acquired for separate pixels in the frame. These pixels are spread throughout the frame. Thus, if most of them have significantly changed their values simultaneously, the intensities throughout all of the scene are probably affected. In such case, the current backgrounds of the intensity, color, and edge lines are discarded. The current detection line is used to initialize new backgrounds. In the following frames, the backgrounds are updated as in the original IoVDL method.

Increased motion sensitivity improves the detection of the objects that are similar to the background. Furthermore, some of the distinguishable features of a fast moving object can skip the single detection line of the IoVDL method. The proposed improvement is to use a thicker detection line using several adjacent rows of pixels to construct the detector (Fig. 2.7). In order to detect the movement, each row is subtracted from the corresponding row of the previous frame. Then the maximum difference of each column among the adjacent lines is used to create a single pixel wide line $\mathbf{l}_d$. This combined line is then processed for interval creation in the same manner as previously.

Similar approach can be used to increase the sensitivity of every other parameter of the object detector. The background subtraction of intensity, edge or color lines is acquired for each adjacent row separately, then it is combined as in Fig. 2.7.

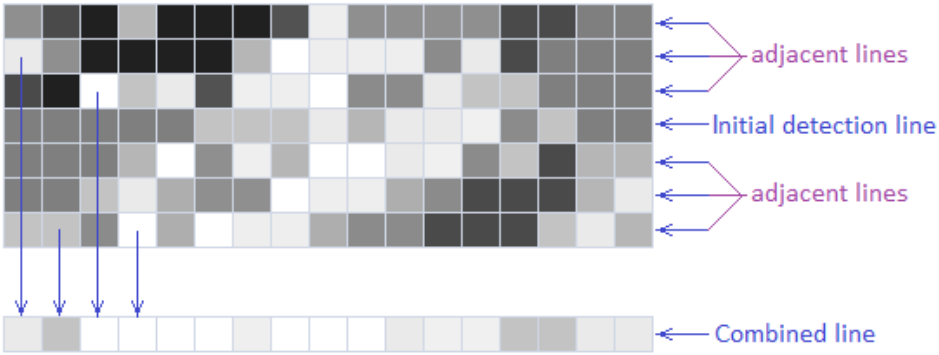


Fig. 2.7. Combining several adjacent lines for a single detector.

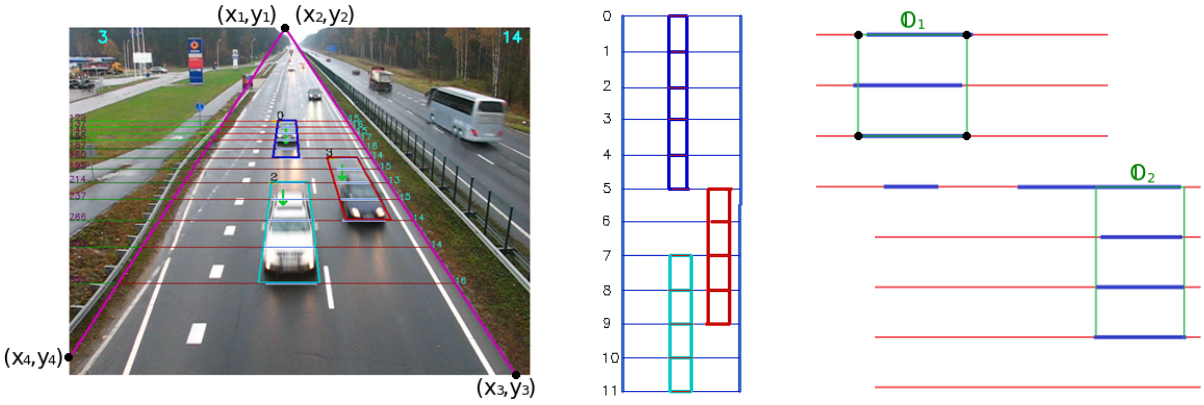
The described use of several adjacent lines is not the same concept as the use of several virtual detector lines described in the next section. In the first case, several lines are used to form a single line detector. The latter concept is the use of several detectors in a frame. Each of those detectors can be formed from one or several adjacent lines.

2.3. Extension of the IoVDL Method for Object Characterization

2.3.1. Combining Intervals into Objects

The method proposed in Subsection 2.2 is efficient and in some ways already more flexible than similar virtual detector-based approaches. As with all ROI based methods, the visual field of IoVDL is limited to a small region in the frame. In this subsection, an extension of IoVDL is proposed. It obtains information from a much wider area of the frame, while still processing only a few detection lines. Several virtual detectors are combined in order to count objects with increased precision, track their movement, measure their speed and dimensions. The positive features of the interval-based approach are maintained and the computing power is significantly reduced compared to other object tracking methods that process all pixels in the tracking area.

Several detection lines are placed in the frame so that each line is perpendicular to the movement of objects. In addition, the detection lines are parallel and equidistant to each other on the actual world plane. Fig. 2.8 a shows an example, where several detectors are placed on the road scene in order to detect and track vehicles. Fig. 2.8 b shows the same detectors in world coordinates. These coordinates depict the scene as if one would look at the road (the world plane) directly from above. The dimensions in the world coordinates correspond to the actual real-world dimensions, so an object that creates an interval of length C_i on one of the detectors will also create the same length intervals on the other detectors that it crosses. This makes it easier to compare intervals from different detectors.



(a) (b) (c)

Fig. 2.8. Several virtual detectors in the frame.

(a) detection lines in a frame plane; (b) detection lines, intervals and detected objects in a world coordinates (after the projective transform); (c) combining intervals to form objects.

Different detection lines may contain intervals that are created by the same moving object. These intervals are combined into a rectangular virtual object that is described by four corner points. An object's coordinates are determined by its constituent intervals. The corners of the

rectangle are located on the detection lines. The x coordinates are determined by computing the median values of the ends of the object's intervals. Adding new and deleting old intervals tracks the object's movement through the detectors. Fig. 2.8 c shows two such virtual objects $\mathbb{O}_1$ and $\mathbb{O}_2$.

In order to update the object, its coordinates are compared to all the intervals that do not yet belong to any object. If the difference between coordinates is smaller than some threshold, the interval is added to the object. This threshold is smaller if the interval is located in the direction of the object's movement. If several objects are candidate owners of the free interval, the value of the coordinate difference and the movement direction of each candidate are considered to find the most fitting one. Conversely, if no object is near enough to the free interval, then a new object is created from this interval. Initially, a new object consists of a single interval.

When an interval is closed on the detection line, it is also removed from the object. In some cases, a still existing interval may be removed from the object if it is too different from the majority of the object's other intervals. An example of such outlier is shown in Fig. 2.8 c, where one of the intervals of the object $\mathbb{O}_2$ is significantly longer than the rest of its intervals. When all of the object's intervals are closed or removed, the object is also closed.

2.3.2. Camera Calibration

The proposed method compares intervals and creates objects in world coordinates, which are usually not the same as the frame coordinates. This means, that at every new position of the camera, the algorithm must be calibrated. This is accomplished by manually setting four points in the frame, so that these points correspond to a rectangle in the real world (points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ in Fig. 2.8 a). The endpoints of the detection lines are placed on the side edges of this rectangle. So in the case of vehicle monitoring shown in Fig. 2.8 a, it is convenient to place these points on the sides of the road. The first detection line starts at a previously defined row. Its column coordinate is automatically found as an intersection between the defined row and the line between points (x_1, y_1) and (x_4, y_4).

The four corresponding world coordinates $(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4)$ of the calibration points must also be defined. To measure the real world values of the object's size and speed, the defined rectangle in the u, v coordinates must correspond to its actual size in the world.

The specified point pairs are used by the algorithm to find the coefficients (a, b, c, d, e, f, g, h) of the appropriate projective transform between the frame plane and the world plane. This is done by solving equations (2.2).

$$\begin{aligned} ax_i + by_i + c &= gx_iu_i + hy_iu_i + u_i, \\ dx_i + ey_i + f &= gx_iv_i + hy_iv_i + v_i. \end{aligned} \tag{2.2}$$

Four corresponding points are enough to fully solve the system (2.2), so using the known

coordinates x, y, u, v the (2.2) becomes:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1u_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2u_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3u_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4u_4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} \tag{2.3}$$

The found coefficients are used to transform any point in the frame plane to a point in the world plane using equations (2.4).

$$u = \frac{ax + by + c}{gx + hy + 1}, \quad v = \frac{dx + ey + f}{gx + hy + 1}. \tag{2.4}$$

In the world coordinates, the algorithm automatically finds the ending of the first detection line. Then all other detectors are placed so that they are all parallel to each other, perpendicular to the movement of the object, and the distances between all adjacent virtual lines are the same. Then the corresponding detectors are set in the frame plane so that the appropriate pixels of the frame are processed. Equations (2.5) are used to find the coefficients of the inverse projective transform.

$$\begin{gathered} a_{\text{inv}} = \tfrac{e - f \cdot h}{a \cdot e - b \cdot d}, \quad b_{\text{inv}} = \tfrac{c \cdot h - b}{a \cdot e - b \cdot d}, \quad c_{\text{inv}} = \tfrac{b \cdot f - c \cdot e}{a \cdot e - b \cdot d}, \\ d_{\text{inv}} = \tfrac{f \cdot g - d}{a \cdot e - b \cdot d}, \quad e_{\text{inv}} = \tfrac{a - c \cdot g}{a \cdot e - b \cdot d} \quad f_{\text{inv}} = \tfrac{d \cdot c - a \cdot f}{a \cdot e - b \cdot d}, \\ g_{\text{inv}} = \tfrac{d \cdot h - e \cdot g}{a \cdot e - b \cdot d}, \quad h_{\text{inv}} = \tfrac{b \cdot g - a \cdot h}{a \cdot e - b \cdot d}. \end{gathered} \tag{2.5}$$

Since all the lines in the world coordinates are parallel to the x-axis, they may be inclined in the frame coordinates. The Bresenham's line algorithm is used to efficiently select the pixels of the inclined detection lines.

The projective transform of a scene or a region is also used in other works on vehicle detection [32], [88], [90]. This transformation of the whole region has to be carried out at every frame, so it becomes a significant drain on the computational resources. In the proposed interval-based approach, the detector positions are transformed once. Then, at each frame, only the intervals have to be transformed. Only two coordinates are needed to describe an interval, so only those are transformed. This processing step of the proposed method is once again computationally more efficient than the alternatives.

2.3.3. Object Characterization

In the world coordinates, all existing virtual objects are analyzed to obtain their parameters.

The direction of the object's movement is initially acquired when its rectangle is created. If the object appears at one of the upper detection lines, its direction is assumed to be from top to bottom. Similarly, objects that appear at the bottom of the analyzed region are assumed to be moving up. In further frames, the direction depends on a confidence of the current assumption and the last added interval. If it is added at the front of the object (judging by the assumed direction), then the direction is not changed and the confidence of the assumed direction is increased by 1. If the new interval contradicts the assumed direction, the confidence is decreased by 1. When the confidence becomes negative, the assumed direction is changed to the opposite.

The found direction helps to remove some intervals that do not actually belong to the object. Ideally, an object would consist of as many intervals as the number of detectors it is currently crossing. However, in some cases, the virtual object may consist of intervals that are separated by an empty detector. It is important to determine, whether these intervals are really created by the same object, or maybe the virtual object has to be divided into separate objects. The second case more often affects the intervals at the back of the object. One object may have just left the detection line, while another one enters it at the same position. The algorithm may fail to close the existing interval before a new object begins to prolong its existence. The proposed approach to deal with such cases is to throw away the N_{ri} rear intervals if they are separated from the rest of the object's intervals by at least N_{ri} empty detectors. Therefore, the knowledge of the movement's direction helps to separate different objects.

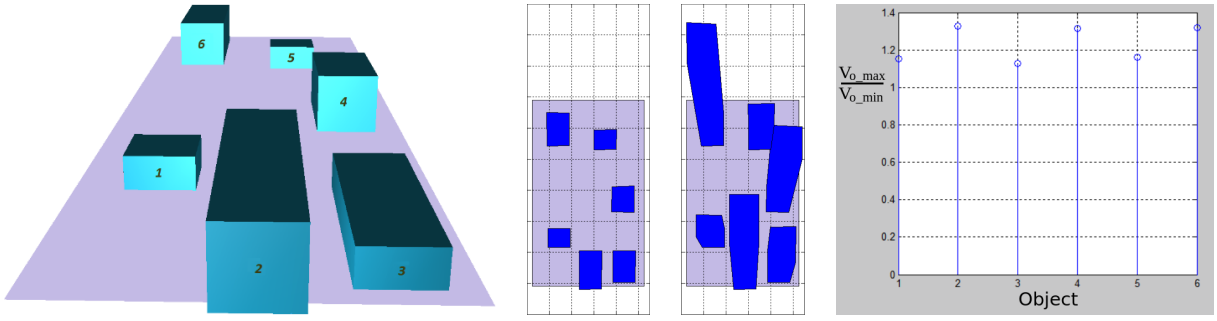
An additional parameter that is acquired by determining the intervals of the object is its movement trajectory. One can draw a line through the centers of all intervals of a specific object and see how the object has moved through the detectors.

Speed is determined when the object has crossed all detection lines, so the obtained value is an average speed in the region of interest. The distance between two intervals of different detectors is divided by the time passed between the creation of these intervals. Such speed estimate is computed between every possible pair of the detection lines. A final value of the object's speed is a trimmed mean value of the separate measurements.

Size. The proposed method models the detected objects as boxes and allows the acquisition of three size parameters: length, width, and height. The length is acquired by multiplying the object's speed with the lifespan of its intervals. As with all the other parameters, the trimmed mean is used to compute the final value of the length.

The initial length of the intervals in the world coordinates (C_{i}) is approximately equivalent to the width of the object. When the camera is facing the ground plane directly from above, only the top of the objects is visible. However, in practical applications, such as vehicle detection, the camera may be inclined towards the ground plane, so it sees not only the top but also the front of

the vehicles. Fig. 2.9 a shows a synthetic example of such fame scene with 6 objects. Fig. 2.9 b shows a perfect top view of this scene, while Fig. 2.9 c is the actual perspective transform of Fig. 2.9 a into world coordinates.



(a) (b) (c) (d)

Fig. 2.9. Detection of height from an inclined camera.

(a) synthetic scene with six objects; (b) top view of the scene (a); (c) projective transform of the scene (a); (d) ratios of object's largest and smallest widths in the projection (c).

Assuming that the objects in the scene 2.9 a are moving towards the camera, the top face of the objects reaches the detection line later than their front face. It may broaden the interval because the top is closer to the camera and appears bigger in the frame. This seeming broadening of the object's width is also visible in Fig. 2.9 c. The widths of the intervals in the frame coordinates translate to the width V_{o} of the object in the world coordinates. The change in the object's width from smallest $V_{\mathrm{o\_min}}$ to largest $V_{\mathrm{o\_max}}$ is indicative of the object's height H_{o}. These widths are saved for each interval of the object and their trimmed mean values $V_{\mathrm{o\_min}}$, $V_{\mathrm{o\_max}}$ are calculated when the object leaves all the detectors. If the height of the camera above the world plane H_{c} is known, the height of the object is found by equation (2.6).

$$H_{\mathrm{o}} = H_{\mathrm{c}} \frac{V_{\mathrm{o\_max}}}{V_{\mathrm{o\_max}} + V_{\mathrm{o\_min}}} \tag{2.6}$$

Even when the height of the camera H_{c} is not known, one can calibrate the height detection algorithm by measuring the difference of the ratio $V_{\mathrm{o\_max}}/V_{\mathrm{o\_min}}$ of two objects with known height. The synthetic example in Fig. 2.9 demonstrates the relation of the interval length in the world coordinates with the height of the object. In Fig. 2.9 a, the height of objects 1, 3, and 5 is 1. The height of 2, 4, 6 is 2. Objects have different lengths (from first to last): 2, 8, 6, 4, 1, 6. Fig. 2.9 d shows the resulting ratios of each object's maximal and minimal widths in Fig. 2.9 c. These ratios are well-correlated with the actual height of the objects.

The described method for measuring the object's height is limited to objects that can be reasonably modeled by a box. The width of the top and the bottom of the objects must be approximately the same.

Class of some objects can be determined by using the already measured parameters. In the example of vehicle detection, one can use the size to distinguish motorcars and trucks. Since all the parameters are acquired in the world coordinates, they are in the correct units of measure (meters, seconds). To classify an object, its parameters are compared to user-defined thresholds. If two out of three size parameters indicate that the detected object is a truck, then it is also the final class of the object.

Fig. 2.10 shows an example of actual measurements of the vehicle sizes. Measurement points corresponding to trucks are circled. The example shows that in some cases the separate dimension of the object may be measured incorrectly, so it is meaningful to combine information about all acquired dimensions when classifying.

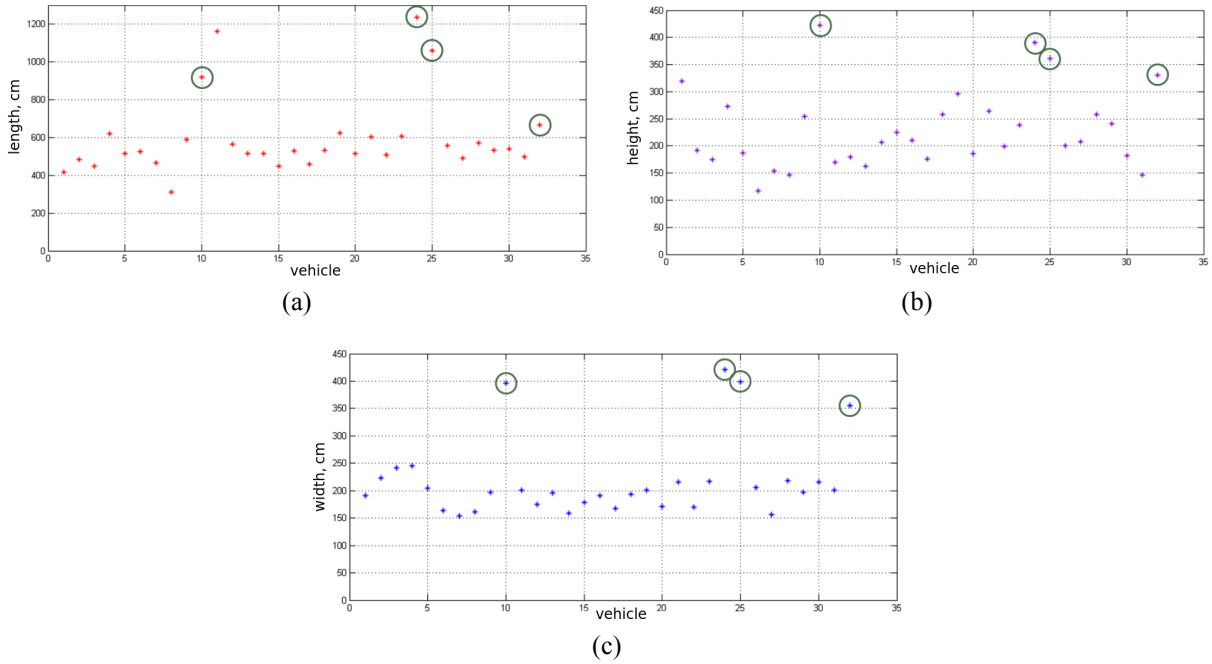


Fig. 2.10. Measurements of vehicle size.

2.4. RNN Based Virtual Detection Line

The methods described in previous subsections relied on hand-crafted features. Each new application of those methods, such as detection of vehicles or people, may require some changes in the algorithm, such as the rate of background adaptation and the particular combination of used features (edges, colors, etc.). In this subsection, an efficient machine learning method is developed. In this method, the concrete features of the detection model are learned from the data, not set by hand. Similarly to the previously proposed IoVDL method, the detection line approach is used. The pixels from the line are processed by a recurrent neural network, so the method is named RNN-VDL.

2.4.1. Architecture

The first step of the proposed method asks the user to define a detection line in a frame. The example virtual detection line is depicted in Fig. 2.11. The detector is a C_l-pixel long and 1-pixel high line which is perpendicular to the movement of vehicles. The aim of such line is to detect moving objects that cross it. In the proposed method, only the detector's pixels are processed, while the rest of the frame is ignored. Different lengths C_l can be used, since the defined line of pixels (vector $\mathbf{x}$) is resized to a fixed shape of $N_\mathrm{fl} \times 1$ prior to entering the neural network. The N_fl is the number of the input neurons of the neural network. The reduced architecture of the proposed artificial neural network is depicted in Fig. 2.11.

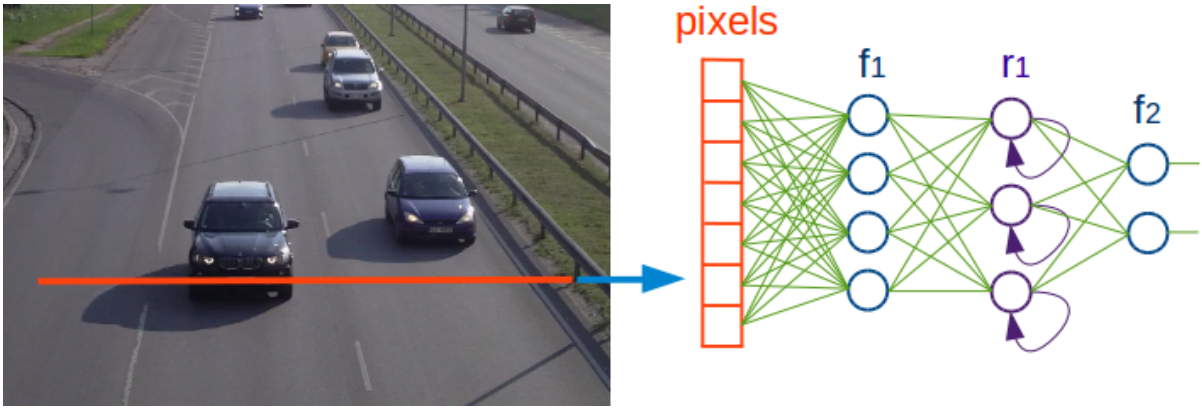


Fig. 2.11. RNN based virtual detection line (RNN-VDL).

In the concrete proposed implementation, the first layer of the network is a fully connected layer with $N_\mathrm{fl} = 100$ neurons. At frame t, the output vector of a fully connected layer is $\mathbf{h}^{(t)} = \rho(\mathbf{W}_\mathrm{hx}\mathbf{x}^{(t)} + \mathbf{b}_\mathrm{h})$. The vector $\mathbf{h}^{(t)}$ is then forwarded to a recurrent LSTM layer with $N_\mathrm{r} = 100$ neurons. The concrete recurrent layer considers $S = 40$ previous frames to compute its output. This property is the reason why RNN is used in the proposed detector. Such internal memory allows the network not only to recognize if there are some objects on the detection line but also determine if some objects have recently left the line. A single line is also sufficient to find out the movement direction of the object. The detailed computations of LSTM layer were already discussed in Subsection 1.4.4.

The final layer of the proposed network is another fully connected layer with $N_\mathrm{f2} = 10$ neurons that correspond to 10 possible output classes. All activation functions in the network are linear rectifiers $\rho(x) = max(0, x)$. The goal of training such model is to learn all the weights, so that the network returns a preferred class for each sequence of input vectors $\mathbf{x}^{(t-S)}, \mathbf{x}^{(t-S+1)}, \mathbf{x}^{(t-S+2)} \ldots \mathbf{x}^{(t)}$. There are different ways how to choose the expected class for each frame; therefore, the next subsection is dedicated to finding the appropriate type of labeling for the object detection task.

2.4.2. Data Labeling Modes for RNN-VDL

The output class labels of the proposed method may represent several possible events on the detection line. In this paper, three labeling modes are proposed. In the first mode, the output label corresponds to the current number of objects on the detection line. For example, if currently there are two moving objects on the detector, the corresponding label for this frame is $\{2\}$.

If the application of the detector is to count passing objects, then the first labeling mode has an essential drawback. Two closely following objects might create a sequence of labels $(\{1\},\{1\},\{1\},\{1\},\{1\})$. Two objects have crossed the line, but the correctly predicted sequence does not allow to determine that two simultaneous events happened at the third frame of this sequence. At this frame, one object has left the line, while another object has entered it.

The labels of the second proposed mode show how many objects have left the line at the current frame. This mode allows unambiguous counting of objects that have fully crossed the detection line. If several objects leave the line simultaneously, their count is also the label for this frame. For example, the sequence of labels $(\{0\},\{0\},\{2\},\{1\},\{0\})$ means that no object has left the detection line in the first two frames. Then 2 objects left the line in the third frame and one more object left the line in the fourth frame. This mode also eases the task of manually labeling the training data since the labeler only needs to add one label per object, not per every frame. However, the labeled dataset of this kind will be unbalanced since in many practical applications most of the labels will be zeros. Furthermore, the labeling of the training data must be precise. With accurate labeling, label $\{1\}$ will be given to the first frame after the object has left the line. In another example, the inaccurate labeling may result in label $\{1\}$ given to a frame that is two frames removed from the actual event. In such example, the dataset now has two very similar but contradicting training examples.

The final proposed labeling mode is a combination of two previous modes and is depicted in Fig. 2.12. The human labeler still only needs to mark the frames where the objects have left the detection line; however, the actual label of any sequence in the video is determined by counting how many objects have left the line in the last $S/2$ frames. In the example of Fig. 2.12, five objects have passed the detection line in the last 28 frames. Each row of the image represents pixels of the detection line at consecutive frames. The first row is the first frame, while the last row is the 28th frame. A digit inside each row shows, how many objects have left the detection line at this frame. In the example, the length of sequence $S = 22$. The latest half of the sequence is colored. The number of departed objects in this part is the final label of the sequence. So the label of current and the following sequence is $\{3\}$. A sequence after that will be labeled $\{2\}$.

While the trained network predicts the label of each frame (analyzing the sequence of S last frames), the actual detection of a moving object takes into account several separate predictions. The global counter increases only if the currently predicted count is greater than the current global count for at least $S/4$ consecutive frames. In this way, the influence of the prediction errors at each individual frame is reduced; however, the object is detected with a small delay.

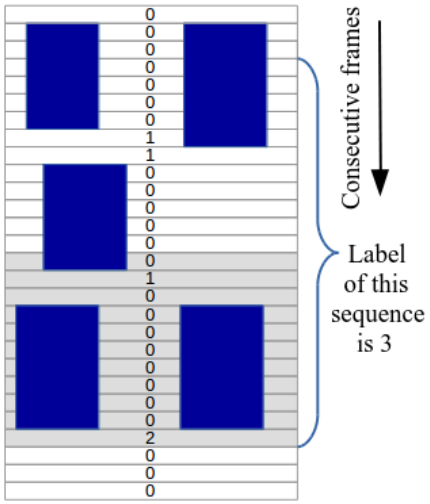


Fig. 2.12. Labeling mode 3 for RNN-VDL.

This spatio-temporal image shows 28 consecutive frames on a detection line that was crossed by 5 objects.

2.5. Summary

Subsection 2.1 provides an overview of efficient video processing methods for moving object detection. It indicates several drawbacks of existing approaches. The following subsections propose novel efficient methods that overcome some of the identified drawbacks.

The developed IoVDL method is more flexible than the alternatives since the interval approach allows the detection of the object anywhere on the virtual line. The method detects objects of various widths, without the possibility that wide objects may activate some adjacent detectors. Such objects simply result in longer intervals created on the line. Furthermore, since the intervals incorporate some information about the size of the object, the IoVDL method can even be used to classify the objects. This possibility to characterize detected objects is expanded by combining several proposed detectors in a frame. The extension is still computationally efficient and can detect the direction of the movement, trajectory, speed and size of the objects.

Finally, the idea of an efficient detection line is combined with a machine learning approach. The use of a recurrent neural network has the potential to make line detectors even more robust and adaptable in changing environments. In order to train this method, a specific training data is needed, so the next section is dedicated to the development of data labeling methods that would speed up the acquisition of training data for different applications.

3. TRAINING DATASET ACQUISITION METHODS

Supervised machine learning depends on the availability of labeled data where the training examples include the correct labels. A learned model should learn to predict these labels from the examples. The success of convolutional neural networks in computer vision can be partly attributed to the existence of such datasets as ImageNet [13] and MS COCO [16]. New applications of deep learning demand the creation of new datasets. For example, a Visual Genome [91] dataset allows training of systems that can classify and describe images, as well as answer questions about objects and their relations in these images.

The labeling of data is a laborious manual task. The creation of ImageNet and Visual Genome involved the use of Amazon's Mechanical Turk [92], which is a paid crowdsourcing service. The platform allows to carry out a large-scale labeling of data by dividing the work into smaller tasks and employing many humans without expert knowledge in the computer vision field.

The object detection algorithm proposed in this paper significantly differs from the ImageNet competition winning algorithms, which are usually based on the CNN. Therefore, the RNN-VDL cannot be trained with such available labeled datasets as ImageNet. A custom training dataset has to be created in order to train the proposed RNN based system. Furthermore, a new set of labeled data has to be created for detection of each type of object. Even the detection of the same object can require differently labeled datasets. For example, one application may demand the detection of all vehicles crossing the detection line, while another needs to detect only vehicles moving in a concrete direction. Therefore, it is important to acquire new datasets quickly and cheaply.

3.1. Format of the Labeled Data for RNN-VDL

The required labeled data for the proposed RNN-VDL method consists of two matrices. The first matrix is a spatio-temporal training image which will be used as an input for training or validation of the learning algorithm. A row of this spatio-temporal image corresponds to the pixels on the detection line at a specific frame. Therefore, the number of rows in the training image is equal to the number of frames in the input video, while the number of columns is equal to the length of the detection line. It is shown in Fig. 3.1 a, where the video is depicted as a box in x, y, and time dimensions. The training image is marked as a plane that slices the video box at the row X_{l} and is parallel to the x-time plane.

An example of the training image is shown in Fig. 3.1 b. While training the RNN model, this image is split into S frame long overlapping sequences. Each sequence needs an appropriate label that indicates the situation on the detection line at the last frame of the sequence.

The second matrix consists of labels for each row in a spatio-temporal image. Thus, the label matrix has the same number of rows as the training image. Each label is encoded as a one-hot

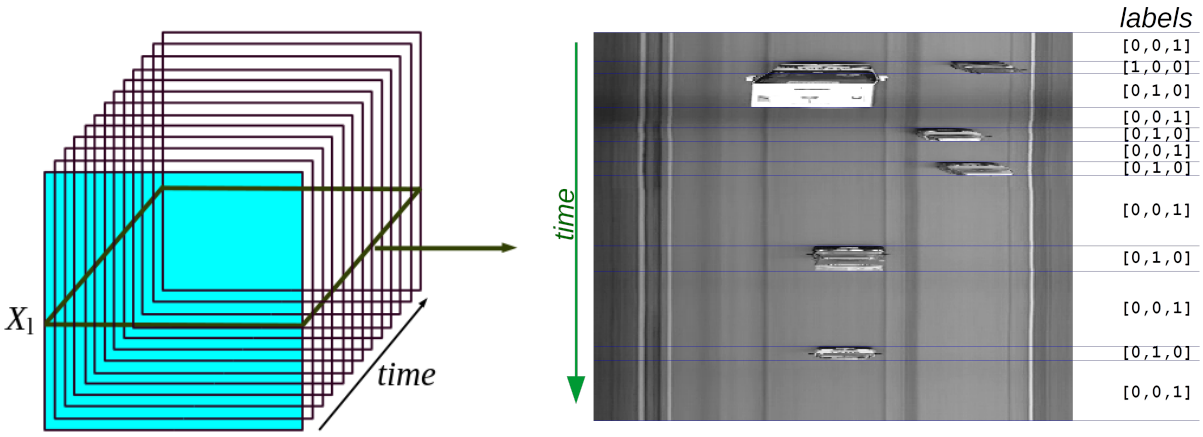


Fig. 3.1. The desired result of labeling.

(a) a slice of the input video that corresponds to a spatio-temporal training image; (b) a spatio-temporal image of vehicles that have crossed the detection line and the corresponding labels of its rows.

vector. In such encoding, only one element of the vector has value 1, whereas all other elements are zeros. The encoded label determines which element is not zero. For example, if there are three possible labels $\{0\}$, $\{1\}$, $\{2\}$, their one-hot encodings are $[0, 0, 1]$, $[0, 1, 0]$, $[1, 0, 0]$. Each column of the label matrix is a one-hot vector, so the matrix has as many columns as there are possible labels.

Fig 3.1 b shows the labels of a spatio-temporal training image. In the example, regions with different labels are separated by thin lines. The first labeling mode is used, so the label of each row encodes the number of moving objects on the detection line in the corresponding frame. The size of the label vector is 3, so in the current example, there are three possible classes – the detection line is being crossed by either none, one or two objects.

3.2. Proposed GUIs for Data Labeling

The conventional labeling of data demands significant human labor. An appropriate interface could make the labeling faster or even more precise. This Thesis develops new graphical user interface-based methods that facilitate the creation of labeled data.

The labeling of the video means the labeling of all its frames. Therefore, the human labeler needs to see the relevant information of each frame in order to make the labeling decision. In some cases, the user needs to see the entire frame to label it correctly. However, in the case of region-based detection approaches, it might suffice to see only the detector part of each frame. In such cases, the effective GUI can display the relevant parts of several frames at once. In this way, the human labeler can go more quickly through the whole video than by inspecting each frame separately.

The consecutive frames in a video are often similar and might often have the same label. So a good interface would allow the user to quickly determine and assign the same label to the consecutive frames.

This Thesis takes into account the mentioned considerations for effective labeling interfaces and proposes two GUI-based labeling methods. The first approach can be used to create a dataset for RNN-VDL training as well as to label video frames for a broader range of machine learning models. The second approach is partly automatic and it is fine-tuned for the detection line-based methods, such as the RNN-VDL. This more specialized labeling approach is not usable for object detectors that process the whole frame. However, it is partly automatic, and its specific GUI makes the human part of the labeling significantly faster than using the fully manual approach.

3.2.1. Manual Labeling

The first proposed method to acquire training matrices is a graphical user interface for manual labeling of each frame.

In the beginning of the labeling process, the GUI displays the first frame of the video. The user has to define the position of the detection line so that it can be drawn on the frame. At each frame, the user has to inspect the detection line and press a number on the keyboard. A label corresponding to this number is converted to a one-hot vector, which is appended to the matrix of labels. The current detection line becomes a new row in the 2D spatio-temporal training image.

After the keypress, the GUI displays the next frame of a video and awaits new user input. If the user holds a number key instead of pressing and releasing it, the video in the GUI is played without interruptions. In this case, the labels corresponding to the pressed key are appended to the label matrix at every frame. Releasing the number key stops the video, and the program waits for the next input from the user. In this way, the method takes advantage of the observation that consecutive frames often have the same label. For example, when there are no objects on the detection line and the user sees that no object approaches the line, the holding of the corresponding label key quickly labels a chunk of consecutive frames. When the user notices an approaching object, the key is released, and following frames are labeled one by one.

This same method can be used for non-detection line based labeling. In that case, the user has to count all the objects in the frame, not only the ones on the detection line. After the user has labeled the frame, the program appends the whole frame to a 3D spatio-temporal matrix, whereas the shape of the label matrix remains the same as it was in the detection line approach.

The proposed method accelerates the creation of labeled data by making it easy for the user to go through the videos and annotate each frame. However, the sequential inspection of all frames still consumes a significant amount of human time and effort. The semi-automatic labeling approach developed in the following subsection takes advantage of the specifics of the RNN-VDL object detector and makes the labeling of data even faster.

3.2.2. Semi-automatic Labeling

The semi-automatic approach begins by showing the user the first frame of the video (Fig. 3.2). The user defines the position of the virtual detection line by pointing to the ends of the line with the cursor. If in the first frame some objects of interest intersect the detection line, the user is advised to fast forward the video by pressing any key on the keyboard. When there are no moving objects on the detection line, the user launches the automatic labeling procedure.

The automatic part of the labeling method is based on the background subtraction, and it offers a choice between two approaches.

The first is partly inspired by the IoVDL object detection method. The first empty detection line is saved as a background line $\mathbf{l}_{\text{bg}}$. The following frames are processed in their original order. The background line is subtracted from the line of the current frame $\mathbf{l}_{\text{g}}$, and the absolute value of the difference is thresholded with a manually chosen threshold, resulting in line $\mathbf{l}_{\text{t}}$.

The background line adapts to the changes in the video. In the proposed labeling method, the background should adapt to the current frame if there are no objects on the detection line. The threshold line $\mathbf{l}_{\text{t}}$ consists of N_{bg} background pixels and N_{fg} foreground pixels. This information is used in the formula (3.1) to either update or retain the background line:

$$\mathbf{l}_{\text{bg}}^{(t)} = \begin{cases} \mathbf{l}_{\text{bg}}^{(t-1)} \cdot (1-E) + \mathbf{l}_{\text{g}}^{(t)} \cdot E & \text{if } N_{\text{fg}}/N_{\text{bg}} < T_{\text{u}} \\ \mathbf{l}_{\text{bg}}^{(t-1)} & \text{otherwise,} \end{cases} \tag{3.1}$$

where E is an adaptation rate with a value range from 0 to 1. The greater the E, the faster the line adapts to the changes in the frame. However, if the method erroneously determines that the current frame is empty, the big value of E quickly results in a wrong background line. The value of threshold T_{u} can be customized to the specific video so that it corresponds to the minimal expected width of the objects.

The second background subtraction approach is based on the Gaussian Mixture-based Background/Foreground Segmentation Algorithm proposed in [93] and [94]. In Gaussian mixture models, the recent history of pixel intensities $(I^{(t-S)}, ..., I^{(t-1)}, I^{(t)})$ is modeled as a mixture of Gaussians. The probability of the current pixel value:

$$\lambda(I^{(t)}) = \sum_{i=1}^{N_{\text{g}}} W_i^{(t)} \cdot \gamma(I^{(t)}, M_i^{(t)}, \mathbf{P}_i^{(t)}), \tag{3.2}$$

where N_{g} is the number of used Gaussians; $M_i^{(t)}$ is the mean value of i^{th} Gaussian at frame t; $\mathbf{P}_i^{(t)}$ is the corresponding covariance matrix; $W_i^{(t)}$ is the weight for each Gaussian so that

$\sum_{i=1}^{N_g} W_i^{(t)} = 1$; and $\gamma(I, M, \mathbf{P})$ is the Gaussian density:

$$\gamma(I, M, \mathbf{P}) = \frac{1}{(2\pi)^{N_g/2}|\mathbf{P}|^{1/2}} e^{-\frac{1}{2}(I-M)'\mathbf{P}^{-1}(I-M)}. \tag{3.3}$$

In this specific implementation, the covariance matrix is assumed to be $\mathbf{P} = P^2\mathbf{M}_1$, where $\mathbf{M}_1$ is an identity matrix.

Each new value of the pixel is used to update the model, specifically to update the weight W (3.4), the mean value M (3.5), and the variance P^2 (3.6) of all the Gaussians:

$$W_i^{(t)} = W_i^{(t-1)} + E \cdot (L_i^{(t)} - W_i^{(t-1)}), \tag{3.4}$$

$$M_i^{(t)} = M_i^{(t-1)} + L_i^{(t)} \cdot (E/W_i^{(t)})(I^{(t)} - M_i^{(t-1)}), \tag{3.5}$$

$$P_i^{2^{(t)}} = P_i^{2^{(t-1)}} + L_i^{(t)} \cdot (E/W_i^{(t)})(a_i' a_i - P_i^{2^{(t-1)}}) \tag{3.6}$$

where $a_i = I^{(t)} - M_i^{(t)}$; E is a learning rate, which limits the influence of older frames. After the update, the weights are renormalized, so that they add up to one.

The current value of the pixel is compared to the existing Gaussians to find which of them are good models. The ownership parameter $L_i^{(t)} = 1$ for a close Gaussian with the biggest W. $L_i^{(t)} = 0$ for all other distributions. The distance is defined by equation (3.7) and the close Gaussians are the ones where D_i^2 is less than a manually set threshold.

$$D_i^2(I^{(t)}) = \frac{a_i' a_i}{P_i^2}. \tag{3.7}$$

If the new value I does not fit to any existing Gaussian, a new distribution number $N_g + 1$ is generated. The initial values of a new distribution are $W_{N_g+1} = E$, $M_{N_g+1} = I$. The initial variance P_{N_g+1} has a manually chosen value. If the maximal M is reached, the Gaussian with the smallest weight is discarded. In the specific algorithm from [94], the number of Gaussians for each pixel of the line may differ and is determined automatically.

Some of the Gaussians of the mixture may correspond to the background. In many cases, they have significant weight and low variance. In the used approach, the Gaussian with the largest W is assumed to be the background. The new pixels, which are closest to this Gaussian, are segmented as background, while other pixels are the foreground.

Both discussed background subtraction approaches result in thresholded lines, which are concatenated into a spatio-temporal image. The width of this image is equal to the length of the detection line. When all frames are processed, the height becomes equal to the number of frames in the input video. In the proposed GUI (Fig. 3.2), the user can observe the creation of the spatio-temporal images. If the first background subtraction approach is used, then one of the visualized spatio-temporal images is the intermediate step after the background subtraction and before the thresholding. Disabling this visualization makes the processing faster.

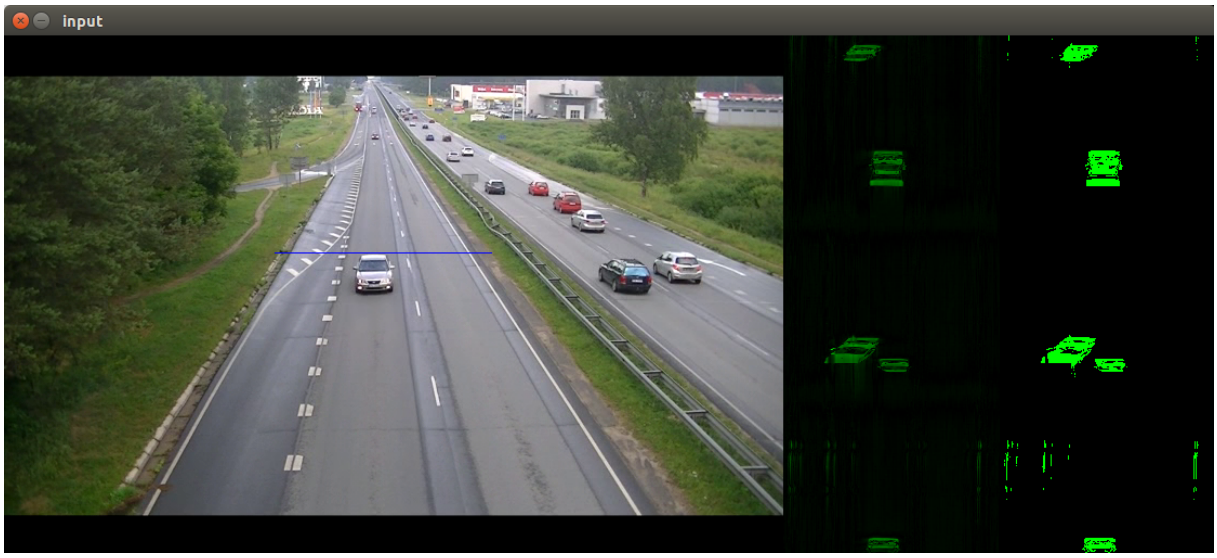


Fig. 3.2. The first window of the semi-automatic labeling GUI.

When all frames are processed, the acquired spatio-temporal threshold image undergoes further processing in order to reduce the impact of the noise. First, the morphological closing with a kernel of size $(3, 3)$ is applied. The closing consists of two consecutive morphological operations – erosion and dilation.

In a set notation, all the white (foreground) pixels of the image can be described as:

$$\mathbf{I}_{\text{fg}} = \{(x, y) | \mathbf{I}(x, y) = 1\}. \tag{3.8}$$

The erosion of binary image $\mathbf{I}$ and a kernel $\mathbf{K}$ is described by equation (3.9).

$$\mathbf{I}_{\text{fg}} \ominus \mathbf{K} = \{(x, y) \in \mathbb{Z}^2 | (x + u, y + v) \in \mathbf{I}_{\text{fg}}, \forall (u, v) \in \mathbf{K}\}. \tag{3.9}$$

In our case the $\mathbf{K}$ is a 3×3 size square, which is applied at every possible position in the image. If the square touches at least one background pixel in the image, the image's pixel under the center of the $\mathbf{K}$ becomes a background pixel. The opposite operation to erasion is dilation:

$$\mathbf{I}_{\text{fg}} \oplus \mathbf{K} = \{(x + u, y + v) | (x, y) \in \mathbf{I}_{\text{fg}}, (u, v) \in \mathbf{K}\}. \tag{3.10}$$

In this case, the pixel (x, y) is white in the resulting image, if the kernel at this position touches at least one white pixel in the image.

The closing operation, which combines erosion and dilation, removes small holes in the white regions of the spatio-temporal image. The closing is followed by the morphological opening with a kernel $(5, 5)$. The opening consists of dilation followed by closing, so it removes the small white objects in the image. After the morphological operations, the values of all background pixels fully surrounded by the foreground pixels are changed to 1.

Next step is the automatic labeling of the processed spatio-temporal threshold image. Continuous white regions are detected at each row of the image. If such regions are wide enough, then they most probably correspond to the objects on the detection line. The number of the white segments longer than the minimal expected width of objects $minWidth$ becomes the temporal label of each row. These labels correspond to the first proposed mode of labeling.

The automatic stage of the labeling method does not consume human time, but it is also less accurate than humans. The next labeling stage consists of GUI (Fig. 3.3) which allows fast manual detection and correction of the errors made in the automatic stage. Fig. 3.3 a depicts a region of a spatio-temporal image which was successfully labeled by the automatic process. Each blue line in Fig. 3.3 depicts a frame at which the label on the detection line changes. The number on the right of each dividing line is the corresponding label of the rows under this line.

In Fig. 3.3 b, another region of spatio-temporal image is depicted. In this region, automatic labeling has made two errors pointed out by the red circles (these circles are not part of the GUI).

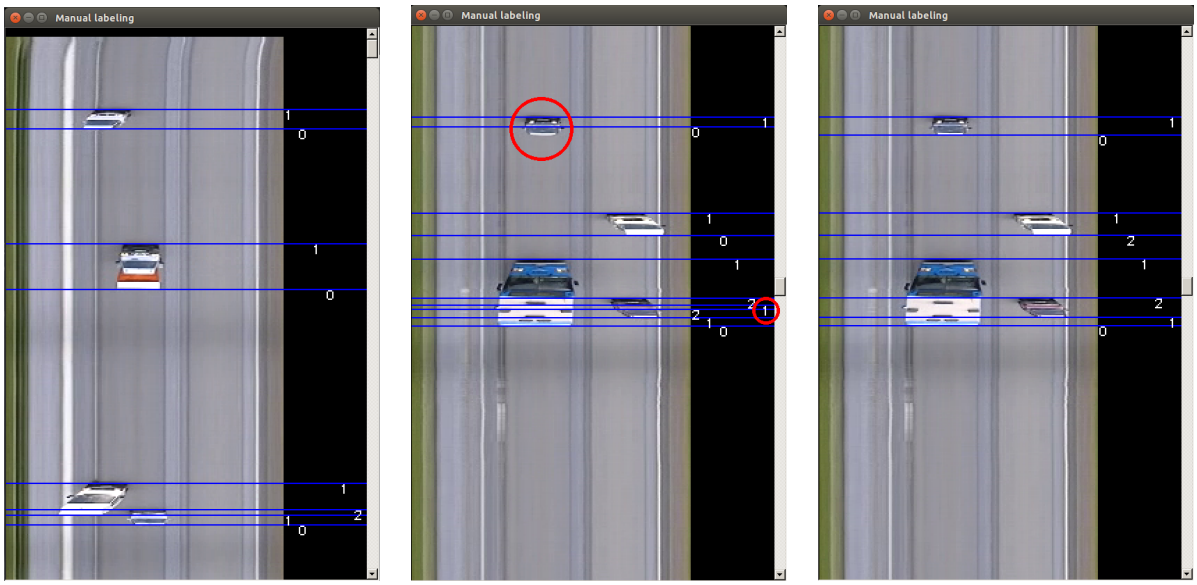
The proposed GUI allows the user to scroll through the whole labeled spatio-temporal image and correct the errors with the mouse and keyboard. The human labelers see the spatio-temporal image, therefore they can inspect many frames simultaneously. In this way, the speed of the manual part of the labeling is significantly increased compared to the methods where each frame is labeled separately.

The error correction interface is made convenient by allowing the labeler to point out the location of interest with the cursor and by applying different correction operations, depending on which mouse button is pressed. A typical computer mouse has two buttons, but there are more than two operations that a labeler may want to carry out. The proposed solution is to use the same button to carry out different operations depending on whether the closest label line is above or below the cursor's current position.

The error at the top of the image in Fig. 3.3 b can be corrected by moving the line with the label {0} downwards so that it corresponds with the ending of the vehicle. The new position of the label line is pointed out with the cursor, while the pressing of the right mouse button indicates the specific type of operation. If the closest line to the cursor is upwards, then the operation is to change the line's position to the current position of the cursor.

Conversely, if the closest line is located below the cursor, then this line is deleted. The labels under the now deleted line become equal to the labels above. Therefore, in order to correct the second mistake of the automatic labeling in Fig. 3.3 b, the user has to point the cursor close and above the line with the label {1} and then click the right mouse button. In the specific example, two lines will disappear. Since all the labels under the line labeled as {1} will become {2}, the next line, which previously divided the region of labels {1} from the region with labels {2}, becomes redundant and is automatically removed. The corrected labels are shown in Fig. 3.3 c.

The labeling GUI also offers a convenient way of adding new label lines. To set the label for the line that one wants to add, the user presses the corresponding number key on the keyboard.



(a) (b) (c)

Fig. 3.3. GUI for inspection and correction of errors of automatic labeling.

Then a left-click adds a new dividing line at the position of the cursor.

When the user has inspected and corrected the whole spatio-temporal image, the labels are encoded as one-hot vectors, and they are saved along with the spatio-temporal image.

3.2.3. Different Labeling Modes in Semi-automatic Labeling

As was described in Subsection 2.4.2, the labeling in Fig. 3.1 is only one of several possible kinds of labeling for RNN-VDL. In this mode, the label of each frame corresponds to the number of moving objects on the detector.

The developed semi-automatic labeling approach is also capable to return two other kinds of labels. The automatic part of the program is the same as in Subsection 3.2.2, but now it is followed by a step that converts labels from mode 1 to mode 2. In the second mode, only the frames with at least one object leaving the detection line have a label other than $\{0\}$. This conversion takes the matrix of labels as an input. It analyzes the rows starting with the first and moving downwards. If the temporal label of the row i is larger than the label of $i+1$, then the final label of the row $i+1$ becomes equal to the difference of labels i and $i+1$. The final labels of all other rows become $\{0\}$.

The manual part of the method also changes. The pressing of the space bar switches between the two modes of manual error correction. In the second mode, the left-click above the dividing line moves this line one row upwards. A left-click below the line deletes this line, and the rows under this line get the same label that was above the now deleted line.

The right-click creates a new dividing line, but in order to accelerate the labeling process,

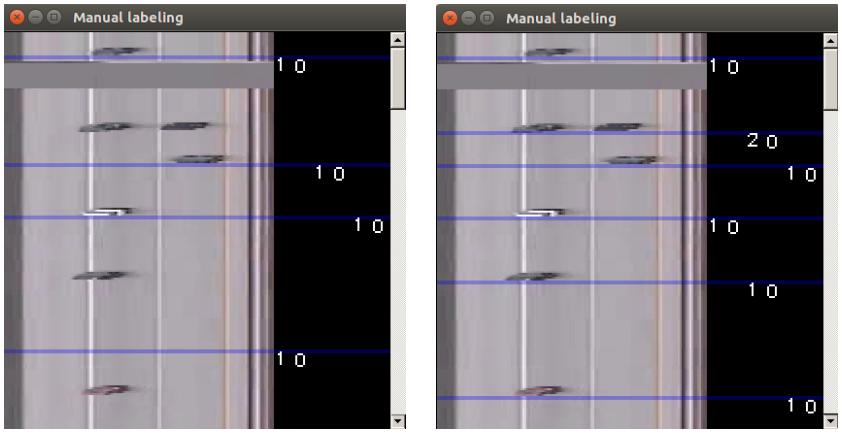
the precise position of this new line is determined automatically. A small region of the selected row is sequentially compared to the same regions on the neighboring rows above. The algorithm places the new dividing line where the difference between original row and some upward row is bigger than a set threshold. This process allows the user to point somewhere under the object and the program automatically puts the dividing line right under the object in the spatio-temporal image. If this position already has a line with the label $\{a\}$, then the label changes to $\{a+1\}$.

The results of the automatic and manual labeling of mode 2 are shown in Fig. 3.4. In this example, the dividing line with the label $\{2\}$ was created by the user by right-clicking once somewhere under each of the two objects that this label corresponds to.

The label mode 3 can be automatically acquired from the label mode 2, so no additional changes in the manual part of the proposed labeling method are necessary.

3.2.4. Implementation of Labeling GUIs

Both GUI-based labeling approaches are implemented in Python 3 language. The code for the manual labeling program is given in Appendix 1. It uses *OpenCV* and *numpy* libraries. The current implementation allows three different labels for each frame, but more can be easily added, if necessary.



(a) (b)

Fig. 3.4. The GUI for semi-automatic data labeling in second labeling mode.

(a) result of automatic labeling containing two mistakes; (b) manually corrected labeling.

The code for the developed semi-automatic GUI is given in Appendix 2. It depends on the *numpy*, *OpenCV*, and *PyQt4* libraries. This implementation includes the proposed choices of two background acquisition methods in the automatic part of the program and two labeling modes in the manual part of the program.

Fig. 3.4 demonstrates additional advantage of the semi-automatic labeling approach. Near the top of the spatio-temporal image, there is a rectangular gray region. It corresponds to an

artifact in the input video, when several consecutive frames are blank. The used labeling mode demands the labeling of only those frames, where at least one object has left the detector. All other frames, including the section of the artifact, have a default label {0}. Therefore, if one wishes to train an RNN-VDL model that is insensitive to different video artifacts, one only needs to get input videos with such artifacts but does not have to specifically label such occurrences.

An additional benefit of the implemented labeling methods is that the labeled data for the RNN-VDL is significantly smaller than the unlabeled input video. The video is converted to a spatio-temporal image which does not contain any pixels that are not on a detection line.

3.3. Generation of Labeled Data Using a 3D Game Engine

The semi-automatic labeling still consumes human labor, so in this subsection a fully automatic labeling approach is examined. Recent literature in the field [95]–[98] shows a potential solution for dealing with the lack of labeled data, which is to train the DL models on synthetic data.

Computer vision can benefit from the increasingly realistic computer-generated imagery, where each object has a defined label and a known position in the 3D scene. For example, authors of [95] used a 3D game engine to create scenes with a tower made of blocks. The physics simulation determines if the tower will fall. A CNN was trained only on these synthetic images and learned to distinguish the stable and unstable towers. Tests proved that this network also learned to predict the stability of similar simple towers in real-world photographs. A similar study and conclusions about the physics simulation of simple rectangular objects are presented in [96].

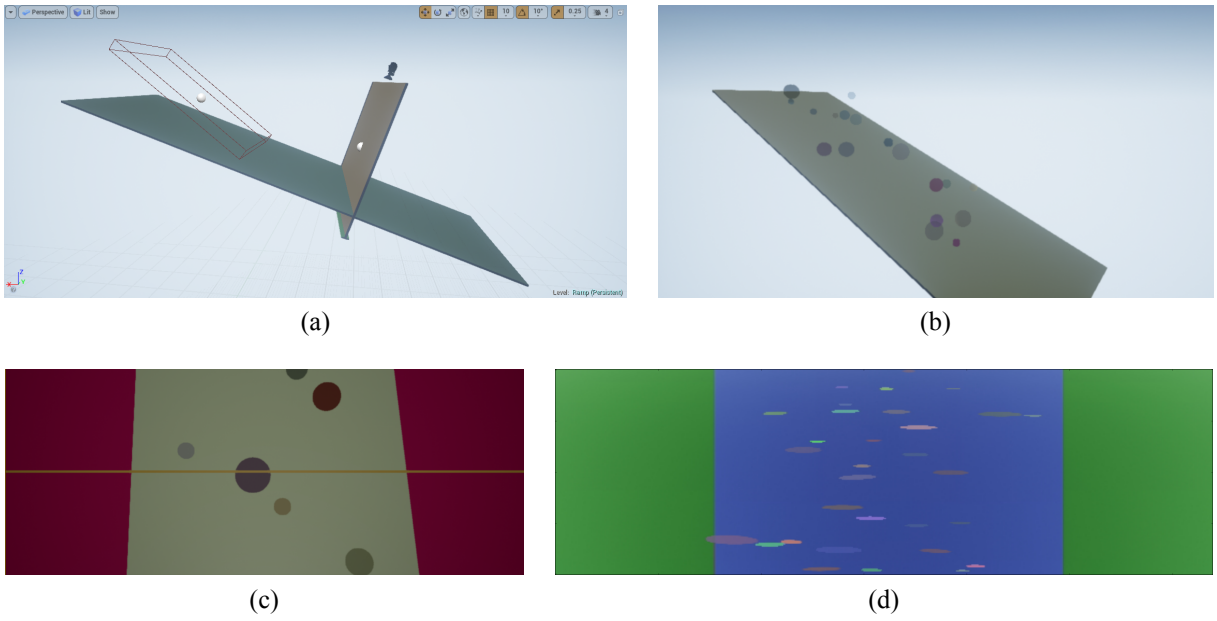
The authors of [97] successfully used synthetic data to improve the performance of a segmenting and tracking network for self-driving vehicles. The network was pre-trained on virtual videos that included diverse weather and lighting conditions. The further training was done with real labeled data. The pre-trained model performed better than the model trained entirely on real videos. The research resulted in a computer generated Virtual KITTI dataset. Another example includes work described in [98], which used a 3D CAD modeling simulator and reinforcement learning to teach a flying drone to avoid collisions. No real world data was used in the training.

The proposed RNN-VDL method could also benefit from the generated data, so a simulation environment was created in the Unreal Engine 4, which is a 3D game engine. The aim of this environment is to create data for a simple task – the detection of moving spherical objects.

The setup of this simulation is shown in Fig. 3.5 a. This environment consists of an inclined base plate, sphere generator object, detection plate, lighting element, and camera. The generator object is a transparent volume which generates randomly sized and colored spheres. The spheres appear at random moments in time and at random points inside the generator's volume. The new spheres are falling down due to the gravitational force of the physics engine. The fall is stopped

by the base plate, and since it is inclined, all spheres are moving towards the detection plate. This plate is perpendicular to the base plate and parallel to the direction of the camera.

The detection plate allows the spheres to pass through, but it detects the beginning and the end of the collision with each sphere. This information is used to label each frame. As was discussed previously, different labeling modes are possible. If the detection mode corresponds to the number of objects on a detection line, then the simulation's label for the current frame is the number of spheres currently colliding with the detection plate. At each frame of the simulation, this label is saved to the hard drive along with the frame from the camera.



(a) (b)

(c) (d)

Fig. 3.5. Generation of training data with a 3D game engine.

(a) the simulation setup; (b) simulation in process; (c) view from the camera; (d) the resulting spatio-temporal training image.

From the point of view of the camera, the detection plane looks like a line (Fig. 3.5 c), which corresponds to the virtual detection line in the RNN-VDL method. When the simulation is running, the detection plate and the sphere generator are invisible to the camera, as shown in Fig. 3.5 b. Only the middle row of the resulting camera frame is needed to train the proposed object detector, so only this line is saved along with the corresponding label. Separately from this simulation, a Python script (Appendix 3) takes these saved rows and creates a spatio-temporal training image (Fig. 3.5 d). This code also creates additional training examples by adding a Gaussian noise to the created spatio-temporal image.

The more diverse examples the model sees during the training, the better it might generalize on unseen data. In order to add variety to the simulation, the lighting of the scene slowly changes direction and intensity. Also, the color of the base plate is changing. These changes are smooth, but in a random direction on the control sliders for intensity and color channels.

The simulation environment created in the Unreal Engine 4 works with simple objects - spheres. This proof of concept shows that training data for the proposed RNN-VDL method can be acquired fully automatically. In order to use this simulation for real-world applications, the simple models of the objects and the environment would have to be replaced with more realistic ones. In some applications, the main setup of the simulation could remain similar to the one proposed here. For example, in vehicle detection, the highway scene can be simulated in a similar manner to the rolling sphere simulation. On the other hand, a scene with moving people is much more difficult to simulate, and it may require sophisticated animations of human movement and gait.

3.4. Summary

The practical use of the proposed machine learning-based object detector (RNN-VDL) depends on the availability of labeled training data. This section proposed different methods for the acquisition of such data, beginning with manual labeling and ending with a fully automatic approach. Although these solutions for faster data labeling were developed for a specific RNN based method, the proposed ideas might be of interest to a broader machine learning community, especially with respect to challenges that arise when dealing with temporal and sequential data.

Among the proposed methods, the GUI for manual labeling of each frame is the simplest to implement. It can be used to label data for a broader range of object detection methods, not just virtual line detectors. However, it is still a slow labeling approach that demands sizeable human effort. The fully automatic approach uses a 3D game engine and can generate a large amount of varied and certainly correctly labeled data. Since the computer graphics are getting more and more lifelike, in the near future this approach may become important for machine learning-based computer vision. The time-consuming part of this approach is the creation of an appropriate simulation environment. Unlike the simple manual labeling, it also requires specific human skills to create realistic and practically usable data.

Out of the three proposed approaches, the semi-automatic labeling GUI turned out to be the most convenient data acquisition method for the RNN-VDL object detector. It uses the ideas and experience gained from the non-machine learning object detection approach IoVDL. First, it tries to label the data automatically. Some settings, such as the threshold value after the background subtraction, are modifiable by the user. Thus, the automatic part of the method can be customized for different types of video (for example, day or night). The errors of the automatic labeling can be conveniently inspected and corrected by the user using a novel GUI. Using the semi-automatic approach, a single labeler is able to prepare the necessary data for the training and testing of the proposed RNN based object detector, which is shown by the tests described in the following section.

4. APPLICATION AND EXPERIMENTAL RESULTS

The computer vision approaches proposed in Section 2 are well-suited for vehicle detection on the roads. A typical capability of the Intelligent Transportation System (ITS) is the detection and counting of vehicles on a specified segment of the road. The use of image processing and computer vision for this task has been researched for more than thirty years [32], [83], [90]. A survey paper [77] offers a comprehensive overview of the different approaches proposed before 2003, while the approaches of later years are discussed in a survey [99]. However, it is still a challenge to make computationally efficient methods that accurately detect vehicles in the changing outdoor environment. Additional difficulties arise because vehicles may significantly differ from each other, they may be moving or static, they may not stand out from the background, and they may occlude each other. Thus, the CV field continues to produce more efficient, robust, and capable methods. The methods developed in Subsections 2.2–2.4 aim to be both efficient and adaptable.

Another desirable feature of traffic monitoring systems is the ability to measure various parameters of the detected vehicles (movement direction, speed, size, trajectory). The classification of vehicles is also important for ITS [25], [84]. For example, the distinction between light and heavy vehicles helps to improve traffic planning and road pavement maintenance. It also allows different charges to be applied for trucks in auto-toll systems. And it can be used to warn too high vehicles before bridges.

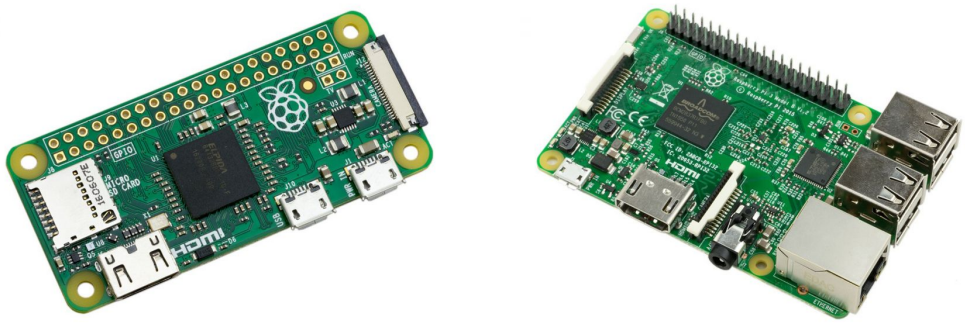
The methods developed in this work can detect vehicles as well as distinguish trucks from light cars. This section aims to test if the proposed computationally efficient methods are accurate in the practical ITS related applications.

4.1. Efficient Vehicle Counting with IoVDL

4.1.1. Computational Efficiency

The IoVDL method (Subsection 2.2) is implemented in C++ programming language. It uses image processing library *OpenCV* for video input/output and common image processing tasks. One of the ambitions of this Thesis has been to develop the methods that can be used on low-cost devices with limited computational power. In order to prove the efficiency of the proposed IoVDL algorithm, it is implemented on a Raspberry Pi Zero single-board computer. The Zero (Fig. 4.1 a) is the smallest ($65 \times 30 \times 5$ mm) and the cheapest ($\sim$5 EUR) Raspberry Pi model, with a 1 GHz single-core CPU and 512 MB RAM.

Figure 4.2 shows how many frames per second the IoVDL can process while running on Raspberry Pi Zero. During the test, the frames are acquired in real time by a Raspberry camera module V2, which is able to catch up to 90 low-resolution frames per second. Frame acquisition takes a significant portion of the running time, which depends on the frame resolution. There-



(a) (b)

Fig. 4.1. Raspberry Pi Zero and Raspberry Pi 3 Model B [100].

fore, Fig. 4.2 shows the speed measurements at three different resolutions: 320×280, 640×480, 800×600.

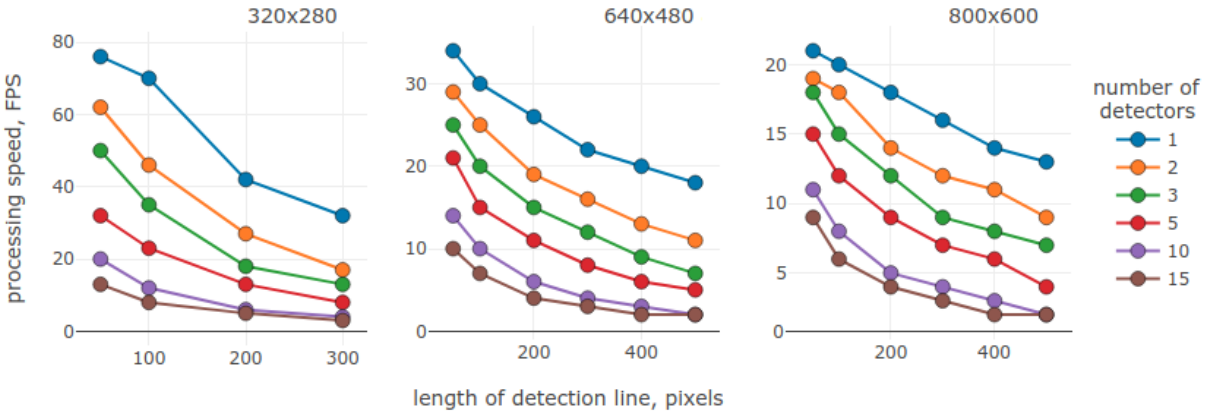
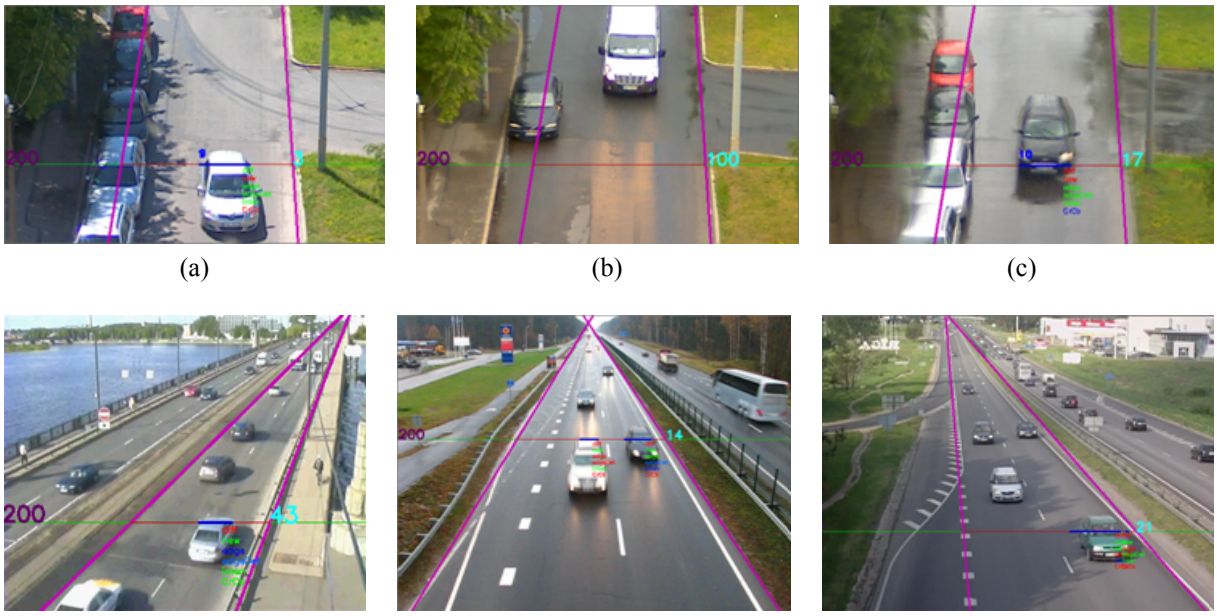


Fig. 4.2. IoVDL processing speed on Raspberry Pi Zero.

4.1.2. Vehicle Counting Accuracy

In order to assess vehicle counting accuracy of the IoVDL method, a single detection line is tested on six different real-life videos. The first three videos (Fig. 4.3 a,b,c) were taken by the same camera in different weather and lighting conditions. Video (a) was taken in daytime when the movement of shadows caused significant and rapid changes in the background intensity. Video (b) was taken shortly after rain when the wet road caused strong reflections of the headlights. Video (c) was recorded during a heavy rain. The resolution of these videos is 640×480. The traffic intensity is small, and the number of occlusions is not significant. The remaining videos (Fig. 4.3 d,e,f) include roads with several traffic lanes and more intensive traffic. The results of the tests on these videos are shown in Table 4.1.

The accuracy in the result's table shows how closely the vehicle detection by the IoVDL cor-



(a) (b) (c)

(d) (e) (f)

Fig. 4.3. Test videos for vehicle counting using IoVDL.

responds to the ground-truth data. The vehicles missed by IoVDL as well as the false detections reduce the accuracy. Table 4.1 also shows the type of each error.

The tests indicate two principal sources of error. In some occlusion cases, the two or more vehicles are counted as a single object. An opposite error occurs when the IoVDL algorithm counts a single vehicle as two or more. This problem occurs when an interval closes before the car entirely passes the detection line. In such situation, an additional interval may be created for the already counted vehicle. The other observed but less common sources of error are people and birds crossing the detection line and the shadows of the vehicles.

The tests show that the proposed method is robust to changes in lighting. The moving shadows of clouds and trees are not detected as vehicles. The heavy rain does not significantly affect the performance of the IoVLD. Also, the headlight reflections on the wet road do not create additional problems.

An additional conclusion from the tests is that the counting accuracy of the proposed detection method depends on the placement of the camera. Vehicle occlusion caused one-third of the detection errors. Therefore, a good camera position would be directly above the road facing downwards, so that closely following vehicles do not occlude each other in the frames. However, in such placement, the camera sees only a small segment of the road. So in practice, the camera is often tilted towards the horizon. This way one can use the same camera to monitor a wider area of the road and track the vehicle movement.

The influence of camera placement is especially pronounced in test (d). The camera is not only tilted towards horizon but is also placed on the side of the road. In this video, vehicles from the closest lane sometimes occlude the ones on the neighboring lane, which explains why the

IoVDL method performs worse in test (d) than in other tests.

As was discussed in Subsection 2.1, there are other efficient but less flexible methods that use a small detection region to count objects in a video. The papers proposing these methods report similar test results. In [81], the counting accuracy for daytime videos varies from 70.31 % to 98.39 %. The method in [84] has the accuracy of 97.73 %. In [82], the precision of the method is 86 %–96 %, while the recall is 89 %–96 %. Since different videos are used to evaluate all the methods above, the accuracy rates are indicative but not conclusively comparable.

Table 4.1

Vehicle Counting with IoVDL

Video	Ground-truth	Detected	Errors			Counting accuracy
			occlusion	counted as two	other	
a)	100	105		5		95.0 %
b)	100	105		4	1	95.0 %
c)	100	100	3	3		94.0 %
d)	77	71	7	3	2	84.4 %
e)	42	44			2	95.2 %
f)	48	46	2			95.8 %

4.2. Vehicle Classification

In Subsection 2.3.3, the proposed IoVDL method is extended so it can detect an object's parameters. The measured size parameters can be used to classify the object, which is a useful ability for the highway monitoring systems.

4.2.1. Existing Classification Approaches

In [84], vehicles are classified by measuring their length. It is then compared to predefined values, which divide the vehicles into different classes. The method in [88] measures and compares three vehicle dimensions.

The classification criteria may also be learned using machine learning. The evaluation of the measured parameters may be accomplished by different classifiers, such as support vector machines, k-nearest neighbors, decision trees and others. In [87] and [101], the neural networks are used to classify vehicles into three categories based on their size.

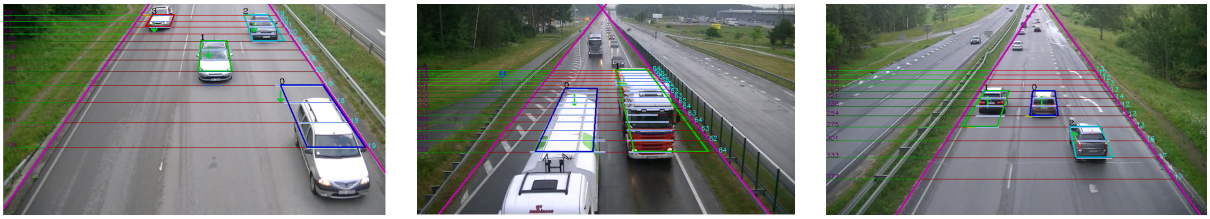
Model-based approaches proposed in [102] and [103] are vehicle tracking methods, which also include the classification ability. Each vehicle is compared to a model of each possible class. For the precise classification of the vehicle type, it may not be practical to have a detailed geometric model for each possible vehicle, so the methods use deformable 3D mesh models. For example, [104] uses deformable models, which align to the shape of the detected vehicles.

Although these deformable models can be both detailed and generic, such approach is computationally demanding.

Similarly to the simpler approaches, the method proposed in this Thesis can distinguish a limited number of vehicle classes (light vehicle or truck). The added classification capability of the IoVDL conforms to the overall objective of this paper, which is the development of efficient algorithms which are useful in such applications as Intelligent Transportation Systems.

4.2.2. Classification with IoVDL

The extended version of IoVDL developed in Subsection 2.3 uses several detection lines to track and classify vehicles. The classification capability of the extended IoVDL is tested on several videos with different weather conditions and different camera positions. Example frames from the tests are shown in Fig. 4.4.



(a) (b) (c)

Fig. 4.4. Tests of extended IoVDL.

Each of the nine test videos is approximately seven minutes long. The roads in these videos have 2–3 lanes. In the tests, the extended IoVDL consists of 12–16 detection lines. In the real world plane, the detection lines have a spacing of 1–3 meters and evenly cover a portion of the road. In most of the videos vehicles are moving towards the camera, while in video (c) the vehicles are moving in the opposite direction. It is raining in videos (f, g, h).

Table 4.2 depicts the counting and classification results of the extended IoVDL method. A single virtual detection line counts the vehicles based on the creation and closing of the intervals on this line. The extended method consists of several such IoVDL detectors, and the vehicle is counted, when the intervals from different detectors form a virtual object.

The Table compares the vehicle counting of a single IoVDL detector and the extended IoVDL. The middle detection line of the extended method is chosen as a single line detector. Additionally, Table 4.2 depicts the counting result of a more straightforward combination of all detection lines, where the final count of vehicles is the mean value of the counts returned by the separate detectors. The comparison of results shows that the proposed extension of IoVDL returns closer counts to ground-truth than a single IoVDL detector. The average count of all the separate detection lines is also less accurate than that of the extended IoVDL.

Table 4.2

Traffic Monitoring with Extended IoVDL

Video	Ground-truth		Detection			Classification	
	all vehicles	trucks	Single IoVDL	IoVDL average	Extended IoVDL	TP trucks	FP trucks
a)	32	4	30	30	32	4	0
b)	122	8	121	124	122	8	0
c)	159	8	168	164	158	7	1
d)	43	0	45	44	44	0	0
e)	221	12	232	235	231	7	0
f)	208	5	209	209	209	5	1
g)	264	12	233	239	239	12	3
h)	117	4	118	118	116	4	1
i)	88	7	92	93	88	6	0

Table 4.2 also depicts the classification accuracy of the algorithm. It shows the actual number of vehicles in different videos, and how many of those vehicles are trucks.

The classification is based on the detected size parameters of the vehicle. If two out of three measured dimensions of the vehicle indicate that the detected object is a truck, then this is also the final verdict of the algorithm. In the test videos, the algorithm correctly classifies 88 % of the trucks (TP), while the number of false positives (FP) is small.

In order to improve the accuracy of vehicle counting and classification, the number of detection lines can be increased. More closely located lines may improve the accuracy because it is more probable that the space between closely following vehicles is noticed, and these vehicles are not counted as a single car. However, each additional line increases the needed computations.

4.3. Vehicle and People Counting with RNN Based Line Detector

4.3.1. Training the RNN-VDL

The proposed RNN based object detection and data labeling methods are implemented in Python 3, using the TensorFlow framework. Appendix 4 shows the code that defines the TensorFlow graph of the RNN network. The nodes of the graph include the placeholders for the input vector **x** and label vector **y**, and the variables for weights **W** and biases **b**. After the definition of all the layers of the network that computes the predicted label $\hat{\mathbf{y}}$, the code in Appendix 4 also defines the cost function, which compares labels **y** and $\hat{\mathbf{y}}$. The particular function used to train RNN-VDL is $softmax\_cross\_entropy\_with\_logits(\hat{\mathbf{y}}, \mathbf{y})$. It applies the softmax function:

$$\mathbf{y}_{\text{sm}}(j) = \frac{e^{\mathbf{y}(j)}}{\sum_{i=1}^{N_\text{c}} e^{\mathbf{y}(i)}}, \quad j = 1, 2, ..., N_\text{c}, \tag{4.1}$$

where N_c is the size of the vector $\mathbf{y}$ (number of classes). Each element in the output vector of the softmax function has a value in the range $[0, 1]$, and the sum of all elements is 1.

The same TensorFlow function also measures the distance between the predicted and the actual labels. It uses the cross-entropy loss:

$$\delta_{\text{ce}}(\mathbf{y}_{\text{sm}}, \hat{\mathbf{y}}_{\text{sm}}) = -\sum_{j=1}^{N_\text{c}} \mathbf{y}_{\text{sm}}(j) \cdot \log \hat{\mathbf{y}}_{\text{sm}}(j). \tag{4.2}$$

The goal of training the network is to find weights $\mathbf{W}$ and biases $\mathbf{b}$ that minimize the cost δ_{ce}. As can be seen in Appendix 4, the minimization is accomplished by the Adam optimizer [105].

Appendix 5 shows the code for training the defined network. In this code, the user can choose to load the weights of an already pre-trained network. Otherwise, the weights of the network are automatically initialized with small random values.

The labeled datasets are acquired by the semi-automatic labeling method discussed in Subsection 3.2.2. The data is further prepared for training. The $data\_preparation$ module that is used by the training code, is shown in Appendix 6. It loads the training spatio-temporal images (frames and labels), normalizes them, and resizes the input row of pixels so that it can be fed into the previously defined neural network. The code randomly divides the data into training and validation sets, where the user chooses the corresponding ratio.

This module also includes an optional data balancing method. It is used in the case of data labeling mode 2 when the frame label depicts the number of objects that have left the detection line in this frame. Such labeling often leads to a dataset, where most of the labels are $\{0\}$. In such cases, the network training can quickly achieve high accuracy by always predicting class $\{0\}$. The balancing method makes additional copies of the rare class examples so that at the training time, the different classes have a similar number of samples.

The training code in Appendix 5 also divides the data into batches. The examples in a batch are processed in parallel. The processing of all batches is called an epoch. At the end of each epoch, the program determines the loss of the model and its accuracy on the training and validation data. The progress of these values can optionally be saved as a line graph.

4.3.2. Object Counting Accuracy

The aim of combining RNN and a detection line approach was to develop an efficient detector that can be retrained for detection of different kinds of objects. In order to demonstrate the versatility of RNN-VDL, the method is applied to two applications – vehicle and people

counting.

Vehicle counting network needs a substantial amount of video data for training and testing. Unlabeled highway videos used in this Thesis were acquired from open traffic streaming cameras [106] and [107]. Example frames are depicted in Fig. 4.5 a,b. They include different lighting and weather conditions. Both cameras acquire 25 FPS.

(a) (b) (c)

Fig. 4.5. Videos used for testing the RNN-VDL method.

Two frames from open traffic cameras and a frame from a different camera that was not used in the training of the RNN-VDL model.

The videos from the first camera have a resolution of 650×480, the second – 352×240. The videos are in poor quality and include many compression artifacts. For more convenient labeling, the video stream is downloaded and stored in 1.5–2 minute long videos. The detection line for each short video is set manually and differs in length. It varies between 200 to 400 pixels in the videos from the first camera and 70 to 150 pixels from the second camera. Altogether 427 short videos are labeled. 100 of those videos are separated as test-set and used only to get the final results (Table 4.3). The rest of the videos are further divided into training (70 %) and validation (30 %) data.

After 6 hours of training accelerated by a single NVIDIA Tesla K20 GPU, the network learns to correctly classify 99.4 % of training sequences and 95.6 % of validation sequences.

While training, the network tries to correctly classify each frame. In the practical applications, it is often not crucial to count the moving object at the exact frame as it is marked in the labeled data. Furthermore, the labeling also includes subjectivity of the labeler. If the model detects the object two frames later than human and no additional false objects are detected, the result is still valid for practical applications. Therefore, to test the performance of the trained model, the following testing procedure is implemented.

The labels of the test set are acquired in the same way as the labels for training data. These labels are stored as a $N_z \times N_c$ matrix, where N_z is the number of frames and N_c is the number of possible classes. All frames with the label greater than 0 correspond to the detection of at least one vehicle.

When the test video is forwarded through the trained network, the test algorithm chooses

Table 4.3

Object Detection Results Using RNN-VDL

Camera	Vehicles 1		Vehicles 2		People
	RNN-VDL	IoVDL	RNN-VDL	IoVDL	RNN-VDL
Ground truth count	4238	4238	121	121	285
Predicted count	4278	3693	108	120	245
True Positives	4059	3623	103	119	209
False Positives	219	70	5	1	36
False Negatives	179	615	18	2	76
Precision	0.949	0.981	0.954	0.992	0.853
Recall	0.958	0.855	0.851	0.983	0.733
F_1 score	0.953	0.914	0.900	0.987	0.788

the same detection line in the frame that was used for labeling. The predicted classes are also saved in a $N_z \times N_c$ matrix. Corresponding pairs of detections are found in the predicted and labeled matrices. The pairs are not allowed to be more than 25 frames apart (one second in the used videos). If frames with labels greater than $\{1\}$ do not have a corresponding pair, then each detection of an object contributing to this label can be paired with a separate detection in the other matrix. For example, if the prediction matrix has a frame with label $\{2\}$ which has no corresponding $\{2\}$ in the label matrix, then the $\{2\}$ from one matrix can be paired with two frames with labels $\{1\}$ from another matrix. If the model counts vehicles without any errors, all detection events in the predicted matrix have a corresponding pair in the labeled matrix. Otherwise, all left-over detection events in the prediction matrix are false positive detections, while all unpaired events in the label matrix correspond to false negative detections.

Table 4.3 compares the vehicle detection results of the RNN-VDL and IoVDL methods on different videos. The Table shows the actual number of objects in the test videos and the number returned by the methods. By using the semi-automatic labeling method and the testing procedure described in the previous paragraph, it is possible to acquire the type of every error. Accordingly, Table 4.3 depicts the precision ($TP/(TP+FP)$), the recall ($TP/(TP+FN)$), and the F_1 score for each test:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{4.3}$$

The test videos in column *Vehicles 1* were acquired by the same highway cameras that recorded the training videos for the RNN-VDL (Fig. 4.5 a,b). The training and testing videos were recorded on different days. According to the F_1 score, the RNN-VDL method performs better on these videos than the IoVDL. The machine learning approach makes a similar amount of both kind of errors, so the returned overall count of the vehicles is close to the ground truth count. In turn, the IoVDL method rarely makes false positive detections but misses many cars.

When the IoVDL algorithm was developed, it was mostly designed to work on videos that were similar in resolutions and quality to the one used in the test *Vehicles 2*. Unsurprisingly,

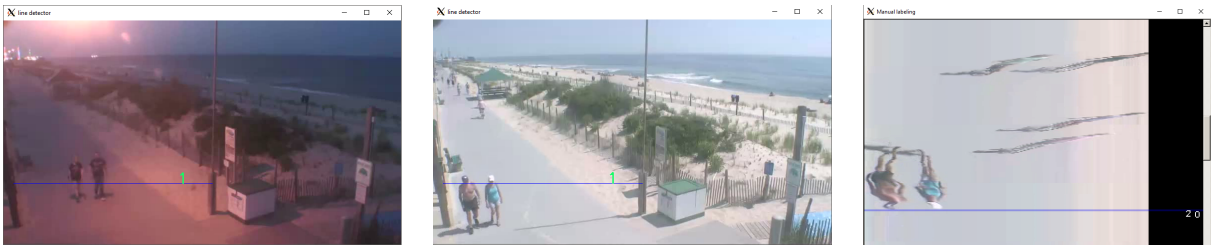
IoVDL performs well on this test.

The primary goal of the additional test *Vehicles 2* is to determine if the trained RNN-VDL model can be used on videos from previously unseen cameras without the additional finetuning. The test video has a different resolution (700×480) and fewer artifacts than the training videos. The length of the detection line is 456 pixels, which is significantly longer than in training.

The performance of RNN-VDL on the test video *Vehicles 2* is worse than on *Vehicles 1* but still usable in some applications. The neural network has learned some generalization of vehicle counting. The recommended way to use the trained model on an entirely new road is to acquire and label a small dataset on the target location with the particular camera. The small dataset might be enough to fine-tune a network that is pre-trained on other data. This idea is supported by the literature on transfer learning in deep networks [69], [108].

The tests show that both proposed detection methods have a similar precision and recall to those of the alternative virtual detector based methods.

People counting videos were also acquired from an open camera stream [109]. Fig. 4.6 a,b. shows example frames at night and daytime. 517 2-minute videos were labeled and divided into train and test sets (463 and 108 videos respectively). The resolution of the videos is 800×450 and the frame rate is 15 FPS.



(a) (b) (c)

Fig. 4.6. Detection of people using RNN.

In order to show the additional benefits of using the recurrent network, the people counting task is modified so that only people going in one direction are counted. Therefore, the single detection line has to ignore the people that are crossing the line from bottom to top. The corresponding labeling can be seen in the labeled spatio-temporal image in Fig. 4.6 c. This example is a snapshot of the semi-automatic labeling process proposed in Subsection 3.2.2. Although six people have crossed the line in the shown time, only two have passed the line in the correct direction. They left the detector simultaneously; therefore, the labeler has labeled only one frame with the label $\{2\}$.

Along with the different training data, the only other difference between the vehicle and people detectors is the length of the analyzed frame sequence. The RNN of the people detector bases its current prediction on a larger amount of past frames than that of the vehicle detector.

This choice is based on the observation that people are slower than vehicles and the crossing of the line generally takes more frames than in highway traffic videos.

The people counting results are depicted in column *People* of Table 4.3. In the used videos, the intensity of the flow of people is similar in both directions. Therefore, if the detection line could not differentiate between the directions, there would be a lot of false positive detection instances.

4.3.3. Speed of RNN-VDL

The training of the RNN based detector has been done on a GPU. However, because of the computational efficiency of the proposed method, the most of the practical applications of the already trained model would probably use CPU. The size of the detection line has a little effect on the speed of the method since all lines are resized to a specified length before entering the RNN. This length N_{fl} is set in the training process. For the RNN-VDL tests in this section, the length is 100 pixels. The size parameters of the network, such as the number of layers and neurons and its depth in time S, are the essential speed affecting parameters. Fig. 4.7 shows, how the measured speed of the detection algorithm depends on the temporal depth of the RNN. The speed is measured on Intel® Xeon® CPU E5-2650 v2 @ 2.60 GHz. The resolution of the input video is 800×450, and the length of the detection line is 400 pixels.

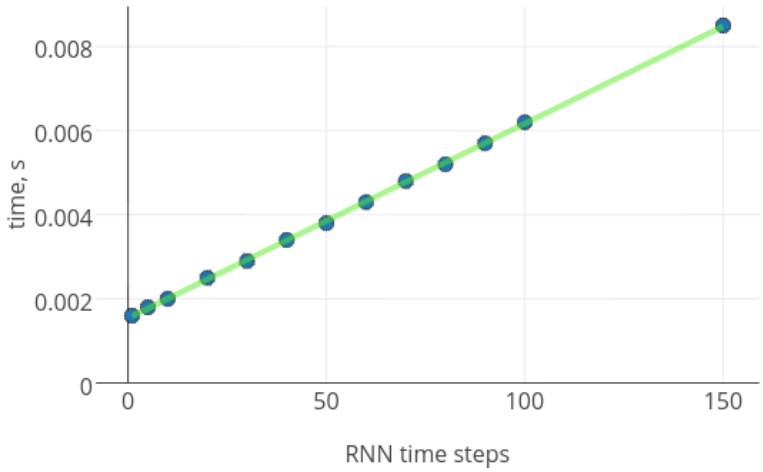


Fig. 4.7. Measurement of the RNN-VDL inference speed on a CPU

The speed test shows a linear dependence between inference speed and S. The choice of depth depends on the specific task. Slow moving objects and high frame rate of the input video may require RNNs with deeper memory. In the current example, the video speed is 15 FPS. So when using an RNN that can remember 50 previous steps, the information from up to three seconds in the past can be used by the model to decide the current output of the network. Based on the training data, the model learns which parts of the past three seconds are important for the detection of different objects. For example, the RNN may learn to assign different attention to

each of the last 50 time steps in the cases of objects that directly cross the detection line and those that stop on the line before leaving.

In order to compare the inference speed of RNN-VDL and that of the state-of-the-art object detector YOLO, both methods were tested on the same single processor. The [110] implementation of YOLO's fast version *tiny-yolo* processed one image in 0.34 seconds, while the proposed RNN-VDL method needed 0.0035 seconds (when $S = 40$).

Such modern object detection methods as R-CNN and YOLO analyze and detect objects in a whole frame instead of a region of interest, which makes them significantly slower than detection line-based methods. R-CNN and YOLO can be used on a broader variety of tasks and return more accurate results than region based methods. However, the RNN-VDL is computationally more efficient and well suited for real-time object detection in video, while being more versatile than other detection region based methods (including IoVDL).

To further prove that the proposed RNN based method can be used with limited computing resources, it is successfully implemented and used on Raspberry Pi 3 Model B computer (Fig. 4.1 b). Pi 3 has Quad Core 1.2 GHz Broadcom BCM2837 64 bit CPU and 1 GB of RAM. With $S = 40$, the RNN-VDL needs 0.025 seconds to process a line from a single frame of a size of 352×240 pixels.

4.4. Summary

The Thesis proposes two object detection methods – the hand-crafted IoVDL algorithm and a machine learning approach RNN-VDL.

The IoVDL was built for the specific task of traffic monitoring. Its implementation on the cheapest Raspberry Pi computer demonstrates the computational efficiency of the approach. The IoVDL is not the only efficient vehicle detector; however, it is more flexible than the conventional region and detection-line based approaches. The tests on different videos show that the same IoVDL implementation successfully works on videos from different roads with different number and direction of lanes (or no lanes at all). Also, it automatically adapts to the changes in lighting and weather. The accuracy of the method is similar to that claimed by the alternative efficient methods.

Another advantage of the interval-based method is its extended version, where several detection lines are combined to monitor broader region of the road. This extension can also acquire parameters of the vehicles – direction, speed, length, width, and height. The size parameters are used to classify vehicles into motorcars and trucks. Tests show that most vehicles are classified correctly. 88 % of trucks were detected and classified as such, while only 6 out of 1186 detected compact cars were incorrectly classified as trucks. An automatic toll payment system might demand even more precise classification with an almost 100 % accuracy, whereas traffic planning and road pavement maintenance do not need such high accuracy. In turn, these applications

require the deployment of many sensors when implemented in urban areas with many streets. Thus, these applications can significantly benefit from the low-cost classification solution of extended IoVDL. In addition, tests indicate that combination of several detection lines increases the accuracy of vehicle counting.

Another proposed method RNN-VDL is a supervised machine learning approach. Its performance is substantially dependent on the labeled training data. The model trained in this Thesis is capable of detecting vehicles in different videos from different sources; although the results are better when the RNN-VDL is tested on videos from the same road and camera that were used for its training. As with many deep learning approaches, the generalization capability of an RNN-VDL model might be improved by extending the training dataset with videos from different sources.

The options for use of RNN-VDL are even more versatile than those of IoVDL. People counting test shows that RNN-VDL can be used to detect different kinds of objects. Thanks to the internal memory of the RNN, the trained model can distinguish the direction in which people cross the detection line. Therefore, the RNN-based method is computationally efficient and more widely applicable than conventional region-based detectors.

5. CONCLUSION

This Thesis researched and developed video processing methods for moving object detection. One of the target applications of such methods is vehicle detection by intelligent transportation systems. The analysis of different road sensors in Subsection 2.1 indicated that in the road setting the practical applicability of different detection methods depends on their ability to adapt to the changing conditions. Furthermore, when deploying many detectors for coverage of a large area, the cost of processing devices becomes essential. Hence this paper paid particular attention to the computationally efficient methods. Accordingly, this Thesis aimed to improve efficient detection of moving objects in a video.

Six tasks were defined to achieve the set aim.

1. To perform a review of literature about object detection in images and videos. This task was accomplished in Section 1. Unsurprisingly, the computationally simpler methods, such as thresholding and background subtraction turned out to be less robust than feature detectors (SIFT, SURF, ORB).

The literature review acknowledged that state-of-the-art object detection in images is currently achieved by deep learning methods, especially the convolutional neural network-based approaches (R-CNN, YOLO). However, it was also noted that recurrent neural networks, which have proven themselves in a time series related tasks, could be better suited for video processing. Unlike the conventional CNN architectures, the recurrent networks use the information from past frames when making decisions about the current one. Despite the object detection capabilities of the deep learning methods, the analysis of literature did not find the existing CNN and RNN-based detection methods practically usable for a real-time video processing, since these methods demand significant computational resources.

2. To identify efficient approaches for detection of moving objects in videos. It was accomplished in Subsection 2.1, which showed that a common trait of efficient methods is a significant reduction of the pixels that are processed in each frame. Such methods consider only pixels of a defined region (virtual detector). The identified drawback of these methods is that objects have to enter the virtual detector in order to be detected, so these methods are applicable when the objects of interest do not deviate from their usual trajectories. In transportation applications, this condition does not always hold true and is sometimes violated even on the roads with clearly marked lanes. If a virtual detector is small, it misses the vehicles that deviate from the expected trajectory. On the other hand, if the detector covers several lanes, there is a problem of separating vehicles that have simultaneously entered the detection region. This deficiency of the existing efficient detection methods is what the novel methods developed in this Thesis were set to overcome. The virtual detector approach was the starting point for accomplishing the third task of the Thesis.

3. To develop efficient object detection methods with improved capabilities compared to the existing efficient methods. The first novel method IoVDL was developed in Subsection 2.2. It processes a line of pixels to detect objects. While these objects are crossing the detection line, the IoVDL creates and maintains corresponding intervals, which are used to make a detection decision. In this manner, the proposed IoVDL method copes with the usual drawbacks of virtual detectors. In the vehicle detection context, the interval approach does not require unchanging trajectories of vehicles. The IoVDL is applicable on the roads where lanes are not specified or their number changes, depending on the time of the day. Several intervals can be created and maintained on the detection line at the same time, so the IoVDL is capable of separately detecting several objects that enter the virtual detector simultaneously.

Subsection 2.3 demonstrated an additional advantage of the novel interval approach by developing the extended IoVDL method. By placing several IoVDL detectors in the frame and combining intervals from different detectors, the extended IoVDL is able to capture the object's movement through the detectors. Additionally, the method characterizes the object by measuring its speed and size parameters and uses these measurements to classify the object. The description of the extended IoVDL showed that it is still computationally efficient, since only the pixels of detection lines are processed. Compared to the original IoVDL, the additional detection lines decrease the overall processing speed in mostly linear proportion to the number of added detectors. For example, ten detection lines are approximately ten times less efficient than a single detector, but they are still significantly more efficient than the methods that process the whole frame. For example, if the frame resolution is 640×480 and there are ten 600 pixel long detection lines, the background subtraction on these lines needs approximately $(640 \times 480)/(600 \times 10) \approx 51$ times less computational resources than the background subtraction of the whole frame. Additional operations of extended IoVDL are the processing of intervals. Each interval of a detection line is described by two integers, so these additional operations have little effect on the overall efficiency of the extended IoVDL.

As was indicated in the literature review, the most flexible and robust object detection methods in computer vision are currently based on artificial neural networks. For this reason, another novel video processing approach for moving object detection was proposed in Subsection 2.4. The RNN-VDL method combines the idea of the efficient detection line and a recurrent neural network. Similarly to IoVDL, the new method processes a single line of pixels and detects objects that cross this line. The pixels of the detection line are forwarded through the LSTM network, which uses the current and past information to form the result.

The output of the RNN-VDL model tries to predict the label of the current frame. The frames can be labeled in different ways, and in Subsection 2.4.2 three modes of labeling are proposed. It was concluded that different proposed modes are better suited for different types of object detection. For example, mode 1 is the best approach when one wants to determine the current number of objects on the detector line at each frame. On the other hand, mode 2 and mode 3

are better suited for the counting of objects. The mode 3 labels the current frame by the number of objects that have left the detection line in some number of previous frames. For the counting task, this way of labeling results in unambiguous labels, balanced dataset, and less work for the labeler than in mode 1.

The use of the powerful machine learning technique indicated that the RNN-VDL method could be used in even more versatile applications than IoVDL. However, this approach requires a significant amount of labeled data for training.

4. To develop data acquisition methods and acquire the data needed for the development and testing of proposed methods. Section 3 is dedicated to methods for the acquisition of labeled data. It was noted, that because of the novelty of the RNN-VDL method, the existing annotated datasets are not directly usable for its training. Unlike the conventional CNN-based object detectors, the training of RNN-VDL demands labeled videos instead of images. Furthermore, each frame of the video should be labeled based on the situation on the single detection line, while the existing datasets of videos label each frame based on the whole scene. Therefore, this Thesis proposes novel labeling approaches.

In Subsection 3.2, a GUI based-approach was developed to speed up the manual labeling of each frame. It was concluded that this approach could be used to label data for different kinds of detection methods, not only the detection line-based ones. However, there is a reasonable assumption that a more specialized labeling method could speed up the labeling of data even more than a versatile approach. Thus, in Subsection 3.2.2 a novel semi-automatic method was developed.

The proposed semi-automatic labeling method performs an initial automatic labeling of the data using background subtraction. Then an implemented GUI allows a human user to quickly detect and correct the mistakes made by the automatic labeler. The practical merit of the method was verified by creating labeled datasets that were used for training and testing the RNN-VDL method. These datasets are available in [111].

Another proposed approach of acquiring labeled data was is use of a 3D game engine to create synthetic data. The discussion in Subsection 3.3 indicated that this method has a great future potential; however, it currently requires significant human skills to create realistic simulations of real-world environments and objects. This approach may be considered when dealing with the detection of visually simple objects in an indoor environment.

5. To implement and conduct experimental research on the proposed methods. All the proposed detection and labeling methods were implemented in code. The experimental tests of the methods are described in Section 4.

The IoVDL method was tested on the vehicle counting task (Subsection 4.1). By using different videos, it was determined that IoVDL performs similarly in different weather and traffic conditions; however, its accuracy depends on the placement of the camera relative to the road. On average, vehicles were counted with an accuracy of approximately 95 %, which is close to

the results claimed by the alternative efficient methods.

In Subsection 4.2, the vehicle counting and classification capabilities of the extended IoVDL were tested on several videos from different locations. The proposed method for combining intervals of different lines turned out to be a more precise vehicle counter than a single IoVDL detector. The count returned by the single detector deviated from the ground truth on average by 4.4 %, while the count returned by the extended IoVDL – by 2.0 %. The simple averaging of the results of each separate line resulted in a 4.0 % error.

The extended IoVDL can characterize the objects by measuring their parameters. The reliability of the determined size parameters was tested on a classification task, where the dimensions of the vehicles were used to determine which detected vehicles were trucks. The method correctly identified 88 % of the trucks.

The RNN-VDL method was also tested on the vehicle detection task. In Subsection 4.3, it was compared with the IoVDL. Experiments showed, that when the test videos were similar to the data used in the training of RNN-VDL, the machine learning approach performed better than IoVDL. In the case of significantly different video, the hand-crafted IoVDL made fewer errors.

Implementation of IoVDL within the framework of this paper is more efficient than the implementation of RNN-VDL. It was demonstrated by implementing and running the IoVDL on a Raspberry Pi Zero computer (single core 1 GHz CPU, 512 MB RAM), while the implementation of RNN-VDL demanded a more powerful Raspberry Pi 3 model B (quad core 1.2 GHz CPU, 1 GB RAM).

An additional advantage of the RNN-VDL was demonstrated by the test on people counting. It showed that after training the network with different data, the same architecture could be used for a different task without any changes in the algorithm. The processing of a single line in the frame could even differentiate between people crossing the line in different directions. This ability makes the RNN-VDL approach more broadly applicable than existing efficient methods.

6. To draw conclusions about the results of this Thesis. The final task of the Thesis is done in the current section. The main conclusion about object detection methods developed in this paper is that their properties and test results demonstrate that the aim of this Thesis is successfully achieved – the developed methods improve the efficient detection of moving objects in a video. In addition, the accomplished tasks prove four statements defined at the beginning of this Thesis.

Statement 1. The developed IoVDL method detects moving objects in a video with a similar accuracy and computational efficiency as alternative virtual detection region-based methods while being less sensitive to the differing number and trajectories of objects that cross the detection region. The description of the IoVDL method in Subsection 2.2 demonstrates the less limited applicability of the proposed method than that of the conventional efficient methods. This conclusion in combination with the test results in Subsection 4.1 proves the first statement.

Statement 2. The combination of several IoVDL detectors (extended IoVDL) tracks and measures the speed and size parameters of objects with less computational resources than conventional tracking methods that process the whole frame. The description of the extended IoVDL method in Subsection 2.3 indicated that its computational demands are approximately equal to those of a single IoVDL detection line multiplied by the number of used detectors. However, practical measurements in Subsection 4.1 showed that doubling the number of detectors does not lead to processing of the frames twice as slow. The reason is that additional processing operations, which are not dependent on the number of detectors (video capturing, frame reading), consume a considerable portion of the computational resources.

The intended use of extended IoVDL is that the number of detectors is much smaller than the number of rows in a frame, so the method processes much fewer pixels than conventional tracking methods which process while frame. Nevertheless, as shown in the tests in Subsection 4.2, the method is able to track and characterize objects in a wide area of the video, which proves the second statement.

Statement 3. The combination of virtual line detector approach with a recurrent neural network results in an adaptive virtual detector RNN-VDL that can be retrained for the detection of different kinds of objects without changes in its architecture and the handcrafted feature engineering. The recurrent network-based method RNN-VDL was successfully trained and tested on vehicle and people counting tasks (Subsection 4.3), which proves the third statement.

Statement 4. The speed of labeling the several hour long video data for the training of recurrent neural network-based object detector RNN-VDL is increased at least ten times by the developed semi-automatic labeling method. The labeled dataset for the training of RNN-VDL was acquired by a single labeler using the semi-automatic labeling approach developed in this Thesis. The aim of this approach is to increase the speed of data labeling compared to the manual labeling of each frame sequentially.

The benefits of proposed labeling approach can be demonstrated in an extreme case of a video without any objects. The conventional computer monitors display 60 frames per second. So in order to sequentially determine if an hour long 30 FPS video contains no moving objects, the labeler would need at least 30 minutes.

In the case of semi-automatic labeling, the human is presented with a spatio-temporal image that contains information about as many frames as there are rows in this image. The resolution of a typical computer monitor is 1920×1080, which means that in a single glance, the labeler determines that 1080 consecutive frames are empty. This potentially allows one to label an hour-long video 18 times faster than using a sequential approach.

This extreme example does not take into account the limitations of human perception, which would slow down both methods. It would slow the sequential approach more because it includes 1080 times more frame changes.

Both approaches become slower with the presence of objects in the video. Still, the proposed GUI based semi-automatic labeling consumes less of human time, since it allows the user to faster jump over video parts with no label change. This analysis demonstrates the truth of the fourth statement.

Additional conclusions after the experimental tests of the developed methods concern their practical application. The combination of the labeling method and the RNN-VDL makes the proposed approach ready for practical use. To apply this technique to a new task, the user has to record new videos, then label them with the help of the proposed semi-automatic approach. Labeled data is used to train the RNN-VDL. The code for the model as well as the code for its training are given in the Appendix.

The IoVDL provides even more efficient moving object detection than the RNN-VDL; however, this method is tested only on the vehicle detection task. Detection of other kinds of objects would require some changes in the algorithms code. If the performance of the IoVDL is not accurate enough in a new setting, then some manual fine-tuning may help. The sensitivity of the IoVDL can be increased or decreased by changing the number of adjacent lines that form a line detector (discussed in Subsection 2.2.3). In addition, a manually set bias can be added to the different threshold values that are found automatically. Based on the camera frame-rate and the speed of objects, it is also possible to vary the parameter that makes IoVDL ignore too brief intervals.

As was shown in this closing section, all the tasks set at the beginning of this Thesis were fulfilled. The contributions of this Thesis might be of interest to researchers as well as practitioners in the field of computer vision. The developed data labeling approaches contribute to the field of deep learning, where the lack of labeled data is one of the greatest current challenges. The developed methods IoVDL, extended IoVDL, and RNN-VDL introduce a novel way of making computationally efficient vision-based moving object detectors. These approaches were mostly tested and used on the roads; however, the flexibility of the methods, especially that of the RNN-VDL, opens up the possibilities for their use in other practical applications.

APPENDICES

Manual data labeling GUI code

```
import numpy as np
import cv2

# PARAMETERS
VIDEO_NAME = "/home/roberts/data/traffic/Begri_2013_004.mpg"
LINE_R = 300
LINE_WIDTH = 1
LINE_C_BEGIN = 151
LINE_C_END = 550

# Capture the video
cap = cv2.VideoCapture(VIDEO_NAME)
frames_count = int( cap.get(cv2.CAP_PROP_FRAME_COUNT) )

# Initialize
line_save = np.empty([frames_count, LINE_WIDTH, LINE_C_END-LINE_C_BEGIN],
    dtype="uint8")
labels = np.empty([frames_count, 3])
cv2.startWindowThread()
cv2.namedWindow("input video")

# Process the video
for i in range(frames_count):
    ret, frame = cap.read()

    line = frame[LINE_R : LINE_R + LINE_WIDTH,
                 LINE_C_BEGIN : LINE_C_END].copy()
    cv2.line(frame, (LINE_C_BEGIN, LINE_R), (LINE_C_END, LINE_R),
             (255, 0, 0), thickness=1, lineType=8, shift=0)

    cv2.imshow("input video", frame)

    key = cv2.waitKey(0)
    key_char = chr(key & 255)
    if key_char == '0':
        label = [0, 0, 1]
    elif key_char == '1':
        label = [0, 1, 0]
    elif key_char == '2':
        label = [1, 0, 0]

    # ... add more keys/labels if needed
```

```
        elif  key_char == 'q':
            line_save = line_save[0:i  ,:,:]
            labels  = labels[0: i ,:]
            break

        else:
            print("WARNING: wrong key!")

        line  = cv2.cvtColor(line,  cv2.COLOR_BGR2GRAY)
        line_save[i  ,:,:]   = line
        labels[i ,:]   = label


# Save the results
line_save = np.squeeze(line_save)

np.save(VIDEO_NAME[ 0 : VIDEO_NAME.find('.') ] + "_line_"
        + str(LINE_R) + ".npy" , line_save)
np.save(VIDEO_NAME[ 0 : VIDEO_NAME.find('.') ] + "_line_"
        + str(LINE_R) + "_labels.npy", labels)

cap.release()
cv2.destroyAllWindows()
```

Semi-automatic data labeling GUI code

```
'''
A semi-automatic labeling of data for object detection in video
using virtual detection line.

    Example use:
        python3 sa-labelling_2.py -vidname myvideo.avi -labelMode onTheLine

    First, define the ending points of the detectin line in the frame using two
    right-clicks. First point must be on the left of the second.
    Press 'e' to start automatic labeling.
    Press 'q' to stop the visualisation of automatic labeling if enabled (-vizInput).

    Two labeling modes ar implemented:
        1) The label of current frame is equal to the number of objects on the detection
            line.
            * Right-click: if closest label line is upwards, it is moved to the
              cursor postition.
            * Right-click: if closest line is downwards, it is deleted.
            * Left-click: adds a new label line at the cursor position. Its label
              is the last number pressed on the keyboard.
        2) The label of current frame is equal to the number of objects that have left the
        detection line at the current frame.
            * Righ-click: automatically finds the closest edge of the object
              upwards from cursor position. Puts a new line there with label {1} or
              increases existing label by 1.
            * Left-click: if closest label line is downwards, moves this line one row up.
            * Left-click: if closest label line is upwards, decrease its label by 1.

    Use mouse weel to scroll trough the spatio-temporal image.
    Press 'Space' to switch between the labeling modes.
    Press 'Esc' when finished to save the labels and data to hdd.
    (Resulting files are saved in the directory of input video)
'''

import sys
import cv2
import numpy as np
import argparse
from PyQt4 import QtGui
from PyQt4 import QtCore
```

```
# Input arguments
parser = argparse.ArgumentParser()
parser.add_argument('-vidName', '--vidName',
                    required = False, default = 'test_videos/night.avi')
parser.add_argument('-nClasses', '--nClasses',
                    required = False, default = 5, type = int)
parser.add_argument('-vizInput', '--vizInput',
                    required = False, default = False)
parser.add_argument('-labelMode', '--labelMode',
                    required = False, default = 'leftTheLine')
parser.add_argument('-autoMode', '--autoMode',
                    required = False, default = 'subtraction')
parser.add_argument('-subThr', '--subThr',
                    required = False, default = 10, type = int)
args = parser.parse_args()

VIDEO_NAME = args.vidName
N_CLASSES = args.nClasses
VIZ_INPUT = args.vizInput
LABEL_MODE = args.labelMode # 'leftTheLine' or 'onTheLine'
AUTO_METHOD = args.autoMode # 'subtraction' or 'mog2'
SUBTRACTION_THR = args.subThr

print("Preparing the input..")

cap = cv2.VideoCapture(VIDEO_NAME)
frames_count = int( cap.get(cv2.CAP_PROP_FRAME_COUNT) )
frame_height = int( cap.get(cv2.CAP_PROP_FRAME_HEIGHT) )
fps = int( cap.get(cv2.CAP_PROP_FPS) )
if (frames_count == 0):
    print("ERROR: could not determime the frame count!")
    print("Counting frames manualy..")
    while(True):
        ret, frame = cap.read()
        if not ret:
            break
        frames_count += 1

    print("  frame count", frames_count)


# Set the position and length of detection line with two mouse clicks
lineSet = False
click1 = False
click2 = False
def set_the_line(event, x, y, flags, param):
    global LINE_R, LINE_C_BEGIN, LINE_C_END, lineSet, click1, click2
    if event == cv2.EVENT_LBUTTONDOWN:
```

```
        if (click2 == False and click1 == True):
            LINE_C_END = x
            cv2.line(frame, (LINE_C_BEGIN, LINE_R), (LINE_C_END, LINE_R),
                (255, 0, 0), thickness=1, lineType=8, shift=0)
            click2 = True
            lineSet = True

        if (click1 == False):
            LINE_R = y
            LINE_C_BEGIN = x
            cv2.circle(frame, (LINE_C_BEGIN, LINE_R), radius=1,
                color=(255, 0, 0), thickness=3, lineType=8, shift=0)
            click1 = True

        cv2.imshow("input", frame)


# Create dislpay window for visualisation purposes
cv2.namedWindow("input")
cv2.setMouseCallback("input", set_the_line)


# Progress bar function
def progress_bar(name, val, end_val, bar_length=20):
    percent = float(val) / end_val
    hashes = '#' * int(round(percent * bar_length))
    spaces = ' ' * (bar_length - len(hashes))
    sys.stdout.write("\r" + name + ": [{0}] {1}%".format(hashes + spaces,
        int(round(percent * 100))))
    sys.stdout.flush()


# Initialize
current_label = 0
matricesDefined = False
backgroundHalfReady = False
background_ready = False
fgbg_mog = cv2.createBackgroundSubtractorMOG2()


# Process frames (create thesholded spatio-temporal image)
i = 0
while i < frames_count-1:
    ret, frame = cap.read()
    if ret == True:

        if (lineSet):

            if (matricesDefined == False):
```

```
        # Define the matrices
        line_spatio_temporal = np.zeros((frames_count, LINE_C_END -
            LINE_C_BEGIN))
        spatio_temporal = np.zeros((frames_count,
            LINE_C_END-LINE_C_BEGIN, 3))
        spatio_temporal_bw = np.zeros((frames_count,
            LINE_C_END-LINE_C_BEGIN))
        if VIZ_INPUT:
            vizbuf_diff = np.zeros((frame_height,
                LINE_C_END-LINE_C_BEGIN))
            vizbuf_thr = np.zeros((frame_height,
                LINE_C_END-LINE_C_BEGIN))
        matricesDefined = True

    line_color = frame[LINE_R : LINE_R + 1,
                LINE_C_BEGIN : LINE_C_END].copy()
    line = cv2.cvtColor(line_color, cv2.COLOR_BGR2GRAY)
    line_spatio_temporal[i,:] = line.copy()


    cv2.line(frame, (LINE_C_BEGIN, LINE_R), (LINE_C_END, LINE_R),
            (255, 0, 0), thickness=1, lineType=8, shift=0)


if (backgroundHalfReady and background_ready == False):
    background_ready = True
    background = line.copy()

if (background_ready == False):
    cv2.imshow("input", frame)
    key = cv2.waitKey(0)
    key_char = chr(key & 255)
    if key_char == 'q':
        print("The line was not ok!")
        quit()
    elif key_char == 'e':
        i -= 1
        backgroundHalfReady = True


# Subtract the background
if (background_ready):

    if AUTO_METHOD == "subtraction":
        diff_image = cv2.absdiff(line,background)
        ret, diff_image_thr = cv2.threshold(diff_image,
                SUBTRACTION_THR, 255, cv2.THRESH_BINARY)
```

```
            # Update the background
            if current_label == 0:
                background = cv2.addWeighted(background, 0.95, line, 0.05, gamma=0)

        elif AUTO_METHOD == "mog2":
            diff_image = fgbg_mog.apply(line)
            diff_image_thr = diff_image


        # Create the spatio-temporal image
        spatio_temporal[i,:] = line_color/255
        spatio_temporal_bw[i,:] = diff_image_thr

        # Count the objects:
        if (sum(sum(diff_image_thr))/255 > 0.2 * line.shape[1]):
            current_label = 1
        else:
            current_label = 0

        # Visualize
        if VIZ_INPUT:
            vizbuf_diff[0:frame_height-1,:] = vizbuf_diff[1:,:]
            vizbuf_diff[frame_height-1,:] = diff_image / 255
            vizbuf_thr[0:frame_height-1,:] = vizbuf_thr[1:,:]
            vizbuf_thr[frame_height-1,:] = diff_image_thr

            input_vizualisation = np.zeros((frame.shape[0],
                        frame.shape[1] + vizbuf_diff.shape[1]*2, 3))
            input_vizualisation[0:frame.shape[0], 0:frame.shape[1], :] = frame/255

            buffer_1 = frame.shape[1] + vizbuf_diff.shape[1]
            input_vizualisation[:, frame.shape[1]:buffer_1, 1] = vizbuf_diff
            input_vizualisation[:, buffer_1:buffer_1 + vizbuf_diff.shape[1], 1] =
                vizbuf_thr

            cv2.imshow("input", input_vizualisation)
            keyr = cv2.waitKey(1)
            key_charr = chr(keyr & 255)
            if key_charr == 'q':
                break
        else:
            cv2.destroyWindow("input")

        progress_bar("Background subtraction:", i, frames_count, bar_length=20)
    i += 1
cv2.destroyWindow("input")
```

```
print("Processing spatio-temporal image..")

# Morphological processing
spatio_temporal_bw_raw = spatio_temporal_bw.copy()
kernel = np.ones((3,3), np.uint8)
kernel2 = np.ones((5,5), np.uint8)
spatio_temporal_bw = spatio_temporal_bw.astype(np.uint8)
spatio_temporal_bw = cv2.morphologyEx(spatio_temporal_bw, cv2.MORPH_CLOSE,
    kernel)
spatio_temporal_bw = cv2.morphologyEx(spatio_temporal_bw, cv2.MORPH_OPEN,
    kernel2)


# Fill the holes in white blobs
im_floodfill = spatio_temporal_bw.copy().astype(np.uint8)
h, w = im_floodfill.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)
cv2.floodFill(im_floodfill, mask, (0,0), 255)
im_floodfill_inv = cv2.bitwise_not(im_floodfill)
spatio_temporal_bw = spatio_temporal_bw | im_floodfill_inv


def count_objects_on_the_line(lineThr, objectWidth):
    '''
    Counts how many objects are on the detection line.
    Objects are connected white pixels that are wider than
    arg[1] * widthOfTheLine
    '''
    line_width = lineThr.shape[0]
    j = 0
    curr_px = 0
    objects = 0
    px_counter = 0

    while j < line_width:
        prev_px = curr_px
        curr_px = lineThr[j]
        if curr_px > prev_px:
            px_counter = 0
        elif curr_px < prev_px:
            if px_counter > objectWidth * line_width:
                objects += 1
            px_counter = 0
        else:
            px_counter += 1
        j += 1

    return objects
```

```
def label_currently_on_the_line(imSpaTempThr):
    print()
    print("Using the 'currently on the line' labeling mode")

    labels = np.zeros((imSpaTempThr.shape[0], 1))

    i = 0
    while i < imSpaTempThr.shape[0]:
        lineThr = imSpaTempThr[i,:]
        labels[i] = count_objects_on_the_line(lineThr, 0.1)

        progress_bar("Automatic labelling:", i, spatio_temporal_bw.shape[0],
            bar_length=20)
        i += 1

    return labels


def label_currently_left(imSpaTempThr):
    print()
    print("Using the 'currently left the detection line' labeling mode")

    labels = np.zeros((imSpaTempThr.shape[0], 1))
    i = 0
    line_width = imSpaTempThr.shape[1]
    objectsCurrent = 0
    while i < imSpaTempThr.shape[0]:
        lineThr = imSpaTempThr[i,:]
        objectsPrevious = objectsCurrent
        objectsCurrent = count_objects_on_the_line(lineThr, 0.1)
        if objectsPrevious - objectsCurrent > 0:
            labels[i] = objectsPrevious - objectsCurrent
        else:
            labels[i] = 0

        progress_bar("Automatic labelling:", i, spatio_temporal_bw.shape[0],
            bar_length=20)
        i += 1

    return labels


# Choose the method
if LABEL_MODE == 'onTheLine':
    labels = label_currently_on_the_line(spatio_temporal_bw)
if LABEL_MODE == 'leftTheLine':
    labels = label_currently_left(spatio_temporal_bw)
```

```
def draw_gui_image():
    '''
    Draws labels to the spatio-temporal GUI image
    '''
    gui_image = np.zeros((spatio_temporal.shape[0], spatio_temporal.shape[1] + 60, 3))
    gui_image[0:spatio_temporal.shape[0], 0:spatio_temporal.shape[1]] = spatio_temporal
    current_label = 0
    change_delay = 0
    label_coord_x = spatio_temporal.shape[1]

    # Put lines
    for i in range(labels.shape[0]):
        previous_label = current_label
        current_label = labels[i,0]
        if current_label != previous_label:

            alpha = float(0.3)
            frame_overlay = gui_image.copy()
            cv2.line(frame_overlay, (0, i), (gui_image.shape[1], i),

                     (1, 0, 0), thickness=1, lineType=8, shift=0)
            cv2.addWeighted(frame_overlay, alpha, gui_image, 1 - alpha, 0, gui_image)

    # Put text
    for i in range(labels.shape[0]):
        previous_label = current_label
        current_label = labels[i,0]

        label_coord_x == spatio_temporal.shape[1]
        if current_label != previous_label:

            cv2.putText(gui_image, str(int(current_label)), (label_coord_x, i+7),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (1,1,1))

            if label_coord_x == spatio_temporal.shape[1]+50:
                label_coord_x = spatio_temporal.shape[1]
            else:
                label_coord_x += 10

    # Make the image Qt ready
    gui_image = cv2.cvtColor(np.asarray(gui_image*255, dtype=np.uint8),
                cv2.COLOR_BGR2RGB).copy()
    return gui_image
```

```
def find_closest_object(mousePosX, mousePosY):
    ''' Finds the closest edge of an object upwards the current cursor position. '''

    regionHeight = 20
    regionWidth = 10
    if mousePosY < regionHeight:
        regionHeight = mousePosY

    region = spatio_temporal[mousePosY-regionHeight:mousePosY,
            mousePosX-regionWidth / 2: mousePosX+regionWidth / 2, :]
    region = region.astype(np.float32)
    region = cv2.cvtColor(region, cv2.COLOR_BGR2GRAY)

    originalLine = region[regionHeight-1, :]

    jump = regionHeight-1
    maxDif = 0
    firstDif = 10000

    imax = regionHeight-2

    for i in reversed(range(regionHeight-2)):
        dif = abs(originalLine - region[i,:])
        sumDif = np.sum(dif)
        if i == regionHeight - 3:
            firstDif = sumDif


        if sumDif > maxDif:
            maxDif = sumDif
            imax = i

        if firstDif == 0:
            firstDif = sumDif

        else:
            if sumDif > 5 * firstDif:
                jump = i
                break

        if i == 0:
            jump = imax

        previousSumDif = sumDif

    jumpCoord = mousePosY - (regionHeight - jump)

    return(jumpCoord+1)
```

```
def change_label(event, mousePosX, mousePosY, newGuiLabel, LABEL_MODE):

    if newGuiLabel == 100:
        if event.button() == QtCore.Qt.LeftButton:

            jumpCoord = find_closest_object(mousePosX, mousePosY)

            labels[jumpCoord, 0] += 1
            labels[jumpCoord + 1, 0] == 0
        if event.button() == QtCore.Qt.RightButton:
            for i in range(50):
                i_forward = mousePosY + i
                i_backward = mousePosY - i
                if i_backward < 0:
                    break
                if i_forward >= labels.shape[0]:
                    break
                else:
                    if labels[i_backward, 0] > 0:
                        labels[i_backward, 0] -= 1
                        break
                    if labels[i_forward, 0] > 0:
                        labels[i_forward, 0] -= 1
                        labels[i_forward - 1] += 1
                        break



    else:
        # Add a new line with a label previously set by a keyboard key
        if event.button() == QtCore.Qt.LeftButton:
            oldLabel = labels[mousePosY, 0]
            for i in range(mousePosY, labels.shape[0]):
                if LABEL_MODE == 'onTheLine':
                    if labels[i, 0] == oldLabel:
                        labels[i, 0] = newGuiLabel
                    else:
                        break

                if LABEL_MODE == 'leftTheLine':
                    if i == mousePosY:
                        labels[i, 0] = newGuiLabel
                    else:
                        if labels[i, 0] == oldLabel:
                            labels[i, 0] = 0
                        else:
                            break
```

```
            # Find closest existing line
            if event.button() == QtCore.Qt.RightButton:
                i = 0
                currentLabel = labels[mousePosY, 0]
                while mousePosY + i < labels.shape[0] and mousePosY - i > 0:
                    forward = mousePosY + i
                    backward = mousePosY - i
                    if (labels[forward, 0] != currentLabel):
                        oldLabel = labels[forward,0]
                        f = forward
                        while (f < labels.shape[0]):
                            if labels[f, 0] == oldLabel:
                                labels[f, 0] = currentLabel
                                f += 1
                            else:
                                break
                        break

                    # If closest line is up
                    if (labels[backward, 0] != currentLabel):
                        if LABEL_MODE == 'onTheLine':
                            oldLabel = labels[backward,0]
                            b = backward
                            while (b != mousePosY):
                                b += 1
                                labels[b, 0] = oldLabel
                            break

                        if LABEL_MODE == 'leftTheLine':
                            oldLabel = labels[backward,0]
                            b = backward
                            while b != mousePosY:
                                labels[b, 0] = 0
                                b += 1
                            labels[mousePosY] = oldLabel
                            break

                    i += 1

        gui_image = draw_gui_image()
        return gui_image


def save_labels():
    ''' Converts labels to one-hot vectors and saves them to hdd'''
    print("Saving results .. ")
```

```
        labels_one_hot = np.zeros((labels.shape[0], N_CLASSES))
        for i in range(labels.shape[0]):
            labels_one_hot[i, int(labels[i,0])] = 1

        name = VIDEO_NAME[0:VIDEO_NAME.find('.')] + "_line_" + str(LINE_R) + "_"
            + str(LINE_C_BEGIN) + "_" + str(LINE_C_END)

        np.asarray(line_spatio_temporal, dtype=np.int8)
        np.save(name + ".npy", line_spatio_temporal[:-1,:])
        np.save(name + "_labels.npy", labels_one_hot[:-1,:])


gui_image = draw_gui_image()

windowHeight = 800
bytesPerComponent = 3
imageHeight, imageWidth, bytesPerComponent = gui_image.shape
bytesPerLine = bytesPerComponent * imageWidth


# THE GUI
print()

class Window(QtGui.QMainWindow):

    def __init__(self):

        self.imSizeMult = 2

        super(Window, self).__init__()
        self.setGeometry(50,50, self.imSizeMult * imageWidth + 20, windowHeight)
        self.setWindowTitle("Manual labeling")
        self.newGuiLabel = 0
        self.home()

    def home(self):

        # Show the image
        self.imageLabel = QtGui.QLabel(self)
        QI = QtGui.QImage(gui_image, imageWidth, imageHeight,
                          bytesPerLine, QtGui.QImage.Format_RGB888)

        imageSize = QI.size()
        imageSize.setWidth(imageSize.width() * self.imSizeMult)
        imageSize.setHeight(imageSize.height() * self.imSizeMult)
        QI = QI.scaled(imageSize, QtCore.Qt.KeepAspectRatioByExpanding)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(QI))
        self.imageLabel.setGeometry(QtCore.QRect(0, 0, 2*imageWidth, 2*imageHeight))
```

```
        self.imageLabel.mousePressEvent = self.mouse_press

        # Scroll
        scrollArea = QtGui.QScrollArea(self)
        scrollArea.setWidget(self.imageLabel)
        scrollArea.setGeometry(0, 0, 2*imageWidth + 20, windowHeight)
        scrollArea.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)

        self.show()

    def mouse_press(self, event):
        mousePos = event.pos()
        mousePosX = int(mousePos.x() / self.imSizeMult)
        mousePosY = int(mousePos.y() / self.imSizeMult)

        gui_image = change_label(event, mousePosX, mousePosY, self.newGuiLabel,
            LABEL_MODE)

        QI = QtGui.QImage(gui_image, imageWidth, imageHeight,
                          bytesPerLine, QtGui.QImage.Format_RGB888)
        imageSize = QI.size()
        imageSize.setWidth(imageSize.width() * self.imSizeMult)
        imageSize.setHeight(imageSize.height() * self.imSizeMult)
        QI = QI.scaled(imageSize, QtCore.Qt.KeepAspectRatioByExpanding)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(QI))

    def keyPressEvent(self, event):
        self.newGuiLabel = event.text()
        if event.key() == QtCore.Qt.Key_Space:
            self.newGuiLabel = 100
        if event.key() == QtCore.Qt.Key_Tab:
            self.newGuiLabel = 0
        if event.key() == QtCore.Qt.Key_Escape:
            save_labels()
            print("All done!")

            QtGui.QApplication.quit()


def run_gui():
    app = QtGui.QApplication(sys.argv)
    GUI = Window()
    sys.exit(app.exec_())

run_gui()
```

Preparing the data generated by the Unreal Engine 4

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
import random

# PARAMETERS
VISUALIZE = False
LINE_WIDTH = 600
CLASS_NR = 5
DATA_NAME = '18'
DATA_PATH = '/home/roberts/data/balls/'


framePath = DATA_PATH + 'frames_' + DATA_NAME + '/'
labelPath = DATA_PATH + 'labels_' + DATA_NAME + '/'

frameNr = len(os.listdir(framePath))
imOut = np.zeros((frameNr-1, LINE_WIDTH))
labelsOut = np.zeros((frameNr-1, CLASS_NR))


for i in range(3, frameNr+2):

    # Process frames
    if i < 10:
        imName = 'frames ' + '0'
    else:
        imName = 'frames '

    image = cv2.imread(framePath + imName + str(i) + '.jpeg')
    imageG = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    line = imageG[8,:]
    imOut[i-3,:] = line

    # Convert label mode 1 to label mode 2
    with open(labelPath + str(i) + '.txt', 'r') as f:
        lines = f.readlines()
        label = int(lines[0])
        if label > CLASS_NR-1:
            label = CLASS_NR-1

        labelsOut[i-3, label] = 1
```

```
# Save training matrices
np.save(DATA_PATH + 'train/' + DATA_NAME + '.npy', imOut)
np.save(DATA_PATH + 'train/' + DATA_NAME + '_labels.npy', labelsOut)


# Add Gaussian noise
gaussX = round(random.random() * 10)
gaussY = round(random.random() * 10)
if gaussX % 2 == 0:
    gaussX += 1
if gaussY % 2 == 0:
    gaussY += 1

frames = cv2.GaussianBlur(imOut,(gaussX, gaussY),0)

mean = 0
sigma = random.random()
noise = np.zeros((frames.shape))
cv2.randn(noise, mean, sigma)

frames = frames + noise

if VISUALIZE:
    print("sigma", sigma, "gaussX", gaussX, "gaussY", gaussY)

    plt.imshow(frames[:100,:], cmap='gray')
    plt.show()

np.save(DATA_PATH + 'train/' + 'noise_' + DATA_NAME + '.npy', frames)
np.save(DATA_PATH + 'train/' + 'noise_' + DATA_NAME + '_labels.npy', labelsOut)
```

Definition of the RNN model

```
with tf.device(GPU_USED):

    # tf graph input
    x = tf.placeholder("float", [None, N_STEPS, N_INPUT])
    istate = tf.placeholder("float", [None, 2*N_HIDDEN])
    y = tf.placeholder("float", [None, N_CLASSES])

    # Define weights
    weights = {
        'hidden': tf.Variable(tf.random_normal([N_INPUT, N_HIDDEN])),
        'hidden_2': tf.Variable(tf.random_normal([N_HIDDEN, N_HIDDEN_2])),
        'out': tf.Variable(tf.random_normal([N_HIDDEN_2, N_CLASSES]))
    }
    biases = {
        'hidden': tf.Variable(tf.random_normal([N_HIDDEN])),
        'hidden_2': tf.Variable(tf.random_normal([N_HIDDEN_2])),
        'out': tf.Variable(tf.random_normal([N_CLASSES]))
    }


    # The Net
    def RNN(_X, _istate, _weights, _biases):

        # Input shape: (batch_size, N_STEPS, N_INPUT)
        _X = tf.transpose(_X, [1, 0, 2])
        _X = tf.reshape(_X, [-1, N_INPUT])

        _X = tf.nn.relu(tf.matmul(_X, _weights['hidden']) + _biases['hidden'])

        # Split data because rnn cell needs a list of inputs
        _X = tf.split(0, N_STEPS, _X)

        # Define a lstm cell with tensorflow
        lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(N_HIDDEN, forget_bias=1.0,
                            state_is_tuple=False)

        # Get lstm cell output
        outputs, states = tf.nn.rnn(lstm_cell, _X, initial_state=_istate)

        # Linear activation
        return tf.matmul(outputs[-1], _weights['out']) + _biases['out']
```

```
# Evaluate the net
pred = RNN(x, istate, weights, biases)
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y))
optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE).minimize(cost)

# Initialize the variables
init = tf.initialize_all_variables()

saver = tf.train.Saver()
```

Training code for RNN-VDL

```
import os
import pickle
import numpy as np
import pylab as pl
import tensorflow as tf
from random import randint
from shutil import copyfile
import matplotlib.pyplot as plt
import data_preparation as dataprep


# PARAMETERS
MODEL_NAME = "people_counter"
DATA_PATH = "/home/roberts/data/people/train_small/"
CONTINUE_LEARNING = False
PRETRAINED_PATH = "models/detection_1/"
PRETRAINED_MODEL = "train0.962_val0.829.ckpt"
SAVE_LOG = True
PLOT_WIDTH = 50
GPU = 1
BALANCE_SEQUENCES = False
ALLOW_SOFT_PLACEMENT = True

# Network parameters
N_INPUT = 100
N_STEPS = 40
N_HIDDEN = 100
N_HIDDEN_2 = 100
N_CLASSES = 10

# Training parameters
LEARNING_RATE = 2e-4
EPOCHS = 5000
VISUALIZATION = True
BATCH_SIZE = 500
DISPLAY_STEP = 1
VALIDATION = True
TRAIN_VAL_RATIO = 0.7


print("Preparing input data.. \n")

linesTrain, labelsTrain, linesVal, labelsVal = dataprep.load_input(DATA_PATH,
```

```
        TRAIN_VAL_RATIO, N_INPUT)
labelsTrain = dataprep.convert_label_format(labelsTrain, N_STEPS, N_CLASSES)
labelsVal = dataprep.convert_label_format(labelsVal, N_STEPS, N_CLASSES)


# Save the parameters and hyperparameters
os.makedirs("models/" + MODEL_NAME, exist_ok=True)

N_CLASSES = labelsTrain[0].shape[1]
f = open("models/" + MODEL_NAME + "/parameters.pickle", 'wb')
pickle.dump([N_INPUT, N_STEPS, N_HIDDEN, N_HIDDEN_2, N_CLASSES,
    LEARNING_RATE], f)
f.close()

# Choose the device CPU/GPU
os.environ["CUDA_VISIBLE_DEVICES"] = str(GPU)
GPU_USED = "/gpu:" + str(GPU)

dataprep.print_data_stats("Training data", labelsTrain, N_CLASSES)
dataprep.print_data_stats("Validation data", labelsVal, N_CLASSES)


print("Defining the network..\n")
if CONTINUE_LEARNING:
    exec(open(PRETRAINED_PATH + "linedetect-net.py").read())
    copyfile(PRETRAINED_PATH + "linedetect-net.py", "models/" +
            MODEL_NAME + "/linedetect-net.py")
else:
    exec(open("linedetect-net.py").read())
    copyfile("linedetect-net.py", "models/" + MODEL_NAME + "/linedetect-net.py")


print("Preparing sequences and batches..\n")

sqPosition, nrSq, sqLabels = dataprep.sequence_positions(linesTrain,
                                                         labelsTrain, N_STEPS)
sqPositionVal, nrSqVali, sqLabelsVal = dataprep.sequence_positions(linesVal,
                                                         labelsVal, N_STEPS)


if BALANCE_SEQUENCES:
    balancer = dataprep.get_sequence_balancer(sqLabels)
    sqPosition = sqPosition + balancer
    balancerVal = dataprep.get_sequence_balancer(sqLabelsVal)
    sqPositionVal = sqPositionVal + balancerVal

    print("  Class relations after sequence balancing")
    classRelation = np.zeros(labelsTrain[0][0].shape)
    for position in sqPosition:
```

```
            classNow = np.argmax(labelsTrain[position[0]][[position[1]]])
            classRelation[classNow] += 1
        print("  ", classRelation, "\n")

# Number of batches to cover (almost) all training data
batches_total = np.floor(len(sqPosition) / BATCH_SIZE)
batches_total_val = np.floor(len(sqPositionVal) / BATCH_SIZE)

batch_x = np.zeros((BATCH_SIZE, N_STEPS, N_INPUT))
batch_y = np.zeros((BATCH_SIZE, N_CLASSES))

print("    ", len(sqPosition), "train", len(sqPositionVal), "val sequences")
print("    Total:", EPOCHS, "epochs,", BATCH_SIZE, "sequences of",
                                            N_STEPS, "time steps in a batch")
print("    Training:", batches_total, "batches")
print("    Validation:", batches_total_val, "batches \n\n")

# Create a single batch consisting of _BATCH_SIZE randomly selected sequences
def next_batch(_BATCH_SIZE, _batch, _perm, _lines, _labels, _sqPosition):
    for sequence_idx in range(_BATCH_SIZE):
        currentSqPos = _sqPosition[_perm[sequence_idx + _BATCH_SIZE * _batch]]
        batch_x[sequence_idx,:,:] = _lines[ currentSqPos[0] ][
            currentSqPos[1]:currentSqPos[1]+N_STEPS ]
        batch_y[sequence_idx,:] = _labels[ currentSqPos[0] ][currentSqPos[1] + N_STEPS -
            1, :]
    return batch_x, batch_y


print("Training..\n")

epoch_viz = []
acc_train_viz = []
acc_val_viz = []
cost_viz = []
plotCounterBig = 0
perm = np.arange(len(sqPosition))
perm_val = np.arange(len(sqPositionVal))
acc_val = 0

# Launch the session using the graph defined in linedetect-net.py
with
    tf.Session(config=tf.ConfigProto(allow_soft_placement=ALLOW_SOFT_PLACEMENT))
    as sess:
    if CONTINUE_LEARNING:
        saver.restore(sess, PRETRAINED_PATH + PRETRAINED_MODEL)
    else:
        sess.run(init)

    # Keep training until max iterations reached
```

```
for epoch in range(EPOCHS):
    avg_cost = 0
    acc_train_vec = []
    acc_val_vec = []

    np.random.shuffle(perm)

    for batch in range(int(batches_total)):
        batch_xs, batch_ys = next_batch(BATCH_SIZE, batch, perm,
                                        linesTrain, labelsTrain, sqPosition)
        sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys,
                                       istate: np.zeros((BATCH_SIZE,
                                           2*N_HIDDEN))})

        if epoch % DISPLAY_STEP == 0:
            # Get accuracy on the single batch
            acc_train_vec.append(sess.run(accuracy,
                                 feed_dict={x: batch_xs, y: batch_ys,
                                 istate: np.zeros((BATCH_SIZE, 2*N_HIDDEN))}))

            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys,
                                 istate: np.zeros((BATCH_SIZE, 2*N_HIDDEN))})

    if epoch % DISPLAY_STEP == 0:
        if (VALIDATION):
            for batch in range(int(batches_total_val)):
                batch_xs, batch_ys = next_batch(BATCH_SIZE, batch, perm_val,
                                                linesVal, labelsVal, sqPositionVal)
                acc_val_vec.append(sess.run(accuracy,
                                   feed_dict={x: batch_xs, y: batch_ys,
                                   istate: np.zeros((BATCH_SIZE,
                                       2*N_HIDDEN))}))
            acc_val = np.mean(acc_val_vec)

        # Get accuracy on the whole training set
        acc_train = np.mean(acc_train_vec)
        saver.save(sess, "models/" + MODEL_NAME + "/train" +
                   str(round(acc_train,3)) + "_val" +
                   str(round(acc_val,3)) + ".ckpt")

        if VISUALIZATION:
            try:
                epoch_viz.append(epoch)
                acc_train_viz.append(acc_train)
                cost_viz.append(avg_cost)
                if (VALIDATION):
                    acc_val_viz.append(acc_val)
```

```
            if ( SAVE_LOG == True and epoch != 0 and epoch %
                PLOT_WIDTH == 0):

                    plt.figure()
                    pl.plot(epoch_viz, acc_train_viz, 'b')
                    if (VALIDATION):
                        pl.plot(epoch_viz, acc_val_viz, 'g')

                    plt.grid(True)
                    plt.xlabel('epochs')
                    plt.ylabel('accuracy')

                    partLogName = ( "models/" + MODEL_NAME + '/accuracy_'
                                    + str(plotCounterBig) + '.ps' )
                    plt.savefig(partLogName, bbox_inches='tight')

                    plt.figure()
                    plt.plot(epoch_viz, cost_viz, 'r')
                    plt.grid(True)
                    plt.xlabel('epochs')
                    plt.ylabel('cost')
                    partLogName = ( "models/" + MODEL_NAME + '/cost_'
                                    + str(plotCounterBig) + '.ps')
                    plt.savefig(partLogName, bbox_inches='tight')

                    plotCounterBig += 1

                    epoch_viz = []
                    acc_train_viz = []
                    acc_val_viz = []
                    cost_viz = []

            except KeyboardInterrupt:
                break

        print ("Epoch:", '%03d' % (epoch+1), "of", str(EPOCHS),
               "cost=", "{:.4f}".format(avg_cost),
               " Train accuracy=", "{:.3f}".format(acc_train),
               " Val accuracy=", "{:.3f}".format(acc_val))

print ("Optimization Finished!")
```

Data preparation for RNN-VDL training

```
import os
import glob
import numpy as np
from scipy.misc import imresize

# Resize and typecast the line
def prepare_line(_line, n_input):
    _line = _line.astype(float) / 255.
    _line = _line.astype(float)

    if ( n_input != _line.shape[1] ):
        line_res = np.empty((_line.shape[0], n_input))
        for k in range(_line.shape[0]):
            line_res[k,:] = imresize(_line[k:k+1,:], (1, n_input),
                                     interp='nearest', mode='F')
        _line = line_res
    return _line


def load_input(datasets_path, trainValRatio, n_input):
    '''
    Loads several labeled videos and their labels (both are spatio-temporal images).
    Input data is normalised and the width of input lines is reduced,
    so that lines from different videos (lines with different sizes)
    can be fed into the same network with n_input number of inputs.
    Names of spatio-temporal images must end with a number (i.e. video1_300.npy),
    while the according labels should end with phrase 'labels' (i.e.
        video1_300_labels.npy).
    '''

    # Variables for the names (paths) of input data
    data = []
    dataTrain = []
    dataVal = []

    linesTrain = []
    labelsTrain = []
    linesVal = []
    labelsVal = []

    owd = os.getcwd()
    os.chdir(datasets_path)
```

```
    for file in glob.glob("*[0-9].npy"):
        data.append(datasets_path + file[0:file.find('.npy')])
    os.chdir(owd)
    print("    loading", len(data), "  files ..")

    # Divide the data in to train and validation parts
    permData = np.arange(len(data))
    np.random.shuffle(permData)
    nrTrainData = round(trainValRatio * len(data))

    for i in range(len(data)):
        if i <= nrTrainData:
            dataTrain.append(data[permData[i]])
        else:
            dataVal.append(data[permData[i]])

    # Load and resize the training data
    for i in range(len(dataTrain)):
        line_i = np.load(dataTrain[i] + ".npy")
        labels_i = np.load(dataTrain[i] + "_labels.npy")
        line_i = prepare_line(line_i, n_input)
        linesTrain.append(line_i)
        labelsTrain.append(labels_i)

    # Load and resize the validation data
    for i in range(len(dataVal)):
        line_i = np.load(dataVal[i] + ".npy")
        labels_i = np.load(dataVal[i] + "_labels.npy")
        line_i = prepare_line(line_i, n_input)
        linesVal.append(line_i)
        labelsVal.append(labels_i)

    print("    ", len(dataTrain), 'training /', len(dataVal), "validation")

    return linesTrain, labelsTrain, linesVal, labelsVal


def convert_label_format(labelsOld, n_steps, n_classes):
    ''' Converts labels from mode 2 to mode 3 '''

    labelsNew = []
    countedSeqLength = int(n_steps / 2)
    for strip in labelsOld:
        newStrip = np.zeros((strip.shape[0], n_classes))
        for r in range(strip.shape[0]):
            begining = r - countedSeqLength
            if begining < 0:
                begining = 0
```

```
                newLabel = 0
                for l in range(begining, r):
                    maxId = np.argmax(strip[l,:])
                    newLabel += maxId
                newStrip[r, newLabel] = 1

            labelsNew.append(newStrip)

        return labelsNew


def print_data_stats(dataName, labels, n_classes):
    class_nr = []
    class_nr = np.zeros((n_classes))
    class_percent = []
    for i in range(n_classes):
        for j in range(len(labels)):
            class_nr[i] = class_nr[i] + sum(labels[j][:, i])

    one_percent = (sum(class_nr)) / 100
    print(class_nr, "\n")

    print(" ", dataName, ":")
    for i in range(n_classes):
        class_percent.append( round(class_nr[i] / one_percent, 2) )
        print("  Class", i, ":", class_percent[i], "%")


def sequence_positions(lines, labels, n_steps):
    ''' Prepares the variable sqPositions that points to the positions
    of every sequences in all the input spatio-temporal data. '''

    sqLabels = []
    nrSq = []
    sqPosition = []

    if labels != []:
        classNr = labels[0][0].shape
        for i in range(classNr[0]):
            sqLabels.append([])

        for i in range(len(lines)):
            nrSq.append(lines[i].shape[0] - n_steps + 1)
            for j in range(nrSq[i]):
                sqPosition.append((i, j))
                for c in range(classNr[0]):
                    if labels[i][j][c] != 0:
                        sqLabels[c].append((i,j))
```

```
    return sqPosition, nrSq, sqLabels


def get_sequence_balancer(sqLabels):
    maxClNr = 0
    for i in range(len(sqLabels)):
        clNr = len(sqLabels[i])
        if clNr > maxClNr:
            maxClNr = clNr

    sqBalancer = []

    for i in range(len(sqLabels)):
        clNr = len(sqLabels[i])
        while ( clNr != 0 and (maxClNr > clNr) ):
            sqBalancer = sqBalancer + sqLabels[i]
            clNr = clNr + len(sqLabels[i])

    return sqBalancer
```

BIBLIOGRAPHY

[1] DiCarlo, J. J., Zoccolan, D., Rust, N. C. How does the brain solve visual object recognition? In: *Neuron*, 2012, 73(3), 415–434.

[2] Saxena, A., Driemeyer, J., Ng, A. Y. Robotic grasping of novel objects using vision. In: *The International Journal of Robotics Research*, 2008, 27(2), 157–173.

[3] Brosnan, T., Sun, D.-W. Improving quality inspection of food products by computer vision – a review. In: *Journal of food engineering*, 2004, 61(1), 3–16.

[4] Maglogiannis, I., Doukas, C. N. Overview of advanced computer vision systems for skin lesions characterization. In: *IEEE transactions on information technology in biomedicine*, 2009, 13(5), 721–733.

[5] Kang, J., Hsu, C.-H., Wu, Q., Liu, S., Coster, A. D., Posner, B. A., Altschuler, S. J., Wu, L. F. Improving drug discovery with high-content phenotypic screens by systematic selection of reporter cell lines. In: *Nature biotechnology*, 2016, 34(1), 70–77.

[6] Parkhi, O. M., Vedaldi, A., Zisserman, A. Deep Face Recognition. In: *BMVC 1(3)*. 2015, p. 6.

[7] Junior, J. C. S. J., Musse, S. R., Jung, C. R. Crowd analysis using computer vision techniques. In: *IEEE Signal Processing Magazine*, 2010, 27(5), 66–77.

[8] Coifman, B., Beymer, D., McLauchlan, P., Malik, J. A real-time computer vision system for vehicle tracking and traffic surveillance. In: *Transportation Research Part C: Emerging Technologies*, 1998, 6(4), 271–288.

[9] Li, H., Shen, C. Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs. In: *arXiv preprint arXiv:1601.05610*, 2016, p. 17.

[10] Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., et al. An empirical evaluation of deep learning on highway driving. In: *arXiv preprint arXiv:1504.01716*, 2015, p. 7.

[11] He, K., Zhang, X., Ren, S., Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. 2015, 1026–1034.

[12] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., Zisserman, A. The pascal visual object classes (voc) challenge. In: *International journal of computer vision*, 2010, 88(2), 303–338.

[13] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. In: *International Journal of Computer Vision (IJCV)*, 2015, 115(3), 211–252.

[14] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, 779–788.

[15] Krizhevsky, A., Sutskever, I., Hinton, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems.* 2012, 1097–1105.

[16] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L. Microsoft coco: Common objects in context. In: *European conference on computer vision.* Springer 2014, 740–755.

[17] Freivalds, K., **Kadiķis**, R., Greitāns, M. System and method for video-based vehicle detection. Patent (EP 2700054 B1), Apr. 2015.

[18] **Kadiķis**, R., Freivalds, K. Efficient video processing method for traffic monitoring combining motion detection and background subtraction. In: *Proceedings of the Fourth International Conference on Signal and Image Processing 2012 (ICSIP 2012).* Springer, 2013, 131–141.

[19] **Kadiķis**, R., Freivalds, K. Vehicle classification in video using virtual detection lines. In: *Sixth International Conference on Machine Vision (ICMV 13).* International Society for Optics and Photonics 2013, 90670Y–90670Y.

[20] **Kadiķis**, R. Registration method for multispectral skin images. In: *Radioelektronika (RADIOELEKTRONIKA), 2015 25th International Conference.* IEEE 2015, 232–235.

[21] Nauris, D., **Kadiķis**, R., Nesenbergs, K. Vehicle type and license plate localisation and segmentation using FCN and LSTM. In: *New Challenges of Economic and Business Development 2017, Proceedings of Reports.* University of Latvia 2017, 132–140.

[22] Tamošiūnas, M., Jakovels, D., Ļihačovs, A., Kilikevičius, A., Baltušnikas, J., **Kadiķis**, R., Šatkauskas, S. Application of fluorescence spectroscopy and multispectral imaging for non-invasive estimation of GFP transfection efficiency. In: *8th International Conference on Advanced Optical Materials and Devices.* International Society for Optics and Photonics 2014, 94210M–94210M.

[23] Jakovels, D., Saknite, I., Bliznuks, D., Spigulis, J., **Kadiķis**, R. Benign-A typical nevi discrimination using diffuse reflectance and fluorescence multispectral imaging system. In: *International Conference on BioPhotonics (BioPhotonics).* IEEE 2015, 1–4.

[24] Tamošiūnas, M., **Kadiķis**, R., Saknīte, I., Baltušnikas, J., Kilikevičius, A., Lihachev, A., Petrovska, R., Jakovels, D., Šatkauskas, S. Noninvasive optical diagnostics of enhanced green fluorescent protein expression in skeletal muscle for comparison of electroporation and sonoporation efficiencies. In: *Journal of biomedical optics*, 2016, 21(4), 045003–045003.

[25] Mimbela, L. E. Y., Klein, L. A. Summary of vehicle detection and surveillance technologies used in intelligent transportation systems. 2007, p. 218.

[26] Sun, C., Ritchie, S. G. Individual vehicle speed estimation using single loop inductive waveforms. In: *Journal of Transportation Engineering*, 1999, 125(6), 531–538.

[27] Bellucci, P., Cipriani, E. Data accuracy on automatic traffic counting: the SMART project results. In: *European transport research review*, 2010, 2(4), 175–187.

[28] Yu, X., Sulijoadikusumo, G., Li, H., Prevedouros, P. Reliability of Automatic Traffic Monitoring with Non-Intrusive Sensors. In: *ICCTP 2011: Towards Sustainable Transportation Systems* 2011, 4157–4169.

[29] Karpathy, A. *What I learned from competing against a ConvNet on ImageNet* [online]. [viewed 2 August 2017]. Available from: http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

[30] Lee, G.-G., Kim, B.-s., Kim, W.-Y. Automatic estimation of pedestrian flow. In: *ICDSC 2007. First ACM/IEEE International Conference on Distributed Smart Cameras.* IEEE 2007, 291–296.

[31] Otsu, N. A threshold selection method from gray-level histograms. In: *IEEE transactions on systems, man, and cybernetics*, 1979, 9(1), 62–66.

[32] Rodríguez, T., García, N. An adaptive, real-time, traffic monitoring system. In: *Machine Vision and Applications*, 2010, 21(4), 555–576.

[33] Ji, X., Wei, Z., Feng, Y. Effective vehicle detection technique for traffic surveillance systems. In: *Journal of Visual Communication and Image Representation*, 2006, 17(3), 647–658.

[34] Gupte, S., Masoud, O., Martin, R. F., Papanikolopoulos, N. P. Detection and classification of vehicles. In: *IEEE Transactions on intelligent transportation systems*, 2002, 3(1), 37–47.

[35] Cheung, S.-C. S., Kamath, C. Robust background subtraction with foreground validation for urban traffic video. In: *EURASIP Journal on Advances in Signal Processing*, 2005, 2005(14), 2330–2340.

[36] Canny, J. A computational approach to edge detection. In: *IEEE Transactions on pattern analysis and machine intelligence*, 1986, (6), 679–698.

[37] Stauffer, C., Grimson, W. E. L. Learning patterns of activity using real-time tracking. In: *IEEE Transactions on pattern analysis and machine intelligence*, 2000, 22(8), 747–757.

[38] Szeliski, R. Computer vision: algorithms and applications. Springer Science & Business Media, 2010, p. 957.

[39] Venot, A., Lebruchec, J., Roucayrol, J. A new class of similarity measures for robust image registration. In: *Computer vision, graphics, and image processing*, 1984, 28(2), 176–184.

[40] Maglogiannis, I. Automated segmentation and registration of dermatological images. In: *Journal of Mathematical Modelling and Algorithms*, 2003, 2(3), 277–294.

[41] Cideciyan, A. V. Registration of ocular fundus images: an algorithm using cross-correlation of triple invariant image descriptors. In: *IEEE Engineering in Medicine and Biology Magazine*, 1995, 14(1), 52–58.

[42] Harris, C., Stephens, M. A combined corner and edge detector. In: *Alvey vision conference.* 15 (50), Manchester, UK 1988, 147–152.

[43] Mikolajczyk, K., Schmid, C. An affine invariant interest point detector. In: *Computer Vision—ECCV 2002*, 2002, 128–142.

[44] Bay, H., Tuytelaars, T., Van Gool, L. Surf: Speeded up robust features. In: *Computer vision–ECCV 2006*, 2006, 404–417.

[45] Lowe, D. G. Distinctive image features from scale-invariant keypoints. In: *International journal of computer vision*, 2004, 60(2), 91–110.

[46] Matas, J., Chum, O., Urban, M., Pajdla, T. Robust wide-baseline stereo from maximally stable extremal regions. In: *Image and vision computing*, 2004, 22(10), 761–767.

[47] Rosten, E., Drummond, T. Fusing points and lines for high performance tracking. In: *ICCV 2005. 10th IEEE International Conference on Computer Vision.* IEEE 2005, 1508–1515.

[48] Rosten, E., Drummond, T. Machine learning for high-speed corner detection. In: *Computer Vision–ECCV 2006*, 2006, 430–443.

[49] Lowe, D. G. Distinctive image features from scale-invariant keypoints. In: *International journal of computer vision*, 2004, 60(2), 91–110.

[50] Ke, Y., Sukthankar, R. PCA-SIFT: A more distinctive representation for local image descriptors. In: *CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* 2 IEEE 2004, II506–II513.

[51] Ojala, T., Pietikäinen, M., Harwood, D. A comparative study of texture measures with classification based on featured distributions. In: *Pattern recognition*, 1996, 29(1), 51–59.

[52] Rublee, E., Rabaud, V., Konolige, K., Bradski, G. ORB: An efficient alternative to SIFT or SURF. In: *ICCV 2011. IEEE International Conference on Computer Vision.* IEEE 2011, 2564–2571.

[53] Rosin, P. L. Measuring corner properties. In: *Computer Vision and Image Understanding*, 1999, 73(2), 291–307.

[54] Calonder, M., Lepetit, V., Strecha, C., Fua, P. Brief: Binary robust independent elementary features. In: *Computer Vision–ECCV 2010*, 2010, 778–792.

[55] Rad, R., Jamzad, M. Real time classification and tracking of multiple vehicles in highways. In: *Pattern Recognition Letters*, 2005, 26(10), 1597–1607.

[56] Karnowski, J. *AlexNet + SVM* [online]. [viewed 5 December 2017]. Available from: https://jeremykarnowski.files.wordpress.com/2015/07/alexnet2.png

[57] Zeiler, M. D., Fergus, R. Visualizing and understanding convolutional networks. In: *European conference on computer vision.* Springer 2014, 818–833.

[58] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D. Backpropagation applied to handwritten zip code recognition. In: *Neural computation*, 1989, 1(4), 541–551.

[59] Grangier, D., Bottou, L., Collobert, R. Deep convolutional networks for scene parsing. In: *ICML 2009 Deep Learning Workshop.* 3 2009, p. 2.

[60] Farabet, C., Couprie, C., Najman, L., LeCun, Y. Scene parsing with multiscale feature learning, purity trees, and optimal covers. In: *arXiv preprint arXiv:1202.2160*, 2012, p. 9.

[61] Shelhamer, E., Long, J., Darrell, T. Fully convolutional networks for semantic segmentation. In: *IEEE transactions on pattern analysis and machine intelligence*, 2017, 39(4), 640–651.

[62] Girshick, R., Donahue, J., Darrell, T., Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014, 580–587.

[63] Uijlings, J. R. R., Sande, K. E. A. van de, Gevers, T., Smeulders, A. W. M. Selective Search for Object Recognition. In: *International Journal of Computer Vision*, 2013, 104(2), 154–171.

[64] *R-CNN: Regions with Convolutional Neural Network Features* [online]. [viewed 22 December 2017]. Available from: https://github.com/rbgirshick/rcnn

[65] Girshick, R. Fast r-cnn. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2015, 1440–1448.

[66] Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., Zisserman, A. The pascal visual object classes challenge: A retrospective. In: *International journal of computer vision*, 2015, 111(1), 98–136.

[67] Ren, S., He, K., Girshick, R., Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In: *Advances in neural information processing systems*. 2015, 91–99.

[68] Redmon, J., Angelova, A. Real-time grasp detection using convolutional neural networks. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE 2015, 1316–1322.

[69] Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S. CNN features off-the-shelf: an astounding baseline for recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, 806–813.

[70] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 1–9.

[71] Redmon, J., Farhadi, A. YOLO9000: better, faster, stronger. In: *arXiv preprint arXiv: 1612.08242*, 2016, p. 9.

[72] Lipton, Z. C., Berkowitz, J., Elkan, C. A critical review of recurrent neural networks for sequence learning. In: *arXiv preprint arXiv:1506.00019*, 2015, p. 38.

[73] Werbos, P. J. Backpropagation through time: what it does and how to do it. In: *Proceedings of the IEEE*, 1990, 78(10), 1550–1560.

[74] Williams, R. J., Zipser, D. A learning algorithm for continually running fully recurrent neural networks. In: *Neural computation*, 1989, 1(2), 270–280.

[75] Hochreiter, S., Schmidhuber, J. Long short-term memory. In: *Neural computation*, 1997, 9(8), 1735–1780.

[76] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., Schmidhuber, J. LSTM: A search space odyssey. In: *IEEE transactions on neural networks and learning systems*, 2017, p. 12.

[77] Kastrinaki, V, Zervakis, M., Kalaitzakis, K. A survey of video processing techniques for traffic applications. In: *Image and vision computing*, 2003, 21(4), 359–381.

[78] Kim, J.-W., Choi, K.-S., Choi, B.-D., Ko, S.-J. Real-time vision-based people counting system for the security door. In: *International Technical Conference on Circuits/Systems Computers and Communications*. 2002, 1416–1419.

[79] Li, L., Han, S., Asama, H., Duan, F. An automatic parts detection system based on computer vision. In: *35th Chinese Control Conference (CCC)*. IEEE 2016, 9493–9498.

[80] Nie, Z., Hung, M.-H., Huang, J. A Novel Algorithm of Rebar Counting on Conveyor Belt Based on Machine Vision. In: *Journal of Information Hiding and Multimedia Signal Processing*, 2016, 7(2), 425–437.

[81] Lei, M., Lefloch, D., Gouton, P., Madani, K. A video-based real-time vehicle counting system using adaptive background method. In: *SITIS'08. IEEE International Conference on Signal Image Technology and Internet Based Systems.* IEEE 2008, 523–528.

[82] Anan, L. Video vehicle detection algorithm based on virtual-line group. In: *APCCAS 2006. IEEE Asia Pacific Conference on Circuits and Systems.* IEEE 2006, 1148–1151.

[83] Michalopoulos, P. G. Vehicle detection video through image processing: the autoscope system. In: *IEEE Transactions on vehicular technology*, 1991, 40(1), 21–29.

[84] Zhang, G., Avery, R., Wang, Y. Video-based vehicle detection and classification system for real-time traffic data collection using uncalibrated video cameras. In: *Transportation Research Record: Journal of the Transportation Research Board*, 2007, (1993), 138–147.

[85] Yue, Y. A traffic-flow parameters evaluation approach based on urban road video. In: *Int. J. Intell. Eng. Syst*, 2009, 2(1), 33–39.

[86] Mithun, N. C., Rashid, N. U., Rahman, S. M. Detection and classification of vehicles from video using multiple time-spatial images. In: *IEEE Transactions on Intelligent Transportation Systems*, 2012, 13(3), 1215–1225.

[87] Ha, D., Lee, J.-M., Kim, Y.-D. Neural-edge-based vehicle detection and traffic parameter extraction. In: *Image and vision computing*, 2004, 22(11), 899–907.

[88] Lai, A. H., Fung, G. S., Yung, N. H. Vehicle type classification from visual-based dimension estimation. In: *Intelligent Transportation Systems, 2001. Proceedings.* IEEE 2001, 201–206.

[89] Bresenham, J. E. Algorithm for computer control of a digital plotter. In: *IBM Systems journal*, 1965, 4(1), 25–30.

[90] Beymer, D., McLauchlan, P., Coifman, B., Malik, J. A real-time computer vision system for measuring traffic parameters. In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* IEEE 1997, 495–501.

[91] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. In: *International Journal of Computer Vision*, 2017, 123(1), 32–73.

[92] Paolacci, G., Chandler, J., Ipeirotis, P. G. Running experiments on amazon mechanical turk. In: *Judgment and Decision Making*, 2010, 411–419.

[93] Zivkovic, Z. Improved adaptive Gaussian mixture model for background subtraction. In: *ICPR 2004. Proceedings of the 17th International Conference on Pattern Recognition.* 2 IEEE 2004, 28–31.

[94] Zivkovic, Z., Van Der Heijden, F. Efficient adaptive density estimation per image pixel for the task of background subtraction. In: *Pattern recognition letters*, 2006, 27(7), 773–780.

[95] Lerer, A., Gross, S., Fergus, R. Learning Physical Intuition of Block Towers by Example. In: *arXiv preprint arXiv:1603.01312*, 2016, p. 11.

[96] Li, W., Azimi, S., Leonardis, A., Fritz, M. To Fall Or Not To Fall: A Visual Approach to Physical Stability Prediction. In: *arXiv preprint arXiv:1604.00066*, 2016, p. 20.

[97] Gaidon, A., Wang, Q., Cabon, Y., Vig, E. Virtual worlds as proxy for multi-object tracking analysis. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, 4340–4349.

[98] Sadeghi, F., Levine, S. CAD2RL: Real single-image flight without a single real image. In: *arXiv preprint arXiv:1611.04201*, 2016, p. 12.

[99] Zhang, J., Wang, F.-Y., Wang, K., Lin, W.-H., Xu, X., Chen, C. Data-driven intelligent transportation systems: A survey. In: *IEEE Transactions on Intelligent Transportation Systems*, 2011, 12(4), 1624–1639.

[100] *MODMYPI* [online]. [viewed 21 September 2017]. Available from: https://www.modmypi.com/

[101] Fazli, S., Mohammadi, S., Rahmani, M. Neural network based vehicle classification for intelligent traffic control. In: *International Journal of Software Engineering & Applications*, 2012, 3(3), p. 17.

[102] Sullivan, G. D., Baker, K. D., Worrall, A. D., Attwood, C., Remagnino, P. Model-based vehicle detection and classification using orthographic approximations. In: *Image and vision computing*, 1997, 15(8), 649–654.

[103] Koller, D. Moving object recognition and classification based on recursive shape parameter estimation. In: *Proc. 12th Israel Conf. Artificial Intelligence, Computer Vision.* 2728 1993, p. 10.

[104] Leotta, M. J., Mundy, J. L. Vehicle surveillance with a generic, adaptive, 3d vehicle model. In: *IEEE transactions on pattern analysis and machine intelligence*, 2011, 33(7), 1457–1469.

[105] Kingma, D., Ba, J. Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980*, 2014, p. 15.

[106] *Caltrans Live Traffic Cameras* [online]. [viewed 19 April 2017]. Available from: http://www.dot.ca.gov/video/

[107] *Meryland Traffic Cameras* [online]. [viewed 19 April 2017]. Available from: http://www.traffic.md.gov/trafficcameras/

[108] Yosinski, J., Clune, J., Bengio, Y., Lipson, H. How transferable are features in deep neural networks? In: *Advances in neural information processing systems*. 2014, 3320–3328.

[109] *EarthCam Seaside Park Cam* [online]. [viewed 20 June 2017]. Available from: http://www.earthcam.com/usa/newjersey/seasidepark/?cam=seasidepark

[110] *Darkflow* [online]. [viewed 6 July 2017]. Available from: https://github.com/thtrieu/darkflow

[111] **Kadiķis**, R. *Labeled dataset for training and testing virtual detection-line based object detectors* [online]. Riga: EDI, 2017 [viewed 28 November 2017]. Available from: http://vault.edi.lv/index.php/s/yxGPtI2Jw2yWhLR